On the Correctness and Efficiency of Lock-Free Expandable Tries for Tabled Logic Programs

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*

PADL, San Diego, January 2014

Tabling in Prolog Systems

- **Tabling** is an implementation technique that overcomes some of the limitations of Prolog resolution.
  - 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 resolution.
  - 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.

Tabling in Prolog Systems

- **Tabling** is an implementation technique that overcomes some of the limitations of Prolog resolution.
  - 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
  - **Yap Prolog [ICLP 2012]**

Table Space - Example

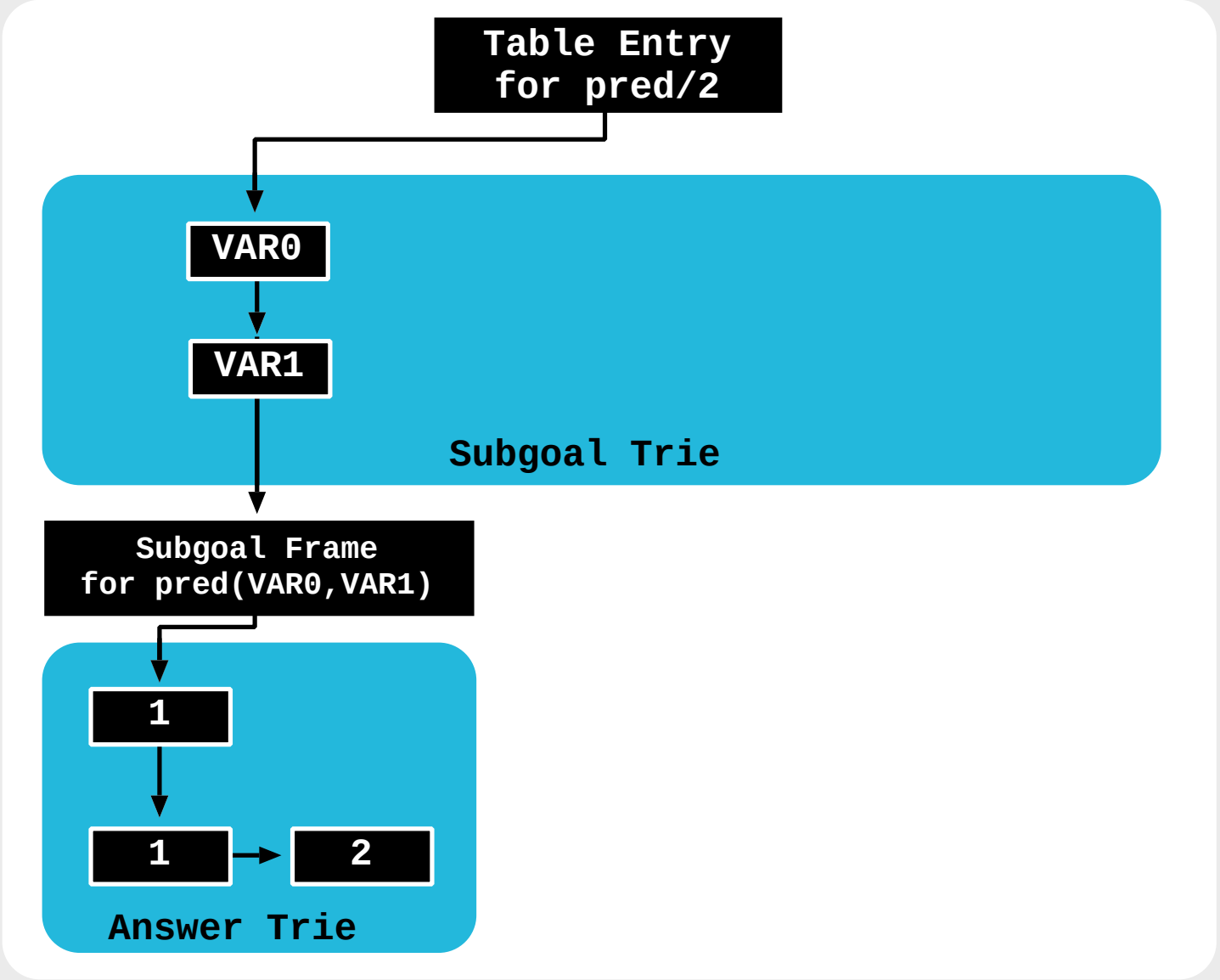


Table Space - Example

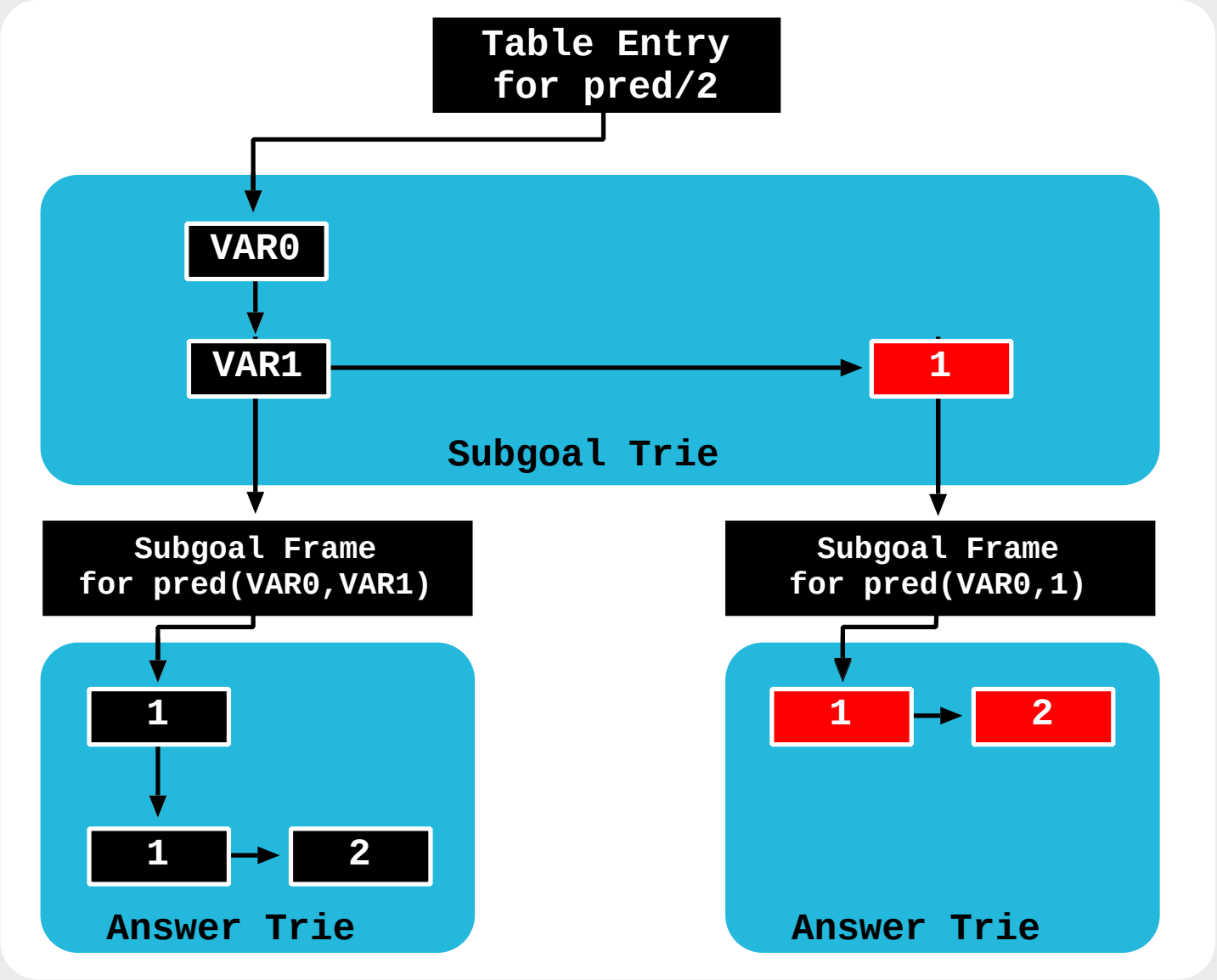


Table Space - Trie Level Internals

- All trie levels have one **parent (P)** node and at least one **child (C)** node.
- Only **search** and **insert** operations are executed on the trie levels.
- **Insertion** of new nodes is done on the **head of the chain**, until a **threshold** is achieved.

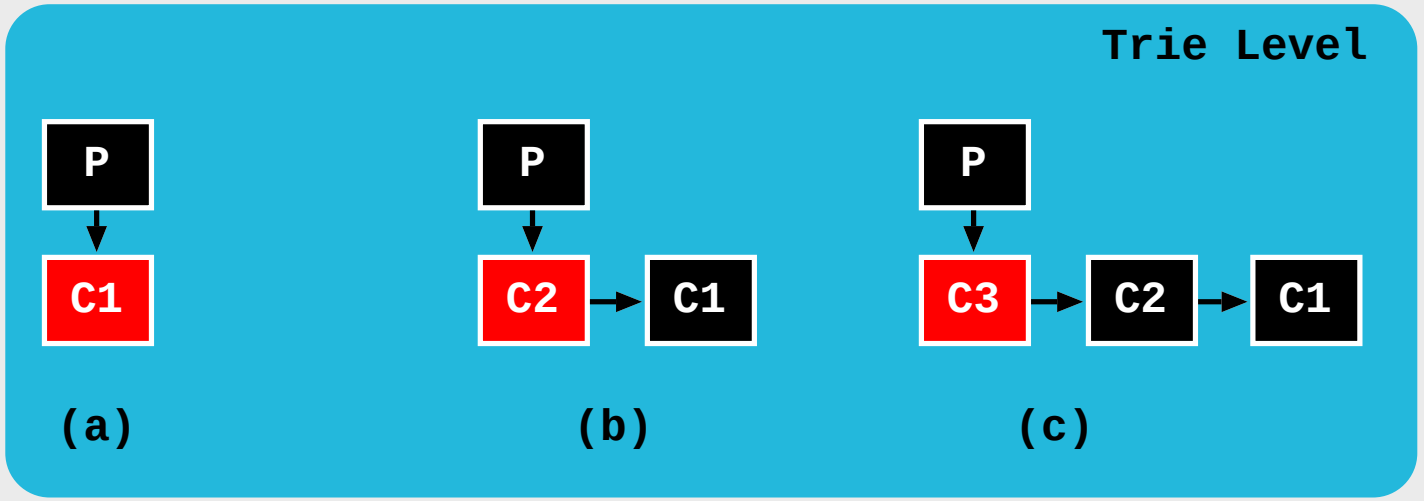


Table Space - Trie Level Internals

- When the **threshold** is achieved, a **hashing mechanism with separating chaining** is added to the level.
- The **hash H** node stores generic information about the level.
- The **value K** is the number of bucket entries.
- When the hash becomes **saturated**, it is **expanded** to a new hash with $2 * K$ bucket entries.

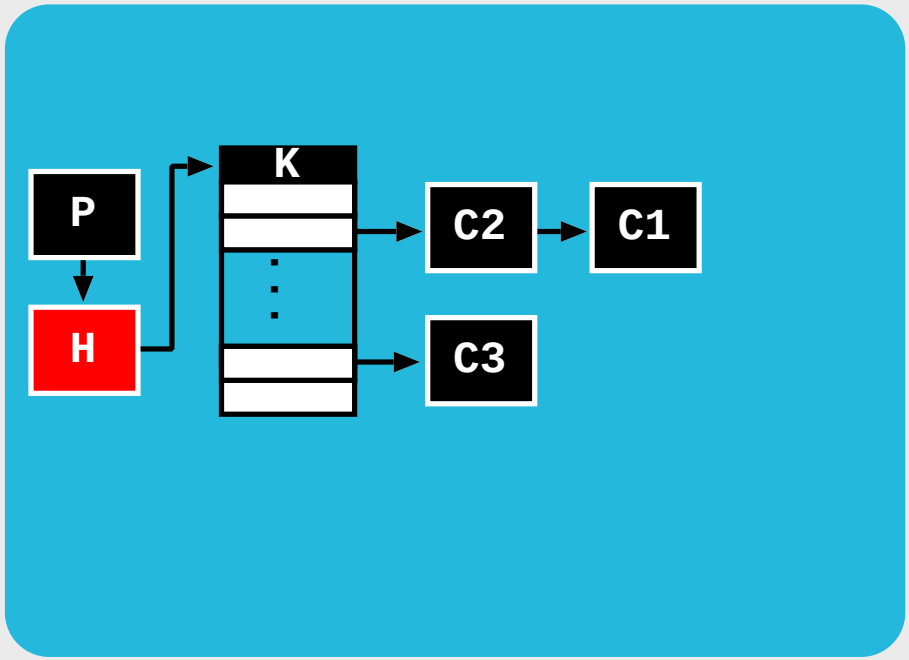


Table Space - Trie Level Internals

- When the **threshold** is achieved, a **hashing mechanism with separating chaining** is added to the level.
- The **hash H** node stores generic information about the level.
- The **value K** is the number of bucket entries.
- When the hash becomes **saturated**, it is **expanded** to a new hash with $2 * K$ bucket entries.

