Logical Aspects of Massively Parallel and Distributed Systems

Frank Neven
Hasselt University
PODS Tutorial

June 29, 2016

Logical aspects of massively parallel and distributed systems

Topics	Setting
Complexity of query evaluation Foundations of query optimization	*synchronized parallel systems*
Monotone recursive queries	*asynchronous distributed systems*

Topic #1: complexity of query evaluation

synchronous setting

Question

How can we understand the complexity of query processing on shared-nothing parallel architectures?

Topic #1: complexity of query evaluation

synchronous setting

Question

How can we understand the complexity of query processing on shared-nothing parallel architectures?

- Examples: MapReduce, PigLatin, Hive, Dremel, Shark, . . .

Topic #1: complexity of query evaluation

synchronous setting

Question

How can we understand the complexity of query processing on shared-nothing parallel architectures?

- Examples: MapReduce, PigLatin, Hive, Dremel, Shark, ...
- Characteristics:
  - Large numbers of servers
  - Computation proceeds in rounds consisting of a communication and computation phase

Topic #1: complexity of query evaluation

synchronous setting

Question

How can we understand the complexity of query processing on shared-nothing parallel architectures?

- Examples: MapReduce, PigLatin, Hive, Dremel, Shark, ...
- Characteristics:
  - Large numbers of servers
  - Computation proceeds in rounds consisting of a communication and computation phase
- Focus: *join queries* and *Shares/HyperCube algorithm*

Topic #2: foundations of query optimization

synchronous setting

Question

How can we avoid reshuffling of data when evaluating queries?

Topic #2: foundations of query optimization

synchronous setting

Question

How can we avoid reshuffling of data when evaluating queries?

- Framework for reasoning on data partitioning
- Parallel-correctness and transferability
- Focus: *join queries and single-round parallel algorithms*

Topic #3: Datalog and monotone queries

asynchronous setting

Question

When can we avoid synchronization barriers in asynchronous parallel systems?

Topic #3: Datalog and monotone queries

asynchronous setting

Question

When can we avoid synchronization barriers in asynchronous parallel systems?

- Coordination-freeness
- CALM: consistency and logical monotonicity
- Focus: *monotone and Datalog queries (and extensions)*

Outline

Outline

Massively Parallel Communication Model (MPC)

Aim of MPC [Koutris, Suciu, PODS 2011]
Understand the complexity of parallel query processing on shared-nothing architectures

Examples
MapReduce, Spark, Pig, Hive, Dremel, . . .

Massively Parallel Communication Model (MPC)

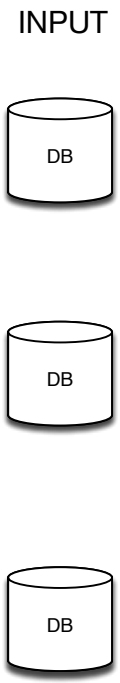
Aim of MPC [Koutris, Suciu, PODS 2011]
Understand the complexity of parallel query processing on shared-nothing architectures

Examples
MapReduce, Spark, Pig, Hive, Dremel, ...

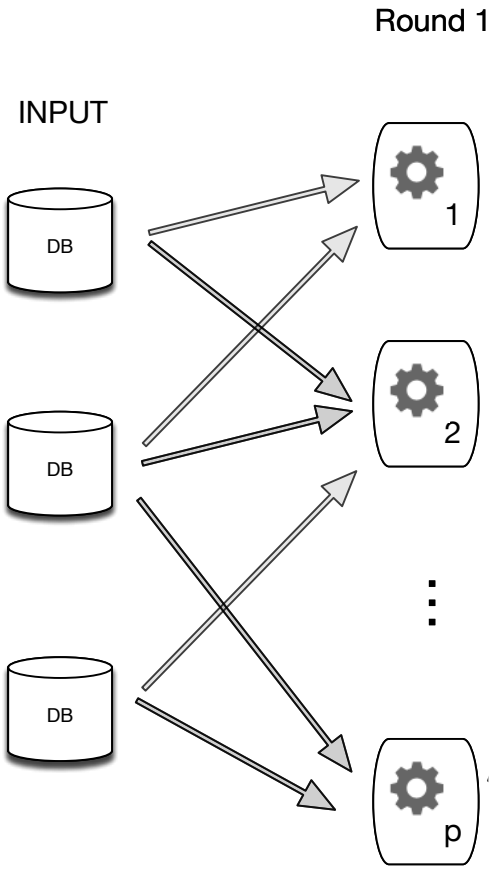
Characteristics

- Computation is performed by p servers connected by a complete network of private channels
- Computation proceeds in rounds:
  - *Communication phase:* servers exchange data
  - *Computation phase:* each server performs only local computation

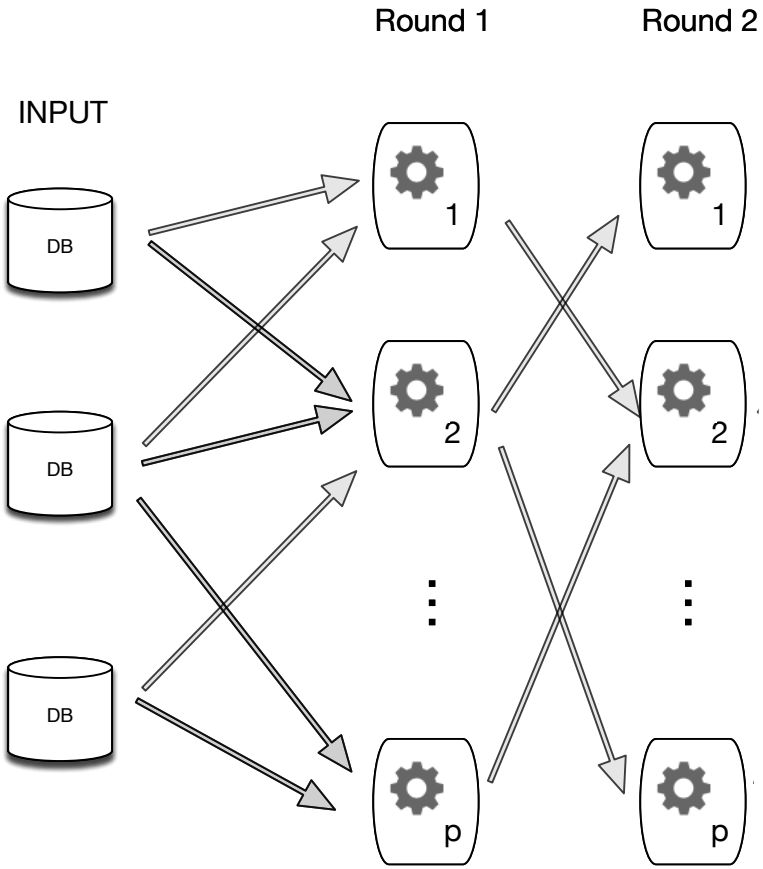
Massively Parallel Communication Model



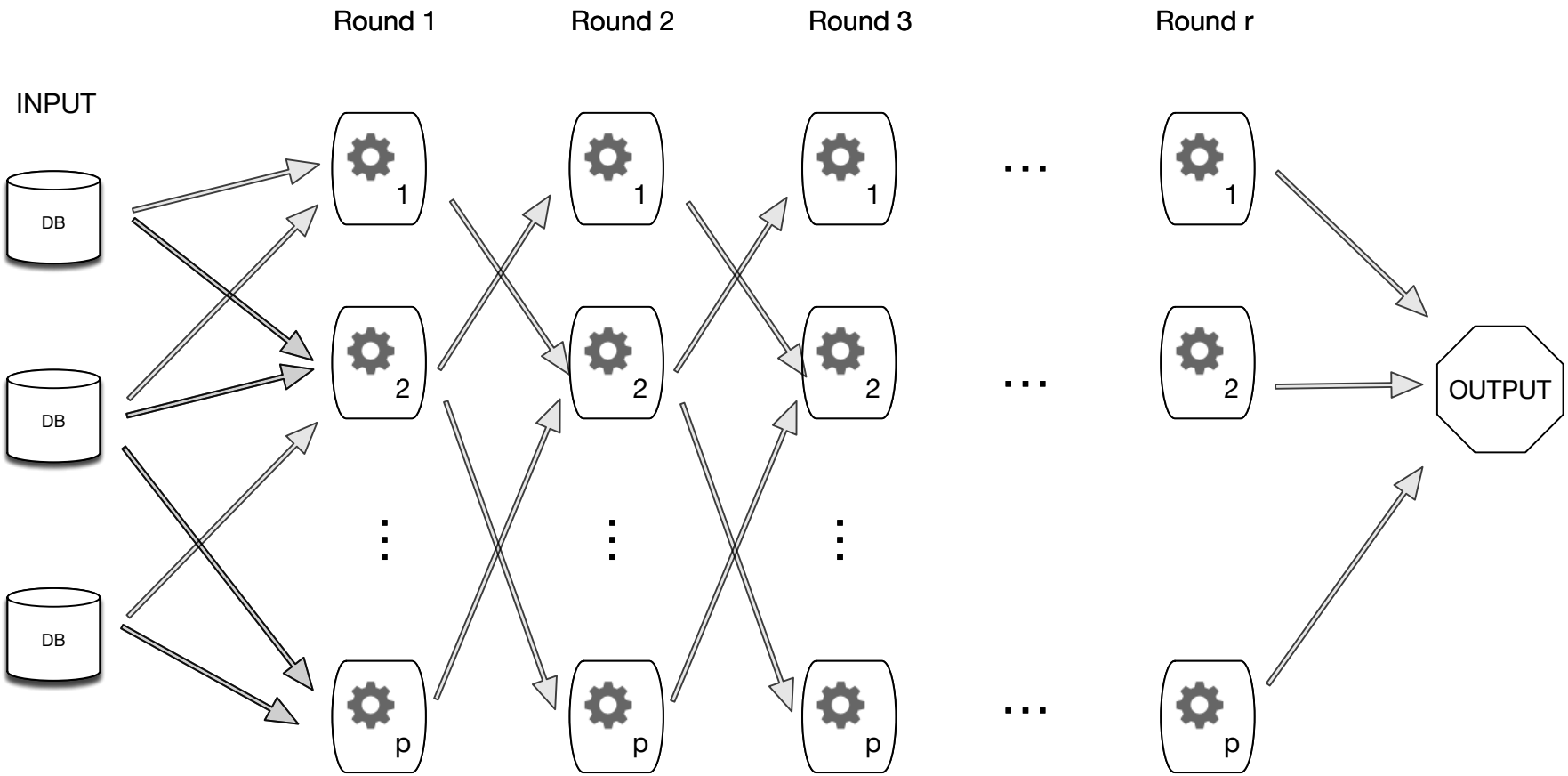
Massively Parallel Communication Model



Massively Parallel Communication Model



Massively Parallel Communication Model



How do we measure complexity?

Possible measures

1. Load:
   - The load at each server is the amount of data received by a server during a particular round.
   - Minimization targets:
     - Total load (communication cost)
     - Maximal load (load balancedness)
2. Number of rounds

Typically ignoring the complexity of local computations.

Outline

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

- Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$ (3-ary query)
- Identify every server with a coordinate in the 3-dimensional hypercube

$$[1, \alpha_x] \times [1, \alpha_y] \times [1, \alpha_z].$$

- α_x is the share of variable x
  (similar for y and z)

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

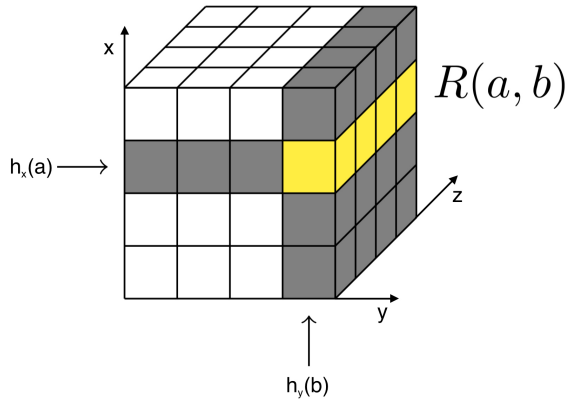
- Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$ (3-ary query)
- Identify every server with a coordinate in the 3-dimensional hypercube

$$[1, \alpha_x] \times [1, \alpha_y] \times [1, \alpha_z].$$

- α_x is the share of variable x
  (similar for y and z)
- hash function h_x mapping domain values to $[1, \alpha_x]$
  (similar for y and z)

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

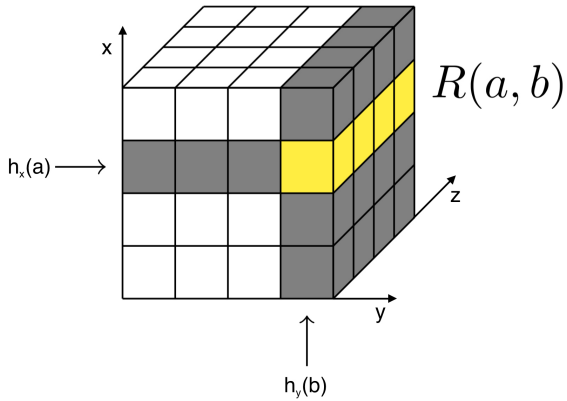


Communication phase:

- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple α_z times)

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

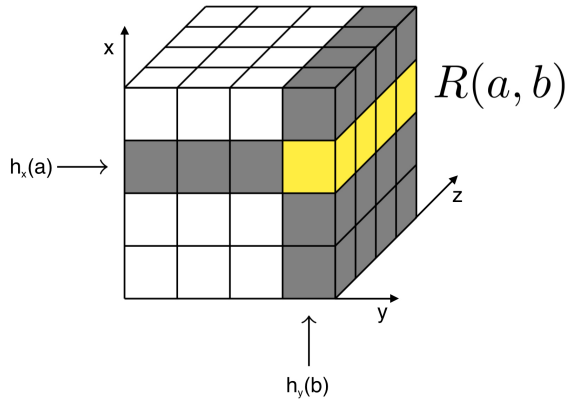


Communication phase:

- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple α_z times)
- $S(b, c)$ is sent to every server $(\star, h_y(b), h_z(c))$ (replicating every tuple α_x times)
- $T(c, a)$ is sent to every server $(h_x(a), \star, h_z(c))$ (replicating every tuple α_y times)

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$



Communication phase:

- $R(a, b)$ is sent to every server $(h_x(a), h_y(b), \star)$ (replicating every tuple α_z times)
- $S(b, c)$ is sent to every server $(\star, h_y(b), h_z(c))$ (replicating every tuple α_x times)
- $T(c, a)$ is sent to every server $(h_x(a), \star, h_z(c))$ (replicating every tuple α_y times)

Computation phase: All servers evaluate the triangle query.

