Requirements Engineering Appendix

Classic System Modeling Approaches

Les Waguespack, Ph.D.

Copyright and References

* Requirements Engineering, Kotonya & Sommerville, Wiley, Chichester, West Sussex, England, ISBN 0-471-97208-8
* Software Requirements Engineering, Second Edition, Richard H. Thayer and Merlin Dorfman, eds., pp. 7-22. Los Alamitos, Calif.: IEEE Computer Society Press, 1997.
* Use Case Modeling, Bittner & Spence, Addison-Wesley / Pearson Education, Inc., Boston, MA, ISBN 0-201-70913-9
* Writing Effective Use Cases, Cockburn, Addison-Wesley, Boston, MA, ISBN 0-201-70225-8
* UML and the Unified Process - Practical Object-Oriented Analysis and Design, Arlow & Neustadt, Addison-Wesley / Pearson Education, Inc., Boston, MA, ISBN 0-201-77060-1
* Business Modeling With UML, Eriksson & Penker, Wiley, Indianapolis, IN, ISBN 0-471-29551-5
* UML 2 Toolkit, Eriksson, Penker, Lyons & Fado, Wiley, Indianapolis, IN, ISBN 0-471-46361-2
* Enterprise Modeling With UML Designing Successful Software Through Business Analysis, Addison-Wesley, Reading, MA, ISBN 0-201-43313-3
* Object Oriented Systems Engineering, Waguespack, course notes CS390, CS460, CS630, CS771, Computer Information Systems Department, Bentley College, Waltham, MA.
* Essentials of Systems Analysis and Design, Valacich, George & Hoffer: ISBN: 0-13-185462-3,
* Modern Database Management, Hoffer, Prescott * McFadden: ISBN: 0-13-145320222521-1.

Outline

* Conceptual Modeling with ER
* Enhanced Entity-Relationship Modeling
* Logical Data Modeling
* Data Flow Modeling
* Logic Modeling

Conceptual Modeling with E-R

Les Waguespack, Ph.D.

Adapted from Topi 2004

Requirement Structuring

* Data modeling
  * Modeling relevant entities and their relationships
  * Entity-Relationship Diagrams (ER, EER)
* Process modeling
  * Modeling how data flows between and is transformed by business processes
  * Data Flow Diagrams (DFDs)
* Logic modeling
  * Modeling processing logic and timing of events within processes
  * Structured English, decision tables, decision trees, state-transition diagrams and tables

Data Modeling

- Conceptual Data Modeling
  - Focus on the problem domain, uses terminology and notations understandable for users
  - Requirements structuring during the Analysis phase of the SDLC
  - (Enhanced) Entity-Relationship (EER) modeling
- Logical Data Modeling
  - Conversion of the conceptual data model into logical structures that can later be implemented
  - Normalization
  - Relational model
- Physical Data Modeling
  - Database implementation and optimization using a DBMS

Life Cycle Data Modeling

* Identification and Selection: Enterprise-wide data model (E-R model with only entities)
* Initiation and Planning: Conceptual data model (E-R model with only entities for a specific project)
* Analysis: Conceptual data models (full E-R)
* Logical Design: Logical data models (relational)
* Physical Design: Physical file and DB design
* Implementation: Database and file definitions

Conceptual Data Modeling

* Part of requirements structuring
* The process of creating a meaningful representation of organizational data
* Describing the structure of organizational data: the objects of interest and the rules governing their interrelationships --> creating one possible view of the organization
* Independent of any technological considerations (for example, not dependent on any specific database management system)

Conceptual Data Model

- A **graphical representation** of the relevant entities and their relationships within the domain of interest
- Uses **business terminology** familiar to the users and other non-technical participants of the analysis process
- Utilizes a **semantically richer** set of concepts than, for example, the relational model does
  - Stronger expressive power

Conceptual Data Model's Purpose

- Forms a fairly stable foundation on which other aspects of the systems development process can be built
  - Fundamental business entities and their relationships are more stable than business processes
- Ensuring various aspects of the integrity of data is vitally important
  - Rules describing relationships between business entities
- Important tool for both **user – systems analyst** and **systems analyst – developer** communication

Conceptual Data Model Offers...

* Understanding the definitions and structure of organizational data is an essential part of understanding the nature of the organization and its business
* Conceptual data modeling forms a stable basis for the development of organizational databases
* Necessary also and particularly when alternative development methodologies are used (prototyping, RAD)

Entity-Relationship (ER) Model

* Created by Dr. Peter Chen and first published in ACM Transactions on Database Systems (1976) "The Entity Relationship Model - Towards a Unified View of Data"
* Later augmented with additional semantic constructs leading to the Enhanced Entity-Relationship model

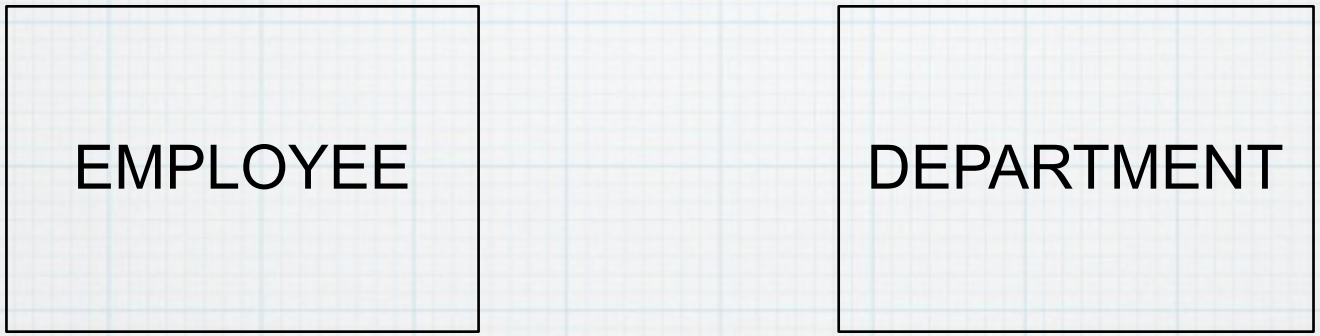
Fundamental E-R Concepts

* Entity
* Attribute
  * Identifier
* Relationship
  * Degree and cardinality
* Associative entity

Entity

* Person, place, object, thing, event, or concept
  * about which the organization wishes to maintain data and
  * that can be uniquely identified
* Name with a singular noun
* Examples: Product, Order, Order line, Student, Course, City
* Entity definition is an essential component of the model
* Essential difference between entity type (class) and entity instance

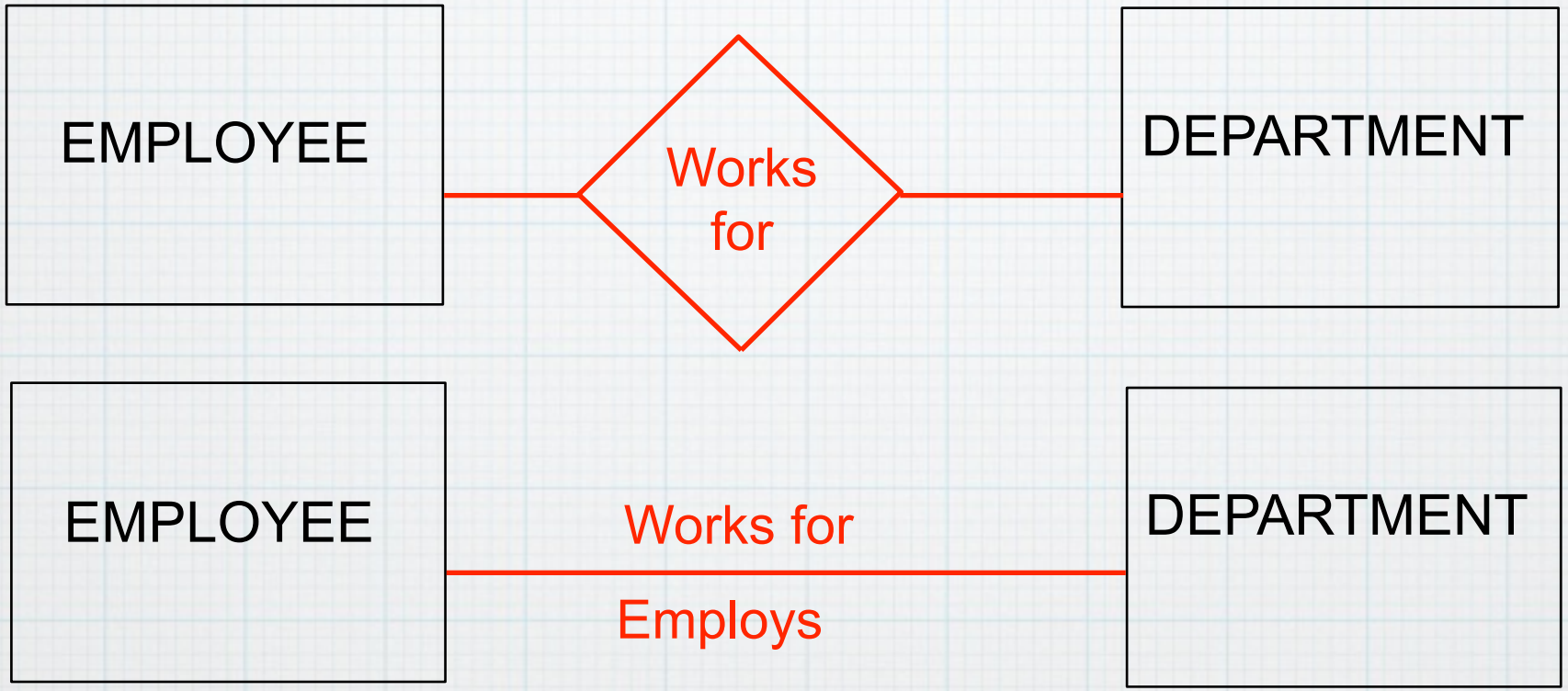
Entity Notation



Relationship

* An association between the instances of one or more entity types
* Named with a verb phrase connecting entities
* Directional
* Event (something, for example, a business transaction, happens) or structural relationship (for example, a whole-part structure)

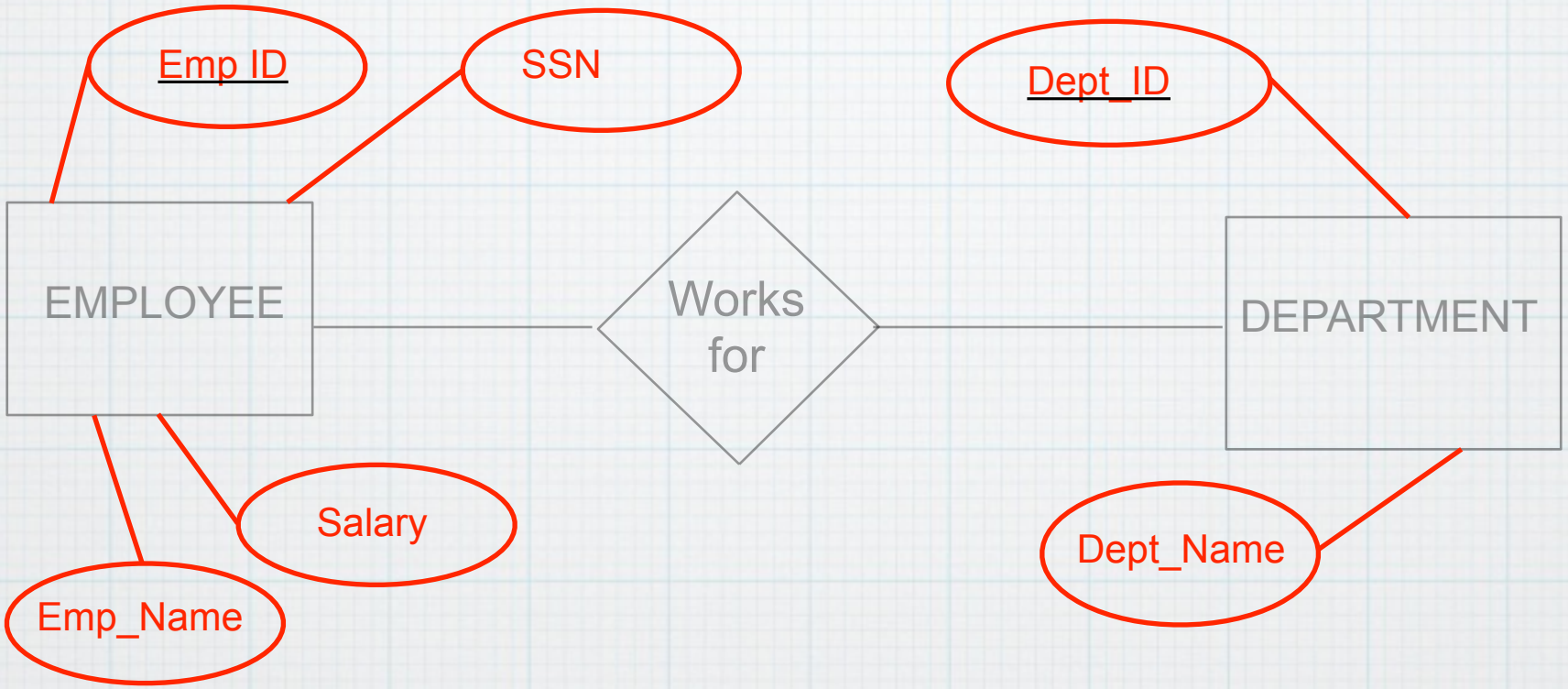
Relationship Notations



Descriptors

- **Attributes** associated with entities and relationships
  - Properties or characteristics describing an entity or a relationship
- **Identifier**
  - An attribute that uniquely identifies each instance of an entity

Attributes


Emp ID
SSN
Dept_ID
EMPLOYEE
Works for
DEPARTMENT
Salary
Dept_Name
Emp_Name

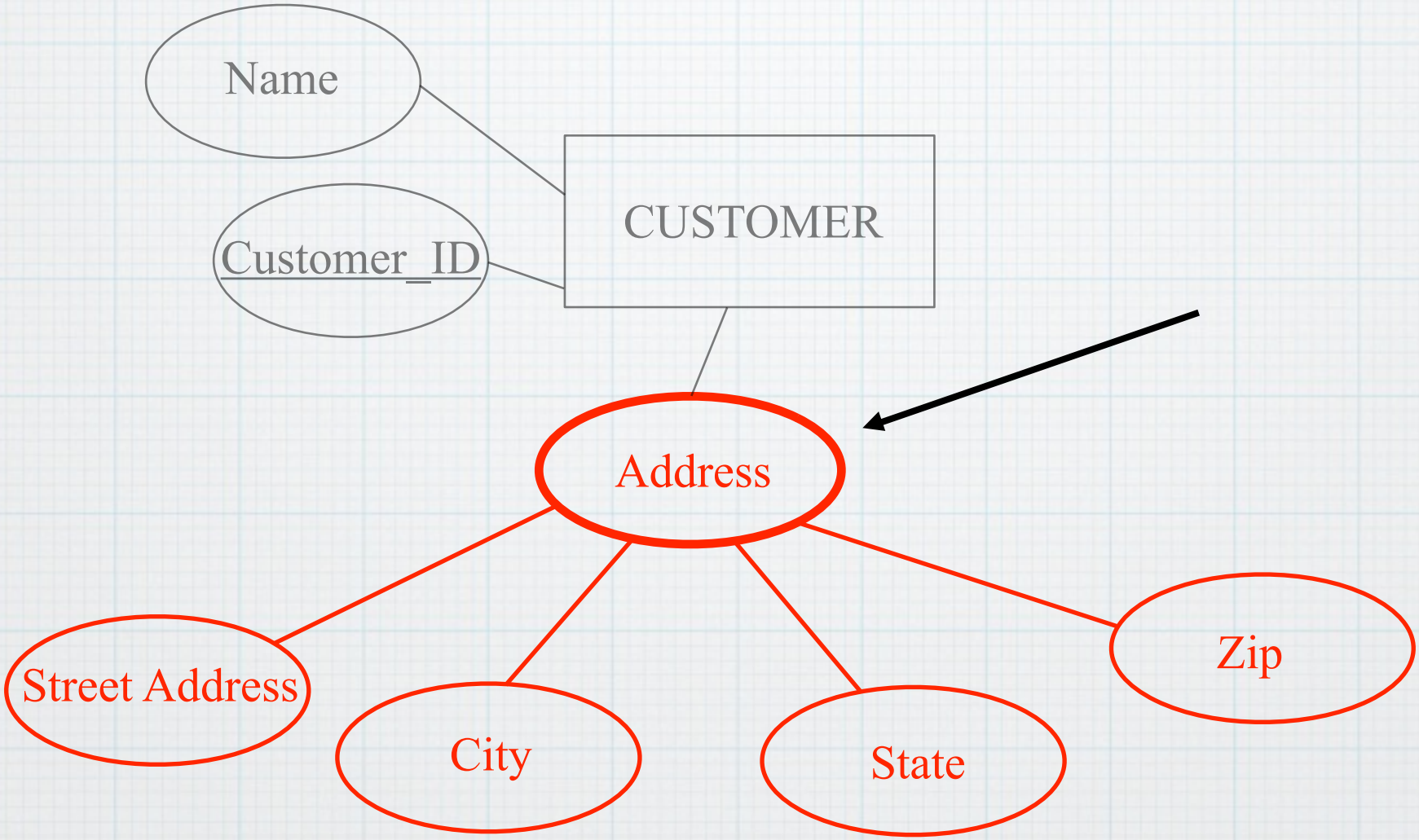
A Good Identifier Choice . . .

* Has a stable value
* Is guaranteed to have valid (not null) values
* Does not have components with a built-in meaning, such as year, region code, product line, etc.
* Has only one part or few parts

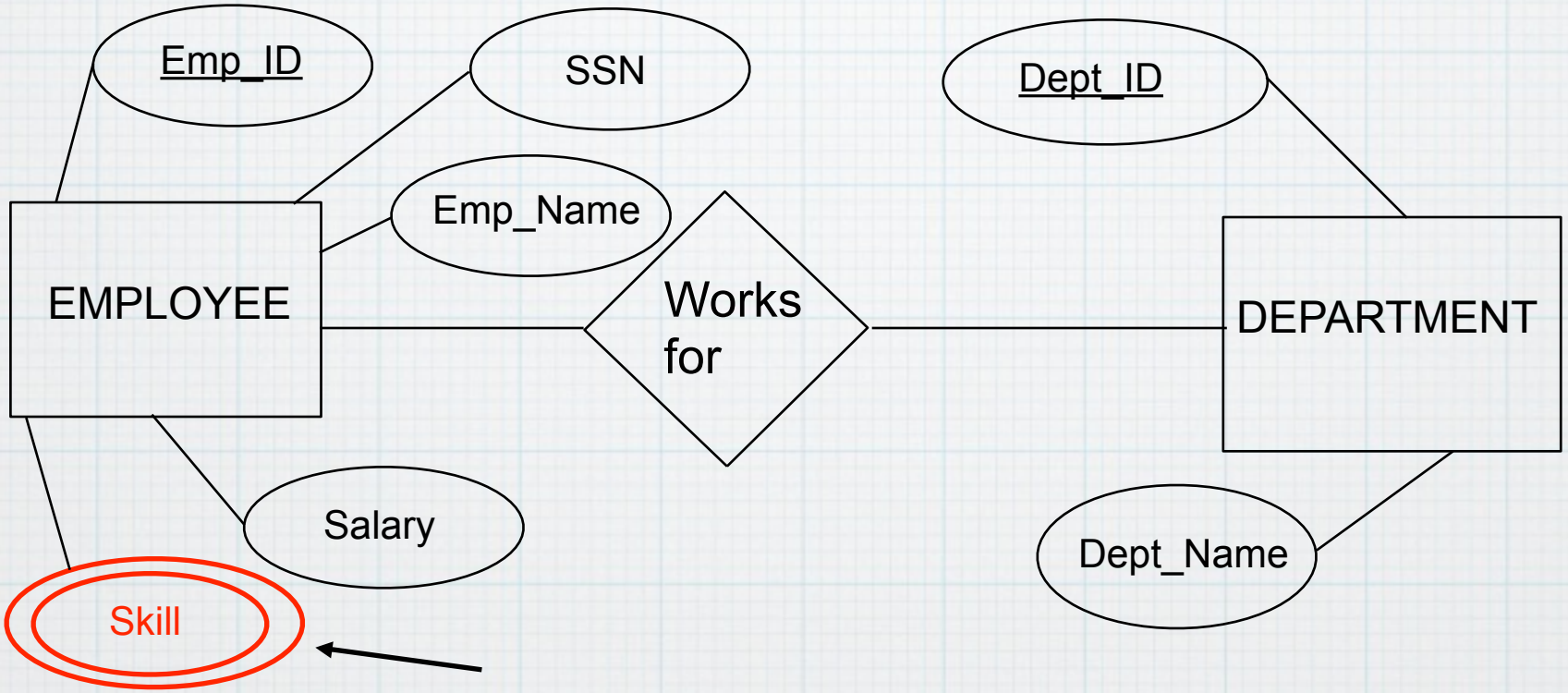
Attribute Constructs

- **Composite** attribute
  - An attribute that can be further divided into components
- **Multivalued** attribute
  - An attribute that can have multiple values for each entity instance
- **Repeating group**
  - A group of two or more multivalued attributes that are logically related

