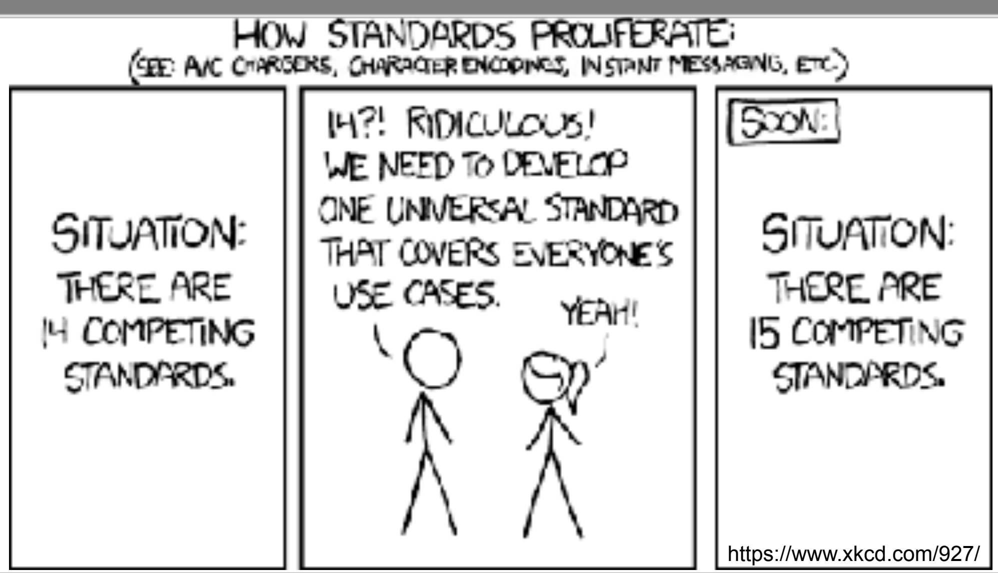
Managing Cluster Fragmentation using ConcurrencyLimits

HTCondor Workshop 2018

<u>Max Fischer</u>, Andreas Petzold, Manfred Alef

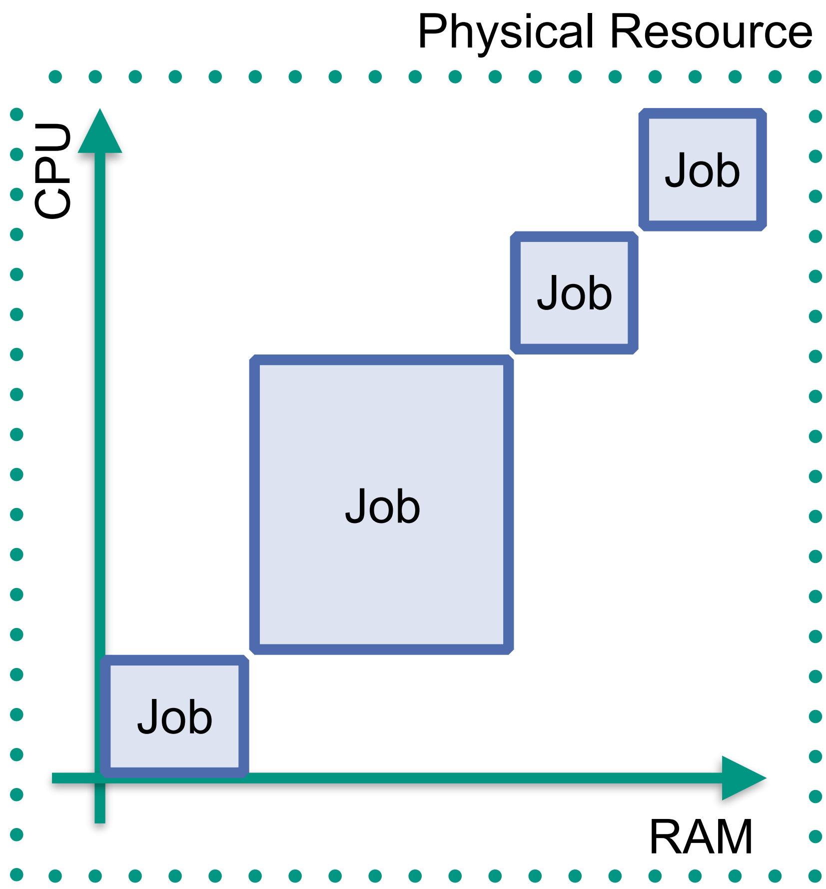
Steinbuch Centre for Computing / Institute for Experimental Particle Physics



https://www.xkcd.com/927/

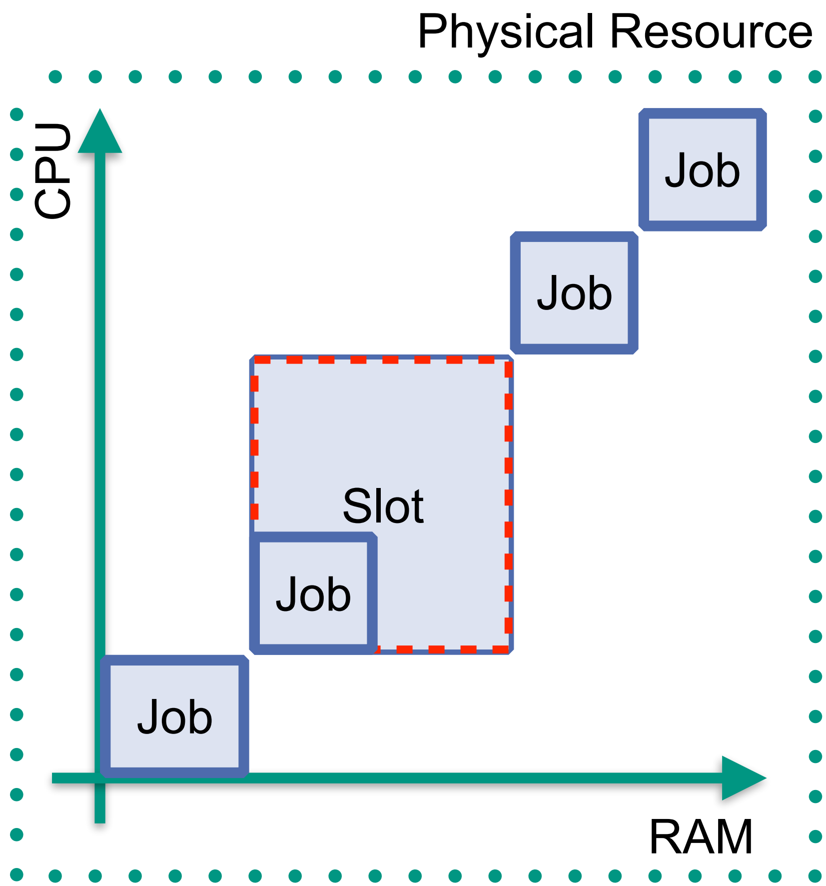
Fragmentation in a Nutshell

- Workers are larger than jobs
  - Each job gets resource share
  - Sum of jobs matches workers



Fragmentation in a Nutshell

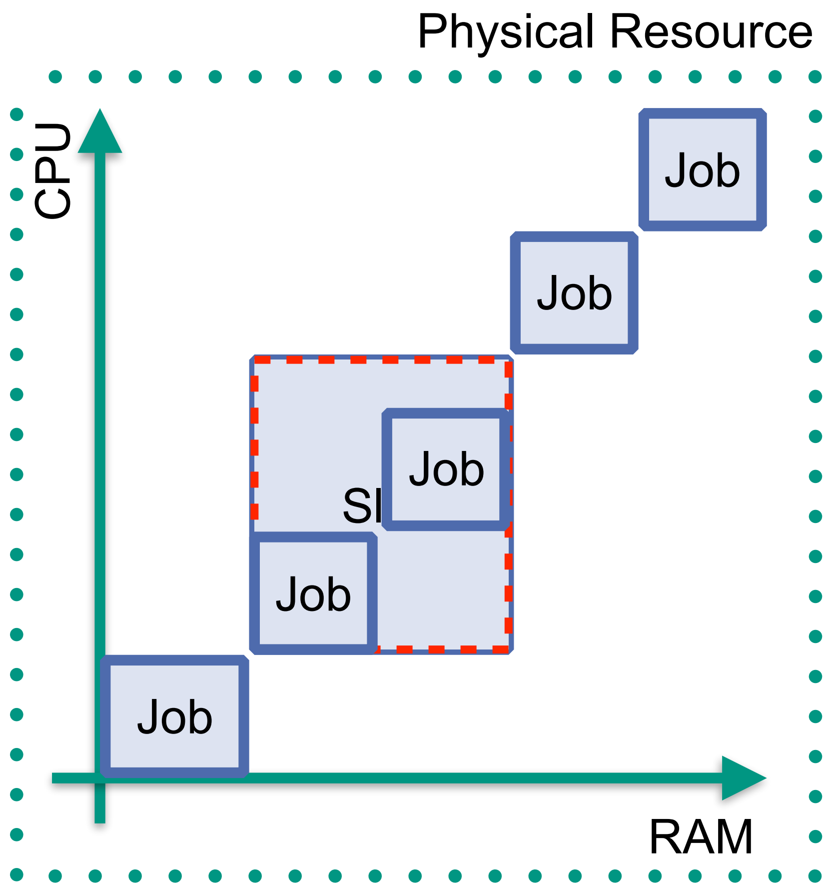
- Workers are larger than jobs
  - Each job gets resource share
  - Sum of jobs matches workers
- Small jobs fragment workers
  - Block resource partitions partly
  - Remainder unfit for large jobs



Fragmentation in a Nutshell

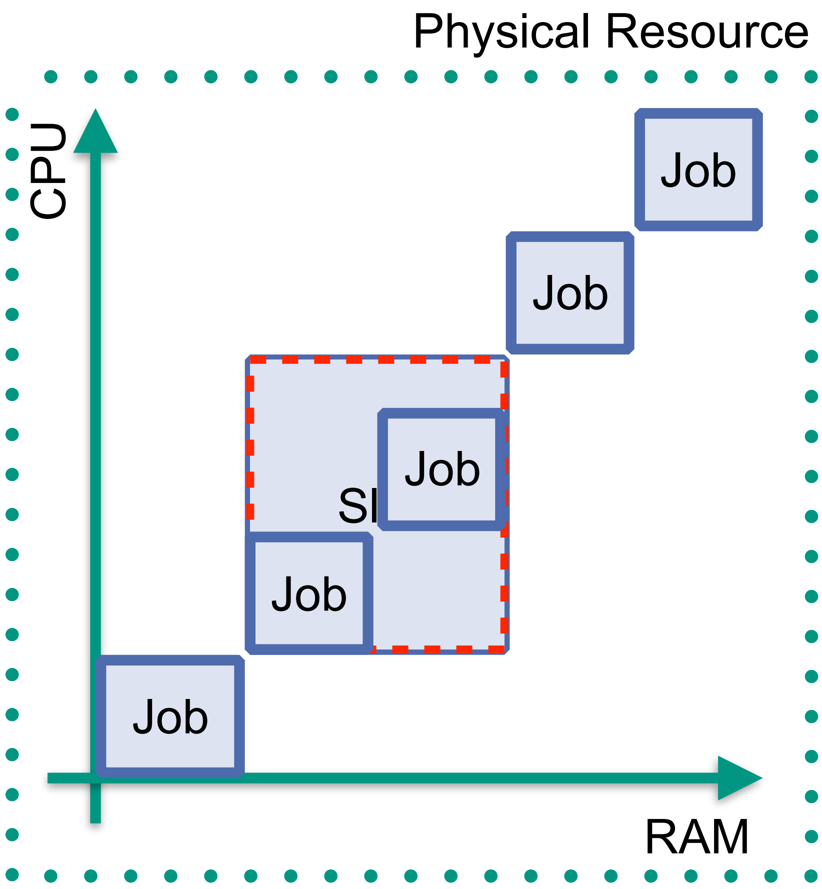
- Workers are larger than jobs
  - Each job gets resource share
  - Sum of jobs matches workers
- Small jobs fragment workers
  - Block resource partitions partly
  - Remainder unfit for large jobs
- Critical when resources are scarce
  - Efficiency requires jobs to start
  - Only small jobs suitable…

GridKa/HEP:
- 1 or 8 CPU
- 2-4GB RAM



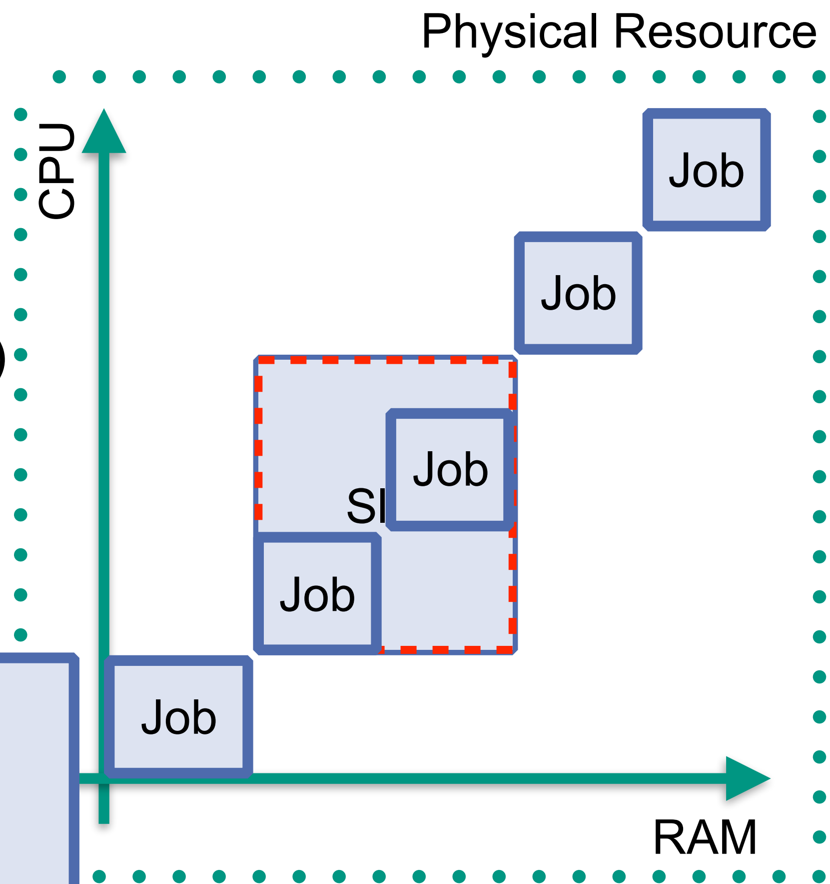
The Fragmentation Challenge

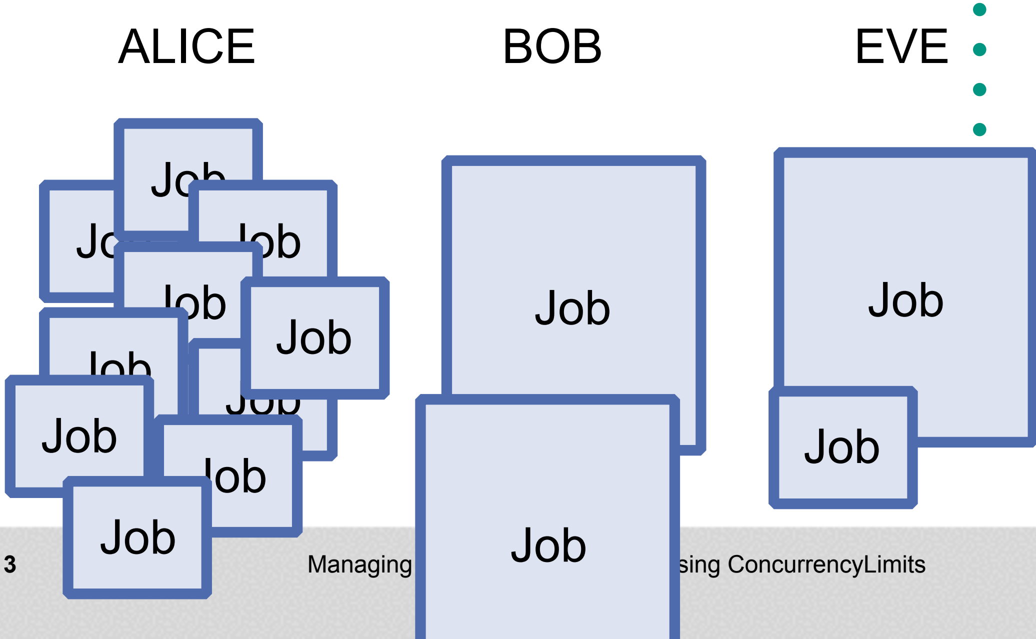
- Objective 1: Cluster Efficiency
  - Minimise unused resources
  - Run best jobs to fill resources



