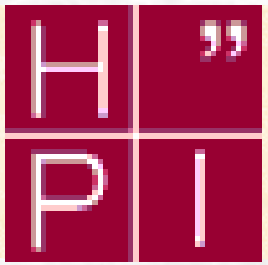


Aspektorientierte Programmierung mit Microsoft .NET

Dynamisches Aspekt-Weben

Wolfgang Schult

OSM – Operating Systems and Middleware

Hasso-Plattner-Institute Potsdam

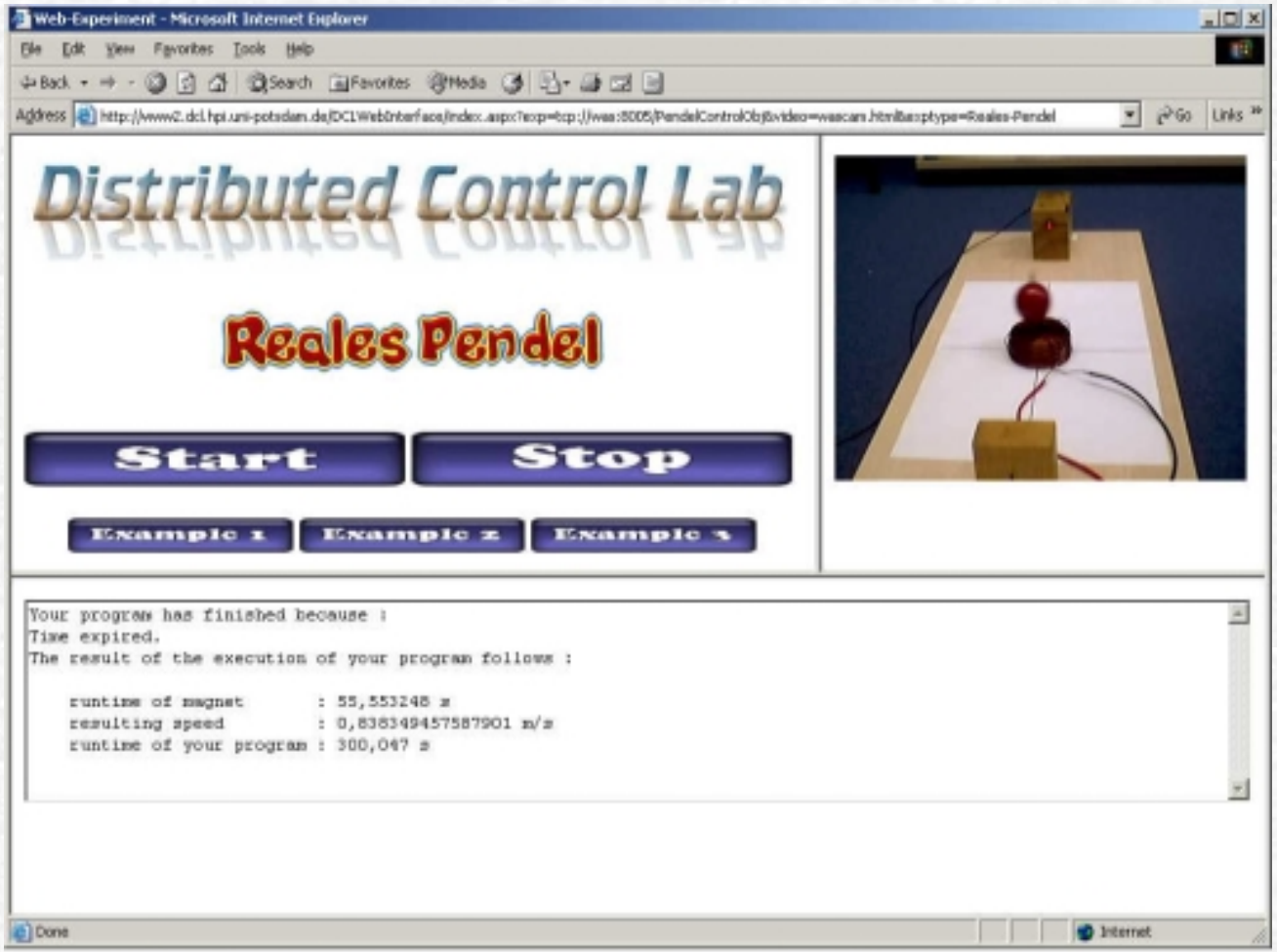
Überblick

1. Was machen wir…
2. Motivation
3. AOP
4. Dynamisches Aspekt-Weben
   - Idee
   - Performancemessungen
5. Aspekte im Einsatz
   - Dynamische Ressourcenoptimierung
   - Ergebnisse
6. Zusammenfassung

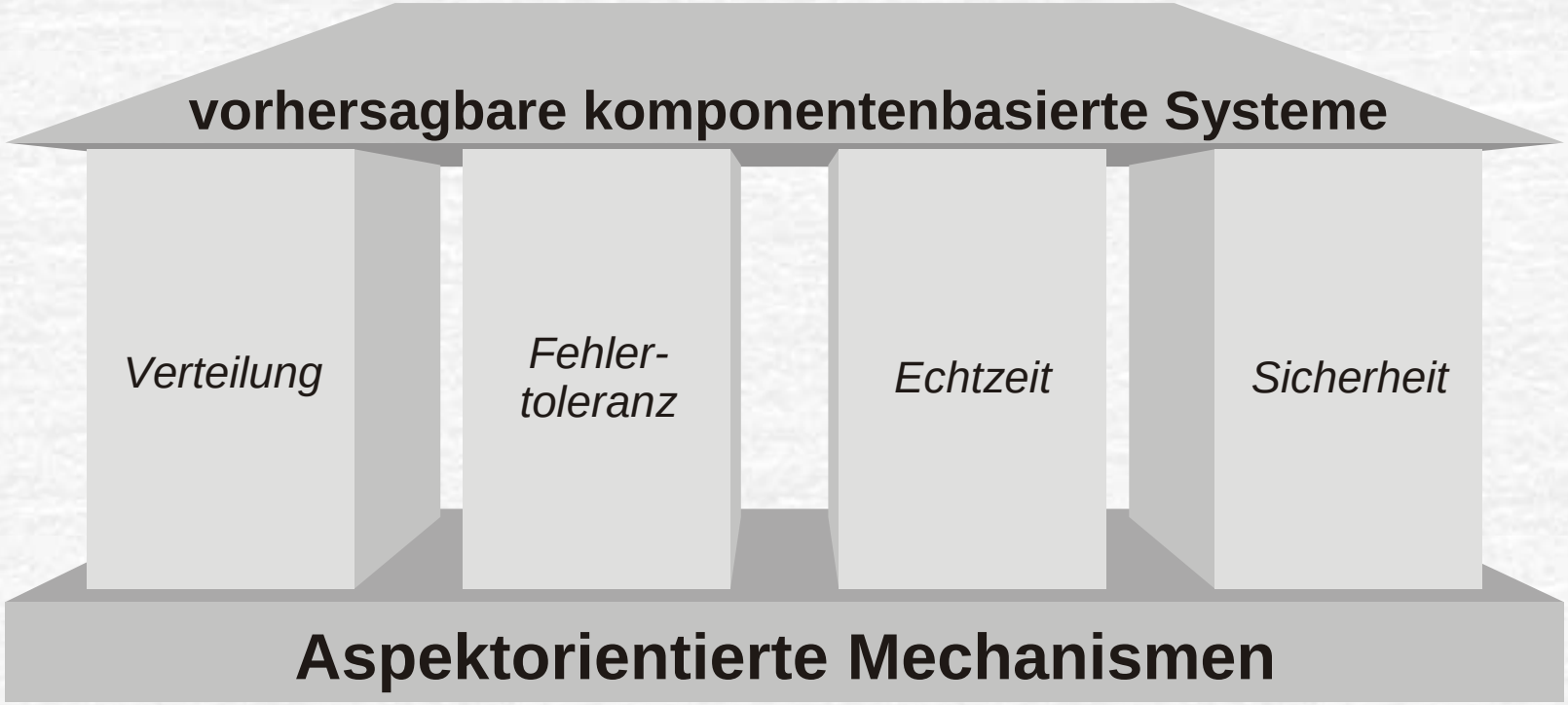
Distributed Control Lab – ein konfigurierbares Robotersystem

- Verschiedene Roboterkonfigurationen
  - Verschiedene Aktoren/Sensoren
  - Exprimente mit Kontrollalgorithmen
- Koordinierte Aktionen von mehreren Robotern
- Dynamische (Re-) Konfiguration als Sicherheitsmechanismus

