Logic Programming (CTM 12.3-12.5)

Constraint Programming: Constraints and Computation Spaces

Carlos Varela

Rensselaer Polytechnic Institute

December 3, 2024

Generate and Test Example: Finding digit pairs that add to 10

- Using generate and test to do combinatorial search:

```
digit(D) :- between(0,9,D).

pairAdd10(D1,D2) :-
      digit(D1),
      digit(D2),
      D1 + D2 =:= 10.

allPairs(L) :-
     findall(p(D1,D2),pairAdd10(D1,D2),L).
```

Finding palindromes

- Find all four-digit palindromes that are products of two-digit numbers:

```
palindrome(S) :-
    digit(D1), digit(D2), digit(D3), digit(D4),       % generate
    S is (10*D1+D2)*(10*D3+D4),
    S >= 1000,                                        % test
    mod(div(S,1000),10) =:= mod(S,10),                % 1st = 4th
    mod(div(S,100),10) =:= mod(div(S,10),10).         % 2nd = 3rd

allPalindromes(S,L) :- findall(P,palindrome(P),S),length(S,L).
```

Propagate and Search

- The *generate and test* programming pattern can be very inefficient (e.g., Palindrome program explores 10000 possibilities).

- An alternative is to use a *propagate and search* technique.

  Propagate and search filters possibilities during the generation process, to prevent combinatorial explosion when possible.

Propagate and Search

Propagate and search approach is based on three key ideas:

- *Keep partial information*, e.g., "in any solution, X is greater than 100".
- *Use local deduction*, e.g., combining "X is less than Y" and "X is greater than 100", we can deduce "Y is greater than 101" (assuming Y is an integer.)
- *Do controlled search*. When no more deductions can be done, then search. Divide original CSP problem P into two new problems: (P ^ C) and (P ^ ¬C) and where C is a new constraint. The solution to P is the union of the two new sub-problems. Choice of C can significantly affect search space.

```
?- X #> 100.
X in 101..sup.

?- X #< Y, X #> 100.
X in 101..sup,
X#=<Y+ -1,
Y in 102..sup.

?- X #< Y, X #> 100,
(Y#<105 ; Y#>=105).
X in 101..103,
X#=<Y+ -1,
Y in 102..104 ;
X in 101..sup,
X#=<Y+ -1,
Y in 105..sup.
```

Propagate and Search Example

- Find two digits that add to 10, multiply to more than 24:

```
:- use_module(library(clpfd)).
q(D1,D2) :-
    D1 in 0..9, D2 in 0..9,     % initial constraints
    D1+D2 #= 10,     % D1 and D2 cannot be 0.
     % reduces search space from 100 to 81 possibilities
    D1*D2 #>= 24,    % D1 and D2 must be in 4..6.
     % reduces search space to 9 possibilities
    D1 #< D2.
     % D1 must be 4 or 5 and D2 must be 5 or 6.
     % reduces search space to 4 possibilities
     % It does not find unique solution D1=4 and D2=6.
```

```
?- q1(D1,D2).
D1 in 0..9,
D2 in 0..9.

?- q2(D1,D2).
D1 in 1..9,
D1+D2#=10,
D2 in 1..9.

?- q3(D1,D2).
D1 in 4..6,
D1+D2#=10,
D1*D2#=_A,
D2 in 4..6,
_A in 24..36.

?- q(D1,D2).
D1 in 4..5,
D1#=<D2+ -1,
D1+D2#=10,
D1*D2#=_A,
D2 in 5..6,
_A in 24..30.
```

Propagate and Search Example(2)

- Find a rectangle whose perimeter is 20, whose area is greater than or equal to 24, and width less than height:

```
rectangle([W,H]) :-
    W in 0..9, H in 0..9,
    W+H #= 10,
    W*H #>= 24,
    W #< H.
rectangleSolve(rect(W,H)) :-
    rectangle([W,H]),
    label([W,H]).
?- rectangleSolve(S).
S = rect(4, 6).
```

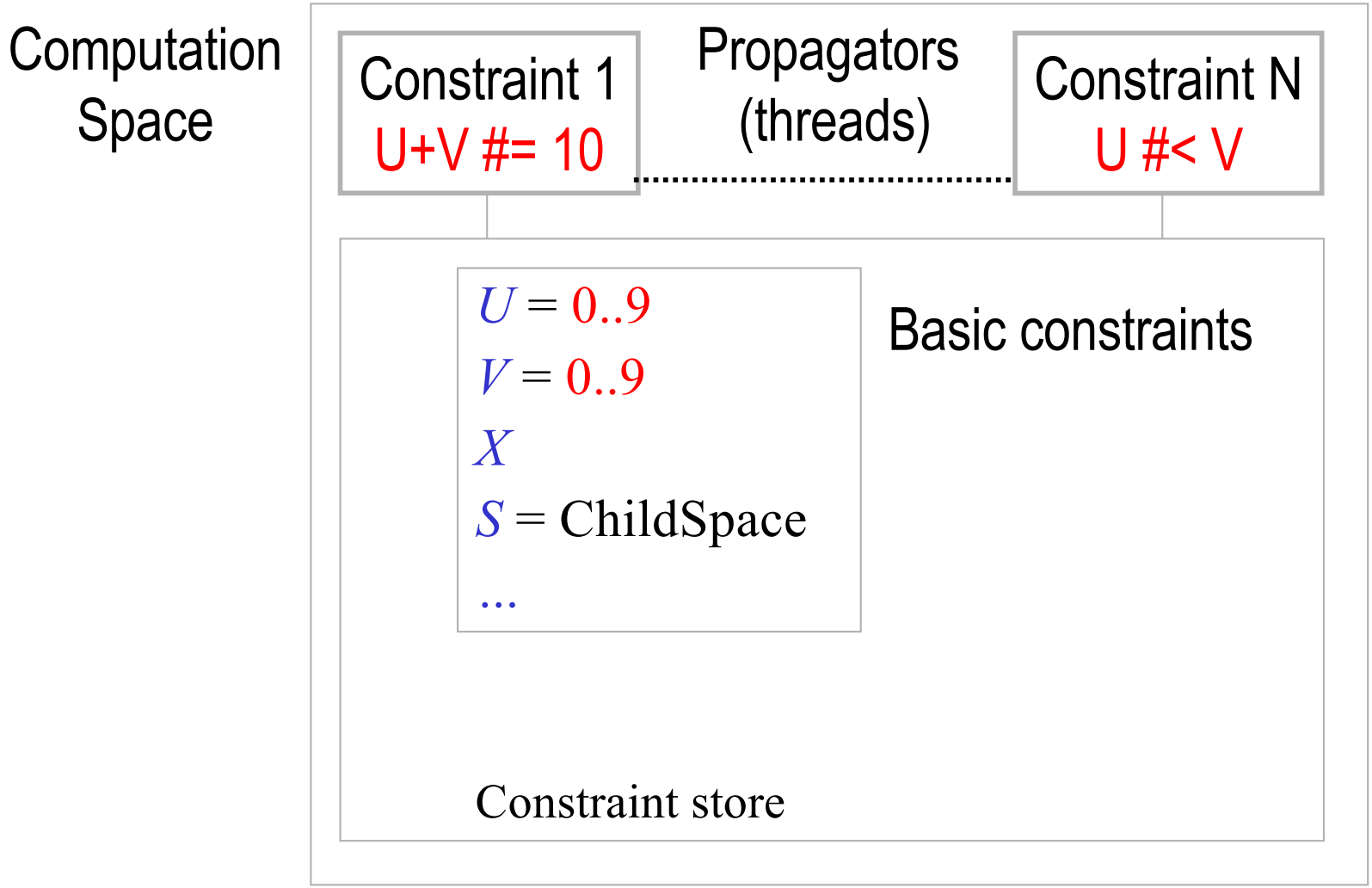
Constraint-based Computation Model

- Constraints are of two kinds:
  - *Basic constraints*: represented directly in the single-assignment store. For example, X in 0..9.
  - *Propagators*: constraints that use local deduction to propagate information across partial values by adding new basic constraints. For example, X+Y #= 10.
- A *computation space* encapsulates basic constraints and propagators. Spaces can be nested, to support distribution and search strategies.
  - Distribution strategies determine how to create new computation spaces, e.g., a subspace assuming X = 4 and another with X \= 4.
  - Search strategies determine in which order to consider subspaces, e.g., depth-first search or breadth-first search.

