Towards Multi-Threaded Local Tabling Using a Common Table Space

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*

ICLP, Budapest - Hungary, September 2012

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)

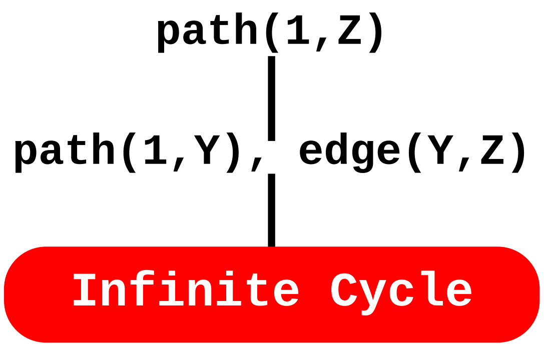
SLD Resolution: Infinite Cycles

```
path(X,Z) :- path(X,Y), edge(Y,Z).
path(X,Z) :- edge(X,Z).

edge(1,2).
edge(2,1).
```

```
path(1,Z)
```

SLD Resolution: Infinite Cycles

```
path(X,Z) :- path(X,Y), edge(Y,Z).
path(X,Z) :- edge(X,Z).

edge(1,2).
edge(2,1).
```



SLD Resolution: Infinite Cycles

```
path(X,Z) :- edge(X,Y), path(Y,Z).
path(X,Z) :- edge(X,Z).

edge(1,2).
edge(2,1).
```

```
path(1,Z)
```

SLD Resolution: Infinite Cycles

```
path(X,Z) :- edge(X,Y), path(Y,Z).
path(X,Z) :- edge(X,Z).

edge(1,2).
edge(2,1).
```

```
          path(1,Z)
              |
edge(1,Y), path(Y,Z)
              |
          path(2,Z)
              |
edge(2,Y), path(Y,Z)
              |
          path(1,Z)
              |
       Infinite Cycle
```

Tabling in Logic Programming

- **Tabling** is an implementation technique that overcomes some of the limitations of SLD resolution.
  - Tabled subgoals are evaluated by storing their answers in an appropriate data space, called the **table space**.
  - Variant 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 Logic Programming

- **Tabling** is an implementation technique that overcomes some of the limitations of SLD resolution.
  - Tabled subgoals are evaluated by storing their answers in an appropriate data space, called the **table space**.
  - Variant 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 and Ciao Prolog.

Tabling in Logic Programming

- **Tabling** is an implementation technique that overcomes some of the limitations of SLD resolution.
  - Tabled subgoals are evaluated by storing their answers in an appropriate data space, called the **table space**.
  - Variant 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 and Ciao Prolog.

- The two most successful tabling strategies are **batched evaluation** and **local evaluation**. This work is focused on **local evaluation**, i.e., the evaluation **fails**, whenever a **new** or a **repeated answer** is **found**.

