Hierarchical CDFDs and Modules

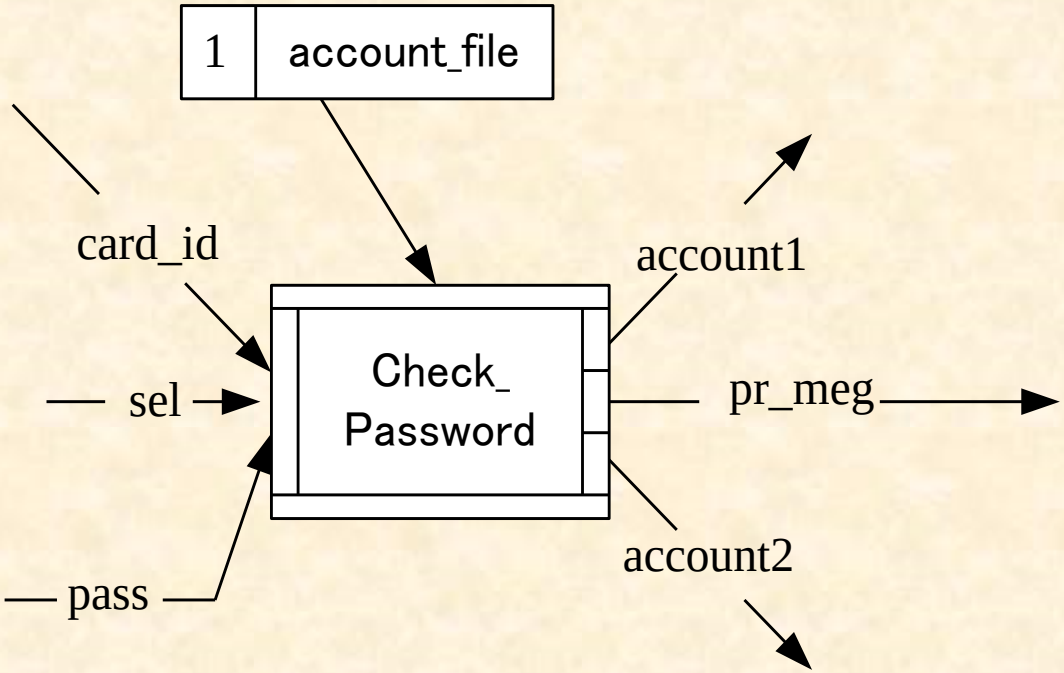
The motivation for building hierarchical CDFDs:

- It is almost impossible to construct only one level CDFD and module for a complex system.
- It is necessary to organize the developers within a team so that each of them can concentrate on a single part of the entire system independently and all the developers can conduct their activities concurrently.

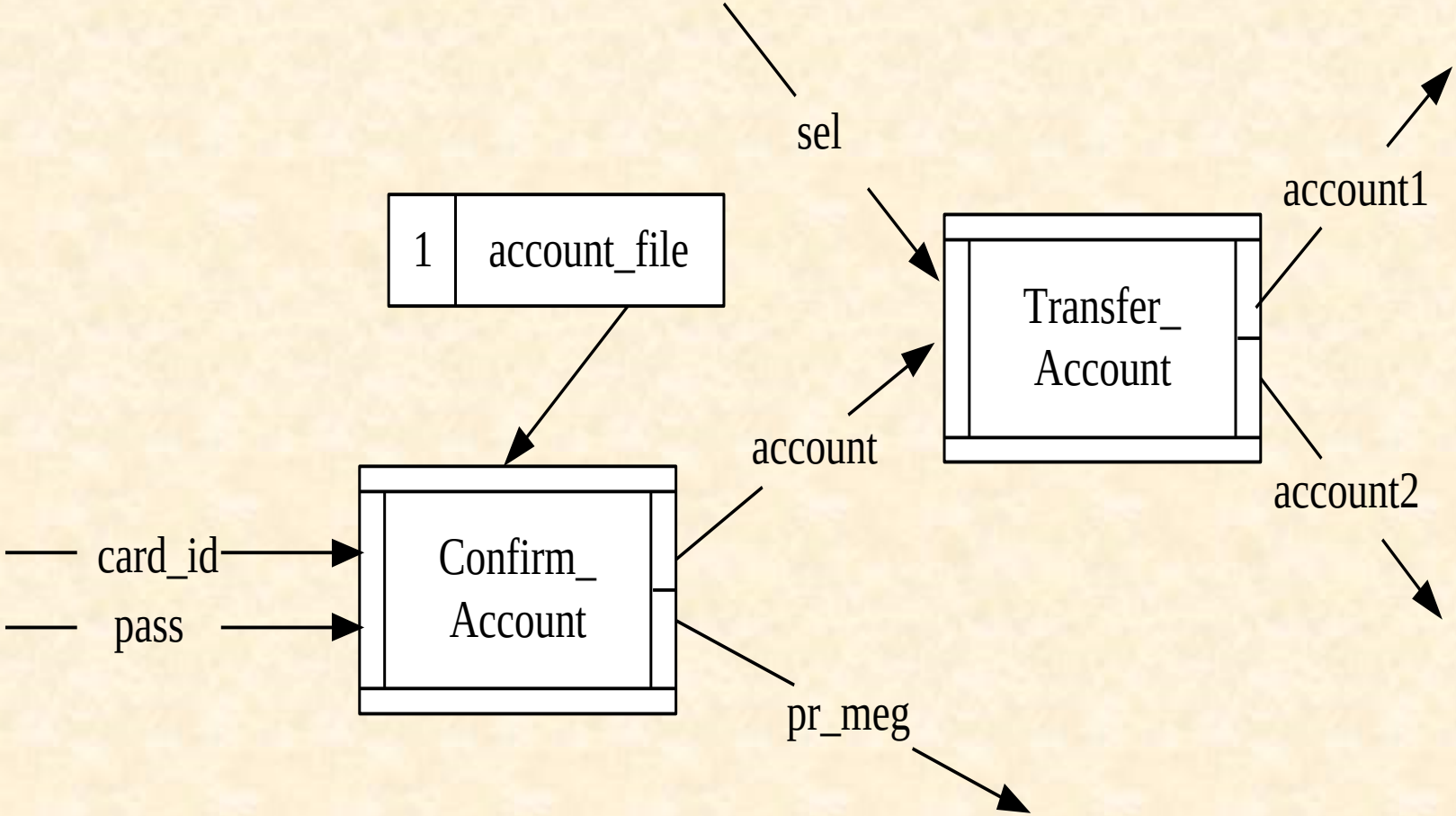
Process decomposition

Process decomposition is an activity to break up a process into a lower level CDFD.

