CSE 451: Operating Systems
Winter 2022

Module 14
Windows MM

Gary Kimura

Memory Management (continued)

- Windows Paging
- Windows Kernel Heap
- Wickedly Fun Exam Question

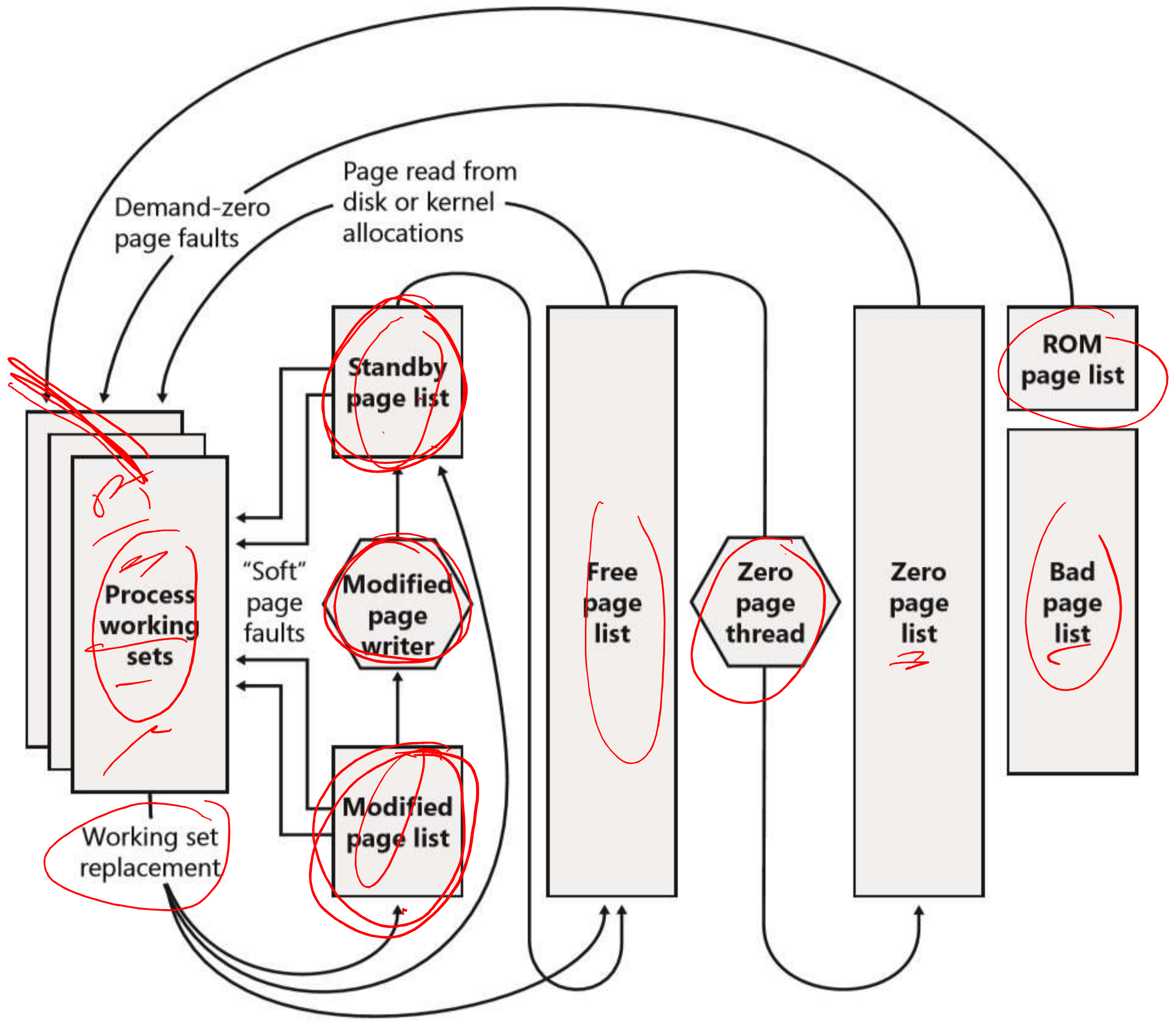


Figure 7-35 State diagram for page frames

Page States

- Active (also called Valid)
- Transition
- Standby
- Modified
- Modified no-write
- Free
- Zeroed
- Rom
- Bad

