HPC-Systeme
Management Tools:
Schedulers und Batch Queueing Systeme

Prof. Dr. Volker Gülzow

Dr. Yves Kemp

SS 2017

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

Aus: E-Mail Signratur 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.

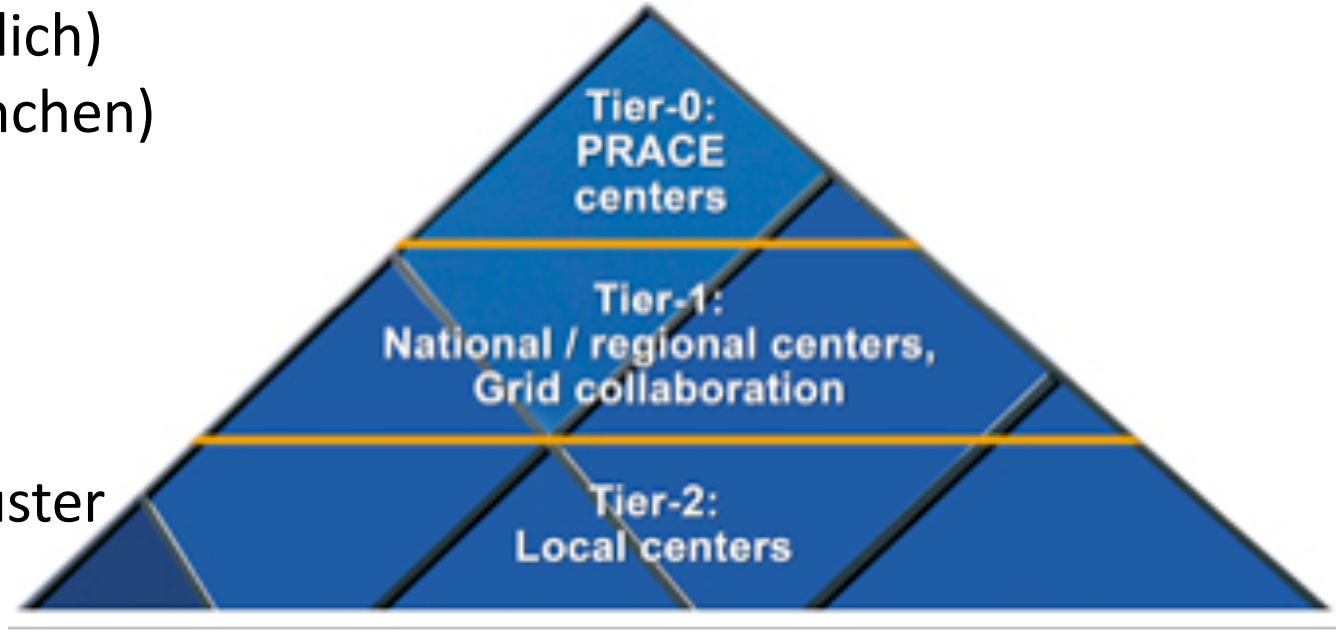
http://www.gauss-centre.eu/gauss-centre/EN/HPCservices/HowToApply/LargeScaleProjects/largeScaleJUQUEEN.html

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

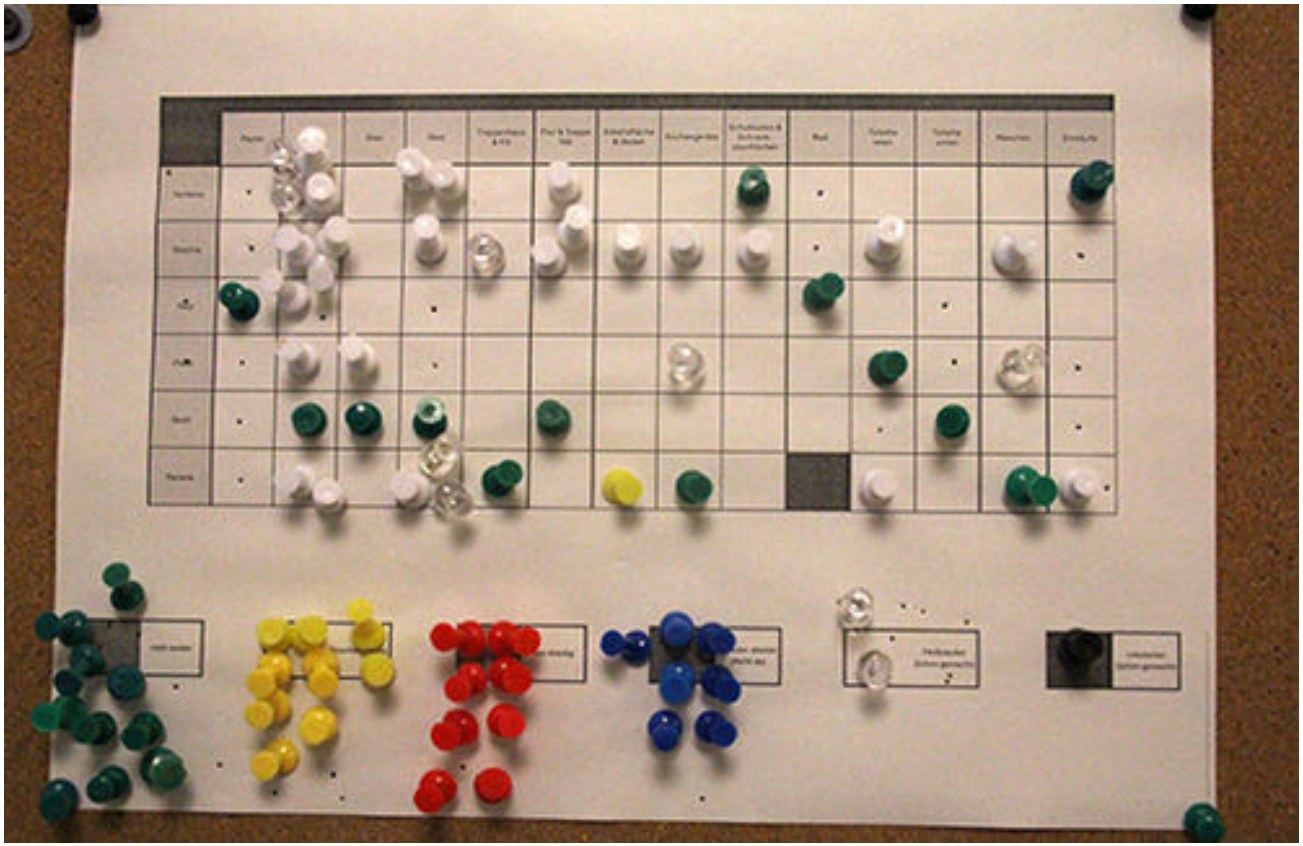
```
> cat steer.list
server1 23
server2 42
...

