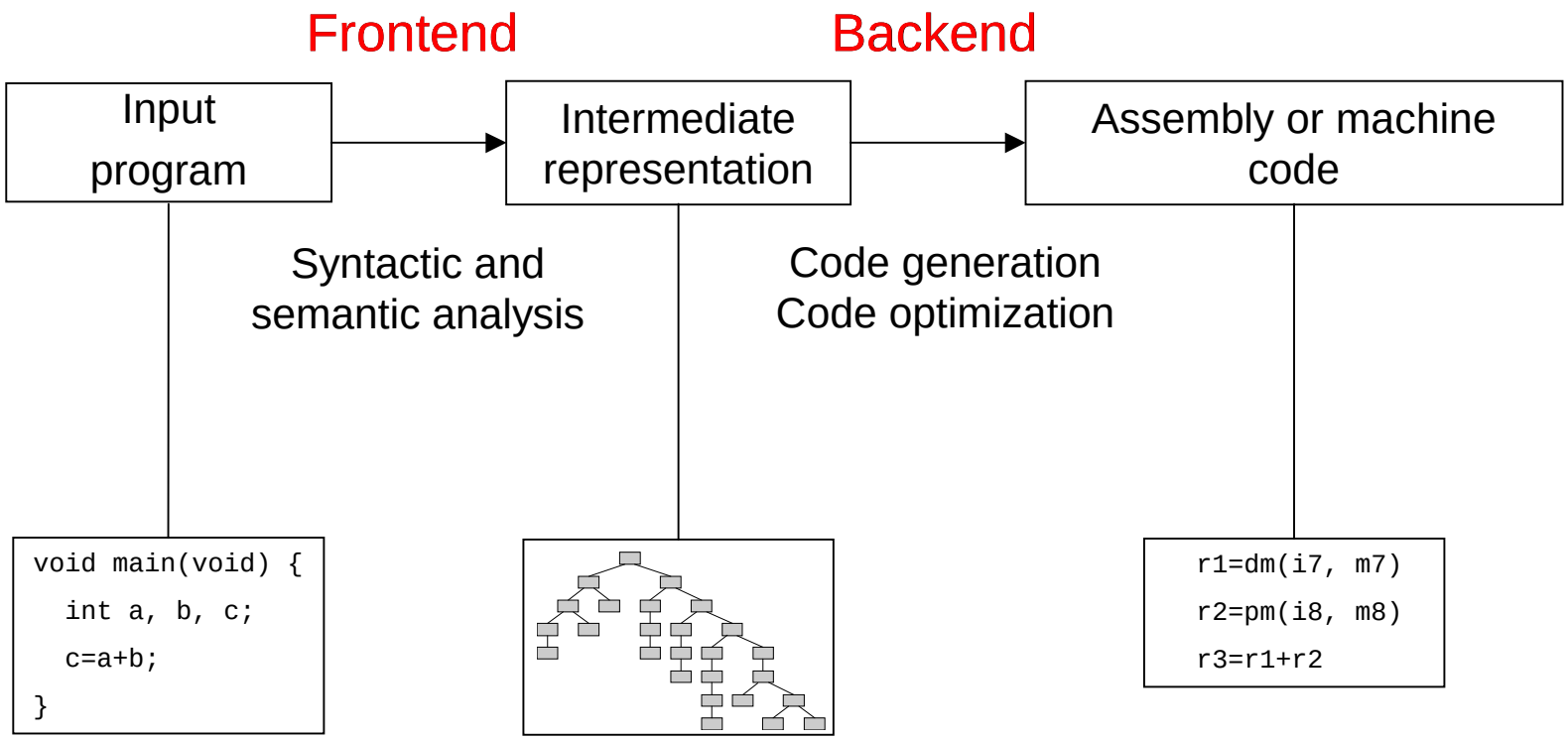
