Autonomic Computing im Grid

Michael Neumann

mneumann@ntecs.de

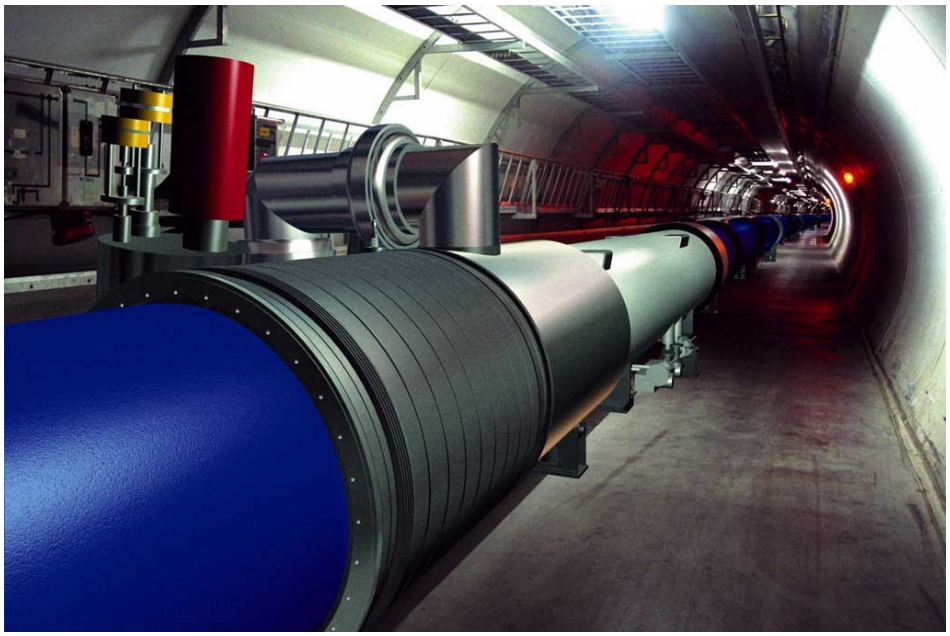
Betreuerin: Jie Tao

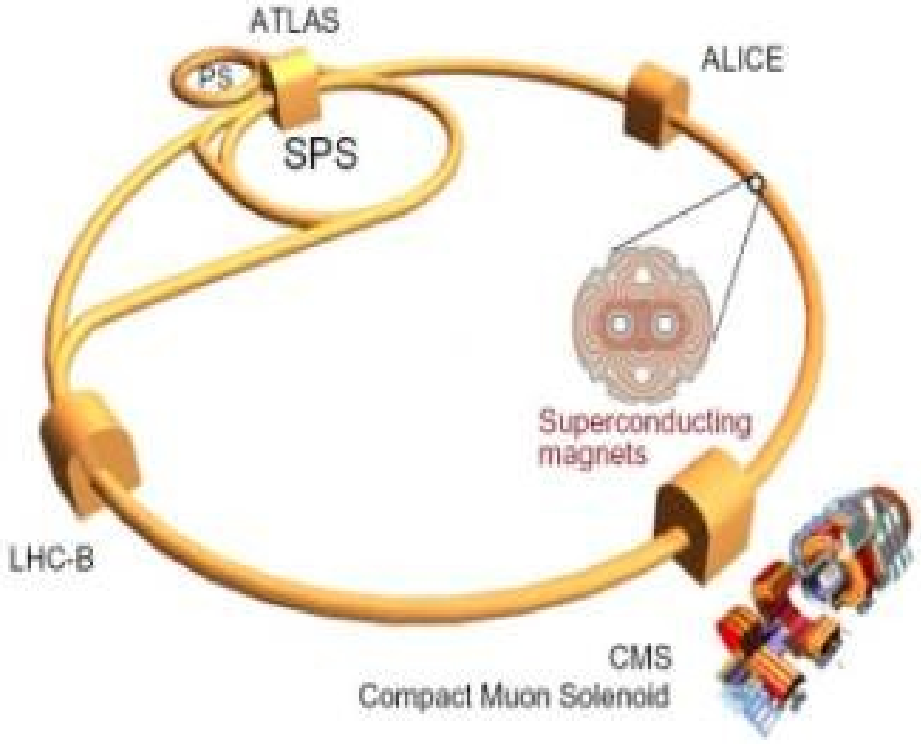
Inhalt

- Motivation
- Herausforderungen im Grid
- OptimalGrid (I)
- Task-Scheduling im Grid (II)
- Autonome Applikationen (III)
- Zusammenfassung

Motivation

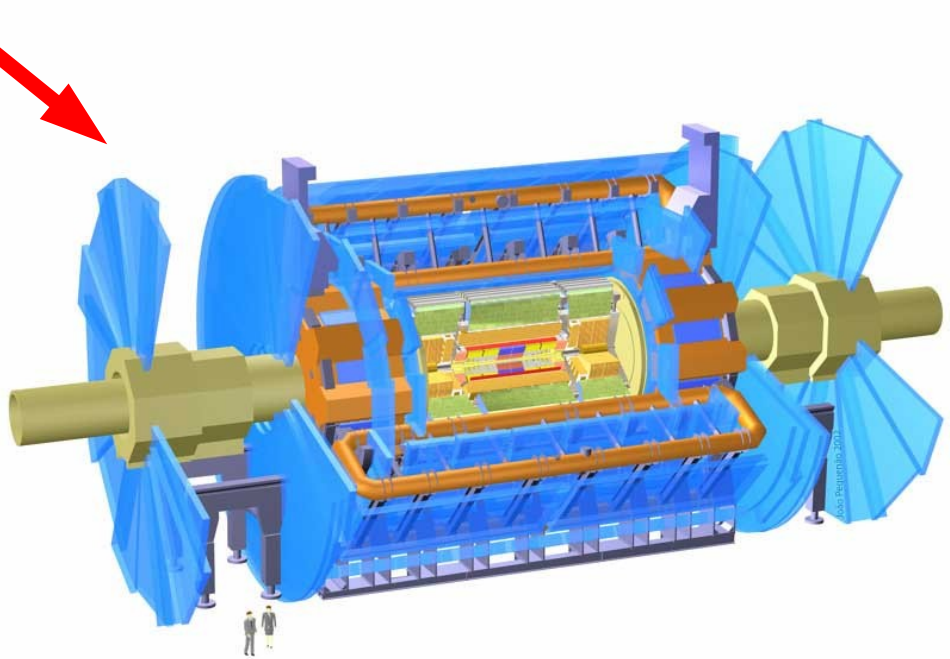
- LHC – Large Hadron Collider @ CERN





Large Hadron Collider (LHC)