The Fragmentation Challenge

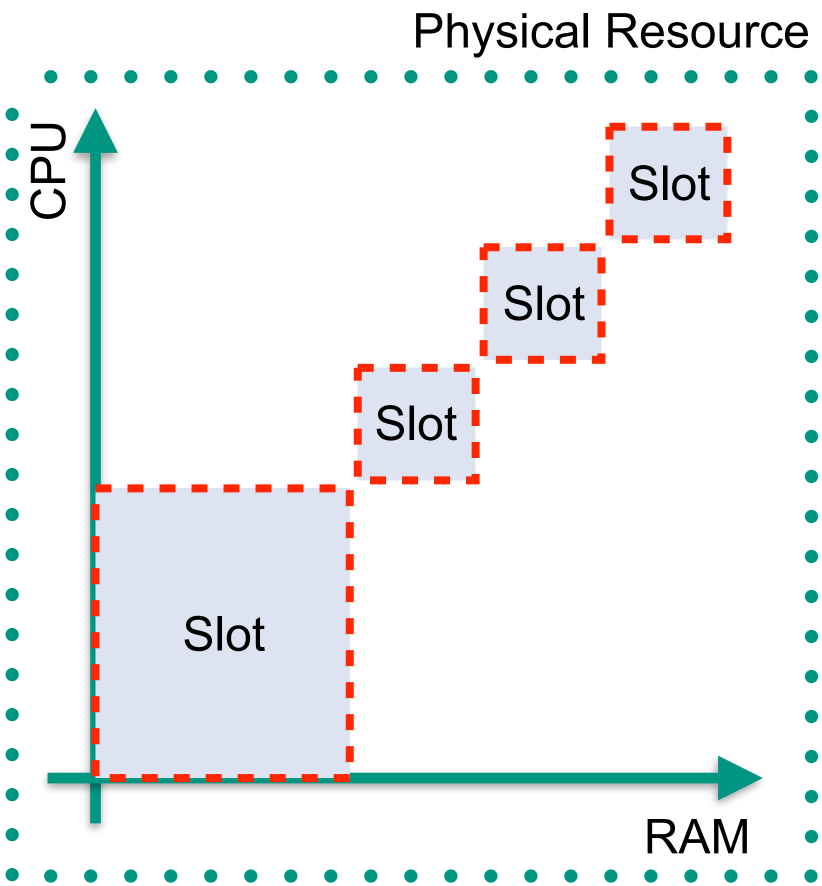
- Objective 1: Cluster Efficiency
  - Minimise unused resources
  - Run best jobs to fill resources
- Objective 2: User Fairshare
  - Minimise advantages for users
  - Run all jobs to match demand(s)





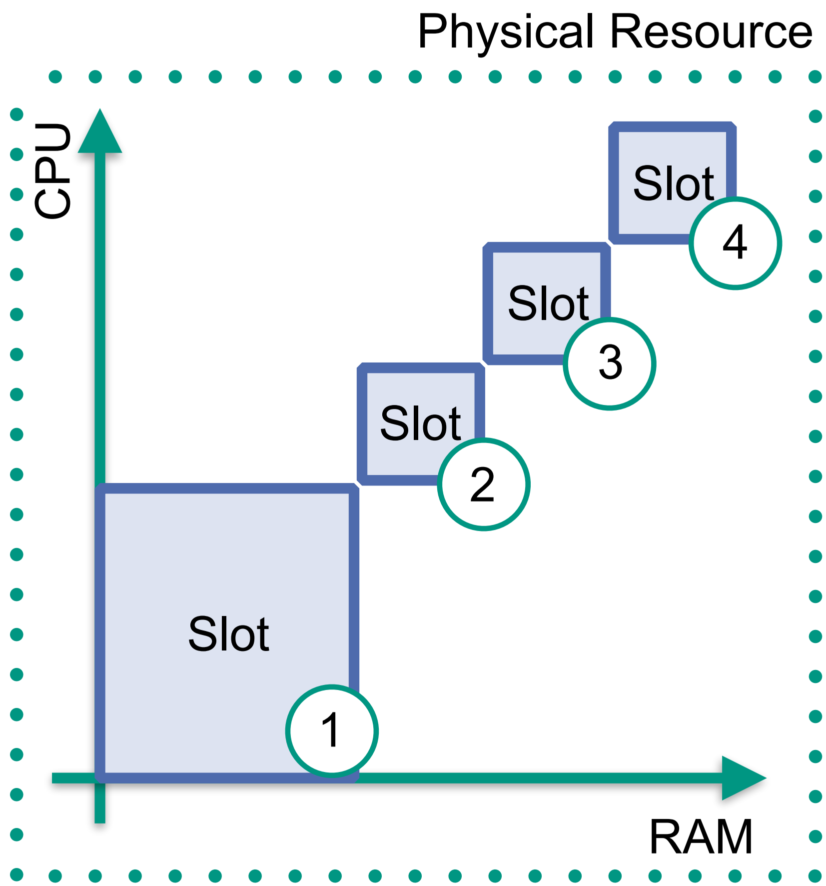
Approach A: Fixed Worker Reservations

- Fixed partitions/slots of resources
  - Each slot has guaranteed share
  - Customisable per worker
- Not suitable for dynamic demands
  - Inefficiency on job switch/lack
  - Adjustments local to workers



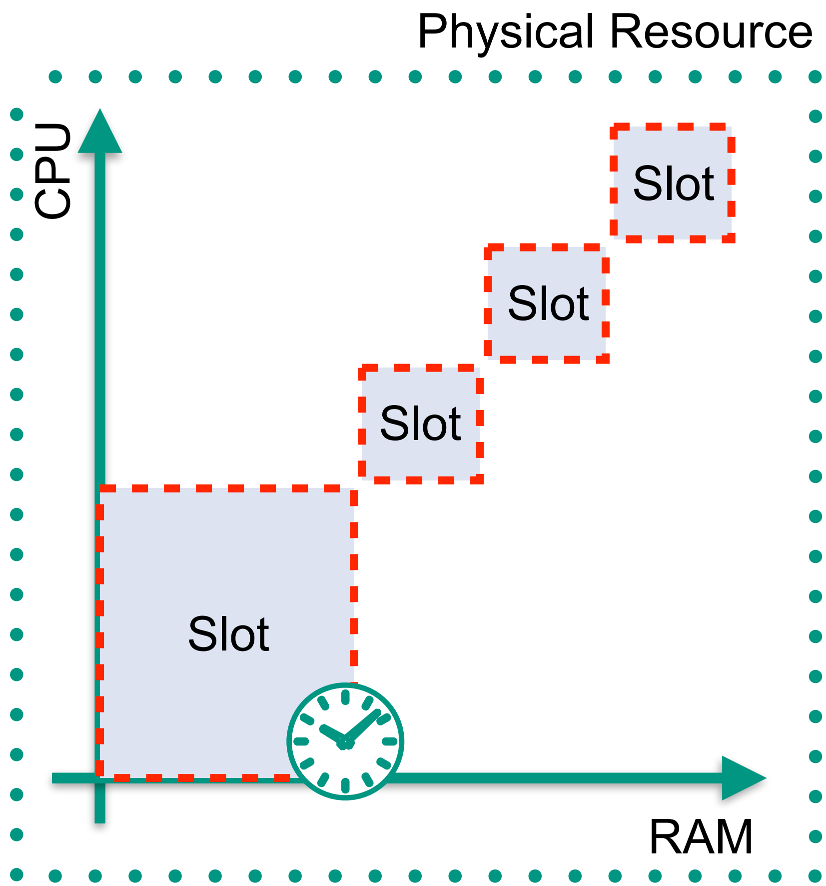
Approach B: Scheduling Priority

- Pick large jobs first for scheduling
  - Requires pseudo-groups
  - Single knob on negotiator
- Relies on stable user demand
  - Steady job pressure and mix
  - Antithesis to fair share



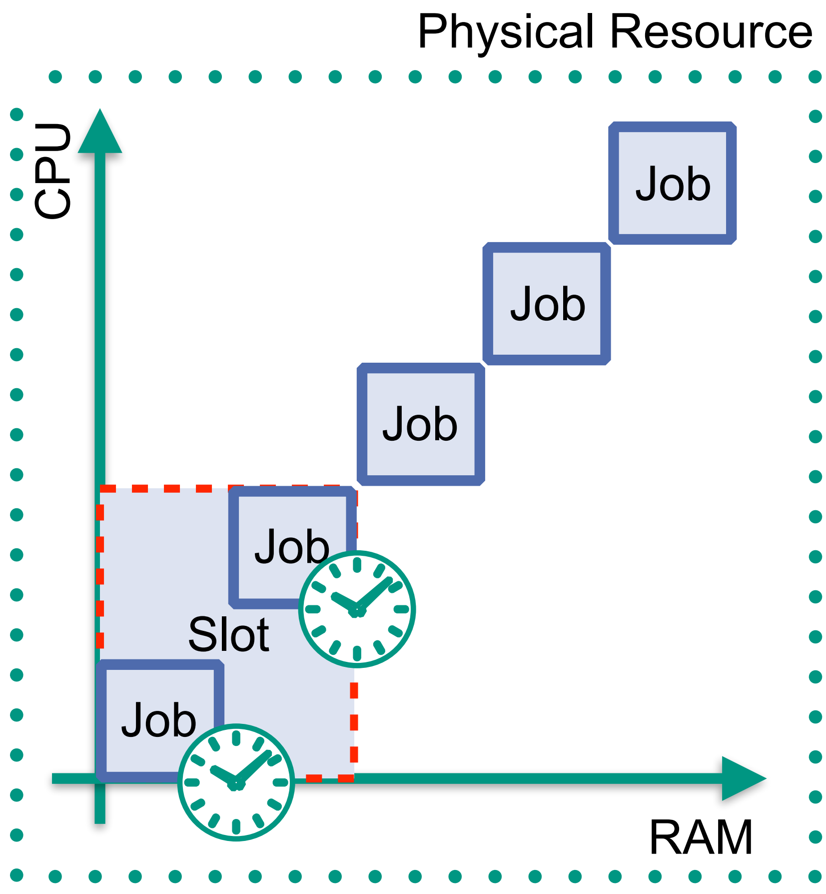
Approach C: Slot Grace Period

- Protect large slots shortly
  - Only perfect matches allowed
  - Comparable to reservation
- Weak partitioning of cluster
  - Jobs start freely on their share
  - Ideal for long and stable jobs



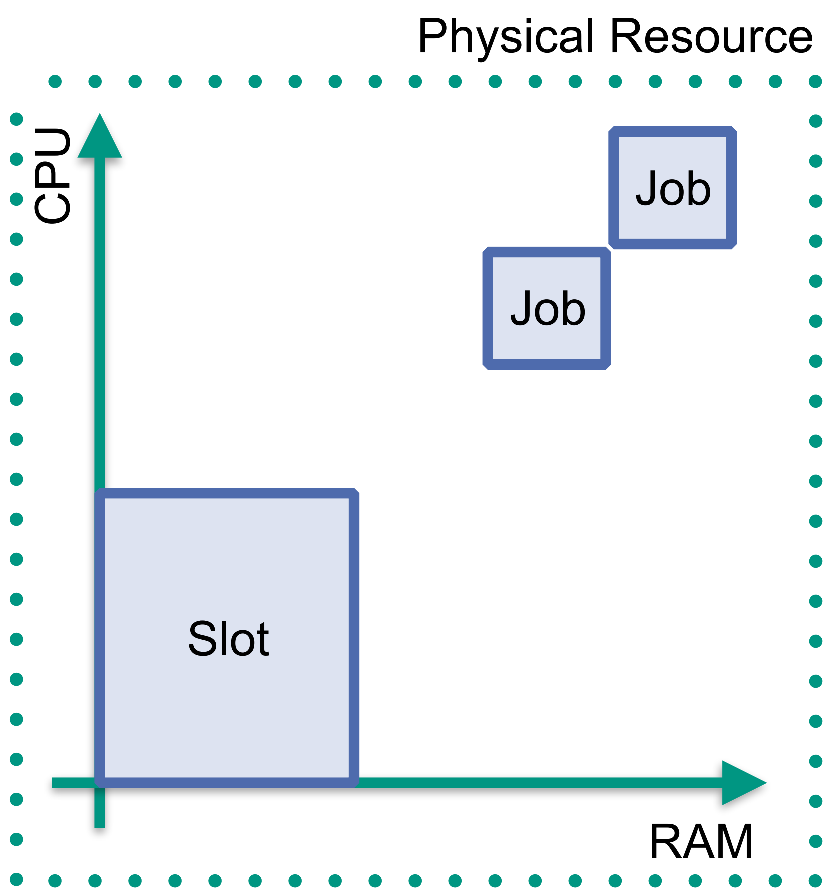
Approach D: DEFRAG

- Drain workers to gain large slots
  - Global daemon picks workers
  - Absolute or relative strategy
- Damage control for fragmentation
  - Recovers fragmented workers
  - Drain rate limits responsiveness



Approach E: Void

- Just ignore fragmentation
  - Rely on quotas, priorities, …
  - Easy to set up (who knew?)
- Shuffle small jobs for least hurt
  - Depth first filling of workers
  - Needs high turnaround of jobs



Useful notes on Approaches A-E

Karlsruher Institut für Technologie

- Preventing bad states is preferable
  - Difficult to recover from fragmentation
  - Help the scheduler converge to an ideal state

Useful notes on Approaches A-E

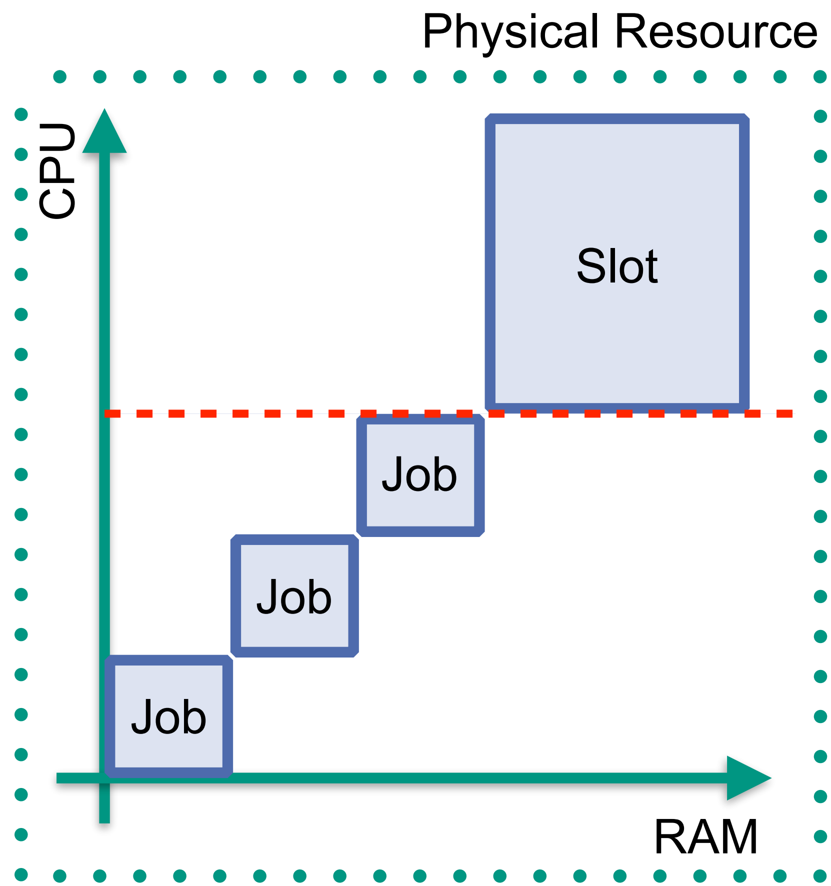
- Preventing bad states is preferable
  - Difficult to recover from fragmentation
  - Help the scheduler converge to an ideal state
- Mixing approaches is a challenge in itself
  - Good luck understanding what goes on…
  - Best with global + local approach (Priorities + DEFRAG)

Useful notes on Approaches A-E

- Preventing bad states is preferable
  - Difficult to recover from fragmentation
  - Help the scheduler converge to an ideal state
- Mixing approaches is a challenge in itself
  - Good luck understanding what goes on…
  - Best with global + local approach (Priorities + DEFRAG)
- Should stay responsive to changes in user demands
  - Be prepared for *unintentionally malicious* users
  - Global knobs preferable for fast adjustment

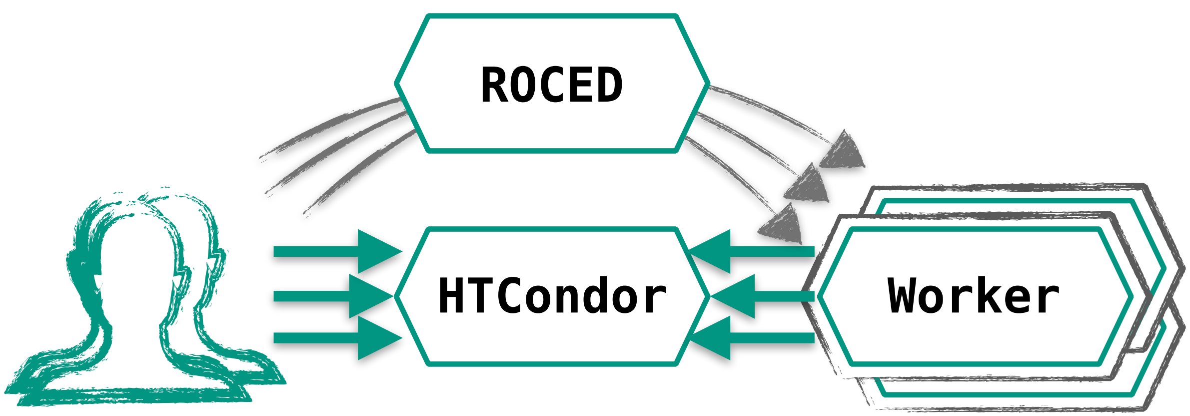
Approach F: ConcurrencyLimits

- Tag jobs as using made-up resource
  - Allows for global limits
  - Tagging done by wrapper/CE
- Create virtual partition inside cluster
  - Jobs freely allocated to workers
  - Hard limit on what can go wrong



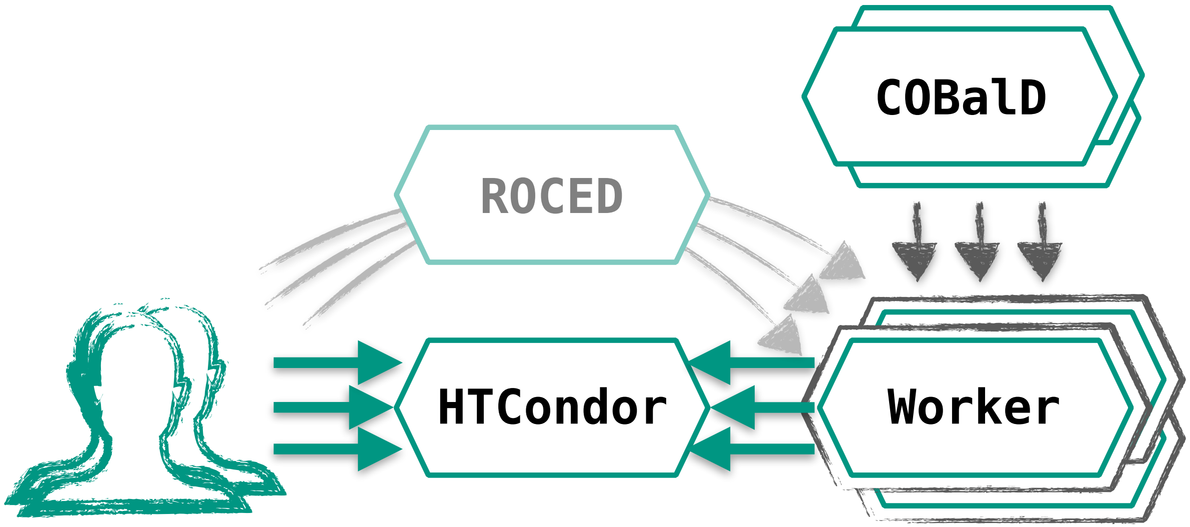
concurrency_limits=slots.mc:8
concurrency_limits=slots.sc:1

What's the Cloud got to do with it?



