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).
```

```
1.a(X)
```

SLD Resolution: Infinite Cycles

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

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

```
1.a(X)
   |  c1
2.b(X)
```

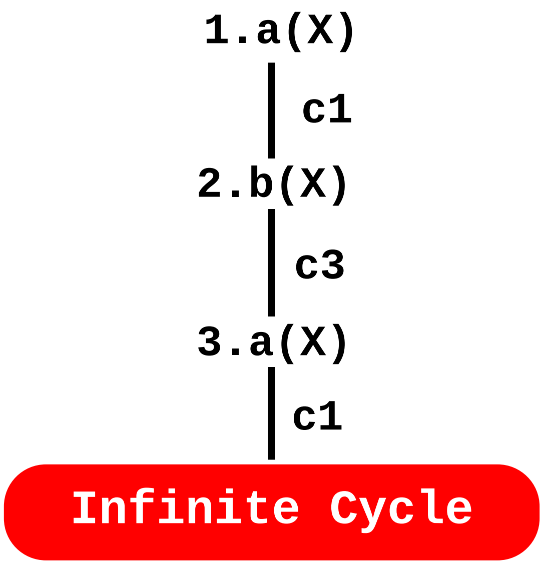
SLD Resolution: Infinite Cycles

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

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

```
1.a(X)
   |  c1
2.b(X)
   |  c3
3.a(X)
```

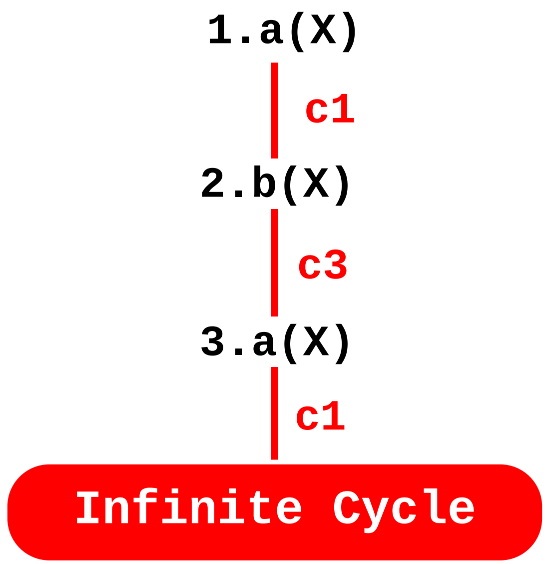
SLD Resolution: Infinite Cycles

```
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

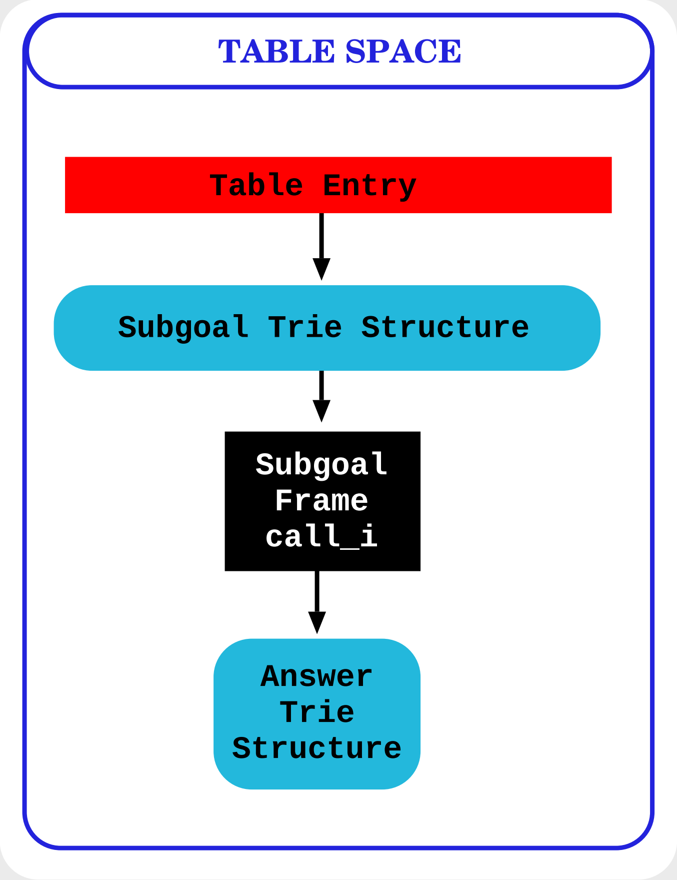
- An **Abstraction layer** with **high-level constructors** that provide access to the **dynamic programming (tabling)** support:
  - Instruction: **:- table predicate/arity.**
  - Scheduling: **:- tabling_mode(predicate, batched).**

- **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**.

