On Scaling Dynamic Programming Problems with a Multithreaded Tabling System

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*

SEPS 2014, Portland, Oregon, United States

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.
- **Multithreading** combined with **Tabling**:
  - XSB Prolog
  - **YapTab-Mt** **[ICLP 2012]**.

Our Motivation

- **In this work** we extend the **YapTab-Mt** framework to support **asynchronous parallelism**.
  - The **key idea** is that a **thread does not wait** for **other threads** to **compute** a **sub-problem** ...
  - ... but **is able** to **use the result** of the **sub-problem**, if another **thread** has **already computed** it.

Our Motivation

- **In this work** we extend the **YapTab-Mt** framework to support **asynchronous parallelism**.
  - The **key idea** is that a **thread does not wait** for **other threads** to **compute** a **sub-problem** ...
  - ... but **is able** to **use the result** of the **sub-problem**, if another **thread** has **already computed** it.

- Use the **YapTab-Mt** to **scale the execution** of two well-know **dynamic programming problems** that can be found in many domains:
  - **0-1 Knapsack**: logistics, manufacturing, finance or telecommunications.
  - **Longest Common Subsequence (LCS)**: sequence alignment, which is a fundamental technique for biologists to investigate the similarity between species.

Our Motivation

- **In this work** we extend the **YapTab-Mt** framework to support **asynchronous parallelism**.
  - The **key idea** is that a **thread does not wait** for **other threads** to **compute** a **sub-problem** ...
  - ... but **is able** to **use the result** of the **sub-problem**, if another **thread** has **already computed** it.

- Use the **YapTab-Mt** to **scale the execution** of two well-know **dynamic programming problems** that can be found in many domains:
  - **0-1 Knapsack**: logistics, manufacturing, finance or telecommunications.
  - **Longest Common Subsequence (LCS)**: sequence alignment, which is a fundamental technique for biologists to investigate the similarity between species.

- Compare **parallelization techniques**:
  - **Top-Down** vs **Bottom-Up**.

YapTab-Mt - Advantages

- **Abstraction layer** for the **dynamic programming (tabling)** support is provided with a **single instruction**:
  - **:- table predicate/arity.**
  - Example **:- table knapsack/3.**

YapTab-Mt - Advantages

- **Abstraction layer** for the **dynamic programming (tabling)** support is provided with a **single instruction**:
  - **:- table predicate/arity.**
  - Example **:- table knapsack/3.**

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

YapTab-Mt - Advantages

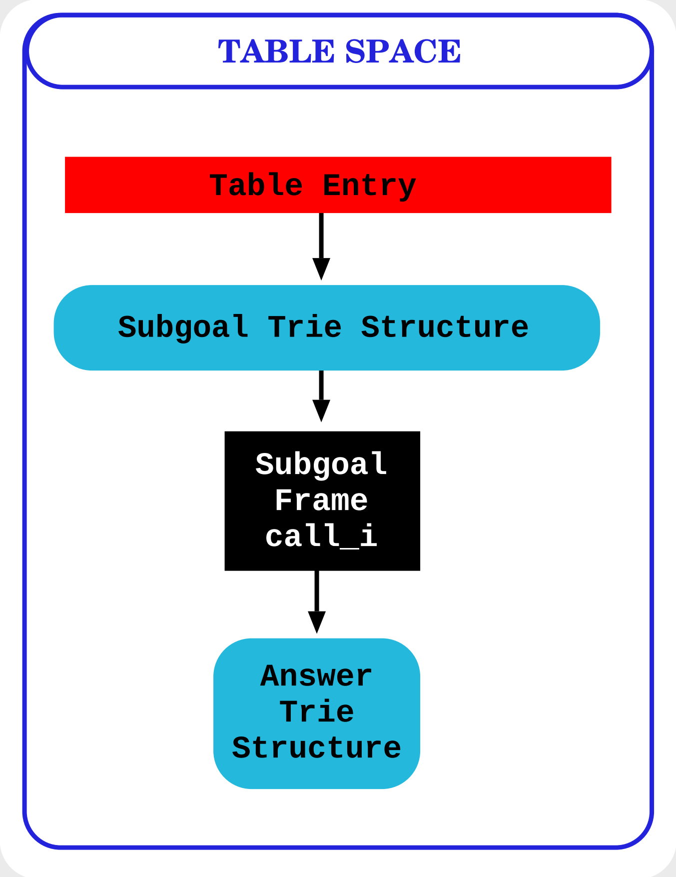
- **Abstraction layer** for the **dynamic programming (tabling)** support is provided with a **single instruction**:
  - **:- table predicate/arity.**
  - Example **:- table knapsack/3.**

- **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**.

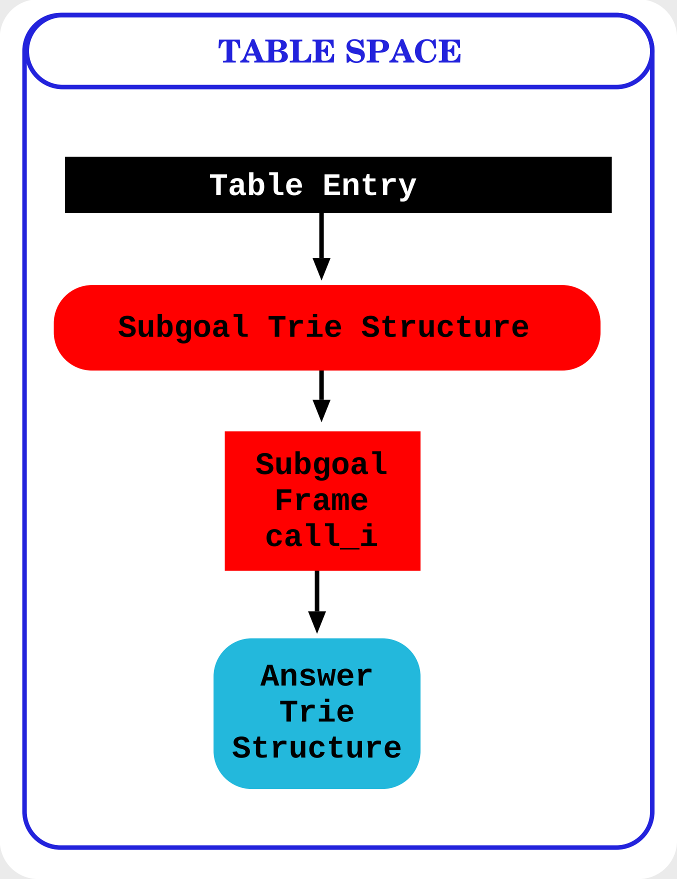
Internal Table Space Architecture

- **Table Entry**: stores generic about the predicates.
  - **table knapsack/3**.



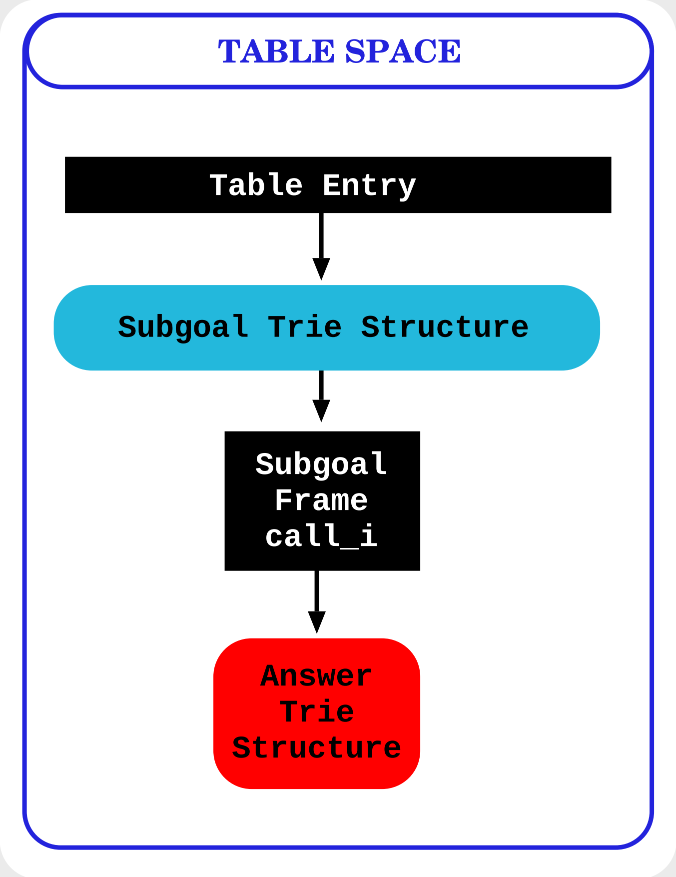
Internal Table Space Architecture

- **Table Entry**: stores generic about the predicates.
  - **table knapsack/3**.
- **Subgoal Trie Structure**: stores the **identifier** of the computations.
  - **knapsack**(**item_i**, **capacity_c**, Profit).