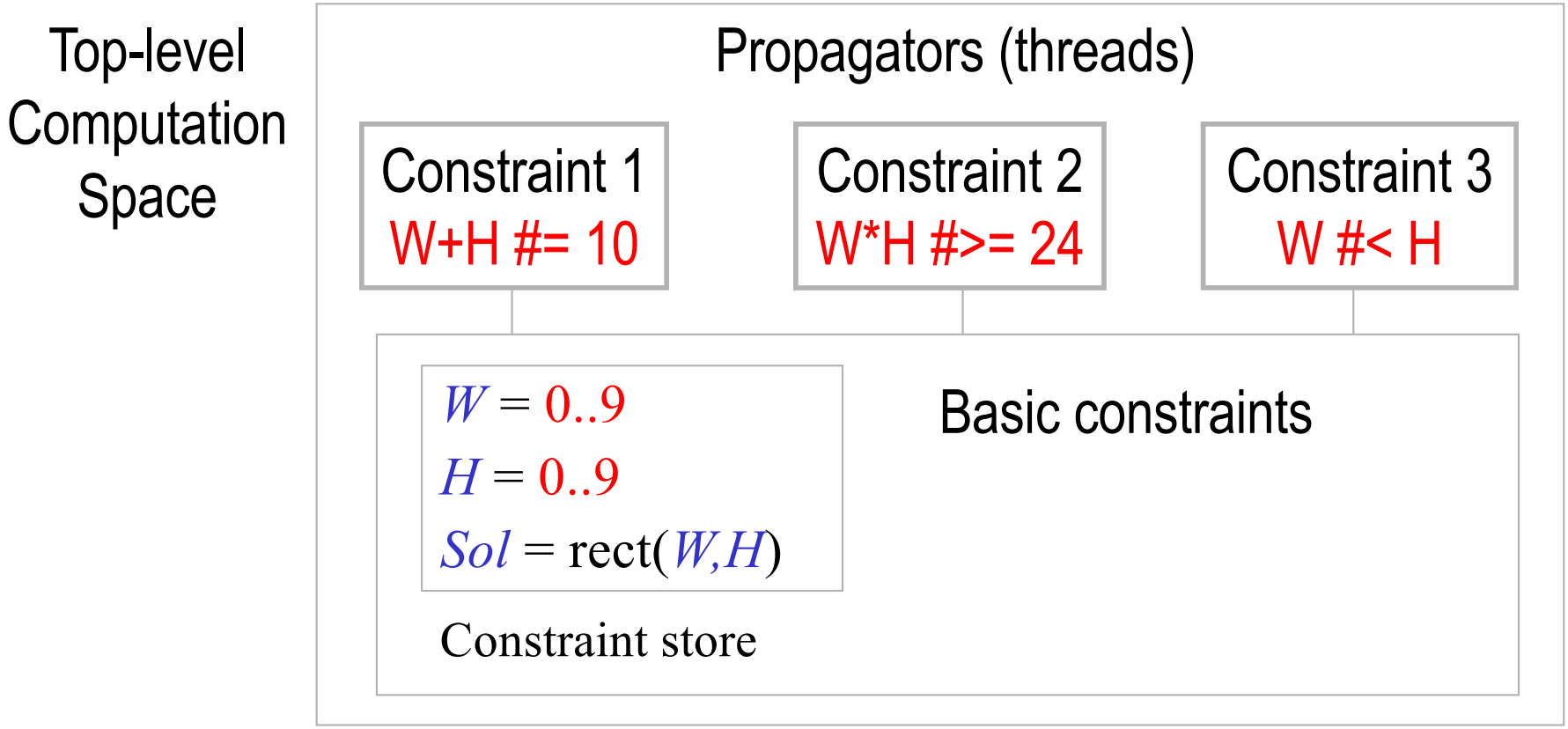
Variables and partial values

- Declarative variable:
  - is an entity that resides in a single-assignment store, that is initially unbound, and can be bound to exactly one (partial) value
  - it can be bound to several (partial) values as long as they are compatible with each other
- Partial value:
  - is a data-structure that may contain unbound variables
  - When one of the variables is bound, it is replaced by the (partial) value it is bound to
  - A complete value, or value for short is a data-structure that does not contain any unbound variables

