A Component Language for Structured Concurrent Programming

Luc Bläser

ETH Zürich / LBC Informatik

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LBC Informatik
Software-Ingenieure ETH

Talk at Microsoft, Redmond
29 September 2008

Motivation

Problems of object-orientation

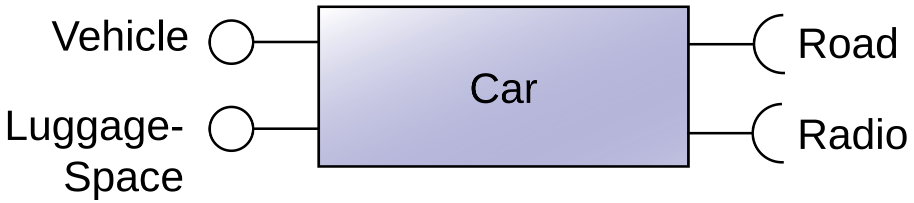
- References
  - Flat object structures without explicit hierarchies
  - Intended encapsulation is not guaranteed
- Inheritance
  - Forced combination of polymorphism and reuse
  - Limited single inheritance or multi-inheritance conflicts
- Concurrency
  - Unnecessarily blocking interactions via method calls
  - Threads operating on passive objects without control

A New Programming Model

Component concept

- General abstraction unit at runtime
- Strict encapsulation
  - External dependencies only allowed via explicit interfaces
- Component can offer and require interfaces
  - Offered interfaces represent own external facets of a component
  - Required interfaces are to be provided by other components
- Multi-instantiation from a component template

COMPONENT **Car**
  OFFERS **Vehicle**, **LuggageSpace**
  REQUIRES **Road**, **Radio**
  (* implementation *)
END Car



Component Instances

Declarations:

car1, car2: Car;

vehicle: ANY(Vehicle, LuggageSpace | Road, Radio)

any component template which

- offers **at least** Vehicle and LuggageSpace
- requires **at most** Road and Radio

Dynamic collection of component instances

- Index identifies an instance within the collection:

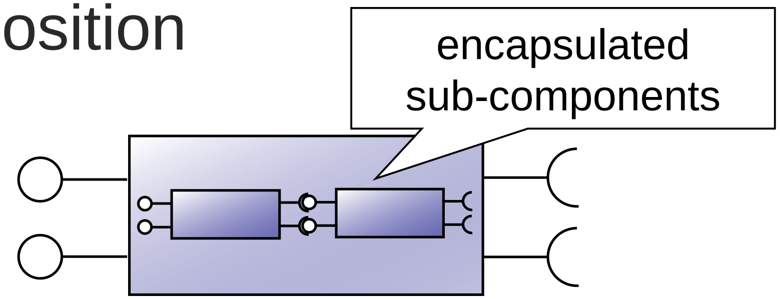
  car[state: TEXT; number: INTEGER]: Car

- Possible instances:

