# Good Ideas - Revisited

Niklaus Wirth

September 2005

Moscow State University

Computer Architecture

Programming Languages

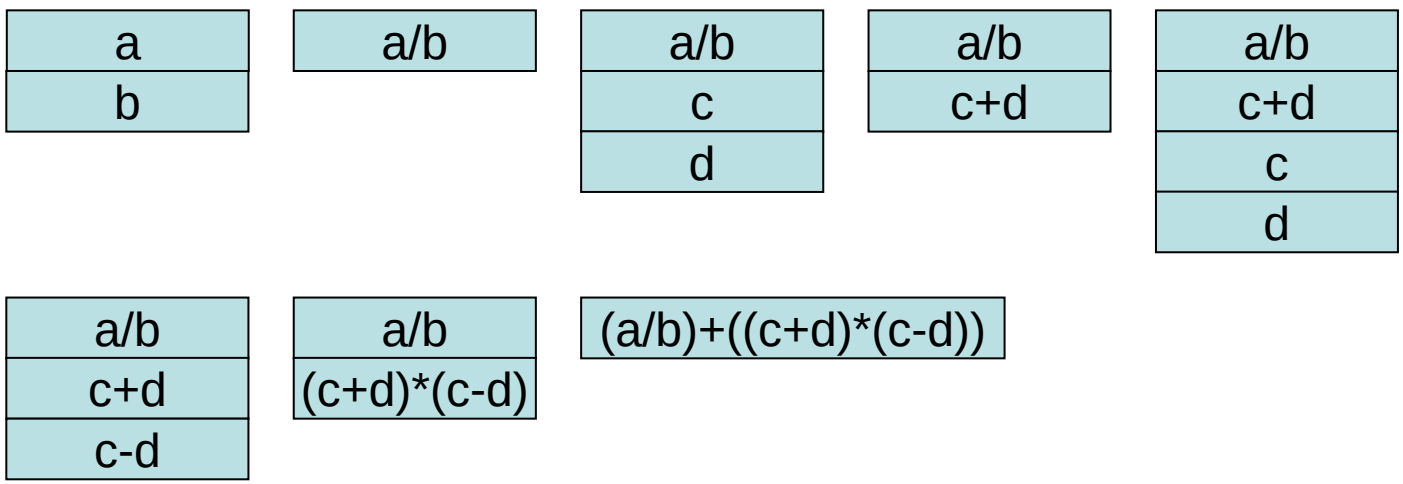
Miscellaneous Techniques

Programming Paradigms

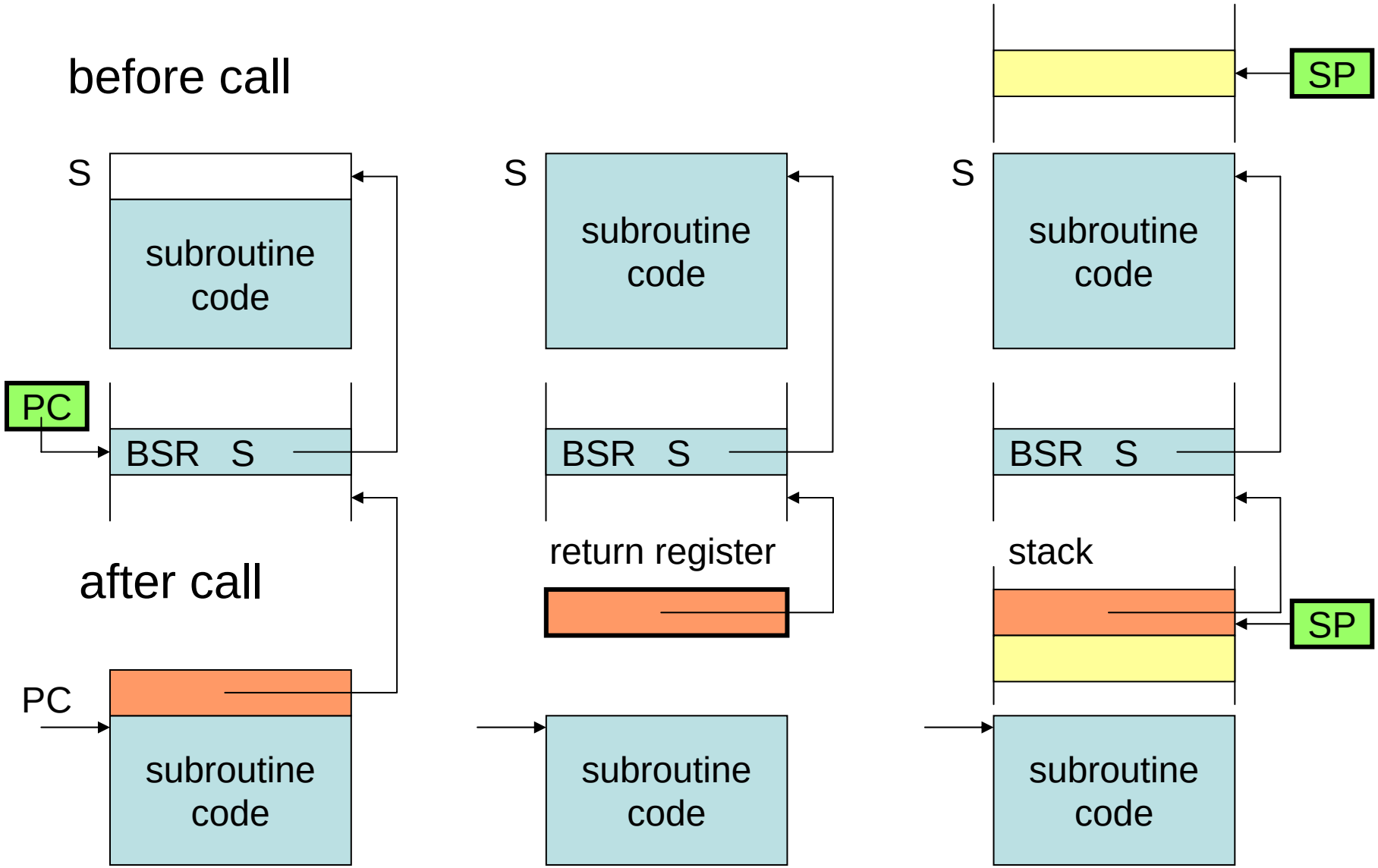
# Expression stacks

(a/b) + ((c+d)*(c-d))

