Structure Charts

The Structure Chart

- Structure chart
  - A hierarchical diagram showing the relationships between the modules of a computer program
  - The objective of structured design is to create a top-down decomposition of the functions to be performed by a given program in a system – use a structure chart to show this
  - E.g. shows functions and subfunctions (such as *Calculate base amount*, *Calculate overtime amount* etc.)
  - Uses rectangles to represent each such module (the basic component of a structure chart
  - Higher-level modules are “control” modules that control flow of execution (call lower level modules which are the “worker bee” modules that contain program logic)

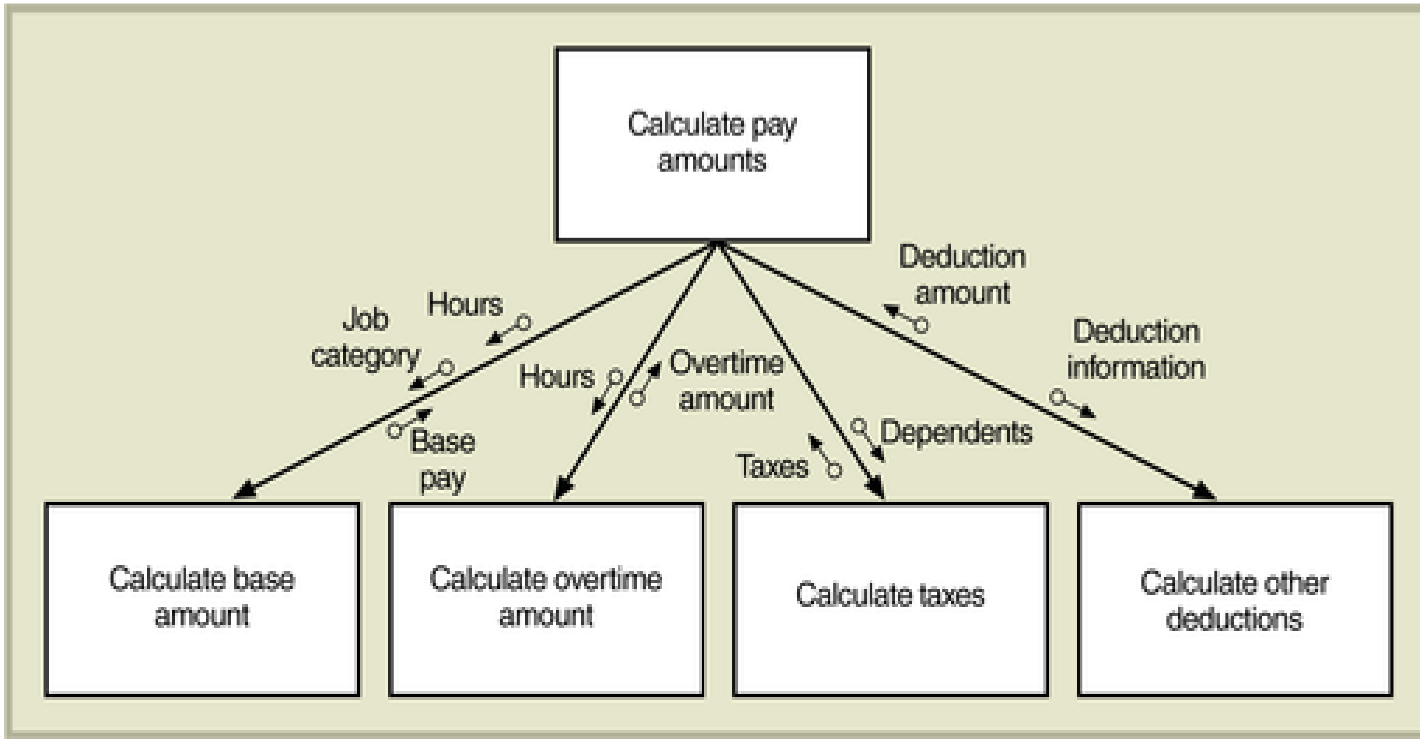
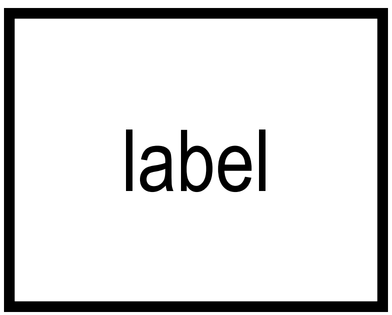


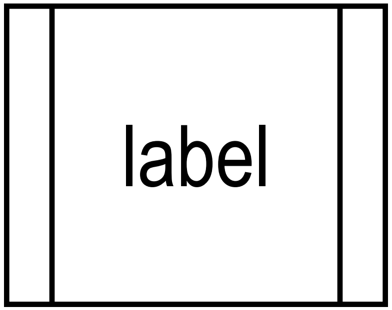
FIGURE 9-9
A simple structure chart for the Calculate pay amount.

Structure Charts - Module

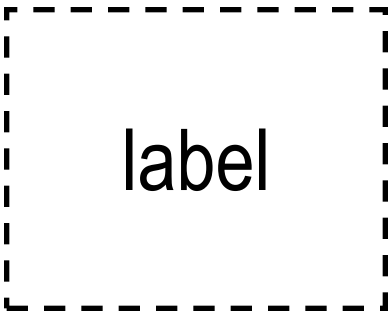


- process / subroutine / task
- unit of execution
- accepts parameters as inputs
- produces parameters as outputs
- parameters: data or control
- can be invoked and can invoke
- label: verb
- linked to module specification

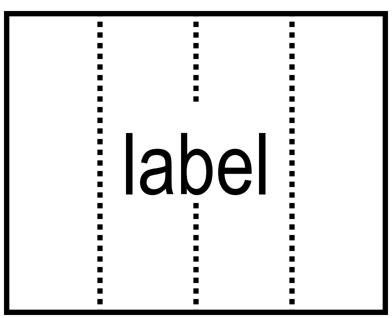
Structure Charts - Special Modules



- predefined (reused) module
- highly useful

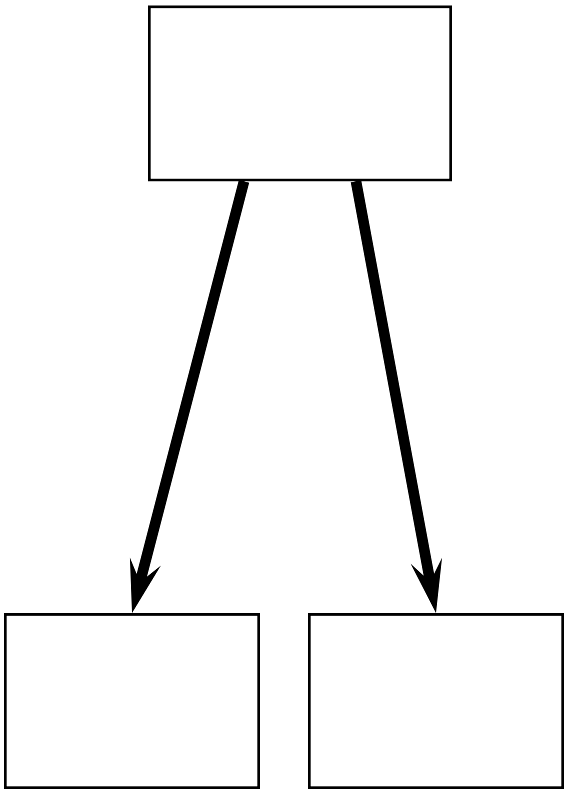


- “macro” module
- avoid



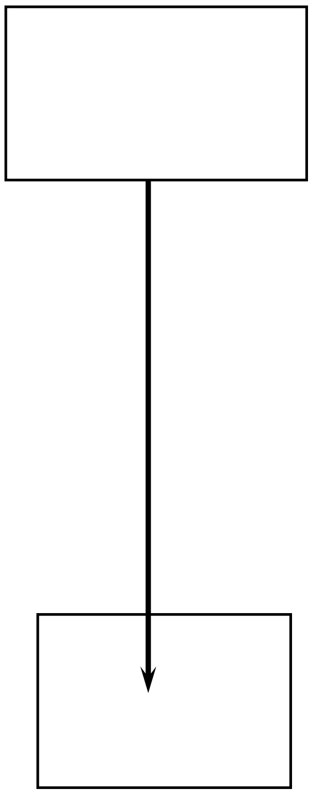
- multi-entry module
- avoid

Structure Charts - Invocation / Call



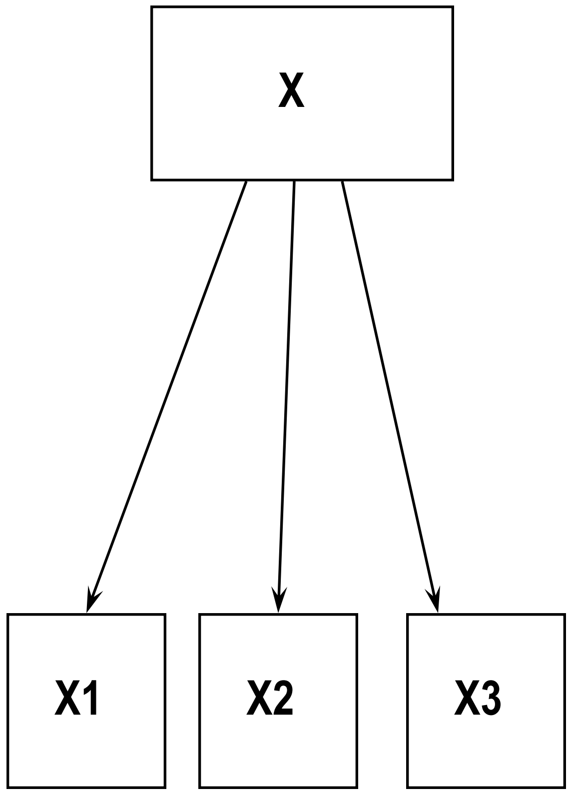
- call of subordinate module
- connector element
- NOT a data flow
- one specific form of control flow
- has a direction
- no split or join
- NO label

Structure Charts - Invocation / Jump to Address



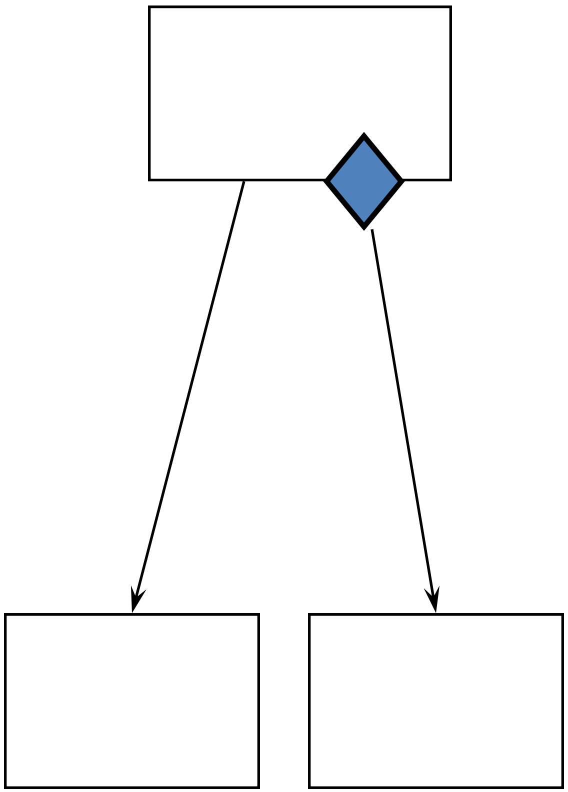
- call jumps INTO invoked module
- assembler type of programming
- modification at run-time
- avoid

