

Lecture No. 9

4.2 Typical Processes

Now we shall discuss processes which are typically modeled using data flow diagrams. These processes transform data in one or the other way but these are found in almost all the automated systems. Following are the examples

- Processes that take inputs and perform certain computations. For example, Calculate Commission is a process that takes a few inputs like transaction amount, transaction type, etc and calculates the commission on the deal.
- Processes which are involved in some sort of decision-making. For example, in a point of sales application a process may be invoked that determines the availability of a product by evaluating existing stocks in the inventory.
- Processes that alter information or apply a filter on data in a database.

For example , an organization is maintaining an issue log of the issues or complaints that their clients report. Now if they want to see issues which are outstanding for more then a weeks time then a filter would have to be applied to sort out all the issues with Pending status and whose initiation date is a week old.

- Processes that sort data and present the results to users. For example, we pass an array of arbitrary numbers to a QuickSort program and it returns an array that contains the sorted numbers.
- Processes that trigger some other function/process

For example, monthly billing that a utility company like WAPDA, PTCL generates. This is a trigger that invokes the billing application every month and it prepares and prints all the consumer bills.

- Actions performed on the stored data. These are called CRUD operations and described in the next subsection

CRUD Operations

These are four operations as describes below

- **Create:** creates data and stores it.
- **Read:** retrieves the stored data for viewing.
- **Update:** makes changes in an stored data.
- **Delete:** deletes an already stored data permanently.

4.3 Adding Levels of Abstraction to Data Flow Modeling

As we have already described that in data flow modeling only those processes can be expressed that perform certain processing or transformation of information. Now the question arises how far these processes need to be expressed? As a single process like CalculateCommission as described in the above section, can be described in sufficient detail such that all of its minute activities can be captured in the data flow diagram. However, if we start adding each bit of system functionality in a single data flow diagram, it would become an enormously large diagram to be drawn on a single piece of paper. Moreover, requirement analysis is an ongoing activity in which knowledge expands as you dig out details of processes. Therefore, it may not be possible for an

analyst to know each bit of all the processes of the system from the very beginning. Keeping the complexity of systems in view, data flow modeling technique has suggested disseminating information of a system in more than just one levels of abstraction. What are these levels, please see below for a discussion

Context Level Data Flow Diagram

In a top-down system analysis, an analyst is required to develop high level view of the system at first. In data flow modeling, this high-level view is the Context level data flow diagram. In this diagram, system's context is clarified such that all the external agents or entities with which the system interacts are captured. It captures the details of what information flows between the system and these external entities, and what outputs are generated against inputs from these external agents and so on. So, the analyst probes out all the external agents that may involve persons, organizations or other systems who directly interacts with this system and their specific involvement in the system. At this level, systems internal details are not exposed, as we want to see system behavior as a black box.

Detailed Data Flow diagrams

Once context of a system has been captured using context level diagram, the analyst would expand his activities and start digging out system's internal details. Therefore, the same context level diagram is further expanded to include all major processes of the system that make up system functionality. So, instead of portraying system as a black box entity, the analyst would add processes that deal with the external agents and produces certain outputs. This is level one of a data flow model.

In level two of data flow model, instead of refining the previous levels further, we take one process from the level one diagram and expands it in a level two diagram. Hence, a level one diagram that depict the whole system, may be expanded to more than one level two diagrams each of which describes exactly one process in detail which were listed in level one diagram as simply an oval (process or transform).

This process may continue to any level of details as the analyst can conveniently captures. Where diagram at a specific level is a refinement of one of the processes listed in a previous level. By adding levels of abstraction to a data flow diagram, it becomes natural for a software engineer or a requirement analyst to readily express his knowledge about the system in an appropriate level of data flow model that corresponds only to a specific set of functionality.