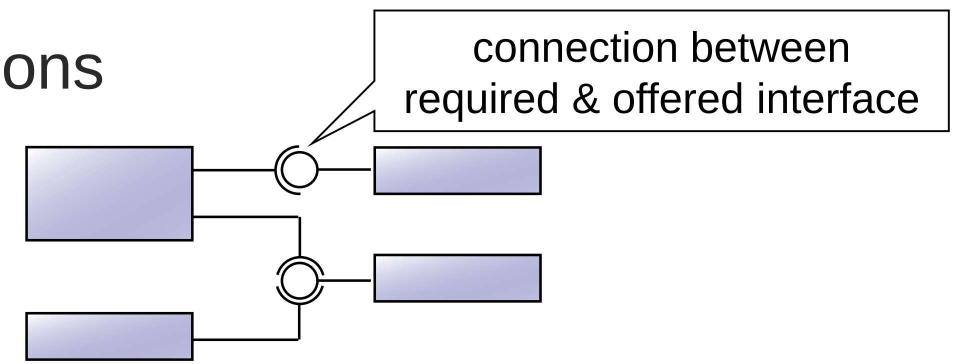
  car[“ZH”, 965231] car[“SO”, 11] …

Component Relations

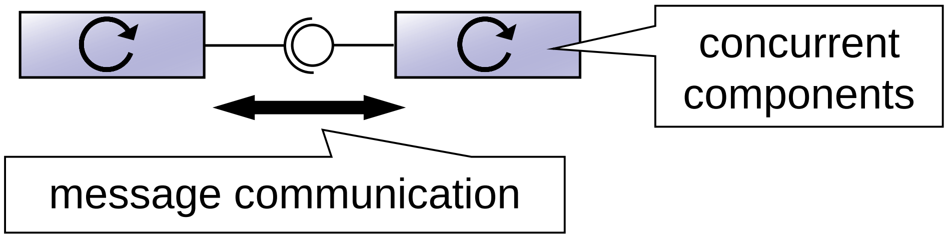
- Hierarchical composition



- Interface connections



- Communication-based interactions

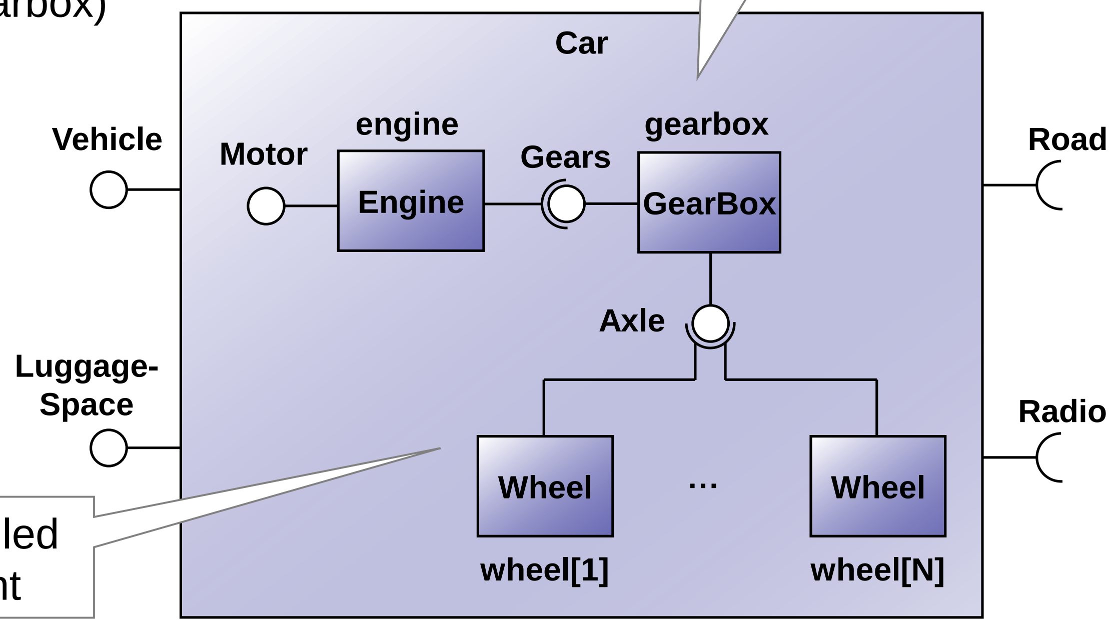


Hierarchical Composition

COMPONENT Car …
VARIABLE
  engine: Engine;
  gearbox: GearBox;
  wheels[n: INTEGER]: Wheel
BEGIN
  NEW(engine); NEW(gearbox);
  CONNECT(Gears(engine), gearbox);
  FOR i := 1 TO N DO
    NEW(wheel[i]);
    CONNECT(Axle(wheel[i]), gearbox)
  END
END Car

