8. The Complete View: Some Software Architectures

Aims

- Chapter 4 Architectural Language
  (6 Translation to Programming Languages)
  5 Subarchitectures
  now complete Architectures
- Chapter 4 Textual Architecture Language (plus code of bodies)
  5 Mostly on graphical language level
  now completely on graphical level
- Now solution for Telegram example: batch system
  Card Box example: interactive system
  Coffee Machine example: embedded system
- Furthermore, also big examples compiler: batch system
  tool environment: interactive system
- Importance of Brainstorming “What can change ?“
- Show architecture of a system in different versions

Contents of Chapter 8

to be elaborated:

big example for embedded system still missing

When ? Why ?

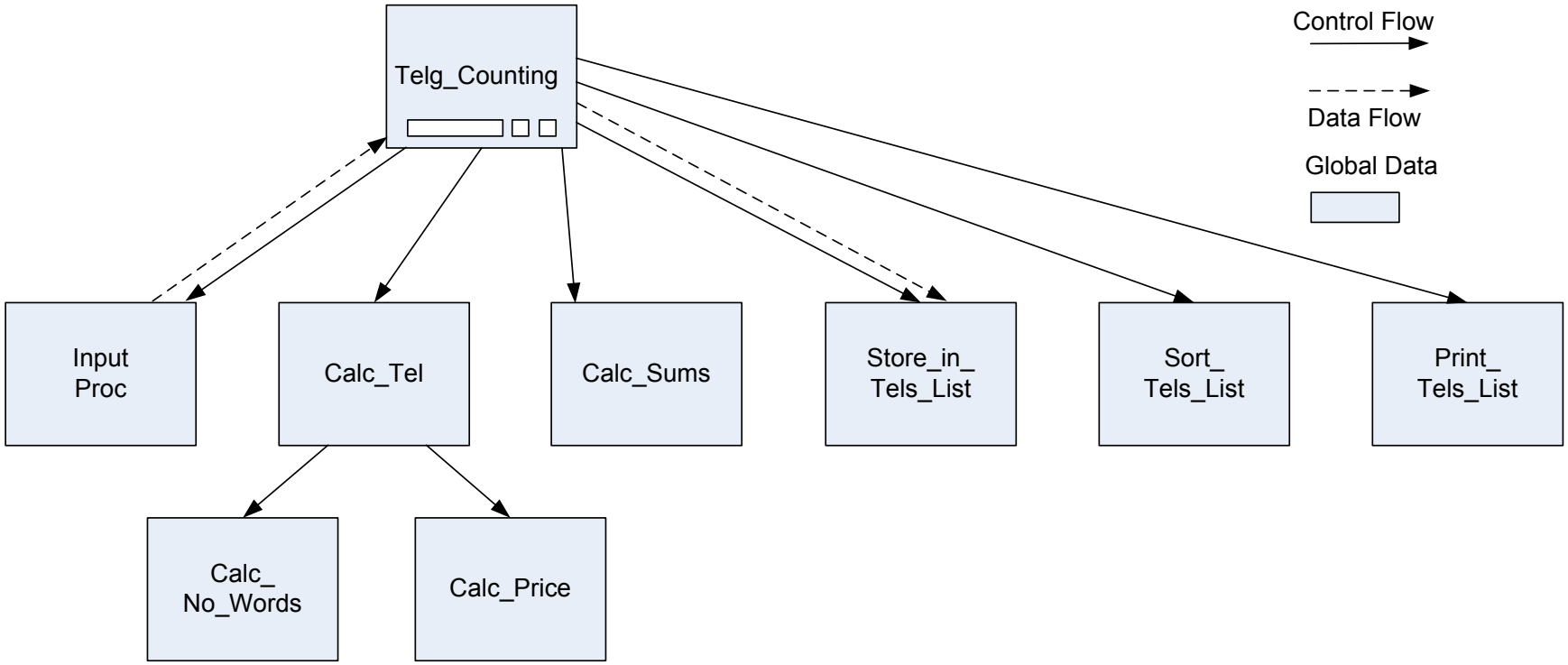
- When:
  - » after requirements specification: possible extensions / restrictions of outside functionality
  - » after architectural design: possible realization changes

  ⇨ now get answers to both questions

- Why:
  - » have probable extensions / modifications in mind
  - » develop the architecture for variations of system

- Importance of change discussion:
  - » get all locations of the system where data abstraction has to be applied

Repeat: Old Telegram Solution



What Was Wrong ? (Repetition)

- Structure of “Solution”
  - » Only functional components → subfunction relation
  - » Architecture is tree
    - Top-down development
    - Missing components
    - Thinking only in functional abstraction
  - » Tree is partially ordered
  - » Required tasks match architecture 1:1
  - » Global data, data flows
- Mistakes
  - » Commonalities not recognized, e.g. telegram syntax analysis
  - » Components of very different complexity
    - Calc_Sums is a two lines program
    - Input_Processing more complex, includes syntax analysis
  - » No components for data
  - » Change of requirements → change of architecture

Discussion of Telegram Example: Changes

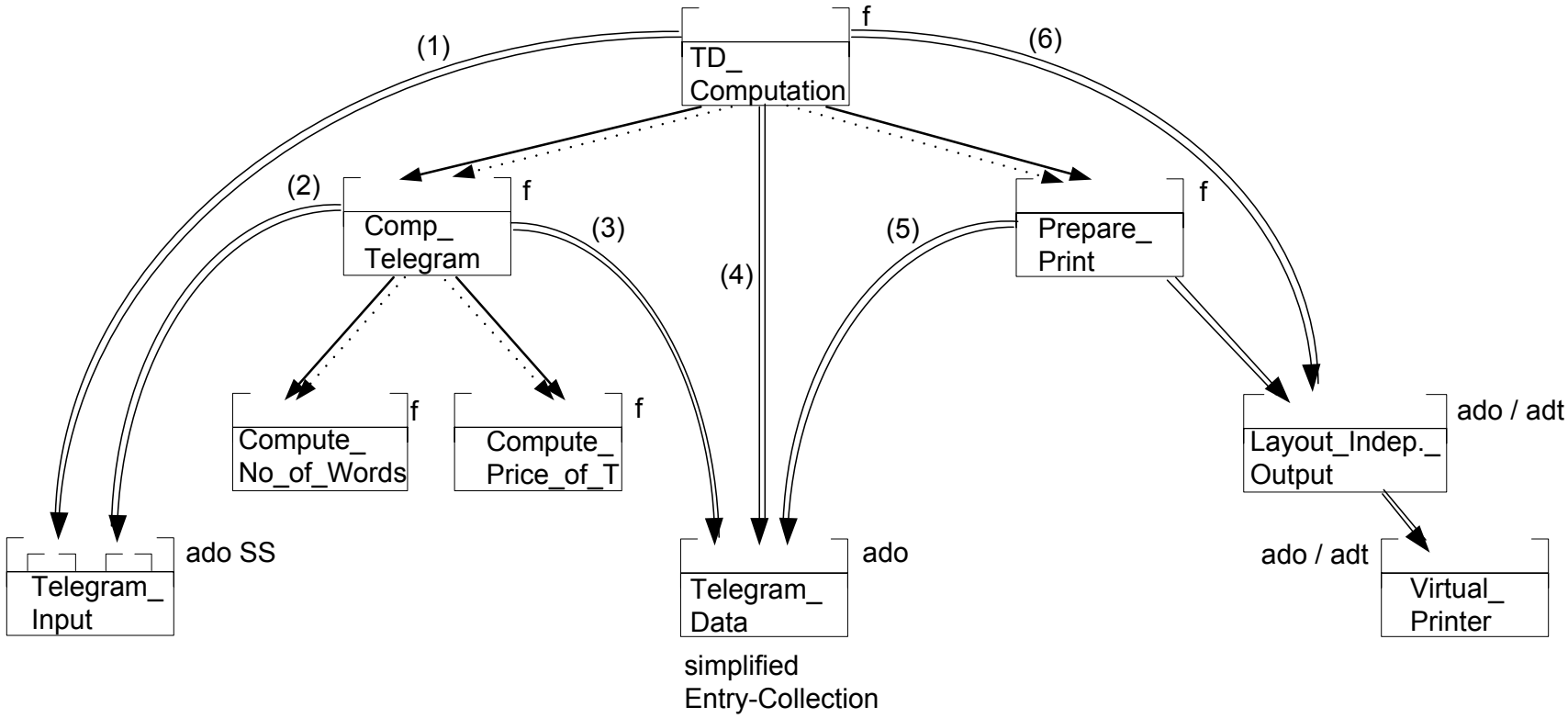
- Input
- Processing / Computation
- Storing Intermediate Data
- Output
- Global Changes

