Balancing Code Order and Loop Structure in Binary Code Analysis

Shadmaan Hye[1] Matthew LeGendre[2] Katherine E. Isaacs[1]

[1]University of Utah, [2]Lawrence Livermore National Laboratory

Problem Statement

In program analysis, it is often helpful to consider the code in both its line-of-code order and its loop order.

Existing layouts prioritize one at the cost of the other. Our new layout balances both of these concerns

Solution: We introduce a novel layout that balances instruction order with loop structures, simplifying navigation.

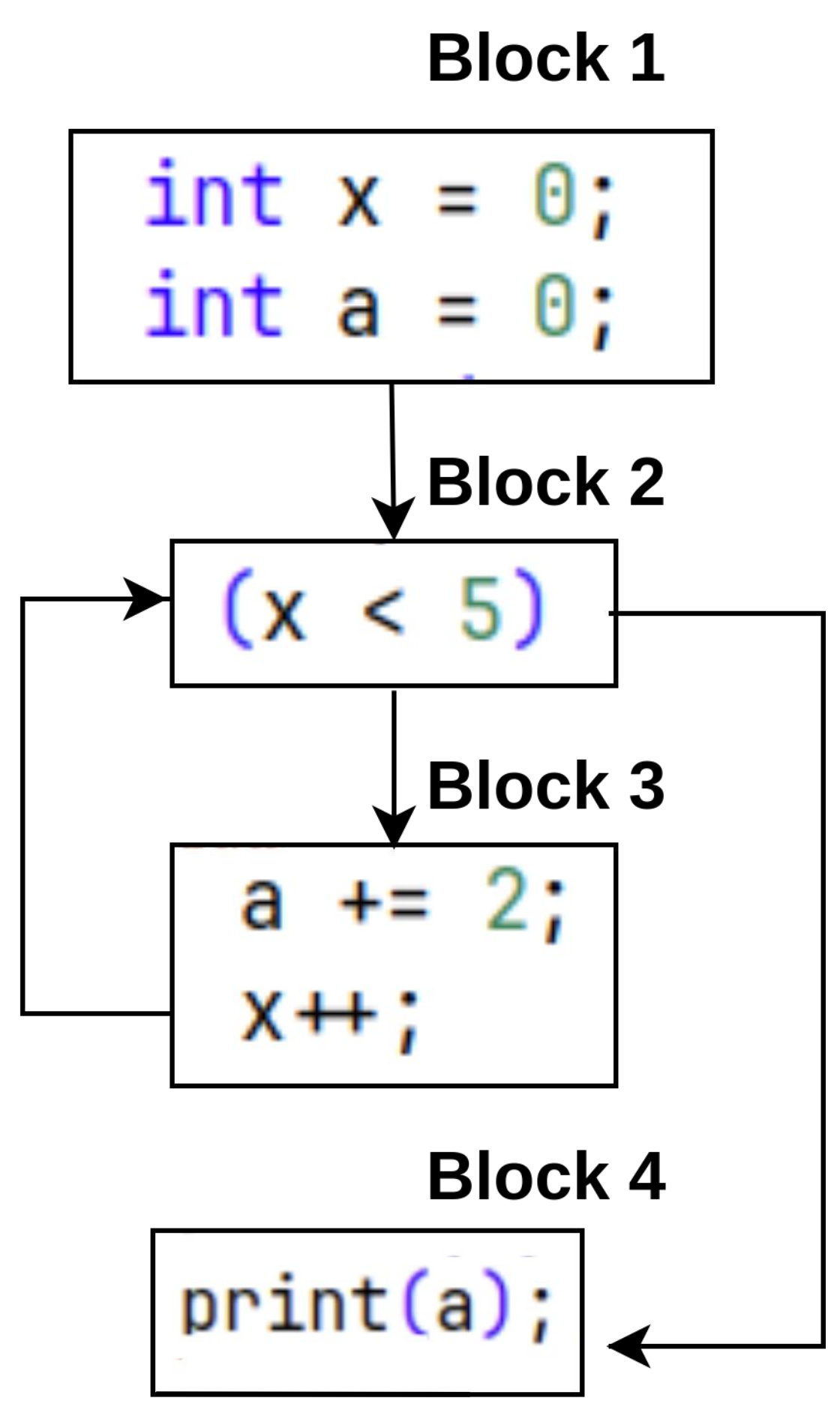
Code Order vs. Loop Structure

Source Code Example

Code Order

```
int x = 0;
int a = 0;
while (x < 5) {
    a += 2;
    x++;
}
print(a);
```

