Introduction

- Having only the target file of an application, we want to reduce its running time by optimizing and parallelizing its binary code at runtime.
- We are interested in enough running processes, since we trade off the overhead of optimization against the amount of time gained when executing the optimized code.

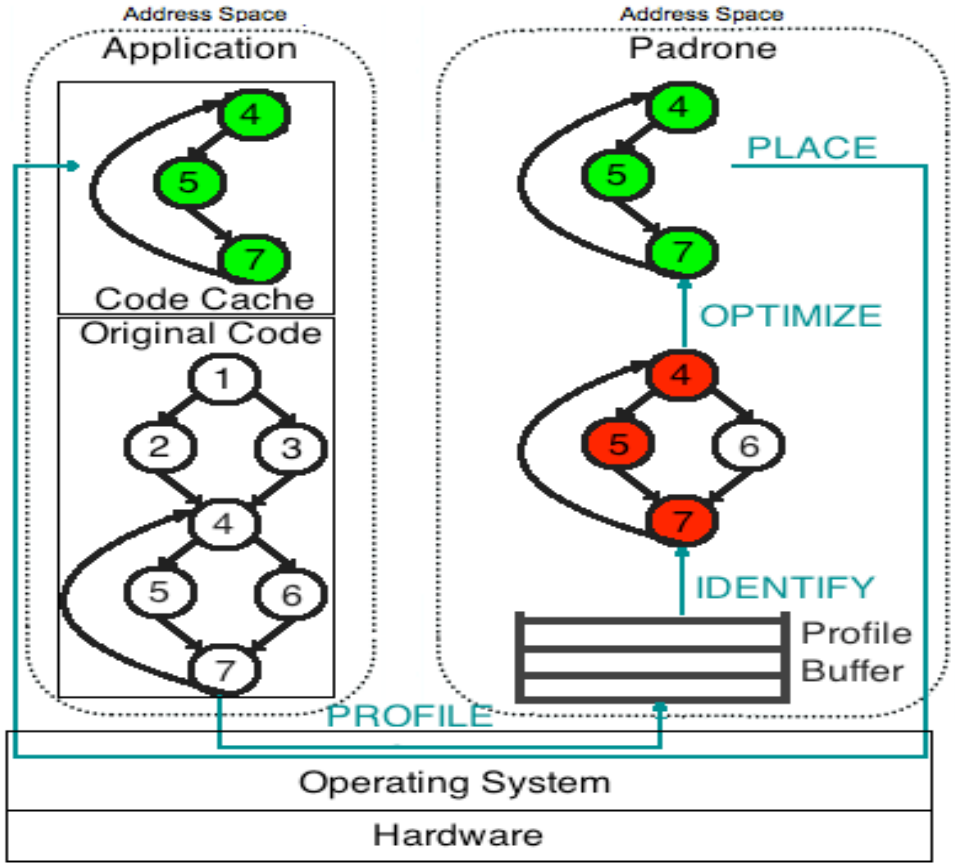
Motivations

- A target code compatible with a family of processors can take advantage from the features of new processors (backward compatibility).
- Complement the optimizations that the compiler cannot perform by taking advantage of runtime information.
- Profiling information?

Padrone as an infrastructure

- Padrone is a platform for dynamic binary analysis and optimization.
- Its main services:
  - Providing the count of certain types of hardware events:
    Instructions executed, cache misses, and branch mispredicted.
  - Providing a high level information from a binary code:
    Decoding a stream of bytes as x86 instructions.
    Finding out the function to which specific address belongs.
    Constructing the CFG of a function.
  - Injecting binary stream of bytes to the address space of another running process.

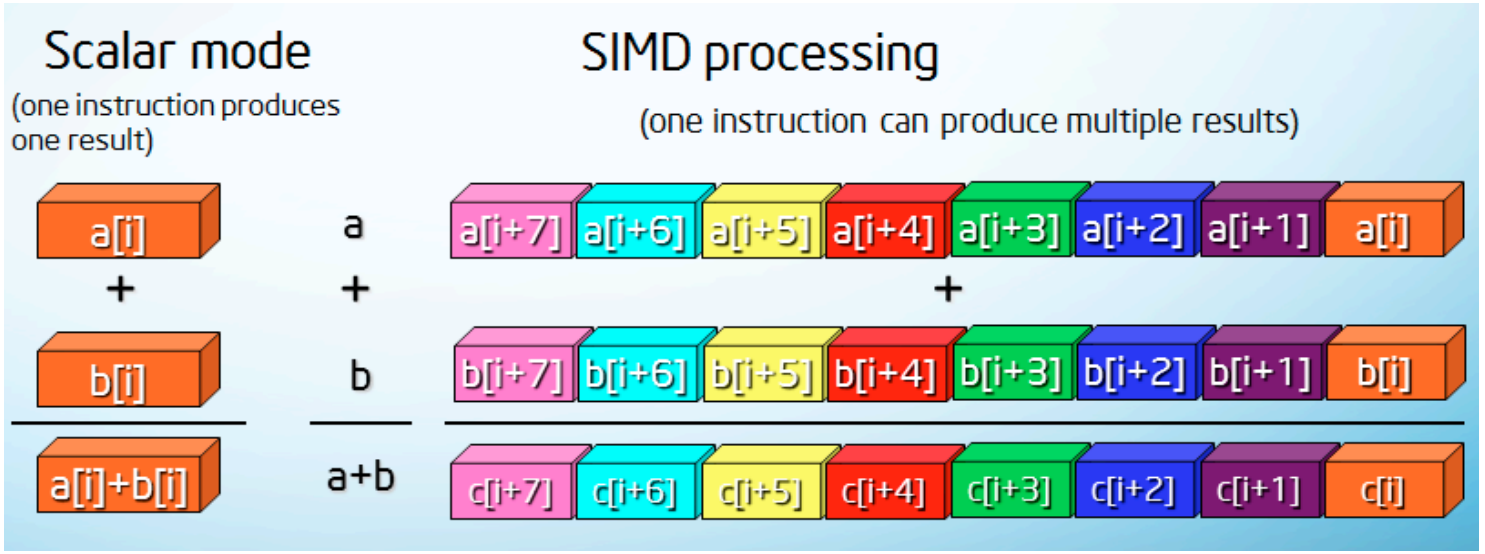
Different interactions during the optimization



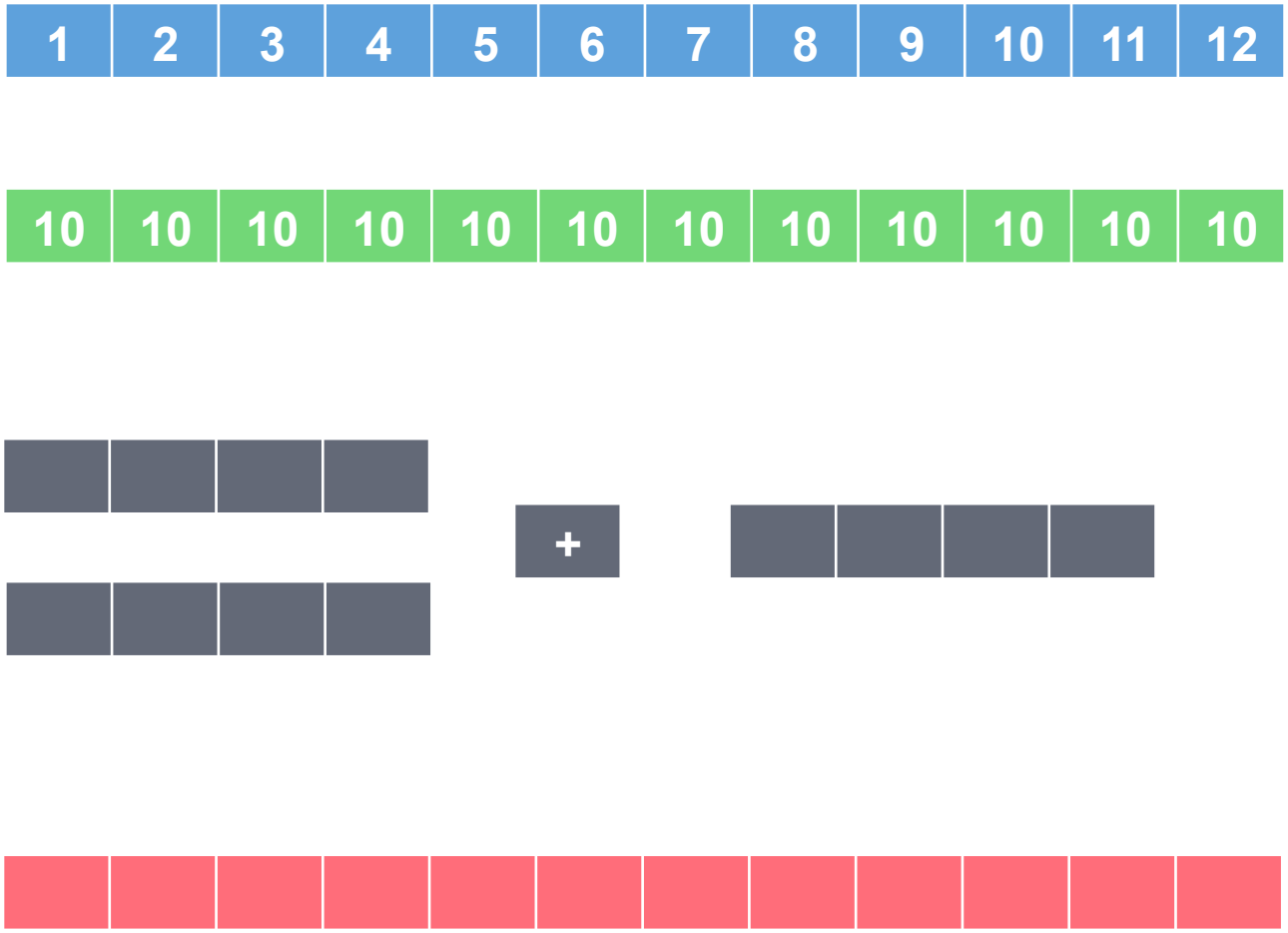
What is vectorization?

```
For ( i = 0; i < NUM_ELTS; i + 1){
        c[i] = a[i] + b[i];
}
```

```
For ( i = 0; i < NUM_ELTS; i + 8){ // one operation for each eight data elts
        c[i … i+7] = a[i ... i+7] + b[i ... i+7];
}
```