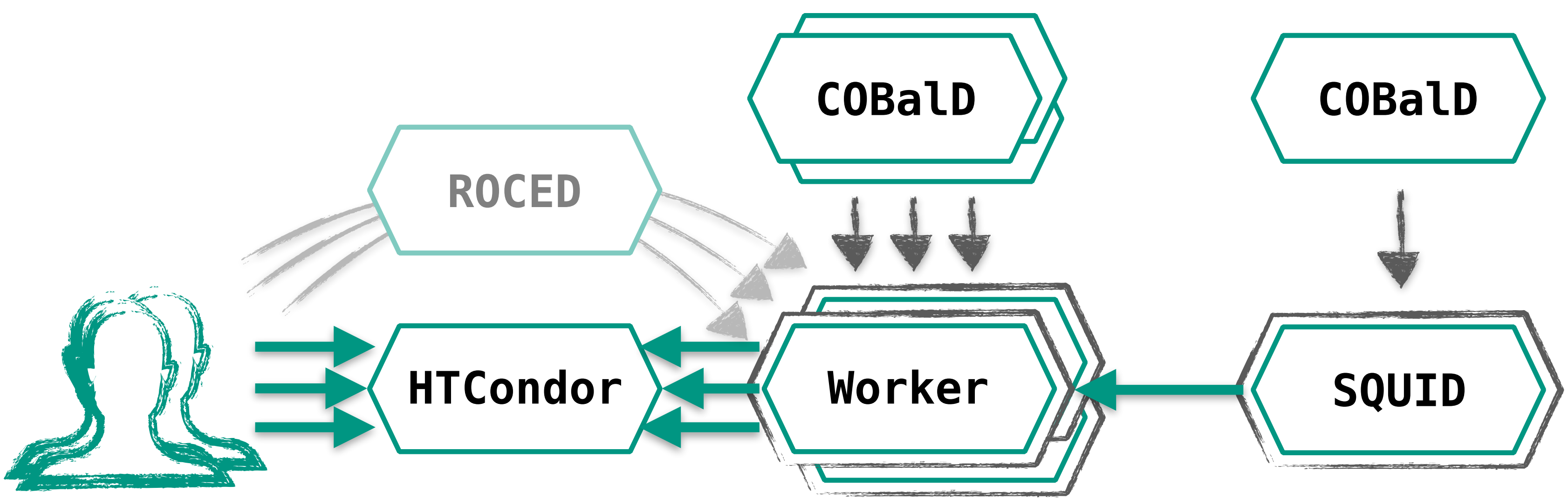
What's the Cloud got to do with it?

- Investigating new approach for opportunistic resources
  - Monitor usage to see what to add/remove
  - Batch System Scheduler decides what is useful



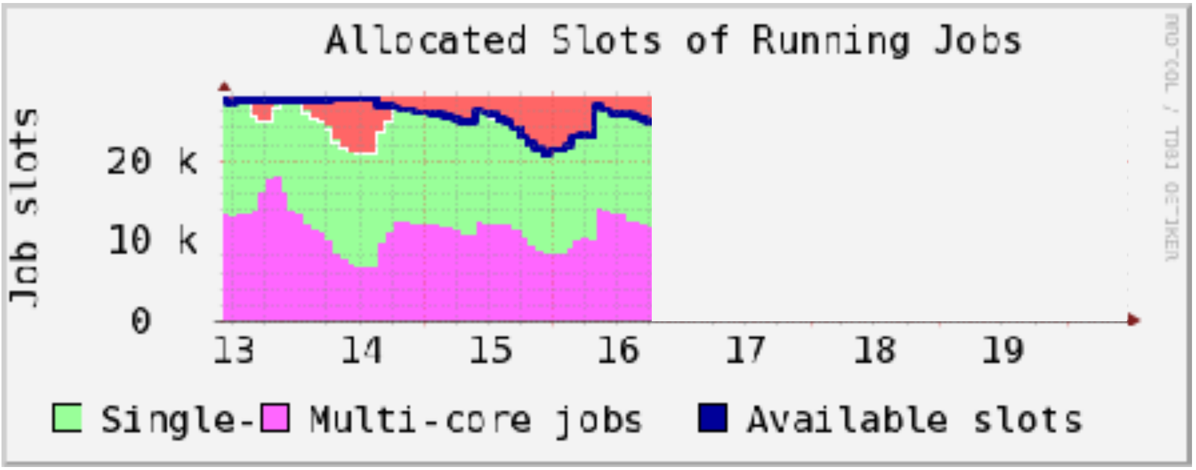
What's the Cloud got to do with it?

- Investigating new approach for opportunistic resources
  - Monitor usage to see what to add/remove
  - Batch System Scheduler decides what is useful
- Not restricted to batch system + workers + jobs
  - Just care for an abstract *good* and *bad*
  - Now what could we that for…



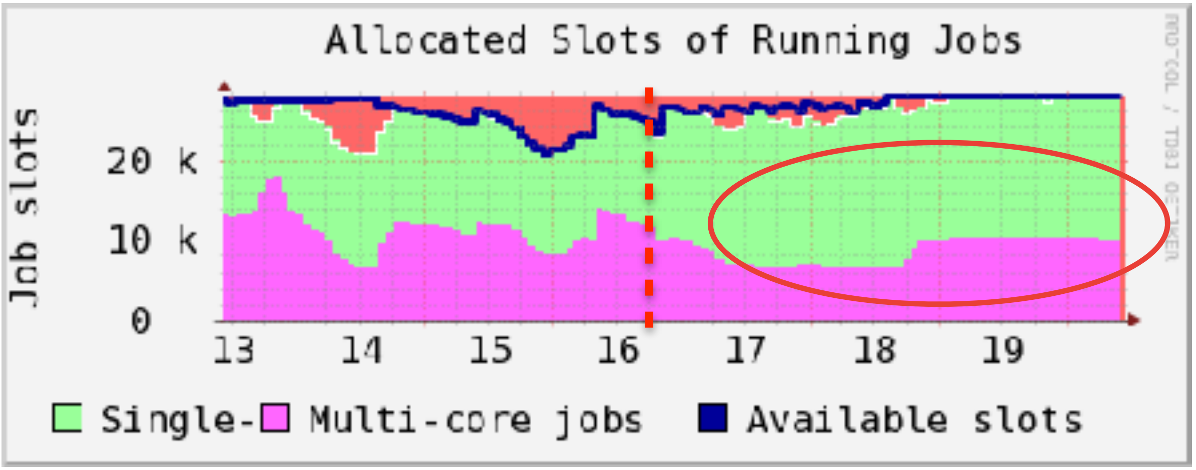
Flexibility Study: Condor ConcurrencyLimits

- GridKa limits small jobs via `ConcurrencyLimits`
  - Prevents fragmentation of large resource blocks
  - Big Red Button to change partitioning on the fly



Flexibility Study: Condor ConcurrencyLimits

- GridKa limits small jobs via ConcurrencyLimits
  - Prevents fragmentation of large resource blocks
  - Big Red Button to change partitioning on the fly
- COBalD manages this as single Resource Pool
  - Only access ConcurrencyLimits, no underlying Pilot/VM/…
  - Pool represents absence of SC slots
  - "If MC slots are not utilised, allow for more SC slots“



Conclusion

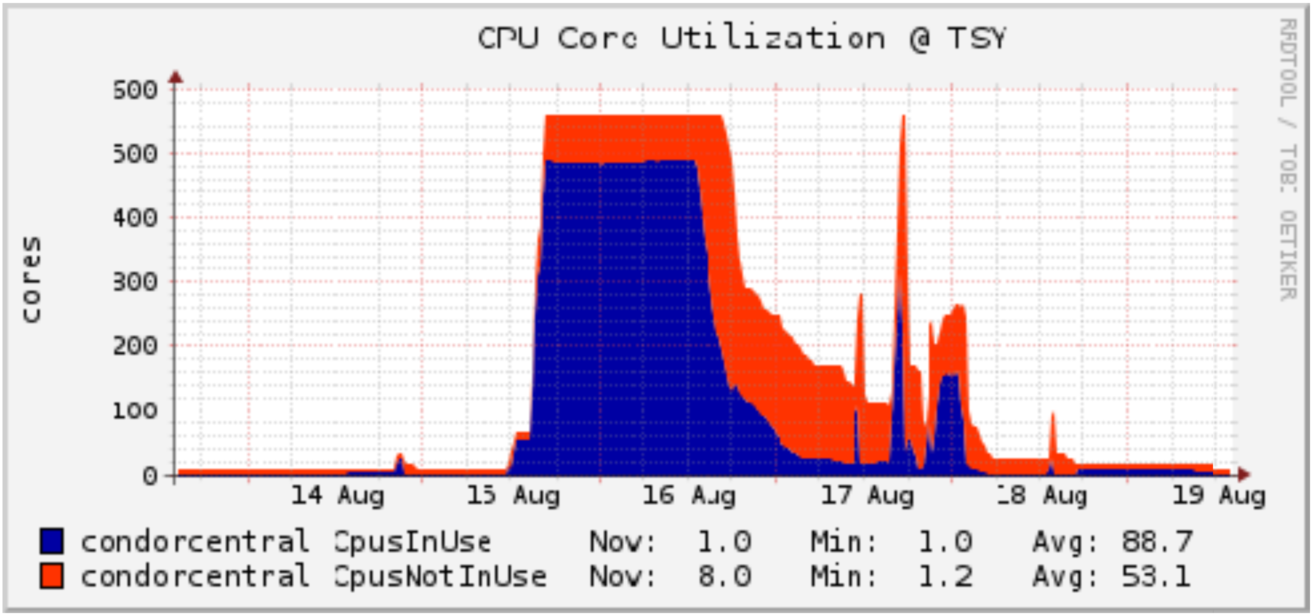
- Fragmentation poses significant challenge for large clusters
  - Not helped by whims of users, empty jobs, pilots, …
  - Several approaches to combat this problem
  - Take your pick and mix and match as needed
- What worked for us: ConcurrencyLimits
  - Tag jobs to allow for global limits of job groups
  - Easy and quick to setup, monitor and adjust
  - Scheduler can focus on efficiency and shares
- Not the end of the story - lots of knobs we can invent!
  - Managing ConcurrencyLimits like opportunistic resources works
  - Can we manage more than a binary system?

Backup

Resources

- COBalD: http://cobald.readthedocs.io/
- ROCED: https://github.com/roced-scheduler/ROCED
- TARDIS: https://github.com/giffels/tardis
- COBalD Simulation: https://git.scc.kit.edu/fq8360/cobalt_sim
- COBalD Demo: https://github.com/MaineKuehn/cobald_demo

ROCED Efficiency in HNSciCloud



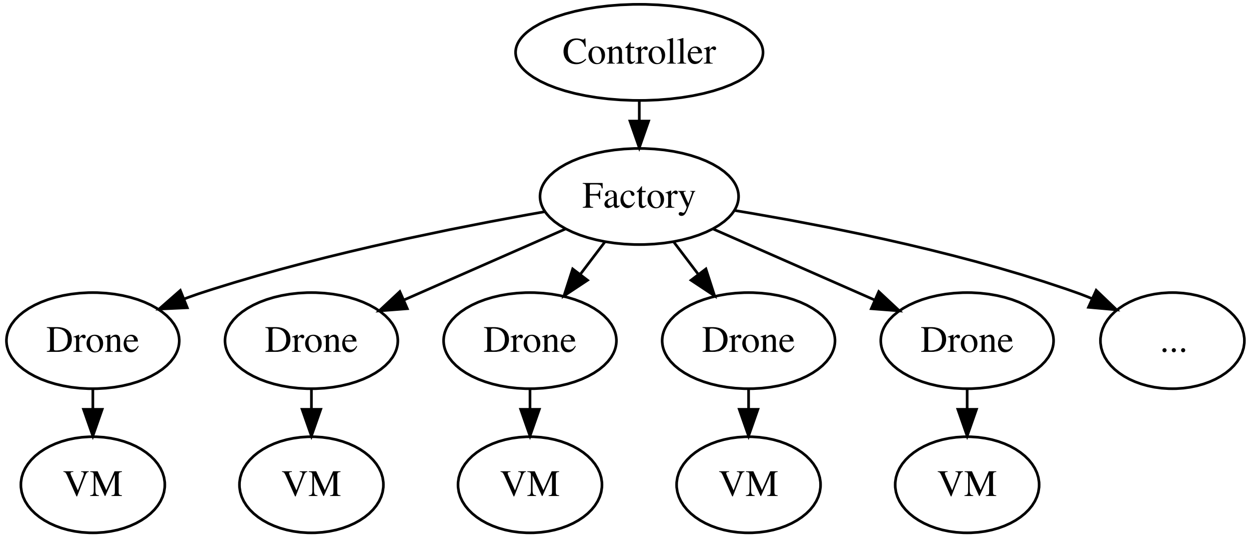
Current Work: HNSciCloud Integration

Karlsruher Institut für Technologie

- Integration of HNSciCloud resources with GridKa Batch System
  - Separate HTCondor instance for Exoscale/OTC VMs
  - Use Cloud resources *if* adequately utilised by Pilots

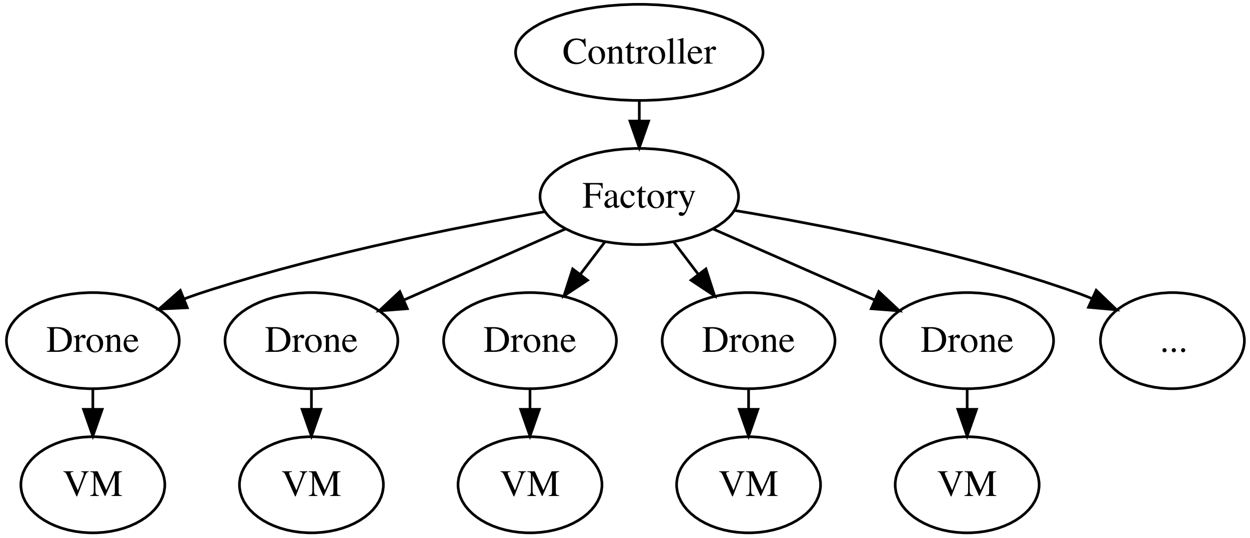
Current Work: HNSciCloud Integration

- Integration of HNSciCloud resources with GridKa Batch System
  - Separate HTCondor instance for Exoscale/OTC VMs
  - Use Cloud resources *if* adequately utilised by Pilots



Current Work: HNSciCloud Integration

- Integration of HNSciCloud resources with GridKa Batch System
  - Separate HTCondor instance for Exoscale/OTC VMs
  - Use Cloud resources *if* adequately utilised by Pilots
- TARDIS: Drone Pools aggregated in dynamically sized Composite
  - Each drone represents and manages single VM
  - Scalability from asynchronicity and composition