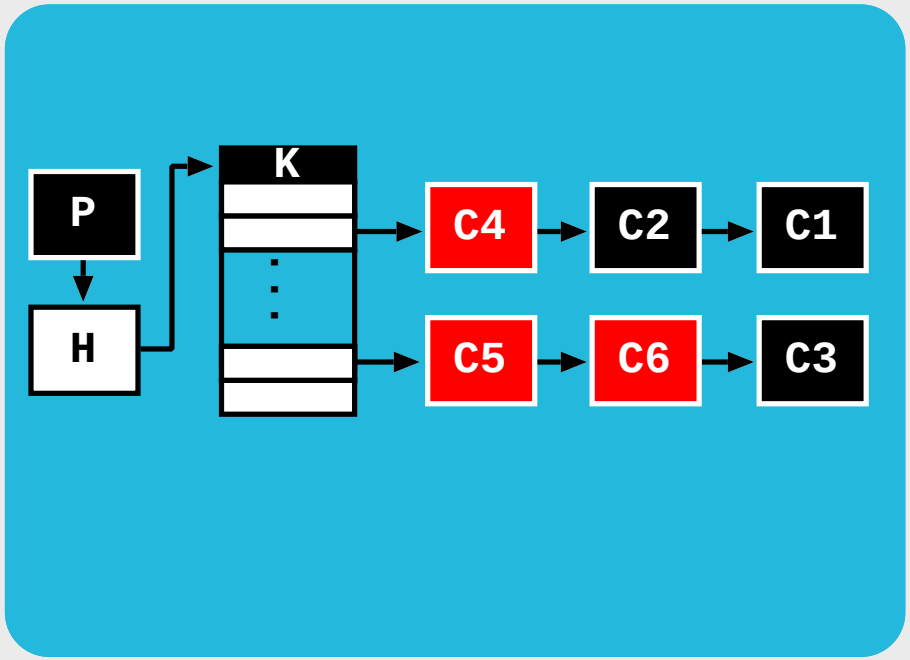
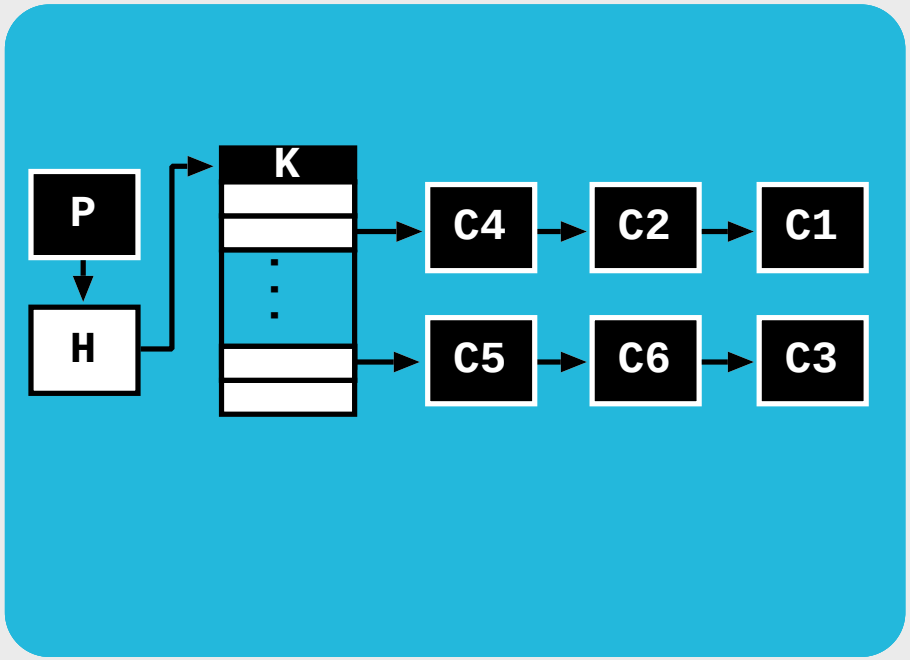


Table Space - Trie Level Internals

- When the **threshold** is achieved, a **hashing mechanism with separating chaining** is added to the level.
- The **hash H** node stores generic information about the level.
- The **value K** is the number of bucket entries.
- When the hash becomes **saturated**, it is **expanded** to a new hash with $2 * K$ bucket entries.



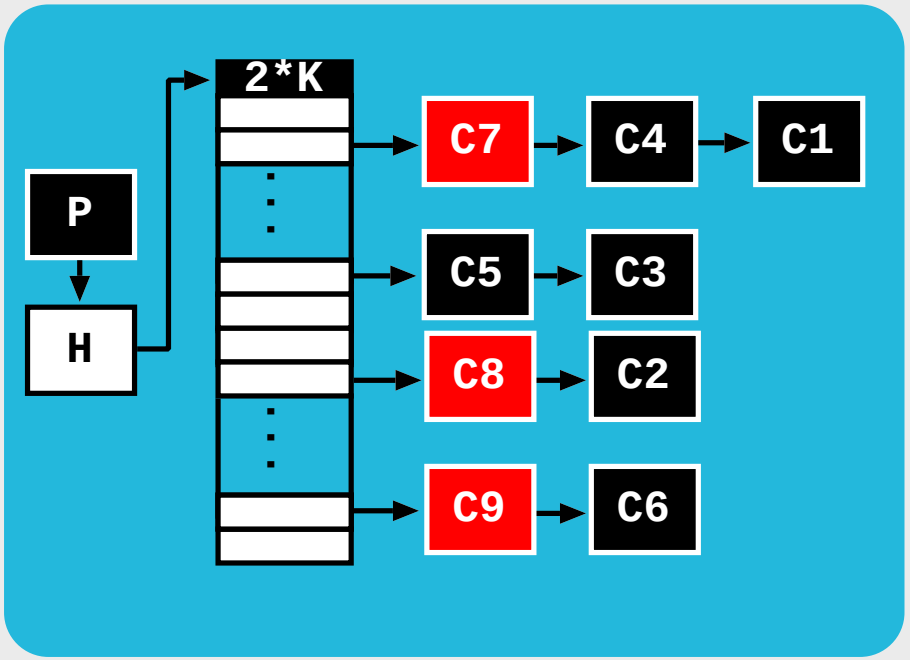
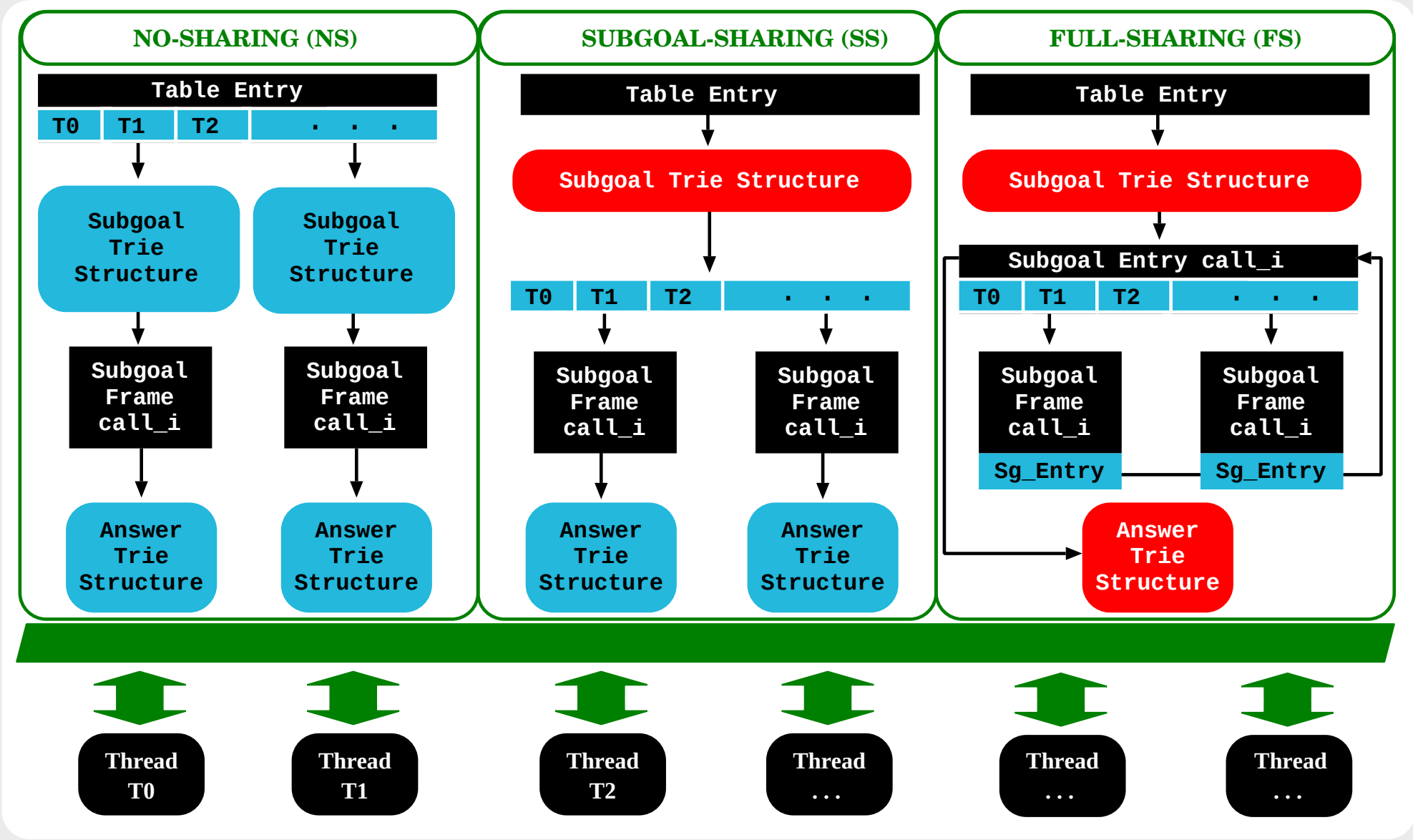


Table Space - Multithreaded Designs



Our Approach - Basic Concepts

- **Until now** to deal with concurrency we used **locks**:
  - Lock Type:
    - Standard Locks
    - TryLocks.
  - Lock Location:
    - Field per trie node
    - Global array of lock entries.
- The **expansion** of the hash **locked** the **insertion** and could in some cases **delay** the **search** operation (inefficiency).

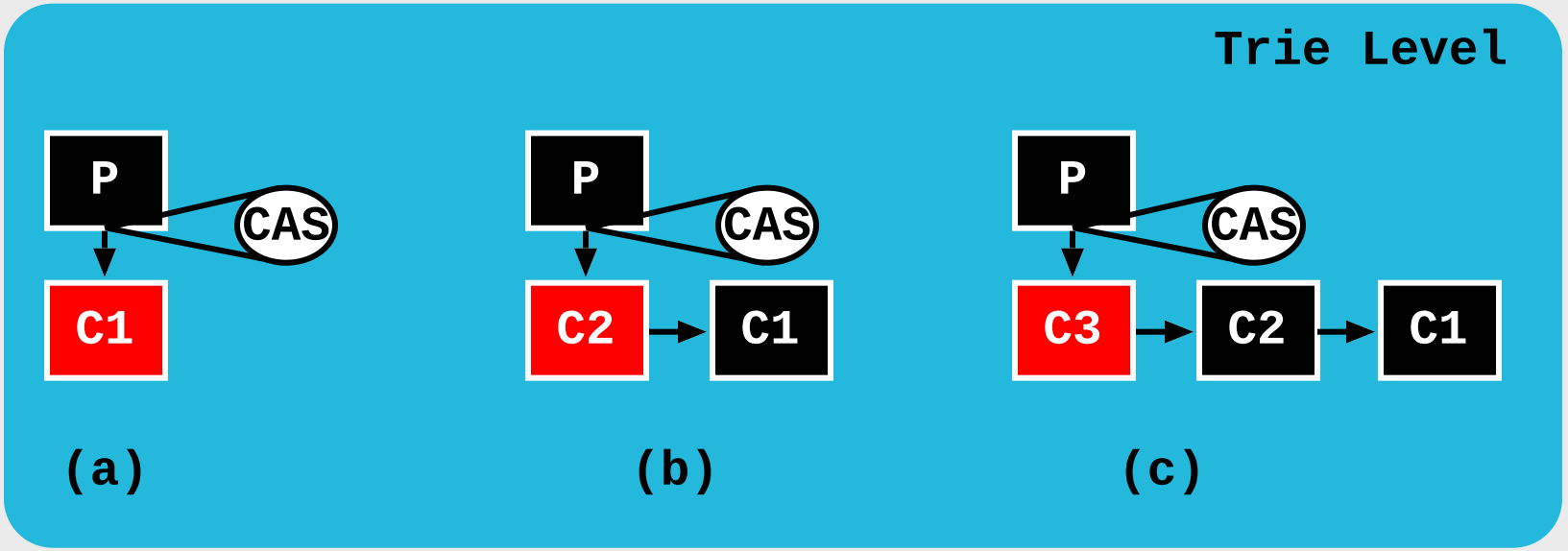
Our Approach - Basic Concepts

- **Until now** to deal with concurrency we used **locks**:
  - Lock Type:
    - Standard Locks
    - TryLocks.
  - Lock Location:
    - Field per trie node
    - Global array of lock entries.
- The **expansion** of the hash **locked** the **insertion** and could in some cases **delay** the **search** operation (inefficiency).
- **With this work** we are interested in **reducing** the granularity of the **synchronization**, by taking advantage of the **CAS (Compare-and-Swap)** operation.
  - Nowadays **can be found** on many of the **common architectures**.
  - At the **heart** of many **lock-free objects**.

