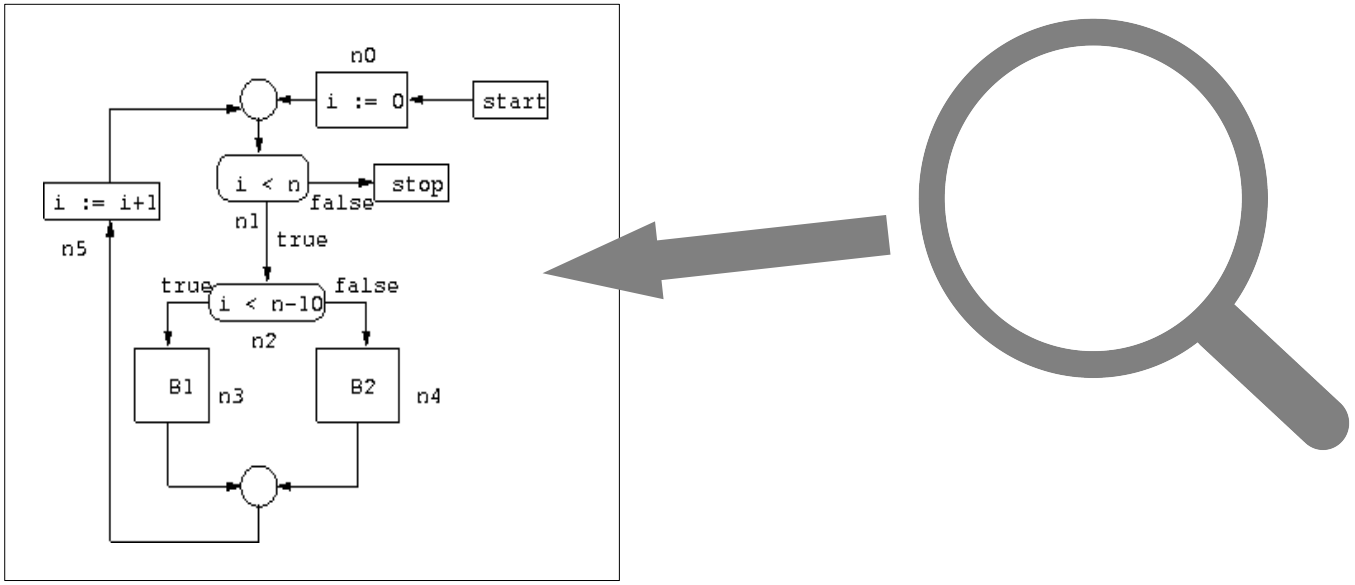
Dynamic Program Analysis
Lecture 11: Memcheck



Memcheck

A Valgrind tool to detect potential memory-related errors in C/C++ programs

Can detect the following problems:

- Accessing memory you shouldn't, like over/underrunning heap blocks, overrunning the top of the stack, accessing memory that has been freed
- Using undefined values, (not initialised, or derived from undefined values)
- Incorrect freeing of heap memory (double-freeing heap blocks, mismatched use of `malloc` versus `free`)
- Overlapping `src` and `dst` pointers in `memcpy`, and related functions
- Memory leaks

Limitations

Memcheck may miss errors, since it is based on testing (unsafe analysis)

May also consider correct memory accesses as incorrect, in rare cases (false positives)

However, an error report is a very strong indication that there is something wrong

Cost of Running Memcheck

Running memcheck is quite costly:

- Slowdown of execution is typically 10-50 times
- Code to be analyzed is typically blown up 12-18 times

Due to the extensive instrumentation needed to check memory references

The price to pay for the analysis. There seems to be no easy way around this

Memcheck output

Reconsider this example of an illegal memory access report:

```
==25832== Invalid read of size 4
==25832==    at 0x8048724: BandMatrix::ReSize(int, int, int) (bogon.cpp:45)
==25832==    by 0x80487AF: main (bogon.cpp:66)
==25832== Address 0xBFFFF74C is not stack'd, malloc'd or free'd
```

The illegal address `0xBFFFF74C` is read

It is read from address `0x8048724`, in the function `BandMatrix::ReSize`

This function is in turn called from address `0x80487AF` in `main`

The address read is deemed to be neither a valid stack address nor in a (active or deallocated) heap object

Use of uninitialized value

Consider this piece of code:

```
int main()
{
  int x;
  printf ("x = %d\n", x);
}
```

Memcheck will give the following error message:

```
Conditional jump or move depends on uninitialised value(s)
  at 0x402DFA94: _IO_vfprintf (_itoa.h:49)
  by 0x402E8476: _IO_printf (printf.c:36)
  by 0x8048472: main (tests/manuel1.c:8)
```

The problem is the attempt to print the value of `x`

Illegal Freeing of Memory

An example of illegal deallocation with `free`:

```
Invalid free()
  at 0x4004FFDF: free (vg_clientmalloc.c:577)
  by 0x80484C7: main (tests/doublefree.c:10)
 Address 0x3807F7B4 is 0 bytes inside a block of size 177 free'd
  at 0x4004FFDF: free (vg_clientmalloc.c:577)
  by 0x80484C7: main (tests/doublefree.c:10)
```

Due to that the object already has been deallocated

Memory Leaks

Memory leaks are caused by heap objects not being properly deallocated

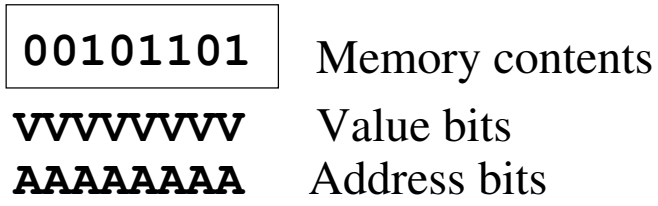
Can cause a system to eventually run out of memory

Memcheck will examine the heap after the run, classify the remaining objects there, and print a report. Example:

```
LEAK SUMMARY:
 definitely lost: 48 bytes in 3 blocks.
 indirectly lost: 32 bytes in 2 blocks.
   possibly lost: 96 bytes in 6 blocks.
 still reachable: 64 bytes in 4 blocks.
     suppressed: 0 bytes in 0 blocks.
```

How does Memcheck Work?

