

# Conjunctive Queries: Fast Evaluation

(Chapter 18 of DBT)

# Complexity of Query Evaluation

**Theorem:** CQ-Evaluation is NP-complete and in PTIME in data complexity

**Proof:**

**(NP-membership)** Guess-and-check:

- Consider a database  $D$ , a CQ  $Q(x_1, \dots, x_k) :- \text{body}$ , and a tuple  $(a_1, \dots, a_k)$  of values
- Guess a substitution  $h : \text{terms}(\text{body}) \rightarrow \text{terms}(D)$
- Verify that  $h$  is a match of  $Q$  in  $D$ , i.e.,  $h(\text{body}) \subseteq D$  and  $(h(x_1), \dots, h(x_k)) = (a_1, \dots, a_k)$

**(NP-hardness)** Reduction from 3-colorability

**(in PTIME)** For every substitution  $h : \text{terms}(\text{body}) \rightarrow \text{terms}(D)$ , check if  $h(\text{body}) \subseteq D$  and  $(h(x_1), \dots, h(x_k)) = (a_1, \dots, a_k)$

# Complexity of Query Evaluation

**Theorem:** CQ-Evaluation is NP-complete and in PTIME in data complexity

Evaluating a CQ  $Q$  over a database  $D$  takes time  $|D|^{O(|Q|)}$

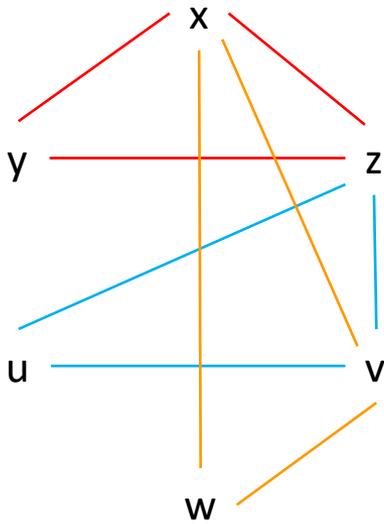
isolate “good” classes of CQs for which query evaluation is tractable **in combined complexity**

Graph-based vs Hypergraph-based classes

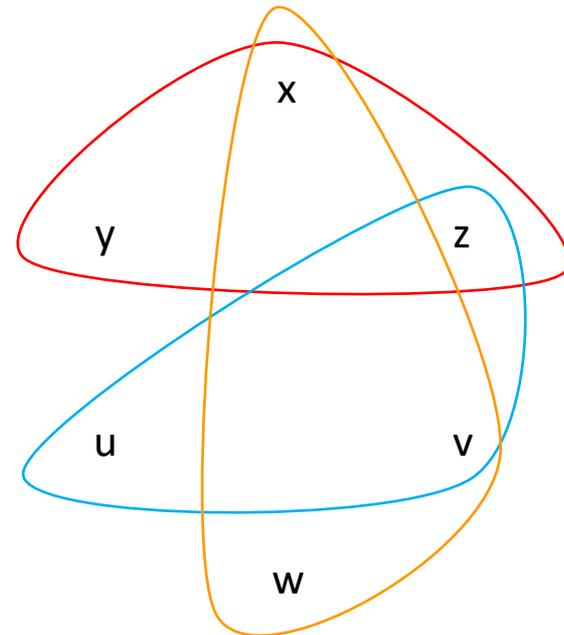
# (Hyper)graph of Conjunctive Queries

$Q \text{ :- } R(x,y,z), R(z,u,v), R(v,w,x)$

graph of  $Q$  -  $G(Q)$



hypergraph of  $Q$  -  $H(Q)$



# “Good” Classes of Conjunctive Queries

measures how close a graph is to a tree

- Graph-based

- CQs of bounded **treewidth** - their graph has bounded treewidth

measures how close a hypergraph is to an acyclic one

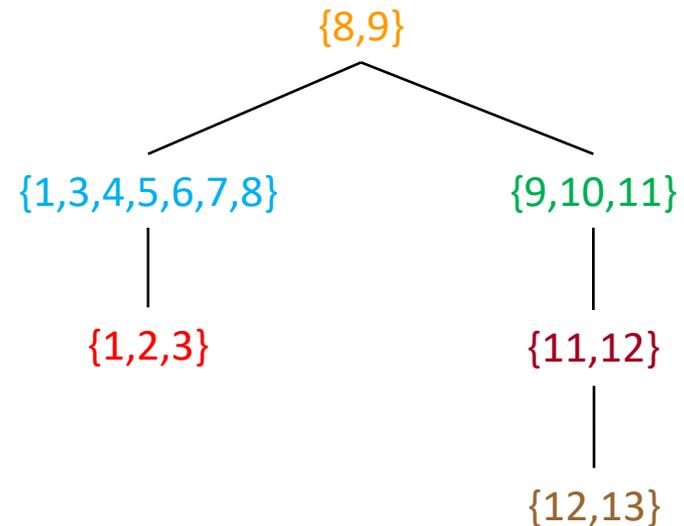
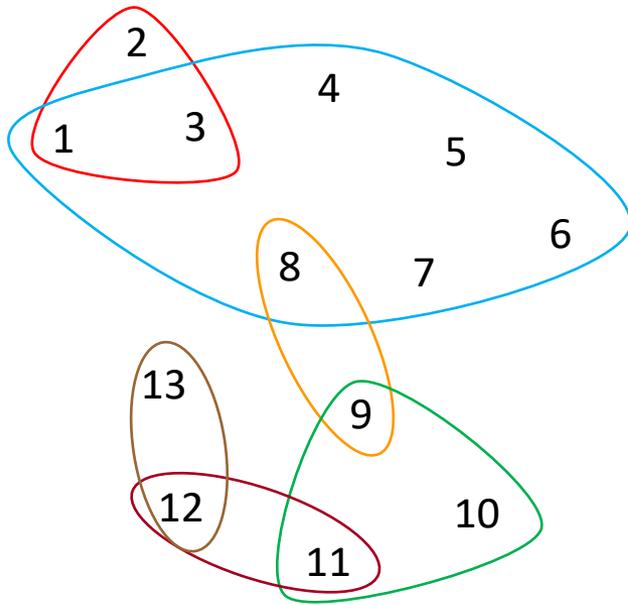
- Hypergraph-based:

- CQs of bounded **hypertree width** - their hypergraph has bounded hypertree width
- **Acyclic CQs** - their hypergraph has **hypertree width 1**

# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

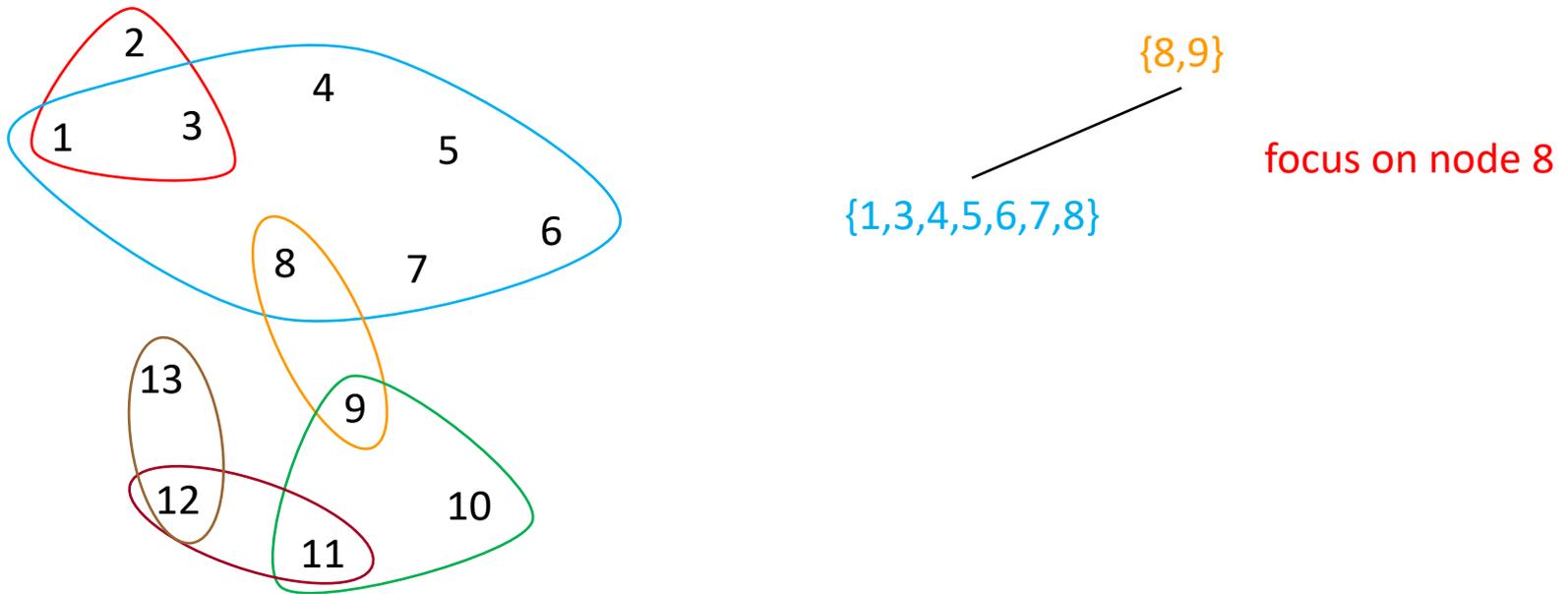
1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$



# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

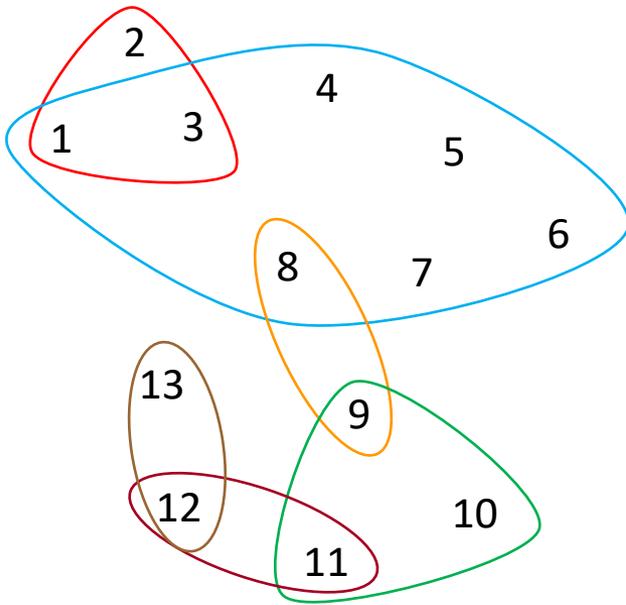
1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$



# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$



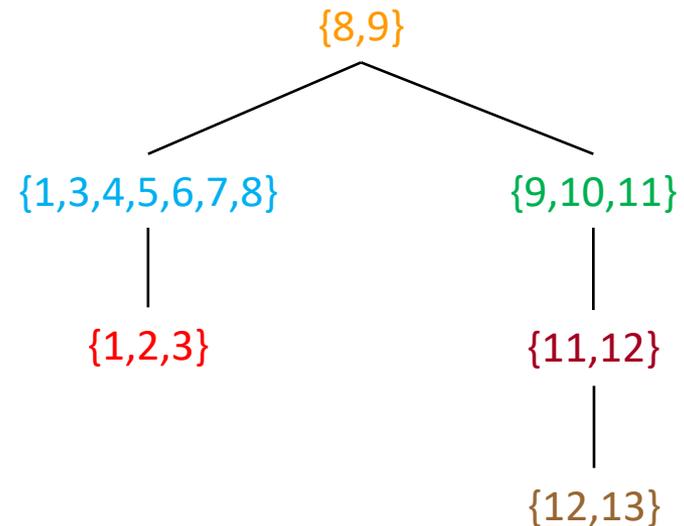
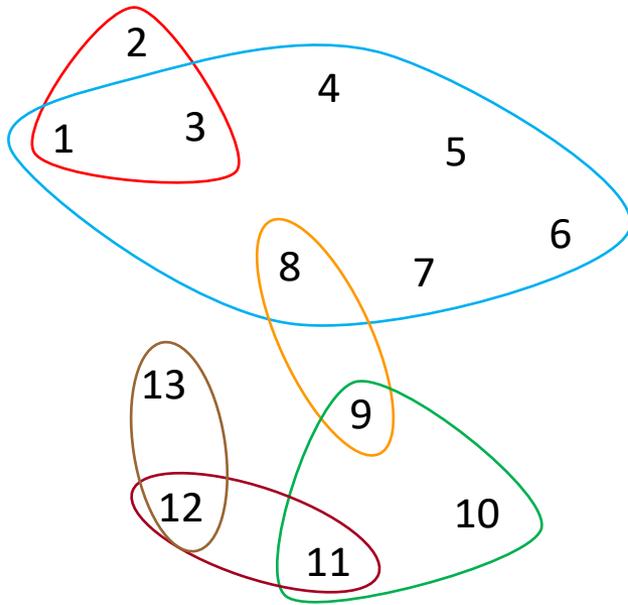
focus on node 11



# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

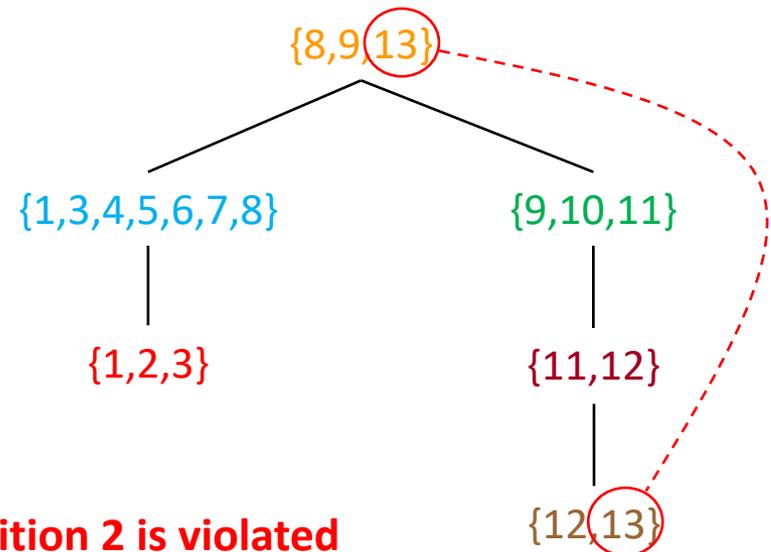
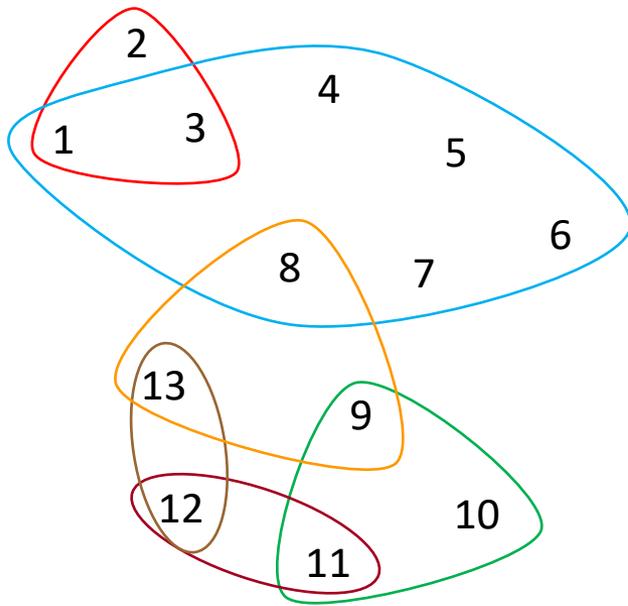
1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$



# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$



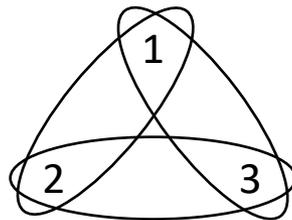
**condition 2 is violated**

# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$

**Definition:** A hypergraph is **acyclic** if it has a join tree



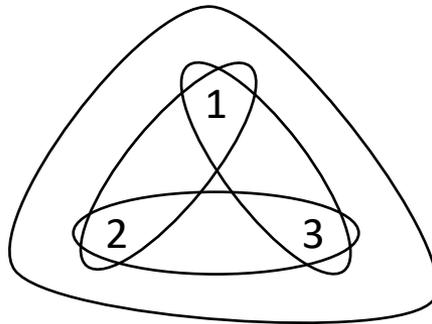
prime example of a cyclic hypergraph

# Acyclic Hypergraphs

A **join tree** of a hypergraph  $\mathbf{H} = (V, E)$  is a labeled tree  $\mathbf{T} = (N, F, L)$ , where  $L : N \rightarrow E$  such that:

1. For each hyperedge  $e \in E$  of  $\mathbf{H}$ , there exists  $n \in N$  such that  $e = L(n)$
2. For each node  $u \in V$  of  $\mathbf{H}$ , the set  $\{n \in N \mid u \in L(n)\}$  induces a connected subtree of  $\mathbf{T}$