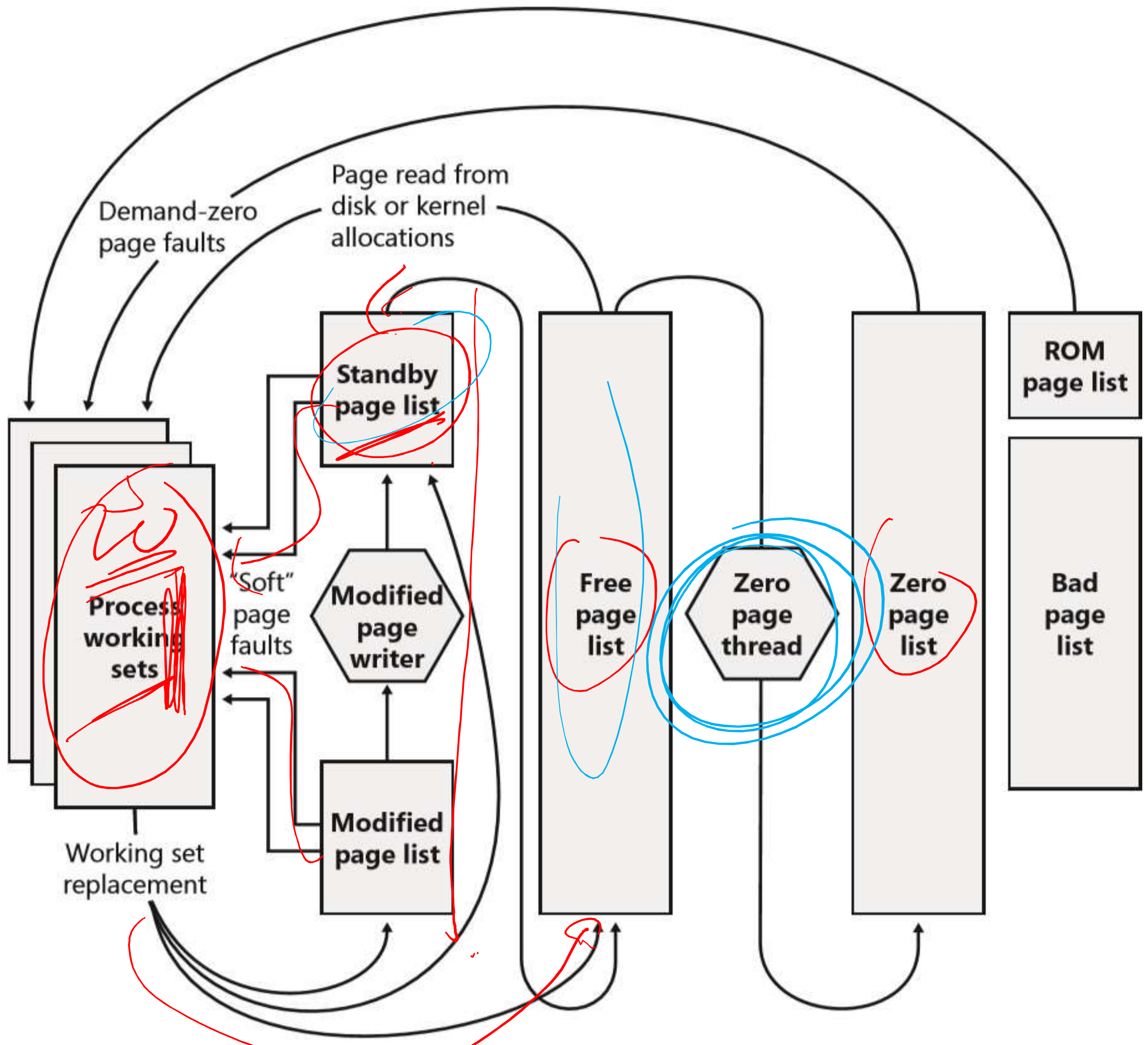


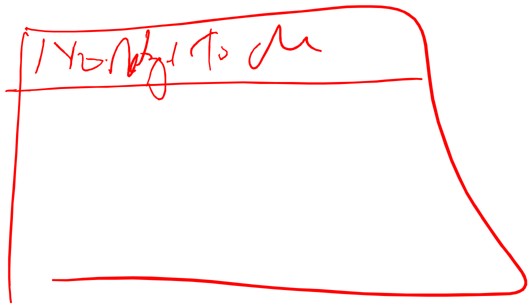
Figure 7-35 State diagram for page frames