Structure Charts - Sequence of Execution



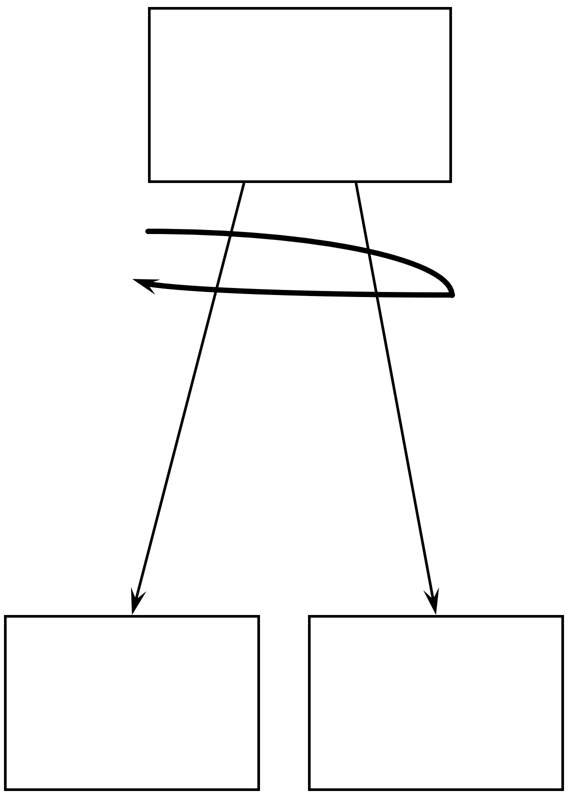
- sequence of subordinate modules in the diagram is not reflecting a binding sequence of invocation
- sequence of invocation is defined in the specification of the super-ordinate module
- module specification is the decisive element

Structure Charts - Conditional Execution



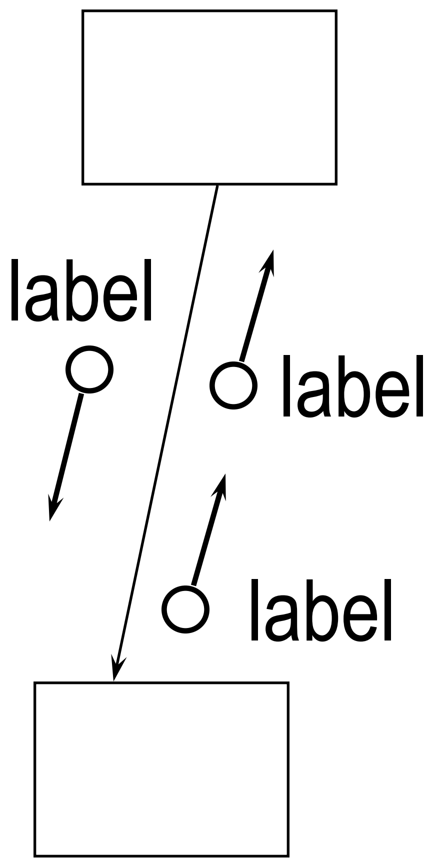
- call of subordinate module depends on a condition
- no label
- condition is defined in the module specification
- module specification is the decisive element

Structure Charts - Loops in the Execution



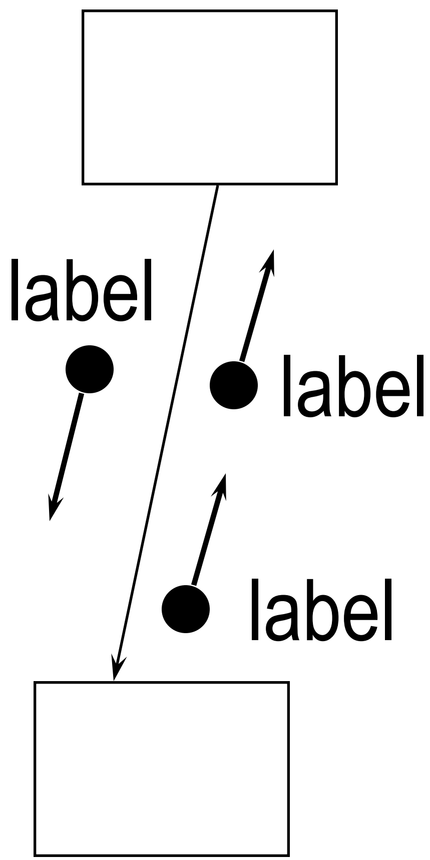
- call of subordinate modules runs in a loop
- no label or condition
- loop (and its condition) is defined in the module specification
- module specification is the decisive element

Structure Charts - Data Flow



- flow of information
- data transfer
- bound to invocation
- has a direction
- no splits or joins
- label: noun
- specified in data-dictionary

Structure Charts - Control Flow

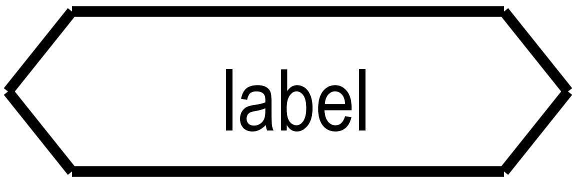


- flow of control (<> invocation)

  ==> control execution path

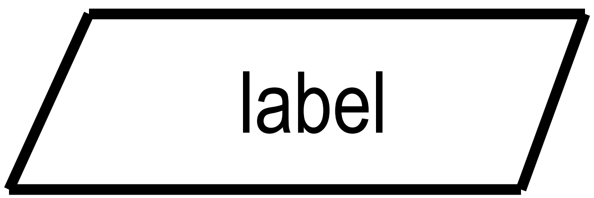
  of targeted module
- bound to invocation
- has a direction
- no splits or joins
- label: flag, decision, condition
- specified in data-dictionary

Structure Charts - Data Store



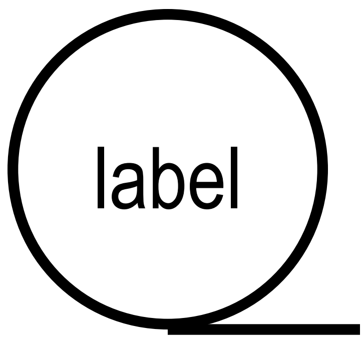
- storage for permanent data
- database / file
- passive; no activity beyond basic retrieval capacity
- serviced by a module
- label: noun
- specified in data-dictionary and/or with an ER-diagram

Structure Charts - Devices / Interfaces



- provides connection to peripheral devices
- origin / destination of external data flows (controls)
- not part of the software to be developed
- label: noun
- specified in data-dictionary

Structure Charts - SW Infrastructure

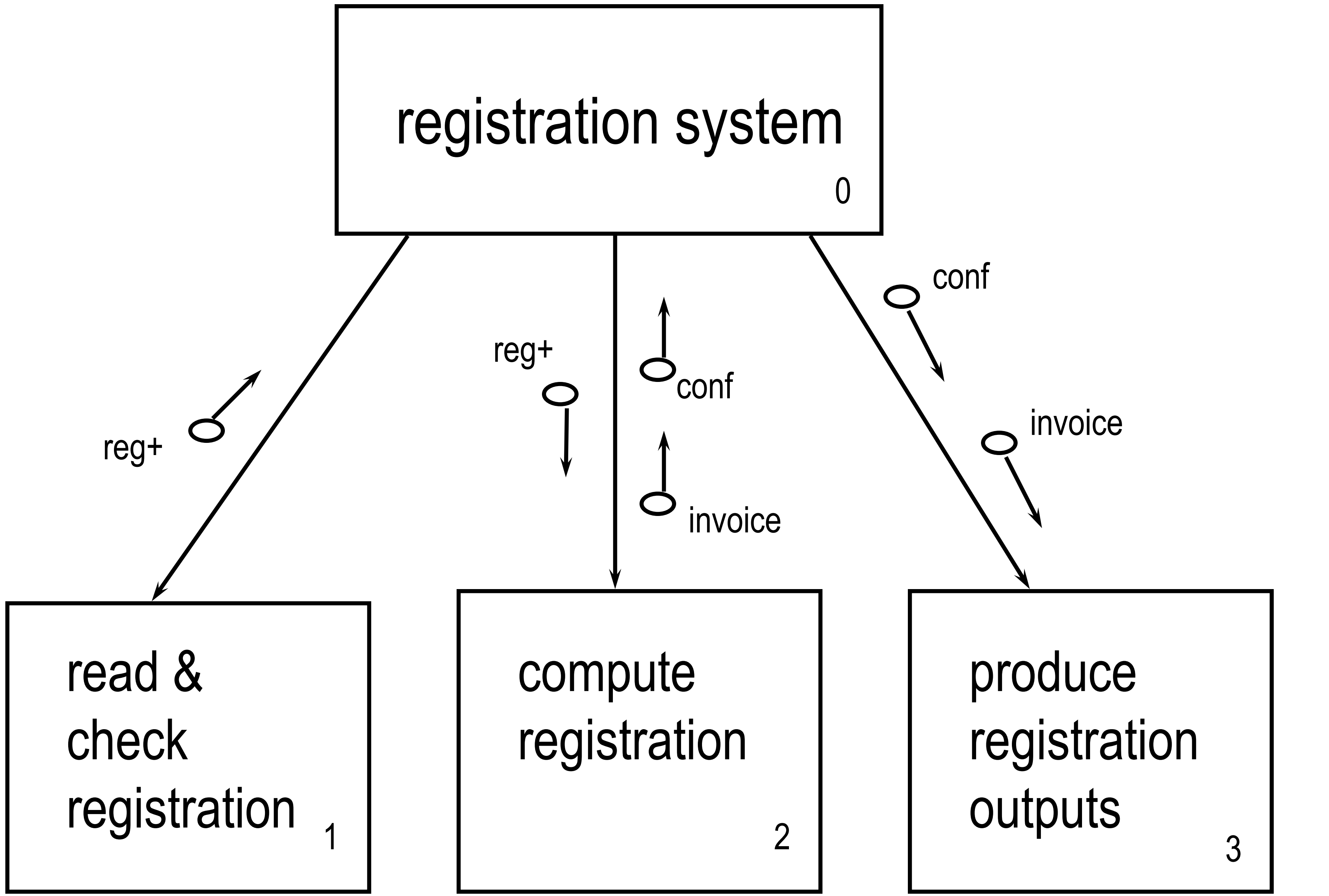


- provides connection to external systems, databases, operating system, etc.
- origin / destination of external data flows (controls)
- not part of the software to be developed
- label: noun
- specified in data-dictionary