} Now we discuss !

Categorization: Functionality
Realization
} changes

Same discussion for the other examples: Card Boxes,…

The New Telegram Solution

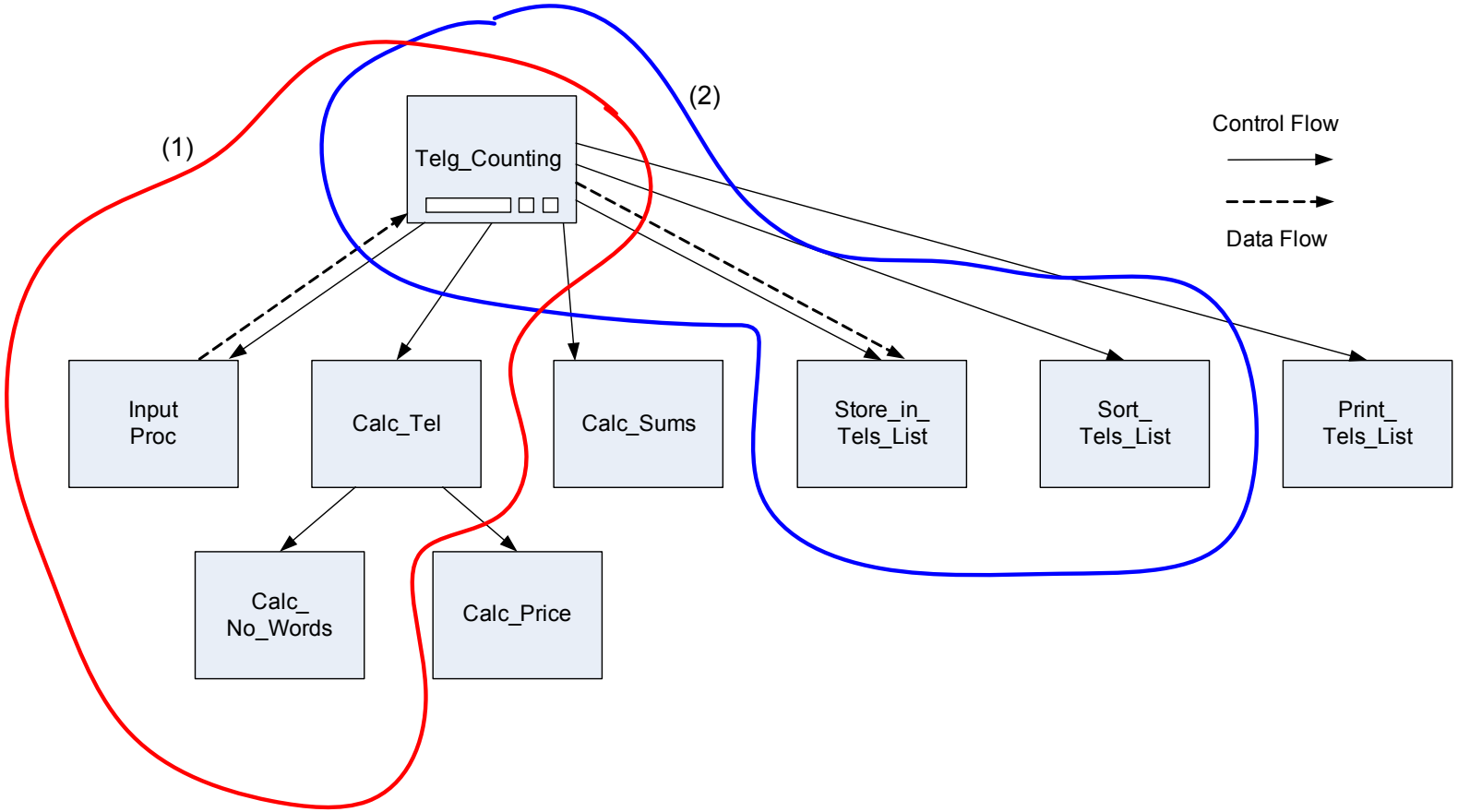


(1) Open / Close / File empty
(2) Read Telegram Data Components
(3) Store Triple
(4) Open / Close / Sort
(5) Reset / Read Current Triple
(6) Open / Close

Discuss:

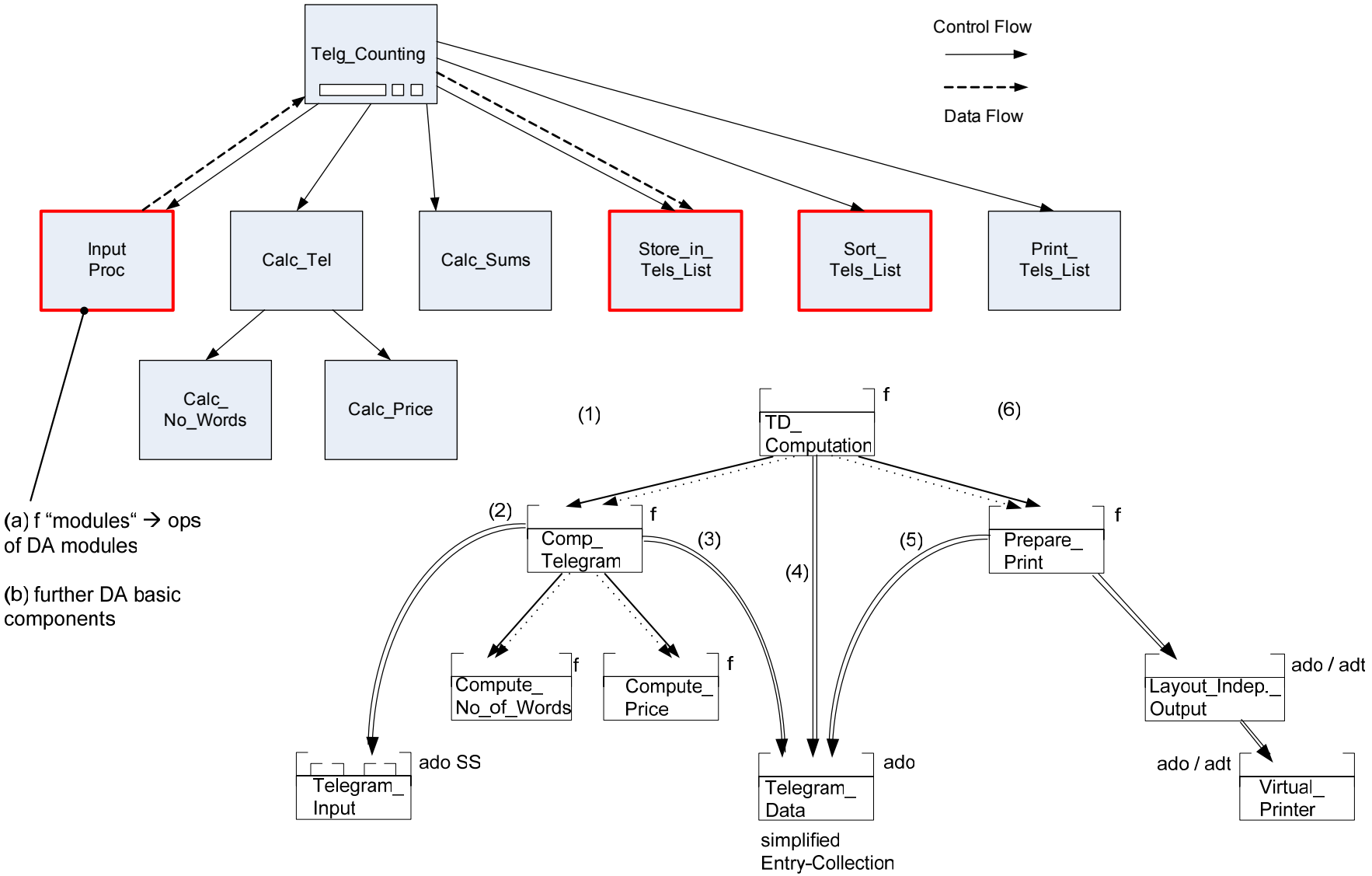
- Realization Change
- Functionality Change
- Estimate Changes

Big Changes if Data Abstraction Is Ignored



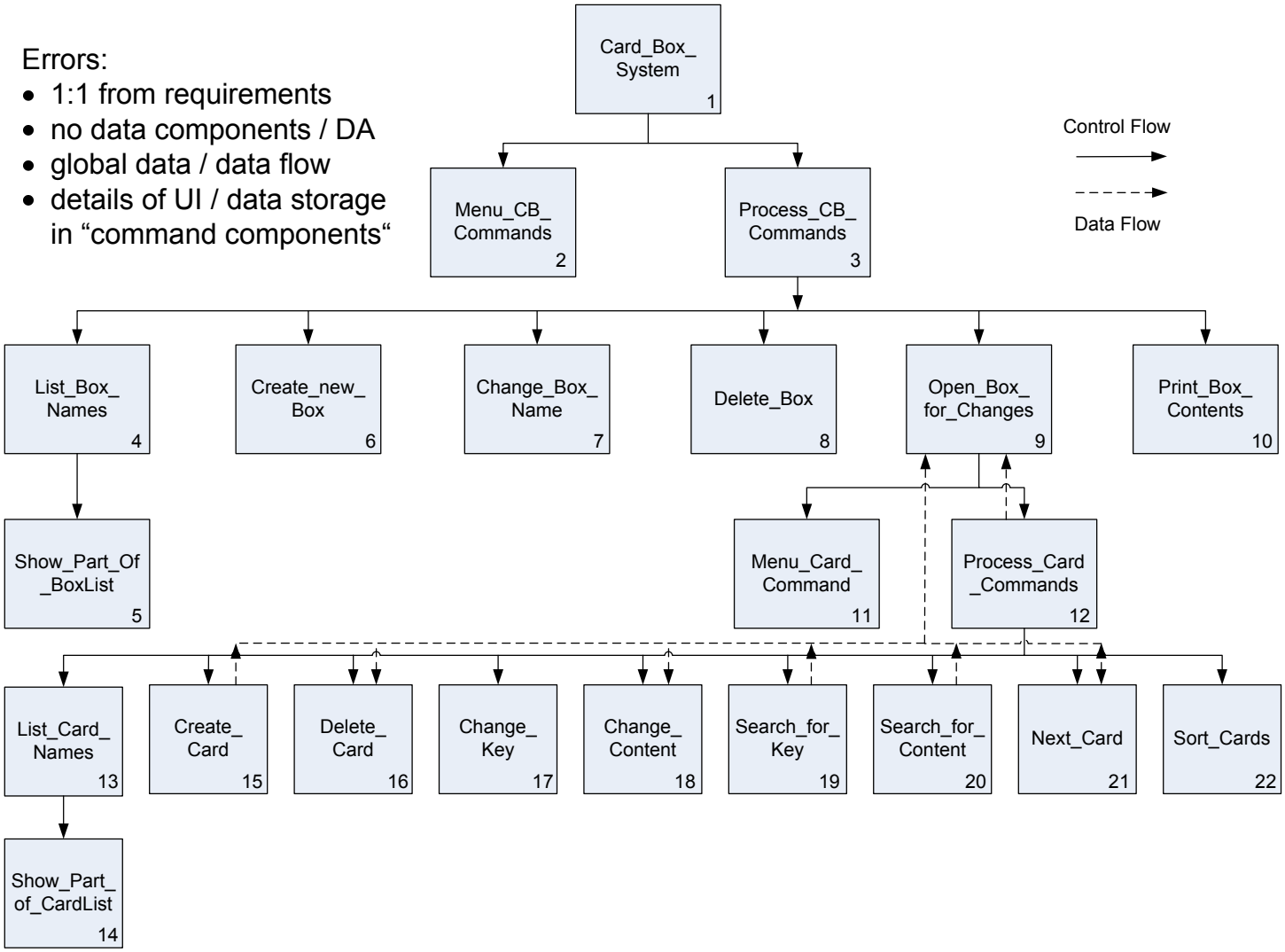
Especially, if system is restructured such that data abstraction is obeyed
Example: (1) telegram data are structures differently (but same info)
(2) other structure / realization of triple list