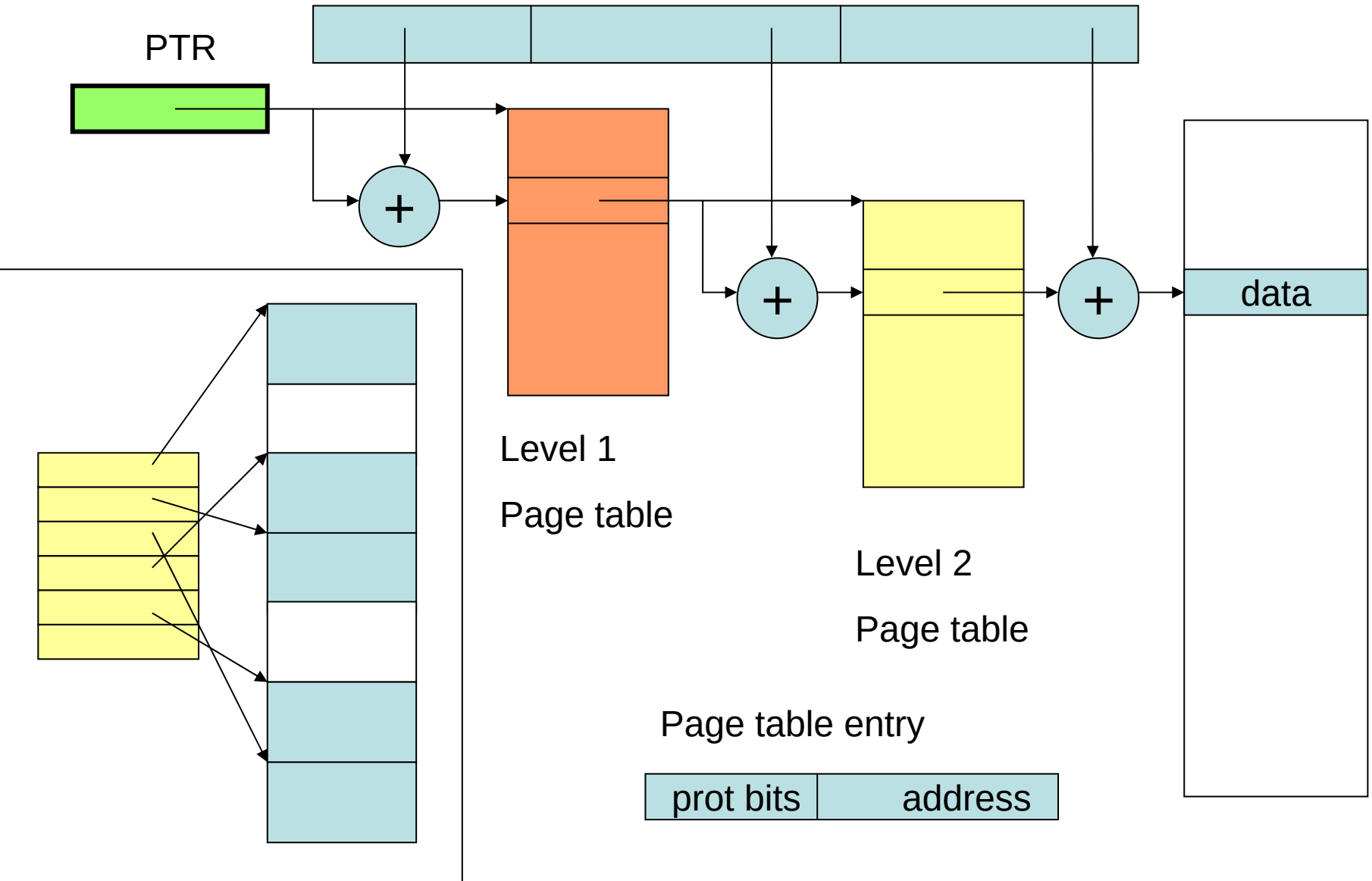
a b / c d + c d - * +



# Subroutine return addresses



# Virtual addressing



# Benefit and cost of virtual addressing

Benefits:

- Simplifies allocation in multiprocessing
- Simplifies reuse of pages
- Protects from illegal access

Cost:

- Multiple memory access via page tables
- Additional hardware
- Efficiency requires cashes

# Simple vs. complex instruction sets

Simple instruction set -> simple hardware

What led to complex instruction sets?

- High-level language constructs
- Wish for code density
- Microprogramming
- Descriptor-architectures (B5500)

The concept of computer families

The return to RISC architectures

# Programming Language Features

- Notation and syntax
- Algol’s FOR statement
- Algol’s OWN variables
- Algol’s name parameter

# Notation and Syntax

- Assignment operator  x = y  x := y

  z = x ++ y  b = x == y

  x +++++y  ++x+++y+1

  x+++++y+1==++x+++y

  x+++y++==x+++++y+1
- APL:  x-y-z means x-(y-z), but not (x-y)-z
- Confusion between statement and expression
- Statements are executed
- Expressions are evaluated