variables as containers for components

encapsulated sub-components

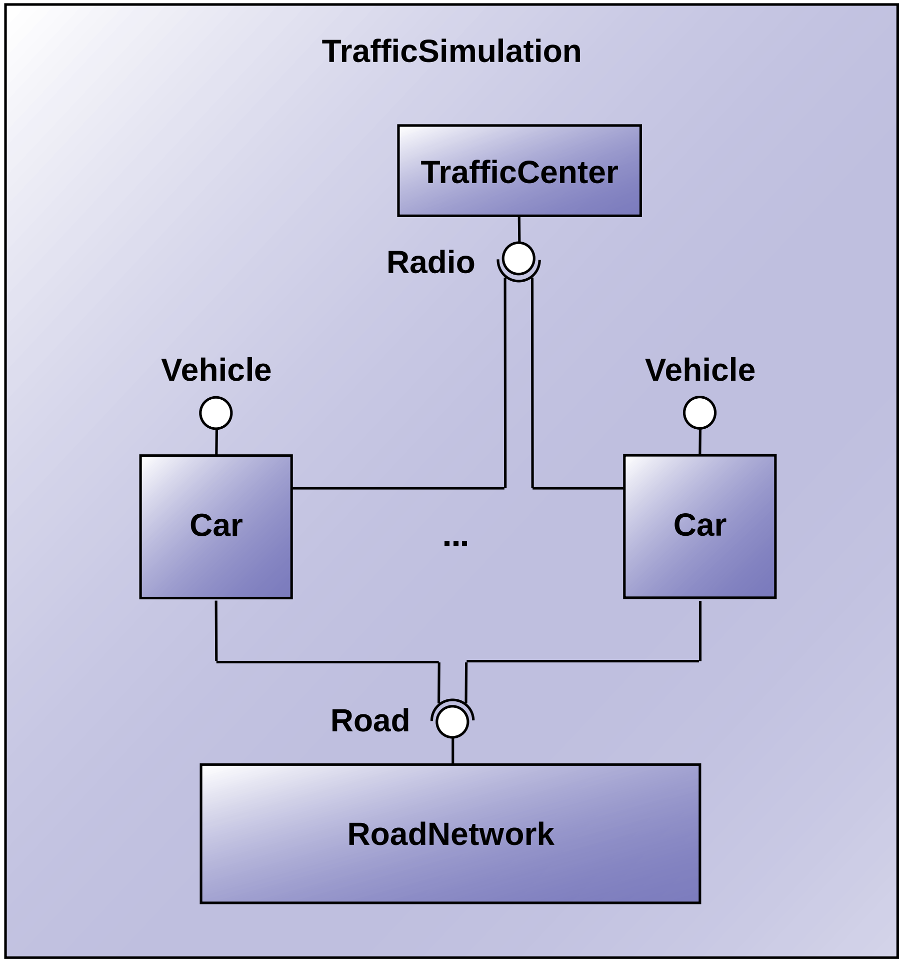


structure exclusively controlled by surrounding component

Dynamic Composition

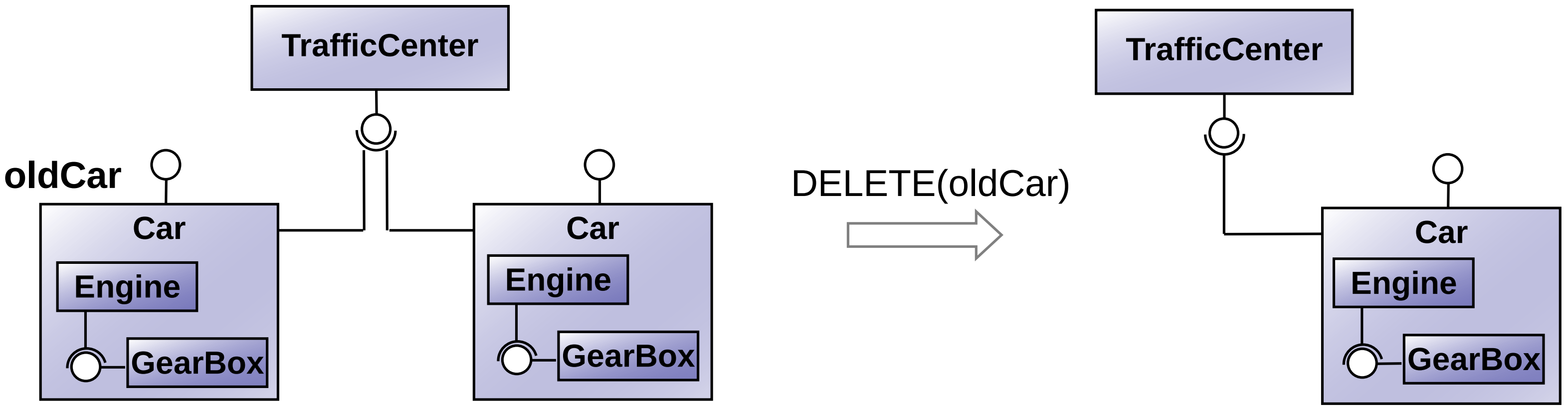
```
COMPONENT TrafficSimulation
VARIABLE
  car[licenseNo: INTEGER]: Car;
  road: RoadNetwork;
  news: TrafficCenter
BEGIN
  NEW(road); NEW(news);
  REPEAT
    id := GetNewLicenseNo();
    NEW(car[id]);
    CONNECT(Road(car[id]), road);
    CONNECT(Radio(car[id], news)
  UNTIL EnoughCars()
END TrafficSimulation
```

