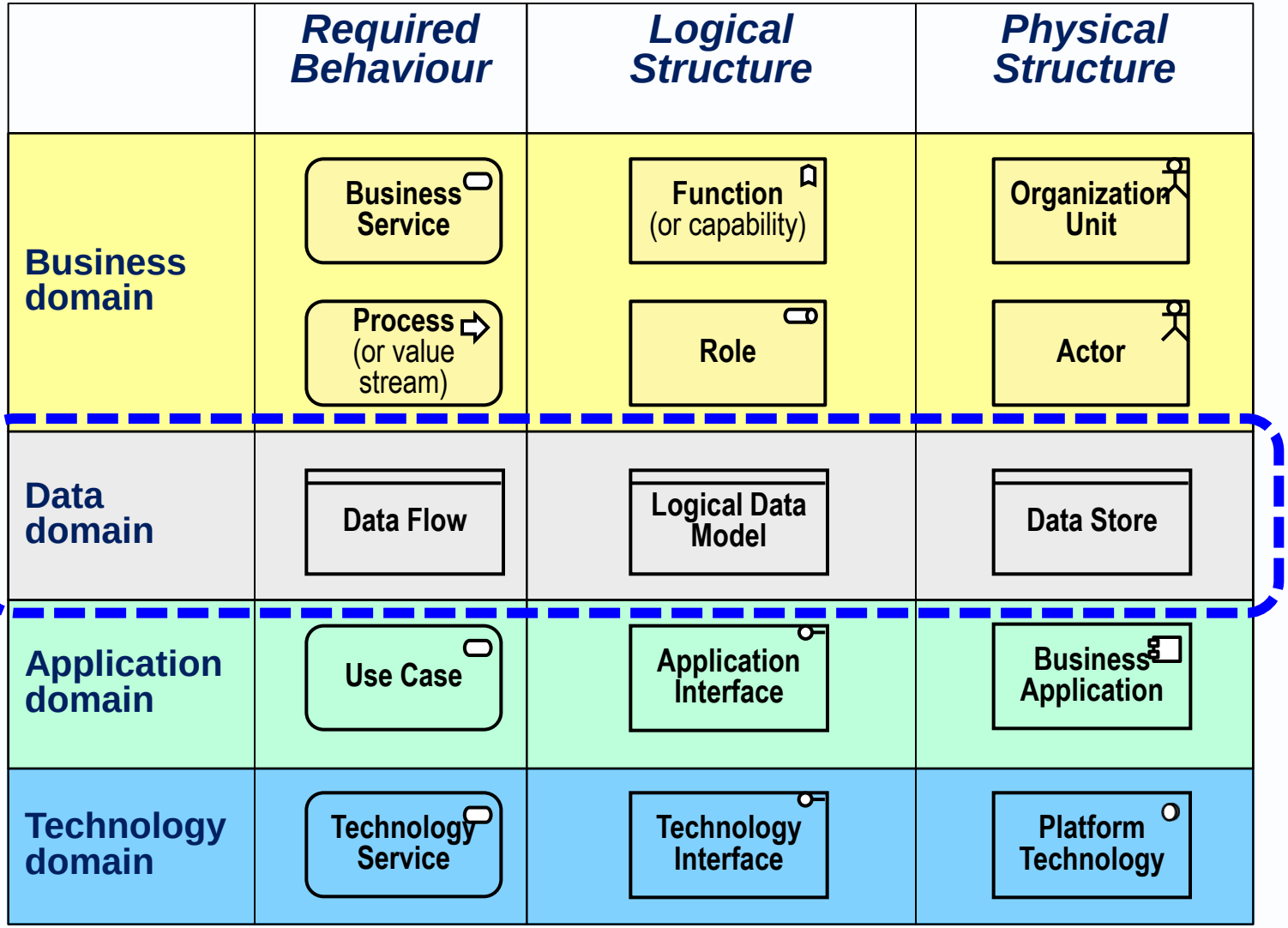
Avancier Methods (AM)

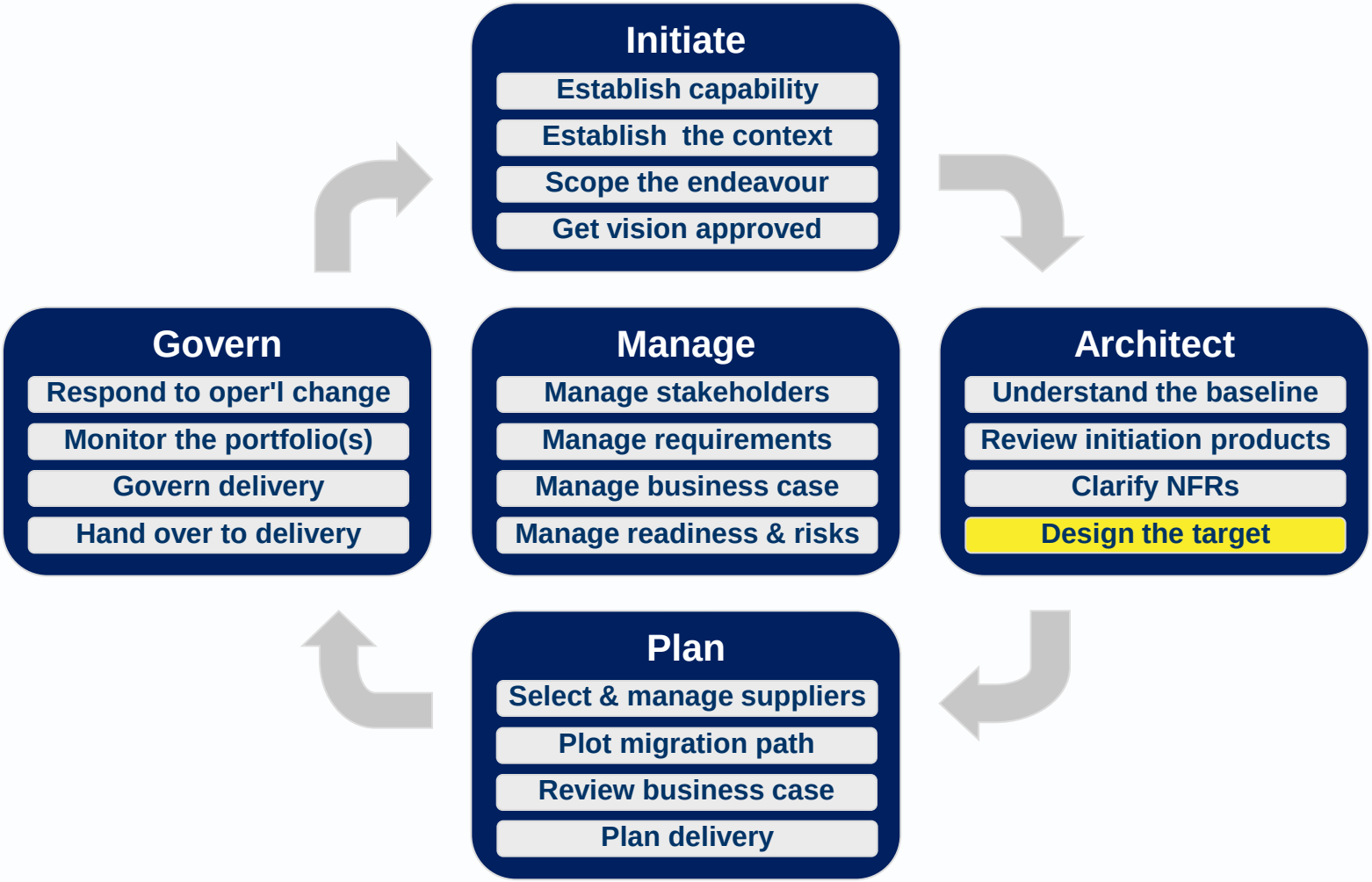
Data Architecture

Define data store (logical and physical)

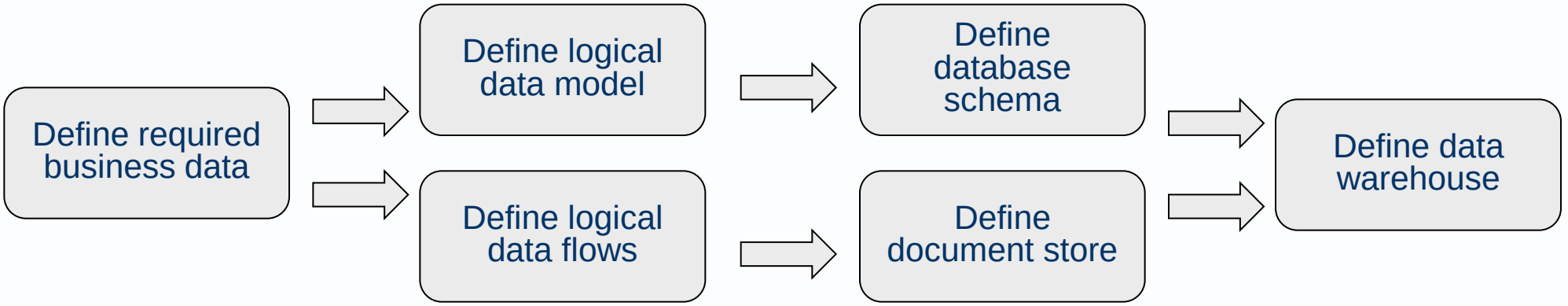
Which domain are we working in?



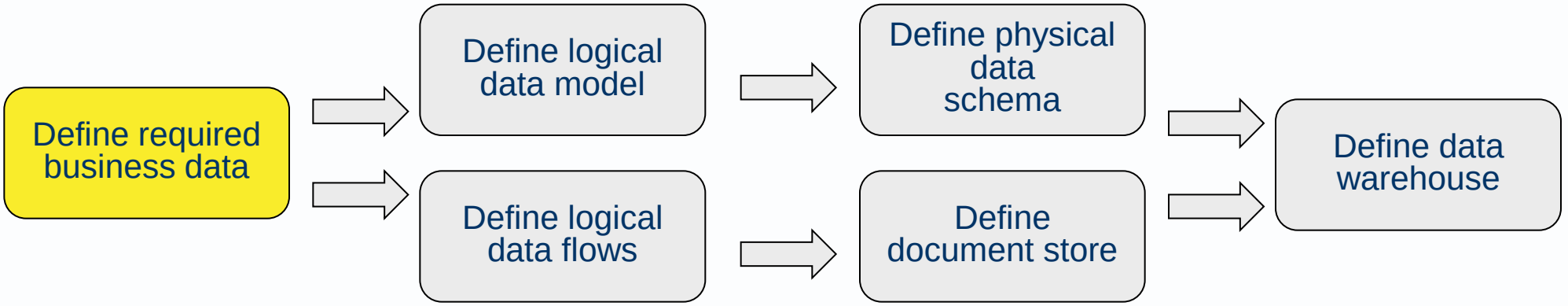
AM level 2 process



Define business data stores and flows



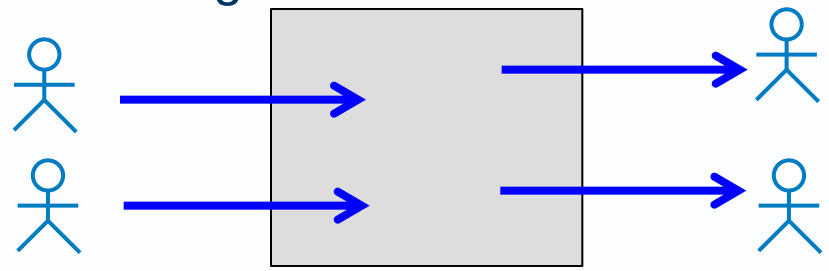
AM level 3 and 4 process: Define required business data



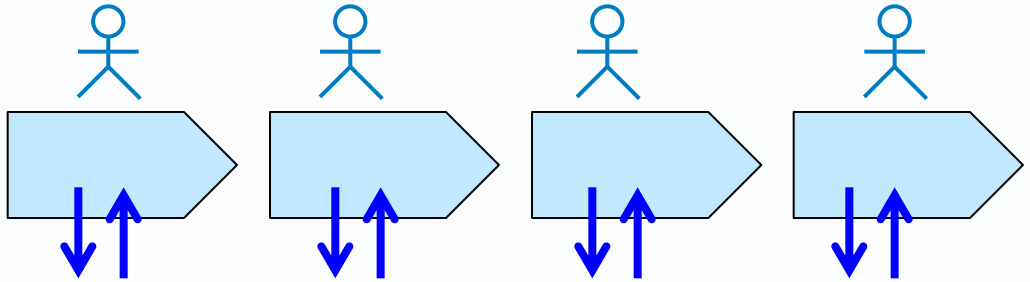
1. Identify where data is created and used
2. Define data created and used in business activities
3. Define data dictionary

Identify where data is created and used

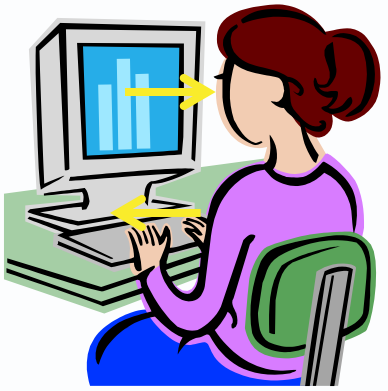
- Context diagram



- Value stream / scenario diagrams (showing OPOPOT activities)



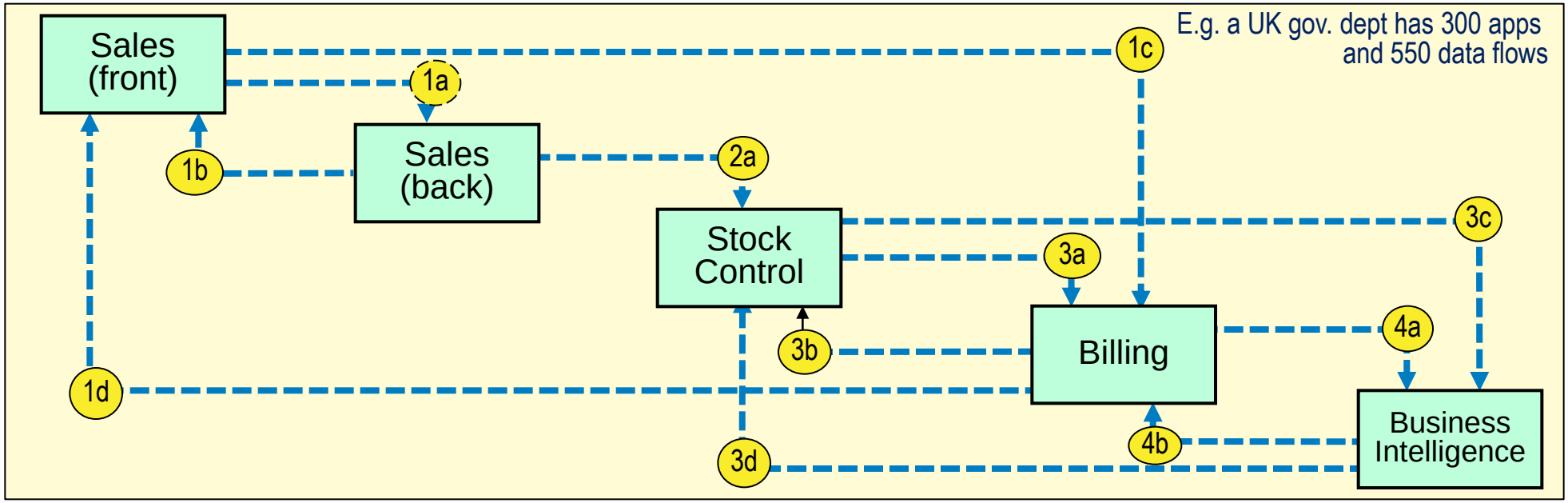
- Client devices and user interfaces



Data flow documentation (solution level)

