Database Systems 2

Lecture 4

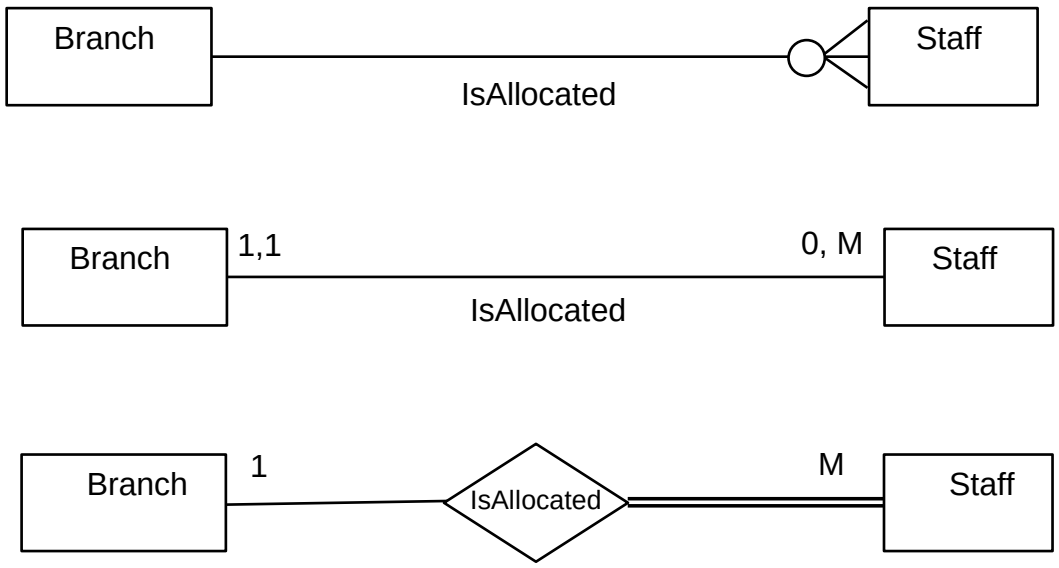
Implementing Optionality

Optionality

Also known as Participation Constraints.

An entities participation in a relationship can either be **total** or **partial**.

Also known as **mandatory** or **optional**.



Mapping 1:1 relationships

Before tackling a 1:1 relationship, we need to know its optionality.

There are three possibilities. The relationship can be:

1. mandatory at both ends
2. mandatory at one end and optional at the other
3. optional at both ends

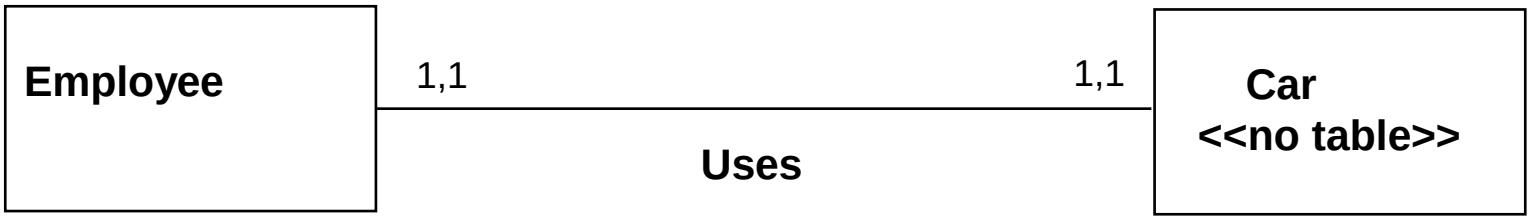
Mandatory at both ends

If the relationship is mandatory at both ends it is often possible to subsume one entity type into the other.

- The choice of which entity type subsumes the other depends on which is the most important entity type (more attributes, better key, semantic nature of them).
- The key of the subsumed entity type becomes a normal attribute.
- If there are any attributes in common, the duplicates are removed.
- The primary key of the new combined entity is usually the same as that of the original more important entity type.

1:1 relationships

Mandatory – Mandatory



Each Employee must use one and only one Car.
Each Car must be used by one and only one Employee.

The two entities can be collapsed into one.
The skeleton table would look like this:

Employee(employeeNo, ..., carNo, ...)

The dots indicate other attributes of an employee and of a car.

employeeNo	employeeName	carNo	make	model
E1	Shaw	W123DRY	Ford	Mondeo
E2	Keats	Y469JWS	Ford	Focus
E3	Byron	Y743PDJ	Fiat	Punto

The Uses relationship is represented implicitly by the presence of employeeNo and carNo in same table, rather than by two linked tables.

