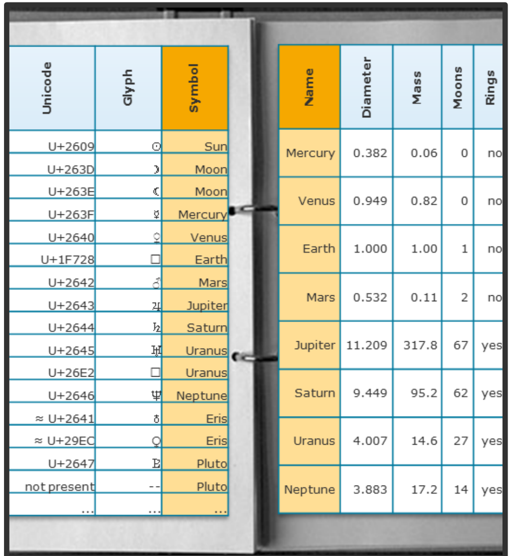
Advanced IND detection methods

Sebastian Kruse
sebastian.kruse@hpi.de
G-3.1.13 (2.5.13)

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Agenda



BINDER
divide & conquer
based IND detection

Sign	House	Domicile	Detriment	Exaltation
Aries	1st House	Mars	Venus	Sun
Taurus	2nd House	Venus	Pluto	Moon
Gemini	3rd House	Mercury	Jupiter	N/A
Cancer	4th House	Moon	Saturn	Jupiter
Leo	5th House	Sun	Uranus	Neptune
Virgo	6th House	Mercury	Neptune	Pluto, Mercury
Libra	7th House	Venus	Mars	Saturn
Scorpio	8th House	Pluto	Venus	Uranus
agittarius	9th House	Jupiter	Mercury	N/A
Capricorn	10th House	Saturn	Moon	Mars

SINDY
scaling out
IND detection

Name	Mass	Orbital radius	Rotation period	Atmosphere
Mercury	0.06	0.47	58.64	minimal
Venus	0.82	0.72	−243.02	CO_2, N_2
Earth	1.00	1.00	1.00	N_2, O_2, Ar
Mars	0.11	1.52	1.03	CO_2, N_2, Ar
Jupiter	317.8	5.20	0.41	H_2, He
Saturn	95.2	9.54	0.43	H_2, He
Uranus	14.6	19.22	−0.72	H_2, He
Neptune	17.2	30.06	0.67	H_2, He

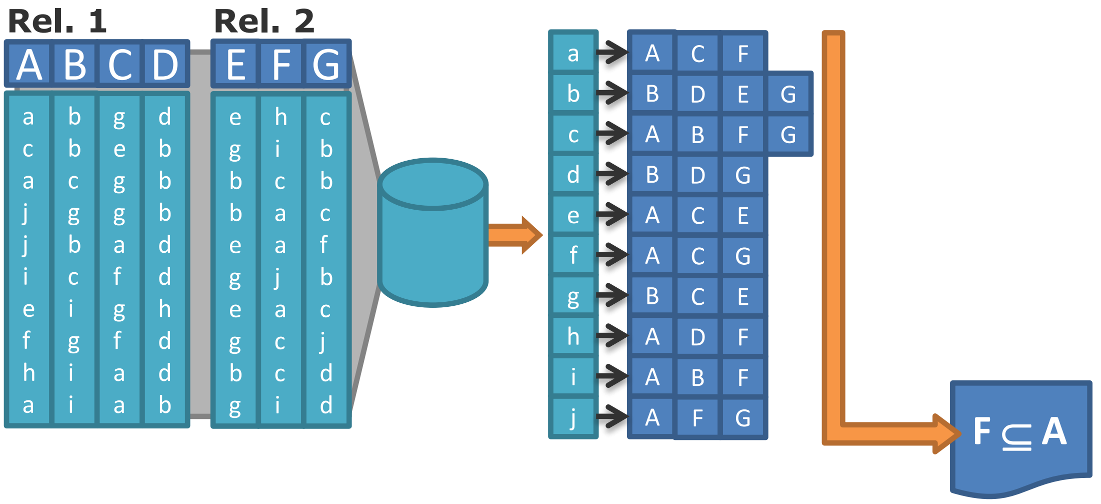
High arity
techniques to deal
with high-arity INDs

Unary IND complexity (Repetition)

- Unary IND discovery has a complexity of $O(n^2)$ (n: number of attributes)
  - Databases often comprise thousands of columns
    → millions of IND candidates to be checked
- Checking an IND candidate requires "aligning" the values of the involved columns
  - Databases often comprise millions or billions of tuples
    → huge amounts of data need to be re-organized
- Call for efficient, robust, and scalable IND discovery strategies.

DeMarchi‵s algorithm (Repetition)

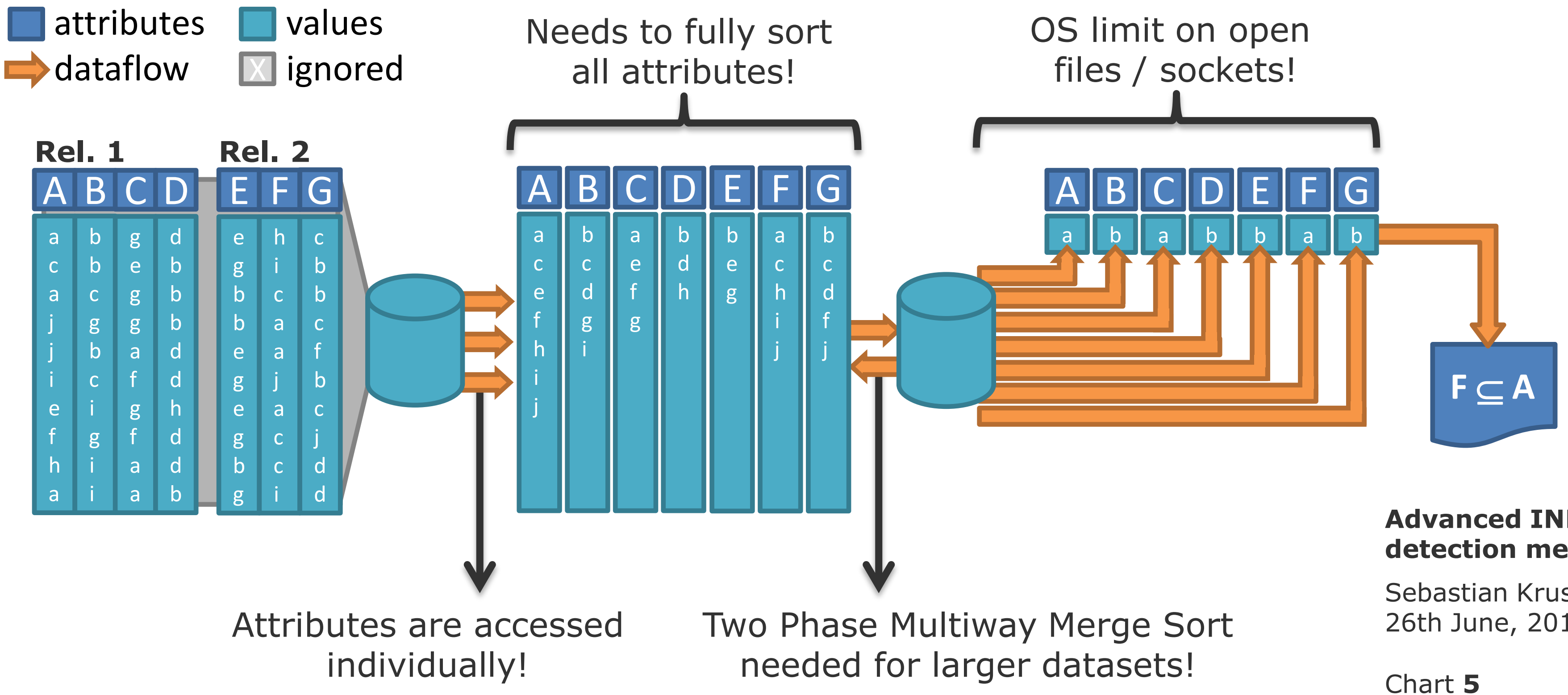
attributes values
dataflow ignored



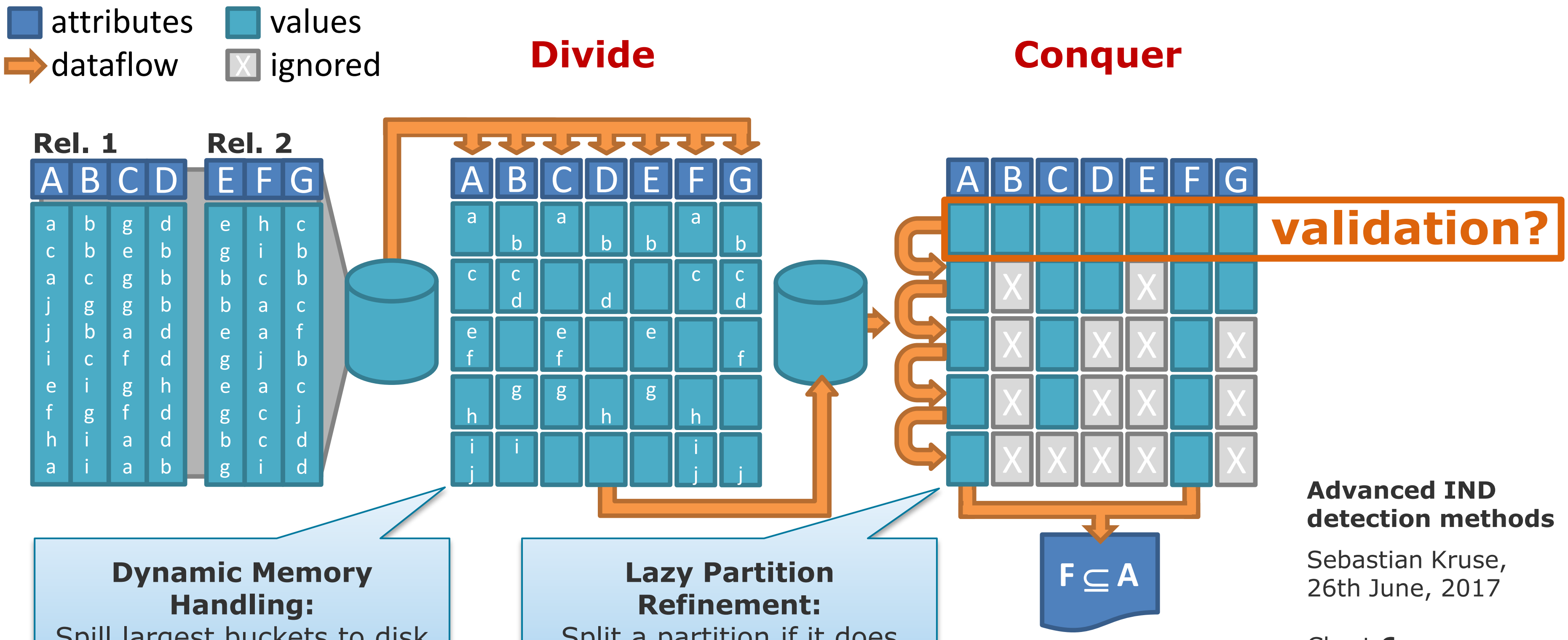
All intersections are executed, but not all are necessary!

Needs to fit into main memory!

SPIDER algorithm (Repetition)

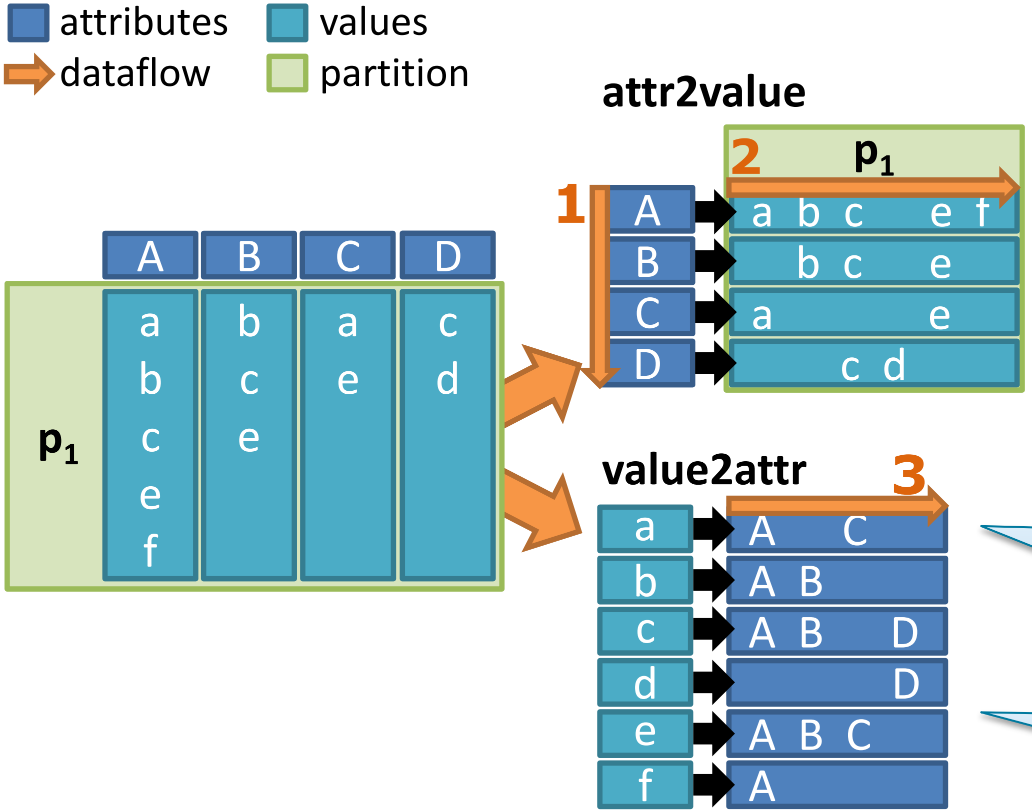


BINDER algorithm – workflow (unary)



$$size(bucket) = \sum_{string \in bucket} 8 + 8 \cdot \left\lceil \frac{64 + 2 \cdot |string|}{8} \right\rceil$$

BINDER algorithm – validation

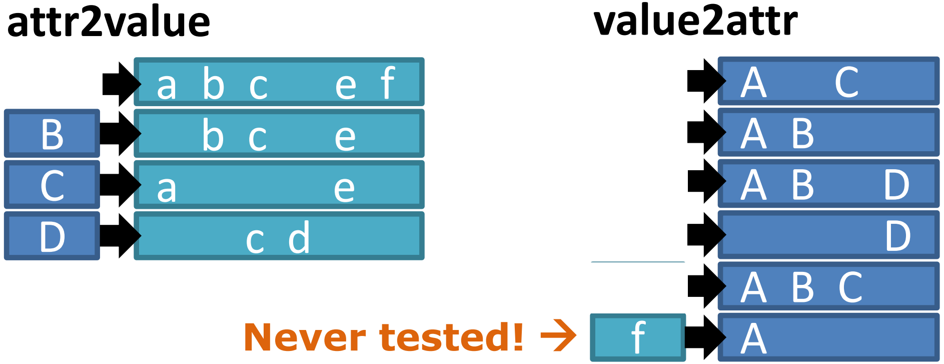


1. Iterate attributes
2. Iterate values
3. If value2attr entry exists
   - Intersect candidates with this list
   - Remove value2attr entry
   - If attribute removed from all candidates
     - Remove entry from attr2value

Both indexes fit into main memory due to the partitioning!

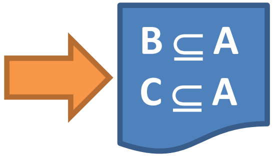
see DeMarchi's algorithm

BINDER algorithm – validation example



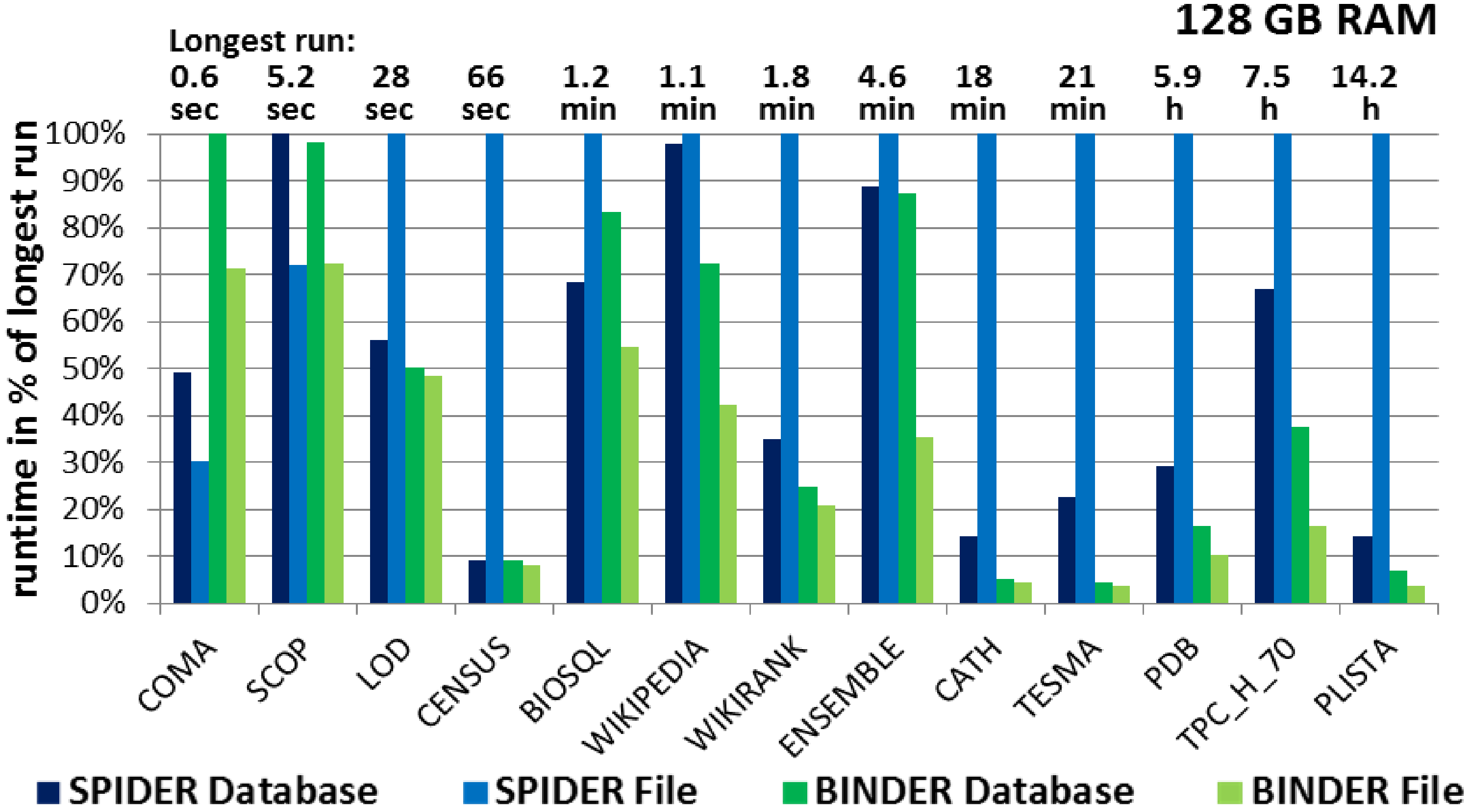
	A	B	C	D
look up	B,C,D	A,C,D	A,B,D	A,B,C

