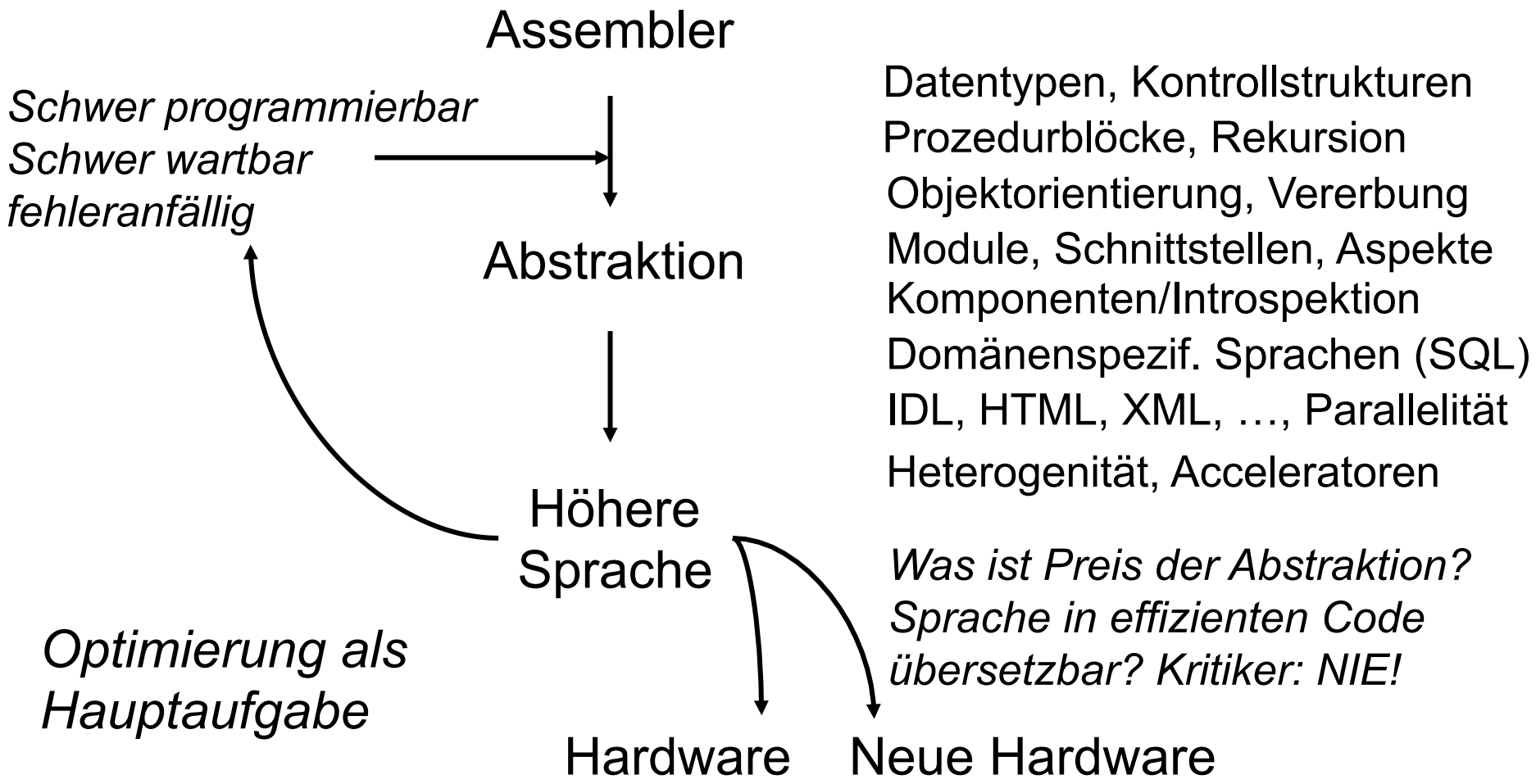
Angebote der Informatik 2 für Ihre Vertiefung
→ Fach *Programmiersysteme*