Avancier

Data Flow (aka Application Communication) Diagram



Data Flow (aka Interface) Catalogue

Id	Flow Name	Source	Destination	Content
1a	Order entry	Sale (front)	Sale (back)	Ref. 999
1b	Order accepted	Sale (back)	Sale (front)	Ref. 999
2a	Notification	Sale (back)	Stock Control	Ref. 999

Data flow (aka Interface) catalogue

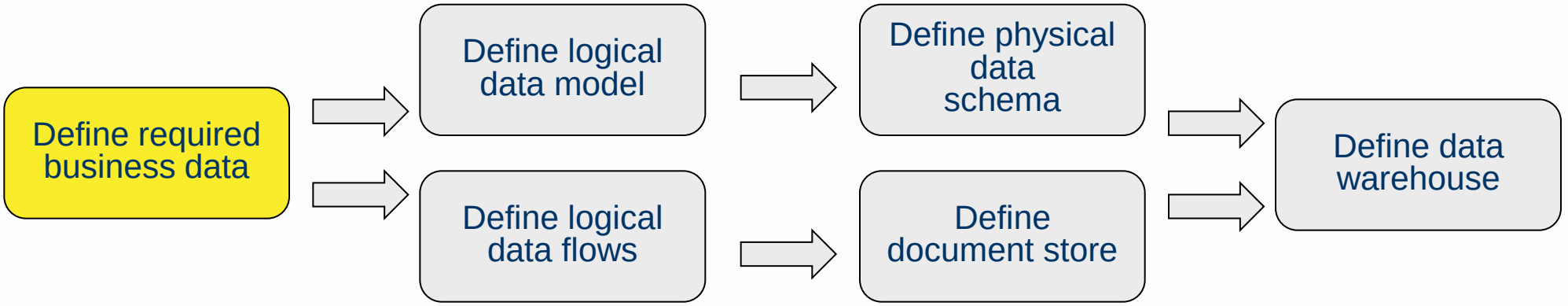
Avancier

- Catalog the key data flows

FLOW	Functional			Non-functional					Media	
Name	Source	Destination	Content	Frequency	Vol.	Confident.	Integrity	Avaialbility	Technology	Protocol
Order entry	CRM	Sales	Ref. 999	1K per day		High	Medium	24*7	Web	http
Order accepted	Sales	CRM	Ref. 999	1K per day		Medium	Medium	0900-1800	Web	http
Notification	Sales	Stock	Ref. 999	100 per day		High	Medium	24*7	Web	http

- Like many such illustrations, this shows what *could* be documented
- Understanding what is possible in theory is a precursor to deciding what to do in practice.

AM level 3 and 4 process: Define required business data



1. Identify where data is created and used
2. Define data created and used in business activities
3. Define data dictionary

Define data used

Avancier

Salesman wants

Customer Order History
- **Customer id**
- **Customer name and address**
- Orders Placed
  - **Order id**
  - **Order value**
  - Products Ordered
    - **Product type**
    - **Product amount**
  - Products Ordered End
- Order Placed End

Customer Order History END

Product manager wants

Product Demand Report
- **Product type**
- **Amount on hand**
- Products ordered
  - **Product amount**
  - **Order id**
- Products Ordered End

Product Demand Report End

Define data created and maintained

- Customer creates

Shopping Basket
Customer id
Order id
Order value
Products Ordered
Product type
Product amount
Products Ordered End
Shopping Basket

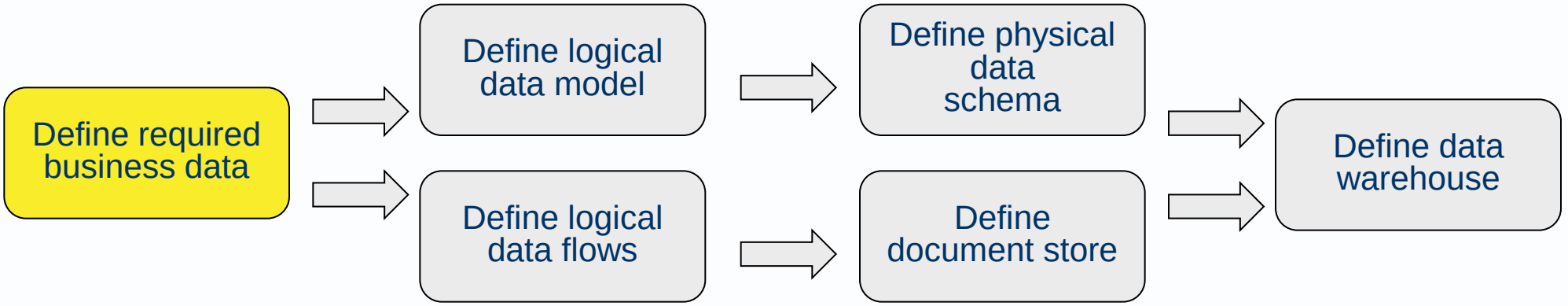
- The HR department maintains a spreadsheet of all employees

Human resources
Employee Number, Name, Role, Grade

- The sales manager has a card file with all salesmen in it

Salesman card file
Employee Number, Name, Commission Rate, Sales Area

AM level 3 and 4 process: Define required business data



1. Identify where data is created and used
2. Define data created and used in business activities
3. Define data dictionary

Data dictionary (solution level)

1. Define entities and items in I/O data flows
2. Define data that must be remembered for future activity
3. Define business rules associated with data items

Name	Facts	Constraints	Derivation rule
Currency Code	abbreviates **Currency**	is a three letter **String** in the range defined at ref. 999…	
Currency	denominates a **Value**		
Item Value	is an attribute of an **Order Item** is associated with] **Currency**	is a **Number** in the range 0 to 999	= **Product Amount Ordered** * **Unit Price**
Order Value	is an attribute of **Order** is calculated from **Item Values**	is a **Number** in the rang 0 to 9999	= sum of **Item Values** for an **Order** - **Discount**

Assign items to primitive data types (e.g. as in Java)

Avancier

Primitive data types

- Boolean
- Character
- Integer
  - Byte
  - Short
  - Integer
  - Long integer
- Floating point
- Double floating point

User defined data types

- Name (Character)
- City (Character)
- Order value (Integer)

Type	Contains	Default	Size	Range
boolean	true or false	false	1 bit	NA
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
double	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

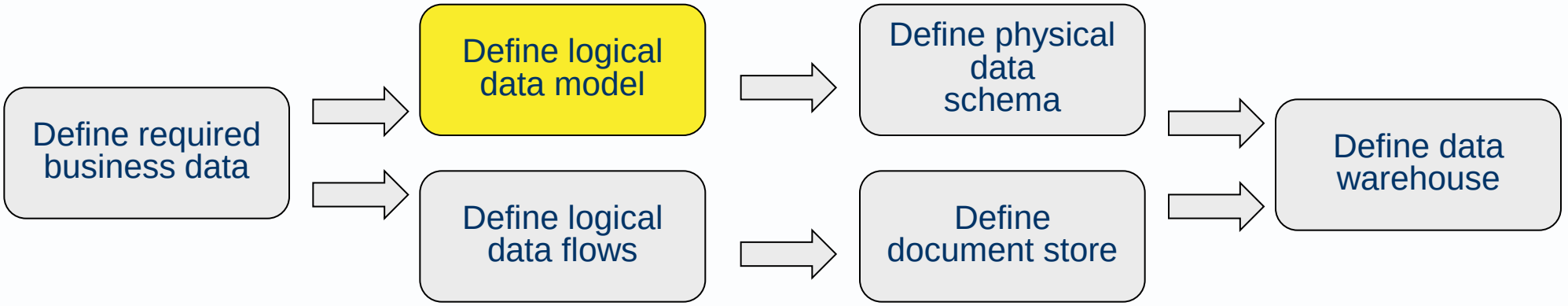
Complex data types

A source of many system integration problems

Complex data types (simple data structures)

- Date
  - DD
  - MM
  - YYYY
- Person
  - Title
  - First name
  - Last name
- Address
  - Address Line 1
  - Address Line 2
  - Address Line 3
  - City
  - County/State
  - Postcode

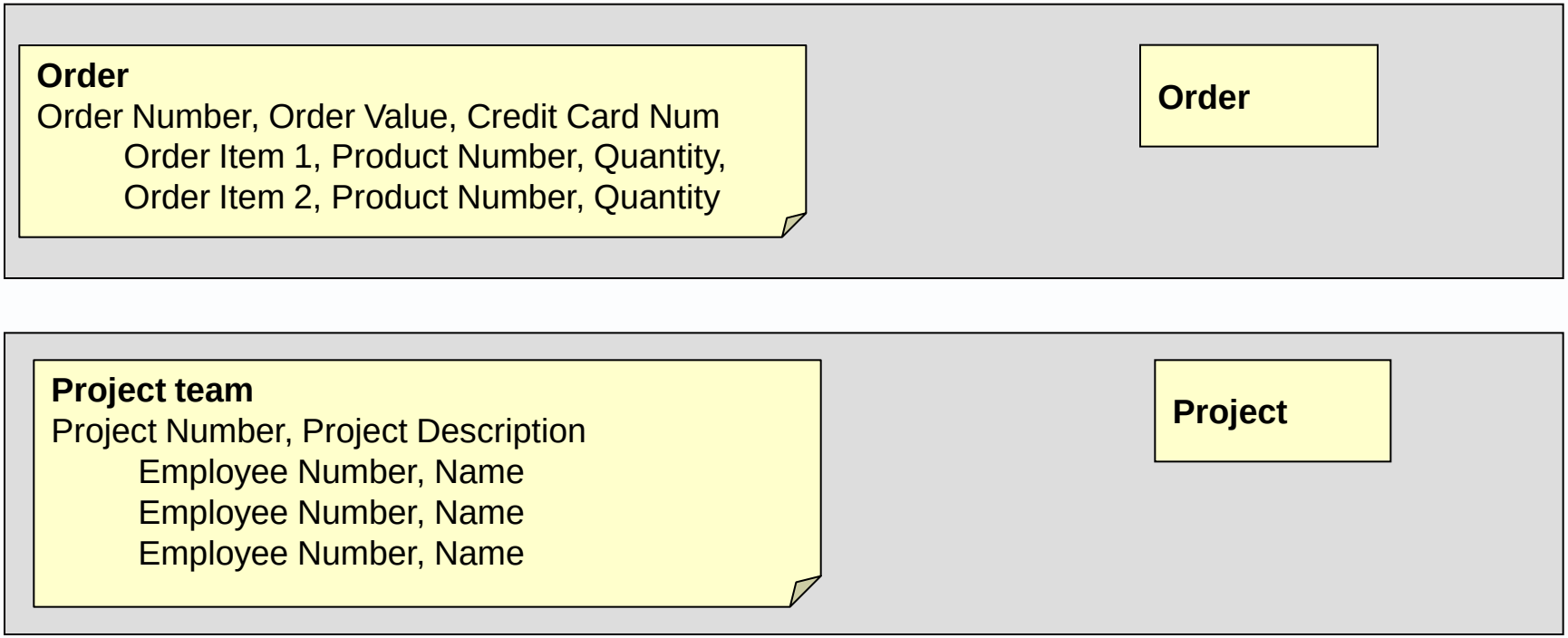
AM level 3 and 4 process: Define logical data model



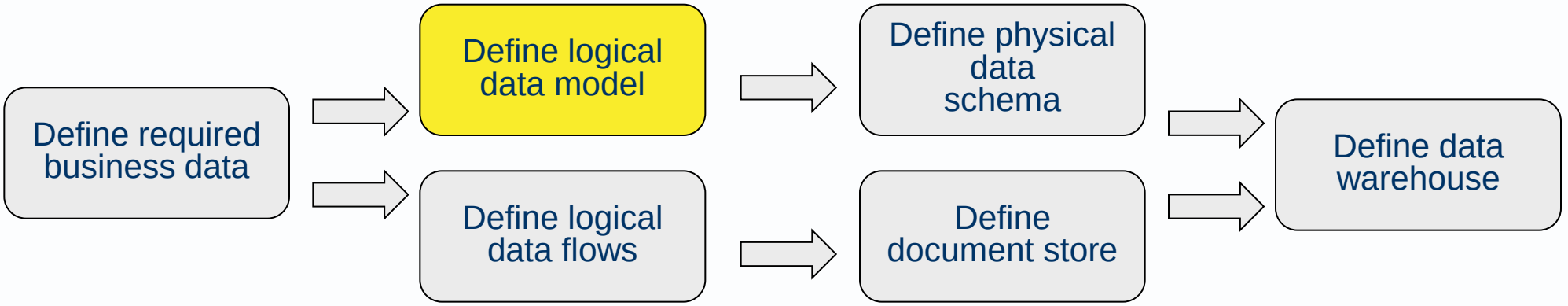
1. Analyse I/O documents to find “entities” identified by primary keys.
2. Define a logical data model - by “normalising” and relating the entities
3. Validate the data structure

