An Empirical Study of Memory Sharing in Virtual Machines

Sean Barker*, Timothy Wood^,

Prashant Shenoy*, and Ramesh Sitharaman*

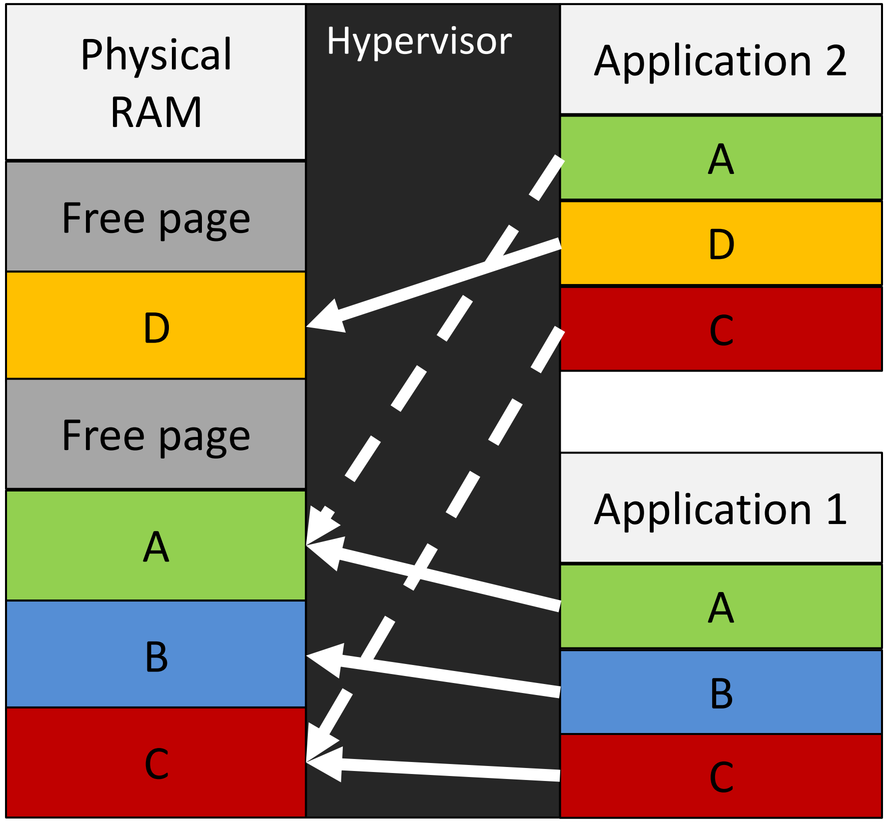
*University of Massachusetts Amherst

^The George Washington University

USENIX ATC'12

Virtualization and Page Sharing

- Virtualization
  - Flexible mapping of resources to users
  - Efficient resource sharing to maximizing the benefits
- Page Sharing
  - Eliminate identical pages of memory
  - Virtual pages mapped to physical pages
  - Hypervisor detects duplicates
  - Replaced with copy-on-write references



Content Based Page Sharing Systems

- VMware ESX Server [SIGOPS 02]
  - Periodic memory scanning to detect duplicates
  - >30% memory savings
- Difference Engine [OSDI 08]
  - Sub-page sharing and patching
  - >60% memory savings
- Satori [USENIX ATC 09]
  - Sharing of short-lived pages
  - >90% of possible sharing captured

Problems in Page Sharing

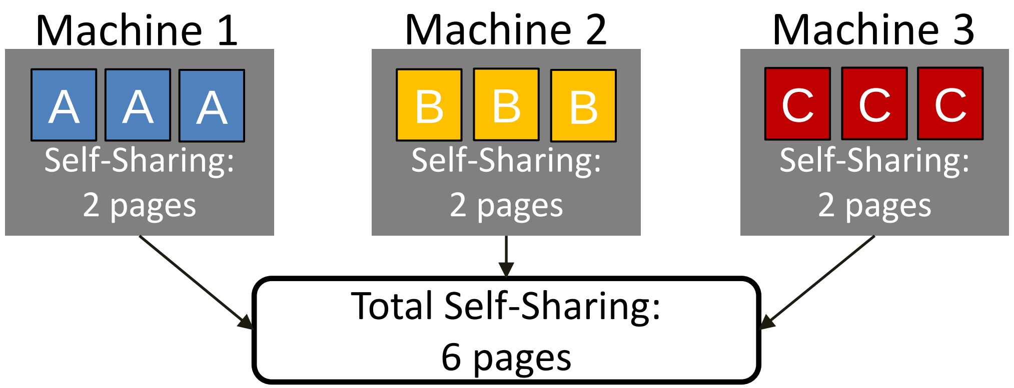
- What levels of sharing are possible in typical real-world machines?
- What are the factors that impact sharing potential?
  - OS family? Versions? Applications?
- How emerge in technologies impact sharing?
  - New OS technologies?
  - VDI (Virtual desktop infrastructure) farms?
  - LAMP(**L**inux, **A**pache, **M**ySQL, **P**HP/**P**erl/**P**ython) clusters?
- **Target: Provide practical insights into these questions through a careful study of memory data**

