# VIRTUAL MEMORY


***Submitted by(Group-2 )***

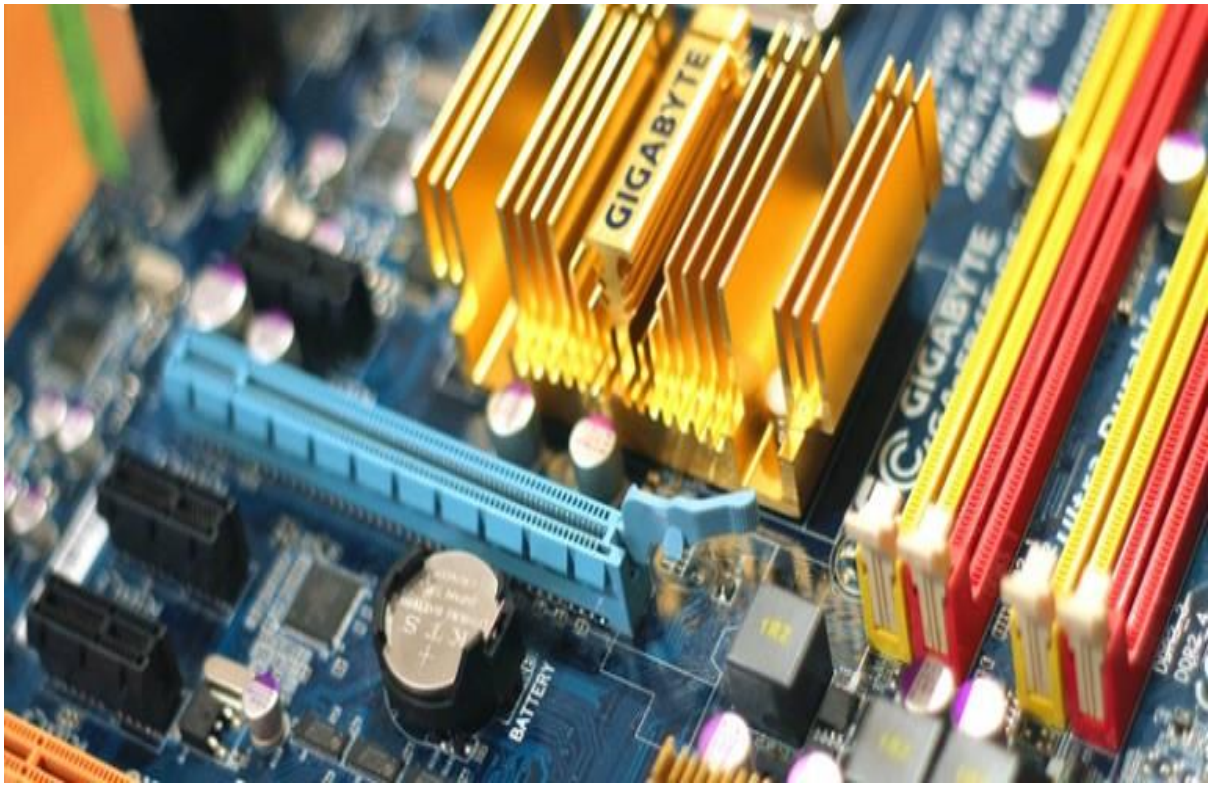

| **Name of Students** | **Examination Roll No.** |
|---|---|
| DEBAPRIYO BHOWMICK | 11600222047 |
| SUBHRANIL BHUNIA | 11600222123 |
| SATAKSHI PODDAR | 11600222096 |
| PRITI SHAW | 11600222071 |

**Under the supervision of**
**MS.SASMITA SUBHADARSINEE CHOUDHURY**
**ASSISTANT PROFESSOR , DEPTARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Department of Computer Science & Engineering,**
**MCKV Institute of Engineering**
**243, G.T. Road(N)**
**Liluah, Howrah – 711204**


