Multithreaded Tabling for Logic Programming

Miguel Areias

Department of Computer Science,
Faculty of Sciences,
University of Porto, Portugal
miguel-areias@dcc.fc.up.pt

Prolog and SLD Resolution

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

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

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

Tabling in Prolog Systems

- **Tabling** or **memoing** is an implementation technique that **overcomes some of the limitations** of the standard **Prolog** resolution:

```
:- table path/2.

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

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

Tabling in Prolog Systems

- **Tabling** or **memoing** is an implementation technique that **overcomes some of the limitations** of the standard **Prolog** resolution:

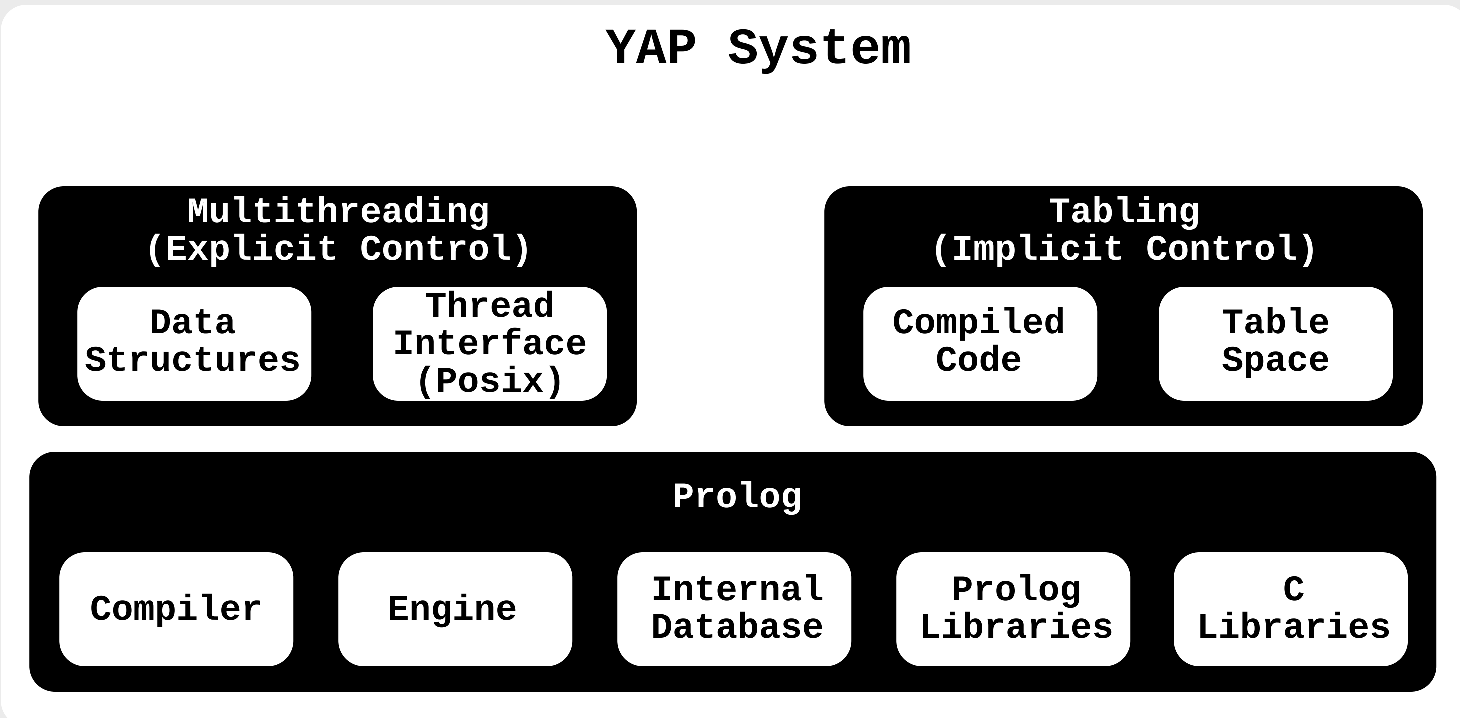
```
:- table path/2.

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

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

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

U.PORTO FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

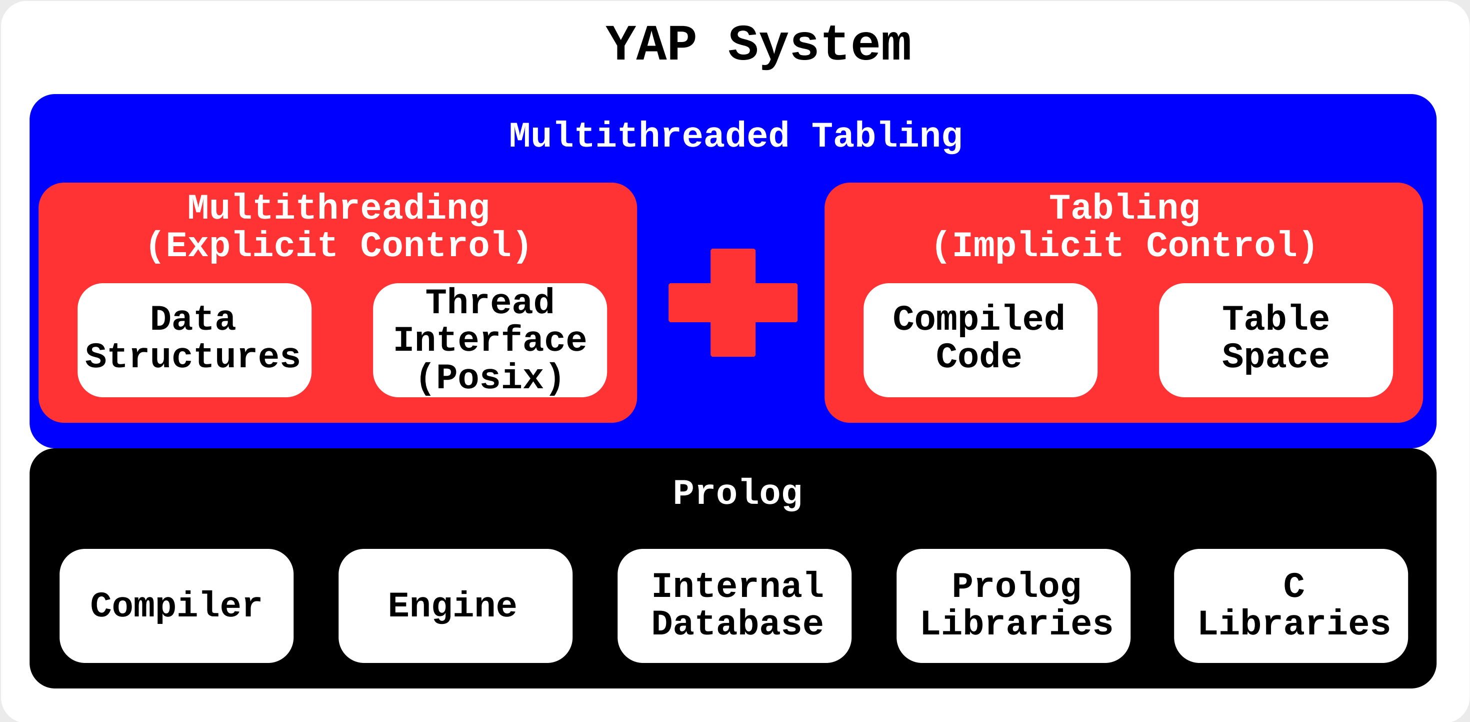
Multithreaded Tabling - Overview

- A novel **Multithreaded Tabling** framework aimed to support **concurrent evaluation** of tabled logic programs.



Multithreaded Tabling - Overview

- A novel **Multithreaded Tabling** framework aimed to support **concurrent evaluation** of tabled logic programs.



Multithreaded Tabling - Overview

- A novel **Multithreaded Tabling** framework aimed to support **concurrent evaluation** of tabled logic programs.