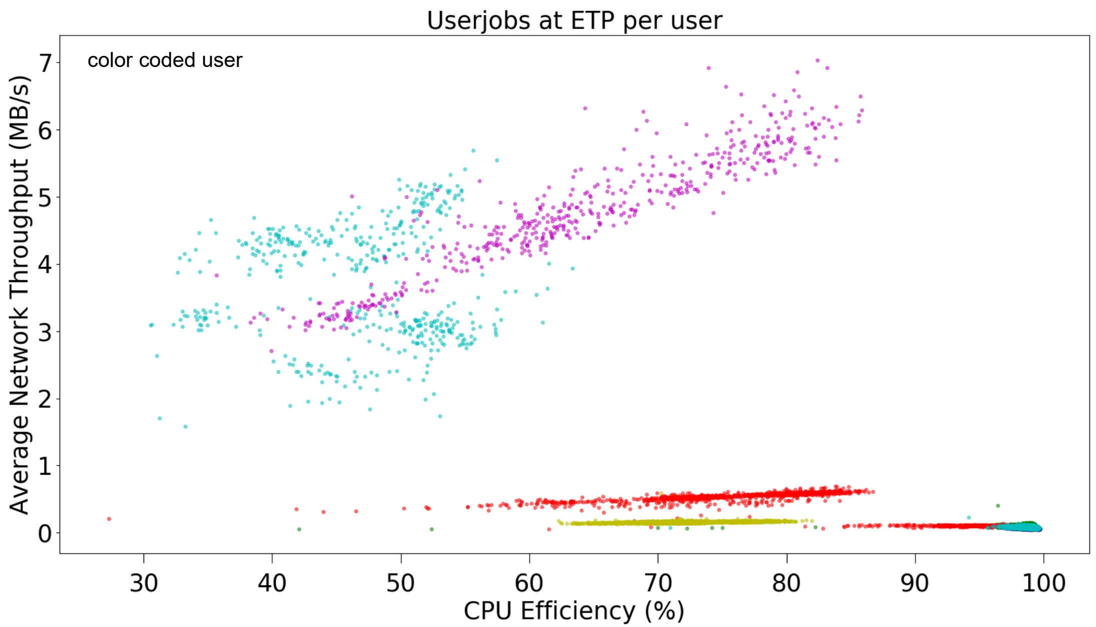


Current Research: Implicit Network Scheduling

- Integrate Network availability and congestion into provisioning
  - Congested network as bottleneck for opportunistic resources
  - Non-linear interference and noticeable measurement overhead

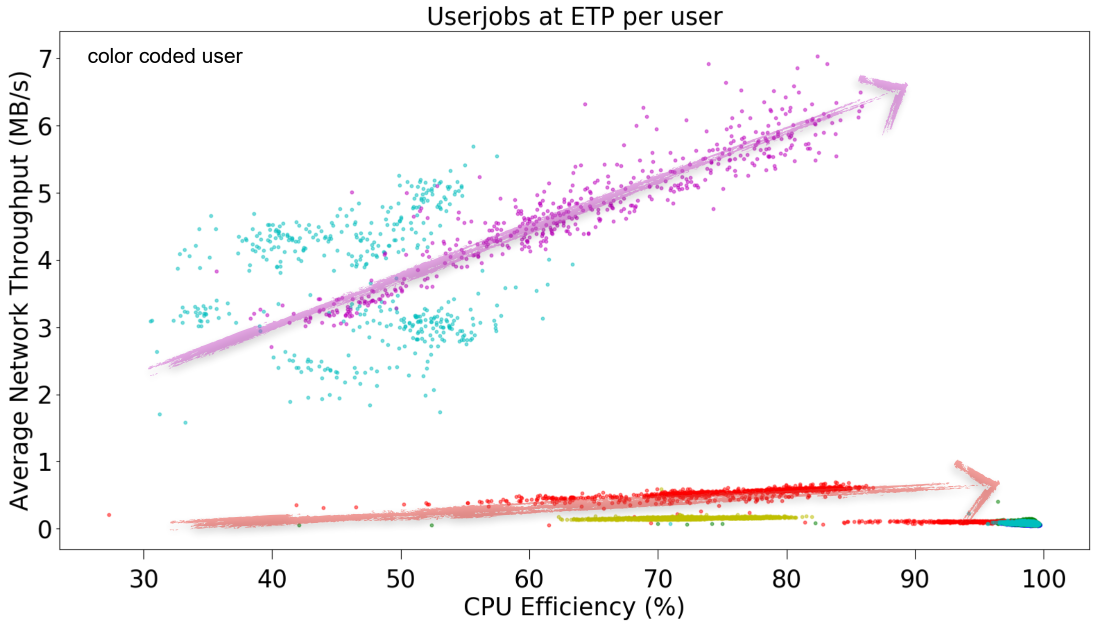
Current Research: Implicit Network Scheduling

- Integrate Network availability and congestion into provisioning
  - Congested network as bottleneck for opportunistic resources
  - Non-linear interference and noticeable measurement overhead



Current Research: Implicit Network Scheduling

- Integrate Network availability and congestion into provisioning
  - Congested network as bottleneck for opportunistic resources
  - Non-linear interference and noticeable measurement overhead
- Implicitly respect network congestion via side-effects
  - Prove CPU efficiency as utilisation to reflect congestion
  - Show benefit/limitation of optimising for CPU utilisation

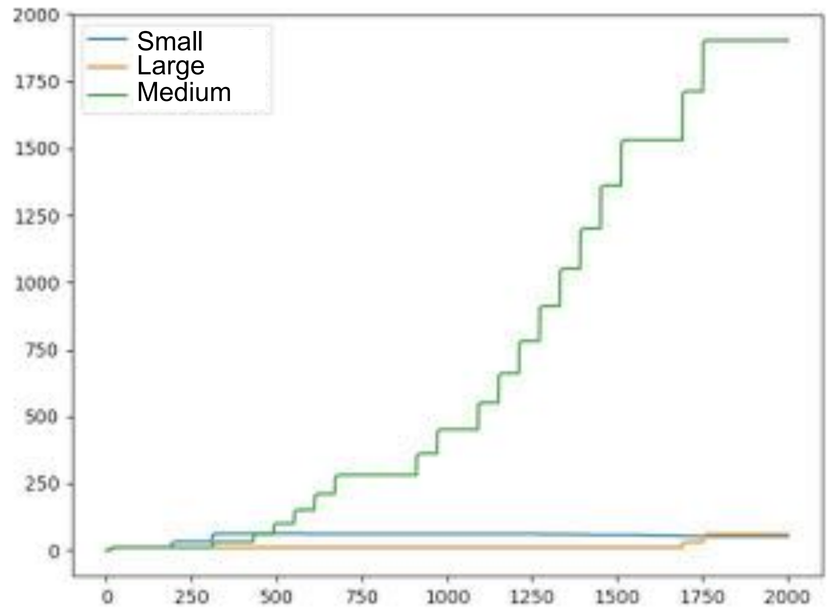


Current Research: Simulation and Strategies

- Modular design allows to replace active components
  - Simulated resource acquisition and utilisation with real Controllers
  - Replicated most of HTCondor queue and scheduling functionality

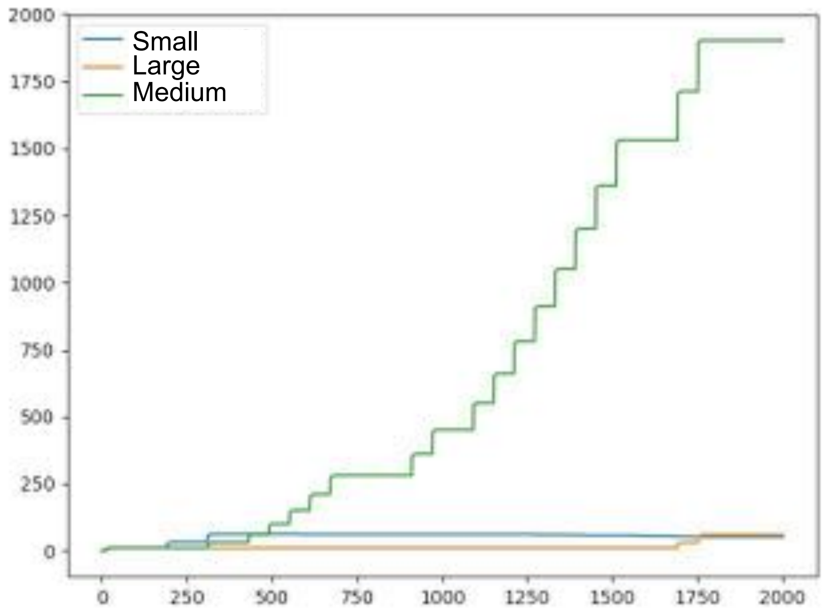
Current Research: Simulation and Strategies

- Modular design allows to replace active components
  - Simulated resource acquisition and utilisation with real Controllers
  - Replicated most of HTCondor queue and scheduling functionality

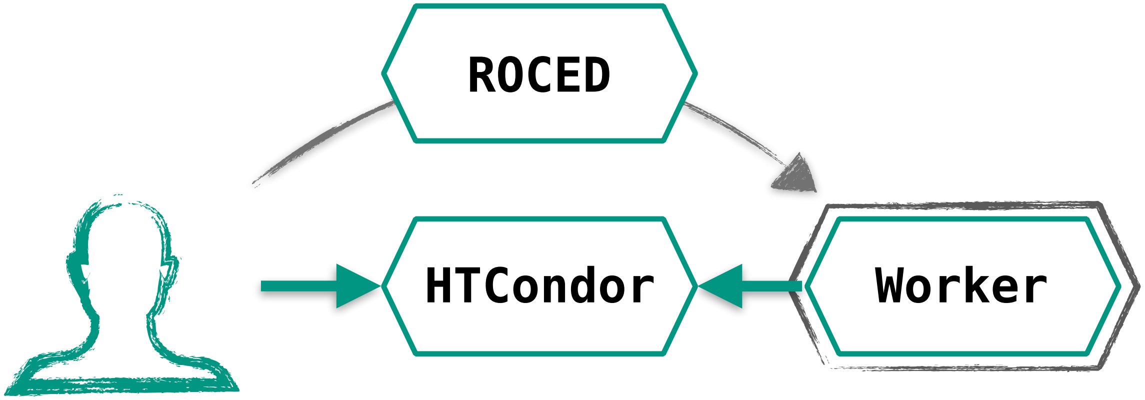


Current Research: Simulation and Strategies

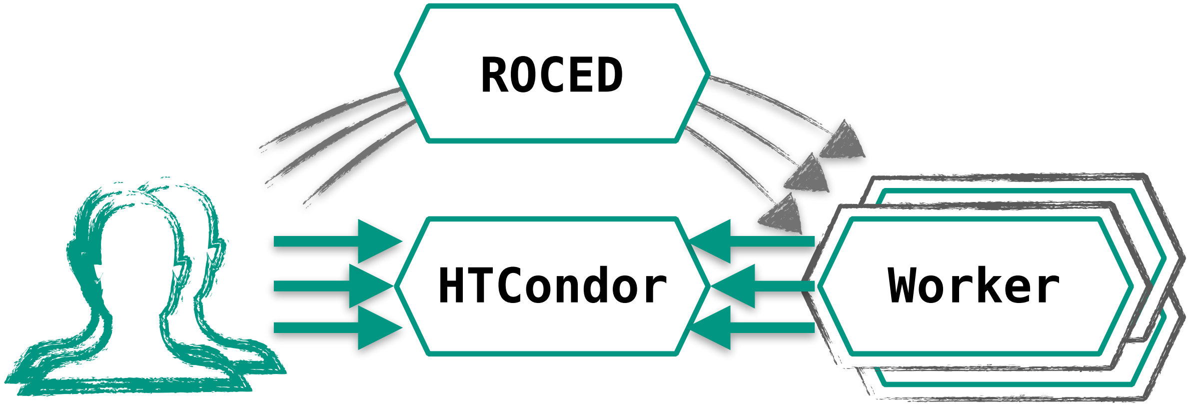
- Modular design allows to replace active components
  - Simulated resource acquisition and utilisation with real Controllers
  - Replicated most of HTCondor queue and scheduling functionality
- Investigating optimal control strategies for various situations
  - Responsiveness to fast increase/decrease in job pressure
  - Multiple pools providing different resources for job sizes



ROCED Resumé

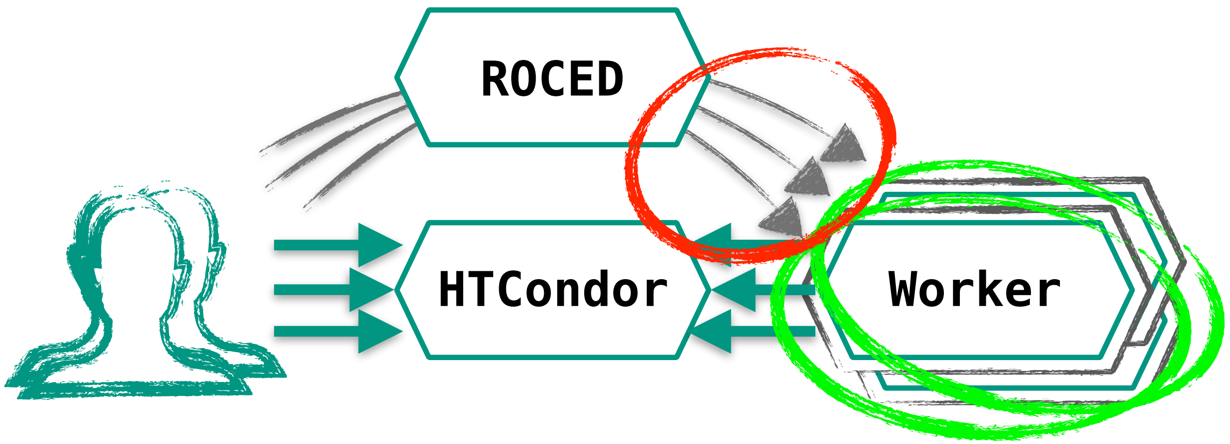


ROCED Resumé



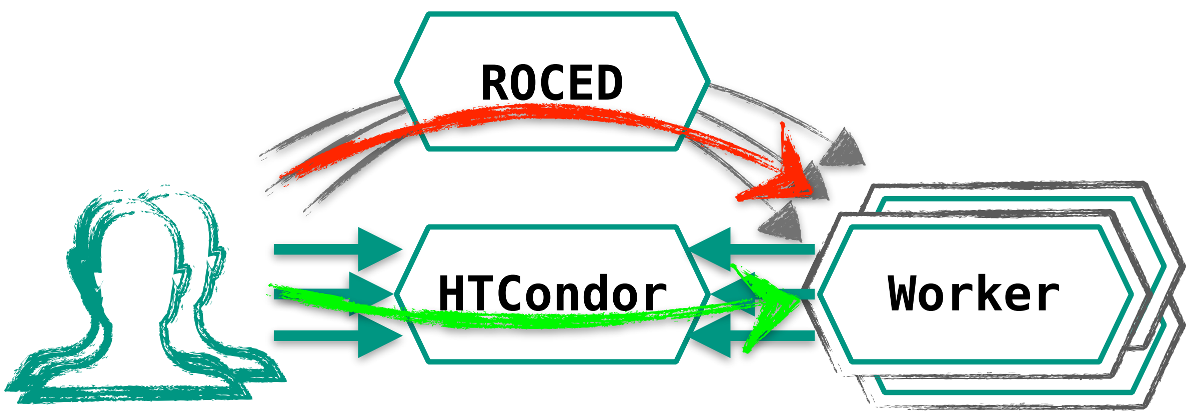
ROCED Resumé

- Dynamic resources matching user demand
  - Trivial to support new providers for many users
  - Difficult to manage several providers for many users



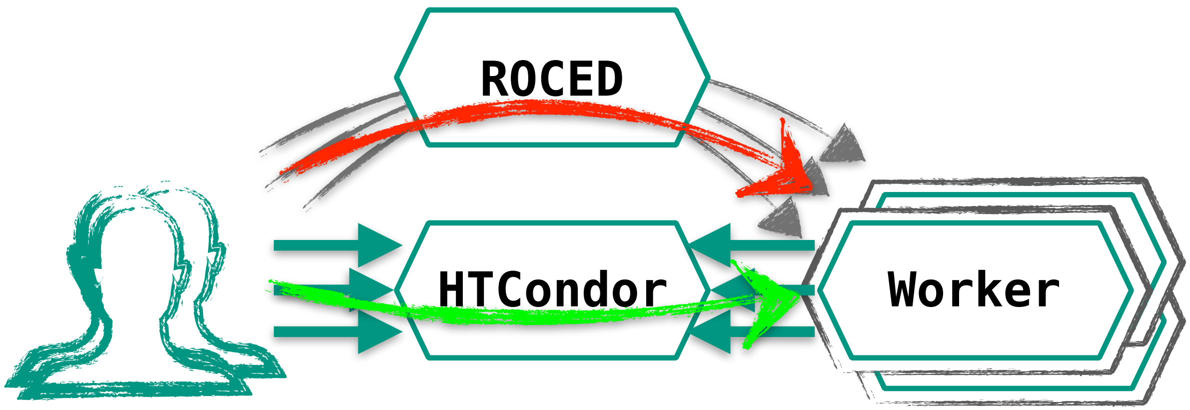
ROCED Resumé

- Dynamic resources matching user demand
  - Trivial to support new providers for many users
  - Difficult to manage several providers for many users
- Resource aggregation in overlay batch system
  - Unreliable to predict resources required for jobs
  - Efficient to integrate resources, then match jobs