**May 2024**

# VIRTUAL MEMORY



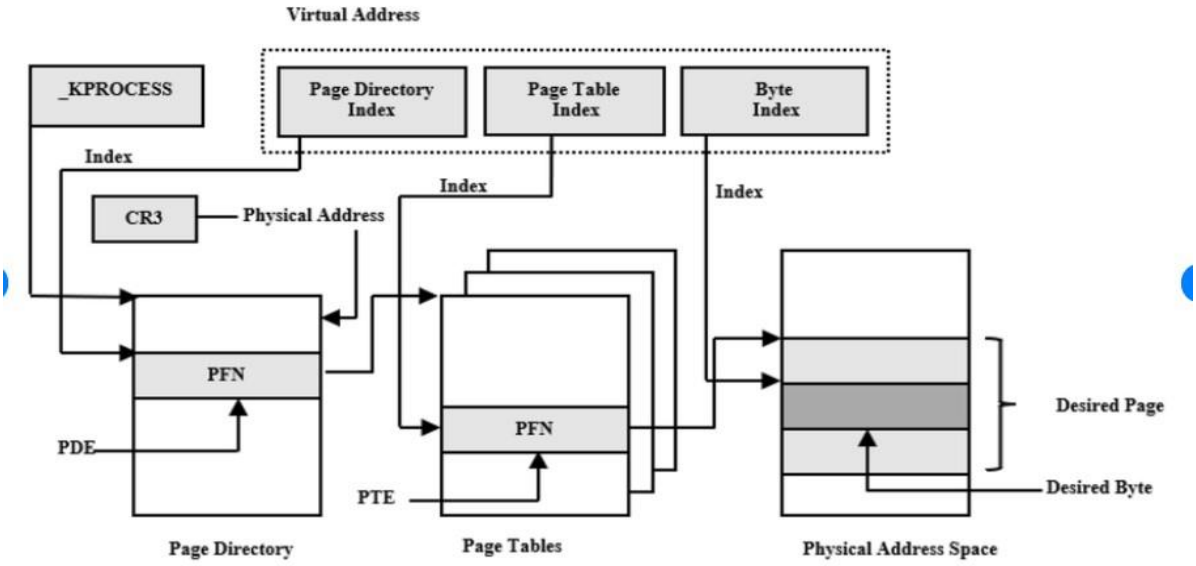
# Enhancing Reliability During Physical Memory Forensics: Strategies and Practices

In recent years, memory forensics has become an essential tool for digital investigators. It allows them to recover valuable evidence that would be lost if traditional methods were used. However, physical memory is constantly changing due to its dynamic and volatile nature. This instability can lead to unreliable results in memory forensics, specifically issues with correctness, completeness, accuracy, or consistency. While current research focuses on memory acquisition and analysis techniques, ensuring reliability throughout the process has received less attention.

This paper addresses the critical issue of reliability in contemporary memory forensics. It aims to establish a comprehensive and systematic approach to guaranteeing reliability during physical memory acquisition. To achieve this, the paper examines various memory management aspects crucial for reconstructing reliable evidence. This includes exploring concepts like physical memory address layout, memory address space, memory address translation, memory access methods, and the internal memory data structures that underpin memory forensics.

The paper then identifies significant challenges encountered during memory forensics and proposes a set of best practices and recommendations for different stages of the process. Implementing these practices can help maximize the reliability of the collected evidence.

Finally, the paper uses experiments with three popular memory acquisition tools (FTK Imager, Magnet RAM Capture, and Belkasoft RAM Capturer) on different sized Windows 10 machines (32 GB and 16 GB) to validate the proposed recommendations. It also outlines important areas for future research in memory forensics reliability.




A low-level representation of virtual-to-physical address translation [43]

**FIGURE 1:- A low representation of virtual-to-physical address traslation**

The field of volatile memory acquisition and analysis is rapidly evolving due to its proven value in forensic investigations. Various memory acquisition tools exist for major operating systems, but their accuracy, speed, and ease of use vary significantly. Continued research and development are crucial to create a single tool that effectively balances these competing demands.

Volatility has become the dominant memory extraction tool, employed by researchers alongside most memory forensic methods. These methods can be categorized as dynamic analysis within a sandbox environment, scanning approaches, or machine learning techniques.

Sandbox-based methods offer a more comprehensive picture of malware behavior compared to scanning methods. However, they can be rendered ineffective by malware equipped with sandbox evasion capabilities. Scanning methods, while fast to implement and commonly used in commercial software, struggle to identify novel malware and provide a limited view of its behavior.

While machine learning algorithms show promise in volatile memory forensics, their results often require verification on larger datasets for conclusive evidence.

# Logged Virtual Memory

Logged virtual memory (LVM) offers a powerful mechanism for tracking write operations within a computer's virtual address space. By maintaining a record of changes made to designated memory regions, LVM proves instrumental for various applications. Here's a closer look at its benefits:

- **Rollback and Persistence:** Applications that require the ability to revert to previous states or ensure data remains intact across system restarts heavily rely on LVM. This includes parallel simulations, where experiments can be rewound to analyze different scenarios, and memory-mapped object-oriented databases, which guarantee data integrity despite potential crashes.
- **Enhanced Debugging:** LVM serves as a valuable tool for debugging processes. By logging memory writes, developers gain insights into program behavior and can pinpoint the exact memory locations where issues arise. This detailed log allows for more efficient debugging compared to traditional methods.
- **Distributed System Consistency:** Maintaining consistency across geographically distributed systems presents a challenge. LVM can be leveraged to track changes made to shared data and ensure all replicas remain synchronized. This facilitates data integrity and avoids inconsistencies that could disrupt system operation.