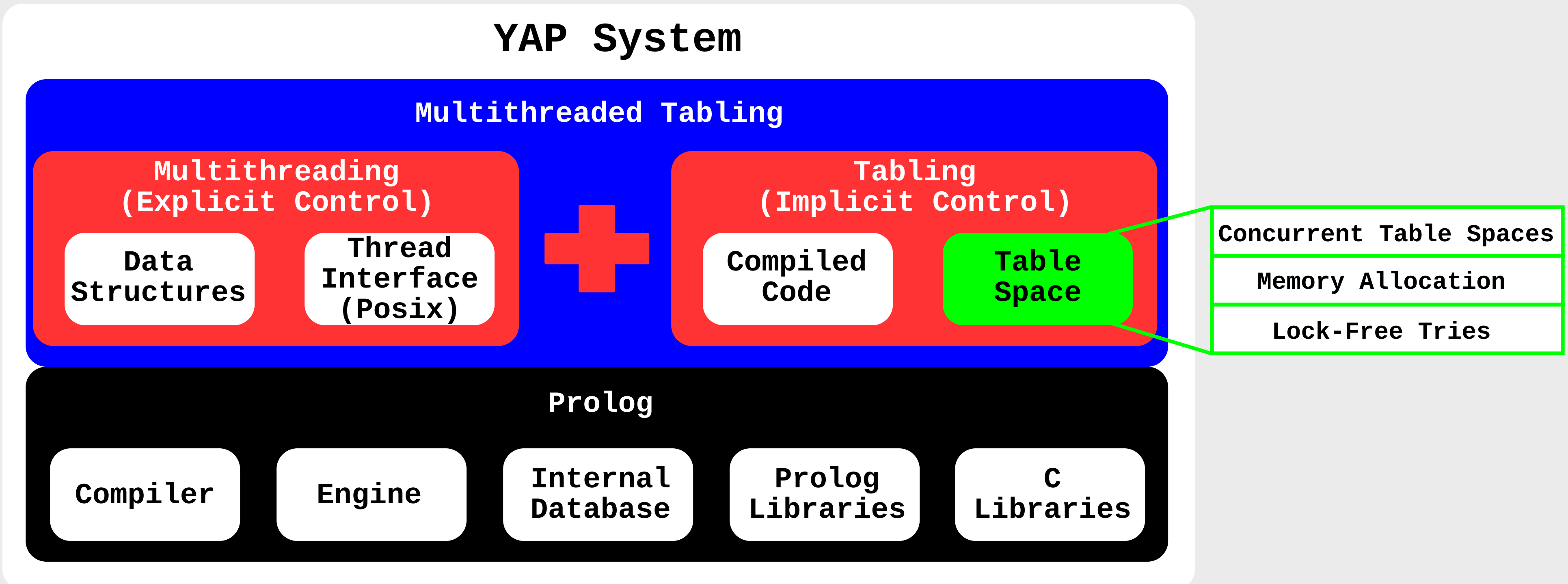


Table Space - Example

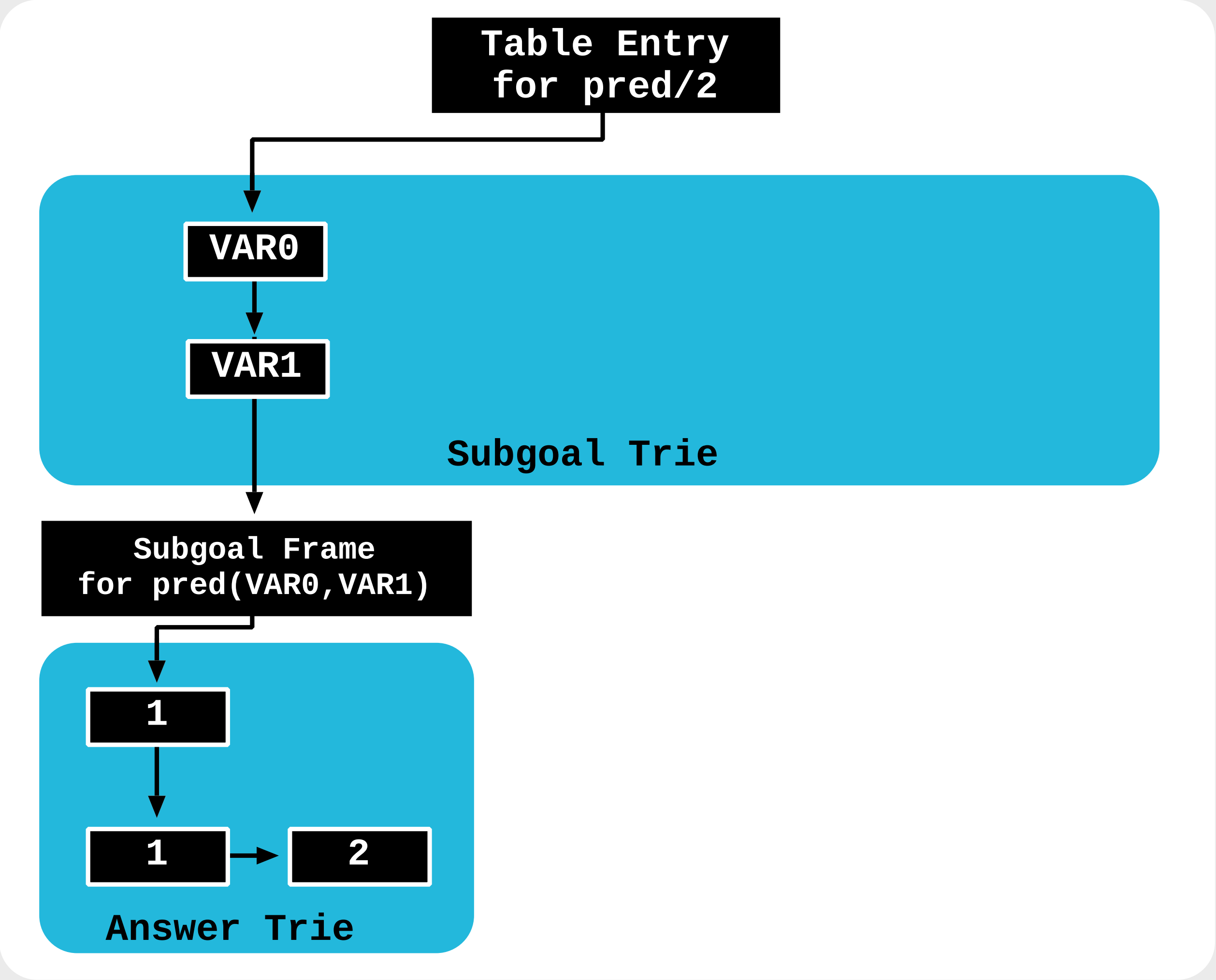
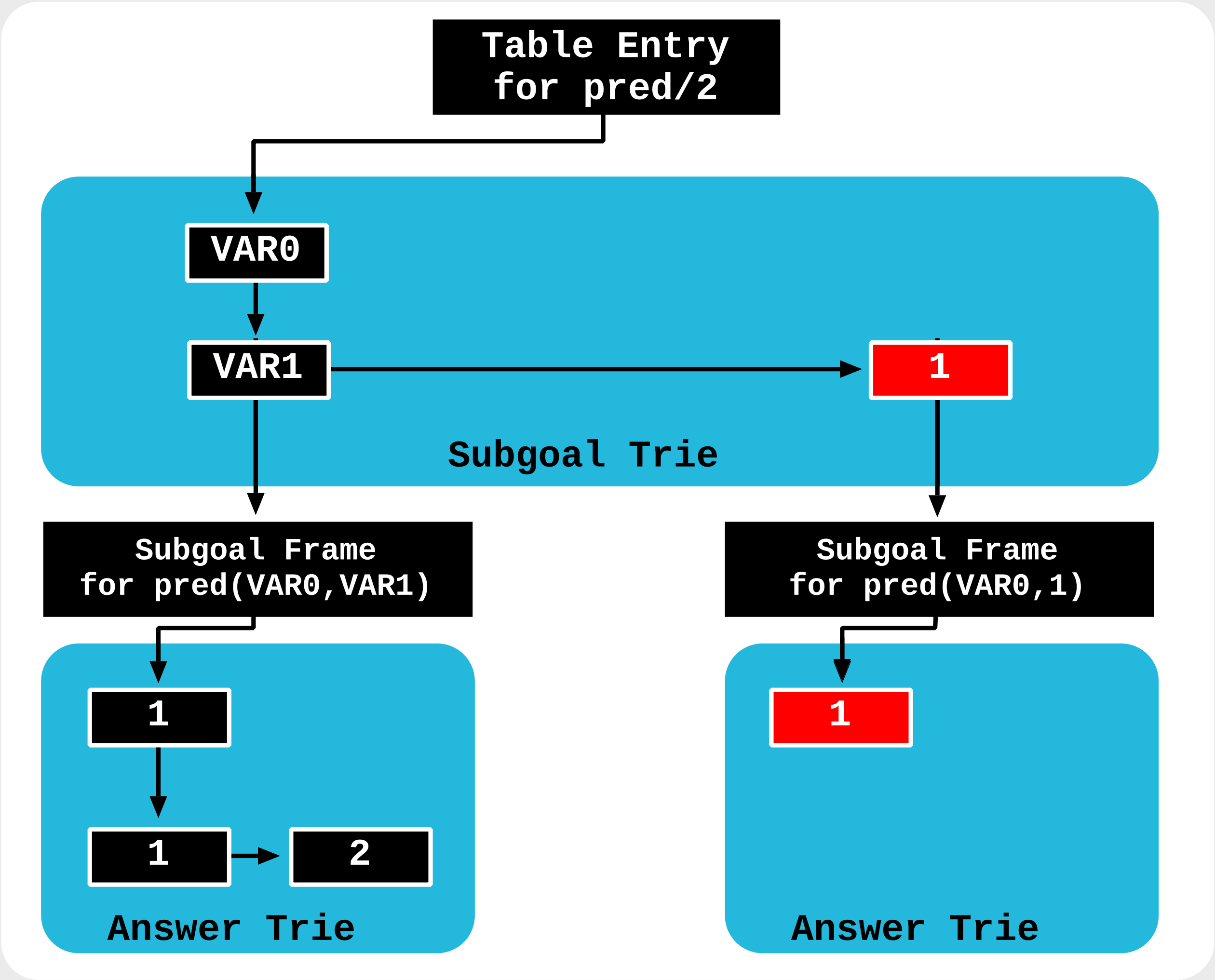
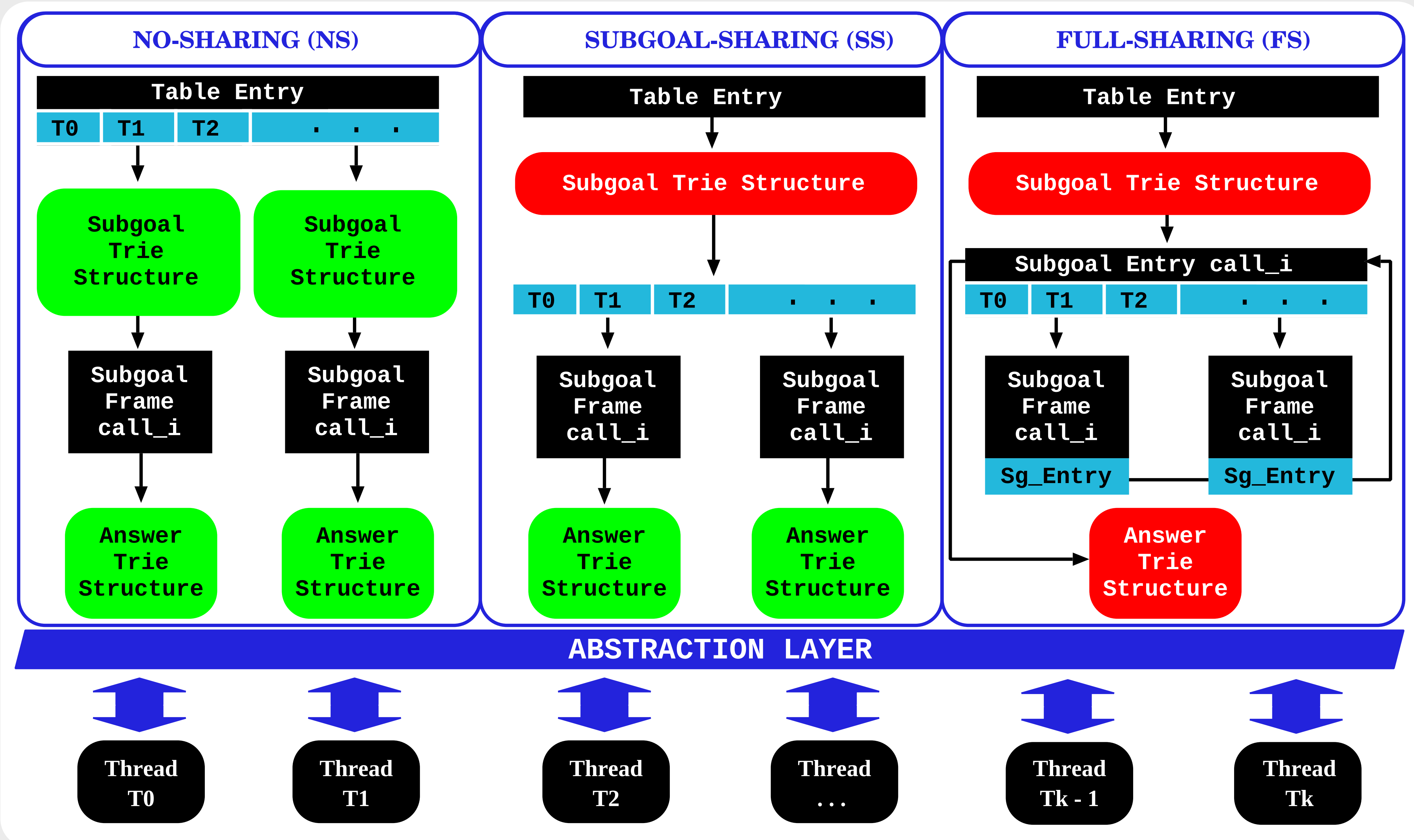


