# Singleton: Systemwide Page Deduplication in Virtual Environments

**Prateek Sharma** , Purushottam Kulkarni

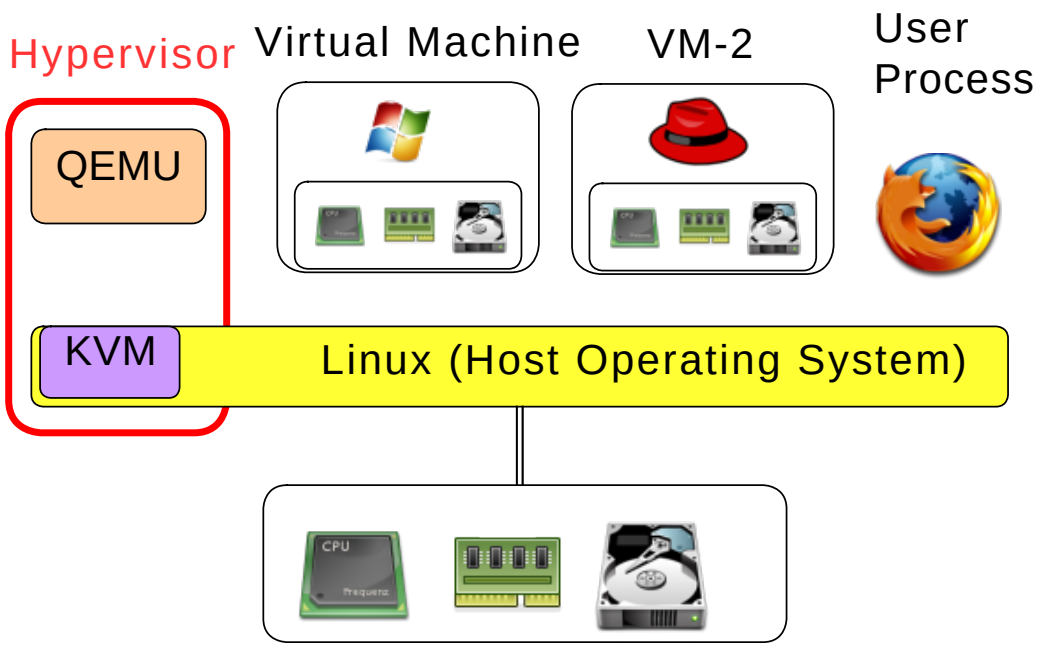
Dept. of Computer Science & Engineering
Indian Institute of Technology Bombay

