**Chapter 7: Analysing System Needs**

Table of Contents

[Data Flow Diagrams 3](#_Toc63529884)

[Advantages 3](#_Toc63529885)

[Basic Symbols 4](#_Toc63529886)

[Context Diagrams 6](#_Toc63529887)

[Developing Data Flow Diagrams 7](#_Toc63529888)

[Diagram 7](#_Toc63529889)

[Child Diagrams 11](#_Toc63529890)

[Checking for Errors 13](#_Toc63529891)

[Logical and Physical Data Flow Diagrams 16](#_Toc63529892)

Now that we have gone through all of the problems of the current system and analysed user requirements, it is time to take a look at what the system we are trying to develop actually needs. We need to be able to represent the proposed system in such a way that the different stakeholders can understand how the system will work.

The output from the system analysis stage is what developers will be using to create things like ER diagrams and get started on developing the system.

## Data Flow Diagrams

Data flow diagrams allow us to graphically understand the flow of data through our system. We can see where the data is coming from, how it is processed and where the data goes. Thus, the data flow diagram shows system inputs, processes, system outputs and data storage.

With a data flow diagram, database engineers will be able to understand what data they need, what they need to store and what they do not need to store.

### Advantages

* Data flow diagrams allow us to avoid making technical commitments. We do not have to specify what exact technology we will be using, since it may still be too early to make those decisions, allowing us to instead use more generic terms. For example, instead of stating that we will be using Oracle SQL or MySQL, we simply have a symbol that indicates data storage. This symbol could mean absolutely any form of data store.
* We can understand how the different systems and subsystems are related. We will come back to this point and talk about it in more detail while discussing context diagrams.
* If we create a data flow diagram for the current system instead of the proposed one, one advantage is that it is easy to understand how the system works and communicate that information to users.
* A data flow diagram for the proposed system allows us to analyse the system as a whole, giving us an image of how big and complex the system really is and the many different intricate steps in between.

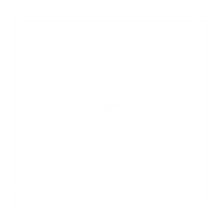
### Basic Symbols

Data flow diagrams consist only of four different symbols. However, we will be able to represent absolutely any system, no matter how complex, using just these four symbols.

It is not necessarily true that exactly these symbols are always used everywhere, but the meanings are the same.

#### Double Squares

These are used to represent external entities. External entities are bodies that are outside the scope of the system. Essentially, the system will not be modifying anything that belongs to that entity.

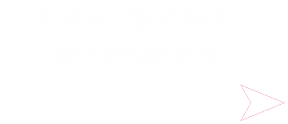


External entities are usually either a source or a destination of data. For example, if our system is working with accounting information, we will need to get that data from the accounts department. Thus, the accounts department is a source of data for our system, but our system is not actually able to modify anything in that department. This means the accounts department is an external entity. Similarly, every person related to the system, such as a student or a customer, is an external entity, since they provide information to our system without the system actually modifying the data in that entity.

External entities are named using nouns.

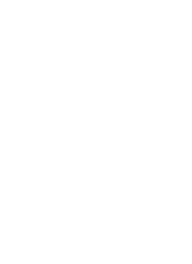
#### Arrows

These represent the movement of data from one point to another. Nouns are used with the arrows to show what data is flowing. We do not specify the format of the data, e.g. if it is sent using a physical medium, since we are not focusing on the technical implementation of the data flow.



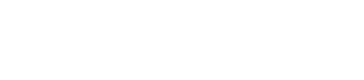
#### Rectangles with Rounded Corners

These are processes that transform the data in some way. The name given to the process should clearly describe what the process does. The process is also given an ID.