Comparison Old – New Telegram Example Architecture



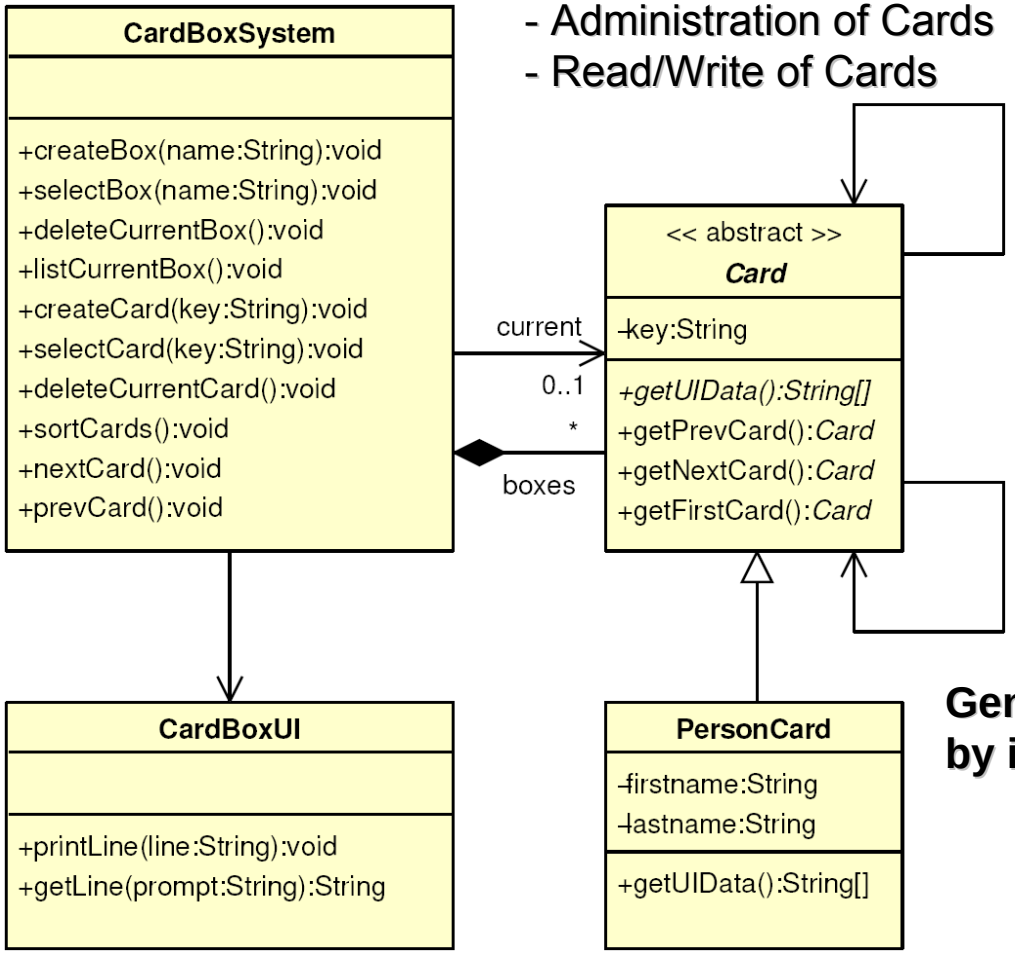
Repeat: Old Card Box Solution

Errors:
- 1:1 from requirements
- no data components / DA
- global data / data flow
- details of UI / data storage in “command components“



Repeat: Old Card-Box OO Solution

Multiple design decisions in one module
- Administration of Boxes
- Administration of Cards
- Read/Write of Cards



Multiple design decisions
- Entry
- Collection
- Integration
- Representation/Transformation
- Sorting

Genericity realized by inheritance

Summary: What Was Wrong ?

- Example System:
  - » Small Batch System: functional decomposition
  - » Small Interactive Problem: functional decomposition, OO

- Characterization:
  - » Functional decomposition
    - Tree structure, partially ordered
    - Global data flow

    } 1:1 mapping from functional requirements
  - » OO structure
    - One inheritance structure, again a tree

    } 1:1 mapping from application entities

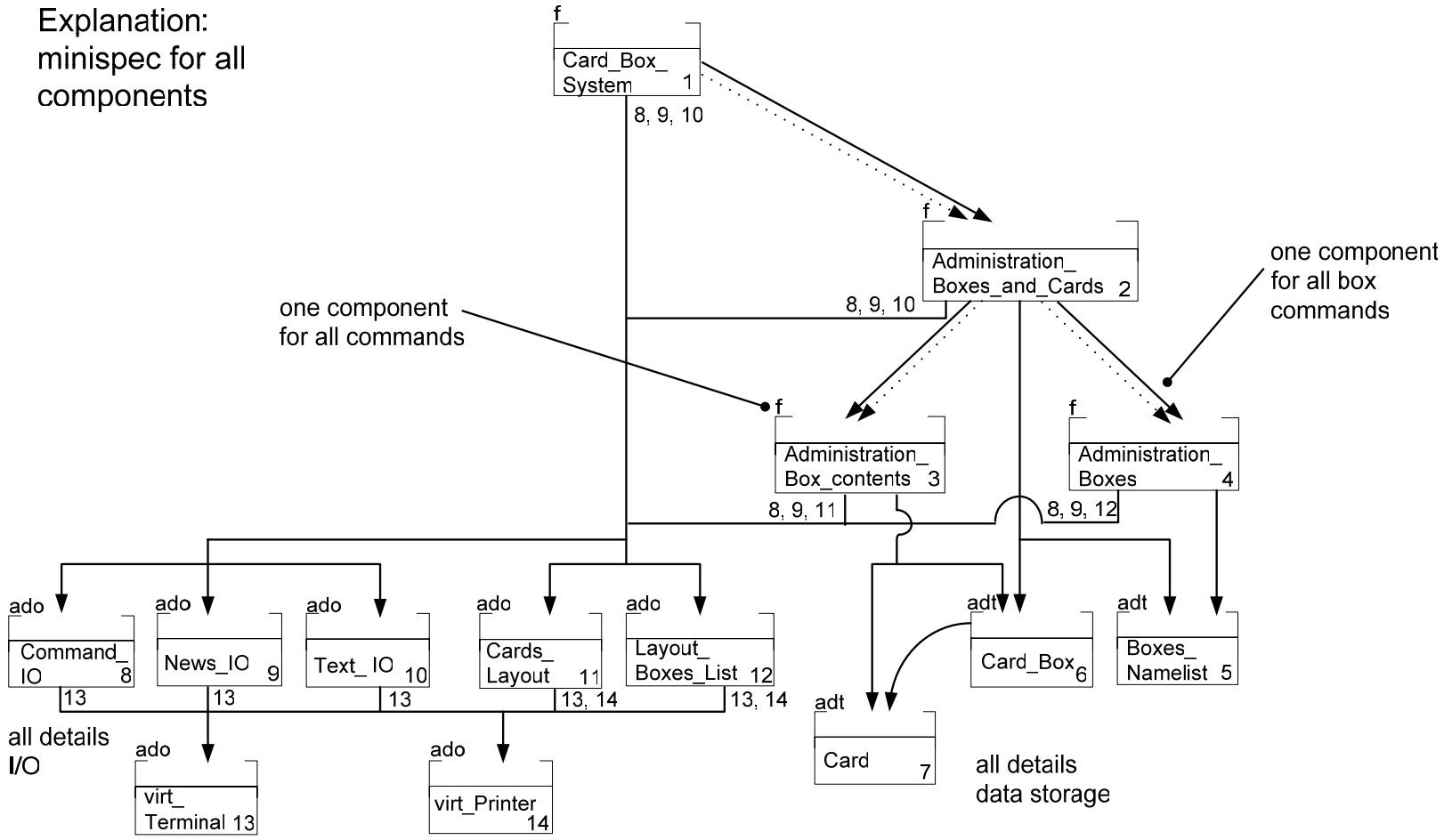
- Errors:
  - » Functional decomposition:
    - Data components missing
    - Data changes cause severe system changes
    - Requirements changes also cause severe changes
  - » OO decomposition:
    - Components with multiple decisions
    - Layering: classification, not use
    - Inheritance misunderstood
    - Requirements changes cause severe system changes