- Protonenkollisionen mit 14 TeV
- 1000 Millionen Kollisionen / sec
- 5 Detektoren (ATLAS, CMS, ALICE, LHCb und TOTEM)
- Aufgaben:
  - Entdecken neuer Teilchen (Higgs)
  - Anti-Materie vs. Materie



Herausforderung 1:
Grosse, verteilte Community

- 5000 weltweit verteilte Forscher wollen Daten und Ergebnisse nutzen.
- z.B. ATLAS:
  - 1850 Physiker
  - 150 Unis
  - 34 Länder

Herausforderung 2:
Komplexität des Gesamtsystems

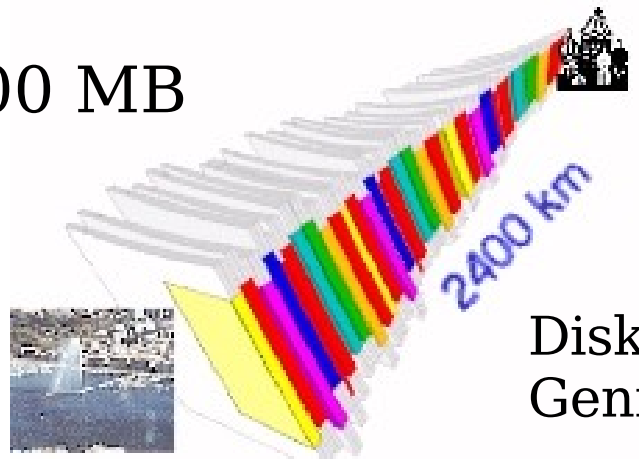
- Software-Aufwand:
  - 1000 Mann-Jahre pro Experiment
- Interoperabilität muss gewährleistet sein.

Quelle: Jürgen Knobloch

Herausforderung 3:
Riesige Datenmengen

- LHC produziert jährlich ca. 15 Petabyte an Daten.
- Berechnungen liefern weitere 100 PB.
- ATLAS: 100 Events von 10^9/sec mit je 3 MB werden gespeichert. Im Jahr: 1 PB.

