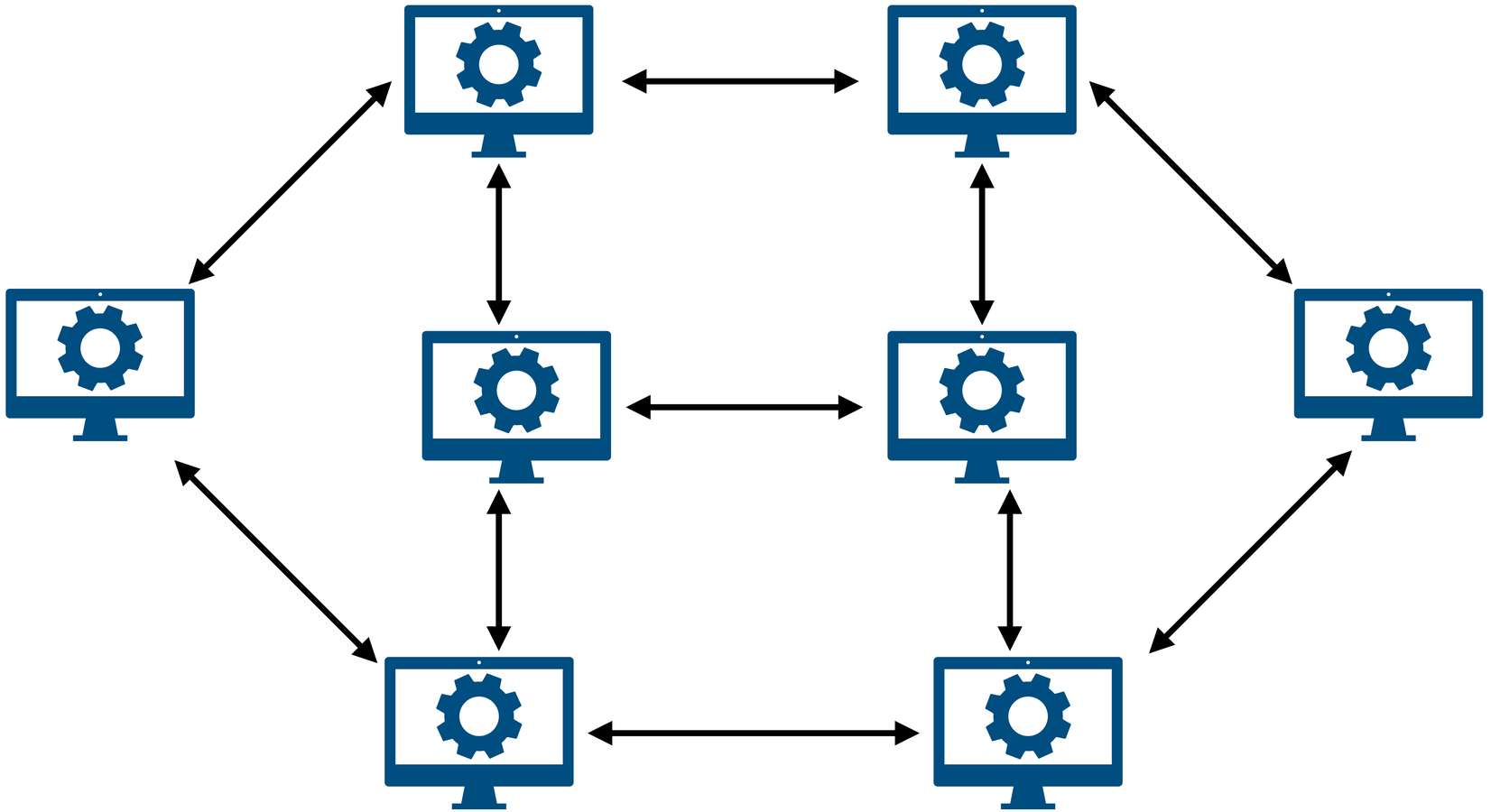
Optimizing Stratified Datalog with Count

