Constants Matter: Implementing Minion, a fast Constraint Solver

Chris Jefferson

Disclaimer

- *Not many of the following results have been extensively benchmarked.*
- *A combination of:*
  - *Small ad-hoc experiments at the time.*
  - *Experience with other programs.*
  - *The voices in my head.*

Minion Disclaimer

- *Minion is not "The Fastest Solver"*
- *Minion is quite buggy.*
  - *Currently 48 hour bugfix turnaround.*
- *Minion lacks many global constraints.*
- *Minion is a "part time" project for a small number of people.*

Black Box Solving

- *Minion, a GPL Constraint Solver.*
  - *http://minion.sourceforge.net*
- *Optimised for solving large, hard problems.*
- *Number of design decisions inspired by SAT solver, such as zChaff.*
  - *e.g. careful design of data structures.*

Minion Limitations

- *Some things are easy to add.*
  - *New constraints.*
- *Some things are reasonably easy to add.*
  - *New search heuristics.*
- *Some things very hard to add.*
  - *Add constraints, variables during search.*

Minion Input

- *Minion is supposed to be used with a (not very good) text file format.*
- *Can be linked as a library, but not supported.*
  - *Have to be very careful to do things in the right order.*

Motivation: SAT Solvers

- *Most SAT solvers are highly optimised black boxes.*
- *Made a group of design decisions.*
- *Most of these are very hard to change.*
- **VERY** *fast*

Results - QG7.13

	Nodes/sec	*Slower than SAT*
ILOG 6.3	*25*	*197*
Minion	*397*	*12*
WL-Minion	*1,728*	*2.8*
MiniSAT	*4,932*	*1*

Simple != Bad

- *Many features of Minion are stupid.*
- *Don't test a feature's efficiency by turning it off.*
  - *Reimplement code to take advantage of extra simplicity.*
- *Avoiding the overheads can make up for lacking the feature?*

“Stupid” Design choices

- *No adding / removing constraints or variables during search.*
  - *Memory blocks can’t expand mid-search.*
- *Just copy all of memory on branch.*
  - *If state large, only store some states like GECode (since last night).*
- *Variable representations mostly bit arrays.*

Static & Locality

- *Before search decide all the memory which will be required:*
  - *Backtrackable - variables, state.*
  - *Nonbacktrackable - lists of triggers.*
- *Put it all in one place, as compactly as possible.*
  - *Leads to a slow startup.*

Advantages of Static

- *Memory locality.*
  - *Not benchmarked enough.*
- *Everything in a fixed location.*
  - *Very little indirection, constraints have pointers directly to variable's memory location.*
  - *about 30% speedup*

‘Smart’ Pointers

- *To rearrange memory, Minion uses ‘Smart’ pointers.*
  - *Smartness in constructor, pointer class contains a single pointer.*
  - *During search, identical to a normal pointer.*

‘Smart’ Pointers

- *Log location of all pointers, and how much memory allocated.*
- *In one sweep, move all data together and rearrange compactly.*
- *Re-write the pointers to point to the new location.*