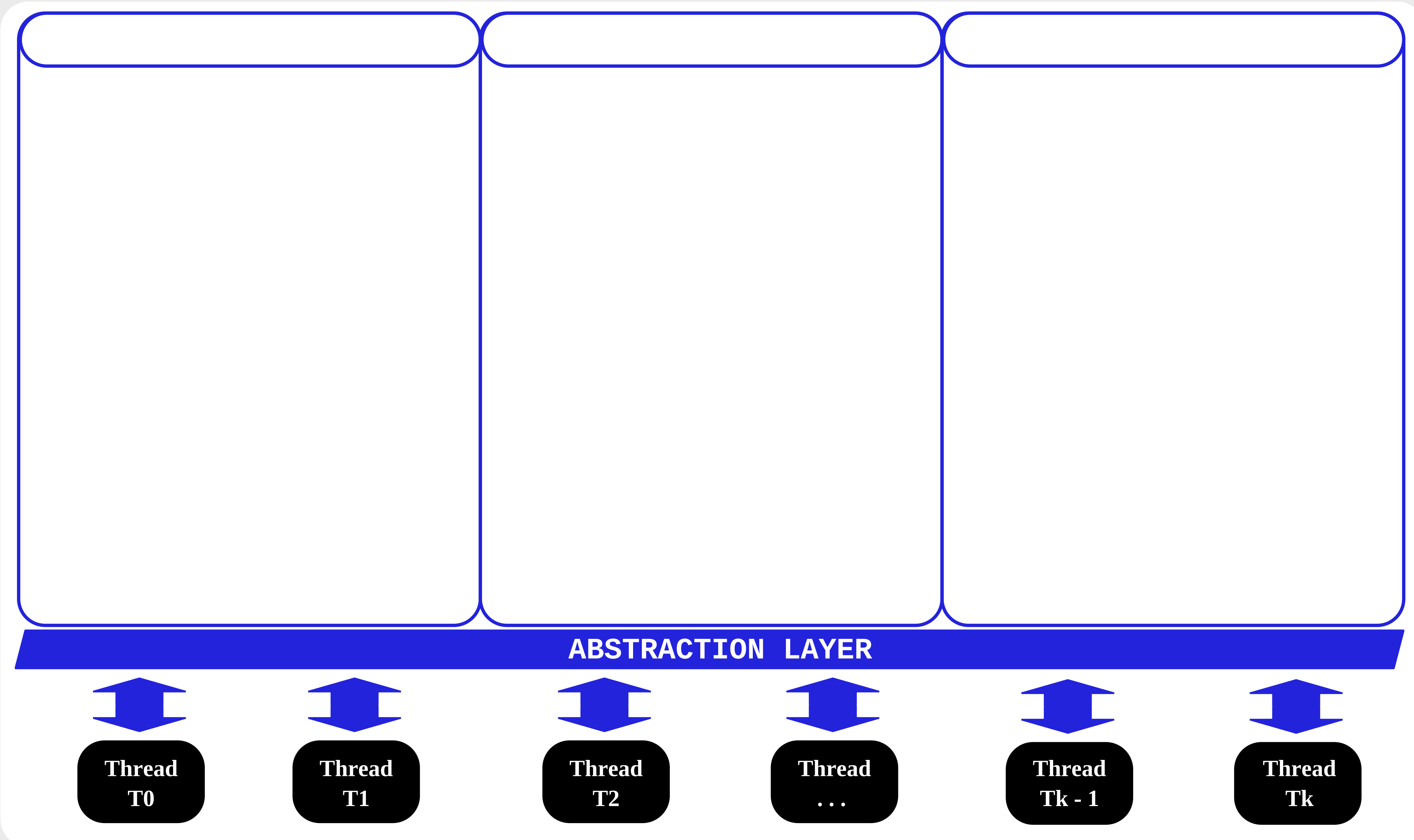


Internal Table Space Architecture

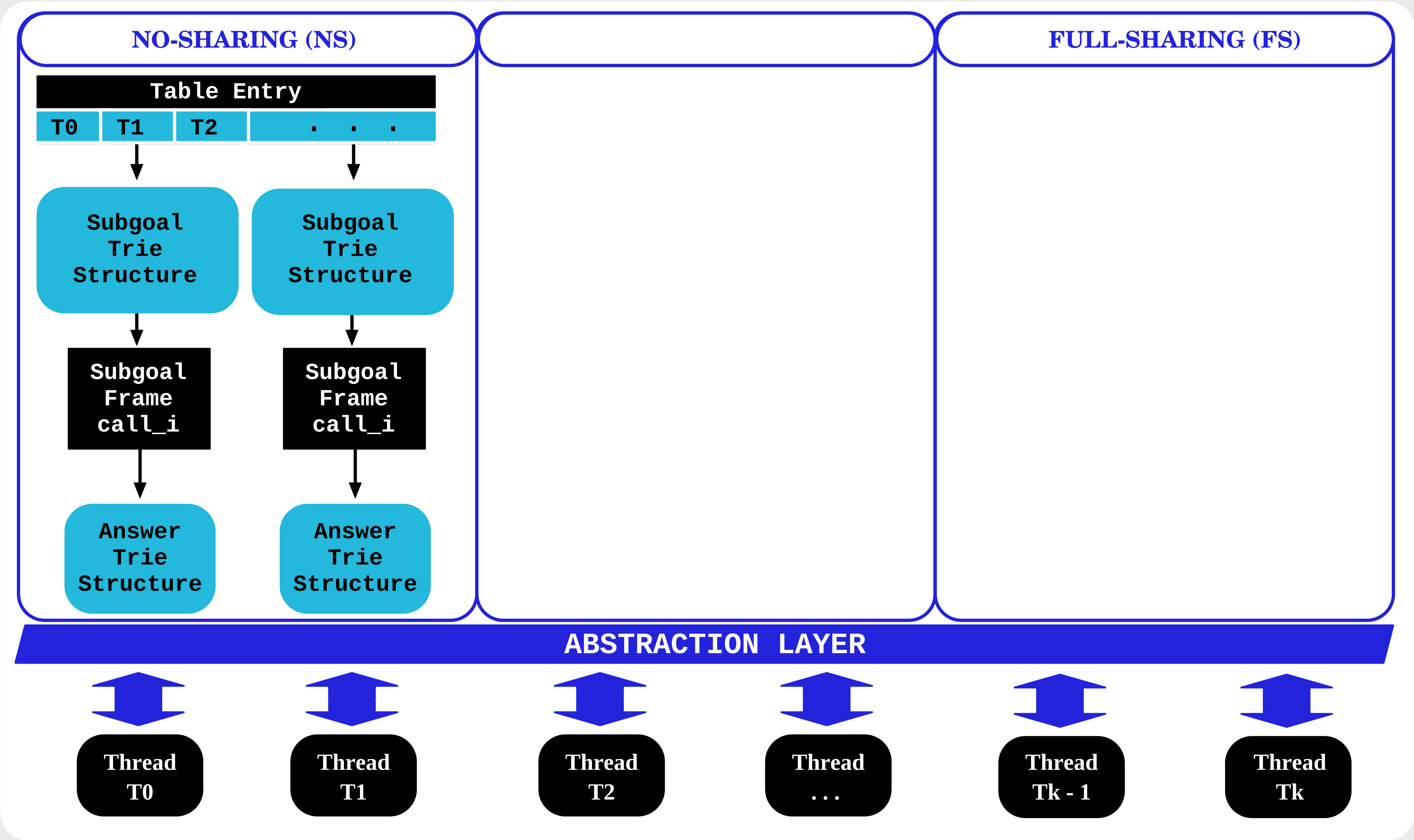
- **Table Entry**: stores generic about the predicates.
  - **table knapsack/3**.
- **Subgoal Trie Structure**: stores the **identifier** of the computations.
  - **knapsack**(**item_i**, **capacity_c**, Profit).
- **Answer Trie Structure**: stores the **answers** of the computations.
  - **knapsack**(item_i, capacity_c, **Profit**).



