

Quasi-Infinite Heaps

Vitaliy Lvin, Emery Berger

University of Massachusetts Amherst

*Joint work with Ben Zorn
(Microsoft Research)*



Infinite Heap Semantics

- Normally: memory errors $\Rightarrow \perp$
- Consider **infinite-heap** allocator:
 - All news *fresh*;
ignore **delete**
 - No dangling pointers, invalid frees, double frees
 - Every object **infinitely large**
 - No buffer overflows, data overwrites
- Transparent to correct program
- **“Erroneous” programs sound**



Approximating Infinite Heaps

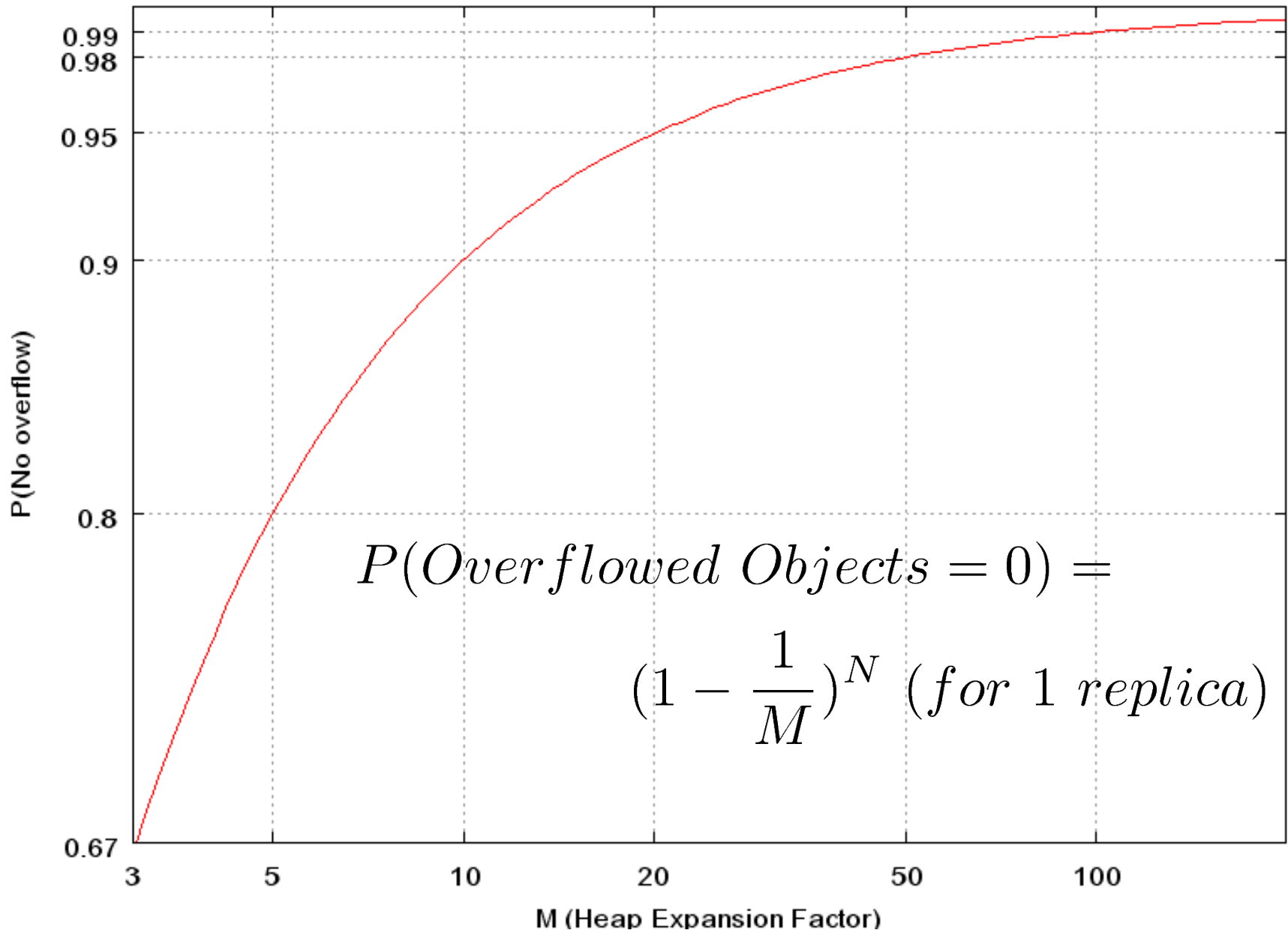
■ DieHard:

- Approximate ∞ -heaps with M-heaps (e.g., 2)
- Increases odds of **benign** errors
- **Probabilistic** memory safety
 - i.e., $P(\text{no error}) \geq n$
- **Protection increases as M increases**



Probabilistic Memory Safety

$P(\text{Overflowed Objects}=0)$ for overflows affecting 1 object



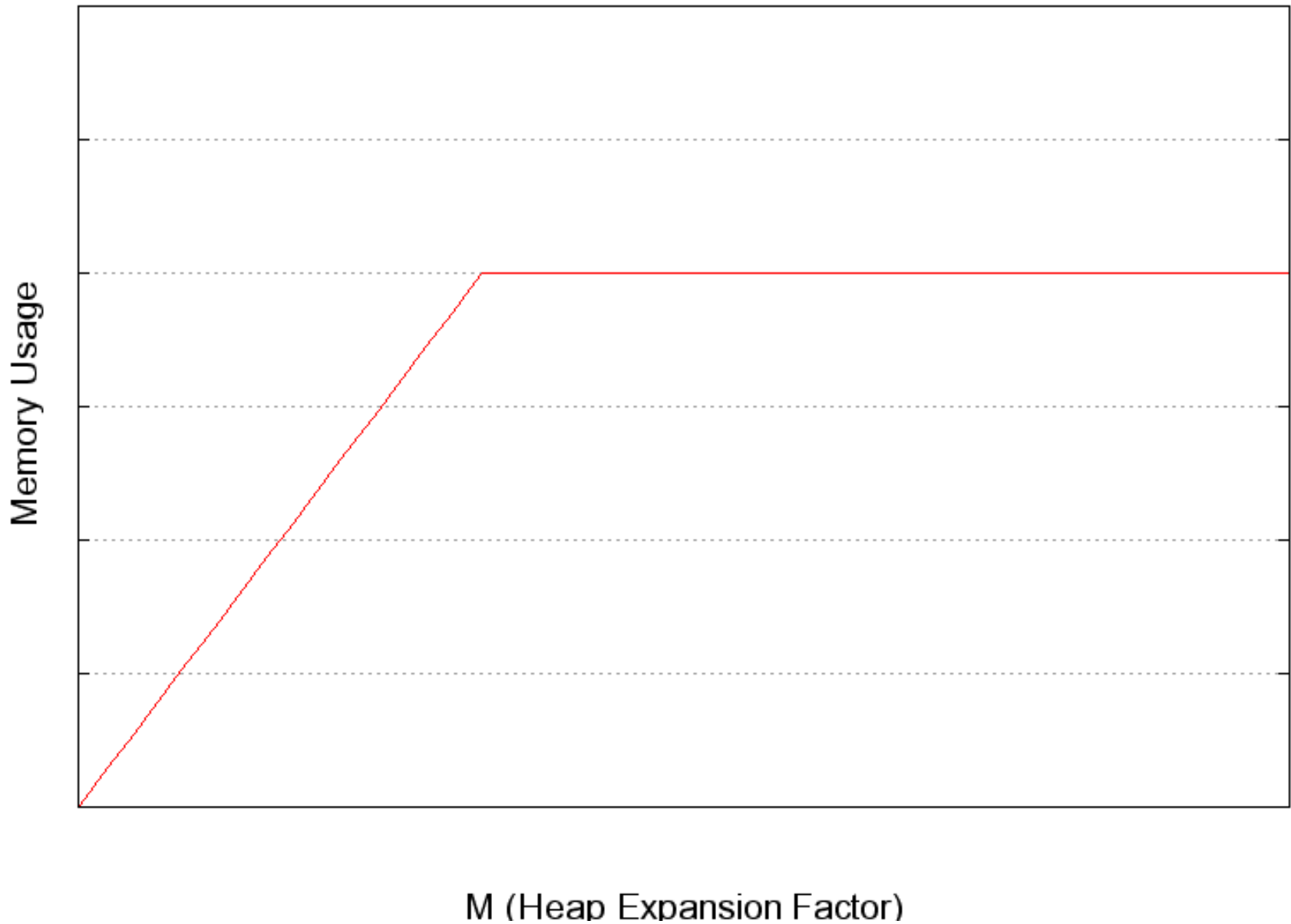
What if $M = \text{Huge}$?