HPDC 2012

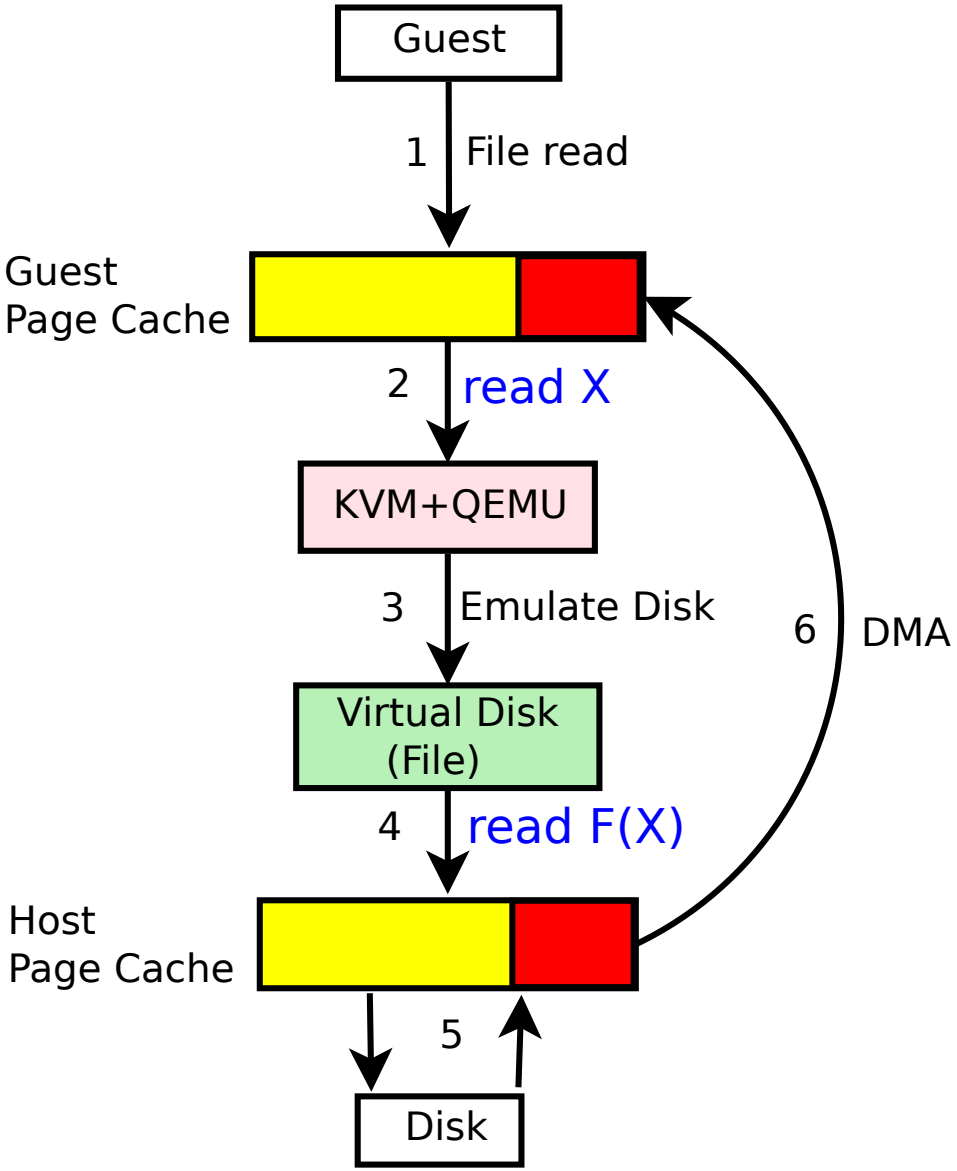
# Virtualization with KVM

- Virtual hardware devices emulated by **QEMU** (Quick EMUlator)

- Uses existing OS services
  1. VM ≡ process.
  2. VM memory ≡ malloc
  3. Virtual disk ≡ file

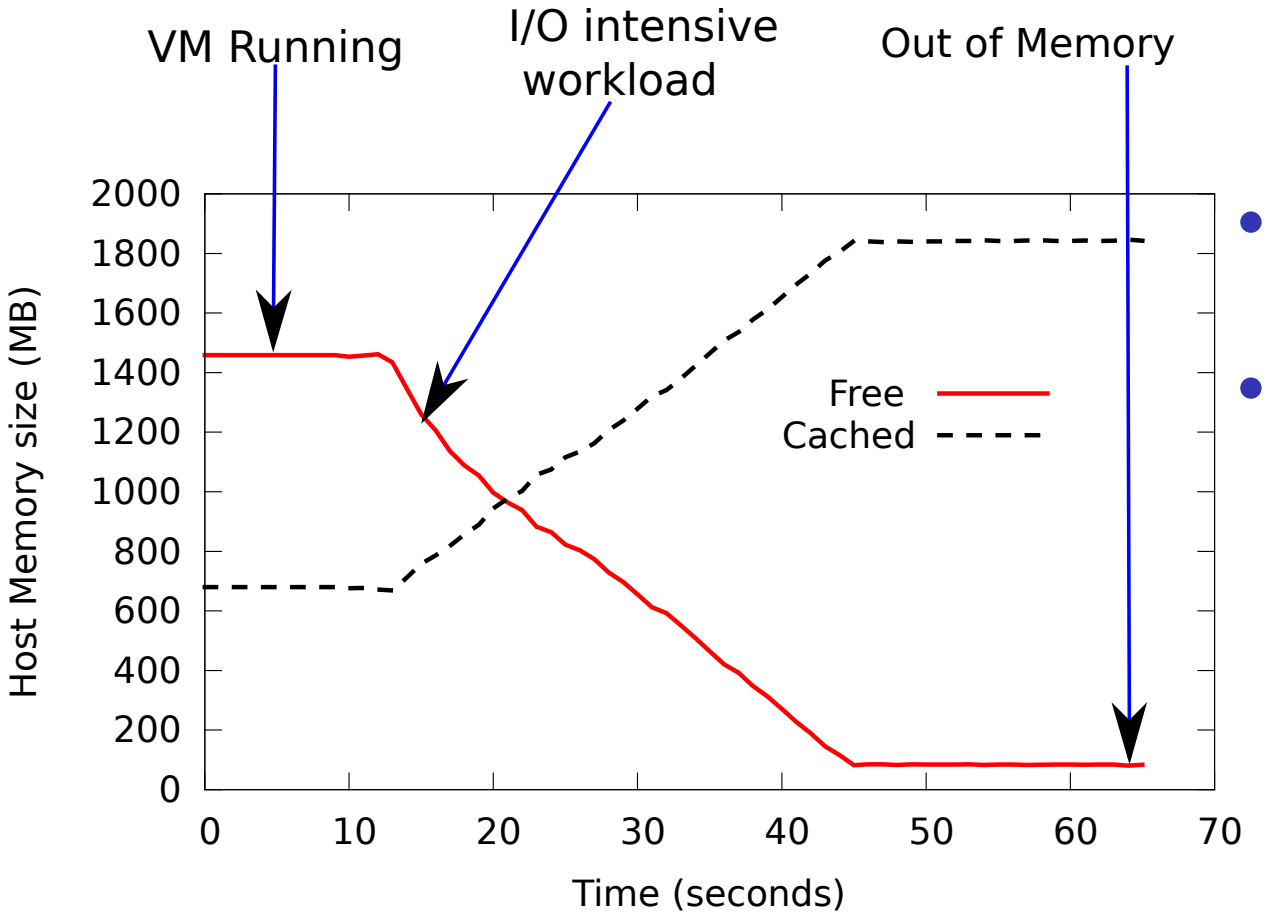


# Guest Disk I/O




1. Guest application reads from file
2. Guest page-cache miss *implies* disk read X
3. KVM+QEMU emulate disk as a file.
4. `read` syscall for block F(X) on guest Virtual Disk File.
5. Host file-system read via host page-cache
6. Page copied to guest via KVM/QEMU DMA.