Shares/HyperCube algorithm: single-round multi-join evaluation [Afrati, Ullman, EDBT 2010], [Beame, Koutris, Suciu, PODS 2013]

Triangle Query $H(x, y, z) \leftarrow R(x, y), S(y, z), T(z, x)$

Property

For every valuation

$$V = \{x \mapsto a, y \mapsto b, z \mapsto c\}$$

the required facts

$$\{R(a, b), S(b, c), T(c, a)\}$$

meet at server with coordinate

$$(h_x(a), h_y(b), h_z(c))$$

Shares/HyperCube: some results

- minimize total load: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]

Shares/HyperCube: some results

- minimize total load: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]
- minimize maximal load [Beame, Koutris, Suciu, PODS 2013, 2014]
  - lower bound: any 1-round algorithm requires at least load

$$M/p^{1/\tau^*(Q)}$$

    per server ($\tau^*(Q)$ is optimal fractional vertex cover number of Q)
  - HyperCube provides matching upper bound

Shares/HyperCube: some results

- minimize total load: solving Lagrangean equations [Afrati, Ullman, EDBT 2010]
- minimize maximal load [Beame, Koutris, Suciu, PODS 2013, 2014]
  - lower bound: any 1-round algorithm requires at least load

$$M/p^{1/\tau^*(Q)}$$

    per server ($\tau^*(Q)$ is optimal fractional vertex cover number of Q)
  - HyperCube provides matching upper bound
- Practical? [Chu, Balazinska, Suciu, SIGMOD 2015] [Afrati, Ullman, TKDE 2011]
  - rounding issues
  - *strong* for queries with large intermediate results
  - *weak* on queries with small output

Intermediate wrap-up:

Summary:

- MPC as a model to study parallel query evaluation
- Shares/HyperCube fundamental algorithm

Intermediate wrap-up:

Summary:

- MPC as a model to study parallel query evaluation
- Shares/HyperCube fundamental algorithm

Research opportunities:

- Skew [Beame, Koutris, Suciu, PODS 2014] [Afrati, Stasinopoulos, Ullman, Vassilakopoulos, 2015]
- Multi-round
  - GYM [Afrati, Joglekar, R, Salihoglu, Ullman, 2014]
  - lower-bound: any multi-round algorithm requires at least load

$$M/p^{1/\rho^*(Q)}$$

per server ($\rho^*(Q)$ is optimal fractional edge cover number of Q) [Beame, Koutris, Suciu, ICDT 2016]
  - matching upper bound for binary relations [Ketsman, Suciu, 2016]

Outline

Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts

Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy $\mathbf{P}$: a function mapping every fact $\mathbf{f}$ to a subset of servers $\mathbf{P}(\mathbf{f})$

Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy **P**: a function mapping every fact **f** to a subset of servers **P**(**f**)

Generic one-round algorithm
(relative to a distribution policy **P** and a query Q)

1. Reshuffle all data according to **P**
2. Evaluate Q locally at every server
3. Output then is the union of all local results

Observations on HyperCube

Distribution policies

- HyperCube reshuffles data on the granularity of facts
- Can be modeled by a distribution policy $\mathbf{P}$: a function mapping every fact $\mathbf{f}$ to a subset of servers $\mathbf{P}(\mathbf{f})$

Generic one-round algorithm
(relative to a distribution policy $\mathbf{P}$ and a query Q)

1. Reshuffle all data according to $\mathbf{P}$
2. Evaluate Q locally at every server
3. Output then is the union of all local results

Potential for query optimization

Can we avoid the reshuffle step in the generic one-round algorithm?

Reasoning on data distribution

First scenario. The data is already distributed over the servers.
Can we use the generic one-round algorithm to evaluate a given query using the current distribution?

- *if yes:* no data reshuffling needed
- *if no:* choose another distribution policy that works

Reasoning on data distribution

First scenario. The data is already distributed over the servers.
Can we use the generic one-round algorithm to evaluate a given query using the current distribution?

- *if yes:* no data reshuffling needed
- *if no:* choose another distribution policy that works

Decision problem: parallel-correctness

Reasoning on data distribution

Second scenario.
It may be unpractical to reason about distribution policies:

- too complex
- current distribution policy is unknown or hidden
- not chosen yet

Reasoning on data distribution

Second scenario.
It may be unpractical to reason about distribution policies:

- too complex
- current distribution policy is unknown or hidden
- not chosen yet

Looking for ways to infer that two queries can *always* be evaluated correctly after another *without* an intermediate reshuffling step.

Reasoning on data distribution

Second scenario.
It may be unpractical to reason about distribution policies:

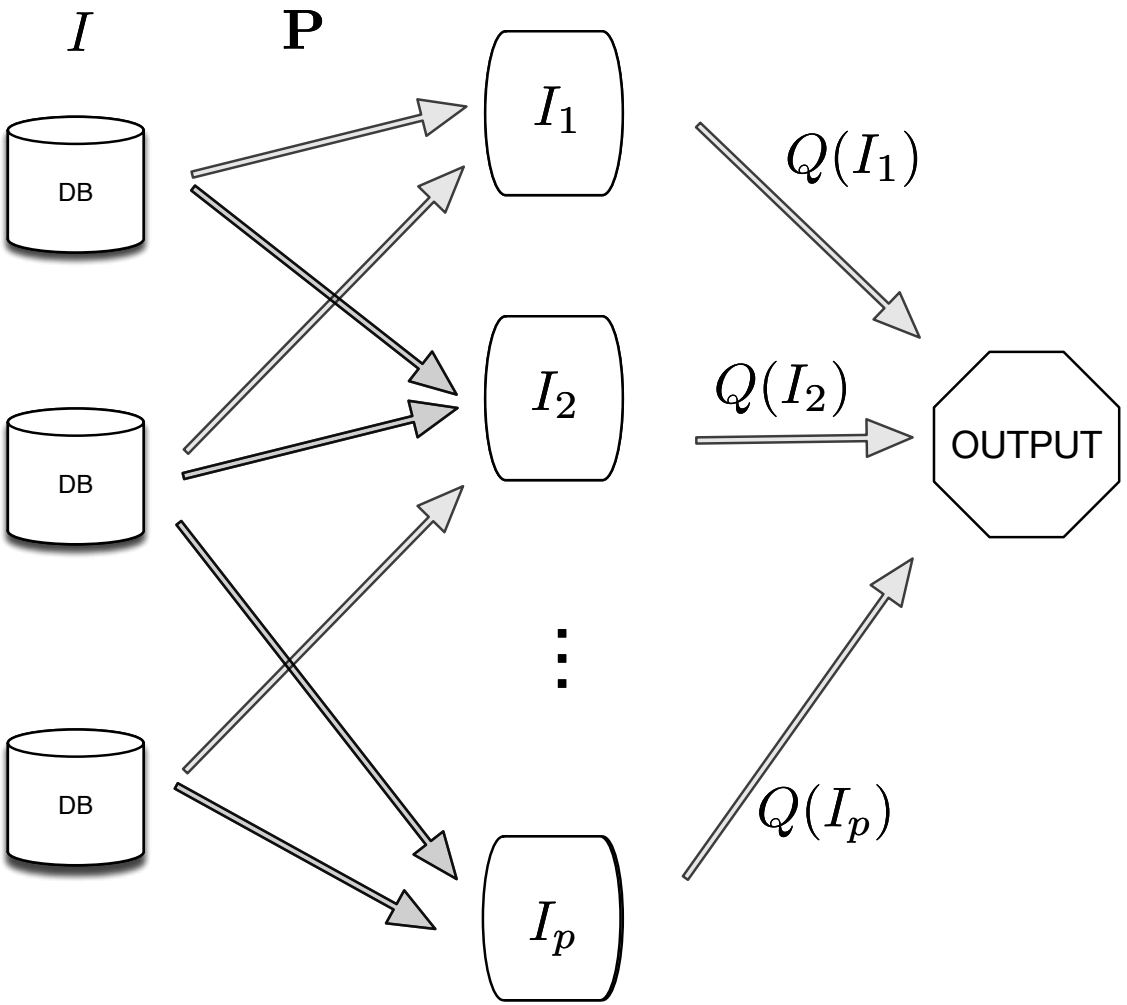
- too complex
- current distribution policy is unknown or hidden
- not chosen yet

Looking for ways to infer that two queries can *always* be evaluated correctly after another *without* an intermediate reshuffling step.

Decision problem: parallel-correctness transfer

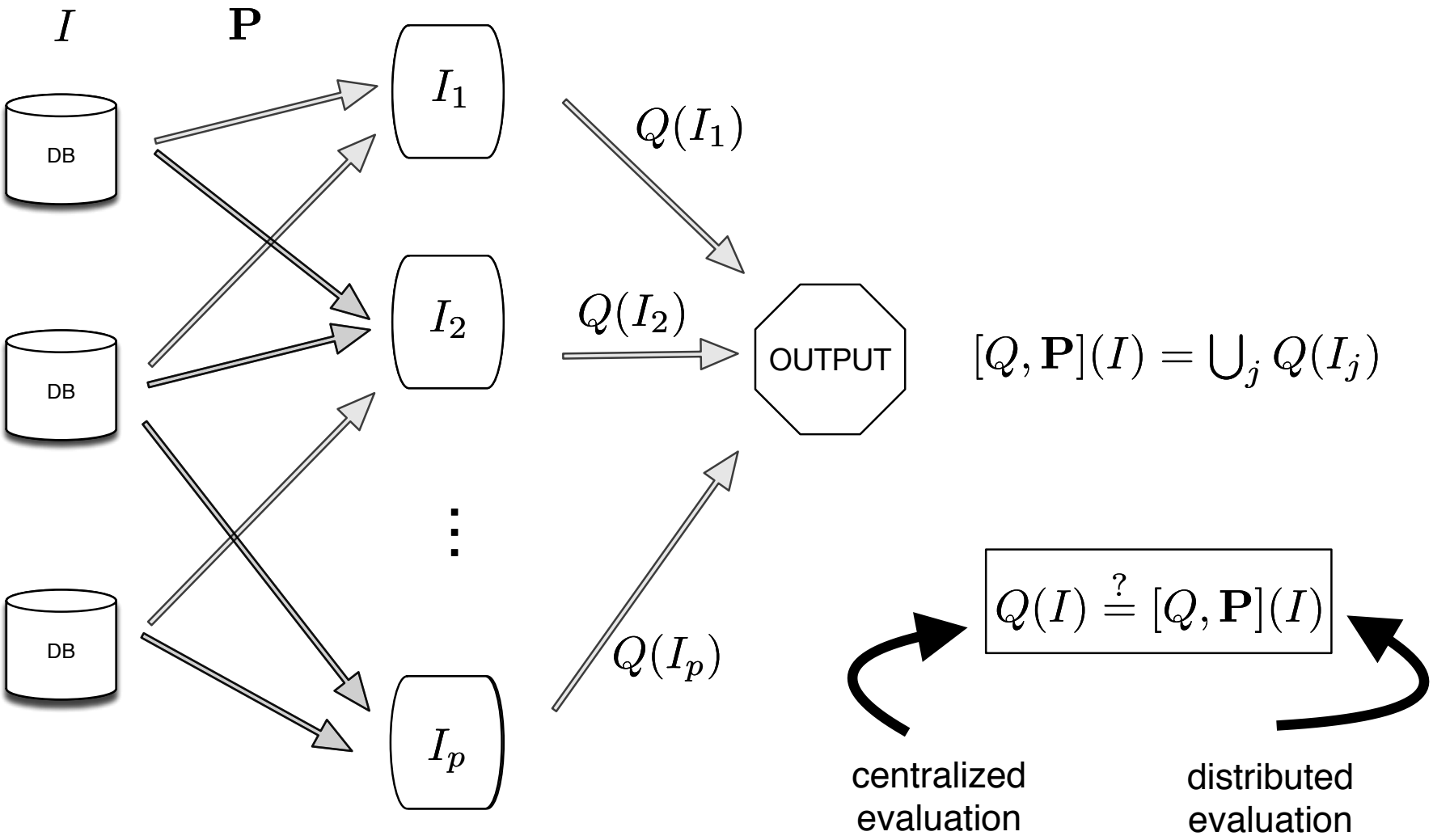
Outline

Parallel-correctness



$$[Q, \mathbf{P}](I) = \bigcup_j Q(I_j)$$

Parallel-correctness



Parallel-correctness: a bit more formal

Definition

A query Q is parallel-correct on I w.r.t. $\mathbf{P}$ iff $Q(I) = [Q, \mathbf{P}](I)$.

If Q is monotone, then always $Q(I) \supseteq [Q, \mathbf{P}](I)$

Parallel-correctness: a bit more formal

Definition

A query Q is parallel-correct on I w.r.t. $\mathbf{P}$ iff $Q(I) = [Q, \mathbf{P}](I)$.

If Q is monotone, then always $Q(I) \supseteq [Q, \mathbf{P}](I)$

Definition

A query Q is parallel-correct w.r.t. $\mathbf{P}$ iff Q is parallel-correct w.r.t. $\mathbf{P}$ on *every* I.

Parallel-correctness for conjunctive queries

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x, z)}_{\text{head}_Q} \leftarrow \underbrace{R(x, y), S(y, z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x, z)}_{\text{head}_Q} \leftarrow \underbrace{R(x, y), S(y, z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.
If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x,z)}_{\text{head}_Q} \leftarrow \underbrace{R(x,y), S(y,z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.
If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

Observation

If for every valuation V for Q, there is a node κ in the network for which

$$V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa), \qquad (\dagger)$$

then Q is parallel-correct w.r.t. $\mathbf{P}$

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x,z)}_{\text{head}_Q} \leftarrow \underbrace{R(x,y), S(y,z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.
If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

Observation

If for every valuation V for Q, there is a node κ in the network for which

$$V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa), \qquad (\dagger)$$

then Q is parallel-correct w.r.t. $\mathbf{P}$

- $(\dagger)$ implies that for every I, $Q(I) \subseteq [Q, \mathbf{P}](I)$

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x,z)}_{\text{head}_Q} \leftarrow \underbrace{R(x,y), S(y,z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.
If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

Observation

If for every valuation V for Q, there is a node κ in the network for which

$$V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa), \qquad (\dagger)$$

then Q is parallel-correct w.r.t. $\mathbf{P}$

- $(\dagger)$ implies that for every I, $Q(I) \subseteq [Q, \mathbf{P}](I)$
- $Q(I) \supseteq [Q, \mathbf{P}](I)$ follows by monotonicity

Parallel-correctness for conjunctive queries

A valuation V for a conjunctive query

$$Q : \underbrace{H(x,z)}_{\text{head}_Q} \leftarrow \underbrace{R(x,y), S(y,z)}_{\text{body}_Q}$$

is a mapping from the variables of Q to domain elements.
If $V(\text{body}_Q) \subseteq I$ then $V(\text{head}_Q) \in Q(I)$.