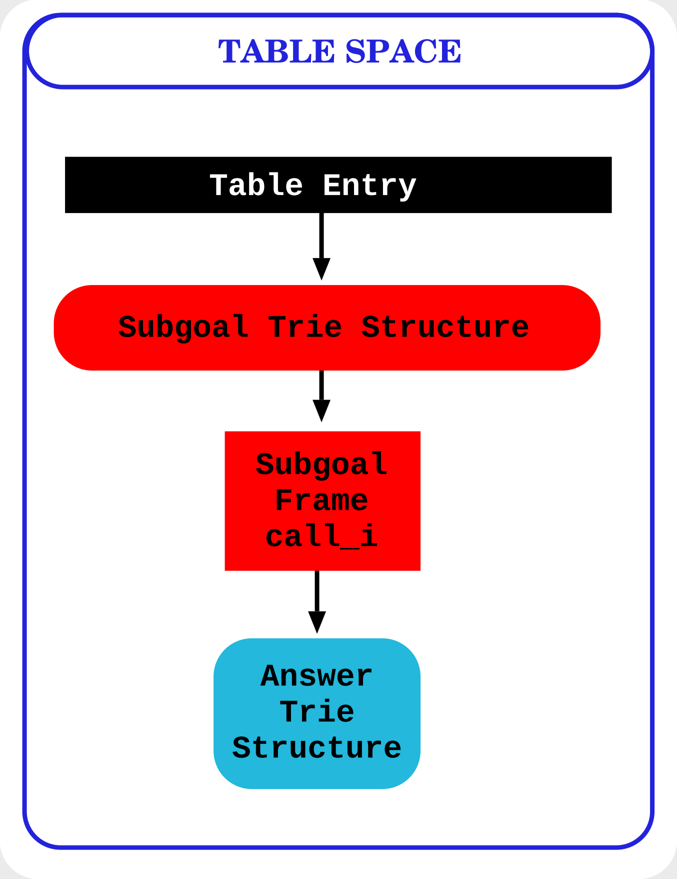
Internal Table Space Architecture

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



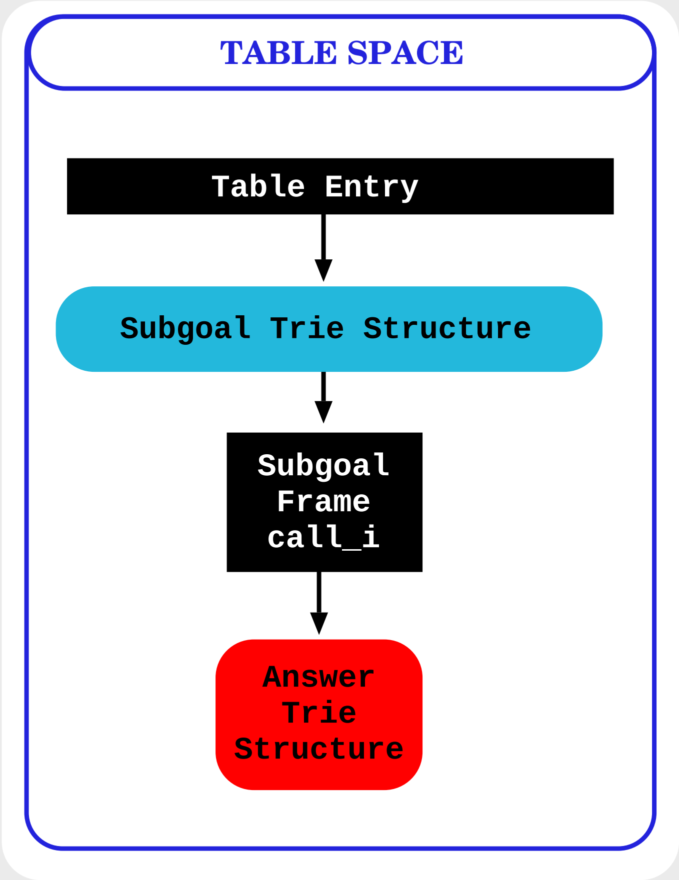
Internal Table Space Architecture

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



Internal Table Space 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).