Prof. Dr. Michael Philippsen

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

DEPARTMENT INFORMATIK
LEHRSTUHL FÜR PROGRAMMIERSYSTEME
(INFORMATIK 2)

Ewiges Dilemma der Programmiersysteme

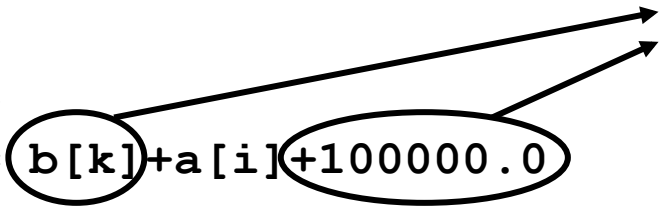


Programmoptimierung am Beispiel

Original (Fortran):

```
subroutine tricky(a,b,n,m,k)
integer n,m,k
real a[m], b[m]


do i=1,n
  a[i] = b[k]+a[i]+100000.0
end do
return
```



Transformierte Version:

```
subroutine tricky(a,b,n,m,k)
integer n,m,k
real a[m], b[m]

C = b[k]+100000.0
do i=1,n
  a[i] = a[i]+C
end do
return
```

Auf den ersten Blick korrekt, aber völlig falsch! 4 grobe Fehler!

Fehler des Transformationsbeispiels (1)

Original:

```
do i=1,n
  a[i] = b[k]+a[i]+100000.0
end do
return
```

Transformierte Version:

```
C = b[k]+100000.0
do i=1,n
  a[i] = a[i]+C
end do
return
```

Überlauf:

Wenn `b[k]` sehr groß und `a[i]` negativ sind, dann führt die Bildung von `C` vor der Schleife zu einem Überlauf, der im Original nicht auftrat.

Selbst wenn Überlauf auch im Original bei einem i auftritt: man beobachtet mit `print`-Anweisungen andere Ausgaben.

Fehler des Transformationsbeispiels (2)

Original:

```
do i=1,n
  a[i] = b[k]+a[i]+100000.0
end do
return
```

Transformierte Version:

```
C = b[k]+100000.0
do i=1,n
  a[i] = a[i]+C
end do
return
```

Wertveränderung:

Operationen auf Gleitkommazahlen sind nicht assoziativ.

Unterschiedliche Reihenfolge der Additionen kann zu (leicht) geänderten Werten führen.

Fehler des Transformationsbeispiels (3)

Original:

```
do i=1,n
  a[i] = b[k]+a[i]+100000.0
end do
return
```

Transformierte Version:

```
C = b[k]+100000.0
do i=1,n
  a[i] = a[i]+C
end do
return
```

Speicherbereichsverletzung:

Mit `n<1` wird die Schleife im Original nicht betreten.
Der Ausdruck b[k] ist bei k>m zwar illegal wird aber bei einer nicht betretenen Schleife nicht ausgewertet.

In der transformierten Version wird b[k] immer ausgewertet und führt daher ggf. zu einem Fehler.

Fehler des Transformationsbeispiels (4)

Original:

```
do i=1,n
  a[i] = b[k]+a[i]+100000.0
end do
return
```

Transformierte Version:

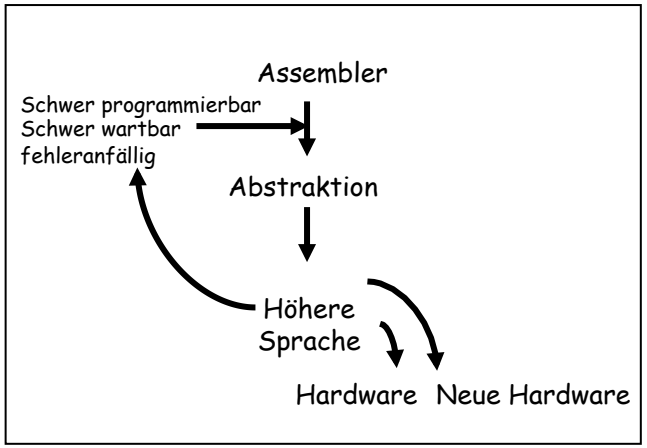
```
C = b[k]+100000.0
do i=1,n
  a[i] = a[i]+C
end do
return
```

