Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Data Profiling

Functional Dependency Discovery

Thorsten Papenbrock
G-3.1.09, Campus III
Hasso Plattner Institut

Definition 2.1 (Functional dependency)**.** Given a relational instance r for a schema R. The *functional dependency* $X \rightarrow A$ with $X \subseteq R$ and $A \in R$ is *valid* in r, iff $\forall t_i, t_j \in r : t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A]$.

FD discovery is in $\mathcal{O}(n^2 \cdot 2^m \cdot (\frac{m}{2})^2)$

Overview

Functional Dependency Discovery

- Motivation
- Approaches
- fdep
- HyFD
- Evaluation
- Metanome

Functional Dependency Discovery

- **Motivation**
- Approaches
- fdep
- HyFD
- Evaluation
- Metanome

Functional Dependency

ID	Name	Size	Type	Weak	Strong	GYM	Leader	Reward
25	Pikachu	0.4	electric	ground	water	Vermillion	Lt. Surge	Thunder
26	Raichu	0.8	electric	ground	water	Vermillion	Lt. Surge	Thunder
29	Nidoran	0.5	poison	ground	gras	Viridian	Giovanni	Earth
37	Vulpix	0.6	fire	water	ice	null	null	null
38	Ninetails	1.1	fire	water	ice	null	null	null
63	Abra	0.9	psychic	ghost	fighting	null	null	null
64	Kadabra	1.3	psychic	ghost	fighting	Saffron	Sabrina	Marsh
65	Alakazam	1.5	psychic	ghost	fighting	Saffron	Sabrina	Marsh
150	Mewtwo	2.0	psychic	ghost	fighting	null	null	null

http://bulbapedia.bulbagarden.net

Data Profiling

FD Discovery

Functional Dependency

ID	Name	Size	Type	Weak	Strong	GYM	Leader	Reward
25	Pikachu	0.4	electric	ground	water	Vermillion	Lt. Surge	Thunder
26	Raichu	0.8	electric	ground	water	Vermillion	Lt. Surge	Thunder
29	Nidoran	0.5	poison	ground	gras	Viridian	Giovanni	Earth
37	Vulpix	0.6	fire	water	ice	null	null	null
38	Ninetails	1.1	fire	water	ice	null	null	null
63	Abra	0.9	psychic	ghost	fighting	null	null	null
64	Kadabra	1.3	psychic	ghost	fighting	Saffron	Sabrina	Marsh
65	Alakazam	1.5	psychic	ghost	fighting	Saffron	Sabrina	Marsh
150	Mewtwo	2.0	psychic	ghost	fighting	null	null	null

http://bulbapedia.bulbagarden.net

Definition FD: **X** → **A**

- All values in X uniquely define the values in A.
- If $t_1[X] = t_2[X]$, then $t_1[A] = t_2[A]$.

Functional Dependency

HPI Hasso Plattner Institut

Redundancy!

ID	Name	Size	Type	Weak	Strong	GYM	Leader	Reward
25	Pikachu	0.4	electric	ground	water	Vermillion	Lt. Surge	Thunder
26	Raichu	0.8	electric	ground	water	Vermillion	Lt. Surge	Thunder
29	Nidoran	0.5	poison	ground	gras	Viridian	Giovanni	Earth
37	Vulpix	0.6	fire	water	ice	null	null	null
38	Ninetails	1.1	fire	water	ice	null	null	null
63	Abra	0.9	psychic	ghost	fighting	null	null	null
64	Kadabra	1.3	psychic	ghost	fighting	Saffron	Sabrina	Marsh
65	Alakazam	1.5	psychic	ghost	fighting	Saffron	Sabrina	Marsh
150	Mewtwo	2.0	psychic	ghost	fighting	null	null	null

Type → Weak, Strong

GYM → Leader, Reward

http://bulbapedia.bulbagarden.net

Data Profiling
FD Discovery

Motivation

Functional Dependency

-25 cells!

ID	Name	Size	Type	GYM
25	Pikachu	0.4	electric	Vermillion
26	Raichu	0.8	electric	Vermillion
29	Nidoran	0.5	poison	Viridian
37	Vulpix	0.6	fire	null
38	Ninetails	1.1	fire	null
63	Abra	0.9	psychic	null
64	Kadabra	1.3	psychic	Saffron
65	Alakazam	1.5	psychic	Saffron
150	Mewtwo	2.0	psychic	null

Type	Weak	Strong
electric	ground	water
poison	ground	gras
fire	water	ice
psychic	ghost	fighting

GYM	Leader	Reward
Vermillion	Lt. Surge	Thunder
Viridian	Giovanni	Earth
Saffron	Sabrina	Marsh

http://bulbapedia.bulbagarden.net

Data Profiling

FD Discovery

Motivation

Functional Dependency Discovery

ID	Name	Size	Type	Weak	Strong	GYM	Leader	Reward
25	Pikachu	0.4	electric	ground	water	Vermillion	Lt. Surge	Thunder
26	Raichu	0.8	electric	ground	water	Vermillion	Lt. Surge	Thunder
29	Nidoran	0.5	poison	ground	gras	Viridian	Giovanni	Earth
37	Vulpix	0.6	fire	water	ice	null	null	null
38	Ninetails	1.1	fire	water	ice	null	null	null
63	Abra	0.9	psychic	ghost	fighting	null	null	null
64	Kadabra	1.3	psychic	ghost	fighting	Saffron	Sabrina	Marsh
65	Alakazam	1.5	psychic	ghost	fighting	Saffron	Sabrina	Marsh
150	Mewtwo	2.0	psychic	ghost	fighting	null	null	null

Type → Weak, Strong

GYM → Leader, Reward

http://bulbapedia.bulbagarden.net

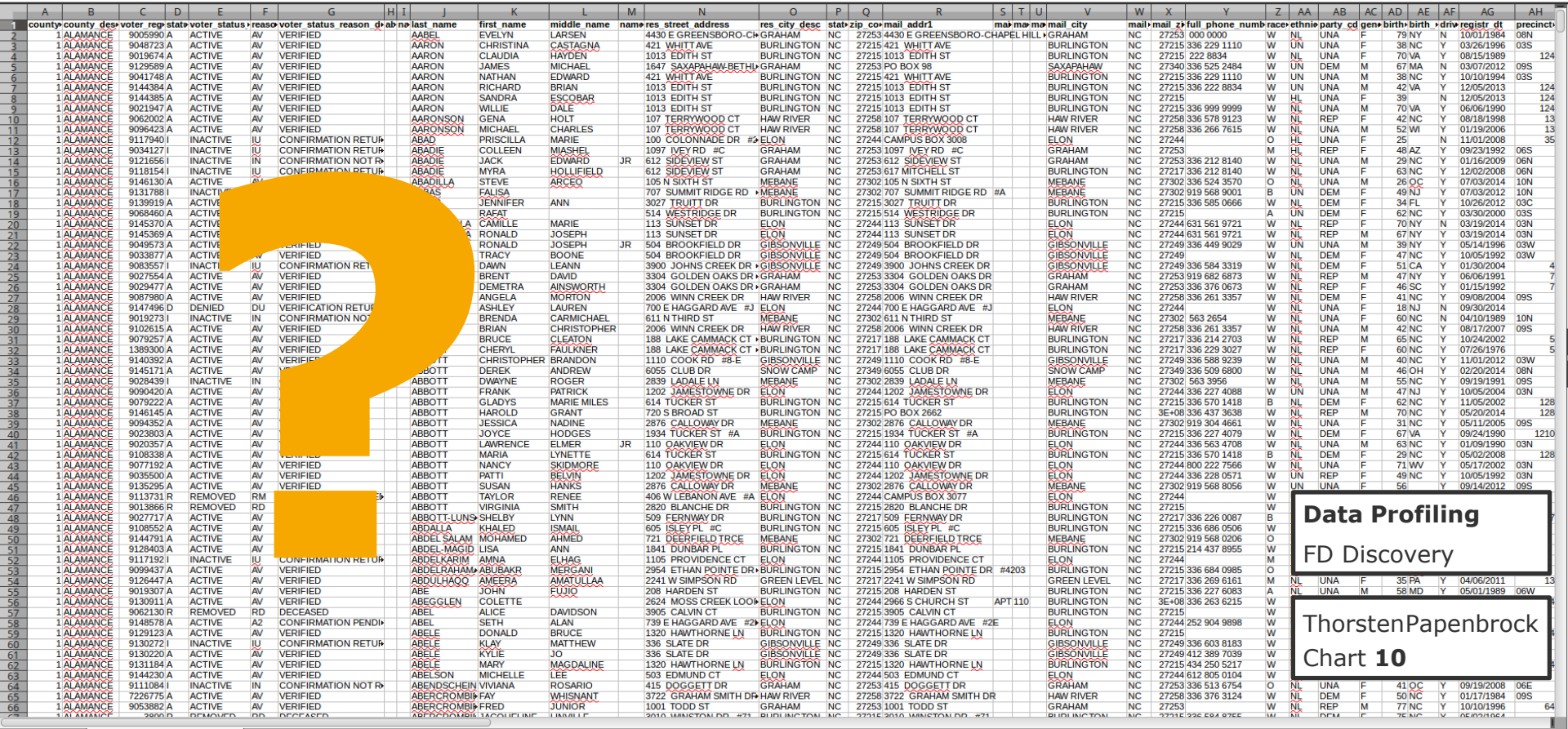
Data Profiling

FD Discovery

ThorstenPapenbrock

Motivation

Functional Dependency Discovery



Data Profiling

FD Discovery

ThorstenPapenbrock

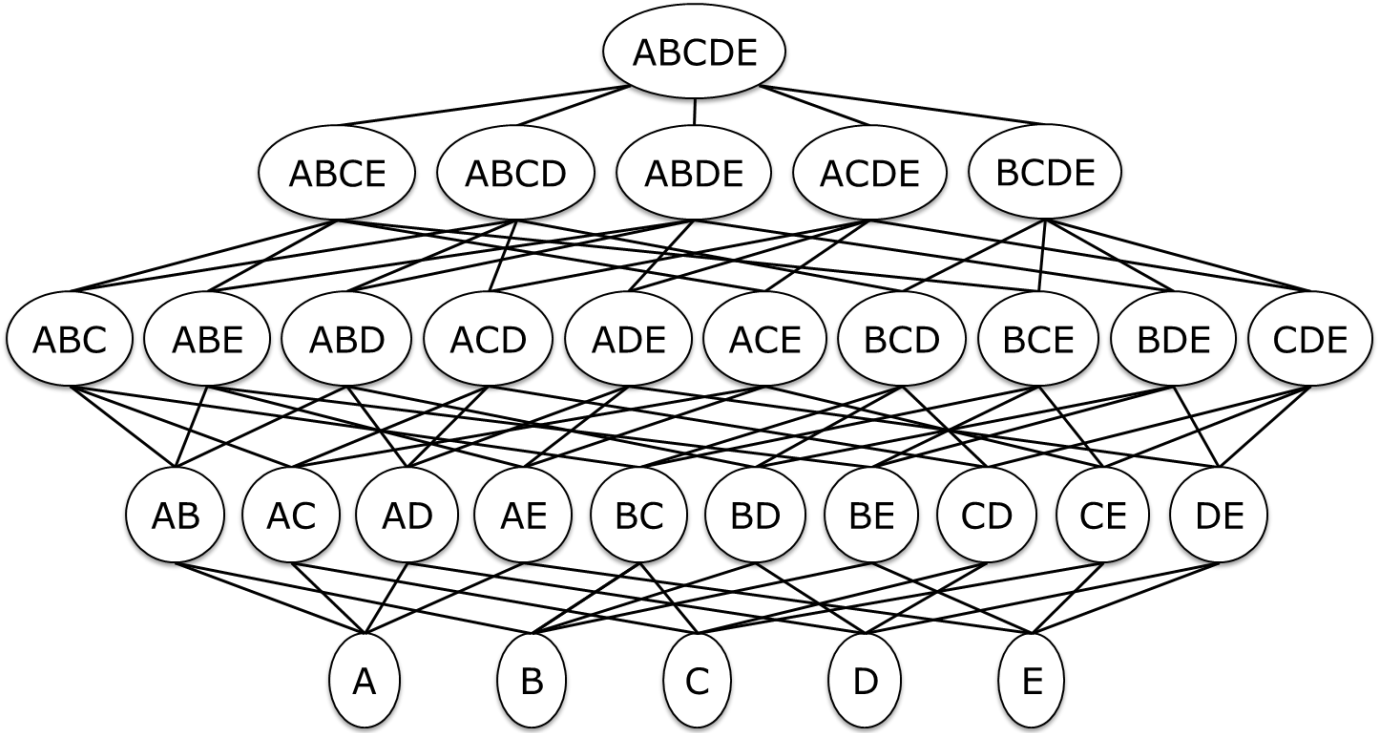
Chart **10**

Overview

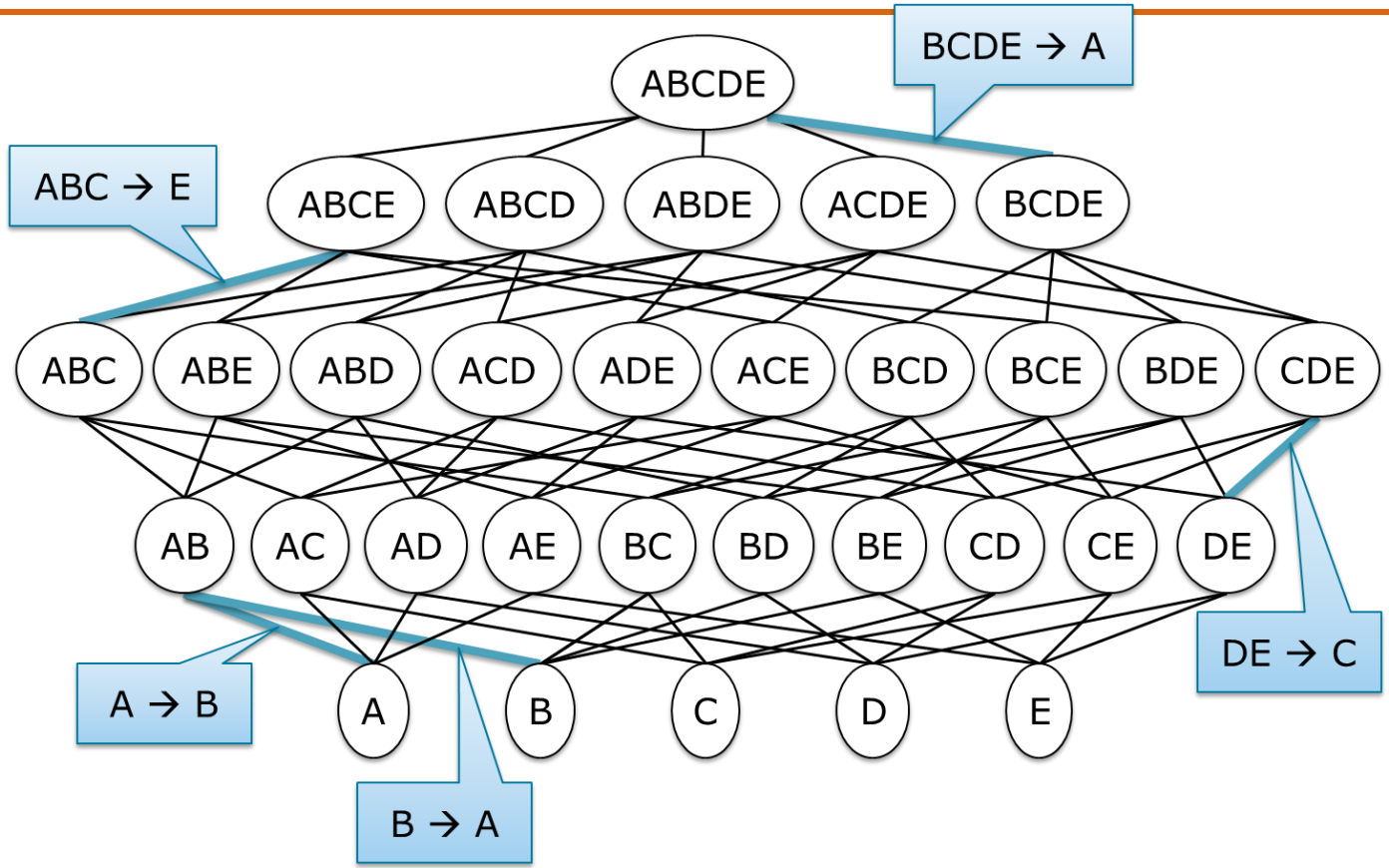
Functional Dependency Discovery

- Motivation
- **Approaches**
  - Lattice Traversal
  - Dependency Induction
  - Difference- and Agree-Set
- fdep
- HyFD
- Evaluation
- Metanome