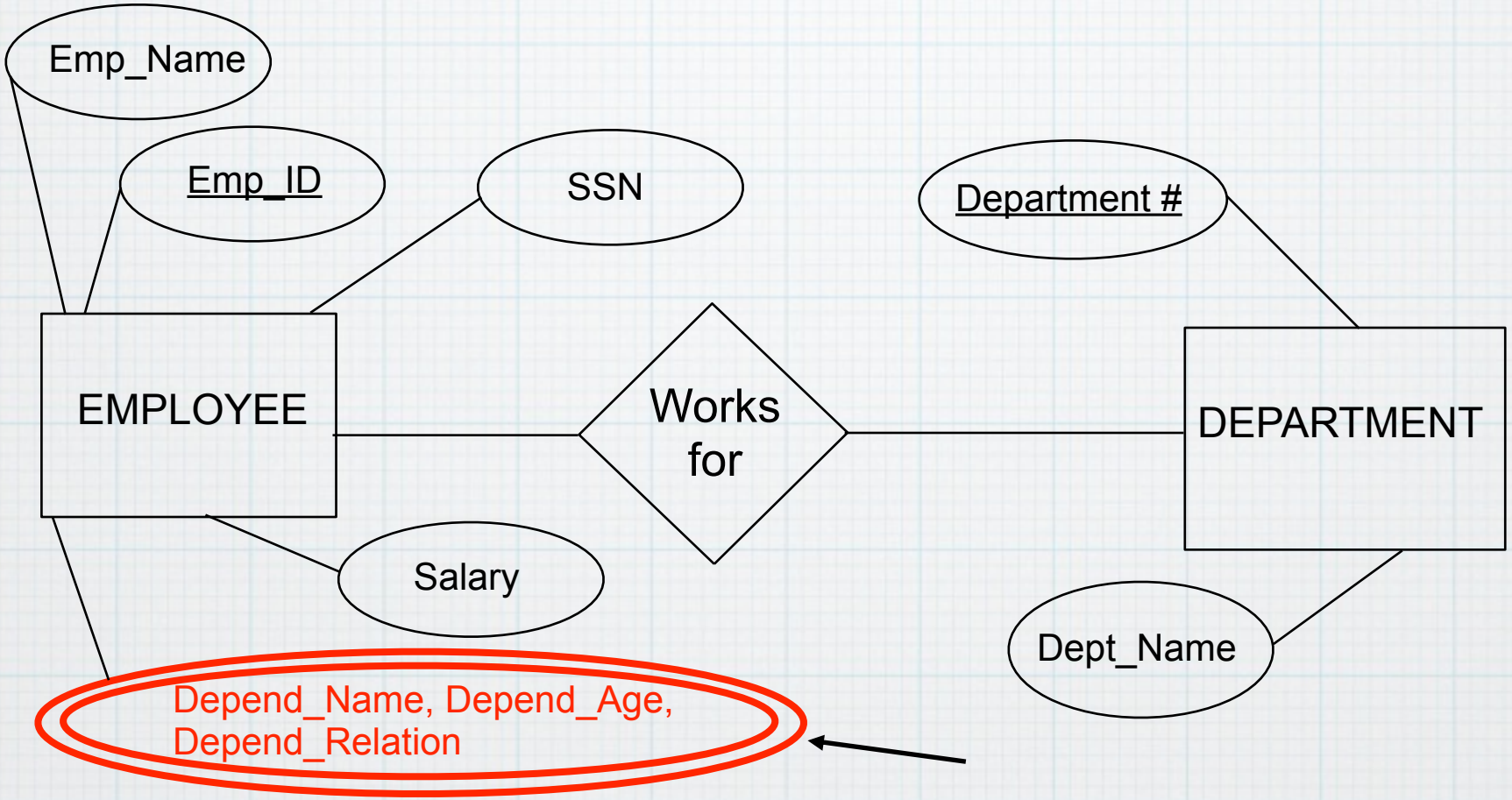
Composite Attribute



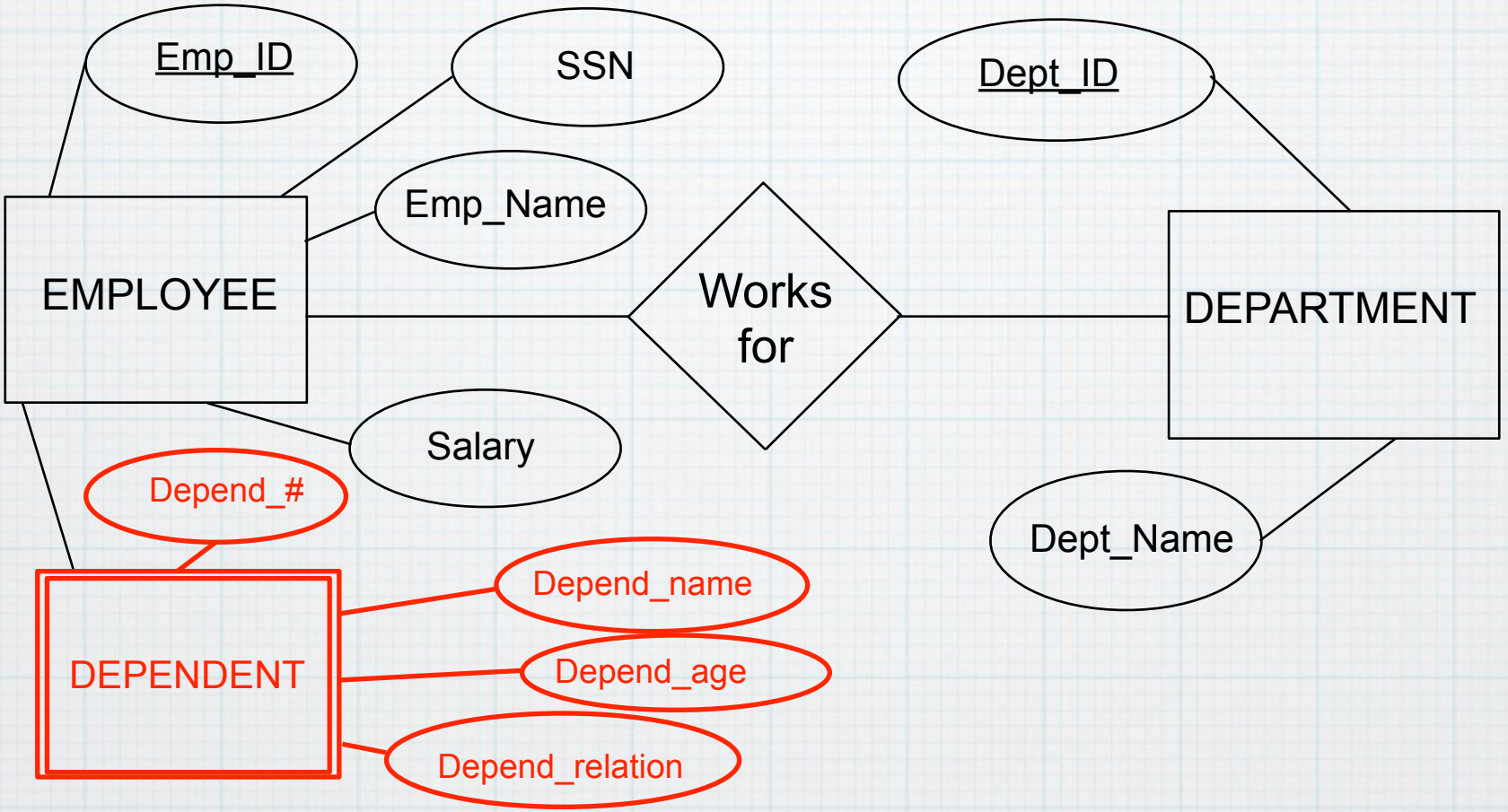
Multivalued Attribute



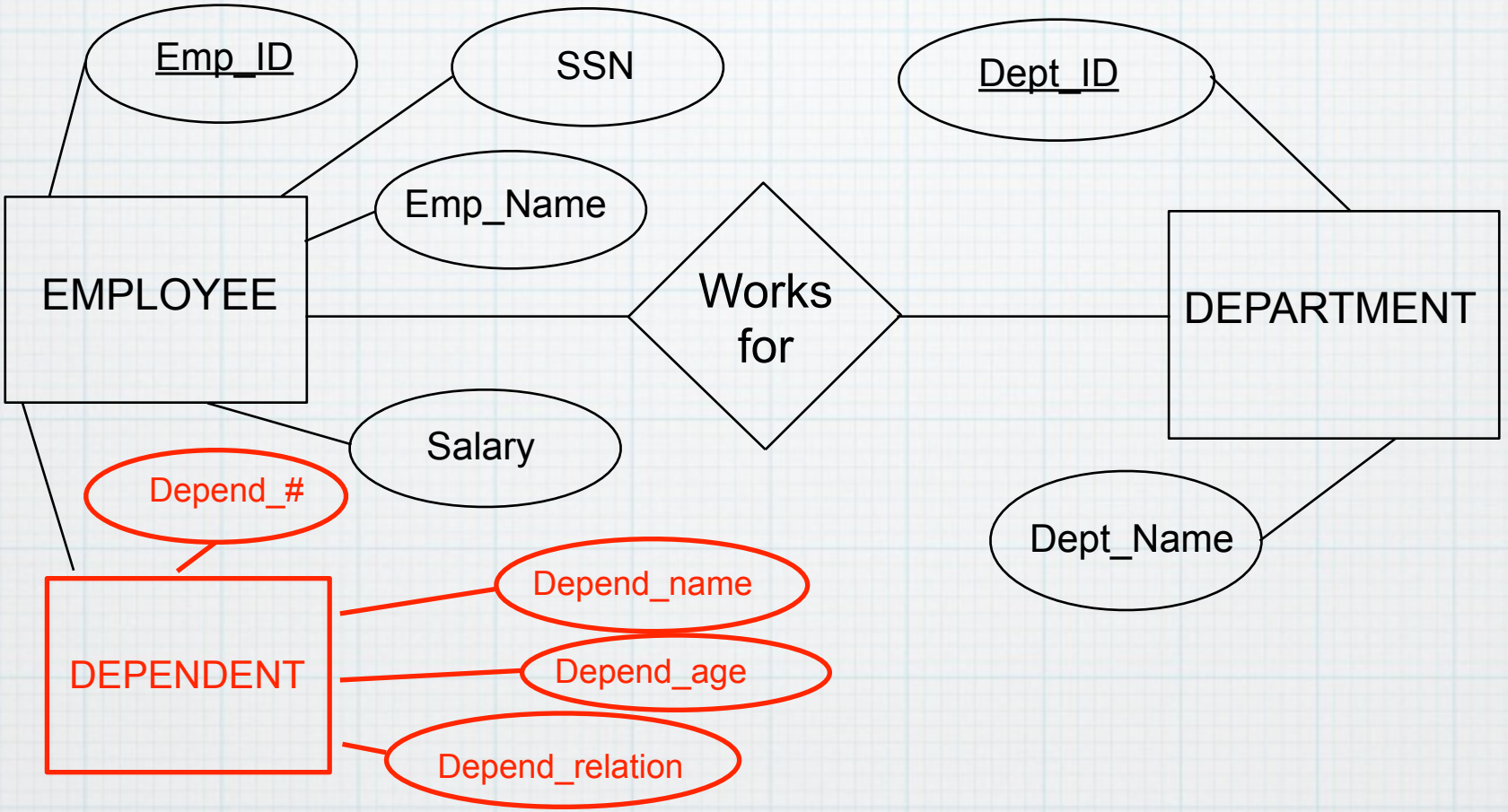
Repeating Group



Preferred Repeating Group Notation



Or...

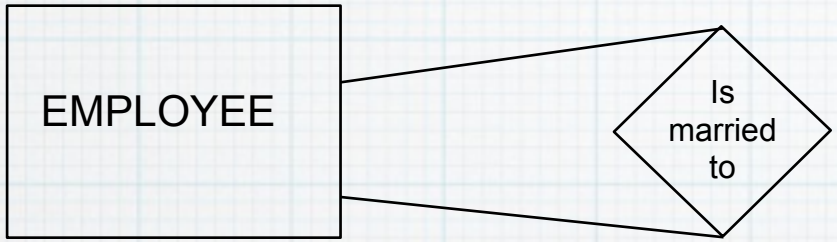

Emp_ID
SSN
Dept_ID
Emp_Name
EMPLOYEE
Works for
DEPARTMENT
Salary
Depend_#
Dept_Name
Depend_name
DEPENDENT
Depend_age
Depend_relation

Relationship Characteristics

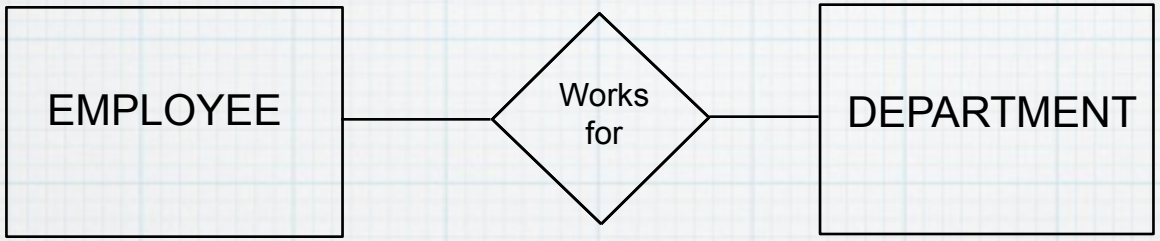
- **Degree** of a relationship
  - The number of entity types involved
  - Most often limited to a maximum of three entities (no theoretical limit)
  - Unary (recursive), binary, ternary

- **Cardinality** of a relationship
  - The lower and upper bounds for the number of instances involved on each side of a relationship
  - Zero, one, many, or a specific number
  - Notation: 1:1, 1:M, M:N

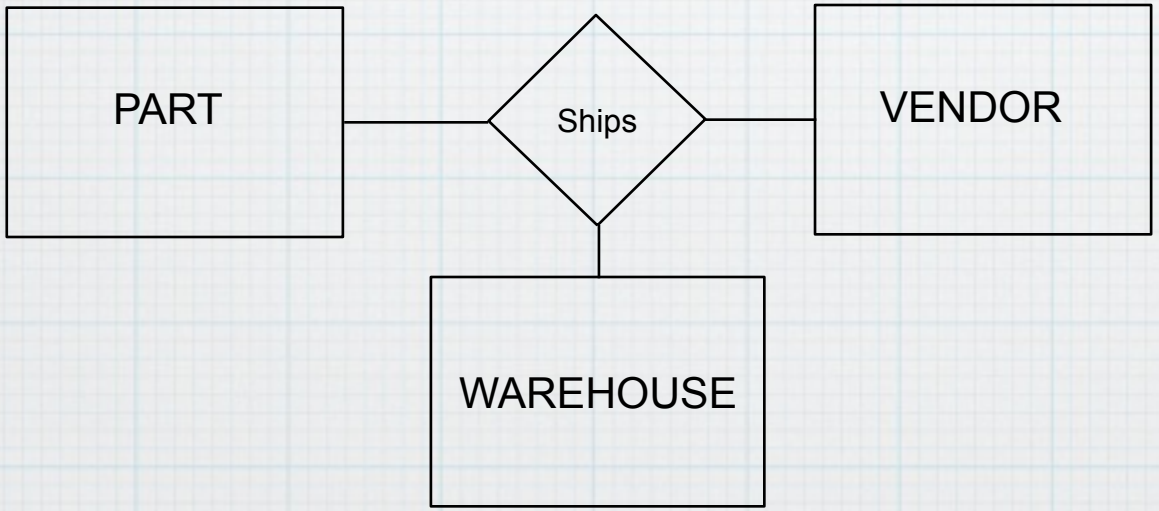
Degree



Unary

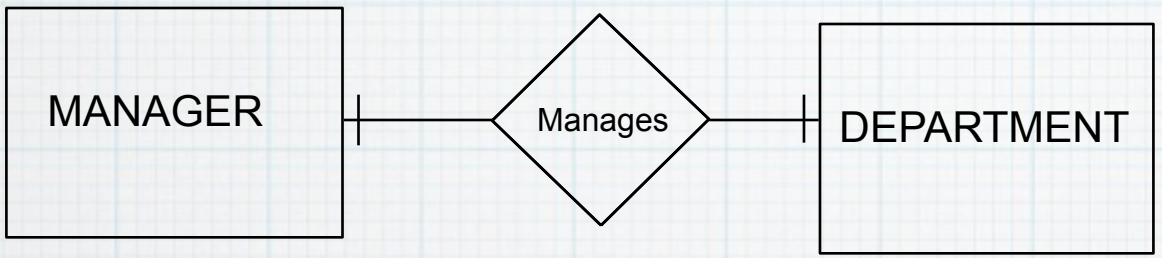


Binary

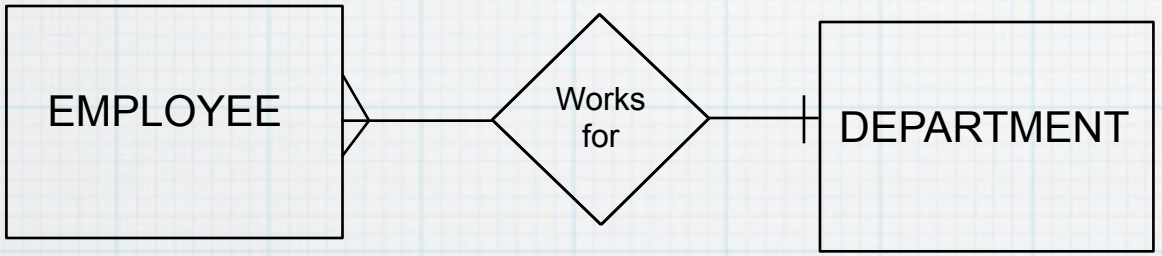


Ternary

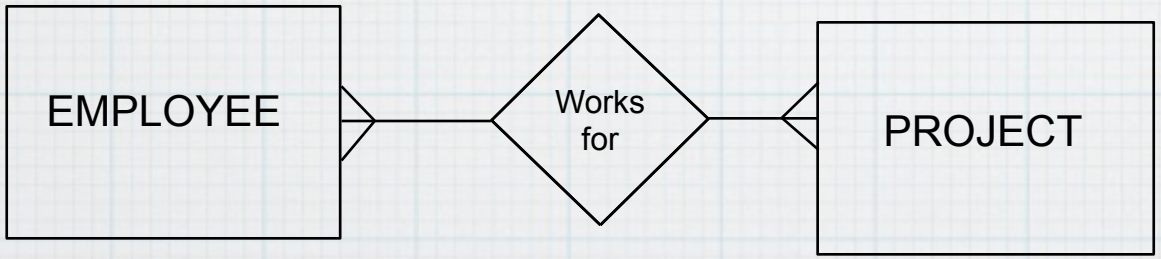
Cardinality (maximums)



One-to-one

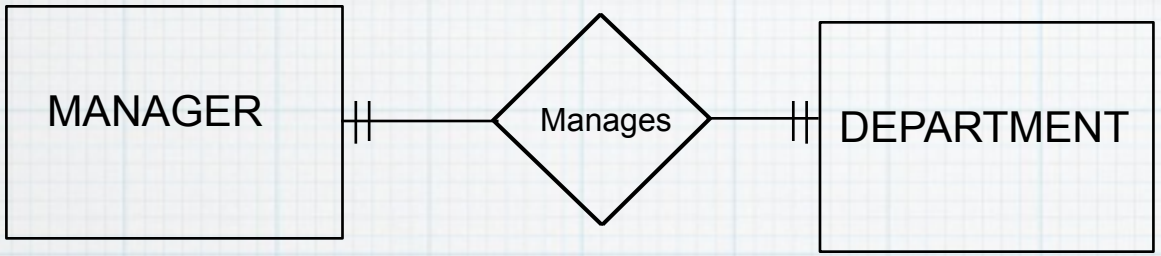


One-to-many

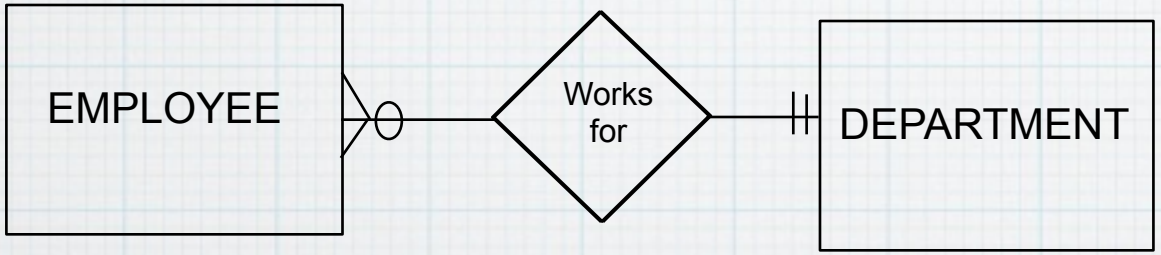


Many-to-many

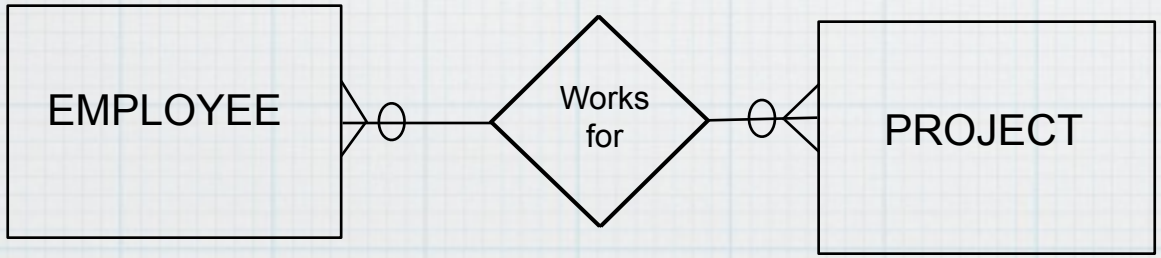
Cardinalities (minimums & maximums)