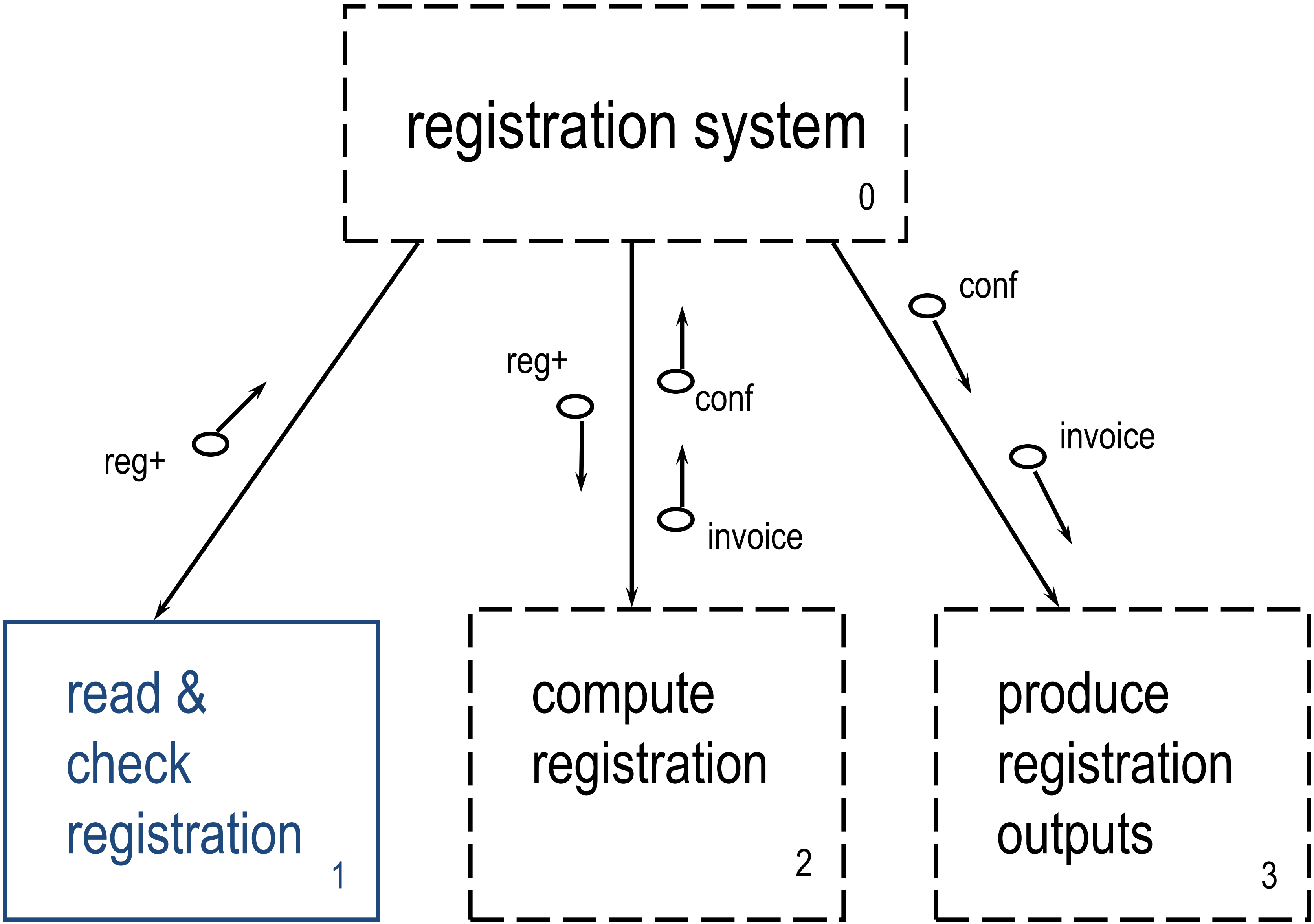
STRUCTURE CHARTS

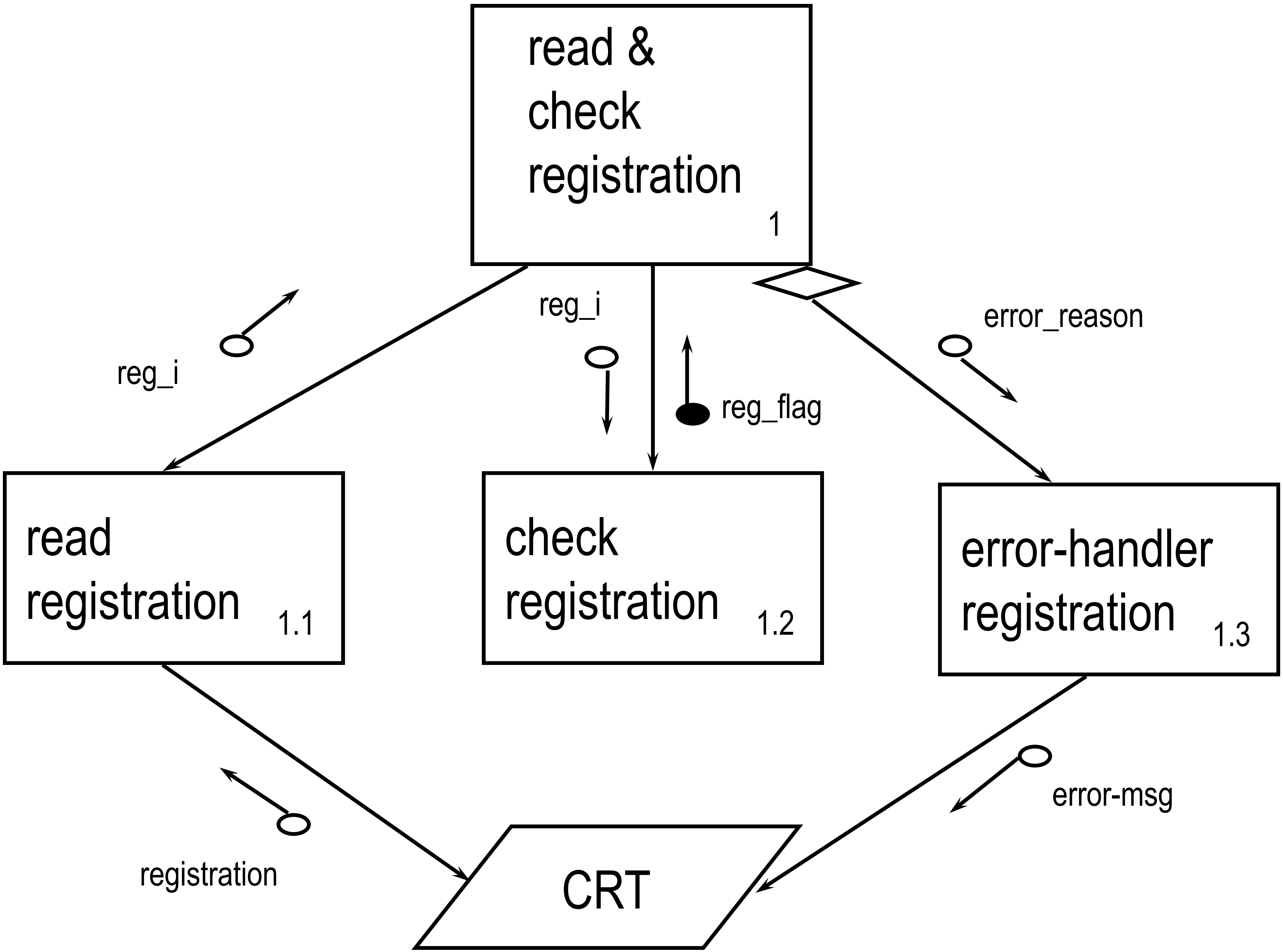
An Example:

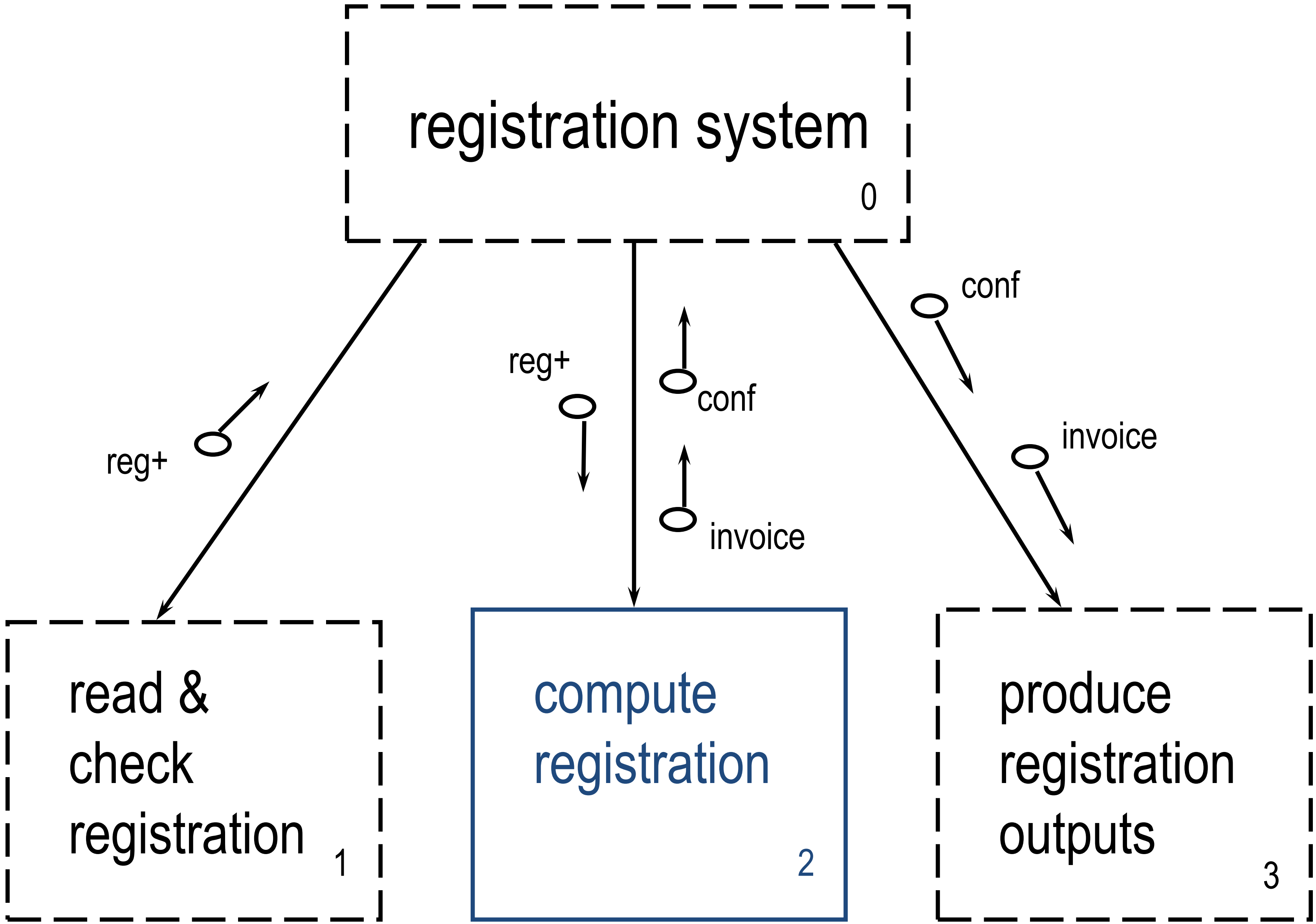
The Registration System

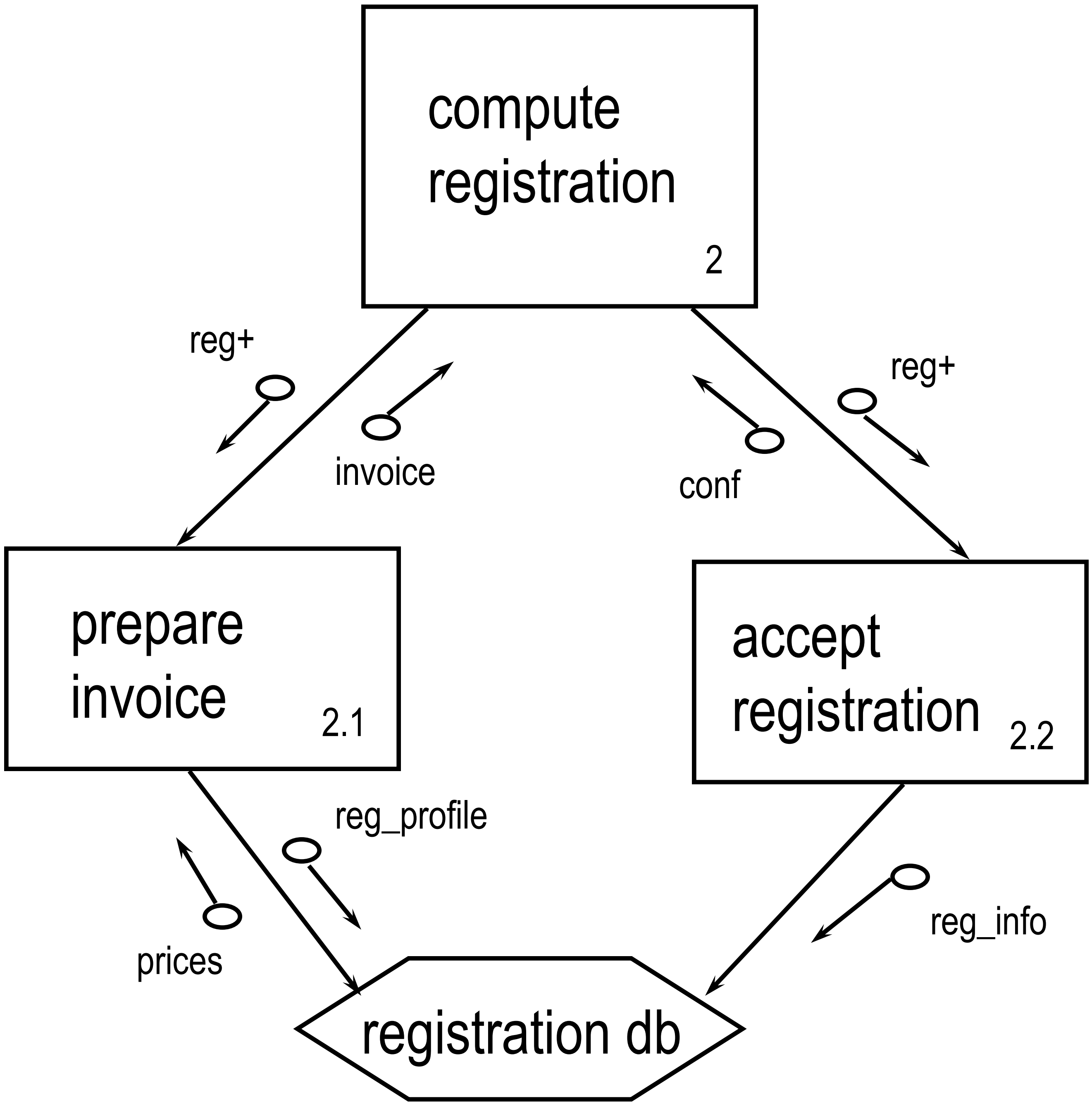


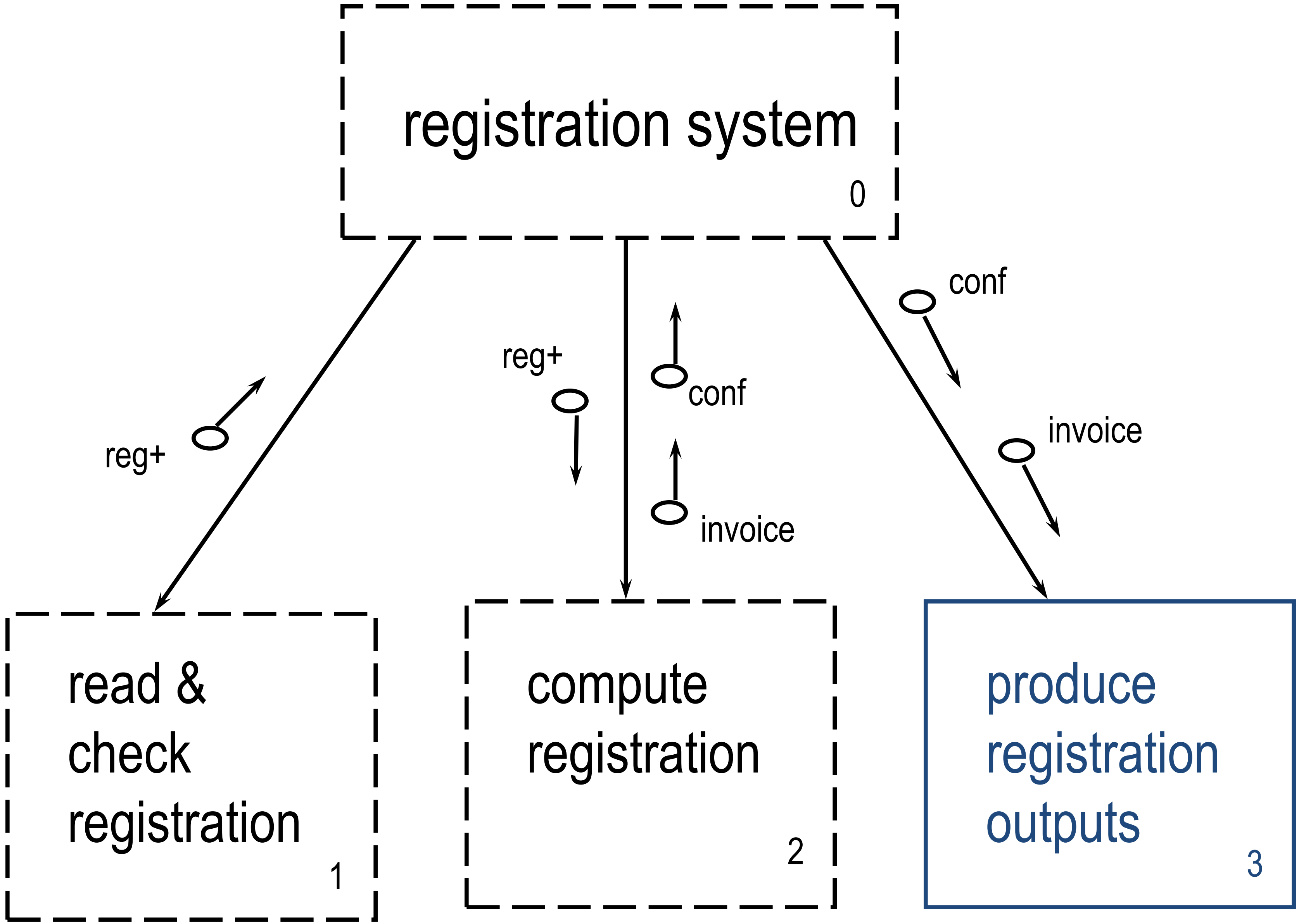
registration system
0
reg+
reg+
conf
invoice
conf
invoice
read & check registration
1
compute registration
2
produce registration outputs
3


registration system
0
reg+
reg+
conf
invoice
conf
invoice
read &
check
registration
1
compute
registration
2
produce
registration
outputs
3

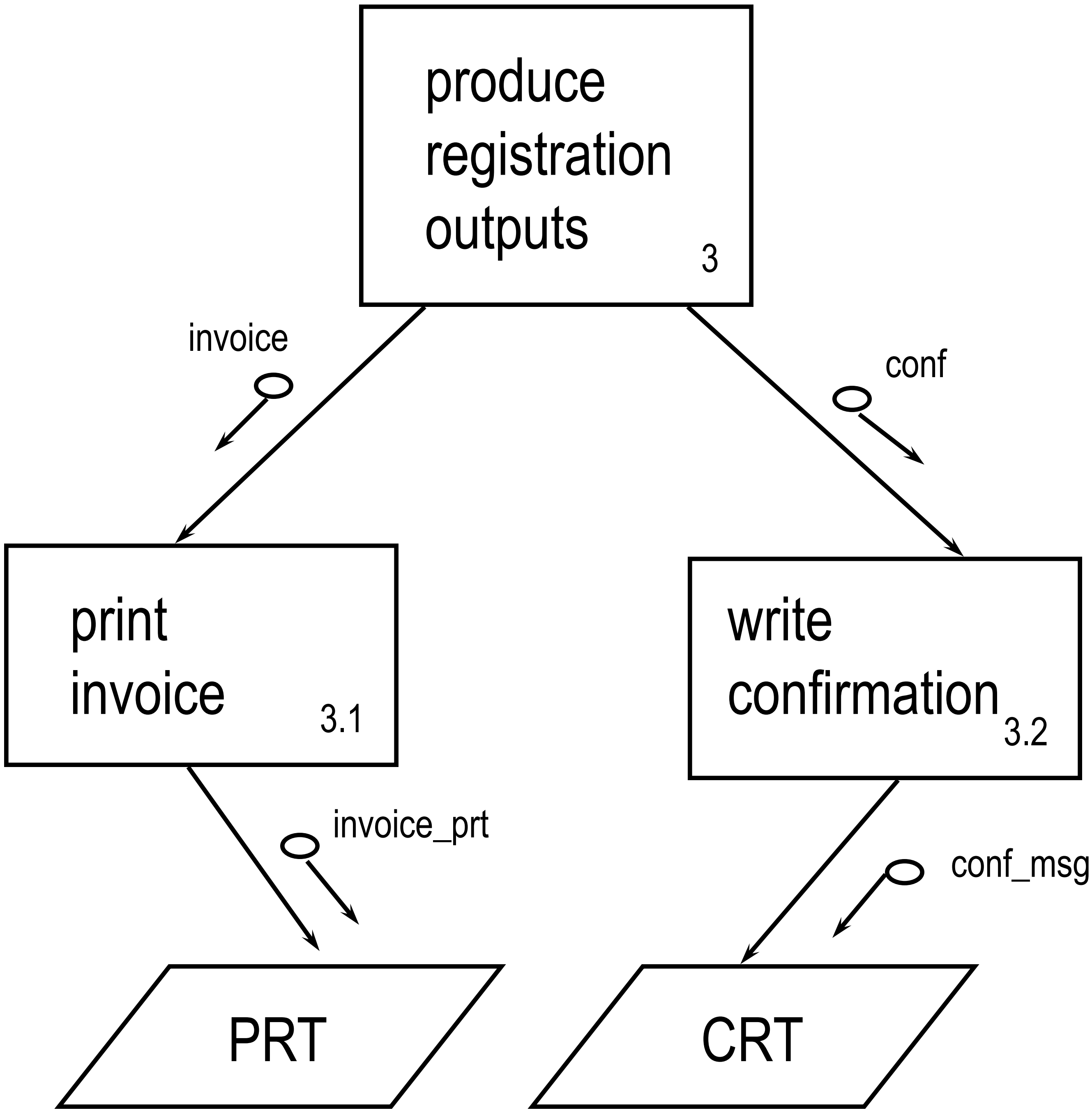

read &
check
registration
1
reg_i
reg_i
reg_flag
error_reason
read
registration
1.1
check
registration
1.2
error-handler
registration
1.3
registration
error-msg
CRT