Analyse I/O documents to find “entities” identified by primary keys

- Look for primary keys used in the business



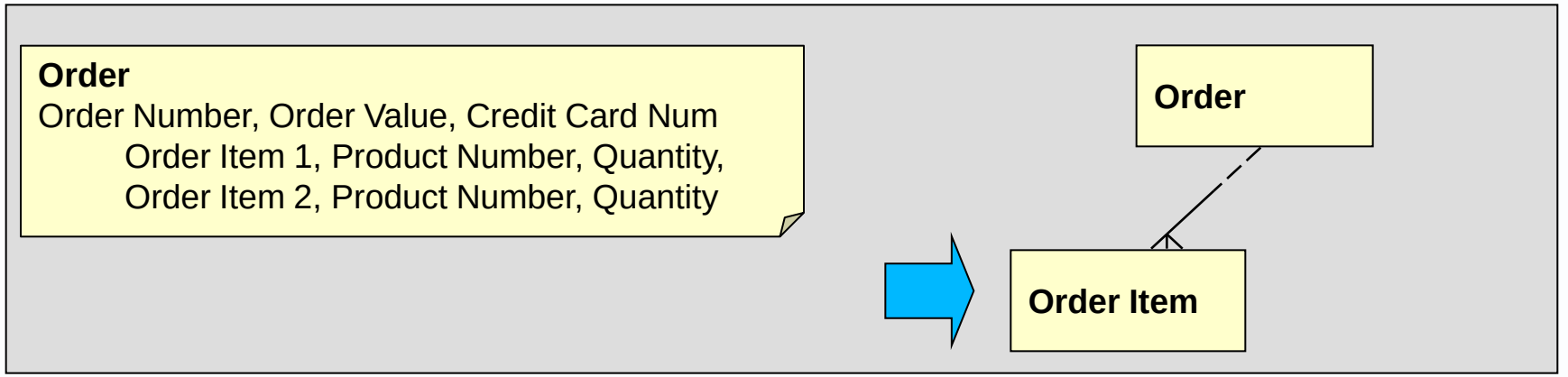
AM level 3 and 4 process: Define logical data model

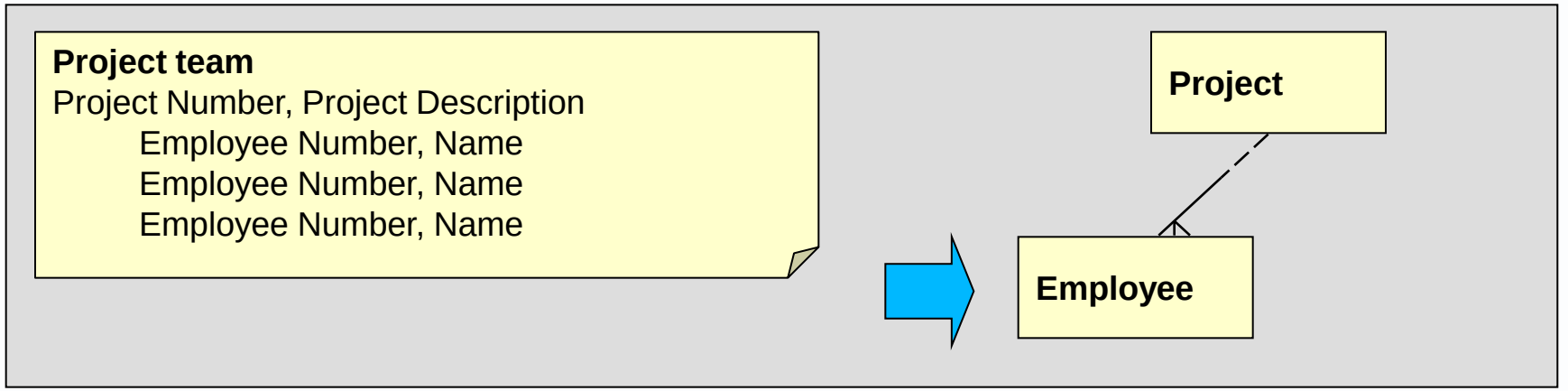


1. Analyse I/O documents to find “entities” identified by primary keys.
2. Define a logical data model - by “normalising” and relating the entities
3. Validate the data structure

First normal form

- Separate a repeating group into a detail entity



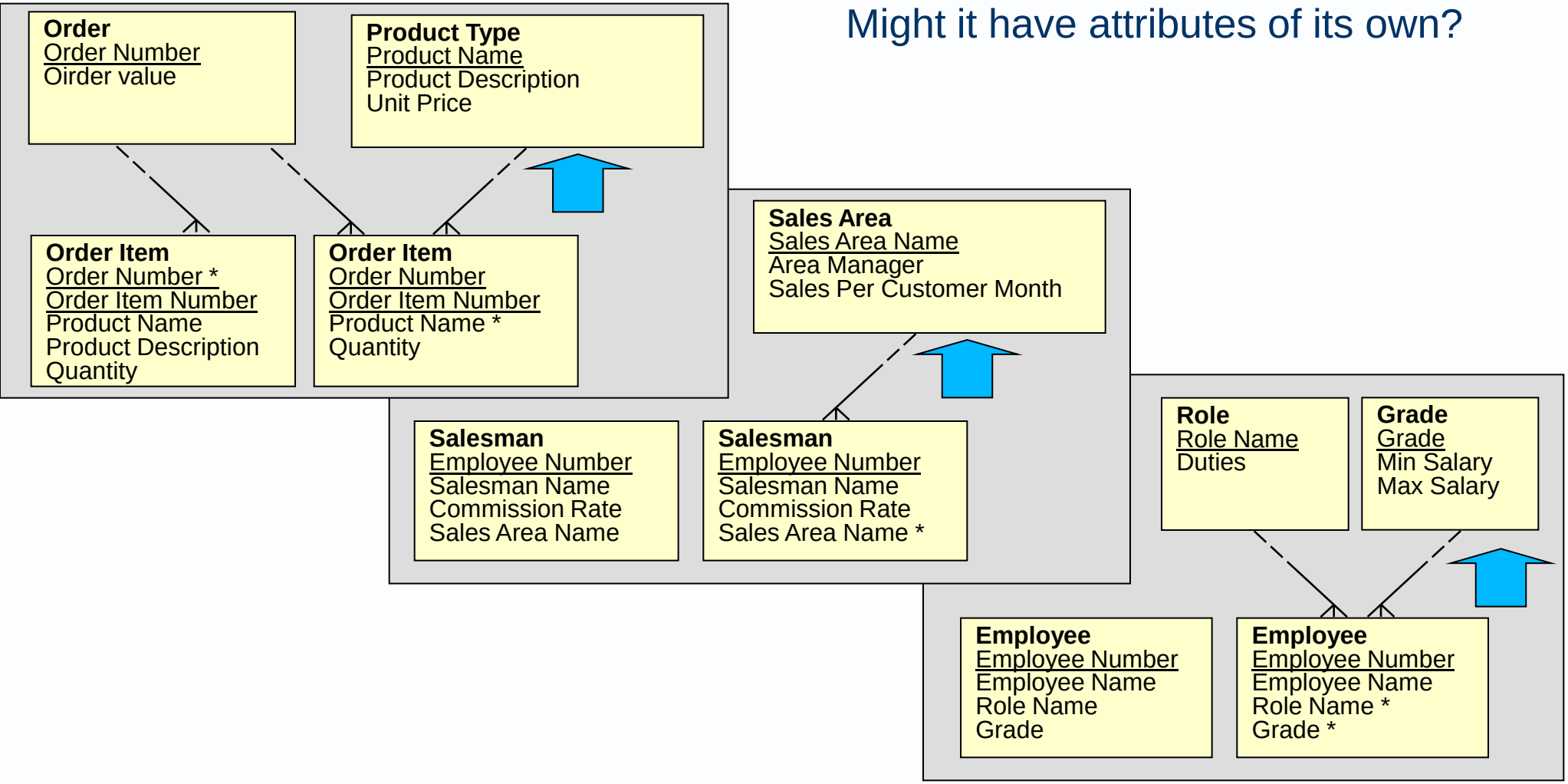


2nd and 3rd normal forms: raise attributes to become an entity

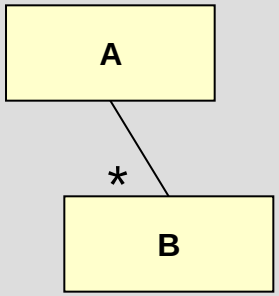
Avancier

Is an attribute significant as an entity in its own right?

Might it have attributes of its own?



Consider an association from both ends

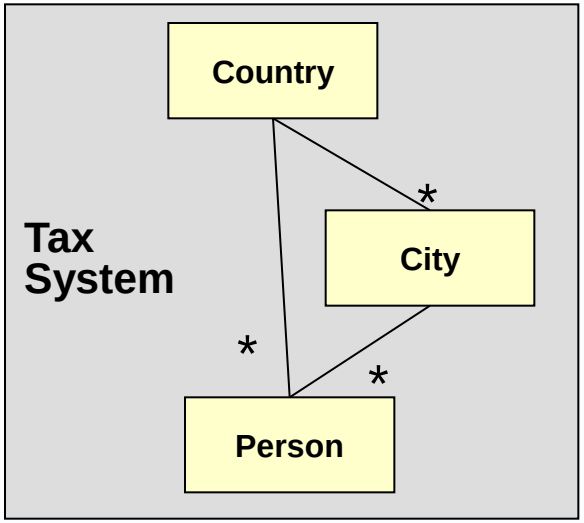


- Does A relate to more than one B? Yes
- Does B relate to more than one A? No

Look for constraints in triangles

- Does one Country contain more than one City? Yes
- Is one City located in more than one Country? No

- Is one City the residential location of more than one Person? Yes
- Does one Person reside in more than one City? No

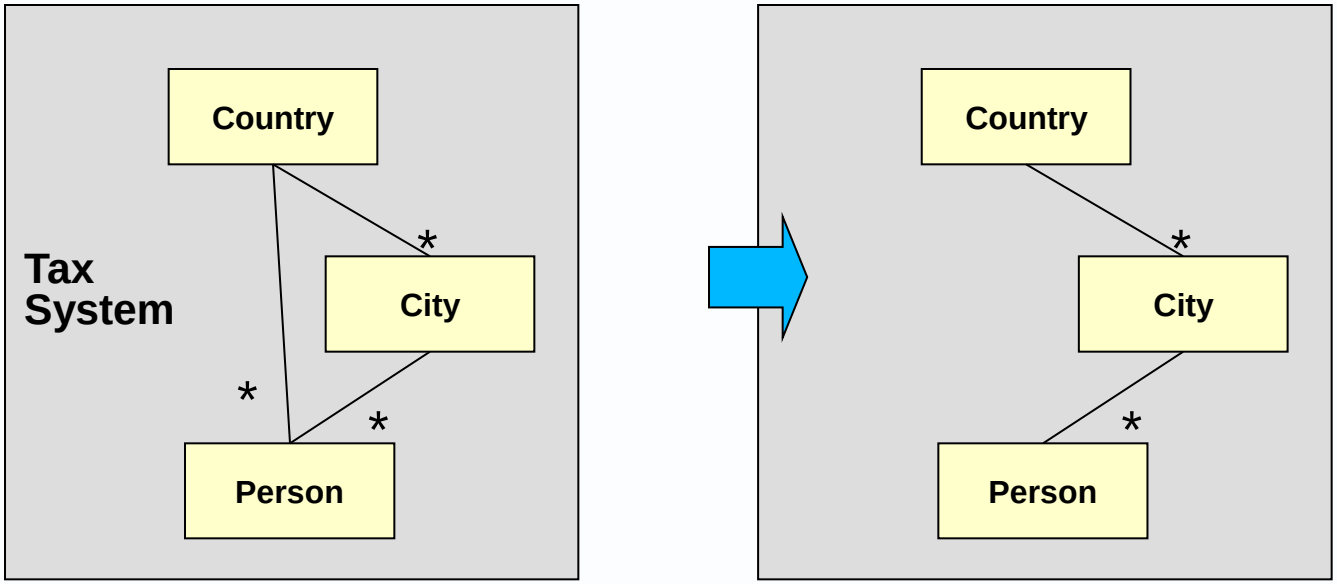


Is there redundant information we could remove?

- Does one Country tax more than one Person? Yes
- Does one Person pay tax in more than one Country? No

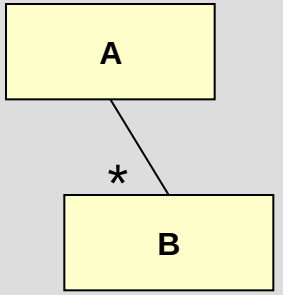
Look for constraints in triangles - remove redundant relationships

- Are the same Persons found down both long and short relationships? Yes (say)



Consider an association from both ends

- Does A relate to more than one B? Yes
- Does B relate to more than one A? No



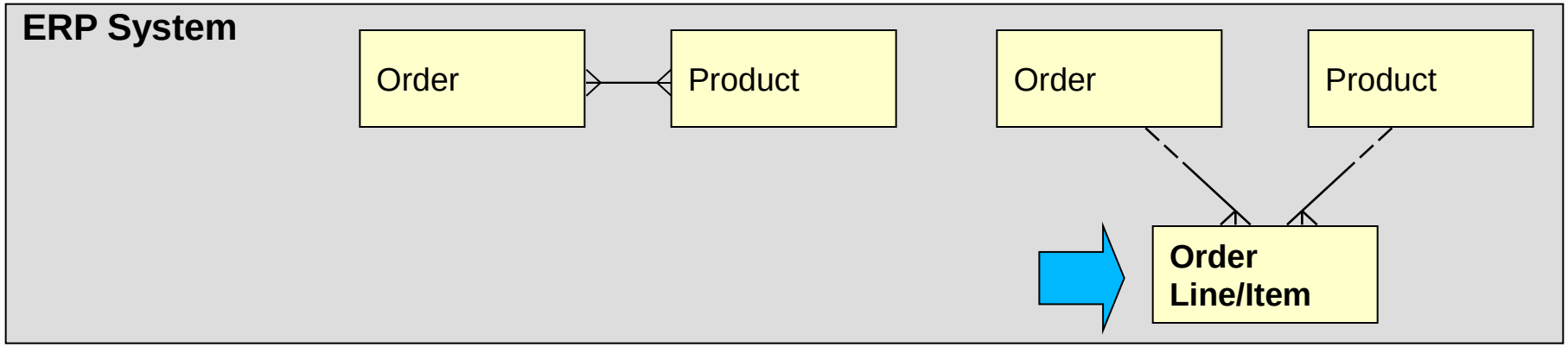
- Does X relate to more than one Y? Yes

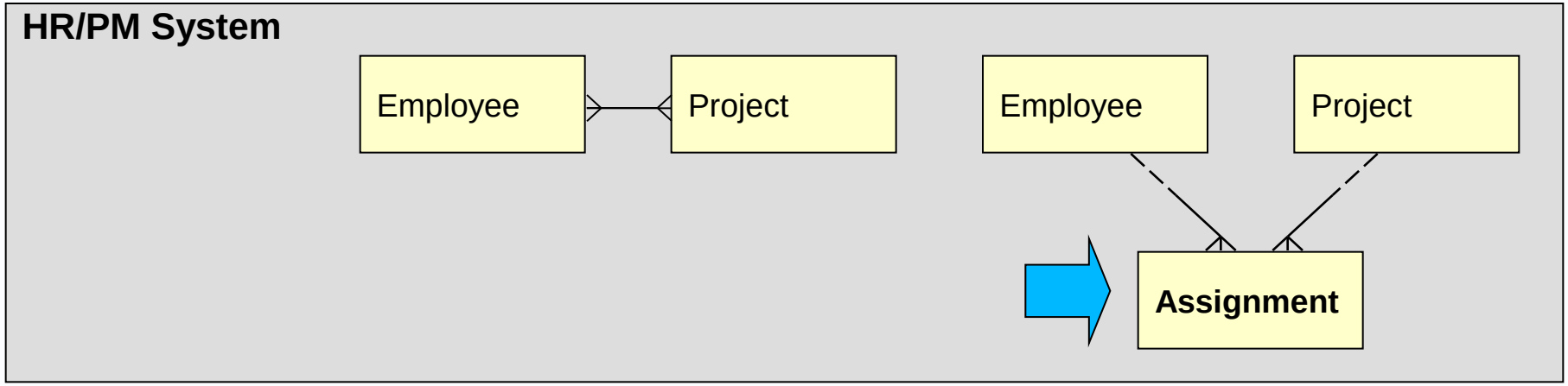
- Does Y relate to more than one X? Yes

Look for link entities to resolve N-to-N associations

Avancier

What is the link entity that connects them? What is the name of the event or the thing that joins one entity to one of the other entity?