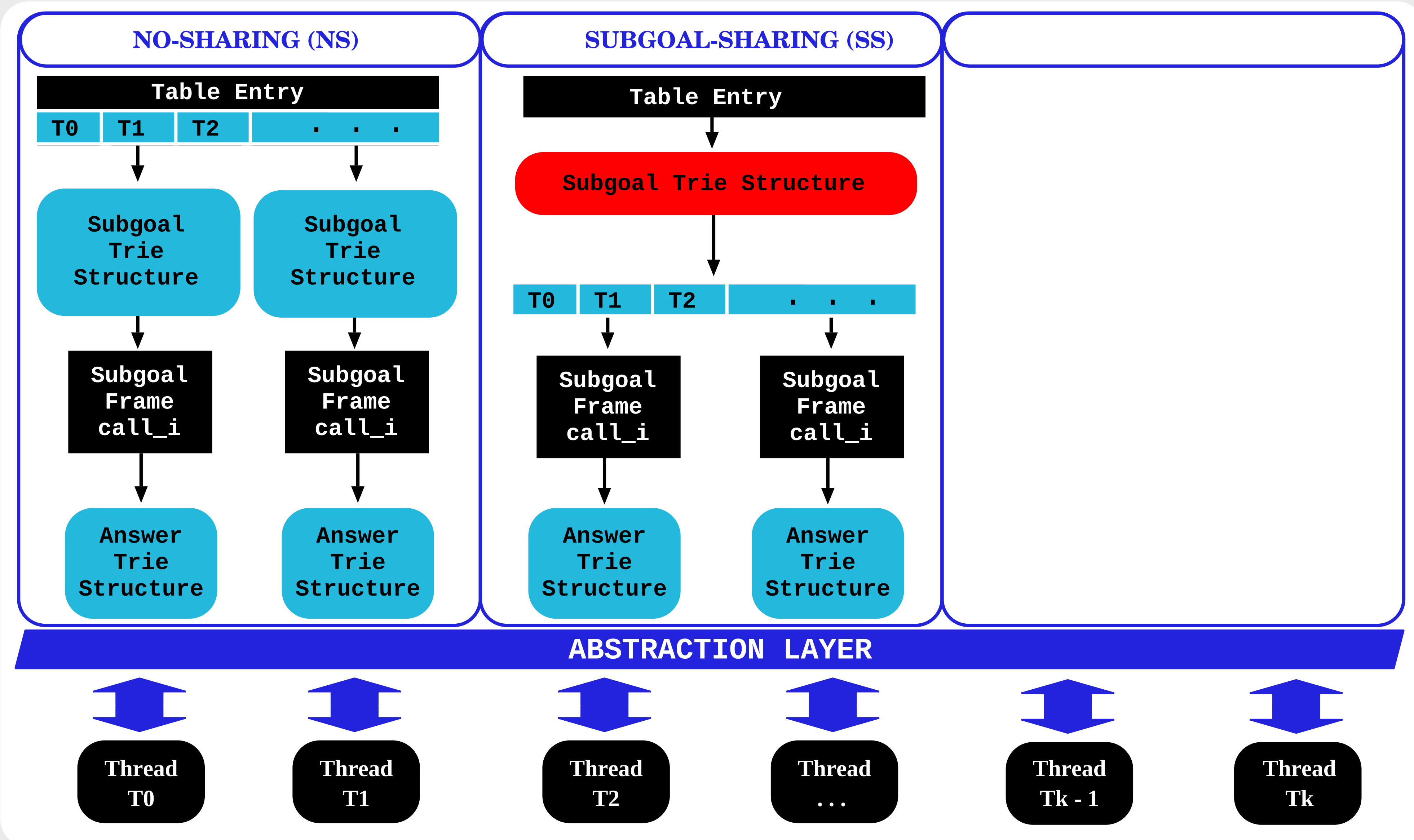
YapTab-Mt - Internal Architecture



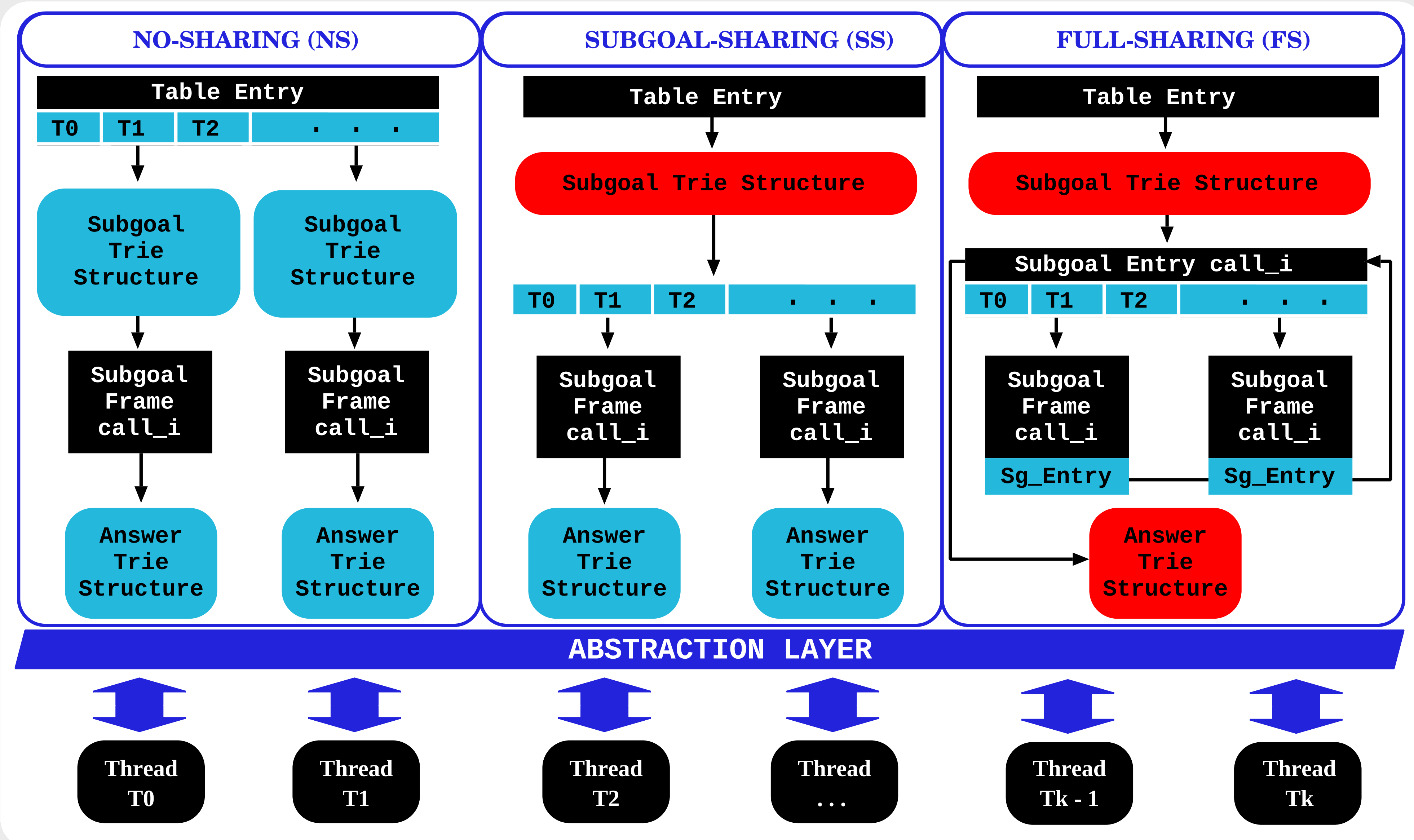
YapTab-Mt - Internal Architecture



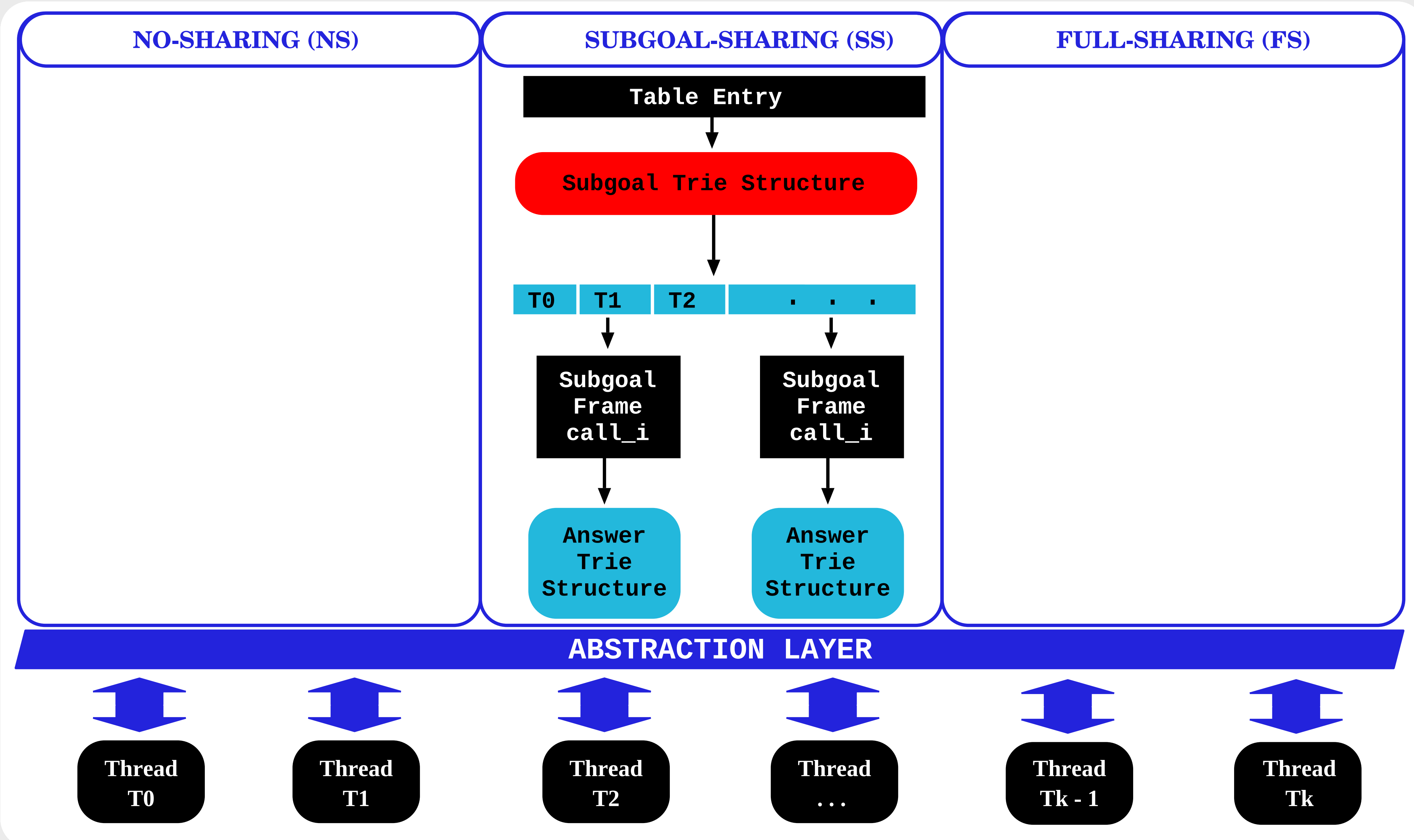
YapTab-Mt - Internal Architecture



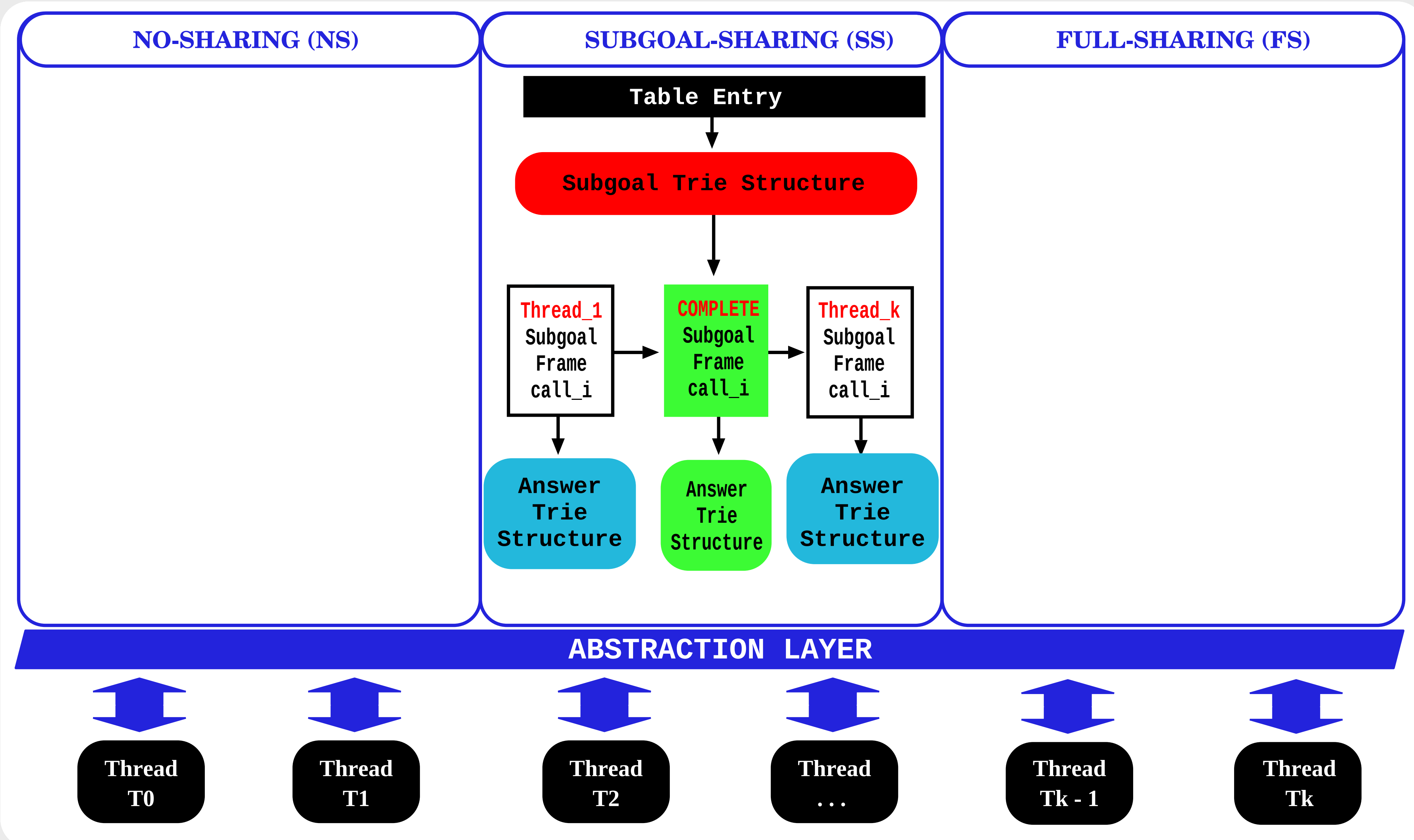
YapTab-Mt - Internal Architecture



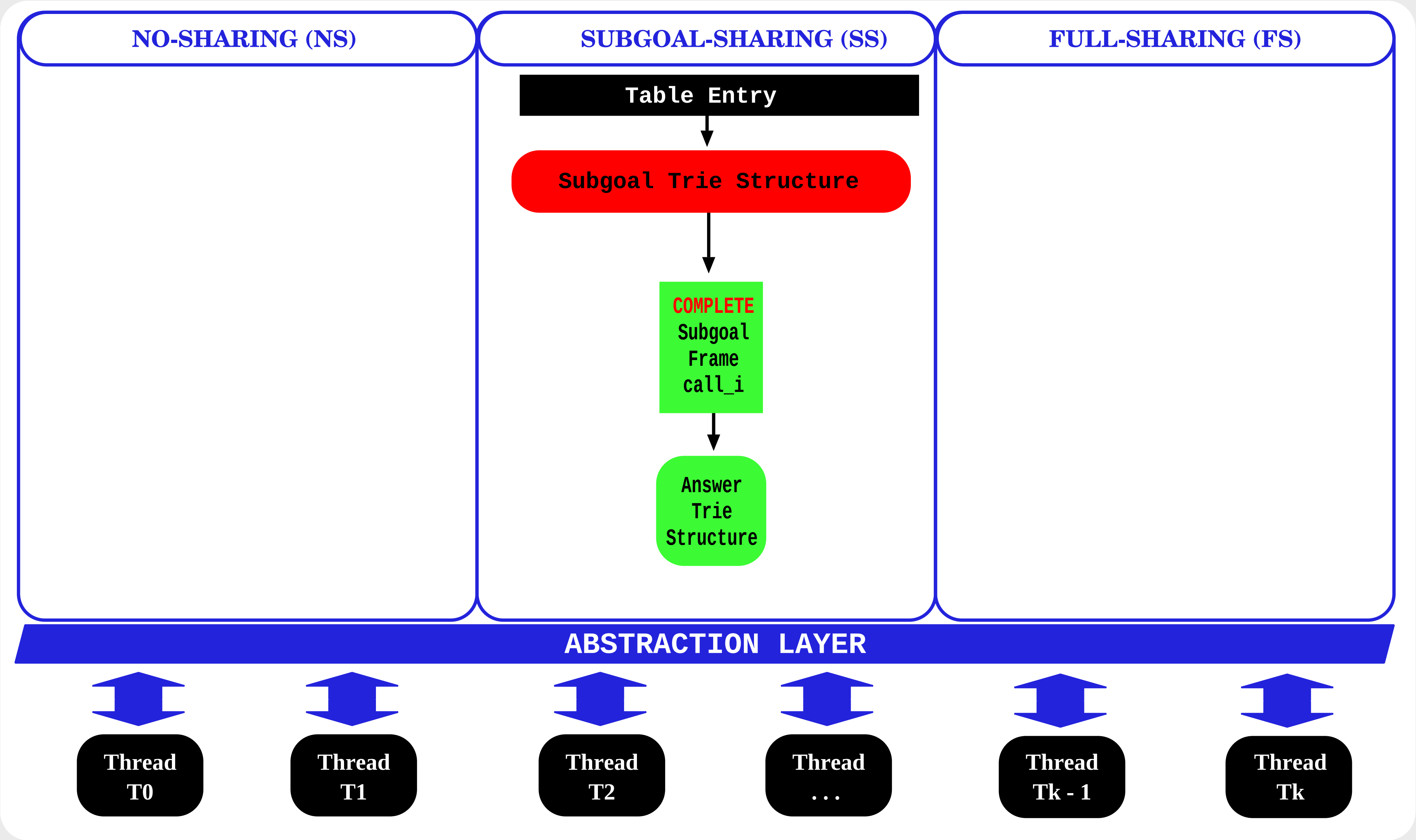
YapTab-Mt - Internal Architecture



YapTab-Mt - Internal Architecture