Data Collection

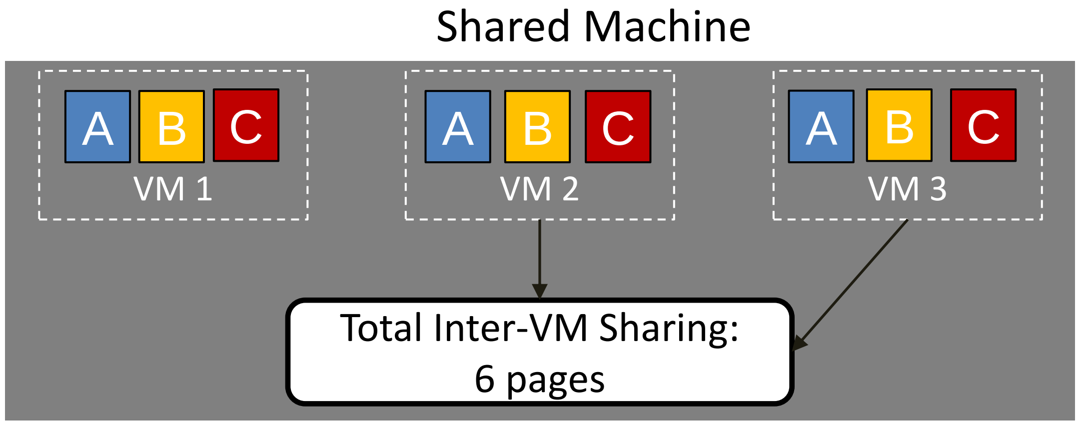
- Traces from uncontrolled machines
  - ~50 real machines (server/desktop mix)
  - Uncontrolled user workloads
  - Memory snapshots every 30 minutes
- Traces from controlled VMs
  - Mixed 32/64bit version of Mac/Win/Linux VMs
  - 3 application setups for each VM:
    - No workload (freshly booted)
    - Server apps (LAMP stack)
    - Desktop apps (office, browser, media player)
- Trace: the content type of page; the process(es) using the page.

Types of Sharing

- Self-Sharing: sharing within individual VMs
  - E.g., multiple zero pages

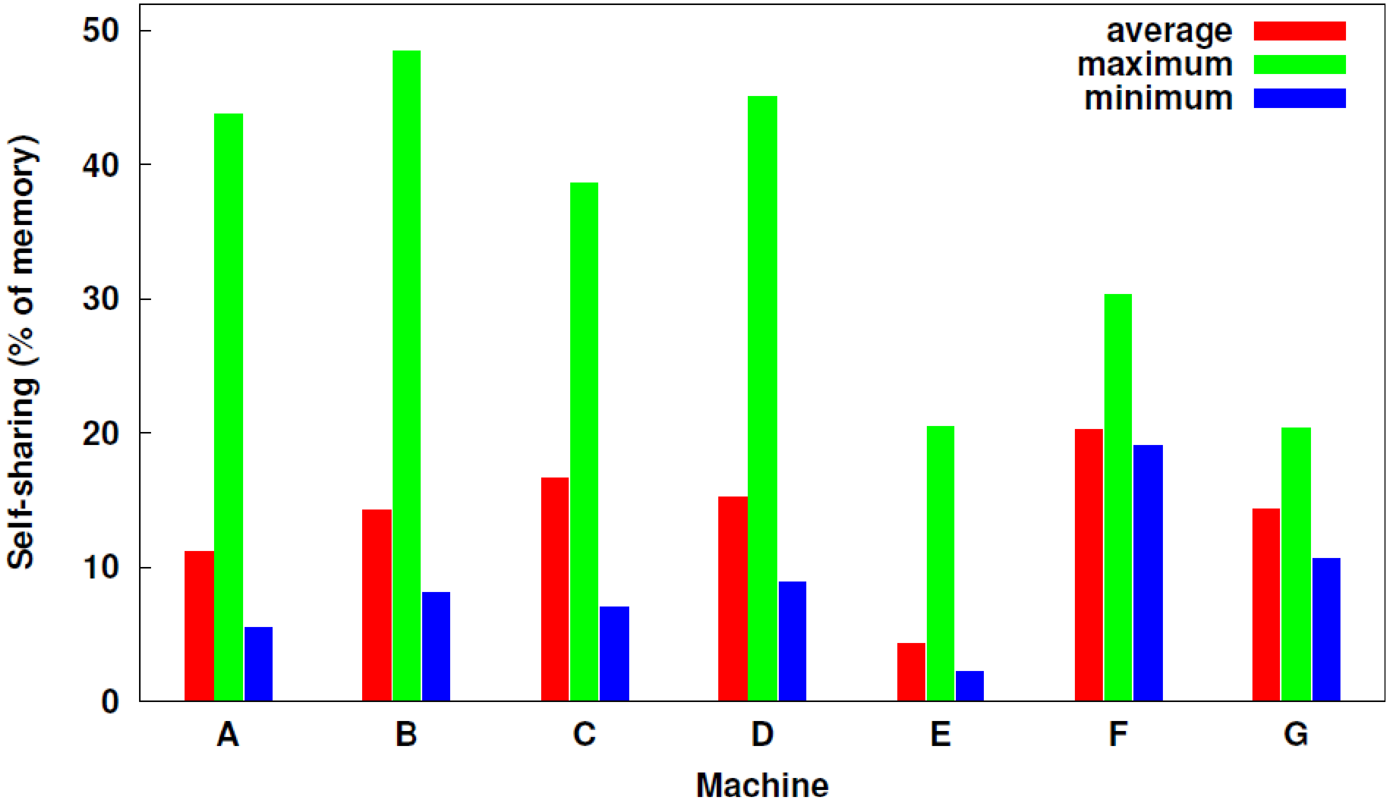


- Inter-VM sharing: sharing across multiple VMs
  - E.g., shared OS states, applications