  } these design errors occur in practice

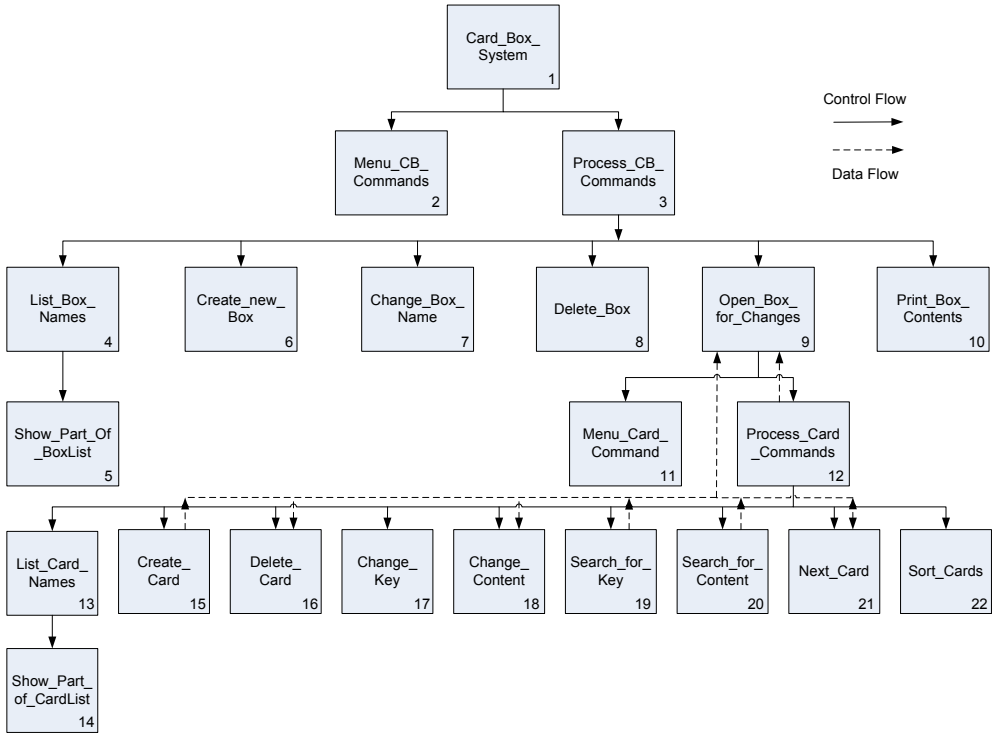
New Card-Box Solution

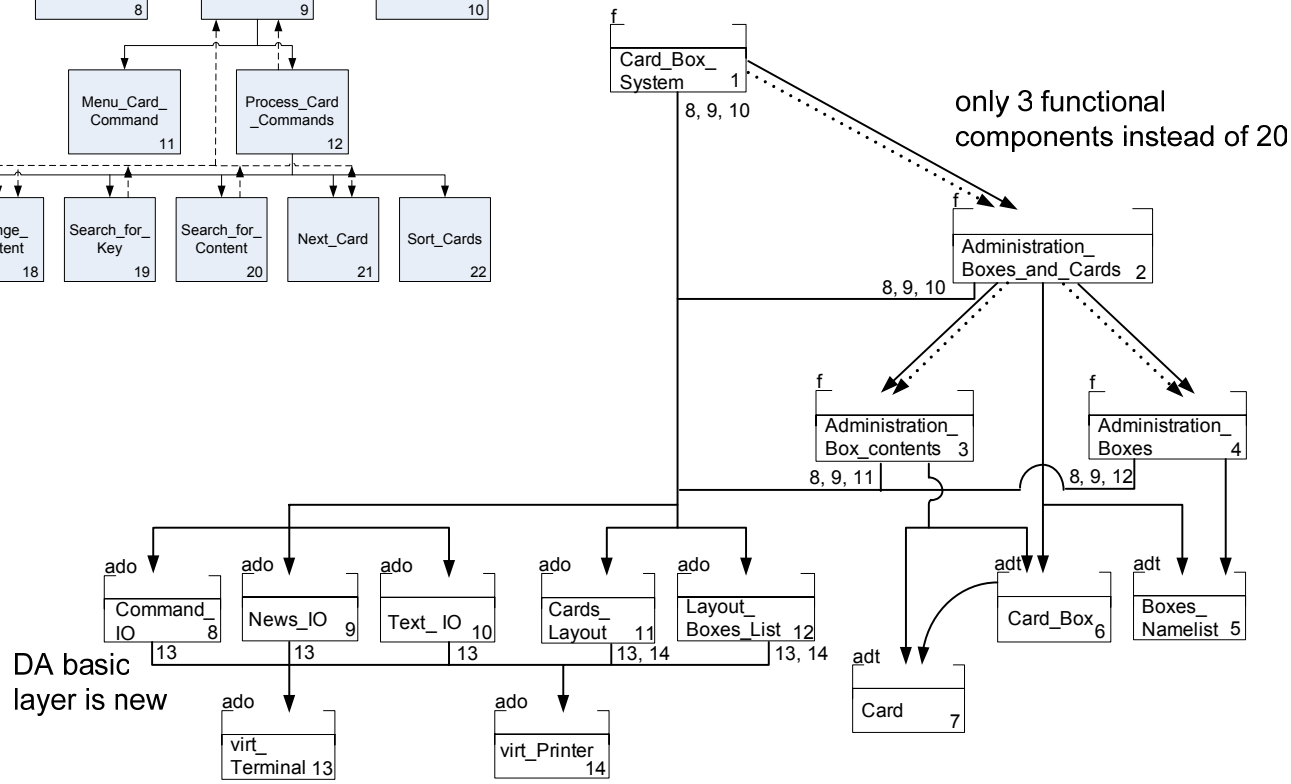
Explanation:
design decisions

Explanation:
minispec for all components



Comparison Old – New Solution

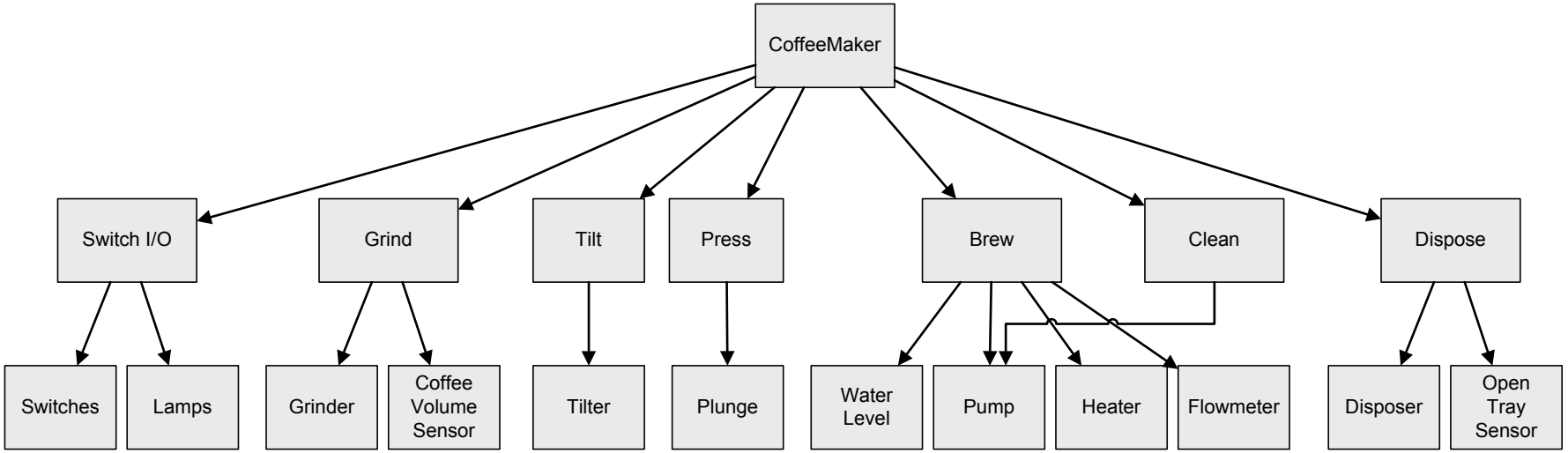




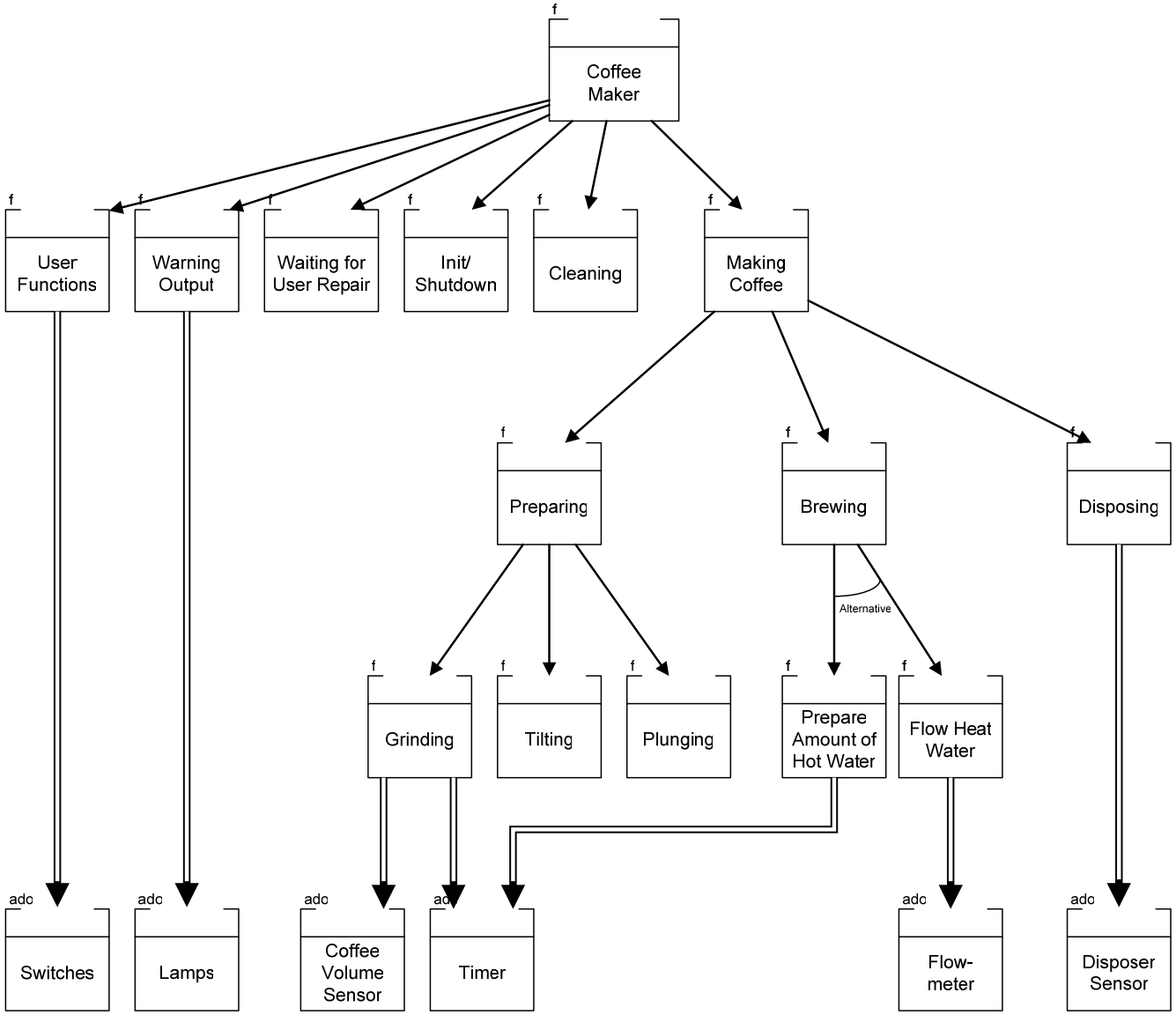
see book

Coffee Maker – Revisited

- realization functionality and not abstract functionality
- direct usage of underlying sensors, actuators and components (e.g. switches, pump, water level sensor, etc.)
- mixed up standard and special functions (e.g. startup/shutdown, error and exception handling)



Coffee Maker – How it should look

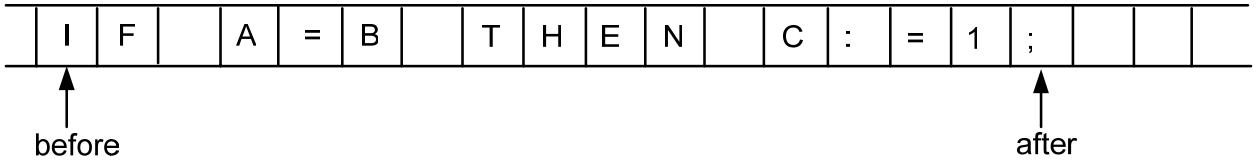


Batch Systems

- Structure class examples
  - » business administration: payroll accounting, …
  - » math. techn. application: experiment evaluation,
  - » FEM computation, …
  - » systems software: compiler, …
- Regard compiler example (importance for informatics education):
  - » well-known
  - » structure can be ”derived”
  - » elaborated reuse approaches
- Two approaches
  - » Multi-phase compiler (see Aho, Ullman, Sethi): scanner and parser mostly generated
  - » One-phase compiler (see Wirth): ”mechanical” process for realizing a recursive descent compiler

Recursive Descent Compiler

- Idea
  - » nonterminal symbol → procedure for translation
  - » before and after situation:



  - » all translation tasks in one procedure:
    - lex analysis, context-free analysis, context-sensitive analysis, addressing, code generation
  - » also compilation error handling:
    - localization, detection, messaging, evtl. repair, recovery
  - » nonterminals, recursive mutual relationships:
    - recursively calling procedures
    - top-down translation
    - LL(1) grammar necessary

Rules and Procedure Bodies

grammar for simple language

```
statement ::= [ ident := expression |
                call ident |
                begin statement {; statement} end |
                if condition then statement |
                while condition do statement ]
procedure statement (…) is
    … -- necessary declarations, see below
begin
    if sym = ident then … -- translation assignment
    elsif sym = callsym then …  -- translation procedure call
    elsif sym = ifsym then …  -- translation conditional statement
    elsif sym = beginsym then …   -- translation block
    elsif sym = whilesym then …  -- translation while loop
    end if;
    …
end statement;
```

lex. analysis is activated during parsing

this part is a 1:1 mapping of rule: context-free analysis

Wirth:
body can be “derived“ from rule´s r.h.s

extended grammar to allow simple errors

error handling

context-sensitive analysis

addressing

code generation

Compiler Structure

Nonterminal symbol → Procedure (simple functional module)