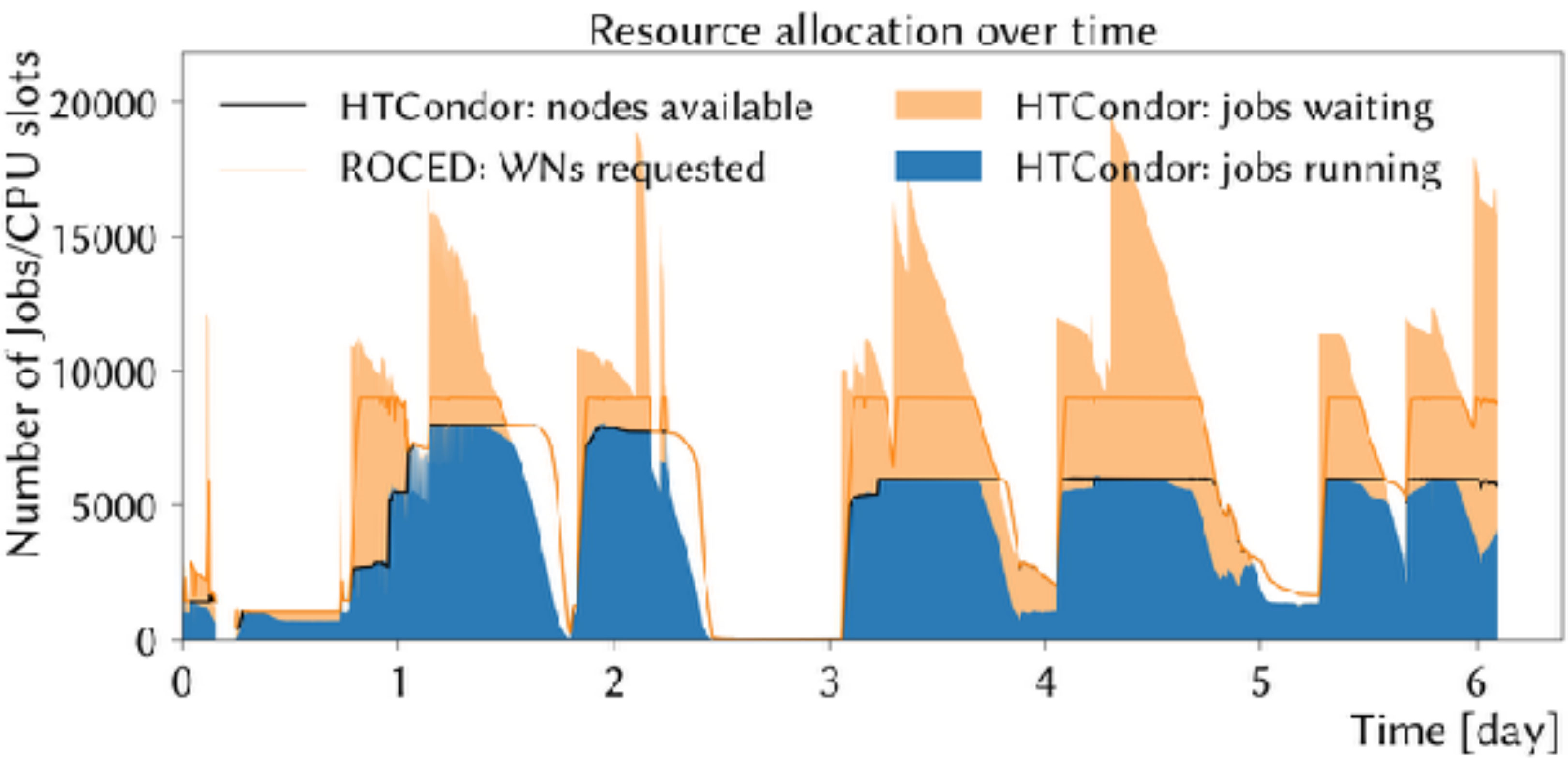


ROCED Resumé

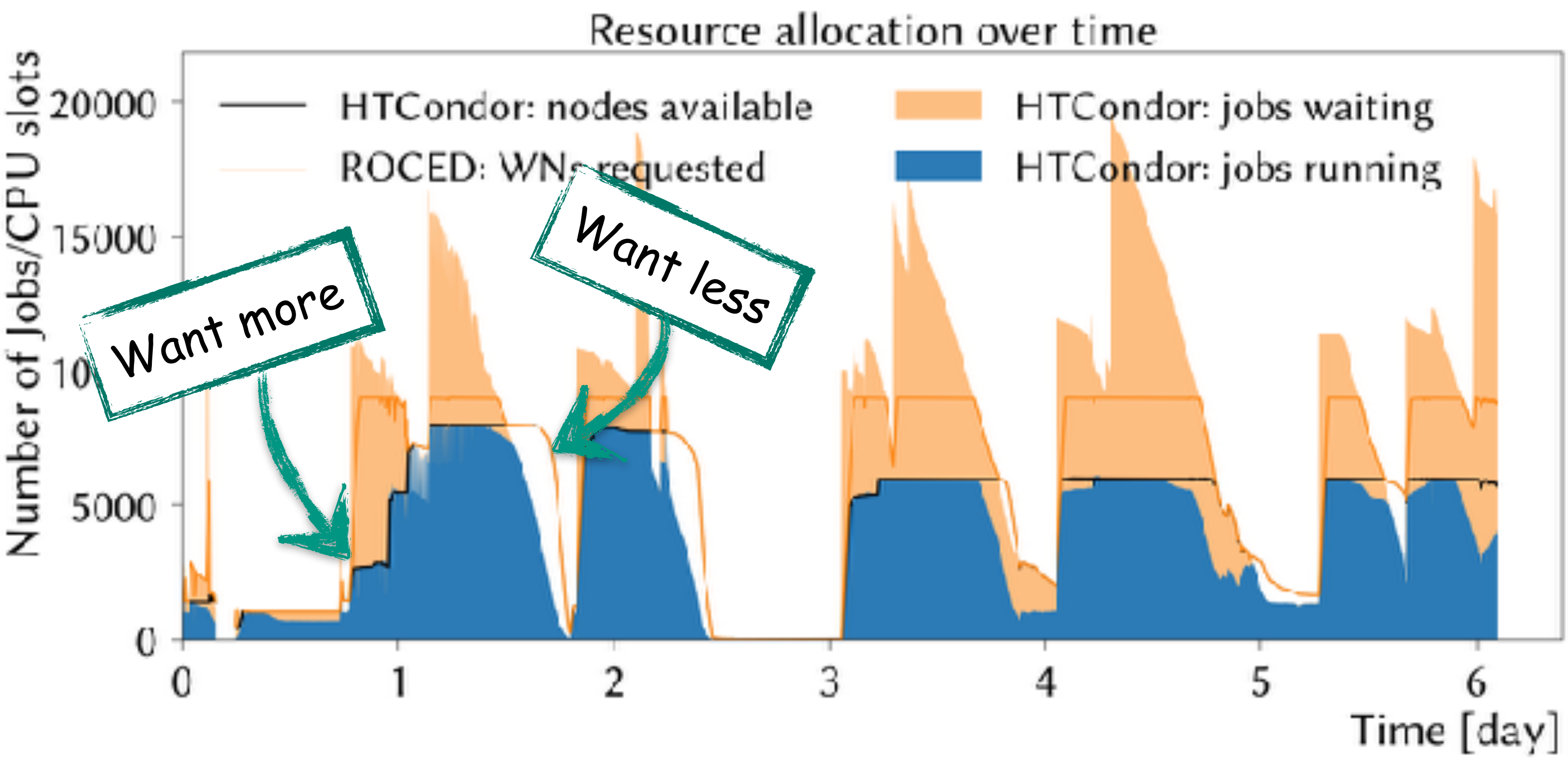
- Dynamic resources matching user demand
  - Trivial to support new providers for many users
  - Difficult to manage several providers for many users
- Resource aggregation in overlay batch system
  - Unreliable to predict resources required for jobs
  - Efficient to integrate resources, then match jobs
- Yet it really works!



Pragmatic View to Resource Management

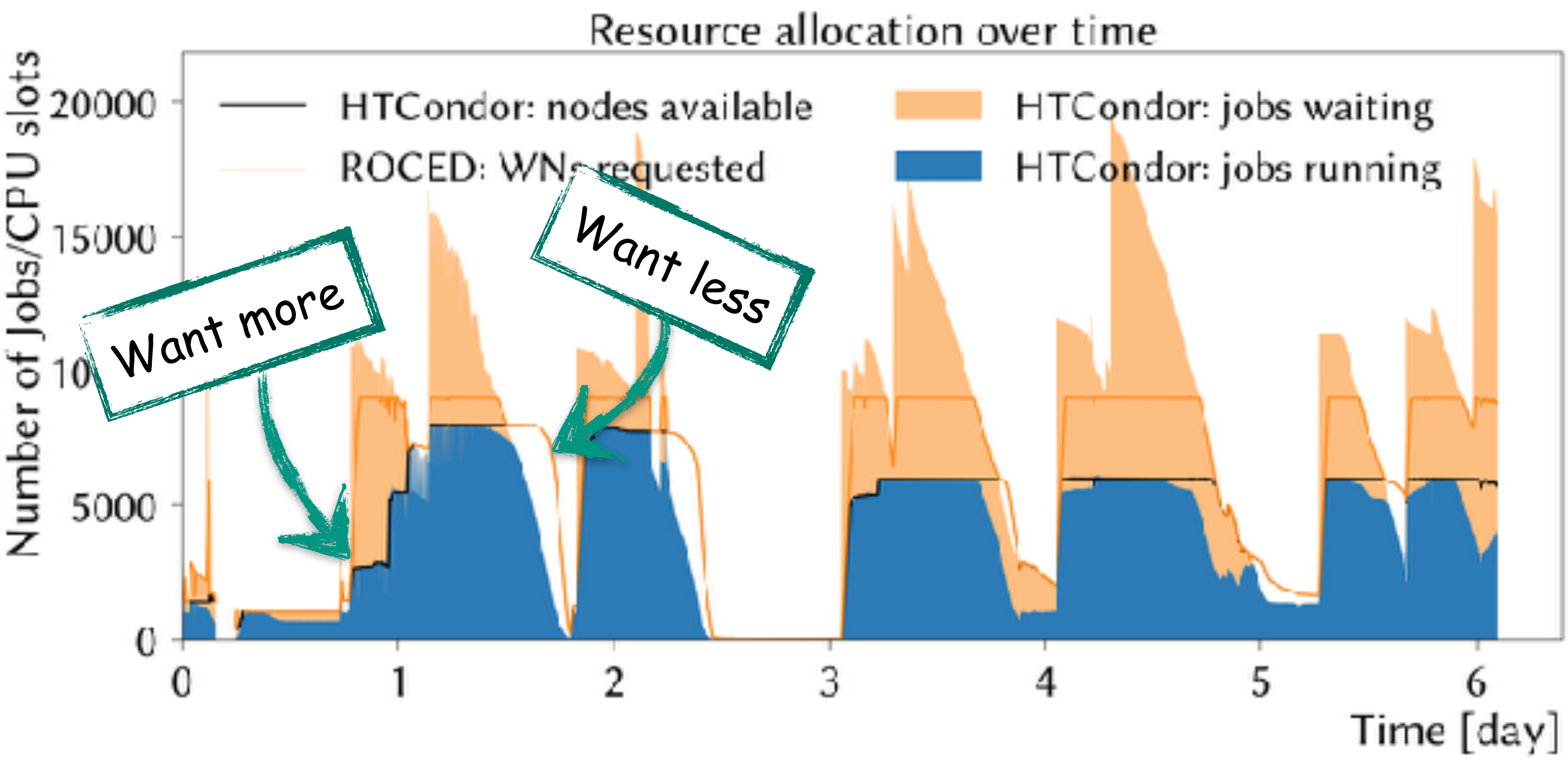


Pragmatic View to Resource Management



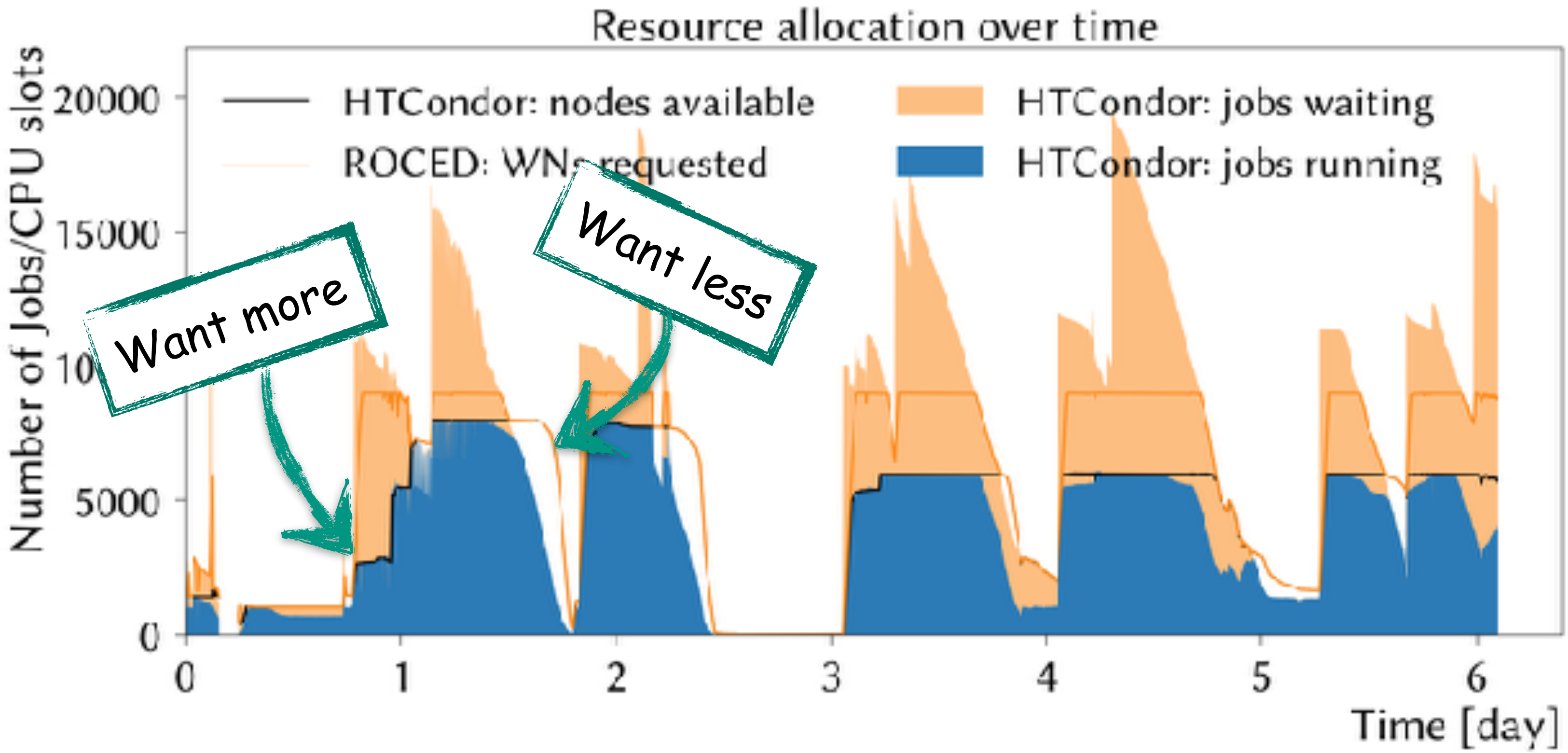
Pragmatic View to Resource Management

- New development for scalability and maintainability in HNSciCloud

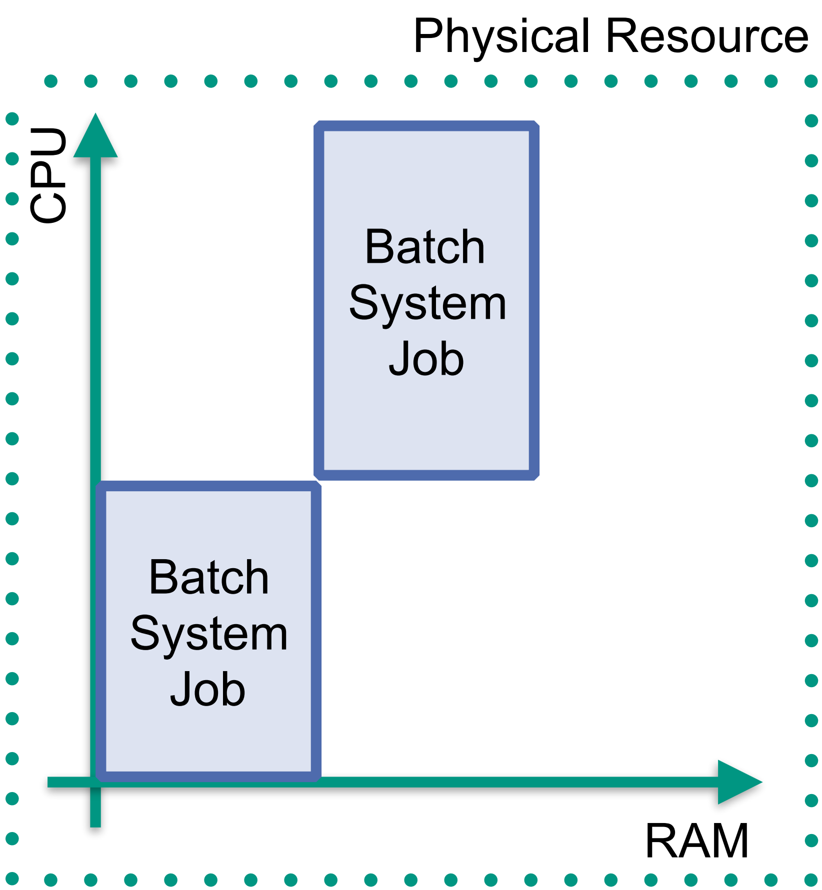


Pragmatic View to Resource Management

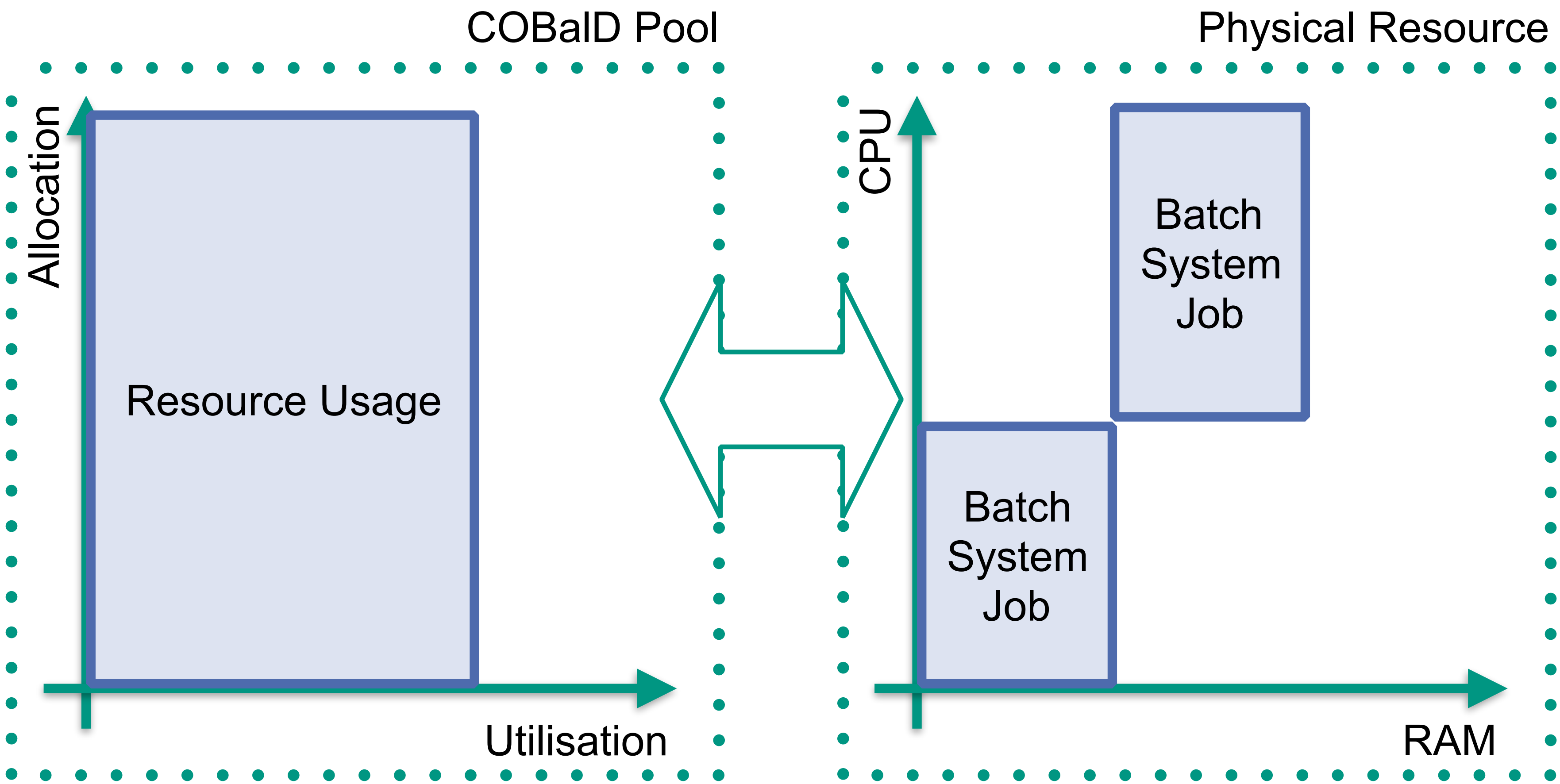
- New development for scalability and maintainability in HNSciCloud
- Simple logic: more used resources, less unused resources
  - COBalD only watches, creates and disables resources
  - Batch system scheduler selects appropriate resources



