Watched Literals in SAT and CP

Topics in this Series

- Why SAT & Constraints?
- SAT basics
- Constraints basics
- Encodings between SAT and Constraints
- Watched Literals in SAT and Constraints
- Learning in SAT and Constraints
- Lazy Clause Generation + SAT Modulo Theories

Legal Warning

- Watched literals may be patented
  - it's not so clear to a non-lawyer like me
- US Patent 7,418,369, August 26, 2008:
  - "Method and System for Efficient Implementation of Boolean Satisfiability"
  - Covers Chaff, Watched Literals, VSIDS

Legal Warning

- May not be an everyday problem
  - http://tinyurl.com/satpatent
  - Sharad Malik says ok for noncommercial use:
  - "The chaff software and related intellectual property have been freely available for research purposes and will continue to be available for free use by the research community for non-commercial purposes. This includes the development of other SAT solvers with this technology as well as their research use."
- But I don't know if that stands up in court
- Or what happens if you put it open source code
  - which then is used commercially

Patent in Constraints?

- As far as I know, WL patent doesn't cover Watched Literals in Constraints to be covered later
- And I know for *certain* that we have not applied for a patent for our work on it

Patents

- Software patents arouse great passions
  - I'm somewhere in the middle
  - But I'm shocked they had a patent pending for years and never told anyone
  - Please don't do this!

Watched Literals

- Key technique in ***the*** SAT propagation algorithm
  - i.e. unit propagation
- Introduced with the SAT solver Chaff
  - *Chaff: Engineering an Efficient SAT Solver* by Moskewicz, Madigan, Zhao, Zhang, Malik, DAC 2001.
  - though with precursors (of course)
    - Especially Head-Tail lists by Stickel/Zhang
- Carried over to Constraint Propagation in Minion
  - *Watched Literals for Constraint Propagation in Minion,* by Gent, Jefferson, Miguel, CP 2006.

First key idea

- *There is **no work** on backtracking*
  - Example of *not* restoring state on backtracking
    - ensuring that when we return ...
    - ... state is equivalent in vital ways but *not* identical
- This is super cute but ...
  - **oversold** as the key idea of WLs
  - in my opinion anyway

Second key idea

- *There can be **no work** in propagation*
  - If a value is deleted, we may do nothing at all
  - Even though the value is in the constraint
- This is super cute and ...
  - **undersold** as the key idea of WLs
  - in my opinion anyway
- A big difference between 0 and O(1)

Watched Literal Propagation in SAT

- Remember: Unit propagation fires when all but one literal is assigned false
- Idea: If two variables are either unassigned or assigned true, no need to do anything.
- So just find two variables which satisfy this condition.
- If can't find two, may have to propagate or

‘Watched Literals’

- Different from normal triggers (in Constraints):
  - Able to move around.
  - Not restored on backtrack.

Propagation Example

0/1	0/1	0/1	0/1
a	b	c	d

Triggers:

- a ∨ b ∨ c ∨ d

Propagation Example

0	0/1	0/1	0/1
a	b	c	d

Triggers:

- *a* assigned false.
- Update pointer.

Propagation Example

0	0/1	0/1	0/1
a	b	c	d

Triggers:

- *a* assigned false.
- Update pointer.

Propagation Example

0/1	0/1	0/1	0/1
a	b	c	d

Triggers:

- Backtrack. *a* unassigned.
- **Pointers do not move back**

Propagation Example

0/1	**1**	0/1	0/1
a	**b**	c	d

Triggers:

- If *b* is assigned true,
  pointer doesn't move.

Propagation Example

0	0/1	0/1	**0**
a	b	c	**d**

Triggers:

- If other variables assigned, nothing happens!
- Can't emphasise enough ....

Propagation Example

0	0/1	0/1	**0**
a	b	c	**d**

Triggers:

- **NOTHING HAPPENS**
- Zero work takes place

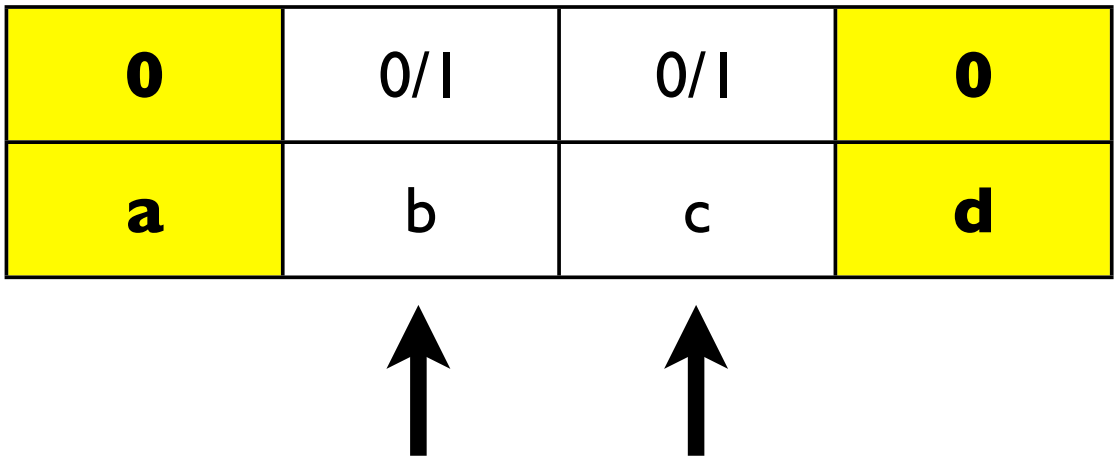
Propagation Example

0	0/1	0/1	**0**
a	b	c	**d**

Triggers:

- The unwatched literals a/d cause no work
- Not even checking there is nothing to do
  - because that would be O(1)

Propagation Example



Triggers:

- The unwatched literals a/d cause no work
- Because there is no trigger attached to them

Propagation Example

0	**0**	0/1	0
a	**b**	c	d

Triggers:

- If we cannot find something new & unassigned to watch...
-

Propagation Example

0	0	**1**	0
a	b	**c**	d

Triggers:

- We can set the remaining literal
- i.e. do unit propagation since this clause is unit

Propagation Example

0	0	**1**	0
a	b	**c**	d

Triggers:

- Leave triggers where they are!

Propagation Example

0	0/1	0/1	0
a	b	c	d

Triggers:

- Triggers in the right place to continue after backtracking.

Advantages of WL

- **ZERO** cost if a literal not watched.
- **ZERO** cost on backtrack.

Watched Literals in SAT

- Really come into their own on large clauses
  - probably not worthwhile on 3-SAT, for example
- E.g. if I have 100 variables in clause
  - I still only need to watch 2
  - and 98% of the time I will do no work
  - As if my problem was 98% smaller!
- We can handle problems with many large clauses
- Which links with explanations & learning
  - since those clauses are often big

Watched Literals in SAT

- A key technique in modern SAT solvers
- Sadly, under analysed
  - Everyone uses them
  - Everyone thinks why they work well
  - But few to no experiments showing really why

Porting to Constraints

- Nothing too deep
- Have trigger on literals instead of variables (or bounds)
  - trigger = event that causes propagator to be called
  - literal = variable/value pair, e.g. x=7
- Allow triggers to move during search
  - can lead to horrible bugs without huge caution
- Care in coming up with correct sets of watches
  - for each constraint we want to use

Element Example

M_1	1	2	3	4
M_2	1	2	3	4
M_3	1	2	3	4

Index	1	2	3

Result	1	2	3	4

- What do we need to watch?
- Enough to *support* every value

Element Example

M_1	1	2	3	4
M_2	1	2	3	4
M_3	1	2	3	4

Index	1	2	3

Result	1	2	3	4

- What do we need to watch?
- Enough to *support* every value
- Start with Index
- **`M[1] = 1`**

Element Example

M_1	1	2	3	4
M_2	1	2	3	4
M_3	1	2	3	4

Index	1	2	3

Result	1	2	3	4

- What do we need to watch?
- Enough to *support* every value
- Start with Index
- **M[2] = 2**

