EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

SArTagnan

A parallel portfolio SAT solver with lockless physical clause sharing

Stephan Kottler and Michael Kaufmann

University of Tuebingen

Outline

1. Motivation
2. Parallel Solving
   - Physical clause sharing
   - Communication of threads
3. Portfolio Solving
4. Summary

STATE-OF-THE-ART SOLVING

CDCL

- partial assignment
- decisions based on variable activity
- conflict analysis
- restarts

STATE-OF-THE-ART SOLVING

CDCL

- partial assignment
- decisions based on variable activity
- conflict analysis
- restarts

DMRP

- complete assignment (ref. point)
- decisions based on unsat clauses
- slower than CDCL but less decisions

KINDS OF PARALLELISATION

- Division of search space (guiding path)
- Portfolio solving

CLAUSE SHARING

Most solvers: copy learnt clauses of other threads

KINDS OF PARALLELISATION

- Division of search space (guiding path)
- Portfolio solving

CLAUSE SHARING

Most solvers: copy learnt clauses of other threads

MAIN AIM

- Real / physical sharing of data
- Threads work together

  $\Rightarrow$ Any thread may benefit from strengthened clause
- No use of OS locks

BASIC CONCEPT TO SHARE DATA

- Shared data / objects contain *user-mask*
  - user-mask initialised by creating thread
  - Any thread can release object (clear bit)
  - Last thread destructs object
- Compare and Swap operation

BASIC CONCEPT TO SHARE DATA

- Shared data / objects contain *user-mask*
  - user-mask initialised by creating thread
  - Any thread can release object (clear bit)
  - Last thread destructs object
- Compare and Swap operation

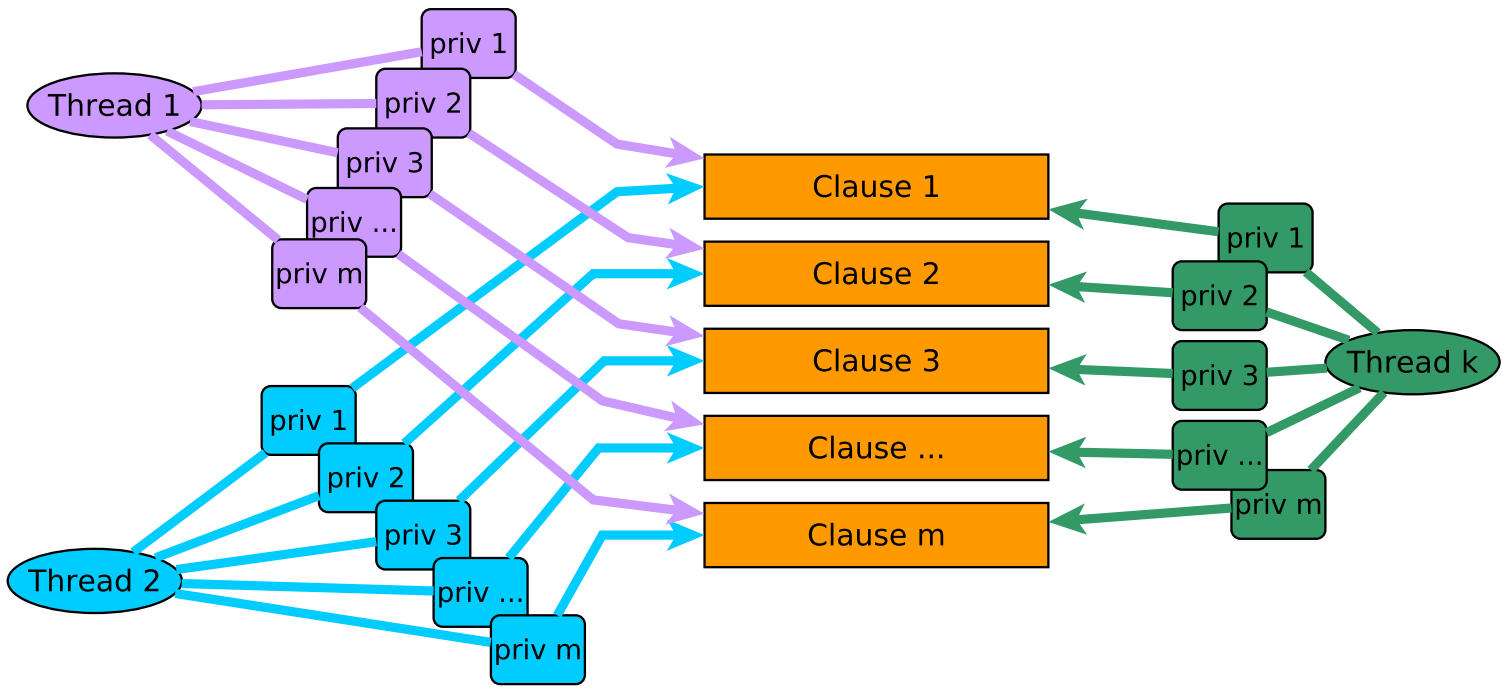
```
SharedObj{
  umask;
  ...
}
```

```
void release(SharedObj obj, tId){
  do{
    old = obj.umask;
    new = clear bit 'tId' in old;
  }while(!exchange(obj.umask,old,new));
  if(new == 0) destruct(obj);
}
```