**Beyond Foundational Applications:**

While the aforementioned applications showcase the core functionalities of LVM, its potential extends even further. Here are some emerging use cases:

- **Security Analysis:** LVM can be employed to monitor memory accesses for suspicious activity. By tracking writes to specific regions, security professionals can detect potential malware attempting to tamper with critical system components.
- **Performance Optimization:** Analyzing LVM logs can reveal memory access patterns within applications. This information can be used to optimize memory usage and improve overall system performance.
- **Forensic Investigations:** In the event of a system breach or security incident, LVM logs can provide valuable forensic evidence. By examining written data, investigators can reconstruct the sequence of events and identify the source of the attack.

**Technical Considerations and Future Directions:**

This paper explores LVM as an extension to existing virtual memory systems. Our prototype implementation demonstrates that LVM can be integrated with minimal modifications. Furthermore, our performance measurements highlight the efficiency of LVM compared to alternative logging techniques.

Looking ahead, research on LVM can explore further optimizations for specific use cases. Additionally, integrating LVM with hardware memory management units can potentially enhance performance and security. By fostering collaboration between hardware and software engineers, LVM can be positioned as a cornerstone technology for reliable and secure computing.

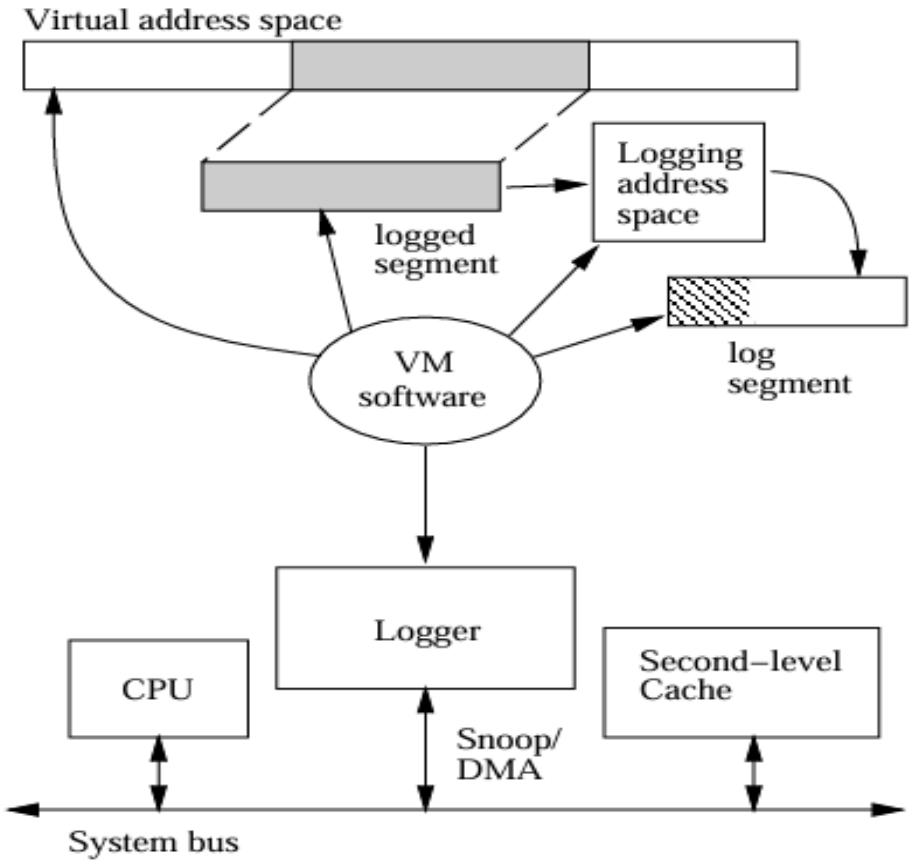



Figure 4: Prototype Implementation Block Diagram

**FIGURE 2:- prototype Implementation Block Diagram**

Logged virtual memory (LVM) enhances a computer's operating system by extending its virtual memory system. This extension allows for efficient logging of updates made to specific regions within virtual memory. This logging capability proves valuable across a broad spectrum of applications, encompassing distributed shared memory systems and debugging tools.

One of the key benefits of LVM is its minimal impact on applications. It necessitates minimal modifications to existing applications and introduces negligible runtime overhead. This is because writing to logged memory regions incurs almost no additional burden on the writing process itself. From an implementation standpoint, LVM only requires a single addition to the virtual memory interface: the ability to associate a dedicated log segment with a specific virtual memory region that needs to be tracked.

