On Extending a Full-Sharing Multithreaded Tabling Design with Batched Scheduling

Miguel Areias and Ricardo Rocha
CRACS & INESC-TEC LA
Faculty of Sciences, University of Porto, Portugal
miguel-areias@dcc.fc.up.pt *ricroc@dcc.fc.up.pt*

Yap Prolog: *http://www.dcc.fc.up.pt/~vsc/Yap*
Project SIBILA: *http://cracs.fc.up.pt/*

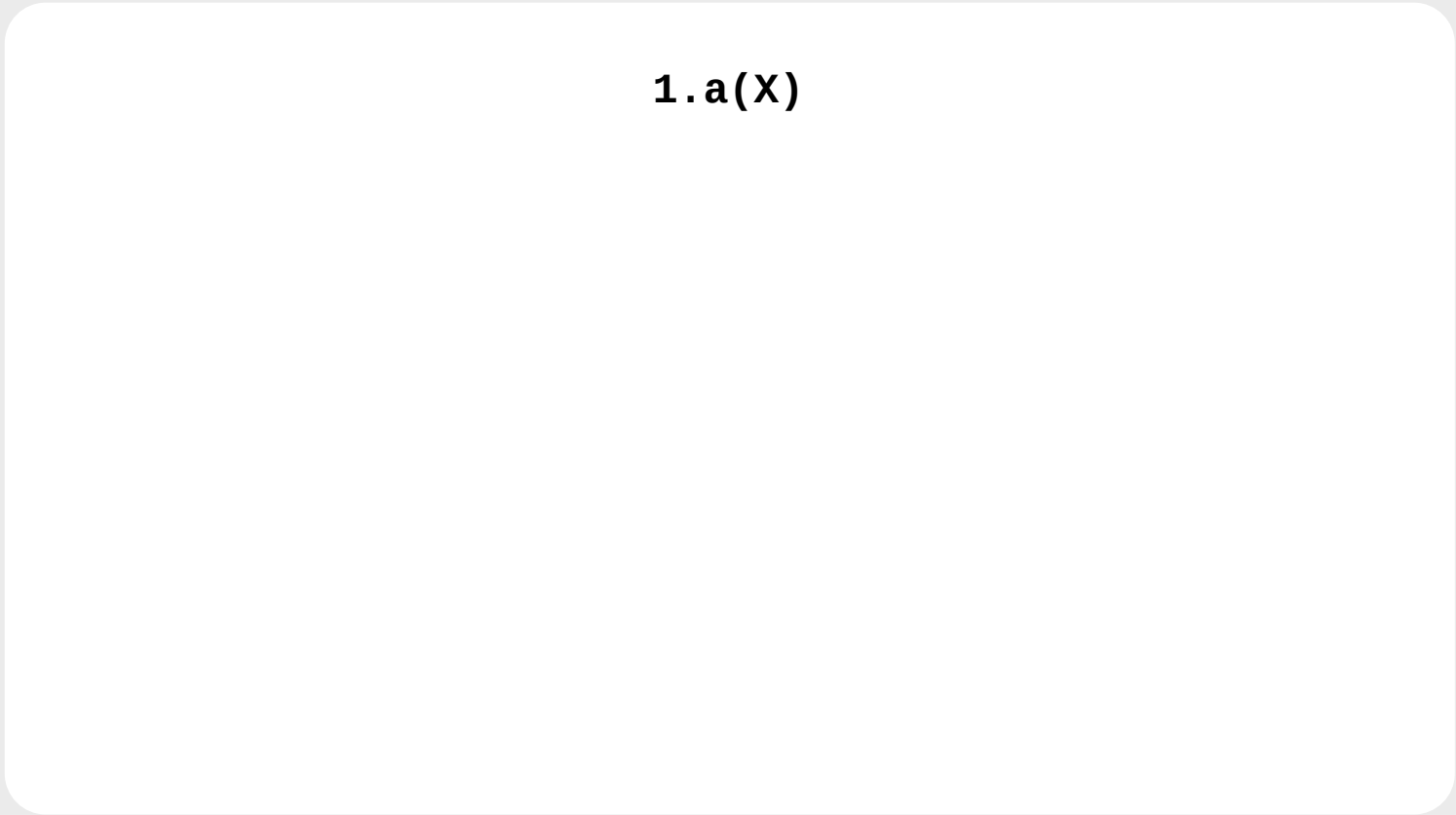
SLATE'15 SYMPOSIUM ON LANGUAGES, APPLICATIONS AND TECHNOLOGIES

Prolog and SLD Resolution

- Prolog systems are known to have good performances and flexibility, but they are based on SLD resolution, which limits the potential of the Logic Programing paradigm.
- SLD resolution cannot deal properly with the following situations:
  - **Positive Infinite Cycles** (insufficient expressiveness)
  - **Negative Infinite Cycles** (inconsistence)
  - **Redundant Computations** (inefficiency)

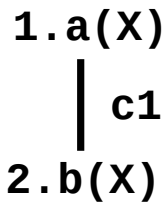
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



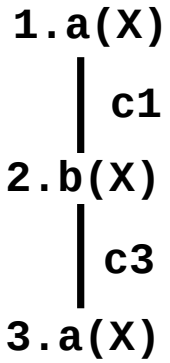
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



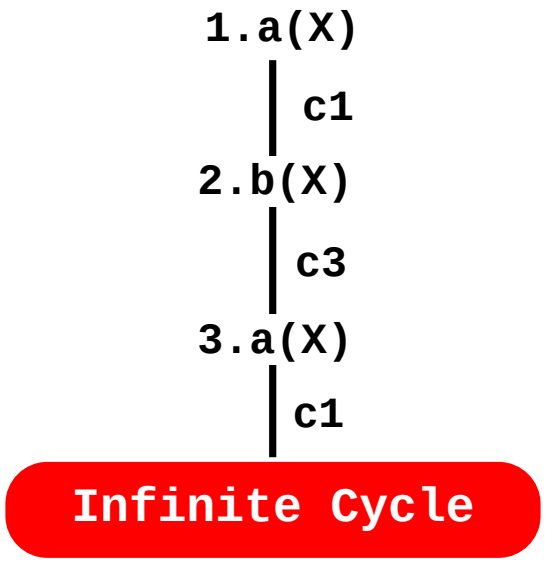
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



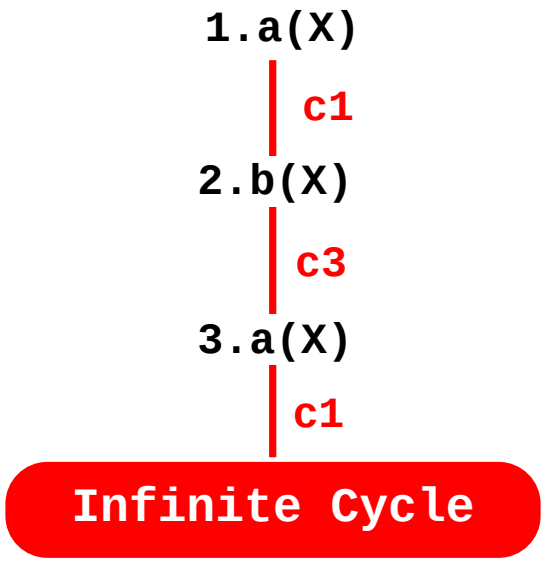
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



Tabling in Prolog Systems

- **Tabling** is an **implementation technique** that **overcomes** some of the **limitations** of **Prolog** systems:
  - Tabled subgoals are evaluated by storing their answers in an appropriate data space, called the **table space**.
  - Repeated calls to tabled subgoals are resolved by **consuming** the answers already stored in the table instead of **being re-evaluated** against the program clauses.

Tabling in Prolog Systems

- **Tabling** is an **implementation technique** that **overcomes** some of the **limitations** of **Prolog** systems:
  - Tabled subgoals are evaluated by storing their answers in an appropriate data space, called the **table space**.
  - Repeated calls to tabled subgoals are resolved by **consuming** the answers already stored in the table instead of **being re-evaluated** against the program clauses.

- Implementations of **Tabling** are currently available in systems like:
  - XSB Prolog, **Yap Prolog**, B-Prolog, ALS-Prolog, Mercury, Ciao Prolog and more recently Picat.
- **Multithreading** combined with **Tabling**:
  - XSB Prolog
  - **YapTab-Mt [ICLP 2012]**.

YapTab-Mt - Advantages

- An **abstraction layer** with **high-level constructors** that provide access to the **dynamic programming (tabling)** support:
  - Instruction: **:- table predicate/arity.**
  - Scheduling: **:- tabling_mode(predicate, batched).**

YapTab-Mt - Advantages

- An **abstraction layer** with **high-level constructors** that provide access to the **dynamic programming (tabling)** support:
  - Instruction: **:- table predicate/arity.**
  - Scheduling: **:- tabling_mode(predicate, batched).**

- **Thread API** is **POSIX Threads compliant**:
  - **Management** - creating, joining , yielding, etc.
  - **Monitoring** - statistics, properties, etc.
  - **Synchronization** - mutex creation, statistics, etc.

YapTab-Mt - Advantages

- An **abstraction layer** with **high-level constructors** that provide access to the **dynamic programming (tabling)** support:
  - Instruction: **:- table predicate/arity.**
  - Scheduling: **:- tabling_mode(predicate, batched).**

- **Thread API** is **POSIX Threads compliant**:
  - **Management** - creating, joining , yielding, etc.
  - **Monitoring** - statistics, properties, etc.
  - **Synchronization** - mutex creation, statistics, etc.

- Write complex **dynamic programming** applications using the **Prolog** programming language.
  - **Procedures** in **Prolog** can be written as **logical specifications**, which are closer to **mathematical notation**.

Table Space - Internal Architecture

- **Table Entry**: stores generic about the predicates.
  - **:-table predicate/2**.

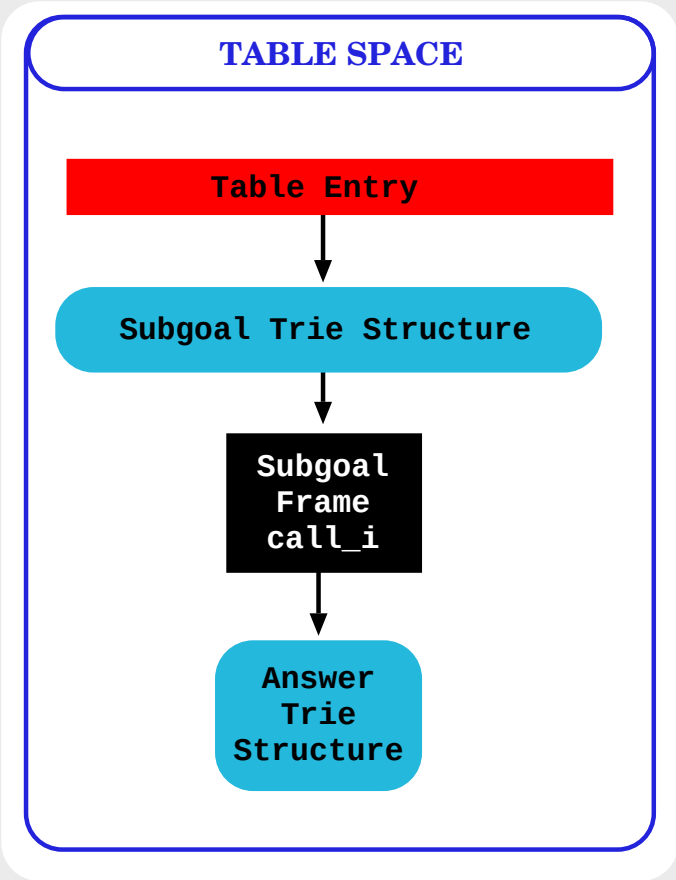


Table Space - Internal Architecture

- **Table Entry**: stores generic about the predicates.
  - **:-table predicate/2**.
- **Subgoal Trie Structure**: stores the **identifier** of the computations.
  - **predicate**(**computation_id**, Answer).

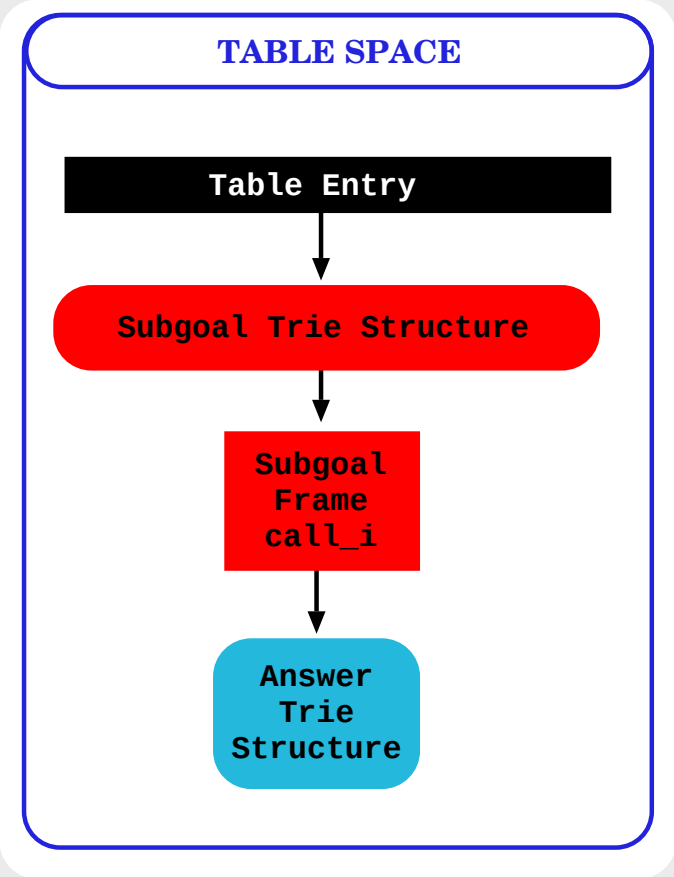


Table Space - Internal Architecture

- **Table Entry**: stores generic about the predicates.
  - **:-table predicate/2**.
- **Subgoal Trie Structure**: stores the **identifier** of the computations.
  - **predicate**(**computation_id**, Answer).
- **Answer Trie Structure**: stores the **answers** of the computations.
  - **predicate**(computation_id, **Answer**).