Paging Features

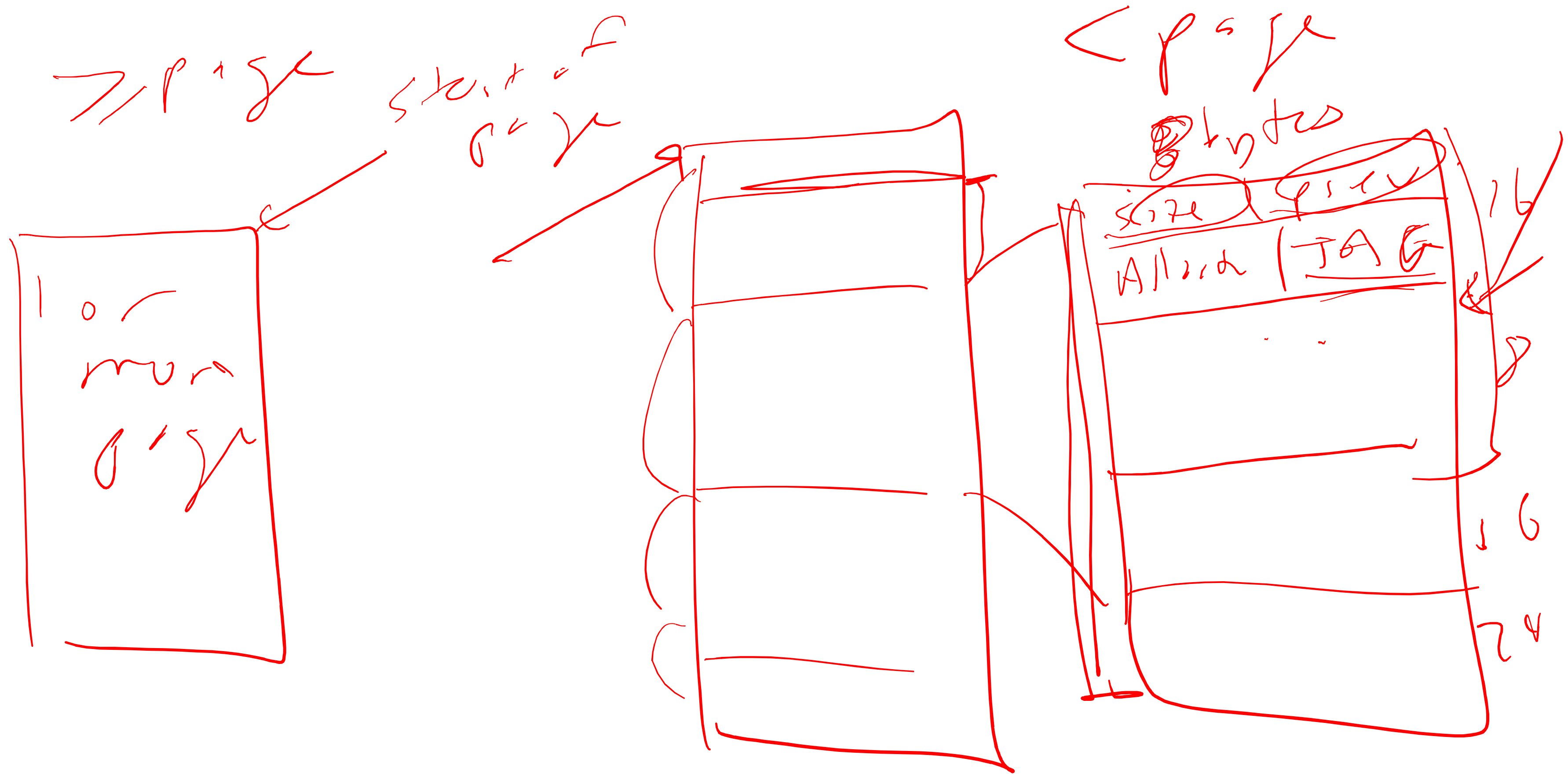
- Local and Global page replacement
- LRU on top of FIFO
- Hard and Soft page faults

Windows Kernel Pool (aka Heap)

- Boundary tagged
- Paged and Nonpaged / NonPaged Must Succeed
- Lookaside lists
- Node type codes to help quickly identify objects in the pool