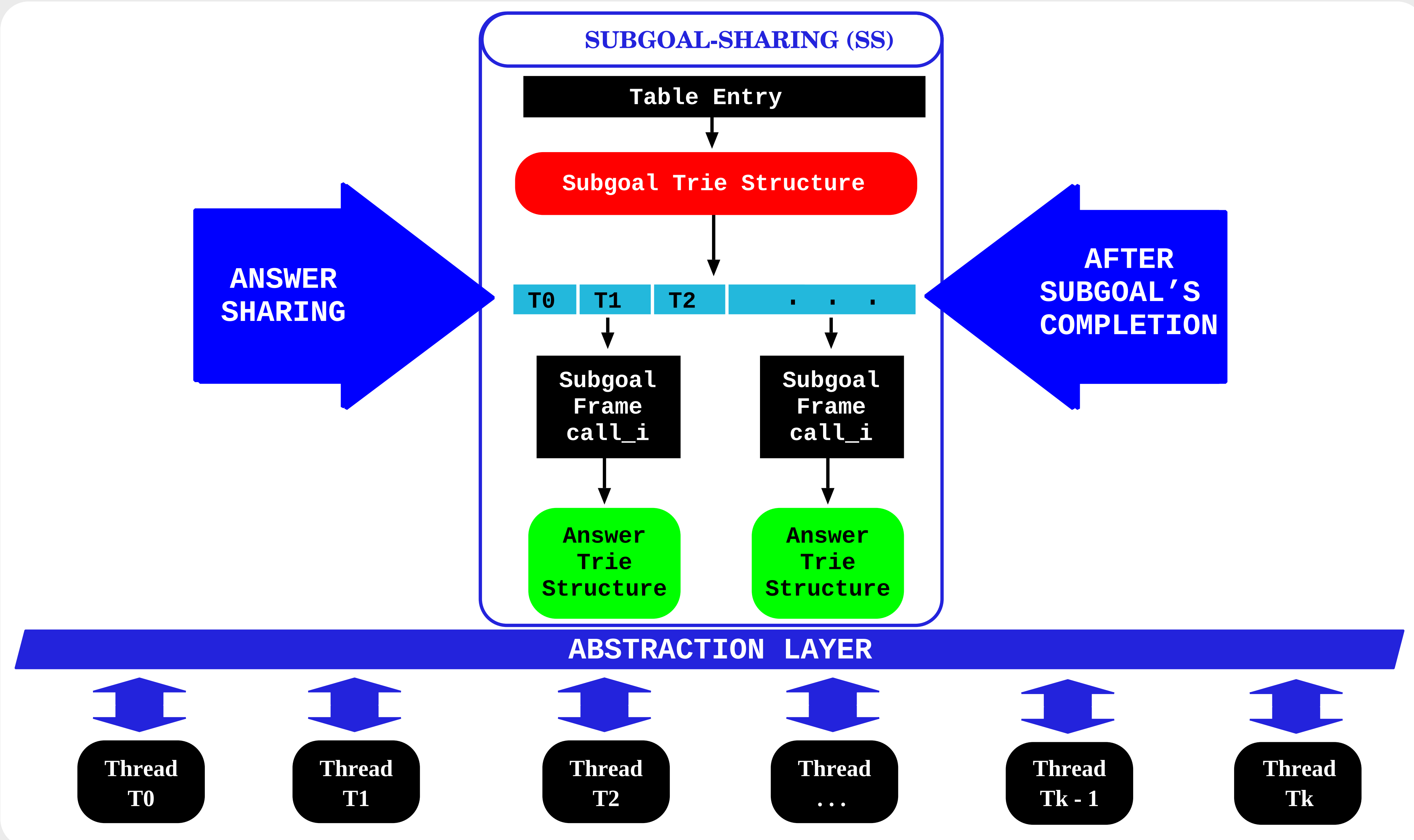
Table Space - Example



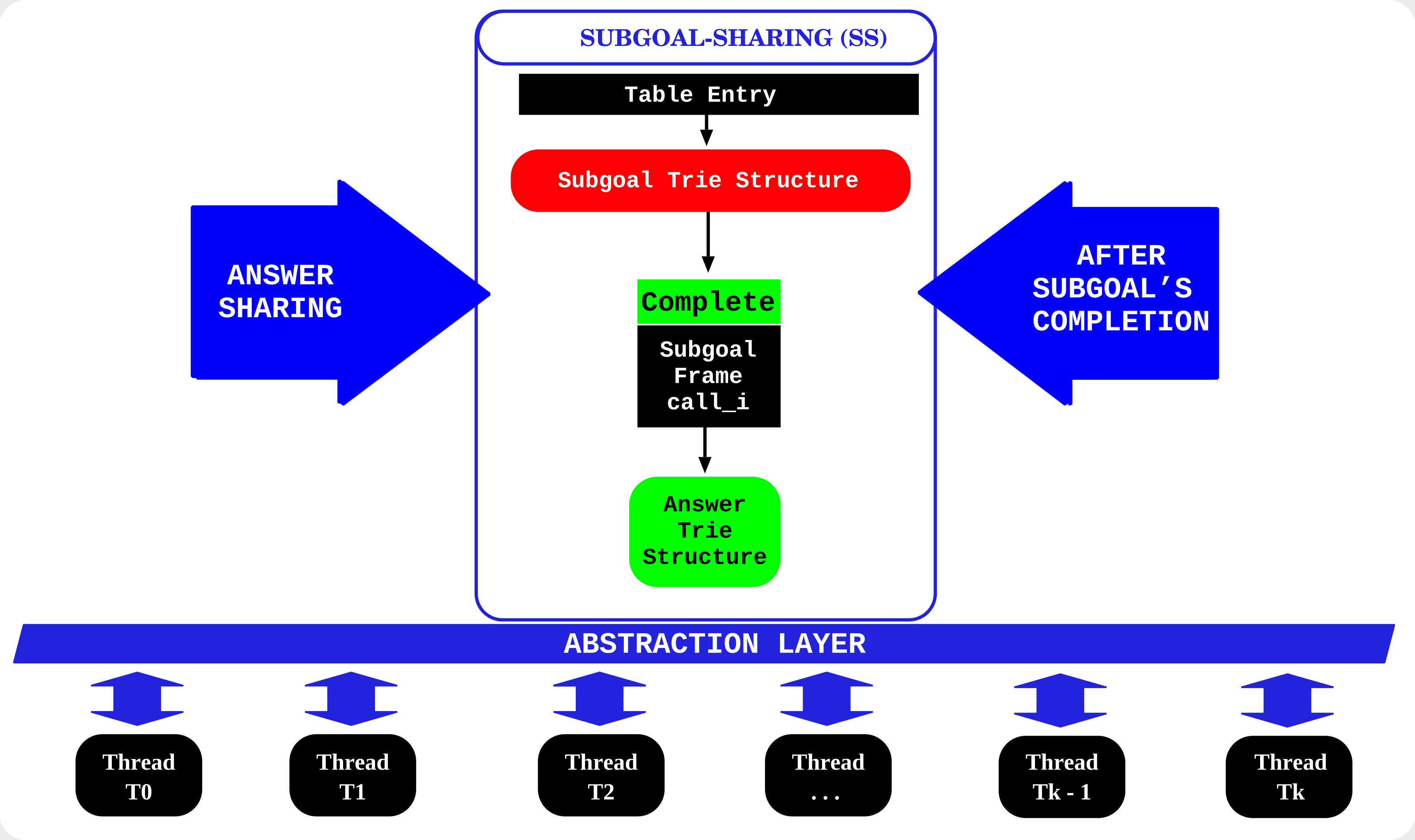
YapTab-Mt - Internal Architecture



YapTab-Mt - Internal Architecture

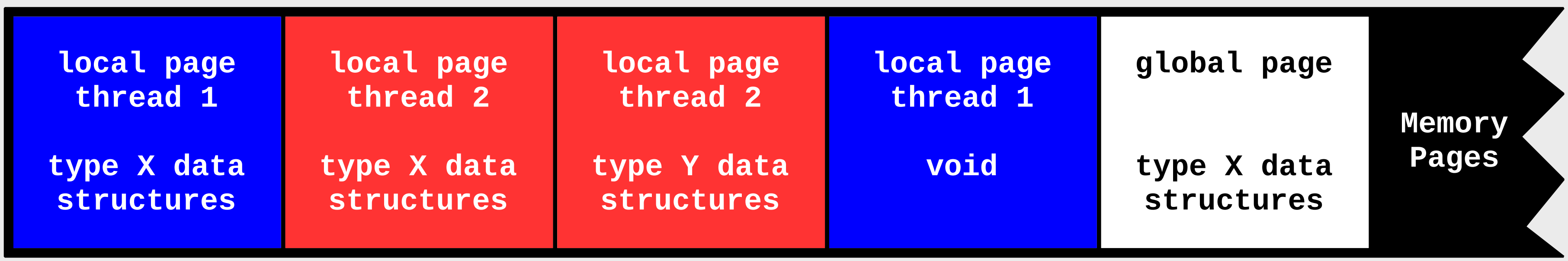


YapTab-Mt - Internal Architecture



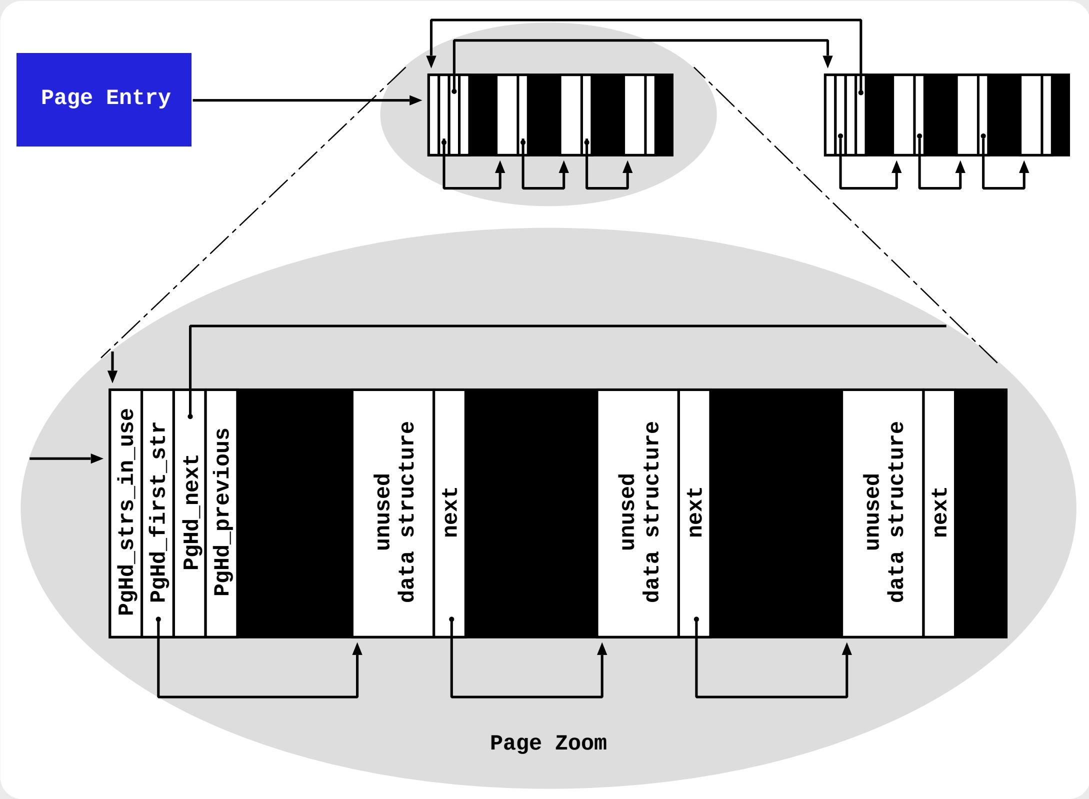
Concurrent Memory Allocation

- **Local** and **Global Page Heaps** per object type.
- **Global** and **Local Void Heaps** for the allocation of objects when **Local Page Heaps** run empty.
- **Global Page Heaps** used for the deallocation of shared objects.
- Allocation/Deallocation of objects is always done via Local Page Heaps, except for the main thread that performs garbage collection on the Global Page Heaps.



Concurrent Memory Allocation

- **Advantages**:
  - **Improve** data **locality**.
  - **Reduce synchronization** in **allocation** of **new objects**.
  - **Reduce** the **dependency** of the **operating system's memory allocator** performance.



Lock-Free Tries - Motivation

- **Our initial approach** to deal with concurrency was to use **locks**:
  - Lock Type:
    - Standard Locks.
    - Try-Locks.
  - Lock Location:
    - Field per trie node.
    - Global array of lock entries.
- However ... **lock-based data structures** have their performance **restrained** by multiple problems, such as: **priority inversion**, **convoying**, **contention**, **mutual exclusion**.

Lock-Free Tries - Motivation

- **Our initial approach** to deal with concurrency was to use **locks**:
  - Lock Type:
    - Standard Locks.
    - Try-Locks.
  - Lock Location:
    - Field per trie node.
    - Global array of lock entries.
- However ... **lock-based data structures** have their performance **restrained** by multiple problems, such as: **priority inversion**, **convoying**, **contention**, **mutual exclusion**.
- Take advantage of the **CAS (Compare-and-Swap)** operation, to reduce the granularity of the **synchronization**.
  - Nowadays **can be found** on many of the **common architectures**.
  - At the **heart** of many **lock-free objects**.

U.PORTO FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Lock-Free Tries - Motivation

- **Lock-free linearizable objects** permit a **greater concurrency** since **semantically consistent** (non-interfering) operations **may execute in parallel**.
- Several **lock-free models** do exist:
  - Shalev and Shavit **Split-Ordered Lists**
  - Prokopec **Concurrent Tries**
  - Cliff's **Non-Blocking Hash Tables**.
- However ... **none** of the existent models is specifically **aimed** for an environment with the **characteristics** of our **tabling framework**.
  - Support for the **concurrent deletion** of nodes **increases** the **complexity** of the models.

Lock-Free Tries - Internals

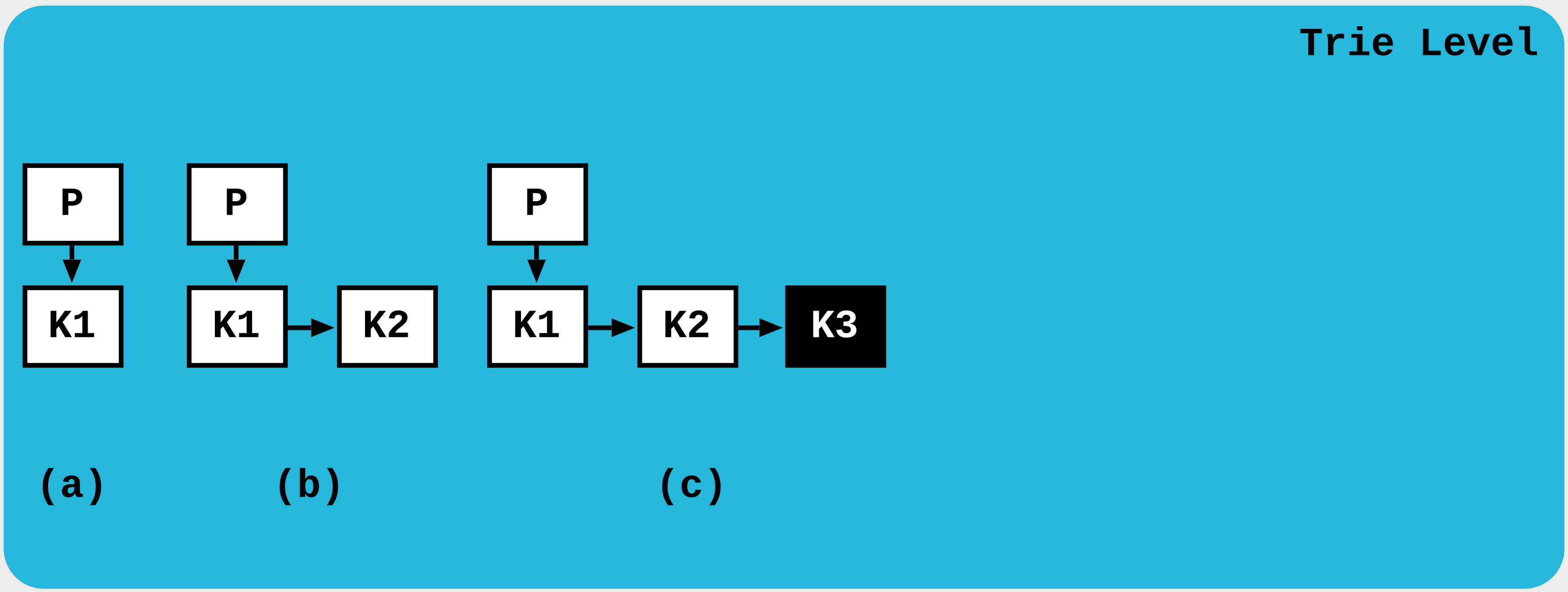
- A **trie level** is defined by a **parent (P)** node and at least one **child (K)** node.
- Only **lookup** and **insert** operations are executed.
- **Insertion** of new nodes is done in a **chain**, until a **threshold** is achieved and afterwards a **hashing system** is included in the trie level.

Lock-Free Tries - Internals

- A **trie level** is defined by a **parent (P)** node and at least one **child (K)** node.
- Only **lookup** and **insert** operations are executed.
- **Insertion** of new nodes is done in a **chain**, until a **threshold** is achieved and afterwards a **hashing system** is included in the trie level.

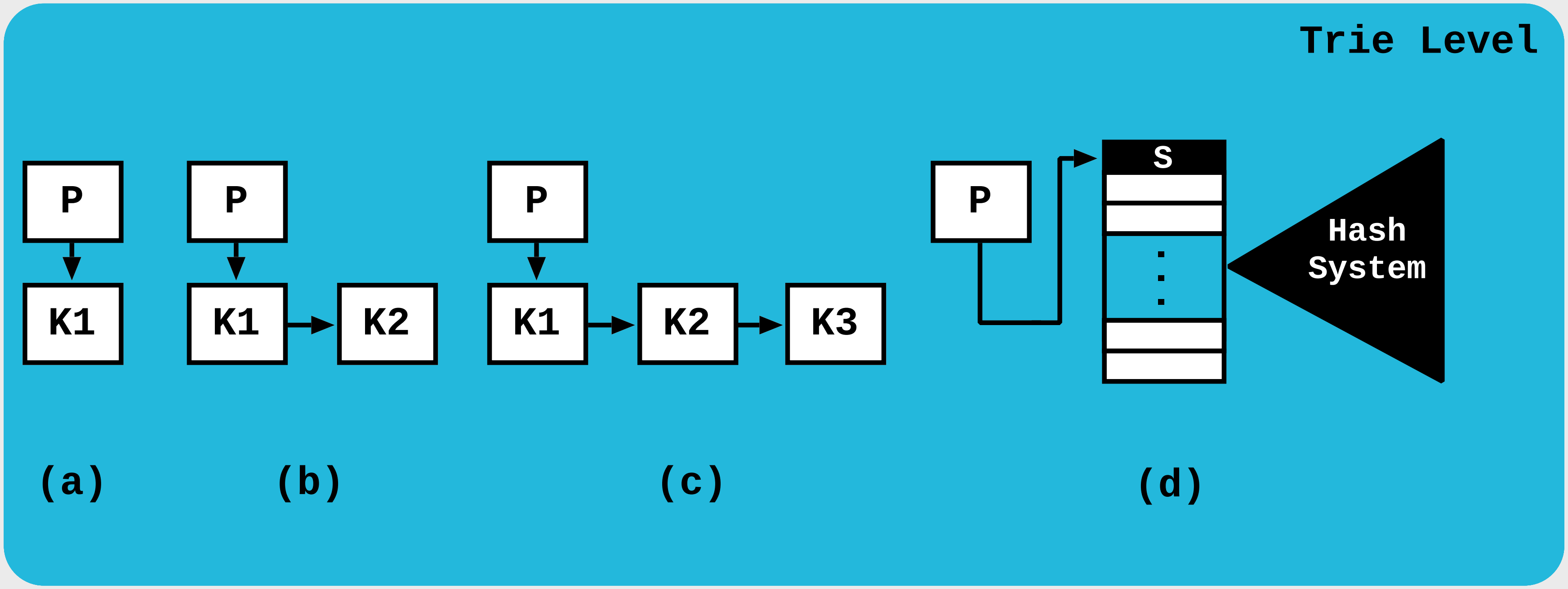
Lock-Free Tries - Internals

- A **trie level** is defined by a **parent (P)** node and at least one **child (K)** node.
- Only **lookup** and **insert** operations are executed.
- **Insertion** of new nodes is done in a **chain**, until a **threshold** is achieved and afterwards a **hashing system** is included in the trie level.



