XXI International Conference
Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL'2019),
15–18 October 2019, Kazan Federal University, Kazan, Russia

Big Data Processing and Analytics Inside DBMS

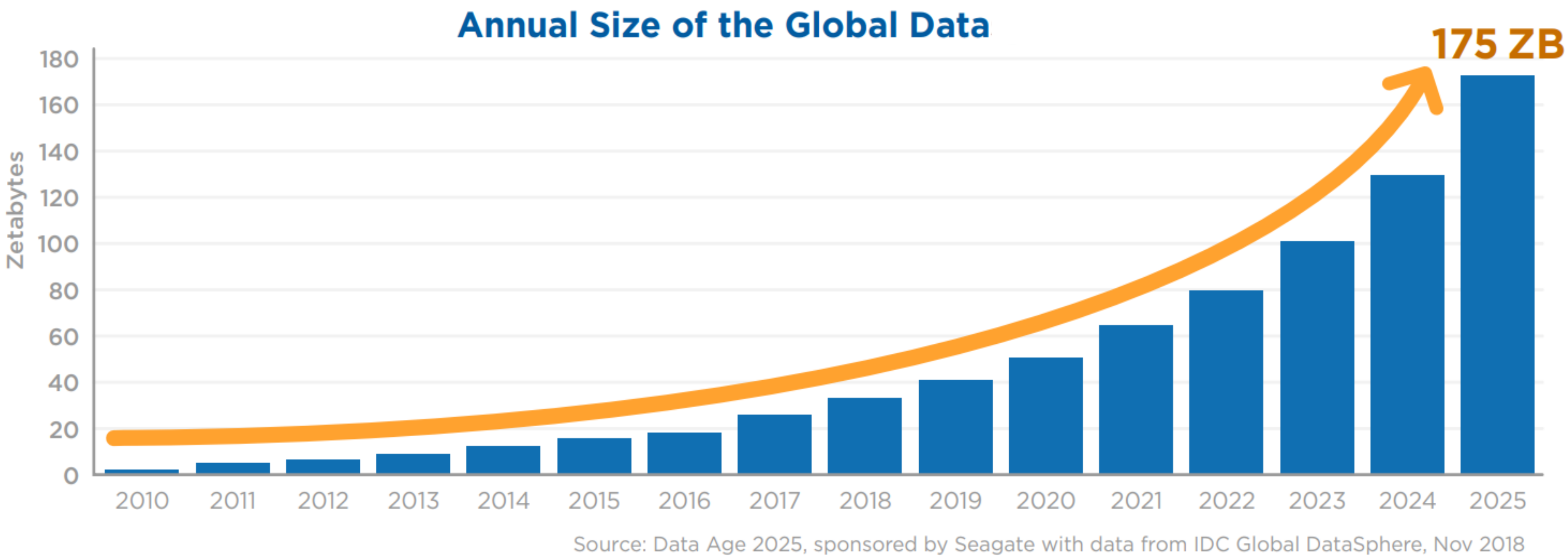
Mikhail Zymbler, Yana Kraeva, Alexander Grents, Anastasiya Perkova, Sachin Kumar

South Ural State University, Chelyabinsk, Russia

This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463) and by the Ministry of Science and Higher Education of the Russian Federation (government orders 2.7905.2017/8.9 and 14.578.21.0265).

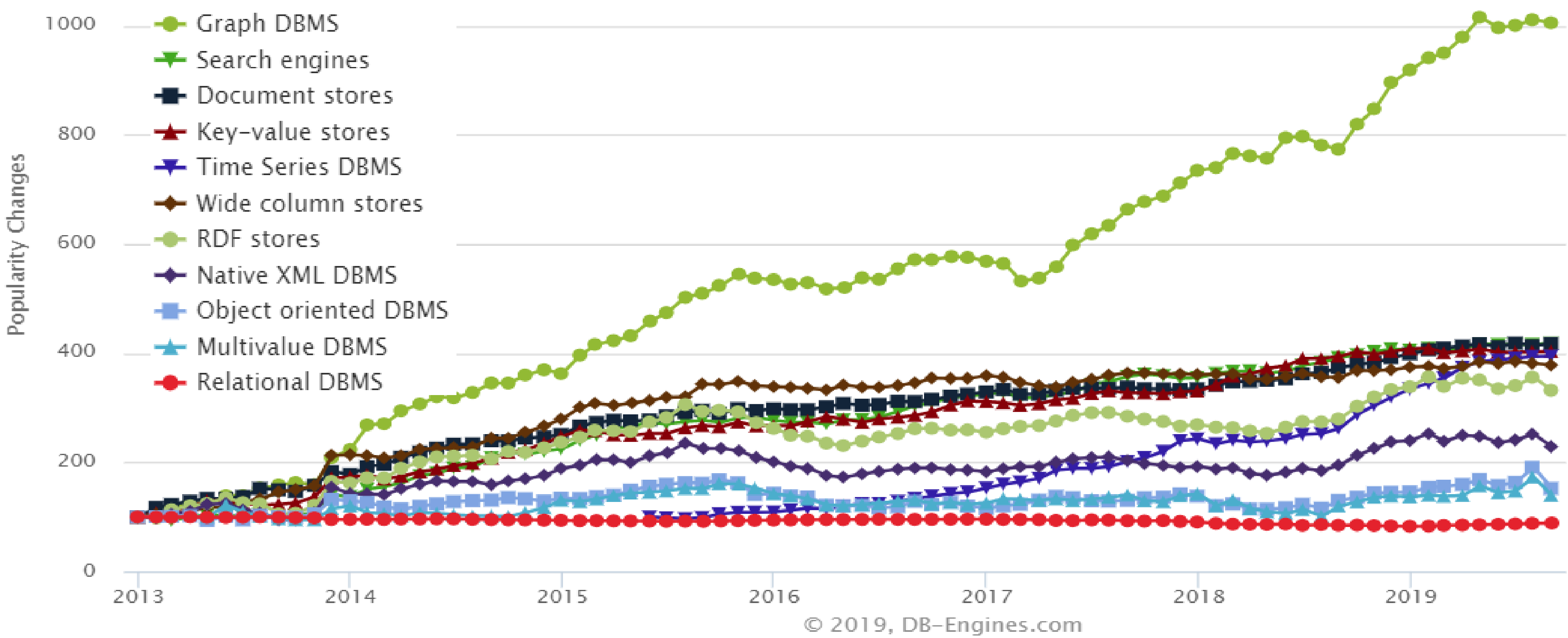
Big Challenges of Big Data

Huge amounts of various data grow fast



What data management systems do we need?

NoSQL systems dominate?

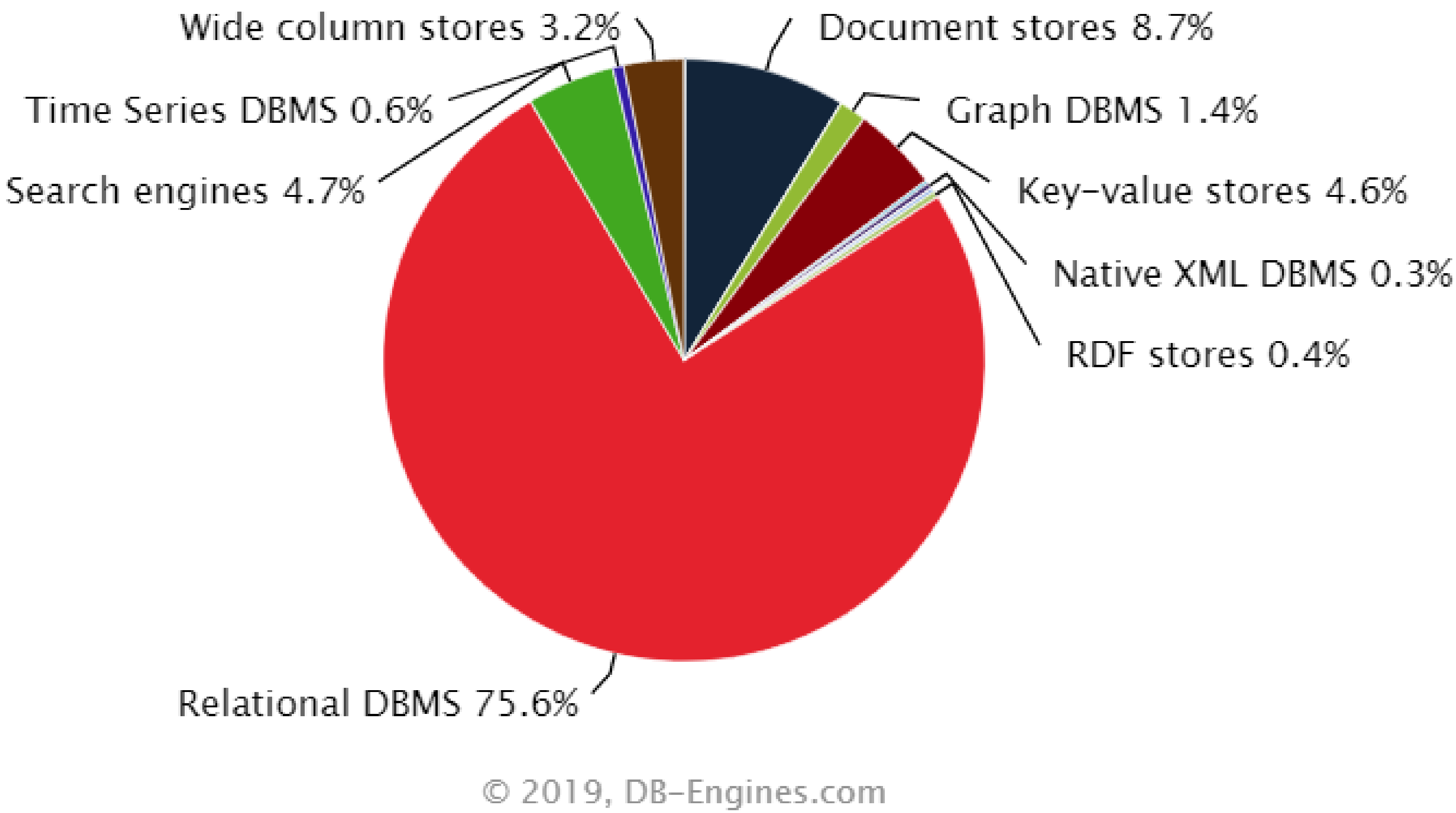


Popularity of systems by their mentions in

- Google Trends
- Newsfeeds (Google, Yandex, etc.)
- Tech discussions (Stack Overflow, etc.)
- Job offers (Simply hired, etc.)
- Professional nets (LinkedIn, etc.)
- Social nets (Twitter, etc.) ,

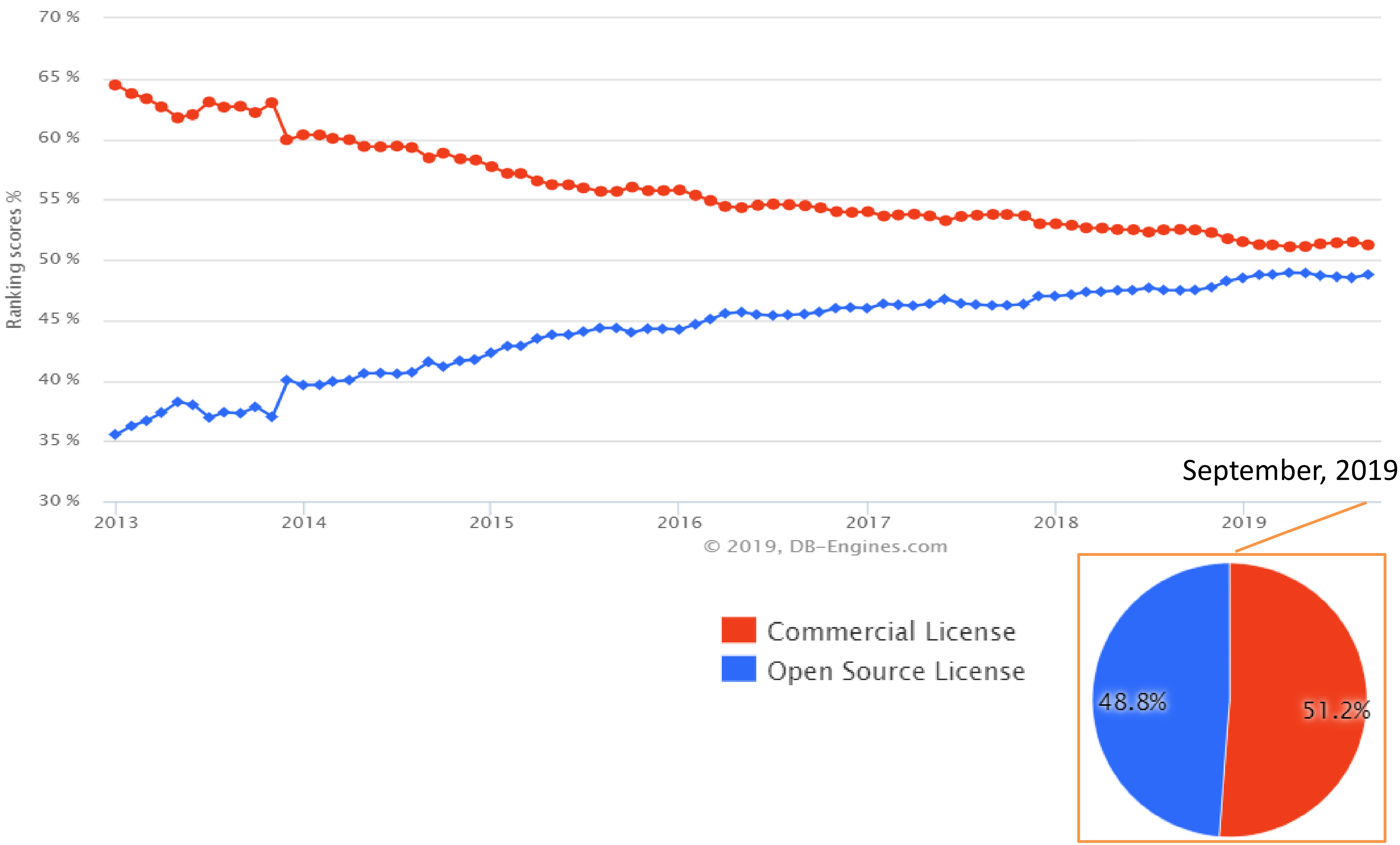
September 2019

Indeed, No! RDBMSs dominate!

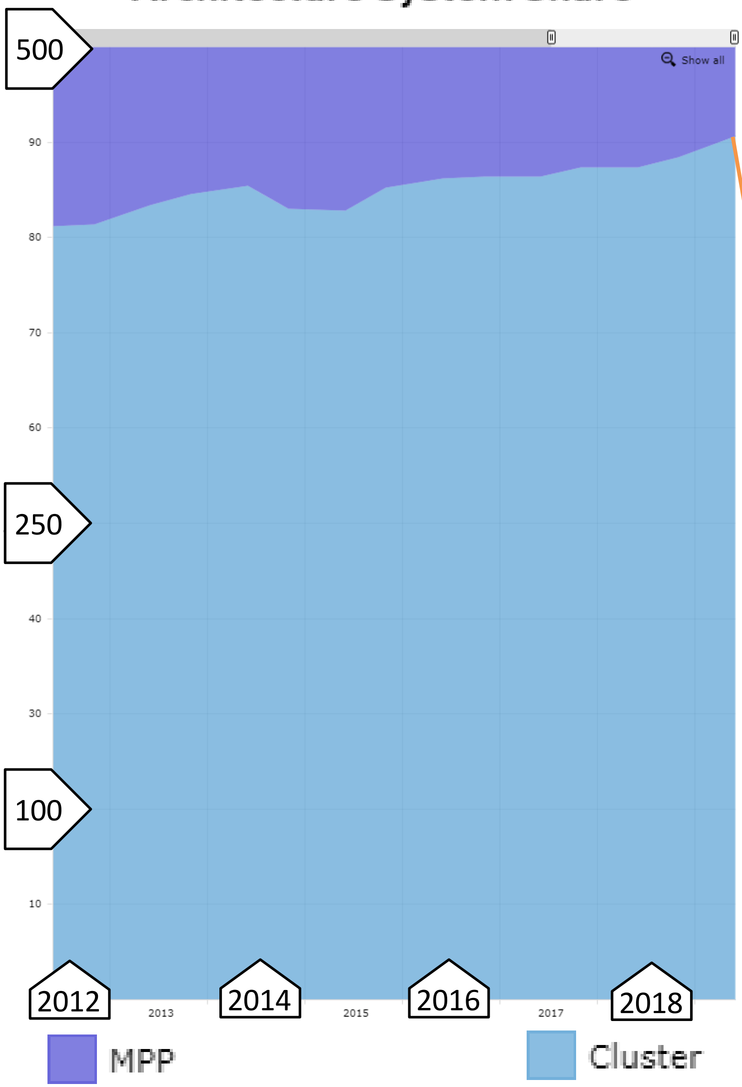


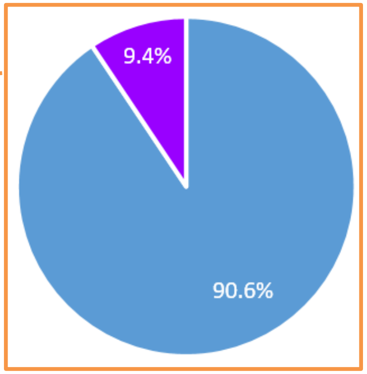
Popularity of systems per category, September 2019

Open source vs Commercial



TOP 500 The List.: HPC clusters dominate





https://top500.org

Cluster

MPP

Why embed parallelism into serial DBMS?

- **Proprietary parallel DBMSs** are very expensive to buy or develop from scratch

PDBMS	Price per year
IBM DB2 Parallel Edition	**$11,016**
Oracle Real Application Cluster	**$29,692**
Teradata Data Warehouse Appliance	**$4,597,784**

- **Open-source DBMSs** are mostly serial but can be (softly) modified to make it parallel
- At last (but not least), in Russia, **import substitution** matters!

How can we parallelize query execution?

**Partition data and apply SPMD paradigm
(Single Program, Multiple Data)**

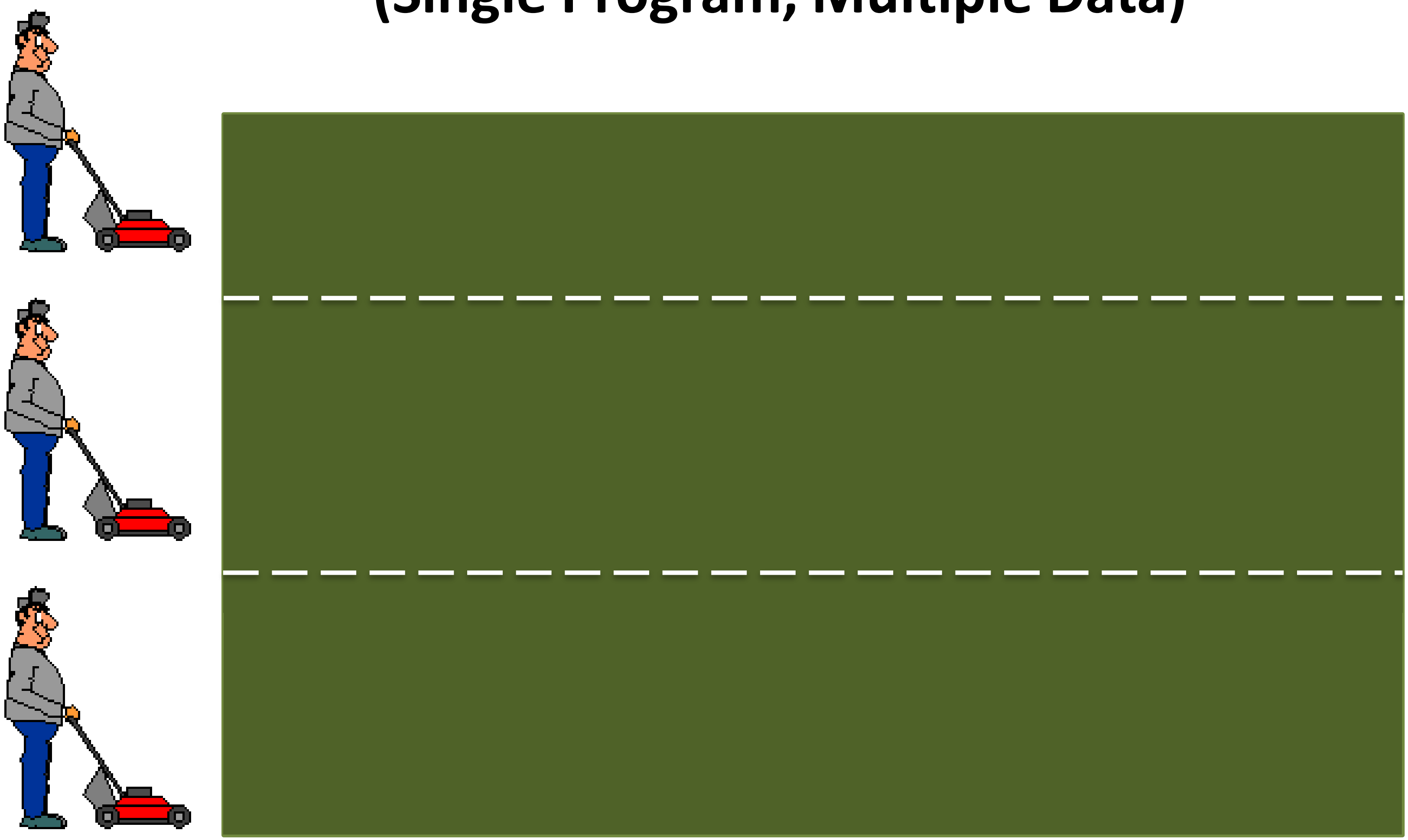
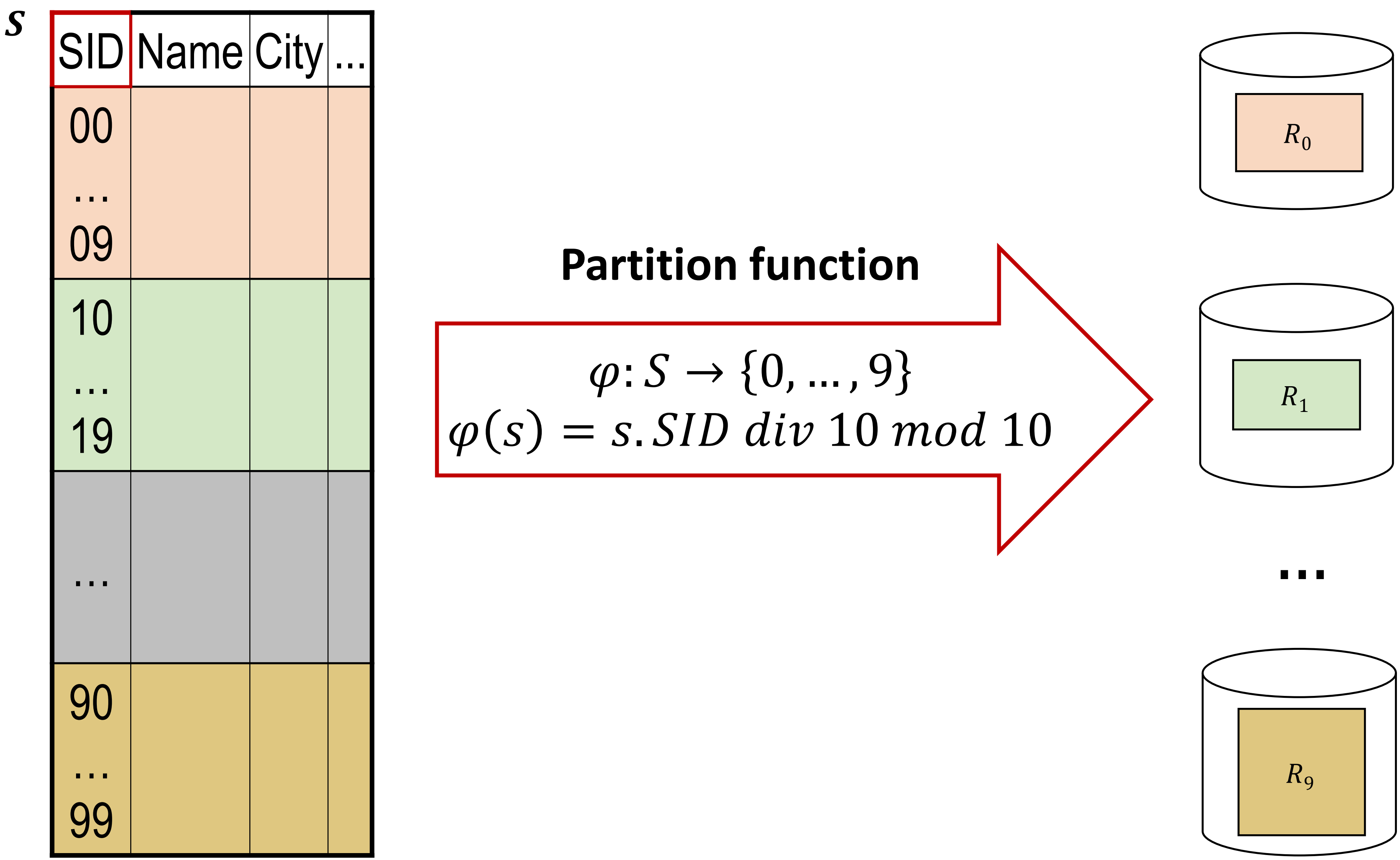
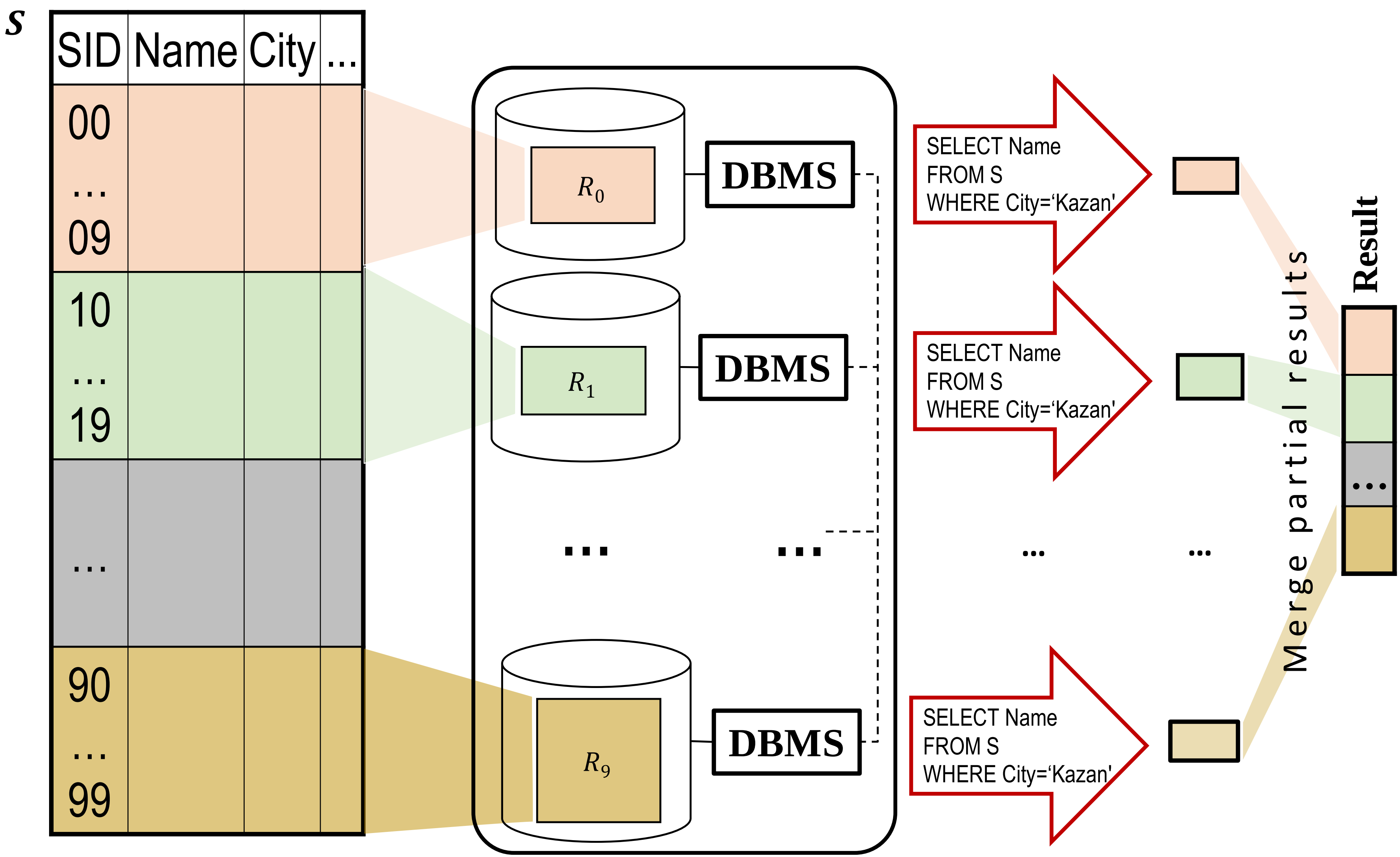