Loop Structure

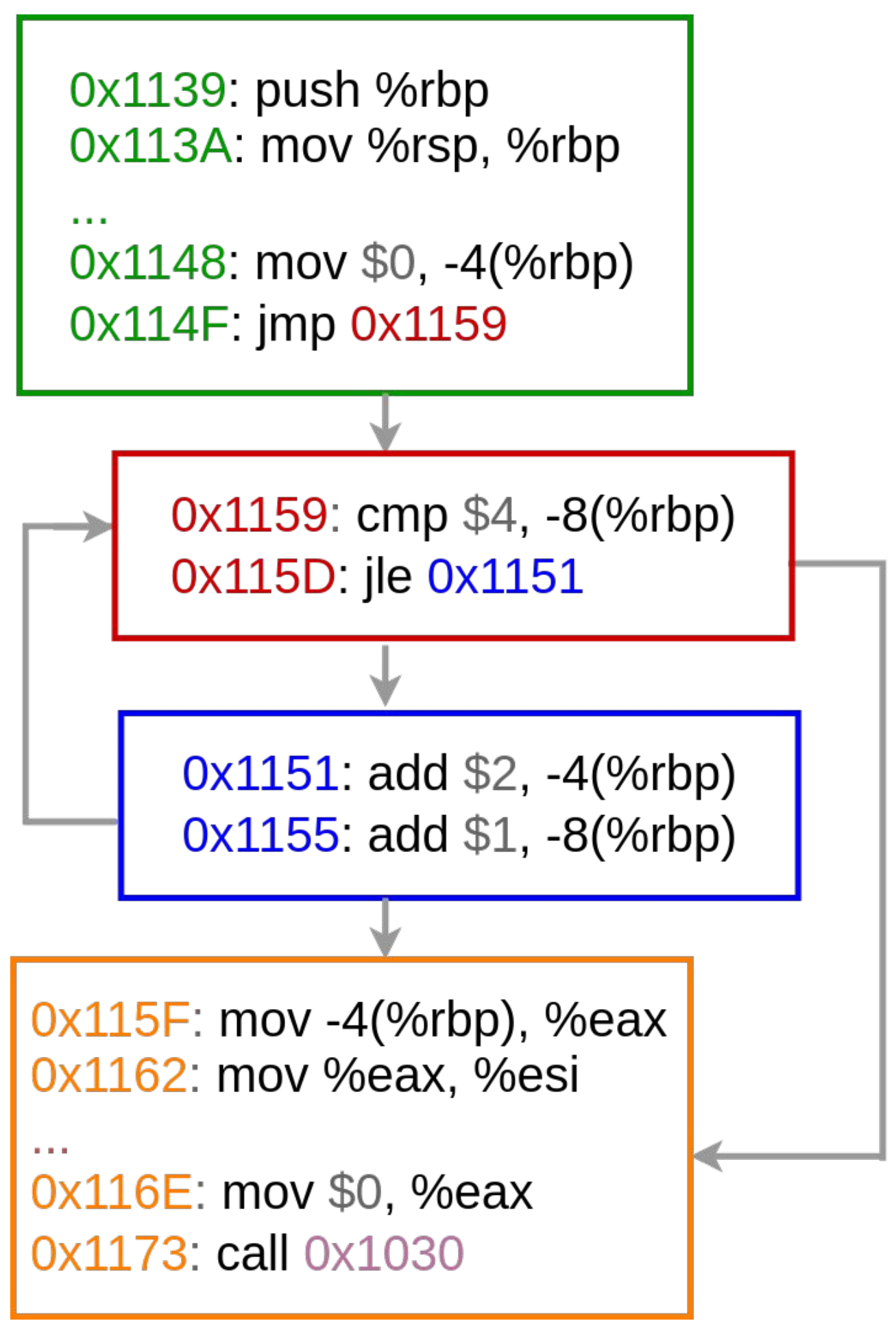


Binary Code Example

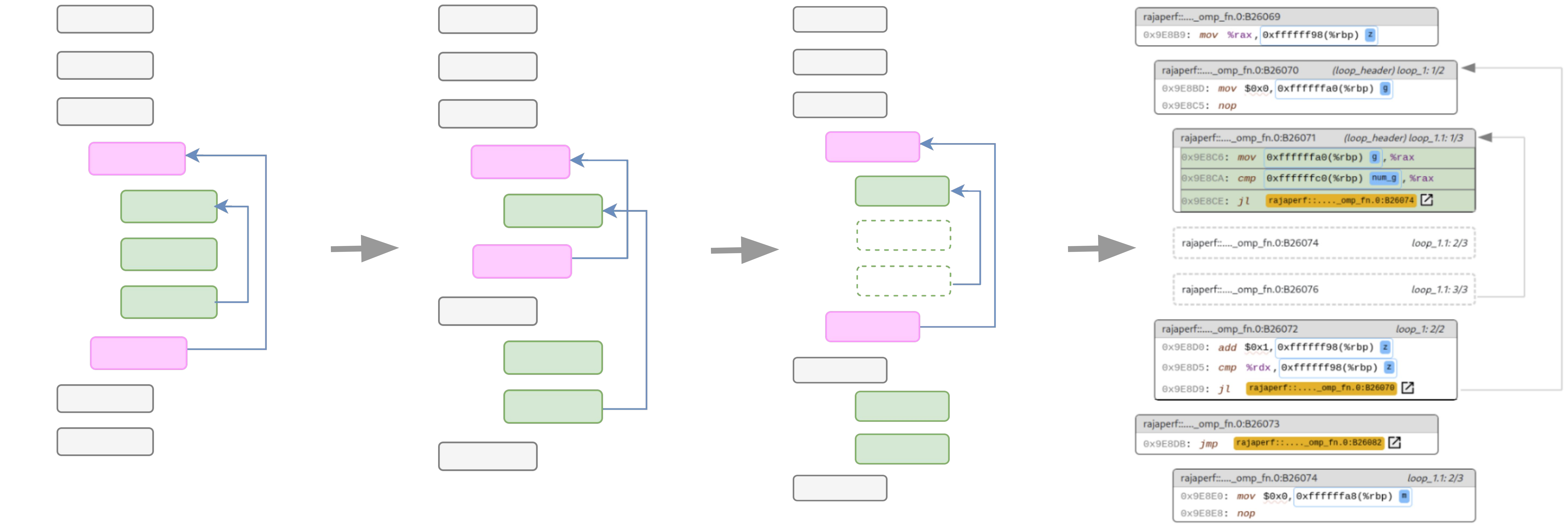
Code Order

```
0x1139: push %rbp
0x113A: mov %rsp, %rbp
...
0x1148: mov $0, -4(%rbp)
0x114F: jmp 0x1159

0x1151: add $2, -4(%rbp)
0x1155: add $1, -8(%rbp)

0x1159: cmp $4, -8(%rbp)
0x115D: jle 0x1151

0x115F: mov -4(%rbp), %eax
0x1162: mov %eax, %esi
...
0x116E: mov $0, %eax
0x1173: call 0x1030
```

