MIPRO 2017 40th Jubilee International Convention
Opatija, Croatia, May 22–26, 2017

Accelerating Dynamic Itemset Counting on Intel Many-core Systems

Mikhail Zymbler

South Ural State University, Chelyabinsk, Russia

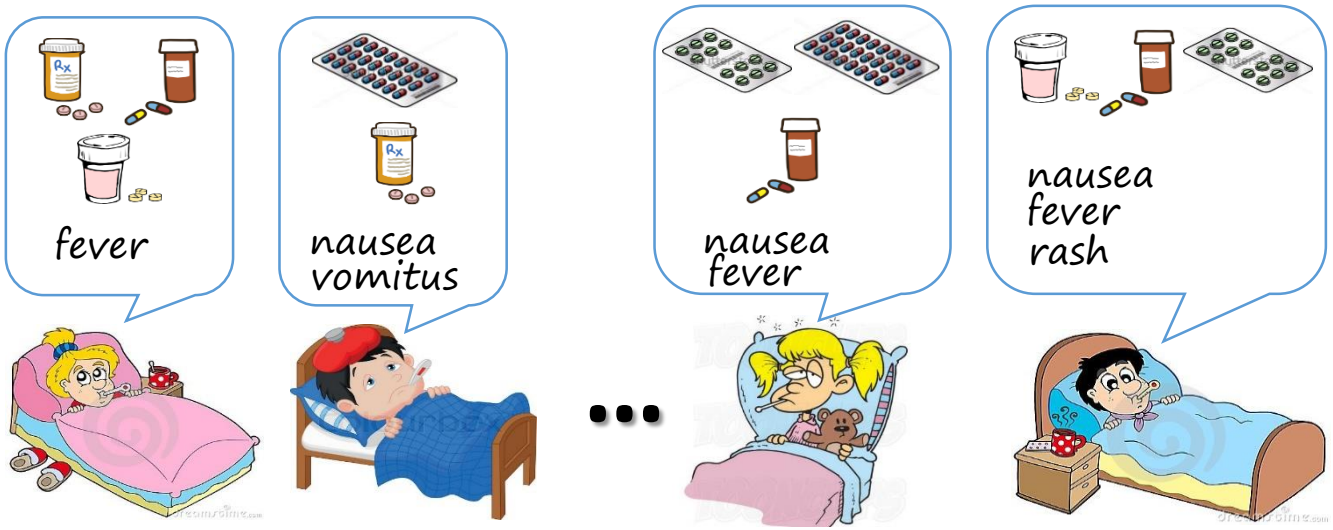
mzym@susu.ru

This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463), by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and by the Ministry of education and science of Russian Federation (government order 2.7905.2017).

Counting frequent itemsets: use cases

- Supermarket: what items are often bought together?

- Medicine: what side-effects are produced by drugs?



Problem statement (1/2)

- Items and itemsets
  - *Item* is literal
  - *Itemset* is a finite set of itemsets. k-*itemset* consists of $k \geq 1$ items
- Transactions
  - *Transaction* is a tuple <TID, k-itemset>
  - Transaction database D
- Support
  - *Support count* of an itemset is a number of transactions that contain the given itemset

D

TID	Itemset
10	nuts, beer, diapers
20	coffee, beer, diapers
30	coffee, beer, diapers, bread
40	milk, nuts, bread
50	coffee, milk, nuts, diapers, bread

- Example:
  - *supp*({beer})=3
  - *supp*({nuts,beer})=1
  - *supp*({coffe,diapers})=2
  - *supp*({beer,diapers})=3
  - *supp*({coffe,diapers,bread})=2

Problem statement (2/2)

- Frequent itemset
  - Minimum support threshold $minsup$ is predefined by a user
  - *Frequent k-itemset* (k≥1) is a k-itemset with support greater or equal than $minsup$ (otherwise it is *infrequent*)
  - L_k is a set of all the frequent k-itemsets
- Problem of itemset counting
  - for the given D and $minsup$ count all the L_k (k≥1)

D

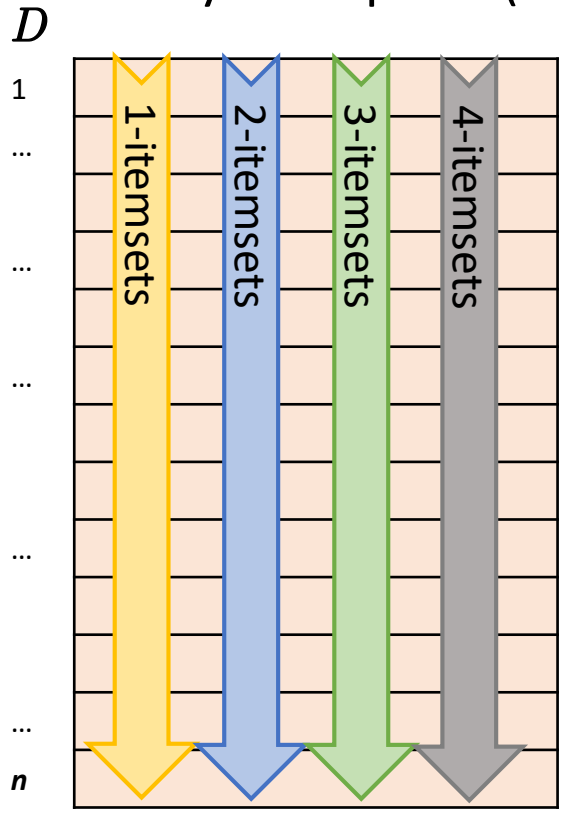
TID	Itemset
10	nuts, beer, diapers
20	coffee, beer, diapers
30	coffee, beer, diapers, bread
40	milk, nuts, bread
50	coffee, milk, nuts, diapers, bread

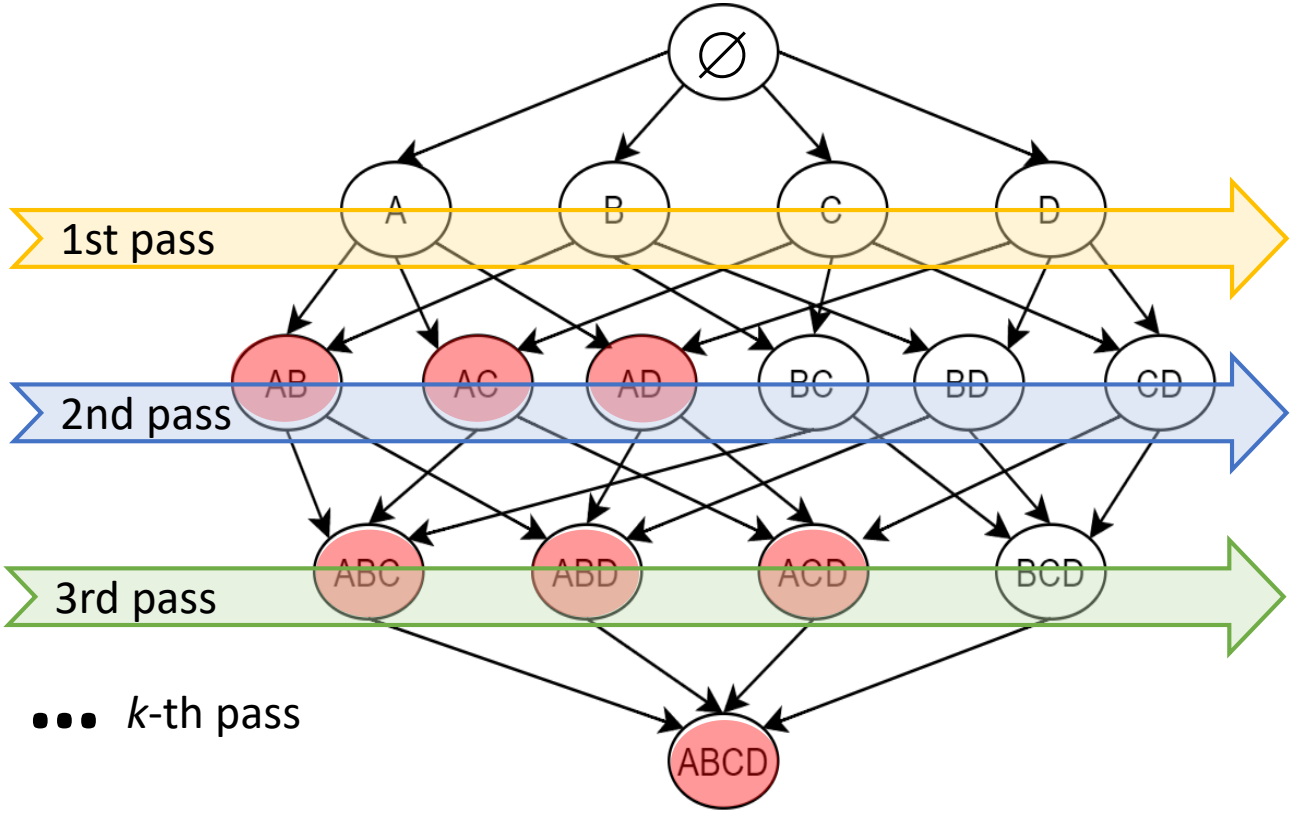
- *minsup*=3
  - L_1={{nuts}: supp=3, {beer}: supp=3, {diapers}: supp=4, {bread}: supp=3, {coffe}: supp=3}
  - L_2={{beer,diapers}: supp=3}
- *minsup*=2
  - L_1={{nuts}: supp=3, {beer}: supp=3, {diapers}: supp=4, {bread}: supp=3, {coffee}: supp=3, {milk}: supp=2}
  - L_2={{coffee,beer}: supp=2, {beer,diapers}: supp=3, ...}
  - L_3={{coffee,diapers,bread}: supp=2}}

Apriori counting: classical algorithm

by R. Agrawal et. al, 1994

- *Candidates*
  - itemsets of $L_{k\text{-}1}$ are joined to form new k-itemsets to be counted
- *Pruning*
  - any infrequent (k-1)-itemset cannot be a subset of a frequent k-itemset



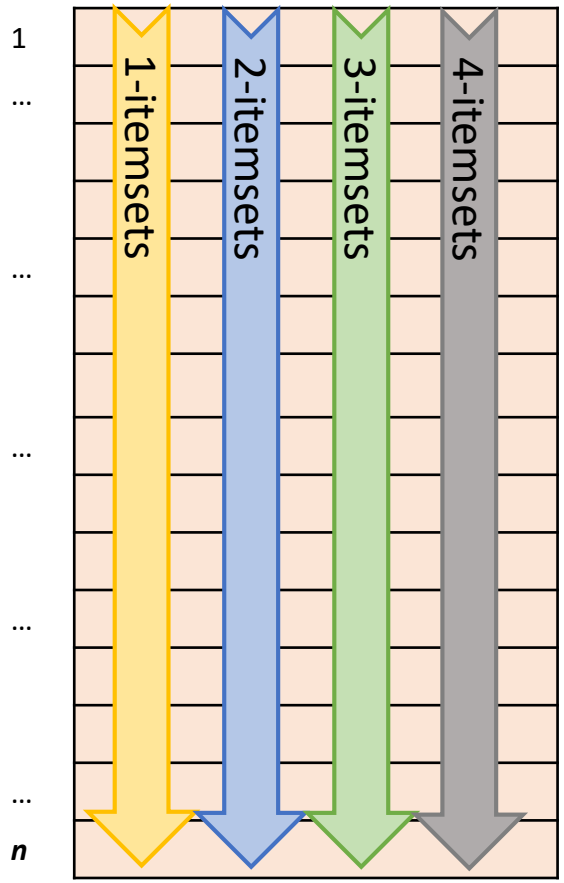


Dynamic Itemset Counting (DIC)

