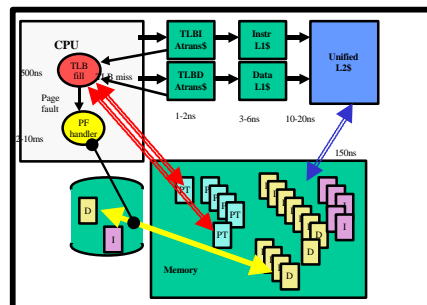


Main memory and Its virtual extension

- Or -

how seven levels of caching can make your life miserable

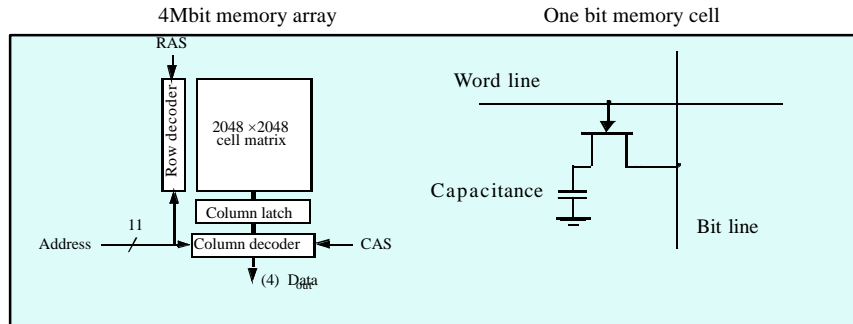


Main memory characteristics

Performance of main memory:

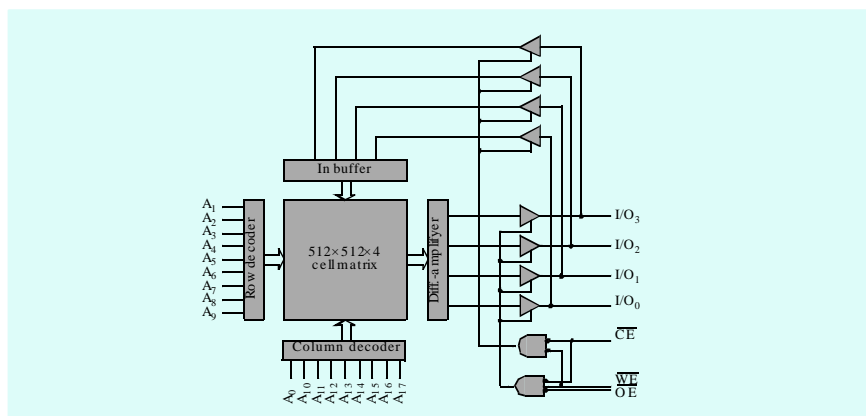
- *Access time*: time between address is latched and data is available (~50ns)
- *Cycle time*: time between requests (~100 ns)
- *Total access time*: from ld to REG valid (~150ns)
- Main memory is built from **DRAM**: Dynamic RAM
- 1 transistor/bit ==> more error prone and slow
- Refresh and precharge
- Cache memory is built from **SRAM**: Static RAM
 - about 4-6 transistors/bit

DRAM organization



- The address is multiplexed Row/Address Strobe (RAS/CAS)
- Thin organizations x4 or x1 to decrease pin load
- Refresh of memory cells decreases bandwidth
- Bit-error rate creates a need for error-correction (ECC)

SRAM organization



- Address is typically not multiplexed
- Each cell consists of about 4-6 transistors
- Wider organization (x18 or x36), typically few chips
- Often parity protected (ECC becoming more common)