Mandatory

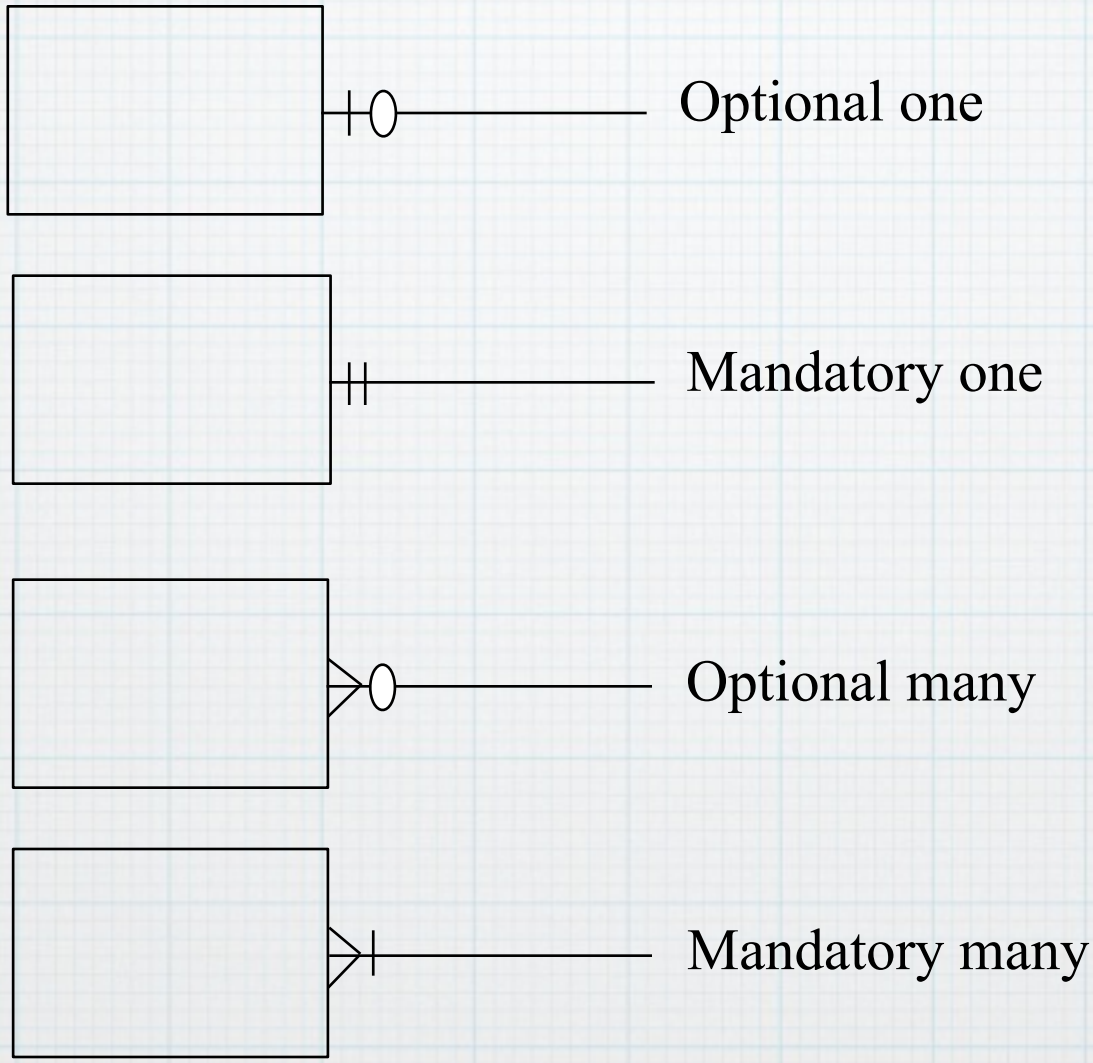


One mandatory,
One optional

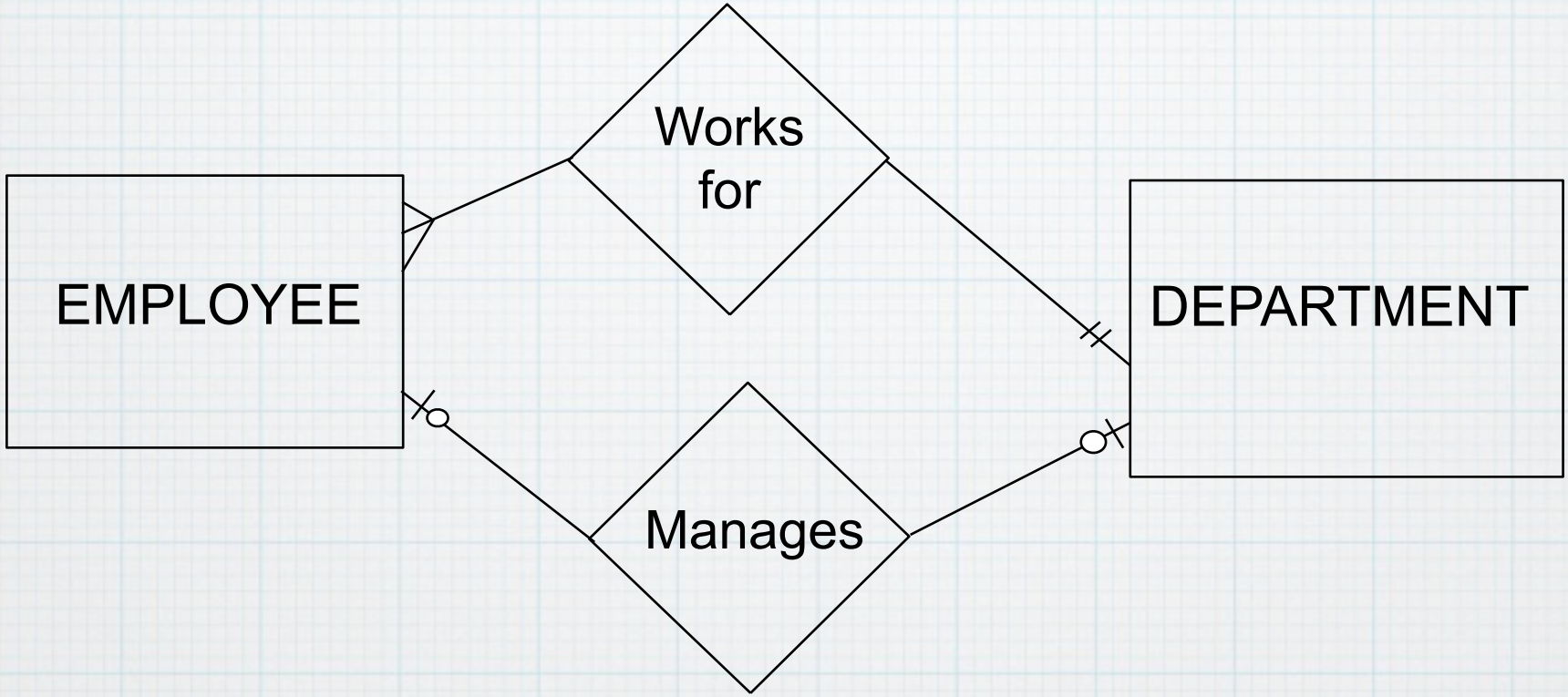


Both optional

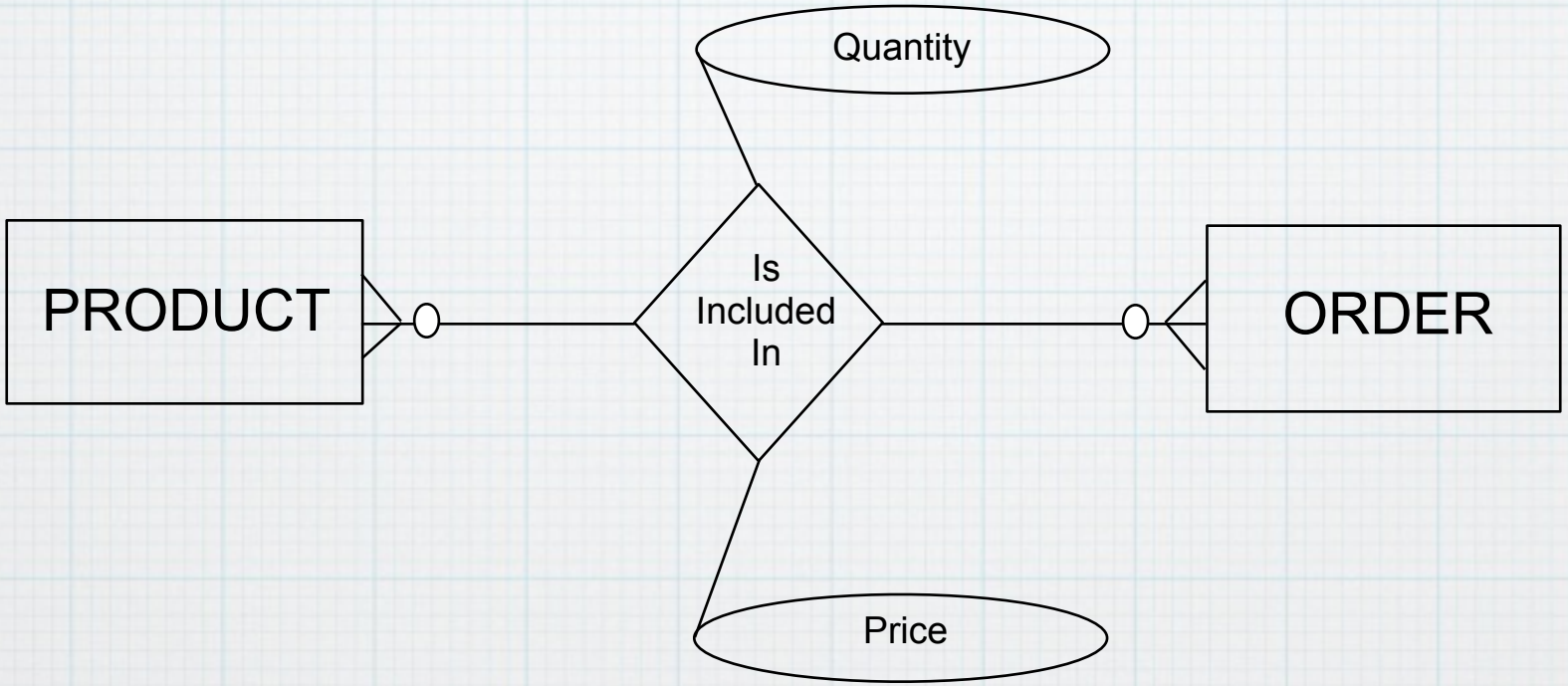
Optional vs Mandatory



Relationship Multiplicity



Attributed Relationships



Possible with many-to-many and one-to-one relationships

Enhanced Entity-Relationship Modeling

Les Waguespack, Ph.D.

Adapted from Topi 2004

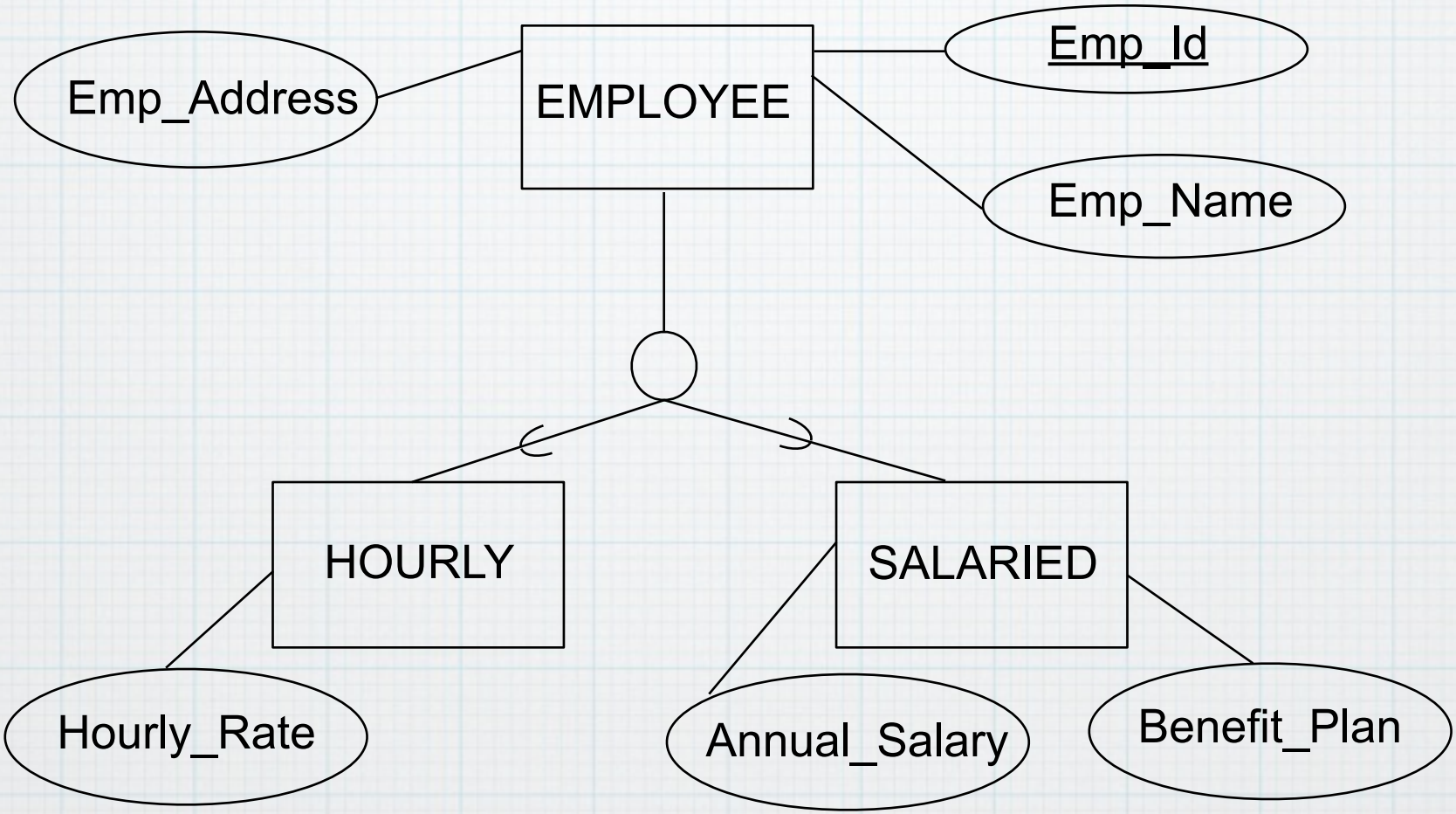
Extending E-R Modeling

* The basic E-R modeling technique was sufficient for a long time, and it became very popular
* It is, however, lacking of several important features, which make it possible to model the real world more accurately
* Enhanced E-R modeling (EER) addresses these concerns by adding several new modeling constructs
  * Our focus: subtype/supertype

Sub/Super-types

- General entity type (supertype) and two or several specialized entity types (subtypes)
- Examples:
  - STUDENT as a supertype and GRADUATE STUDENT and UNDERGRADUATE STUDENT as subtypes
  - EMPLOYEE as a supertype and HOURLY EMPLOYEE and SALARIED EMPLOYEE as subtypes

Sub/Super-type Syntax



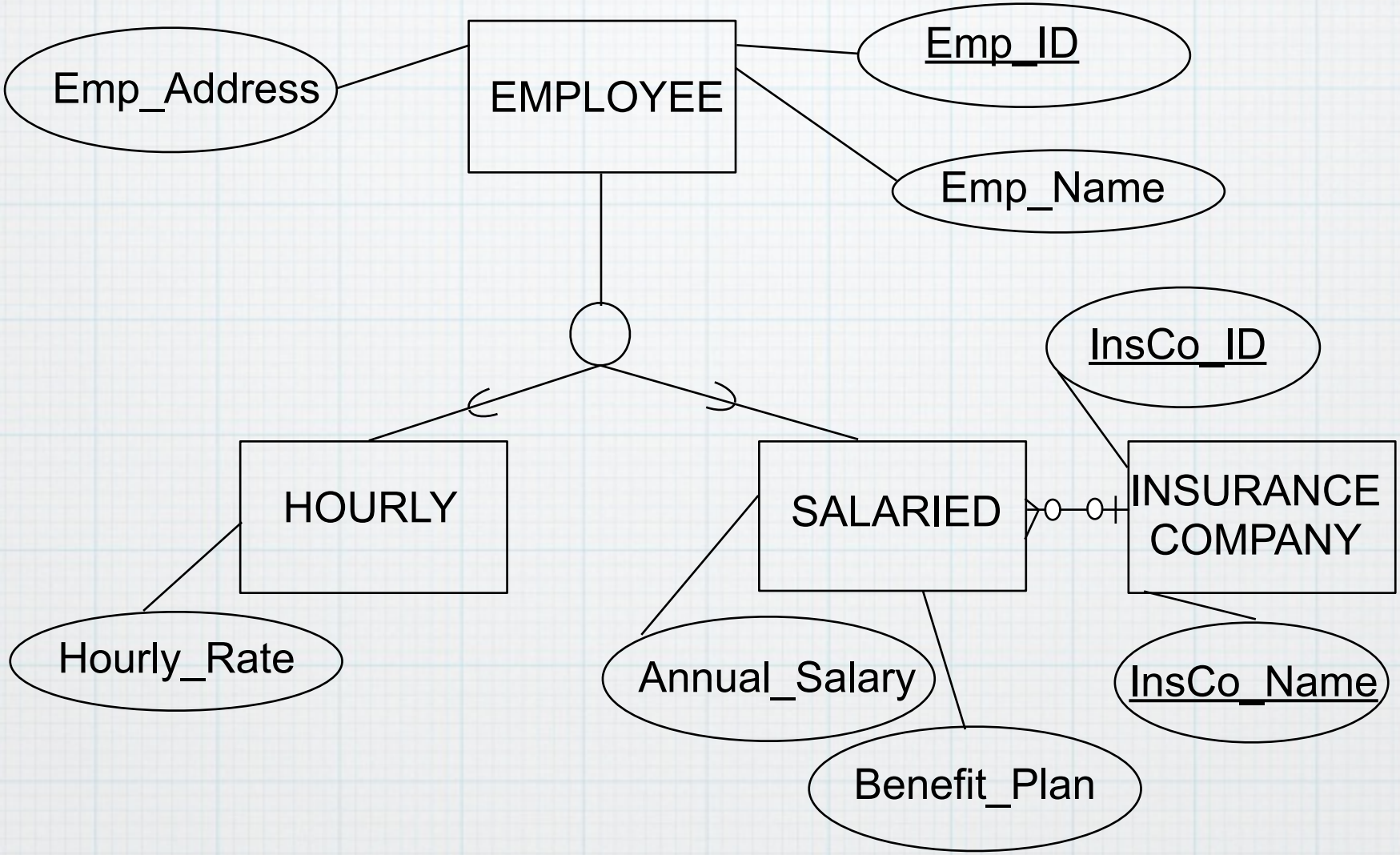
Attribute Inheritance

- An entity instance of a subtype possesses not only values of its own attributes but also values for the supertype
  - For example, a salaried employee Fred Smith has an annual salary of $45,000 and benefit plan #1, but in addition to this, he has attribute values for employee ID (123222), name (Fred Smith), and address (1242 Smith Rd, Anytown, IL)
- A subtype instance must also be an instance of the supertype

Sub/Super-type Opportunities

* Some attributes apply only to some of the entity instances
  * For example, the attribute Benefit Plan applies only to salaried employees
* Some instances of an entity participate in relationships which are unique to that subset
  * For example, there could be a relationship between the Salaried Employee subtype and Insurance Company signifying a health insurance relationship

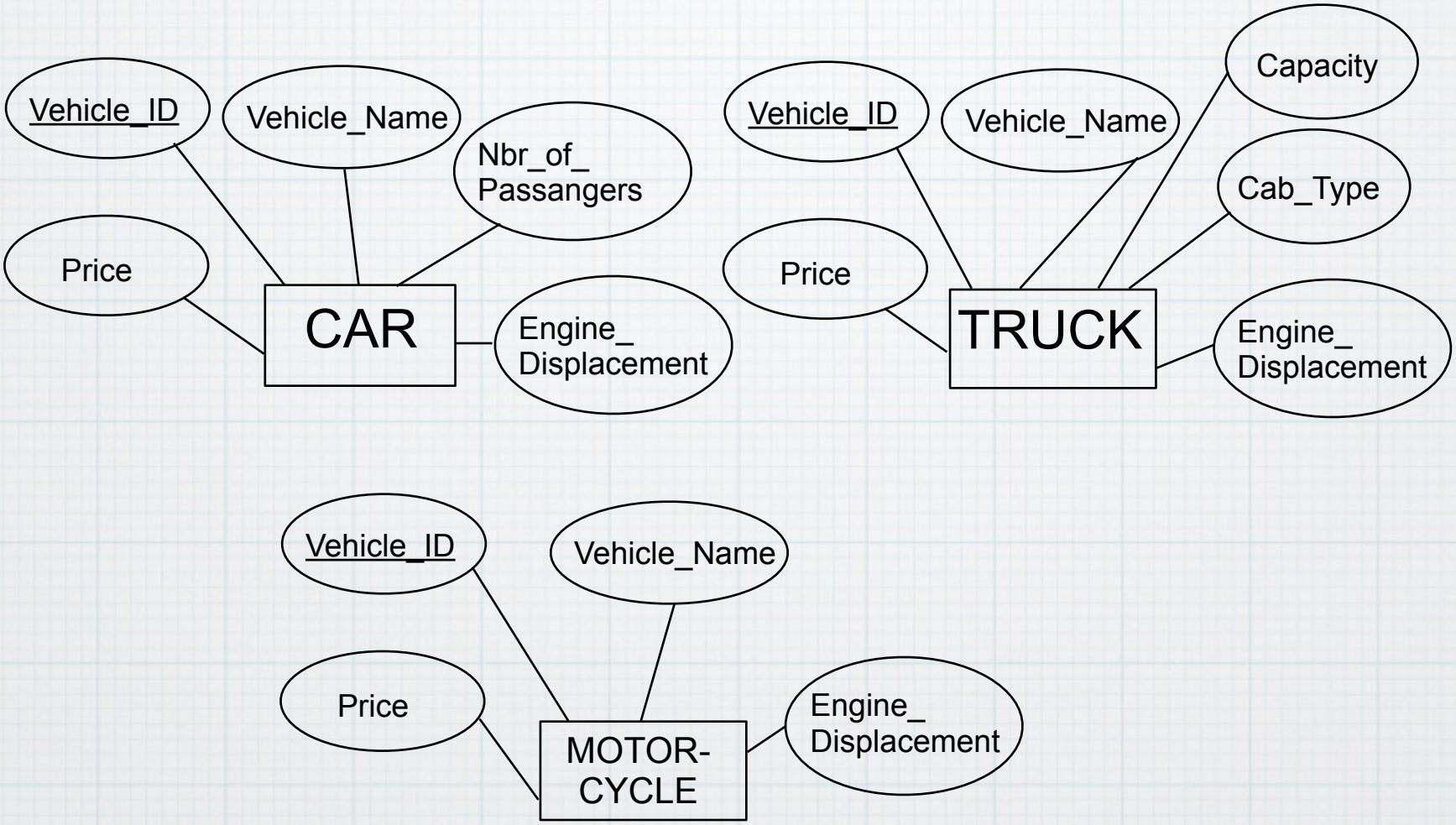
Specialized Relationship



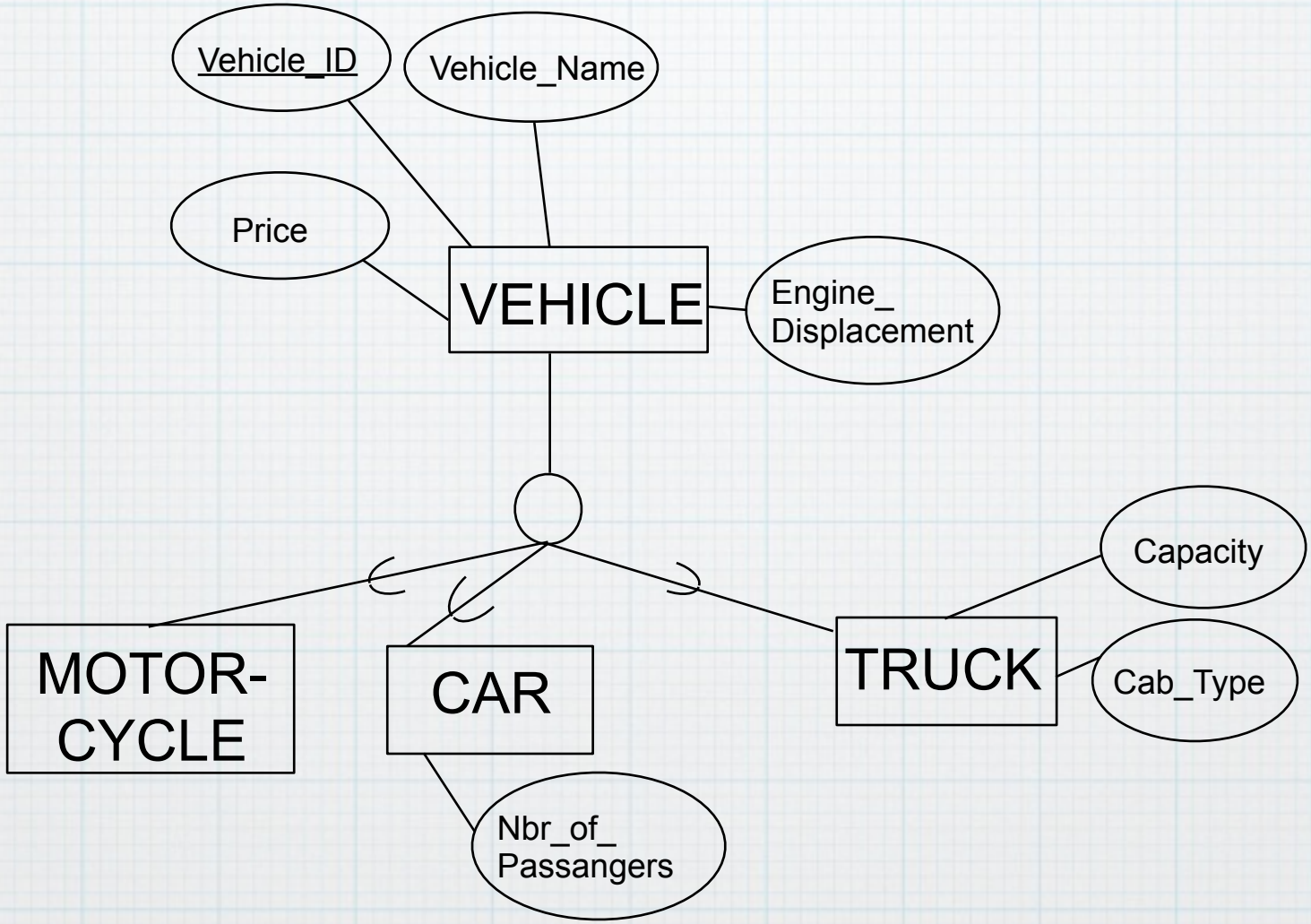
Mental Models for Developing Super/Sub-type Relationships