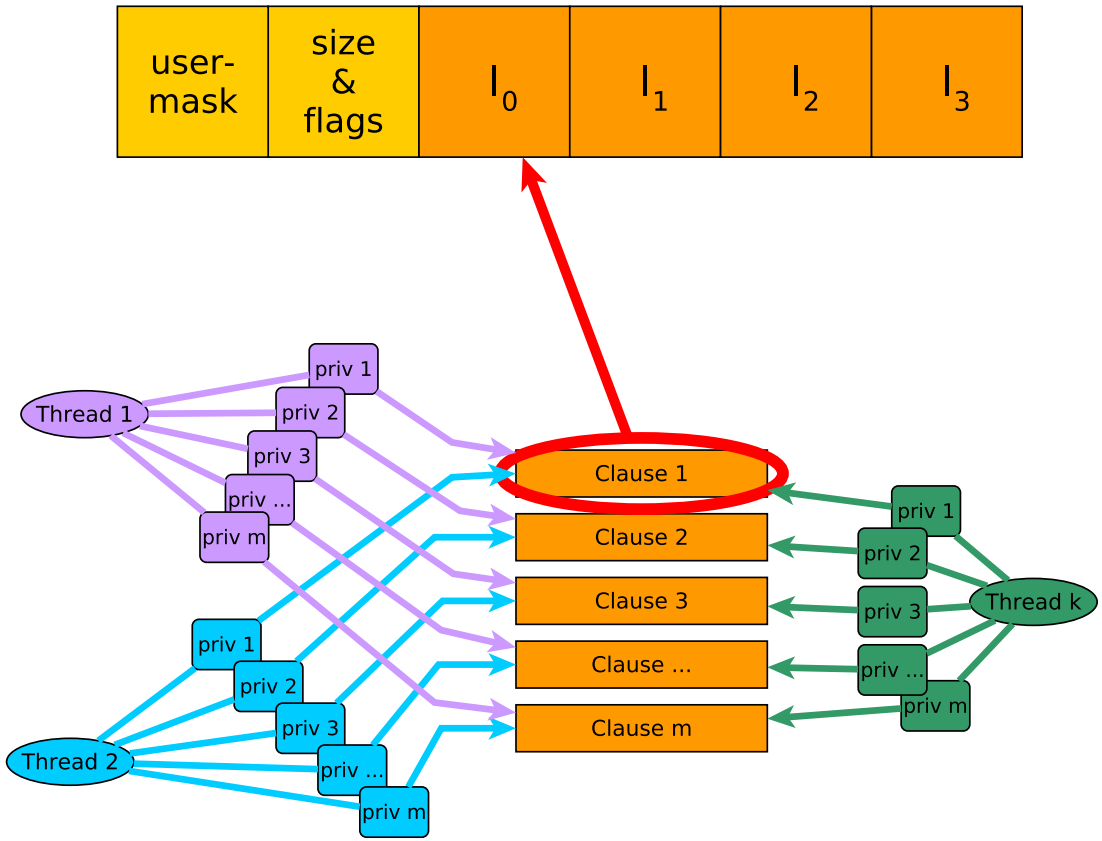
PHYSICAL SHARING OF CLAUSES

Have one instance of a clause

Indirection to access clause (thread private data)



SHARED CLAUSE' ARCHITECTURE



REFERENCING CLAUSES IN CDCL

- Unit propagation
- Conflict analysis
- Garbage collection

REFERENCING CLAUSES IN CDCL

- Unit propagation
- Conflict analysis
- Garbage collection

OBSERVATION

Whenever a clause is referenced at least one watched literal is known

REFERENCING CLAUSES IN CDCL

- Unit propagation
- Conflict analysis
- Garbage collection

OBSERVATION

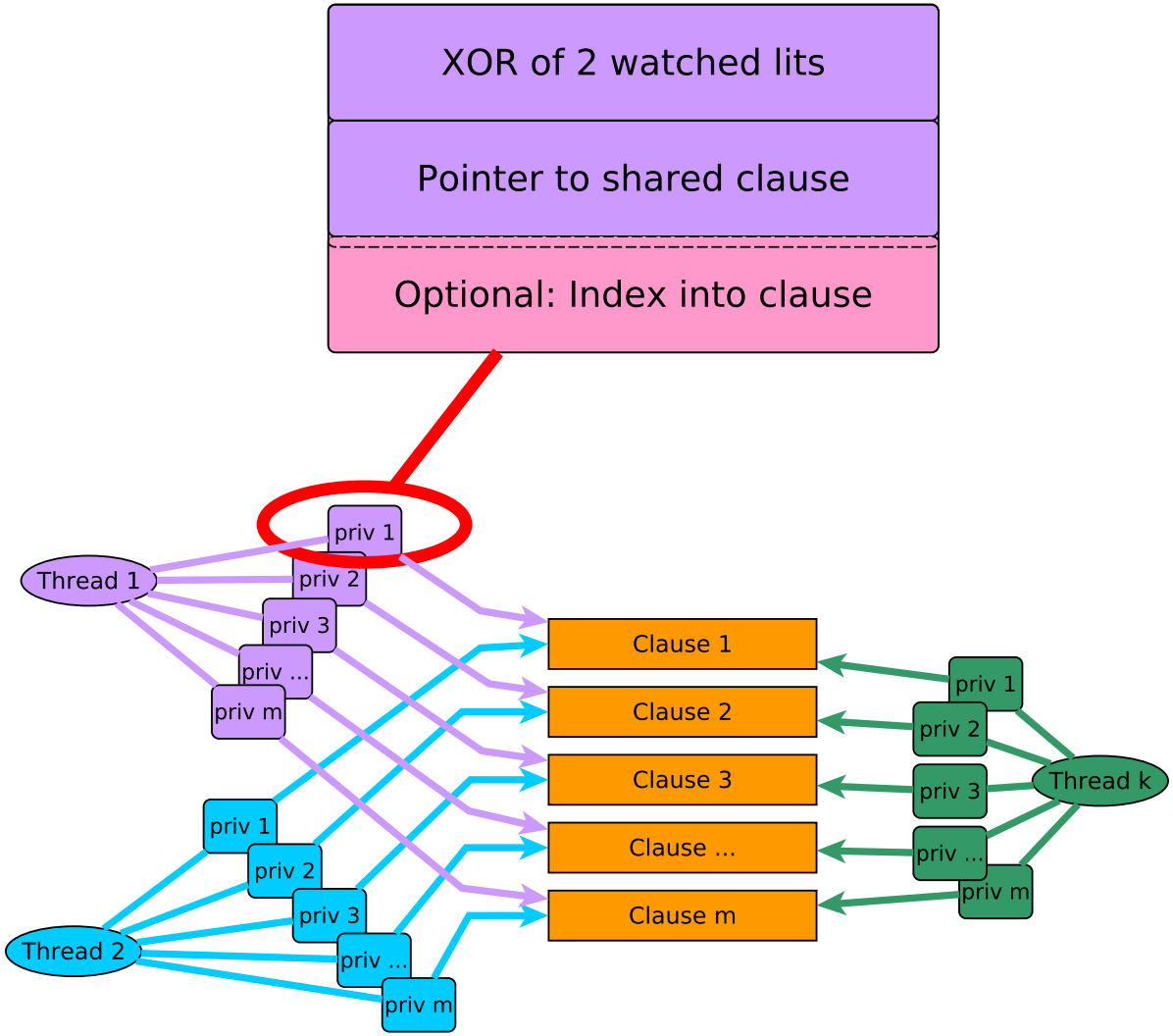
Whenever a clause is referenced at least one watched literal is known

LEMMA

Two watched literals l_i, l_j can be stored by one value: $C_w = l_i$ xor l_j.

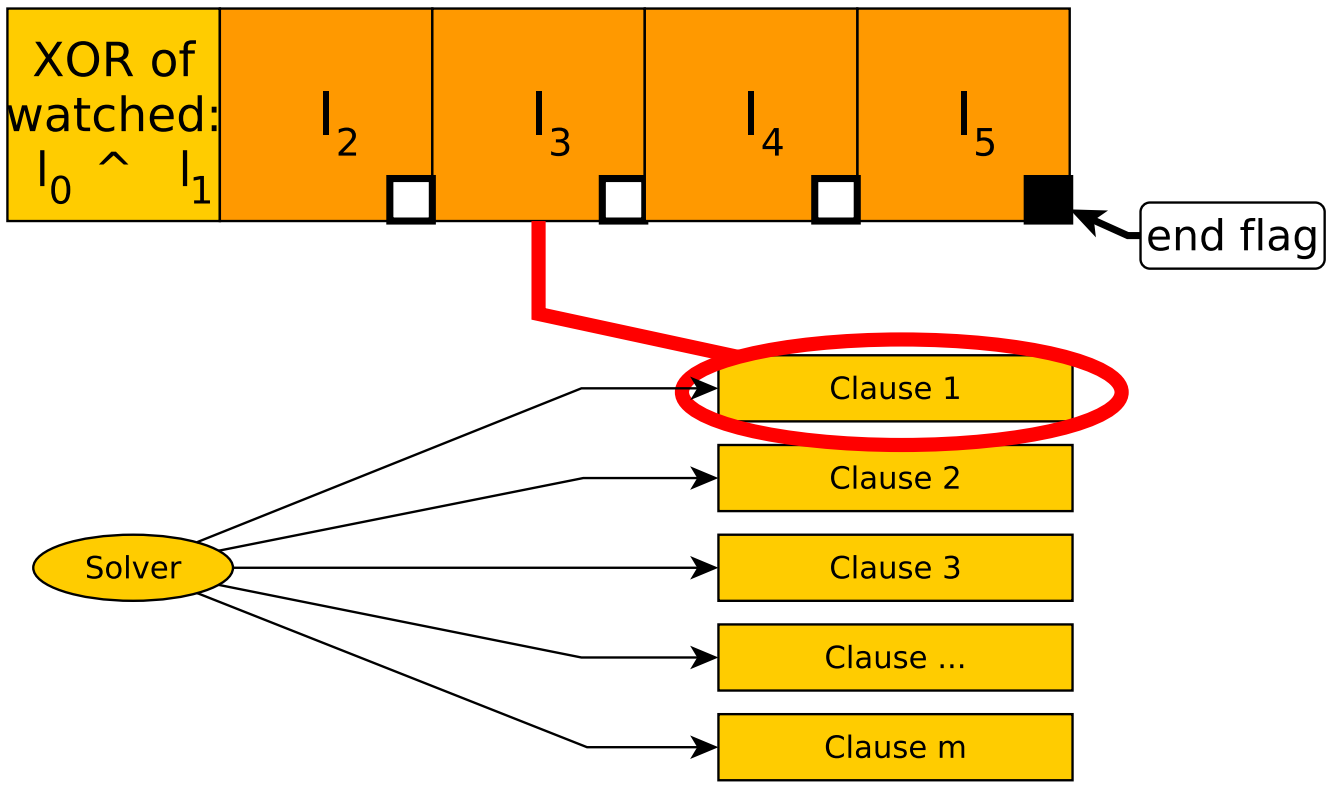
(l_i xor $C_w \rightarrow l_j$)

SHARED CLAUSE' ARCHITECTURE II



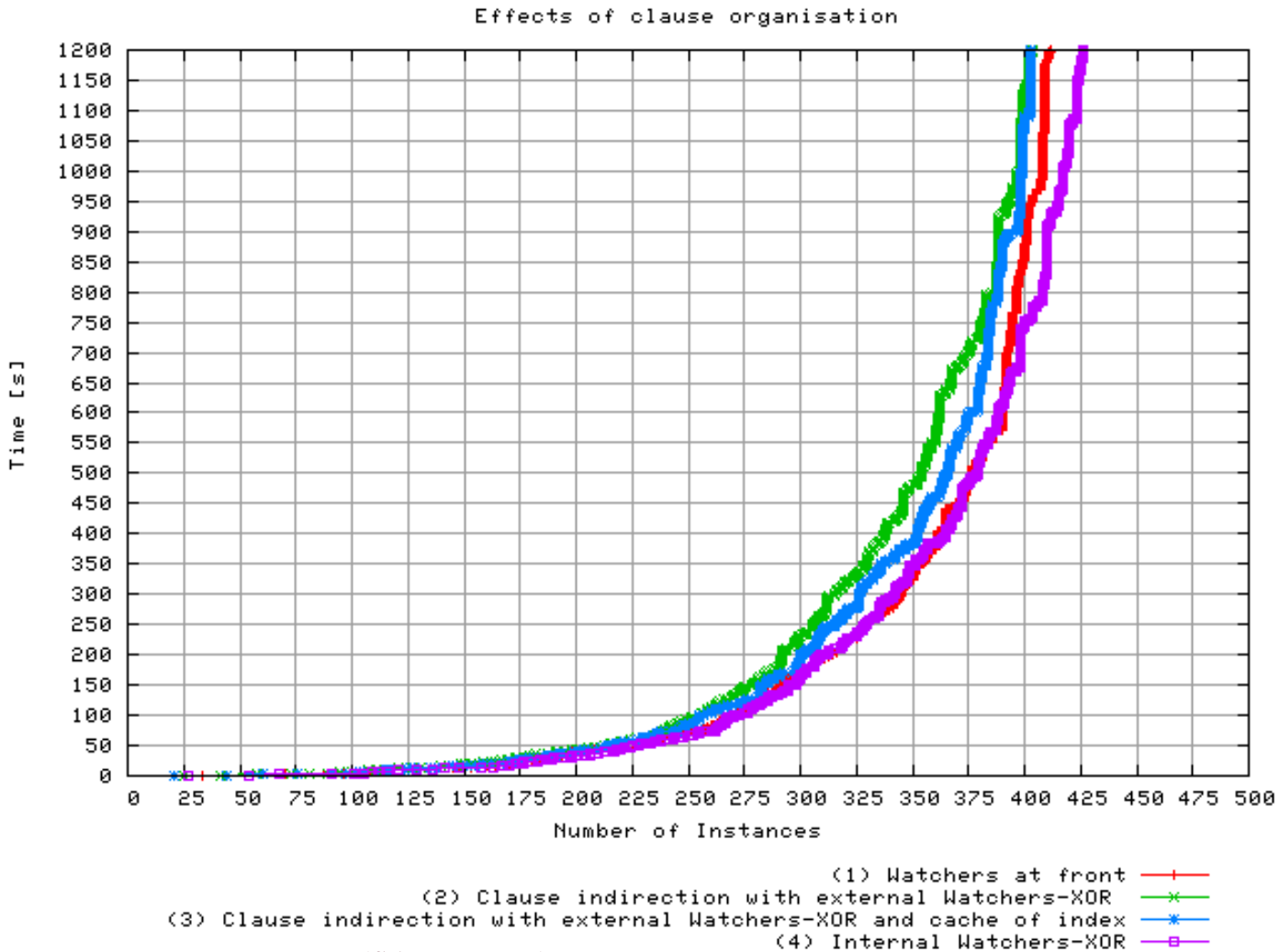
DIGRESS TO SEQUENTIAL SOLVERS

- Order of literals may be modified
- Store clause C with $|C| - 1$ integers



OVERHEAD OF CLAUSE ORGANISATION

Comparison of different implementations with single thread



COMMUNICATION OF THREADS

Message queues used to send . . .

- a new clause (may be new version)
- notification on variable elimination
- variable replacement
- heuristic information

COMMUNICATION OF THREADS

Message queues used to send . . .

- a new clause (may be new version)
- notification on variable elimination
- variable replacement
- heuristic information

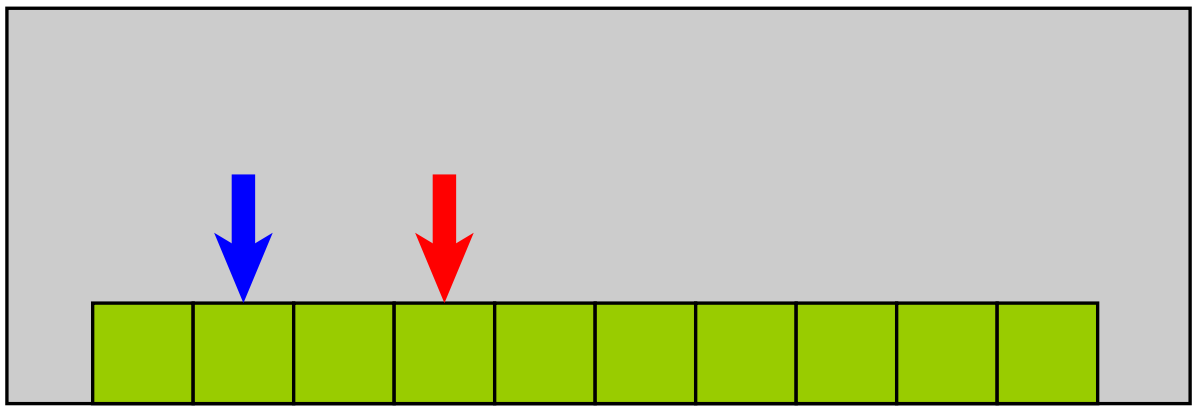
! Messages not only for heuristics

! Keep order of messages

! No OS locks

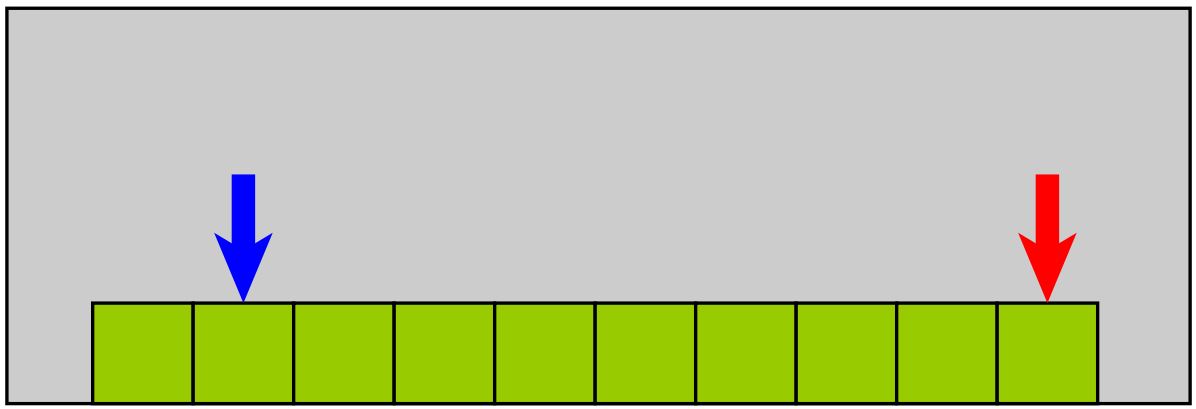
LOCKLESS QUEUES

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead



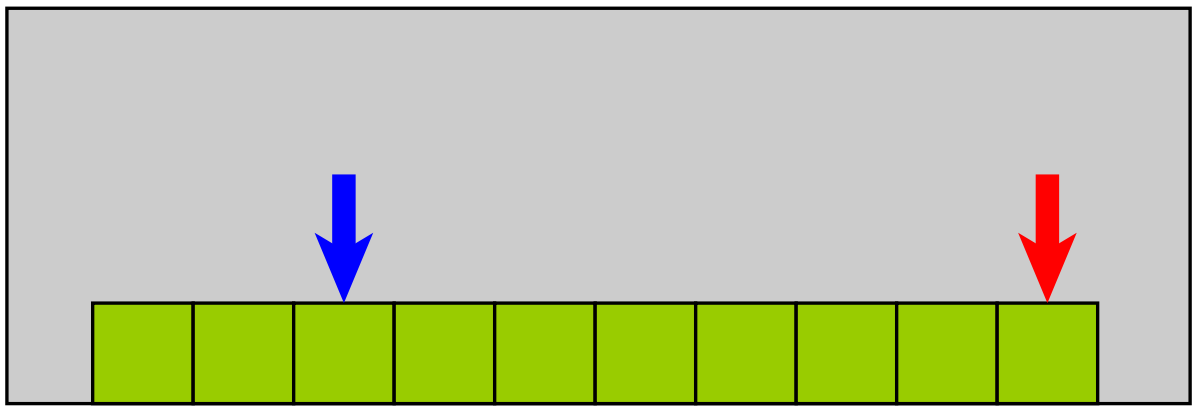
LOCKLESS QUEUES

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead



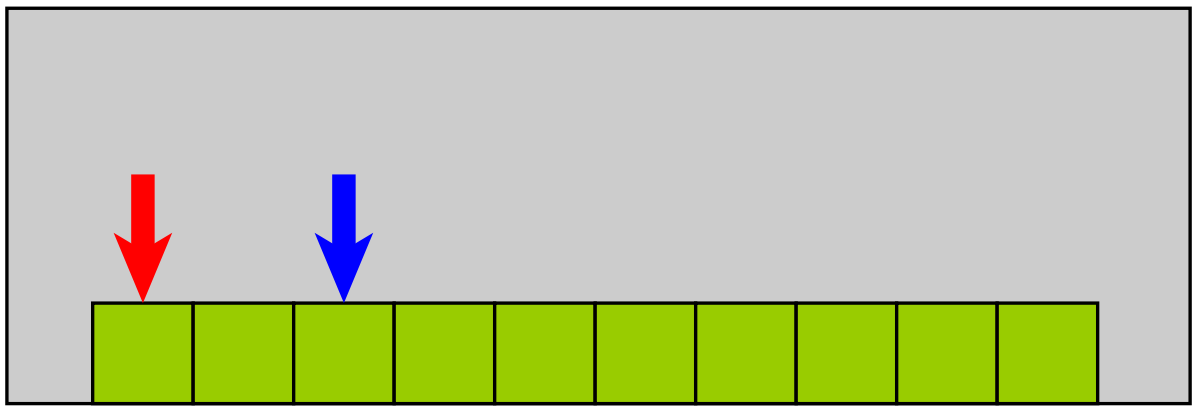
LOCKLESS QUEUES

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead



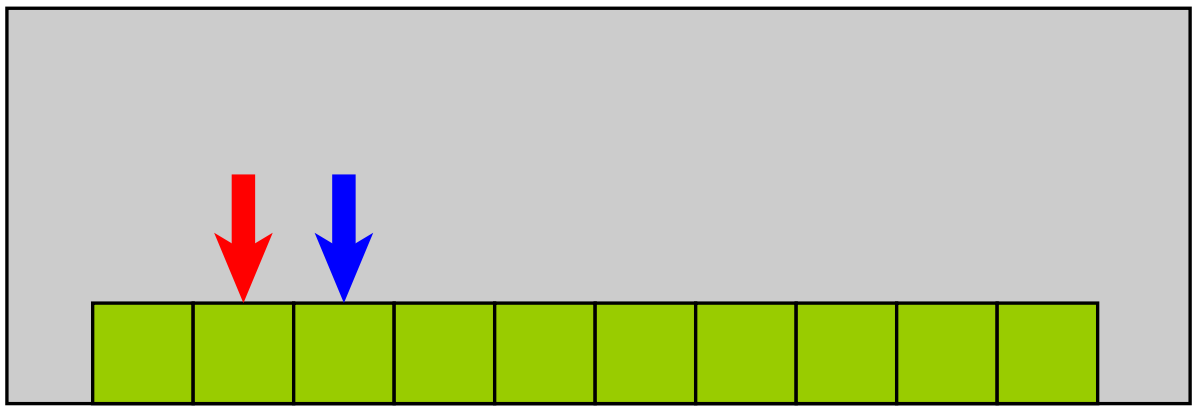
LOCKLESS QUEUES

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead



LOCKLESS QUEUES

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead



DYNAMIC SIZE

- Write operation may fail
- Write operation may overwrite unseen data

IDEA

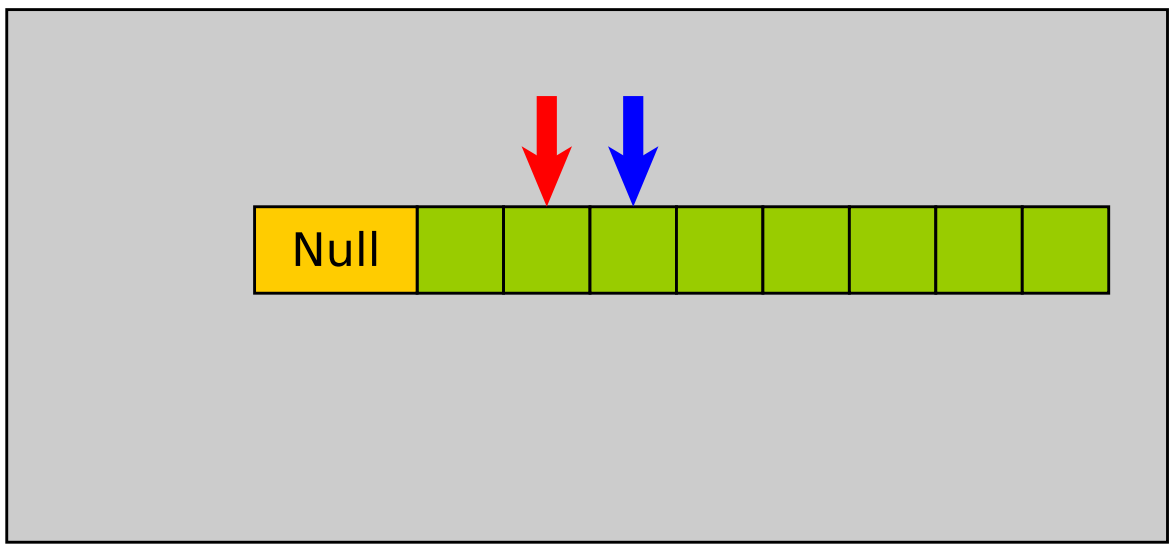
Queue links to available update

DYNAMIC SIZE

- Write operation may fail
- Write operation may overwrite unseen data

IDEA

Queue links to available update

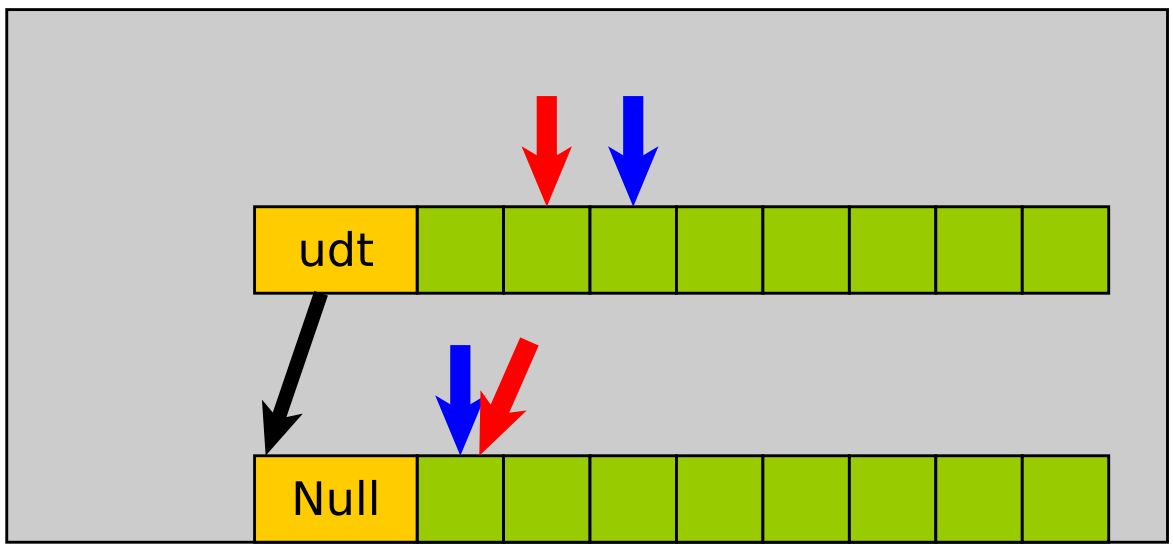


DYNAMIC SIZE

- Write operation may fail
- Write operation may overwrite unseen data

IDEA

Queue links to available update

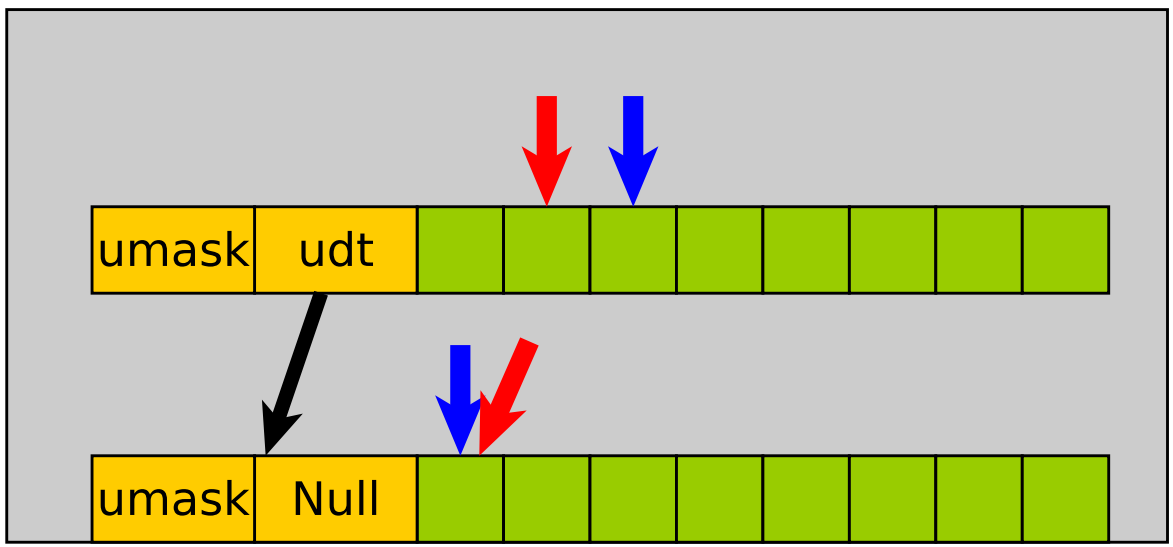


DYNAMIC SIZE

- Write operation may fail
- Write operation may overwrite unseen data

IDEA

Queue links to available update



Queue has several reading threads!

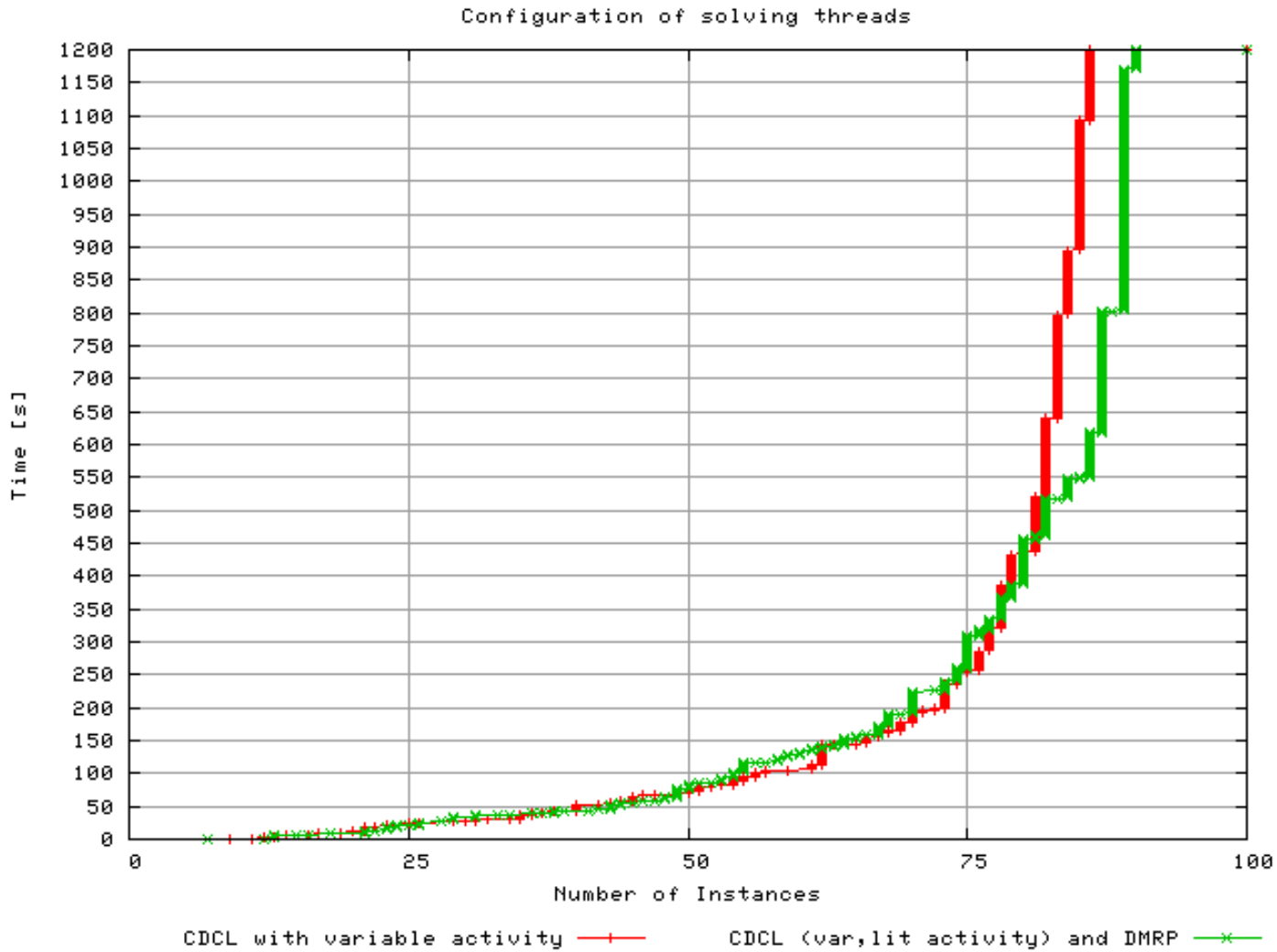
DIFFERENT STRATEGIES

- 6 of 8 threads apply CDCL (different settings)
  - Activity of Variables / Literals
  - Glucose / Static / Geometric / Luby restart schemes
- Dedicated simplification thread
  - satElite like simplification
  - Asymmetric branching / vivification
  - SCC computation and removal of redundant binaries
- Connect work - DMRP
  - At each restart: init reference point to set each variable to predominant value among all threads
  - Learn 'interesting' clauses

TAKE ADVANTAGE OF SHARING

- Simplification of clause DB is shared immediately
- On-the-fly clause subsumption done by any thread
  $\Rightarrow$ Any thread may benefit
- Lazy hyper binary resolution

DMRP & LITERALS ACTIVITY



SUMMARY

- Physical clause sharing
- XOR idea to store watched literals

  parallel and sequential solvers
- Communication without OS locks

CHALLENGES

! Has to run in parallel
  - Difficult to measure speedup
  - Computation time

? Logging without influencing course of events

CLAUSE COPYING STILL FASTER

	plingeling	ManySAT 1.5	ManySAT 1.1	SArTagnan	antom
#solved (in 1st run):	78	75	72	70	67
#solved SAT/UNSAT (in first run):	23/55	19/56	18/54	18/52	19/48
average time per solved instance:	97.7	143.9	124.0	86.5	83.1
#solved in 2nd run:	79	74	73	70	65
#solved in 3rd run:	79	74	71	72	68
#solved in at least one out of three runs:	80	78	76	76	69
rank:	1	2	3	4	5

Thank you for your attention!