Table Space - Overview

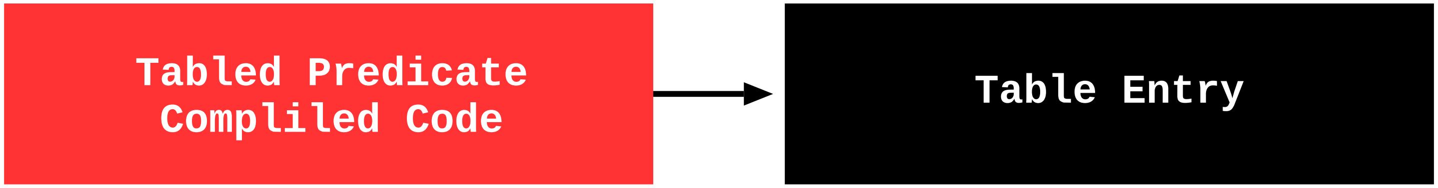


Table Space - Overview

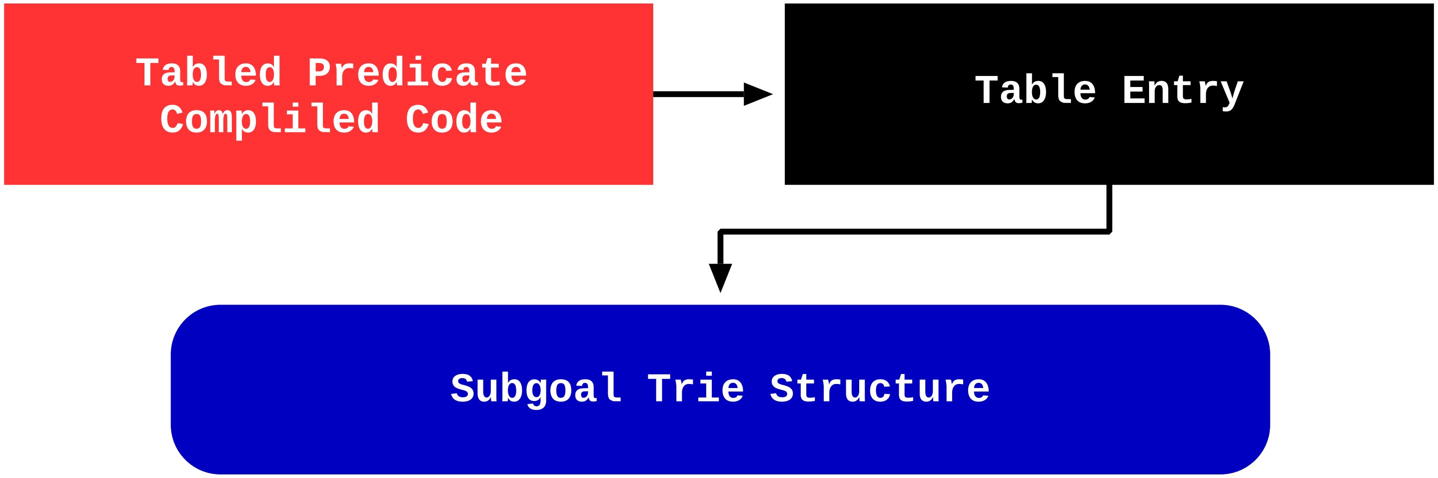


Table Space - Overview

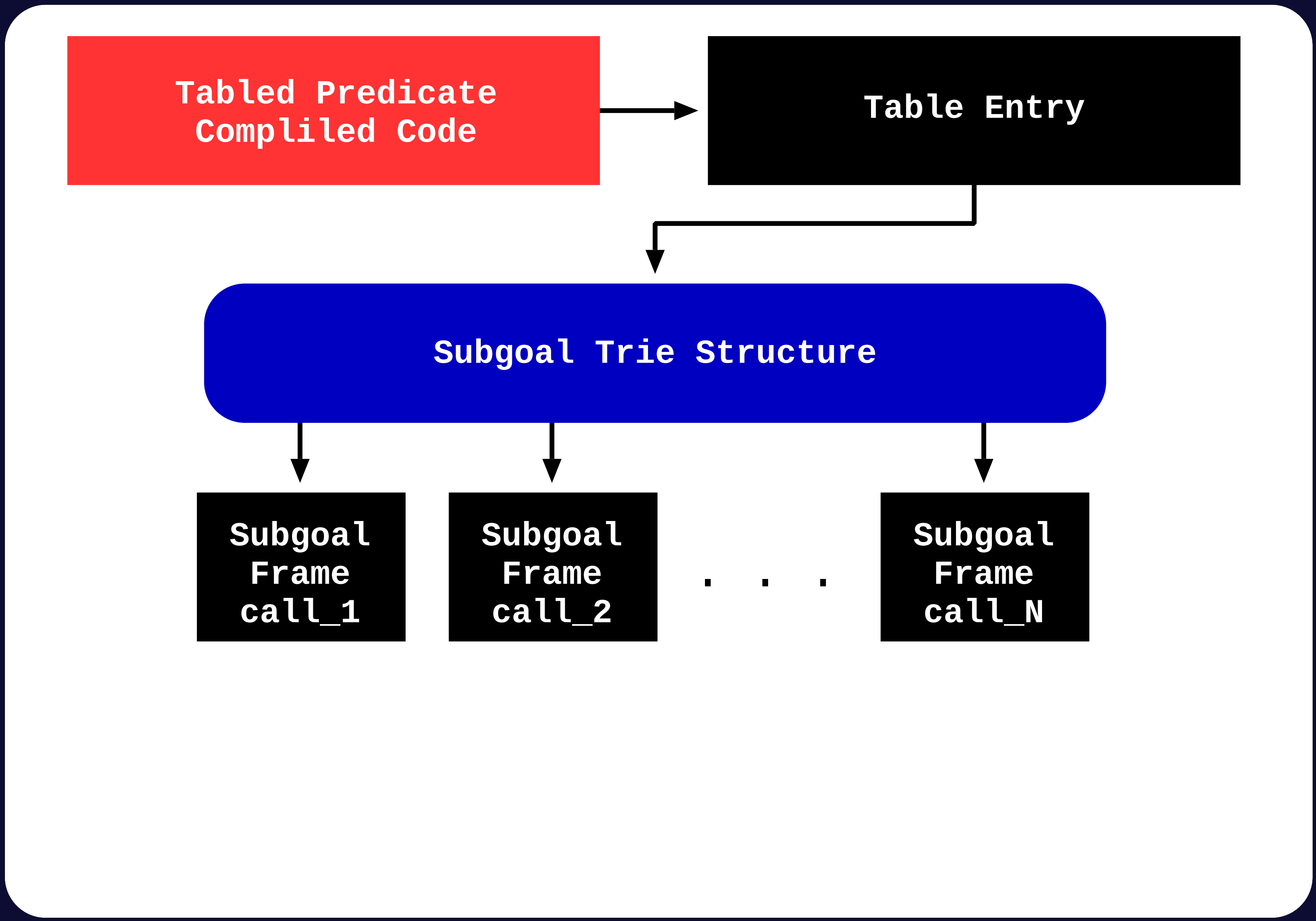


Table Space - Overview

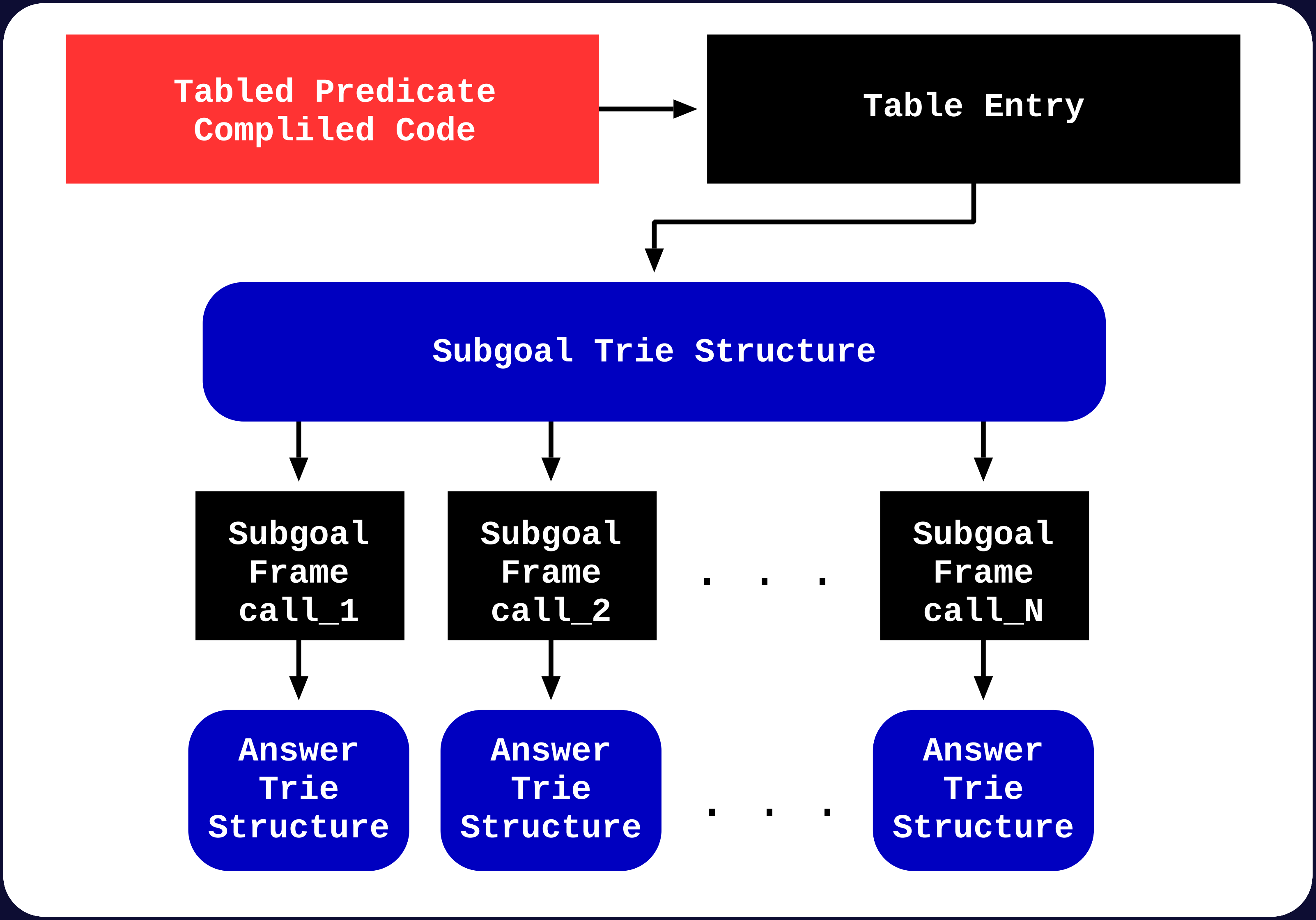


Table Space - Example

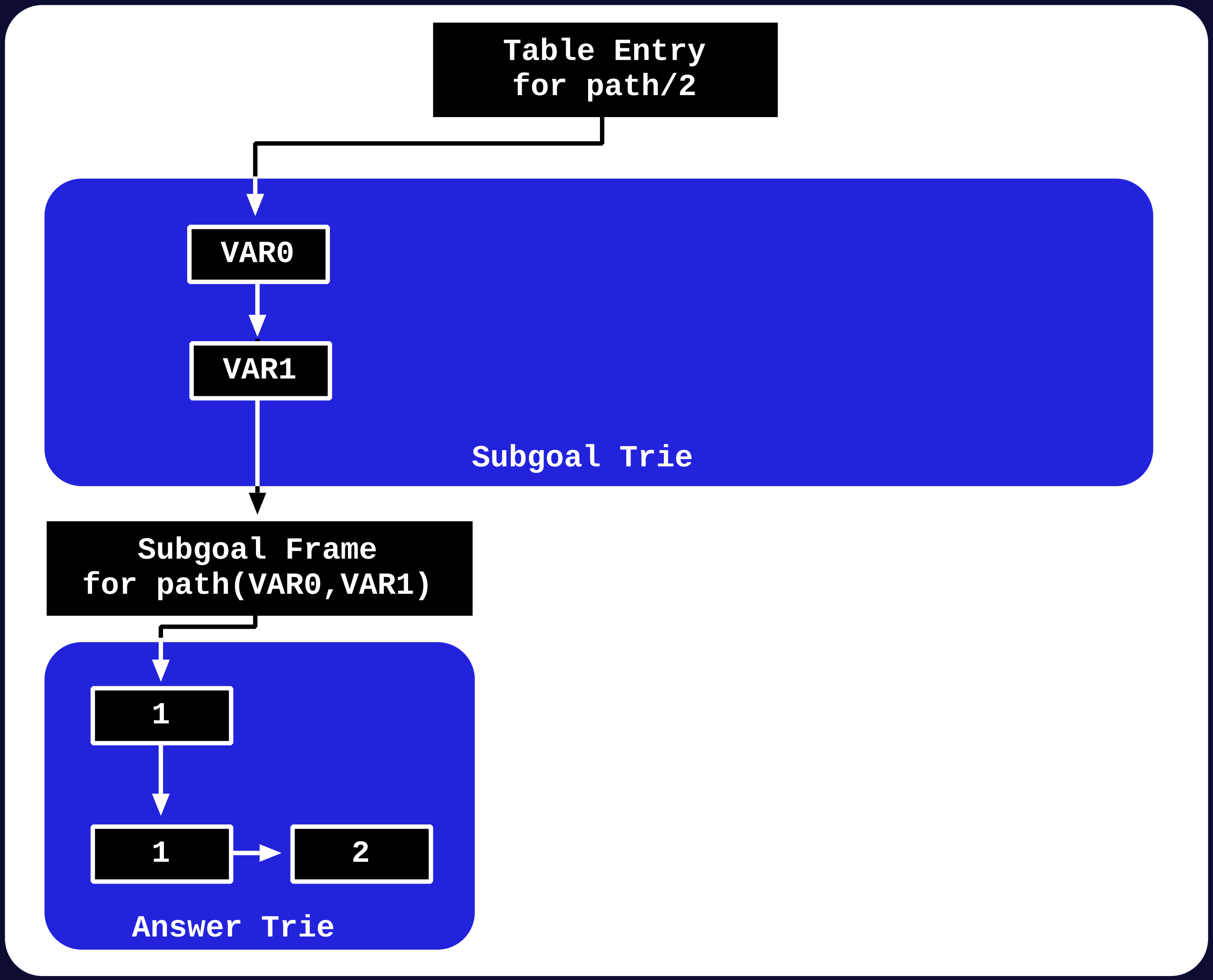
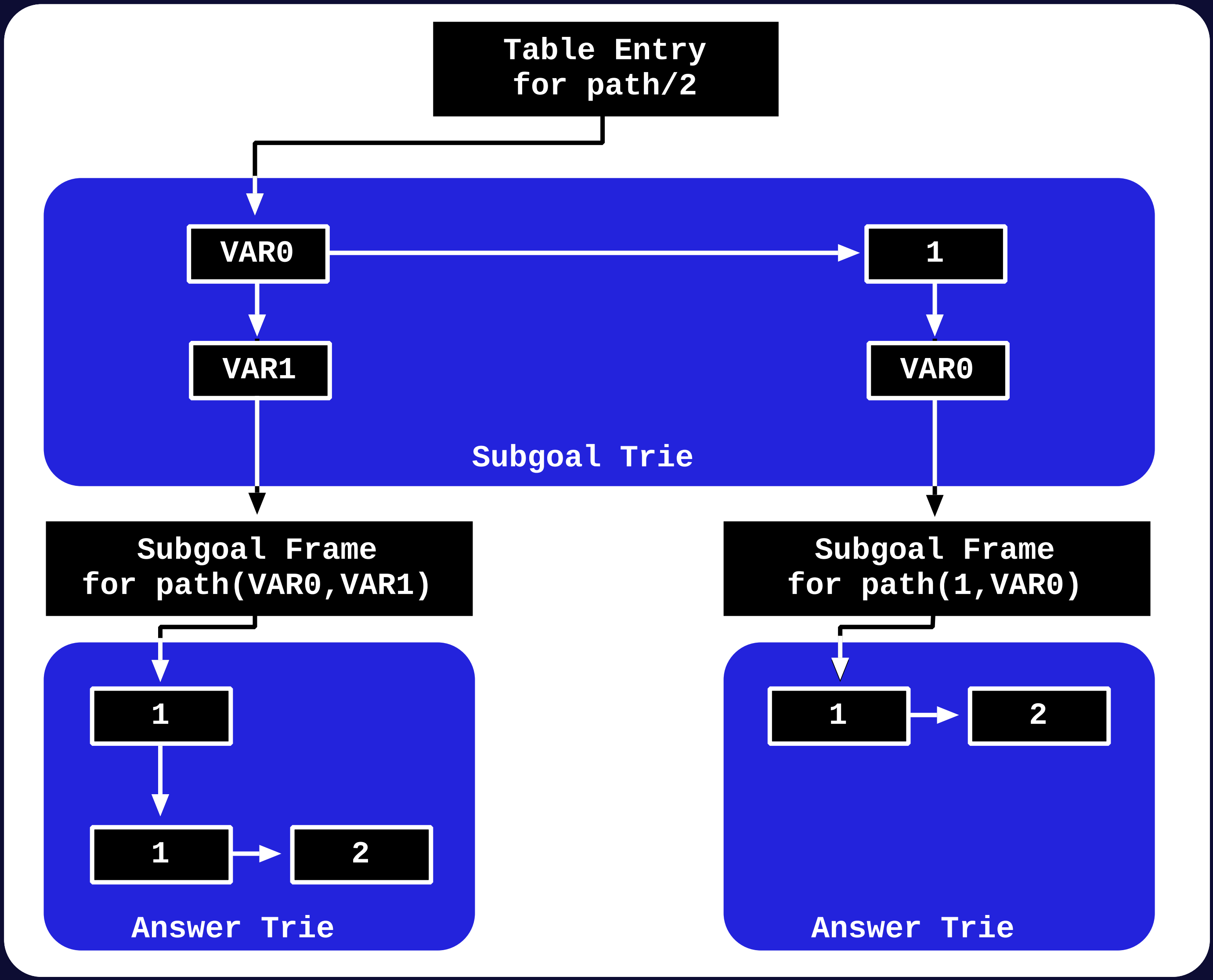


Table Space - Example



Multi-Threaded Tabling

- **Multi-threading** is currently supported by several well-known Prolog, but until now, XSB was the only Prolog system that was able to **combine** multi-threading with tabling.

Multi-Threaded Tabling

- **Multi-threading** is currently supported by several well-known Prolog, but until now, XSB was the only Prolog system that was able to **combine** multi-threading with tabling.
- In XSB, tables may be either **shared** or **private** between threads.
- For **shared** tables, XSB uses a semi-naive approach that, when a set of subgoals computed by different threads is mutually dependent, then a **usurpation algorithm** synchronizes threads and a single thread assumes the computation of all subgoals, turning the remaining threads into consumer threads.

Multi-Threaded Tabling

- **Multi-threading** is currently supported by several well-known Prolog, but until now, XSB was the only Prolog system that was able to **combine** multi-threading with tabling.
- In XSB, tables may be either **shared** or **private** between threads.
- For **shared** tables, XSB uses a semi-naive approach that, when a set of subgoals computed by different threads is mutually dependent, then a **usurpation algorithm** synchronizes threads and a single thread assumes the computation of all subgoals, turning the remaining threads into consumer threads.