- What if we make the heaps huge?
 - e.g., $M=1000$
- **Pros:**
 - *Dramatically* reduces risk of overflow
- **Cons:**
 - Increase memory consumption by 1000x(?)



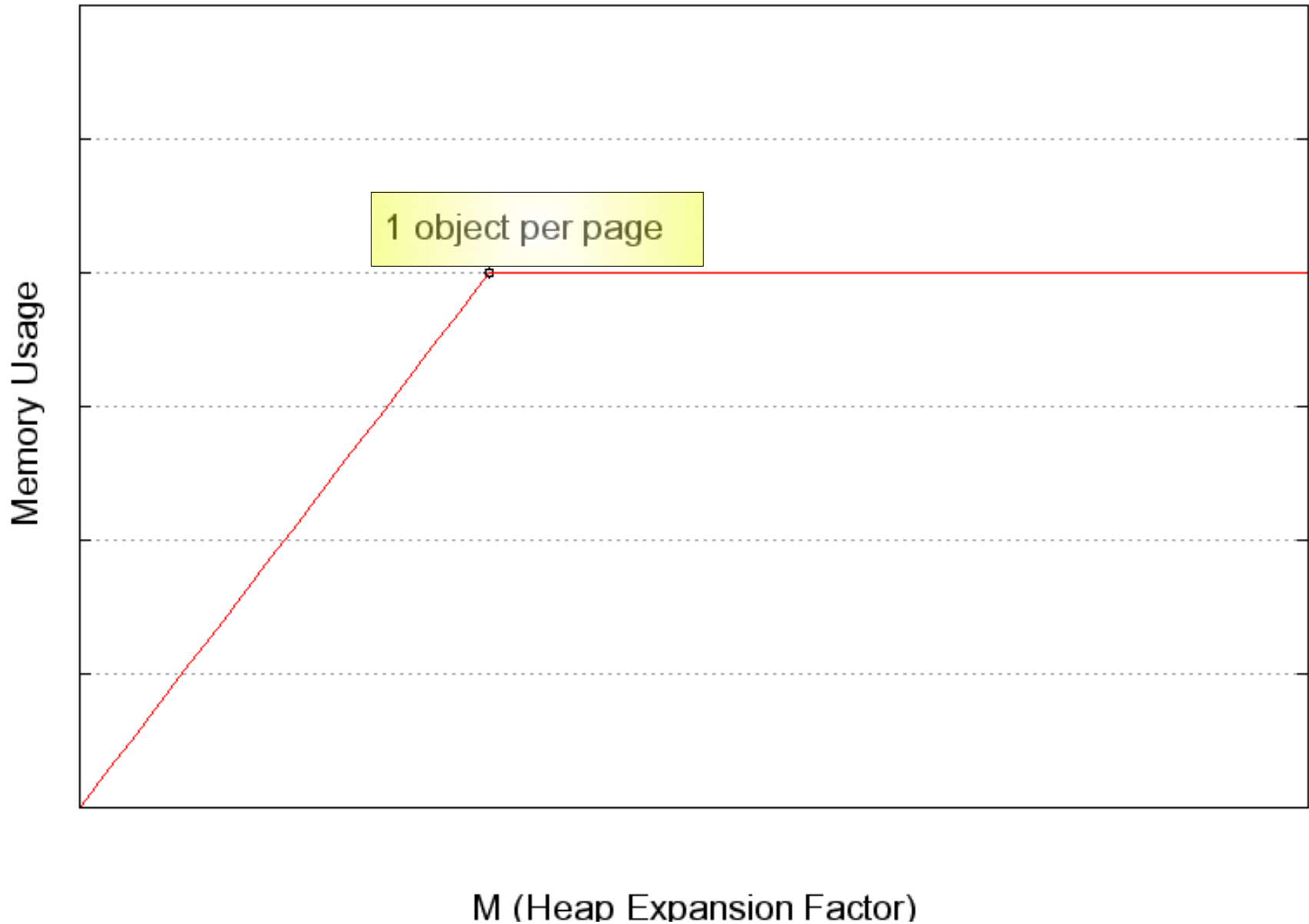
Virtual Memory Usage

VM usage as function of heap expansion factor



Virtual Memory Usage

VM usage as function of heap expansion factor



Quasi-Infinite Heaps

- **Idea:**
 - one object per page – each *far* apart
 - Take advantage of large address spaces
 - Especially 64-bit
 - Closer to “infinite” heaps
- **Key challenge:**
 - Reducing physical memory consumption
 - Spend address space, not RAM
- Initial empirical results
 - Running time