Example:



This process can be decomposed into the CDFD on the next slide:

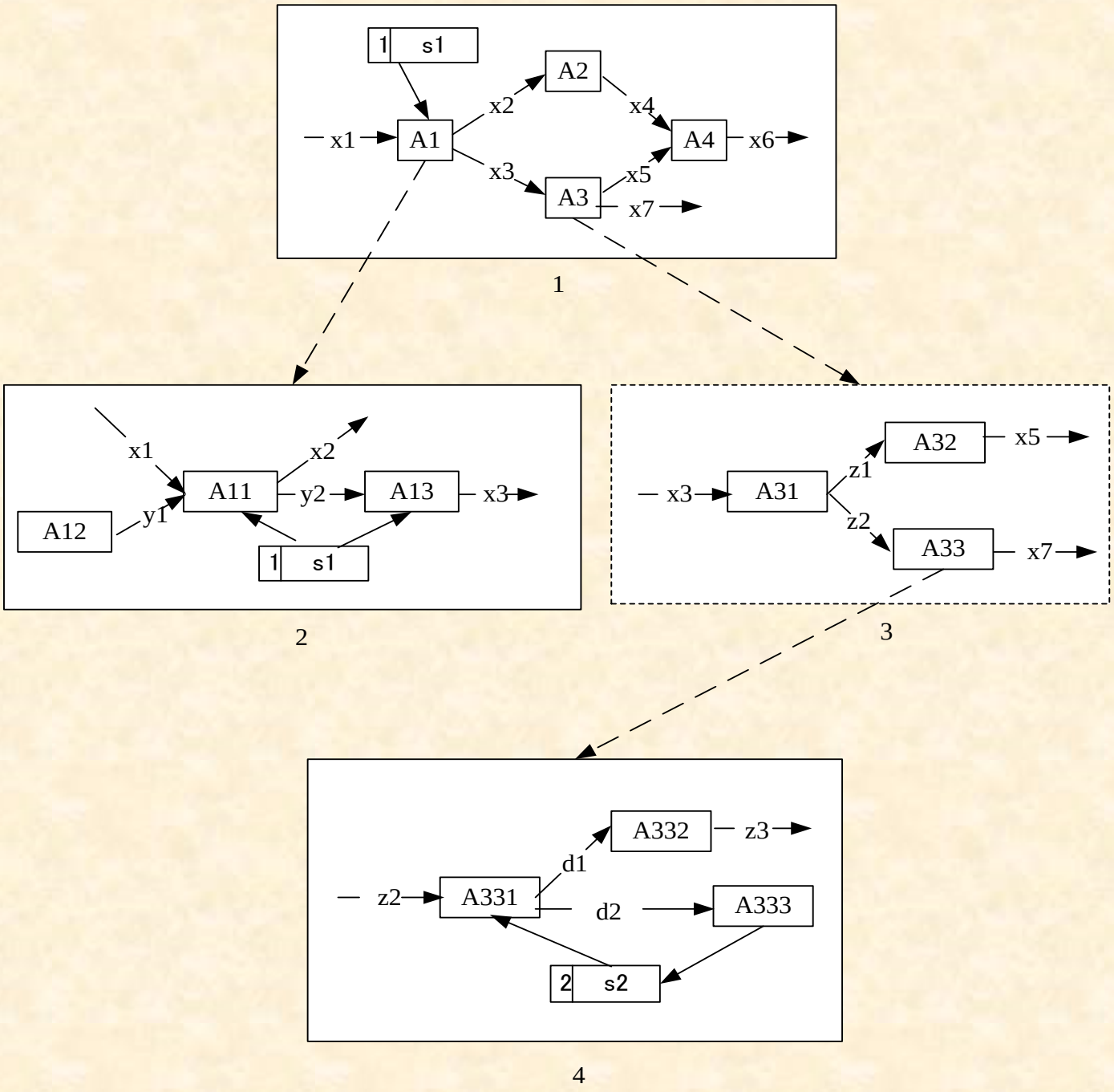

sel
account1
1
account_file
Transfer_
Account
account
account2
card_id
Confirm_
Account
pass
pr_meg

```
module Check_Password_decom / SYSTEM_ATM;
var    ext  account_file: set of Account;
behav CDFD_2; /* Assume the CDFD in Figure 2 is numbered
   2. */
process Init()
end_process;

process Confirm_Account(card_id: nat0, pass: nat0)
                    account: Account | pr_meg: string
 ...
end_process;
process Transfer_Account(sel: bool, account: Account)
                     account1: Account | account2: Account
 ...
end_process;
end_module;
```

Definition If process A is decomposed into a CDFD, we call the CDFD the decomposition of process A and process A the high level process of the CDFD.

Decompositions can be carried out until all the lowest level processes are simple enough to be formally defined. As a result, a hierarchical CDFDs will be constructed.


1
s1
A1
A2
A3
A4
x1
x2
x3
x4
x5
x6
x7
1
2
A11
A12
A13
x1
x2
x3
y1
y2
1
s1
2
A31
A32
A33
x3
z1
z2
x5
x7
3
A331
A332
A333
z2
d1
d2
z3
2
s2
4

The associated module hierarchy of this CDFD hierarchy is outlined as follows:

```
module SYSTEM_Example;
 ...
var    s1: Type1;
behav CDFD_1;
process Init;
process A1

decom: A1_decom; /*Module A1_decom is associated with the
                                  decomposition of A1. */
end_process;

process A2;

process A3
decom: A3_decom; /*Module A3_decom is associated with the
   decomposition                                  of A3. */
end_process;

process A4;
end_module;
```

```
module A1_decom;
 ...
 var  ext s1: Type1;
 behav CDFD_2;
 process Init;
 process A11;
 process A12;
 process A13;
 end_module;
 module A3_decom;
   ...
 behav CDFD_3;
 process Init;
 process A31;
 process A32;
 process A33
  decom: A33_decom; /*Module A33_decom is associated with the
                                  decomposition of A33. */
 end_process;
 end_module;
```

```
module A33_decom;
  ...
 var  s2: Type1;
 behav CDFD_4;
 process Init;
 process A331;
 process A332;
 process A333;
end_module.
```

Handling stores in decomposition

Generally speaking, a store accessed by a high level process must also be drawn in the decomposition of the process, and must be accessed in the same way, probably by several processes.

Definition An external store of a CDFD is a store that is accessed by a high level process of the entire specification.

There are two kinds of external stores:

(1) a store local to a high level CDFD. Usually it is declared after the keyword ext.

(2) existing external store that is global to all the CDFD in the hierarchy. Usually such stores are declared with the sharp mark, such as

ext #file: int

Definition A local store of a CDFD is a store that is introduced for the first time in the CDFD.

Scope

When defining a module, we may need to refer to some type or function or any components defined in another module. In this case, we need a scope rule to constrain the reference of those components.

To give a formal definition of the scope rule, we need the following definitions.

Definition Let process A be defined in module M1. If A is decomposed into a CDFD associated with module M2, we say that process A is decomposed into module M2.