- For **private** tables, XSB allocates a private table for each subgoal called and the table is only available for the calling thread.

Multi-Threaded Tabling

- **Private** tables have the **advantage** of being easier to implement.
  - Do not have all the associated issues of **locking**, **synchronization** and **potential deadlocks**.
  - **Fix-point detection is easier** because the evaluation of tables does not causes inter-dependency between threads.

Multi-Threaded Tabling

- **Private** tables have the **advantage** of being easier to implement.
  - Do not have all the associated issues of **locking**, **synchronization** and **potential deadlocks**.
  - **Fix-point detection is easier** because the evaluation of tables does not causes inter-dependency between threads.

- **Private** tables have the following **disadvantages** if several threads are calling the same tables:
  - **Memory usage** - stress on the memory allocator causes a degradation of performance;
  - **Waste of CPU resources** - the same table is evaluated by all of the calling threads.

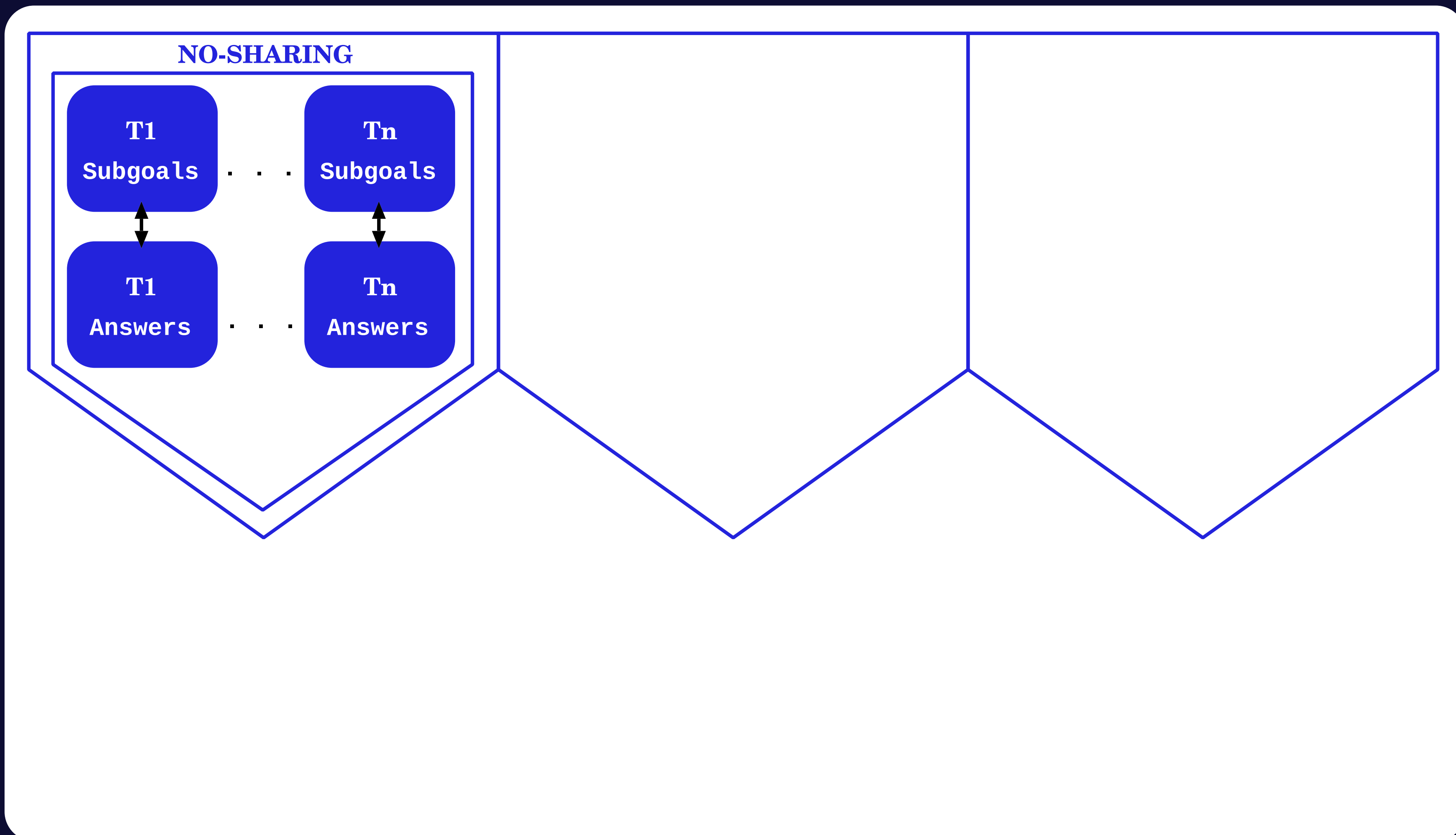
Multi-Threaded Tabling

- **Private** tables have the **advantage** of being easier to implement.
  - Do not have all the associated issues of **locking**, **synchronization** and **potential deadlocks**.
  - **Fix-point detection is easier** because the evaluation of tables does not causes inter-dependency between threads.

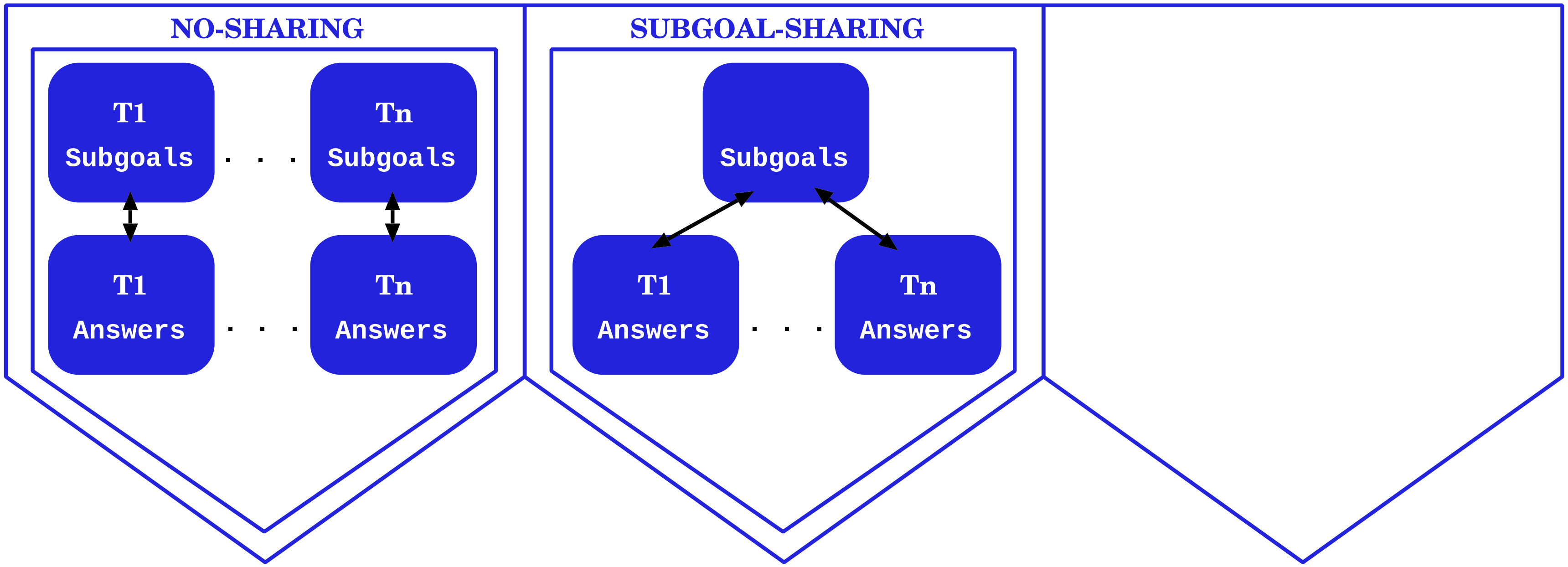
- **Private** tables have the following **disadvantages** if several threads are calling the same tables:
  - **Memory usage** - stress on the memory allocator causes a degradation of performance;
  - **Waste of CPU resources** - the same table is evaluated by all of the calling threads.