The Car attributes are said to be posted into the Employee table.

When not to combine

There are a few reason why you might not combine a 1:1 mandatory relationship.

- the two entity types represent different entities in the 'real world'.
- the entities participate in very different relationships with other entities.
- efficiency considerations when fast responses are required or different patterns of updating occur to the two different entity types.

If not combined…

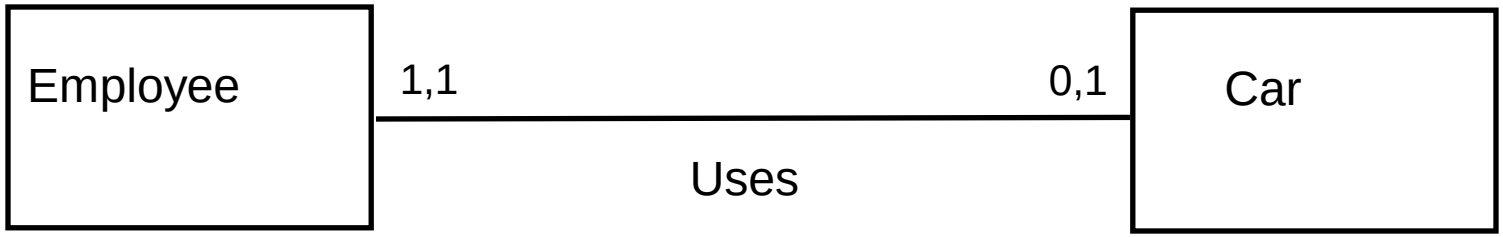
If the two entity types are kept separate then the association between them must be represented by a foreign key.

- The primary key of one entity type comes the foreign key in the other.

- It does not matter which way around it is done but you should not have a foreign key in both entities.

1:1 relationships

Mandatory – Optional



Each Employee may use one Car, but does not have to have a car.

Each Car must be used by one and only one Employee.

This means that attributes of cars are not attributes of employees in general, but just of some employees.

If we stored all of the data in one table, it might give rise to Null fields

employeeNo	employeeName	carNo	make	model
E1	Shaw	W123DRY	Ford	Mondeo
E2	Keats	NULL	NULL	NULL
E3	Byron	Y743PDJ	Fiat	Punto

A better solution would be to keep the tables separate and use a foreign key.

Employee (employeeNo, ...)

Car (carNo, ... , employeeNo*)

Note that the relationship Uses is represented by posting the key of Employee, (employeeNo), into the table for Car. This forms a logical link, and what is more, it will never be null because every car is used by an employee.

employeeNo	employeeName
E1	Shaw
E2	Keats
E3	Byron
E5	Shelley
E6	Scott

carNo	make	model	employeeNo*
W123DRY	Ford	Mondeo	E1
Y469JWS	Ford	Focus	E6
Y743PDJ	Fiat	Punto	E3

```
CREATE TABLE car
{
carNo                   VARCHAR(8),
make                    VARCHAR(20),
model                   VARCHAR(20),
FK_employee_No          VARCHAR(2) NOT NULL,
PRIMARY KEY (carNo),
FOREIGN KEY FK_employee_No REFERENCES employee(employeeNo)
}
```

What is the NOT NULL constraint doing?

1:1 relationships

Optional - Optional

Each Employee may use one and only one Car, but does not have to have a car.

Each Car may be used by one and only one Employee, but may not be used by anyone.

This means that we cannot represent the Uses relationship by posting employeeNo into the Car table, nor by posting carNo into the Employee table, as this might give rise to null fields in both cases.

The solution is to represent **Uses** by a separate table

- **Entity Tables**

```
Employee( employeeNo,  .....)
Car(carNo, ..... )
```

Relationship Tables

```
Uses( employeeNo, carNo, ........)
```

Both employeeNo and carNo are candidate identifiers for the Uses table.

Question

Why do we only need to use one of the candidate identifiers in this case?