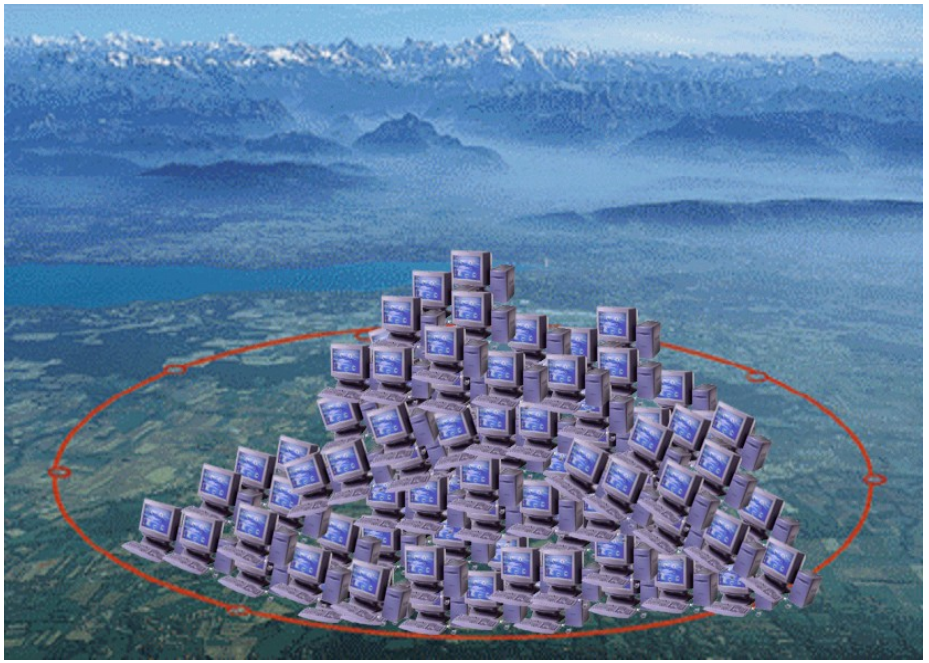
1.000.000.000 MB



Diskettenstapel von Genf nach Moskau

Herausforderung 4:
Benötigte Rechenkapazität

- Gewaltige Rechenkapazitäten werden benötigt.
- z.B. ATLAS: 50.000 PC's

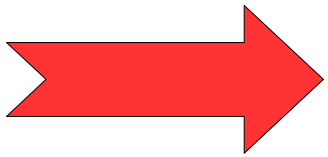


Quelle: Jürgen Knobloch

„Mountains of CPU's“

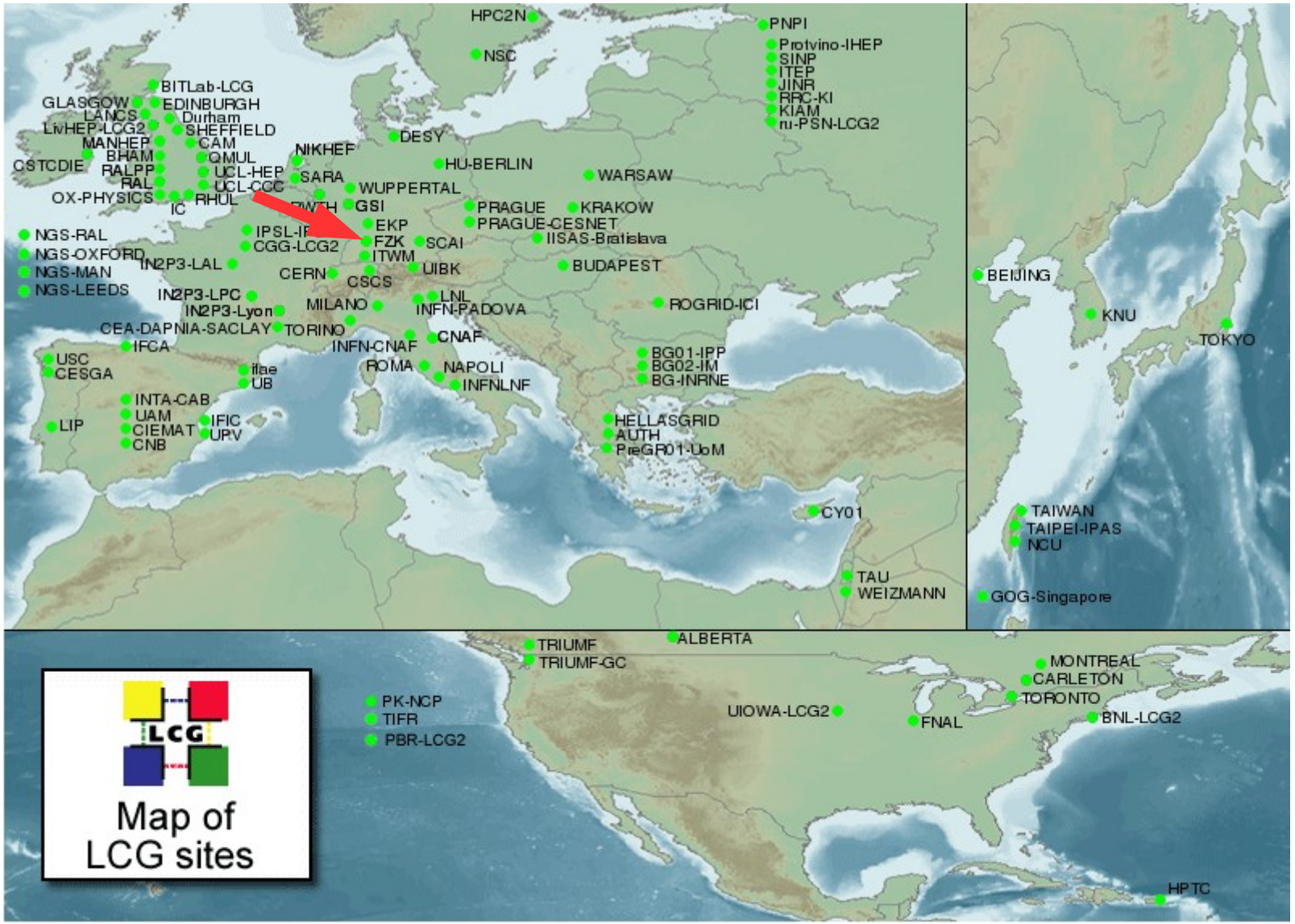
Fazit

Ein Rechenzentrum alleine kann Aufgabe nicht erfüllen, selbst bei optimistischster Prognose bei Hardware-Entwicklung.



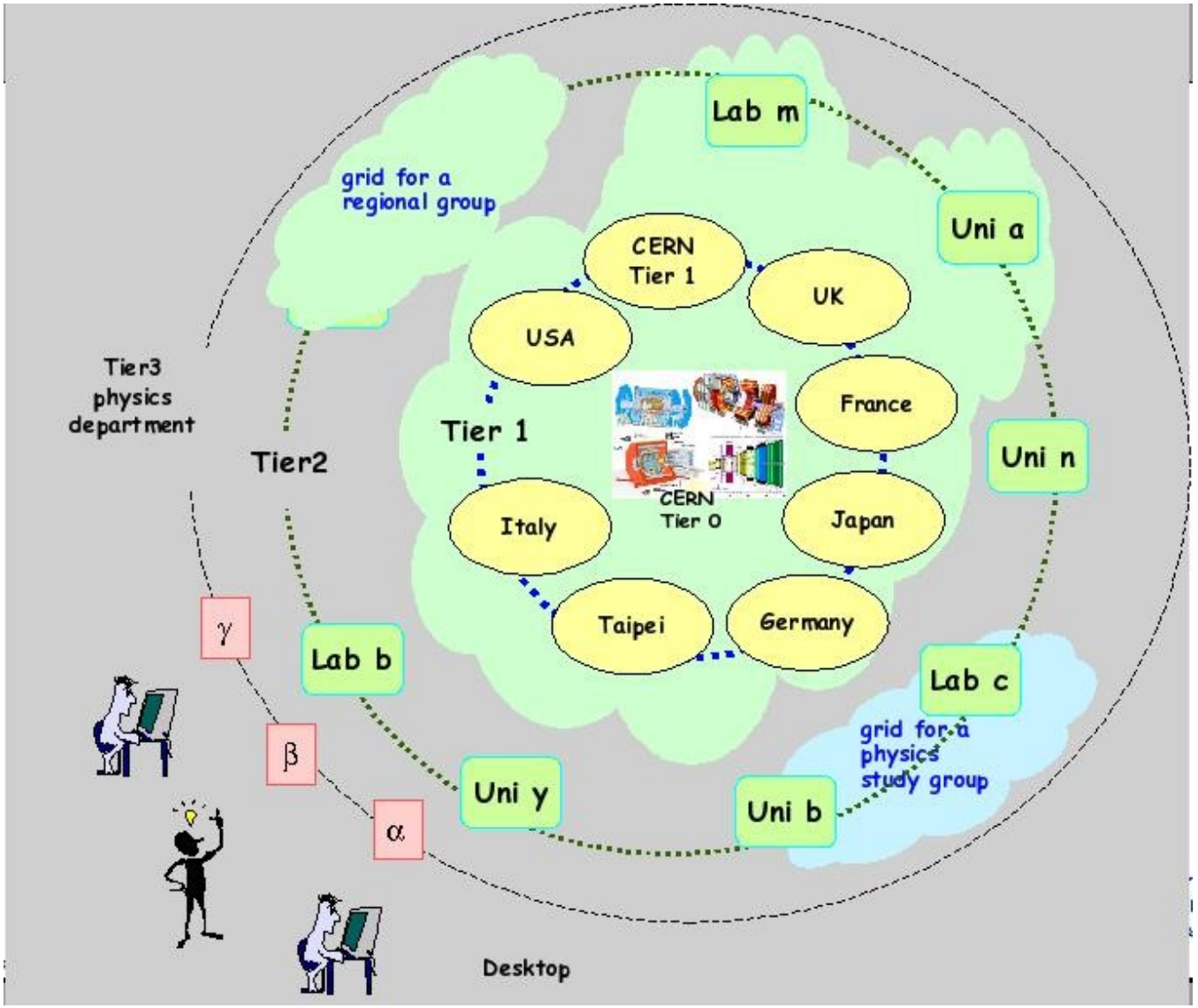
Vorhandene Rechenzentren werden zum **LHC Computing Grid** vereint.

LHC Computing Grid



Quelle: CERN

Architektur vom LCG



Quelle: Jürgen Knobloch

Weitere Anwendungsgebiete

- Molekular-Forschung (Bio/Chemie)
- Wettervorhersagen
- Computer Aided Engineering (CAE)
  - Crash-Tests
  - Werkstoffverhalten
- Finanzsimulationen
- Simulationen von Öl-Bohrungen oder Sprengstoffen.
- ...

Vision

- Grid: Computing Power aus der Steckdose?