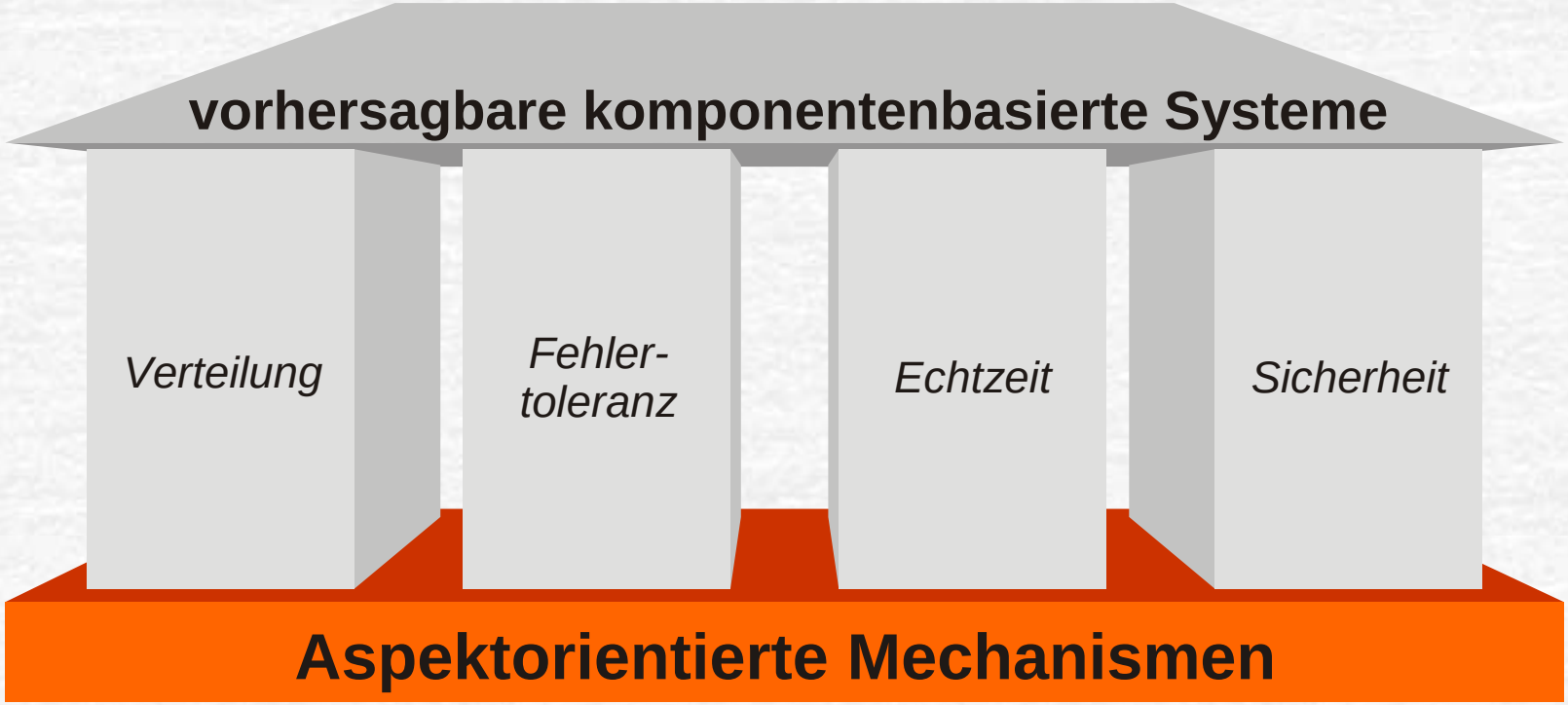
Distributed Control Lab – Physikalisches Pendel



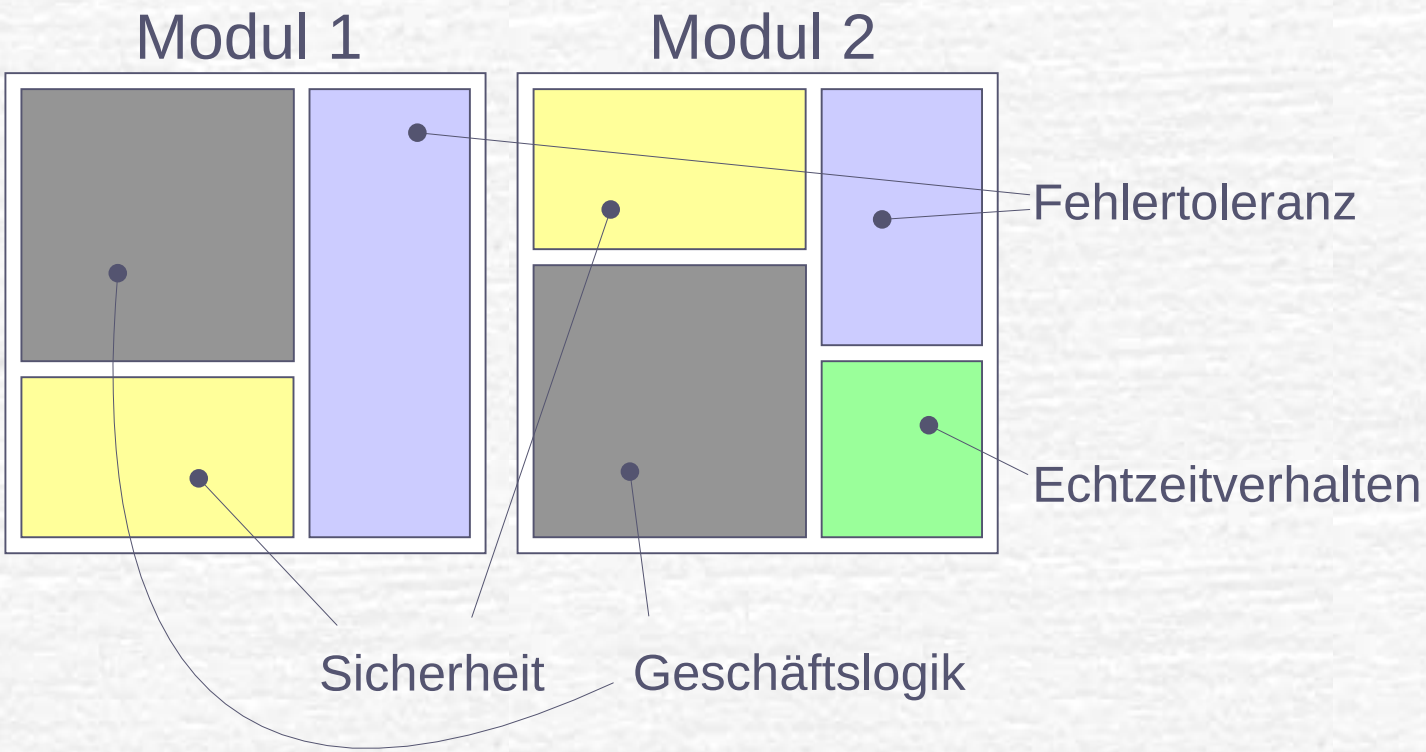
Motivation


vorhersagbare komponentenbasierte Systeme
Verteilung
Fehler-
toleranz
Echtzeit
Sicherheit
Aspektorientierte Mechanismen

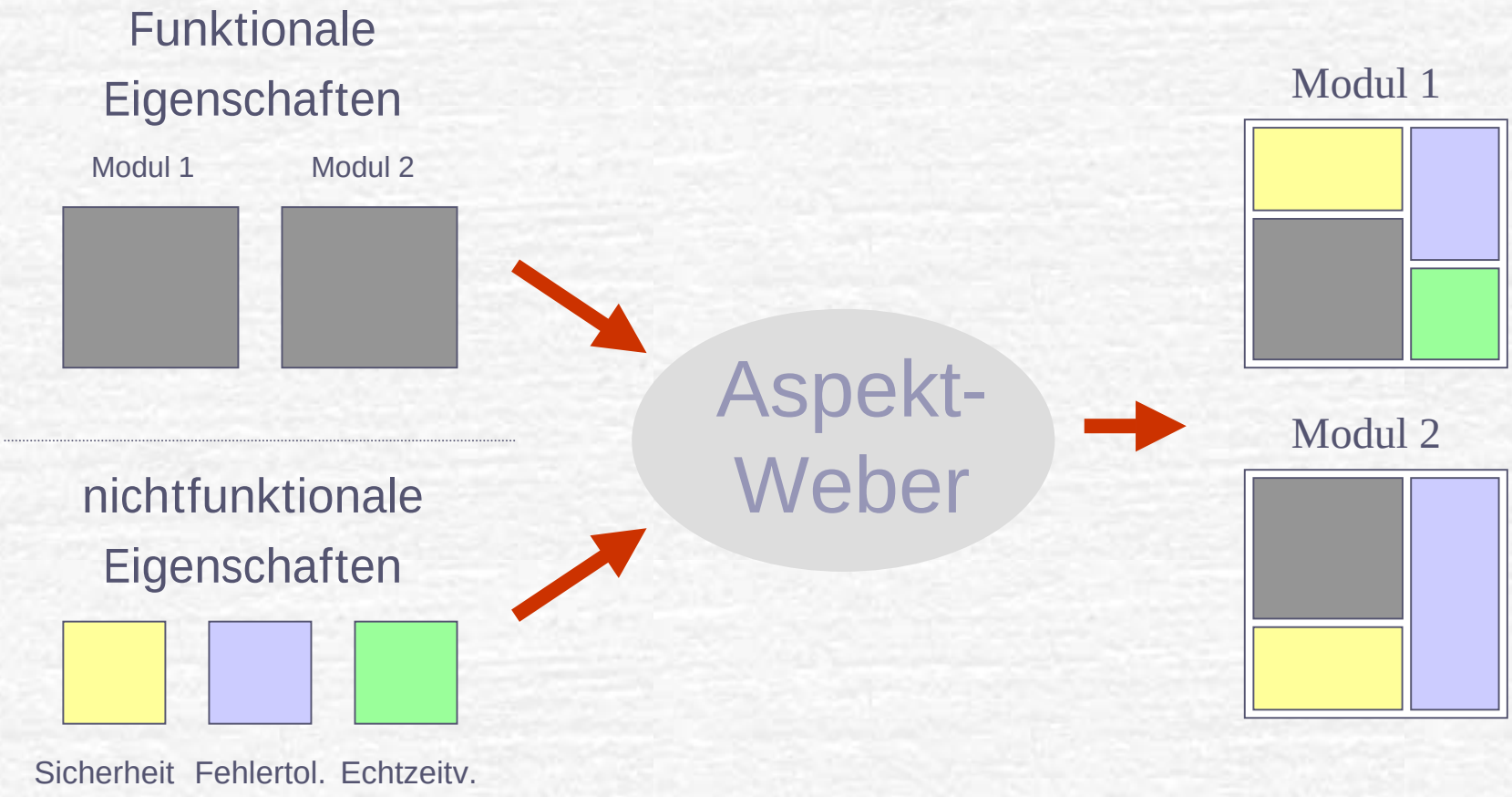
Motivation


vorhersagbare komponentenbasierte Systeme
Verteilung
Fehler-
toleranz
Echtzeit
Sicherheit
Aspektorientierte Mechanismen