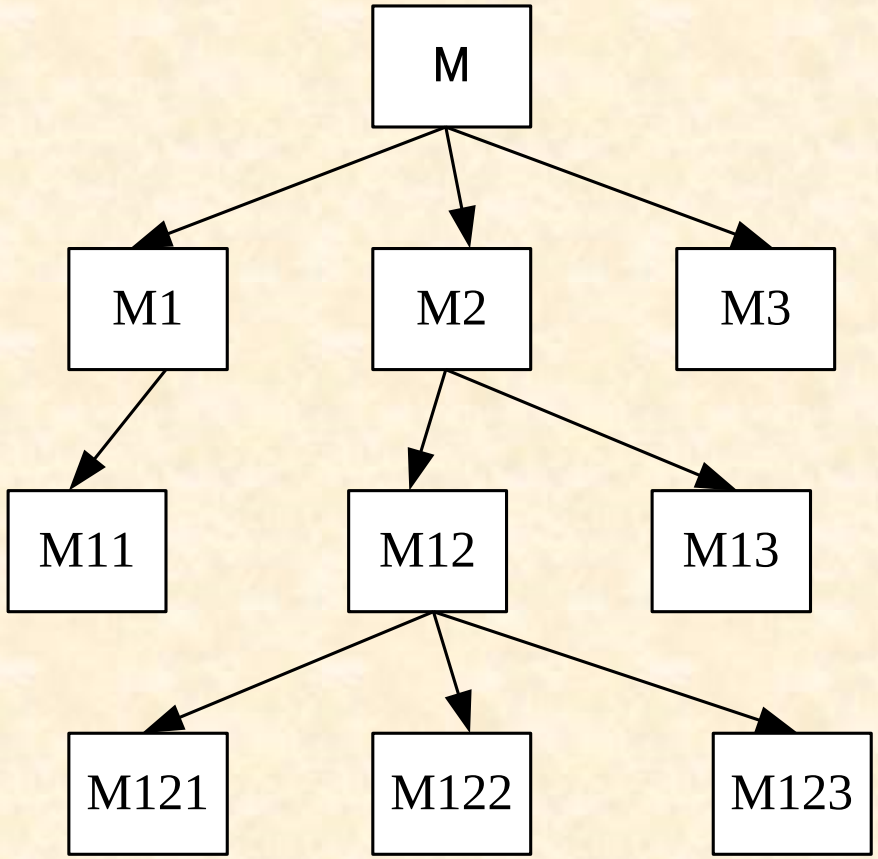
Definition If process A defined in module M1 is decomposed into module M2, M2 is called child module of M1, while M1 is called parent module of M2.

Definition Let A_1, A_2, ..., A_n be a sequence of modules where n >1. If A_1 is the parent module of A2, and A2 is the parent module of A_3, ..., and A_n - 1 is the parent module of A_n, then we call A1 ancestor module of A_n and A_n descendant module of A_1.

Definition If module A is neither ancestor module nor descendant module of module B, A and B are called relative modules.

Example:

(1) M2 is the parent module of M12.
(2) M2 is an ancestor module of M122.
(3) M11 and M13 are relative modules. M13 and M121 are relative modules.



Scope rules:

- Let M_c be a type or constant identifier declared in module M1. Then, the scope of the effectiveness of this declaration is M1 and its all descendant modules.
- Let M_c be declared in both module M1 and its ancestor module M. Then, M_c declared in M1 has higher priority in its scope than the same M_c declared in M.
- Let M_f be a function defined in M1. Then, the scope of M_f is M1 and its all descendant.
- Let M_c be a type, constant identifier, or function declared in module M1. Then, if M_c is used in its relative module M2, it must be used in the form:

M1.M_c

Approaches to constructing specifications

The outline of this part:

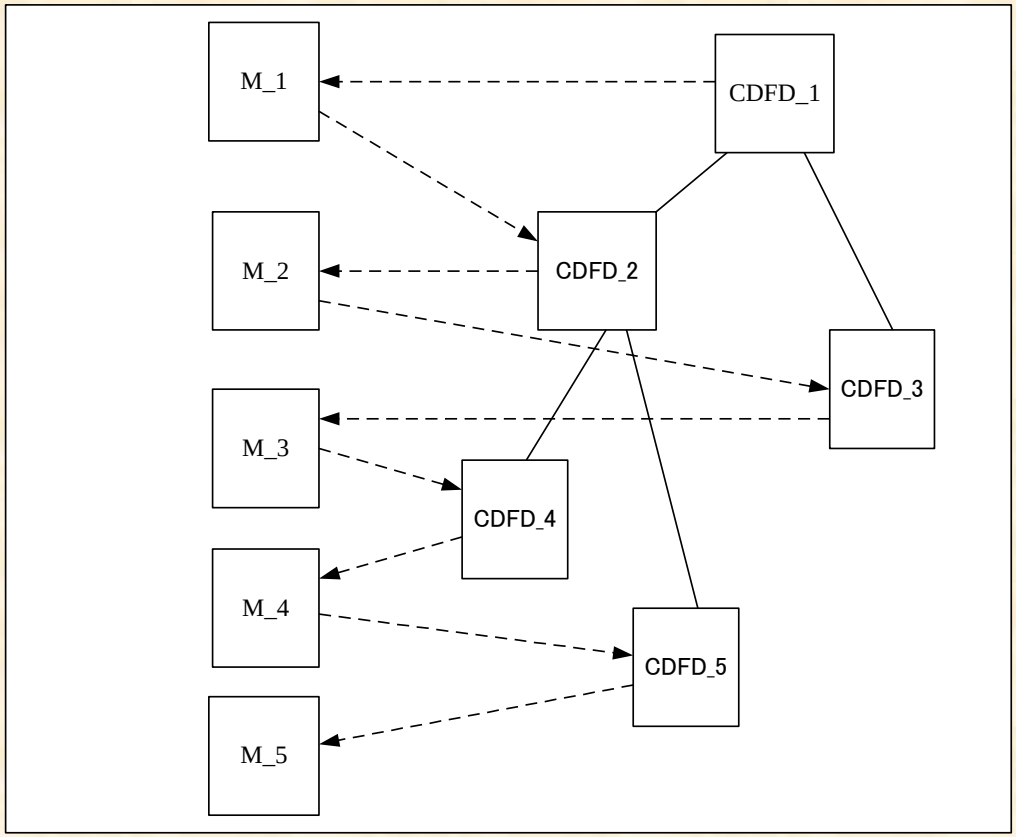
- The top-down approach
- The bottom-up approach
- The middle-out approach
- Comparison of the approaches

The top-down approach

Two strategies can be taken in the top-down approach: the CDFD-module-first strategy and the CDFD-hierarchy-first strategy

(1) The CDFD-module-first strategy

The fundamental idea of this strategy is that after a CDFD is constructed, its associated module must be defined precisely, before any decomposition of processes in this CDFD takes place. After both the CDFD and module are finalized, another decomposition can take place. Such a process goes on until no process needs further decomposition.