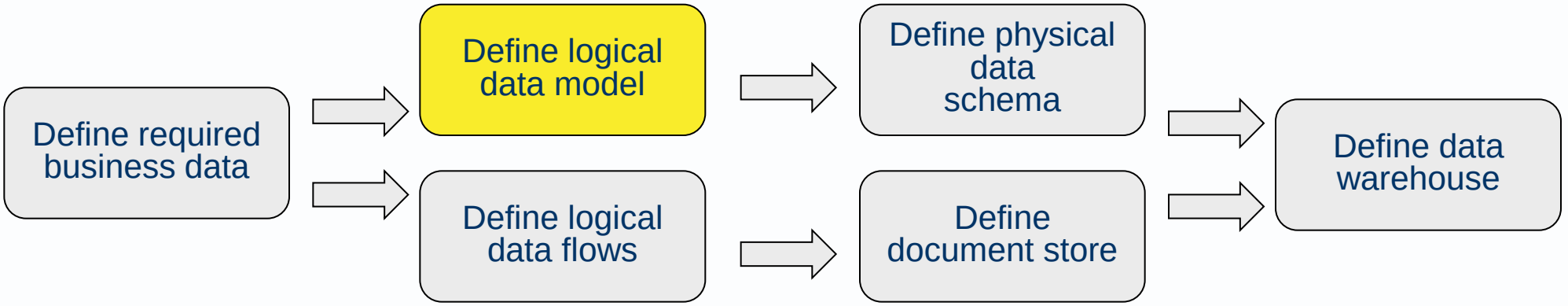
1998: an e-business idea

- 1998
- Amazon are selling books
- We are going to be the Amazon of hotel booking

- We have found some sales partners (travel agents)
- We have found some hotel partners (hotel chains)

- Our application will be deployed in travel agents
- The agent will use the application to help a customer find a hotel
- Then make booking a room for them
- And send a confirmation email to the customer

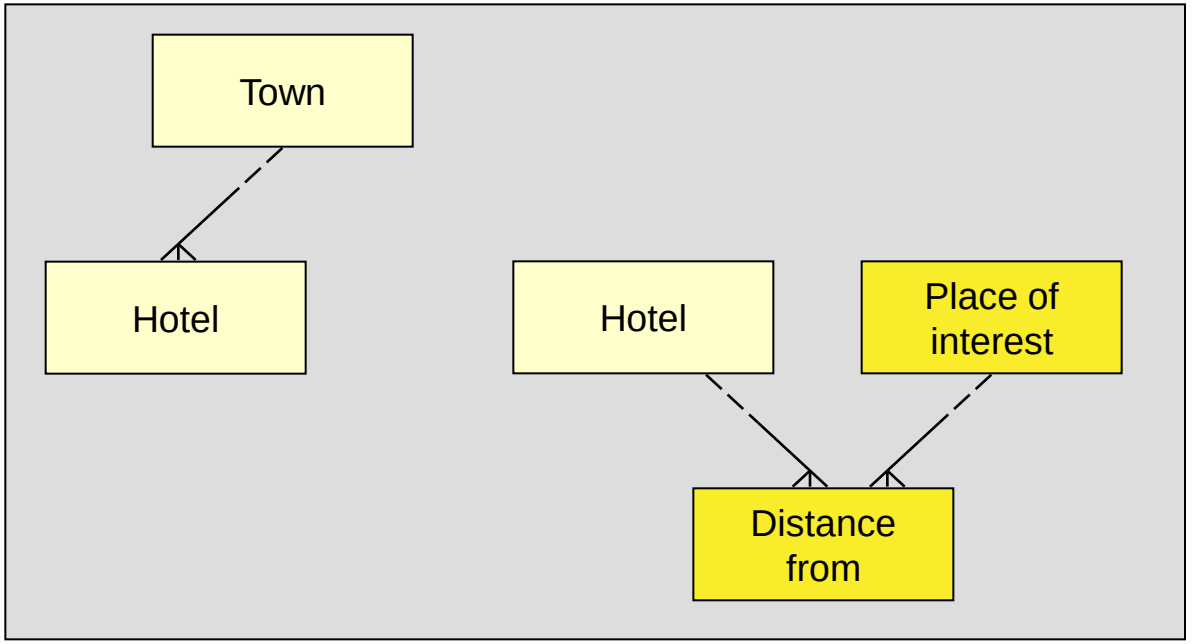
AM level 3 and 4 process: Define logical data model



1. Analyse I/O documents to find “entities” identified by primary keys.
2. Define a logical data model - by “normalising” and relating the entities
3. Validate the data structure

Validation 1: Is it flexible enough to meet user needs?

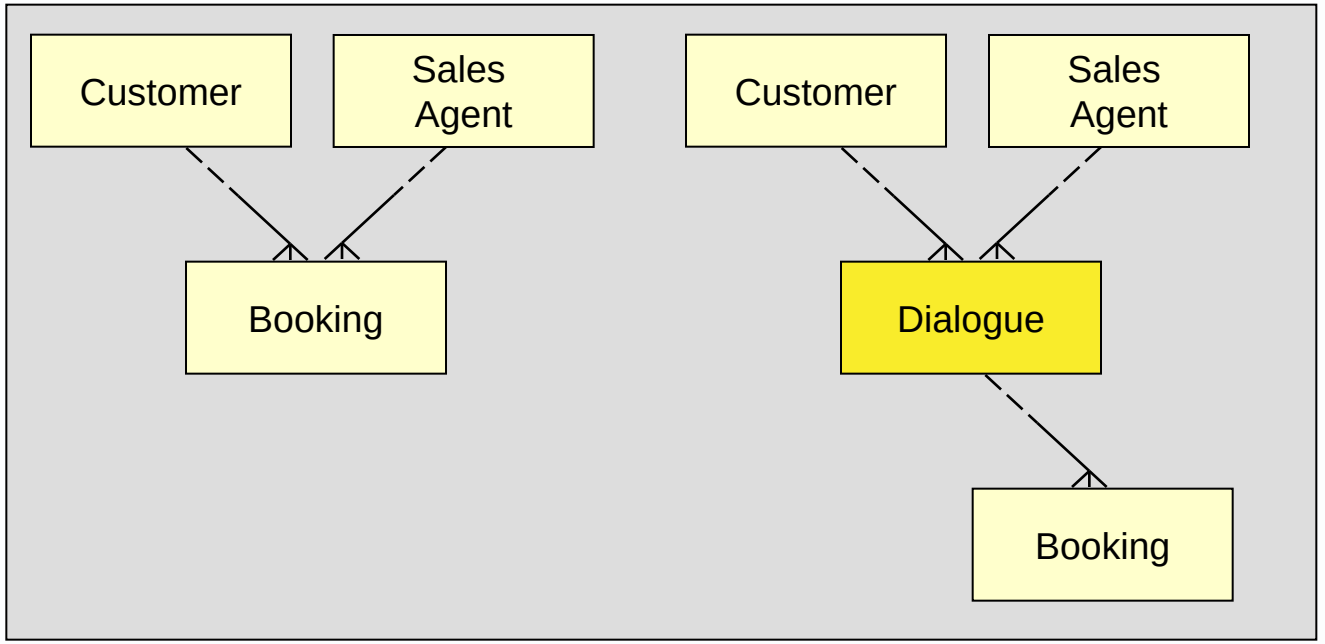
- Is some generalisation needed?
- Do we search for hotels only within a town?
- A more flexible search introduces an N-to-N association



Validation 2: Does it record enough events?

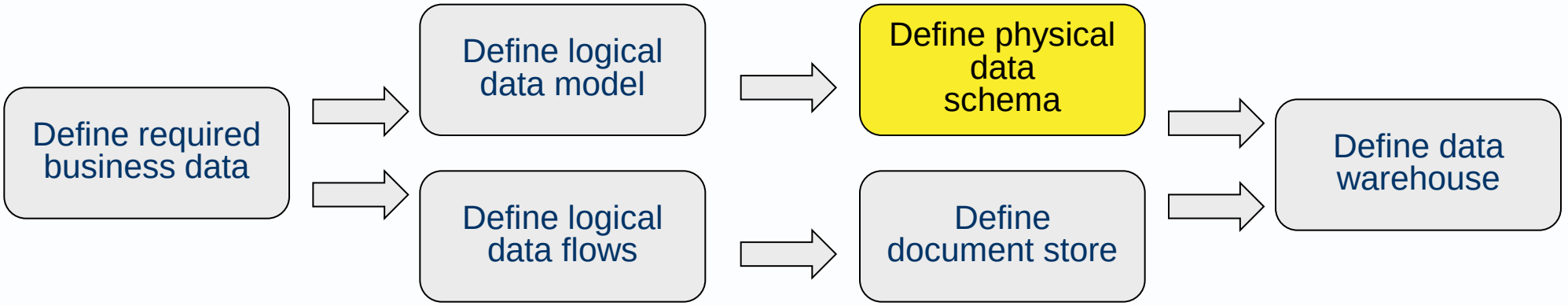
- Does the business need to remember **transient events** (e.g. credit and debit transactions) as well as **persistent entities** (e.g. current accounts)?

- Consider **recording enquiry interactions** as well as updates



- Argos record enquiries for out-of-stock items as well as orders

AM level 3 and 4 process: Define the physical data schema



1. Refine the design for flexibility and performance
2. Refine the design to meet CIA and scalability requirements

Refine the design for flexibility and performance

- Identify processes to be supported, especially
  - batch input and output processes
  - predicted queries and required reports
  - processes that are frequent or have long access paths
- Ensure the data model contains data needed by those processes
- Facilitate process access paths
  - Do access path analysis
  - Add derivable data
  - Add derivable relationships
  - Otherwise denormalise the data structure

Design a data model to meet requirements - outputs

- **Data store model**

Data analysis

Required output data flow

CUSTOMER entity	
Customer number	Primary Key
Customer name	
Customer address	

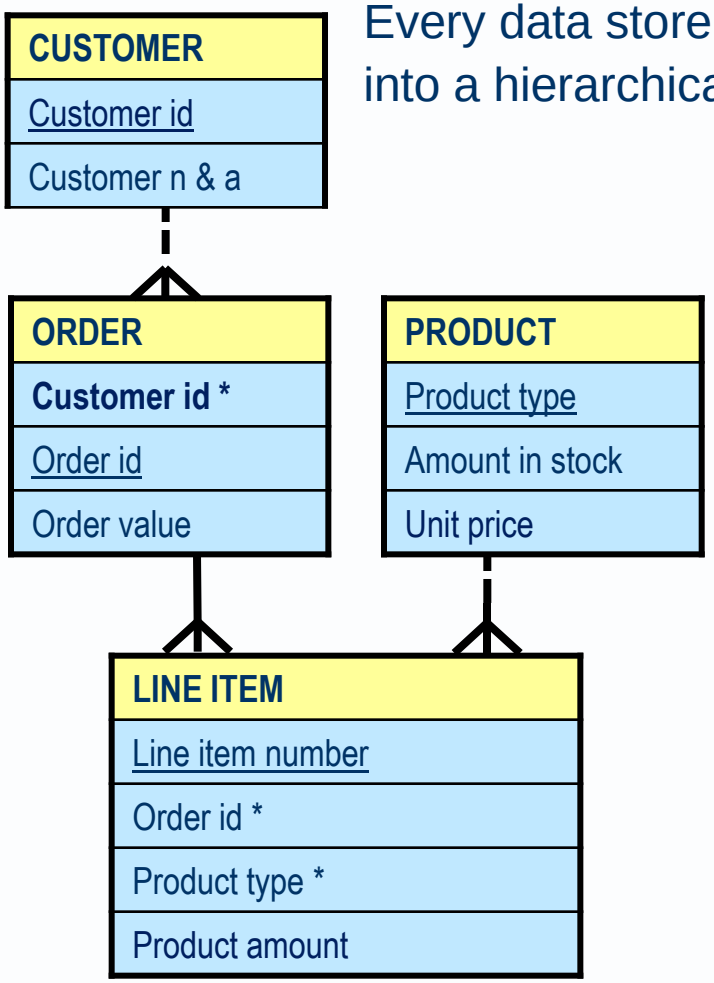
Data analysis

ORDER entity	
Customer number	Foreign Key
Order number	Primary Key
Order amount	
Product type	

Customer Order History SEQUENCE
- **Customer number**
- **Customer name**
- **Customer address**
- Set of Orders ITERATION
  - Order SEQUENCE
    - **Order number**
    - **Order amount**
    - **Product type**
  - Order END
- Set of Orders END

Customer Order History END

Consider how data will be serialised into a required data flow



Every data store can be serialised into a hierarchical / sequential data flow

Serialisation

Customer Order History
- **Customer id**
- **Customer name and address**
- Orders Placed
  - **Order id**
  - **Order value**
  - Products Ordered
    - **Product type**
    - **Product amount**
  - Products Ordered End
- Order Placed End

Customer Order History END

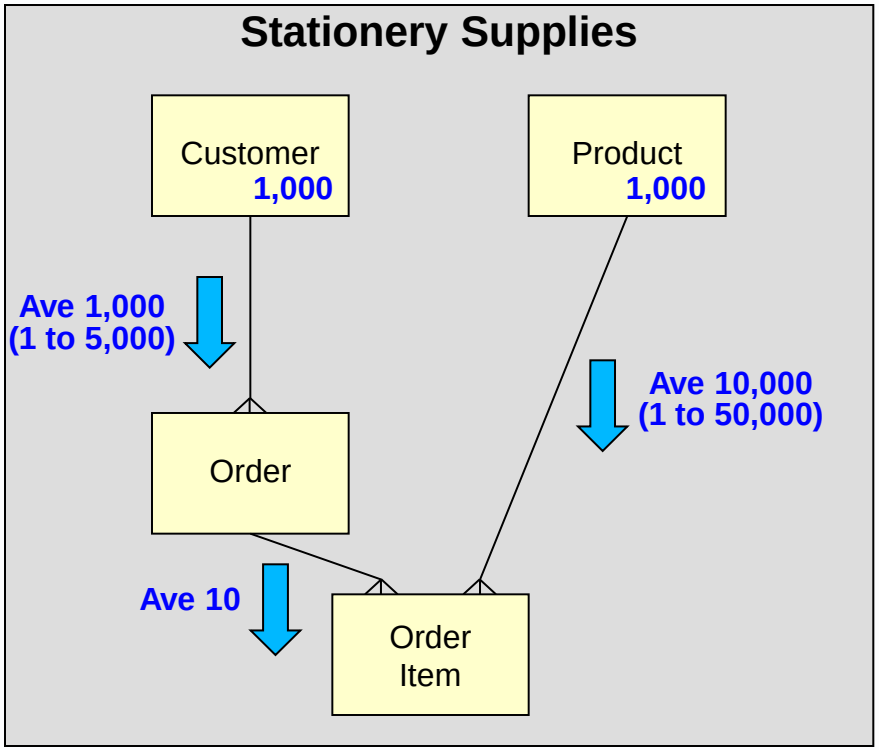
Serialisation

Product Demand
- **Product type**
- **Amount on hand**
- Products ordered
  - **Product amount**
  - **Order id**
- Products Ordered End

Product Demand End

Don’t forget the numbers: school stationery supplies

- Given a Logical Data Model, define
  - the volumes of kernel entities,
  - the population of each relationship,
  - expected growth rates.