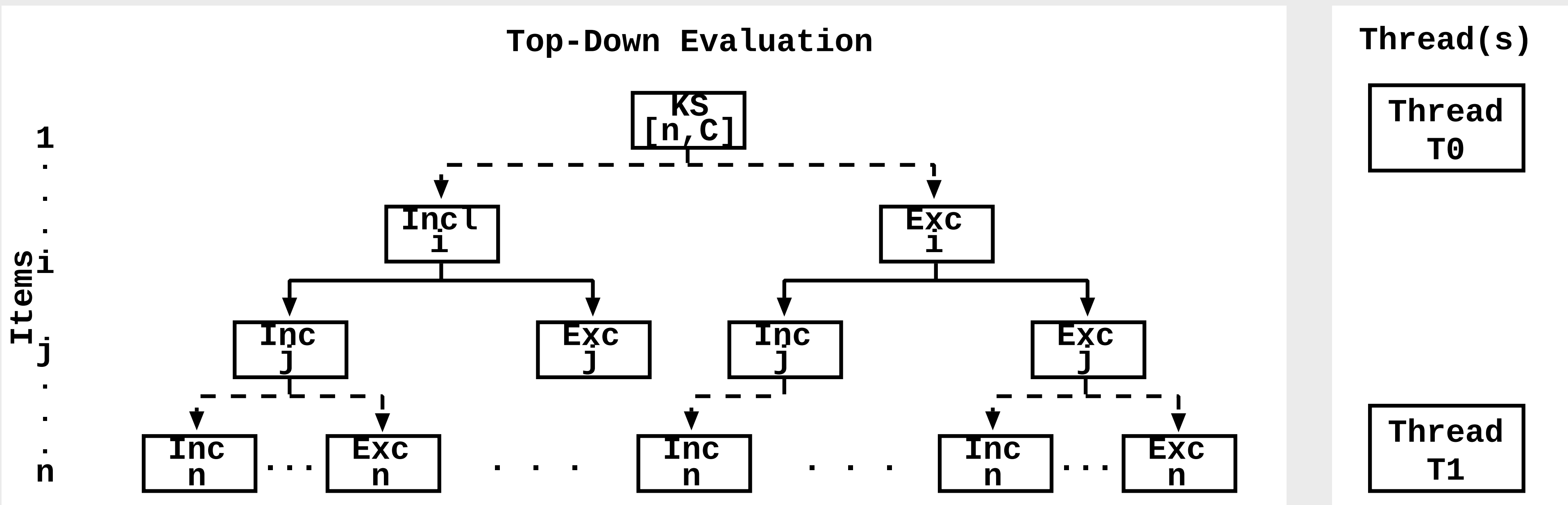


YapTab-Mt - Internal Architecture



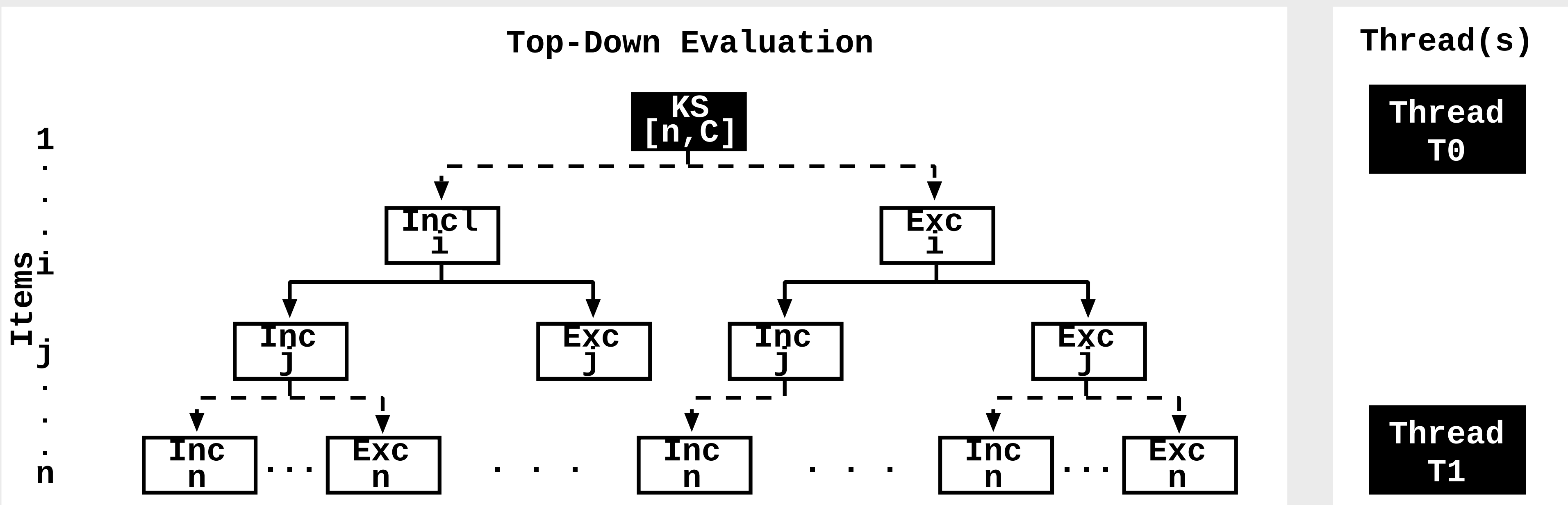
0-1 Knapsack Problem (Top-Down)

- An **item** is **included or excluded** from the **Knapsack** whether it **belongs or not** to the **best solution** of the problem.
- Thread(s) scheduling:
  - Threads **begin** their evaluation in the **top query**.
  - **Disperse threads** through the **evaluation tree** using **random branch orders**.