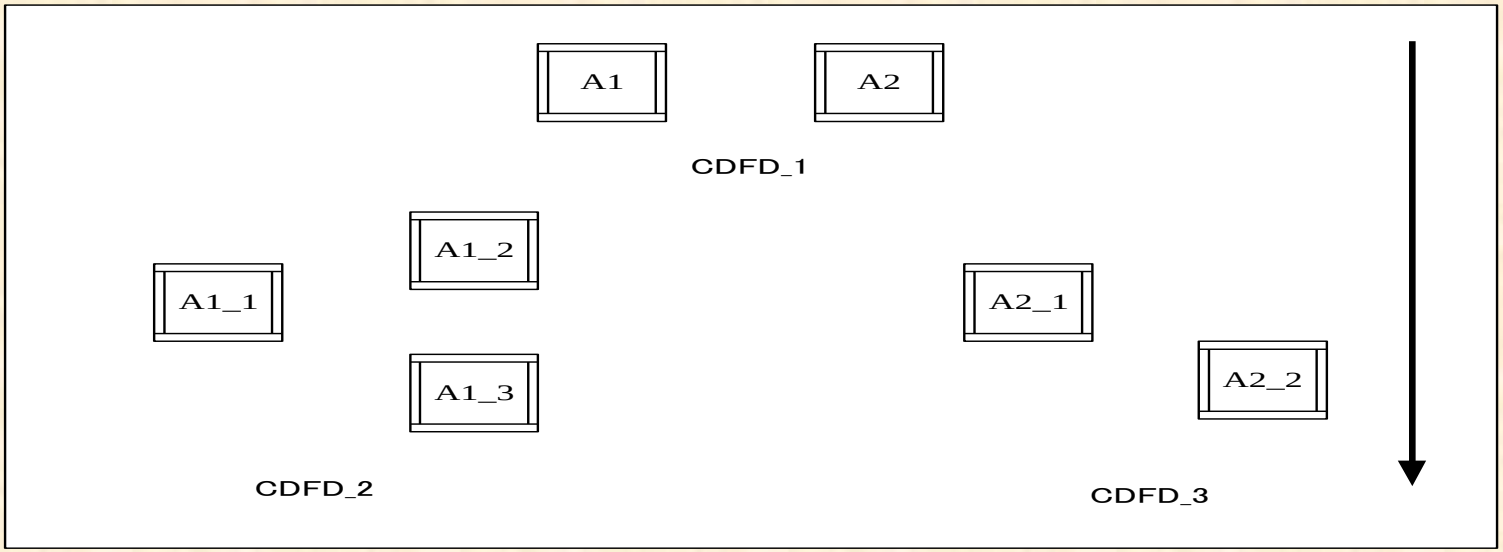


The following guidelines may be useful in determining whether a process needs a decomposition:

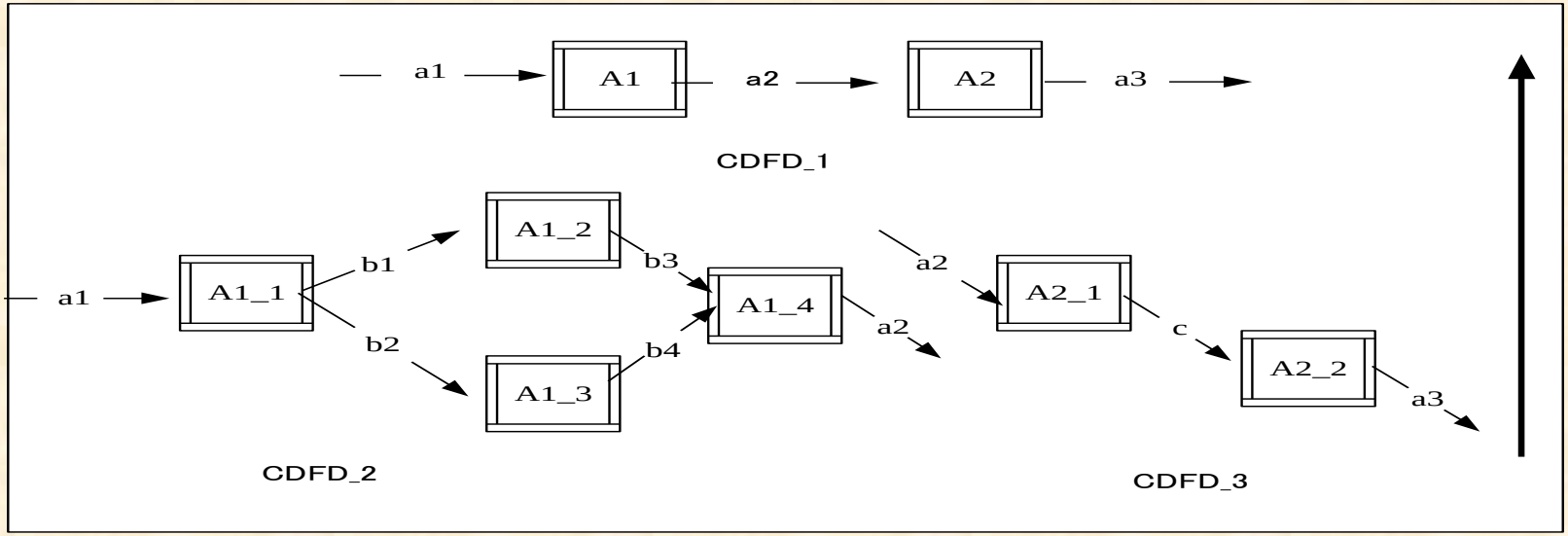
1. If the relation between the input and output data flows of a process cannot be expressed without further information, the decomposition of this process should be considered.
2. If the behavior of a process involves a sequence of actions, this process needs to be decomposed.
3. If the postcondition of a process is too complex to be written in a concise manner, it may need decomposition.

(2) The CDFD-hierarchy-first strategy

Building a specification using the CDFD-hierarchy-first strategy starts with the construction of the CDFD hierarchy by decomposition of processes, and then proceeds to define the modules of the CDFDs involved in the CDFD hierarchy, after it is completed.