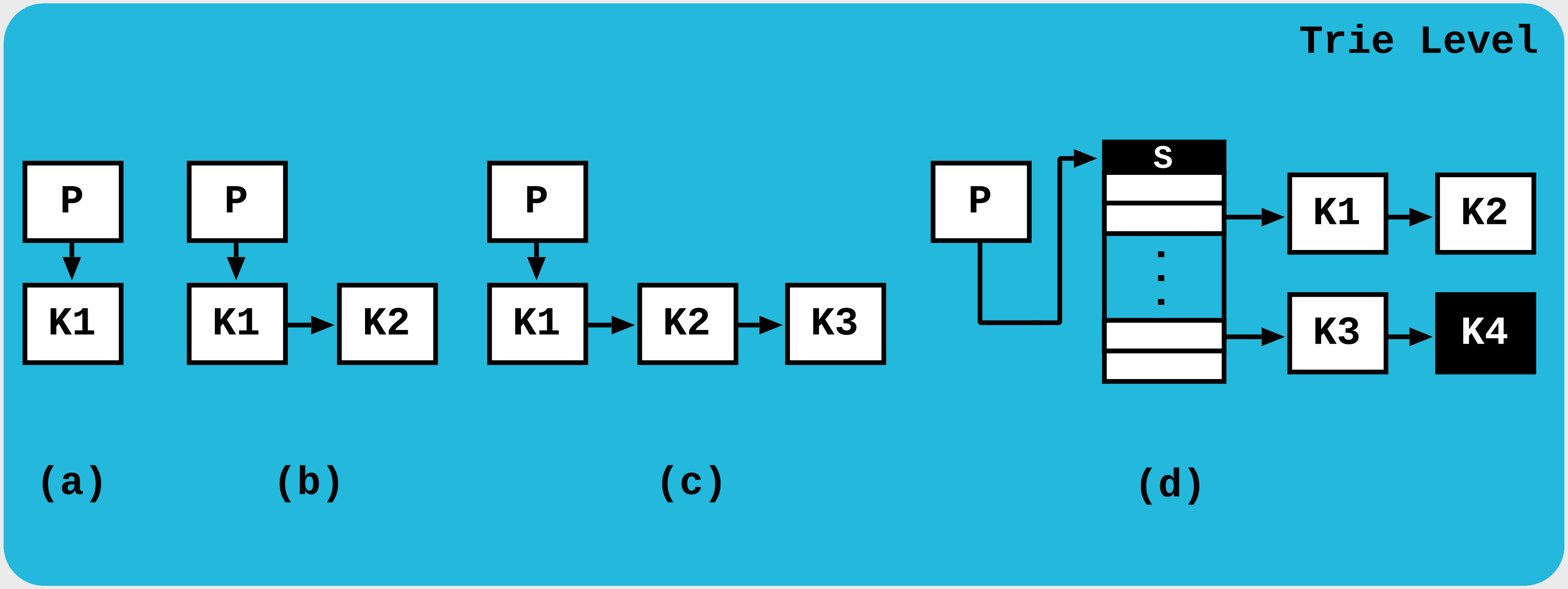
Lock-Free Tries - Internals

- A **trie level** is defined by a **parent (P)** node and at least one **child (K)** node.
- Only **lookup** and **insert** operations are executed.
- **Insertion** of new nodes is done in a **chain**, until a **threshold** is achieved and afterwards a **hashing system** is included in the trie level.

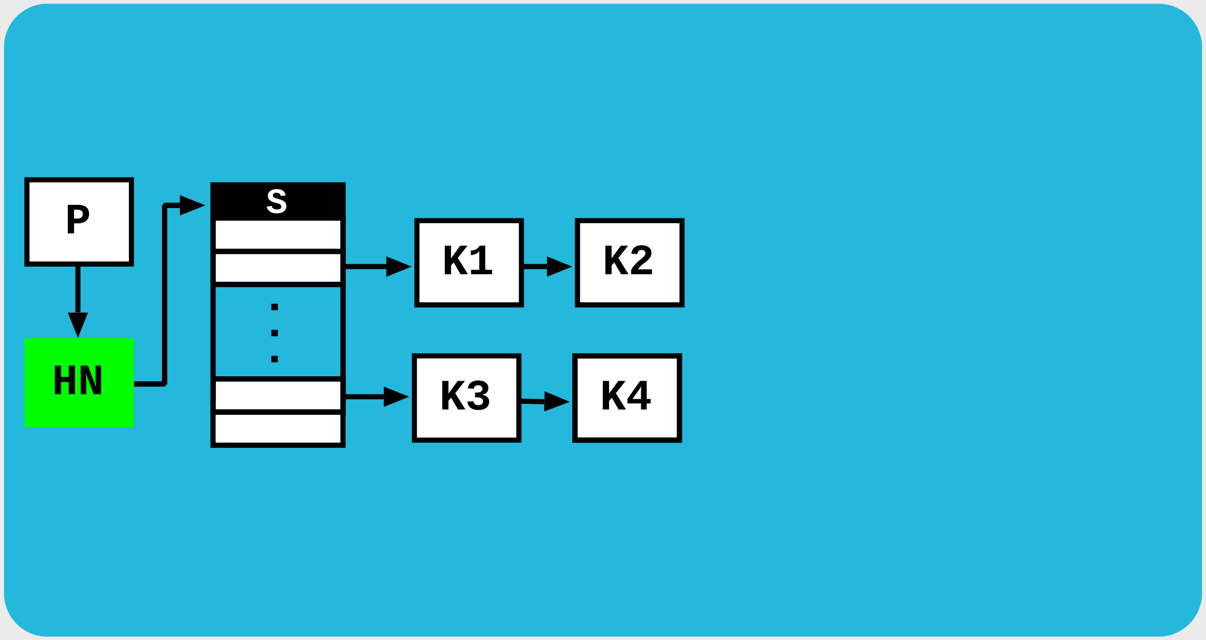


Lock-Free Tries - Internals

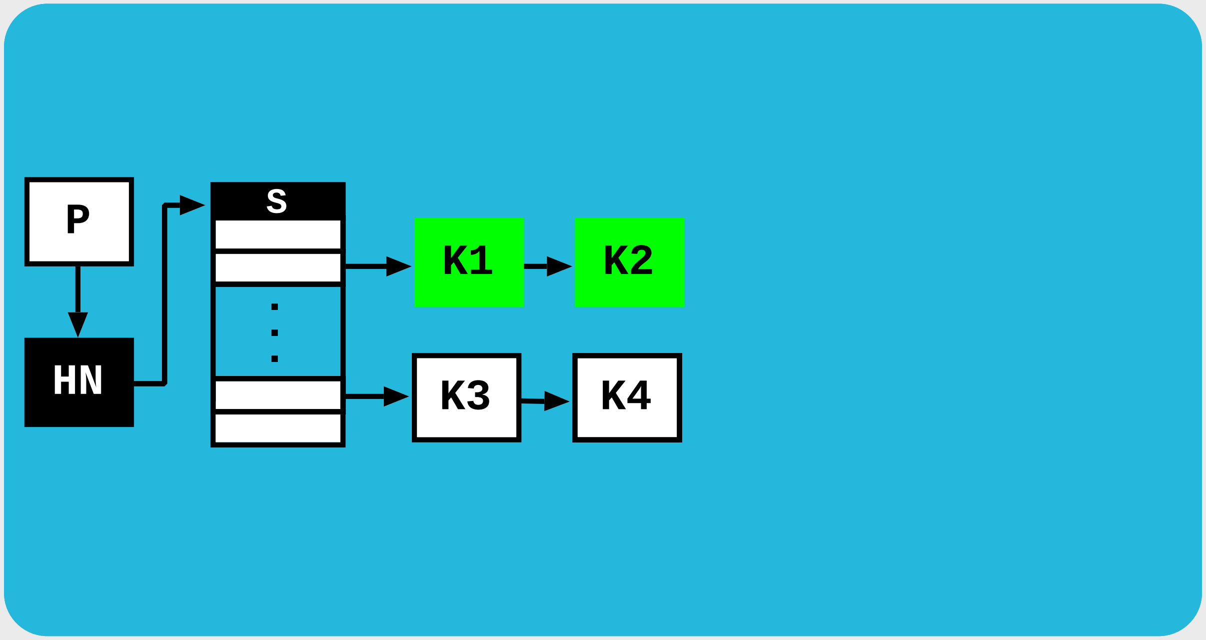
- A **trie level** is defined by a **parent (P)** node and at least one **child (K)** node.
- Only **lookup** and **insert** operations are executed.
- **Insertion** of new nodes is done in a **chain**, until a **threshold** is achieved and afterwards a **hashing system** is included in the trie level.



