HPC-Systeme:
Cluster Management:
- Config &Administration
- Job Management

Prof. Dr. Volker Gülzow

Dr. Yves Kemp. Dr. Sergey Yakubov

SS 2018

Management Tool

Management der Systeme aus Sicht der Nutzer:

Job Handling, Workflow, Benutzung durch mehrere Nutzer

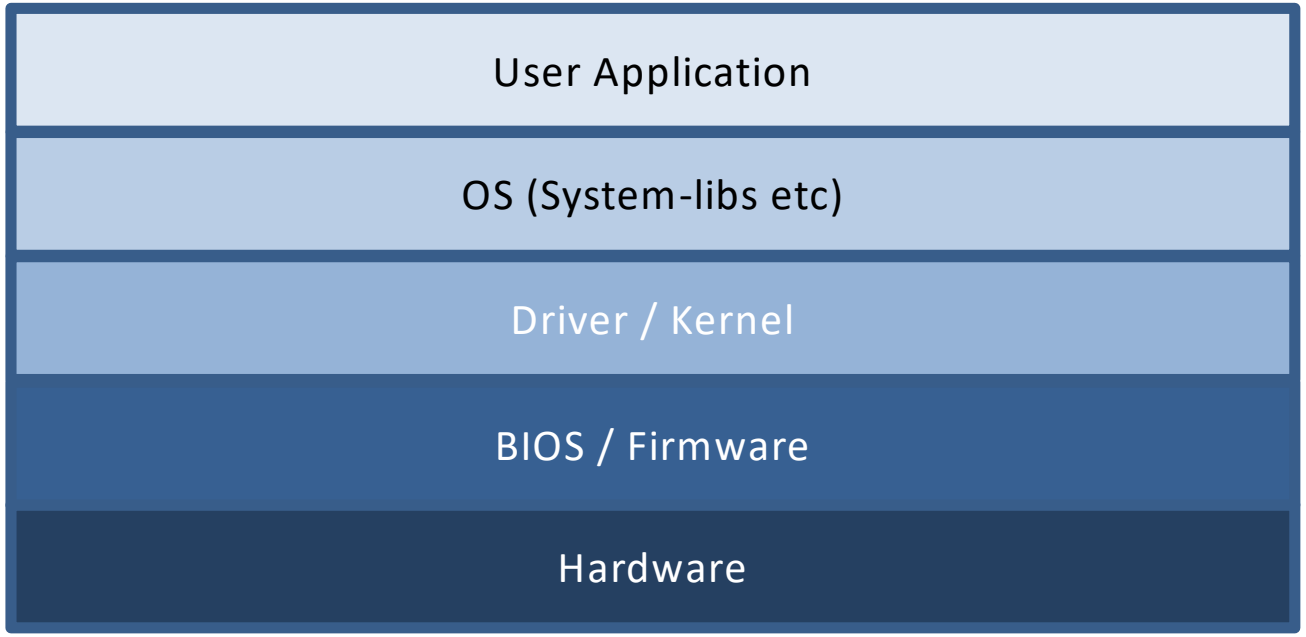
-> Zweiter Teil dieser Vorlesung (und Übung)

Management der Systeme aus Sicht von Sys-Admins:

Installation, Konfiguration, Management, Life-Cycle-Management

- Speziell für grosse homogene Cluster
- -> Erster Teil dieser Vorlesung

Server, Firmware, Driver, OS etc.



Kompatibilitäts-Matrix

- Um einwandfrei miteinander zu funktionieren müssen alle Komponenten entsprechende, kompatible Versionen haben
- Upgrades in einer Schicht ziehen häufig Upgrades in anderen Schichten mit sich
- Ihre Aufgabe als Admin:
  - Change-Logs lesen
  - Änderungen an Test-Systemen ausprobieren
  - Versionsstände monitoren
  - Tools zum möglichst automatischen Upgrade einsetzen
- Je komplexer die Umgebung, umso schwieriger die Abhängigkeiten

Homogene Cluster: Ja, gerne!

- Es ist auf jeden Fall ratsam, homogene Cluster einzusetzen (Hardware, Software, Configuration...)
- Reduktion der Komplexität
- Dies ist auch das übliche Beschaffungs-Vorgehen bei grossen HPC-Clustern
- Leider gibt es viele Cluster, bei denen dieses Vorgehen nicht einfach möglich ist
  - zB auch am DESY: Fortlaufende, inkrementelle Erweiterung der Infrastruktur
- Bei einfachen Farmen (ohne schnellen Interkonnekt) ist die Homogenität weniger wichtig

Installation des Betriebsystems

- Server verfügen über Netzwerk-Boot
- PXE: Preboot eXecution Environment
- DHCP und TFTP
- Kernel und RAMDisk werden über Netzwerk geladen und gebootet

Disk-less boot systeme

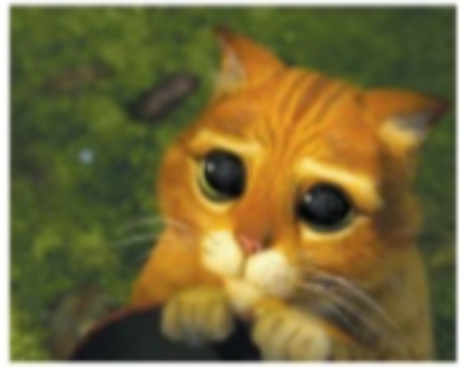
- Das kann es schon gewesen sein: Eventuell hat der Server keine interne Festplatte, oder die Festplatte wird nicht für das OS benutzt
- **/tmp ?** Irrelevant, es gibt üblicherweise ein schnelles Cluster-File-System
- **/etc ?** Kann entweder über Konfig-Management-Systeme angepasst werden, oder auch von einem Netzwerk-Filesystem kommen
- **/var ?** Logging geht an zentralen Log-Host
- **Swap ?** HPC Nodes haben typischerweise viel RAM

Konfiguration Management

- 1 Server: Einfach:
  - ssh root@server
  - vi /etc/service.conf
- Mehrere: Einfach
  - vi /local/etc/service.conf
  - for i in `cat server.list`; do scp /local/etc/service.conf root@${i}:/etc; done
- Oder?

Service Model

Borrowed from
@randybias at Cloudscaling
http://www.slideshare.net/randybias/the-cloud-revolution-cyber-press-forum-philippines



- Pets are given names like pussinboots.cern.ch
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health

- Cattle are given numbers like vm0042.cern.ch
- They are almost identical to other cattle
- When they get ill, you get another one

- Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

https://www.slideshare.net/gmccance/cern-data-centre-evolution

Konfiguration Management

- Nicht nur ein Tool, eher ein Mindset, oder Workflow
- **Ziel**:
- Beschreiben von gewünschten Konfigurationen / Endzuständen
- Kategorisierung der unterschiedlichen Systeme nach unterschiedlichen gewünschten Endzuständen
- Nachvollziehbarkeit der Änderungen an den Endzuständen
- Überprüfbarkeit der erreichten Endzustände

Haben Sie das Wort „Zustand“ gelesen?

Zustandsbezogene Konfiguration

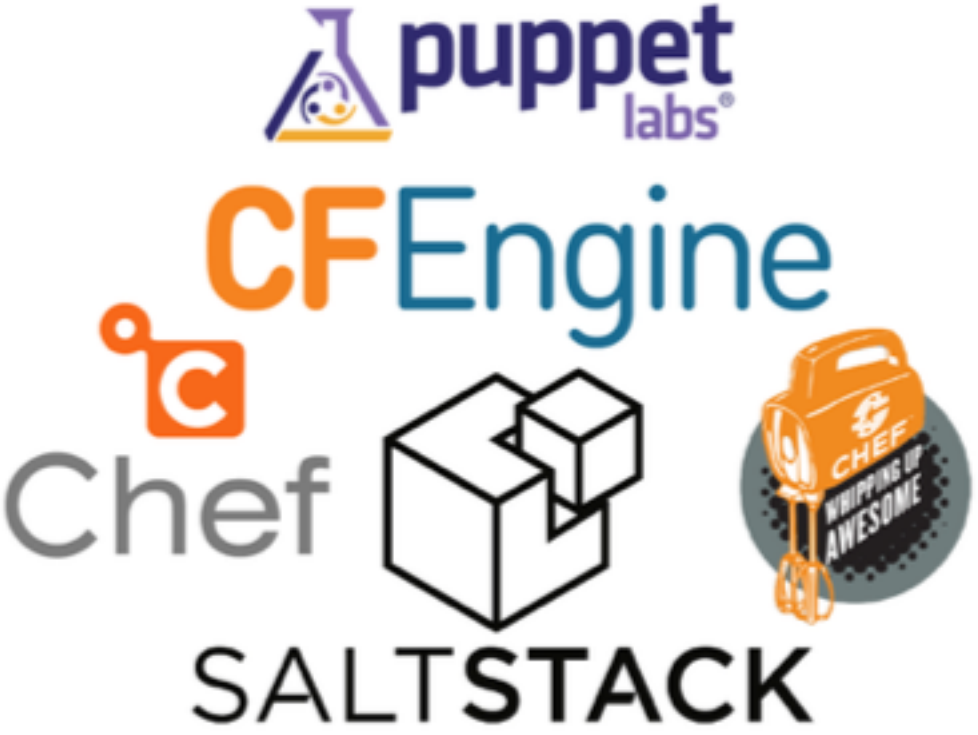
- „Ich will Zustand XYZ haben“
- vs
- „Ich mache die Aktion ABC“

- Dies mag Sie als Informatiker erstmal nicht schocken, ist aber ein Mentalitätswandel!
  - Üblicherweise arbeiten Sie nicht zustands-bezogen!

Beispiele

- CFEngine … 1993 von Mark Burgess entwickelt. Mark Burgess hat auch einige theoretische Grundlagen zu Configuration Management beschrieben
- Chef
- Ansible
- Salt
- Puppet (DESY)
- …

http://www.tomsitpro.com/articles/open-source-cloud-computing-software,2-754-10.html



Unterschiede zwischen den Produkten, zB:

- Skalierbarkeit, Performance
- Rollenbasiertes Arbeiten, Mandantenfähigkeit
- Portierbarkeit: Viele unterschiedliche Betriebsysteme/Flavours
- Eineindeutigkeit, Vorhersagbarkeit, Reproduzierbarkeit und Idempotenz
- „Ich kann ein Buch dazu kaufen“ oder „Ich kann einen Consultant einkaufen“
- Es soll eine möglichst große Community geben, die ggbfs. Lösungen teilt
- Push vs Pull (mittlerweile verschwimmt dieser Unterschied immer mehr)

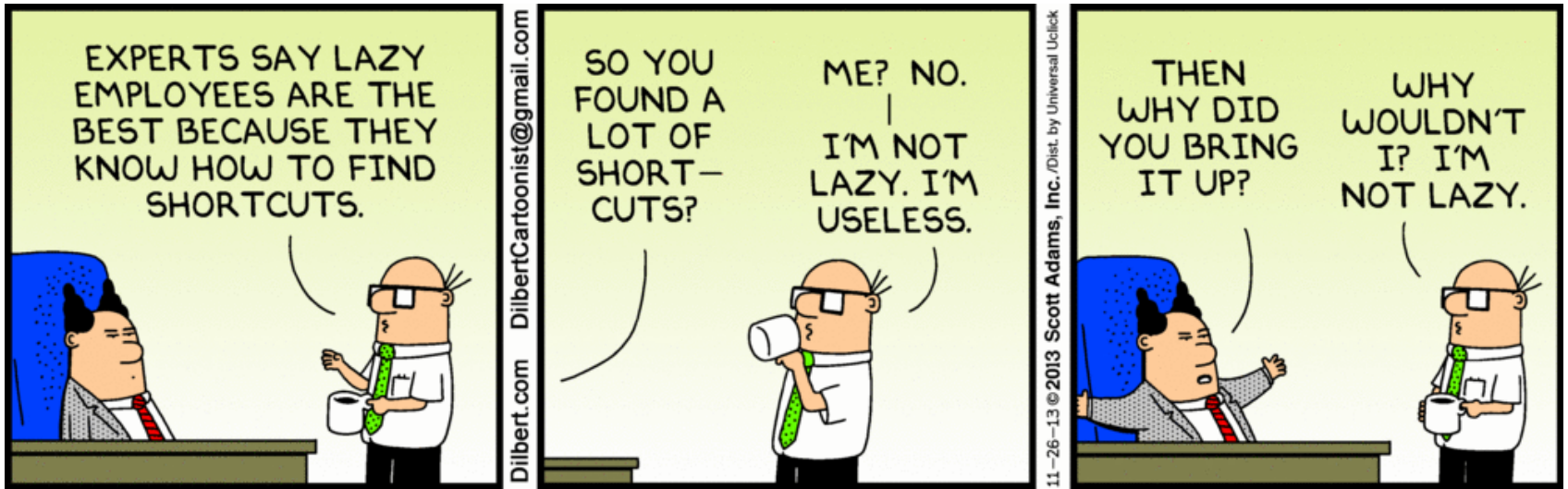
Auszug aus einem Puppet Beispiel-Manifest

```
user { 'dave':
      ensure     => present,
      uid        => '507',
      gid        => 'admin',
      shell      => '/bin/zsh',
      home       => '/home/dave',
      managehome => true,
    }
```