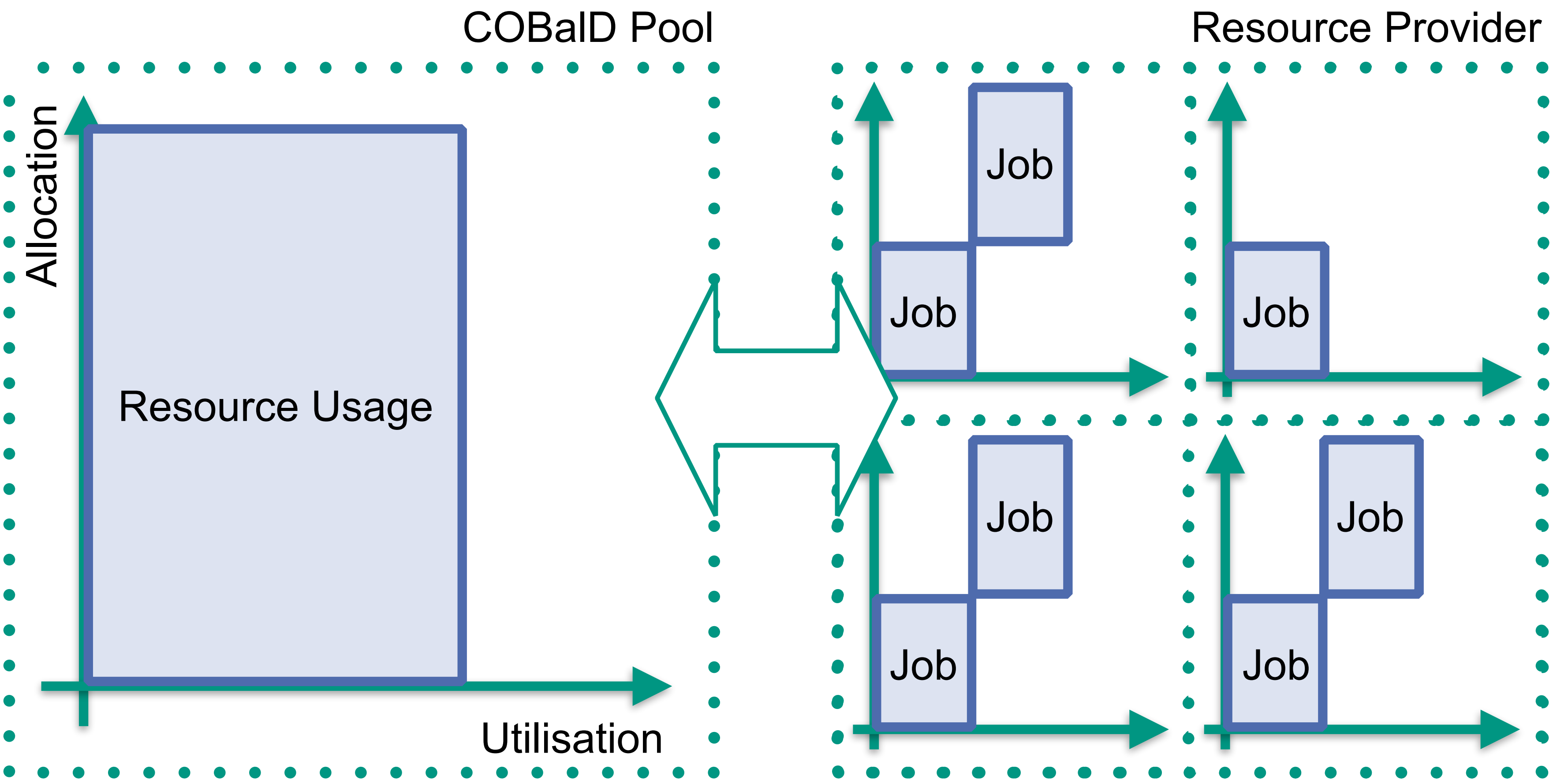
COBalD Resource Pool Model



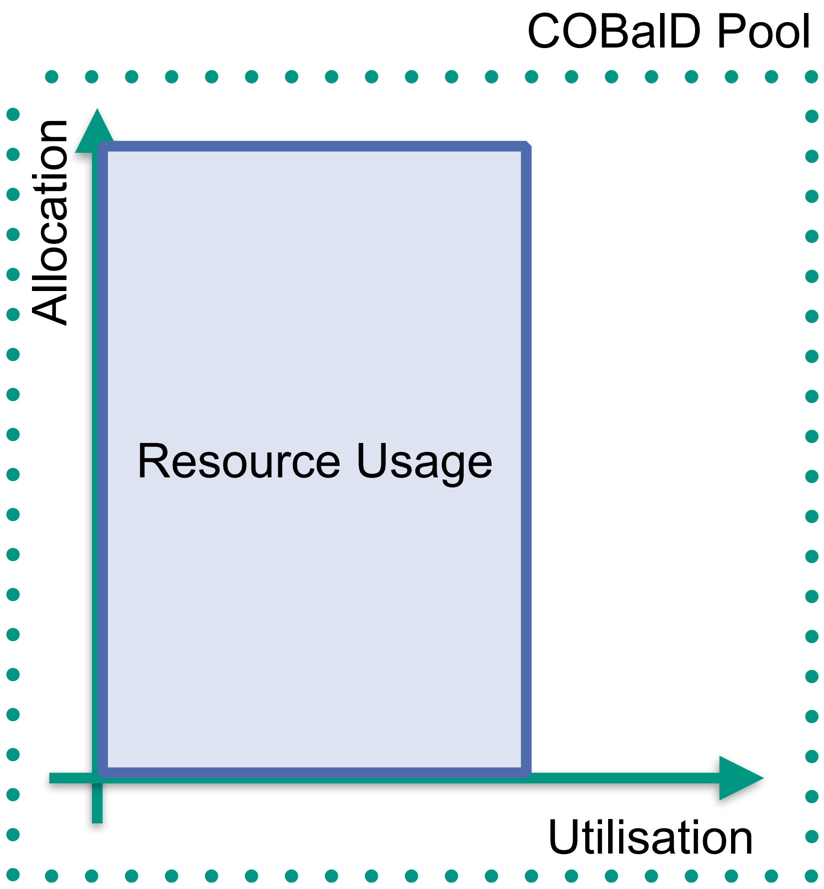
COBaID Resource Pool Model



COBalD Resource Pool Model

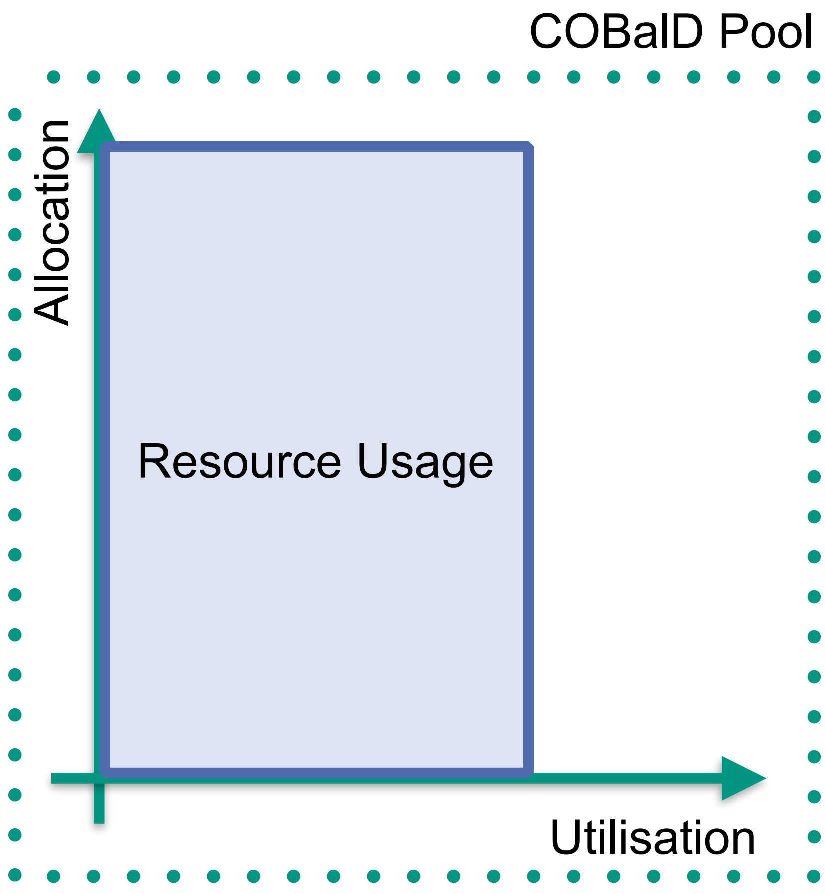


COBalD Resource Pool Model

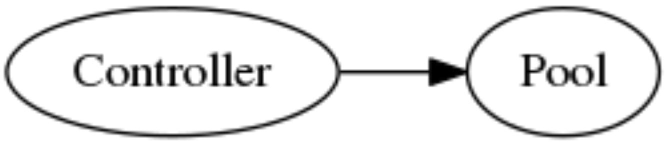


COBalD Resource Pool Model

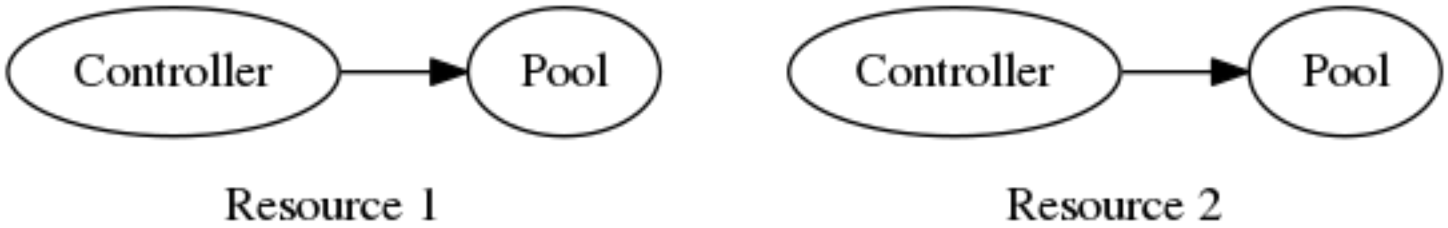
```
if utilisation < self.low_utilisation:
    return supply * self.low_scale
elif allocation > self.high_allocation:
    return supply * self.high_scale
```