Constraint-based computation model

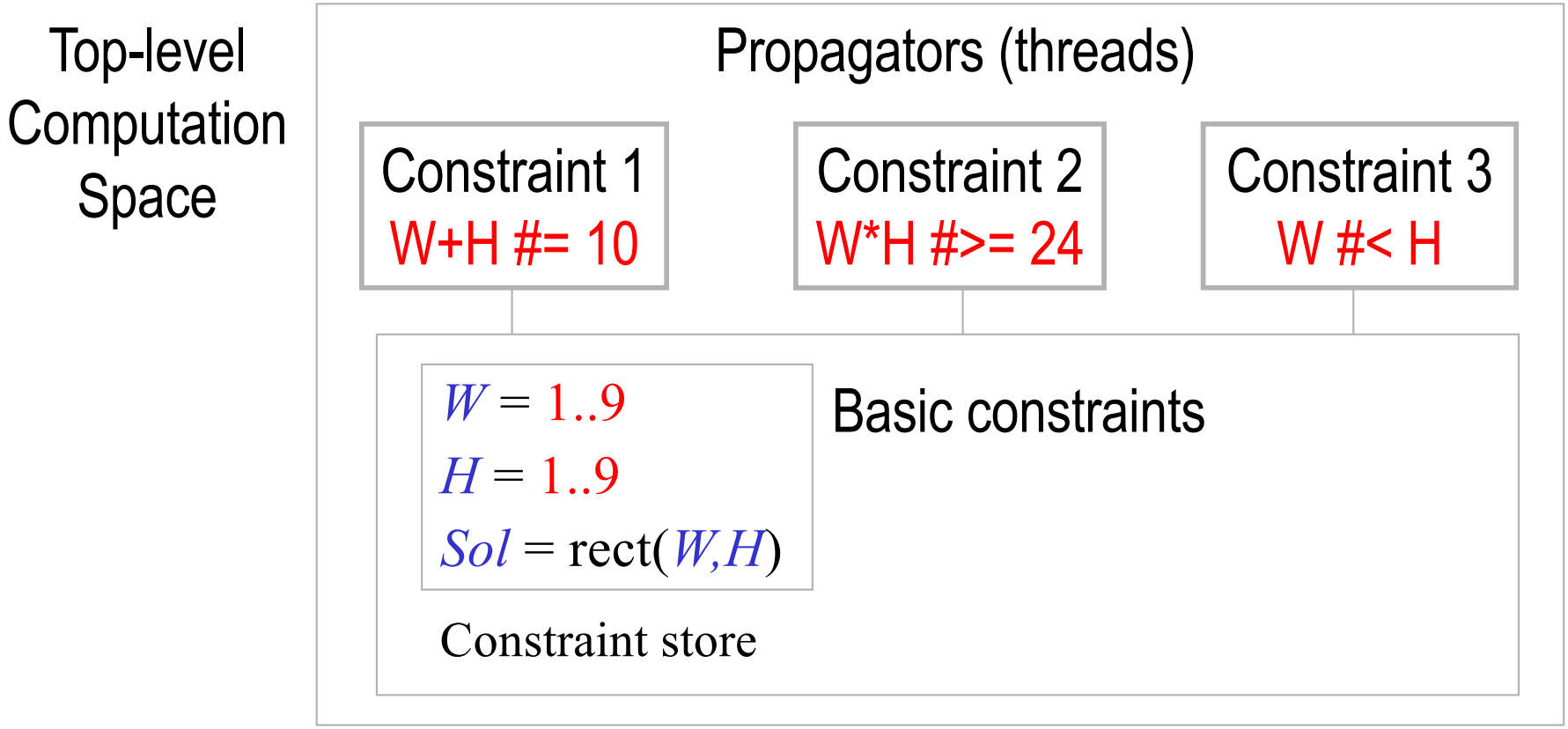


Constraint propagation: Rectangle example



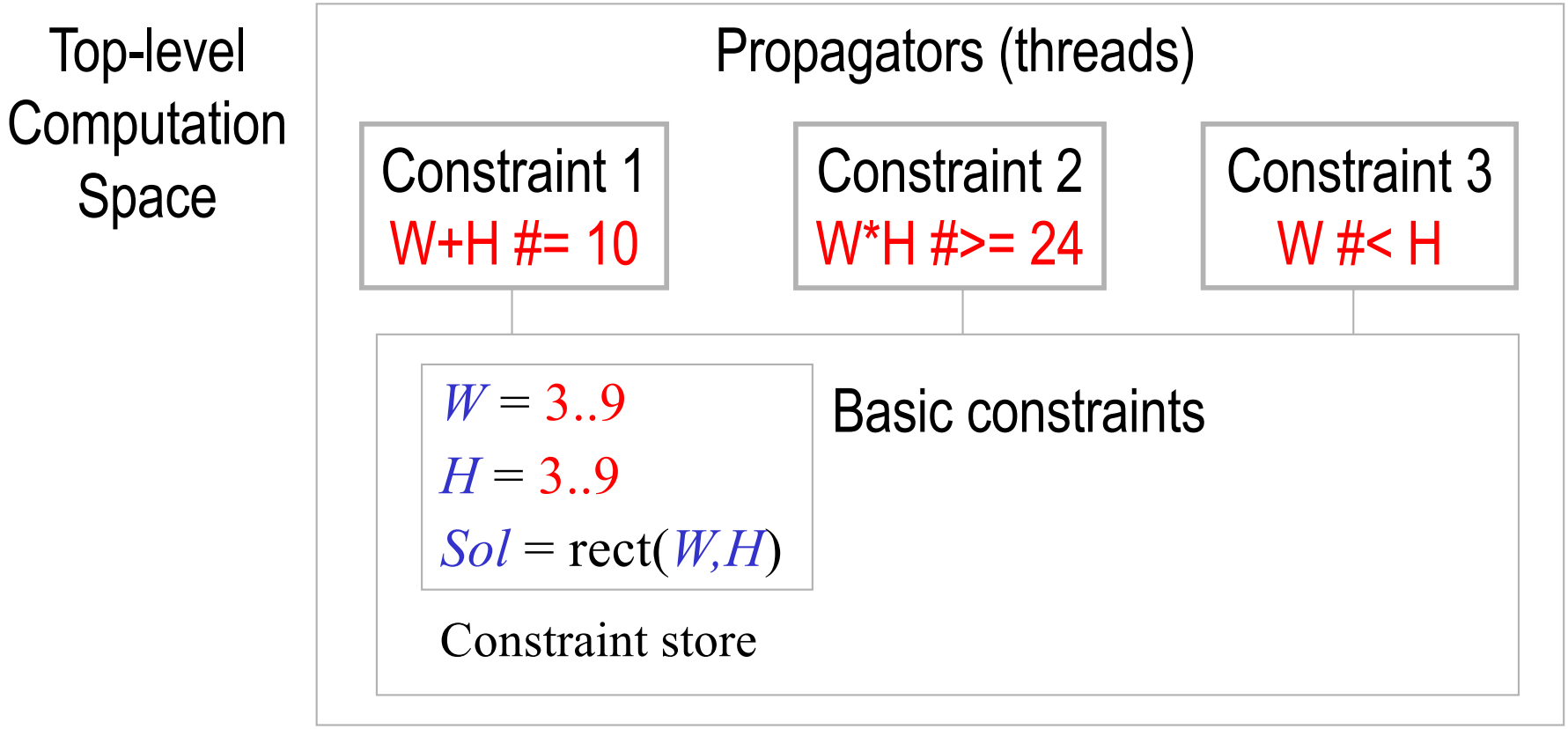
Let us consider propagator 1: W+H #= 10 → W cannot be 0; H cannot be 0.

Constraint propagation: Rectangle example



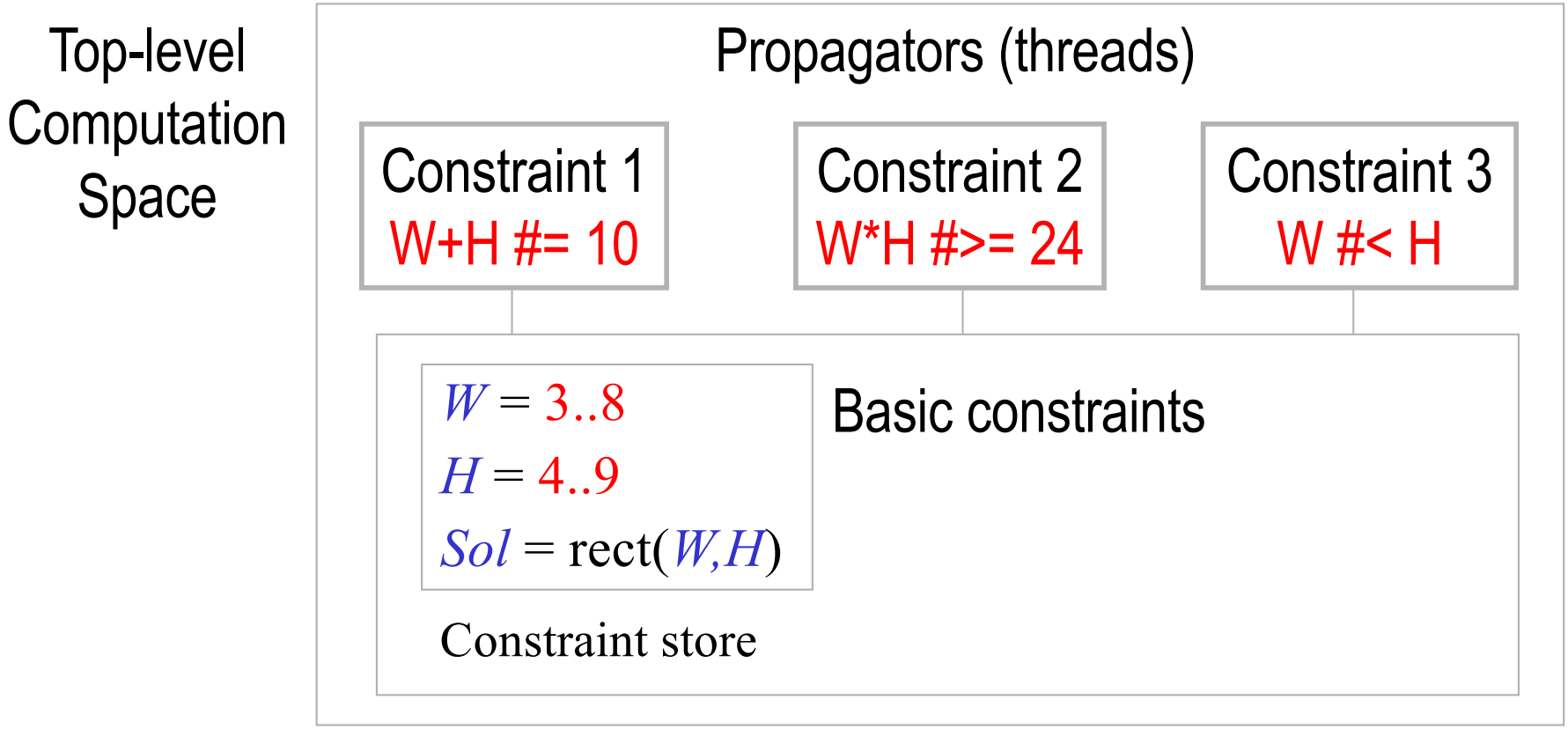
Let us consider propagator 2: W*H #>= 24 → W or H cannot be 1 or 2.

Constraint propagation: Rectangle example



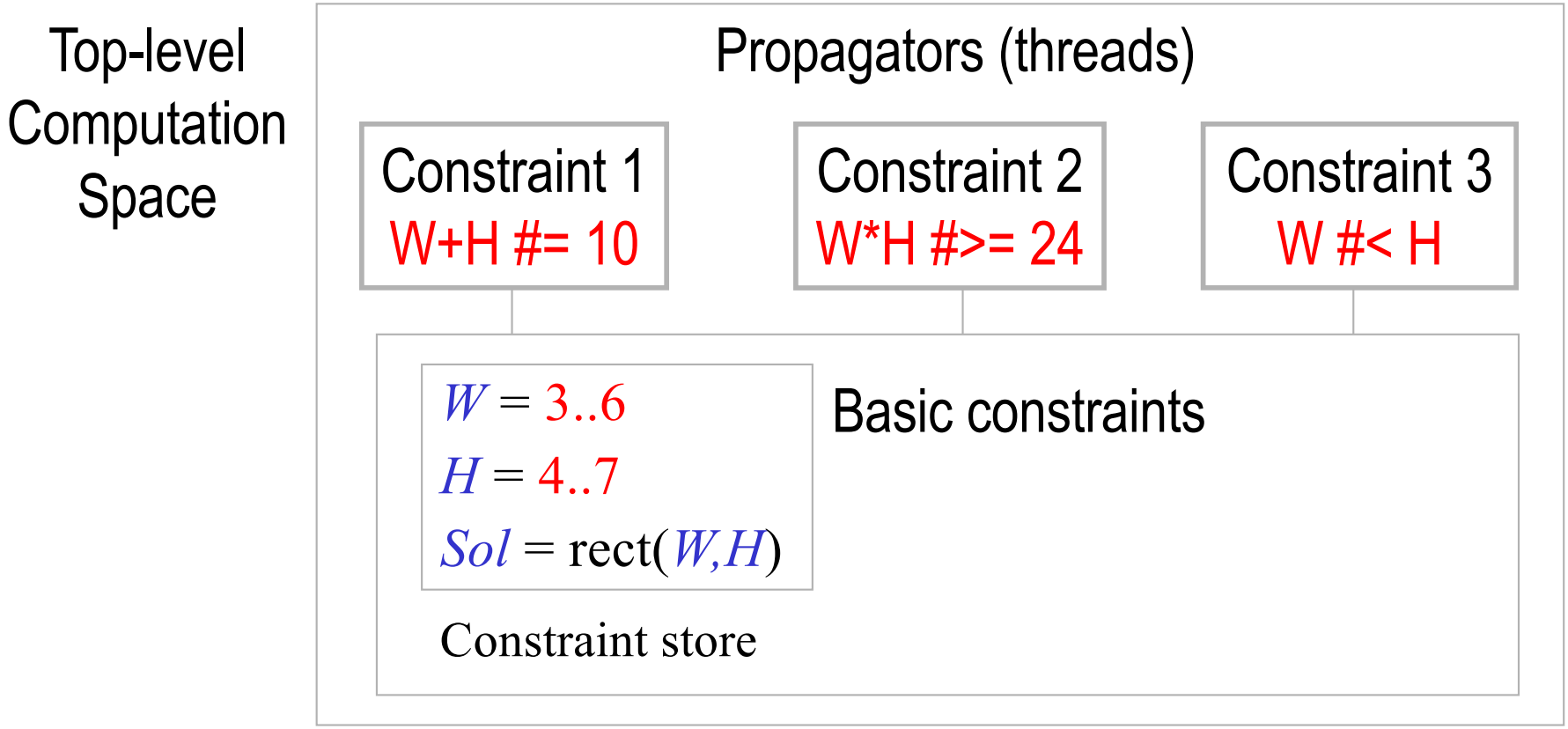
Let us consider propagator 3: W #< H → W cannot be 9, H cannot be 3.

