Tabling and Or-Parallelism in Yap Prolog: Past, Present and Future

Ricardo Rocha
CRACS & INESC TEC
University of Porto, Portugal
ricroc@dcc.fc.up.pt

Joint work with Fernando Silva, Flávio Cruz, Inês Dutra, João Raimundo, João Santos, Miguel Areias and Vítor Santos Costa

WLPE 2011, Lexington, Kentucky, USA, July 2011

Tabling in Yap: Past

- **Past (2000-2008)**
  - **Tabling Engine [TAPD'00]**
    Support for sequential tabling based on the SLG-WAM model.

Tabling in Yap: Past

- **Past (2000-2008)**
  - **Tabling Engine [TAPD'00]**
    Support for sequential tabling based on the SLG-WAM model.
  - **Tabling and Implicit Or-Parallelism [ICLP'01,TPLP'05]**
    Support for implicit or-parallelism in tabled logic programs based on the SLG-WAM and environment copying models.

Tabling in Yap: Past

- **Past (2000-2008)**
  - **Tabling Engine [TAPD'00]**
    Support for sequential tabling based on the SLG-WAM model.
  - **Tabling and Implicit Or-Parallelism [ICLP'01,TPLP'05]**
    Support for implicit or-parallelism in tabled logic programs based on the SLG-WAM and environment copying models.
  - **Dynamic Mixed-Strategy Evaluation [ICLP'05]**
    Support for the dynamic intermixing of the two most successful tabling evaluation strategies, batched and local evaluation, at the subgoal level.

Tabling in Yap: Past

- **Past (2000-2008)**
  - **Tabling Engine [TAPD'00]**
    Support for sequential tabling based on the SLG-WAM model.
  - **Tabling and Implicit Or-Parallelism [ICLP'01,TPLP'05]**
    Support for implicit or-parallelism in tabled logic programs based on the SLG-WAM and environment copying models.
  - **Dynamic Mixed-Strategy Evaluation [ICLP'05]**
    Support for the dynamic intermixing of the two most successful tabling evaluation strategies, batched and local evaluation, at the subgoal level.
  - **Handling Incomplete and Complete Tables [ICLP'06,PADL'07]**
    New techniques for making tabling more efficient when dealing with incomplete tables and more robust when recovering memory from the tables.

Tabling in Yap: Past

➤ **Past (2000-2008)**

- **Tabling Engine [TAPD'00]**
  Support for sequential tabling based on the SLG-WAM model.
- **Tabling and Implicit Or-Parallelism [ICLP'01,TPLP'05]**
  Support for implicit or-parallelism in tabled logic programs based on the SLG-WAM and environment copying models.
- **Dynamic Mixed-Strategy Evaluation [ICLP'05]**
  Support for the dynamic intermixing of the two most successful tabling evaluation strategies, batched and local evaluation, at the subgoal level.
- **Handling Incomplete and Complete Tables [ICLP'06,PADL'07]**
  New techniques for making tabling more efficient when dealing with incomplete tables and more robust when recovering memory from the tables.
- **Program Transformation with Tabling Primitives [ICLP'07,PADL'08]**
  Support for tabling by applying source level transformations to a tabled program and by using specific external tabling primitives, implemented with the C language interface of Yap, to provide direct control over the search strategy. This work was the basis for tabling support in Ciao Prolog.

Tabling in Yap: Present and Future

- **Present (2009-2011): already available on Yap's repository**
  - Global Trie **[PADL'09,ICLP'09,EPIA'11]**
  - Compact Lists **[PADL'10]**
- **Present (2009-2011): to be synchronized soon, hopefully ;)**
  - Linear Tabling **[PADL'10,ICLP'11]**
  - Call Subsumption **[JELIA'10,ICLP'11,EPIA'11]**
  - Tabling Modes and Answer Subsumption

Tabling in Yap: Present and Future

- **Present (2009-2011): already available on Yap's repository**
  - Global Trie **[PADL'09,ICLP'09,EPIA'11]**
  - Compact Lists **[PADL'10]**
- **Present (2009-2011): to be synchronized soon, hopefully ;)**
  - Linear Tabling **[PADL'10,ICLP'11]**
  - Call Subsumption **[JELIA'10,ICLP'11,EPIA'11]**
  - Tabling Modes and Answer Subsumption
- **Future (2011-...)**
  - Multi-Threaded Tabling
  - Call Subsumption for Linear Tabling
  - Incremental Tabling
  - Negation
  - Co-Induction

Or-Parallelism in Yap: Past, Present and Future

- **Past (1999-2003)**
  - **Or-Parallelism for Shared Memory [EPIA'99,EUROPAR'00]**
    Support for implicit or-parallelism based on the environment copying model.
  - **Or-Parallelism For Distributed Memory [EPIA'03]**
    Support for implicit or-parallelism based on the stack splitting model.

Or-Parallelism in Yap: Past, Present and Future

➤ Past (1999-2003)

- ♦ **Or-Parallelism for Shared Memory [EPIA'99,EUROPAR'00]**
  Support for implicit or-parallelism based on the environment copying model.
- ♦ **Or-Parallelism For Distributed Memory [EPIA'03]**
  Support for implicit or-parallelism based on the stack splitting model.

➤ Present (2010-2011)

- ♦ **Or-Parallelism using Threads [ICLP'10]**
  Redesign of the or-parallel model based on the environment copying model to exploit or-parallelism based on a multi-threaded implementation.

Or-Parallelism in Yap: Past, Present and Future

- **Past (1999-2003)**
  - **Or-Parallelism for Shared Memory [EPIA'99,EUROPAR'00]**
    Support for implicit or-parallelism based on the environment copying model.
  - **Or-Parallelism For Distributed Memory [EPIA'03]**
    Support for implicit or-parallelism based on the stack splitting model.
- **Present (2010-2011)**
  - **Or-Parallelism using Threads [ICLP'10]**
    Redesign of the or-parallel model based on the environment copying model to exploit or-parallelism based on a multi-threaded implementation.
- **Future (2011-...)**
  - **Explicit Parallel Constructs**
    Use of explicit high-level parallel constructs to trigger parallel execution.
  - **Teams of Workers for Shared/Distributed Memory**
    Design of a new parallel platform that combines environment copying with stack splitting to scale-up on clusters of multi-core processors.

In This Talk

- **Tabling**
  - Global Trie
  - Call Subsumption
  - Tabling Modes and Answer Subsumption
  - Multi-Threaded Tabling
- **Or-Parallelism**
  - Explicit Parallel Constructs
  - Teams of Workers for Shared/Distributed Memory

Tabling in Logic Programming

- Tabling is an implementation technique that overcomes some limitations of traditional Prolog systems in dealing with **recursion** and **redundant sub-computations**.

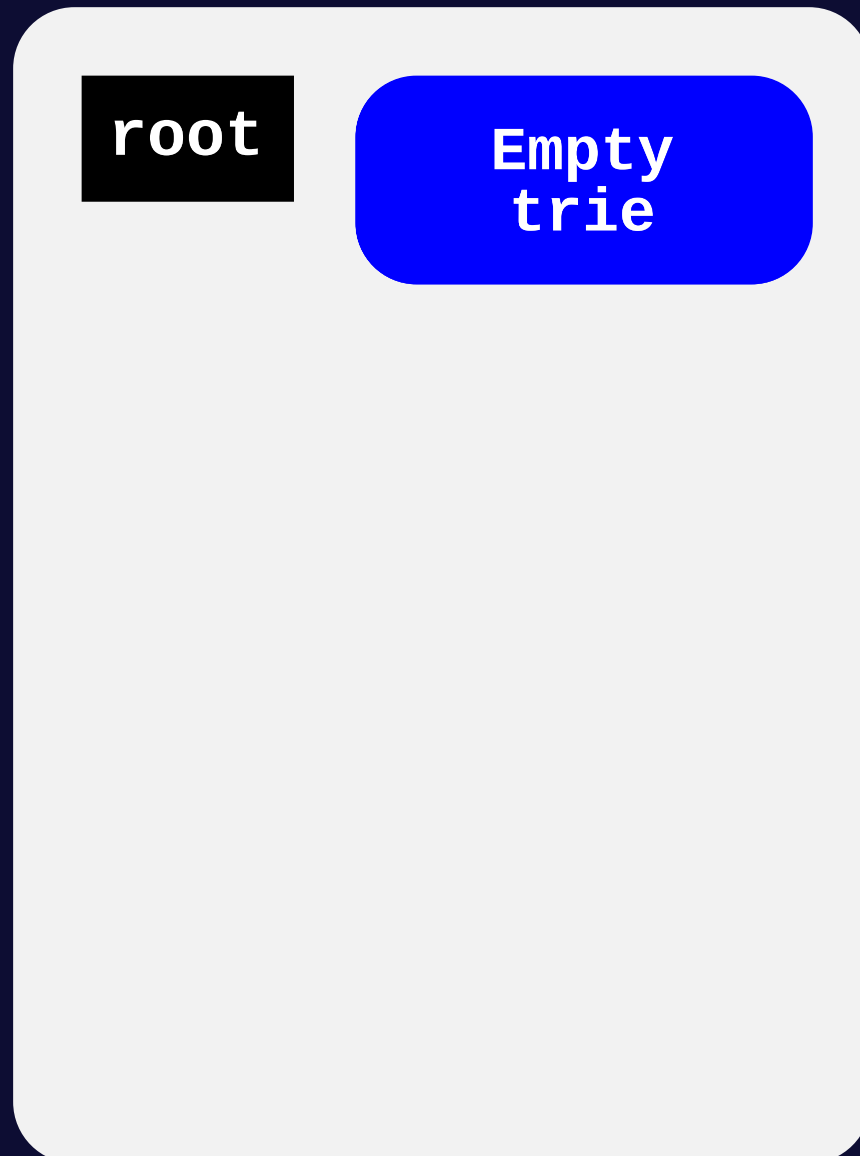
- It extends the standard SLD resolution method by adding new tabling operations.
  - First calls to tabled subgoals are evaluated as usual through the execution of Prolog code but answers are inserted into a **table space**.
  - **Similar calls** are evaluated by **consuming answers from the table space** that were generated by the corresponding similar subgoal, instead of re-evaluating them against the program clauses.
  - As new answers are found, they are inserted into the table space and **returned to all similar calls**.