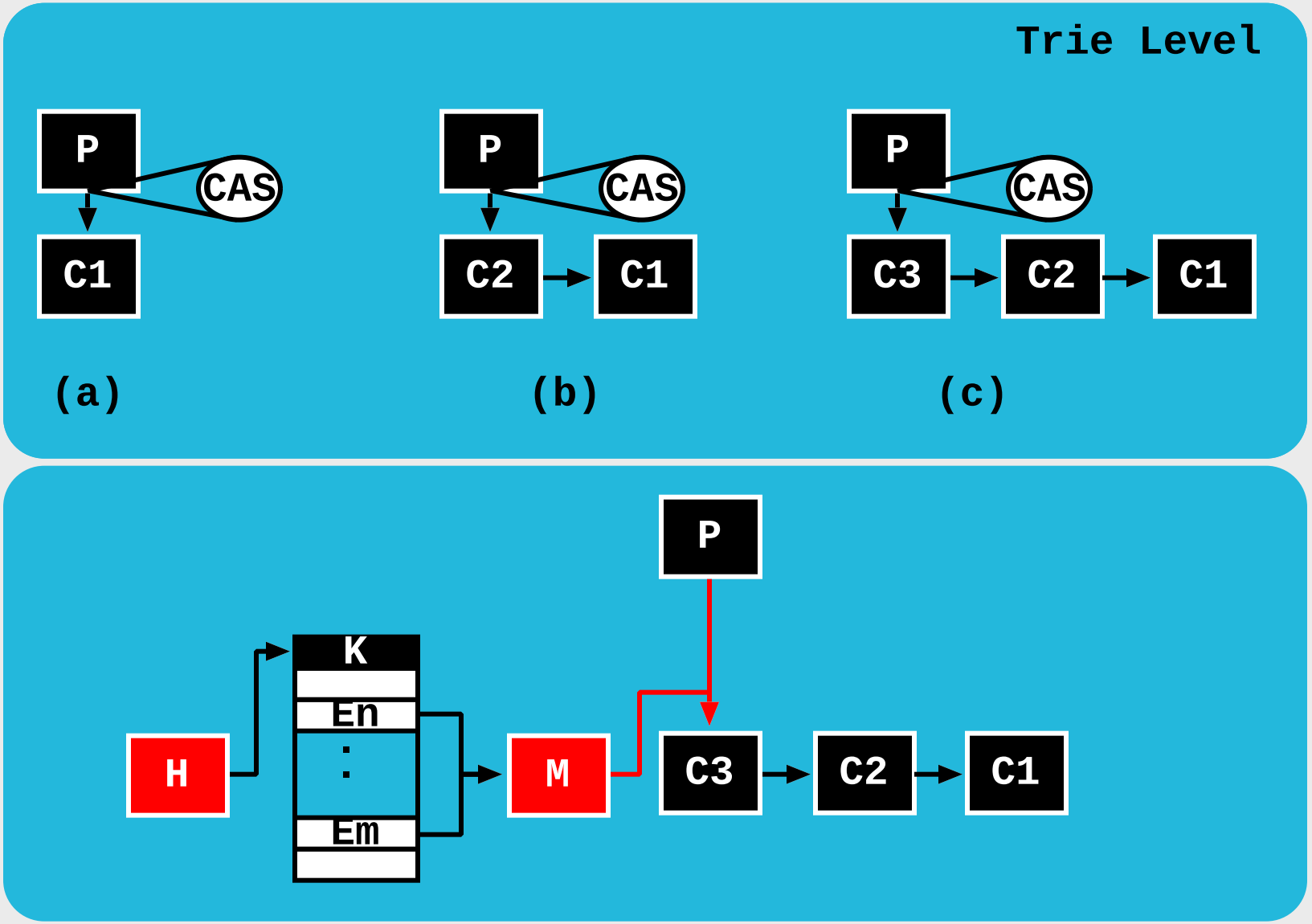
Our Approach - Basic Concepts

- **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**.
- **None** of the existent models is specifically **aimed** for an environment with the **characteristics** of our **tabling framework**.
  - Support for **deletion** of nodes **increases** the **complexity** of the models.
  - The model should be as **efficient** as possible on **search operation** (Completion of Table Space).