- In this work, we propose **an alternative view** to XSB's approach. In our proposal, each thread views its tables as private but, at the engine level, we use a **common table space** where tables are shared among all threads.

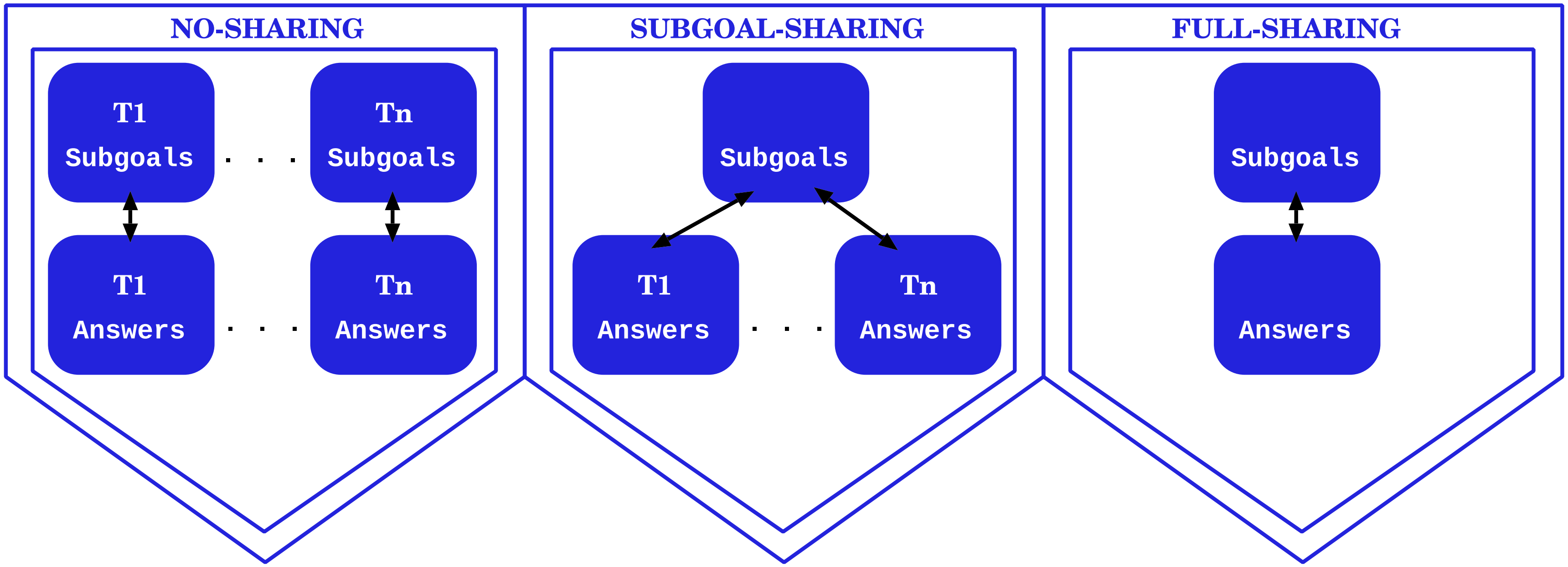
Multi-Threaded Tabling - Our Framework



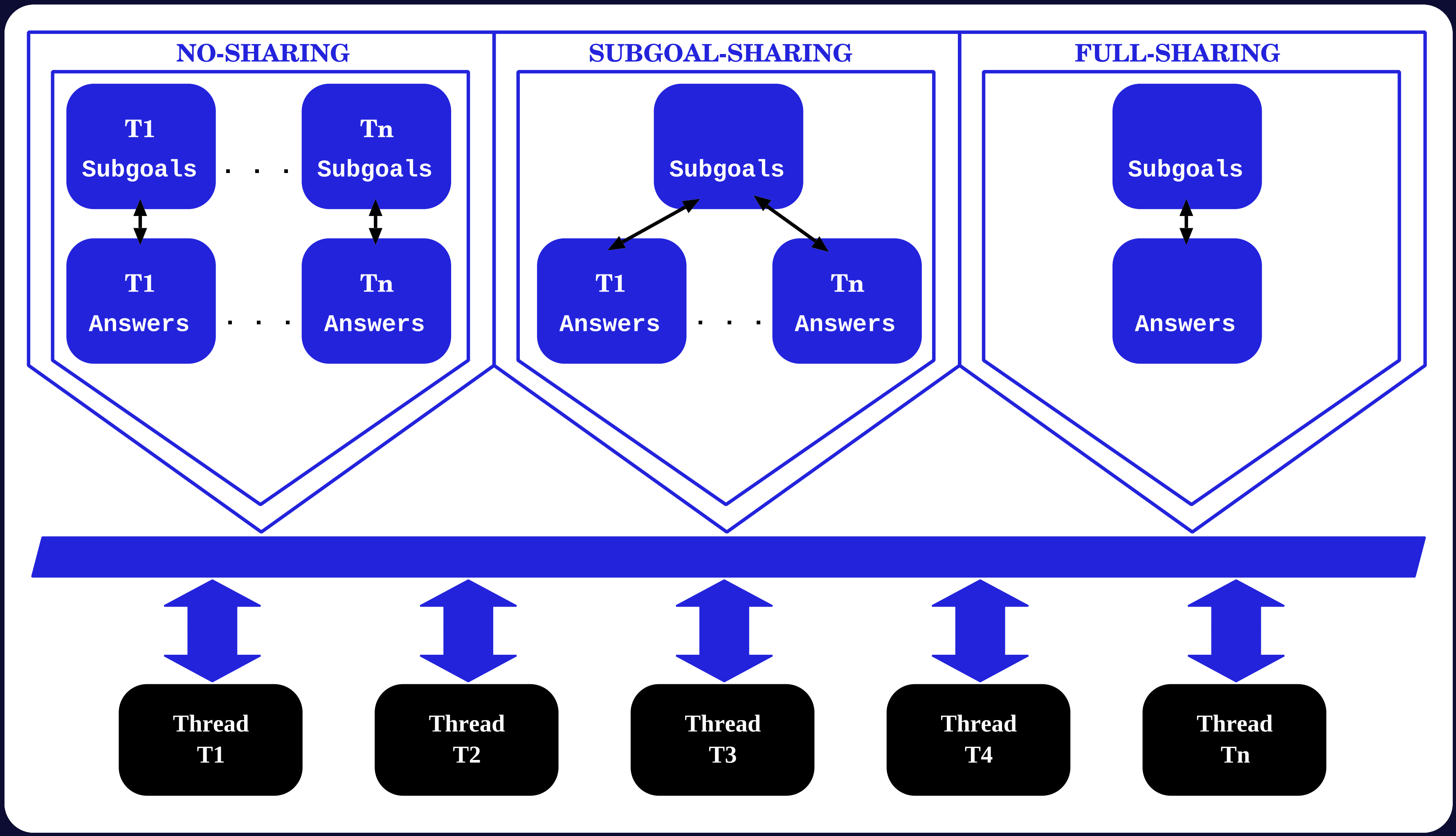
Multi-Threaded Tabling - Our Framework



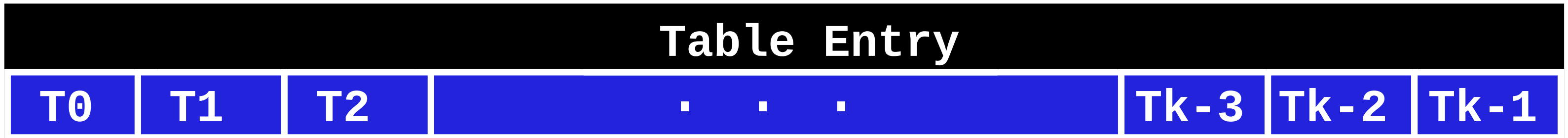
Multi-Threaded Tabling - Our Framework



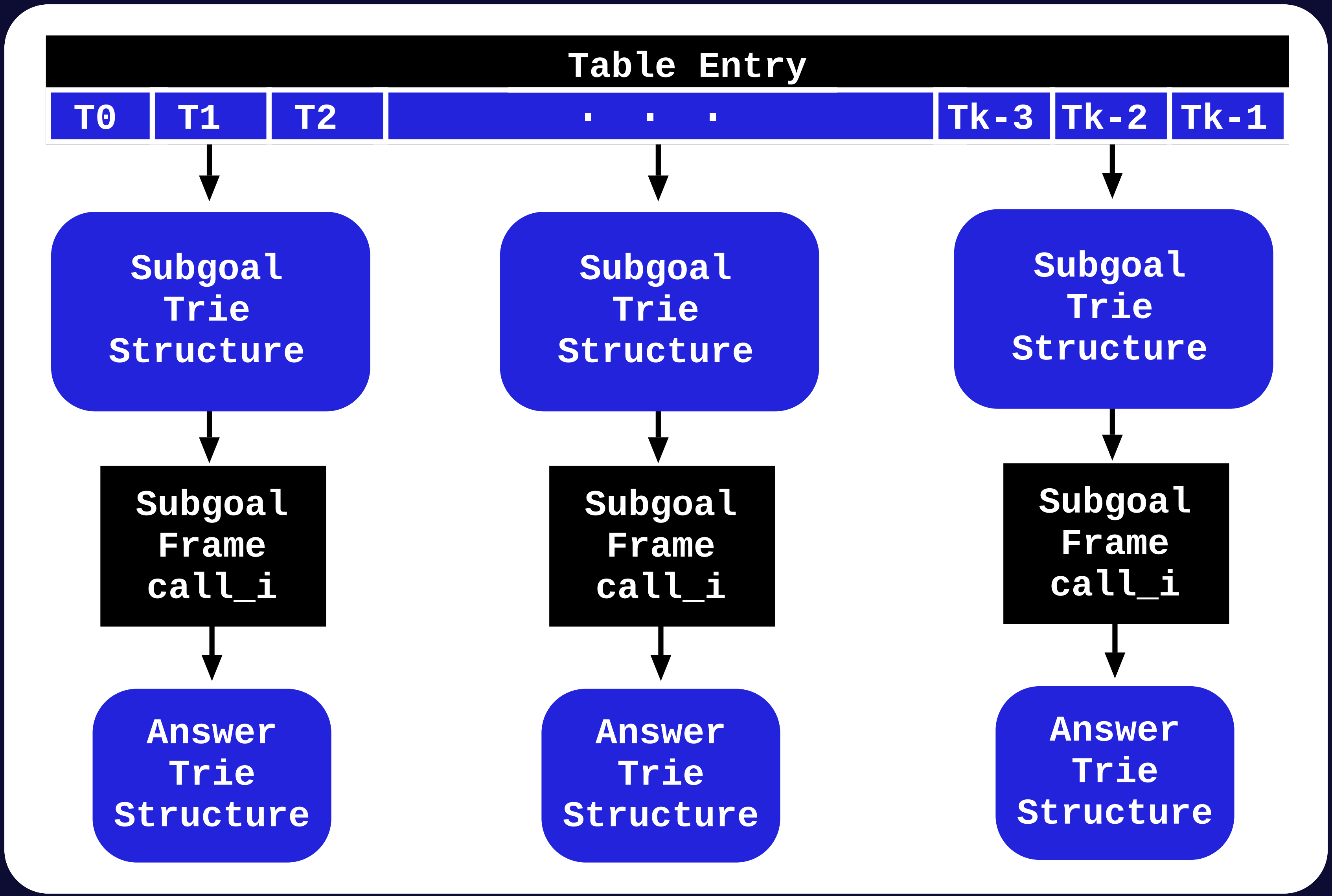
Multi-Threaded Tabling - Our Framework



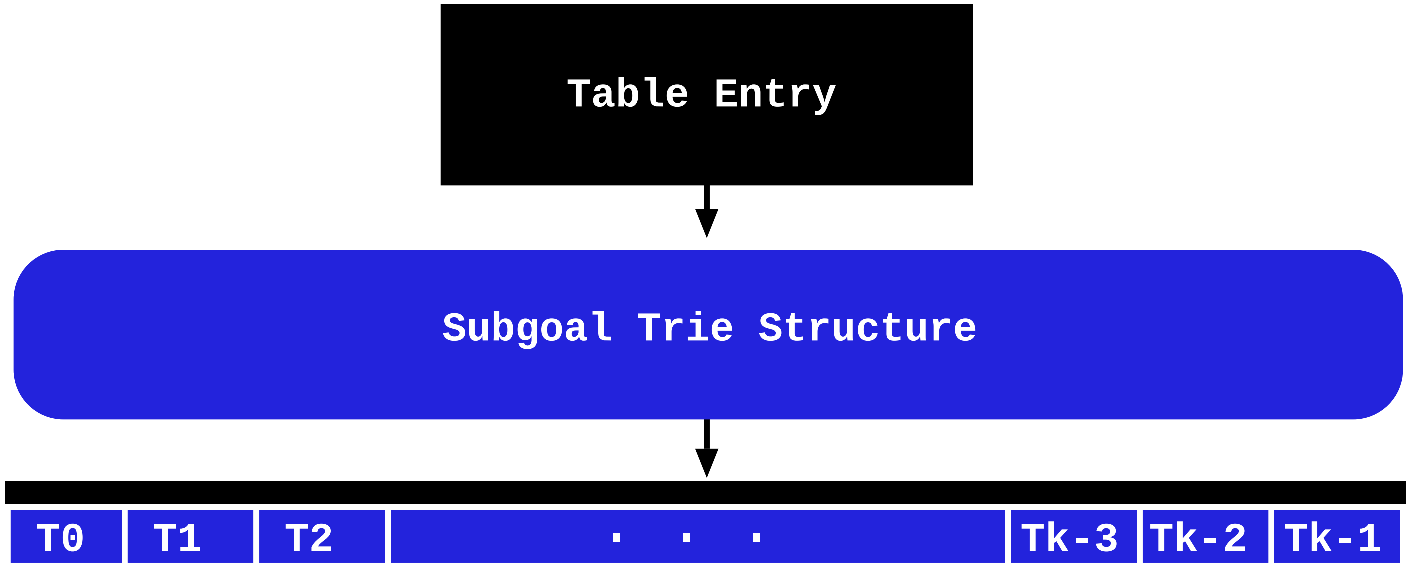
Multi-Threaded Tabling - No Sharing



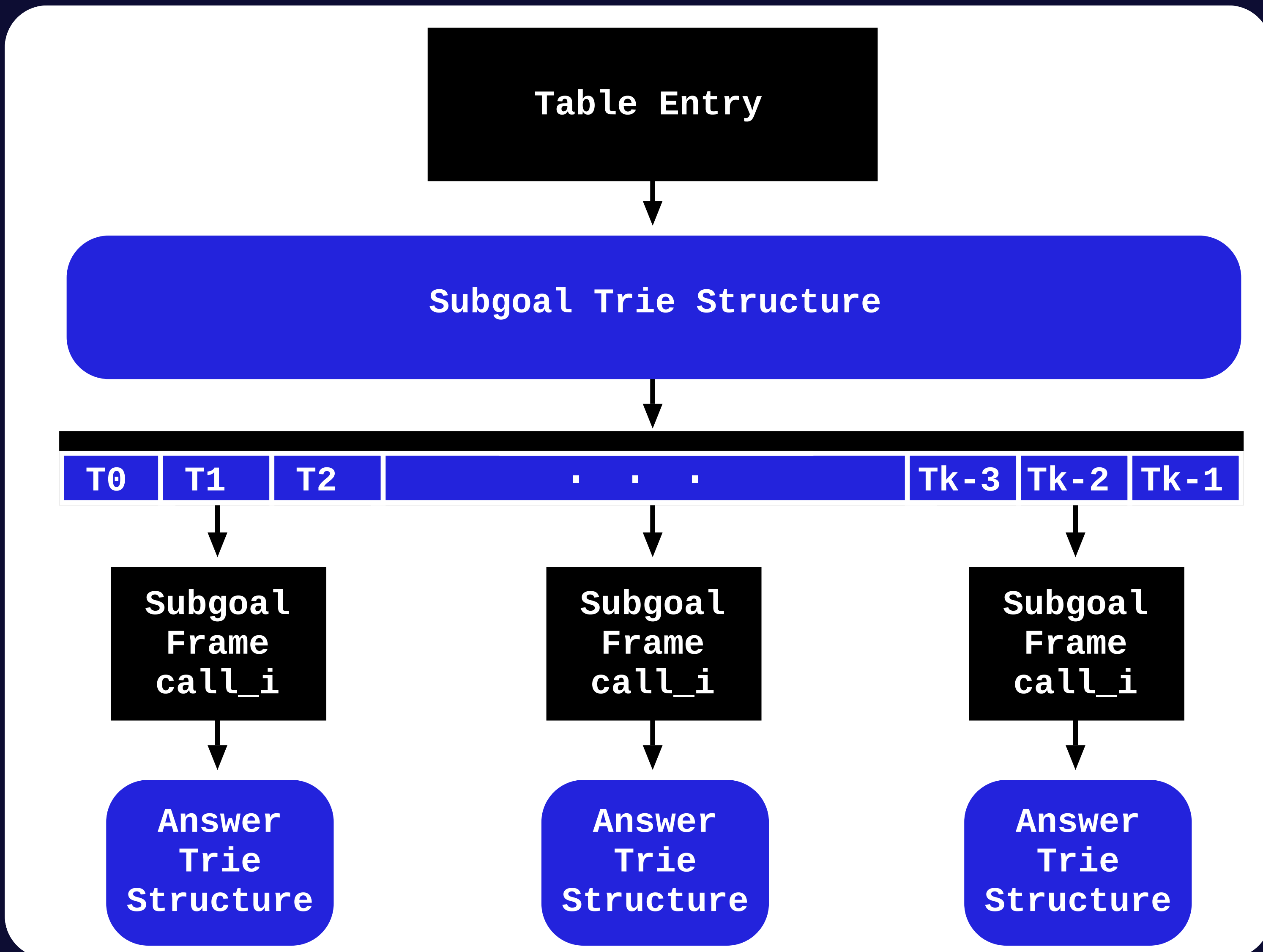
Multi-Threaded Tabling - No Sharing



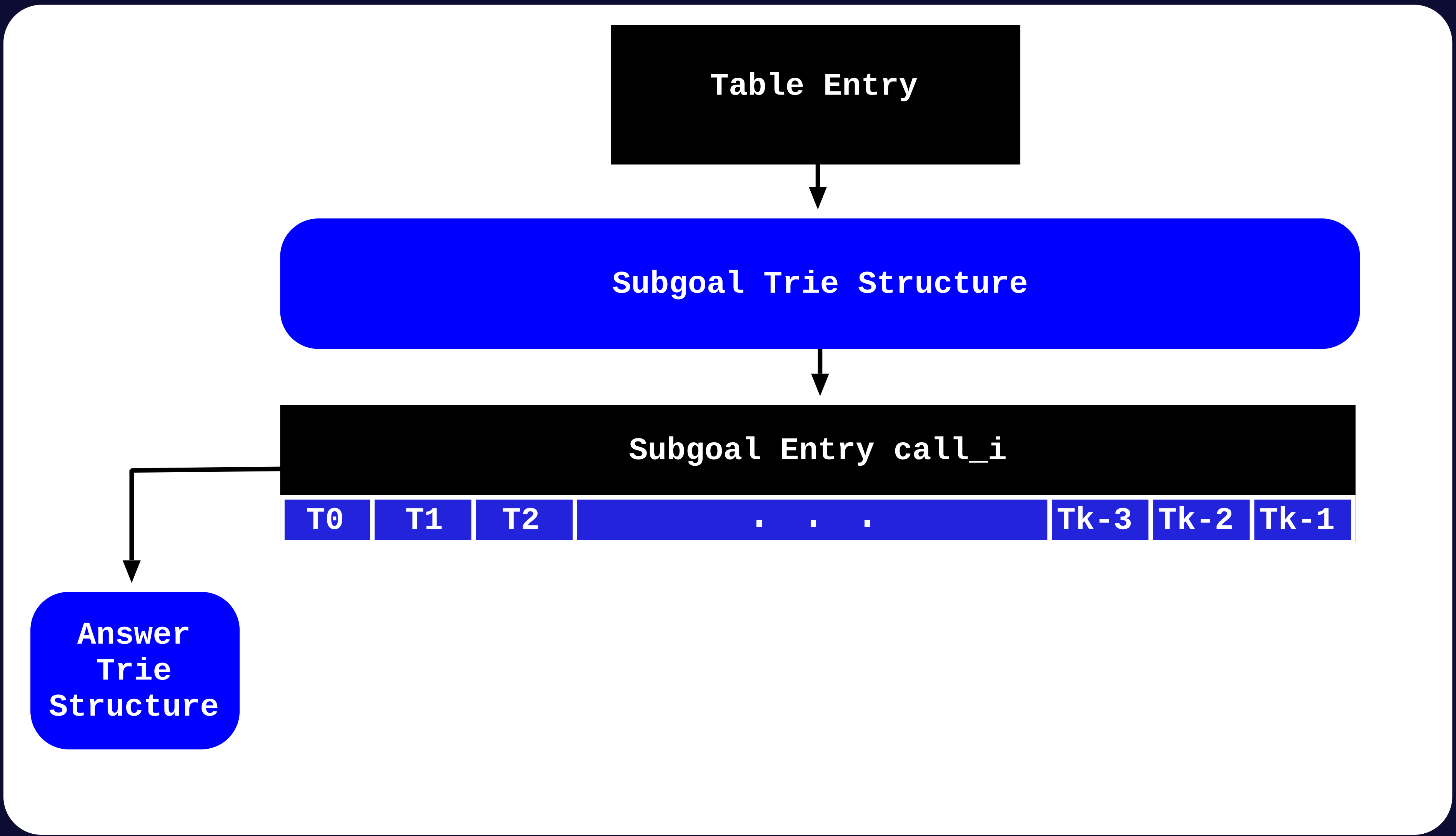
Multi-Threaded Tabling - Subgoal Sharing



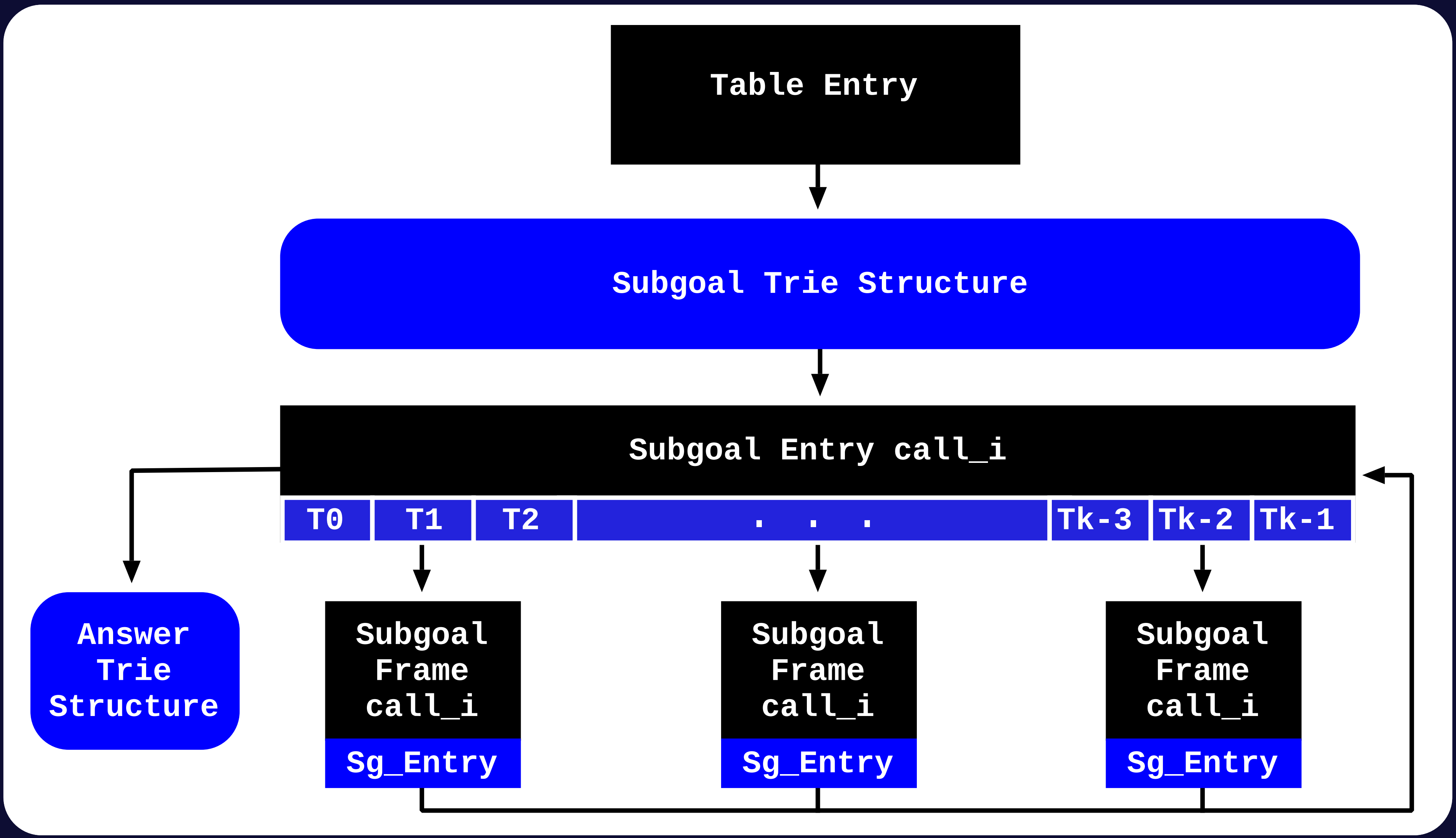
Multi-Threaded Tabling - Subgoal Sharing