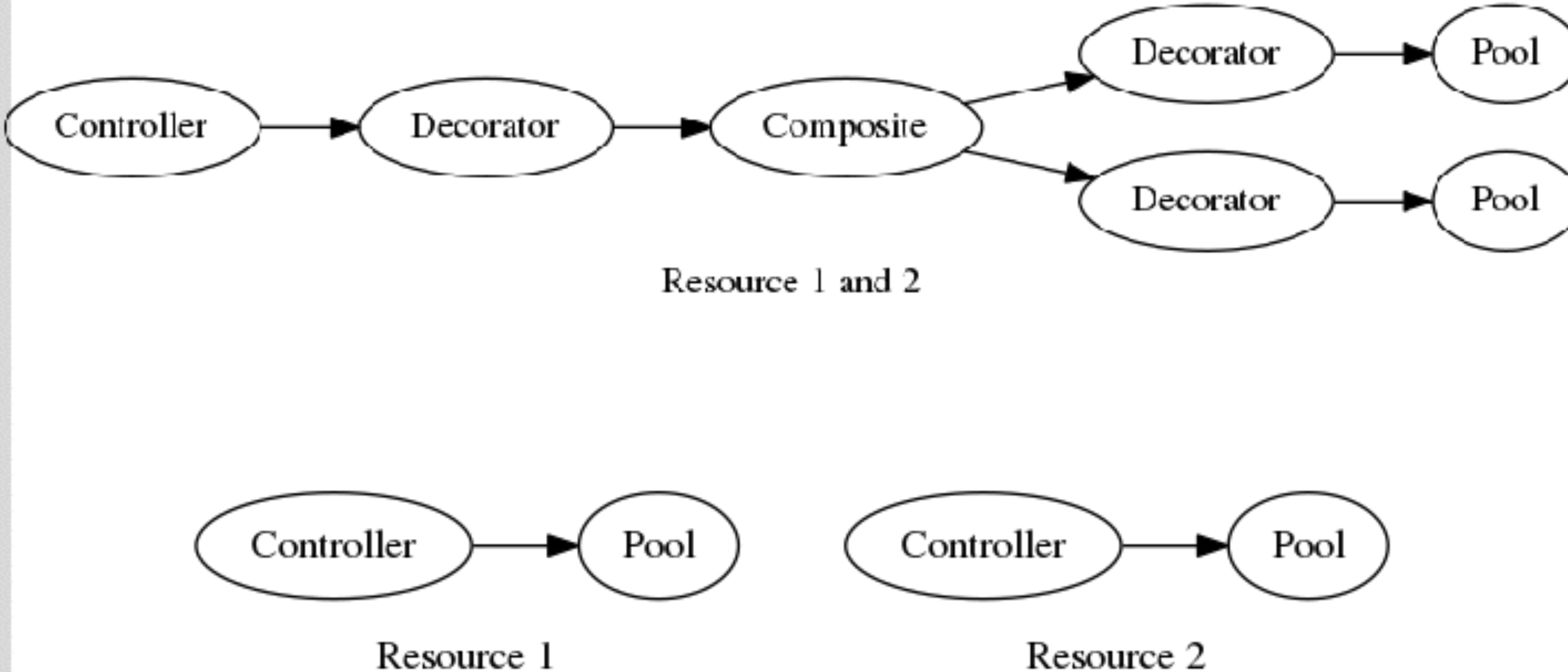
COBalD Resource Pool Model

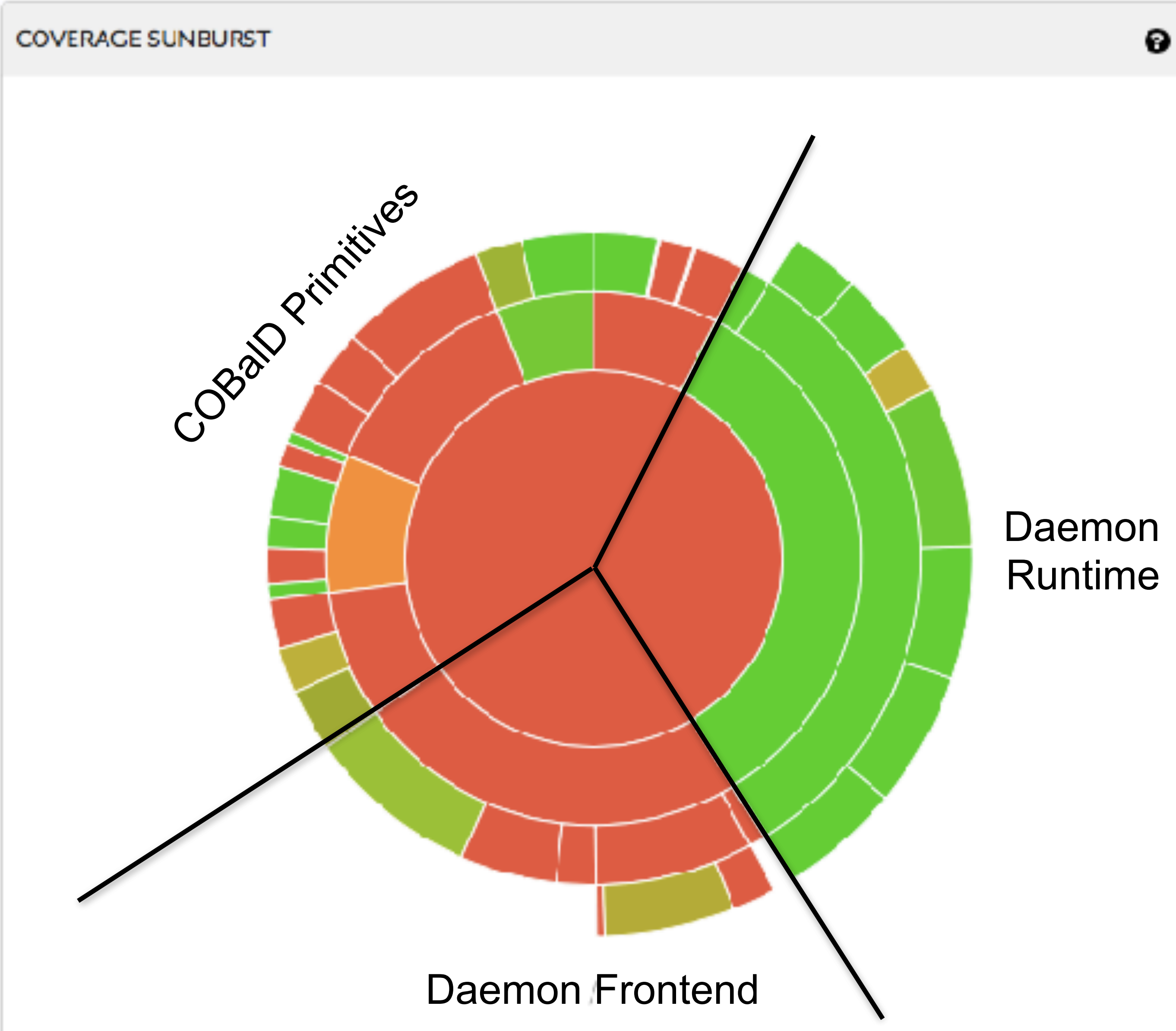


COBalD Resource Pool Model



COBalD Resource Pool Model



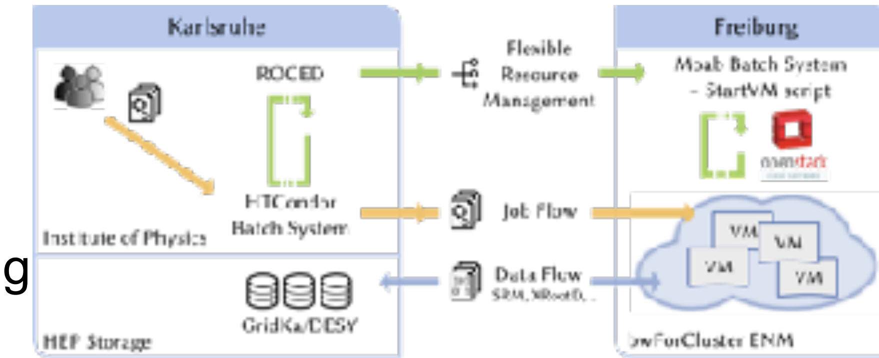

COVERAGE SUNBURST
COBalD Primitives
Daemon
Runtime
Daemon Frontend

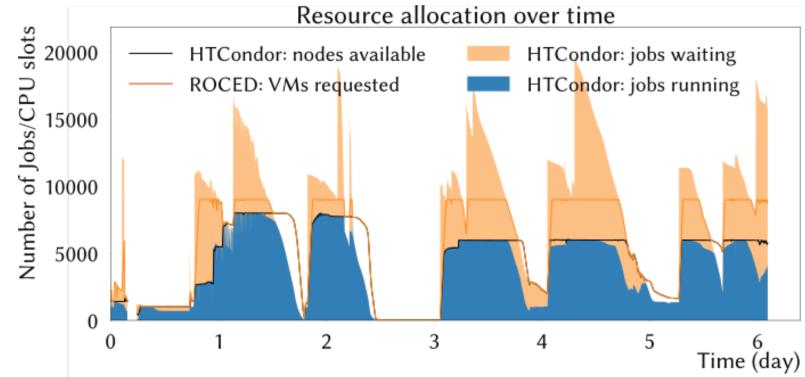
Opportunistic Resources

- ForHLR2 (KIT) - HPC
  - Backfilling and cycle stealing
  - Docker/Singularity container and permanent, shared cvmfs cache
- NEMO (Freiburg) - HPC
  - Fairshare in batch system
  - Virtual machines and semi-permanent Squids for cvmfs
- 1und1, Amazon, … - Public Cloud
  - Bought/donated resources
  - Virtual machines and temporary Squids for cvmfs
- ETP Desktop (KIT) - Desktops
  - Cycle stealing (day), temporary worker nodes (night)
  - Docker Container and permanent Squids for cvmfs
- GridKa - Grid Site
  - Fairshare in batch system
  - Pilot jobs and existing cvmfs

Success Story - Opportunistic “Tier 1“ for a Day

- Dynamically shared HPC Centre at Freiburg (three diverse communities)
- Virtualization is key component to:
  - Allow dynamic resource partitioning
  - Meet OS & software requirements
- ROCED cloud scheduler developed at KIT
  - On-demand resource provisioning
  - Transparent resource integration
- Suitable for CPU-intense workflows

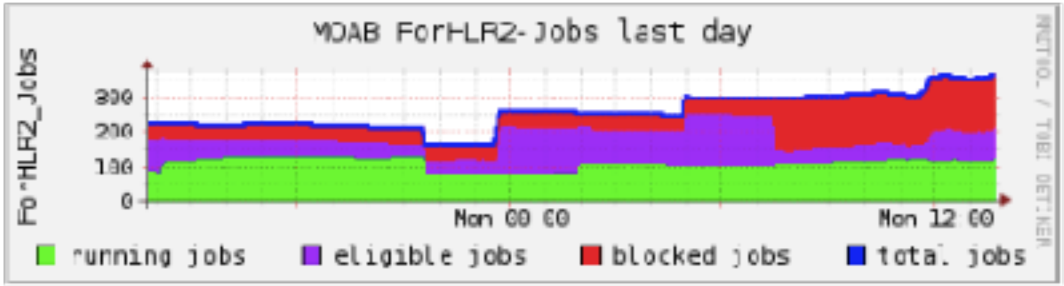


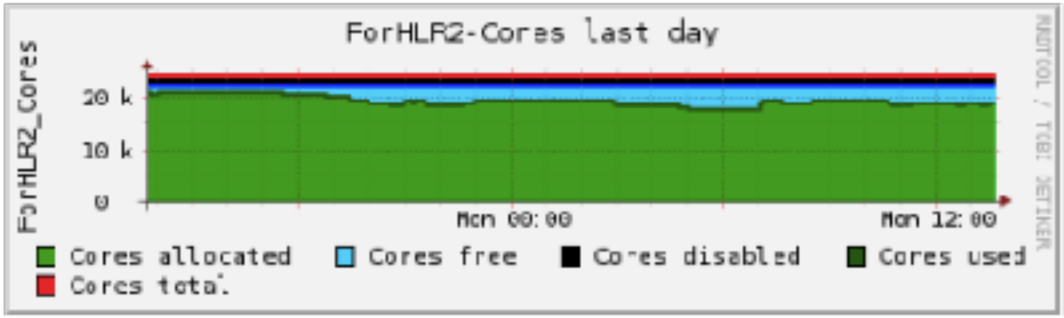


[ACAT17MS, JOP898TH, JOP762TH, JOP664TH]

Typical situation at HPC clusters

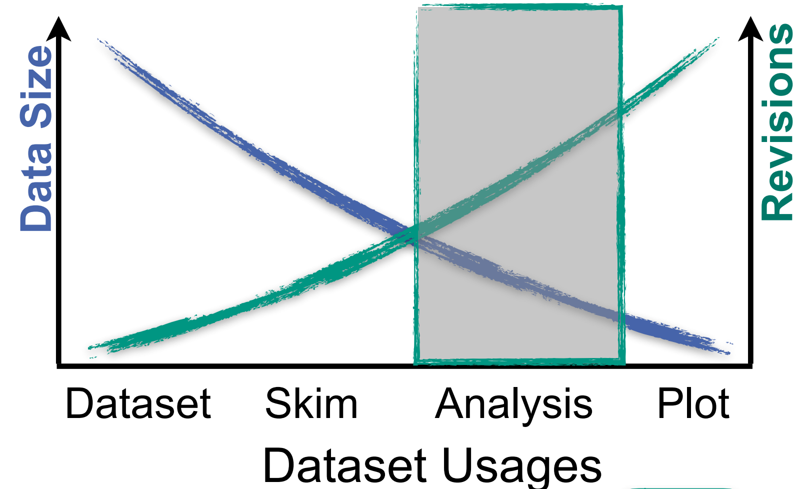
- small number of big multi node jobs
- unused cores / nodes due to scheduling of big jobs
- could back filled with short running single core jobs (e.g. HEP jobs)

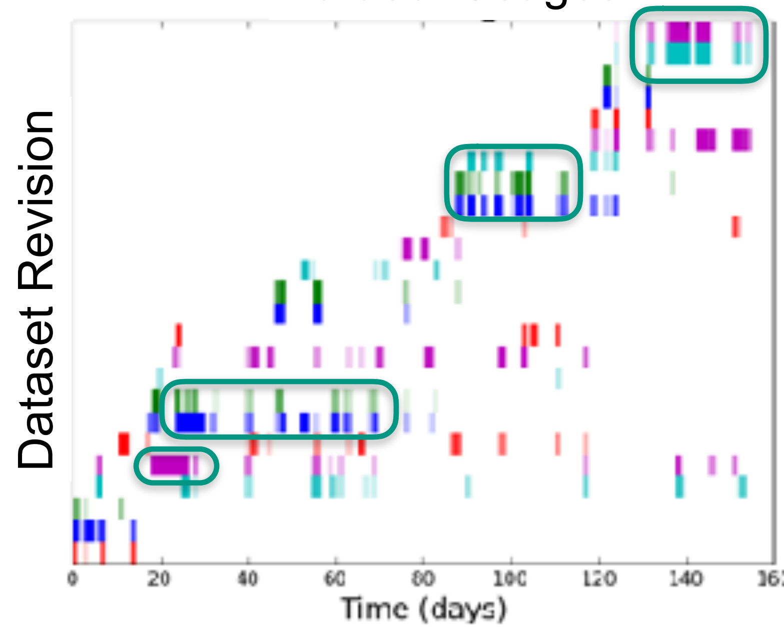




Context: HEP End User Data Analysis

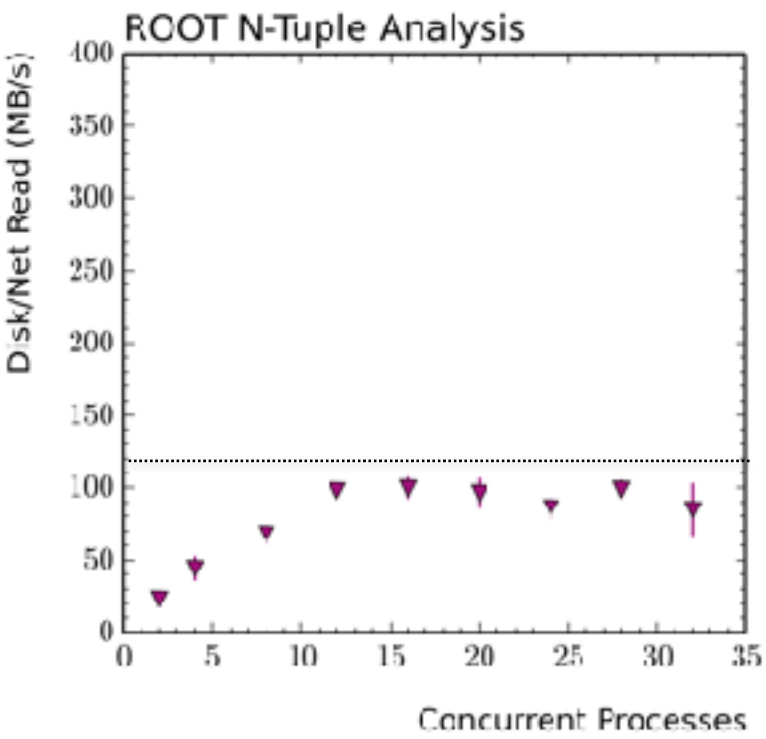
- Hierarchical, iterative workflows
  - Reduction of data size
  - Increase of iterations
  - Dedicated processing environments
- Data intense analyses on Tier 3
  - Local batch system
  - Network accessed fileservers
  - Optimized input data sets
- Usage suitable for caching
  - Repeated processing of same input
  - Highly dependent on input rate
  - Limited by network bandwidth

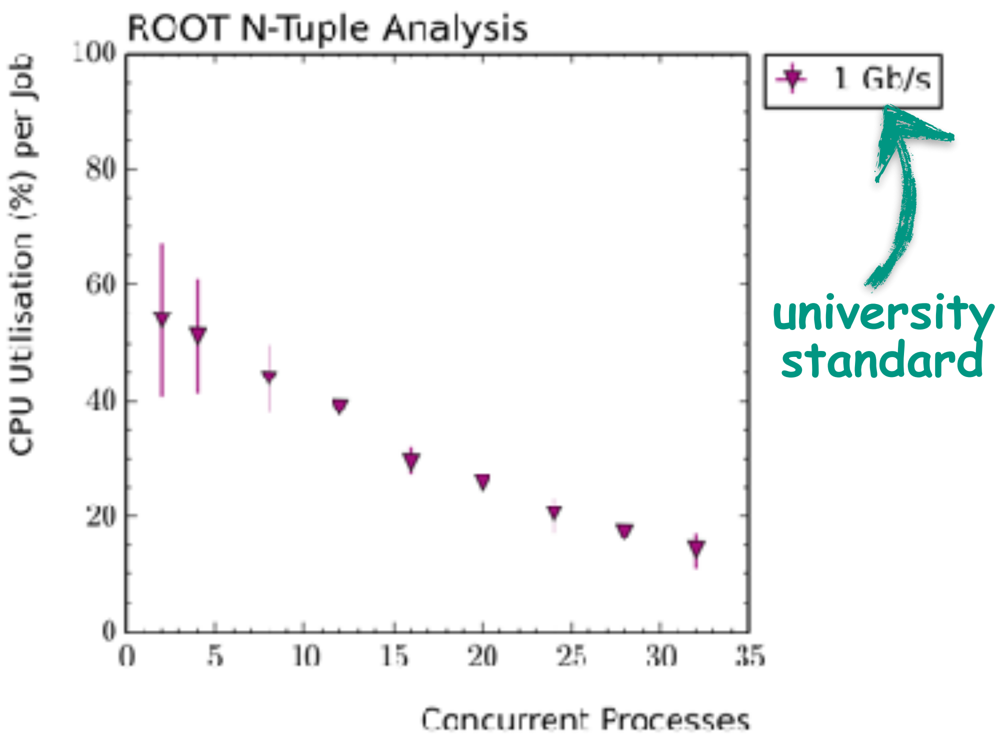




I/O Performance Evaluation

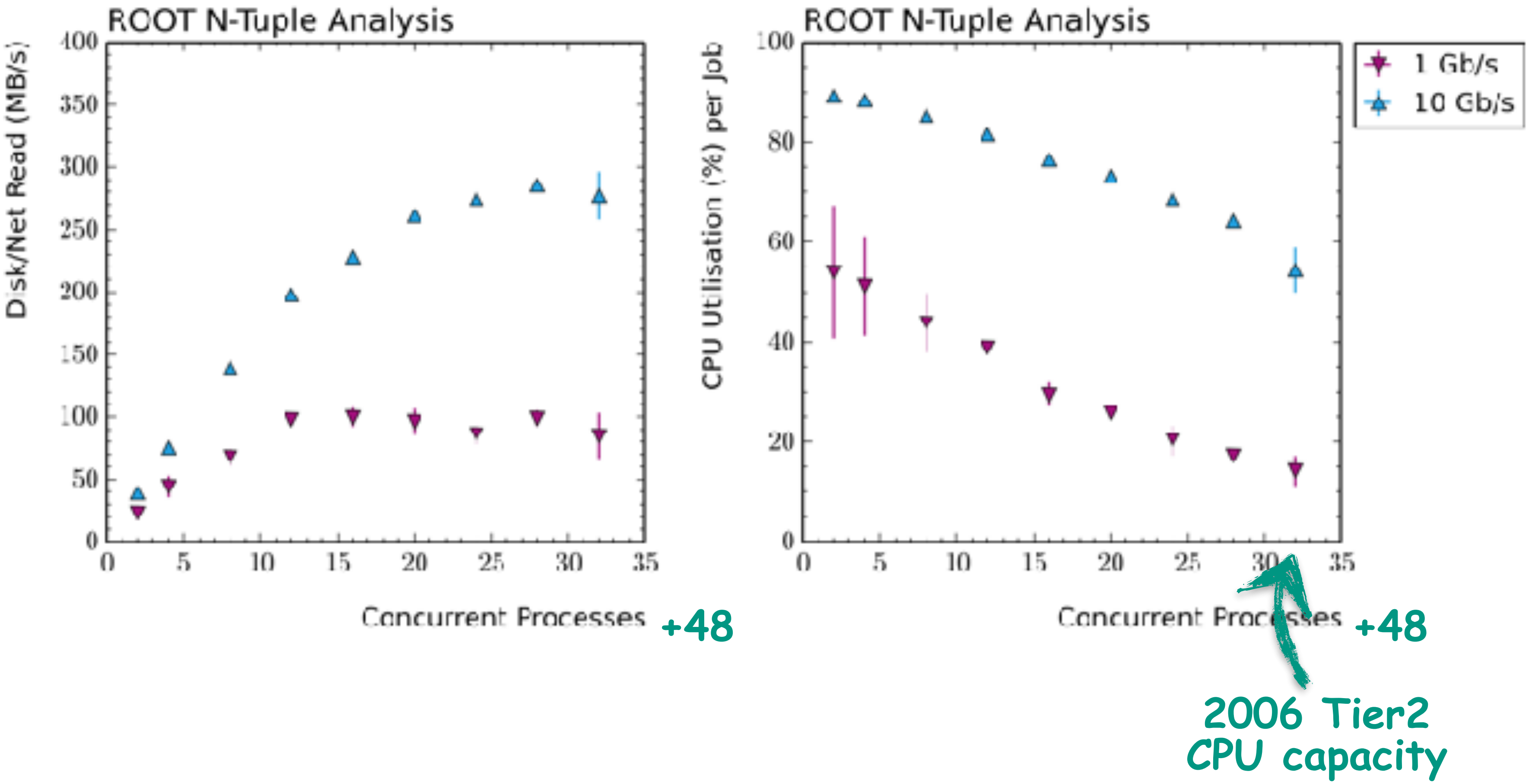
- CMS jet calibration analysis (ROOT n-tuple)