number of cars only known at runtime



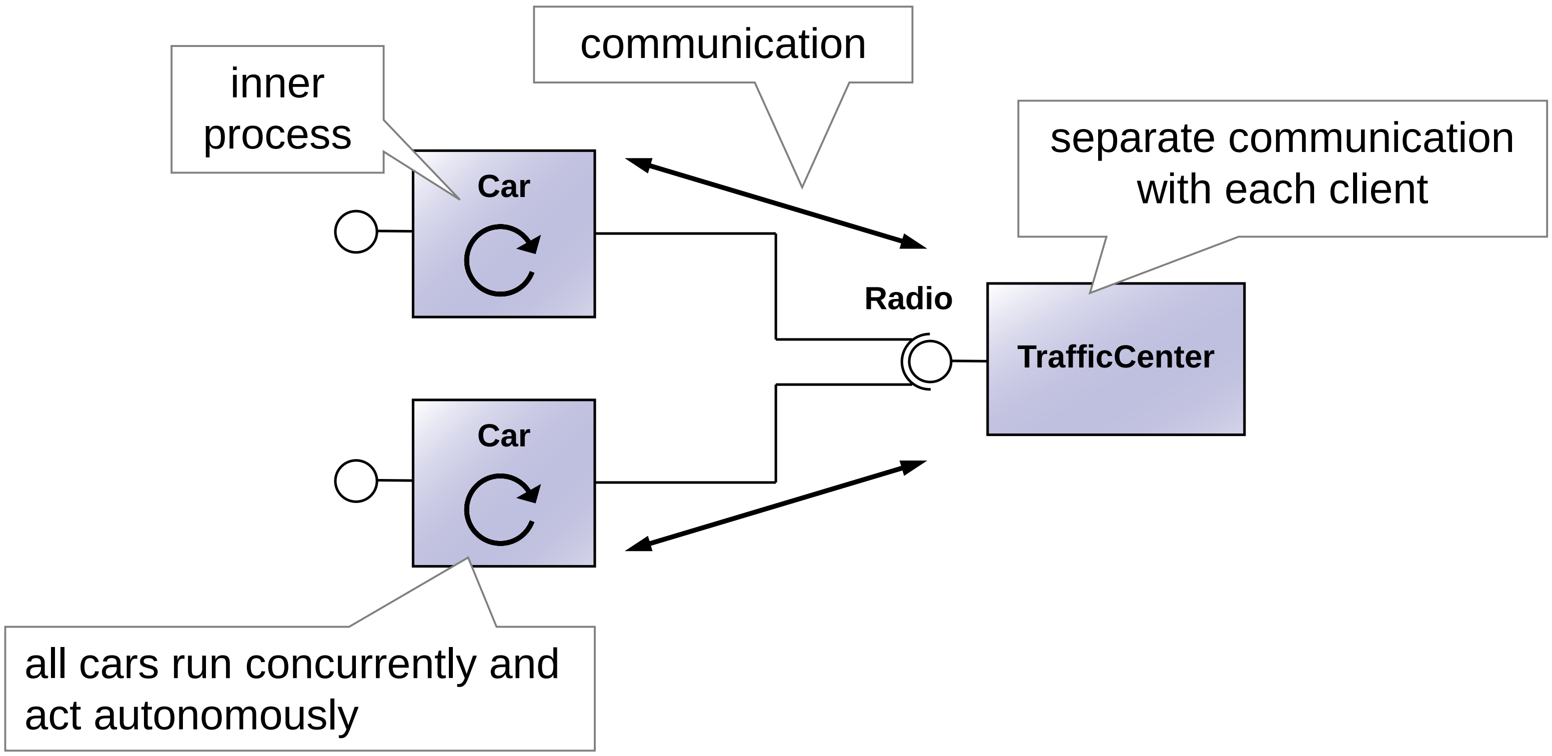
Pointer-Free Structuring

- Interface connections versus references
  - Interface connections only set by the surrounding component
  - Explicitly declared incoming and outgoing connection points
- Hierarchy of component networks
- Hierarchical lifetimes
  - Deletion of a component => automatic deletion of sub-components
  - Explicit deletion of a single component => interface disconnection
- Safe memory management without garbage collector



Concurrency und Interactions

- Each component runs its own inner processes
- Components interact by message communication via interfaces



Communication

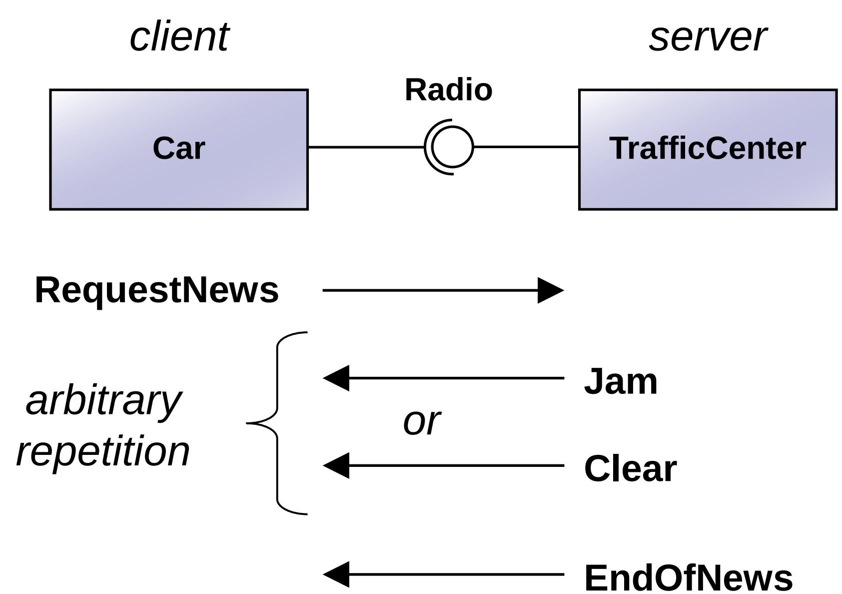
- Server maintains a statefull communication with each client individually
- Sending and receiving messages according to a protocol

communication protocol in EBNF

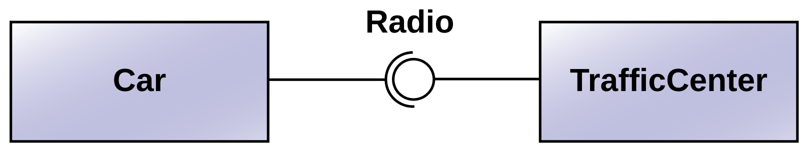
repetition

alternative

```
INTERFACE Radio
  IN RequestNews(location: TEXT)
  {
    OUT Jam(roadNo: INTEGER)
  |
    OUT Clear(roadNo: TEXT)
  }
  OUT EndOfNews
END Radio
```



Component Implementation



send message

receive test

receive message

separate service process per client

```
COMPONENT Car REQUIRES Radio
BEGIN
  Radio!RequestNews(here);
  REPEAT
    IF Radio?Jam THEN
      Radio?Jam(x) (* bypass x *)
    ELSIF Radio?Clear THEN
      Radio?Clear(x) (* can take x *)
    END
  UNTIL Radio?EndOfNews;
  Radio?EndOfNews
END Car
```

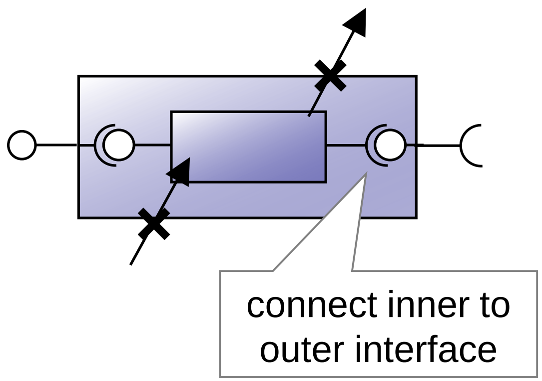
```
COMPONENT TrafficCenter OFFERS Radio
  IMPLEMENTATION Radio
  BEGIN {SHARED}
    ?RequestNews(location);
    FOREACH road x at location DO
      IF x jammed THEN !Jam(x)
      ELSE !Clear(x)
      END
   END;
   !EndOfNews
  END Radio
END TrafficNews
```

compiler-checked exclusion of races

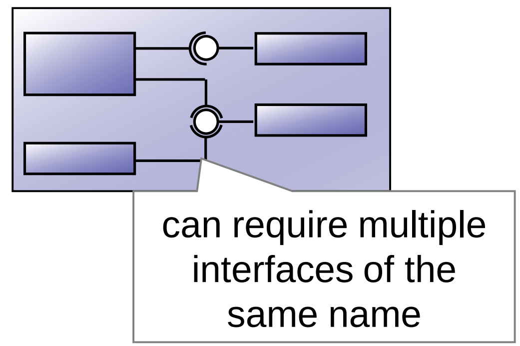
monitor synchronisation only *inside* a component