#### Open-Ended Rectangles

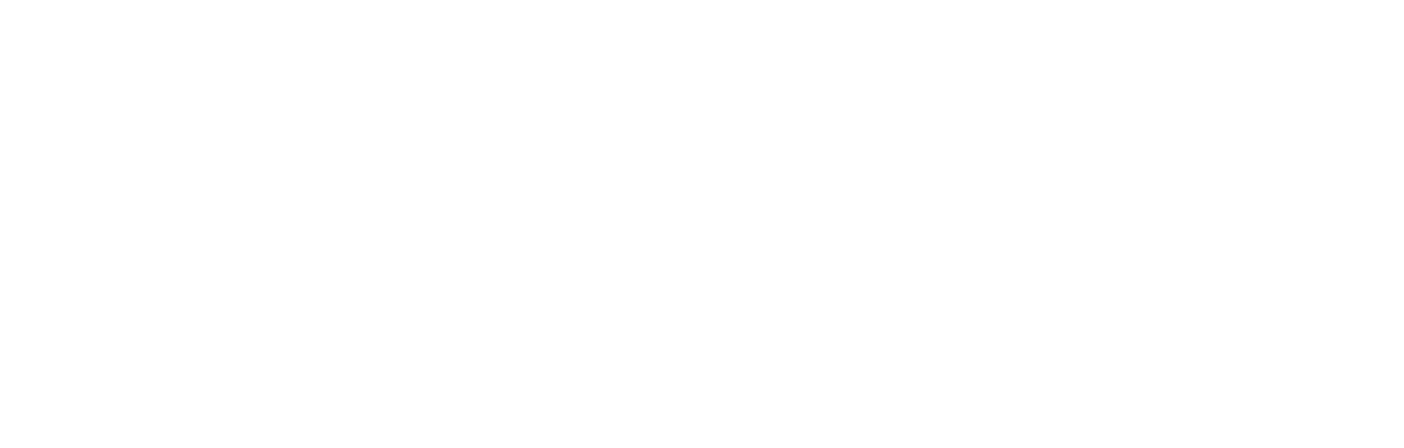
These represent data storage, where data is stored and can be retrieved from later on. The storage itself must be given a particular identification, which is conventionally , , and so on, to indicate different data storages. The data store is also given a name that tells us what data is being stored there.



The data store does not tell us what medium is being used. It could be anything from software like Oracle SQL to a physical filing cabinet.

Note that temporary storages, like a temporary computer file, are not shown on the data flow diagram.

### Context Diagrams



The context diagram is the highest level of a data flow diagram. It represents the entire system we are creating. As such, every single external entity is shown, arrows are shown indicating the data flow to and from those external entities, and a single process is shown. There is only one process, process , which represents the whole system. It has the same name as the system itself.

Note that we do not show any data storage in the context diagram and the internal processes of the system are not shown either. We also should not use a single arrow pointing in both directions, since the data flowing to a process is never the same as the data flowing out of it. If we take data from an entity and also give it back some data, we should use two separate arrows.

### Developing Data Flow Diagrams

The steps to developing a data flow diagram are:

1. Make a list of business activities and use them to determine the external entities, data flows, processes and data stores.
2. Create a context diagram that shows the external entities and data flows to and from the system, but do not show any detailed processes or data stores.
3. Draw Diagram 0. Show processes here, but keep them general. Show data stores as well.
4. Create child diagrams for each of the processes in Diagram 0.
5. Check for errors and make sure labels assigned everywhere are meaningful.

Note that there are two more steps that deal with something called a physical data flow diagram, but we will not be dealing with that. We are only working with the five steps above, which are part of something called a logical data flow diagram.