Conceptually, for every bit in memory, memcheck adds a *value bit* V and an *address bit* A:



The value bit is set if the corresponding memory bit contains a *valid value*

The address bit is set if the corresponding memory bit is at an address that is *legitimate to access*

(Note that we need to differ between these bits. Consider, e.g., a newly allocated block in the heap, where the values are not yet initialized)

(In reality, the representation is compressed. Typical overhead is 25%)

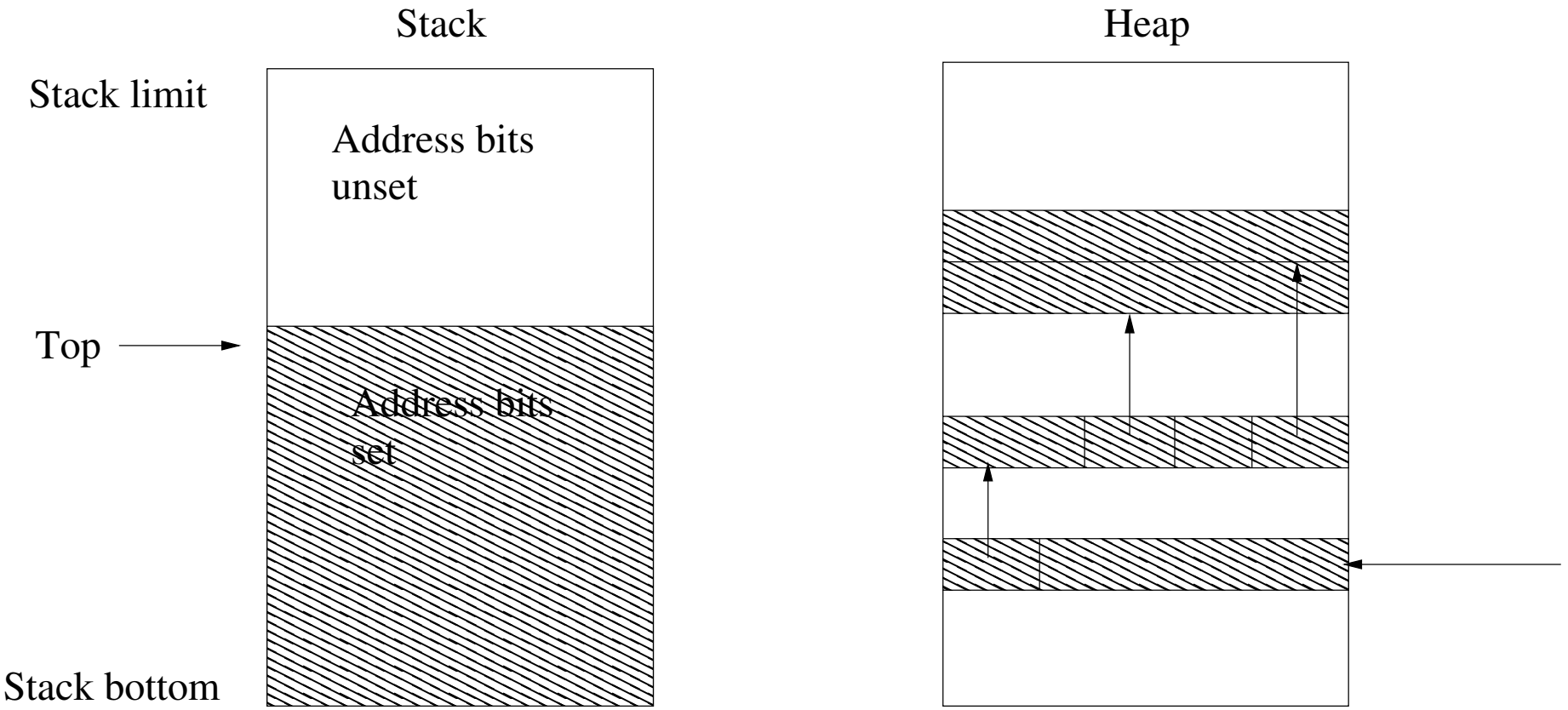
How to Set/Reset Value and Address Bits

Value bits are set when the corresponding memory is written, or initialized, with valid values

For all other locations, the value bit will be unset

When a value is copied/moved, its value bits will follow

Address bits will be set when the memory area (on stack or heap) is allocated, and cleared when deallocated. For global data areas the address bits are statically set


Stack
Heap
Stack limit
Address bits unset
Top
Address bits set
Stack bottom

How Memcheck Uses the Bits

Easy to check memory errors: just check if some value or address bit is unset for a memory access!

The address bits are checked immediately

Illegal values, however, can be read and copied around

Memcheck will report an error only when the illegal value is used as follows:

- to generate a memory address,
- to affect the control flow, and
- as parameter to a system call

(These cases can affect the observable behavior. Including more cases could yield many false alarms)

Summing Up

Valgrind provides a popular framework for dynamic analysis of C/C++ code

Especially memcheck is much used

Free software, unix only

Simple to run if you're accustomed to unix environment and command-line processing

Running times may be a problem. Slows down the verification phase

Still much better than unsystematic testing for memory errors