EBNF → Dependency Graph → Architecture Diagram

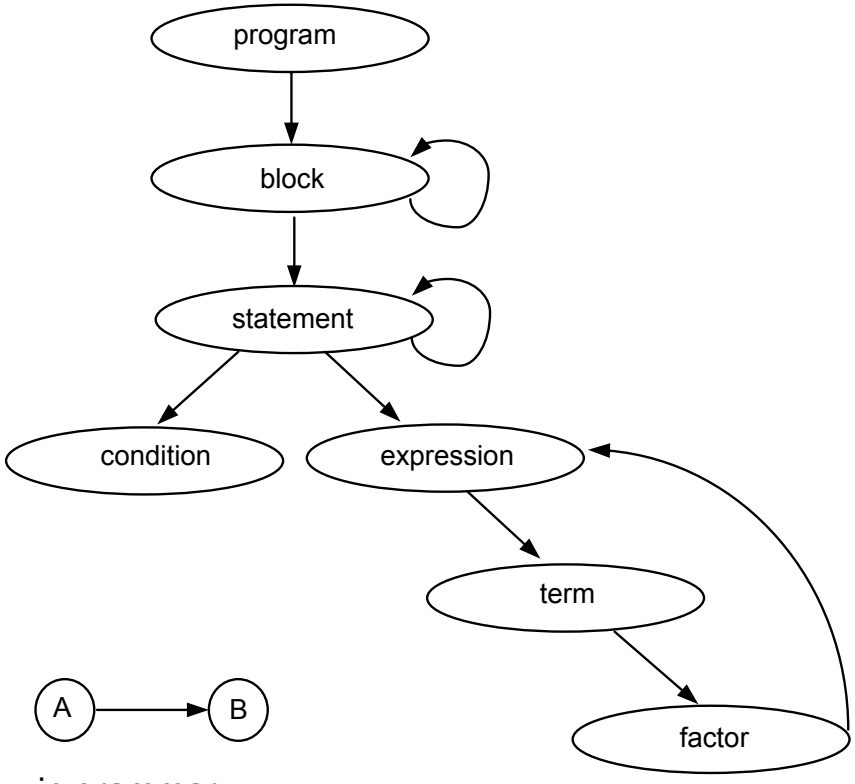
Spanning Tree of Diagram: contains relation

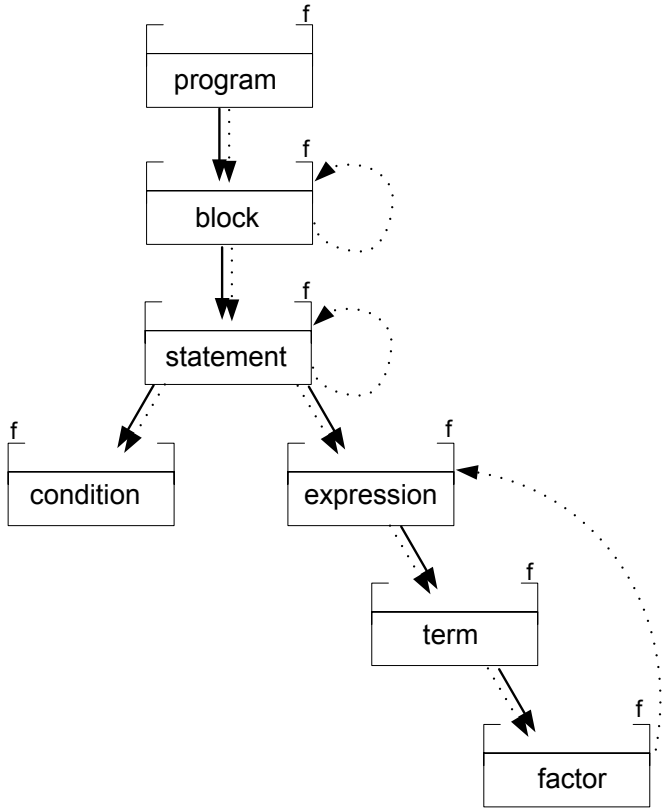
Further relations: local usability

Simple Example Programming Language

```
program     ::= block .
block       ::= [ const ident = number {, ident = number} ; ]
                [ var ident {, ident} ; ]
                { procedure ident ; block ; } statement
                {statement}
statement   ::= [ ident := expression |
                  call ident |
                  begin statement {; statement} end |
                  if condition then statement |
                  while condition do statement ]
condition   ::= odd expression |
                expression relop expression
expression  ::= [ addop ] term { addop term }
term        ::= factor { multop factor }
factor      ::= ident | number | (expression)
                  …
```

Dependency Graph and Architecture





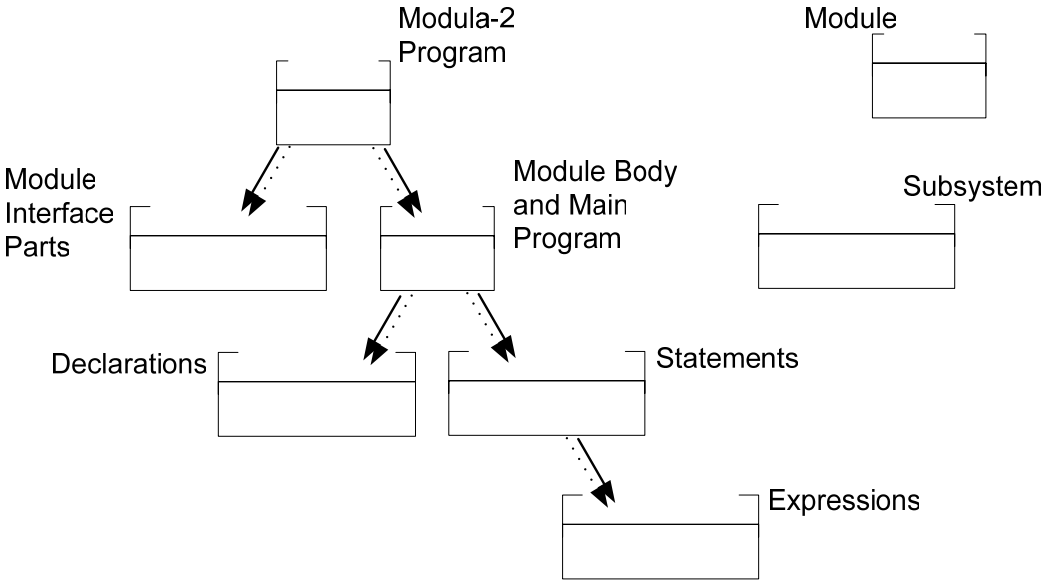
in grammar:
B appears on the right side of A rule