Element Example

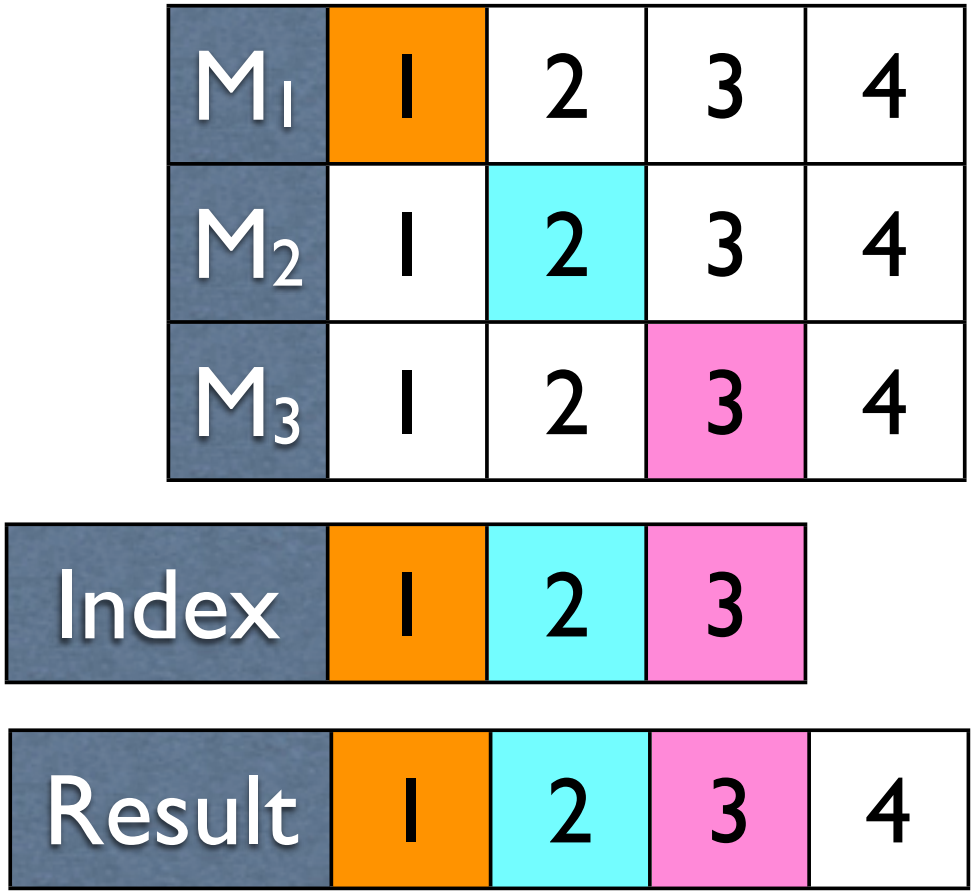
M_1	1	2	3	4
M_2	1	2	3	4
M_3	1	2	3	4

Index	1	2	3

Result	1	2	3	4

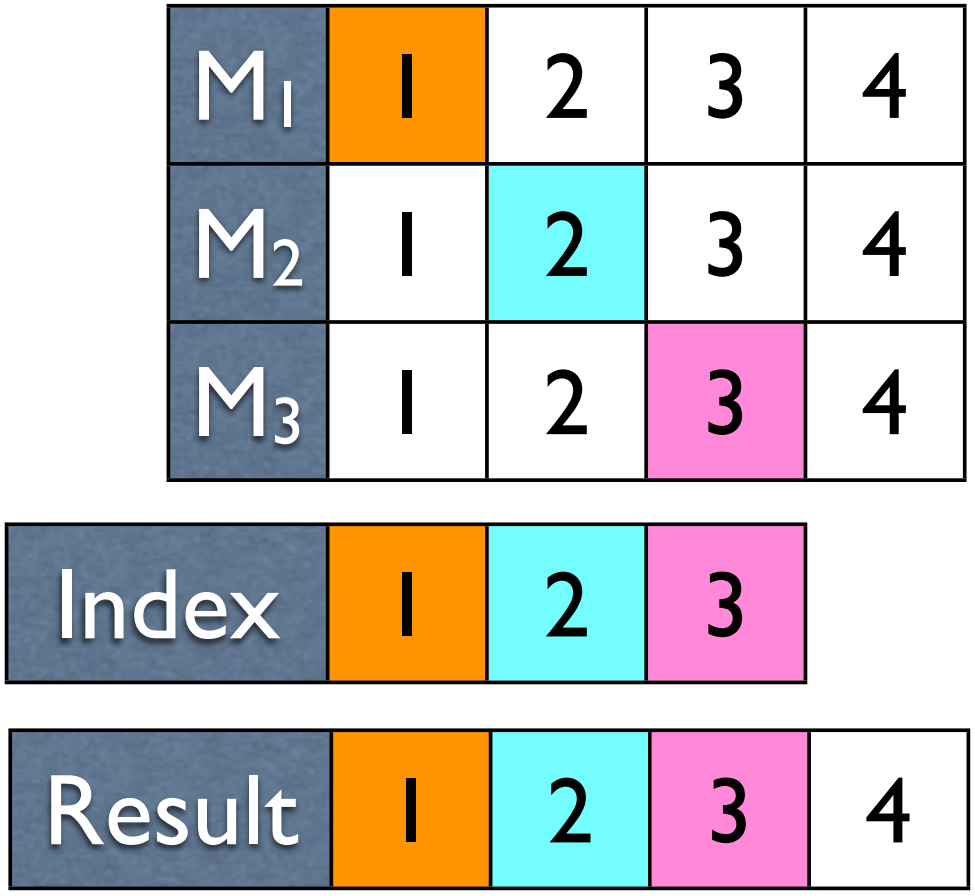
- What do we need to watch?
- Enough to *support* every value
- Start with Index
- **M[3] = 3**

Element Example



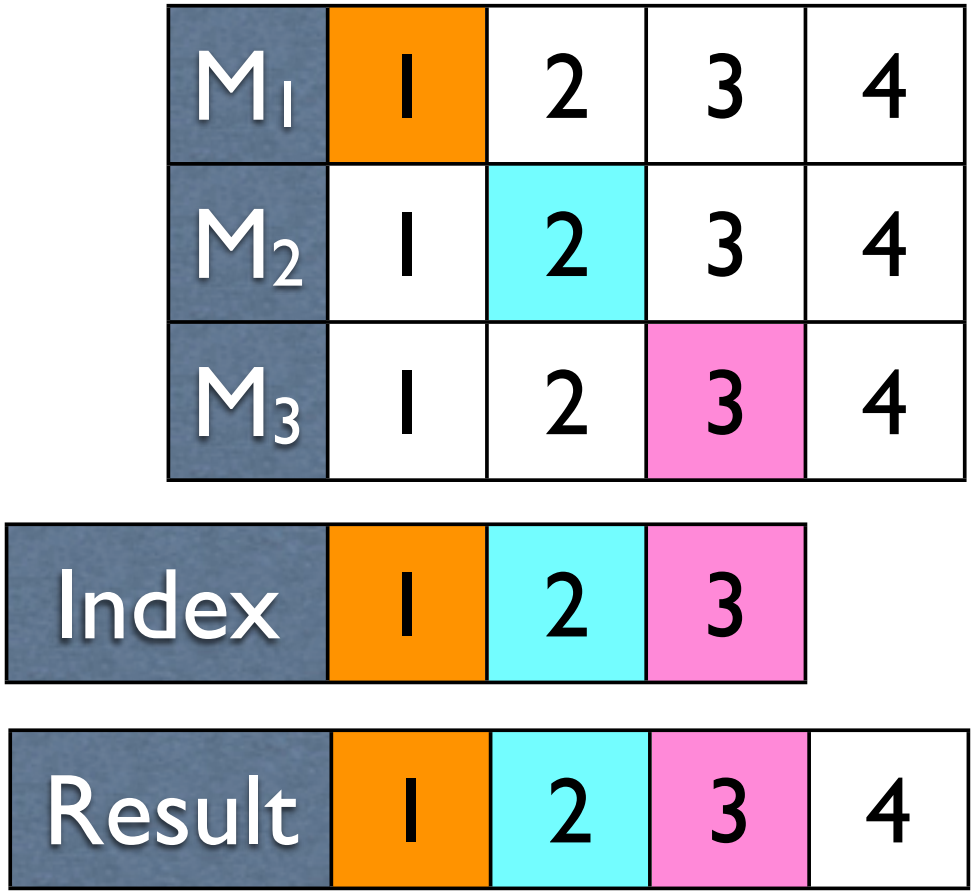
- What do we need to watch?
- Enough to *support* every value
- We've supported every value of Index
- And 1,2,3 of Result
- And some of M

Element Example



- What do we need to watch?
- Enough to *support* every value
- We've supported many values
- Are we done?
- Almost ...

Element Example



- What do we need to watch?
- Enough to *support* every value
- Must support ...
- **Result = 4**

