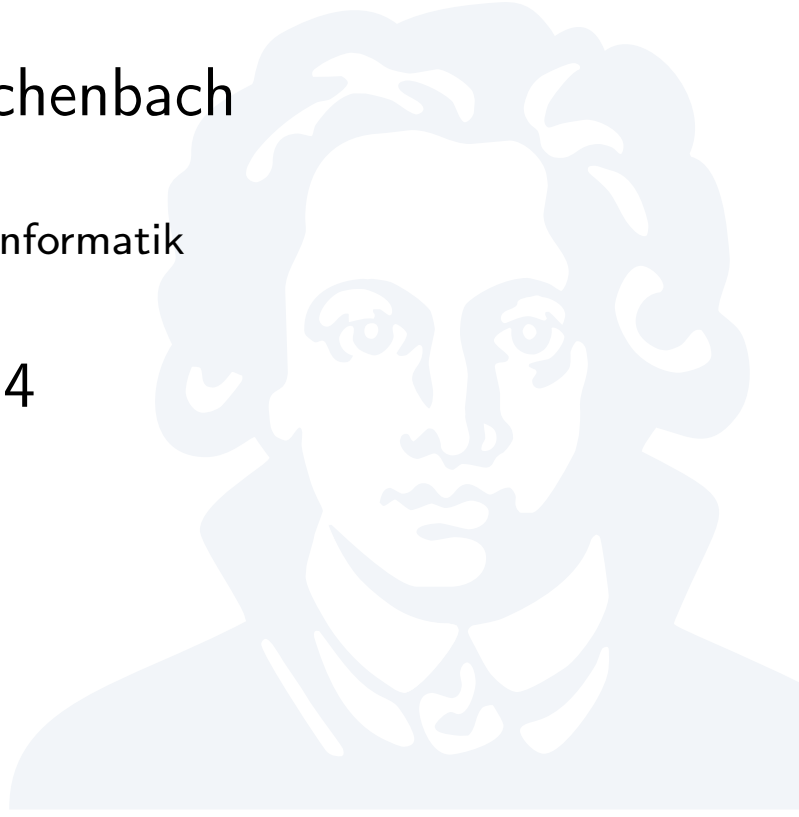
Foundations of Programming Languages

Parameter Passing

Prof. Dr. Christoph Reichenbach

Fachbereich 12 / Institut für Informatik

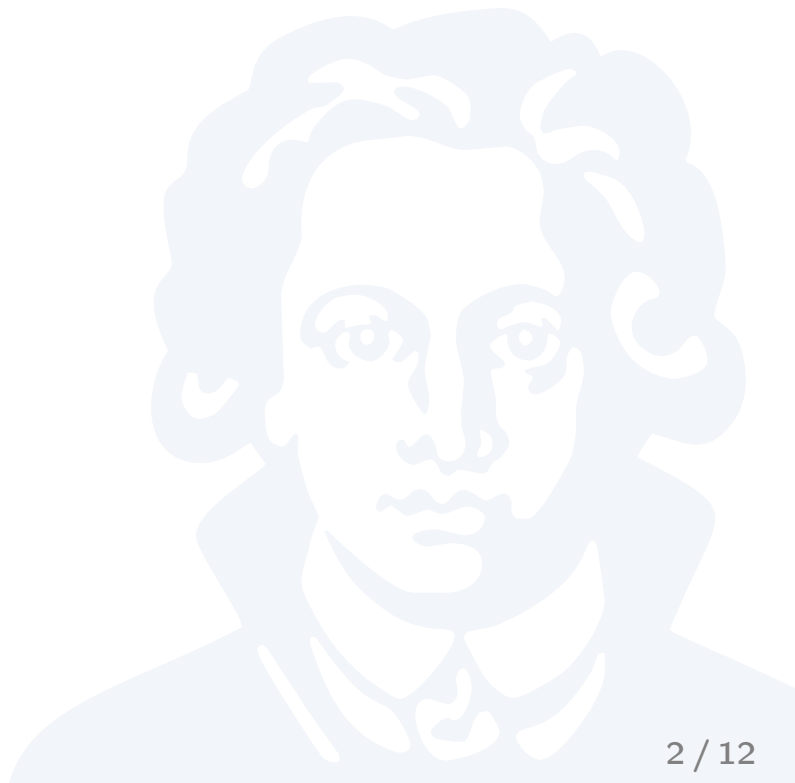
29. Oktober 2014



Parameter Passing Modes

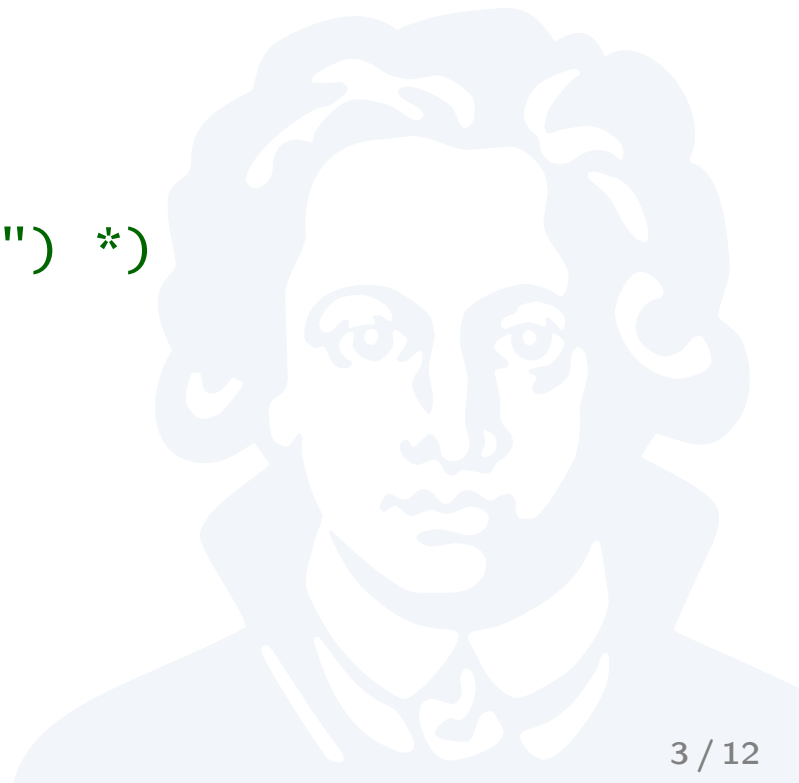
```
f(a, b, c)
```

- Different forms of parameter passing
  - Different semantics
  - Different implementation strategies
- Forms:
  - *pass-by-value*
  - *pass-by-result*
  - *pass-by-value-result*
  - *pass-by-reference*
  - *pass-by-name*