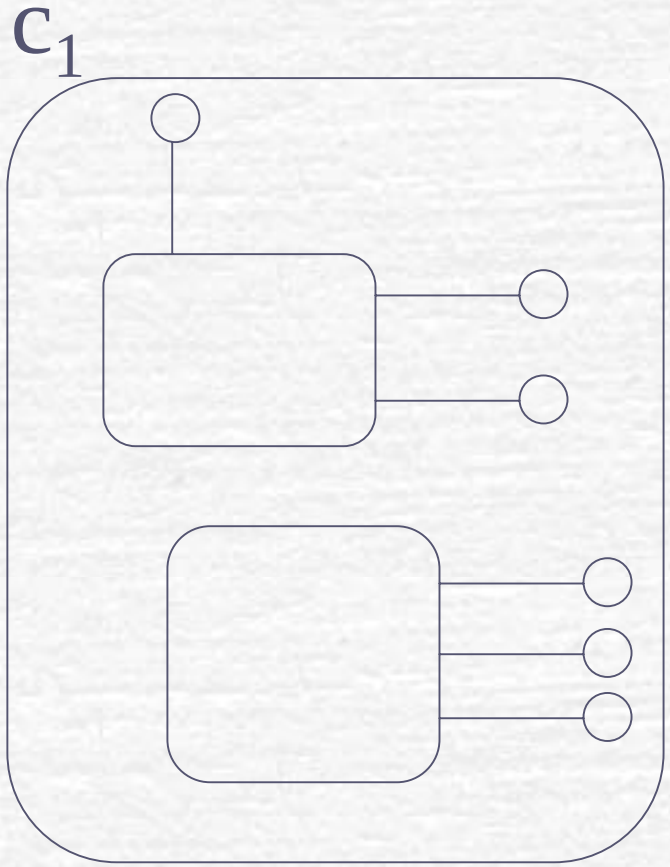
nichtfunktionale Eigenschaften



Lösung


Funktionale Eigenschaften
Modul 1
Modul 2
nichtfunktionale Eigenschaften
Sicherheit
Fehlertol.
Echtzeitv.
Aspekt-Weber
Modul 1
Modul 2

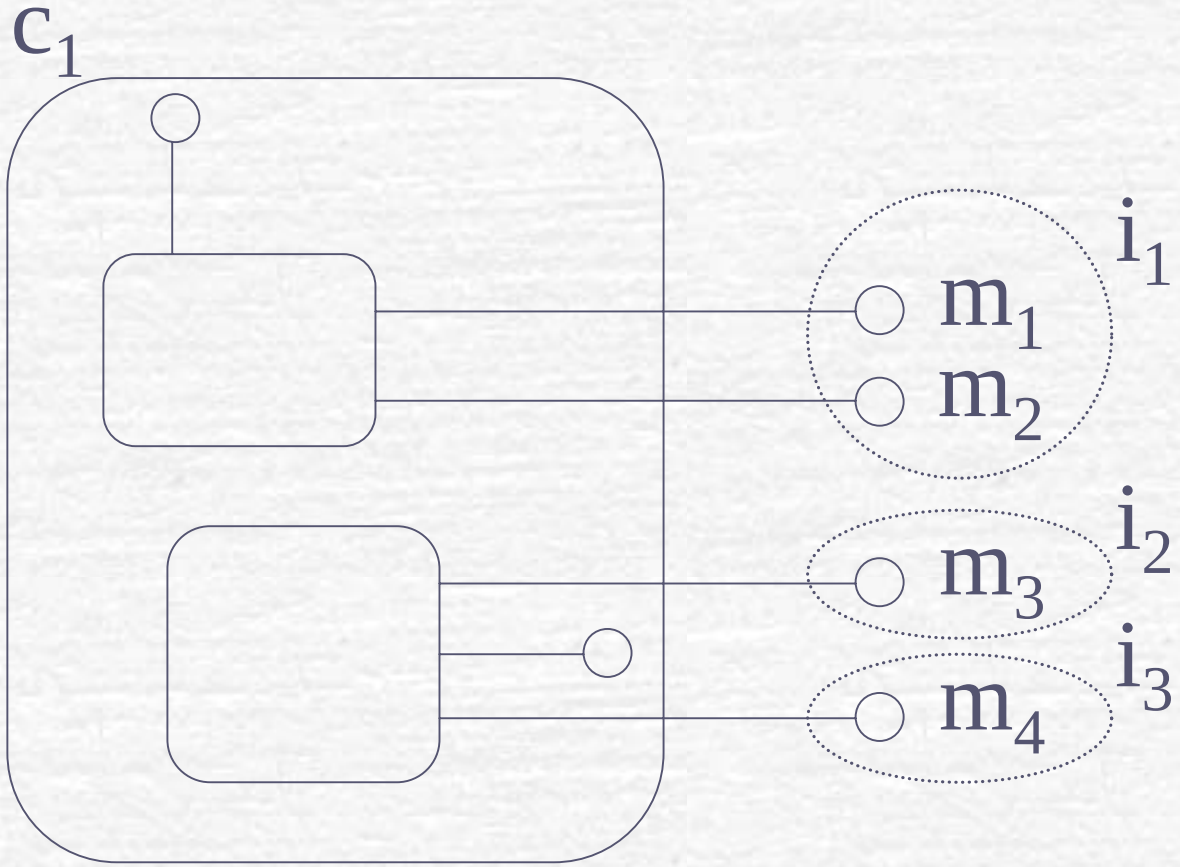
Begriffe



Eine Klasse implementiert Methoden, Felder und Properties

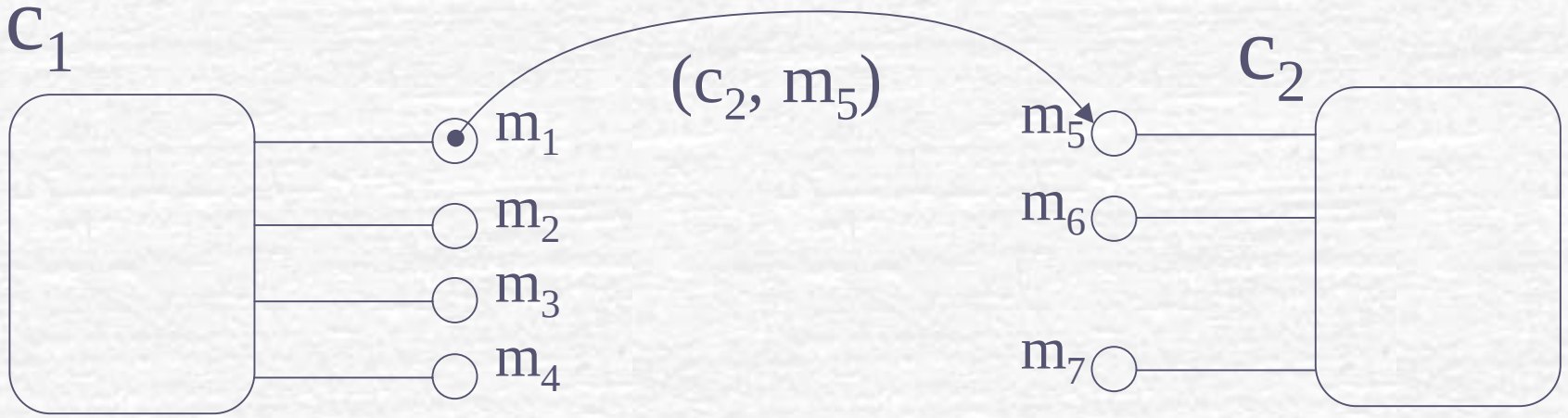
Eine Komponente enthält im allgemeinen mindestens eine Klasse

Begriffe



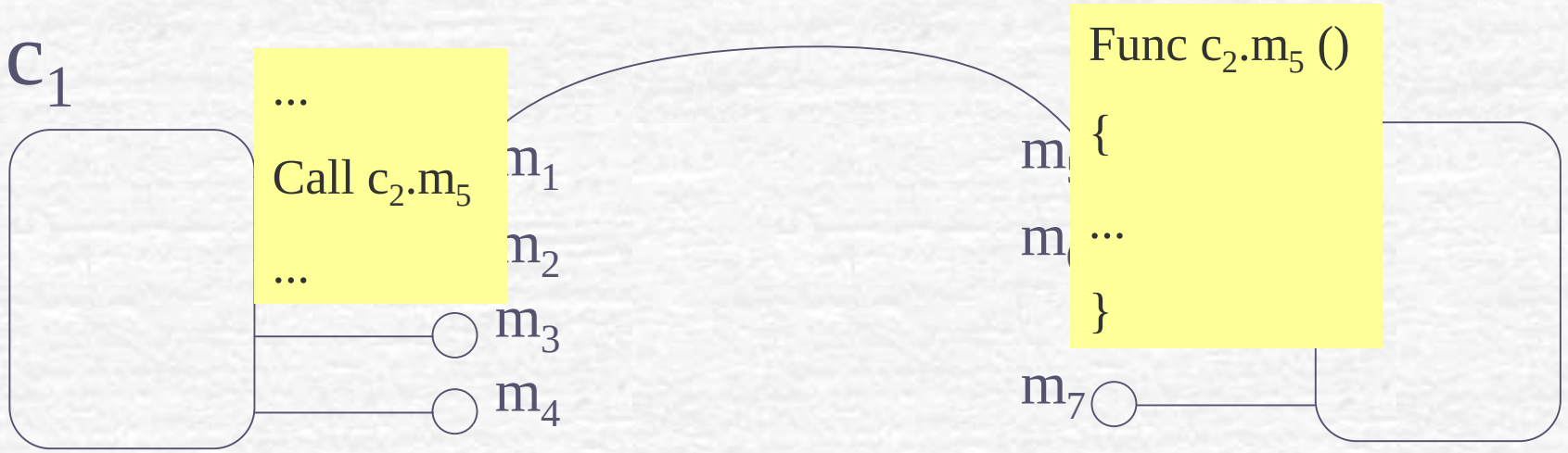
Eine Komponente macht Ihre Methoden durch Interfaces nach außen sichtbar

Kommunikation zwischen Komponenten (Klassen)



Der Aufruf einer Methode wird durch das Paar $(c_d, m_{d,k})$ $c_d \in C$, $m_{d,k} \in M_d$ beschrieben