# Does Virtualization eat-up all Free Memory?

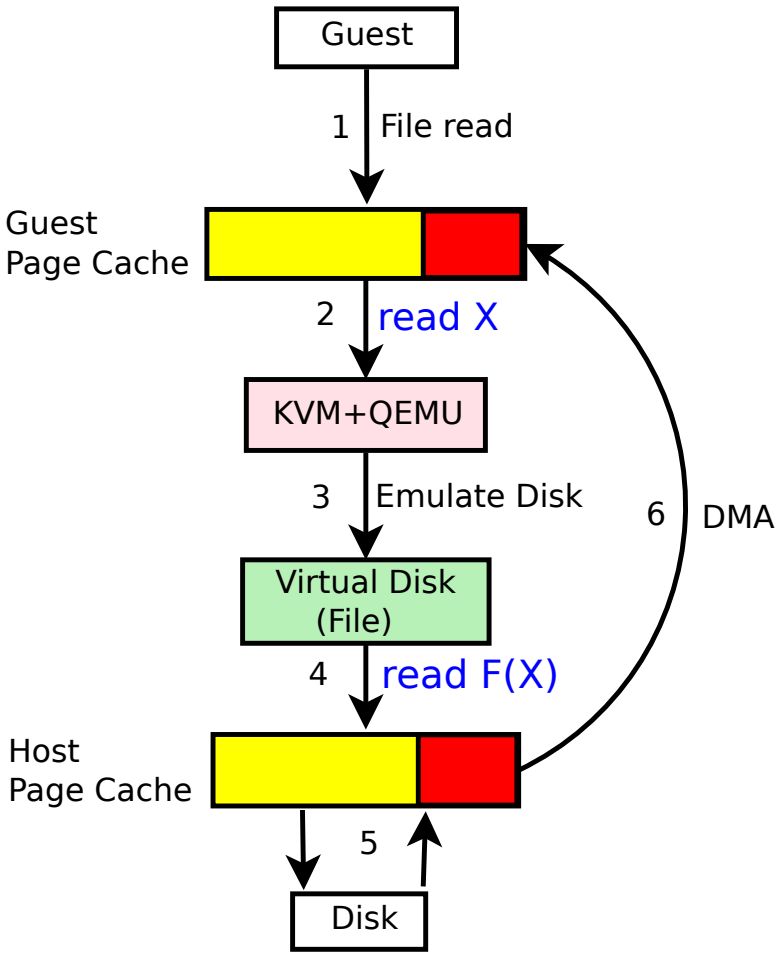



- Free memory : 1.4 GB $\rightarrow$ 0
- Corresponding increase in cached memory(Page Cache)

# Host Caching Pitfalls

Memory Available for VMs=Host Memory−**Host Cached Memory**

- Cache size is dynamic and workload dependent.
- Pages present in both host and guest page-caches. (Double-caching)
- Unified page eviction $\Longrightarrow$ swapping of guest pages and Out-Of-Memory killing.



# Potential Existing Solutions

## Bypass Host Cache with Direct-I/O

- Use `O_DIRECT` flag for virtual disk file.
- Adversely impacts guest performance due to unoptimized kernel code-paths (upto 2x slowdown for guest I/O)

## Fadvise `DONT_NEED`

- Fadvise instructs the OS to keep/discard cache pages of a file.
- Not sticky, needs to be called periodically

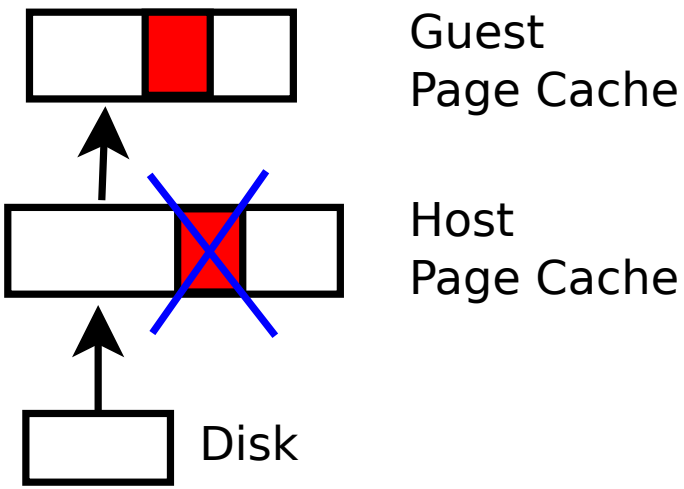
## Exclusive Caching

- Eviction based placement: keep blocks evicted from higher level caches (*DEMOTE*)
- Implemented in Virtualization context in *Geiger* (paravirtualized solution, tracks guest activity explicitly)
- Cannot reliably track all guest evictions(black-box assumption)