Fernando Hechavarría Fajardo, Heba Aamer, and Bas Ketsman

Vrije Universiteit Brussel, Belgium

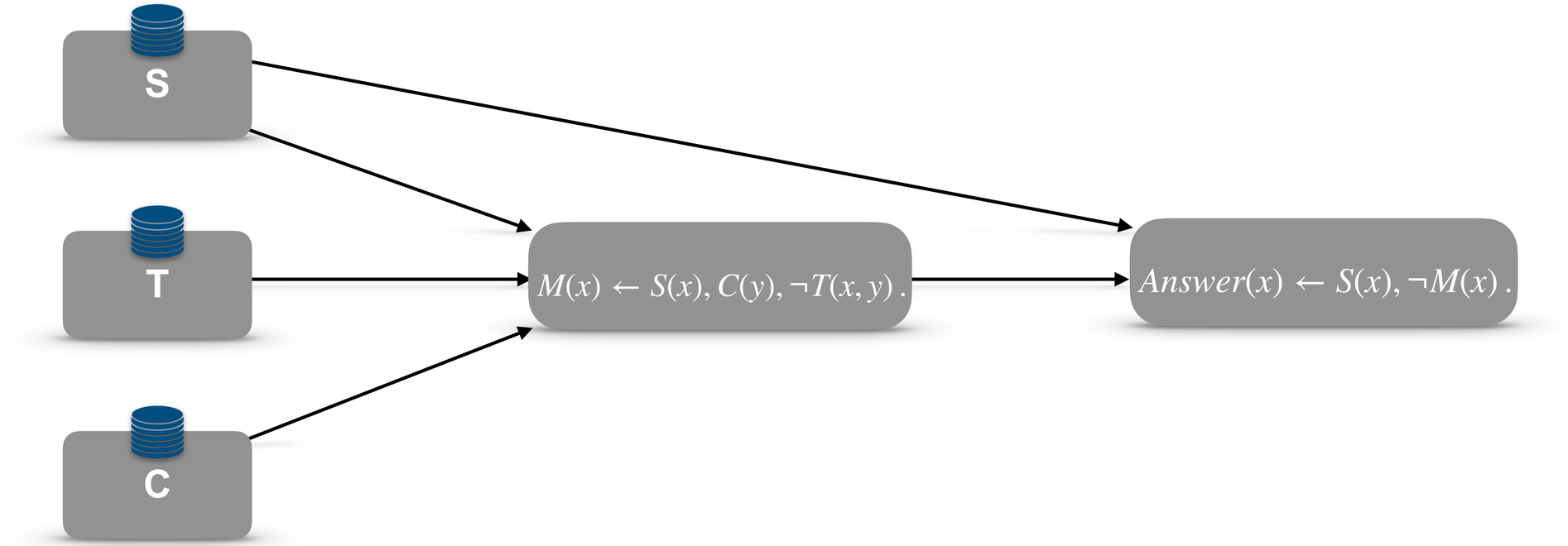
22 November 2024

Distributed Systems