> 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

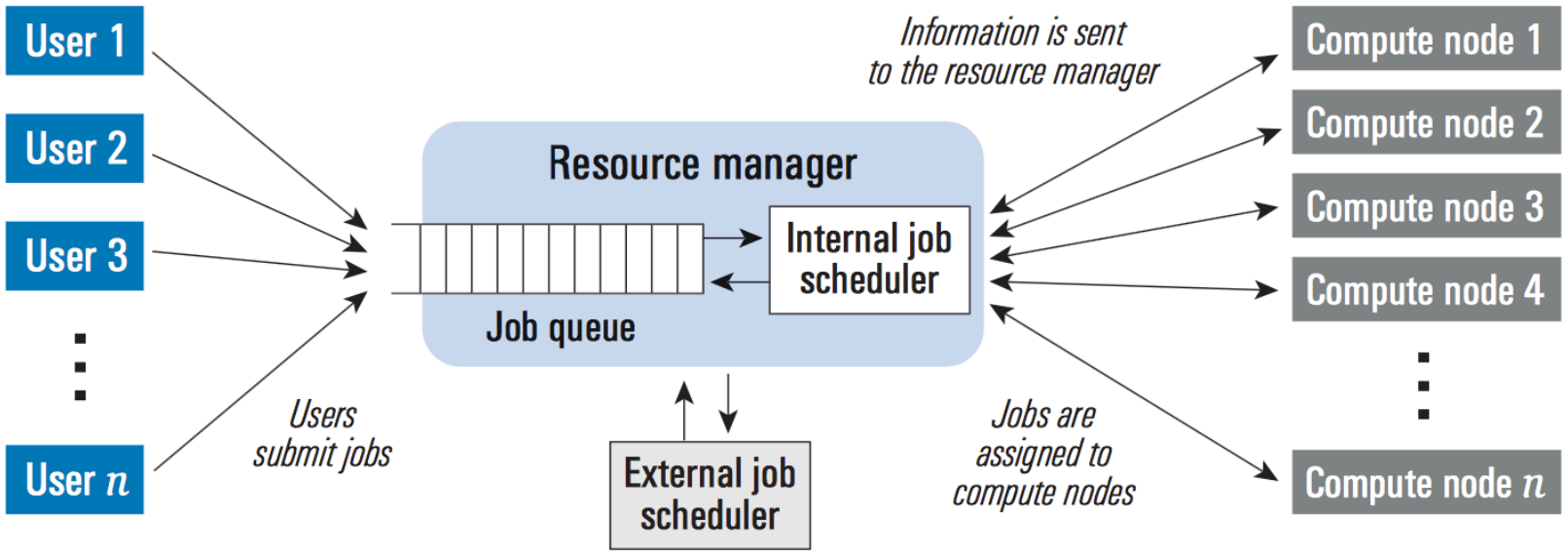
- Vorteile:
  - Kalender ist einfach zu bedienen
  - Jobs sind einfach per ssh abzuschicken, kein Anpassen der Software notwendig
  - Interaktivität ist gewährleistet
  - Feste Reservierungen (zB für Strahlzeiten) möglich
- Nachteile:
  - Sehr statisch, kann nicht auf variierende Laufzeiten eingehen
  - 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 Produkte (persönliche Auswahl)

- Platform Load Sharing Facility (LSF), kommerzielles Produkt von IBM
- UnivaGrid Engine (SunGridEngine, danach OracleGridEngine)
  - Kommerzielles Produkt der Firma Univa
  - Einige Forks aus der quelloffenen Zeit von SUN … allerdings nicht sonderlich aktiv
- TORQUE/Maui
  - TORQUE: Batch System (OpenSource), fork von PBS
  - Maui: Scheduler:(OpenSource), Entwickelt von Adaptive
- PBSpro: Kommerzielle Weiterentwicklung von PBS, Altair
- Moab: Kommerzielle Weiterentwicklung von Maui, Adaptive
- SLURM: OpenSource, Enwicktlung und kommerzieller Support durch SchedMD
- HTCondor: OpenSource, University of Wisconsin

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

Eine kleine Leidensgeschichte vom DESY

- Vor 2000: Setup von kleineren Farmen (~100 Server) mit LSF und anderen Systemen
- 2004: Maui+Torque für Grid Cluster (HTC)
- 2006: SunGridEngine für BIRD (HTC)
- 2007: SunGridEngine für NAF (HTC)
- 2010: Erstes HPC Cluster, Kalender-Reservierung
- 2012: SonOfGridEngine für NAF2 / BIRD
- 2013: Torque+Eigenentwicklung für Grid Cluster
- 2015: HTCondor für Grid Cluster
- 2016: SLURM für HPC Cluster
- 2017: Geplant: HTCondor für NAF2 / BIRD + Zusammenlegung mit Grid

Scheduling Verfahren

- Queueing Systeme
  - Verschiedenen Queue (Warteschlangen) existieren, zB für verschiedenen Nutzer, Projekte, QoS, Resources-Anforderungen, ...
  - Innerhalb einer Queue werden die Requests sortiert (zB FIFO, FairShare, Wartezeit, ...)
  - Der nächste (passende) Job innerhalb einer sortierten Queue wird einer frei gewordenen Ressource zugewiesen
- Planning Systeme
  - Jobs müssen die Laufzeit spezifizieren.