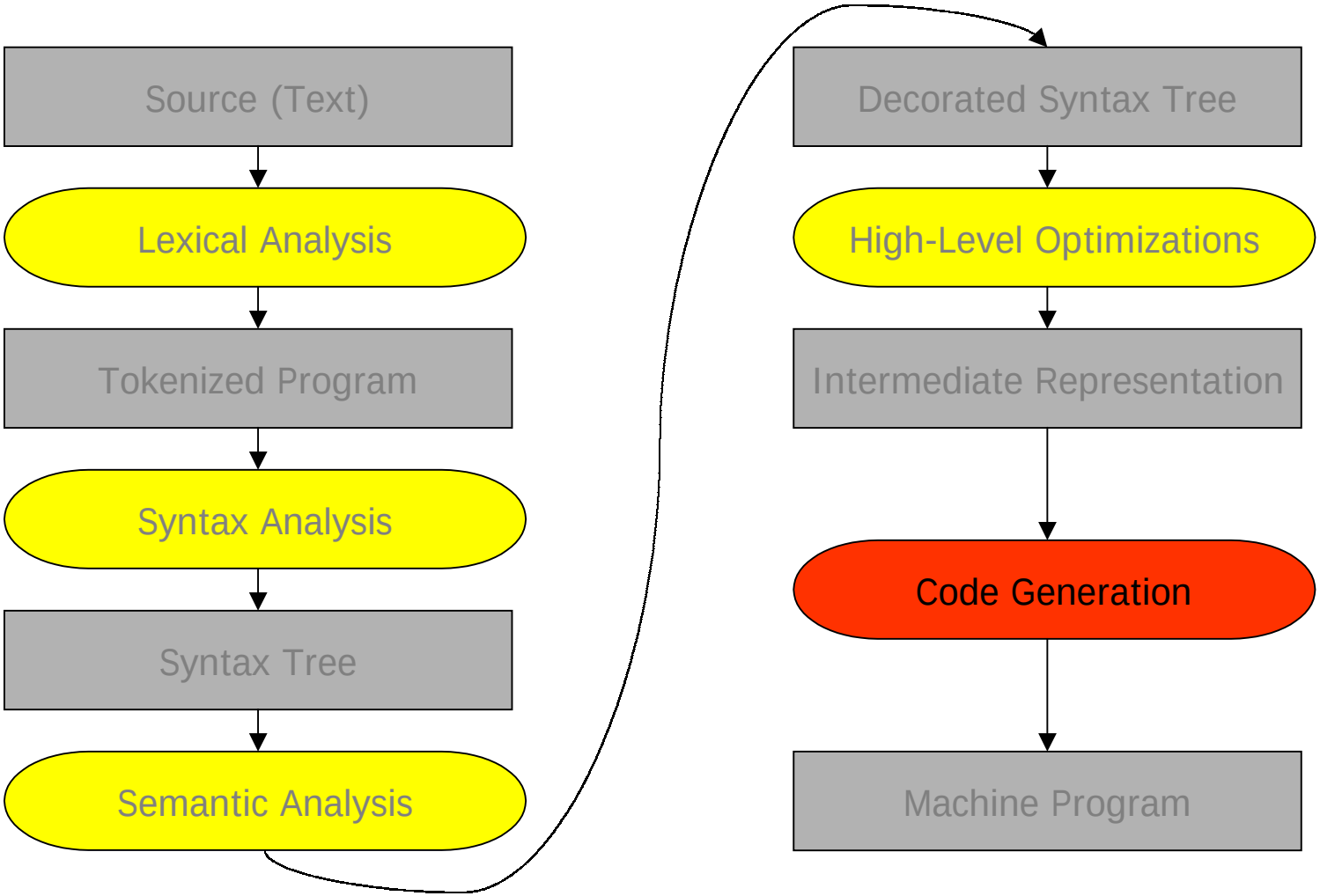
Compiler Structure