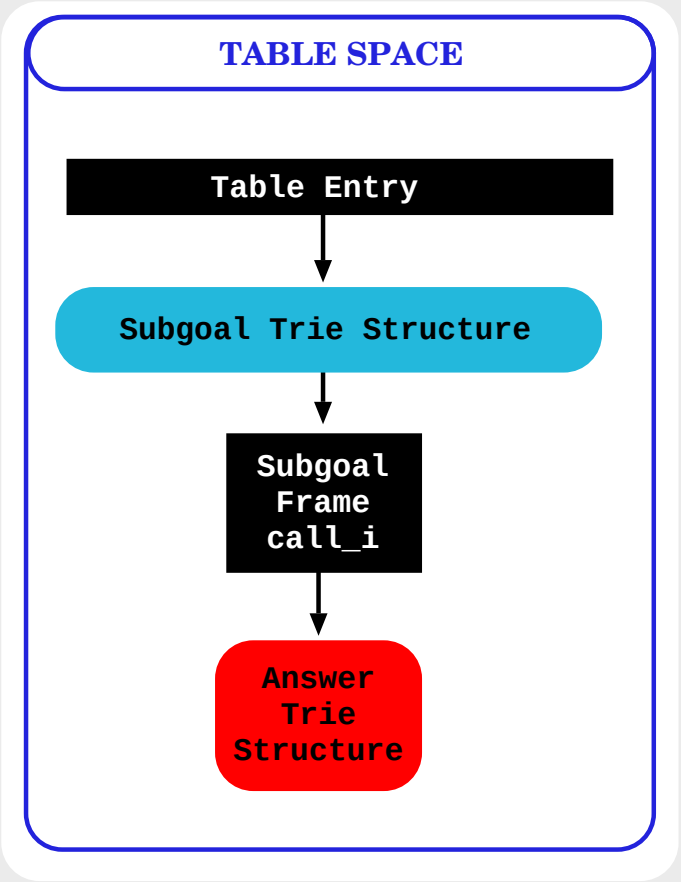


Table Space - Example

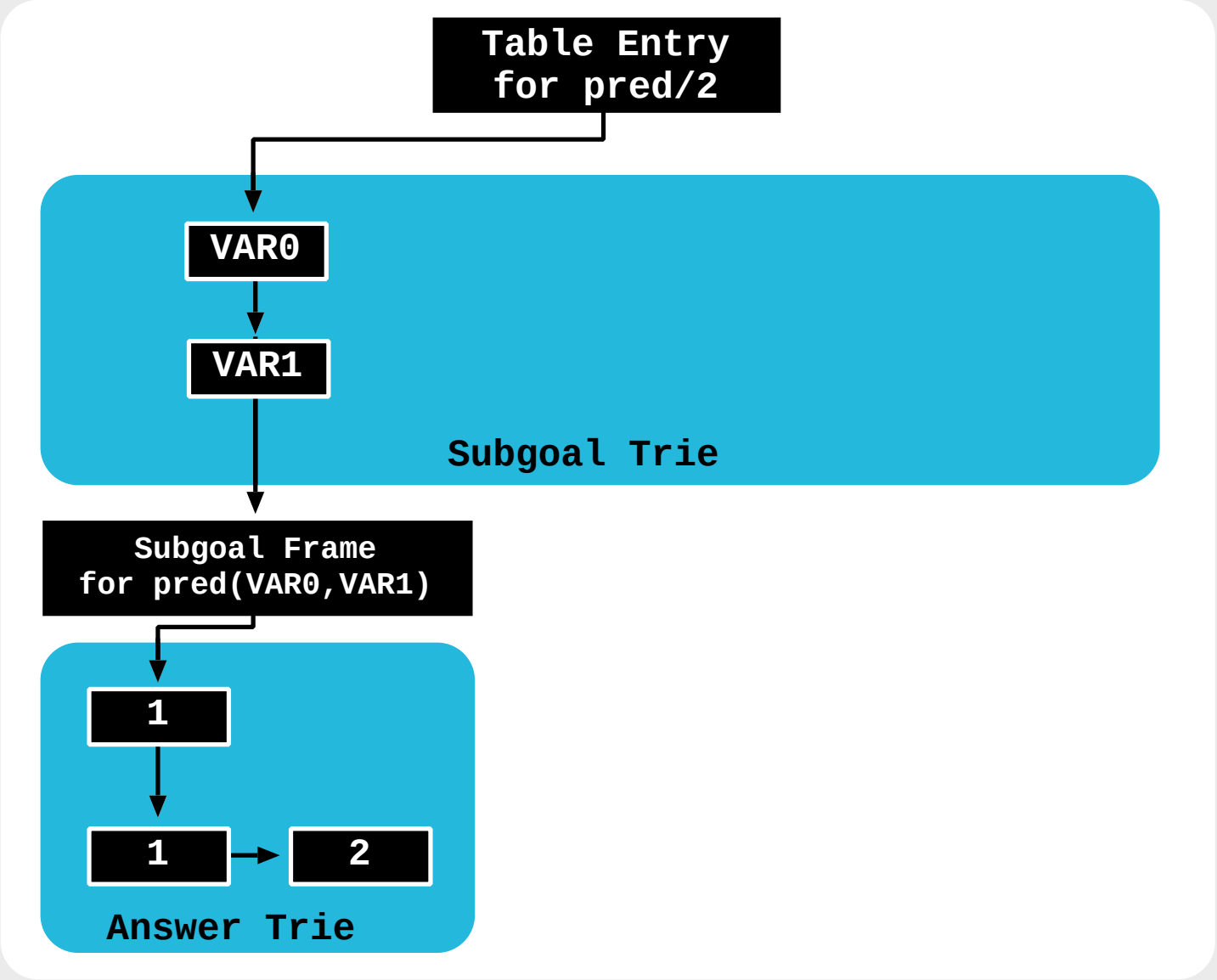
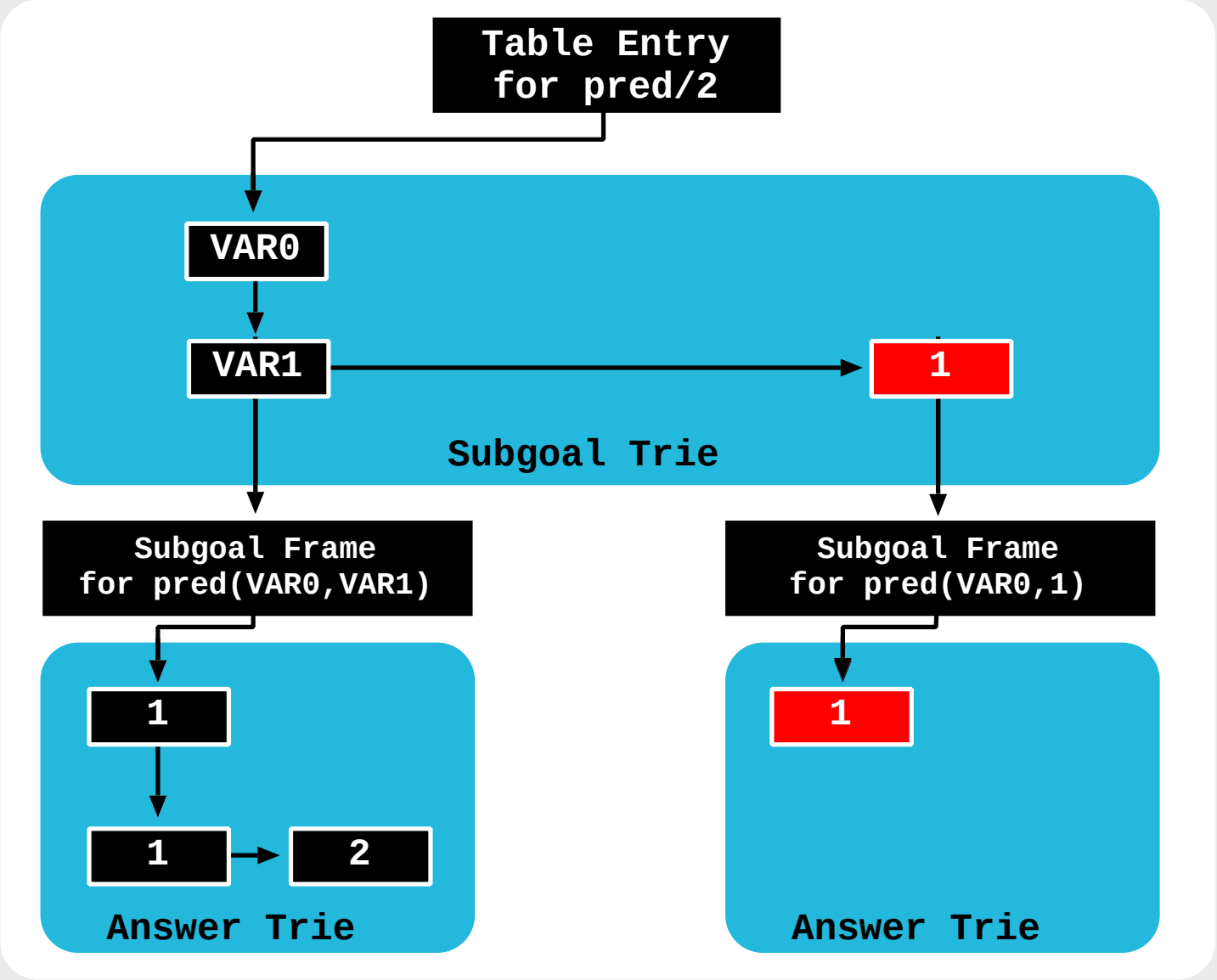
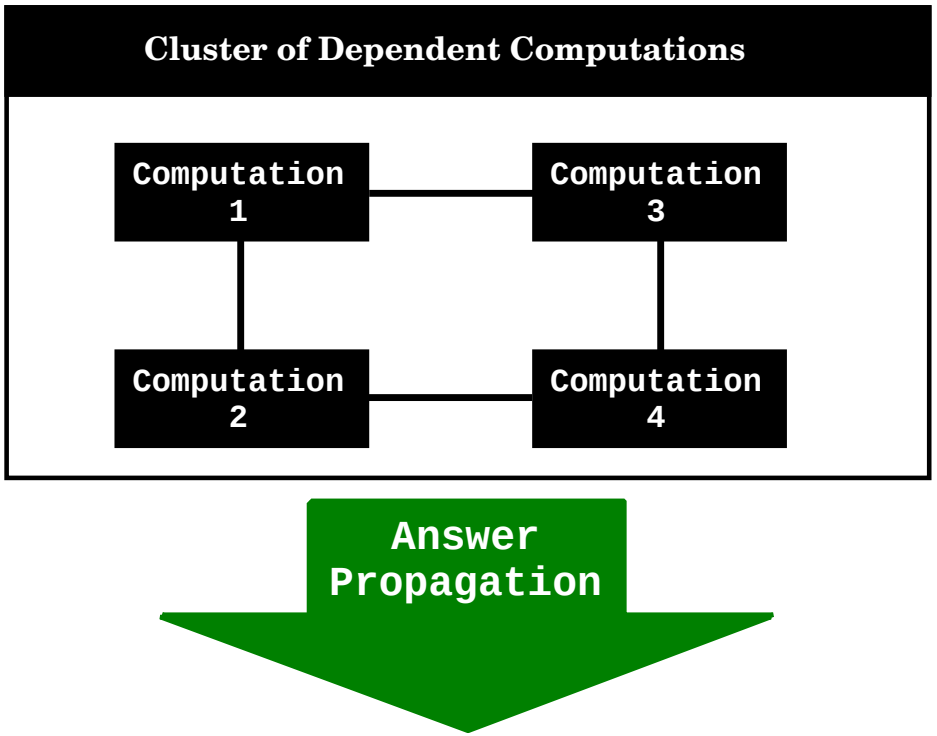


Table Space - Example



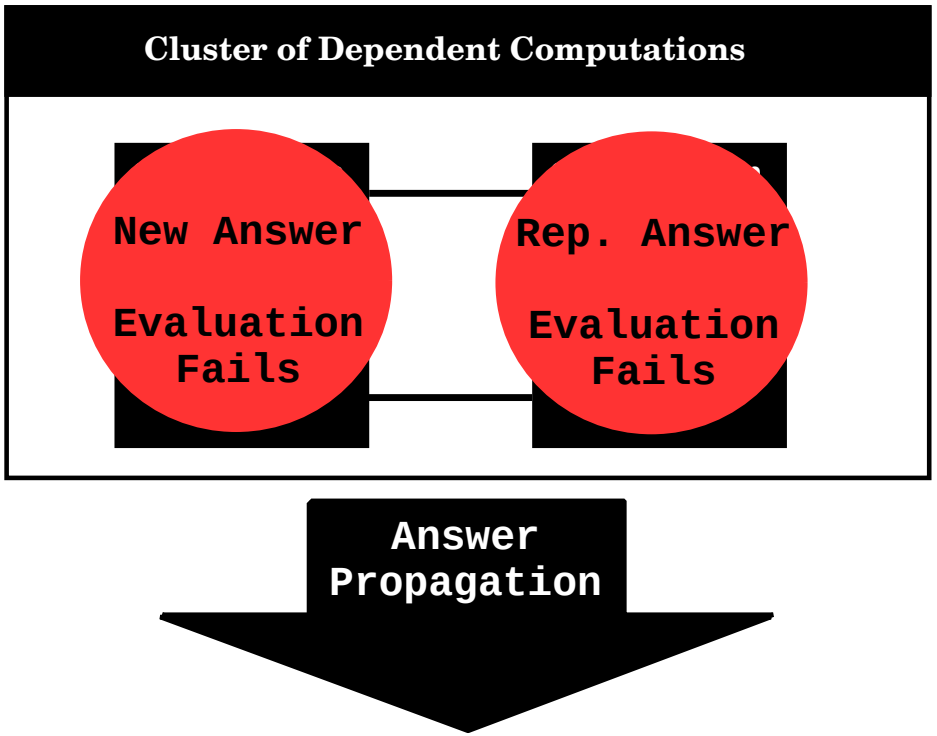
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING



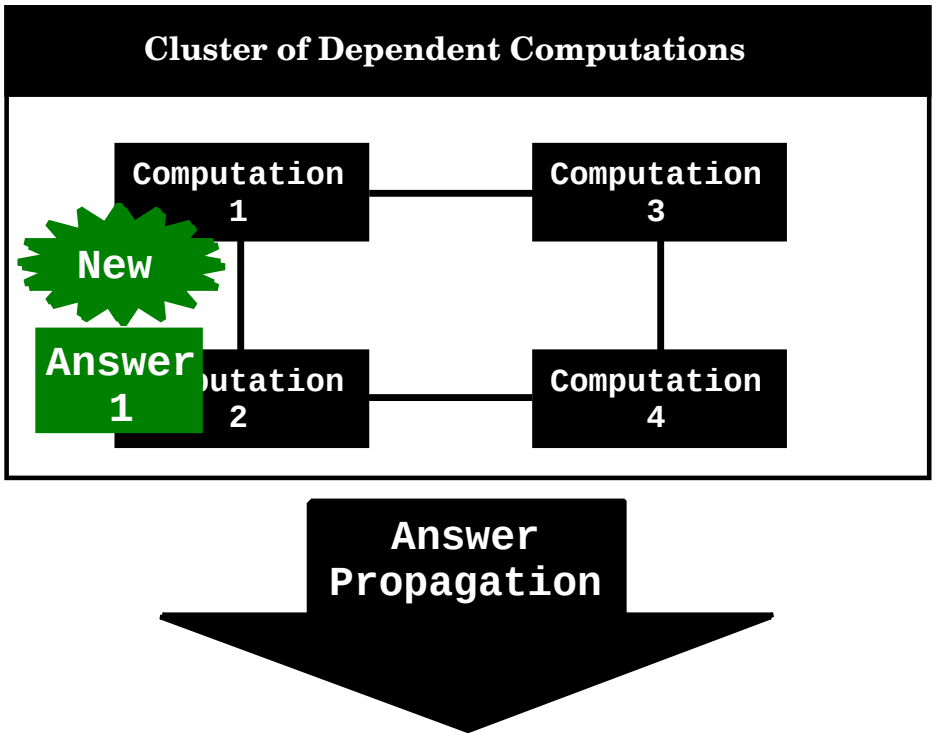
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING



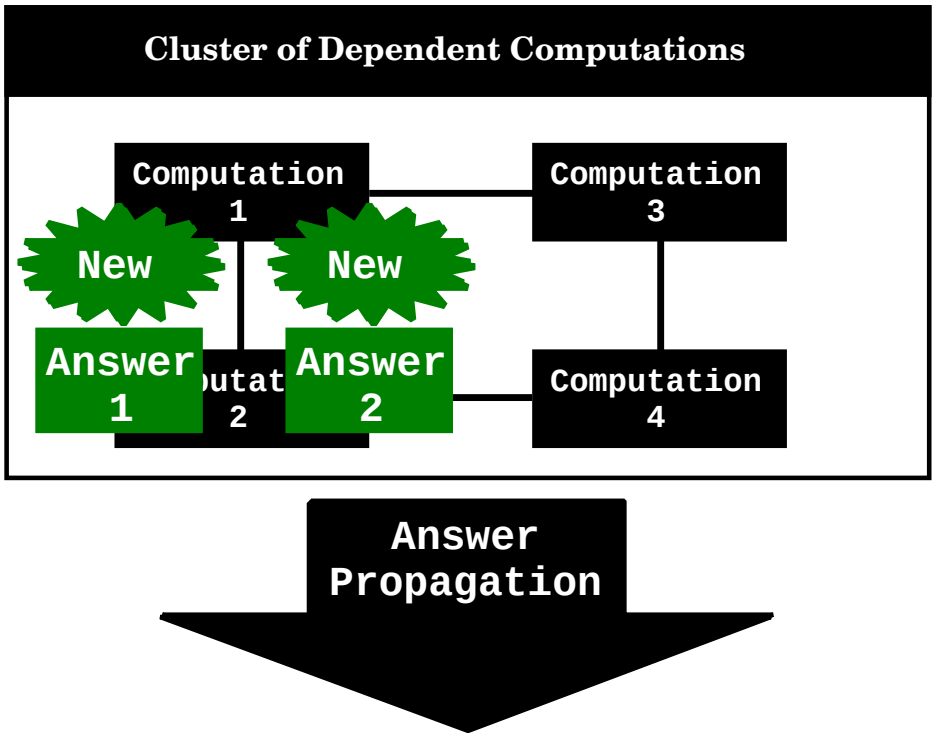
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING



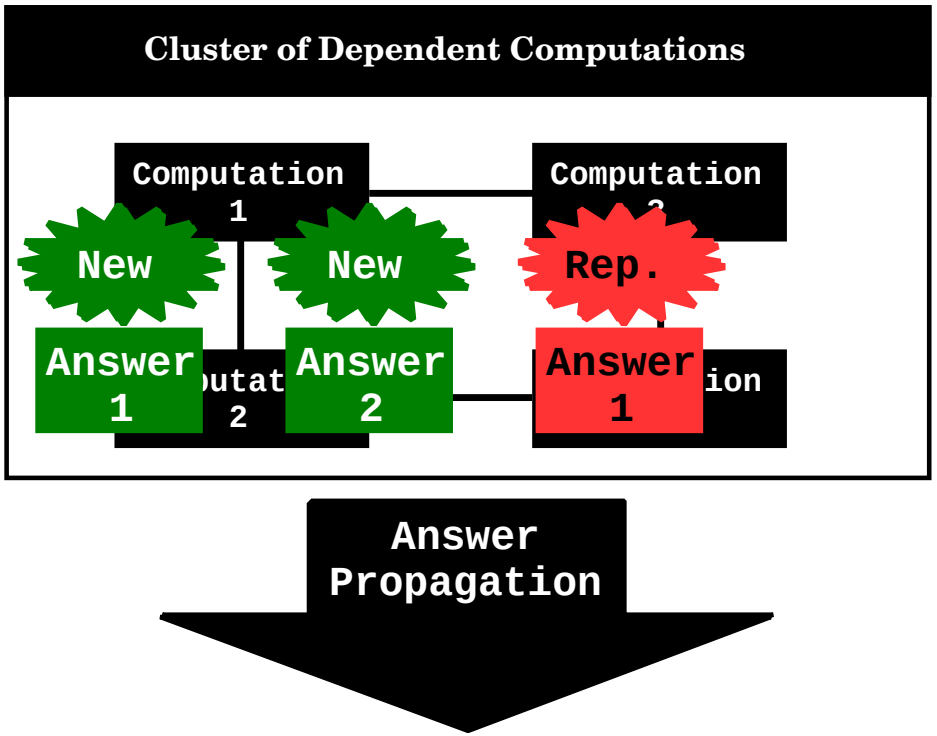
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING



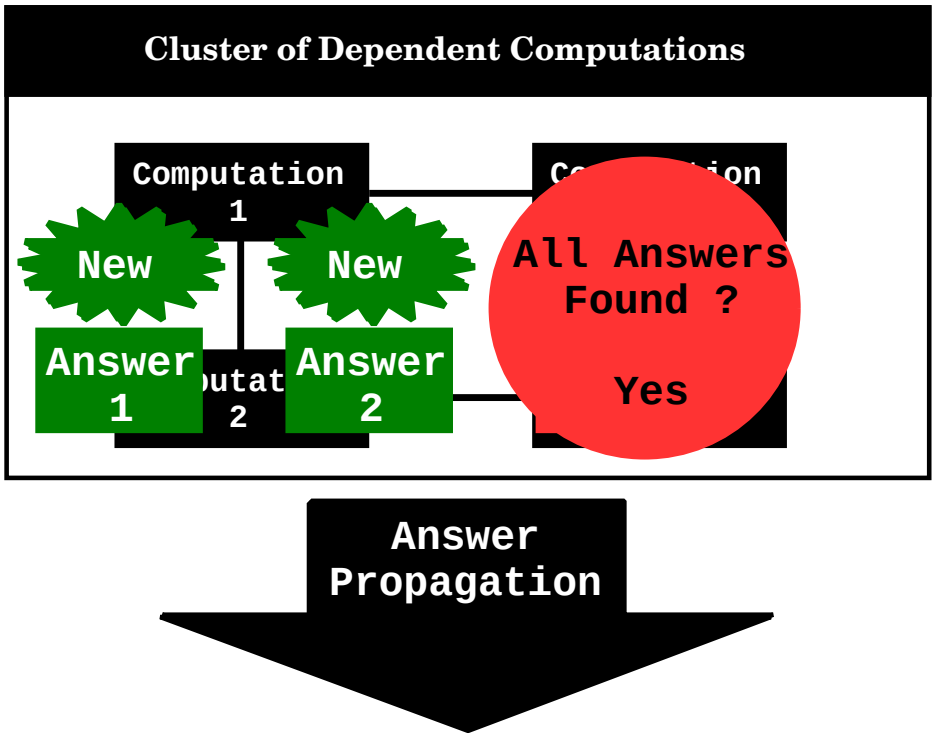
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING



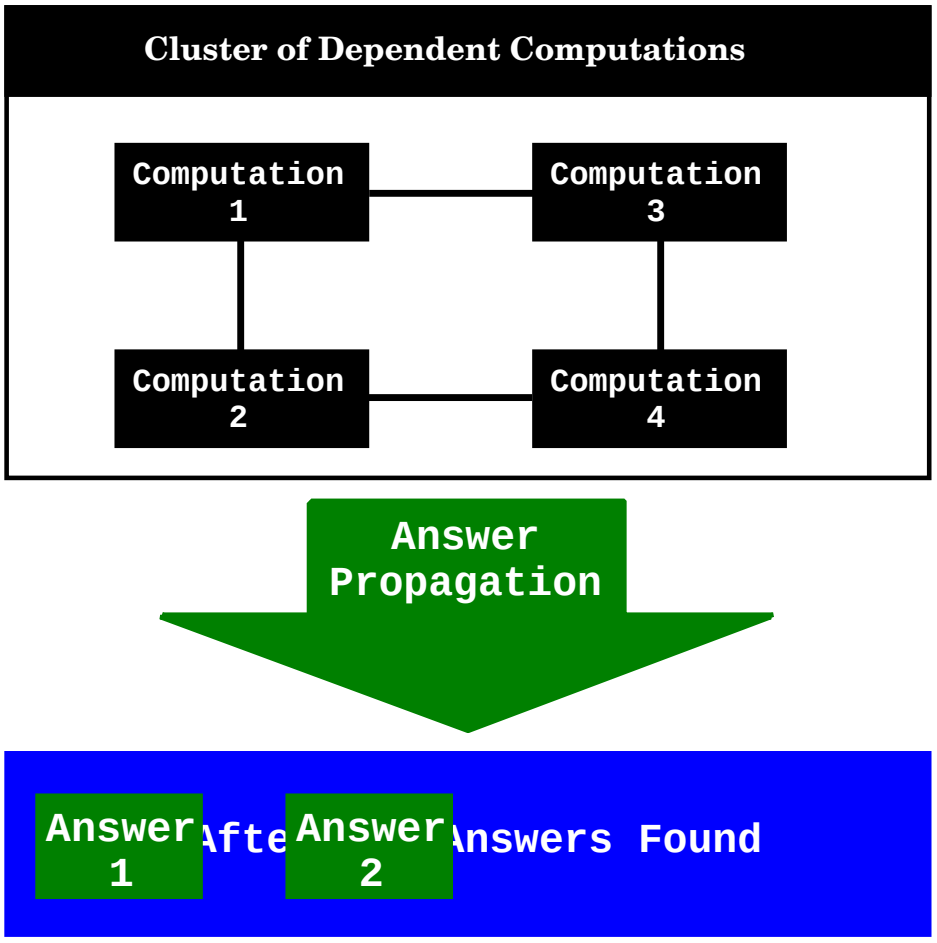
Tabling Scheduling Strategies - Local vs Batched