Constraint propagation: Rectangle example



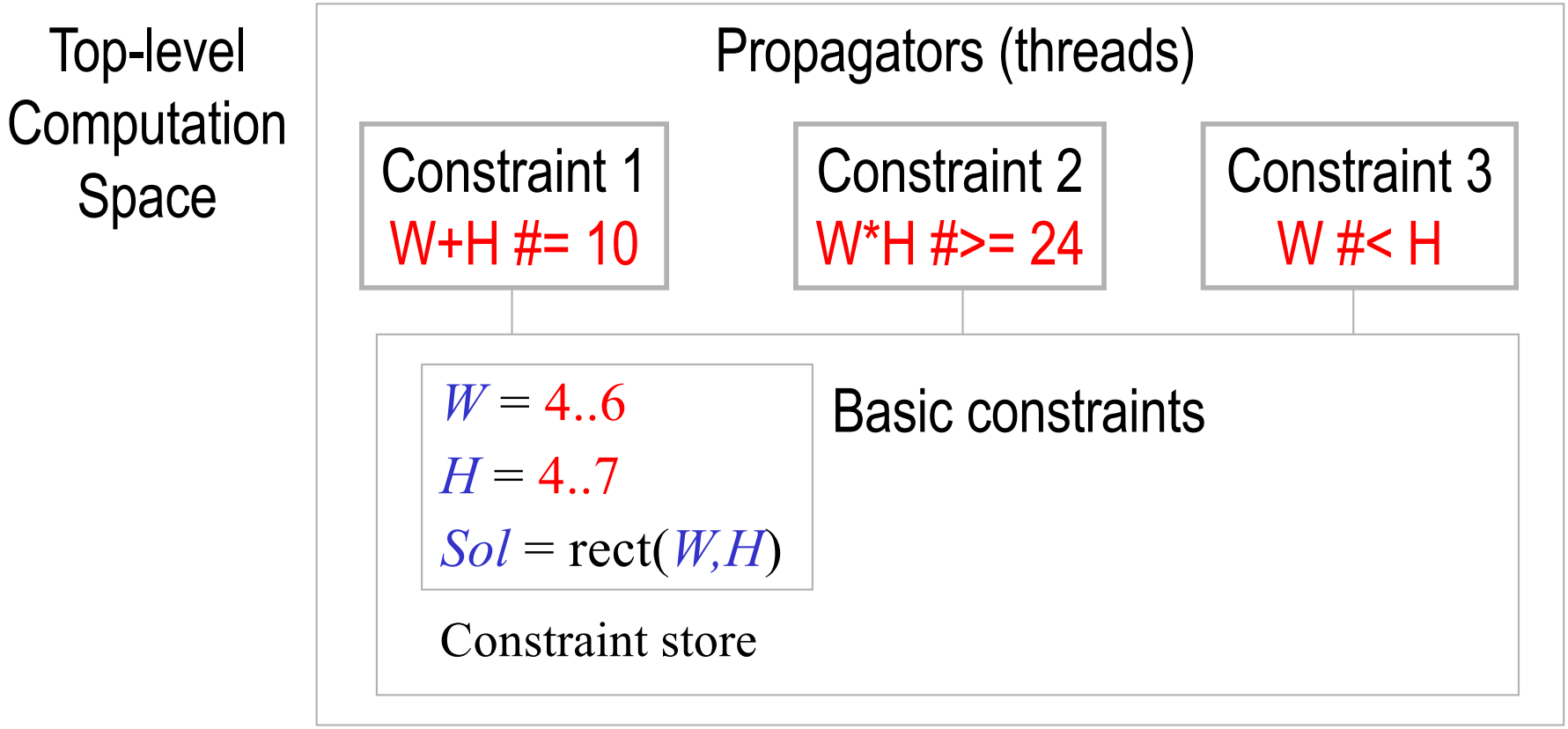
Let us consider propagator 1 **again**: W + H #= 10 →
W >= 3 implies H cannot be 8 or 9,
H >= 4 implies W cannot be 7 or 8.

Constraint propagation: Rectangle example



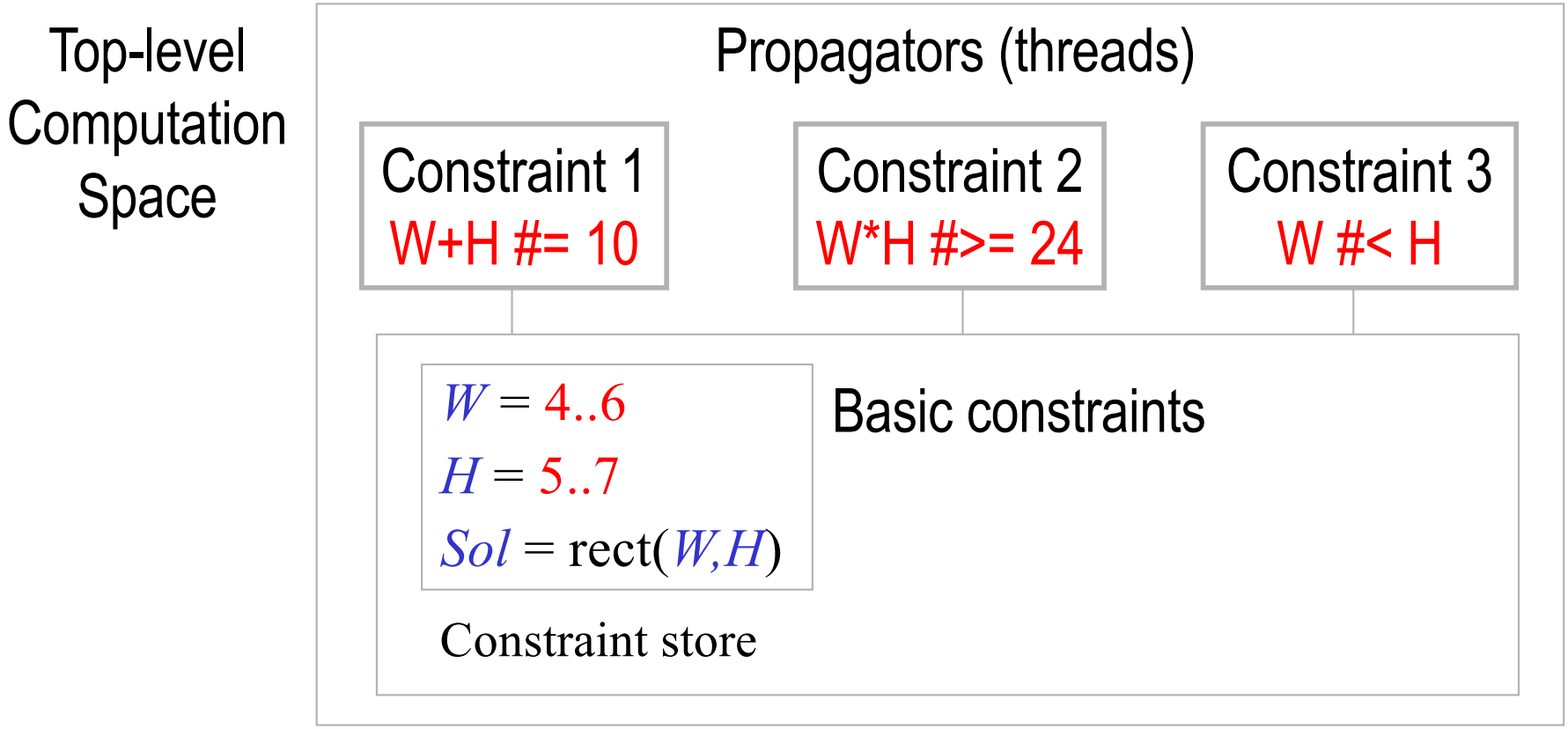
Let us consider propagator 2 **again**: W * H #>= 24 → H <= 7 implies W cannot be 3.