Table partitioning



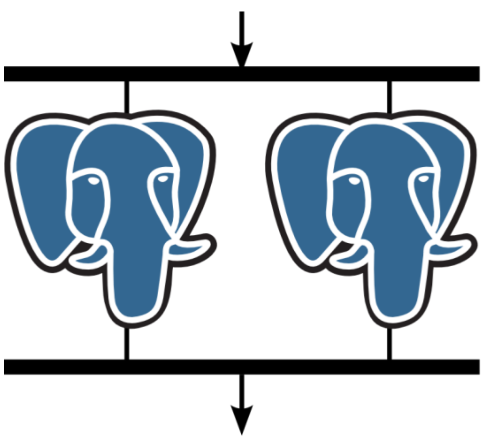
Parallel query execution



From serial to parallel DBMS

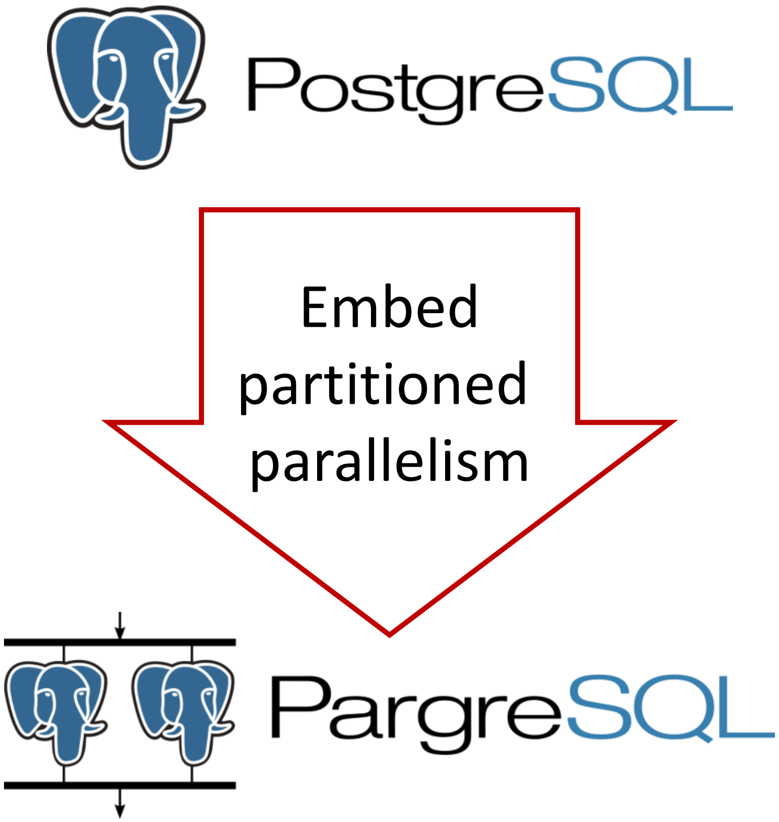
PostgreSQL

Embedding partitioned parallelism



PargreSQL

Basic “How to?”s



1. Partition a table
2. Disseminate a query
3. Merge results
4. Exchange data
5. Run an application

Table partitioning

PostgreSQL

CREATE TABLE T
(A int, B int);

PargreSQL

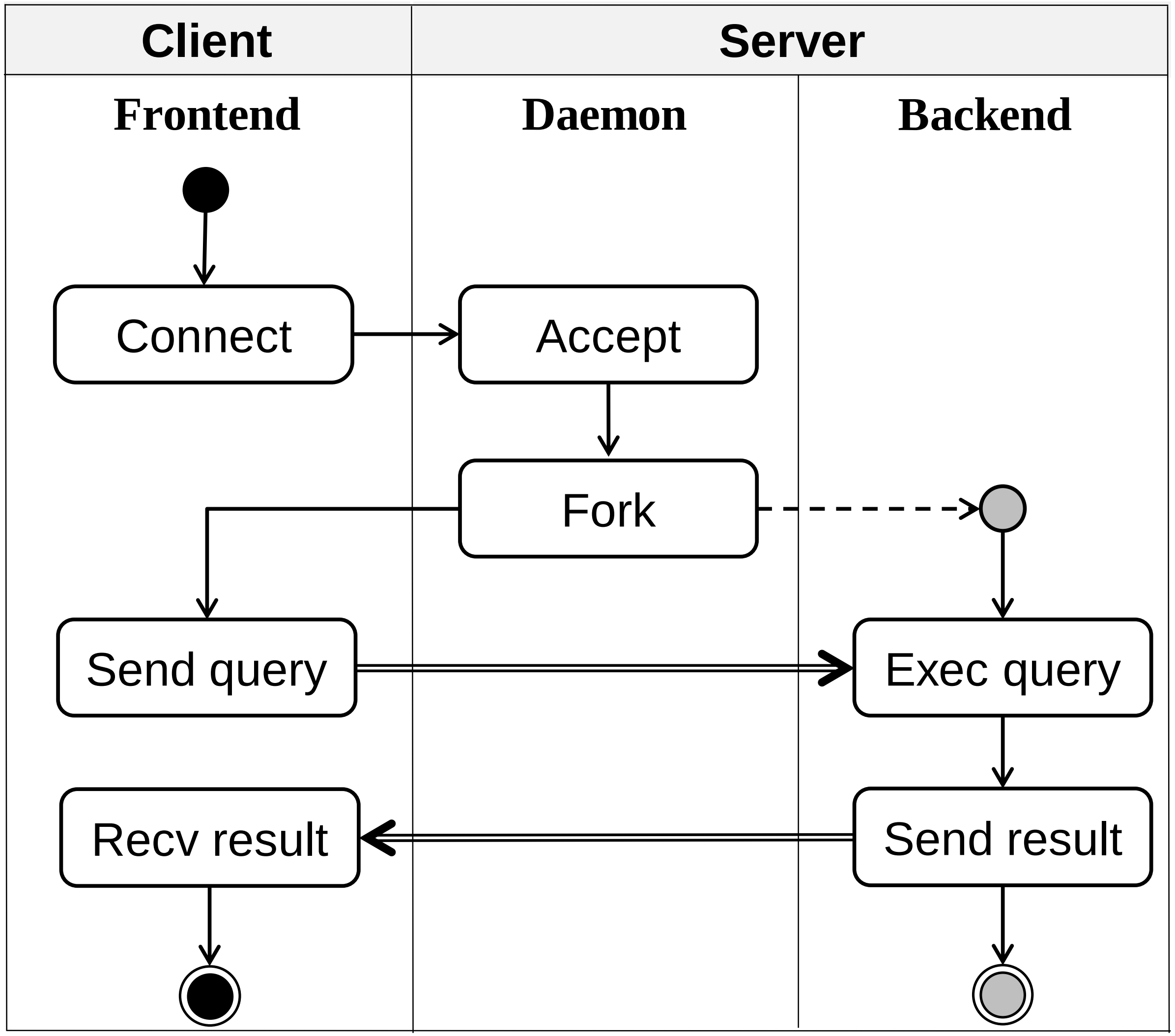
CREATE TABLE T
(A int, B int)
WITH (FRAGATTR = B);

Partitioning function:

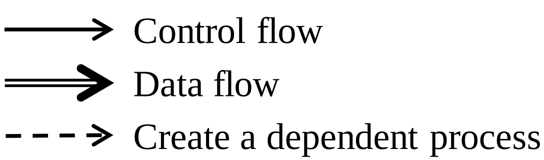
$$\varphi(t) = t.B \ mod \ P$$

where P is number of servers

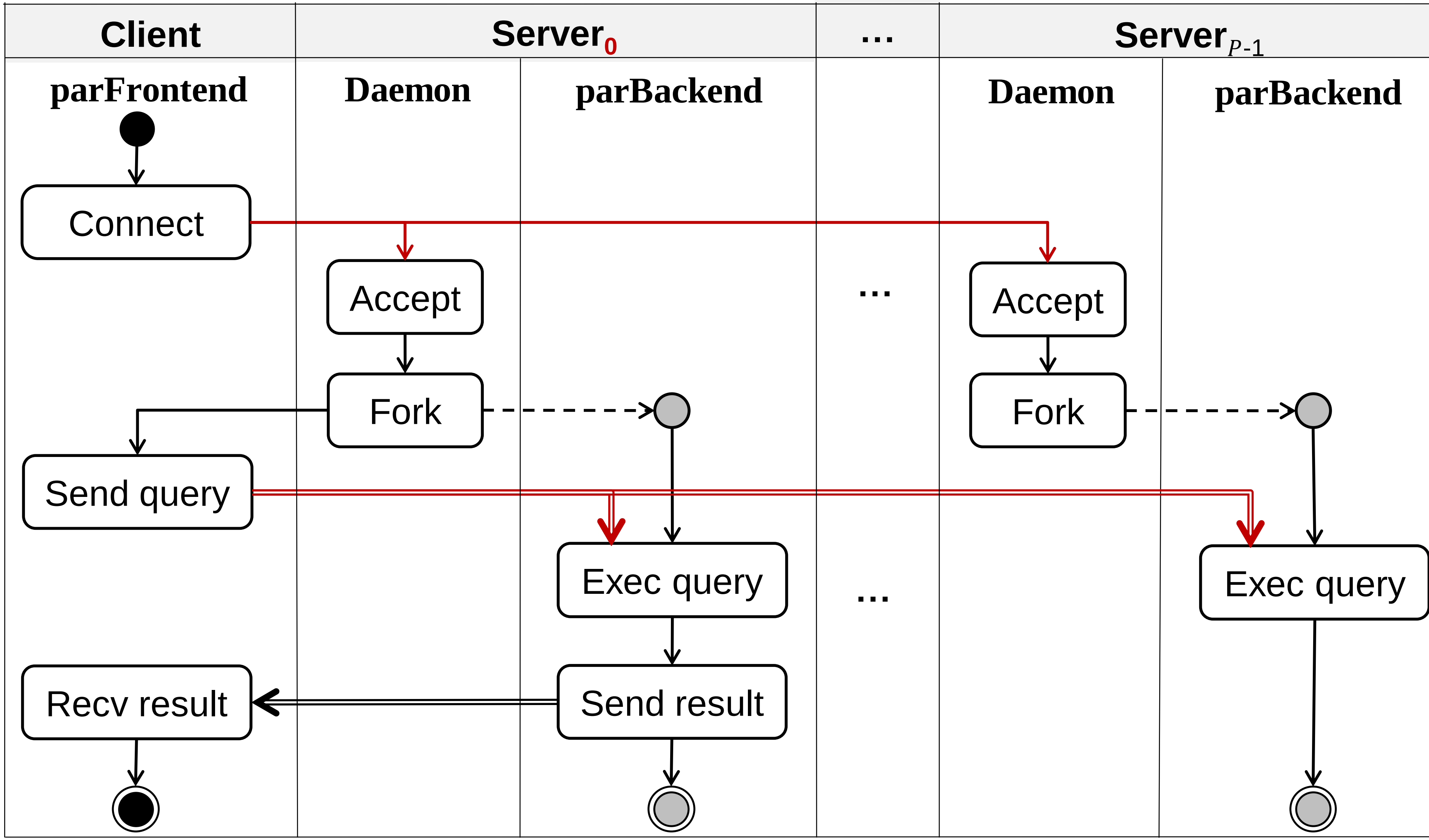
Processes



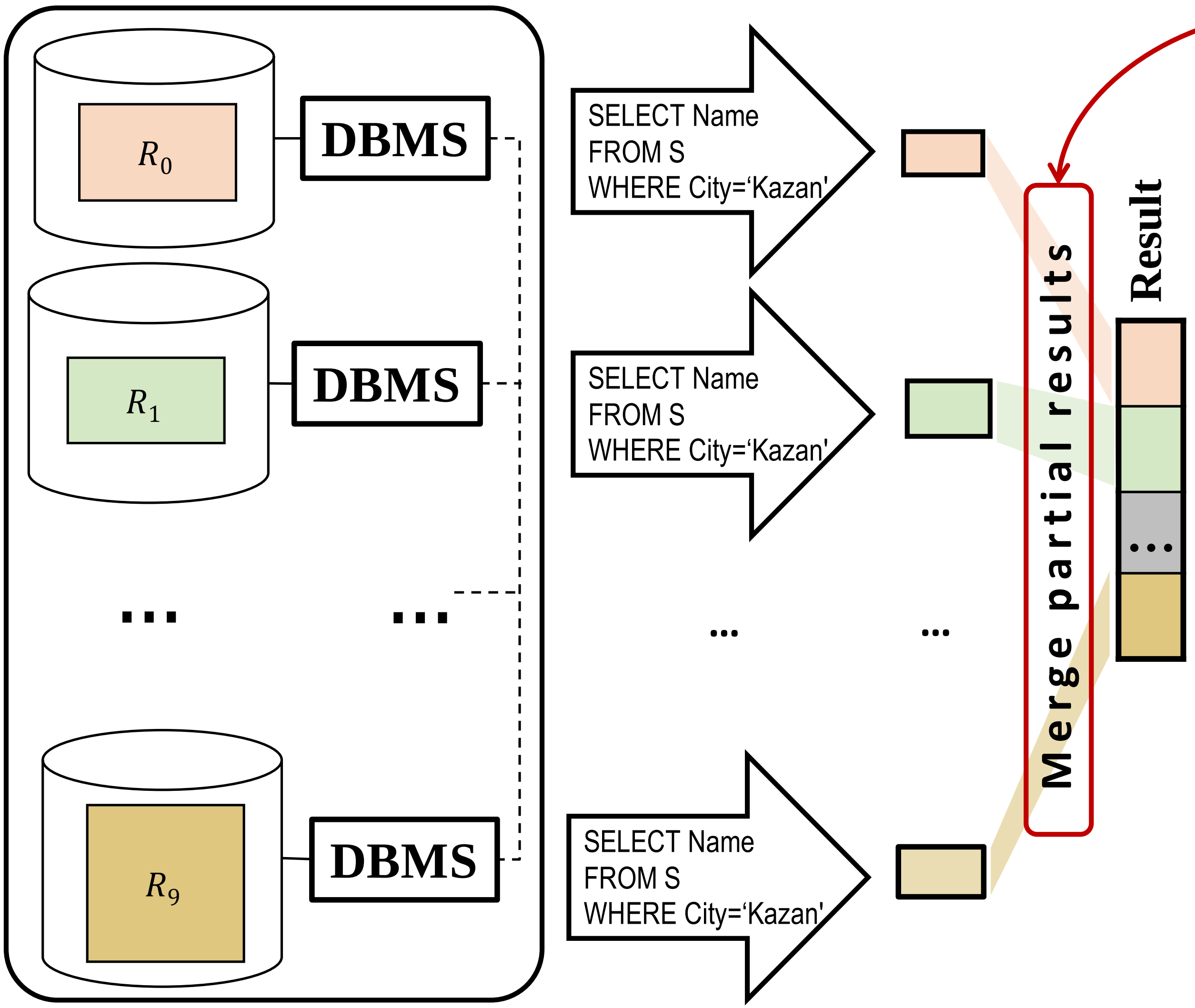
- **Frontend**
  client application with PostgreSQL's libpq-fe library
- **Daemon**
  server process to accept client's connection and create Backend instance to process client's queries
- **Backend**
  server process to execute client's query and send the results to the client



PargreSQL Processes



Merging results



How 0th server **merges partial results** and **sends the result to a client**?

An example on data exchanges: join tables

S

SID	Name	City	…
00	Horns&Hoofs	Odessa	
11	RaznoExport	Moscow	
22	UralTrak	Chelyabinsk	
…	…	…	

$$\varphi(s) = s.SID\ div\ 10\ mod\ 10$$

P

PID	Name	Price	…
00	Nail	100	
11	Bolt	75	
99	Screw	10	
…	…	…	

$$\varphi(p) = p.PID\ div\ 10\ mod\ 10$$

SP

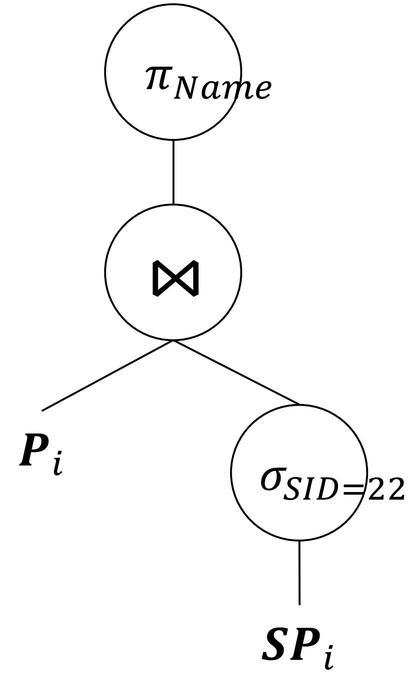
SID	PID	Qty
00	02	5000
11	01	500
22	00	50
11	02	100
22	02	500

$$\varphi(sp) = sp.PID\ div\ 10\ mod\ 10$$

Data exchanges not needed

Get the names of parts supplied by supplier with ID 22:

```
SELECT Name
FROM P, SP
WHERE P.PID=SP.PID AND SP.SID=22
```



P

PID	Name	Price	...
00	Nail	100	
11	Bolt	75	
99	Screw	10	
...	...	...	

$\varphi(p) = p.PID\ div\ 10\ mod\ 10$

SP

SID	PID	Qty
00	02	5000
01	11	500
22	99	50
...	...	...