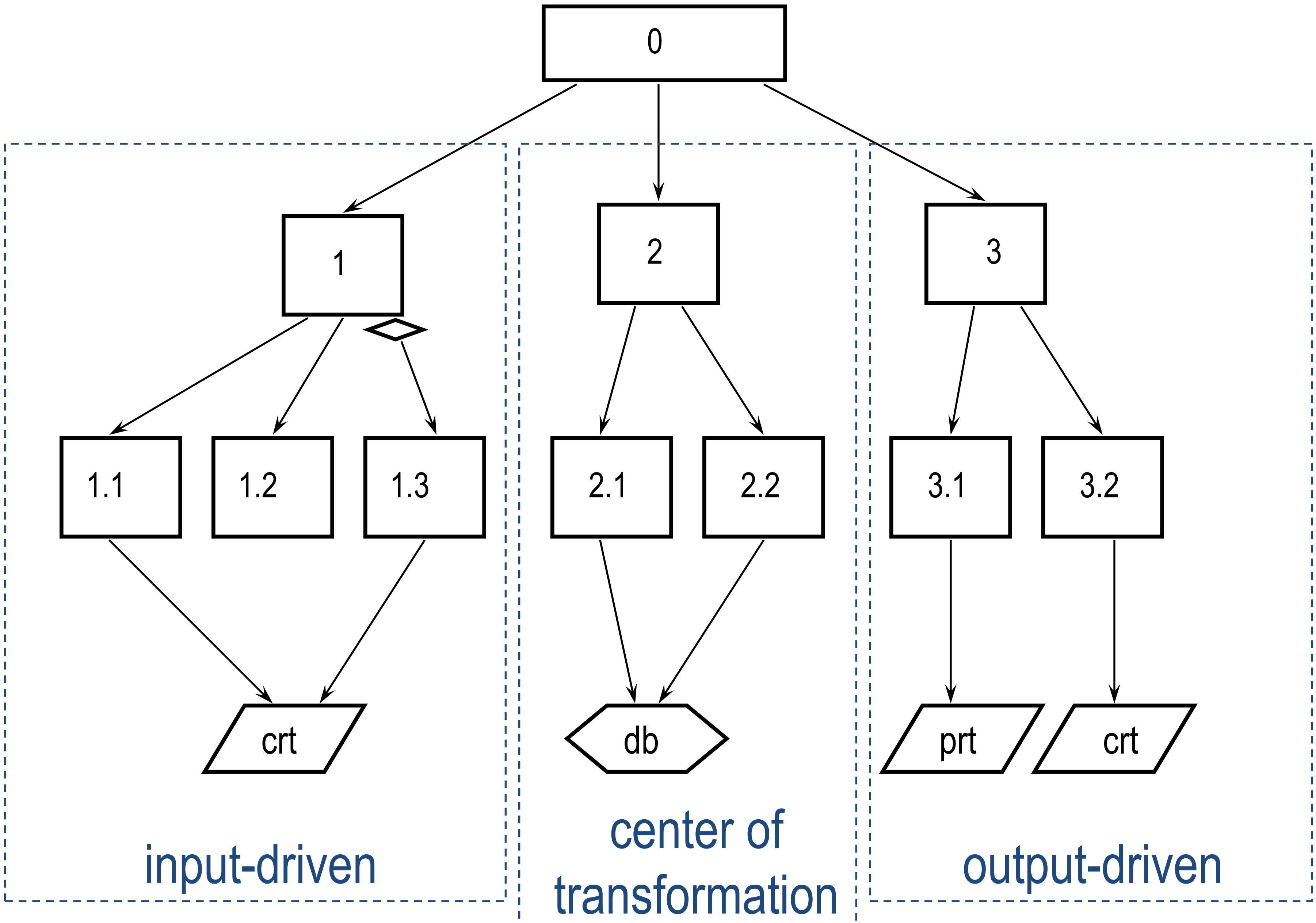

registration system
0
reg+
reg+
conf
invoice
conf
invoice
read &
check
registration
1
compute
registration
2
produce
registration
outputs
3


compute registration
2
reg+
invoice
reg+
conf
prepare invoice
2.1
accept registration
2.2
reg_profile
prices
reg_info
registration db



registration system
0
reg+
reg+
conf
invoice
conf
invoice
read & check registration
1
compute registration
2
produce registration outputs
3


produce registration outputs
3
invoice
conf
print invoice
3.1
write confirmation
3.2
invoice_prt
conf_msg
PRT
CRT

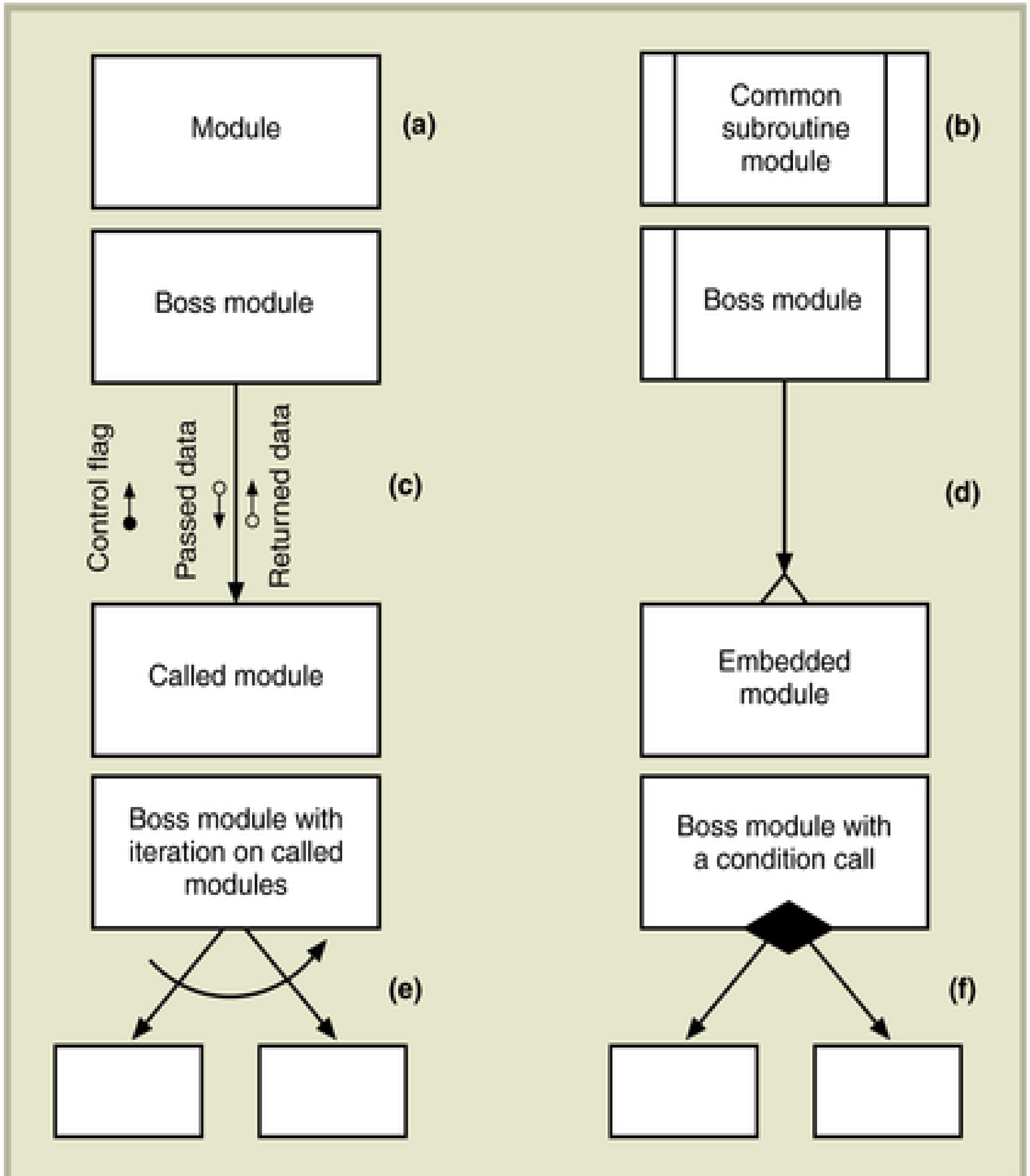

0
1
2
3
1.1
1.2
1.3
2.1
2.2
3.1
3.2
crt
db
prt
crt
input-driven
center of transformation
output-driven

Structure Chart characteristics

- Structure chart is based on idea of modular programming (and top-down programming)
  - Breaking a complex problem down into smaller modules
  - Modules are well formed with high internal cohesiveness and minimum data coupling
  - Vertical lines connecting the modules indicaqte calling structure
  - Little arrows next to the lines show the data passed between modules (inputs and outputs)
    - <u>Data couples:</u> individual items that are passes between modules
    - <u>Program call:</u> the transfer of control from a module to a subordinate module to perform a requested service (can be implemented as e.g. a function or procedure call)

F I G U R E 9 - 1 0

Structure chart symbols.



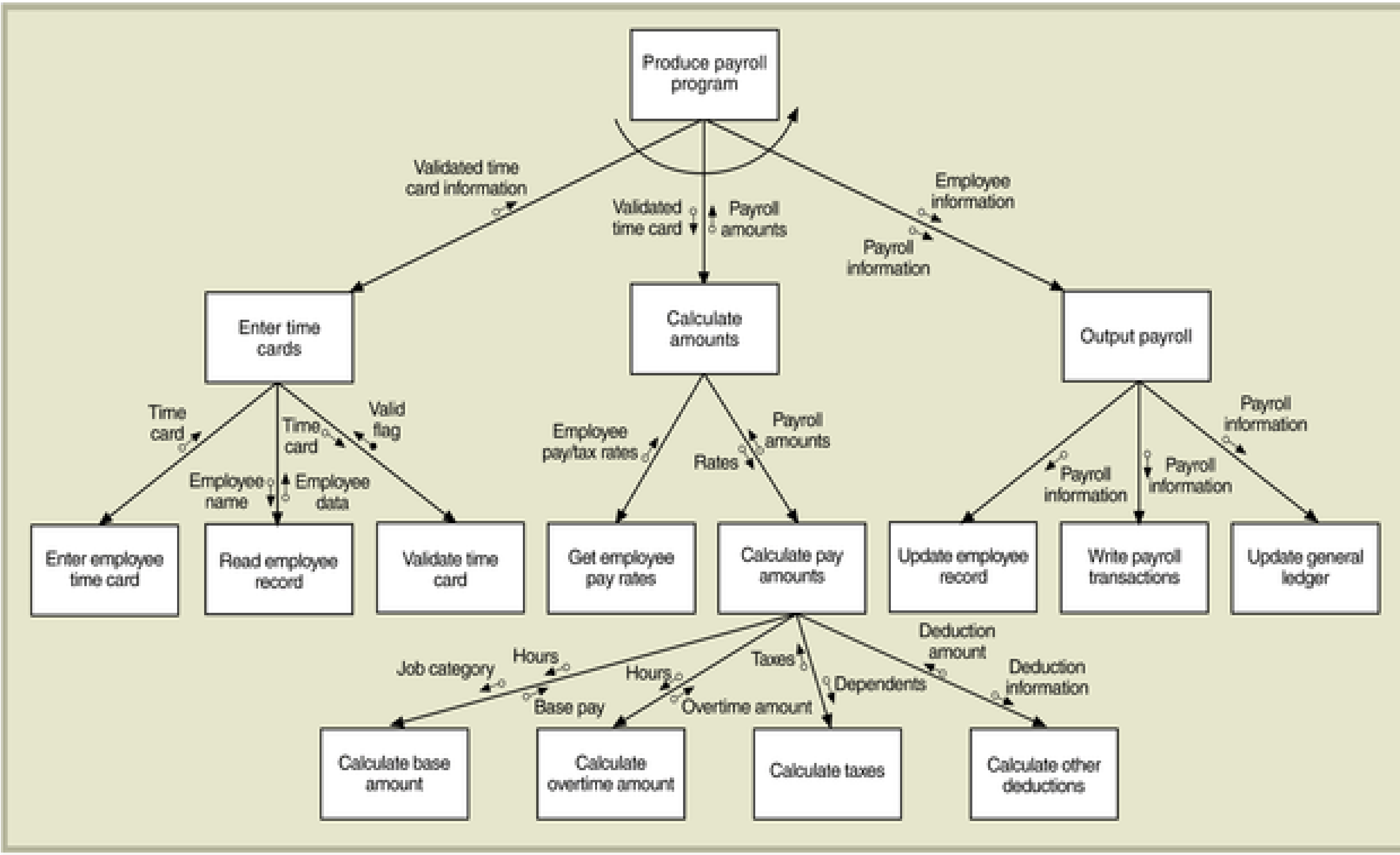


FIGURE 9-11

A structure chart for the entire Calculate payroll program.