  - *pass-by-need*



Pass-By-Value

```
subprogram f(a, b)
begin
  b := a + 1;
  print("a=", a, ", b=", b);
end

var x := 1;
f(x, x);      (* print("a=1, b=2") *)
print(x);     (* print(1) *)
```



Parameter Evaluation Order

```
subprogram p(a)
begin
  print(a);
  return a;
end

f(p(1), p(2)); (* print 1,2 or 2,1? *)
```

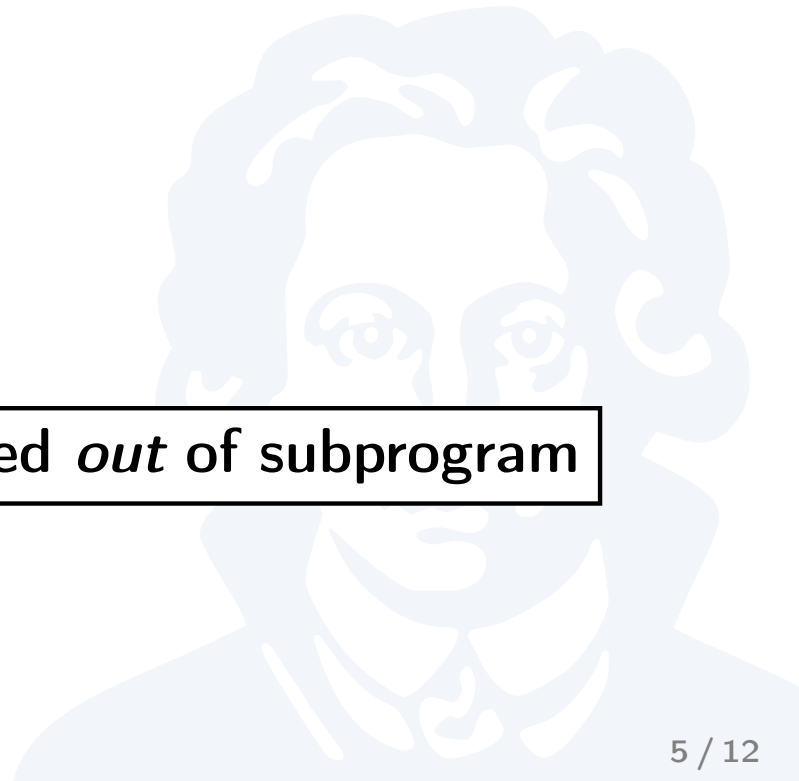
- *Evaluation order*: order in which parameters are computed
- Different languages choose different orders:
  - left-to-right (Java, ML, ...)
  - right-to-left
  - undefined (C, ...)