  - Bei der Submission wird vom Planning System, anhand des aktuellen Planes, der Start-Zeitpunkt und die Ressource zugewiesen

Unterschiede der beiden prinzipiellen Verfahren

	queuing system	**planning system**
planned time frame	present	present and future
submission of resource requests	insert in queues	replanning
assignment of proposed start time	no	all requests
runtime estimates	not necessary[1]	mandatory
reservations	not possible	yes, trivial
backfilling	optional	yes, implicit
examples	PBS, NQE/NQS, LL	CCS, Maui Scheduler[2]

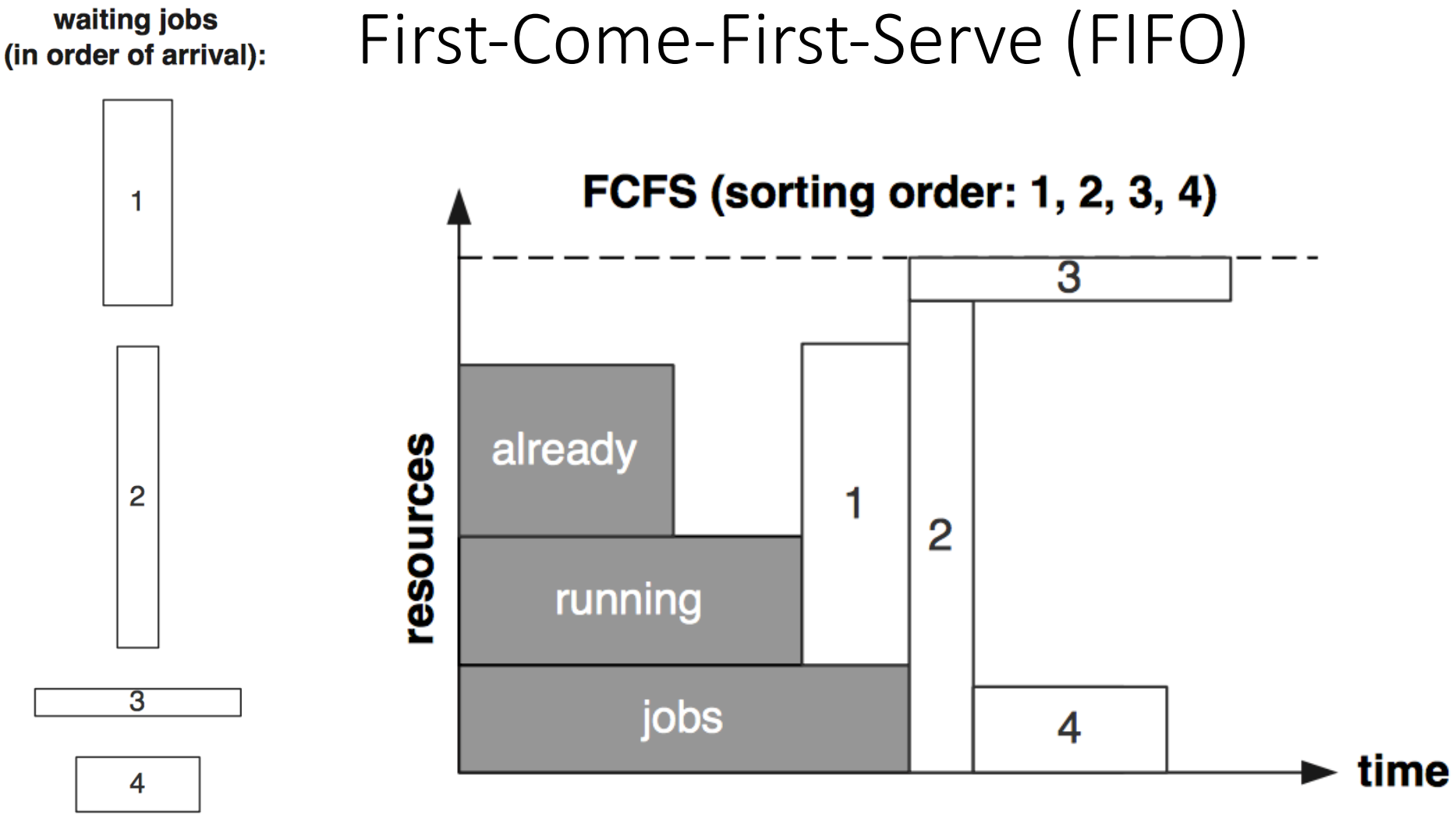
[1] exception: backfilling

[2] According to [15] Maui may be configured to operate like a planning system.