1. Iterate attributes
2. Iterate values
3. If value2attr entry exists
   - Intersect candidates with this list
   - Remove value2attr entry
   - If attribute removed from all candidates
     - Remove entry from attr2value



BINDER – divide & conquer based IND detection

BINDER evaluation

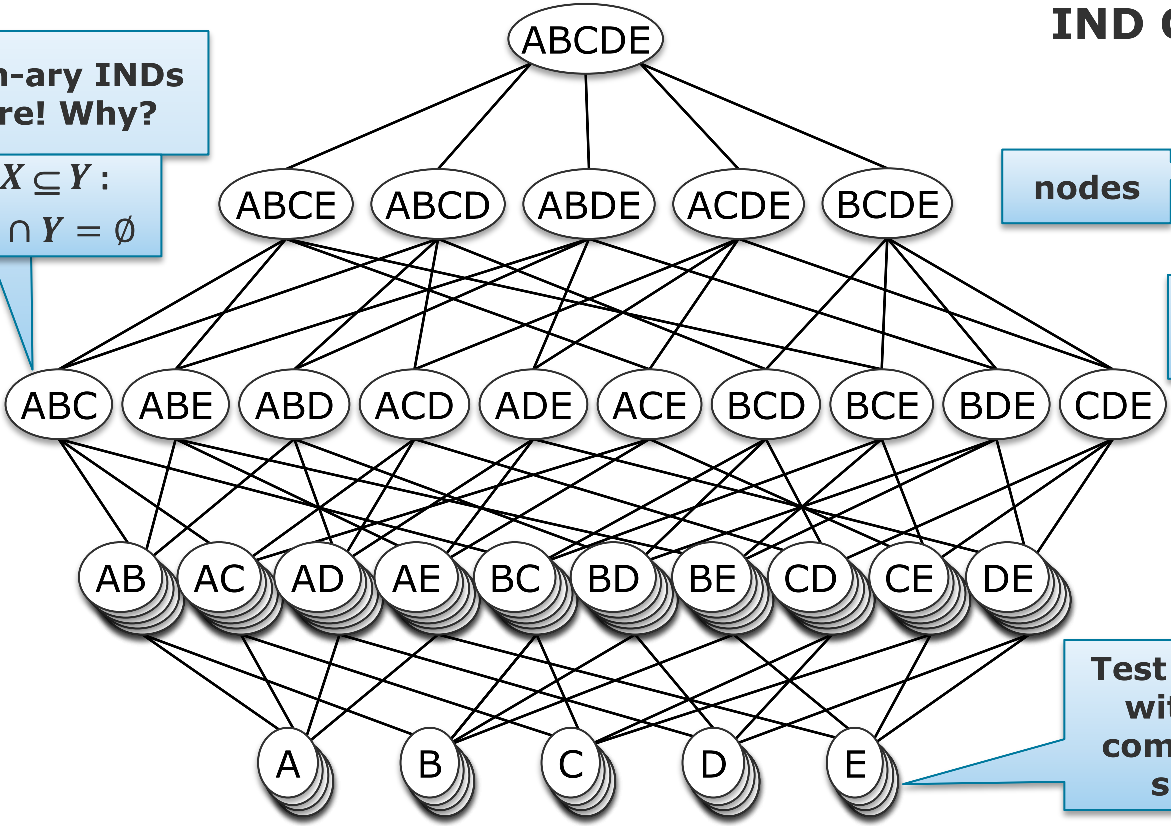


N-ary IND detection complexity

No n-ary INDs here! Why?

$X \subseteq Y$:

$X \cap Y = \emptyset$



IND Candidates in level k:

$$\underbrace{\binom{n}{k}}_{X} * \underbrace{\binom{n-k}{k}}_{Y} * k!$$

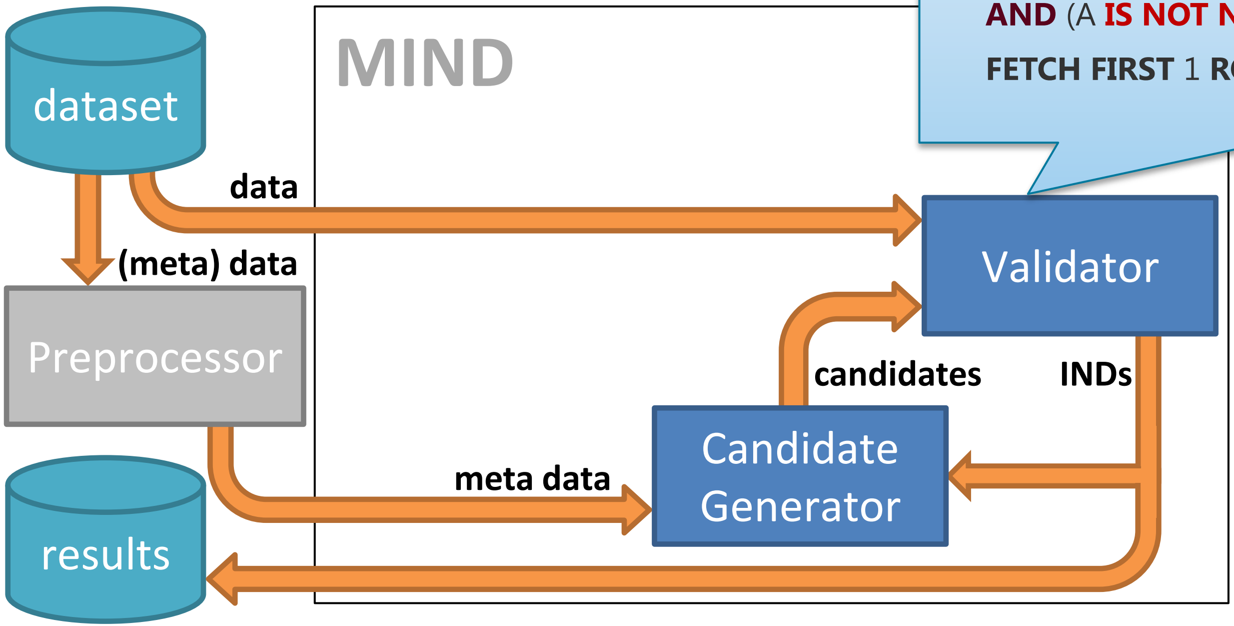
nodes (X)

other, non-overlapping nodes (Y)

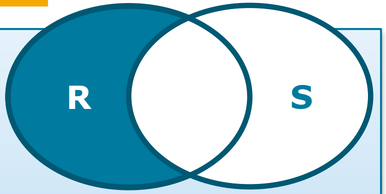
all permu-tations ($k!$)

Test combination with all other combinations of same size!

MIND (Recap)



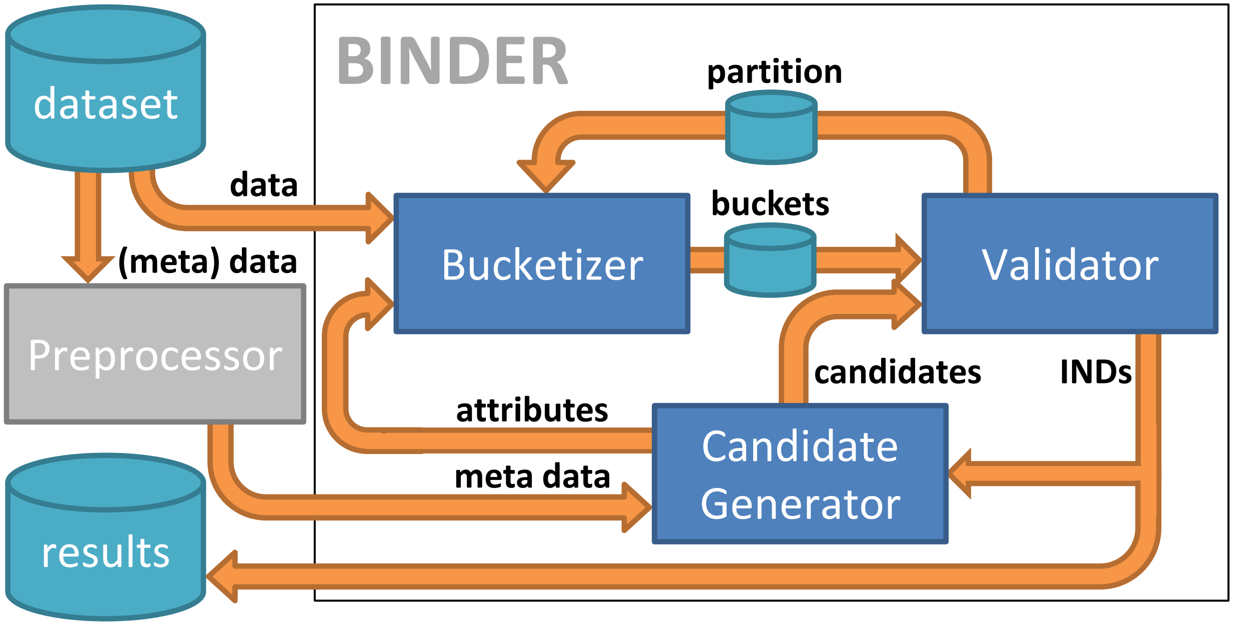
Validate A,B ⊆ C,D:




```
SELECT R.A, R.B, S.C, S.D
FROM R LEFT OUTER JOIN S ON A = C AND B = D
WHERE (C IS NULL AND D IS NULL)
AND (A IS NOT NULL OR B IS NOT NULL)
FETCH FIRST 1 ROWS ONLY;
```

```
SELECT fromTable.AAGE, fromTable.ACLSWKR,
       fromTable.ADTIND, fromTable.ADTOCC,
       fromTable.AGI, fromTable.AHGA
FROM CENSUS6 fromTable LEFT OUTER JOIN
      CENSUS exceptTable ON
      fromTable.AAGE = exceptTable.AAGE AND
      fromTable.ACLSWKR = exceptTable.ACLSWKR
AND
      fromTable.ADTIND = exceptTable.ADTIND AND
      fromTable.ADTOCC = exceptTable.ADTOCC AND
      fromTable.AGI = exceptTable.AGI AND
      fromTable.AHGA = exceptTable.AHGA
WHERE exceptTable.AAGE IS NULL AND
       exceptTable.ACLSWKR IS NULL AND
       exceptTable.ADTIND IS NULL AND
       exceptTable.ADTOCC IS NULL AND
       exceptTable.AGI IS NULL AND
       exceptTable.AHGA IS NULL
AND (fromTable.AAGE IS NOT NULL OR
      fromTable.ACLSWKR IS NOT NULL OR
      fromTable.ADTIND IS NOT NULL OR
      fromTable.ADTOCC IS NOT NULL OR
      fromTable.AGI IS NOT NULL OR
      fromTable.AHGA IS NOT NULL)
FETCH FIRST 1 ROWS ONLY;
```

N-ary BINDER – workflow



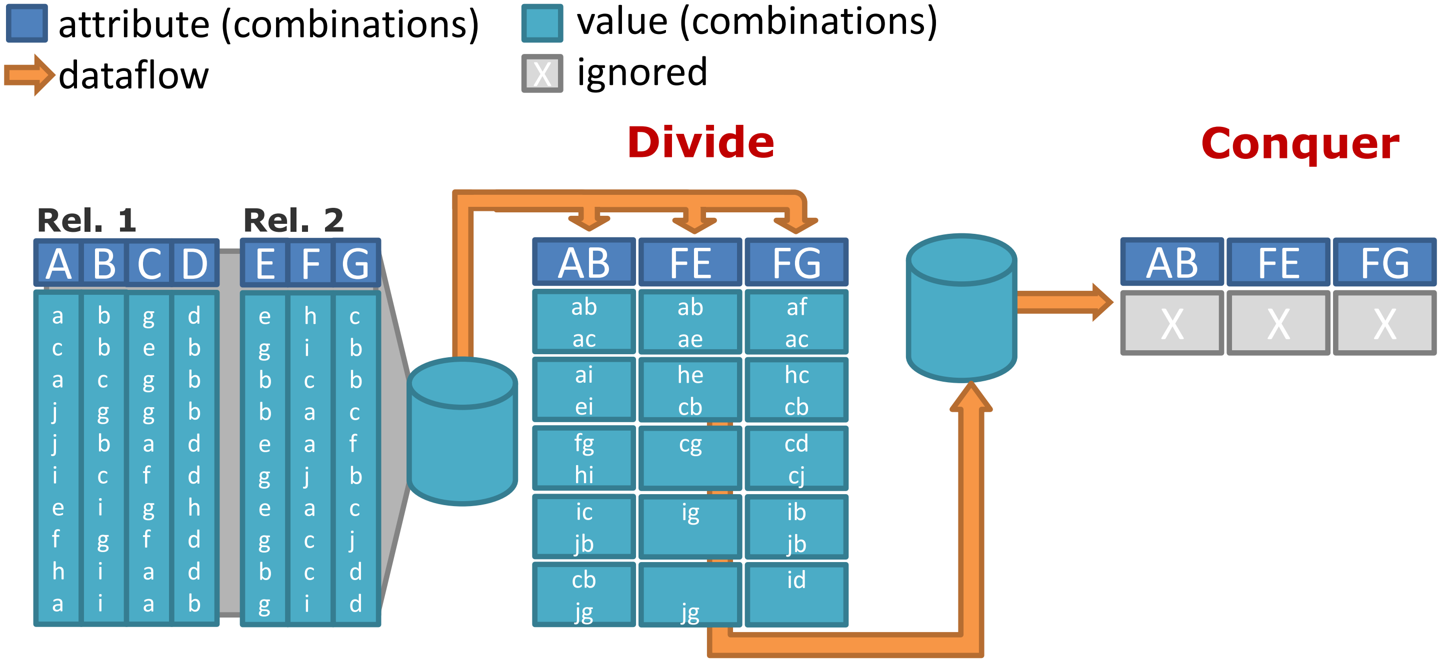
N-ary BINDER – candidate generation

- **Apriori algorithm:**
  - Bottom-up lattice traversal strategy
  - Authors: R.Agrawal and R. Srikant
  - Publication: Fast algorithms for mining association rules in large databases
  - Input: all frequent item sets of size n
  - Output: all candidate frequent item sets of size n+1
- **Adaption for n-ary IND detection:**
  - Let R_i be the i-th relation in the relational schemata R. For each valid IND $R_j[X] \subseteq R_k[Y]$ with $|X|=|Y|=n$ generate all IND candidates $R_j[XA] \subseteq R_k[YB]$ so that:

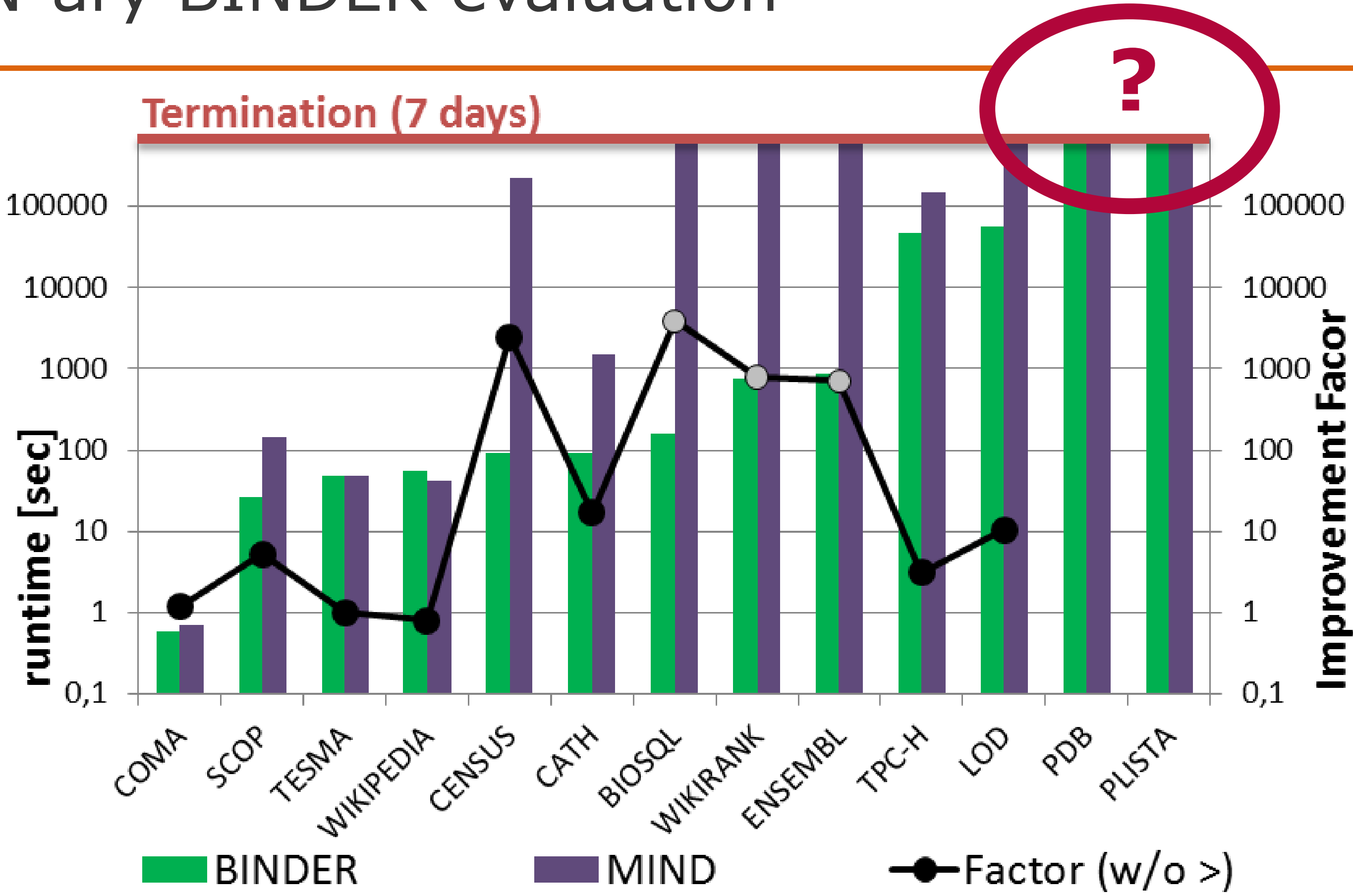
    1. $R_j[X] \subseteq R_k[Y]$ and $R_j[A] \subseteq R_k[B]$ (both are valid INDs)
    2. $\forall X_i \in X: Xi < A$ (INDs are permutable; do not generate them twice)
    3. $A \notin XY,\ B \notin XY$ and $R_j[A] \neq \{\}$ (do not generate degenerate candidates)

