Dynamisches Aspektweben in C und C++: DAO C++ und µDyner

Referent: André Zweschke

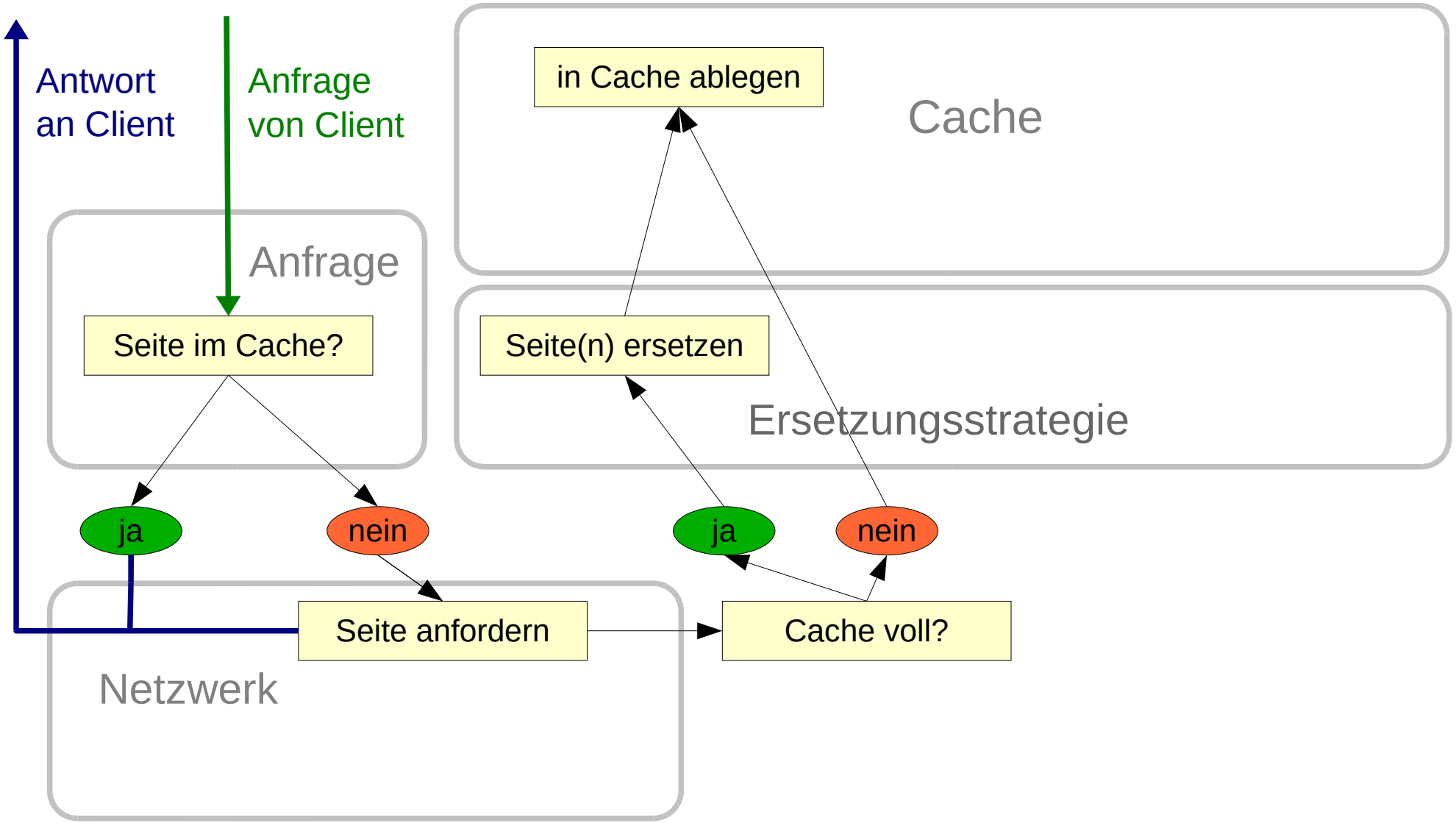
Inhalt

1. Motivation: Web-Caches
2. Dynamic Aspect Oriented C++ (DAO C++)
3. µDyner
    1. für C
    2. Erweiterung für C++
4. Zusammenfassung und kritische Betrachtung