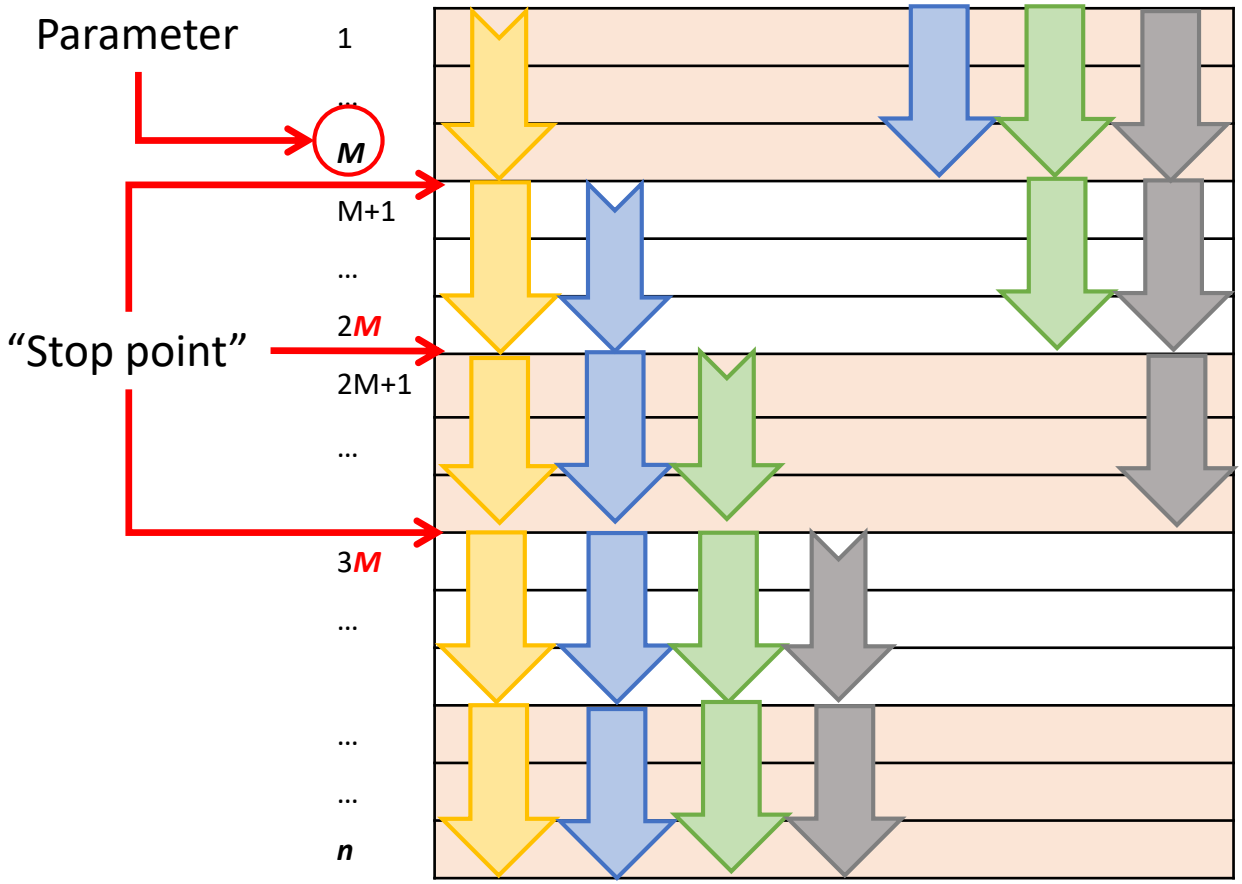
by S. Brin, R. Motwani, J. Ullmann, S. Tsur, 1997

- We may start counting of *(k+1)*-itemset before the k-th pass in case we suspect that this itemset is frequent

Apriori: 4 passes

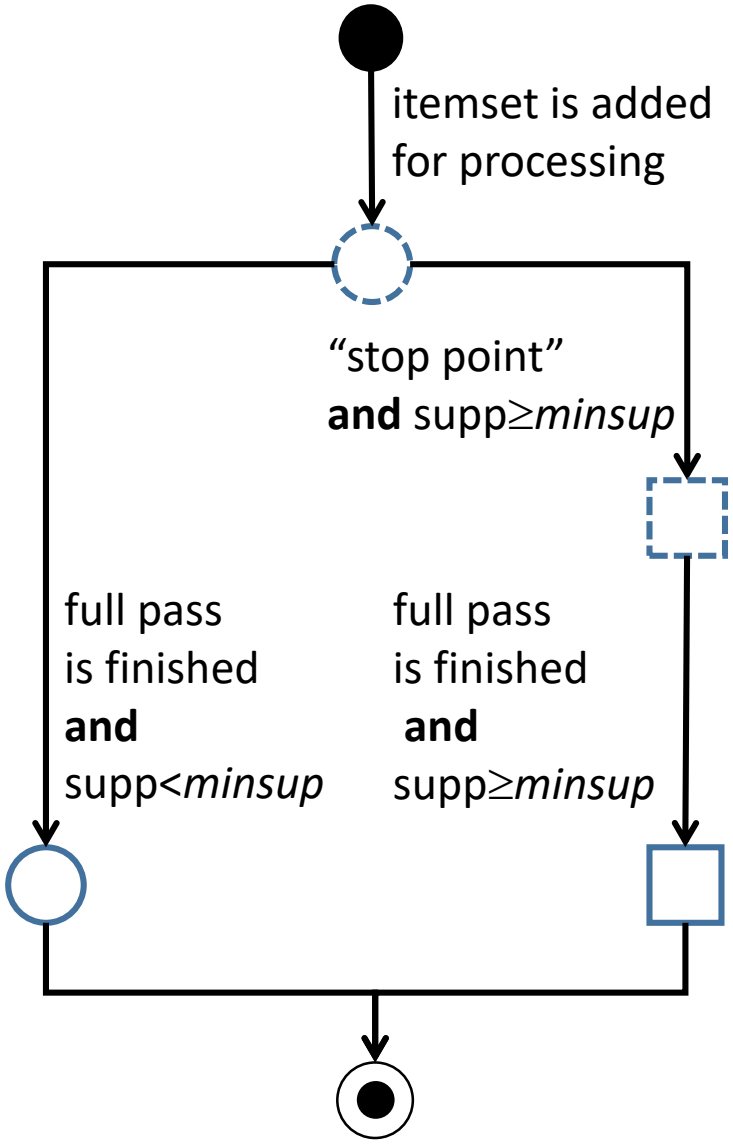


DIC: less than 2 passes



DIC: types of itemsets

- **suspected infrequent**
  - initial status (“candidate”)
  - counting should be carried on
- **suspected frequent**
  - intermediate status
  - counting should be carried on
- **confirmed infrequent**
  - final status, not result
  - counting should be stopped
- **confirmed frequent**
  - final status, result
  - counting should be stopped



Direct bit representation

- Transaction (itemset) bitmask: $Bitmask(T)$
  - i-th bit of bitmask is set to 1 $\Leftrightarrow$ itemset $i \in T$
  - otherwise i-th bit is set to 0
- Bit mapping of database: $Bitmap(D)$
  - $Bitmap(D)$={$Bitmask(T_j)$: 1≤j≤n}

D
3
2, 6, 7, 9

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													1		
							1		1	1				1	

$Bitmap(D)$
4
354

- Advantages
  - Reduce size of transaction database
    We suppose that D fits in main
  - Speed up support counting
    $I \subseteq T \Leftrightarrow Bitmask(I)$ **AND** $Bitmask(T)$=$Bitmask(I)$

Implementation of data structures

- Transaction database (bit map)
  - unsigned long long int * **BITMAP**;
  - **BITMAP** = _mm_malloc(n*sizeof(unsigned long long int)) // Data alignment
- Itemset is struct {
  - unsigned long long mask[m/sizeof(unsigned long long)] // Bit mask
  - int k // Number of items in the itemset
  - int stopNo // Number of "stop point"
  - int supp // Support of the itemset
  - int fig // Type: one of  } **itemset**;
- Itemsets
  - C++ STL (Standard Template Library)
  - vector<struct itemset> **DASHED** // Set of  itemsets
  - vector<struct itemset> **SOLID** // Set of  itemsets
  - vector<struct itemset> **CAND** // Candidate itemsets

Parallelization: hardware & technologies

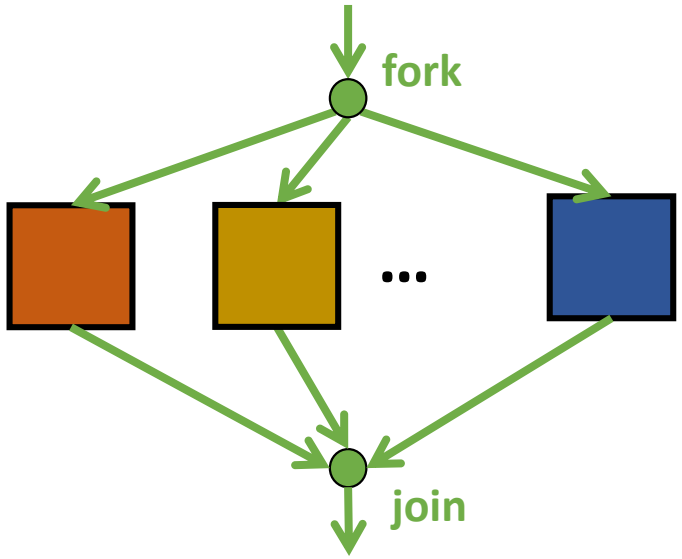
Specification	CPU	Coprocessor
Model, Intel Xeon	X5680	Phi SE10X
# of physical cores	6	61
Hyper threading factor	2	4
# of logical cores	12	244
Frequency, GGz	3.33	1.1
Performance, TFLOPS	0.371	1.076
Memory size, Gb	24	8

OpenMP technology

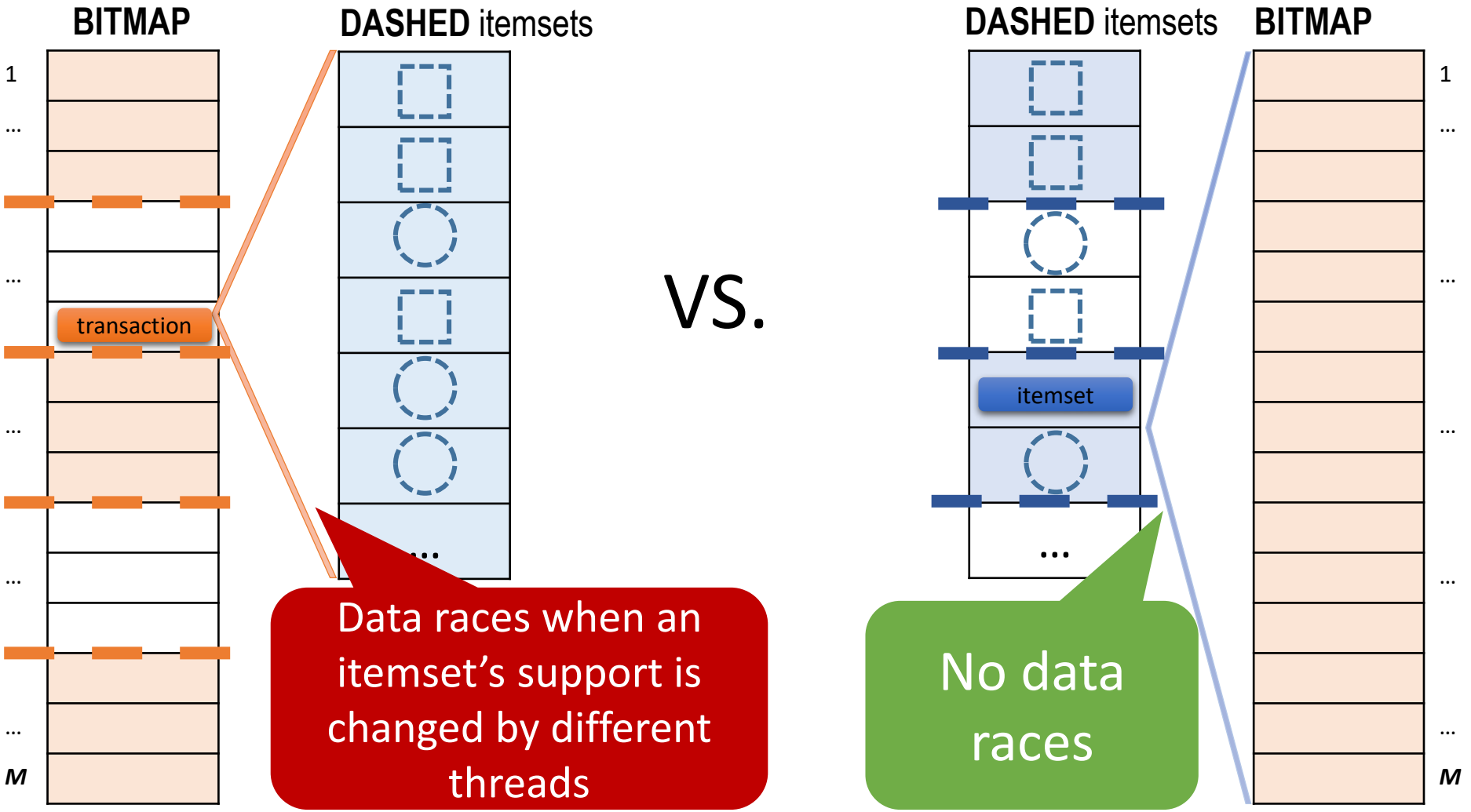
```
#pragma omp parallel for
for (i=0; i<n; i++) {
  ...
}
```