Queues

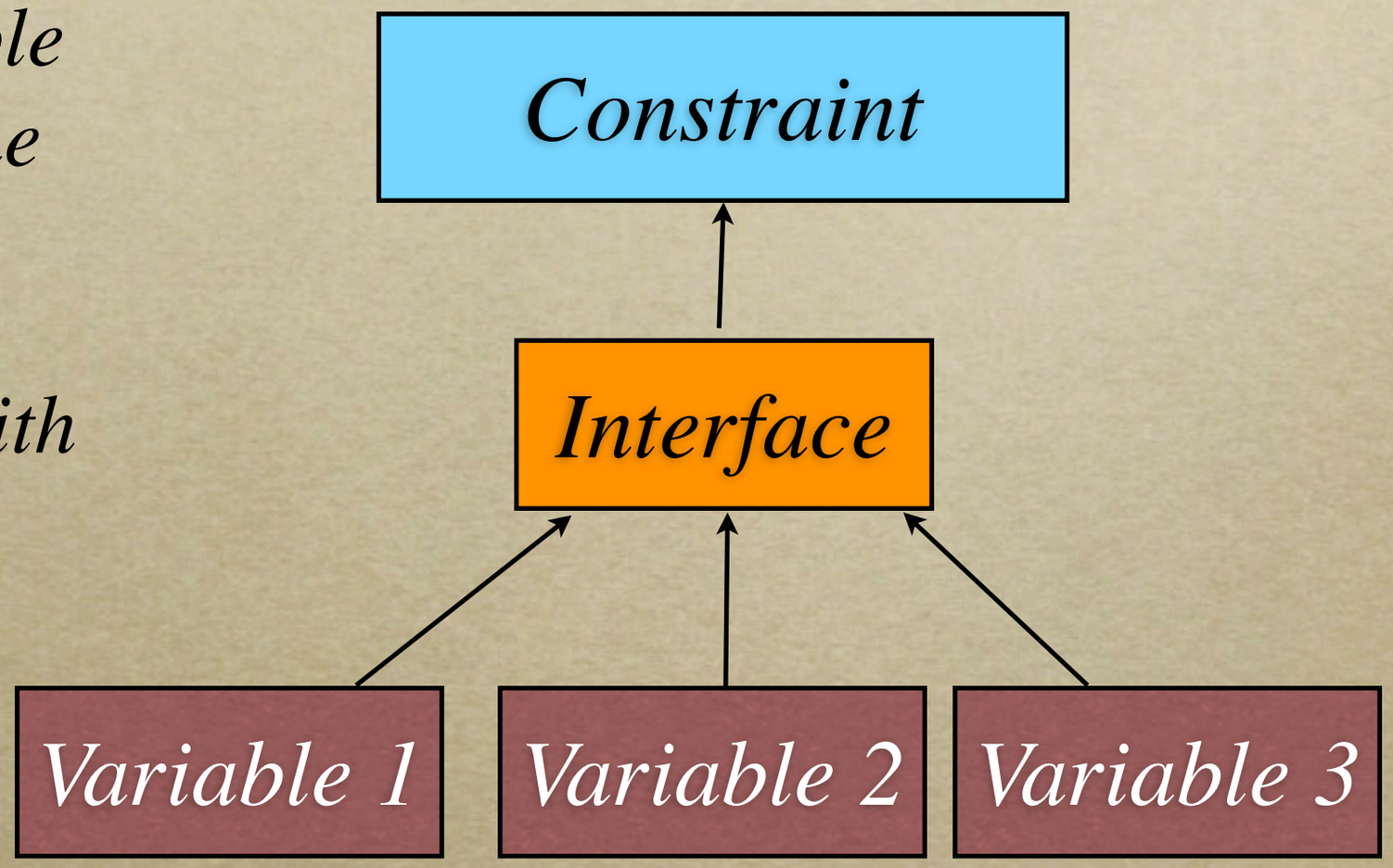
- *Trigger queues are fixed before search.*
- *Queue consists of pointers to these queues.*
  - *Removes ability to rearrange queue.*
  - *Constraints cannot be added/revoked.*
  - *Avoids copying the queue.*

Variables in Minion

- *No "best" variable representation.*
- *Specialise for:*
  - *Booleans & Small domains*
  - *Bound variables*
  - *Constants*
- *Lacking : Full large domains.*
  - *To come....*

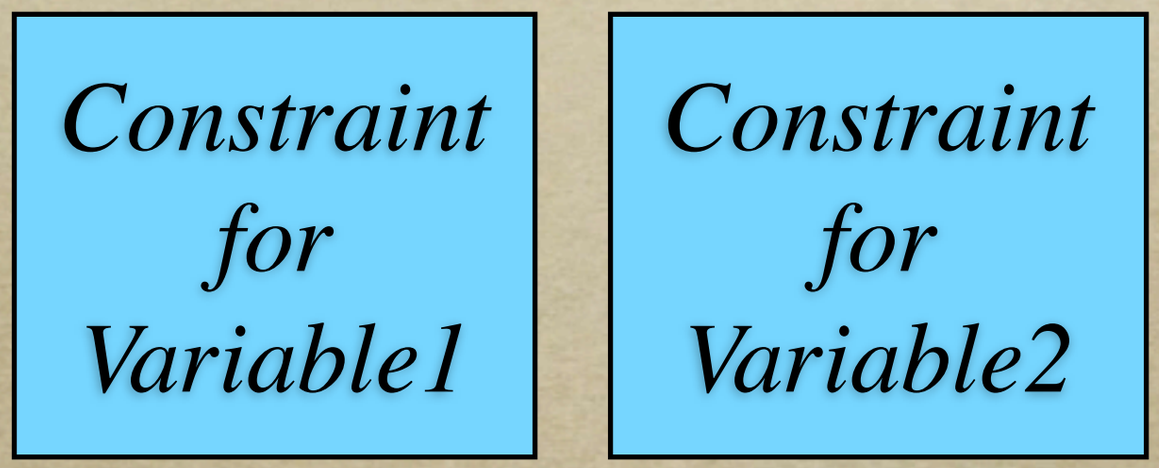
Many Variable Types

- *Implementing multiple variable types for one constraint:*
- *Abstract interface with variables chosen at run-time.*
- *No inlining.*
- *Half speed on SAT.*