Element Example

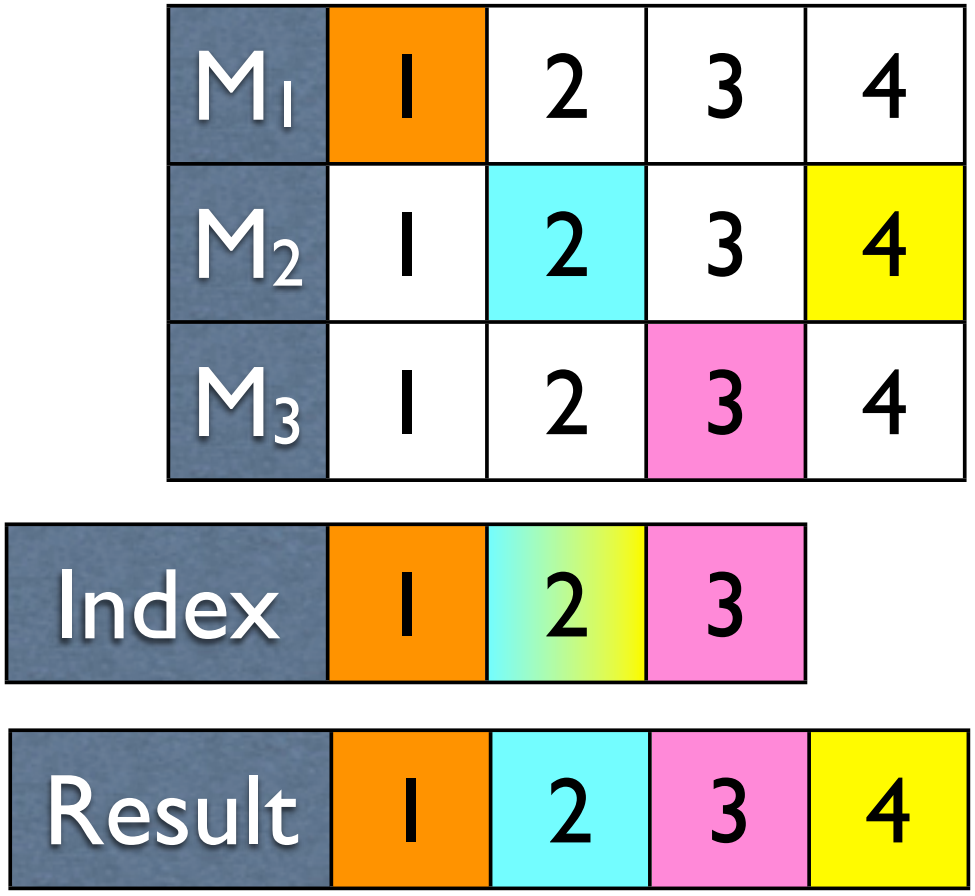
M_1	1	2	3	4
M_2	1	2	3	4
M_3	1	2	3	4

Index	1	2	3

Result	1	2	3	4

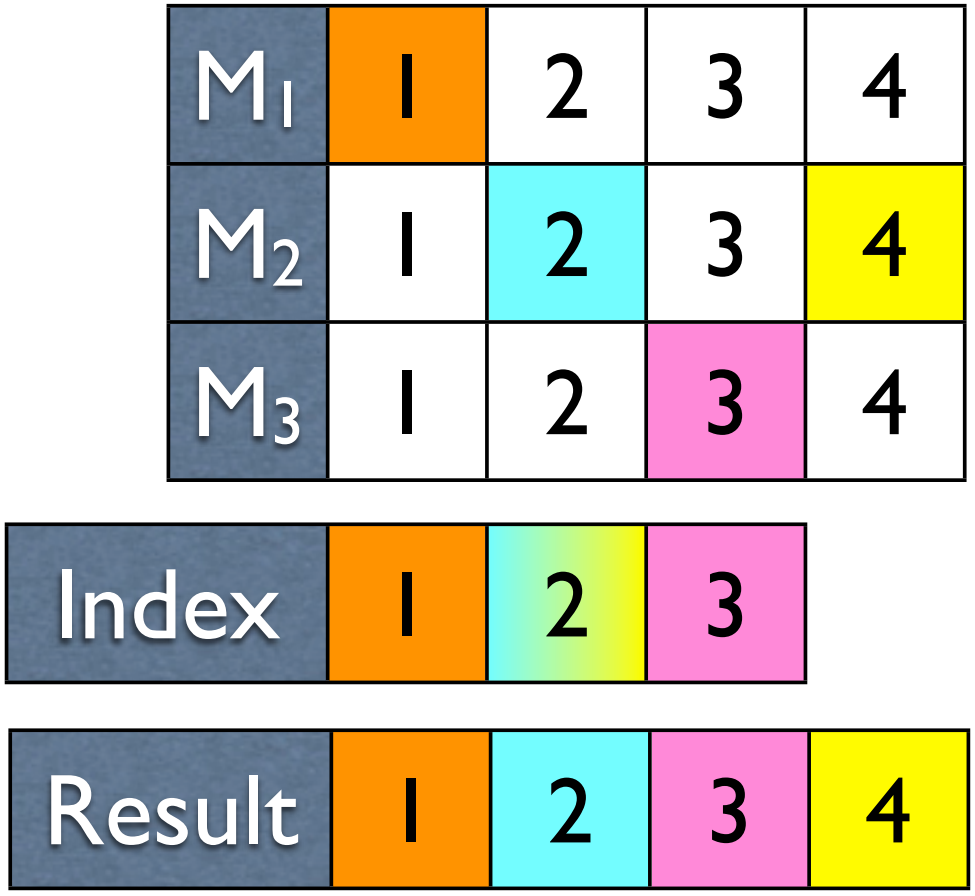
- What do we need to watch?
- Enough to *support* every value
- Must support ...
- **Result = 4**
- **M[2] = 4**

Element Example



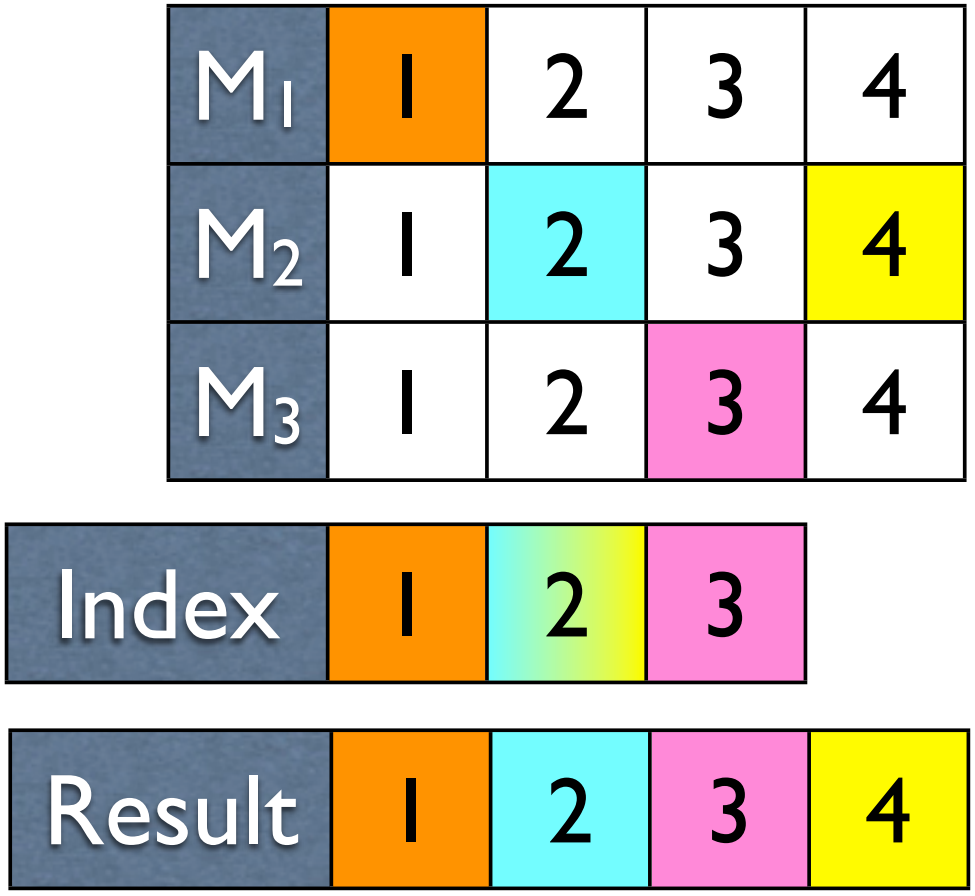
- What do we need to watch?
- Enough to *support* every value
- We've supported every value of Index
- And Result
- And some of M

Element Example



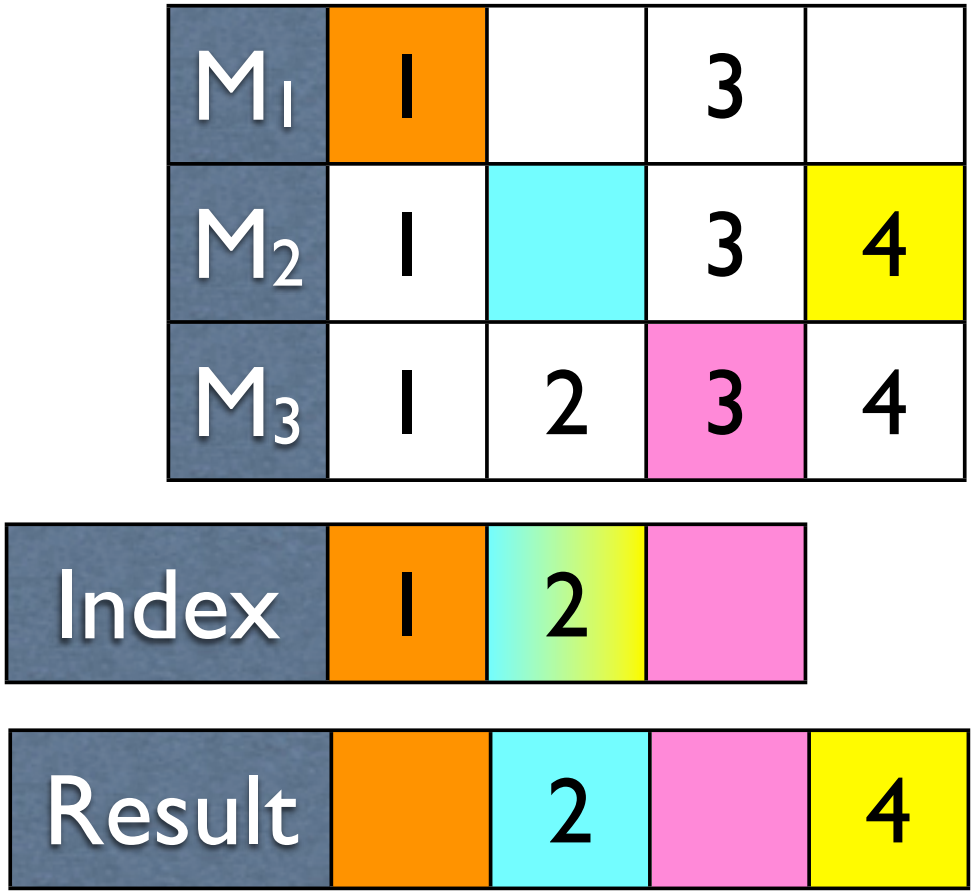
- What do we need to watch?
- Enough to *support* every value
- We've supported every value of Index
- And Result
- And **ALL** of M

All of M?



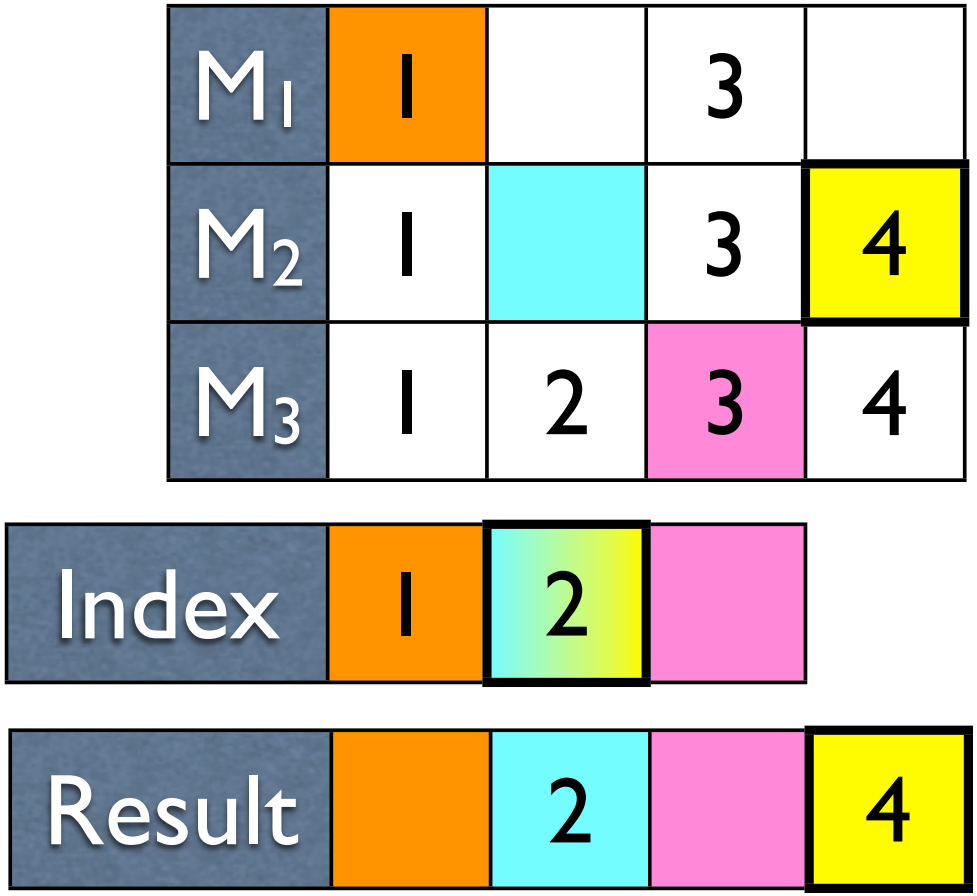
- How have we supported **all** of M?
- Many values are unwatched
- M[Index] = Result
  - While there's two values of Index ...
  - All values of M are possible

Watching literals...