BINDER algorithm – workflow (n-ary)

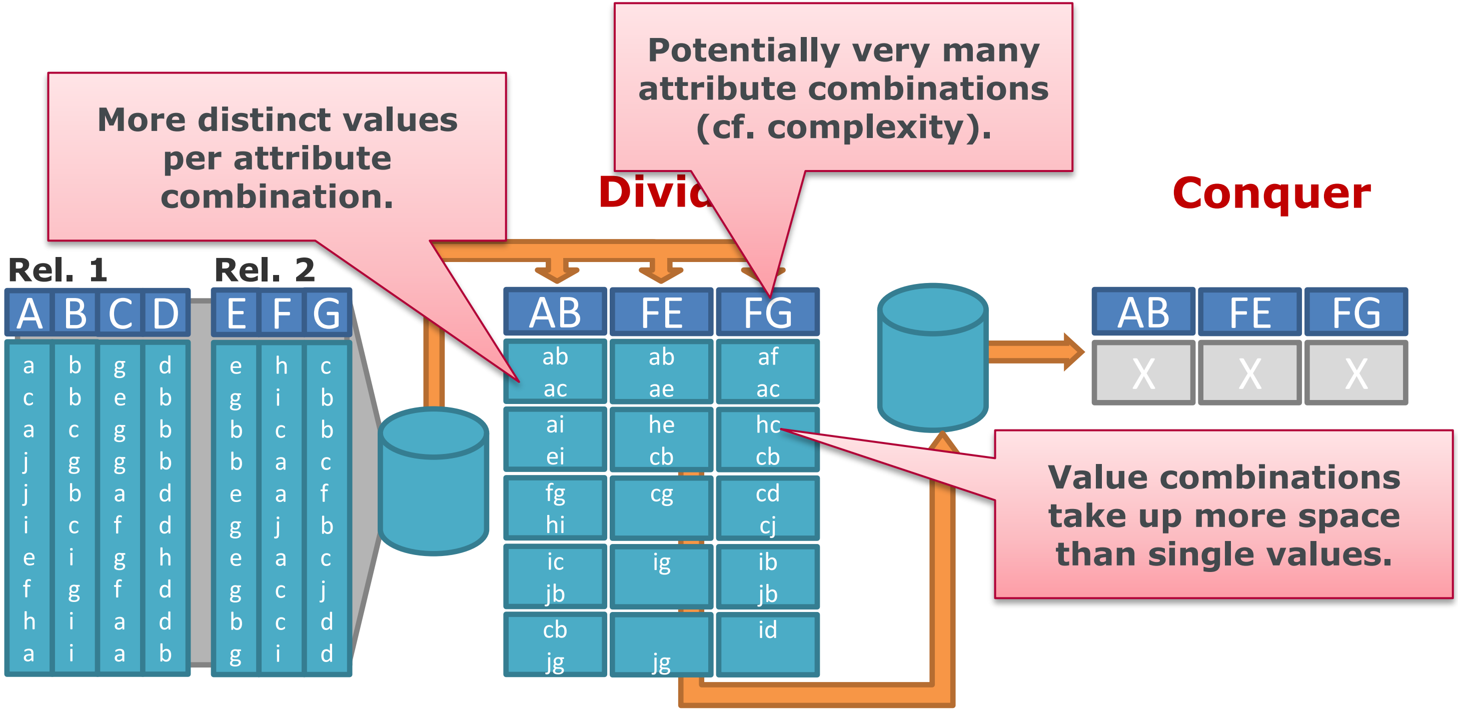
Assume that we need to check AB ⊆ FE and AB ⊆ FG.



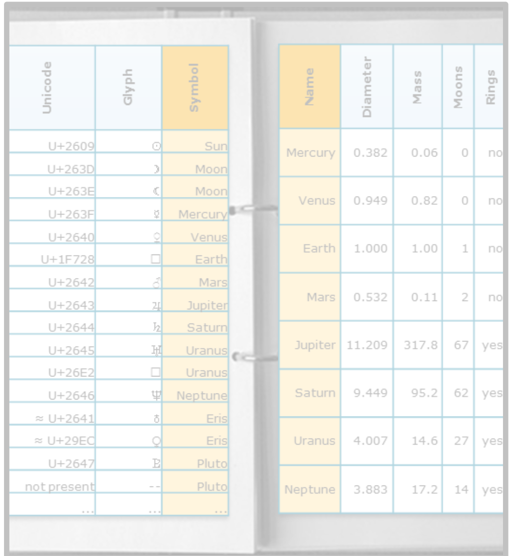
N-ary BINDER evaluation



N-ary BINDER evaluation



Agenda



BINDER
divide & conquer
based IND detection

Sign	House	Domicile	Detriment	Exaltation
Aries	1st House	Mars	Venus	Sur
Taurus	2nd House	Venus	Pluto	Moor
Gemini	3rd House	Mercury	Jupiter	N/A
Cancer	4th House	Moon	Saturn	Jupiter
Leo	5th House	Sun	Uranus	Neptune
Virgo	6th House	Mercury	Neptune	Pluto, Mercury
Libra	7th House	Venus	Mars	Saturn
Scorpio	8th House	Pluto	Venus	Uranus
agittarius	9th House	Jupiter	Mercury	N/A
Capricorn	10th House	Saturn	Moon	Mars

SINDY
scaling out
IND detection

Name	Mass	Orbital radius	Rotation period	Atmosphere
Mercury	0.06	0.47	58.64	minimal
Venus	0.82	0.72	−243.02	CO_2, N_2
Earth	1.00	1.00	1.00	N_2, O_2, Ar
Mars	0.11	1.52	1.03	CO_2, N_2, Ar
Jupiter	317.8	5.20	0.41	H_2, He
Saturn	95.2	9.54	0.43	H_2, He
Uranus	14.6	19.22	−0.72	H_2, He
Neptune	17.2	30.06	0.67	H_2, He

High arity
techniques to deal
with high-arity INDs

The hardware side

- Some problems are intrinsically hard
  - “defeat it with iron”: use more/better hardware for the computation

- **Scalability != efficiency**
  - Efficient = fast / spare resources
  - Scalable = improvement by leveraging more resources

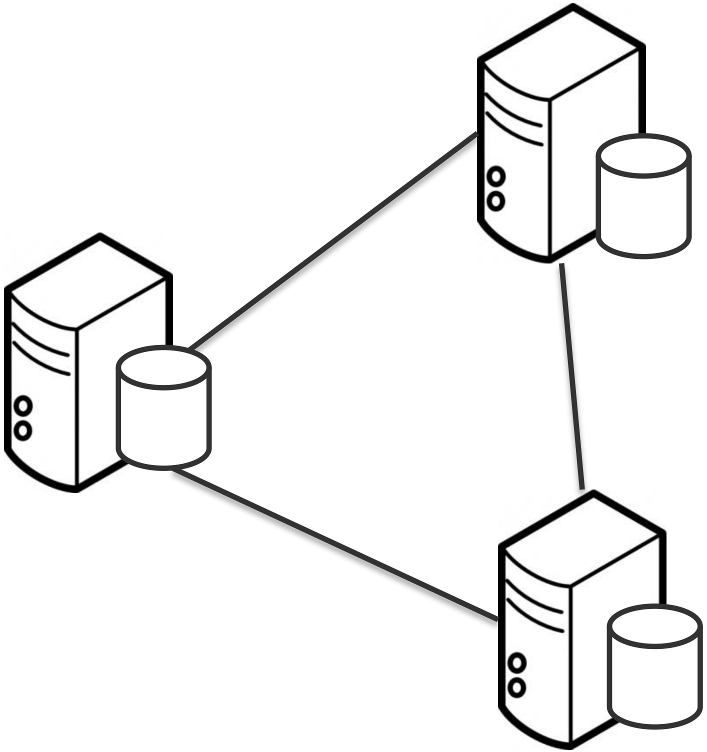
“Do the same work in less time.” ≠ “Do more work in the same time.”

Scaling dimensions

- **Scale up**: faster CPUs/disk, more main memory, …
  - lowest impact on code
  - expensive, limited, shift of bottlenecks
- **Scale in**: more cores, (RAID)
  - thread-level parallelization, cache coherency
  - limited, shift of bottlenecks
- **Scale out**: computer clusters
  - Actors, message passing, data partition
  - Less limited, most complicated

- Is problem suited to certain scaling direction?

Scale out: Setting



- Multiple independent nodes
- can communicate and exchange data
- oftentimes data distributed among nodes

- no shared state
- network new potential bottleneck
  - network topology relevant
- fault tolerance important
- load balancing important

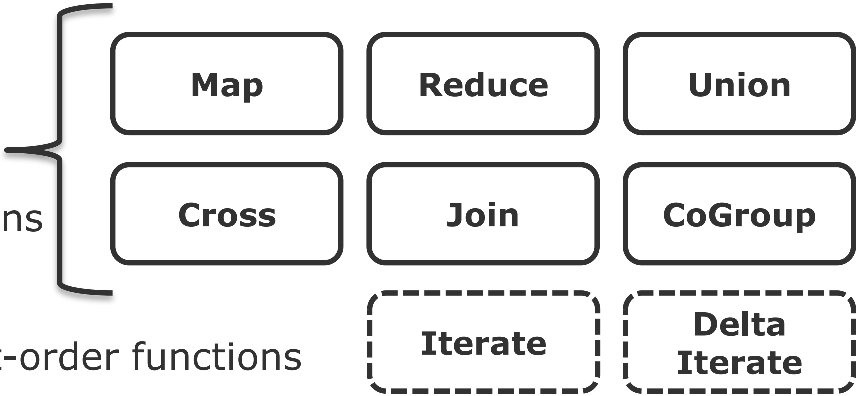
Distributed Application Frameworks

NB: overview not complete!

JAQL	Meteor	Pig	Hive	**Script languages**
Hadoop	Flink/ Stratosphere	Spark	Storm	**Operator-centric frameworks**
YARN	Mesos	Akka	Zookeeper	**Low-level frameworks / resource management**
HDFS	HBase	S3	Cassandra	**Storage**

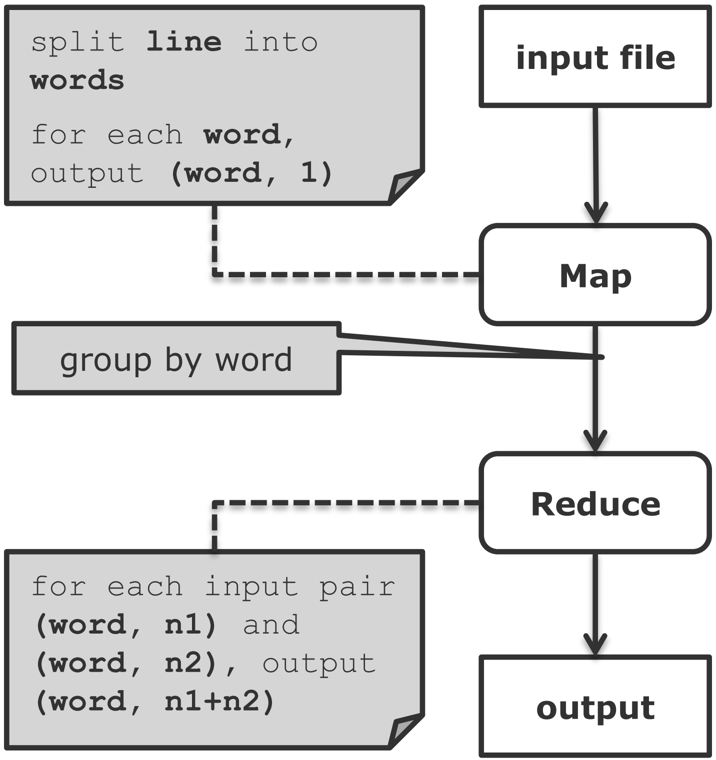
Writing an Apache Flink program

- Four ingredients
  - Datasets
  - First-order functions
    - high-level data operations
  - Second-order functions
    - one passed to each first-order functions
    - refine semantics of first-order functions
    - transform data
  - Directed acyclic graph
    - starts with data sources, ends in data sinks
    - describes workflow

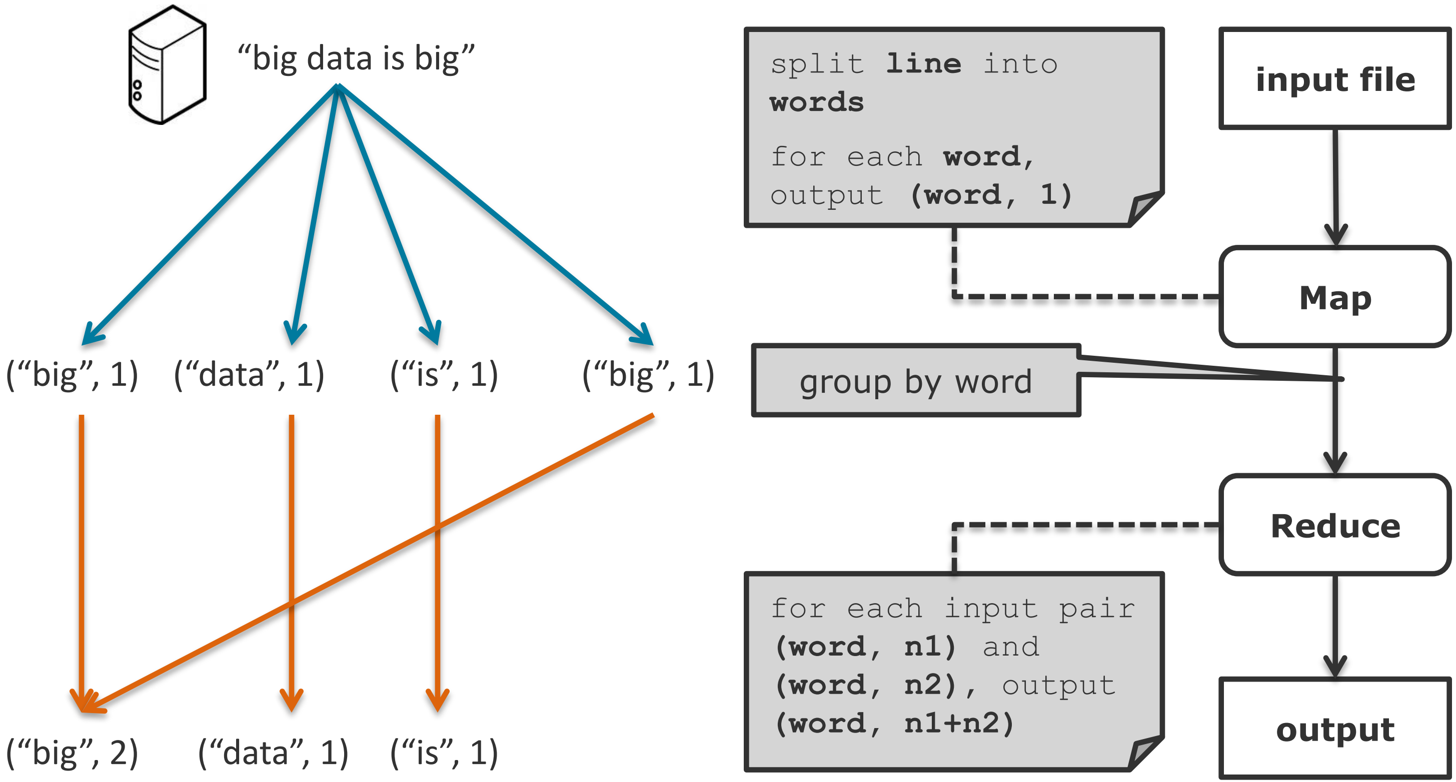


Classic example: word count

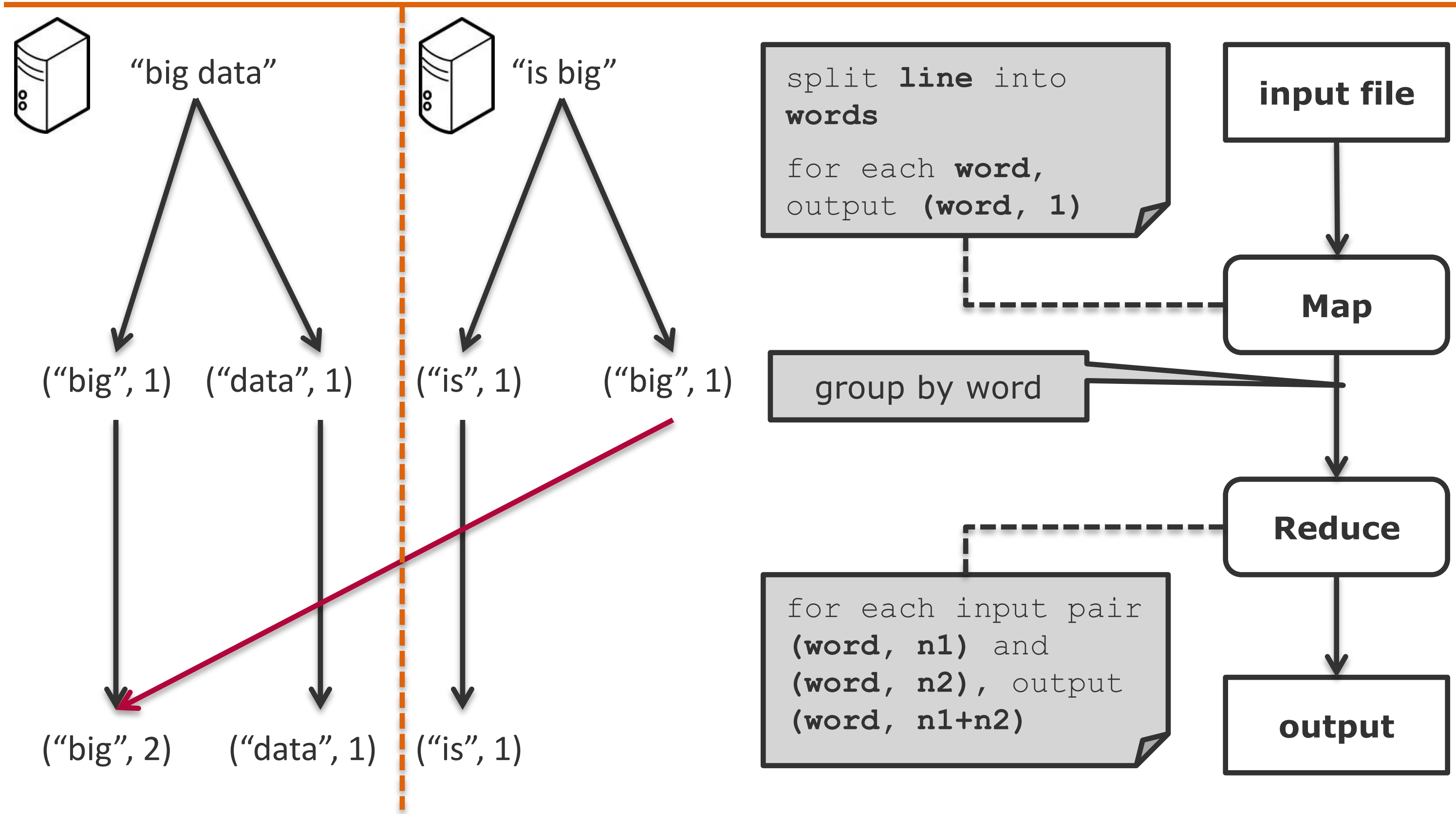
- implemented as DAG with two first-order functions (and two second-order functions)
- specifies
  - operations on a logical level
- does not specify
  - how to parallelize
  - data serialization and shipping
  - handling when available main memory is exceeded
  - fault tolerance
  - …