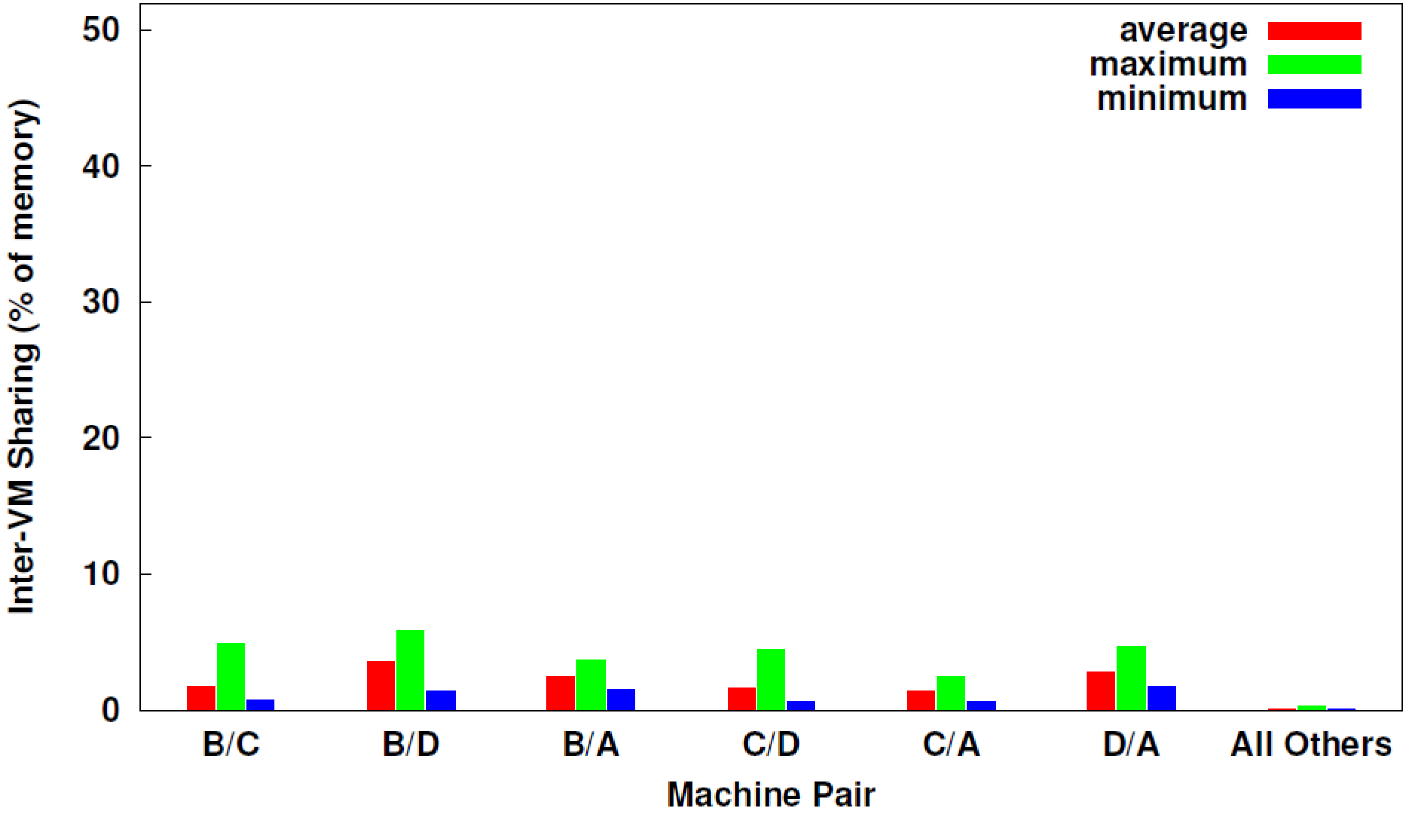
Real-world Sharing Potential

- Self-Sharing:
  - Average sharing of 14% in any machine (Excluding zero pages)
  - High sharing potential surrounded by long periods of much lower sharing potential



Real-world Sharing Potential

- Inter-VM Sharing:
  - Observed minimal (<2%) inter-VM sharing potential
  - <0.1% sharing in 15 of 21 pairings