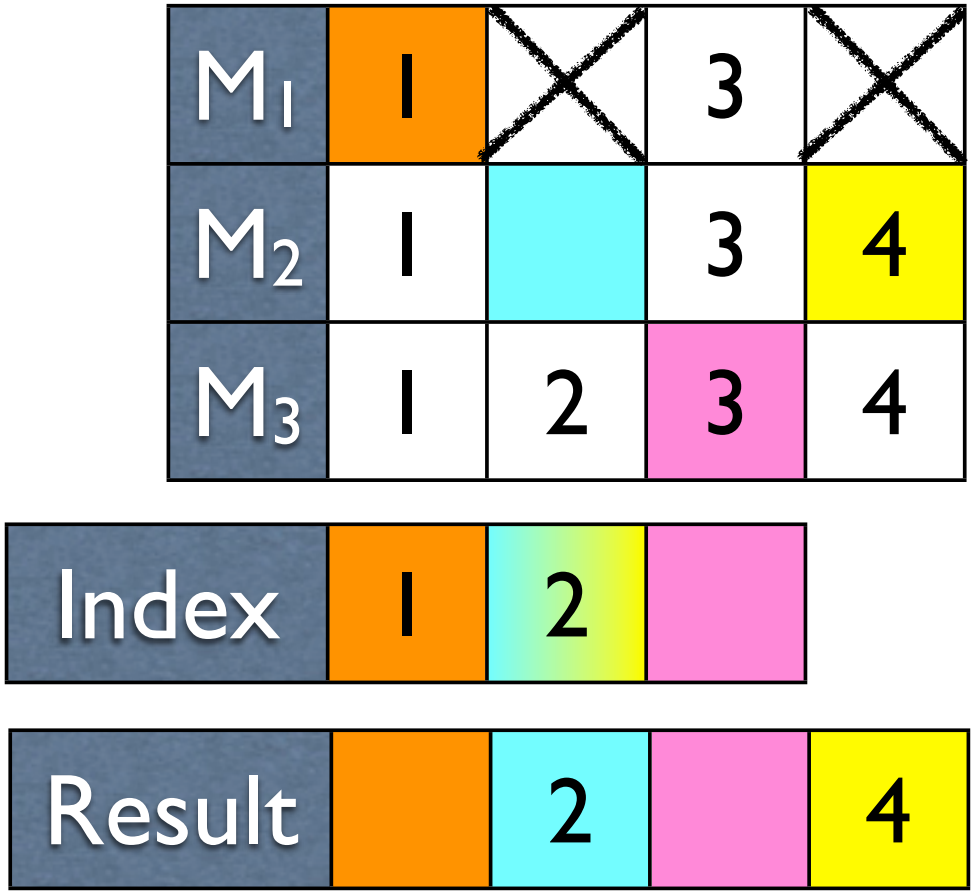
- What happens when literals get deleted?

Watching literals...



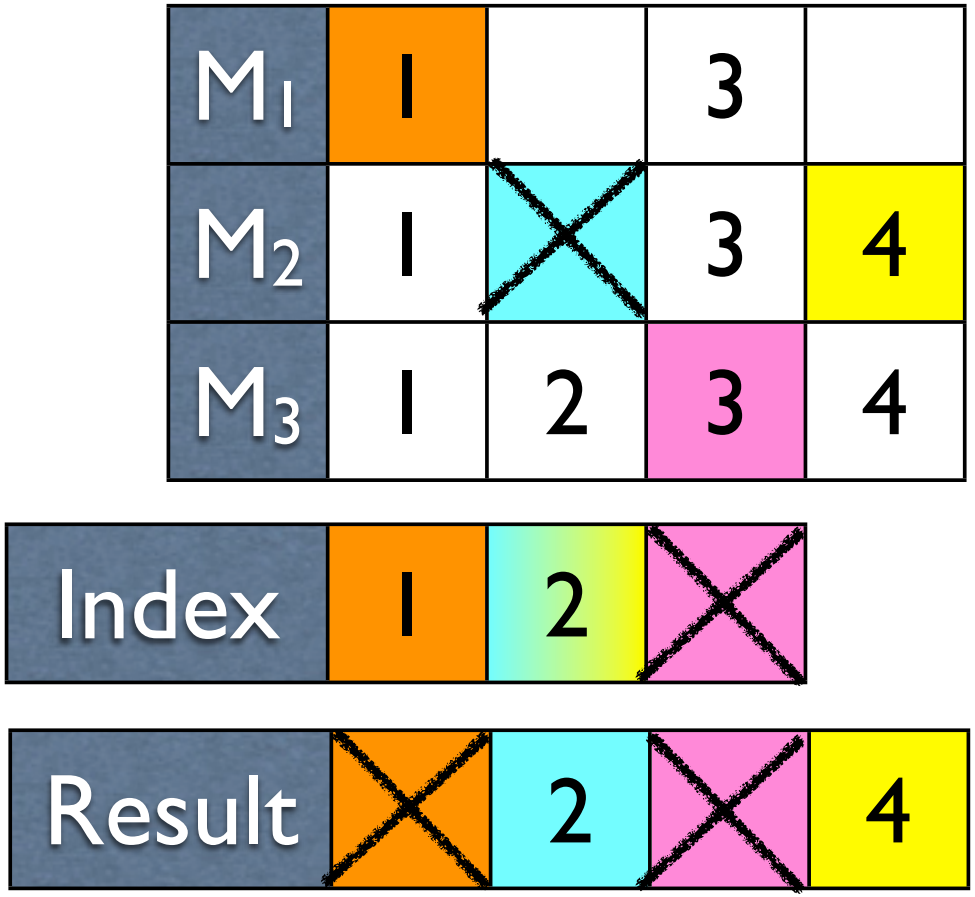
- What happens when literals get deleted?
- **Nothing** ...
- ... for supports where all watched literals still there
- even though domain of every variable involved has changed

Watching literals...



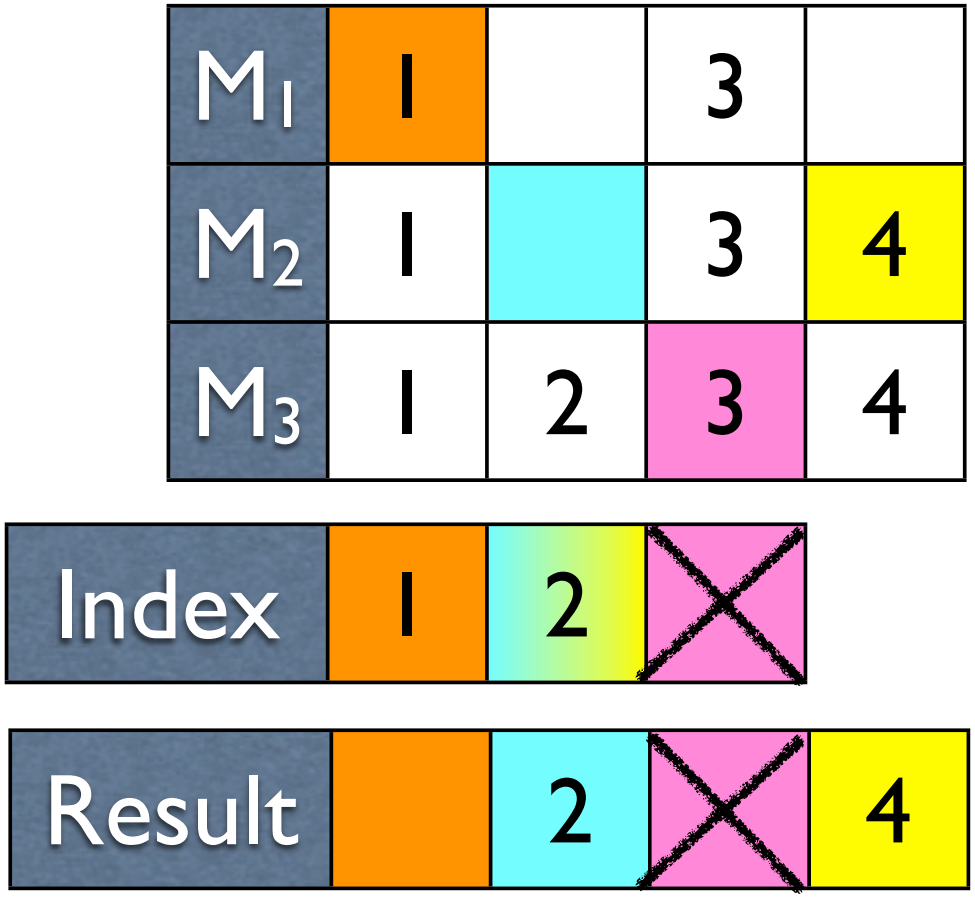
- What happens when literals get deleted?
- **Nothing** ...
  - ... if the literals were not watched
- Huge difference between **Nothing** and O(1)

Watching literals...



