Eine komponentenorientierte Sprache für pointerfreie parallele Programmierung

Luc Bläser
Institut für Computersysteme
ETH Zürich
blaeser@inf.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

KPS'07, Timmendorfer Strand,
10. – 12. Oktober 2007

Stand der Programmiertechnik

Mängel in objektorientierten Sprachen

- Strukturierung
  - Referenzen
    - Beliebiges Verlinken von Objekten => Unstrukturierte Abhängigkeiten
    - Fehlende hierarchische Komposition => Objekt kann keine anderen (dynamisch erzeugte) Objekte kapseln
  - Vererbung
    - Unbegründeter Zwang zur Hierarchisierung auf Typebene
    - Unpassende Kombination von Polymorphismus und Code-Wiederverwendung
- Nebenläufigkeit
  - Methoden
    - Blockierende Prozeduraufrufe statt echtem Nachrichtenaustausch
    - Nur einfache Input-Output Interaktion => komplexere zustands-behaftete Interaktion nicht unterstützt
  - Threads
    - Im Nachhinein auf prozedurales Model aufgesetzt
    - Threads operieren auf beliebigen Objekten ohne klare Spezifikation der potentiellen Abhängigkeiten => fehleranfällig (z.B. *races*)

Die Komponentensprache

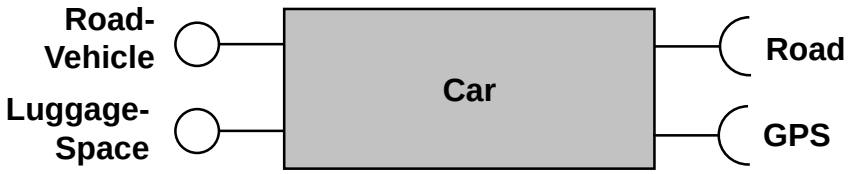
Komponentenbegriff

- Abstraktionseinheit zur Laufzeit
- Strikte Kapselung
  - Äußere Abhängigkeiten nur via expliziten Schnittstellen zugelassen
- Komponente kann Schnittstellen anbieten und erfordern
  - *angebotene* Schnittstellen repräsentieren eigene Facetten der Komponente
  - *erforderte* Schnittstellen sind von anderen Komponenten anzubieten
- Mehrfacherzeugung von Schablone

Statische Schablone

```
COMPONENT Car
  OFFERS RoadVehicle, LuggageSpace
  REQUIRES Road, GPS;
  (* implementation *)
END Car;
```

Laufzeitexemplar



Komponentenexemplare

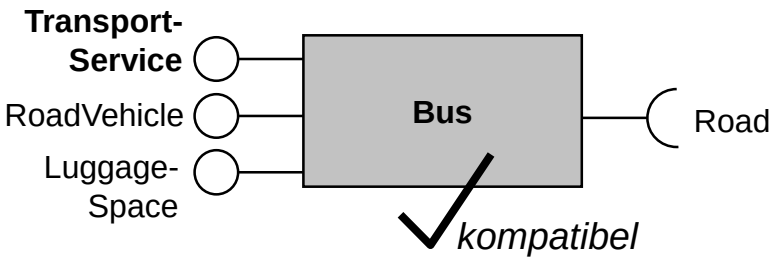
Deklaration:

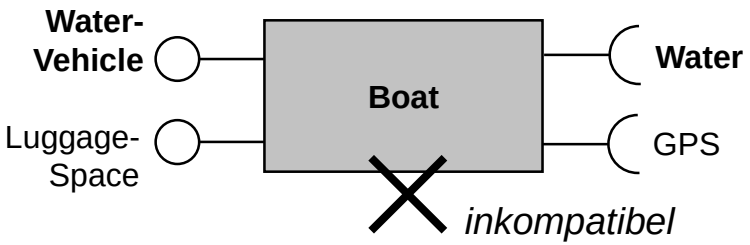
car1, car2: Car;

roadVehicle: **ANY(**RoadVehicle, LuggageSpace **|** Road, GPS**)**

roadVehicle ist eine Komponente einer beliebigen Schablone, welche

- mindestens RoadVehicle und LuggageSpace anbietet
- höchstens Road und GPS erfordert