Evaluation order is irrelevant with referential transparency

Pass-By-Result

```
subprogram f(a, out b)
begin
  b := a + 1;
  print("a=", a");
end

var x := 1;
f(x, x);      (* print("a=1") *)
print(x);     (* print(2) *)
```

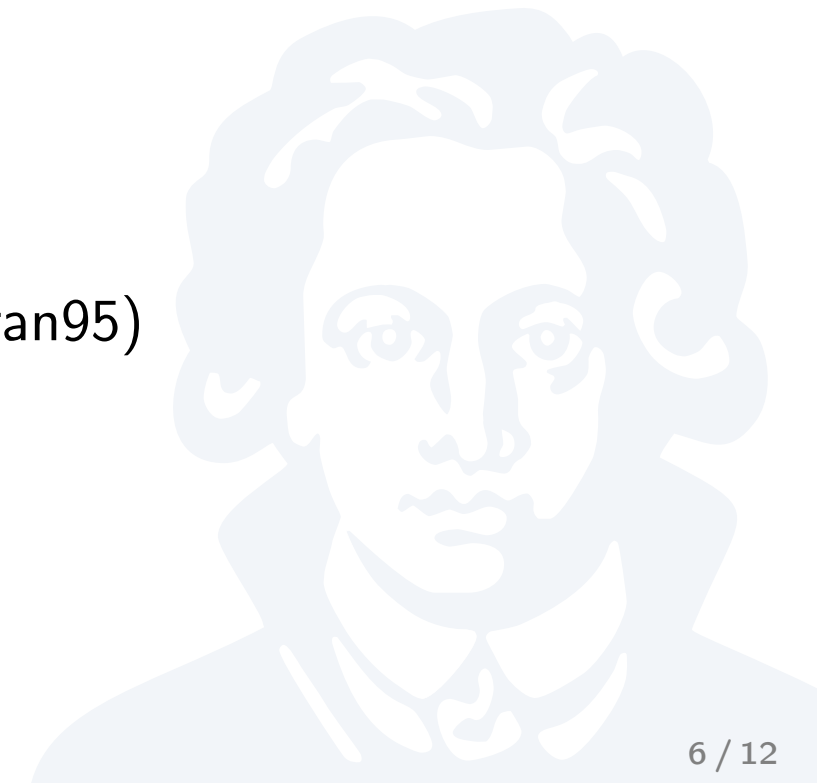
Pass-By-Result parameters are copied *out* of subprogram



Pass-By-Result: Return Order

```
subprogram f(out a, out b)
begin
  a := 0;
  b := 1;
end

var x;
f(x, x);        (* x = ? *)
```

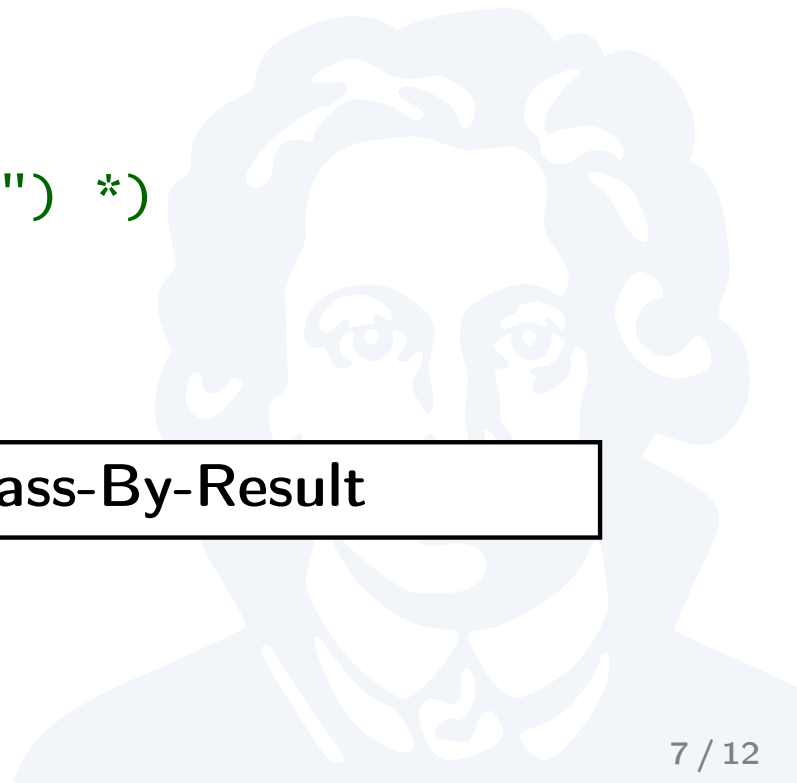
- Language defines copy-out order:
  - left-to-right ($x = 1$) (e.g., Fortran95)
  - right-to-left ($x = 0$)
  - disallowed (e.g., Ada95)