(a) Top-dwon for processes

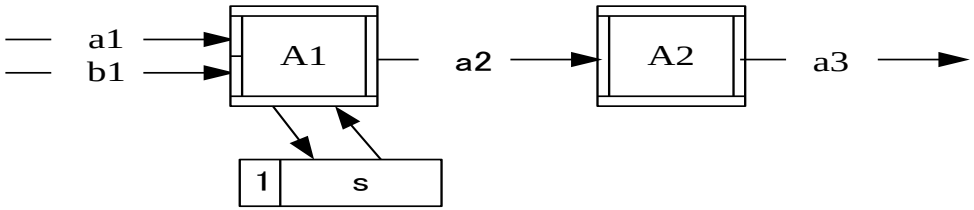


(b) Bottom-up for data flows and stores

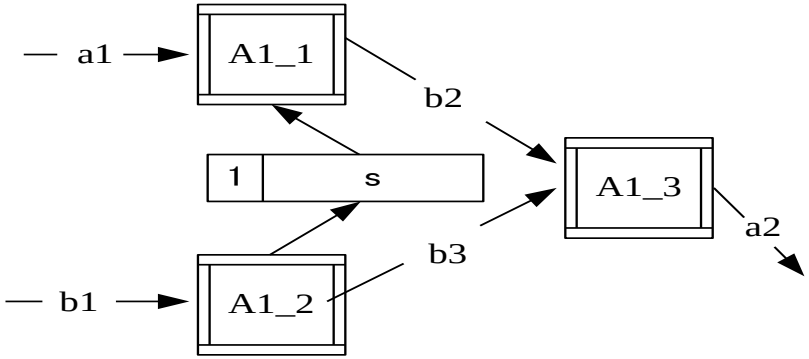
The middle-out approach

Constructing a specification by the middle-out approach usually starts with the building of the CDFDs and modules modeling the functions that are most familiar to the developer and crucial to the system. Then, the system is built by

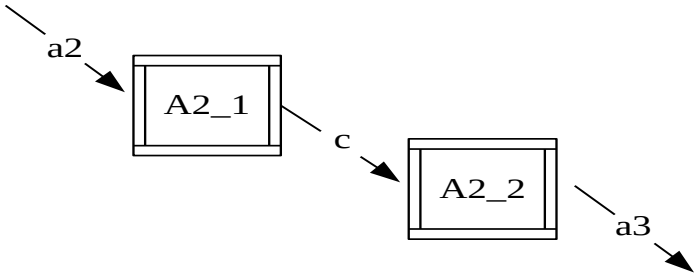
- decomposing some of the introduced processes
- synthesizing some of those processes to form high level processes.



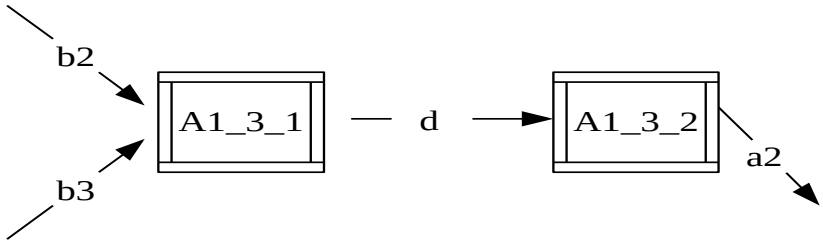
CDFD_1



CDFD_2



CDFD_3



CDFD_4

When synthesizing CDFDs into high level processes, we can follow the following guidelines:

- If there are more than two input data flows to different starting processes of a CDFD, the CDFD needs to be abstracted into a high level process that defines precisely the relationship among those input data flows.
- If two processes in a CDFD access the same store in both reading and writing manner, this CDFD needs to be considered for abstraction to define that concurrent execution of the two processes is impossible (e.g., processes A1_1 and A1_2 in CDFD_2).
- If two CDFDs have relations in terms of data flows, they need to be abstracted into high level processes and the connections between these processes need to be formed in the high level CDFD (e.g., process A1_3 in CDFD_2 and process A2_1 in CDFD_3 share the same data flow a2).

Comparison of the approaches

1. The advantages and weakness of the top-down approach:

(1) Advantages:

- It is effective and intuitive in providing sub-goals or sub-tasks to support the current goal or task, and in developing ideas with little information (abstraction) into ideas with more information (decomposition).
- It provides a good global view of data flows and stores that may be used across CDFDs at different levels, thus the consistency in using data flows and stores can be well managed during the decomposition of high level processes.

(2) Weakness:

It may cause frequent modifications of high level processes, data flows, stores, and even the entire CDFDs, as with the progress of decomposition of high level processes, due to the lack of sufficient knowledge about what data flows and stores will be used or produced by the processes in the lower level CDFDs.

2. The advantages and weakness of the middle-out approach

(1) Advantages:

- It may be more effective and natural than the top-down approach, because it always starts with modeling the most familiar and crucial functions.
- It also takes a flexible way to utilize the top-down and the bottom-up approaches. Taking which approach usually stems from natural demands during the construction of the entire specification.

(2) Weakness:

The developer may not be easy to take a global view of the specification in the early stage, thus data flows, stores, and processes created in different CDFDs may overlap or defined inconsistently.

3. How to use the top-down and the middle-out approach?

- Use the middle-out approach for requirements analysis and requirements specification constructions, especially for the semi-formal ones, because the most familiar and important functional requirements are often focused in the early stage of requirements analysis.
- Use the top-down approach for design, because the designer usually has a fair understanding of the functional requirements after studying the semi-formal requirements specification, and needs to take a global view in structuring the entire system.

Class exercise 10

1. Answer the questions:

a. what is a hierarchy of CDFDs?

b. what is a hierarchy of modules?

c. what is the relation between module hierarchy and CDFD hierarchy?

d. what is the relation between a CDFD and its high level process in a CDFD hierarchy?

e. what does it mean by saying that modules M1 and M2 are relative modules?