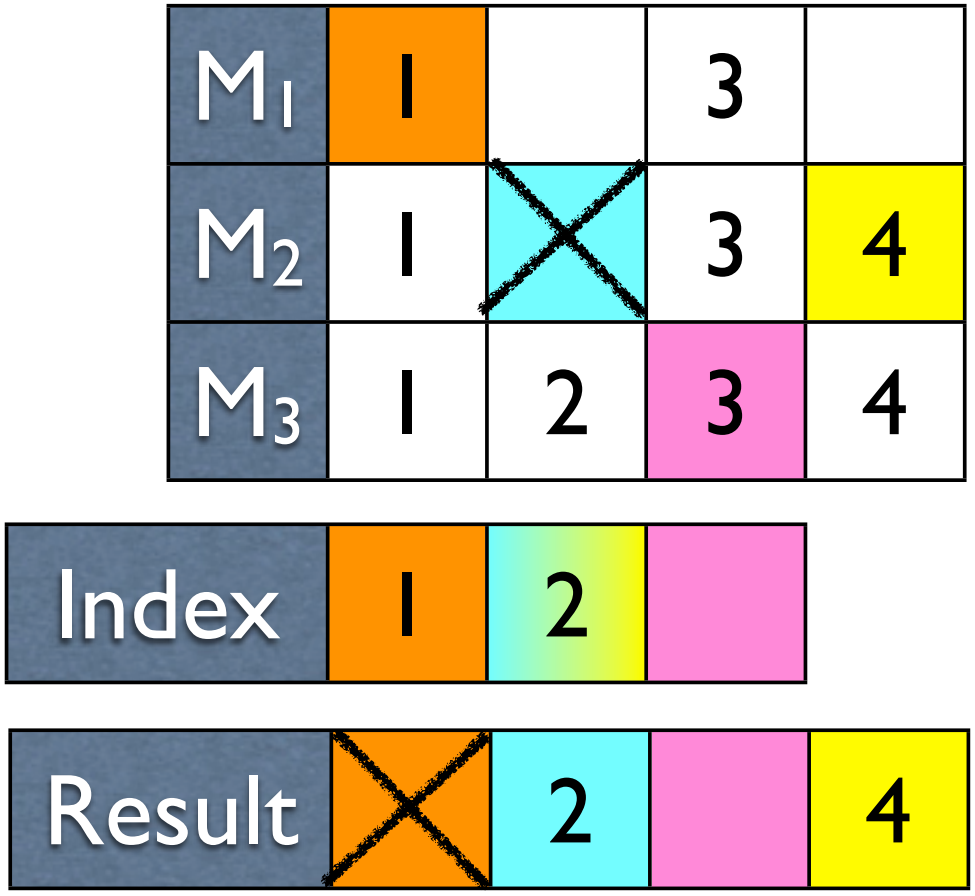
- What happens when literals get deleted?

Watching literals...



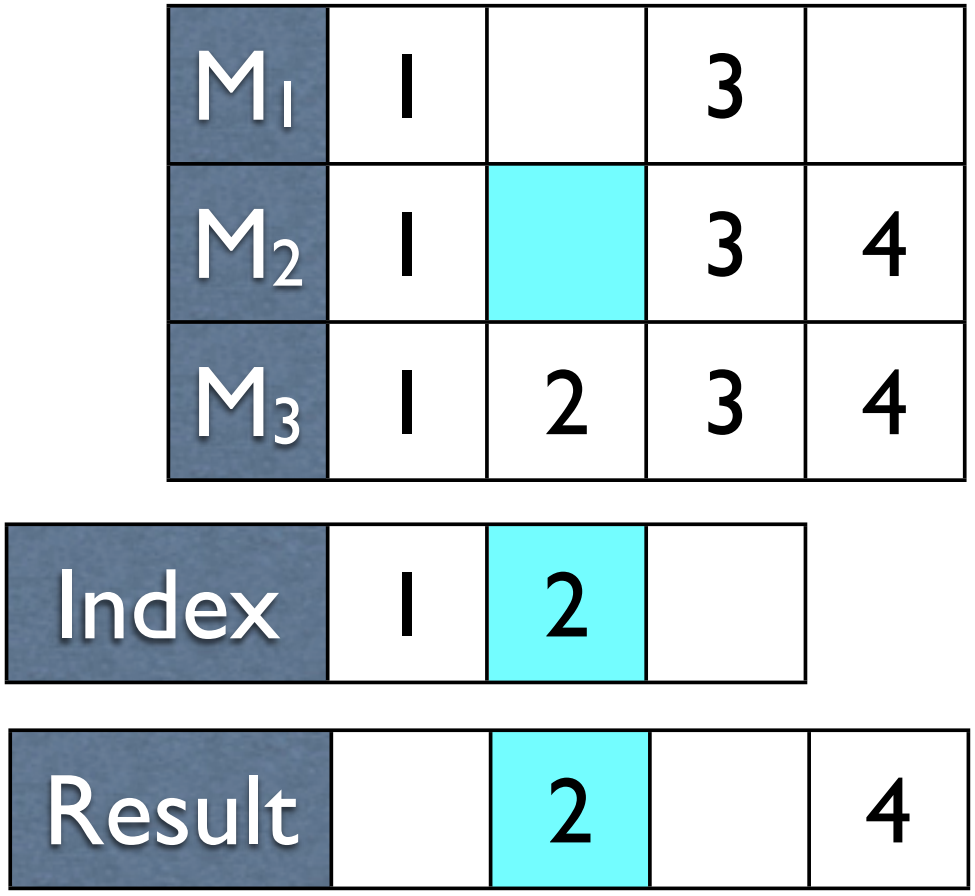
- What happens when literals get deleted?
- **Very little ...**
  - if values being supported have been deleted
- We don't even move watches
  - when we backtrack they will come back to life

Watching literals...



- What happens when literals get deleted?
- **Real work ...**
- ... If deleted literal was watching active support
- ... we must find new support (watches)
- or we will remove values

Element Example



- **Result = 2?**

Element Example

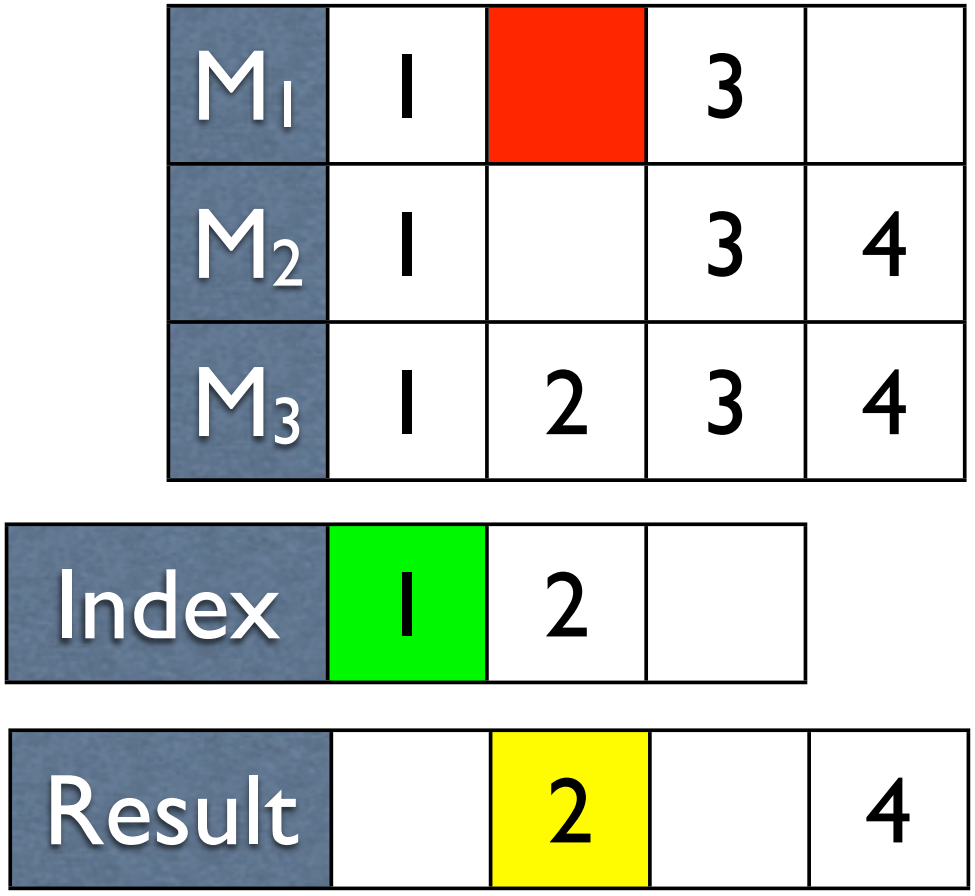
M_1	1		3	
M_2	1		3	4
M_3	1	2	3	4

Index	1	2	

Result		2		4

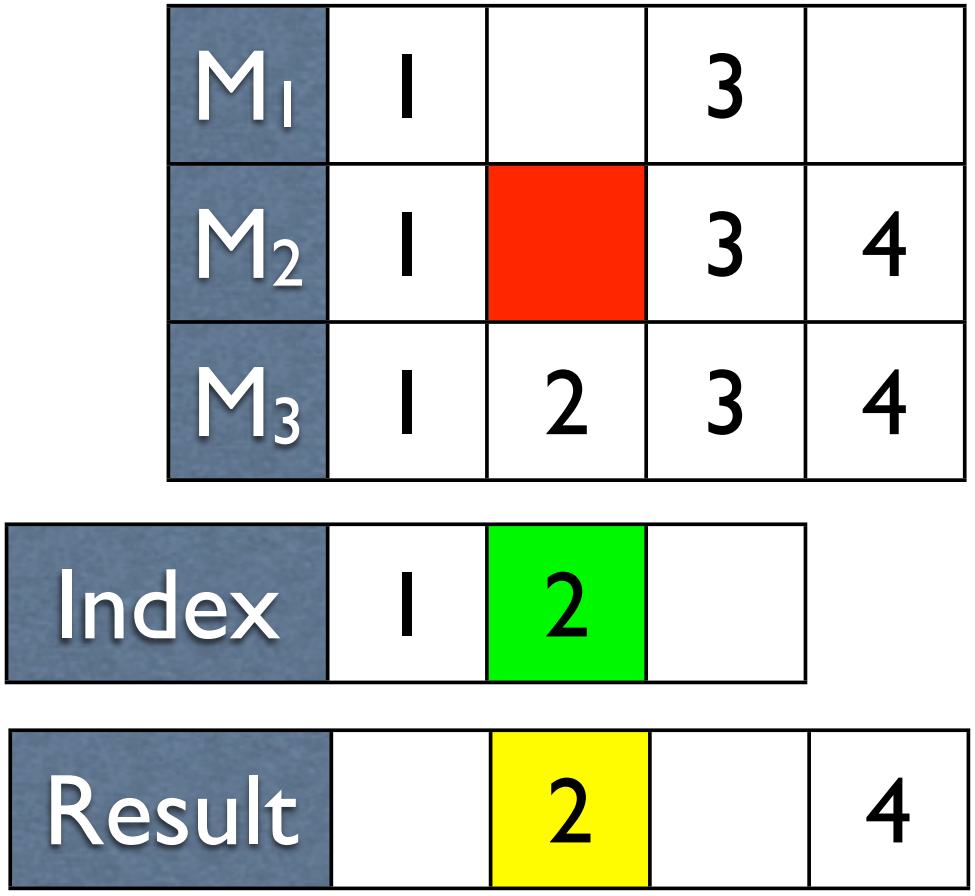
- **Result = 2?**
- None of three possible ways work ...
- ... i.e. provide new watches