Table Space

- **Can be accessed to:**
  - Look up if a subgoal is in the table and, if not, insert it.
  - Look up if a newly found answer is in the table and, if not, insert it.
  - Load answers for similar subgoals.
- **Implementation requirements:**
  - Fast look-up and insertion methods.
  - Compactness in representation of logic terms.

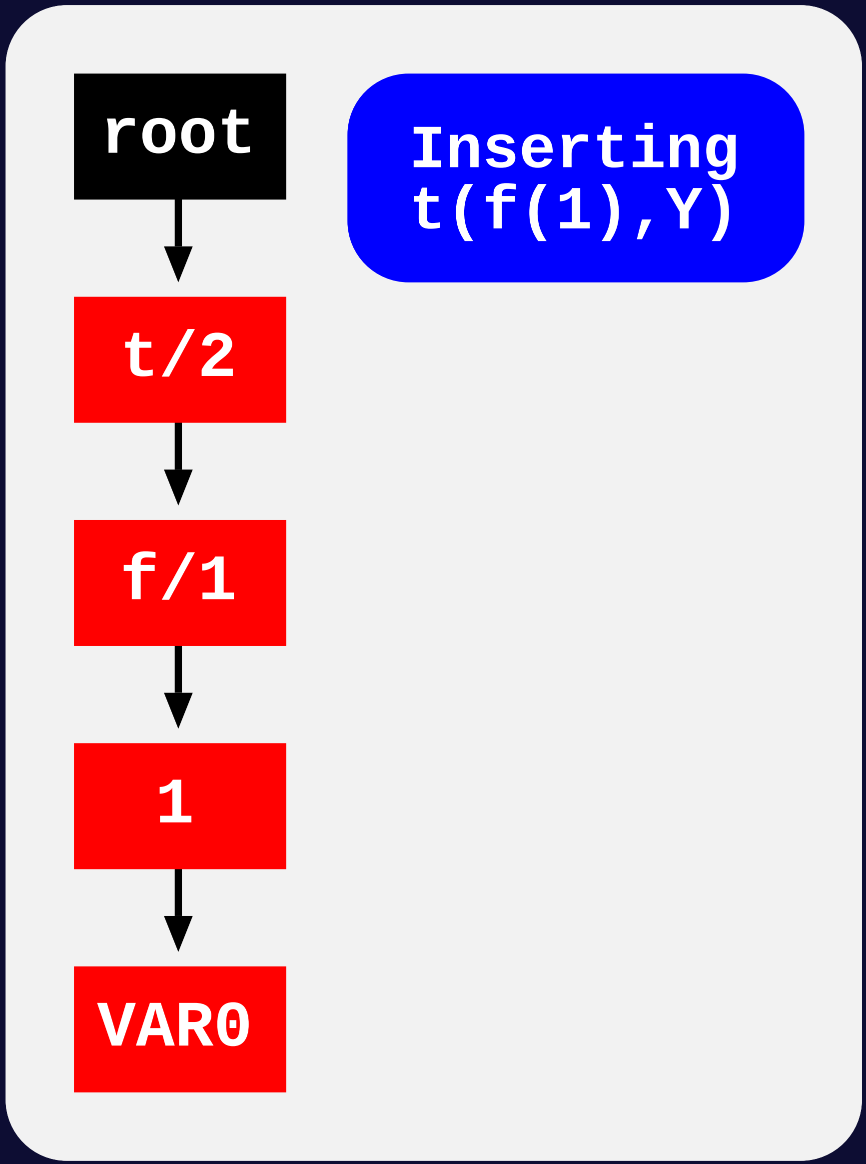
Using Tries to Represent Terms

- Tries are trees in which common prefixes are represented only once.
- Each different path through the nodes in the trie corresponds to a term.
- Terms with common prefixes branch off from each other at the first distinguishing token.



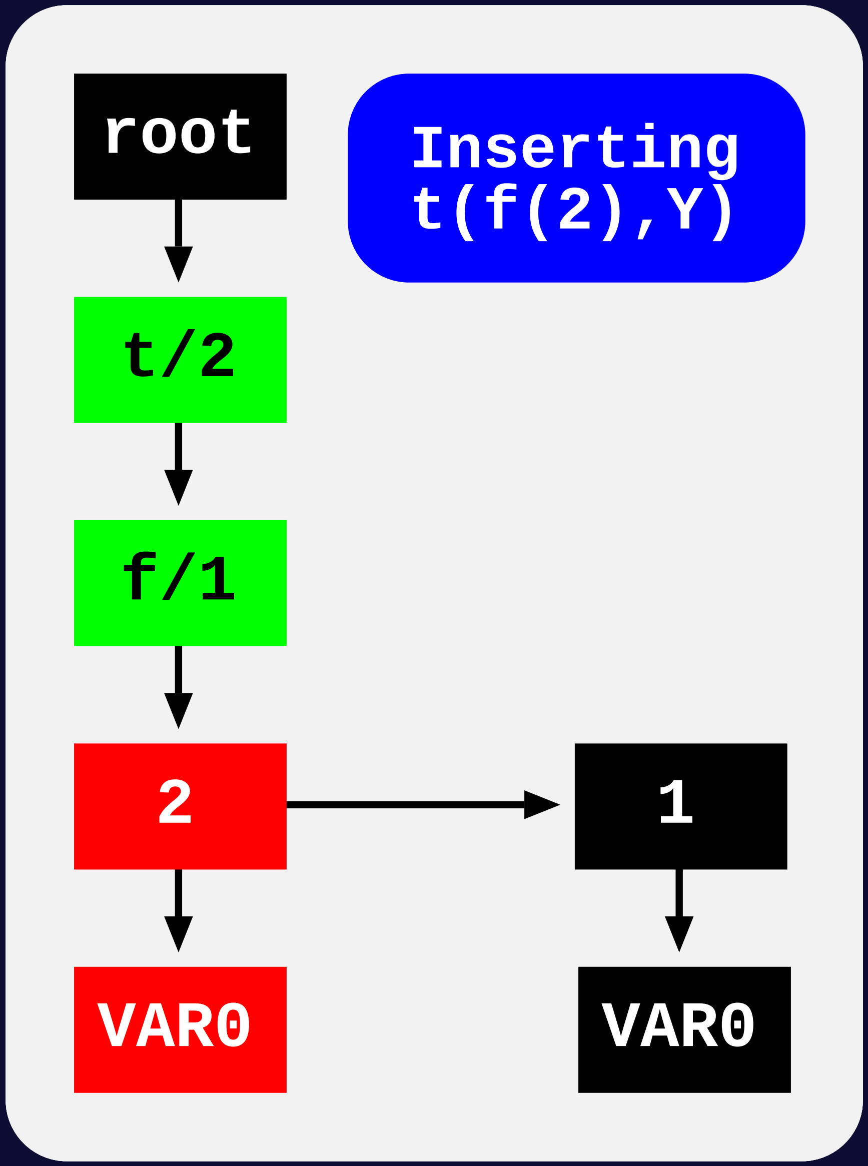
Using Tries to Represent Terms

- Tries are trees in which common prefixes are represented only once.
- Each different path through the nodes in the trie corresponds to a term.
- Terms with common prefixes branch off from each other at the first distinguishing token.