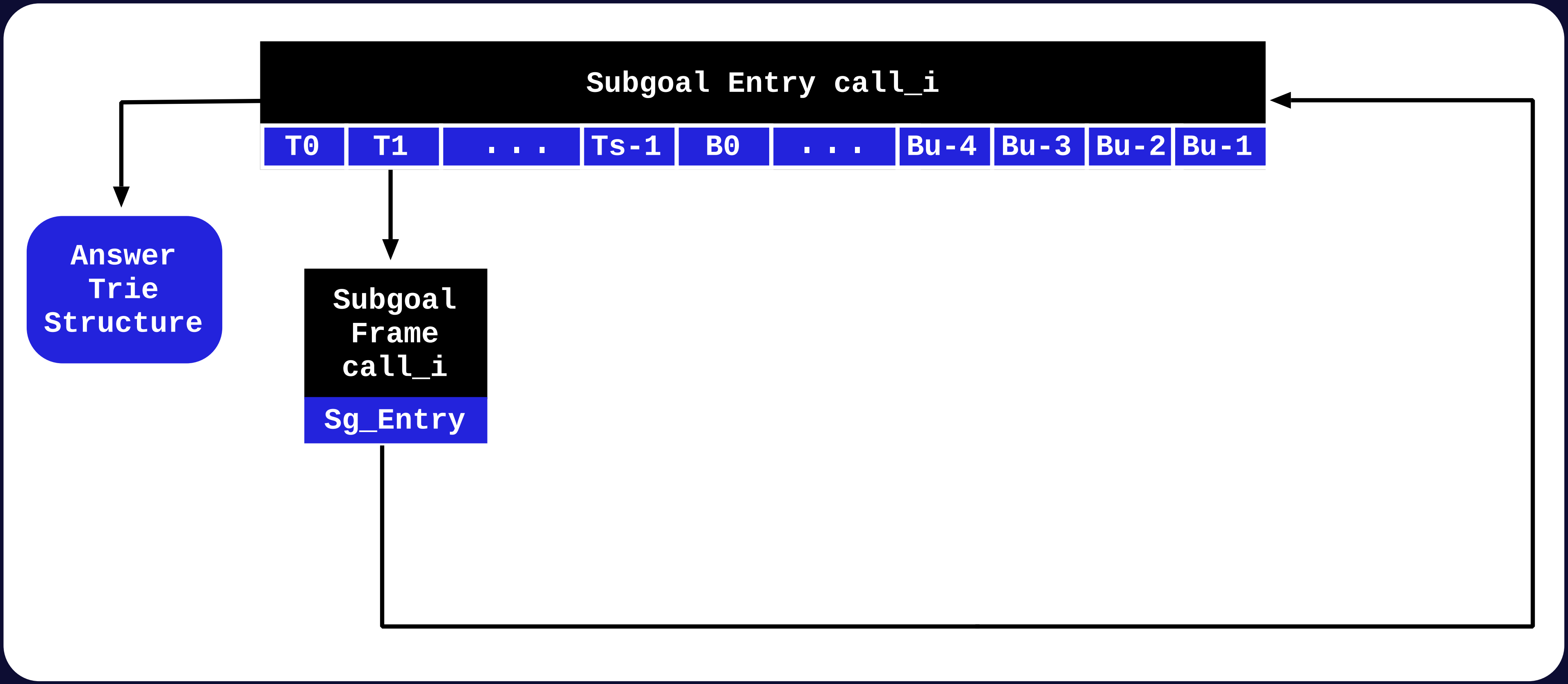
Multi-Threaded Tabling - Full Sharing



Multi-Threaded Tabling - Full Sharing



Implementation Details - Bucket Arrays



Implementation Details - Bucket Arrays

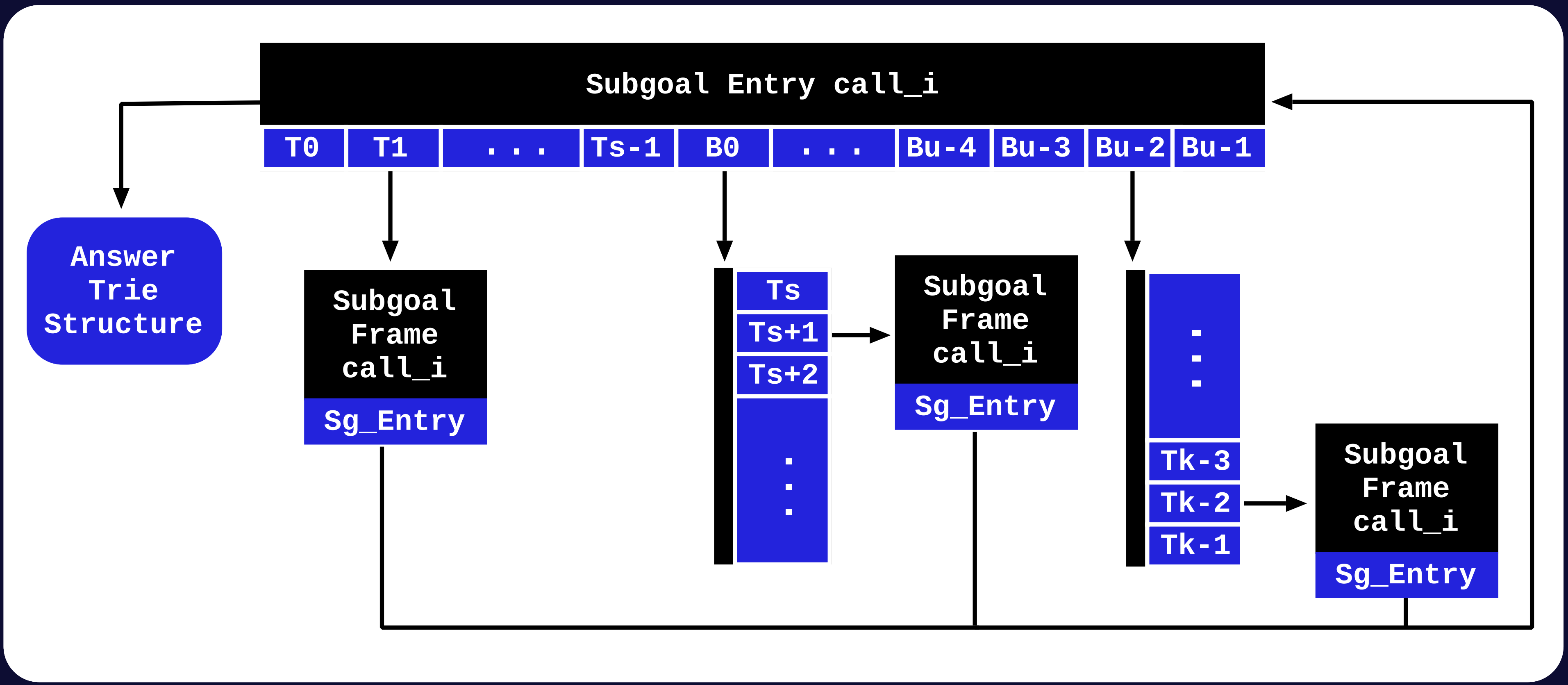


Table Locking Schemes - TLEL vs TLNL/TLWL

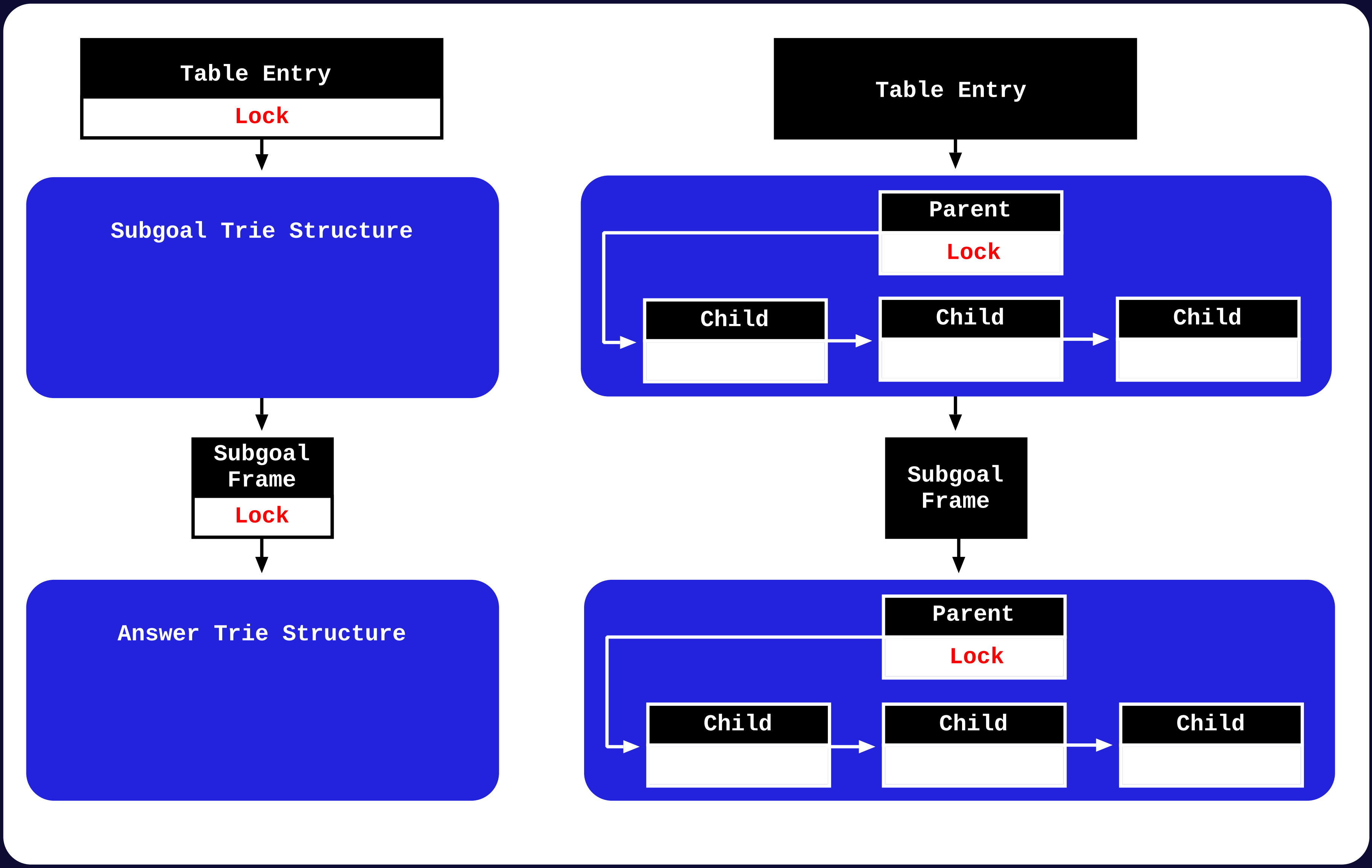


Table Locking Schemes - TLEL vs TLNL/TLWL

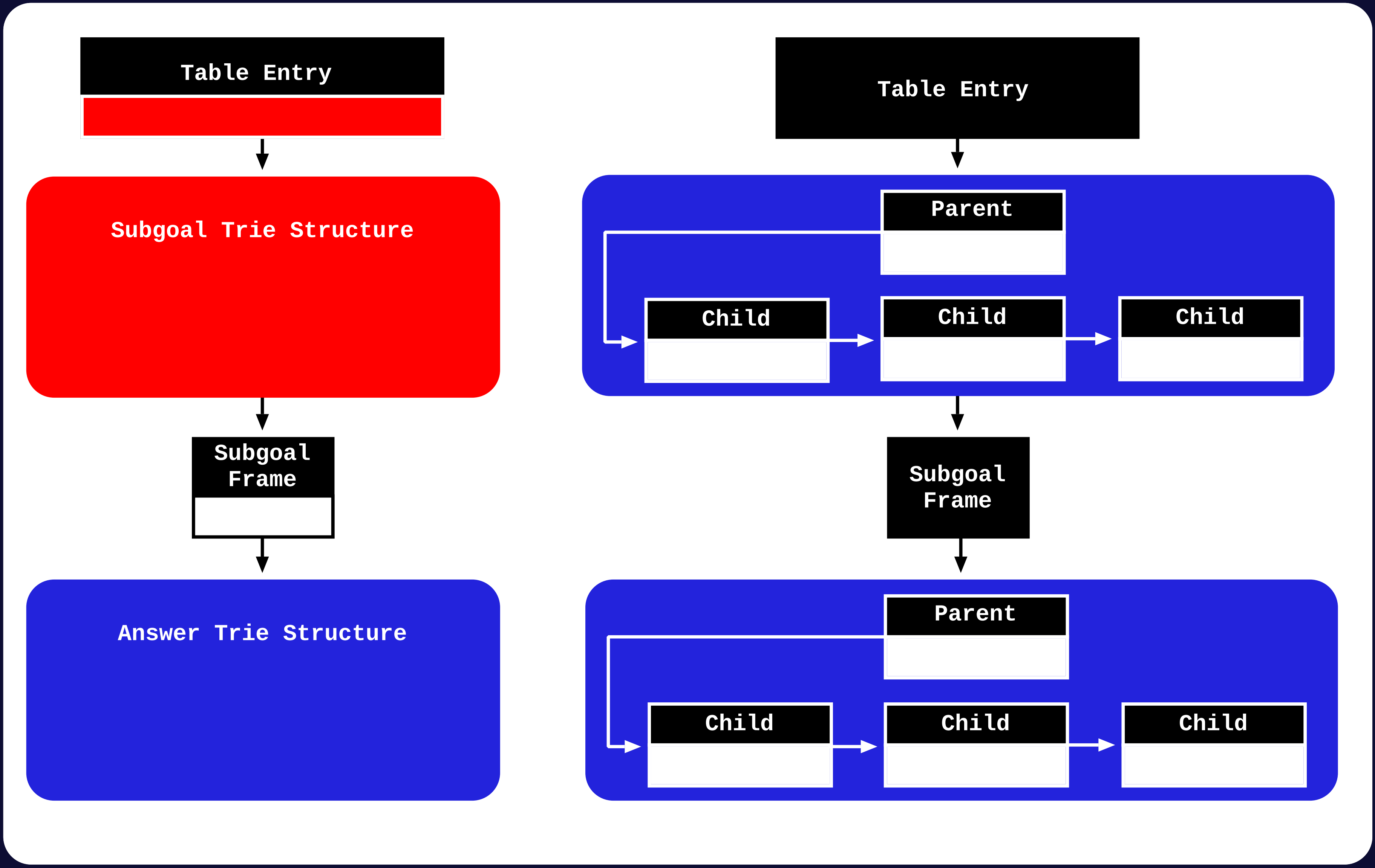


Table Locking Schemes - TLEL vs TLNL/TLWL

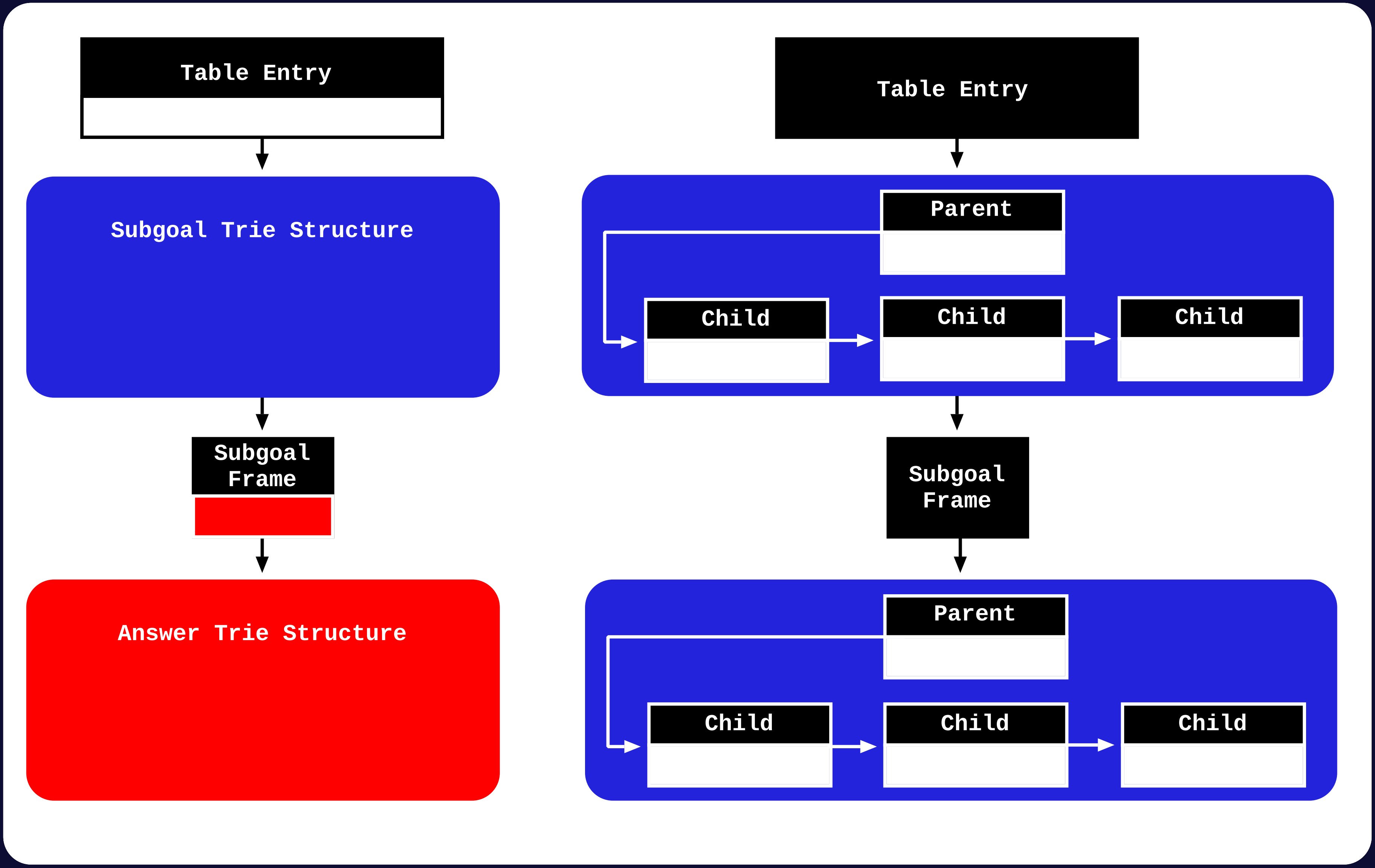


Table Locking Schemes - TLEL vs TLNL/TLWL

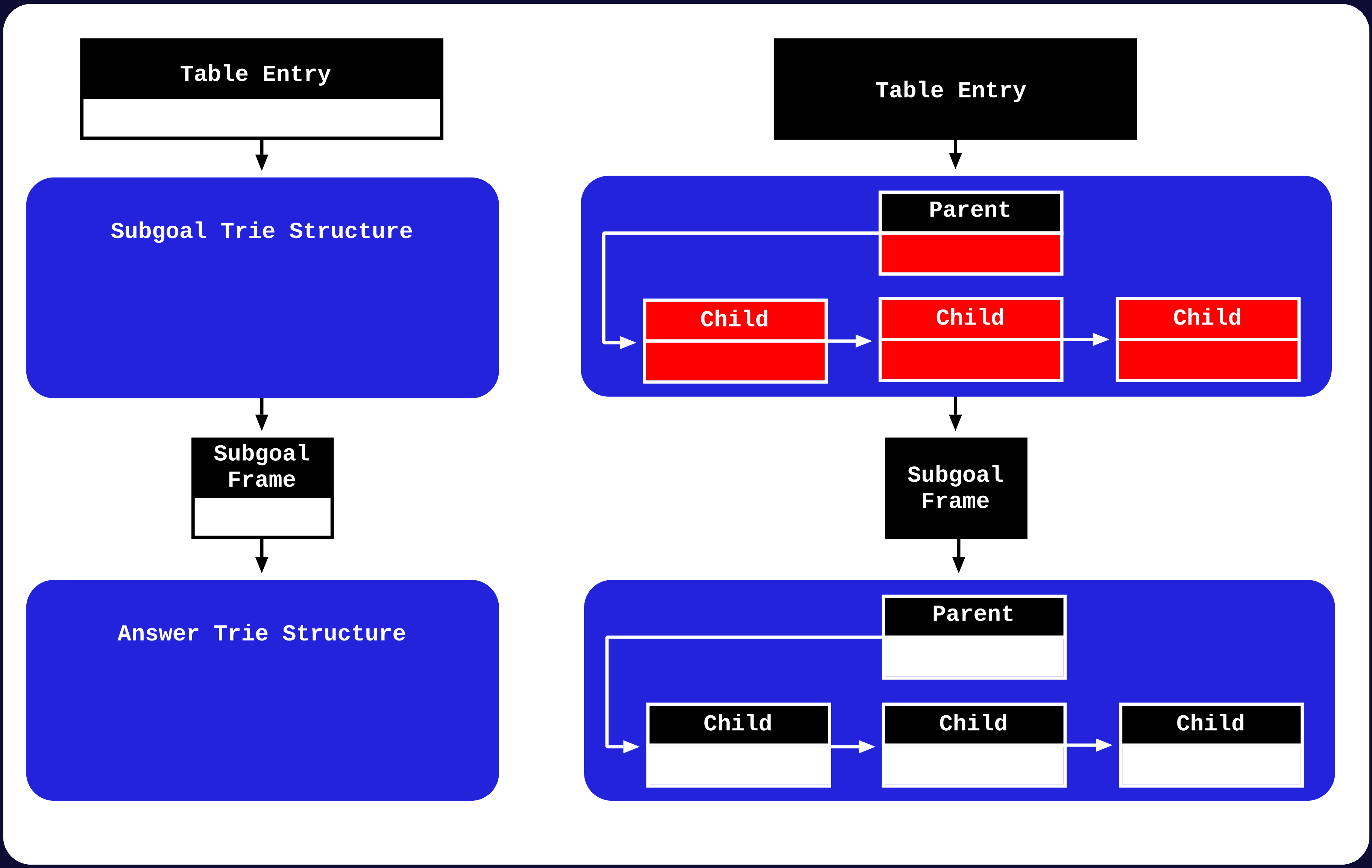


Table Locking Schemes - TLEL vs TLNL/TLWL

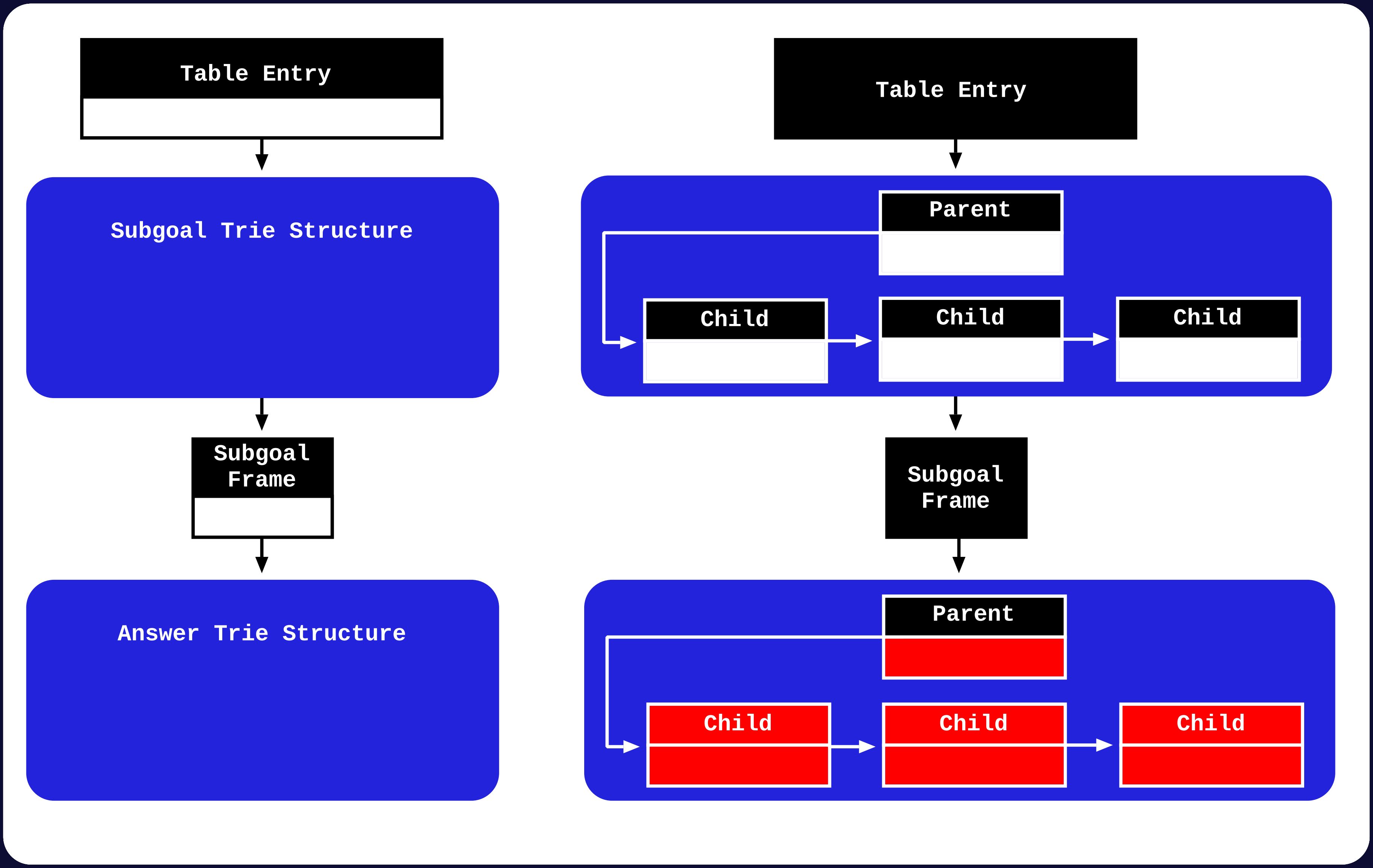


Table Locking Schemes - TLWL

Original Approach

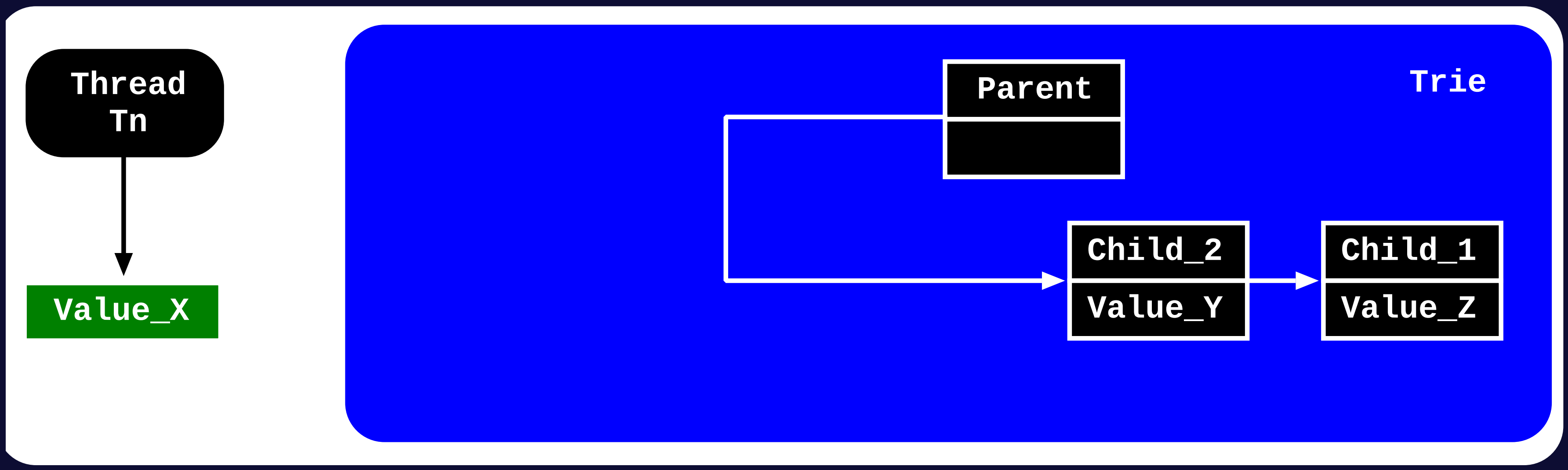


Table Locking Schemes - TLWL

Original Approach

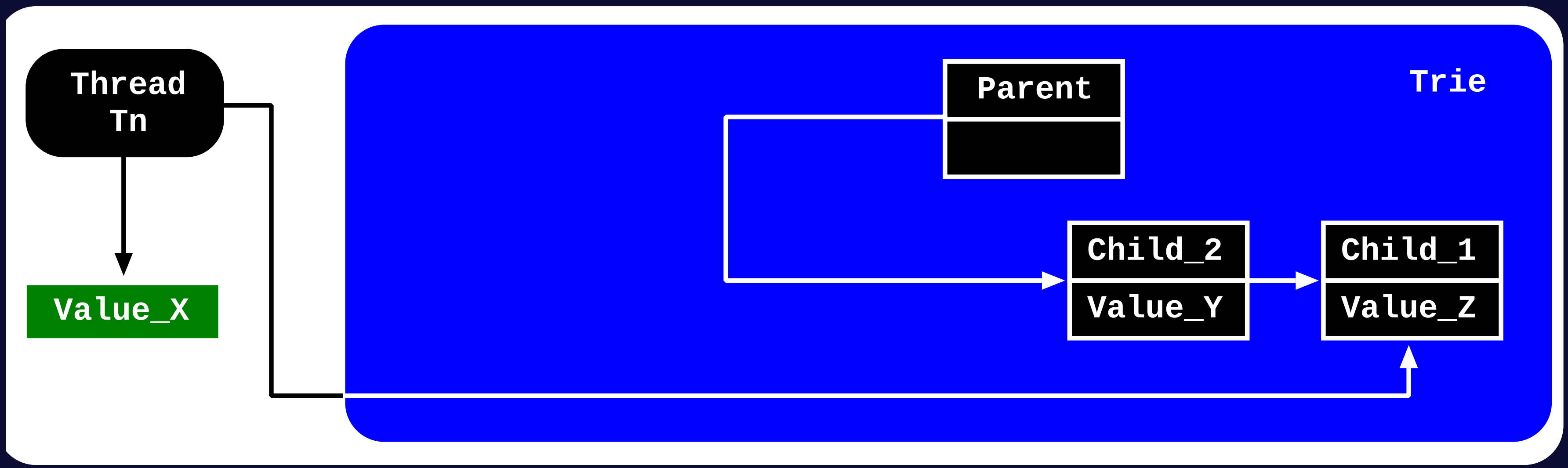


Table Locking Schemes - TLWL

➤ Original Approach

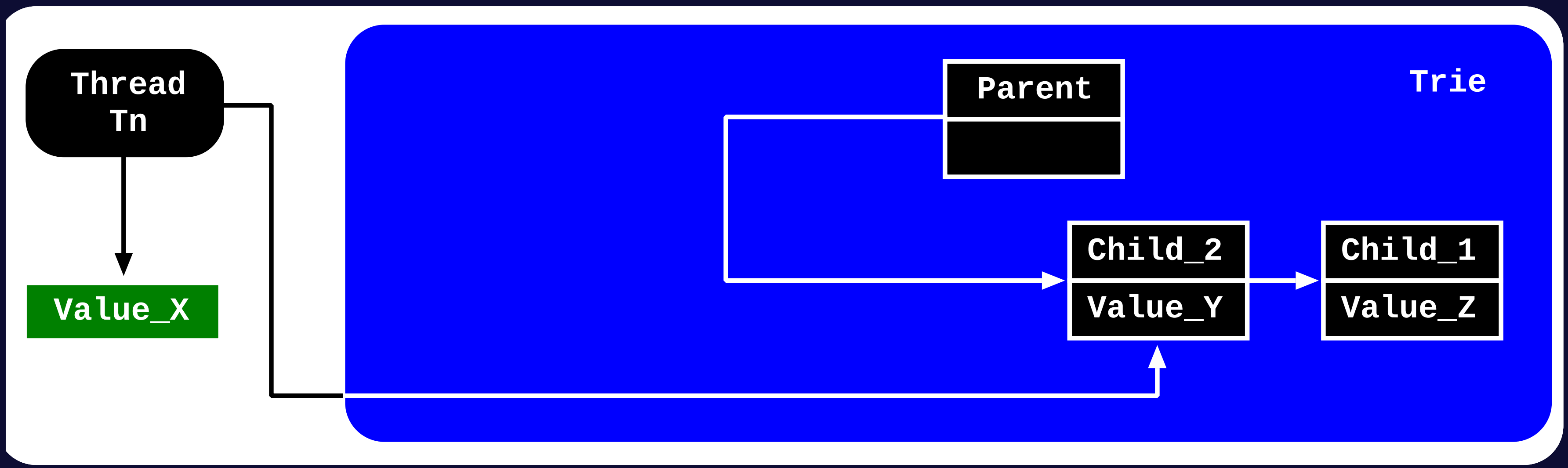


Table Locking Schemes - TLWL

Original Approach

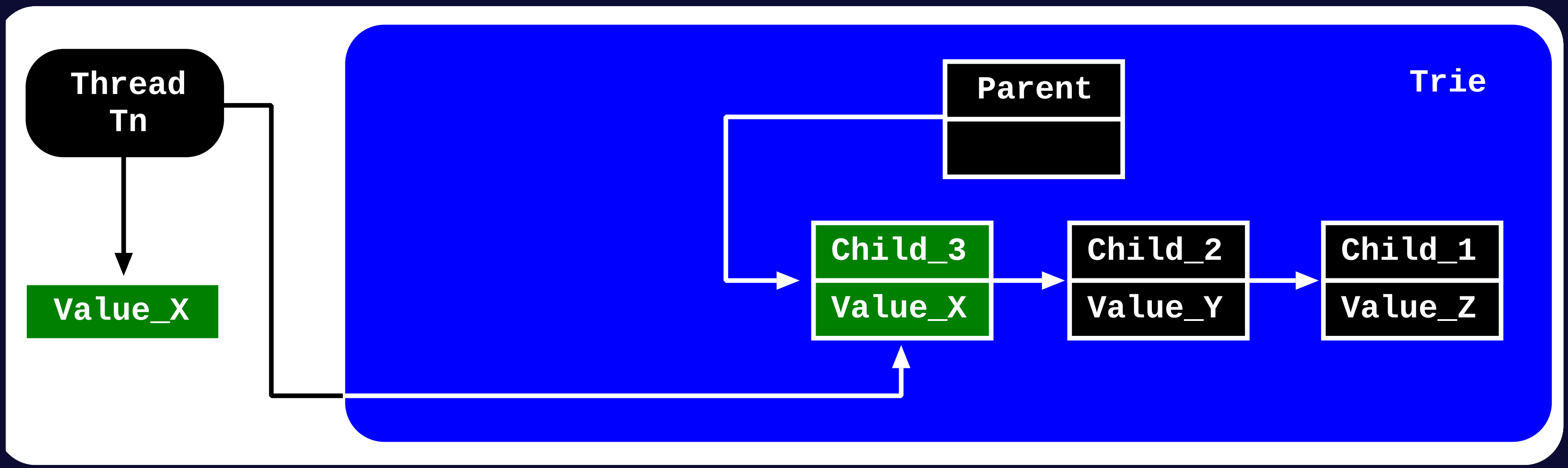


Table Locking Schemes - TLWL

- **Original Approach**

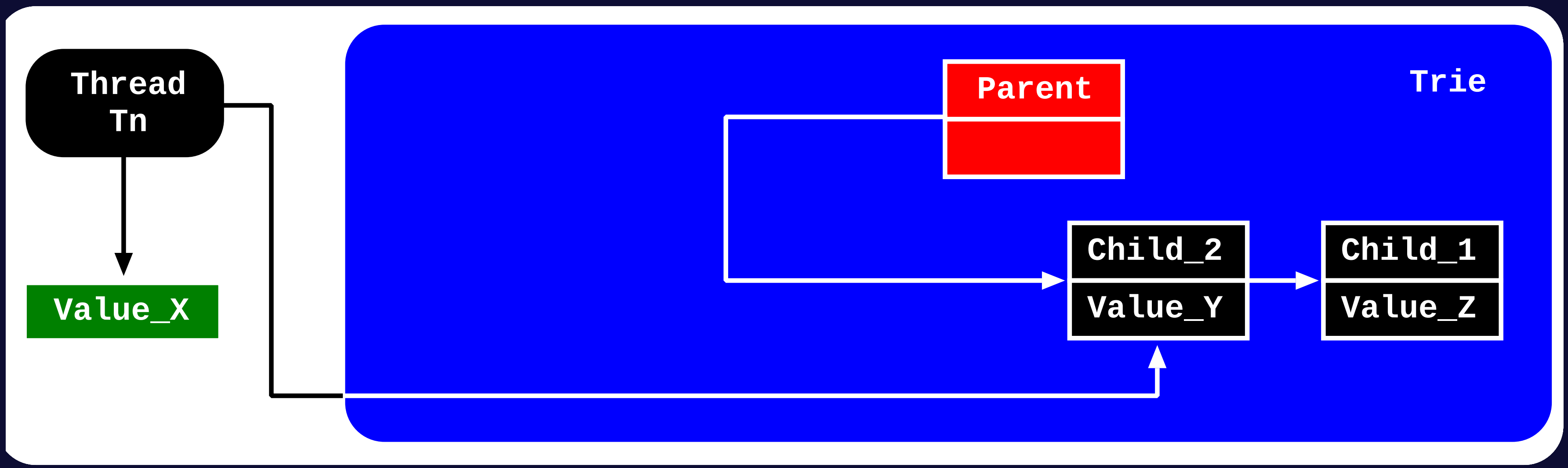


Table Locking Schemes - TLWL

➤ Original Approach

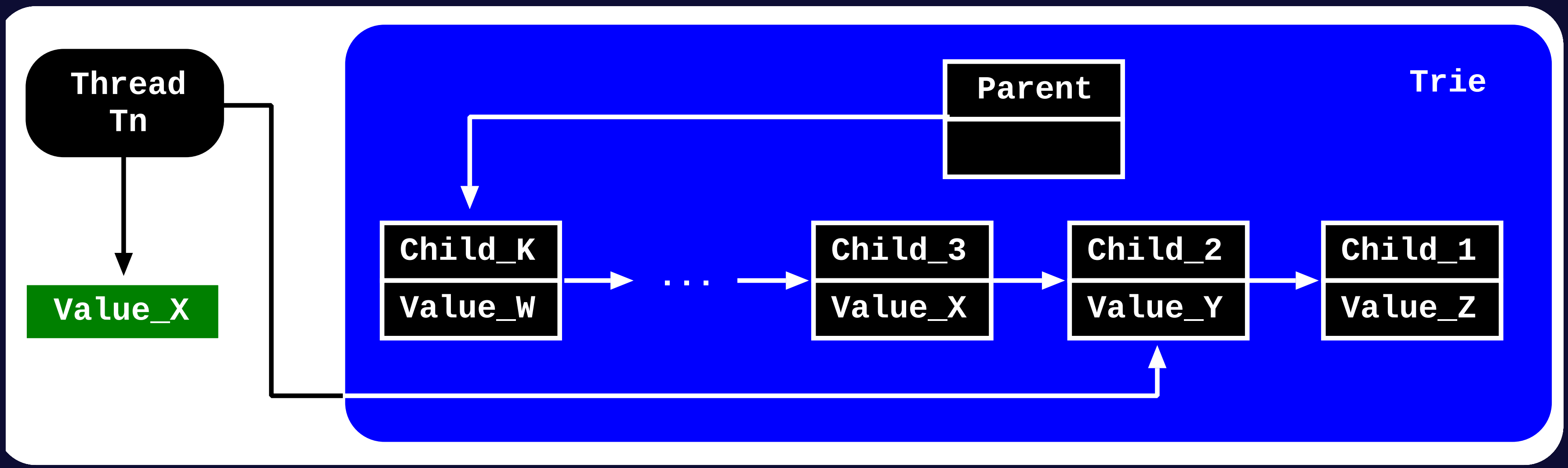


Table Locking Schemes - TLWL

Original Approach

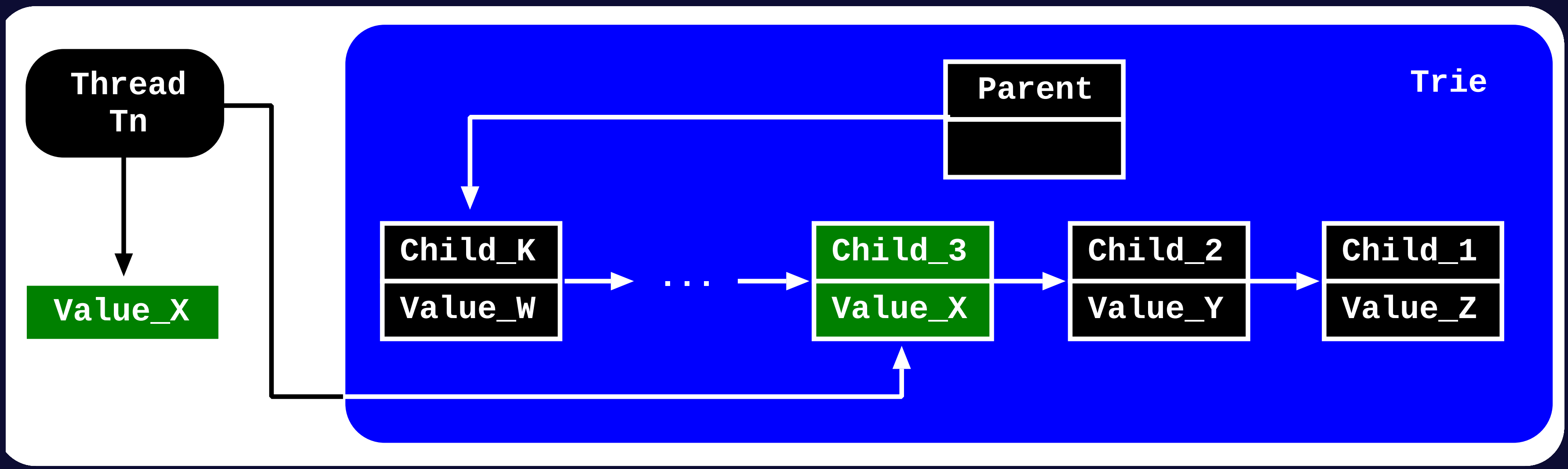


Table Locking Schemes - TLWL

➤ Try-Locks Approach

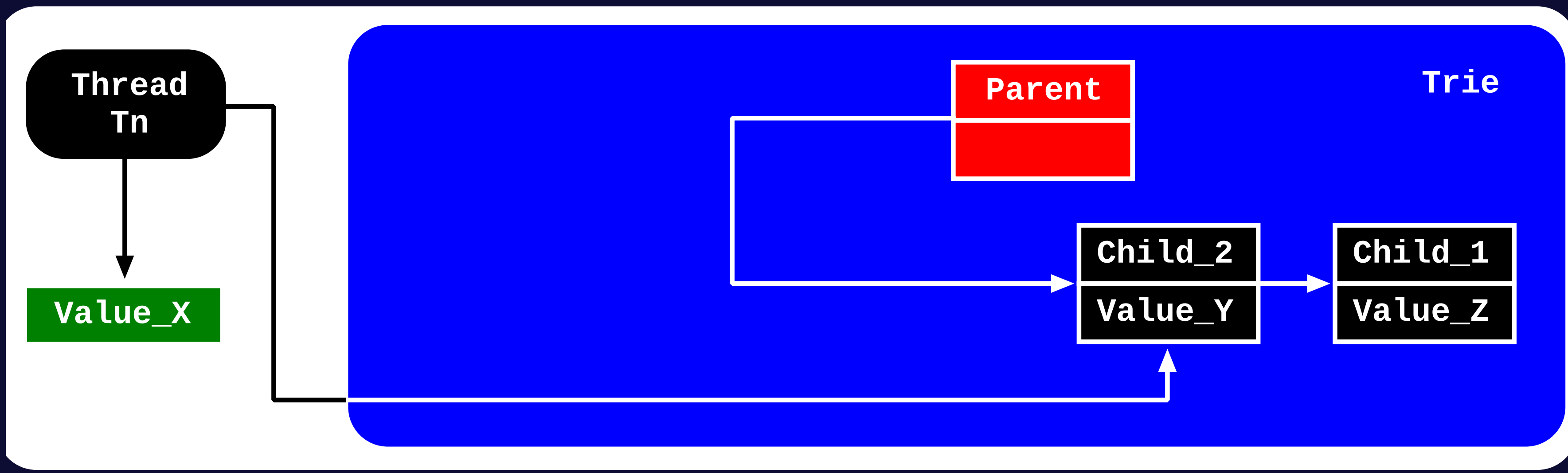


Table Locking Schemes - TLWL

Try-Locks Approach

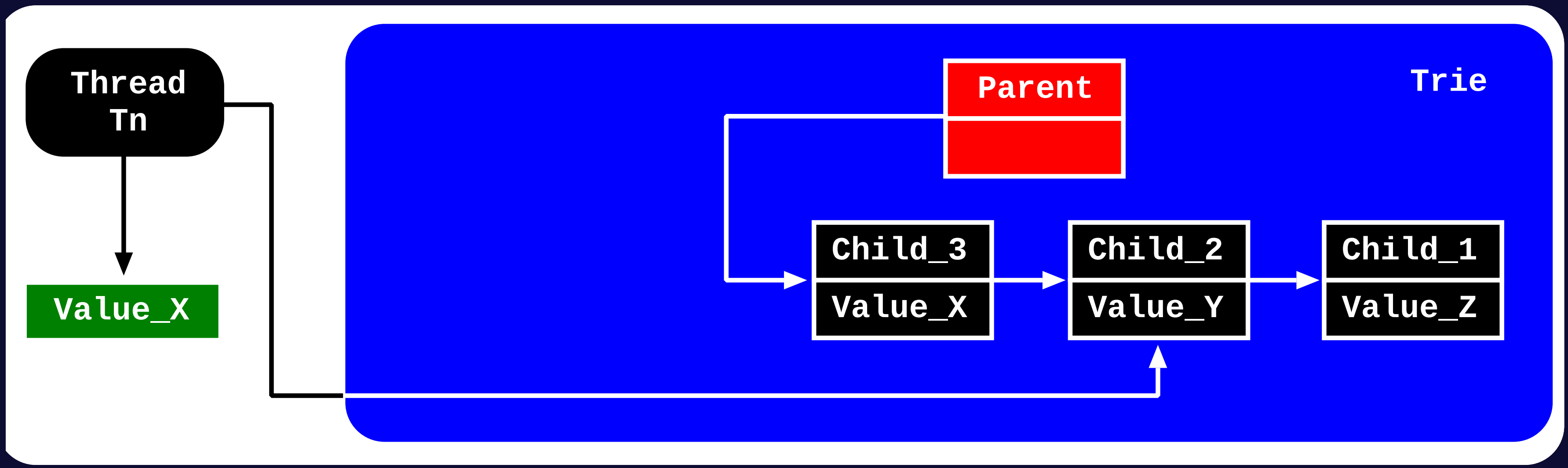
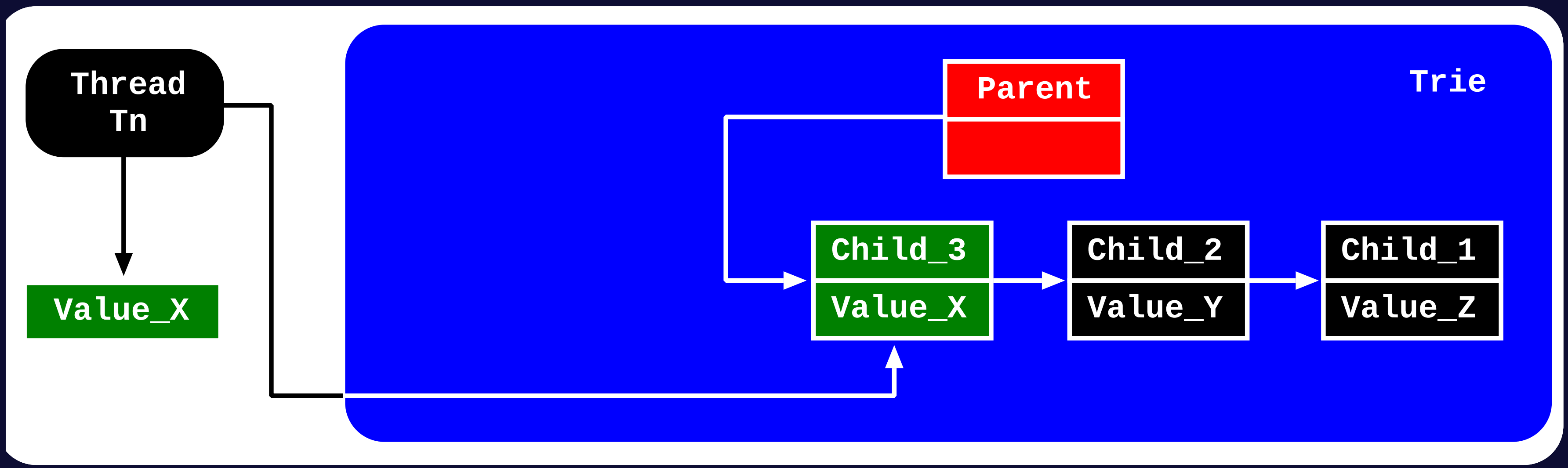


Table Locking Schemes - TLWL

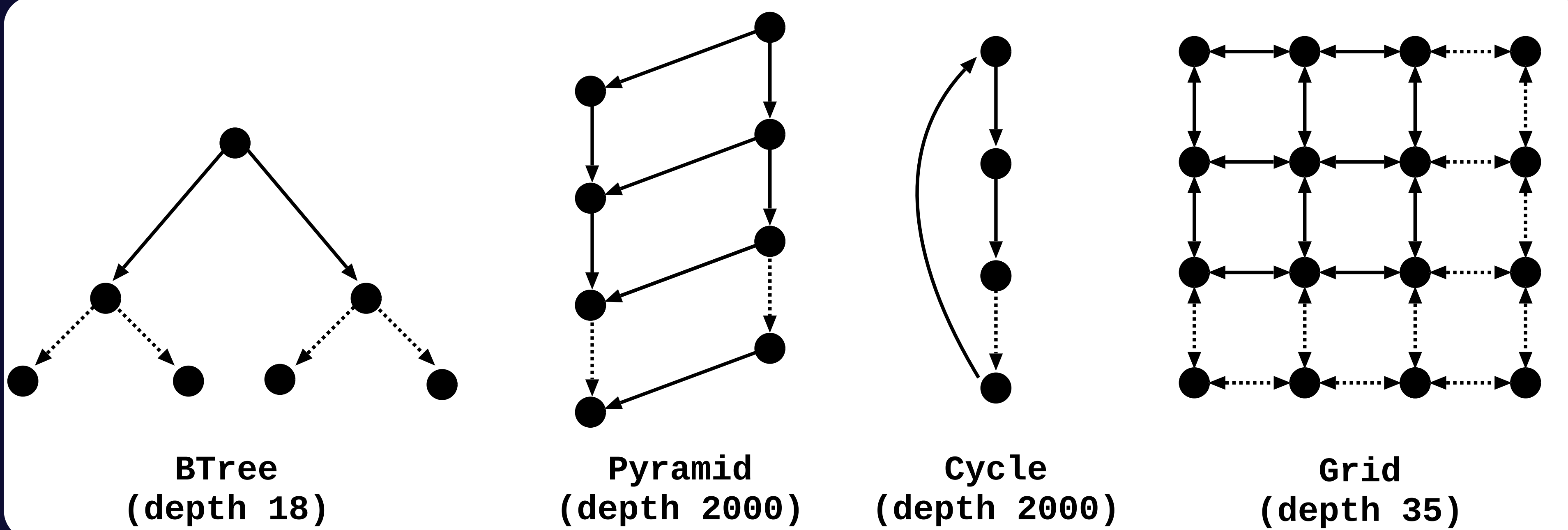
- **Try-Locks Approach**



Path Problem - Configuration

```
path_left(X,Z)  :- path_left(X,Y) , edge(Y,Z).
path_left(X,Z)  :- edge(X,Z).
```

```
path_right(X,Z) :- edge(X,Y) , path_right(Y,Z).
