Locality of Reference and Virtual Memory
Keeping Data Close

In working with memory in any computer system, we want to access data as quickly as possible.
Keeping Data Close

CPU Registers: 1 word /register
___ general purpose registers
Keeping Data Close

CPU Cache: Stores 4 KB “pages” of memory

Intel i9-10900KF: 256 KB /CPU (L2) + 20 MB (L3)
Keeping Data Close

RAM: 128 GB in our “Class Computer”
Key Idea: Locality of Reference
System Memory (RAM)

All computers have a fixed amount of RAM:

[1]:
System Memory (RAM)

[2]:

System Memory (RAM)

[3]:
System Memory (RAM)

To help to organize RAM, we will break RAM up into chunks called:

$ getconf PAGESIZE$
Virtual Memory

An abstraction between ___________ and
__________________________________.
Virtual Memory

• A _________________ translates:

• Page tables are **NOT** shared:
<table>
<thead>
<tr>
<th>P1 Page Table</th>
<th>RAM</th>
<th>P2 Page Table</th>
<th>P3 Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0x0000</td>
<td>[0]</td>
<td>[0]</td>
</tr>
<tr>
<td>[1]</td>
<td>0x1000</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>[3]</td>
<td>0x3000</td>
<td>[3]</td>
<td>[3]</td>
</tr>
<tr>
<td>[8]</td>
<td>0x8000</td>
<td>[8]</td>
<td>[8]</td>
</tr>
<tr>
<td>[9]</td>
<td>0x9000</td>
<td>[9]</td>
<td>[9]</td>
</tr>
<tr>
<td>[10]</td>
<td>0xA000</td>
<td>[10]</td>
<td>[10]</td>
</tr>
<tr>
<td>[12]</td>
<td>0xC000</td>
<td>[12]</td>
<td>[12]</td>
</tr>
<tr>
<td>[13]</td>
<td>0xD000</td>
<td>[13]</td>
<td>[13]</td>
</tr>
<tr>
<td>[14]</td>
<td>0xE000</td>
<td>[14]</td>
<td>[14]</td>
</tr>
<tr>
<td>[15]</td>
<td>0xF000</td>
<td>[15]</td>
<td>[15]</td>
</tr>
</tbody>
</table>
Can we meet all allocation requests?
Are we limited to just RAM?
Advantages of a Virtual Memory System?
Virtual Memory

- [External Storage]:
unsigned int *array = malloc(SIZE * SIZE *
    sizeof(unsigned int));

// Add data to each element of the array:
printf(" Start of `array`: %p\n", array);
printf(" End of `array`: %p\n",
    &array[(SIZE*SIZE)-1]);
1: Load Program

2: Run `./programCode` (Page #1)
   - `malloc(4000)`

3. Run `./programCode` (Page #2)
   - `malloc(10000)`
   - Open hiddenImage.png
   - Read all of image

4: Run `./programCode` (Page #3)
   - Access OG 4 KB
   - Finish program
Q1: What is the range of possible file sizes for hiddenImage.png?
Q2: What is the range of possible file sizes for ./programCode?
Q3: What is the size of the heap immediately before the program finishes?