Notes on Structure Chart

- Each module does a very specific function
- Module at top of the tree is the boss module
  - Function is to call modules on level below, pass information to them
- Middle level modules control the modules below
  - Call them and pass data
- At the very bottom, the leaves contain the actual algorithms to carry out the program functions
- Call of modules is left to right across the tree

Developing a Structure Chart

- Transaction analysis
  - The development of a structure chart based on a DFD that describes the processing for several types of transactions
  - Uses as input the system flow chart and the event table <u>to develop the top level</u> of the tree in a structure chart
- Transform analysis
  - The development of a structure chart based on a DFD that describes the input-process-output data flow
  - Used <u>to develop the subtrees</u> in a structure chart – one for each event in the program

Transaction Analysis

- First step is to identify the major programs
  - Can do by looking a the system flow chart and identifying the major programs (e.g. see figure 9-8 – the system flow chart for RMO example)
- For a subsystem or program we want to make a structure chart for, we can look a the event-partitioned DFD to identify the major processes
  - See next slide, shows event-partitioned DFD for the order-entry subsystem for RMO example, showing <u>5 major processes</u>
  - <u>These major processes will become the modules in the resulting high level structure chart (see slide after that)</u>

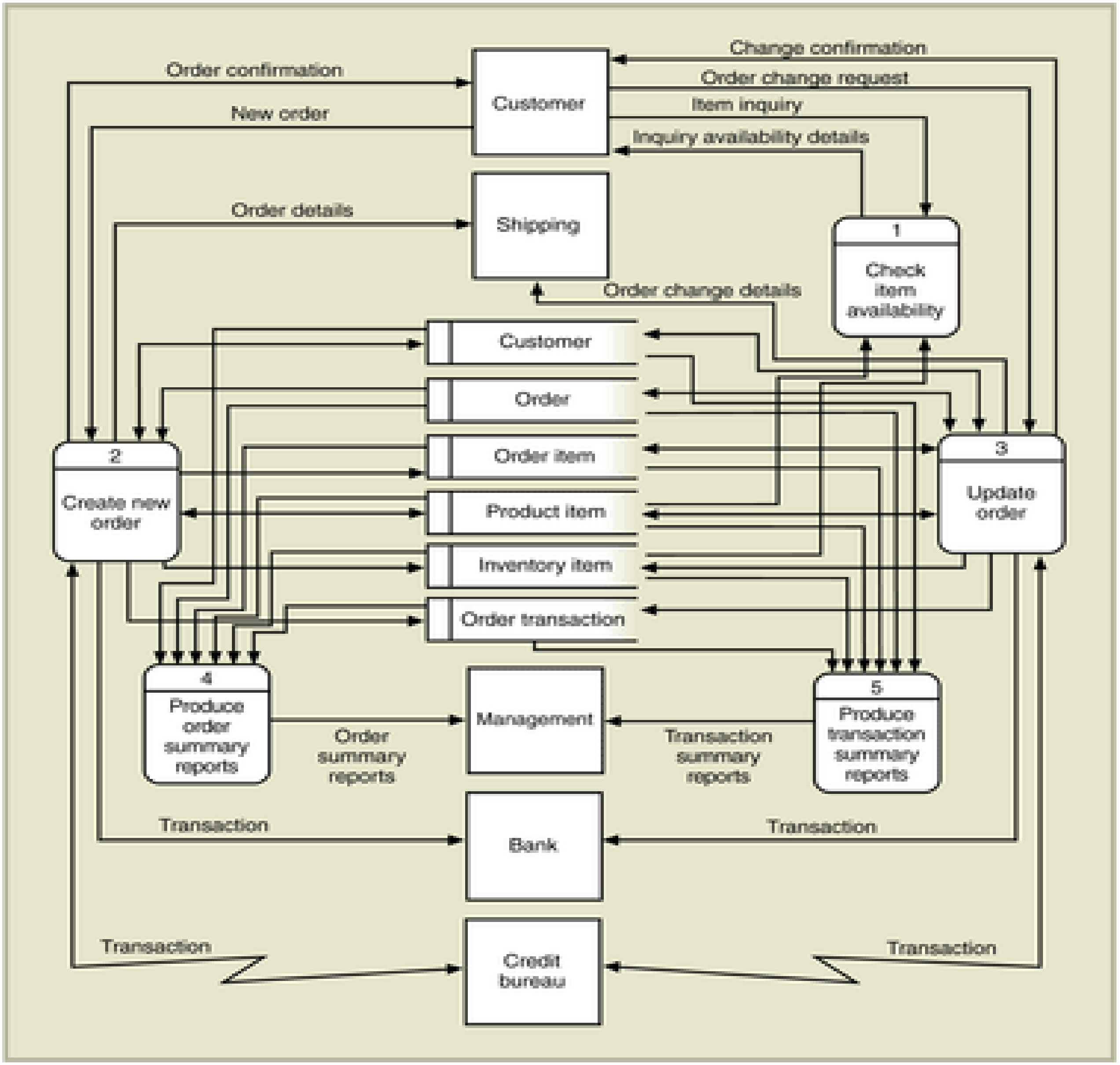


FIGURE 9-12
Event-partitioned DFD for the order-entry subsystem.

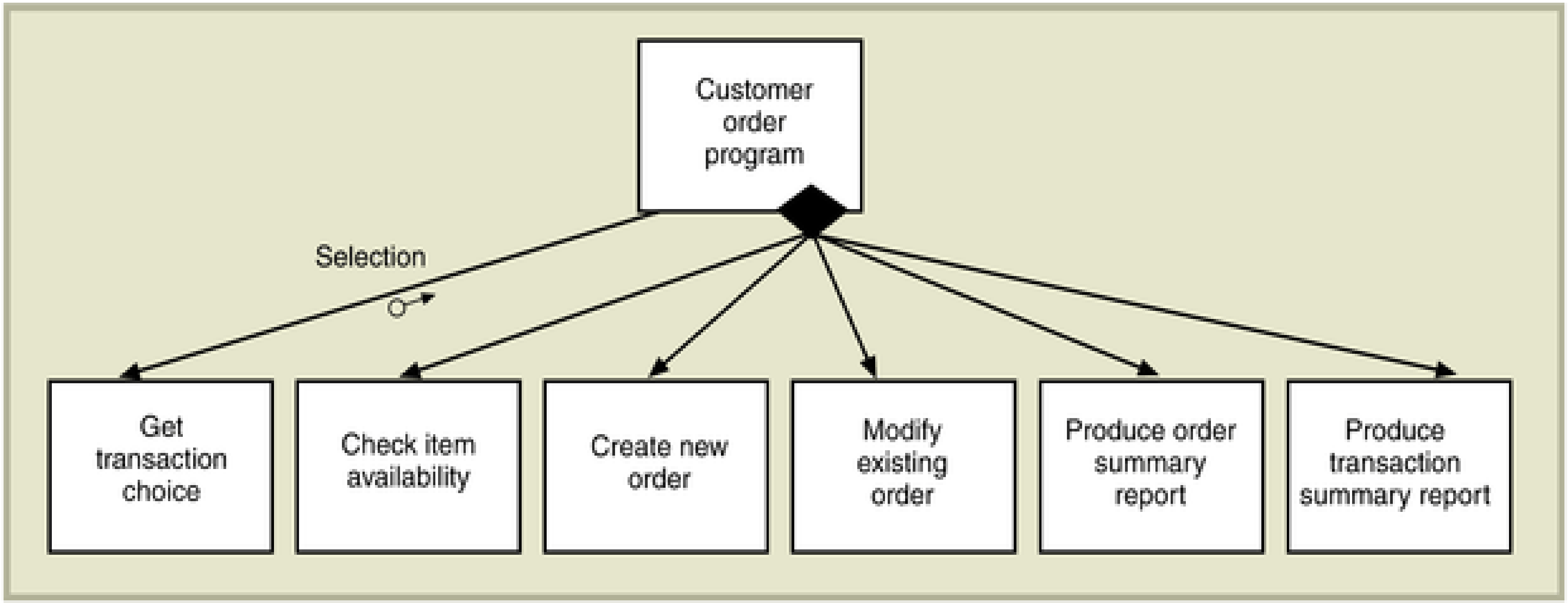


FIGURE 9-13
High-level structure chart for the customer order program.

Transform Analysis

- Based on idea that computer program “transforms” input data into output data
- Structure charts developed with transform analysis usually have 3 main subtrees
  - Input subtree to get data
  - A calcuate subtree to perform logic
  - An output subtree to write the results
- Can create by rearranging elements from
  - DFD fragment for an event (e.g. “create new order”)
  - The detailed DFD for that event
- E.g. see next two slides for “create new order” DFD fragment, and its corresponding detailed DFD