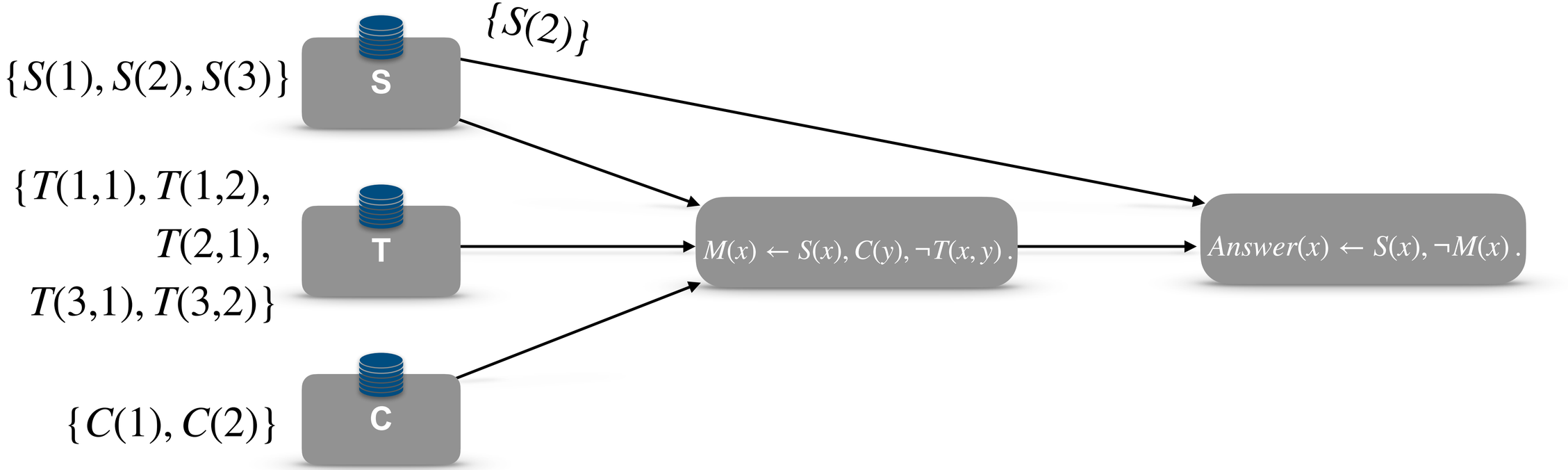


Distributed Datalog Programs

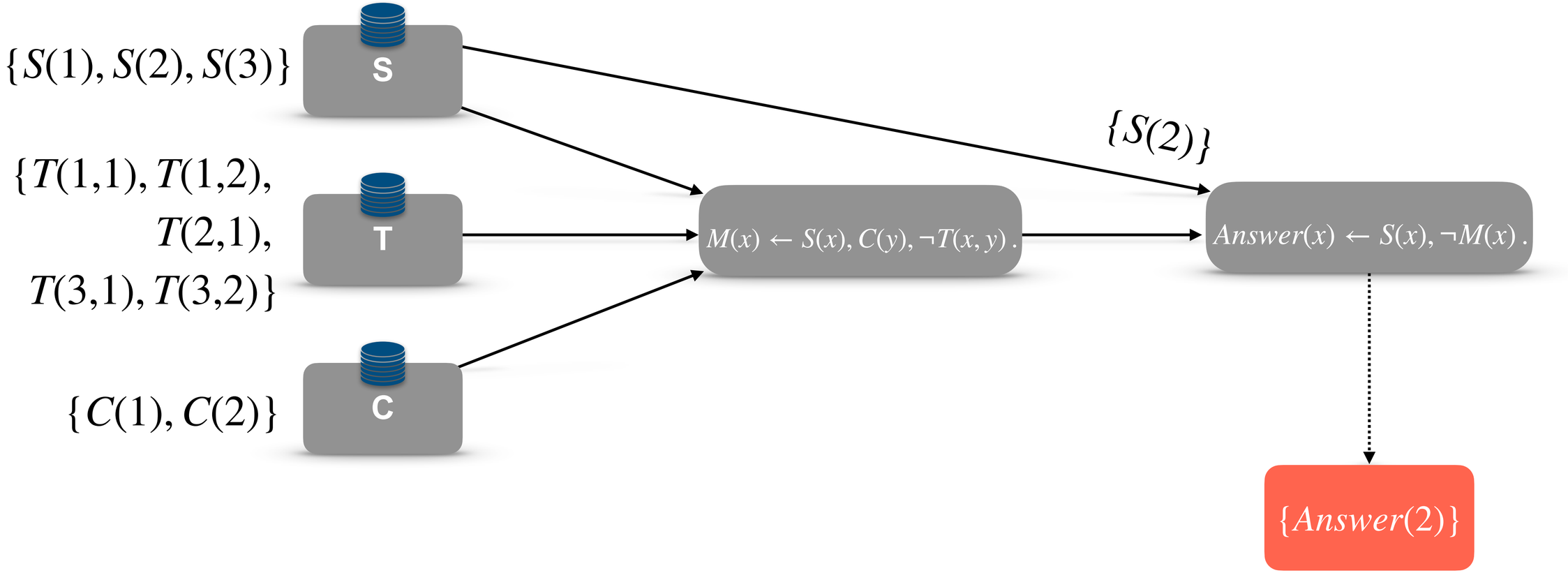
Find all *students (S) taking (T) all courses (C)*.