**Definition:** A hypergraph is **acyclic** if it has a join tree



but this is acyclic

# Relevant Algorithmic Tasks

ACYCLICITY

**Input:** a conjunctive query  $Q$

**Question:** is  $Q$  acyclic? or is  $H(Q)$  acyclic?

$\{Q \in \mathbf{CQ} \mid H(Q) \text{ is acyclic}\}$



**ACQ-Evaluation**

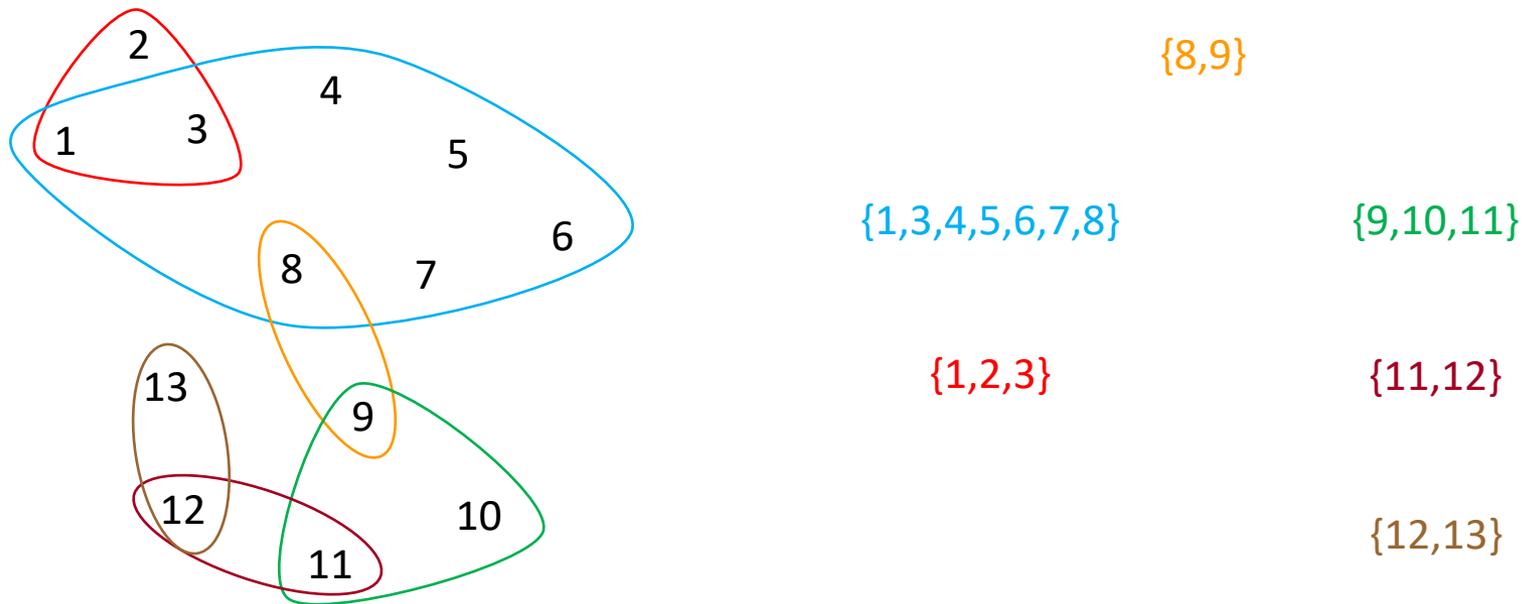
**Input:** a database  $D$ , an acyclic conjunctive query  $Q$ , and a tuple  $(a_1, \dots, a_k)$  of values

**Question:**  $(a_1, \dots, a_k) \in Q(D)$ ?

# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

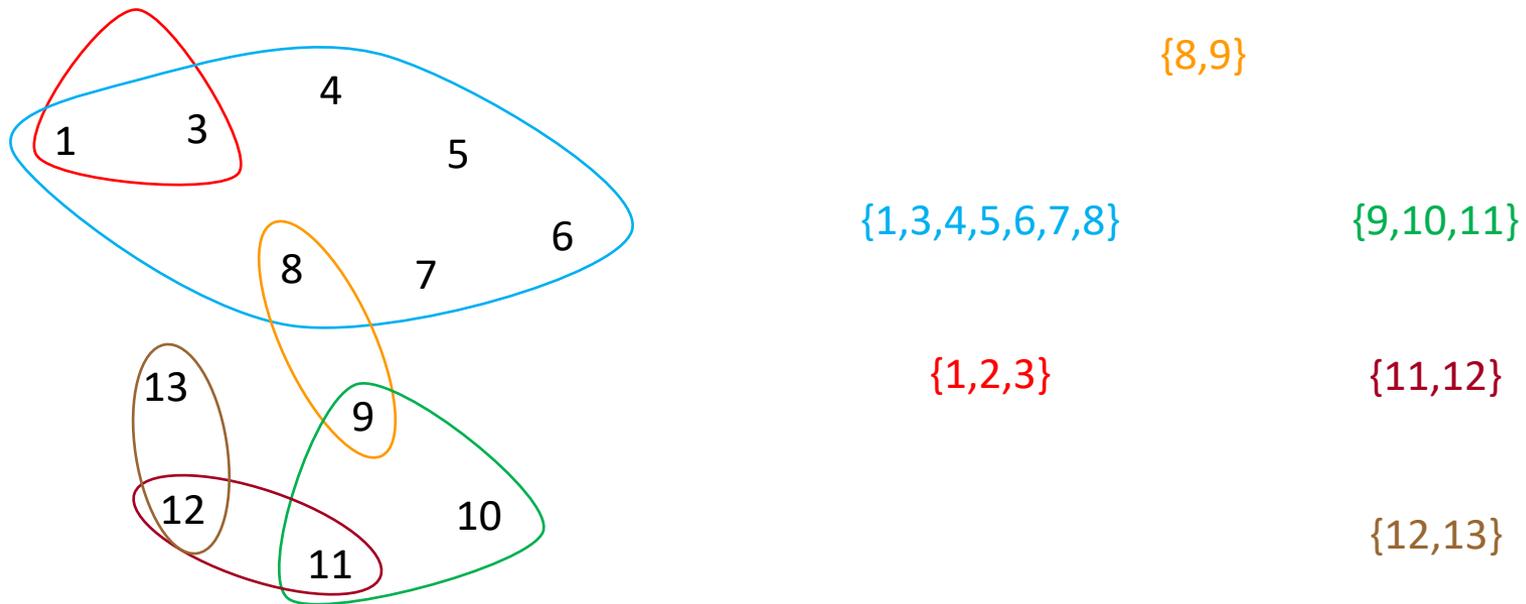
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

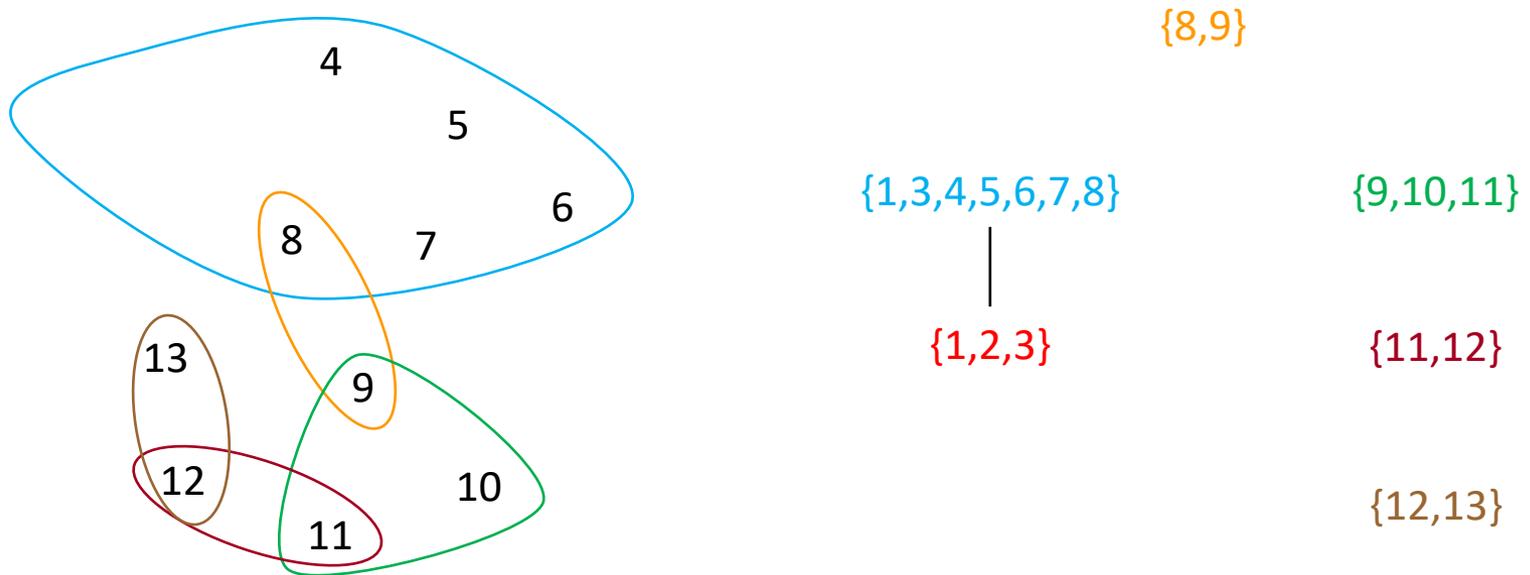
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

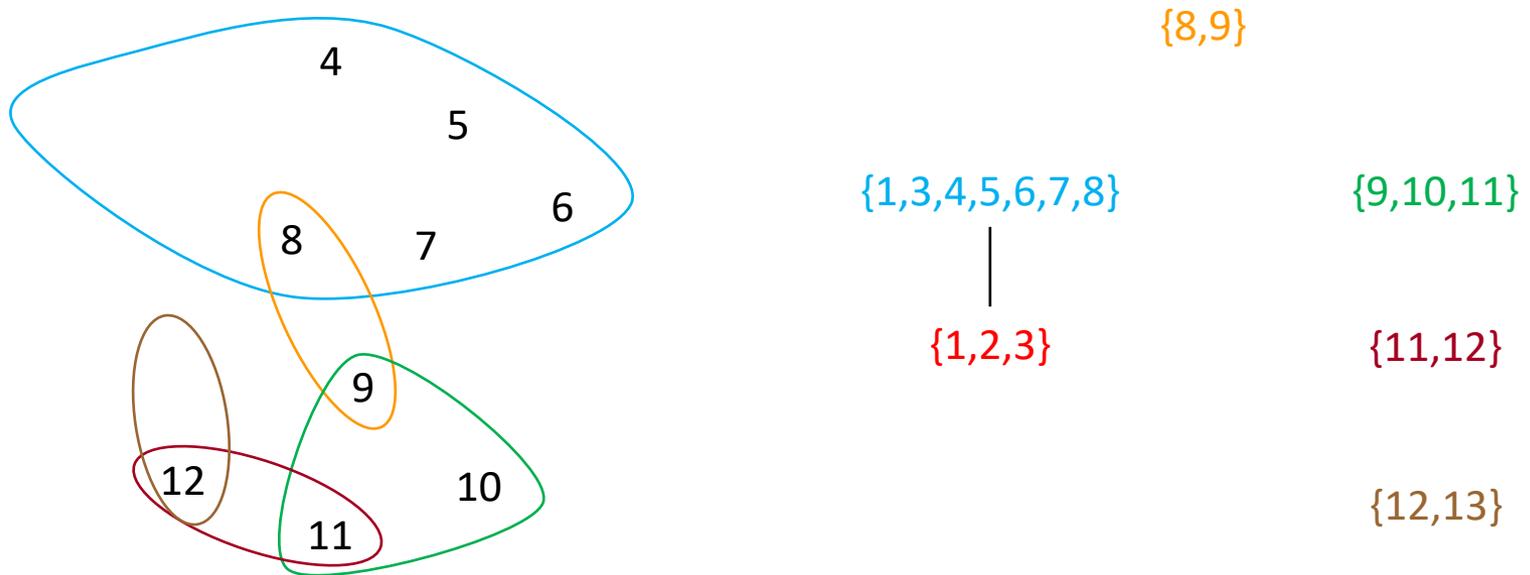
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

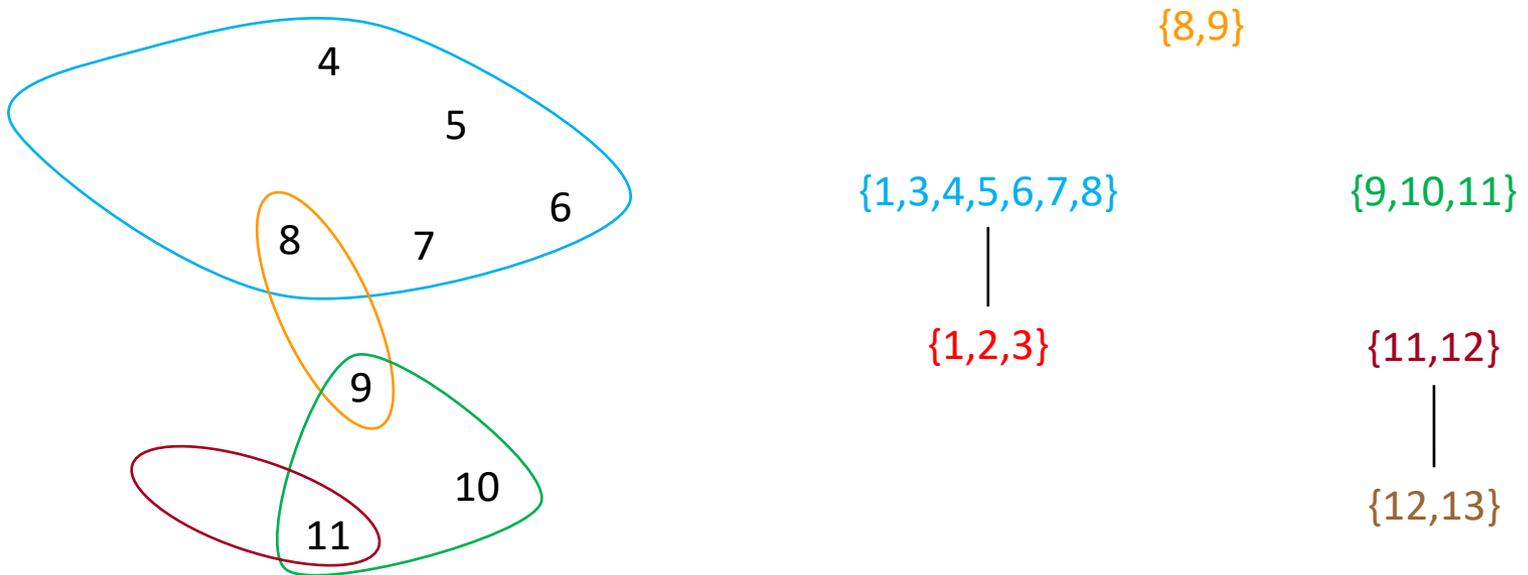
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

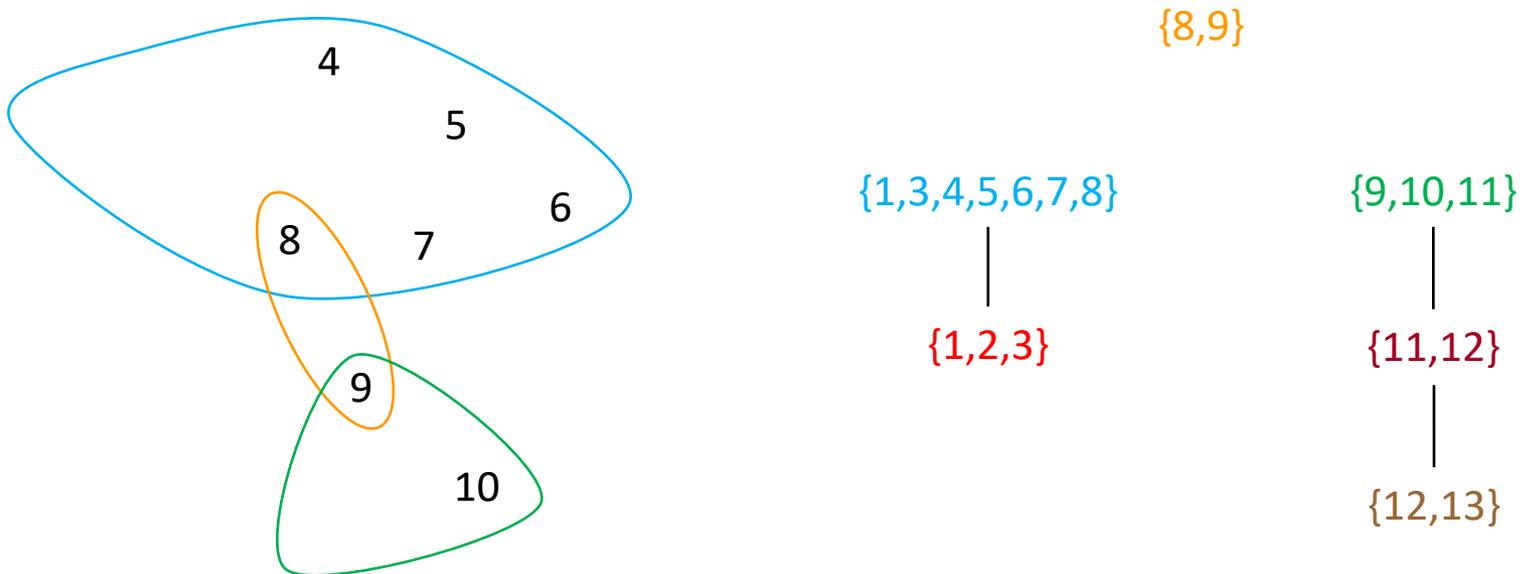
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

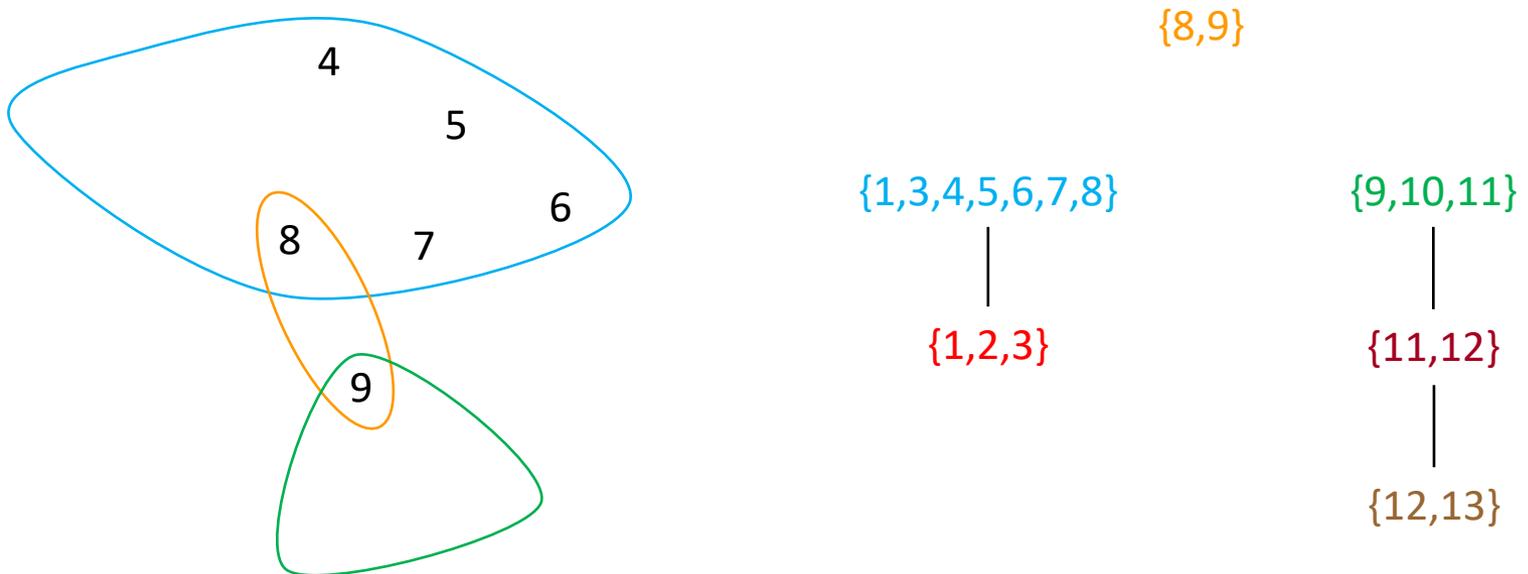
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

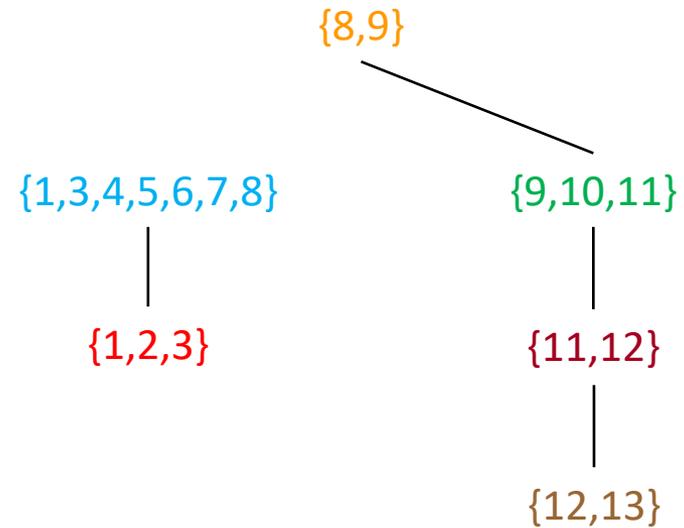
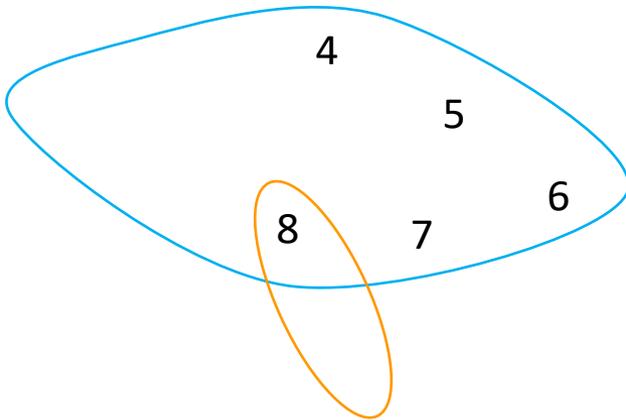
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

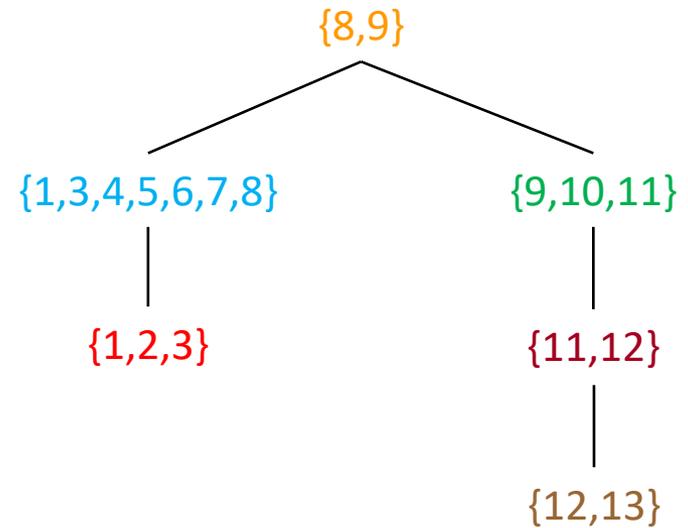
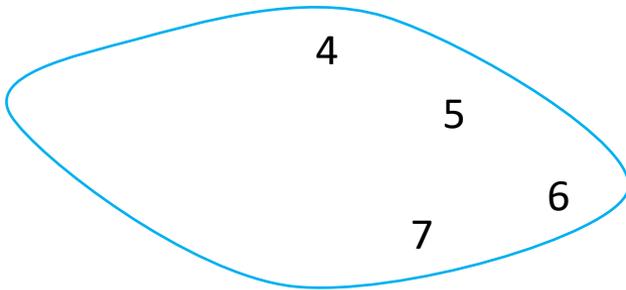
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

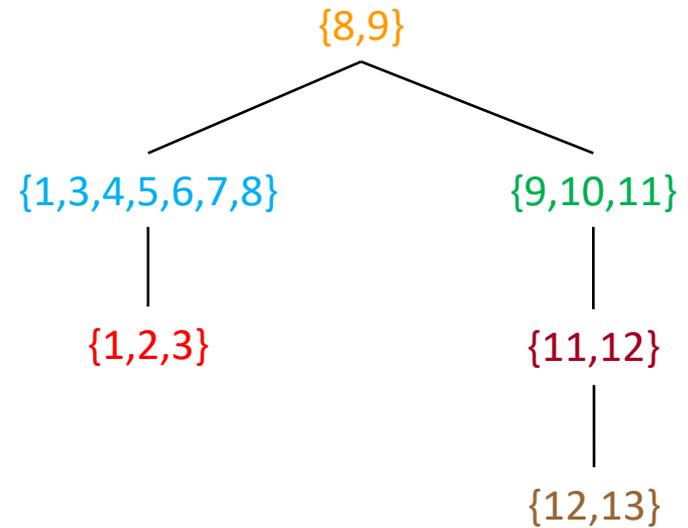
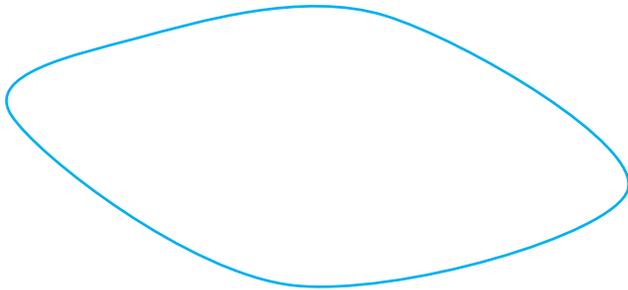
1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges

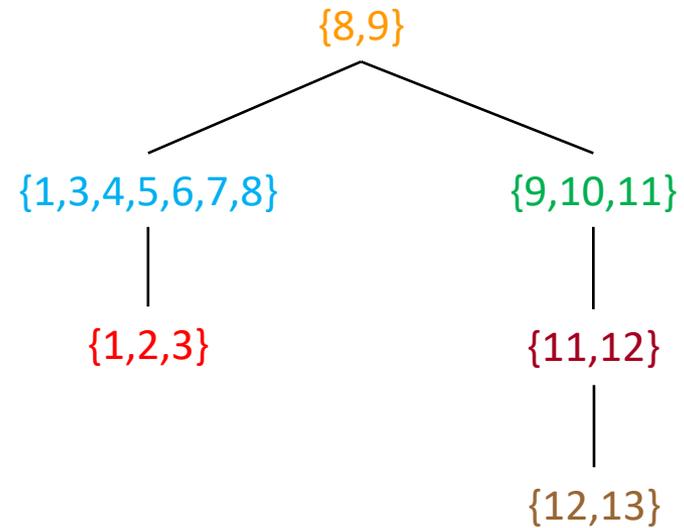


# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges

***empty hypergraph***



# Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges

**Theorem:** A hypergraph  $\mathbf{H}$  is acyclic iff  $\text{GYO}(\mathbf{H}) = \emptyset$



checking whether  $\mathbf{H}$  is acyclic is feasible in polynomial time, and if it is the case, a join tree can be found in polynomial time



**Theorem:** ACYCLICITY is in PTIME

# Checking Acyclicity

**Theorem:** ACYCLICITY is in PTIME

**NOTE:** actually, we can check whether a CQ is acyclic in time  $O(|Q|)$

**linear time** in the size  $Q$

# Evaluating Acyclic CQs

**Theorem:** ACQ-Evaluation is in PTIME

**NOTE:** actually, if  $H(Q)$  is acyclic, then  $Q$  can be evaluated in time  $O(|D| \cdot |Q|)$   
**linear time** in the size of  $D$  and  $Q$

# Yannakaki's Algorithm

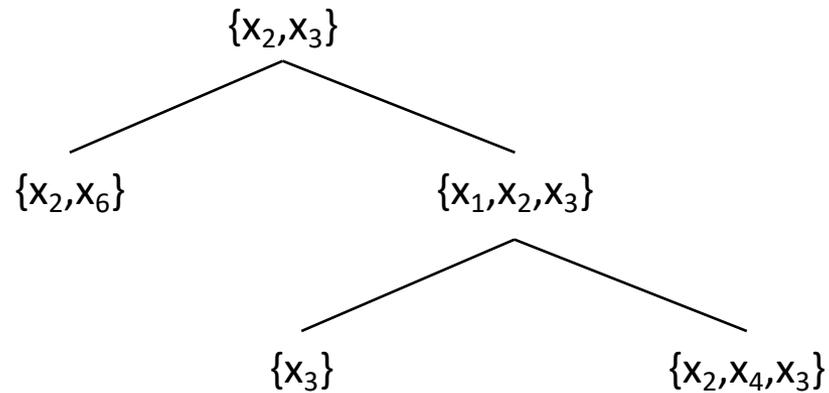
Dynamic programming algorithm over the join tree

Given a database  $D$  and an acyclic Boolean CQ  $Q$

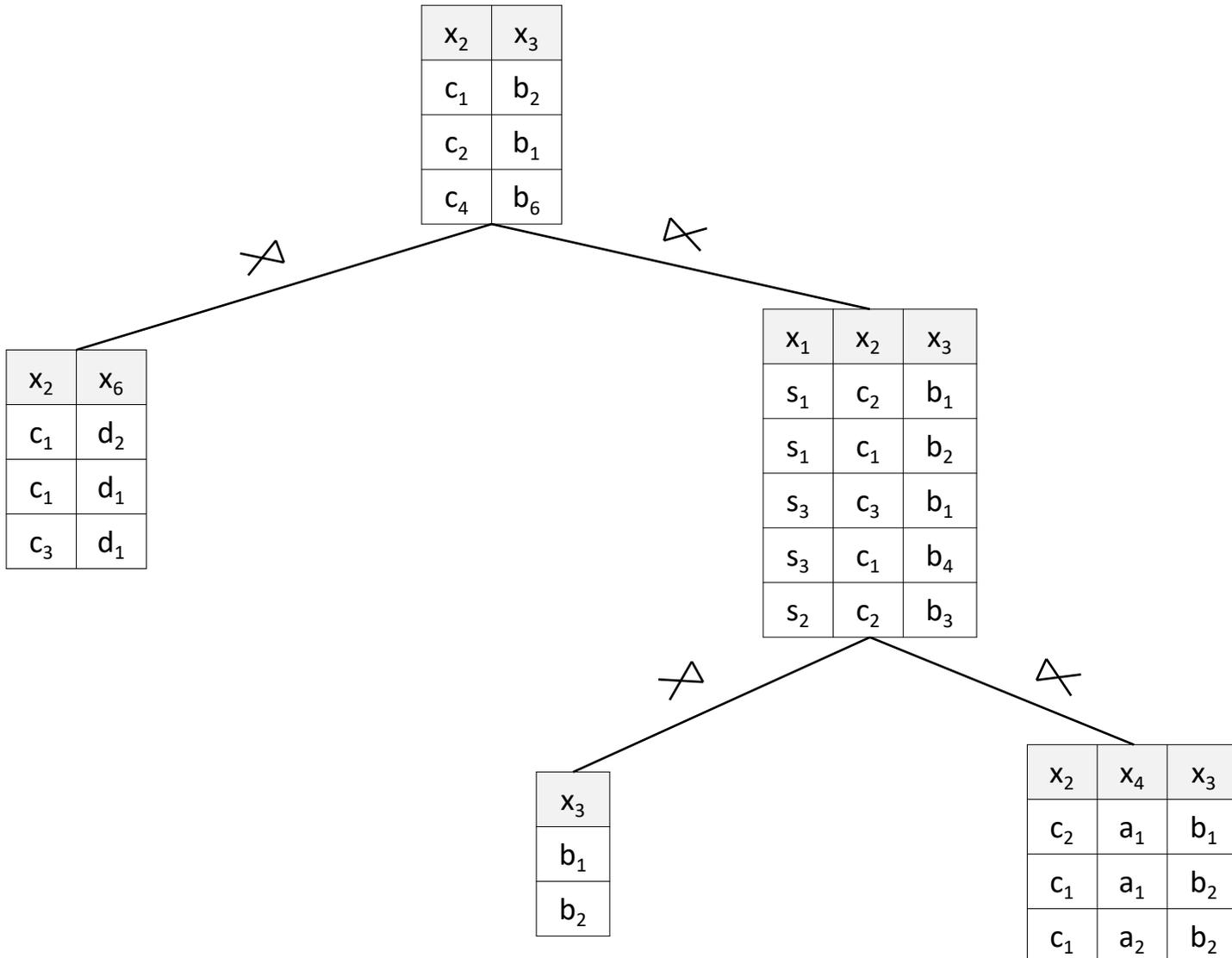
1. Compute the join tree  $T$  of  $H(Q)$
2. Assign to each node of  $T$  the corresponding relation of  $D$
3. Compute semi-joins in a bottom up traversal of  $T$
4. Return YES if the resulting relation at the root of  $T$  is non-empty;  
otherwise, return NO

# Yannakaki's Algorithm: Step 1

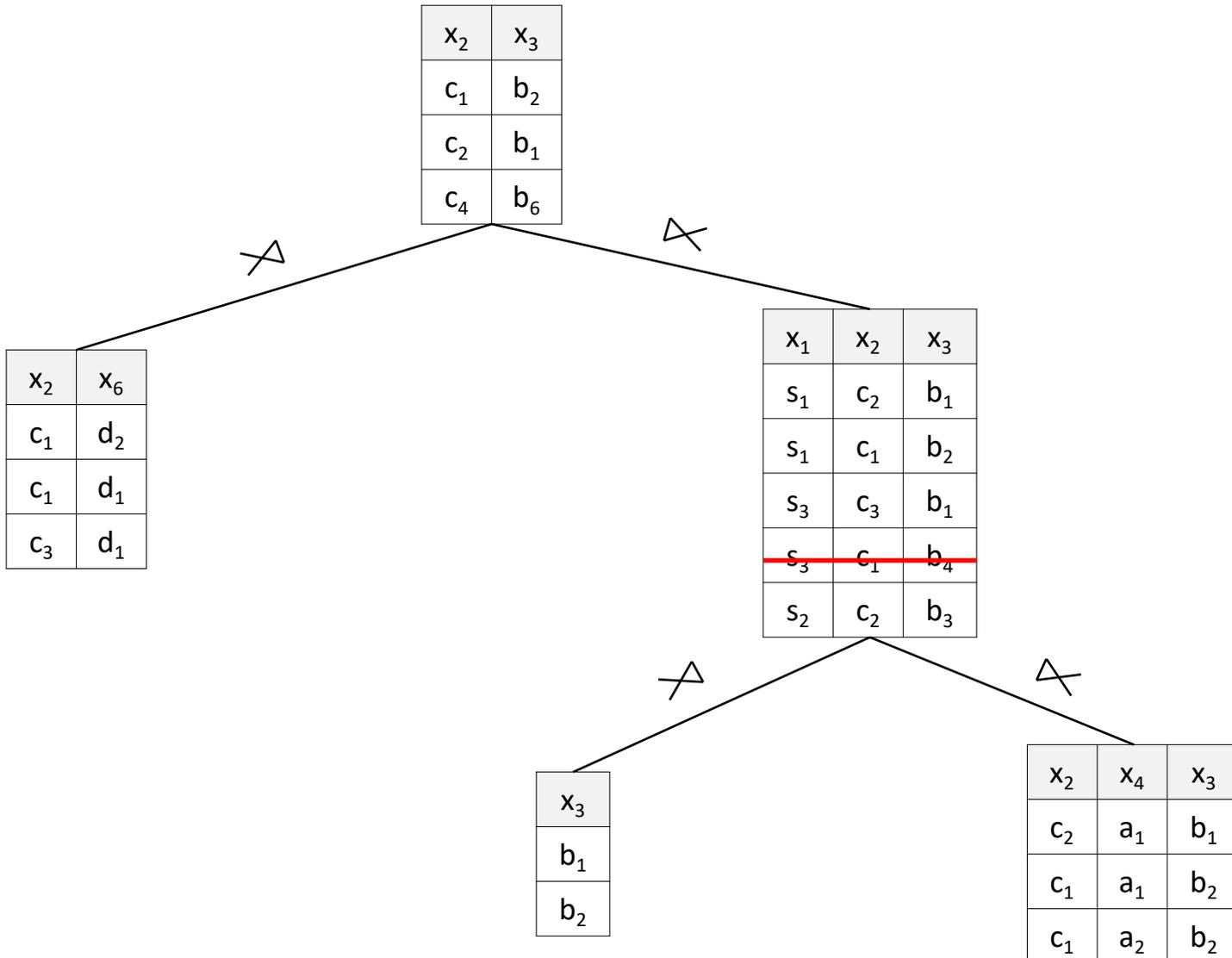
Q :-  $R_1(x_1, x_2, x_3), R_2(x_2, x_3), R_3(x_2, x_6), R_4(x_3), R_5(x_2, x_4, x_3)$



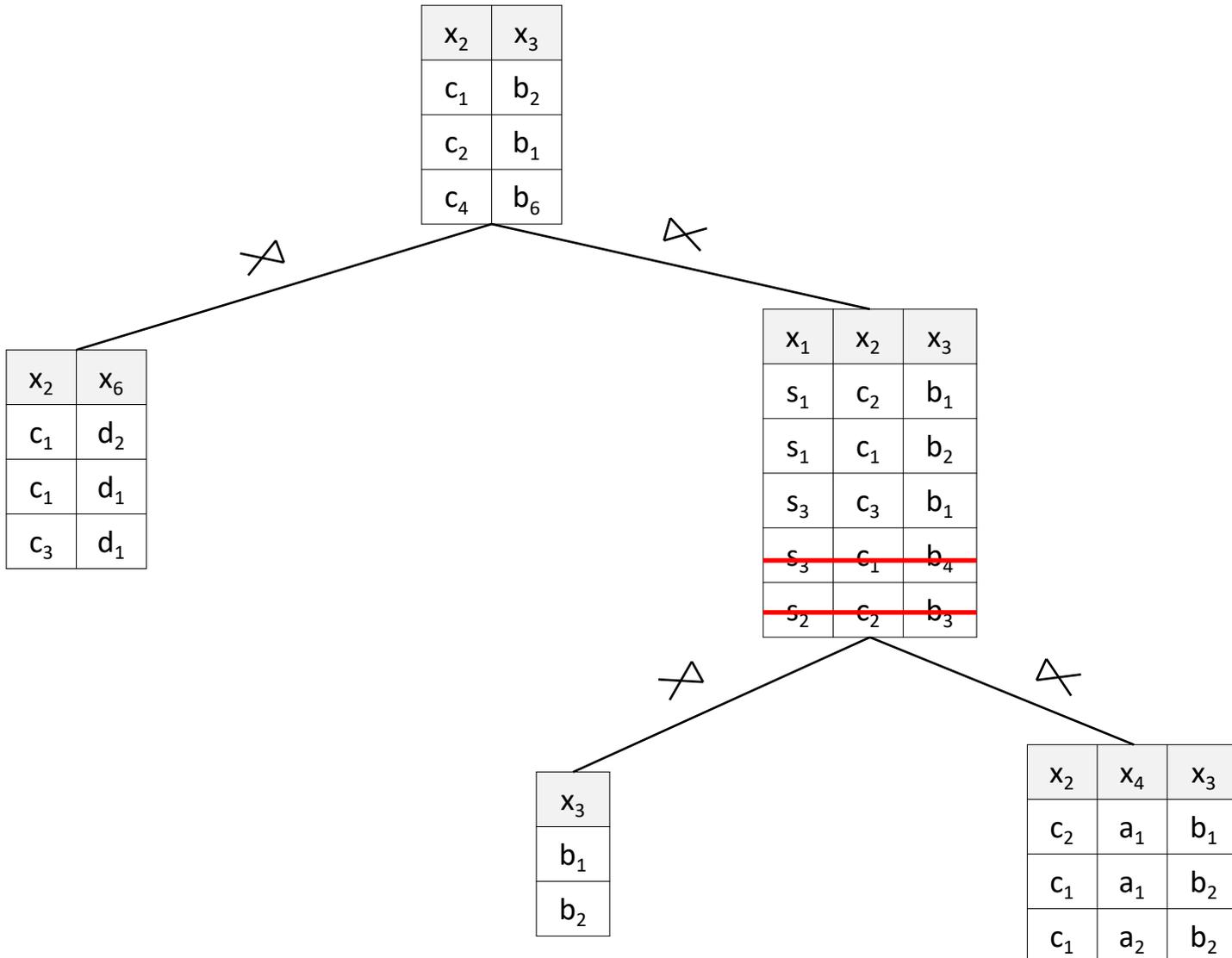
# Yannakaki's Algorithm: Step 2



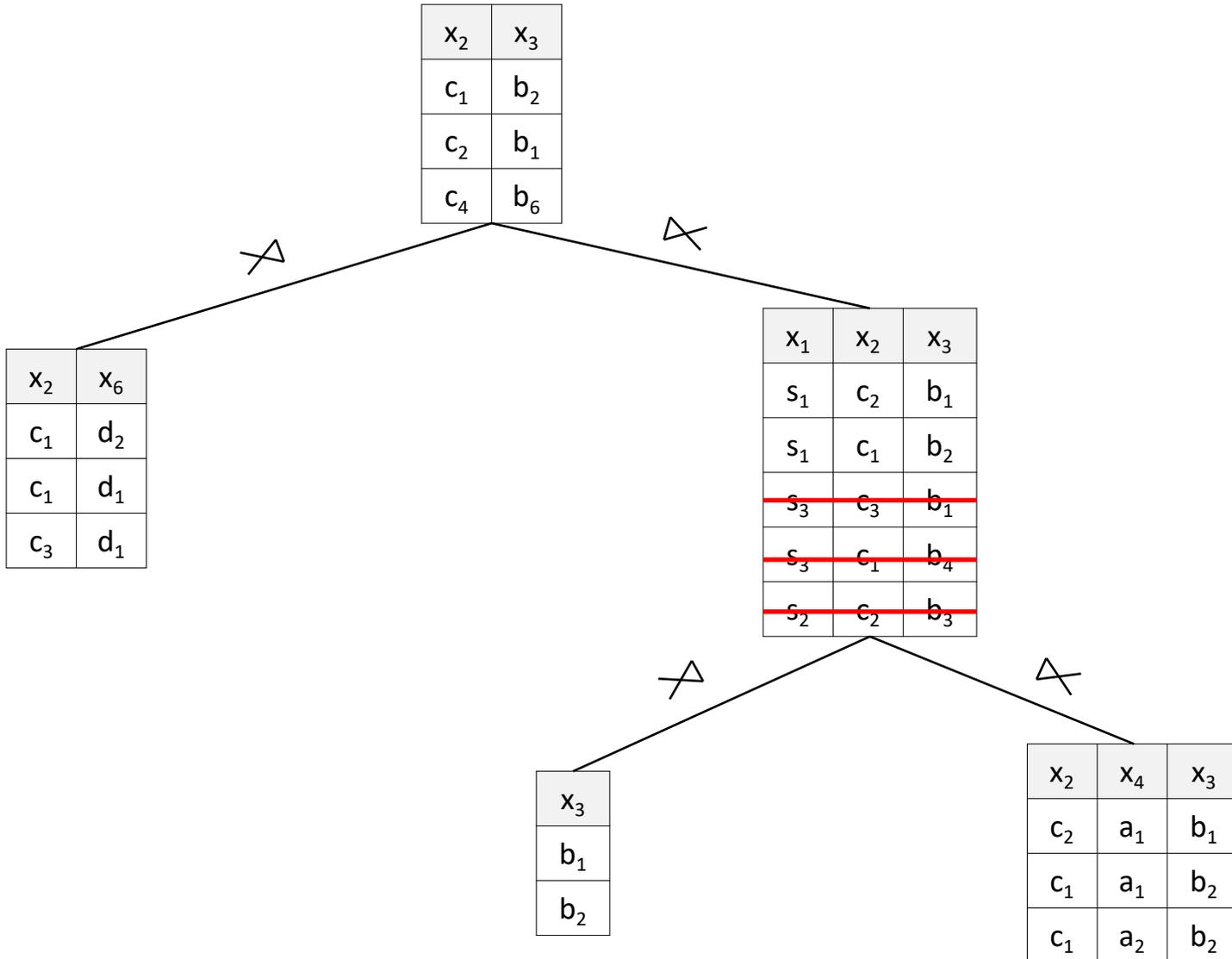
# Yannakaki's Algorithm: Step 3



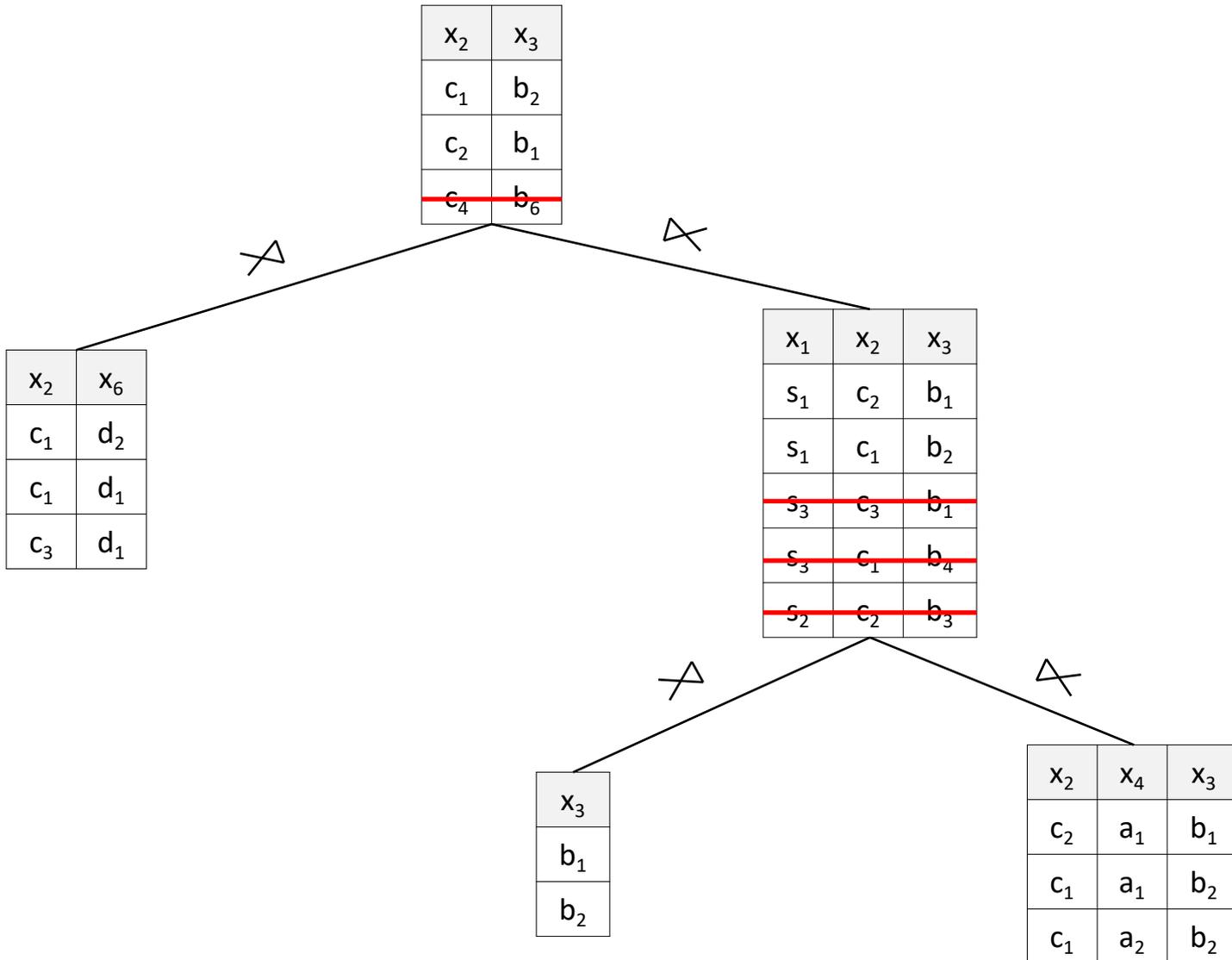
# Yannakaki's Algorithm: Step 3



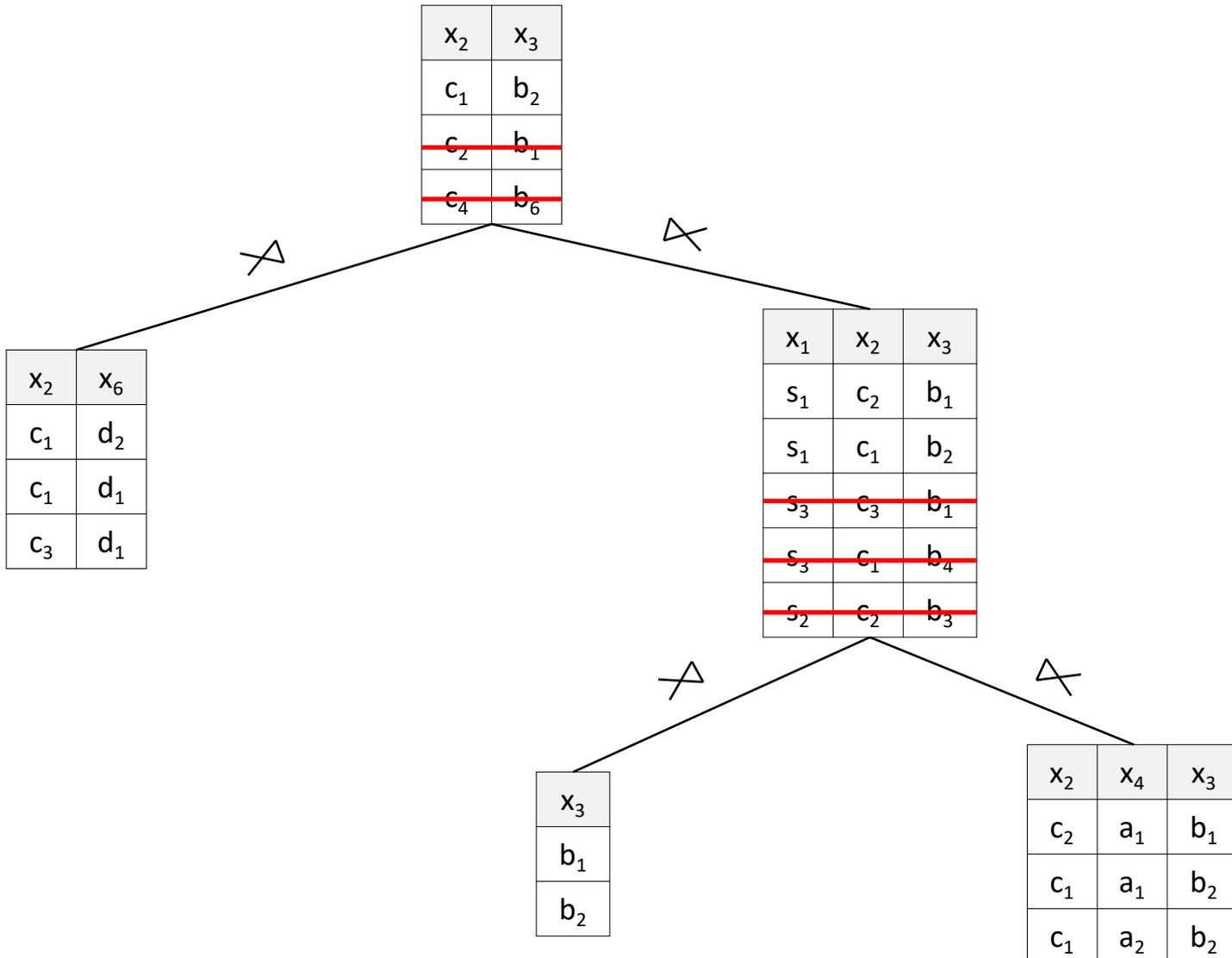
# Yannakaki's Algorithm: Step 3



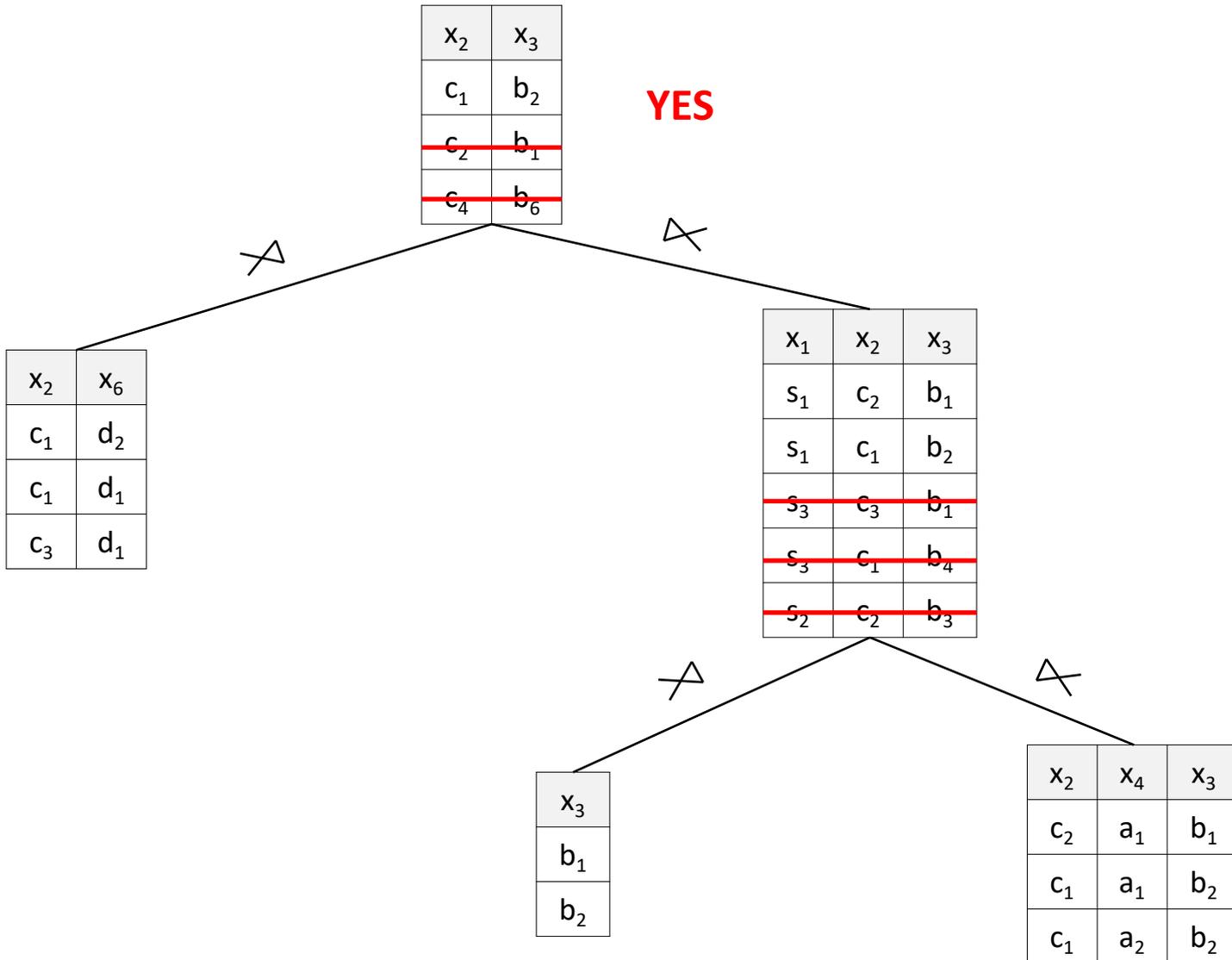
# Yannakaki's Algorithm: Step 3



# Yannakaki's Algorithm: Step 3



# Yannakaki's Algorithm: Step 4



# Recap

- “Good” classes of CQs for which query evaluation is tractable - conditions based on the graph or hypergraph of the CQ
- Acyclic CQs - their hypergraph is acyclic, can be checked in linear time
- Evaluating acyclic CQs is feasible in linear time (Yannakaki’s algorithm)