Element Example



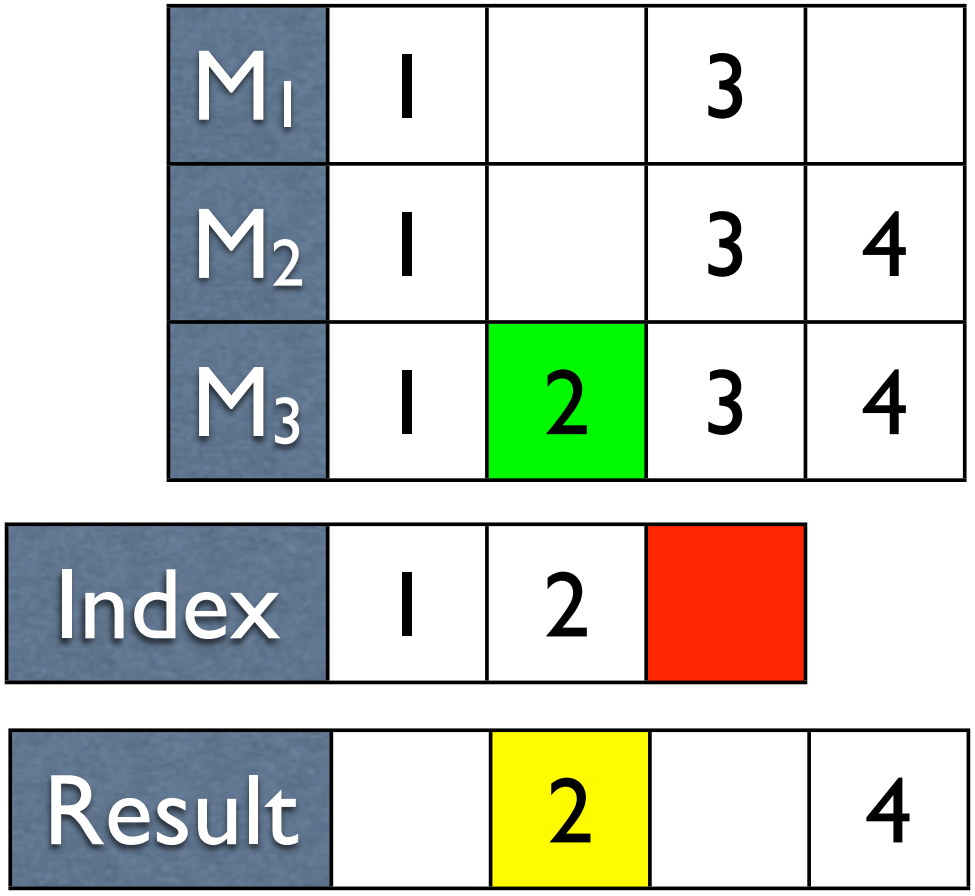
- **Result = 2?**
- None of three possible ways work

Element Example



- **Result = 2?**
- None of three possible ways work

Element Example



- **Result = 2?**
- None of three possible ways work

Element Example

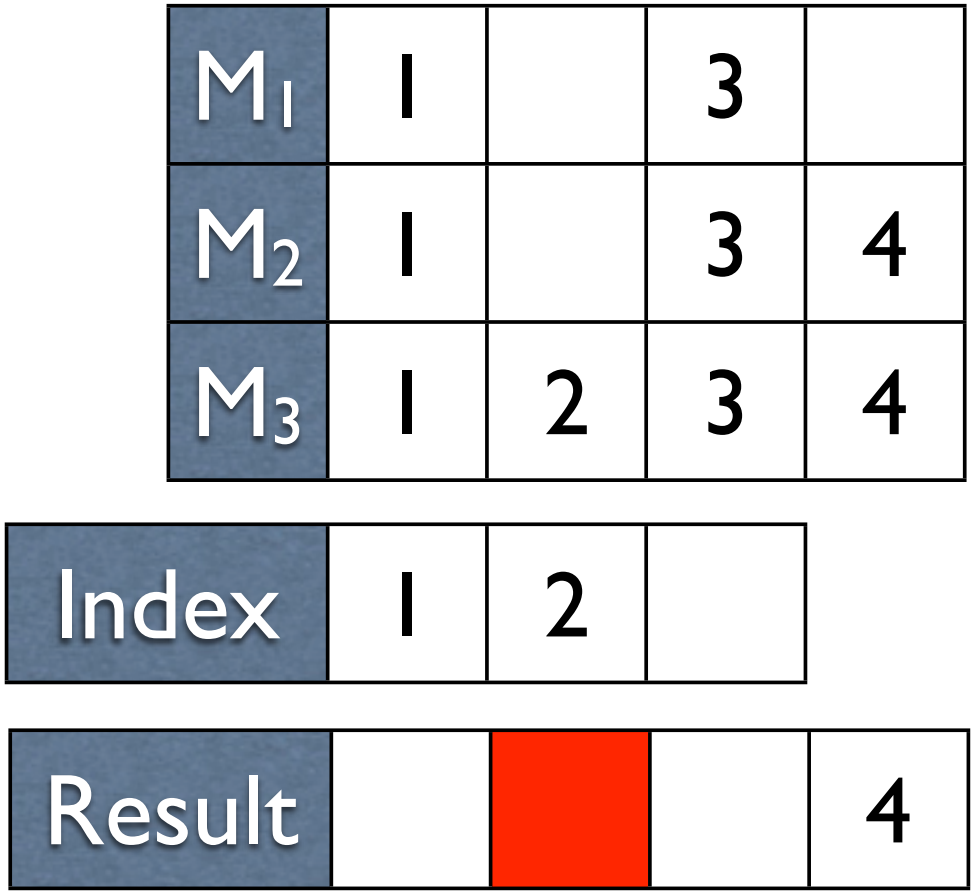
M_1	1		3	
M_2	1		3	4
M_3	1	2	3	4

Index	1	2	

Result		2		4

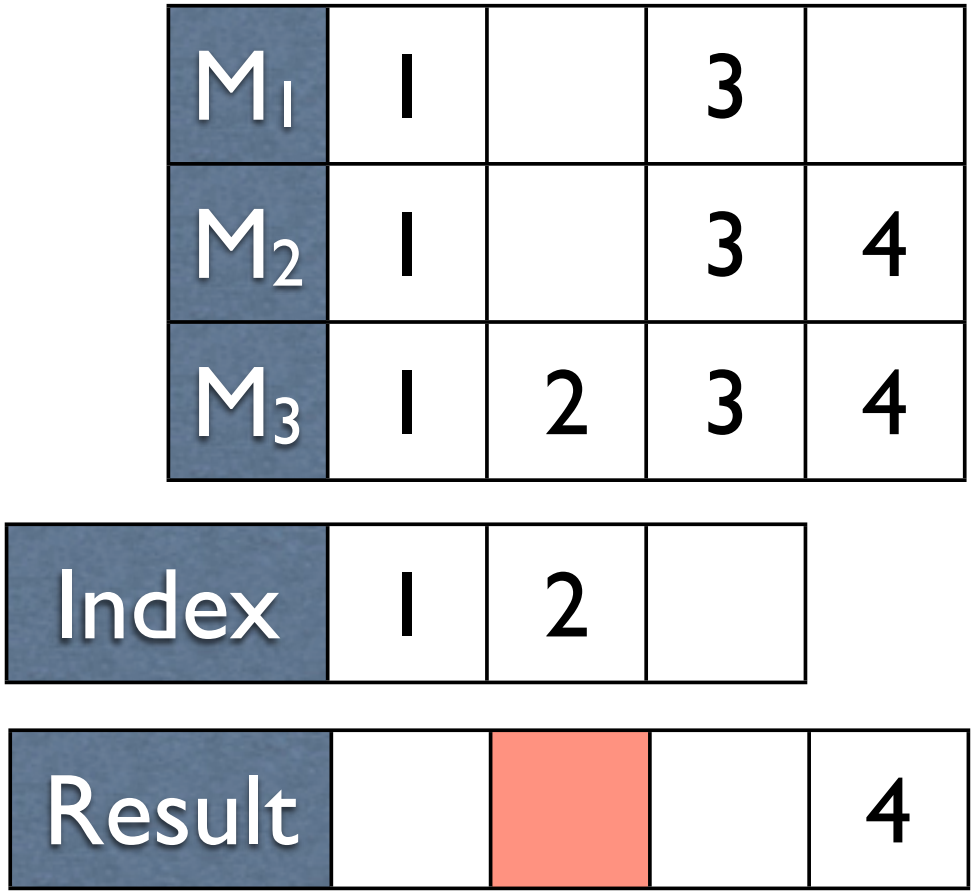
- **Result = 2?**
- None of three possible ways work