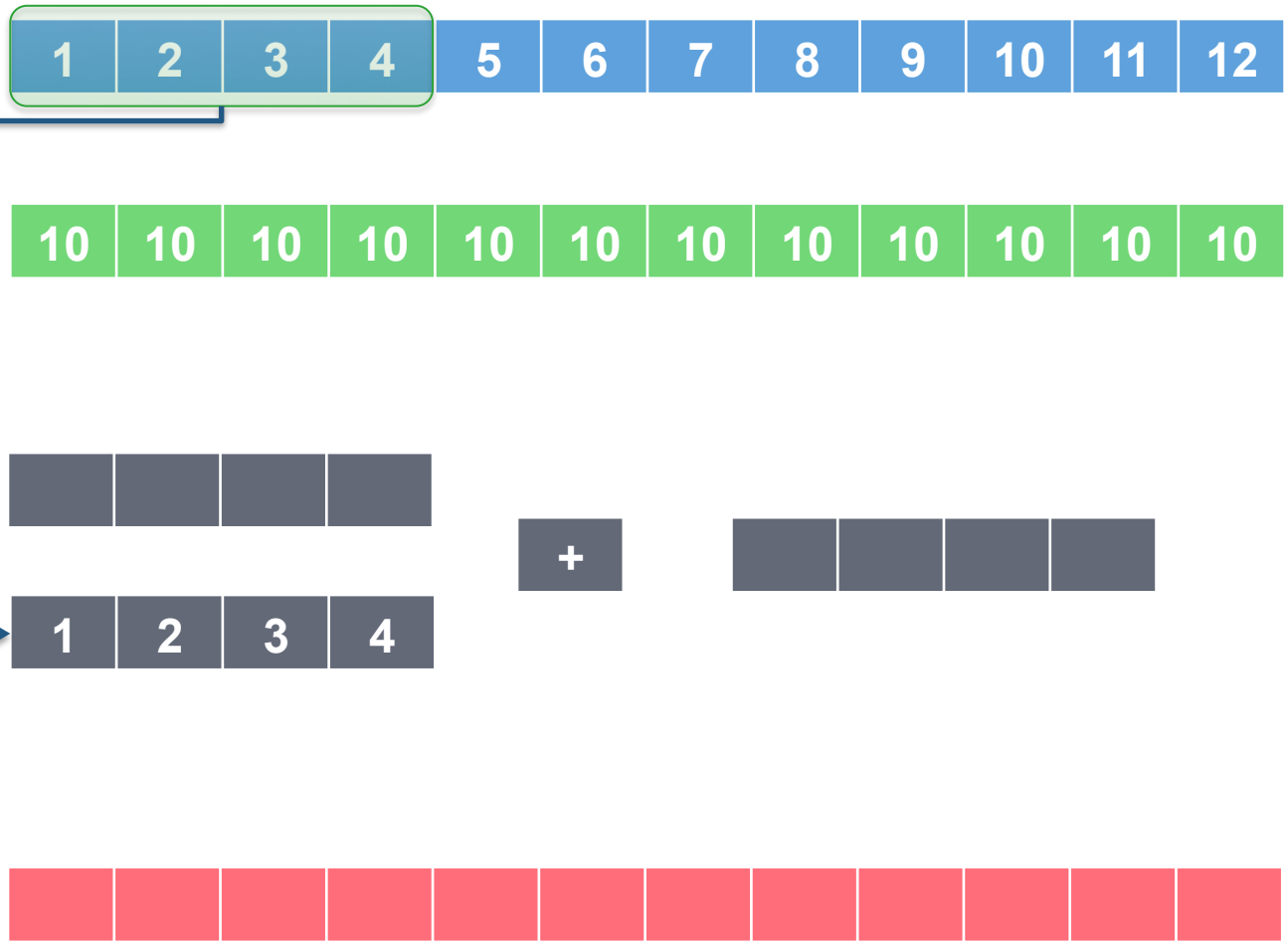


Vectorization example



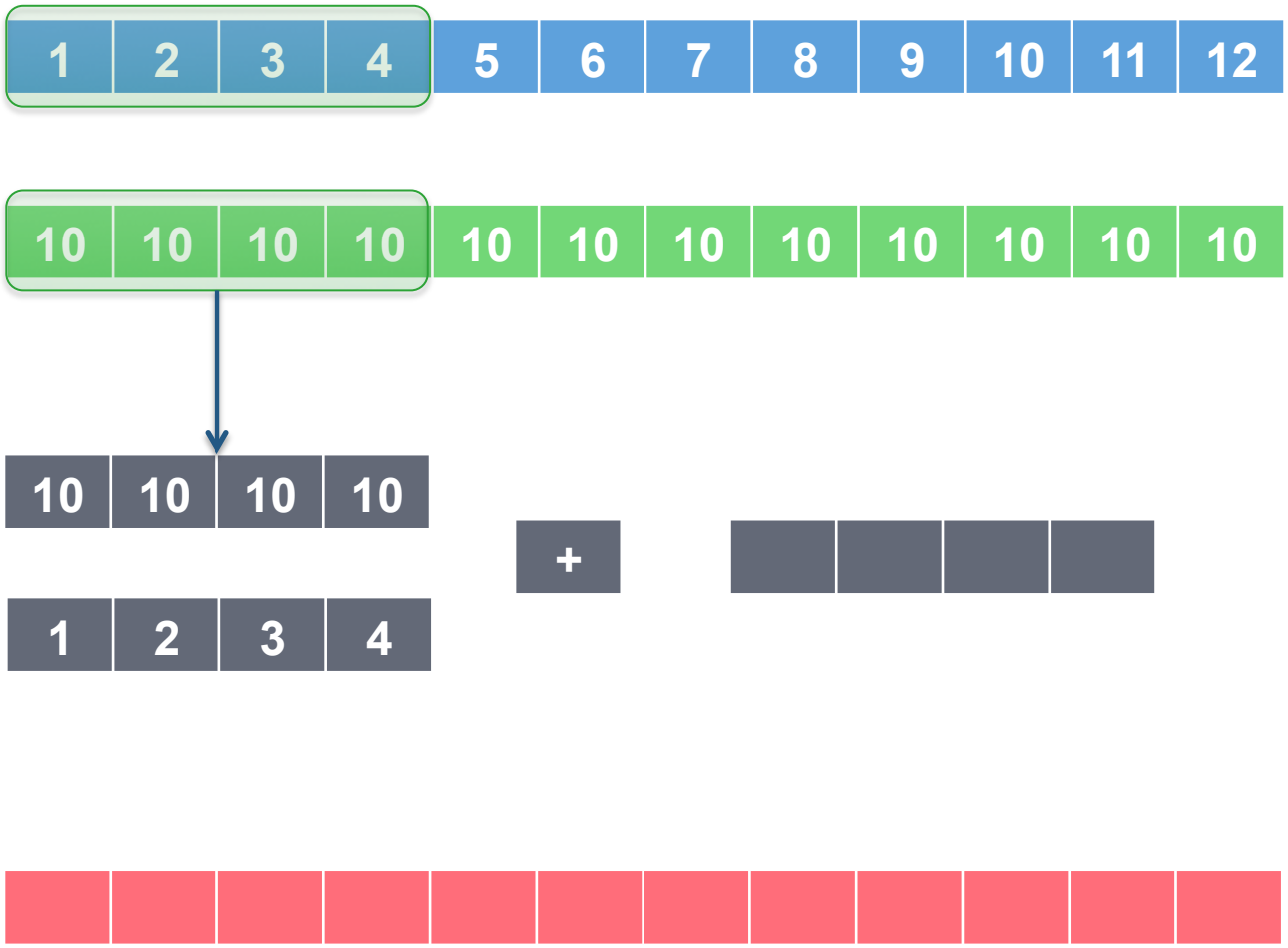

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



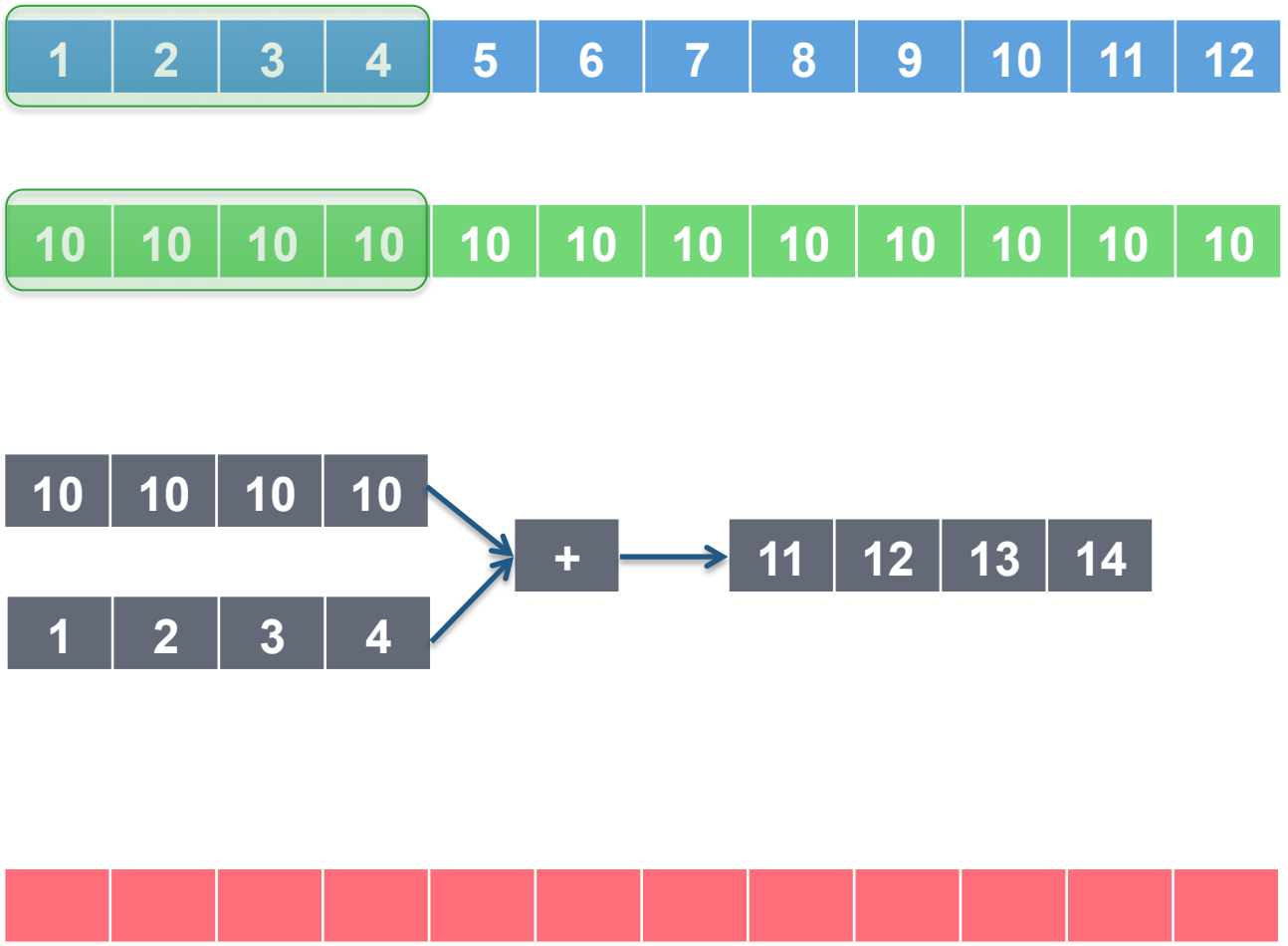

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