Our measurements indicate that LVM delivers significant performance improvements while maintaining a modest hardware and operating system complexity. This is further supported by our functional prototype. Compared to existing alternatives, LVM emerges as the superior choice.

Generating logs solely through the operating system, without hardware support (essentially trapping every write operation), becomes impractical except for scenarios with very low write activity. User-mode logging approaches, achieved through automatic code instrumentation (involving compiler or language support, or through sandboxing), exhibit lower performance and introduce additional overhead for application processes. This software-centric approach is akin to implementing virtual memory entirely in software, without hardware support. While theoretically possible, it introduces a dependency on a large and complex software infrastructure for code instrumentation. Additionally, it assumes a limited rate of memory access. This software-intensive approach seems unjustified for the sole purpose of avoiding a modest amount of additional hardware.

Finally, manually annotating each potential write operation for logging is a time-consuming and error-prone process. It also shares some of the overhead drawbacks associated with the automatic instrumentation approach.

.

# A novel approach to enhance distributed virtual memory

Distributed virtual memory (DVM) has emerged as a promising approach to enhance system memory performance, particularly in clustered computing environments. This paper delves into practical challenges associated with DVM and explores avenues for further performance optimization. We propose a novel distributed algorithm for DVM management that leverages cluster caching to bolster existing DVM techniques.

The core elements of our algorithm encompass address translation, page out policies, and page replacement strategies. To rigorously evaluate its effectiveness, we conduct performance assessments across diverse testbeds under varying system loads. The results demonstrate the superiority of our proposed technique compared to both conventional DVM implementations and traditional virtual memory (CVM) systems. Notably, our algorithm achieves significant reductions in page faults – surpassing practical DVM by 15% and CVM by an impressive 25%.

**Beyond the Abstract: Expanding on the DVM Landscape**

This paper extends the discussion on DVM beyond the core algorithm. Here's an exploration of the broader landscape:

- **Motivation for DVM:** Traditional virtual memory systems often encounter limitations in handling memory demands within large-scale distributed computing environments. DVM addresses this challenge by enabling shared memory access across geographically dispersed computing nodes (often clustered together), effectively creating a unified virtual address space. This fosters improved memory utilization and potentially reduces overall system memory requirements.
- **Challenges and Considerations:** While DVM offers advantages, it also introduces complexities. Network communication overhead can become a bottleneck, potentially negating performance gains. Additionally, maintaining consistency of data across distributed nodes is crucial to ensure data integrity. Our proposed algorithm specifically addresses these challenges by optimizing page management and leveraging cluster caching to minimize network traffic.
- **Future Directions:** Research on DVM continues to evolve. Future work may explore:

- **Integration with Hardware:** Investigating how DVM can be tightly coupled with hardware memory management units to potentially offload processing tasks and further enhance performance.
- **Security Considerations:** Addressing security concerns that arise when managing shared memory across distributed systems. This might involve implementing robust authentication and access control mechanisms.
- **Heterogeneous Systems:** Exploring the applicability of DVM in scenarios involving heterogeneous computing environments, where nodes may possess varying hardware and software configurations.

By addressing these considerations and delving into promising research directions, DVM has the potential to revolutionize memory management in distributed computing, paving the way for a future of scalable, efficient, and secure distributed systems.