fork

join

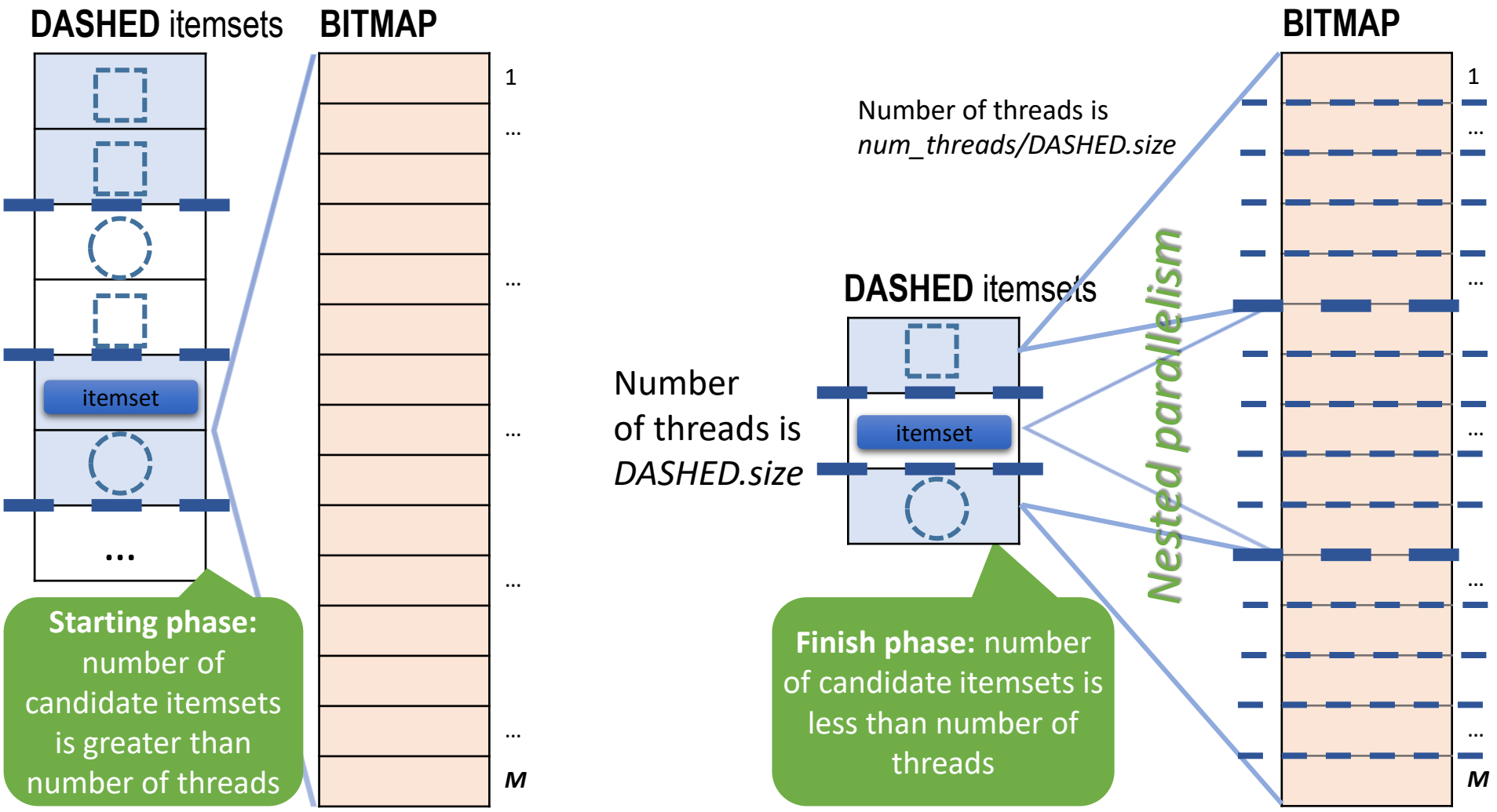


Parallelization: counting of itemsets



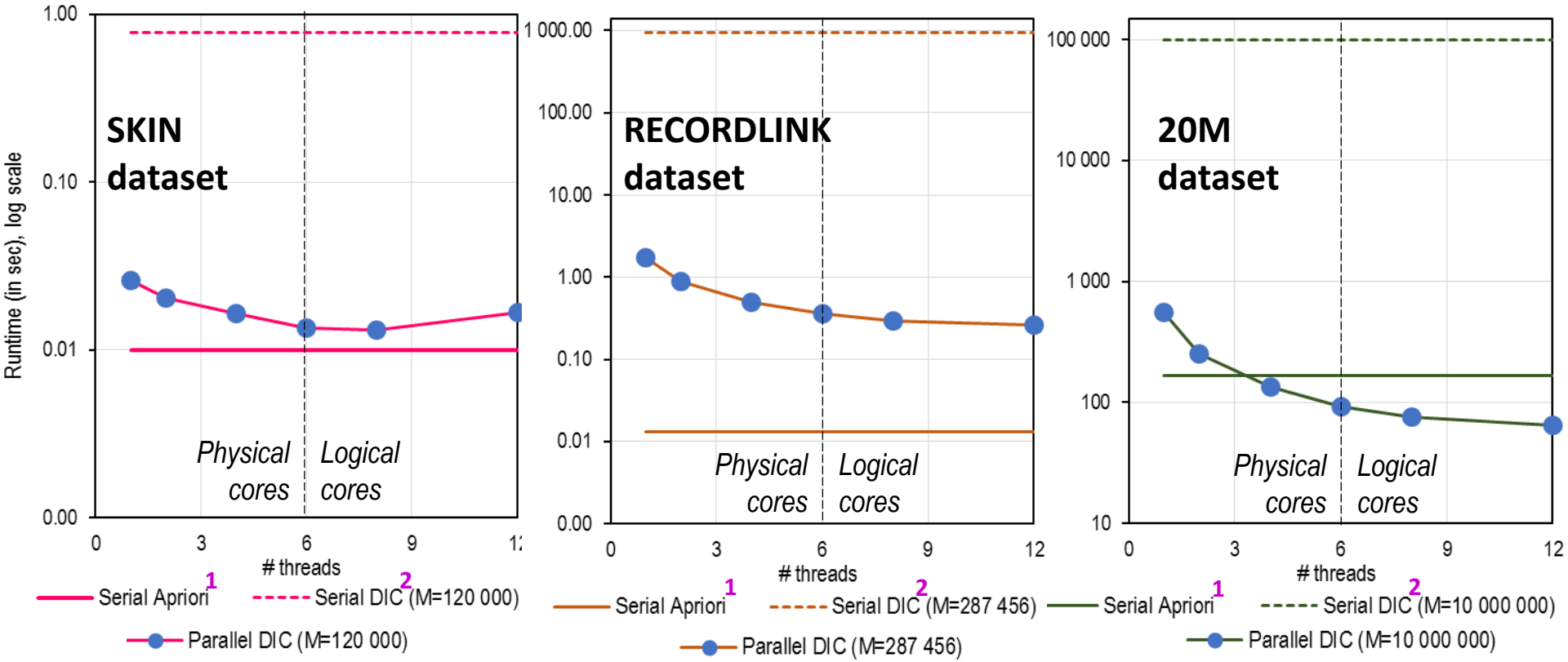
Parallelization: load balancing

- Number of itemsets in **DASHED** changes after the "stop point"



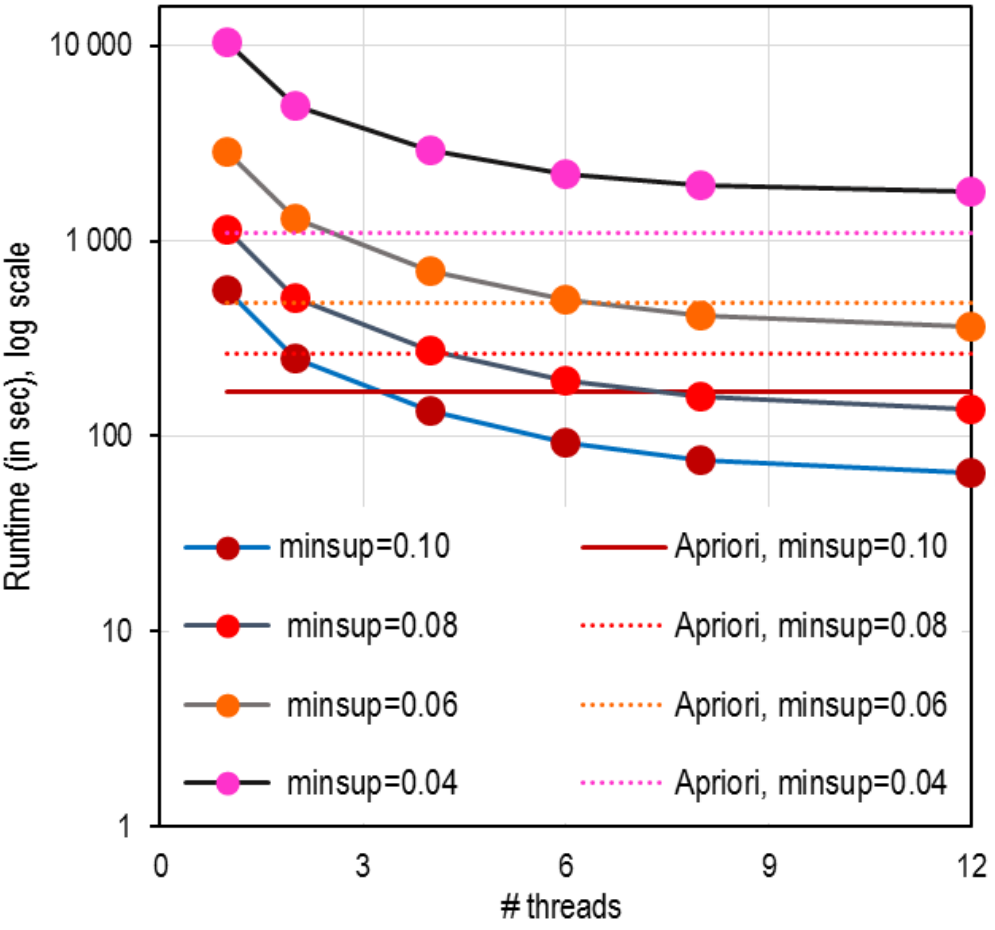
Experiments: performance (CPU)

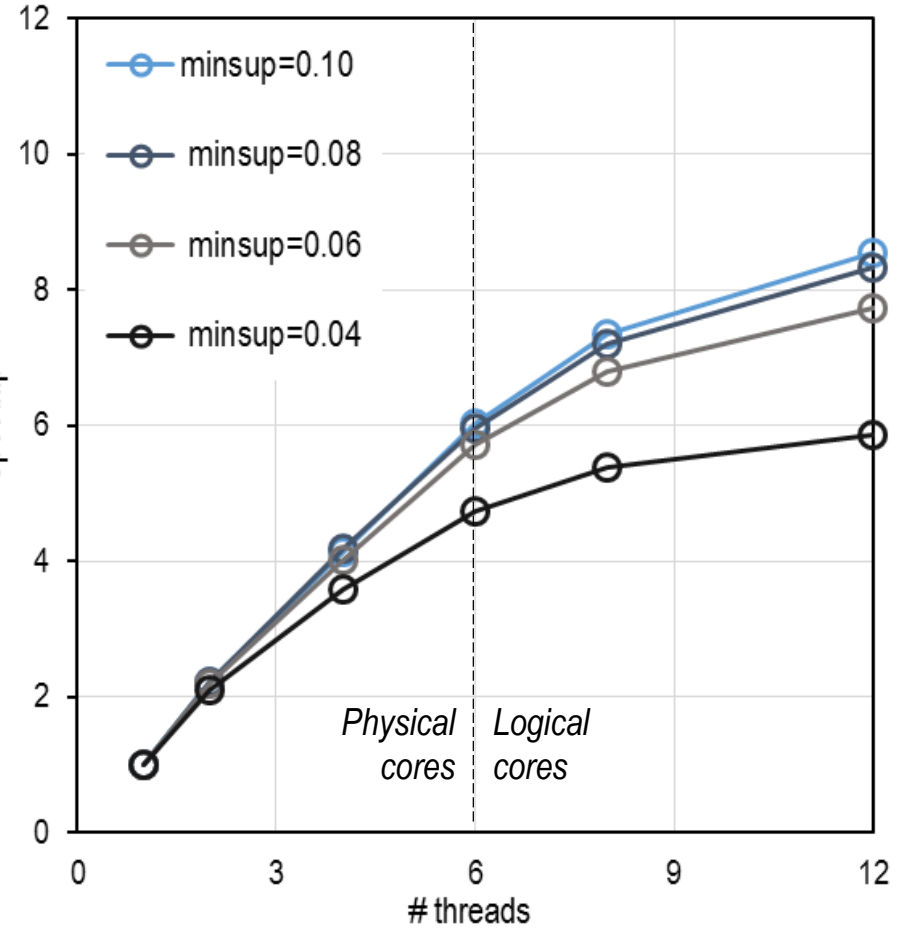
[1] By C. Borgelt
[2] By B. Goethals



Dataset	# of transactions	# of items	Reference
SKIN	245 057	11	Dhall A., et. al. Adaptive Digital Makeup. ISVC 2009. LNCS, vol. 5876. Springer, 2009, pp. 728–736.
RECORDLINK	574 912	29	Sariyar M., et al. Controlling False Match Rates in Record Linkage Using Extreme Value Theory. J. of Biomed. Informatics, vol. 44, no. 4, pp. 648–654, 2011.
20M	20 000 000	64	Synthetically generated via IBM Quest, like authors of the DIC. Avg transaction length is 20, avg length of frequent itemset is 10.