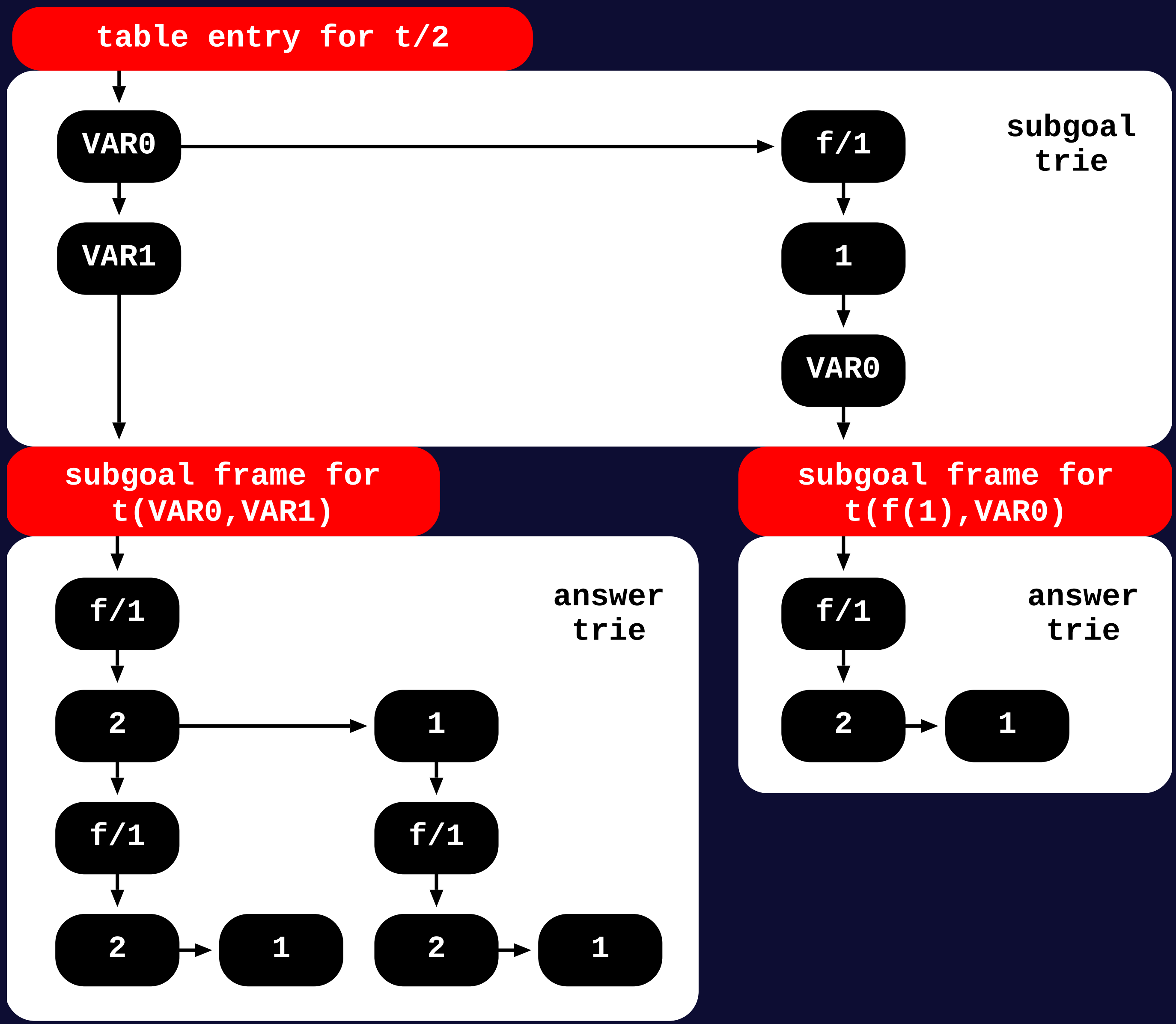


Using Tries to Represent Terms

- Tries are trees in which common prefixes are represented only once.
- Each different path through the nodes in the trie corresponds to a term.
- Terms with common prefixes branch off from each other at the first distinguishing token.



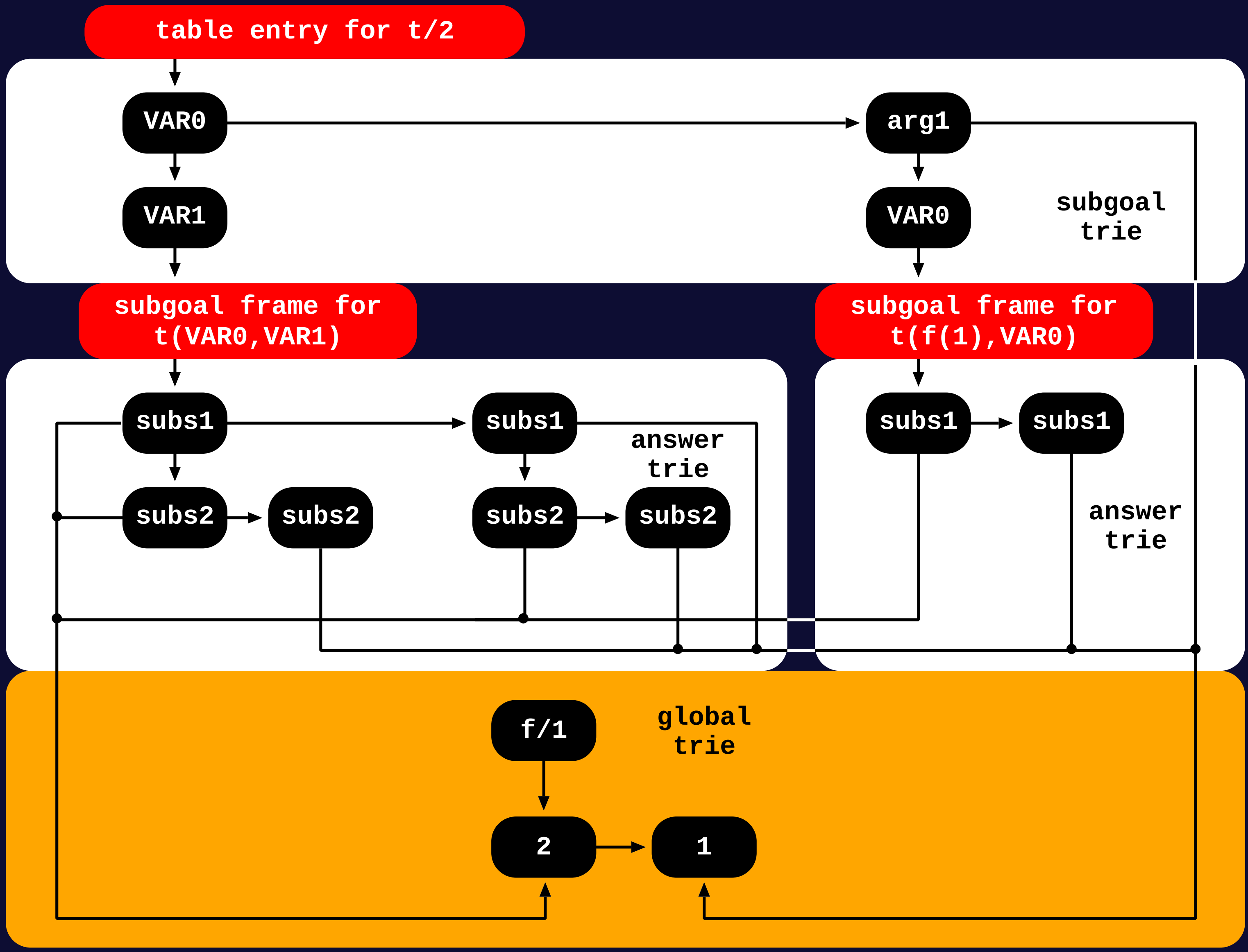
Using Tries to Represent the Table Space



GT-T: Global Trie for Terms

- In GT-T, all **argument and substitution compound terms** appearing in tabled subgoal calls and/or answers are **represented only once in the GT**, thus preventing situations where these terms are represented more than once in different trie data structures.
- Each path in the original subgoal and answer tries is composed of a **fixed number** of trie nodes representing the number of argument or substitution terms in the corresponding tabled subgoal call or answer.

GT-T: Global Trie for Terms



GT-ST: Global Trie for Subterms

- The GT-ST maximizes the sharing of the tabled data that is **structurally equal at a second level**, by avoiding the representation of equal compound subterms, and thus preventing situations where the representation of those subterms occur more than once.
- Although GT-ST uses the same GT-T's tree structure for implementing the GT, every different path in the GT can now **represent a complete term or a subterm of another term, but still being an unique term**.

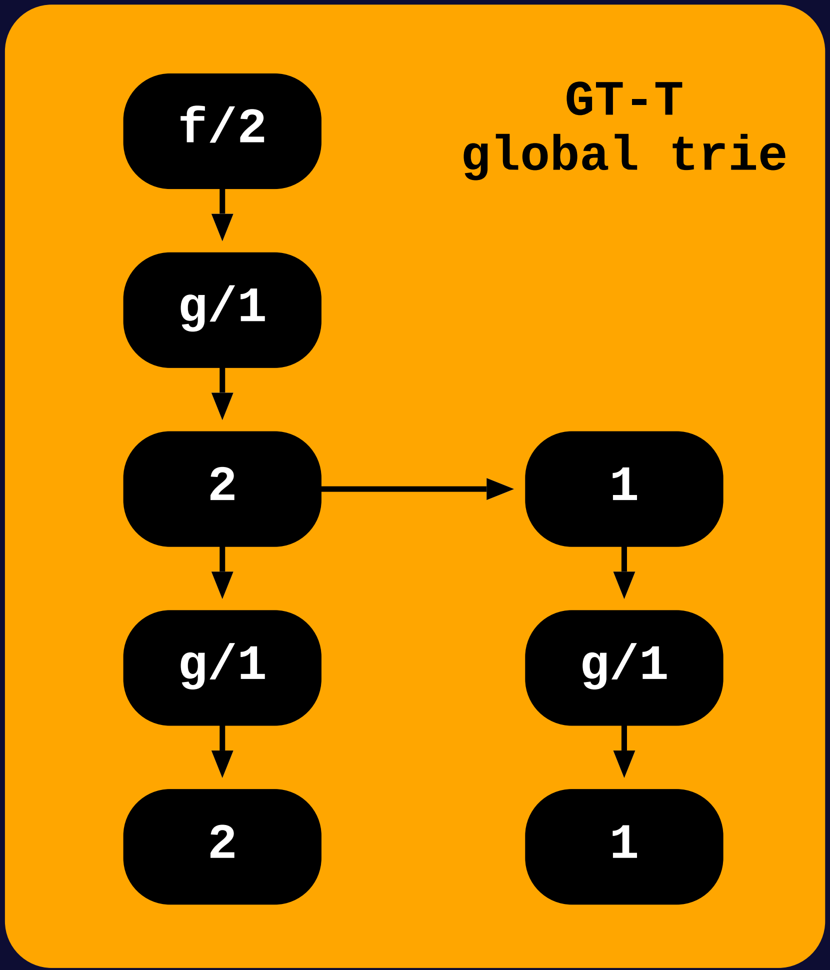
GT-ST: Global Trie for Subterms

- Consider, for example, the insertion of the terms **f(g(1),g(1))** and **f(g(2),g(2))** in the GT-T...

```
:- table t/2.

t(X,Y) :- term(X),
          term(Y).

term(f(g(1),g(1))).
term(f(g(2),g(2))).
```