path_right(X,Z) :- edge(X,Z).
```



Experimental Results - Path Left Benchmark

Execution Time (Overhead Ratio To NS): 1 Working Thread (WT)

Config	NS	SS_T	FS	FS_T	XSB
BTree	2,966	2,998 (1.01)	3,826 (1.29)	3,864 (1.30)	**2,798 (0.94)**
Pyramid	3,085	3,159 (1.02)	3,256 (1.06)	3,256 (1.06)	**2,928 (0.95)**
Cycle	3,828	3,921 (1.02)	3,775 (0.99)	3,798 (0.99)	**3,357 (0.88)**
Grid	1,743	**1,791 (1.03)**	2,280 (1.31)	2,293 (1.32)	2,034 (1.17)
	Average	(1.02)	(1.16)	(1.17)	**(0.98)**

Experimental Results - Path Left Benchmark

Execution Time (Overhead Ratio To NS): 1 Working Thread (WT)

Config	NS	SS_T	FS	FS_T	XSB
BTree	2,966	2,998 (1.01)	3,826 (1.29)	3,864 (1.30)	**2,798 (0.94)**
Pyramid	3,085	3,159 (1.02)	3,256 (1.06)	3,256 (1.06)	**2,928 (0.95)**
Cycle	3,828	3,921 (1.02)	3,775 (0.99)	3,798 (0.99)	**3,357 (0.88)**
Grid	1,743	**1,791 (1.03)**	2,280 (1.31)	2,293 (1.32)	2,034 (1.17)
	Average	(1.02)	(1.16)	(1.17)	**(0.98)**

Overhead Ratio To NS With 1 WT: 16 and 24 WTs

Config	16 WTs					24 WTs				
	NS	SS_T	FS	FS_T	XSB	NS	SS_T	FS	FS_T	XSB
BTree	9.85	9.78	6.88	**4.81**	5.11	25.65	25.42	8.03	**5.97**	8.09
Pyramid	7.67	7.79	3.74	**3.40**	4.40	24.92	24.88	5.86	**4.48**	7.02
Cycle	7.32	7.38	3.73	**3.25**	4.36	22.39	23.05	5.95	**4.08**	6.99
Grid	5.99	6.00	3.77	3.15	**2.41**	19.82	19.80	4.65	**4.46**	5.30
Average	7.71	7.74	4.53	**3.65**	4.07	23.20	23.29	6.12	**4.75**	6.85

Experimental Results - Path Right Benchmark

Execution Time (Overhead Ratio To NS): 1 Working Thread (WT)

Config	NS	SS_T	FS	FS_T	XSB
BTree	4,568	5,048 (1.11)	5,673 (1.24)	5,701 (1.25)	**3,551 (0.78)**
Pyramid	2,520	2,531 (1.00)	3,664 (1.45)	3,673 (1.46)	**2,350 (0.93)**
Cycle	2,761	**2,773 (1.00)**	3,994 (1.45)	3,992 (1.45)	2,817 (1.02)
Grid	2,109	**2,110 (1.00)**	3,097 (1.47)	3,117 (1.48)	2,462 (1.17)
	Average	(1.03)	(1.40)	(1.41)	**(0.97)**