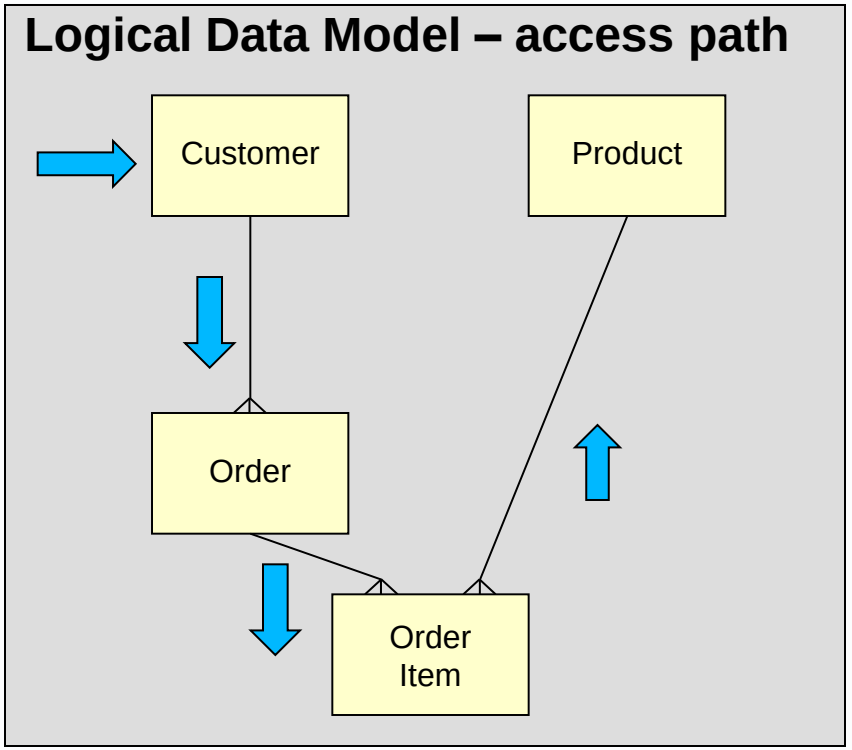
The typical customer (a school)
has placed 1,000 orders,
each with 10 items,
for our stationery products

These numbers influence
both system usage
and physical design

Design a data model to meet requirements - processes

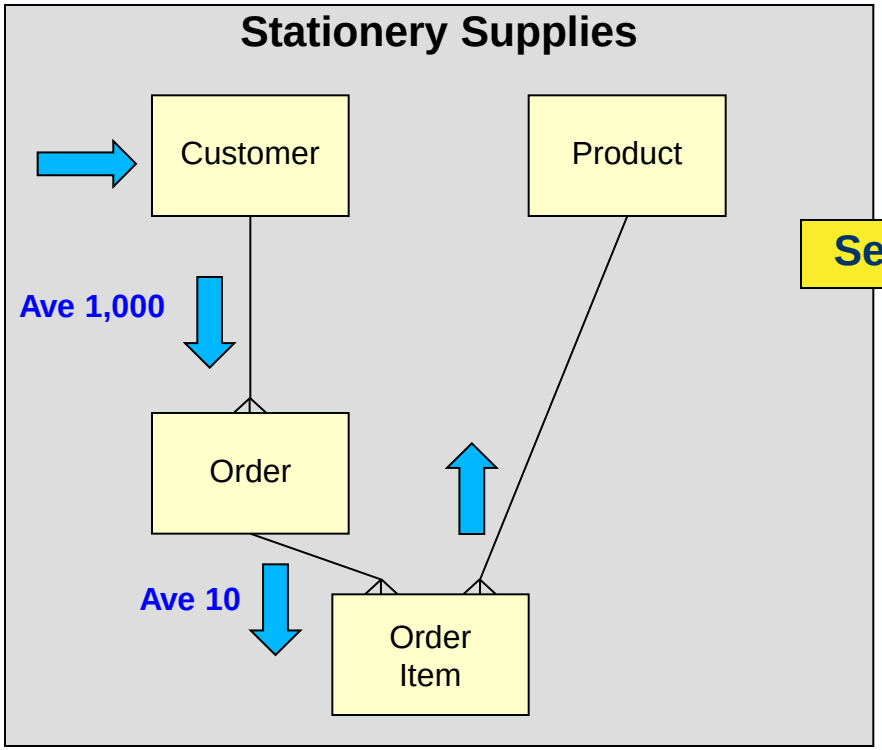
- [An artifact] that shows the route that a process takes through a data structure
- It is used to validate the data structure and study performance issues.

List all the products that a customer has ordered



The first system release

- Before calling, a salesman requests a customer's order history
- The application prints out a list of 10,000 order items.
- "That's no good! I only want a list of products the customer has ordered!
  - (on average, 15 product types)

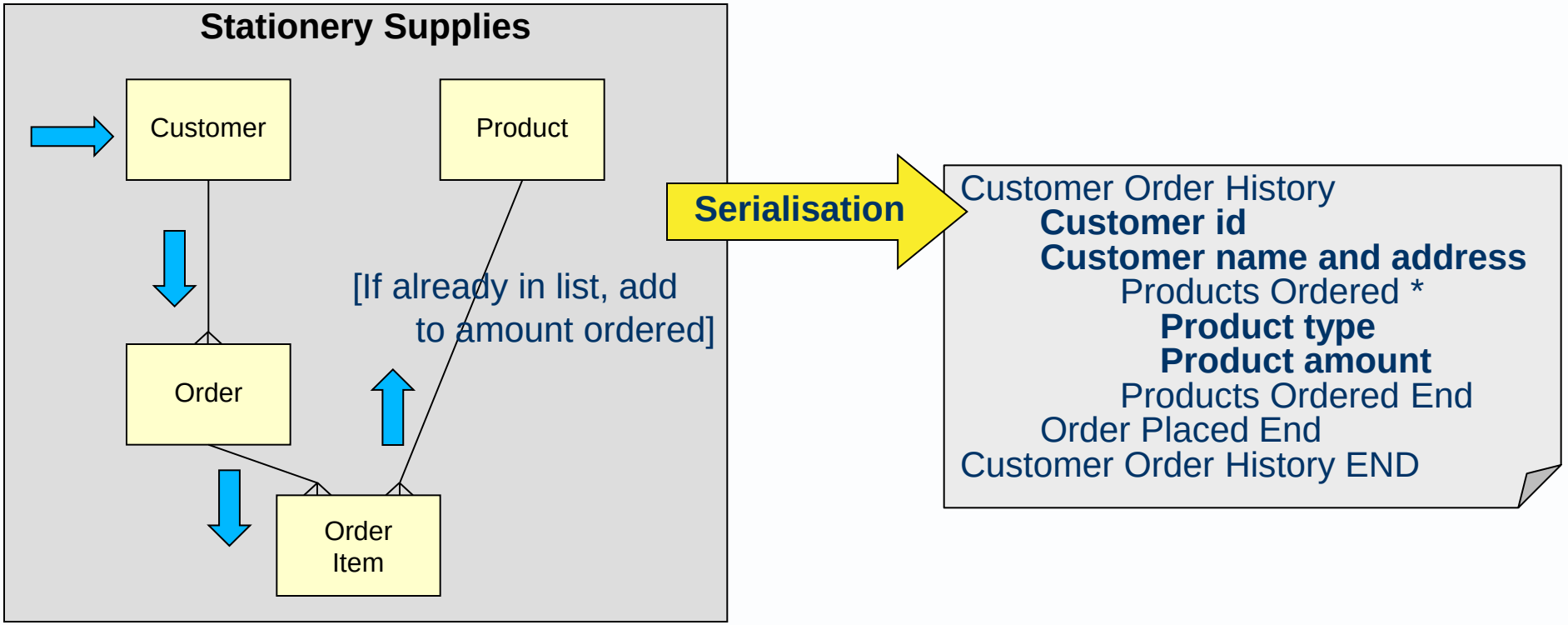


Serialisation

Customer Order History
Customer id
Customer name and address
Orders Placed *
Order id
Order value
Products Ordered *
Product type
Product amount
Products Ordered End
Order Placed End
Customer Order History END

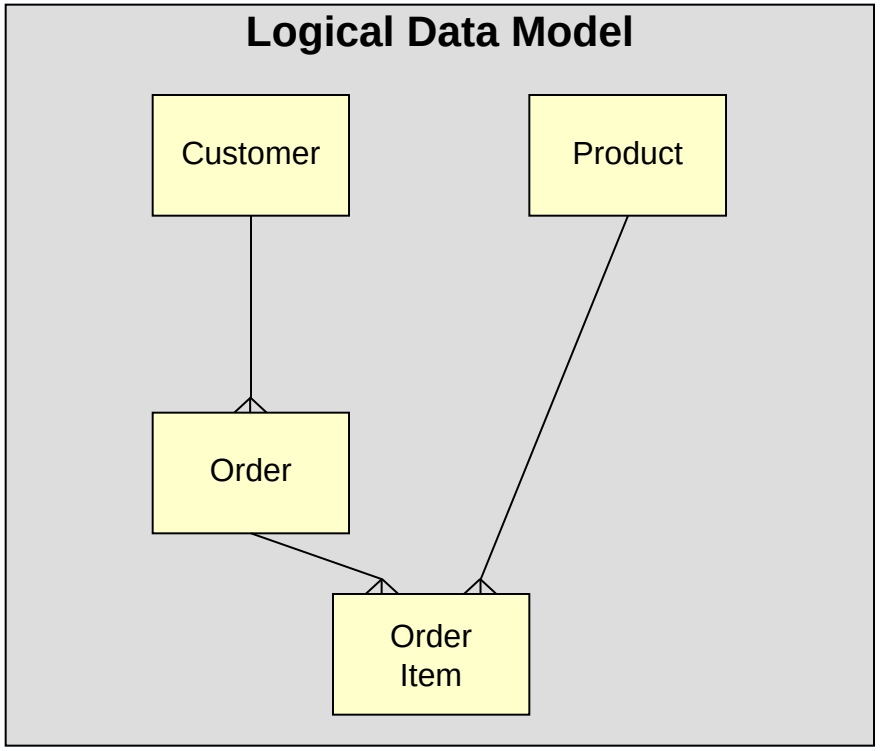
System change request

- Query: “List the products ordered by a customer, with total amounts”
- First, is it feasible? Yes.



Facilitate access paths

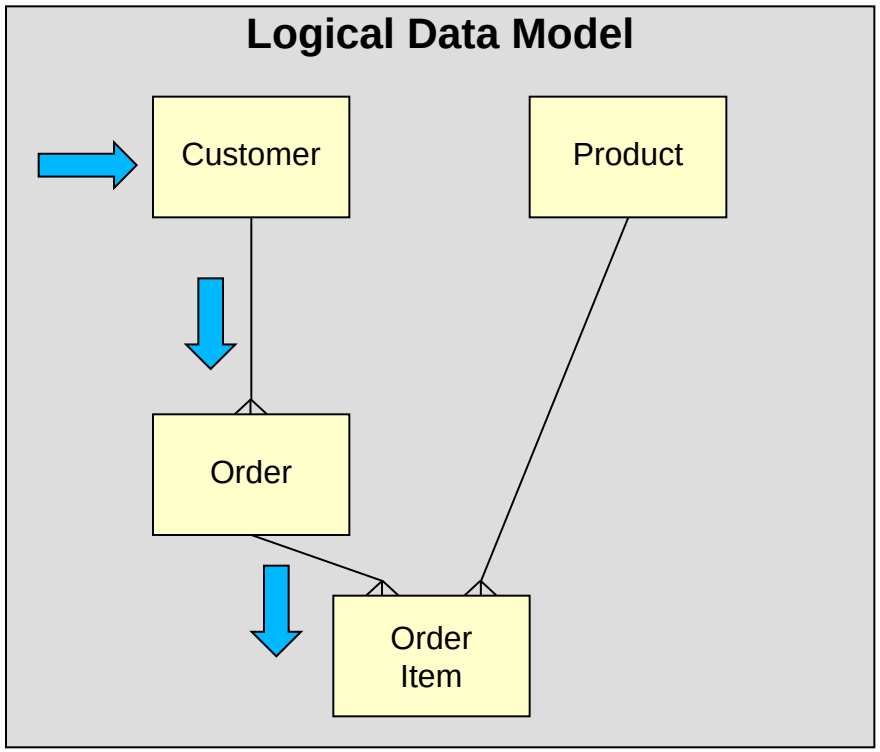
- Make sure critical processes can readily access the data they need
- Do access path analysis
- Add derivable data and/or relationships
- Otherwise denormalise the data structure

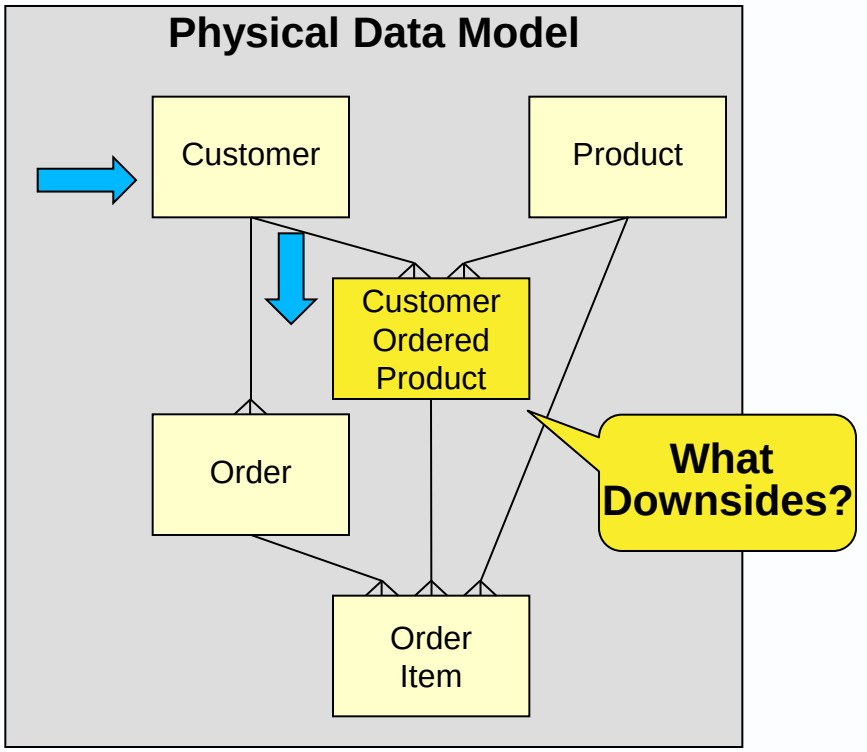


Do access path analysis – is it fast enough?