Error Detection and Correction

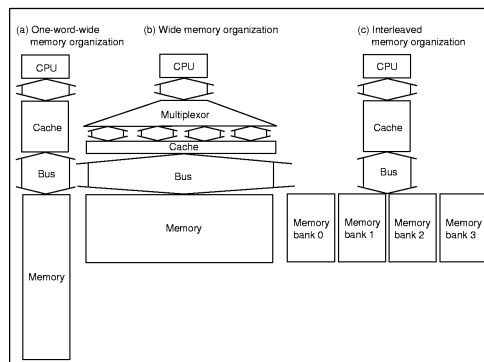
Error-correction and detection

- E.g., 64 bit data protected by 8 bits of ECC
 - Protects DRAM and high-availability SRAM applications
 - Double bit error detection
 - Chip kill detection (all bits of one chip stuck at all-1 or all-0)
 - Single bit correction
 - Need “memory scrubbing” in order to get good coverage

Parity

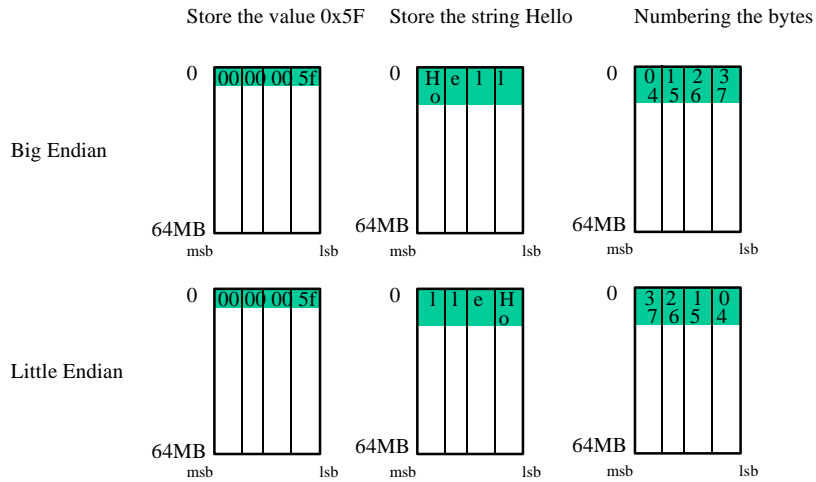
- E.g., 8 bit data protected by 1 bit parity
 - Protects SRAM and data paths
 - Single-bit crash and burn detection

Improving main memory performance

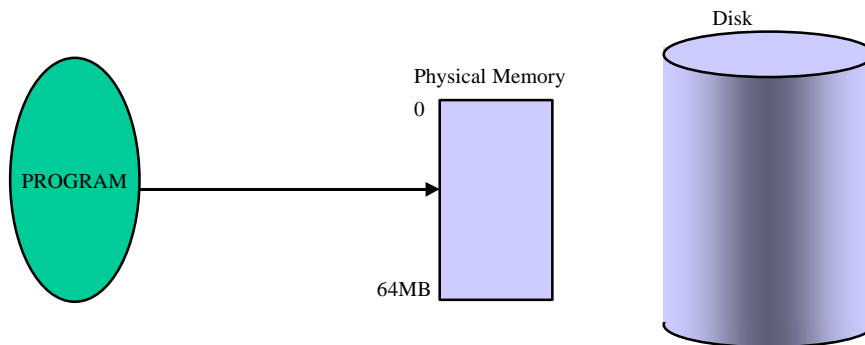


- Page-mode => faster access within a small distance
- Improves bandwidth per pin -- not time to critical word
- Single wide bank improves access time to the complete CL
- Multiple banks improves bandwidth