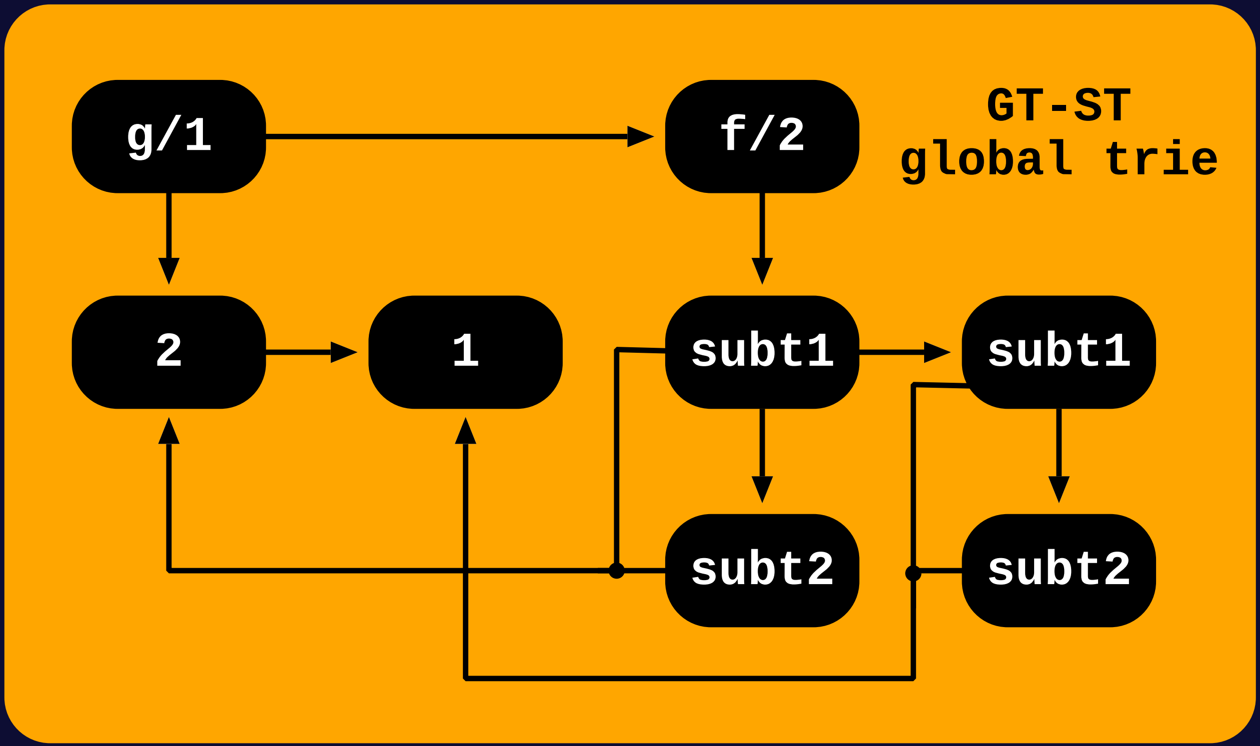
GT-ST: Global Trie for Subterms

- ... and in the GT-ST.

```
:- table t/2.

t(X,Y) :- term(X),
          term(Y).

term(f(g(1),g(1))).
term(f(g(2),g(2))).
```



Global Trie: Experimental Results

Terms	GT-T/YapTab				GT-ST/YapTab			
	Mem	Store	Load	Comp	Mem	Store	Load	Comp
1,000 ints	1.00	1.05	1.00	1.00	1.00	1.09	1.11	1.07
1,000 atoms	1.00	1.04	1.01	1.02	1.00	1.04	1.03	1.08
1,000 f/1	1.00	1.32	1.16	2.10	1.00	1.34	1.17	2.13
1,000 f/2	**0.50**	1.10	1.14	1.84	**0.50**	1.06	1.11	1.88
1,000 f/4	**0.25**	**0.81**	**0.98**	1.44	**0.25**	**0.78**	1.04	1.53
1,000 f/6	**0.17**	**0.72**	**0.72**	1.38	**0.17**	**0.66**	**0.71**	1.36
1,000 [ ]/1	**0.50**	1.08	1.05	1.61	**0.50**	1.10	1.02	1.58
1,000 [ ]/2	**0.25**	**0.80**	**0.94**	1.38	**0.25**	1.00	1.05	1.48
1,000 [ ]/4	**0.13**	**0.63**	**0.54**	**0.96**	**0.13**	**0.89**	**0.66**	1.14
Average	**0.53**	**0.95**	**0.95**	**1.42**	**0.53**	**0.99**	**0.99**	**1.47**

Memory usage and store/load times for a t/5 tabled predicate that simply stores in the table space terms defined by `term/1` facts, called with all combinations of one and two free variables in the arguments.

Global Trie: Experimental Results

Terms	GT-T Mem (MB) Total/GT	GT-T Times (ms) Str/Ld/Cmp	GT-ST/GT-T Mem Total/GT	GT-ST/GT-T Times Str/Ld/Cmp
f/1				
500,000 g/1	17.17/7.63	126/28/51	1.44 / 2.00	1.55 / 1.14 / 1.00
500,000 g/3	32.43/22.89	198/34/61	1.24 / 1.33	3.29 / 1.12 / 1.25
500,000 g/5	47.68/38.15	293/47/83	1.16 / 1.20	1.46 / 1.00 / **0.99**
f/2				
500,000 g/1	32.43/22.89	203/38/71	1.00 / 1.00	1.28 / 1.13 / 1.09
500,000 g/3	62.94/53.41	45/60/103	**0.76** / **0.71**	1.18 / **0.84** / **0.95**
500,000 g/5	93.46/83.92	438/111/146	**0.67** / **0.64**	1.10 / **0.67** / **0.80**
f/3				
500,000 g/1	47.68/38.15	296/50/89	**0.84** / **0.80**	2.87 / 1.02 / 1.03
500,000 g/3	93.46/83.92	616/142/164	**0.59** / **0.55**	1.25 / **0.80** / **0.85**
500,000 g/5	139.24/129.7	832/197/224	**0.51** / **0.47**	**0.96** / **0.67** / **0.74**
Average			**0.96** / **0.97**	**0.93** / **0.97** / **0.91**

Memory usage and store/load times for a t/1 tabled predicate that simply stores in the table space terms defined by `term/1` facts.

Call Subsumption

- In general, we can distinguish two main approaches to determine similarity between tabled subgoals.
  - **Call by Variance:** subgoal A is similar to B if they are the same by renaming the variables.
    Example: **p(X,1,Y)** and **p(Y,1,Z)** are **variants** because both can be renamed into **p(VAR0,1,VAR1)**.

Call Subsumption

- In general, we can distinguish two main approaches to determine similarity between tabled subgoals.
  - **Call by Variance:** subgoal A is similar to B if they are the same by renaming the variables.
    Example: **p(X,1,Y)** and **p(Y,1,Z)** are **variants** because both can be renamed into **p(VAR0,1,VAR1)**.
  - **Call by Subsumption:** subgoal A is similar to B if A is more specific than B (or B is more general than A).
    Example: **p(X,1,2)** is more specific than **p(Y,1,Z)** because there is a **substitution {Y=X, Z=2}** that makes **p(X,1,2)** an **instance** of **p(Y,1,Z)**.

Call Subsumption

➤ Advantages

- ♦ **Less code is executed** because subsumed subgoals can reuse answers instead of executing their own code.
- ♦ **More answers are shared** across subgoals, therefore there is less redundancy in the table space.

➤ Disadvantages

- ♦ **More strict semantics** (with some extra-logical features of Prolog, such as the `var/1` predicate, call by subsumption should not be used as it can produce wrong results).
- ♦ The mechanisms to support subsumption-based tabling are **harder to implement**.

Retroactive Call Subsumption

- We have developed a new resolution extension called **Retroactive Call Subsumption (RCS)** that supports subsumption-based tabling by allowing full sharing of answers among subsumptive subgoals, independently of the order they are called.
  Example: if **p(1,X)** is called **before or after** **p(X,Y)**, **p(1,X)** **will reuse** the answers from **p(X,Y)**. This is not the case in XSB Prolog, because if **p(1,X)** is called before **p(X,Y)**, **no reuse will occur**.