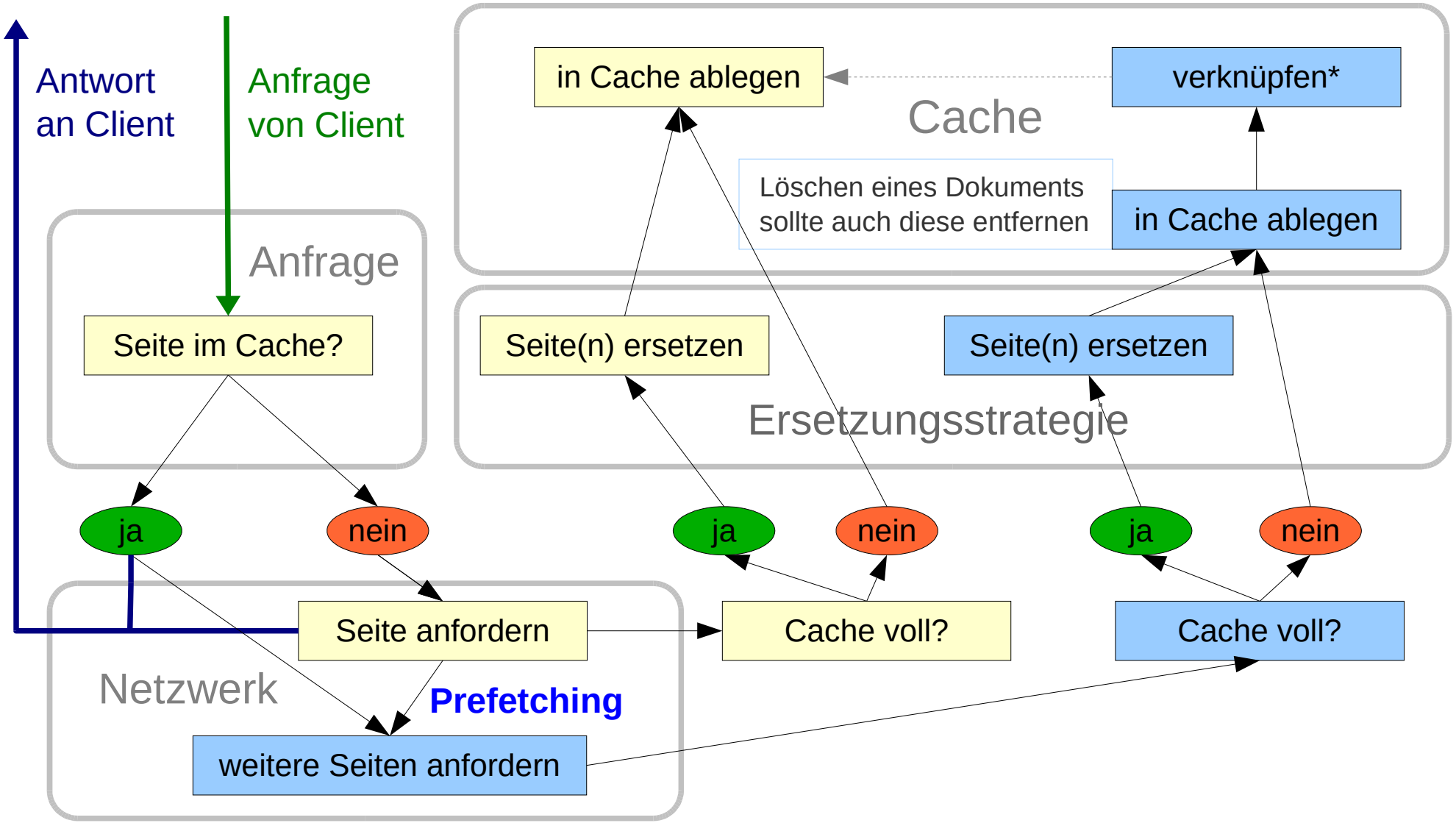
Motivation: Web-Caches

- helfen Bandbreite im Internet zu sparen
- Webseiten werden schneller bereitgestellt
- weitere Verringerung der Latenz durch Prefetching
  - ➔ die Daten sind dann schon im Cache wenn sie angefordert werden
- ➔ einfache Strategie: alle verlinkten Seiten anfordern
  - ✔ die Wartezeit für den Anwender nimmt ab
  - ✗ die Internet-Auslastung nimmt zu
- ➔ auf Nutzerverhalten und Webanwendungen zugeschnittener Algorithmus nötig