- The typical customer (a school) has placed 1,000 orders, each with 10 items, for our stationery products
- But each as ordered only 15 products

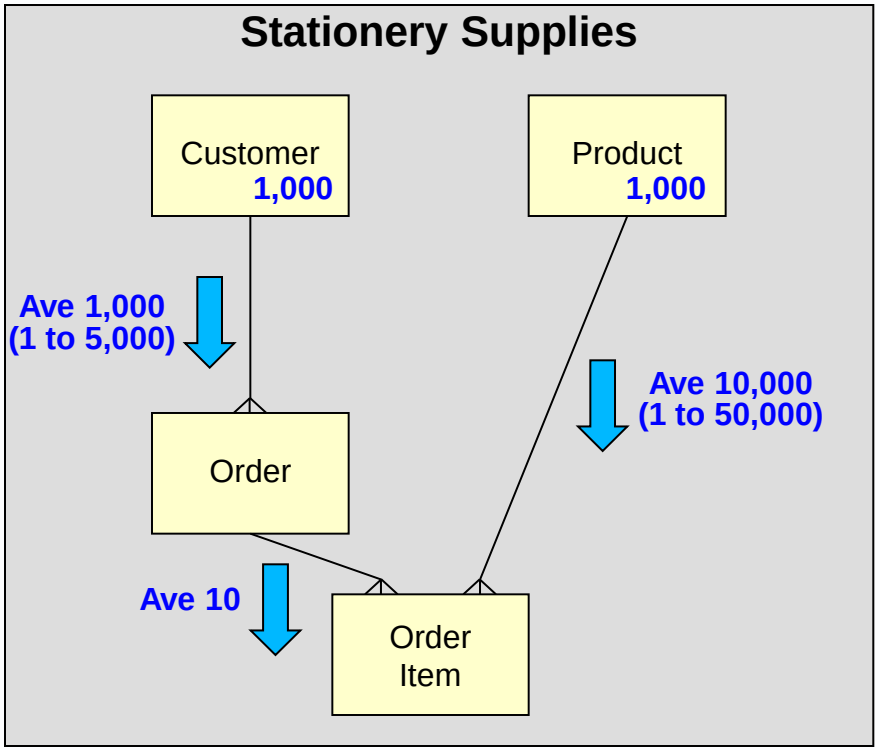
- You can remove redundant accesses by adding redundant data or redundant relationships





Don’t forget the numbers

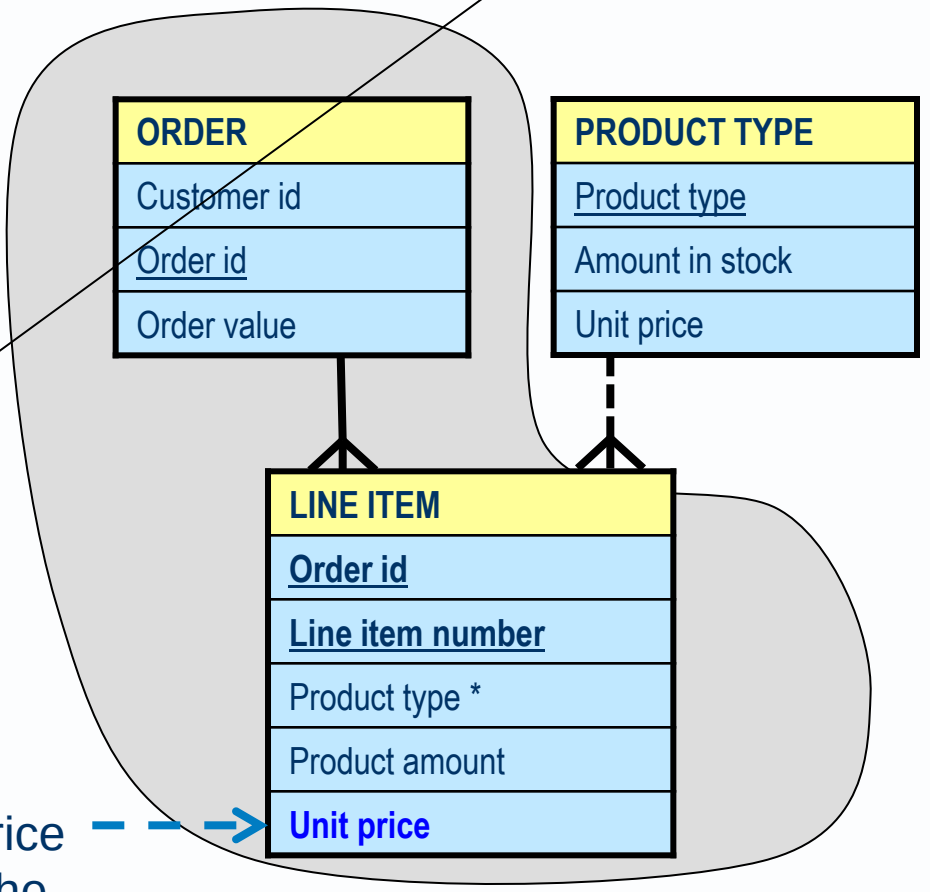
- Given a Logical Data Model, define
  - the volumes of kernel entities,
  - the population of each relationship,
  - expected growth rates.



The numbers influence both system usage and physical design

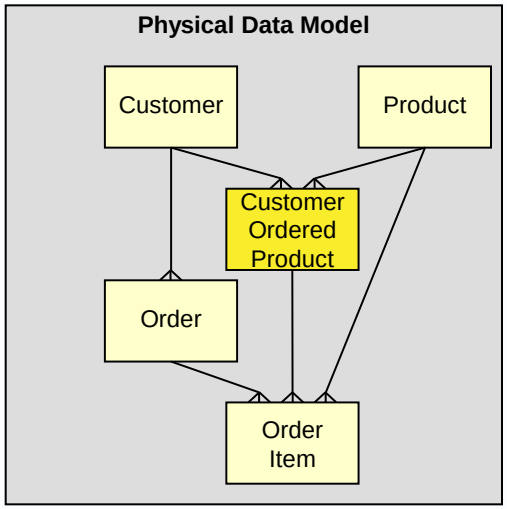
Clustering related data on the same block/page

- Old-fashioned optimisation technique: cluster detail entities with *one* of several parents
- Rule of thumb: “the least dependent occurrence rule”
  - An Order has few Order Lines
  - A Product has many
  - So cluster the Order Lines on the same block/page as the Order
- Copying the price stops the price change transaction affecting the aggregate, which is probably the business rule anyway

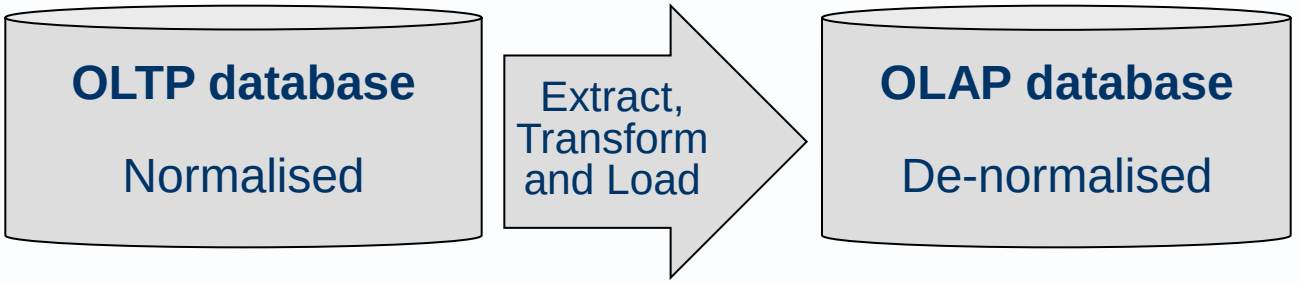


Separation of update and query data stores

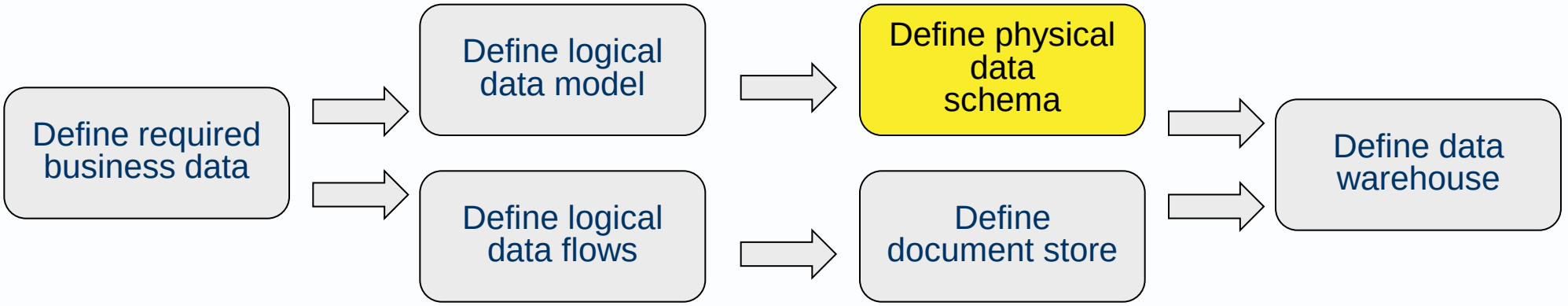
- You can
  - reduce redundant data accesses by
  - adding redundant data, relationships and indexes.



- Where that is not enough, you can
  - Separate update data store from query data store
  - And de-normalise the query data store



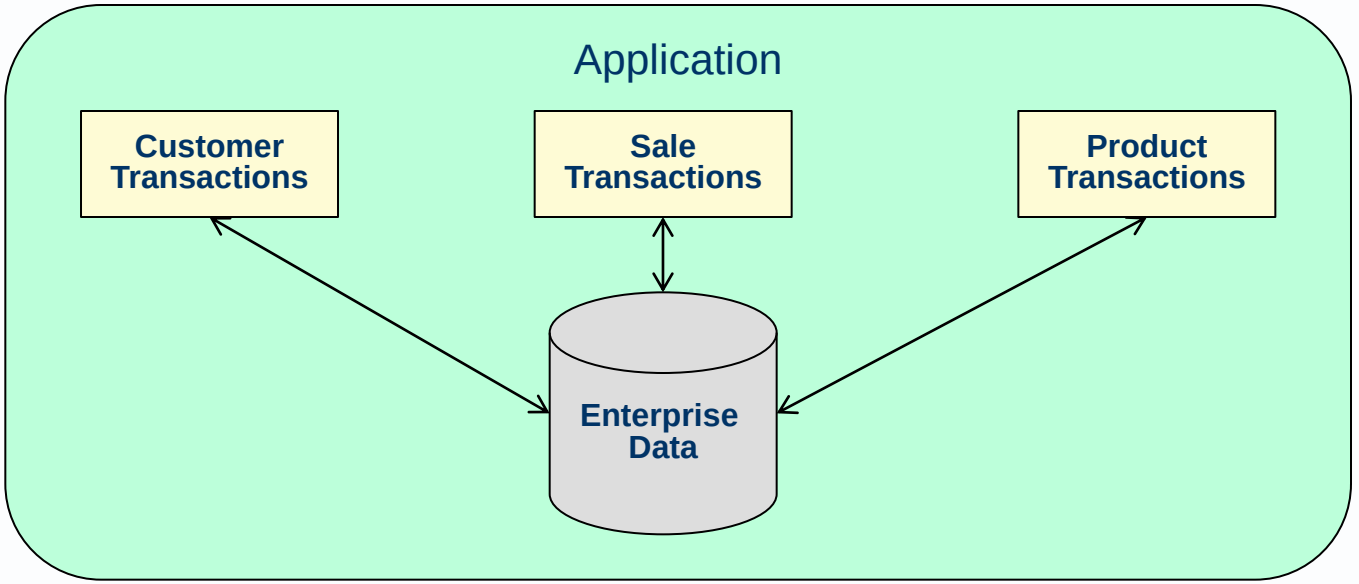
AM level 3 and 4 process: Define the physical data schema



1. Refine the design for flexibility and performance
2. Refine the design to meet CIA and scalability requirements

Design for integrity

- You may consolidate an enterprise's business data into a large database.
  - "It is not hard to speculate about, if not realize, very large, very complex systems implementations, extending in scope and complexity to encompass an entire enterprise." John Zachman, 1987

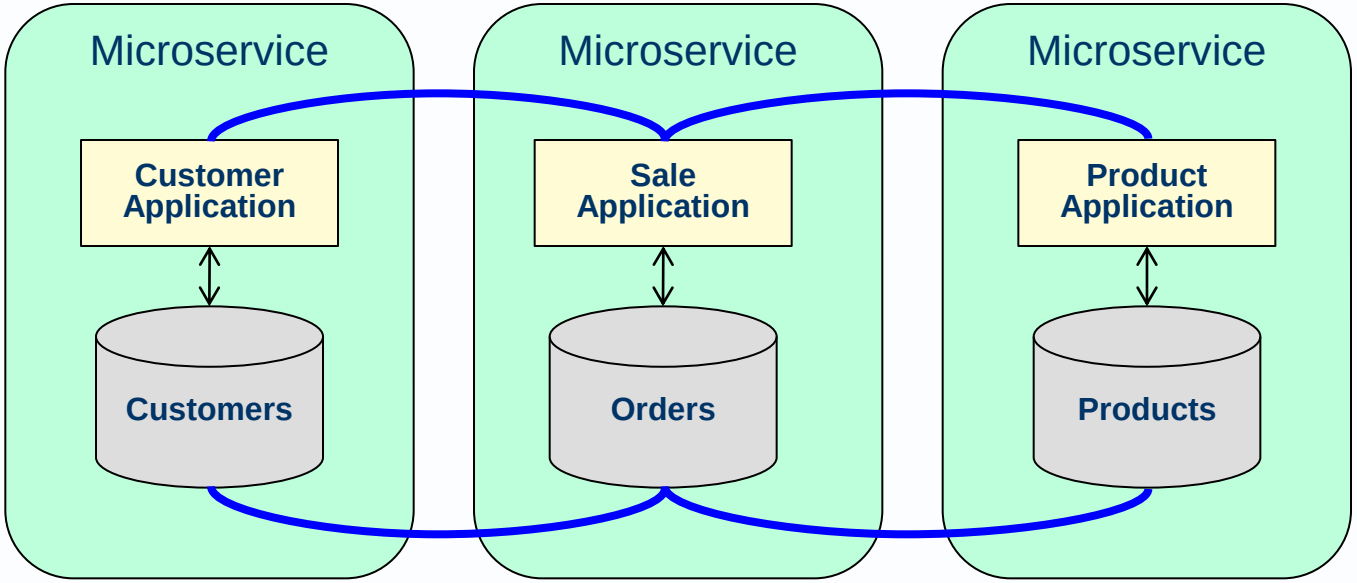


- And it appears SAP pursued this strategy for many years.