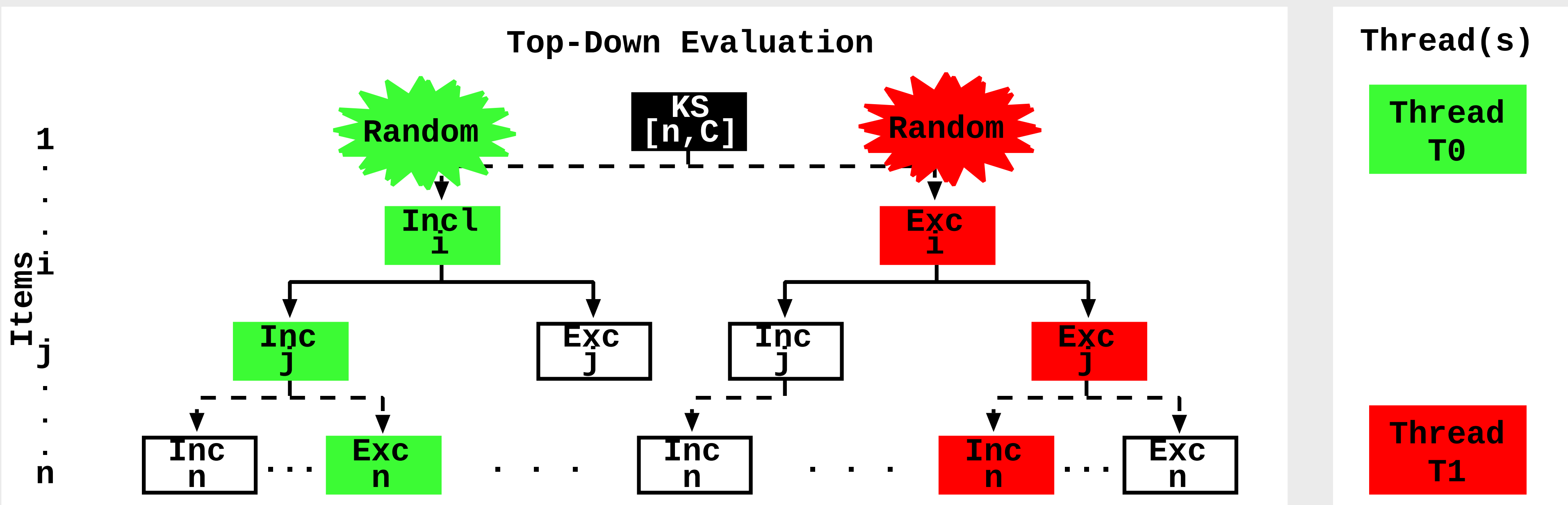
0-1 Knapsack Problem (Top-Down)

- An **item** is **included or excluded** from the **Knapsack** whether it **belongs or not** to the **best solution** of the problem.
- Thread(s) scheduling:
  - Threads **begin** their evaluation in the **top query**.
  - **Disperse threads** through the **evaluation tree** using **random branch orders**.