Dynamische Kollektion von Komponenten

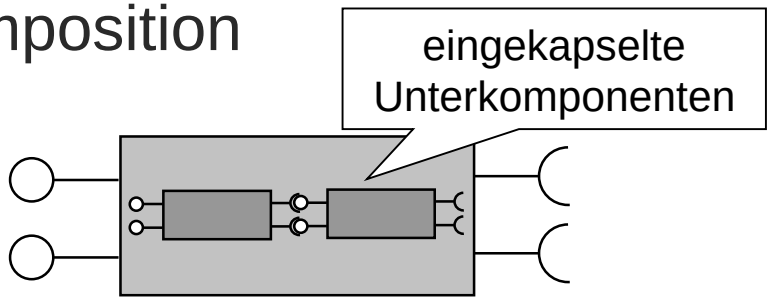
- Index zur Identifizierung

  car[state, code: TEXT]: Car

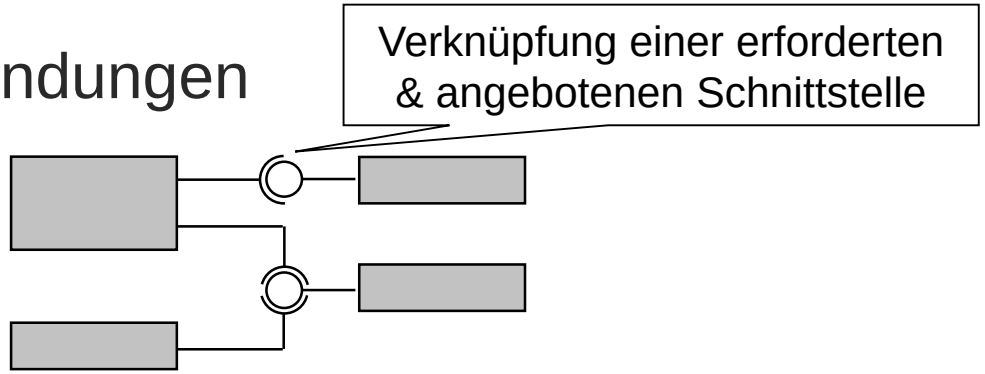
  Mögliche Komponenten: car[“M”, “FX-1211”] car[“HL”, “AA-111”] …