Overhead Ratio To NS With 1 WT: 16 and 24 WTs

Config	16 WTs					24 WTs				
	NS	SS_T	FS	FS_T	XSB	NS	SS_T	FS	FS_T	XSB
BTree	13.82	13.13	10.57	**5.54**	6.33	29.53	27.36	10.16	**6.76**	10.38
Pyramid	17.09	17.00	14.85	8.15	**5.94**	46.25	45.31	10.86	10.42	**10.31**
Cycle	17.96	18.17	17.05	8.36	**6.63**	47.89	47.60	11.49	**10.76**	10.99
Grid	9.52	9.48	7.13	5.53	**3.75**	26.58	27.80	7.50	6.96	**6.41**
Average	14.60	14.44	12.40	6.90	**5.66**	37.56	37.02	10.00	**8.73**	9.52

Experimental Results - Model Checking Benchmark

Execution Time (Overhead Ratio To NS): 1 Working Thread (WT)

Config	NS	SS_T	FS	FS_T	XSB
IProto	2,517	**2,449 (0.97)**	2,816 (1.12)	2,828 (1.12)	3,675 (1.46)
Leader	3,726	**3,800 (1.02)**	3,830 (1.03)	3,897 (1.05)	10,354 (2.78)
Sieve	23,645	**24,402 (1.03)**	24,479 (1.04)	25,201 (1.07)	27,136 (1.15)
	Average	**(1.01)**	(1.06)	(1.08)	(1.80)

Overhead Ratio To NS With 1 WT: 16 and 24 WTs

Config	16 WTs					24 WTs				
	NS	SS_T	FS	FS_T	XSB	NS	SS_T	FS	FS_T	XSB
IProto	4.15	4.20	1.60	**1.55**	1.92	7.16	7.31	1.71	**1.63**	2.14
Leader	**1.02**	1.04	1.05	1.07	2.80	**1.02**	1.04	1.05	1.07	2.79
Sieve	**1.01**	1.04	1.05	1.08	1.15	**1.02**	1.04	1.06	1.08	1.15
Average	2.06	2.09	1.24	**1.23**	1.95	3.07	3.13	1.27	**1.26**	2.03

Experimental Results - Statistics

Characteristics Of The Benchmarks: 1 Working Thread (WT)

Bench	Tabled Subgoals			Tabled Answers			
	calls	trie nodes	trie depth	unique	repeated	trie nodes	trie depth
Path Left							
BTree	1	3	2/2/2	1,966,082	0	2,031,618	2/2/2
Pyramid	1	3	2/2/2	3,374,250	1,124,250	3,377,250	2/2/2
Cycle	1	3	2/2/2	4,000,000	2,000	4,002,001	2/2/2
Grid	1	3	2/2/2	1,500,625	4,335,135	1,501,851	2/2/2
Path Right							
BTree	131,071	262,143	2/2/2	3,801,094	0	3,997,700	1/2/2
Pyramid	3,000	6,001	2/2/2	6,745,501	2,247,001	6,751,500	1/2/2
Cycle	2,001	4,003	2/2/2	8,000,000	4,000	8,004,001	1/2/2