$\varphi(sp) = sp.PID\ div\ 10\ mod\ 10$

All records are processed by the servers on which these records are stored, so **no records to be transferred**

Data exchanges needed

Get the names of suppliers who supply part with ID 99:

```
SELECT Name
FROM S, SP
WHERE S.SID=SP.SID AND SP.PID=99
```

$\boldsymbol{S}$

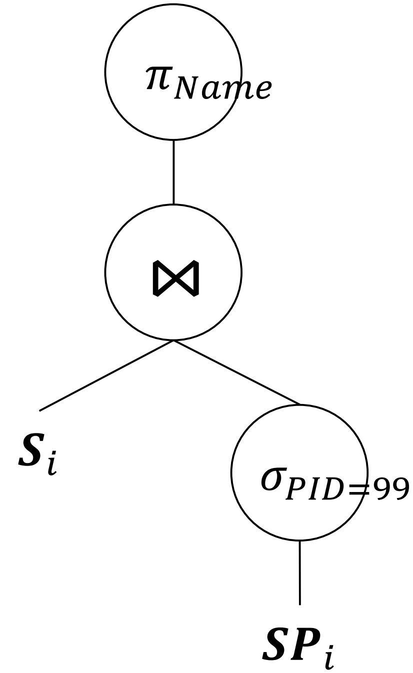
SID	Name	City	…
00	Horns&Hoofs	Odessa	
11	RaznoExport	Moscow	
22	UralTrak	Chelyabinsk	
…	…	…	

$\varphi(s) = s.SID\ div\ 10\ mod\ 10$

$\boldsymbol{SP}$

SID	PID	Qty
00	02	5000
01	11	500
22	99	50
…	…	…

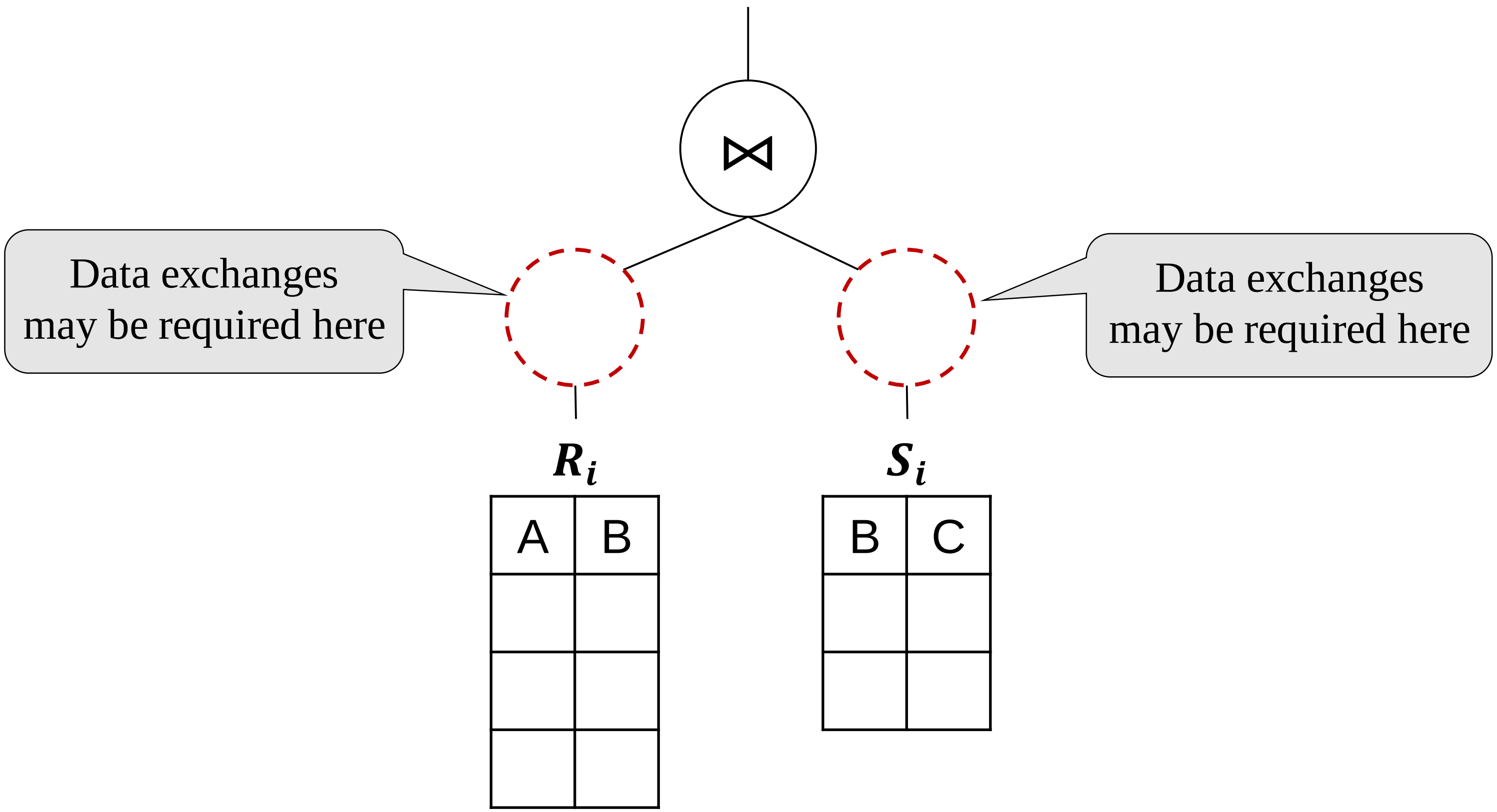
$\varphi(sp) = sp.PID\ div\ 10\ mod\ 10$



This record is **stored on the 9th server** but to be processed it **must be transferred to the 2th server**

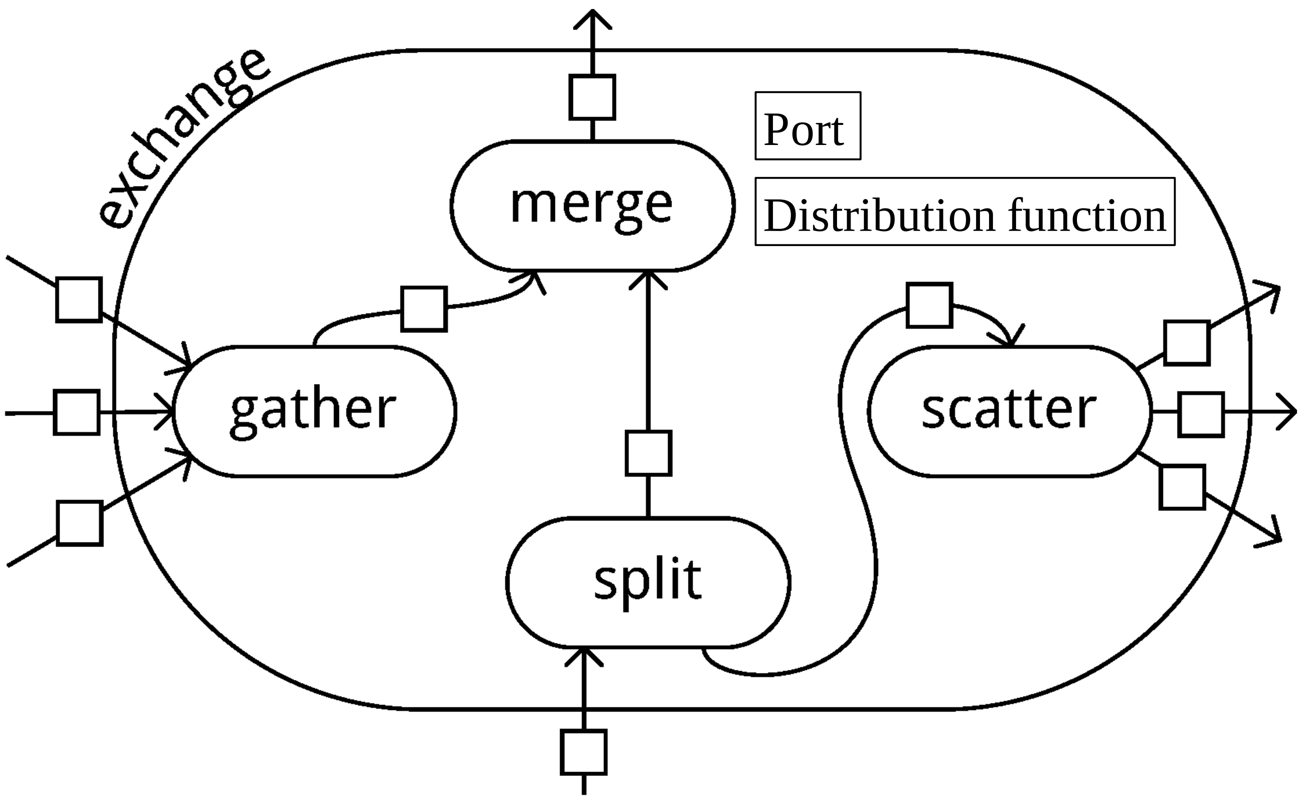
So, we need exchanges

$$R \bowtie S$$

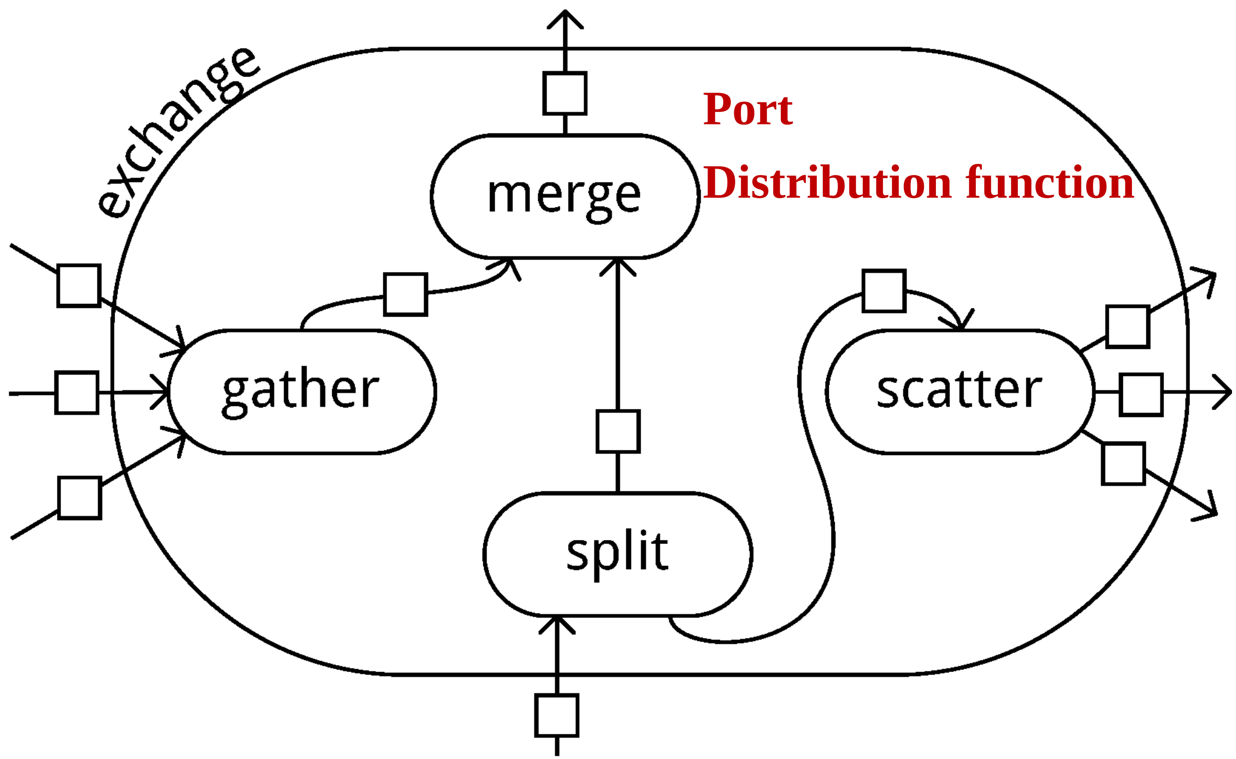


Did you want an EXCHANGE? We have it…

It is an **empty operator of relational algebra**,
which is **executed by the serial DBMS engine in normal mode**
but it **encapsulates the parallelism**

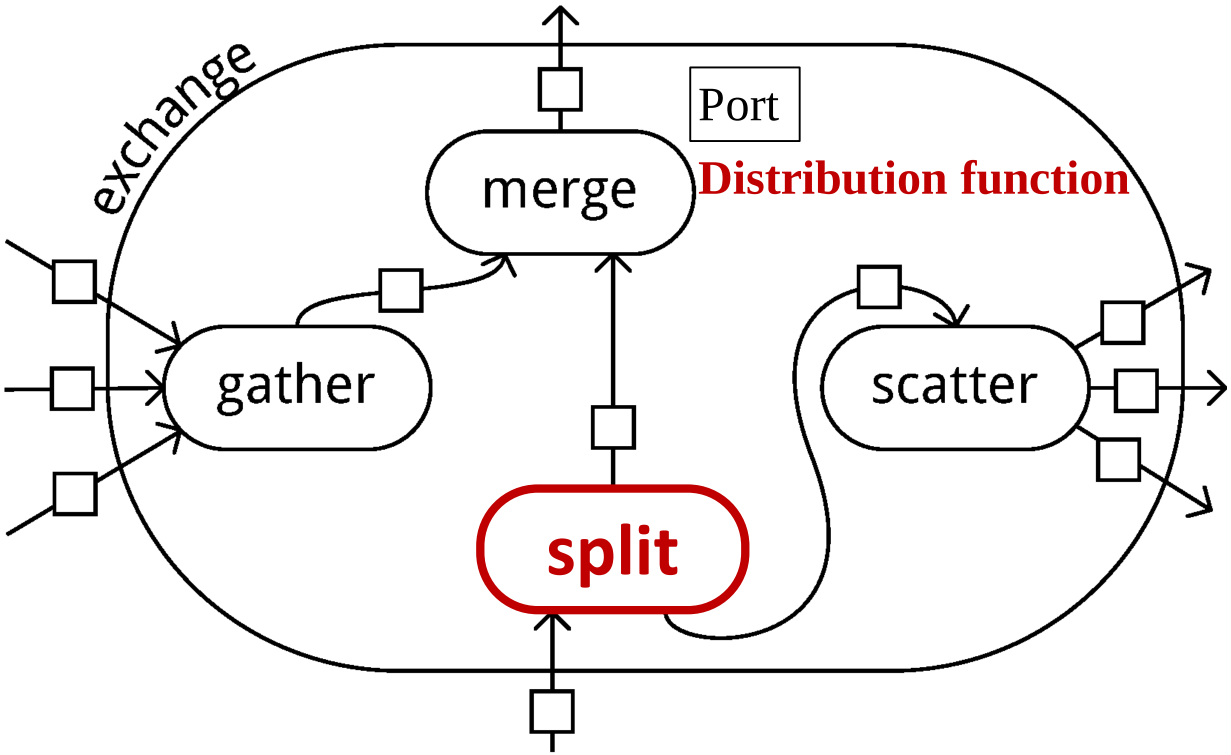


EXCHANGE operator



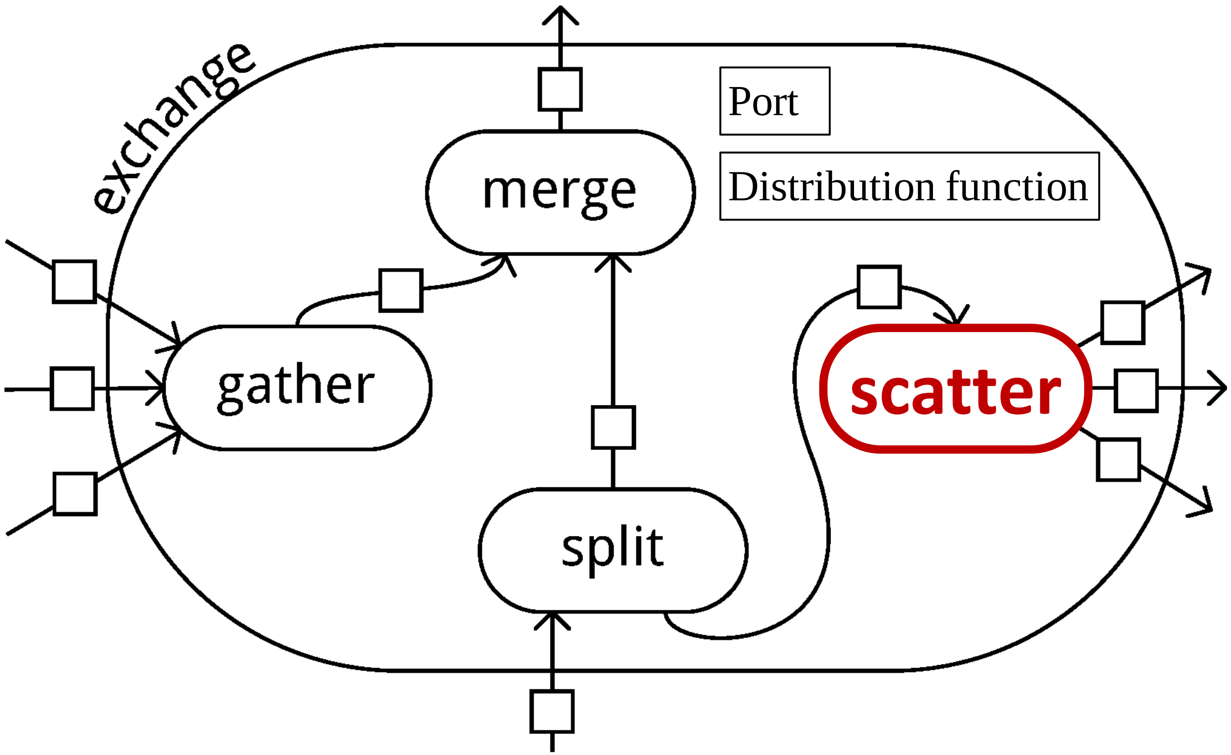
- **Port** is a serial number of EXCHANGE in a query
- **Distribution function** $\psi: R \rightarrow \{0, \ldots, P-1\}$ calculates a server where the given record must be processed

EXCHANGE operator



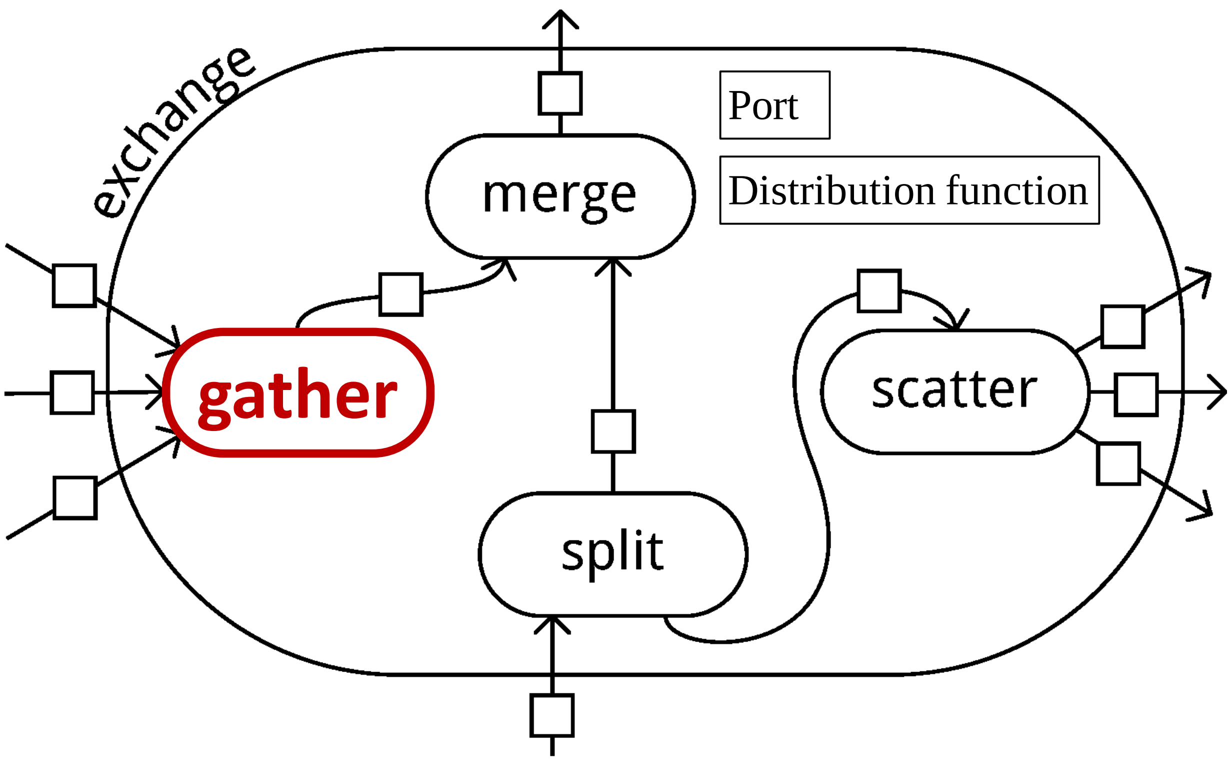
- **Split** calculates the distribution function ψ from the given record
- If ψ returns current server then the record is passed to Merge otherwise it is passed to Scatter

EXCHANGE operator



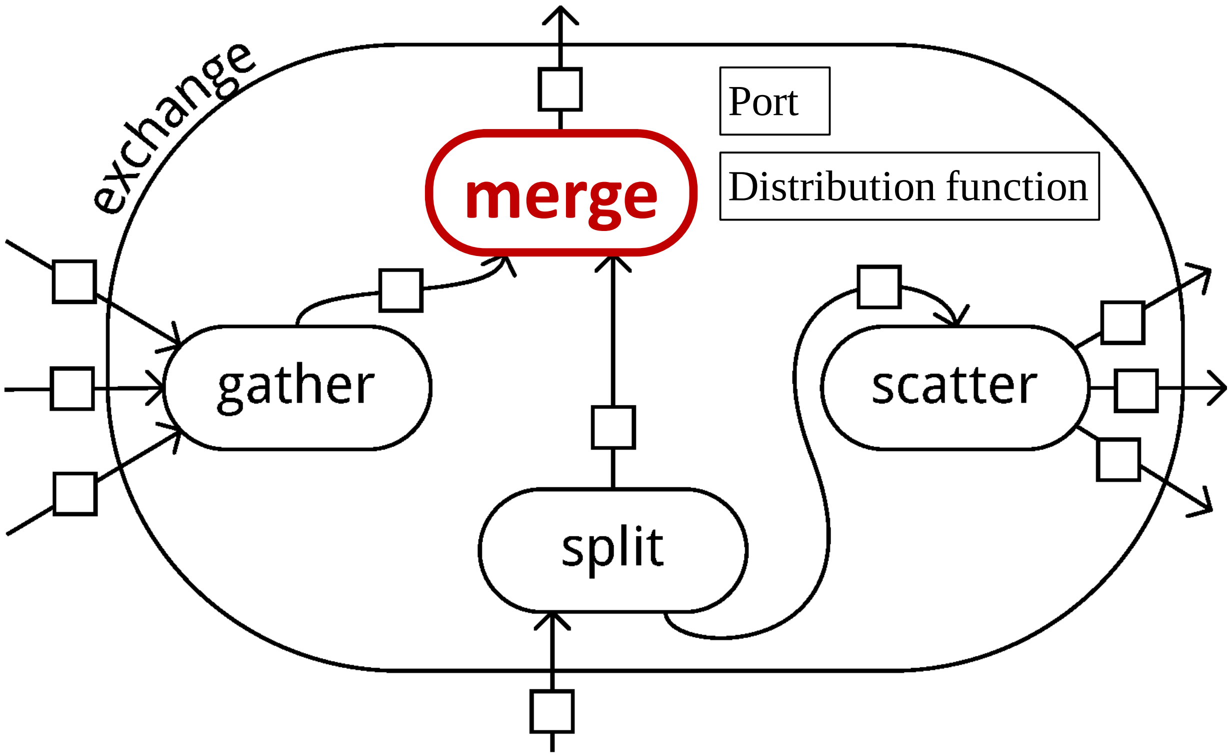
- **Scatter** transfers the given record to alien servers

EXCHANGE operator



- **Gather** receives the current server's records from alien servers and pass them to Merge

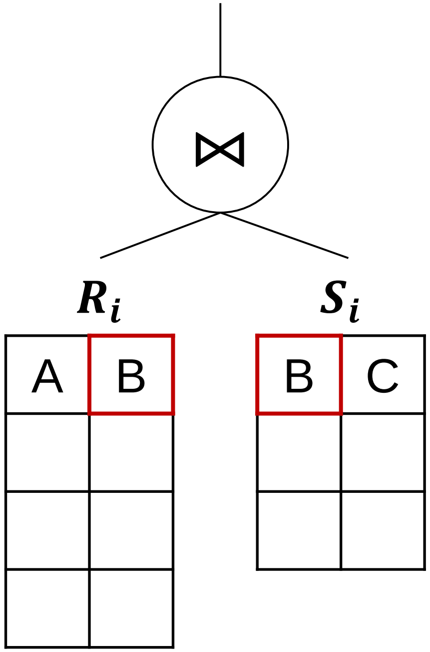
EXCHANGE operator

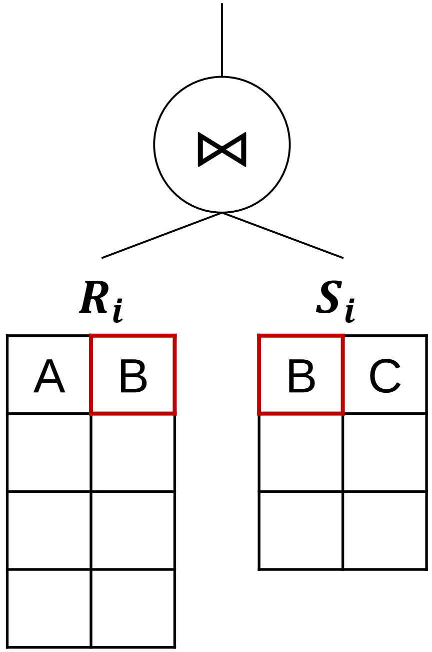


- **Merge** outputs records
  from Gather and Split by rotation

Query parallelization, case 1

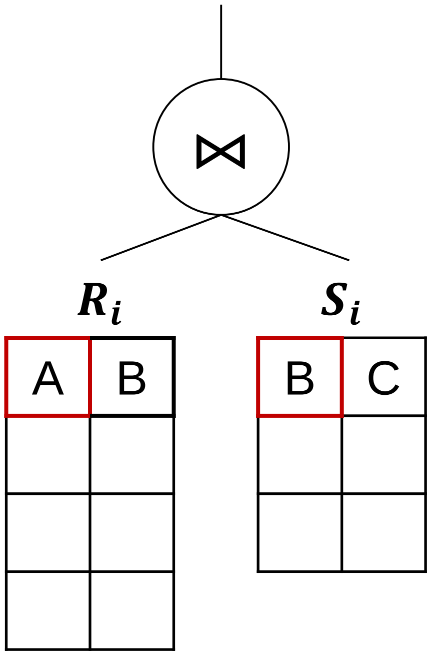
R and S are partitioned by the join attribute using the same partition function