Komponentenbeziehungen

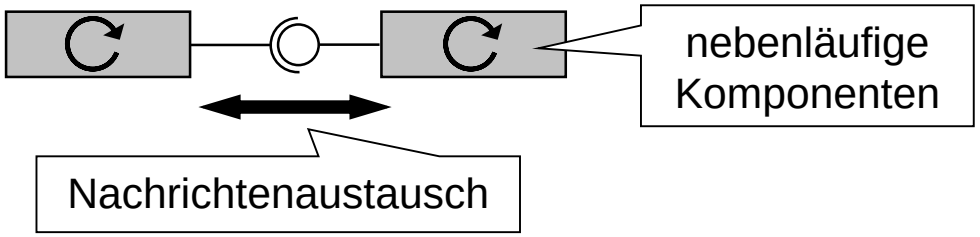
- Hierarchische Komposition



- Schnittstellenverbindungen



- Kommunikationsbasierte Interaktionen

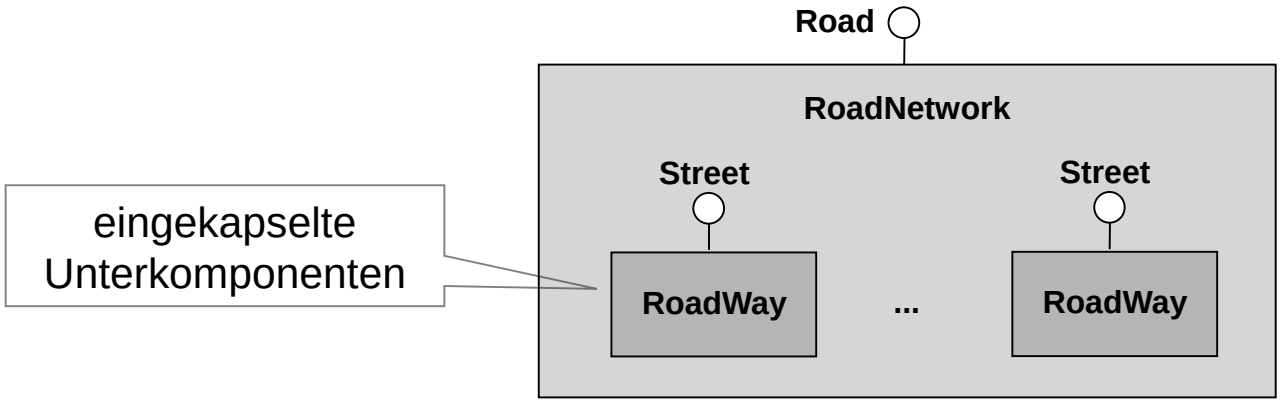


Hierarchische Komposition

- Jede Komponente kann beliebig viele Unterkomponenten beinhalten

```
COMPONENT RoadNetwork OFFERS Road;
VARIABLE
  street[no: INTEGER]: RoadWay;
BEGIN
  FOREACH street n in input file DO
    NEW(street[n])
  END
END RoadNetwork;
```