Software-Bereitstellung & faule Sys-Admins

- Ein fauler Sys-Admin installiert immer nur Pakete aus Repositories
- `./configure & make & make install` kennt er nur vom Hörensagen

- … Und der faule Sys-Admin hat Recht!
- Selbst-Kompilierte Software ist auf lange Sicht nicht zu maintainen



Betrieb von grossen HPC Clustern

- Typischerweise klare Projektphasen:
  - Planung und Beschaffung
  - Aufbau, Installation, Test, Optimierung, Abnahme
  - Betrieb
    - Typischerweise langfristig geplante Maintenance-Phasen
  - In enger Zusammenarbeit mit Hersteller

  - “BG/Q Updates have been installed. Please recompile your applications” … ~einmal jährlich

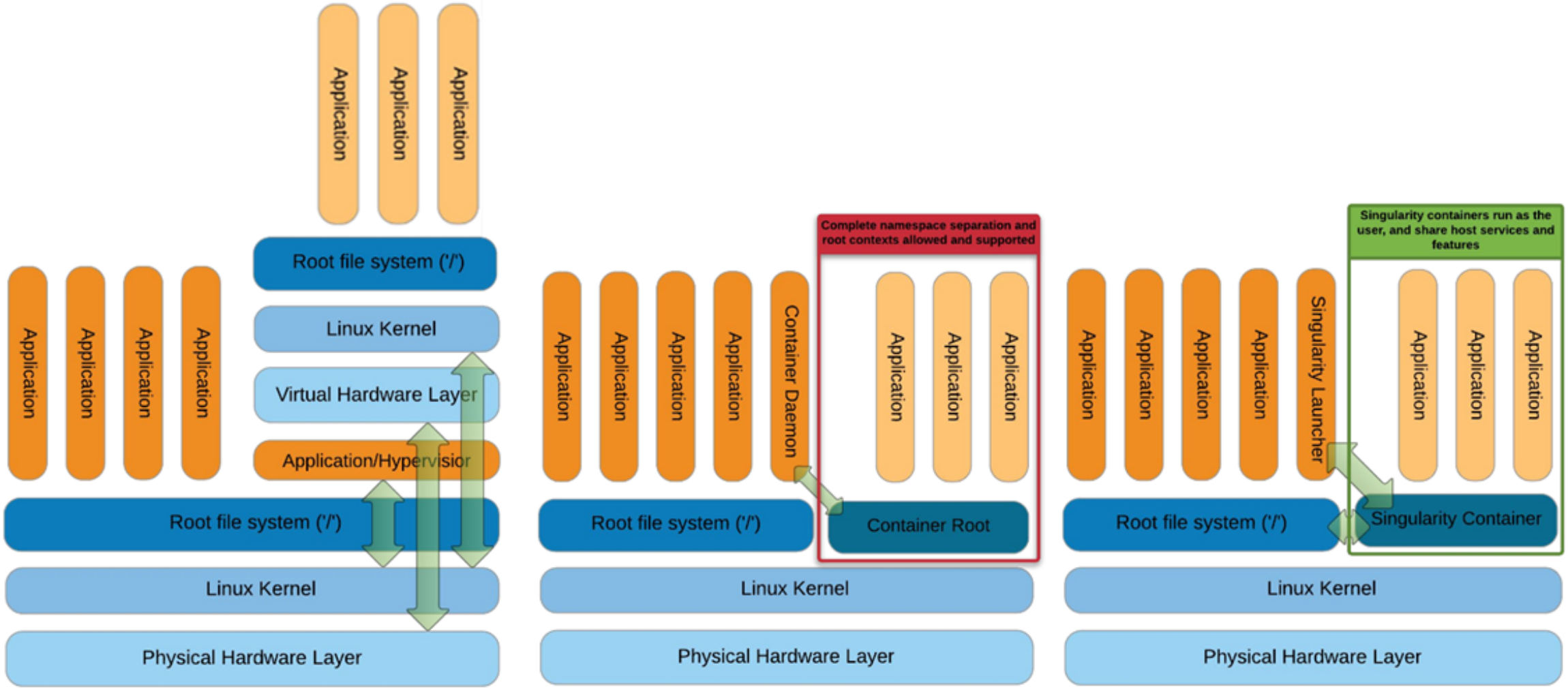
Und Cloud?
Virtualisierung?
Container?

HPC in der Cloud (HPCaaS)

- Ja, ist ein Trend
- ... Und basiert (oft) auf Virtualisierung
- Ist aber typischerweise abgespecktes HPC
- ZB hat Amazon zwar HPC im Angebot
- Allerdings (meines Wissens) kein InfiniBand, sondern nur 10 GE, in etwas kontrollierteren Umgebungen
- Nicht schlecht, und für viele Applikationen ausreichend
  - Aber halt nicht für alle

Container Technologien

http://www.hpcadvisorycouncil.com/events/2017/stanford-workshop/pdf/GMKurtzer_Singularity_Keynote_Tuesday_02072017.pdf#43



Container & HPC: Viel Bewegung und Interesse

- Kein Performance-Verlust, ggbfs. Performance-Gewinn
- Trennung von OS (installiert und konfiguriert von faulen und konservativen Sys-Admins) und Applikationen (entwickelt von hyperaktiven Software-Gurus)
- Ermöglicht es, im Userland die geeigneten Libraries und geeignete Optimierungen zu verwenden

Container suchen gerade ihren Weg

- Welches Produkt?
- Welchen Einsatz?
- Zusammenspiel mit Puppet & Co?
- Klassische Pakete RPM/DEB?
- Potential & Gefahren?

Container im Hamburg Hafen
ap
Quelle: http://www.planet-wissen.de/hamburger-hafenimpressionen-100.html

... Und jetzt Scheduling

The point of the HPC scheduler is to keep everyone equally unhappy

Aus: E-Mail Signatur von James A. Peltier, HPC Coordinator, Simon Fraser University
… es gibt aber unzählige Varianten von diesem Satz

Zuteilung von Rechenzeit

- Grosse (wissenschaftliche) HPC Zentren:
- Erstmal:
  - Typischerweise Anträge, die dann begutachtet werden
  - Viel Forschung
  - Viel Papierarbeit
  - ... Und dann erst eventuell die Möglichkeit zu rechnen

Beispiel: **Application for Projects on JUQUEEN**

Regular calls and calls for large-scale-projects

The GCS/NIC-call is open for project leaders, whose affiliation is in Germany (or is a foreign office of a German institution). Also eligible are German scientists if they are working in an international organisation with significant German participation (e.g. CERN, ESA, ESO) and do not have a permanent position there. Interested researchers from abroad (within Europe) are invited to apply via PRACE.

Project applications for JUQUEEN may be submitted by any scientist qualified in his or her respective field of research. Computing resources are allocated on the basis of independent referees' reports. Apart from the **scientific relevance of the project**, an important criterion for the allocation of computing resources is that the project can make **efficient use of the computer and use a large number of processors in parallel for the simulations**.

http://www.gauss-centre.eu/gauss-centre/EN/HPCservices/HowToApply/LargeScaleProjects/largeScaleJUQUEEN.html

Beispiel: **Application for Projects on JUQUEEN**

The following criteria must be met by the projects in order to be eligible:

- Scientific excellence.
- Clear scientific goals and verifiable milestones on the way to reach these goals.
- Preliminary studies that show **good scaling behaviour** of the program to very high processor numbers (at least 8192 cores) under production conditions (i.e. typical parameter sets and problem sizes of the planned project, including I/O). Contact sc@fz-juelich.de for a test account.
- A detailed and clearly arranged work schedule in form of a table or Gantt chart.

Well-founded and detailed demonstration of the required runtime of the program and the total required CPU time.

Please consult also the detailed technical guidelines for projects applying for JUQUEEN
The smallest amount of computing time that should be requested is 5 Mill. core hours.

http://www.gauss-centre.eu/gauss-centre/EN/HPCservices/HowToApply/LargeScaleProjects/largeScaleJUQUEEN.html

Beispiel: **Application for Projects on JUQUEEN**

Review process

Large-scale projects (35 million core hours or more) are peer-reviewed by a committee of the GCS. If approved they will get increased user support and preferential processing of their jobs.

Regular projects are peer-reviewed by a committee of the John von Neumann Institute for Computing (NIC) on behalf of GCS.

http://www.gauss-centre.eu/gauss-centre/EN/HPCservices/HowToApply/LargeScaleProjects/largeScaleJUQUEEN.html

Beispiel: **Application for Projects on JUQUEEN**

Application

Computing time periods are yearly - with the possibility of application twice per year - and will begin on 1 May and on 1 November each year.

The next call for proposals will be announced on 10 July 2017. The deadline for the next application will be on **14 August 2017, 17:00 CET** for the computing time period from 1 November 2017 until 31 October 2018.

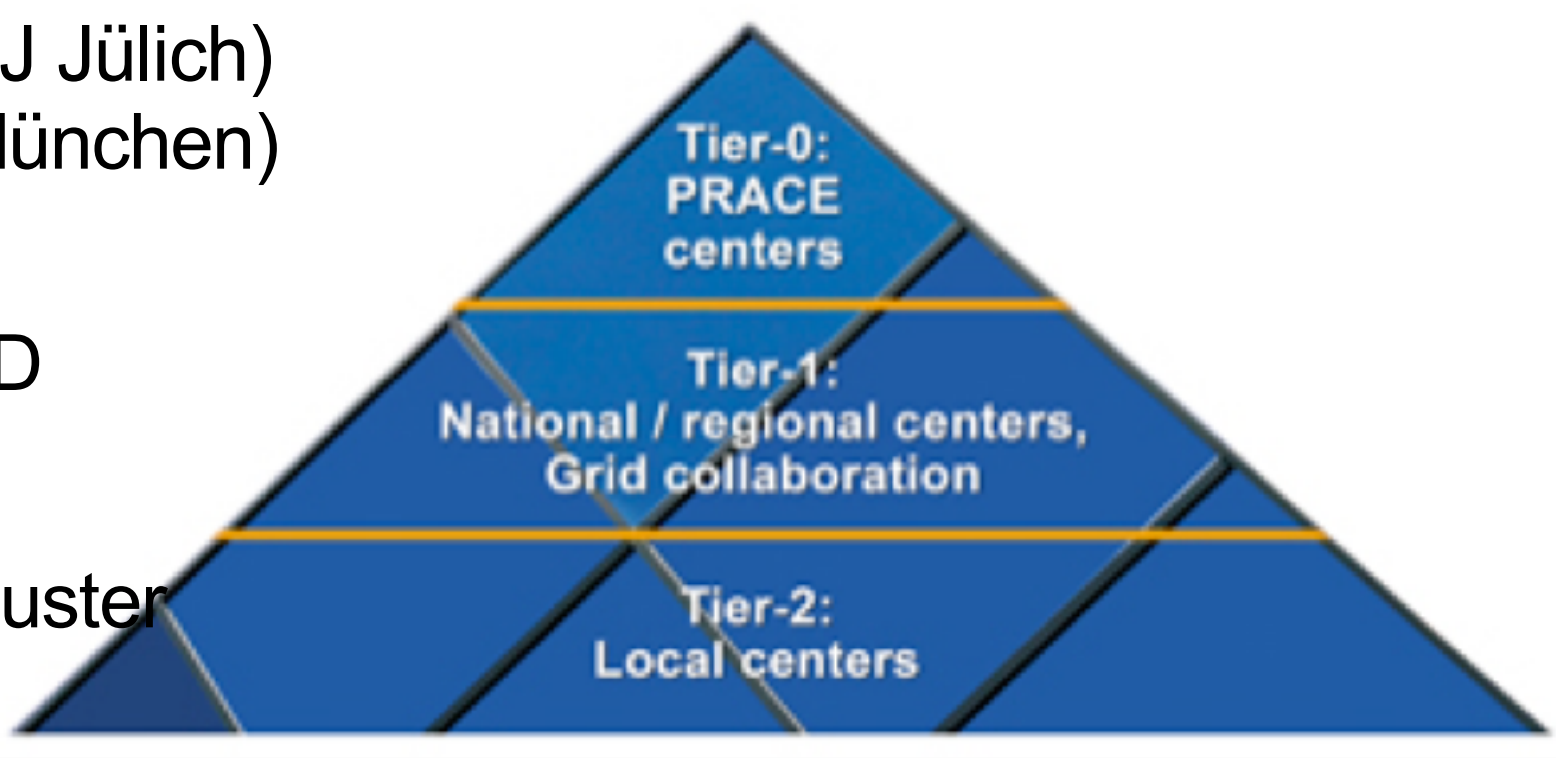
http://www.gauss-centre.eu/gauss-centre/EN/HPCservices/HowToApply/LargeScaleProjects/largeScaleJUQUEEN.html