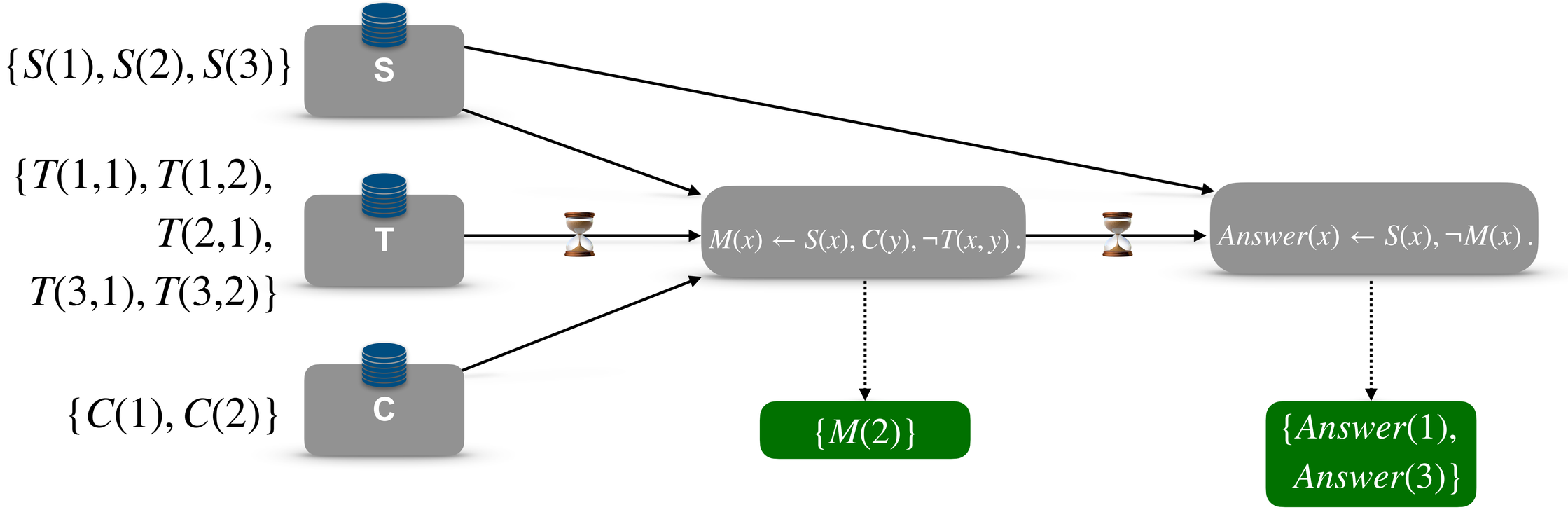
Distributed Datalog Programs



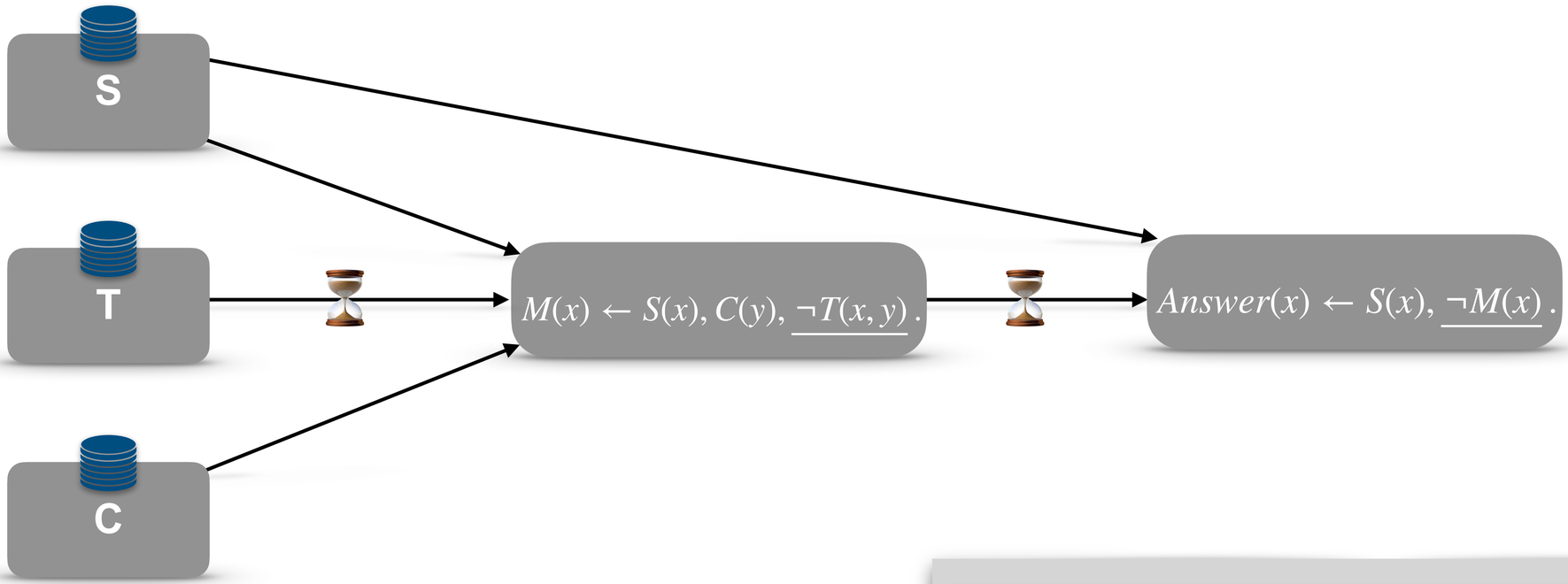
Distributed Datalog Programs



Distributed Datalog Programs



Synchronization ≈ Strata



$$\frac{M(x) \leftarrow S(x), C(y), \neg T(x,y)\,.}{Answer(x) \leftarrow S(x), \neg M(x)\,.}$$

Goal

How can we reduce synchronization overhead?

Positive Programs

- Positive Datalog programs are confluent.
- Positive Datalog programs can only express monotone queries.

Confluence is undecidable for Datalog programs with negation.