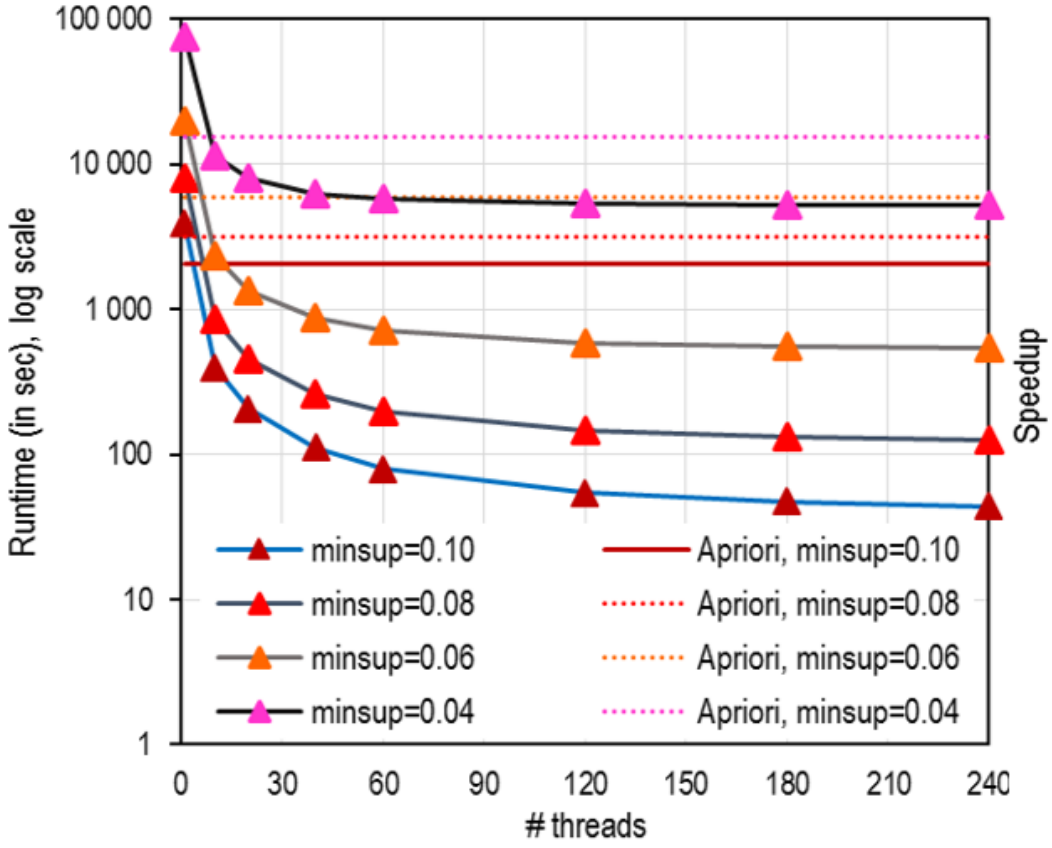
Experiments: Scalability w.r.t. *minsup* (CPU)

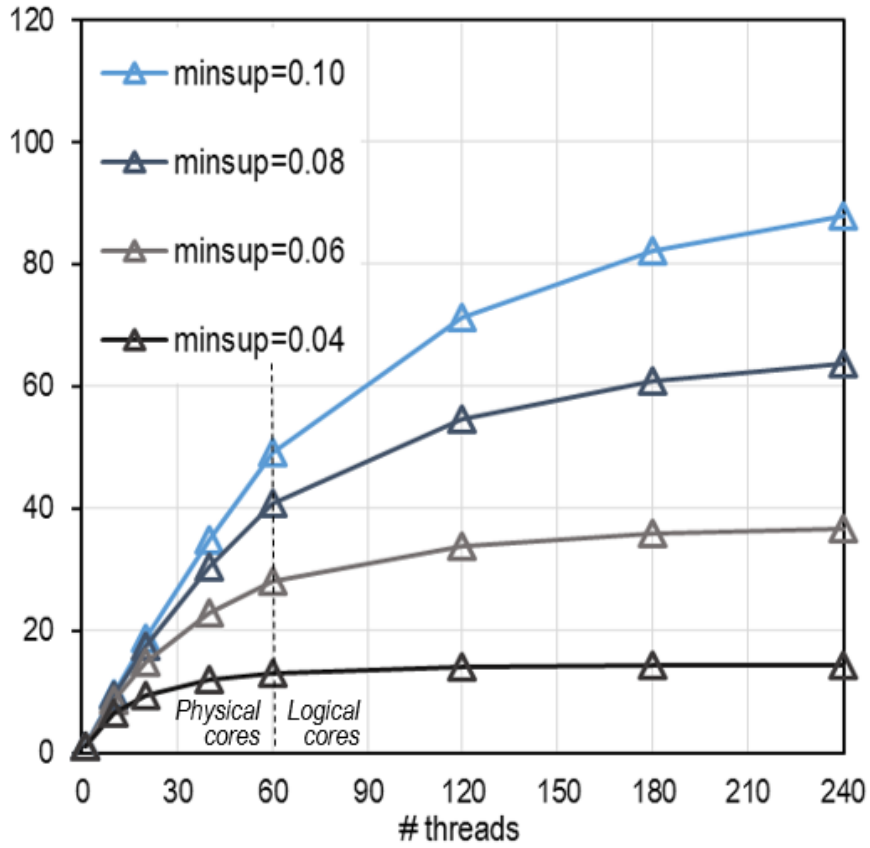




Dataset: *20M*
M=10^7

Experiments: Scalability w.r.t. *minsup* (Phi)





Dataset: *20M*
M=10^7

Conclusion and future work

- We design and implement parallel DIC for Intel many-core systems
- Experiments show good scalability of the algorithm
- Further research: extend the algorithm to the case of cluster computing system with Intel Xeon Phi processors onboard

Thank you for paying attention! Questions?

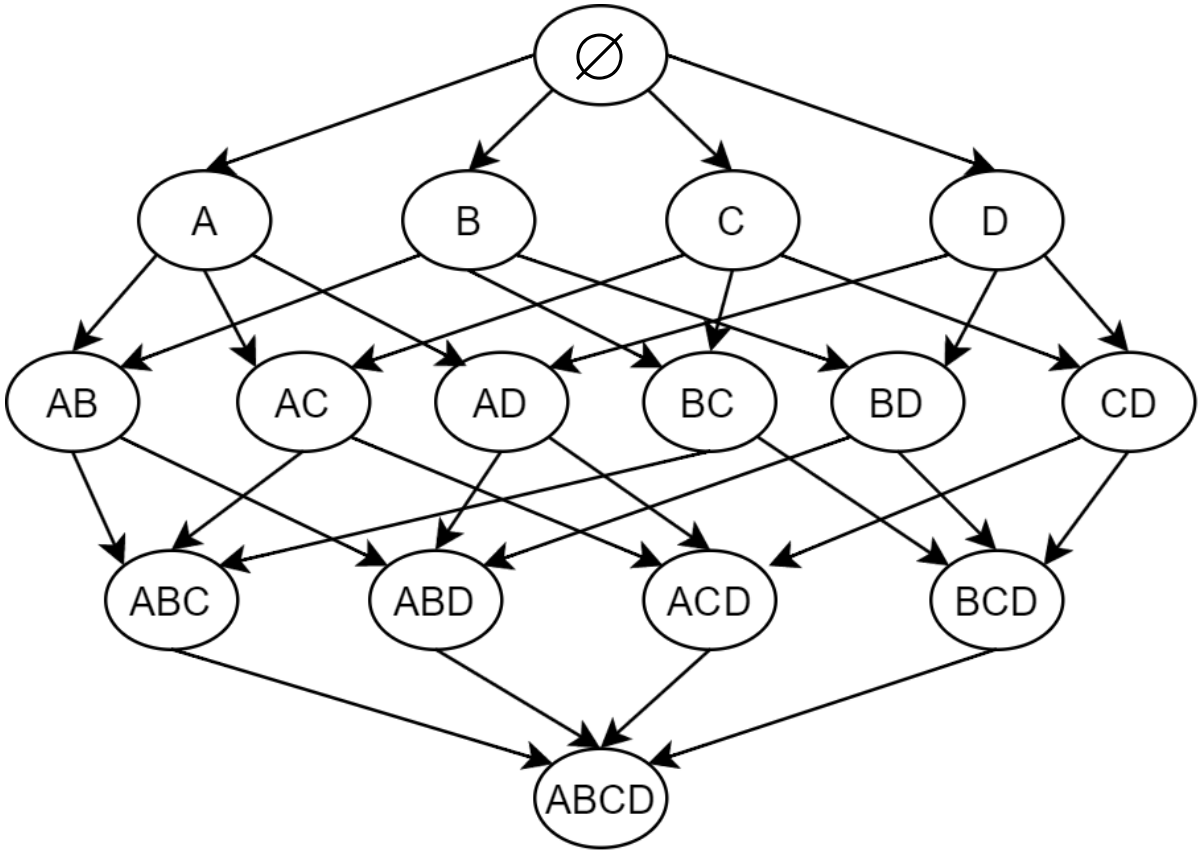
Mikhail Zymbler

Supplemental slides

DIC: how it works (1/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle ∅

Dashed Box ∅

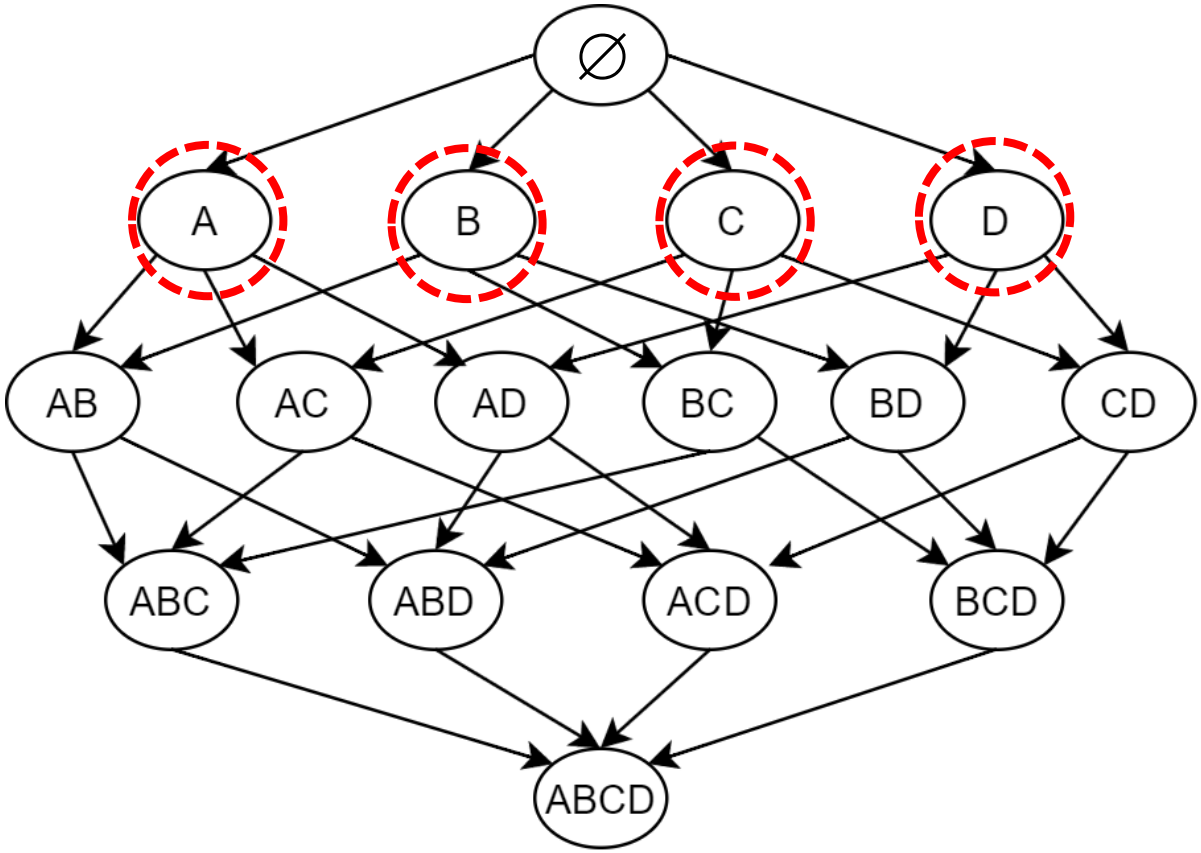
Solid Circle ∅

Solid Box ∅

DIC: how it works (2/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
A: 0; B: 0;
C: 0; D: 0