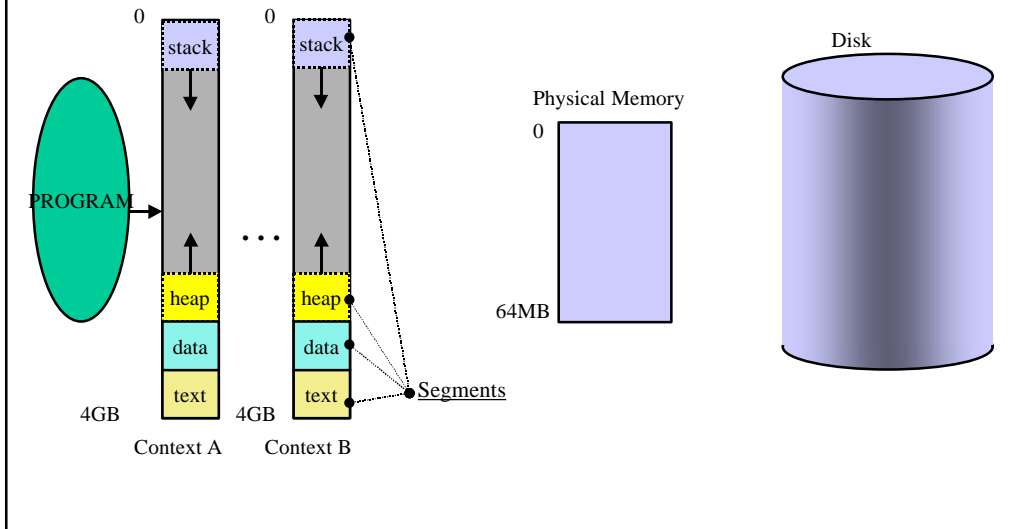
Physical memory, little endian



Physical Memory



Virtual and Physical Memory



Virtual memory-- closing the widest technology gap

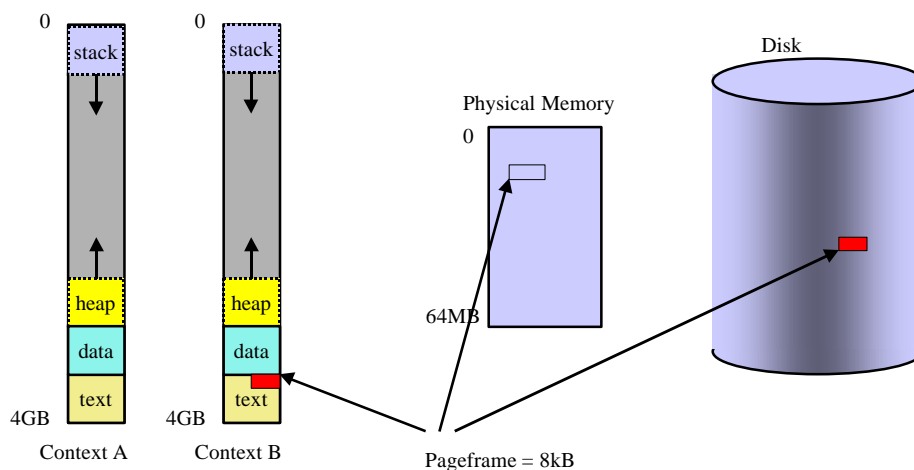
Several reasons to use VM:

- Large sequential address space
- Hide the memory hierarchy from the application
- Create a new namespace for the compiler
- Several processes sharing the same physical memory
- Protection of memory from fatal bugs
- Security of information
- Detect and kill wild processes