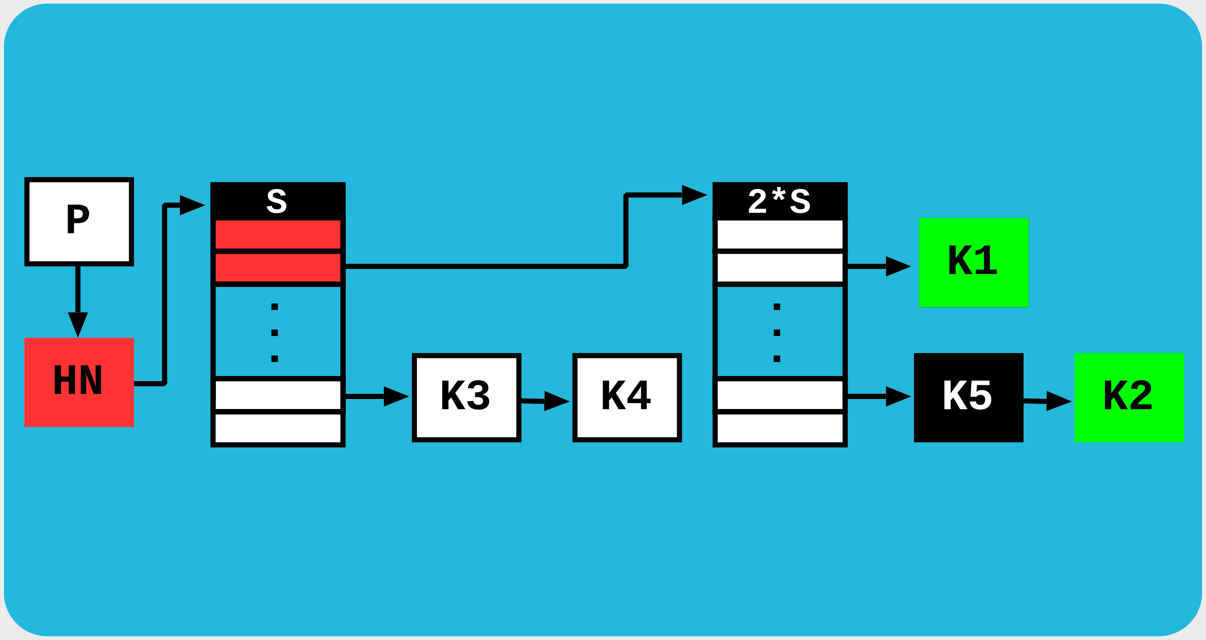
Lock-Free Tries - The First Approach



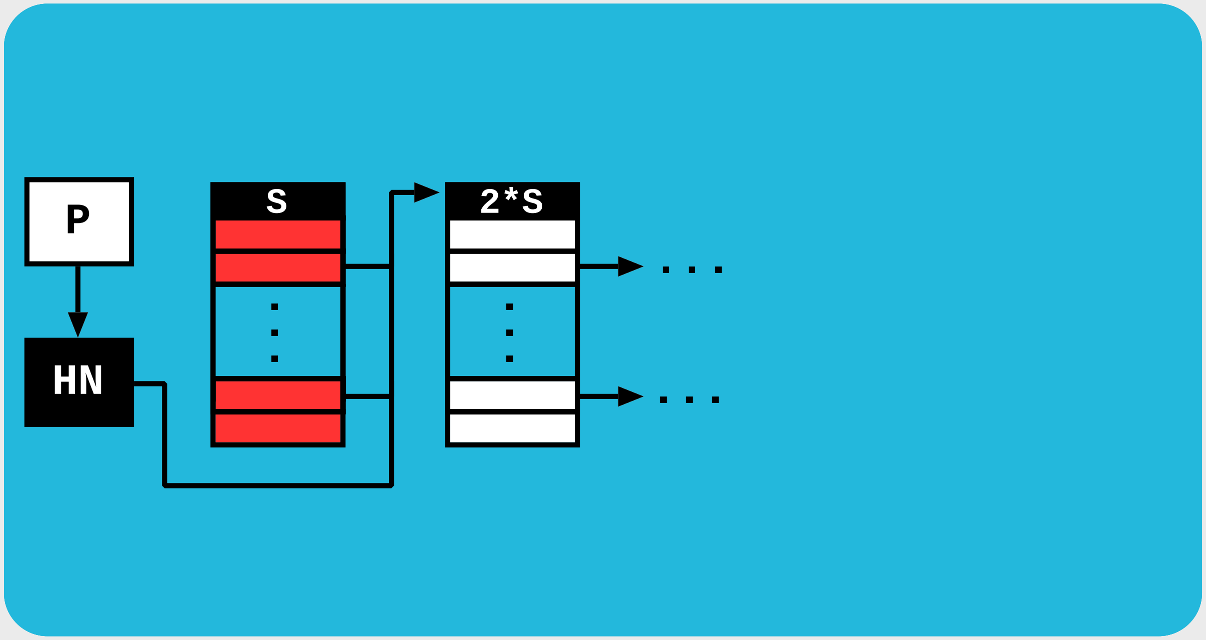
Lock-Free Tries - The First Approach



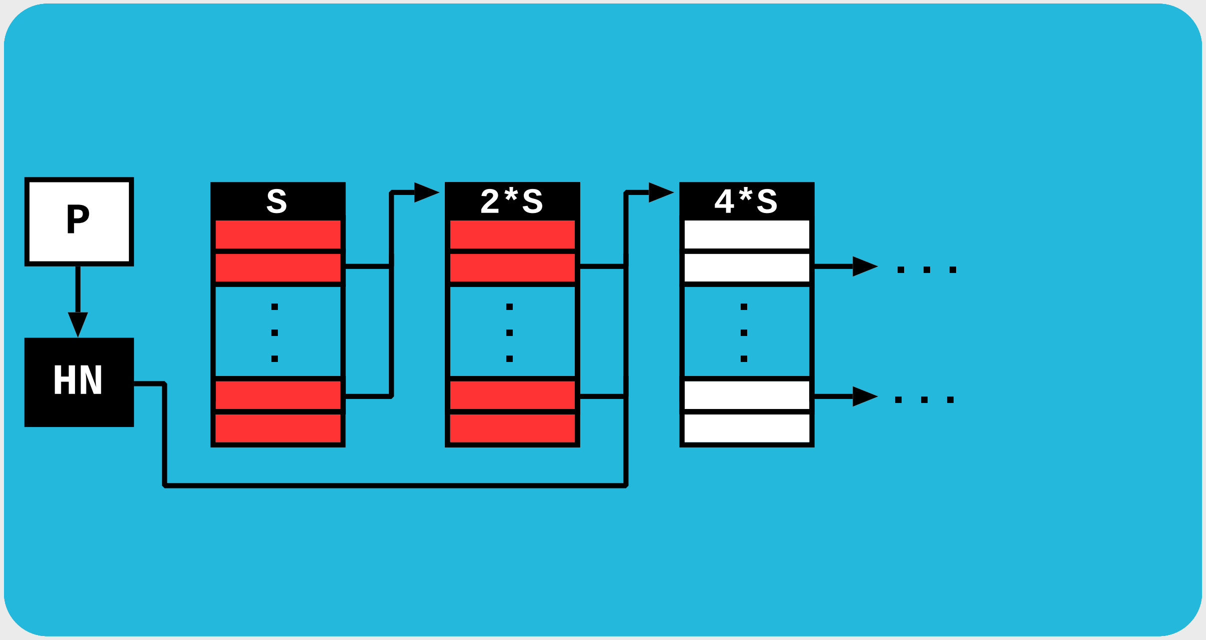
Lock-Free Tries - The First Approach



Lock-Free Tries - The First Approach

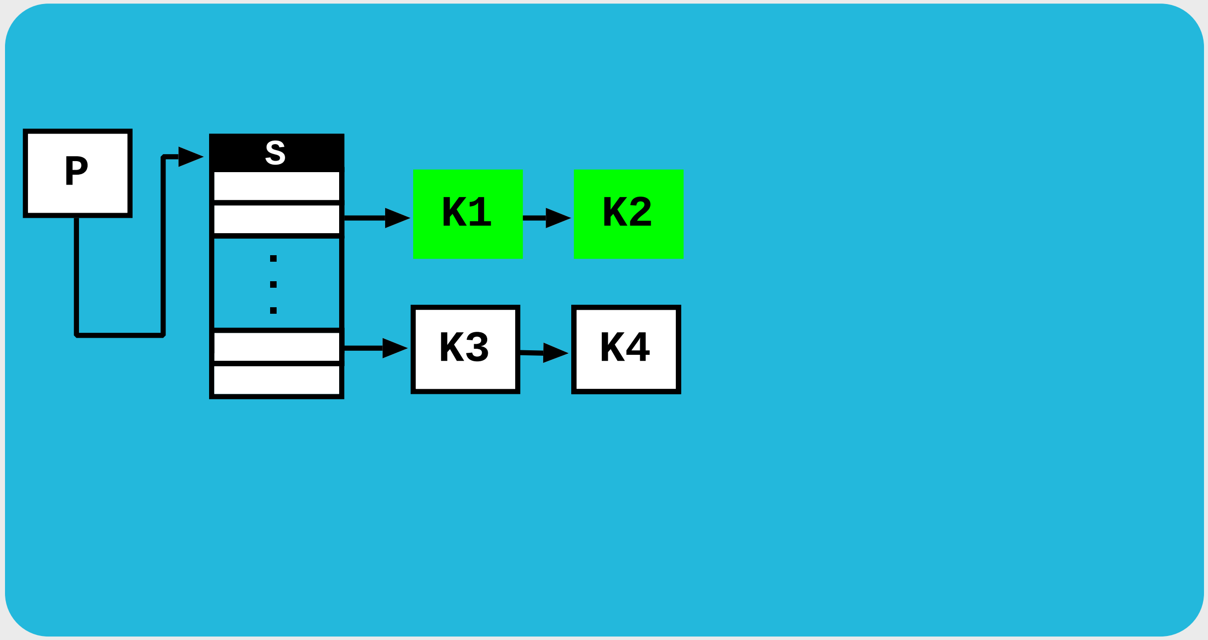


Lock-Free Tries - The First Approach

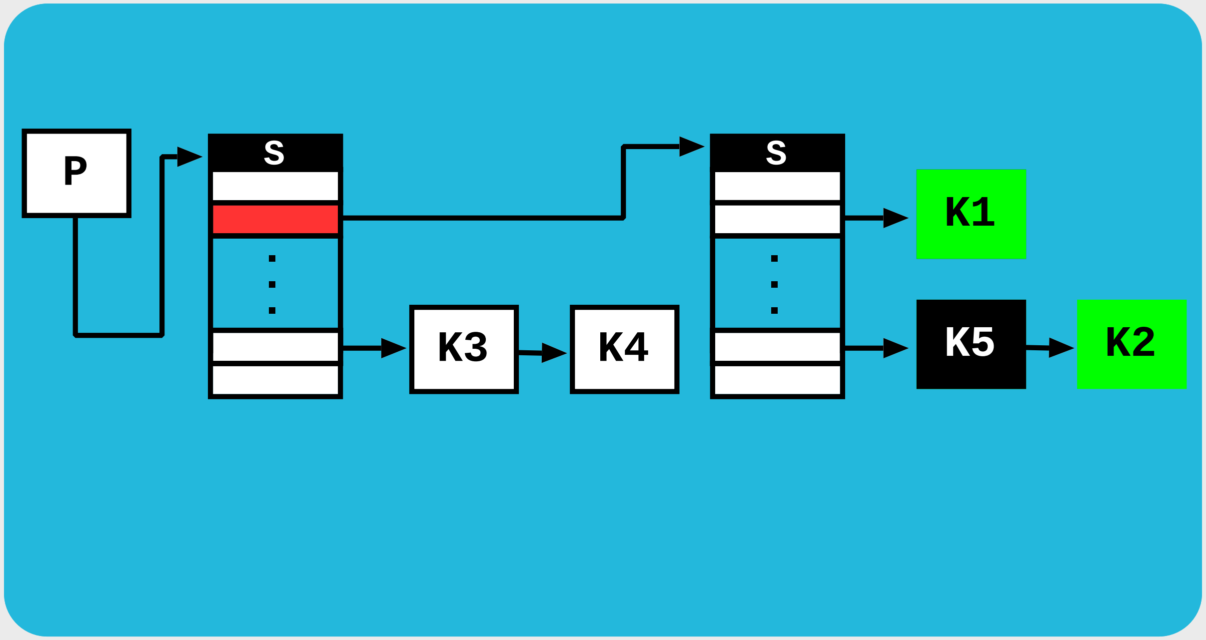


- **Disadvantages**:
  - **False-Sharing** effects: **concurrency points** at **bucket entries**.
  - **Hash Expansion**: **all bucket entries** are **expanded regardless** of the **number of nodes** within their chains.
  - **Bucket arrays with different sizes**: **ineffective integration** in a **page-based memory allocator**.

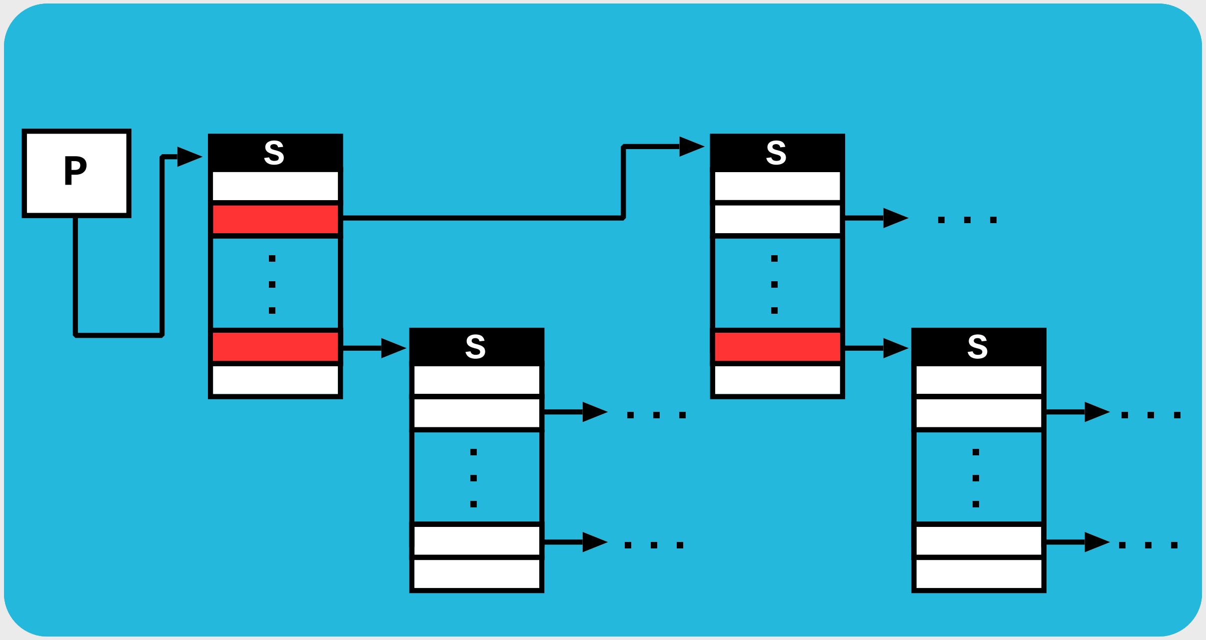
Lock-Free Hash Tries - The Second Approach



Lock-Free Hash Tries - The Second Approach



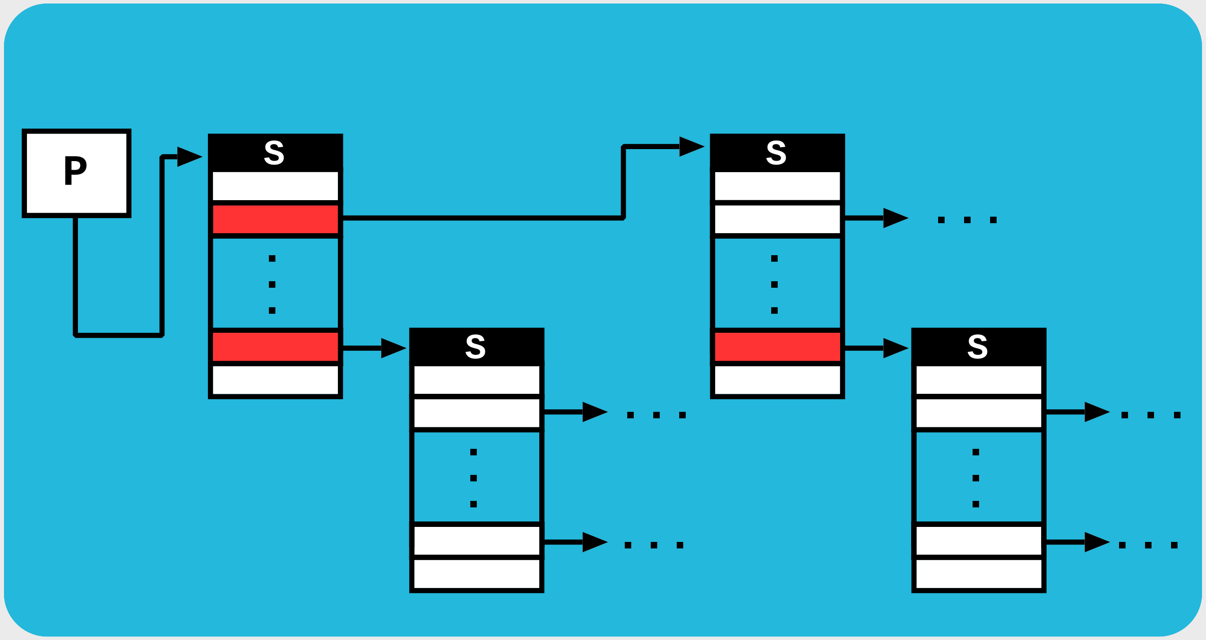
Lock-Free Hash Tries - The Second Approach



➤ **Advantages**:

- **Reduce False-Sharing** effects: **concurrency points** are within chain nodes.
- **Hash Expansion**: Only the saturated buckets entries are expanded. We can also **define** an **upper-bound for number of nodes** within a **bucket chain**.
- **All Bucket arrays** have the **same size**.

Lock-Free Hash Tries - The Second Approach



- **Advantages**:
  - **Reduce False-Sharing** effects: **concurrency points** are within chain nodes.
  - **Hash Expansion**: Only the saturated buckets entries are expanded. We can also **define** an **upper-bound for number of nodes** within a **bucket chain**.
  - **All Bucket arrays** have the **same size**.
- Possible **Disadvantage**:
  - The search operation might be slower.