Dashed Box
∅

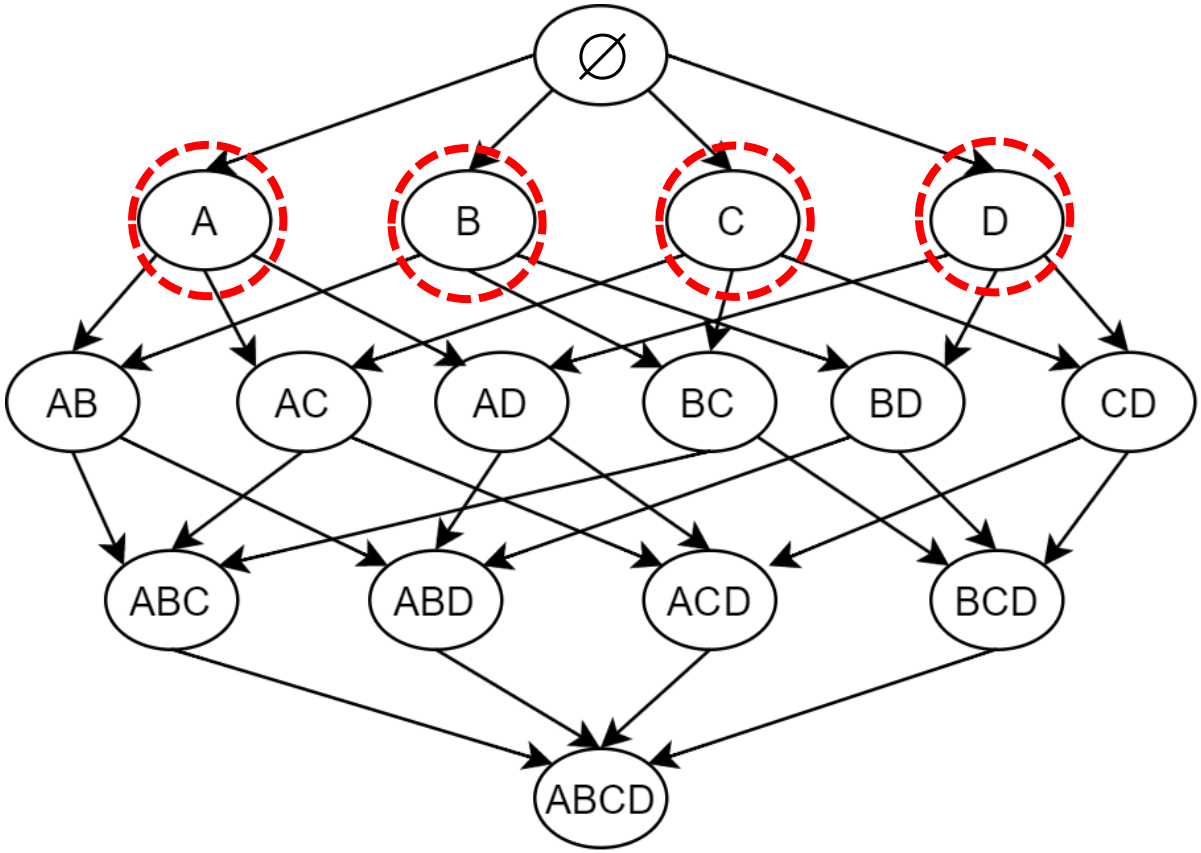
Solid Circle
∅

Solid Box
∅

DIC: how it works (3/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
A: 4; B: 4;
C: 3; D: 4

Dashed Box
∅

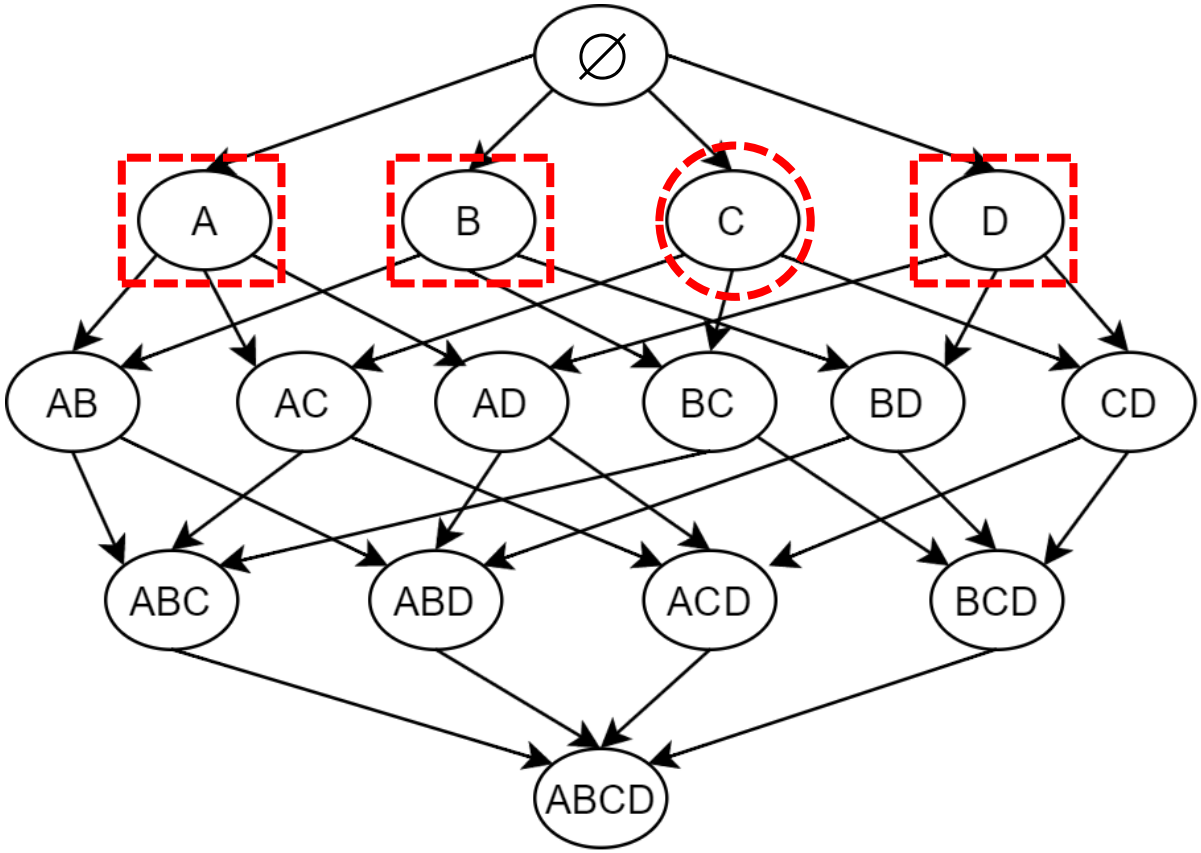
Solid Circle
∅

Solid Box
∅

DIC: how it works (4/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
C: 3

Dashed Box
A: 4;
B: 4;
D: 4

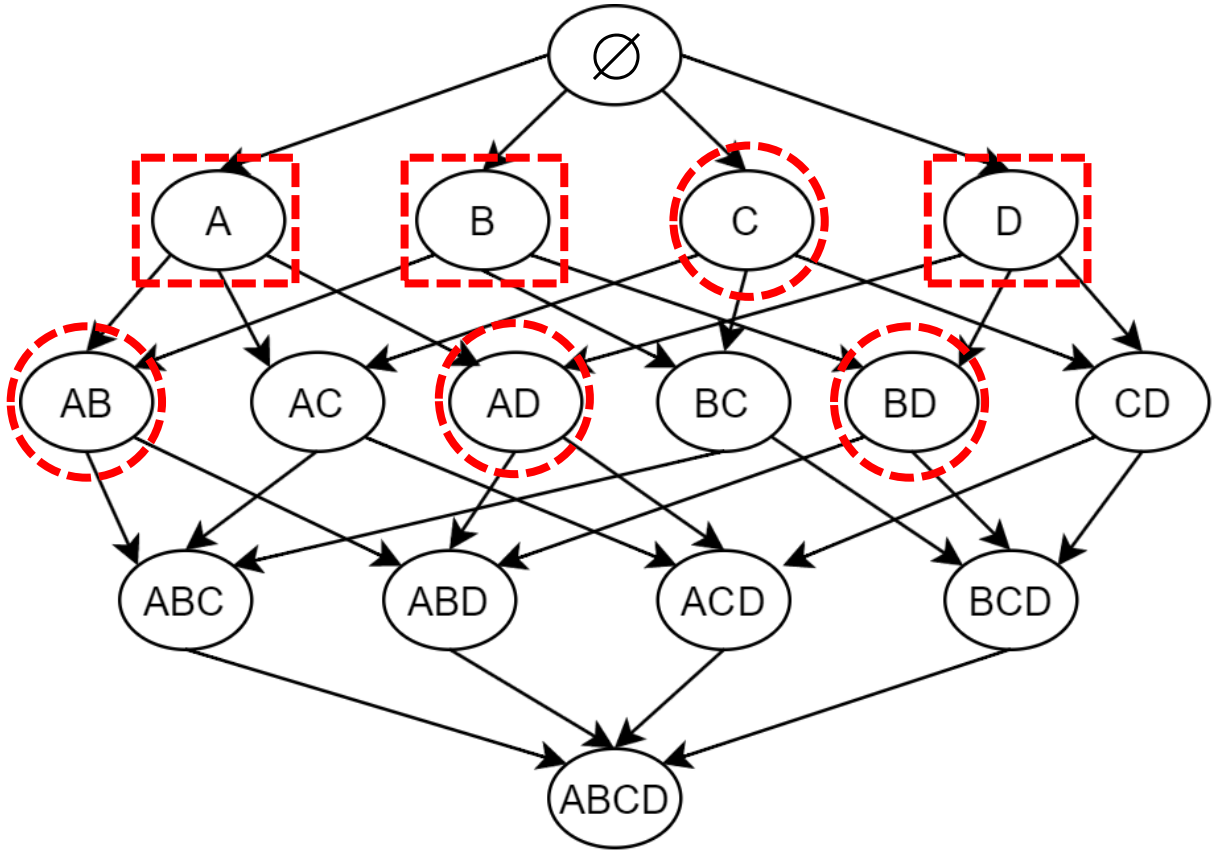
Solid Circle
∅

Solid Box
∅

DIC: how it works (5/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
C: 3; AB: 0;
AD: 0; BD: 0

Dashed Box
A: 4;
B: 4;
D: 4

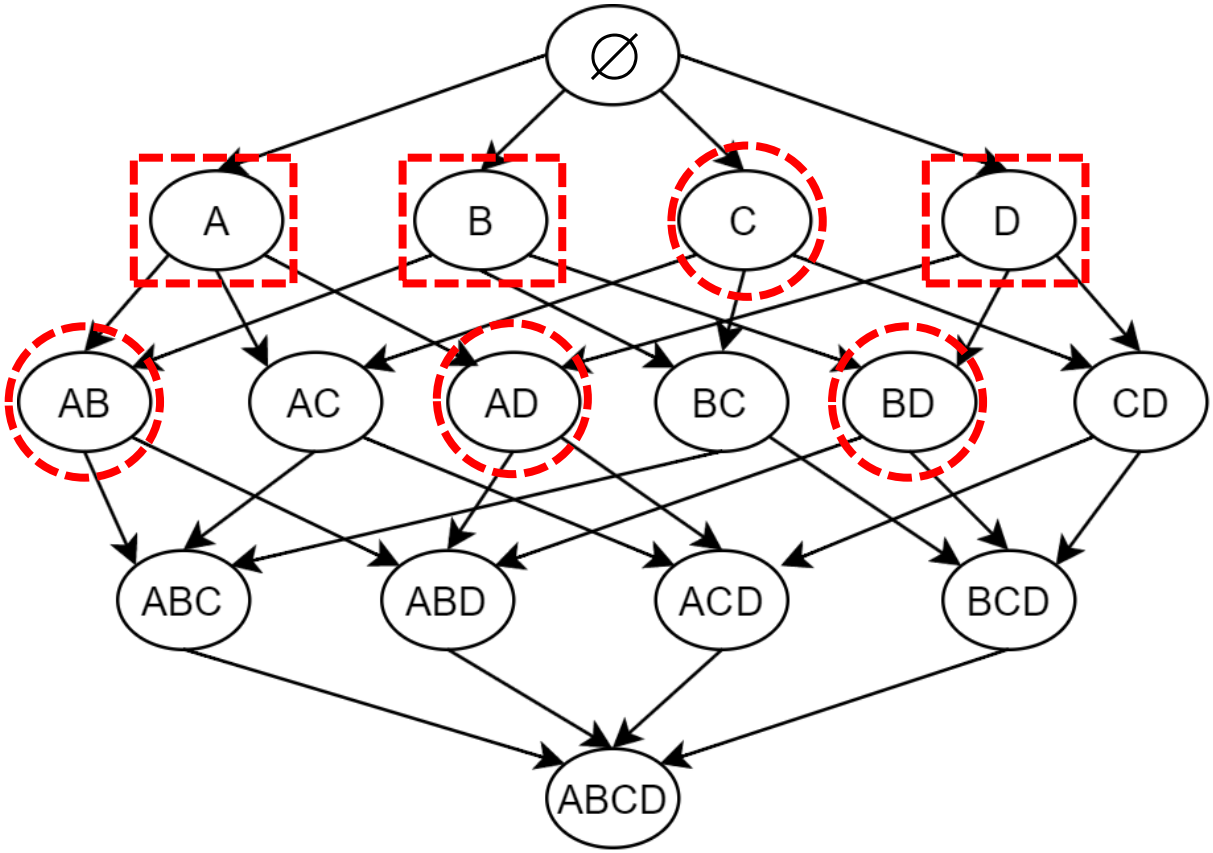
Solid Circle
∅

Solid Box
∅

DIC: how it works (6/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
C: 3; AB: 2;
AD: 0; BD: 0