Classic example: word count

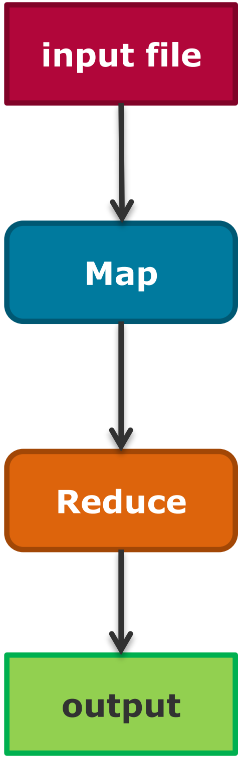


Classic example: word count



Classic example: word count

```
public class CountWords() {
  public static void main(String[] args) {
    ExecutionEnvironment env =
      ExectionEnvironment.getExectutionEnvironment();
    env.readTextFile(args[0])
      .flatMap(
        (String line, Collector<WordCount> out) ->  {
          Arrays.stream(line.split("\\W+"))
          .forEach(t -> out.collect(new WordCount(t, 1)))
        })
      .groupBy("word")
      .sum("count")
      .print();
    env.execute();
  }
}
```



Intrinsic limitations of IND algorithms

- Observations: all IND algorithms follow a common pattern

Algorithm	Phase 1 Data Reorganization	Phase 2 Comparison
De Marchi	Create Inverted Index	Intersect Attribute Groups
SPIDER	Sort Columns	Simultaneous Iteration
BINDER	Partition Columns	In-Memory Partition Comparison

- e.g., IND $A \subseteq B$
  - to prove, need to read A completely
  - to disprove, need to read B completely
- Data reorganization is the most expensive phase
  - I/O-heavy workload, but other phase brings considerable I/O as well

General IND approach

original database

A	B	C	D
a	b	g	d
c	b	e	b
a	c	g	b
j	g	g	b
j	b	a	d
i	c	f	d
e	i	g	h
f	g	f	d
h	i	a	d
a	i	a	b

E	F	G
e	h	c
g	i	b
b	c	b
b	a	c
e	a	f
g	j	b
e	a	c
g	c	j
b	c	d
g	i	d

$$X \subseteq Y \Leftrightarrow \forall x \in X : x \in Y$$

full outer join (w/o duplicates)

A	B	C	D	E	F	G
a	-	a	-	-	a	-
-	b	-	b	b	-	b
c	c	-	-	-	c	c
-	-	-	d	e	-	d
e	-	-	-	-	-	-
f	-	f	-	g	-	f
-	g	g	-	-	-	-
h	-	-	h	-	h	-
i	i	-	-	-	i	-
j	-	-	-	-	j	j

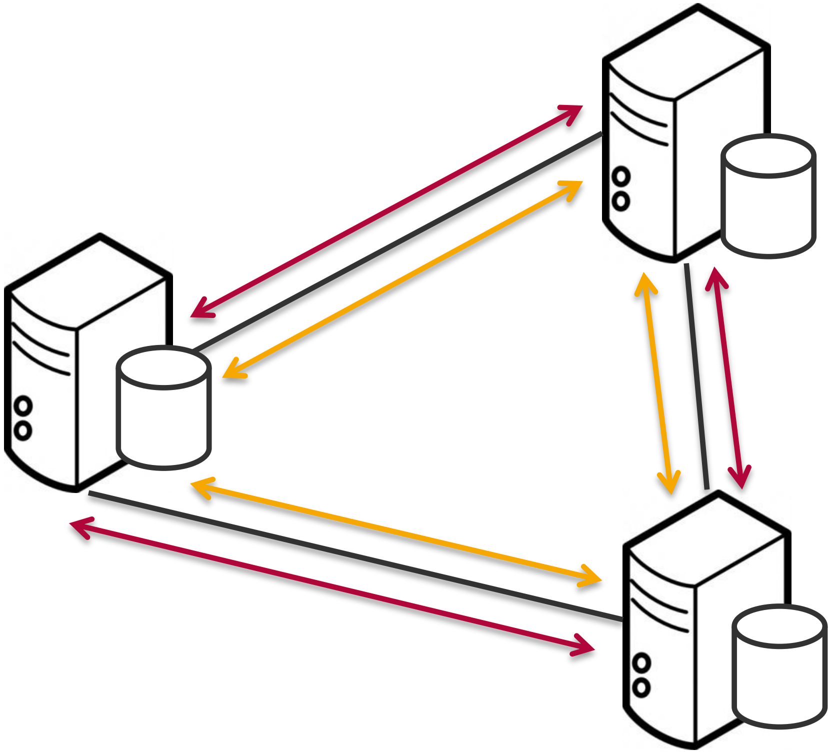
$$X \subseteq Y \Leftrightarrow \forall t \in X \bowtie Y : t[X] \neq \bot \rightarrow t[Y] \neq \bot$$

attribute groups

A	C	F		
B	D	E	G	
A	B	F	G	
D	E	G		
A				
A	E	G		
B	C			
A	B	C	D	F
A	B	F		
A	F	G		

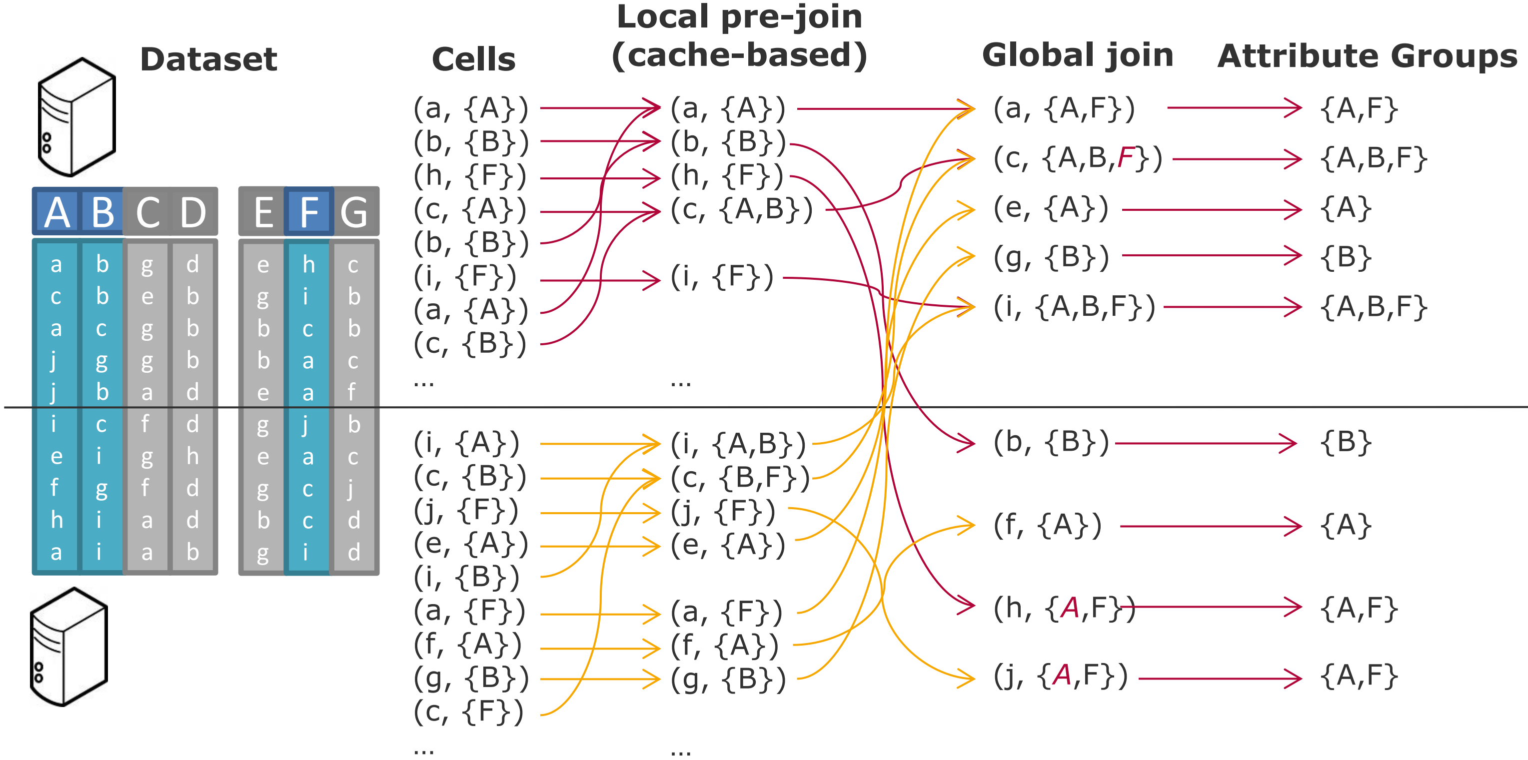
$$X \subseteq Y \Leftrightarrow \forall G \in \mathcal{G} : X \in G \rightarrow Y \in G \Leftrightarrow Y \in \bigcap_{G \in \mathcal{G} | Y \in G} G$$