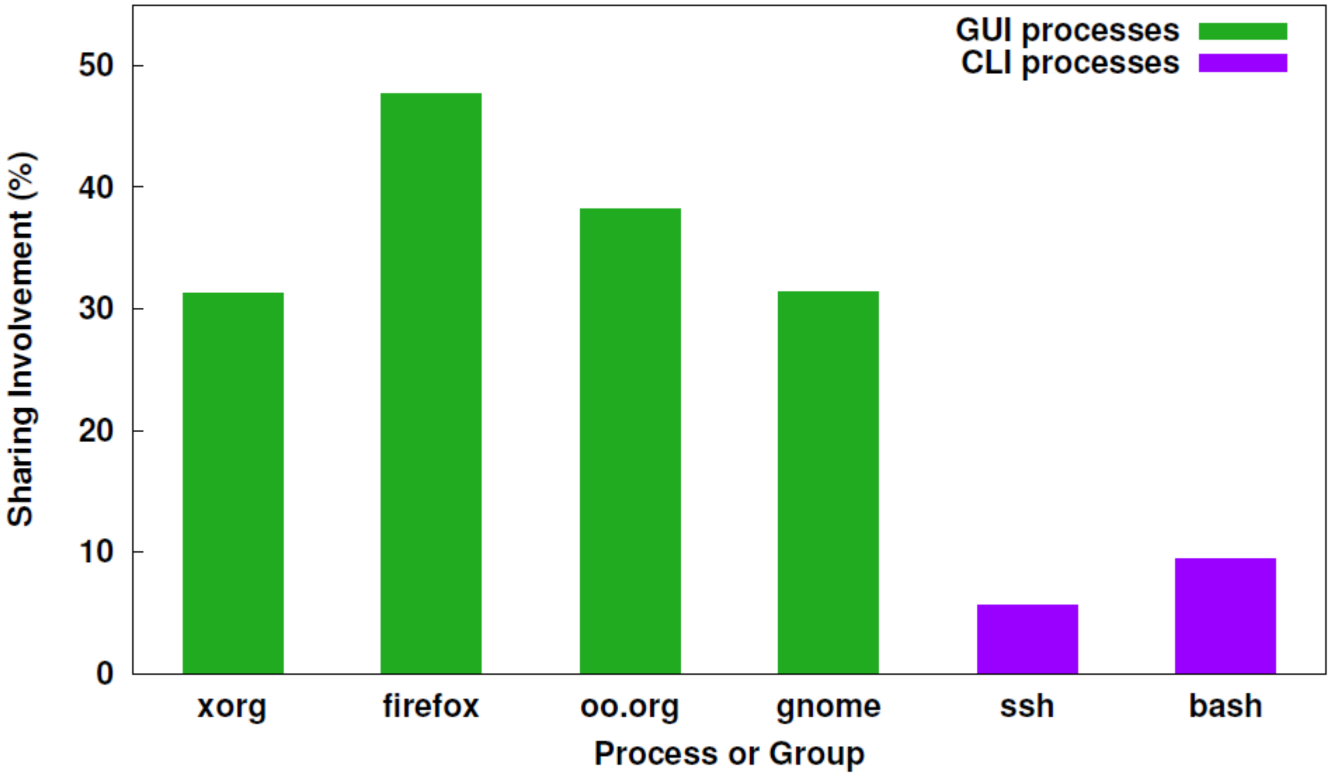


Real-world Sharing Potential

- Typical **15%** possible sharing observed
  - Significant, but less than expected from synthetic workloads
- Most (**85+%**) sharing derived from self-sharing
  - What about collocating many VMs?
  - All 7 machines page sharing still **80+%** from self-sharing
- Self-sharing doesn't require virtualization!
  - Could capture it within a VM or nonvirtualized host
- Self-sharing is significant, but what causes it?

Self-Sharing Case Study

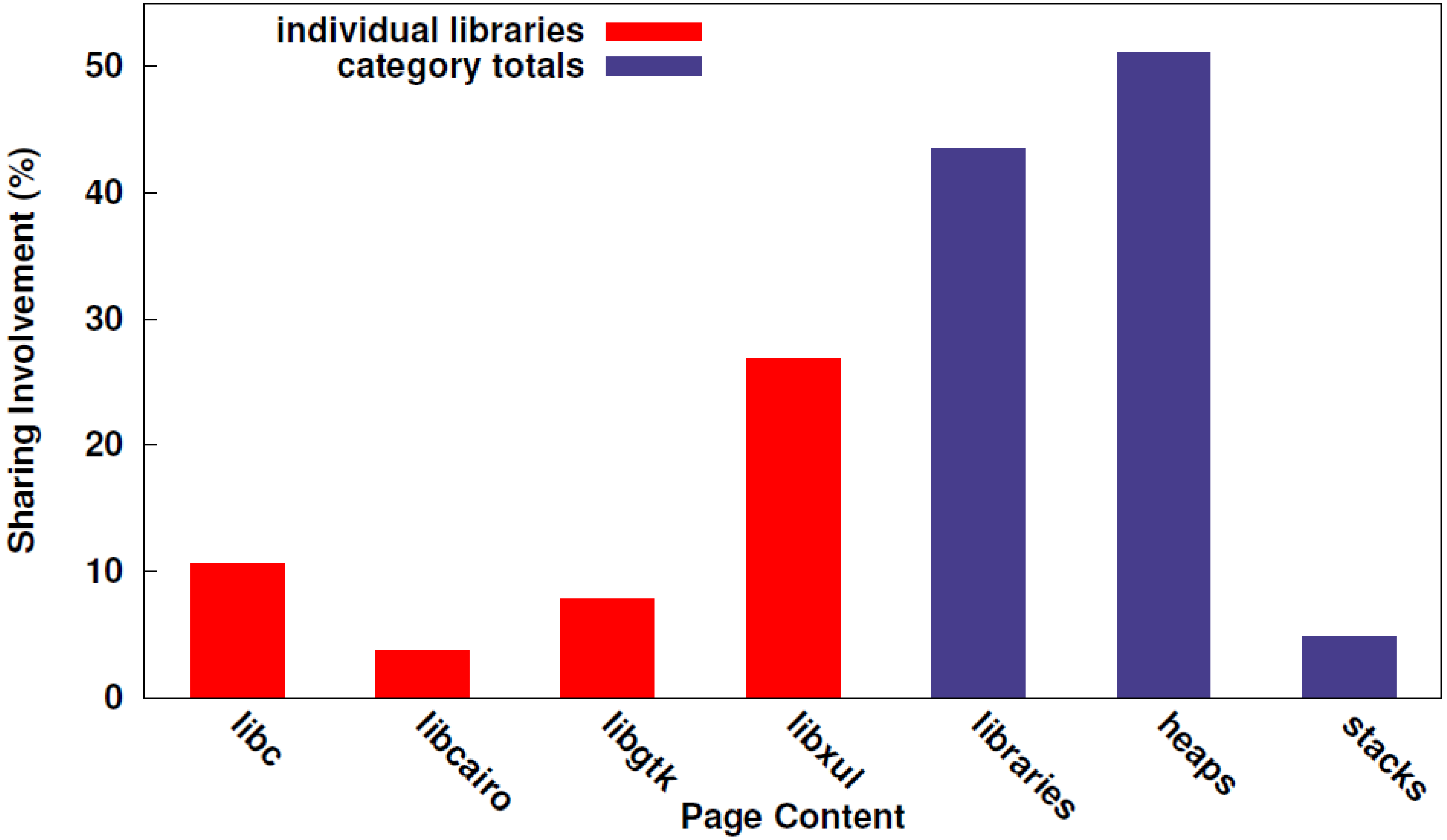
- Self-Sharing by Process
  - >30% sharing processes GUI apps/libraries
  - <20% sharing from other system processes
  - Memory footprint likely dominated by GUI



Process self-sharing resulting from user workload

Self-Sharing Case Study

- Self-Sharing by Content
  - **94%** sharing from libraries and heaps
  - Possibly from recreated data structures
  - 2.3 MB sharing from single Xorg heap page (~600 copies)