Different Variable Types

- *Implement **each** constraint for **each** type of variable.*
- *Fast.*
- *Have to write too much.*



Compile-time Interfaces

- *Define a minimal interface and **compile** each constraint with each variable type.*
- *Compiler optimisation removes the interface.*
- *Allows most constraints to have a single implementation.*

Templates

- *Specialisations are done with **templates***
- *A C++ compile-time feature which allows function to be compiled for multiple types.*

Templates

```
template<class var_array>
.....
void propagate(int prop_val)
  {
    int remove_val = var_array[prop_val].getAssignedValue();
    int array_size = var_array.size();
    for(int i = 0; i < array_size; ++i)
    {
       if(i != prop_val)
              var_array[i].removeFromDomain(remove_val);
    }
}
```

Template Advantages

- *Templates have a bad reputation.*
- *Could just write function many times, substituting in types and avoid templates.*
- *Templates are **identical** to do this.*

Template Problems

- *Templates do not solve all problems.*
- *Optimisers are not capable of doing higher-level reasoning.*
  - *On Booleans, domain reduction is the same as assignment.*
  - *Reducing code in these cases can produce benefits.*
  - *50% faster Boolean sum.*

Template Problems

- *Need to generate all possible functions you might want at **compile time.***
  - *Long compile times.*
  - *Exponential growth as more options added.*
- *Generate the solver for each problem?*

Variable Interface

- *Two general methods:*
  - *Variables deal with inconsistency.*
    - *a.setMax(1), a.setMin(2) - fails.*
  - *Constraints deal with inconsistency.*
    - *a.assign(1) invalid if not in domain.*

Variable Interface

- *Variable propagation makes constraints easier to write.*
- *Small overhead when you know that assignment is valid*
- *Provide "unchecked" methods.*

Boolean Variables

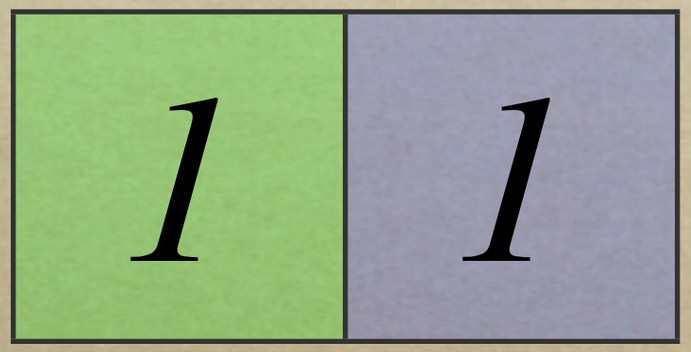
- *Simplest kind of variable.*
- *Many problems have huge number of Booleans.*

Represent with two bits:

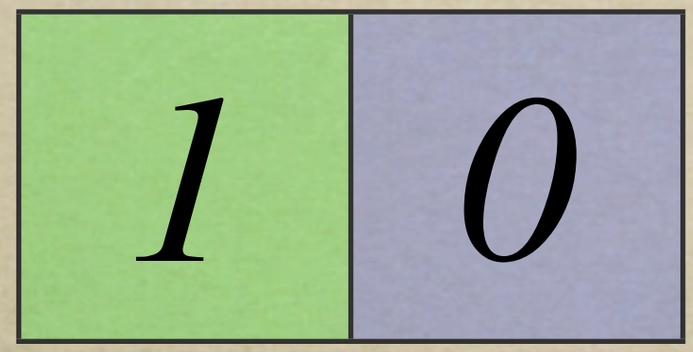
Is Assigned

Value Assigned

Boolean Variables



Assigned True

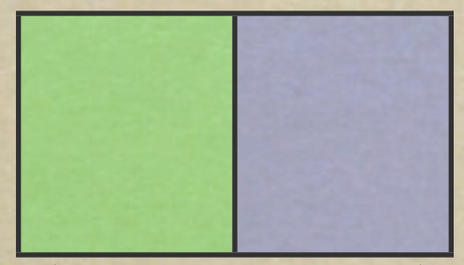


Assigned False

Boolean Unassigned

Boolean Variables

- *The "assignment" bit is not backtracked!*



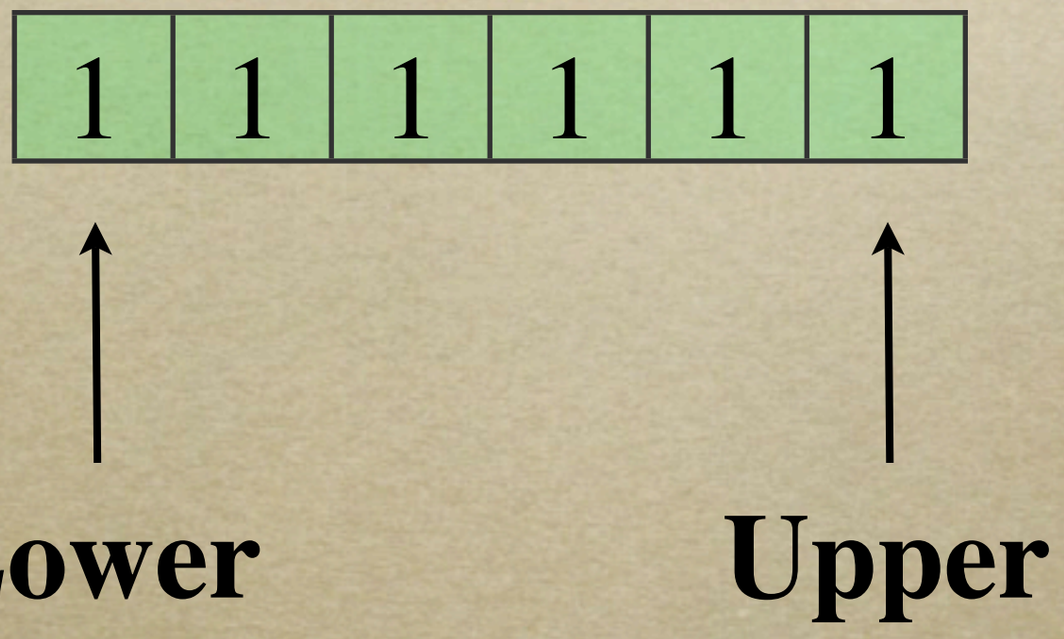
- *If variable still assigned, has same value.*
- *If unassigned, value unused.*

Bound Variables

- *Store only the current upper and lower bounds.*
- *Very low memory requirements.*
- *Loss of information*
  - *{1,3,5} → [1 .. 5] → {1,2,3,4,5}*
- *Space / Time tradeoff.*

Discrete Variables

- *Use a Boolean array for domain values.*



- *Store upper and lower bounds.*
- *Array only valid between pointers.*
  - *'O(1)' bound update.*

Variable Mappers

- *Consider you want X = -Y.*
- *Given X's internal state, Y's is redundant.*
- *Provide "Variable Mappers"*

Minus Mapper Example

- *getMax( ) { return -X.getMin( ); }*
- *setMax(int i) { X.setMin(-i); }*
- *inDom(int i) { return X.inDom(-i); }*
- *Have to set up all triggers the other way around as well...*

Mapper Advantages

- *Can often remove many variables*
- *Makes constraints easier to implement.*
  - *Weighted sum = normal sum + mappers.*
- *Implementation in one constraint no faster.*
  - *But mappers do cost (division).*

SumLeq in Minion

- *Weighted SumLeq (mappers)*
- *Positive Weighted SumLeq (mappers)*
- *-1 / 1 Weighted SumLeq (mappers)*
- *SumLeq*
- *Boolean SumLeq (simplified implementation)*
- *Boolean Sum to Constant (WL).*

Complex Mappers

	Col1	*Col2*	*Col3*
Row1	*B1*	*B2*	*B3*
Row2	*B4*	*B5*	*B6*
Row3	*B7*	*B8*	*B9*

- *9 Booleans, 3 Row variables, 3 Column variables.*
- *Channelling / Viewpoints for free(ish)!*
- *Not in public version yet.*

Propagation in CP

- *Constraints attach a trigger to each variable they want to be informed about.*
- *Different types of trigger:*
  - *Value Removed.*
  - *Bounds Changed.*
  - *Variable Assigned.*

Watched Literals

- *Inspired by SAT.*
- *Different from normal triggers:*
  - *Each constraint has a fixed number.*
  - *Cheap to move to different literals.*
  - *Not moved back to original place on backtrack.*

Watched Literals

- *Watched literals can be used to implement a number of constraints.*
  - *See conference talk.*
- *They require additions deep inside the solver.*

Why Watched Literals?