- RCS selectively prunes the evaluation of a subgoal S when a more general subgoal G appears later on.

- RCS works by pruning the execution branch of S and then by restarting the evaluation of S as a consumer. By doing that, we **save execution time** by not executing code that would generate a subset of the answers we can find by executing G.

Retroactive Call Subsumption: Challenges

- **Keep execution consistent after pruning [JELIA'10]**
  - Build a subgoal dependency tree.
  - Update the low-level stacks related to the pruned subgoals.
  - New operations and evaluation strategies that can handle multiple scenarios in order to ensure correct completion.

- **Compute the set of subsumed subgoals executing [ICLP'11]**
  New algorithms and extensions to the table space to efficiently retrieve the set of subsumed subgoals.

- **Ensure that new consumers will not consume repeated answers [EPIA'11]**
  New table space organization where answers are represented only once.

Retroactive Call Subsumption: Experimental Results

Program	Yap Prolog Variant/RCS	Yap Prolog Subsumption/RCS
left_first	0.89	0.95
left_last	0.88	0.90
double_first	1.07	**1.09**
double_last	1.05	**1.10**
genome	450.33	0.74
reach_first	2.54	**1.76**
reach_last	3.22	**1.87**
flora	3.17	**1.17**
fib	1.95	**2.02**
big	13.26	**13.66**

For programs where the time needed to retrieve the answers for the subsumed subgoal offsets the time spent executing the code, RCS performs slightly worse.

Tabling Modes and Answer Subsumption

➤ Mode Declaration

- **:- table p(M1,M2,...,Mn).**

➤ Available Modes

- **index** (index argument)
- **first** (keeps first answer)
- **last** (keeps last answer)
- **all** (keeps all answers)
- **min** (keeps minimum answer)
- **max** (keeps maximum answer)

Tabling Modes

:- table p(index,index,first).

Answers		Table Space
1. **p(1,2,10)**	New	1
2. **p(1,2,6)**	Repeats 1	1
3. **p(1,3,5)**	New	1 / 3
4. **p(1,3,6)**	Repeats 3	1 / 3
5. **p(1,2,8)**	Repeats 1	1 / 3

Tabling Modes

:- table p(index,index,first).

Answers		Table Space
1. **p(1,2,10)**	New	1
2. **p(1,2,6)**	Repeats 1	1
3. **p(1,3,5)**	New	1 / 3
4. **p(1,3,6)**	Repeats 3	1 / 3
5. **p(1,2,8)**	Repeats 1	1 / 3

:- table p(index,index,min,all).

Answers		Table Space
1. **p(1,2,10,[1,3,2])**	New + Removed by 2	1
2. **p(1,2,6,[1,4,2])**	Better Than 1	2
3. **p(1,3,5,[1,3])**	New	2 / 3
4. **p(1,3,6,[1,9,3])**	Worse Than 3	2 / 3
5. **p(1,2,6,[1,5,2])**	Equal To 2	2 / 3 / 4

Answer Subsumption

:- table p(index,index,min).

Answers		Table Space
1. **p(1,2,3)**	New + Removed by 2	1
2. **p(1,2,2)**	Better Than 1	2
3. **p(2,3,3)**	New	2 / 3
4. **p(1,Y,1)**	New	2 / 3 / 4
5. **p(1,5,3)**	New	2 / 3 / 4 / 5

Answer Subsumption

:- table p(index,index,min).

Answers		Table Space
1. **p(1,2,3)**	New + Removed by 2	1
2. **p(1,2,2)**	Better Than 1	2
3. **p(2,3,3)**	New	2 / 3
4. **p(1,Y,1)**	New	2 / 3 / 4
5. **p(1,5,3)**	New	2 / 3 / 4 / 5

answer_subsumption(p(_,_,C),min,C).

Answers		Table Space
1. **p(1,2,3)**	New + Removed by 2	1
2. **p(1,2,2)**	Better Than 1+ Removed by 4	2
3. **p(2,3,3)**	New	2 / 3
4. p(1,Y,1)	4a. **p(1,Y,1)**	
	4b. **p(1,2,1)**	3 / 4a / 4b
5. p(1,5,3)	5a. **p(1,5,1)**	3 / 4a / 4b / 5a

Multi-Threaded Tabling

- Despite the availability of both threads and tabling in Prolog compilers such as XSB, Yap and Ciao, the implementation of these two features such that they work together is not an easy task.
- Until now, XSB was the only system combining tabling with multi-threading:
  - **Private Tables**: each thread keeps it **own copy of the table space**, thus avoiding concurrency between threads.
  - **Shared Tables**: when a set of subgoals computed by different threads is mutually dependent, then a **usurpation operation** synchronizes threads and a single thread assumes the computation of all subgoals, turning the remaining threads into consumer threads.

Multi-Threaded Tabling

- The basis for our work is also on multi-threaded tabling using shared tables, but we propose an alternative view to XSB's approach.

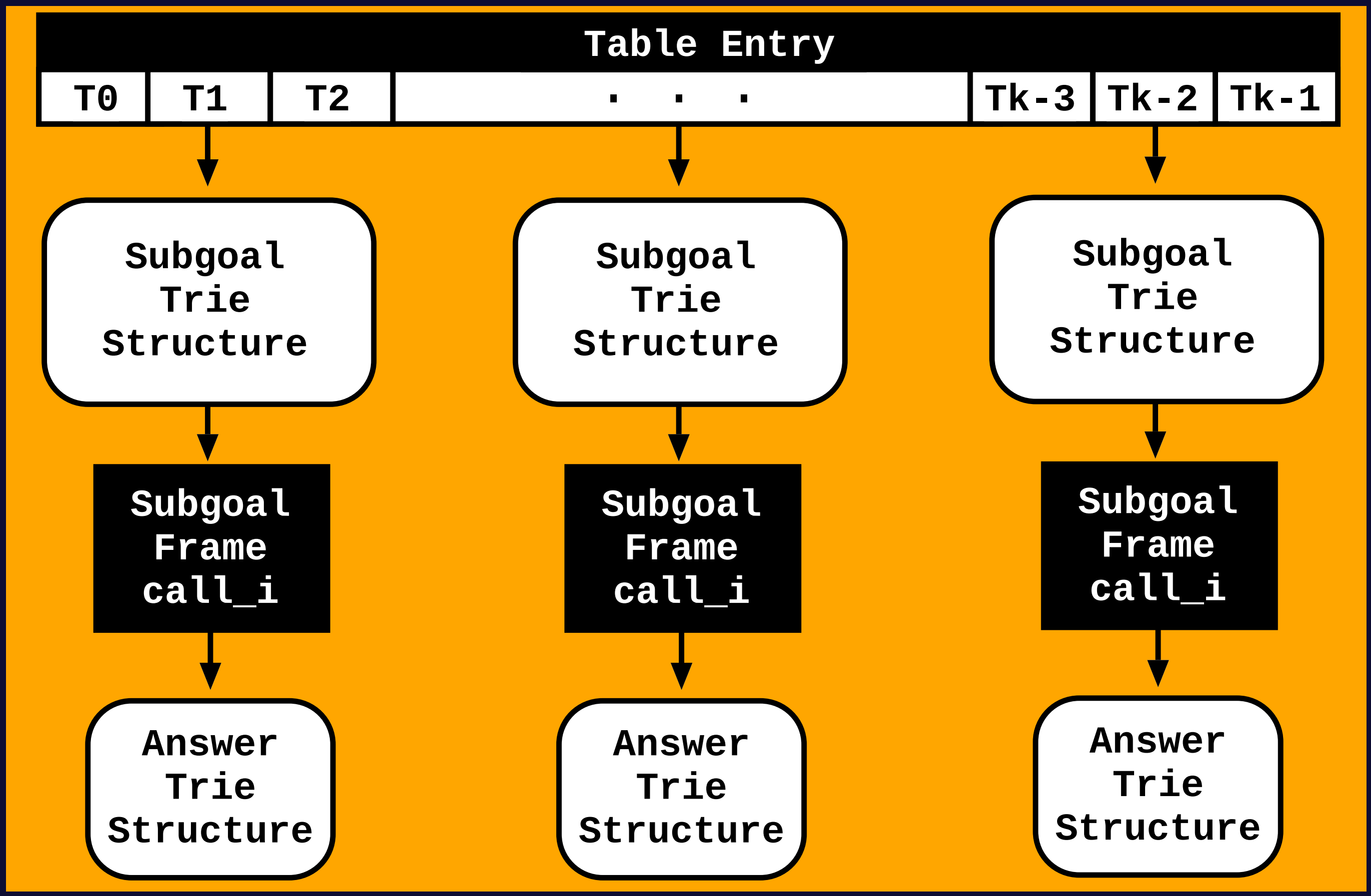
Multi-Threaded Tabling

- The basis for our work is also on multi-threaded tabling using shared tables, but we propose an alternative view to XSB's approach.
- In our proposal, **each thread was its own tables**, i.e., from the thread point of view the tables are private, but **at the engine level we use a common table space**, i.e., from the implementation point of view the tables are shared among all threads.