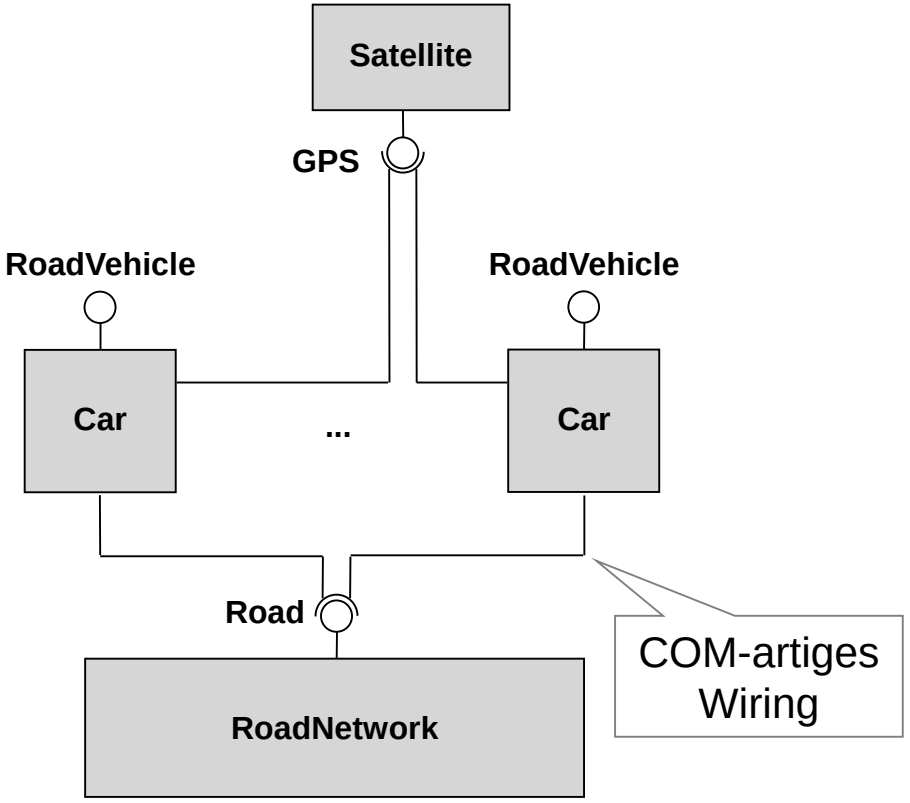
Variablen als Behälter für Komponenten



Schnittstellenverbindungen

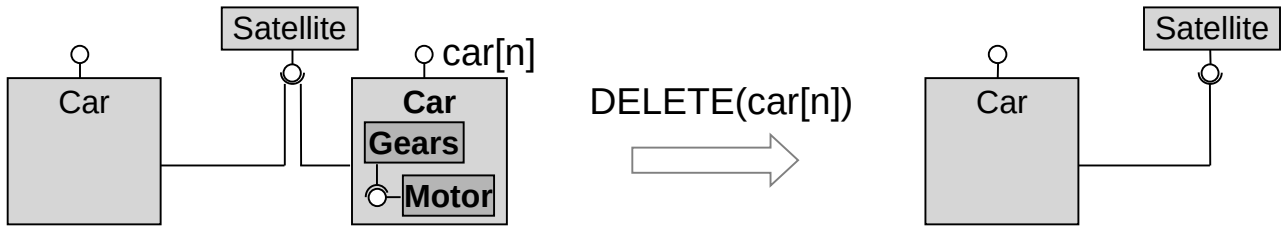
- Jede erforderte Schnittstelle kann mit einer angebotenen Schnittstelle des gleichen Namens verbunden werden

```
COMPONENT TrafficSimulation;
VARIABLE
  car[id: INTEGER]: Car;
  road: RoadNetwork;
  satellite: Satellite;
BEGIN
  NEW(road); NEW(satellite);
  FOREACH car id in simulation file DO
    NEW(car[id]);
    CONNECT(Road(car[id]), road);
    CONNECT(GPS(car[id]), satellite)
  END
END TrafficSimulation;
```