- *Watched literals involve more overhead than a normal trigger.*
  - *Stored as linked list rather than array.*
- *Constraints (hopefully) watch very few literals in the variables.*
- *Improvements to many constraints*

Minion's Watched Literals

- *Each constraint has a fixed number of watched literals, declared at the start.*
- *Watched literals can be moved.*
  - *Also connected to no variable.*
- *Watched literals never created or deleted during search.*

WL Implementation

```
struct DynamicTrigger
{
    DynamicTrigger* prev;
    DynamicTrigger* next;
    DynamicConstraint* constraint;
    int info; // Let constraint store a note.
};
```

WL Implementation

- *For each literal, keep a pointer to start of list.*
- *Doubly linked list for every literal.*

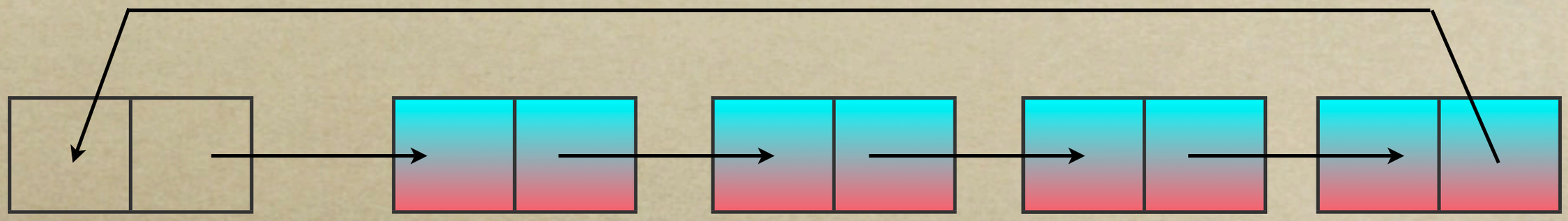
WL Queues

- *When a literal is deleted, if anything is watching it, put it on a queue.*
- *Queue is not watches, but literals which have a non-empty list of watches.*
- *This is necessary.*

Watched Literal queue

- *Watches move around all the time.*
- *Is important to make sure a watch is never triggered if it is moved to a non-deleted literal.*