Herausforderungen im Grid

- Heterogene, sich dynamisch ändernde Umgebung.
  - Applikationen müssen sich an Umgebung und veränderte Bedingungen selbstständig anpassen können. Evtl. auch zur Laufzeit.
    - Selbst-Konfiguration
    - Context-Awareness
  - Knoten-Ausfälle müssen kompensiert werden können.
    - Selbst-Heilung

Herausforderungen im Grid

- Ressourcen sollen möglichst effizient genutzt werden:
  - Lastausgleich + gutes Task-Scheduling
    - Selbst-Optimierend
  - Monitoring
  - Wechseln von Algorithmen in bestimmten Situationen
    - Context-Awareness
    - Autonome Rekonfiguration

Herausforderungen im Grid

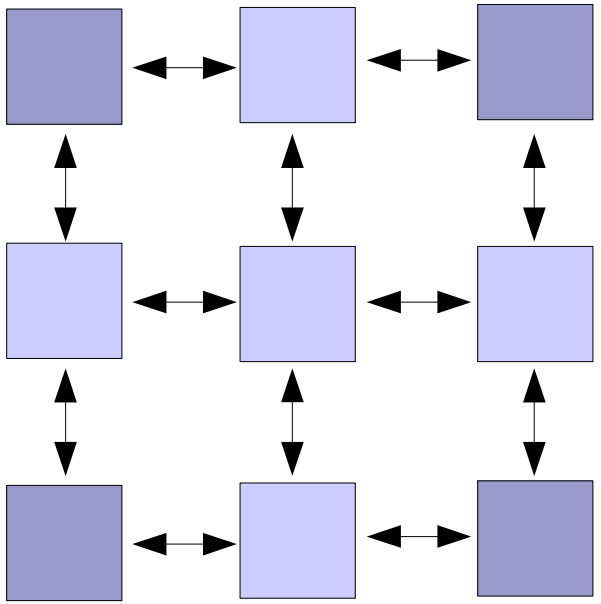
- Software Komplexität in Griff bekommen:
  - Komponenten-Ansatz
  - Trennung von Zielvorgabe und Mechanismus
    $\rightarrow$ Flexibilität
  - Dynamisches Zusammensetzen von Komponenten anhand Zielvorgaben und Regeln.

Teil I

OptimalGrid

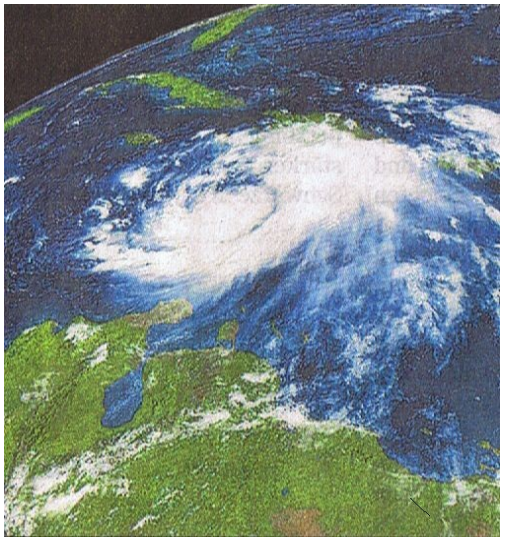
Almaden OptimalGrid

- Zum Lösen von **connected-parallel** Problemen
  - Finite Elemente (FEM)
  - Zellulare Automaten
- Selbst-
  - Konfigurierend
  - Optimierend
  - Heilend
- Adaptiert sich an Grid-Umgebung.



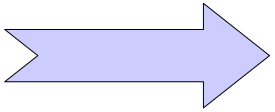
Finite Elemente Methode

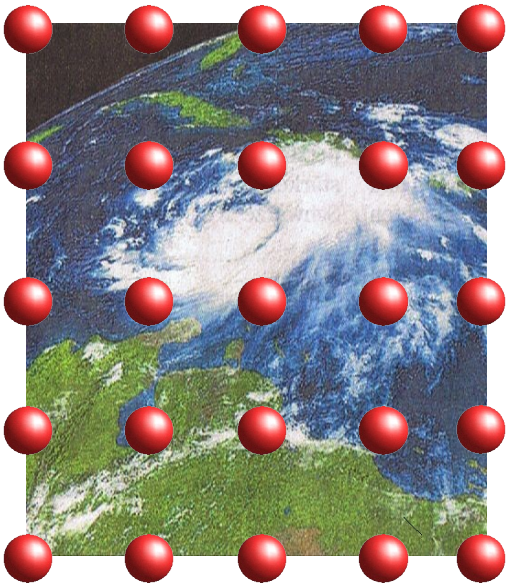
- Kontinuierlicher Wertebereich wird in Zellen zerlegt (Diskretisierung).
- nicht notwendigerweise äquidistant