Experimental Results - Worst Case Scenarios

Threads		NS	
		Initial	Current
	Min	1.00	0.53
1	Avg	1.00	0.78
	Max	1.00	1.06
	Min	1.07	0.66
8	Avg	2.35	0.85
	Max	5.06	1.12
	Min	1.02	0.85
16	Avg	5.13	0.98
	Max	11.17	1.16
	Min	1.24	0.91
24	Avg	8.42	1.15
	Max	18.33	1.72
	Min	1.33	**1.05**
32	Avg	12.94	**1.51**
	Max	26.67	**2.52**

Experimental Results - Worst Case Scenarios

Threads		NS		SS	
		Initial	Current	Initial	Current
1	Min	1.00	0.53	0.99	0.54
	Avg	1.00	0.78	1.11	0.84
	Max	1.00	1.06	1.40	1.04
8	Min	1.07	0.66	1.00	0.66
	Avg	2.35	0.85	2.50	0.92
	Max	5.06	1.12	5.37	1.20
16	Min	1.02	0.85	1.09	0.82
	Avg	5.13	0.98	5.01	1.04
	Max	11.17	1.16	11.19	1.31
24	Min	1.24	0.91	1.22	1.02
	Avg	8.42	1.15	8.02	1.22
	Max	18.33	1.72	18.50	1.81
32	Min	1.33	**1.05**	1.32	**1.07**
	Avg	12.94	**1.51**	11.43	**1.54**
	Max	26.67	**2.52**	25.96	**2.52**

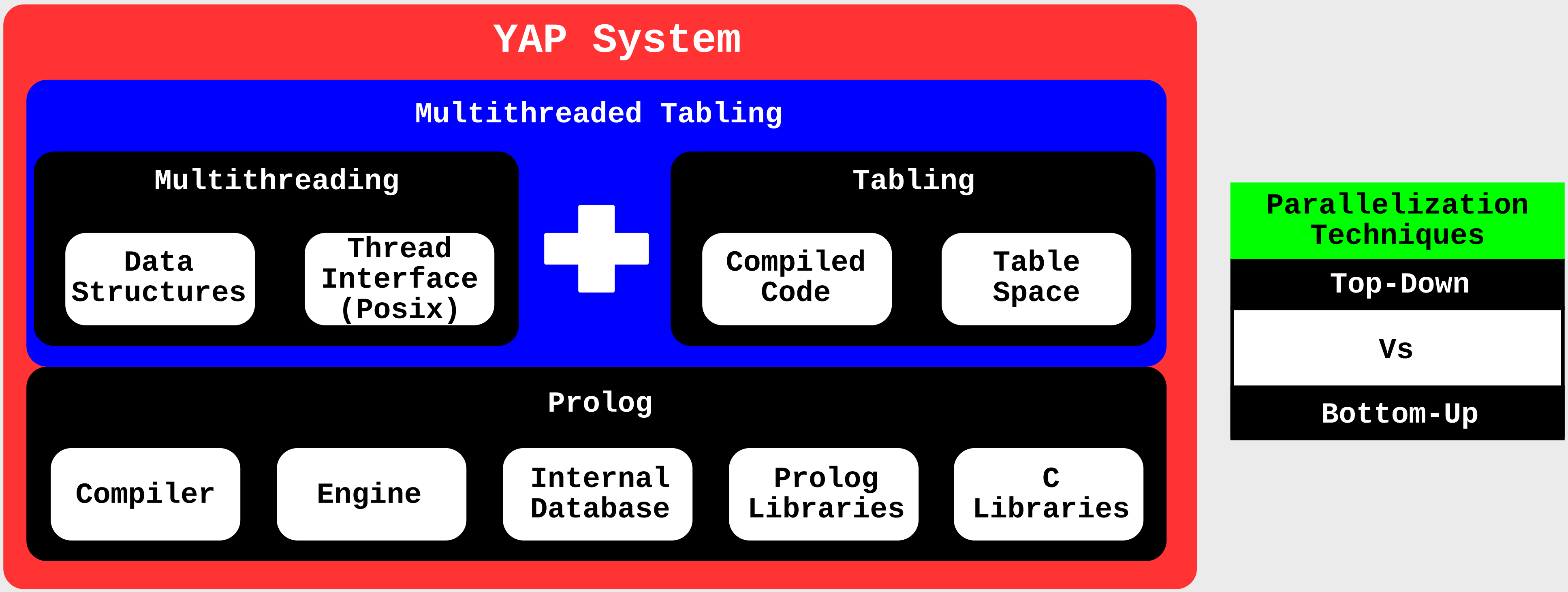
Experimental Results - Worst Case Scenarios

Threads		NS		SS		FS	
		Initial	Current	Initial	Current	Initial	Current
1	Min	1.00	0.53	0.99	0.54	1.05	1.01
	Avg	1.00	0.78	1.11	0.84	1.39	1.30
	Max	1.00	1.06	1.40	1.04	1.73	1.76
8	Min	1.07	0.66	1.00	0.66	1.07	1.16
	Avg	2.35	0.85	2.50	0.92	3.58	1.88
	Max	5.06	1.12	5.37	1.20	7.12	2.82
16	Min	1.02	0.85	1.09	0.82	1.06	1.17
	Avg	5.13	0.98	5.01	1.04	4.48	1.97
	Max	11.17	1.16	11.19	1.31	9.30	3.14
24	Min	1.24	0.91	1.22	1.02	1.27	1.16
	Avg	8.42	1.15	8.02	1.22	5.13	2.06
	Max	18.33	1.72	18.50	1.81	10.56	3.49
32	Min	1.33	**1.05**	1.32	**1.07**	1.36	**1.33**
	Avg	12.94	**1.51**	11.43	**1.54**	5.88	**2.24**
	Max	26.67	**2.52**	25.96	**2.52**	12.32	**3.71**

U.PORTO FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

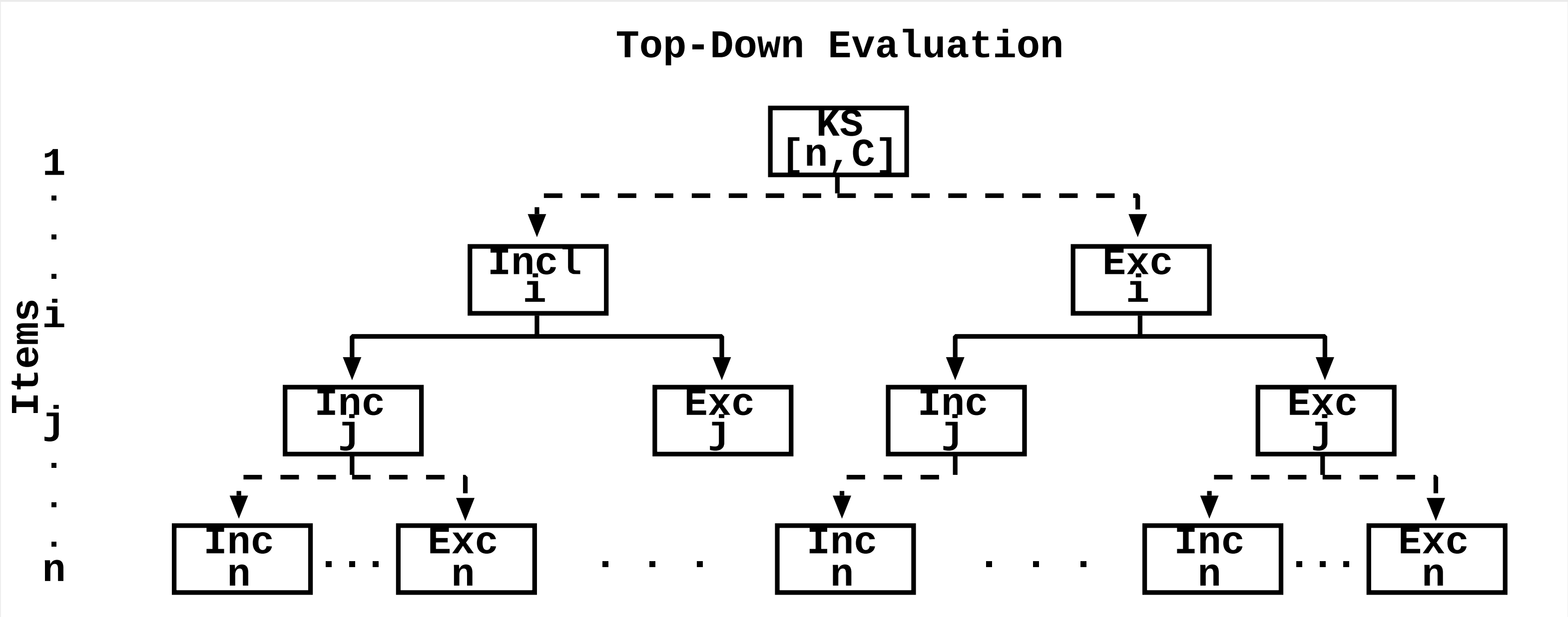
Applications

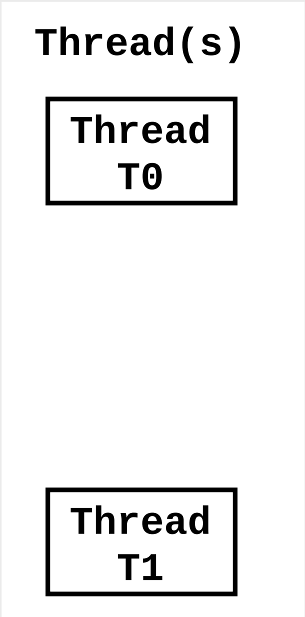
- Used the **Subgoal-Sharing** design to **scale the execution** of two well-know **dynamic programming problems** that can be found in many domains:
  - **0-1 Knapsack**: logistics, manufacturing, finance or telecommunications.
  - **Longest Common Subsequence (LCS)**: sequence alignment, which is a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



0-1 Knapsack Problem (Top-Down)

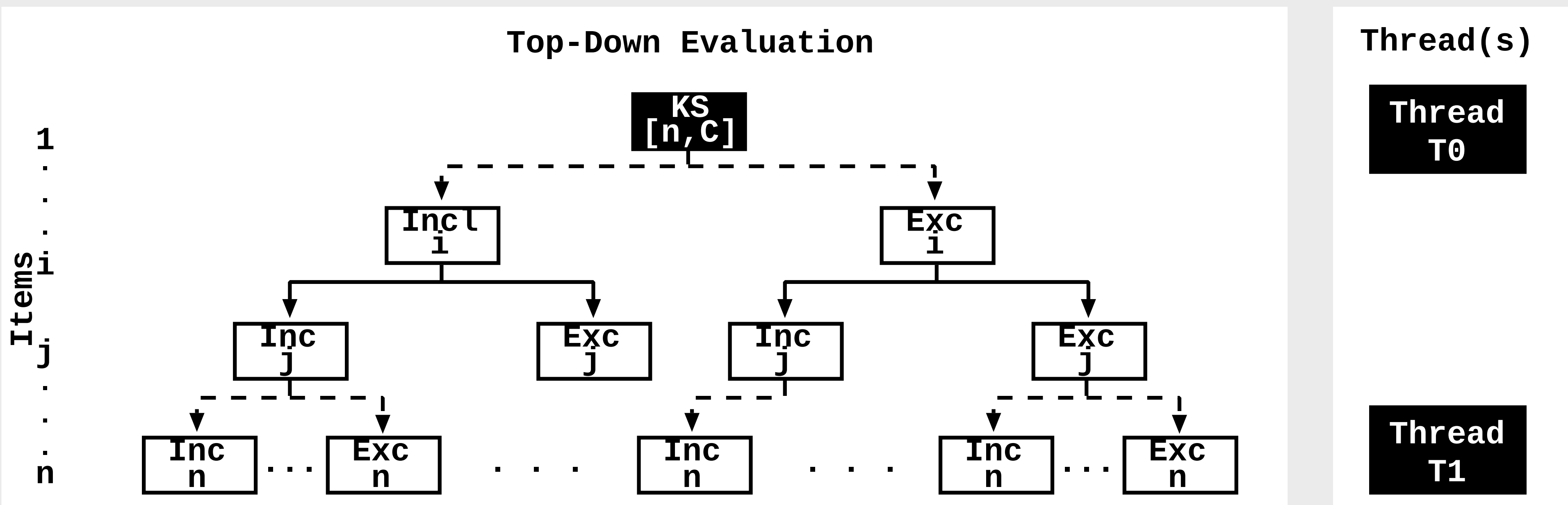
- An **item** is **included or excluded** from the **Knapsack** whether it **belongs or not** to the **best solution** of the problem.
- Thread(s) scheduling:
  - Threads **begin** their evaluation in the **top query**.
  - **Disperse threads** through the **evaluation tree** using **random branch orders**.