* Generalization
  * A bottom-up process for identifying general characteristics of several specialized entity types
* Specialization
  * A top-down process of identifying subtypes based on an already identified supertype

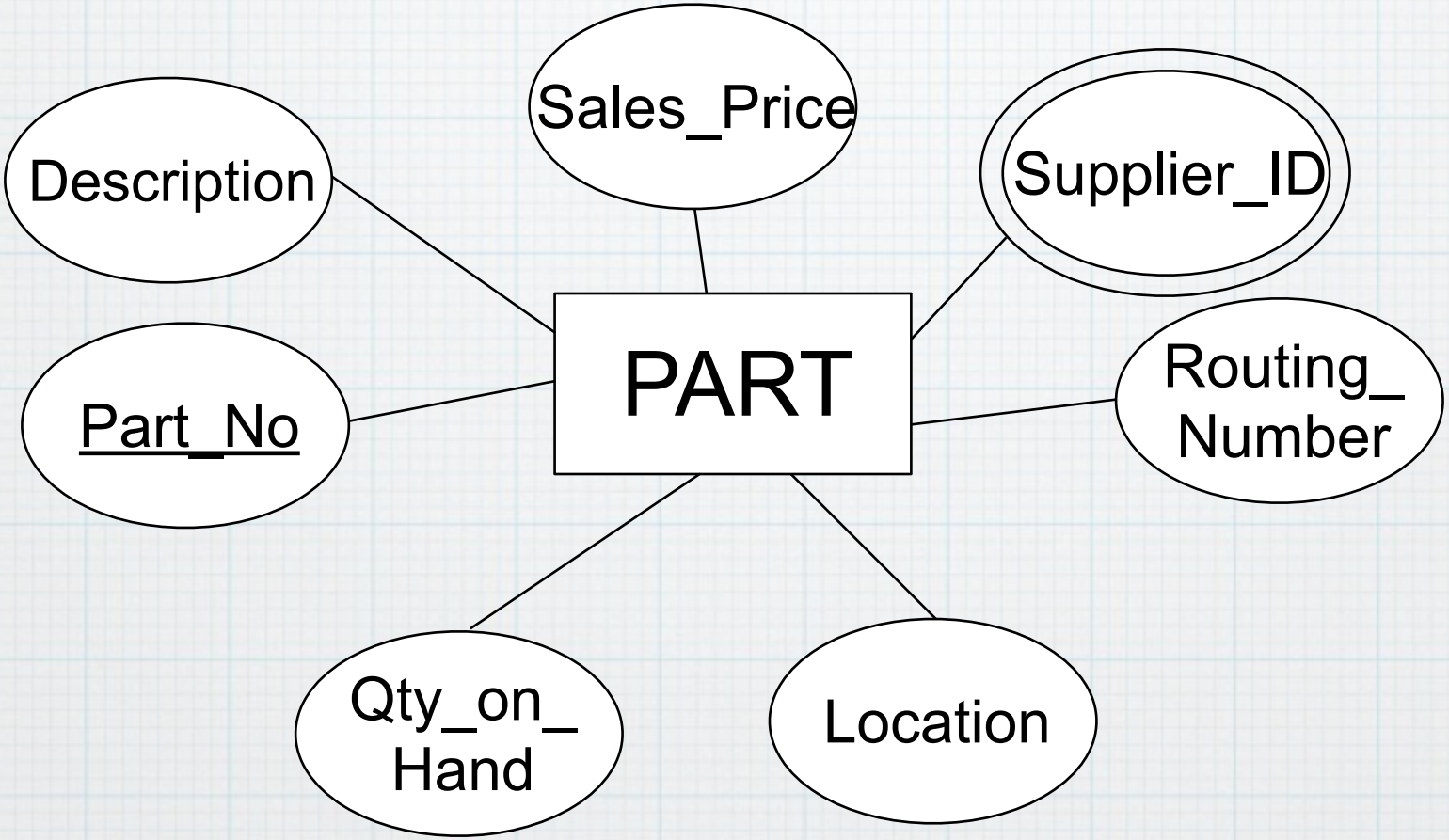
Three Separate Entity Types



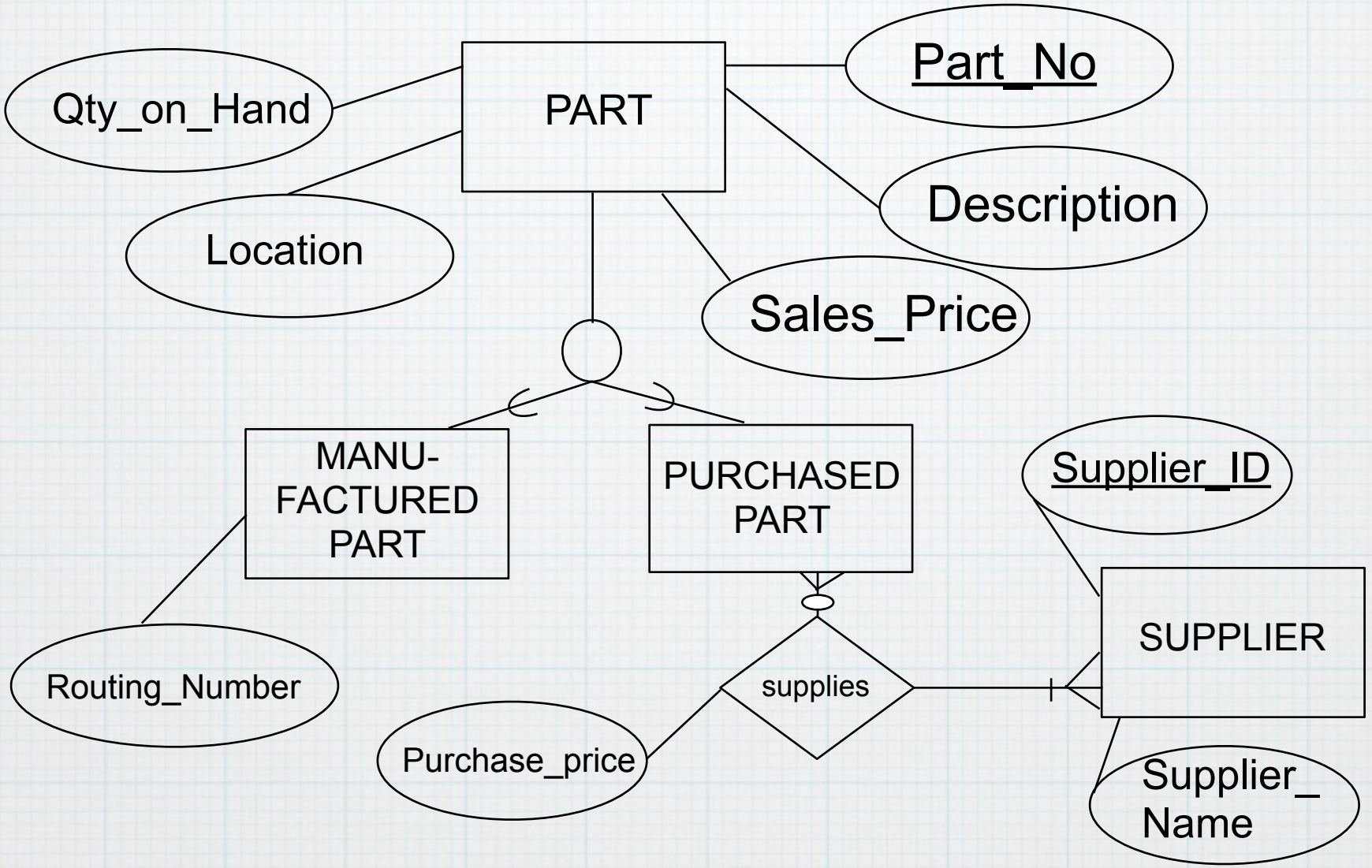
Generalized VEHICLE Supertype



Entity Type PART



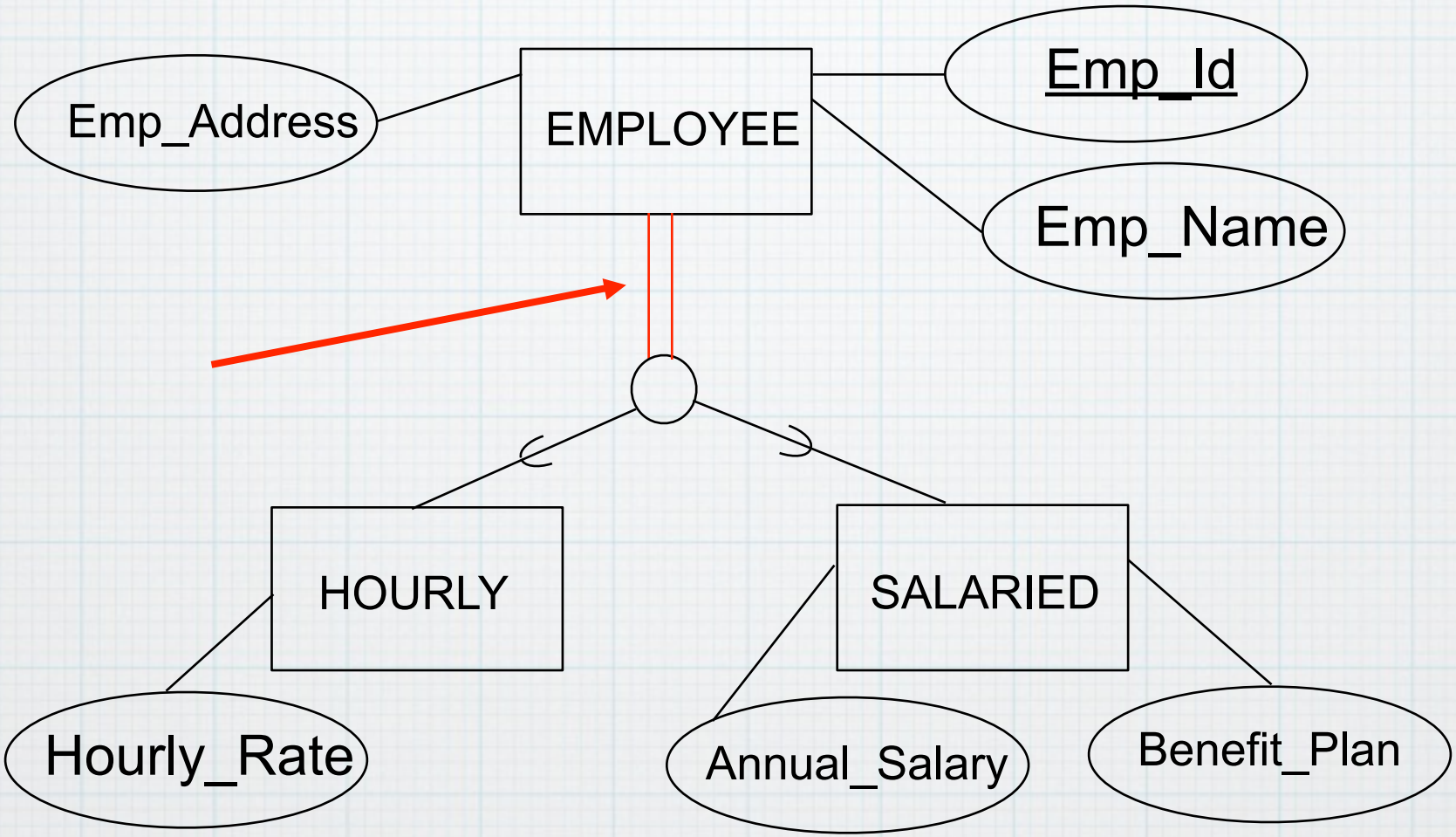
PART Divided Into Specialized Types



Constraints

- Completeness constraint
  - Does every instance of the supertype also have to be an instance of one of the subtypes
  - Total specialization rule: Yes
  - Partial specialization rule: No

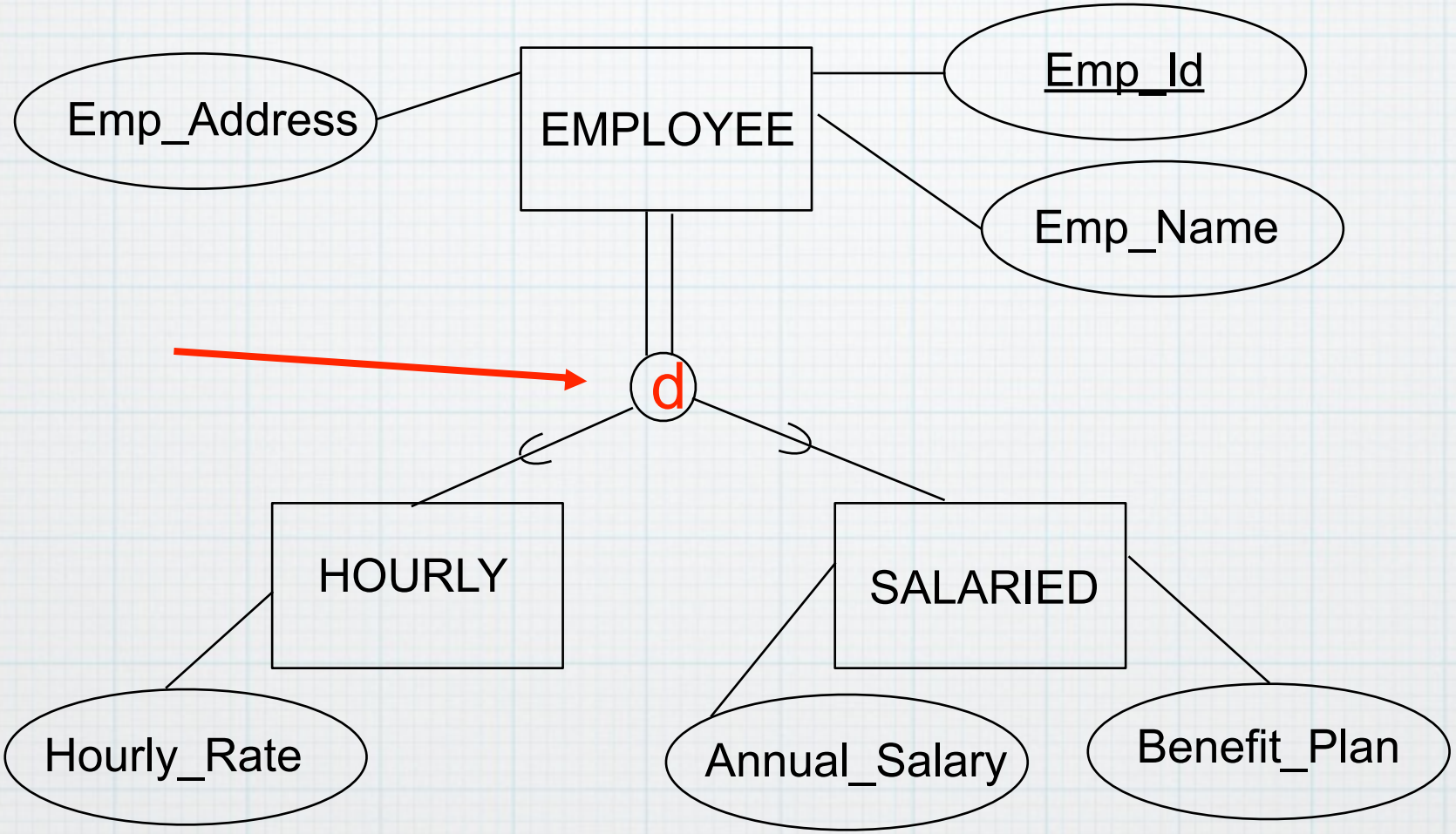
Notation for Total Specialization



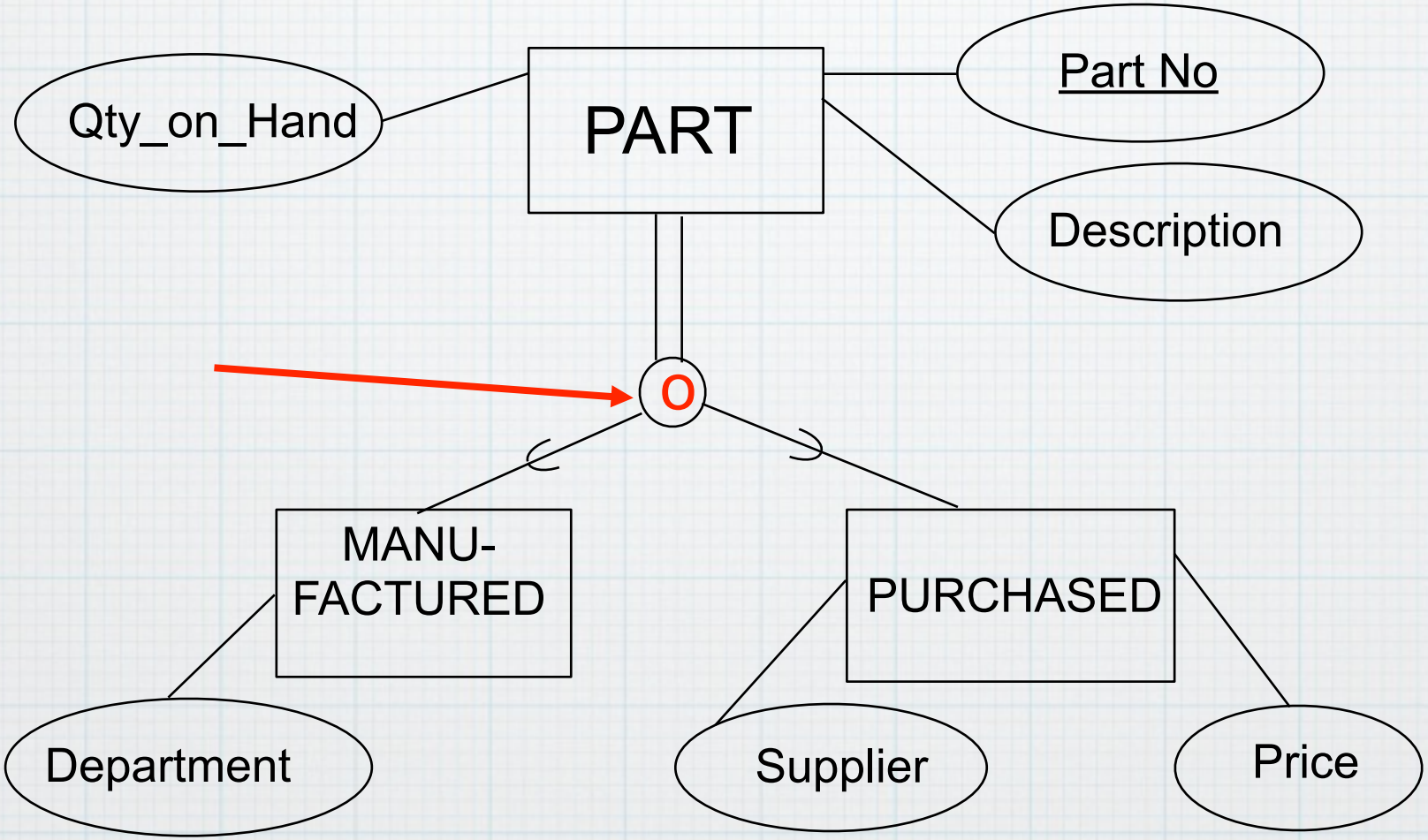
Constraints (more)

- Disjointness constraint
  - Can an instance of the supertype be an instance of multiple subtypes simultaneously?
  - **Disjoint rule**: at any given time, an instance of the supertype can be an instance of only one of the subtypes
  - **Overlap rule**: an instance of the supertype can be an instance of multiple subtypes at a particular time