Tabling Scheduling Strategies

- The two most successful tabling scheduling strategies are **Local** e **Batched**. They define the behavior of the tabling mechanism whenever it finds a **new answer** for a tabled call.

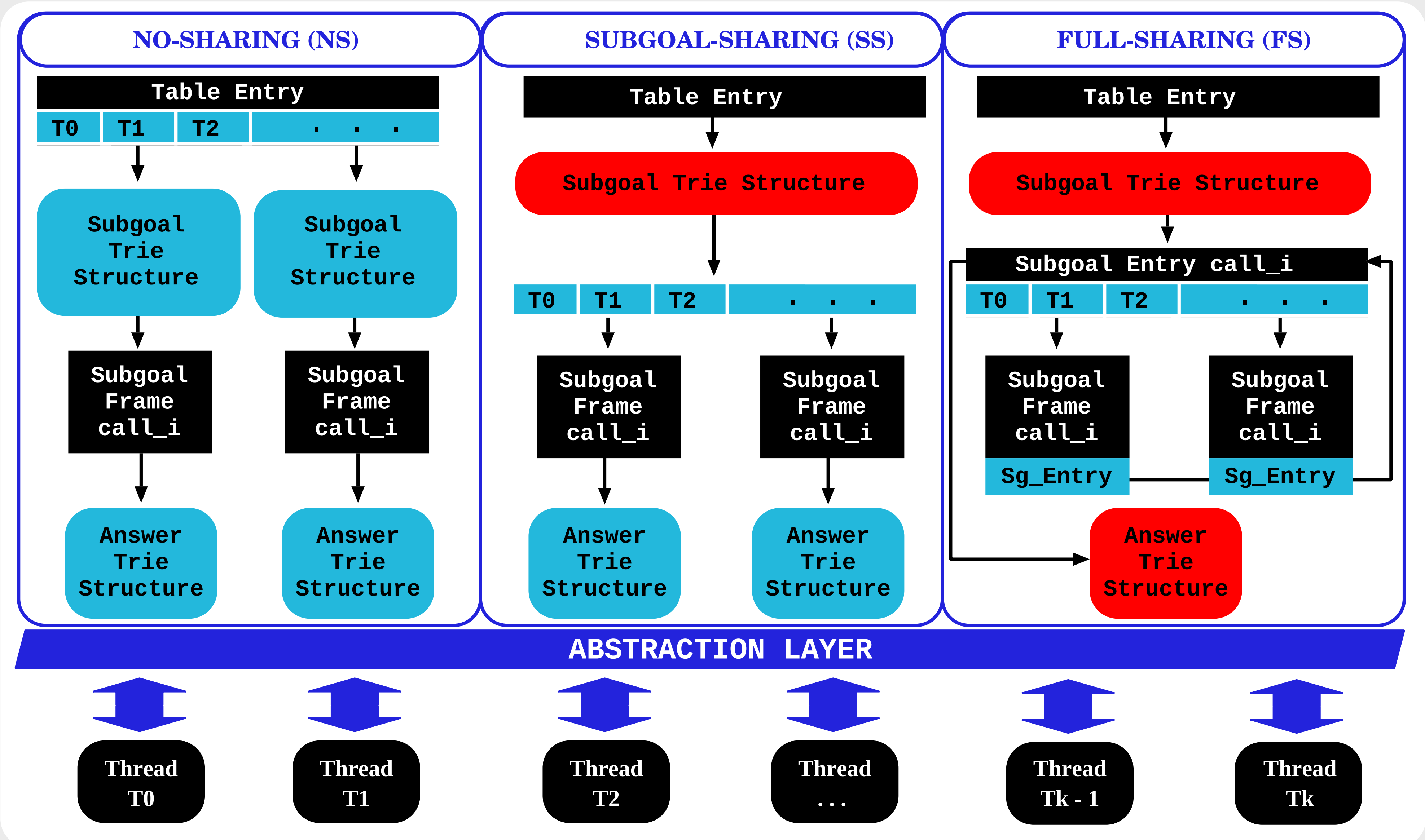
FAZER FIGURA COM LOCAL VS BATCHED. CLUSTER THE DEPENDENCIAS EM QUE EM LOCAL AS RESPOSTAS APENAS SAO CONSUMIDAS SOMENTE APOS TODAS AS RESPOSTAS SEREM ENCONTRADAS. EM BATCHED É O CONTRARIO