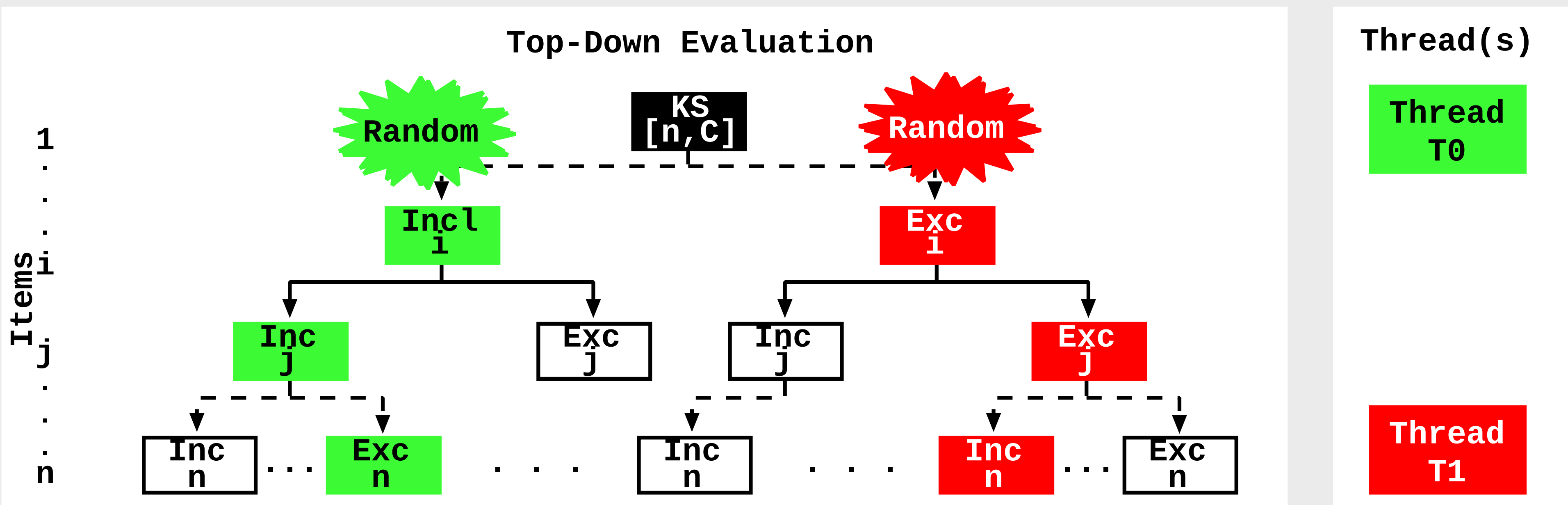
0-1 Knapsack Problem (Top-Down)

- 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**.



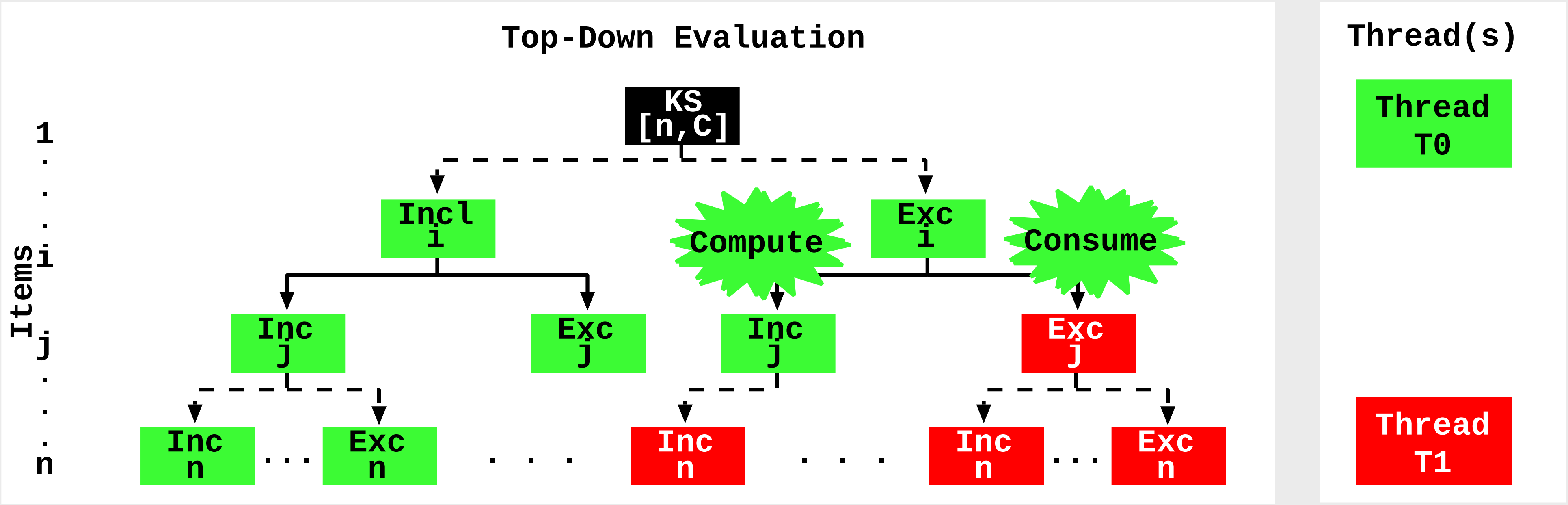
0-1 Knapsack Problem (Top-Down)

- 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**.



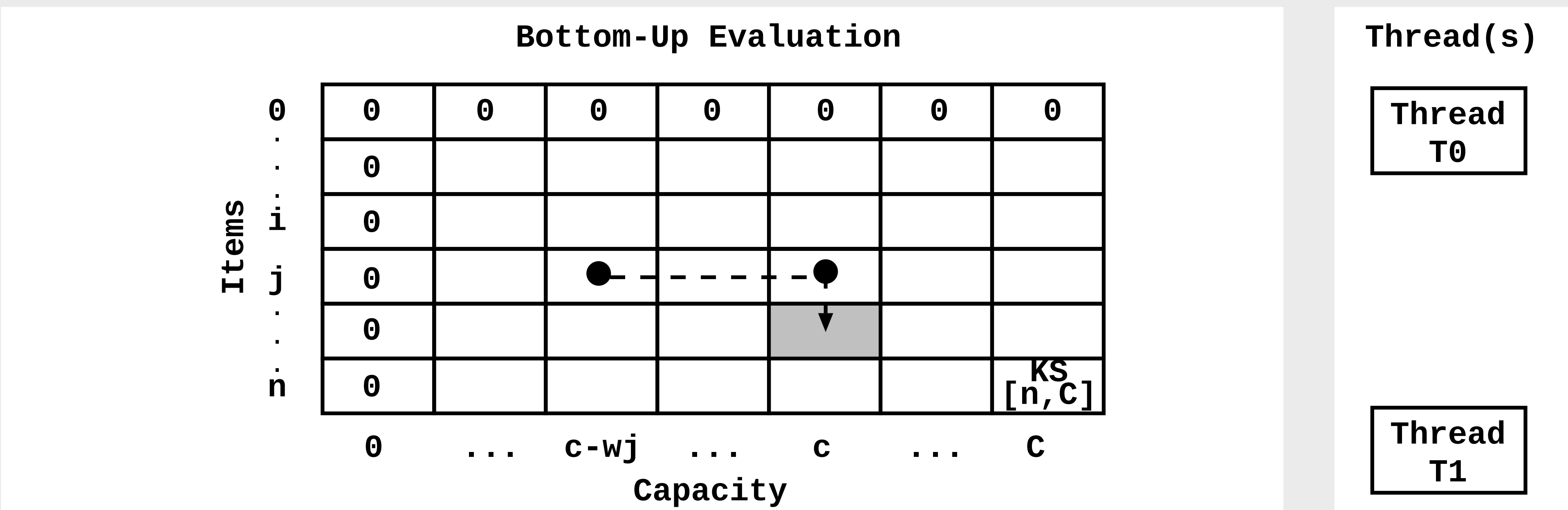
0-1 Knapsack Problem (Top-Down)

- 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**.



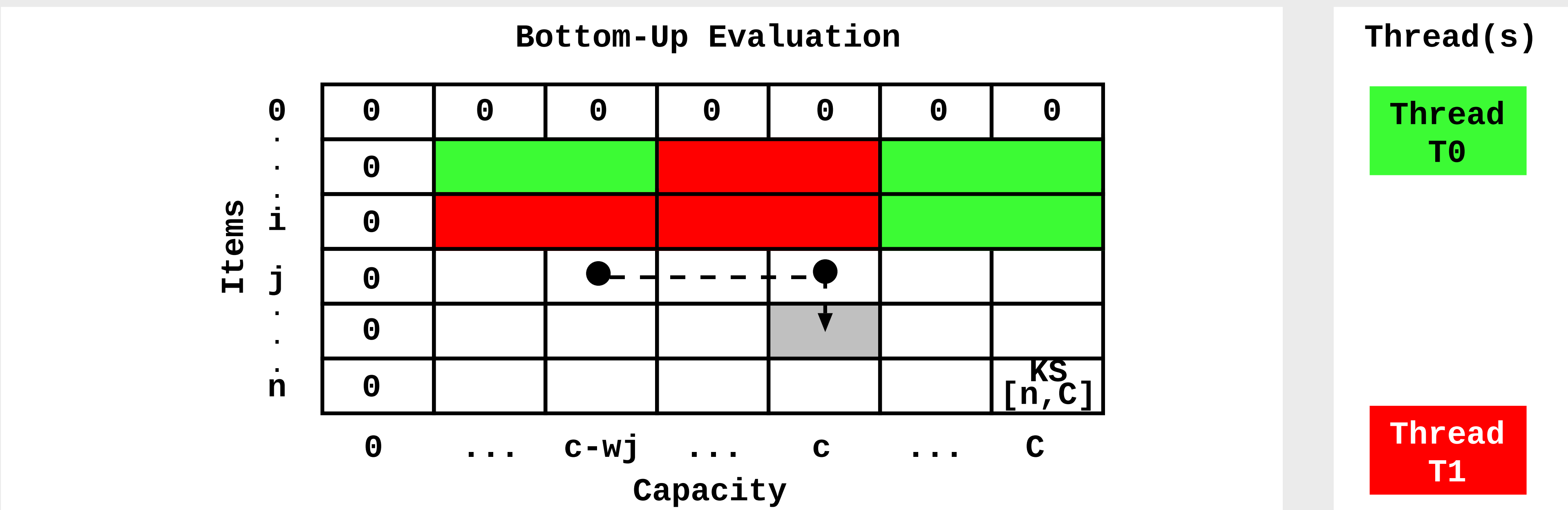
0-1 Knapsack Problem (Bottom-Up)

- 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**.



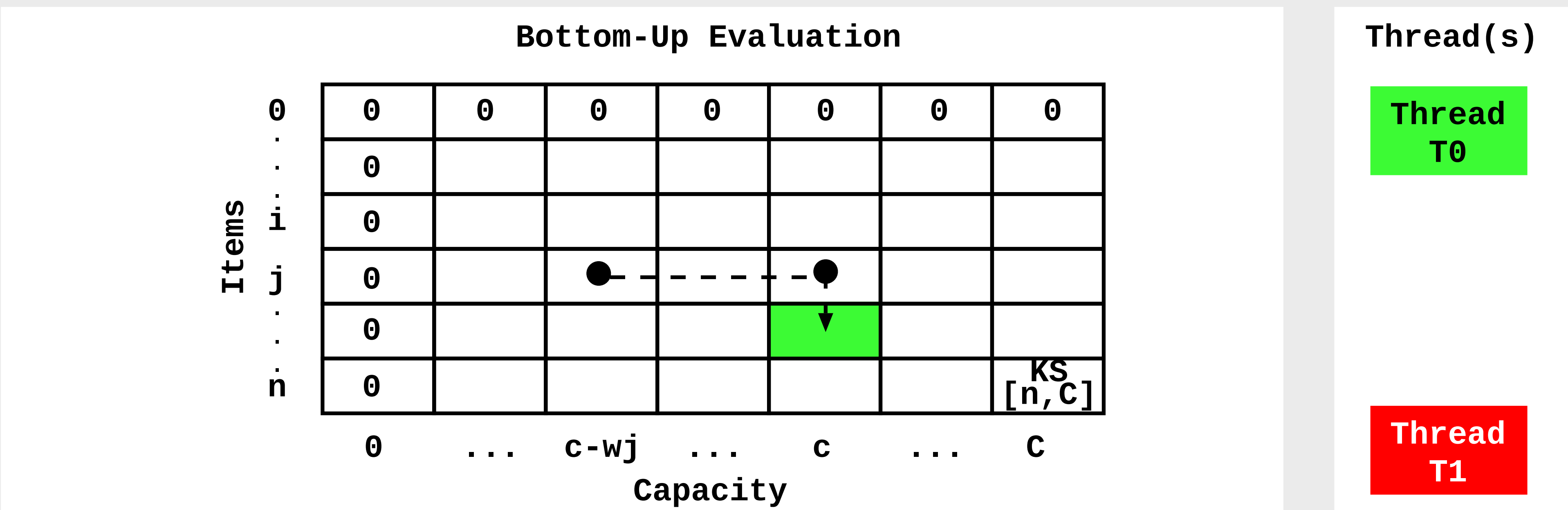
0-1 Knapsack Problem (Bottom-Up)

- 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**.



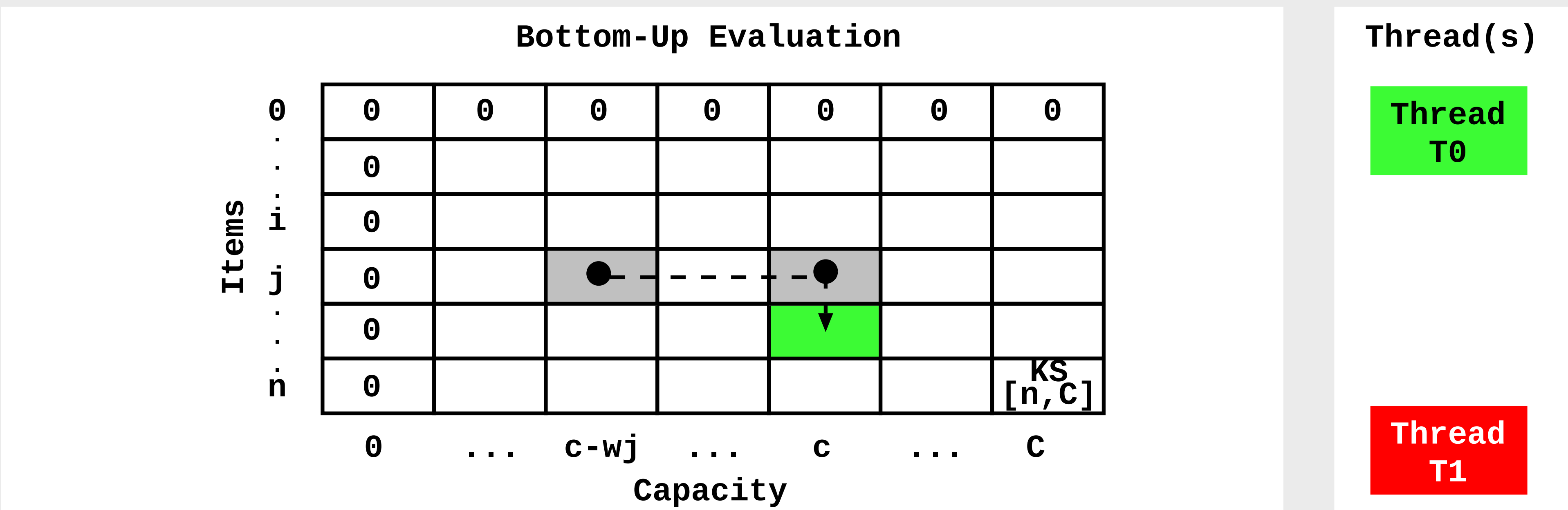
0-1 Knapsack Problem (Bottom-Up)

- 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**.