Verweben von Aspektcode



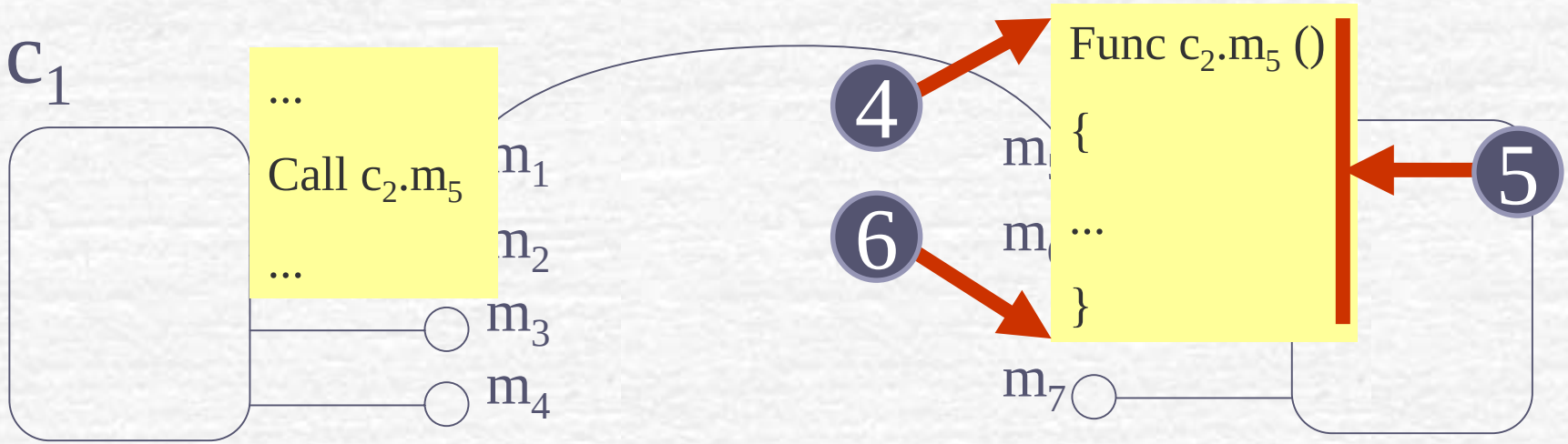
Potentielle *Verwebungspunkte* für *Aspektcode* sind Stellen, an denen Komponenten in Interaktion miteinander treten

Verweben von Aspektcode

Im Kontext der aufrufenden Kompenente:

1. Vor dem Methodenaufruf
2. Anstelle des Methodenaufrufes
3. Nach dem Methodenaufruf

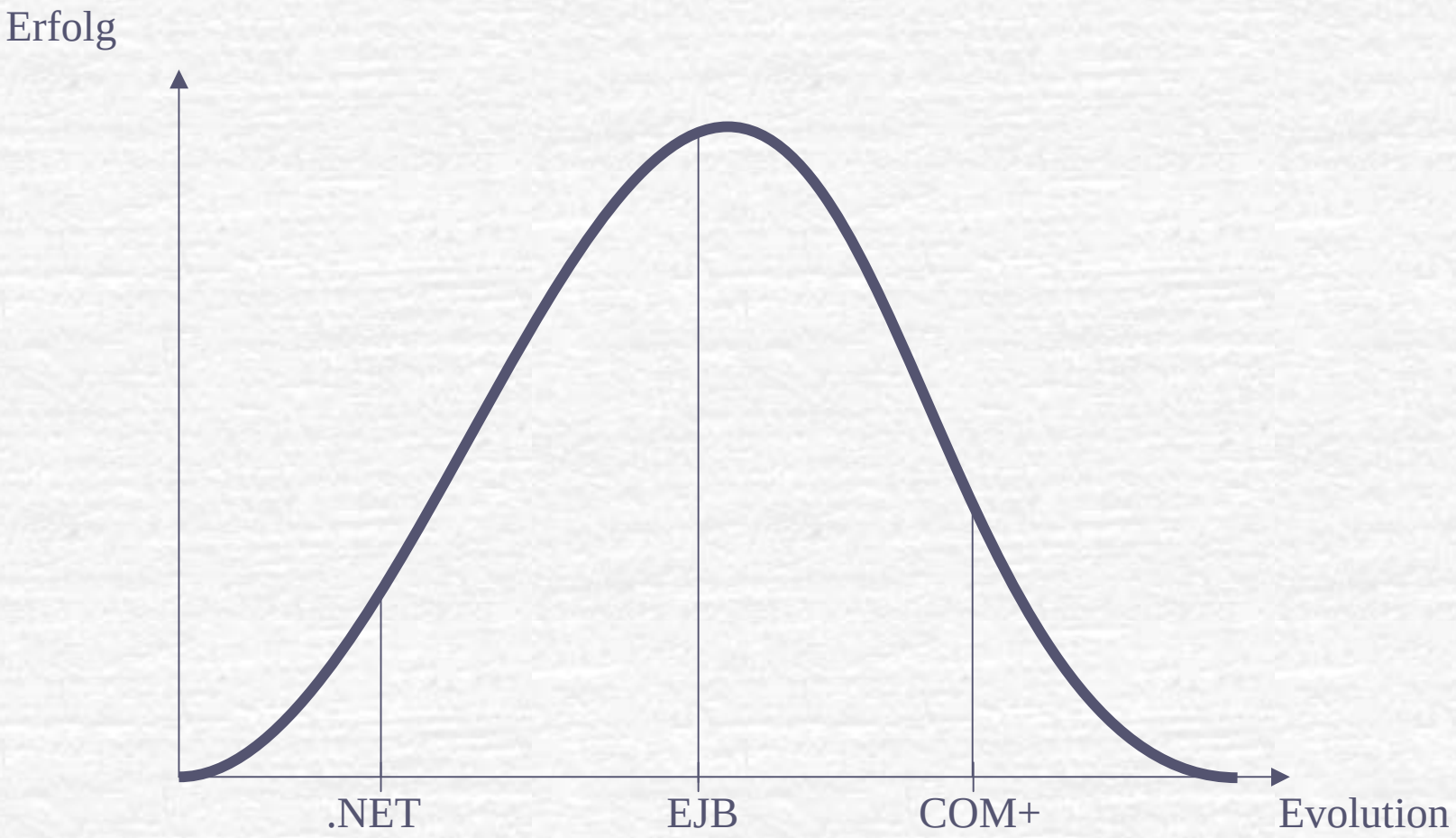
Verweben von Aspektcode



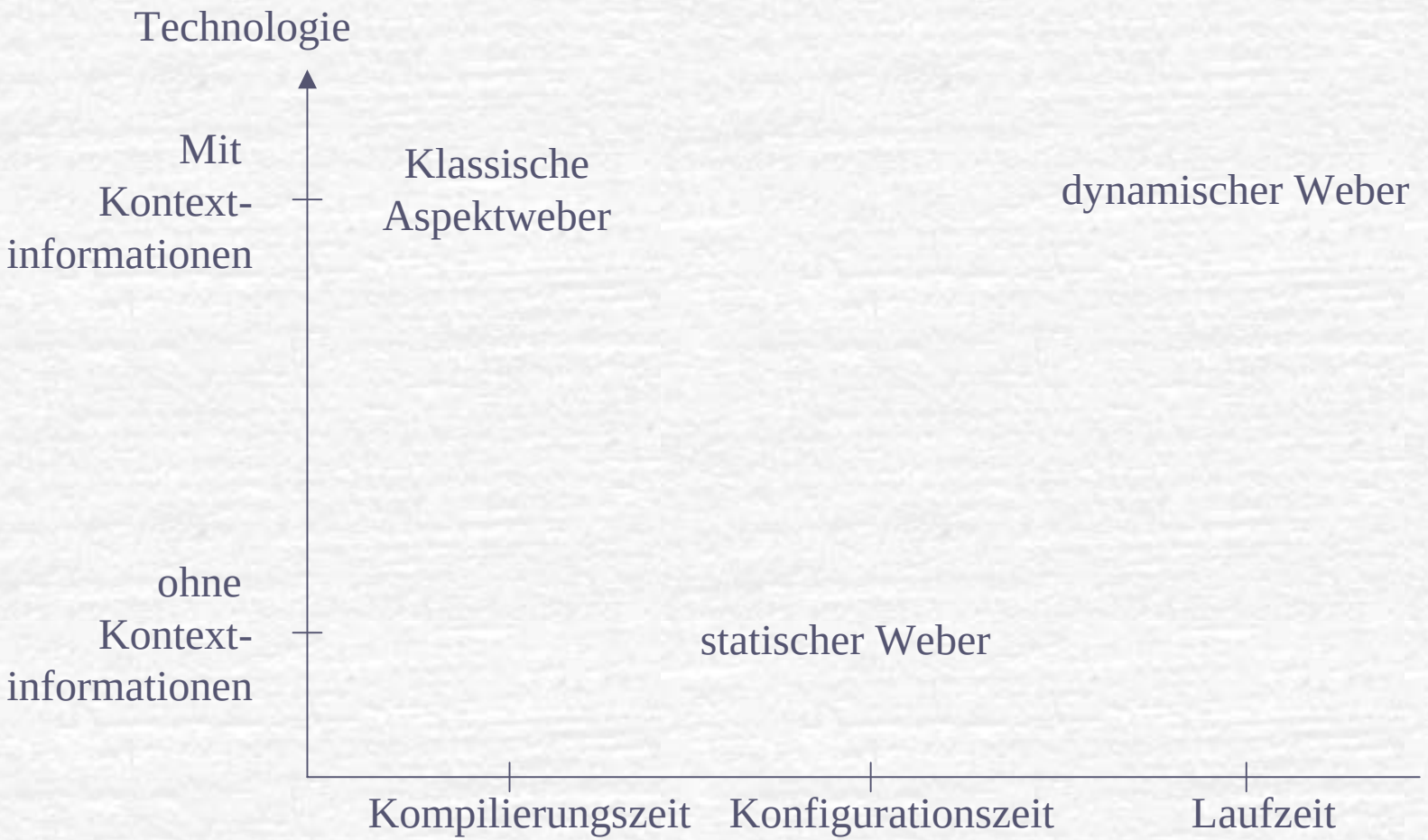
Im Kontext der aufgerufenden Kompenente:

4. Vor Eintritt in die Methode
5. Anstelle der Methode
6. Nach Verlassen der Methode

Warum .NET?



