Karlsruher Institut für Technologie

GridKa

Introduction into COBaldD
Workshop Intro

ErUM Data Cloud Workshop

Max Fischer, Manuel Giffels, Eileen Kühn, Matthias Schnepf

Steinbuch Centre for Computing / Institute for Experimental Particle Physics

ErUM Data Cloud Workshop

- Goal: Information exchange about COBalD/TARDIS
  - What is it? What is it good for? Why?
  - What it can/cannot do for you
  - What you would like to do with it
- We're a small group today - discussion is open!

10:30 - 11:30	***Overview COBalD/TARDIS***	
11:30 - 13:00	***Lunch***	
13:00 - 14:30	***Use-cases, feature requests and your expectations***	
14:30 - 16:00	***Tutorials: Hands-on COBalD/TARDIS***	

R&D Environment and Background

- Two collaborating HEP Computing Groups at KIT
  - SCC: GridKa Tier 1, focus on throughput and production systems
  - ETP: Institute Tier 3, focus on responsiveness and prototypes

R&D Environment and Background

- Two collaborating HEP Computing Groups at KIT
  - SCC: GridKa Tier 1, focus on throughput and production systems
  - ETP: Institute Tier 3, focus on responsiveness and prototypes
- Three major topics of Research and Development
  - Adaptive placement of input data via Caches
    M. Fischer et al., Opportunistic Data Locality for End User Data Analysis, Journal of Physics **898**, 5 (2017)
  - On-demand processing resources via VMs/container („ROCED“)
    T. Hauth et al., On-demand provisioning of HEP Compute Resources on Clouds Sites and Shared HPC Centers, Journal of Physics **898**, 5 (2017)
  - Runtime classification of payloads in overlay batch systems
    E. Kühn et al., A scalable architecture for online anomaly detection of WLCG batch jobs, Journal of Physics **762**, 1 (2016)

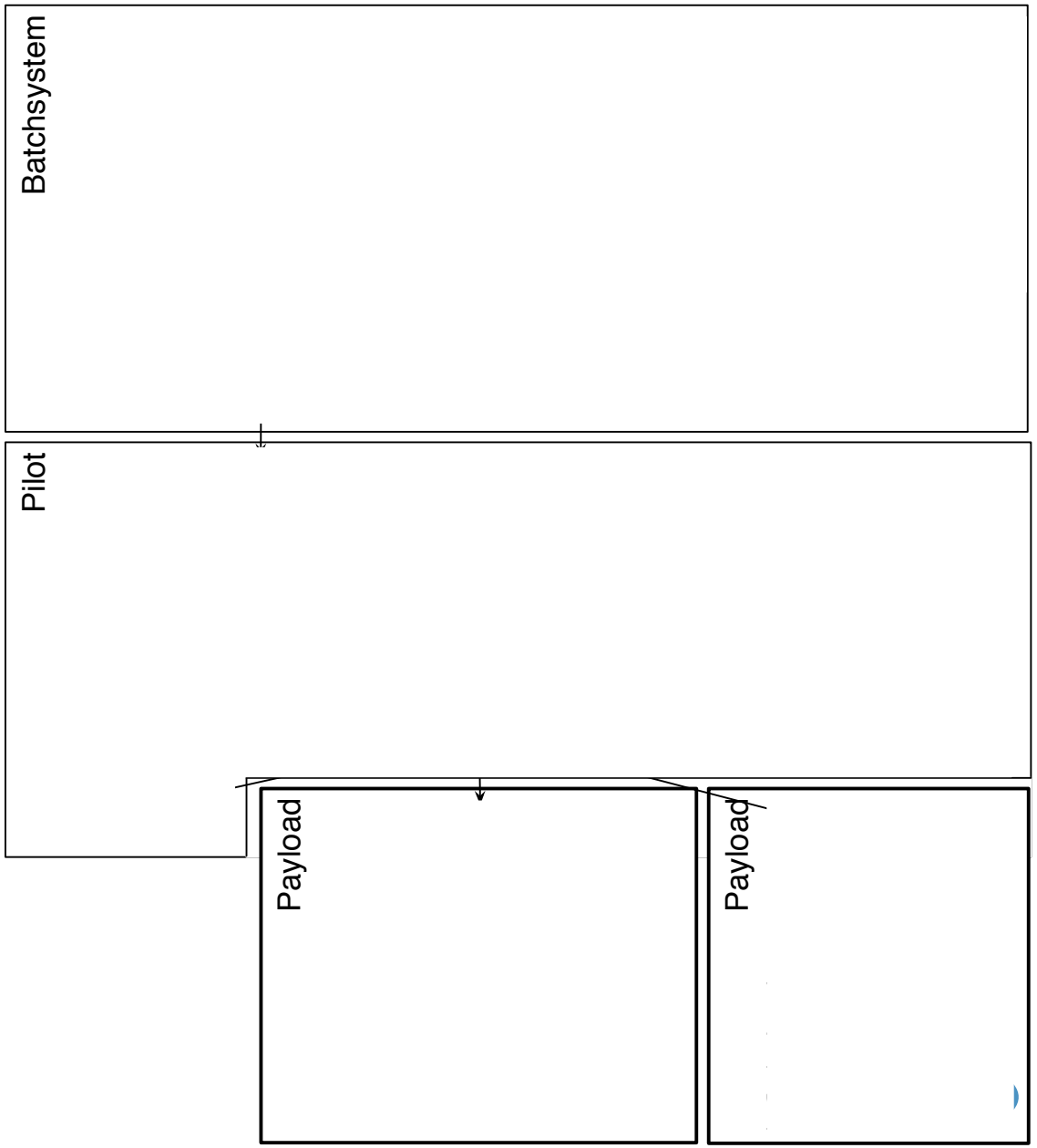
R&D Environment and Background

- Two collaborating HEP Computing Groups at KIT
  - SCC: GridKa Tier 1, focus on throughput and production systems
  - ETP: Institute Tier 3, focus on responsiveness and prototypes
- Three major topics of Research and Development
  - Adaptive placement of input data via Caches
    M. Fischer et al., Opportunistic Data Locality for End User Data Analysis, Journal of Physics **898**, 5 (2017)
  - On-demand processing resources via VMs/container („ROCED“)
    T. Hauth et al., On-demand provisioning of HEP Compute Resources on Clouds Sites and Shared HPC Centers, Journal of Physics **898**, 5 (2017)
  - Runtime classification of payloads in overlay batch systems
    E. Kühn et al., A scalable architecture for online anomaly detection of WLCG batch jobs, Journal of Physics **762**, 1 (2016)
- GridKa: Shift from supporting tools to service automation
  - Previously involved in benchmarking and monitoring (M. Alef)
  - Need automation at our scale, establish services while we’re at it…

Pilots: Classification

[Finalised/PhD E. Kühn]

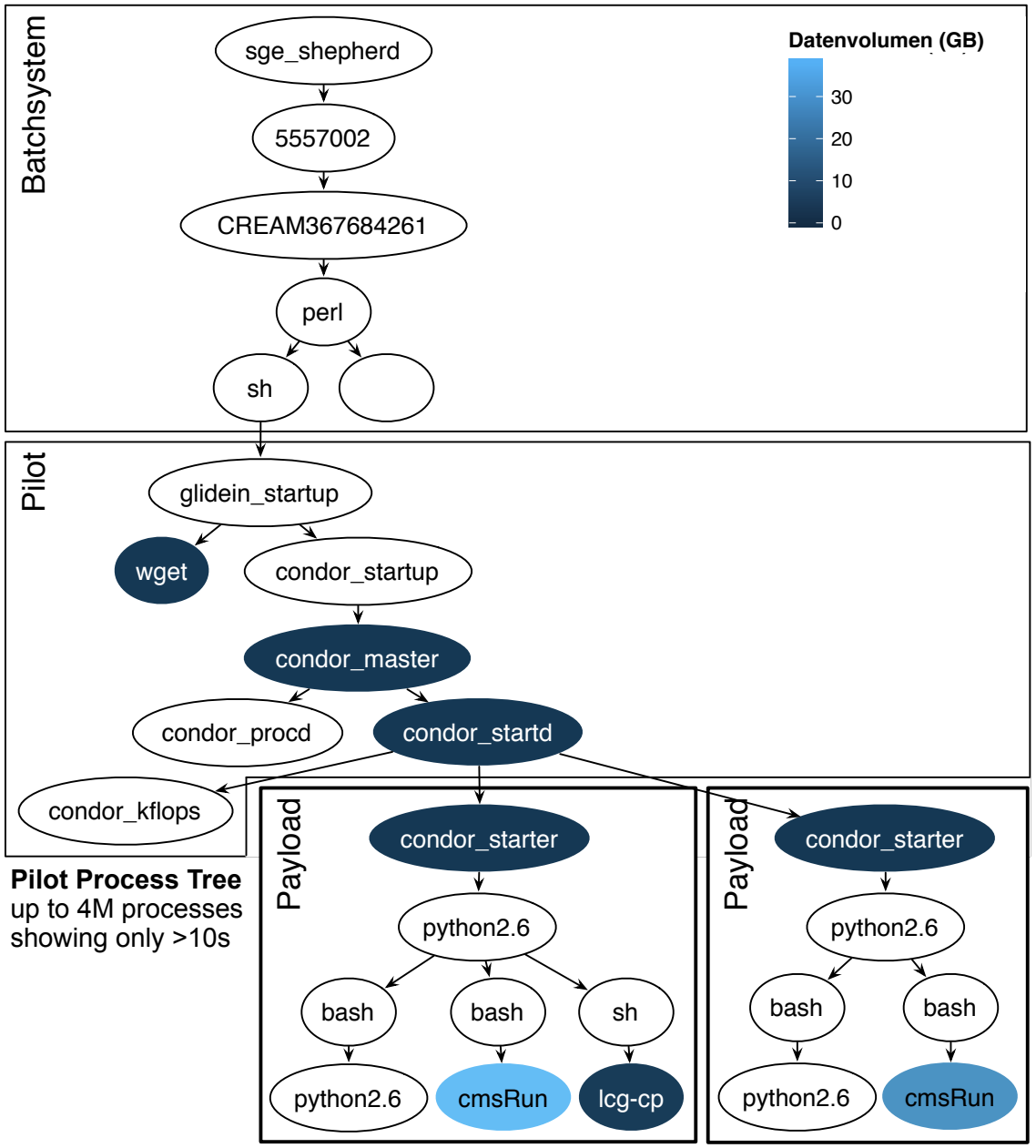
- Schedule Pilots not Jobs
  - Only know estimate for wrapping Pilot
  - Jobs are late-bound as Payloads



Pilots: Classification

[Finalised/PhD E. Kühn]

- Schedule Pilots not Jobs
  - Only know estimate for wrapping Pilot
  - Jobs are late-bound as Payloads
- Runtime Clustering
  - Cluster processes as dynamic tree stream
  - Exceeds knowledge of owning VOs

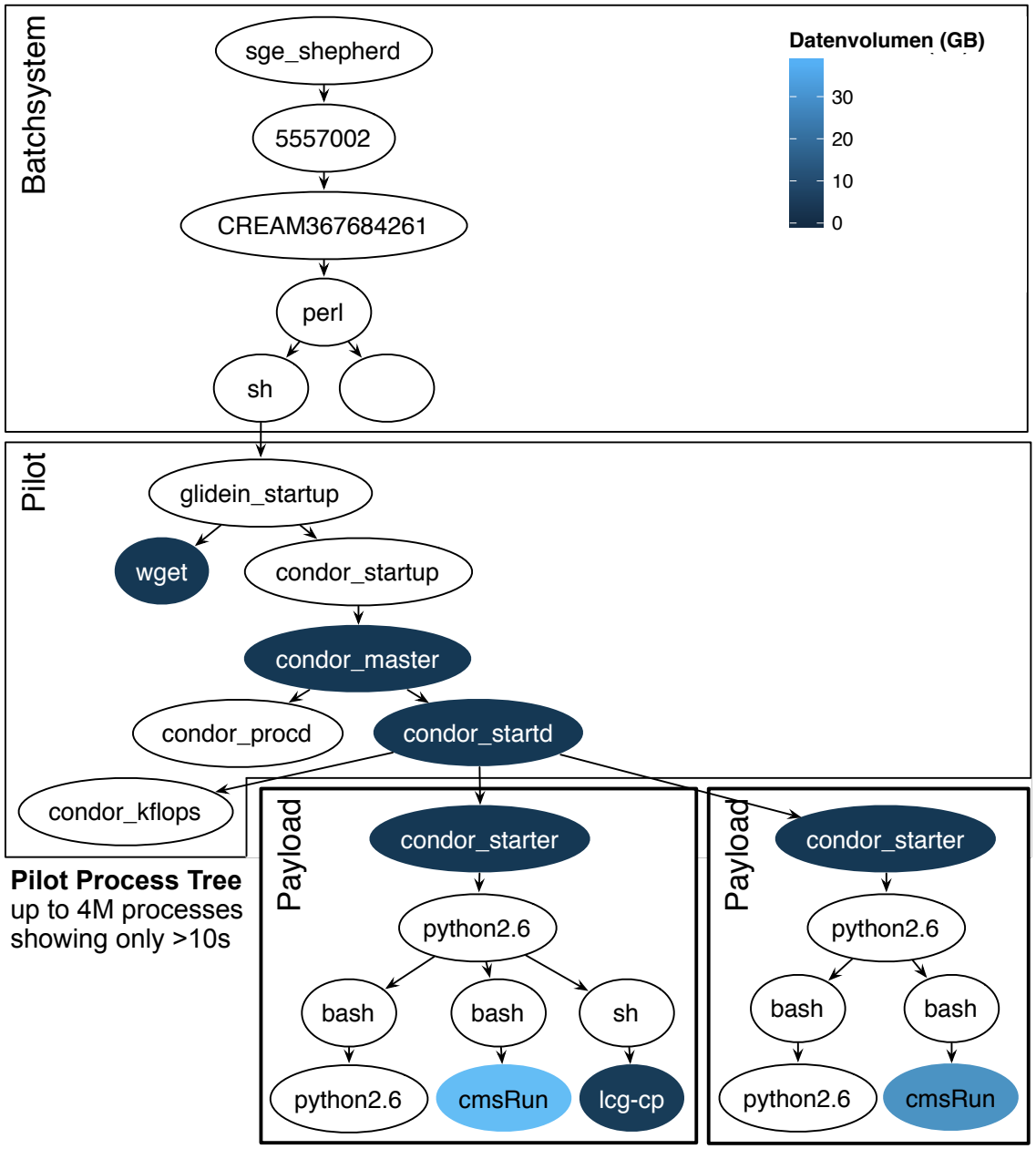


Pilot Process Tree
up to 4M processes
showing only >10s

Pilots: Classification

[Finalised/PhD E. Kühn]