LOCAL SCHEDULING

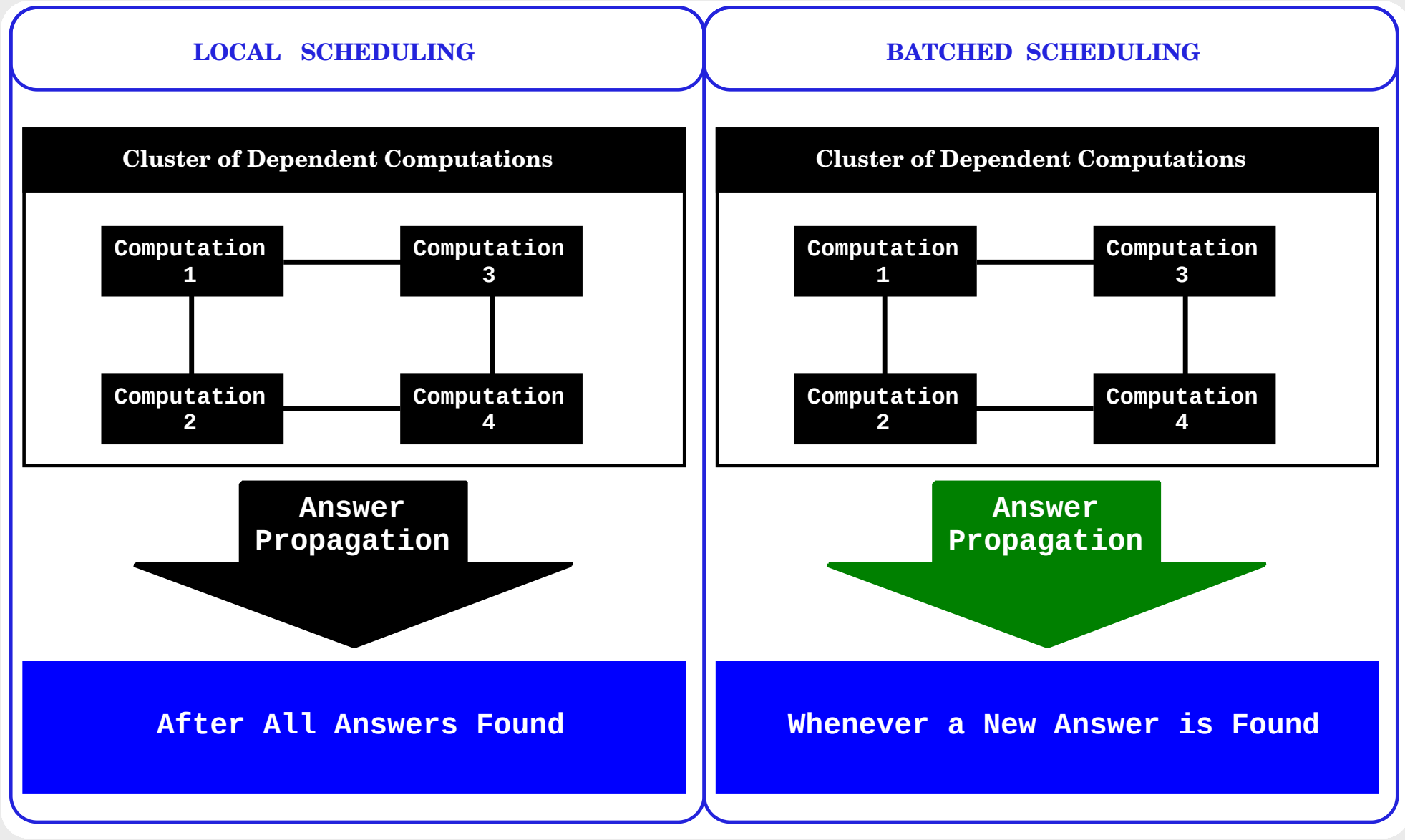


Tabling Scheduling Strategies - Local vs Batched

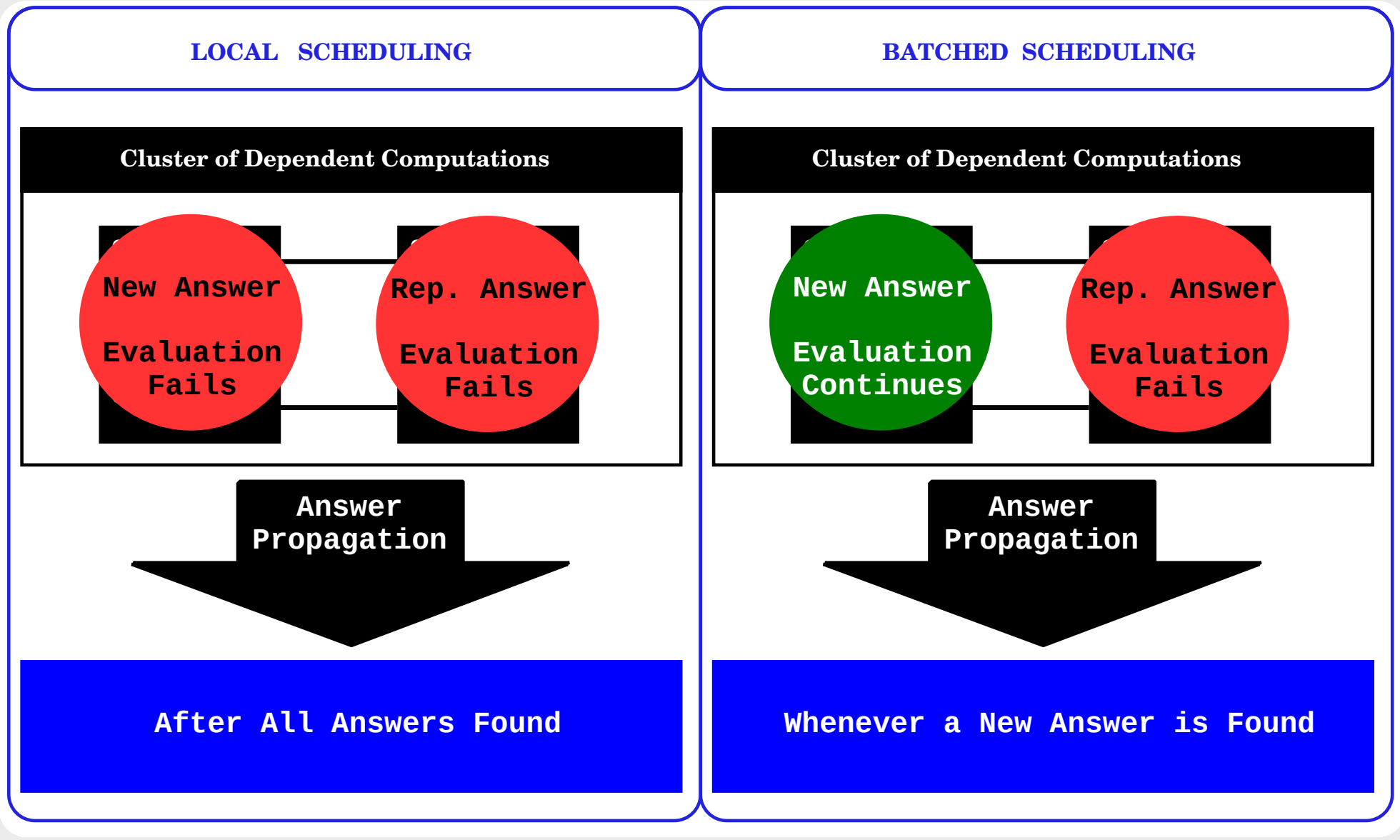
LOCAL SCHEDULING



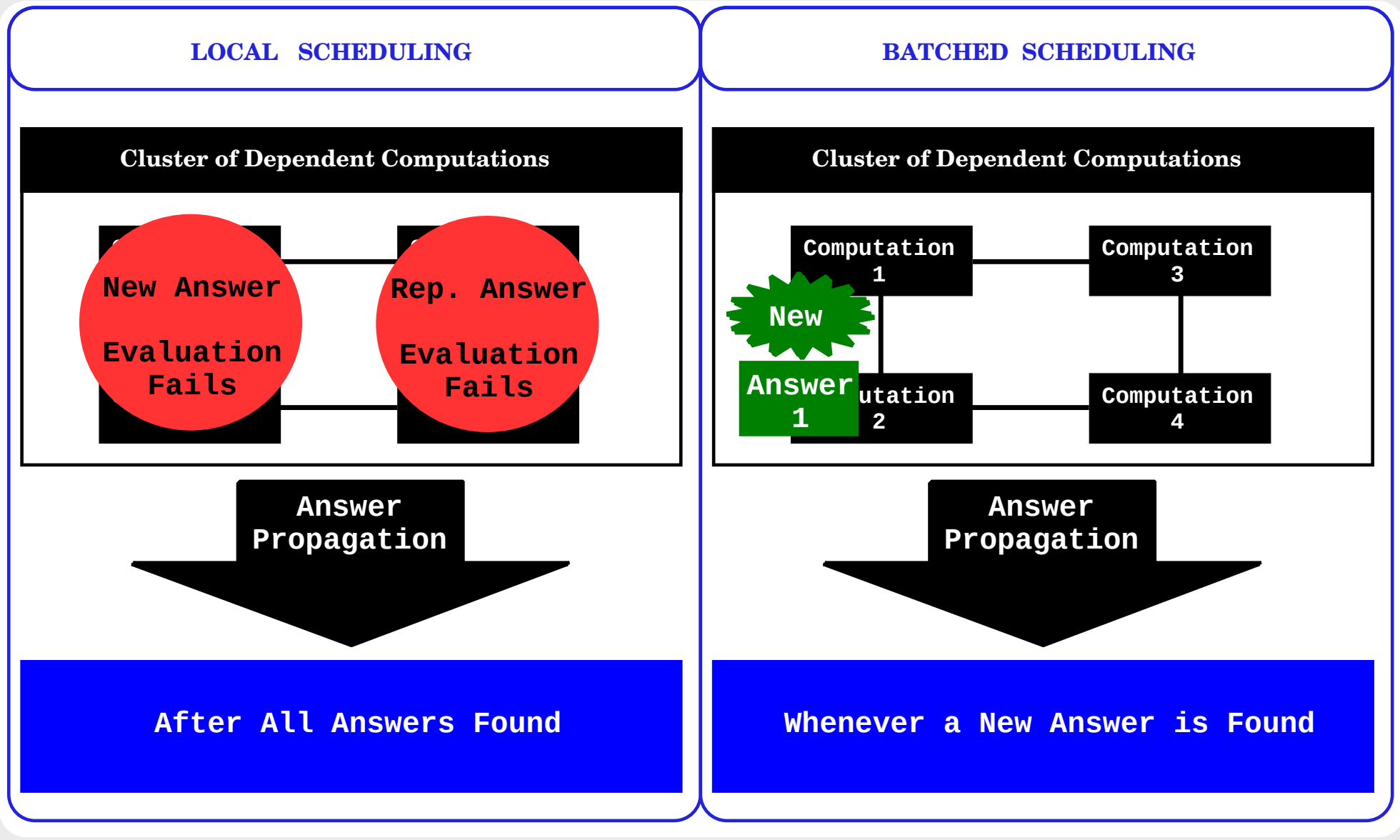
Tabling Scheduling Strategies - Local vs Batched



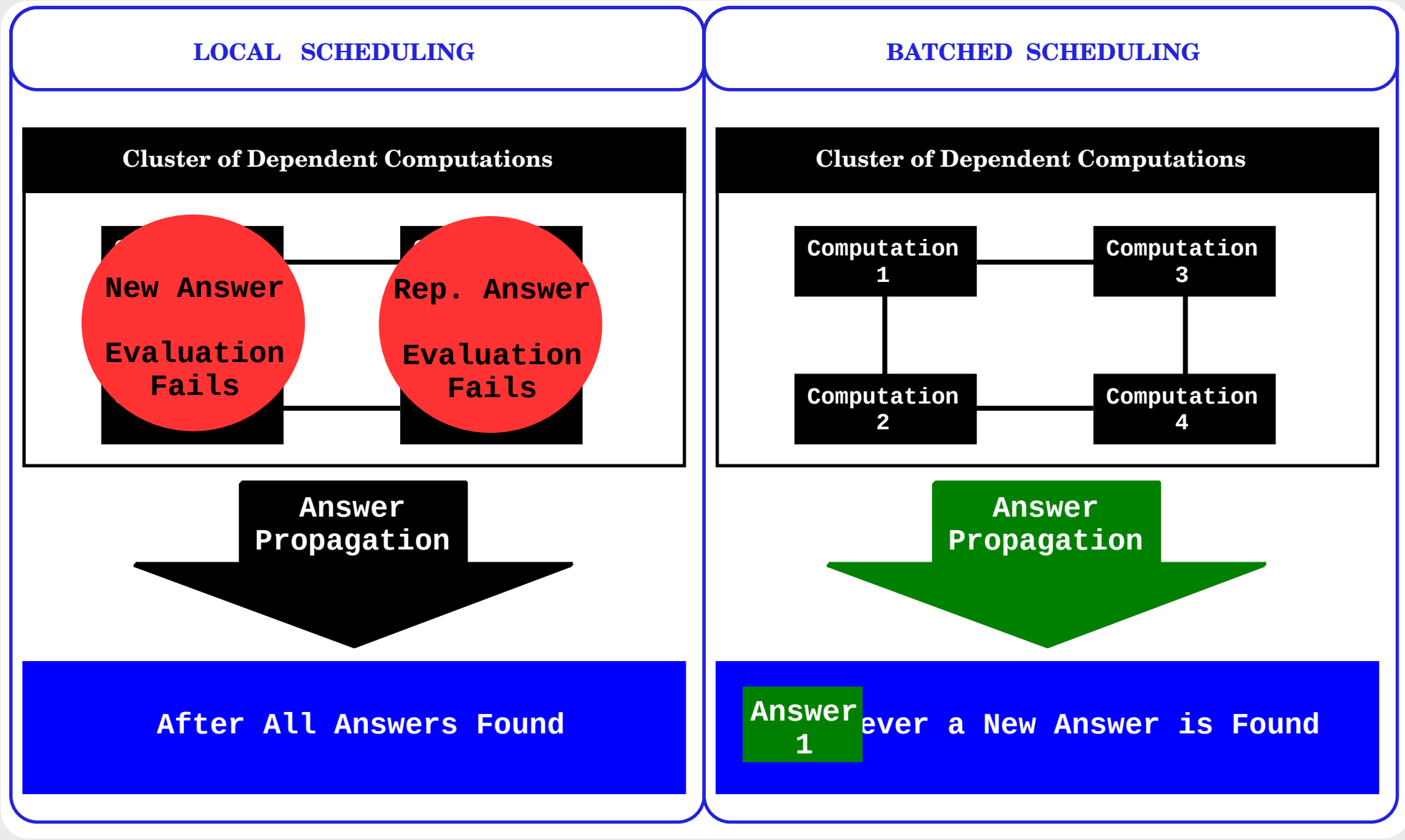
Tabling Scheduling Strategies - Local vs Batched



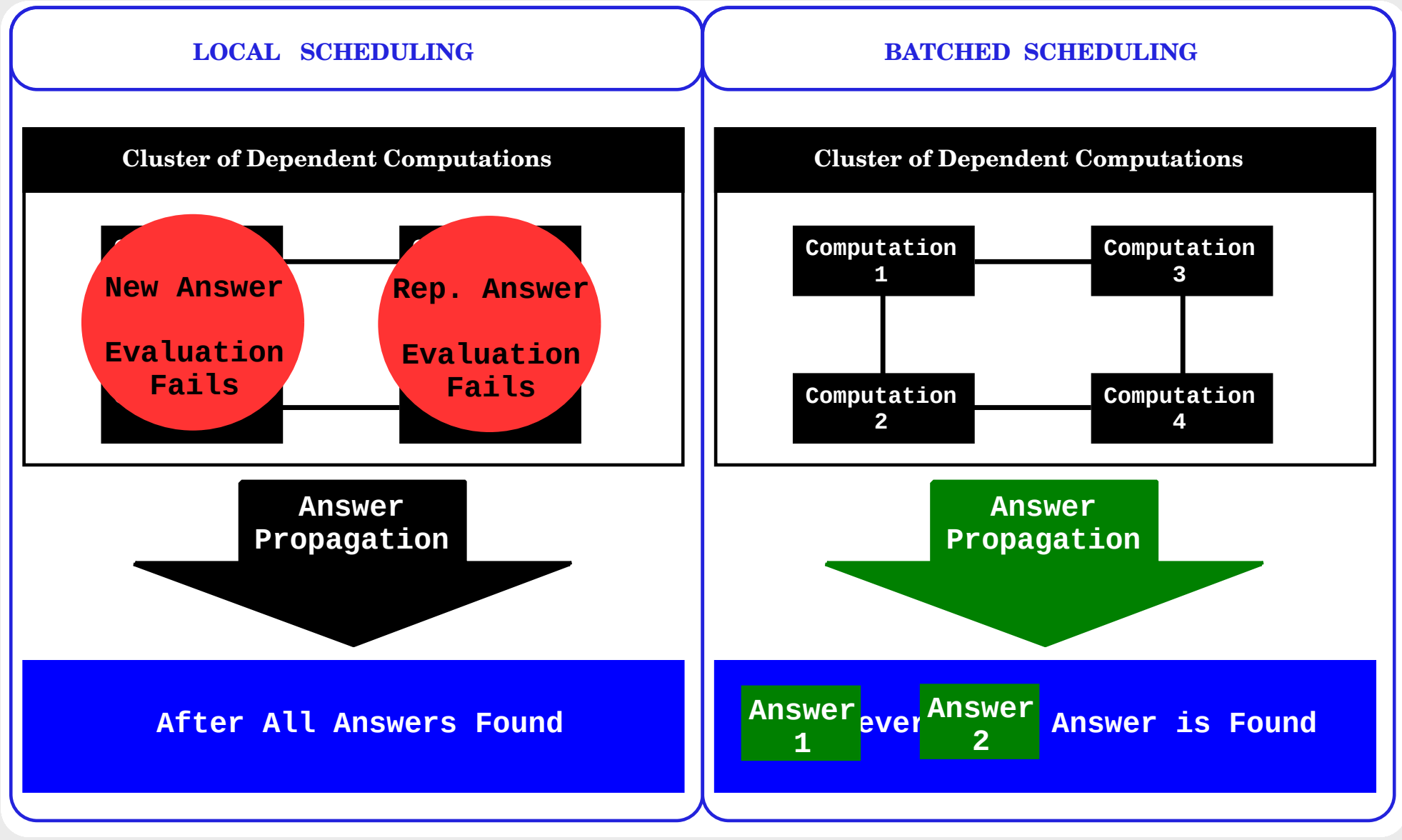
Tabling Scheduling Strategies - Local vs Batched