Distributed IND detection: general idea

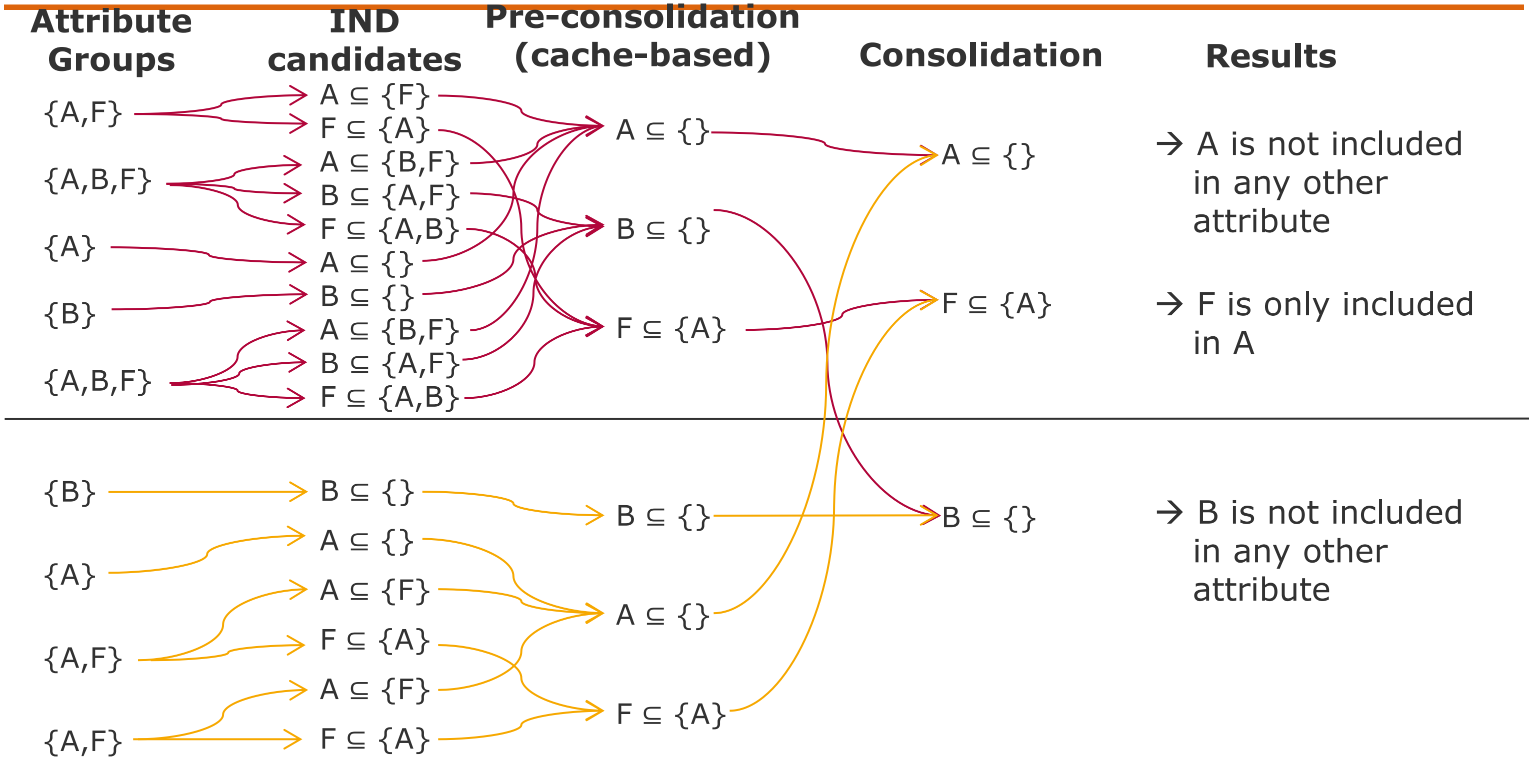


1. Calculate full outer join
2. Intersect attribute groups

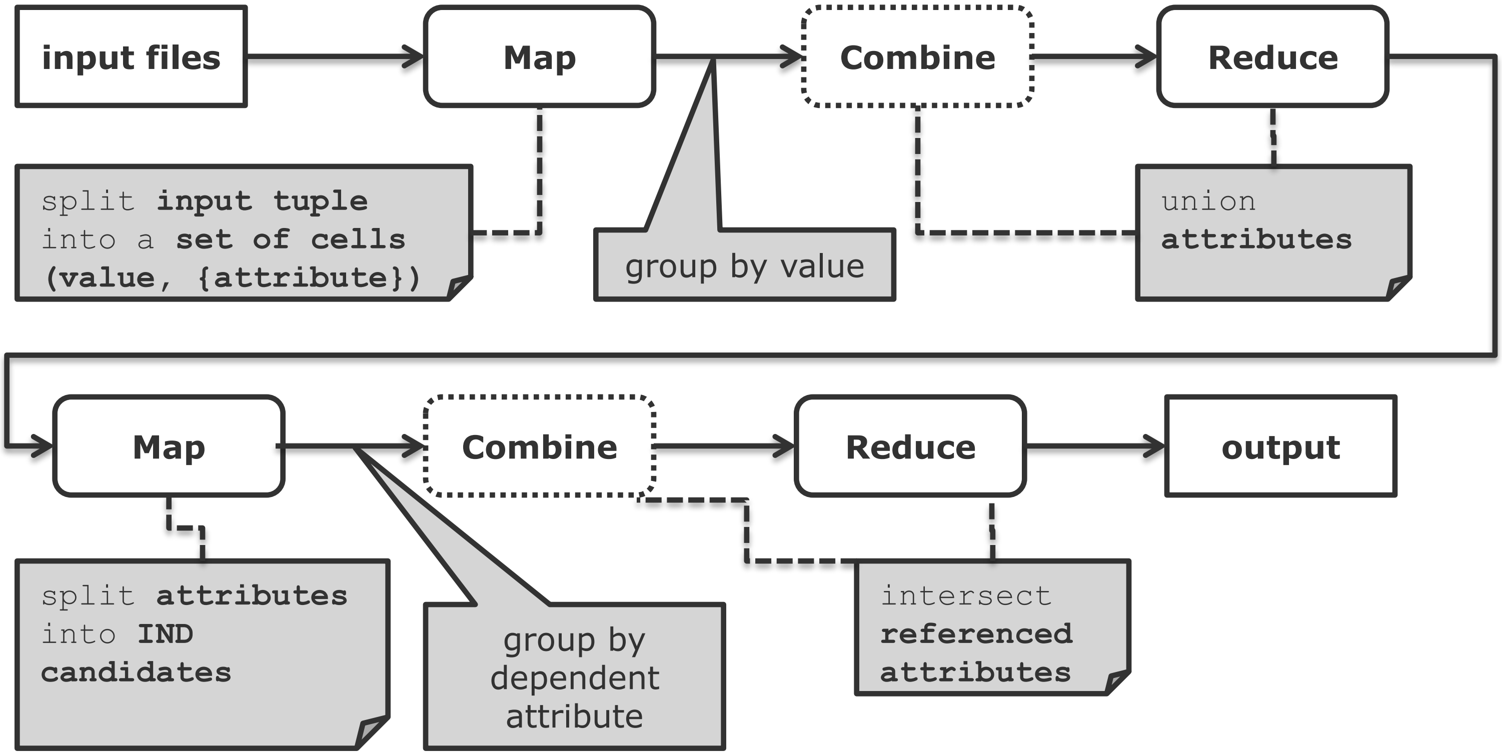
SINDY: Calculate outer join



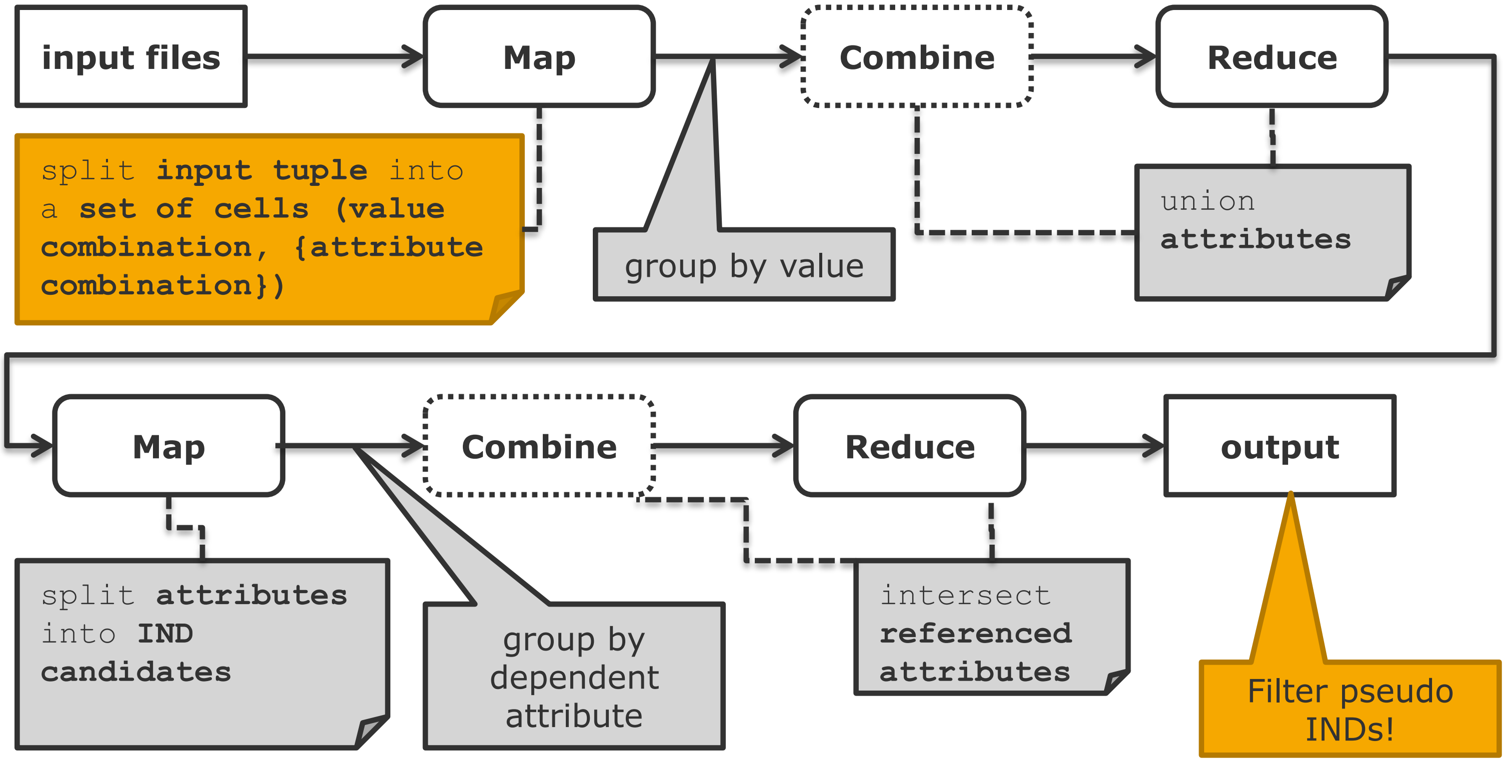
Determine INDs from join result



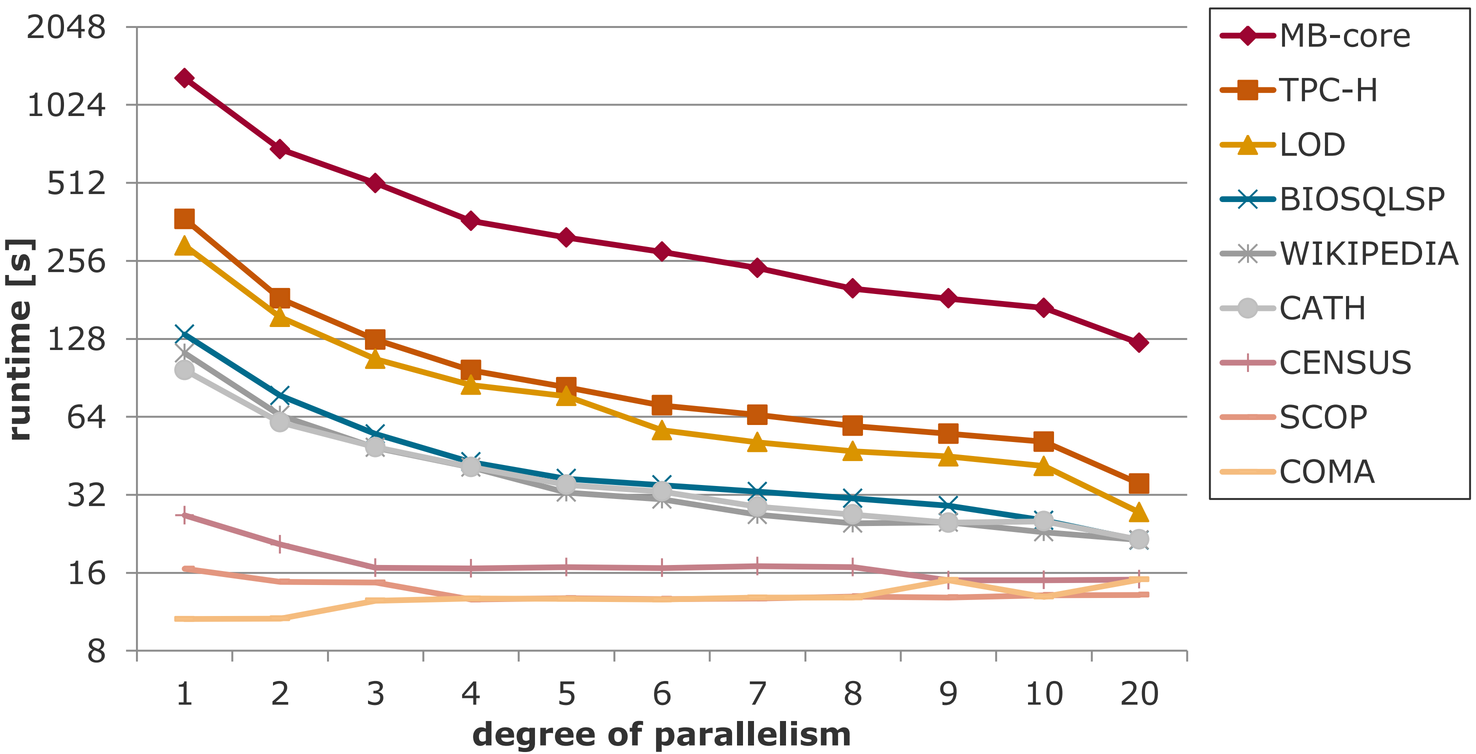
Implementation on Flink (unary INDs)



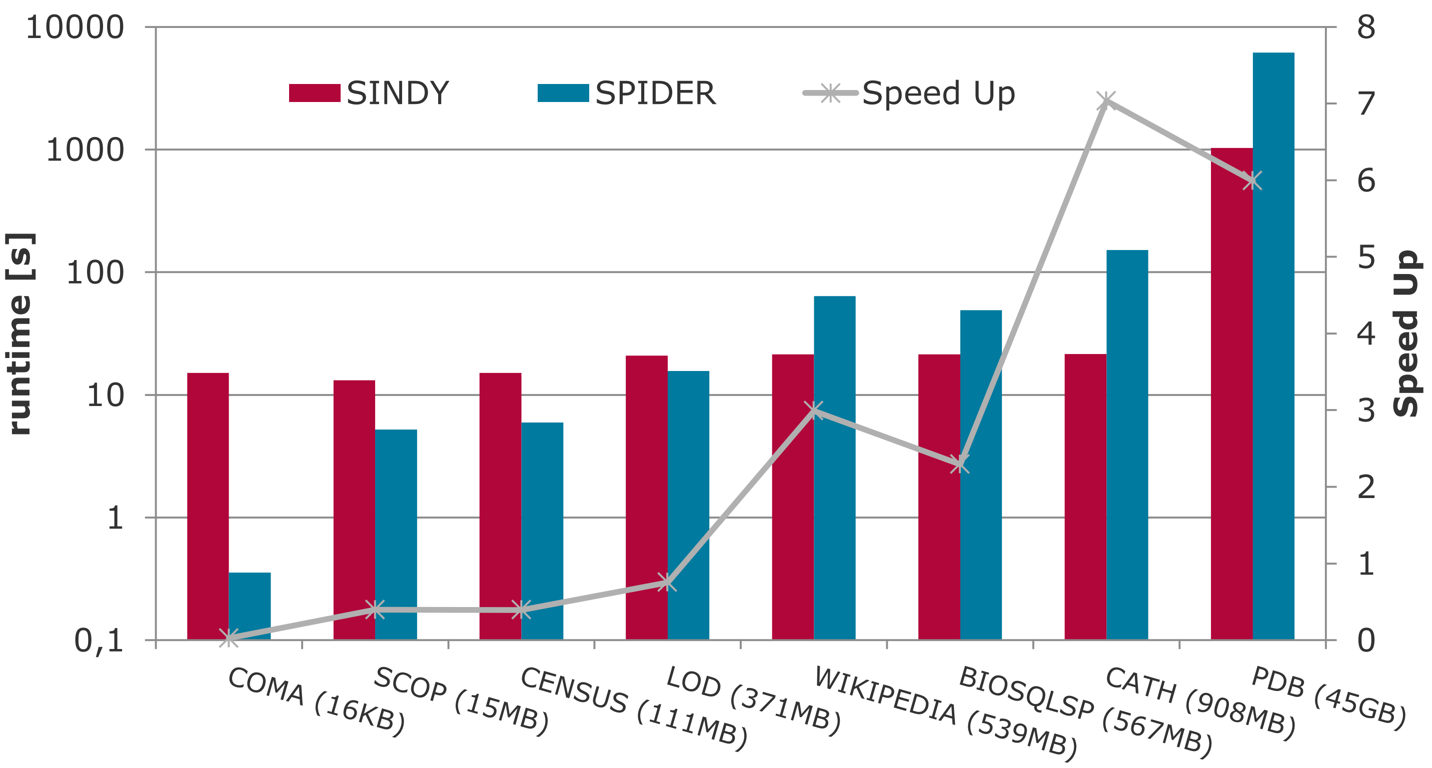
Implementation on Flink (n-ary INDs)



Scale-out behavior



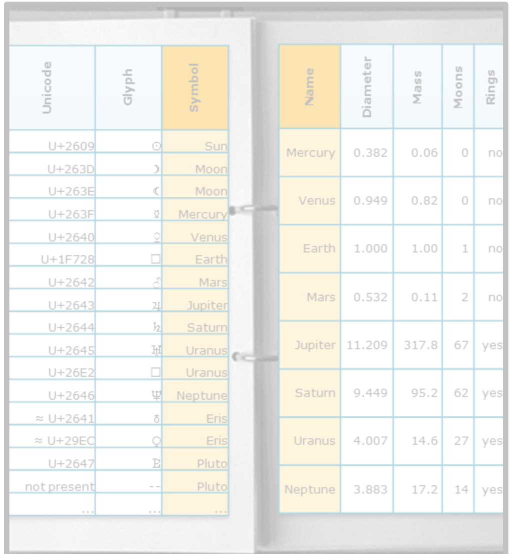
Performance comparison with SPIDER



Conclusions

- Problem of IND detection can be scaled out with the Map/Reduce paradigm
  - Comes with a certain loss of control
- SINDY does not employ pruning (except for apriori proceeding)
  - A general problem for distributed algorithms
  - Not a big issue for IND detection
- Only suitable for large datasets
- Arising questions
  - To what extent is attribute scaling possible? → MANY
  - What if some INDs are n-ary for some larger n?

Agenda



Unicode	Glyph	Symbol
U+2609	☉	Sun
U+263D	☽	Moon
U+263E	☾	Moon
U+263F	☿	Mercury
U+2640	♀	Venus
U+1F728	□	Earth
U+2642	♂	Mars
U+2643	♃	Jupiter
U+2644	♄	Saturn
U+2645	♅	Uranus
U+26E2	□	Uranus
U+2646	♆	Neptune
≈ U+2641	♁	Eris
≈ U+29EC	⧬	Eris
U+2647	♇	Pluto
not present	--	Pluto
...	...	...

Name	Diameter	Mass	Moons	Rings
Mercury	0.382	0.06	0	no
Venus	0.949	0.82	0	no
Earth	1.000	1.00	1	no
Mars	0.532	0.11	2	no
Jupiter	11.209	317.8	67	yes
Saturn	9.449	95.2	62	yes
Uranus	4.007	14.6	27	yes
Neptune	3.883	17.2	14	yes

BINDER
divide & conquer
based IND detection

Sign	House	Domicile	Detriment	Exaltation
Aries	1st House	Mars	Venus	Sur
Taurus	2nd House	Venus	Pluto	Moor
Gemini	3rd House	Mercury	Jupiter	N/A
Cancer	4th House	Moon	Saturn	Jupiter
Leo	5th House	Sun	Uranus	Neptune
Virgo	6th House	Mercury	Neptune	Pluto, Mercury
Libra	7th House	Venus	Mars	Saturn
Scorpio	8th House	Pluto	Venus	Uranus
agittarius	9th House	Jupiter	Mercury	N/A
Capricorn	10th House	Saturn	Moon	Mars

SINDY
scaling out
IND detection

Name	Mass	Orbital radius	Rotation period	Atmosphere
Mercury	0.06	0.47	58.64	minimal
Venus	0.82	0.72	−243.02	CO_2, N_2
Earth	1.00	1.00	1.00	N_2, O_2, Ar
Mars	0.11	1.52	1.03	CO_2, N_2, Ar
Jupiter	317.8	5.20	0.41	H_2, He
Saturn	95.2	9.54	0.43	H_2, He
Uranus	14.6	19.22	−0.72	H_2, He
Neptune	17.2	30.06	0.67	H_2, He

High arity
techniques to deal
with high-arity INDs

Motivation

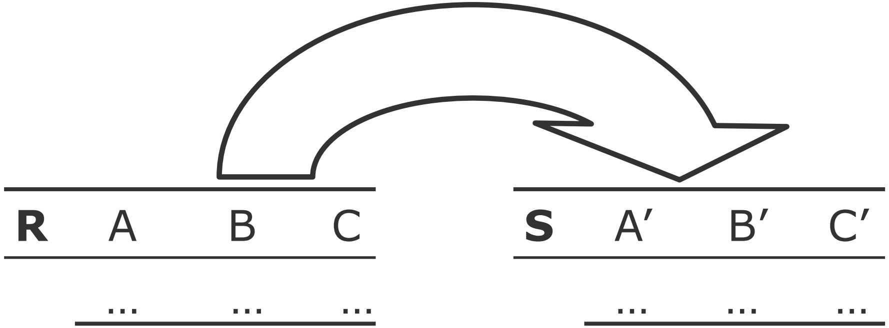
- High-arity INDs do not go well together with apriori-based algorithms
  - Consider an IND of arity *n*
  - Then there are 2^n-2 sub-INDs to be verified
  - No pruning possible
  - Recall the hardness of *n*-ary IND discovery

- Different approaches necessary
  - Cf. TANE and HyFD

Motivation

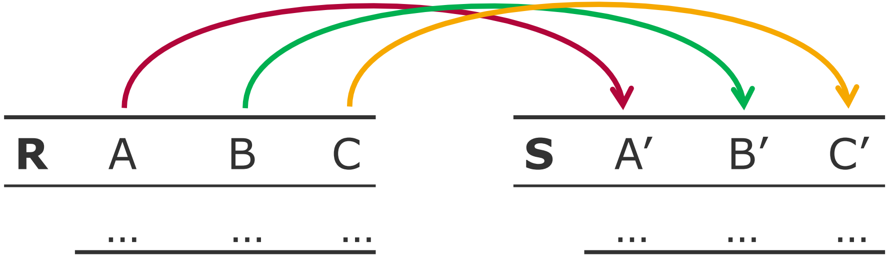
- Most (maximal) INDs are of low arity, but we do find high-arity INDs when…