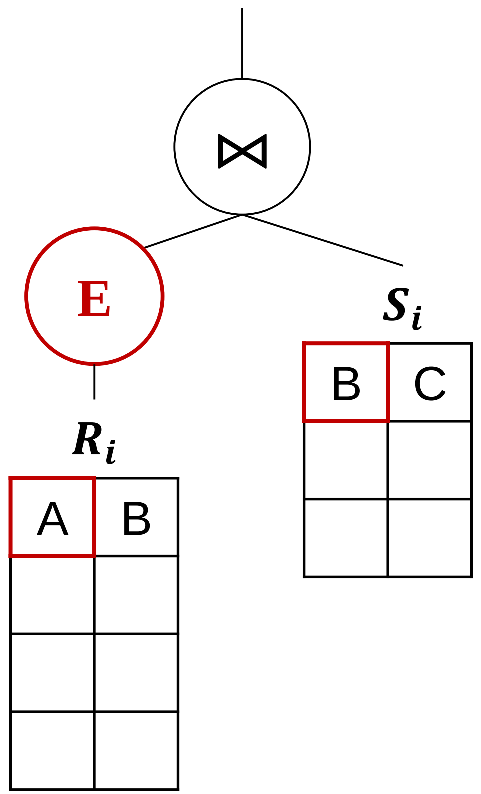




Query parallelization, case 2

R is partitioned by A and S is partitioned by B using the φ_S partition function



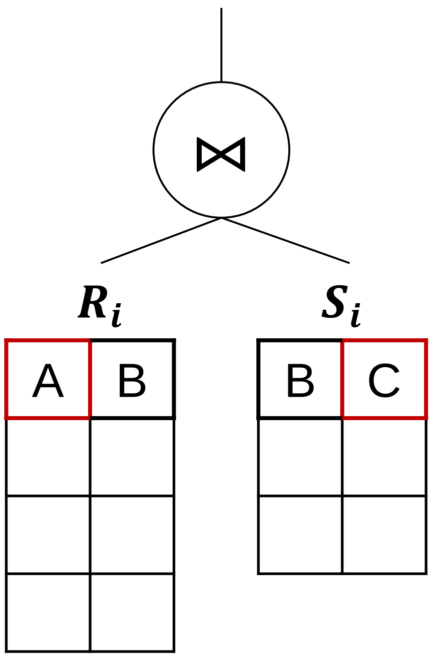


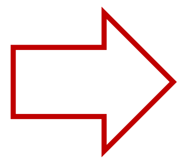
Distribution function:

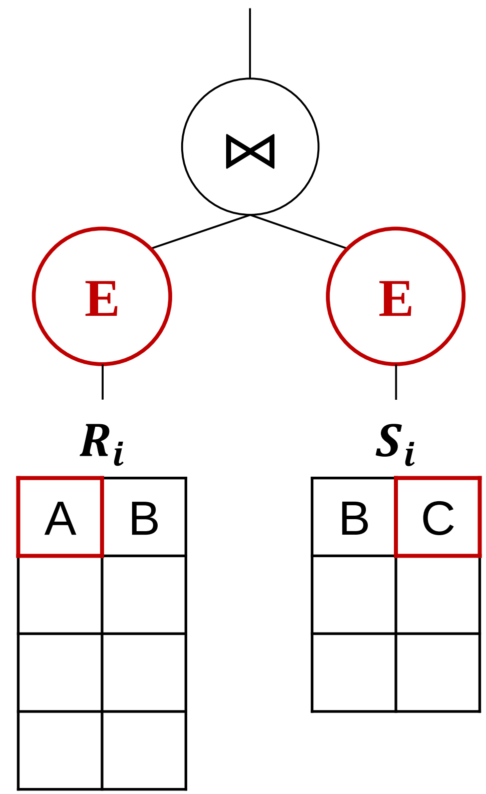
$\psi_R(r.B) = \varphi_S(r.B)$

Query parallelization, case 3

R is partitioned by A and S is partitioned by C







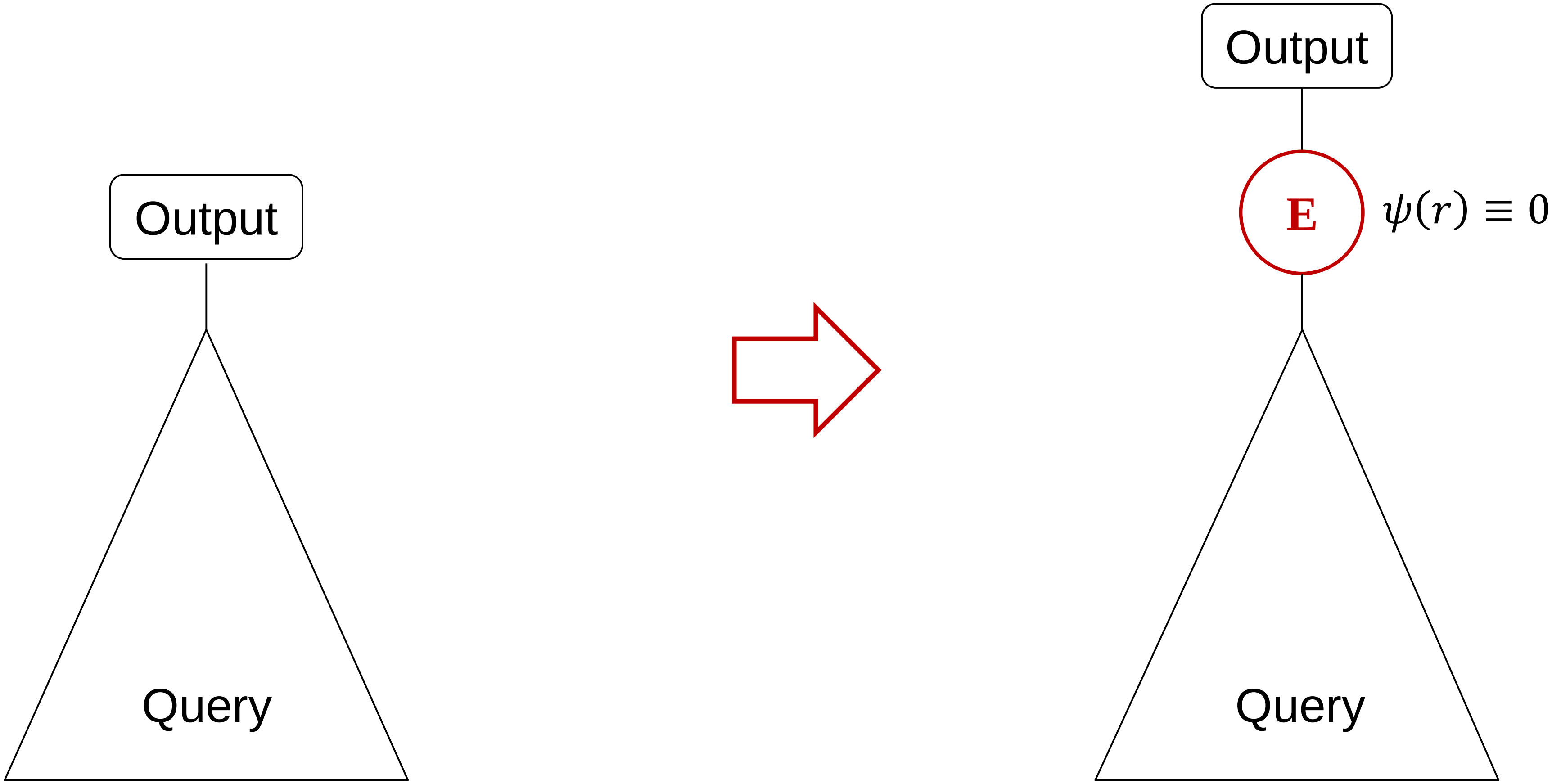
Distribution functions:

$$\psi_R(r.B) = f(r.B)$$
$$\psi_S(s.B) = f(s.B)$$

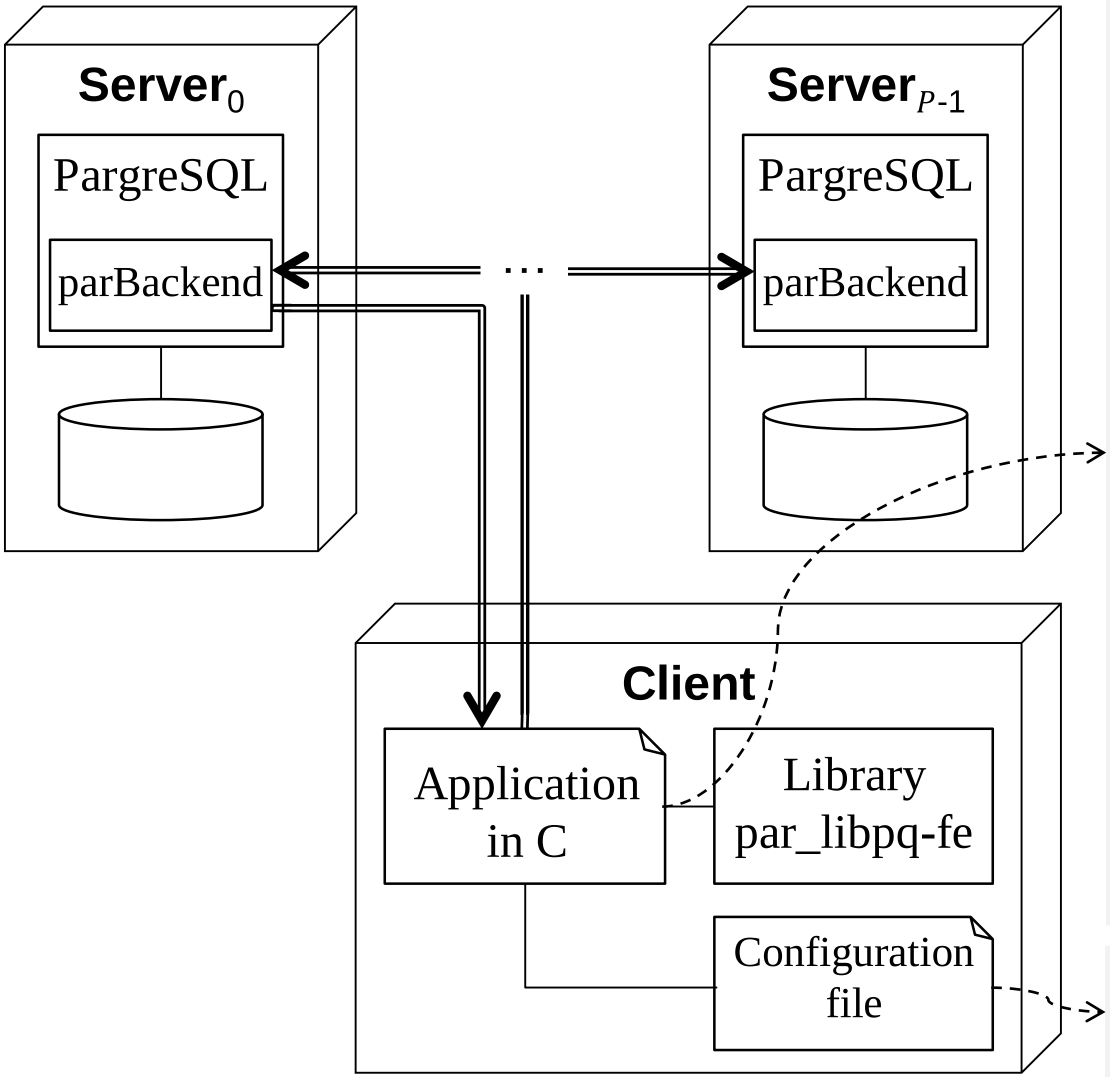
$$f: D_B \rightarrow \{0, \ldots, P-1\}$$

Merging partial results

To merge partial results on the 0th server, we just need **one more EXCHANGE with trivial distribution function**



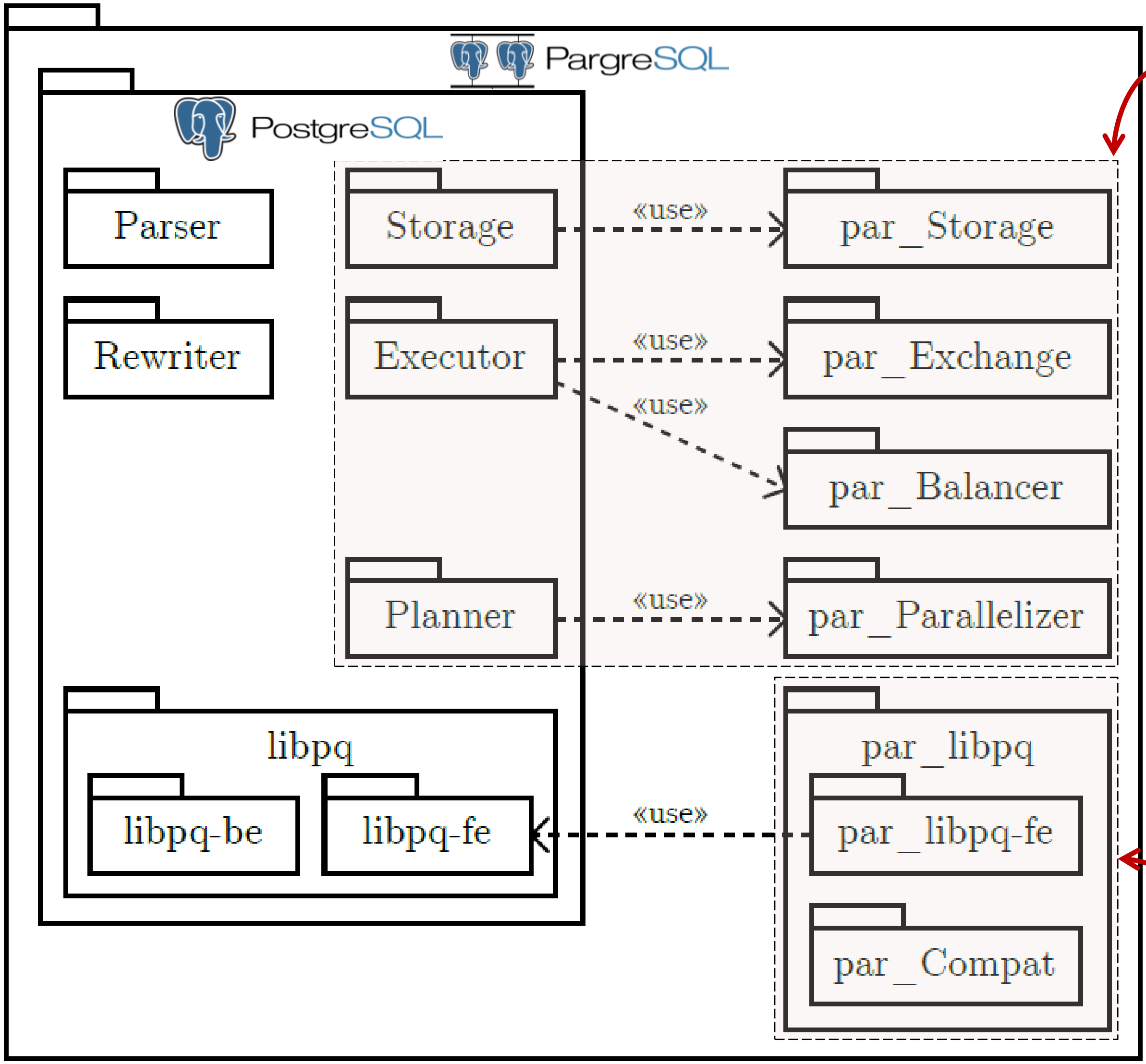
Running PargreSQL application



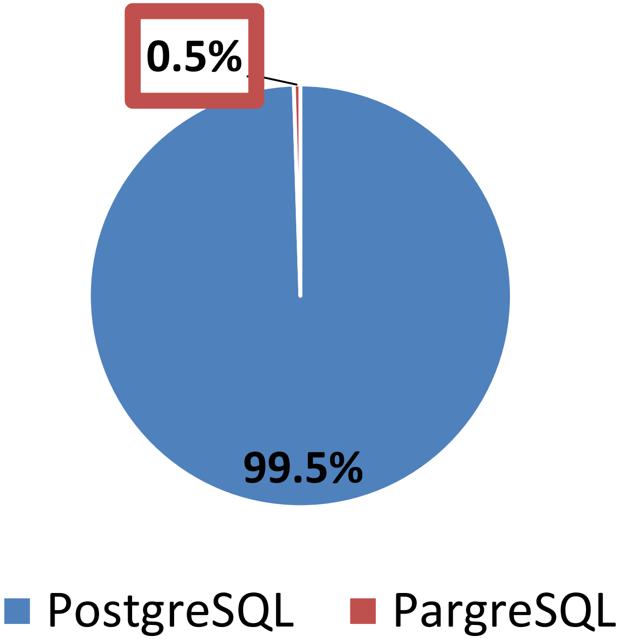

```
#include <par_libpq-fe.h>

PGconn *conn;
char *conninfo = "dbname=… hostaddr=… port=… ";
PGresult *res;

void main(int argc, char * argv[]) {
  conn = PQconnectdb (conninfo);

  res = PQexec(conn,
    "CREATE TABLE R (A INT, B INT)
  WITH (FRAGATTR = B);");

  res = PQexec(conn,
    "INSERT INTO R VALUES (2501, 1755);
    INSERT INTO R VALUES (2, 4);
    INSERT INTO R VALUES (8, 1); ");

  res = PQexec(conn, "SELECT * FROM R;");
  PQprint(stderr, res);

  PQclear(res);
  PQfinish(conn);
}
```

```
# Connection strings for each PargreSQL instance
dbname=postgres hostaddr=10.4.5.204 port=5432
dbname=postgres hostaddr=10.4.5.205 port=5432
```

ParqreSQL is just a prototype



Modified PostgreSQL source codes



New ParqreSQL source codes

PargreSQL is just a prototype…

TPC-C test results

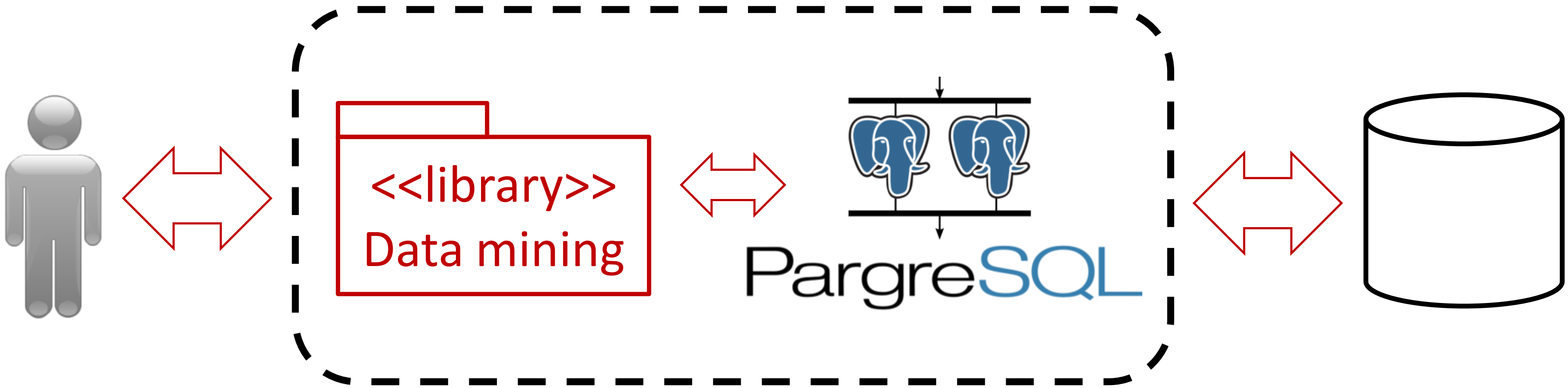
Rank	Company	System	Performance (tpmC)	DBMS	OS
1	Oracle	SPARC SuperCluster with T3-4 Servers	30 249 688	Oracle Database 11g R2 Enterprise Edition w/RAC w/Partitioning	Oracle Solaris 10 09/10
2	IBM	IBM Power 780 Server Model 9179-MHB	10 366 254	IBM DB2 9.7	AIX Version 6.1
3	Oracle	Sun SPARC Enterprise T5440 Server Cluster	7 646 486	Oracle Database 11g Enterprise Edition w/RAC w/Partitioning	Sun Solaris 10 10/09
	SUSU	Tornado SUSU supercomputer	2 202 531	PargreSQL	Linux CentOS 6.2
4	HP	HP Integrity rx5670 Cluster Itanium2/1.5 GHz-64p	1 184 893	Oracle Database 10g Enterprise Edition	Red Hat Enterprise Linux AS 3

Why Data Mining inside DBMS?

Significant OVERHEAD

Why Data Mining inside DBMS?

Let us move algorithms closer to data!

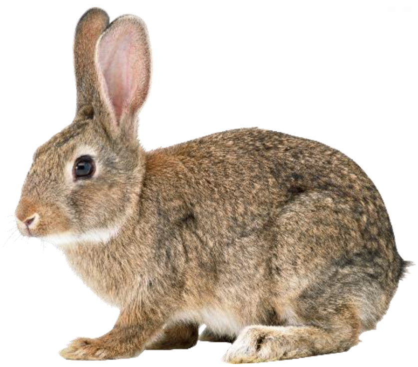


- No export and import data overhead
- All the DBMS services are available for free (query optimization, indexing, data security, etc.)