Detailed Compiler Structure

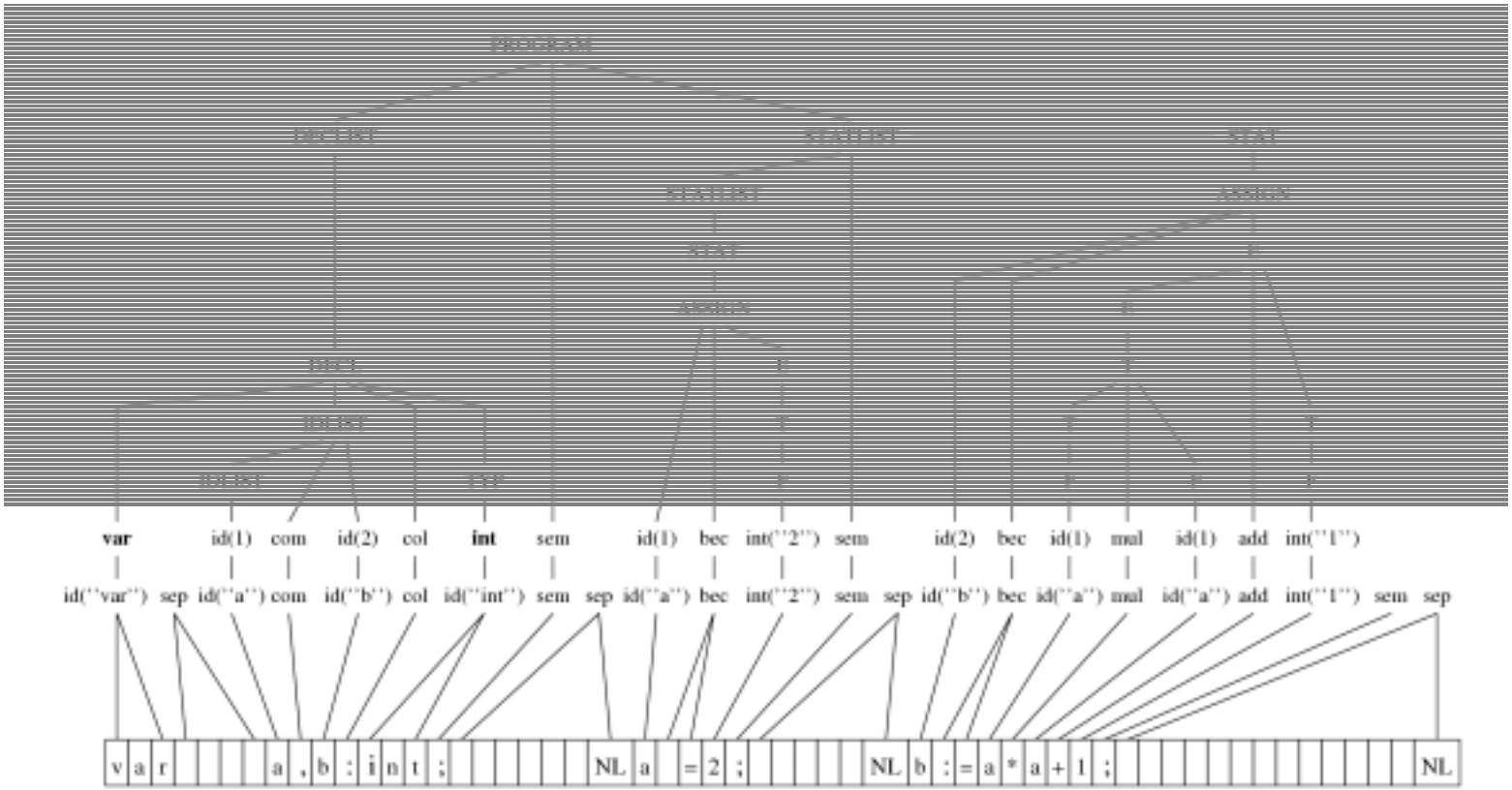


Lexical Analysis (Scanning & Screeing)

- Input: Program text as sequence of characters.
  Output: Program text as sequence of symbols (tokens).

1. Read Input file.
2. Report errors about symbols illegal in the programming language.
3. Screening subtask:
   - Identify language keywords and standard identifiers.
   - Elimininate "white-spaces", e.g. consecutive blanks and newlines.
   - Count line numbers.

Lexical Analysis (Scanning)