Overall Pool Layout

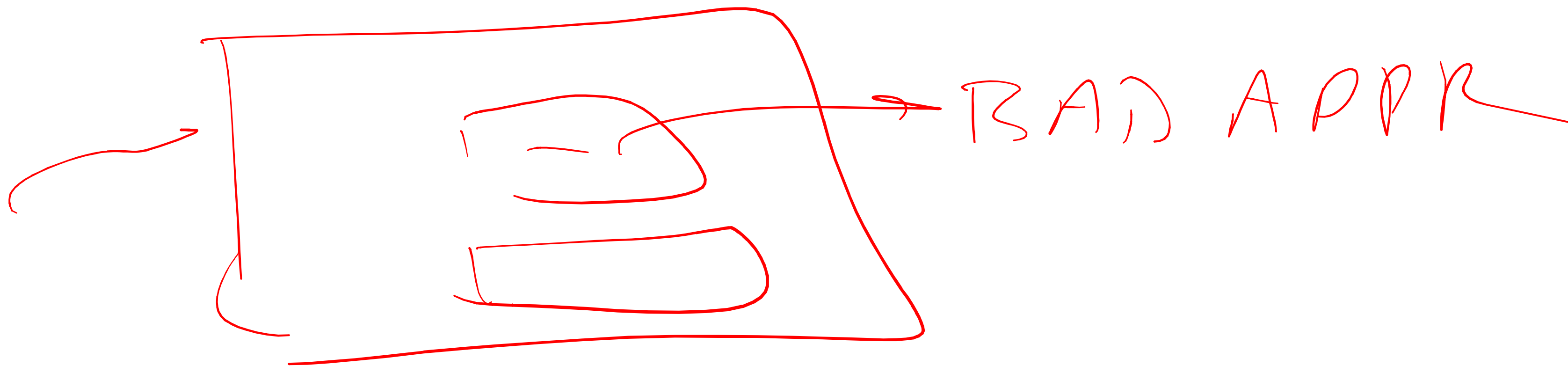


Debugging pool corruption

- Checked build versus Free build
- Debugging pool corruption bugs, often stale pointers or allocation overruns
- 0xDEADBEEF and 0xBAADF00D
- Extra code to check for pool corruption
- A pointer hack I used to catch a bad actor (data alignment fault)

| pool |

Pool Corruption



A Fun Exam Question from 2013

- Examine how long it takes a user mode program writing to an array of integers.
- Assumptions
  - The entire array will fit into physical memory ~~(no paging)~~
  - The system is pretty much idle except for this program
- First malloc() the array, and then
- Time how long it takes to write to every element of the array using various access patterns.

The Actual Exam Question

Consider the following program that allocates a multi-megabyte sized array of unsigned longs, and then times how long it takes to write to every array location. The program varies the pattern it uses to write to each array location based on a stride that changes between each pass through the array.

For example a stride of 1 makes one pass through the array accessing locations 0,1,2,… until the end of the array is reached. A stride of 2 makes two passes through the array, first accessing locations 0,2,4,… and then accessing locations 1,3,5,… until the end of the array is reached. The program starts with a stride value of 1 and then increases it, based on user input, until the stride is equal to half the size of the array. The program times how long it takes, in seconds, for each new stride through the array.

The program takes three parameters, first is the number of megabytes to allocate to the array, the second and third parameters are the multiplication and additive factors used to compute the stride. For example, the parameters "2 1 1" allocate a 2MB array testing stride values of 1,2,3,4,…, 131072. Note that 131072 is the halfway point in a 2MB integer array. The parameters "2 2 0" allocate a 2MB array testing stride values of 1,2,4,8,16,…,131072. In other words the stride value doubles each time.