Data mining inside PDBMS: fuzzy clustering

A

B



C

D

E

Objects	C_1	C_2
A	0.90	0.10
B	0.80	0.20
C	0.15	0.85
D	0.30	0.70
E	0.25	0.75

Fuzzy *c*-Means (FCM) algorithm generalizes *k*-Means and performs clustering where **each object belongs to all clusters** at the same time with different *membership degree*

Fuzzy c-Means clustering

- Input
  - $X = \{x_1, x_2, \ldots, x_n\}$ is a set of objects, $x_i \in \mathbb{R}^d$
  - k is number of clusters
- Output
  - $U \in \mathbb{R}^{n \times k}$ is *membership matrix* where $u_{ij} \in (0,1)$ is a membership degree of object x_i in cluster c_j:

$$\forall i \sum_{j=1}^{k} u_{ij} = 1$$

  - $C \in \mathbb{R}^{k \times d}$ is *matrix of centroids* where row c_j is center of j-th cluster

x	$x_{i,1}$	$\ldots$	$x_{i,d}$
1			
$\ldots$			
n			

u	1	$\ldots$	k
1			
$\ldots$			
n			

c	$c_{j,1}$	$\ldots$	$c_{j,d}$
1			
$\ldots$			
k			

Fuzzy c-Means clustering

- *Objective function* to be minimized

$$J_{FCM}(X,k,m) = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}^2\, ED^2(x_i, c_j)$$

- Computation of centroids

$$c_{j\ell} = \frac{\sum_{i=1}^{n} u_{ij}^2 \cdot x_{i\ell}}{\sum_{i=1}^{n} u_{ij}^2}$$

- Computation of memberships

$$u_{ij} = \sum_{t=1}^{k} \frac{ED(x_i, c_t)}{ED(x_i, c_j)}$$

Fuzzy c-Means clustering

- **Input:** X, k
- **Output:** U, C
- **Method**

$s := 0$

$U^{(0)} := rand(0..1)$

repeat

  Compute $C^{(s)}$ as in **(1)**

  Compute $U^{(s)}$ **and** $U^{(s+1)}$ as in **(2)**

  $s := s + 1$

until $\max_{ij}\left\{\left|u_{ij}^{(s)} - u_{ij}^{(s-1)}\right|\right\} \geq \varepsilon$

(1) $c_{j\ell} = \frac{\sum_{i=1}^{n} u_{ij}^2 \cdot x_{i\ell}}{\sum_{i=1}^{n} u_{ij}^2}$

(2) $u_{ij} = \sum_{t=1}^{k} \frac{ED(x_i, c_t)}{ED(x_i, c_j)}$

pgFCM: relational schema

Object matrix $\boldsymbol{X}$

	x_1	$\cdots$	x_d
1	1.0	$\cdots$	2.1
$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	3.4	$\cdots$	2.9

Vertical object table SV

i	l	val
1	1	1.0
$\vdots$	$\vdots$	$\vdots$
n	d	2.9

Horizontal object table SH

i	x_1	$\cdots$	x_d
1	1.0	$\cdots$	2.1
$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	3.4	$\cdots$	2.9

Centroid matrix $\boldsymbol{C}$

	x_1	$\cdots$	x_d
1	2.2	$\cdots$	8.1
$\vdots$	$\vdots$	$\ddots$	$\vdots$
k	3.4	$\cdots$	6.9

Centroid table C

j	l	val
1	1	2.2
$\vdots$	$\vdots$	$\vdots$
k	d	6.9

Now we can use queries with **aggregation by rows SUM()**

Membership matrix $\boldsymbol{U}$

	1	$\cdots$	k
1	0.2	$\cdots$	0.1
$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	0.8	$\cdots$	0.1

Membership table $U^{(s)}$

i	j	val
1	1	0.2
$\vdots$	$\vdots$	$\vdots$
n	k	0.1

Membership table $UT^{(s+1)}$

i	j	val
1	1	0.2
$\vdots$	$\vdots$	$\vdots$
n	k	0.1

pgFCM: computations

- **Computing centroids**

```
INSERT INTO C
  SELECT R.j, SV.l, sum(R.s * SV.val) / sum(R.s) AS val
  FROM (
    SELECT i, j, U.val^m AS s
    FROM U) AS R, SV
  WHERE R.i = SV.i
  GROUP BY j, l;
```

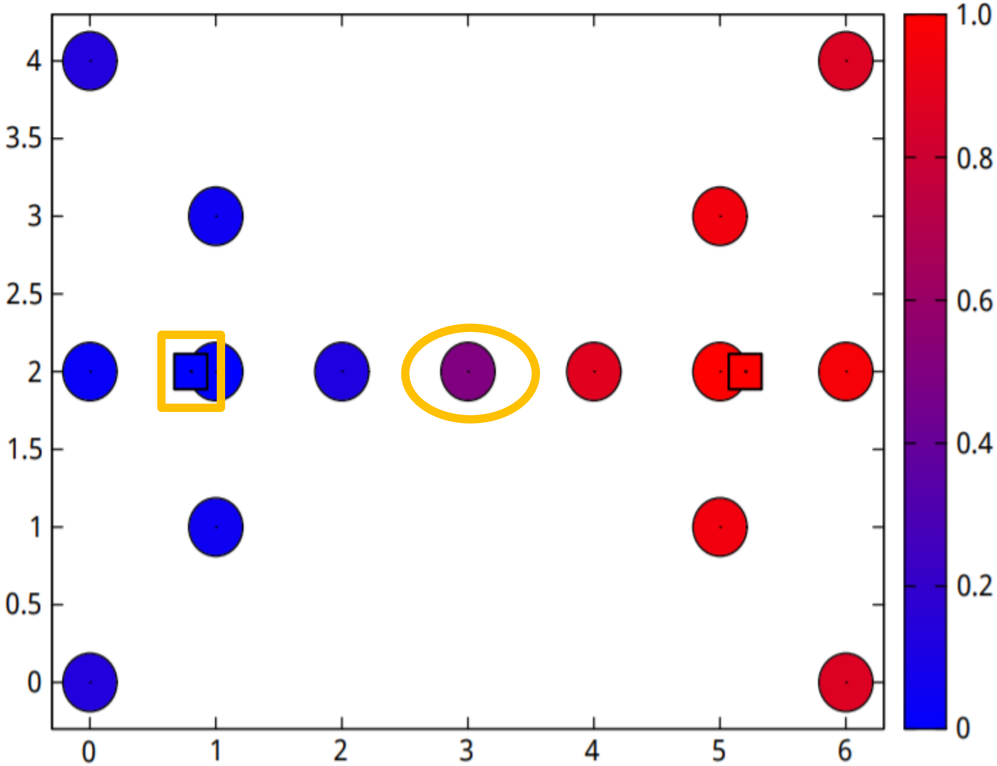
- **Computing Euclidean distances**

```
INSERT INTO SD
  SELECT i, j, sqrt(sum((SV.val - C.val)^2)) as dist
  FROM SV, C
  WHERE SV.l = C.l;
  GROUP BY i, j;
```

- **Computing memberships**

```
INSERT INTO UT
  SELECT i, j, SD.dist^(2.0^(1.0 - m)) * SD1.den AS val
  FROM (
    SELECT i, 1.0 / sum(dist^(2.0^(m - 1.0))) AS den
    FROM SD
    GROUP BY i) AS SD1, SD
  WHERE SD.i = SD1.i;
```

pgFCM: results



$k = 2$

Membership table U

i	j	val
1	1	0.86
1	2	0.13
2	1	0.97
2	2	0.02
…	…	…
8	1	0.49
8	2	0.50
…	…	…
15	1	0.13
15	2	0.86

Horizontal object table SH

i	x_1	x_2
1	0	0
2	0	2
3	0	4
8	3	2
	…	
15	6	0

Centroid table C

j	l	val
1	1	0.79
1	2	2.0
2	1	5.2
2	2	1.99

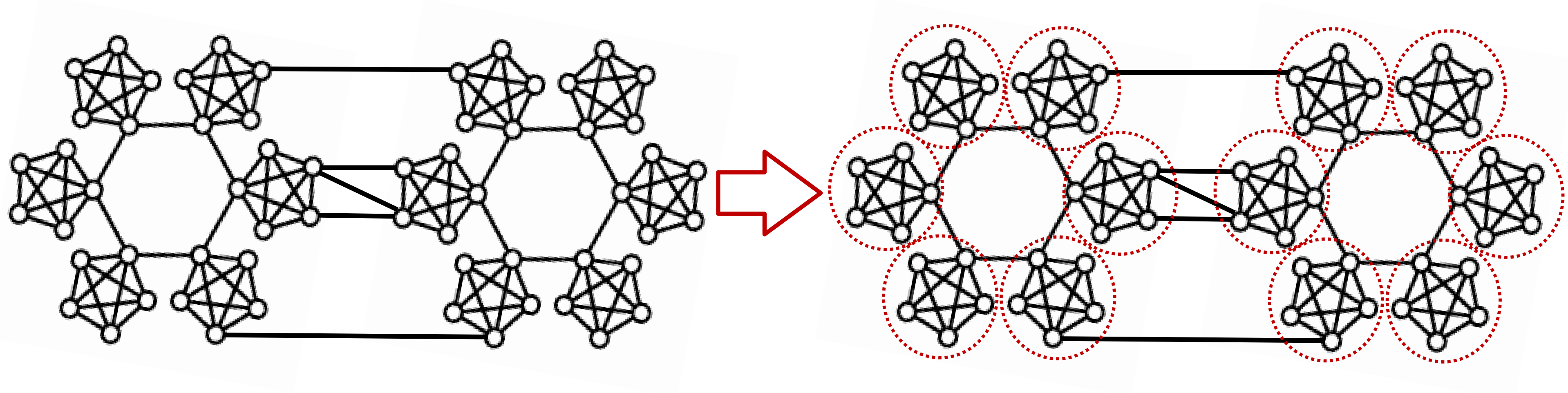
pgFCM vs analogs

Algorithm	Dataset		Time, sec.				*pgFCM*, sec.
	n	d	Clustering	**Export**	**Import**	Total	
WCFC [1)]	$4.9 \cdot 10^6$	41	5100	**203**	**10**	5 313	**5 182**
BigFCM [2)]	$1.1 \cdot 10^7$	28	189	**397**	**25**	611	**531**

[1)] Hidri M.S., *et al.* Speeding up the large-scale consensus fuzzy clustering for handling Big Data. Fuzzy Sets and Systems. 2018. vol. 348. pp. 50–74.

[2)] Ghadiri N., *et al*. BigFCM: Fast, precise and scalable FCM on Hadoop. Future Generation Comp. Syst. 2017. vol. 77. pp. 29–39.

Data mining inside PDBMS: communities

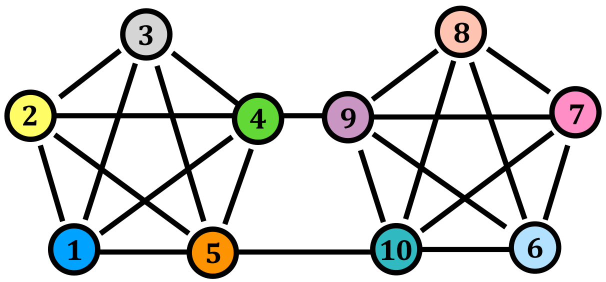


Community detection problem:

Split a big social graph into a set of subgraphs where vertices of each subgraph have dense connections with each other and sparse connections with vertices of other subgraphs

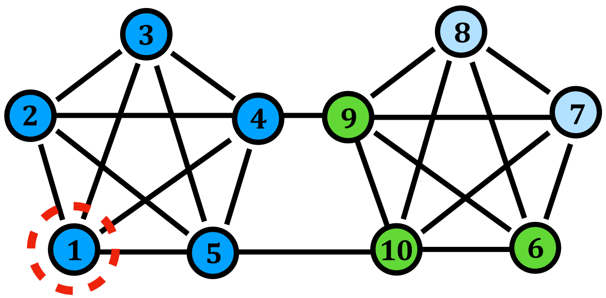
Community detection: ideas

Input graph



for each vertex v
 Assign a community label to v

Label propagation



repeat
 for each vertex v
  for each neighbor vertex u
   Compute their affinity as

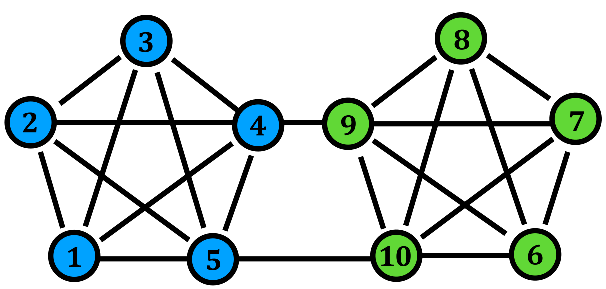
$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

  for each community C
   Compute v's degree of membership as

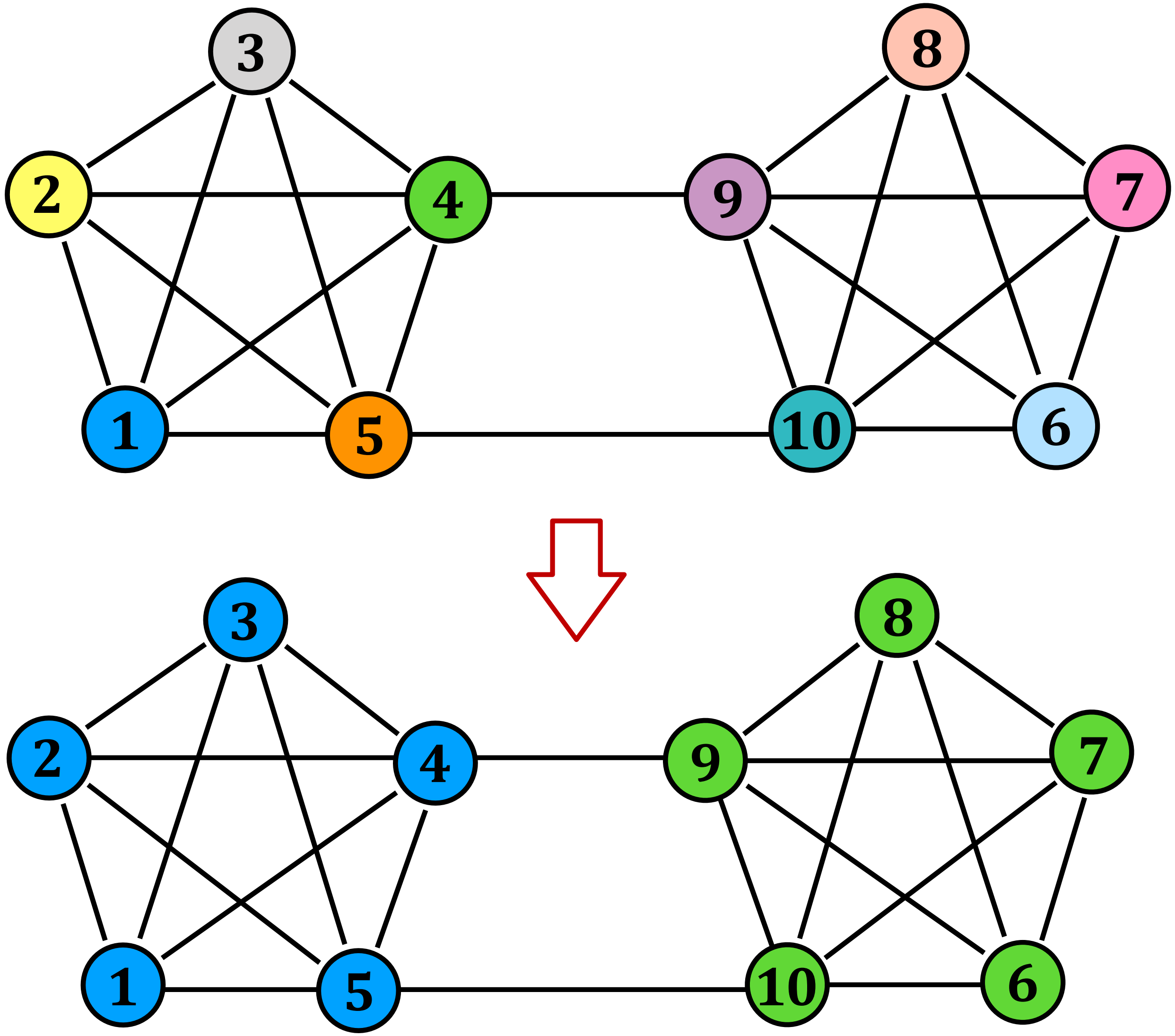
$$d(v,C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v,u)}{\sum_{u \in \mathcal{N}_v} afty(v,u)}$$

  Assign a community label with highest d
until a given part or all vertices keep labels

Final graph



Community detection: relational schema



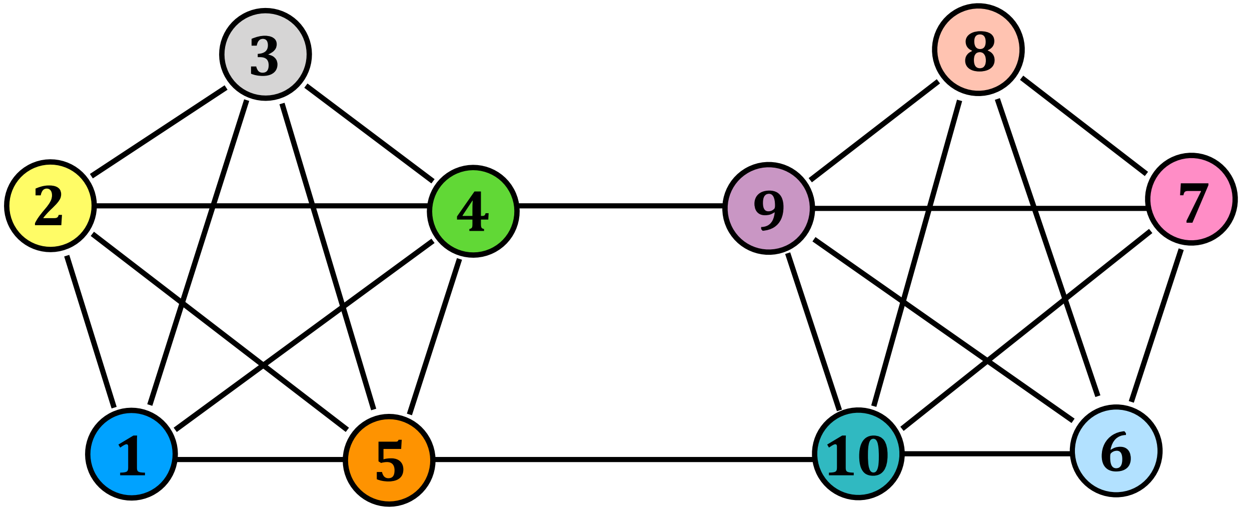
GRAPH table

v	u	w
1	2	1
1	3	1
…		
8	9	1
9	10	1

VERTEX table

v	c
1	1
2	1
…	
9	4
10	4

Community detection: label propagation



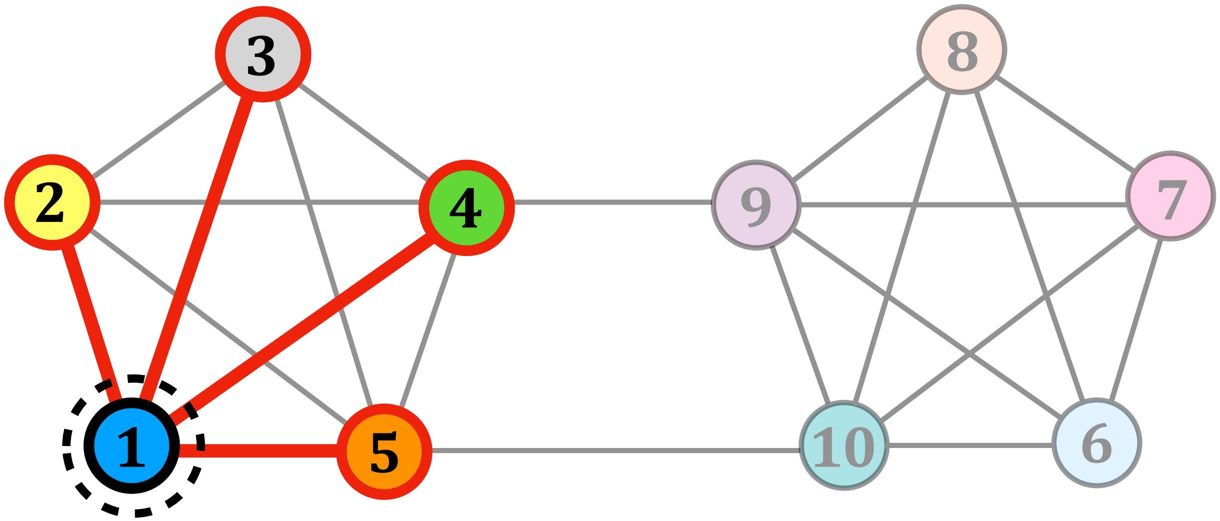
$$afty(v,u) = \frac{w(v,u)}{\sum_{i\in\mathcal{N}_v} w(v,i)}$$

AFF_TMP_WNBR

v	wnbr
…	

```
INSERT INTO AFF_TMP_WNBR
    SELECT v, sum(w) as wnbr
    FROM AFF_TMP_SUBG
    GROUP BY v;
```

Community detection: label propagation



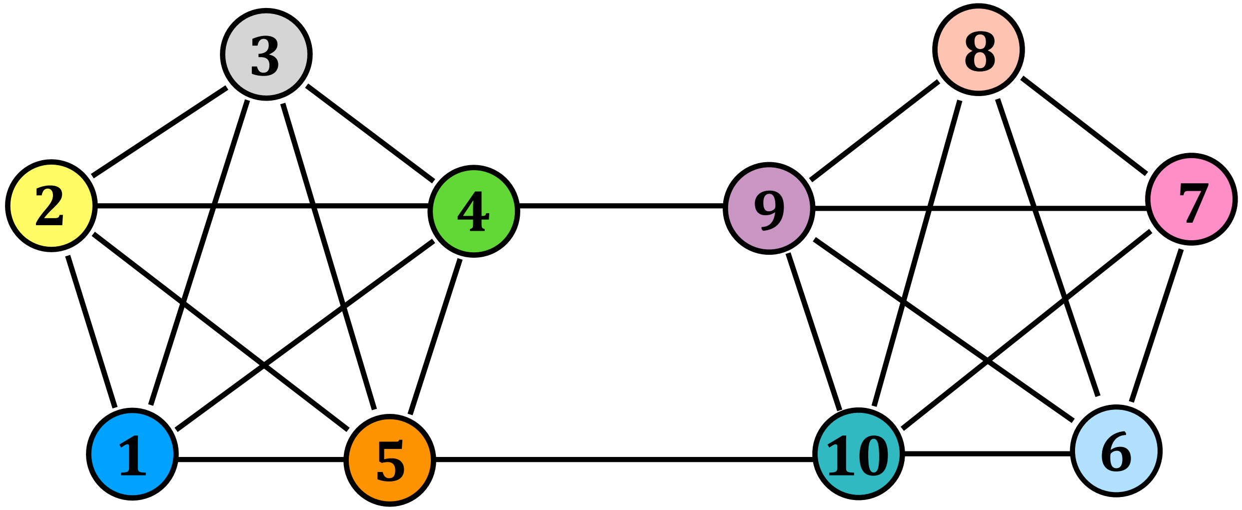
$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

AFF_TMP_WNBR

v	wnbr
1	4
…	

INSERT INTO **AFF_TMP_WNBR**
SELECT v, sum(w) as wnbr
FROM AFF_TMP_SUBG
GROUP BY v;

Community detection: label propagation



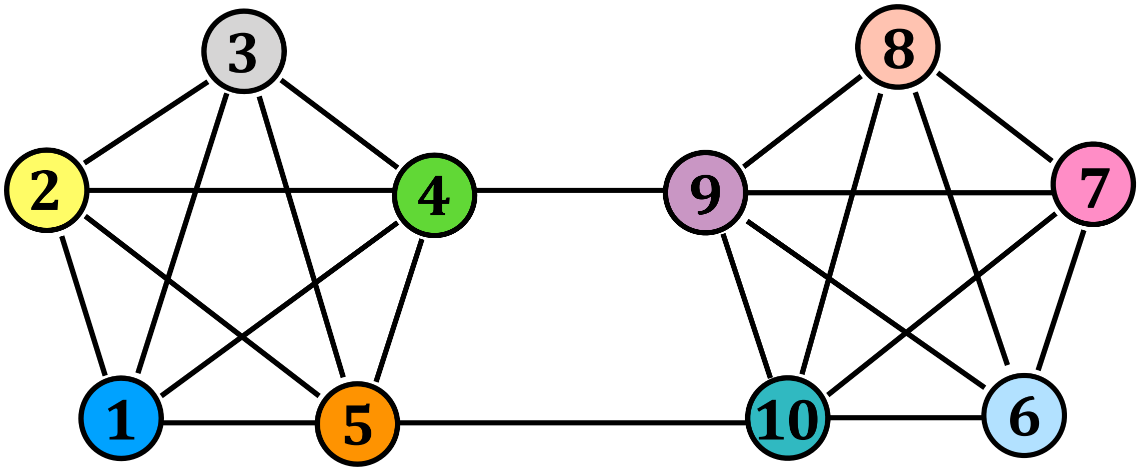
$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

AFF_TMP_WNBR

v	wnbr
1	4
2	4
…	
9	5
10	5

INSERT INTO **AFF_TMP_WNBR**
SELECT v, sum(w) as wnbr
FROM AFF_TMP_SUBG
GROUP BY v;

Community detection: label propagation



$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

- INSERT INTO **AFF_TMP_WNBR**
  SELECT v, sum(w) as wnbr
  FROM AFF_TMP_SUBG
  GROUP BY v;
- INSERT INTO **AFFINITY**
  SELECT X.v, u, w/wnbr as afty
  FROM AFF_TMP_SUBG AS X,
  AFF_TMP_WNBR AS Y
  WHERE X.v = Y.v;

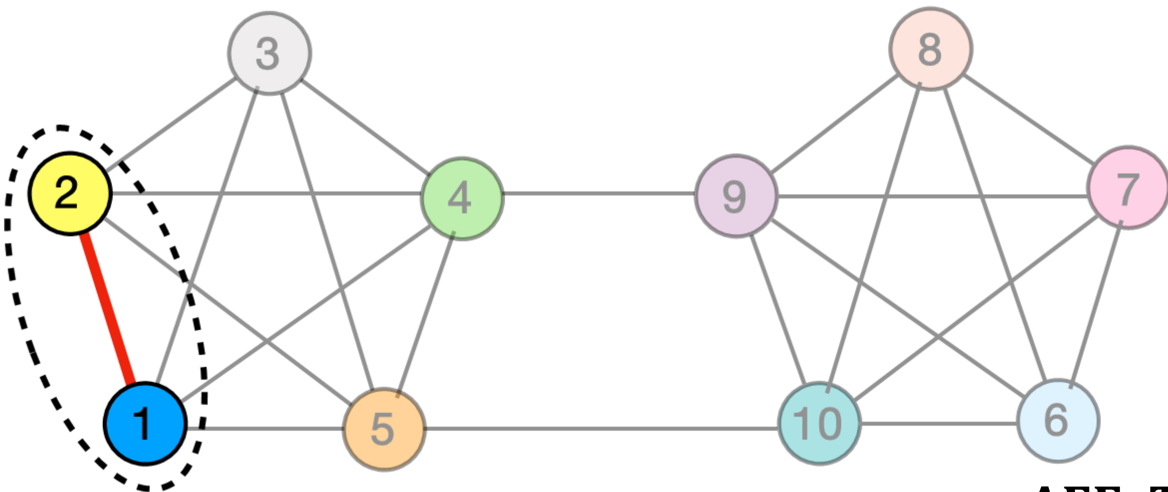
AFF_TMP_WNBR

v	wnbr
1	4
2	4
…	
9	5
10	5

AFFINITY

v	u	afty
…		

Community detection: label propagation



$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

- INSERT INTO **AFF_TMP_WNBR**
  SELECT v, sum(w) as wnbr
  FROM AFF_TMP_SUBG
  GROUP BY v;
- INSERT INTO **AFFINITY**
  SELECT X.v, u, w/wnbr as afty
  FROM AFF_TMP_SUBG AS X,
  AFF_TMP_WNBR AS Y
  WHERE X.v = Y.v;