Dashed Box
A: 7;
B: 8;
D: 4

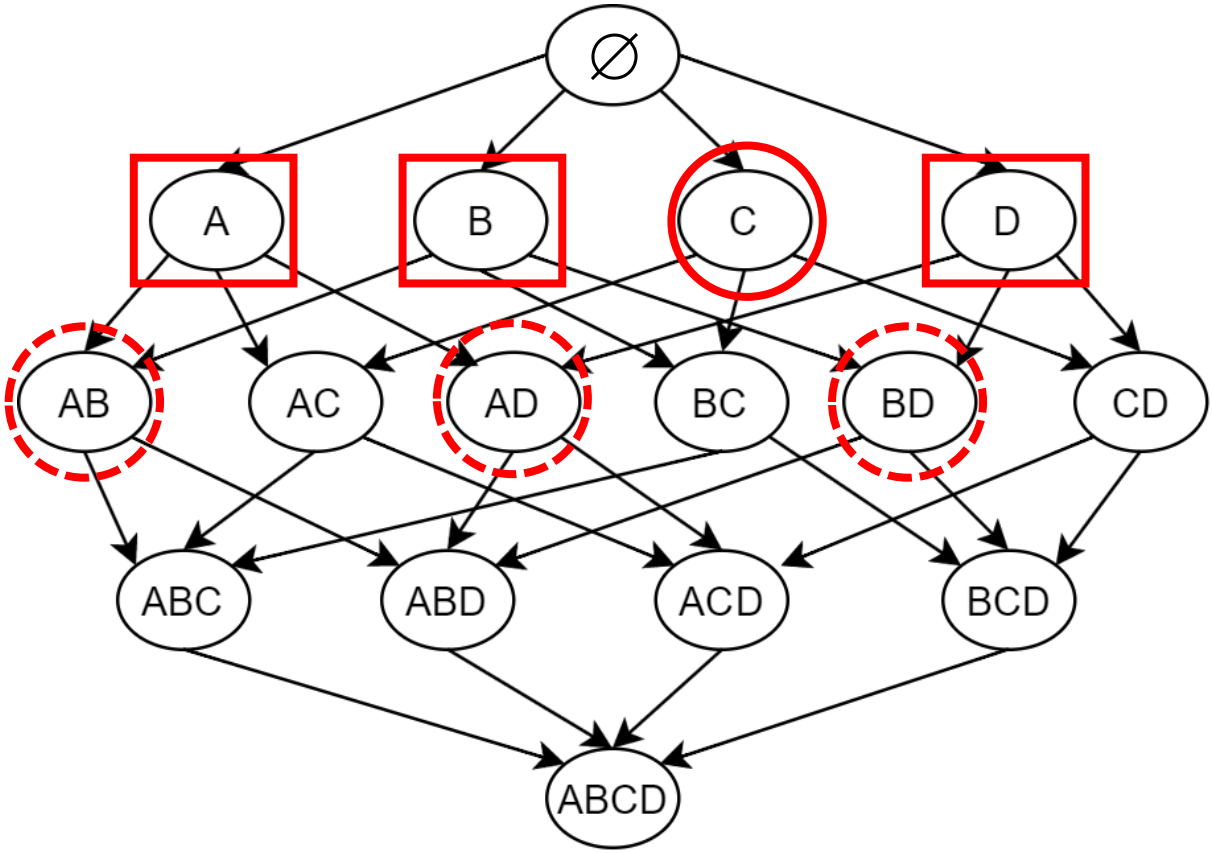
Solid Circle
∅

Solid Box
∅

DIC: how it works (7/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
AB: 2; AD: 0;
BD: 0

Dashed Box
∅

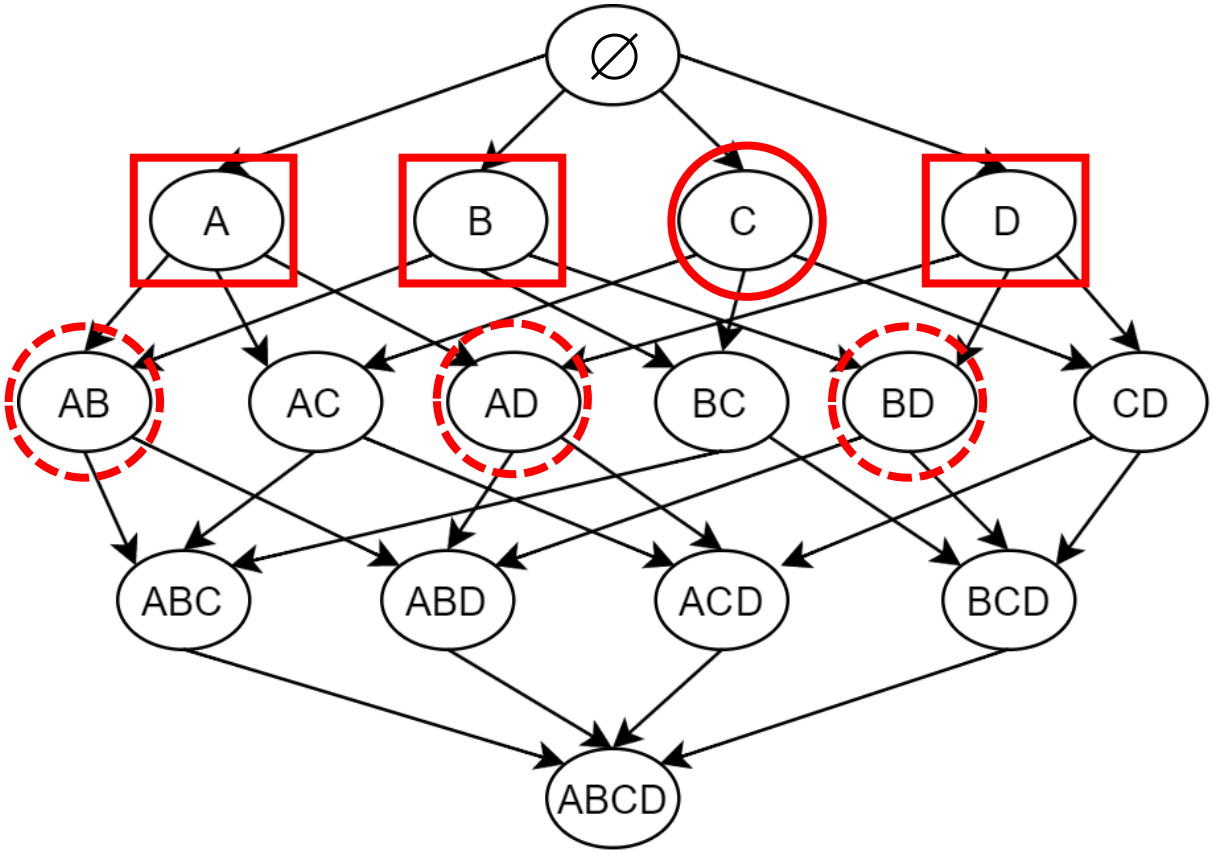
Solid Circle
C: 3

Solid Box
A: 7;
B: 8;
D: 4

DIC: how it works (8/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
AB: 5; AD: 3;
BD: 4

Dashed Box
∅

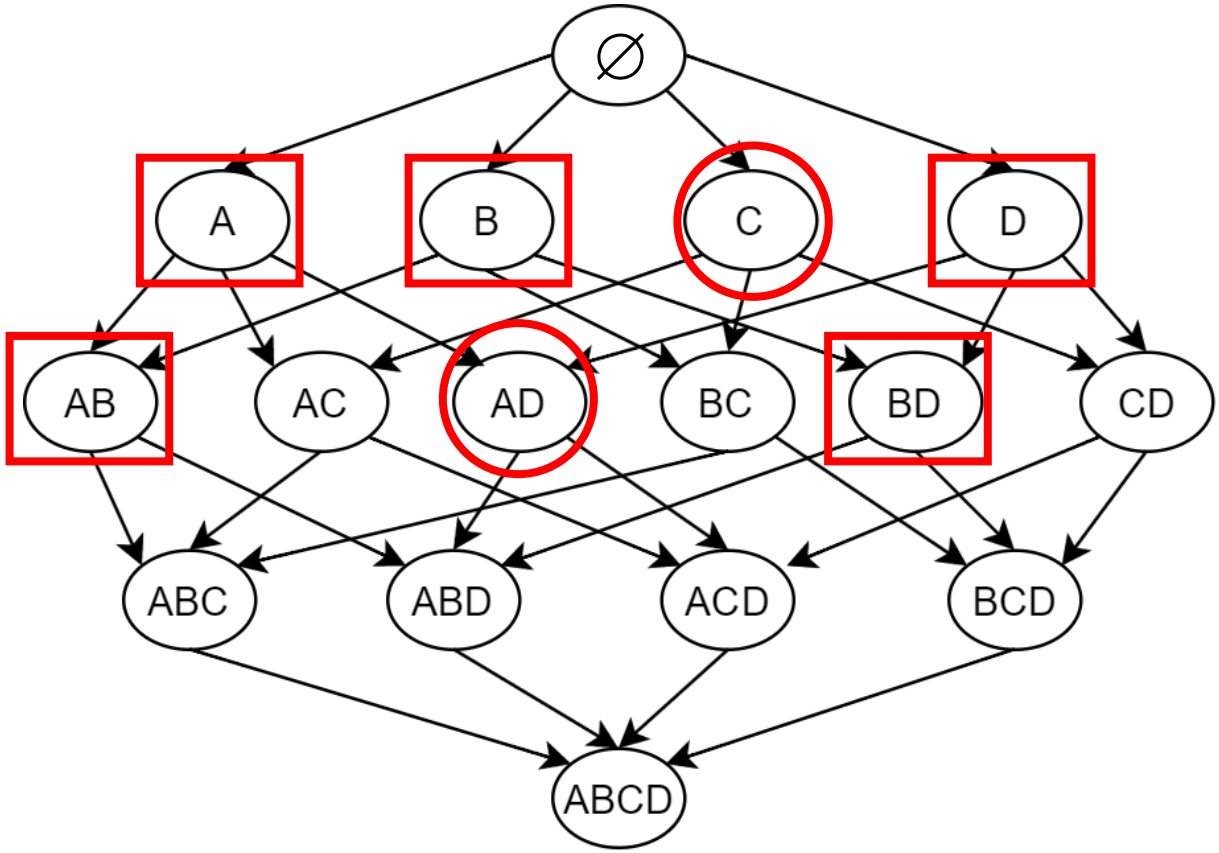
Solid Circle
C: 3

Solid Box
A: 7;
B: 8;
D: 4

DIC: how it works (9/9)

D *M*=5
minsup=4

1	A C
2	B C D
3	A B D
4	A B D
5	A B C D
6	A
7	B
8	B
9	A B
10	A B



Dashed Circle
∅

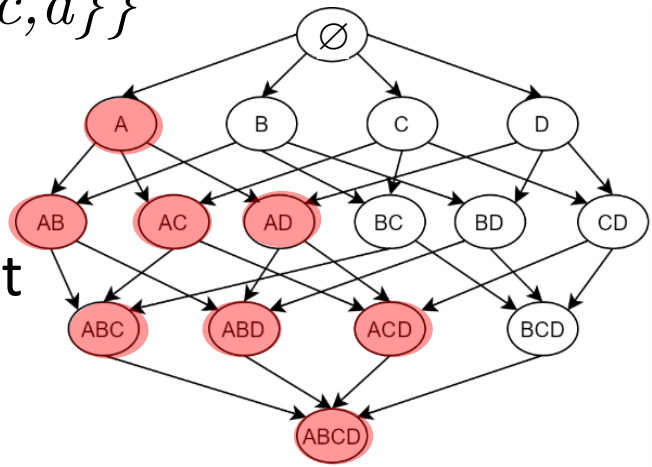
Dashed Box
∅

Solid Circle
C: 3;
AD: 3

Solid Box
A: 7; B: 8;
D: 4; AB: 5;
BD: 4

Algorithm Apriori

- Method
  - k:=1; Find L_1
  - for each k≥2
    - Using L_{k-1} generate C_k, set of candidate itemsets
    - Prune clearly infrequent itemsets from C_k
    - Find L_k by scanning D and counting support of k-itemsets of C_k
    - if L_k=∅ then STOP
- Candidate generation: $JOIN$
  - L_3=$\{\{a,b,c\},\{a,b,d\},\{a,c,d\},\{a,c,e\},\{b,c,d\}\}$
  - C_4 =L_3 $JOIN$ L_3=$\{\{a,b,c,d\},\{a,c,d,e\}\}$
- Pruning: Apriori principle
  - Each non-empty subset of frequent itemset must be frequent itemset