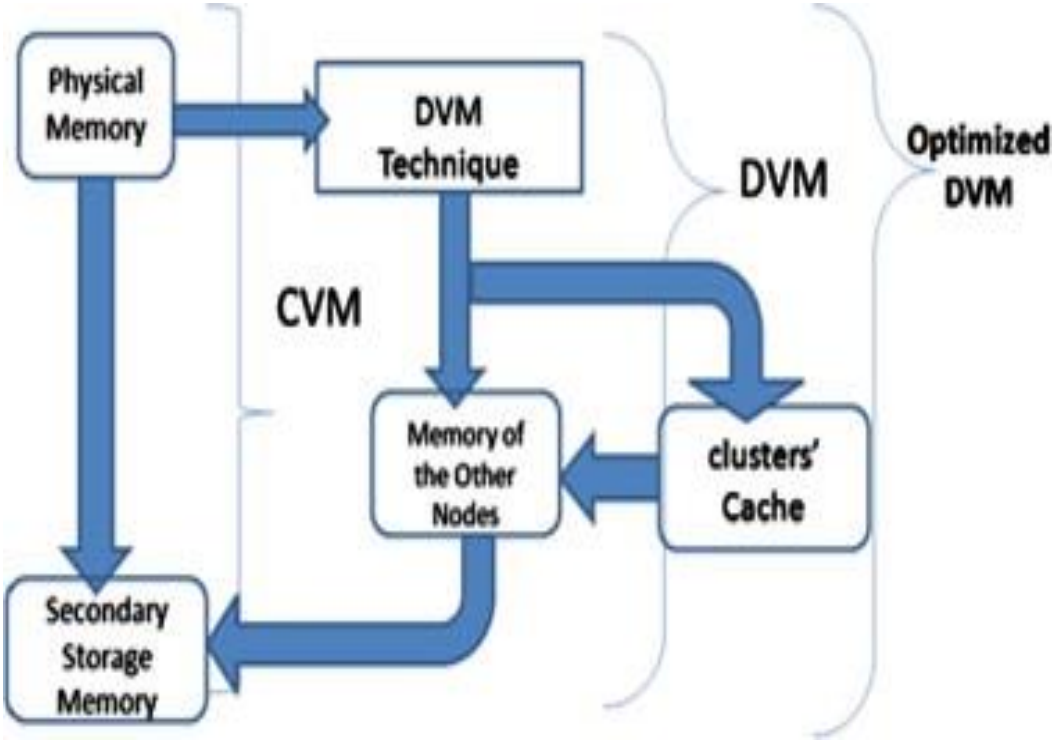



**FIGURE 3:- Overview of CVM, DVM and optimized DVM.**

This paper has convincingly demonstrated the effectiveness of distributed virtual memory (DVM) systems in optimizing memory performance for distributed computing environments. We went beyond theoretical advantages by proposing a practical implementation approach through our novel distributed algorithm.

This algorithm stands out for its ability to dynamically assess system memory status and distribute pages across cluster nodes, ensuring balanced load distribution. The utilization of cluster-specific caches further enhances performance by minimizing network traffic and optimizing data access patterns.

The evaluation results were highly promising, with our approach achieving a significant reduction in page faults – 15% compared to practical DVM solutions and a remarkable 25% compared to conventional virtual memory (CVM) systems.

**Beyond the Current Work: Exploring the Potential of DVM**

While this paper lays a solid foundation, the future of DVM holds immense potential. Here, we delve into some exciting areas for further exploration:

- **Cache Coherence Mechanisms:** Robust cache coherence protocols are essential for maintaining data consistency across distributed caches. These protocols ensure that all nodes possess the latest version of shared data, preventing inconsistencies and safeguarding data integrity in a multi-cache environment. Future research can focus on implementing and optimizing such protocols within the DVM framework.
- **Adaptive Distributed Algorithms:** Developing adaptive algorithms capable of dynamically adjusting page management strategies based on real-time system conditions is a promising area of research. These algorithms could consider factors such as network traffic patterns, individual node memory usage, and application-specific access patterns. By adapting to these dynamic conditions, the DVM system can optimize performance for diverse workloads and system configurations, maximizing efficiency across various scenarios.
- **Heterogeneous Cluster Integration:** Expanding the applicability of DVM to encompass heterogeneous clusters is a crucial step towards broader adoption. In such clusters, nodes may possess varying hardware architectures and operating systems. The DVM system would need to adapt its page management strategies to accommodate these differences. Future work can explore techniques to ensure efficient memory utilization and balanced load distribution across such heterogeneous environments.
- **Security Enhancements in DVM:** Security considerations in DVM systems demand further exploration. Implementing robust access control mechanisms to regulate which nodes have permission to access specific memory pages is essential. Additionally, securing communication protocols is crucial to safeguard data integrity during network transfers

between nodes. Future research can focus on developing and integrating robust security measures into DVM systems.

By actively pursuing these promising research directions, we can unlock the full potential of DVM and usher in a new era of highly performant, scalable, and secure distributed computing environments.

## Efficient Virtual Memory for Big Memory Servers

This paper sheds light on a critical performance bottleneck for "big-memory" server workloads like databases, in-memory caches, and graph analytics systems. Our analysis reveals that these workloads suffer significant performance penalties due to page-based virtual memory. Even when utilizing large page sizes, these applications can incur as much as 10% of their execution cycles wasted on TLB (Translation Lookaside Buffer) misses.

**Understanding the Bottleneck:**

The TLB is a high-speed cache that stores recently used virtual-to-physical address translations. When a program needs to access memory, it first checks the TLB. If the translation is found, the access is significantly faster compared to consulting the larger page table in memory. However, TLB misses occur when the required translation isn't present in the cache, forcing the system to fetch it from the page table, incurring a performance penalty.