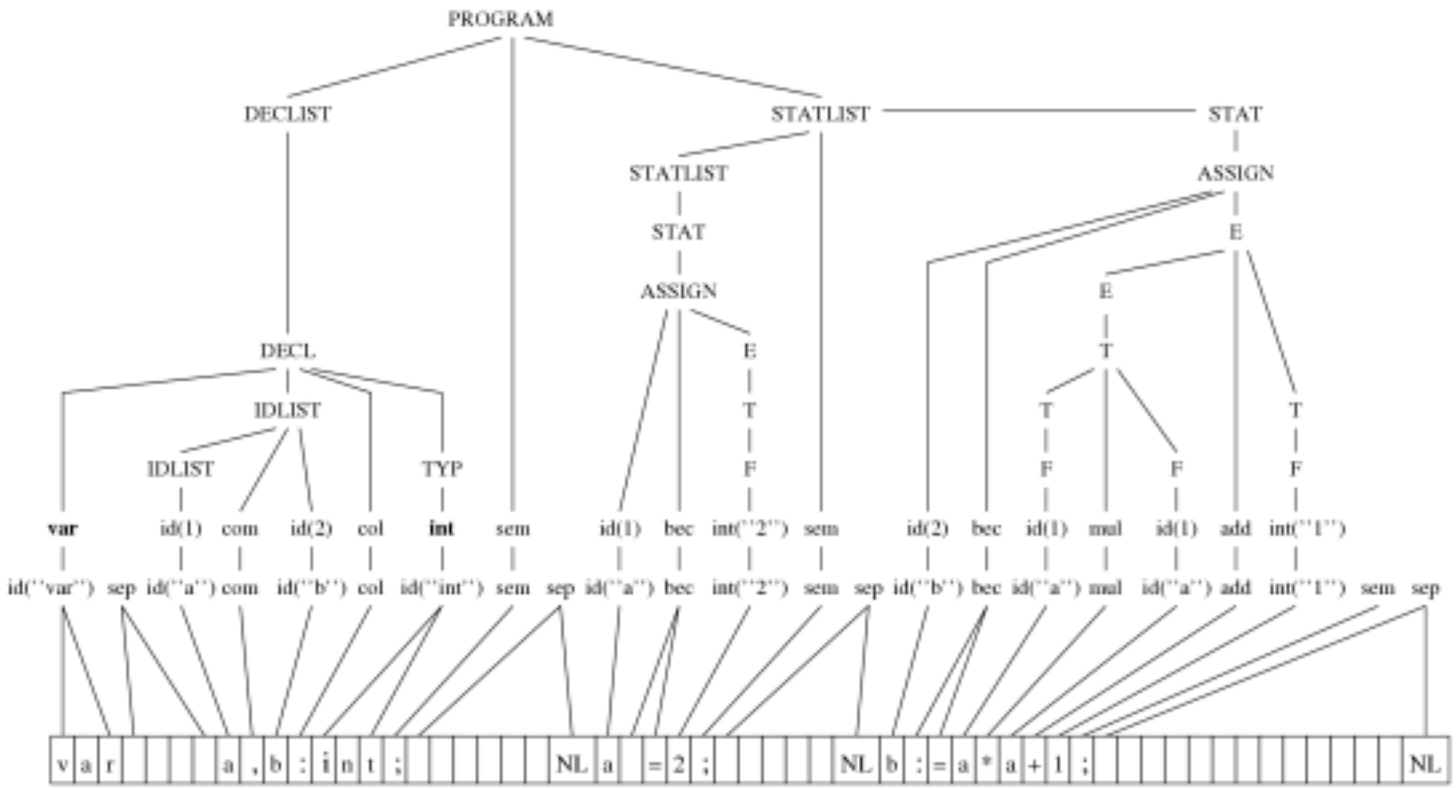


Syntax Analysis (Parsing)

- Input: Sequence of symbols (tokens).
  Output: Structure of the program:
  - concrete syntax tree,
  - abstract syntax tree, or
  - parse (derivation).

- Syntax errors:
  - report (as many as possible) syntax errors,
  - diagnose syntax errors,
  - correct syntax errors.

Syntax Analysis (Parsing)