- Schedule Pilots not Jobs
  - Only know estimate for wrapping Pilot
  - Jobs are late-bound as Payloads
- Runtime Clustering
  - Cluster processes as dynamic tree stream
  - Exceeds knowledge of owning VOs
- Practical information limit
  - Apply observations to successive pilots
  - Only able to observe current, partial state

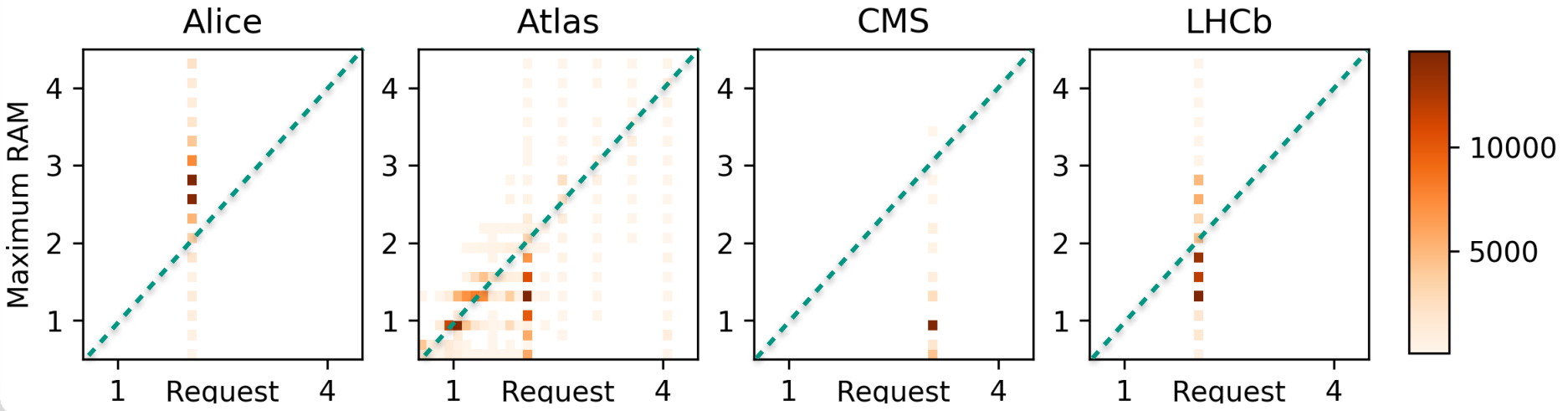


Pilot Process Tree
up to 4M processes
showing only >10s

Pilots: Requirement Estimations

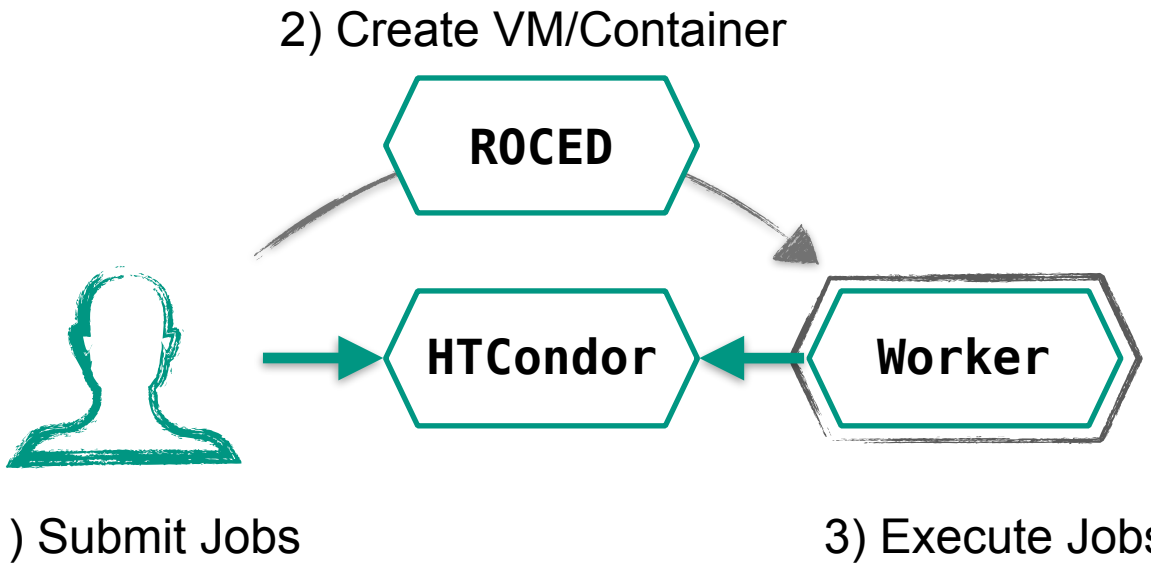
[Proof of Concept/MSc S. Lange]

- Attempt to predict precise pilot requirements for scheduling
  - Transfer monitored features from finished to queued pilots
  - Investigate possible gains from heuristic approaches
- Viable to use statistics and machine learning for short-term prediction
  - Improve average resource request accuracy from 75% to 90%
  - Only reliable up to 10 min before pilot is scheduled!



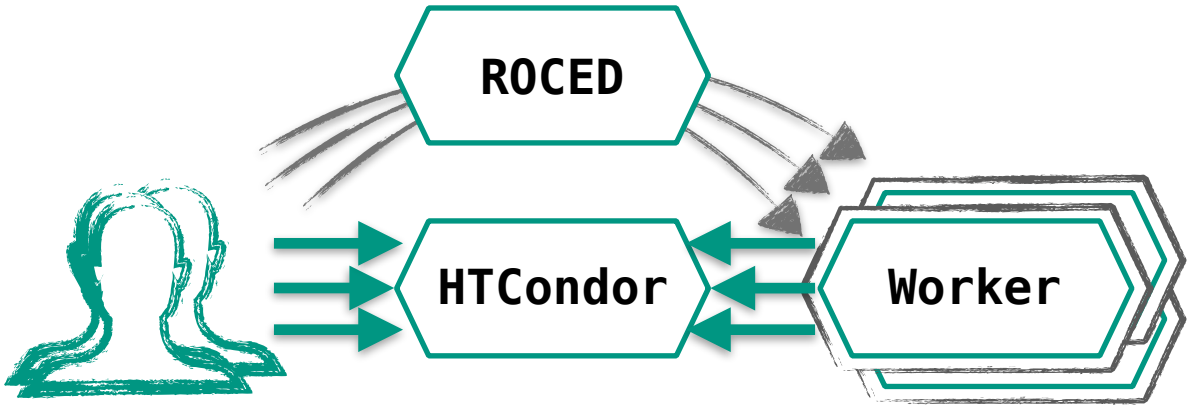
Opportunistic Resources: ROCED (2010-present)

[**R**esponsive **O**n-**D**emand **C**loud **E**nabled **D**eployment]



Opportunistic Resources: ROCED (2010-present)

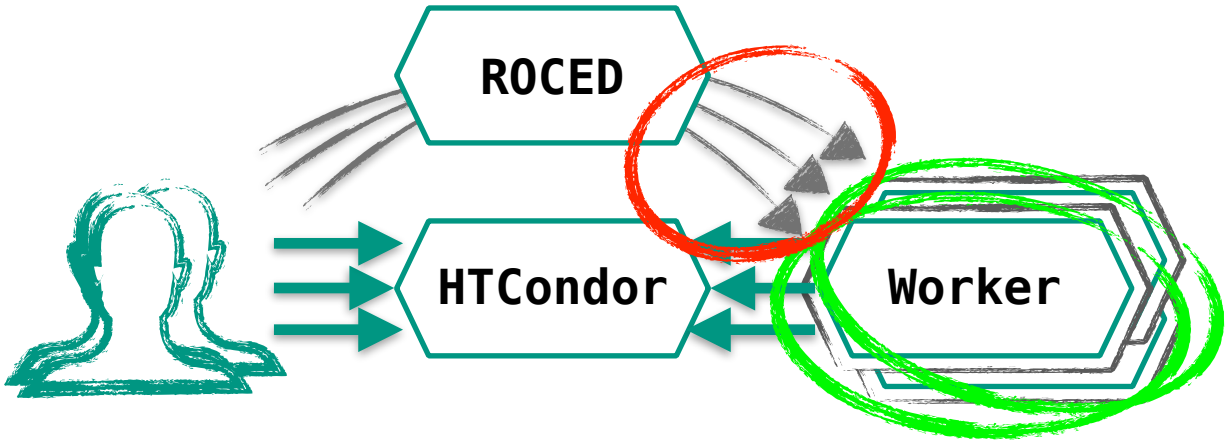
[**R**esponsive **O**n-**D**emand **C**loud **E**nabled **D**eployment]



Opportunistic Resources: ROCED (2010-present)

[**R**esponsive **O**n-**D**emand **C**loud **E**nabled **D**eployment]

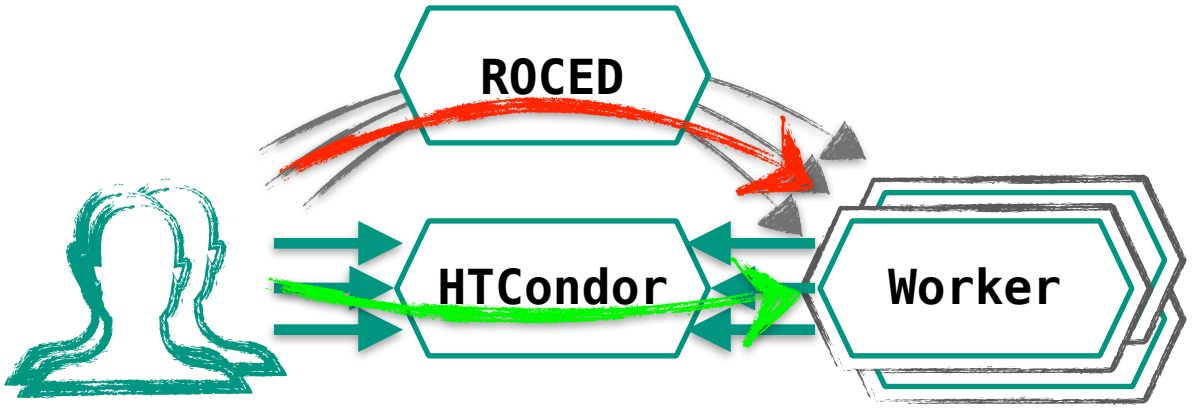
- Dynamic resources matching user demand
  - Trivial to support new providers for many users
  - Difficult to manage several providers for many users



Opportunistic Resources: ROCED (2010-present)

[**R**esponsive **O**n-**D**emand **C**loud **E**nabled **D**eployment]

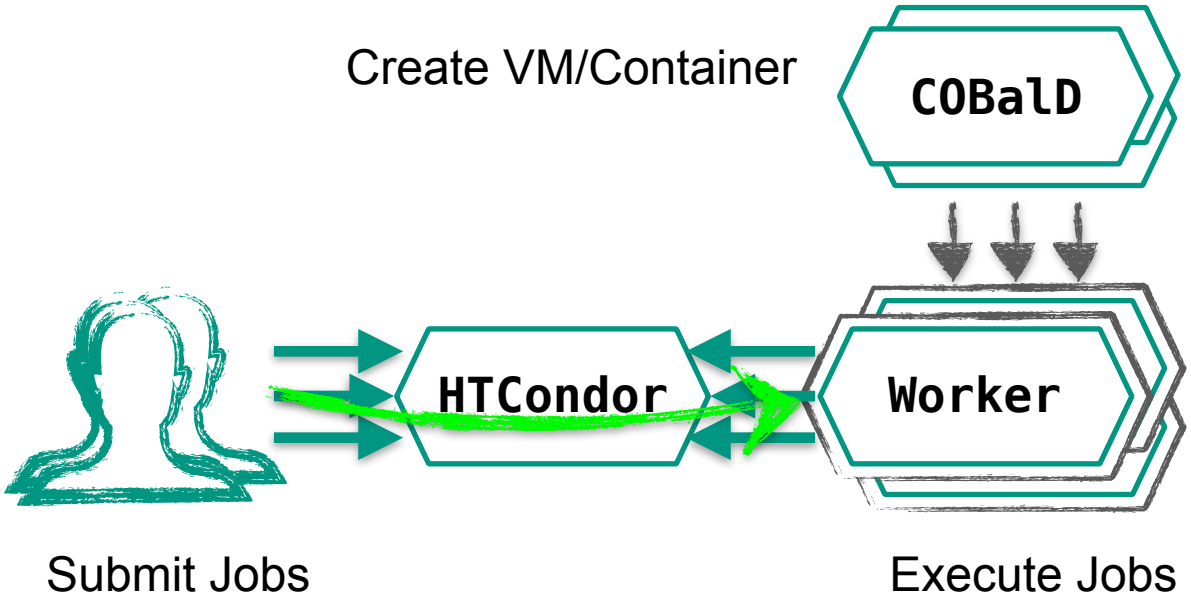
- Dynamic resources matching user demand
  - Trivial to support new providers for many users
  - Difficult to manage several providers for many users
- Resource aggregation in overlay batch system
  - Unreliable to predict resources required for jobs
  - Efficient to integrate resources, then match jobs