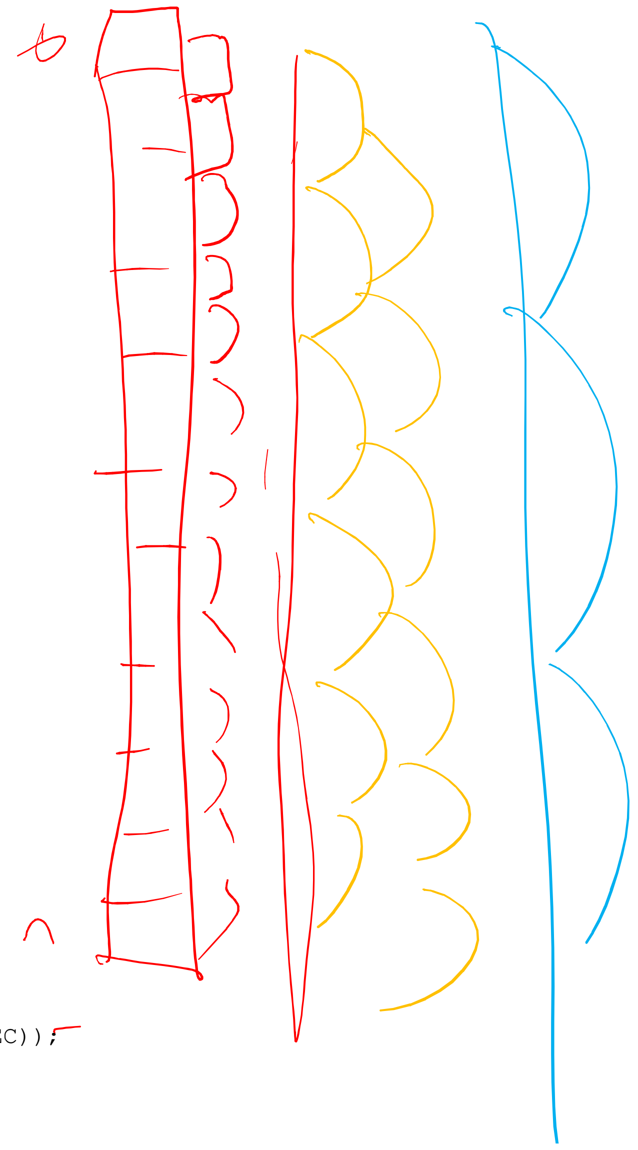
```
void main (int argc, char *argv[])
{
    clock_t StartTime, EndTime;
    unsigned long *Array, Size, StrideTimes, StridePlus, i,j,k;

    sscanf(argv[1], "%lu", &Size);
    sscanf(argv[2], "%lu", &StrideTimes);
    sscanf(argv[4], "%lu", &StridePlus);

    printf("Size = %luMB\n", Size);

    //  Allocate a test array
    Size = 1024*1024*Size;
    if ((Array = malloc(Size)) == NULL) {
        printf("malloc failed\n");
        return;
    }
    Size /= 4;

    //  Now test it for strides from 1 to size/2
    printf("  Stride   Seconds\n");
    for (i = 1; i < Size/2; i = (i*StrideTimes)+StridePlus) {
        printf("%8lu", i);
        StartTime = clock();
        for (j = 0; j < i; j++) {
            for (k = j; k < Size; k += i) {
                Array[k] = k;
            }
        }
        EndTime = clock();
        printf(", %8.3f\n", ((double)(EndTime - StartTime)/CLOCKS_PER_SEC));
    }
}
```