Constraint propagation: Rectangle example



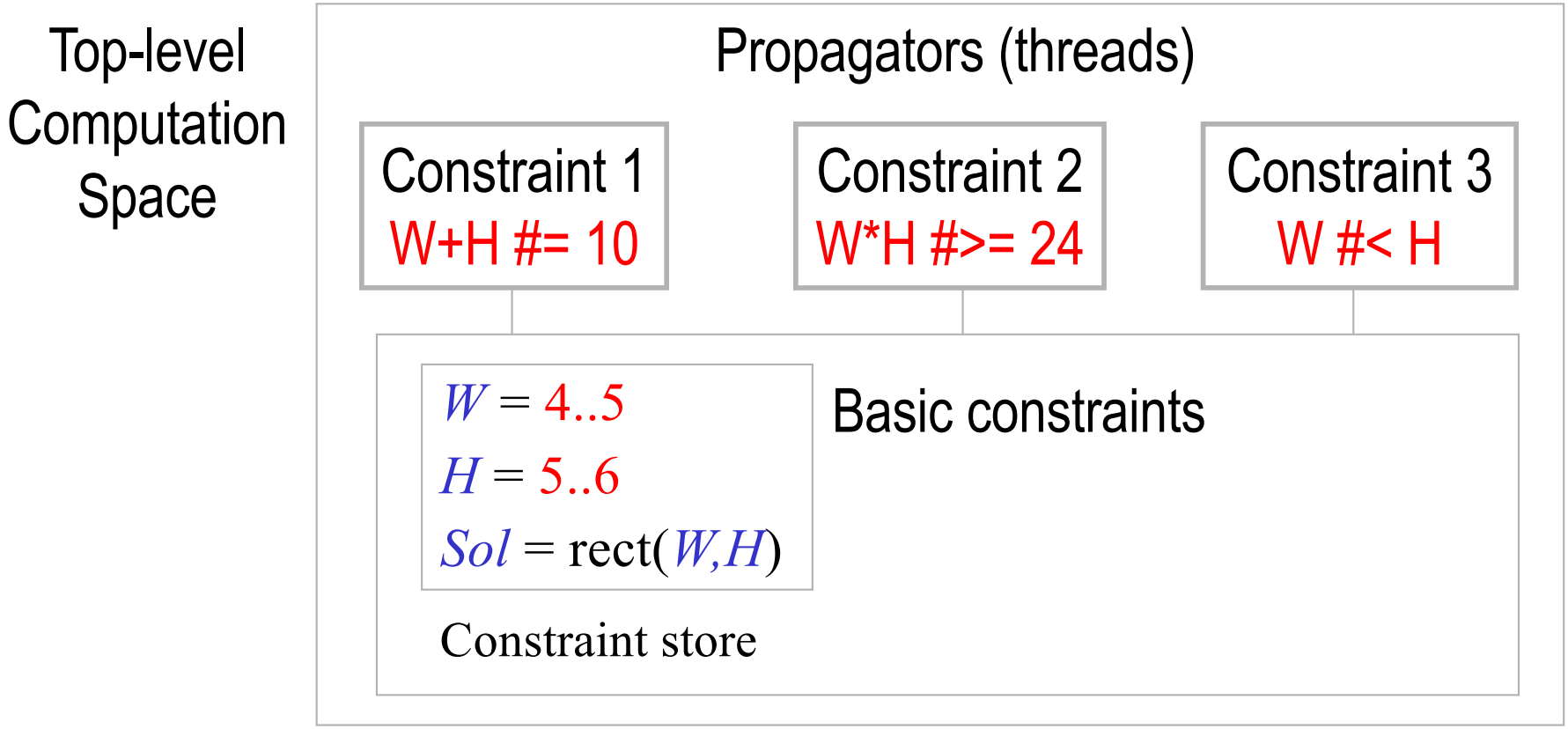
Let us consider propagator 3 **again**: W #< H → H cannot be 4.

Constraint propagation: Rectangle example



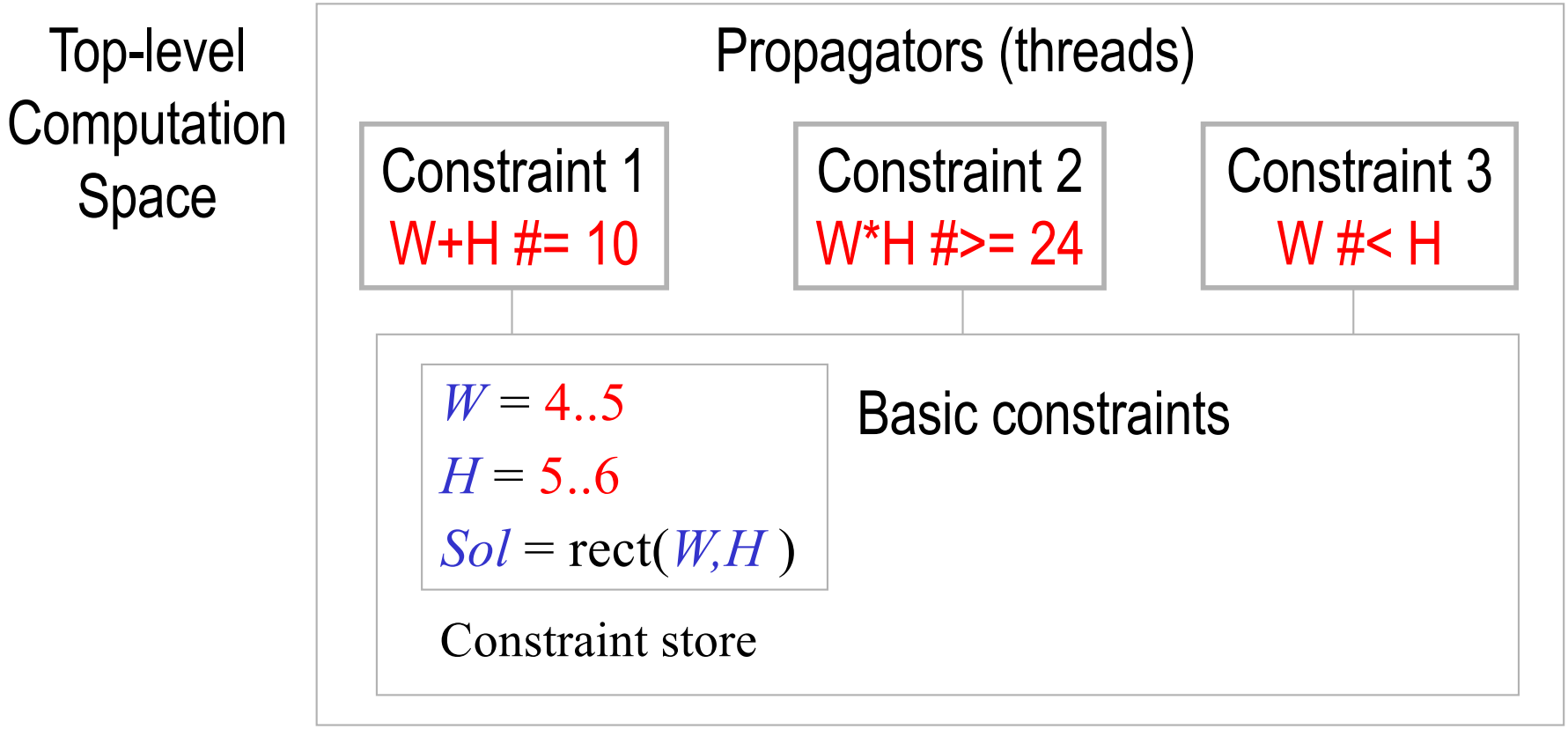
Let us consider propagator 1 **once more**: W + H #= 10 → H cannot be 7
W cannot be 6.

Constraint propagation: Rectangle example



Let us consider propagator 3 **once more**: W #< H → no new ranges.

Constraint propagation: Rectangle example



We have reached a **stable** computation space state: no single propagator can add more information to the constraint store.

Search

Once we reach a stable computation space (no local deductions can be made by individual propagators), we need to do search to make progress.