# Syntax (Algol and Pascal)

**if** b **then** S0 **if** b **then** S0 **else** S1

**if** b0 **then if** b1 **then** S0 **else** S1

**if** b0 **then if** b1 **then** S0 **else** S1

**if** b0 **then if** b1 **then** S0 **else** S1

if p then for i := 1 step 1 until n do if q then S1 else S2

if p then for i := 1 step 1 until n do if q then S1 else S2

**if** b **then** S0 **end** **if** b **then** S0 **else** S1 **end**

# Algol's complicated **for** ststement

```
for i := 1 step 1 until n do a[i] := 0
for i := 2, 3, 5, 7, 11 do a[i] := 0
for i := x, x+1, x*(y+z) do a[i] := 0
for i := i+1 while i < n do a[i] := 0
for i := x-3, x step k until y, y+7, while z < 20
   do a[i] := 0
for i := 1 step 1 until i+1 do a[i] := 0
for i := 1 step i until i do i := -i
```

# Algol's **own** variables

```
procedure P(x, y);
begin integer z; z := x; x := y; y := z
end

real procedure random;
begin own real x; x := (x*a + b) mod c;
   randon := x
end
```

# Algol's name parameter

**real procedure** square(x); real x; square := x*x

square(a)  literally means  a*a

square(sin(a)*cos(a))  stands for
  sin(a)*cos(a)*sin(a)*cos(a)

**real procedure** square(x);
  value x; real x; square := x*x

stands for

  **begin real** x'; x' := x; square := x'*x' **end**

# Algol's Jensen device

```
real procedure sum(k, x, n);
begin real s;
   for k := 1 step 1 until n do s := s+x;
   sum := x
end
```