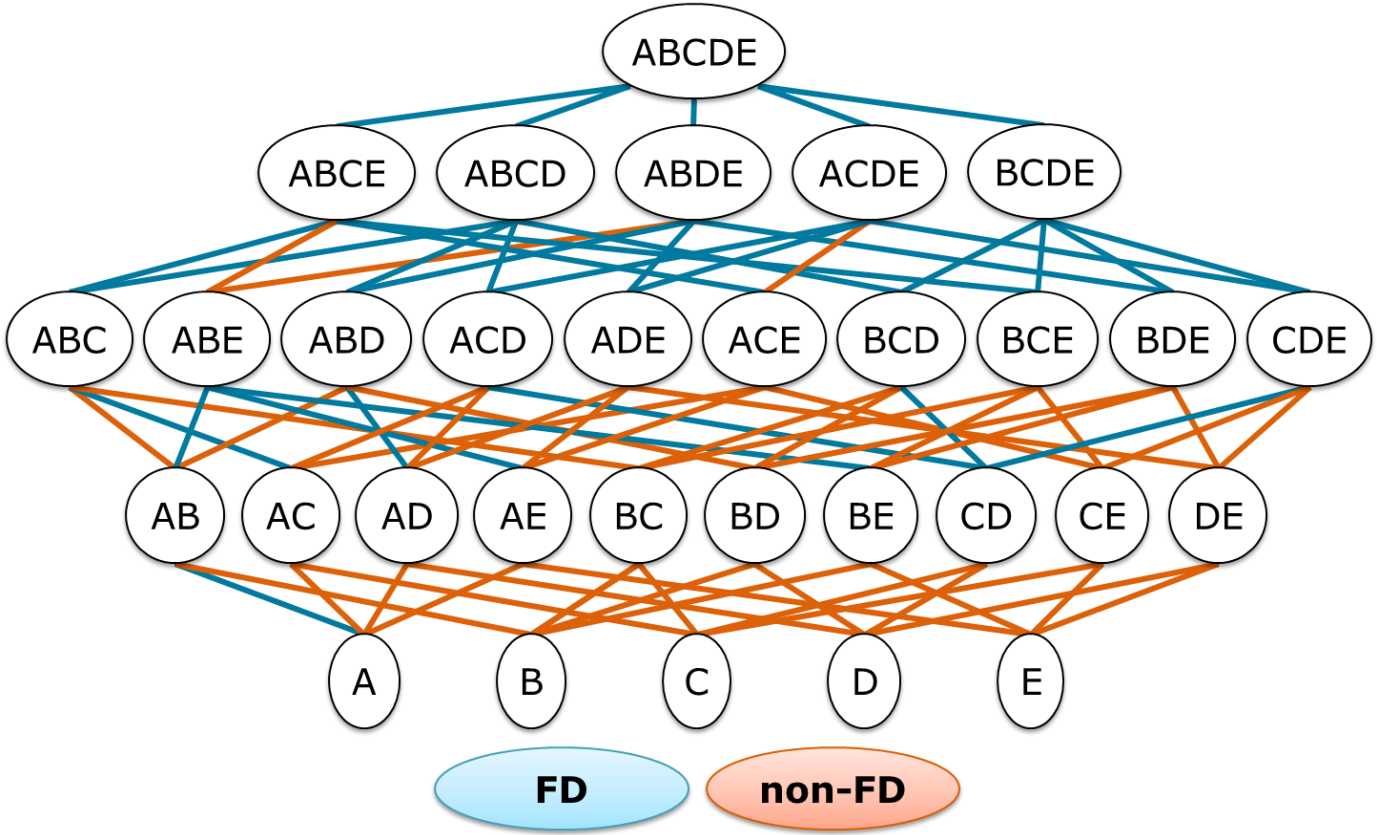
Lattice Traversal



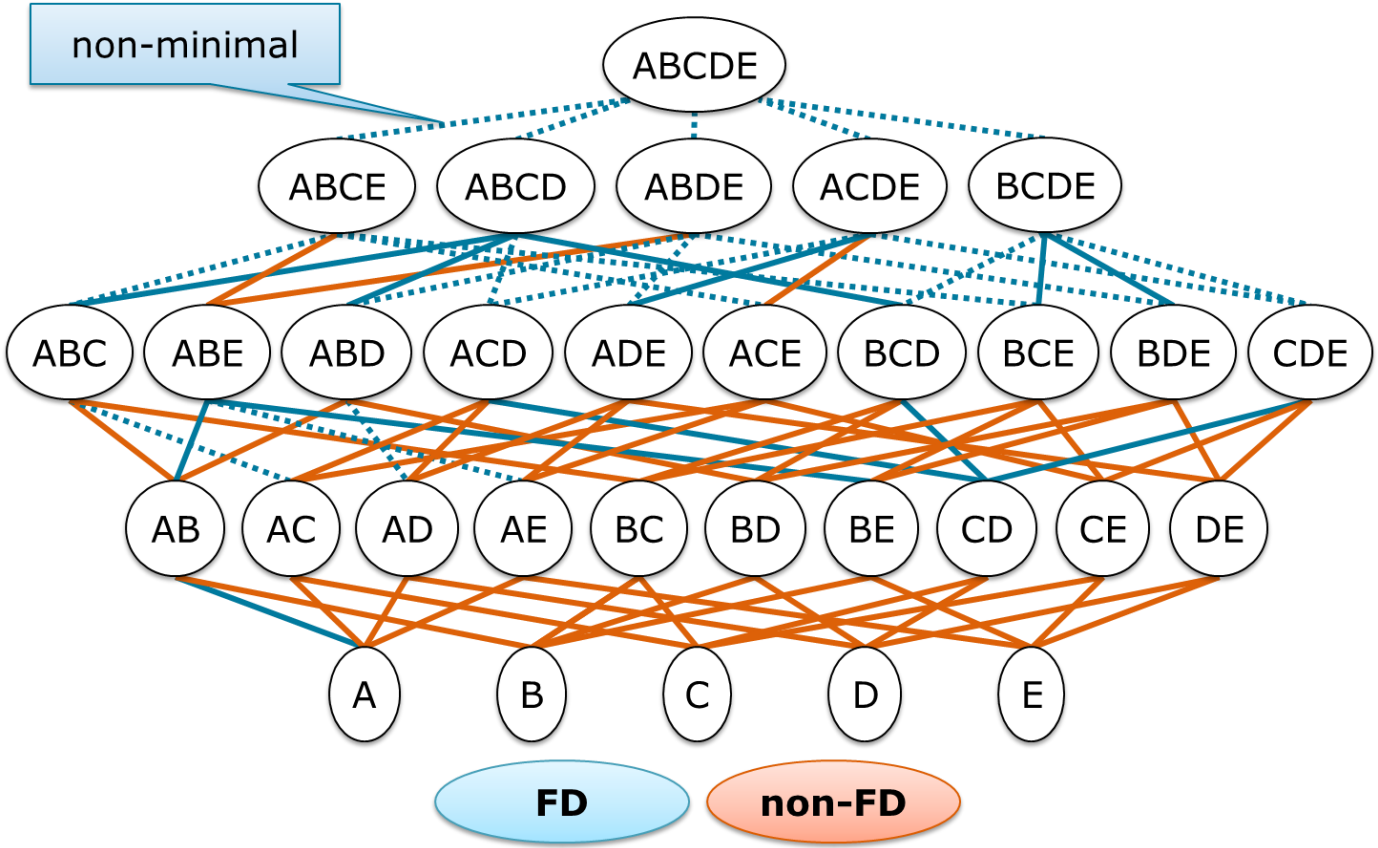
Lattice Traversal



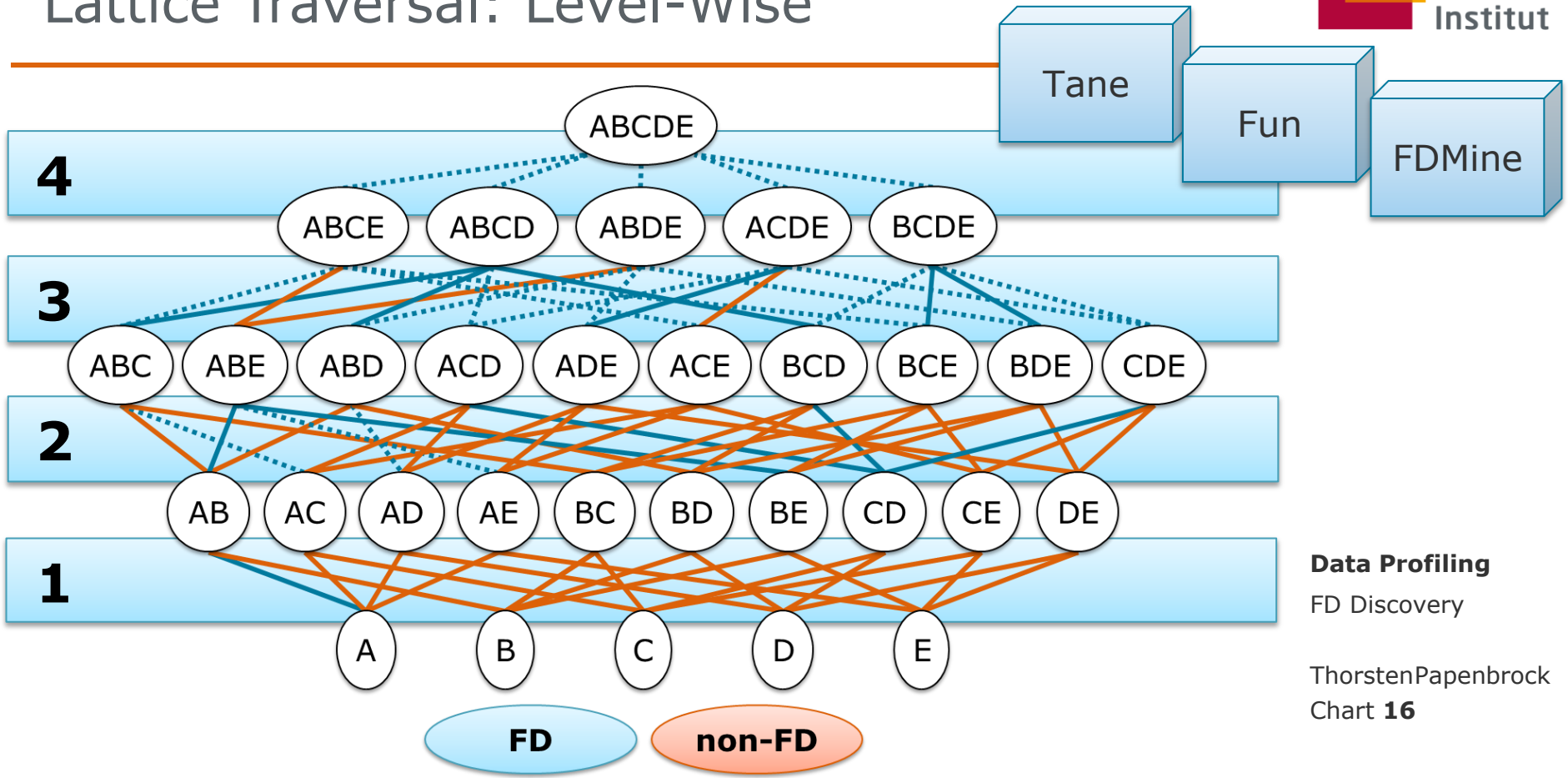
Lattice Traversal



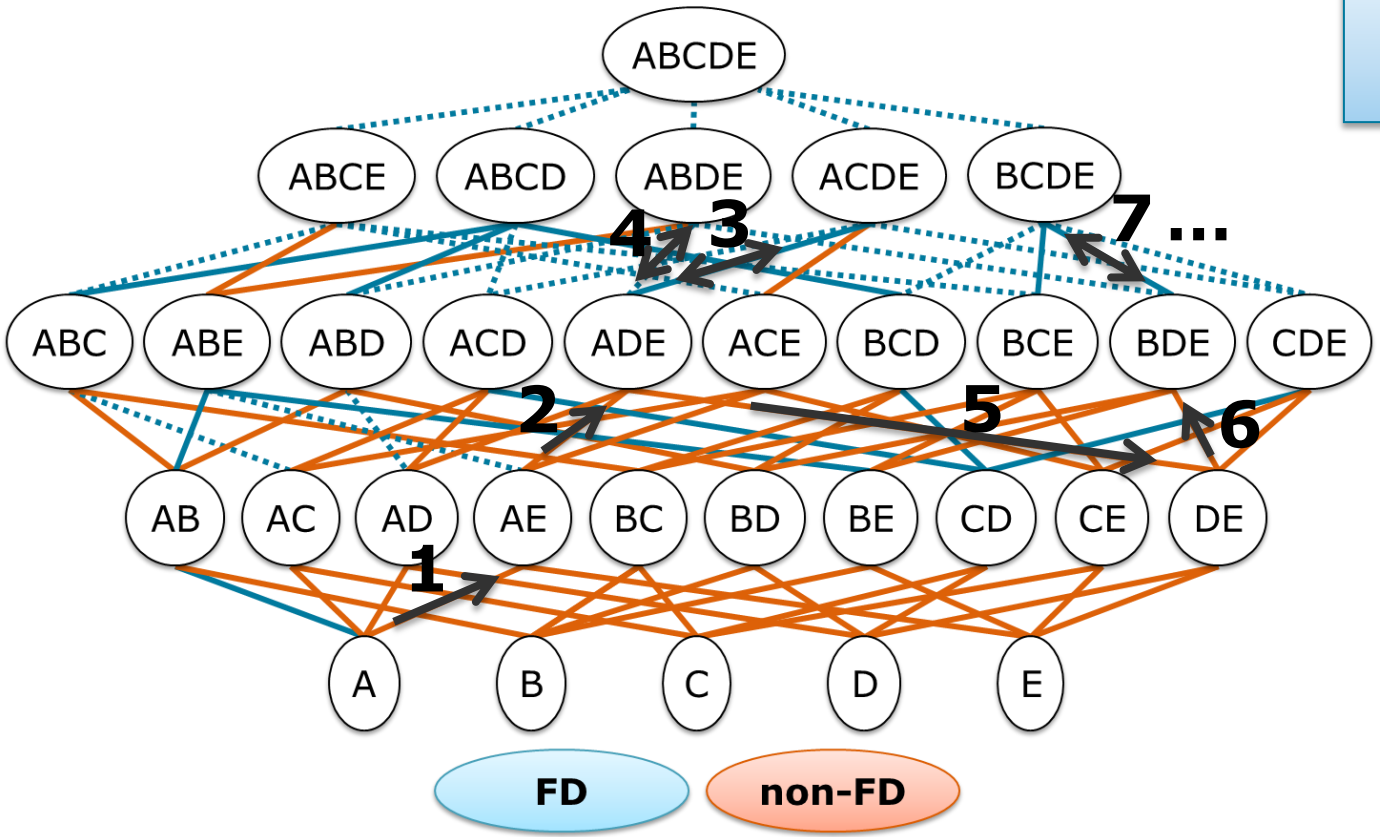
Lattice Traversal



Lattice Traversal: Level-Wise



Lattice Traversal: Depth-First (Random Walk)



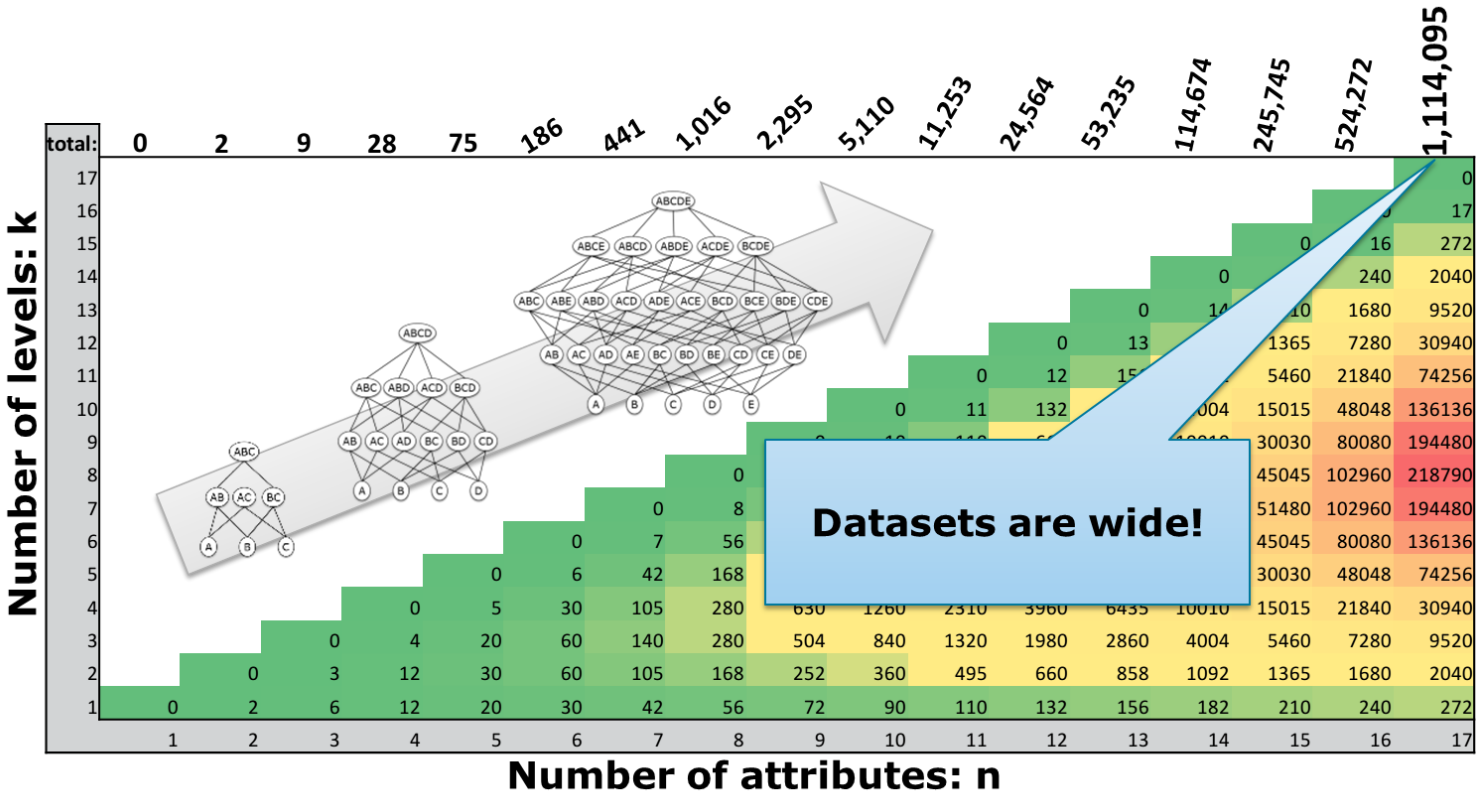
Lattice Traversal: Depth-First (Random Walk)

DFD

Comparison to the level-wise traversal:

- Powerful pruning (up- and downwards) 
- Hole filling and minimization is expensive 
- Unreliable performance due to randomness 

Lattice Traversal

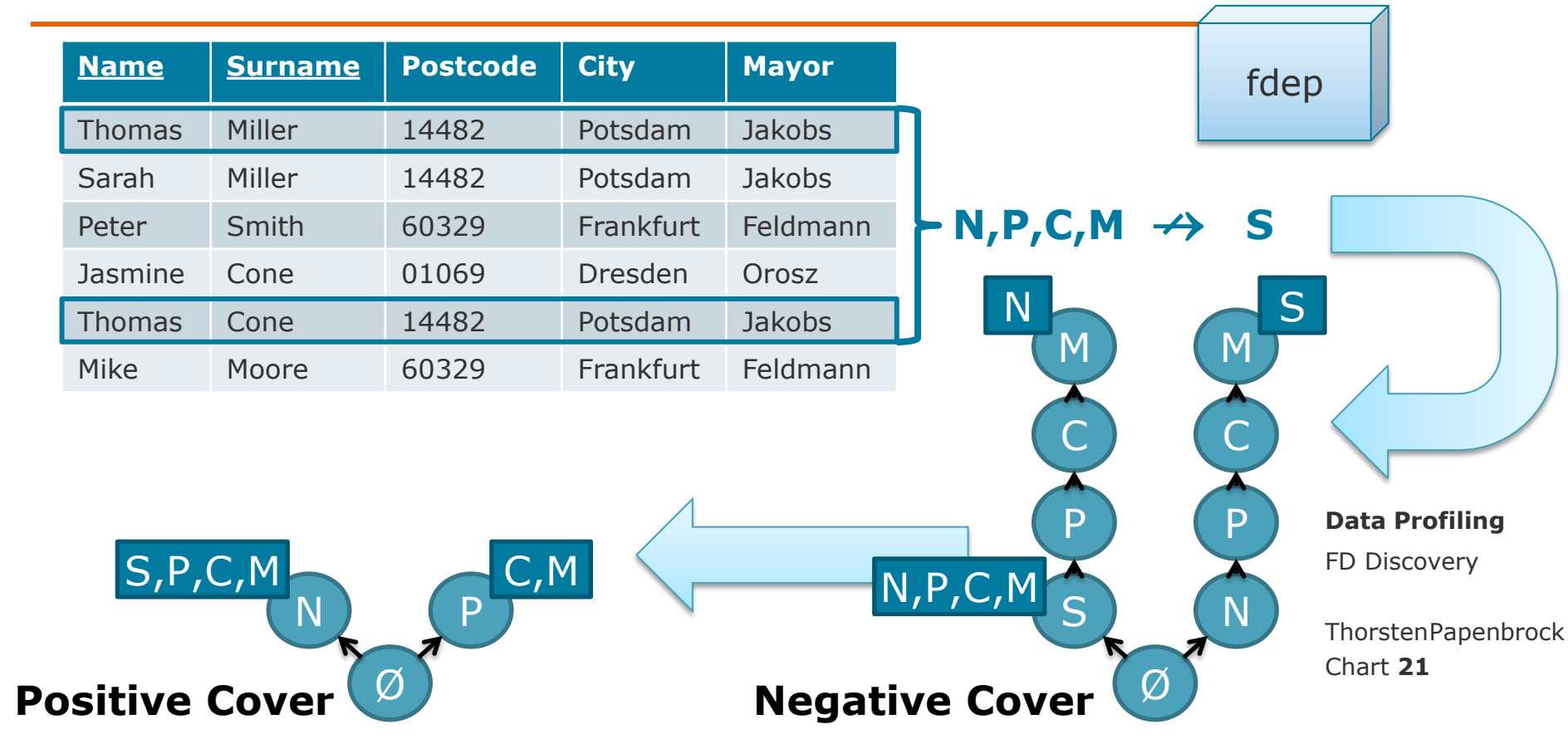


Lattice Traversal

Lattice traversal algorithms: The algorithms TANE [Huhtala et al., 1999], FUN [Novelli and Cicchetti, 2001], FD_MINE [Yao et al., 2002], and DFD [Abedjan et al., 2014c] conceptually arrange all possible FD candidates in a powerset lattice of attribute combinations and then traverse this lattice. The first three algorithms search through the candidate lattice level-wise bottom-up using the *apriori-gen* candidate generation [Agrawal and Srikant, 1994], whereas DFD applies a depth-first random walk. Lattice traversal algorithms in general make intensive use of pruning rules and their candidate validation is based on position list indixes (see Section 2.4). They have been shown to perform well on long datasets, i.e., datasets with many records, but due to their candidate-driven search strategy, they scale poorly with the number of columns in the input dataset.

Dependency Induction

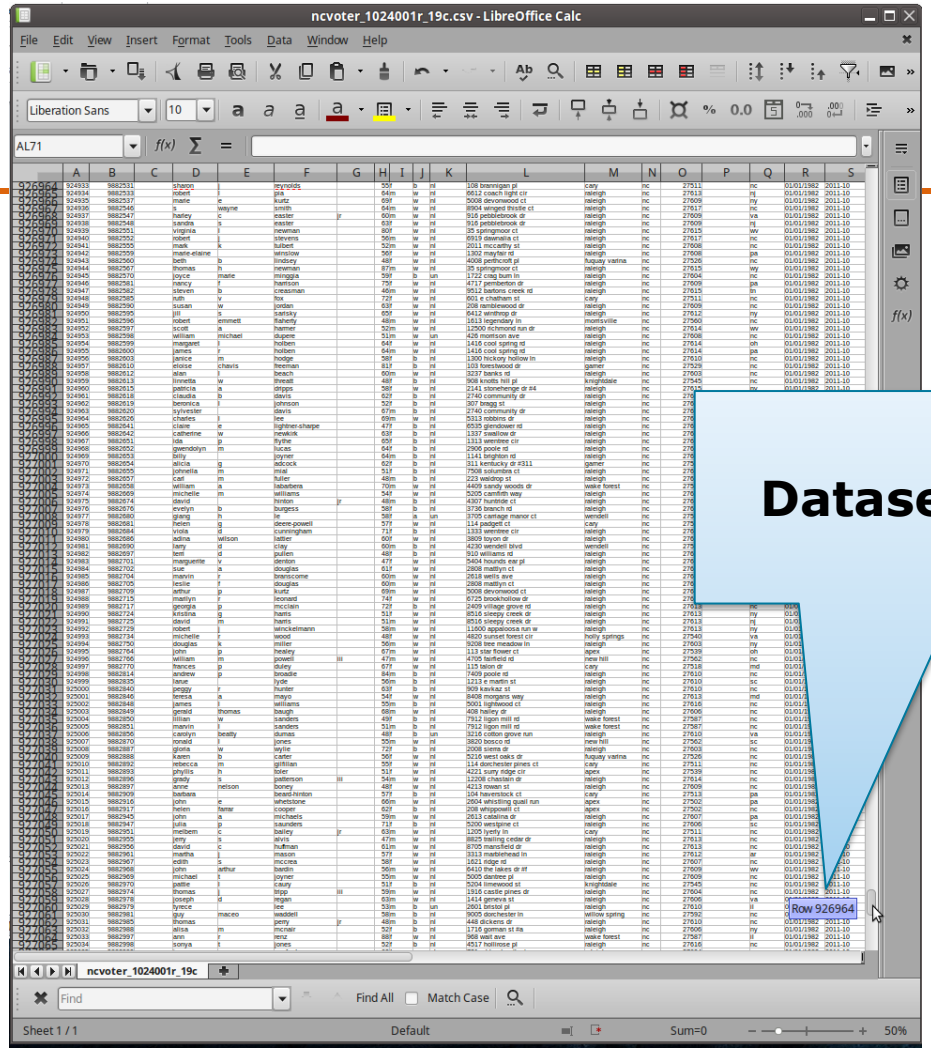
Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann



Data Profiling

FD Discovery

Dependency



Data Profiling

FD Discovery

Dependency Induction

Dependency induction algorithms: The FDEP [Flach and Savnik, 1999] algorithm also compares all records pair-wise to find all *invalid* functional dependencies. This set is called *negative cover* and is stored in a prefix tree. In contrast to DEP-MINER and FASTFDS, FDEP translates this negative cover into the set of valid functional dependencies, i.e., the *positive cover*, not by forming complements but by successive specialization: The positive cover initially assumes that each attribute functionally determines all other attributes; these functional dependencies are then refined with every single non-FD in the negative cover. Apart from the fact that the pair-wise comparisons do not scale with the number of records in the input dataset, this discovery strategy has proven to scale well with the number of attributes.