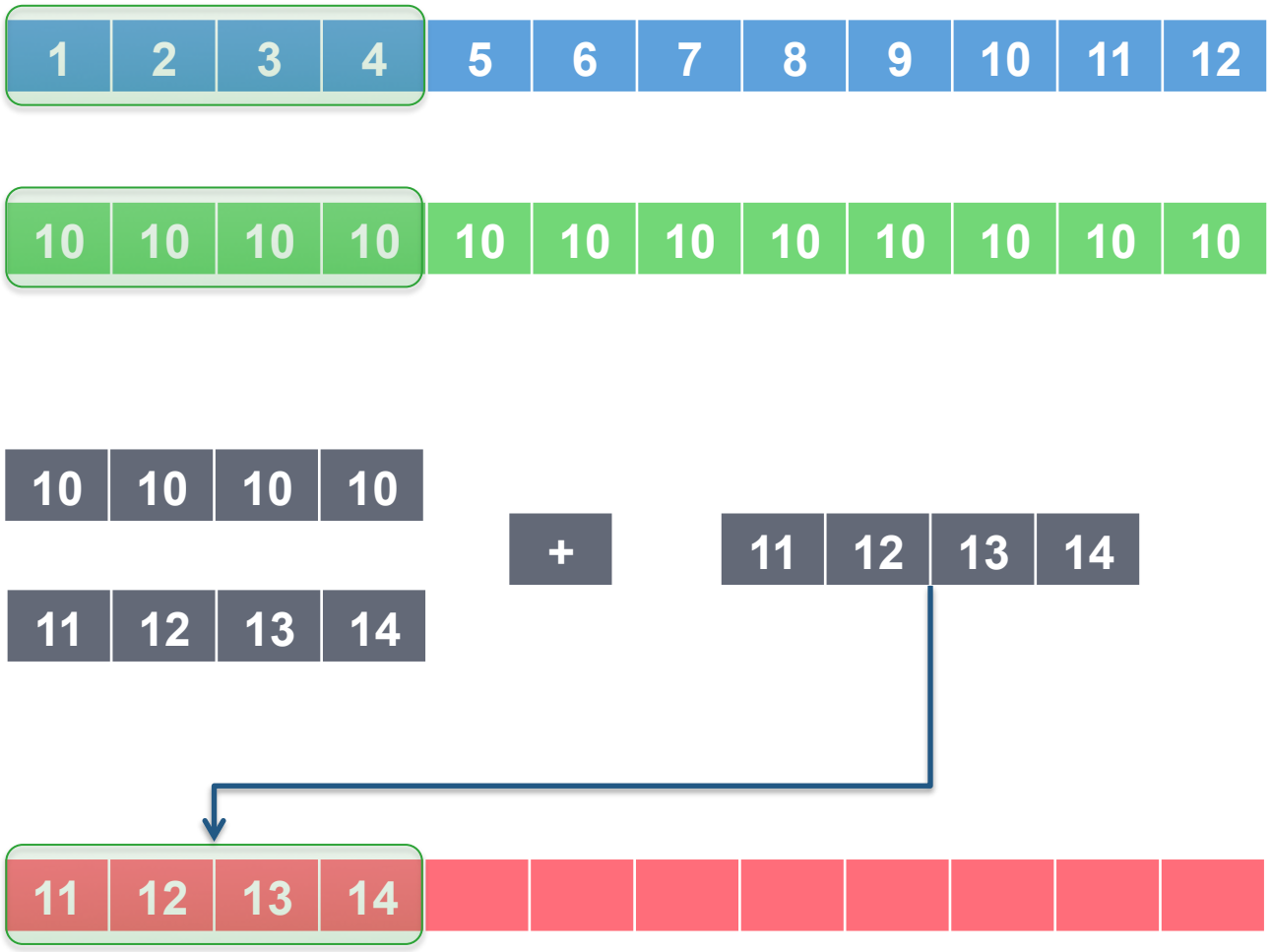

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



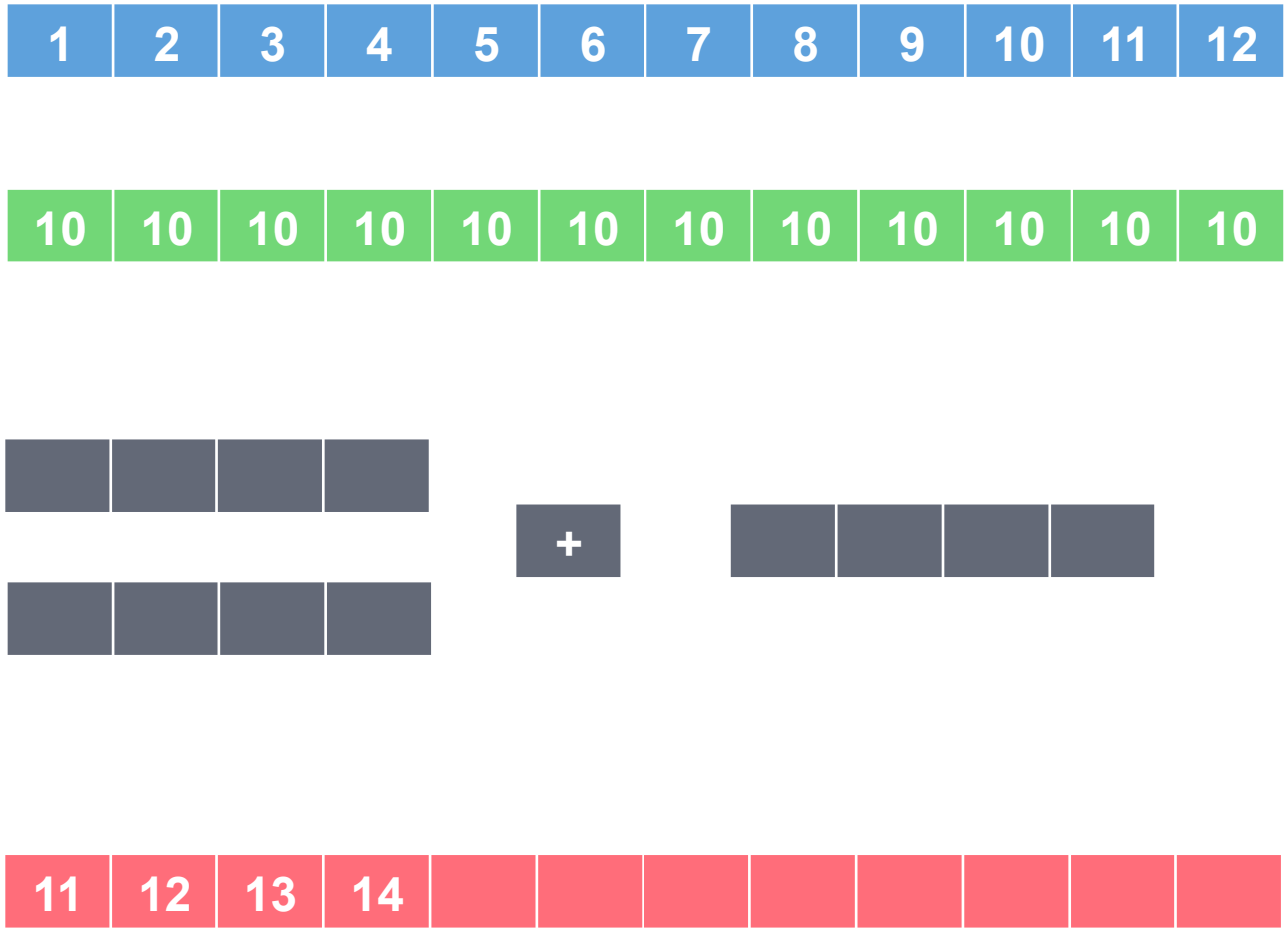

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



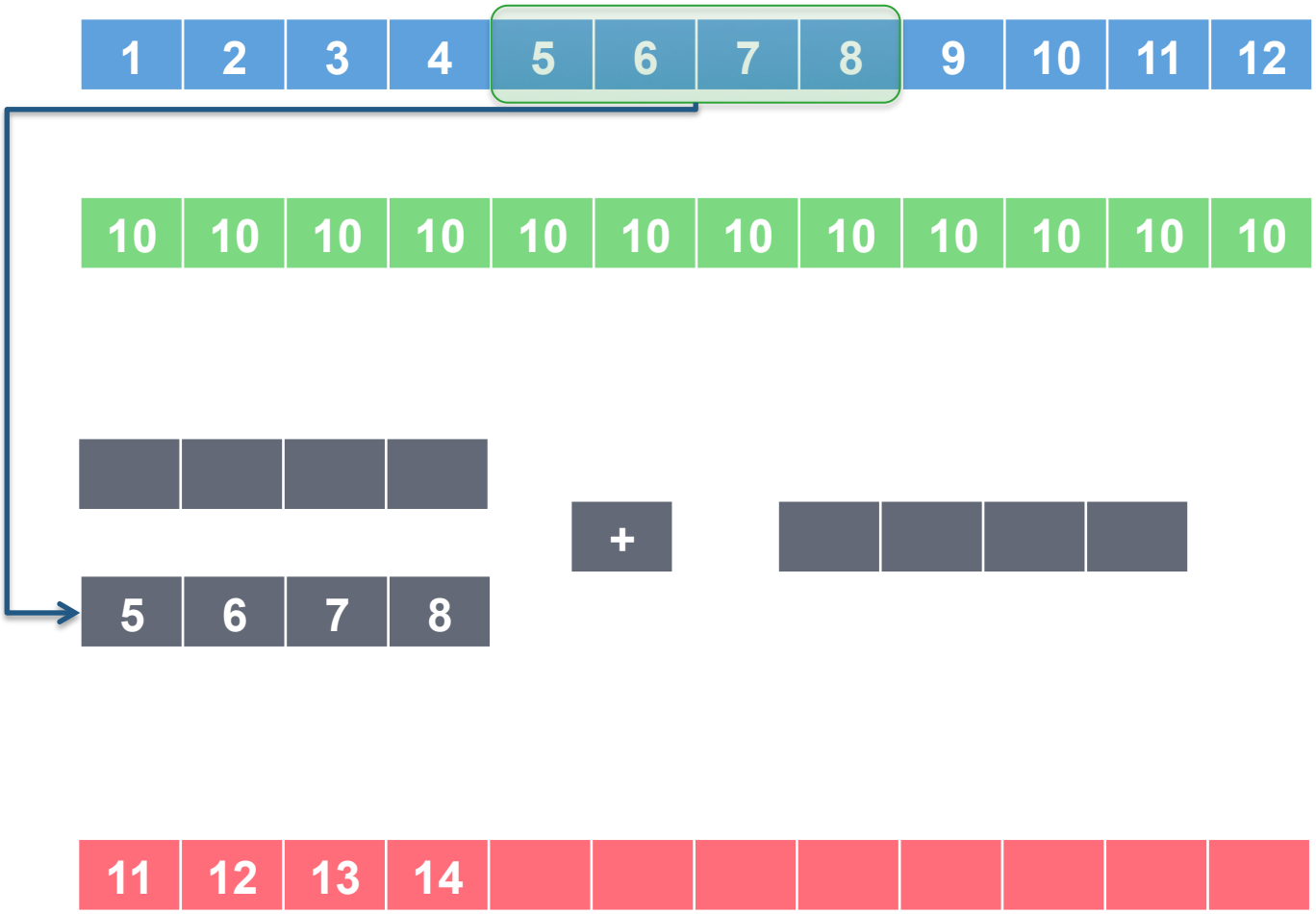

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