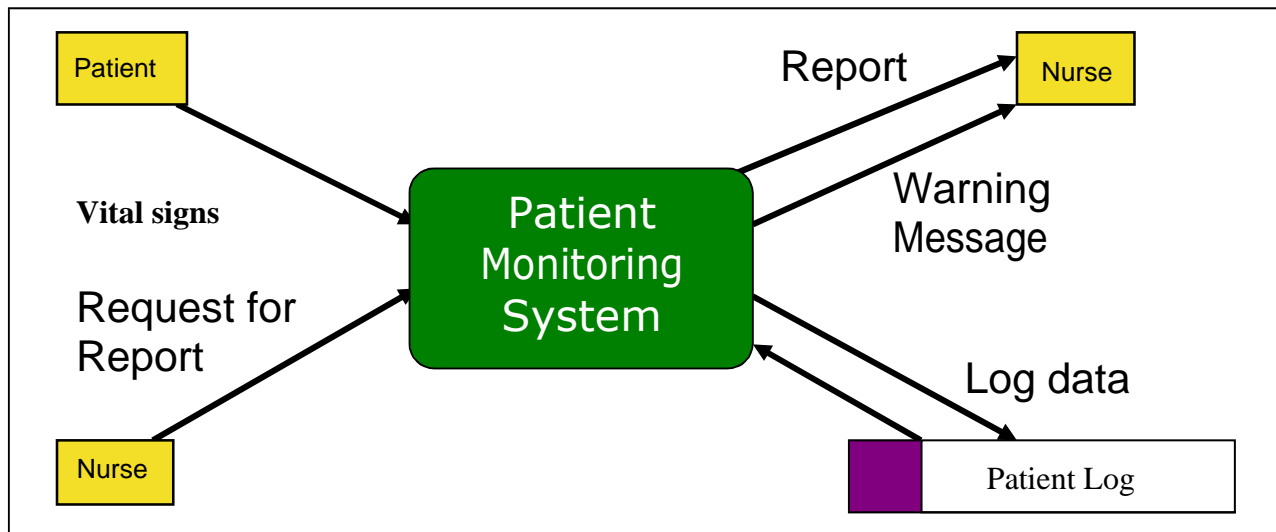
It should be noted here that the number of external agents and their inputs to the system and the outputs that the system would return to them, should remain the same throughout different levels of a data flow model. It should be considered a mistake if context level diagram contains three external agents, which are providing two inputs each, and getting one output in return but at level one, we add one more external agent or input or the outputs. This would make level one model inconsistent with the context level diagram. This is true for any level of data flow model. For instance, at level two the number of external agents, inputs and outputs shown in (all of level two) diagrams should match exactly with the external agents, inputs and outputs shown in level one diagram. Therefore, disseminating information at an appropriate level of abstraction with the additional check of inter-level consistency makes data flow modeling a very powerful domain-modeling tool. After this discussion, we shall give the reader an example in

which a system is modeled using data flow modeling technique where three levels of abstraction have been developed.