Observation

If for every valuation V for Q, there is a node κ in the network for which

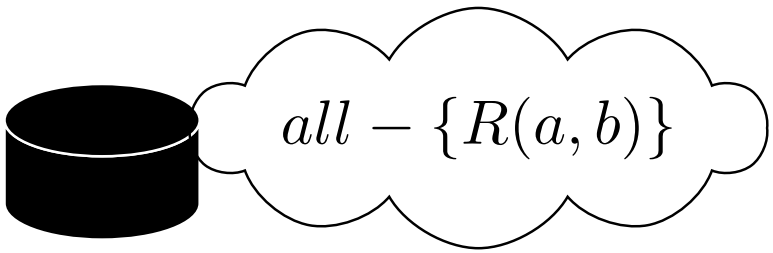
$$V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa), \qquad (\dagger)$$

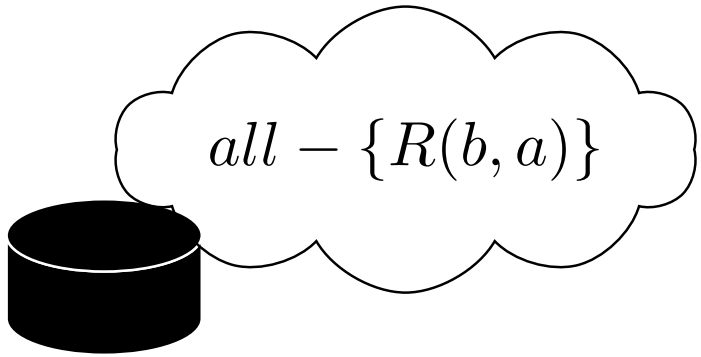
then Q is parallel-correct w.r.t. $\mathbf{P}$

- $(\dagger)$ implies that for every I, $Q(I) \subseteq [Q, \mathbf{P}](I)$
- $Q(I) \supseteq [Q, \mathbf{P}](I)$ follows by monotonicity
- Every CQ is parallel-correct w.r.t. its HyperCube distribution

Sufficient but not a necessary condition: counterexample

Distribution policy $\boldsymbol{P}$

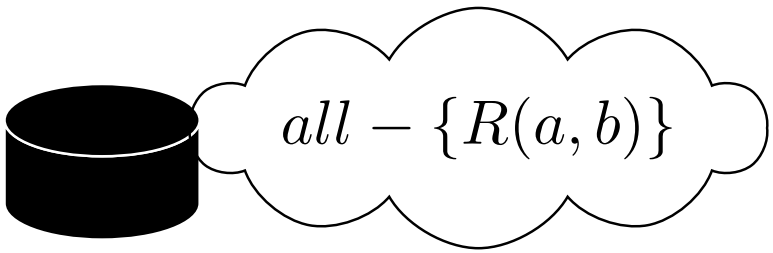


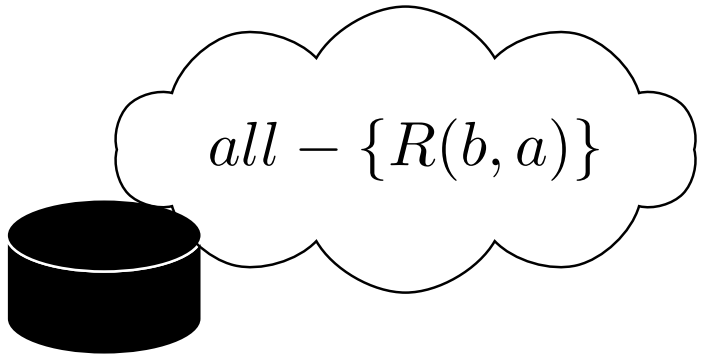


$$Q : H(x,z) \leftarrow R(x,y), R(y,z), R(x,x)$$

Sufficient but not a necessary condition: counterexample

Distribution policy $\boldsymbol{P}$





$$Q : H(x,z) \leftarrow R(x,y), R(y,z), R(x,x)$$

$V = \{x, z \to a, y \to b\}$

Derives:

$H(a,a)$

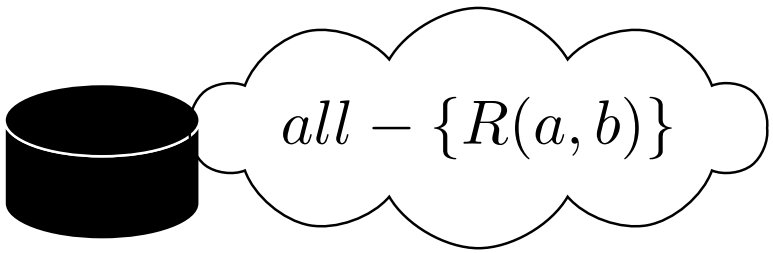
$V(\text{body}_Q)$ requires:

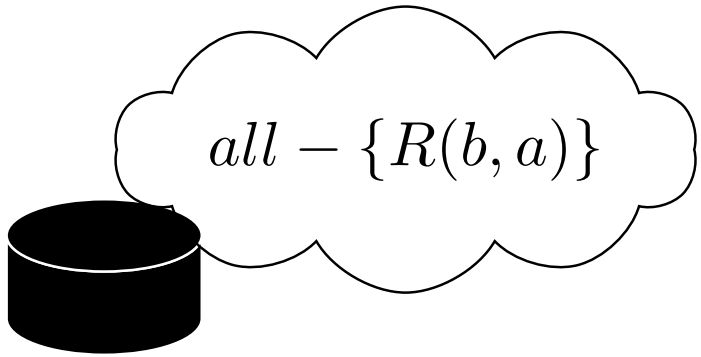
$R(a,b) \quad R(b,a) \quad R(a,a)$

$\nexists \kappa : V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$

Sufficient but not a necessary condition: counterexample

Distribution policy $\boldsymbol{P}$





$$Q : H(x,z) \leftarrow R(x,y), R(y,z), R(x,x)$$

$V = \{x, z \to a, y \to b\}$

Derives:

$H(a,a)$

$V(\text{body}_Q)$ requires:

$R(a,b)$ $R(b,a)$ $R(a,a)$

$\nexists \kappa : V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$

$V' = \{x, y, z \to a\}$

Derives:

$H(a,a)$

$V'(\text{body}_Q)$ requires:

$R(a,a)$

$\exists \kappa : V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$

Parallel-correctness for CQs: characterization

Lemma

Q is parallel-correct w.r.t. $\mathbf{P}$ *iff*

for every minimal valuation V *for* Q,

there is a node κ *in the network for which*

$$V(body_Q) \subseteq \mathbf{P}^{-1}(\kappa).$$

Definition

V is minimal if no V' exists, where

- $V'(\text{head}_Q) = V(\text{head}_Q)$; and,
- $V'(\text{body}_Q) \subsetneq V(\text{body}_Q)$.

Parallel-correctness for CQs: complexity

Lemma

Q is parallel-correct w.r.t. $\mathbf{P}$ iff
for every minimal valuation V for Q,
there is a node κ in the network for which

$$V(body_Q) \subseteq \mathbf{P}^{-1}(\kappa).$$

Parallel-correctness for CQs: complexity

Lemma

Q is parallel-correct w.r.t. $\mathbf{P}$ *iff*
for every minimal valuation V *for* Q,
there is a node κ *in the network for which*

$$V(body_Q) \subseteq \mathbf{P}^{-1}(\kappa).$$

- bound on the size of representations of nodes κ
- bound on the complexity of the test $V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$.

Parallel-correctness for CQs: complexity

Lemma

Q is parallel-correct w.r.t. **P** *iff*
for every minimal valuation V for Q,
there is a node κ in the network for which

$$V(body_Q) \subseteq \mathbf{P}^{-1}(\kappa).$$

- bound on the size of representations of nodes κ
- bound on the complexity of the test $V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$.

Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)

Deciding whether a CQ is parallel-correct w.r.t. **P** *is Π_2^P-complete.*

Parallel-correctness for CQs: complexity

Lemma

Q is parallel-correct w.r.t. **P** *iff*
for every minimal valuation V for Q,
there is a node κ in the network for which

$$V(body_Q) \subseteq \mathbf{P}^{-1}(\kappa).$$

- bound on the size of representations of nodes κ
- bound on the complexity of the test $V(\text{body}_Q) \subseteq \mathbf{P}^{-1}(\kappa)$.

Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)

Deciding whether a CQ is parallel-correct w.r.t. **P** *is Π_2^P-complete.*

Reduces to reasoning on properties of minimal valuations.

Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $UCQ^{\neq}$ is parallel-correct is Π_2^P-complete.

Reduces again to reasoning on minimal valuations.

Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $UCQ^{\neq}$ is parallel-correct is Π_2^P-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $CQ^{\neg}$ is parallel-correct is coNEXPTIME-complete.

Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $UCQ^{\neq}$ is parallel-correct is Π_2^P-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $CQ^{\neg}$ is parallel-correct is coNEXPTIME-complete.

- Non-monotone: parallel-soundness, parallel-completeness

Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $UCQ^{\neq}$ is parallel-correct is Π_2^P-complete.

Reduces again to reasoning on minimal valuations.

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $CQ^{\neg}$ is parallel-correct is coNEXPTIME-complete.

- Non-monotone: parallel-soundness, parallel-completeness
- Upper bound: bound on size of smallest counterexample.
- Lower bound: reduction from $CQ^{\neg}$ containment

Parallel-correctness beyond CQs: adding unions, inequalities, negation

Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $UCQ^{\neq}$ is parallel-correct is Π_2^P-complete.

Reduces again to reasoning on minimal valuations.

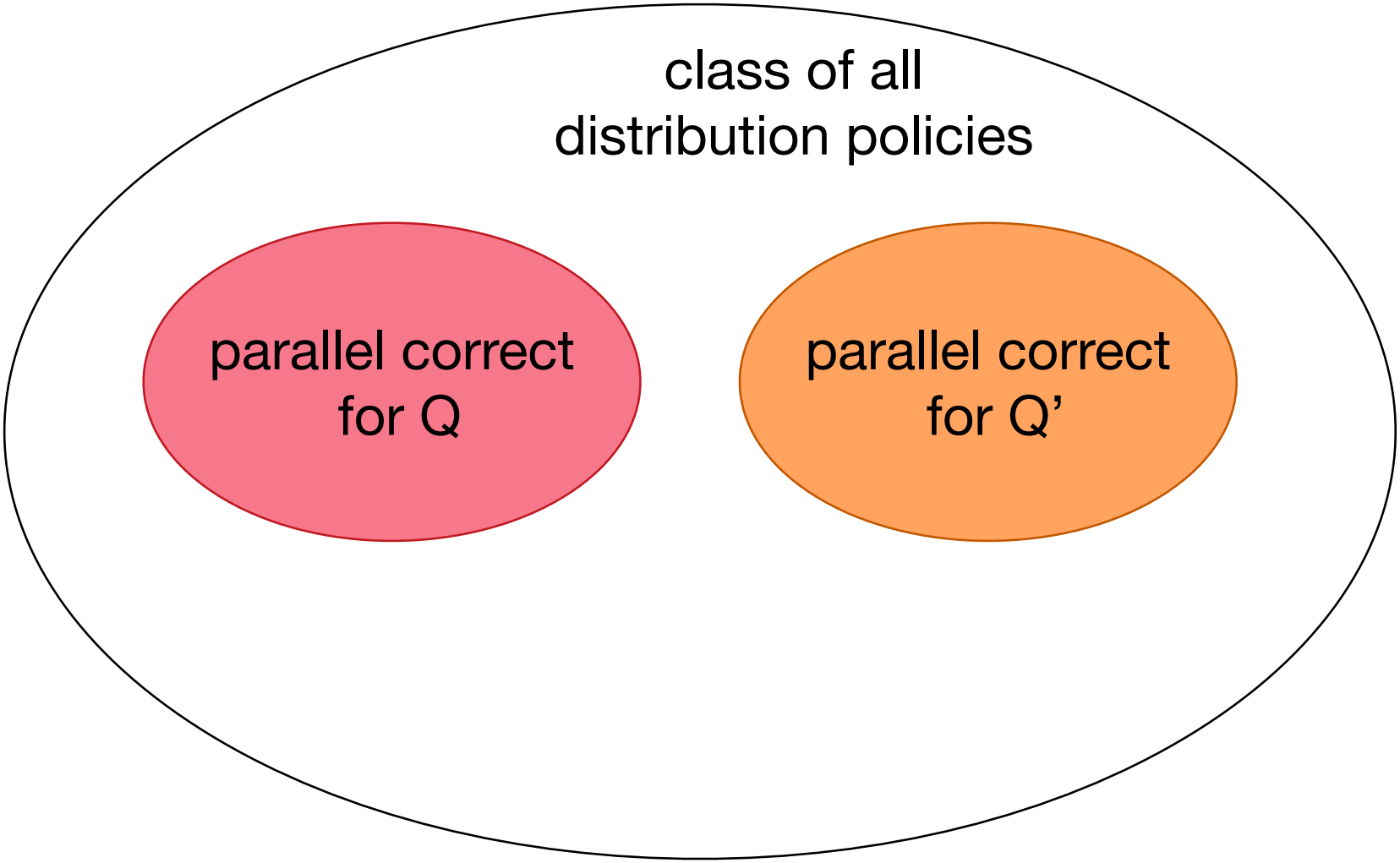
Theorem (Geck, Ketsman, N., Schwentick, 2016)

Deciding whether a $CQ^{\neg}$ is parallel-correct is coNEXPTIME-complete.

- Non-monotone: parallel-soundness, parallel-completeness
- Upper bound: bound on size of smallest counterexample.
- Lower bound: reduction from $CQ^{\neg}$ containment
- Requires reasoning on possible counterexamples that can be of exponential size.

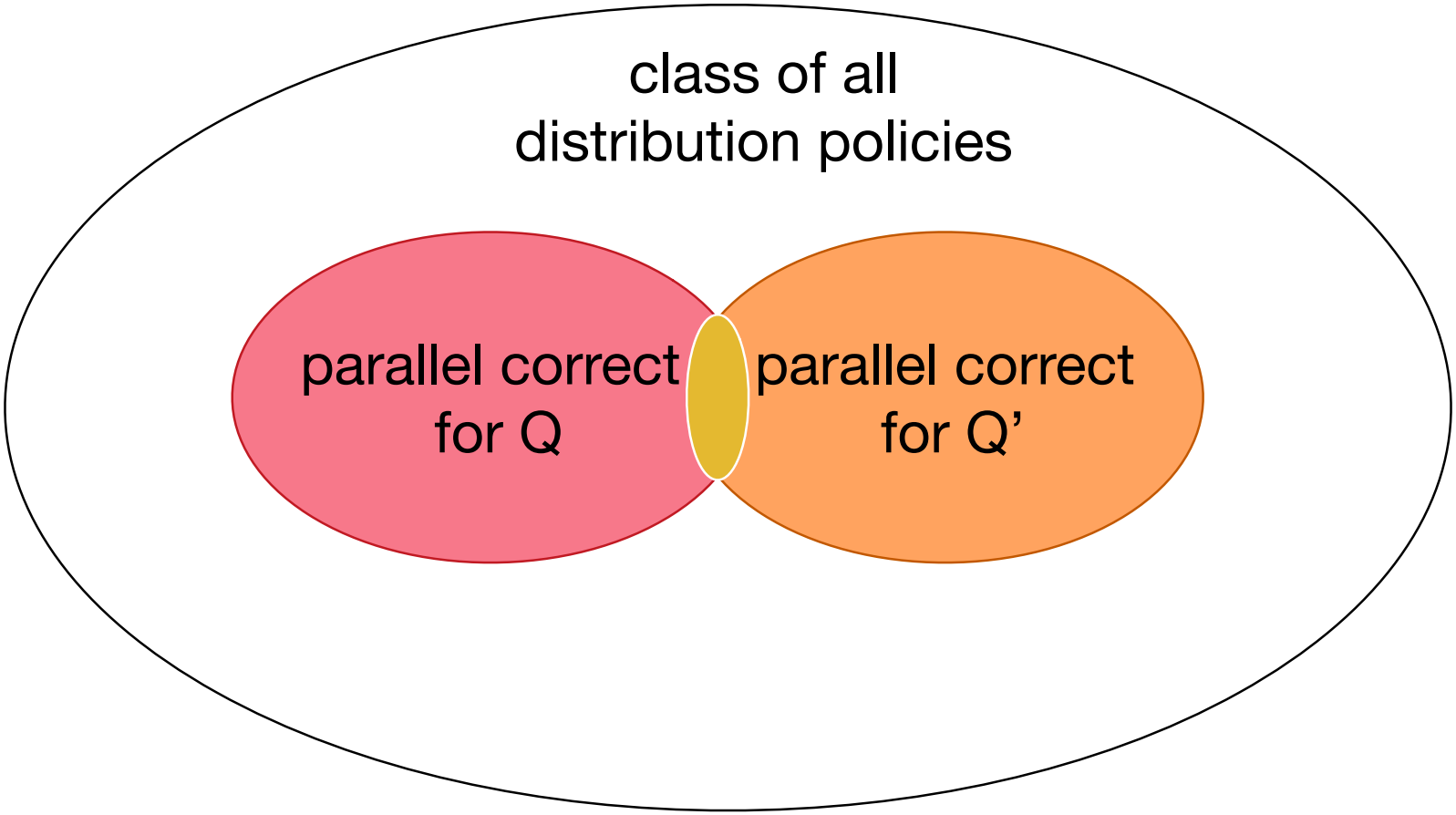
Outline

Evaluating Q' after Q (for free)