Aliase/Teil-Aliase:

In Fortran können sich die Arrays **a** und **b** (teilweise) überlappen. In anderen Sprachen können **a** und **b** Aliase für (Zeiger auf) das gleiche Objekt sein.

Eine versteckte Änderung von **b[k]** macht sich in der Original-schleife bei nachfolgenden **a[i]** bemerkbar.
Das ist in der transformierten Version nicht der Fall.

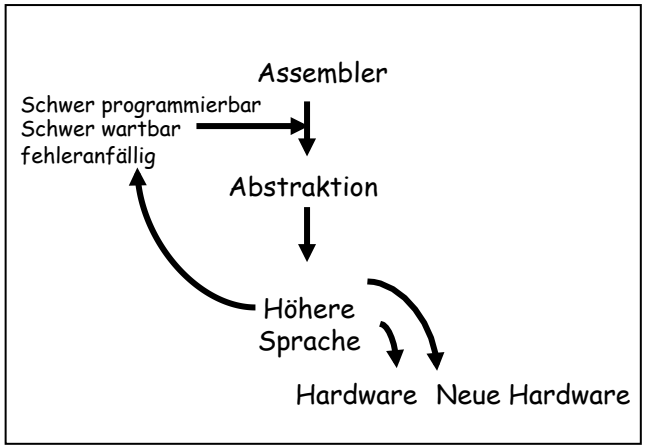
Aktuelle Situation



Für Desktop & Server:
x Zyklen durchlaufen;
Recht gute Ergebnisse;
ständig neue Fragen.

- Kann Java jemals so schnell werden wie C/C++?
  - Just-in-Time-Übersetzer
  - Generationen-basierte Speicher-bereiniger (garbage collector)
- Wie nutzt man Caches effizient?
  - Cache in Sprache sichtbar (register)?
  - Optimierung beim Übersetzungslauf?
  - Laufzeitdaten sammeln und bei Übersetzung nutzen?
- Neue HW-Entwicklung
- Erhöhung des Abstraktionsniveaus:
  - Komponenten,
  - Aspektorientierte Programmierung