Quelle: PZ/dpa

kontinuierlich





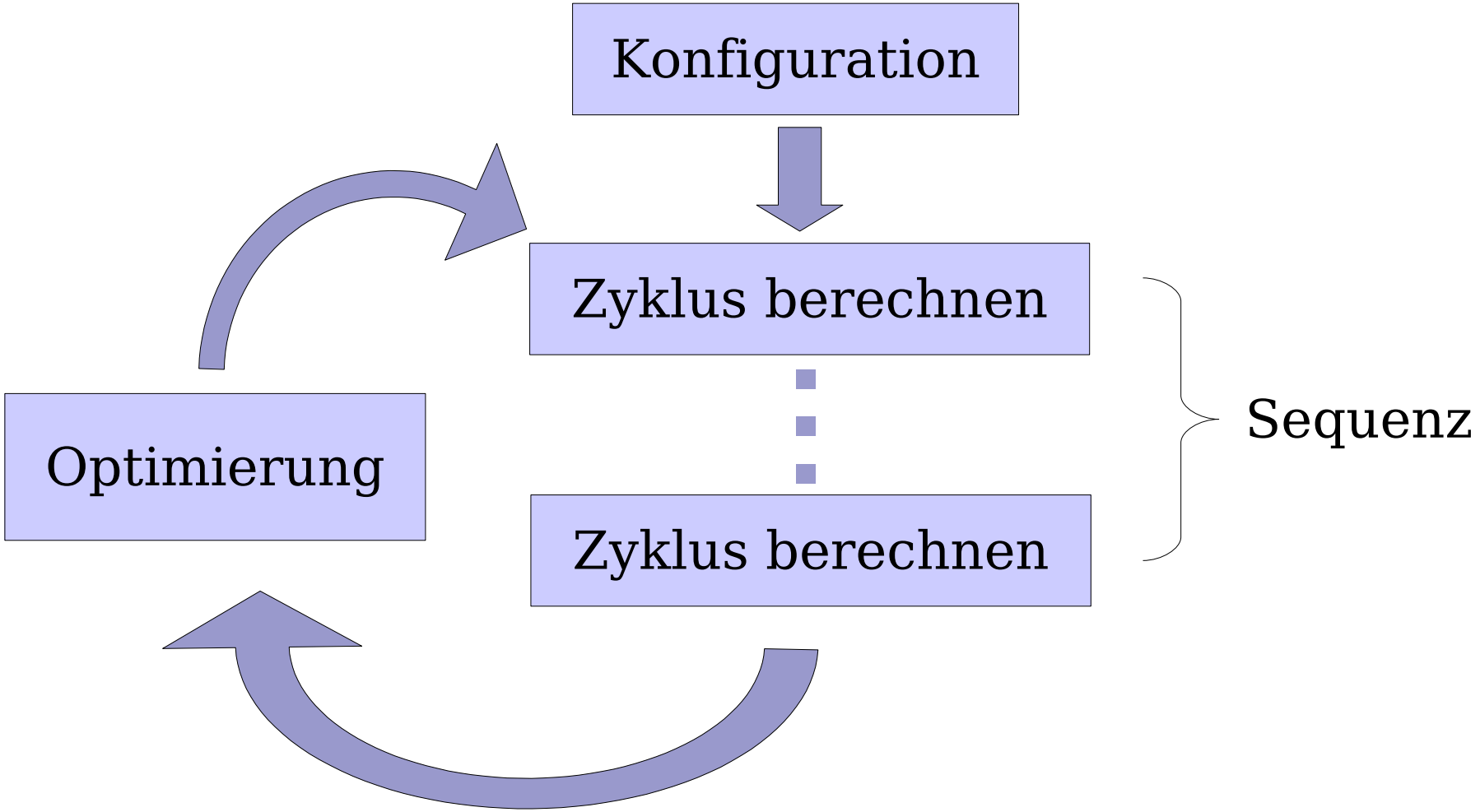
diskret

OptimalGrid

- Zellen (OPC) werden zu Zell-Kollektionen (OPCC) zusammengefasst, um den Verwaltungsaufwand zu minimieren.
- Jeder Grid-Knoten besitzt variable Anzahl an Zell-Kollektionen (VPP).

OPC: Original Problem Cell
OPCC: OPC Collection
VPP: Variable Problem Partition

Ablauf


Konfiguration
Zyklus berechnen
Zyklus berechnen
Optimierung
Sequenz

Selbst-Konfiguration

- Problem Builder:
  - Eingabe:
    - Welche Knoten stehen zur Verfügung?
    - Normalisierte Performance + vorhandender Speicher der Knoten.
    - Netzwerk-Latenz
    - Problem-Charakteristik
  - Ausgabe:
    - Optimale Zahl zu verwendender Knoten.
    - Initiale Verteilung der Zell-Kollektionen an die VPPs.

Selbst-Optimierung

- Der **Autonomic Program Manager** (APM) überwacht die Performance der Knoten.
- Am Ende jeder Sequenz werden die Zell-Kollektionen der Last entsprechend umverteilt (Load-Balancing).

Ziel: Optimale Laufzeit

Selbst-Heilung

- Die Werte aller Eck-Zellen werden nach jedem Zyklus „historisch“ gespeichert.
- Am Ende jeder Sequenz werden alle Zellen gespeichert.
- Fällt ein Knoten aus, so werden seine Zell-Kollektionen an lebende Knoten verteilt und der Recovery Mode eingeleitet.
- gesteuert vom APM.

Recovery Mode

- Die durch den Ausfall des Knotens verlorenen Zell-Kollektionen werden auf die Werte vom Sequenz-Beginn zurückgesetzt.
- Nun werden diese Zell-Kollektionen bis zum aktuellen Zyklus durchgerechnet (unter Verwendung der historischen Eck-Zellen).
- Die anderen Zell-Kollektionen müssen solange warten.

Teil II

Task-Scheduling im Grid

Task-Scheduling im Grid

- Manuell:
  - Erstellen des Schedules von Hand.
  - Im Grid nicht akzeptabel, da Anzahl und Art der Knoten dynamisch zur Laufzeit variieren kann.
  - Ausserdem: Aufwendig und fehleranfällig.
  - Problem bei mehreren Benutzern.
  - Machbar bei Clustern.
  - Kein Scheduling Overhead.

Task-Scheduling im Grid

- Preemptiv+Migration:
  - Tasks können auf andere Knoten migriert werden
  - Migration jedoch „teuer“ (Zeit, Ressourcen)
  - Applikation muss i.d.R. angepasst werden
  - Gute Auslastung kann erreicht werden

Task-Scheduling im Grid

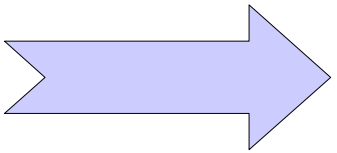
- Nicht-Preemtiv:
  - Angestossener Task läuft bis zum Schluss, kann nicht migriert werden.
  - Um gute Auslastung zu erreichen braucht man einen sehr guten Schedule!
  - Daher: Scheduler sollte Informationen über Laufzeitverhalten und Ressourcenbedarf der Applikationen verfügen.
    - Muss vom Programmierer mitgegeben werden. Aufwand!

Der Titan Scheduler

- Ziele:
  - Minimieren der durchschnittlichen Deadline-Überschreitung (average delay)
  - Minimieren der Laufzeit
  - Minimieren der Leerlauf-Zeit (Idle-time)
  - Maximieren der Ressourcen-Auslastung
  - Quality of Service
- Adaptiert sich an dynamische Umgebung.
- Keine Task-Preemption/Migration

Voraussetzungen

- Laufzeit & Ressourcenbedarf einer Task muss (unabhängig von der verwendeten Hardware) bekannt sein, sonst kann kein guter Schedule gefunden werden und Deadlines nicht eingehalten werden.