A different notation for n-ary INDs



$$R[A, B, C] \subseteq S[A', B', C']$$

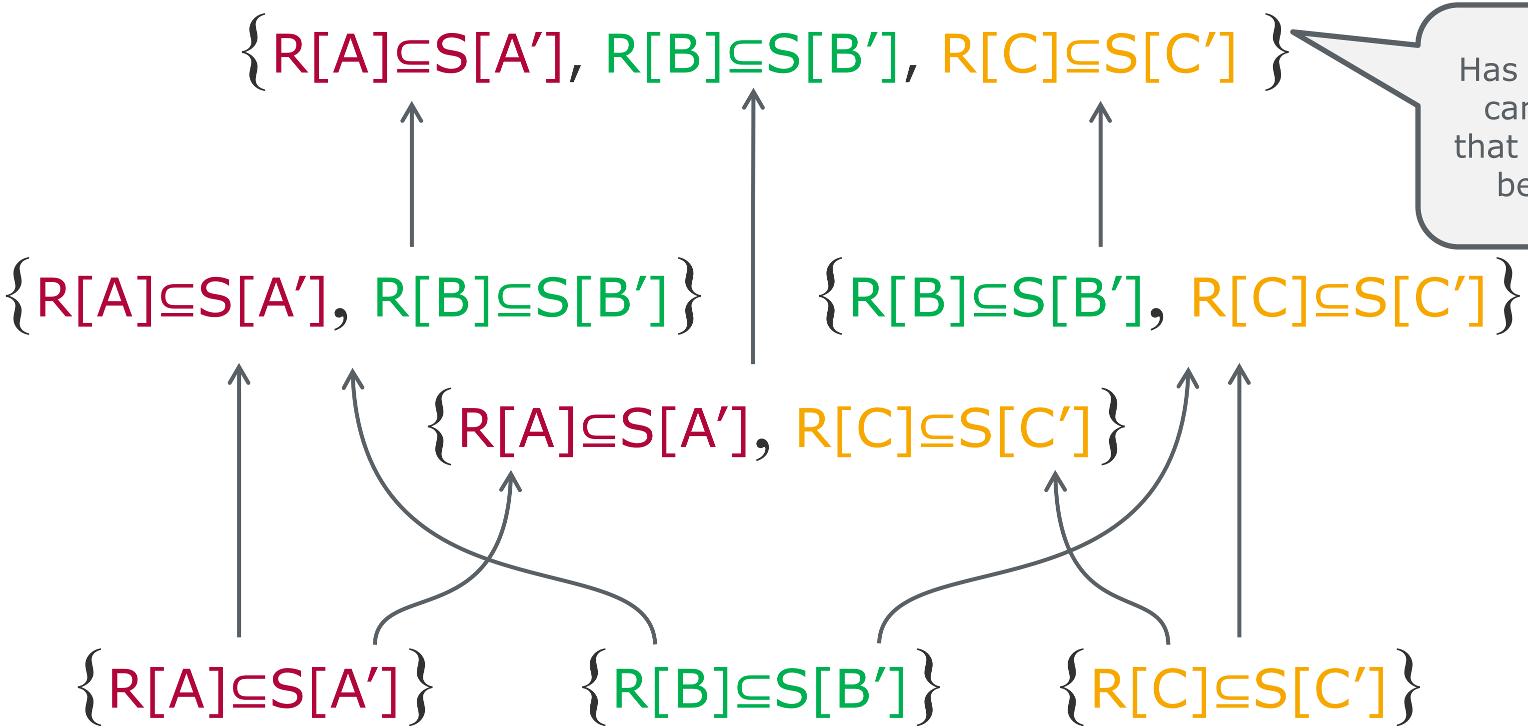
A different notation for n-ary INDs



$$\text{ens}(\,R[A, B, C] \subseteq S[A', B', C']\,) =$$

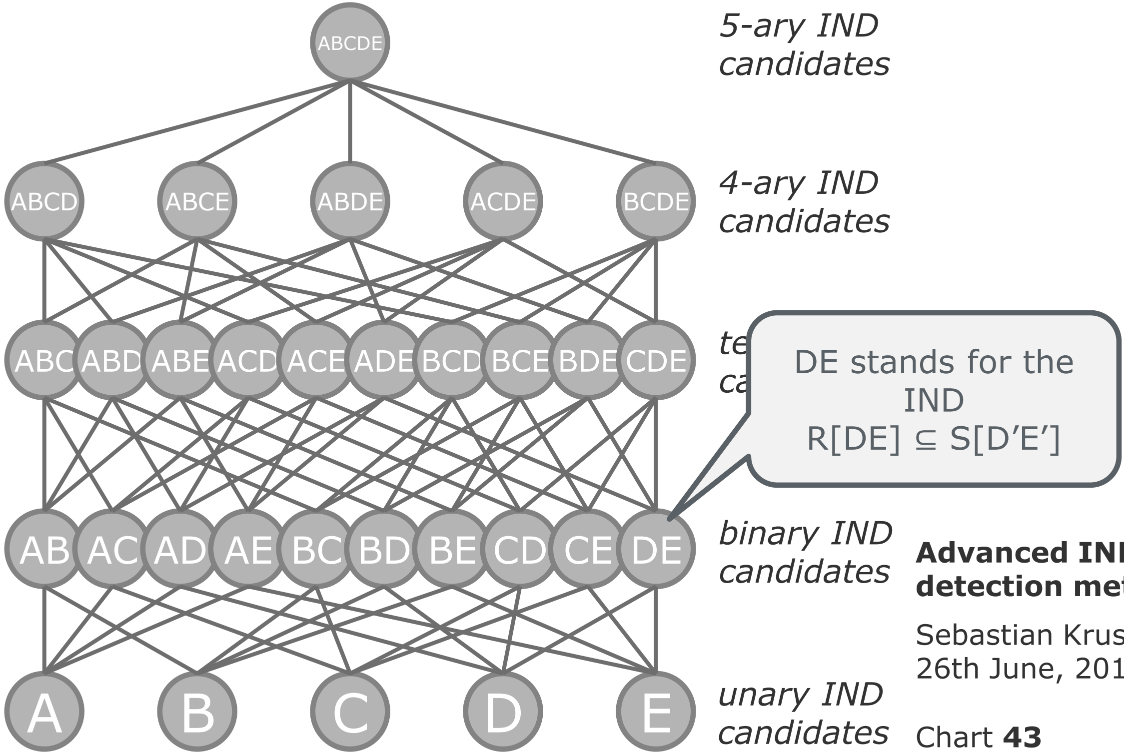
$$\{R[A] \subseteq S[A'],\ R[B] \subseteq S[B'],\ R[C] \subseteq S[C']\}$$

A different notation for n-ary INDs

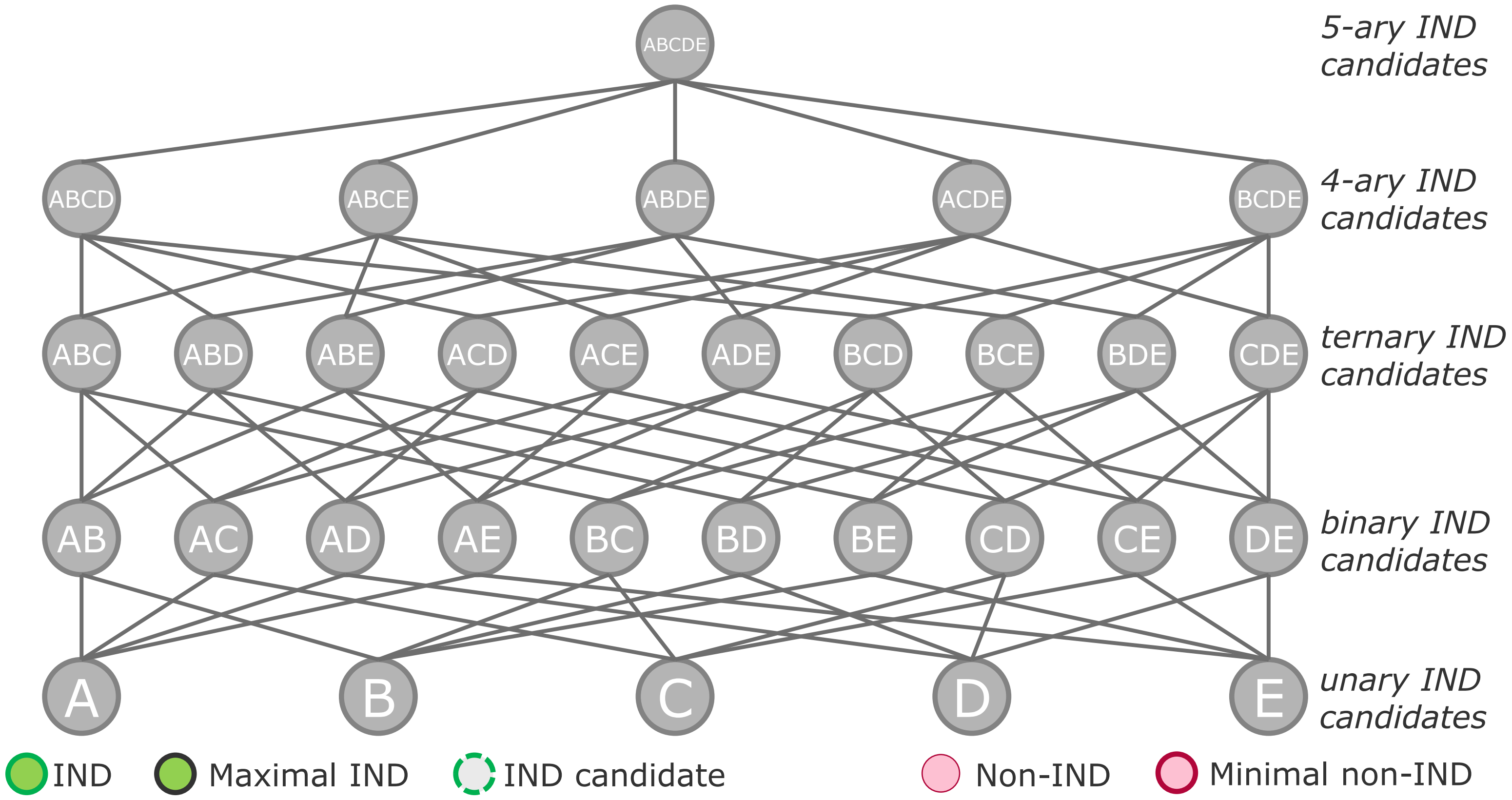


Has 2^n-2 "sub-candidates" that all need to be tested.

Optimistic and pessimistic strategies

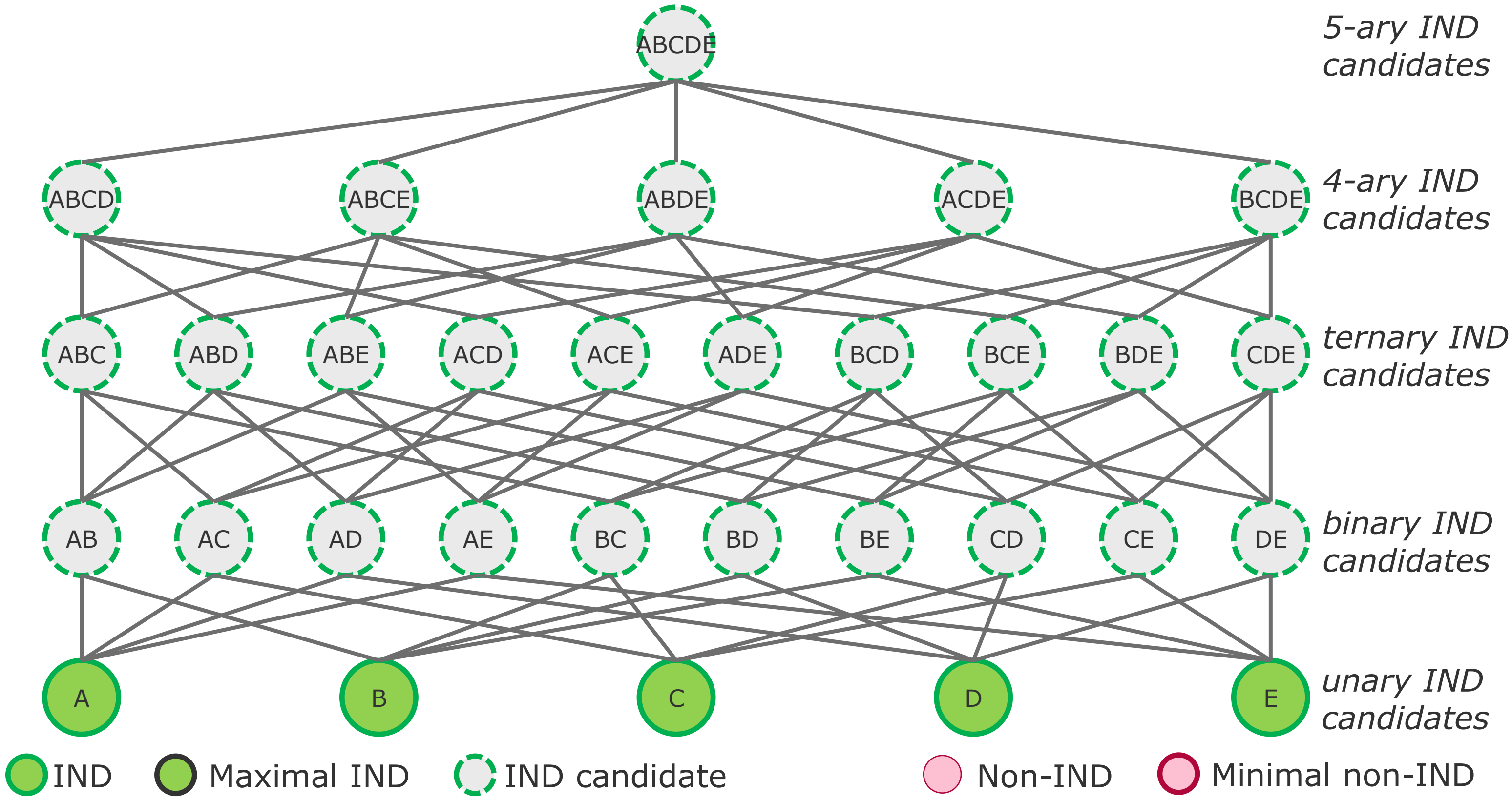


IND borders

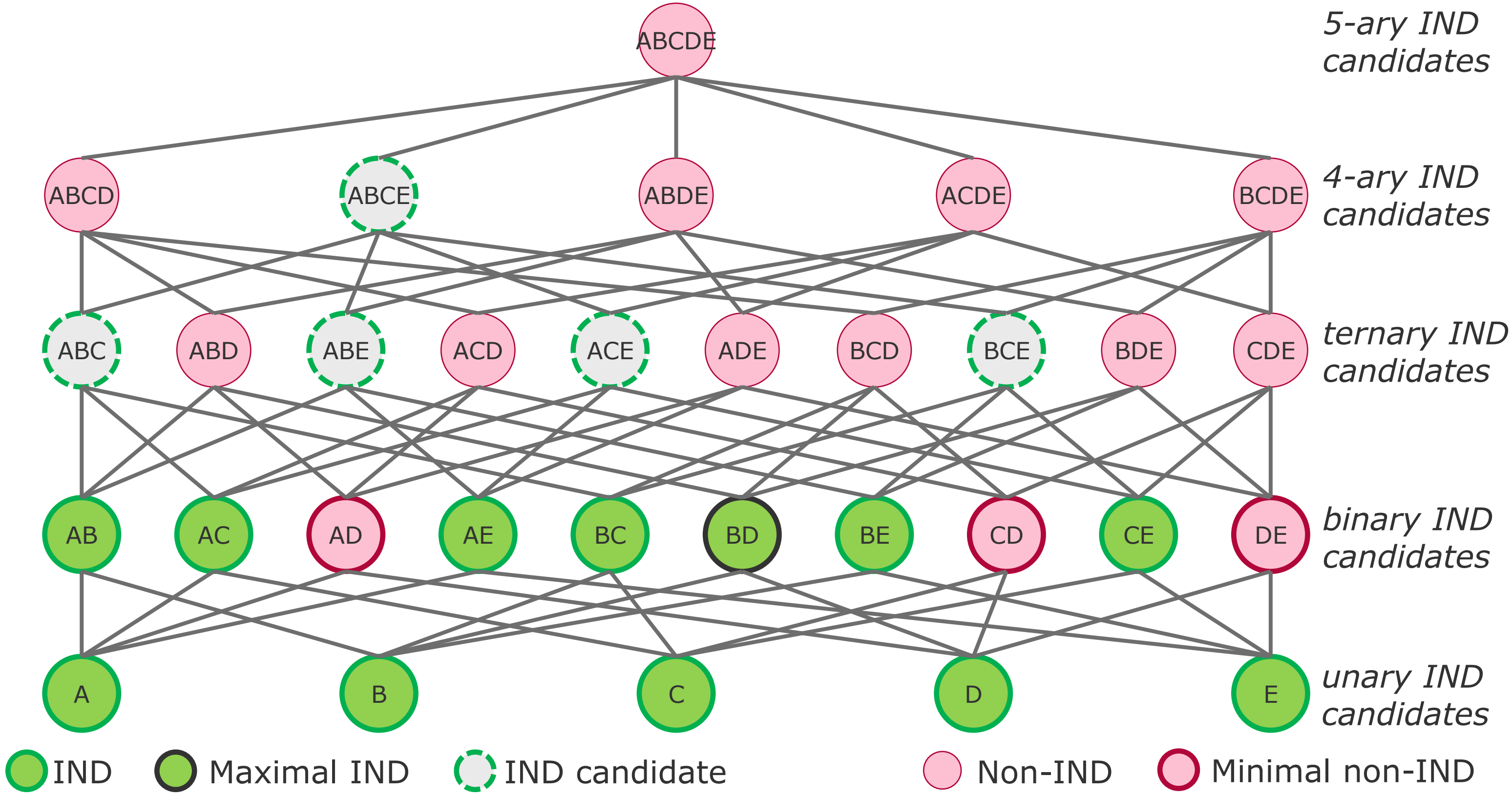

ABCDE
5-ary IND candidates
ABCD
ABCE
ABDE
ACDE
BCDE
4-ary IND candidates
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
ternary IND candidates
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
binary IND candidates
A
B
C
D
E
unary IND candidates
IND
Maximal IND
IND candidate
Non-IND
Minimal non-IND

High-arity INDs - ZigZag

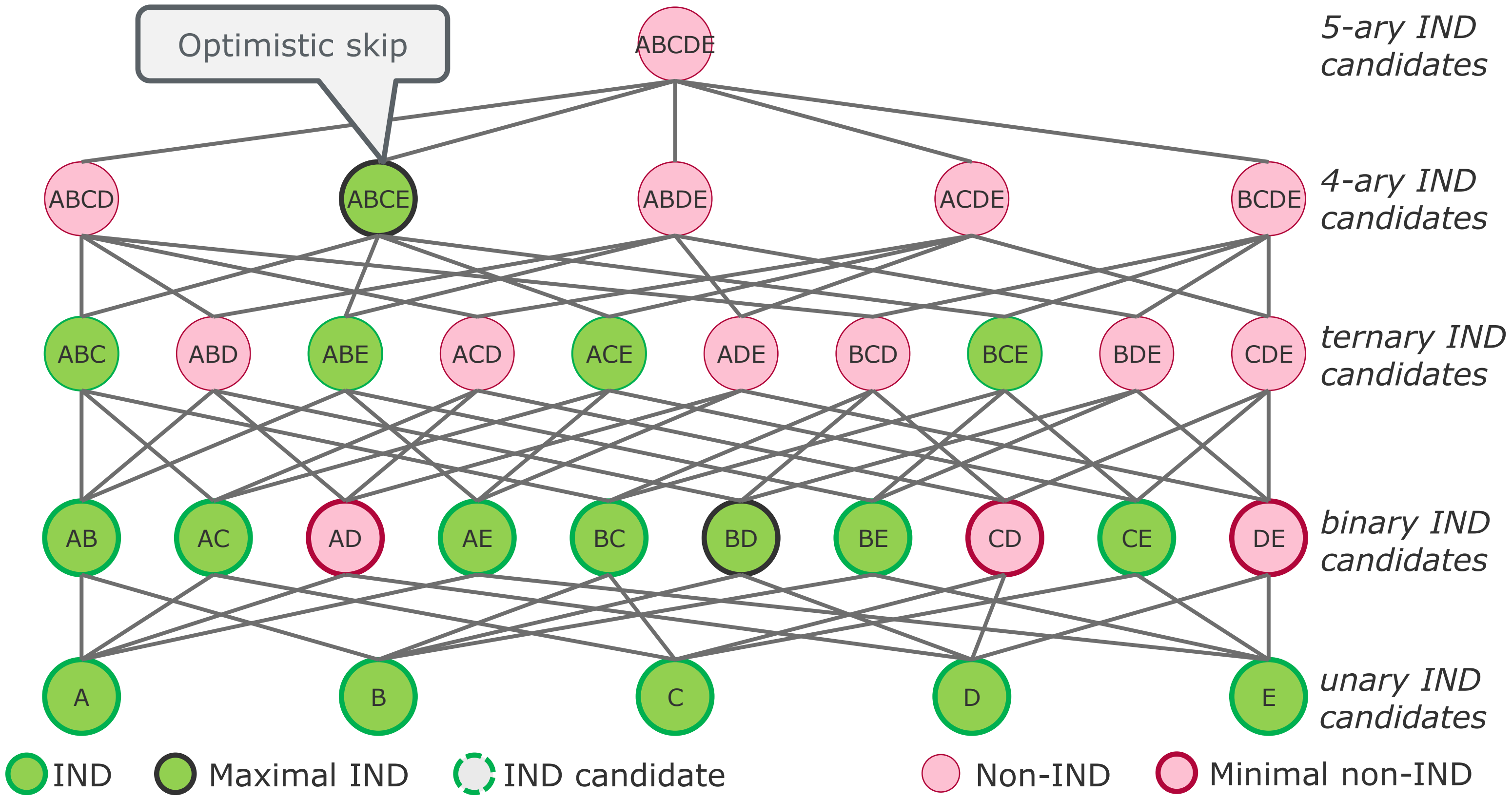
IND borders



IND borders


ABCDE
ABCD
ABCE
ABDE
ACDE
BCDE
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
5-ary IND candidates
4-ary IND candidates
ternary IND candidates
binary IND candidates
unary IND candidates
IND
Maximal IND
IND candidate
Non-IND
Minimal non-IND

IND borders


Optimistic skip
ABCDE
ABCD
ABCE
ABDE
ACDE
BCDE
ABC
ABD
ABE
ACD
ACE
ADE
BCD
BCE
BDE
CDE
AB
AC
AD
AE
BC
BD
BE
CD
CE
DE
A
B
C
D
E
5-ary IND candidates
4-ary IND candidates
ternary IND candidates
binary IND candidates
unary IND candidates
IND
Maximal IND
IND candidate
Non-IND
Minimal non-IND

Calculate the optimistic IND border

- Given two tables, and a set of known INDs and/or non-INDs, how can we determine the *optimistic IND border*?
  - All IND candidates that are (i) not known non-INDs and (ii) are maximal w.r.t. property (i).