Implementationsraum von Aspektwebern



Idee...

- Weber für Microsoft .NET auf Basis von Reflektionsinformationen
- Vorteil:
  - Verwebung kann zur Konfigurationszeit oder zur Laufzeit vorgenommen werden
  - Komponenten können in binärform vorliegen
  - Sprachunabhängig (kein Parser benötigt)

Dynamisches Aspektweben

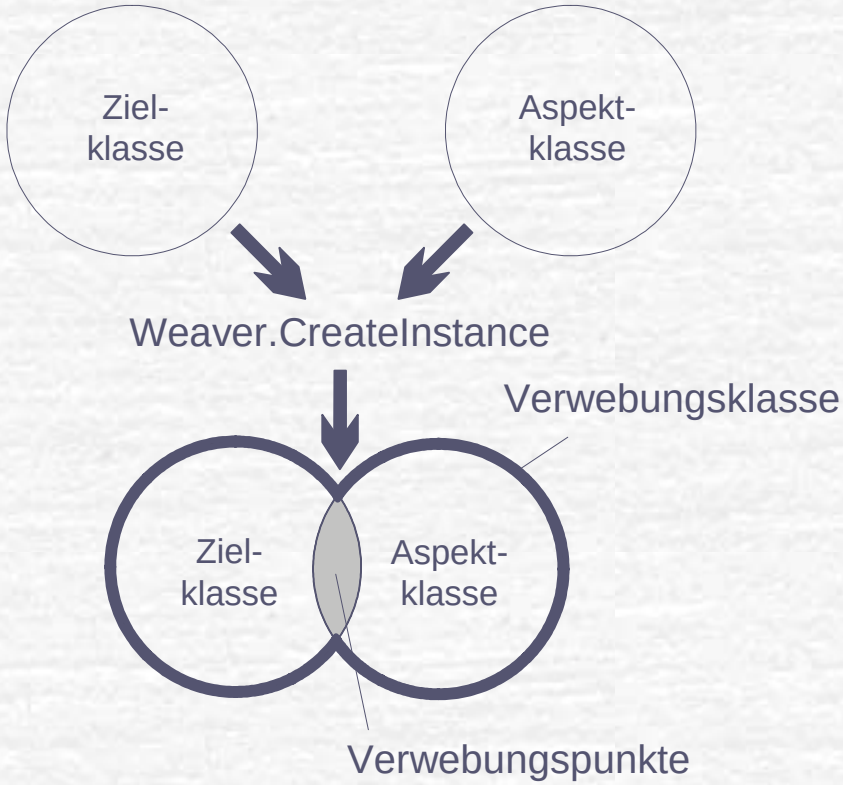
- Aspektweben normalerweise zur Kompilierungszeit
- Schwierig bei konträren Aspekten
  - Anforderungen in Desingphase müssen zur Laufzeit nicht mehr gültig sein
  - Für manche Konfigurationen werden bestimmte Aspekte nicht benötigt

Dynamisches Aspektweben

- Entscheidung zur Laufzeit
  - Verwebung läuft erst bei der Instanzierung der Komponente
  - Komponente kann mit Aspektinformationen ausgeliefert werden, die Entscheidung kann aber auch erst zur Laufzeit erfolgen
- Ressourcennutzung kann optimiert werden
  - Wenn für bestimmte Szenarien nicht benötigt

Dynamisches Aspektweben

- Eine ***Aspektklasse*** wird mit einer ***Zielklasse*** durch die ***Weaver*** Komponente verwoben
- ***Weaver.CreateInstance*** gibt ein Objekt der ***Verwebungsklasse*** zurück
- Der Weber verwebt beide Klassen an definierten ***Verwebungspunkten***



Eine Aspektklasse

```
public class Trace:Aspect
{
   [call(Invoke.Before)]
   Public void m5()
   {
   // Aspektcode
   }
}
```

Der Aspektcode soll vor der Ausführung der Methode m5 eingewoben werden

Eine Aspektklasse

```
public class Trace:Aspect
{
   [call(Invoke.Instead,
   Alias="m*")]
   Public object
   traceall(params
   object[] args)
   {
   // Aspektcode
   }
}
```

Der Aspektcode soll anstelle der Ausführung jeder Methode deren Name mit m beginnt mit beliebiger Signatur eingewoben werden

(Beschreibung durch Benutzung von Signatur-Wildcards)

Eine Aspektklasse

```
public class Trace:Aspect
{
   [call(Invoke.Instead,
   Alias="m*")]
   Public object
   traceall(params
   object[] args)
   {
      Context.Invoke(args)
   }
}
```

Der Aufruf der originalen Funktion ist über den Aspektkontext möglich

Wo ist die Komponente?

Entweder:

```
[Trace]
Public class c2
{
   // …
}
```

Oder:

```
C2 c=
Weaver.CreateInstance(
  typeof(c2), null,
  new Trace());
```

- Die Aspektklasse kann statisch als Attribut vor die Zielklasse geschrieben werden
- Dem Weber kann ein Aspektobjekt dynamisch zur Laufzeit übergeben werden

Performancemessungen

- Overhead durch Weben (Instantiierung)
  - Mit verschiedenen Aspekten und ohne Aspekt
  - Unterschiedlich komplexe Aspekte
- Overhead bei Methodenaufrufen
  - Funktionsmix (12 Zuweisungen, 8 Kontrollflussanweisungen, vier Methodenaufrufe)
  - Jeder vierte mit Aspekt verwoben
  - Verwebung *anstelle* und *vor* Ausführung der Methode
  - Variable Parameteranzahl
- Testumgebung:
  - Dual Pentium III 256 MB RAM Windows 2000(MP)
- Messungen mit einem Testprogramm
  - insgesamt 500 Durchläufe
  - Mit High-Resolution-Counter

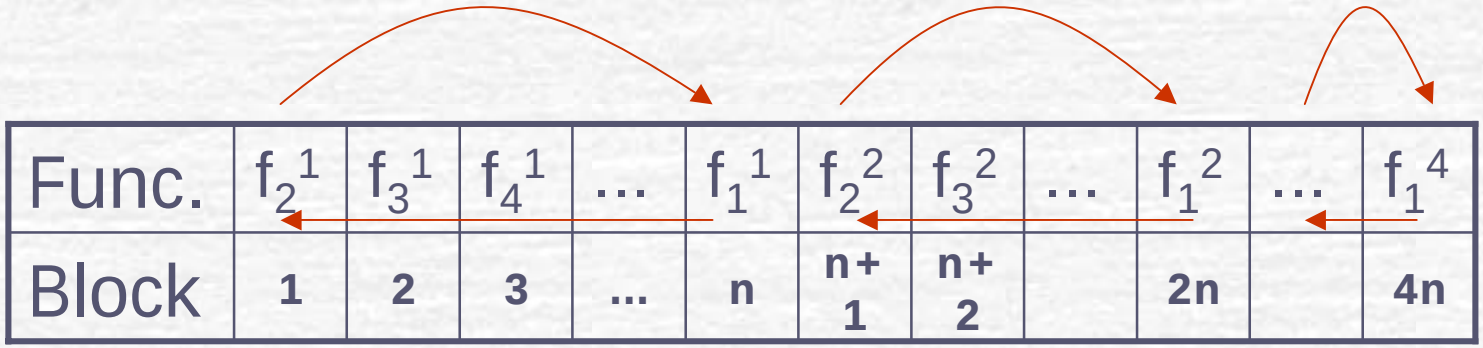
Performancemessungen

- Testprogramm:
  - Aufteilung in Blöcke
  - Durch Rotieren der Testfunktionen
  - Jede Testfunktion 4 mal wiederholt