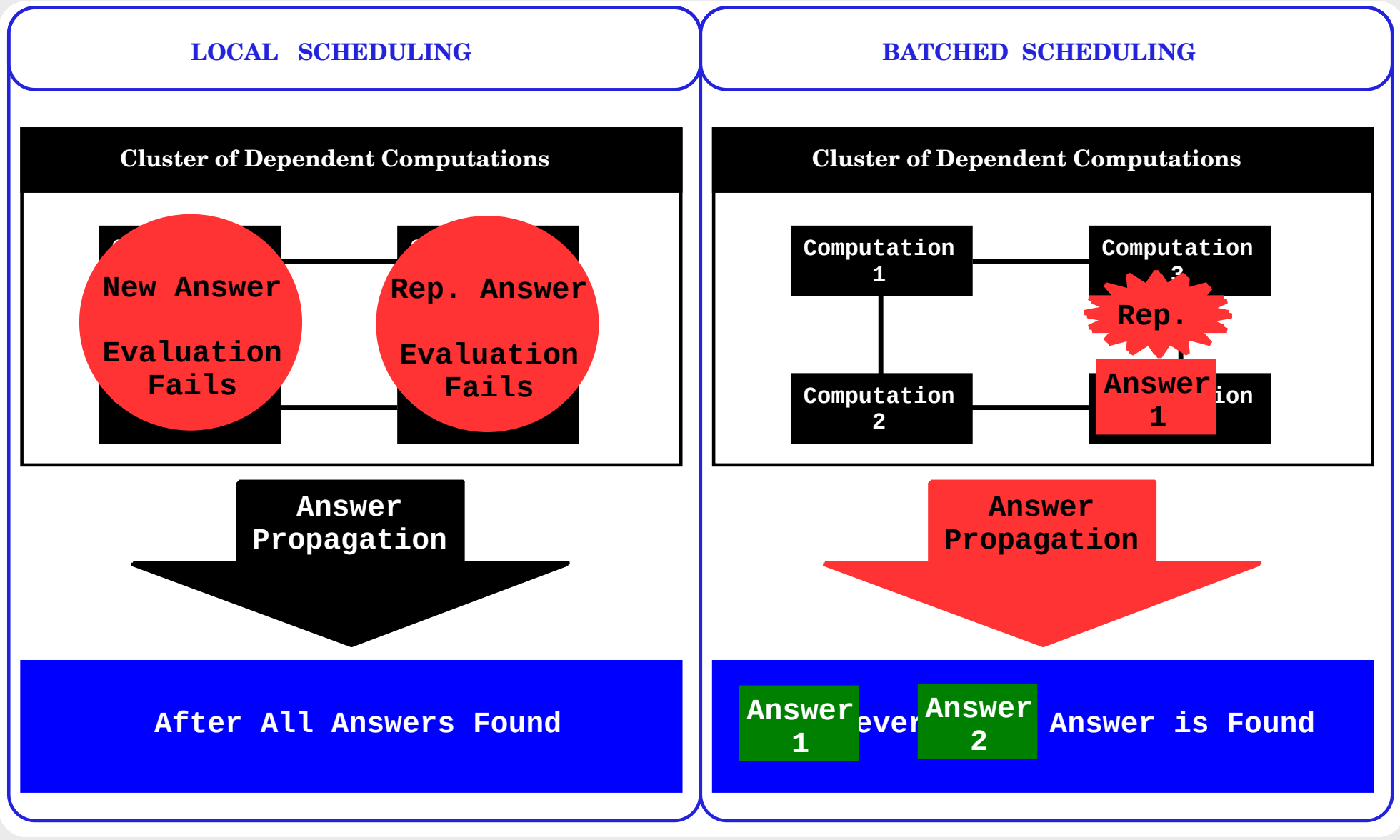
Tabling Scheduling Strategies - Local vs Batched



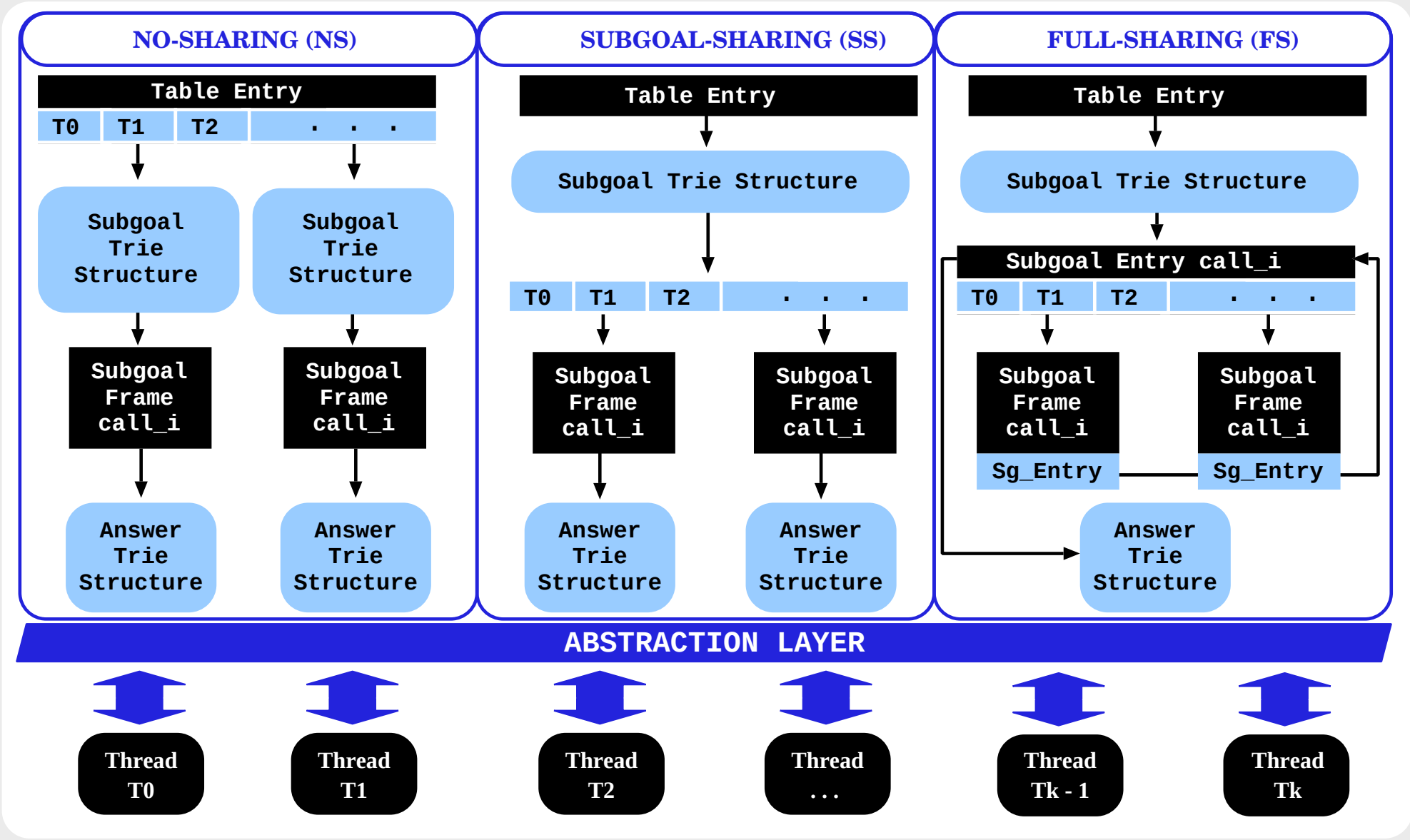
Tabling Scheduling Strategies - Local vs Batched



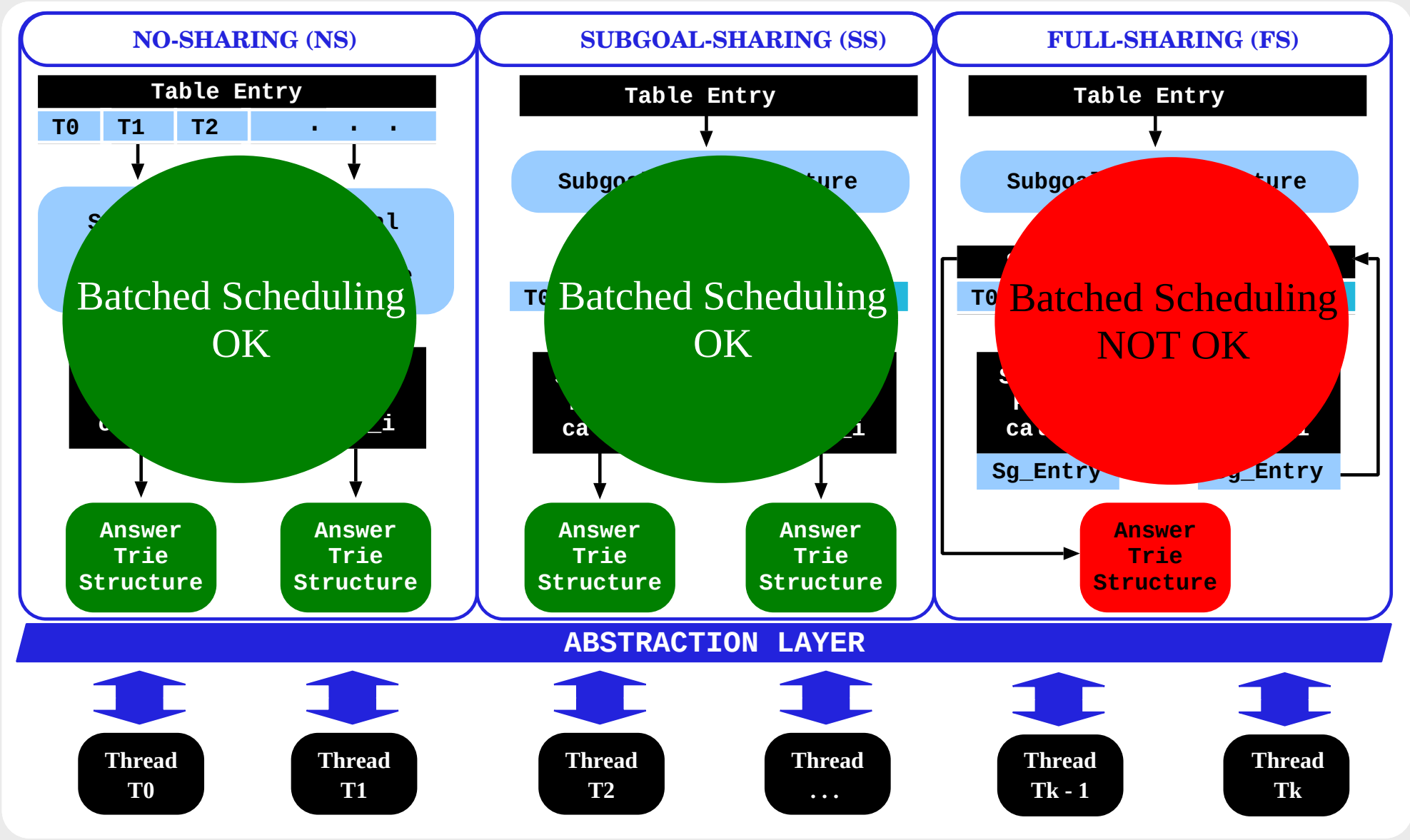
Tabling Scheduling Strategies - Local vs Batched