Loop Structure



Our Novel Layout Approach



Ideal case: Indention matches loop nesting

Real case: Loops not contiguous in binary code

Dotted Pseudo blocks suggest ideal case while preserving real order

Novel Layout of Loop Structure using Pseudo or dotted blocks

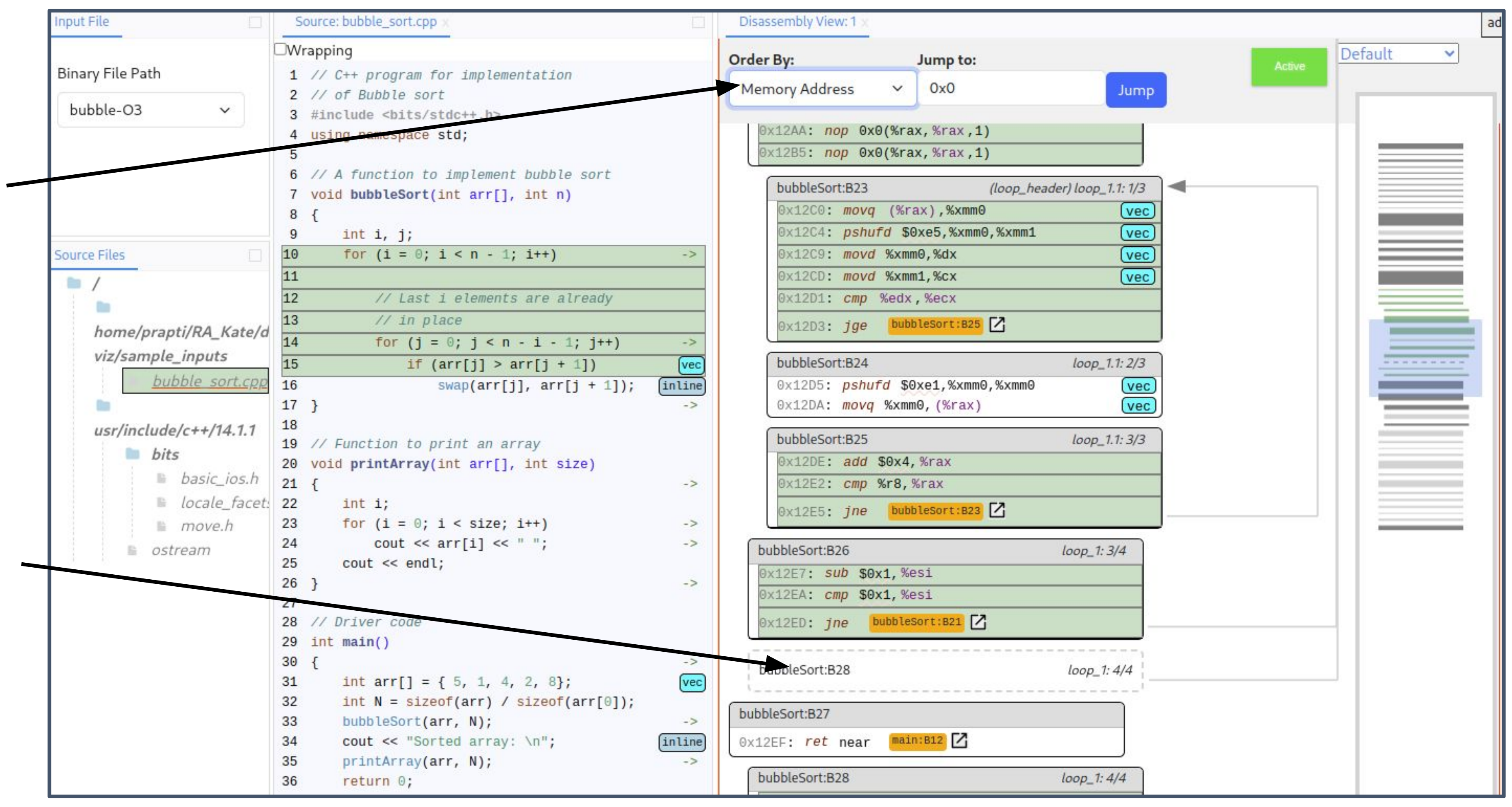
```
rajaperf::...._omp_fn.0:B26069
0x9E8B9: mov %rax, 0xffffff98(%rbp) z

    rajaperf::...._omp_fn.0:B26070            (loop_header) loop_1: 1/2
    0x9E8BD: mov $0x0, 0xffffffa0(%rbp) g
    0x9E8C5: nop

        rajaperf::...._omp_fn.0:B26071        (loop_header) loop_1.1: 1/3
        0x9E8C6: mov 0xffffffa0(%rbp) g , %rax
        0x9E8CA: cmp 0xffffffc0(%rbp) num_g , %rax
        0x9E8CE: jl rajaperf::...._omp_fn.0:B26074

        rajaperf::...._omp_fn.0:B26074        loop_1.1: 2/3

        rajaperf::...._omp_fn.0:B26076        loop_1.1: 3/3

    rajaperf::...._omp_fn.0:B26072            loop_1: 2/2
    0x9E8D0: add $0x1, 0xffffff98(%rbp) z
    0x9E8D5: cmp %rdx, 0xffffff98(%rbp) z
    0x9E8D9: jl rajaperf::...._omp_fn.0:B26070

rajaperf::...._omp_fn.0:B26073
0x9E8DB: jmp rajaperf::...._omp_fn.0:B26082

        rajaperf::...._omp_fn.0:B26074        loop_1.1: 2/3
        0x9E8E0: mov $0x0, 0xffffffa8(%rbp) m
        0x9E8E8: nop
```