- $FIND_2$
  - Determine hypergraph cliques based on INDs

- ZigZag
  - Determine hitting sets based on non-INDs

Calculate the optimistic IND border

- Example:
  - Unary INDs = {A, B, C, D, E}
  - INDs = {AB, AC, AE, BC, BD, BE, CE}
  - non-INDs = {AD, CD, DE}
- Goal:
  - find all maximal sets ⊆ ABCDE that are no supersets of AD, CD, or DE
- General strategy
  - Determine minimal sets that intersect with all non-INDs (ACE, D)
  - Remove these minimal sets from ABCDE (BD, ABCE)

Calculate the optimistic IND border (simple)

- **Input:** non-INDs N

N=[AD, CD, DE]

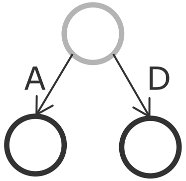
Calculate the optimistic IND border (adv.)

- **Input:** non-INDs N, unary INDs U   N=[AD, CD, DE], U={A, B, C, D, E}

Calculate the optimistic IND border (adv.)

- Current state

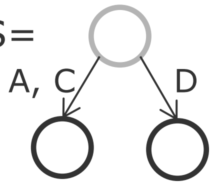
N=[CD, DE], S=



Calculate the optimistic IND border (adv.)

- Current state

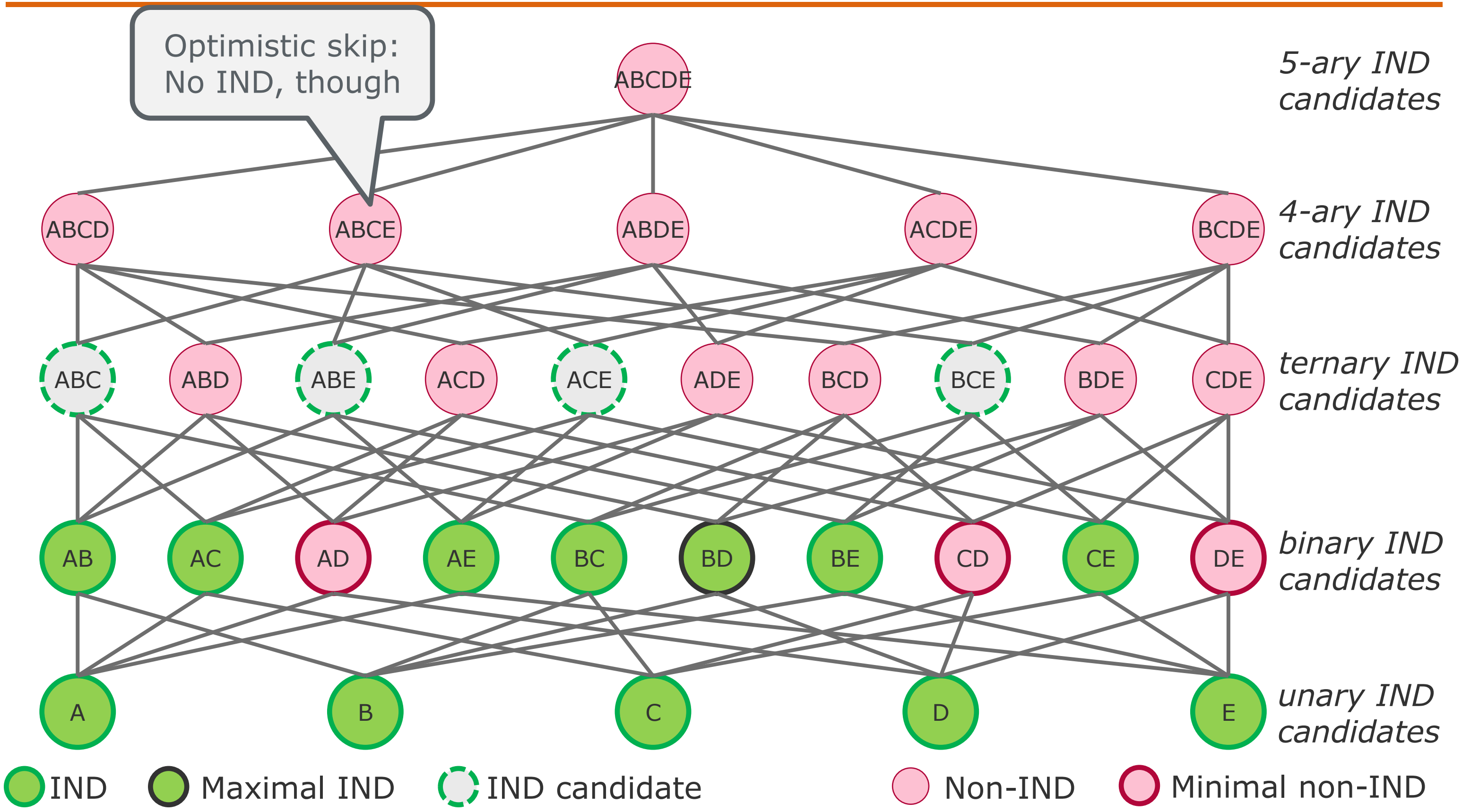
N=[DE], S=



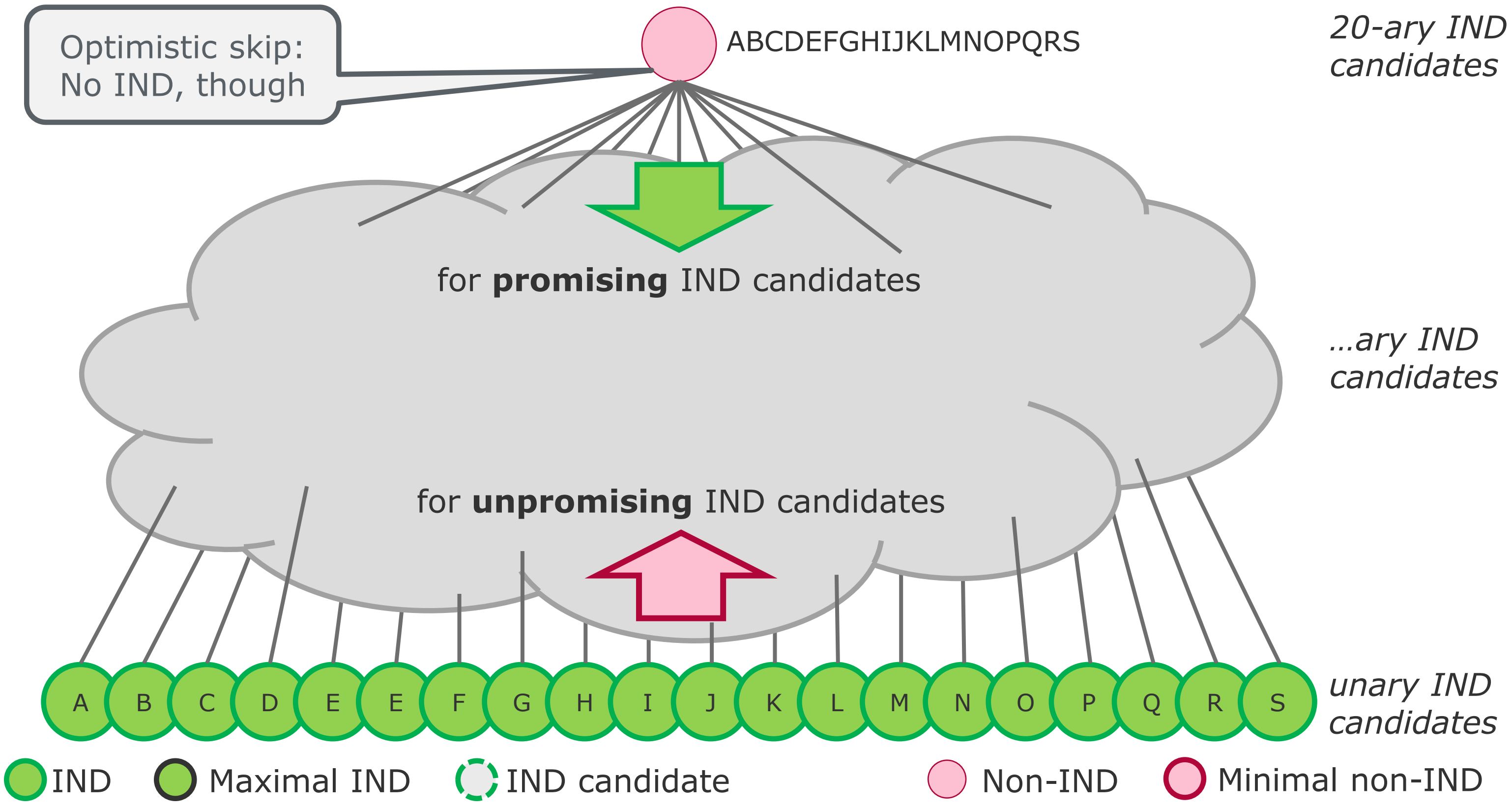
Calculate the optimistic IND border (adv.)

- For the attentive student:
  (6) update S with all elements in L except a subset is already in S
  - What if an existing solution is a superset of a new solution?
- This is not possible (inductive proof):
  - Assume, we introduced a new element N that is a subset of some existing element E in S
  - Then we would have obtained N from some N'=N\I for some unary IND in U.
  - Hence, S must have been already in an inconsistent state.
  - Initially, S={∅}, which is a consistent state.
  - What about two elements being added in a single iteration, though?
    → Figure out yourselves.

Non-INDs in the optimistic IND border



Non-INDs in the optimistic IND border



Non-INDs in the optimistic IND border

- Promising IND candidate $R[X] \subseteq S[Y]$ if $g'_3(R[X] \subseteq S[Y]) \leq \varepsilon$
- $g'_3(R[X] \subseteq S[Y])$: proportion of distinct values in $R[X]$ to be removed, such that $R[X] \subseteq S[Y]$ is a valid IND
  - Alternative: proportion of tuples to be removed from R
- Example: $g'_3(R[Planet] \subseteq S[Planet]) = 1 / 10 = 0.1$
  - The value "Ceres" has to be removed out of 10 distinct values

R

Planet	Mean distance	Relative mean distance
Mercury	57.91	1
Venus	108.21	1.86859
Earth	149.6	1.3825
Mars	227.92	1.52353
Ceres	413.79	1.81552
Jupiter	778.57	1.88154
Saturn	1,433.53	1.84123
Uranus	2,872.46	2.00377
Neptune	4,495.06	1.56488
Pluto	5,869.66	1.3058

S

Planet	Calculated (in AU)	Observed (in AU)	Perfect octaves	Actual distance
Mercury	0.4	0.387	0	0
Venus	0.7	0.723	1	1.1
Earth	1	1	2	2
Mars	1.6	1.524	4	3.7
Asteroid belt	2.8	2.767	8	7.8
Jupiter	5.2	5.203	16	15.7
Saturn	10	9.539	32	29.9
Uranus	19.6	19.191	64	61.4
Neptune	38.8	30.061	96	-96.8
Pluto	77.2	39.529	128	127.7

Non-INDs in the optimistic IND border

1. **Input:** tables R and S, pessimistic levels k, promising error ε
2. Calculate lower k levels with a pessimistic approach (e.g., BINDER)
3. Calculate optimistic IND border from non-INDs
4. **For each** IND candidate I in the optimistic IND border
   1. Calculate error $g'_3(I)$ of I
   2. **If** $g'_3(I) = 0$ **then output** IND I
   3. **Else if** $g'_3(I) \leq \varepsilon$ **then** traverse lattice top-down breadth-first from I
   4. **Else** add all $k+1$-ary parent IND candidates of I to the pessimistic IND candidates
5. Check all pessimistic IND candidates
6. **If** there are open IND candidates, set k=k+1 and start over with **step 3**

Practical considerations

- A strategy for handling more than two tables is missing
- Several optimizations are possible, e.g., not all kinds of unary INDs can be combined to valid n-ary IND candidates (attribute repetition)
- Empirical evidence on the actual advantages of optimistic IND discovery is missing
  → Thorough evaluation all IND algorithms is called for!
- The original article on ZigZag proposes to do use SQL-based error checks for *n*-ary INDs (cf. MIND)
  → the traversal strategy, however, is orthogonal to IND error checks
  → more efficient techniques, such as those of BINDER and SINDY, could be used instead

FDs cause high-arity INDs

- Consider following dependencies: *[ZIP′, City′]* ⊆ *[ZIP, City]* and *ZIP* → *City*
- **Rule 1)** Then *ZIP′* → *City′* is also a valid FD.
  - Because FDs cannot be violated by removing tuples.
- Additionally, consider *[Name′, ZIP′]* ⊆ *[Name, ZIP]*
- **Rule 2)** Then *[Name′, ZIP′, City′]* ⊆ *[Name, ZIP, City]* is an IND.
  - Because if *t[ZIP]* = *t[ZIP′]*, then *t[City]* = *t[City′]*.

Name	Zip	City
Tim	10627	Berlin
Tom	10627	Berlin
Tom	14482	Potsdam
Sandy	10324	Berlin
Inge	14469	Potsdam

Students

Name′	Zip′	City′
Tim	10627	Berlin
Tom	10627	Berlin
Inge	14469	Potsdam

HPI Students

Augmentation rules

- Idea: split INDs into "core" INDs and "augmentation rules"
  - IND: *[Name', ZIP'] ⊆ [Name, ZIP]*
  - AR: *[ZIP'] ⊆ [ZIP] → [City'] ⊆ [City]*
- Separates INDs into core and supplemental INDs
  - Useful for foreign key discovery and understanding
- Potentially reduces the size of the result set
  - Speed up discovery and make results more manageable

Name	Zip	City
Tim	10627	Berlin
Tom	10627	Berlin
Tom	14482	Potsdam
Sandy	10324	Berlin
Inge	14469	Potsdam

Students

Name'	Zip'	City'
Tim	10627	Berlin
Tom	10627	Berlin
Inge	14469	Potsdam

HPI Students

Augmentation rule discovery

- Assume, we know *[ZIP’] ⊆ [ZIP]* and *[ZIP’, City’] ⊆ [ZIP, City]*
  - Problem: we need to know if *ZIP → City* is an FD to tell if *[ZIP’] ⊆ [ZIP] → [City’] ⊆ [City]* is an AR.
- Solution 1: Discover FDs beforehand (e.g., with HyFD).
- Solution 2: Check relevant FD candidates on-the-fly.
  - *ZIP → City* ↔ $|\pi(ZIP)| = |\pi(ZIP, City)|$ (cf. TANE)
  - We have to group our data anyways, so we can “piggyback” the counting of distinct values at little extra cost.