### Diagram

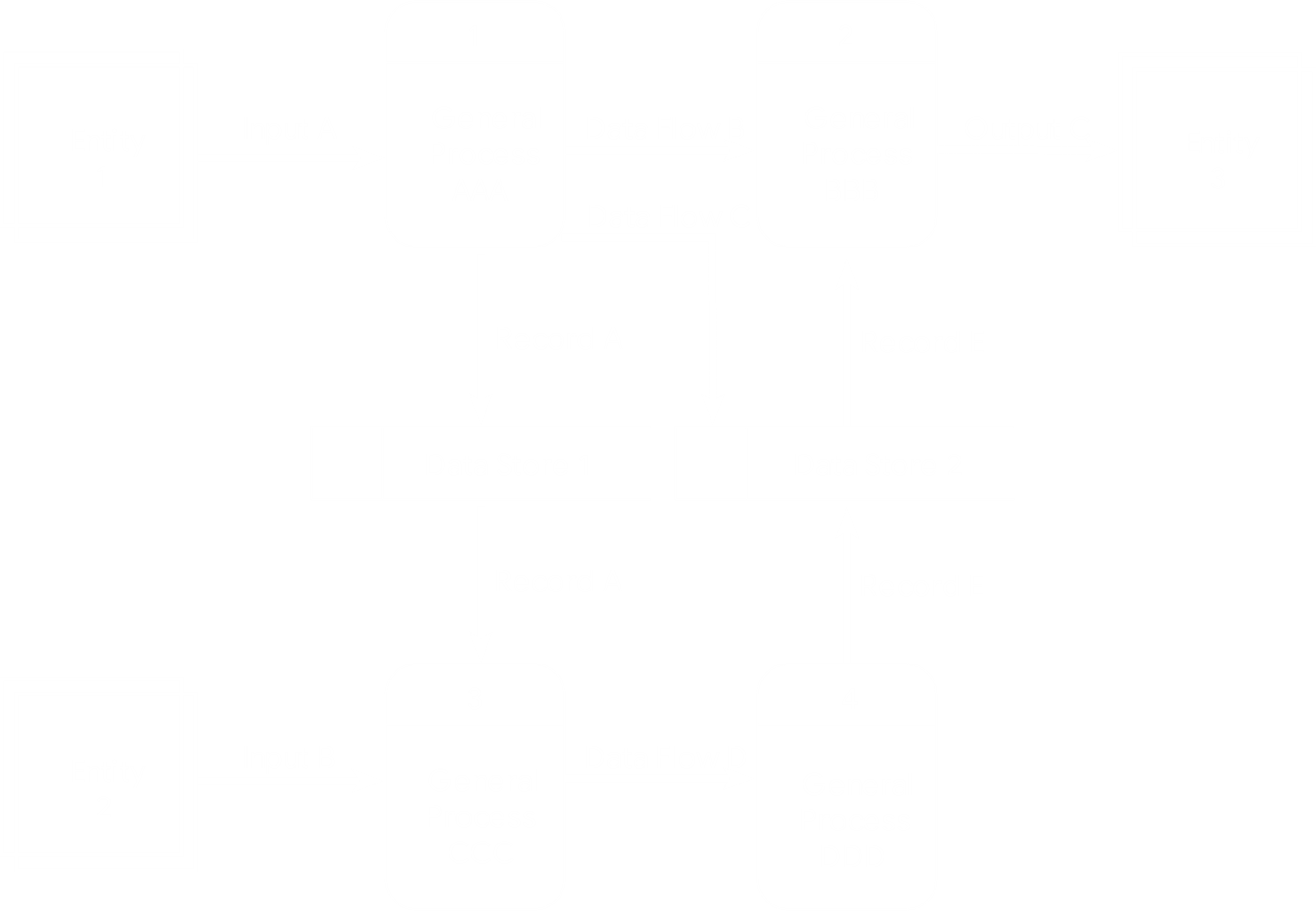


Diagram is an exploded version of the context diagram. It contains all of the main details of the system, the ones we do not show on the context diagram, hence the name of the diagram. Each process is numbered and major data stores and all external entities are included.

We need to remember that, although we are including all the major details of the system, we are not including every single detail. Diagram 0 can contain at most 9 processes, so we need to know where to stop. If we find ourselves needed more processes, we can combine a few together. Further details of each of the processes can be included in child diagrams if necessary. If we want to show more details of process 1 for example, we will explode it and show its details in another diagram, diagram 1. Thus, the hierarchy of data flow diagrams goes context diagram diagram 0 child diagrams.

Data flow diagrams are not necessarily sequenced diagrams, meaning the actual sequence of steps in the system need not be followed. Rather, we follow the sequence of data flow. We can work forwards, from input data flow sources, and backwards, from output data flow destinations.