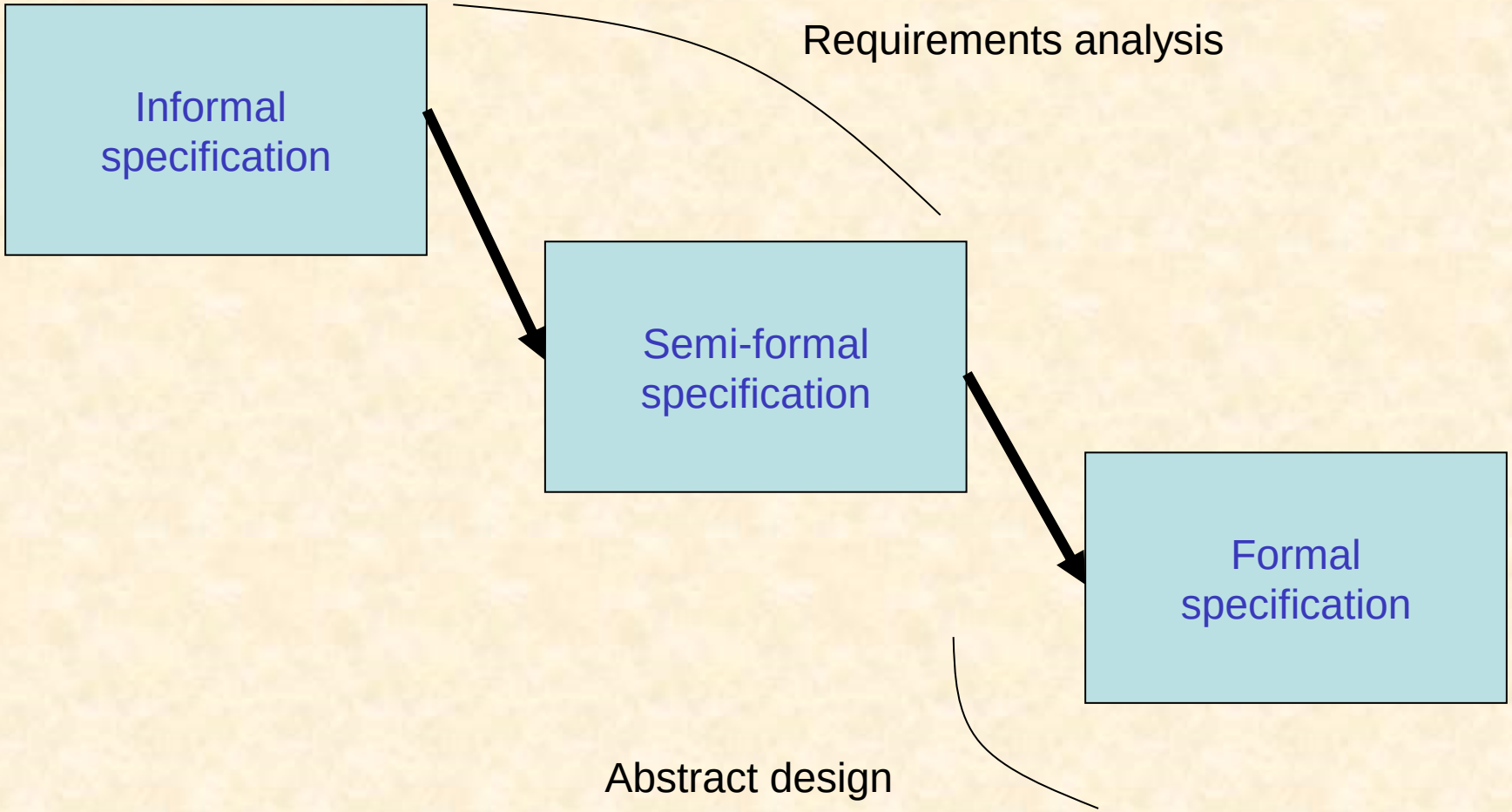
f. what is the scope of a variable, type identifier, constant identifier, invariant, function, and a process?

II.5 The SOFL Three-Step Approach to Writing Formal Specifications

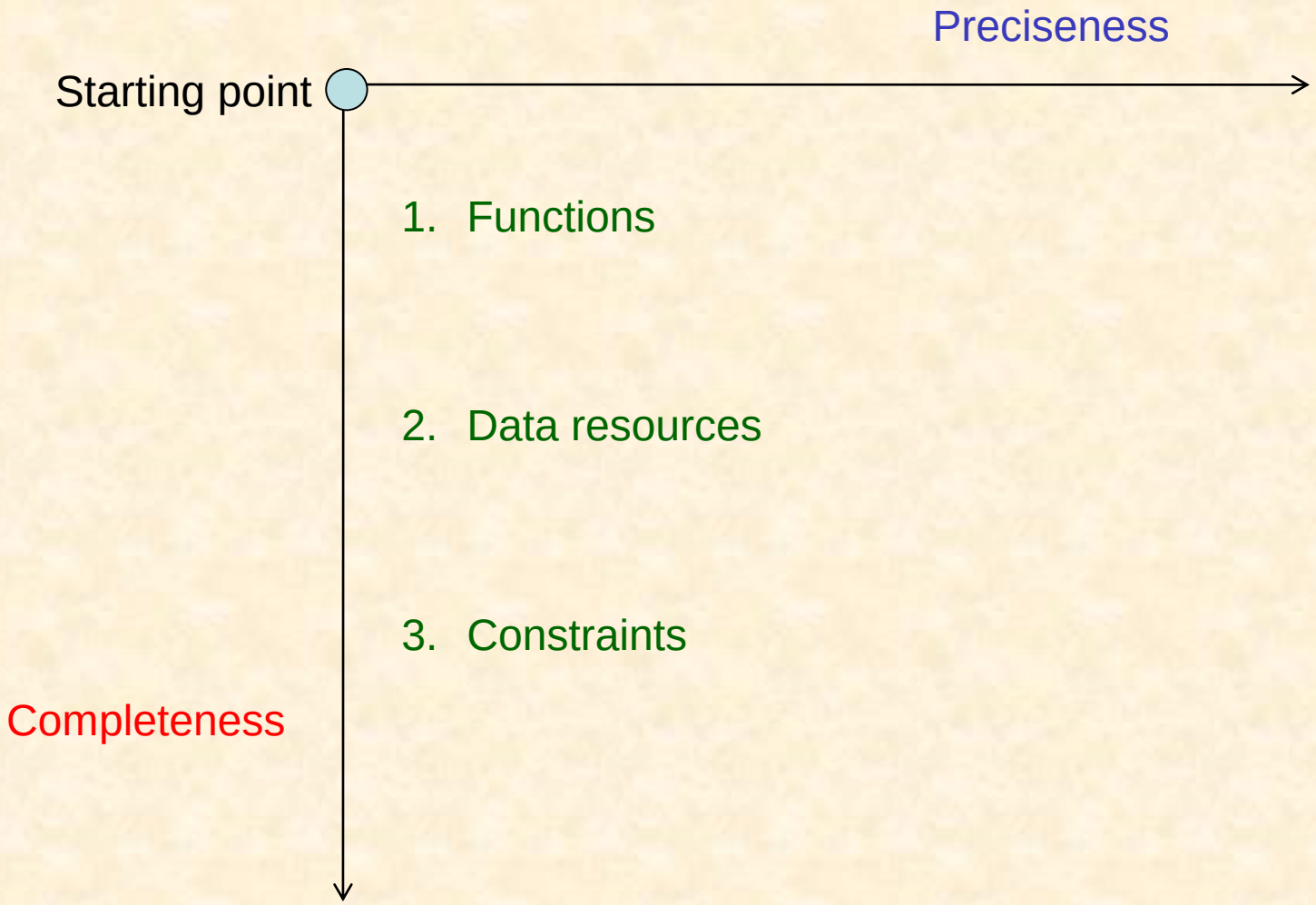
Question: how can a formal specification be effectively constructed with good qualities (completeness, consistency, validity)?

The SOFL three-step approach is a solution!