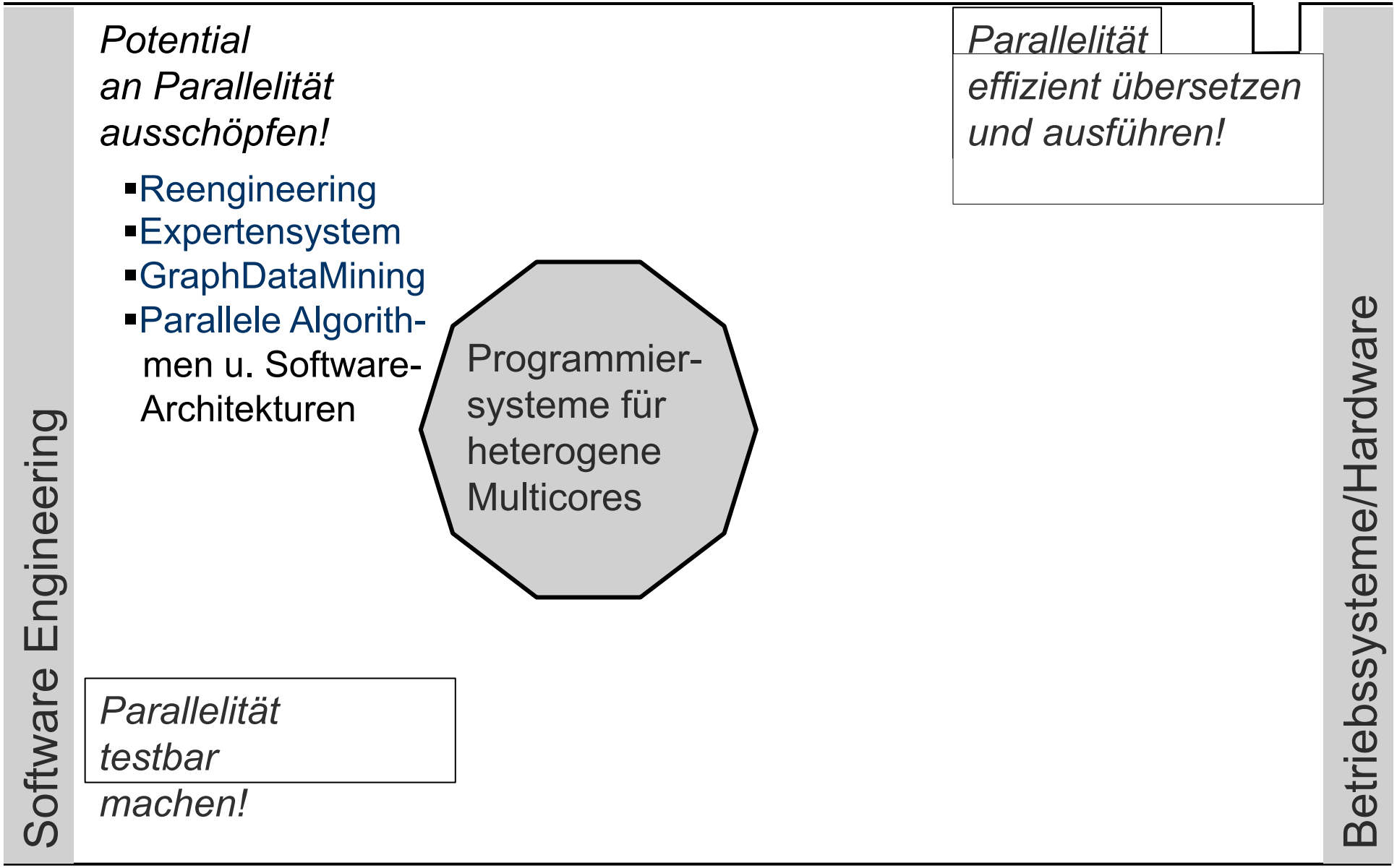
Nicht-„klassische“ Hardware



Viel früher im Zyklus.
Schlechte Optimierung wirkt sich viel stärker aus:
- Latenz versus Register
- Code zu groß für HW

- Parallele Hardware
  - Verschiedenste (auch heterogene) Architekturen
  - Latenz/Bandbreite des Netzes
  - Lokalität/Verteilung von Daten und Berechnungen
  - Deklarative Synchronisation
  - Divergenz auf GPUs
  - Freiheit von Wettlaufsituationen
- Eingebettete Systeme
  - Optimierungsziel: Größe
  - Optimierungsziel: Stromsparen

Informatik 2 – Programmiersystemforschung

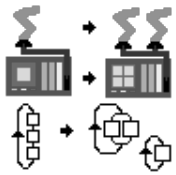


Informatik 2 – Programmiersystemforschung

Software Engineering

Potential an Parallelität ausschöpfen!