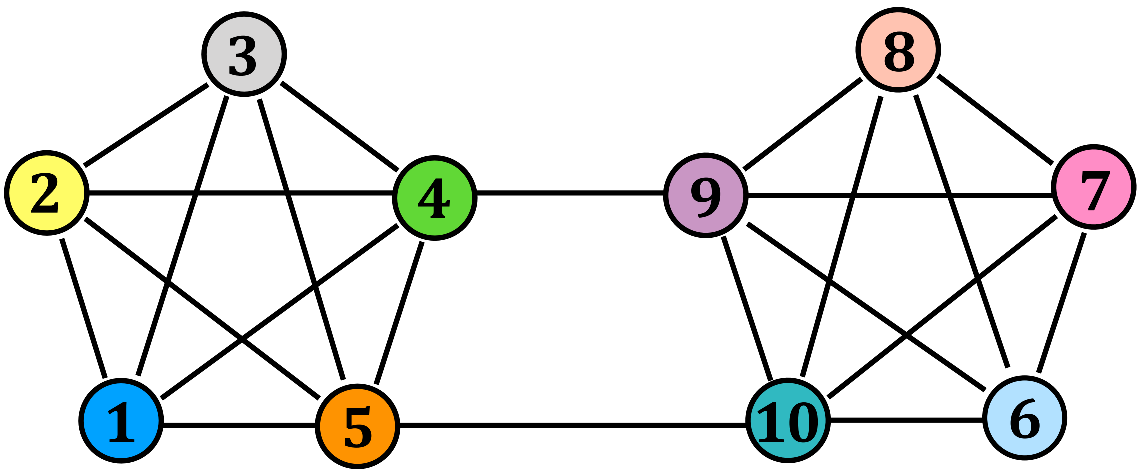
AFF_TMP_WNBR

v	wnbr
1	4
2	4
…	
9	5
10	5

AFFINITY

v	u	afty
1	2	$\frac{1}{4} = 0.25$
…		

Community detection: label propagation



$$afty(v,u) = \frac{w(v,u)}{\sum_{i \in \mathcal{N}_v} w(v,i)}$$

- INSERT INTO **AFF_TMP_WNBR**
  SELECT v, sum(w) as wnbr
  FROM AFF_TMP_SUBG
  GROUP BY v;
- INSERT INTO **AFFINITY**
  SELECT X.v, u, w/wnbr as afty
  FROM AFF_TMP_SUBG AS X,
  AFF_TMP_WNBR AS Y
  WHERE X.v = Y.v;

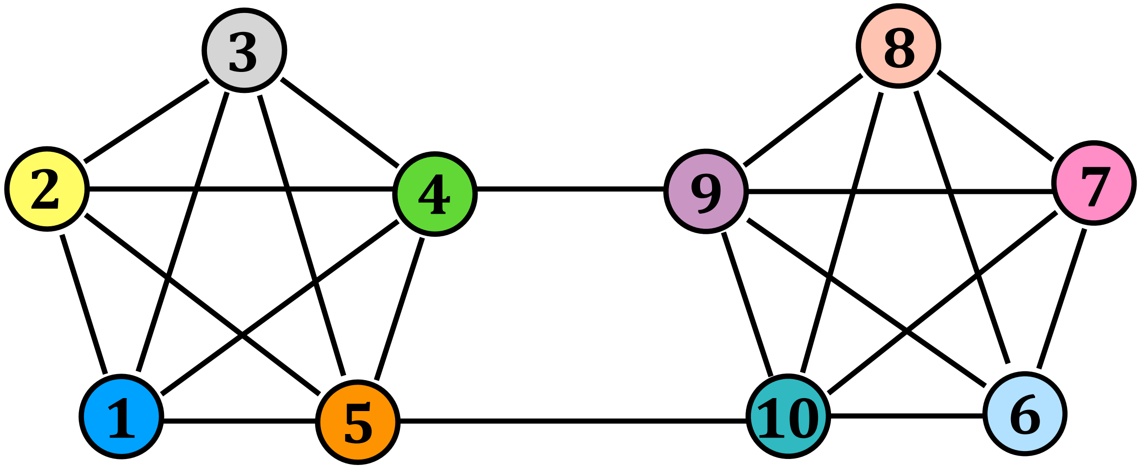
AFF_TMP_WNBR

v	wnbr
1	4
2	4
…	
9	5
10	5

AFFINITY

v	u	afty
1	2	0.25
1	3	0.25
1	4	0.25
…		
10	7	0.2
10	8	0.2
10	9	0.2

Community detection: label propagation



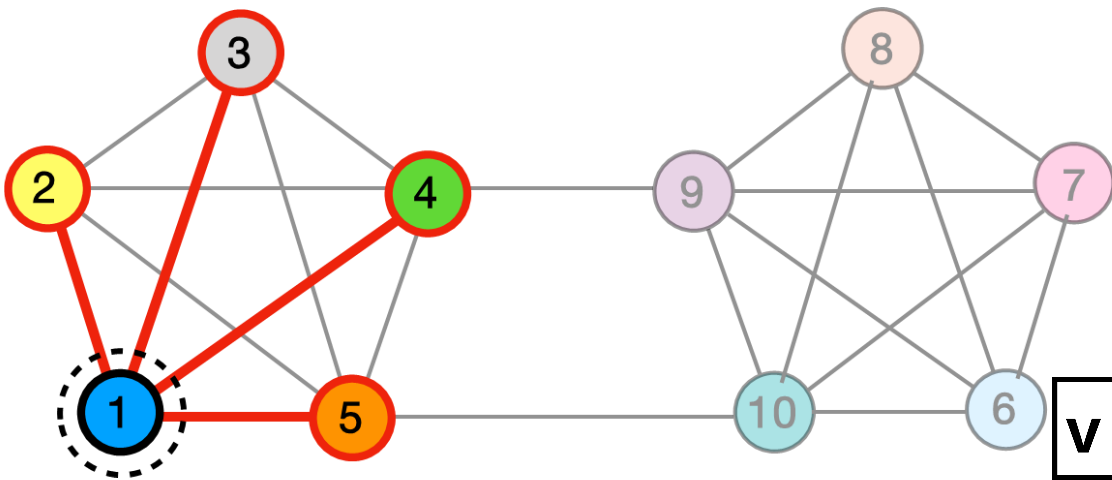
$$d(v,C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v,u)}{\sum_{u \in \mathcal{N}_v} afty(v,u)}$$

```
INSERT INTO COMM_TMP_AFNBRALL
        SELECT v, sum(afty) as afnrall
        FROM AFFINITY
        GROUP BY v;
```

COMM_TMP_AFNRALL

v	afnrall
...	

Community detection: label propagation



$$d(v, C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v, u)}{\sum_{u \in \mathcal{N}_v} afty(v, u)}$$

```
INSERT INTO COMM_TMP_AFNBRALL
    SELECT v, sum(afty) as afnrall
    FROM AFFINITY
    GROUP BY v;
```

AFFINITY

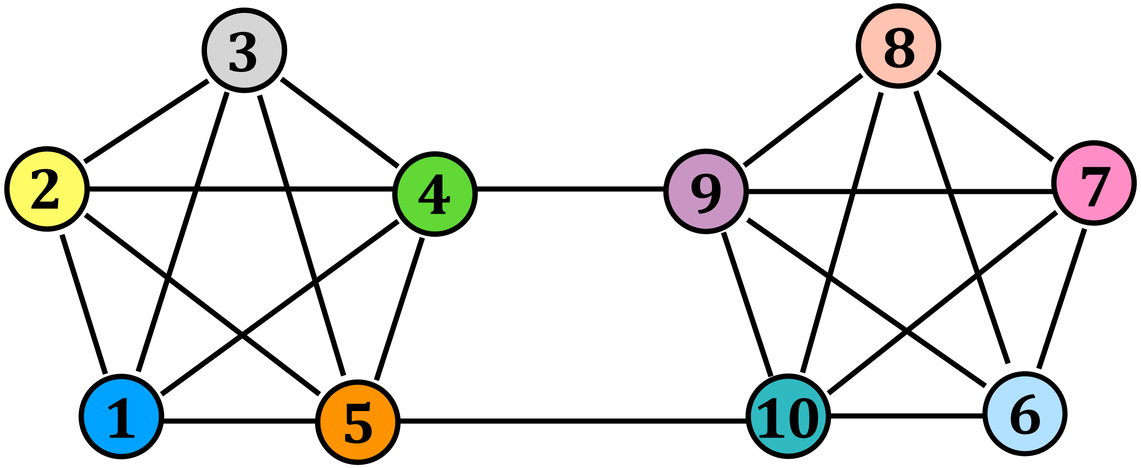
v	u	afty
1	2	0.25
1	3	0.25
1	4	0.25
1	5	0.25
...		
10	8	0.2
10	9	0.2

$\sum$

COMM_TMP_AFNBRALL

v	afnrall
1	1
...	

Community detection: label propagation



$$d(v, C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v,u)}{\sum_{u \in \mathcal{N}_v} afty(v,u)}$$

- INSERT INTO **COMM_TMP_AFNBRALL**
  SELECT v, sum(afty) as afnrall
  FROM AFFINITY
  GROUP BY v;
- INSERT INTO **COMM_TMP_AFNBRCOM**
  SELECT AFFINITY.v, VERTEX.c, sum(afty)
  FROM AFFINITY, VERTEX
  WHERE AFFINITY.u = VERTEX.v
  GROUP BY AFFINITY.v, VERTEX.c;

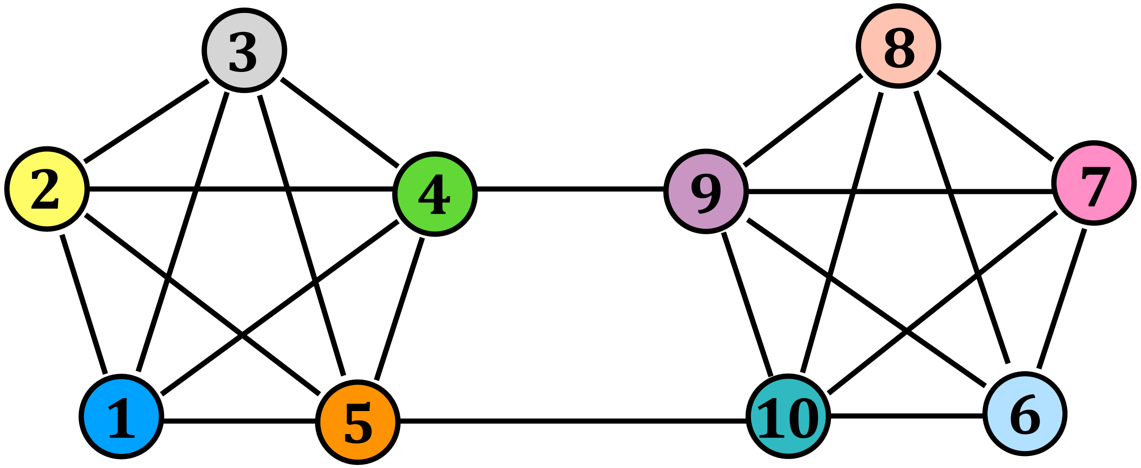
COMM_TMP_AFNBRALL

v	afnrall
1	1
2	1
…	
9	1
10	1

COMM_TMP_AFNBRCOM

v	u	afty
1	2	0.25
1	3	0.25
1	4	0.25
…		
10	7	0.2
10	8	0.2
10	9	0.2

Community detection: label propagation



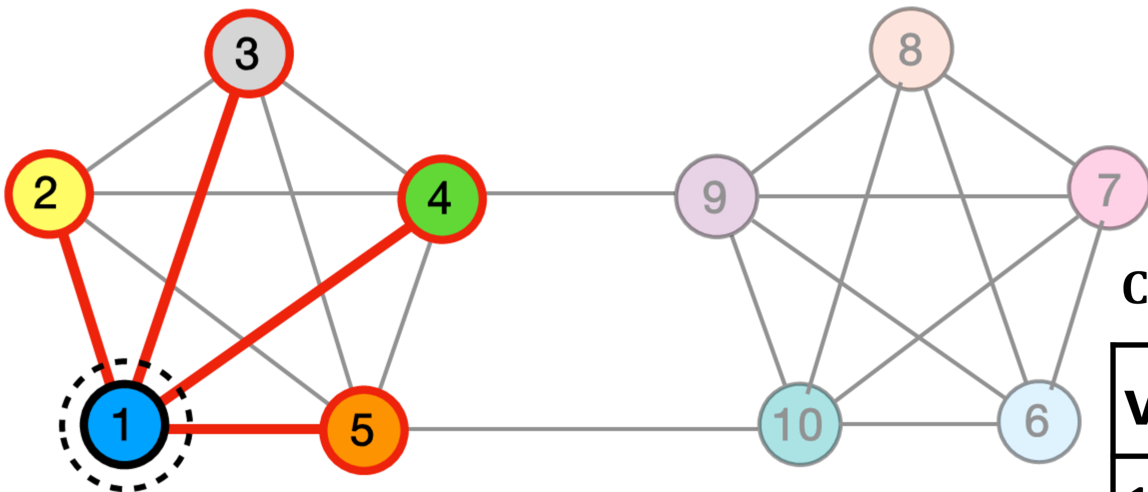
$$d(v,C)=\frac{\sum_{u\in\mathcal{N}_v \wedge \mathcal{L}_u=C} afty(v,u)}{\sum_{u\in\mathcal{N}_v} afty(v,u)}$$

COMMUNITY

v	c	d
1	2	0.25
1	3	0.25
1	4	0.25
…		
10	7	0.2
10	8	0.2
10	9	0.2

```
INSERT INTO COMMUNITY
        SELECT X.v, c, afnbrcom/afnbrall as d
        FROM COMM_TMP_AFNBRALL as X,
            COMM_TMP_AFNBRCOM as Y
        WHERE X.v = Y.v;
```

Community detection: label propagation



$$d(v, C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v, u)}{\sum_{u \in \mathcal{N}_v} afty(v, u)}$$

- INSERT INTO **COMMUNITY**
  SELECT X.v, c, afnbrcom/afnbrall as d
  FROM COMM_TMP_AFNBRALL as X,
  COMM_TMP_AFNBRCOM as Y
  WHERE X.v = Y.v;

- INSERT INTO **COMM_TMP_DMAX**
  SELECT v, max(d) as dmax
  FROM COMMUNITY
  GROUP BY v;

COMMUNITY

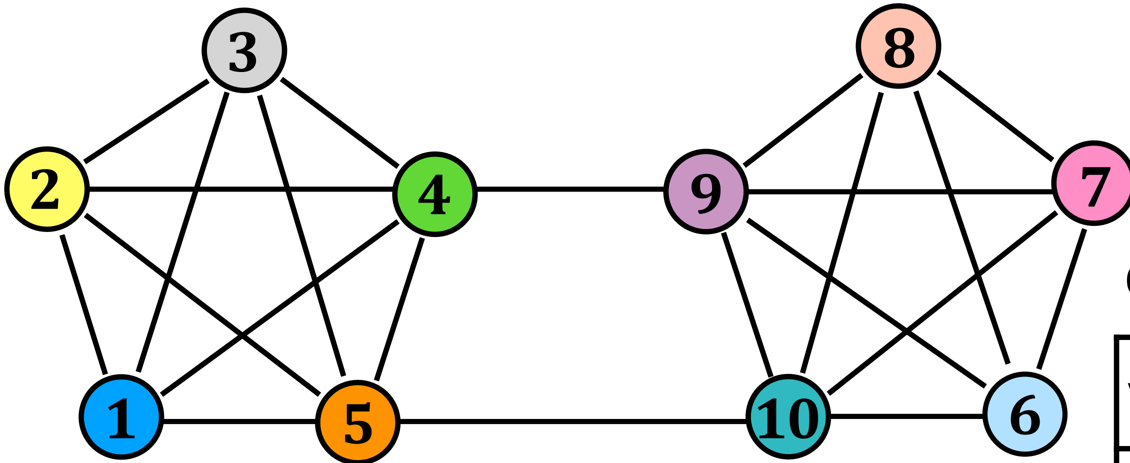
v	c	d
1	2	0.25
1	3	0.25
1	4	0.25
1	5	0.25
…		
10	8	0.2
10	9	0.2

max

COMM_TMP_DMAX

v	dmax
1	0.25
…	

Community detection: label propagation



$$d(v, C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v, u)}{\sum_{u \in \mathcal{N}_v} afty(v, u)}$$

- INSERT INTO **COMMUNITY**
  SELECT X.v, c, afnbrcom/afnbrall as d
  FROM COMM_TMP_AFNBRALL as X,
  COMM_TMP_AFNBRCOM as Y
  WHERE X.v = Y.v;
- INSERT INTO **COMM_TMP_DMAX**
  SELECT v, max(d) as dmax
  FROM COMMUNITY
  GROUP BY v;

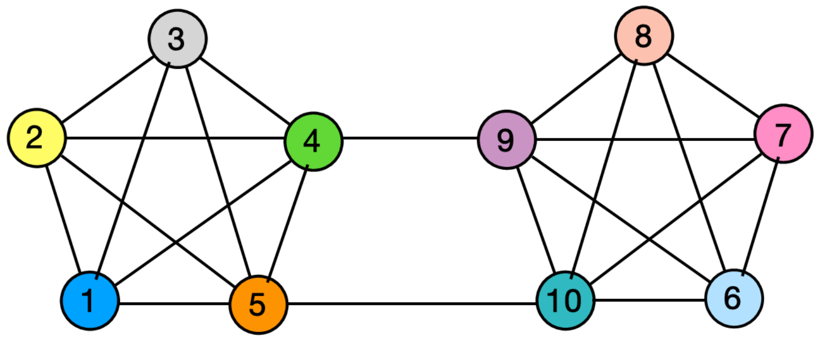
COMMUNITY

v	c	d
1	2	0.25
1	3	0.25
1	4	0.25
1	5	0.25
…		
10	8	0.2
10	9	0.2

COMM_TMP_DMAX

v	dmax
1	0.25
2	0.25
…	
9	0.2
10	0.2

Community detection: label propagation



$$d(v,C) = \frac{\sum_{u \in \mathcal{N}_v \wedge \mathcal{L}_u = C} afty(v,u)}{\sum_{u \in \mathcal{N}_v} afty(v,u)}$$