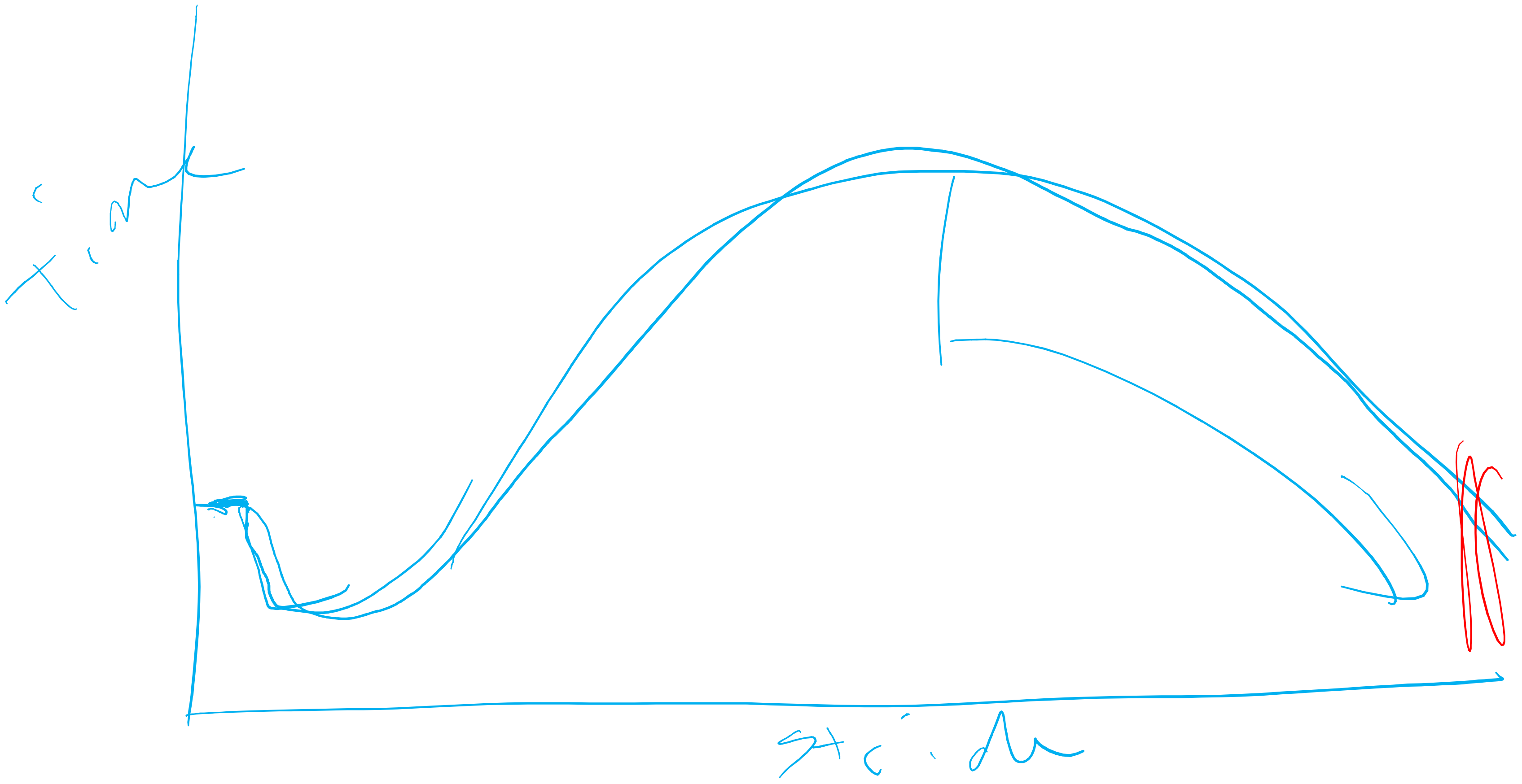
This program was run on both Windows and Linux systems with 4GB of RAM. Here is the data for a run of “1024 2 0” on a Linux system.

Size = 1024 MB

Stride	Seconds	Stride	Seconds	Stride	Seconds
1,	1.920	512,	11.890	262144,	18.250
2,	1.560	1024,	12.190	524288,	14.710
4,	2.780	2048,	12.960	1048576,	9.810
8,	5.530	4096,	14.280	2097152,	5.310
16,	11.450	8192,	16.510	4194304,	4.100
32,	16.470	16384,	22.070	8388608,	3.820
64,	15.000	32768,	22.390	16777216,	3.760
128,	13.390	65536,	21.510	33554432,	2.170
256,	12.310	131072,	20.270	67108864,	1.170