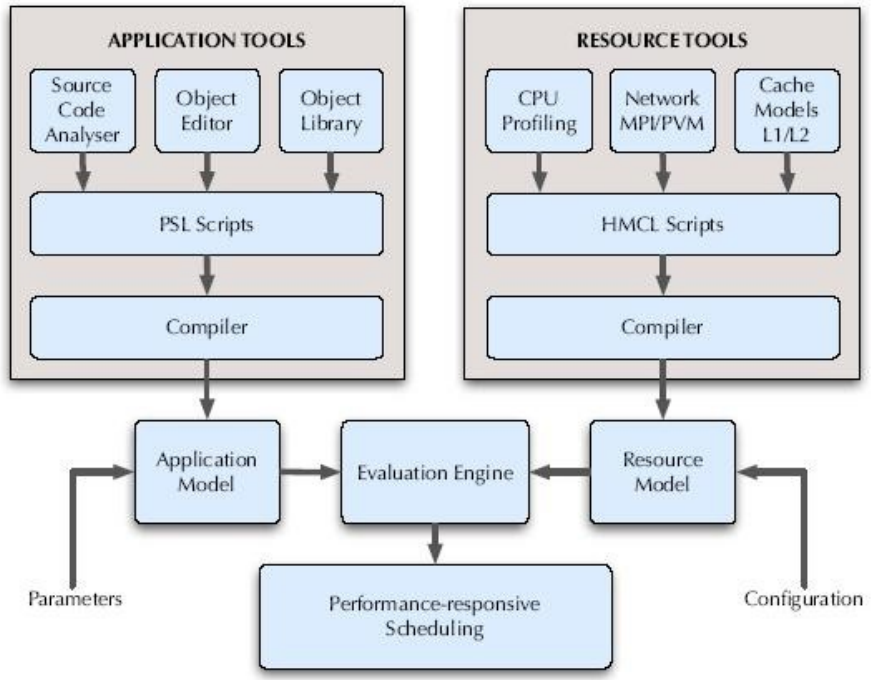


Performance-Modellierungssprache:
PACE (Performance Analysis and Characterization Environment)

PACE Toolkit

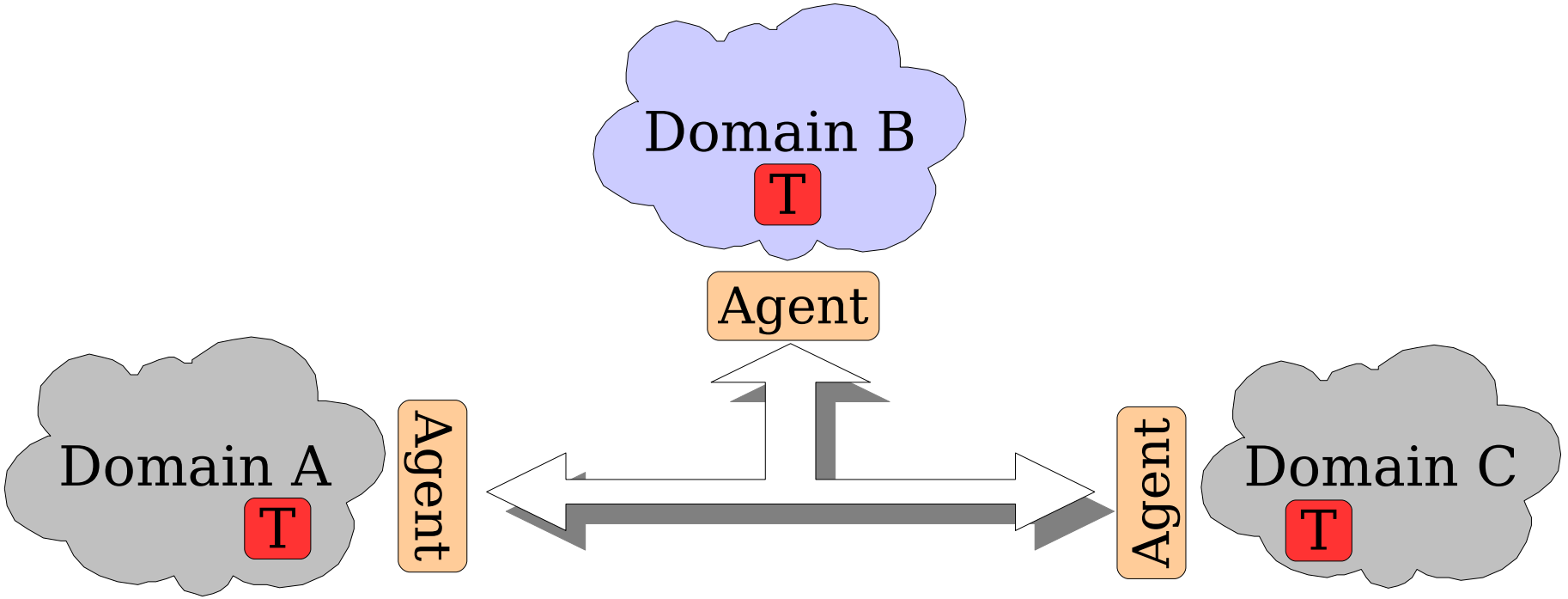
- Getrennte Performance-Beschreibungen für Applikation und Hardware.
- Ermöglicht: Evaluierung einer Applikation auf unterschieder Hardware (Skalierung).
- Liefert quantitative Daten wie:
  - geschätzte Laufzeit
  - geschätzter Speicher-bedarf

Quelle: Univ. Warwick



Aufbau von Titan

- High-level, Domain übergreifender Lastausgleich (P2P Agents)
- Intra-Domain Scheduling



Task-Submission

- PACE-Modell des Tasks wird zur Schätzung der Laufzeit herangezogen.
- Wenn Task-Deadline erfüllbar, füge Task zur Schedule-Queue hinzu.
- Ansonsten: An benachbarte Titan-Scheduler (Inter-Domain) weiterleiten. Falls keiner Deadline einhält, wähle Schedule mit geringster Deadline-Überschreitung.

Optimierung

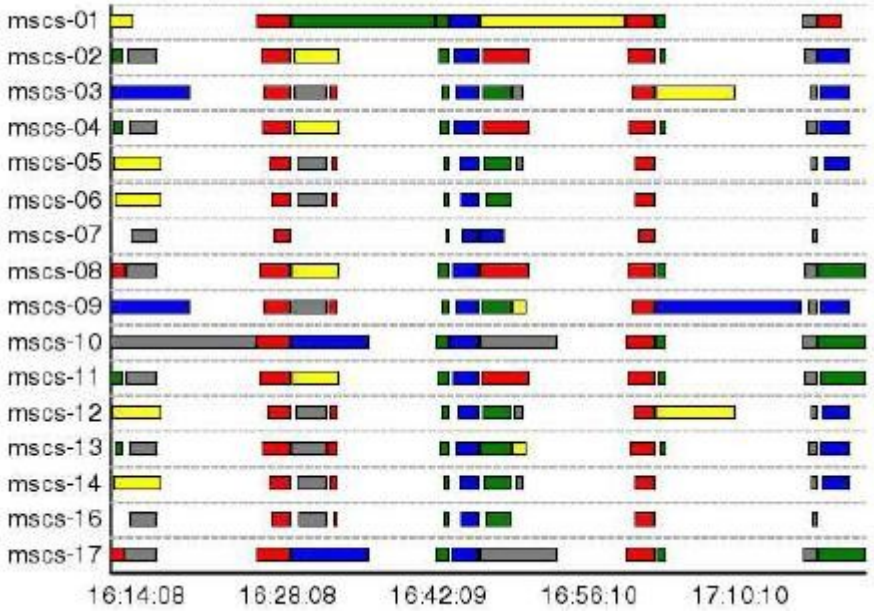
- Schedule-Queue enthält Tasks, die auf Ihre Ausführung warten.
- Schedule-Queue wird ständig durch Genetischen Algorithmus (GA) optimiert und umgeordnet:
  - Balanciert Parameter die in Konflikt stehen
  - Optimiere Laufzeit, Leerlauf und Deadline-Fehler
- Kombiniert Hardware und Applikations-Modell (PACE) zur Laufzeit.
- Tasks vom Begin der Queue werden an Condor zur Ausführung auf Hardware weitergeleitet.