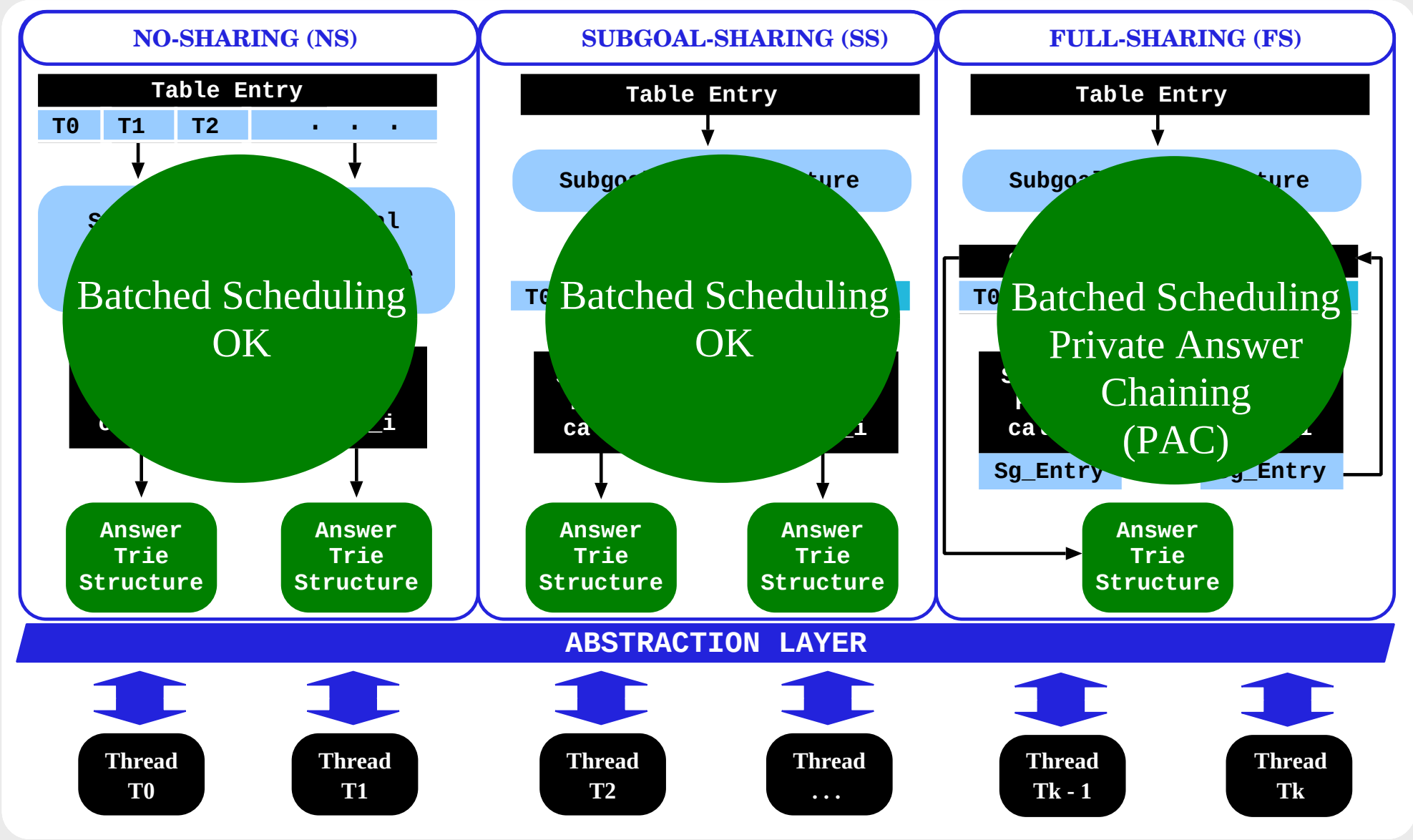
YapTab-Mt - Internal Architecture



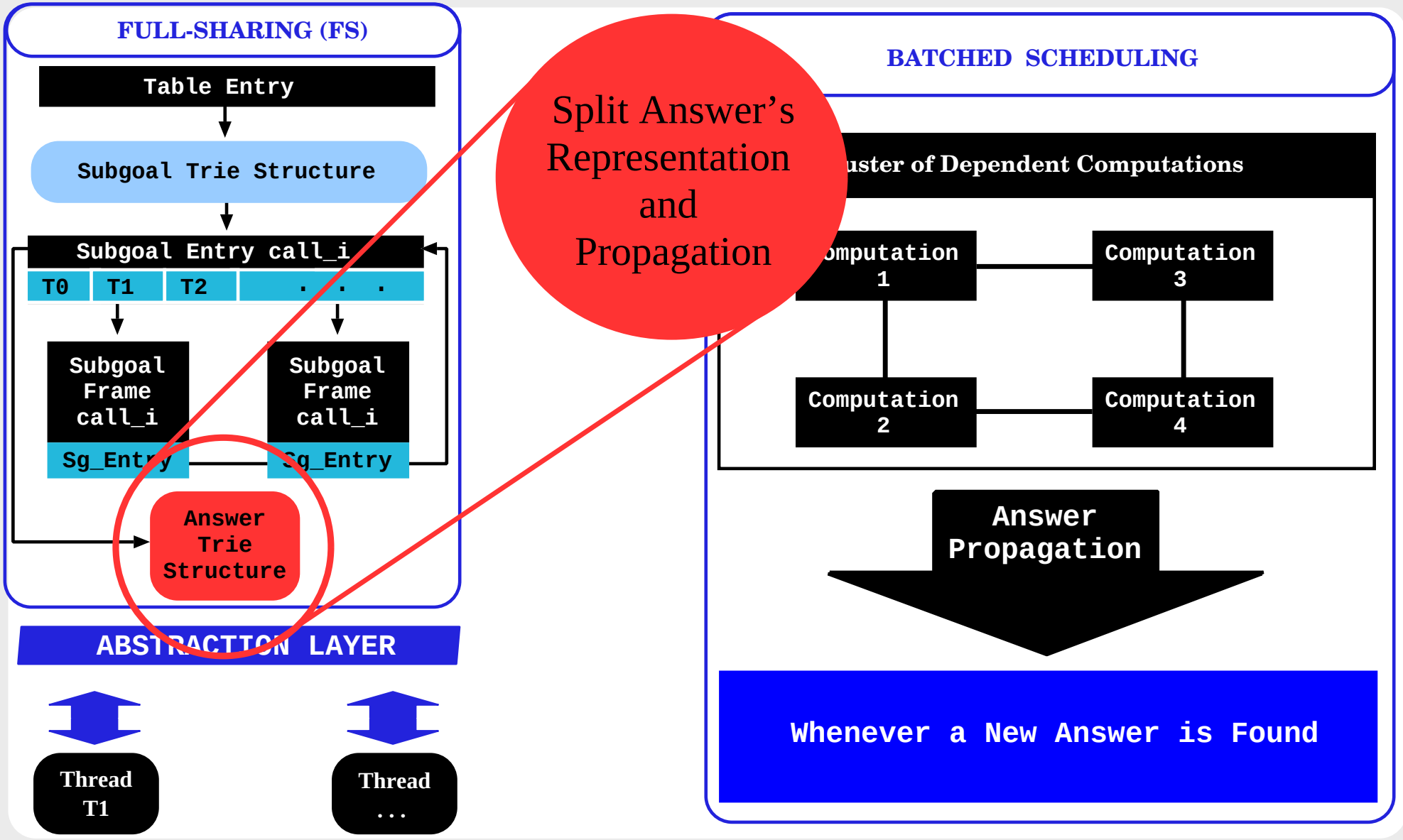
YapTab-Mt - Internal Architecture



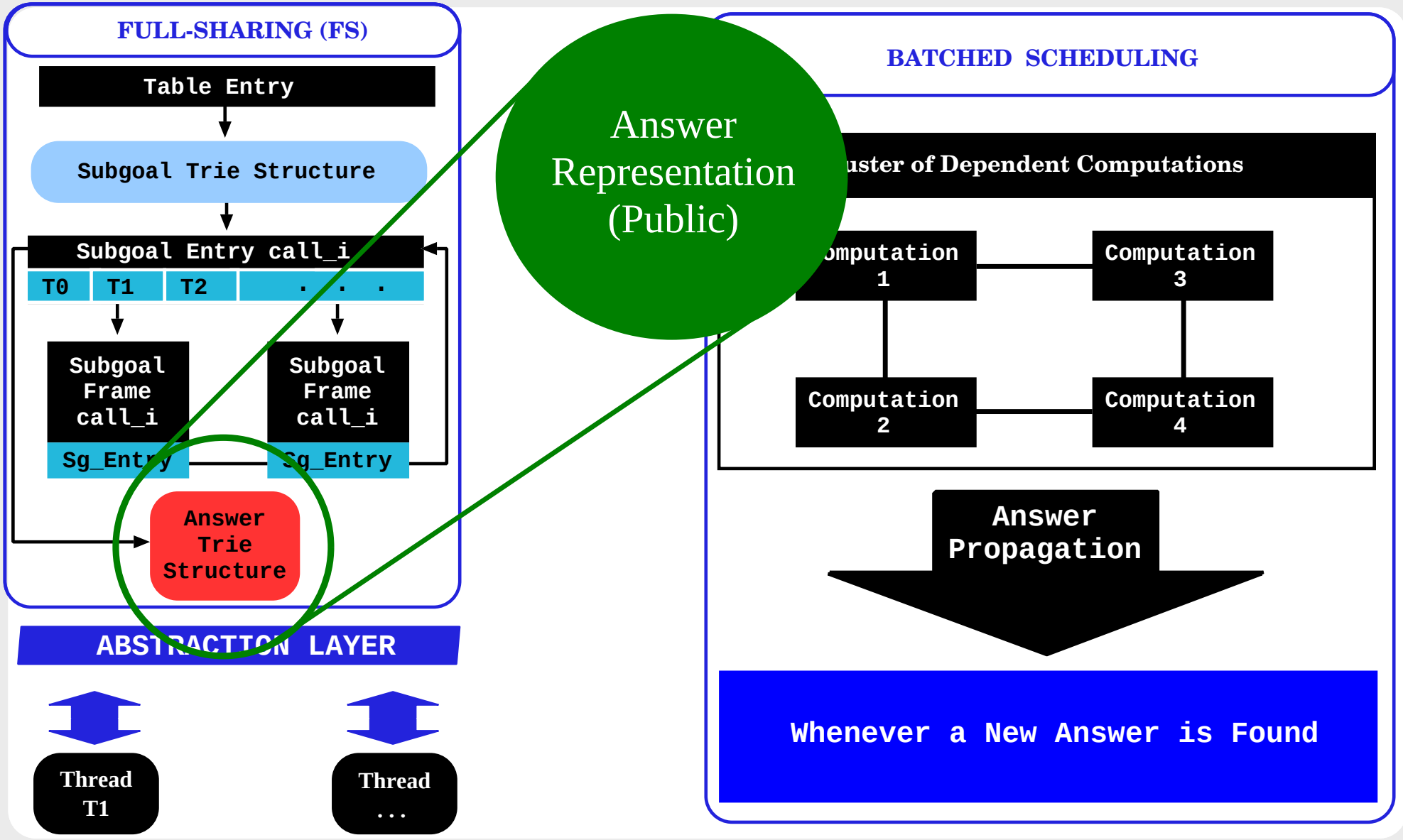
YapTab-Mt - Internal Architecture



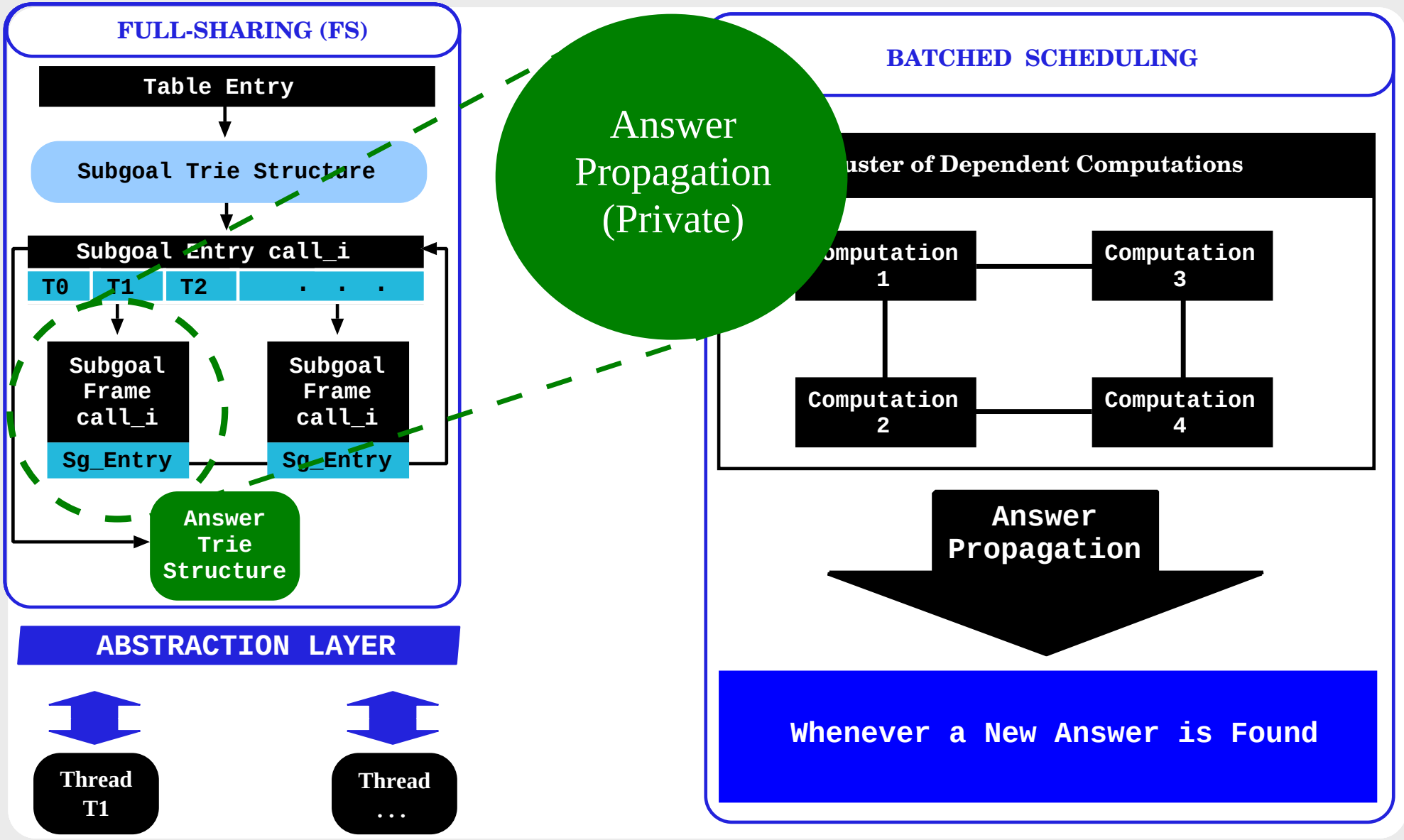
Private Answer Chaining - Key Idea



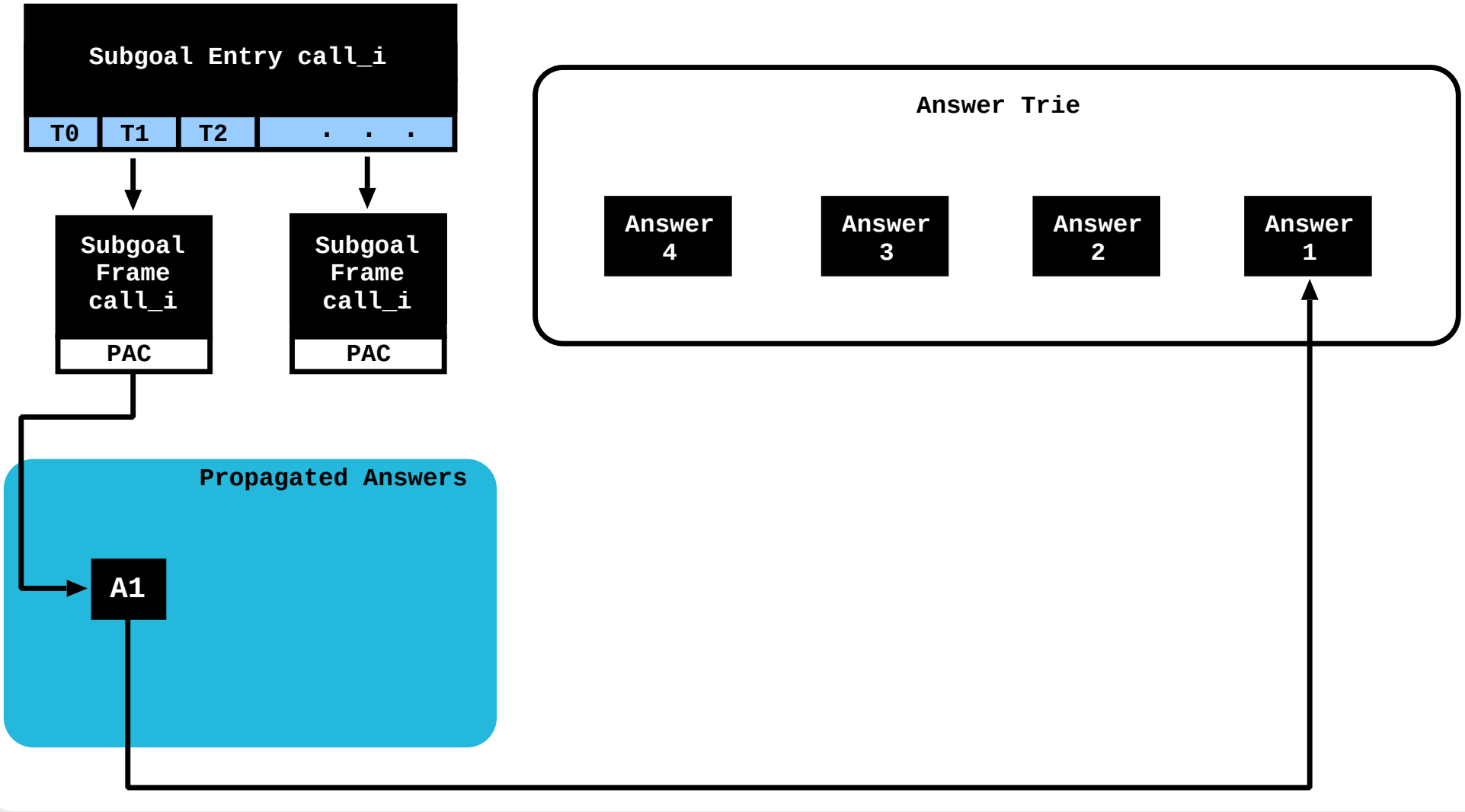
Private Answer Chaining - Key Idea



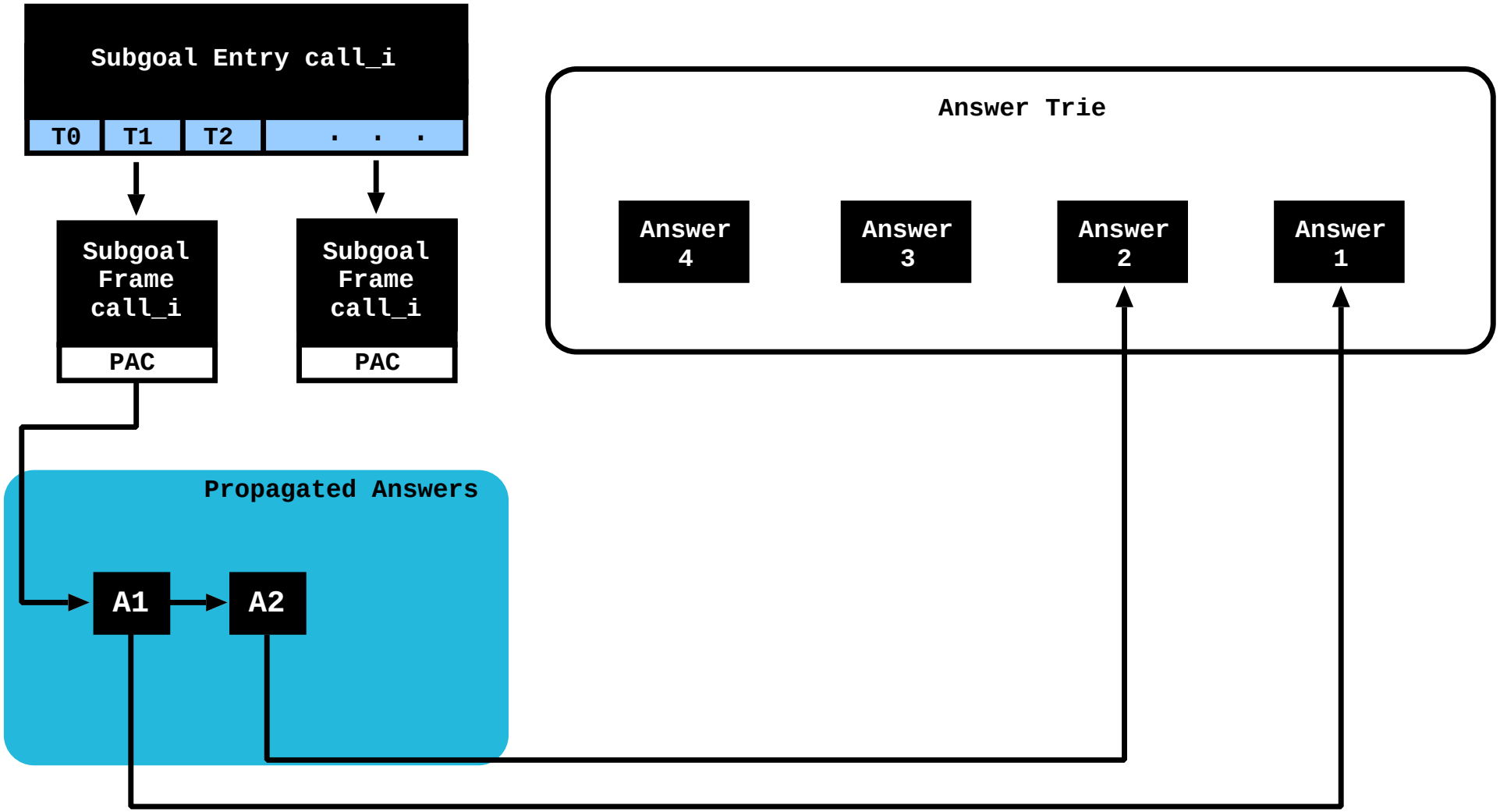
Private Answer Chaining - Key Idea



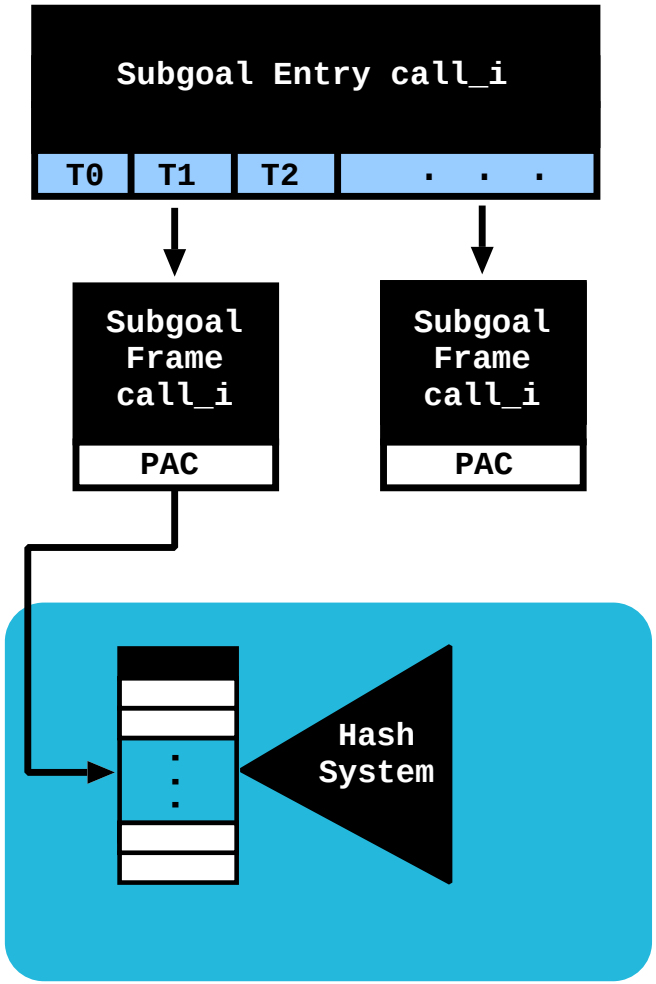
Private Answer Chaining - Internals

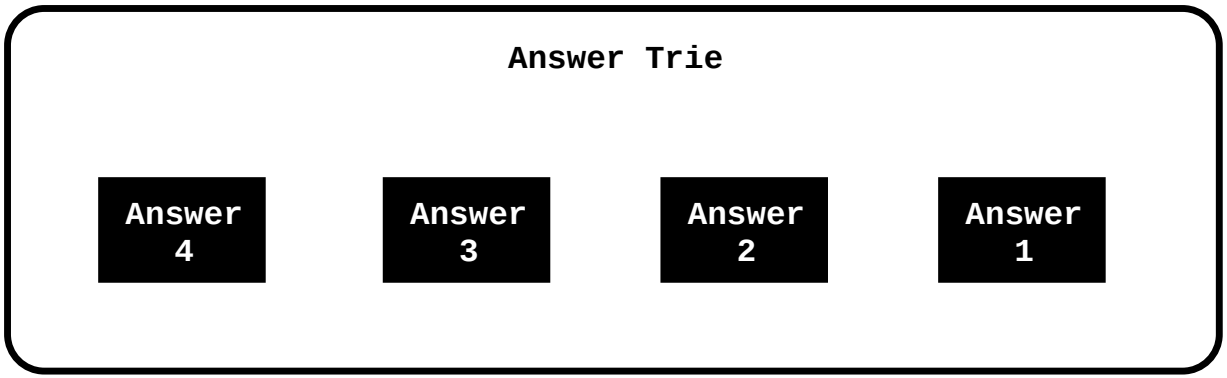


Private Answer Chaining - Internals

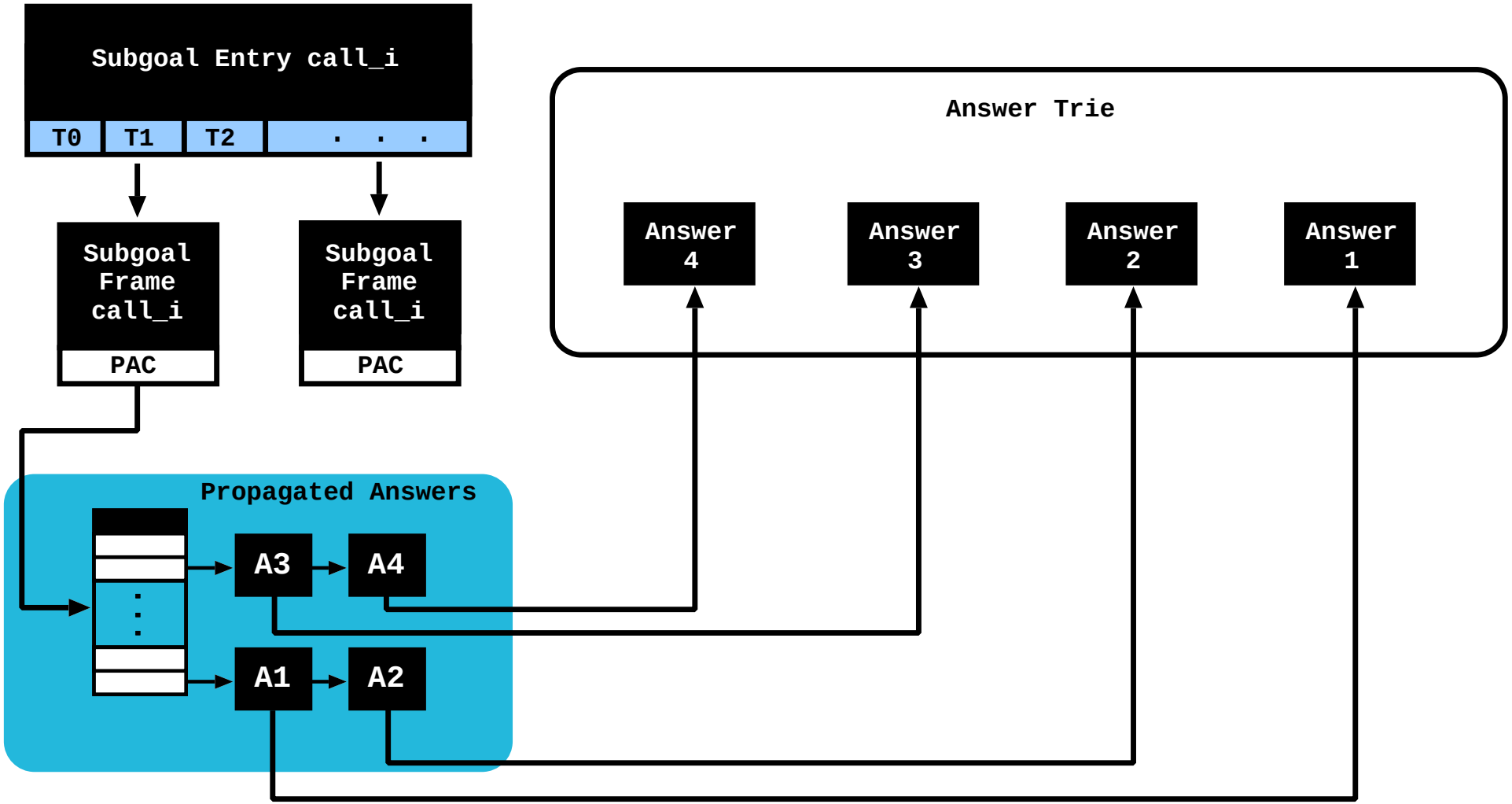


Private Answer Chaining - Internals

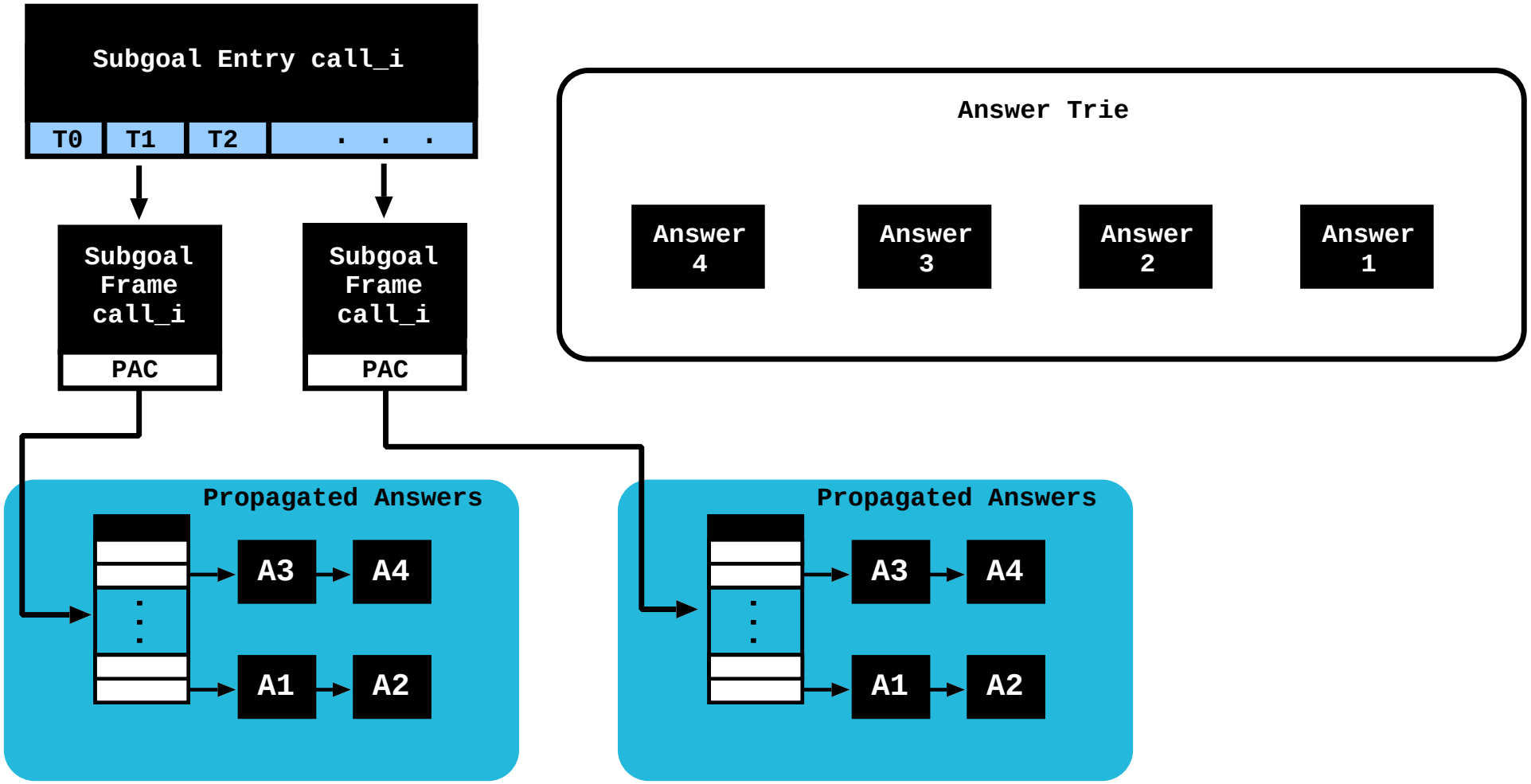




Private Answer Chaining - Internals



Private Answer Chaining - Internals



Experimental Results - Worst Case Scenarios

Threads		NS		FS	
		Local	Batched	Local	Batched
1	Min	**0.53**	0.55	1.01	**0.95**
	Avg	**0.78**	0.82	**1.30**	1.46
	Max	1.06	**1.05**	**1.76**	2.33
8	Min	0.66	**0.63**	1.16	**0.99**
	Avg	**0.85**	0.88	**1.88**	1.95
	Max	**1.12**	1.14	**2.82**	3.49
16	Min	0.85	**0.75**	1.17	**1.06**
	Avg	**0.98**	1.00	**1.97**	2.08
	Max	**1.16**	1.31	**3.14**	3.69
24	Min	**0.91**	0.93	1.16	**1.09**
	Avg	**1.15**	1.16	**2.06**	2.19
	Max	1.72	**1.60**	**3.49**	4.08