Grid	1,226	2,453	2/2/2	3,001,250	8,670,270	3,003,701	1/2/2
Model Checking							
IProto	1	6	5/5/5	134,361	385,423	1,554,896	4/51/67
Leader	1	5	4/4/4	1,728	574,786	41,788	15/80/97
Sieve	1	7	6/6/6	380	1,386,181	8,624	21/53/58

Conclusions

- We have presented a **new approach** to multi-threaded tabled evaluation of logic programs using a local evaluation strategy.
- Our approach have three designs :
  - **No-Sharing (NS)**;
  - **Subgoal-Sharing (SS)**;
  - **Full-Sharing (FS)**
    - Experiments do show that the **FS** design can **effectively reduce** the **memory consumption** (almost linear to the number of working threads) and the **execution time** (even in worst case scenarios).

Conclusions

- We have presented a **new approach** to multi-threaded tabled evaluation of logic programs using a local evaluation strategy.
- Our approach have three designs :
  - **No-Sharing (NS)**;
  - **Subgoal-Sharing (SS)**;
  - **Full-Sharing (FS)**
    - Experiments do show that the **FS** design can **effectively reduce** the **memory consumption** (almost linear to the number of working threads) and the **execution time** (even in worst case scenarios).
- Further work will include following features:
  - **Yap's memory allocator** - better support for multi-threading;
  - support for **lock-free synchronization** (current version is **lock-based**);
  - support for batched scheduling for the **FS** design.

Thank You !!!

Miguel Areias and Ricardo Rocha
CRACS & INESC-TEC LA
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*

Projects LEAP and HORUS: *http://cracs.fc.up.pt/*

COMPETE
PROGRAMA OPERACIONAL FACTORES DE COMPETITIVIDADE
QREN QUADRO DE REFERÊNCIA ESTRATÉGICO NACIONAL PORTUGAL 2007.2013
UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional
FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA