SAT II

Watch 2-Literal Refinement

- For each clause select arbitrarily two literals and connect only those to the assignment.
- When an assignment literal is set, if the clause is true, ignore. If it becomes false, search for another undefined literal. If there is no, propagate the literal or detect contradiction.
- When backtracking don't do anything.

Benefits

- Only two literals of a clause are connected.
- No updates for backtracking needed.

Memory Management Motivation

Why Design Your Own Memory Management?

- Debugging: memory Leaks, runtime behaviour
- Performance: `free()`, `malloc()` typically bad at many small blocks
- Limit Memory Consumption: program has to to take care
- Push local behaviour

Basic Memory Management Ideas

Administration

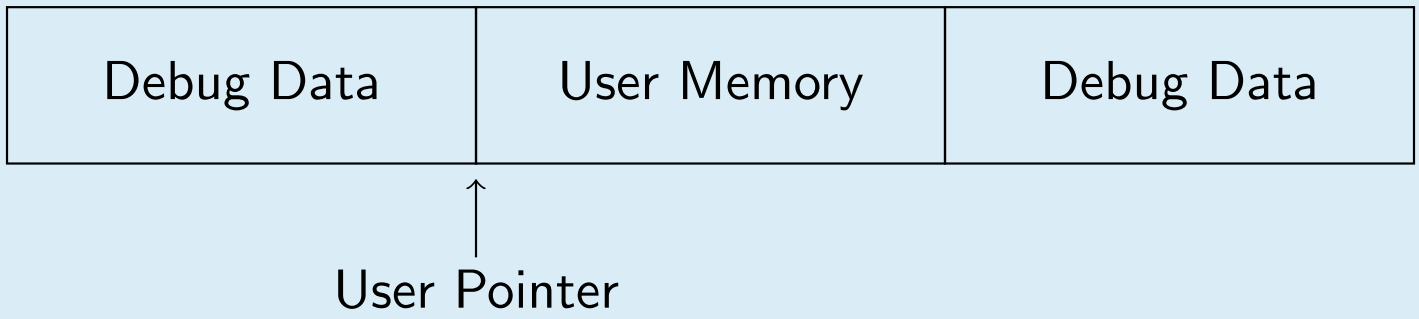
- Allocate memory for small blocks in big chunks: pages
- Map memory calls for big blocks to `free()`, `malloc()`
- Support page sharing small blocks of different sizes

Debugging: Check for

- Leaks: support allocation link
- Read/Write to freed/not properly allocated memory
- Read/Write over block size bounds
- Size/Type confusion of blocks: hence `memory_free()` gets additional size argument

Memory Blocks: Debug Mode

Memory Block Layout

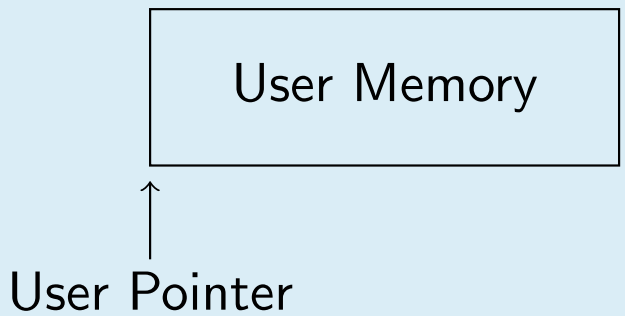


Debug Data

- Block Size
- Block Status
- Block Read
- Block Write
- Block Allocation Link
- Block Overwrite Buffer

Memory Blocks: Shipping Mode

Memory Block Layout



Memory Management Control Parameters

- No Memory Management
- Allocation Page Size
- Small Block Max Size
- Small Block Min Size
- Number of Shared Page Sizes
- Size of Debug Space
- Alignment Size

Memory Block Administration

Small Blocks: Array Of Pointers to Resource Cells

```
typedef struct MEMORY_RESOURCEHELP {
  POINTER free;          /* next free block in list  */
  POINTER next;          /* next fresh block         */
  POINTER page;          /* head of page list        */
  POINTER end_of_page;   /* end of current page      */
  int     total_size;    /* block size incl. debug data */
  int     aligned_size;  /* block size excl. debug data */
  int     offset;        /* offset last usable block    */
} MEMORY_RESOURCE;
```

Big Blocks: Global Variable to Doubly Linked List

```
typedef struct MEMORY_BIGBLOCKHEADERHELP {
  struct MEMORY_BIGBLOCKHEADERHELP * previous, * next;
} MEMORY_BIGBLOCKHEADERNODE, * MEMORY_BIGBLOCKHEADER;
```