Opportunistic Resources: ROCED (2010-present)

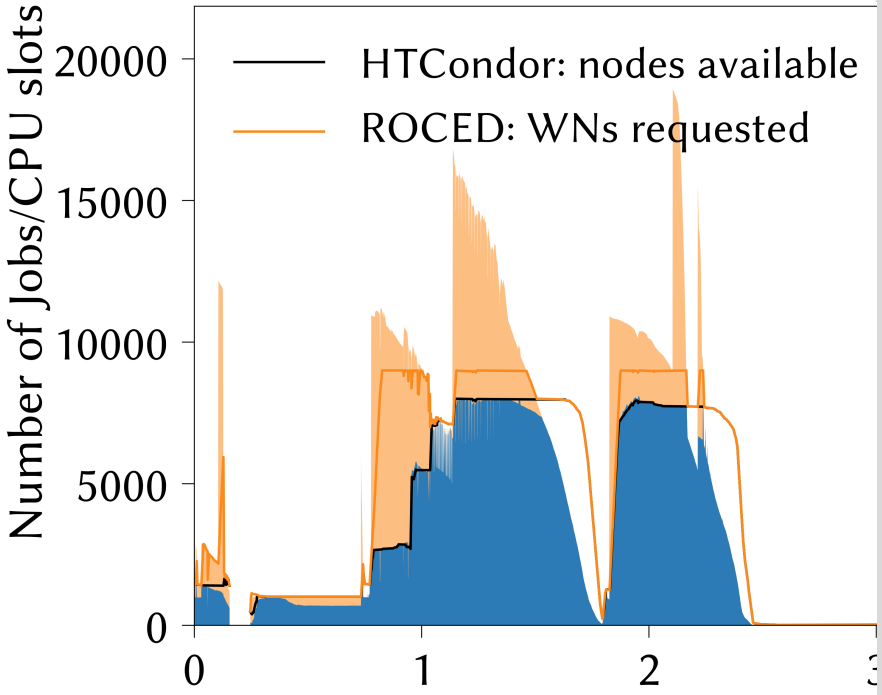
[**R**esponsive **O**n-**D**emand **C**loud **E**nabled **D**eployment]

- 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



Simpler approach: COBalD (2018-present)

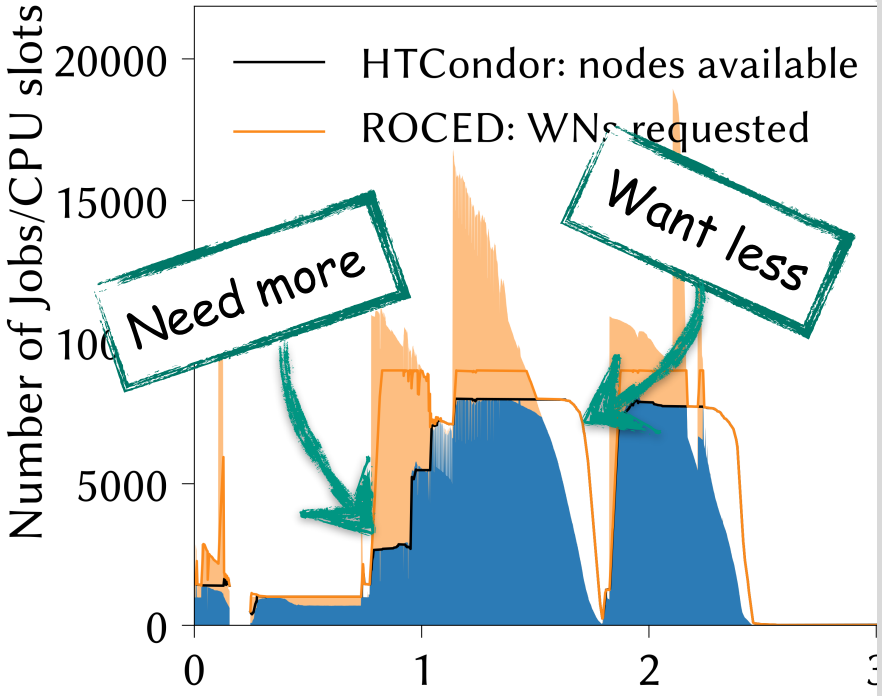
[**C**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]



Simpler approach: COBalD (2018-present)

[**C**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]

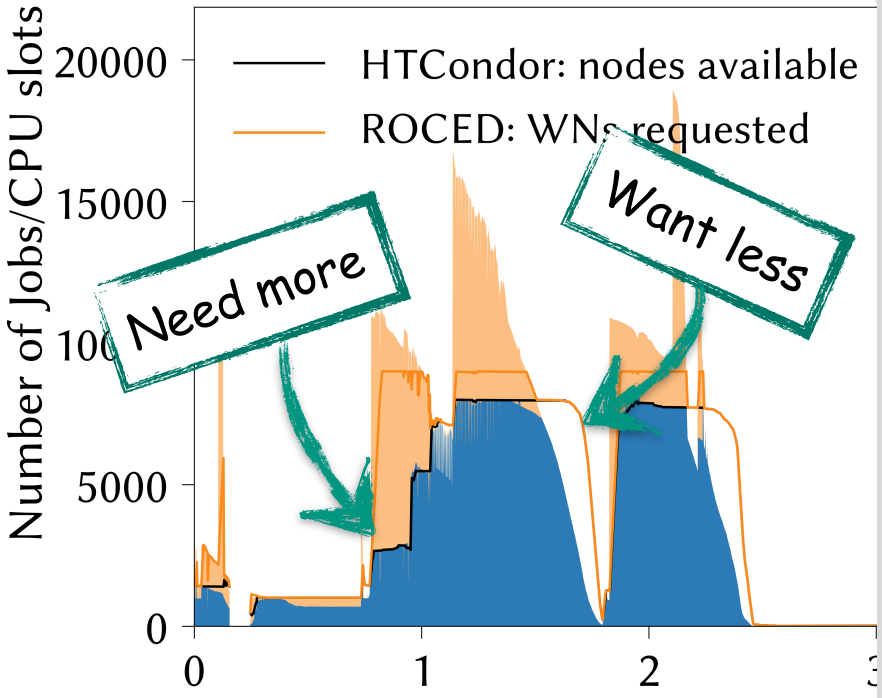
- Look at what is used, not what is requested
  - Simple logic: more used resources, less unused resources
  - Regular batch system scheduler decides what to use
  - COBalD only creates/destroys resources



Simpler approach: COBalD (2018-present)

[**C**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]

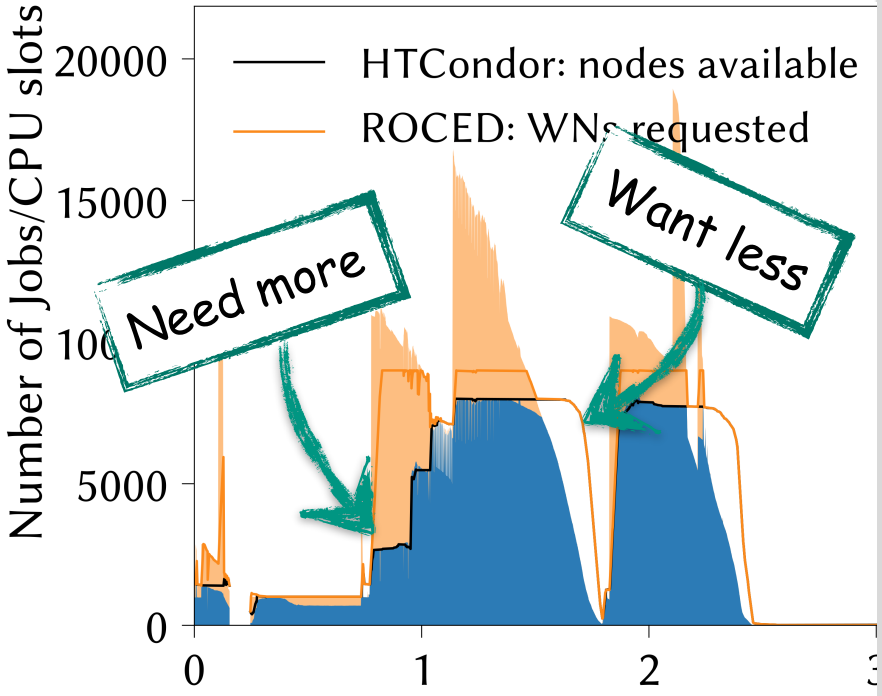
- Look at what is used, not what is requested
  - Simple logic: more used resources, less unused resources
  - Regular batch system scheduler decides what to use
  - COBalD only creates/destroys resources
- Generic design for any resources
  - COBalD just knows (un-)used
  - CPU, CPU+RAM, CPU+DISK, CPU+DISK+RAM, …
  - Partitioned multi-core slots



Simpler approach: COBalD (2018-present)

[**C**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]

- Look at what is used, not what is requested
  - Simple logic: more used resources, less unused resources
  - Regular batch system scheduler decides what to use
  - COBalD only creates/destroys resources
- Generic design for any resources
  - COBalD just knows (un-)used
  - CPU, CPU+RAM, CPU+DISK, CPU+DISK+RAM, …
  - Partitioned multi-core slots
- Native composition / concurrency
  - Aggregate same resources to entire pools managed as one
  - Several COBalDs manage pools for same overlay batch system



Design Decisions/Motivations

- Leading principle: We don’t/can’t have/know the perfect solution
  - Unpredictable systems: Unknowns, Latencies, Complexity
  - Split responsibility: Resource Admins, Overlay Admins, Users
  - Diverse targets: Availability, Authentication, Technology, QoS, …

Design Decisions/Motivations

- Leading principle: We don’t/can’t have/know the perfect solution
  - Unpredictable systems: Unknowns, Latencies, Complexity
  - Split responsibility: Resource Admins, Overlay Admins, Users
  - Diverse targets: Availability, Authentication, Technology, QoS, …
- Mixture of software framework and configurable service
  - COBalD/TARDIS offers standard components with *some* knobs
  - Modular components - select/write your own as required!

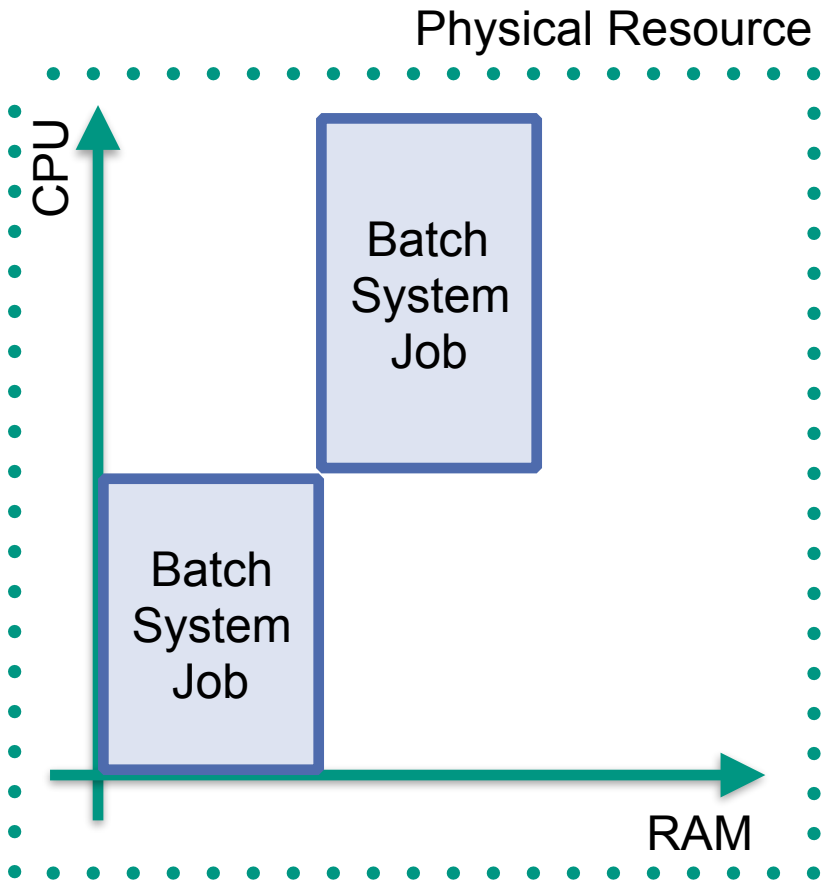
Design Decisions/Motivations

- Leading principle: We don’t/can’t have/know the perfect solution
  - Unpredictable systems: Unknowns, Latencies, Complexity
  - Split responsibility: Resource Admins, Overlay Admins, Users
  - Diverse targets: Availability, Authentication, Technology, QoS, …
- Mixture of software framework and configurable service
  - COBalD/TARDIS offers standard components with *some* knobs
  - Modular components - select/write your own as required!
- COBalD defines how to compose, not how to implement
  - Just know resources (Pool) and managing them (Controller)
  - Internal implementation unrestricted (boot VM, container, fork, …)

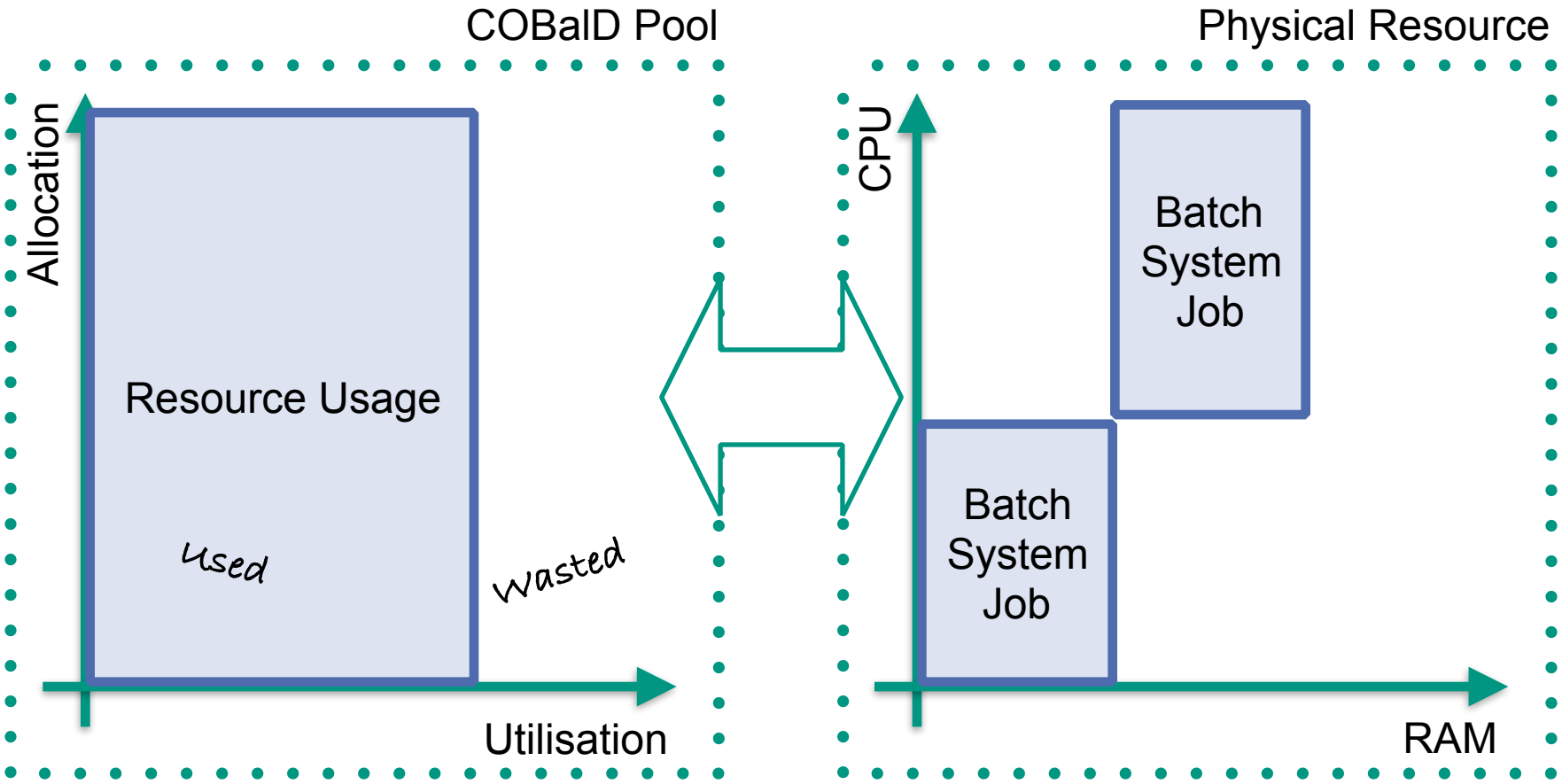
Design Decisions/Motivations

- Leading principle: We don’t/can’t have/know the perfect solution
  - Unpredictable systems: Unknowns, Latencies, Complexity
  - Split responsibility: Resource Admins, Overlay Admins, Users
  - Diverse targets: Availability, Authentication, Technology, QoS, …
- Mixture of software framework and configurable service
  - COBalD/TARDIS offers standard components with *some* knobs
  - Modular components - select/write your own as required!
- COBalD defines how to compose, not how to implement
  - Just know resources (Pool) and managing them (Controller)
  - Internal implementation unrestricted (boot VM, container, fork, …)
- Concrete resource definition/lifecycle managed by plugins

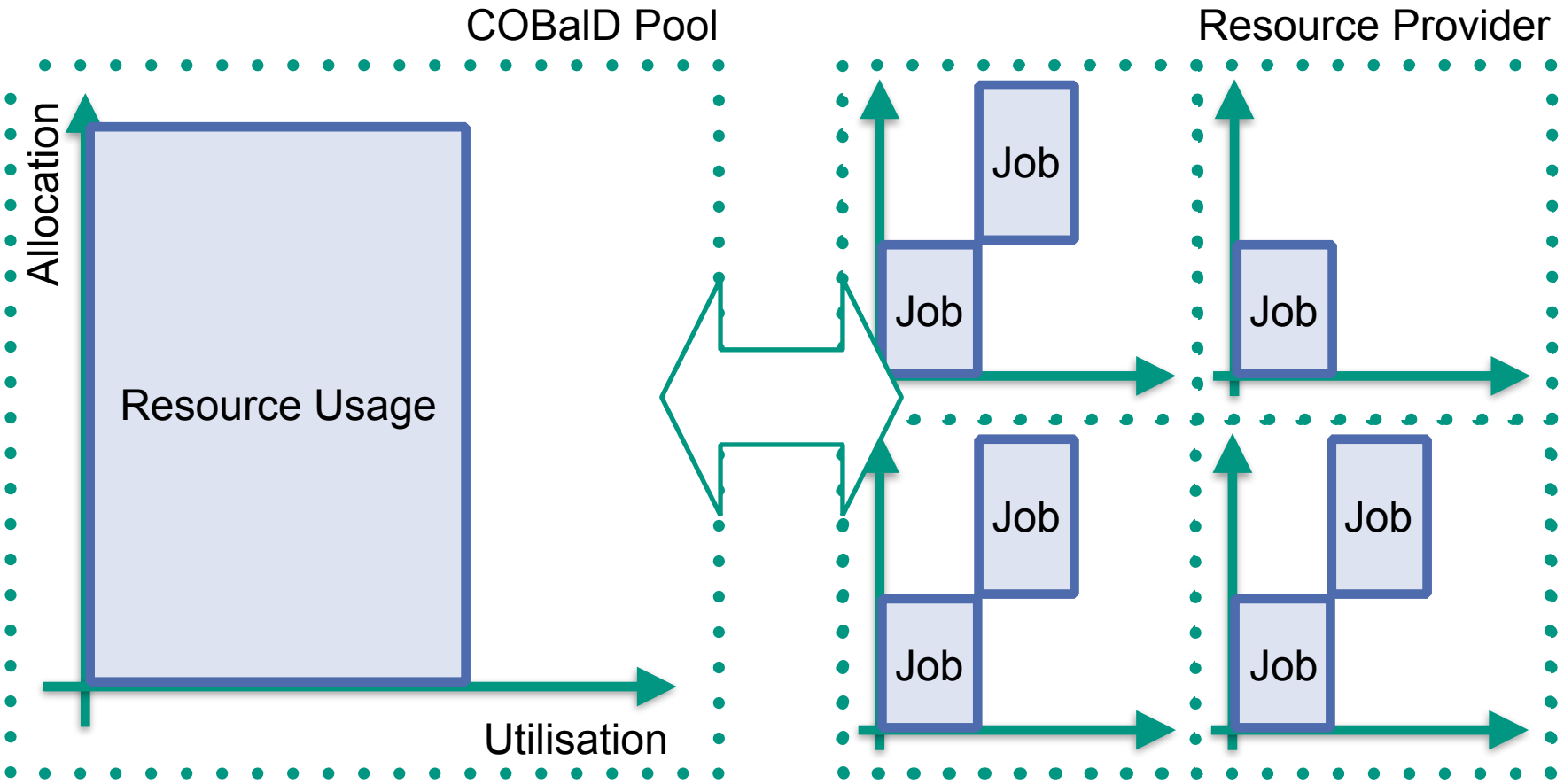
COBalD Resource Pool Model



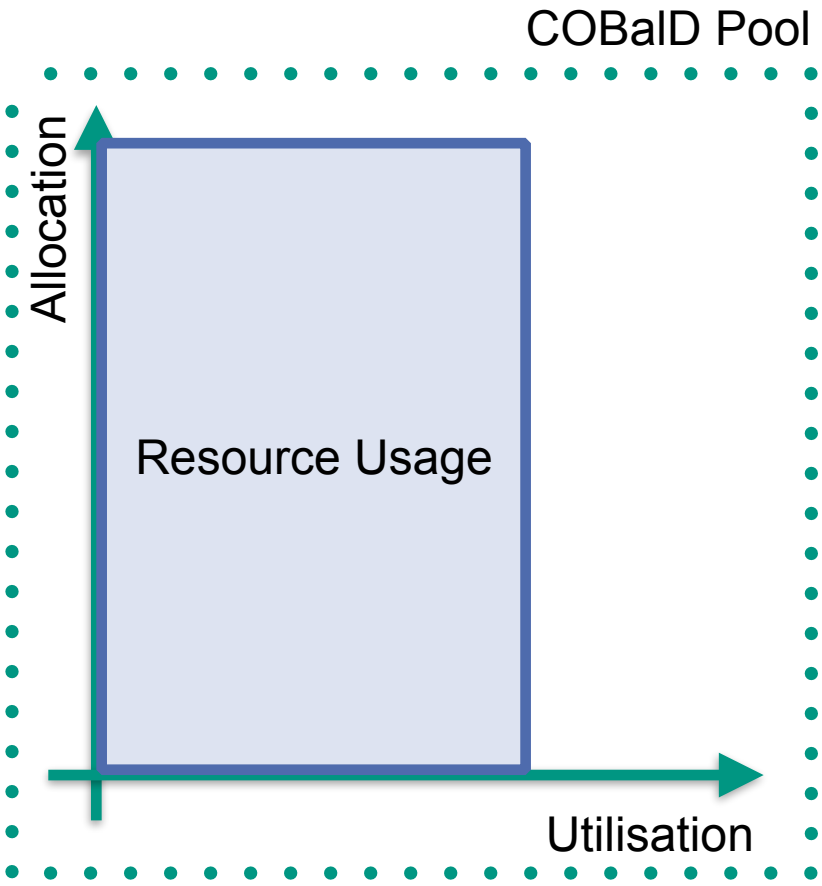
COBalD Resource Pool Model



COBalD Resource Pool Model

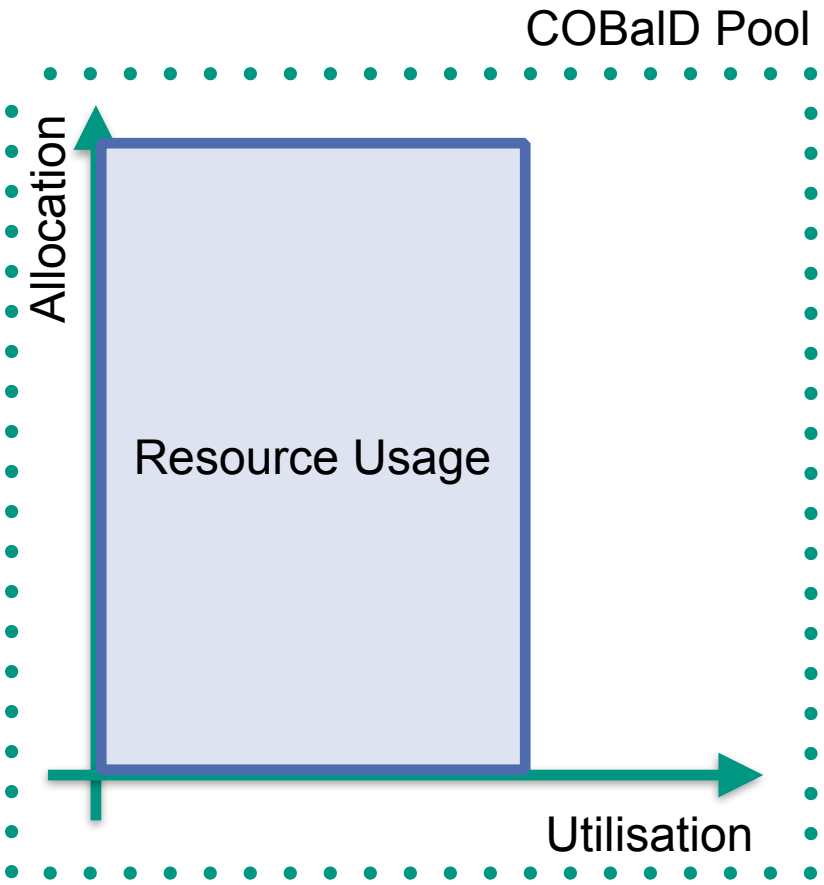


COBalD Resource Pool Model



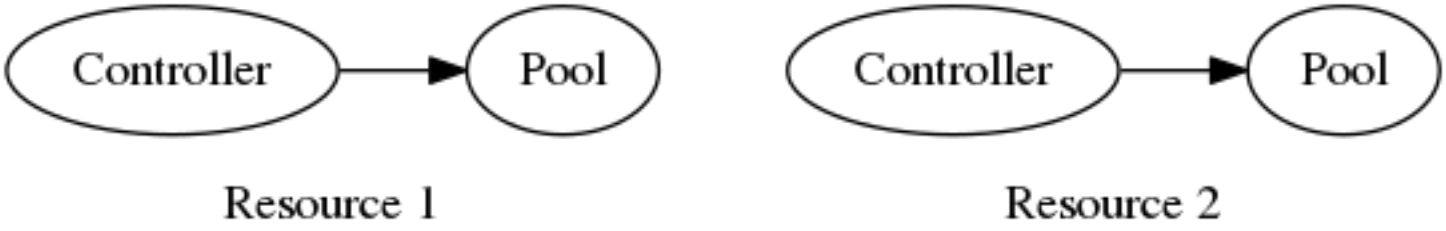
COBalD Resource Pool Model

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

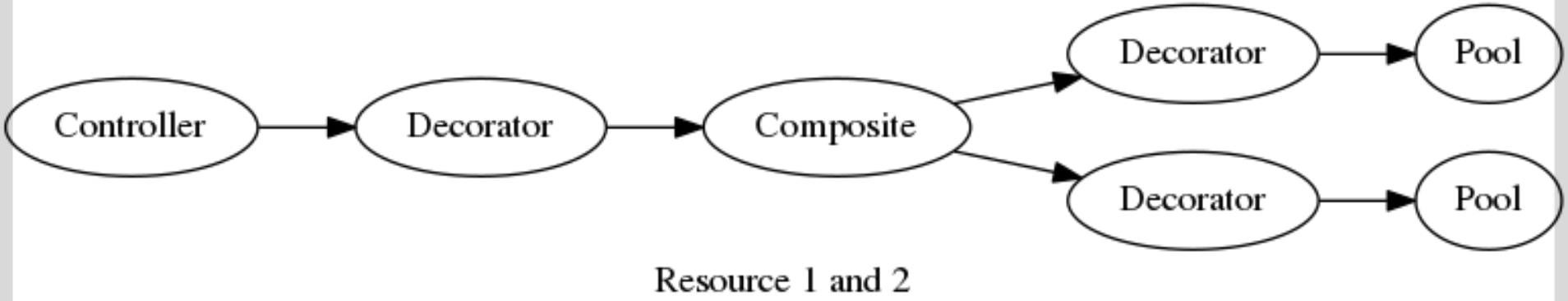


COBalD Resource Pool Model

COBalD Resource Pool Model

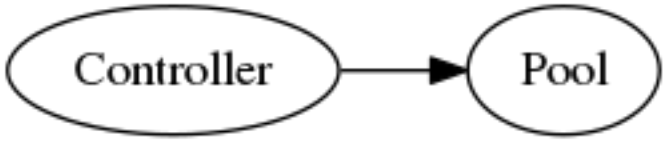


COBalD Resource Pool Model



Resource 1 and 2

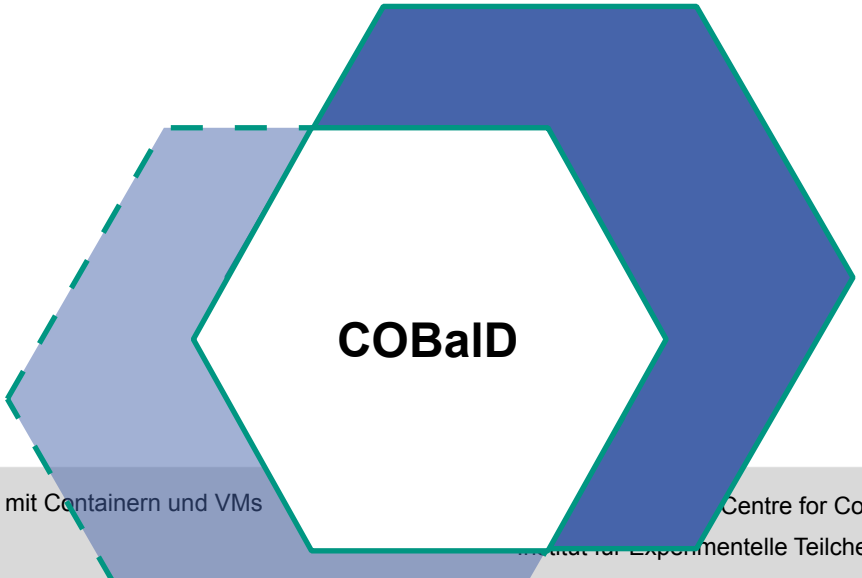
Resource 1



Resource 2

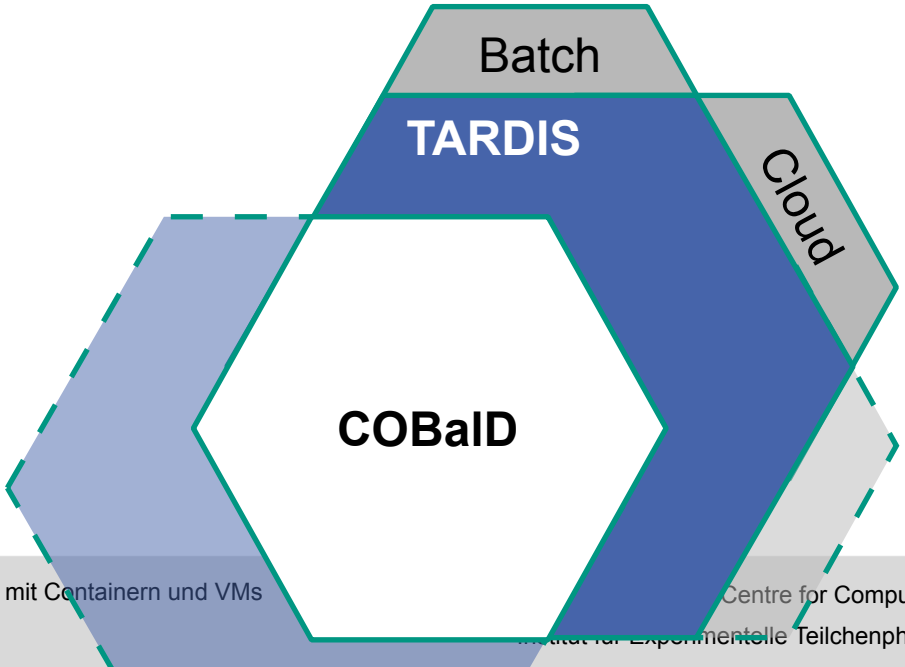
COBalD, TARDIS, …?

- COBalD: generic core service/framework
  - General purpose Controllers, Decorators, Composites, …
  - Service: daemon, monitoring, configuration, multitasking, …
  - Extensions: `cobald.remote`, …



COBalD, TARDIS, …?

- COBalD: generic core service/framework
  - General purpose Controllers, Decorators, Composites, …
  - Service: daemon, monitoring, configuration, multitasking, …
  - Extensions: `cobald.remote`, …
- TARDIS: opportunistic HEP resources
  - Purpose-built Pools that represent overlay batch systems
  - Acquisition (Cloud, Batch, …) and integration (VM, container, …)
  - Knobs, knobs, knobs…

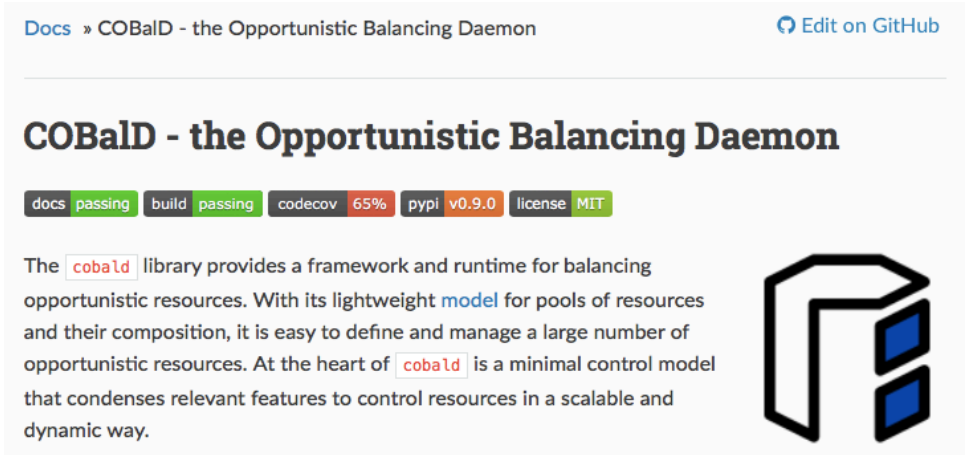


```python
from cobald.controller.stepwise import stepwise

from cobald_demo.cpu_pool import CpuPool
from cobald_demo.draw_line import DrawLineHook

# initial controller skeleton from base case
@stepwise
def control(pool: Pool, interval):
    return 10


# additional rules above specific supply thresholds
@control.add(supply=10)
def quantized(pool: Pool, interval):
    if pool.utilisation < 0.5:
        return pool.supply - 1
    elif pool.allocation > 0.5:
        return pool.supply + 1


@control.add(supply=100)
def continuous(pool: Pool, interval):
    if pool.utilisation < 0.5:
        return pool.supply * 1.1
    elif pool.allocation > 0.5:
        return pool.supply * 0.9


# create controller pipeline from skeleton
pipeline = control.s(interval=10) >> DrawLineHook.s() >> CpuPool()
```

```
@service(flavour=threading)
class CpuPool(Pool):
    """
    Dummy pool showing how demand is adjusted based on allocation/utilisation
    """
    #: we cannot really adjust the number of cores...
    demand = 0.0

    @property
    def supply(self):
        """The current supply this pool could provide"""
        #: TODO: make this drag behind instead of just copying demand
        return self.demand

    @property
    def allocation(self):
        """Fraction of the supply actively allocated for use"""
        # how much resources are used from our (faked) resource pool
        try:
            return self._cpu_percent / self.demand
        except ZeroDivisionError:
            return 2.0

    #: Fraction of the supply actively used
    utilisation = allocation

    def __init__(self, interval=0.1):
        self.interval = interval
        self.demand = 0.0
        self._cpu_percent = psutil.cpu_percent(0.1)

    # entry point for the ``service`` background process
    def run(self):
        while True:
            self._cpu_percent = psutil.cpu_percent(self.interval)
            time.sleep(0.0)
```

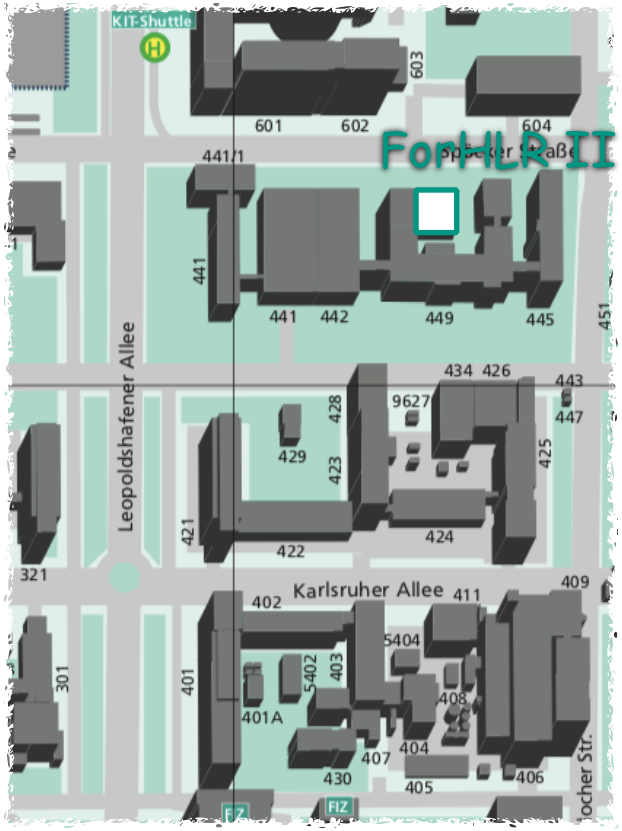
Conclusion

- Long History of using Opportunistic Resources for Physics
  - Easy to get them, hard to use them
  - We are now beyond being able to micromanage
- COBalD/TARDIS: lightweight approach to Opportunistic Resources
  - COBalD: Service core, resource composition, generic components
  - TARDIS: opportunistic HEP resources using Cloud, Batch, …



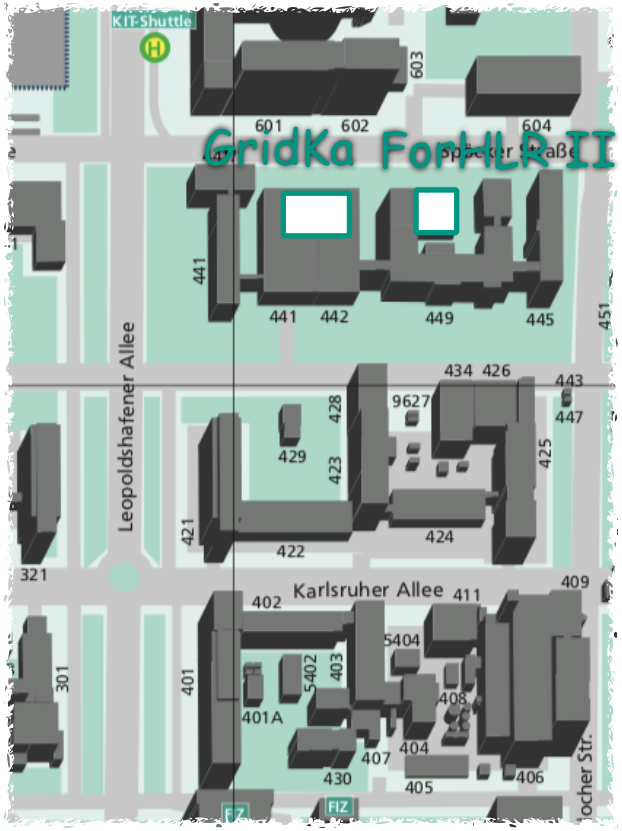
Kontext: Hochenergiephysik (HEP) Computing

- Globale Kollaborationen für Beschleunigerphysik
  - Experimente: LHC (CERN), Belle-2 (KEK)
  - Je 3k-8k Wissenschaftler aus ~150 Instituten



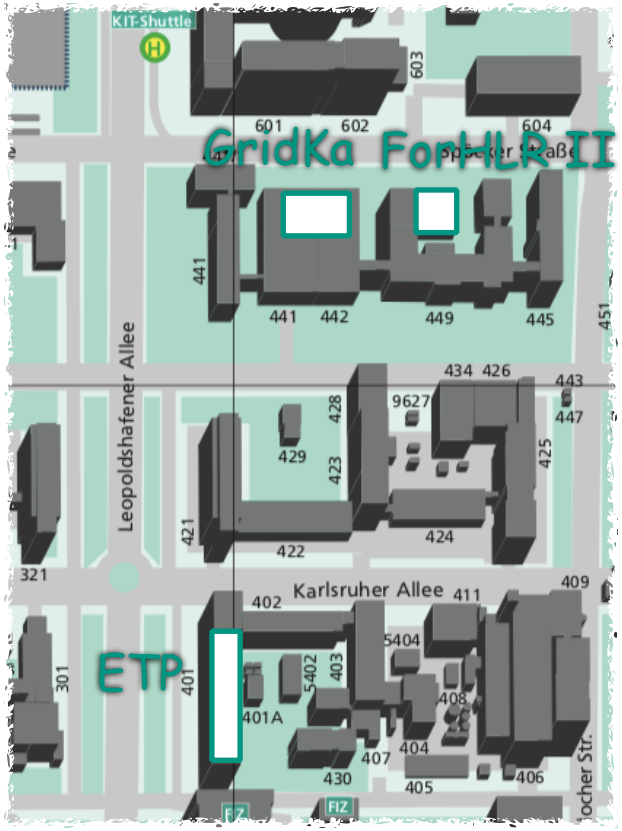
Kontext: Hochenergiephysik (HEP) Computing

- Globale Kollaborationen für Beschleunigerphysik
  - Experimente: LHC (CERN), Belle-2 (KEK)
  - Je 3k-8k Wissenschaftler aus ~150 Instituten
- Analysen nutzen weltweites Computing Grid
  - ~200 Rechenzentren + Institutscluster
  - Dedizierte Ressourcen und Support
  - GridKa Tier 1 (deutschsprachiger Raum)



Kontext: Hochenergiephysik (HEP) Computing

- Globale Kollaborationen für Beschleunigerphysik
  - Experimente: LHC (CERN), Belle-2 (KEK)
  - Je 3k-8k Wissenschaftler aus ~150 Instituten
- Analysen nutzen weltweites Computing Grid
  - ~200 Rechenzentren + Institutscluster
  - Dedizierte Ressourcen und Support
  - GridKa Tier 1 (deutschsprachiger Raum)
- Kollaborationen aus Analysegruppen
  - Koordinierte Simulation, Datenaufbereitung
  - Unabhängige Physikanalysen
  - KIT ETP Gruppe (CMS/Belle-2)



Paradigmenwechsel: außerhalb des WLCG

- Ziel: Nutzung externer, bestehender Ressourcen
  - Nutzung allgemeiner Wissenschaftsinfrastruktur
  - Technischer Transfer für neue Datenwissenschaften
  - Dynamischer Ausgleich kurzfristiger Lastspitzen

Paradigmenwechsel: außerhalb des WLCG

- Ziel: Nutzung externer, bestehender Ressourcen
  - Nutzung allgemeiner Wissenschaftsinfrastruktur
  - Technischer Transfer für neue Datenwissenschaften
  - Dynamischer Ausgleich kurzfristiger Lastspitzen
- Unterschiedliche Nutzungsarten
  - Statischer Anteil: HPC+Antrag
  - Opportunistisch: HPC/Cloud Backfilling
  - On-demand: Public Cloud

Paradigmenwechsel: außerhalb des WLCG

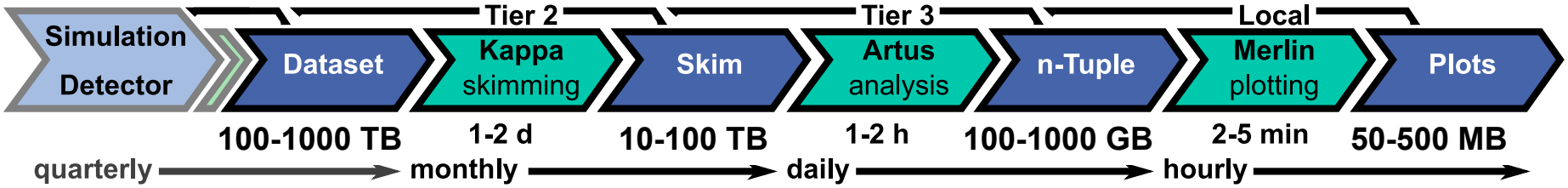
- Ziel: Nutzung externer, bestehender Ressourcen
  - Nutzung allgemeiner Wissenschaftsinfrastruktur
  - Technischer Transfer für neue Datenwissenschaften
  - Dynamischer Ausgleich kurzfristiger Lastspitzen
- Unterschiedliche Nutzungsarten
  - Statischer Anteil: HPC+Antrag
  - Opportunistisch: HPC/Cloud Backfilling
  - On-demand: Public Cloud
- Cloud-artige Erweiterung der GridKa/ETP Ressourcen
  - Statisch: Freiburg HPC NEMO
  - Opportunistisch: Karlsruhe ForHLR, Cycle-Stealing
  - On-demand: 1und1 Cloud, TSystems Cloud, Amazon EC2, …

HEP Nutzer, Workflows und Jobs

- Trivial parallele HTC Workflows pro Datensatz
  - Pro Workflow 1.000-10.000 unabhängige Jobs
  - Datenrate 10 KB/s/Core bis 20 MB/s/Core
  - Dynamische Laufzeit 15 min - 72 h

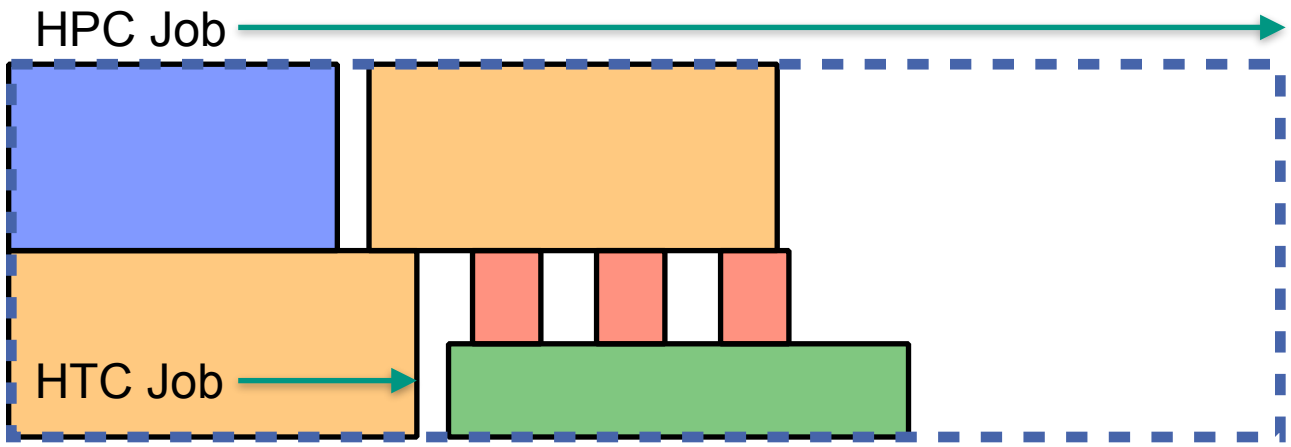
HEP Nutzer, Workflows und Jobs

- Trivial parallele HTC Workflows pro Datensatz
  - Pro Workflow 1.000-10.000 unabhängige Jobs
  - Datenrate 10 KB/s/Core bis 20 MB/s/Core
  - Dynamische Laufzeit 15 min - 72 h
- Angepasst auf Grid-Umgebung
  - Tiers mit globalem, geteiltem Speicher
  - Grid-APIs, privilegierte Applikationen
  - Anforderung: 1 | 8 Core, 2 - 4 GB/Core



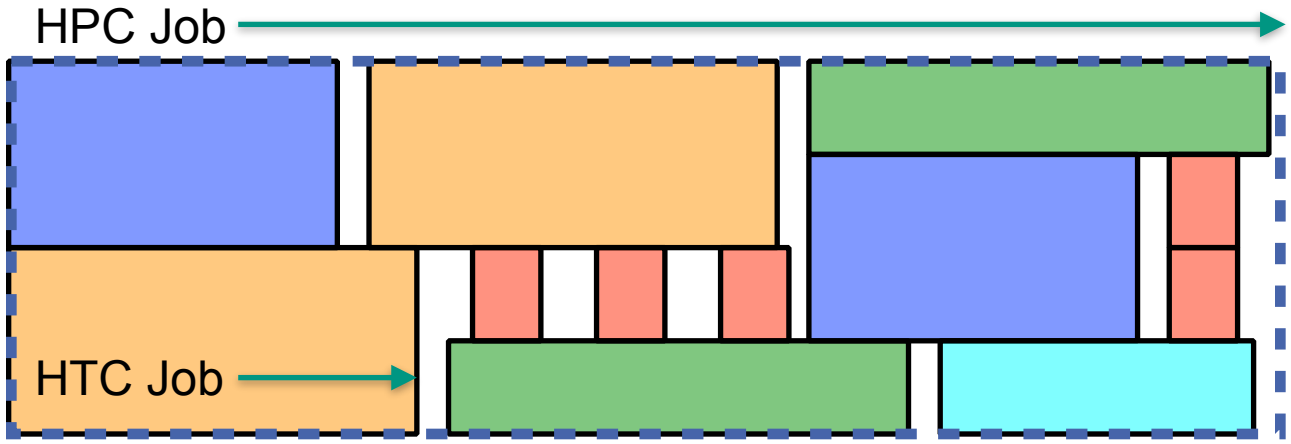
Fundament: Multi-User HTC Jobs für HPC

- Kombination mehrerer HTC Jobs zu einem HPC Job
  - Meta-Job belegt kompletten Knoten für feste Zeit
  - Ressourcen Kombination: kleine Jobs nebeneinander
  - Zeitliche Kombination: kurzlaufende Jobs nacheinander



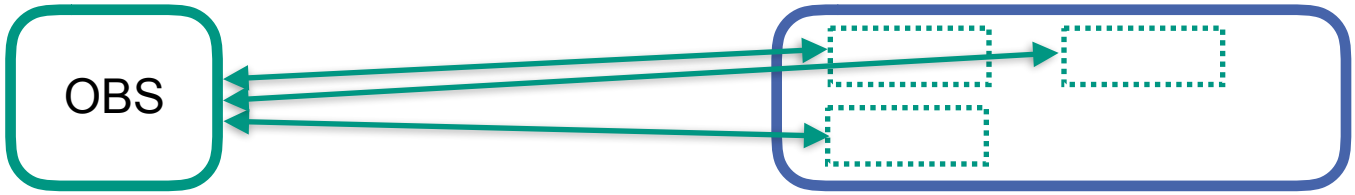
Fundament: Multi-User HTC Jobs für HPC

- Kombination mehrerer HTC Jobs zu einem HPC Job
  - Meta-Job belegt kompletten Knoten für feste Zeit
  - Ressourcen Kombination: kleine Jobs nebeneinander
  - Zeitliche Kombination: kurzlaufende Jobs nacheinander
- Mehrstufiges Scheduling vereinfacht Handhabung
  - HPC: Whole-node Scheduling für ganze Nutzergruppen
  - HTC: Individuelles Scheduling mit niedriger Latenz
  - Gruppe definiert interne Priorität/Share/Grenzen



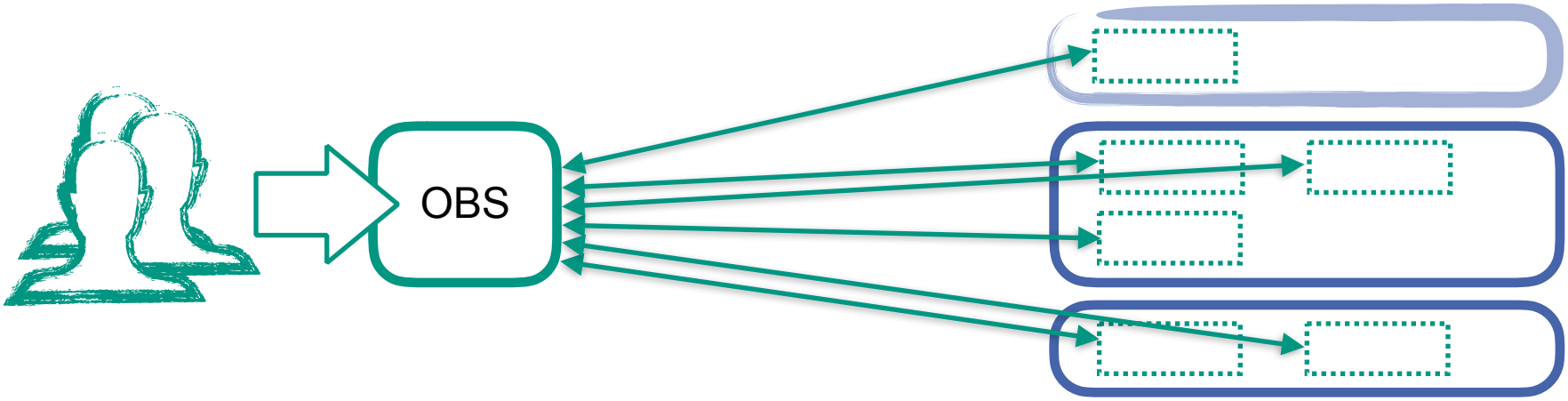
HPC als Cloud: Overlay Batch System

- HPC Workernode als temporärer HTC Workernode
  - Gruppe sendet HTC Workernode Daemon als Meta-Job
  - Verbindet sich zu separatem HTC Batch System der Gruppe
  - HTC Workernode führt dynamisch „echte“ Jobs aus



HPC als Cloud: Overlay Batch System

- HPC Workernode als temporärer HTC Workernode
  - Gruppe sendet HTC Workernode Daemon als Meta-Job
  - Verbindet sich zu separatem HTC Batch System der Gruppe
  - HTC Workernode führt dynamisch „echte“ Jobs aus
- Gruppennutzer interagieren nur mit Overlay Batch System (OBS)
  - Bekannte APIs, Tools, Authentifizierung, Ansprechpartner
  - Isoliert von HPC Batch System und anderen Gruppen
  - Kombination von Ressourcen mehrerer Batch Systeme

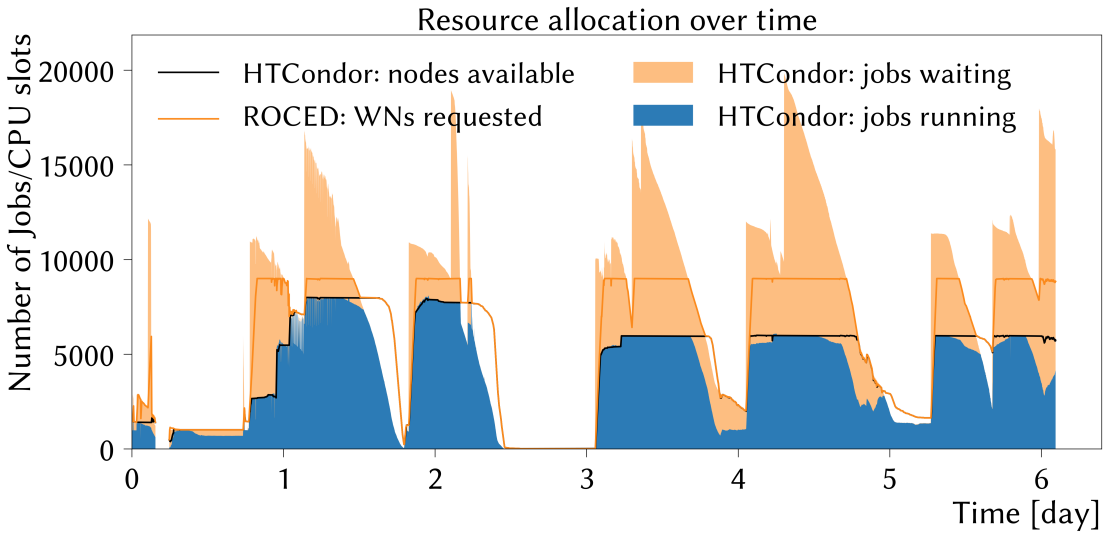


Dynamische Ressourcen (Ansatz A)

- Meta-Scheduler submittiert Meta-Jobs nach Bedarf
  - Monitoring der wartenden Jobs im OBS
  - Berechnung der idealen Ressourcen
  - Sendet passende Meta-Jobs für aktuellen Bedarf

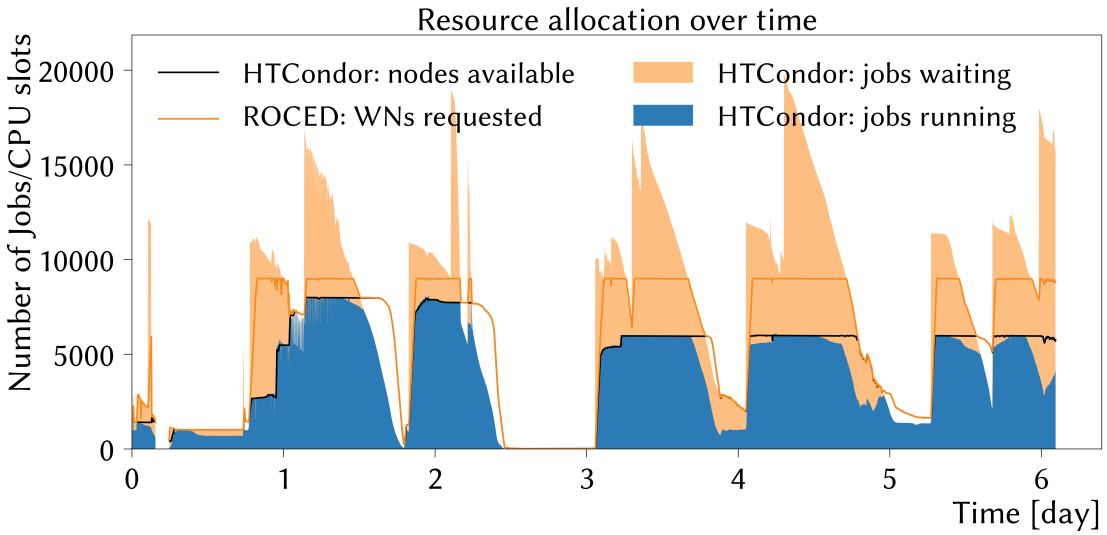
Dynamische Ressourcen (Ansatz A)

- Meta-Scheduler submittiert Meta-Jobs nach Bedarf
  - Monitoring der wartenden Jobs im OBS
  - Berechnung der idealen Ressourcen
  - Sendet passende Meta-Jobs für aktuellen Bedarf
- ROCED (2010-2018) [**R**esponsive **O**n-Demand **C**loud **E**nabled **D**eployment]
  - Dynamische Ressourcen für ETP Analysegruppe
  - Freiburg HPC Cluster „NEMO“, 1und1, Amazon EC2, …



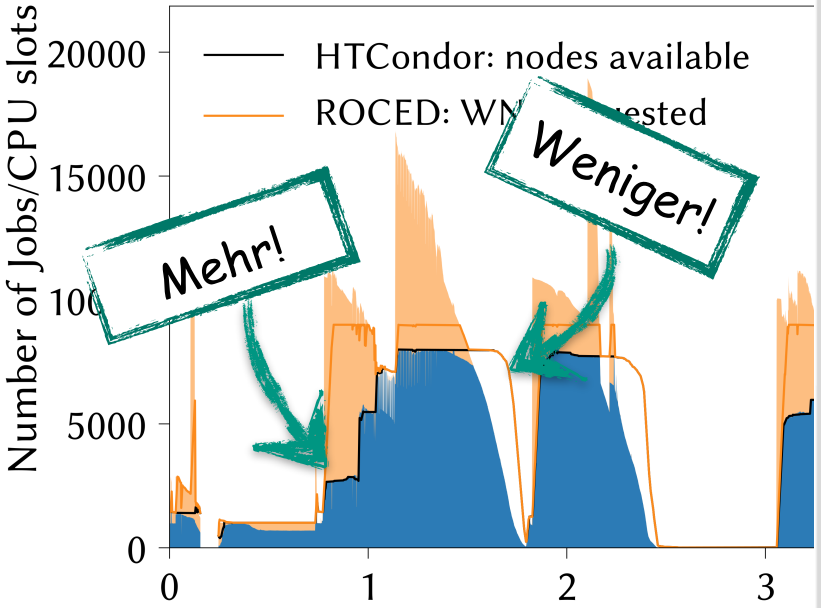
Dynamische Ressourcen (Ansatz A)

- Meta-Scheduler submittiert Meta-Jobs nach Bedarf
  - Monitoring der wartenden Jobs im OBS
  - Berechnung der idealen Ressourcen
  - Sendet passende Meta-Jobs für aktuellen Bedarf
- ROCED (2010-2018) [**R**esponsive **O**n-Demand **C**loud **E**nabled **D**eployment]
  - Dynamische Ressourcen für ETP Analysegruppe
  - Freiburg HPC Cluster „NEMO“, 1und1, Amazon EC2, …
- Nur für einfache Fälle!
  - Minimum 3 Scheduler
  - Hohe Latenzen, viele Regeln, Variablen, …



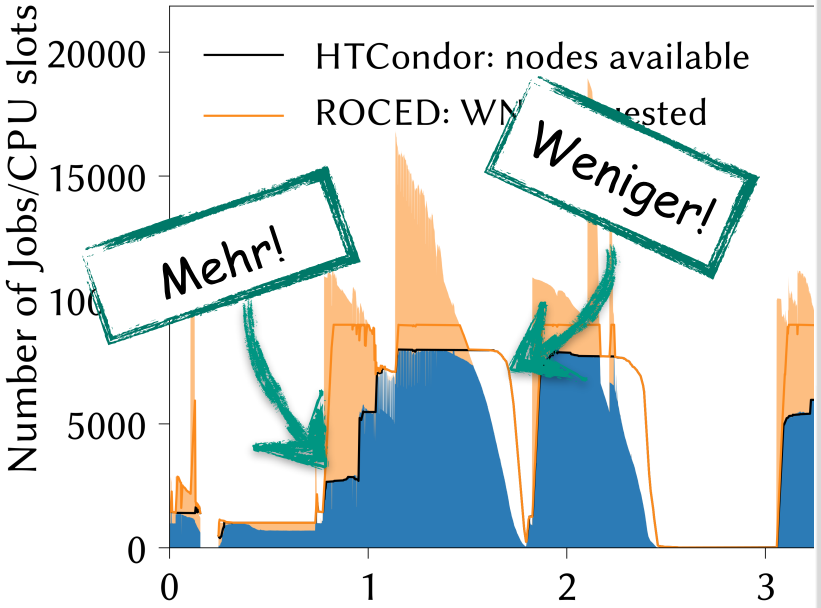
Dynamische Ressourcen (Ansatz B)

- Meta-„Scheduler“ betrachtet Effektivität von Meta-Jobs
  - Monitoring aktiver Meta-Jobs und deren Inhalt
  - Mehr gut genutzte, weniger schlecht genutzte Meta-Jobs
  - Keine Kopplung/Vorhersage mit Batch System



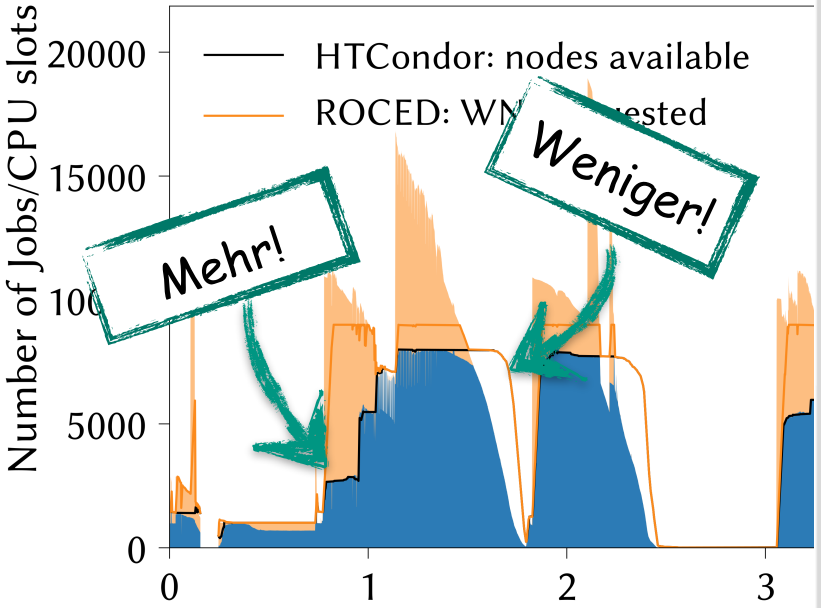
Dynamische Ressourcen (Ansatz B)

- Meta-„Scheduler“ betrachtet Effektivität von Meta-Jobs
  - Monitoring aktiver Meta-Jobs und deren Inhalt
  - Mehr gut genutzte, weniger schlecht genutzte Meta-Jobs
  - Keine Kopplung/Vorhersage mit Batch System
- COBalD (2018-heute) [**c**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]
  - Ressourcen für ETP und GridKa
  - Alle Ressourcentypen von ROCED



Dynamische Ressourcen (Ansatz B)

- Meta-„Scheduler“ betrachtet Effektivität von Meta-Jobs
  - Monitoring aktiver Meta-Jobs und deren Inhalt
  - Mehr gut genutzte, weniger schlecht genutzte Meta-Jobs
  - Keine Kopplung/Vorhersage mit Batch System
- COBalD (2018-heute) [**c**OBalD - the **O**pportunistic **Bal**ancing **D**aemon]
  - Ressourcen für ETP und GridKa
  - Alle Ressourcentypen von ROCED
- Anwendbar für komplexe Fälle
  - Keine Korrelation/Interaktion mehrerer Scheduler
  - Einfache Regelwerke: z.B. „bei N Cores halte X% Reserve“
  - Unabhängige COBalD Instanzen bedienen OBS transparent



Software Umgebung nach Bedarf

- Spezielle Anforderungen der HEP Community
  - Individuelle Analysesoftware und Frameworks
  - Dependencies auf seltene Libraries, Hilfsprogramme, …
  - Spezifische Protokolle/APIs (VOMS Proxy, XRootD, DCache, …)

Software Umgebung nach Bedarf

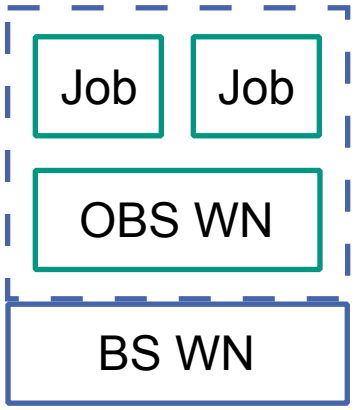
- Spezielle Anforderungen der HEP Community
  - Individuelle Analysesoftware und Frameworks
  - Dependencies auf seltene Libraries, Hilfsprogramme, …
  - Spezifische Protokolle/APIs (VOMS Proxy, XRootD, DCache, …)
- Support durch HPC Zentren unrealistisch
  - Software nicht generell sinnvoll einsetzbar
  - Unterschiedliche Anforderungen je Kollaboration
  - Community-spezifische Quellen

Software Umgebung nach Bedarf

- Spezielle Anforderungen der HEP Community
  - Individuelle Analysesoftware und Frameworks
  - Dependencies auf seltene Libraries, Hilfsprogramme, …
  - Spezifische Protokolle/APIs (VOMS Proxy, XRootD, DCache, …)
- Support durch HPC Zentren unrealistisch
  - Software nicht generell sinnvoll einsetzbar
  - Unterschiedliche Anforderungen je Kollaboration
  - Community-spezifische Quellen
- Mehrere getrennte Aufgaben
  - OS, Umgebung, Services, Verzeichnisse, … [MB]
  - Software, Libraries, Hilfsprogramme, … [GB]
  - Geteilte und individuelle Daten, … [TB]

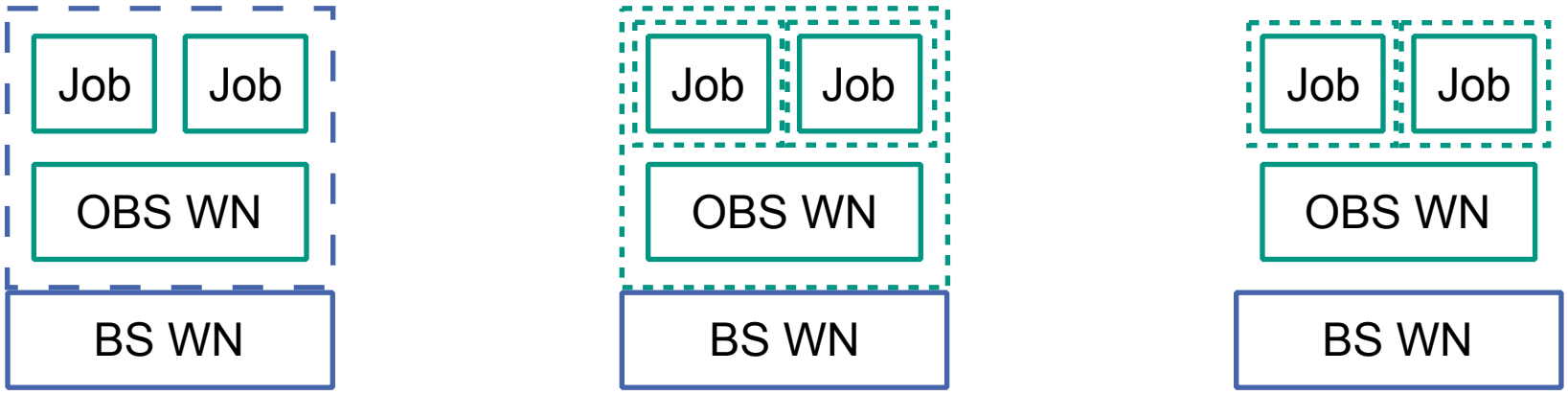
Openstack, Docker, Singularity, uDocker, …

- Vielzahl unterschiedlicher Technologien und Umsetzungen
  - Tradeoff: Isolation, Berechtigungen, Flexibilität, Effizienz, …
  - Fakt: Nicht unsere Entscheidung! (Aber wir reden gerne mit…)
  - Fakt: Nicht (nur) eine technische Frage!



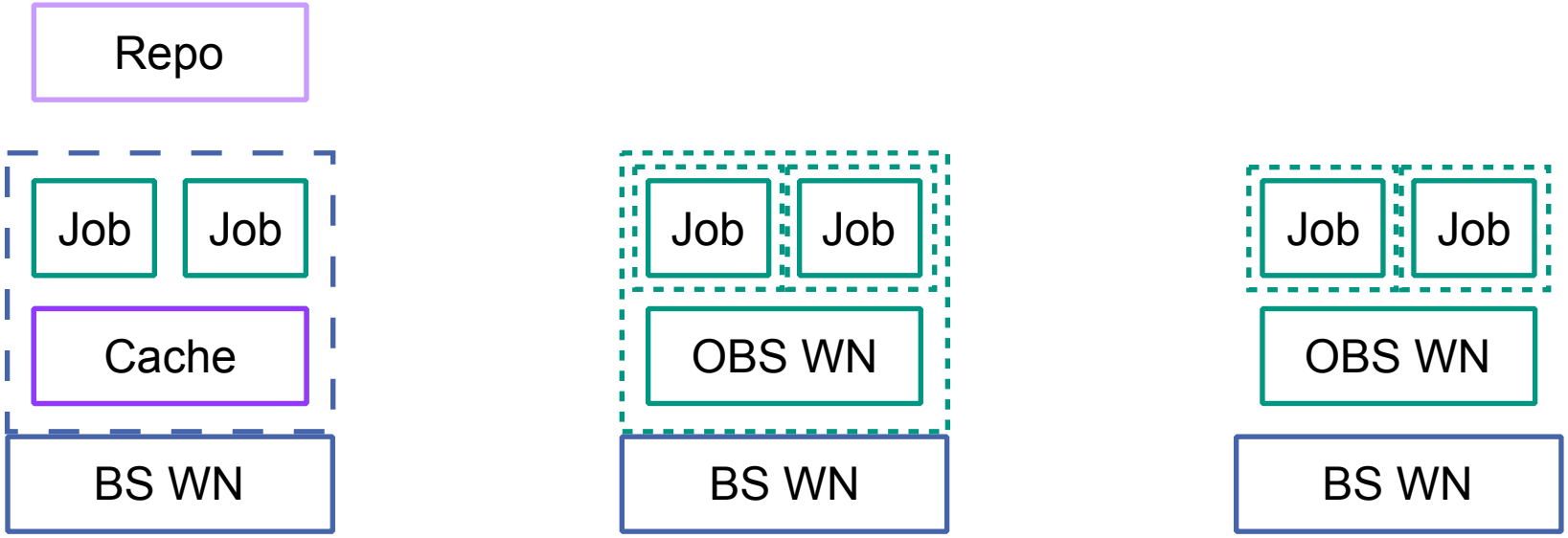
Openstack, Docker, Singularity, uDocker, …

- Vielzahl unterschiedlicher Technologien und Umsetzungen
  - Tradeoff: Isolation, Berechtigungen, Flexibilität, Effizienz, …
  - Fakt: Nicht unsere Entscheidung! (Aber wir reden gerne mit…)
  - Fakt: Nicht (nur) eine technische Frage!
- Nutzungsprofil ergibt sich aus Isolationsmodell
  - Komplette Isolation: Virtuelle Maschine
  - Privilegierte Isolation: Docker, …
  - Starke Isolation: Singularity, …



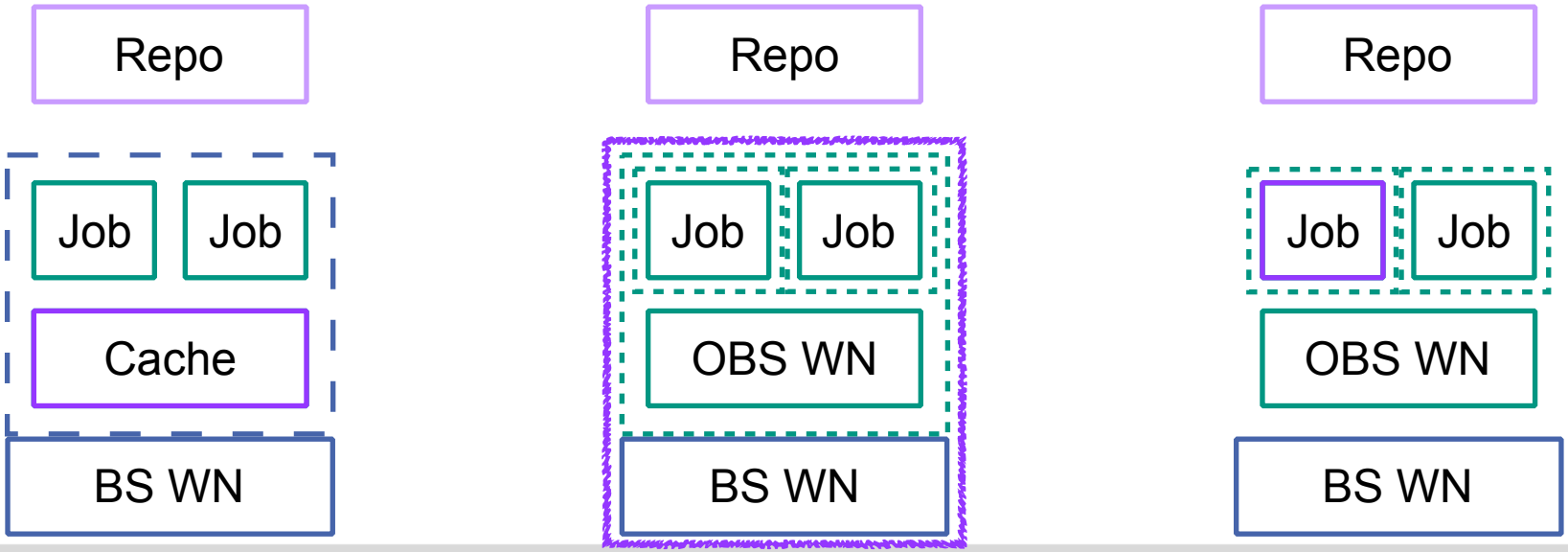
Software On-Demand: CVMFS

- Dynamische Bereitstellung von Nutzersoftware und Frameworks
  - **C**ern **VM** **F**ile**s**ystem: read-only FUSE, HTTP und lokaler Cache
  - Administratoren erlauben, Nutzergruppe managen Repositorien
  - Etabliert im WLCG und verwandten Anwendungsbereichen



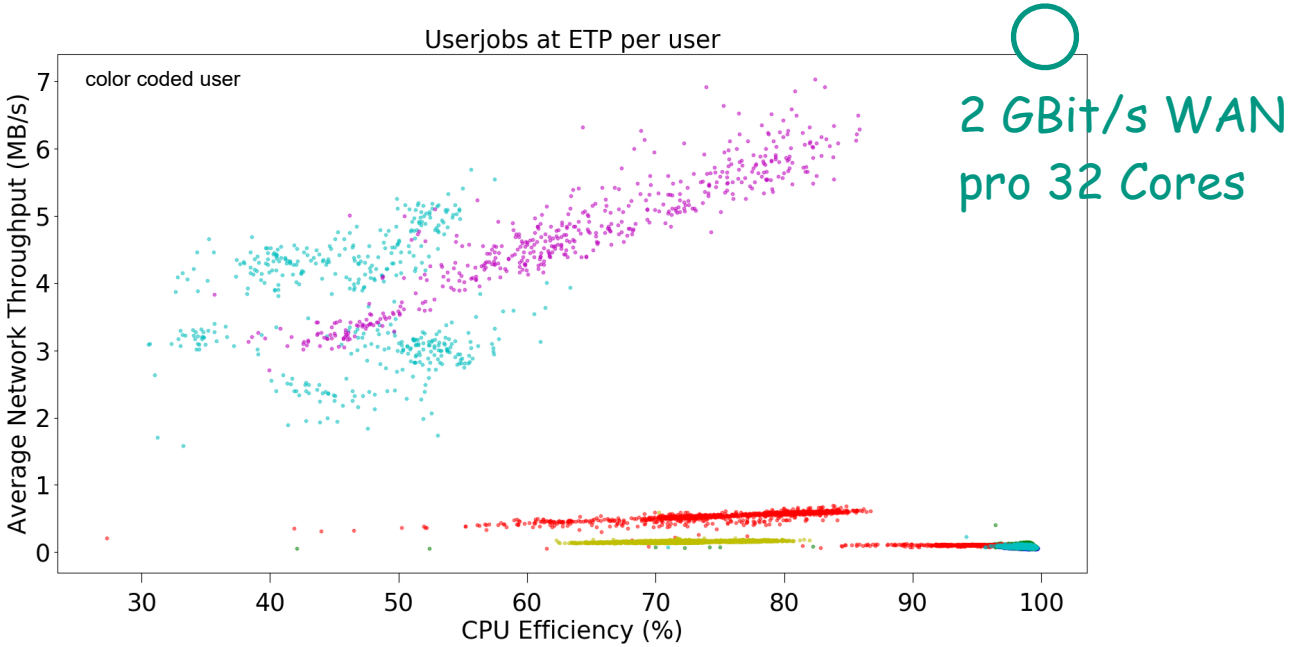
Software On-Demand: CVMFS

- Dynamische Bereitstellung von Nutzersoftware und Frameworks
  - **C**ern **VM** **F**ile**s**ystem: read-only FUSE, HTTP und lokaler Cache
  - Administratoren erlauben, Nutzergruppe managen Repositorien
  - Etabliert im WLCG und verwandten Anwendungsbereichen
- Cache-Effizienz abhängig von Isolation
  - Möglichst hohe Widerverwertbarkeit/Lebensdauer
  - Starke Isolation betrifft auch Caches



Datenanalyse ohne Daten

- HEP Analysen nutzen global verfügbare Datenquellen
  - Streaming über WAN mit 10 KB/s - 20 MB/s pro Core
  - Datenvolumen zu groß für HPC (~500PB pro Experiment)



Datenanalyse ohne Daten

- HEP Analysen nutzen global verfügbare Datenquellen
  - Streaming über WAN mit 10 KB/s - 20 MB/s pro Core
  - Datenvolumen zu groß für HPC (~500PB pro Experiment)
- Indirektes Scheduling der Netzwerkbandbreite durch CPU-Effizienz
  - Saturiertes Netzwerk messbar als schlecht genutzte Meta-Jobs
  - Meta-Scheduler schützt ausgelastete HPC/Cloud/Grid Zentren