COMMUNITY as X

v	c	d
1	2	0.25
1	3	0.25
1	4	0.25
1	5	0.25
…		
10	8	0.2
10	9	0.2

COMM_TMP_ DMAX as Y

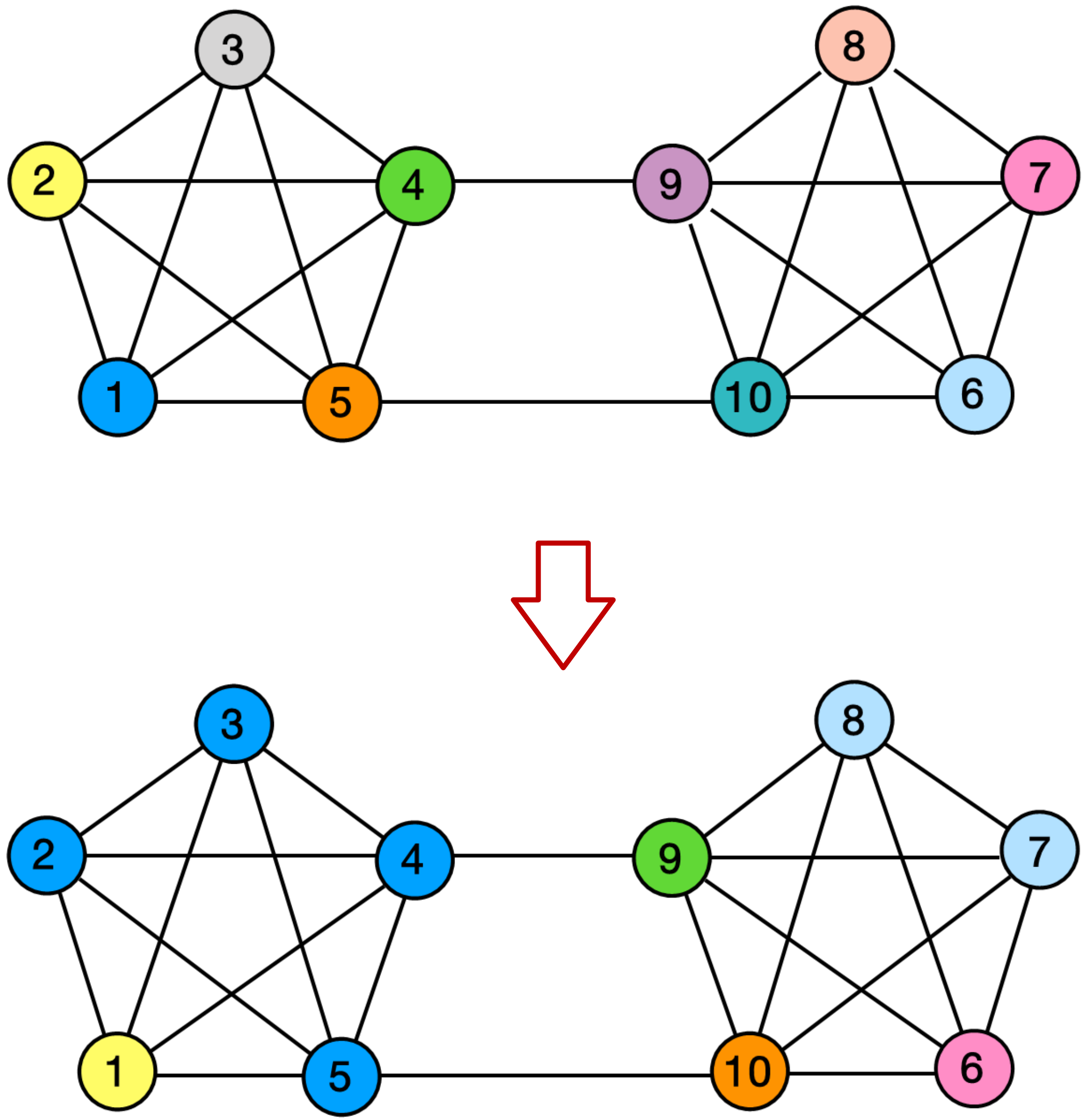
v	dmax
1	0.25
2	0.25
…	
9	0.2
10	0.2

$$\gamma_{a,min(c)}(\pi_{v,c}(X \bowtie Y))$$

VERTEX

v	c
1	2
2	1
…	
9	4
10	5

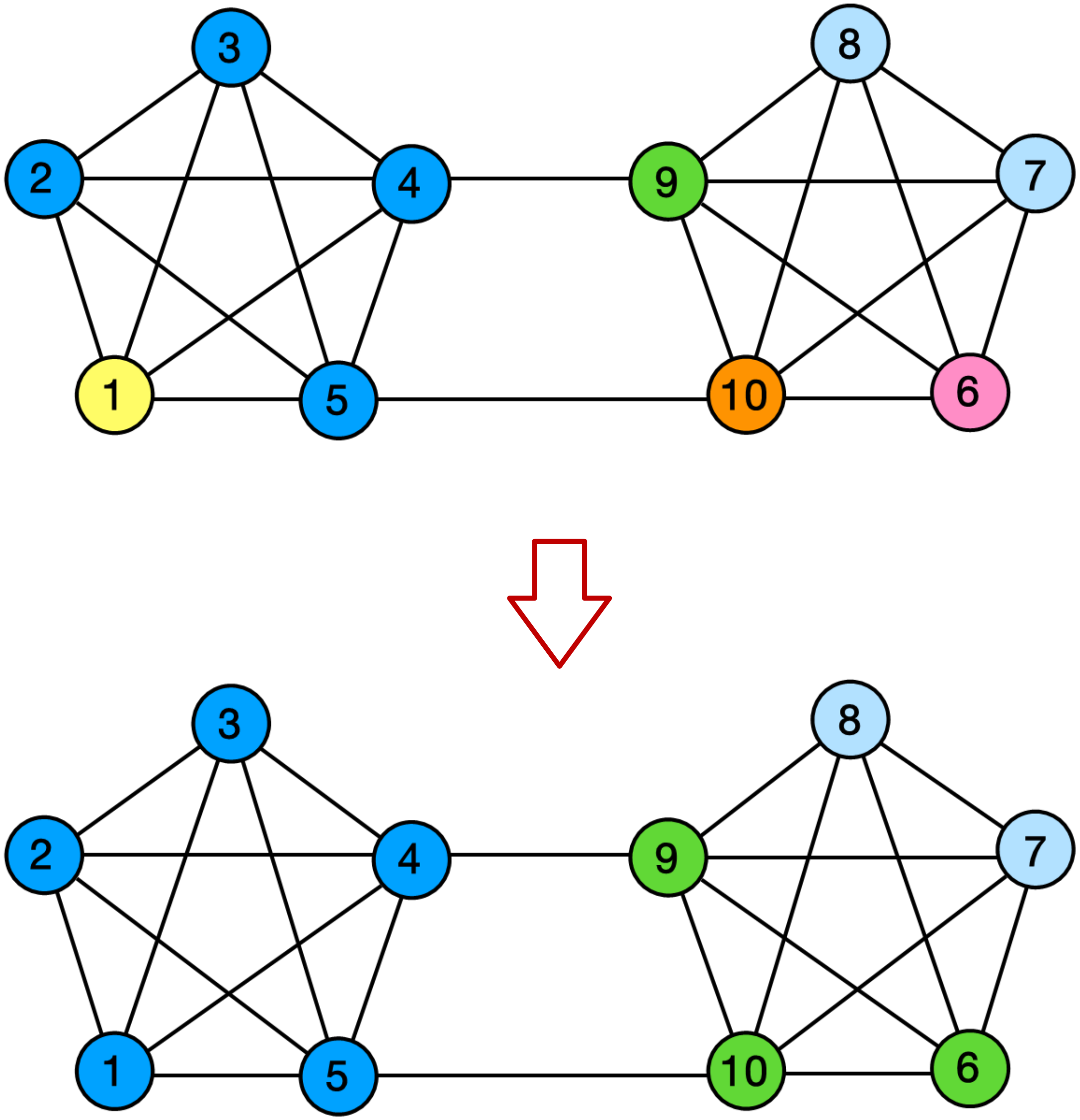
Community detection



VERTEX

v	c
1	2
2	1
3	1
4	1
5	1
6	7
7	6
8	6
9	4
10	5

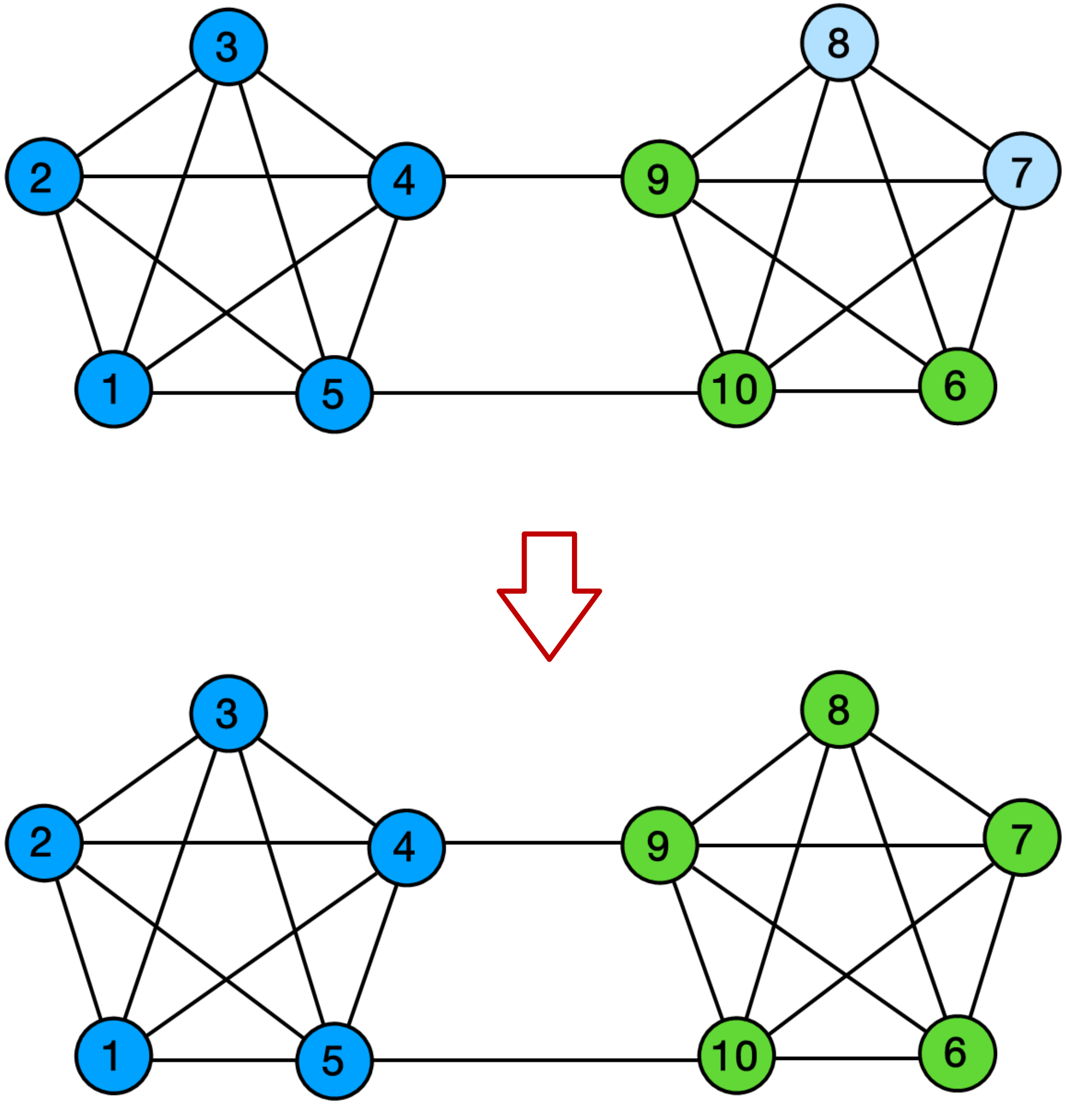
Community detection



VERTEX

v	c
1	1
2	1
3	1
4	1
5	1
6	4
7	6
8	6
9	4
10	4

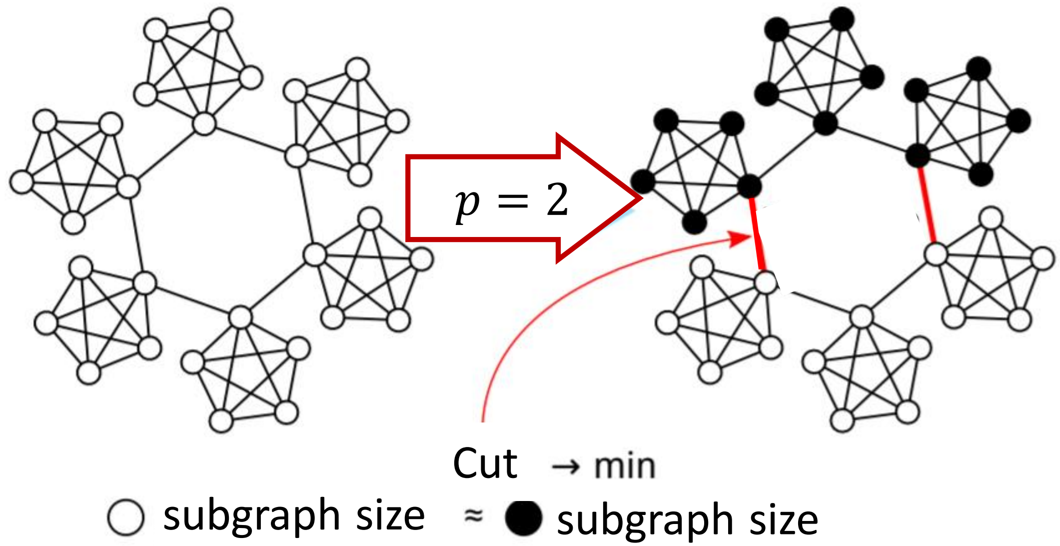
Community detection



VERTEX

v	c
1	1
2	1
3	1
4	1
5	1
6	4
7	4
8	4
9	4
10	4

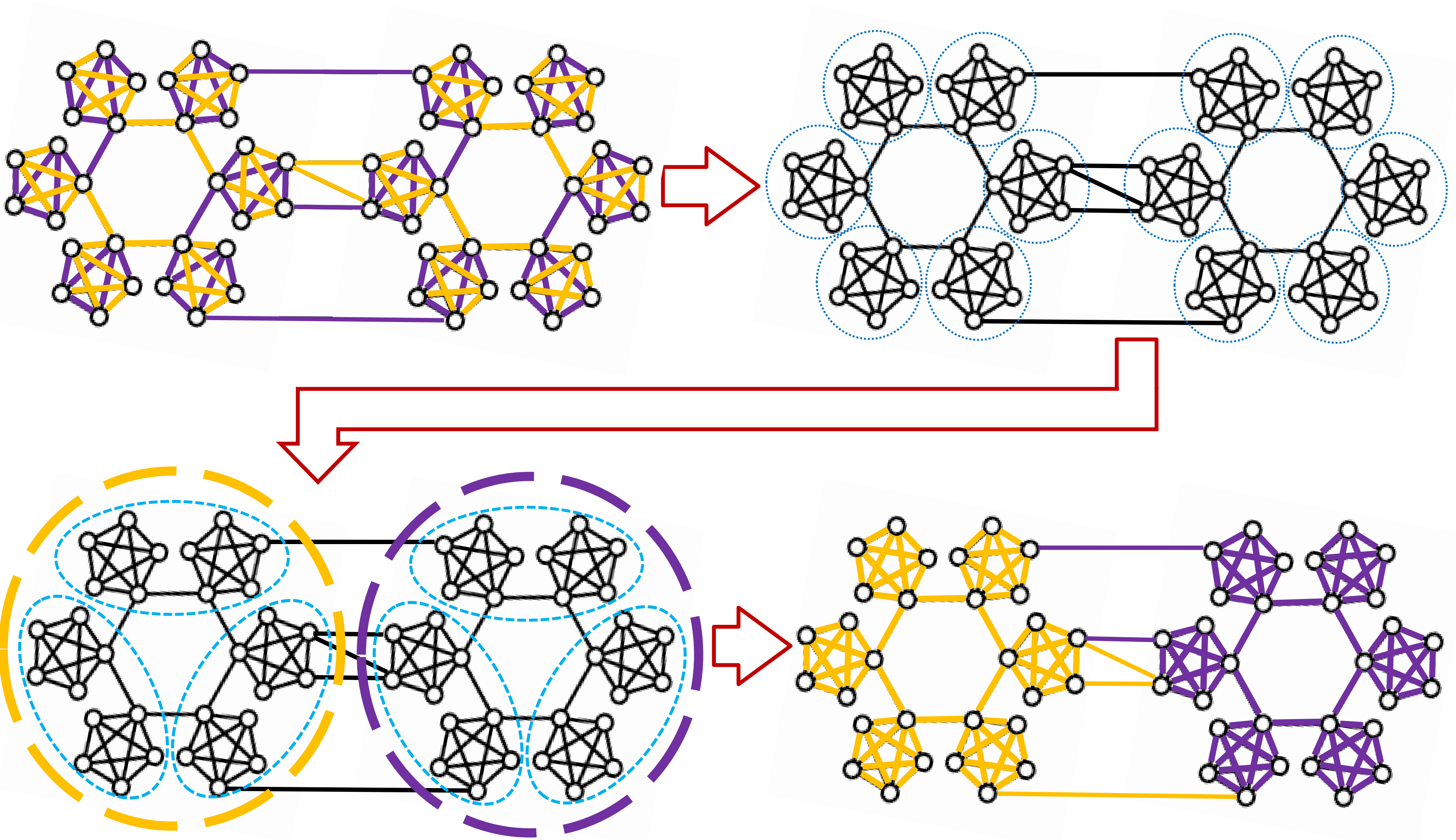
Even more! Graph partitioning in PDBMS



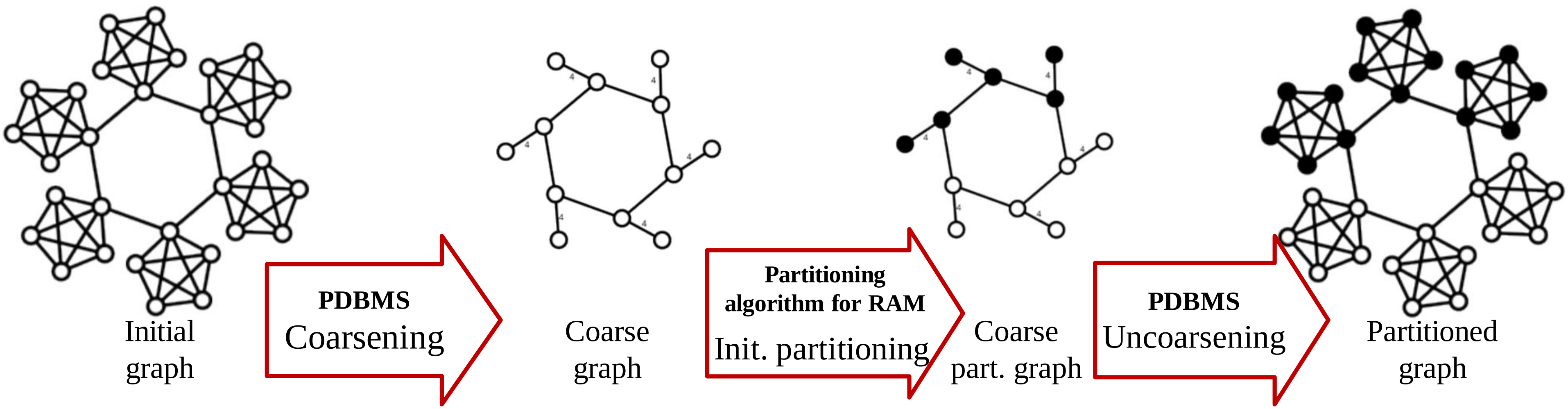
$$G(N, E, w)$$

1. $N = \cup_{i=1}^{p} N_i,\ \forall i \neq j\ N_i \cap N_j = \emptyset,\ p > 1$
2. $w(N_i) \approx \frac{w(N)}{p}\ \ \forall i \in \{1, \dots, p\}$
3. $Cut\ W_{cut} \rightarrow \min,\ W_{cut} \coloneqq \sum_{e \in E_{cut}} w(e),$
   $E_{cut} \coloneqq \{(u, v) \in E \mid u \in N_i, v \in N_j, 1 \leq i, j \leq p, i \neq j\}$

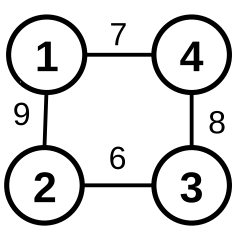
Graph partitioning in PDBMS



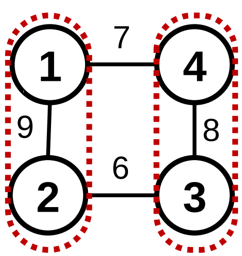
Multilevel graph partitioning in PDBMS



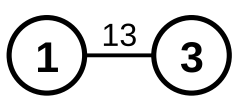
U	*V*	*W*
1	2	9
2	3	6
3	4	8
4	1	7



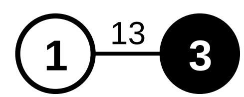
U	*V*
1	2
3	4



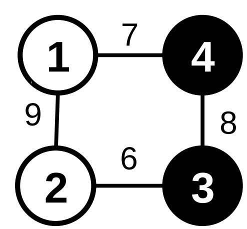
U	*V*	*W*
1	3	13



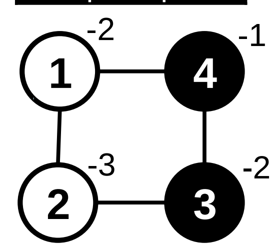
U	*P*
1	**1**
3	**0**



U	*P*
1	**1**
2	**1**
3	**0**
4	**0**



A	*P*	*G*
1	**1**	-2
2	**1**	-3
3	**0**	-2
4	**0**	-1

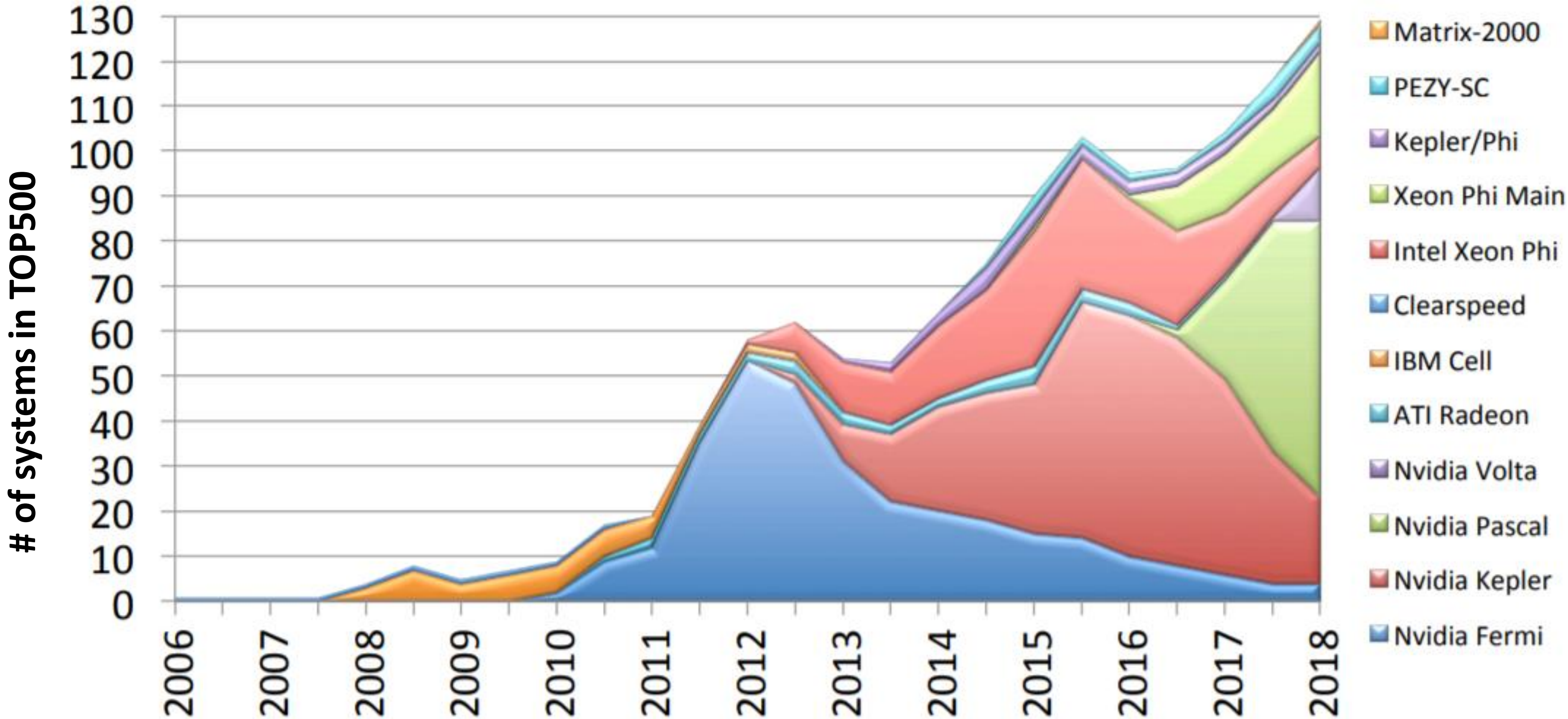


Comparison with analogs

Algorithm	Graph		Time, sec.				***dbPar graph,*** sec.
	$\|N\|$	$\|E\|$	Detection	**Export**	**Import**	Total	
MSP [1)]	10^7	$5 \cdot 10^7$	962	307	36	1 305	**1 189**
ParMETIS [2)]	$7.7 \cdot 10^6$	$1.33 \cdot 10^8$	500	474	30	1 004	**886**
PT-Scotch [3)]	$2.3 \cdot 10^7$	$1.75 \cdot 10^8$	417	652	79	1 148	**897**

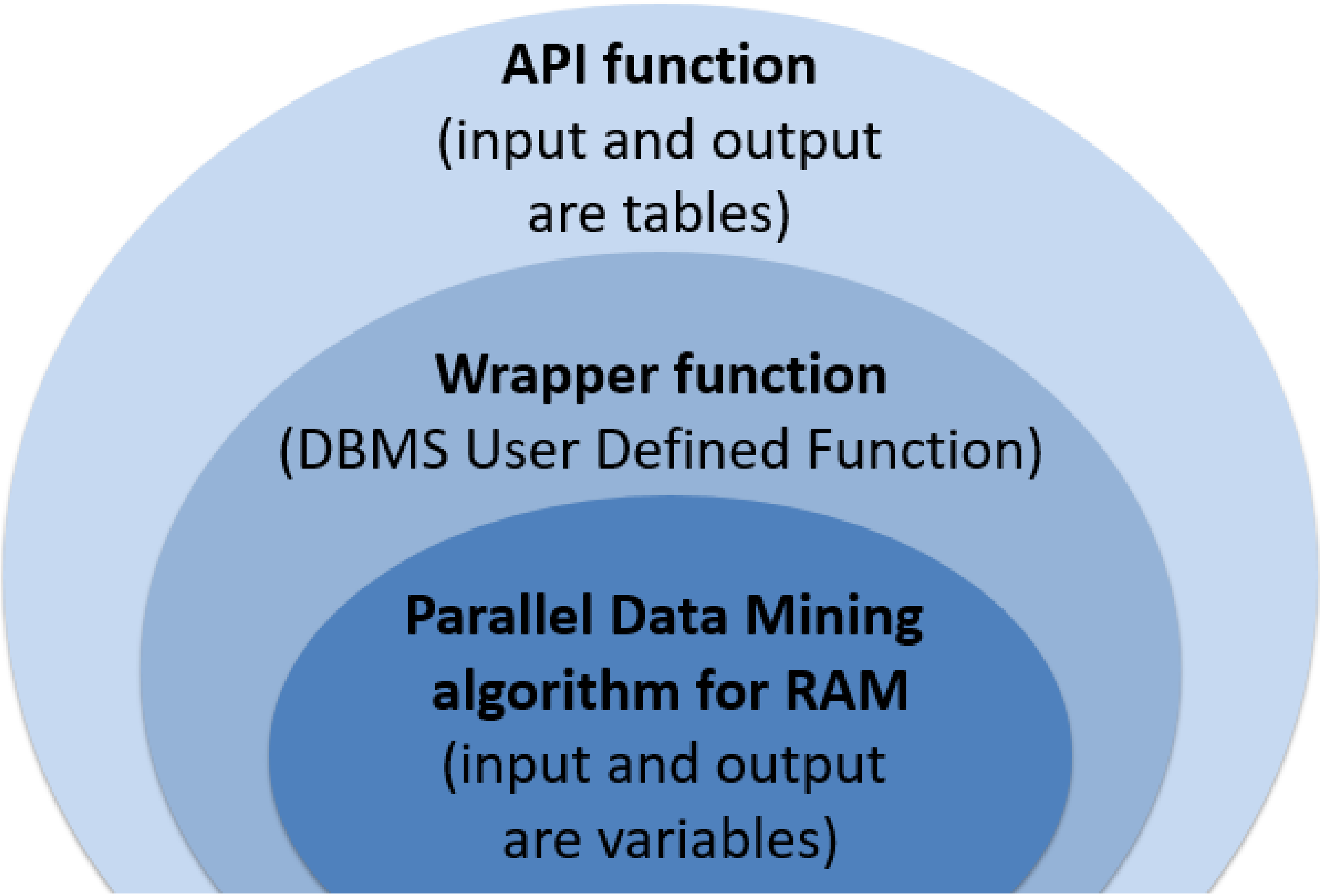
[1)] Zeng Z., *et al.* A parallel graph partitioning algorithm to speed up the large-scale distributed graph mining. BigMine 2012. pp. 61–68.
[2)] Karypis G. METIS and ParMETIS. Enc. of Parallel Computing (Ed. by D.A. Padua). Springer, 2011. pp. 1117–1124.
[3)] Chevalier C., Pellegrini F. PT-Scotch: A tool for efficient parallel graph ordering. Parallel Computing. 2008. vol. 34, no. 6-8. pp. 318–331.

TOP 500 The List.: Many-core accelerators are coming



… and we can use it for in-DBMS Data Mining!