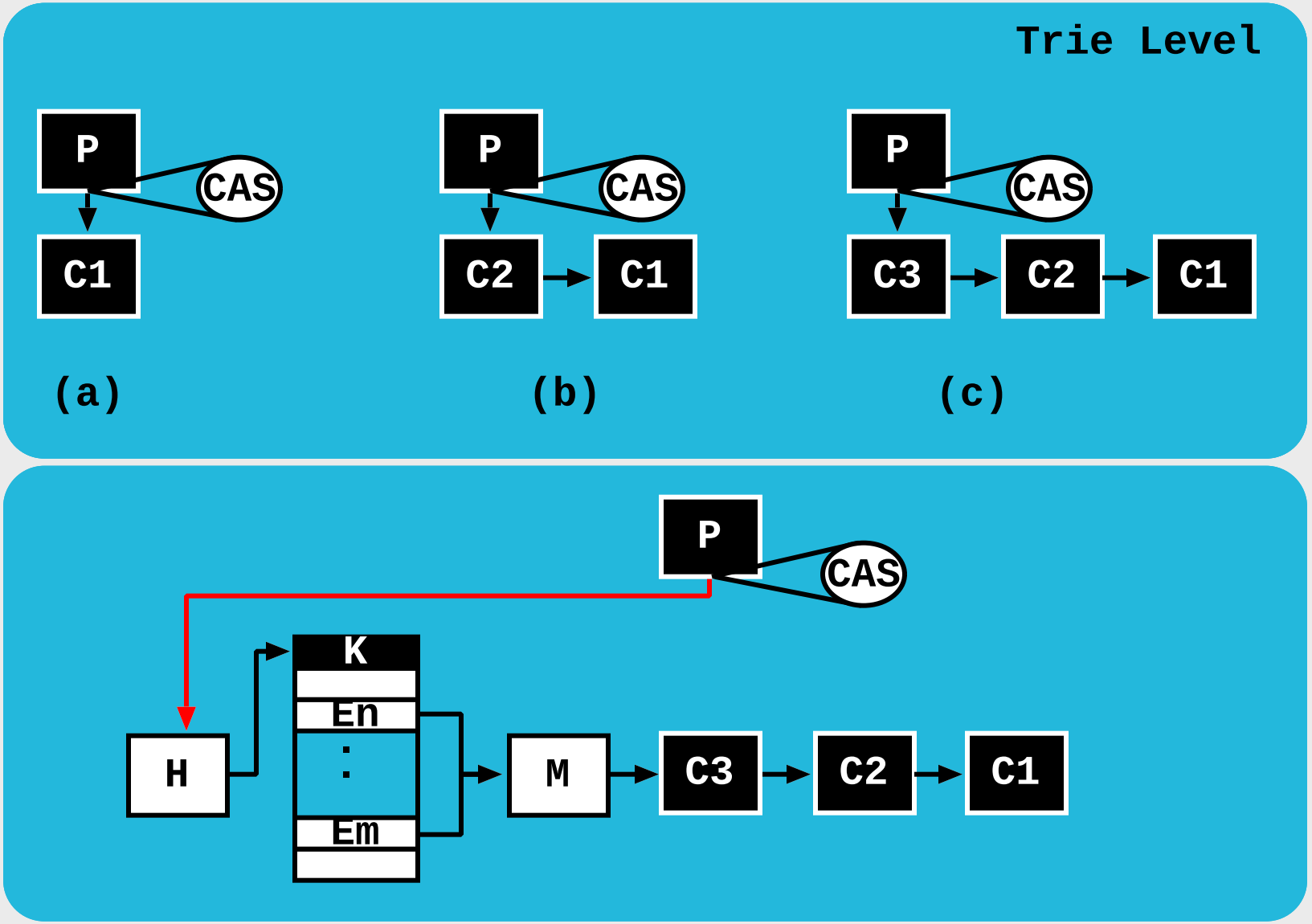
Our Approach - The First Expansion



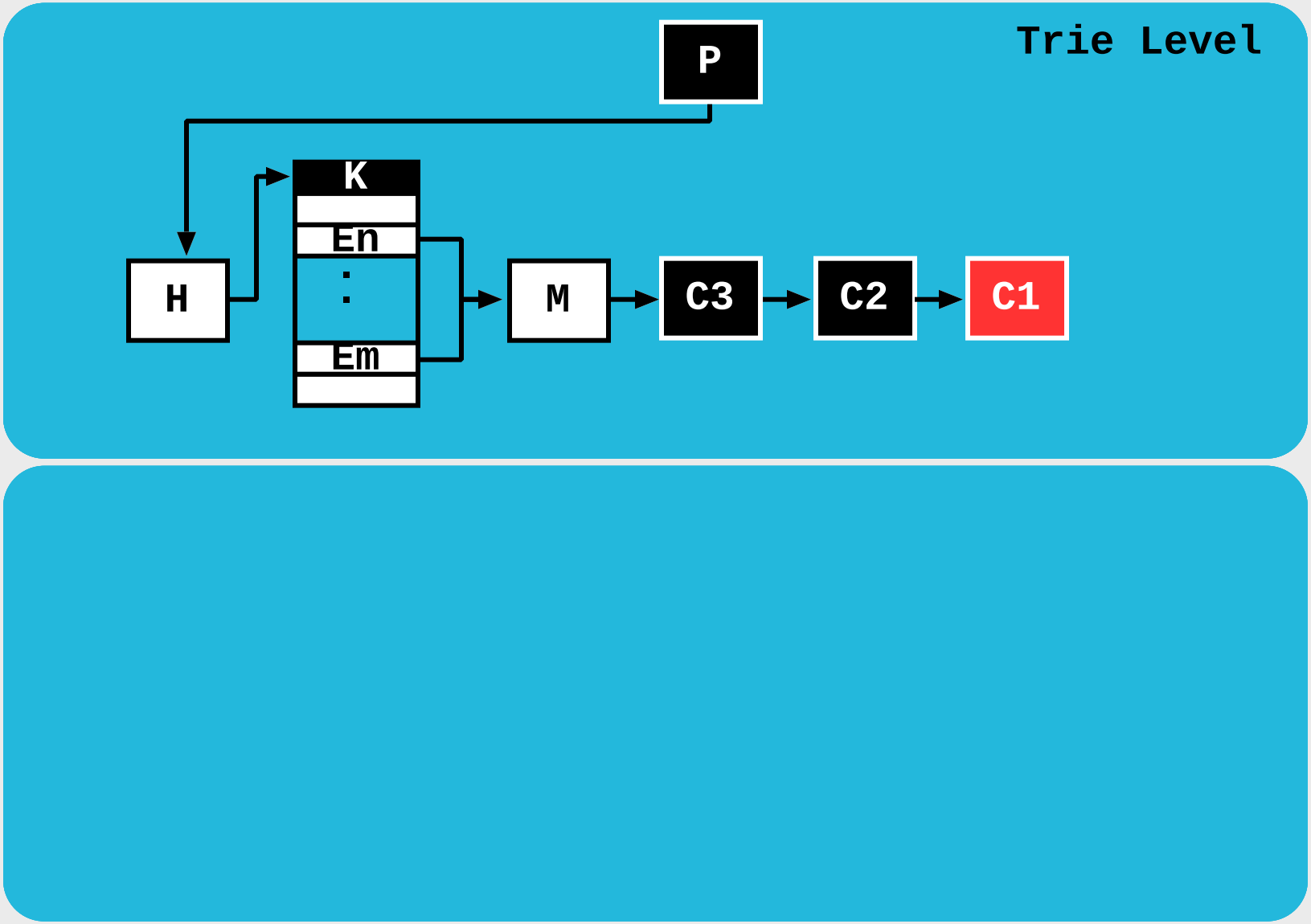
Our Approach - The First Expansion



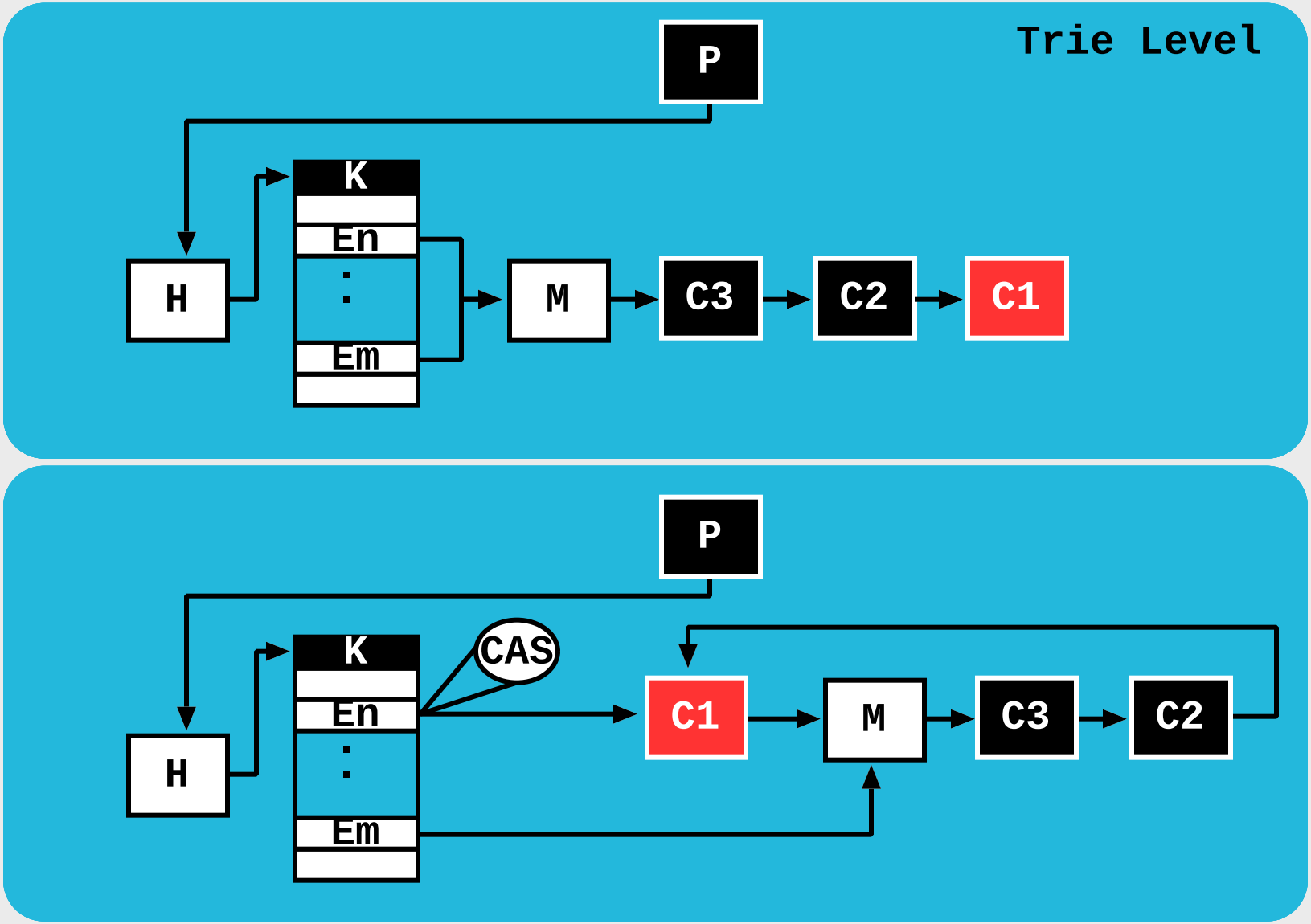
Our Approach - The First Expansion



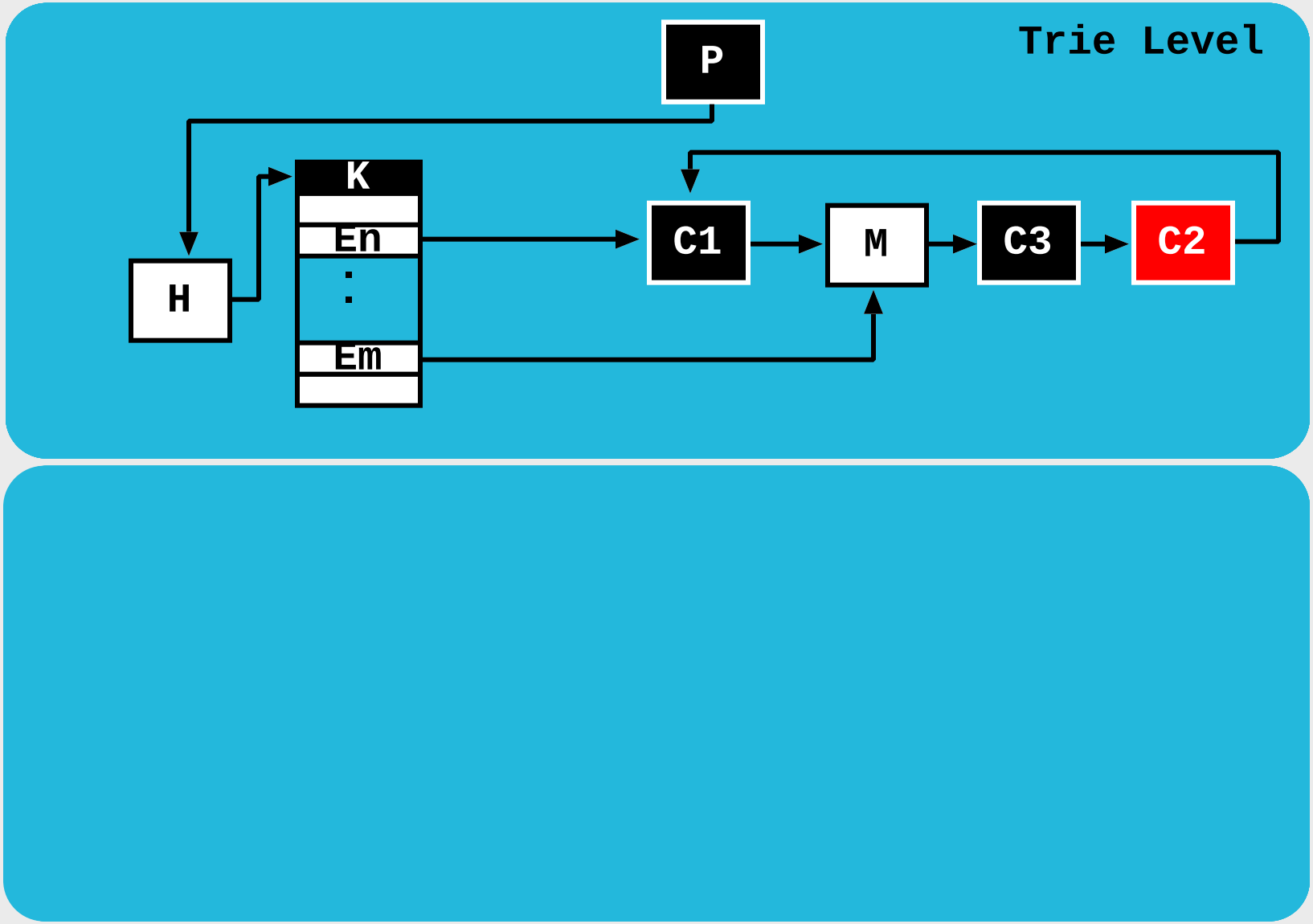
Our Approach - The First Expansion



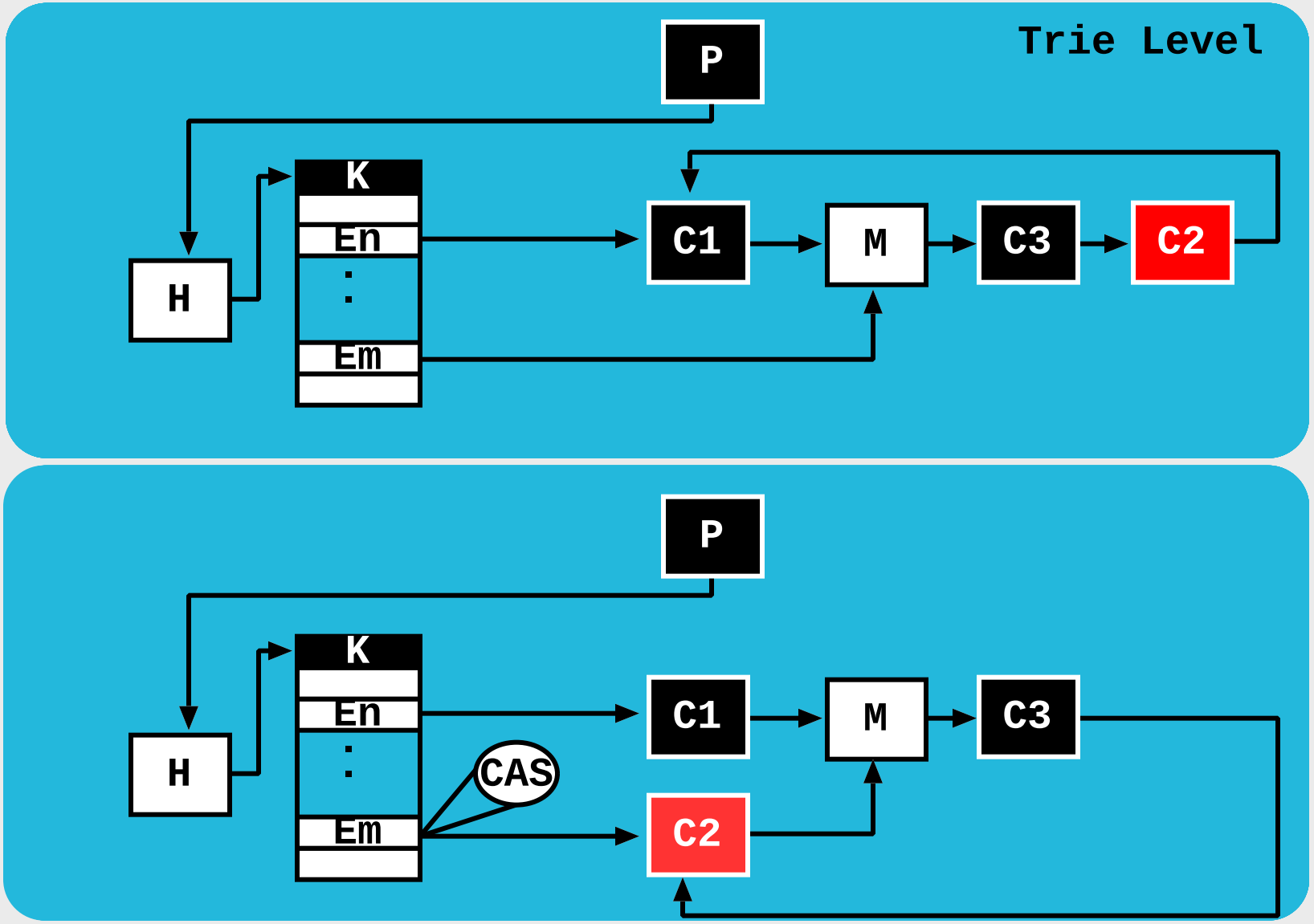
Our Approach - The First Expansion



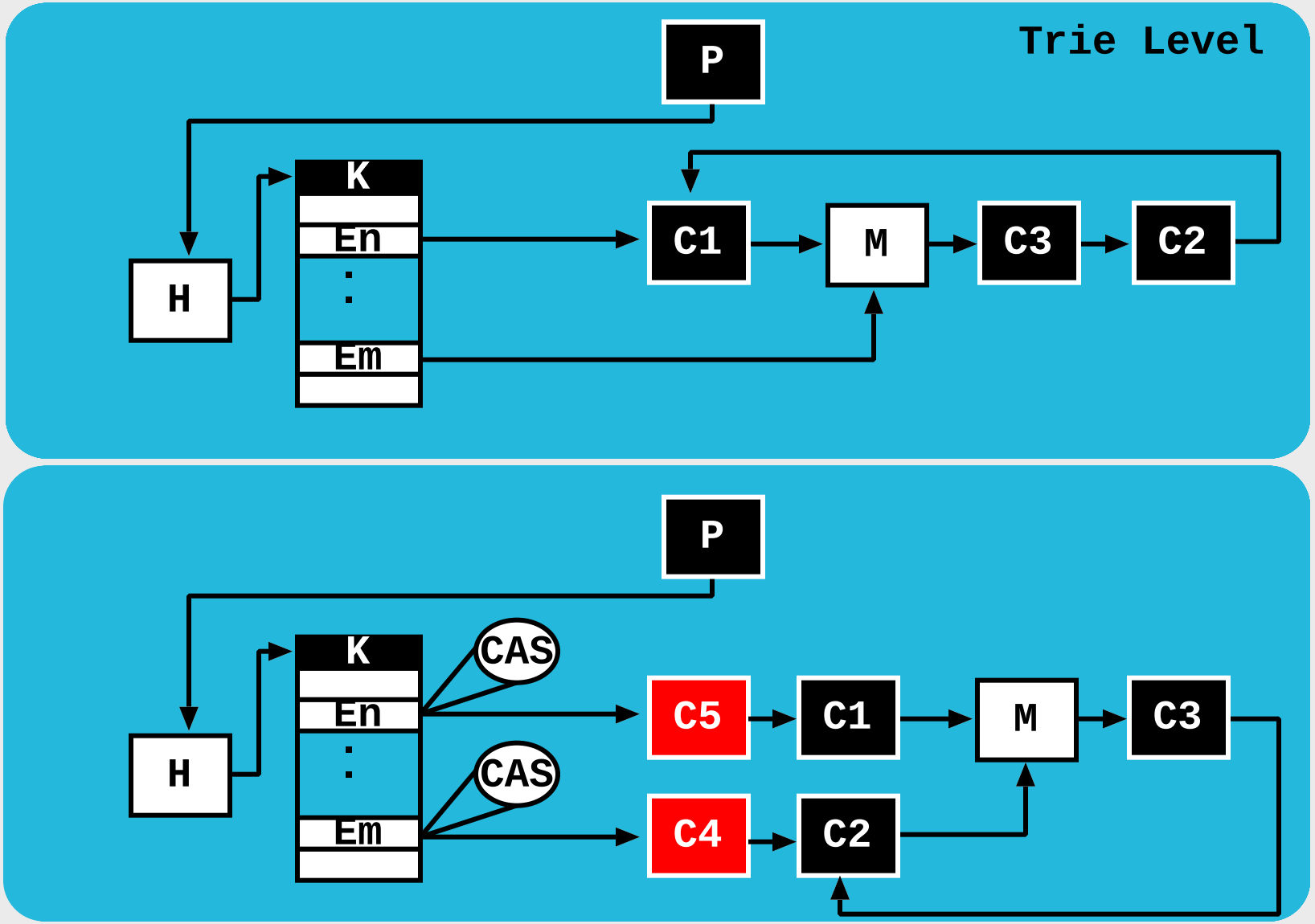
Our Approach - The First Expansion



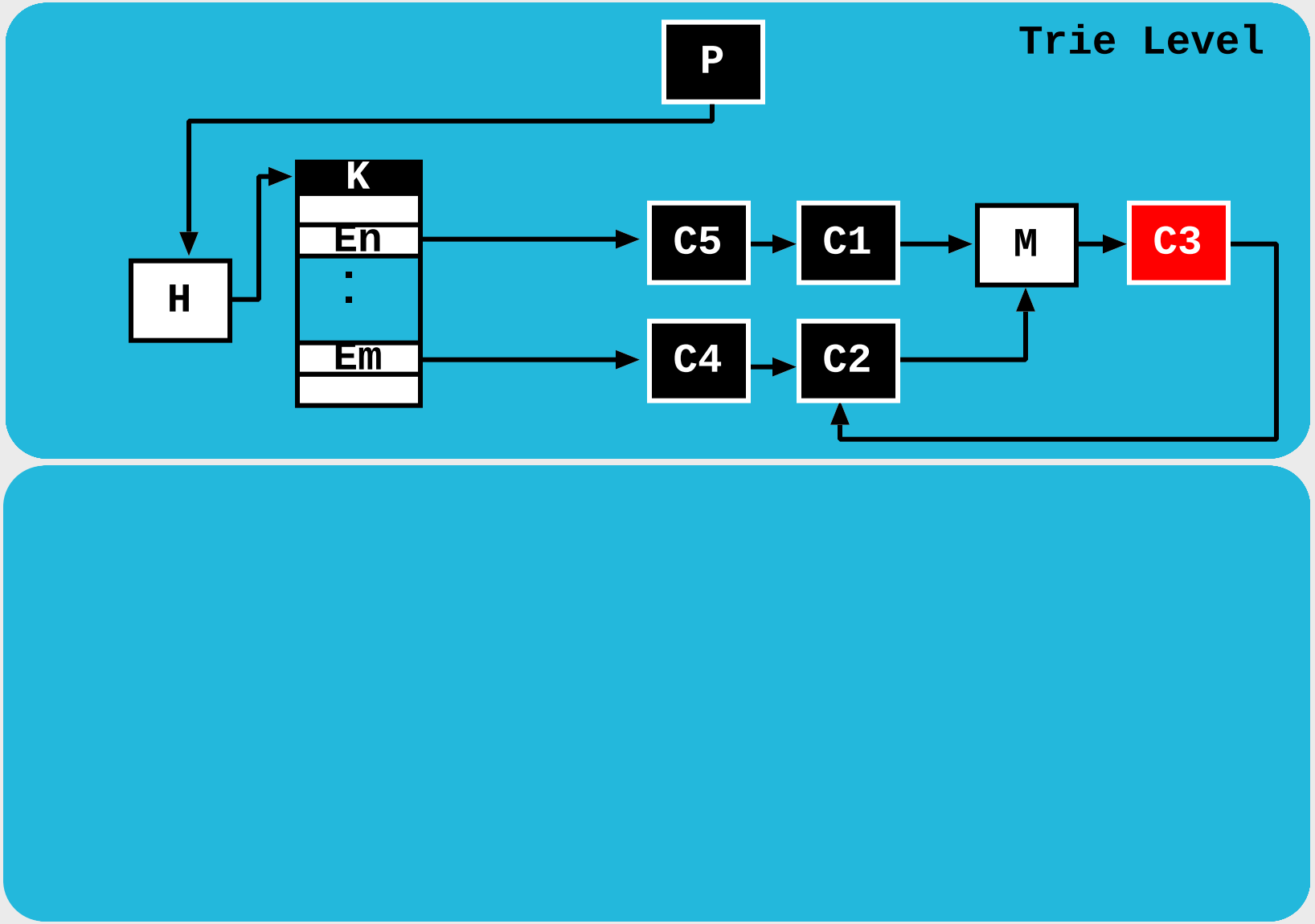
