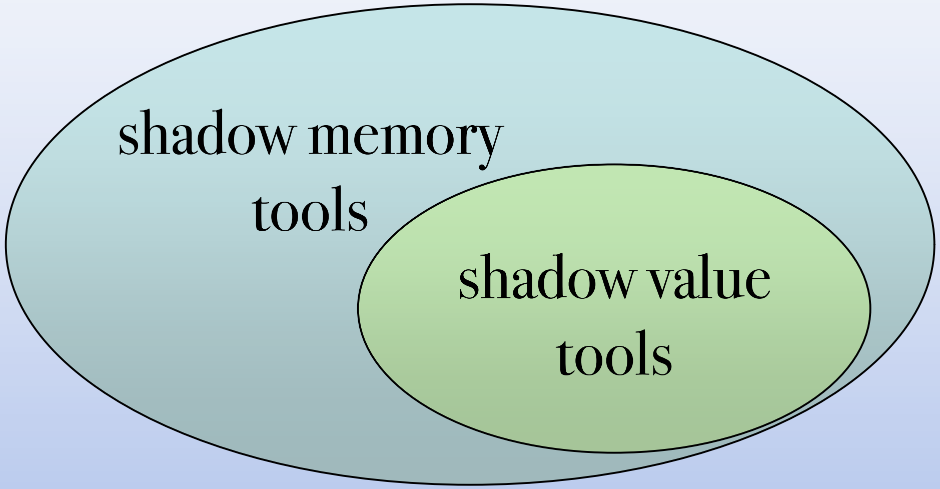
How to Shadow Every Byte of Memory Used by a Program

Nicholas Nethercote – National ICT Australia

Julian Seward – OpenWorks LLP

Shadow memory tools

- Shadow every byte of memory with another value that describes it



shadow memory tools

shadow value tools

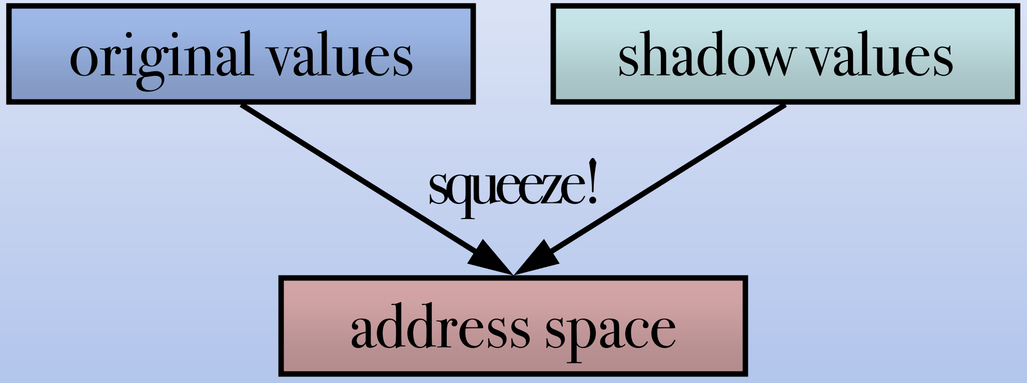
- This talk:
  - Why shadow memory is useful
  - How to implement it well

Examples

	Tool(s)	Shadow memory helps find…
bugs	**Memcheck**, Purify	**Memory errors**
	Eraser, DRD, Helgrind, etc.	Data races
	Hobbes	Run-time type errors
	Annelid	Array bounds violations
security	TaintCheck, LIFT, TaintTrace	Uses of untrusted values
	“Secret tracker”	Leaked secrets
properties	Redux	Dynamic dataflow graphs
	DynCompB	Invariants
	pinSEL	System call side-effects

Shadow memory is difficult

- Performance
  - Lots of extra state, many operations instrumented
- Robustness



original values

shadow values

squeeze!