Disjoint Notation



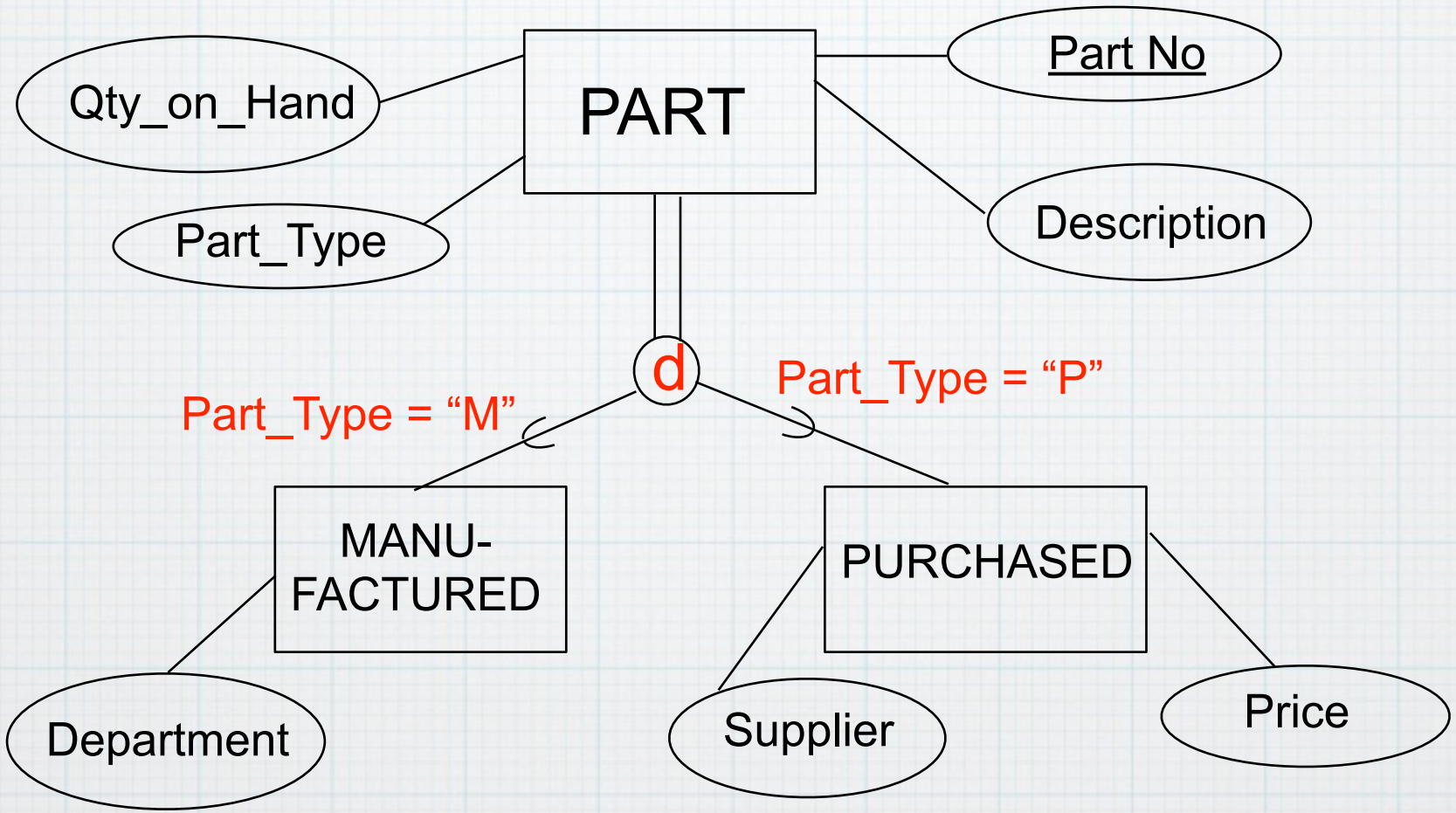
Overlap Notation



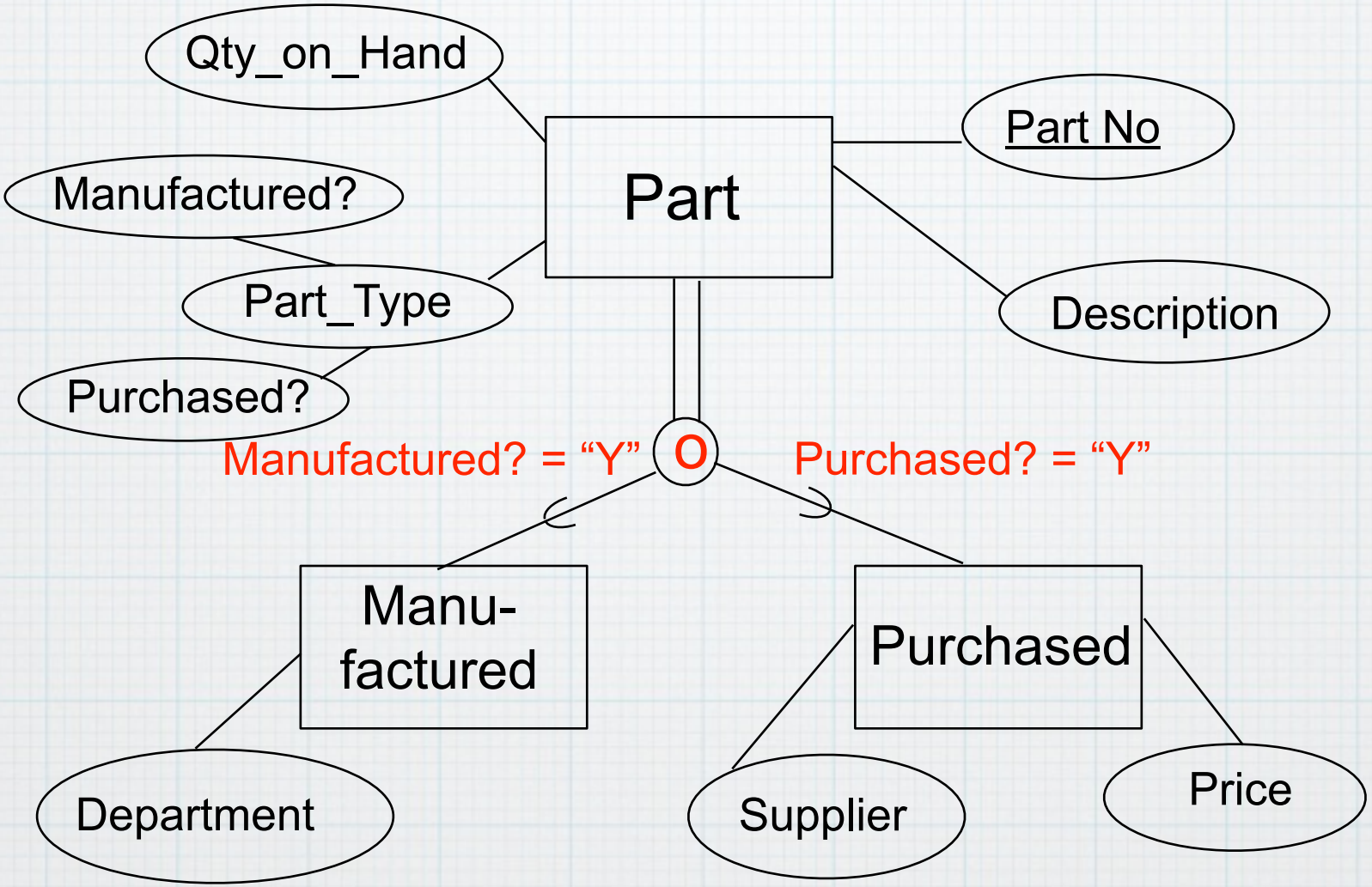
Subtype Discriminator

* An attribute of the supertype the values of which determine the subtype (or subtypes) to which a particular instance of the supertype belongs
  * For example, entity type Employee could have an attribute Employee_Type which can get three values ("H","S", or "C"), each of which corresponds to a particular subtype (Hourly, Salaried, and Contractor)

Subtype Discriminator (disjoint)



Subtype Discriminator (overlap)



Filling Out the Conceptual Model

- Documentation via the data dictionary
- Define:
  - Entities
    - Type (Regular or Weak), written description, identifier
      - regular - exist on their own accord
      - weak - depend on an "owner" or "superior" entity
  - Relationships
    - Type (degree & cardinality), written description, attributes
  - Attributes
    - Domain, written description, are null values accepted?

Domain Definitions

- A set of possible values an attribute can assume
- Characteristics
  - Name
  - Data type (e.g., integer, currency, character/string, date etc.)
  - Length (if applicable)
  - Range (if applicable)
  - List of allowable values (if applicable)
  - Format

Logical Database Design

Les Waguespack, Ph.D.

Adapted from Topi 2004

Logical Database Design

- The process of transforming the conceptual data model into a **logical data model**
- Using the E-R diagram to create a set of well-structured relations that can be later implemented with a relational DBMS
- Emphasis on the relational data model
  - The most common data model in current database applications
  - Relational principles are widely applicable

"Well-Behaved" Relations

* Minimal redundancy
* No anomalies of any type:
  * Insertion
  * Deletion
  * Modification

* Anomaly: an error that happens when a user attempts to update a table that is ill-structured and contains redundant data

"Poorly Behaved" Relation

Part Number	Description	Vendor_ Name	Address	Unit Cost
1234	Logic Chip	Fast Chips	Cupertino	10.00
1234	Logic Chip	Smart Chips	Phoenix	8.00
5678	Memory Chip	Fast Chips	Cupertino	3.00
5678	Memory Chip	Quality Chips	Austin	2.00
5678	Memory Chip	Smart Chips	Falstaff	5.00

"Well Behaved" Alternative

PART

Part_number	Description
1234	Logic Chip
5678	Memory Chip

VENDOR

Vendor_name	Location
Fast Chips	Cupertino
Quality Chips	Austin
Smart Chips	Phoenix

PART_VENDOR

Part_number	Vendor_name	Price
1234	Fast Chips	10
1234	Smart Chips	8
5678	Fast Chips	3
5678	Quality Chips	2
5678	Smart Chips	5

Relational Terminology

* Relation: a named two-dimensional table of data
  * Consists of named columns and unnamed rows
  * Represent entities and certain relationships
* Columns: represent attributes of entities
  * Correspond to **fields** in older terminology
* Rows: represent instances of entities
  * Correspond to **records** in older terminology

Relational Properties

* Entries in columns are **atomic**
* All entries in a specific column are from the same **domain**
* Each row is **unique**
* The sequence of columns is insignificant
* The sequence of rows is insignificant

Examples: Relation Notation

* Employee(Emp_ID, Emp_Last_Name, Emp_First_Name, Emp_Phone, Emp_Address, Emp_Dept)

* Department(Dept_ID, Dept_Name, Dept_Building, Dept_Fax)

Keys

- **Candidate key**: An attribute (or a combination of attributes) that uniquely identifies each instance of an entity type
- **Primary key**: The candidate key that has been chosen to be the unique identifier of a row
- **Composite key**: a primary key that consists of more than one attribute
  - For example:
    - Order_Line(Order_ID, Product_ID, Quantity, Price)

Keys as "Connectors"

- Foreign key: an attribute that creates a relationship with another relation by containing values of the primary key of that other relation
- Example:
  - Employee(Emp_ID, Emp_Last_Name, Emp_First_Name, Emp_Phone, Emp_Address, Emp_Dept)
    - Emp_Dept references Department(Dept_ID)
  - Department(Dept_ID, Dept_Name, Dept_Building, Dept_Fax)

Integrity Constraints

- Domain constraints
  - The values in each of the columns of a relation have to come from a certain domain
- Entity integrity
  - No primary key value can be null
- Referential integrity
  - A foreign key value has to be either null or a valid primary key value in the relation to which the foreign key refers

Reaching LOGICAL from CONCEPTUAL

- **Step 1**: Represent **regular entities**
  - Easy: each regular entity type becomes a relation, each attribute of the entity type becomes a column, and the primary key of the entity becomes the primary key of the relation
  - Break possible composite attributes into components
  - Make sure that multi-valued attributes are transformed into separate relations

Reaching LOGICAL from CONCEPTUAL

* **Step 2**: Represent **weak entities**
  * The owner entity type is converted normally
  * The other attributes of the weak entity type are converted normally, and in addition, the primary key of the owner entity type is added to the relation (and if necessary, another new column for the partial identifier, i.e., the separator of multiple instances of the weak entity)

Reaching LOGICAL from CONCEPTUAL

- **Step 3**: Represent binary relationships
  - One-to-many:
    - First, create relations normally
    - Next, include the primary key attribute of the one-side of the relationship as a foreign key in the many-side relation
  - Many-to-many:
    - First, create relations normally
    - Next, create a new relation to represent the relationship (with primary keys of the original tables as foreign keys, the combination of which becomes the primary key of the new relation)
  - One-to-one:
    - First, create relations normally
    - Next, include the primary key attribute of the mandatory one-side of the relationship as a foreign key in the optional one-side relation

Reaching LOGICAL from CONCEPTUAL

- **Step 4**: Represent associative entities
  - If an associative entity does not have a primary key:
    - Treat as a many-to-many relationship
  - If an associative entity has been assigned a primary key:
    - Treat as a many-to-many relationship but use the primary key of the associative entity as the primary key of the associative relation, too (you still need the primary keys of the main relations as foreign keys in the associative relation)

Reaching LOGICAL from CONCEPTUAL

- **Step 5**: Represent unary relationships
  - One-to-many:
    - Use a recursive foreign key (an attribute that references the primary key of the table itself)
  - Many-to-many
    - First, create the main table
    - Next, create another table for representing the relationship, which has as its primary key a composite attribute that refers twice to the main table

Reaching LOGICAL from CONCEPTUAL

- **Step 6**: Represent ternary relationships
  - First, represent all regular entities with normal relations
  - Next, convert the associative entity into an associative relation with a composite primary key consisting of the keys of all the main entities

Reaching LOGICAL from CONCEPTUAL

- **Step 7**: Represent supertype/subtype relationships
  - Create a separate relation for the supertype and each of its subtypes
  - Give the relation representing the supertype all the common attributes, including the primary key
  - Give the relations representing the subtypes the primary key of the supertypes and the unique attributes
  - Make sure that you have a subtype discriminator

Data Flow Diagrams

Les Waguespack, Ph.D.

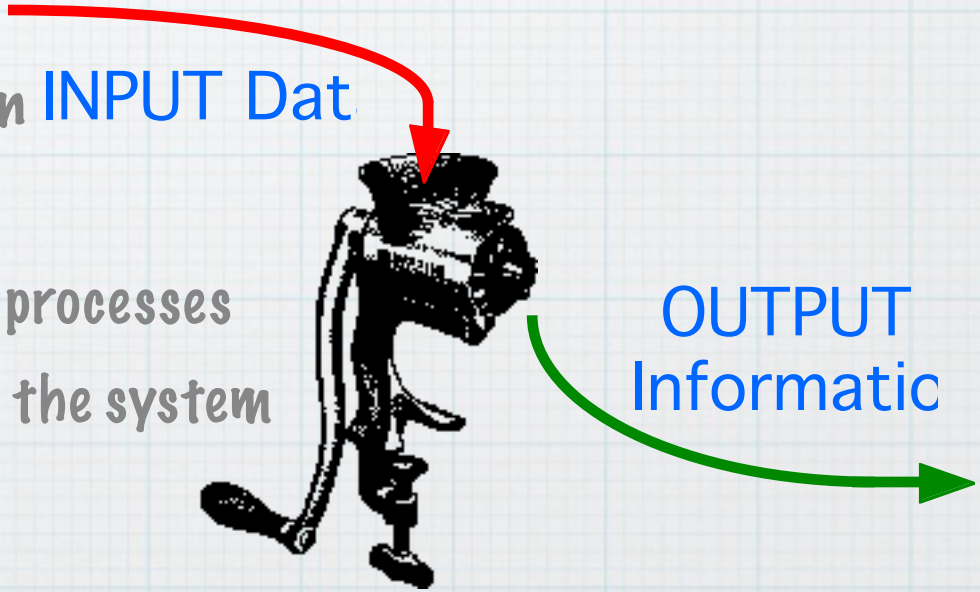
some slides adapted from Heikki Topi

System Modeling

- Data modeling
  - Entity-Relationship Diagrams (ER, EER)
- **Process modeling**
  - Modeling how data flows between and is transformed within business processes
  - Data Flow Diagrams (DFDs)
- Logic modeling
  - Modeling processing logic and timing of events within processes
  - Structured English, decision tables, decision trees, state-transition diagrams and tables

Data Flow Diagramming

* Process Model Specification Tool by (Gane & Sarson) focusing on:
  * sources of information
  * destinations of information
  * paths of information flow
  * information transforming processes
  * repositories of data within the system



Four Types of DFD's

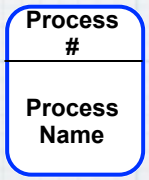
- Current Physical DFD
  - Description of the current system implementation
- Current Logical DFD
  - Description of the current system without technical details; focus on the "what" question
- **Future Logical DFD**
  - The abstract model of the new system
- Future Physical DFD

DFD Symbols Gane & Sarson

* External Entity
  * originator of data or receiver of information
  * aka "sources" and "sinks"

* Data Store
  * a repository for data in the system

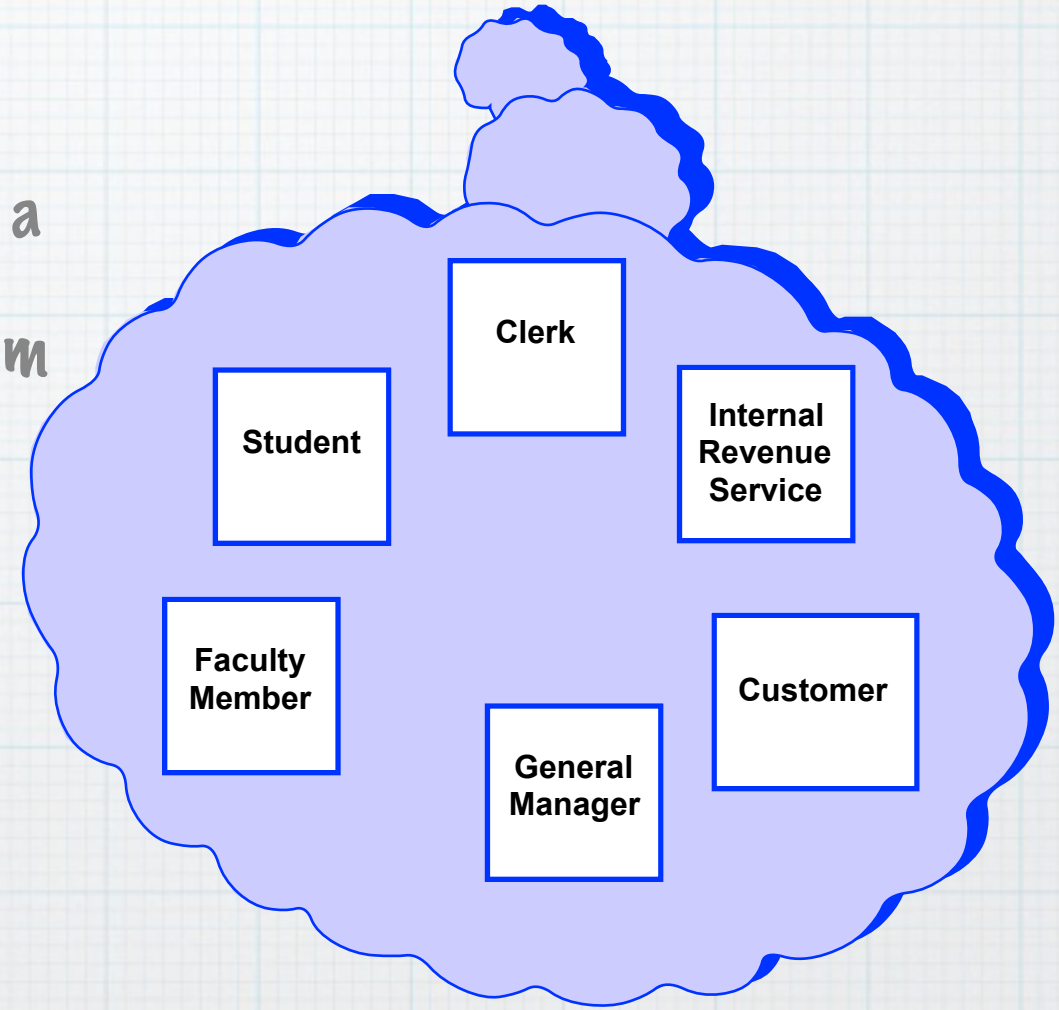
* Process
  * procedure that operates on data
  * collecting, sorting, selecting, summarizing, analyzing, and reporting
  * abstraction level (general vs. detailed)



* Data Flow
  * conduit of data/information between
    * between external entity and process
    * between process and data store
    * between processes

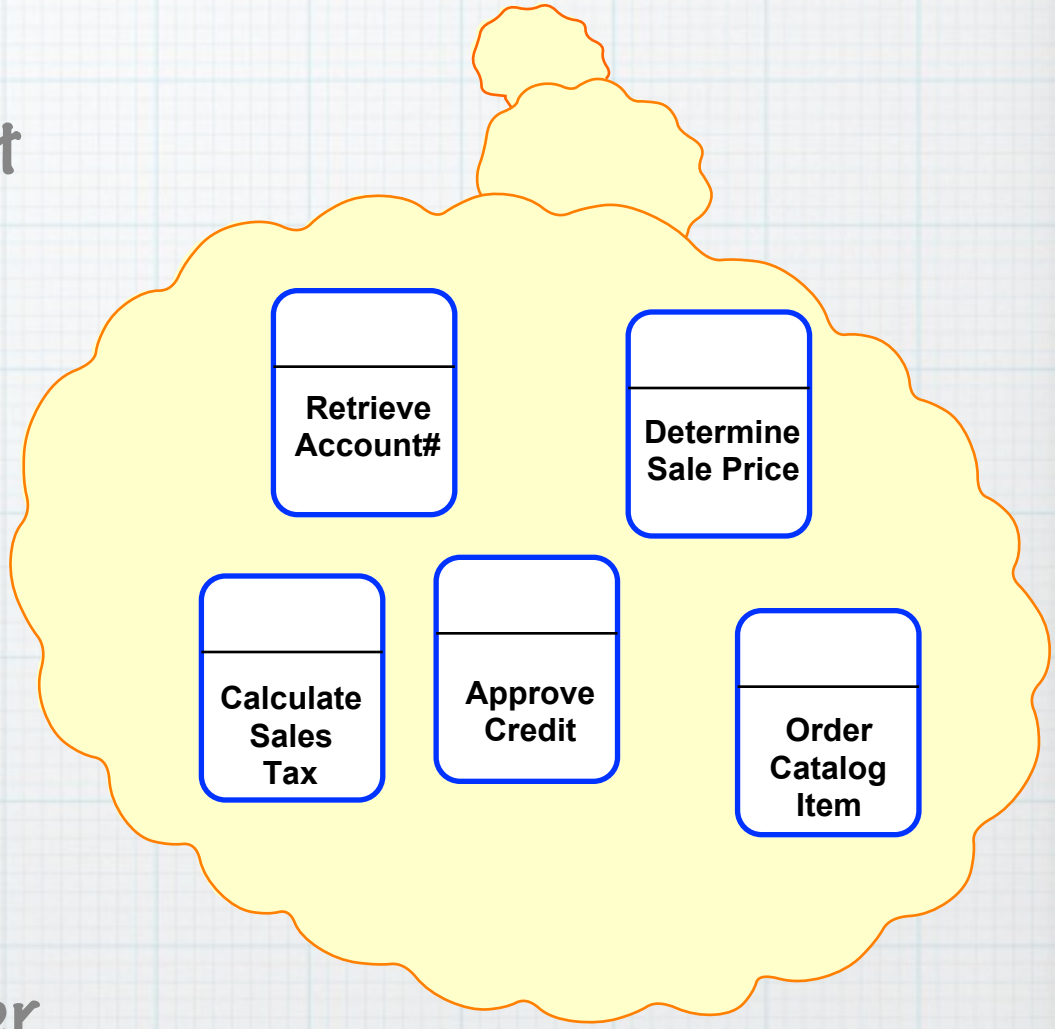
External Entity (sources & sinks)

- External origins and destinations of data
- Every source and sink is a "black box" from the perspective of the system
- Cannot access data stores directly without processes
- They are often organizations, organizational units, individuals, or other systems
- Name with nouns



Processes

* Describes the actions that are taken to transform, store, or distribute data
* Name with verbs
* Can be either computerized, manual, or both
* For example, Receive customer order, Process customer complaint, Distribute daily sales order report

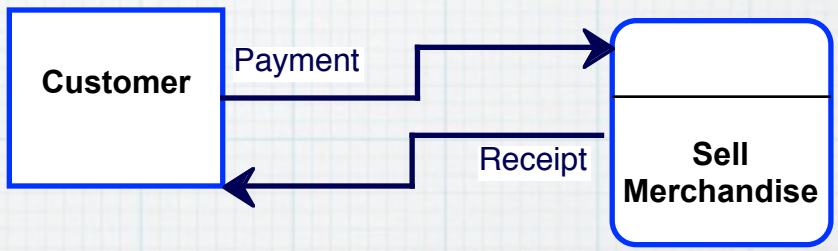


Identifying Processes

- One way to do this: Analysis of Business Events
  - External events – an outside stakeholder interfacing with the system (e.g., customer placing an order; arrival of supplier's shipment; sales rep calling customer)
  - Decision events – human decision is needed to evaluate input from multiple sources (e.g., hiring decision)
  - Time-based (temporal) events (daily, weekly, monthly, etc. actions)
  - State events (e.g., drop of inventory levels below a certain level triggers action)