Web-Cache


Antwort
an Client
Anfrage
von Client
in Cache ablegen
Cache
Anfrage
Seite im Cache?
Seite(n) ersetzen
Ersetzungsstrategie
ja
nein
ja
nein
Seite anfordern
Cache voll?
Netzwerk

Web-Cache


Antwort
an Client
Anfrage
von Client
Anfrage
Seite im Cache?
ja
nein
Netzwerk
Seite anfordern
Prefetching
weitere Seiten anfordern
Cache
in Cache ablegen
verknüpfen*
Löschen eines Dokuments
sollte auch diese entfernen
in Cache ablegen
Ersetzungsstrategie
Seite(n) ersetzen
Seite(n) ersetzen
ja
nein
ja
nein
Cache voll?
Cache voll?

Aspektweben zur Laufzeit

- ➔ Prefetching-Algorithmus ist „*crosscutting concern*“
- AOP scheint am geeignetsten dafür
- Aber: es sollte möglich sein die Strategien zur Laufzeit auszutauschen
  - der Austausch muss sehr schnell erfolgen können
    - für den Nutzer unbemerkt bleiben
  - Web-Cache sind in C oder C++ geschrieben
- ➔ Aspekte für C oder C++
- ➔ müssen zur Laufzeit eingebunden werden können
- ➔ entsprechende Erweiterung der Sprache nötig

Dynamic Aspect Oriented C++

DAO C++

DAO C++

- Komponenten:
  - Präprozessor
    - erzeugt zum dynamischen Aspektweben benötigten Code
  - Standard C++ Compiler
  - AOP-Engine
    - Erzeugung der Zielpunkte zur Laufzeit
    - Aspektweben und -auflösung
    - Advice-Ausführung

Ein Programm in DAO C++

- Ein Aspekt:

```
class NewAspect : public Aspect {
   void advice() {
      advice code
   }
}
```

  - wird von Klasse „Aspekt“ abgeleitet
  - Instanziierung:

```
NewAspect na;
na.setExpression("class_sign, method_sign()")
na.setBefore(true);
na.setAfter(false);
```

  - `setBefore()` und `setAfter()` geben Typ der Advice-Ausfühung an

Aspektweben in DAO C++

- Instanz der AOP_Engine muss erzeugt werden:
  - Ausführung der gewünschen Aktion (z.B. weave)

```
AOP_Engine aop_engine;
aop_engine.weave(&na);

aop_engine.unweave(&na);
aop_engine.refresh(&na);
```