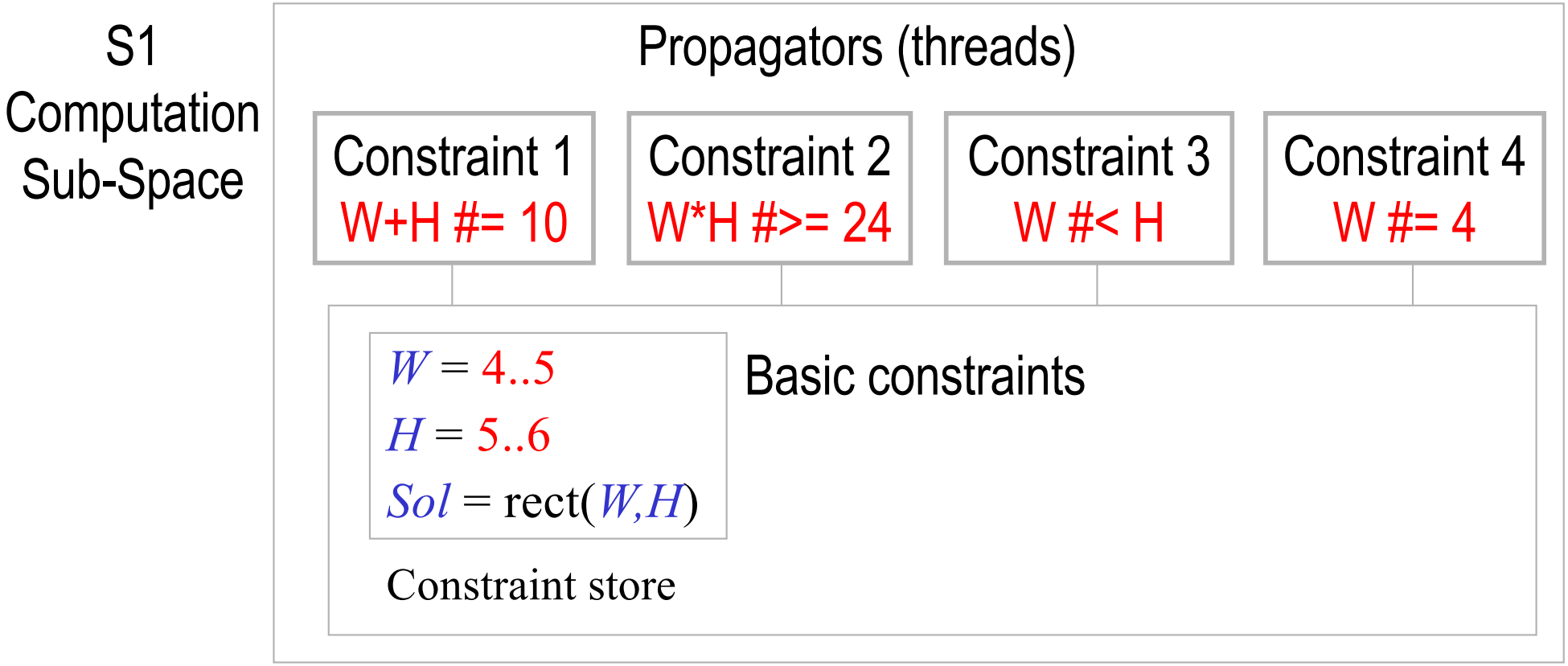
Divide original problem P into two new problems: (P ^ C) and (P ^ ¬C) and where C is a new constraint. The solution to P is the union of the solutions to the two new sub-problems.

In our Rectangle example, we divide the computation space S into two new sub-spaces S1 and S2 with new (respective) constraints:

W #= 4

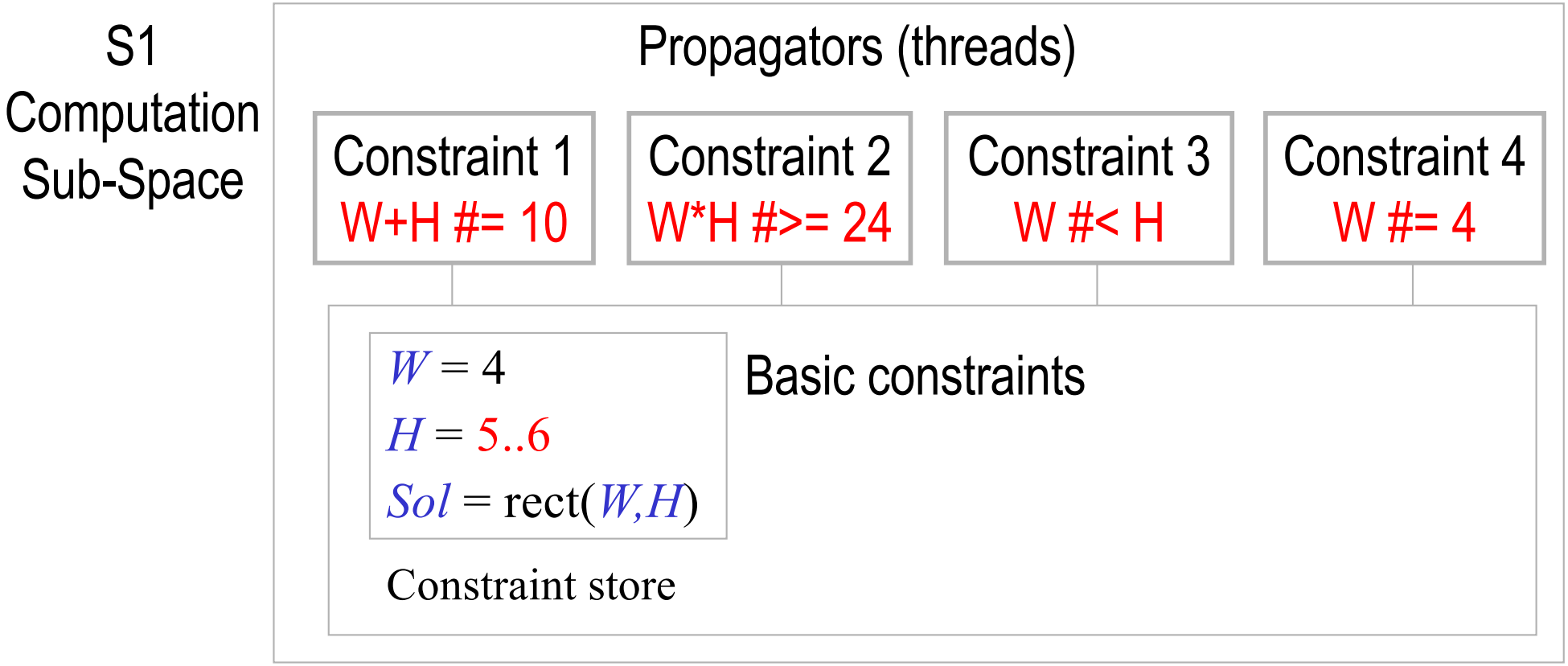
W #\= 4

Computation Space Search: Rectangle example with W#=4



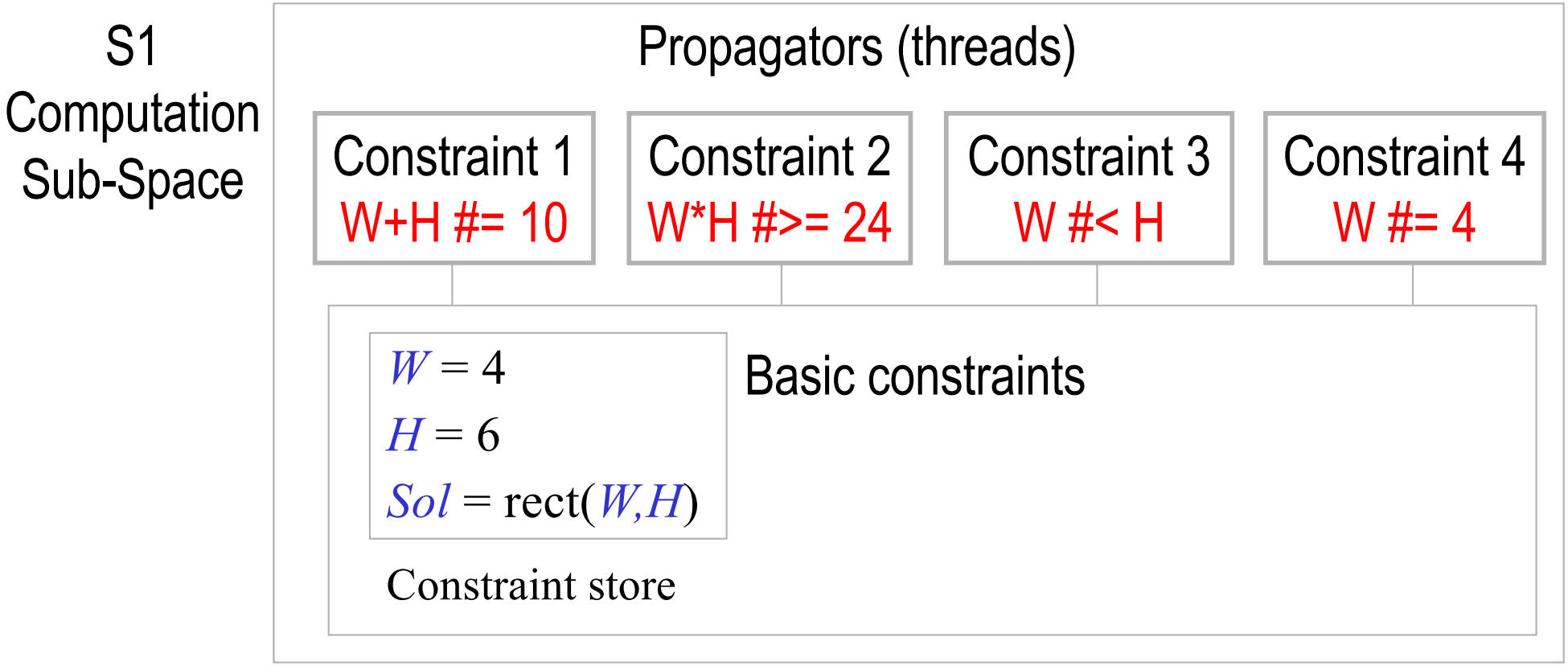
Constraint 4 implies that W = 4.