Patient Monitoring System – A Data Flow Modeling Example

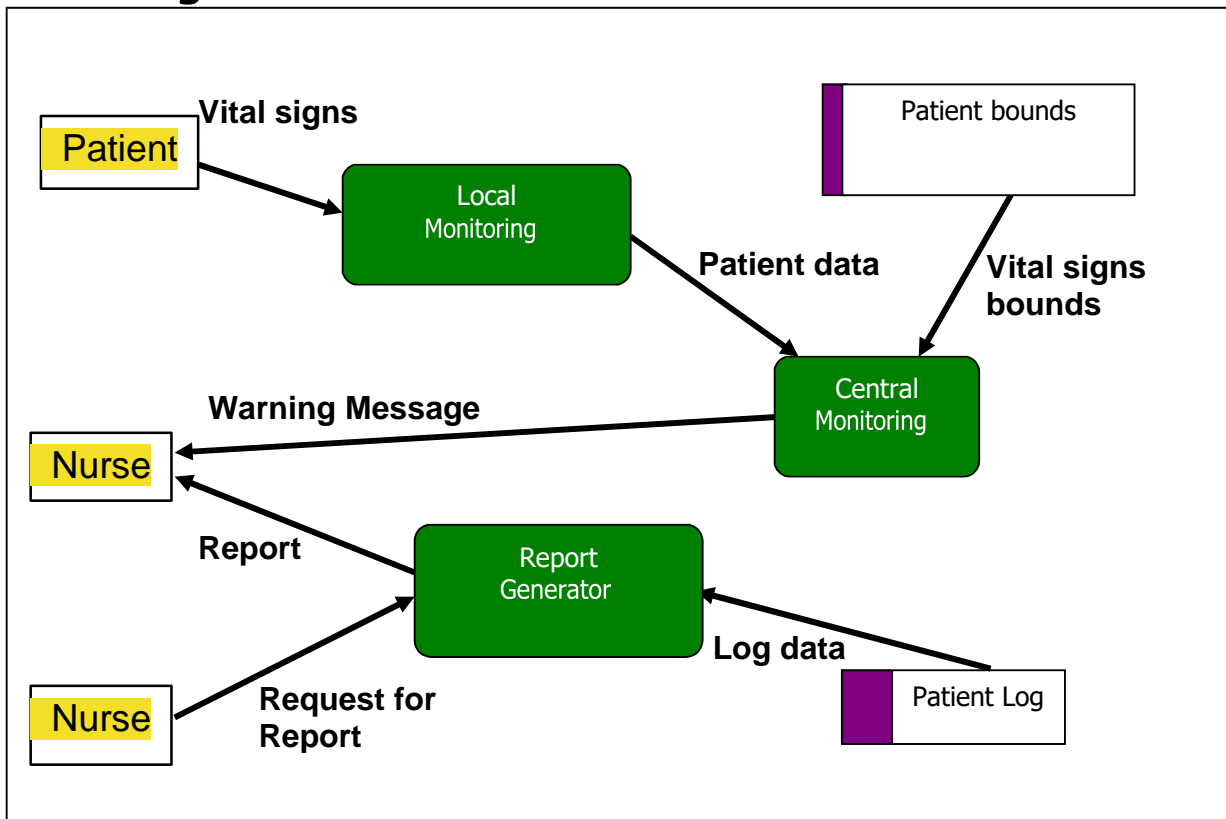
Context Diagram

Following is the 0-level or the context level data flow diagram of the Patient Monitoring System. In this data flow diagram, three external entities (users) are interacting with the centralized system. Point to note here is that in this context level diagram, only one process or transform takes place that is the Patient Monitoring System itself. A patient's vital signs are transmitted to this system which may invoke a warning message to the nurse if these signs fall into the critical range. Nurse may request for a report, which the patient monitoring system retrieves from the patient log, and return it to the nurse again. In this manner the 0-level data flow diagram describes the context of this system.



In order to see detail processes involved in Patient Monitoring System, a level 1 data flow diagram will have to be made. In the following, we are providing level 1 data flow diagram which is a refinement of the level-0 data flow diagram.