Die meisten heutigen Systeme sind queueing Systeme, allerdings mit Elementen von planning Systemen.
Jetzt schauen wir uns ein paar Algorithmen im Detail an

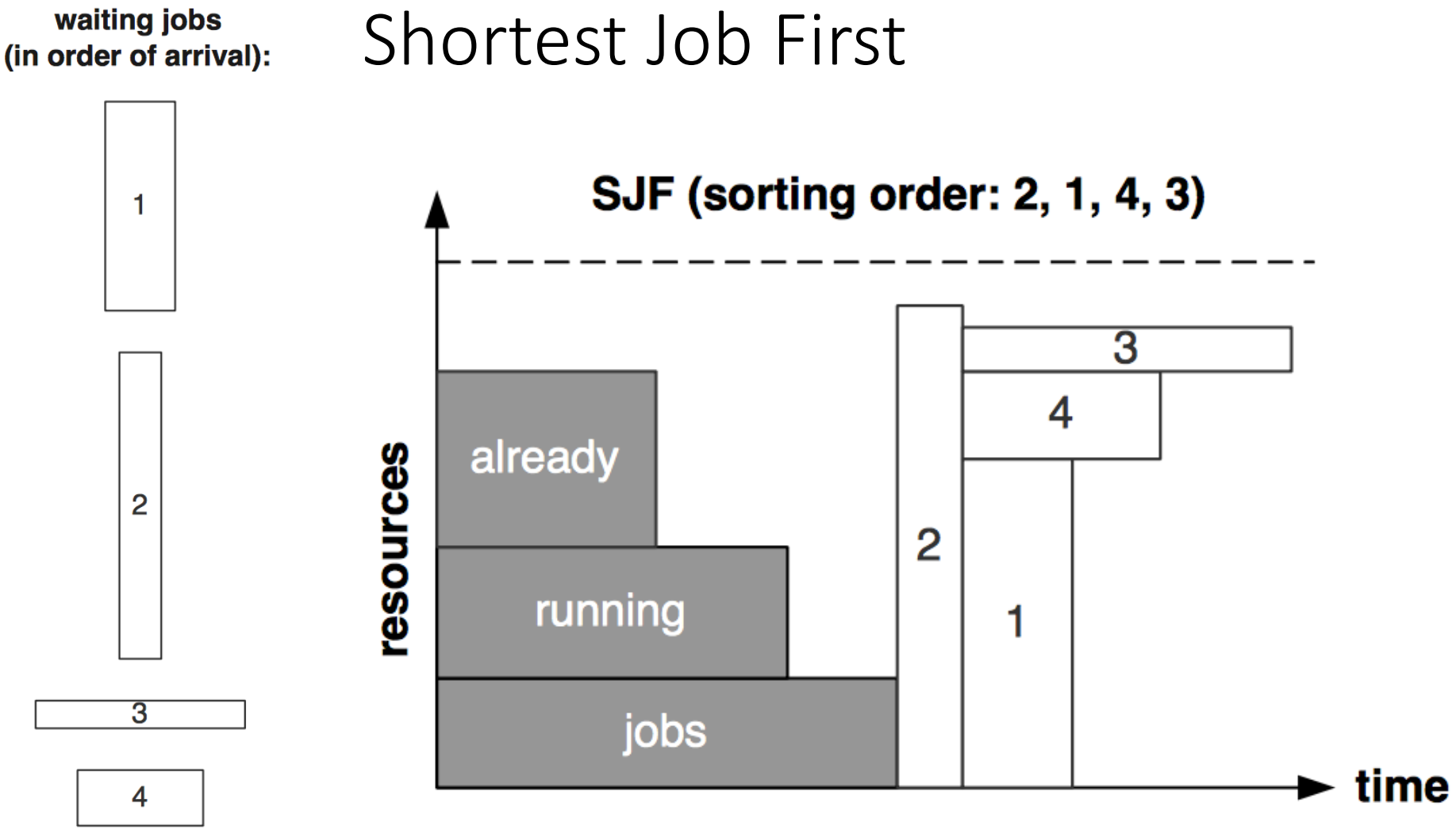
Quelle: http://www.cs.huji.ac.il/~feit/parsched/jsspp03/p-03-1.pdf

First-Come-First-Serve (FIFO)



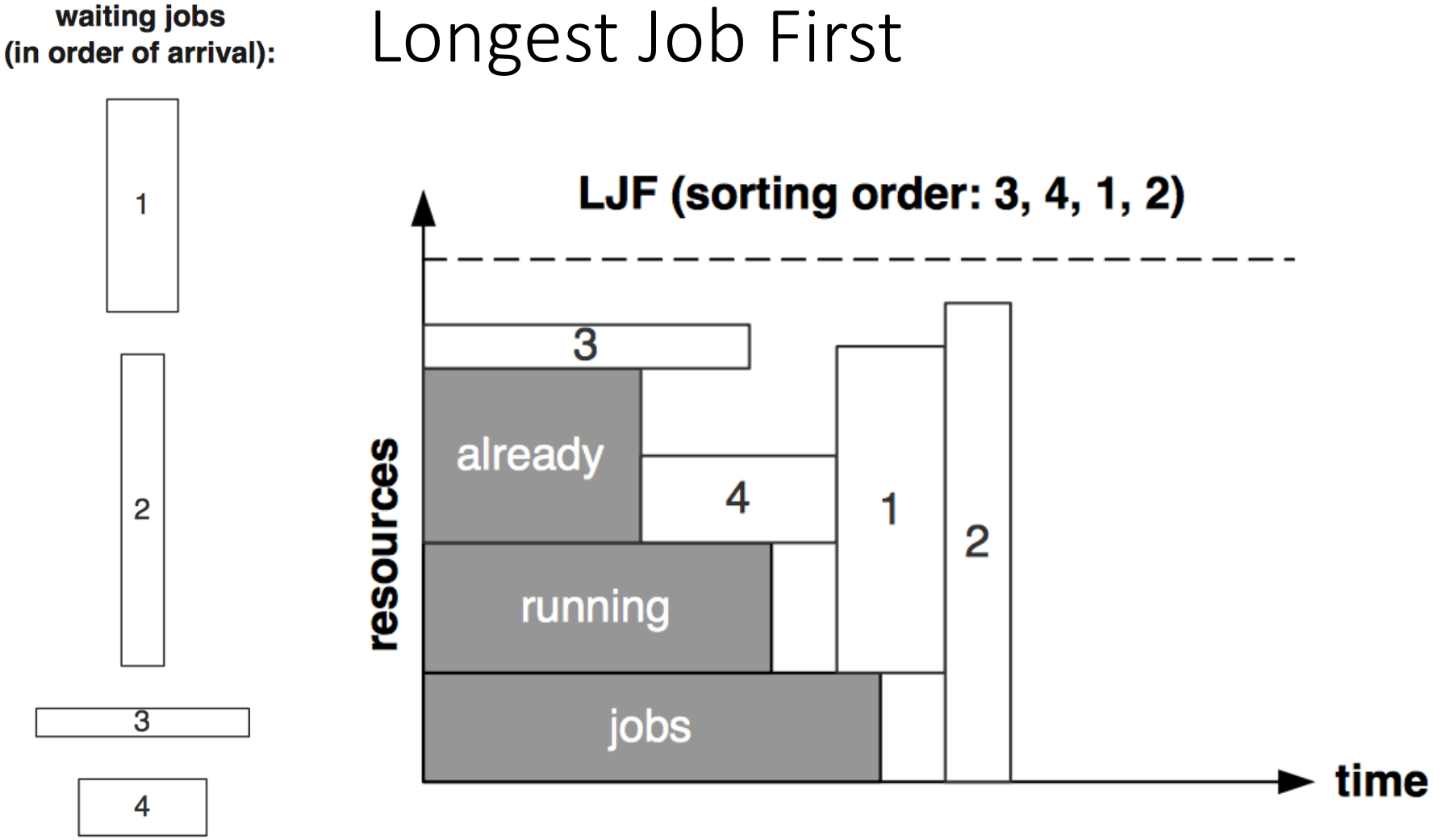
Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

Shortest Job First



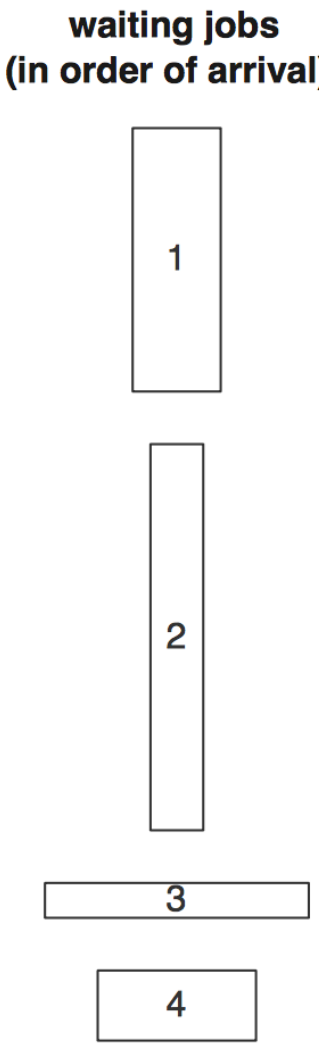
Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

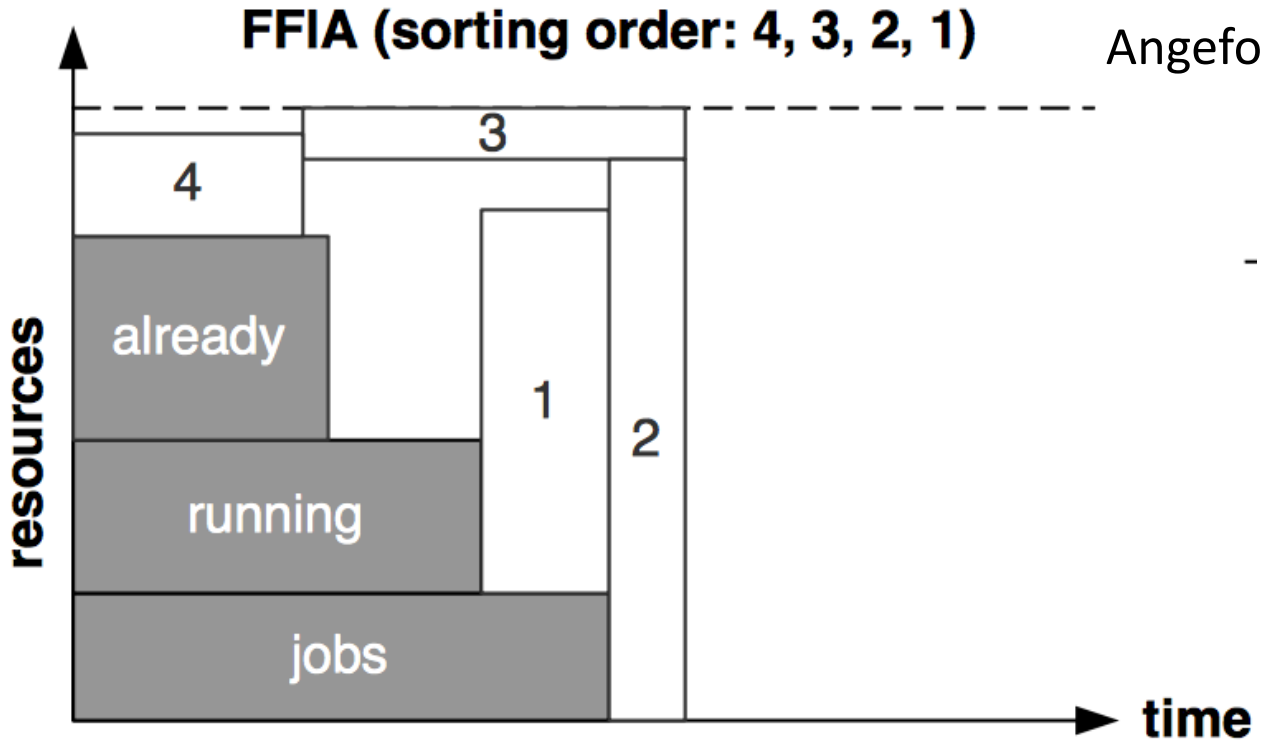
Longest Job First



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
- ...