Computation Space Search: Rectangle example with W#=4



Constraint 1 or 2 imply that H = 6.

Computation Space Search: Rectangle example with W#=4



Since all the propagators are entailed by the store, their threads can terminate.

Computation Space Search: Rectangle example with W#=4

S1
Computation
Sub-Space

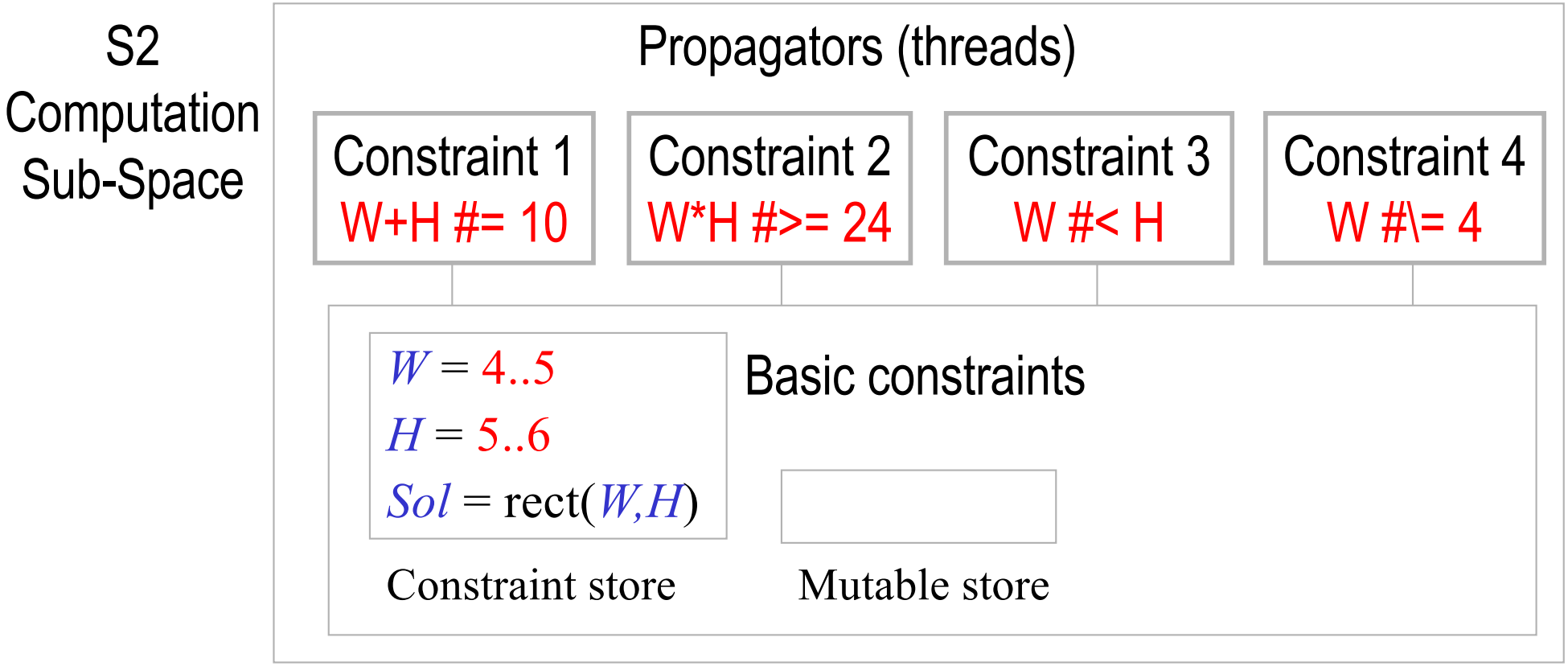
$W = 4$
$H = 6$
$Sol = \text{rect}(W,H)$

Constraint store

Basic constraints

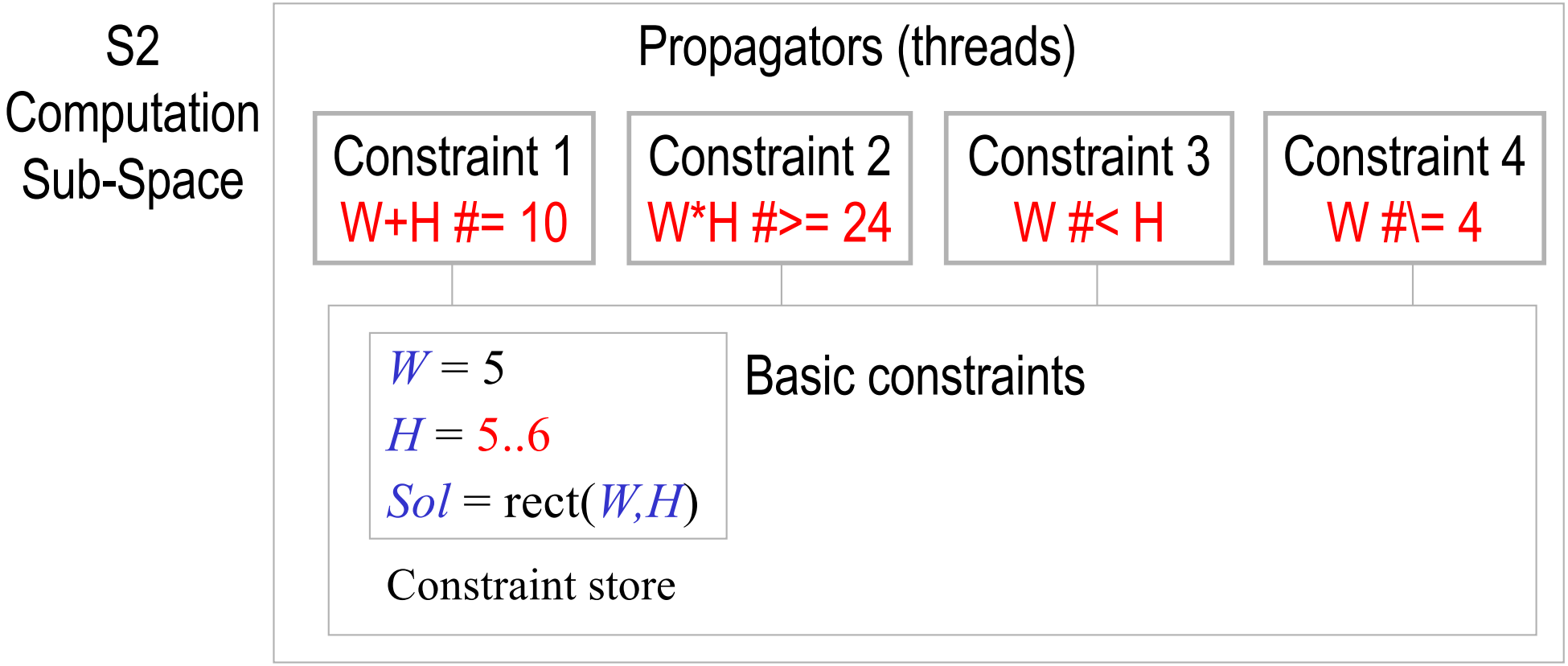
This is the final value store. A solution has been found.
The sub-space can now be **merged** with its parent computation space.