Die HPC Tier Pyramide

ZB. JUQUEEN (FZJ Jülich)
SuperMUC (LRZ München)
... Europaweit

Einige Systeme in D

ZB DESY Maxwell Cluster



http://cnx.org/contents/Ug-5YUe5@1/The-European-e-Infrastructure-

Leben in einem Tier-2 Zentrum

- Keine formellen Anträge, keine formelle Begutachtung
- Es wird aufgepasst (von Admins und von freundlichen Mitnutzern) dass die Jobs „HPC-artig“ sind
- DESY investiert über IT in eine allgemeine Grund-Ausstattung
- Manche Projekte investieren Geld in die Infrastruktur, und erkaufen sich damit gewisse Rechte
- Andere Projekte nutzen die Ressourcen die „übrig bleiben“

- ... Wie das technisch und organisatorisch funktioniert lernen wir in den nächsten Folien

Job Verteilung

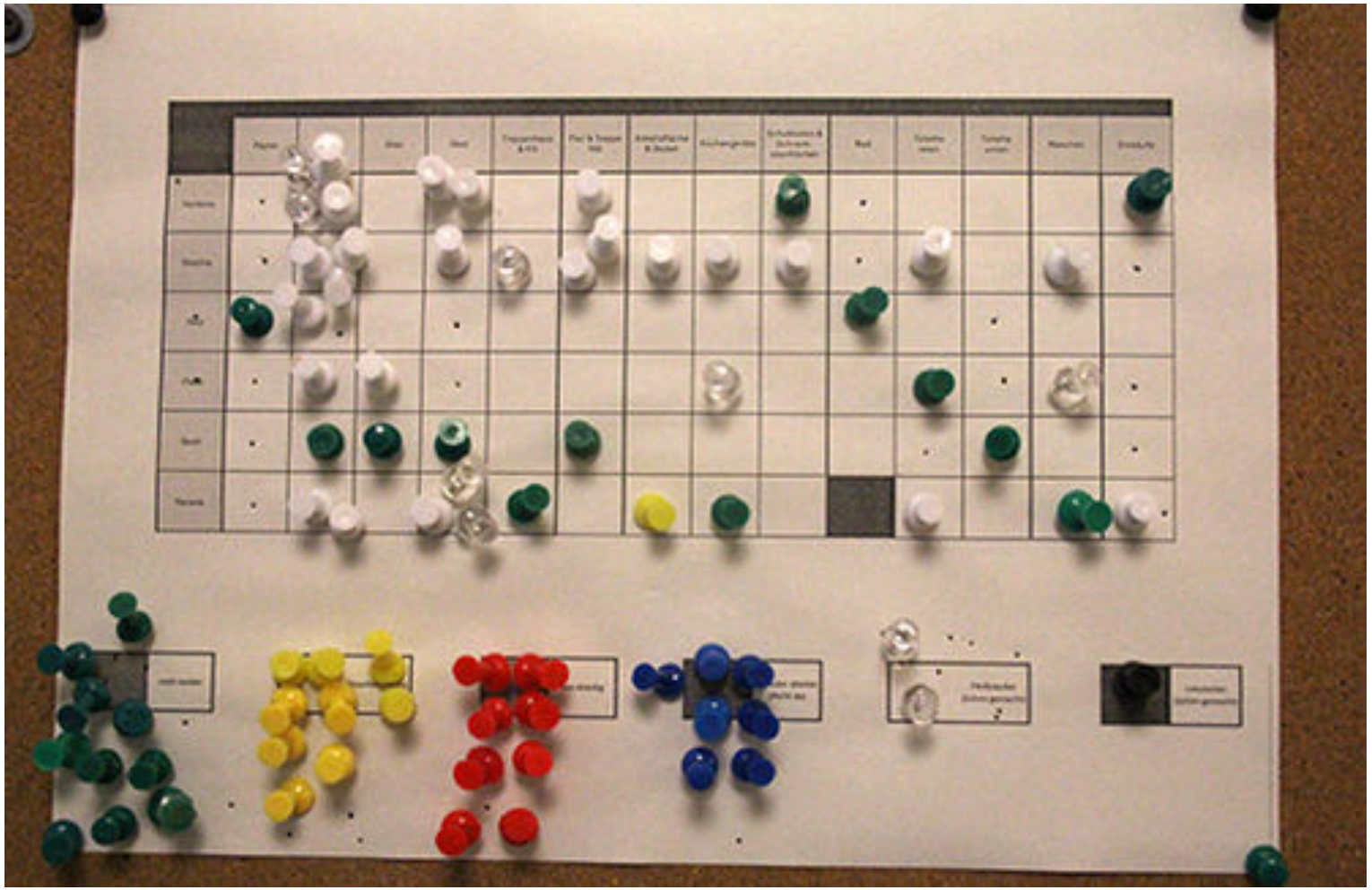
```
> cat steer.list
server1 23
server2 42
...

> cat steer.list | while read node seed; do
    (ssh $node „/home/kemp/run_simulation.sh $seed“ &)
done
```

Probleme

- Woher weiss ich, ob die Knoten gerade frei sind?
  - Ich kann einen anderen Job laufen haben
  - Jemand anderes kann einen Job laufen haben
  - Die Knoten können zB wegen Wartung ausser Betrieb sein

- Wir brauchen einen Scheduler!
  - Mein Wörterbuch sagt: Scheduler heisst u.a. auch: Planer, Disponent
  - Scheduler hat also sowohl eine Software-Seite, als auch eine organisatorische Seite

Einfachstes Beispiel eines Schedulers: Kalender



... Oder deren elektronische Variante

ssh und node-List auf Basis des Kalenders

https://www.zu-daily.de/daily/tiefenbohrung/2015/11-17-bathon-wg-putzplan.php

Kalender als Scheduler

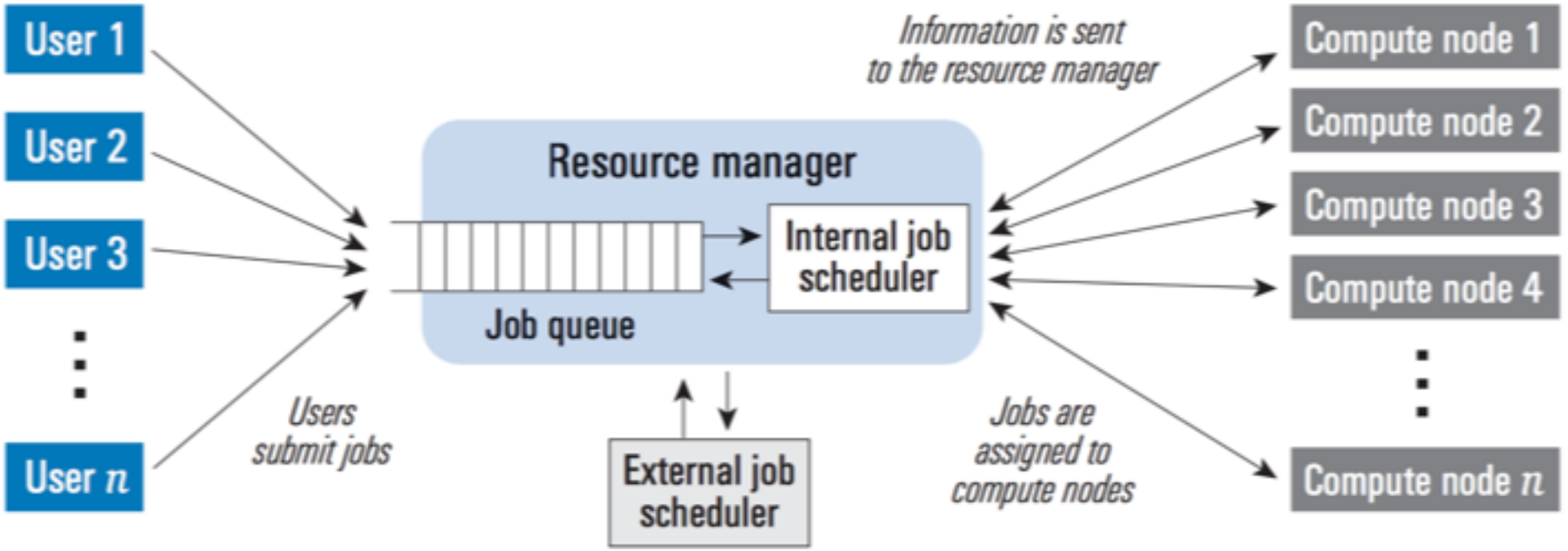
- Vorteile:
  - Kalender ist einfach zu bedienen
  - Jobs sind einfach per ssh abzuschicken, kein Anpassen der Software notwendig
  - Interaktivität ist gewährleistet
  - Feste Reservierungen (zB für Strahlzeiten) möglich
- Nachteile:
  - Sehr statisch, kann nicht auf variierende Laufzeiten eingehen
  - Verschnitt von Ressourcen
  - Reservieren auf Verdacht
- Die Nachteile nehmen schnell Überhand

Scheduler ist nicht alles

- Der Scheduler teilt nur ein: Wer darf wann wo rechnen?
- Das Batch-System nimmt die Jobs entgegen, verwaltet sie, übernimmt das Dispatching auf die Worker-Nodes und Verwaltung der Resultate
  - Das Batch-System arbeitet auch dann wenn Sie schlafen ☺

- Die Trennung ist heute verschwommen, viele Produkte können beides.