Der DAO C++ Präprozessor

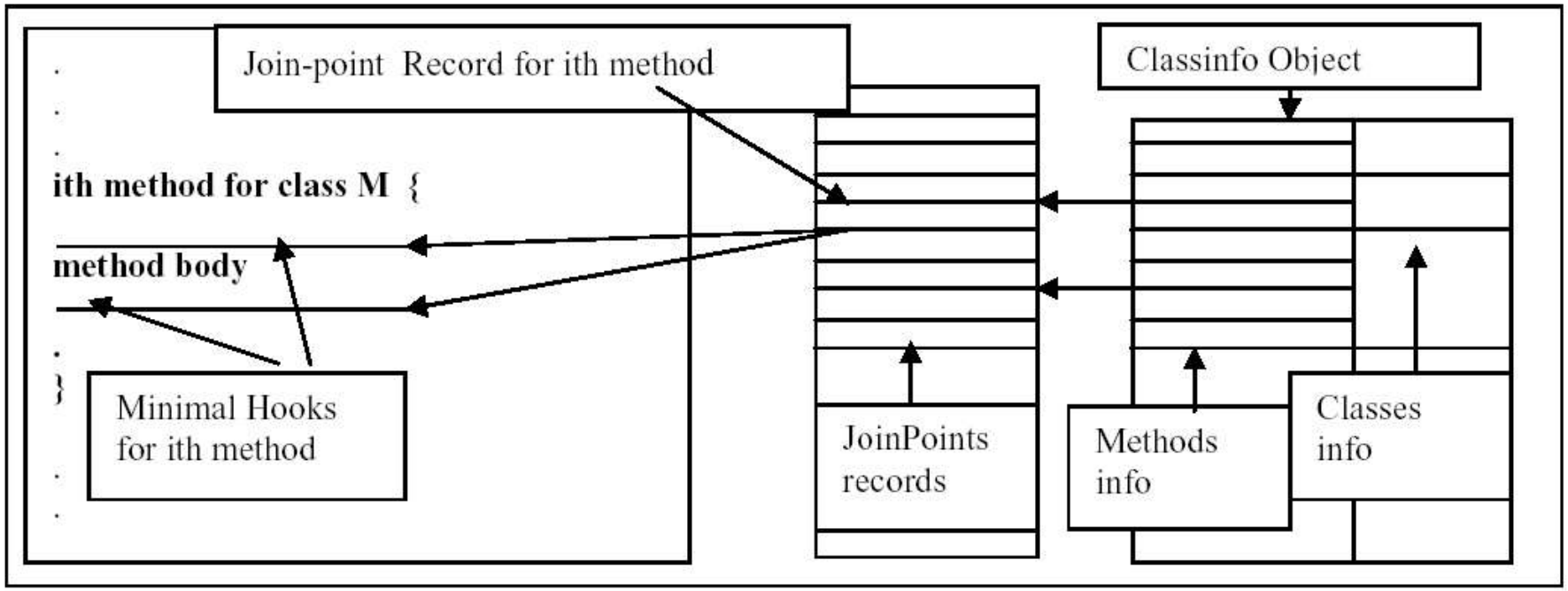
- fügt „*minimal hooks*“ in den Programmcode ein

```
int anymethod (.............) {
   if (JoinPoints [i].BeforeIsActivated)
      RunAdviceBefore(i)
   method body code
   if (JoinPoints [i].AfterIsActivated)
      RunAdviceAfter(i)
}
```

- die „*minimal hooks*“ der After-Advices müssen auch vor jeder Return-Anweisung stehen

Der DAO C++ Präprozessor

- erzeugt Metaobjekt-Daten



DAO C++: Aspektweben

1. Einlesen eines Aspekts
2. Finden der „*pointcuts*“ in den Metadaten
3. alle passenden „*minimal hooks*“ aktivieren

➔ damit ist der Aspekt aktiviert

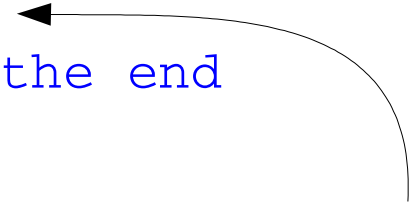
DAO C++ Synchonisation beim Weben

- während des Webens kann es zu Inkonsistenzen kommen
  - ➔ Aspekt an manchen Stellen noch „aktiv“
  - ➔ Weben muss atomar ausgeführt werden
- ➔ jeder Aspekt wird als kritischer Abschnitt betrachtet
  - geschützt durch Semaphore
  - während des Webens ist der Abschnitt gesperrt
  - beim Auflösen eines Aspekts muss zuvor gewartet werden bis der letzte Prozess den Aspekt verlassen hat

DAO C++ Synchonisation beim Weben

- es kann trotzdem zu Inkonsistenzen kommen:

```
int anymethod (.............) {
   minimal hook at beginning
   method body code
   minimal hook at the end
}
```



- Angenommen: Aspektweben beginnt, wenn Programm sich aktuell in diesem Programmteil befindet
  - es befindet sich nicht im Aspektcode
- z.B.: `minimal hook at beginning` belegt eine Ressource, die `minimal hook at the end` wieder freigeben soll