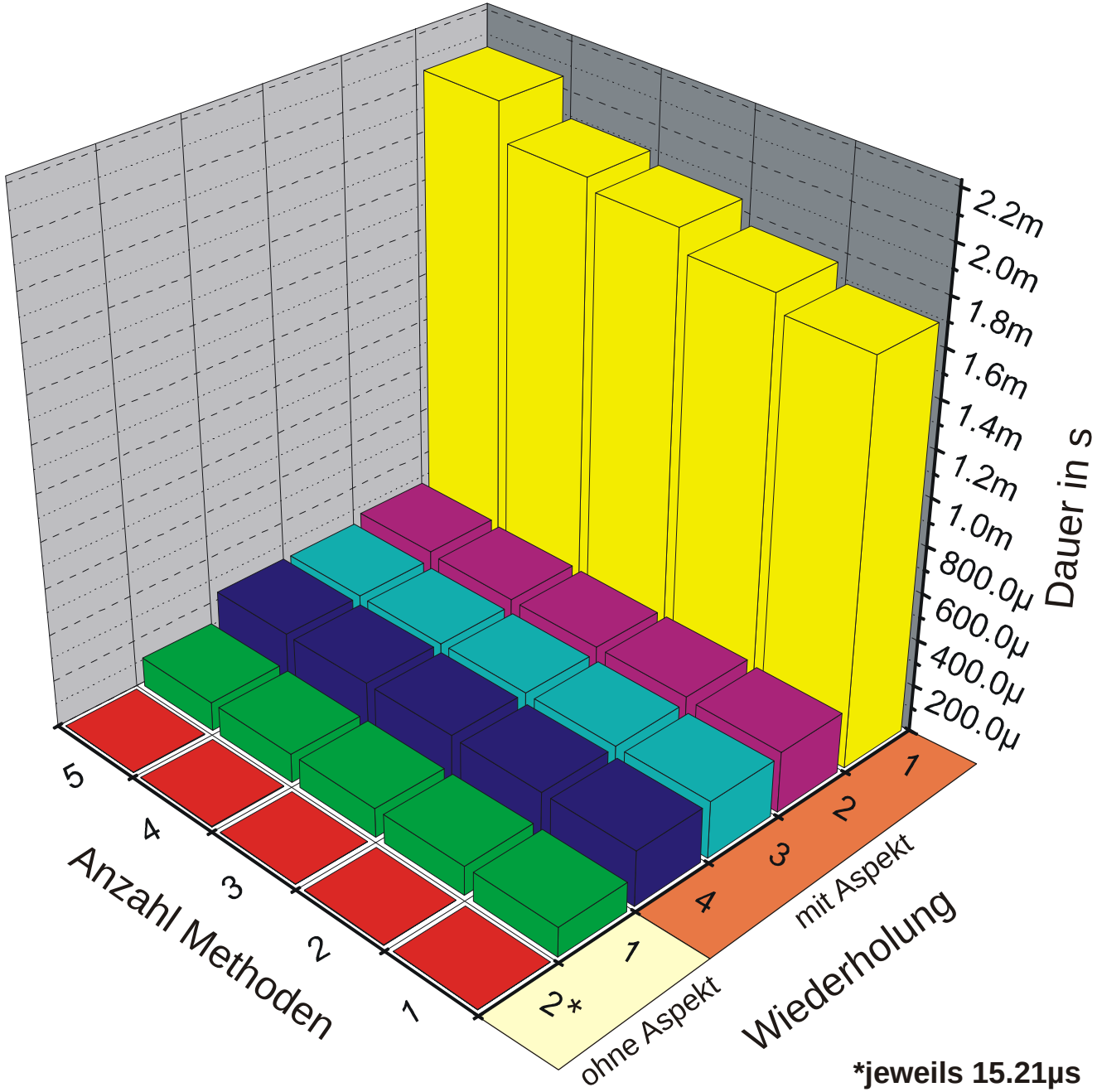
Func.	f_1^1	f_2^1	f_3^1	…	f_n^1	f_1^2	f_2^2	…	f_n^2	…	f_n^4
Block	1	2	3	…	n	n+1	n+2		2n		4n

Performancemessungen

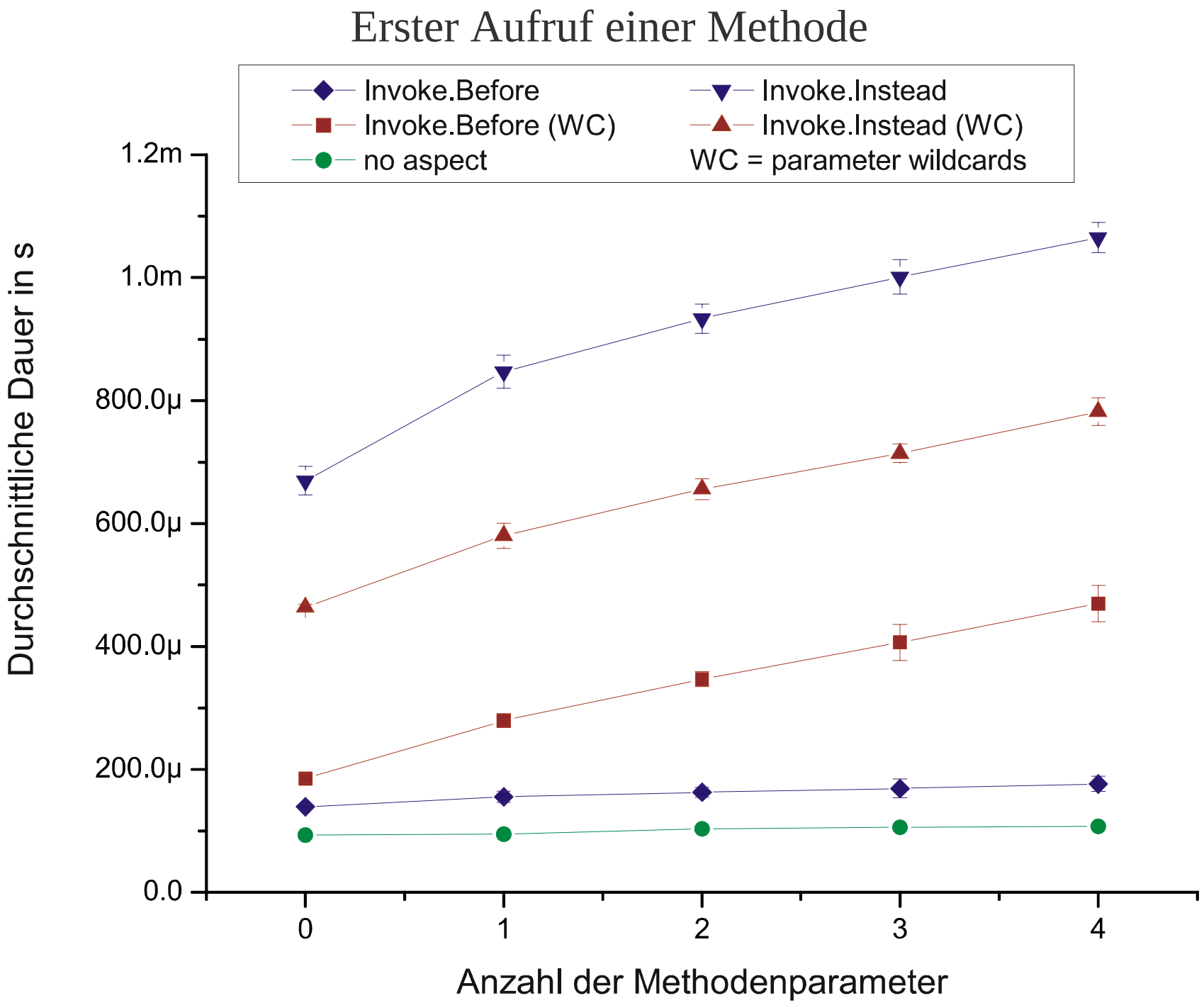
- Testprogramm:
  - Aufteilung in Blöcke
  - Durch Rotieren der Testfunktionen
  - Jede Testfunktion 4 mal wiederholt

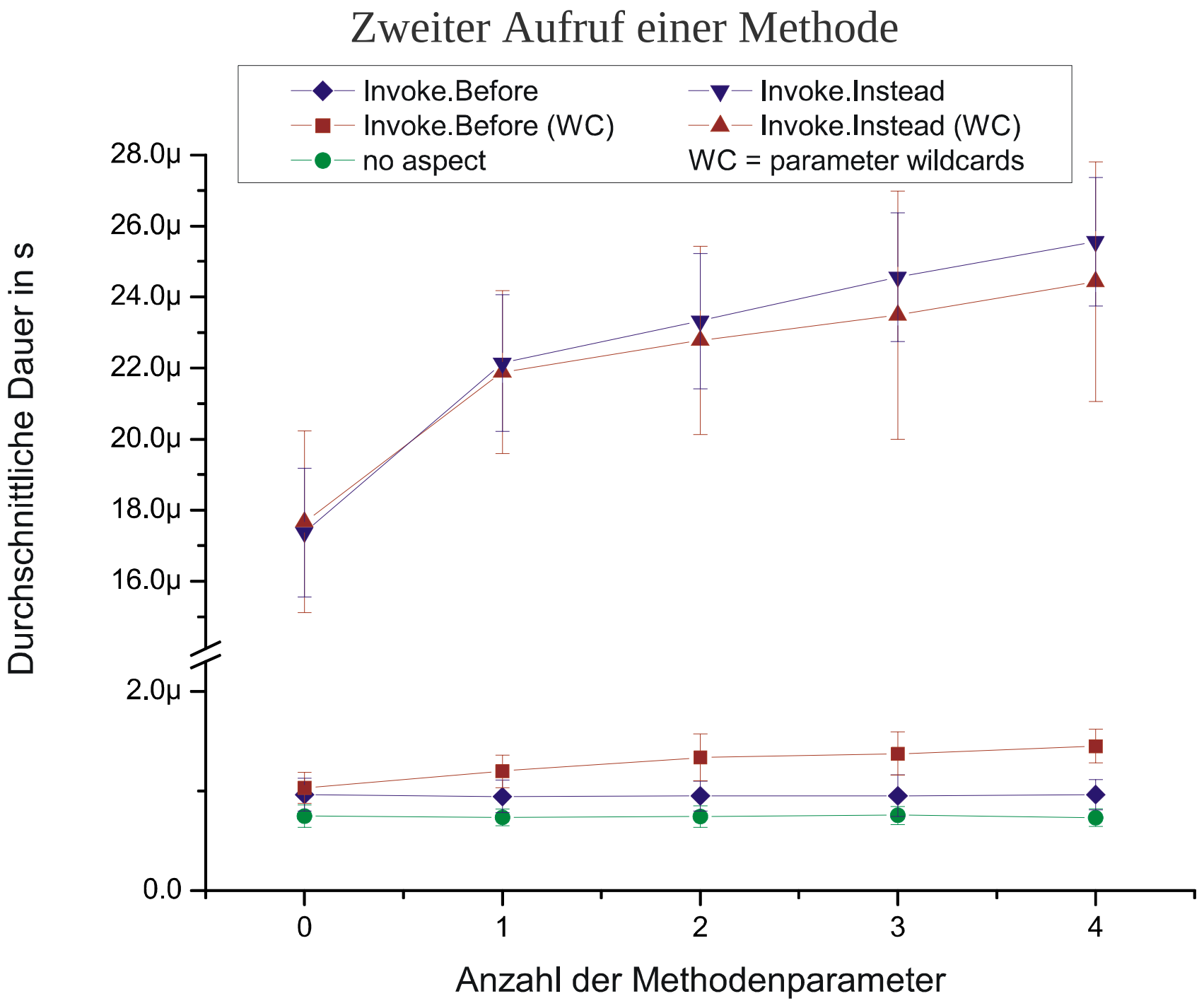


Durchschnittliche Dauer der Instantiierung mit und ohne Aspekt

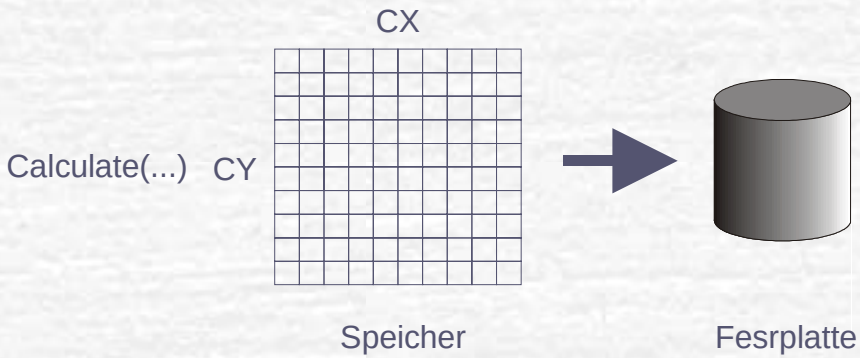


***jeweils 15.21µs**


Erster Aufruf einer Methode
Invoke.Before
Invoke.Instead
Invoke.Before (WC)
Invoke.Instead (WC)
no aspect
WC = parameter wildcards
Durchschnittliche Dauer in s
1.2m
1.0m
800.0µ
600.0µ
400.0µ
200.0µ
0.0
0
1
2
3
4
Anzahl der Methodenparameter