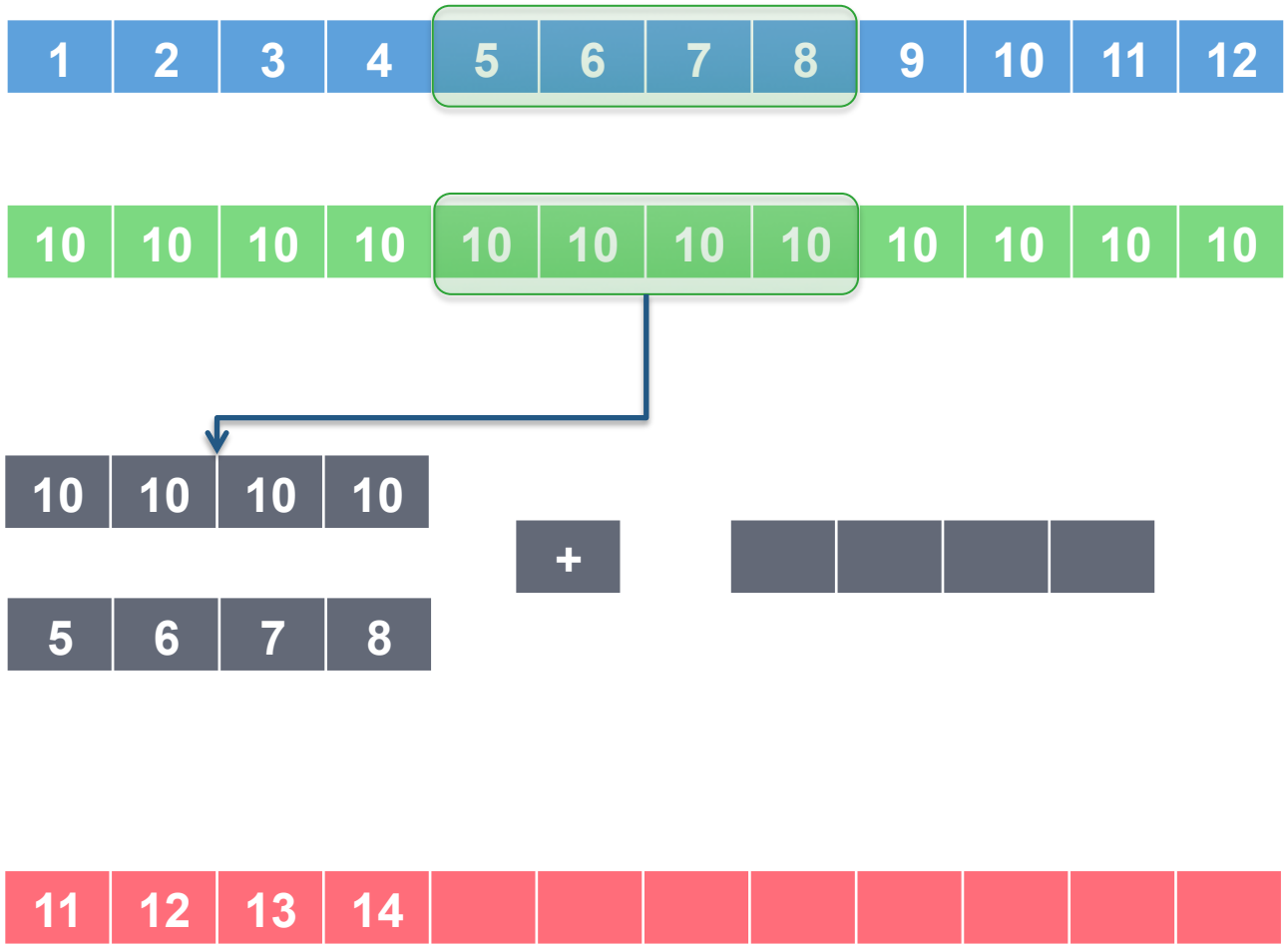

```
For( i  = 4; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



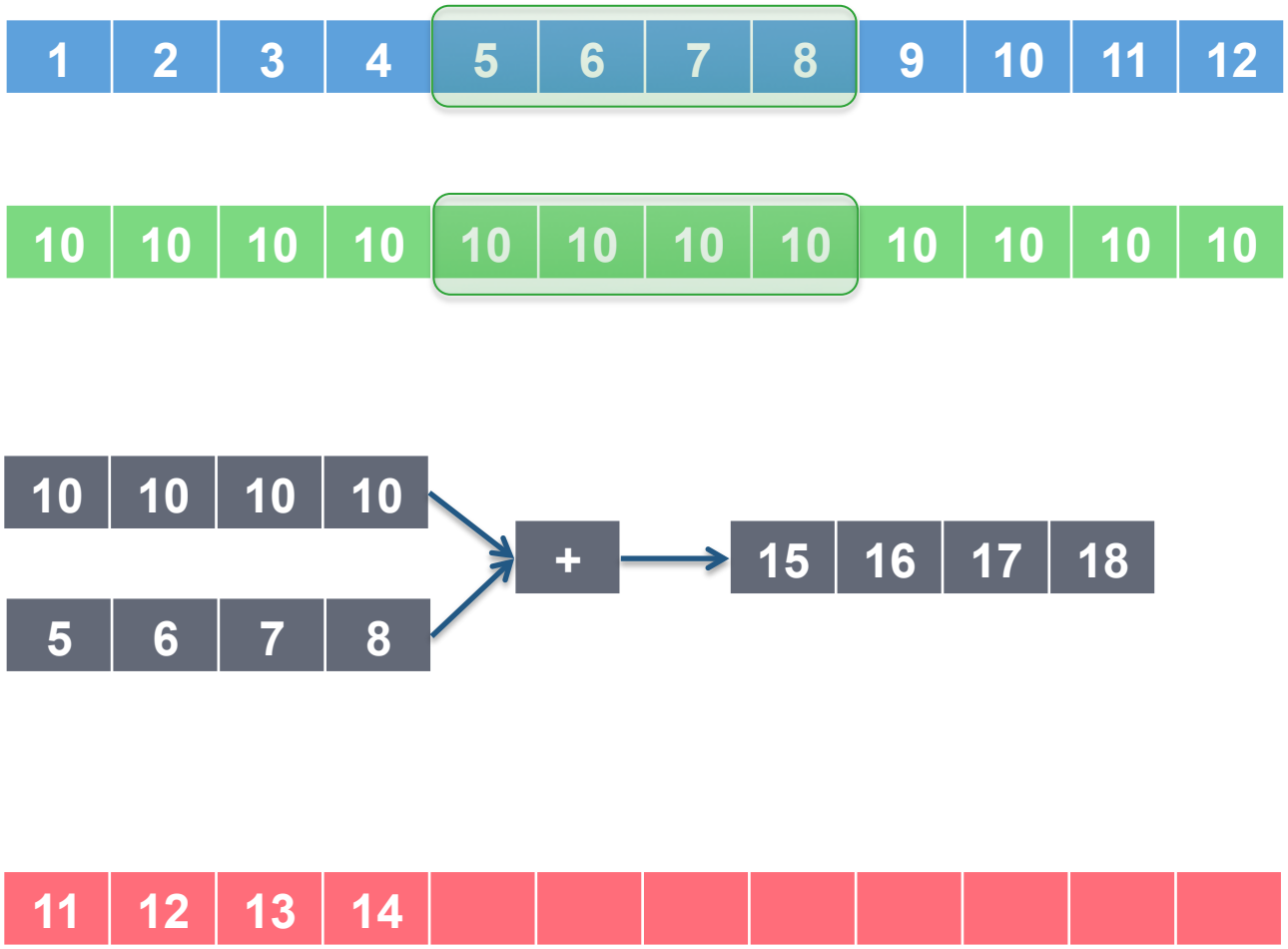

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example



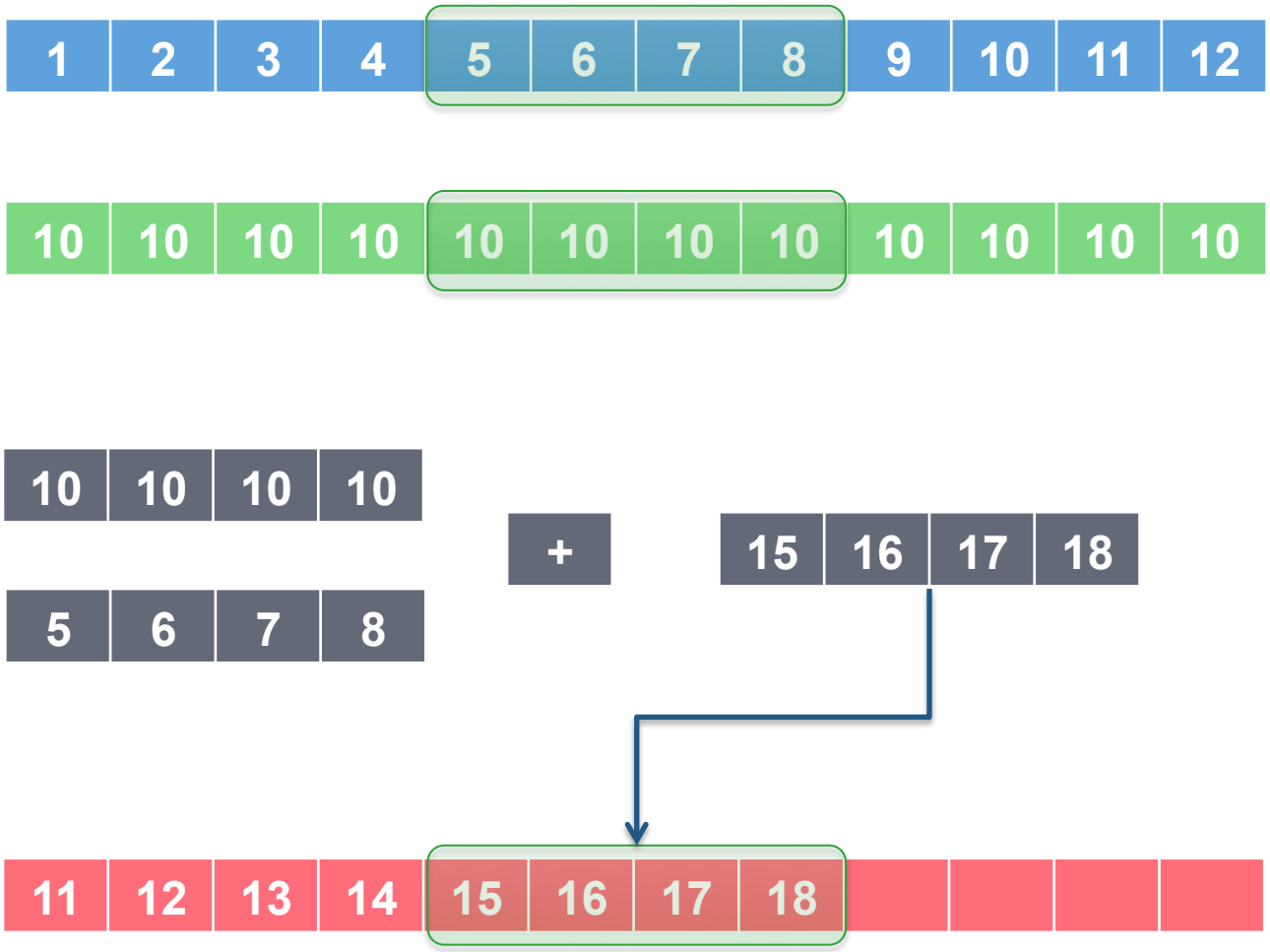