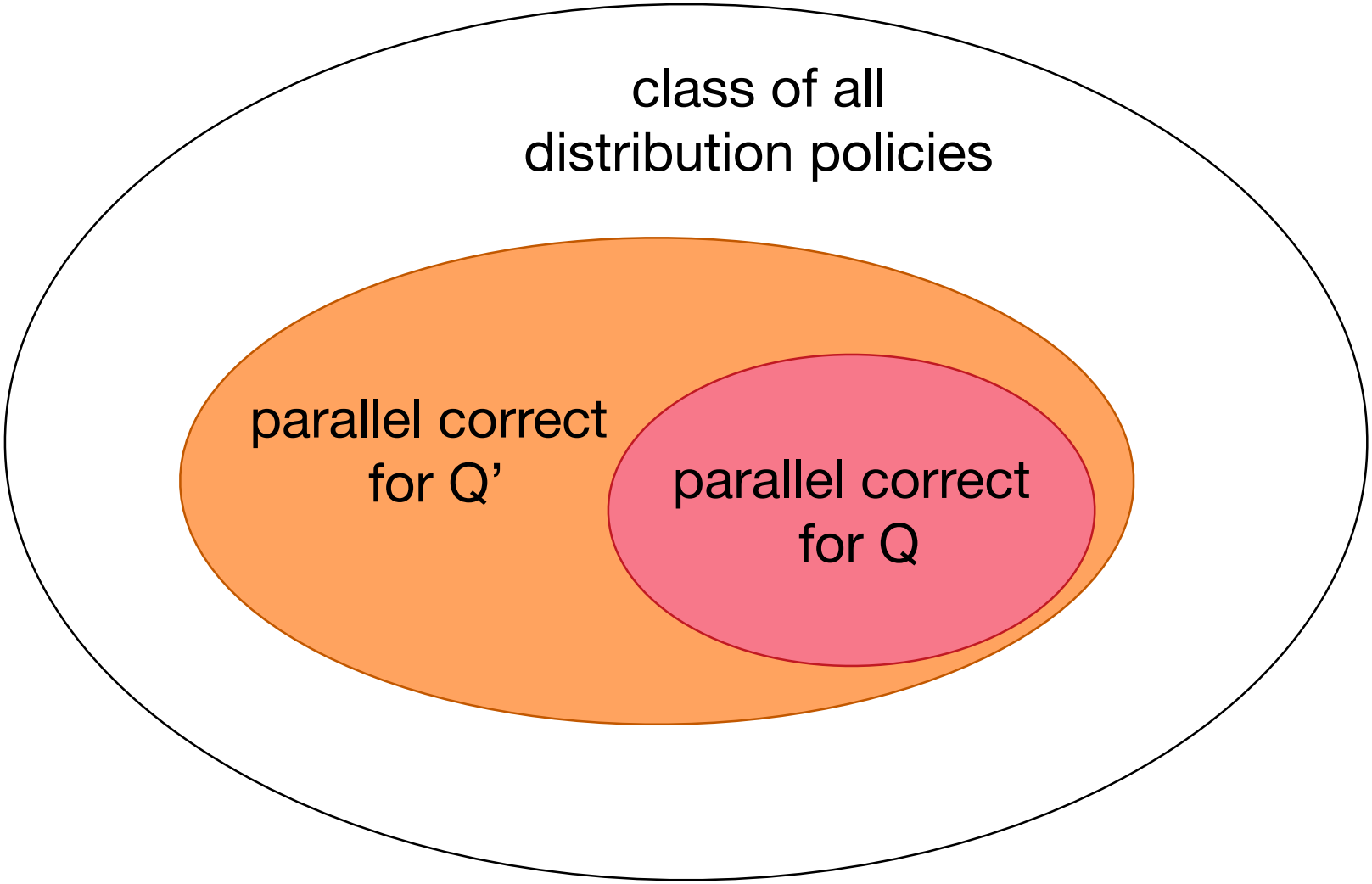
Q' can not be evaluated for free after Q

Evaluating Q' after Q (for free)



Q' can sometimes be evaluated for free after Q

Evaluating Q' after Q (for free)



Q' can always be evaluated for free after Q

Transferability of parallel-correctness

Definition

Parallel-correctness transfers from Q to Q', denoted $Q \rightarrow_{\mathcal{T}} Q'$, iff Q' is parallel-correct on *every* distribution policy for which Q is parallel-correct.

Transferability of parallel-correctness

Definition

Parallel-correctness transfers from Q to Q', denoted $Q \rightarrow_T Q'$, iff Q' is parallel-correct on *every* distribution policy for which Q is parallel-correct.

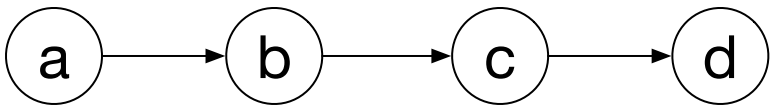
$$Q() \leftarrow R(x,y), R(y,z), R(z,w) \quad \overset{?}{\rightarrow}_T \quad Q'() \leftarrow R(x,y), R(y,x)$$

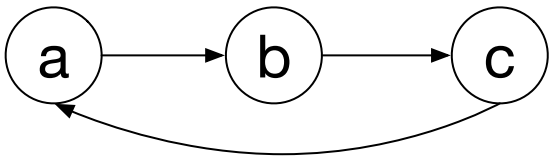
Transferability of parallel-correctness

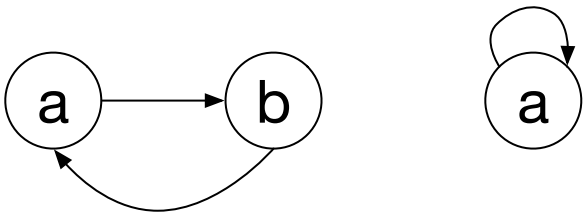
Definition

Parallel-correctness transfers from Q to Q', denoted $Q \rightarrow_T Q'$, iff Q' is parallel-correct on *every* distribution policy for which Q is parallel-correct.

$Q() \leftarrow R(x,y), R(y,z), R(z,w) \quad \overset{?}{\rightarrow}_T \quad Q'() \leftarrow R(x,y), R(y,x)$







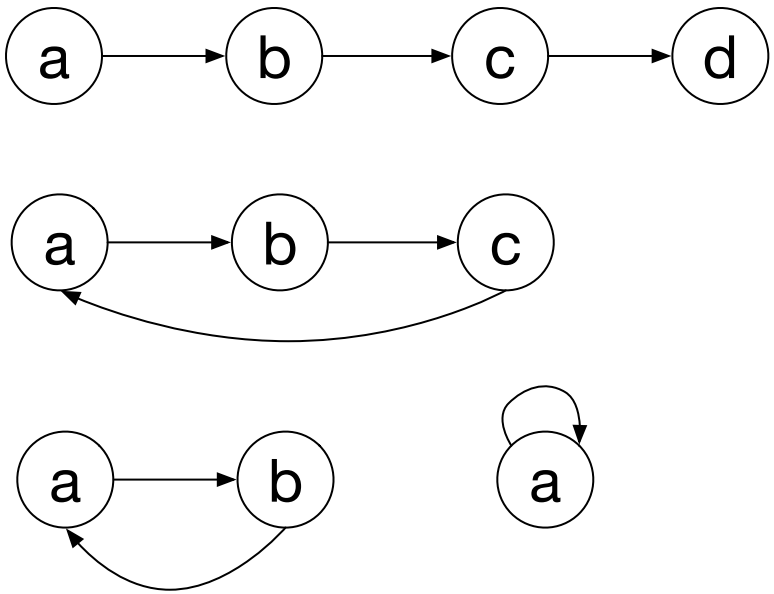
facts required by $V(\text{body}_Q)$

Transferability of parallel-correctness

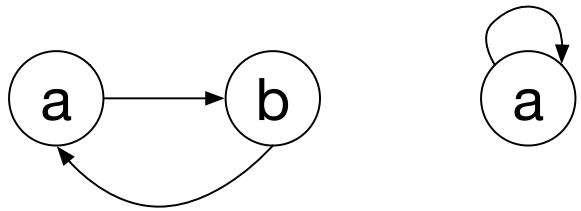
Definition

Parallel-correctness transfers from Q to Q', denoted $Q \rightarrow_T Q'$, iff Q' is parallel-correct on *every* distribution policy for which Q is parallel-correct.

$Q() \leftarrow R(x,y), R(y,z), R(z,w) \quad \overset{?}{\rightarrow}_T \quad Q'() \leftarrow R(x,y), R(y,x)$



facts required by $V(\text{body}_Q)$



facts required by $V(\text{body}_{Q'})$

Transferability for CQs: characterization and complexity

Lemma

For conjunctive queries Q and Q', $Q \rightarrow_T Q'$ iff
for every minimal valuation V' for Q'
there is a minimal valuation V for Q such that

$$V'(body_{Q'}) \subseteq V(body_Q).$$

Transferability for CQs: characterization and complexity

Lemma

For conjunctive queries Q and Q', $Q \rightarrow_T Q'$ iff
for every minimal valuation V' for Q'
there is a minimal valuation V for Q such that

$$V'(body_{Q'}) \subseteq V(body_Q).$$

Theorem (Ameloot, Geck, Ketsman, N., Schwentick, 2015)

Deciding $Q \rightarrow_T Q'$ is Π_3^P-complete.

Only depends on the size of queries! Independent on representation of distribution policies.

Lowering complexity

Lemma

For conjunctive queries Q and Q', $Q \rightarrow_T Q'$ iff
for every minimal valuation V' for Q'
there is a minimal valuation V for Q such that

$$V'(body_{Q'}) \subseteq V(body_Q).$$

Lowering complexity

Lemma

For conjunctive queries Q and Q', $Q \rightarrow_T Q'$ iff
for every minimal valuation V' for Q'
there is a minimal valuation V for Q such that

$$V'(body_{Q'}) \subseteq V(body_Q).$$

Strongly minimal CQs

- A CQ is strongly minimal if every valuation is minimal
- Example: full CQs, CQs without self-joins, ...

Lowering complexity

Lemma

For conjunctive queries Q and Q', $Q \rightarrow_T Q'$ iff
for every minimal valuation V' for Q'
there is a minimal valuation V for Q such that

$$V'(body_{Q'}) \subseteq V(body_Q).$$

Strongly minimal CQs

- A CQ is strongly minimal if every valuation is minimal
- Example: full CQs, CQs without self-joins, ...

Theorem

Let Q be a strongly minimal CQ and Q' be a CQ.

- *Deciding $Q \rightarrow_T Q'$ is NP-complete.*
- *Deciding parallel-correctness of Q w.r.t. a distribution policy $\mathbf{P}$ is in coNP.*

Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Research opportunities:

- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages

Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Research opportunities:

- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages
- Extended setting:
  - more liberal single-round algorithms
  - multi-round algorithms

Intermediate wrap-up: parallel-correctness and transferability

Summary:

- Framework for reasoning about correctness of *arbitrary* distribution policies
- Intuitive characterization for CQs in terms of *minimal valuations*

Research opportunities:

- Current setting:
  - no tractable restriction
  - decidability frontier for more expressive query languages
- Extended setting:
  - more liberal single-round algorithms
  - multi-round algorithms
- Practical considerations
  - e.g., if not-parallel correct, minimal way to move to a PC one

Outline

Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of rounds (communication/computation sequence)

Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of rounds (communication/computation sequence)
- Asynchronous computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit

Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of rounds (communication/computation sequence)
- Asynchronous computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit
- Declarative distributed computing:
  - approach where distributed computations are programmed using extensions of Datalog
  - Examples: Webdamlog [Abiteboul et al. 2013], Netlog [Grumbach, Wang, 2010], Dedalus [Hellerstein 2010], NDlog [Loo et al, 2006], $\mathcal{D}$ [Ma et al, 2013], ...

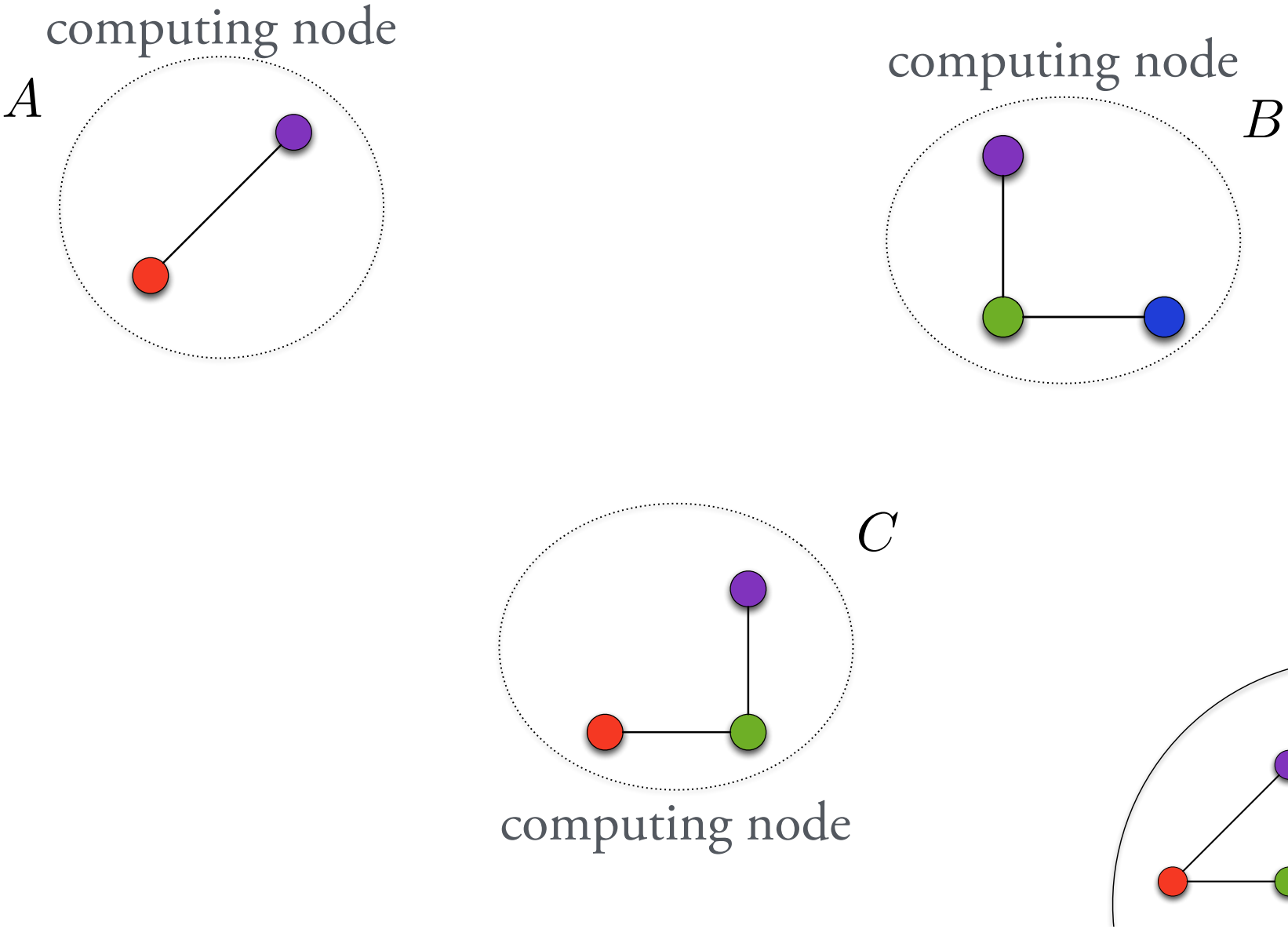
Synchronous versus asynchronous

- MPC makes synchronization explicit by modeling computation as a sequence of rounds (communication/computation sequence)
- Asynchronous computation model:
  - computation nodes are allowed to communicate freely
  - synchronization can be implicit
- Declarative distributed computing:
  - approach where distributed computations are programmed using extensions of Datalog
  - Examples: Webdamlog [Abiteboul et al. 2013], Netlog [Grumbach, Wang, 2010], Dedalus [Hellerstein 2010], NDlog [Loo et al, 2006], $\mathcal{D}$ [Ma et al, 2013], ...
- Our focus:
  - Which fragments/extensions of Datalog can be asynchronously evaluated in a coordination-free manner?
  - *Setting:* messages can never be lost but can be arbitrarily delayed.
  - *Requirement:* output can never be retracted, should eventually be complete