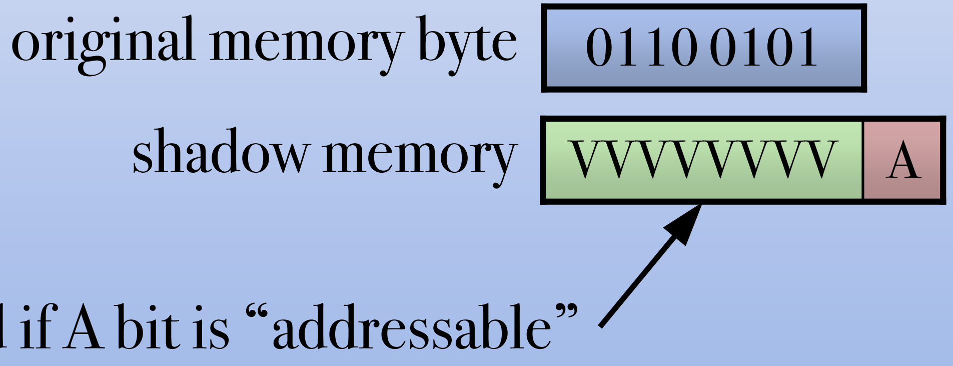
address space

- Trade-offs must be made

An example tool: Memcheck

Memcheck

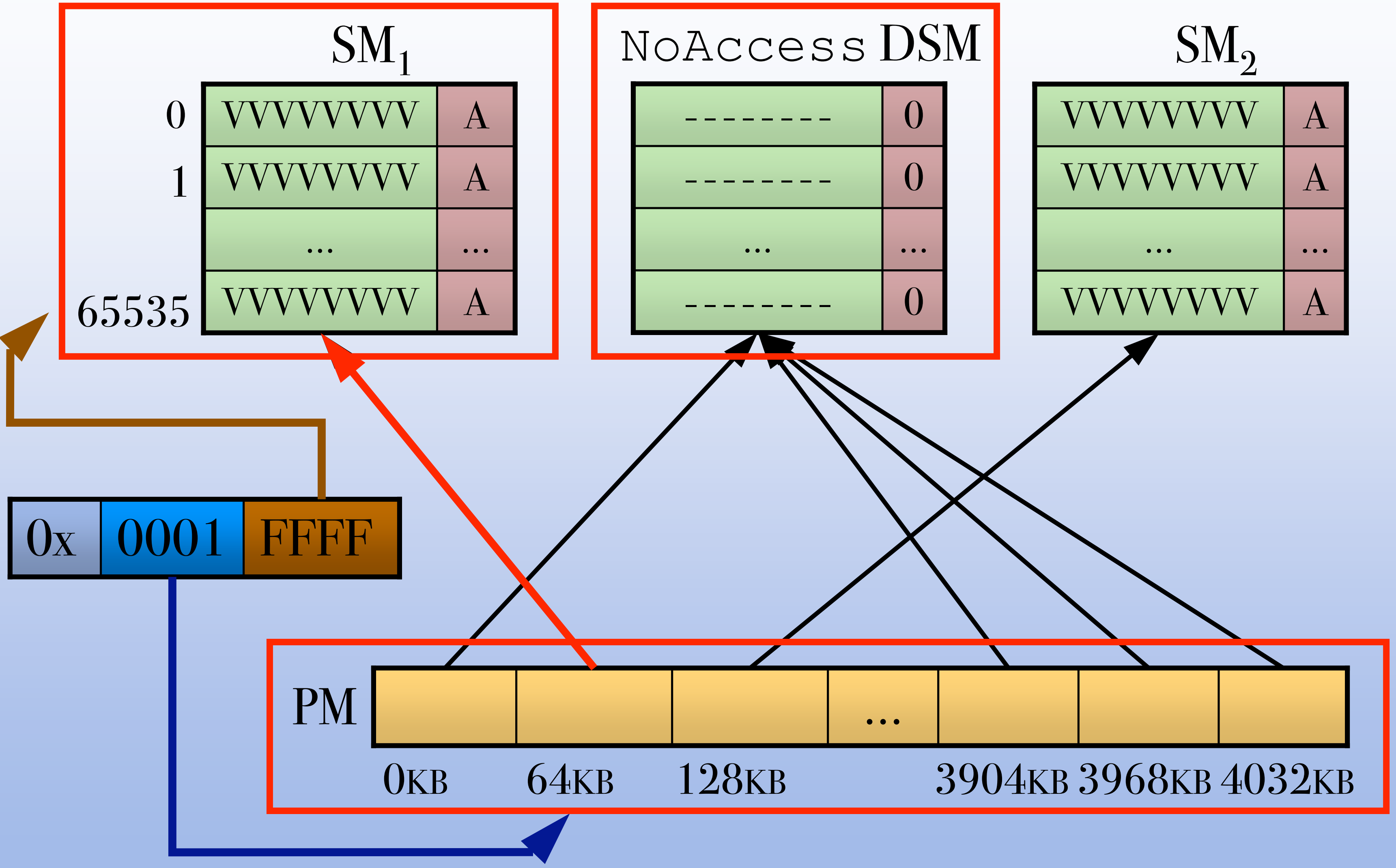
- Three kinds of information:
  - A (“addressability”) bits: 1 bit / memory byte
  - V (“validity”) bits: 1 bit / register bit, 1 bit / memory bit
  - Heap blocks: location, size, allocation function
- Memory information:



V bits only used if A bit is “addressable”

A simple implementation

Basics (I)



SM$_1$

0	VVVVVVVV	A
1	VVVVVVVV	A
	...	...
65535	VVVVVVVV	A

NoAccess DSM

--------	0
--------	0
...	...
--------	0

SM$_2$

VVVVVVVV	A
VVVVVVVV	A
...	...
VVVVVVVV	A

0x	0001	FFFF

Basics (II)

- Multi-byte shadow accesses:
  - Combine multiple single-byte accesses
  - Complain if any unaddressable bytes accessed
  - Values loaded from unaddressable bytes marked as defined
- Range-setting (`set_range`)
  - Loop over many bytes, one at a time
- Range-checking
  - E.g.: `write(fd, buf, n)` -- check `n` bytes in `buf`
- Slow-down: 209.6x

Complications

- Corruption of shadow memory
  - Possible with a buggy program
  - Originally used x86 segmentation, but not portable
  - Keep original and shadow memory far apart, and pray

- 64-bit machines
  - Three- or four-level structure would be slow
  - Two level structure extended to handle 32GB
  - Slow auxiliary table for memory beyond 32GB
  - Better solution is an open research question

Four optimisations

#1: Faster loads and stores

- Multi-byte loads/stores are very common
  - N separate lookups accesses is silly (where N = 2, 4, or 8)
- If access is aligned, fully addressable
  - Extract/write V bits for N shadow bytes at once
  - Else fall back to slow case: 1 in a 1000 or less
- Slow-down: 56.2x
  - 3.73x faster

#2: Faster range-setting

- Range-setting large areas is common
  - Vectorise `set_range`
  - 8-byte stride works well
- Replacing whole SMs
  - If marking a 64KB chunk as `NoAccess`, replace the SM with the `NoAccess` DSM
  - Add `Defined` and `Undefined` DSMs
  - Large read-only code sections covered by `Defined` DSM
- Slow-down: 34.7x
  - 1.62x faster, 1.97x smaller

#3: Faster SP updates

- Stack pointer (SP) updates are very common
- Inc/dec size often small, statically known
  - E.g. 4, 8, 12, 16, 32 bytes
- More specialised range-setting functions
  - Unrolled versions of `set_range()`
- Slow-down: 27.2x
  - 1.28x faster