20 - 12

Illustration of graph



Two questions to answer

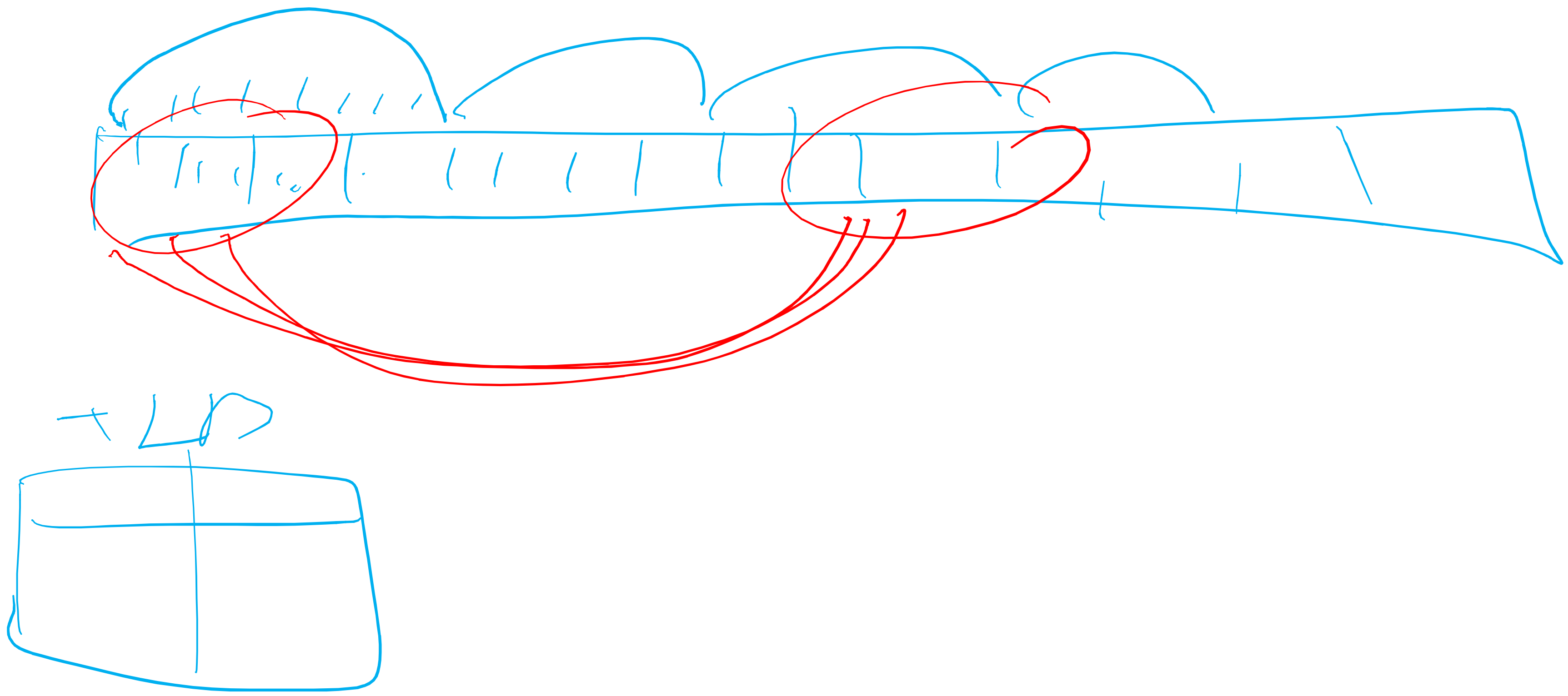
[20 points] Notice how the first pass with a stride of 1 takes longer then the second pass with a stride of 2. This behavior showed up consistently on Linux but not Windows. Please give a plausible explanation for what causes this phenomenon (it might be a mix of both hardware and software), and what the operating system can do to prevent it. You will need to justify your answer.

[20 points] Also notice how the time for each pass increases and then decreases as the stride values grow from 1 to 67108864. Both Windows and Linux exhibited this behavior. Please give a plausible explanation for this phenomenon (it might be a mix of both hardware and software), and what the operating system can do to prevent it. You will need to justify your answer.

Things to consider

- Zero pages
- TLB behavior
- Various cache levels
- Cache line sizes

Possible explanations



Now consider the same program run with "64 1 1". The entire output is too long to include here, but the following is a small section of the output that shows another anomaly that occurs on both Windows and Linux.

```
                                          Size = 64MB
Stride       Seconds                  Stride     Seconds                    Stride       Seconds
     700,        0.109                     726,     0.109                        752,        0.124
     701,        0.140                     727,     0.109                        753,        0.109
     702,        0.093                     728,     0.109                        754,        0.109
     703,        0.109                     729,     0.109                        755,        0.093
   704,         0.530                      730,     0.109                        756,        0.109
     705,        0.109                     731,     0.093                        757,        0.109
     706,        0.093                     732,     0.109                        758,        0.109
     707,        0.109                     733,     0.109                        759,        0.109
     708,        0.109                     734,     0.109                        760,        0.109
     709,        0.109                     735,     0.109                        761,        0.109
     710,        0.109                     736,     0.140                        762,        0.109
     711,        0.109                     737,     0.093                        763,        0.109
     712,        0.109                     738,     0.124                        764,        0.093
     713,        0.109                     739,     0.093                      765,        0.499
   714,         0.202                      740,     0.109                        766,        0.109
     715,        0.109                   741,       0.249                      767,        0.109
     716,        0.093                   742,       0.109                      768,        0.748
     717,        0.109                     743,     0.109                        769,        0.109
     718,        0.109                     744,     0.109                        770,        0.093
     719,        0.093                     745,     0.171                        771,        0.124
     720,        0.171                     746,     0.109                        772,        0.093
     721,        0.093                     747,     0.109                        773,        0.109
     722,        0.109                     748,     0.109                        774,        0.109
     723,        0.109                   749,       0.405                        775,        0.109
     724,        0.093                     750,     0.109                        776,        0.124
     725,        0.109                     751,     0.093                        777,        0.093
```