Difference- and Agree-Set

DepMiner

FastFDs

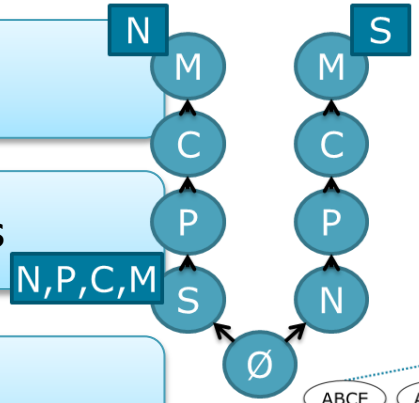
Derive Stripped Partitions

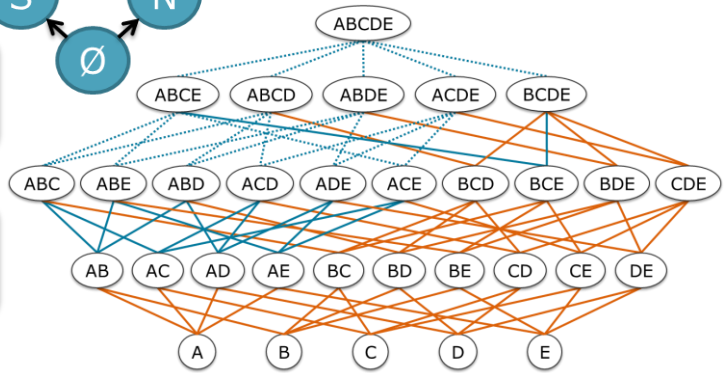
Calculate Agree Sets

Compute Maximal Sets

Derive Complements

Calculate Minimal FDs





Difference- and Agree-Set

Difference- and agree-set algorithms: The algorithms DEP-MINER [Lopes et al., 2000] and FASTFDS [Wyss et al., 2001] analyze a dataset for sets of attributes that agree on the values in certain tuple pairs. These so-called agree-sets are transformed into difference-sets from which all valid FDs can be derived. This discovery strategy scales better with the number of attributes than lattice traversal strategies, because FD candidates are generated only from concrete observations rather than being generated systematically. The required maximization of agree-sets or minimization of difference-sets respectively, however, reduces this advantage significantly. Furthermore, DEP-MINER and FASTFDS scale much worse than the previous algorithms with the number of records, because they need to compare all pairs of records.

Functional Dependency Discovery

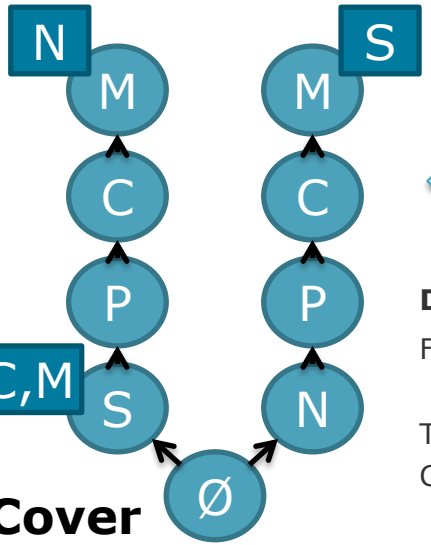
- Motivation
- Approaches
- **fdep**

  Peter A Flach and Iztok Savnik. Database dependency discovery: a machine learning approach. *AI Communications*, 12(3):139–160, 1999.
- HyFD
- Evaluation
- Metanome

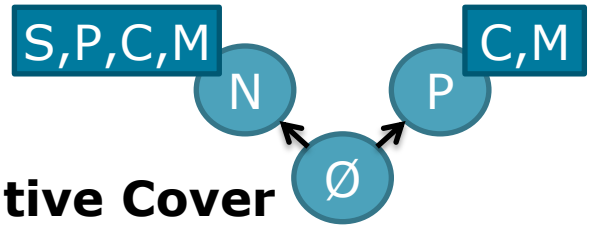
Algorithm Overview

Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann

N,P,C,M ↛ S



N,P,C,M



Positive Cover

Negative Cover

Data Profiling

FD Discovery

Example

Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann

- Surname, Postcode, City, Mayor ↛ Name
- Name, Postcode, City, Mayor ↛ Surname
- Surname ↛ Name, Postcode, City, Mayor

Postcode → City

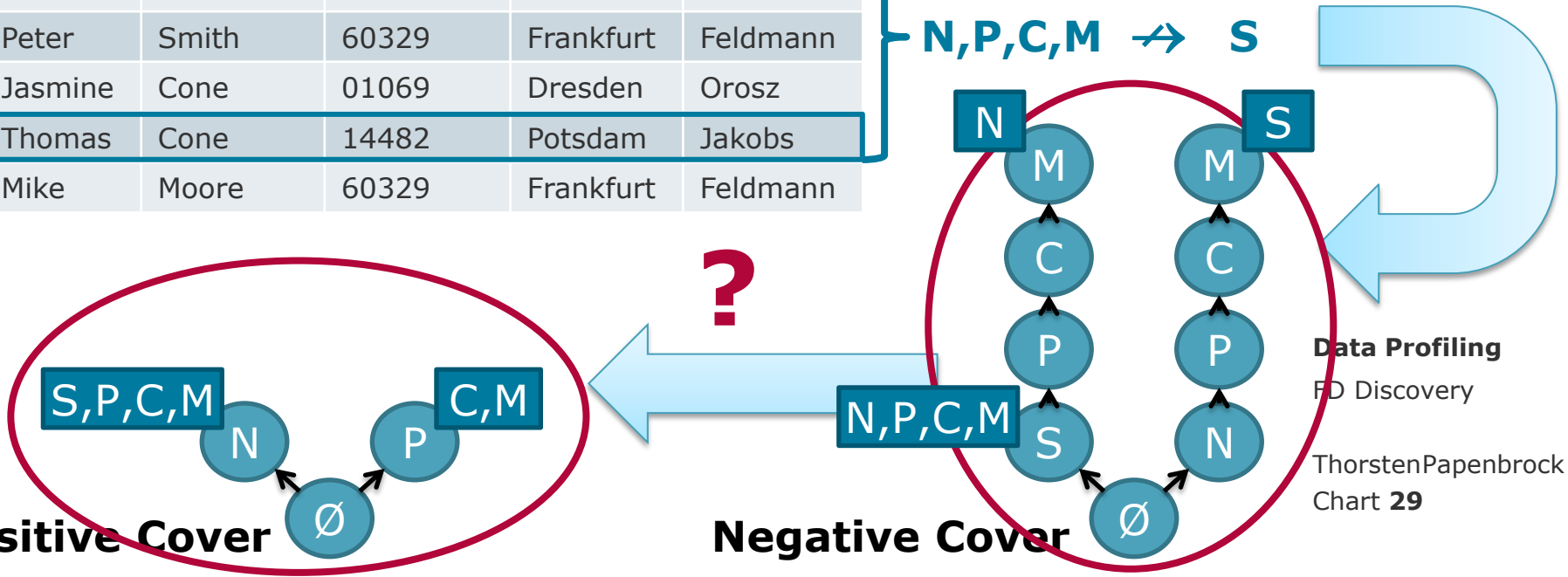
Postcode → Mayor

Name → Surname, …

Algorithm Overview

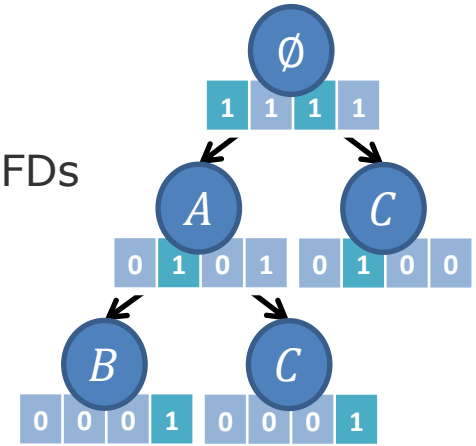
Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann

N,P,C,M ↛ S



FDTree

- Definition: An FDTree is a prefix tree (Trie) with annotations:
  - Nodes represent left-hand-side attributes
  - Paths represent left-hand-sides
  - Annotations (bitsets) represent right-hand-side attributes
  - Marked annotations represent valid FDs
- getGeneralizations(X → Y)
  - Generalization: X′ → Y with X′ ⊆ X
  - Simple depth-first search

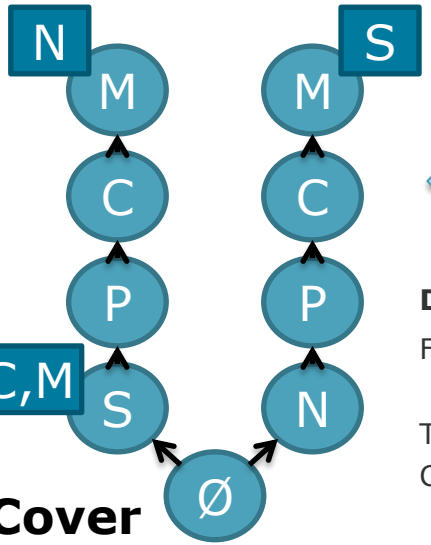


$\emptyset \rightarrow A, C$ $A \rightarrow B$ $C \rightarrow B$
$A, B \rightarrow D$ $A, C \rightarrow D$

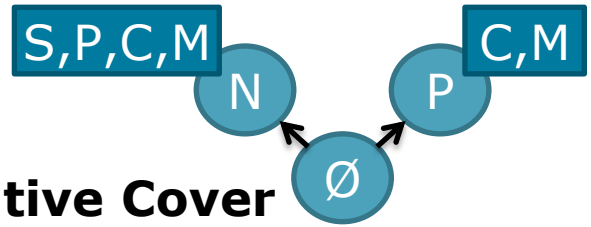
Algorithm Overview

Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann

N,P,C,M ↛ S



?



N,P,C,M

Positive Cover

Negative Cover

Data Profiling

FD Discovery

Cover Inversion

Algorithm 3: Functional Dependency Induction

```
Data: nonFds
Result: fds
1  nonFds ← sort(nonFds, CARDINALITY_DESCENDING);
2  if fds = null then
3      fds ← new FDTree;
4      fds.add(∅ → {0, 1, ..., numAttributes});
5  for lhs ∈ nonFds do
6      rhss ← lhs.clone().flip();
7      for rhs ∈ rhss do
8          specialize(fds, lhs, rhs);
9  return fds;
   function specialize(fds, lhs, rhs)
10 invalidLhss ← fds.getFdAndGenerals(lhs, rhs);
11 for invalidLhs ∈ invalidLhss do
12     fds.remove(invalidLhs, rhs);
13     for attr ∈ [0, numAttributes[ do
14         if invalidLhs.get(attr) ∨
15            rhs = attr then
16             continue;
17         newLhs ← invalidLhs ∪ attr;
18         if fds.findFdOrGeneral(newLhs, rhs) then
19             continue;
20         fds.add(newLhs, rhs);
```

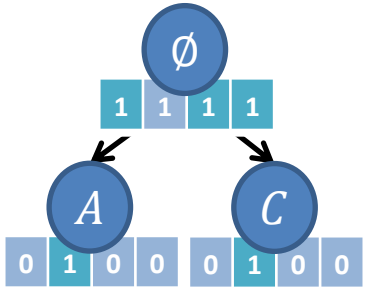
fds:

$\emptyset \rightarrow A, B, C, D$

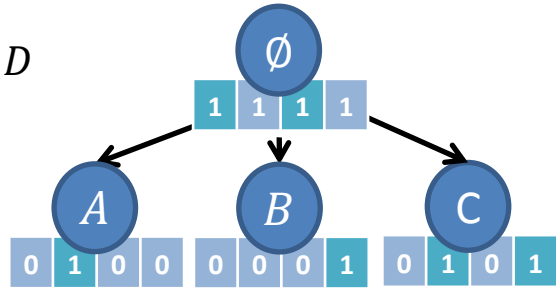
nonFds: $D \nrightarrow B$, $A \nrightarrow D$, $B \nrightarrow D$, $C \nrightarrow D$

specialize():

$D \nrightarrow B$



$A \nrightarrow D$



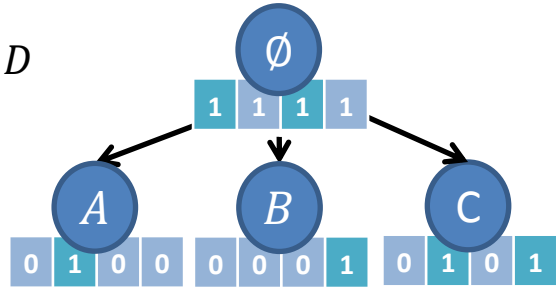
Cover Inversion

Algorithm 3: Functional Dependency Induction

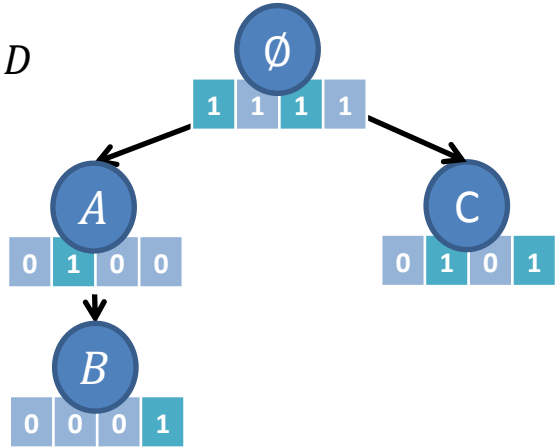
```
Data: nonFds
Result: fds
1  nonFds ← sort(nonFds, CARDINALITY_DESCENDING);
2  if fds = null then
3      fds ← new FDTree;
4      fds.add(∅ → {0, 1, ..., numAttributes});
5  for lhs ∈ nonFds do
6      rhss ← lhs.clone().flip();
7      for rhs ∈ rhss do
8          specialize(fds, lhs, rhs);
9  return fds;

   function specialize(fds, lhs, rhs)
10 invalidLhss ← fds.getFdAndGenerals(lhs, rhs);
11 for invalidLhs ∈ invalidLhss do
12     fds.remove(invalidLhs, rhs);
13     for attr ∈ [0, numAttributes[ do
14         if invalidLhs.get(attr) ∨
15            rhs = attr then
16             continue;
17         newLhs ← invalidLhs ∪ attr;
18         if fds.findFdOrGeneral(newLhs, rhs) then
19             continue;
20         fds.add(newLhs, rhs);
```

nonFds: $D \nrightarrow B$, $A \nrightarrow D$, $B \nrightarrow D$, $C \nrightarrow D$

$A \nrightarrow D$



$B \nrightarrow D$



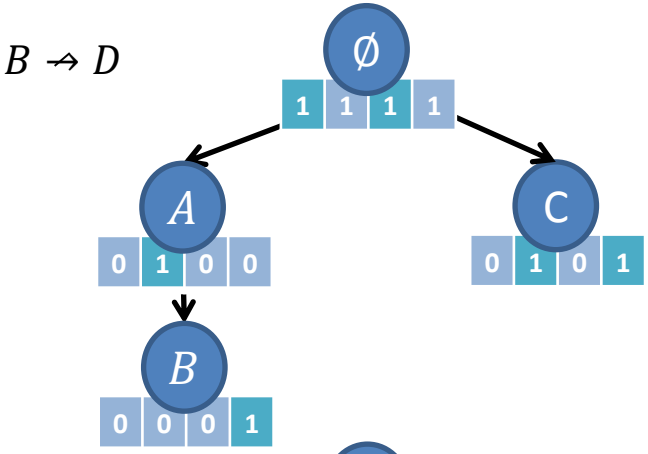
Cover Inversion

Algorithm 3: Functional Dependency Induction

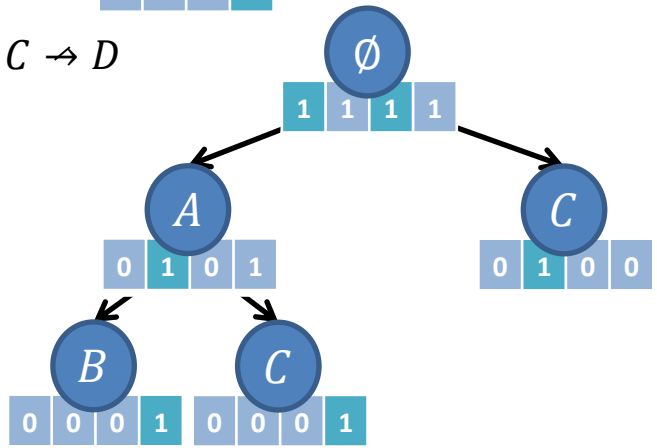
```
Data: nonFds
Result: fds
1  nonFds ← sort(nonFds, CARDINALITY_DESCENDING);
2  if fds = null then
3      fds ← new FDTree;
4      fds.add(∅ → {0, 1, ..., numAttributes});
5  for lhs ∈ nonFds do
6      rhss ← lhs.clone().flip();
7      for rhs ∈ rhss do
8          specialize(fds, lhs, rhs);
9  return fds;

   function specialize(fds, lhs, rhs)
10 invalidLhss ← fds.getFdAndGenerals(lhs, rhs);
11 for invalidLhs ∈ invalidLhss do
12     fds.remove(invalidLhs, rhs);
13     for attr ∈ [0, numAttributes[ do
14         if invalidLhs.get(attr) ∨
15            rhs = attr then
16             continue;
17         newLhs ← invalidLhs ∪ attr;
18         if fds.findFdOrGeneral(newLhs, rhs) then
19             continue;
20         fds.add(newLhs, rhs);
```

nonFds: $D \nrightarrow B$, $A \nrightarrow D$, $B \nrightarrow D$, $C \nrightarrow D$

$B \nrightarrow D$



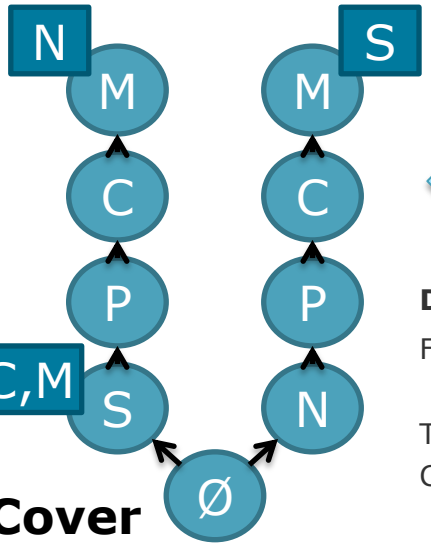
$C \nrightarrow D$



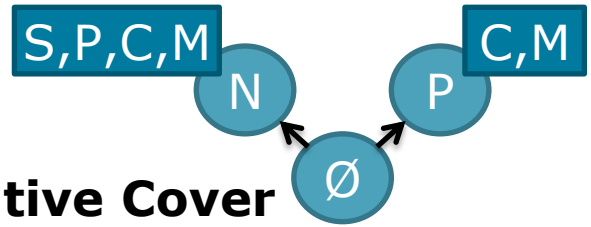
Algorithm Overview

Name	Surname	Postcode	City	Mayor
Thomas	Miller	14482	Potsdam	Jakobs
Sarah	Miller	14482	Potsdam	Jakobs
Peter	Smith	60329	Frankfurt	Feldmann
Jasmine	Cone	01069	Dresden	Orosz
Thomas	Cone	14482	Potsdam	Jakobs
Mike	Moore	60329	Frankfurt	Feldmann

N,P,C,M ↛ S



N,P,C,M



Positive Cover

Negative Cover

Data Profiling

FD Discovery

Functional Dependency Discovery

- Motivation
- Approaches
- fdep

- **HyFD**

Thorsten Papenbrock and Felix Naumann. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 821–833, 2016.