What might determine your choice of identifier?

employeeNo	employee Name
E1	Shaw
E2	Keats
E3	Byron
E5	Shelley
E6	Scott

carNo	make	model
W123DRY	Ford	Mondeo
Y469JWS	Ford	Focus
Y743PDJ	Fiat	Punto
T222POI	Opel	Kadett
Q333TGB	Lotus	Esprit

employeeNo*	carNo*
E1	W123DRY
E6	Y469JWS
E3	Y743PDJ

Mapping 1:M relationships

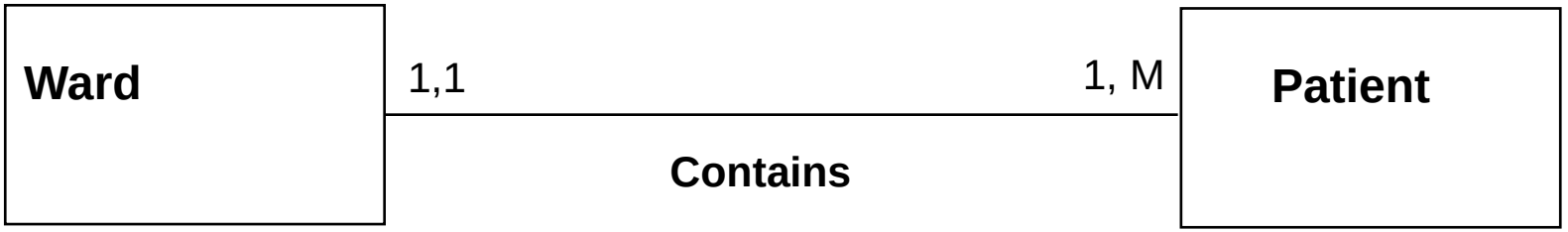
Before tackling a 1:M relationship, we need to know its optionality.

There are four possibilities. The relationship can be:

1. mandatory at both ends
2. mandatory at the 1 end and optional at the M end
3. optional at the 1 end and mandatory at the M end
4. optional at both ends

One:Many relationships

Mandatory (1 : M) Mandatory



Each Ward must contain at least 1, and possibly more Patients.

Each Patient must be assigned to one and only one Ward.

Foreign Key goes in the 'Many' entity

The table types would look like this:

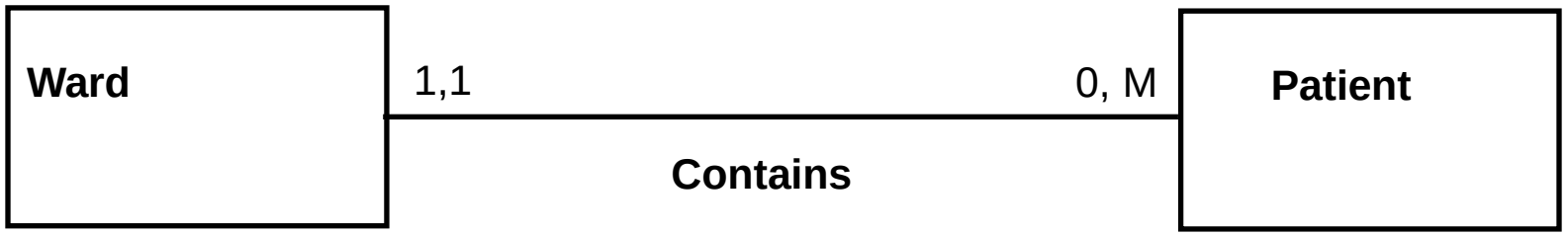
```
Ward( wardName,  ..... )
Patient( patientNo, .... , wardName*)
```

wardName	wardType
Harvey	orthopaedic
Pasteur	geriatic
Lister	cardiac
Jenner	orthopaedic

patientNo	patientName	wardName*
1274	Castle	Harvey
1288	Thomas	Lister
1289	Jones	Lister
1291	Watkins	Jenner
1292	Radford	Lister
1296	Berry	Pasteur

One:Many relationships

Optional (1 : M) Mandatory



Each Ward can contain many Patients (but might not contain any).
Its participation in the 'contains' relationship is optional.

Each Patient must be assigned to one and only one Ward.
Its participation in the 'contains' relationship is mandatory.

As before, have a foreign key in the entity at the 'Many' end of the relationship.

In other words the tables will be arranged exactly as they were for the Mandatory (1:M) Mandatory example above.

wardName	wardType
Harvey	orthopaedic
Pasteur	geriatic
Lister	cardiac

patientNo	patientName	wardName*
1274	Castle	Harvey
1288	Thomas	Lister
1289	Jones	Lister
1292	Radford	Lister

Which ward does not contain any patients?

One:Many relationships

Mandatory (1 : M) Optional

Each Ward must contain at least one, and possibly many, Patients.
Its participation in the 'contains' relationship is mandatory.

Each Patient can be assigned to no more than one Ward, but may not be assigned to any (Outpatients).
Its participation in the 'contains' relationship is optional.