Tabling Scheduling Strategies

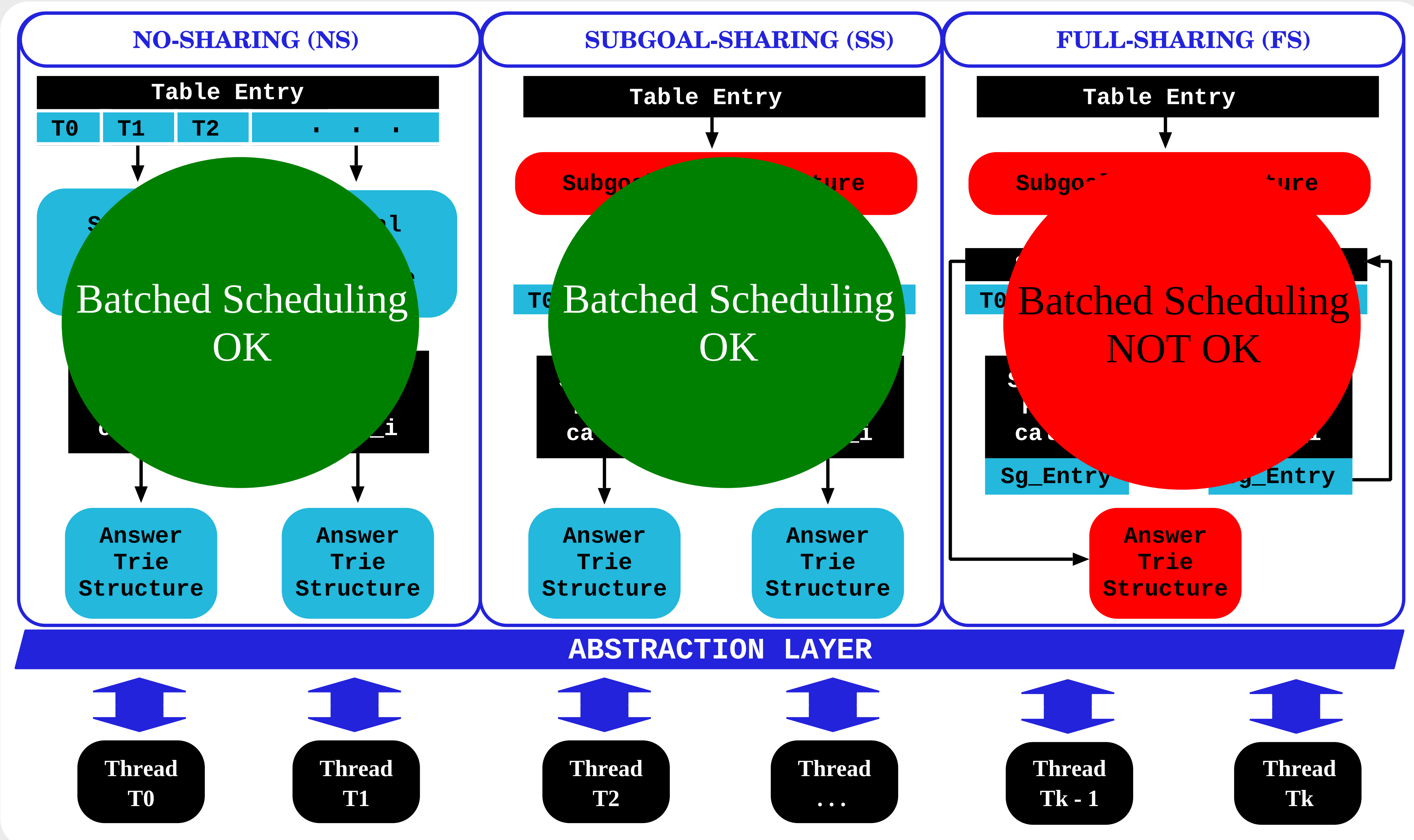
- The two most successful tabling scheduling strategies are **Local** e **Batched**. They define the behavior of the tabling mechanism whenever it finds a **new answer** for a tabled call.