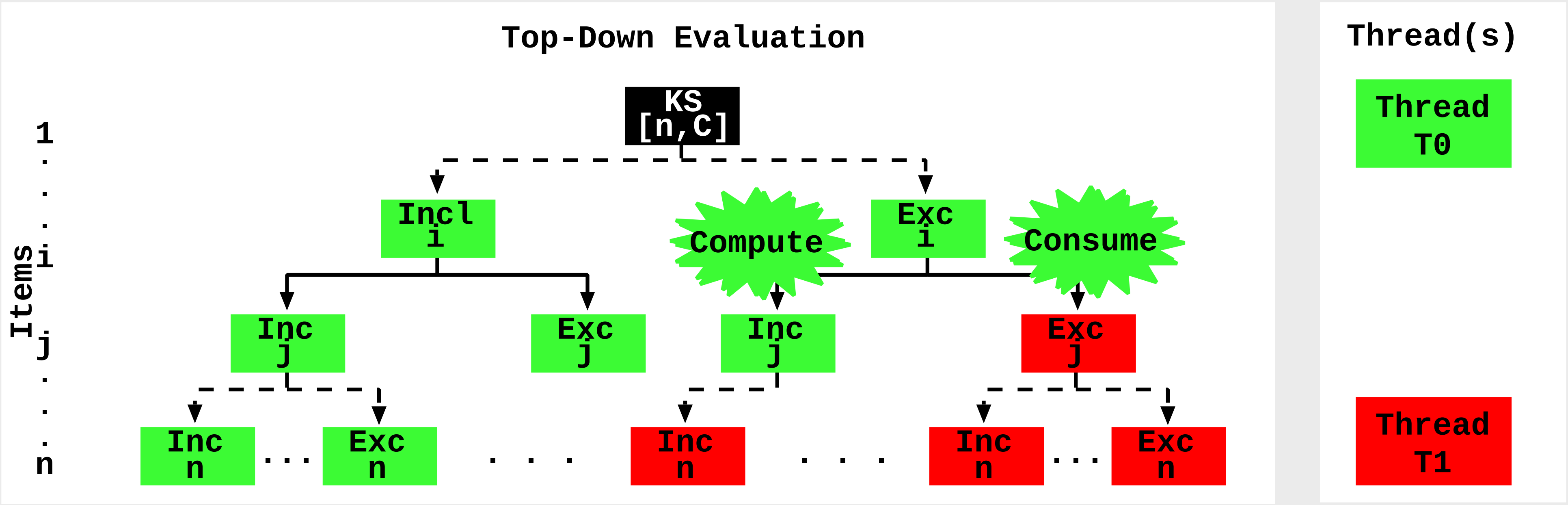
0-1 Knapsack Problem (Top-Down)

- An **item** is **included or excluded** from the **Knapsack** whether it **belongs or not** to the **best solution** of the problem.
- Thread(s) scheduling:
  - Threads **begin** their evaluation in the **top query**.
  - **Disperse threads** through the **evaluation tree** using **random branch orders**.



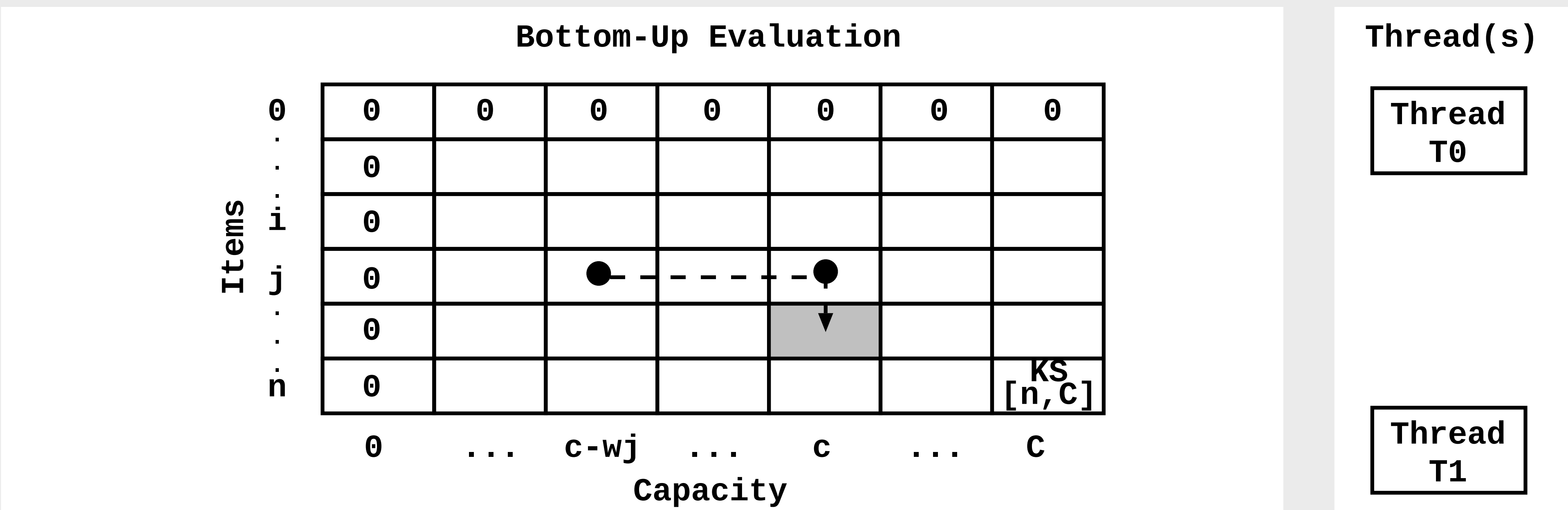
0-1 Knapsack Problem (Top-Down)

- An **item** is **included or excluded** from the **Knapsack** whether it **belongs or not** to the **best solution** of the problem.
- Thread(s) scheduling:
  - Threads **begin** their evaluation in the **top query**.
  - **Disperse threads** through the **evaluation tree** using **random branch orders**.



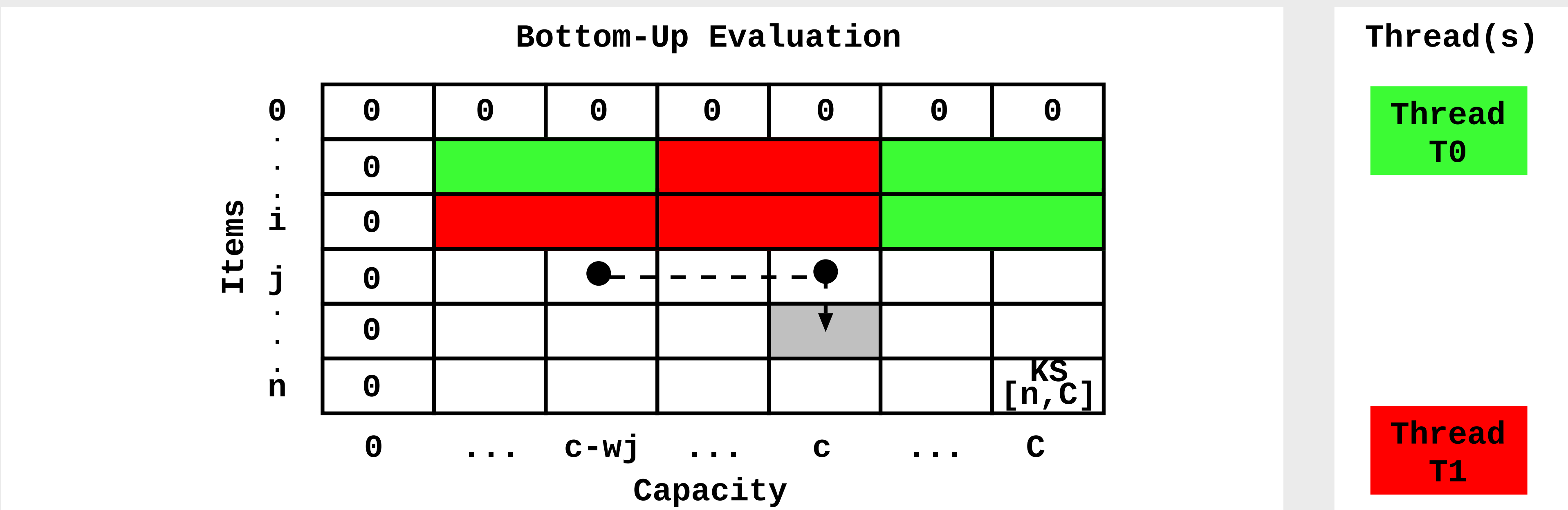
0-1 Knapsack Problem (Bottom-Up)

- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.