Algorithm Apriori (1/2)

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;
```

Algorithm Apriori (2/2)

```
procedure apriori_gen(L_{k-1}:frequent (k-1)-itemsets)
(1)     for each itemset l_1 ∈ L_{k-1}
(2)         for each itemset l_2 ∈ L_{k-1}
(3)             if (l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2])
                    ∧... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] < l_2[k-1]) then {
(4)                 c = l_1 ⋈ l_2; // join step: generate candidates
(5)                 if has_infrequent_subset(c, L_{k-1}) then
(6)                     delete c; // prune step: remove unfruitful candidate
(7)                 else add c to C_k;
(8)             }
(9)     return C_k;

procedure has_infrequent_subset(c: candidate k-itemset;
          L_{k-1}: frequent (k-1)-itemsets); // use prior knowledge
(1)     for each (k-1)-subset s of c
(2)         if s ∉ L_{k-1} then
(3)             return TRUE;
(4)     return FALSE;
```

Input:

- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   L_1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; L_{k-1} ≠ φ; k++) {
(3)       C_k = apriori_gen(L_{k-1});
(4)       for each transaction t ∈ D { // scan D for counts
(5)           C_t = subset(C_k, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ C_t
(7)               c.count++;
(8)       }
(9)       L_k = {c ∈ C_k | c.count ≥ min_sup}
(10)  }
(11)  return L = ∪_k L_k;
```

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

C_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2
I6	1

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
10	I6

C_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2
I6	1

L_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

C_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2
I6	1

L_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2

C_2

Набор
I1,I2
I1,I3
I1,I4
I1,I5
I2,I3
I2,I4
I2,I5
I3,I4
I3,I5
I4,I5

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

C_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2
I6	1

L_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2

C_2

Набор
I1,I2
I1,I3
I1,I4
I1,I5
I2,I3
I2,I4
I2,I5
I3,I4
I3,I5
I4,I5

C_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

C_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2
I6	1

L_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2

C_2

Набор
I1,I2
I1,I3
I1,I4
I1,I5
I2,I3
I2,I4
I2,I5
I3,I4
I3,I5
I4,I5

C_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

L_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

C_3

Набор
I1,I2,I3
I1,I2,I5
I1,I3,I5
I2,I3,I4
I2,I3,I5
I2,I4,I5

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

C_3

Набор
I1,I2,I3
I1,I2,I5
I1,I3,I5
I2,I3,I4
I2,I3,I5
I2,I4,I5

C_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

C_3

Набор
I1,I2,I3
I1,I2,I5
I1,I3,I5
I2,I3,I4
I2,I3,I5
I2,I4,I5

C_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

L_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

C_4

Набор
I1,I2,I3,I5

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

C_4

Набор
I1,I2,I3,I5

C_4

Набор
Ø

STOP

Apriori: how it works

***D**, minsup=2*

TID	Набор
10	I1, I2, I5
20	I2, I4
30	I2, I3
40	I1, I2, I4
50	I1, I3
60	I2, I3
70	I1, I3
80	I1, I2, I3, I5
90	I1, I2, I3
100	I6

L_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

C_4

Набор
I1,I2,I3,I5

C_4

Набор
Ø

STOP

L_1

Набор	SUPP
I1	6
I2	7
I3	6
I4	2
I5	2

L_2

Набор	SUPP
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

L_3

Набор	SUPP
I1,I2,I3	2
I1,I2,I5	2

Serial DIC

Algorithm 1. DIC(in $\mathcal{D}$, in $minsup$, in M, out $\mathcal{L}$)

```
                                          ▷ Initialize sets of itemsets
2: SolidBox ← ∅; SolidCircle ← ∅; DashedBox ← ∅
   DashedCircle ← ℐ
4: while DashedCircle ∪ DashedBox ≠ ∅ do
                          ▷ Scan database and rewind if necessary
6:     Read(𝒟, M, Chunk)
       if EOF(𝒟) then
8:         Rewind(𝒟)
       for all T ∈ Chunk do
10:                                   ▷ Count support of itemsets
           for all I ∈ DashedCircle ∪ DashedBox do
12:            if I ⊆ T then
                   support(I) ← support(I) + 1
14:                           ▷ Generate candidate itemsets
           for all I ∈ DashedCircle do
16:            if support(I) ≥ minsup then
                   MoveItemset(I, DashedBox)
18:                for all i ∈ ℐ do
                       C ← I ∪ i
20:                    if ∀s ⊆ C s ∈ SolidBox∪DashedBox then
                           MoveItemset(C, DashedCircle)
22:            ▷ Check full pass completion for itemsets
           for all I ∈ DashedCircle do
24:            if IsPassCompleted(I) then
                   MoveItemset(I, DashedBox)
26:        for all I ∈ DashedBox do
               if IsPassCompleted(I) then
28:                MoveItemset(I, SolidBox)
   ℒ ← SolidBox
```

Direct bit representation: pros & cons

- + Reduce size of transaction database
  - We suppose that D fits in main memory
- + Speed up support counting
  - $I \sqsubseteq T \Leftrightarrow Bitmask(I)$ **AND** $Bitmask(T) = Bitmask(I)$
  -

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I										1				1		
T								1		1	1				1	

- - Candidate generation is more complex
  - $c \leftarrow Bitmask(a)$ **OR** $Bitmask(b) \Leftrightarrow a$ and b are different in one bit
  - (1,2,3,4)←(1,2,3) **OR** (1,2,4), but (1,2,3) and (1,3,5)→Ø
- - Case of m>sizeof(unsigned long long int) should be considered
  - Implement manually or by The GNU Multiple Precision Arithmetic Library, https://gmplib.org/

Parallel DIC

Algorithm 2. ParalDIC(in $\mathcal{B}$, in $minsup$, in M, out $\mathcal{L}$)

▷ Initialize sets of itemsets
SOLID.$init()$; DASHED.$init()$
$k \leftarrow 1$
for all $i \in 0..m-1$ **do**
  $I.fig \leftarrow$ NIL; $I.bitmask \leftarrow 0$
  $I.mask \leftarrow$ SetBit$(I.mask, i)$
  $I.stop \leftarrow 0$; $I.supp \leftarrow 0$; $I.k \leftarrow k$
  SOLID.$push\_back(I)$
$stop_{max} \leftarrow \lceil \frac{n}{M} \rceil$; $stop \leftarrow 0$
FirstPass(SOLID, DASHED)
while not DASHED.$empty()$ **do**
  ▷ Scan database and rewind if necessary
  $stop \leftarrow stop + 1$
  if $stop > stop_{max}$ **then**
    $stop \leftarrow 1$
  $first \leftarrow (stop-1) \cdot M$; $last \leftarrow stop \cdot M - 1$
  $k \leftarrow k + 1$
  CountSupport(DASHED)
  CutDashedCircle(DASHED)
  GenCandidates(DASHED)
  CheckFullPass(DASHED)
$\mathcal{L} \leftarrow \{I \in$ SOLID$, I.fig =$ BOX$\}$

Parallel DIC: support counting

Algorithm 3. `CountSupport(in out` $DASHED$`)`

```
   if DASHED.size() ≥ num_of_threads then
 2:    #pragma omp parallel for
       for all I ∈ DASHED do
 4:        I.stop ← I.stop + 1
           for all T ∈ B[first] .. B[last] do
 6:            if I.mask AND T = I.mask then
                   I.supp ← I.supp + 1
 8: else
       omp_set_nested(true)
10:    #pragma omp parallel for
        num_threads(DASHED.size())
12:    for all I ∈ DASHED do
           I.stop ← I.stop + 1
14:        #pragma omp parallel for reduction(+:I.supp)
           num_threads(⌈num_of_threads / DASHED.size()⌉)
16:        for all T ∈ B[first] .. B[last] do
               if I.mask AND T = I.mask then
18:                I.supp ← I.supp + 1
```

Parallel DIC: pruning

Algorithm 4. `CutDashedCircle(in out` $DASHED$`)`

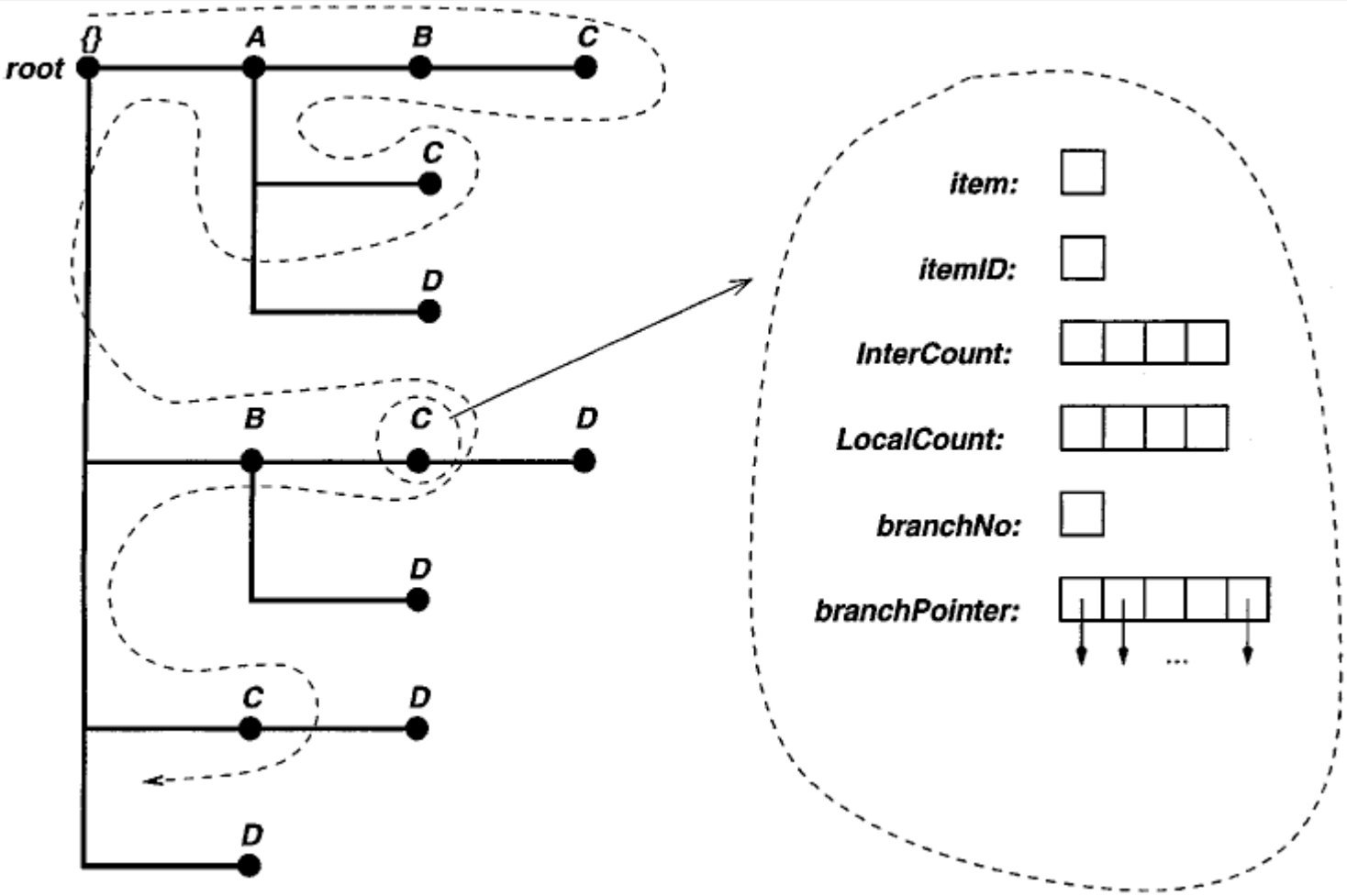
```
    #pragma omp parallel for
 2: for all I ∈ DASHED and I.fig = CIRCLE do
        if I.supp ≥ minsup then
 4:             ▷ Move appropriate itemsets to Dashed Box set
            I.fig ← BOX
 6:     else
                                ▷ Prune clearly infrequent itemset
 8:         supp_max ← I.supp + M · (stop_max − I.stop)
            if supp_max < minsup then
10:             I.fig ← NIL
                        ▷ Prune supersets of infrequent itemset
12:             for all J ∈ DASHED and J.fig = CIRCLE do
                    if I.mask AND J.mask = I.mask then
14:                     J.fig ← NIL
    DASHED.erase(∀I, I.fig = NIL)
```

Parallel DIC: check for full pass completion

Algorithm 5. `CheckFullPass(in out` $DASHED$`)`

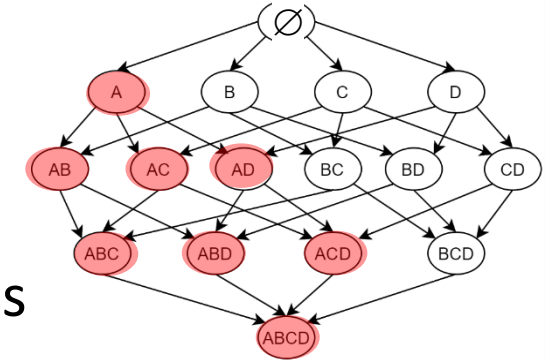
```
    #pragma omp parallel for
2:  for all I ∈ DASHED do
        if I.stop = stop_max then
4:          if I.supp ≥ minsup then
                I.fig ← BOX
6:          SOLID.push_back(I)
            I.fig ← NIL
8:  DASHED.erase(∀I, I.fig = NIL)
```

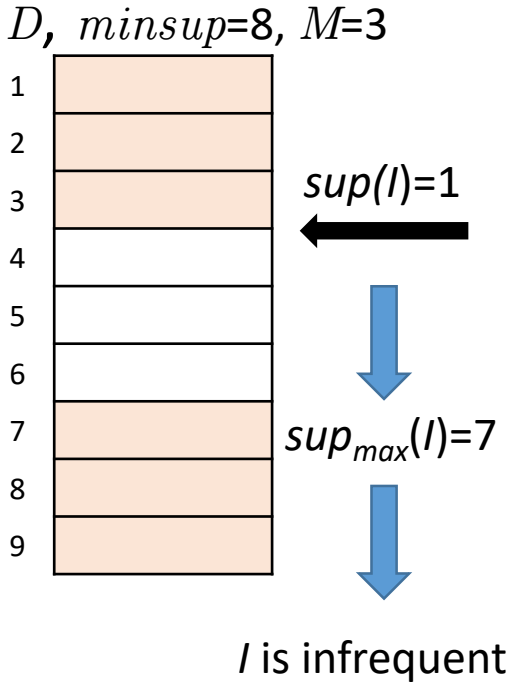
Trie


root
{}
A
B
C
C
D
B
C
D
D
C
D
D
item:
itemID:
InterCount:
LocalCount:
branchNo:
branchPointer:
...