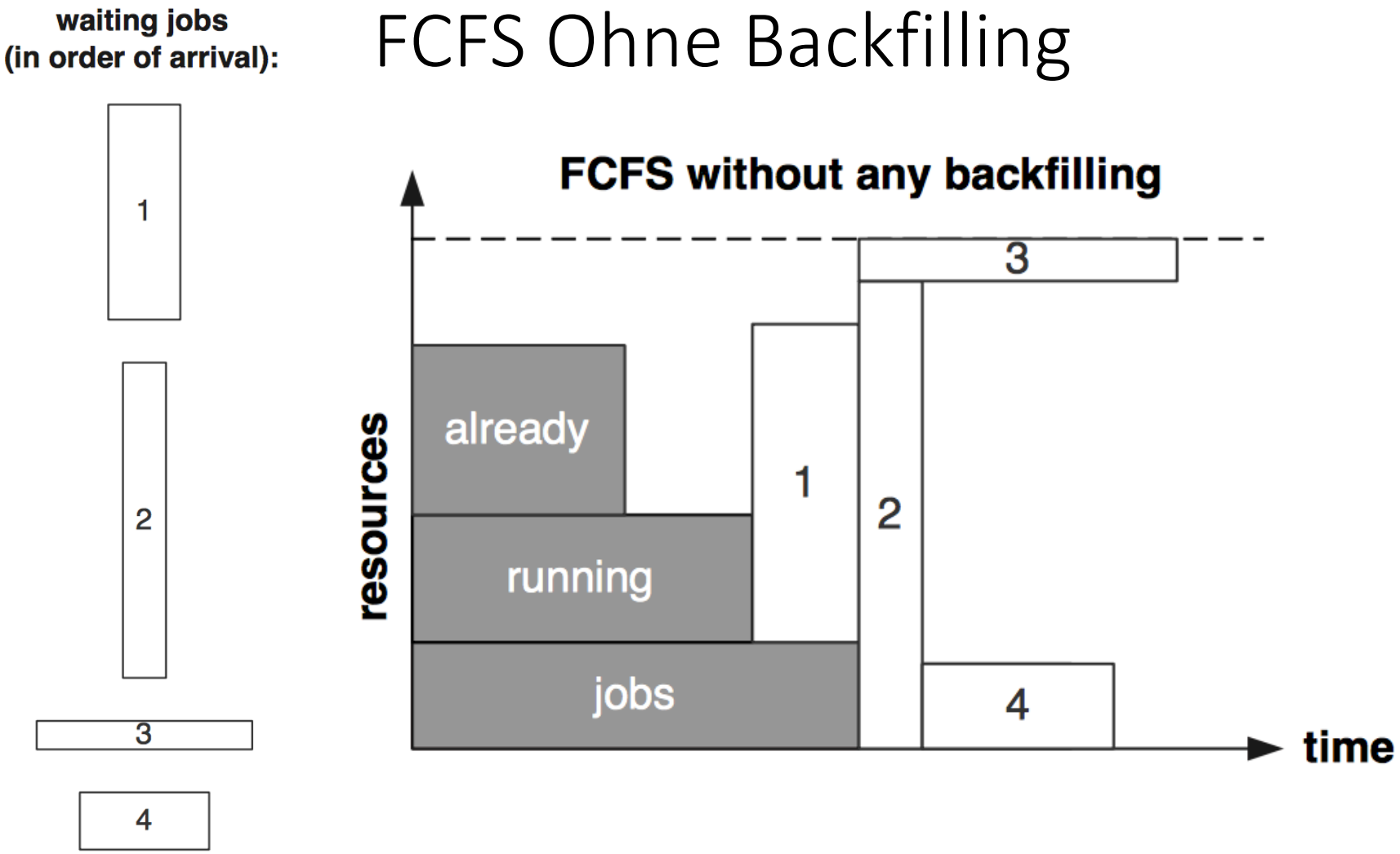
Auswahl-Kriterien

- „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

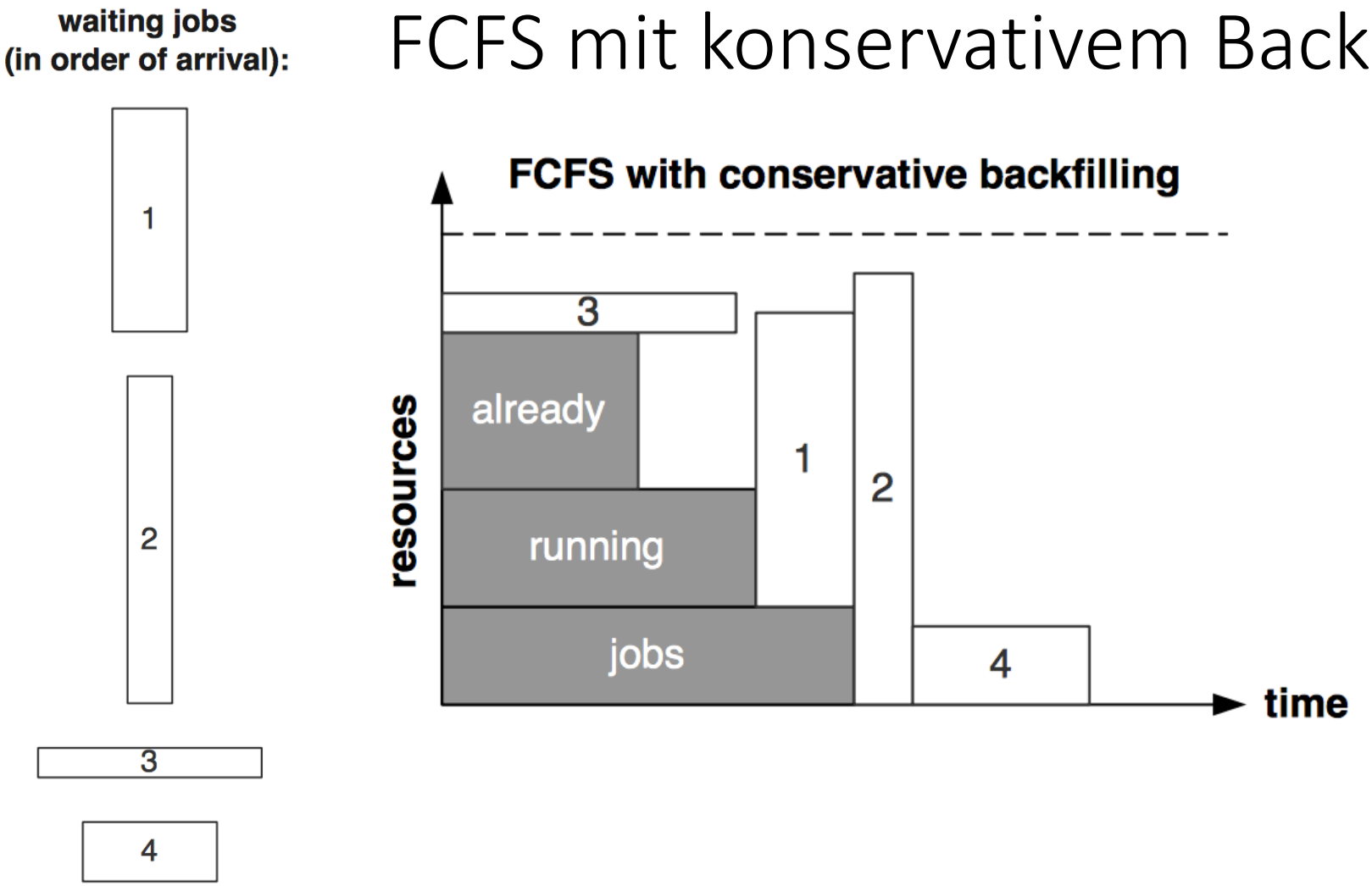
- Idee:
  - Man nehme ein Sortier-Verfahren welches fair und einfach zu verstehen ist, aber eine schlechte Ressourcennutzung bewirkt (Viel Partitionierung bzw. Leerstand)
  - Der Leerstand wird dann mit anderen Jobs befüllt, die deutlich später dran kämen, aber in die Lücken passen würden
- Zwei prinzipielle 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