Our Approach - The First Expansion



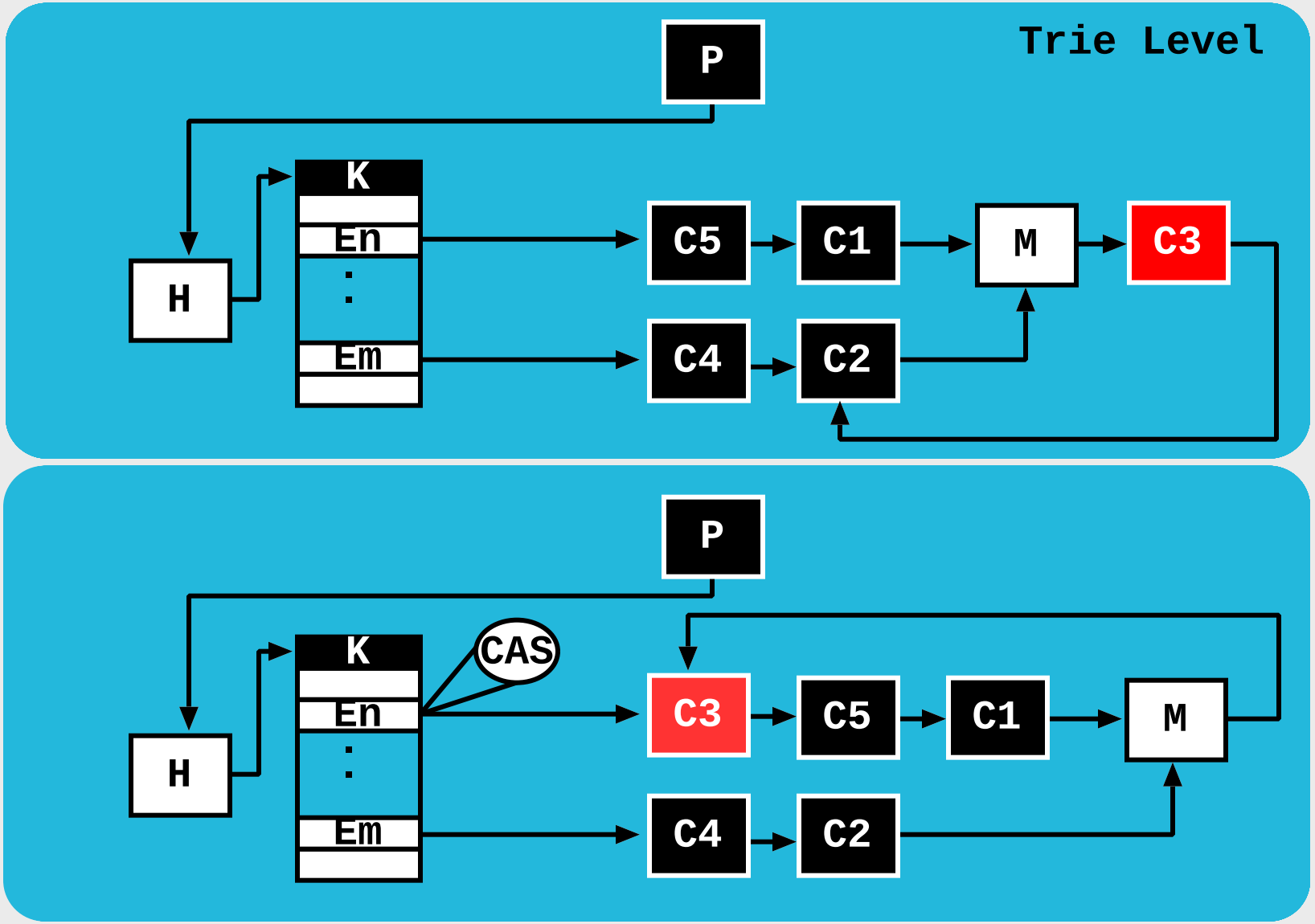
Our Approach - The First Expansion



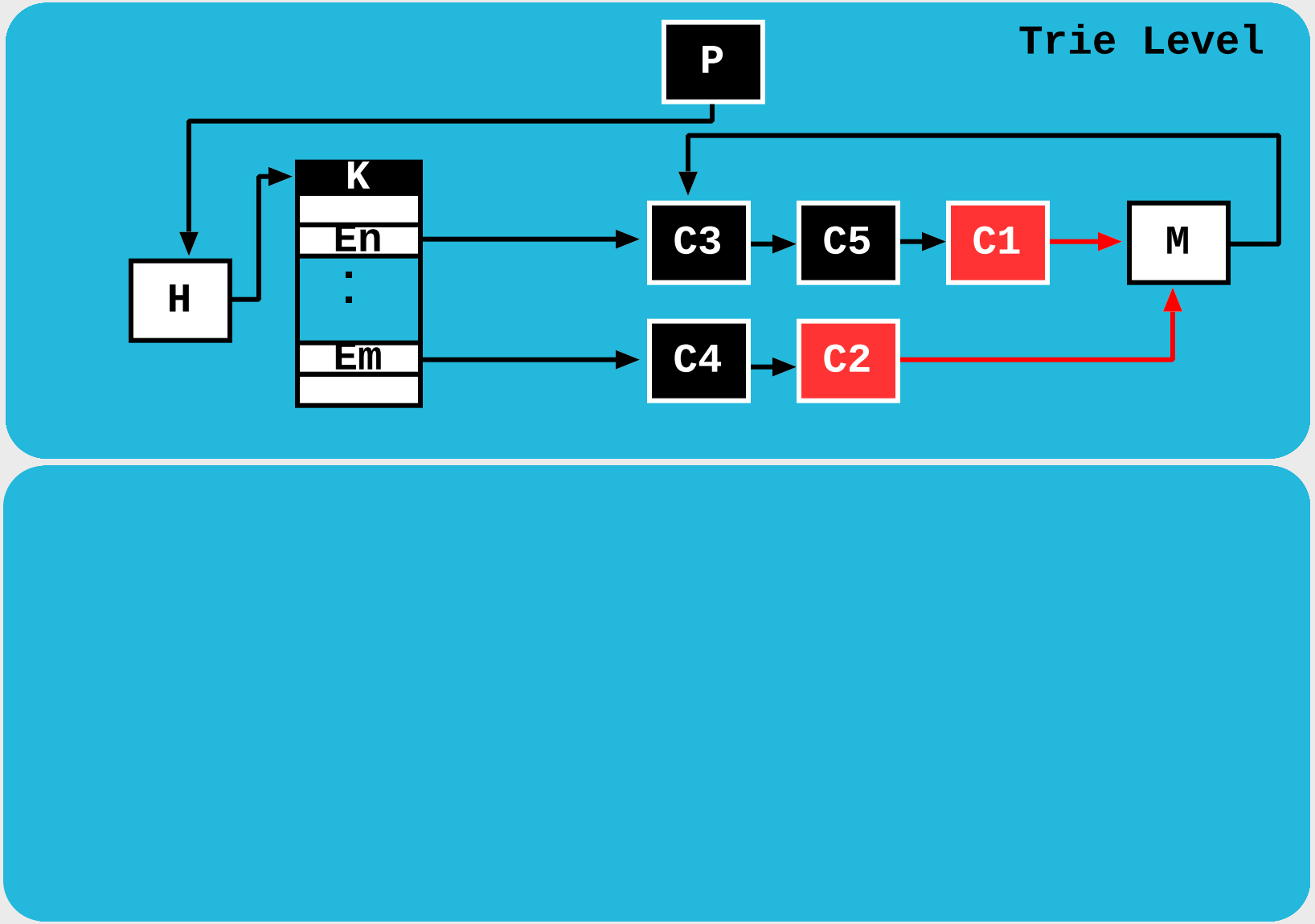
Our Approach - The First Expansion



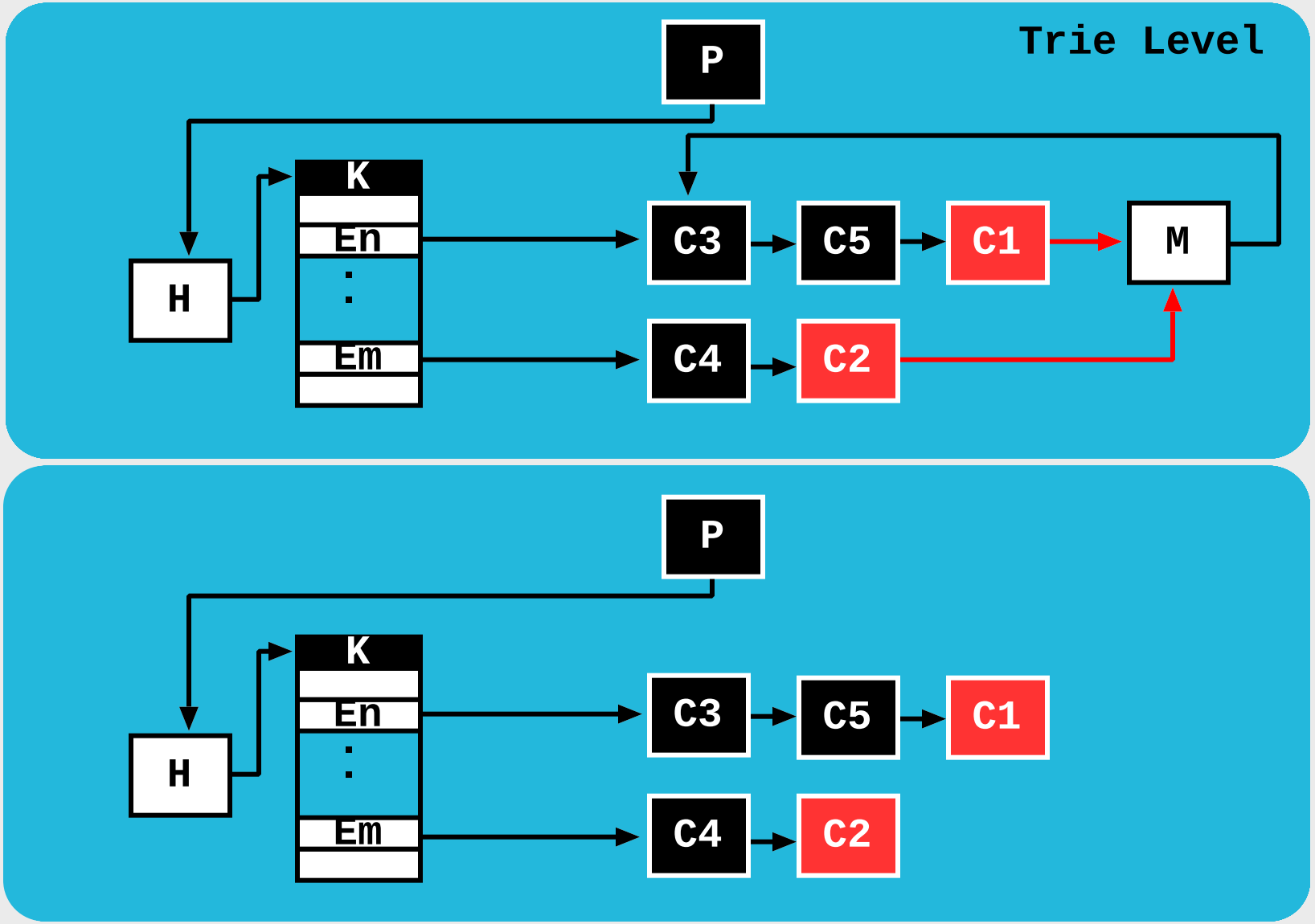
Our Approach - The First Expansion



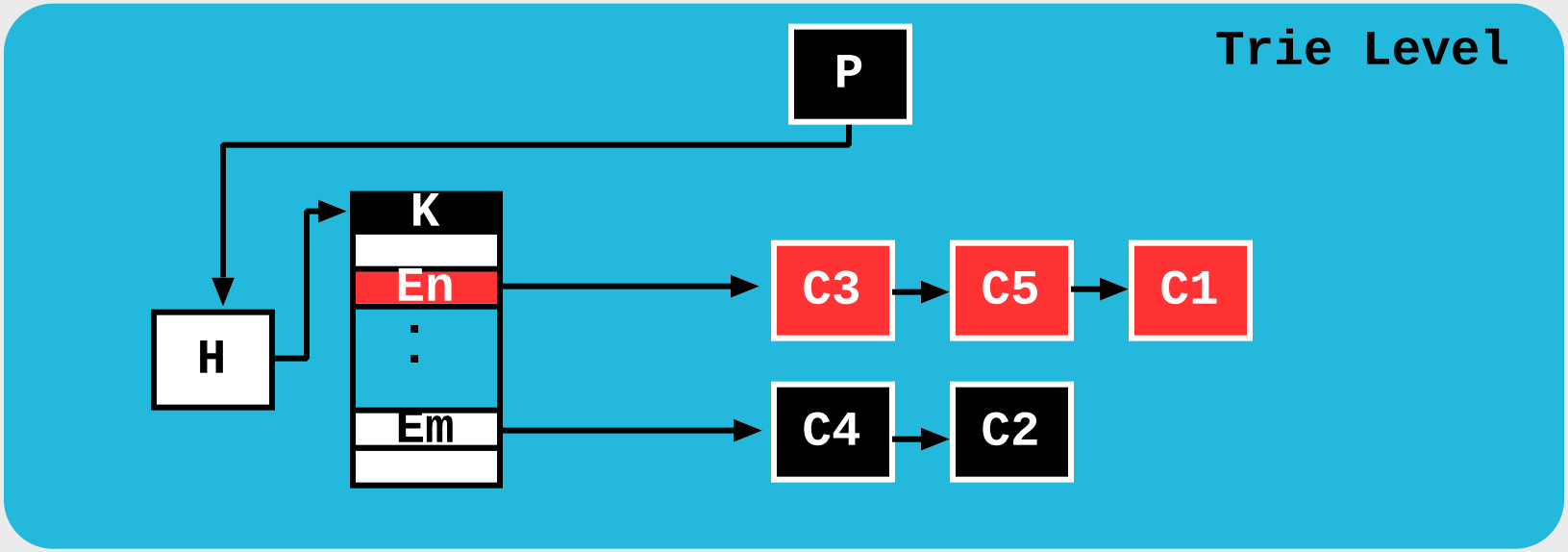
Our Approach - The First Expansion



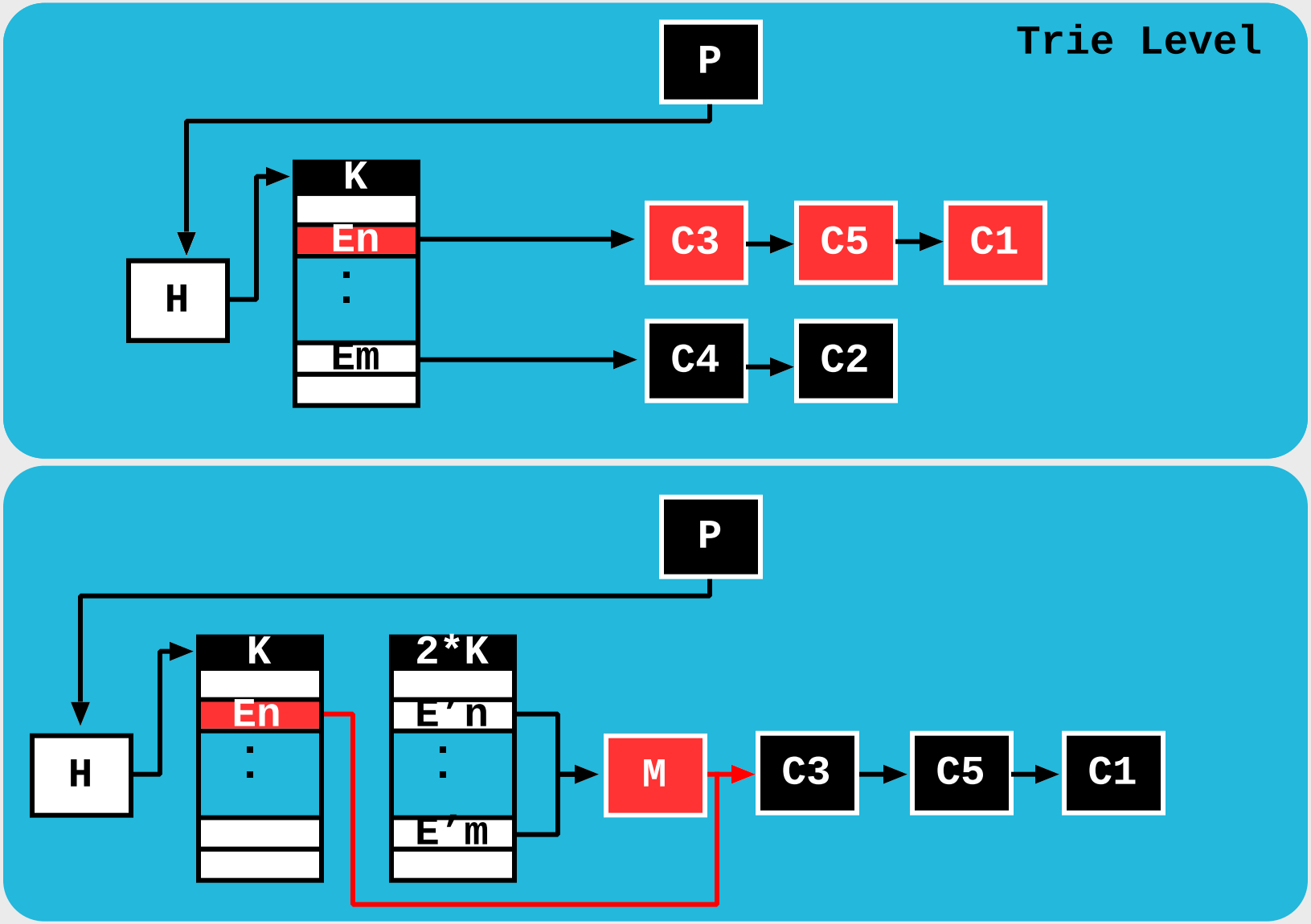
Our Approach - The First Expansion



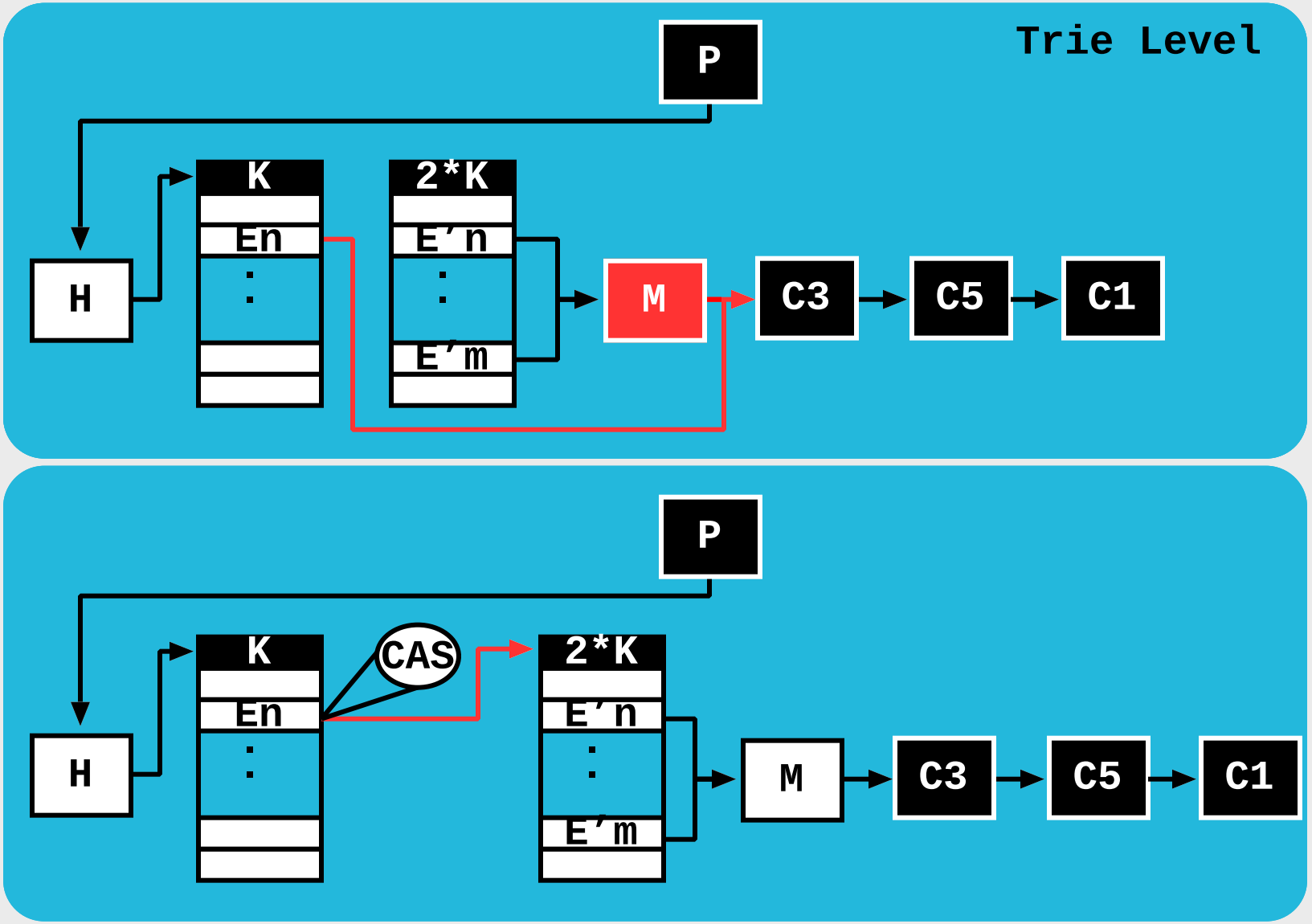
Our Approach - The Second Expansion