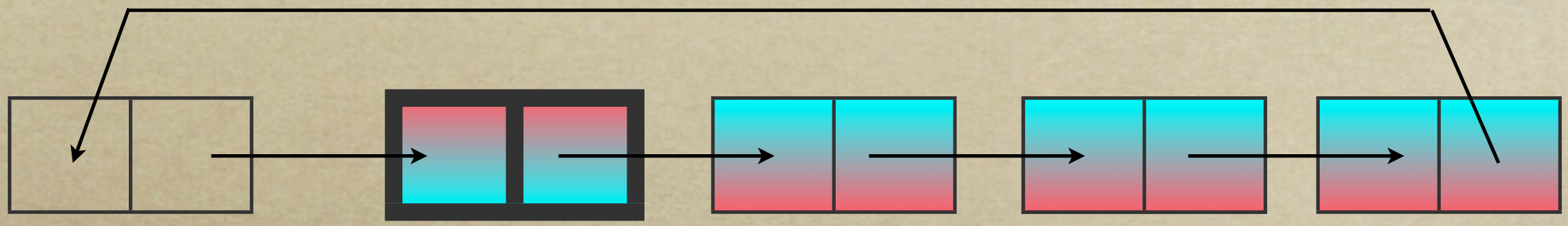
Implementation Difficulty



Base of queue

- *Need to iterate through queue, execute constraint at each position.*

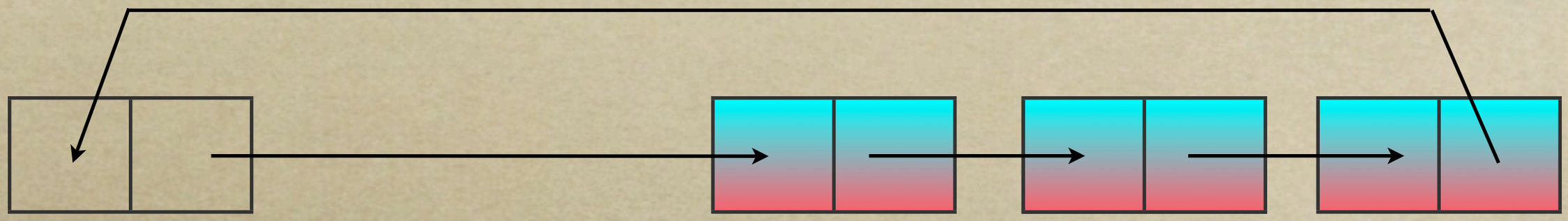
Implementation Difficulty



Base of queue

- *Run first algorithm...*

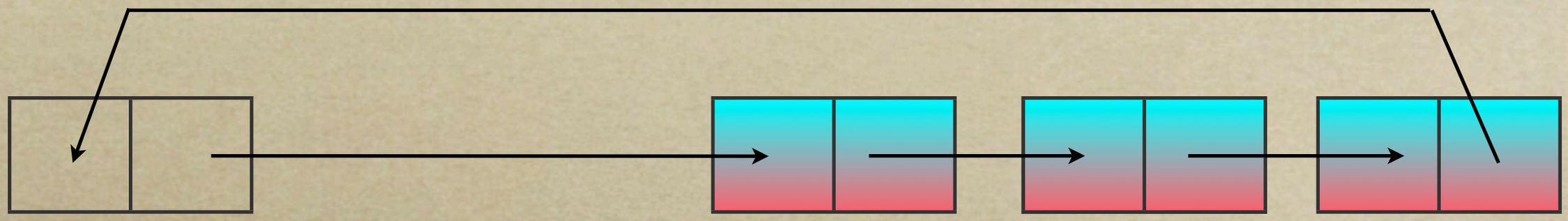
Implementation Difficulty



Base of queue

- *Watch is moved!*

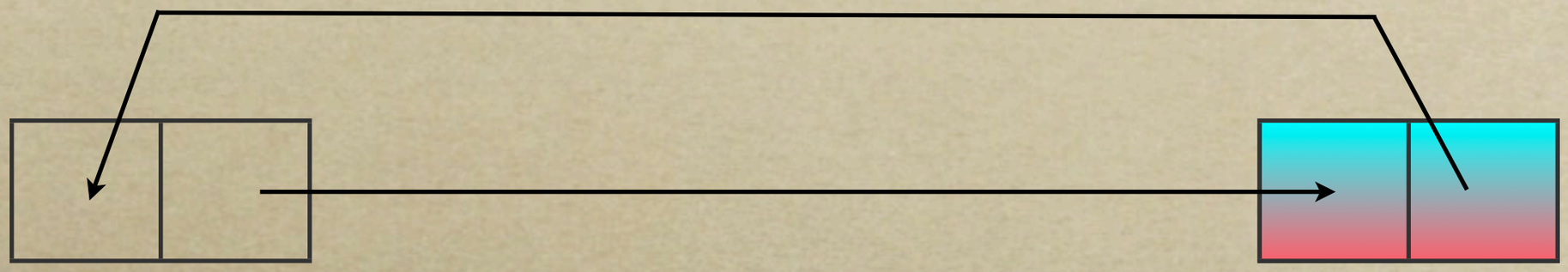
Implementation Difficulty



Base of queue

- *Watch is moved!*
- *Even worse case....*

Implementation Difficulty



Base of queue

- *Other literals move!*

Implementation Difficulty

- *We tried forbidding constraints from doing this kind of thing.*
  - *Implementation becomes very messy and slow.*
- *The current solution is not perfect.*
  - *But it works!*

Minion's Fix

- *Keep a global variable* "nextwatch"
- *Stores which watch will trigger next.*
- *If a watch finds it is* nextwatch, *when it moves set* nextwatch *to its next pointer.*
- *There might be other / better ways of getting around this problem.*

General Conclusion

- *The simplest algorithms are often still fast.*
- *CSP solvers can get close to SAT.*
- *Need better benchmarks of features.*
  - *Hard but important to test not just disabling a feature if it has overhead.*

Minion Conclusion

- *Download Minion!*
  - *http://minion.sourceforge.net*
    - *Send bug reports!*
- *Minion's simplicity means (sometimes) easy to extend*
  - *Watched Literals (more to come).*
  - *New symmetry breaking methods.*