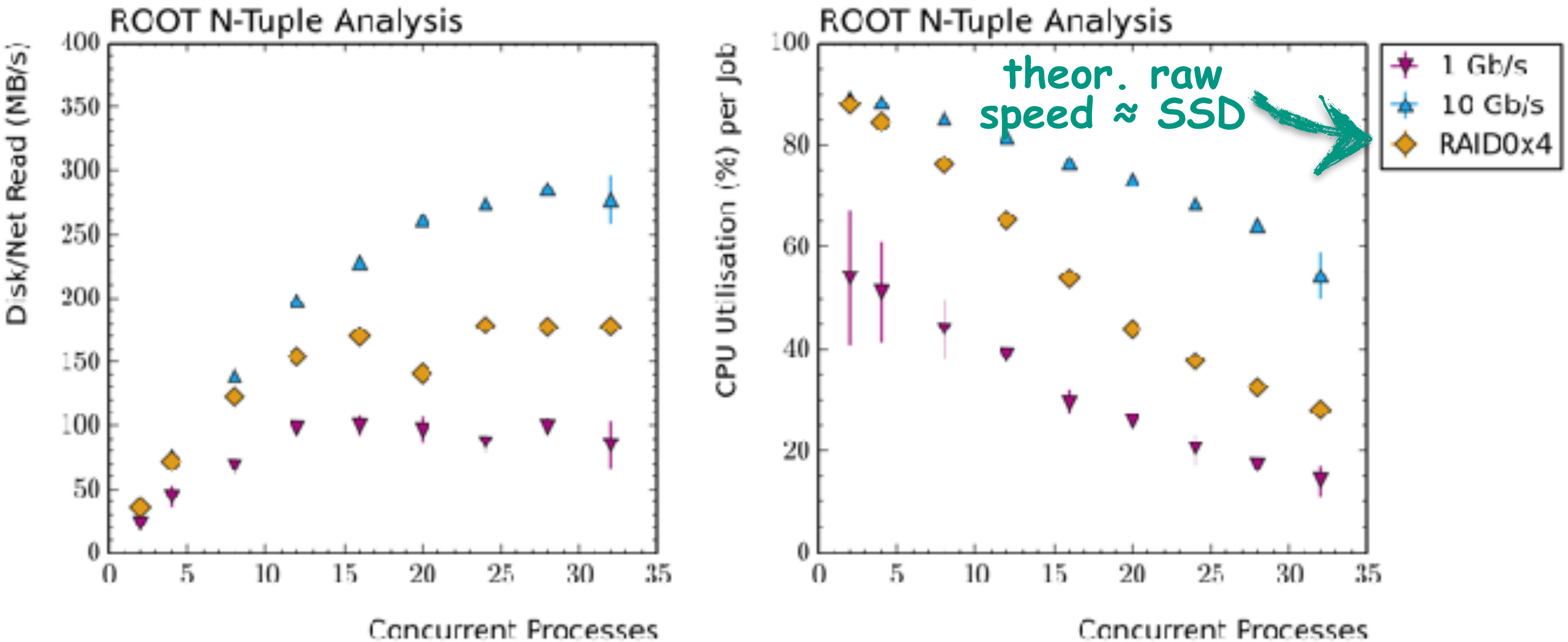
I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test



I/O Performance Evaluation

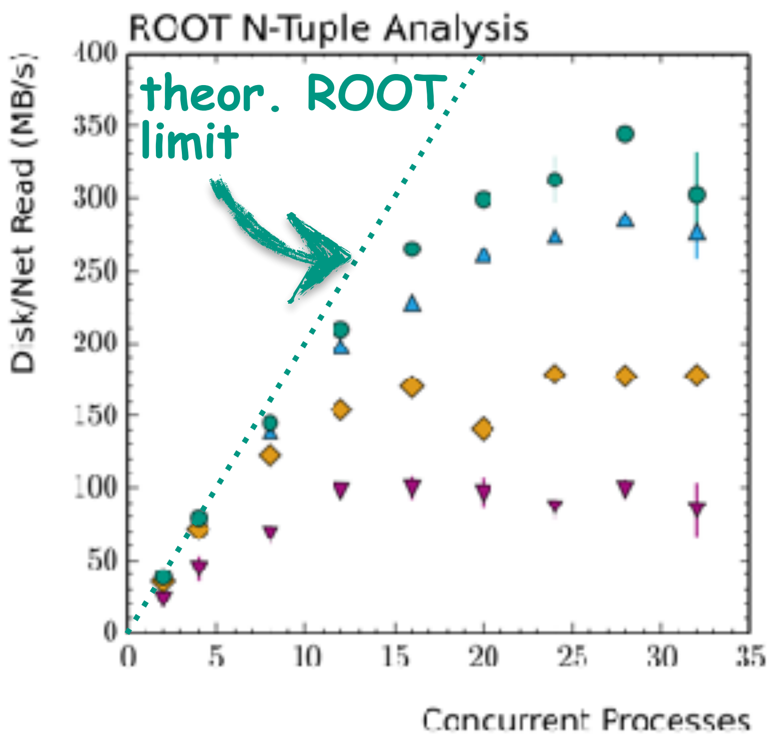
- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test

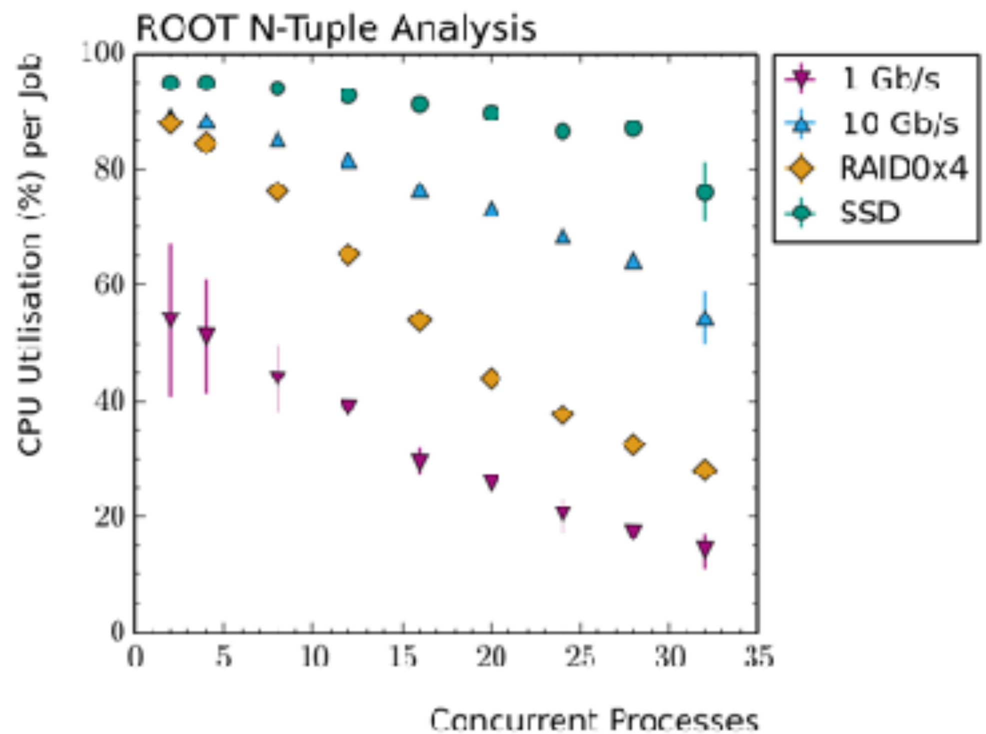


- HDDs limited on concurrent accesses

I/O Performance Evaluation

- CMS jet calibration analysis (ROOT n-tuple)
- Additional 48 concurrent reads from other workers for 10 Gb/s test

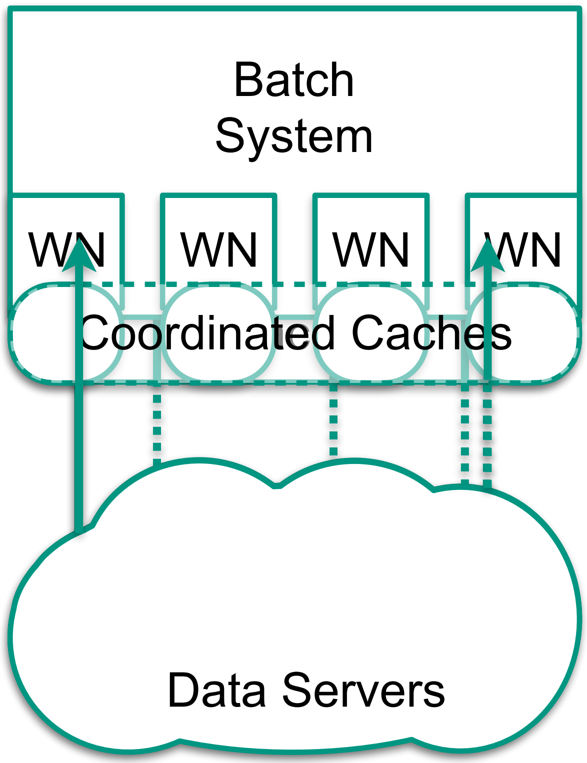




- HDDs limited on concurrent accesses
- SSDs exploit full system capacities

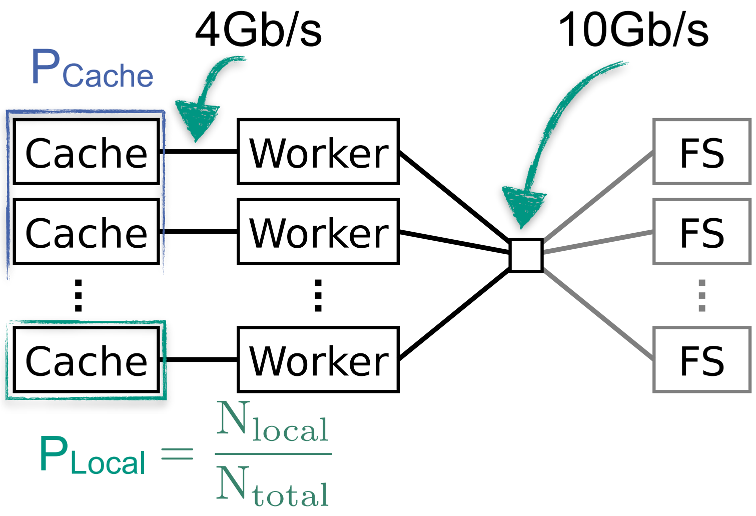
Coordinated Caching: Overview

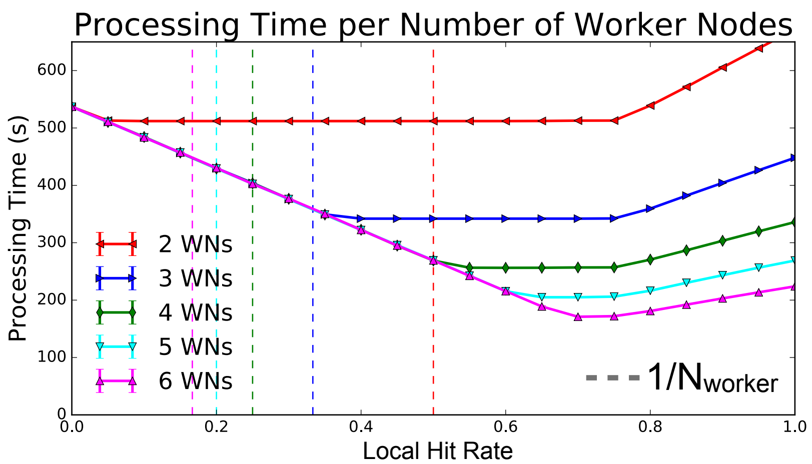
- Caching between batch system and data sources
  - Consumer focused caching
  - Partial data locality for remote files
- Abstracts cache to batch system scale
  - Utilize meta-data of entire user workflows
  - Works on files used by jobs
- Implementation at host granularity
  - Array of individual caches on worker nodes
  - Caches coordinated by global service
  - Some glue for data locality…



Coordinated Caching: Throughput

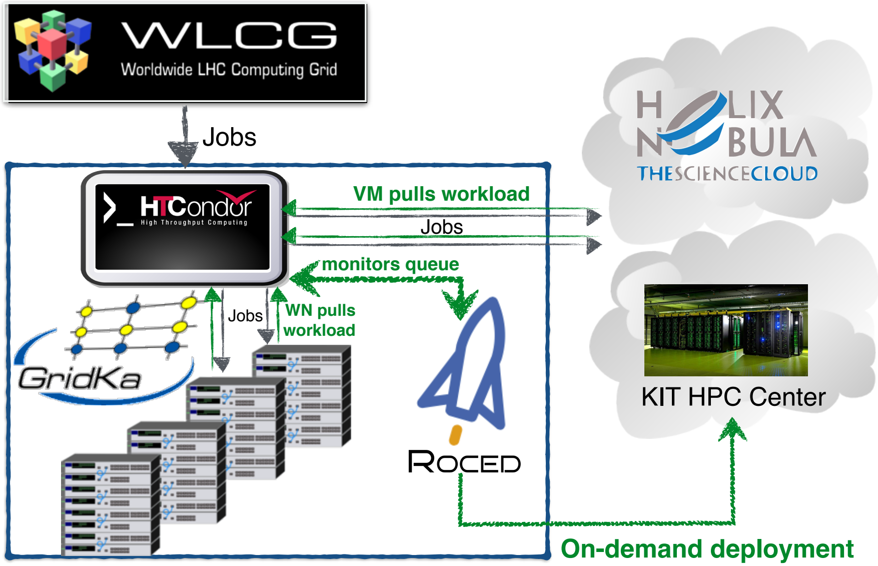
- Batch system throughput simulation
  - Setup of KIT Tier3
  - Parameters: local hit rate, N_{worker}
- Caching allows horizontal scaling
  - Throughput scales with workers…
  - …if jobs are scheduled to data
- Perfect hit rate not ideal
  - Balance remote and cache I/O
  - Potential to…
    - Use simple heuristics
    - Increase effective cache size





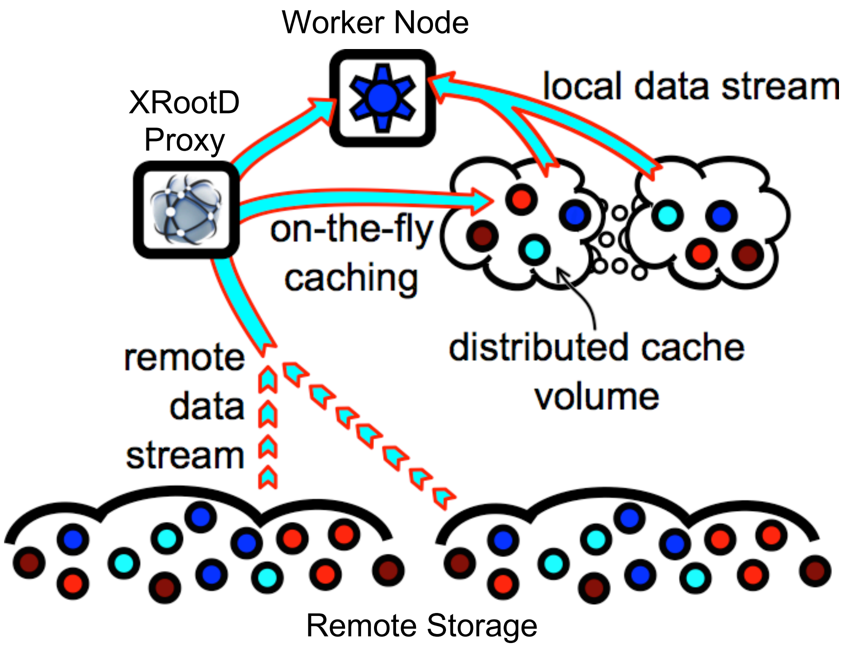
Dynamic Compute Expansion of GridKa Tier 1

- Transparent on-demand integration of opportunistic resources using ROCED
  - Helix Nebula Science Cloud (based on traditional virtualization)
  - KIT HPC Center (FORHLR II) (based on container technology)
- Automated detection and redirection of suitable CPU-intense workflows
- Evaluate ML for scheduling optimizations [JSSPP18MS]



Caching Concepts on Opportunistic Sites

- Opportunistic Resources usually well suited for CPU-intense workflows
- Many opportunistic sites offering fast cloud storage or distributed storage
- Benefit from caching R&D and bring recurrent I/O-intense workflows to the cloud
- Transparent data access also a hot topic in Helix Nebula Science Cloud



Collaboration in developing a xrootd based caching proxy between KIT and GSI

[ACAT17CH]

Resources at KIT

- small local institute cluster with HTCondor
- grid cluster and storage at GridKa
- opportunistic computing resources dynamic included via ROCED in HTCondor pool
  - HPC center at KIT
  - HPC center at Freiburg
  - HNSciCloud
    - OpenTelekomCloud

Scientific Cloud

HPC center at Freiburg

HPC center at KIT

Grid storage at GridKa

Commercial Cloud

ETP