Patient Monitoring System – Level 1 Data Flow Diagram

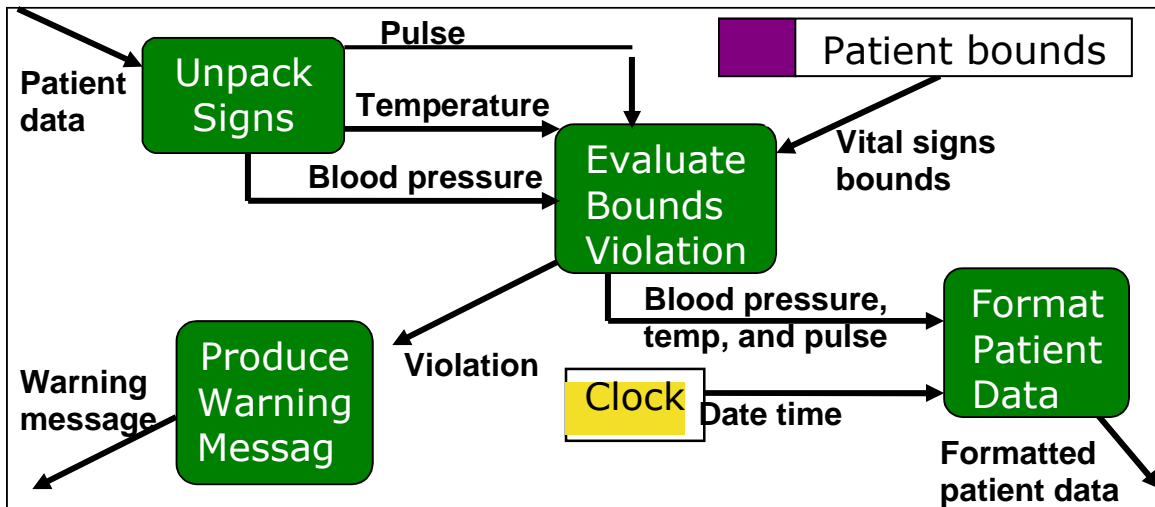


Level 1 data flow diagram is the refinement of the context (0-level) data flow diagram. All the external entities are the same (Nurse, and Patient), however, the process of 'Patient Monitoring System' is further elaborated by the three processes Local Monitoring, Report Generator, and Central Monitoring. The Local Monitoring process transforms vital signs that it receives from Patient entity into Patient data and passes this information to Central Monitoring process. Central Monitoring process retrieves vital signs bounds and compares Patient data and it may generate Warning message if the Patient data goes out of normal Vital signs bounds. A nurse may request for a report, in response the Report Generator process retrieves Log data from Patient Log, generates the report and displays it back to the nurse.

It should be noted here that this level 1 diagram is a further refinement of level 0 diagram such that the underlying system is the same but processes which were hidden in level 0 are represented in this diagram.

A further refinement of this model is also possible if we expand any of these three processes to capture further details. For example, the Local Monitoring process may further be expanded to capture detailed activities involved in the monitoring process. Following is level 2 diagram of Central Monitoring process.

Central Monitoring System – Level 2 Data Flow Diagram



In the above level 2 data flow diagram, Patient's data is sent to the Unpack Signs process which unpacks it and send Pulse, Temperature, and Blood pressure to the Evaluate Bounds Violation process. This process retrieves Vital signs bounds information, compares it with unpacked patient data and sends a violation sign to the Produce Warning Message process upon an out of bound result of the comparison. The patient data is sent to Format Patient Data process that generates the formatted patient data to be maintained against patient profile. In this manner we elaborated the patient monitoring system up to three levels describing different details at each level. In a similar manner, other two processes in level 1 DFD could also be expanded in their respective level two diagrams in order to describe their functionality in more detail.