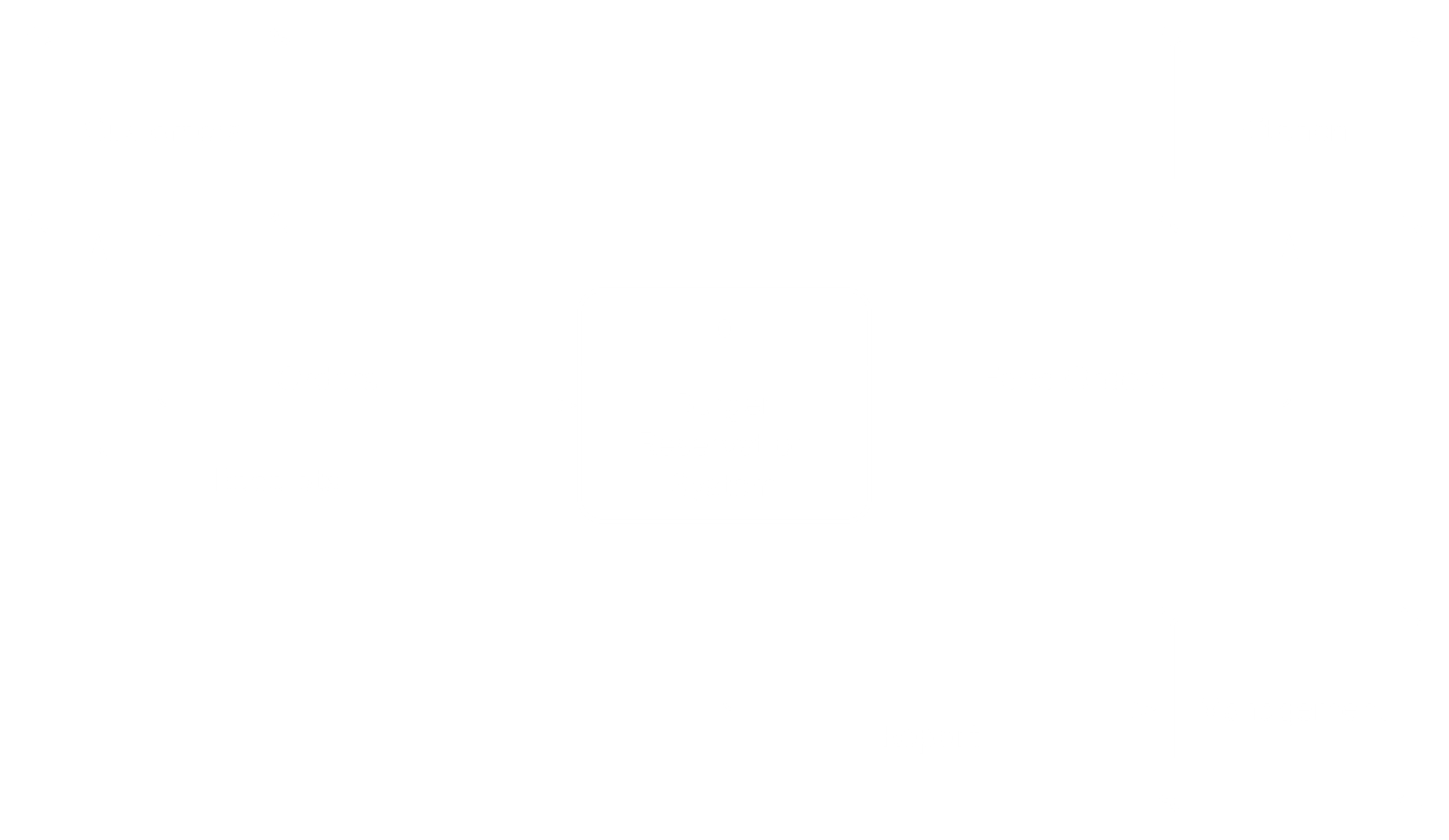
Example

Consider a burger joint where many people, especially students and faculty from a nearby university, eat. The restaurant uses an information system that takes customer orders, sends the orders to the kitchen, gives receipts to customers monitors goods sold and inventory and generates reports for management.