Scheduling	New Answer	Fix-Point Detection
Local	**Fail** the computation to the **current** choice point.	**Consume all the answers**, propagating them to the context of the **previous** call.
Batched	**Consume the answer**, propagating it immediately to the context of the **previous** call.	**Fail** the computation to the **previous** choice point, since the actual was already fully evaluated.

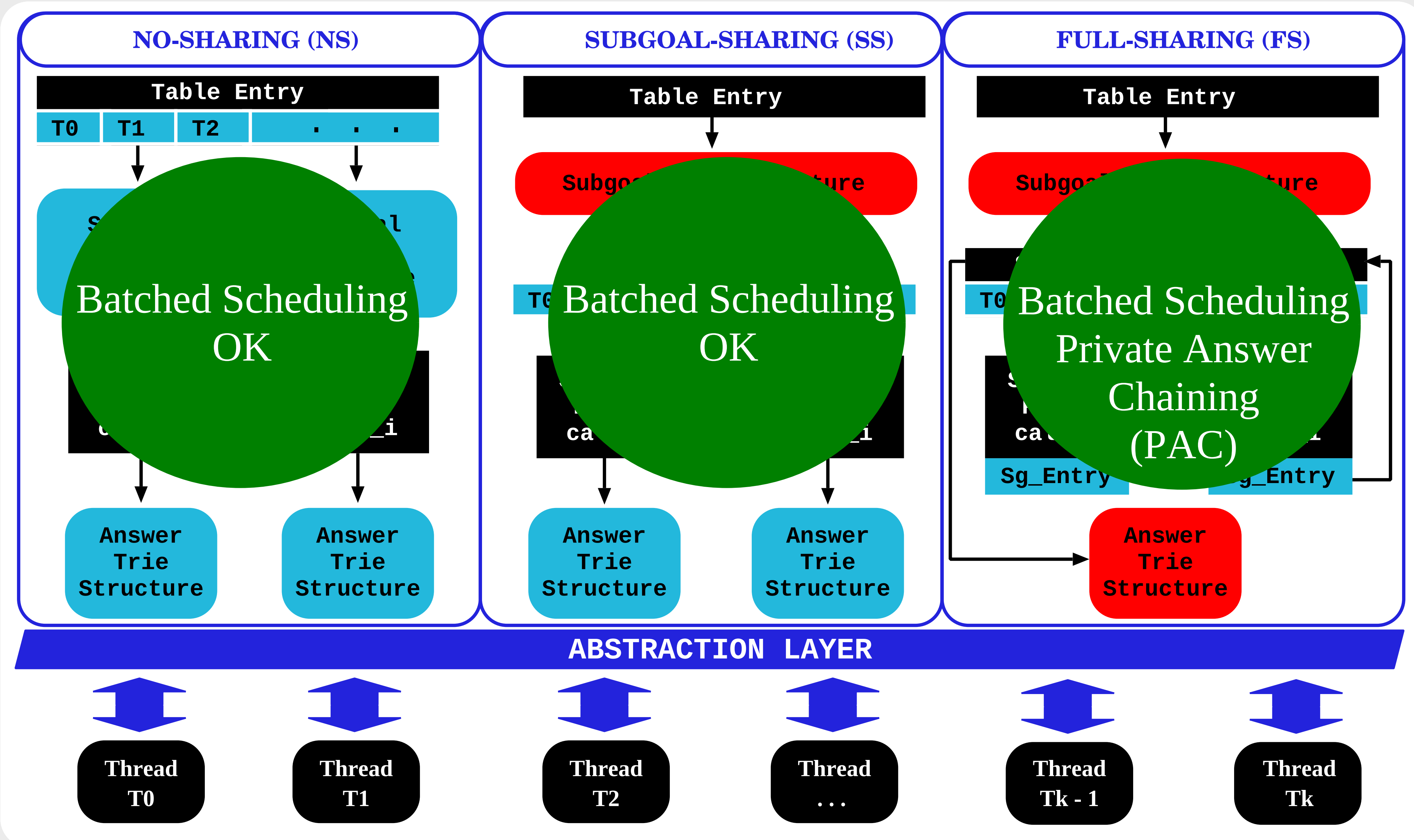
YapTab-Mt - Internal Architecture