Pass-By-Value-Result

```
subprogram f(in-out a, in-out b)
begin
  b := a + 1;
  print("a=", a, ", b=", b);
end

var x := 1;
var y := 1;
f(x, y);      (* print("a=1, b=2") *)
print(y);     (* print(2) *)
```

Combines Pass-By-Value, Pass-By-Result

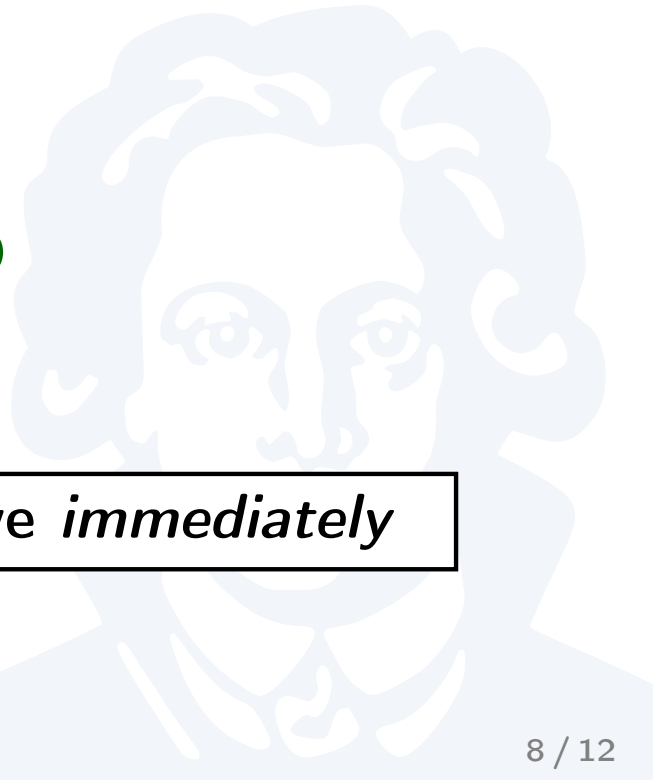


Pass-By-Reference

- Pass *memory address* of parameter

```
subprogram f(ref a, ref b)
begin
  b := a + 1;
  print("a=", a, ", b=", b);
end

var x := 1;
f(x, x);      (* print("a=2, b=2") *)
print(x);     (* print(2) *)
```