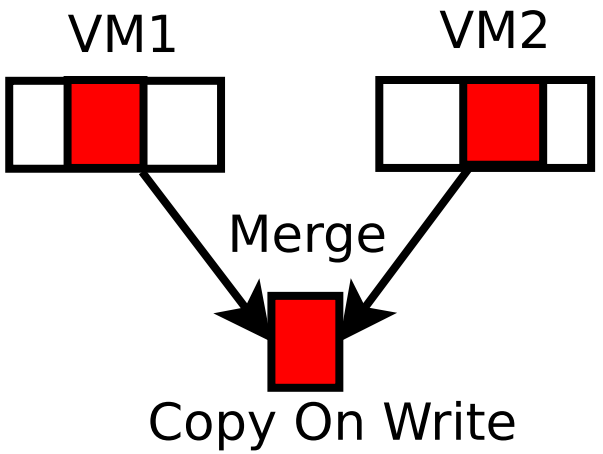
# The Solution : Singleton

## Singleton

We implement a black-box, **exclusive-caching** solution to the double-caching problem by using inter-VM **page-deduplication**.



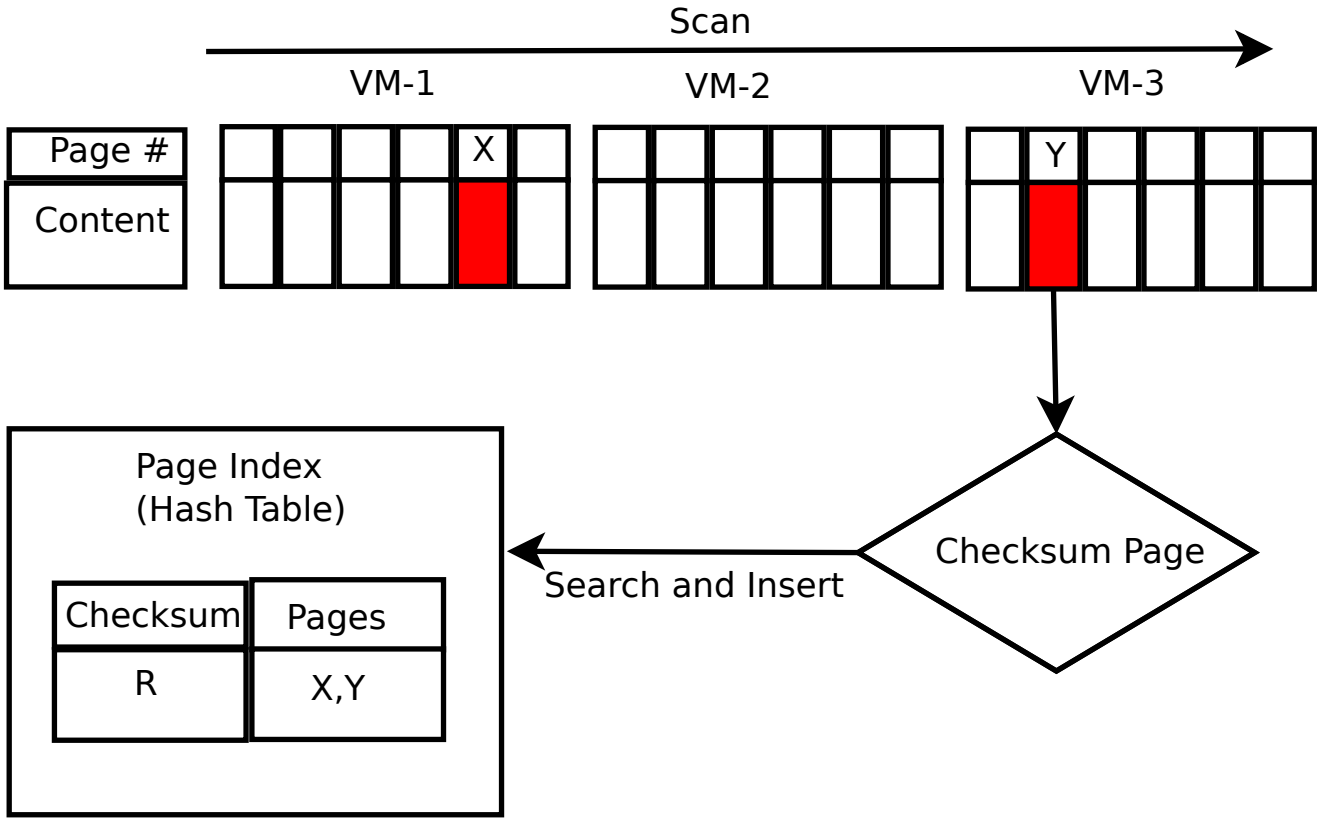

Exclusive Caching




Page Deduplication

# KSM : Kernel Samepage Merging

- Linux kernel thread.
- Identifies and merges duplicate pages.
- Periodically Scans and checksums each page and builds a page-index.



# Exclusive Caching using Singleton

## Key Idea

If a page is present in the guest, drop it from the host page cache.

Cache Scrubbing Algorithm :

```
1. Scan all guest pages and build index H ;
2. For all host page-cache pages p :
2a.     if page in page-index :
2b.         Drop page from host cache
```

# Challenges in cache-scrubbing

1. Page-Scans are expensive.
   - Guest memory sizes and host page-cache size are huge (millions of pages)
2. Frequent Page-Scans are necessary.
   - Page contents are volatile, hence frequent scanning required.
   - Frequent Page-Scans $\implies$ Smaller Host cache

# Cache Scrubbing Optimizations

1. Reduce Page Deduplication overhead.
2. Smart scrubbing : Scrub each VM's host cache separately.
3. Scrubbing frequency control algorithm to balance cache size and overhead.