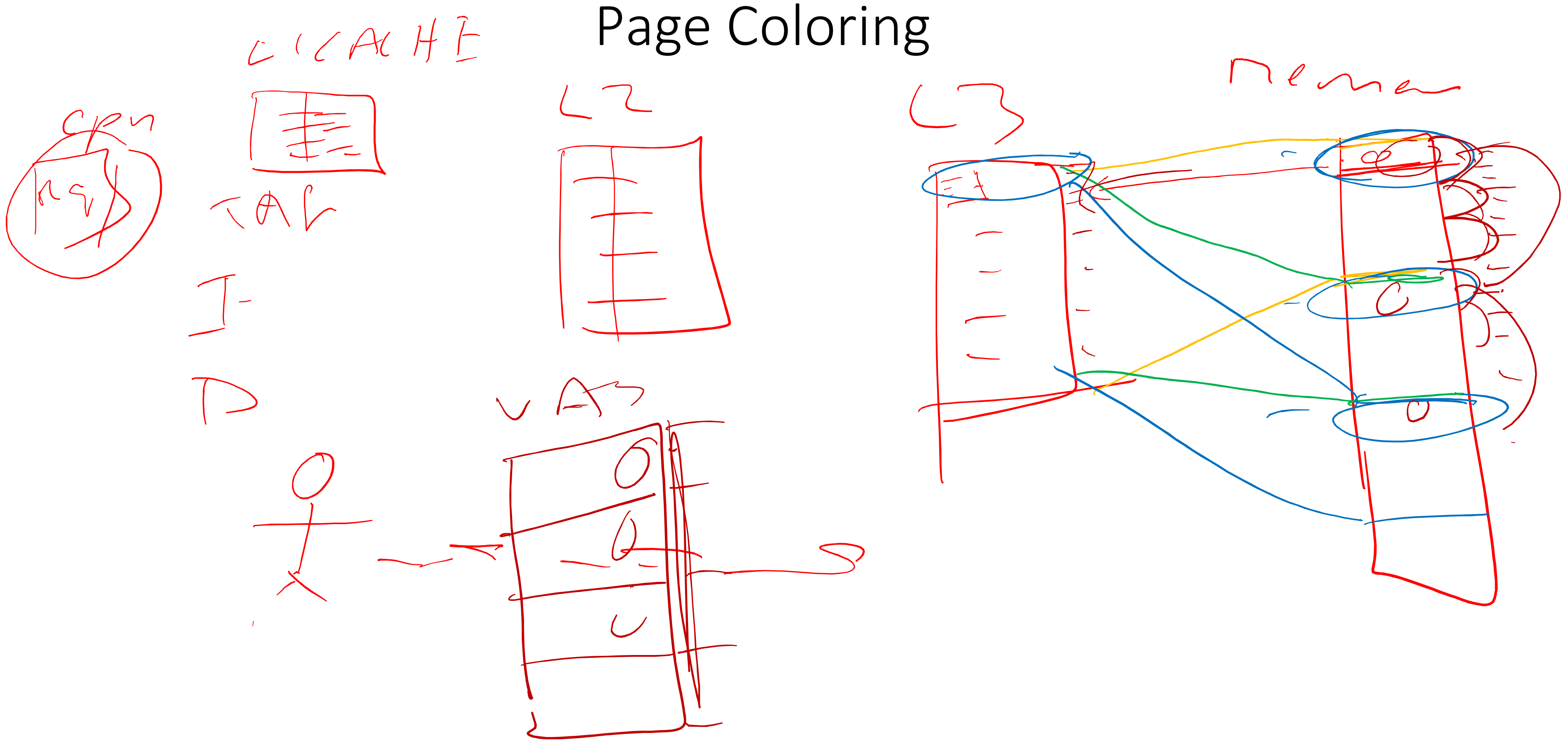
Question to ponder

[20 points] Notice how the times are consistently in the low 100ms range except for an occasional blip in the 200ms to 700ms range. These blips have been underlined. Please offer an explanation for these blips. Your answer needs to offer a plausible explanation of what is causing this anomaly (it might be a mix of both hardware and software), and what the operating system can do to prevent it, if anything. You will need to justify your answer. If you do cannot offer an educated guess on what causes this phenomenon explain what you could do to determine its cause.

Things to consider

- Virtual and physical caches
- Page coloring
- Side channel attacks

Page Coloring



Possible explanations

An observation

“Caches work great, except when they don’t”