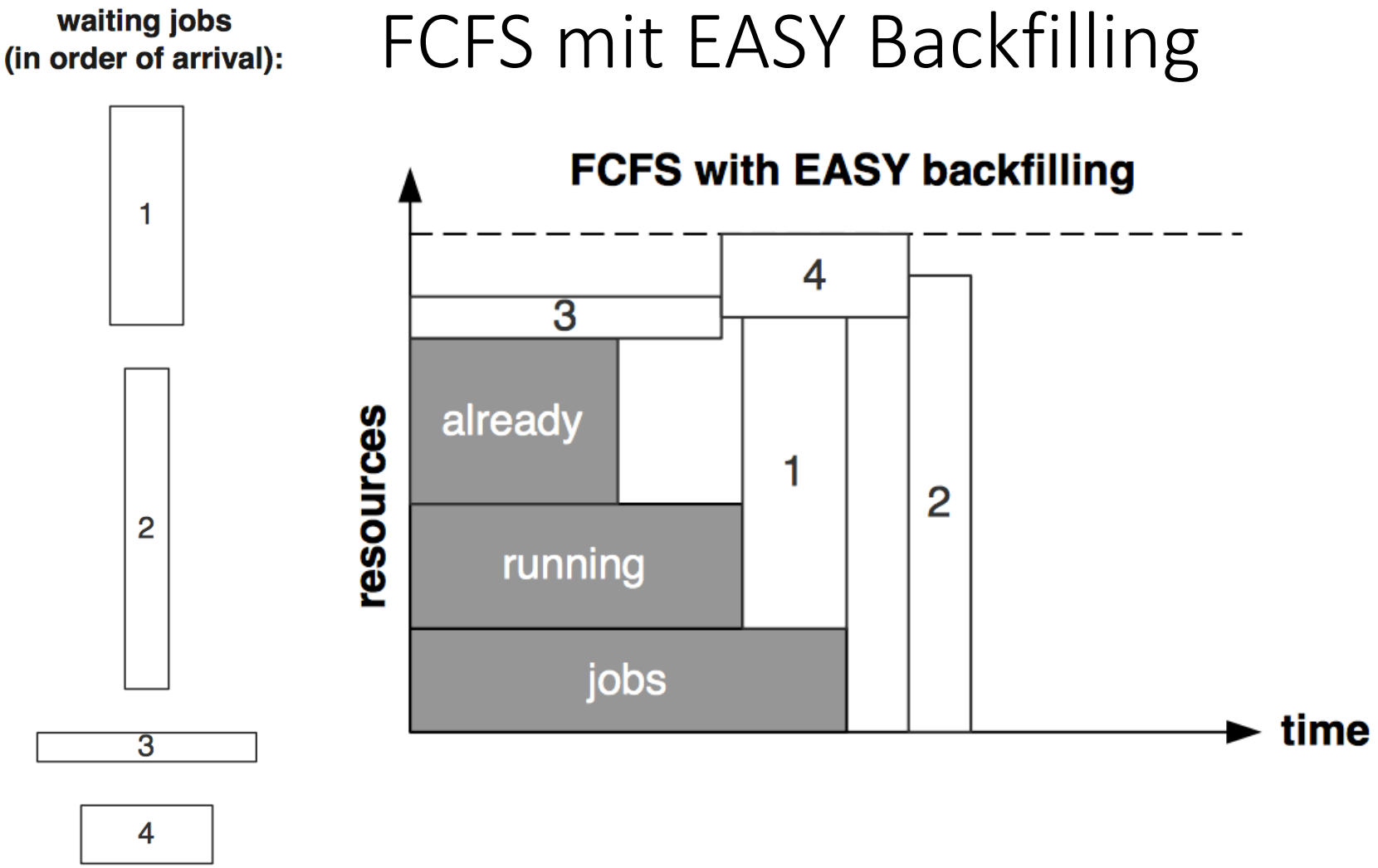
Self-Tuning Job Scheduling Strategies for the Resource Management of HPC Systems and Computational Grids, A. Streit,

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,

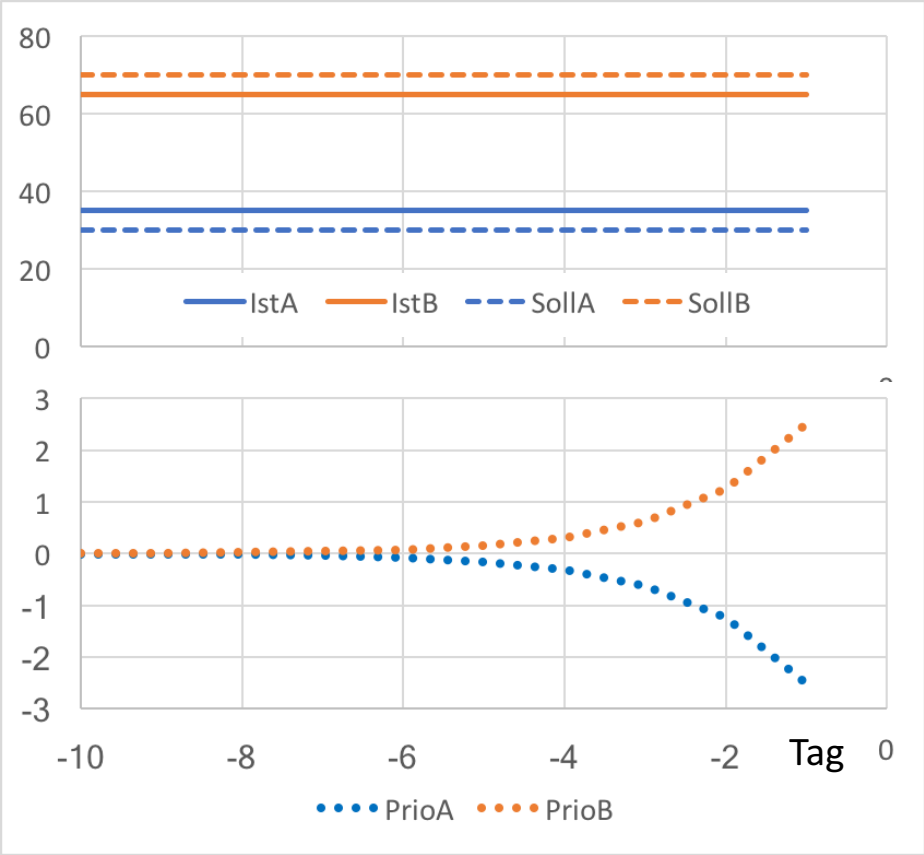
Job-Prozess-Knoten Zuordnung für HPC

- Ein Job kann auf mehrere Knoten aufgeteilt werden
- Typischerweise wird aber jeder Knoten eklusiv 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 Randbedinungen...