Optimierung

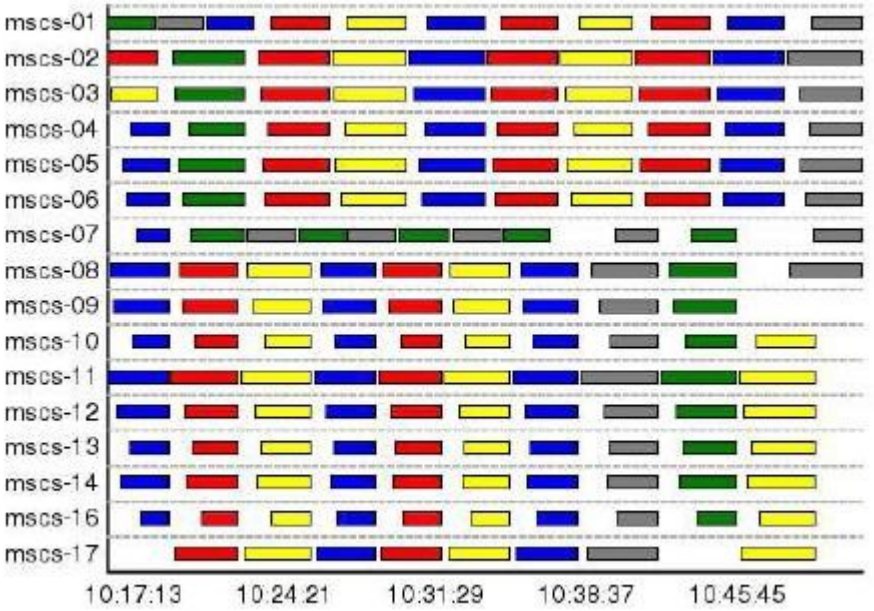
- Titan beobachtet Intra-Domain Resourcen. Gehen Maschinen off-line, oder kommen neue hinzu, so reagiert der GA sofort und optimiert Schedule entsprechend.
- Task beendet: Titan vergleicht benötigte Laufzeit mit vorhergesagter und füttert damit PACE → Nächstes Mal: Genauere Vorhersage.

Ergebnis: Intra-Domain

Quelle: Univ. Warwick



Condor



Titan

2x schneller!

Experiment	Time (m)	Idle (%)
Condor		
arbitrary hosts per task	70.08	61
maximum hosts per task	69.10	28
pre-calculated hosts per task	38.05	14
Condor plus co-scheduler Titan	35.19	21

Ergebnis: Multi-Domain

- Gesamt: 256 Knoten
- 16 Domains à je 16 Knoten
- 6 verschiedene Platformen

Laufzeitreduktion: 2.7 – 5.9

Auslastung: ~ 2x so gut

Deadline-Verbesserung: 32-56%

r	r/s	M	t(s)	ε(s)	ν(%)
200	1	OFF	839	-1	24
200	1	ON	302	78	51
200	2	OFF	812	-36	25
200	2	ON	245	72	50
200	5	OFF	814	-64	24
200	5	ON	218	62	49
500	1	OFF	1784	-112	28
500	1	ON	633	78	57
500	2	OFF	1752	-205	29
500	2	ON	461	66	57
500	5	OFF	1877	-276	27
500	5	ON	305	32	68
1000	1	OFF	2752	-314	36
1000	1	ON	1160	79	61
1000	2	OFF	2606	-493	39
1000	2	ON	669	65	74
1000	5	OFF	2756	-681	36
1000	5	ON	467	-6	77

Quelle: Univ. Warwick

Teil III

Autonome Anwendungen
(AutoMate)

Autonome Anwendungen

- Mensch gibt Strategie vor.
- Anwendung soll sich autonom an geänderte Zielvorgaben anpassen können.
- Zielvorgaben können sein:
  - Höchste Performance
  - Niedrigste Kosten
  - Genauigkeit
  - ...

AutoMate

- Anwendungen setzen sich zusammen aus Komponenten.
- Herkömmlich: Statische Komposition von Komponenten.
- Ziel von AutoMate ist es, die Zusammensetzung der Komponenten dynamisch und autonom anzupassen, je nach:
  - Zielvorgabe
  - Zustand
  - Kontext