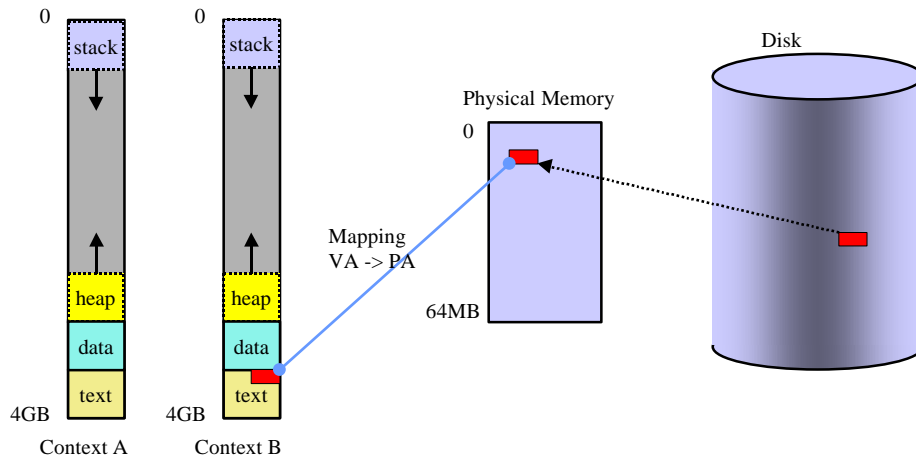
Virtual memory—concepts

- The virtual address space is divided into *pages*
- The physical Address Space (main memory) is divided into *page frames*
- Accessing a page not in main memory is called a *page fault*
- Pages not in main memory are stored on *disk*
- The CPU uses *virtual addresses*
- An *address translation mechanism* is needed

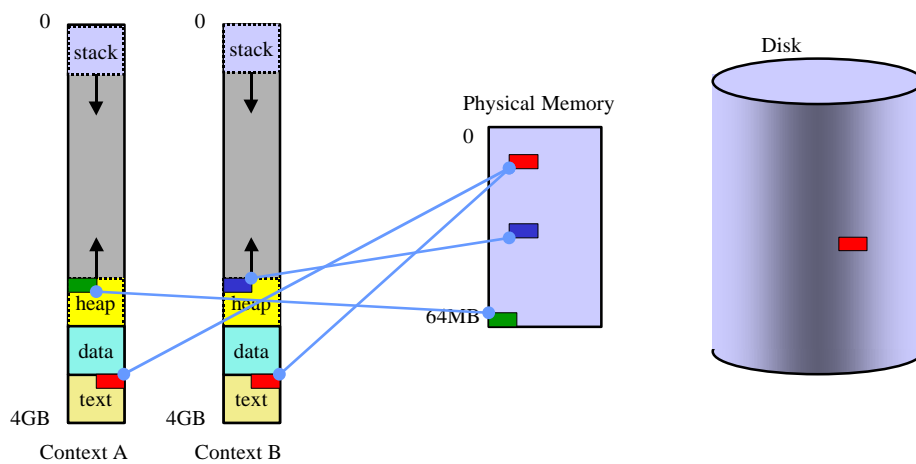
Virtual and Physical Memory



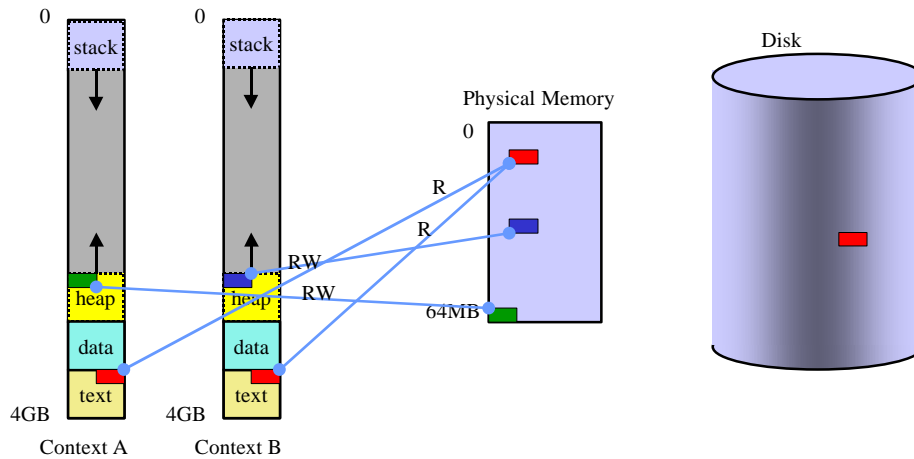
Virtual and Physical Memory



Virtual and Physical Memory

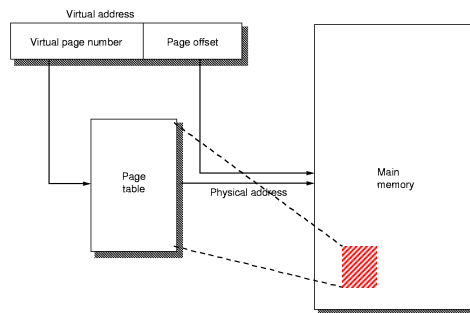


More Than Just Translation: Protection



VM: Block identification

Use a page table stored in main memory:



- Suppose 8 Kbyte pages, 48 bit virtual address
- Page table takes $2^{48}/2^{13} * 4B = 2^{37} = 128 \text{ Gbyte!!!}$

• Solutions:

- **Only one entry per physical page is needed**
- Multi-level page table (dynamic)
- Inverted page table (~hashing)

How do we make the page table lookup fast?

VM: Page replacement

Most important: *minimize number of page faults*

- Page replacement strategies:
 - FIFO—First-In-First-Out
 - LRU—Least Recently Used
 - Approximation to LRU
 - Each page has a *reference bit* that is set on a reference
 - The OS periodically resets the reference bits
 - When a page is replaced, a page with a reference bit that is not set is chosen

VM: Write strategy

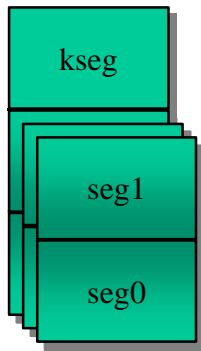
Write back or Write through?

- *Write back!*
- Write through is impossible to use:
 - Too long access time to disk
 - The write buffer would need to be *prohibitively* large
 - The I/O system would need an extremely high bandwidth

Address translation

- Example: The *Alpha 21064*

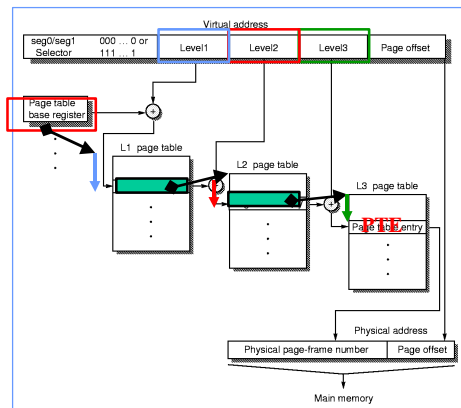
Segment is selected by bit 62 & 63 in addr.



Kernel segment
Used by OS.
Does not use virtual memory.

User segment 1
Used for stack.

User segment 0
Used for instr. & static data & heap

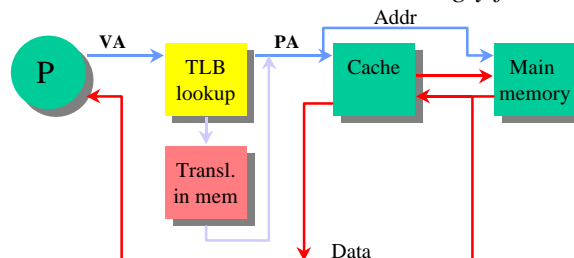


Fast address translation

How do we avoid three extra memory references for each original memory reference?

- Store the most commonly used address translations in a cache—**Translation Look-aside Buffer (TLB)**