#4: Compressed V bits

- Partially-defined bytes (PDBs) are rare
  - Memory: 1 A bit + 8 V bits → 2 VA bits
  - Four states: `NoAccess`, `Undefined`, `Defined`, `PartDefined`
  - Full V bits for PDBs in secondary V bits table
  - Registers unchanged -- still 8 V bits per byte
- Slow-down: 23.4x
  - 4.29x smaller, 1.16x faster
- Obvious in hindsight, but took 3 years to identify

Discussion

- Optimising principles:
  - Start with a simple implementation
  - Make the common cases fast
  - Exploit redundancy to reduce data sizes

- Novelty?
  - First detailed description of Memcheck’s shadow memory
  - First detailed description of a two-level table version
  - First detailed evaluation of shadow memory
  - Compressed V bits

Evaluation

Robustness

- Two-level table is very flexible
  - Small shadow memory chunks, each can go anywhere
- Earlier versions required large contiguous regions
  - Some programs require access to upper address space
  - Some Linux kernels have trouble mmap’ing large regions
  - Big problems with Mac OS X, AIX, other OSes
- Memcheck is robust
  - Standard Linux C and C++ development tool
  - Official: Linux, AIX; experimental: Mac OS X, FreeBSD

SPEC 2000 Performance

Tool	Slow-down	Relative improvement
No instrumentation	4.3x	
Simple Memcheck	209.6x	
+ faster loads/stores	56.2x	3.73x faster
+ faster range-setting	34.7x	1.62x faster, 1.97x smaller
+ faster SP updates	27.2x	1.28x faster
+ compressed V bits	23.4x	1.16x faster, 4.29x smaller
Overall improvement		8.9x faster, 8.5x smaller

- Shadow memory causes about half of Memcheck's overhead

Performance observations

- Performance is a traditional research obsession

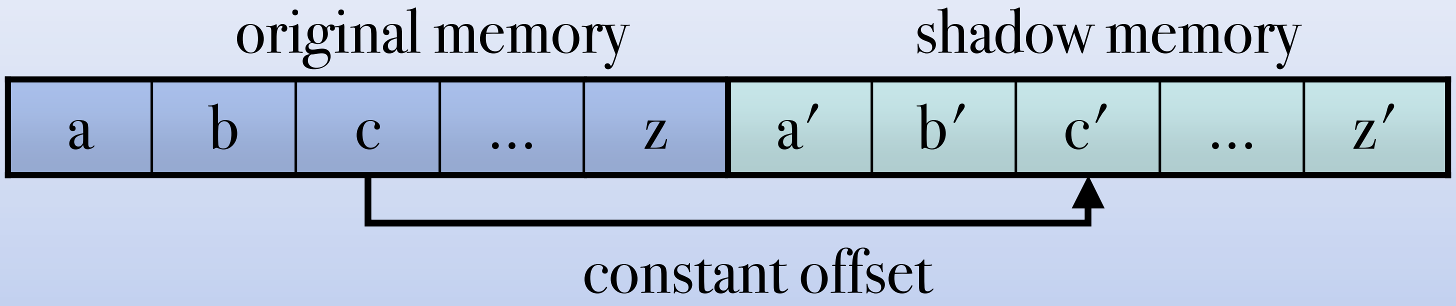
  “The *subjective* issues are important – ease of use and robustness, but performance is the item which would be most interesting for the audience.” (my emphasis)

- Users: slowness is #1 survey complaint
  - But most user emails are about bugs or interpreting results
  - Zero preparation is a big win

- Cost/benefit
  - People will use slow tools if they are sufficiently useful

Alternative implementation

- “Half-and-half”
  - Used by Hobbes, TaintTrace, (with variation) LIFT



original memory

shadow memory

a b c … z a′ b′ c′ … z′

constant offset

- Compared to two-level table
  - Faster
  - Not robust enough for our purposes

If you remember nothing else…

Take-home messages

- Shadow memory is powerful
- Shadow memory can be implemented well
- Implementations require trade-offs

www.valgrind.org