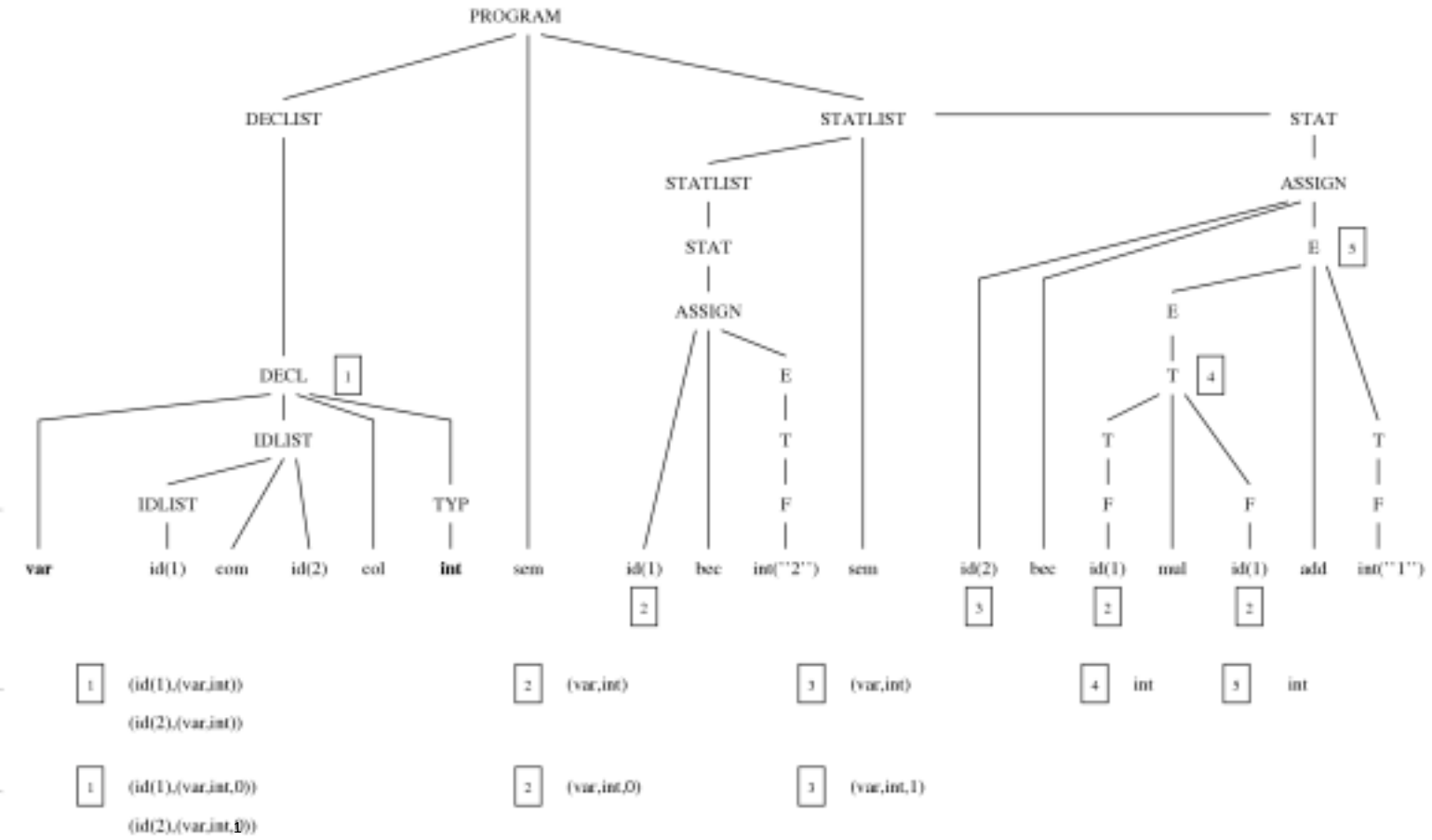


Semantic Analysis

- Input: Abstract syntax tree.
  Output: Abstract syntax tree decorated with attributes, e.g. types of subexpressions.

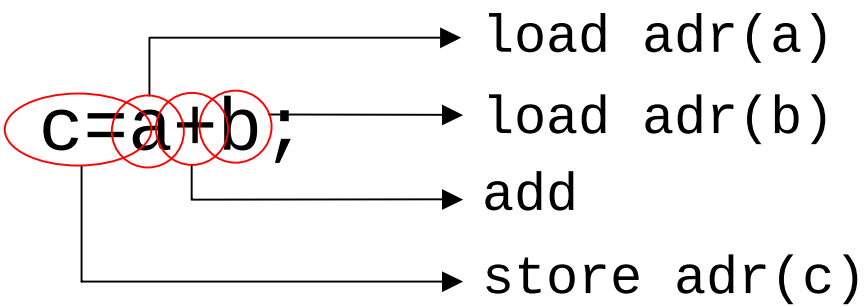
- Report semantic errors, e.g. undeclared variables, type mismatches.
- Resolve usage of variables: identify applied occurrences of variables with their declarations.
- Compute type of every (sub-)expression.