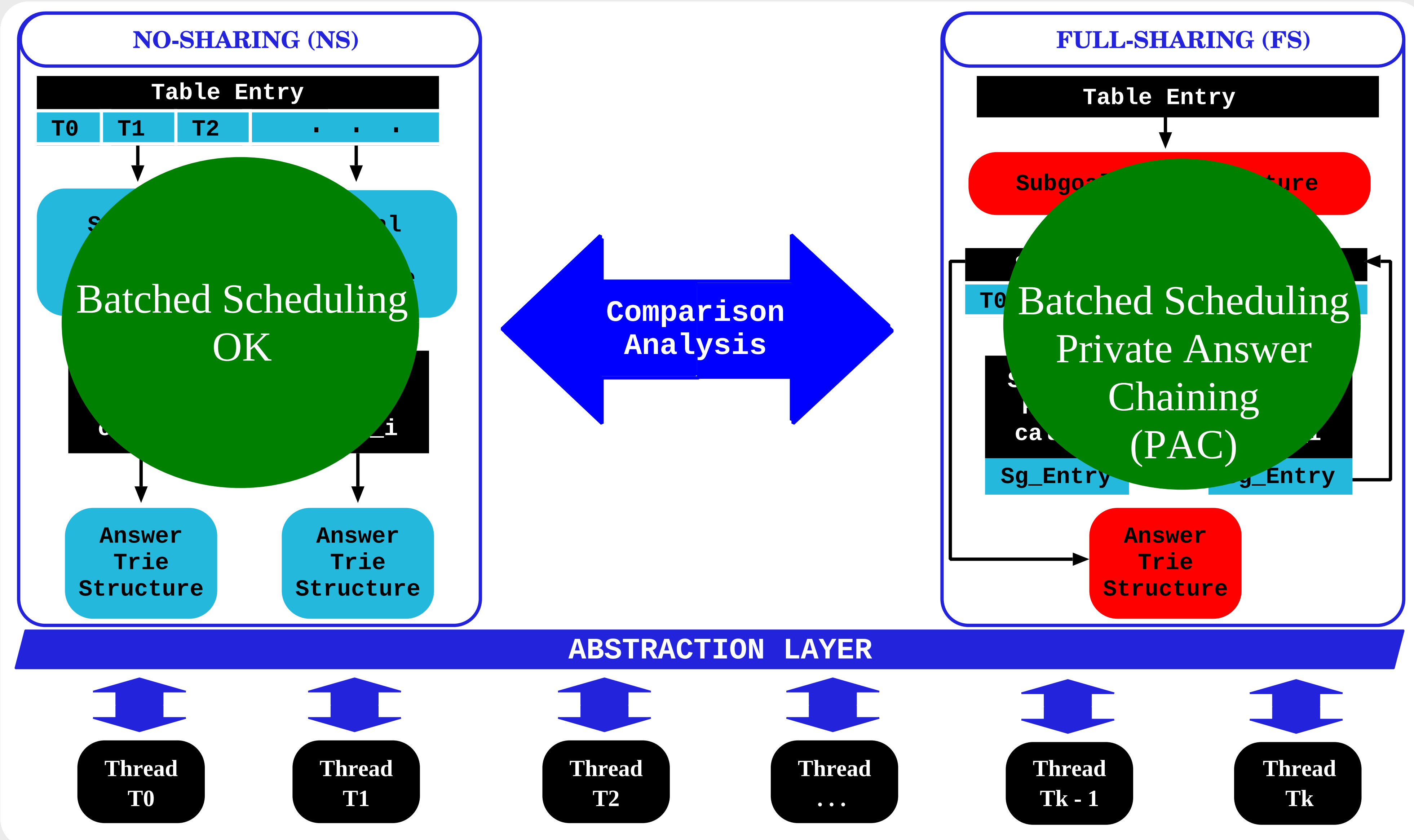
YapTab-Mt - Internal Architecture



YapTab-Mt - Internal Architecture



YapTab-Mt - Internal Architecture

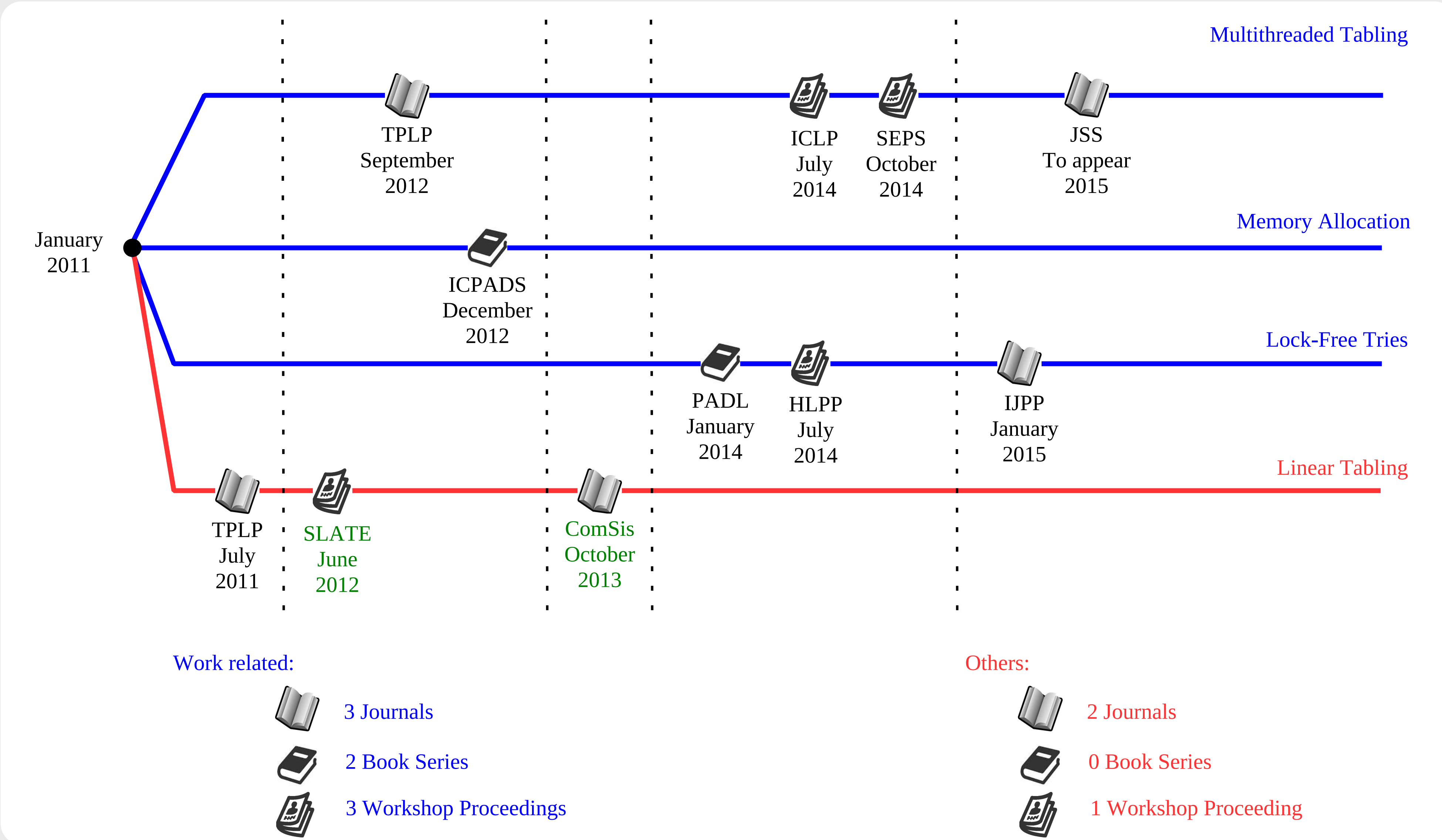


Private Answer Chaining - Key Ideas

