### Paging/Virtual Memory Review

#### User A:
- Virtual Addresses
- Physical Memory
- TLB
- Stack
- Static
- Page Table

#### User B:
- Virtual Addresses
- TLB
- Virtual Addr.
- 64 MB
- Static
- Page Table

### Why virtual memory?

- **Protection**
  - regions of the address space can be read only, execute only...
- **Flexibility**
  - portions of a program can be placed anywhere in physical memory, without relocation
- **Expandability**
  - can leave room in virtual address space for objects to grow
- **Efficient use of fast storage**
  - retain only most important portions of the program in memory
- **Can run programs larger than size of physical memory**

### Three Advantages of Virtual Memory

1. **Translation**:
   - Program can be given consistent view of memory, even though physical memory is scrambled
   - Makes multiple processes reasonable
   - Only the most important part of program ("Working Set") must be in physical memory
   - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later
2. **Protection**
   - Different processes protected from each other
   - Different pages can be given special behavior
     - (Read Only, Invisible to user programs, etc.)
   - Kernel data protected from User programs
   - Very important for protection from malicious programs (viruses)
3. **Sharing**
   - Can map same physical page to multiple users ("Shared memory")

### TLB, Page Table

Memory lookup slow: TLB to reduce performance cost of VM

Need more compact representation to reduce memory size cost of simple 1-level page table, especially for 64-bit address

- 64 bit address space, 4K pages => 2^52 entries in the page table

**Solutions**:
- Multi-leveled page tables
- Inverted page tables

### Comparing the 2 levels of hierarchy

<table>
<thead>
<tr>
<th>Cache Version</th>
<th>Virtual Memory Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block or Line</td>
<td>Page</td>
</tr>
<tr>
<td>Miss</td>
<td>Page Fault</td>
</tr>
<tr>
<td>Block Size: 32-64B</td>
<td>Size: 4K-8KB</td>
</tr>
<tr>
<td>Placement: Direct Mapped, N-way Set Associative</td>
<td>Fully Associative</td>
</tr>
<tr>
<td>Replacement: LRU or Random</td>
<td>Least Recently Used (LRU)</td>
</tr>
<tr>
<td>Write Thru or Back</td>
<td>Write Back</td>
</tr>
</tbody>
</table>

### Picking Page Size

- **Minimize wasted storage**
  - small page minimizes internal fragmentation
  - small page increases size of page table, TLB usage
- **Minimize transfer time**
  - large pages (multiple disk sectors) amortize disk access cost
  - sometimes transfers unnecessary info
  - sometimes prefetches useful data
- **General trend toward larger pages because**
  - big cheap RAM
  - increasing memory - disk performance gap
  - larger processor address spaces