Name	Zip	City
Tim	10627	Berlin
Tom	10627	Berlin
Tom	14482	Potsdam
Sandy	10324	Berlin
Inge	14469	Potsdam

Students

Name’	Zip’	City’
Tim	10627	Berlin
Tom	10627	Berlin
Inge	14469	Potsdam

HPI Students

Augmentation rule discovery

- Special case: columns/INDs with only a single value
  - *[Status']* ⊆ *[Status]* is valid
  - *X* → *Status* is a valid AR for any column *X* in *Students*
- {} → *[Status']* ⊆ *[Status]* is a valid AR
- This is a very frequent case
  - Empty or constant columns can be found in many databases
  - They are highly susceptible to form *n*-ary INDs

Name	Zip	City	Status
Tim	10627	Berlin	Student
Tom	10627	Berlin	Student
Tom	14482	Potsdam	Student
Sandy	10324	Berlin	Student
Inge	14469	Potsdam	Student

Students

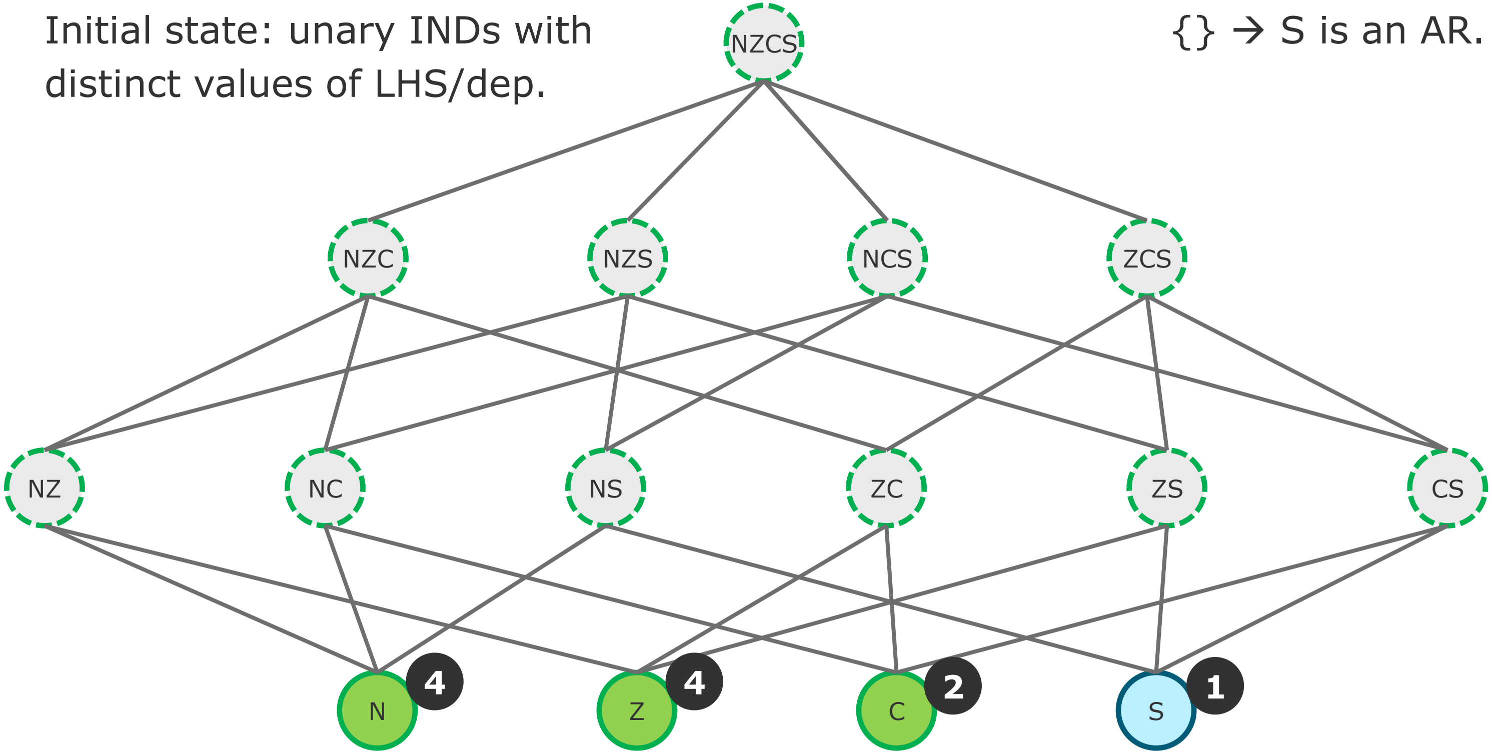
Name'	Zip'	City'	Status'
Tim	10627	Berlin	Student
Tom	10627	Berlin	Student
Inge	14469	Potsdam	Student

HPI Students

Mixed IND and AR discovery

Initial state: unary INDs with distinct values of LHS/dep.

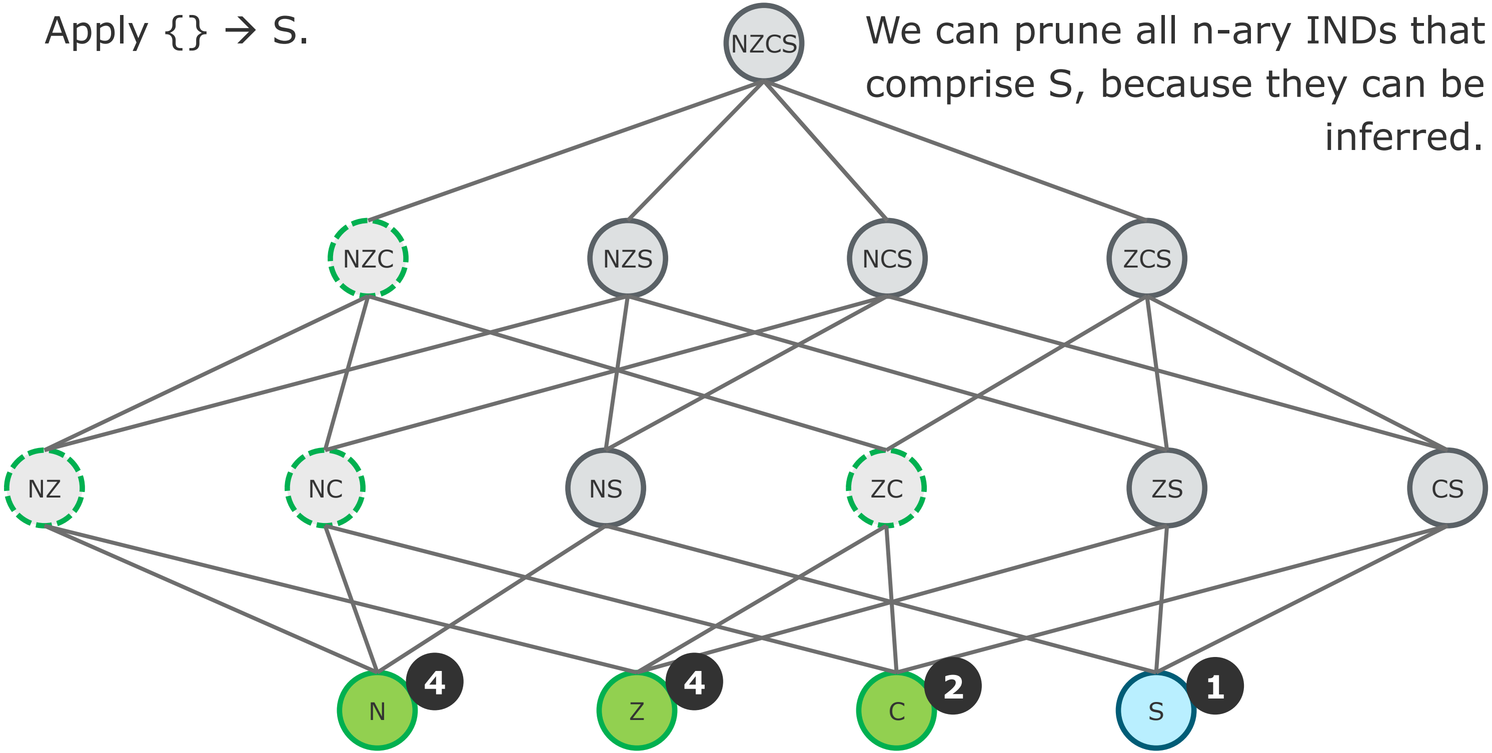
{} → S is an AR.



Mixed IND and AR discovery

Apply {} → S.

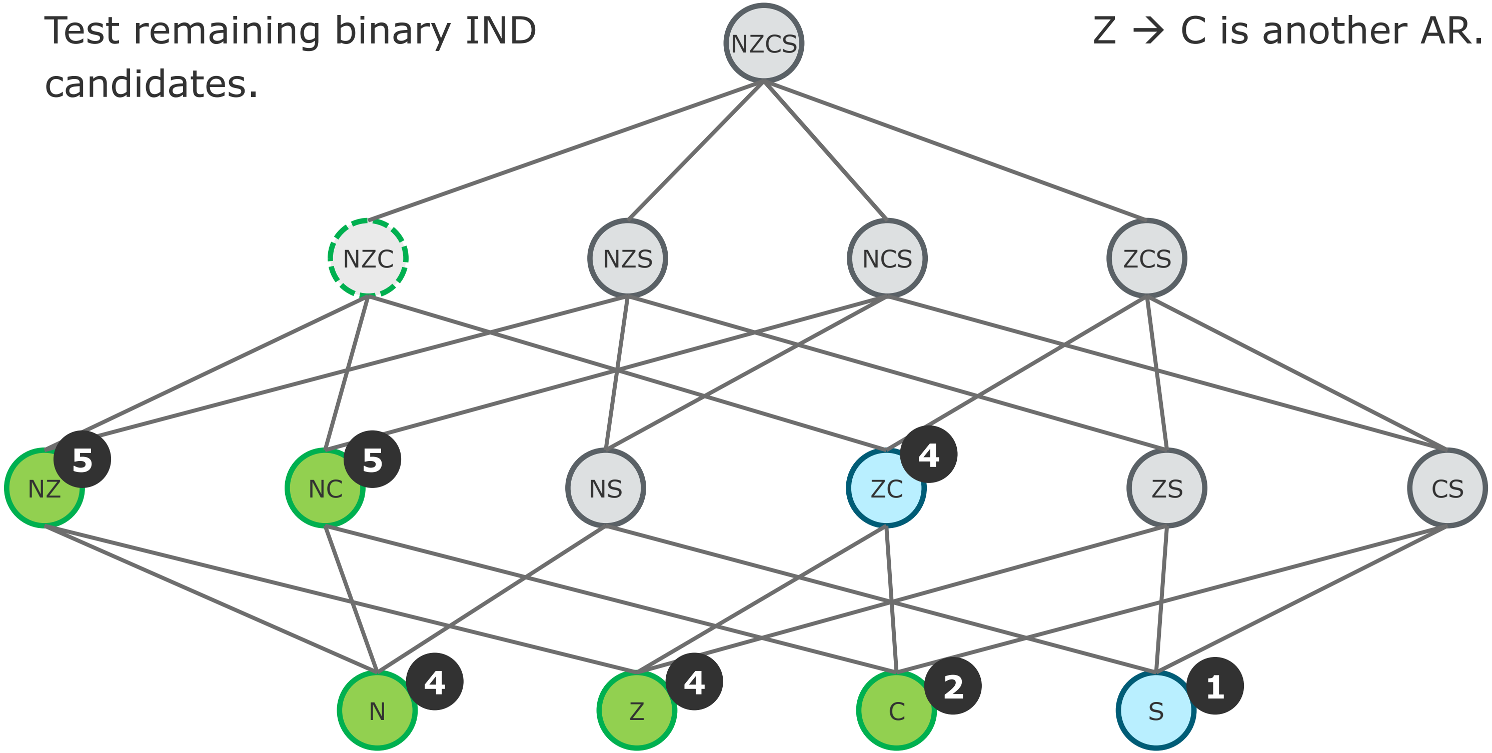
We can prune all n-ary INDs that comprise S, because they can be inferred.



Mixed IND and AR discovery

Test remaining binary IND candidates.

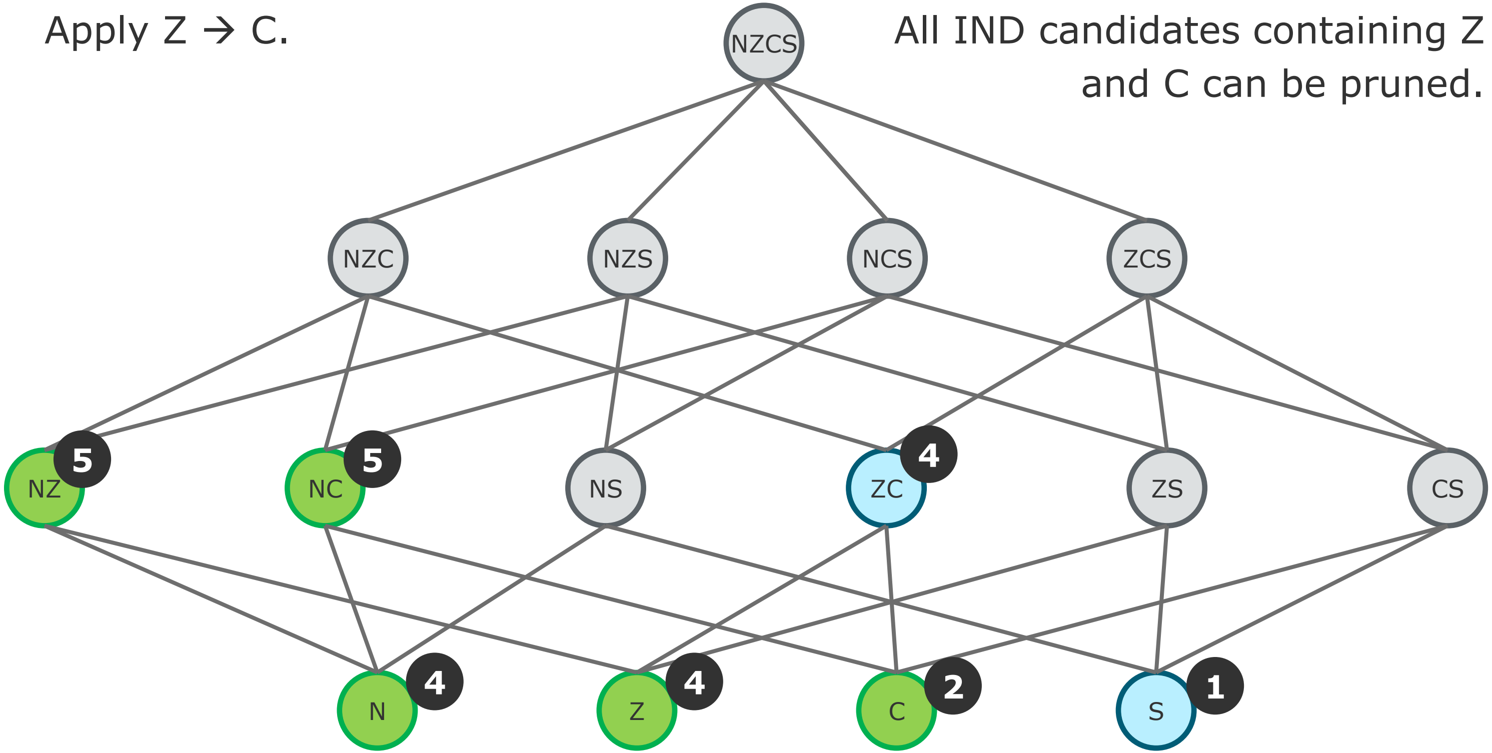
Z → C is another AR.



Mixed IND and AR discovery

Apply Z → C.

All IND candidates containing Z and C can be pruned.



Mixed IND and AR discovery

- Final result:
  - INDs: *[Name', ZIP'] ⊆ [Name, ZIP], [Name', City'] ⊆ [Name, City]*
  - ARs: *[ZIP'] ⊆ [ZIP] → [City'] ⊆ [City], {} → [Status'] ⊆ [Status]*

- Checked only 3 out of 11 (valid) IND candidates.

Name	Zip	City	Status
Tim	10627	Berlin	Student
Tom	10627	Berlin	Student
Tom	14482	Potsdam	Student
Sandy	10324	Berlin	Student
Inge	14469	Potsdam	Student

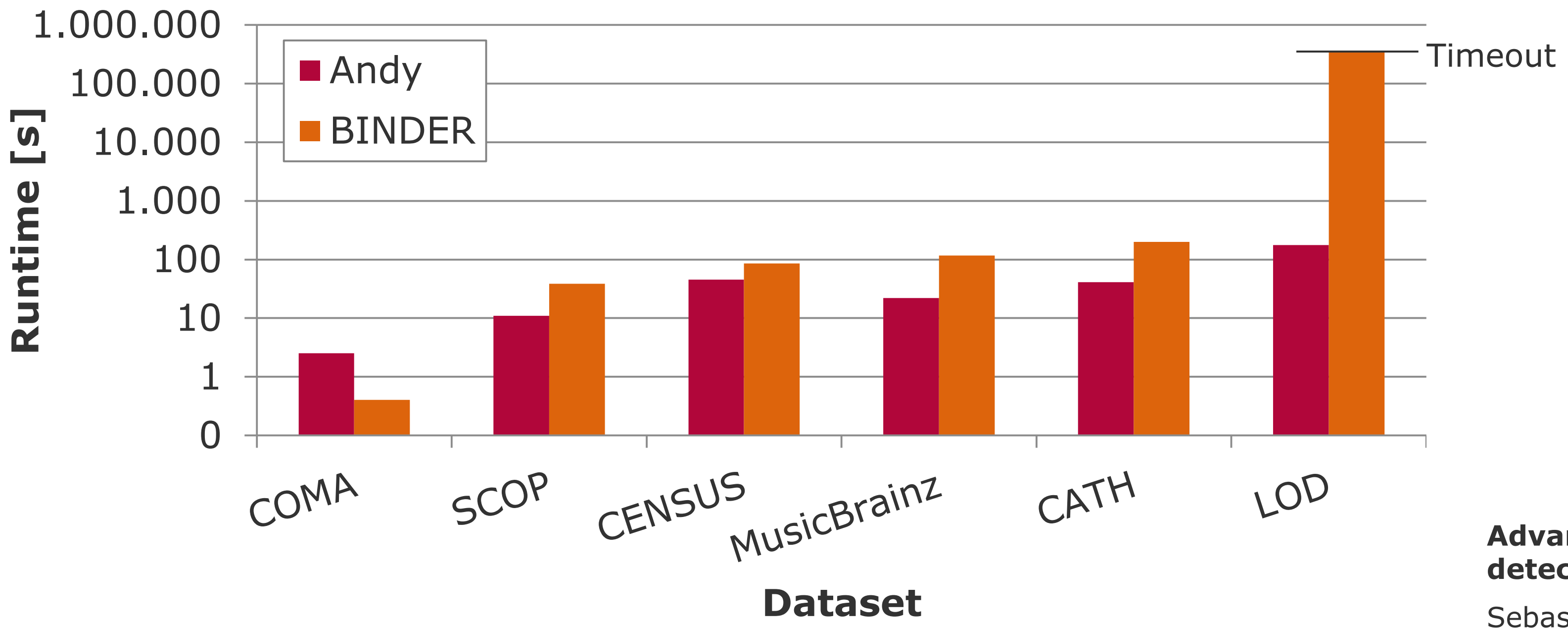
Students

Name'	Zip'	City'	Status'
Tim	10627	Berlin	Student
Tom	10627	Berlin	Student
Inge	14469	Potsdam	Student

HPI Students

Evaluation

Hasso Plattner Institut



- Andy uses SINDY-style candidate checking based on Flink.
  Both run on a single machine but ANDY uses 2 cores/4 threads.

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Advanced IND detection methods

Sebastian Kruse
Thorsten Papenbrock

Literature

- Papenbrock, T., Kruse, S., Quiané-Ruiz, J. A., & Naumann, F. (2015). Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment*, *8*(7), 774-785.
- Kruse, S., Papenbrock, T., & Naumann, F. (2015). Scaling Out the Discovery of Inclusion Dependencies. In *Proceedings of the Conference on Database Systems for Business, Technology, and Web* (pp. 445-454).
- De Marchi, F., & Petit, J. M. (2003). Zigzag: a new algorithm for mining large inclusion dependencies in databases. In *Proceedings of the International Conference on Data Mining* (pp. 27-34).
- Koeller, A., & Rundensteiner, E. A. (2003). Discovery of high-dimensional inclusion dependencies. In *Proceedings of the International Conference on Data Engineering* (pp. 683-685).
- Casanova, M. A., Fagin, R., & Papadimitriou, C. H. (1984). Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, *28*(1), 29-59.