32	Min	1.05	**1.04**	1.33	**1.26**
	Avg	1.51	**1.49**	**2.24**	2.41
	Max	**2.52**	2.63	**3.71**	4.51

Conclusions and Further Work

- **Batched scheduling** can be an **useful strategy** in tabled logic programs that:
  - require an **eager propagation** of answers.
  - do not require the complete set of answers to be found (**partial evaluation** of the **search space**).

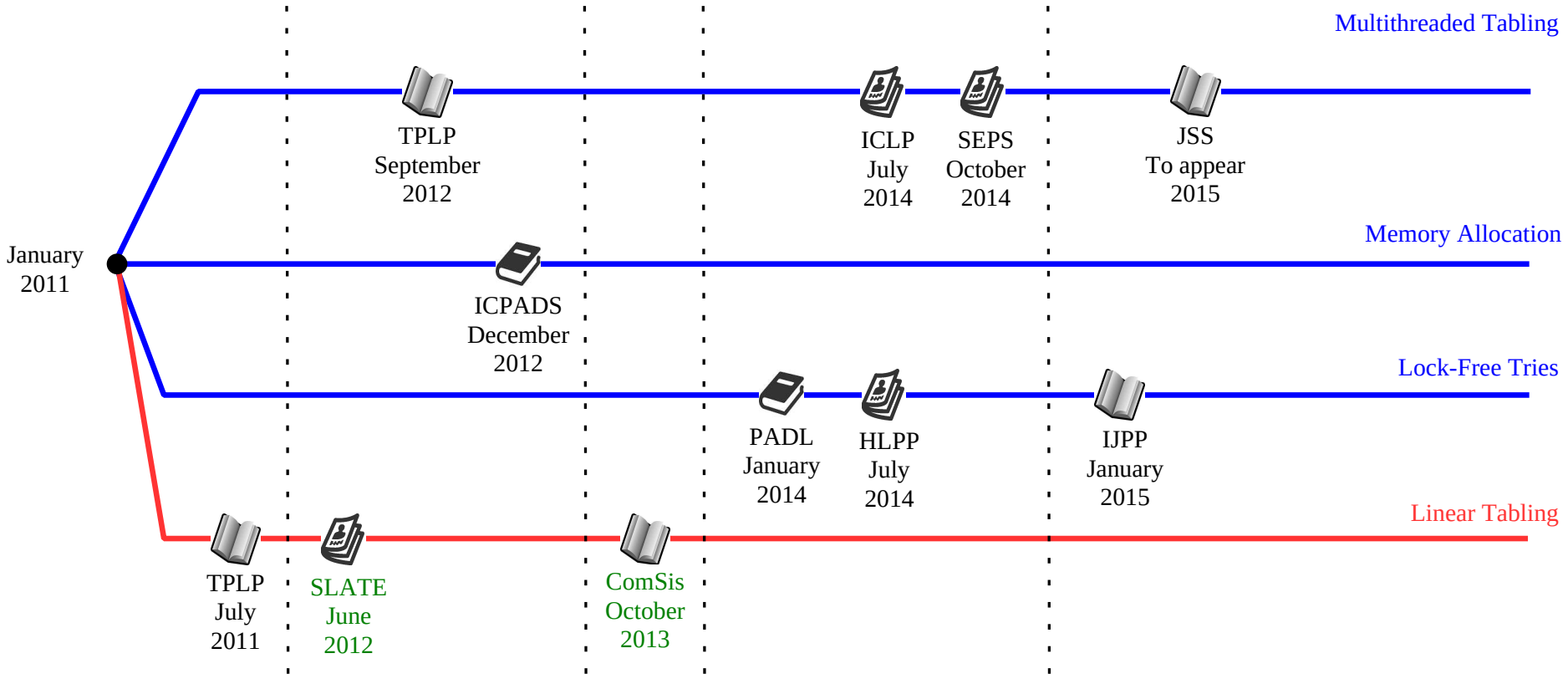
Conclusions and Further Work

- **Batched scheduling** can be an **useful strategy** in tabled logic programs that:
  - require an **eager propagation** of answers.
  - do not require the complete set of answers to be found (**partial evaluation** of the **search space**).
- We have presented the **PAC (Private Answer Chaining)** strategy that **enables** the **combination** of **batched scheduling** with **Full-Sharing**.
  - Split's **answers representation (public)** and **answer propagation (private)**.
- Experimental results showed that the **PAC** strategy is a **good first approach**, however there is still **room for improvement**.

Conclusions and Further Work

- **Batched scheduling** can be an **useful strategy** in tabled logic programs that:
  - require an **eager propagation** of answers.
  - do not require the complete set of answers to be found (**partial evaluation** of the **search space**).