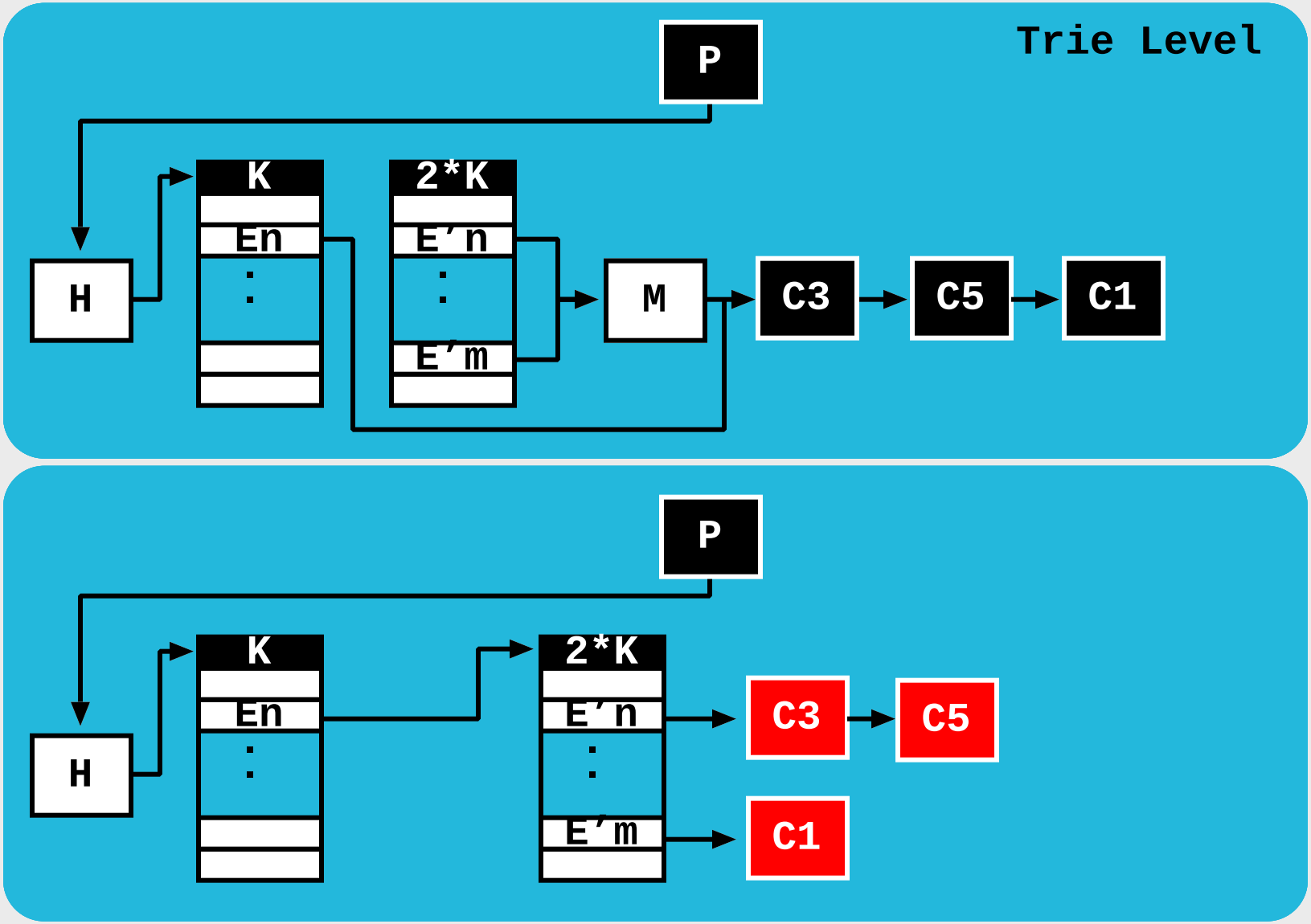
Our Approach - The Second Expansion



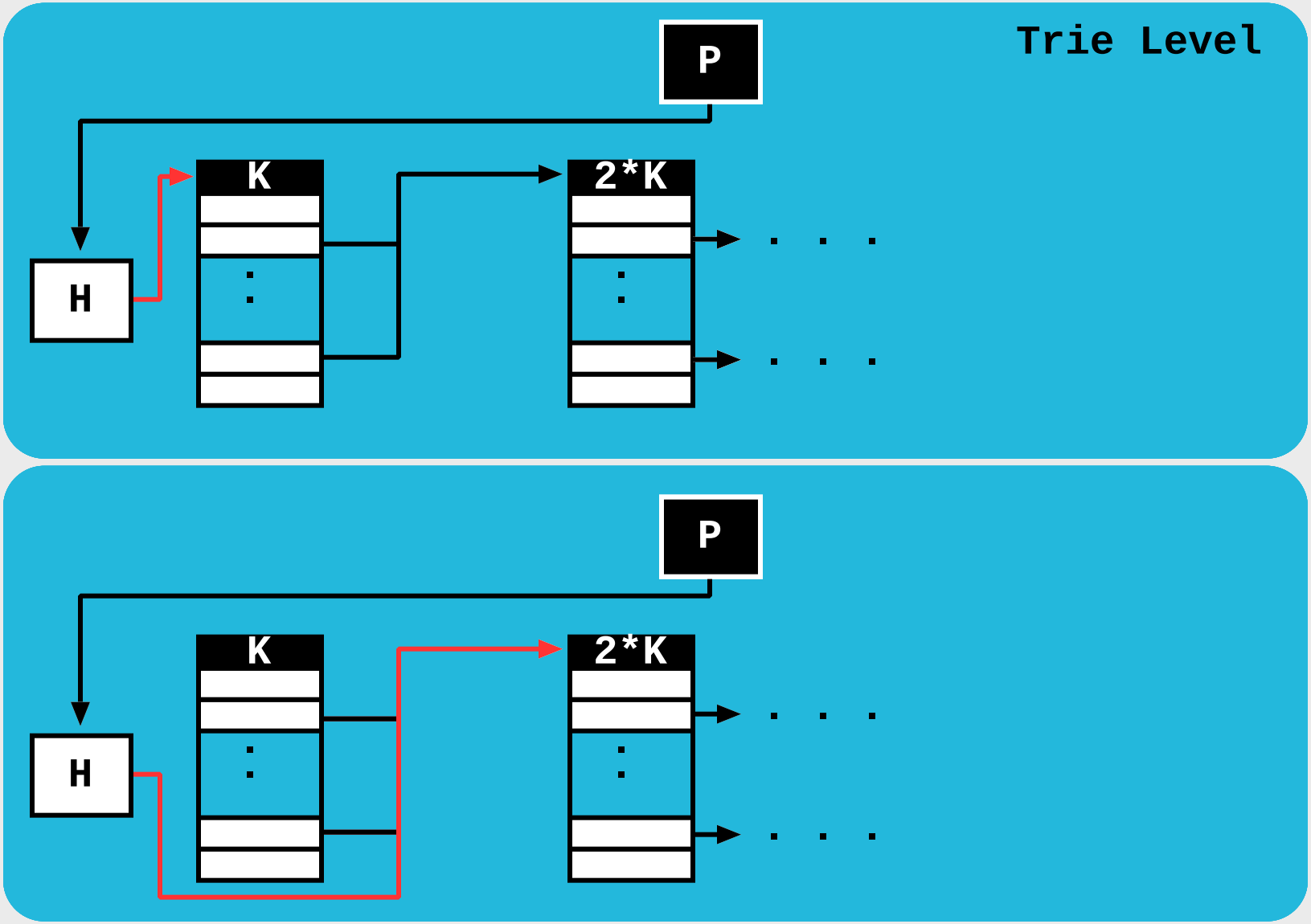
Our Approach - The Second Expansion



Our Approach - The Second Expansion



Our Approach - The Second Expansion



Our Approach - Resume

- **Avoids** the usage of **locks**:
  - **Reduces** the size of the nodes.
  - **Removes problems** associated with **locks**:
    - Contention.
    - Convoying.
    - Priority inversion.

Our Approach - Resume

- **Avoids** the usage of **locks**:
  - **Reduces** the size of the nodes.
  - **Removes problems** associated with **locks**:
    - Contention.
    - Convoying.
    - Priority inversion.

- The **create** and **expand** operations of the **hashing mechanism**:
  - **Does not lock** the **search** operation.
  - **Allow** the **concurrent insertion** of new nodes.
- **Different** nodes can be **inserted simultaneously** in **different bucket entries**.
  - Previous models **locked** all bucket entries to insert a new node.