DAO C++ Performance

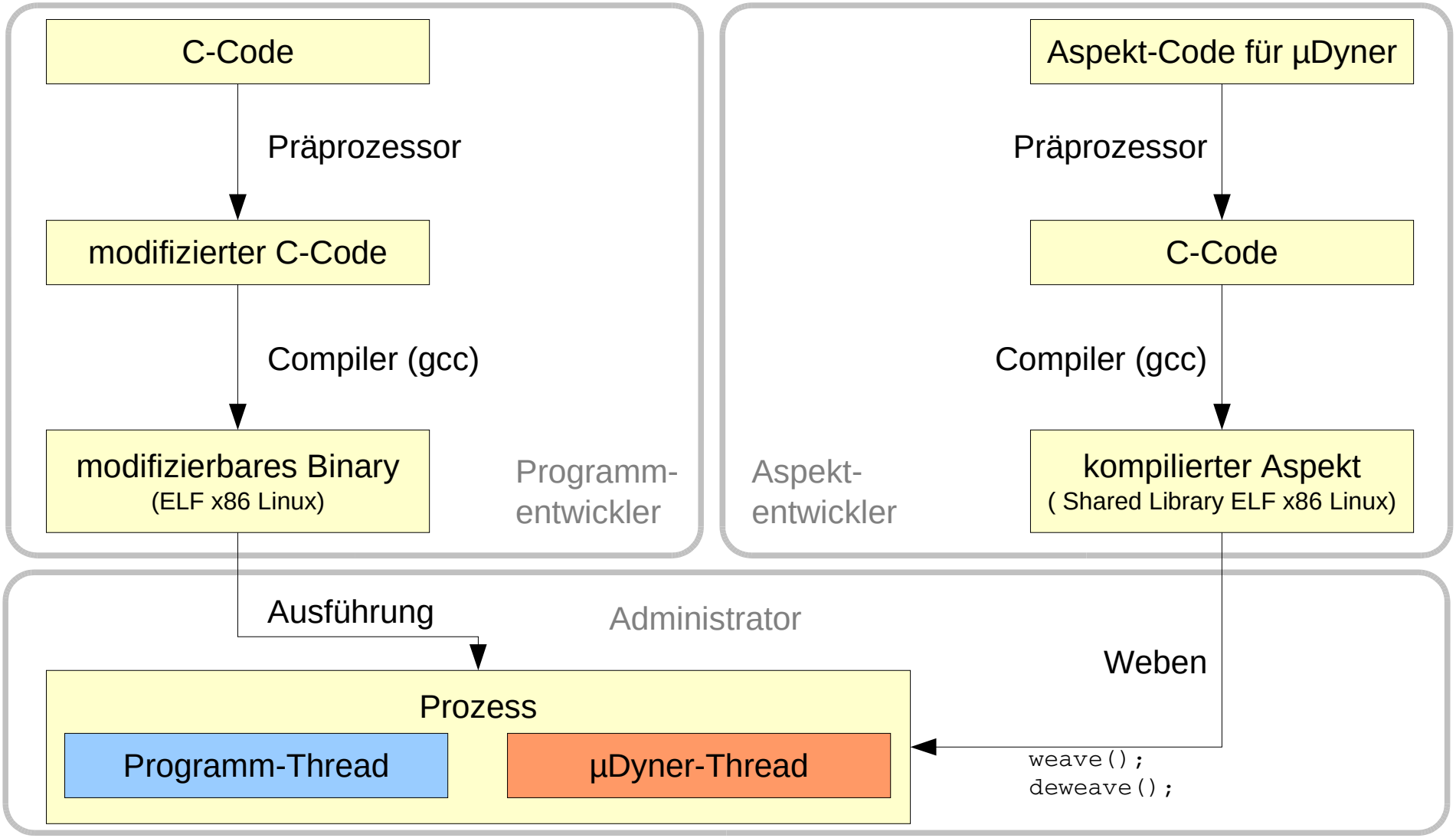
- wenn kein Aspekt eingewoben ist, spürbarer Overhead gegenüber C++ Programm
  - auf *„minimal hooks“* zurückzuführen
  - hängt natürlich auch von Programmstruktur ab: höherer Overhead bei vielen kleinen Funktionen
- mit gewobenem Aspekt gegenüber Programm mit statischem Aspekt großer Overhead
  - wegen Aufruf der AOP-Engine

➔ Ergebnisse von einem Prototyp, durch Optimierung lässt sich die Performance sicherlich steigern

µDyner

(steht für Micro Dynamic Weaver)