Semantic Analysis

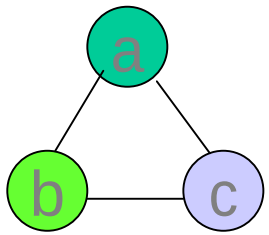


Code Selection, Register Allocation, and Instruction Scheduling

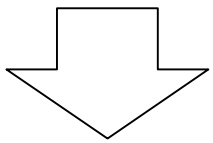
Code selection



Register allocation



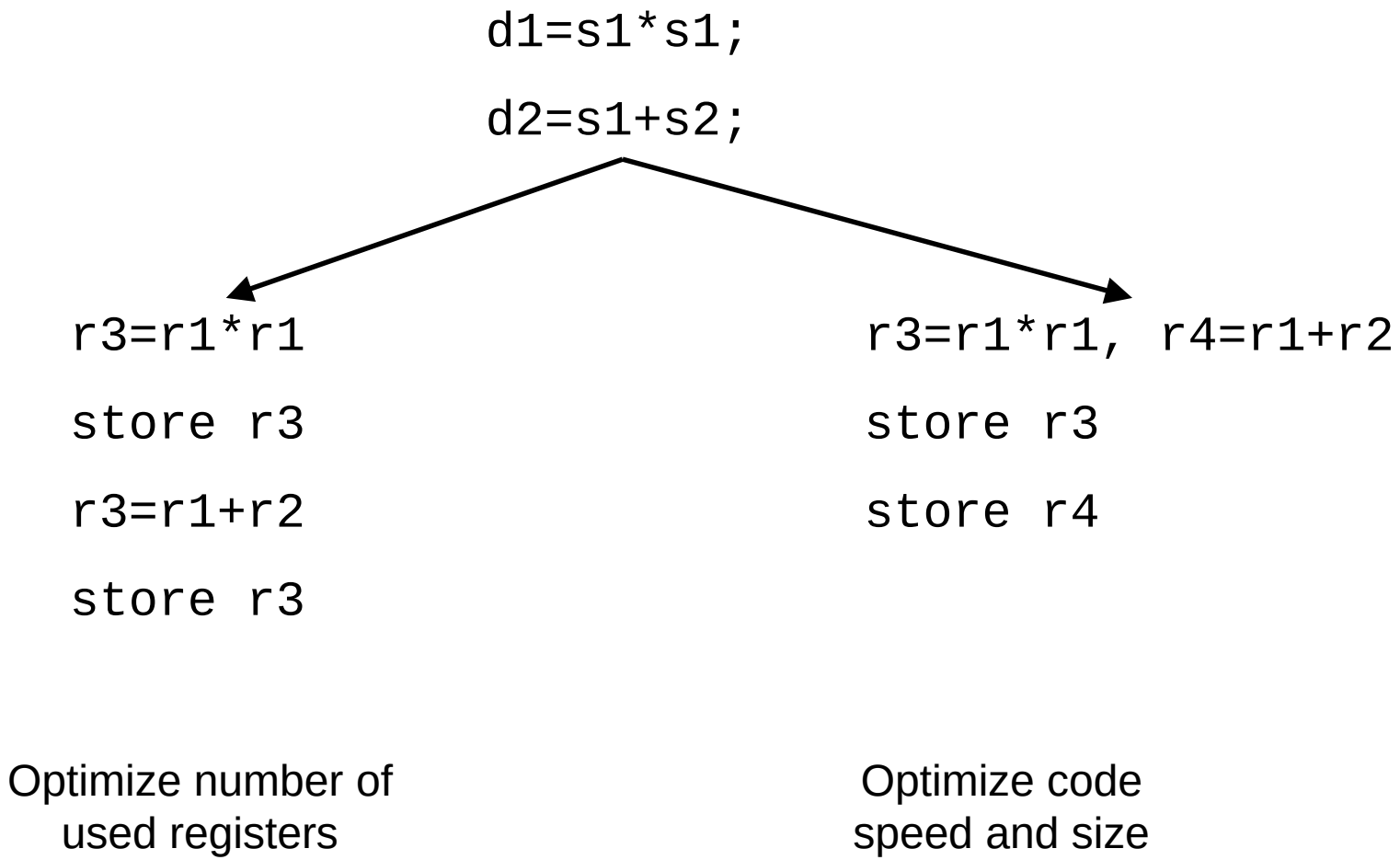
$a \mapsto r1$
$b \mapsto r2$
$c \mapsto r3$



Instruction scheduling

```
r1=load adr(a) || r2=load adr(b)
r3=add r1, r2
store adr(c), r3
```

Phase Coupling Problem



Main Tasks of Code Generation (1)

- Code selection: Map the intermediate representation to a semantically equivalent sequence of machine operations that is as efficient as possible.
- Register allocation: Map the values of the intermediate representation to physical registers in order to minimize the number of memory references during program execution.
  - Register allocation proper: Decide which variables and expressions of the IR are mapped to registers and which ones are kept in memory.
  - Register assignment: Determine the physical registers that are used to store the values that have been previously selected to reside in registers.

Main Tasks of Code Generation (2)

- Instruction scheduling:
  Reorder the produced operation stream in order to minimize pipeline stalls and exploit the available instruction-level parallelism.

- Resource allocation / functional unit binding:
  Bind operations to machine resources, e.g. functional units or buses.

The Code Generation Problem

- Instruction scheduling, register allocation and code selection are NP complete problems.
- In classical approaches they are addressed by heuristic methods in separate phases.
- Unfortunately, all the code generation phases are interdependent, i.e. decisions made in one phase may impose restrictions to the other phases.
- Thus: often suboptimal combination of suboptimal partial results.
- Moreover: specific/irregular hardware features not well covered by standard code generation methods.