```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example




```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Vectorization example




```
For( i  = 0; i < n; i + 4) {
        Read A[i … i + 4];
        Read B[i … i + 4];
        Addition;
        Write C[i … i + 4];
}
```

Code translation: SSE to AVX

- This optimization targets an old vectorized code running on new processor architecture of the same family.
- Therefore, the optimization algorithm does not vectorize a sequential code (since it is already vectorized by the compiler). In fact, it translates an instruction to its equivalent one which processes more data elements.

SSE version	AVX version
4004f0: xor %eax,%eax 4004f8: movaps 0x603060(%rax),%**x**mm0 4004ff: addps 0x601060(%rax),%**x**mm0 400506: movaps %**x**mm0,0x602060(%rax) 40050d: add $0x**10**,%rax 400511: cmp $0x1000,%rax 400517: jne 4004f8 <vecadd+0x8> 400519: repz retq	4004f0: xor %eax,%eax 4004f8: **v**movaps 0x603060(%rax),%**y**mm0 400500: **v**addps 0x601060(%rax),%**y**mm0,%**y**mm0 400508: **v**movaps %**y**mm0,0x602060(%rax) 400510: add $0x**20**,%rax 400514: cmp $0x1000,%rax 40051a: jne 4004f8 <vecadd+0x8> 40051f: retq

Translation from sse into avx issues

- Translation issues related to detecting sse instructions and translating them with respect to alignement constraints.
- Translation's issues related to the vectorized loop boundaries (iterations, reduction, etc).
- Translation's issues related to aliasing.

Code translation: SSE to AVX instructions

- Padrone tool allows detecting hot code and generates a CFG of the function that encapsulates it.
- The optimizing code traverse the CFG. Once a packed sse instruction is found, its binary is translated into avx form.
- The size of sse and avx instructions are not necessarily equal. Therefore, an algorithm that patches the jumps is invoked during the optimization.
- Few avx instructions require data to be aligned on 32 bytes cache line.

Translation issues related to loops' boundaries

- Case where the number of iterations is known at compile time:
  - Case: #iterations % size of avx register == 0:

SSE version	AVX version
4004f0: xor %eax,%eax 4004f8: movaps 0x603060(%rax),%**x**mm0 4004ff: addps 0x601060(%rax),%**x**mm0 400506: movaps %**x**mm0,0x602060(%rax) 40050d: add $0x**10**,%rax 400511: cmp $0x1000,%rax 400517: jne 4004f8 <vecadd+0x8> 400519: repz retq	4004f0: xor %eax,%eax 4004f8: **v**movaps 0x603060(%rax),%**y**mm0 400500: **v**addps 0x601060(%rax),%**y**mm0,%**y**mm0 400508: **v**movaps %**y**mm0,0x602060(%rax) 400510: add $0x**20**,%rax 400514: cmp $0x1000,%rax 40051a: jne 4004f8 <vecadd+0x8> 40051f: retq

Translation issues related to loops' boundaries

- Case where the number of iterations is known at compile time:
  - Case: #iterations % size of avx register != 0:

SSE version	AVX version
4004f0: xor %eax,%eax 4004f8: movaps 0x603060(%rax),%**x**mm0 4004ff: addps 0x601060(%rax),%**x**mm0 400506: movaps %**x**mm0,0x602060(%rax) 40050d: add $0x**10**,%rax 400511: cmp $0x1010,%rax 400517: jne 4004f8 <vecadd+0x8> 400519: repz retq	4004f0: xor %eax,%eax 4004f8: **v**movaps 0x603060(%rax),%**y**mm0 400500: **v**addps 0x601060(%rax),%**y**mm0,%**y**mm0 400508: **v**movaps %**y**mm0,0x602060(%rax) 400510: add $0x**20**,%rax 400514: cmp $0x1000,%rax 40051a: jne 4004f8 <vecadd+0x8> 4005xx: **movaps 0x603060(%rax),%xmm0** 4005xx: **addps 0x601060(%rax),%xmm0** 4005xx: **movaps %xmm0,0x602060(%rax)** 4005xx: retq

Translation issues related to loops' boundaries

- Case where the number of iterations is known until runtime:

SSE version

```
4004f0:   test   %ecx,%ecx
4004f2:   jle    4005bc <vecadd+0xcc>
4004f8:   lea    0x10(%rdx),%r8
4004fc:   lea    0x10(%rdi),%r11
400500:   mov    %ecx,%r10d
400503:   shr    $0x2,%r10d
400507:   cmp    %r8,%rdi
40050a:   lea    0x0(,%r10,4),%r9d
400512:   setae  %al
400515:   cmp    %r11,%rdx
400518:   setae  %r11b
40051c:   or     %r11d,%eax
40051f:   lea    0x10(%rsi),%r11
400523:   cmp    %r8,%rsi
400526:   setae  %r8b
40052a:   cmp    %r11,%rdx
40052d:   setae  %r11b
400531:   or     %r11d,%r8d
400534:   and    %r8d,%eax
400537:   cmp    $0x6,%ecx
40053a:   seta   %r8b
40053e:   test   %r8b,%al
400541:   je     4005be <vecadd+0xce>
```

```
400543:   test   %r9d,%r9d
400546:   je     4005be <vecadd+0xce>
400548:   xorps  %xmm2,%xmm2
40054b:   xor    %eax,%eax
40054d:   xor    %r8d,%r8d
400550:   movaps %xmm2,%xmm0
400553:   add    $0x1,%r8d
400557:   movaps %xmm2,%xmm1
40055a:   movlps (%rdi,%rax,1),%xmm0
40055e:   movlps (%rsi,%rax,1),%xmm1
400562:   movhps 0x8(%rdi,%rax,1),%xmm0
400567:   movhps 0x8(%rsi,%rax,1),%xmm1
40056c:   addps  %xmm1,%xmm0
40056f:   movlps %xmm0,(%rdx,%rax,1)
400573:   movhps %xmm0,0x8(%rdx,%rax,1)
400578:   add    $0x10,%rax
40057c:   cmp    %r8d,%r10d
40057f:   ja     400550 <vecadd+0x60>
```