Repository-Mining: SIFE, C3, SYFEX



Parallelisierung für eingebettete Systeme in Automatisierungs-technik

Verteilte Ereig-nisverarbei-tung

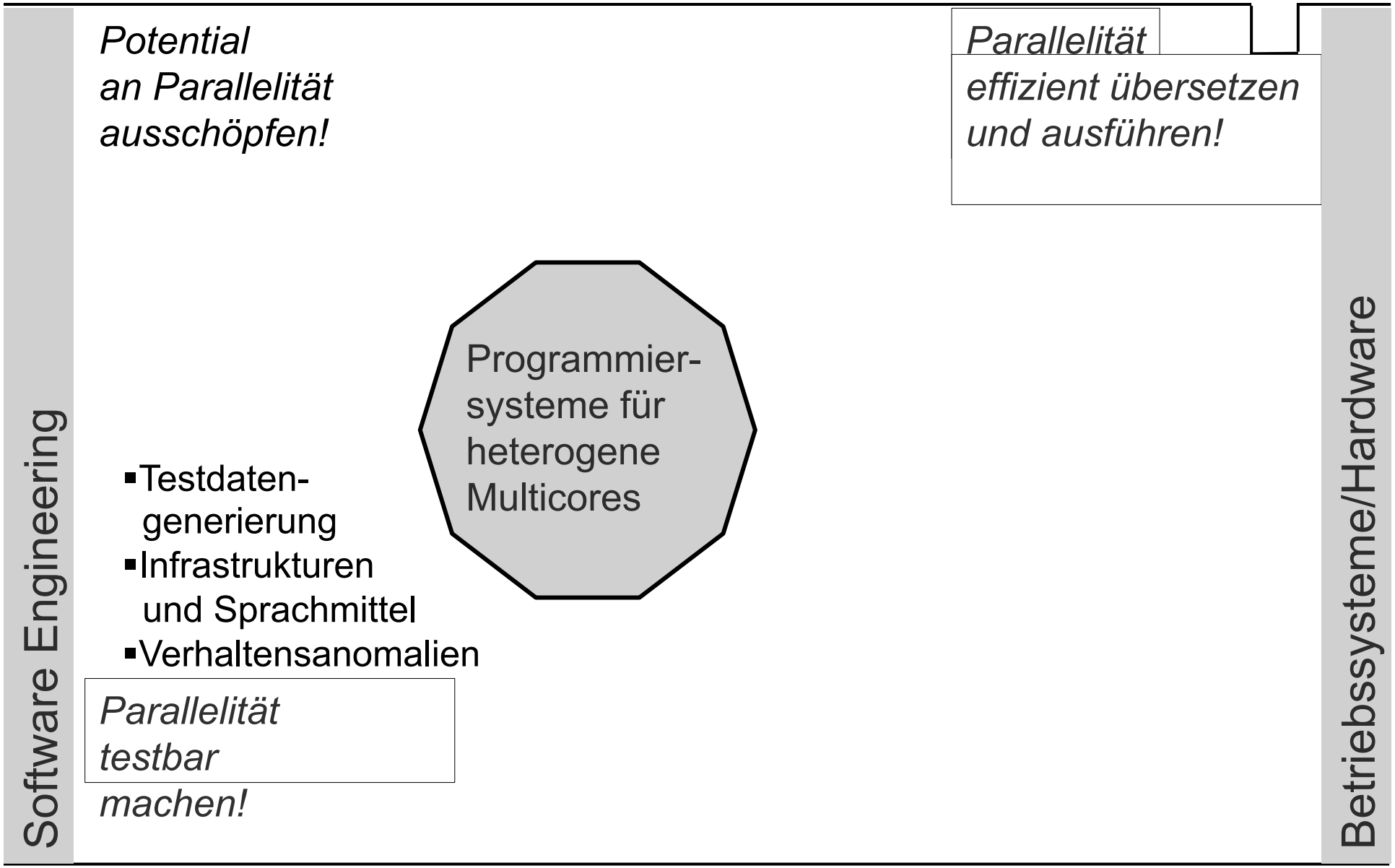
Programmier-systeme für heterogene Multicores

Parallelität effizient übersetzen und ausführen!

Parallelität testbar machen!