in procedure:
B is needed for the translation of A and, therefore, called

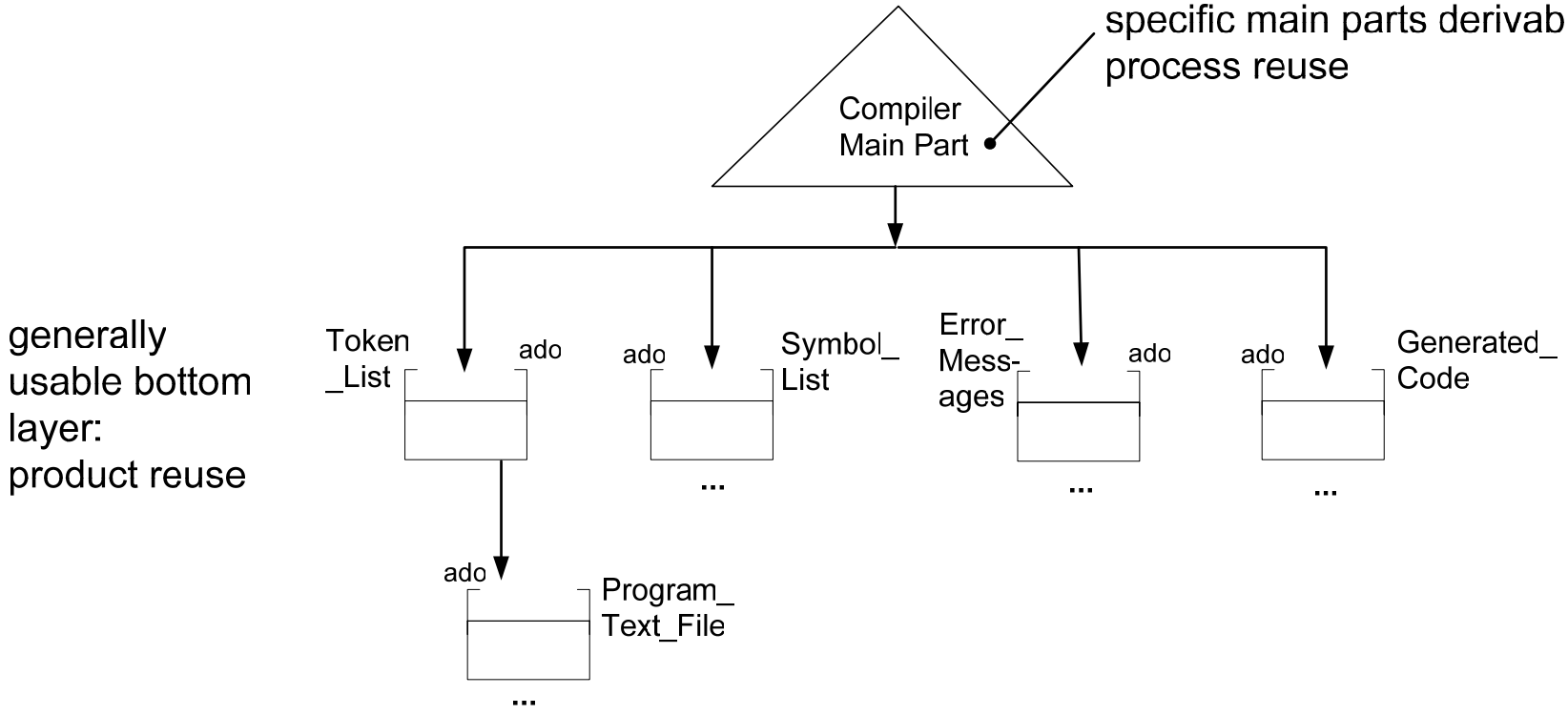
Compressed Architecture

- Pascal, Modula-2
  - » 100 nodes, 10 layers
  - » 1:1 mapping of grammar
- Compression:
  - » look for “complete” contains trees
  - » aggregated interfaces: e.g. different forms of statements
  - » are the main concepts of a programming language

compressed presentation
for Modula-2, Pascal



Generally Usable Components

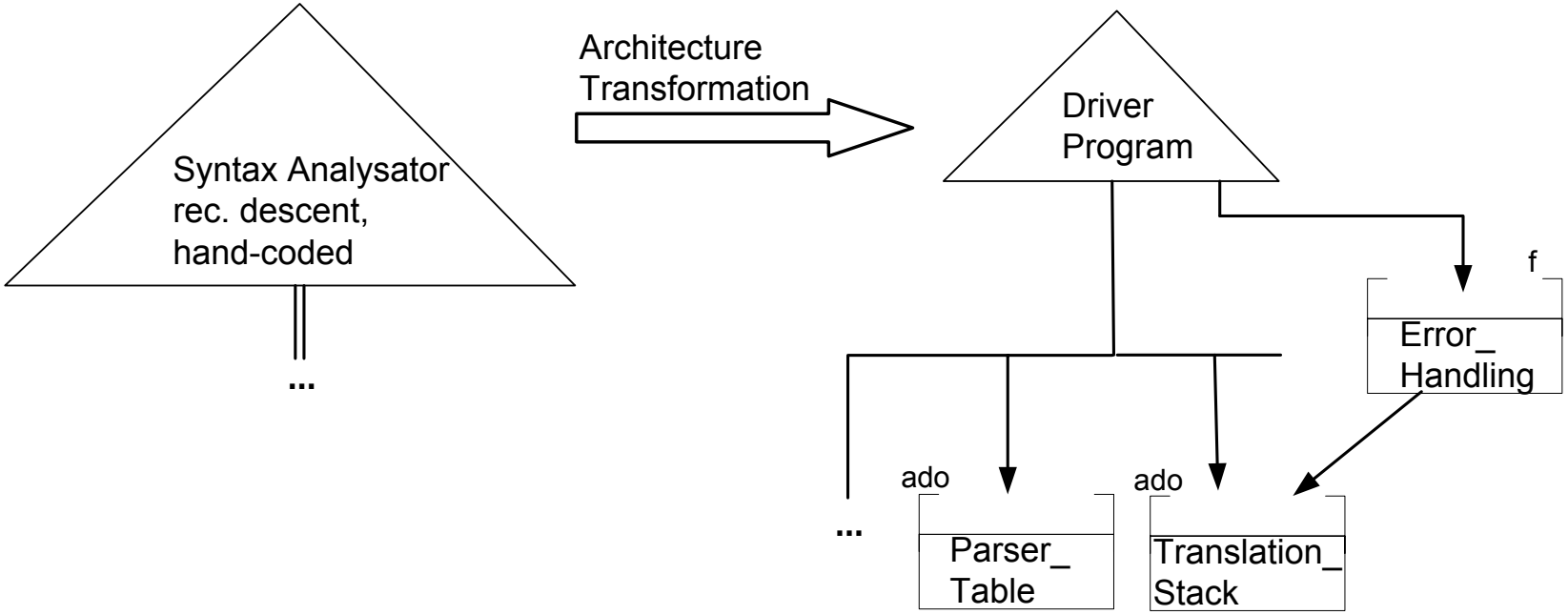


in recursive programs usually on global data

(see motivation for general usability)

Table – driven Compiler

- Restrict to syntax analysis
- “Same” architecture as above
- Now code → tables
- Table exchangeable
  LL(1) table
- Driver program for all LL(1) tables
- Need compiler stack
- Explicit error handling



see book

Complete Architectures

- Structure classes (small example, big example):
  - batch: telegram example, compiler
  - interactive: card-box system, software development environment
  - embedded: coffee machine control, big example still missing
- Adaptability of object –based systems
  - Local changes
  - If nonlocal, can trace changes
  - Measure for adaptability = number of data abstraction modules ?
- Importance of “Change” discussion
  - Design not one system but also “next” variations
  - Get all locations, where data abstraction has to be used