Computation Space Search: Rectangle example with W#\=4



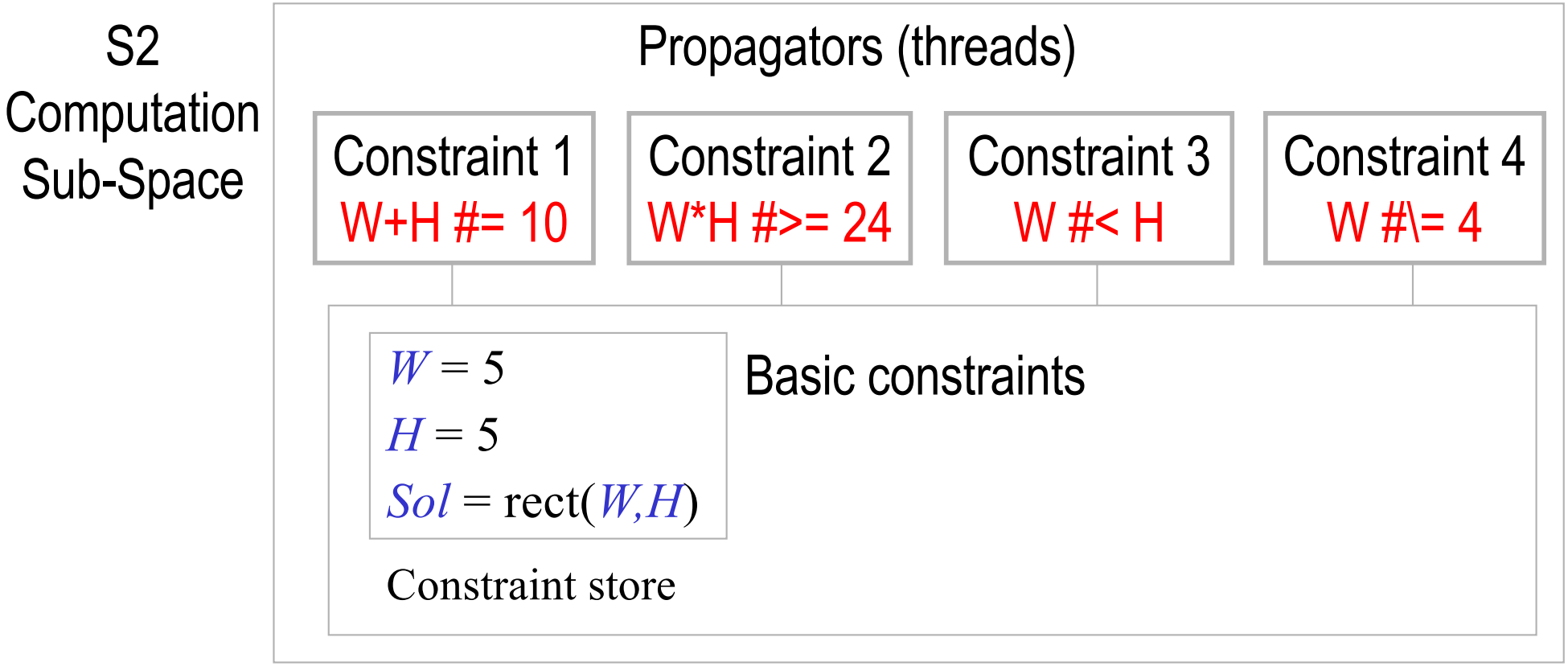
Constraint 4 implies that W = 5.

Computation Space Search: Rectangle example with w\=4



Constraint 1, W+H #= 10 → H = 5.

Computation Space Search: Rectangle example with w\=4



Constraint 3, W #< H, cannot be satisfied: computation sub-space S2 **fails**.

Finding palindromes (revisited)

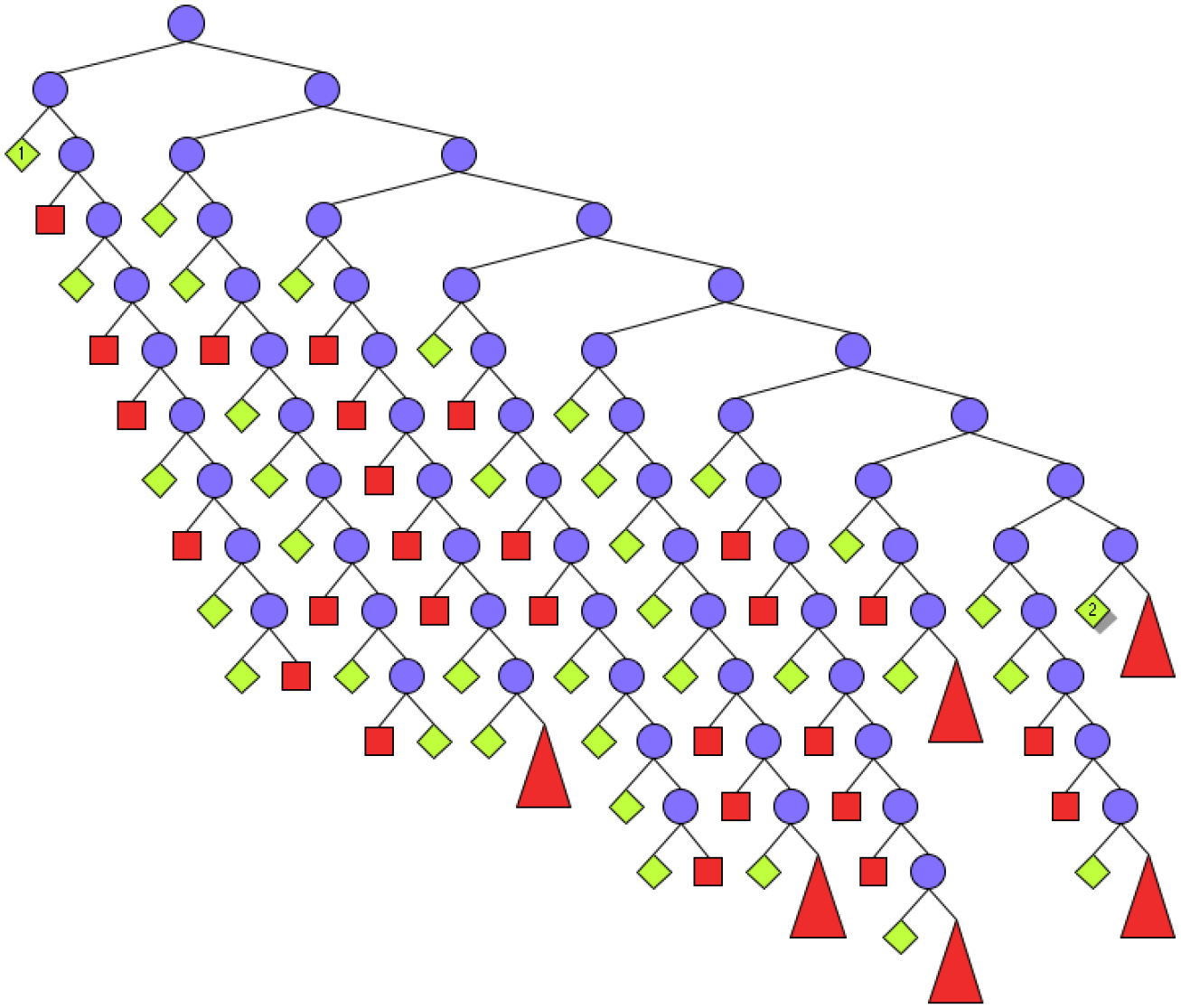
- Find all four-digit palindromes that are products of two-digit numbers:

```
palindrome(A,B,C,X,Y) :-
    A in 1000..9999, B in 0..99, C in 0..99,
    A #= B * C,
    X in 1..9, Y in 0..9,
    A #= X*1000+Y*100+Y*10+X.

palindromeSolve(A) :-
    palindrome(A,_,_,X,Y),
    labeling([ff], [X,Y]).                    % 36 solutions
```

Computation spaces for Palindrome with Oz Explorer

- At top-level, we have X=1, X\=1.
- Green diamonds correspond to **successful** sub-spaces.
- Red squares correspond to **failed** sub-spaces.



SEND + MORE = MONEY

- Find distinct digits S,E,N,D,M,O,R,Y to satisfy the following equation:

```
  S E N D
+ M O R E
-----------
M O N E Y
```

SEND + MORE = MONEY

- Find distinct digits S,E,N,D,M,O,R,Y to satisfy the SEND + MORE = MONEY equation.

```
sendMoreMoney(Sol) :-
    Sol = [S,E,N,D,M,O,R,Y],
    Sol ins 0..9,
    all_distinct(Sol),
    S #\= 0,
    M #\= 0,
                1000*S + 100*E + 10*N + D
  +             1000*M + 100*O + 10*R + E
  #= 10000*M + 1000*O + 100*N + 10*E + Y.

sendMoreMoneySolve(S) :-
    sendMoreMoney(S),
    labeling([ff], S).
```

SEND + MORE = MONEY

- Find distinct digits S,E,N,D,M,O,R,Y to satisfy the SEND + MORE = MONEY equation.

```
?- sendMoreMoney([S,E,N,D,M,O,R,Y]).
S = 9,
M = 1,
O = 0,
E in 4..7,
91*E+D+10*R#=90*N+Y,
all_distinct([9, E, N, D, 1, 0, R, Y]),
N in 5..8,
D in 2..8,
R in 2..8,
Y in 2..8.
```

Constraint propagation reduces search space from 10^8 to 4^2 * 7^3 = 5488 possibilities!

Exercises

89. Try different orders of execution for propagator threads. Do they always end up in the same constraint store? Why or why not?
90. CTM Exercise 12.6.1 (pg 774). Do it in Prolog.
91. CTM Exercise 12.6.3 (pg 775). Do it in Prolog.