The first thing we need to do is identify the different parts and create a context diagram. The different parts are:

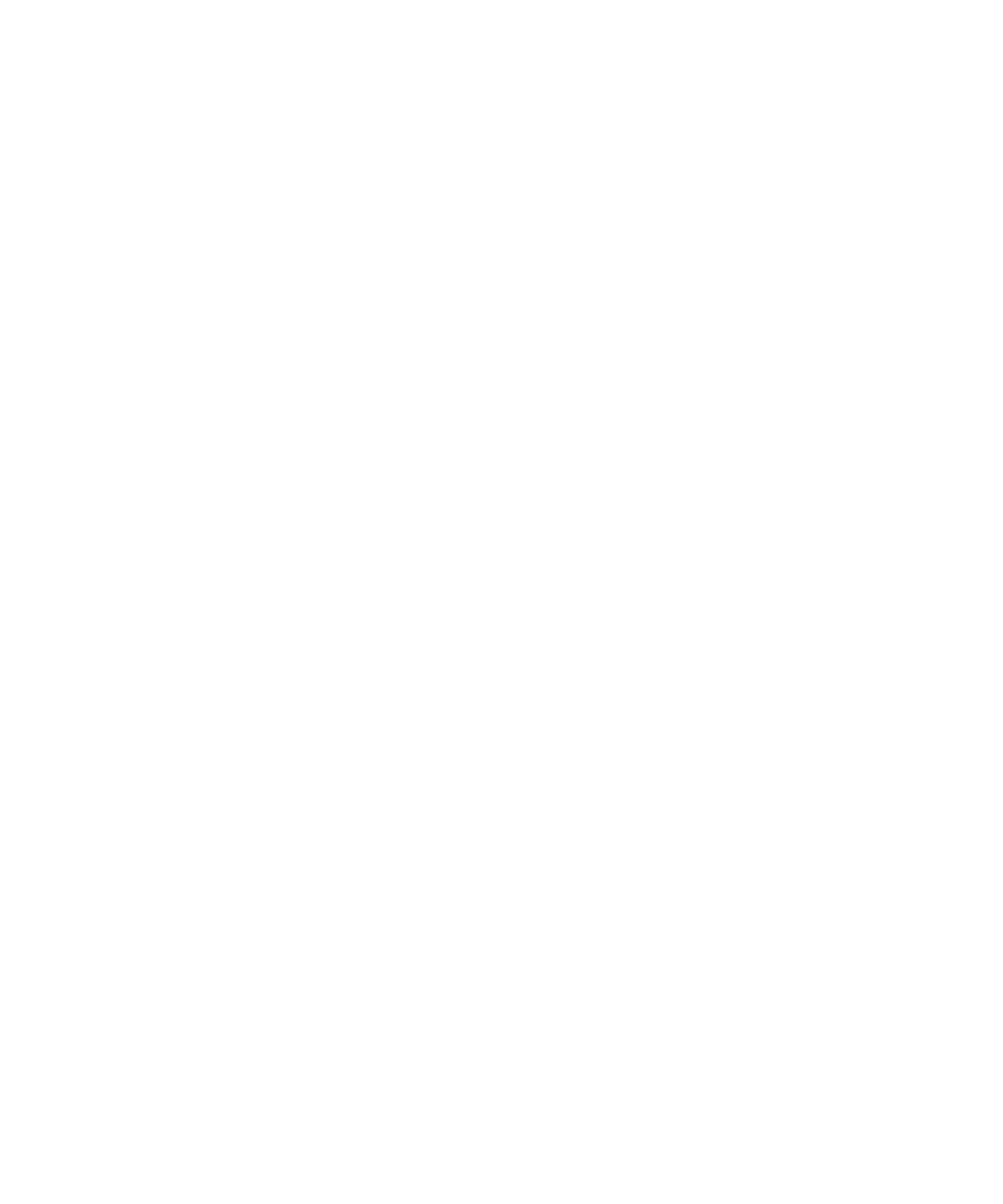
* The single process, process 0
* External entities
  + Customers
  + Kitchen
  + Management
* Data flows
  + Orders from Customers
  + Receipts to Customers
  + Food orders to Kitchen
  + Reports to Management

From this information, we can create the following context diagram:



Of course, in an actual restaurant management system, there will be several other parts, such as payment systems and suppliers. However, since we have not explicitly been told about these parts, we cannot include them in the context diagram.