In this situation it is not possible to post wardName into the Patient table without introducing a null wardName occurrence for every outpatient.

wardName	wardType
Harvey	orthopaedic
Pasteur	geriatic
Lister	cardiac
Jenner	orthopaedic

patientNo	patientName	wardName*
1274	Castle	Harvey
1288	Thomas	NULL
1289	Jones	Lister
1291	Watkins	Jenner
1292	Radford	NULL
1296	Berry	Pasteur

The solution is to represent the relationship **Contains** by a separate table. For each inpatient only, there will be an entry in the Contains table.

The table types would look like this:

Entity Tables

Ward (wardName, ...)
Patient (patientNo, ...)

Relationship Tables

Contains (patientNo *, wardName *,....)

wardName	wardType
Harvey	orthopaedic
Pasteur	geriatic
Lister	cardiac

patientNo	patientName
1274	Castle
1288	Thomas
1289	Jones
1291	Watkins
1292	Radford
1296	Berry

patientNo*	wardName*
1289	Lister
1274	Lister
1288	Harvey
1292	Pasteur

Which patients are not in a ward?

One:Many relationships

Optional (1 : M) Optional

Each Ward can contain many Patients, but might not contain any.

Each Patient can be assigned to one and only one Ward, but may not be assigned to any (Outpatients).

The solution is to represent the relation Contains by a separate table. For each inpatient only, there will be an entry in the Contains table.

The solution is to represent the relationship **Contains** by a separate table. For each inpatient only, there will be an entry in the **Contains** table.

The table types would look like this:

Entity Tables

Ward (wardName, ...)
Patient (patientNo, ...)

Relationship Tables

Contains (wardName*, patientNo*, ...)

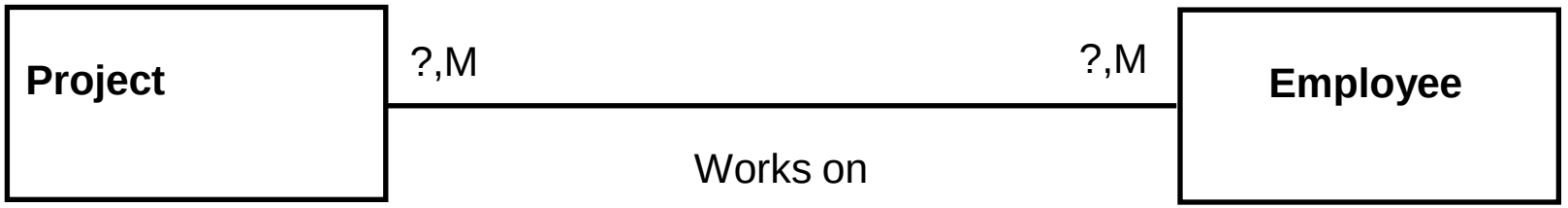
wardName	wardType
Harvey	orthopaedic
Pasteur	geriatic
Lister	cardiac
Jenner	orthopaedic

patientNo	patientName
1274	Castle
1288	Thomas
1289	Jones
1291	Watkins
1292	Radford
1296	Berry

wardName*	patientNo*
Lister	1289
Lister	1274
Jenner	1288
Lister	1292

Which patients are not in a ward?
Which wards do not have patients?

Mapping Many:Many relationships



As we have seen, posting projectNo into the Employee table would create a repeating group of projectNos, and posting the employeeName into the Project table would create a repeating group of employeeNames.

So the general rule for representing a many:many relationship, between two entities is as follows:

REGARDLESS OF PARTICIPATION CONDITIONS….

Define three tables, one for each entity and one for the relationship.

Project

projectNo	projectName
P02	HMV Website
P05	Dixons Database
P10	Amazon Website
P09	BBC iplayer upgrade

Employee

employeeNo	employeeName
1474	Castle
1488	Thomas
1489	Jones
1491	Watkins
1492	Radford
1496	Berry

WorksOn

projectNo*	employeeNo*
P02	1489
P10	1474
P09	1492
P02	1492

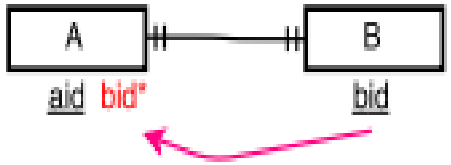
Which project doesn't have any employees assigned to it?

Which employees are not working on a project?

Which employee is working on more than one project?