- 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.

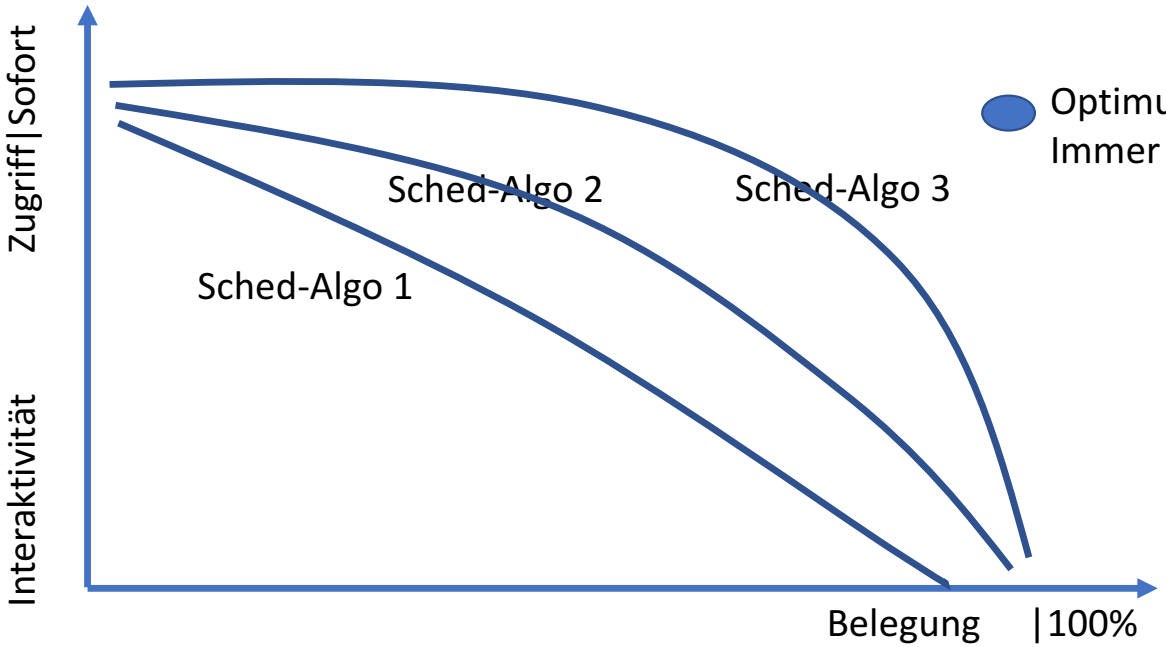
Preemption & Checkpointing

- Typischerweise sendet das Bach-System erstmal “-SIGSTOP(19)“
- Wenn das Programm dieses Signal venünftig 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 Berechung / 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?

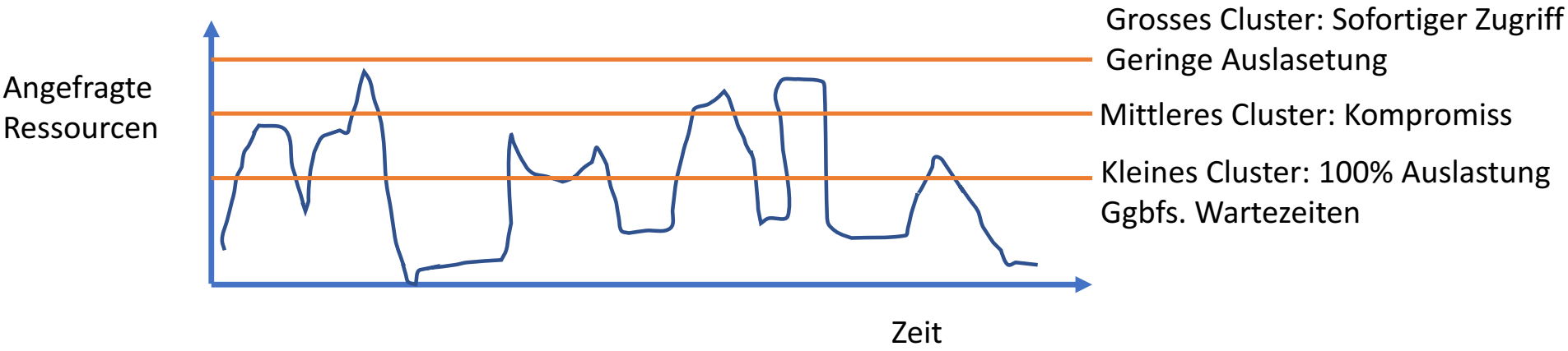
Unterschiedliche Scheduling-Algorithmen



- 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