# Page Deduplication Optimizations

- Singleton uses a highly optimized version of KSM.
- Hash-tables instead of red-black search-trees (2x improvement)
- Spatial-locality "lookahead" heuristic — most shared pages are file-backed, and hence sequential.
- Hardware Assisted scanning: Scan only pages which have been written-to. Uses AMD NPT dirty-bit. (10x speedup) (All results are without this).

# Experimental Setup and Workloads

## The contenders

**Default** No KSM/Singleton running

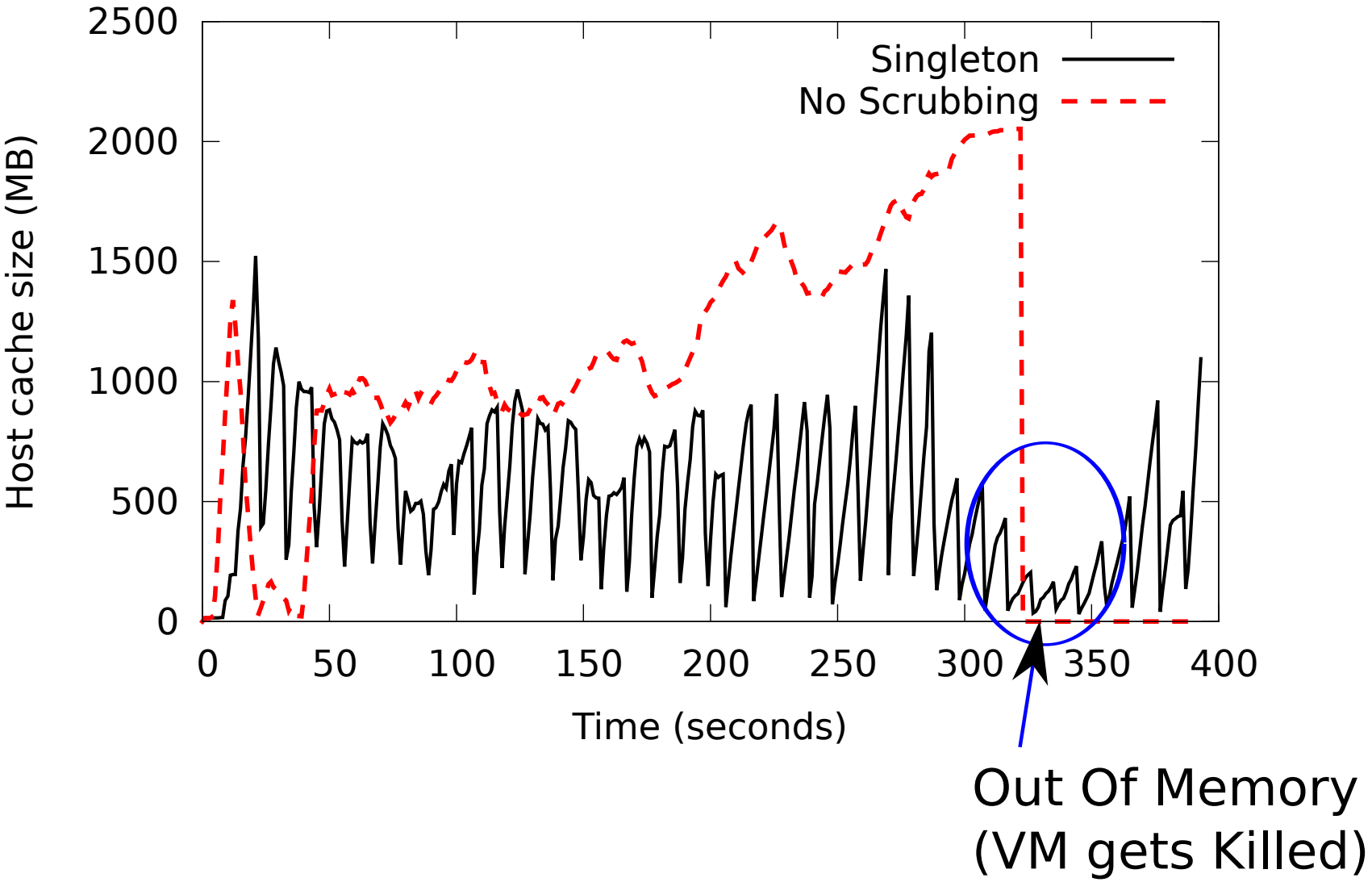
**Fadvise** : Optimized-KSM + fadvise(DONT_NEED) periodically