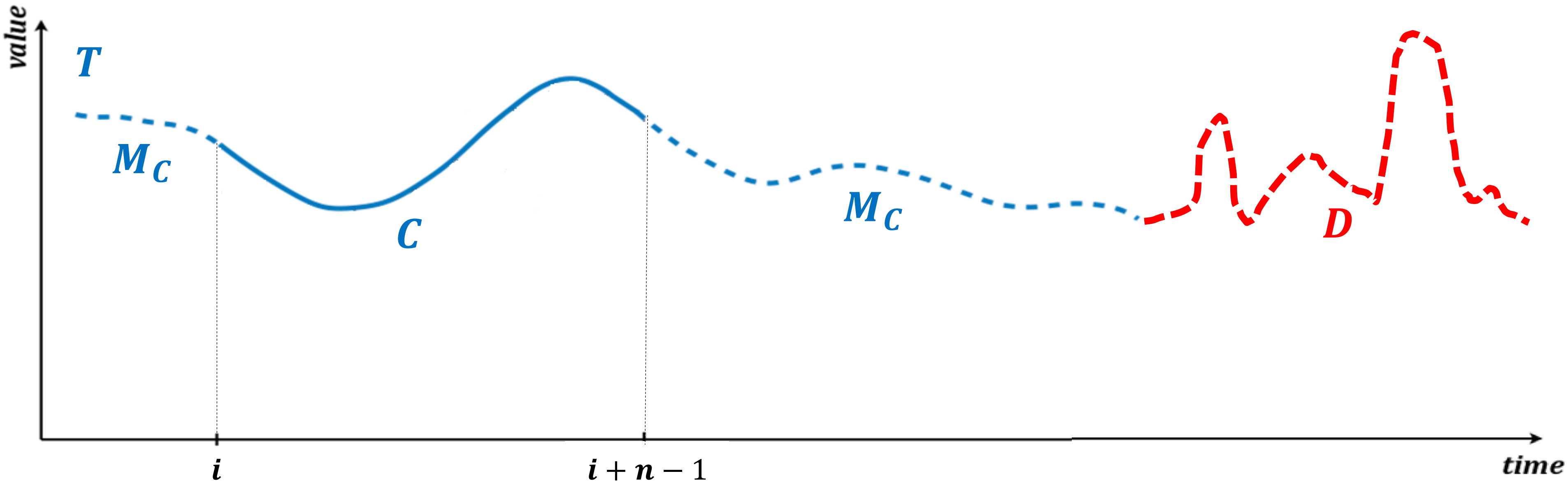
Embedding parallel DM algorithms into DBMS



How should we parallelize algorithms for accelerators?

- **SIMD processing and auto-vectorization**
  - Single Instruction Multiple Data
  - Computations should be organized as **for**-loops without data dependencies, so the compiler will be able to change a set of scalar statements in loop body to one vector operation
- **Data alignment**
  - Data element size should be multiple of vector register size to avoid loop peeling

Accelerating anomaly detection in time series



- Time series: $T := (t_1, \ldots, t_m), t_i \in \mathbb{R}$
- Subsequence: $C := T_{i,n}$ where $n \ll m$
- Non-self match subsequence for C : $M_C := T_{j,n}, |i-j| \geq n$
- **Anomalous subsequence** D:

$$\forall C, M_C \in T \min(ED(D, M_D)) > \min(ED(C, M_C))$$

Anomaly detection: ideas

- We need **index structures** to effectively iterate subsequences of the given time series
- We differ the following types of subsequences
  - **NearAnomalies** are the rarest, and **Others**
  - **Neighbors** are close to the given subsequence, and **Strangers**
- We can avoid brute force and **prune** clearly unpromising subsequences

$$
\begin{array}{l}
d_{anomaly} \leftarrow 0;\ d_{NN} \leftarrow \infty \\
\textbf{for } C_i \in \boldsymbol{NearAnomalies}, \boldsymbol{Others} \\
\quad \textbf{for } C_j \in \boldsymbol{Neighbors}, \boldsymbol{Strangers} \\
\quad\quad d \leftarrow ED^2(C_i, C_j) \\
\quad\quad \textbf{if } \ d < d_{anomaly} \\
\quad\quad\quad \textbf{break} \\
\quad\quad d_{NN} \leftarrow \min(d, d_{NN}) \\
\quad d_{anomaly} \leftarrow \max\left(d_{NN}, d_{anomaly}\right) \\
\quad C_{anomaly} \leftarrow C_i \\
\textbf{return } \{C_{anomaly}, d_{anomaly}\}
\end{array}
$$

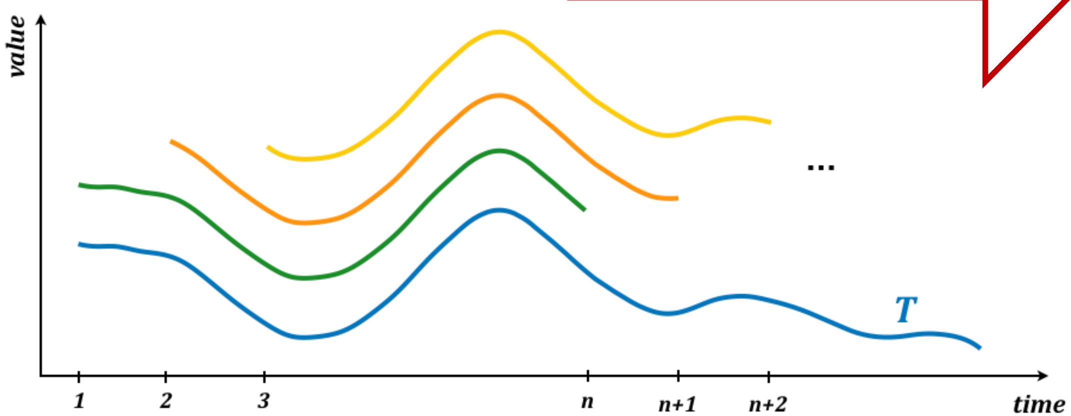
Index structures: Subsequence matrix

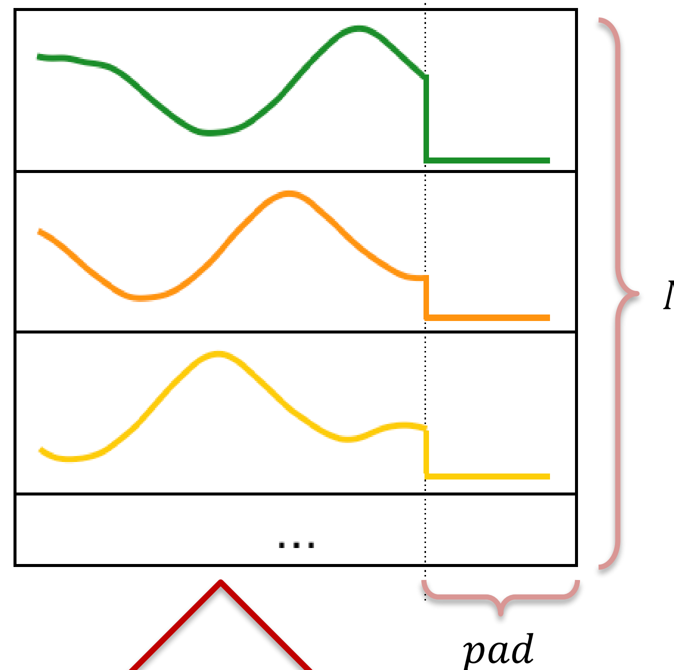
$T \in \mathbb{R}^m$

Alignment

$(n + pad) \vdots width_{VPU}$

$\boldsymbol{S} \in \mathbb{R}^{N \times (n+pad)}$





$N := |T| - n + 1$

$width_{VPU}$

z-normalization

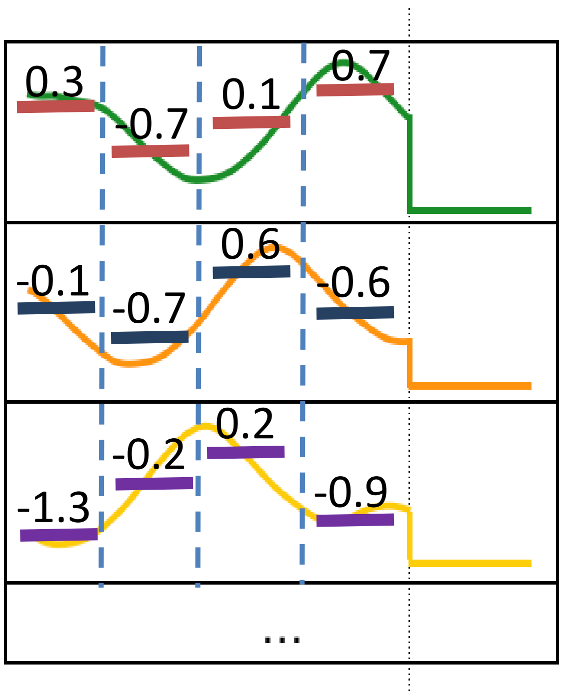
$$\hat{S} = (\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n)$$

$$\hat{s}_i = \frac{s_i - \mu}{\sigma},\ \mu = \frac{1}{n}\sum_{i=1}^{n} s_i\,,\ \sigma^2 = \frac{1}{n}\sum_{i=1}^{n} s_i^2 - \mu^2$$

PAA transformation

Subsequence matrix

$$\boldsymbol{S} \in \mathbb{R}^{N\times(n+pad)}$$



Piecewise Aggregate Approximation

$$PAA(i,k) = \frac{w}{n} \sum_{j=\left(\frac{n}{w}\right)(i-1)+1}^{\left(\frac{n}{w}\right)i} S(k,j)$$

Matrix of PAA codes

$$\boldsymbol{PAA} \in \mathbb{R}^{N\times w}$$

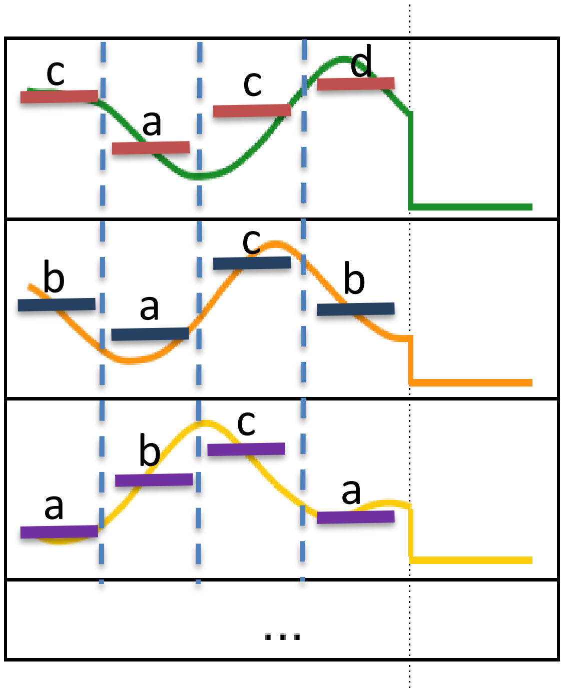
0.3	-0.7	0.1	0.7
-0.1	-0.7	0.6	-0.6
-1.3	-0.2	0.2	-0.9
	...		

w – compression degree (typically w=4)

SAX transformation

Subsequence matrix

$$\boldsymbol{S} \in \mathbb{R}^{N\times(n+pad)}$$



Symbolic Aggregate ApproXimation

$(-\infty; -0.67)$	$[-0.67; 0)$	$[0; 0.67)$	$[0.67; \infty)$
a	b	c	d

Coding table in alphabet A
(typically $|A|$=4)

Matrix of SAX codes

$$\boldsymbol{SAX} \in \mathbb{N}^{N\times w}$$

c	a	c	d
b	a	c	b
a	b	c	a
...			

Indices of SAX codes and near anomalies

Matrix of SAX codes

$$\boldsymbol{SAX} \in \mathbb{N}^{N \times w}$$

1	a	b	d	d
2	c	d	a	b
3	a	b	d	c
	…			
j	a	b	d	c
ℓ	c	d	a	b
u	c	d	a	b
k	a	b	d	d

Count word frequencies

Frequency index of SAX codes

$$\boldsymbol{F_{SAX}} \in \mathbb{N}^{N}$$

2
3
2
…
2
3
3
2
…

Building an index

Index of near anomalies

$$\boldsymbol{Cand} \in \mathbb{N}^{N}$$

1
3
j
k
-
-
-
-

Dictionary and its indices

Dictionary

$\boldsymbol{W_A} \in \mathbb{N}^{|A|^w \times w}$

a	a	a	a
a	a	a	b
…			
a	b	d	c
a	b	d	d
….			
c	d	a	b
…			
d	d	d	d

Matrix of SAX codes

$\boldsymbol{SAX} \in \mathbb{N}^{N \times w}$

1	a	b	d	d
2	c	d	a	b
3	a	b	d	c
	…			
j	a	b	d	c
ℓ	c	d	a	b
u	c	d	a	b
k	a	b	d	d
	…			

Frequency index

$\boldsymbol{F_W} \in \mathbb{N}^{|A|^w}$

0
0
…
2
2
…
3
…
0

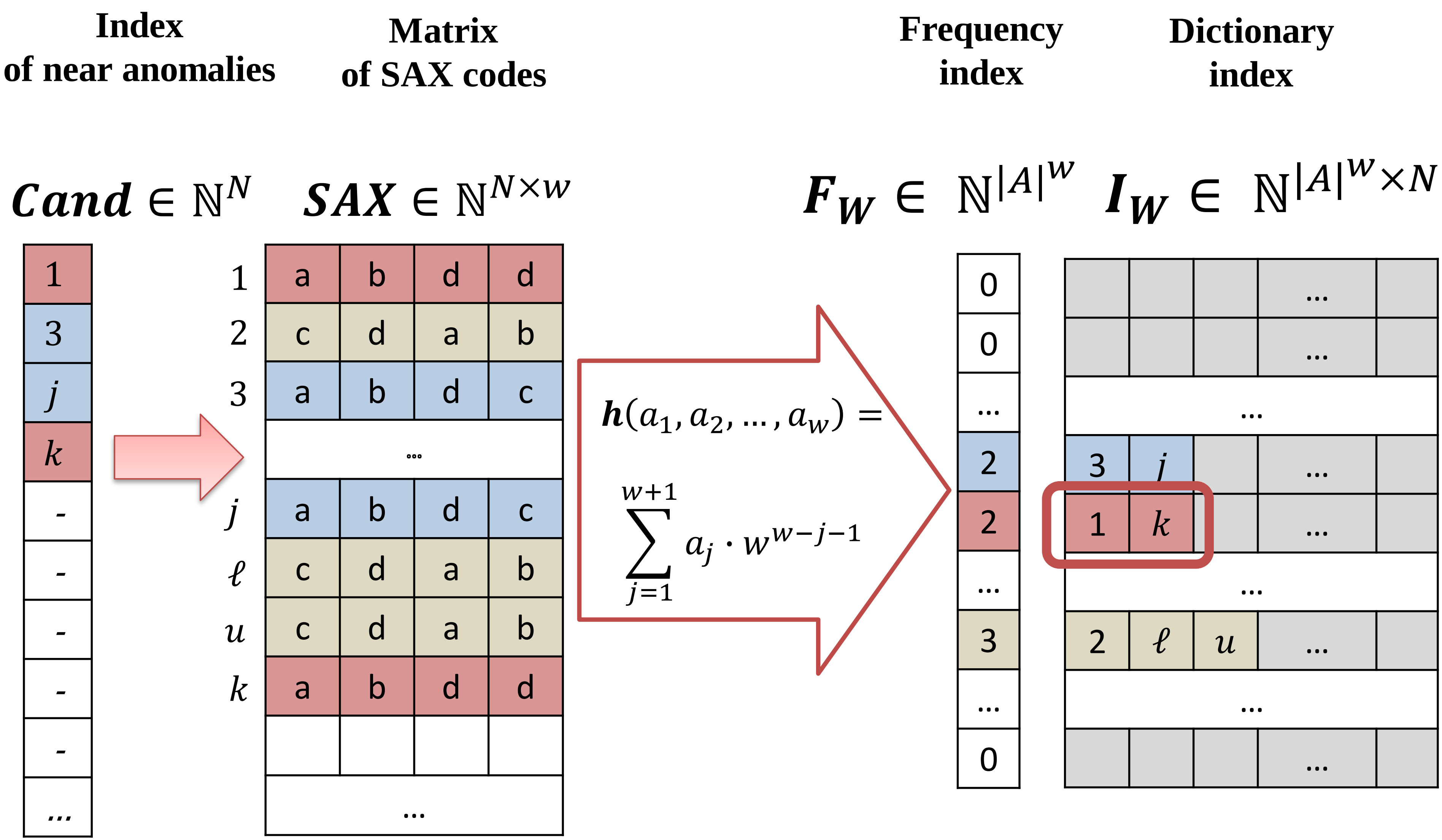
Dictionary index

$\boldsymbol{I_W} \in \mathbb{N}^{|A|^w \times N}$

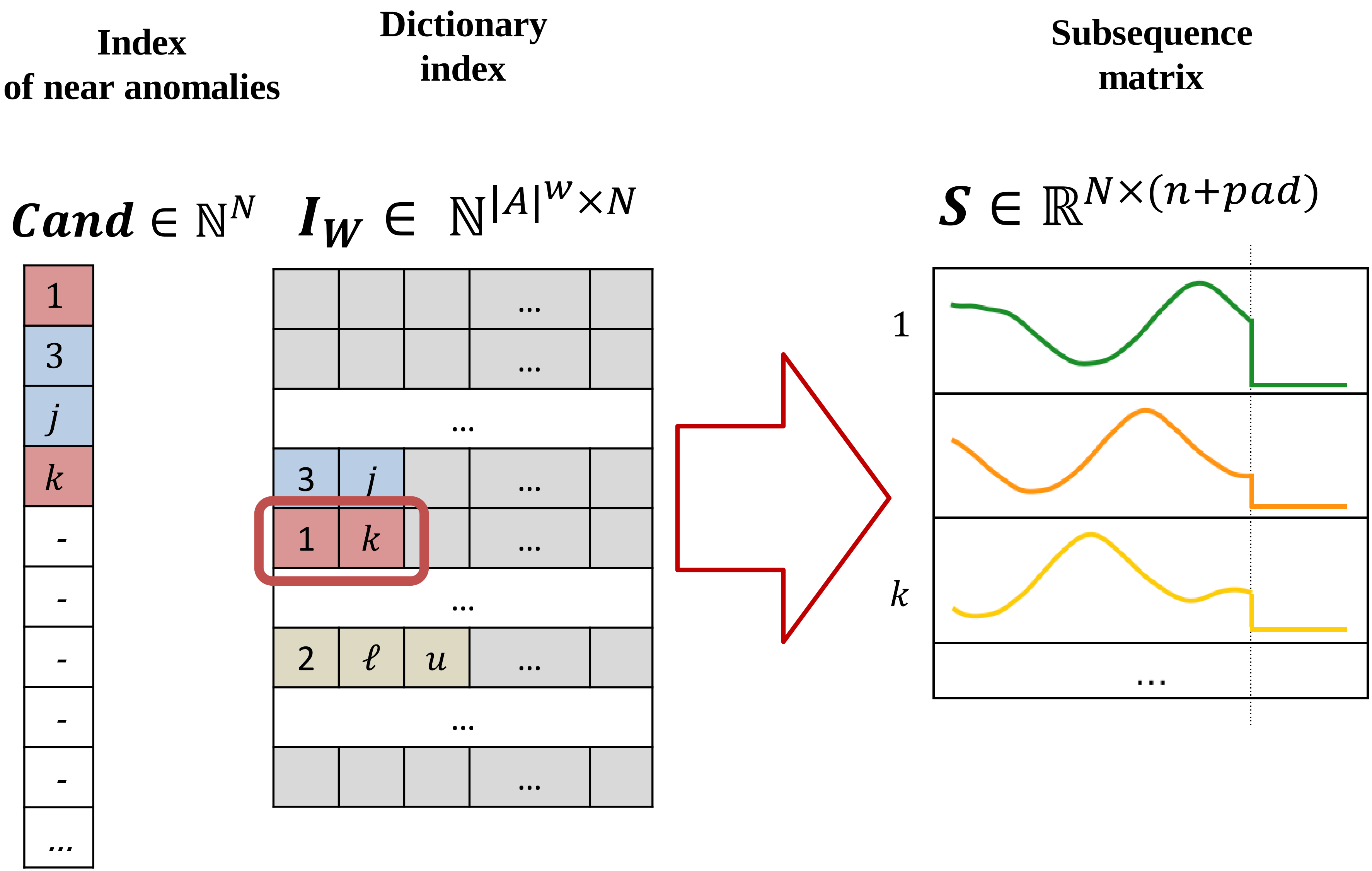
			…	
			…	
…				
3	j		…	
1	k		…	
…				
2	ℓ	u	…	
…				
			…	

Placement of characters in alphabet A by w characters with repetitions

Indirect access to subsequences



Iteration of subsequences



Parallel anomaly detection

1. Select candidates

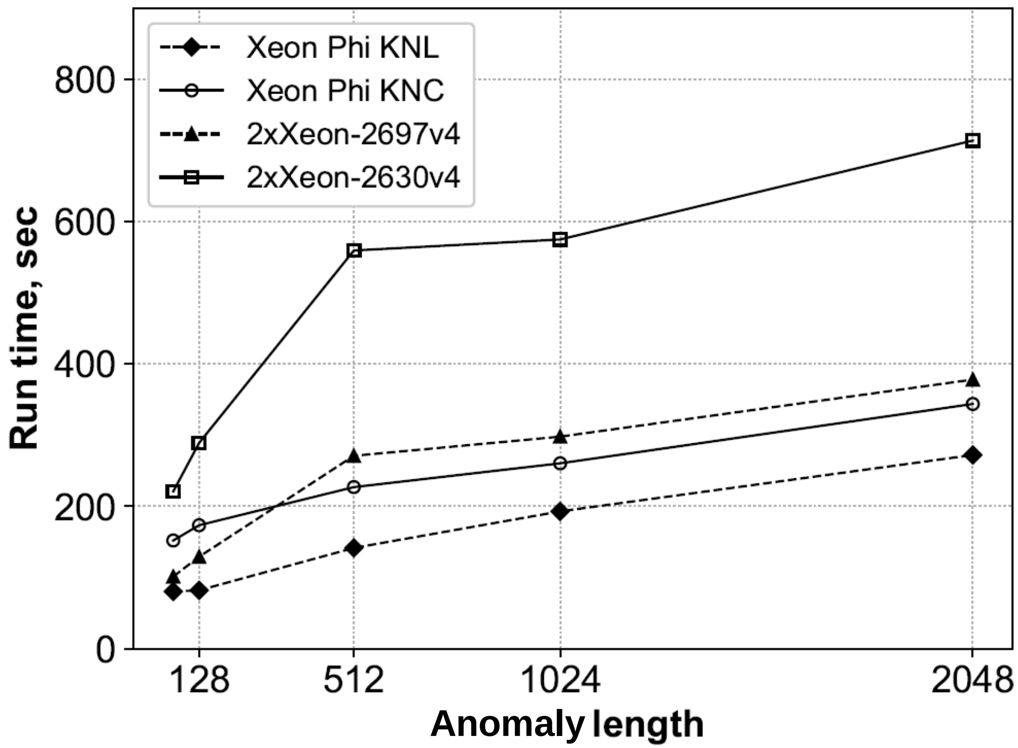
$d_{anomaly} \leftarrow 0$; $d_{NN} \leftarrow \infty$
for $C_i \in \boldsymbol{NearAnomalies}$
  PARALLEL
  for $C_j \in \boldsymbol{Neighbors}, \boldsymbol{Strangers}$
    $d \leftarrow \boldsymbol{ED}^2\,(C_i, C_j)$
    if $d < d_{anomaly}$
      break
    $d_{NN} \leftarrow \min(d, d_{NN})$
  $d_{anomaly} \leftarrow \max(d_{NN}, d_{anomaly})$
  if $d_{anomaly} < d_{NN}$ **then**
    $C_{anomaly} \leftarrow C_i$
return $\{C_{anomaly}, d_{anomaly}\}$

2. Refine

$d_{NN} \leftarrow \infty$
PARALLEL
for $C_i \in \boldsymbol{Others}$
  for $C_j \in \boldsymbol{Neighbors}, \boldsymbol{Strangers}$
    $d \leftarrow \boldsymbol{ED}^2\,(C_i, C_j)$
    if $d < d_{anomaly}$
      break
    $d_{NN} \leftarrow \min(d, d_{NN})$
  $d_{anomaly} \leftarrow \max(d_{NN}, d_{anomaly})$
  if $d_{anomaly} < d_{NN}$ **then**
    $C_{anomaly} \leftarrow C_i$
return $\{C_{anomaly}, \sqrt{d_{anomaly}}\}$

Speedup

Feature \ Device	Intel Xeon Phi SE10X (**KNC**)	Intel Xeon Phi 7290 (**KNL**)	2× Intel Xeon E5-2697v4	2× Intel Xeon E5-2630v4
Number of cores	**61**	**72**	2×16	2×10
Frequency, GHz	1.1	1.5	**2.6**	**2.2**
Peak. performance, TFLOPS	1.076	3.456	0.600	0.390



Conclusions

Big Data processing and analytics inside Parallel Relational DBMS – it is possible and feasible!

Thank you for paying attention! Questions?

Mikhail Zymbler

mzym@susu.ru