FIGURE 9-14
The Create new order *DFD fragment.*

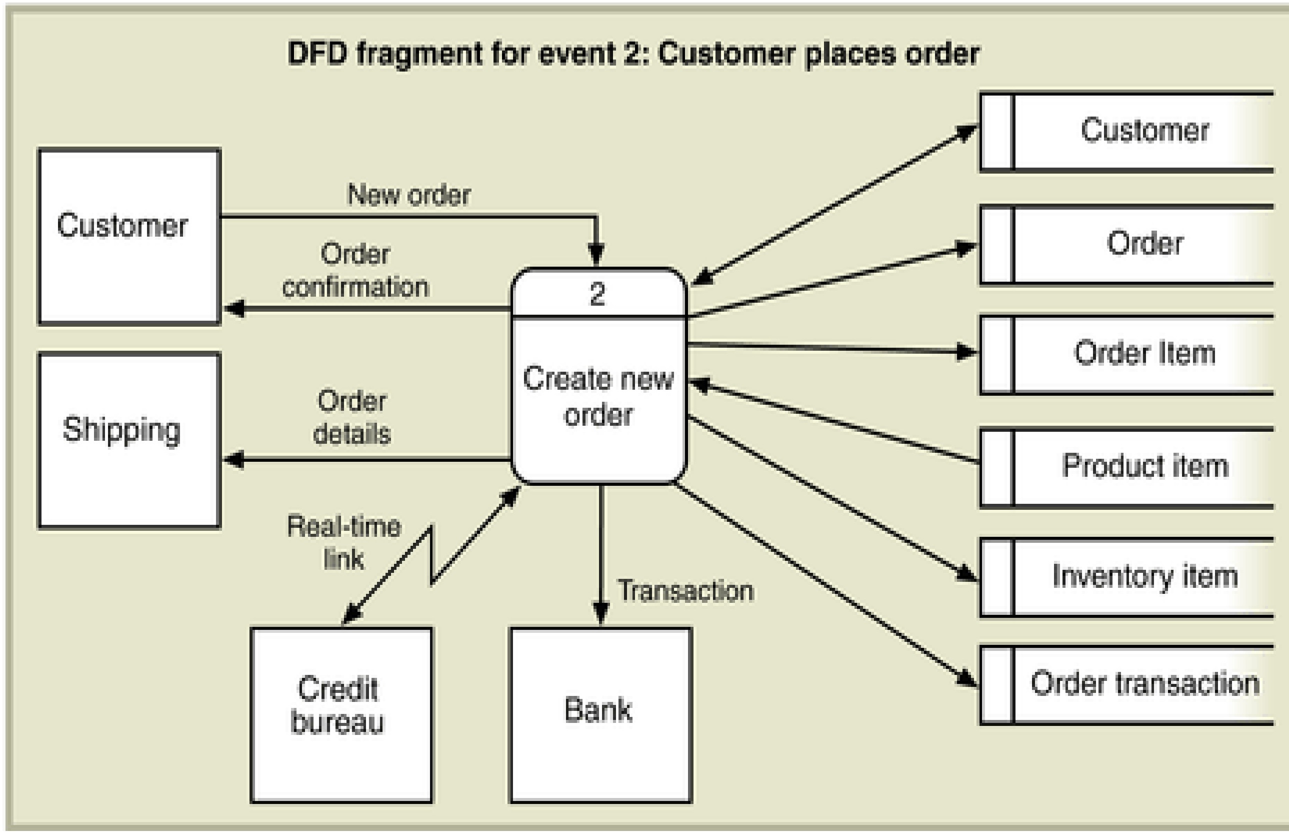
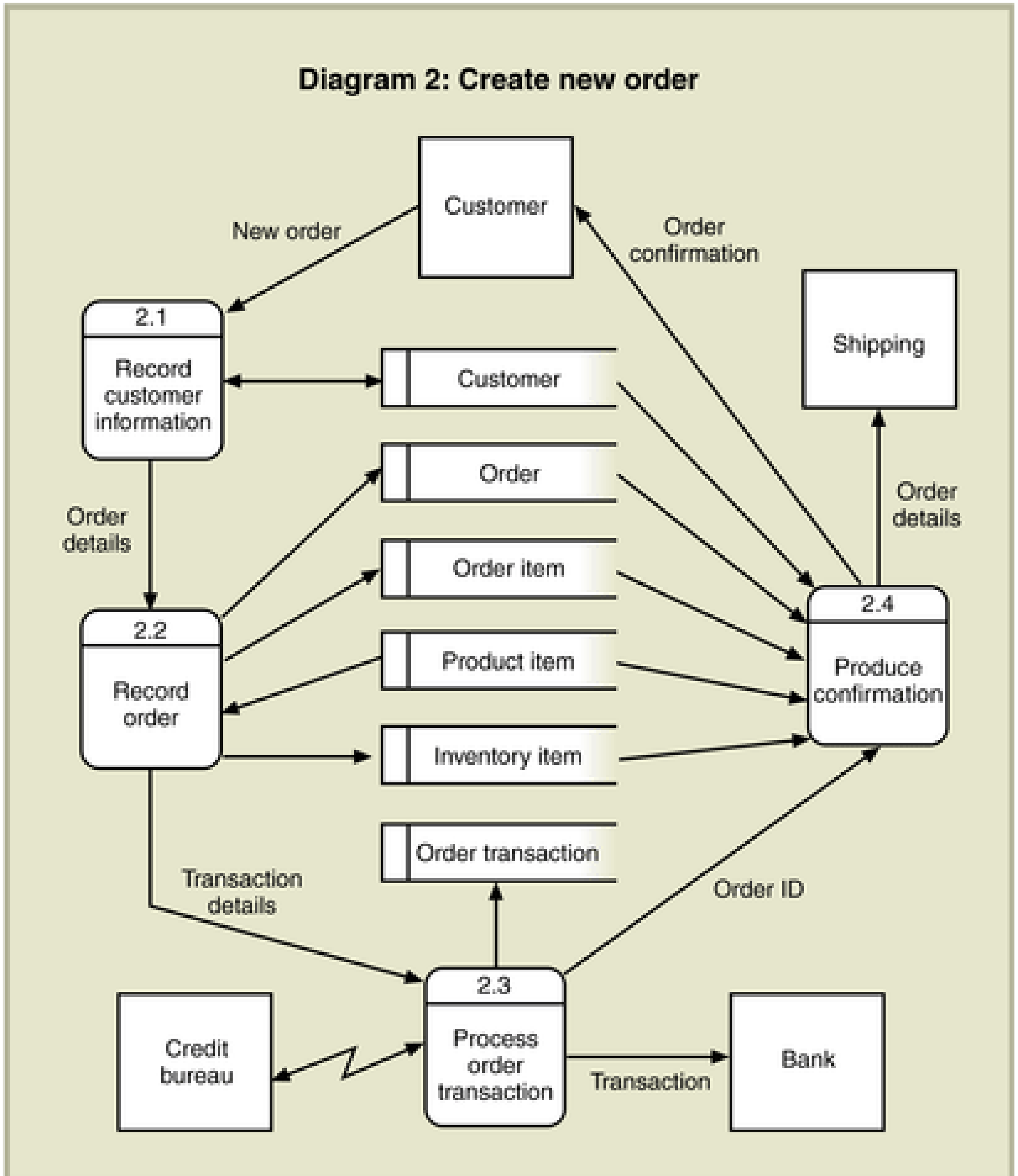


FIGURE 9-15
Exploded view of the Create new order DFD.



Steps for creating the structure chart from the DFD

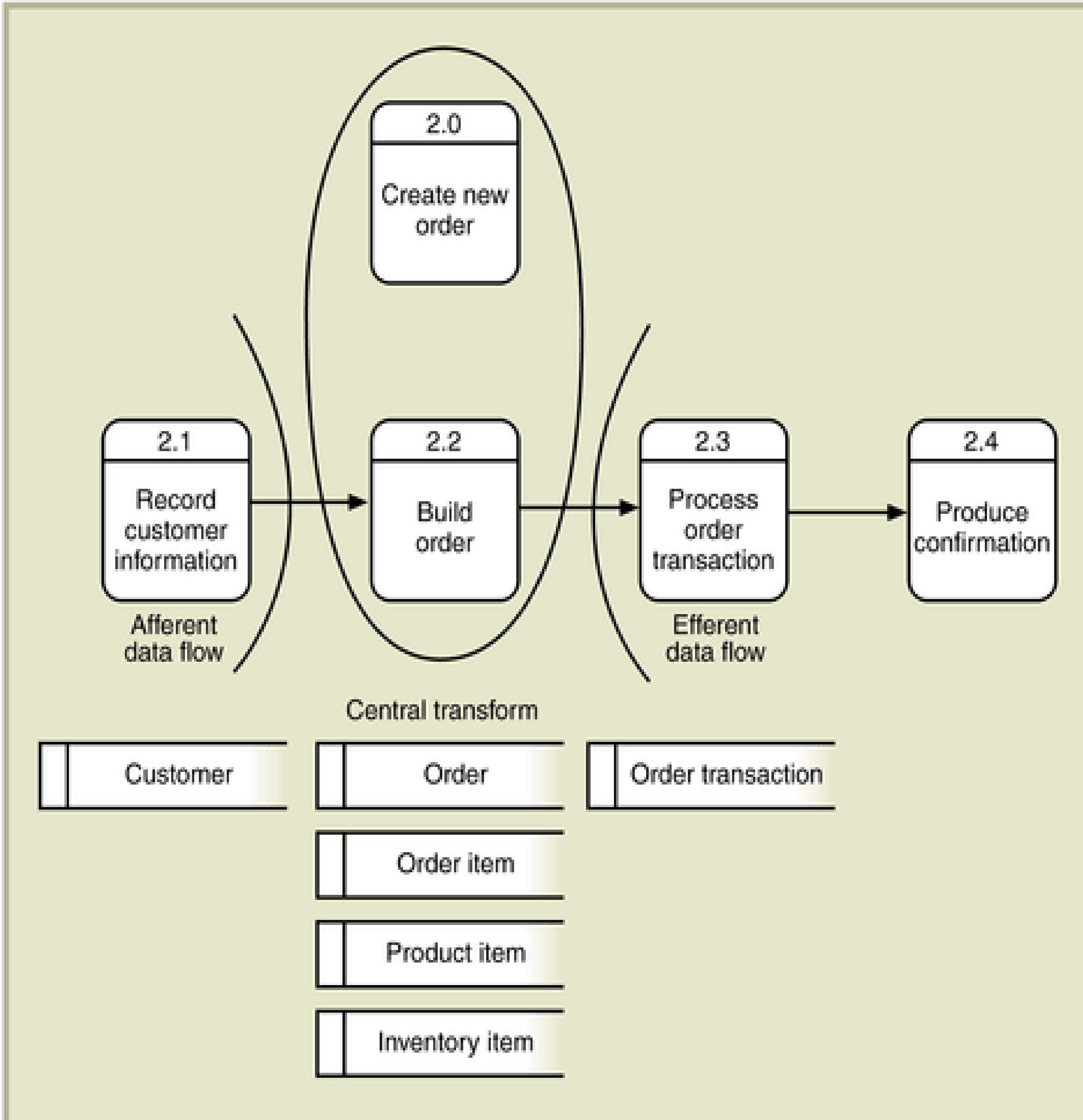
1. Identify the primary information flow
2. Find the process that represents the most fundamental change from an input stream to an output stream – the central transform
   - Afferent data flow: the incoming data flow in a sequential set of process bubbles
   - Efferent data flow: the outgoing data flow from a sequential set of process bubbles
   - Central transform: the central processing bubbles in a transform analysis type of data flow

- In our example, the record (build) order process is the central process

Steps continued

3. Redraw the data flow diagram (DFD) with
   - The inputs to the left
   - The central transform process in the middle
   - The outputs to the right
   - The parent process (top level – e.g. “Create new order”) above the central transform process (e.g. “Build order”)
   - See next slide