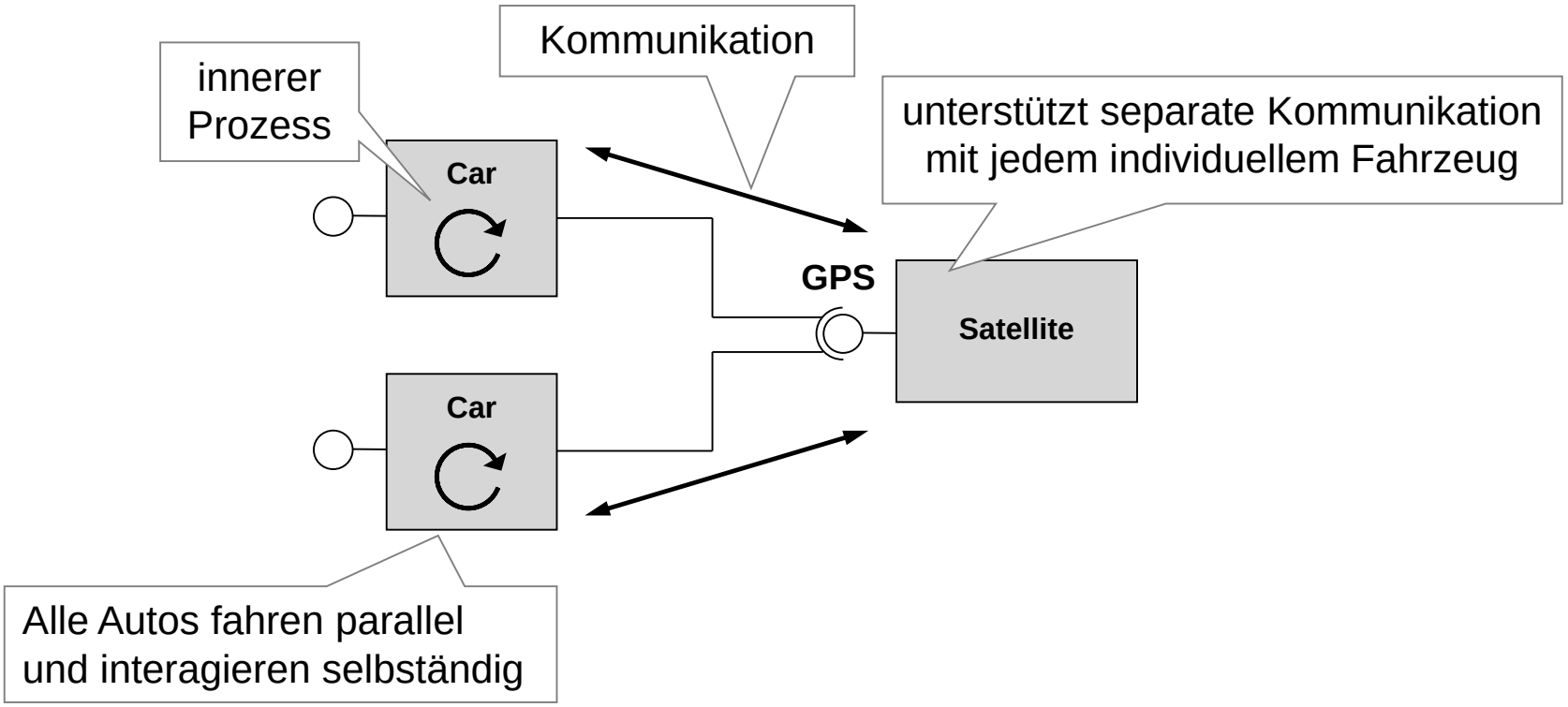
Pointerfreie Strukturen

- Unterschied Schnittstellenverbindungen zu Referenzen
  - Verbindungen nur von der umgebenden Komponente gesetzt
  - Ausgehende und eingehende Verbindungspunkte für jede Komponente klar deklariert
- Hierarchie von Komponentennetzwerken
  - Netzwerkstruktur ausschließlich von der jeweils umgebenden Komponente kontrolliert
- Hierarchische Existenzabhängigkeit
  - Löschen einer Komponente => Automatisches Löschen der Unterkomponenten
  - Explizites Löschen einer Komponente => Verbindungen mit deren Schnittstellen werden getrennt
  - Sichere Speicherverwaltung ohne Garbage Collector



Nebenläufigkeit und Interaktionen

- Jede Komponente führt ihre eigenen inneren Prozesse aus
- Komponenten interagieren mittels Kommunikation über Schnittsstellen



Kommunikation

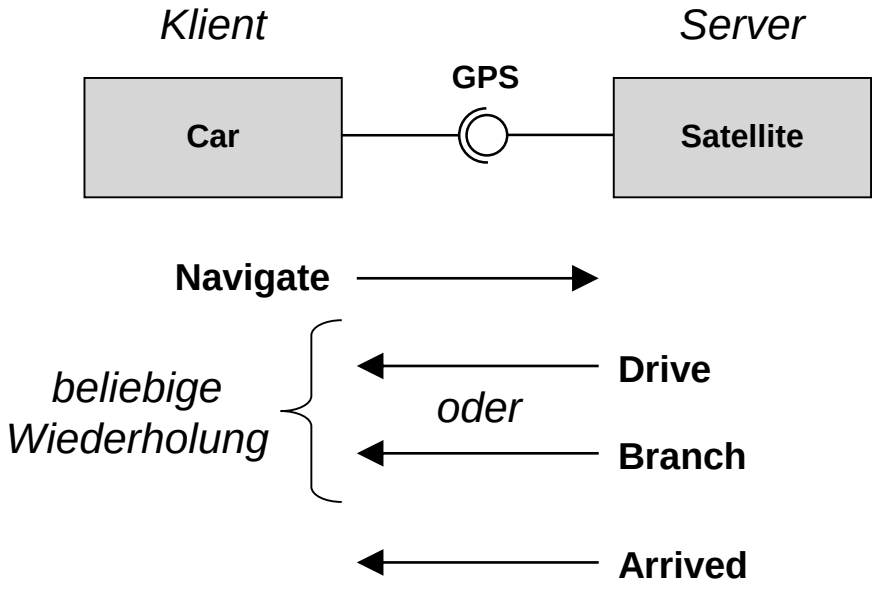
- Separate Kommunikation zwischen jedem Klienten und Server
- Senden und Empfangen von Nachrichten gemäß eines Protokolls