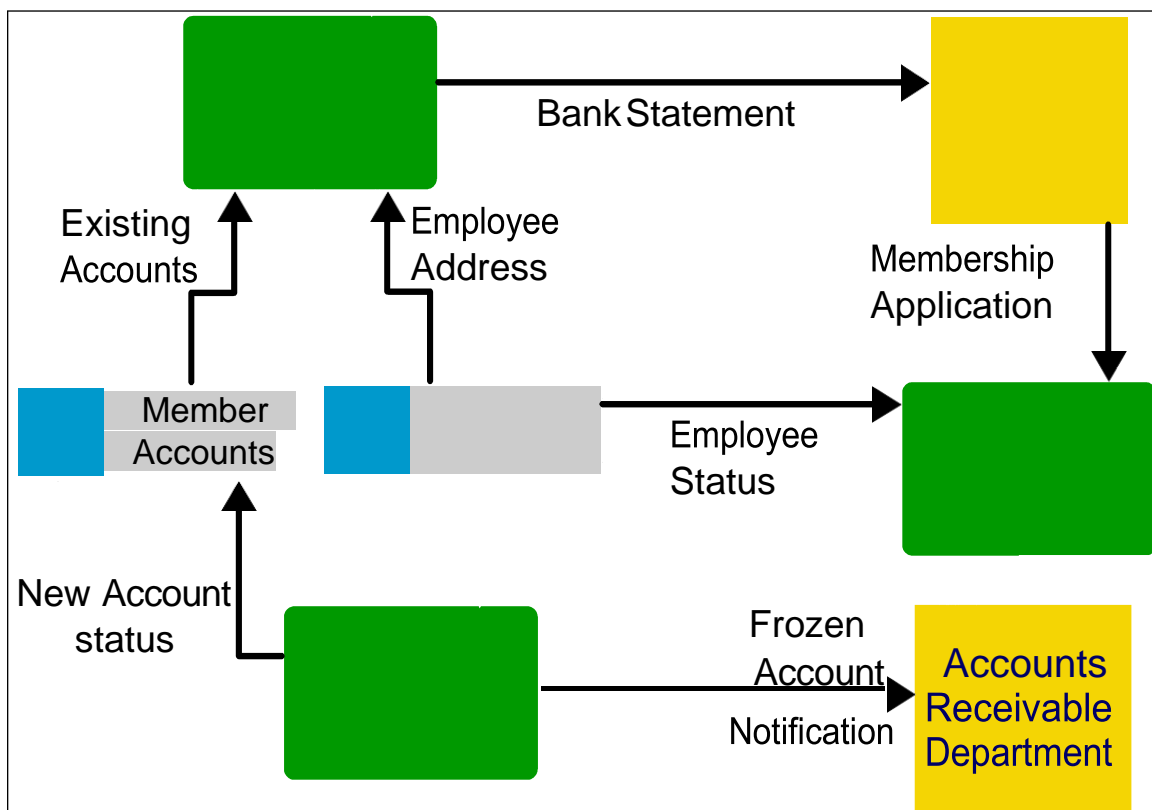
By going through this example, the reader would have learnt how data flow modeling technique helps in understanding domain of a system at different levels of abstractions. In the following sub-section, we shall describe common mistakes that the people do while preparing data flow diagrams.

4.4 Common Mistakes in Data Flow Diagrams

In the following data flow diagram, an accounting system has been described. Three processes are given Generate an Employee Bank Statement, Create a New Member Account, and Freeze Member Account. There are two external entities shown in this diagram Employee and Accounts Receivable Department. The three processes described in this diagram have associated problems. Can you guess these problems?

If you look at the arrows going inside each of these processes and coming out of them, you will observe some peculiarity.

In fact, here we can apply the source and sink analysis that we studied in the last lectures. What does the source and sink analysis suggest? It suggests that in order to check completeness of a requirement, evaluate the sources as well as the sinks of the requirements. Applying this knowledge in this case, we observe the following mistakes



- There is no input for the process Freeze Member Account
- In a similar manner, the process Create a New Member Account does not produce any output.
- Similarly, Generate Employee Bank Statement process is having two inputs and an output but the question really is, do these inputs correspond to the output?

The Freeze Member Account process that does not have any input is an example of a requirement whose source is not known. Similarly, the Create a New Member Account process that does not produce any output is an example of a requirement whose sink has not been specified. Lastly, the Generate an Employee Bank Account process though have two inputs and produces an output but in order to generate a bank statement, all that is

needed is an account number and the time period for which the statement is required. If we analyze the inputs given to this process, we can observe that both of these inputs cannot help in generating the account statement. Therefore, these inputs are irrelevant to the output being generated by this process.