Outline

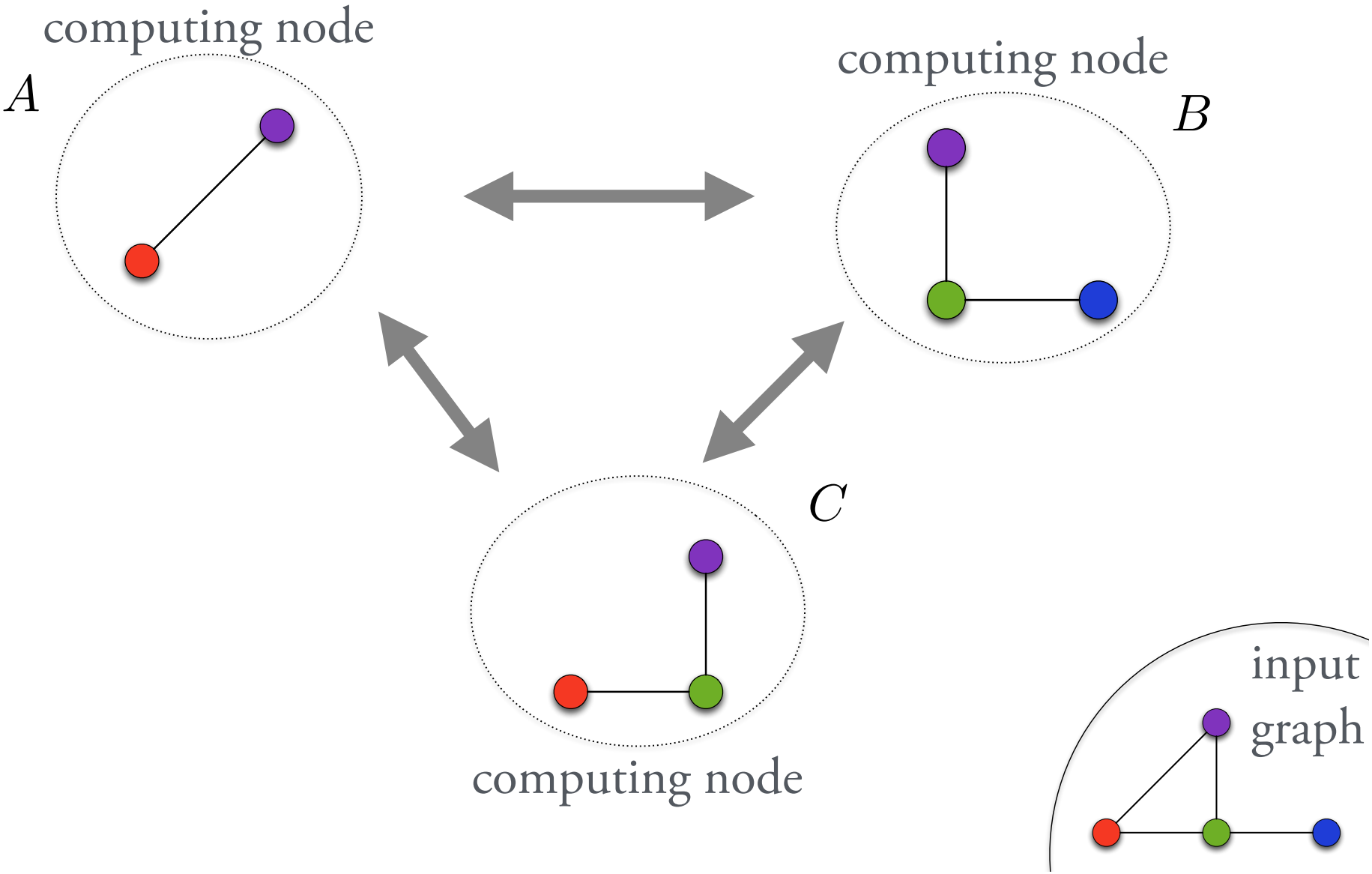
Example

Query: select all triangles



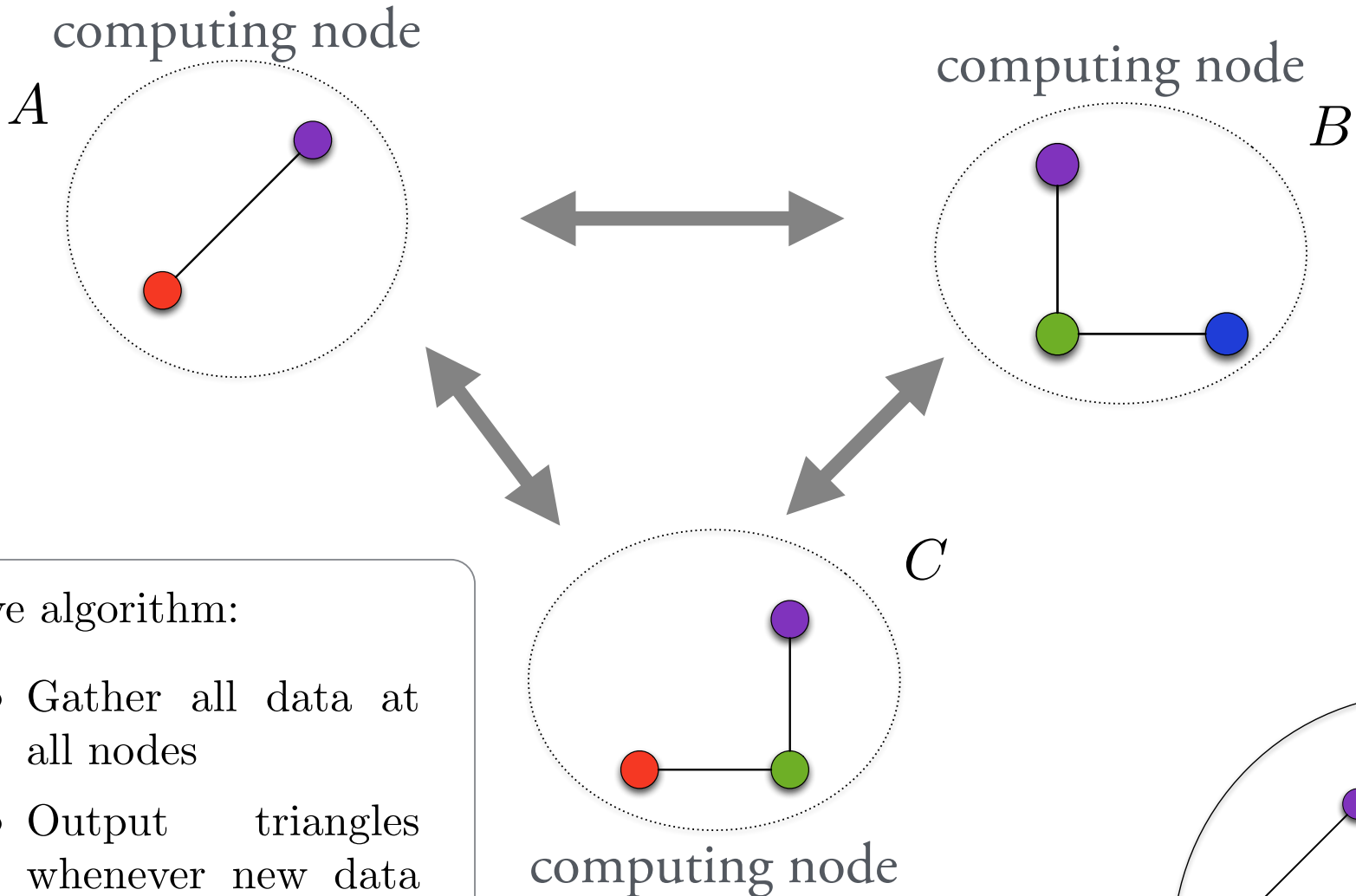
Example

Query: select all triangles



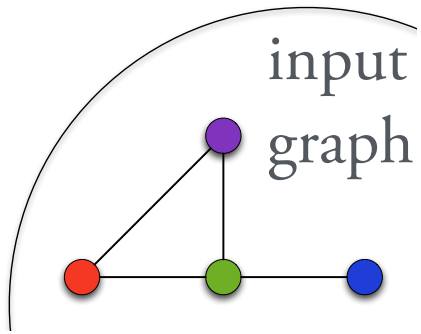
Example

Query: select all triangles



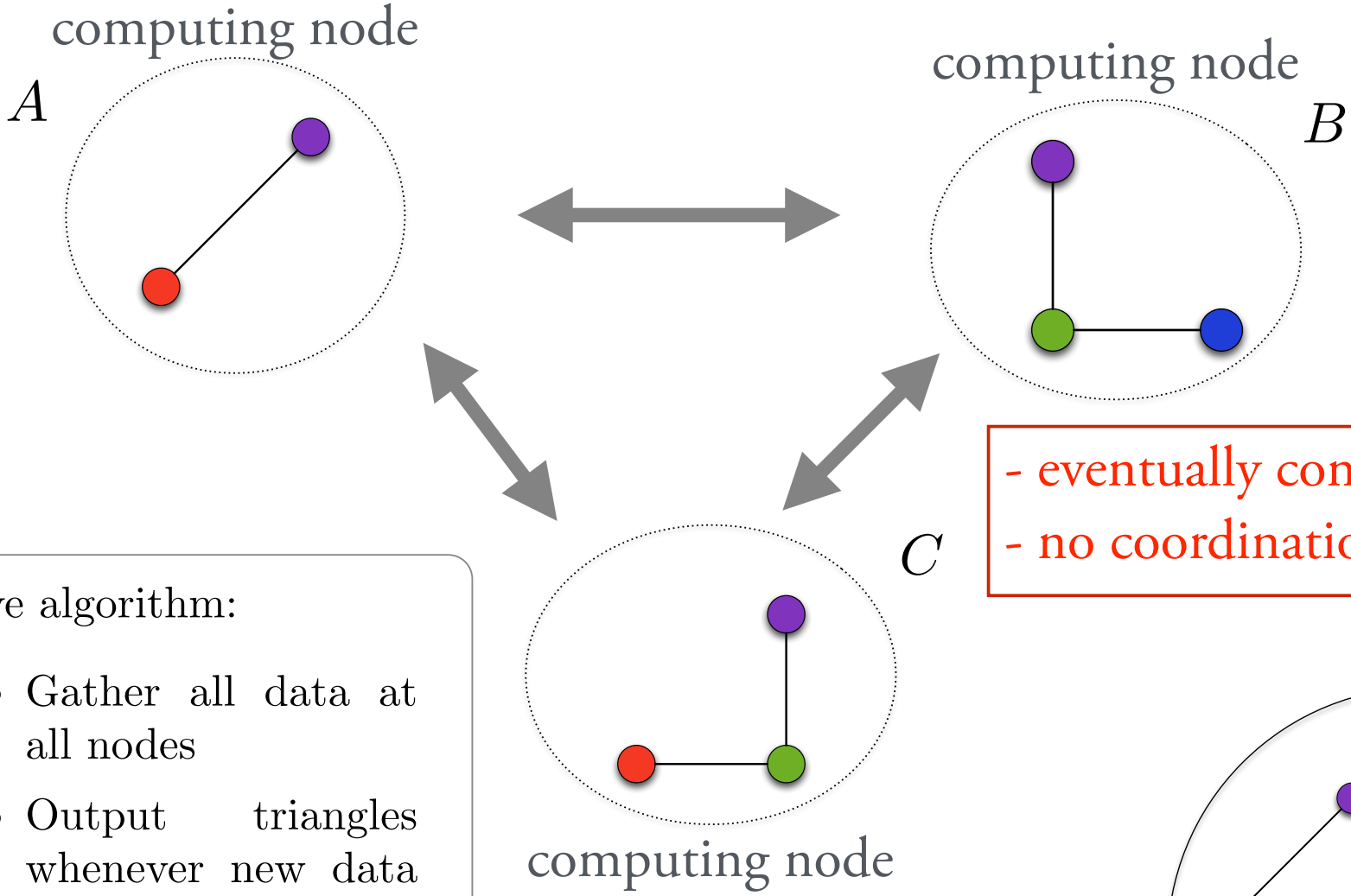
Naive algorithm:

- Gather all data at all nodes
- Output triangles whenever new data arrives



Example

Query: select all triangles

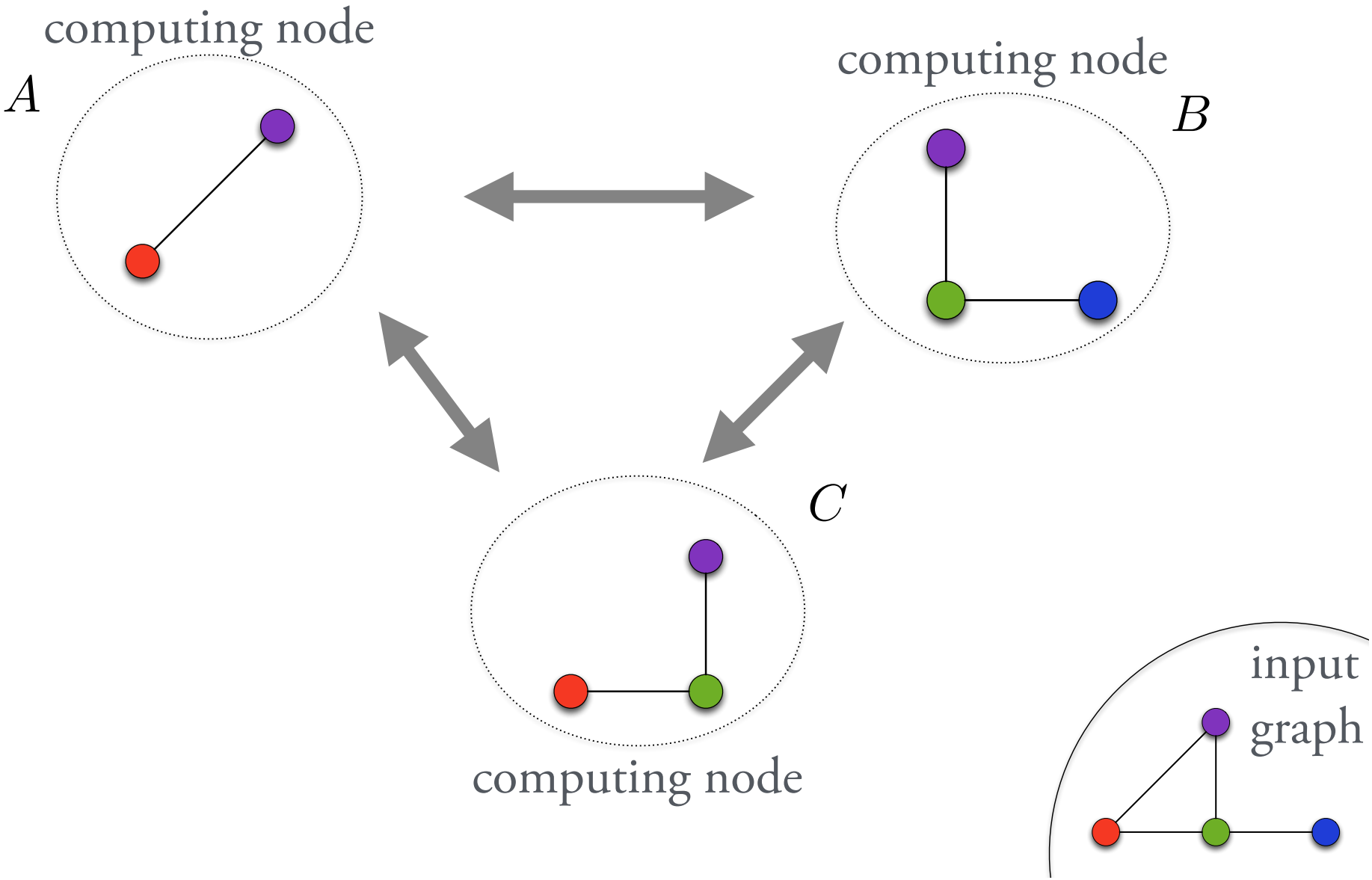


Naive algorithm:

- Gather all data at all nodes
- Output triangles whenever new data arrives

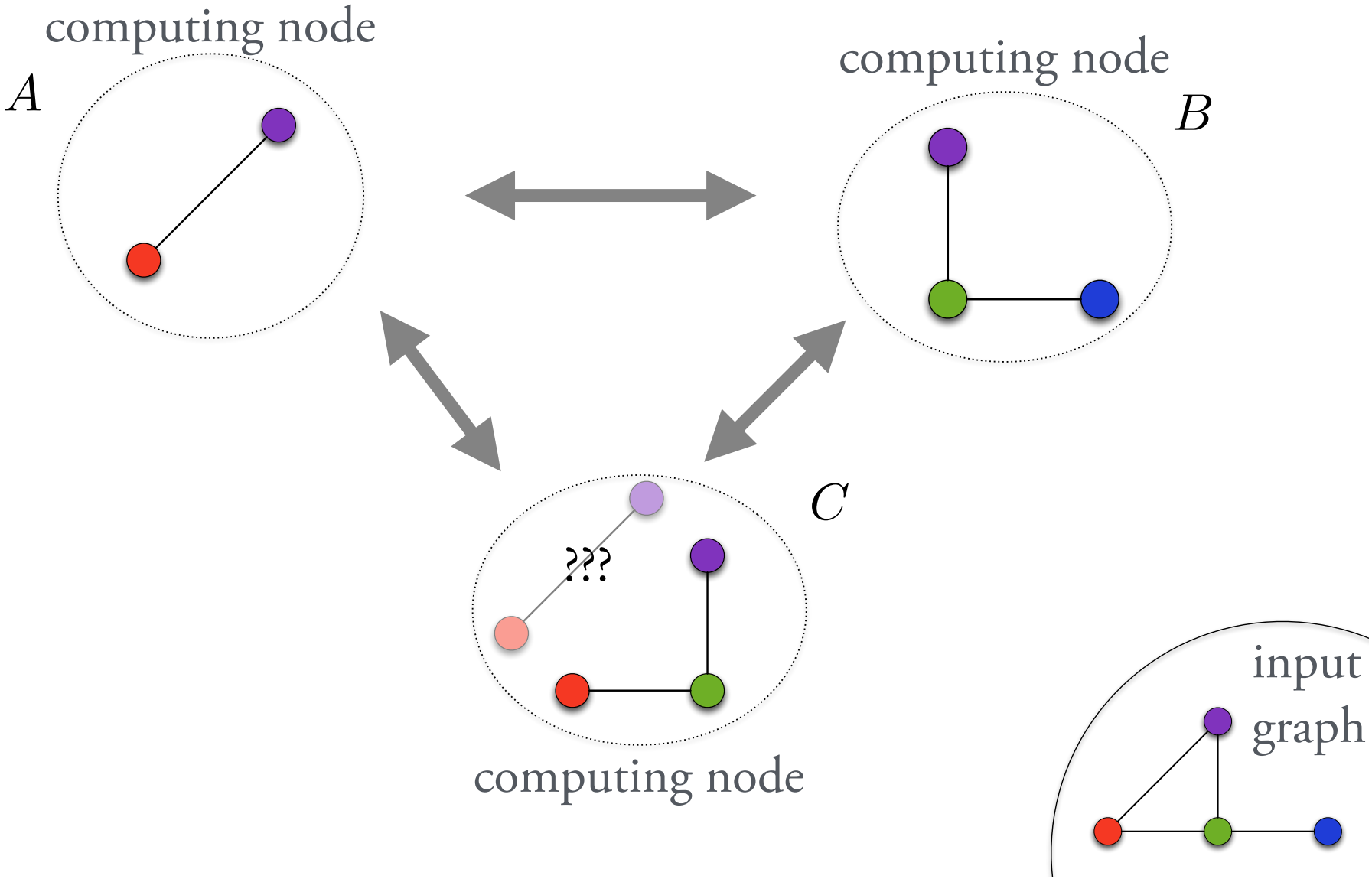
Example

Query: select all *open* triangles



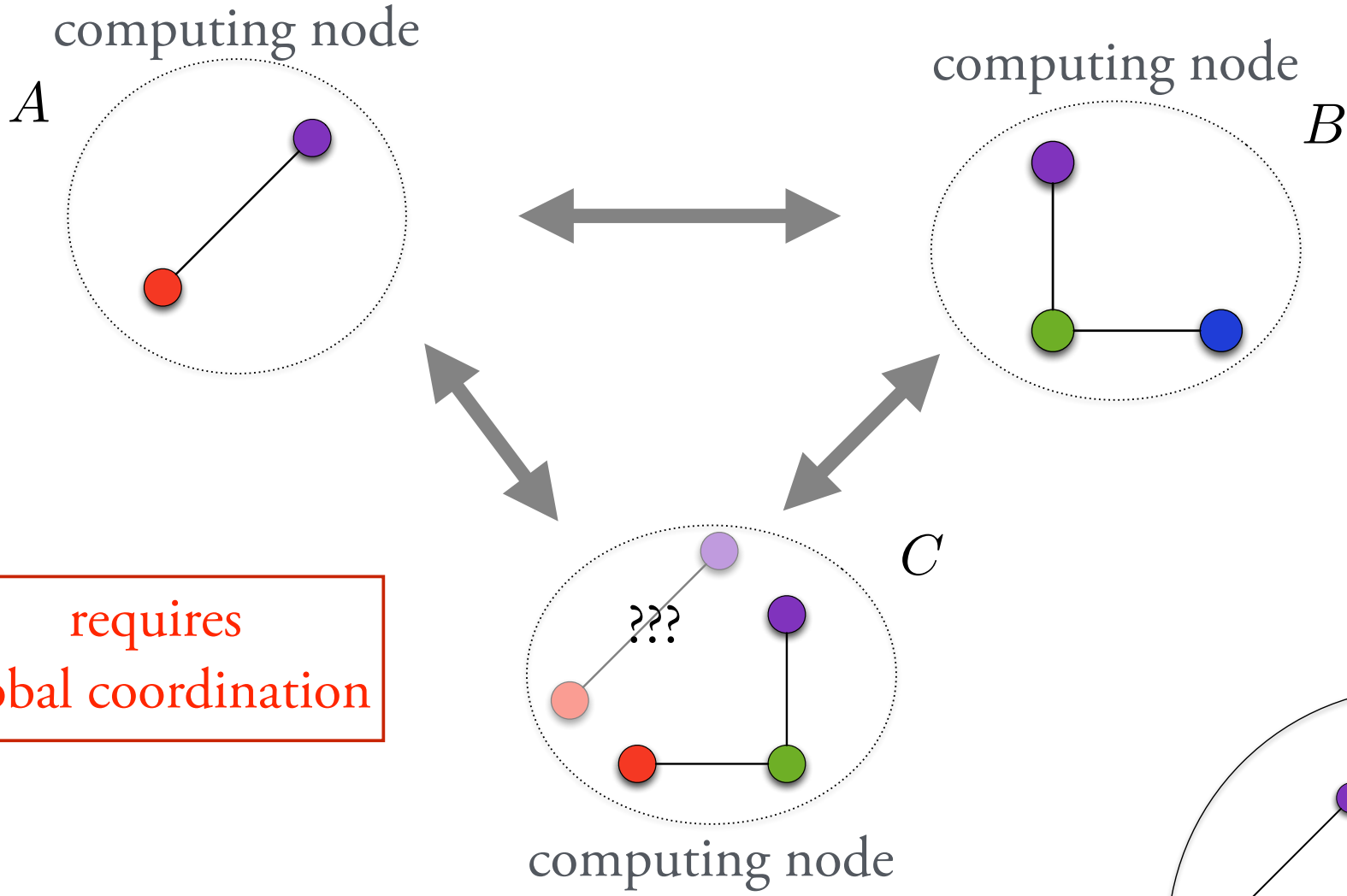
Example

Query: select all *open* triangles



Example

Query: select all *open* triangles



CALM: Consistency And Logical Monotonicity

CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and
eventually consistent execution strategy
iff
the query is monotone

CALM: Consistency And Logical Monotonicity

CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and
eventually consistent execution strategy
iff
the query is monotone

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)

CALM: Consistency And Logical Monotonicity

CALM-conjecture [Hellerstein, PODS 2010]

A query has a coordination-free and
eventually consistent execution strategy
iff
the query is monotone

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)
- [Zinn, Green, Ludäscher, ICDT 2012]: FALSE when nodes have more knowledge on how data is distributed (through distribution policies)

CALM: Consistency And Logical Monotonicity

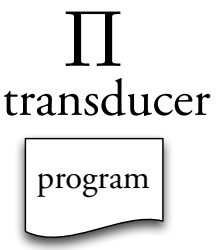
CALM-conjecture [Hellerstein, PODS 2010]

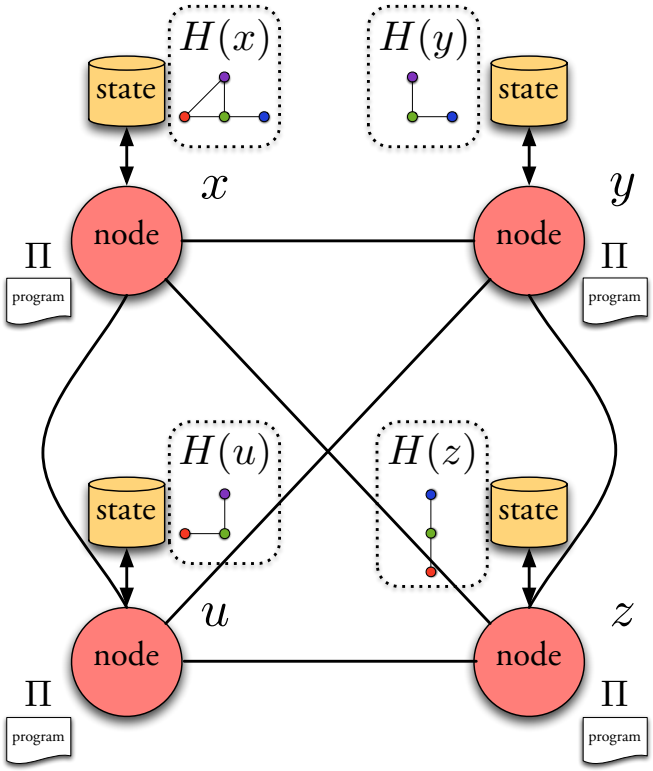
A query has a coordination-free and
eventually consistent execution strategy
iff
the query is monotone

- [Ameloot, N., Van den Bussche, PODS 2011]: TRUE (relational transducer networks)
- [Zinn, Green, Ludäscher, ICDT 2012]: FALSE when nodes have more knowledge on how data is distributed (through distribution policies)
- [Ameloot, Ketsman, N., Zinn, PODS 2014]: TRUE when also refining monotonicity

Outline

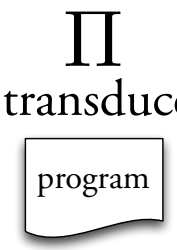
Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]

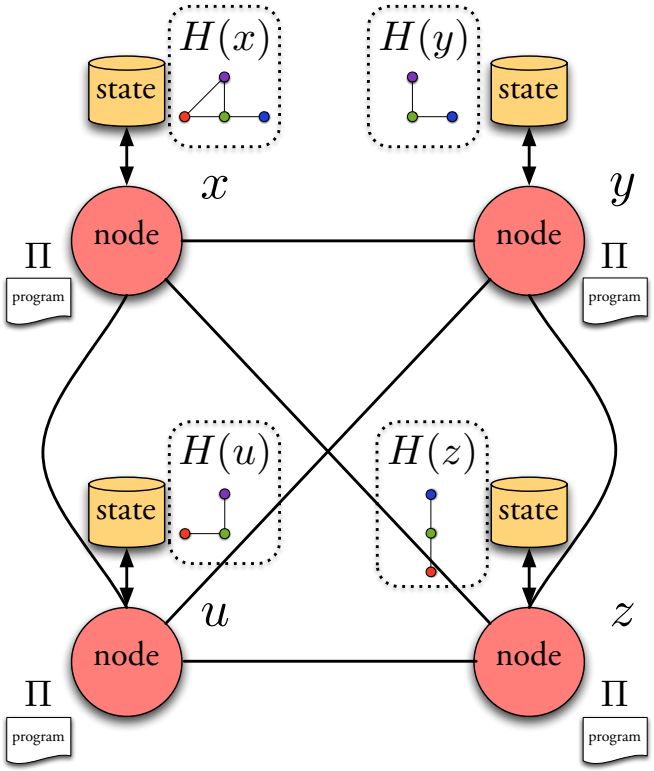




- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]