Architektur des µDyner



„minimal hook“ im Programm

- *„minimal hook“* im Programm
  - vor dem Weben

```
int anymethod (............) {
    JMP 3; NOP; NOP; NOP;
    method body code
}
```

  - am Anfang jeder Funktion werden 5 Byte reserviert
    - 5 Byte für einen Sprungbefehl beim Pentium

      (1 Byte Befehl „JMP“, 4 Byte Sprungadresse)
  - *„inlinen“* von Funktionen muss verhindert werden
    - damit die Funktionen beim Weben gefunden werden können

„minimal hook“ im Programm

- *„minimal hook“* im Programm
  - nach dem Weben

```
int anymethod (............) {
    JMP Hook;
    method body code
}
```

  - Programm springt zu einem *„minimal hook“*

- die Symbole können mit dem Posix-Tool **nm** gefunden werden
  - ➔ liefert Funktionsnamen und -anfang

„minimal hook“

```
save registers
if (aspect_loaded && pointcut()) {
   call aspect
   restore registers
   return aspect result
} else {
   restore registers
   perform original effect of join point
   return result
}
```

- Code wird erst zur Laufzeit eingebunden
  - ➔ Compiler kann sich nicht um Sichern des Kontext kümmern
  - ➔ er „weiß“ zur Übersetzungszeit nicht dass noch Code eingefügt werden soll
- ➔ explizites Sichern und Wiederherstellen der Register notwendig

„minimal hook“

```
save registers
if (aspect_loaded && pointcut()) {
   call aspect
   restore registers
   return aspect result
} else {
   restore registers
   perform original effect of join point
return result
}
```

- globale Variable
- gibt an ob das Aspektweben fertig ist

„minimal hook“

```
save registers
if (aspect_loaded && pointcut()) {
   call aspect
   restore registers
   return aspect result
} else {
   restore registers
   perform original effect of join point
return result
}
```

- prüft den Stack
  - dort muss der Kontext des Aspekts und der durch den Aspekt erweiterten Funktion liegen

„*minimal hook*“ bei Funktionsaufruf

```
save registers used by the hook
if ( aspect_loaded && pointcut() ) {
   restore registers used by the hook
   JMP advice_fn
} else {
   restore registers used by the hook
   JMP fn + 5
}
```

- Sprung zum Aspekt lässt Stack unangetastet
- bei Verlassen des Aspekts Rückkehr zum Anfang der vom Aspekt beeinflussten Funktion

„*minimal hook*“ bei Funktionsaufruf

```
save registers used by the hook
if ( aspect_loaded && pointcut() ) {
   restore registers used by the hook
   JMP advice_fn
} else {
   restore registers used by the hook
   JMP fn + 5
}
```

- Rückkehr zum ersten Befehl nach der Sprunganweisung

„minimal hook“ bei Lesen einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut() ) {
   CALL advice_fn
   MOVE eax r_tgt
   restore registers except r_tgt
} else {
   restore registers
   MOVE vaddr, r_tgt
}
JMP addr + move_instruction_size
```

- Zugriff auf eine globale Variable wird in eine Funktion umgeformt

„minimal hook“ bei Lesen einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut() ) {
   CALL advice_fn
   MOVE eax, r_tgt
   restore registers except r_tgt
} else {
   restore registers
   MOVE vaddr, r_tgt
}
JMP addr + move_instruction_size
```

- Rückgabewert der Advice-Funktion befindet sich in `eax`
- Wertzuweisung: Programm erwartet Zielvariable in `r_tgt`

„minimal hook“ bei Lesen einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut() ) {
   CALL advice_fn
   MOVE eax, r_tgt
   restore registers except r_tgt
} else {
   restore registers
   MOVE vaddr, r_tgt
}
JMP addr + move_instruction_size
```

- Wertzuweisung: Wert wird aus dem Speicher in das Register `r_tgt` geladen

„minimal hook“ bei Lesen einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut() ) {
   CALL advice_fn
   MOVE eax, r_tgt
   restore registers except r_tgt
} else {
   restore registers
   MOVE vaddr, r_tgt
}
JMP addr + move_instruction_size
```

- Rücksprung zur Programmstelle
- globaler Variablenzugriff an beliebiger Stelle im Code möglich