Zweiter Aufruf einer Methode
Invoke.Before
Invoke.Before (WC)
no aspect
Invoke.Instead
Invoke.Instead (WC)
WC = parameter wildcards
Durchschnittliche Dauer in s
28.0µ
26.0µ
24.0µ
22.0µ
20.0µ
18.0µ
16.0µ
2.0µ
0.0
0
1
2
3
4
Anzahl der Methodenparameter

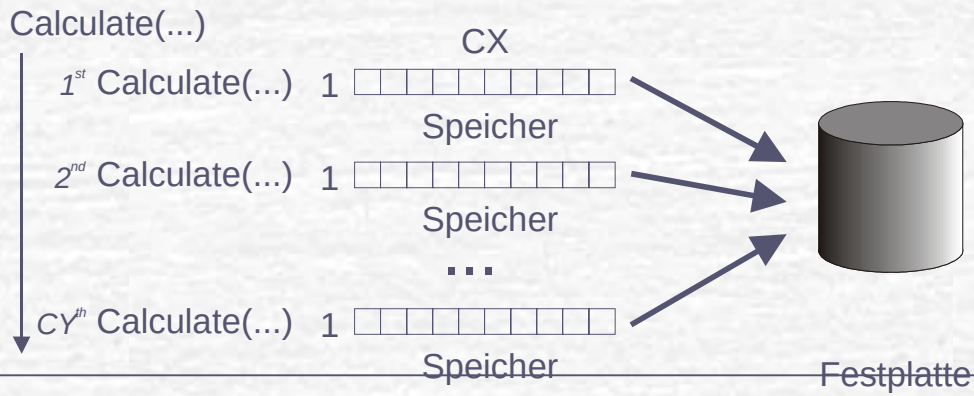
Eine verteilte Mandelbrot-Berechnung



- AOP wird benutzt
  - Um Verteilung von Daten und Berechnung zu kontrollieren
  - Benutzung des Hauptspeichers zu optimieren
- Beide Aspekte haben konträre Ziele
- Tradoffs und eingesetzte Umgebung sind erst zur Laufzeit bekannt

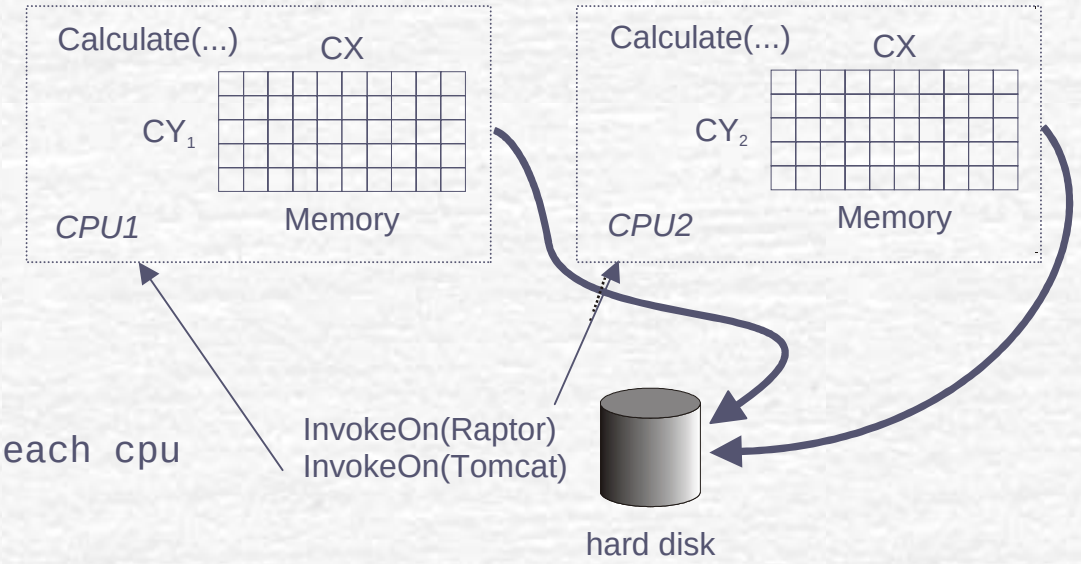
Mandelbrot: Der Speicher-Aspekt

```
public class Memory : Aspect {
   [Call(Invoke.Instead)] // connection point
   public void Calculate(string filename, double x1, double y1,
        double x2, double y2, int xRes, int yRes)
   {
       double dStep=(y2-y1)/((double)yRes); // split up in lines
       for(int i=0;i<yRes;i++) {  // call original function
        Context.Invoke(filename+i.ToString(),x1,y1,x2,y1,xRes,1);
        y1+=dStep;
           ….
```



Mandelbrot: Der Verteilungs-Aspekt

```
public class Distribute:Aspect
{
// instances on other cpu's
private object[] instances;
public override void ctor(
   Weaver weaver,object o,
   object[] args)
{
// Create remote instances for each cpu
}
// the connection point
[Call(Invoke.Instead]
public void Calculate(string filename, double x1, double y1, double x2,
   double y2, int xRes, int yRes)
{
   /* Create work item for each cpu */
}
```