Next, we must create diagram 0.



Notice that, in this diagram, we are showing every major feature as a separate process. The features that monitor goods sold and update the inventory, which were not shown in diagram 0, since they are being considered internal to the system, are shown here. They make use of data stores, which are also shown.

Regarding the generation of reports, since we have not been told explicitly what that data is, we need to consider data from both the data stores in our system. The report will be generated based on this data and sent to management.

Again, notice that we did not add any extra processes like payment systems or receiving supplies, since this information is not provided or indicated in the information we have been given, even if it may seem obvious to us that this information should be present. We also do not show things like food being sent from the kitchen to the customers, since that is not being dealt with by our system. We should not show any interactions between external entities.

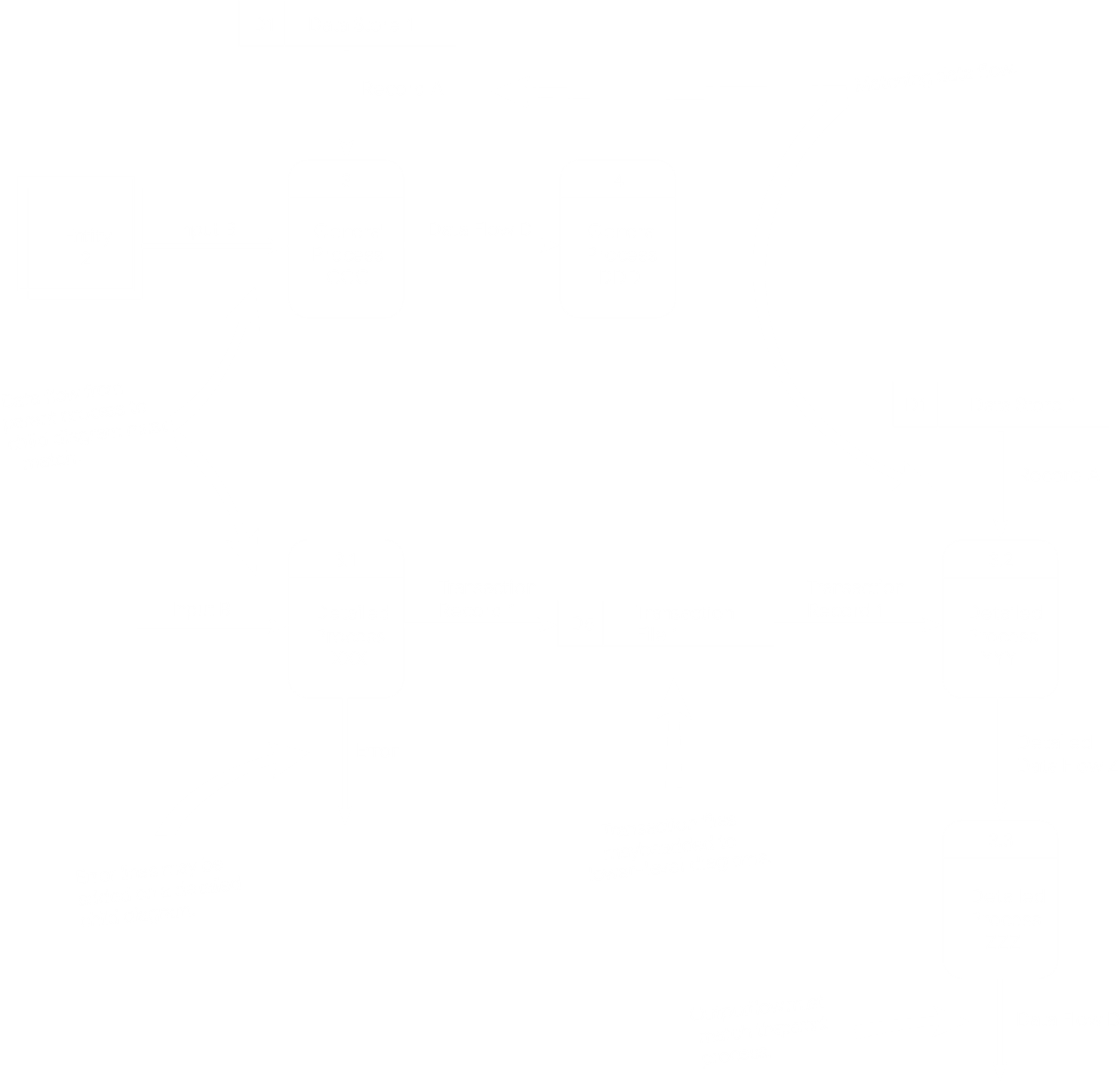
An essential piece of information to remember is that no process just takes data and passes it on. Every output from a process is different from the input. As such, the orders we put into process 1 is formatted in some way before being passed to process 2, 3 and the kitchen.