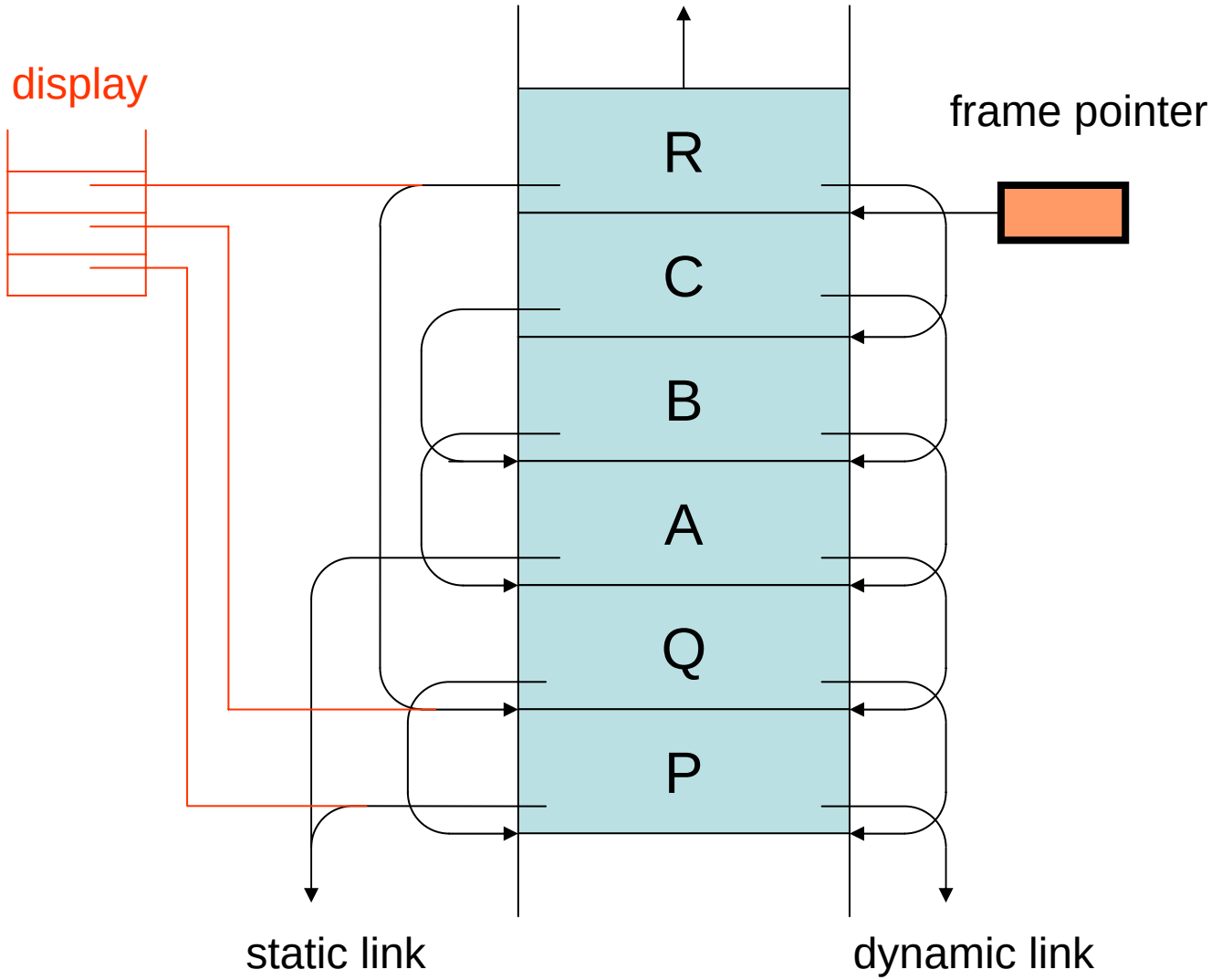
| | |
|---|---|
| $a_1 + a_2 + \ldots + a_{100}$ | sum(i, a[i], 100) |
| **a** × **b** | sum(i, a[i]*b[i], 100) |

# Dijkstra's "display"

```
procedure A(proc h);
begin integer x;
   procedure B;
   begin integer y;
        procedure C;
        begin integer z; h
        end ;
        C
   end ;
   B
end
```

```
procedure P;
begin integer i;
   procedure Q;
   begin integer j;
        procedure R;
        begin integer k;
        end ;
        A(R)
   end ;
   Q
end
```

$\rightarrow P \rightarrow Q \rightarrow A \rightarrow B \rightarrow C \rightarrow R$



# Functional programming

- What distinguishes a functional language from a procedure language?
- FP: exclusively function application
- No state, no variables
- FPLs have sneaked in state and assignment
- No side effects
- Eases detection of potential parallelism
- Academic exercise ?

# Logic programming

- Prolog
- Implementation is a search engine for solutions satisfying given predicates
- Logic inference engine
- In practice requires hints in the form of cuts
- One must understand the functioning of the hidden engine
- An academic exercise ?

# Object-oriented programming

- Technically based on 2 concepts only:
  - procedure types of variables
  - type extension (inheritance)
- View of procedures as belonging to objects
- Terminology

| | |
|---|---|
| object | (record typed) variable |
| class | (record) type |
| method | (record bound) procedure |
| send msg | call procedure |

# “Good ideas” of today?

- Many of the good ideas of their time have become mediocre or even bad ideas
  - Because of technology changes
  - Because of shifts of goals and habits
  - Because of too much emphasis on efficiency
- Which are today’s “good ideas”?
  - Will they also turn mediocre?
  - Or are they bad already now?