Incorporating our Layout into a Program Analysis Interface

We incorporate our new layout in a visualization interface for examining source and binary code.

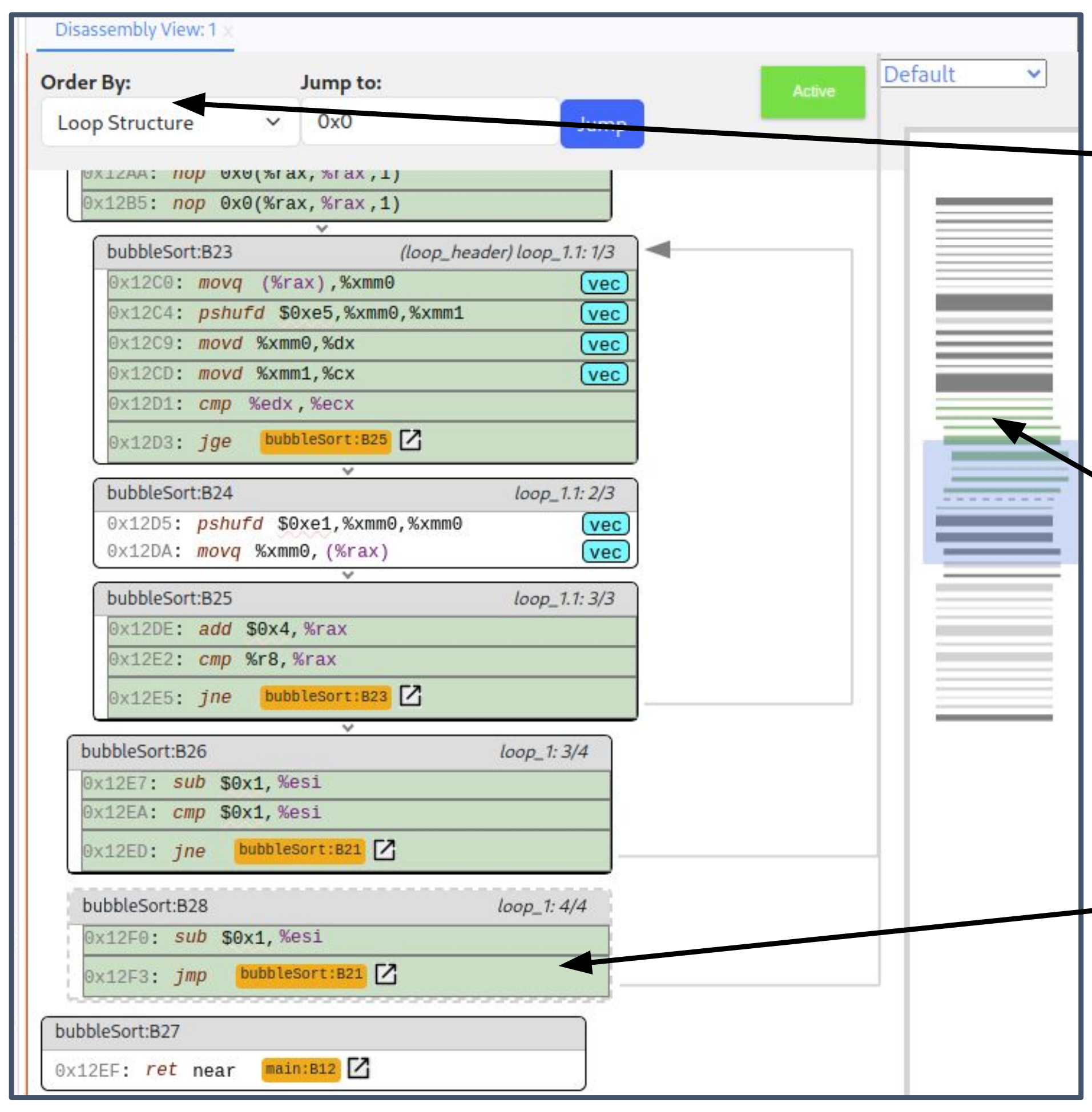
Memory Address order where the blocks are placed according to the address they appear in memory