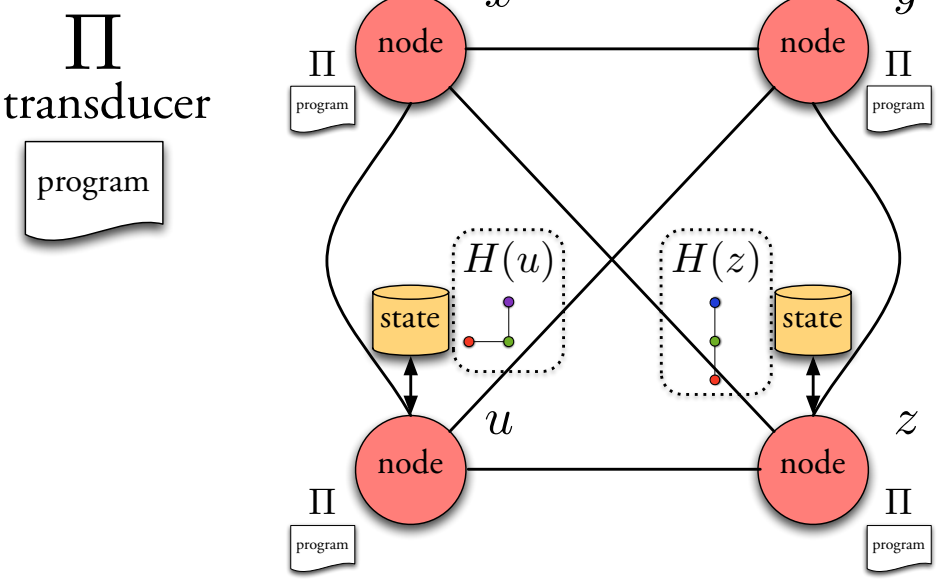
Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]





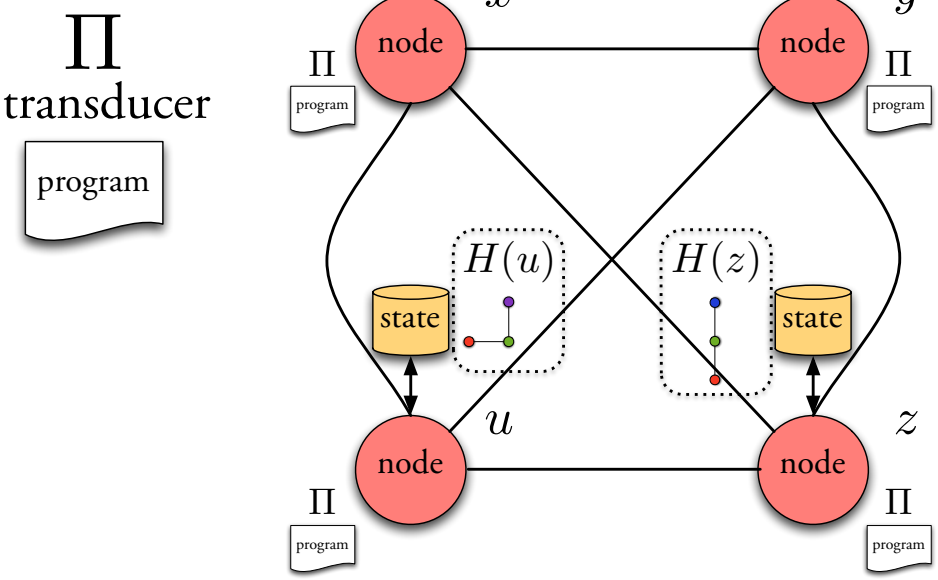
- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC

Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]



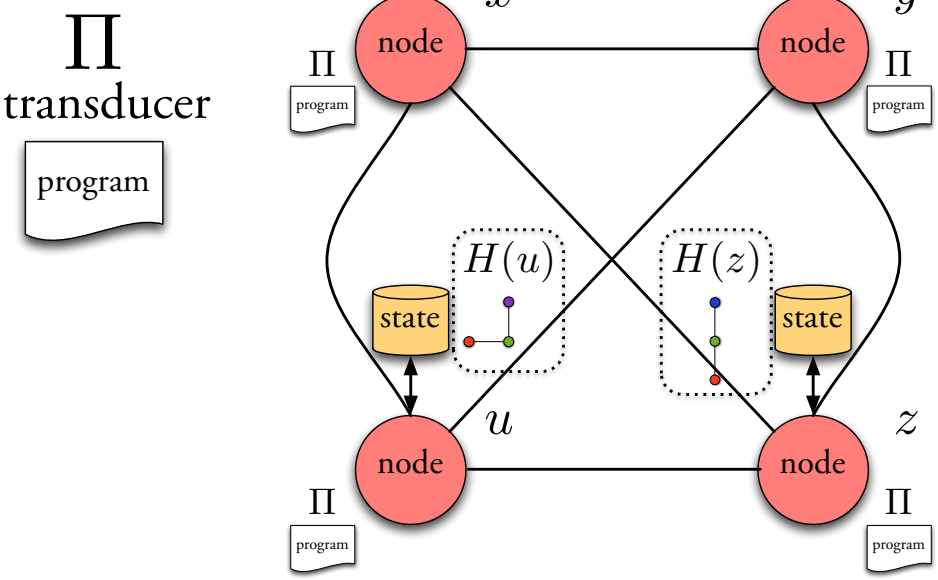
- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution H (can be modelled by a distribution policy $\mathbf{P}_H$)

Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]



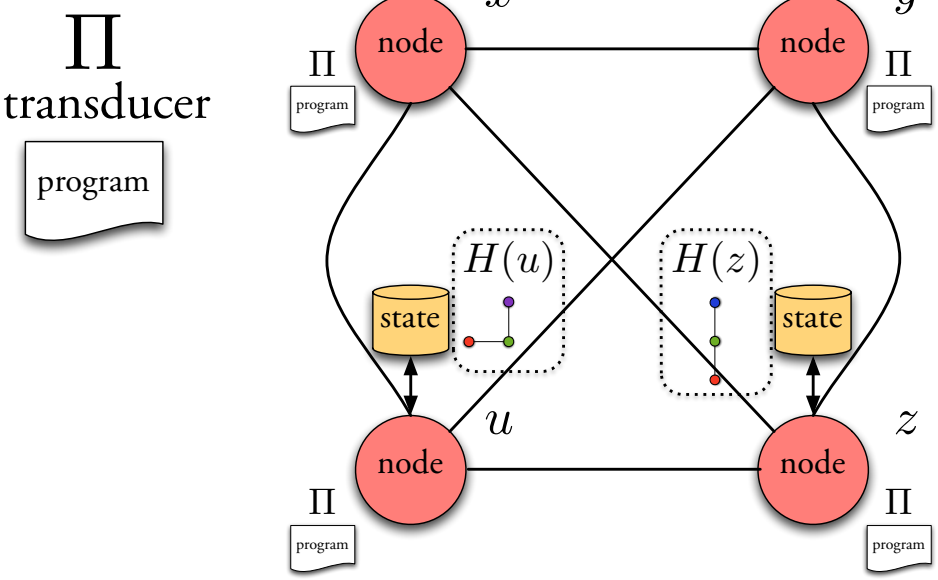
- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution H (can be modelled by a distribution policy $\mathbf{P}_H$)
  - every node runs same program Π

Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]



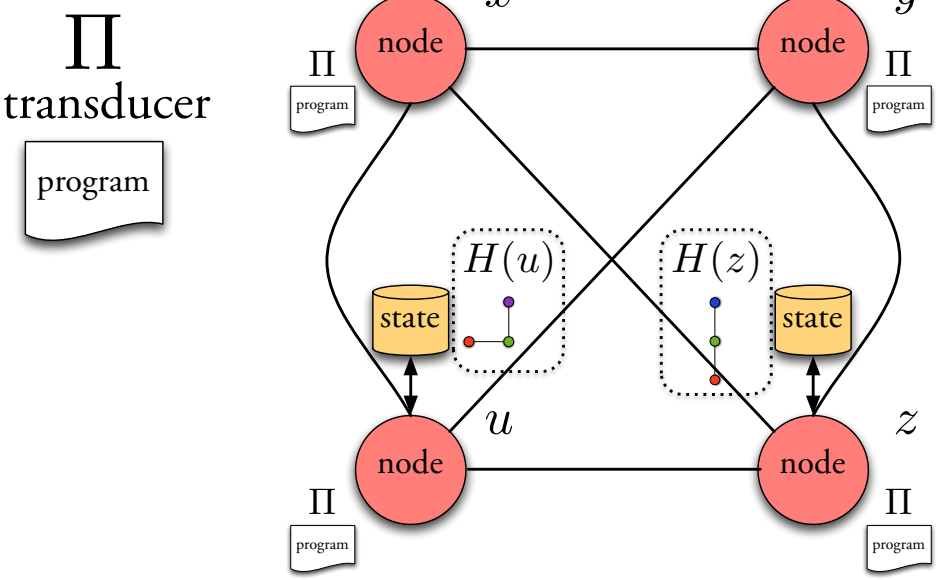
- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution H (can be modelled by a distribution policy $\mathbf{P}_H$)
  - every node runs same program Π
  - no rounds, nodes can send messages to each other (to further reshuffle data)

Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]



- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution H (can be modelled by a distribution policy $\mathbf{P}_H$)
  - every node runs same program Π
  - no rounds, nodes can send messages to each other (to further reshuffle data)
  - message can never get lost, but can take arbitrarily long to arrive

Computation model: relational transducer networks [Ameloot, N., Van den Bussche, 2011]



- based on relational transducers [Abiteboul, Vianu, Fordham, Yesha, PODS 1998]
- differences with MPC
  - initial start distribution H (can be modelled by a distribution policy $\mathbf{P}_H$)
  - every node runs same program Π
  - no rounds, nodes can send messages to each other (to further reshuffle data)
  - message can never get lost, but can take arbitrarily long to arrive
  - computes queries: inflationary and eventually consistent semantics

Computing all queries

Observation

Relational transducer networks compute every query

Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.

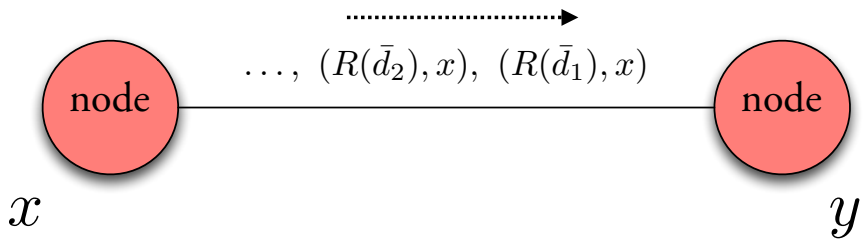
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.



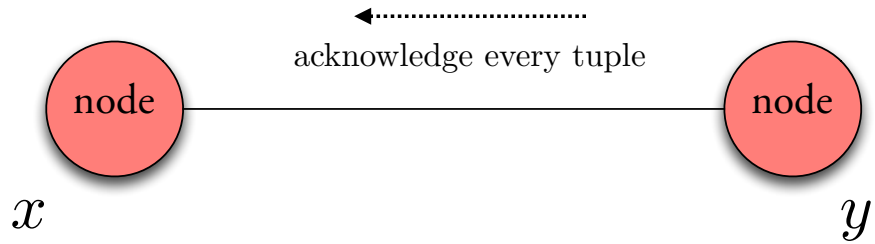
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.



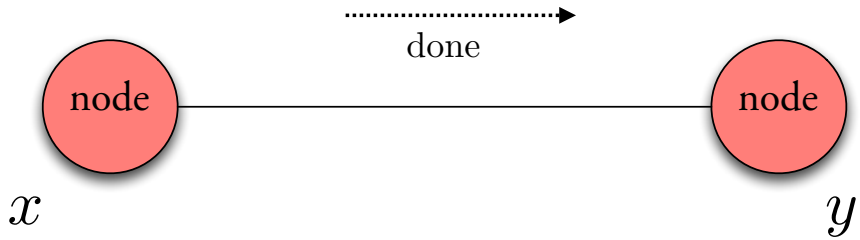
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.



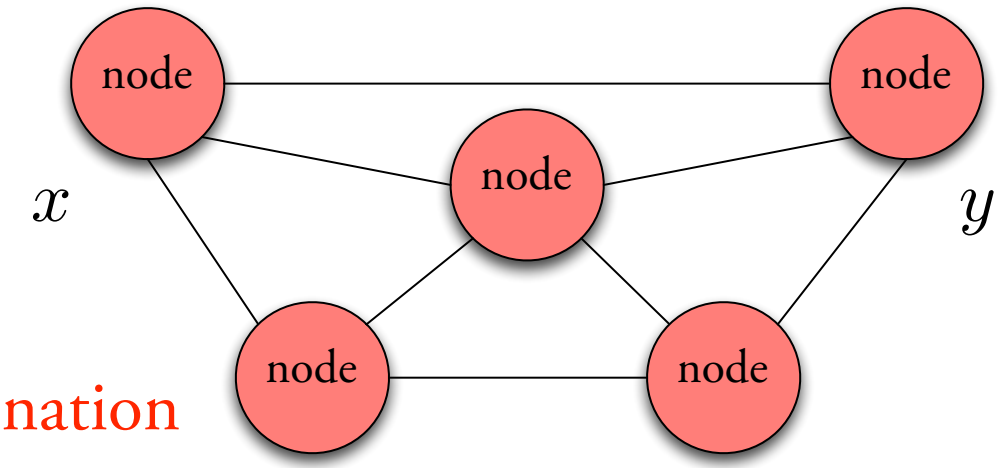
Computing all queries

Observation

Relational transducer networks compute every query

Idea.

- Broadcast all data. Needs coordination (messages can be delayed).
- Locally evaluate query.



global coordination

need to know every node in the network

Coordination: intuition

Coordination

- Gobal consensus
- Reduces parallellism

Coordination: intuition

Coordination

- Gobal consensus
- Reduces parallellism

Coordination-free

- Embarrassingly parallel execution
- Need communication to transfer data

Coordination: intuition

Coordination

- Gobal consensus
- Reduces parallellism

Coordination-free

- Embarrassingly parallel execution
- Need communication to transfer data

Goal

Separate *data*-communication from *coordination*-communication

Coordination-freeness

Informal Definition

A relational transducer network Π is coordination-free if
for all input databases I
there is an *ideal* initial distribution $\mathbf{P}_{H,I}$
on which Π computes the query
without reading input messages.

Coordination-freeness

Informal Definition

A relational transducer network Π is coordination-free if
for all input databases I
there is an *ideal* initial distribution $\mathbf{P}_{H,I}$
on which Π computes the query
without reading input messages.

Intuition:

- On ideal distribution: communication but no coordination
- On non-ideal distribution: only communication for data transfer, not for coordination

Example of coordination-freeness: output all triangles

Transducer program

- Broadcast local edges
- Output a triangle when one can be derived

Coordination-free:

- Ideal distribution: whole database on one node
- Query is computed by only heartbeat transitions

Outline

CALM-Theorem (0)

Theorem (Ameloot, N., Van den Bussche, PODS 2011)

The following are equivalent. Query Q is

- *computable by a coordination-free relational transducer network;*
- *computable by an oblivious relational transducer network; and,*
- *monotone.*

oblivious: not knowing the identities of all other nodes in the network

CALM-Theorem (0)

Theorem (Ameloot, N., Van den Bussche, PODS 2011)

The following are equivalent. Query Q is

- *computable by a coordination-free relational transducer network;*
- *computable by an oblivious relational transducer network; and,*
- *monotone.*

oblivious: not knowing the identities of all other nodes in the network

Corollary:

- coordination-free computations can only define monotone queries

CALM-Theorem (0)

Theorem (Ameloot, N., Van den Bussche, PODS 2011)

The following are equivalent. Query Q is

- *computable by a coordination-free relational transducer network;*
- *computable by an oblivious relational transducer network; and,*
- *monotone.*

oblivious: not knowing the identities of all other nodes in the network

Corollary:

- coordination-free computations can only define monotone queries
- Complete language for coordination-free computation: language defining all monotone queries
  - e.g., Datalog with inequalities and value invention [Hull, Yoshikawa, 1990] [Cabibbo, 1998]

Weaker forms of monotonicity

Definition

A query Q is monotone if $Q(I) \subseteq Q(I \cup J)$ for all instances I and J.