FIGURE 9-16
Rearranged view of the Create new order *DFD.*



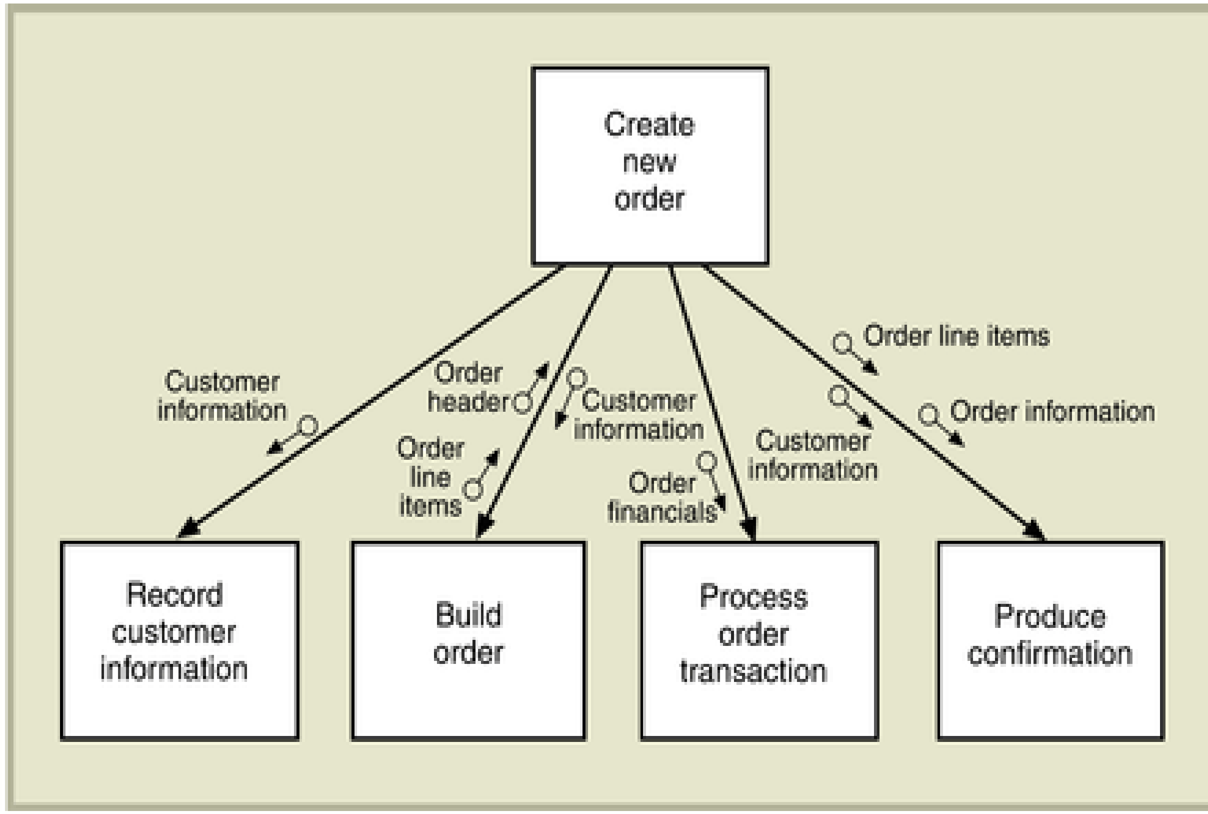
Steps continued

4. Generate the first-draft structure chart including data couples (directly from our rearranged DFD on the previous slide)

- See next slide

FIGURE 9-17

First draft of the structure chart.

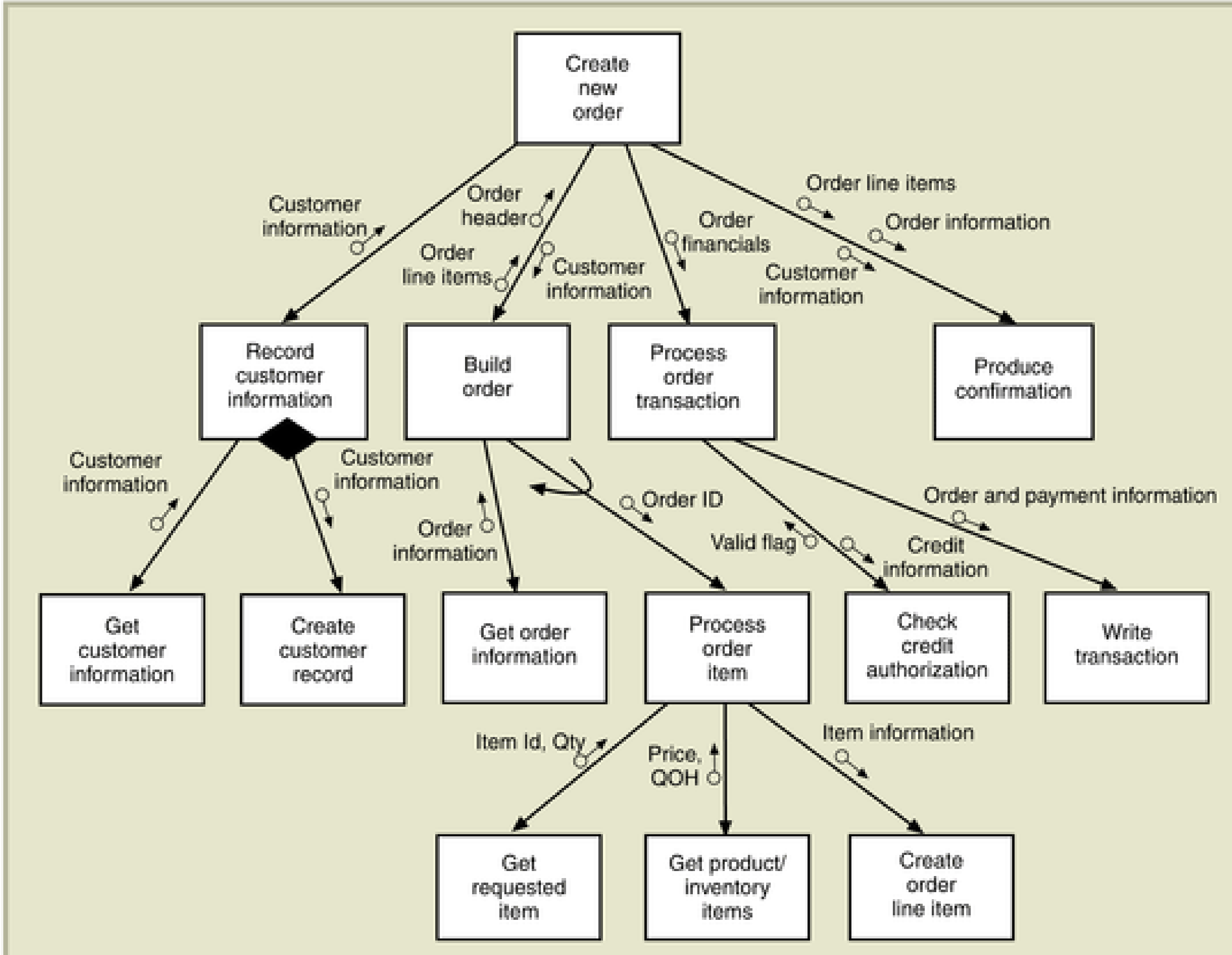


Steps continued

5. Add other modules as necessary to
   - Get input data via the user-interface screens
   - Read and write to the data stores
   - Write output data or reports

- These are lower level modules (utility modules)
- Also add data couples
- See next slide

FIGURE 9-18
The structure chart for the create new order program.



Final Steps

6. Using structured English or decision table documentation, add any other required intermediate relationships (e.g. looping and decision symbols)
7. Make the final refinements to the structure chart based on quality control concepts (to be discussed)

Combining the top-level structure chart with the chart developed by transaction analysis

- Basically “glue” the diagram we made (high-level) using transaction analysis on top of the more detailed diagram (for lower-processing) we just made using transform analysis
- See next slide

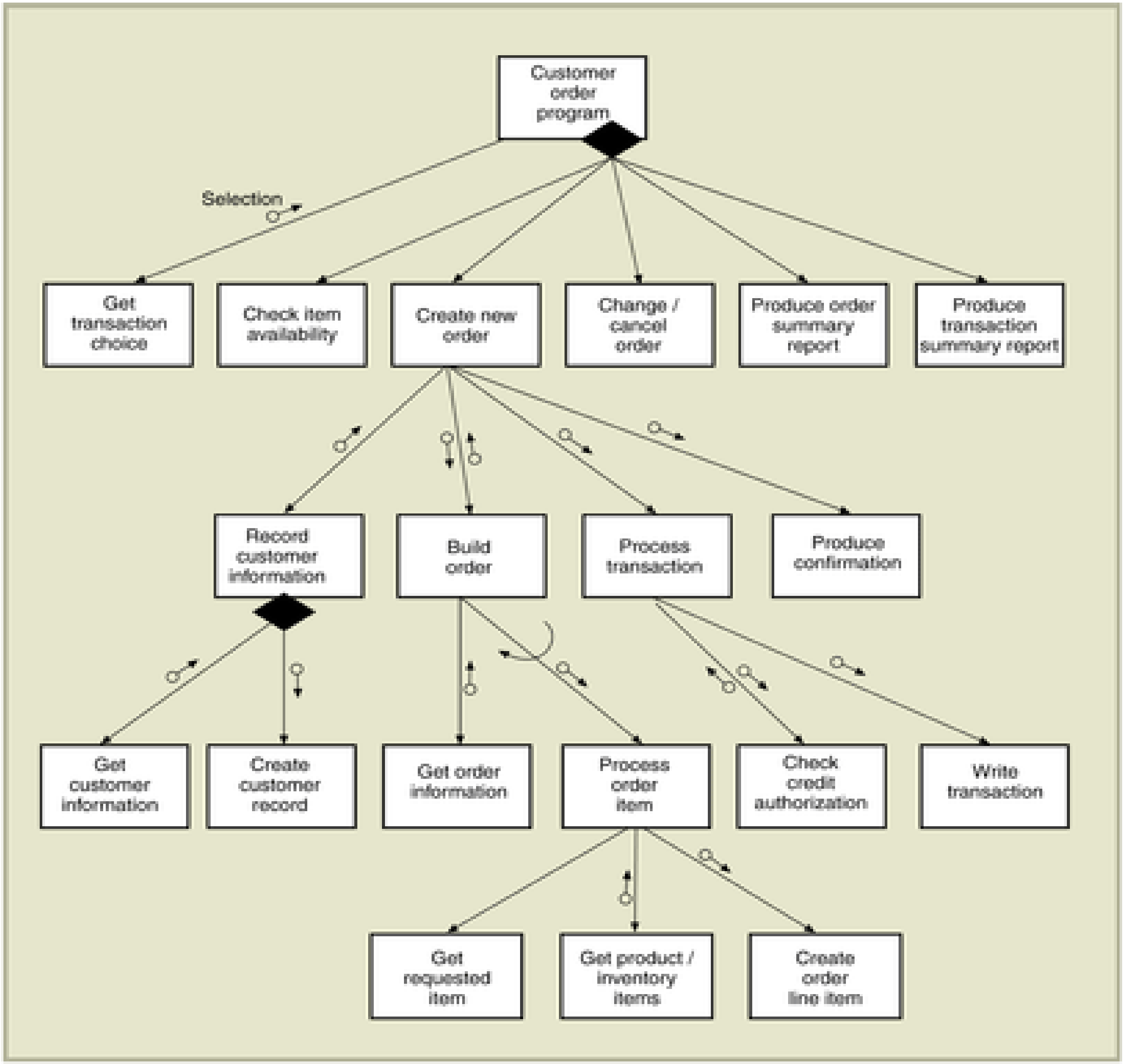


FIGURE 9-19
Combination of structure charts (data couple labels are not shown).

4 Steps

1

Create a structure chart based on the main item,
NOT programming requirements

2

Modify structure chart based on the programming requirements.
Top-down design.

3

Write code to implement the top-down design.
As you mature this will be skipped by going to step 4.

4

Improve readabililty of the code by incorporating functions.