- We have presented the **PAC (Private Answer Chaining)** strategy that **enables** the **combination** of **batched scheduling** with **Full-Sharing**.
  - Split's **answers representation (public)** and **answer propagation (private)**.
- Experimental results showed that the **PAC** strategy is a **good first approach**, however there is still **room for improvement**.
- Further work will include:
  - Use **timestamped tries** to **avoid** the search for the already propagated answers.
  - Seek for **new real-world problems** that allow us to improve and consolidate our framework.

Research Outline



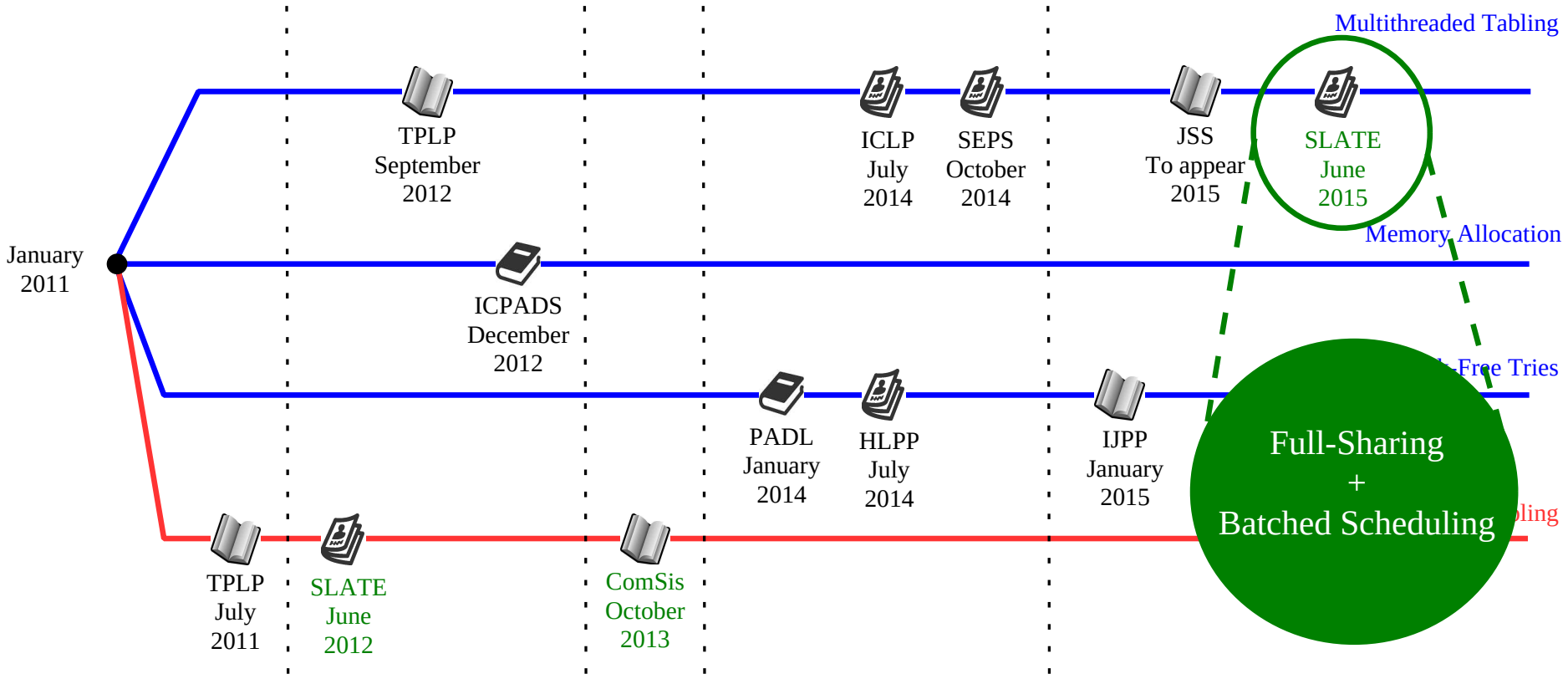
Work related:

- 3 Journals
- 2 Book Series
- 3 Workshop Proceedings

Others:

- 2 Journals
- 0 Book Series
- 1 Workshop Proceeding

Research Outline



Thank You !!!


Scheduling
Tabling
Concurrency
Worst-Case-Scenarios
Subgoal-Sharing
State-Of-The-Art
Private-Answer-Chaining
Subgoal-Tries
Dynamic-Programming
Lock-Free-Hash-Tries
Algorithms
Memory-Allocation
Local-Scheduling
Prolog
Dependency
Synchronization
Logic-Programming
Lock-Freedom
Batched-Scheduling
Posix-Compliant
Progress
Threads
Answer-Tries
Subgoal-Frames
Yap-Prolog
Full-Sharing
No-Sharing
Parallelization
Tabled-Logic-Programming
Declarative-Programming
Sub-Computations
Multi-Threading