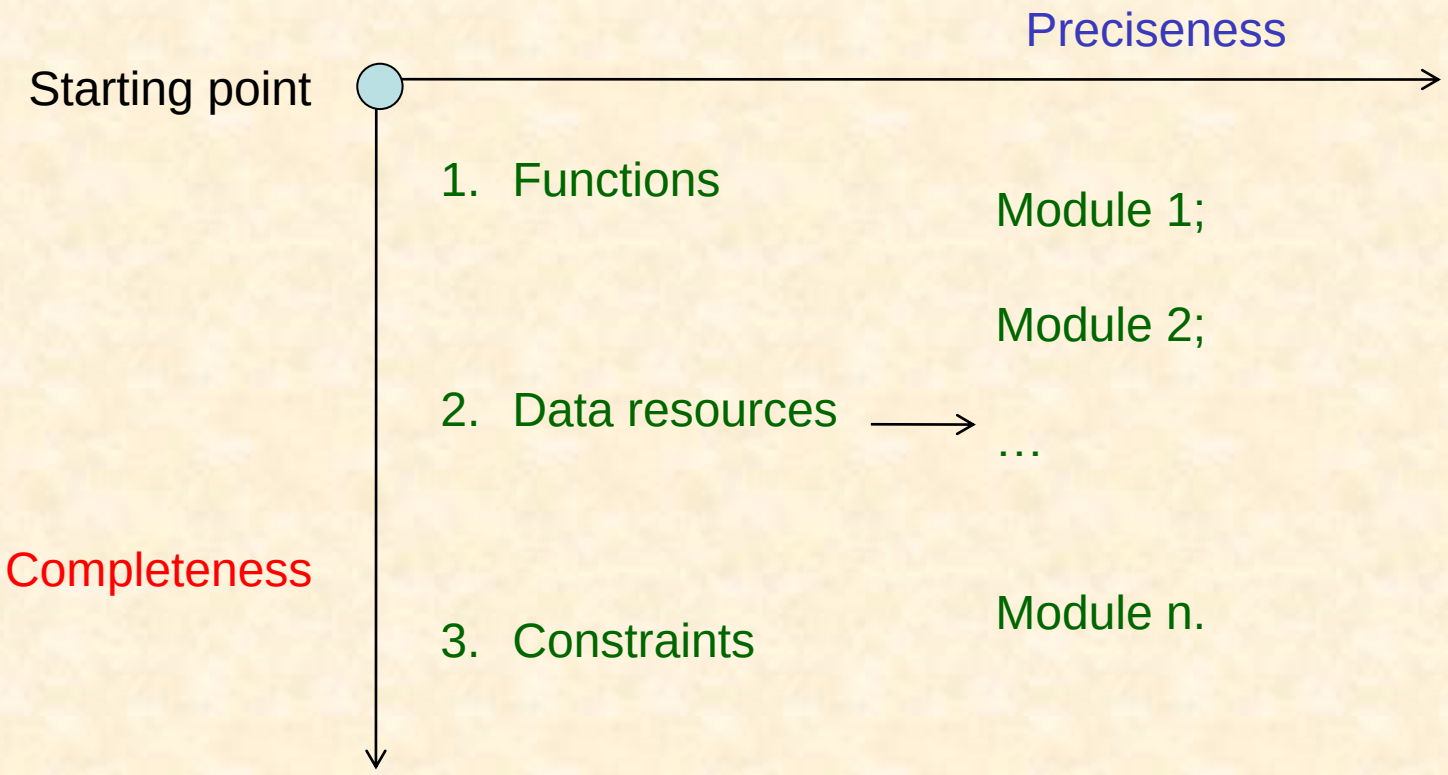
Three-step formal specification:



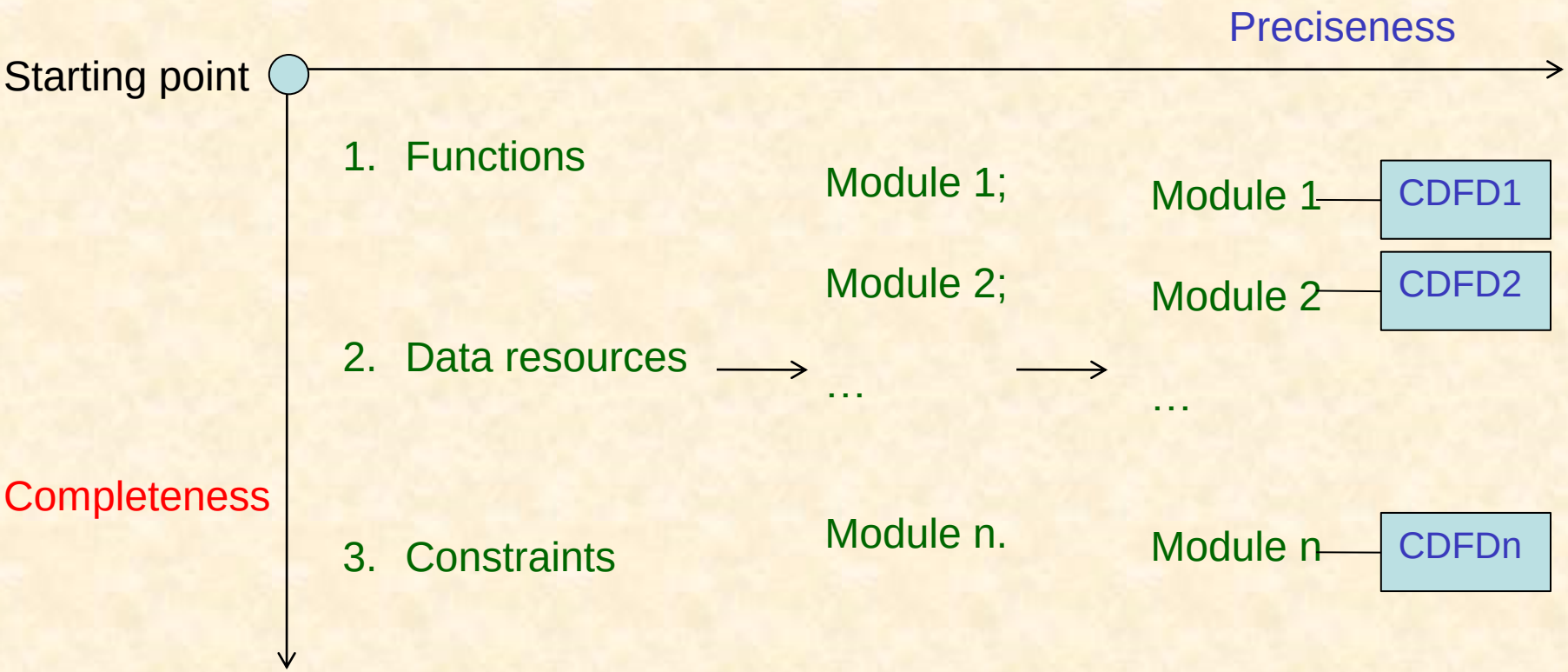
Tasks for informal specification: Capture desired functions, necessary data resources, and constraints on both functions and data resources.