„minimal hook“ beim Schreiben einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut()) {
   PUSH r_src
   CALL advice_fn
   restore registers
   MOVE vaddr, r_src
} else {
   restore registers
   MOVE r_src , vaddr
}
JMP addr + move_instruction_size
```

- Wert wird auf Stack abgelegt
  - ➔ „Advice-Funktion“ erhält Wert als Parameter übergeben

„minimal hook“ beim Schreiben einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut()) {
   PUSH r_src
   CALL advice_fn
   restore registers
   MOVE vaddr, r_src
} else {
   restore registers
   MOVE r_src , vaddr
}
JMP addr + move_instruction_size
```

- Wert wird zurückgespeichert
- damit kann das Programm mit dem Wert weiterarbeiten

„minimal hook“ beim Schreiben einer globalen Variablen

```
save registers
if (aspect_loaded && pointcut()) {
   PUSH r_src
   CALL advice_fn
   restore registers
   MOVE vaddr, r_src
} else {
   restore registers
   MOVE r_src , vaddr
}
JMP addr + move_instruction_size
```

- Wert wird aus dem Register in den Speicher kopiert

µDyner: Aspektweben

- soll zur Laufzeit erfolgen
  - der Code muss konsistent bleiben
- der Pentium kann nur 2 oder 4 Byte Schreiboperationen durchführen
  - es müssen 5 Byte geschrieben werden
- zuerst wird die 2. Hälfte der 5 Bytes an der Sprungstelle überschrieben
- bei globalem Vareiablenzugriff: zuerst Überschreiben des 1. Byte an der Sprungstelle mit INT3
  - ➔ das Programm wird an dieser Stelle unterbrochen

µDyner: Performance

- Einbinden eines Aspekts: die meiste Zeit wird zum Laden der Bibliothek benötigt
- weniger als 1,5% Overhead gegenüber üblichem C-Programm
  - Vergleich durch Funktion get() zum Holen eine Webseite:
    1. als Aspekt
    2. als Funktion in einem C-Programm

C++-Erweiterung für µ Dyner

µDyner: C++-Erweiterung

- Unterschiede im Binär-Format zwischen C und C++
  - *„name mangling“* (Symbol aus Funktionsname und Klassenname)

```
class Bar {
   public:
       int ival();
 }
```

  - ➔ das Symbol sieht dann möglicherweise so aus:

```
ival_3Bar
```

  - verwendet wird Posix-Tool **nm** zum Auslesen der Symbole in C++
- ➔ damit µDyner auch für C++ verwendbar

Zusammenfassung µDyner

- 3 Typen von „*minimal hooks*“
  - Aufruf einer Funktion
  - Lesen einer globalen Variablen
  - Schreiben einer globalen Variablen

Zusammenfassung und kritische Betrachtung

Zusammenfassung

- DAO C++ und µDyner ermöglichen Erzeugung von Aspekten zur Laufzeit
- vor dem Kompilieren muss dazu Code eingefügt werden
- µDyner mit wesentlich ausgefeilteren Methoden
- die Features sind recht eingeschränkt
  - nur die grundlegenden Mechanismen vorhanden
  - bei DAO C++ *before*- und *after*-Advice
  - bei µDyner nur *before*-Advice
  - keine Introduction, Aspektvererbung...

Kritische Betrachtung

- Ist es wirklich sinnvoll dynamische Aspekte zu verwenden?
- noch einmal zurück zum Beispiel Web-Caches:
- ohne dynamische Aspekte:
  - Anhalten – Neustarten eines Systems dauert nur Sekunden
    - es wird einfach die Anwendung ausgetauscht
  - geringerer Overhead mit festen Aspekten
  - alle Features aspektorientierter Programmierung verfügbar
  - die Prefetching-Strategie ist wohl eher selten auszutauschen

Quellen

- DAO C++:

  http://aosd.net/2004/workshops/daw/Proc-2004-Dynamic-Aspects.pdf

  Seite 6-13

- µDyner:

  http://www.emn.fr/x-info/microdyner/uDyner__Papers.html

  – C++-Erweiterung:

  http://pcinfo16.info.emn.fr:8099/options/graduates/thesis/ychen.pdf