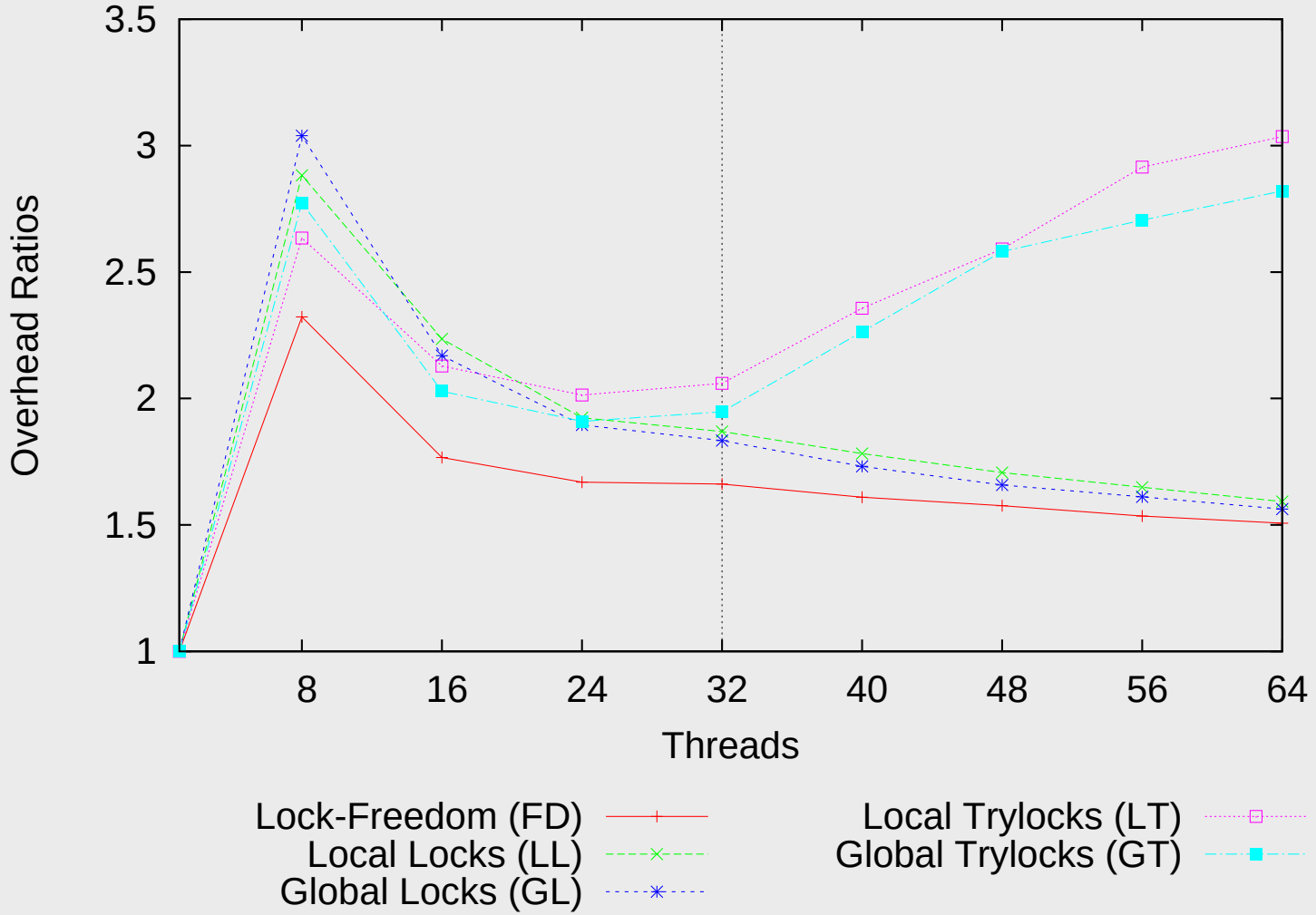
Experimental Results - Benchmark Statistics

Characteristics Of The Benchmarks: 1 Working Thread

Bench	Tabled Subgoals			Tabled Answers			
	Calls	Trie Nodes	Trie Depth	Unique	Repeated	Trie Nodes	Trie Depth
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
Large Joins							
Join2	1	6	5/5/5	2,476,099	0	2,613,660	5/5/5
Mondial	35	42	3/4/4	2,664	2,452,890	14,334	6/7/7
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

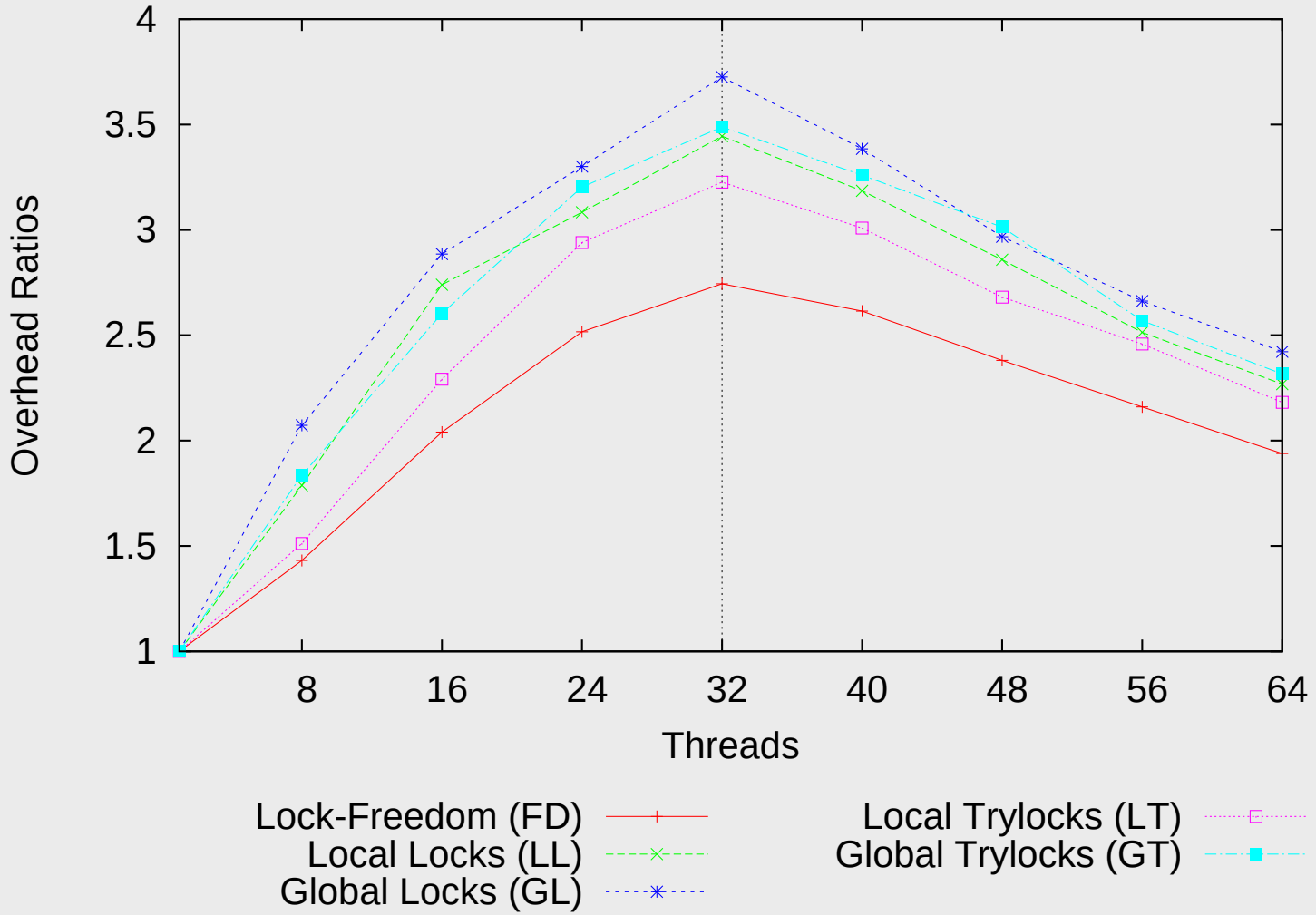
Experimental Results - User Time (FS)

- **Overhead ratios** comparing the **user time** of multiple working threads against the respective **user time** with one thread on a **32 Core AMD**.



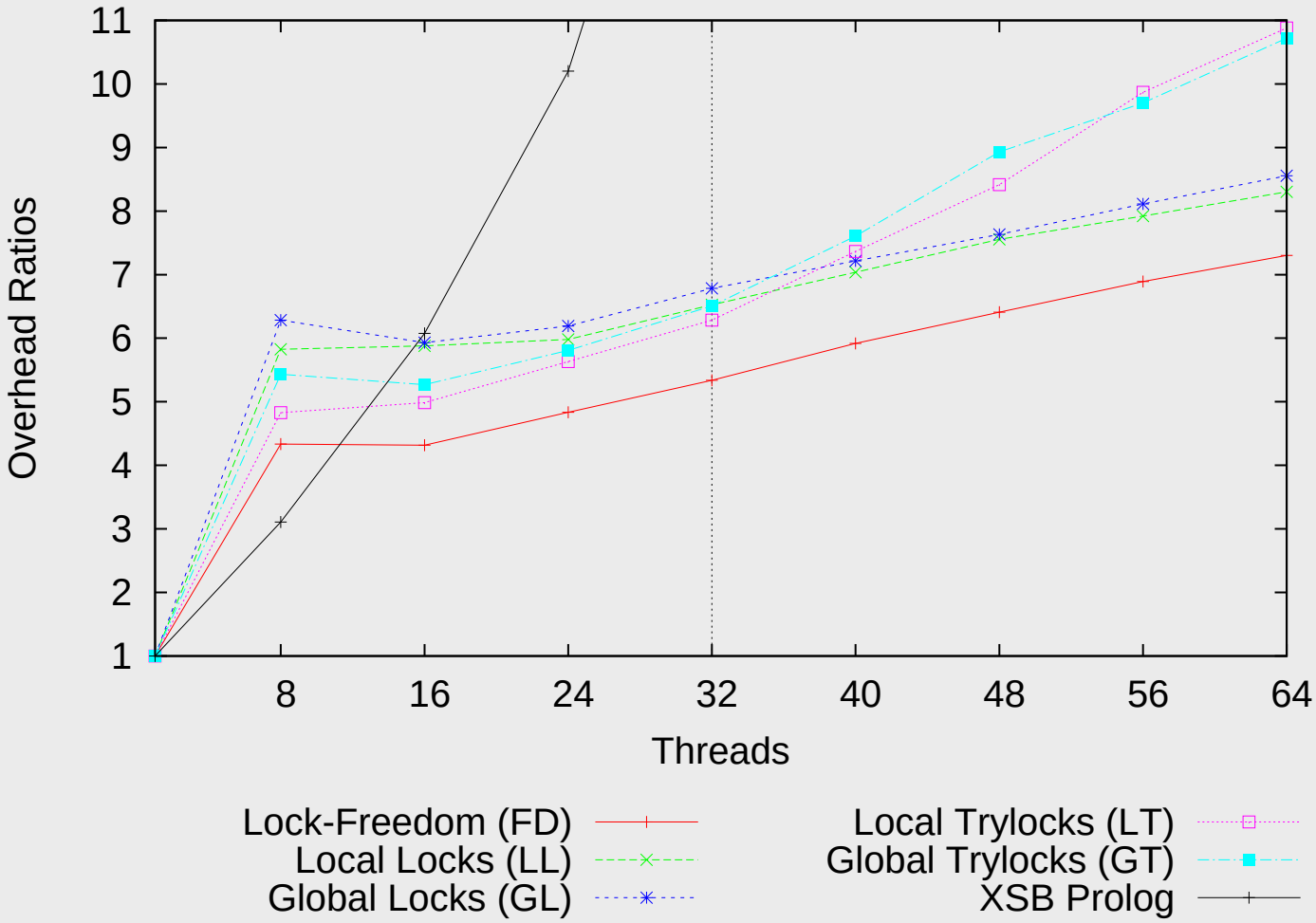
Experimental Results - System Time (FS)

- **Overhead ratios** comparing the **system time** of multiple working threads against the respective **system time** with one thread on a **32 Core AMD**.



Experimental Results - Execution Time (FS)

- **Overhead ratios** comparing the **execution time** of multiple working threads against the respective **execution time** with one thread on a **32 Core AMD**.



Conclusions and Further Work

- We have presented a **novel**, **efficient** and **lock-free** model for expandable trie data structures applied to the multithreaded tabled evaluation of logic programs:
  - Improves the **efficiency** of the concurrent search and insert operations.
  - The **paper discusses** the most relevant **implementation details** and **proves the correctness** of the model.
- Experimental results show that our approach can **effectively** reduce the **execution time** and **scale better**, when increasing the number of threads, than the original lock-based designs.

Conclusions and Further Work

- We have presented a **novel**, **efficient** and **lock-free** model for expandable trie data structures applied to the multithreaded tabled evaluation of logic programs:
  - Improves the **efficiency** of the concurrent search and insert operations.
  - The **paper discusses** the most relevant **implementation details** and **proves the correctness** of the model.
- Experimental results show that our approach can **effectively** reduce the **execution time** and **scale better**, when increasing the number of threads, than the original lock-based designs.
- Further work will include:
  - Extension to support the concurrent deletion of trie nodes (useful for **Mode-directed** tabling).
  - Comparison against other lock-free models.

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/*