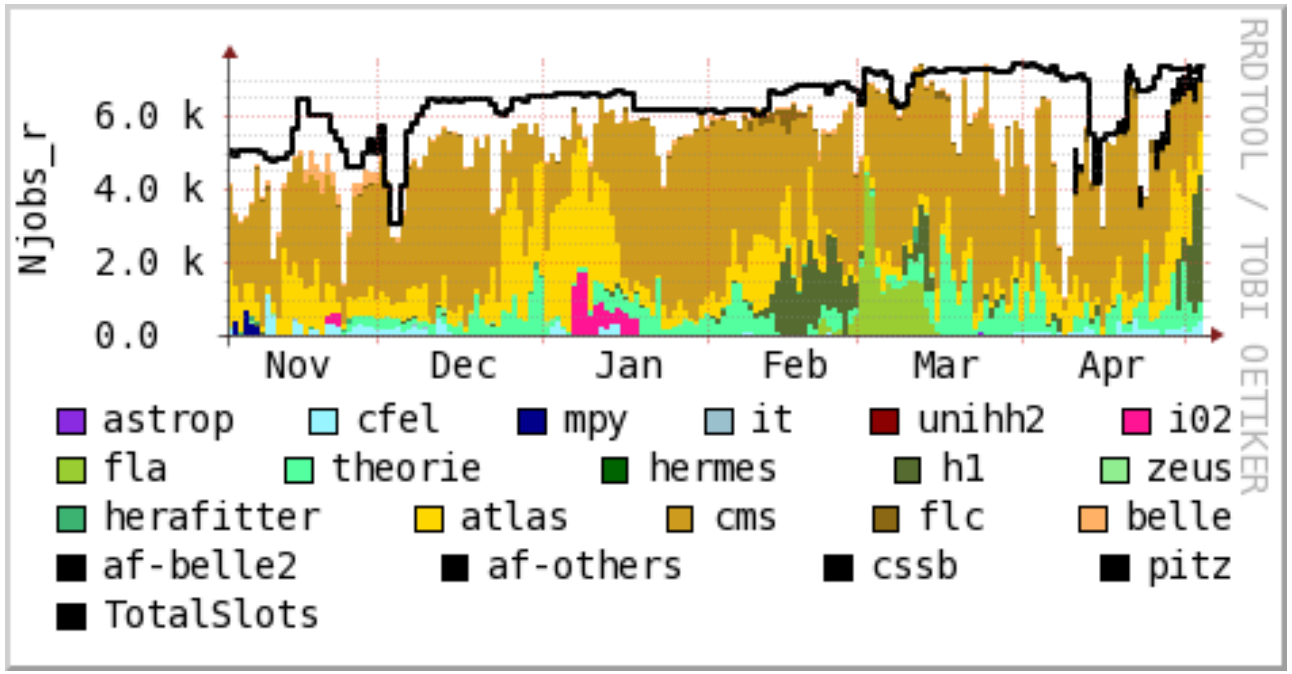
- Bei gegebenem Job-Submission-Profil

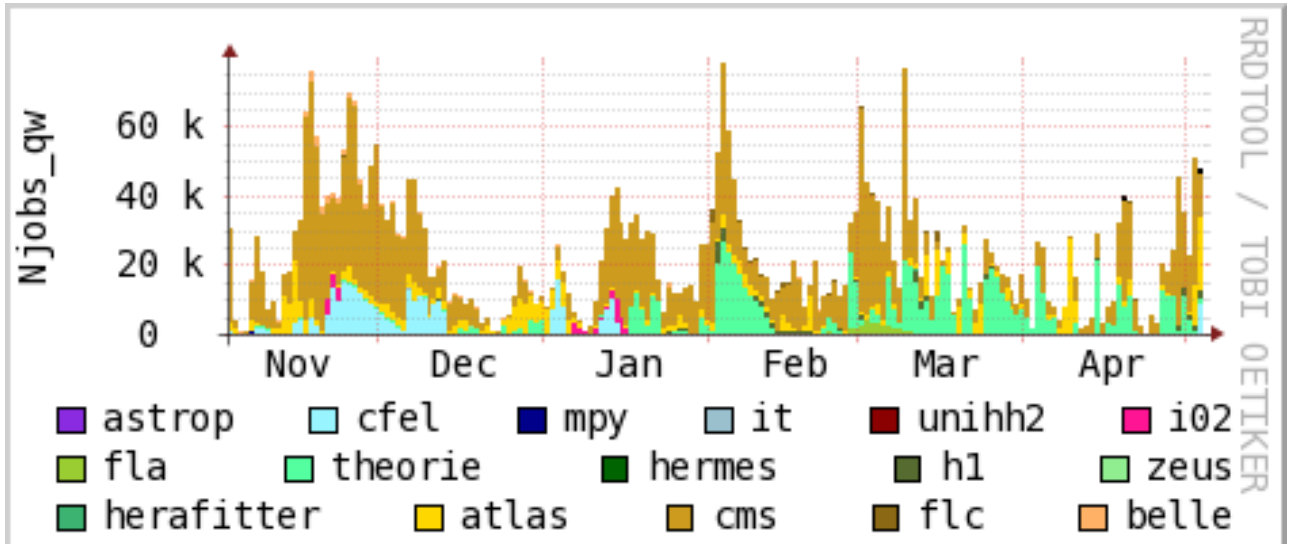


- Wieviel Geld haben Sie? Wie zeitkritisch sind die Applikationen? Wie gut können Sie ihren Nutzern Wartezeiten (und ggbfs. entstehende „Ungerechtigkeiten“ vermitteln)?

Beispiele für Auslastung

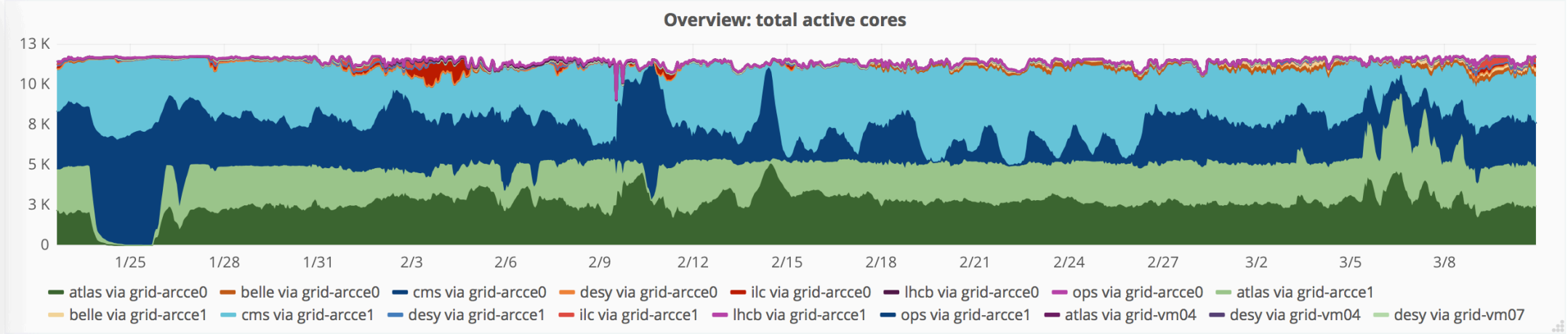
DESY: NAF/BIRD
Viele single-core jobs,
„konferenz-getrieben“





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 Sport-Market, und dazugehöriges „Preemption“.