Design for scalability – functional scaling

- “Distribute data management" to so-called “micro services” that each maintain part of the cohesive data structure

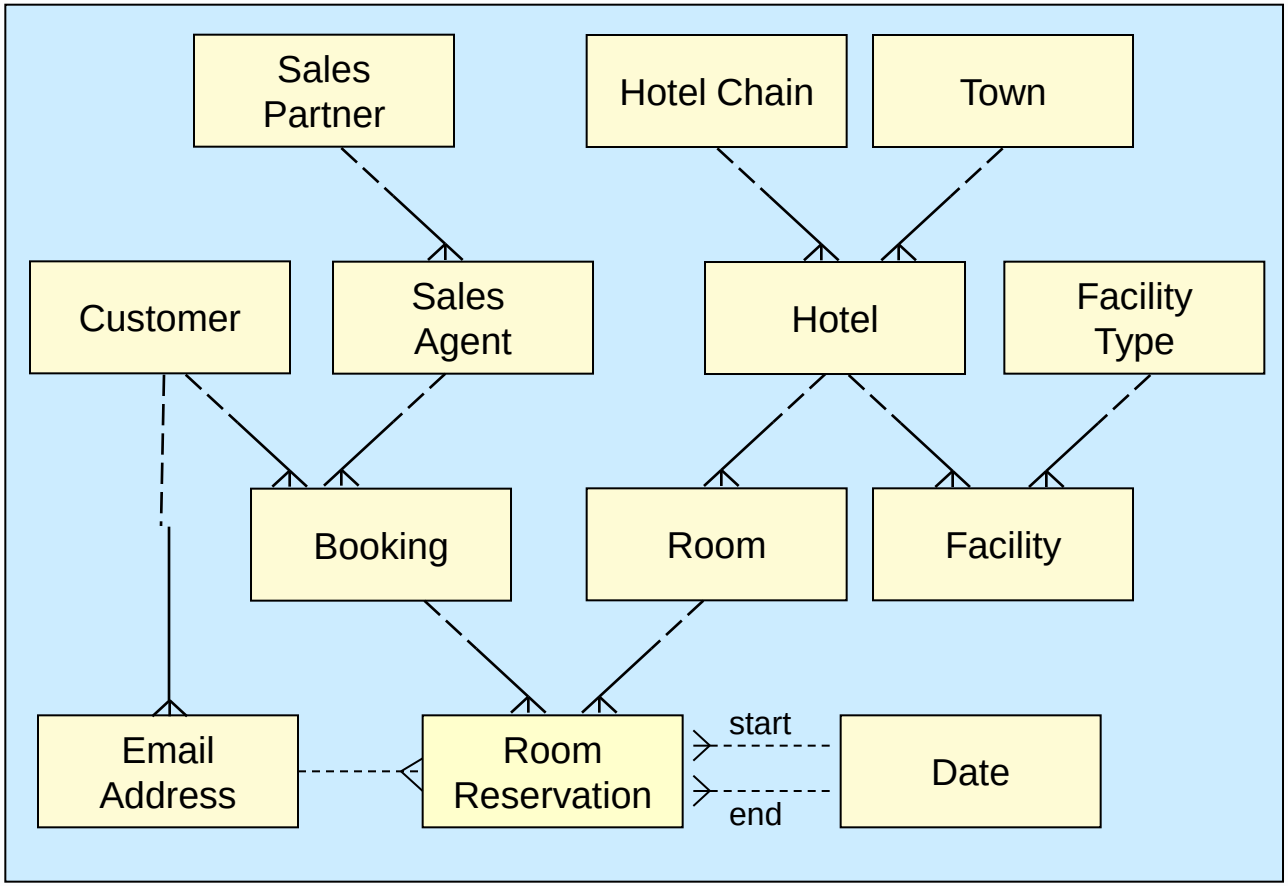
- They may be deployed separately, but are *still coupled*



- Integrity must be maintained by BASE rather than ACID transactions

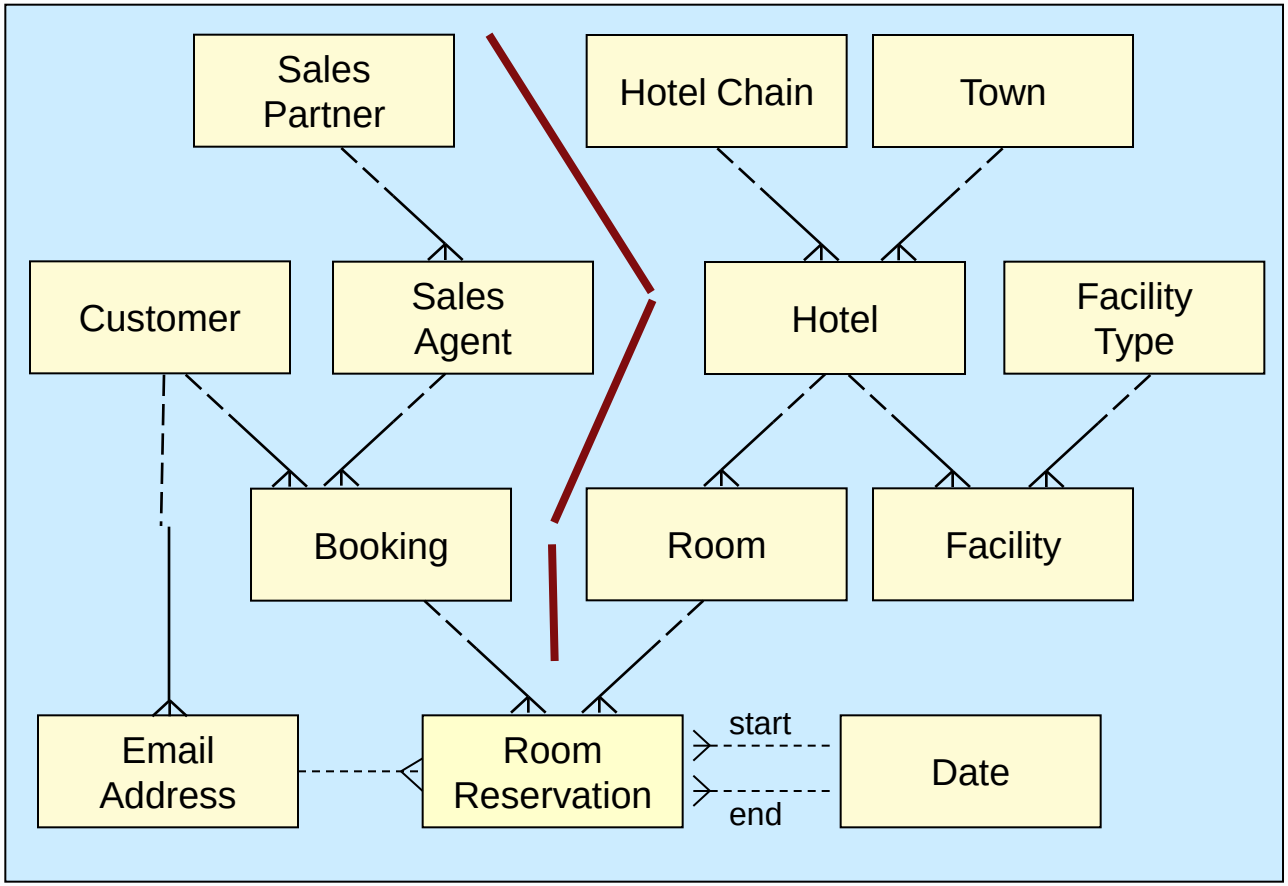
Partitioning a data store for availability and scalability

- How would you divide this logical data model between two data stores?



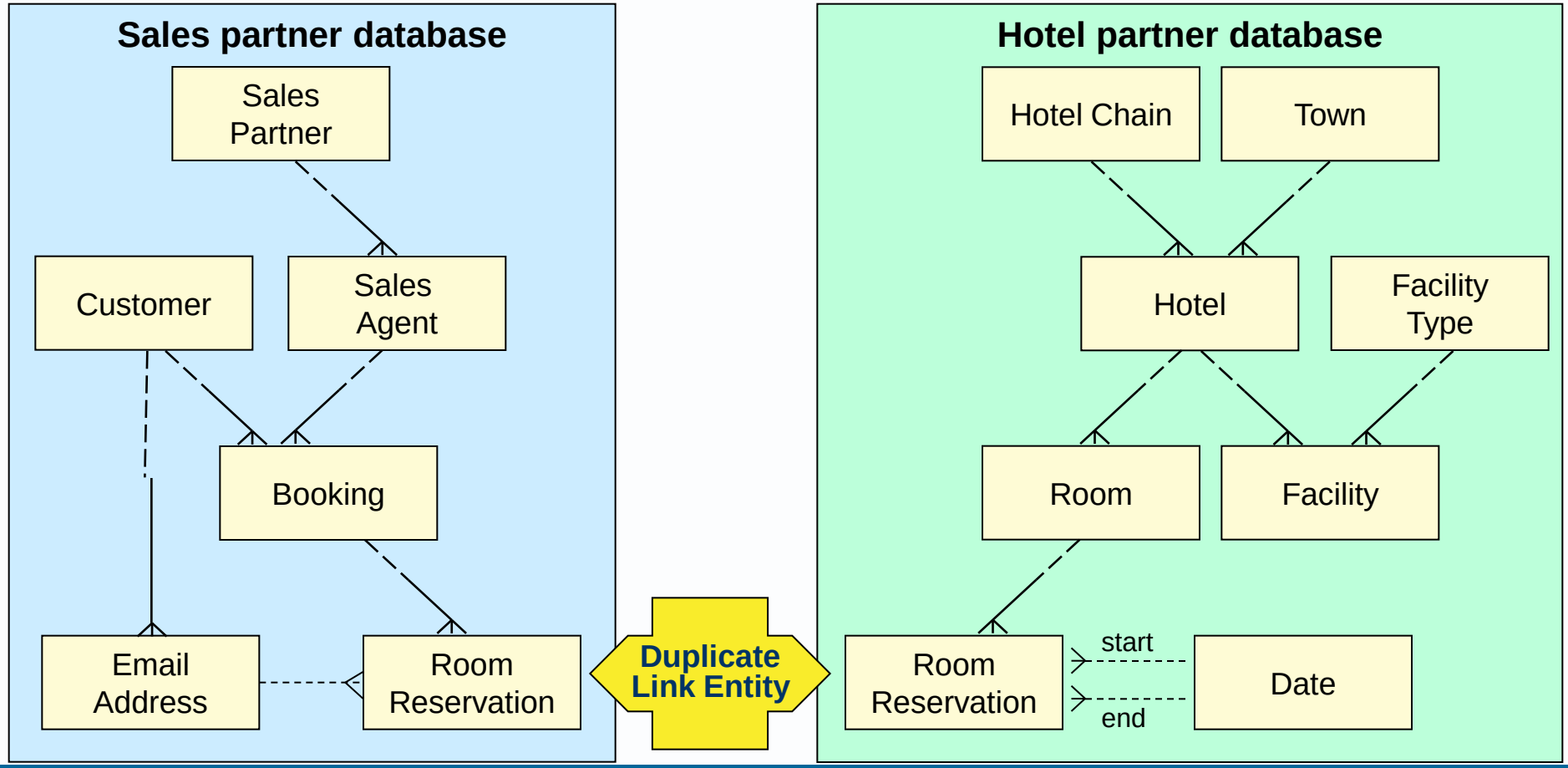
Partitioning a data store for availability and scalability

- How would you divide this logical data model between two data stores?