Multi-Threaded Tabling

- The basis for our work is also on multi-threaded tabling using shared tables, but we propose an alternative view to XSB's approach.
- In our proposal, **each thread was its own tables**, i.e., from the thread point of view the tables are private, but **at the engine level we use a common table space**, i.e., from the implementation point of view the tables are shared among all threads.
- We propose three designs for our common table space approach:
  - **No-Sharing** (similar to XSB with private tables)
  - **Subgoal-Sharing**
  - **Full-Sharing**

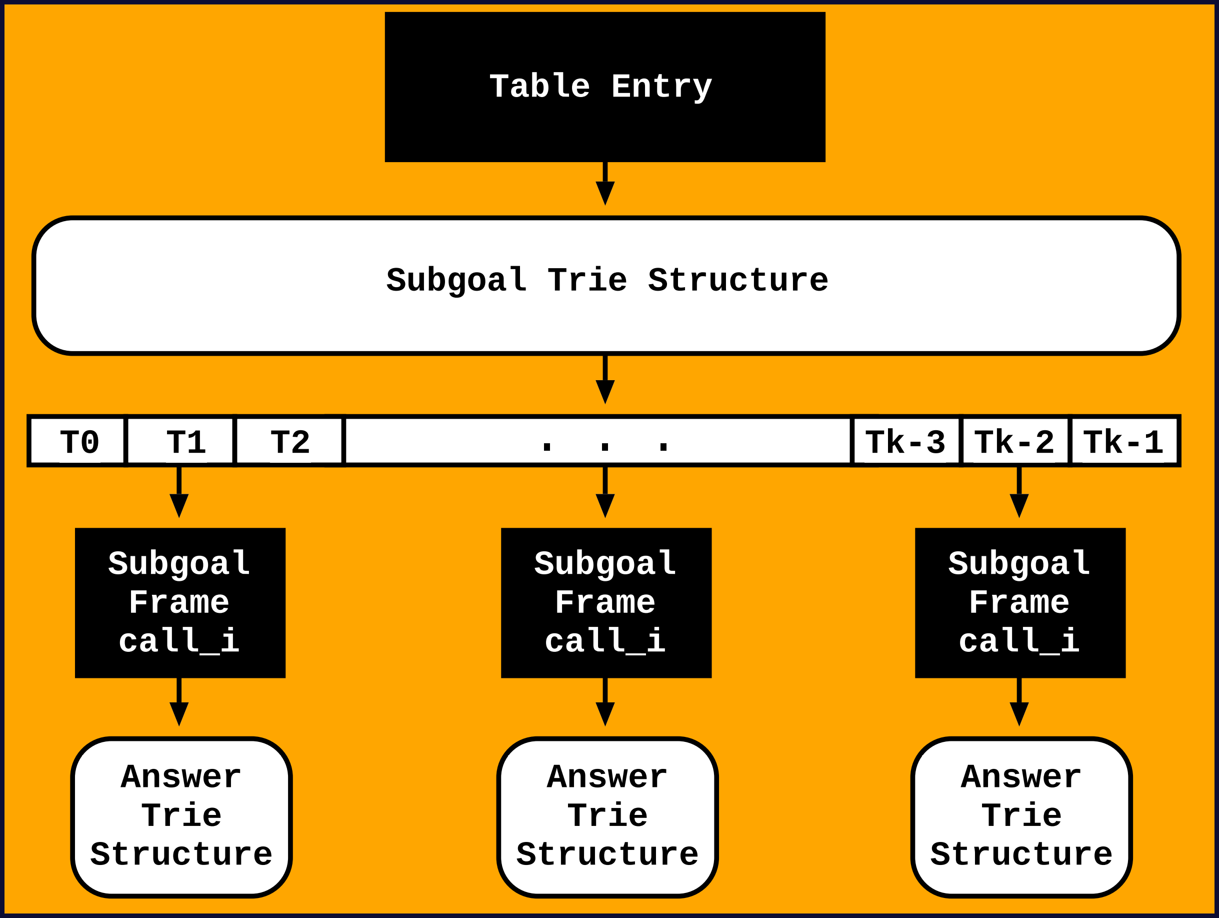
Multi-Threaded Tabling: No-Sharing



Memory usage for a table T assuming NK threads evaluating NS subgoals:

sizeof(TE) + sizeof(BA) +
[sizeof(STS)+ [sizeof(SF) + sizeof(ATS)] * NS] * NK

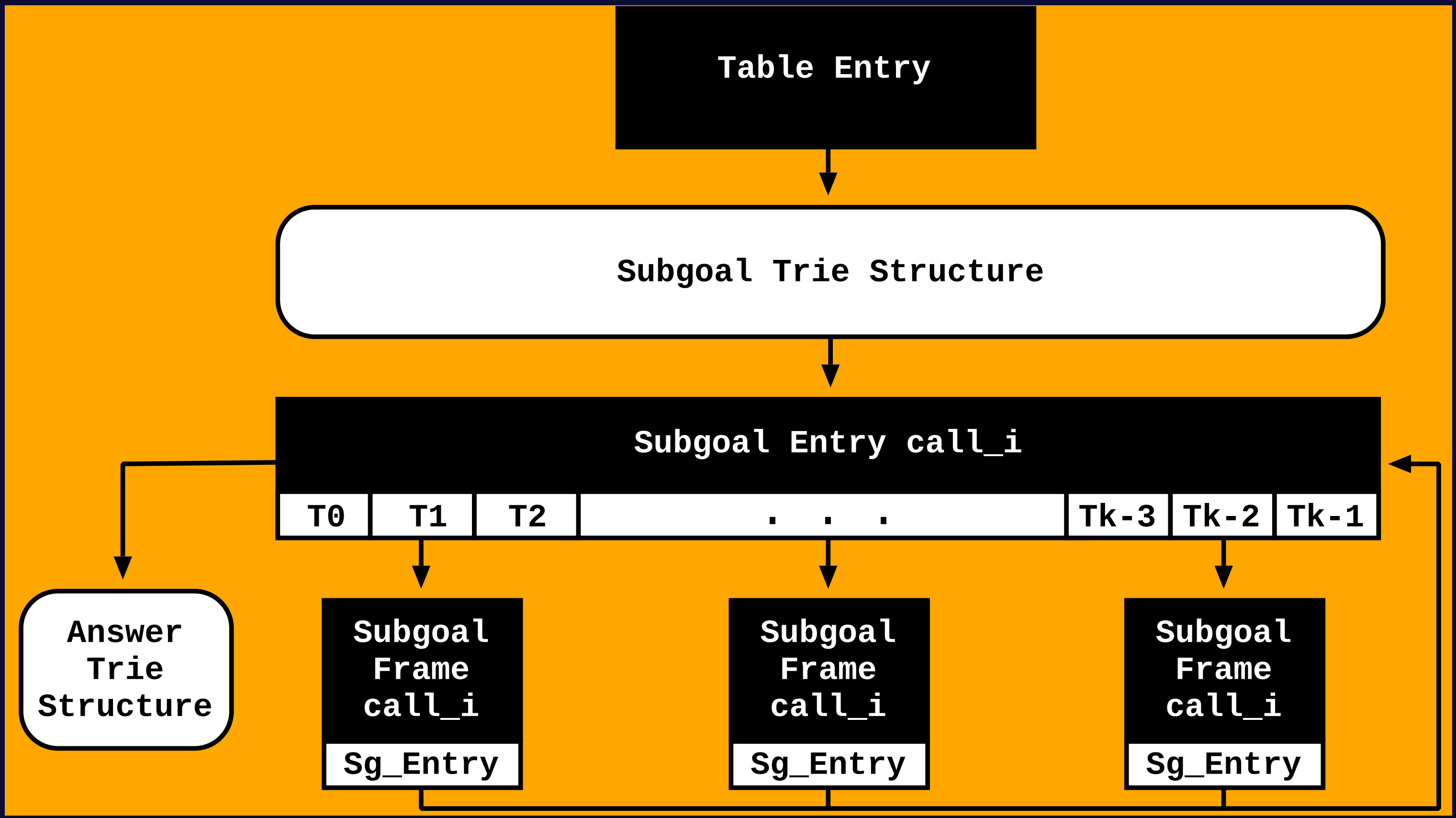
Multi-Threaded Tabling: Subgoal-Sharing



Memory usage for a table T assuming NK threads evaluating NS subgoals:

sizeof(TE) + sizeof(STS) +
[sizeof(BA) + [sizeof(SF) + sizeof(ATS)] * NK] * NS

Multi-Threaded Tabling: Full-Sharing



Memory usage for a table T assuming NK threads evaluating NS subgoals:

sizeof(TE) + sizeof(STS) +

[sizeof(SE) + sizeof(BA) + sizeof(ATS) + sizeof(SF)* NK] * NS

Multi-Threaded Tabling: Experimental Results

Design	Time	Memory
Pyramid 400		
NS	322,799	370,894,136
SS	**1.12**	**-448,984**
FS	**4.30**	**-346,599,384**
Cycle 400		
NS	209,678	247,351,736
SS	**1.13**	**-225,544**
FS	**4.84**	**-231,333,792**
Grid 20		
NS	193,419	247,351,736
SS	0.96	**-225,544**
FS	**1.14**	**-231,333,792**

Execution time (in milliseconds) and total memory usage (bytes) running 16 simultaneous threads, all executing the same query goal.