Typisches Setup



http://www.dell.com/downloads/global/power/ps1q05-20040135-fang.pdf

Einige Kriterien für die Auswahl

- Skalierbarkeit
  - Manche Systeme skalieren über 100k Server
  - Manche Systeme skalieren über 1M Jobs (running & queued)
  - Mittlerweile immer wichtiger: Skalierbarkeit in Clouds möglich?
- Mächtigkeit des Schedulers, implementierte Algorithmen
  - FairShare, Pre-Emption, Backfill, Prioritäten, Reservierung, ...
- Für HPC enorm wichtig:
  - Unterstützung von parallelen Jobs? Also Jobs die gleichzeitig mehrere Server umspannen?
- Stabilität, Administrierbarkeit, Integrierbarkeit mit Meta-Schedulern (zB Grid)
- Kosten

HTC und HPC

- HTC: High-Throughput-Computing
  - Typischerweise viele, lose gekoppelte Jobs (“trivial parallelisierbar“)
  - Entscheidend ist nicht die Peak-Power, sondern der Gesamt-Durchsatz innerhalb eines längeren Zeitraums
- HPC: High-Performance-Computing
  - Typischerweise parallele Jobs mit Inter-Process-Communication
  - Peak-Power ist wichtig, ebenso gute Kommunikation innerhalb des Jobs

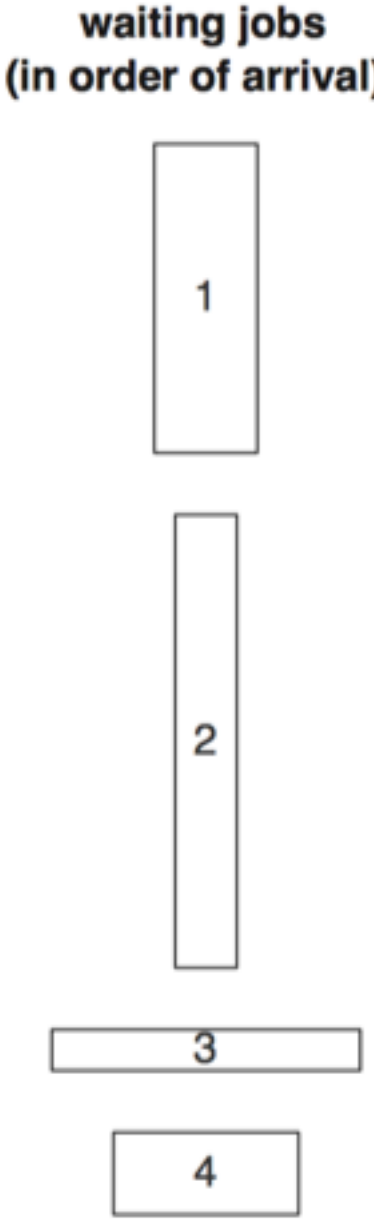
HPC

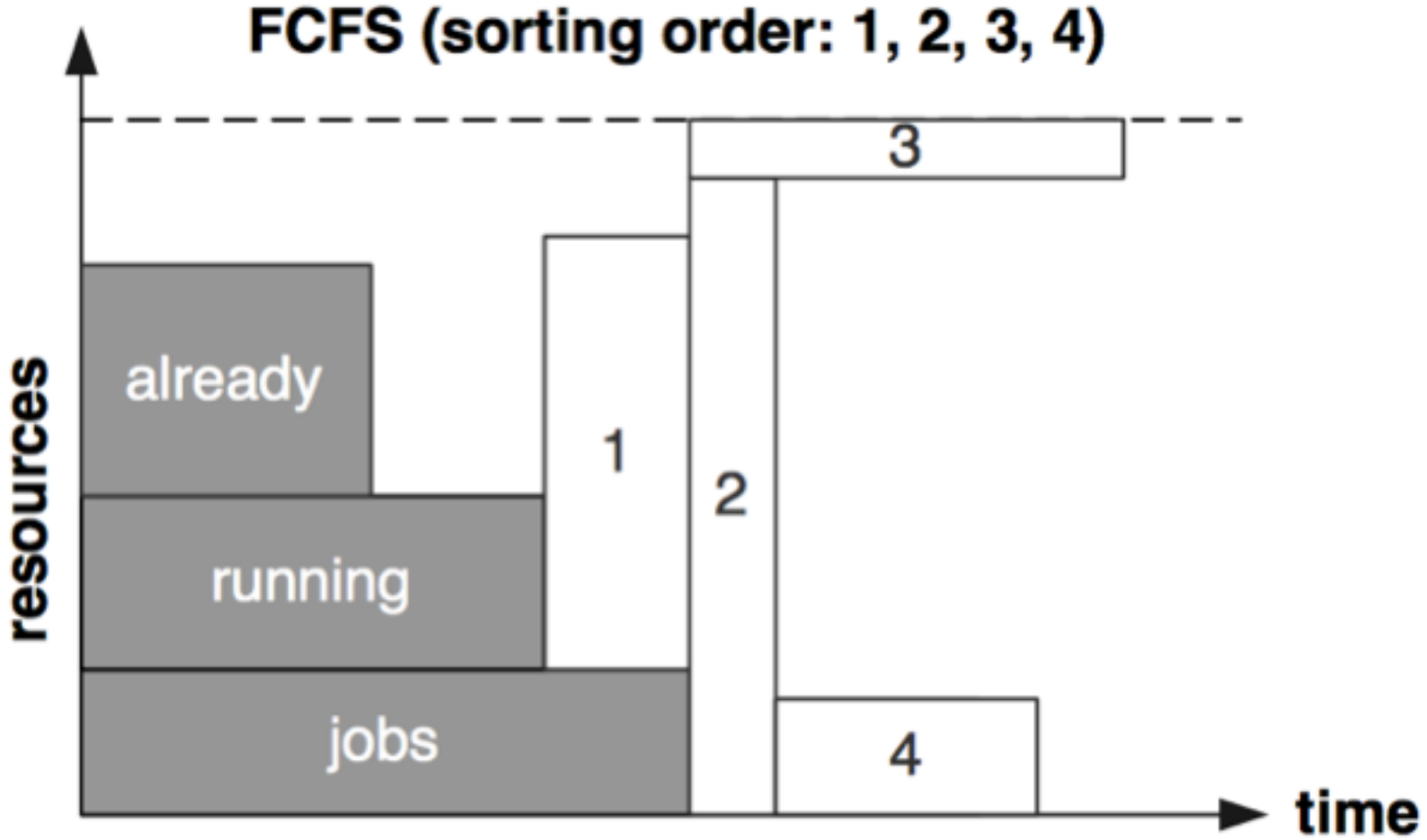
Quelle: http://auto.ferrari.com/de_DE
Folie inspiriert von HTCondor Enwickler

HTC

Quelle: http://www.mirror.co.uk/news/world-news/worlds-worst-traffic-jam-drone-6594560

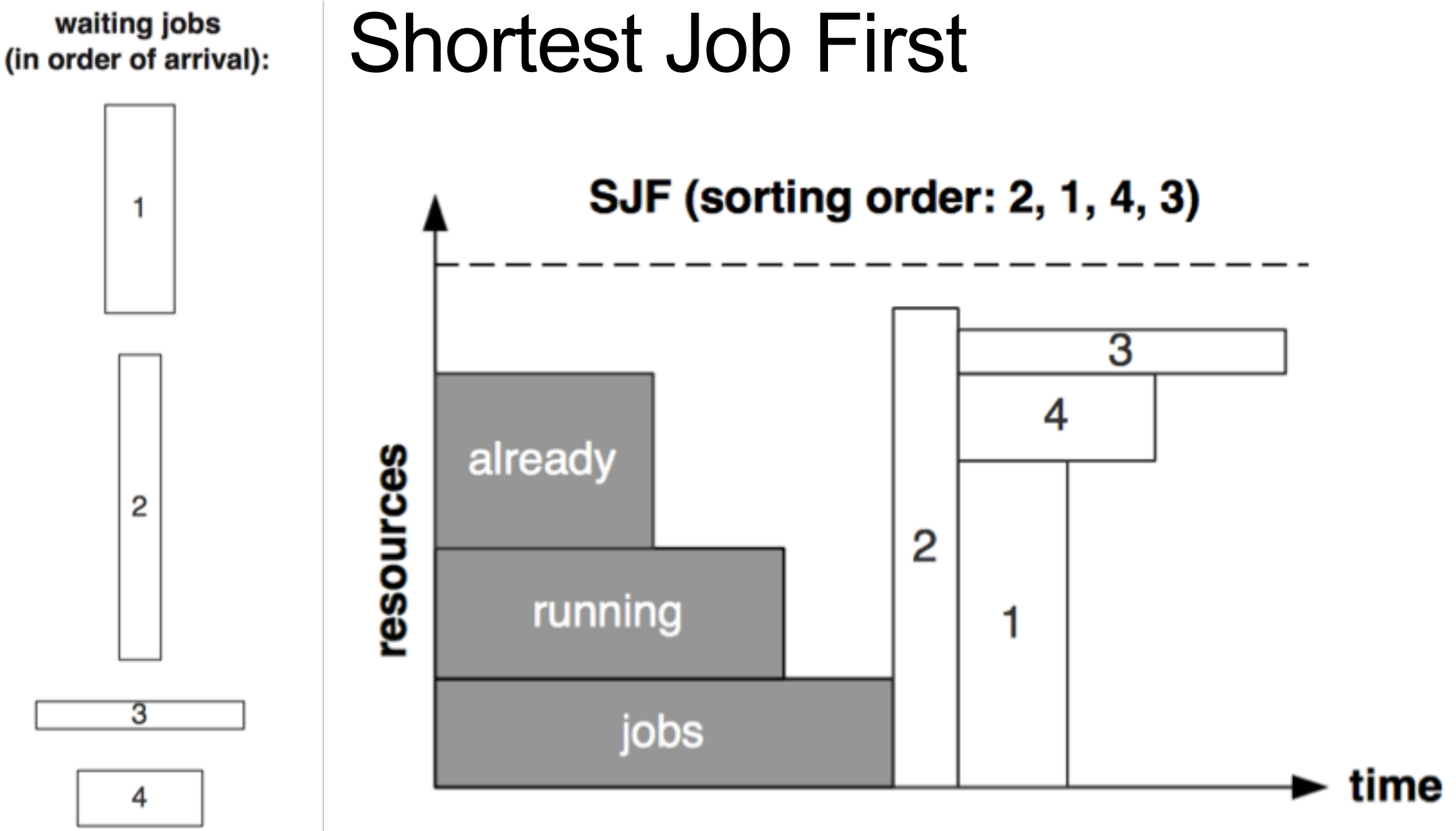
First-Come-First-Serve (FIFO)





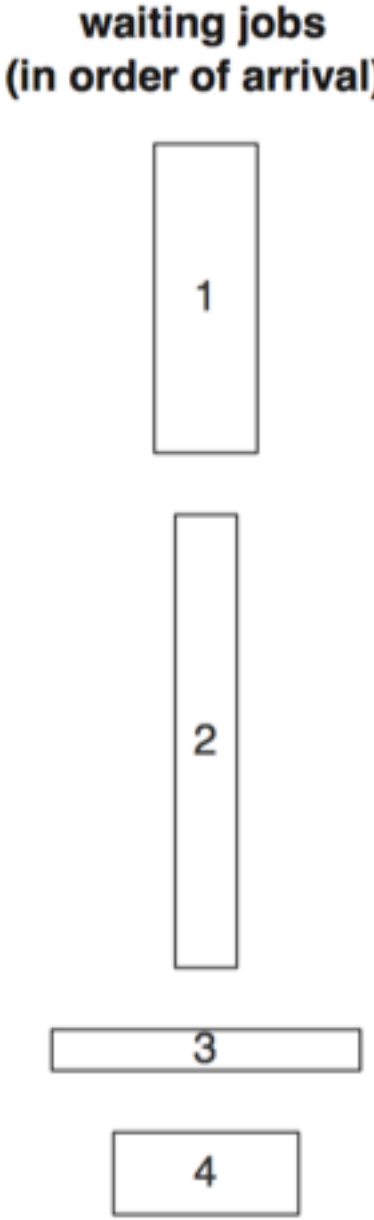
Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