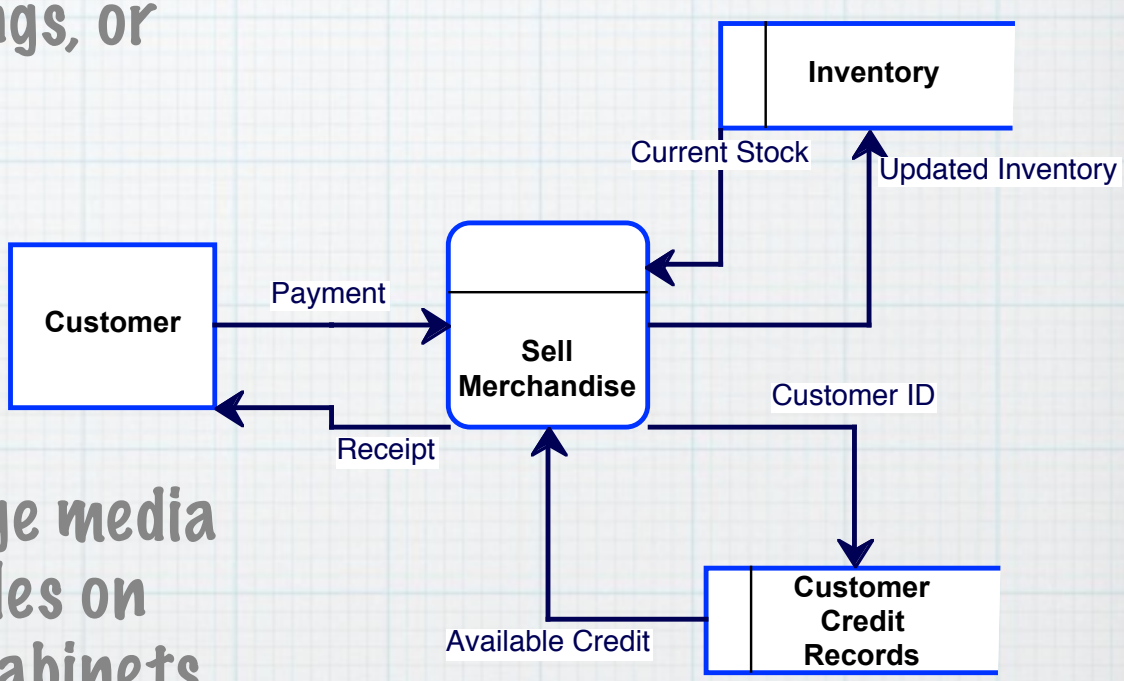
Data Flow

* Data that is in motion
* Data moving together
  * Data flow often consists of several components
* Name with nouns
* Data can flow using various media: paper forms, conversation, computer network, electronic media



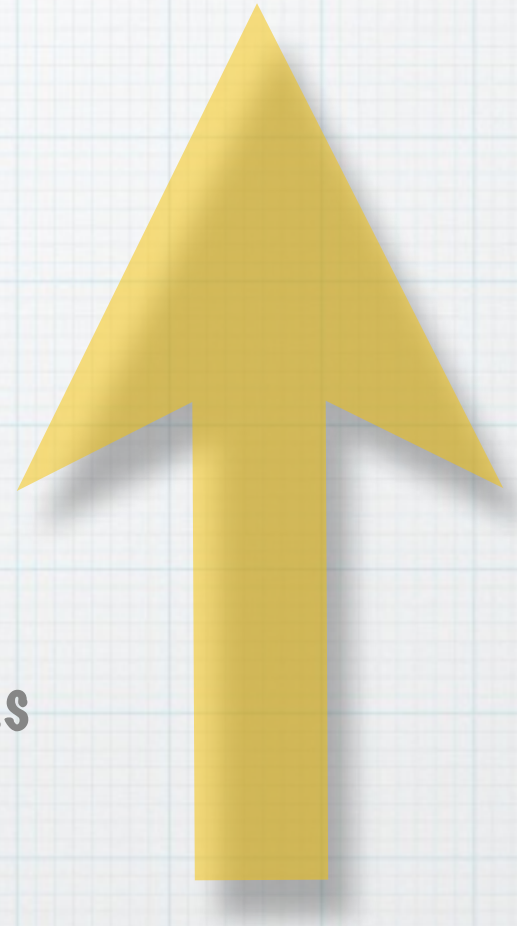
Data Store

- Data at rest
- Data about people, things, or events
  - E.g., customers, sales transactions, discount percentages, etc.
- Name with nouns
- Various physical storage media used: databases and files on computer disks, filing cabinets, desk drawers, notebook (method irrelevant for logical DFDs)



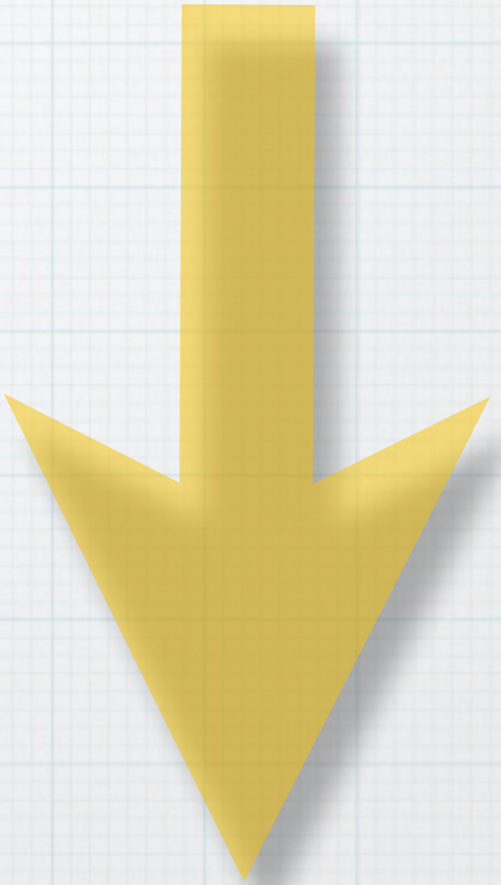
DFD's Bottom-Up

1. Develop system narrative
2. Identify system actions
3. List tasks in order
4. Keep only data transformations
5. Identify cohesive tasks
6. Collect all actions under these tasks
7. Draw an Input Process Output chart for each cohesive task



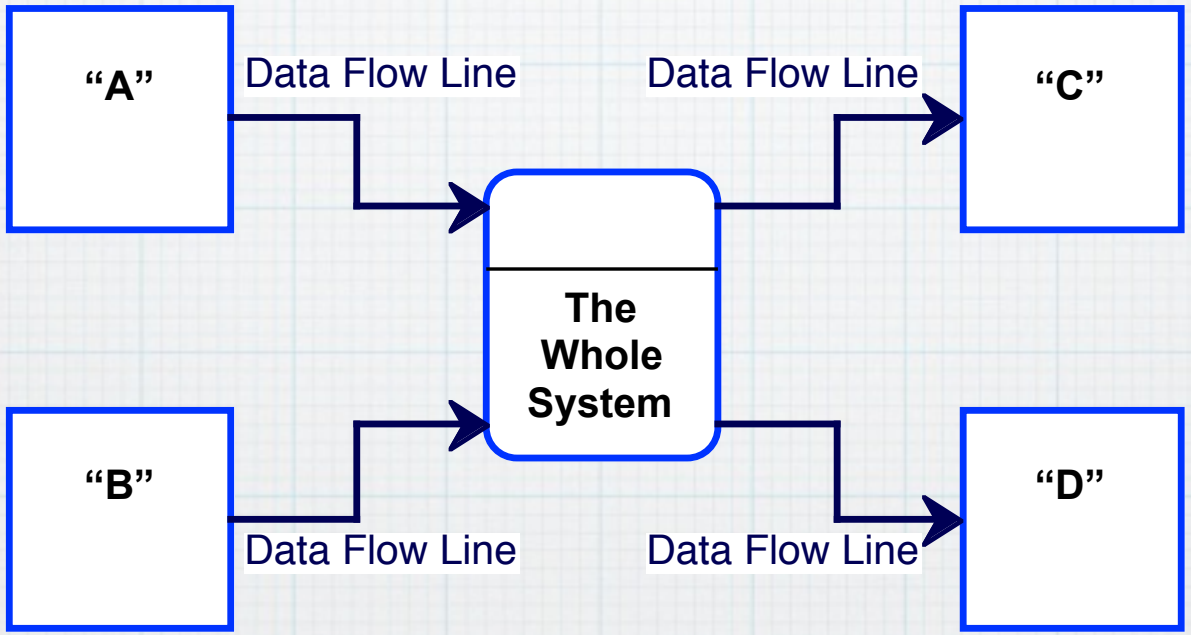
DFD's Top Down

1. "What's the primary task of the system?"
2. "What tasks does that break down into?"
3. Repeat the decomposition until the tasks are "trivial"



Context Diagram

* Shows the entire system as a single process

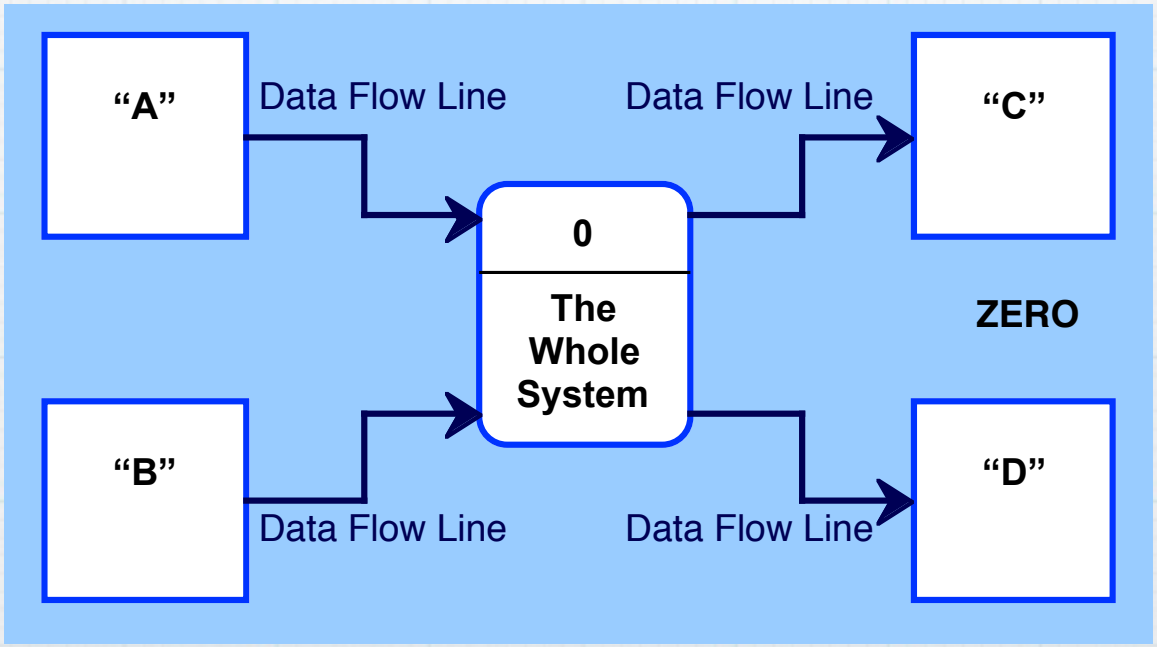


Context Diagram (Level zero)

* "The context level diagram is intended to identify the system boundary with regard to its relationship to any source or sink entities that may interact with it. As such, the context diagram contains only one process, labeled with the name of the system and assigned a zero as its identifier." (Marakas, 2001, p. 127)
* "An overview of an organizational system that shows the system boundaries, external entities that interact with the system, and the major information flows between the entities and the system." (Hoffer & al, 2002, p. 243)

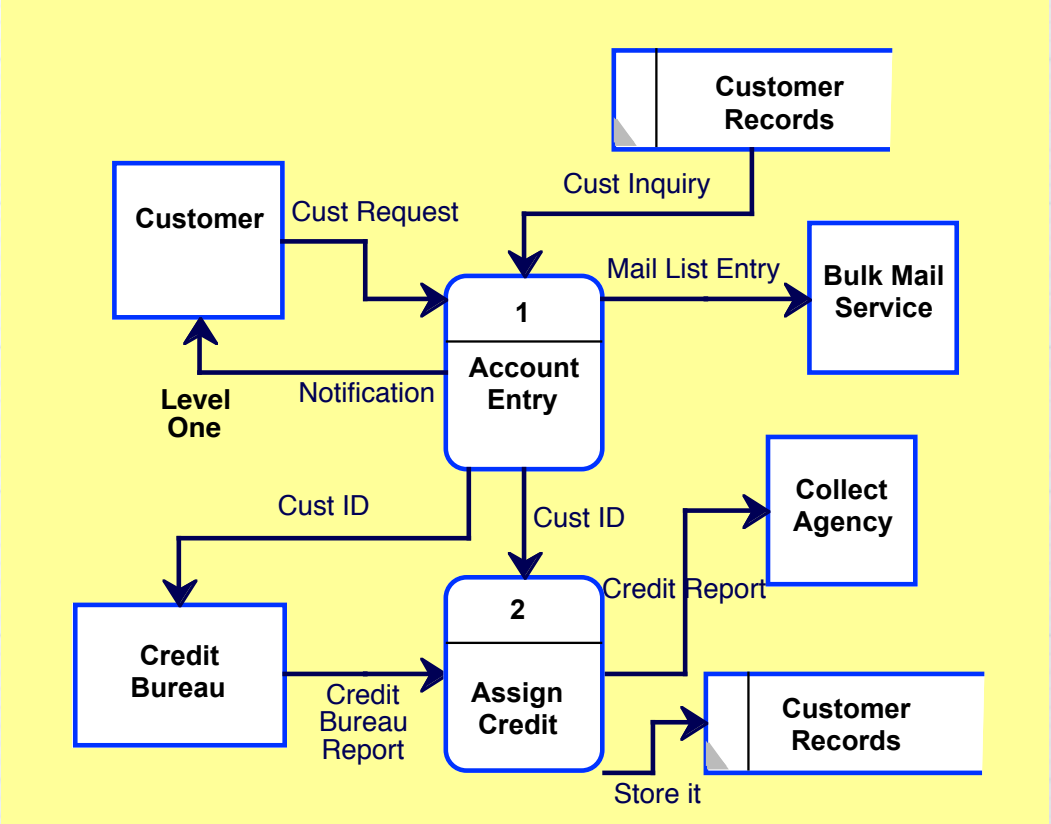
Context Diagram

* The context diagram depicts the most general view
  * Overview of system connections to the "outside world"

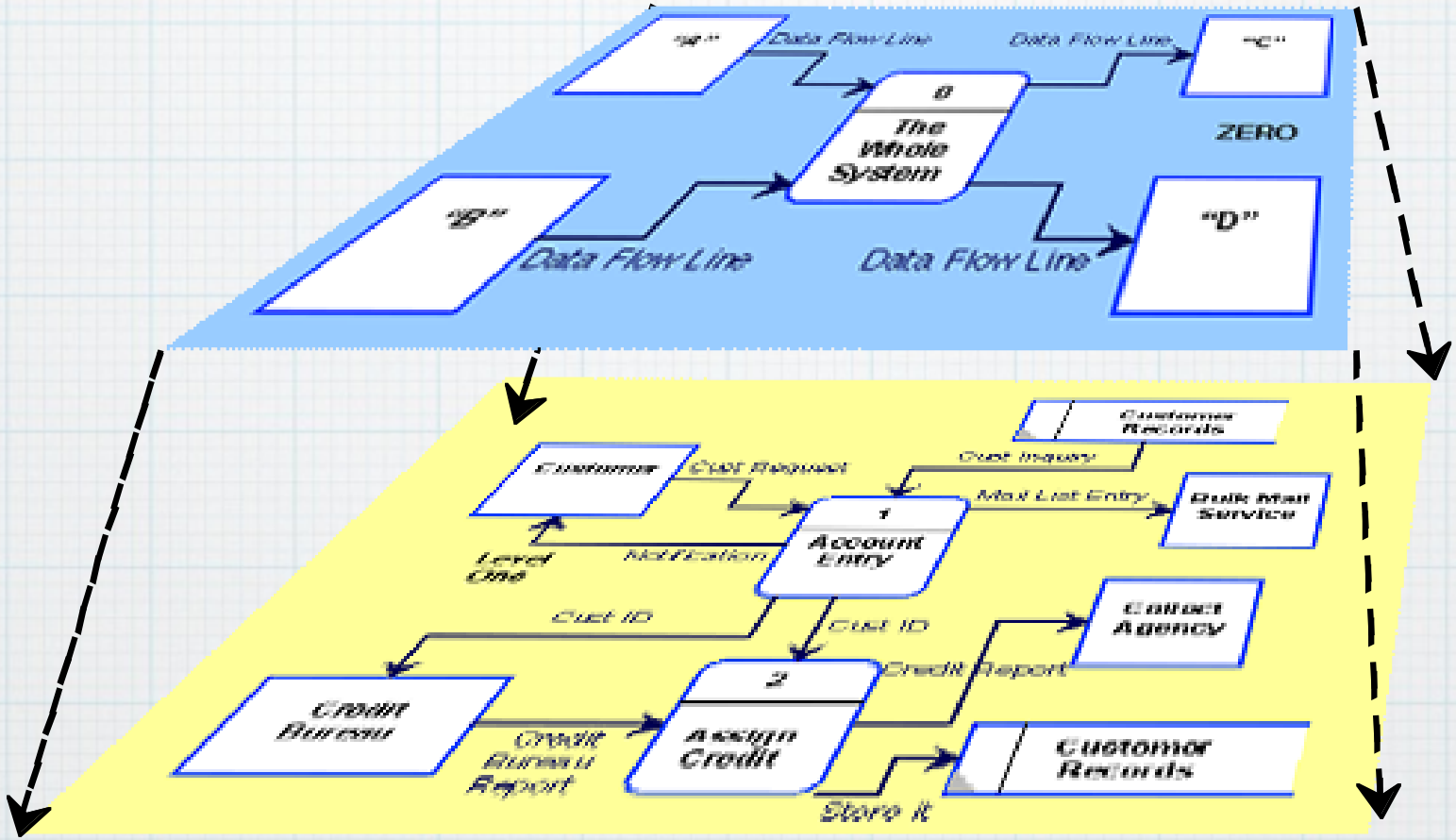


Decomposition "Drilling Down"

* Dividing a system/ process into subsystems/ processes that all perform a well-defined function
* Several levels of decomposition in a large system (subsystems/ processes can be further divided into smaller units)
* Continues until no sub-process can logically be broken any further



"Telescoping Levels of Abstraction"



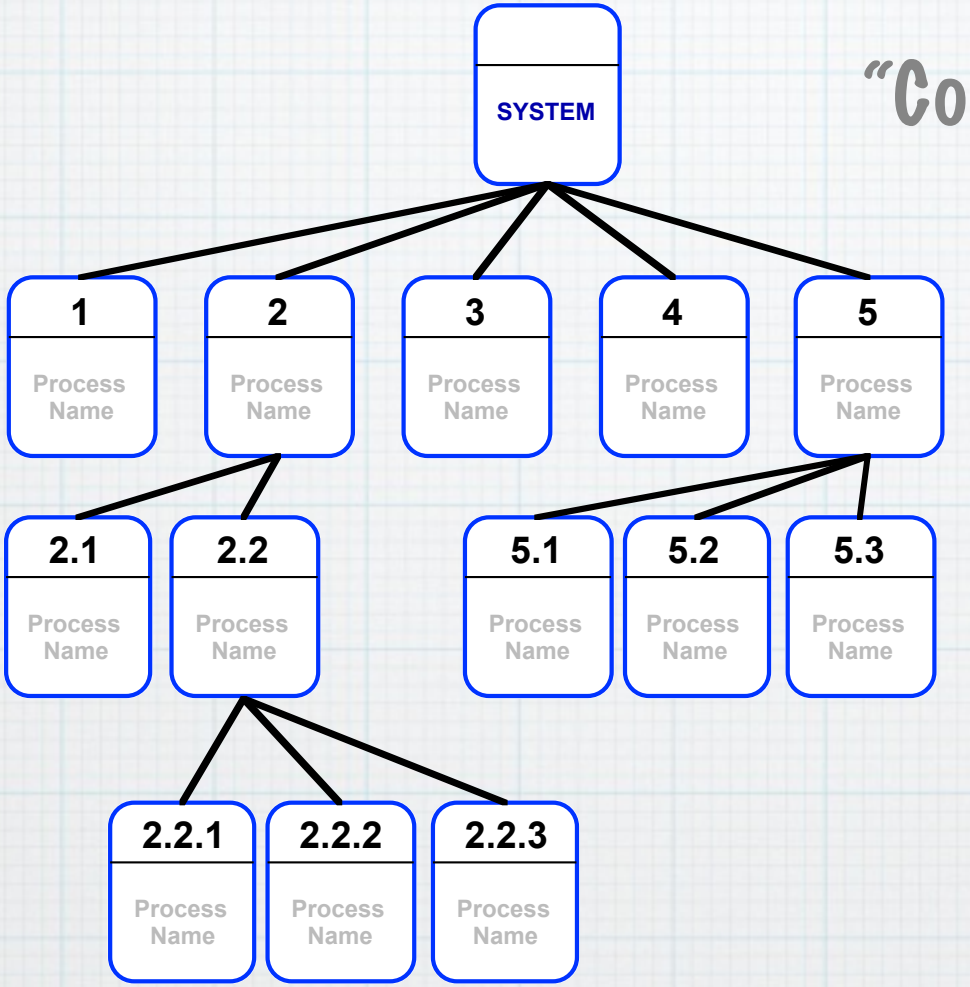
"Balancing" DFD's

* Inputs to and outputs from a process in a data flow diagram should remain the same when the process is decomposed
* A data flow can be split into component data flows that are input for different subprocesses, but the content of the data flow has to remain the same

Identifier in DFD's

- The context level diagram gets the number 0 (zero)
- Level-0 diagram processes are numbered from 1.0 to n.0 (where n = the total number of processes)
- In Level-1 diagrams, the processes are numbered with the number of the Level-0 diagram process that is begin decomposed and a running number for each of the processes (E.g., 2.1, 2.2, etc., if the Level-1 represents the process 2.0 from Level-0)
- The data stores are identified with a letter-number combination; most commonly used letters are S and D (E.g., S1, S2, etc.)

DFD Tree



"Context Diagram Level"

Level Zero

Level One

Level Two

DFD Syntax . . .

* A: No process can have only outputs
* B: No process can have only inputs
* C: Processes are named with verbs
* D: Data cannot move directly from one data store to another data store
* E: Data cannot move directly from an external source to a data store (a process is required)
* F: Data cannot move directly from a data store to an external sink (a process is required)

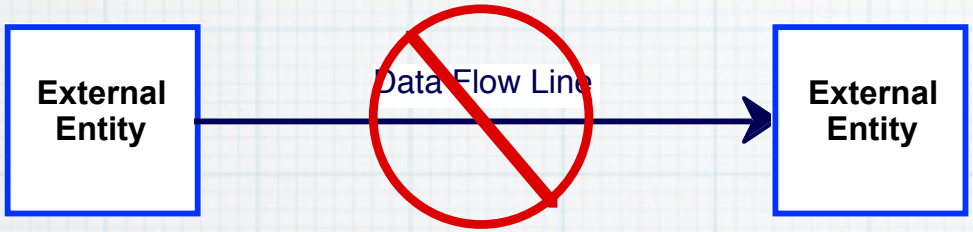
DFD Syntax . . .

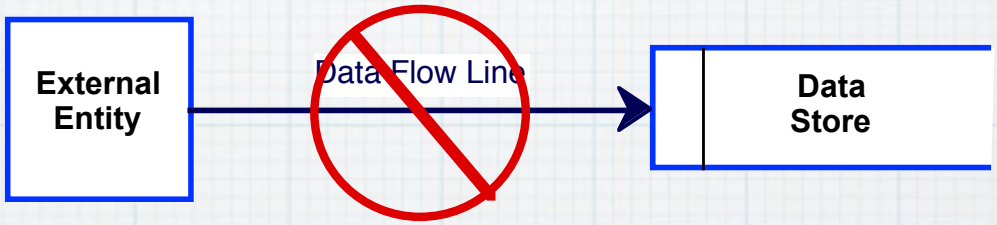
* G: A data store is named with a noun
* H: Data cannot move directly between two external entities (e.g., from a source to a sink)
* I: A source and a sink are named with a noun phrase
* J: A data flow is unidirectional
* K: A fork means that exactly the same data flow goes to two different destinations

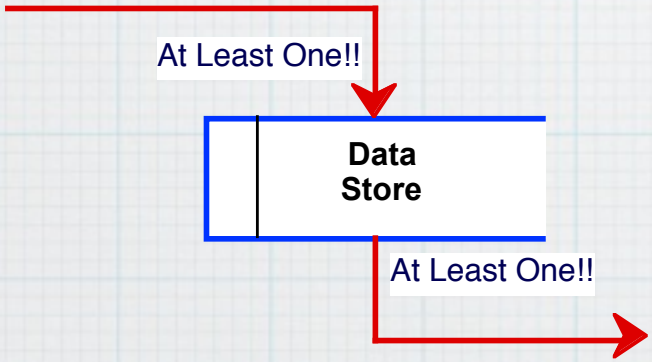
DFD Syntax . . .

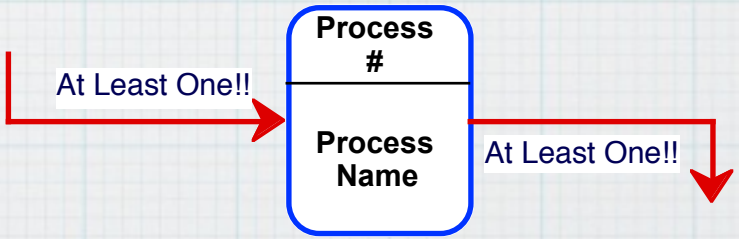
* L: A join means that exactly the same data comes from two different locations to the same destination (very rare)
* M: Loops are not permitted
* N: A data flow is named with a noun phrase. Several nouns can be used as part of the same label if several flows move together as a package

Shorthand DFD Syntax Rules









"Logical" vs. "Physical" DFD's

* Logical DFD
  * Ignores implementation specifics
    * computer configuration
    * data storage technology
    * communication or message passing methods
  * Focuses on the functions performed by the system
    * data collection
    * data to information transformation
    * information reporting
* Physical DFD
  * Reports/Prescribes implementation specifics
    * data entry devices
    * data storage capacities and technologies
    * network and data transfer modes and protocols
  * Describes the information processing "hardware"
    * processing modes (batch, online, interactive)
    * user interface details (forms, report media)
    * connectivity of information processing to user "world"

Producing DFD's

1. Start with "Top-Down" or "Bottom-Up"
2. Sketch some "system context" ideas
3. Identify "next level of detail" process steps below the "previous level"
   a. Define inputs from ones available at previous level
   b. Define any data stores needed to hold "working" data
   c. Define the processes feeding to the previous layer outputs
4. Repeat steps 2 thru 4 until process blocks are trivial to describe

CASE and drawing tools can take a lot of the pain out of "reworking" and polishing any DFD with more than a few processes and more than one level. That's why it's worth learning a CASE tool like Visible Analyst!!

Guidelines . . .

* Identify external entities (source and sinks; sets system boundary)
* Identify inputs and outputs for each entity
* Ignore timing considerations -- draw system as if it never started or will never stop
* Iterate and refine -- usually takes at least 3 drafts of each DFD for each level
* Decompose and balance processes until they are highly cohesive
* Try to keep diagrams to 7-12 symbols

Guidelines . . .

- Completeness
  - Have all components been included and fully described?
- Consistency
  - Between levels
- When to stop?
  - Once you have reached the level of details suitable for primitive DFDs

Some Additional DFD Comments

* DFDs tend to be relatively unstable
* DFDs alone are not enough to describe the requirements at a sufficient level
* Logic modeling tools required to describe the internal logic of the primitive level DFDs
* Some authors have suggested that use cases (a requirements analysis tool developed for object-oriented analysis) can be used together with DFDs
  * For example, utilization of use cases to describe the behavior within low-level processes

Logic Modeling

Les Waguespack, Ph.D.

slides adapted
from Heikki Topi

What is Logic Modeling?

- Part of requirements structuring
- In logic modeling, we
  - Represent the internal structure and (detailed) functionality of processes in the primitive level DFDs
  - Represent the various states a system component can take and the events that lead to a transition from one state to another

What is Logic Modeling? . . .

- A way to formally represent business logic
  - What are the rules that are used to make certain specific decisions?
  - For example: When do our customers get discounts? What rules do we follow regarding overtime?
- A communication tool between analysts and users
- Not linked to any specific technology during the analysis phase

Tools for Logic Modeling

* Structured English
* Decision tables
* Decision trees
* State-transition diagrams

Structured English

- Imperative verbs (doing something) linked together with three basic structures from structured programming
  - 1) Sequence of single statements
  - 2) Selection of one or more statements
    - IF ... THEN ... ELSE
    - CASE i
  - 3) Repetition/iteration
    - DO n TIMES
    - DO ... UNTIL
    - DO ... WHILE
    - FOR EACH ... DO

Example (structured english):

* Process: Generate Orders

* DO
  * READ next Inventory-record
  * IF Quantity-in-stock is less than
  * Minimum-order-quantity THEN
    * GENERATE Order
  * END IF
* UNTIL End-of-file

Example (structured english):

* Process: Calculate Customer Service Charge

* DO WHILE Customer-records remaining
  * READ next Customer-record
  * SELECT CASE
    * CASE 1 (account-type is money-market)
      * IF daily-balance < $1000 for any given day THEN
        * set service-charge to $10
      * ELSE
        * set service-charge to $0
      * END IF
    * CASE 2 (account-type is regular)
      * set service-charge to $3
  * END CASE
* END WHILE

Decision Tables

- A good way to model complex processing logic
- A presentation of all the possible choices and the conditions the choices depend on in a table format
- Three parts
  - Conditions
  - Actions
  - Rules

Example (decision tables):

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Actions	Pay Base Salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Testing Variables

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type						
	Hours worked						
Actions							

Possible Outcomes

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type						
	Hours worked						
Actions	Pay Base Salary						
	Calculate hourly wage						
	Calculate overtime						
	Produce Absence Report						

Tests

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Actions	Pay Base Salary						
	Calculate hourly wage						
	Calculate overtime						
	Produce Absence Report						

Resulting Action Choices

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Actions	Pay Base Salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Redundant Tests

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Actions	Pay Base Salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Minimized Decision Table

	CONDITIONS/ COURSES OF ACTION	Rules					
		1	2	3	4	5	6
Conditions	Employee type	S	H	H	H		
	Hours worked	<40	<40	40	>40		
Actions	Pay Base Salary	X					
	Calculate hourly wage		X	X	X		
	Calculate overtime				X		
	Produce Absence Report		X				

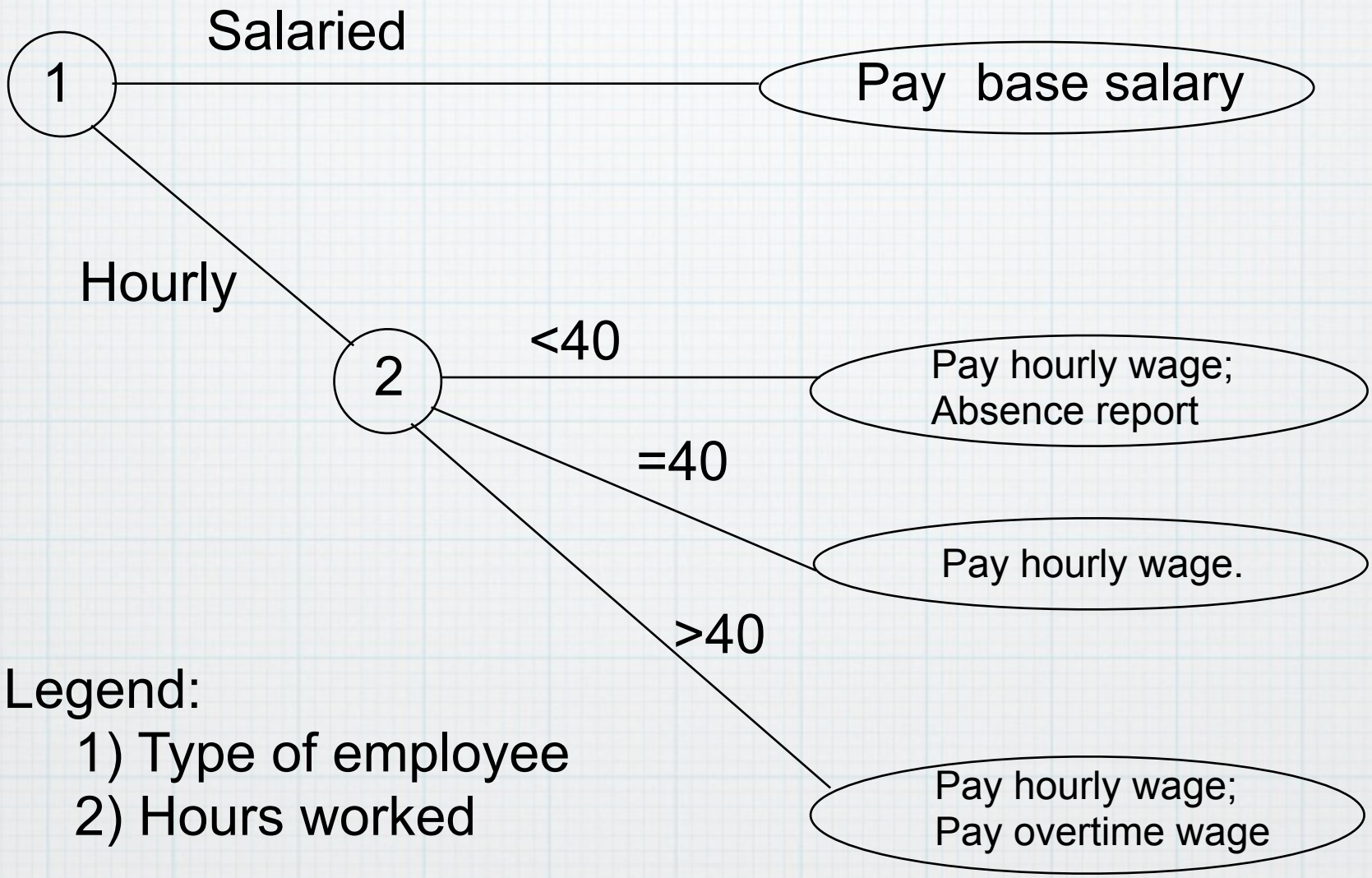
Steps for Building Decision Tables

1. Identify all conditions and values for conditions
2. Identify and name all possible actions
3. List all possible rules
4. Define the actions for each rule
5. Simplify the decision table

Decision Trees

* Another way to graphically represent a decision situation; not inherently better or worse than decision tables (usefulness depends on situation)
* In this context, consists of decision points (nodes), arcs connecting decision points, and actions (ovals)
* Good, intuitive communication tool if the situation is not very complex

Example (decision tree):



Legend:

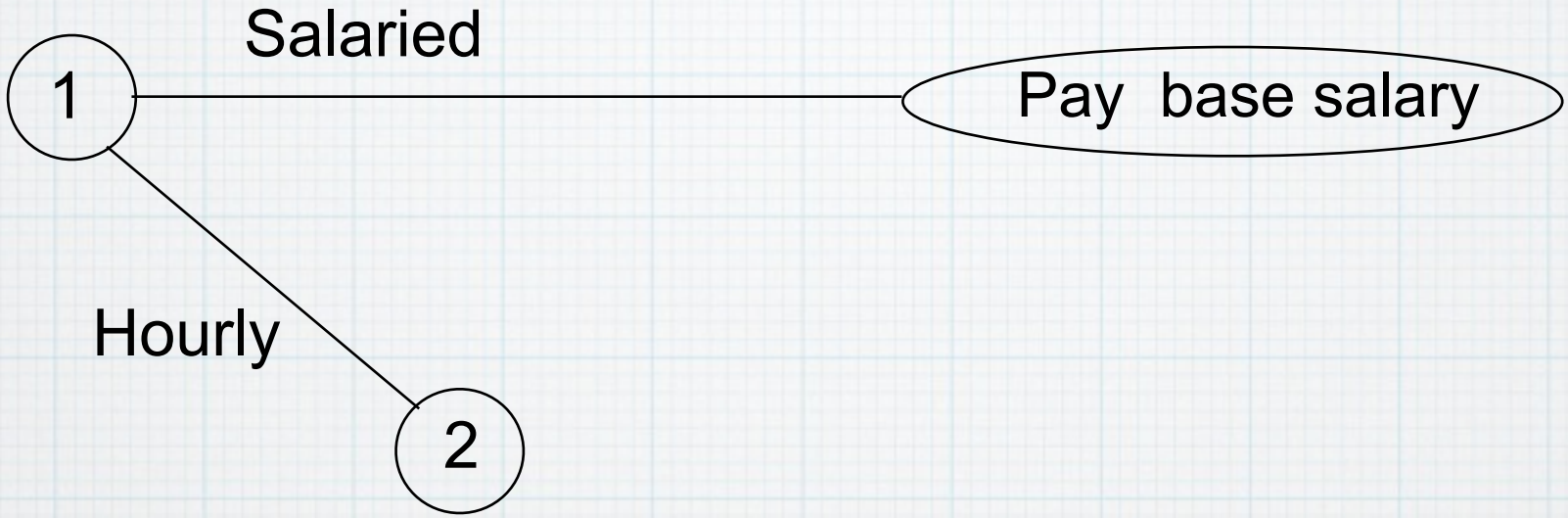
1) Type of employee
2) Hours worked

Example (decision tree):

Legend:

1) Type of employee

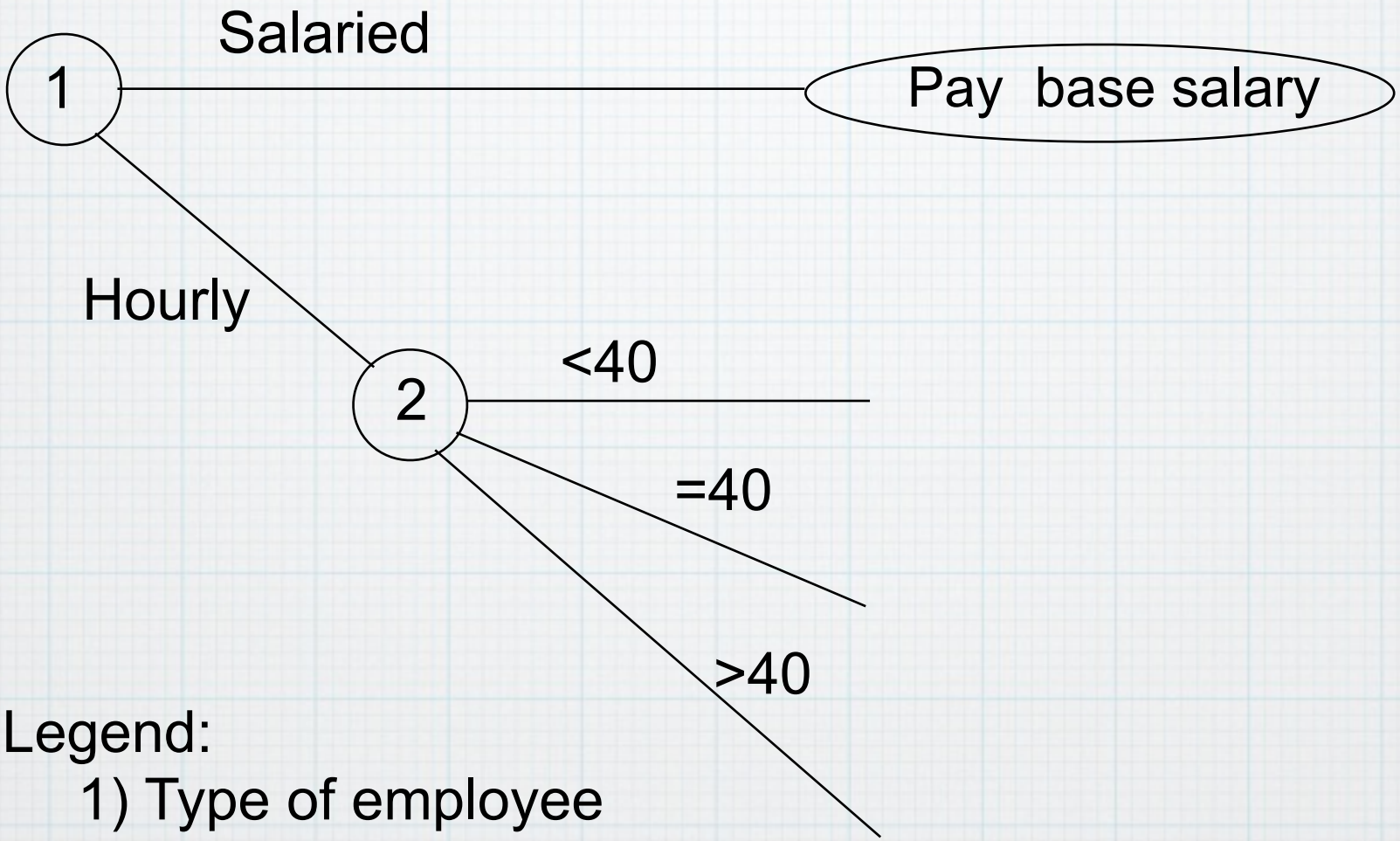
Example (decision tree):



Legend:

1) Type of employee
2) Hours worked

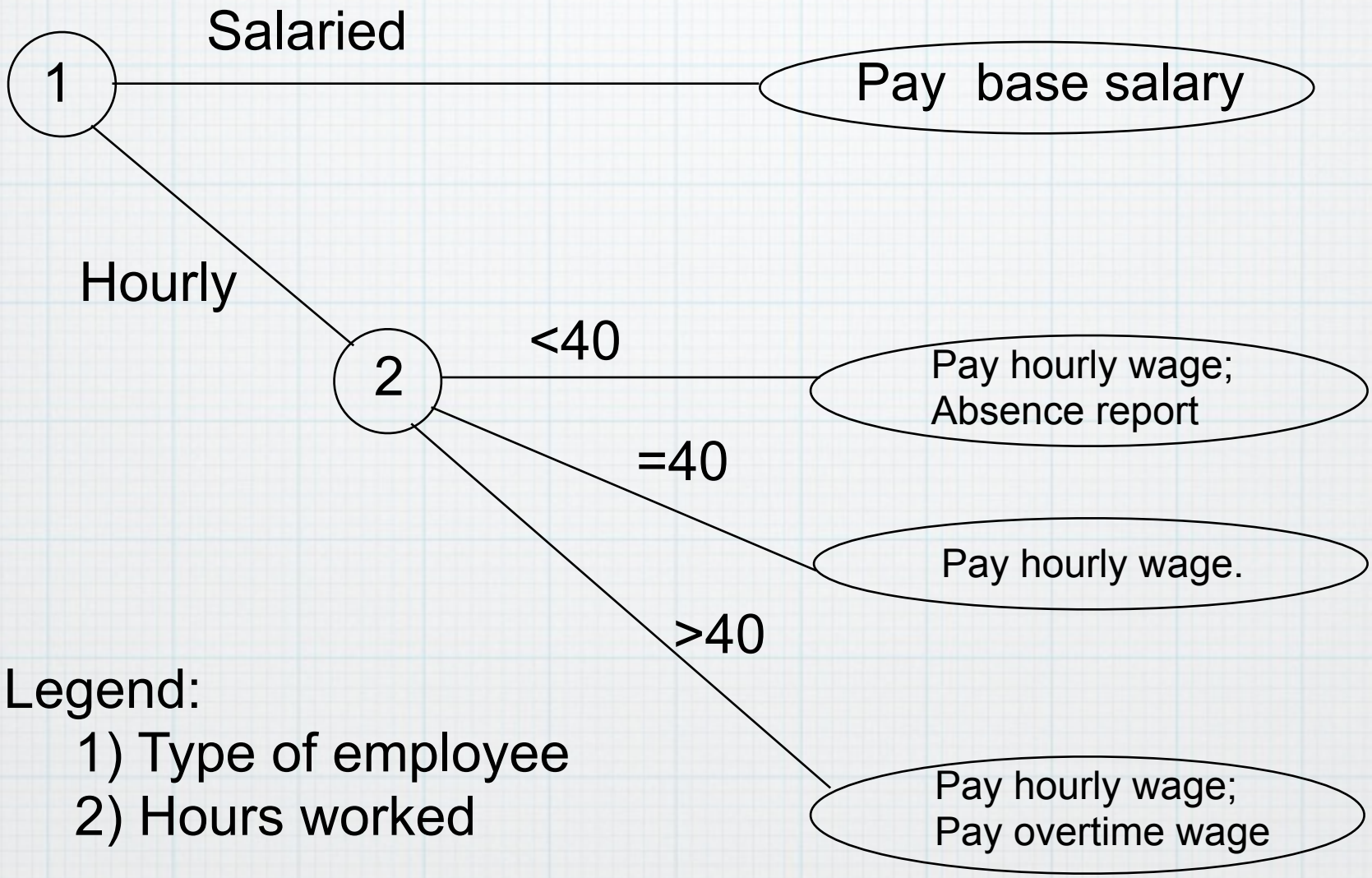
Example (decision tree):



Legend:

1) Type of employee
2) Hours worked

Example (decision tree):