Negation → Incremental Count

$$M(x) \leftarrow S(x), C(y), \neg T(x, y).$$

$$Answer(x) \leftarrow S(x), \neg M(x).$$

$$T_{\#}(x; count\langle\rangle) \leftarrow C(y), T(x, y).$$

$$M(x) \leftarrow S(x), C(y), T_{\#}(x; c), c \leq 0.$$

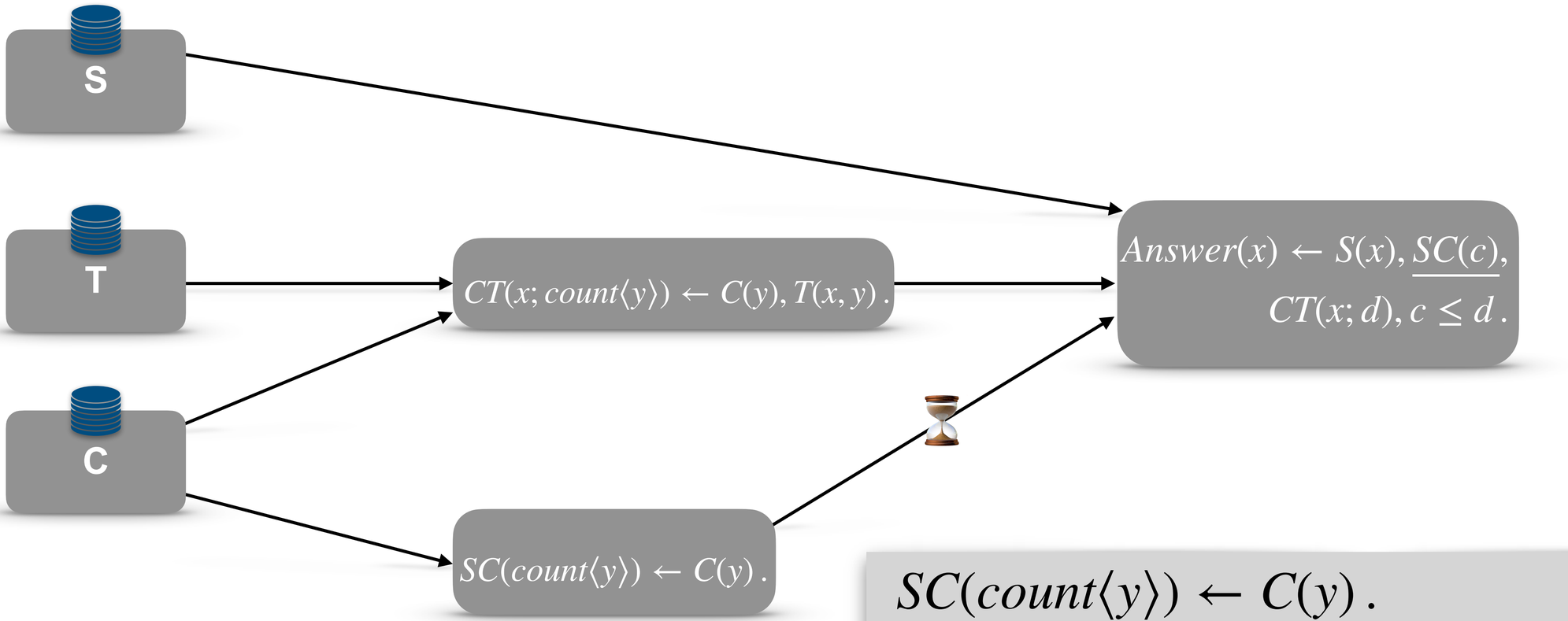
$$M_{\#}(x; count\langle\rangle) \leftarrow M(x).$$

$$Answer(x) \leftarrow S(x), M_{\#}(x; c), c \leq 0.$$

Replacing Negation by Count

- The rewriting does not lead directly to a program with a reduced number of synchronization steps.
- Even Positive Datalog programs with count are not always confluent.

Fewer synchronization steps



$$SC(count\langle y\rangle) \leftarrow C(y)\,.$$

$$CT(x; count\langle y\rangle) \leftarrow C(y), T(x, y)\,.$$
$$Answer(x) \leftarrow S(x), SC(c), CT(x; d), c \leq d\,.$$

Local Optimization through Counting

$$M(x) \leftarrow S(x), C(y), \neg T(x, y)\,.$$

$$Answer(x) \leftarrow S(x), \neg M(x)\,.$$

$$SC(count\langle y \rangle) \leftarrow C(y)\,.$$

$$CT(x; count\langle y \rangle) \leftarrow C(y), T(x, y)\,.$$

$$Answer(x) \leftarrow S(x), SC(c), CT(x; d), c \leq d\,.$$

Local Optimization through Counting

In a program with precisely one QF-generating rule of the form

$$\mathsf{QF}(\boldsymbol{y};) \leftarrow_{\{\mathsf{T}\}} \mathsf{R}(\boldsymbol{x};), \neg\mathsf{T}(\boldsymbol{z};). \tag{1}$$

with $\emptyset \subsetneq \boldsymbol{y} \subsetneq \boldsymbol{x} \cap \boldsymbol{z}$, every rule of the form

$$\mathsf{F}(\boldsymbol{u};) \leftarrow_{\{\mathsf{QF}\}} \mathsf{S}(\boldsymbol{v};), \neg\mathsf{QF}(\boldsymbol{w};). \tag{2}$$