```
400581:   cmp    %r9d,%ecx
400584:   je     4005c3 <vecadd+0xd3>
400586:   movslq %r9d,%rax
400589:   shl    $0x2,%rax
40058d:   add    %rax,%rdi
400590:   add    %rax,%rsi
400593:   add    %rax,%rdx
400596:   xor    %eax,%eax
4005a0:   movss  (%rdi,%rax,4),%xmm0
4005a5:   addss  (%rsi,%rax,4),%xmm0
4005aa:   movss  %xmm0,(%rdx,%rax,4)
4005af:   add    $0x1,%rax
4005b3:   lea    (%r9,%rax,1),%r8d
4005b7:   cmp    %r8d,%ecx
4005ba:   jg     4005a0 <vecadd+0xb0>
4005bc:   repz retq
4005be:   xor    %r9d,%r9d
4005c1:   jmp    400586 <vecadd+0x96>
4005c3:   retq
```

Counter Boundary

Counter Boundary

What is aliasing

- « Aliasing describes a situation in which a data location in memory can be accessed through different symbolic names in the program. » - wikipedia.
- Example:

```
for(i = 0; i < NUM_ELTS; ++i) {
        sx[i+4] = sx[i] + sy[i];
}
```

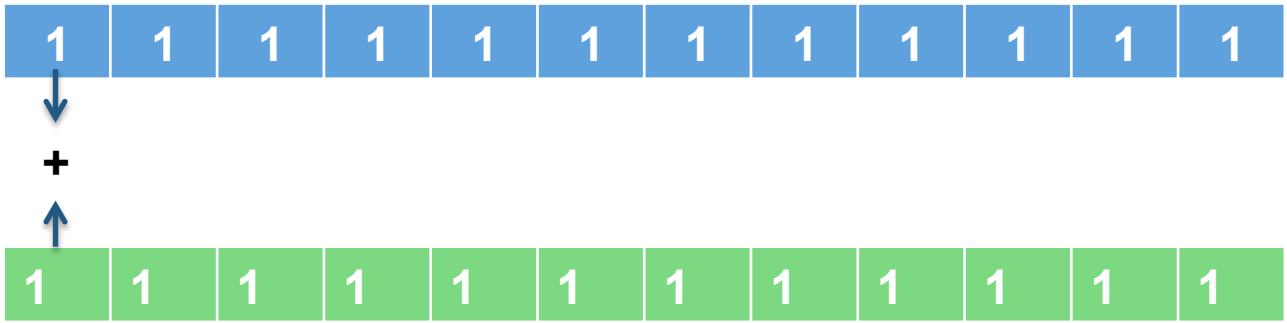
Translation's issues related to aliasing

- A case in which the code is already vectorized for sse architecture but it is not to to translate it into avx form.

Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];

void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```



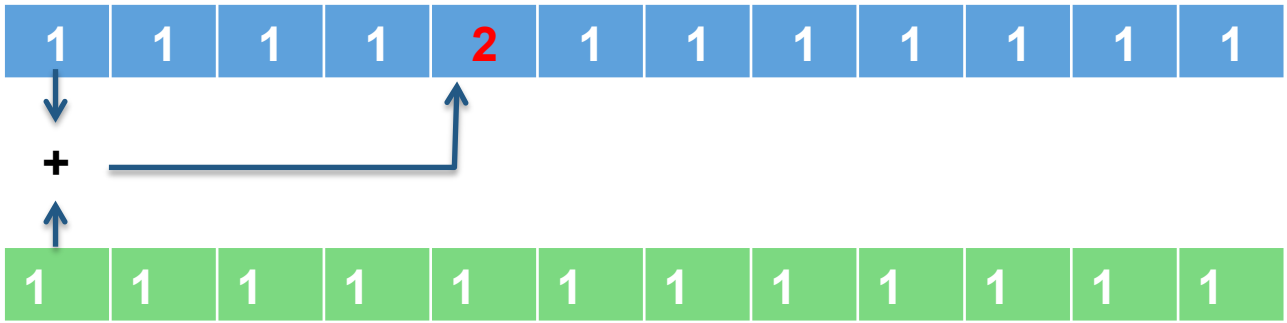
Translation's issues related to aliasing

- A case in which the code is already vectorized for sse architecture but it is not to possible to translate it into avx form.

Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];

void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```



Translation's issues related to aliasing

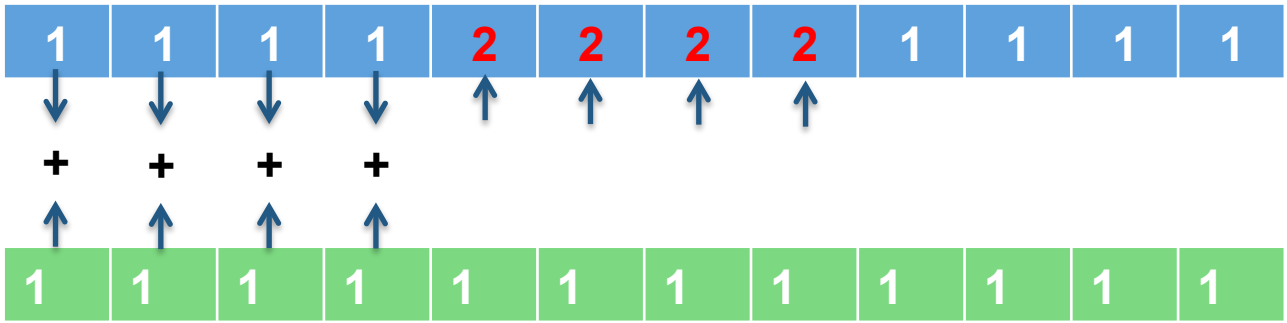
- A case in which the code is already vectorized for sse architecture but it is not to possible to translate it into avx form.

Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];

void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```

After 4 consecutive iterations, the state of array sx[]



Translation's issues related to aliasing

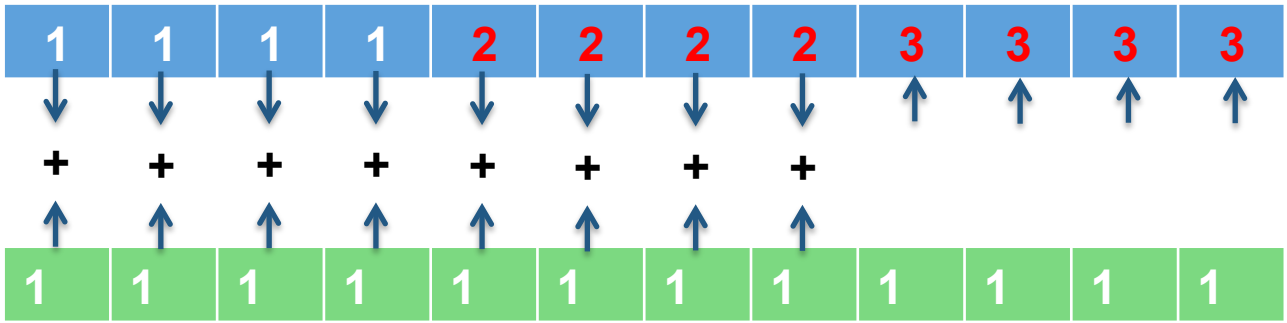
- A case in which the code is already vectorized for sse architecture but it is not to possible to translate it into avx form.

Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];

void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```

After 8 consecutive iterations, the state of array sx[]

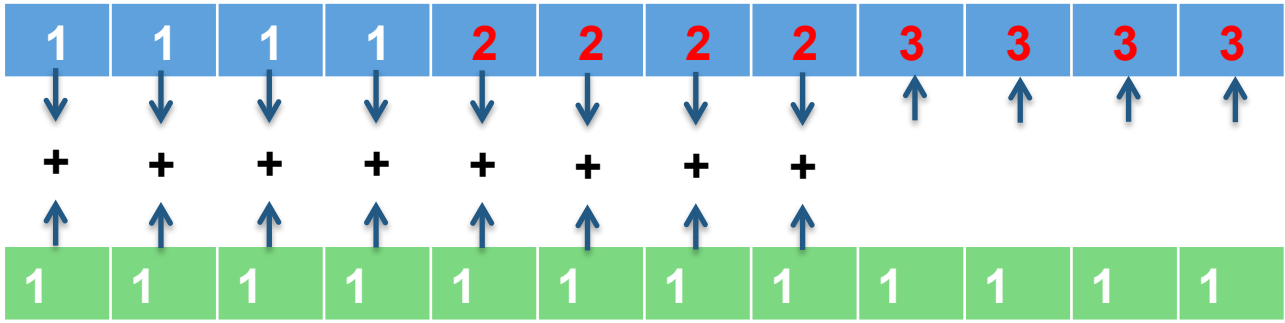


Translation's issues related to aliasing

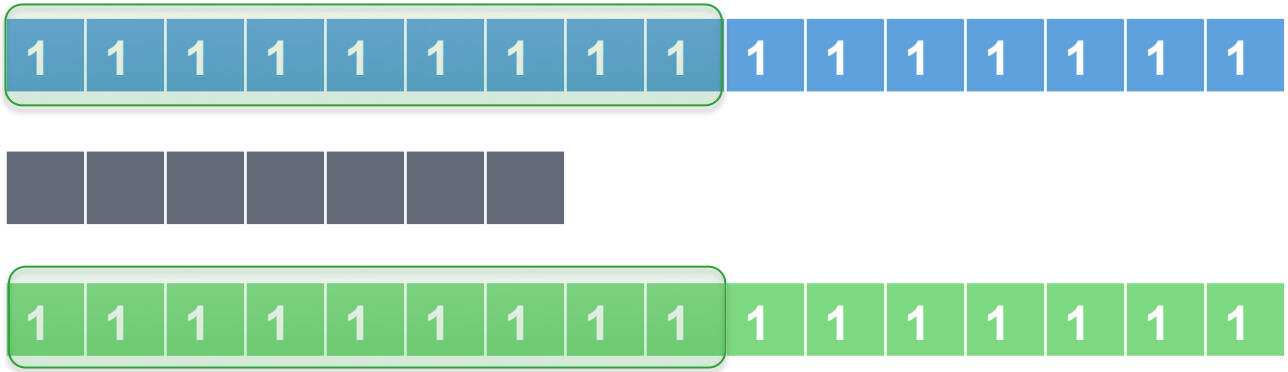
Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];
void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```

+ The sequential execution output:



The vectorized execution by 8 data elements:

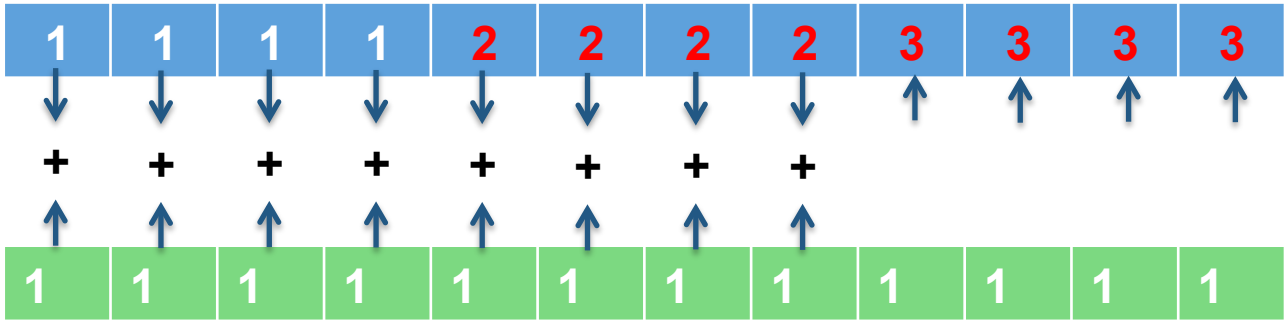


Translation's issues related to aliasing

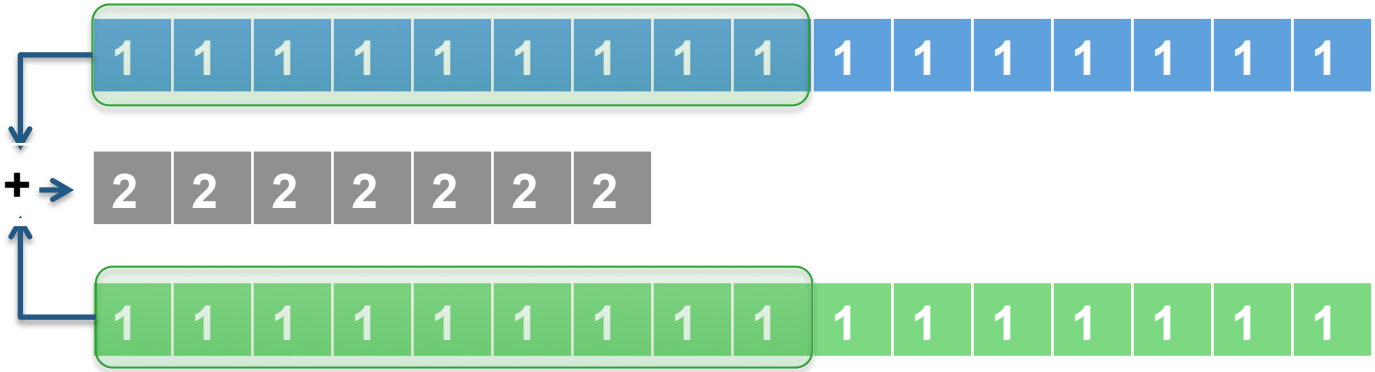
Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];
void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```

+ The sequential execution output:



The vectorized execution by 8 data elements:

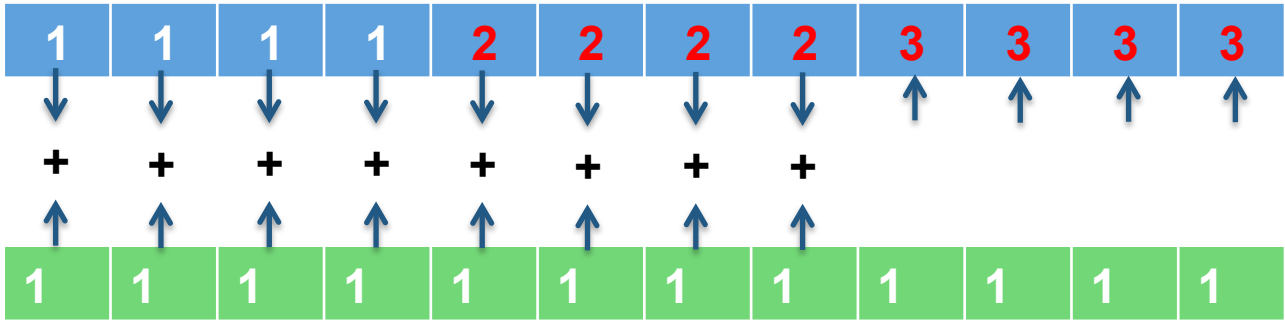


Translation's issues related to aliasing

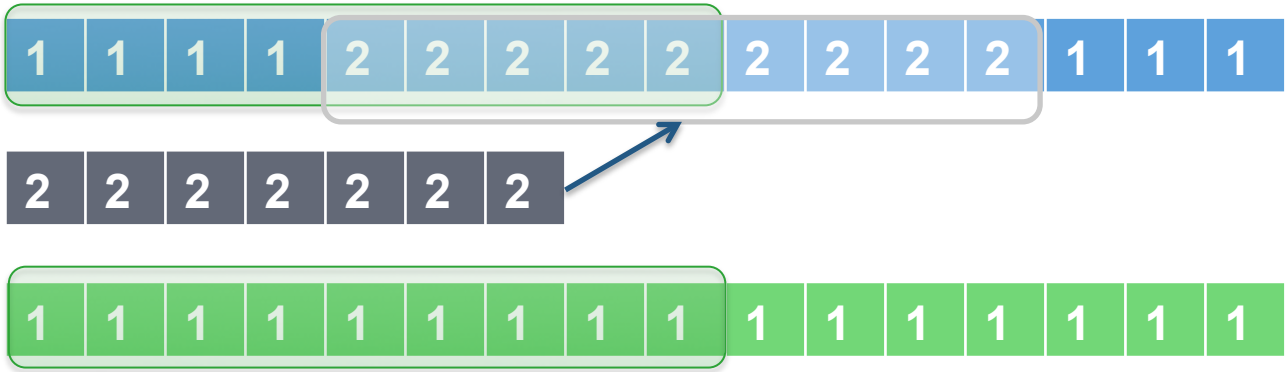
Example:

```
#define NUM_ELTS 10000
float sx[NUM_ELTS];
float sy[NUM_ELTS];
float sz[NUM_ELTS];
void vecadd(void) {
        int i;
        for(i=0; i < NUM_ELTS; ++i) {
                sx[i+4] = sx[i] + sy[i];
        }
}
```

+ The sequential execution output:



The vectorized execution by 8 data elements:



Translation's issues related to aliasing

- Detecting the instructions to modify which are involved in aliasing tests:

SSE version

```
4004f0:   test  %ecx,%ecx
4004f2:   jle   4005bc <vecadd+0xcc>
4004f8:   lea   0x10(%rdx),%r8
4004fc:   lea   0x10(%rdi),%r11
400500:   mov   %ecx,%r10d
400503:   shr   $0x2,%r10d
400507:   cmp   %r8,%rdi
40050a:   lea   0x0(,%r10,4),%r9d
400512:   setae %al
400515:   cmp   %r11,%rdx
400518:   setae %r11b
40051c:   or    %r11d,%eax
40051f:   lea   0x10(%rsi),%r11
400523:   cmp   %r8,%rsi
400526:   setae %r8b
40052a:   cmp   %r11,%rdx
40052d:   setae %r11b
400531:   or    %r11d,%r8d
400534:   and   %r8d,%eax
400537:   cmp   $0x6,%ecx
40053a:   seta  %r8b
40053e:   test  %r8b,%al
400541:   je    4005be <vecadd+0xce>
```

```
400543:   test  %r9d,%r9d
400546:   je    4005be <vecadd+0xce>
400548:   xorps %xmm2,%xmm2
40054b:   xor   %eax,%eax
40054d:   xor   %r8d,%r8d
400550:   movaps %xmm2,%xmm0
400553:   add   $0x1,%r8d
400557:   movaps %xmm2,%xmm1
40055a:   movlps (%rdi,%rax,1),%xmm0
40055e:   movlps (%rsi,%rax,1),%xmm1
400562:   movhps 0x8(%rdi,%rax,1),%xmm0
400567:   movhps 0x8(%rsi,%rax,1),%xmm1
40056c:   addps %xmm1,%xmm0
40056f:   movlps %xmm0,(%rdx,%rax,1)
400573:   movhps %xmm0,0x8(%rdx,%rax,1)
400578:   add   $0x10,%rax
40057c:   cmp   %r8d,%r10d
40057f:   ja    400550 <vecadd+0x60>
```

```
400581:   cmp   %r9d,%ecx
400584:   je    4005c3 <vecadd+0xd3>
400586:   movslq %r9d,%rax
400589:   shl   $0x2,%rax
40058d:   add   %rax,%rdi
400590:   add   %rax,%rsi
400593:   add   %rax,%rdx
400596:   xor   %eax,%eax
4005a0:   movss (%rdi,%rax,4),%xmm0
4005a5:   addss (%rsi,%rax,4),%xmm0
4005aa:   movss %xmm0,(%rdx,%rax,4)
4005af:   add   $0x1,%rax
4005b3:   lea   (%r9,%rax,1),%r8d
4005b7:   cmp   %r8d,%ecx
4005ba:   jg    4005a0 <vecadd+0xb0>
4005bc:   repz retq
4005be:   xor   %r9d,%r9d
4005c1:   jmp   400586 <vecadd+0x96>
4005c3:   retq
```

Reduction example: Array max

```
4004f0:    movdqa 0x348(%rip),%xmm0
4004f8:    mov    $0x601060,%eax
4004fd:    nopl   (%rax)
400500:    pmaxsw (%rax),%xmm0
400504:    add    $0x10,%rax
400508:    cmp    $0x601860,%rax
40050e:    jne    400500 <max_s16+0x10>
```

Vectorized loop

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

Reduction

12	15	23	31	2	7	21	17	13	55	6	25	28	19	62	9

Min	Min	Min	Min	Min	Min	Min	Min

Reduction example: Array max