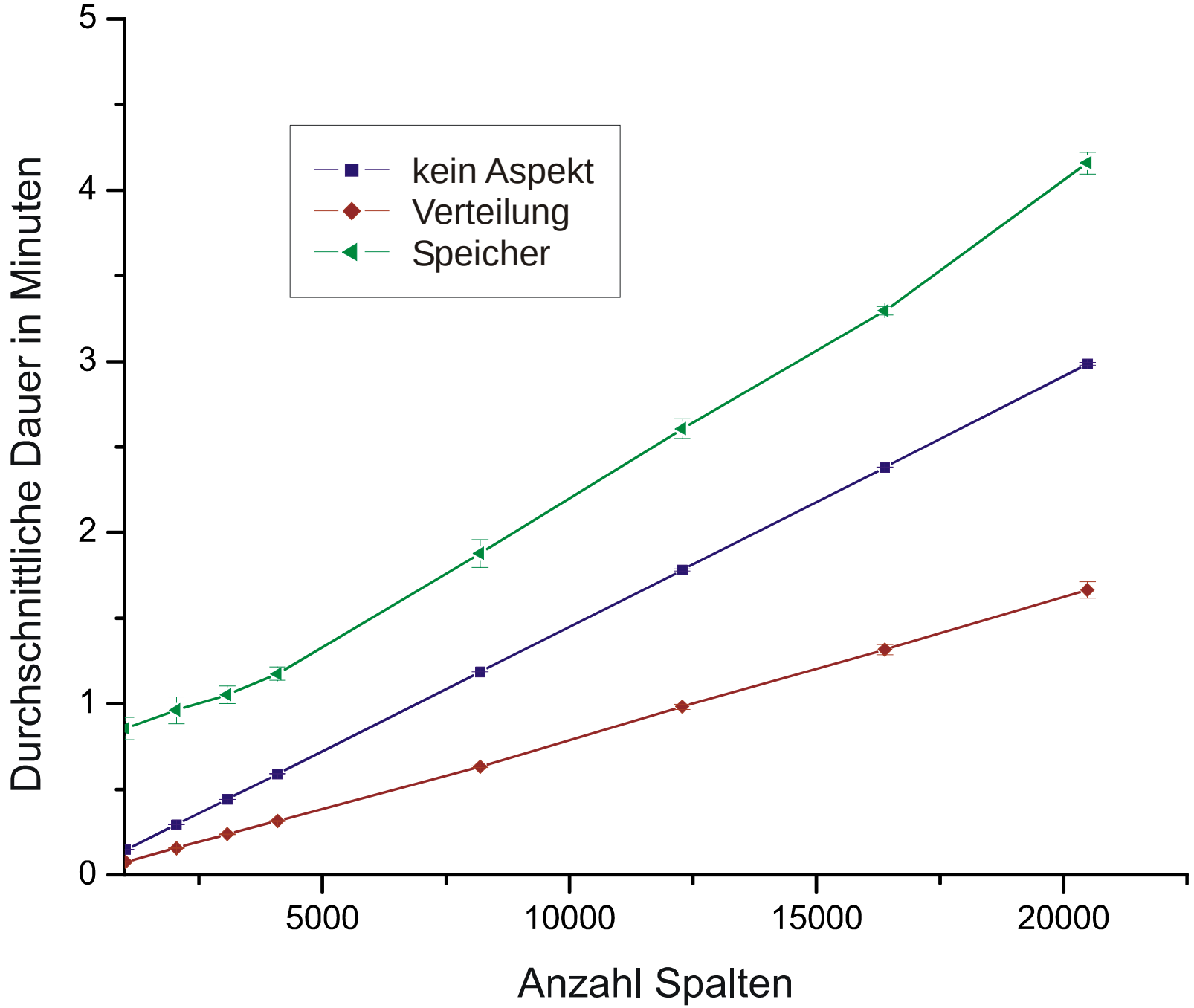
Mandelbrot: Instantiierung

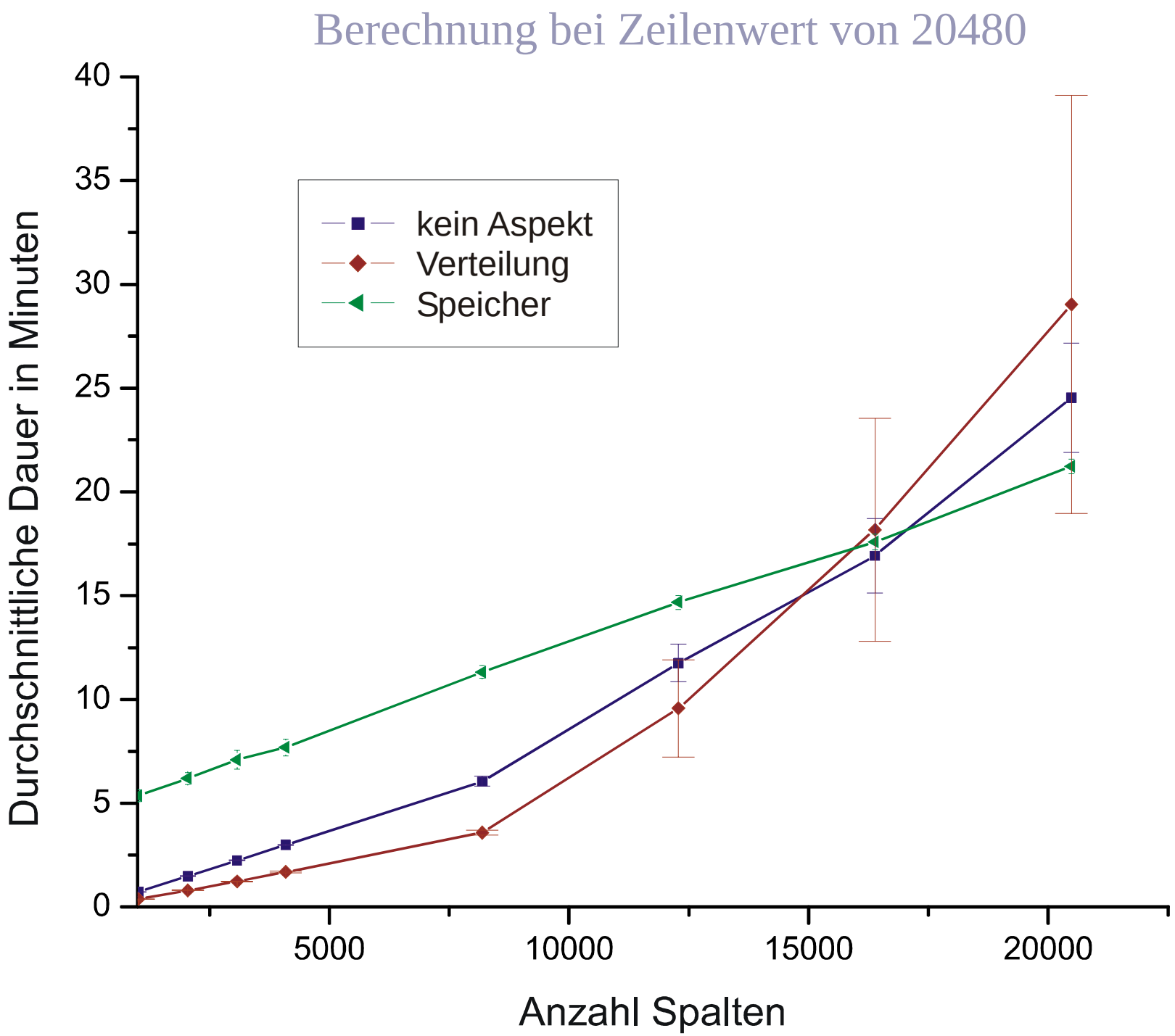
```
Mandelbrot mb;
if(opt_memory.Checked)
   mb=Aspects.Weaver.CreateInstance(typeof(Mandelbrot), null,
   new SaveMemory()) as Mandelbrot;
else if(opt_speed.Checked)
   mb=Aspects.Weaver.CreateInstance(typeof(Mandelbrot), null,
   new Distribute(filename.Text)) as Mandelbrot;
else
    mb=new Mandelbrot();
```

Ergebnisse

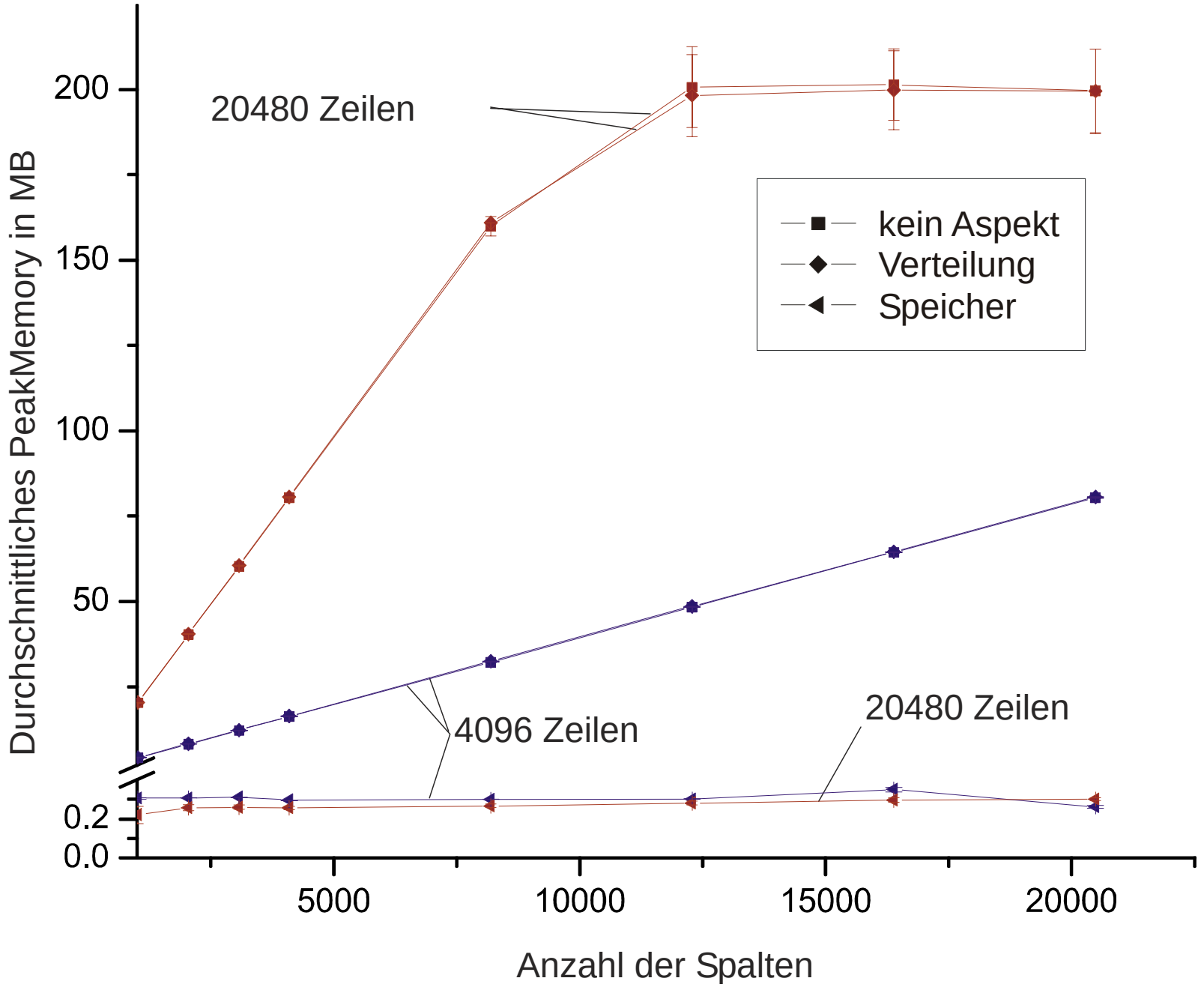
- Testumgebung
  - Dual Pentium III 256 MB RAM Windows 2000(MP)
- Verschiedene Auflösungen von 256*256 bis 20480*20480 ohne und jeweils mit einem der beiden Aspekte
- Gemessen wurde:
  - Dauer der Berechnung
    - DateTime.Now (Auflösung 10ms)
  - Physikalischer Speicherverbrauch
    - PeakWorkingSet
- jeweils 15 Messungen pro Testfall

Berechnung bei Zeilenwert von 4096




Berechnung bei Zeilenwert von 20480
Durchschnittliche Dauer in Minuten
0
5
10
15
20
25
30
35
40
5000
10000
15000
20000
Anzahl Spalten
kein Aspekt
Verteilung
Speicher

Durchschnittlicher Speicherverbrauch



Zusammenfassung

- Dynamisches Weben erlaubt die späte Bindung von Aspektcode und funktionalen Code
- Ob ein eine Komponente mit Unterstützung eines Aspektes geladen werden soll oder nicht, muß erst zur Laufzeit entschieden werden
- Implementation im Kontext von .NET mit sprachinhärenten Mitteln (als Bibliothek)
- Funktioniert mit allen .Net-Sprachen

www.discourse.de

www.dcl.hpi.uni-potsdam.de

Weitere Publikationen

- Aspect-Oriented Programming with C# and .NET. ISORC 2002, Washington, USA, April 29 – May 1 2002.
- Aktuelle Arbeiten unter:

http://www.dcl.hpi.uni-potsdam.de/projects

www.discourse.de

www.dcl.hpi.uni-potsdam.de

Fragen ?

www.discourse.de

www.dcl.hpi.uni-potsdam.de