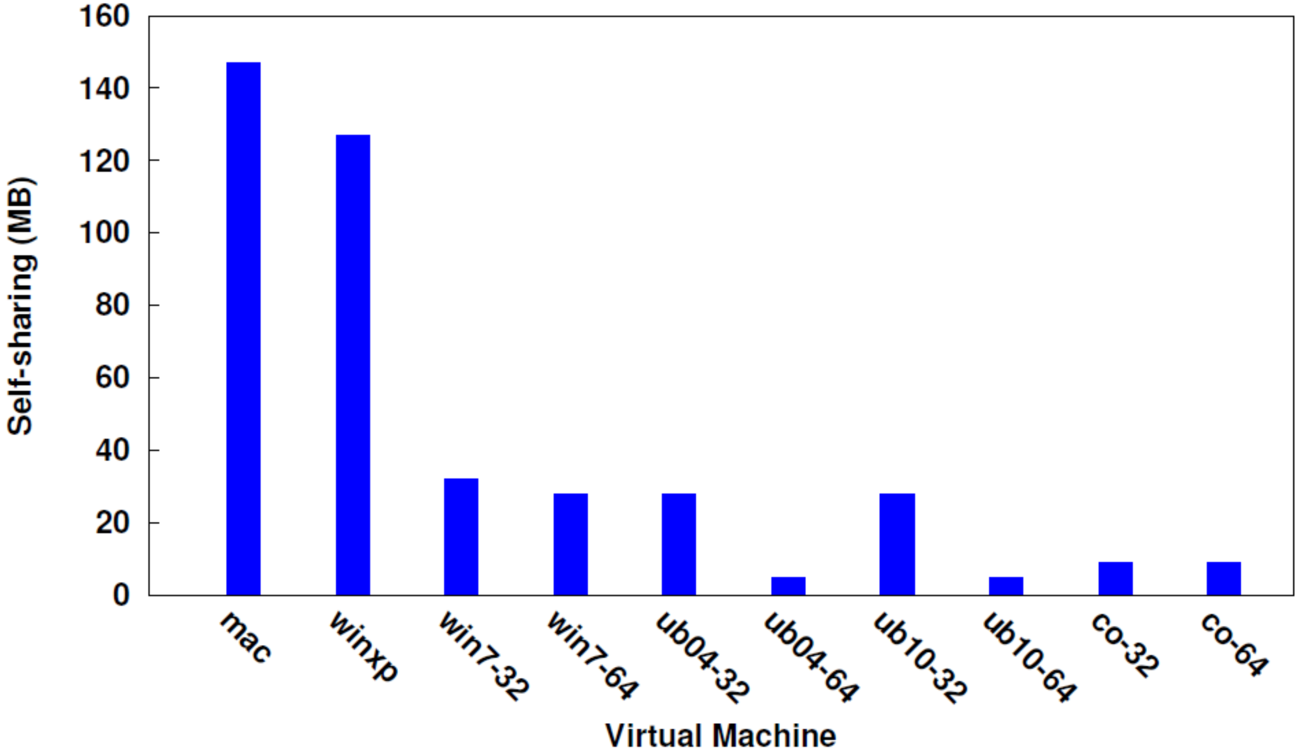
Duplicate data allocations evident in processes

Factors Impacting Sharing

- How do various properties influence sharing?
- Operating system characteristics
  - Family (e.g., Linux or Windows)
  - Version (e.g., Windows XP/7, Ubuntu 10.04/10.10)
  - Architecture (x86 or x64)
- Application setup (LAMP and VDI setups)
- Sharing granularity (number of pages per chunk)
- New OS technologies (e.g., ASLR)

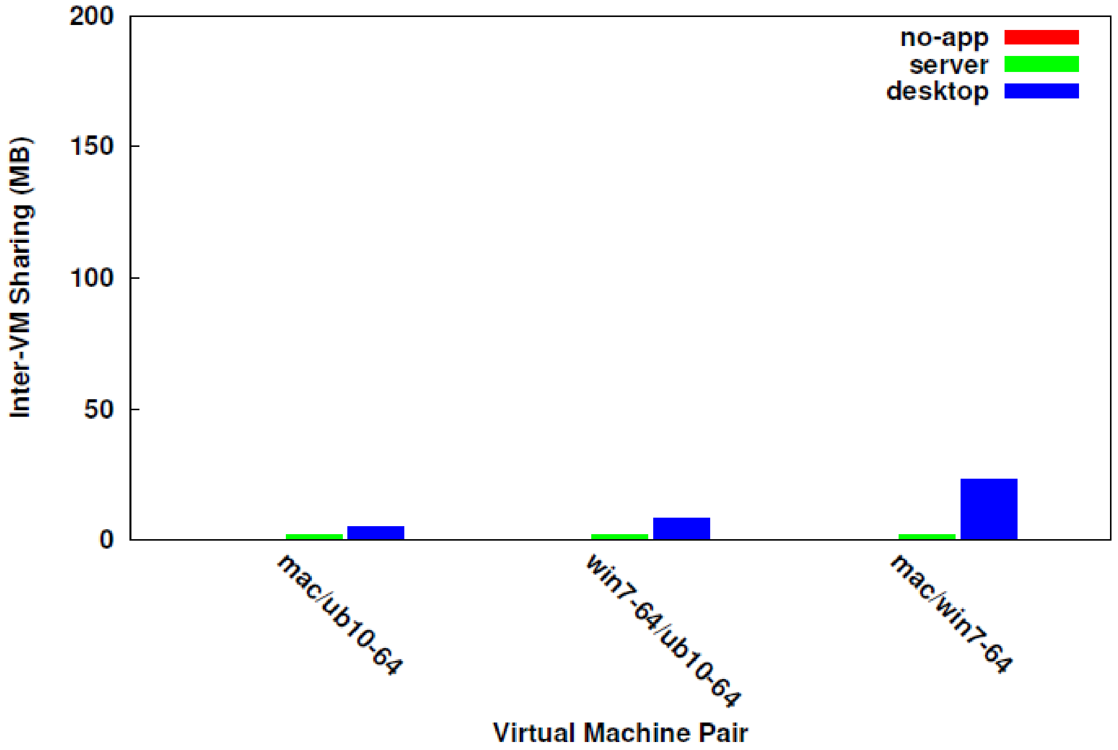
Factors Impacting Sharing

- Self-Sharing across VMs
  - **~100 MB** differences between OS families, major versions (XP/7)
  - **<20 MB** differences between minor versions, architectures