==> *The caches rears their ugly faces again!*



What is the capacity of the TLB

Typical TLB size = 0.5 - 2kB

Each translation entry 4 - 8B ==> 32 - 500 entries

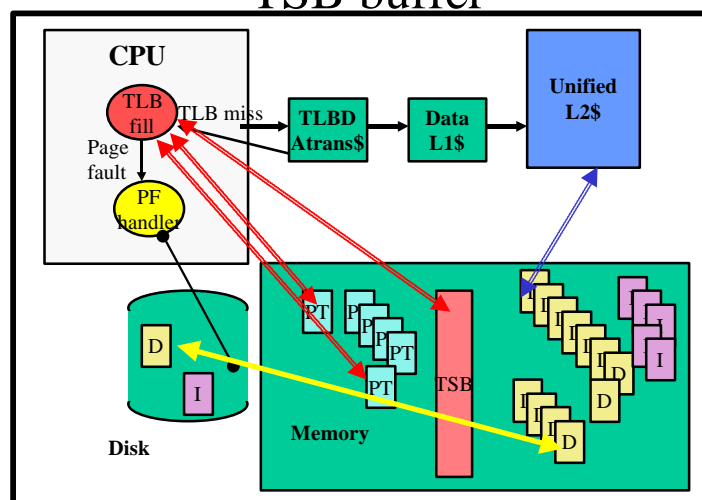
Typical page size = 4kB - 16kB

TLB-reach = 0.1MB - 8MB

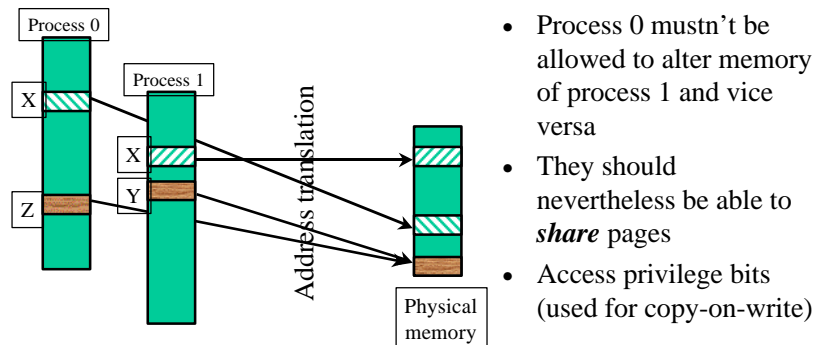
FIX:

- *Multiple page sizes, e.g., 8kB and 8 MB*
- *TSB -- A direct-mapped translation in memory as a "second-level TLB"*

TSB buffer



Protection



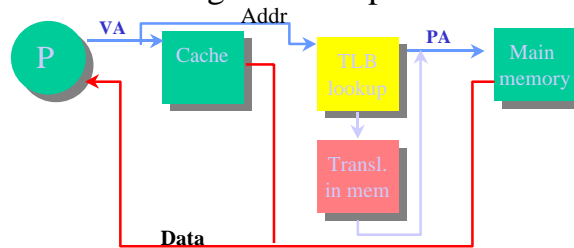
Protection mechanisms

The address translation mechanism can be used to provide memory protection:

- Use *protection attribute bits* for each page
- Stored in the page table entry (PTE) and TLB
- Each page gets its own protection
- If a process does not have permission to, e.g., write to a memory address, this is detected in the address translation and an exception is raised
- *Supervisor/user modes* necessary to prevent user processes from changing e.g. page tables

Do we need a fast TLB?

- We do a TLB lookup for every access to L1!
- Why not cache virtual addresses instead?
 - Move the TLB on the other side of the cache
 - It is only needed for finding stuff in Memory anyhow
 - The TLB can be made larger and slower -- great
- What is wrong with this picture?

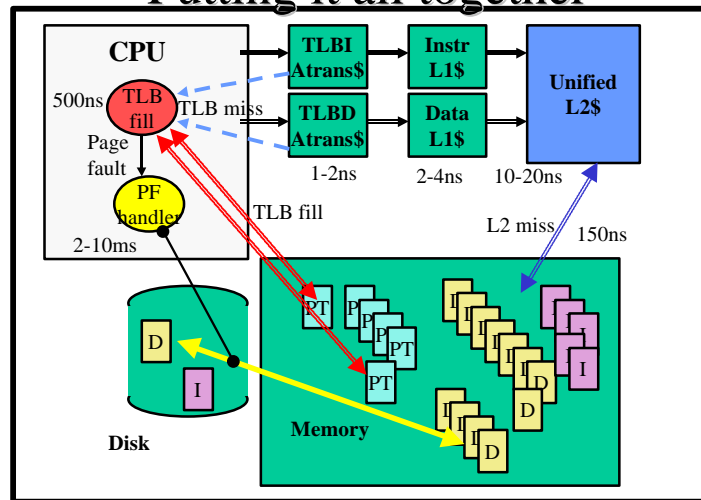


Aliasing Problem

The same physical page may be accessed by several processes using different virtual addresses