### Child Diagrams

Whenever we want to show internal details about a particular process, we will create a child diagram that has the same number as the parent process. In a sense, diagram 0 is also a child diagram.

Some things we need to remember about child diagrams are:

* A child diagram cannot produce output or receive input that the parent process does not also produce or receive.
* We can ignore external entities on child diagrams below diagram 0. Instead, we can just show arrows coming from and going to nowhere for external entities. It would not be wrong to mention them, but it is advisable to not do it.
* It is advisable to include any data stores that are in the parent process in the child diagram as well.
* It is alright to add a few of our own details that are not explicitly mentioned in the given scenario.
* If in case some process does not need to be exploded into a child diagram, since there is not more information we can add, that process will be called a primitive process.



Notice in the diagram above, Record A coming from a data store is simply copied as is from the parent process into the child diagram, but Input B, which comes from an external entity, and Data Flow D, which goes to process 4, only have their arrows shown.

There are also smaller processes and data stores in the child diagram that were not in the previous diagram, since they are internal to process 3.

Notice the way that these processes have been named, with reference to the parent process, as 3.1, 3.2 and so on.

A new part you will notice is that an arrow for errors is shown. This allows us to show any errors that may occur within process 3 that are not necessarily given as output. This will not be considered a new data flow.

### Checking for Errors

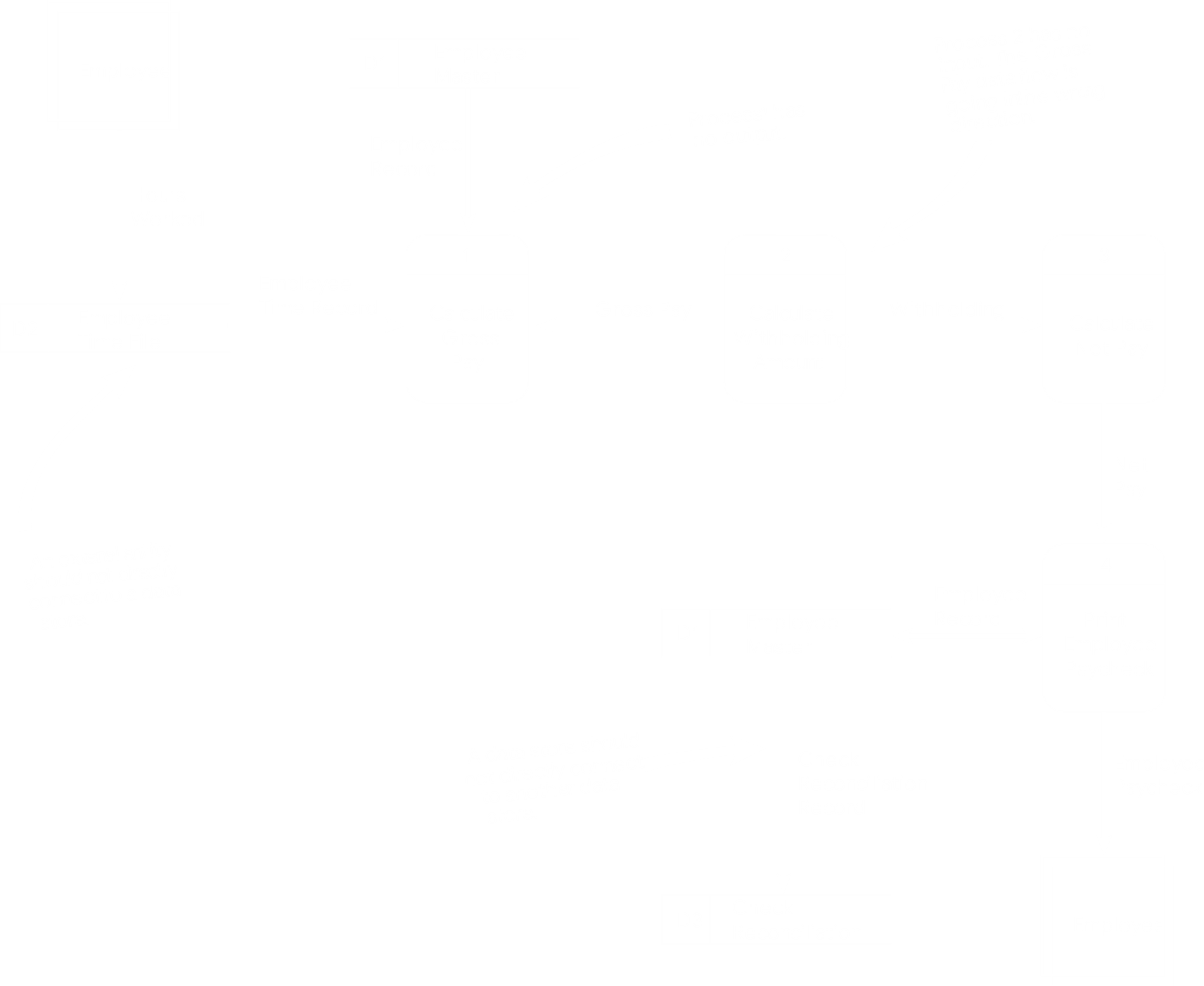
We need to be careful to not make the following mistakes while drawing data flow diagrams:

* Forgetting to include a data flow or pointing an arrow in the wrong direction – This can lead to situations where a particular process has no input and only produces outputs, called a miracle, or there are only inputs and no outputs, called a blackhole.

There may rarely be processes that genuinely do not have any inputs or outputs, but those are generally processes that are so irrelevant that we can simply ignore them. We do not really need to worry about that.

* External entities or data stores connected directly to each other – All data stores and external entities should communicate using some process in the system. It is possible for external entities to communicate with each other, as we have previously seen in the example, but those communications are not relevant to our system. Additionally, external entities should not directly communicate with data stores. Some process needs to be involved in between them.
* Incorrectly labelling processes and data flows
* Including more than nine processes in the data flow diagram – We should not do this since the diagram becomes too cumbersome and messy to understand. One way to avoid making the diagram messy is to have the same external entity in different places. For example, if there is some interaction with customers at the top of the diagram and some more at the bottom, we can show customers two different times. There is no rule that one entity can only be shown once. We can even do this with data stores, as long as we use the same ID and name.
* Omitting data flows
* Unbalanced decompositions, referring to child diagrams not having the same number of inputs and outputs as the main process.
* Bi-directional arrows – A process that is communicating with an external entity or data store should not have a single bi-directional arrow. Even if some input is being taken and some output is being given, two different arrows should be used. The only time this would be allowed is if the process stores some data and then retrieves that same data.

The diagram below is full of such errors:



## Logical and Physical Data Flow Diagrams

Logical data flow diagrams are concerned with how the business operates, not with how the system will actually be constructed. This is why we do not see things like specific data storage mediums in the diagrams we have drawn so far.

Physical data flow diagrams are similar to logical data flow diagrams, but there we would actually show details about the physical implementation of the system.

For example, in a logical data flow diagram, we would simply say that there is a system via which employees have to sign in when they arrive at the office. In a physical data flow diagram, we would mention what the actual method is, for example inserting a security card.

The advantages of having a logical data flow diagram include

* Better communication with users
* More stable systems
* Better understanding of the business by analysts
* Flexibility and maintenance
* Elimination of redundancy and easier creation of the physical model