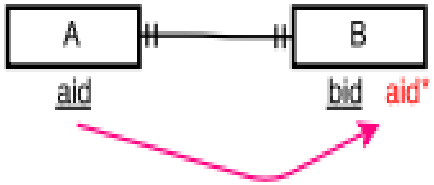
Which project has more than one employee working on it?

1:1


A
B
aid bid*
bid

XOR

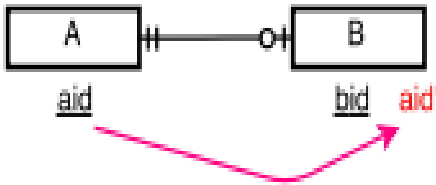

A
B


A
B
aid
bid aid*

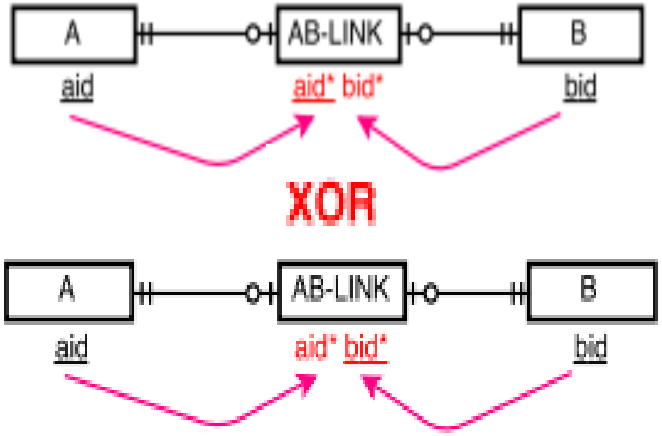
XOR


A
aid

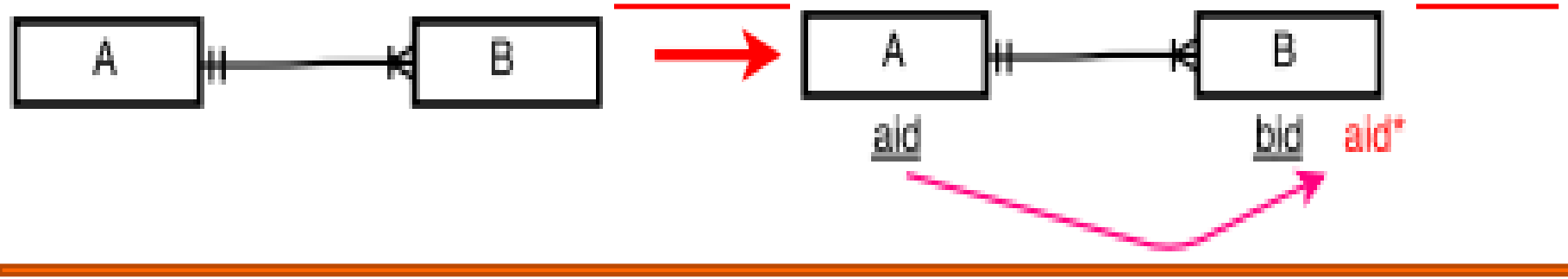

A
B

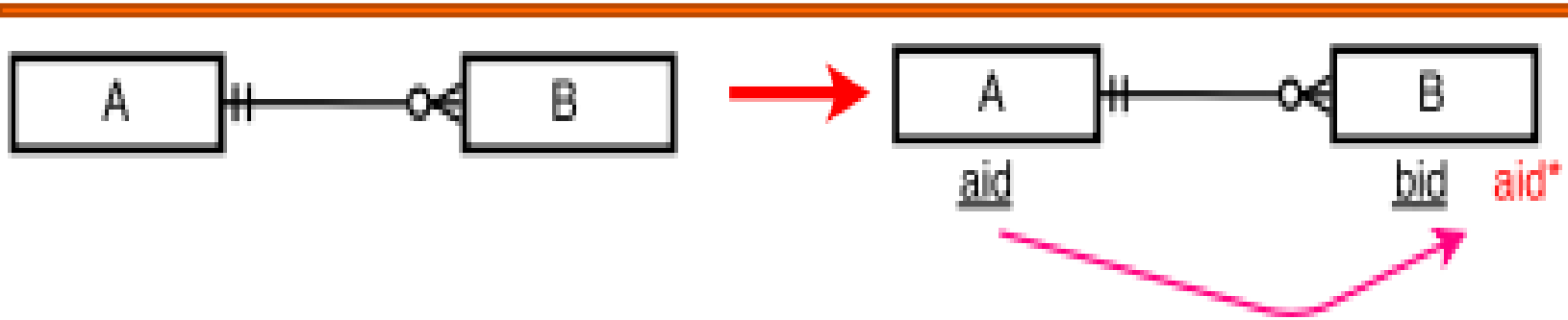

A
B
aid
bid aid*

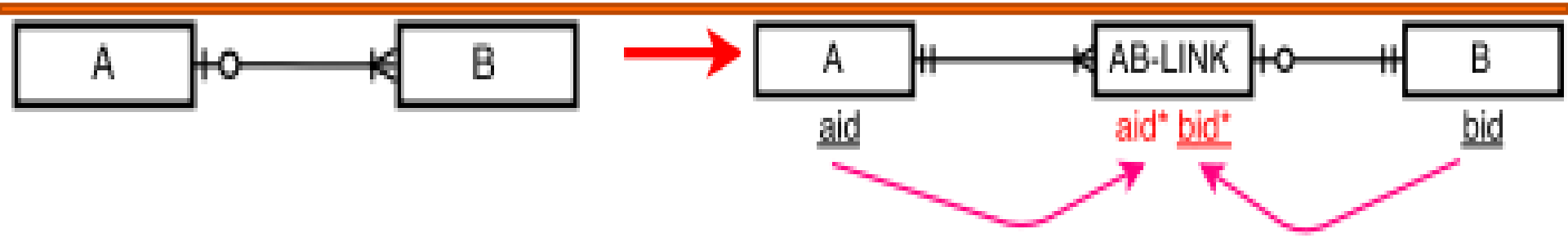