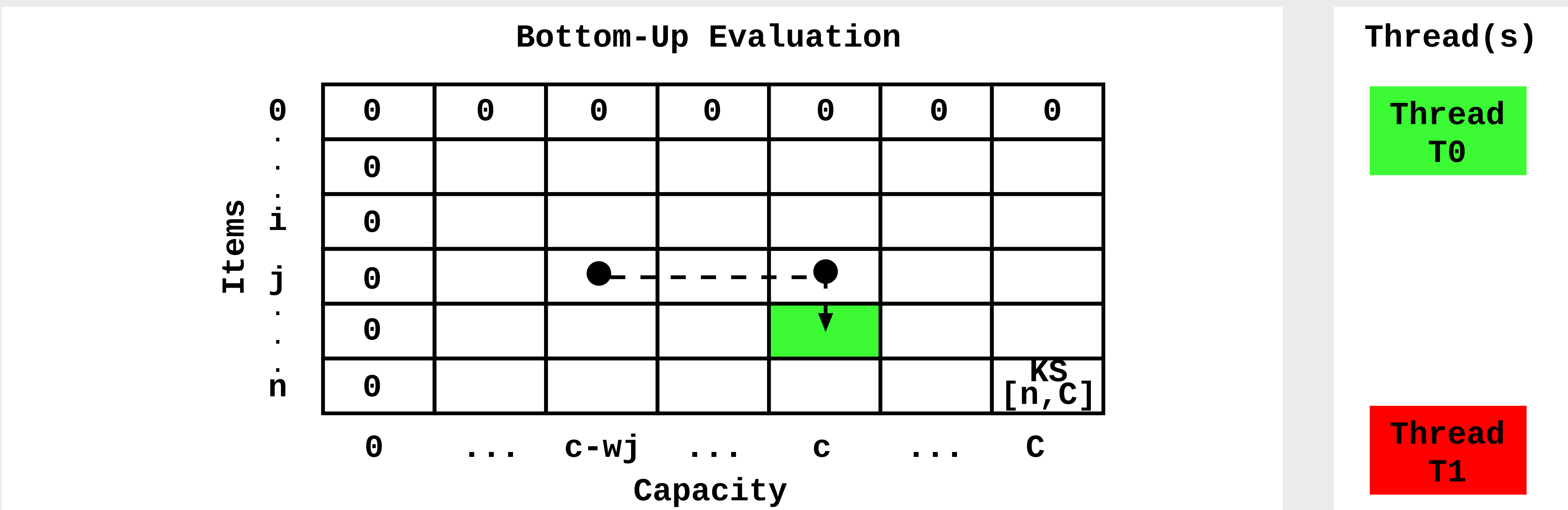
0-1 Knapsack Problem (Bottom-Up)

- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.



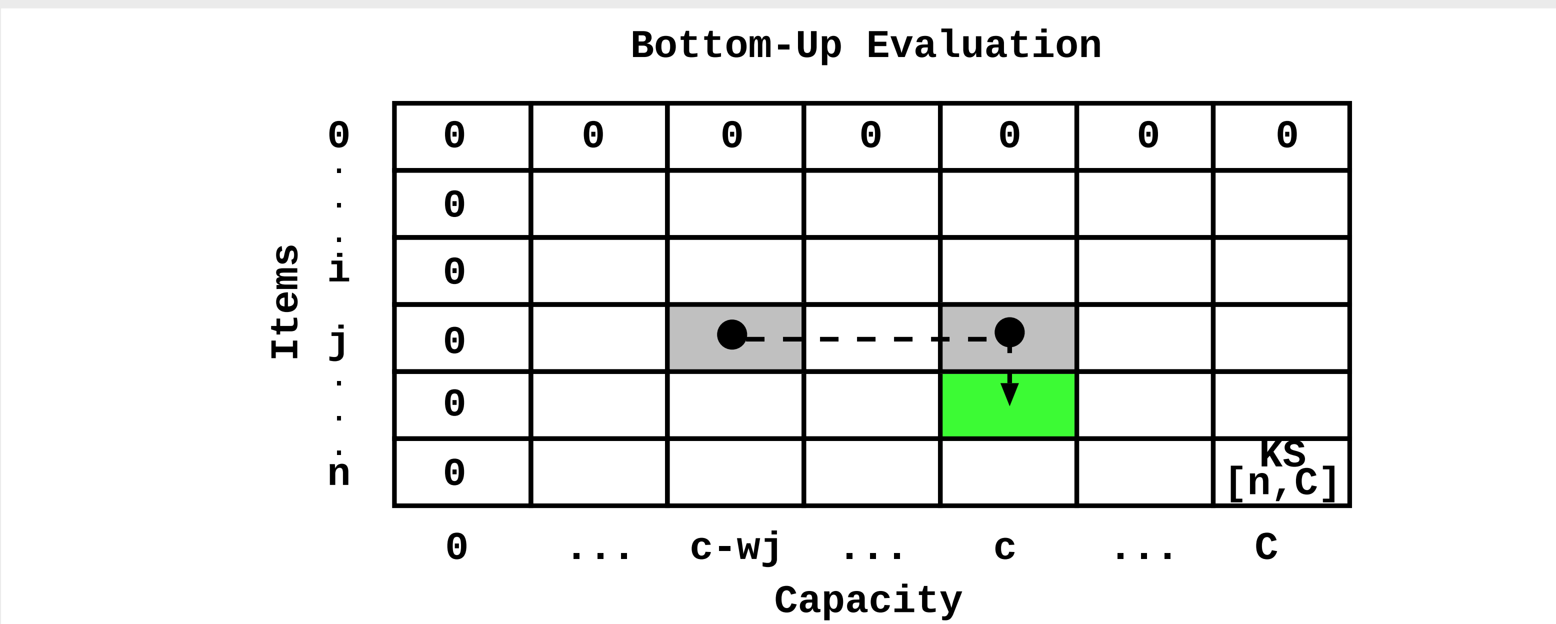
0-1 Knapsack Problem (Bottom-Up)

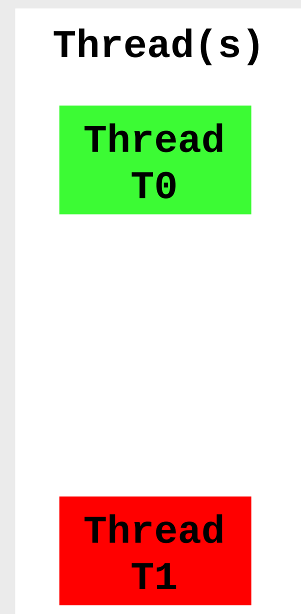
- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.



0-1 Knapsack Problem (Bottom-Up)

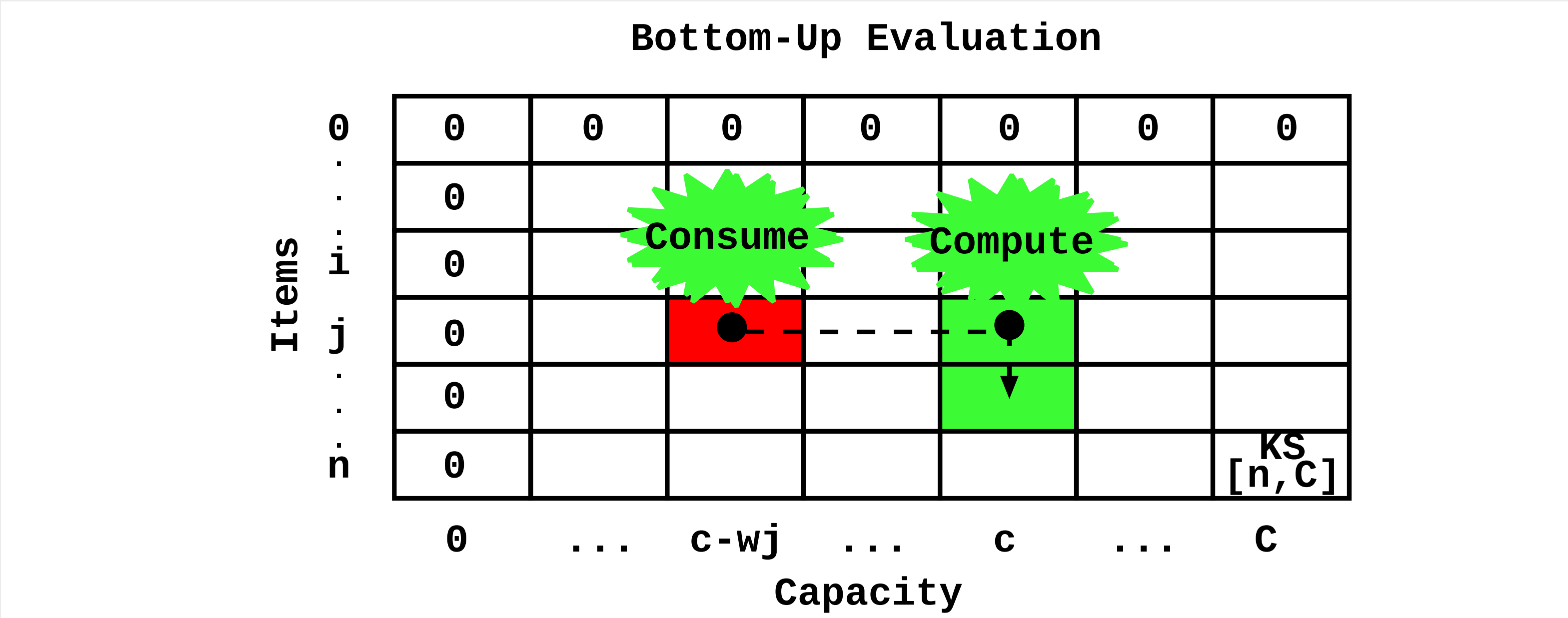
- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.