Unlike Pass-By-Result: Updates effective *immediately*

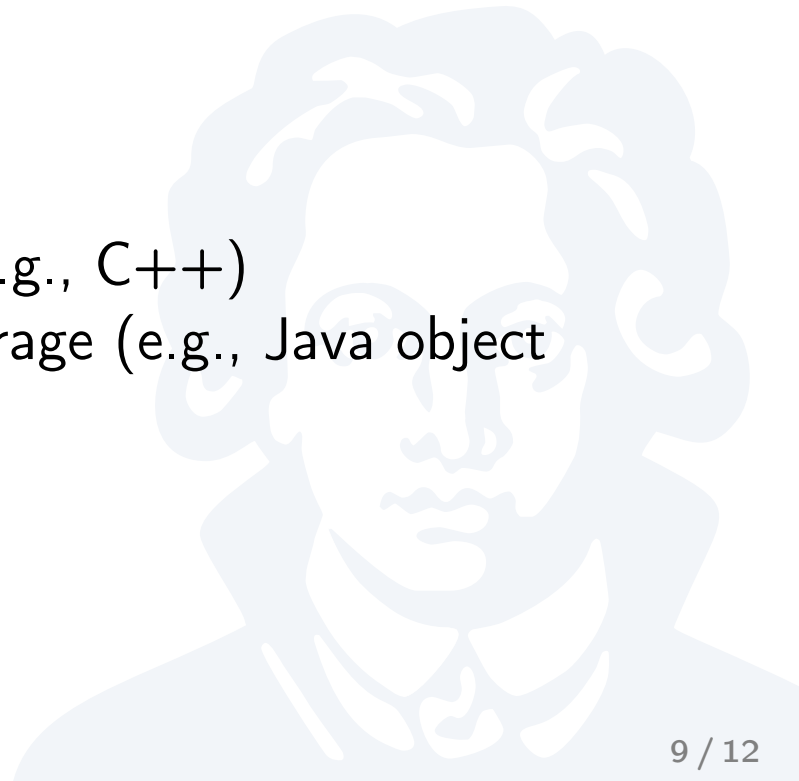


Pass-By-Reference: Viable Parameters

```
subprogram f(ref a)
...
```

```
f(x + 1)
```

- `x + 1` has no memory address
- Options:
  - Language disallows such calls (e.g., C++)
  - Language creates temporary storage (e.g., Java object parameters)



Pass-By-Name

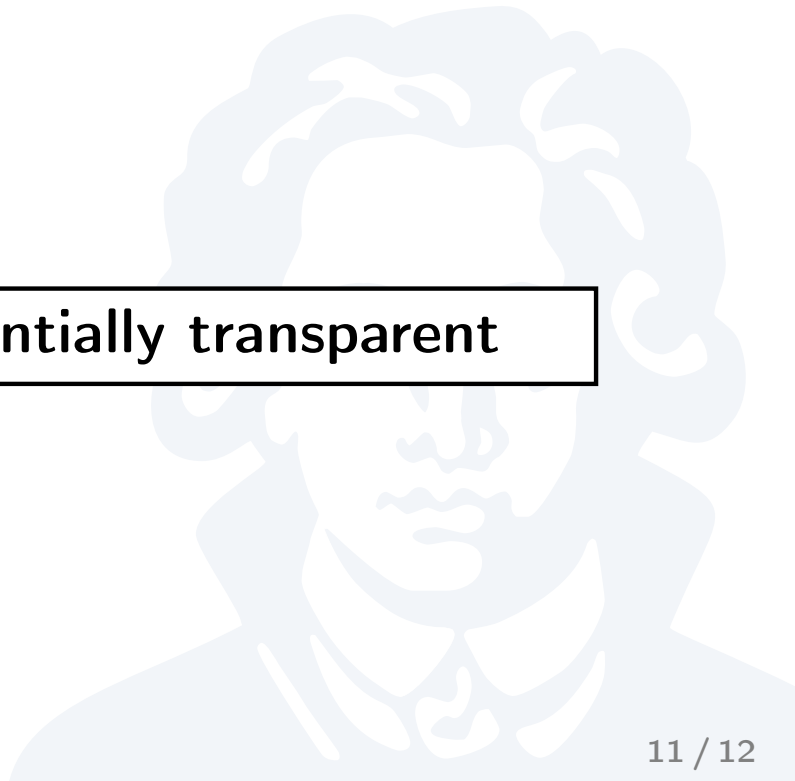
```
var count = 0;
subprogram next(a)
begin
  count := count + a;
  return count;
end

subprogram f(name b)
begin
  print("b1=", b, ", b2=", b);
end

f(next(2));      (* print("b1=2, b2=4") *)
```

Actual parameter is evaluated as many times as needed

Pass-By-Need

- Similar to Pass-By-Name
- Parameter evaluated at most once
- Usage: e.g., Haskell

Complex semantics, unless referentially transparent



Summary

- Different parameter passing modes:
  - *Pass-By-Value*: copy caller $\rightarrow$ callee
  - *Pass-By-Result*: copy callee $\rightarrow$ caller
  - *Pass-By-Value-Result*: caller $\rightarrow$ callee $\rightarrow$ caller
  - *Pass-By-Reference*: memory address
  - *Pass-By-Name*: pass code for evaluating parameter
  - *Pass-By-Need*: as pass-by-name, ensure evaluation happens at most once
- Parameter evaluation order:
  - At call site: which actual parameter do we evaluate first?
  - Relevant for parameters with side effects