Language Features

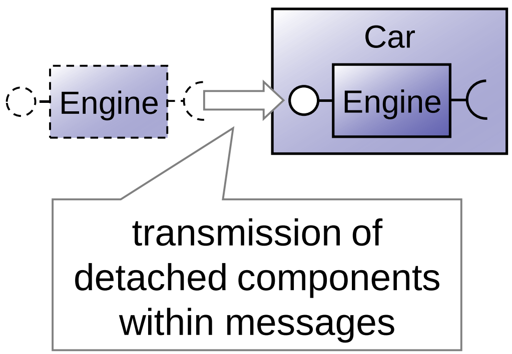
Guaranteed encapsulation



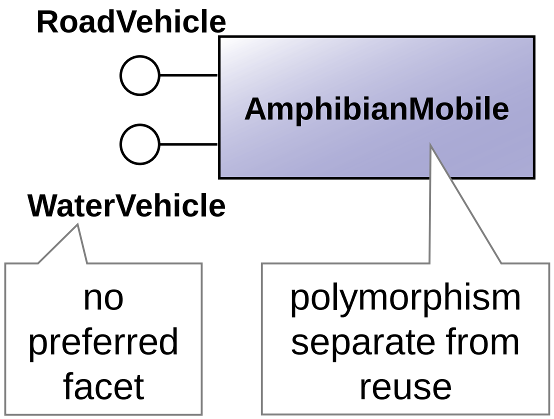
Hierarchical networks



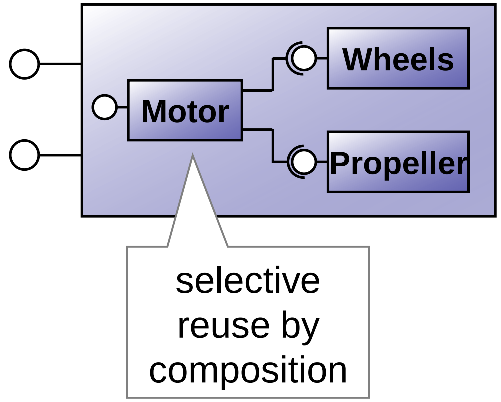
Plug-ins



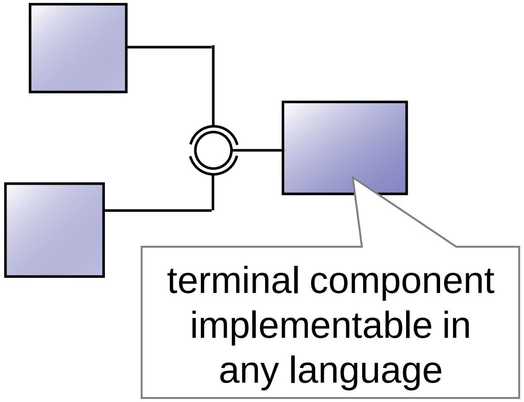
Symmetric polymorphism



Flexible reuse



Interoperability



Runtime System

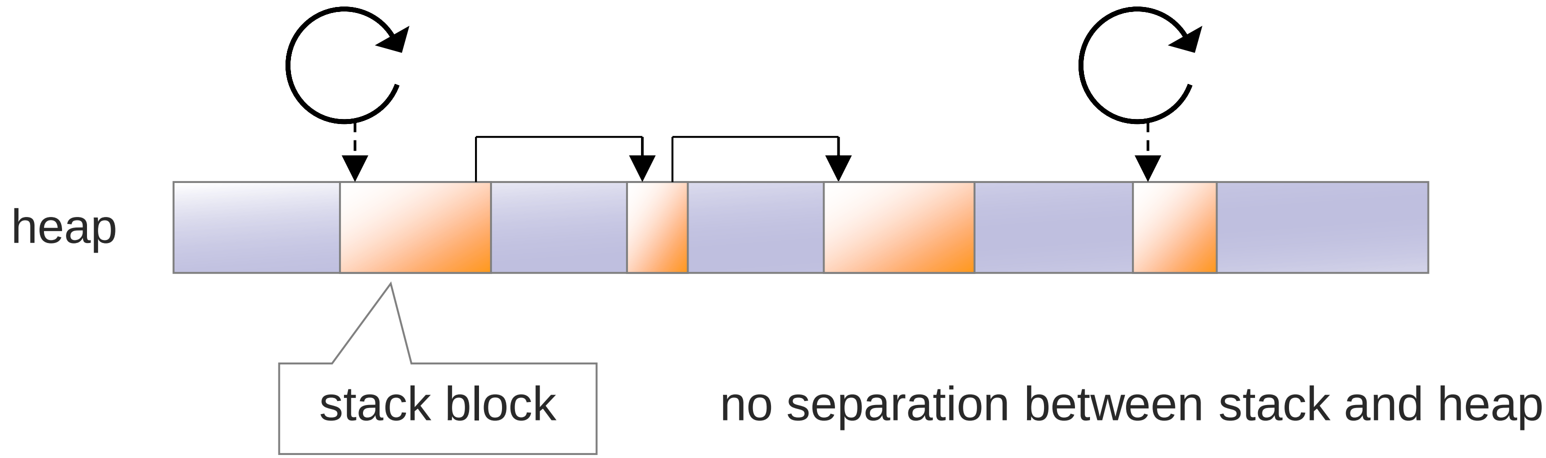
A small operating system for scalable efficient concurrency

- Light-weight processes
  - Dynamic micro stacks
- Fast context switches
  - Direct synchronous switches
  - Economical preemption
- Inbuilt synchronization
  - Protocol-based communication
  - System-managed monitors
- Efficient memory management
  - Hierarchical memory management
  - No virtual memory management