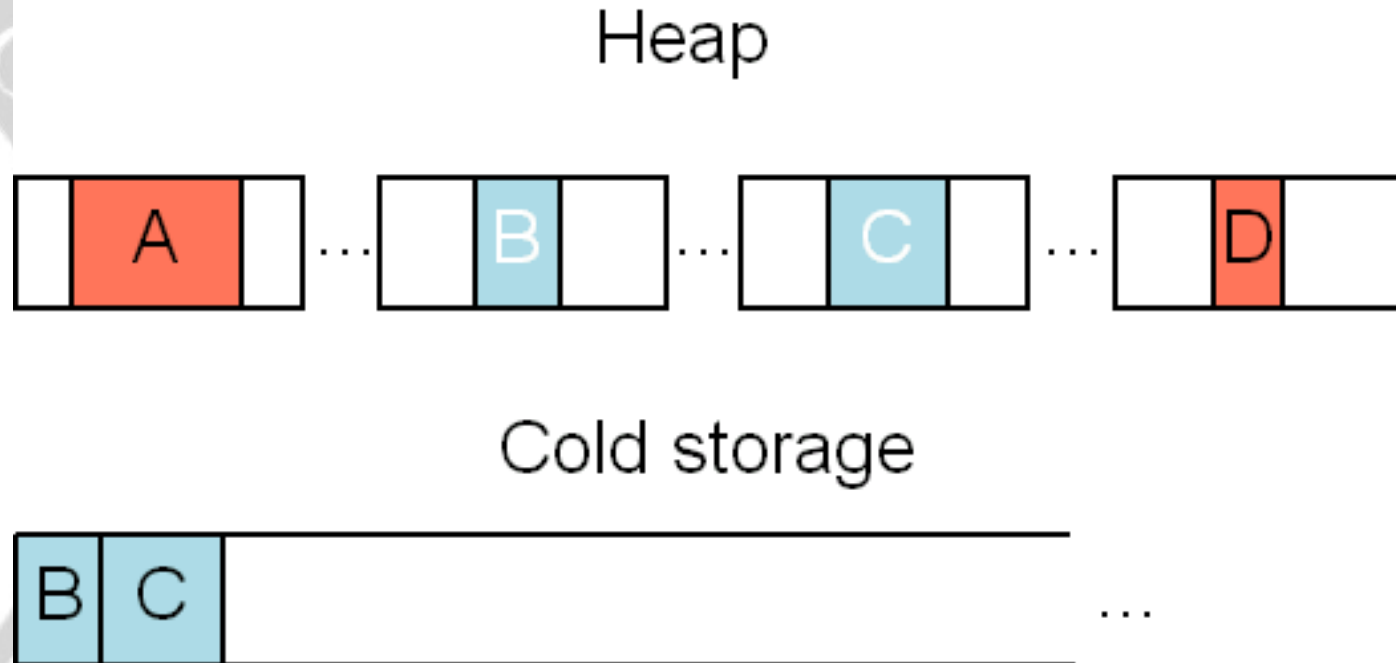


Reducing memory consumption

- Assume locality of reference:
 - 90% time, access 10% data
- Use 2 different heaps
 - **Hot space ("10%")**
 - 1 object per page, empty pages in-between
 - Unrestricted access
 - **Cold space ("90%")**
 - Compacted or even compressed
 - No direct access
 - Copy into Hot space *on demand*



Heap regions



- **A, D: hot**
 - Directly accessible
- **B, C: cold**
 - Pages allocated, protected & evictable (*madvise*)
 - Contents in Cold Storage (ordinary heap)



Managing Hot-Cold Heap Spaces

- Evict objects from hot space
 - Currently using FIFO, fixed # of hot objects
- Page protection traps accesses to cold objects
 - Signal handler copies requested object back into place, unprotects page