Multi-Threaded Tabling: Experimental Results

Design	Execution		Tries		Threads	
	TE	SF	STS	ATS	BA	SE
Pyramid 400						
NS	56	921,600	1,024,640	368,947,200	640	0
SS	0	0	**-960,600**	0	511,616	0
FS	0	**-307,200**	**-960,600**	**-345,888,000**	511,616	44,800
Cycle 400						
NS	56	461,952	513,920	246,375,168	640	0
SS	0	0	**-481,800**	0	256,256	0
FS	0	**-153,984**	**-481,800**	**-230,976,720**	256,256	22,456
Grid 20						
NS	56	461,952	513,920	246,375,168	640	0
SS	0	0	**-481,800**	0	256,256	0
FS	0	**-153,984**	**-481,800**	**-230,976,720**	256,256	22,456

Specific memory usage (bytes) running 16 simultaneous threads, all executing the same query goal.

Explicit Parallel Constructs

- Traditional parallel LP systems usually run in parallel mode from beginning to end and this may severely restrict parallelism when supporting sequential semantics.
  - If we avoid exploiting non-leftmost sub-computations, we may be **restricting the granularity** of the available parallel work.
  - If we allow such sub-computations, we may be executing **speculative work and/or side-effects** that would not be done in a sequential system.

Explicit Parallel Constructs

- Traditional parallel LP systems usually run in parallel mode from beginning to end and this may severely restrict parallelism when supporting sequential semantics.
  - If we avoid exploiting non-leftmost sub-computations, we may be **restricting the granularity** of the available parallel work.
  - If we allow such sub-computations, we may be executing **speculative work and/or side-effects** that would not be done in a sequential system.
- However, most of the execution time is spent in a parallel application is spent in computations that are inherently parallel and independent and only a **small part of the execution time is spent in sequential parts of code**:
  - Initialization code
  - Code to partitioning the data into small sub-tasks
  - Code to aggregate/reduce data from different sub-tasks

Explicit Parallel Constructs

- Most of the recent proposals on parallel programming, where parallelism is exploited explicitly, are trying to **encapsulate** some of the low-level details in more **high-level explicit parallel constructs for well-know patterns** and let the execution model implement them **implicitly**:
  - **OpenMP**
  - **Intel Threading Building Blocks**
  - **Map-Reduce**

Explicit Parallel Constructs

- Most of the recent proposals on parallel programming, where parallelism is exploited explicitly, are trying to **encapsulate** some of the low-level details in more **high-level explicit parallel constructs for well-know patterns** and let the execution model implement them **implicitly**:
  - **OpenMP**
  - **Intel Threading Building Blocks**
  - **Map-Reduce**

- Our approach goes in the opposite direction. It establishes its **foundations on implicit parallelism** and relies on **high-level explicit parallel constructs** to trigger parallel execution.
  - **More declarative**, thus simplifying parallel programming;
  - **Better performance**, since we can benefit from the intrinsic and strong potential that LP has for implicit parallelism;
  - **More general**, can be easily generalized to implement new parallel constructs with minor changes to the low-level parallel engine.

Explicit Parallel Constructs

- Some basic parallel constructs we are interested in are:
  - **parallel/1**
  - **parallel_findall/3**
  - **parallel_once/1**

Explicit Parallel Constructs

- Some basic parallel constructs we are interested in are:
  - **parallel/1**
  - **parallel_findall/3**
  - **parallel_once/1**

```
go1 :- statistics(cputime,[Init,_]),
       parallel(benchmark),
       statistics(cputime,[End,_]),
       Time is End - Init, writeln(Time).
```

Explicit Parallel Constructs

- Some basic parallel constructs we are interested in are:
  - **parallel/1**
  - **parallel_findall/3**
  - **parallel_once/1**

```
go1 :- statistics(cputime,[Init,_]),
       parallel(benchmark),
       statistics(cputime,[End,_]),
       Time is End - Init, writeln(Time).

go2 :- init_something,
       parallel_findall(X,benchmark(X),L),
       do_something_with_results(L).
```

Explicit Parallel Constructs

- As in OpenMP, we can extend the parallel constructs to include **pre-defined directives** that can be used to instruct and/or to pass specific information to the execution system about the computation at hand:
  - **num_workers(expr)**
  - **execution_model(env_copying | stack_splitting)**
  - **if(expr)**
  - **reduction(var,operator)**
  - **cut_safe**
  - **allow_out_of_order_side_effects**

Explicit Parallel Constructs

- As in OpenMP, we can extend the parallel constructs to include **pre-defined directives** that can be used to instruct and/or to pass specific information to the execution system about the computation at hand:
  - **num_workers(expr)**
  - **execution_model(env_copying | stack_splitting)**
  - **if(expr)**
  - **reduction(var,operator)**
  - **cut_safe**
  - **allow_out_of_order_side_effects**

```
go :- init_something(I),
      parallel(benchmark(X),[if(I),reduction(X,sum),cut_safe]),
      do_something_with_result(X).
```

Teams of Workers for Shared/Distributed Memory

- In the past, we have already designed and developed or-parallel systems for shared and distributed memory architectures:
  - **Shared Memory**: support for implicit or-parallelism based on the **environment copying model**.
  - **Distributed Memory**: support for implicit or-parallelism based on the **stack splitting model**.

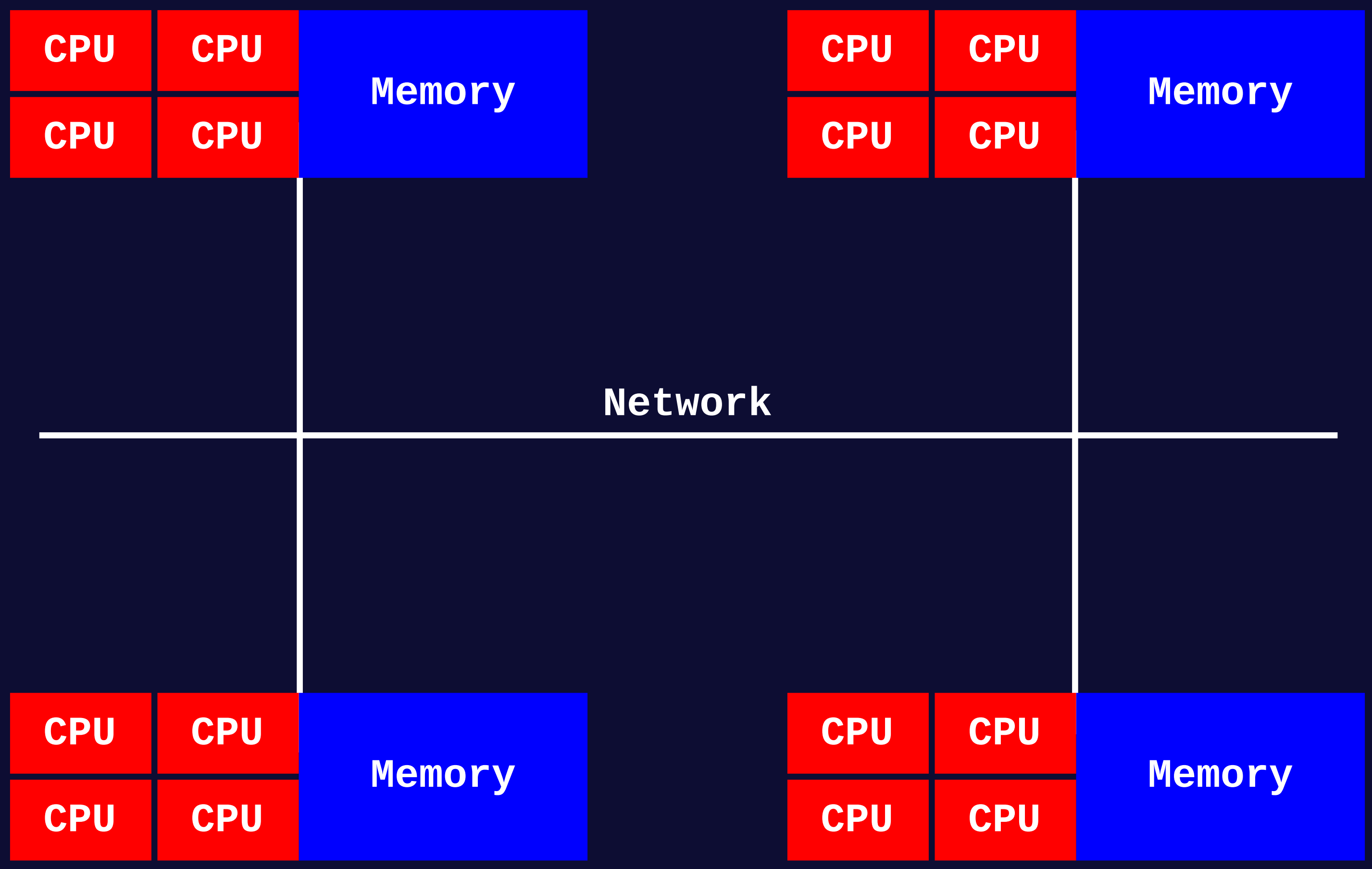
Teams of Workers for Shared/Distributed Memory

- In the past, we have already designed and developed or-parallel systems for shared and distributed memory architectures:
  - **Shared Memory**: support for implicit or-parallelism based on the **environment copying model**.
  - **Distributed Memory**: support for implicit or-parallelism based on the **stack splitting model**.
- Design a new parallel platform that will be able to **take advantage of both models to scale-up on clusters of multi-core processors**.
- For that, we will consider **Teams of Workers**, i.e., workers sharing the same memory address space. Workers executing in different computer nodes cannot belong to the same team, but we can have more than a team in the same computer node.