Tasks for semi-formal specification: (1) Grouping related functions, data resources, and constraints into SOFL modules. (2) Defining necessary data types and variables. (3) Defining the function of each process using pre- and post-conditions at informal level.



Tasks for formal specification: (1) Describe system architecture using hierarchical CDFDs (condition data flow diagram). (2) Formalize the specification of each process in pre- and post-conditions.



A simplified ATM software

1. Informal specification (requirements analysis)

(1) **Functions**:

- Receive commands from the customer
- Confirm the customer's card id and password
- Withdraw
- Inquire about the customer's account balance

（2）**Data resources**:

- Customers’ account information

（3）**Constraints**：

- A customer can only access to his or her own account。
- No customer is allowed to borrow money from the bank。

Hierarchy of informal specification:

1. The functions of the system:

    (1) F1
    (2) F2
    (3) F3
    (4) F4

1.1 F1

    (1) F11
    (2) F12
    (3) F13

1.2 F2

    (1) F21
    (2) F22

• • • • • •

2. Recourses:

3. Constraints:

2. Semi-formal specification (requirements analysis):

```
module SYSTEM_ATM
   type
   Account = composed of
                  account_no: nat
                  password: nat
                  balance: real
                 end
 var
   account_file: set of Account;
 inv
    Account balance must be greater than or equal to zero.

 behav CDFD_No1;
 ...
```

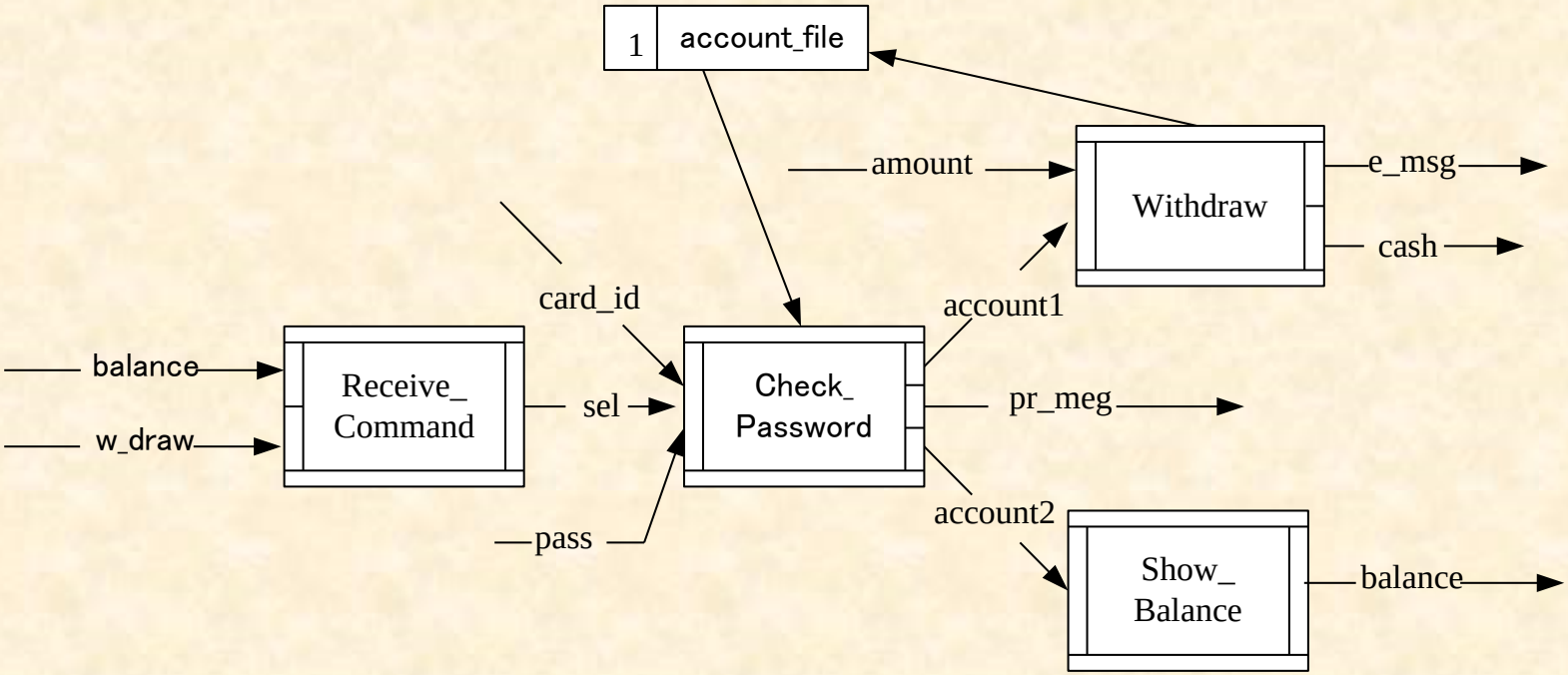
```
process Withdraw(amount: real, account1: Account)
                              e_msg: string | cash: real
 ext wr account_file
 pre account1 is a member of account_file
 post if amount is not greater than the balance of account1
        then supply cash with the same amount as amount, and
                reduce the amount from the balance of the account.
        else output an appropriate error message.
end_process;

...

end_module
```

3. Formal specification (abstract design)

The top level CDFD:



```
module SYSTEM_ATM
   type
   Account = composed of
                    account_no: nat
                    password: nat
                    balance: real
                  end
var
   account_file: set of Account;
inv
    forall[x: Account] | x.balance >= 0;

behav CDFD_No1;
...
```

```
process Withdraw(amount: real, account1: Account)
                        e_msg: string | cash: real
 ext wr account_file
 pre account1 inset account_file
 post if amount <= account1.balance
        then
          cash = amount and
          let Newacc =
              modify(account1, balance -> account1.balance – amount)
           in
            account_file = union(diff(~account_file, {account1}), {Newacc})
       else
         e_meg = "The amount is over the limit. Reenter your amount.")
comment
…
end_process;
end_module
```

Small Project (40%)

A stock reservation and purchase system is required to have the following functions and constraints:

1. Register a customer (name, age, bank account, address, email address, telephone number, and purchased stocks)
2. Register a stock (name, price, unit for sale, limit for selling to one customer)
3. Cancel the information of a registered customer
4. Cancel the information of a registered stock

5. Purchase a stock (the purchased stock details should be recorded in the customer's information and the payment for the stock should be made from the customer's bank account)
6. Sell a stock by a customer (the money resulted from the sale of stock should be put into the customer's bank account)
7. For each kind of stock, a customer can only buy at most 1000 units of the stock.
8. A stock name is unique.
9. A customer's name is also unique.
10. The currency used in the system is RMB.

Requirements for students

(1) Take the SOFL three-step approach to construct a formal specification for the stock system. That is, write an informal specification, semi-formal specification, and formal specification.

(2) Independently finish this small project and and submit it to Dr. Wang Ying before 27th December 2022.