Kommunikations-protokoll in EBNF

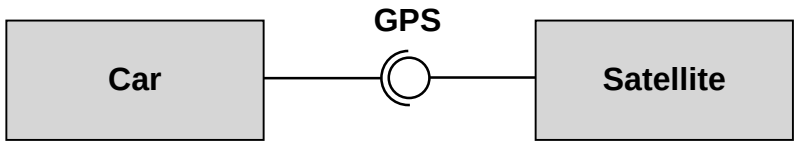
Repetition

Alternative

```
INTERFACE GPS;
  IN Navigate(target: TEXT)
  {
    OUT Drive(meters: INTEGER)
  |
    OUT Branch(junction: TEXT)
  }
  OUT Arrived
END GPS;
```



Implementierung der Komponenten




```
COMPONENT Car REQUIRES GPS;
BEGIN
  GPS!Navigate(target);
  REPEAT
    IF GPS?Drive THEN
      GPS?Drive(m) (* drive *)
    ELSIF GPS?Branch THEN
      GPS?Branch(j) (* take junction *)
    END
  UNTIL GPS?Arrived;
  GPS?Arrived
END Car;
```

sende Nachricht

Empfangstest

empfange Nachricht

separater Service-prozess pro Klient

```
COMPONENT Satellite OFFERS GPS;
  IMPLEMENTATION GPS;
  BEGIN {SHARED}
    ?Navigate(target);
    WHILE not arrived DO
      IF straight route THEN !Drive
      ELSE !Branch(junction)
      END
    END;
    !Arrived
  END GPS;
END Satellite;
```

Montior-Synchronisation *innerhalb* der Komponente

Von Compiler überprüft (keine Races möglich)

Laufzeitsystem

Kleines Betriebssystem

- Leichtgewichtige Prozesse
  - Mikro-Stacks: Größe kann dynamisch wachsen & schrumpfen
  - Ermöglicht sehr hohe Anzahl an Prozessen
- Schnelle Kontext-Wechsel
  - Direkte synchrone Kontext-Wechsel
  - Kosteneffiziente Preemption mittels Code-Instrumentierung
    (Nur benötige Register müssen bei Preemption gesichert werden)
- Sichere und effiziente Speicherverwaltung
  - Garbage Collector nicht mehr benötigt
  - Virtuelle Speicherverwaltung nicht mehr benötigt