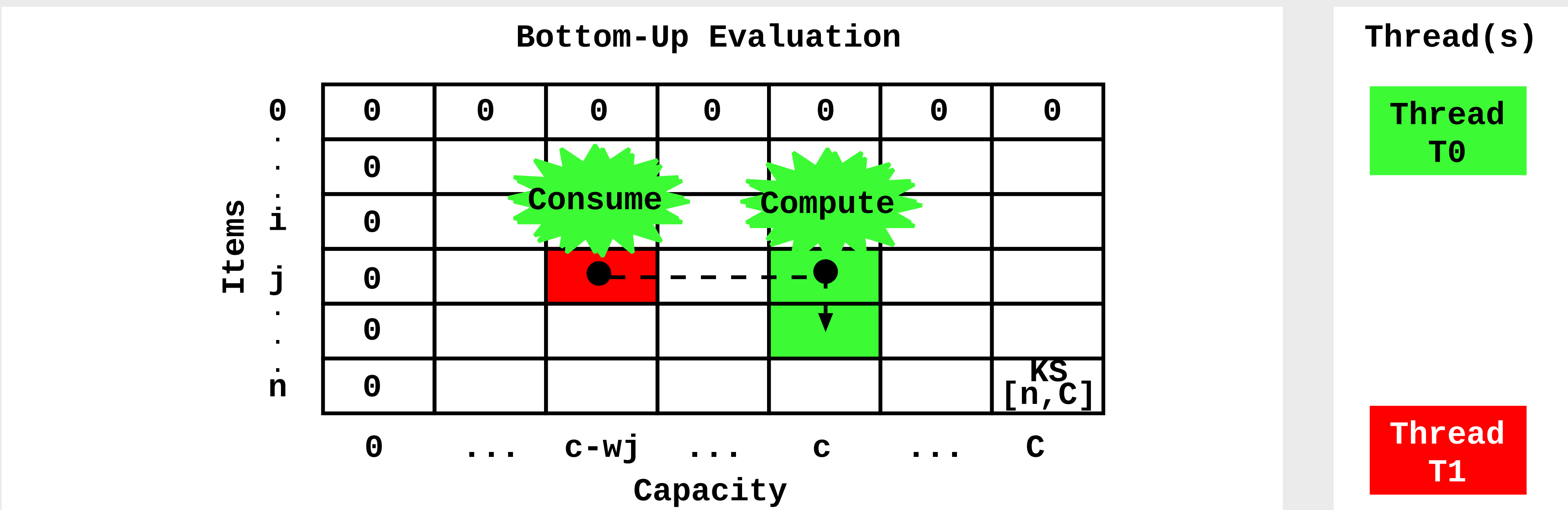
0-1 Knapsack Problem (Bottom-Up)

- 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**.



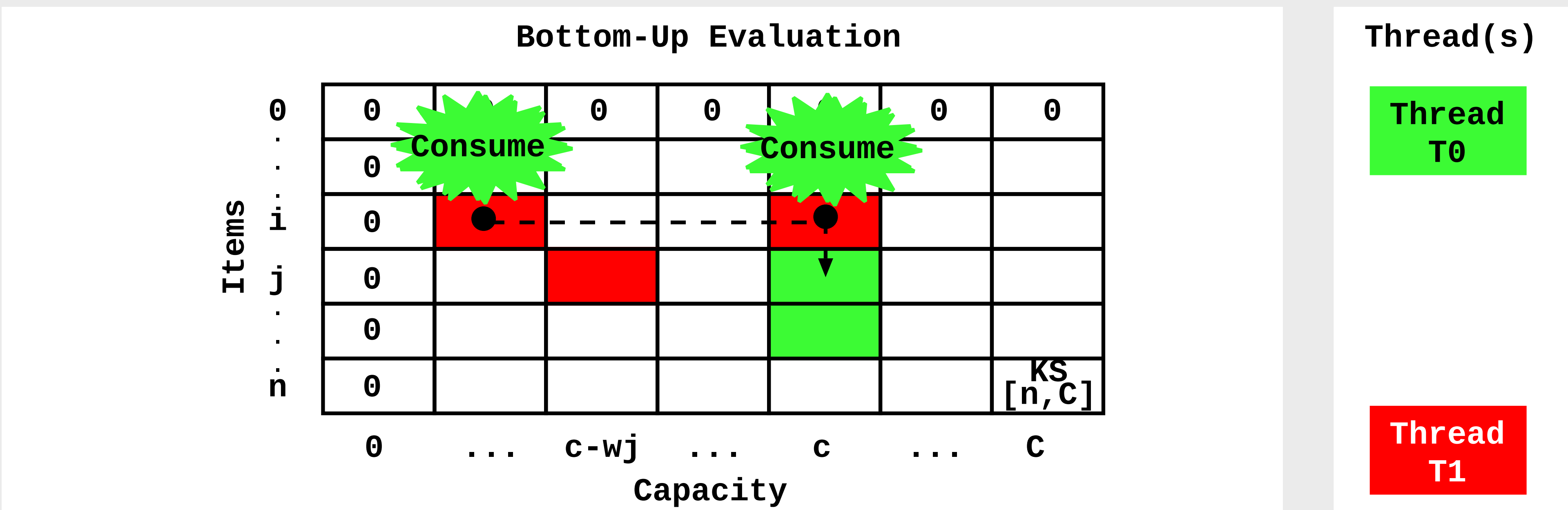
0-1 Knapsack Problem (Bottom-Up)

- 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		Seq. Time (T_{seq})	# Threads (p) Time (T_1) 1	Speedup (T_1/T_p) 8	16	24	32	Best Time (T_{best})
Top-Down Approaches								
YAP No Random	D_{10}	9,508	12,415	n.c.	n.c.	n.c.	n.c.	9,508
	D_{30}	9,246	12,177	n.c.	n.c.	n.c.	n.c.	9,246
	D_{50}	9,480	12,589	n.c.	n.c.	n.c.	n.c.	9,480
YAP Random	D_{10}	14,330	19,316	1.96	2.12	2.04	1.95	9,115
	D_{30}	14,725	19,332	3.57	4.17	4.06	3.93	4,639
	D_{50}	14,729	18,857	4.74	6.28	6.44	6.41	2,930
YAP Random+Offset	D_{10}	19,667	24,444	6.78	12.35	15.44	18.19	1,344
	D_{30}	19,847	25,609	7.15	13.83	17.37	20.47	1,251
	D_{50}	19,985	25,429	7.27	13.70	17.35	20.62	1,233
Bottom-Up Approaches								
YAP	D_{10}	12,614	17,940	7.17	13.97	18.31	22.15	0,810
	D_{30}	12,364	17,856	7.23	13.78	18.26	21.94	0,814
	D_{50}	12,653	17,499	7.25	14.01	18.34	21.76	0,804
XSB	D_{10}	32,297	38,965	0.87	0.66	0.62	0.55	32,297
	D_{30}	32,063	38,007	0.86	0.61	0.56	0.53	32,063
	D_{50}	31,893	38,534	0.84	0.58	0.57	0.57	31,893

U.PORTO FC FACULDADE DE CIÊNCIAS UNIVERSIDADE DO PORTO

Conclusions

- We have presented **novel approaches** for **concurrent**:
  - **table spaces**: **No-Sharing**, **Subgoal-Sharing** and **Full-Sharing**.
  - **memory allocation**, using **Global** and **Local Heaps of Pages** per type of data structure.
  - **tries**: **Lock-Free Tries** and **Lock-Free Hash Tries**.

Conclusions

- We have presented **novel approaches** for **concurrent**:
  - **table spaces**: **No-Sharing**, **Subgoal-Sharing** and **Full-Sharing**.
  - **memory allocation**, using **Global** and **Local Heaps of Pages** per type of data structure.
  - **tries**: **Lock-Free Tries** and **Lock-Free Hash Tries**.

- Experimental results **showed** that we able to **effectively reduce overheads** when our multithreaded tabling system is **exposed to worst case scenarios**.

Conclusions

- We have presented **novel approaches** for **concurrent**:
  - **table spaces**: **No-Sharing**, **Subgoal-Sharing** and **Full-Sharing**.
  - **memory allocation**, using **Global** and **Local Heaps of Pages** per type of data structure.
  - **tries**: **Lock-Free Tries** and **Lock-Free Hash Tries**.

- Experimental results **showed** that we able to **effectively reduce overheads** when our multithreaded tabling system is **exposed to worst case scenarios**.

- Shown the potentially of **Subgoal-Sharing** design with **Answer-Sharing**, by scaling the **0-1 Knapsack** and the **Longest Common Subsequence** problems, which are two well-known **dynamic programming problems**.
  - **Top-Down** vs **Bottom-Up**.

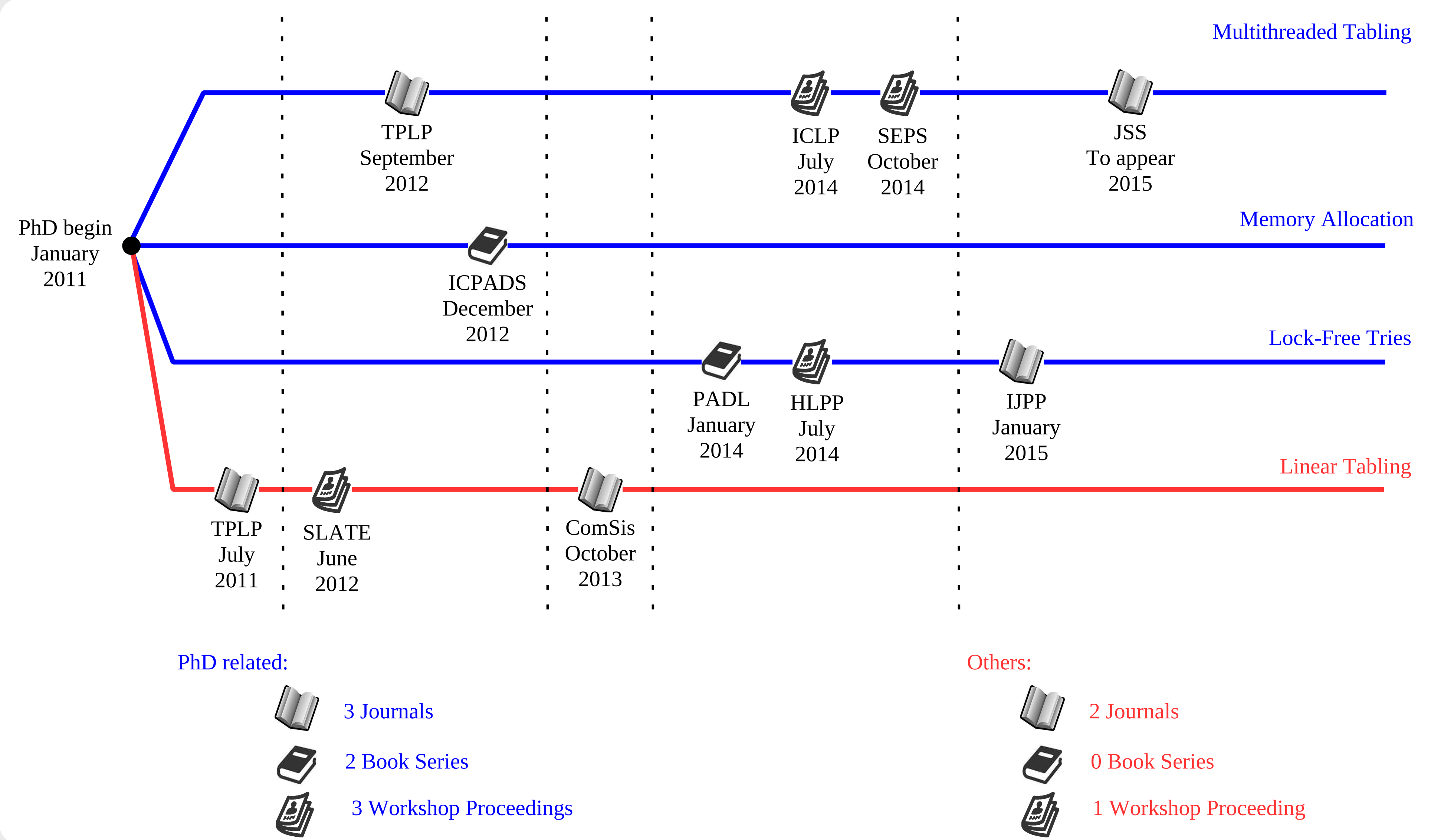
Further Work

- **Further work** will include:
  - **Integrate this work** in the main repository of **Yap** (currently in https://github.com/miar/yap-6.3)
  - Extend the **Full-Sharing** design to support mode-directed tabling.

Further Work

- **Further work** will include:
  - **Integrate this work** in the main repository of **Yap** (currently in https://github.com/miar/yap-6.3)
  - Extend the **Full-Sharing** design to support mode-directed tabling.
  - Support a concurrent multithreaded tabling model similar to the **XSB**'s shared tables (without the usurpation procedure).
  - Extend the **concurrent lock-free trie** proposals to support the **concurrent delete** operation.

Publications during PhD



Thank You !!!