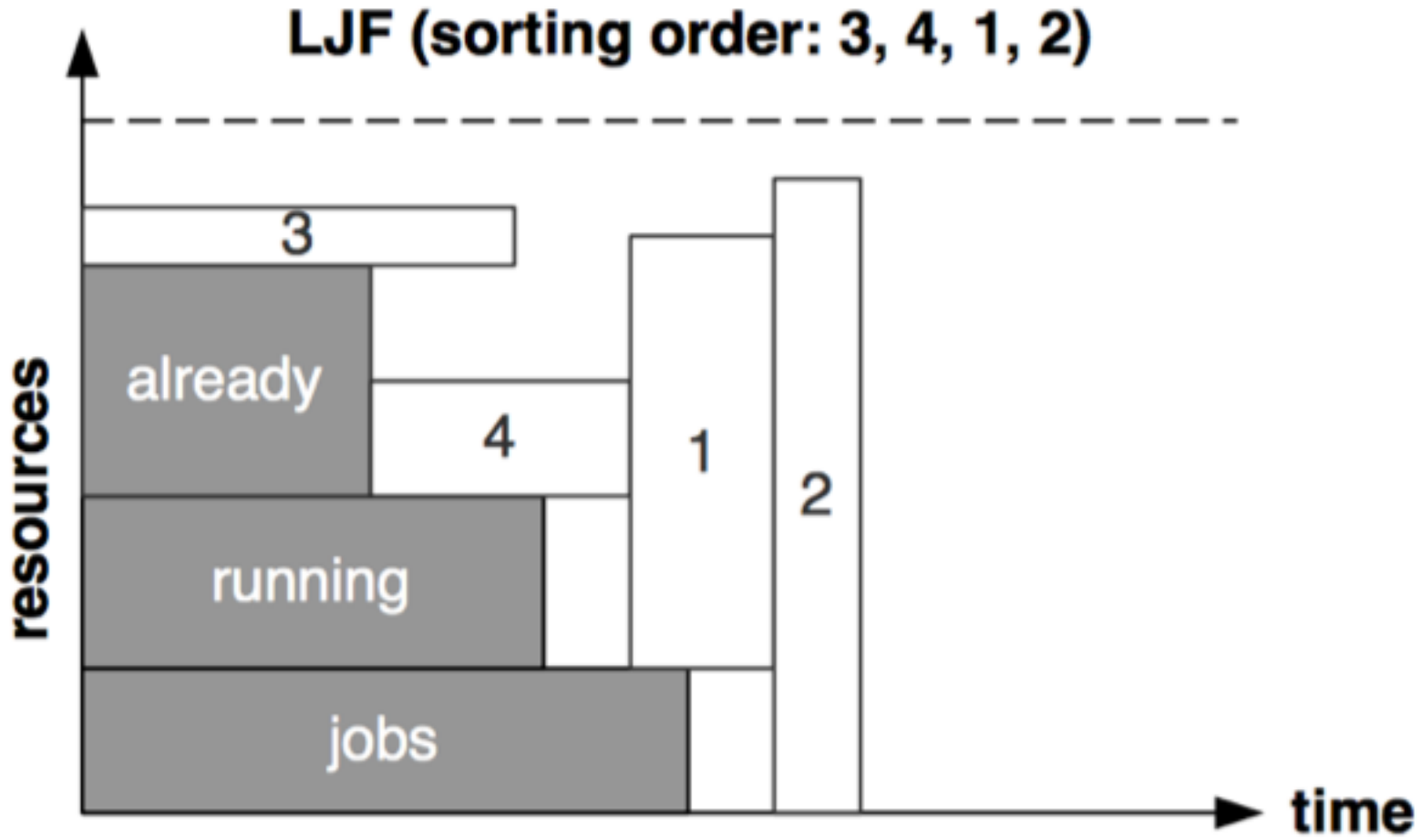
Shortest Job First



Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

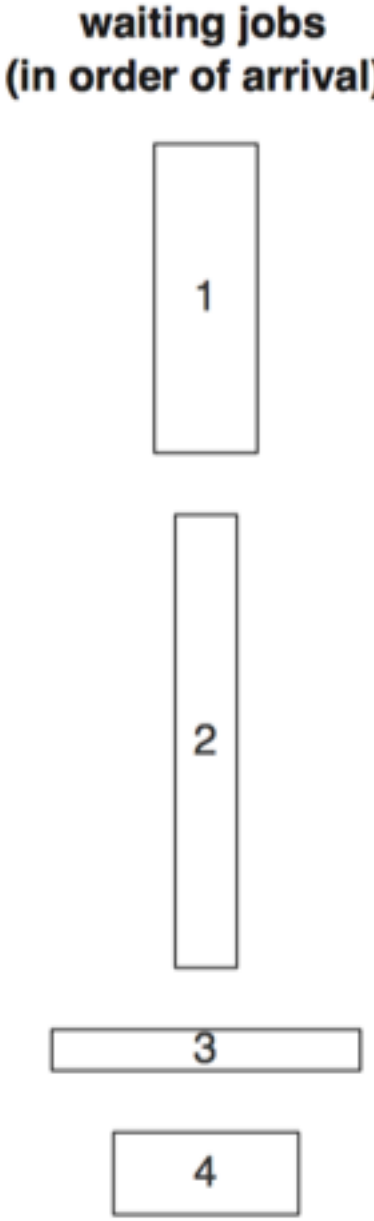
Longest Job First

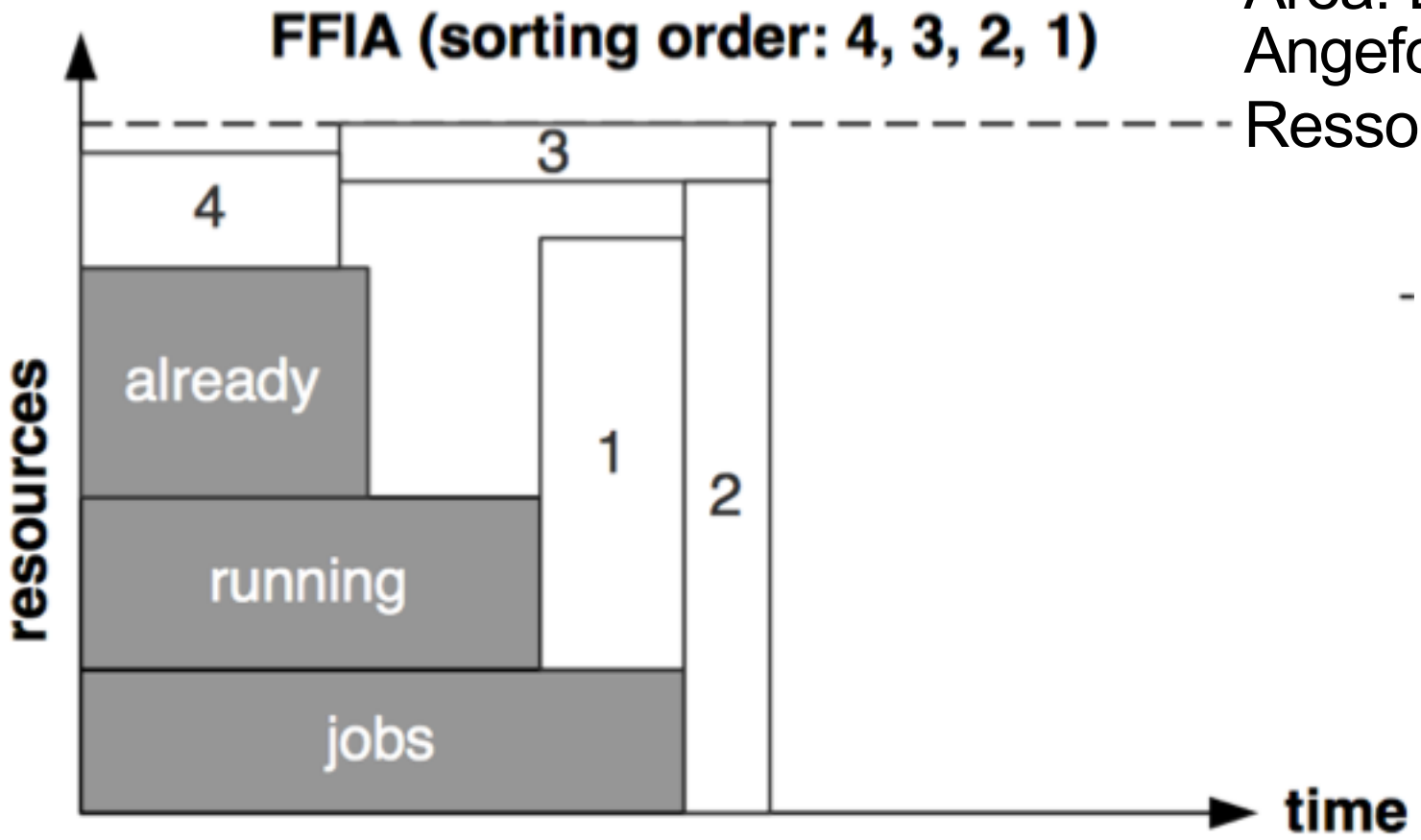




Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

FFIA: First Fit Increasing Area





Area: Laufzeit x Angeforderte Ressourcen

-

Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

Allgemeiner: Sortierungs-Kriterien

- Submit-Zeitpunkt
  - First-Come-First-Serve
  - Last-Come-First-Serve (Unüblich, ... Wäre ein Stack)
- Laufzeit
  - SJF: Shortest Job First (Variante: First Fit Increasing Height = Laufzeit)
  - LJF: Longest Job First (Variante: First Fit Decreasing Height = Laufzeit)
- Fläche = Ressourcen-verbrauch = Laufzeit x angeforderte Ressourcen
  - FFIA: First Fit increasing Area: Abwandlung von SJF, mit Berücksichtigung von Ressourcenverbrauch
- Job-Gewicht
  - ZB. Projekt, oder User-definiertes Gewicht, oder Wartezeit
- Smith-Ratio
  - Job-Gewicht / Fläche
- ...

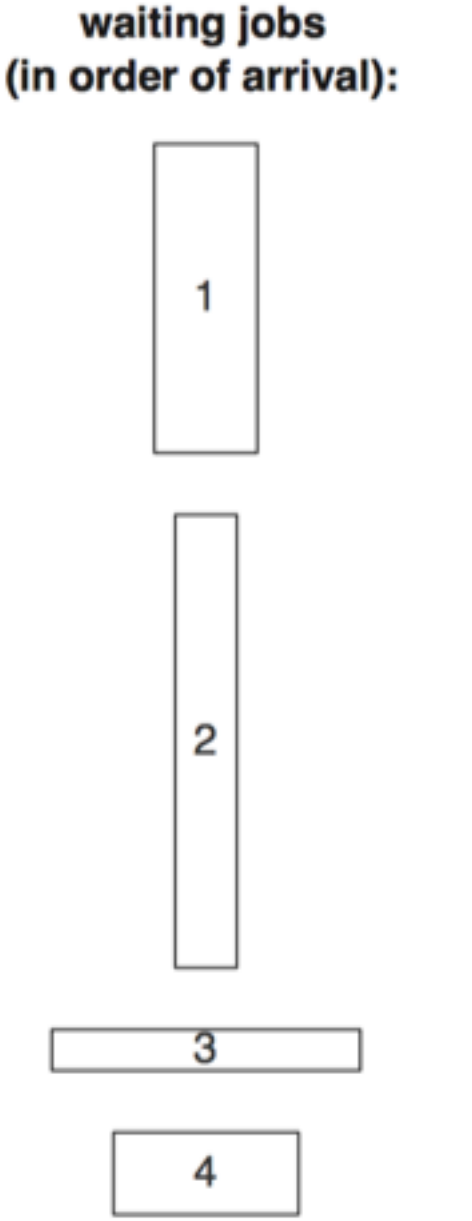
Auswahl-Kriterien

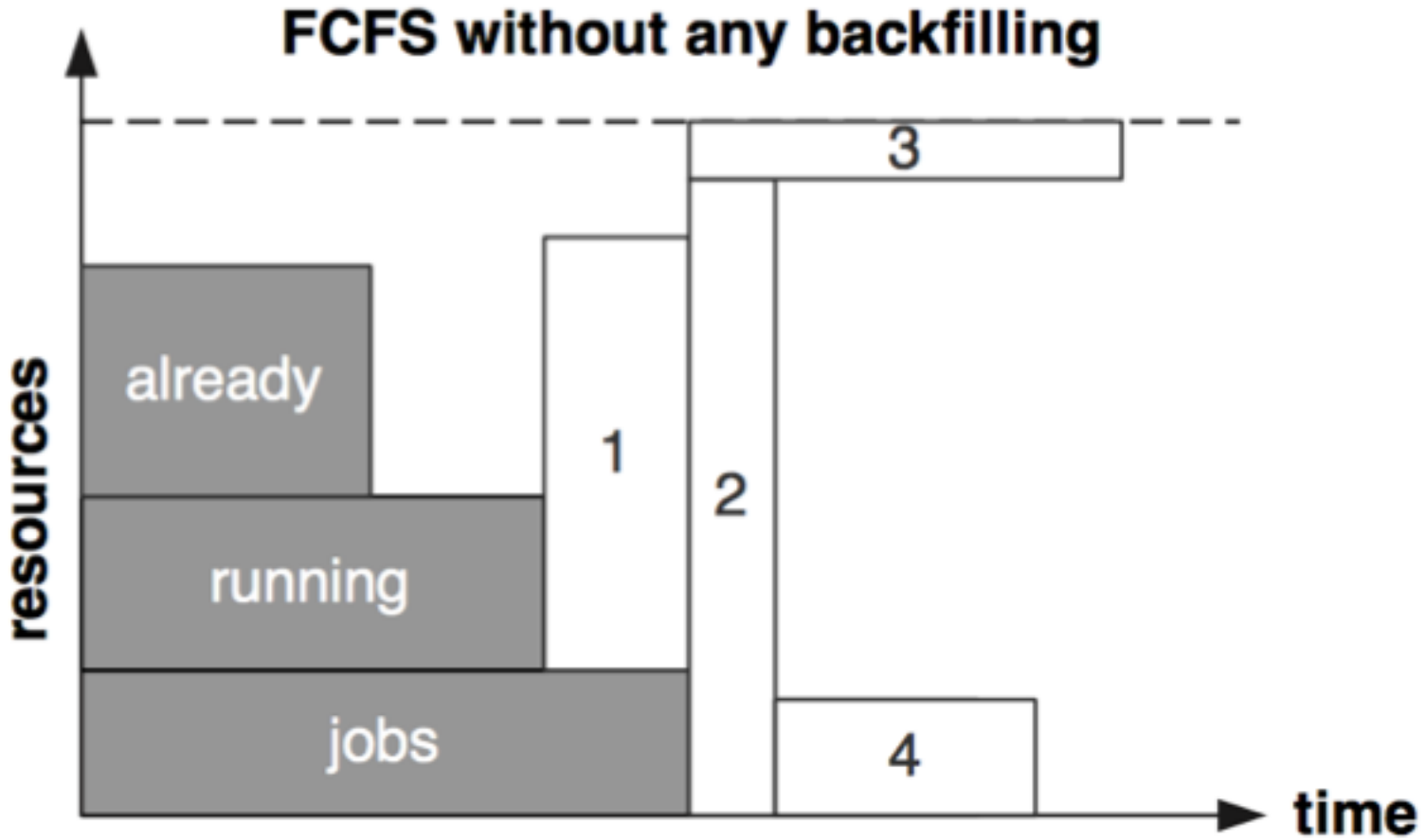
- „Front“: Immer der erste Job wird ausgewählt
  - Macht bei manchen Sortierungen Sinn, zB FCFS
  - Ggbfs. werden die Ressourcen nicht optimal ausgelastet
- „First Fit“: Der erste, passende Job wird ausgewählt
  - Macht bei manchen Sortierungen Sinn, zB FirstFitIncreasingArea
  - Ggbfs. müssen grössere Jobs länger warten
- „Best Fit“: Die ganze Liste wird untersucht, und der am besten passende Job ausgesucht – Bei Gleichstand davon der Erste
  - Beste Auslastung, ggbfs. müssen grössere Jobs länger warten
  - Scheduling-Auffand sehr hoch

Verbesserung: Backfilling

- Idee:
  - Man nehme ein Sortier-Verfahren welches fair und einfach zu verstehen ist, aber eine schlechte Ressourcennutzung bewirkt (Viel Partitionierung bzw. Leerstand)
  - Der Leerstand wird dann mit anderen Jobs befüllt, die deutlich später dran kämen, aber in die Lücken passen würden
- Zwei prinzipielle Arten von Backfilling
  - Konservatives Backfilling: Der Startzeitpunkt der priorisierten Jobs wird nicht geändert
  - EASY Backfilling: Der Startzeitpunkt der priorisierten Jobs kann verzögert werden weil ein Backfill-Job etwas länger braucht und nicht genügend Ressourcen frei sind

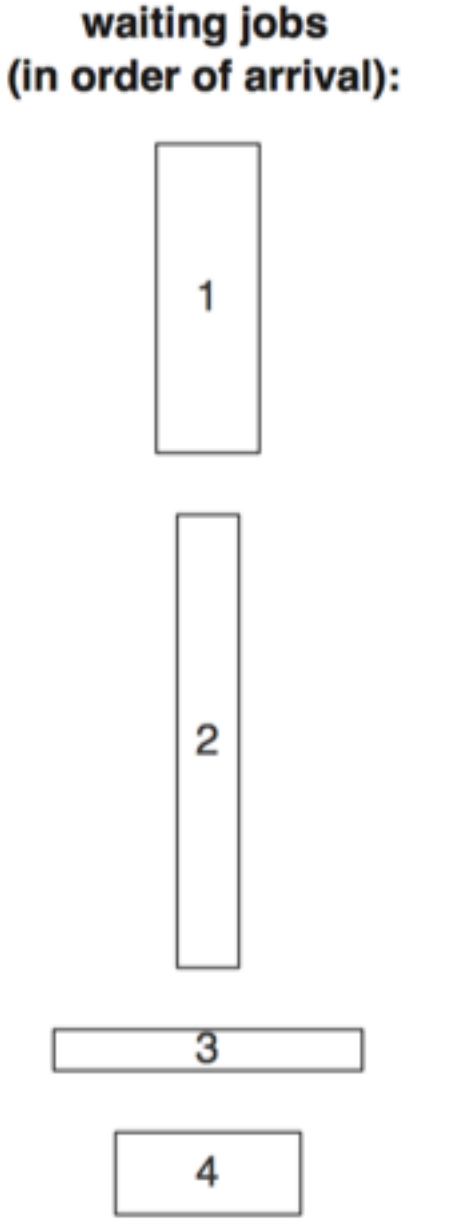
FCFS Ohne Backfilling

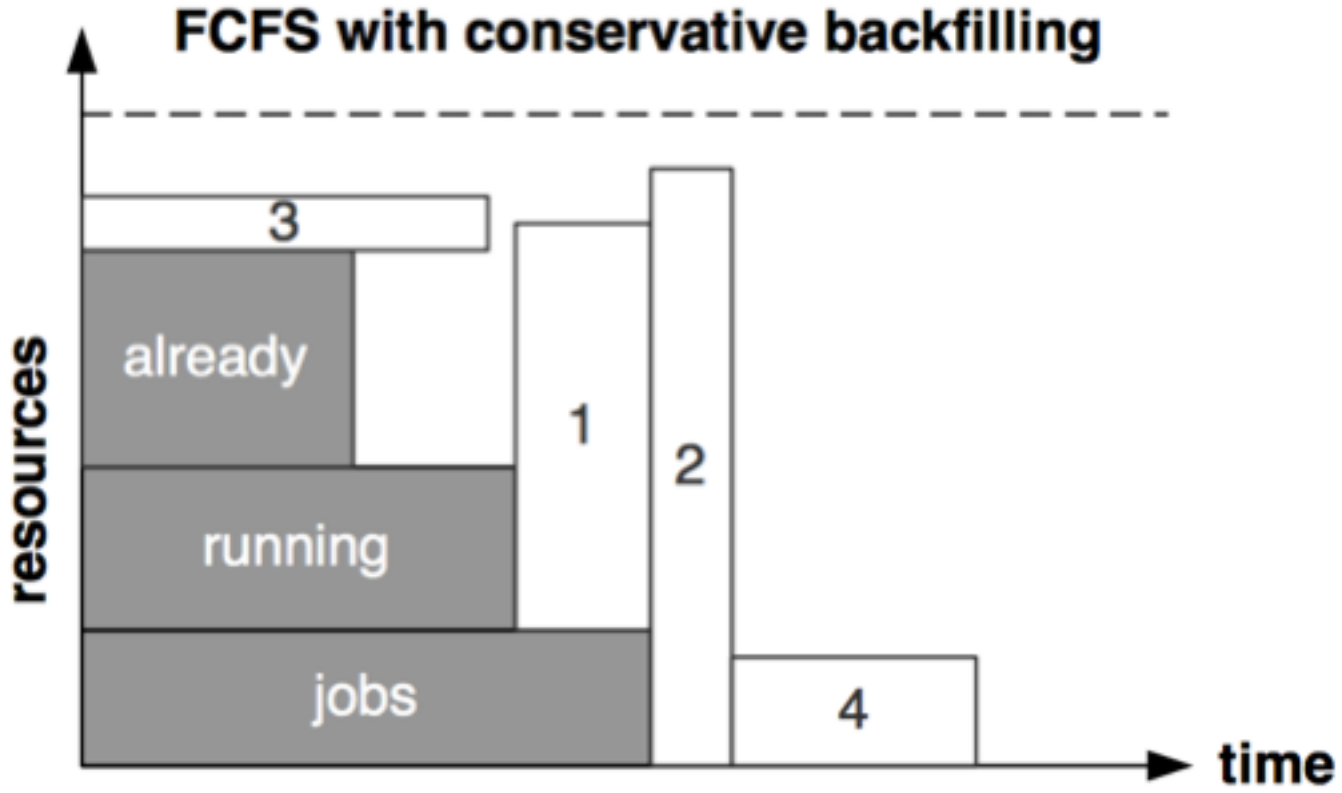




Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

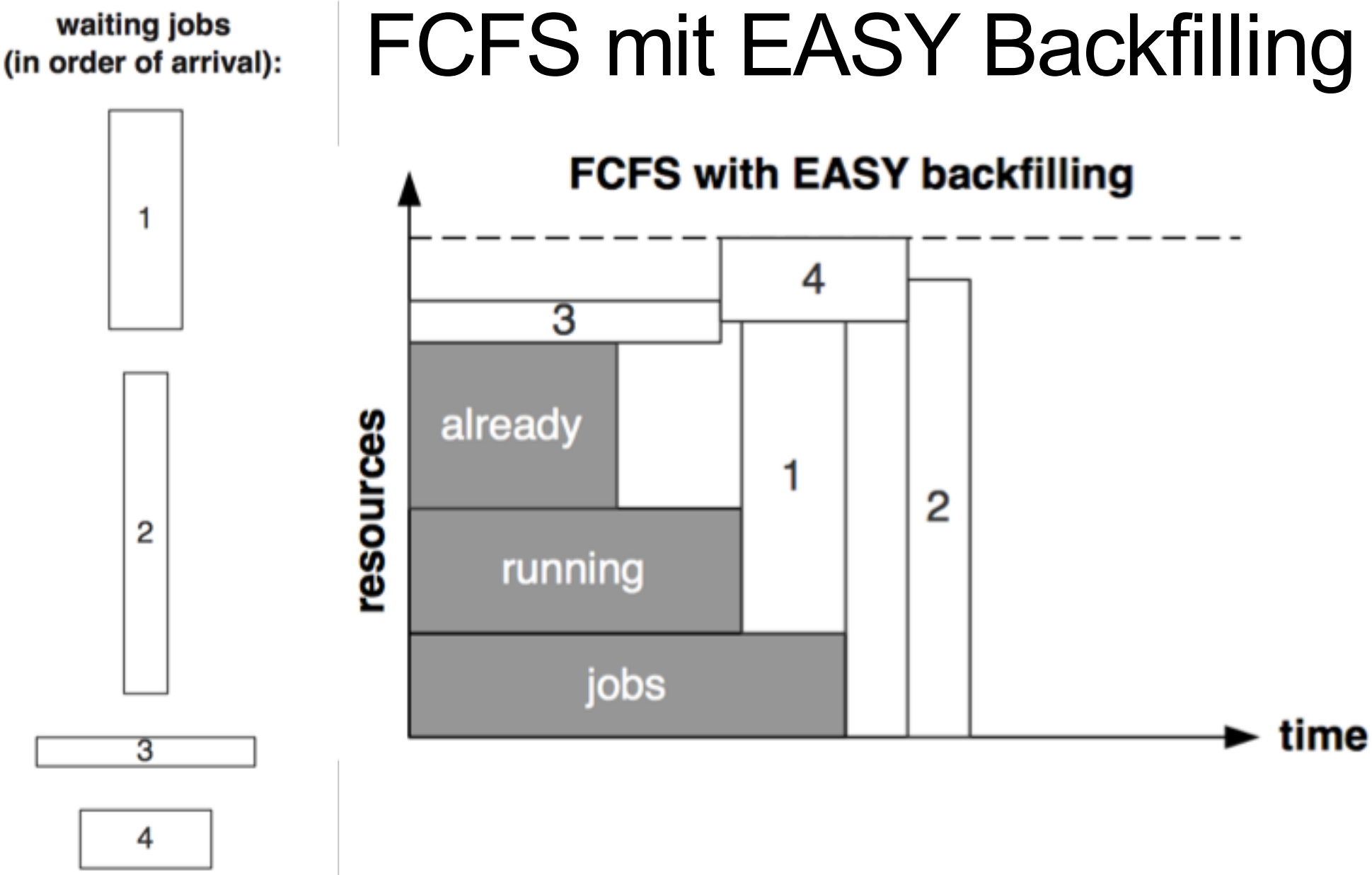
FCFS mit konservativem Backfilling





Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

FCFS mit EASY Backfilling



Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

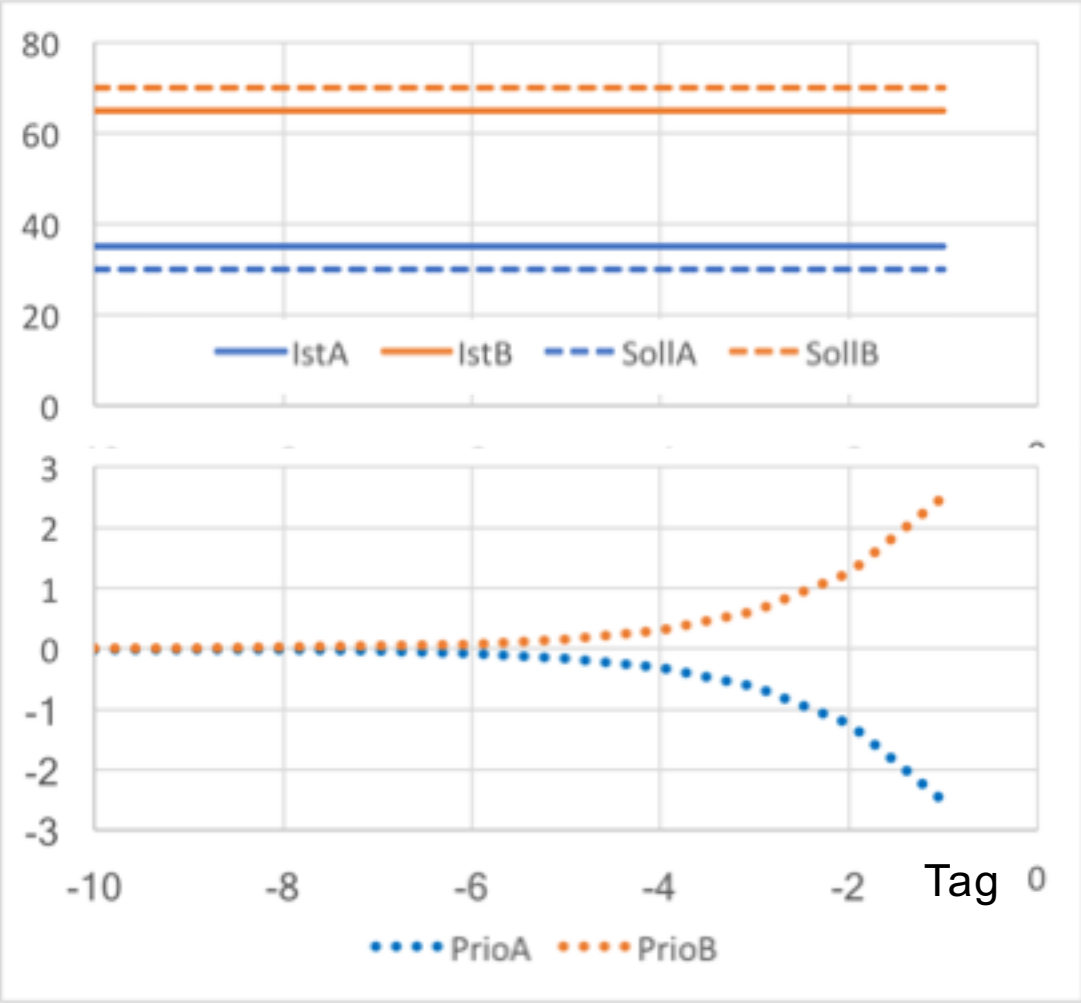
Job-Prozess-Knoten Zuordnung für HPC

- Ein Job kann auf mehrere Knoten aufgeteilt werden
- Typischerweise wird aber jeder Knoten exklusiv von einem Job belegt
- Es wird aktuell vorsichtig mit Co-Location experimentiert, dies sind dann aber zwei oder sehr wenige Jobs die gleichzeitig einem Knoten belegen
  - zB IO-Intensiv und Compute-Intensiv
- Typischerweise gilt aber die exklusive Nutzung eines Knoten durch einen Job, was das Scheduling deutlich vereinfacht
- Im HTC Bereich ist die Grund-Einheit „ein Core“

Fair-Share-Algorithmus

- Oft haben unterschiedliche Gruppen (oder Nutzer, oder Projekte) unterschiedliche Soll-Anteile an einem Cluster
  - zB entsprechend ihrem Anteil an der Beschaffung der Hardware, oder politischen Randbedingungen...
- Der Fair-Share-Algorithmus ist eine Sortier-Methode, die anhand dem Verhalten der Gruppe in der Vergangenheit die Job-Liste so anpasst, dass der Soll-Zustand erreicht werden soll
- Typischerweise gewichtet der Fair-Share-Algorithmus die jüngere Vergangenheit stärker
- Der Fair-Share-Algorithmus kann über mehrere Hierarchie-Ebenen wirken

Beispiel: 2 Gruppen mit gewichtetem Fair-Share



- Zum Tag 0 wird FS eingeschaltet
- Jeder der zehn letzten Tage trägt unterschiedlich zur PrioA bzw. PrioB von Tag 0 bei
- Die effektive Prio ist die Summer über die Tage
  - In diesem Fall
  - EffPrioA = -5
  - EffPrioB = +5

Caveat Fair-Share-Algorithmus

- Funktioniert nur, wenn Jobs in der Queue sind
- Funktioniert nur, wenn Submission Profil ungefähr kontant ist

- Wird typischerweise mit anderen Scheduling-Strategien zusammen verwendet, also mehr-dimensionale Sortierung / Entscheidung
- Führt zu ewig langen Diskussionen

Preemption: Suspenden / Killen von Threads oder Jobs

- Vier Level:
- Keine Preemption: Einmal gestartet wird der Job nicht beeinflusst
- Local Preemption: Einzelne Threads werden preemted, später auf dem gleichen Knoten wieder gestartet
- Migratable Preemption: Einzelne preemted Threads werden später auf anderen Knoten wieder gestartet. Migration der Daten und Anpassen der Konfiguration
- Gang Preemption: Alle aktiven Threads werden gleichzeitig preempted, und später wieder gleichzeitig neu gestartet.

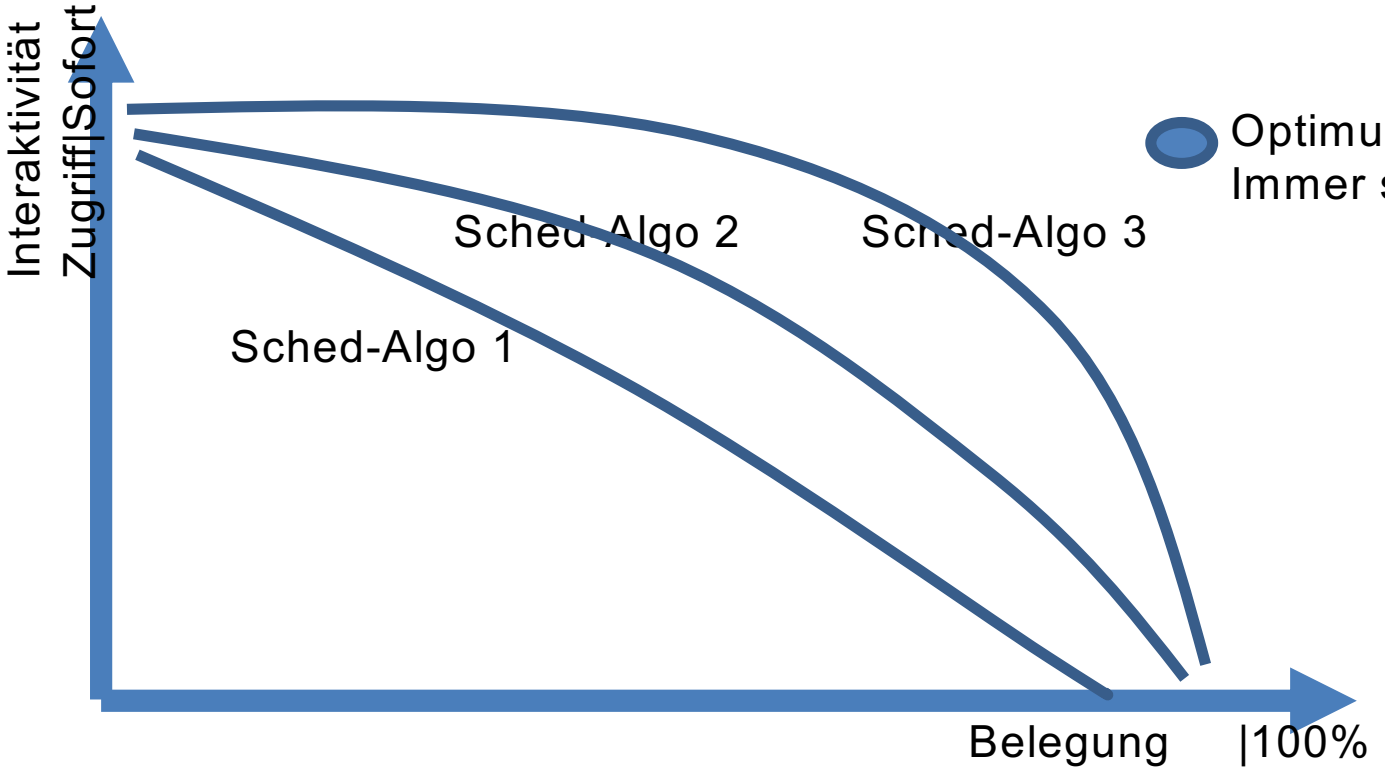
Preemption & Checkpointing

- Typischerweise sendet das Bach-System erstmal “-SIGSTOP(19)“
- Wenn das Programm dieses Signal vernünftig abfängt, kann es seinen aktuellen Status selber sichern, und sich herunterfahren
  - Ohne Behandlung dieses Signals wird der Prozess hier schon beendet
- Nach einer gewissen Zeit wird dann „-SIGTERM (15)“ gesendet
  - Falls das Programm noch nicht beendet wurde
- Checkpointing heisst: Aktive Kommunikation beenden, und den aktuellen Status der Berechnung / Arbeitsspeichers sichern (zB auf Cluster-File-System)

Effizienzen und Optimierungen

- Was ist Effizienz? Wer misst Effizienz?
- Typisches Mass für Job-Effizienz
  - CPU-Time / WALL-Time
- Typisches Mass für Cluster-Effizienz
  - Belegung / #TotCores
- Manche Use-Cases verlangen aber auch
  - Schnellen (interaktiven) Zugriff auf Ressourcen
  - Reservierung (zB für Online-Analysen bei Strahlzeiten)
  - Metrik?

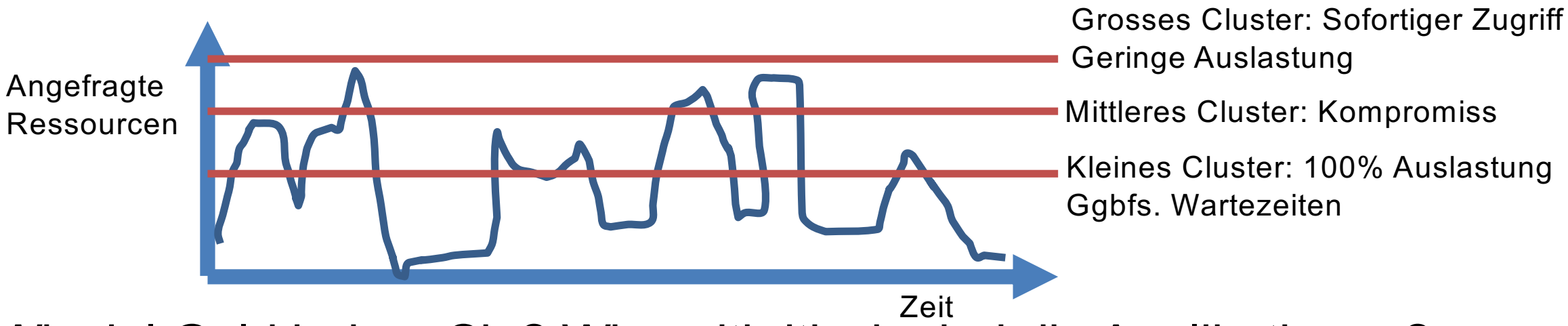
Unterschiedliche Scheduling-Algorithmen



- Das Optimum ist üblicherweise nicht erreichbar
- Verschiedene Scheduling-Algorithmen und Einstellungen können eine bessere Auslastung bewirken
- Typischerweise aber auch sehr viel Psychologie im Spiel

Dimensionierung Cluster

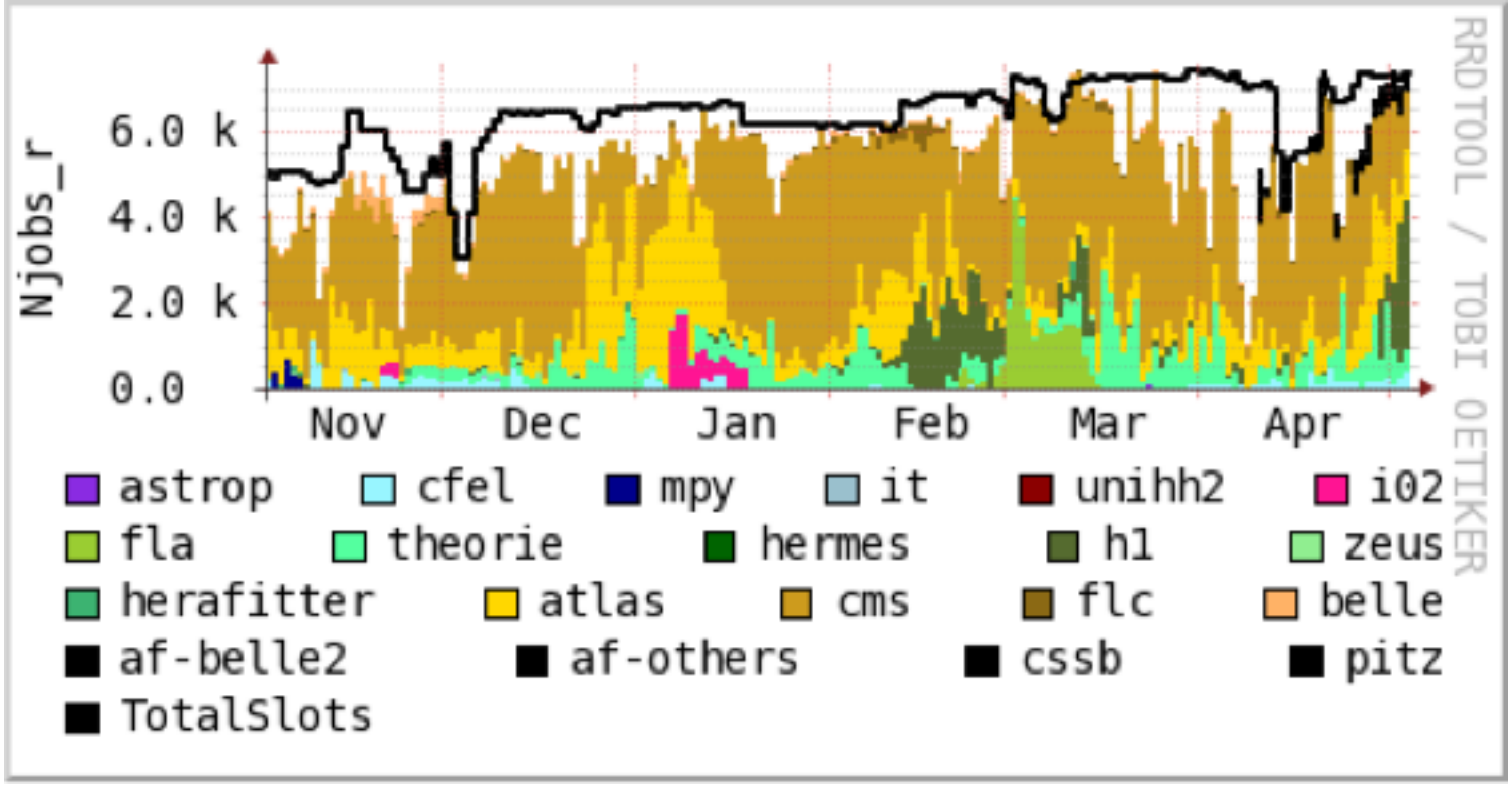
- Bei gegebenem Job-Submission-Profil

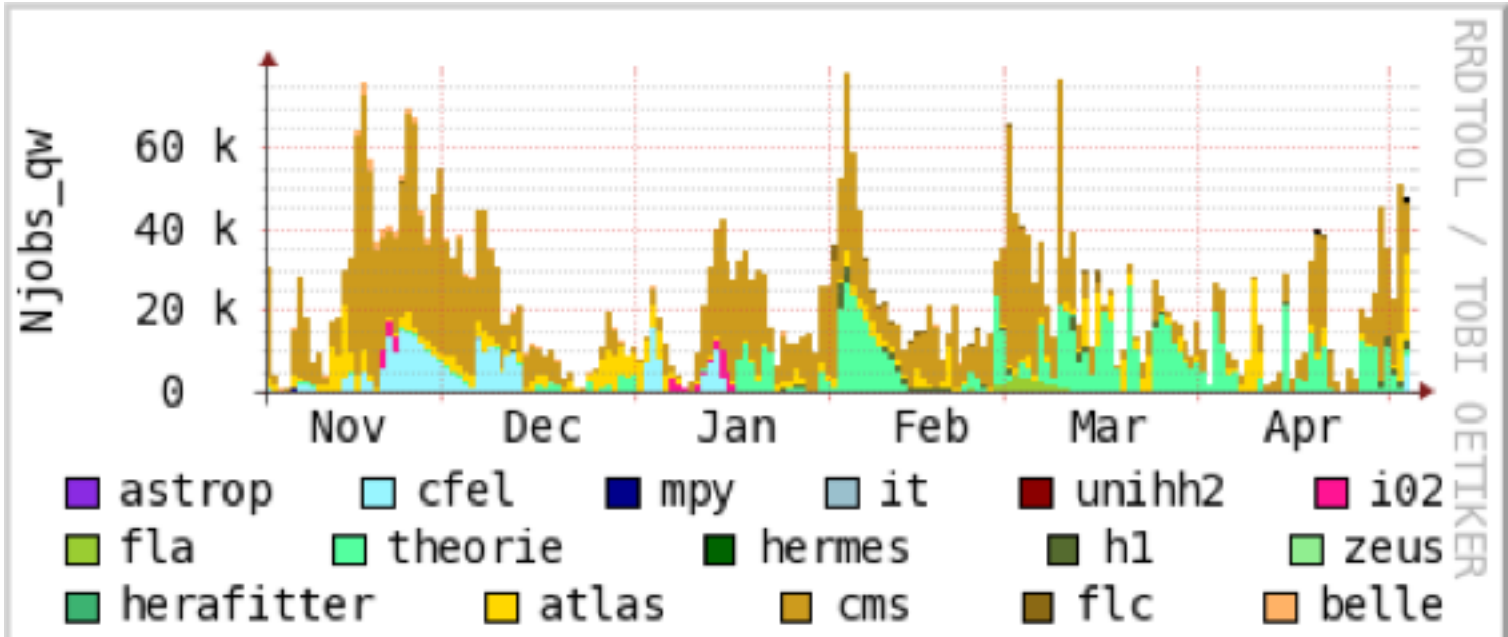


- Wieviel Geld haben Sie? Wie zeitkritisch sind die Applikationen? Wie gut können Sie ihren Nutzern Wartezeiten (und ggbfs. entstehende „Ungerechtigkeiten“ vermitteln)?

Beispiele für Auslastung

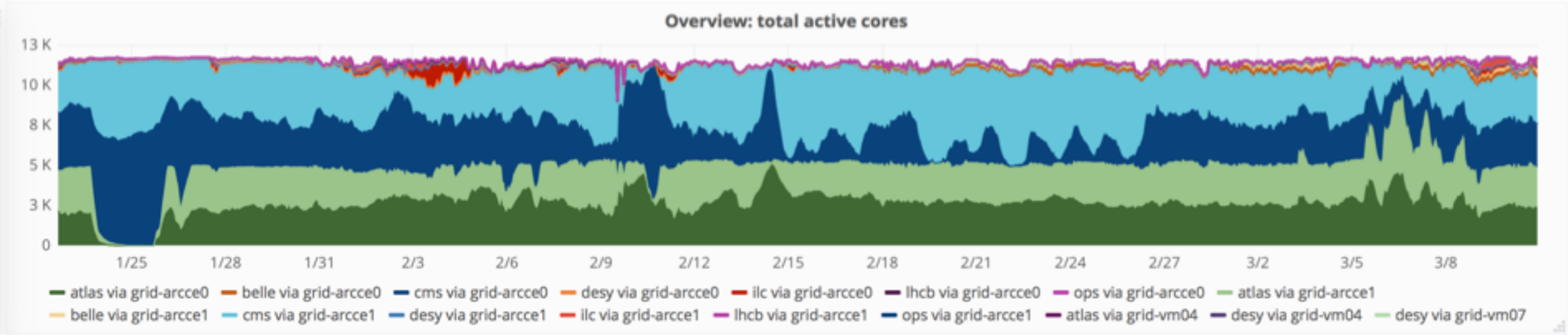
DESY: NAF/BIRD
Viele single-core jobs,
„konferenz-getrieben“





Beispiele für Auslastung

- DESY: SLURM
  - https://grafana.desy.de/dashboard/db/slurm-job-statistics?refresh=1m&orgId=1&from=now-60d&to=now
  - Viele Multi-node Jobs
- DESY: GRID/HTCondor
  - https://grafana.desy.de/dashboard/db/htcondor-cluster-status?orgId=1&from=1485097199252&to=1489180034850
  - Einige-Multi-Core-Jobs, „fabric-artig“



Scheduling im Zeitalter von Cloud

- Die Cloud – unendliche Ressourcen... Wir schreiben das Jahr 2006. Dies ist die Geschichte des ersten kommerziellen Cloud-Anbieters, der mit enormer Rechenleistung unterwegs ist um neue Formen von Computing zu erforschen. Viele Rechenzentren von dem was wir kennen entfernt, dringen Cloud-Anbieter in Grössenordnungen vor, die nie ein Mensch zuvor gesehen hat.

http://s3.amazonaws.com/trektv/trektv/ttv_47/trek_tv_47.jpg

Das Cloud-Versprechen: Genug Rechenleistung

- Die Cloud-Anbieter können schneller ihre Rechenzentren skalieren als dass die Nutzung zunimmt
- Durch die enorme Grösse der Ressourcen mitteln sich Peaks weg
- Braucht man denn noch Scheduling und Queueing und Batch-Systeme?
- Ja:
  - Einmal werden auch HPC Systeme nicht komplett in kommerziellen Clouds verschwinden
  - Dann braucht man auch in Clouds ein Job Management System
  - Kommerzielle Cloud-Anbieter haben einen weiteren Steuerungs-Faktor: Den Preis im Spot-Market, und dazugehöriges „Preemption“.