**Why Big-Memory Workloads are Impacted:**

Our research highlights a key characteristic of big-memory workloads: they often require frequent access to large, contiguous data structures. These data structures, such as database buffer pools and in-memory key-value stores, typically exhibit read-write permission patterns across most pages. Additionally, these workloads are often provisioned with sufficient memory to avoid swapping data to disk and rarely require the full flexibility offered by traditional page-based virtual memory.

**Introducing Direct Segments: A Novel Approach**

To address the TLB miss overhead and unlock performance improvements for big-memory workloads, we propose a novel technique called "direct segments." This approach involves mapping a designated portion of a process's linear

virtual address space with a direct segment, while the remaining virtual address space continues to be managed using page-based virtual memory.

**Benefits of Direct Segments:**

- **Reduced TLB Misses:** Direct segments eliminate the possibility of TLB misses for critical data structures mapped within them. This is achieved by utilizing minimal hardware, specifically base, limit, and offset registers per core, to directly map contiguous virtual memory regions to contiguous physical memory.
- **Performance Gains:** By eliminating TLB misses for frequently accessed data structures, direct segments have the potential to significantly reduce the execution time wasted on these operations. Our experiments demonstrate reductions to less than 0.5% of execution time wasted on TLB misses for the workloads we tested.
- **Flexibility:** The proposed approach maintains flexibility. Memory mapped by a direct segment can be converted back to traditional paging when needed, allowing for dynamic adjustments based on workload requirements.

**Implementation and Future Directions:**

This paper details the prototyping of direct-segment software support for x86-64 architectures running on Linux. Additionally, we present an emulation of direct-segment hardware to assess its potential benefits.

While our initial research shows promising results, further exploration is warranted. Future work could investigate:

- **Integration with Hardware:** Collaborating with hardware designers to explore the feasibility and potential performance improvements of incorporating dedicated direct segment hardware support.
- **Impact on Different Architectures:** Evaluating the effectiveness of direct segments on various hardware architectures beyond x86-64.
- **Workload Diversity:** Testing and refining the direct segment approach across a broader range of big-memory workloads to assess its generalizability.

By continuing research along these lines, direct segments have the potential to become a valuable tool for optimizing performance in big-memory computing environments. This approach could lead to significant performance gains for a wide range of server workloads that rely heavily on large, frequently accessed data structures.

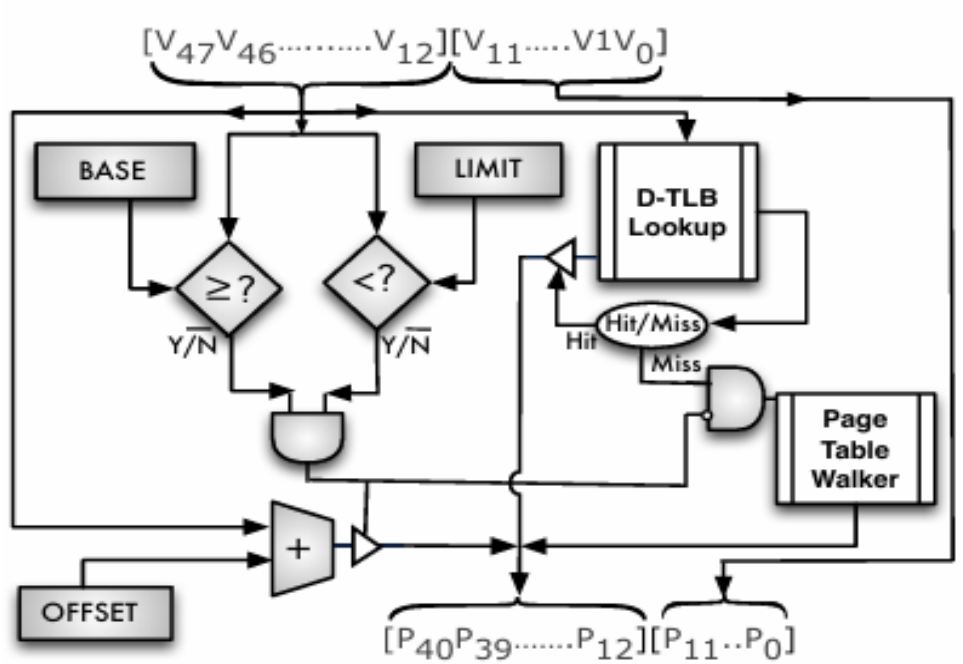



**FIGURE 4:- Logical view of address translation with direct seg ment. Added hardware is shaded.**

**Understanding the Impact of TLB Misses:**

The TLB acts as a high-speed cache, storing recently used translations between virtual and physical memory addresses. When a program needs to access memory,. If the translation is found, the access is significantly faster compared to consulting the larger page table in memory. However, TLB misses occur when the required translation isn't present in the cache, forcing the system to fetch it from the page table, incurring a performance penalty.