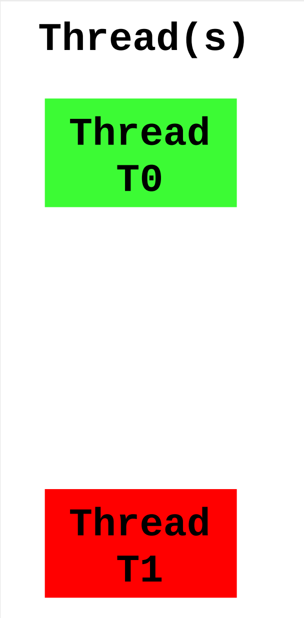




0-1 Knapsack Problem (Bottom-Up)

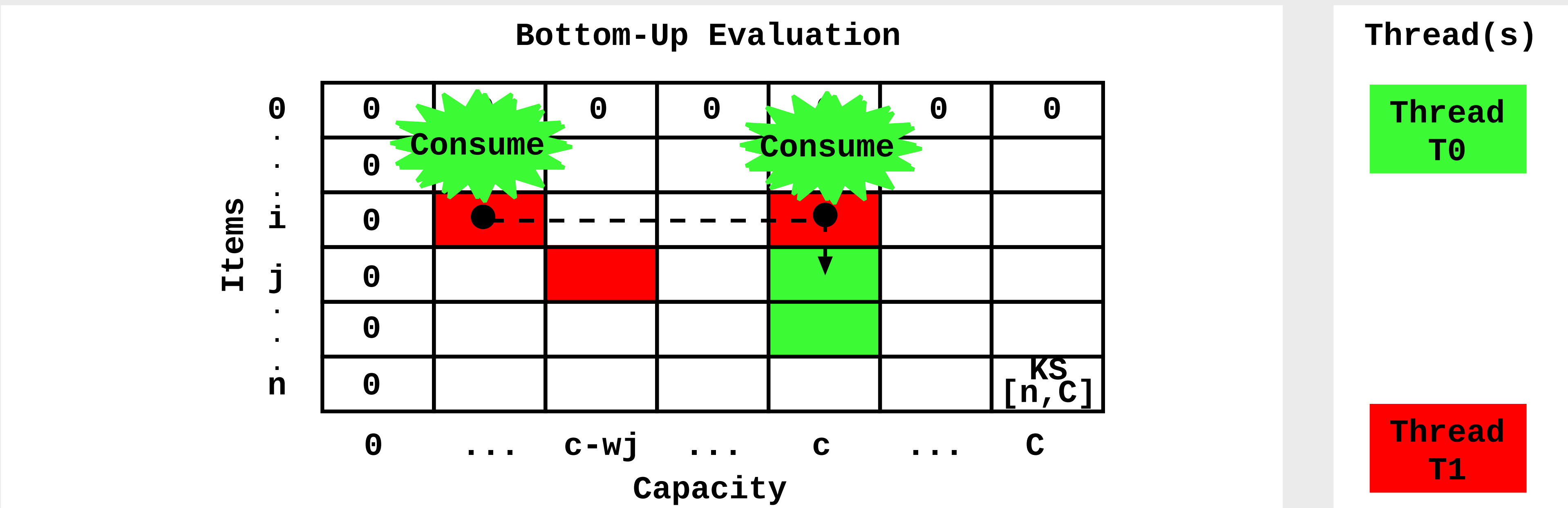
- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.





0-1 Knapsack Problem (Bottom-Up)

- Evaluate the **combination** of **all items** with **all possible capacities** for the **Knapsack**. After all combinations are evaluated, the **best solution** of the problem has the **items that belong** to the **Knapsack**.
- Thread(s) scheduling:
  - **Divide** the **complete combination** in **smaller chunks** and **evaluate them** in the **threads**.



Experimental Results - 0-1 Knapsack Problem

System/Dataset		Time (T_1)	Speedup (T_1/T_p)				Best
		# Threads (p)					
		1	8	16	24	32	Time
Top-Down Approaches							
YAP_{TD_1}	D_{10}	18,319	1.96	**2.10**	2.01	1.89	8,723
	D_{30}	17,664	3.41	**3.96**	3.83	3.62	4,461
	D_{50}	17,828	4.72	6.12	**6.21**	6.07	2,871
YAP_{TD_2}	D_{10}	23,816	6.78	11.95	14.81	**16.79**	1,418
	D_{30}	25,049	7.39	13.63	16.85	**19.35**	1,295
	D_{50}	24,866	7.38	13.67	16.78	**19.23**	1,293
Bottom-Up Approaches							
YAP_{BU}	D_{10}	17,054	7.25	13.32	17.12	**19.60**	0,870
	D_{30}	17,005	7.22	13.47	17.29	**19.64**	0,866
	D_{50}	16,550	7.16	13.29	17.04	**19.60**	0,844
XSB_{BU}	D_{10}	37,338	**0.81**	0.79	0.73	0.54	37,338
	D_{30}	38,245	**0.82**	0.75	0.75	0.56	38,245
	D_{50}	39,100	**0.82**	0.79	0.73	0.54	39,100

Experimental Results - LCS Problem

System/Dataset		# Threads (p)					
		Time (T_1)	Speedup (T_1/T_p)				Best
		1	8	16	24	32	Time
Top-Down Approaches							
YAP_{TD_1}	D_{10}	30,708	**1.53**	1.45	1.40	1.29	20,071
	D_{30}	30,817	**1.53**	1.46	1.38	1.28	20,142
	D_{50}	30,707	**1.52**	1.44	1.39	1.27	20,202
YAP_{TD_2}	D_{10}	42,556	7.25	13.13	16.26	**18.32**	2,323
	D_{30}	42,511	7.21	13.24	16.19	**18.34**	2,318
	D_{50}	42,631	7.21	13.15	16.27	**18.33**	2,326
Bottom-Up Approaches							
YAP_{BU}	D_{10}	27,253	6.97	10.78	14.88	**17.91**	1,522
	D_{30}	27,045	6.88	11.20	14.74	**17.92**	1,509
	D_{50}	27,102	6.97	11.91	14.51	**18.07**	1,500
XSB_{BU}	D_{10}	68,255	n.a.	n.a.	n.a.	n.a.	68,255
	D_{30}	69,700	n.a.	n.a.	n.a.	n.a.	69,700
	D_{50}	70,100	n.a.	n.a.	n.a.	n.a.	70,100

Conclusions and Further Work

- The **0-1 Knapsack** and the **Longest Common Subsequence** problems are two well-know **dynamic programming problems**.
  - We have **discussed** how we were able to **scale the execution** by taking advantage of the **YapTap-Mt** framework.
    - **Top-Down** vs **Bottom-Up**.

Conclusions and Further Work

- The **0-1 Knapsack** and the **Longest Common Subsequence** problems are two well-know **dynamic programming problems**.
  - We have **discussed** how we were able to **scale the execution** by taking advantage of the **YapTap-Mt** framework.
    - **Top-Down** vs **Bottom-Up**.

- **Further work** will include:
  - Use the **YapTap-Mt** framework in **other dynamic programming problems** and **domains**.
  - Compare the **YapTap-Mt** framework with other **domain-specific languages** and **libraries for dynamic programming**.
  - Use **timestamped tries** to **reduce** even further the **memory used** in the table space.

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 SIBILA: *http://cracs.fc.up.pt/*

FCT Grant: *SFRH/BD/69673/2010*

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