- Evaluation
- Metanome

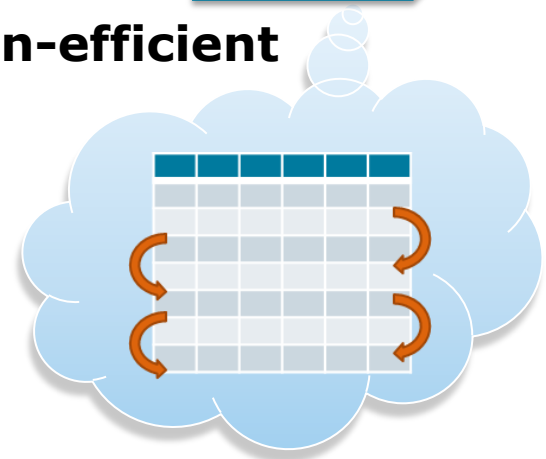
Going Hybrid

FD Discovery Algorithms

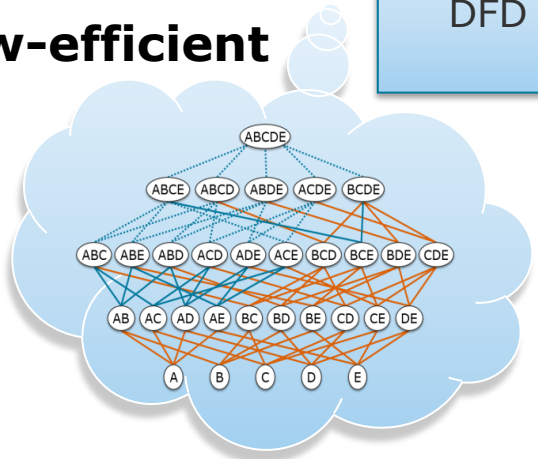
[Flach1999]
Fdep
[Lopes2000]
DepMiner
[Wyss2001]
FastFDs

[Huhtala1999]
Tane
[Novelli2001]
Fun
[Yao2008]
FDMine
[Abedjan2014]
DFD

column-efficient



row-efficient

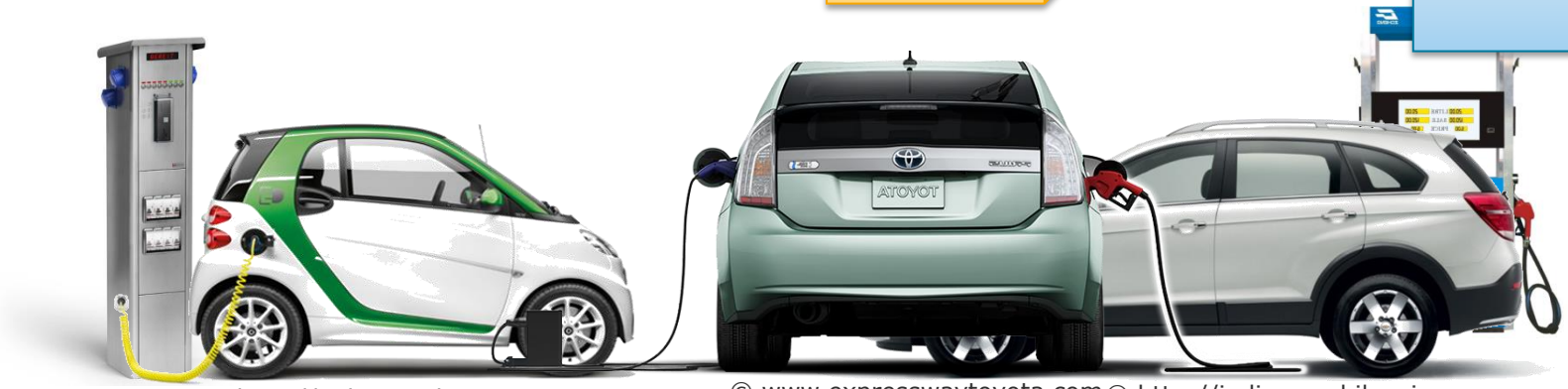


Data Profiling
FD Discovery

Going Hybrid

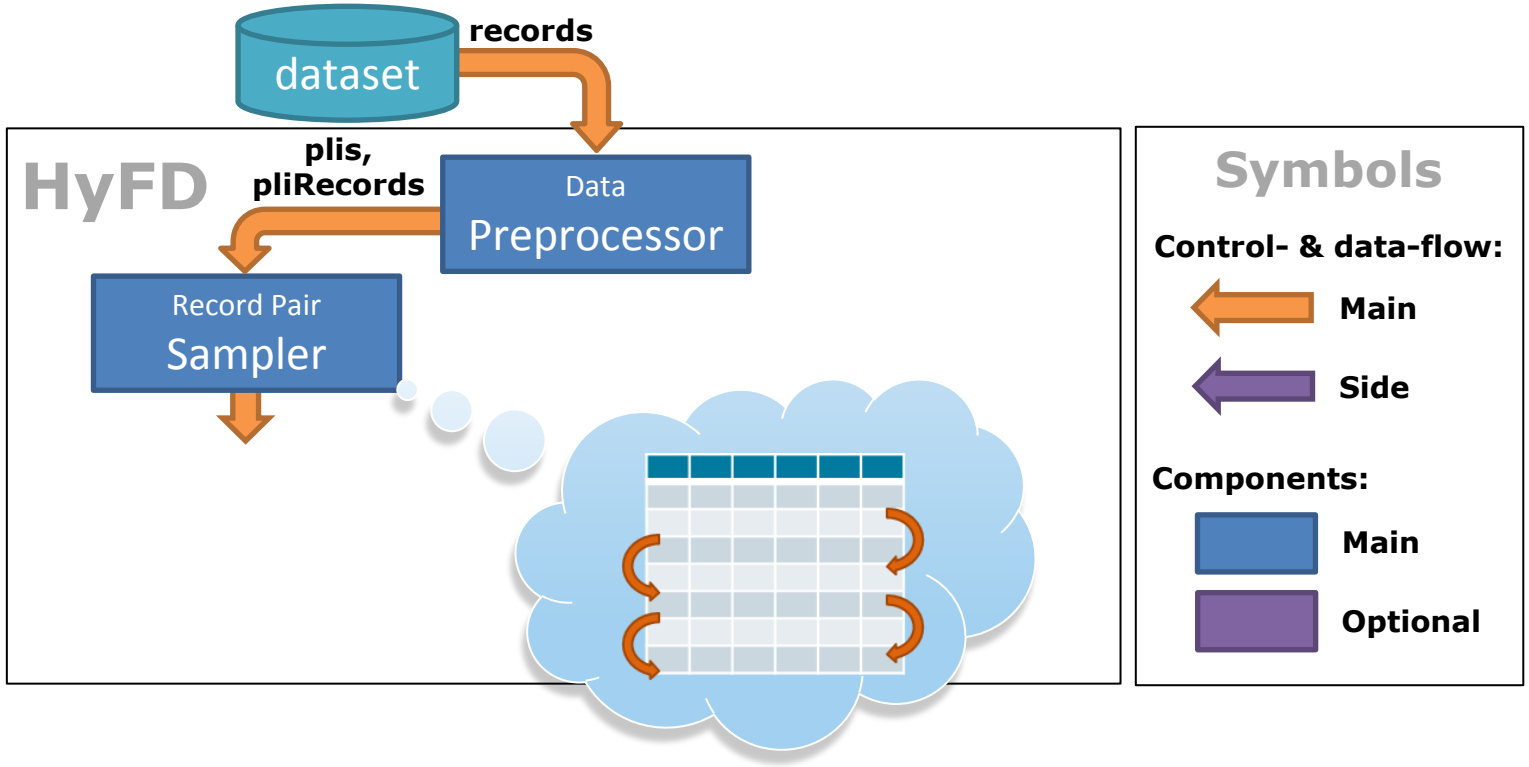
FD Discovery Algorithms

[Flach1999]
Fdep
[Lopes2000]
DepMiner
[Wyss2001]
FastFDs
HyFD
[Huhtala1999]
Tane
[Novelli2001]
Fun
[Yao2008]
FDMine
DFD
[Abedjan2014]

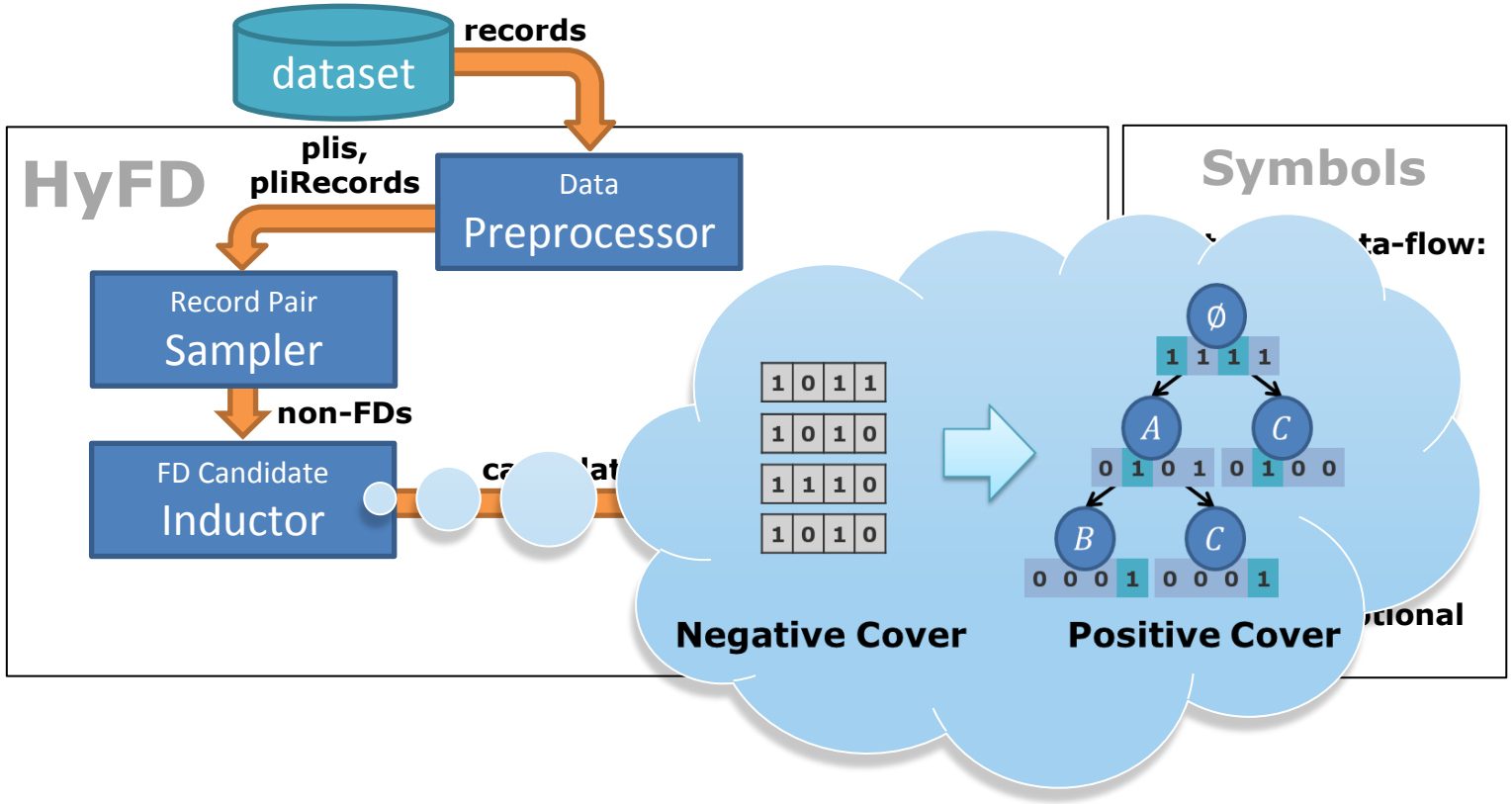


© http://4electric.de © www.expresswaytoyota.com © http://indiancarsbikes.in

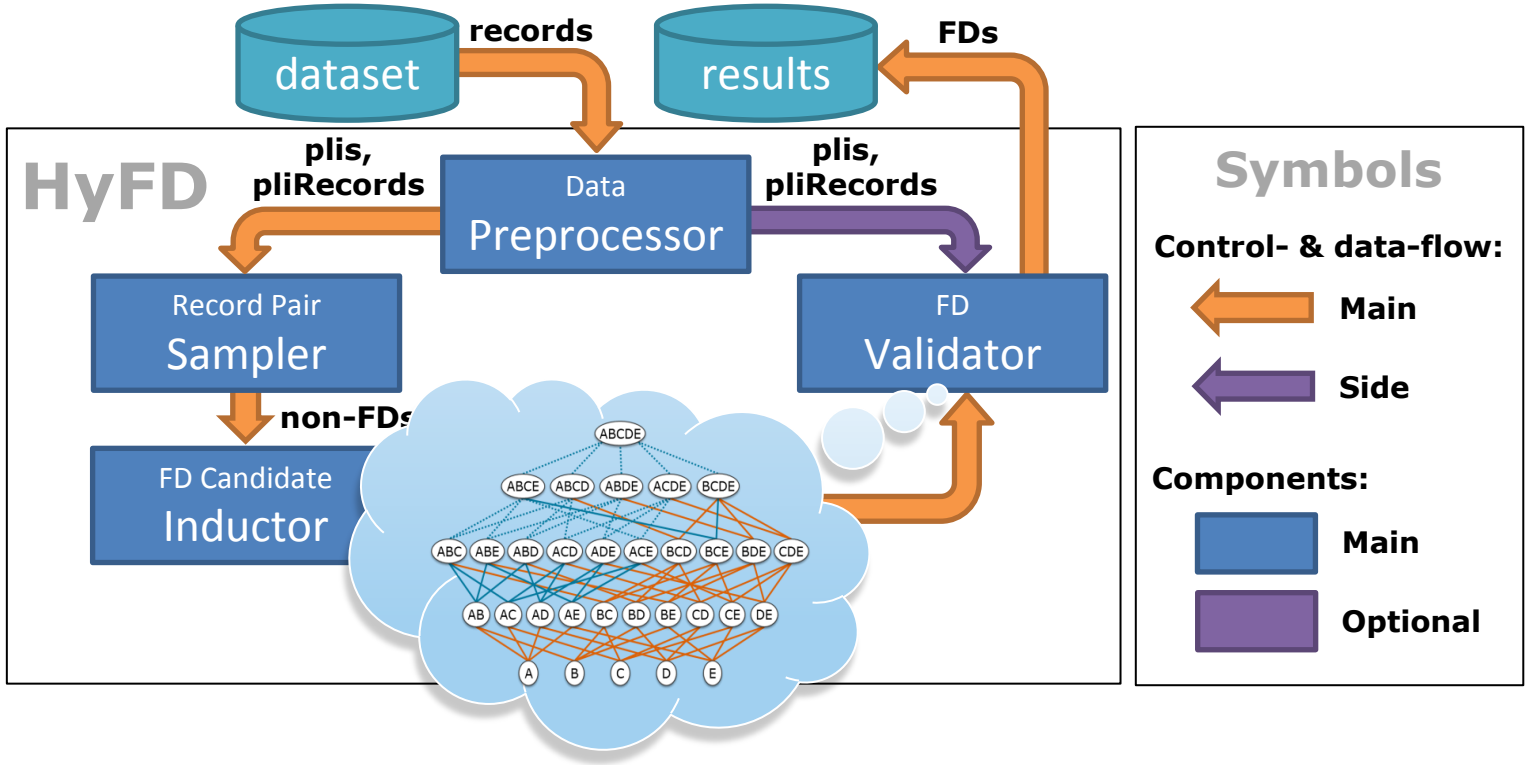
Hybrid FD Discovery



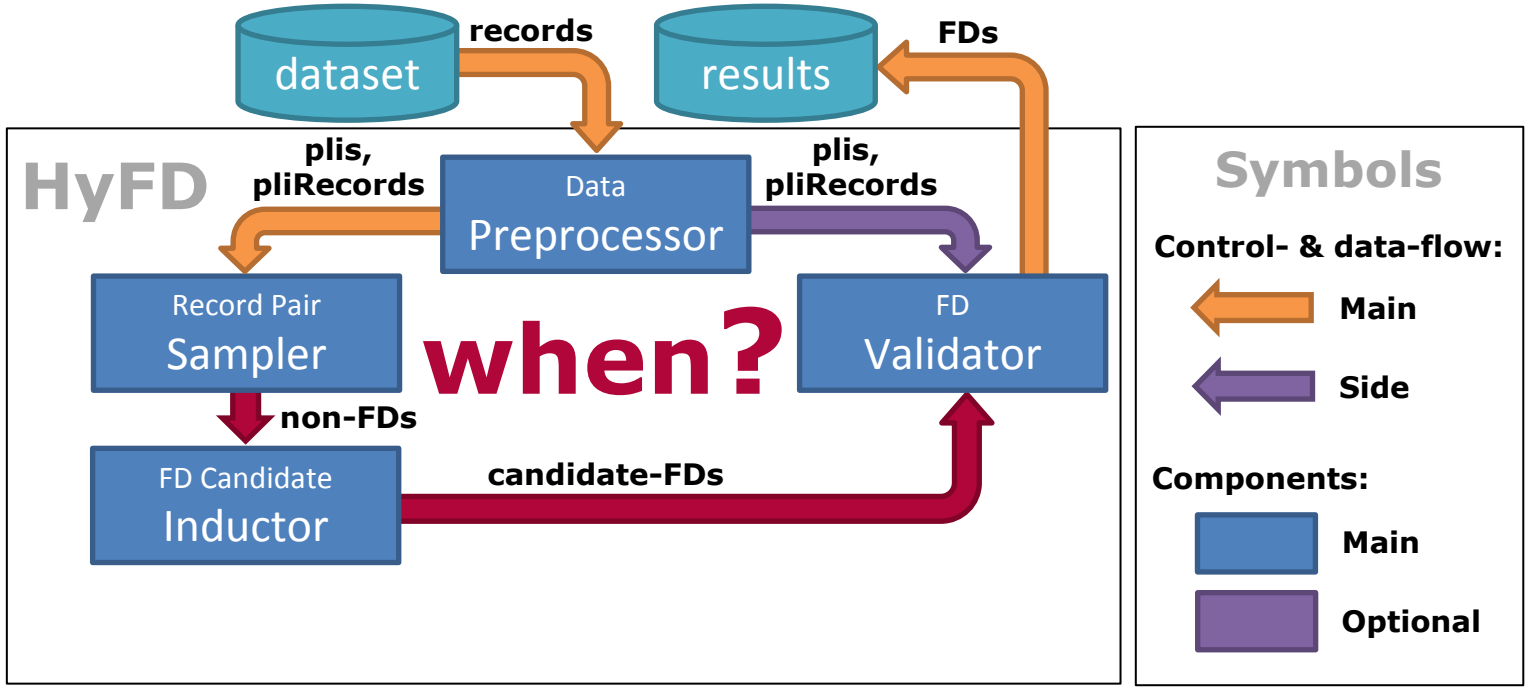
Hybrid FD Discovery



Hybrid FD Discovery



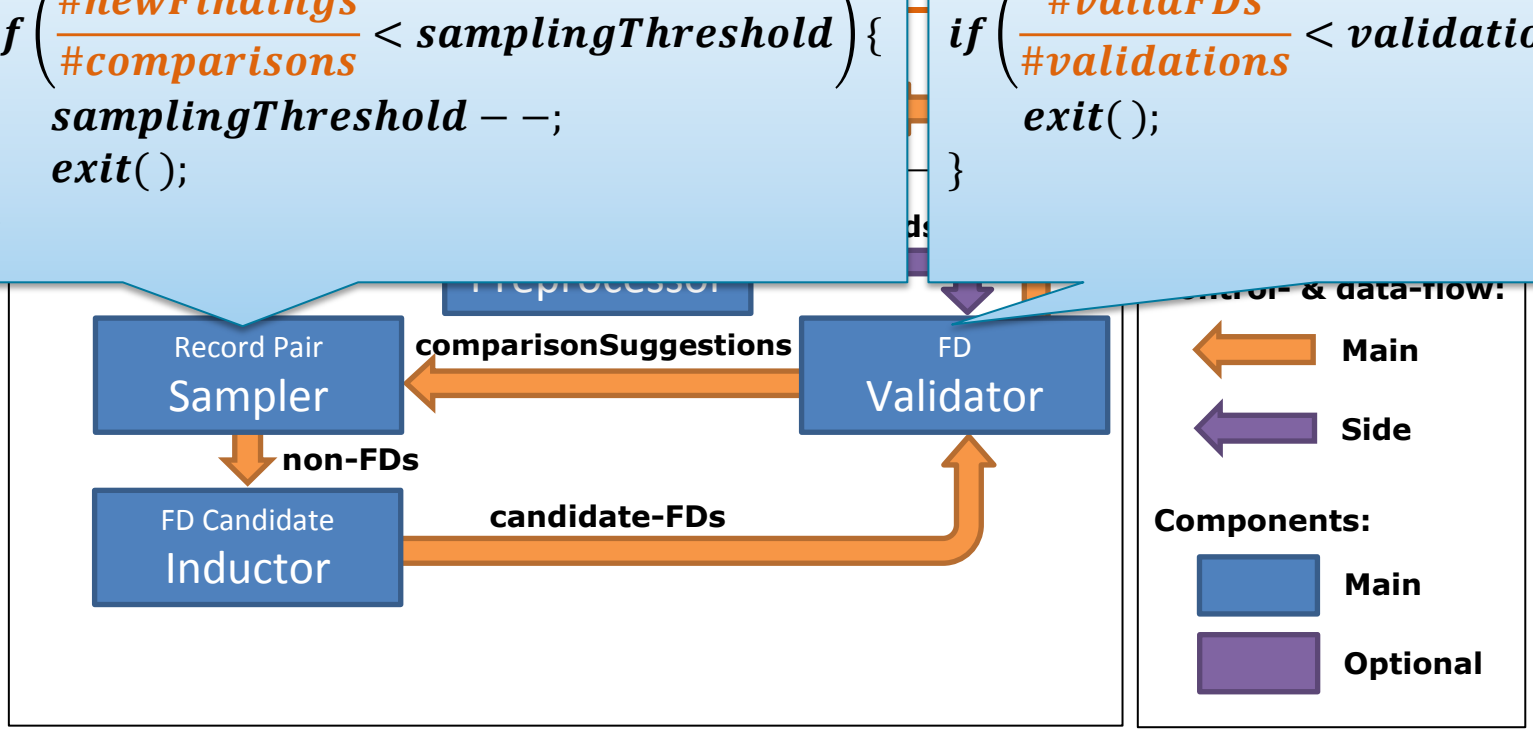
Hybrid FD Discovery



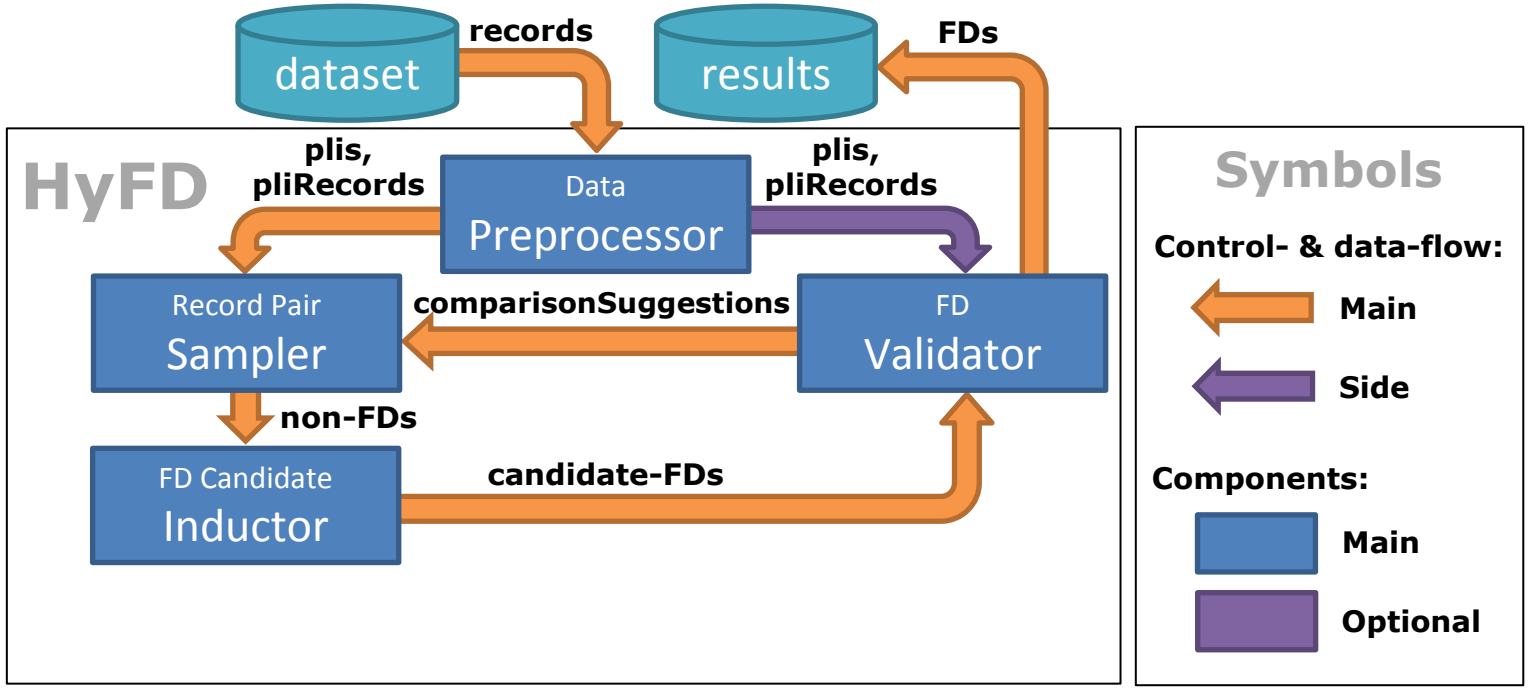
Hybrid FD Discovery

$$if\left(\frac{\#newFindings}{\#comparisons} < samplingThreshold\right)\{$$
$$samplingThreshold--;$$
$$exit();$$
$$\}$$

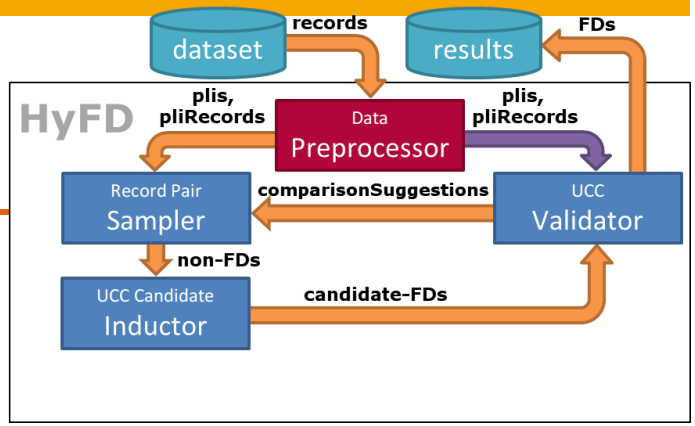
$$if\left(\frac{\#validFDs}{\#validations} < validationThreshold\right)\{$$
$$exit();$$
$$\}$$



Hybrid FD Discovery



Hybrid FD Discovery



position list indexes (**plis**)

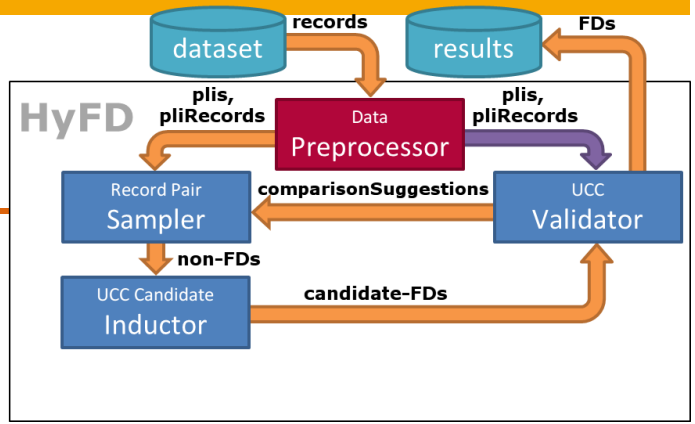
	0	1	2
0	2, 5, 8	1, 7	0, 7, 8
1	0, 4	2, 3, 8, 9	1, 2, 3, 4, 5, 6
2	1, 3, 6, 7	0, 5, 6	

attributeID → clusterID → recordID

Implementation example:
List<List<Integer>> plis;

Data Profiling
FD Discovery

Hybrid FD Discovery



inverted position list indexes (**invertedPlis**)

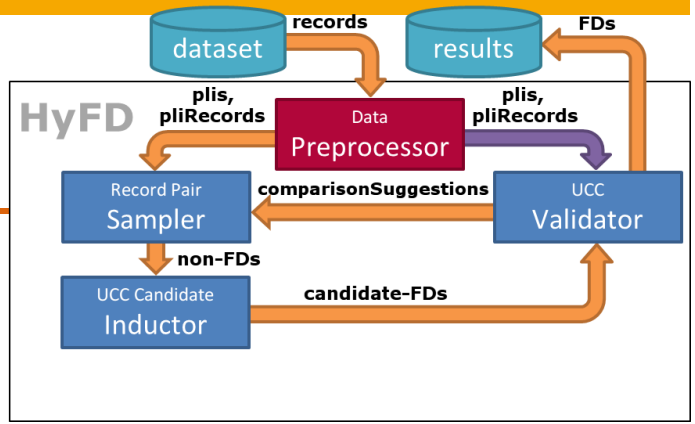
	0		1		2
0	1	0	2	0	0
1	2	1	0	1	1
2	0	2	1	2	1
3	2	3	1	3	1
4	1	4	-	4	1
5	0	5	2	5	1

attributeID → recordID → clusterID

Implementation example:

List<Integer[]> invertedPlis;

Hybrid FD Discovery



position list index records (**pliRecords**)

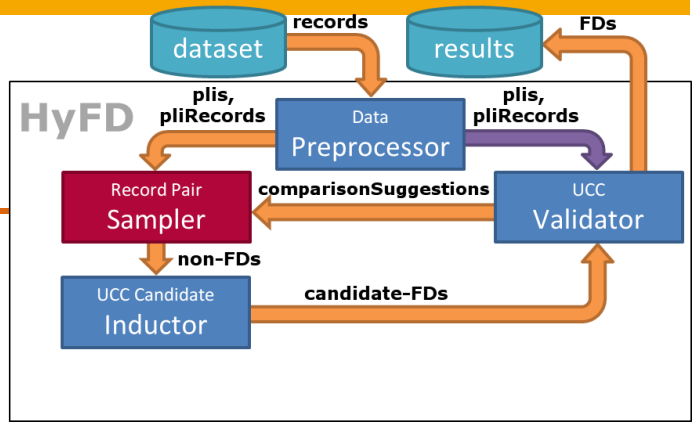
	0	1	2
0	1	2	0
1	2	0	1
2	0	1	1
3	2	1	1
4	1	-	1
5	0	2	1

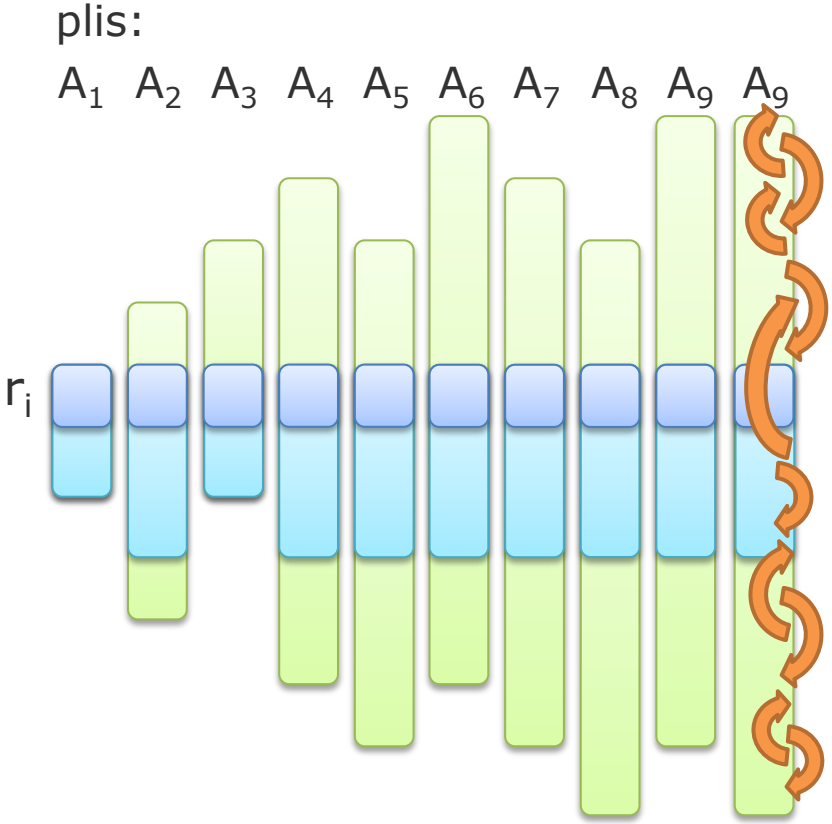
recordID → attributeID → clusterID

Implementation example:
Integer[][] pliRecords;

"Dictionary encoded records using the plis"

Hybrid FD Discovery



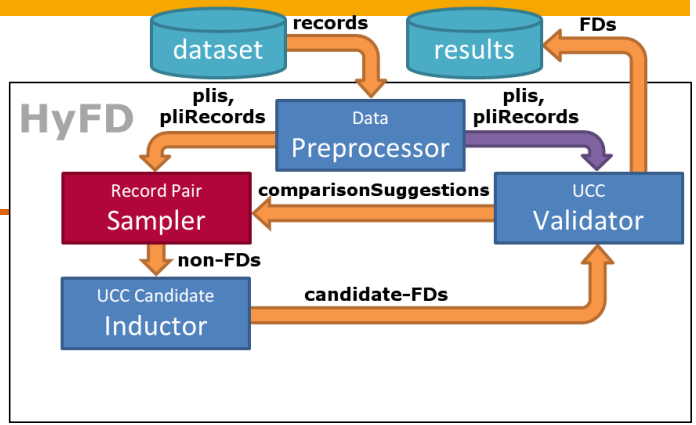


DO NOT COMPARE EVERYTHING!

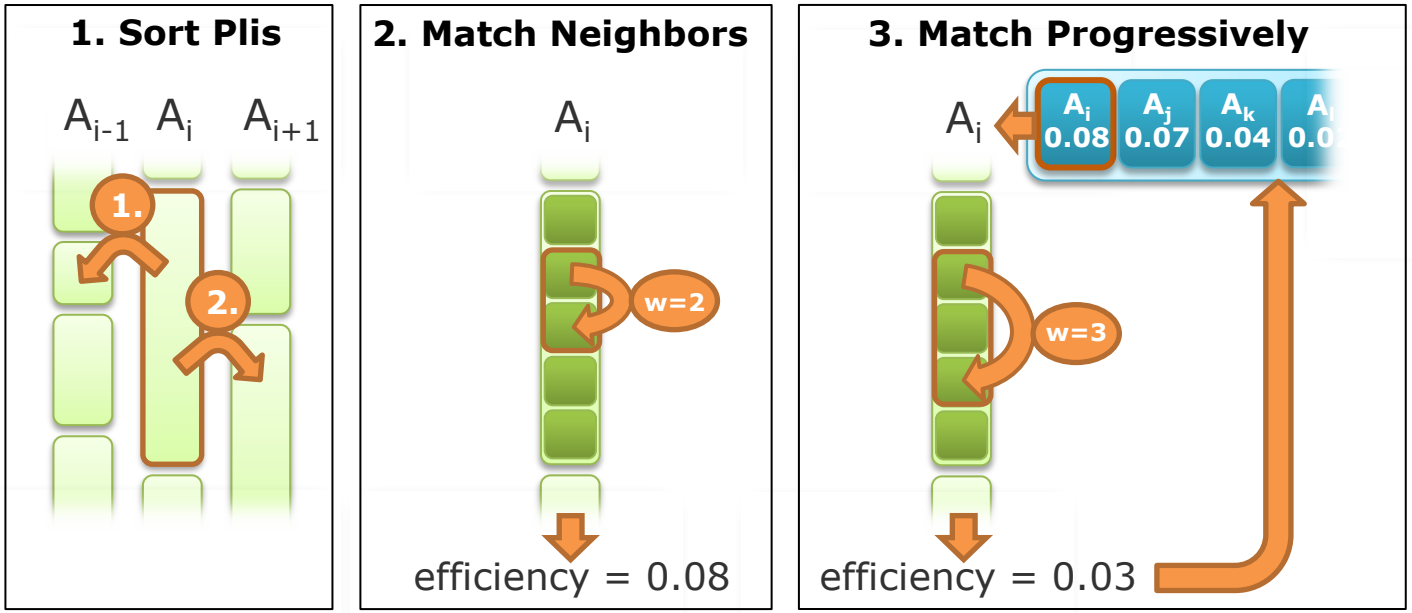
- Focused sampling:
  - Only compare records that share at least one cluster (otherwise they cannot violate any FD).
  - Try to avoid duplicate comparisons.
  - Records that share many clusters prune more effectively.

HyFD

Hybrid FD Discovery



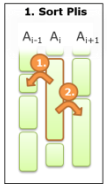
Progressive, focused sampling:

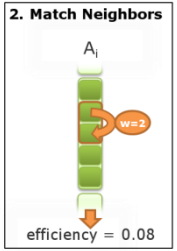


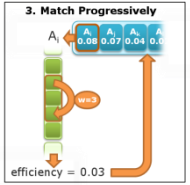
Algorithm 2: Record Pair Sampling

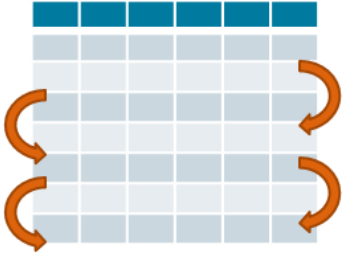
Data: *plis*, *pliRecords*, *comparisonSuggestions*
Result: *nonFds*

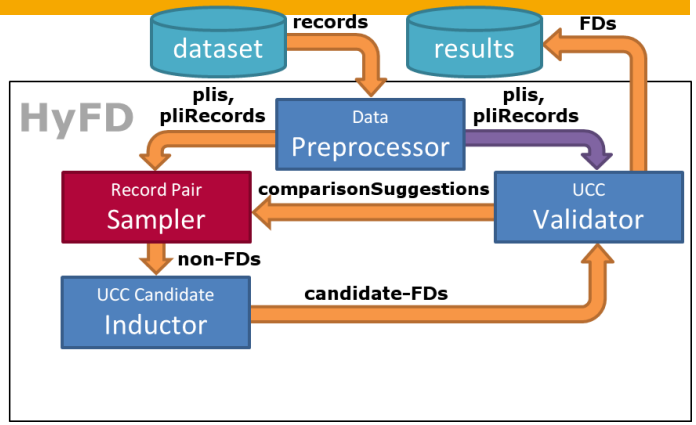
```
1  if efficiencyQueue = ∅ then
2      for pli ∈ plis do
3          for cluster ∈ pli do
4              cluster ← sort(cluster, ATTR_LEFT_RIGHT);
5      nonFds ← ∅;
6      efficiencyThreshold ← 0.01;
7      efficiencyQueue ← new PriorityQueue;
8      for attr ∈ [0, numAttributes[ do
9          efficiency ← new Efficiency;
10         efficiency.attribute ← attr;
11         efficiency.window ← 2;
12         efficiency.comps ← 0;
13         efficiency.results ← 0;
14         runWindow(efficiency, plis[attr], nonFds);
15         efficiencyQueue.append(efficiency);
16 else
17     efficiencyThreshold ← efficiencyThreshold / 2;
18     for sug ∈ comparisonSuggestions do
19         nonFds ← nonFds ∪ match(sug[0], sug[1]);
20 while true do
21     bestEff ← efficiencyQueue.peek();
22     if bestEff.eval() < efficiencyThreshold then
23         break;
24     bestEff.window ← bestEff.window + 1;
25     runWindow(bestEff, plis[bestEff.attribute], nonFds);
26 return newFDsIn(nonFds);
   function runWindow(efficiency, pli, nonFds)
27 prevNumNonFds ← |nonFds|;
28 for cluster ∈ pli do
29     for i ∈ [0, |cluster| − efficiency.window [ do
30         pivot ← pliRecords[cluster[i]];
31         partner ← pliRecords[cluster[i + window − 1]];
32         nonFds ← nonFds ∪ match(pivot, partner);
33         efficiency.comps ← efficiency.comps + 1;
34 newResults ← |nonFds| − prevNumNonFds;
35 efficiency.results ← efficiency.results + newResults;
```











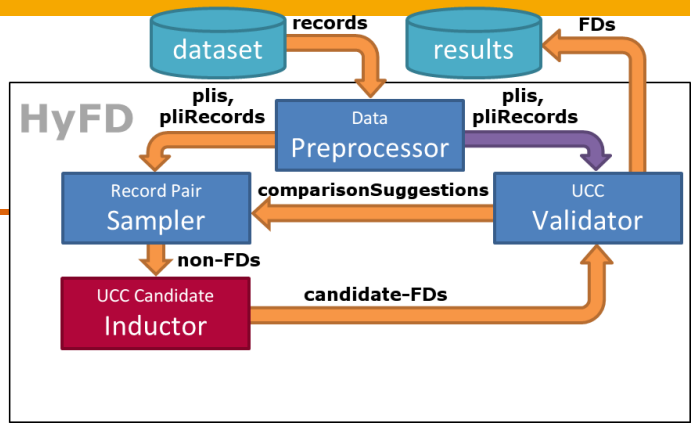
Hasso Plattner Institut

Data Profiling

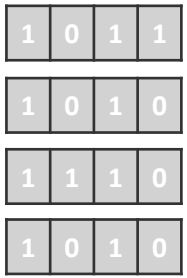
FD Discovery

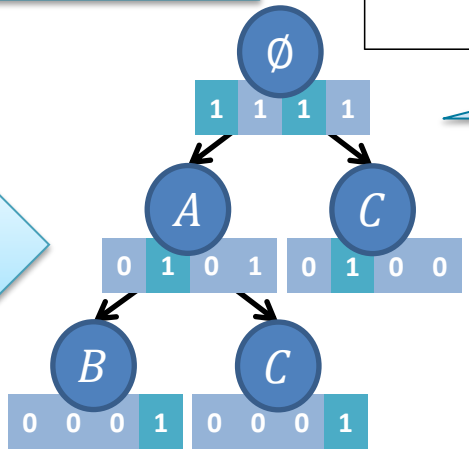
ThorstenPapenbrock

Hybrid FD Discovery



Bitmaps for negative cover





FDTree for positive cover

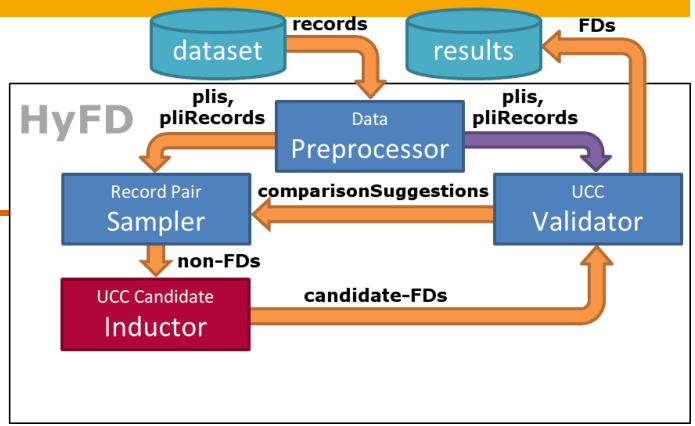
FDTree serves as candidate lattice (and final result!)

Negative Cover (**non-FDs**)

Positive Cover (**candidate-FDs**)

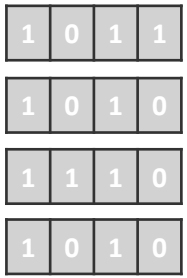
HyFD

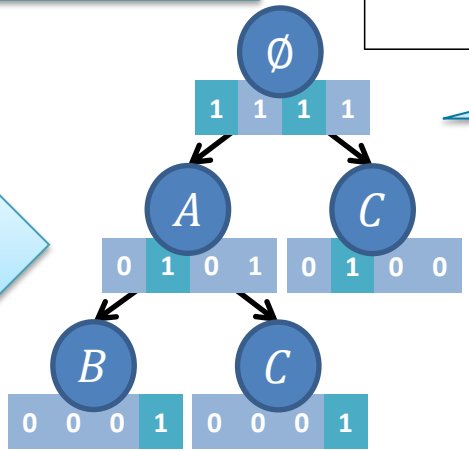
Hybrid FD Discovery



Hasso Plattner Institut

Bitmaps for negative cover





FDTree for positive cover

Negative Cover (**non-FDs**)

Positive Cover (**candidate-FDs**)

Use non-FDs with greatest pruning impact first!

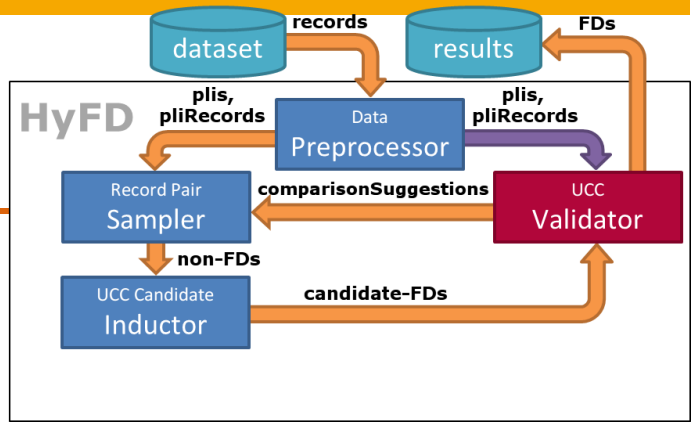
Algorithm 3: Functional Dependency Induction

Data: *nonFds*
Result: *fds*

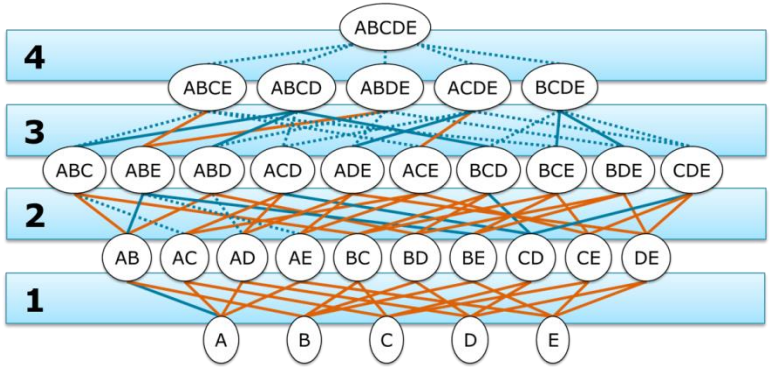
```
1 nonFds ← sort(nonFds, CARDINALITY_DESCENDING);
2 if fds = null then
3     fds ← new FDTree;
4     fds.add(∅ → {0, 1, ..., numAttributes});
5 for lhs ∈ nonFds do
6     rhss ← lhs.clone().flip();
```

Induction = Cover Inversion in fdep

Hybrid FD Discovery



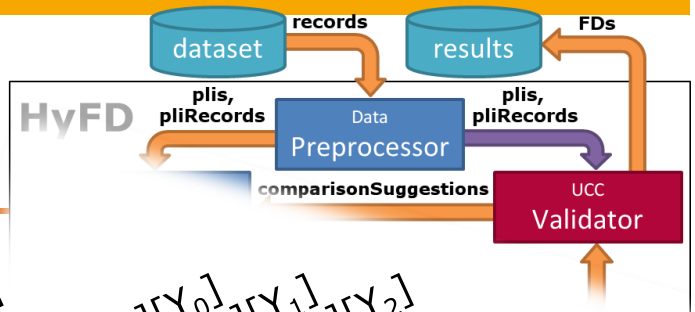
Bottom-up lattice traversal:



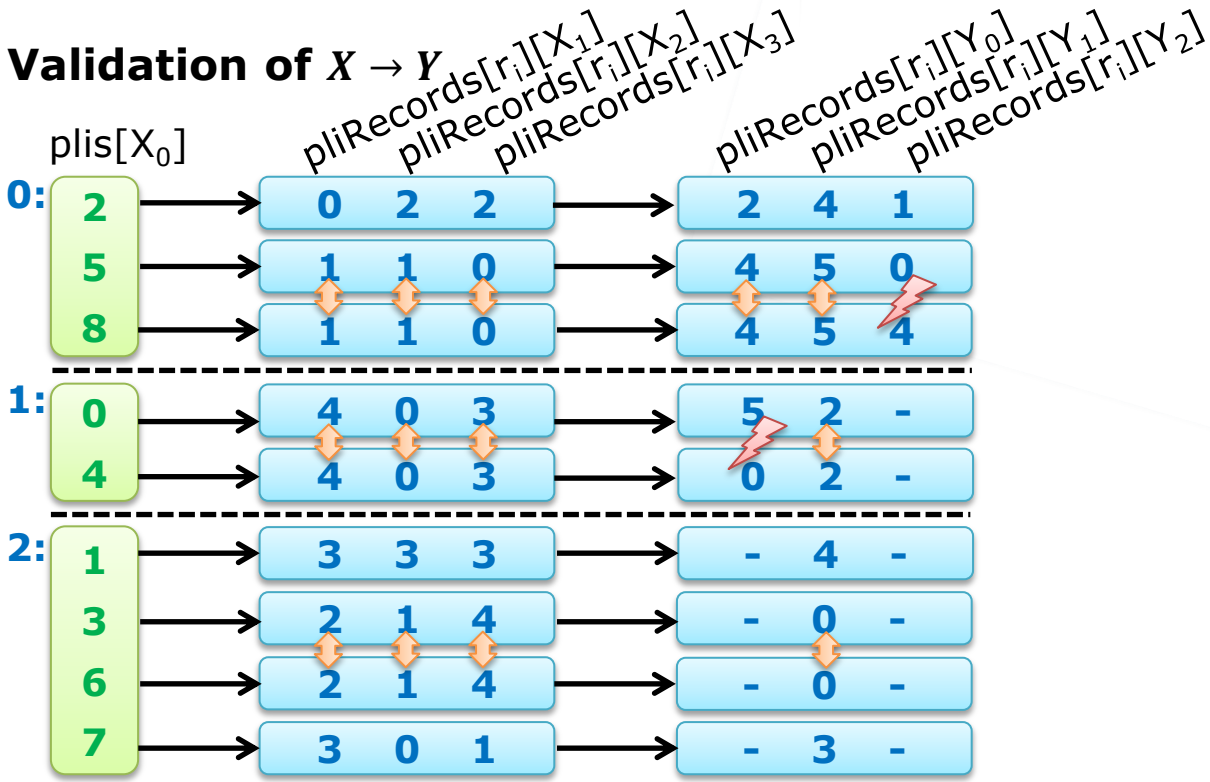
Optimizations:

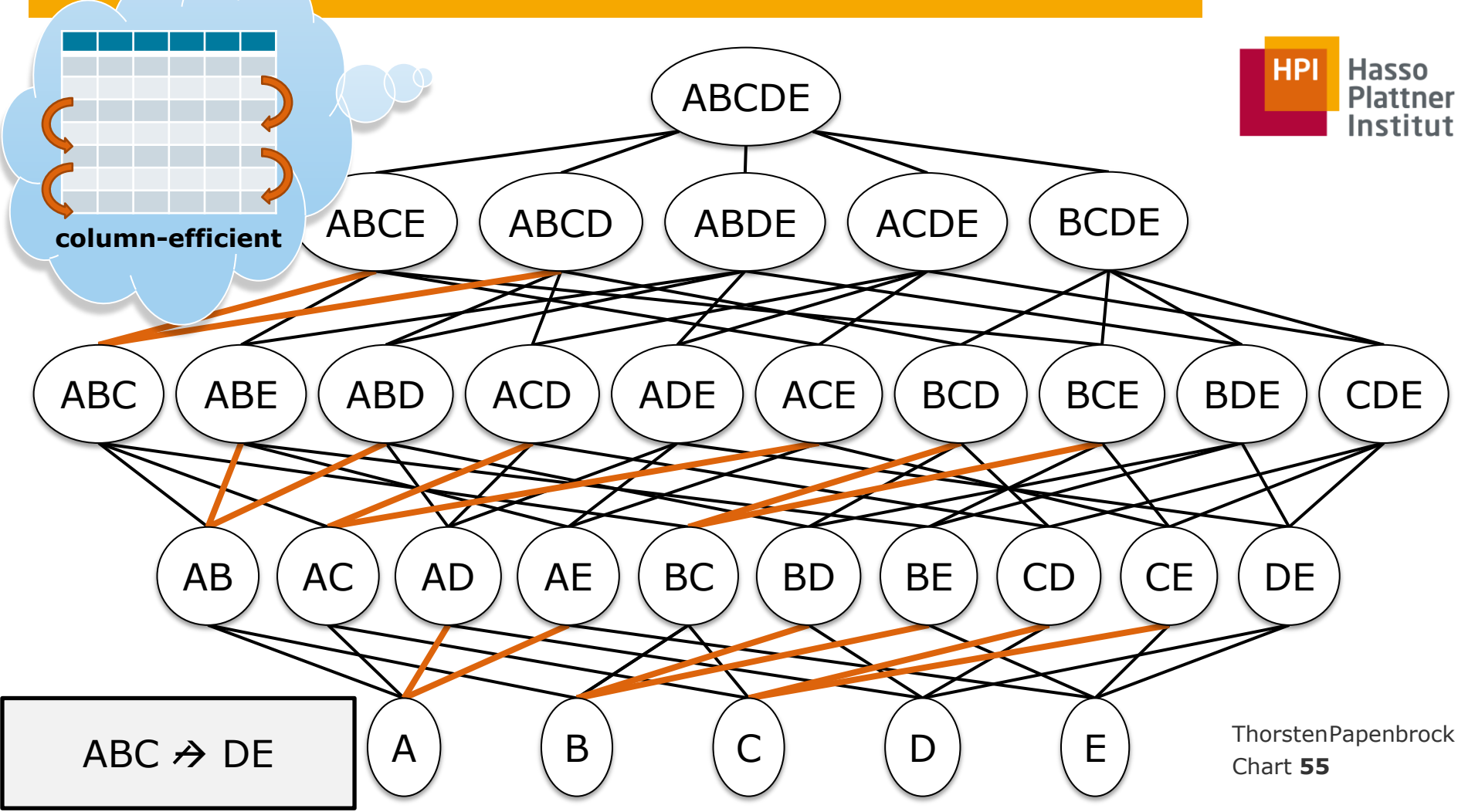
- **Additional Pruning**: Use the results from the sampling.
- **No PLI-caching**: Use only the single column plis and pliRecords for the validaions (do not remember plis of higher arity).
- **Early termination**: Stop PLI intersects if candidates are invalid.
- **Bulk validations**: Check all rhss for one given lhs jointly.

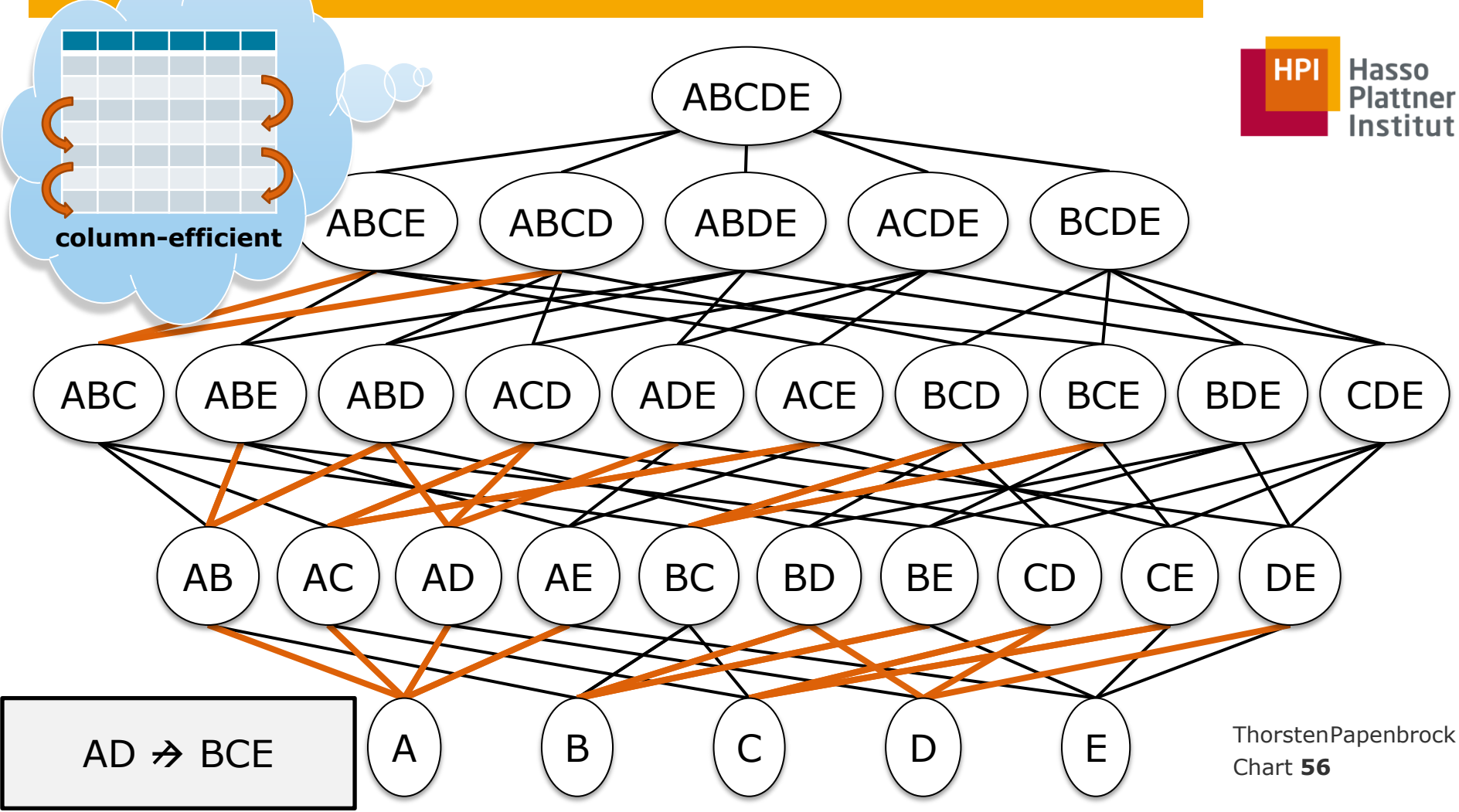
Hybrid FD Discovery

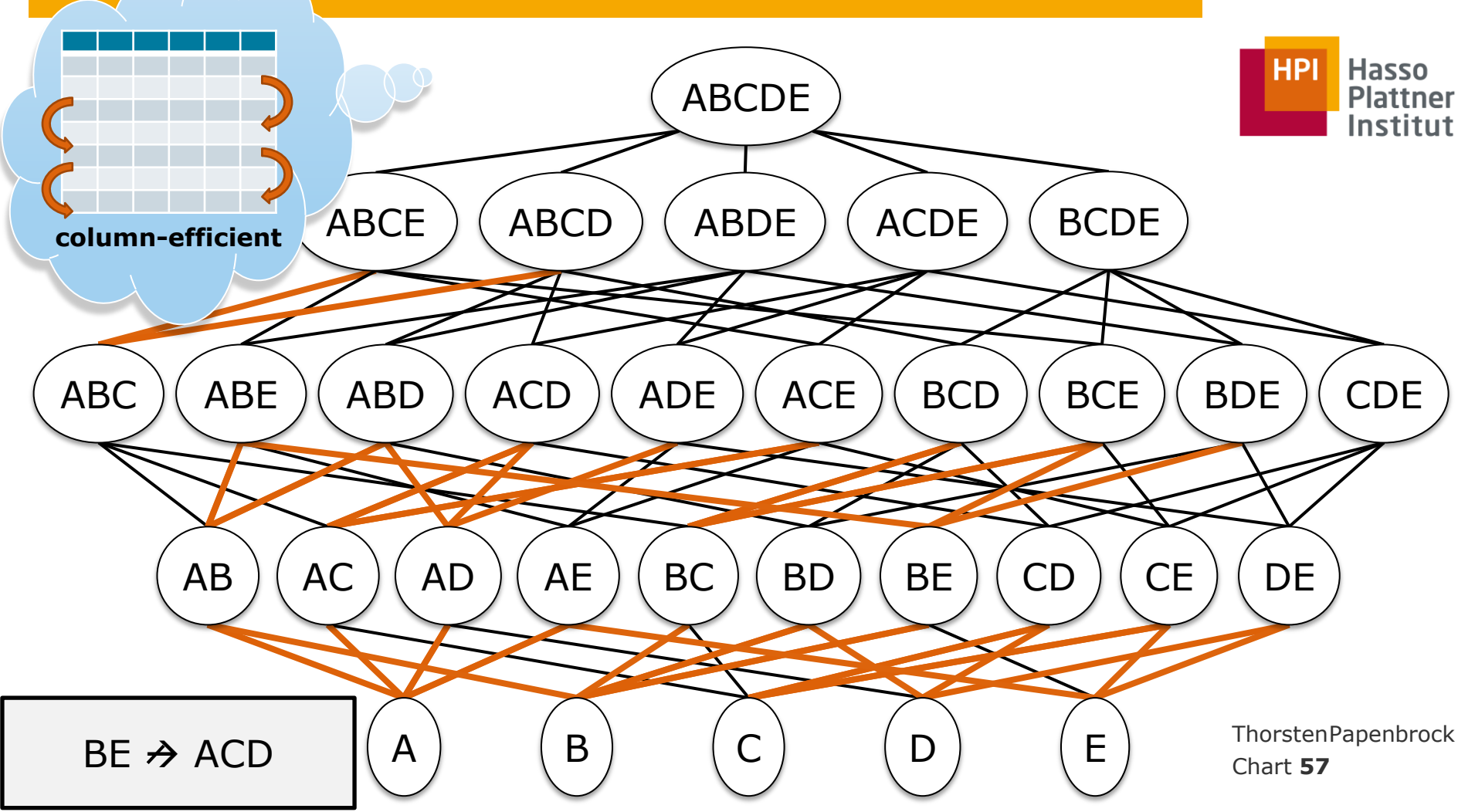


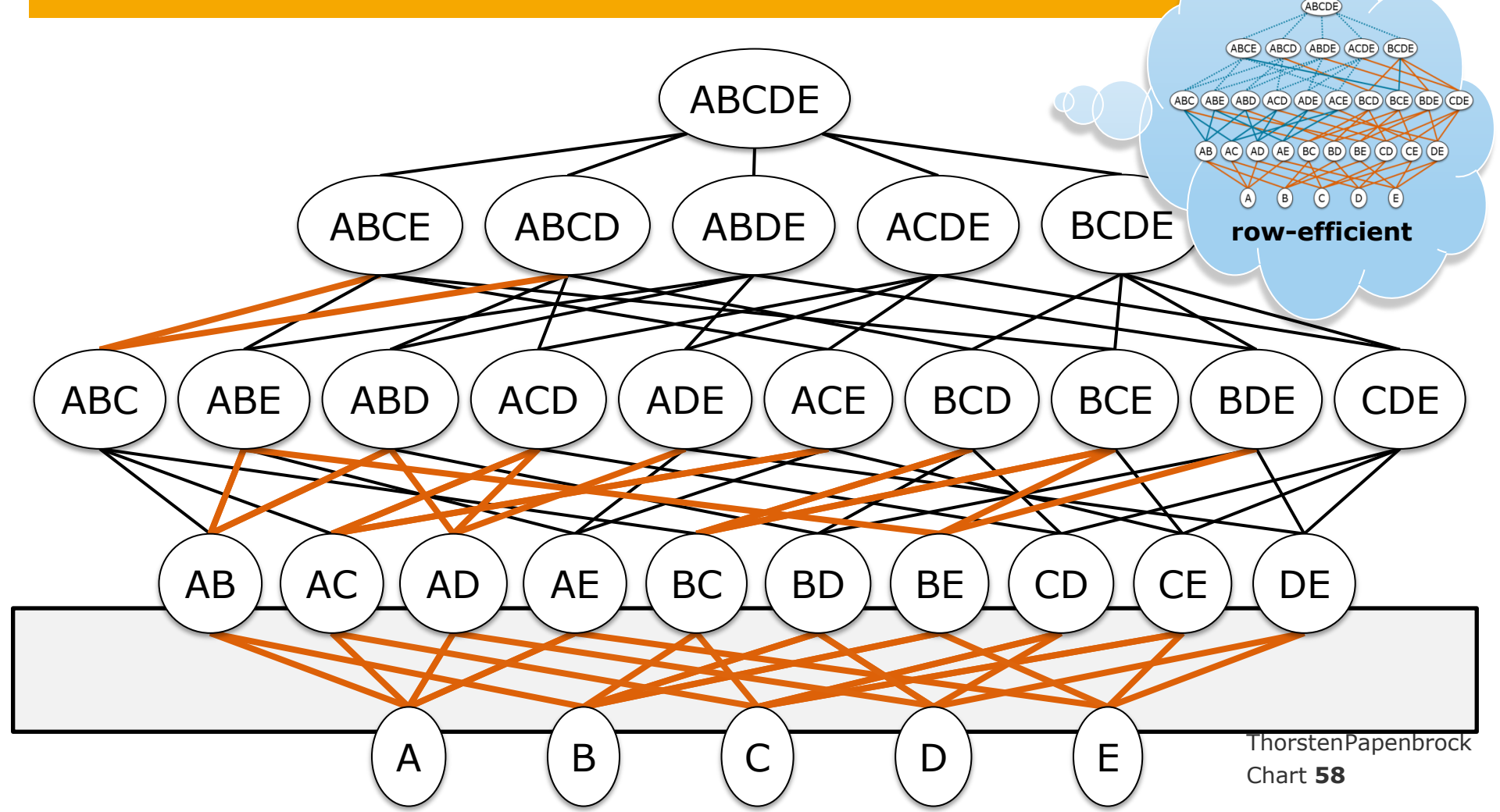
Validation of $X \rightarrow Y$

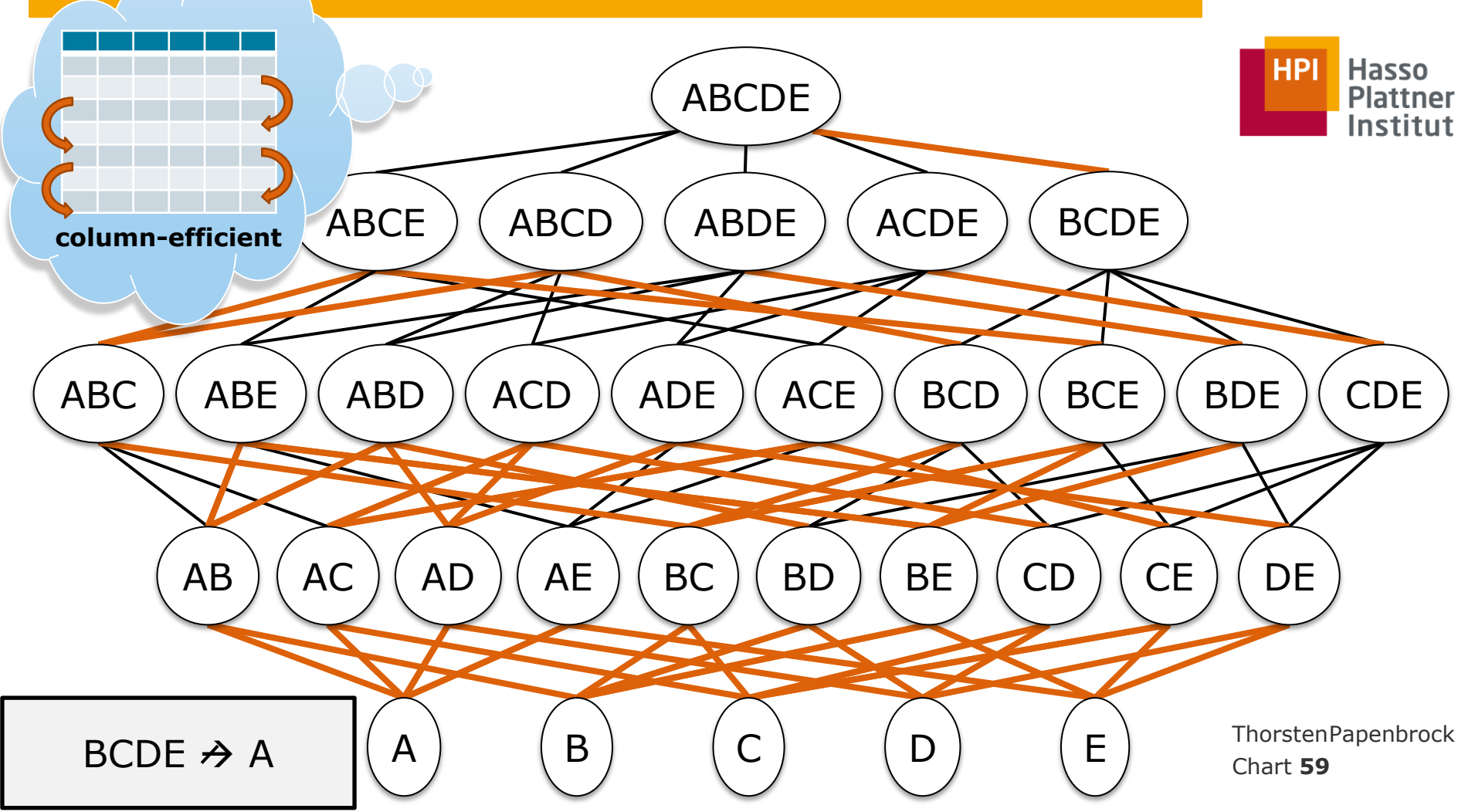


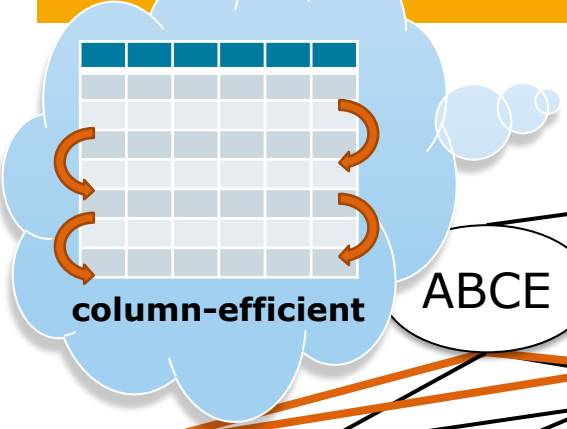

column-efficient
ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
ABC → DE


column-efficient
ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
AD → BCE


column-efficient
ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
BE ↛ ACD

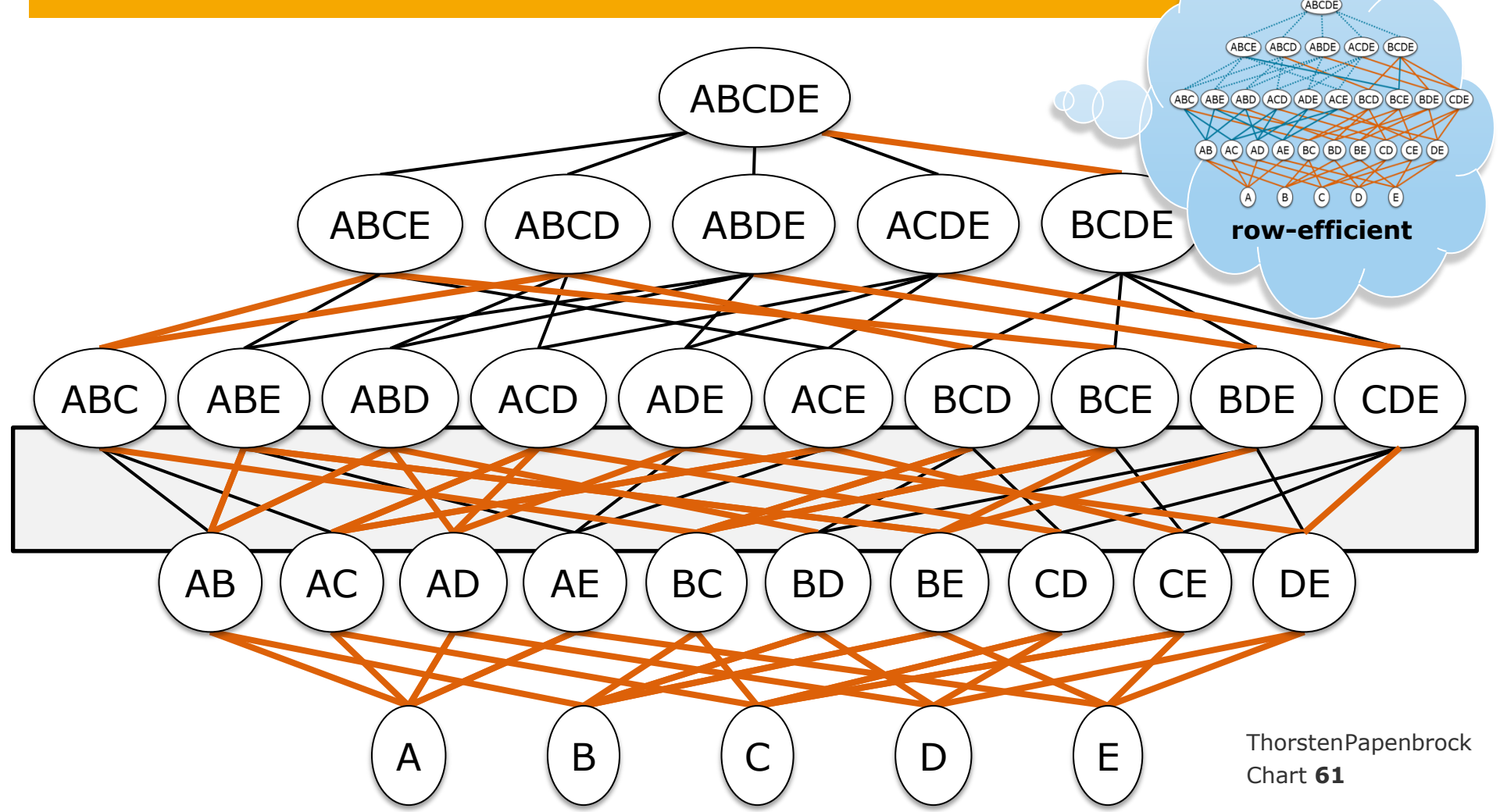

ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
row-efficient

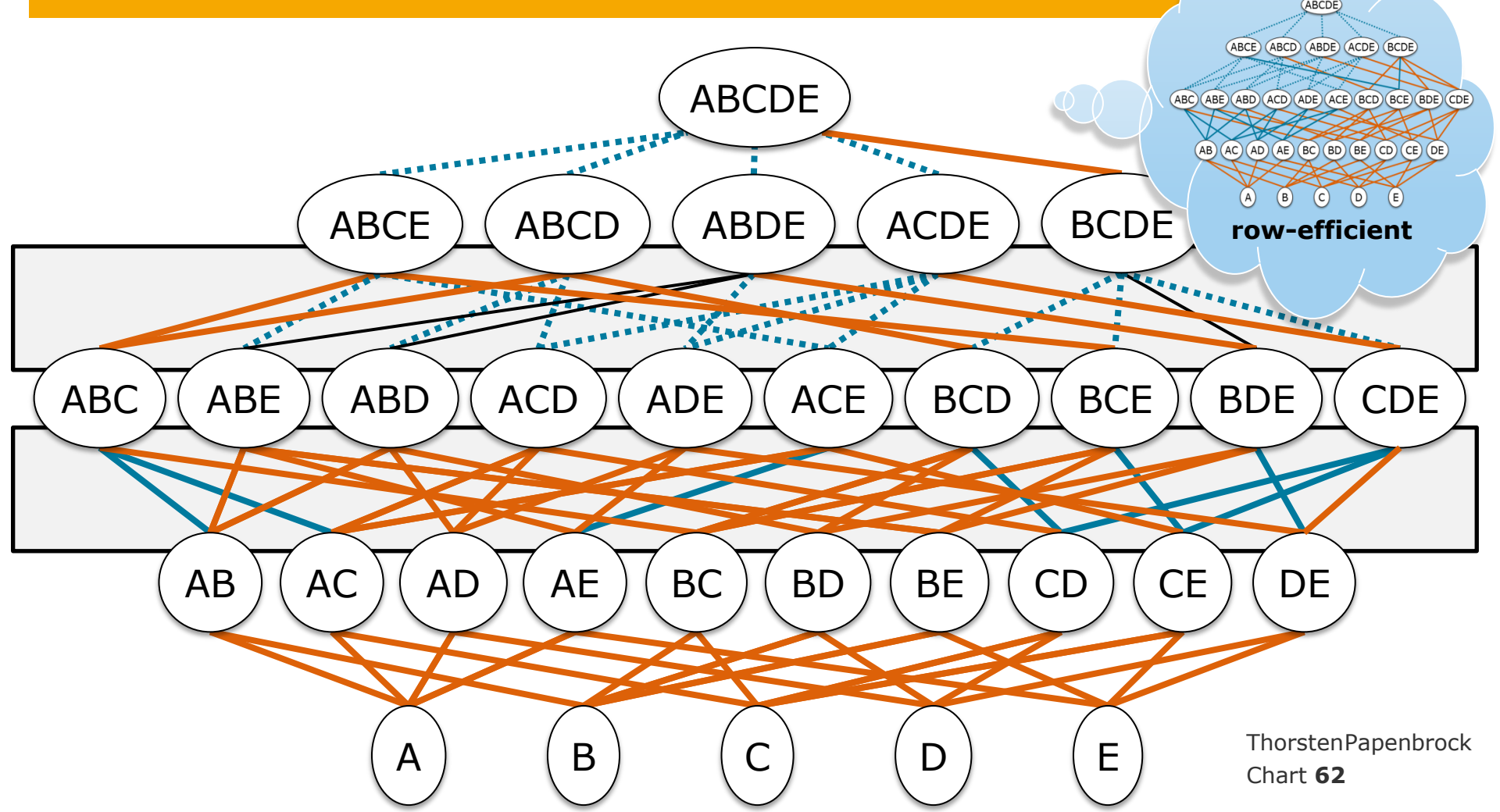

column-efficient
ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
BCDE ↛ A

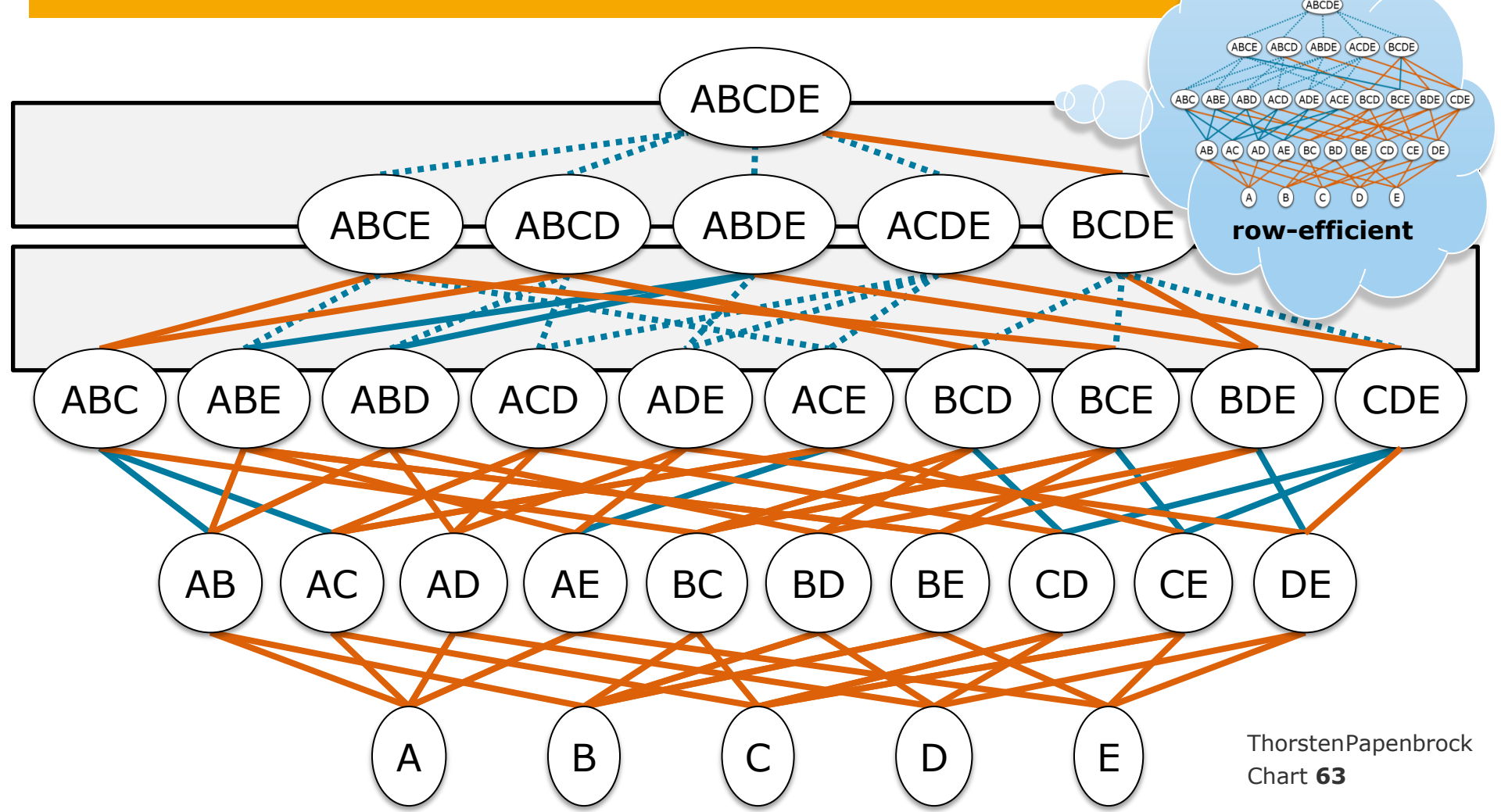

column-efficient

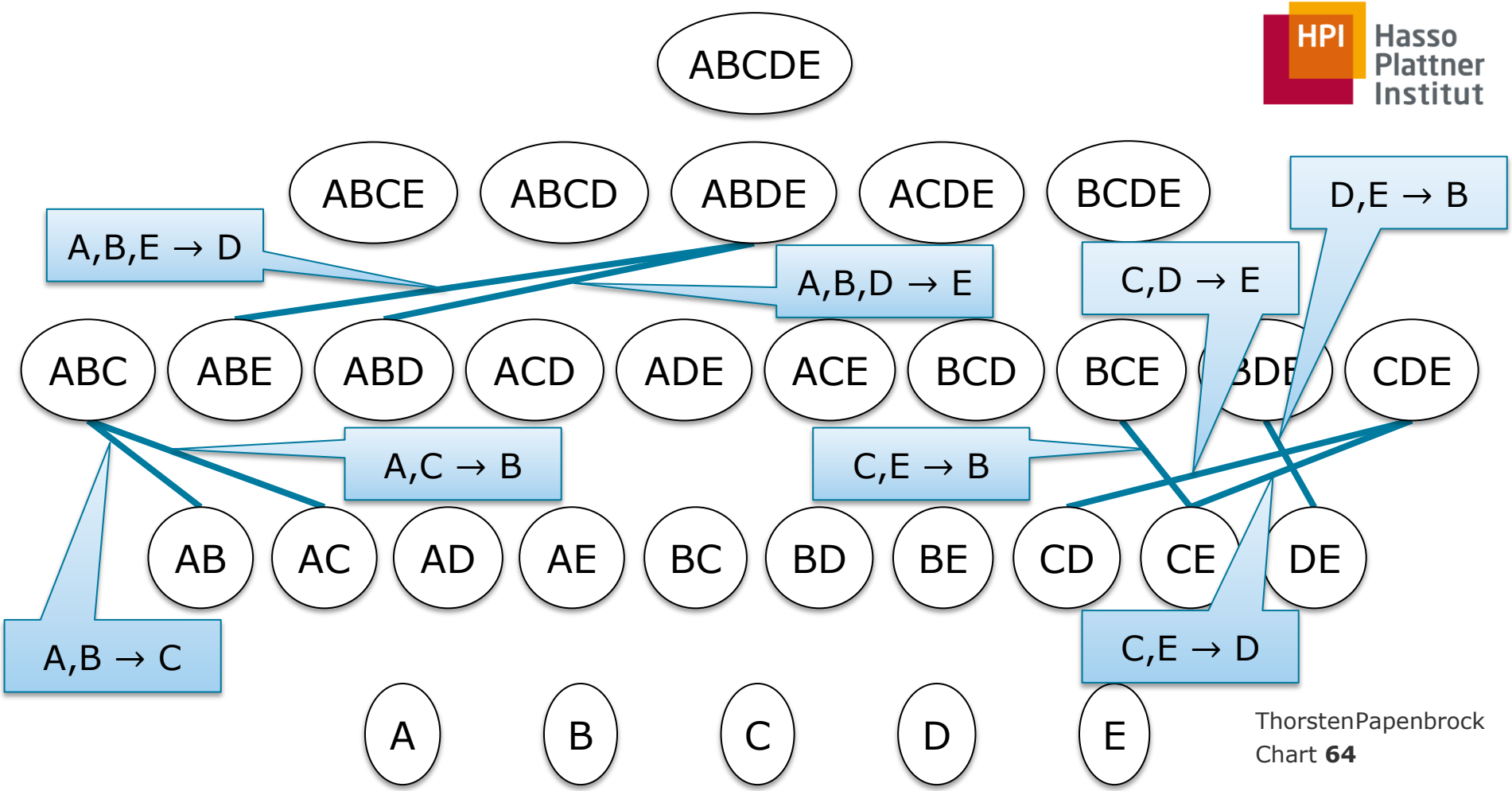
ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E

DE ↛ C

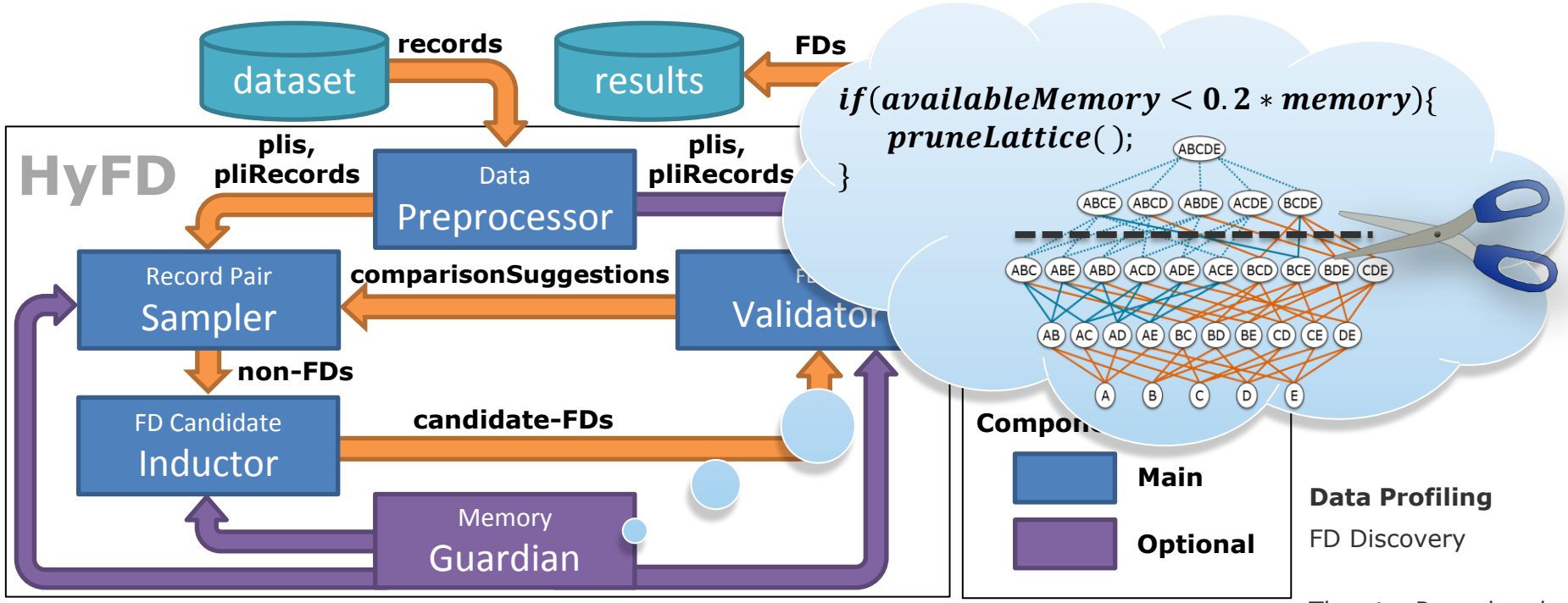

ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
row-efficient


ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
row-efficient

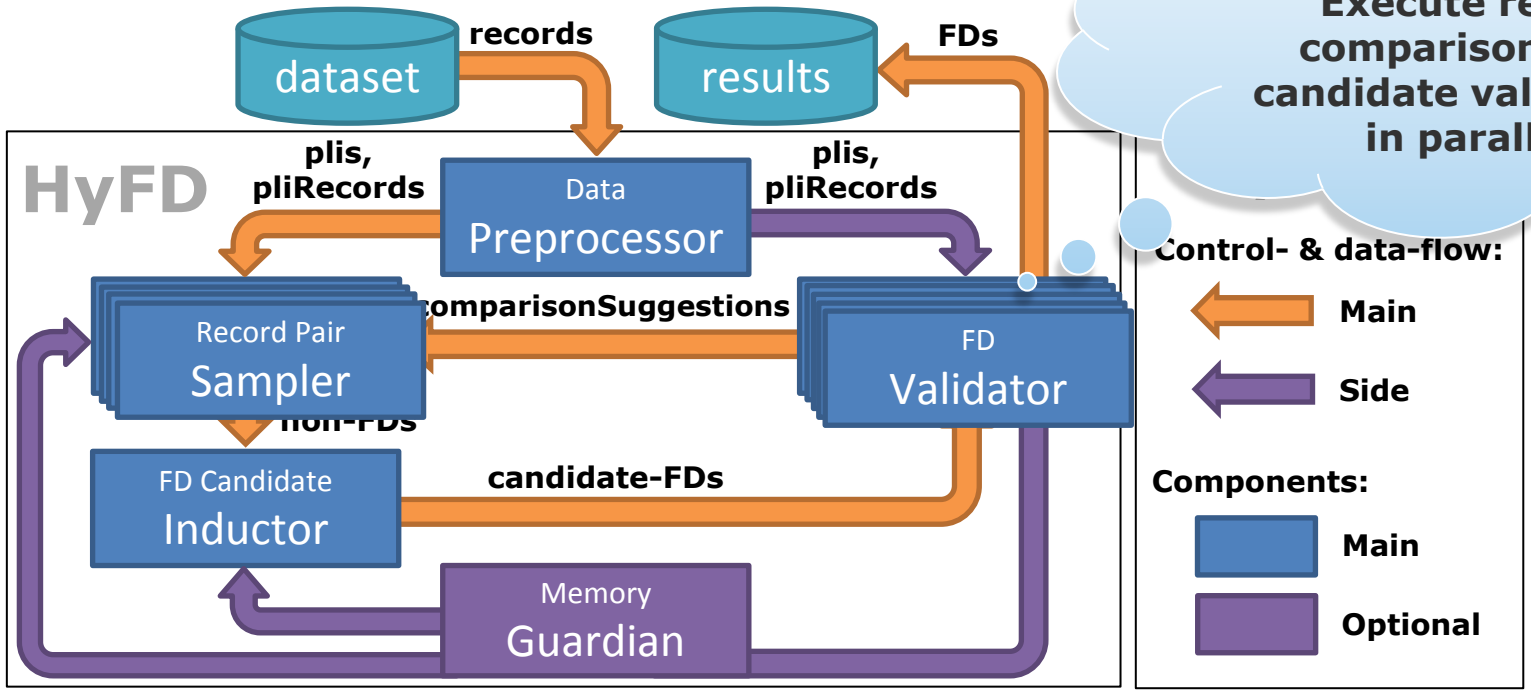

ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
row-efficient


ABCDE
ABCE
ABCD
ABDE
ACDE
BCDE
A,B,E → D
A,B,D → E
C,D → E
D,E → B
ABC
ABE
ABD
ACD
ADE
ACE
BCD
BCE
BDE
CDE
A,C → B
C,E → B
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A,B → C
C,E → D
A
B
C
D
E

Hybrid FD Discovery



Hybrid FD Discovery

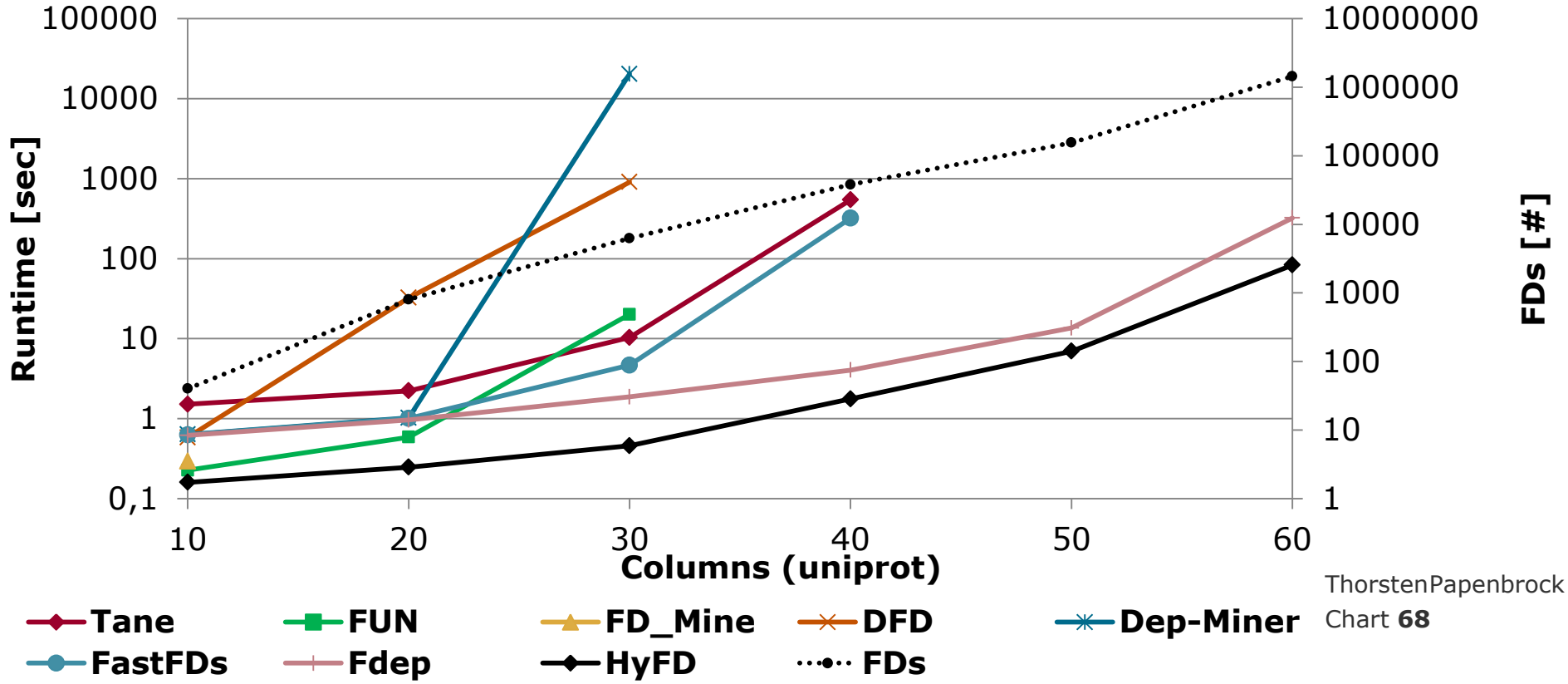


Functional Dependency Discovery

- Motivation
- Approaches
- fdep
- HyFD
- **Evaluation**
- Metanome

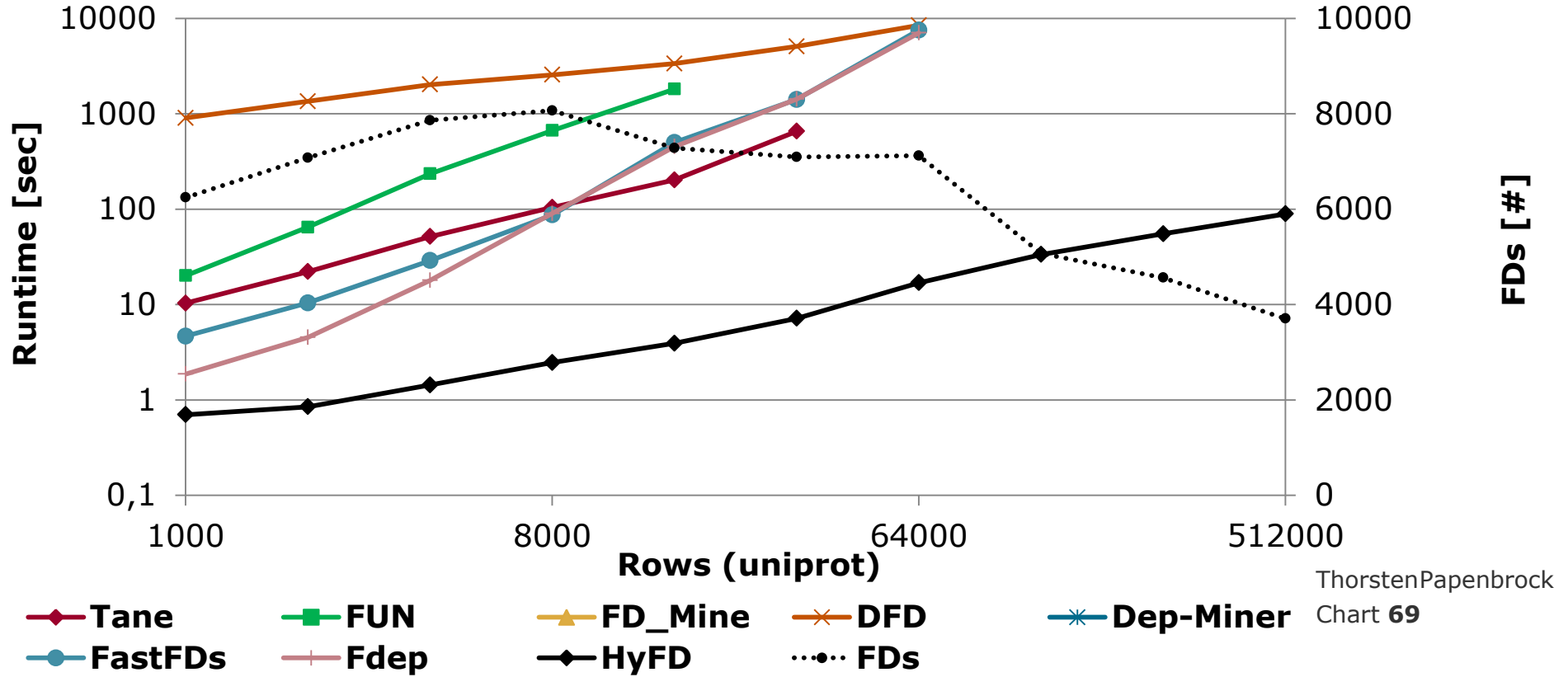
Evaluation

Column Scalability



Evaluation

Row Scalability



Various Datasets

Dataset	Cols [#]	Rows [#]	Size [KB]	FDs [#]
iris	5	150	5	4
balance-scale	5	625	7	1
chess	7	28,056	519	1
abalone	9	4,177	187	137
nursery	9	12,960	1,024	1
breast-cancer	11	699	20	46
bridges	13	108	6	142
echocardiogram	13	132	6	527
adult	14	48,842	3,528	78
letter	17	20,000	695	61
ncvoter	19	1,000	151	758
hepatitis	20	155	8	8,250
horse	27	368	25	128,727
fd-reduced-30	30	250,000	69,581	89,571
plista	63	1,000	568	178,152
flight	109	1,000	575	982,631
uniprot	223	1,000	2,439	>2,437,556

Results larger than 1,000 FDs are only counted **TL**: time limit of 4 hours exceeded **ML**: memory limit of 100 GB exceeded

Table 3.1: Runtimes in seconds for several real-world datasets.

TANE [Huhtala et al., 1999] FUN [Novelli and Cicchetti, 2001] FD_MINE [Yao et al., 2002] DFD [Abedjan et al., 2014c]
DEP-MINER [Lopes et al., 2000] FASTFDS [Wyss et al., 2001] FDEP [Flach and Savnik, 1999] HYFD [Papenbrock and Naumann, 2016]

Various Datasets

Dataset	Cols [#]	Rows [#]	Size [MB]	FDs [#]	HyFD [s/m/h/d]	
TPC-H.lineitem	16	6 m	1,051	4 k	**39 m**	4 m
PDB.POLY_SEQ	13	17 m	1,256	68	**4 m**	3 m
PDB.ATOM_SITE	31	27 m	5,042	10 k	**12 h**	64 m
SAP_R3.ZBC00DT	35	3 m	783	211	**4 m**	2 m
SAP_R3.ILOA	48	45 m	8,731	16 k	**35 h**	8 h
SAP_R3.CE4HI01	65	2 m	649	2 k	**17 m**	10 m
NCVoter.statewide	71	1 m	561	5 m	**10 d**	31 h
CD.cd	107	10 k	5	36 k	**5 s**	3 s

sequential

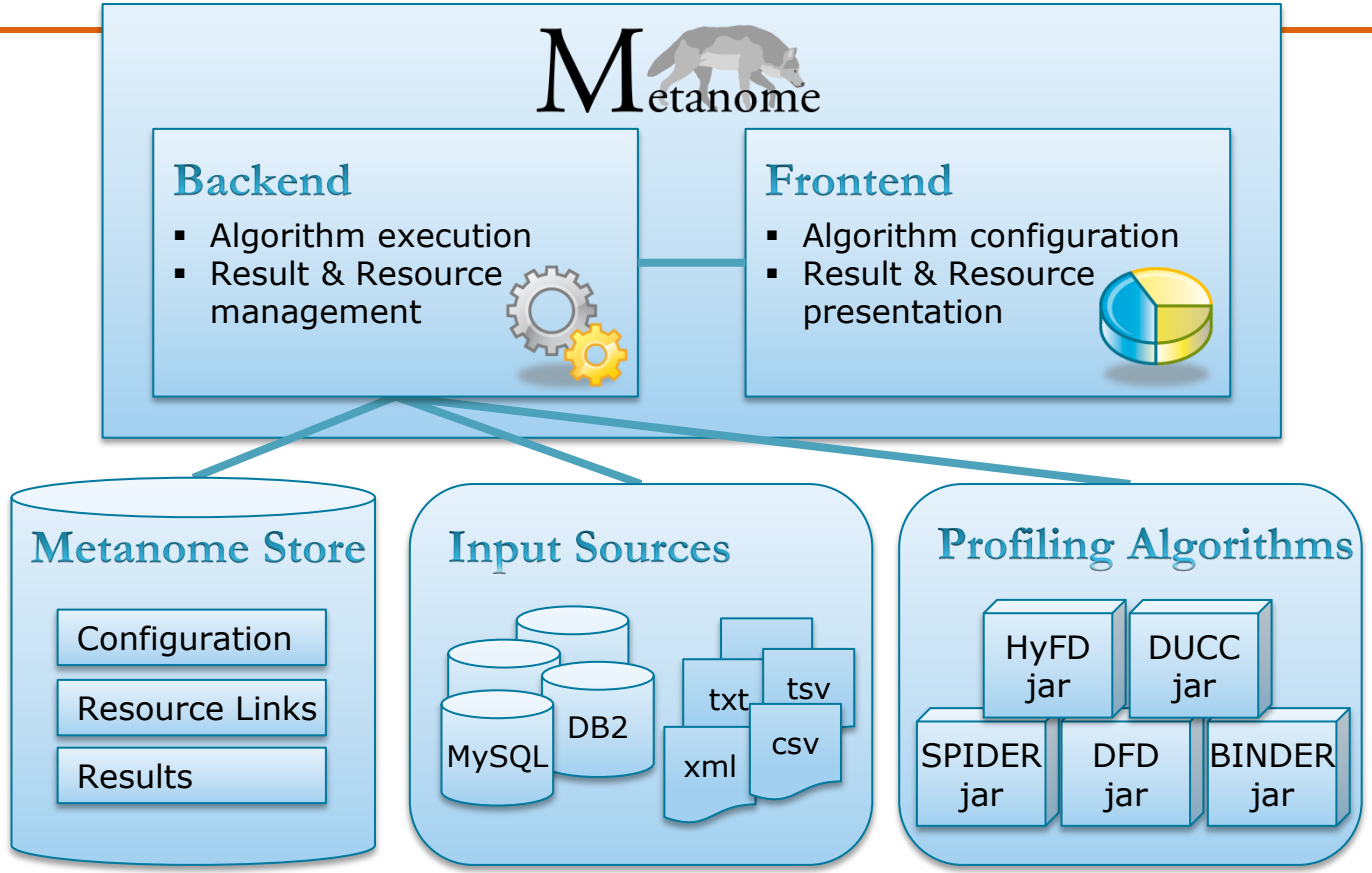
32 x parallel

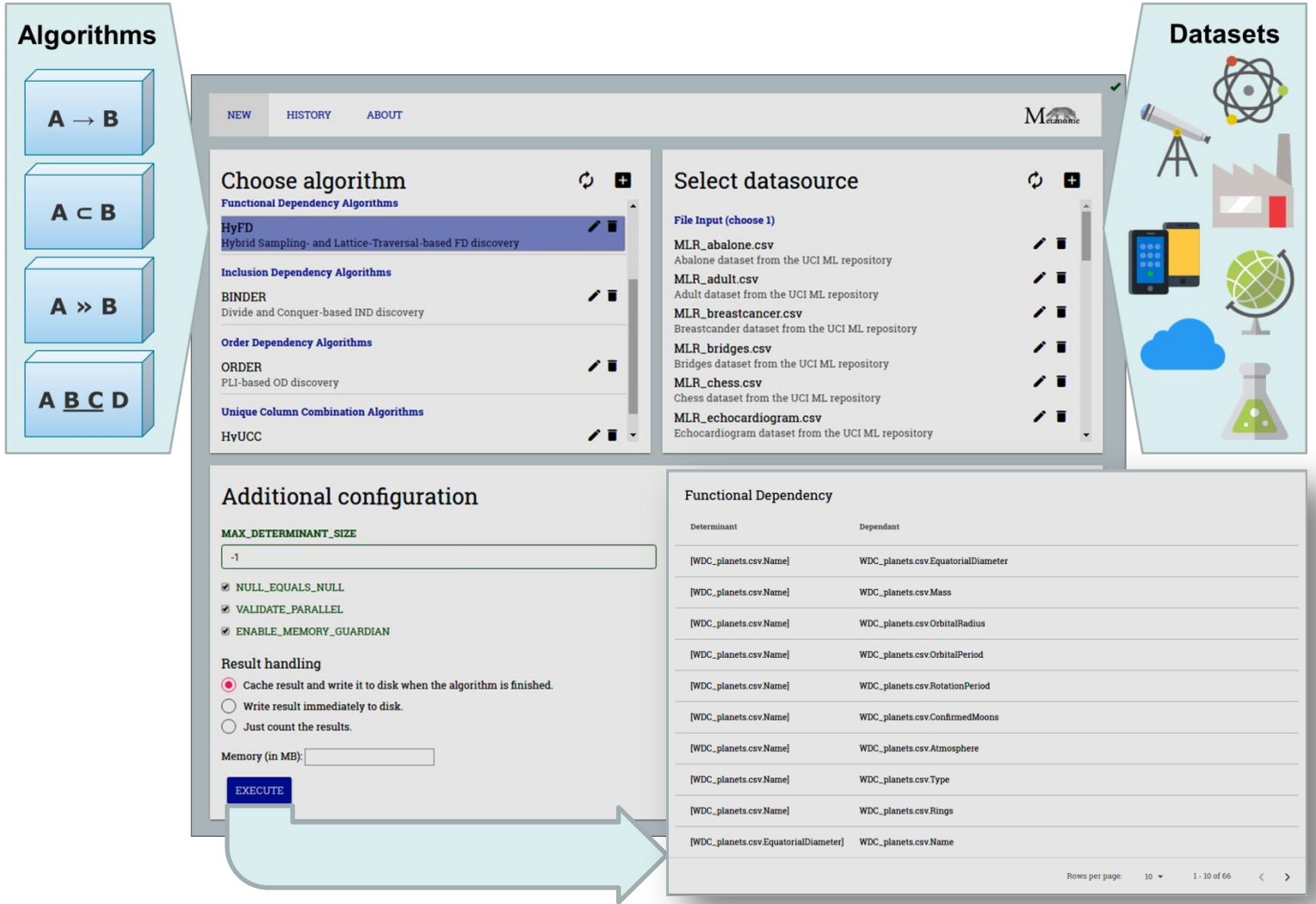
Functional Dependency Discovery

- Motivation
- Approaches
- fdep
- HyFD
- Evaluation
- **Metanome**

www.metanome.de

Architecture





Data Profiling

FD Discovery

ThorstenPapenbrock

Metanome

Developers

Jakob Zwiener

Claudia Exeler

Tanja Bergmann

Moritz Finke

Fabian Tschirschnitz

Carl Ambroselli

Maxi Fischer

Vincent Schwarzer

?

?

Data Profiling

FD Discovery

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Data Profiling
Functional Dependency Discovery

Thorsten Papenbrock
G-3.1.09, Campus III
Hasso Plattner Institut