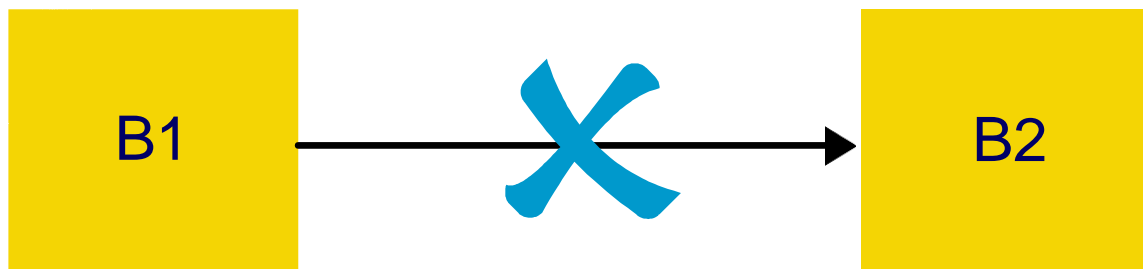
In the above mentioned example, it is evident that by applying the source and sink analysis we determined all the missing inputs and outputs to the processes of this diagram.

In the following subsection, we shall describe actions which are not only mistakes but illegal too for the data flow diagrams.

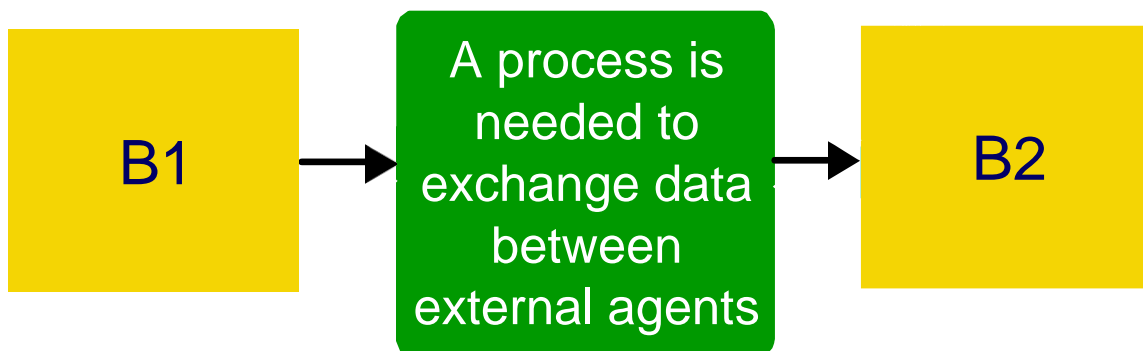
Illegal Data Flows

Directly Communicating External Agents

Following diagram depicts a scenario in which one external entity is directly communicating with another external entity. This form of communication is illegal to be shown in a data flow diagram.

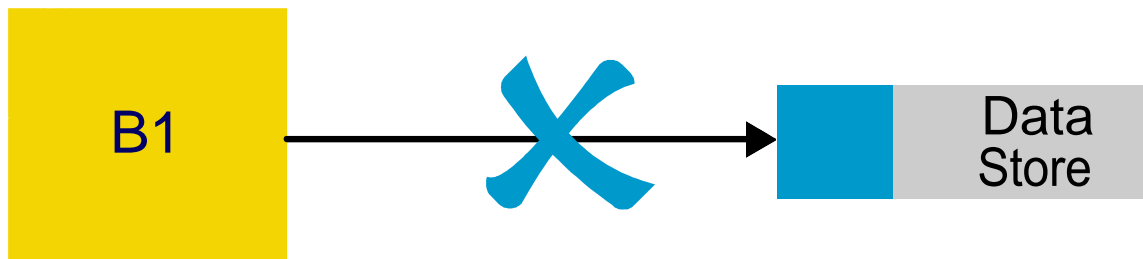


There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity. As we have already described that data flow diagrams should be used to depict processes that transform or process data. Simple data movement from one entity to another should not be described using data flow diagrams.