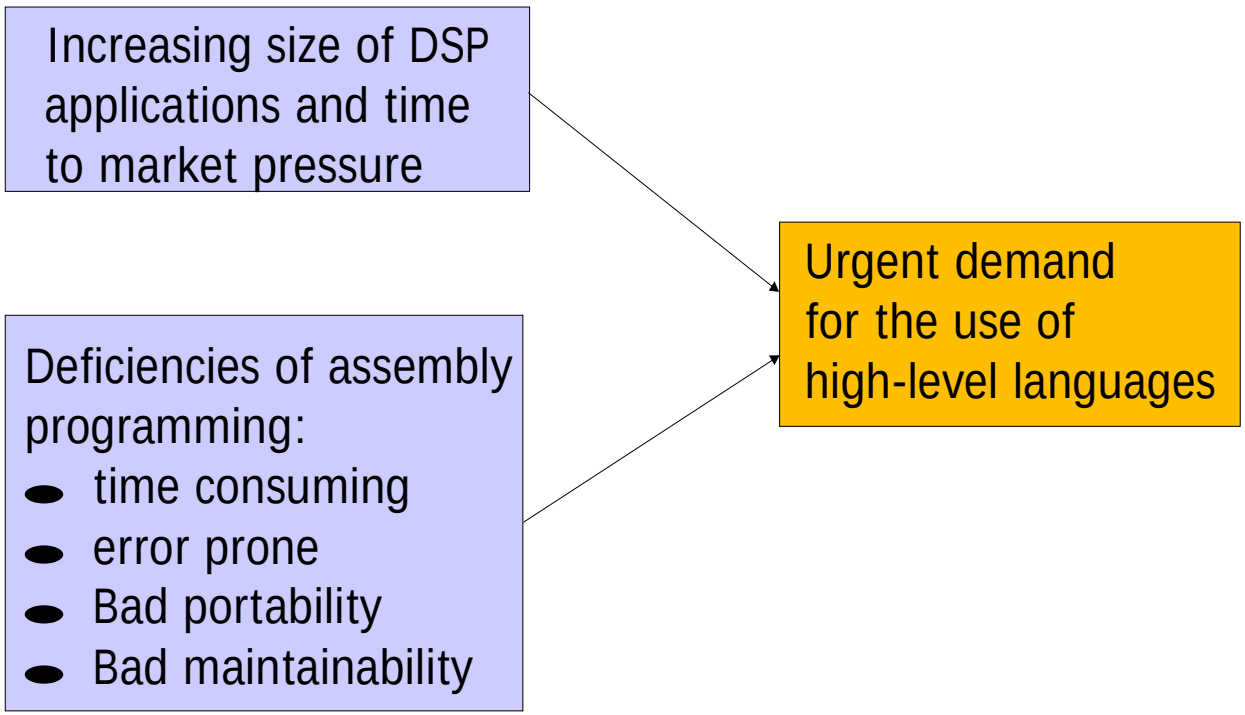
The DSPStone Study

- Evaluation of the performance of DSP compilers and joint compiler/processor systems. Evaluated compilers:
  - Analog Devices ADSP2101,
  - AT&T DSP1610,
  - Motorola DSP56001,
  - NEC mPD77016,
  - TI TMS320C51.
- Hand-crafted assembly code is compared to the compiler-generated code.

- Result: overhead between 100% and 1000% of compiler-generated code is typical !

Compiling for DSPs

- Code quality of traditional high-level language compilers is not satisfactory.
- Thus: Assembly programming.



Case Study: Infineon TriCore

C code (FIR Filter):

```
int i,j,sum;
for (i=0;i<N-M;i++) {
  sum=0;
  for (j=0;j<M;j++) {
    sum+=array1[i+j]*coeff[j];
  }
  output[i]=sum>>15;
}
```

Compiler-generated code (gcc):

```
.L21: add %d15,%d3,%d1
      addsc.a %a15,%a4,%d15,1
      addsc.a %a2,%a5,%d1,1
      mov %d4,49
      ld.h %d0,[%a15]0
      ld.h %d15,[%a2]0
      madd %d2,%d2,%d0,%d15
      add %d1,%d1,1
      jge %d4,%d1,.L21
```