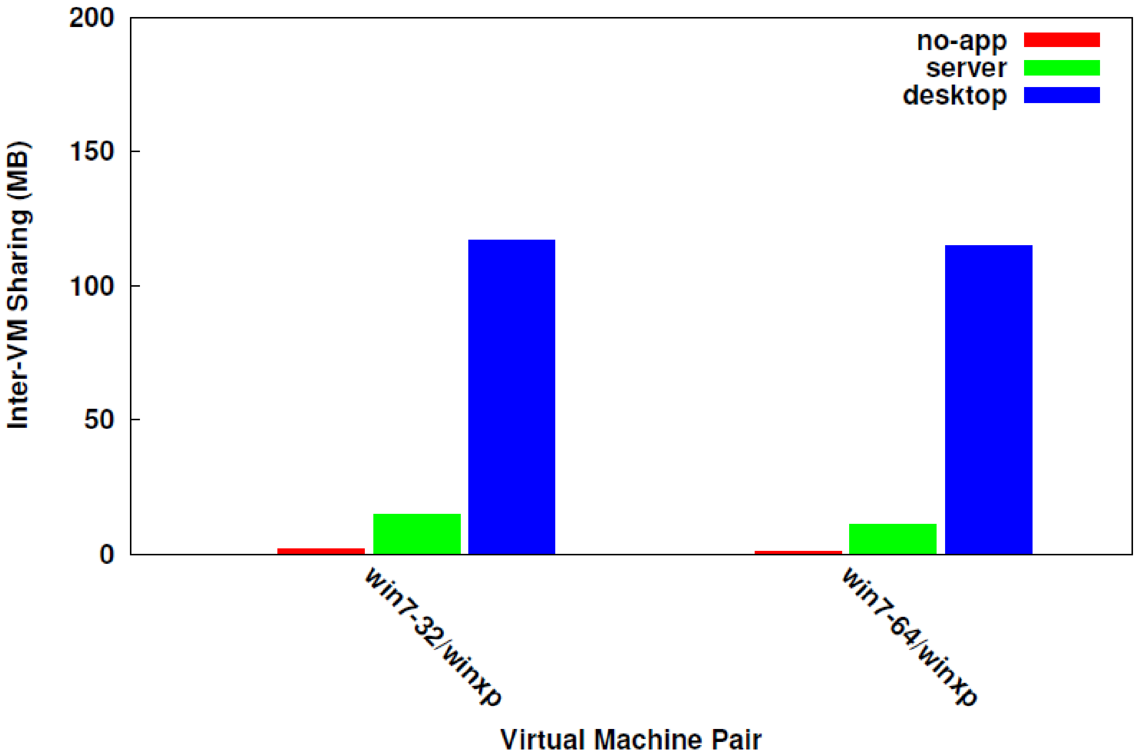


Large self-sharing variations between ‘base’ OSes

Sharing Across VMs



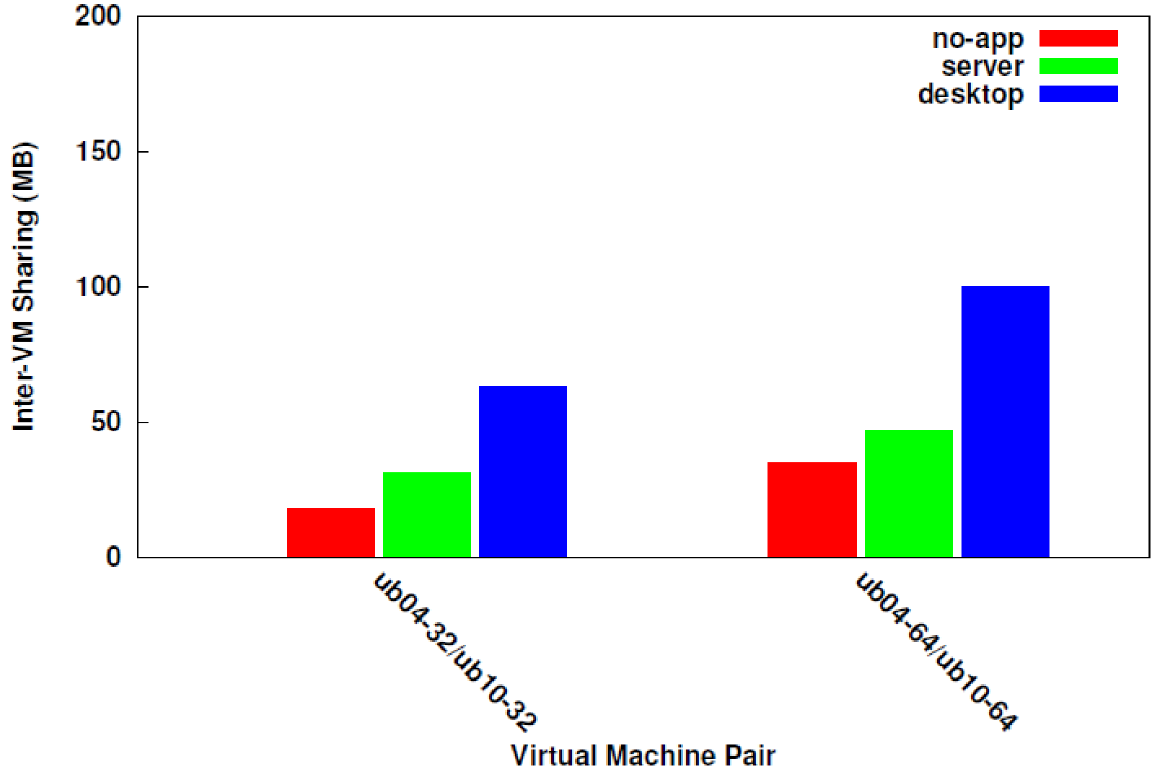
<5 MB across OS families



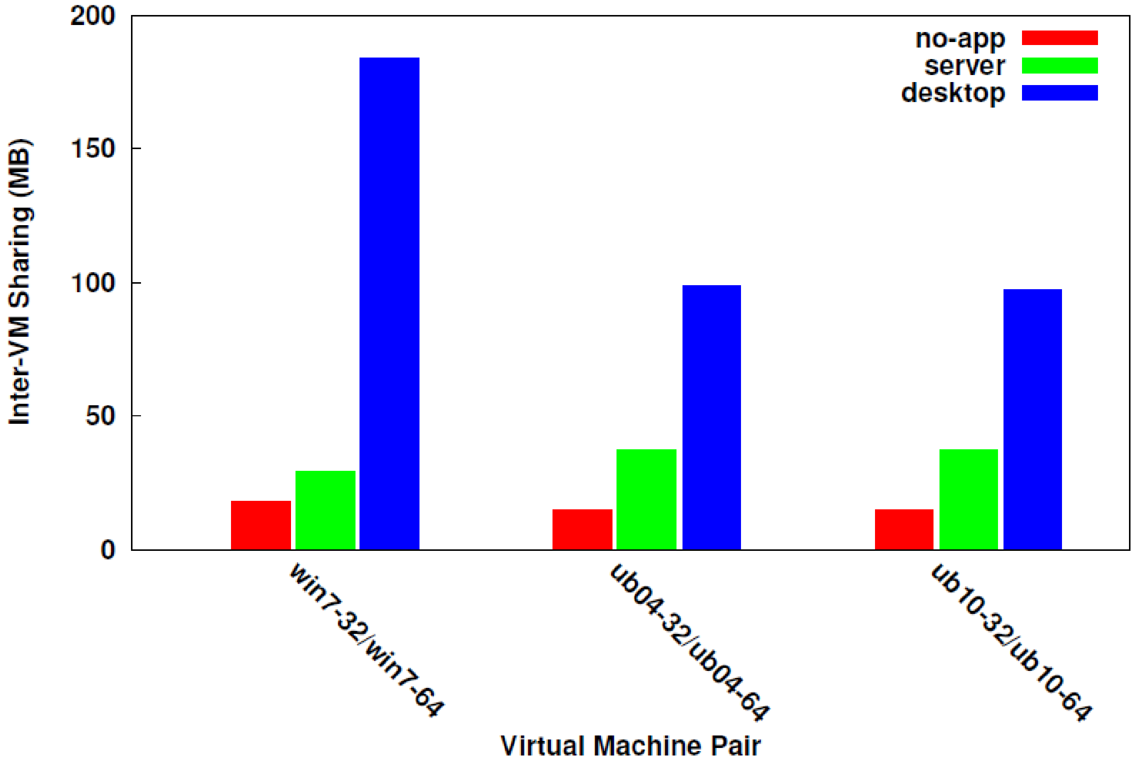
<5 MB base

50+ MB app sharing across major versions

Sharing Across VMs



50+ MB across minor versions

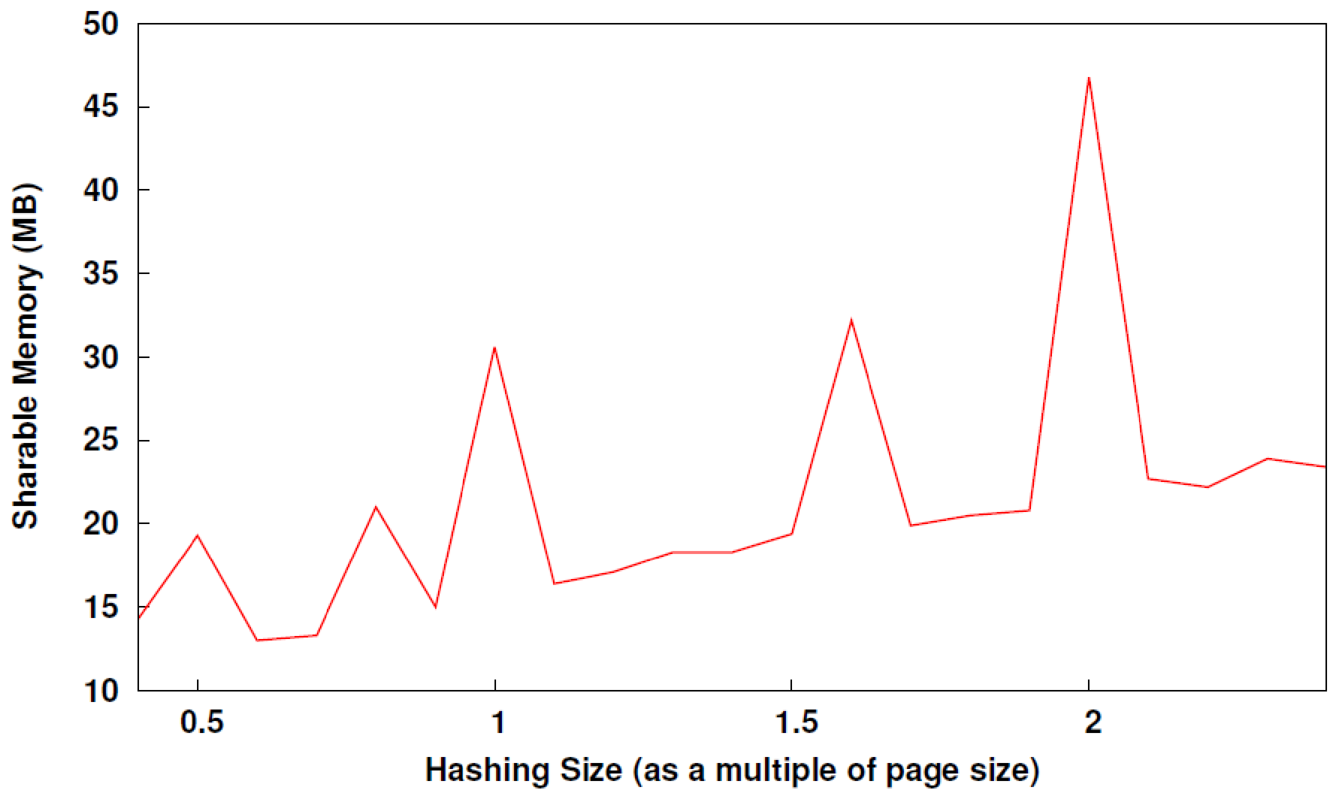


100+ MB across architectures

Hierarchy: family, applications, version, architecture

Sharing Granularity

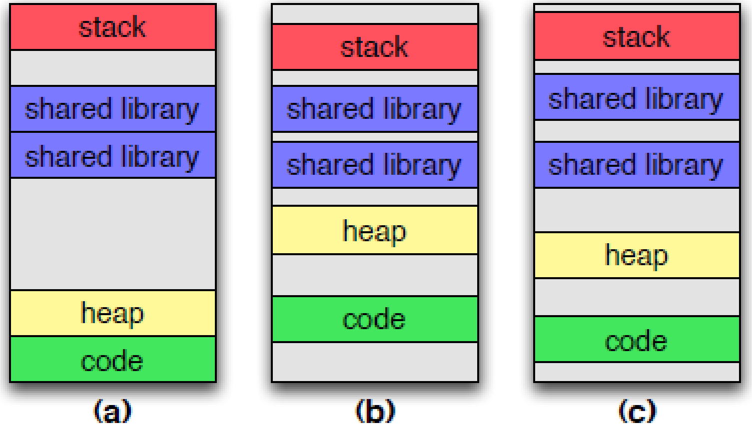
- Share memory chunks of size k (0.4~2.4) pages
  - Only even page divisions provide decent returns
  - Diminishing benefits from smaller chunk sizes



Tradeoff between overhead and sharing potential