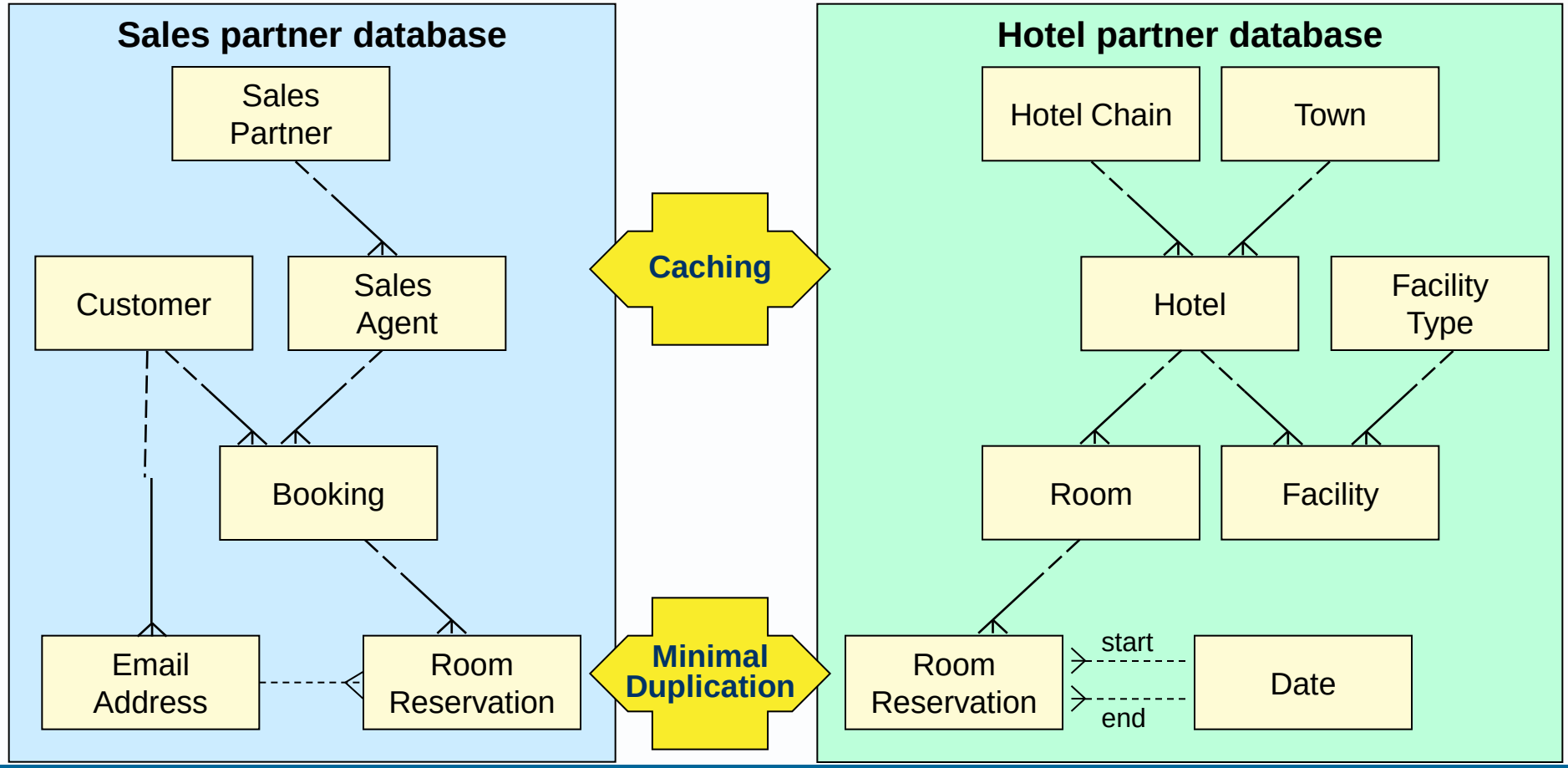
Horizontal partitioning into two data stores

- Divide the network structure into hierarchies: has implications for both software architecture and business architecture… tbd

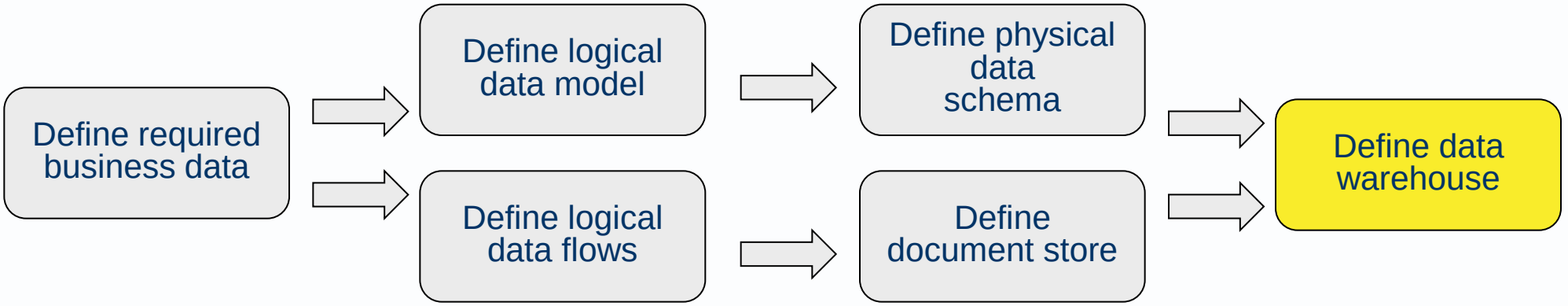


In practice – deliberate duplication of data storage

- To minimise network traffic, availability and contention issues
- A substantial amount of each database may be cached in the other

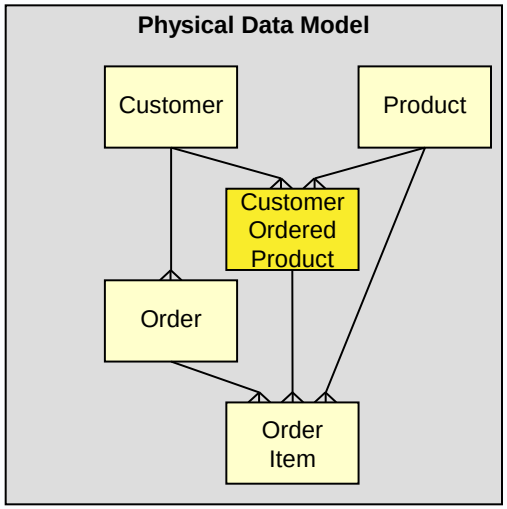


AM level 3 and 4 process: Define logical data model

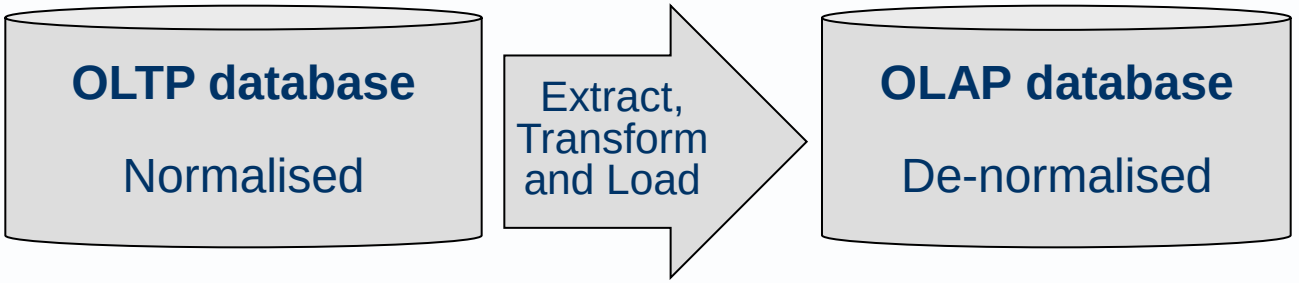


Separation of update and query data stores

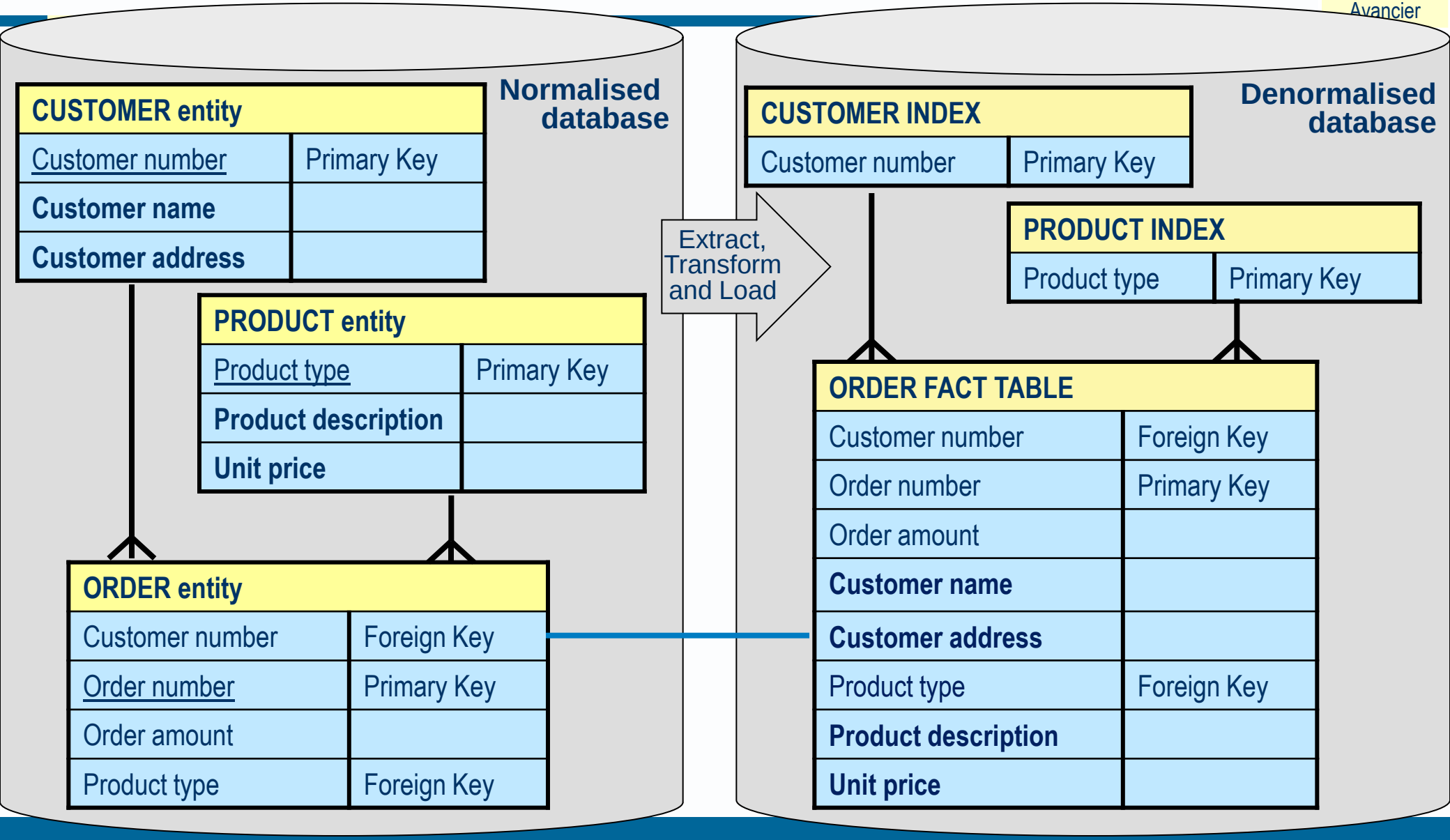
- You can
  - reduce redundant data accesses by
  - adding redundant data, relationships and indexes.



- Where that is not enough, you can
  - Separate update data store from query data store
  - And de-normalise the query data store



Denormalisation for faster enquiry/report processes (naive picture)



SAP https://slideplayer.com/slide/5661890/

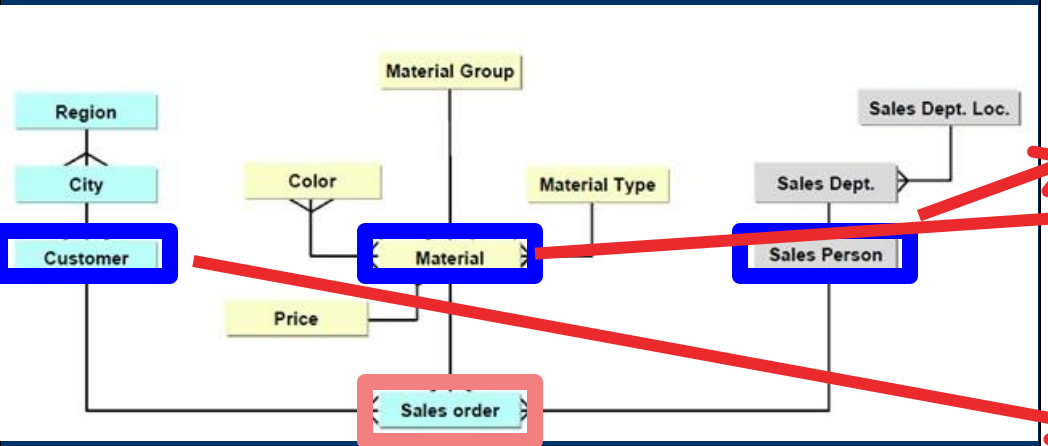
5.2 Data at rest (physical)

Normalisation for update

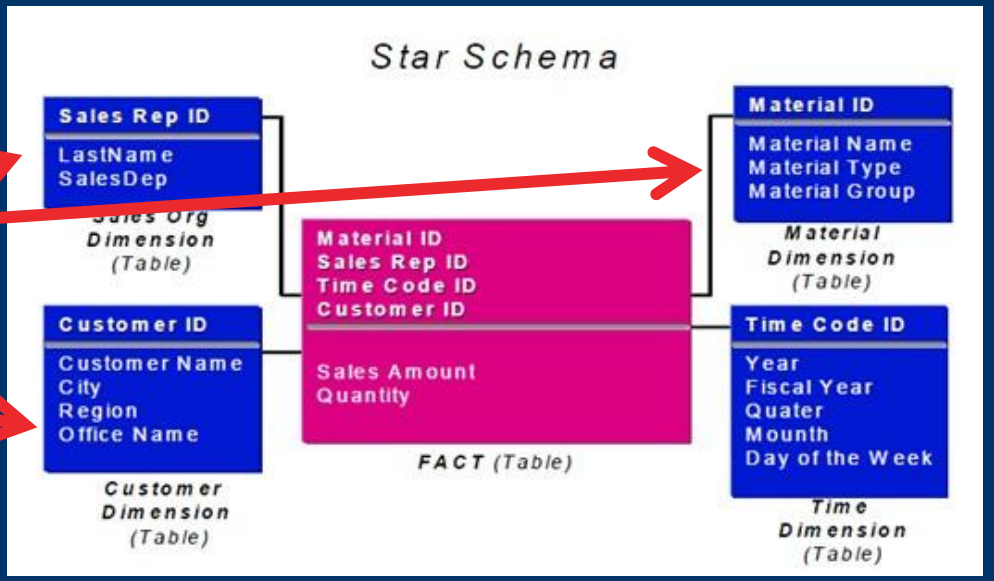
Denormalisation for reports

Avancier

OLTP database



OLAP database



Lots more data modelling advice (buried) on the web site

- Model with a purpose
- Analyse the data in required I/O data flows
- Analyse the data to “1st normal form”
- Analyse the data to “3rd normal form”
- Consider uniquely identifiable attributes as entities
- Study pre and post conditions of process steps
- Consider the same rules as constraints on relationship cardinalities
- Consider (and name) an association from both ends
- Look for constraints in triangles - remove redundant relationships
- Look for redundancy in 1-to-1 associations
- Look for link entities to resolve N-to-N associations
- Look for constraints in double V associations
- Consider recording enquiry interactions as well as updates
- Consider how data will be serialised into data flows
- Do access path analysis
- Denormalise for faster enquiry/report processes
- Consider the wider and longer-term perspective
- Establish data history and versioning requirements
- Beware class hierarchies in models of persistent data
- Use the business’s natural primary keys as a guide
- Consider exclusion arcs in place of subtypes
- Don’t invent super types just because you can
- Minimise multiple inheritance
- Don’t invent concepts you don’t need
- Don’t map all class hierarchies to tables in the same way
- Consider separating type from instance
- Consider roles in place of sub types