Betriebssysteme/Hardware

Informatik 2 – Programmiersystemforschung



Informatik 2 – Programmiersystemforschung

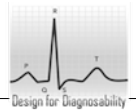
Software Engineering

Potential an Parallelität ausschöpfen!

Parallelität effizient übersetzen und ausführen!

Programmier-systeme für heterogene Multicores

Testwerk-zeuge für embedded Multicore-Software



Design for Diagnosability

Parallelität testbar machen!

Testfallgenerierung für Thread-Interleaving

Betriebssysteme/Hardware

Informatik 2 – Programmiersystemforschung

Software Engineering

Potential an Parallelität ausschöpfen!

Parallelität effizient übersetzen und ausführen!

- **Sprachmittel**
  OpenMP/Java, hybride Sprachen, domänenspezif. Sprachen, PGAS JavaParty, Synchronisationsprimitive, ...

- **Distributed Shared Memory auf Clustern**
  Lokalität, Prefetching, Verbergung der nicht-uniformen Speicherhierarchie, ...
- **Laufzeitsysteme**
  Dynam. Adaption des Parallelitätsgrads, Virtualisierung, optimierte Platzierung von Funktionen, Kommunikations-bibliotheken für heterogene Netze & NOCs
- **Code-Generierung**
  Cuda, OpenCL, GPU, ...
  Portabilität
- **Programmcodeoptimierung**
  RAM-Bedarfsverkleinerung, Stromsparoptimierung, Register-optimierung (für OSEK), Codegrößenreduktion, parallele Daten-flussanalysen, Transaktionen, ...

Parallelität testbar machen!

Betriebssysteme/Hardware

Informatik 2 – Programmiersystemforschung

Software Engineering

Potential an Parallelität ausschöpfen!

Parallelität testbar machen!

Programmier-systeme für heterogene Multicores

Parallelität effizient übersetzen und ausführen!

Tapir: hybride Progammiersprache für verteilte Systeme; Transaktionsspeicher

Jackal: „big“ Java über Cluster, Prefetching, Lokalität, Funktions-verteilung

Transparentes verteiltes Pro-grammieren, Lokalitätsoptimierung, effizienter Methodenfernaufruf

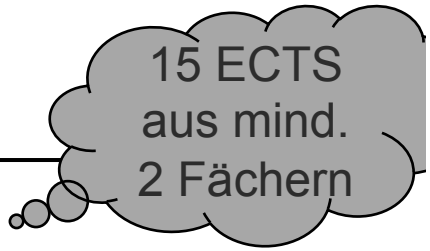
OpenMP-Binding für Java, Dynamische Reparallelisierung, Invasives Rechnen

Domänenspezifische Sprachen und deren Optimierung

Embedded Realtime Language Development Framework

Betriebssysteme/Hardware

Unsere Angebote für das Bachelor-Studium (1)