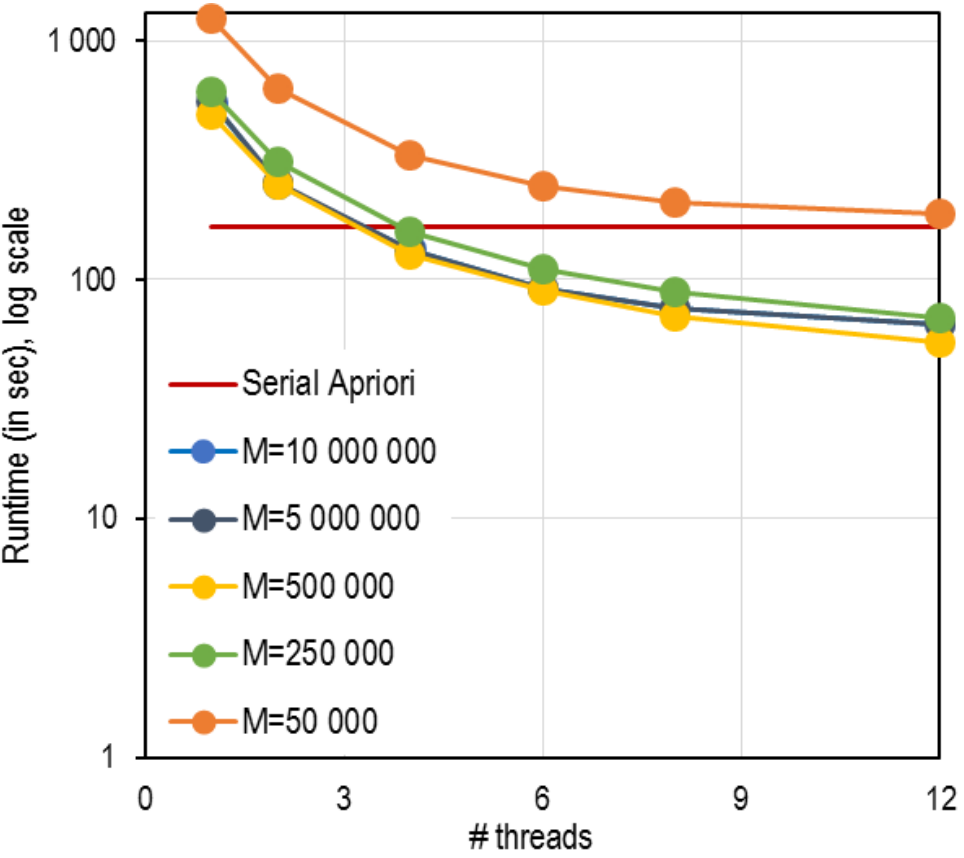
Optimization before parallelization

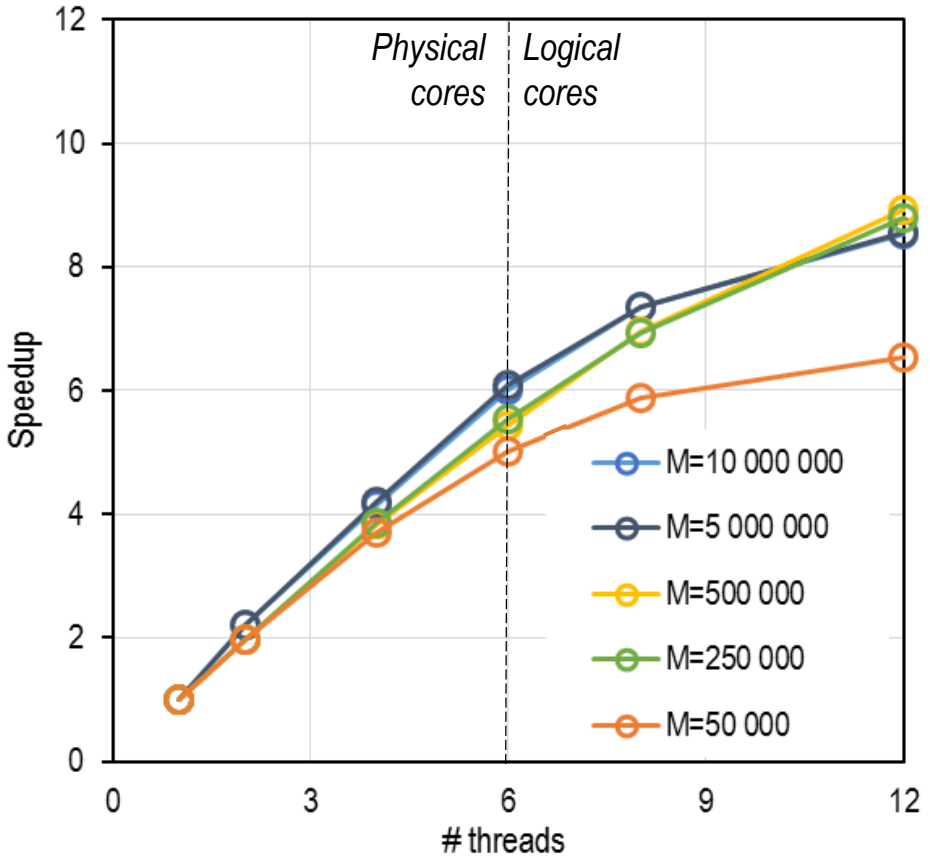
- Make *first full pass* over D
  - find L_1
  - prune all the 1-itemsets that not in L_1
  - finally, get less number of candidate 2-itemsets
- Prune clearly infrequent itemsets
  - Compute sup_{max}, the highest possible support of each itemset at the "stop point" (we suppose that an itemset will occur in the rest transactions)
  - If $sup_{max} < minsup$, then the itemset is infrequent and pruned
  - Prune also supersets of the infrequent itemset





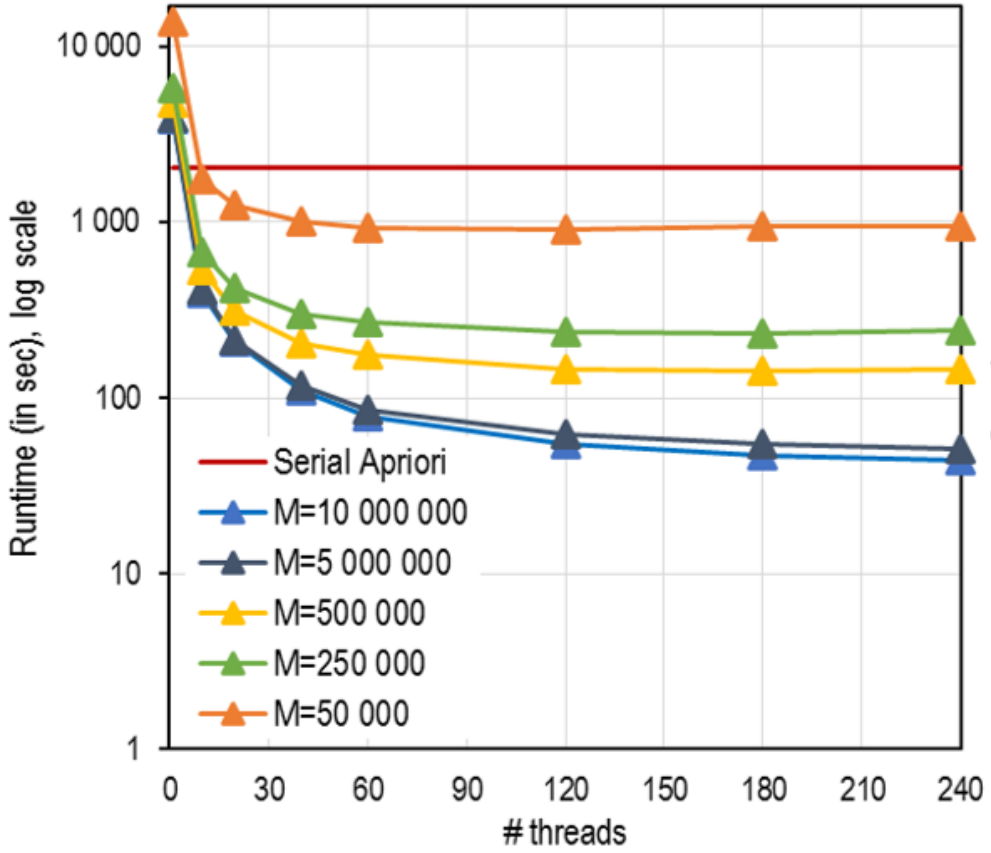
Experiments: Scalability w.r.t. *M* (CPU)

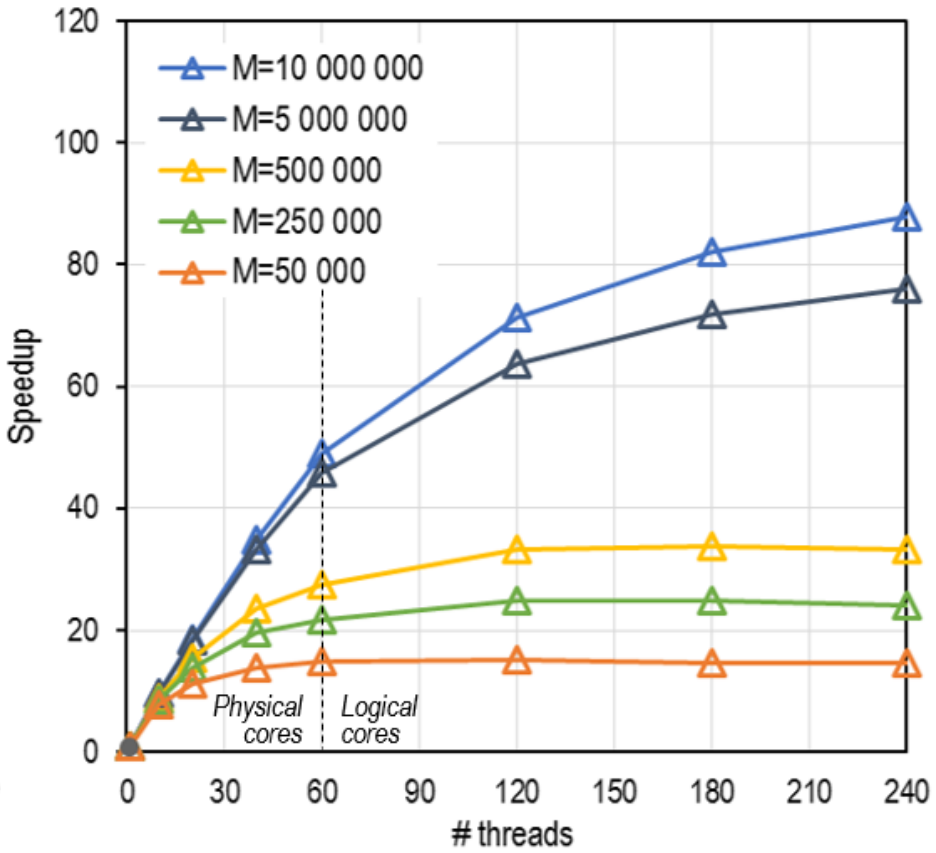




Dataset: *20M*
minsup=0.1

Experiments: Scalability w.r.t. *M* (Phi)





Dataset: *20M*
minsup=0.1