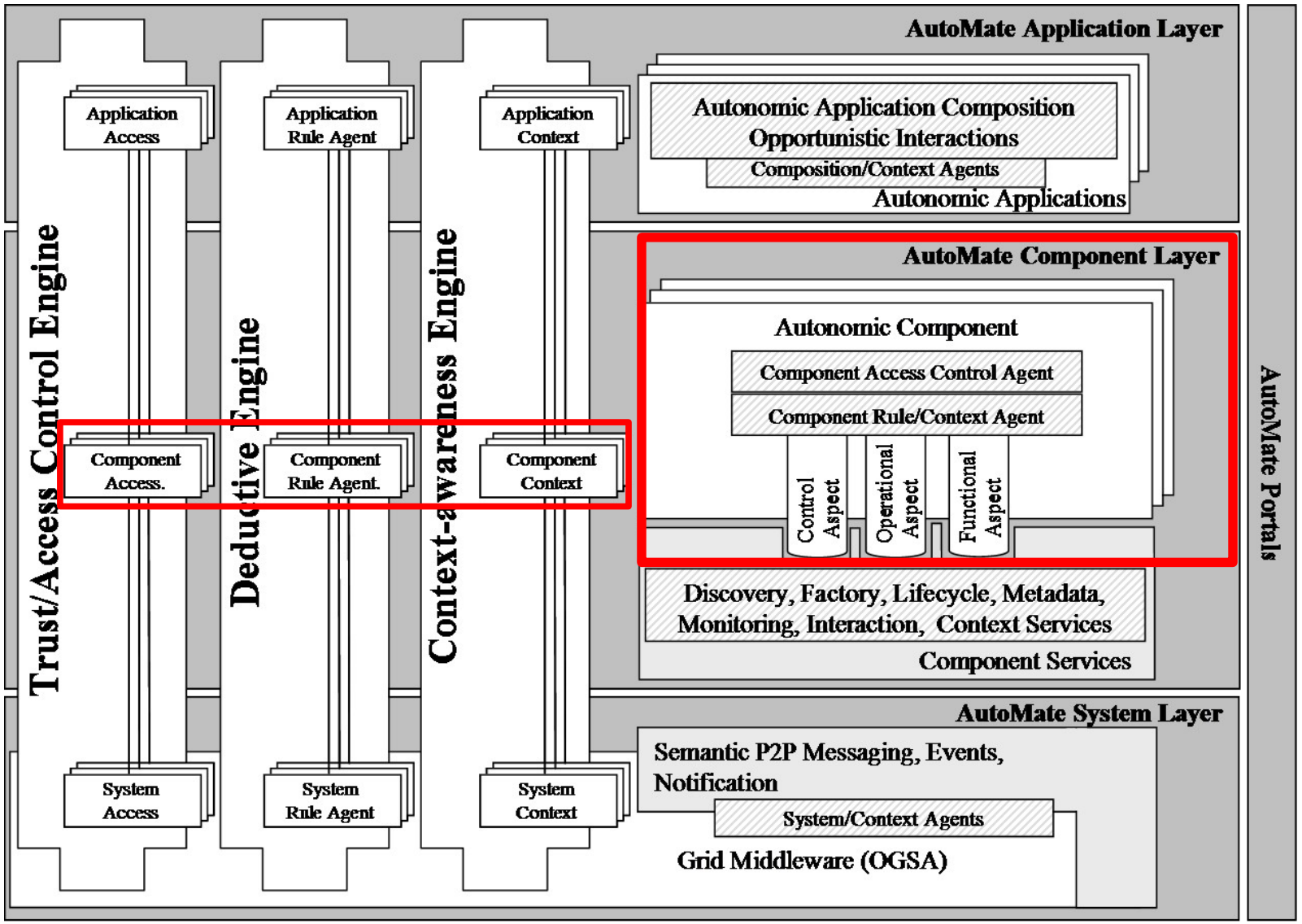
Beispiele

- Bei hoher Last soll auf ein Latenz-tolerantes Netzwerkprotokoll gewechselt werden.
- Wahl von numerischen Algorithmen:
  - Performance vs. Genauigkeit
  - „Ungenaues Ergebnis besser als gar keins.“

Autonome Komponenten

- Trennung von Zielvorgabe und Mechanismus
- AC's exportieren zusätzlich zum herkömmlichen Interface Informationen wie:
  - Welche Funktion erfüllt die Komponente (z.B. „Sortieren“) WAS
  - Komplexität O(?), Performance, Skalierung, Ressourcenbedarf WIE
  - Wer darf Wann zugreifen WER, WANN
  - Interaktion mit Umgebung. Sensoren/Aktoren
  - Regeln

AutoMate



Quelle: Univ. Rutgers

RUDDER / AutoMate

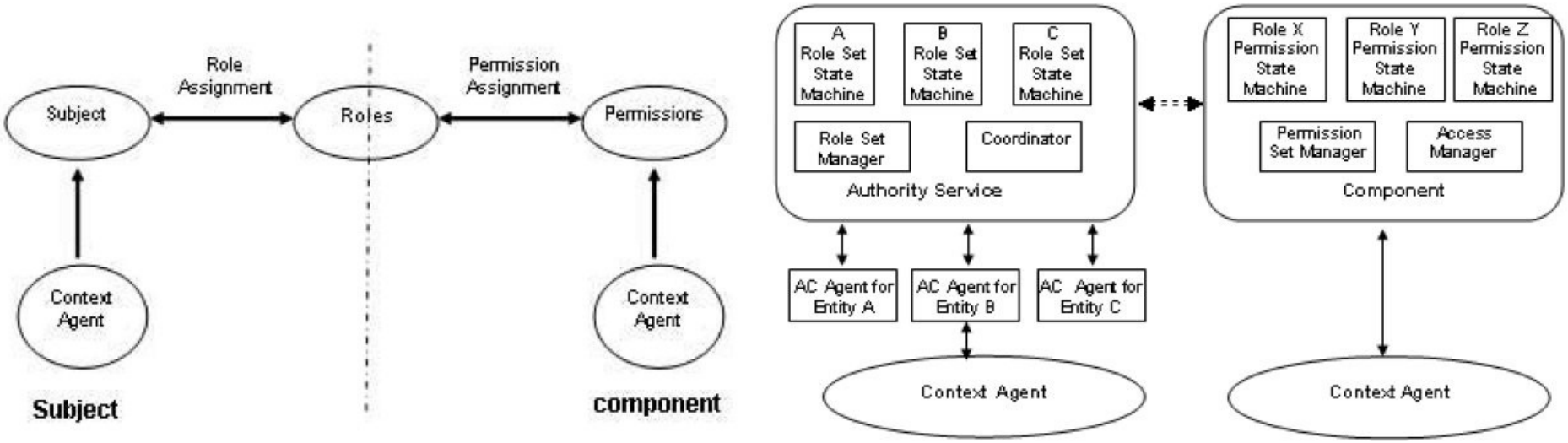
- Deduktive Regel-basierte Engine
- Kernstück von AutoMate. Ermöglicht dynamische Komposition von Komponenten.
- Konfliktauflösung
- Lernmechanismen

```
RULE GrACE_Switch_Partitioner
ON EVENT (Request_Switch_Partitioner)
USE  Context_Service – Define_System_State
USE  Component_Aspects – Define_Application State
If (Computation-communication ratio > cc_threshold) THEN
ACTION (PostEvent "PartitionerType=ISP")
ELSE
ACTION (PostEvent "PartitionerType=G-MISP+SP")
END IF
```

Quelle: Univ. Rutgers

SESAME / AutoMate

- Regelt Zugriff auf Komponenten
  - dynamisch
  - rollenbasiert



Quelle: Univ. Rutgers

Zusammenfassung

- Autonomic Computing im Grid sinnvoll.
  - komplex
  - heterogen
  - dynamisch

  } Computer kann damit besser umgehen
- Mensch gibt nur Strategie vor.

Vielen Dank für Eure Aufmerksamkeit!

Quellen

- CERN: http://cern.ch
- LHC: http://lhc.web.cern.ch/lhc/
- ATLAS: http://atlas.ch
- LCG: http://lcg.web.cern.ch/LCG/
- LCG Design Report (Interessant!): http://cern.ch/lcg/tdr
- EuroGrid: http://www.eurogrid.org/
- Globus-Toolkit: http://www.globus.org/toolkit/
- OptimalGrid: http://www.almaden.ibm.com/software/ds/OptimalGrid/
- PACE, Titan: http://www.dcs.warwick.ac.uk/research/hpsg/
- AutoMate: http://automate.rutgers.edu/