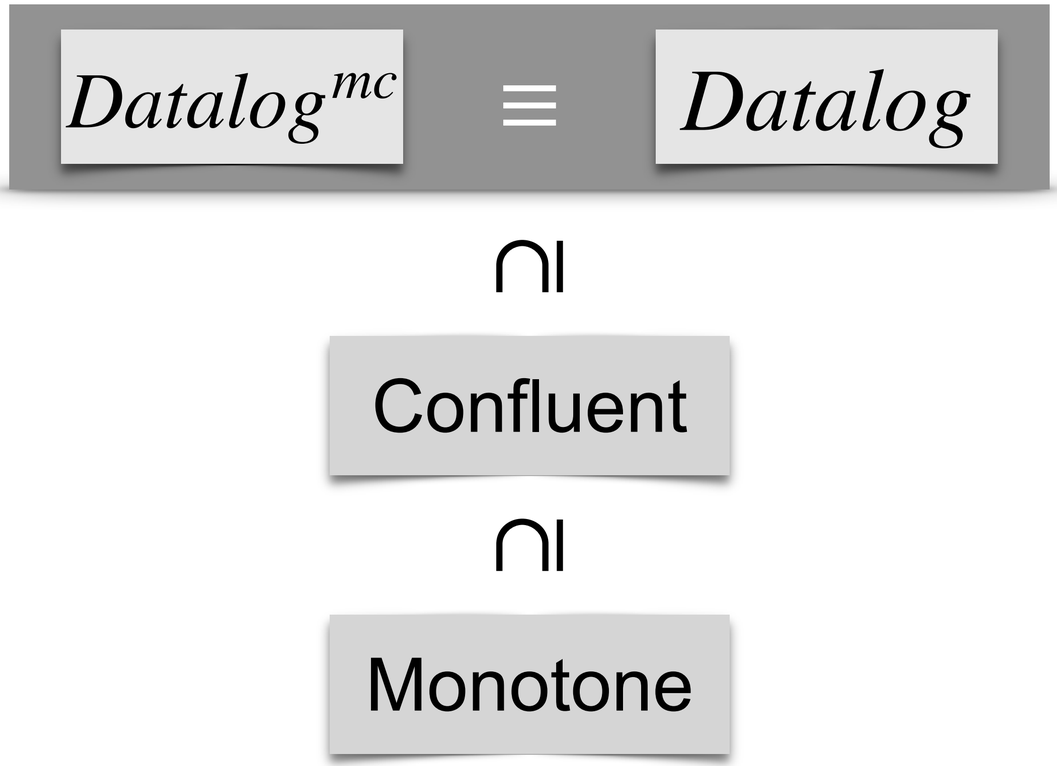
can be replaced by the rules

$$\begin{aligned}
\mathsf{Count}_\top(\boldsymbol{y}; 0) &\leftarrow \mathsf{Adom}(\boldsymbol{y}_1), \ldots, \mathsf{Adom}(\boldsymbol{y}_k).\\
\mathsf{Count}_\top(\boldsymbol{y}; \mathsf{count}\langle \boldsymbol{x} \cap \boldsymbol{z} \rangle) &\leftarrow \mathsf{R}(\boldsymbol{x};).\\
\mathsf{Count}_\leq(\boldsymbol{y}; 0) &\leftarrow \mathsf{Adom}(\boldsymbol{y}_1), \ldots, \mathsf{Adom}(\boldsymbol{y}_k).\\
\mathsf{Count}_\leq(\boldsymbol{y}; \mathsf{count}\langle \boldsymbol{x} \cap \boldsymbol{z} \rangle) &\leftarrow \mathsf{R}(\boldsymbol{y};), \mathsf{T}(\boldsymbol{z};).\\
\mathsf{F}(\boldsymbol{u};) &\leftarrow_{\{\mathsf{Count}_\top\}} \mathsf{S}(\boldsymbol{v};), \mathsf{Count}_\top(\boldsymbol{w}; c), \mathsf{Count}_\leq(\boldsymbol{w}; d), c \leq d.
\end{aligned}$$

Additionally, rule (1) can be removed if rule (2) is the only rule using QF. Note that we assume that relation names $\mathsf{Count}_\top$ and $\mathsf{Count}_\leq$ do not exist in the original program.

Results

We provide a rewriting technique that can reduce the ***required*** synchronization up to half.



Future Work

- Broader identification of confluent programs.
- Use of database-dependent constants to optimize further.
- Experimental validation to prove whether the rewriting technique improves performance in practice.