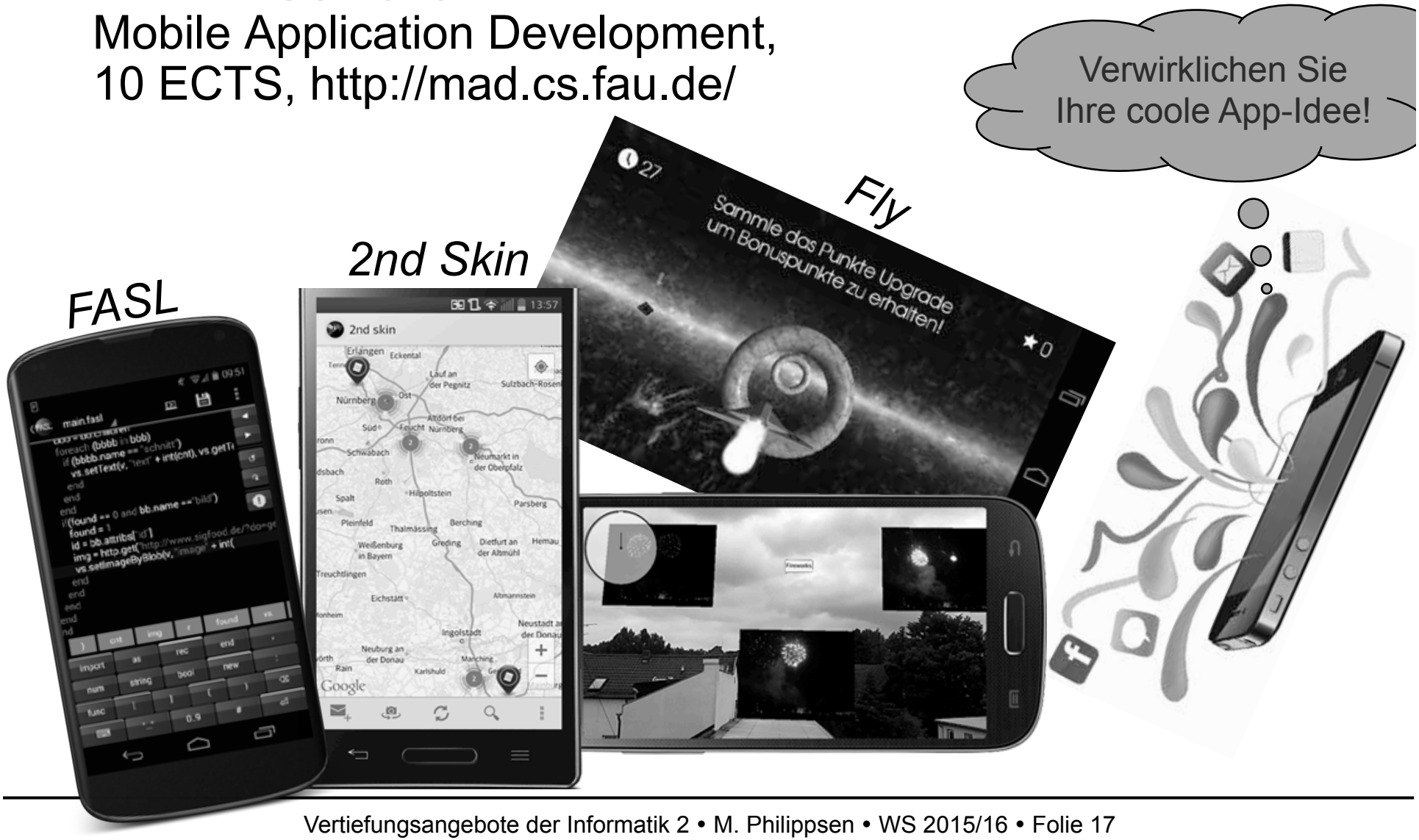
Wahlpflichtmodule für das 5./6. Semester in der Vertiefungsrichtung (Fach) Programmiersysteme:

- Regelmäßige Angebote im WS:
  - Grundlagen des Übersetzerbaus — 2+2 SWS = 7.5 ECTS
    Hausaufgaben und 30 min. mündliche Prüfung
  - ~~Parallele Algorithmen — 2+2 SWS = 5 ECTS~~

- Regelmäßige Angebote im SS:
  - Programmierung und Architekturen von Cluster-Rechnern (zusammen mit Prof. Fey) — 4+2 SWS = 7.5 ECTS
    30 min. mündliche Prüfung
    auch im Fach Rechner-architektur wählbar!
  - ~~Principles of Prog. Languages — 2+2 SWS = 5 ECTS~~

Praktikum (BA) und Projekt (MA)

- MAD im SS 2016
  Mobile Application Development,
  10 ECTS, http://mad.cs.fau.de/



Unsere Angebote für das Bachelor-Studium (3)

Seminare (wechselnde Themen):

- Machine Learning
- Modern Concepts of Distributed Systems
- …

- Hallo Welt für Fortgeschrittene
- Hallo Bath, UK

Programmierwettbewerbe

Wintersemester:

- Just-for-Fun-Wettbewerb im Team zum Schnuppern
- *Am 30.1.2016*

Sommersemester:

- Teamwettbewerb zur Qualifikation für den NWERC, den wir 2009 und 2010 gewonnen haben.