**Big-Memory Workloads and Inefficiencies:**

Our research sheds light on a specific characteristic of big-memory workloads: they frequently rely on accessing large, contiguous data structures. These data structures, such as database buffer pools and in-memory key-value stores, typically exhibit read-write permission patterns across most pages. Interestingly, these workloads are often provisioned with sufficient memory to avoid swapping data to disk and rarely require the fine-grained memory protection capabilities offered by traditional page-based virtual memory.

**Reduced TLB Misses:** Primary regions offer a significant advantage for big-memory workloads. By mapping large anonymous memory regions (allocated based on available physical memory) to a primary region, we can eliminate almost all TLB misses for frequently accessed data structures within this region.

This is achieved through the use of a direct segment, which utilizes minimal hardware resources (base, limit, and offset registers per core) to directly map contiguous virtual memory regions to contiguous physical memory, bypassing the need for TLB lookups.

- **Performance Gains:** Our experiments demonstrate the effectiveness of primary regions. By eliminating TLB misses for frequently accessed data structures, this approach has the potential to significantly reduce the execution time wasted on these operations, leading to substantial performance improvements.
- **Compatibility:** The proposed approach maintains compatibility with existing systems. While primary regions offer significant benefits for specific memory access patterns, other virtual addresses can continue to use conventional page-based virtual memory, ensuring seamless integration with existing applications.

**Implementation and Future Directions:**

This paper details the development of a prototype for primary regions on a Linux 2.6.32 system. Additionally, we present a hardware approximation to assess the potential performance improvements achievable with dedicated hardware support for direct segments.

While our initial research shows promising results, further exploration is warranted. Future work could investigate:

- **Hardware Integration:** Collaborating with hardware designers to explore the feasibility and potential performance benefits of incorporating dedicated hardware support for direct segments within the primary region concept.
- **Workload Diversity:** Testing and refining the primary region approach across a broader range of big-memory workloads to assess its generalizability. This would help identify the specific workload types that benefit most from this technique and potentially lead to workload-aware optimizations.
- **Security Implications:** Investigating the potential security implications of primary regions. While maintaining compatibility with existing systems is important, it's crucial to ensure that the introduction of primary regions doesn't introduce new vulnerabilities or weaken existing security measures. Techniques to mitigate any identified risks would need to be explored.

# REVISITING VIRTUAL MEMORY

Virtual memory, a cornerstone of modern computer systems, relies on paging for memory management. However, the needs and constraints of virtual memory have evolved significantly since the introduction of translation lookaside buffers (TLBs). This paper argues for a reevaluation of virtual memory management due to three key changes:

1. **Exponential Growth in Physical Memory Sizes:** Physical memory capacities have grown dramatically, exceeding a million-fold increase.
2. **Workloads Optimized for In-Memory Operations:** Modern workloads are often sized to avoid data swapping between memory and slower secondary storage, reducing the need for paging functionality.
3. **Energy Efficiency as a Top Priority:** Energy consumption is now a primary design concern, and frequent TLB accesses contribute significantly to overall energy usage.

**The Problem: TLB Bottleneck and Energy Waste**

Despite these changes, level-one TLBs (which cache recently used address translations) have remained relatively static in size and are still accessed on every memory reference. This results in two main issues:

- **TLB Misses:** Large workloads experience significant performance penalties due to wasted execution time spent resolving TLB misses.
- **Energy Inefficiency:** Frequent TLB accesses consume considerable energy, further amplified by the large and highly associative nature of L1 caches required for page-sized data access.

**The Solution: Rethinking Virtual Memory Management**

This thesis proposes three solutions to address the limitations of current virtual memory management and improve overall system performance and energy efficiency:

1. **Direct Segments for Big-Memory Workloads:** Many big-memory workloads allocate most of their memory upfront and don't require paging functionality. Direct segments introduce a hardware-OS cooperative mechanism that bypasses paging for a designated portion of a process's virtual address space. This approach eliminates nearly 99% of TLB misses for a wide range of big-memory workloads, significantly reducing latency overhead.

2. **Opportunistic Virtual Caching (OVC):** Frequent TLB accesses consume significant energy. OVC proposes hardware-OS modifications that expose energy-efficient virtual caching as a dynamic optimization. This approach reduces TLB lookup energy by 94-99% and L1-cache lookup energy by 23% across various workloads.
3. **Merged-Associative TLB for Efficient Large Page Usage:** Large pages can be more effective than direct segments in reducing TLB misses for workloads with frequent memory allocation/deallocation. However, current chip designs often partition TLB resources statically among various page sizes, hindering the performance benefits of large pages. This paper proposes a merged-associative TLB design that dynamically aggregates TLB resources across page sizes, avoiding performance bottlenecks and reducing TLB miss rates by up to 45%.