**Singleton** : Optimized-KSM + Cache scrubbing

## Workloads

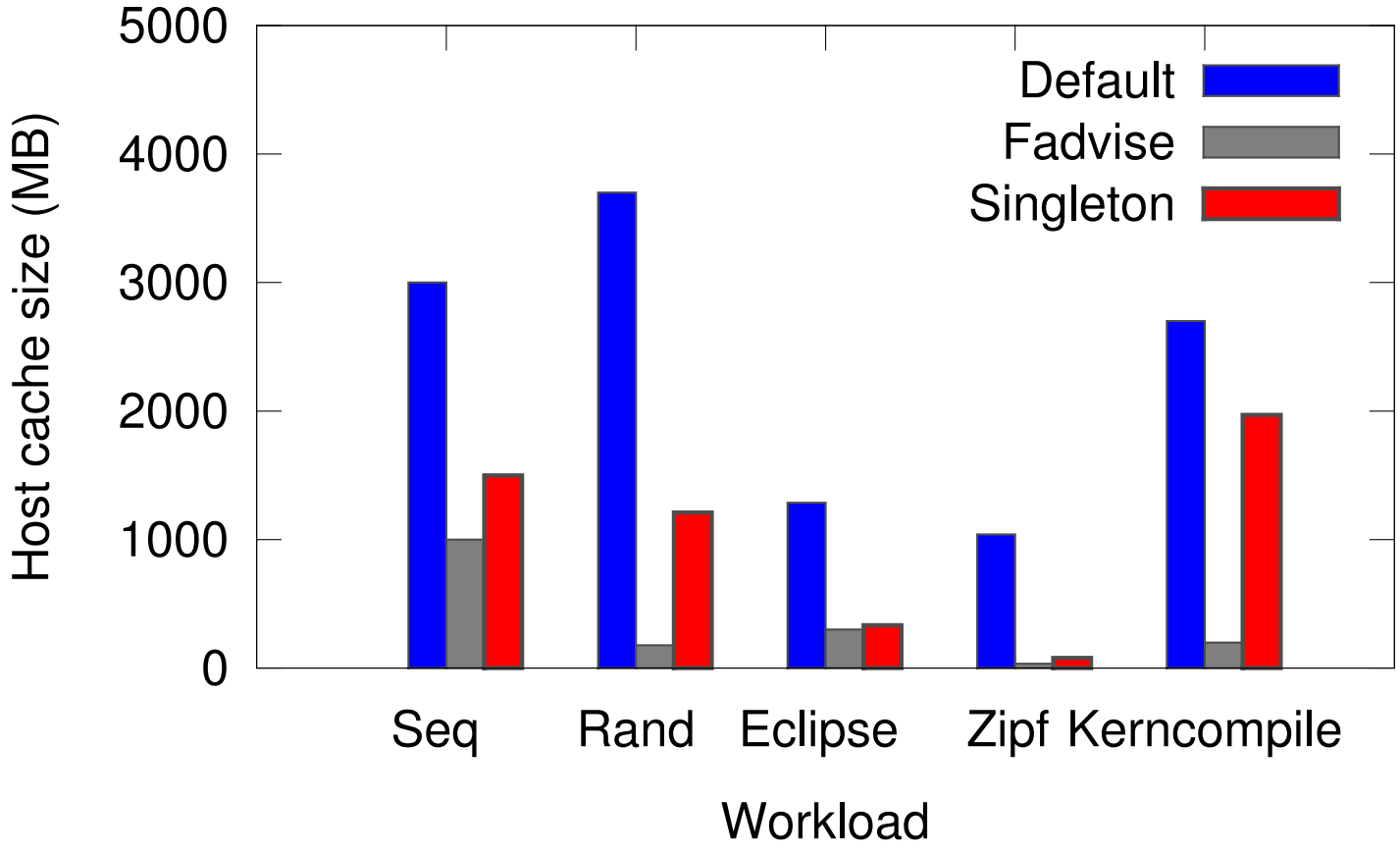
- I/O intensive workloads in VMs to exercise caches and KSM.
- IOZone, Bonnie, Zipf, Kernel-compile, Dacapo-Eclipse