Lehrveranstaltungen:

- 5 ECTS-Seminar „Hallo Welt für Fortgeschrittene“ im SS
- 5 ECTS-Projektseminar „Hallo Bath, UK!“ im WS

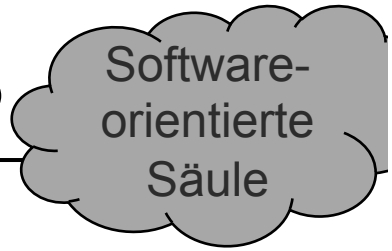
Trainingswochenende

http://icpc.cs.fau.de/

Bachelor- und Master-Arbeiten am Lehrstuhl Informatik 2

- Praktisches Arbeiten, Informatik als Ingenieurdisziplin
- Bachelor-/Masterarbeiten führen Sie
  - an unsere Forschungsthemen und
  - an wissenschaftliches Arbeiten heran.
- Bachelor-/Masterarbeiten sollten eine offene Frage lösen:
  - Keine Papiertiger
  - Keine reinen „Hack-Arbeiten“
  - Ziel: „Beitrag zu international sichtbarem Forschungsergebnis“
- Wichtig sind uns:
  - Projektarbeit, praktische Erfahrung
  - Zügige Bachelor-/Masterarbeiten ggf. mit Industriebeteiligung.

Programmiersysteme im Master-Studium (1)



Regelmäßige Lehrveranstaltungen:

- Interessensschwerpunkt „Übersetzer“
  - Grundlagen des Übersetzerbaus* 2+2 SWS = 7.5 ECTS, WS
    Hausaufgaben und 30 min. mündliche Prüfung
  - Optimierungen in Übersetzern 2+2 SWS = 7.5 ECTS, SS
    Hausaufgaben und 30 min. mündliche Prüfung
  - Ausgew. Kapitel aus dem Ü.-bau 2+2 SWS = 7.5 ECTS, WS
    Blockpraktikum und 30 min. mündliche Prüfung
- Interessensschwerpunkt „Parallel“
  - Programmierung und Architekturen von Cluster-Rechnern* (zusammen mit Prof. Fey) 4+2 SWS = 7.5 ECTS, SS *auch im Fach Rechner-architektur wählbar!*
  - ~~Parallele Algorithmen*~~ ~~2+2 SWS = 5 ECTS, WS~~
  - Optimierungen in Übersetzern 2+2 SWS = 7.5 ECTS, SS

Prüfungen wie oben

*falls noch nicht im BA eingebracht.

Programmiersysteme im Master-Studium (2)

Ergänzende Lehrveranstaltungen (teilweise unregelmäßig)

Lang.

- ~~□ Principles of Progr. Languages*~~ ~~2+2 SWS = 5 ECTS, SS~~
- □ Funktionale Progr. in Haskell* 2+2 SWS = 5 ECTS, SS

pSWT

- □ Analyse & Design OO-SW mit UML* 2+2 SWS = 5 ECTS, SS
  Auch im Fach PSWT wählb.!

Sonst.

- □ Geschichte der Progr.-sprachen 2 SWS = 2.5 ECTS, WS
  ~~(in den meisten Studiengängen nur in Kombination mit „Principles of Prog. Languages“*, dann zusammen 7.5 ECTS)~~
- □ Graphtransformationssysteme 2+2 SWS = 5 ECTS, SS
  Auch in Theorie wählbar!

*falls noch nicht im BA eingebracht.

Ein nettes und motiviertes Team freut sich auf Sie:

Ausführliche Informationen: **`www2.informatik.uni`**…

Persönlich im blauen Hochhaus, 5. Etage