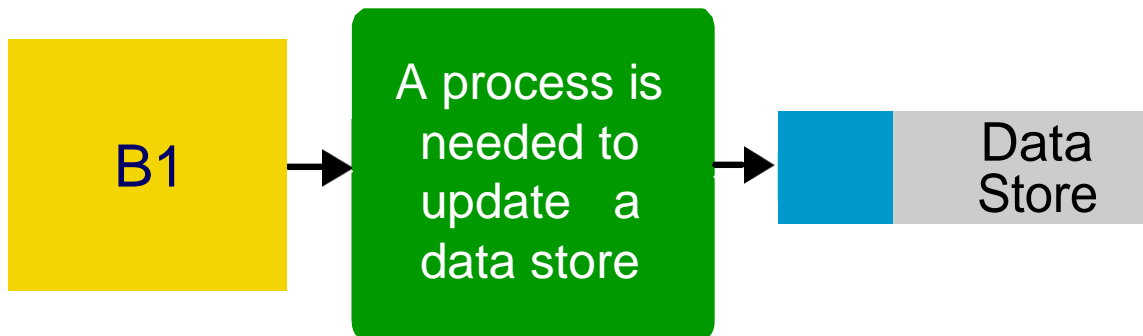


External Agent updating information in a Data Store

As we explained in the above case, a transform/process is needed between communicating entities. This is true even for an External Entity that wants to store/update some information directly in a data store, a transformation would be required.

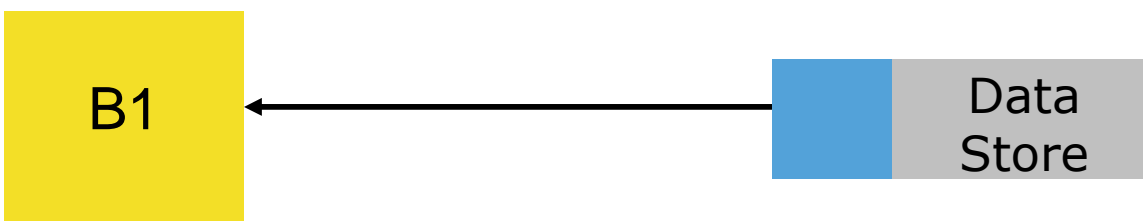


Therefore, a process should be inserted between the interacting entities (external agent, data store) that should store information received from the external agent after processing it.

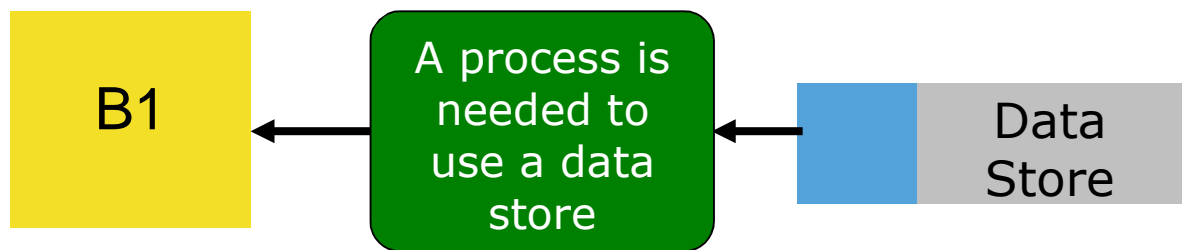


External Agent accessing information from a Data Store

Similarly, an external agent accessing information from a data store directly is also illegal.



Again a data transform/process is needed in this communication. It should be able to retrieve information from the data store and then pass it on to the external agent.



Copying data to a data store

In the following diagram, a data store is shown copying data directly to another data store. This is again illegal as there is not any intermediate process/data transform mentioned.



So, the correct method is again to use a data transform/process between the two data stores. It should retrieve data from one data store and after transforming that data, store it into another data store.