Element Example



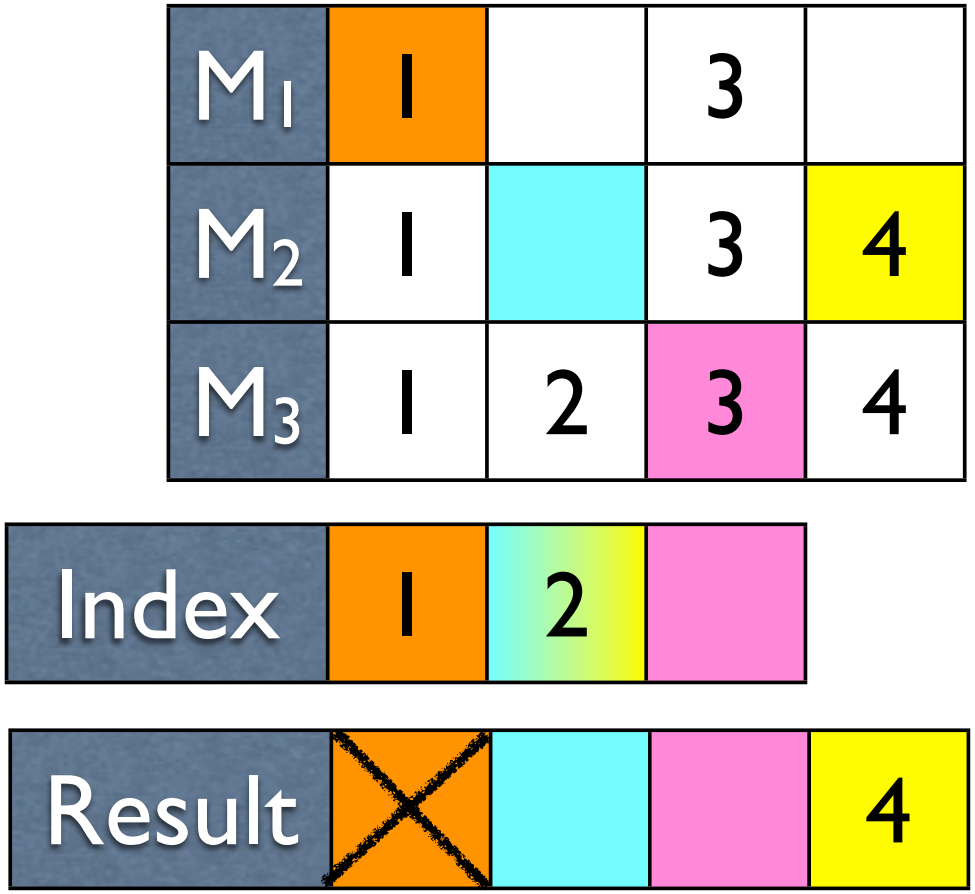
- Remove 2 from domain of Result

Element Example



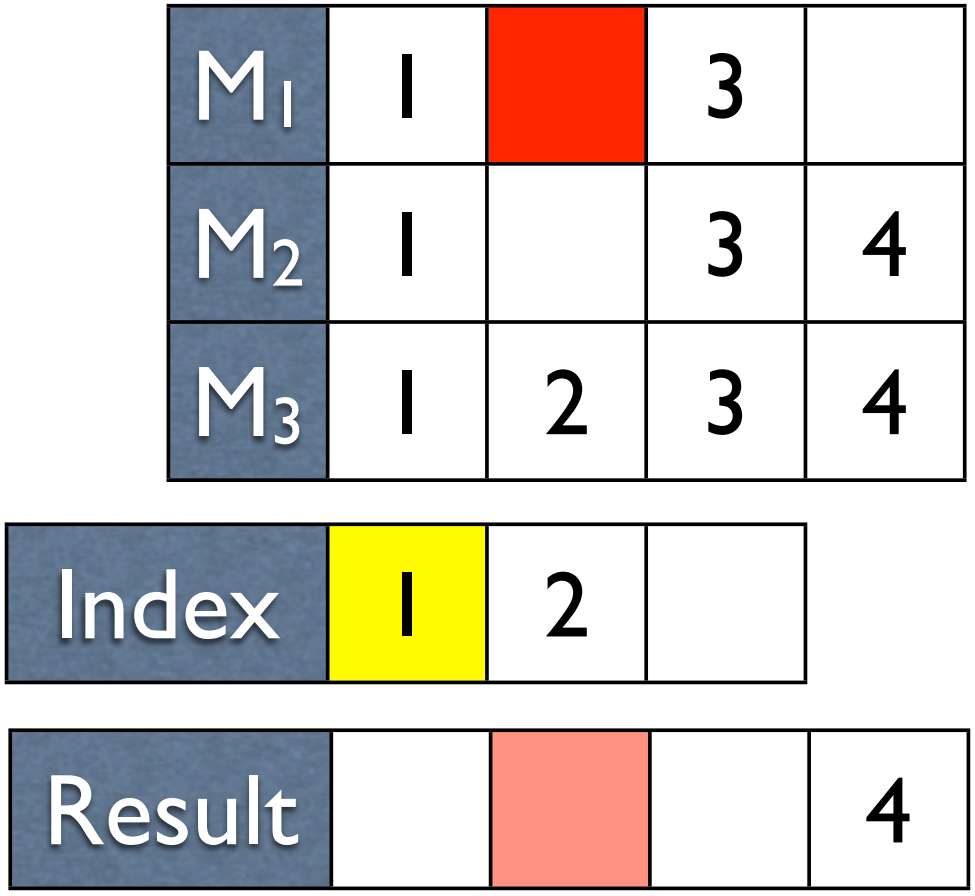
- Remove 2 from domain of Result

Watching literals...



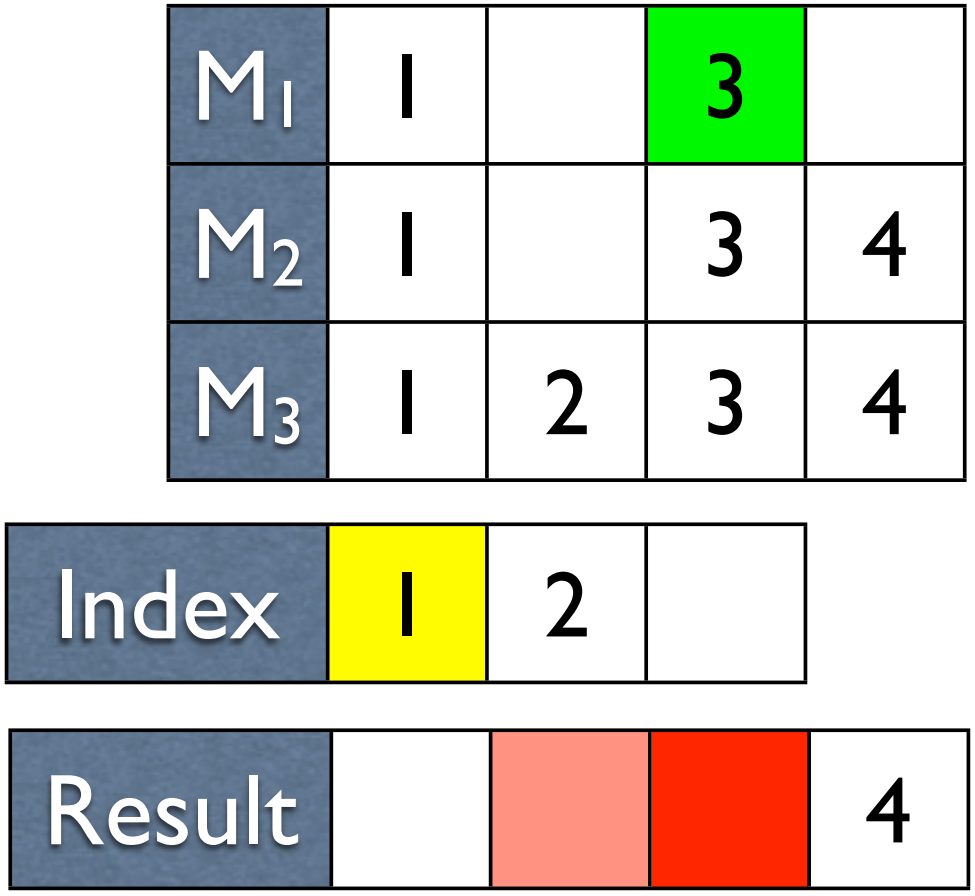
- **Index = 1?**

Element Example



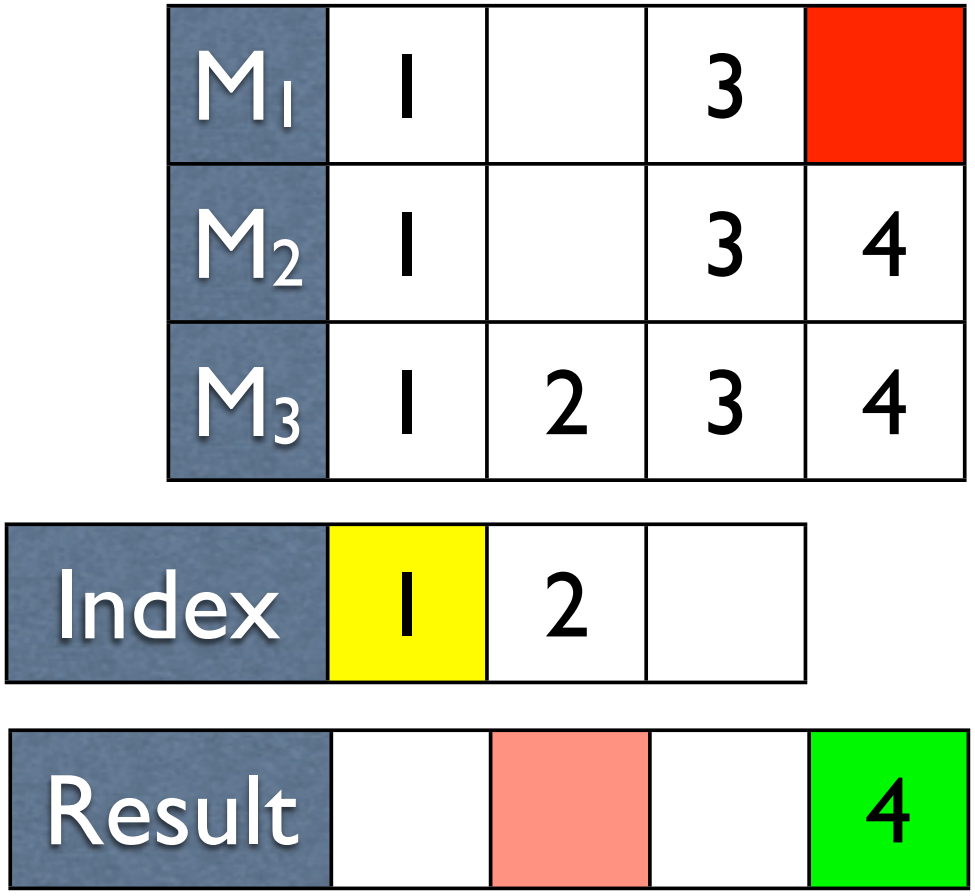
- **Index = 1?**
- No.
- No value of M[1] is the same as a value of Result.