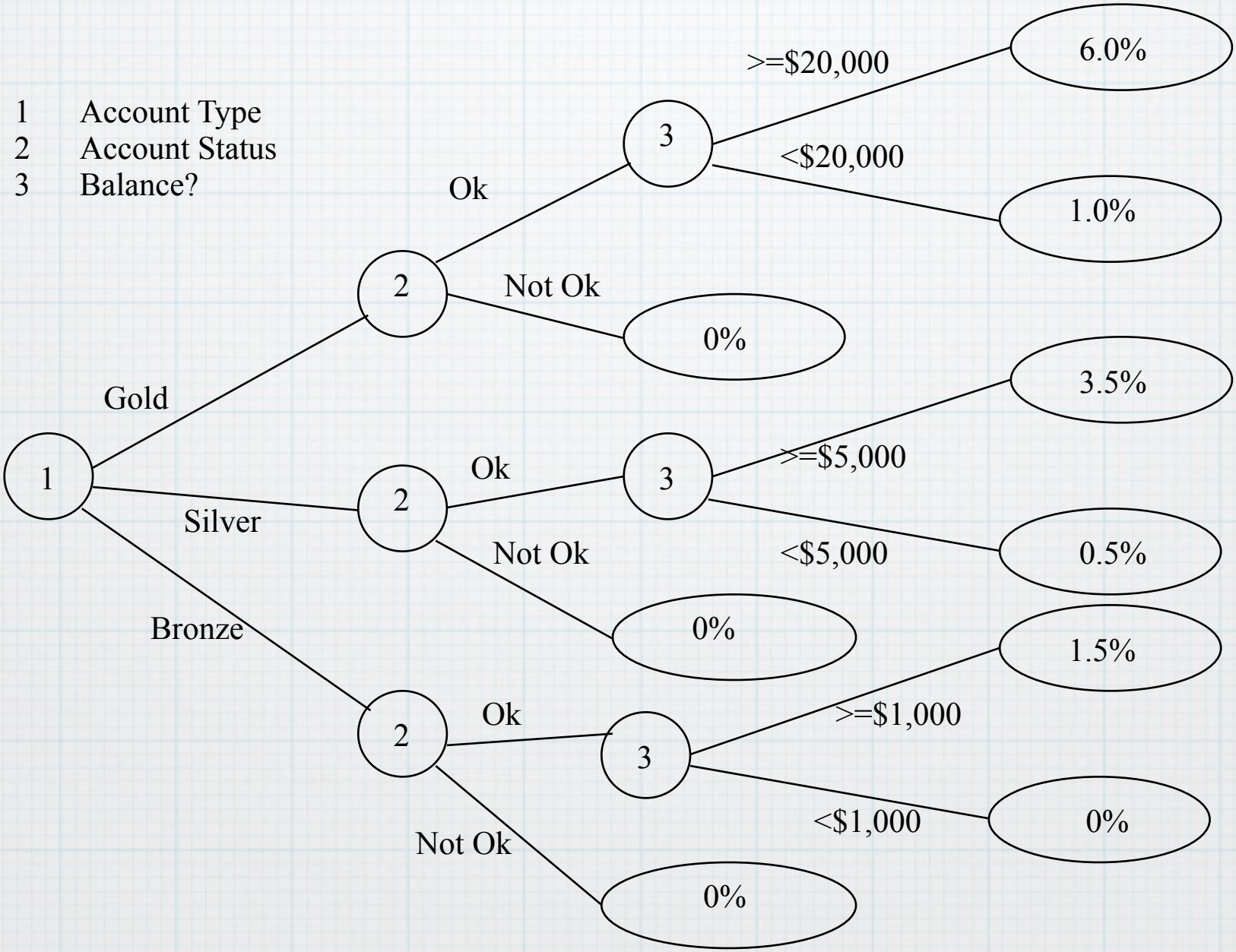


Legend:

1) Type of employee
2) Hours worked

Another Example (decision tree):

1 Account Type
2 Account Status
3 Balance?



Cost/Benefit

Criteria	Decision Tables	Decision Trees
Portraying complex logic	Better	
Portraying simple problems		Better
Supporting Decision Making		Better
Compactness	Better	
Flexibility	Better	

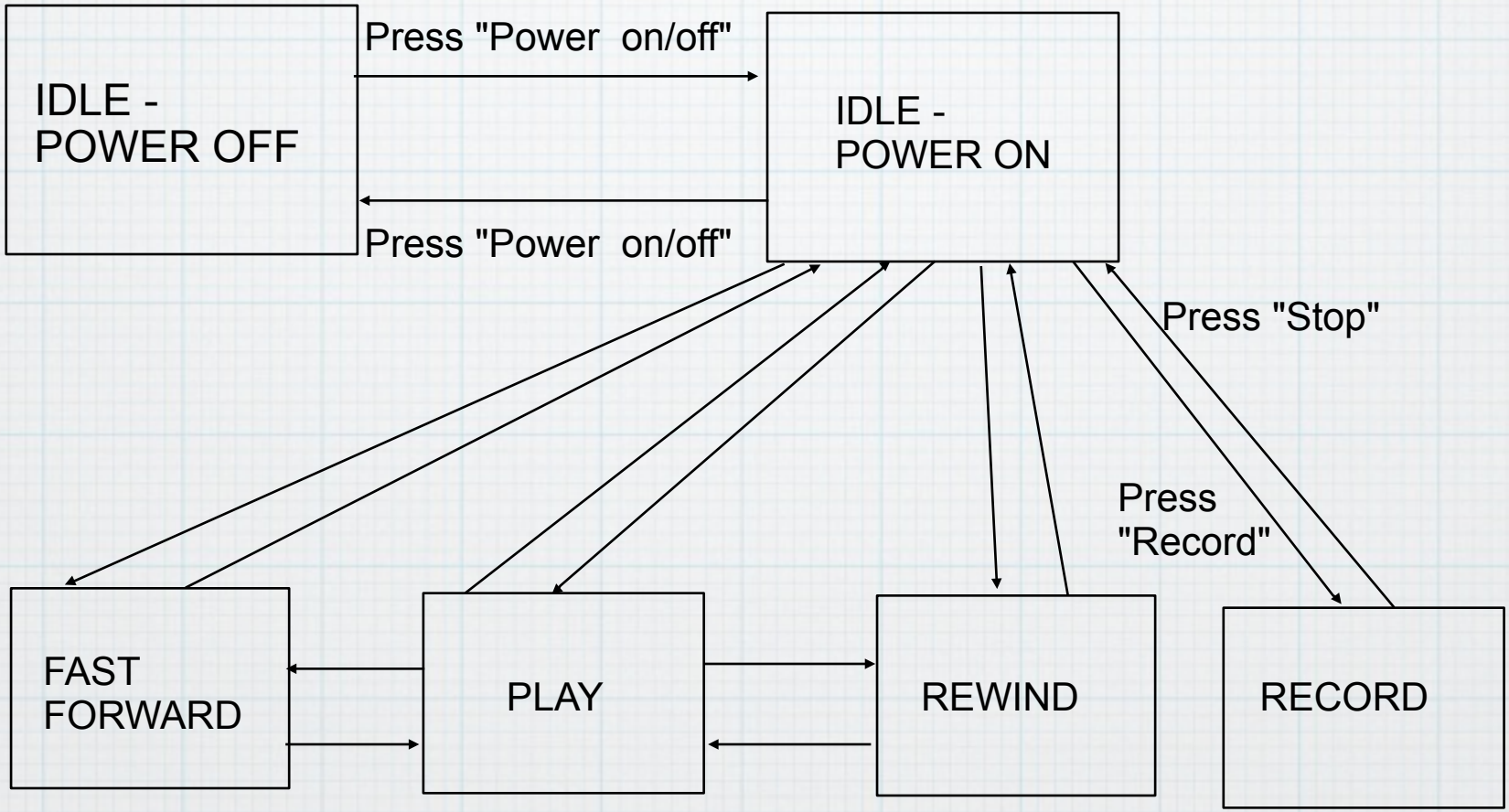
State-Transition Diagrams

* Represent
  * All relevant states of a component of a system
  * The rules defining the transitions between the states (i.e., the events that cause the component to change from one state to another)
* Show the dynamic behavior of a system or object (timing and the order of states are both important)

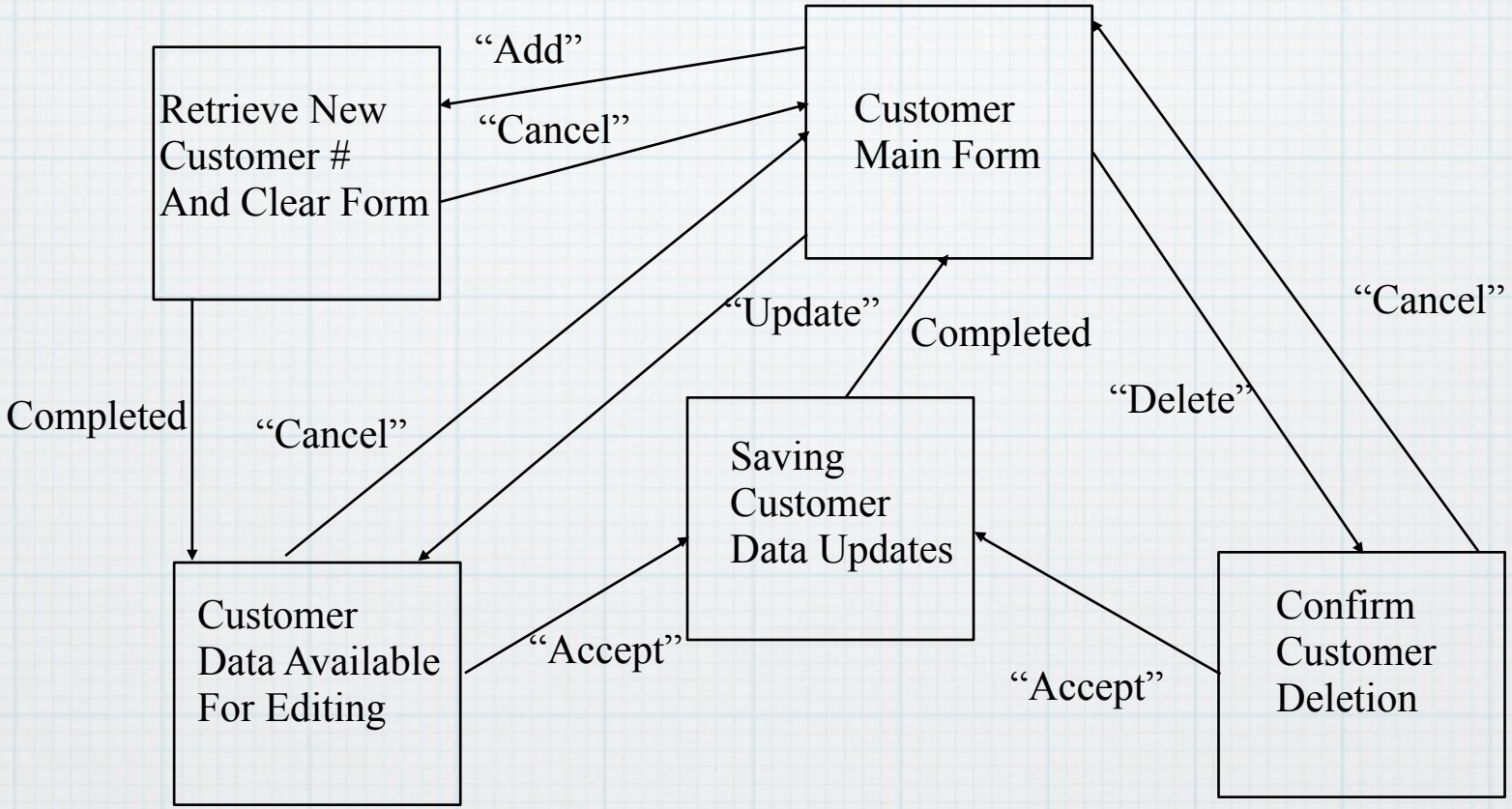
Terminology

- State (represented with a rectangle)
  - Describes the cumulative results of object's behavior
  - Consists of all the values of object's attributes (properties)
- Transition (represented with an arrow)
  - A change in state brought on by some event
- Events (represented with labels)
  - Triggers (discrete events); Passage of time

A simplified state-transition diagram for recording device



A simplified state-transition diagram for a customer management application



Use Cases

* An important new addition to the requirements structuring toolkit
* Widely used in the context of object-oriented analysis and design
* Use cases are a part of UML
* Brief review here

Use Case Elements by Cockburn (2001)

* Name (the goal as a short active verb phrase)
* Context of use
* Scope
* Level (summary, user-goal, subfunction)
* Primary actor
* Stakeholders and interests
* Precondition
* Minimal guarantees
* Success guarantees (end state if everything goes well)

Source: Cockburn, A. (2001). Writing effective use cases. Boston, Addison-Wesley.

Use Case Elements by Cockburn (2001)

* Trigger (what starts the use case)
* Main Success Scenario
  * Steps of the scenario from trigger to goal delivery
* Extensions
  * In practice, alternative scenarios
* Technology and Data Variations List
  * Describe different ways of doing things
* Related Information

Links Between Requirements Structuring Approaches

- Maintain consistency; for example,
  - **DFDs vs. E-R diagrams**: the contents of every data store in DFDs have to be represented in the E-R diagrams
    - Ideally, each data store included in a primitive level DFD represents one entity
  - **DFDs vs. Logic models**: logic models represent the rules for processing that takes place within DFD processes
  - **Logic models vs. E-R diagrams**: all data required for decision making must exist in E-R diagrams

One Possible Integrated Process

* Create (the initial version of) the conceptual data model first and use it as a basis for all later modeling efforts
  * Data-oriented approach to SA&D
  * Capture stable rules regarding entities and relationships
* IF modeling the current system is essential
  * Create current logical DFDs to capture the current process model
  * Capture the current business logic by modeling the rules governing the DFD processes with suitable logic modeling techniques

One Possible Integrated Process

* Word of warning:
  * If you decide to model the current system formally, don't spend too much time on this task; make sure that you capture the essential business requirements embedded in the current system and then go on to model the new system!
* Make sure that you understand the fundamental business rules and express them with logic modeling tools, if this has not already been done
* Develop new logical DFDs to capture the requirements of the new system

One Possible Integrated Process

* Compare the new business requirements and EER diagrams and make sure that everything relevant is represented in the EER diagrams
* Compare the new DFDs and EER diagrams and make sure that these are compatible
* Make sure that the presentation of the business logic is compatible with the new process structure
* Repeat the previous four steps, if necessary, but also remember that you will never achieve a perfect solution

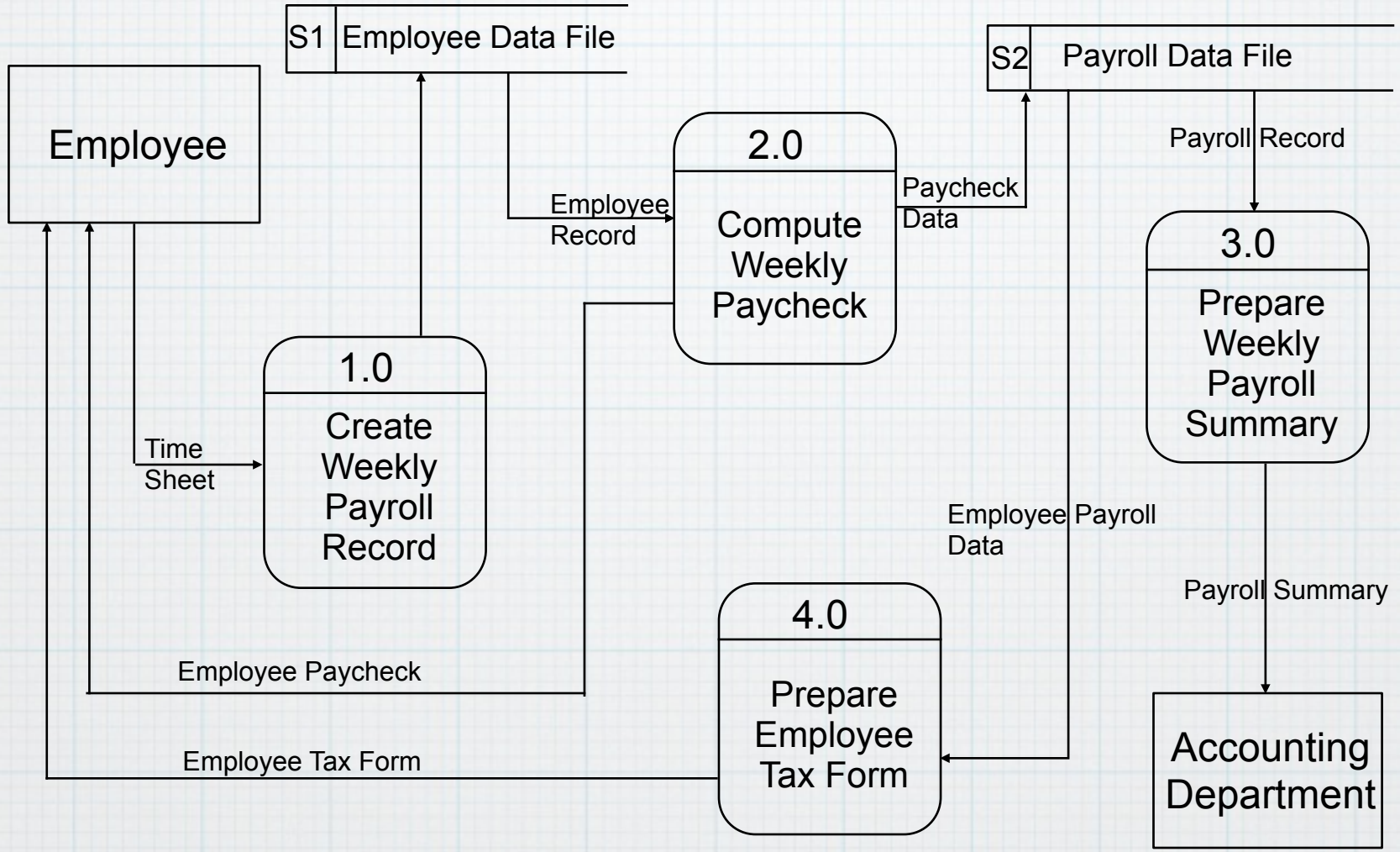
Monitor and Adjust

- Accept the iterative nature of the process
  - You have to switch between requirements determination (information gathering) and the requirements structuring techniques
  - Learning something new from one perspective triggers changes in other views
  - Remember to keep users involved in the process; requirements determination and structuring are closely intertwined and in continuous interaction

Monitor and Adjust

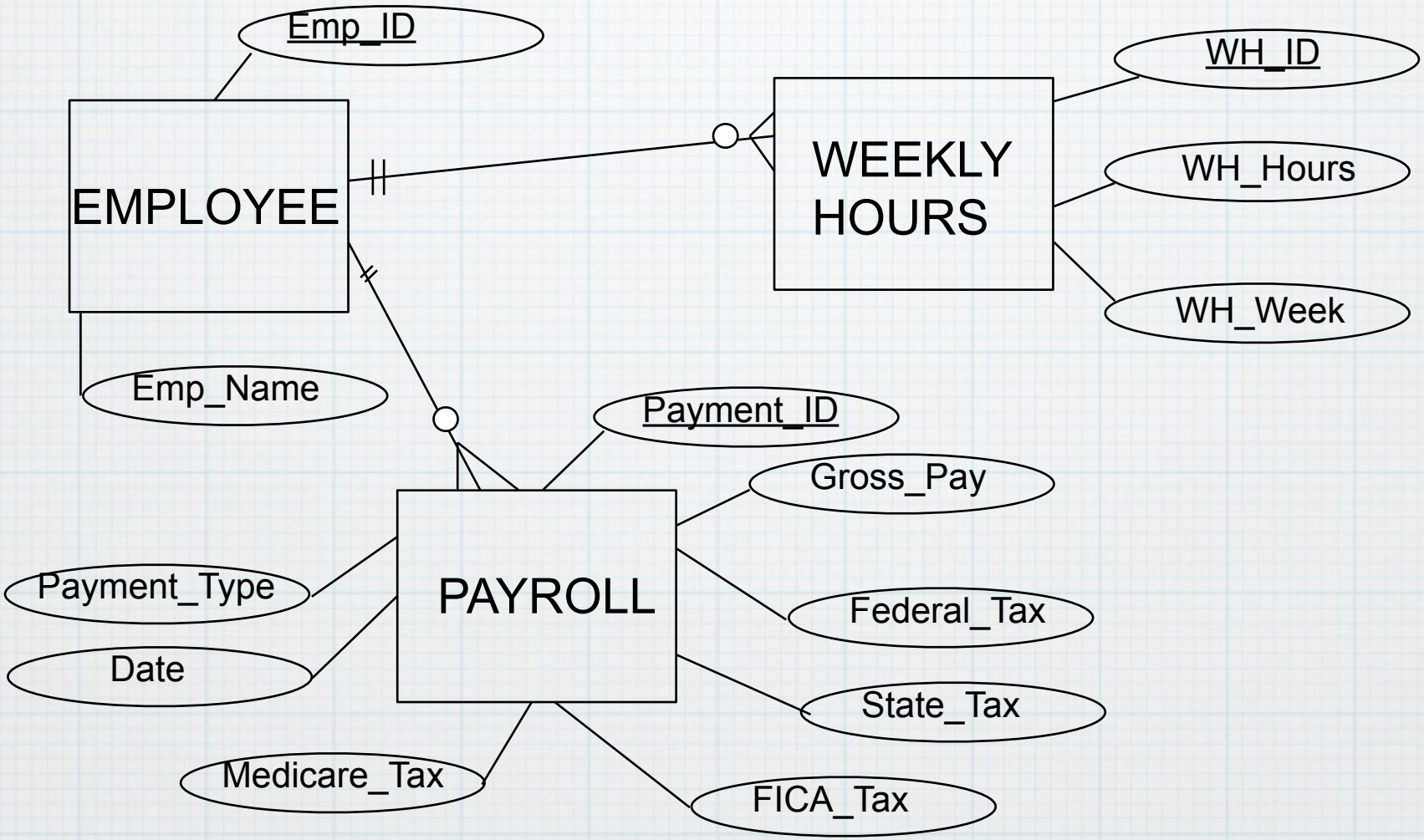
- Use narrative descriptions to supplement/ support diagrams
  - Administrative information (who created, when, where did the info come from)
  - Definitions
  - Descriptions of business requirements that cannot be captured with diagrams
  - Use cases for processes
- It is essential that the documentation is created and stored so that it is easy to maintain
- Try to find tools to help in the process

Level-0 Diagram for the Payroll System



Source: Marakas, G.M., *Systems Analysis and Design*, 2001.

Entity-Relationship Diagram



Project Diary

* A central repository of descriptions of project elements
* Often maintained as a database by a CASE tools
* Includes descriptions of
  * Data elements (entities, relationships, and attributes)
  * Processes
    * Including the internal logic of the processes
    * Use cases
  * Data flows
  * State-transition diagrams
  * Design elements, such as screens, reports, and database structures
  * Narrative project documents
  * Project management documentation

Wrap Up

* Conceptual Modeling with ER
* Enhanced Entity-Relationship Modeling
* Logical Data Modeling
* Data Flow Modeling
* Logic Modeling