Observation

If Q is monotone then $Q(I_s) \subseteq Q(I)$ for any $I_s \subseteq I$.

Weaker forms of monotonicity

Definition

A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Weaker forms of monotonicity

Definition

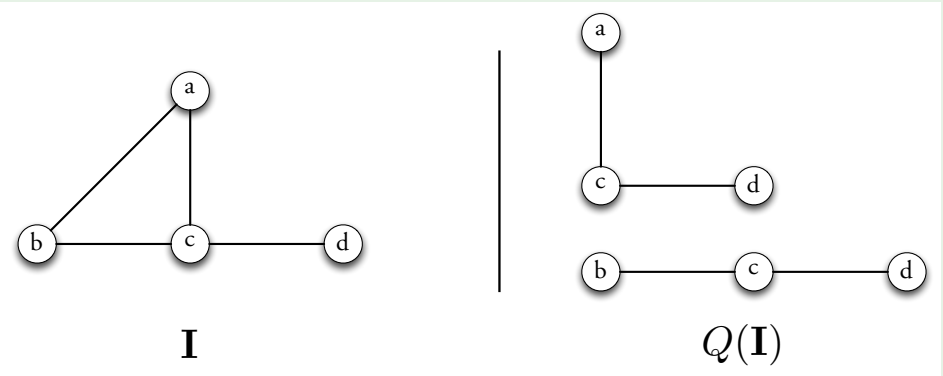
A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Example

Selecting open triangles in a graph is domain-distinct-monotone.



Weaker forms of monotonicity

Definition

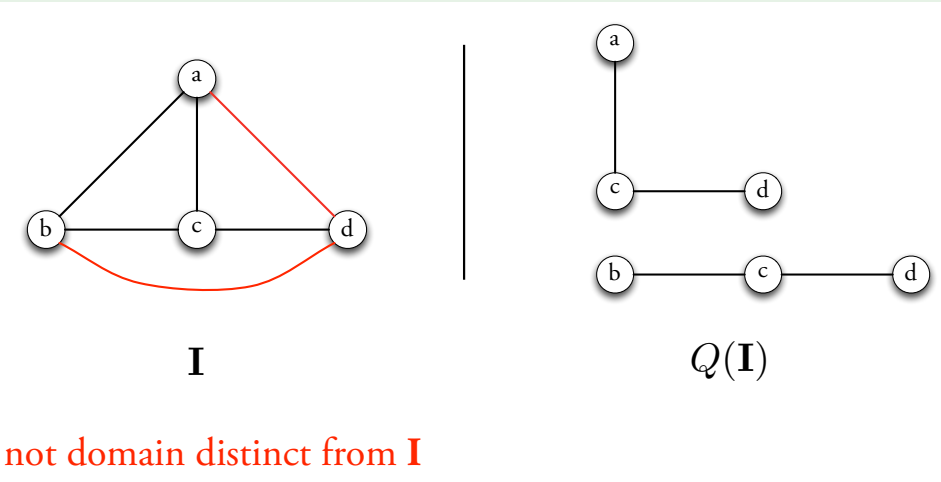
A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Example

Selecting open triangles in a graph is domain-distinct-monotone.



Weaker forms of monotonicity

Definition

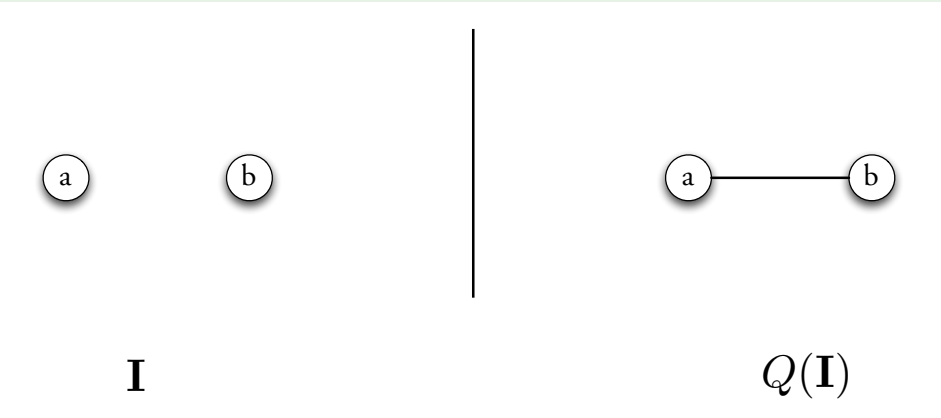
A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Example

Complement of the transitive closure of a graph is not domain-distinct-monotone.



Weaker forms of monotonicity

Definition

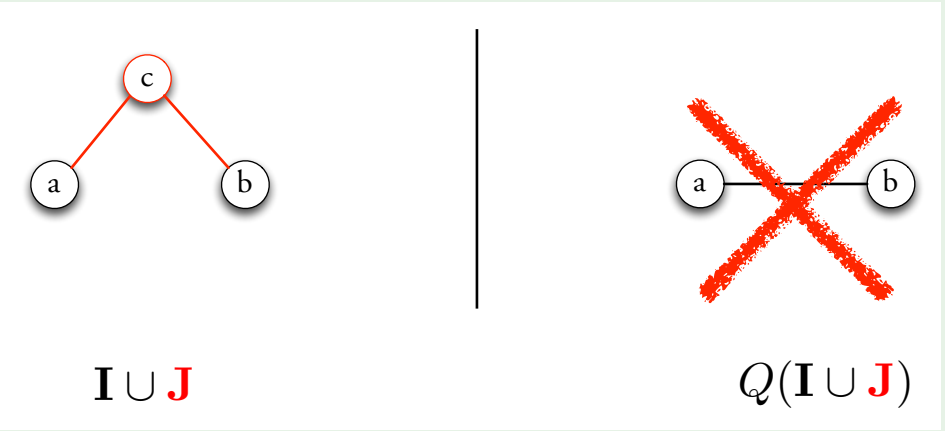
A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Example

Complement of the transitive closure of a graph is not domain-distinct-monotone.



Weaker forms of monotonicity

Definition

A query Q is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain distinct from I.

Observation (preserved under extensions)

When Q is domain-distinct-monotone, then

$$Q(I_{|C}) \subseteq Q(I)$$

for all subsets of the domain C.

Induced subinstance: $I_C = \{\mathbf{f} \in I \mid \text{adom}(\mathbf{f}) \subseteq C\}$

CALM-Theorem (1)

Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query Q is

1. *computable by a coordination-free relational transducer network that has access to the initial distribution;*
2. *domain-distinct-monotone.*

CALM-Theorem (1)

Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query Q is

1. *computable by a coordination-free relational transducer network that has access to the initial distribution;*
2. *domain-distinct-monotone.*

Crux (2)→(1): (very rough)

- every transducer has access to the initial distribution:
  - broadcast both occurring and missing facts
- transducer outputs $Q(J_{|C})$ for every set C on which they have complete knowledge

CALM-Theorem (1)

Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query Q is

1. *computable by a coordination-free relational transducer network that has access to the initial distribution;*
2. *domain-distinct-monotone.*

Corollary:

- Every language defining all domain-disjoint-monotone queries is a complete language for coordination-free computation
- Semi-positive datalog with value invention [Cabibbo, 1998]

Even weaker forms of monotonicity

Definition

A query Q is domain-disjoint-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain disjoint from I.

Even weaker forms of monotonicity

Definition

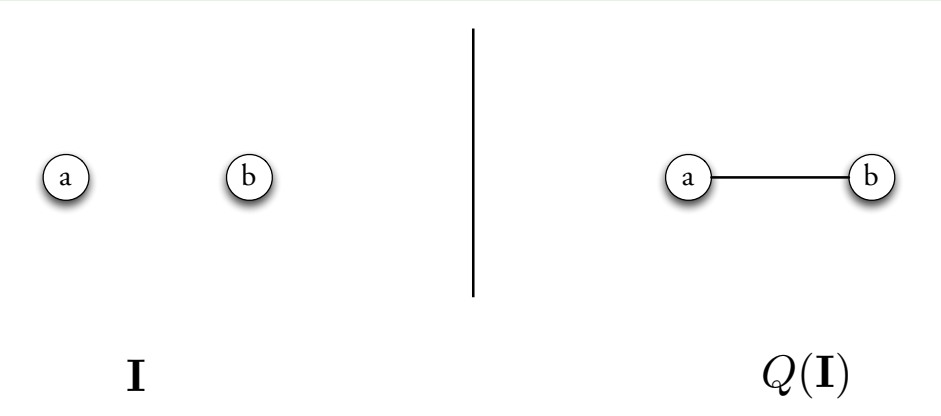
A query Q is domain-disjoint-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain disjoint from I.

Example

Complement of the transitive closure of a graph is domain-disjoint-monotone.



Even weaker forms of monotonicity

Definition

A query Q is domain-disjoint-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances I and J for which J is domain disjoint from I.

Observation (preserved under disjoint extension)

When Q is domain-disjoint-monotone, then

$$Q(C) \subseteq Q(I)$$

for all instances I and connected components C of I.

CALM-Theorem (2)

Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query Q is

1. *computable by a coordination-free relational transducer network under domain-guided distribution policies; and,*
2. *domain-disjoint-monotone.*

CALM-Theorem (2)

Theorem (Ameloot, Ketsman, N., Zinn, PODS 2014)

The following are equivalent. Query Q is

1. *computable by a coordination-free relational transducer network under domain-guided distribution policies; and,*
2. *domain-disjoint-monotone.*

Corollary:

- Every language defining all domain-disjoint-monotone queries is a complete language for coordination-free computation
- Semi-connected Datalog with value invention [Ameloot, Ketsman, N., Zinn, PODS 2014]

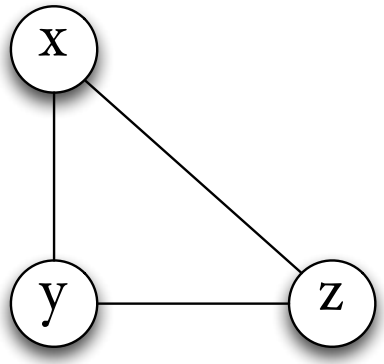
Connected Datalog rule

Definition

A Datalog¬ rule is connected when the graph formed by the positive atoms is connected.

Example

$\varphi \equiv O(x, y, z) \leftarrow E(x, y), E(y, z), E(z, x)$ is connected

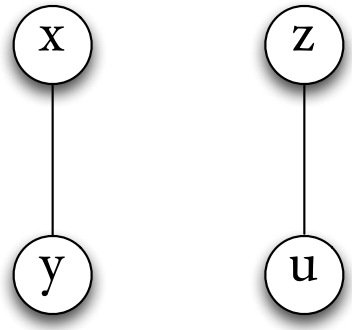


$graph^{+}(\varphi)$

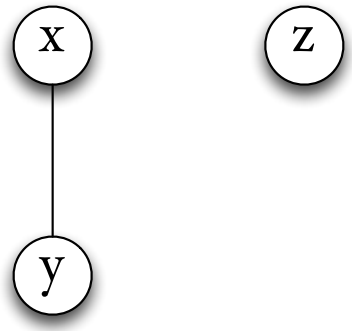
Connected Datalog rule

Example

$O(x, y, z) \leftarrow E(x, y), E(z, u)$ is not connected



$O(x, y, z) \leftarrow E(x, y), \neg E(y, z)$ is not connected



Semi-connected Datalog

Definition

A Datalog$^\neg$ program is semi-connected if all rules are connected except (possibly) those of the last stratum.

Example

$$\begin{array}{lcl} TC(x,y) & \leftarrow & E(x,y) \\ TC(x,y) & \leftarrow & E(x,z), TC(z,y) \\ \hline O(x,y) & \leftarrow & \neg TC(x,y), x \neq y \end{array}$$

- Semi-positive Datalog is strictly included in semi-connected Datalog$^\neg$
- Semi-connected Datalog$^\neg$ defines only domain-distinct-monotone queries

Connected Datalog

Win-move: $\mathsf{win}(x) \leftarrow \mathsf{move}(x, y), \neg\mathsf{win}(y)$

Connected Datalog

Win-move: $\mathrm{win}(x) \leftarrow \mathrm{move}(x, y), \neg\mathrm{win}(y)$

Theorem (Zinn, Green, Ludäscher, ICDT 2012)

The query computing the won positions of the win-move game is in $\mathcal{F}_2$

Connected Datalog

Win-move: $\text{win}(x) \leftarrow \text{move}(x, y), \neg\text{win}(y)$

Theorem (Zinn, Green, Ludäscher, ICDT 2012)

The query computing the won positions of the win-move game is in $\mathcal{F}_2$

Proof. Simulation in semi-monotone fragment of Datalog$_\forall^{\neg\neg}$ with a novel disorderly semantics [Abiteboul, Vianu, 1991].

Connected Datalog

Win-move: $\text{win}(x) \leftarrow \text{move}(x, y), \neg\text{win}(y)$

Theorem (Zinn, Green, Ludäscher, ICDT 2012)

The query computing the won positions of the win-move game is in $\mathcal{F}_2$

Proof. Simulation in semi-monotone fragment of Datalog$^{\neg}_{\forall}$ with a novel disorderly semantics [Abiteboul, Vianu, 1991].

Theorem (Ameloot, Ketsman, N., Zinn, ICDT 2015)

Queries expressed in connected Datalog under the well-founded semantics are in $\mathcal{F}_2$.

CALM overview

$$
\begin{array}{ccccccccc}
\text{Datalog}(\neq) & \subsetneq & \text{wILOG}(\neq) & = & \mathcal{M} & = & \mathcal{F}_0 & = & \mathcal{A}_0 \\
\not\cap & & \not\cap & & \not\cap & & \not\cap & & \not\cap \\
\text{SP-Datalog} & \subsetneq & \text{SP-wILOG} & = & \mathcal{M}_{distinct} & = & \mathcal{F}_1 & = & \mathcal{A}_1 \\
\not\cap & & \not\cap & & \not\cap & & \not\cap & & \not\cap \\
\text{semicon-Datalog}^{\neg} & \subsetneq & \text{semicon-wILOG}^{\neg} & = & \mathcal{M}_{disjoint} & = & \mathcal{F}_2 & = & \mathcal{A}_2
\end{array}
$$

Intermediate wrap-up

Summary:

- CALM-theorems link Datalog, monotonicity, and coordination-freeness on different levels
- Connected Datalog

Intermediate wrap-up

Summary:

- CALM-theorems link Datalog, monotonicity, and coordination-freeness on different levels
- Connected Datalog

Research issues:

- Broadcast algorithms are not efficient (initial steps in [Ketsman, N., ICDT 2015])
- Quantify coordination [Alvaro et al., CIDR 2011, ICDE 2014]
- General theory of asynchronous query evaluation to capture efficient evaluation

Outline

Discussion

- Overview
  - MPC & Shares/HyperCube
  - Parallel-correctness and transferability




  - Coordination-freeness and degrees of monotonicity




- Fascinating research area:
  - exciting new machinery
    - e.g., optimal fractional node/edge coverings
  - old wine in new bottles
    - e.g., decision problems relating to CQs
    - e.g., Datalog and monotonicity

Thank you for your attention

Questions?