Light-Weight Processes

Micro stacks

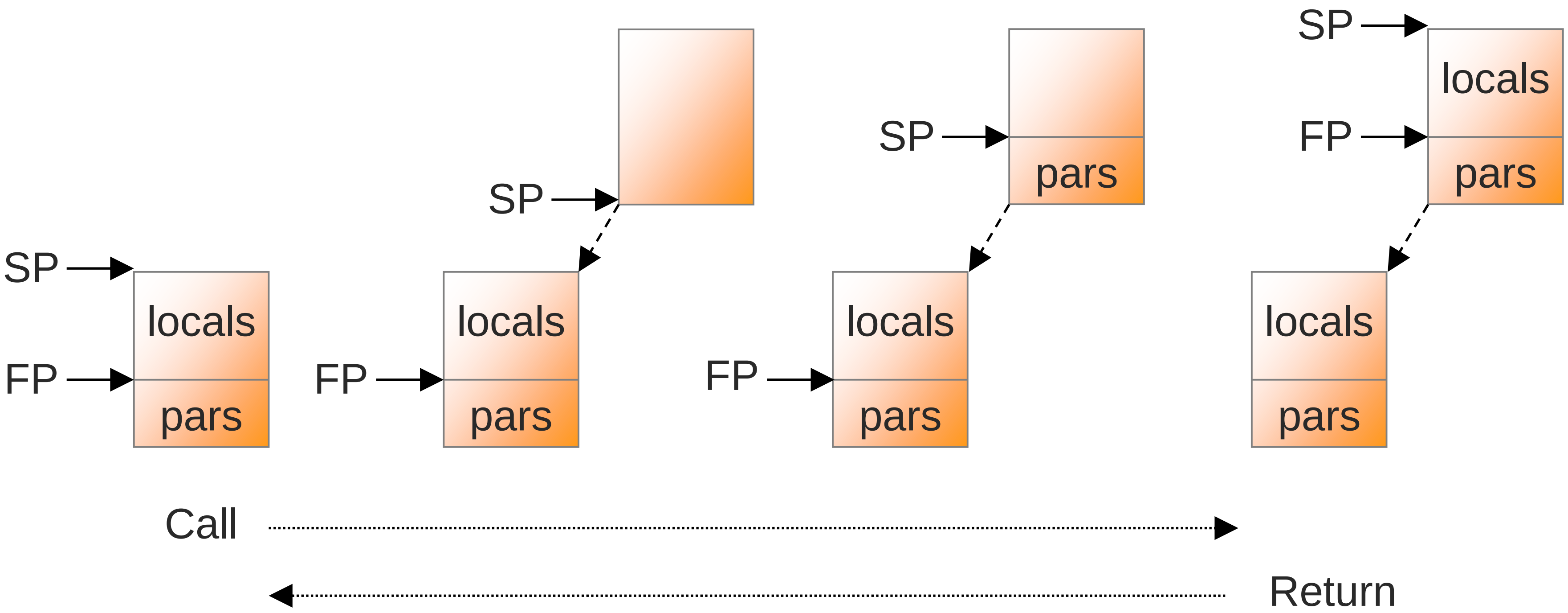
- Arbitrarily small stacks
  - Size not fixed to page granularity
- Stack as a list of blocks of arbitrary size
  - Dynamic extension and reduction



- Initial stack size computed by the compiler
  - Communication instead of methods: less procedure calls
  - Fix stack size for most of the components

Light-Weight Processes

- Dynamic stacks
  - Extension on procedure call and reduction on procedure return
  - Compiler inserts code at a procedure call and return

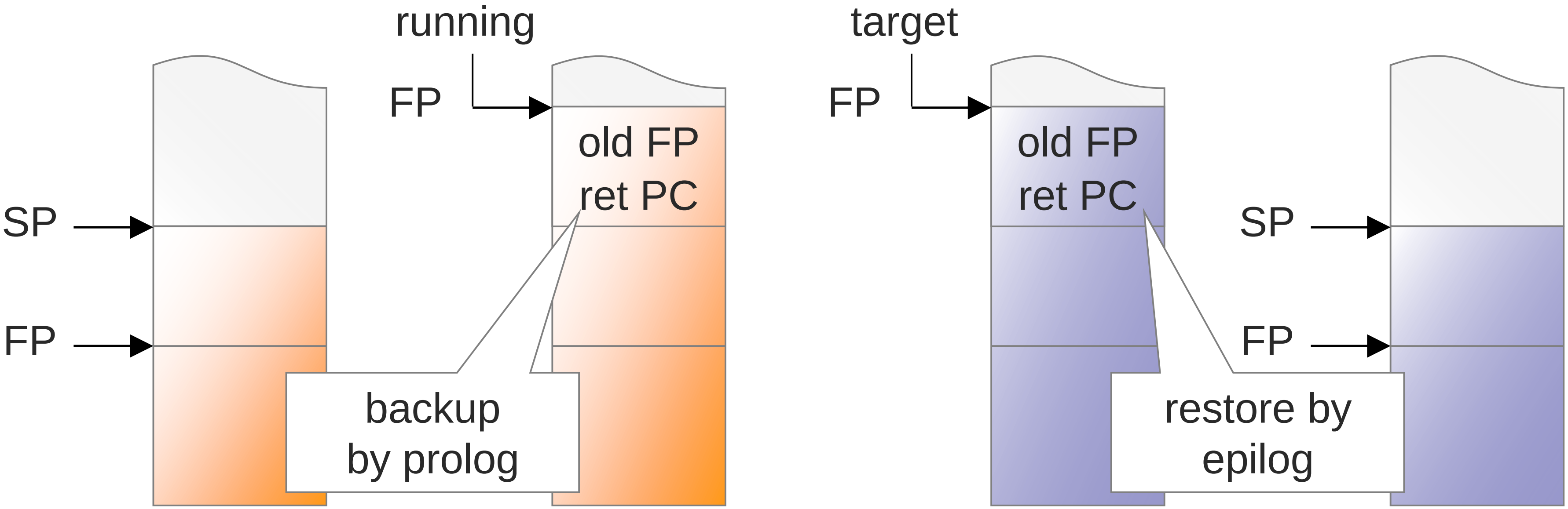


- System calls and interrupts
  - On processor-associated system-stack (run to completion)