- Note: manages pages, not whole objects
 - Allows eviction of cold parts of arrays, etc.



Maintaining soundness

- Buffer overflows
 - Small overflows (within page)
 - Compaction preserves overflowed data across evictions
 - Large overflows (outside page)
 - Trigger signal handler
 - Can fail (now) or allow use of new page
- Dangling pointers
 - Defer frees
 - Evict freed objects to cold space to save space
 - Not yet implemented



Experimental Results

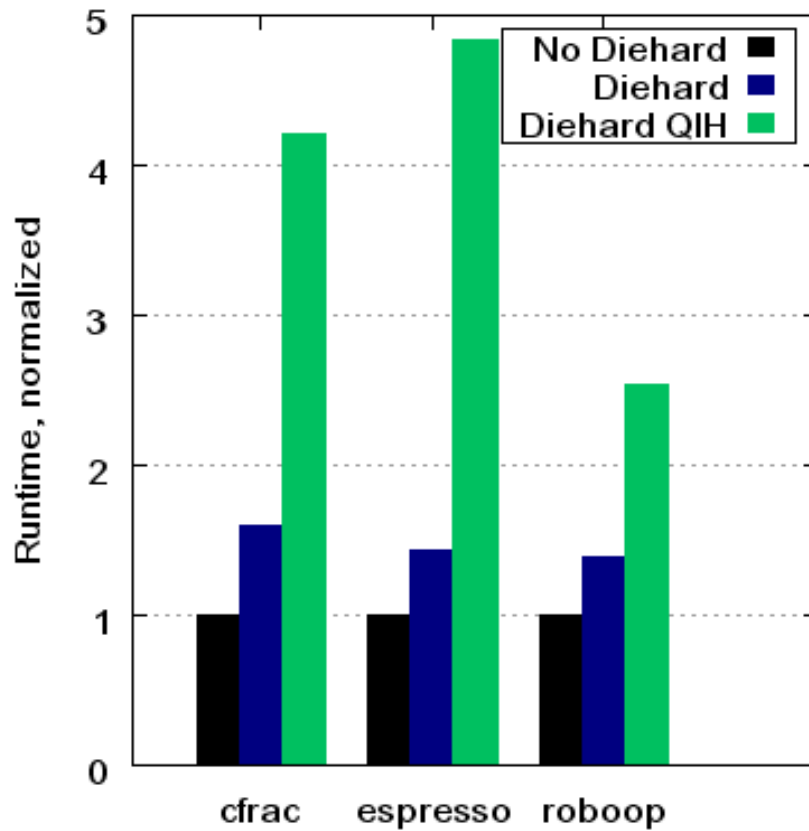
- Ran 3 allocation-intensive benchmarks
 - cfrac
 - espresso
 - robooop
- Compared GNU libc, Diehard and Diehard QIH
- Experimental platform
 - P4 3GHZ, 1 GB RAM, 512KB L2 Cache
 - Linux 2.6
- Measured execution time