Element Example



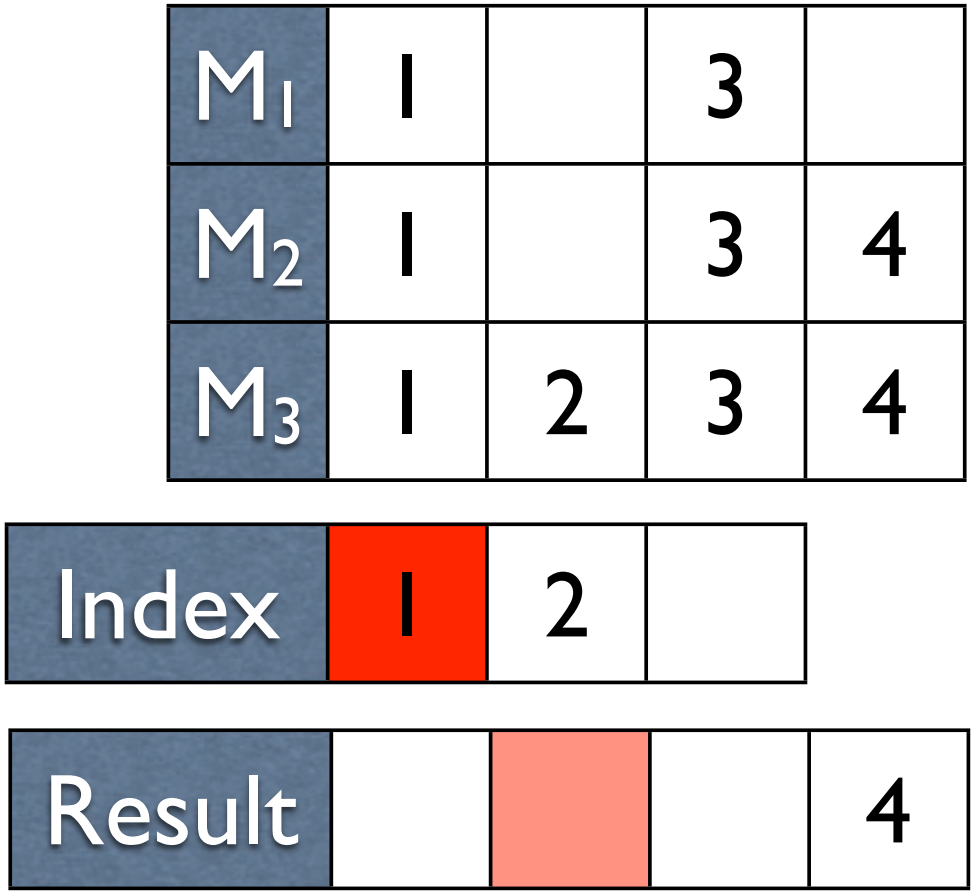
- **Index = 1?**
- No.
- No value of M[1] is the same as a value of Result.

Element Example



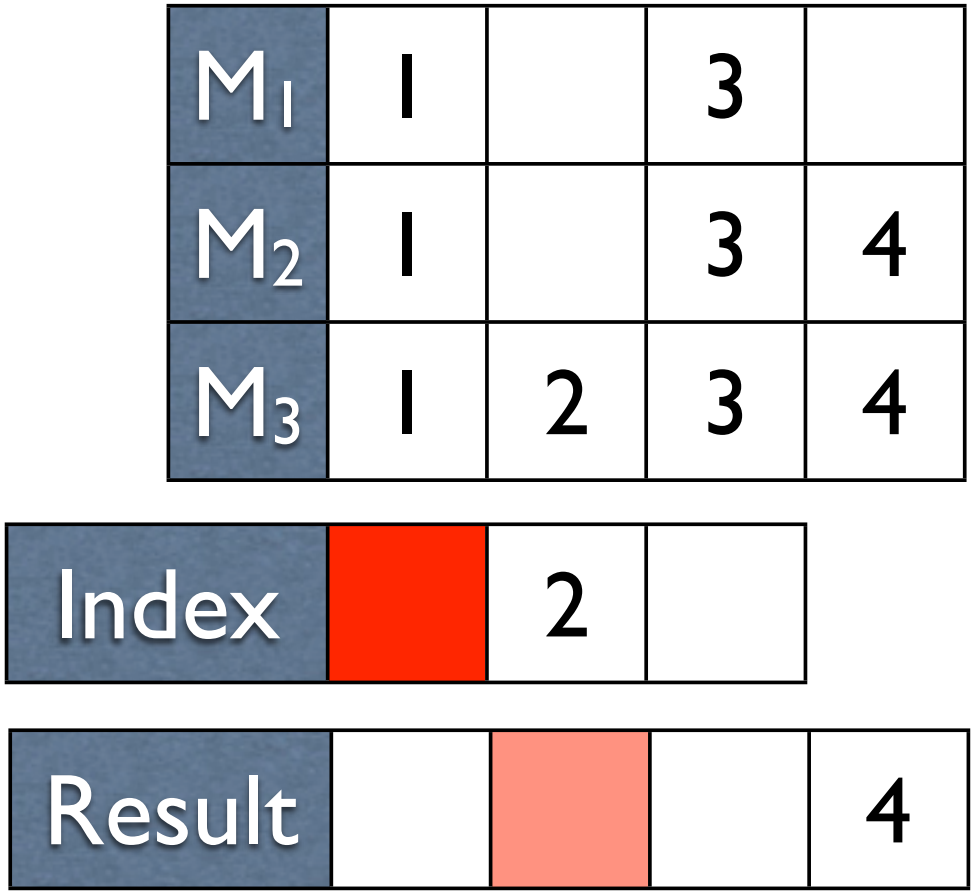
- **Index = 1?**
- No.
- No value of M[1] is the same as a value of Result.

Element Example



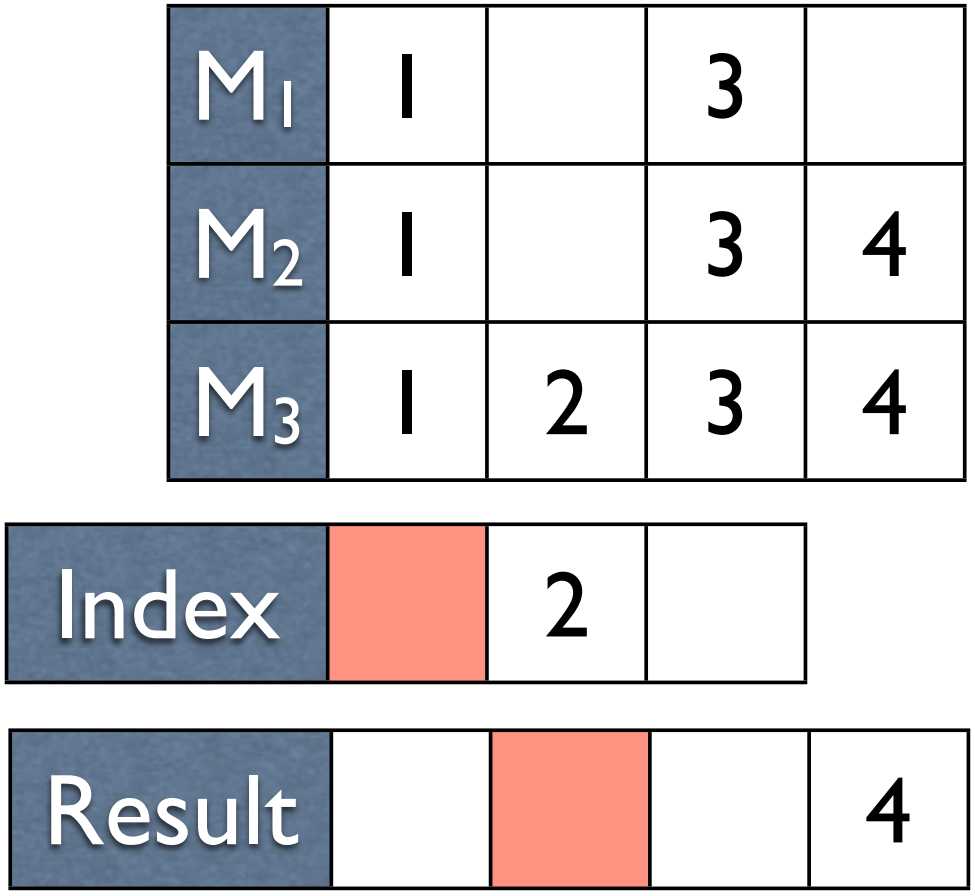
- **Index = 1?**
- No.
- Remove 1 from domain of Index

Element Example



- **Index = 1?**
- No.
- Remove 1 from domain of Index

Element Example



- **Index = 1?**
- No.
- Remove 1 from domain of Index

Key advantage

- If M is vector of size m
  - so Index domain is size m
- And M[i], Result have domain size
- Then we need to watch O(m+n) li
  - one for each value of Index, Result
- But there are O(mn) literals
  - so we often do nothing
- Best way to propagate GAC

Key disadvantage

- Fairly heavyweight infrastructure
  - for operations right in the inner loop of the solver
- Only worthwhile if we win big
- So not if we end up watching most literals
- Can be faster not to do GAC for element

Another advantage

- We can win on space in search
  - which can be critical if search is big
  - and data structures are big
- Because we don't have to backtrack triggers
  - the constraint need put nothing on the backtrack stack
  - memory required for current state of triggers
    - also covers all previous states on this branch
  - i.e. space used by moved triggers reclaimable immediately
- This can be a bigger issue than it sounds

Another Disadvantage

- Constraints get less state, because search may be deeper or higher than when last called.
- Often leads to theoretically worse behaviour.
  - Though in practice this doesn't often matter much

Looping Example

0/1	0/1	0/1	0/1
a	b	c	d

Triggers:

- If triggers backtrack, there is no need to ever loop around from d back to a, as one pass is enough.

My story about Tom Kelsey

```
0 0 2 2 2 2 2 7 7 7
0 0 2 2 2 2 3 7 7 7
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
4 4 2 2 2 2 2 2 2 2
5 5 2 2 2 2 2 3 3 3
5 5 2 2 2 2 2 3 2 3
2 2 2 2 2 2 2 2 2 2
5 5 2 2 2 2 3 3 2 3
4 4 2 2 2 4 4 2 4 2
```

- An example of a *semigroup*
  - mathematicians study these
  - various algebraic constraints
- Enumerated by Minion
  - Distler/Kelsey/Kotthoff
  - 72.9 CPU *years*
  - 50,000 found per CPU *second*

Not a panacea

- I find WL's in CP super cool and fun
  - and *sometimes* much faster
- But they are *not* a universal cure
- Typically use in constraint which is
  - not too tight (lots of allowed tuples)
  - lots of cases where not all vars in support
- Part of a mixed system of triggers