Synchronous Context Switch

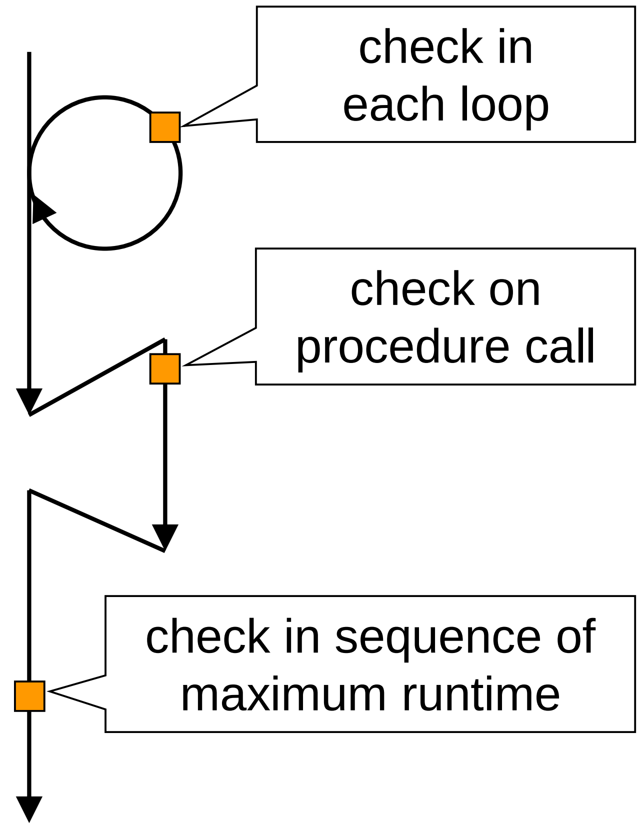
- System call via ordinary procedure call
  - No software interrupt
  - No kernel protection due to safe language
- Direct switch to target process

```
PROCEDURE Switch(target: Process);
BEGIN
  running := REGISTER.FP;
  REGISTER.FP := target
END Switch;
```



Economic Preemption

- Compiler inserts runtime checks in machine code
  - Checks in intervals of guaranteed maximum time
  - Checks initiate preemption on expiration of the time interval
  - Preemption only saves the registers in use on the stack
  - Process does not need unnecessary space to backup unused registers
  - Very fast checks (<0.1% overhead)



register set by the timer interrupt

```
IF Timeout THEN
 Switch(ready)
END
```

call saves the necessary registers

Scaling and Performance

- Maximum number of threads / light weight-processes

Component OS	Windows .NET	Windows JVM	Active Oberon
5,010,000	1,890	10,000	15,700

4GB main memory, City simulation example

- Execution performance for concurrent programs

Program (sec)	Component OS	C#	Java	Oberon AOS
ProducerCons.	**16**	19	130	60
Eratosthenes	**1.8**	6.8	4.6	5.8
TokenRing	**2.1**	22	22	18

6 CPUs Intel Xeon **700MHz**, C# & Java on Windows Server Enterprise Edition

Practical Application (TU Berlin)

Traffic simulation developed in the new language

- More natural modelling
  - Self-active cars
    - All cars drive autonomously and concurrently
    - No explicit program loop moving the cars
    - No explicit parking and waiting queues
  - Virtual time
    - Virtual time corresponds to the time in the simulated world
    - All cars run with a synchronous virtual time
- Faster simulation

explicit discrete event scheduler

Program (min)	Component OS	Thread-based C#	Sequential C++
TrafficSimulation 1,000 cars	**0.04**	33	140
TrafficSimulation 260,000 cars	**76**	*out of memory*	210

6 CPUs Intel Xeon **700MHz**, C# on Windows Server Enterprise Edition

too many threads

Conclusions

A new language for structured concurrent programming

- Conceptual advantages
  - Hierarchically controlled structures instead of references
  - Guaranteed hierarchical encapsulation
  - First-class structured concurrency (race-free)
- Technical advantages
  - High scalability in the number of parallel processes
  - High execution performance for concurrent programs
  - No garbage collector needed for safe memory management
- Practical applicability demonstrated by traffic simulation
  - More natural simulation (self-active cars running in virtual time)
  - Faster than other concurrent and sequential simulations
  - Other concurrent programs have been implemented and run faster

Live Demonstration

Producer Consumer

Token Ring

Traffic Simulation