- Using virtual addresses in a cache will cause confusion: a write by one process may not be observed
- Translation to PA before the 1:st level cache makes it slow
- Flushing the cache on each process switch is slow
- =>VIPT (VirtuallyIndexedPhysicallyTagged) is the answer
 - Direct-mapped cache no larger than a page
 - No more sets than there are cache lines on a page + logic
 - Page coloring can be used to guarantee correspondence between more PA and VA bits (e.g., Sun Microsystems)

Putting it all together



Summary

Cache memories:

- HW-management
- Separate instruction and data caches permits simultaneous instruction fetch and data access
- Four questions:
 - Block placement
 - Block identification
 - Block replacement
 - Write strategy

Virtual memory:

- Software-management
- Very high miss penalty => miss rate must be very low
- Also supports:
 - memory protection
 - multiprogramming

Micro Benchmark Signature

```
for (times = 0; times < Max; times++) /* many times*/  
  
  for (i=0; i < ArraySize; i = i + Stride)  
    dummy = A[i]; /* touch an item in the array */
```

Stepping through the array

```
for (times = 0; times < Max; times++) /* many times*/  
  
  for (i=0; i < ArraySize; i = i + Stride)  
    dummy = A[i]; /* touch an item in the array */
```



0 Array Size = 16, Stride=4



0 Array Size = 16, Stride=8



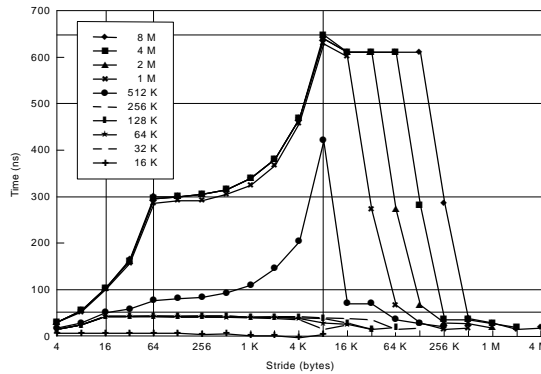
0 Array Size = 32, Stride=4



0 Array Size = 32, Stride=8

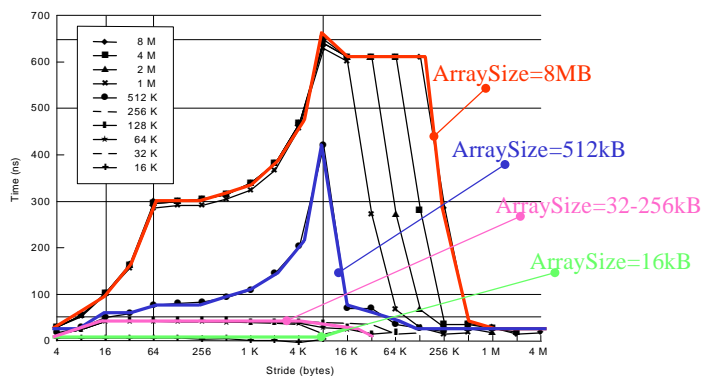
Micro Benchmark Signature

```
for (times = 0; times < Max; times++) /* many times*/  
  
for (i=0; i < ArraySize; i = i + Stride)  
    dummy = A[i]; /* touch an item in the array */
```



Micro Benchmark Signature

```
for (times = 0; times < Max; time++) /* many times*/  
  
for (i=0; i < ArraySize; i = i + Stride)  
    dummy = A[i]; /* touch an item in the array */
```



Micro Benchmark Signature

```
for (times = 0; times < Max; time++) /* many times*/
```

```
  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```