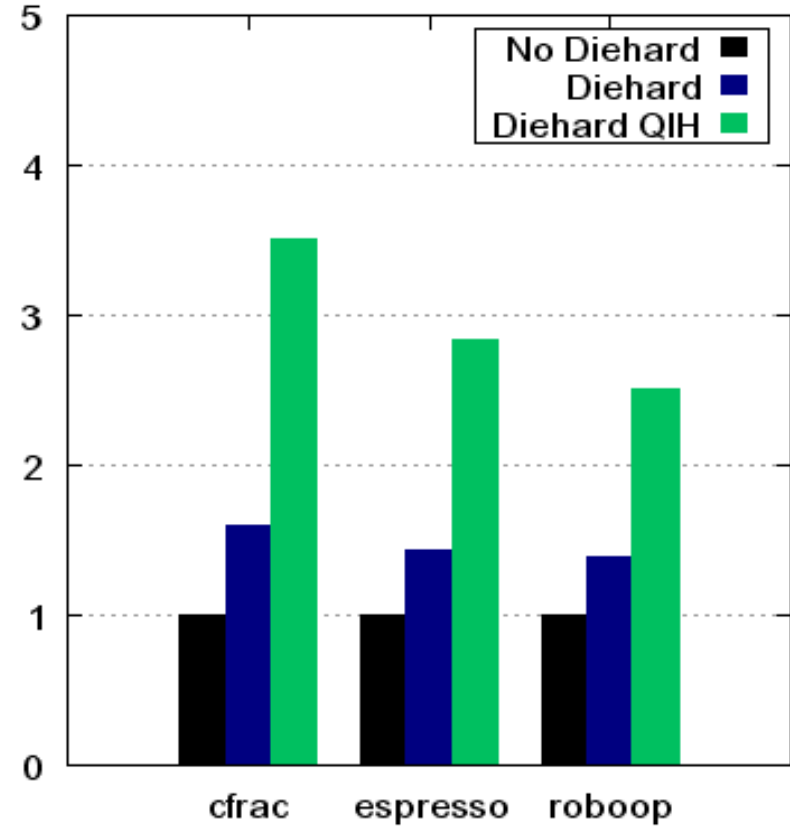
Empirical Results: Runtime



Runtime on Linux, 2000 hot objects



Runtime on Linux, 50000 hot objects



Performance-limiting factors

- Increased cost of allocation/dealloc
- Reduces/Limits spatial locality
- Cache side-effects
 - Increased L2 miss rate
- VM side-effects
 - Increased TLB miss rate
 - Larger page tables



Future work

- Error-injection experiments
- Smarter eviction policies for Hot space
 - Clock with s/w referenced bits
 - Clock with h/w referenced bits (kernel mod)
- Concurrent evictions
 - Can be moved to a separate thread
- Automatic Hot space sizing
 - Adjust to application behavior
- Add object coloring to improve cache performance



Conclusion

- Our Goal:
 - Provide near-infinite heap guarantees against certain memory errors for unaltered C/C++ applications
 - Exploit VM and access locality to do it reasonably efficiently
- Work in progress



Questions?