# Singleton Cache Scrubbing



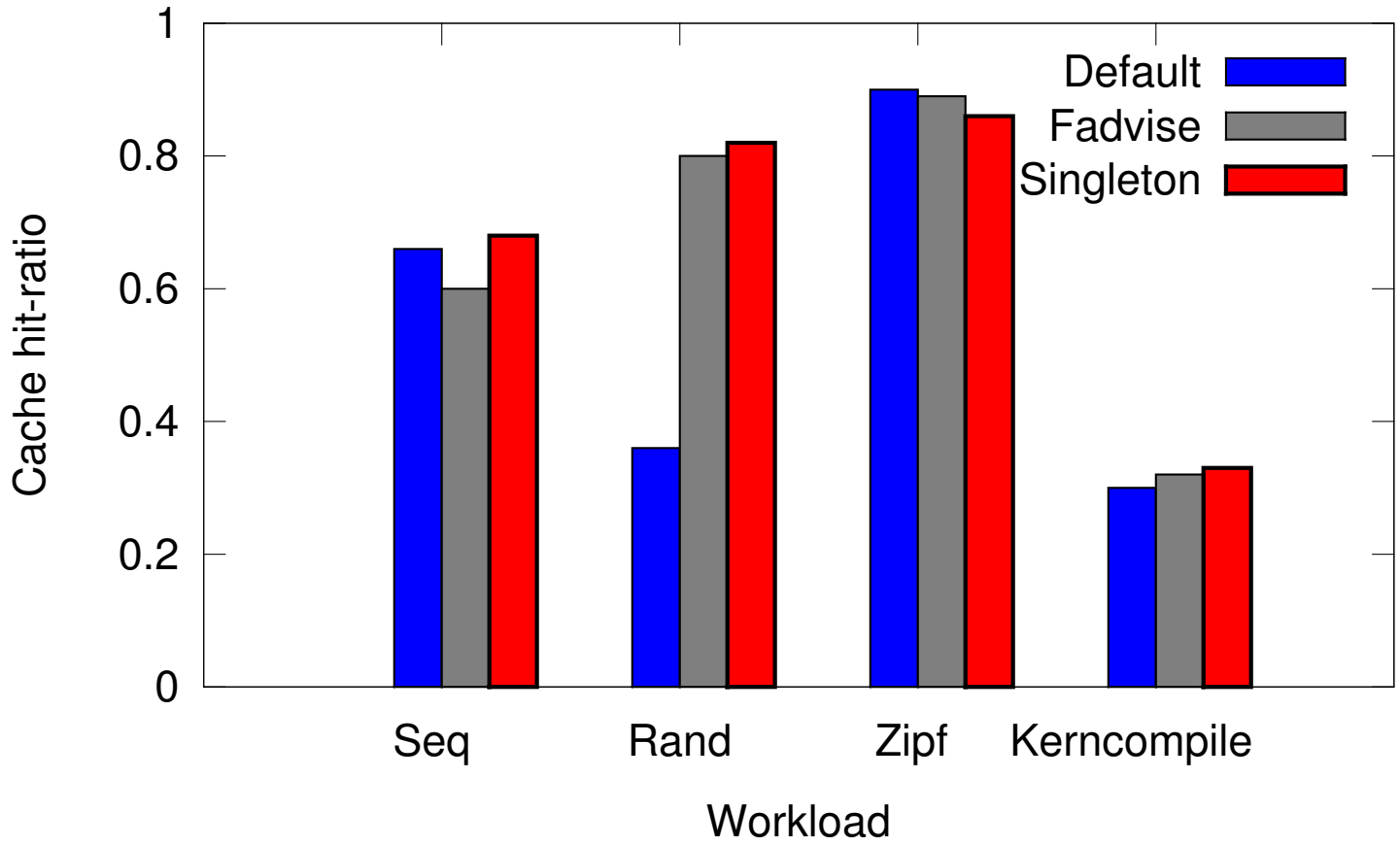

Cache scrubbing results in sharp falls in host cache size. VM is performing heavy I/O

# Reduction in Host cache size




2-10x smaller host page-caches due to cache-scrubbing.
Smaller host cache $\Longrightarrow$ More Free memory

# Increase in Host cache hit-ratios




Host page-cache hit-ratio increases by 2x for random workloads due to exclusive-caching.
Guest I/O performance shows corresponding increase.

# Singleton Overhead

## Sources of overhead

- KSM Scanning
- Cache-Scrubbing

## CPU overhead

CPU utilization of the Singleton kernel thread is $<$ **20%**.

## Load Average

Systemwide load average is upto **2x** lower than default
Primarily due to lower swapping activity.

# Memory Overcommit

- Singleton combines Page Deduplication and exclusive caching.
- Page dedup is a memory overcommitment solution in itself.
- Ideal for desktop workloads :
  - Large amount of common libraries, applications, data
  - Mostly idle

In a mixed workload of desktop, Kernel-compile, developer VMs, Singleton increased number of VMs by **30%.**

| | Sequential | Kerncompile | Desktop | Total |
|---|---|---|---|---|
| Default | 2 | 2 | 4 | 8 |
| Fadvise | 2 | 2 | 4 | 8 |
| Singleton | 2 | 2 | 7 | **11** |

Number of running VMs till system crashes or runs out of memory.

# Singleton: features and benefits

- Singleton : Exclusive-cache solution for Virtual Environments
- Uses inter-VM Page Deduplication
- 2-10x smaller host page-caches
- Upto 40% faster guest I/O.
- Low overhead.
- Increases risk-less memory overcommitment.
- Black-box, non-intrusive. Works with most KVM configurations.

# The Big Picture: Resource Control in Virtualized Environments

*Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem.* - David Wheeler

- The Operating System functions as the hypervisor.
- Both guest and host OS managing resources oblivious of each other.
- Enacting complementary or redundant optimizations.

## Multiple levels of independent resource management

- Multiple levels of paging.
- Multiple levels of disk,packet scheduling.
- Multiple levels of File-Systems and Page-Caches.

Thank You!

Questions ?

# IO Performance

| | Sequential reads (KB/s) | Random reads (KB/s) | Zipf Reads (KB/s) |
|---|---|---|---|
| Default | 4,920 | **240** | 265,000 |
| Fadvise | 4,800 | 280 | 260,000 |
| Singleton | 5,000 | **360** | 270,000 |

Guest I/O performance for various access patterns.

# Overhead

| | Singleton CPU % | Singleton load average | Default load average |
|---|---|---|---|
| Sequential | 17.74 | 5.6 | **12.3** |
| Random | 19.74 | 4.8 | **10.3** |
| Kerncompile | 11.7 | 5.3 | 6.0 |
| Zipf | 10.2 | 4.9 | 4.9 |

Scrubbing overhead and host load averages.