A
B

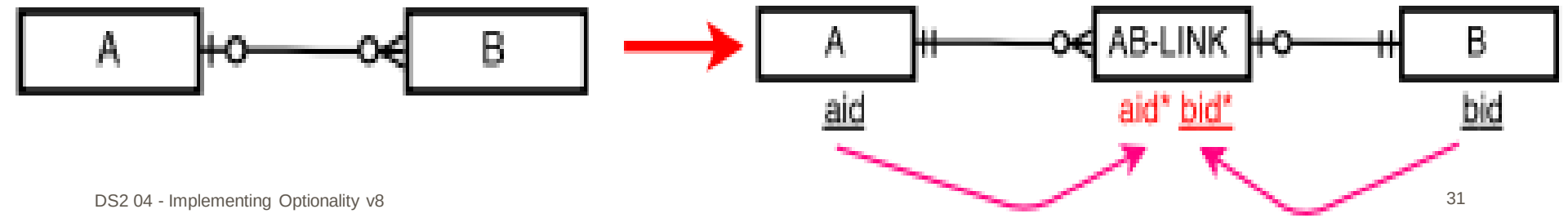

A
AB-LINK
B
aid
aid* bid*
bid
XOR
A
AB-LINK
B
aid
aid* bid*
bid

1:M


A
B
A
B
aid
bid
aid*

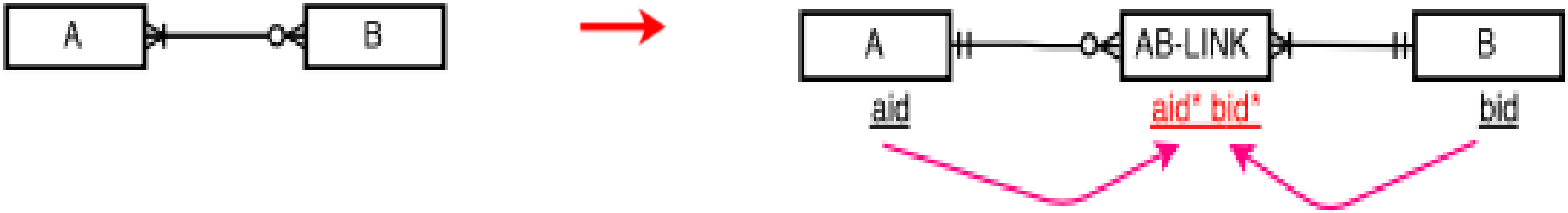

A
B
A
B
aid
bid
aid*


A
B
A
AB-LINK
B
aid
aid* bid*
bid


A
B
A
AB-LINK
B
aid
aid* bid*
bid

M:N


A
B
A
AB-LINK
B
aid
aid* bid*
bid


A
B
A
AB-LINK
B
aid
aid* bid*
bid


A
B
A
AB-LINK
B
aid
aid* bid*
bid