Address Space Layout Randomization

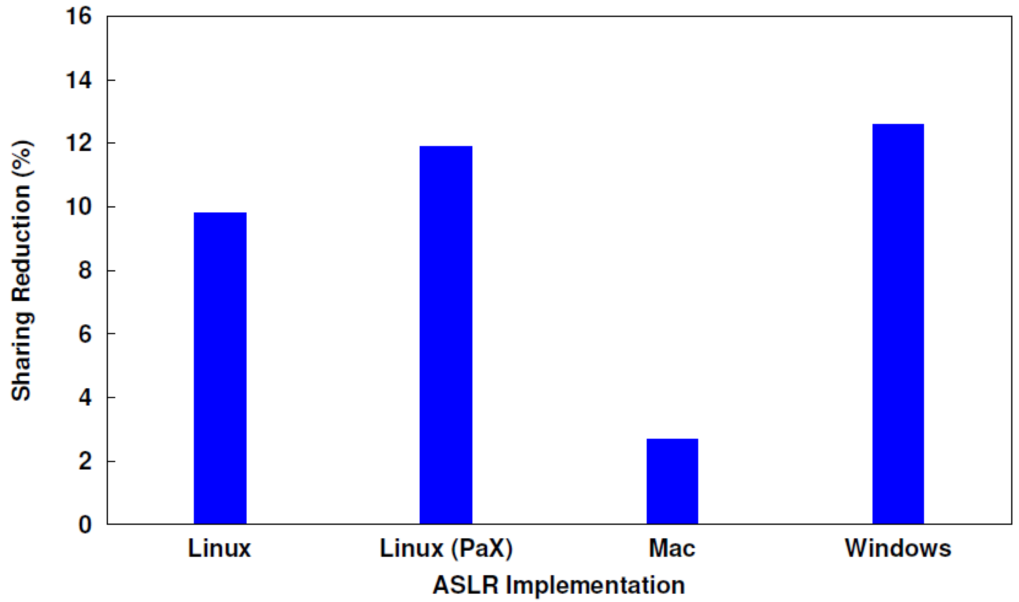
- ASLR scrambles memory to improve system security
  - libraries, code, stack, heap, ...



- Does ASLR have a negative impact on memory sharing?

Address Space Layout Randomization

- Impact of 4 ASLR implementations:
  - Linux: mainline (2.6.32) and PaX
  - Windows 7 (SP1)
  - Mac OS X (Lion)
- Sharing impact of ASLR
  - >10% reduction in three of four cases
  - ‘Better’ (PaX) sharing in Linux worsens impact



ASLR doesn’t prevent sharing but does reduce it

Sharing Factor Observations

- Hierarchy with respect to sharing potential
  - OS family, application setup, OS version, OS architecture
- Platform homogeneity
  - Minimal sharing across heterogeneous systems
  - Significant gains in homogeneous deployments (but still modest absolute levels)
- Finer-grained sharing may be leveraged to improve sharing potential
- OS improvements like ASLR may reduce sharing

Conclusions

- Study into practical issues of page sharing
  - Examined real-world machines and specific sharing scenarios
- Observed real-world sharing around **15%**
  - Significant, but less than expected
  - Largely self-sharing, for which **no virtualization needed**
- Studied a variety of factors impacting sharing
  - Key role of platform homogeneity
  - Varying impact of modifying OS characteristics and applications
  - New technologies may change the impact of sharing