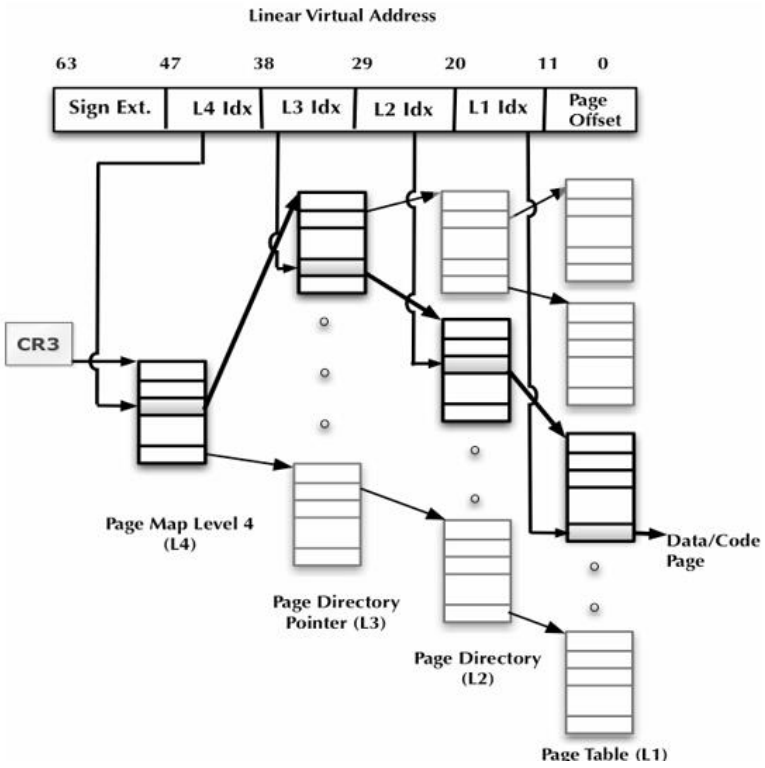



**FIGURE 5 :-. Page table walk in x86-64 with 4KB page size.**

While large pages are a popular way to cut down on TLB miss overhead, supporting them throws a wrench into hardware TLB design. Current processors handle multiple page sizes in two main ways:

1. **Fully Associative TLBs:** These offer maximum flexibility but are tough to scale and often lag behind set-associative TLBs in terms of speed.

2. **Split TLBs:** These use separate set-associative TLBs for each supported page size. This avoids the limitations of fully associative TLBs, but comes with its own issues:
    - **Unpredictable Performance:** Large pages might actually hurt performance with split TLBs.
    - **Resource Waste:** Separate TLBs for each size can lead to wasted resources if workloads favor one size over another.

**The Merged-Associative TLB: A Better Way**

This paper proposes the merged-associative TLB as a solution that fixes the problems of split TLBs without the downsides of fully associative ones.

**Benefits of the Merged-Associative TLB:**

- **Consistent Performance with Large Pages:** Unlike split TLBs, the merged-associative TLB ensures performance doesn't suffer when using large pages.
- **Reduced TLB Miss Rates:** This design lets resources be dynamically allocated across different page sizes. This can significantly reduce TLB miss rates when workloads favor a particular page size.

**The merged-associative TLB essentially combines the flexibility of handling multiple page sizes with efficient resource utilization and consistent performance, especially for workloads that benefit from large pages.**

# *REFERENCES*


[1] Kirmani, M. S., & Banday, M. T. (2024). Enhancing Reliability During Physical Memory Forensics: Strategies and Practices. *SN Computer Science*, *5*(1), 201.

[2] Cheriton, D. R., & Duda, K. J. (1995, December). Logged virtual memory. In *Proceedings of the fifteenth ACM Symposium on Operating systems principles* (pp. 26-38).

[3] Bqerat, A. H., Alouneh, S., & Mohd, B. J. (2012). A novel approach to enhance distributed virtual memory. *Computers & Electrical Engineering*, *38*(2), 388-398.

[4] Basu, A., Gandhi, J., Chang, J., Hill, M. D., & Swift, M. M. (2013). Efficient virtual memory for big memory servers. *ACM SIGARCH Computer Architecture News*, *41*(3), 237-248.

[5] Basu, A. (2013). *Revisiting virtual memory* (Doctoral dissertation, The University of Wisconsin-Madison).