Pseudo blocks (dotted blocks) here visualize ideal case like structure while preserving the memory address order



```
// C++ program for implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8};
    int N = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);
    return 0;
```

```
0x12AA: nop 0x0(%rax,%rax,1)
0x12B5: nop 0x0(%rax,%rax,1)
  bubbleSort:B23                    (loop_header) loop_1.1: 1/3
  0x12C0: movq (%rax),%xmm0                              vec
  0x12C4: pshufd $0xe5,%xmm0,%xmm1                       vec
  0x12C9: movd %xmm0,%dx                                 vec
  0x12CD: movd %xmm1,%cx                                 vec
  0x12D1: cmp %edx,%ecx
  0x12D3: jge bubbleSort:B25
  bubbleSort:B24                    loop_1.1: 2/3
  0x12D5: pshufd $0xe1,%xmm0,%xmm0                       vec
  0x12DA: movq %xmm0,(%rax)                              vec
  bubbleSort:B25                    loop_1.1: 3/3
  0x12DE: add $0x4,%rax
  0x12E2: cmp %r8,%rax
  0x12E5: jne bubbleSort:B23
bubbleSort:B26                      loop_1: 3/4
0x12E7: sub $0x1,%esi
0x12EA: cmp $0x1,%esi
0x12ED: jne bubbleSort:B21
bubbleSort:B28                      loop_1: 4/4
bubbleSort:B27
0x12EF: ret near main:B12
bubbleSort:B28                      loop_1: 4/4
```



```
0x12AA: nop 0x0(%rax,%rax,1)
0x12B5: nop 0x0(%rax,%rax,1)
  bubbleSort:B23                    (loop_header) loop_1.1: 1/3
  0x12C0: movq (%rax),%xmm0                              vec
  0x12C4: pshufd $0xe5,%xmm0,%xmm1                       vec
  0x12C9: movd %xmm0,%dx                                 vec
  0x12CD: movd %xmm1,%cx                                 vec
  0x12D1: cmp %edx,%ecx
  0x12D3: jge bubbleSort:B25
  bubbleSort:B24                    loop_1.1: 2/3
  0x12D5: pshufd $0xe1,%xmm0,%xmm0                       vec
  0x12DA: movq %xmm0,(%rax)                              vec
  bubbleSort:B25                    loop_1.1: 3/3
  0x12DE: add $0x4,%rax
  0x12E2: cmp %r8,%rax
  0x12E5: jne bubbleSort:B23
bubbleSort:B26                      loop_1: 3/4
0x12E7: sub $0x1,%esi
0x12EA: cmp $0x1,%esi
0x12ED: jne bubbleSort:B21
bubbleSort:B28                      loop_1: 4/4
0x12F0: sub $0x1,%esi
0x12F3: jmp bubbleSort:B21
bubbleSort:B27
0x12EF: ret near main:B12
```

Loop Structure Order where all the blocks in a particular loop are placed consecutively

Binary code overview where the indentations visualize a loop structure

Pseudo blocks are replaced with the actual blocks in this order

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. LLNL-POST-868718

www.sci.utah.edu

THE UNIVERSITY OF UTAH