```
4004f0:    movdqa 0x348(%rip),%xmm0
4004f8:    mov    $0x601060,%eax
4004fd:    nopl   (%rax)
400500:    pmaxsw (%rax),%xmm0
400504:    add    $0x10,%rax
400508:    cmp    $0x601860,%rax
40050e:    jne    400500 <max_s16+0x10>
```

Vectorized loop

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

Reduction

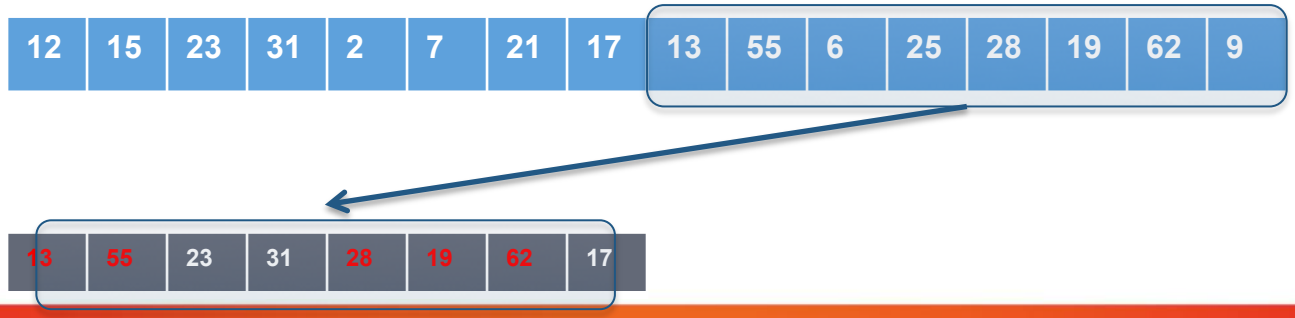
Reduction example: Array max

```
4004f0:    movdqa 0x348(%rip),%xmm0
4004f8:    mov    $0x601060,%eax
4004fd:    nopl   (%rax)
400500:    pmaxsw (%rax),%xmm0
400504:    add    $0x10,%rax
400508:    cmp    $0x601860,%rax
40050e:    jne    400500 <max_s16+0x10>
```

Vectorized loop

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

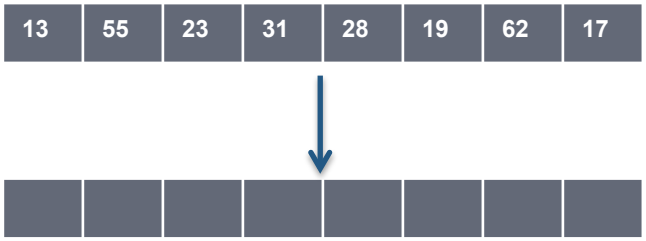
Reduction



Reduction example: Array max

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

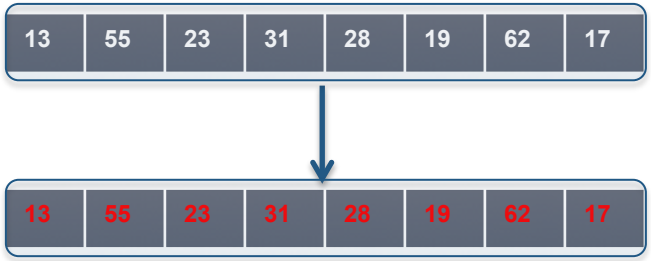
Reduction



Reduction example: Array max

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

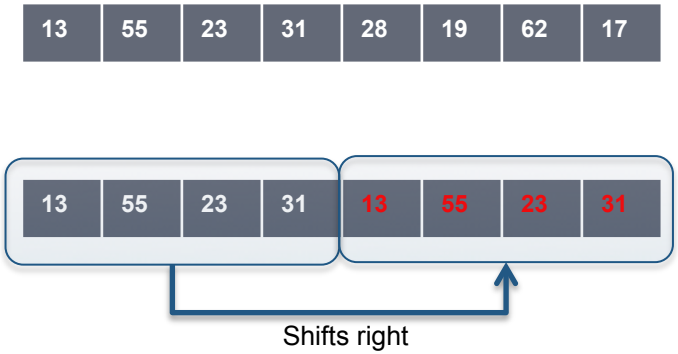
Reduction



Reduction example: Array max

```
400510:   movdqa %xmm0,%xmm1
400514:   psrldq $0x8,%xmm1
400519:   pmaxsw %xmm1,%xmm0
40051d:   movdqa %xmm0,%xmm1
400521:   psrldq $0x4,%xmm1
400526:   pmaxsw %xmm1,%xmm0
40052a:   movdqa %xmm0,%xmm1
40052e:   psrldq $0x2,%xmm1
400533:   pmaxsw %xmm1,%xmm0
400537:   pextrw $0x0,%xmm0,%eax
40053c:   retq
40053d:   nopl   (%rax)
```

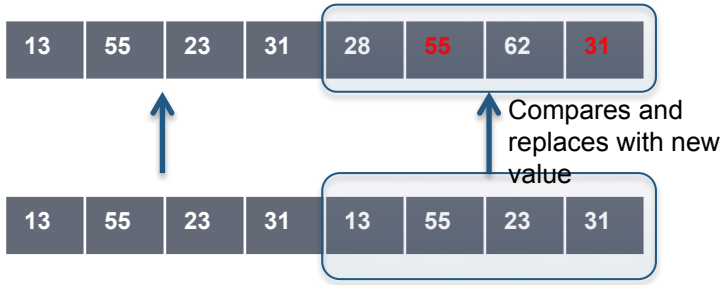
Reduction



Reduction example: Array max

```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

Reduction



Reduction example: Array max

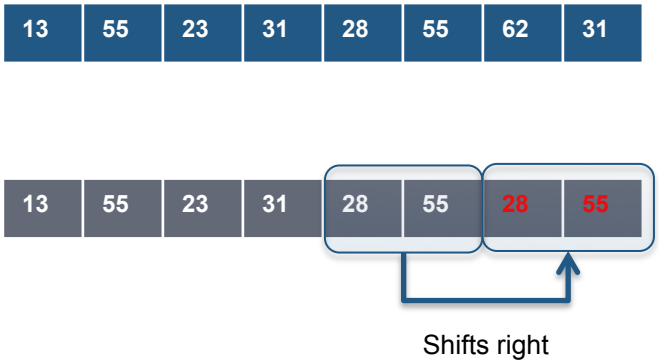
```
400510:    movdqa %xmm0,%xmm1
400514:    psrldq $0x8,%xmm1
400519:    pmaxsw %xmm1,%xmm0
40051d:    movdqa %xmm0,%xmm1
400521:    psrldq $0x4,%xmm1
400526:    pmaxsw %xmm1,%xmm0
40052a:    movdqa %xmm0,%xmm1
40052e:    psrldq $0x2,%xmm1
400533:    pmaxsw %xmm1,%xmm0
400537:    pextrw $0x0,%xmm0,%eax
40053c:    retq
40053d:    nopl   (%rax)
```

Reduction

Reduction example: Array max

```
400510:   movdqa %xmm0,%xmm1
400514:   psrldq $0x8,%xmm1
400519:   pmaxsw %xmm1,%xmm0
40051d:   movdqa %xmm0,%xmm1
400521:   psrldq $0x4,%xmm1
400526:   pmaxsw %xmm1,%xmm0
40052a:   movdqa %xmm0,%xmm1
40052e:   psrldq $0x2,%xmm1
400533:   pmaxsw %xmm1,%xmm0
400537:   pextrw $0x0,%xmm0,%eax
40053c:   retq
40053d:   nopl   (%rax)
```

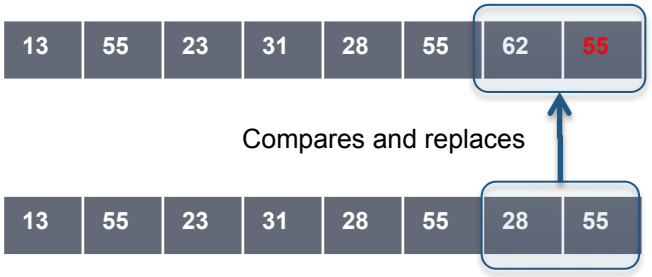
Reduction



Reduction example: Array max

```
400510:   movdqa %xmm0,%xmm1
400514:   psrldq $0x8,%xmm1
400519:   pmaxsw %xmm1,%xmm0
40051d:   movdqa %xmm0,%xmm1
400521:   psrldq $0x4,%xmm1
400526:   pmaxsw %xmm1,%xmm0
40052a:   movdqa %xmm0,%xmm1
40052e:   psrldq $0x2,%xmm1
400533:   pmaxsw %xmm1,%xmm0
400537:   pextrw $0x0,%xmm0,%eax
40053c:   retq
40053d:   nopl   (%rax)
```

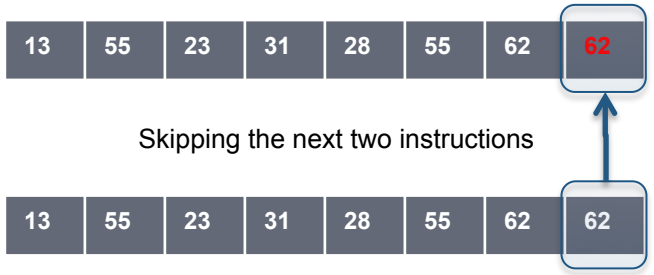
Reduction



Reduction example: Array max

```
400510:   movdqa %xmm0,%xmm1
400514:   psrldq $0x8,%xmm1
400519:   pmaxsw %xmm1,%xmm0
40051d:   movdqa %xmm0,%xmm1
400521:   psrldq $0x4,%xmm1
400526:   pmaxsw %xmm1,%xmm0
40052a:   movdqa %xmm0,%xmm1
40052e:   psrldq $0x2,%xmm1
400533:   pmaxsw %xmm1,%xmm0
400537:   pextrw $0x0,%xmm0,%eax
40053c:   retq
40053d:   nopl   (%rax)
```

Reduction



Conclusion

- To sum up, in our work we want to investigate the possibility of optimizing a running application. Motivated by two main reasons: backward compatibility of the same family of processor architecture and runtime information.

References

http://software.intel.com/en-us/blogs/2012/01/31/

vectorization-find-out-what-it-is-find-out-more