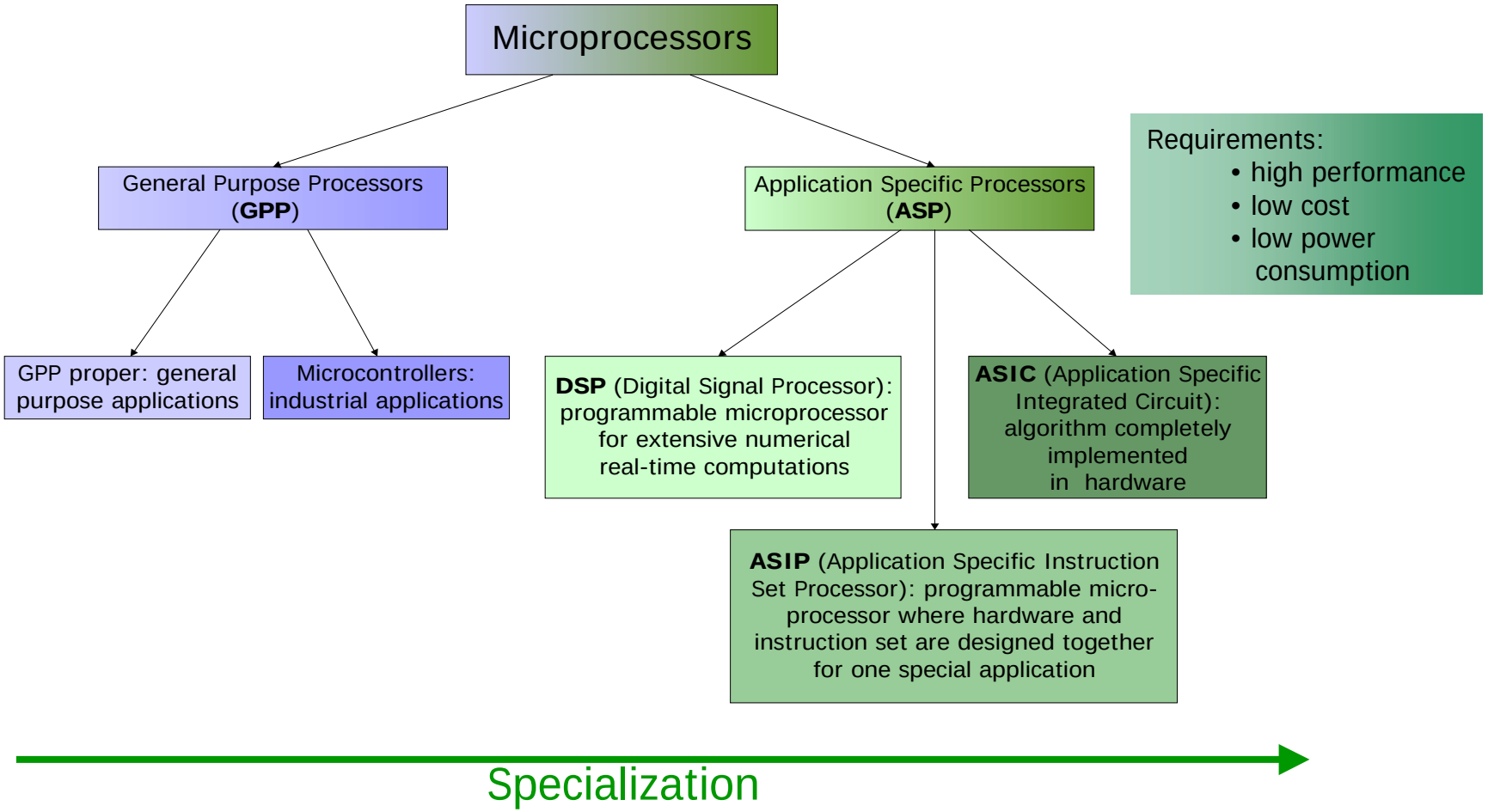
Hand-written code (inner loop):

```
_8: ld16.w d5, [a2+]
    ld16.w d4, [a3+]
    madd.h e8,e8,d5,d4ul,#0
    loop a7,_8
```

- ✓ Zero-Overhead Loops
- ✓ SIMD Instructions
- ✓ Auto-modify addressing

→ **6 times faster!**
(execution in SRAM)

Classification of Microprocessors



Architectural Valuation

- [Campbell, Northrop Grumman Corporation, 1998]
- More efficient architectures will use less energy to complete the same task on the same generation CMOS solid state technology
- Power consumption $P = CV^2 f \Delta N$
  - C : Capacitance
  - V : CPU Core Voltage
  - f : CPU clock frequency
  - ΔN : Number of gates changing state

Architectural Valuation

- Observations:
  - Higher performance by increasing the clock frequency does not change the performance per power ratio

$$\frac{Performance}{Power} = \frac{\frac{O}{Cycle} * \frac{Cycle}{Time}}{CV^2 f \Delta N} = \frac{Of}{CV^2 f \Delta N} = \frac{O}{CV^2 \Delta N}$$

  - A voltage decrease improves performance per power non-linearly

$$\frac{Performance}{Power} = \frac{O}{CV^2 f \Delta N}$$

Architectural Valuation

- Architectural specialization as a measure for how well the architecture fits a given target application.

- Estimation of architectural specialization:
  Performance per power.

Comparion of Performace Per Power Ratios