# IO Performance Isolation

| | Sequential Read speed | Kernel compile time |
|---|---|---|
| Default | 7,734 KB/s | 3165 s |
| Fadvise | 7,221 KB/s | 3180 s |
| Singleton | 7,432 KB/s | **2981 s** |

Impact of I/O interference on the kernel-compile workload.

# Swapping activity

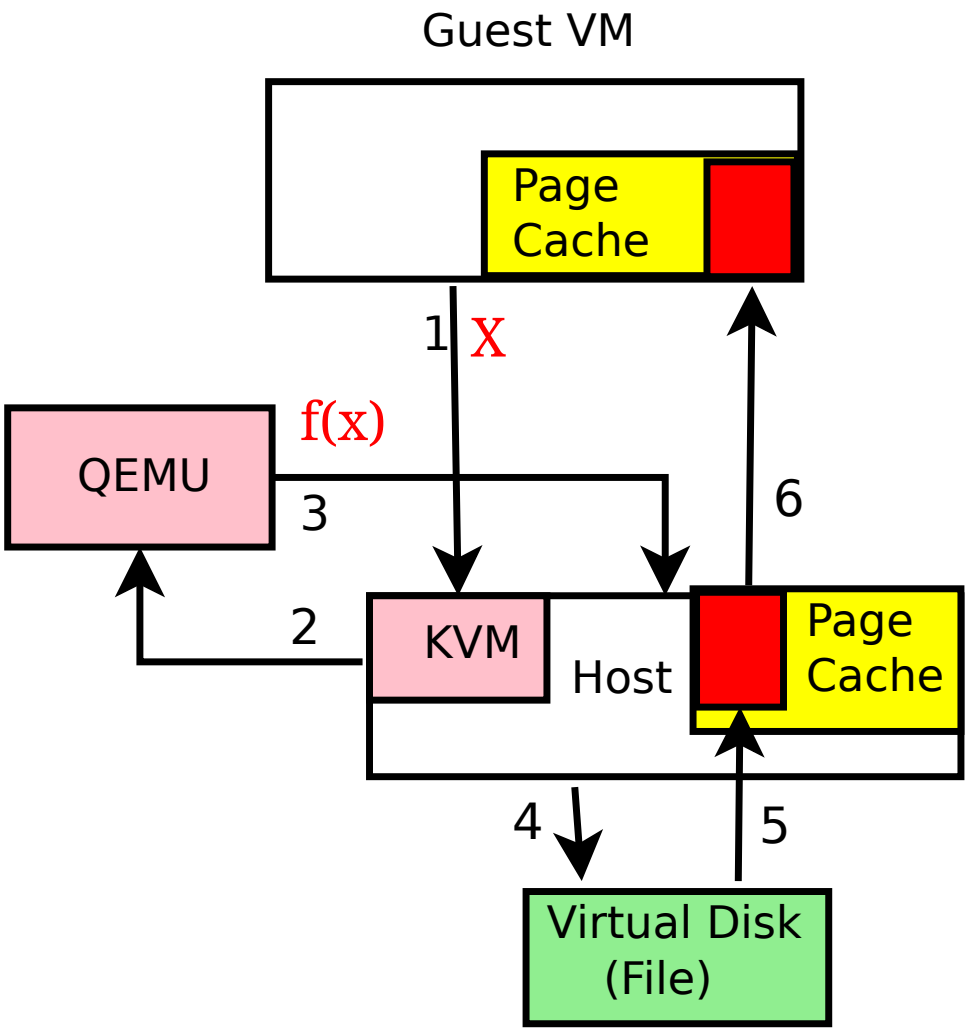
| | Avg. pages scanned/s | Cache pages dropped/s | Scan efficiency |
|---|---|---|---|
| Default | 1,839,267 | 1459 | 0.07 % |
| Singleton | 7 | 109 | 99.87 % |

Page eviction statistics with and without Singleton.

# Page sharing

| Workload | Pages-shared |
|---|---|
| Sequential | 80,000 |
| Random | 133,000 |
| Kerncompile | 350,000 |
| Desktop | 300,000 |
| Eclipse | 215,000 |

# Guest Disk I/O




1. Guest OS issues read request for block X.
2. Trapped by KVM, signals QEMU to handle IO
3. QEMU issues `read` syscall for block f(X) on guest Virtual Disk File.
4. File-system read
5. Read completion
6. Page copied to guest via KVM/QEMU DMA.

# Scrubbing frequency control Algorithm

```
  Update_memory_usage_stats(&Cached, &Free, &Memory);
//Case1: Timer expires. t is current scrub interval
  if(cycle_count-- <= 0) {
    Dropped = scrub_host_cache() ;
   //returns num pages dropped
    prev_t = t ;
    t = prev_t*(Cached + Dropped)/Cached; }
```

# contd

```
//Case2: Memory pressure
 else if(Cached > Memory*th_frac_cached ||
              Free < Memory*th_frac_free) {
    Dropped=scrub_host_cache();
    prev_t = t ;
    t = prev_t*(Cached - Dropped)/Cached; }
  cycle_count=t;
```