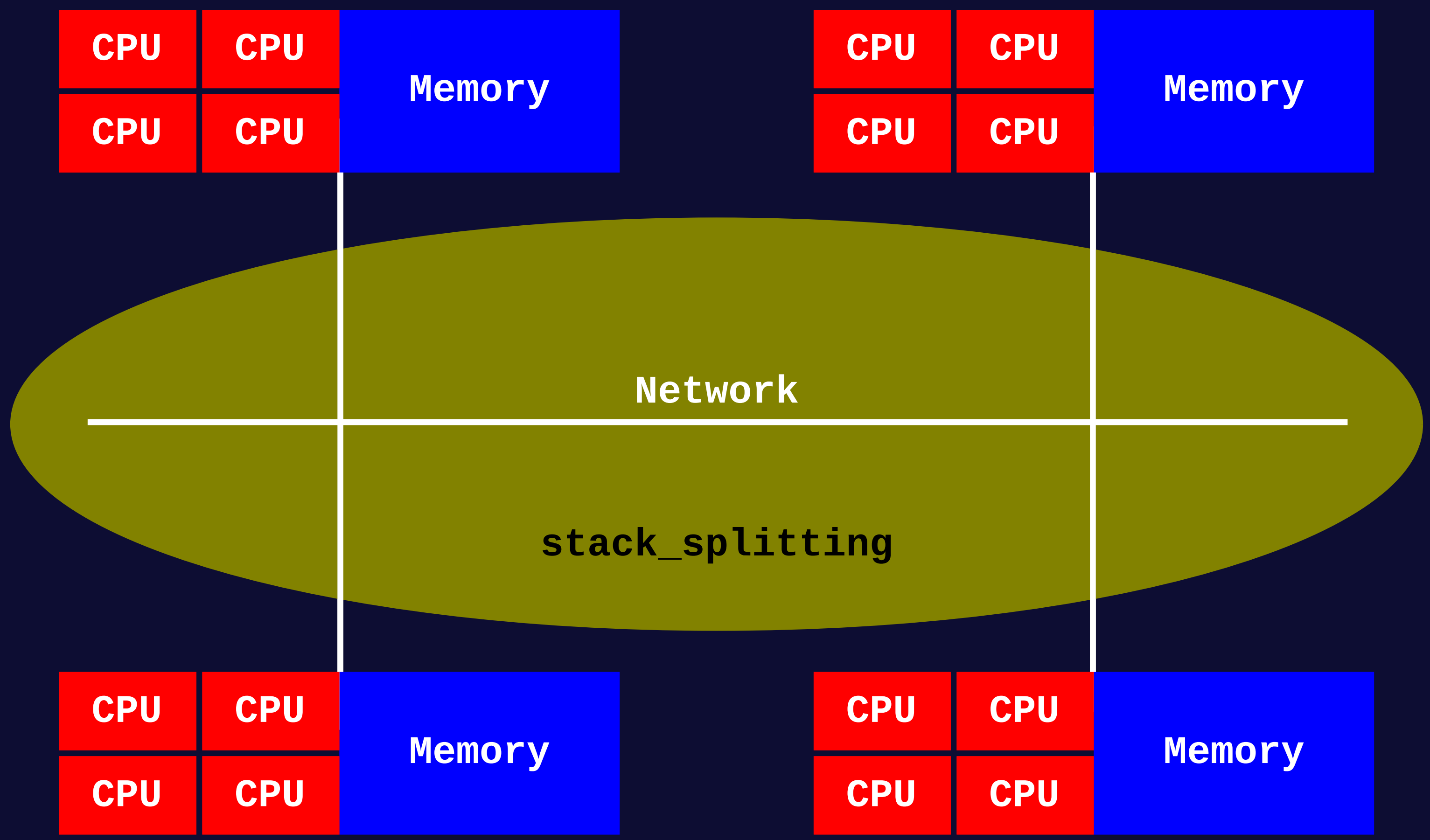
Teams of Workers for Shared/Distributed Memory

- Workers inside a team (shared memory only) can distribute work using:
  - **Environment Copying**
  - **Stack Splitting**
- Teams of workers can distribute work using:
  - **Environment Copying** (shared memory only)
  - **Stack Splitting** (shared and distributed memory)
- This idea is similar to the MPI/OpenMP hybrid programming pattern where MPI is usually used to communicate work among workers in different computer nodes and OpenMP is used to communicate work among workers in the same node.
- By invoking our explicit parallel constructs with proper directives, we will be able to trigger parallel execution of these different combinations of number of workers, teams of workers and execution models.

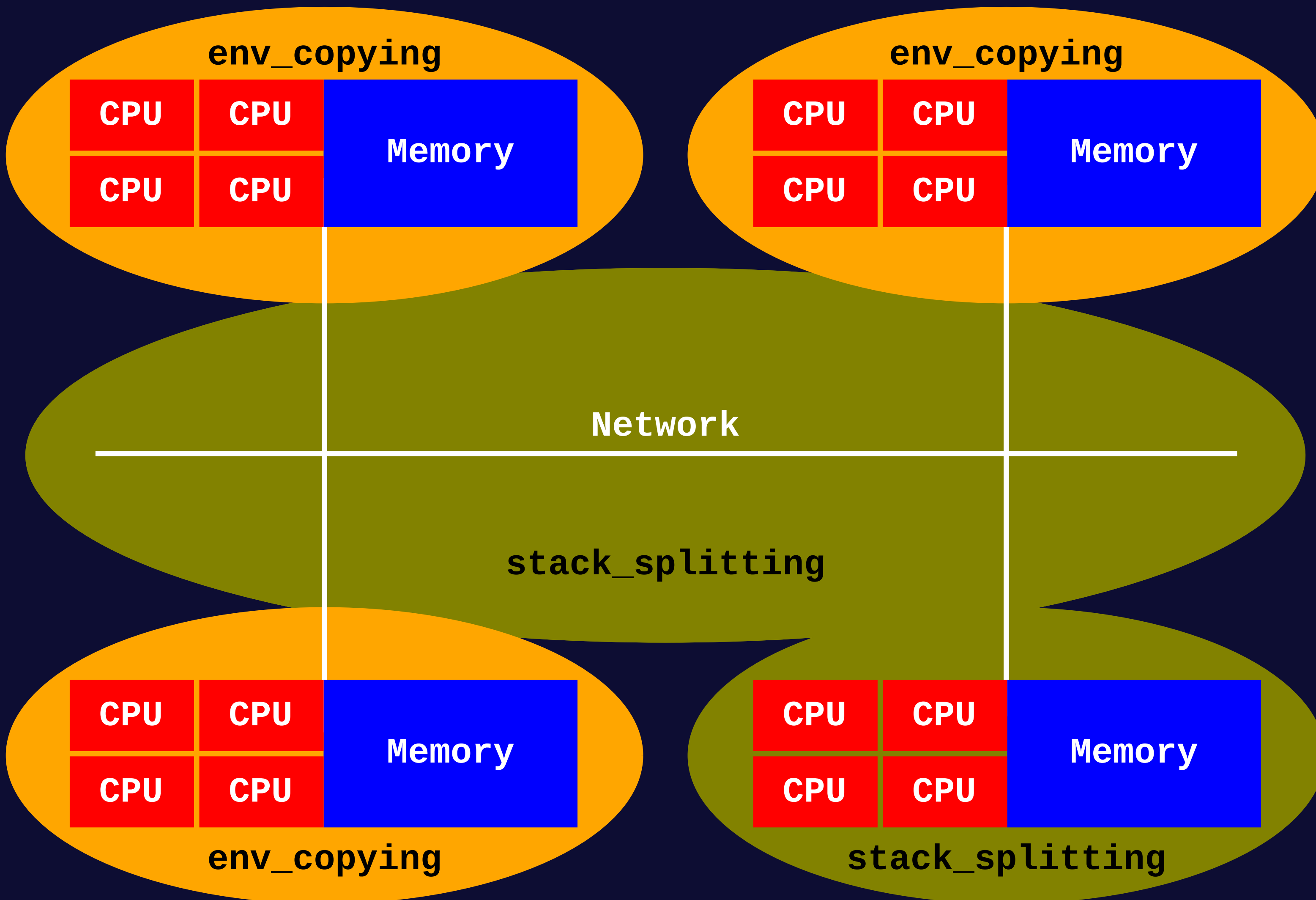
Teams of Workers for Shared/Distributed Memory



Teams of Workers for Shared/Distributed Memory



Teams of Workers for Shared/Distributed Memory



Thank You!

Ricardo Rocha
CRACS & INESC TEC
University of Porto, Portugal

ricroc@dcc.fc.up.pt
http://www.dcc.fc.up.pt/∼ricroc

Yap Prolog: *http://www.dcc.fc.up.pt/∼vsc/Yap*
Project LEAP: *http://www.dcc.fc.up.pt/leap*