Skalierung und Performanz

- Maximale Anzahl Prozesse

Component OS	C#	Java	Oberon AOS
5'010'000	1'890	10'000	15'700

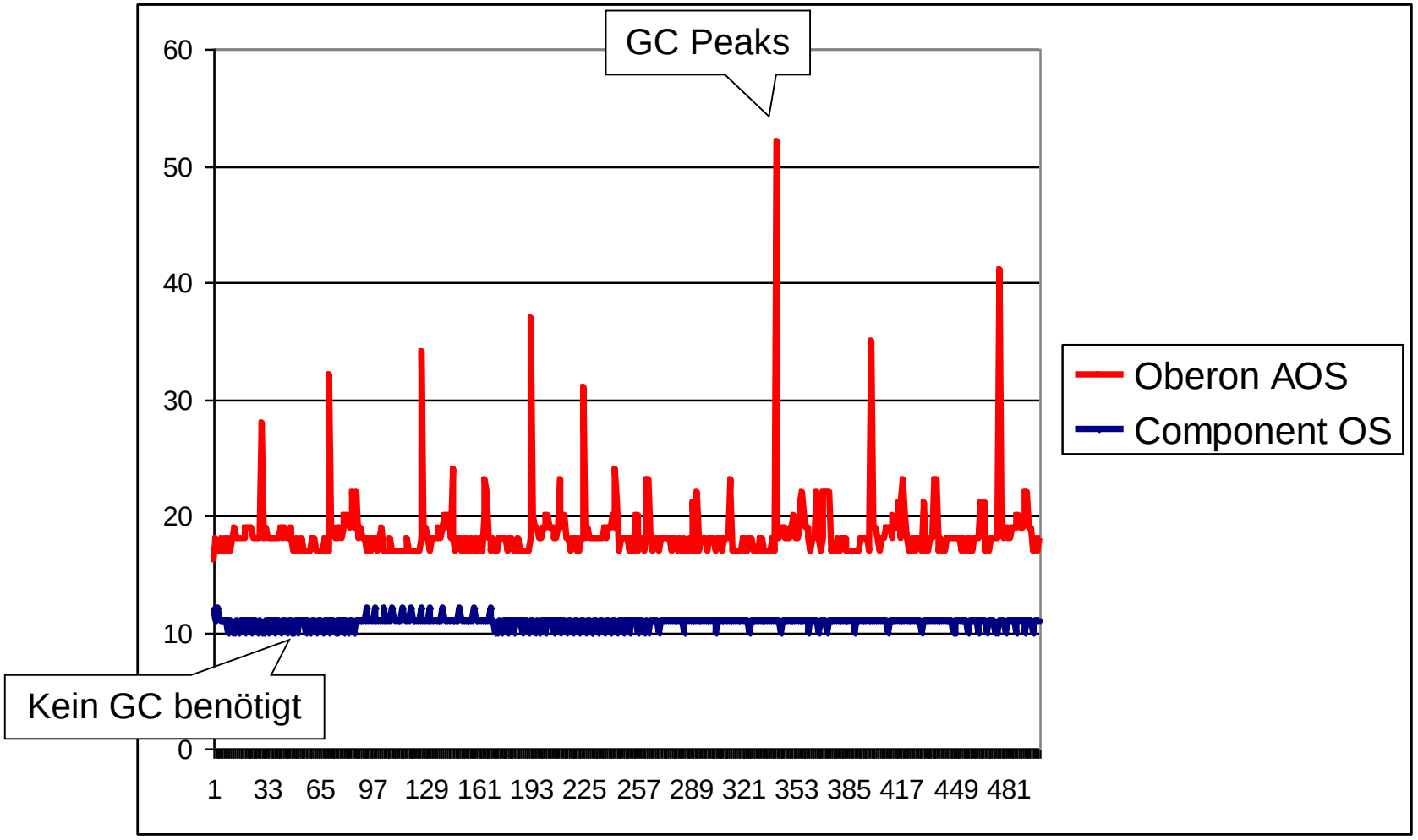
4GB Hauptspeicher, City Benchmark

- Ausführungsgeschwindigkeit

Programm (in s)	Component OS	C#	Java	Oberon AOS
City	**0.24**	0.66	440	4.1
ProducerCons.	**16**	19	130	60
Eratosthenes	**1.8**	6.8	4.6	5.8
News	**2.0**	3.5	3.9	4.6
TokenRing	**2.1**	22	22	18
TrafficSimulation	**0.05**	33	-	-

In Sekunden, 6 CPUs Intel Xeon 700MHz, C# & Java Windows Server Enterprise Edition
C#, Java, AOS: Analoge nebenläufige Programme mit Methoden statt Kommunikation

Vorhersagbarkeit ohne Garbage Collection



500 aufeinander folgende Ausführungen des Small City Programms (in ms)

Schlussfolgerungen

Eine neue Sprache zur Erstellung besser strukturierter nebenläufiger Programme

- Konzeptionelle Vorteile
  - Hierarchische kontrollierte Strukturen statt Pointer
  - Garantierte hierarchische Einkapselung
  - Durchgängige strukturierte Nebenläufigkeit
  - Zustandsbehaftete Kommunikation statt Methodenaufrufe
  - Flexibler Polymorphismus getrennt von Wiederverwendung
  - Wiederverwendung inhärent durch Komposition ermöglicht
- Technische Vorteile
  - Hohe Skalierung in der Anzahl der Prozesse
  - Schnelle Ausführung von nebenläufigen Programmen
  - Kein Garbage Collection mehr nötig
- Praktische Anwendung in Verkehrsimulation
- Projekt-Webseite: http://www.jg.inf.ethz.ch/components