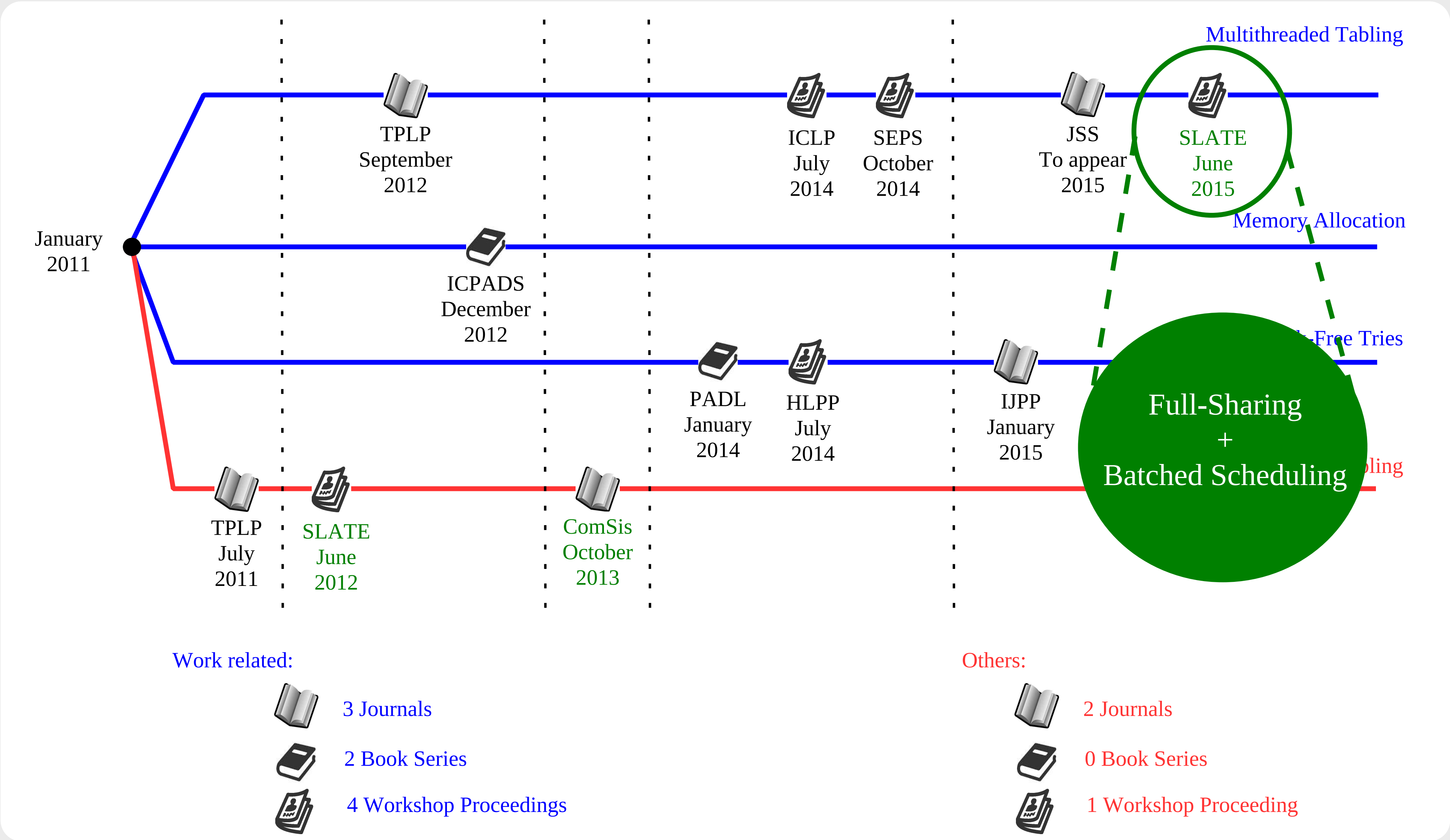
Experimental Results - Worst Case Scenarios

Threads		NS Local	NS Batched	FS Local	FS 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

Research Outline



Research Outline



Thank You !!!