SLP Straight-Line Code Vectorizer

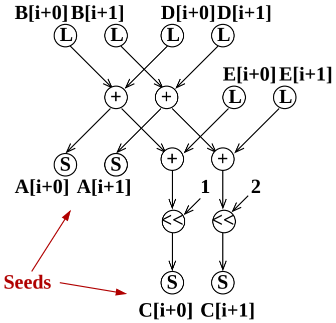
- Superword Level Parallelism [Larsen PLDI'00]
- GCC and LLVM implementations are based on Bottom-Up SLP
- SLP and the loop-vectorizer complement each other:
  - Unroll loop and vectorize with SLP
  - Even if loop-vectorizer fails, SLP could partly succeed
- It is missing features present in the Loop vectorizer (e.g., Interleaved Loads, Predication)
  - Usually run SLP after the Loop Vectorizer

SLP is usually good enough

```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0]  = (tmp1 + E[i+0]) << 1;
C[i+1]  = (tmp2 + E[i+1]) << 2;
```

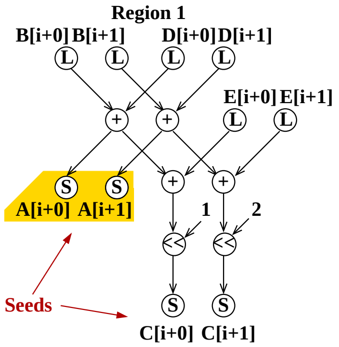
SLP is usually good enough

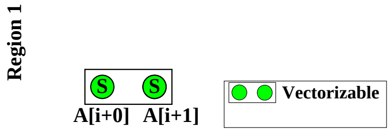
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```



SLP is usually good enough

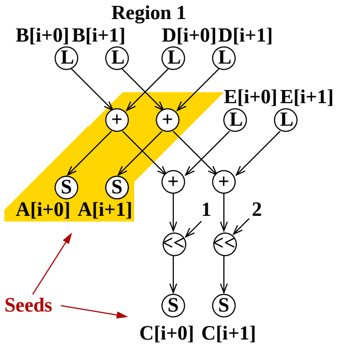
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0]  = (tmp1 + E[i+0]) << 1;
C[i+1]  = (tmp2 + E[i+1]) << 2;
```

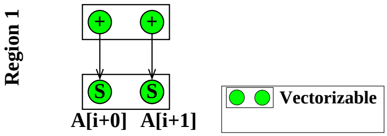




SLP is usually good enough

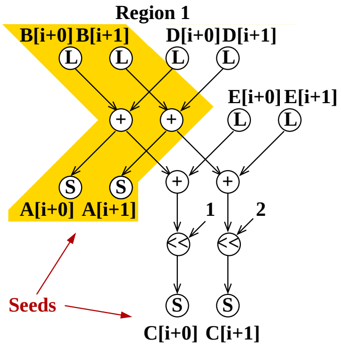
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```

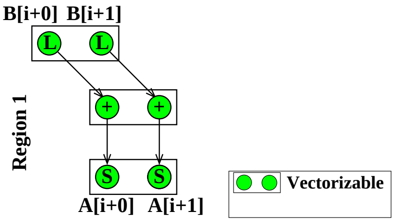




SLP is usually good enough

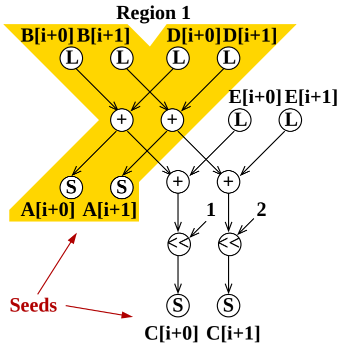
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```

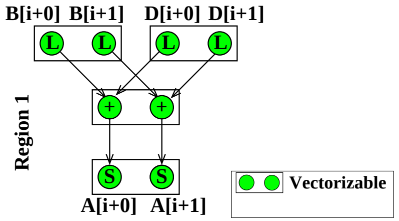




SLP is usually good enough

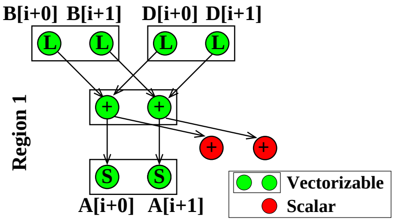
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```

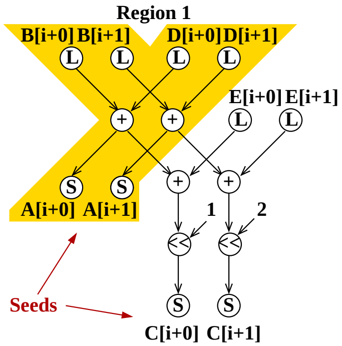




SLP is usually good enough

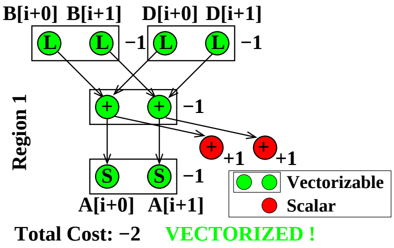
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```



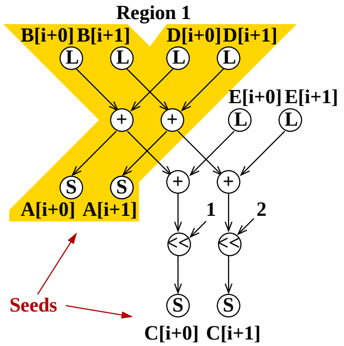


SLP is usually good enough

```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```

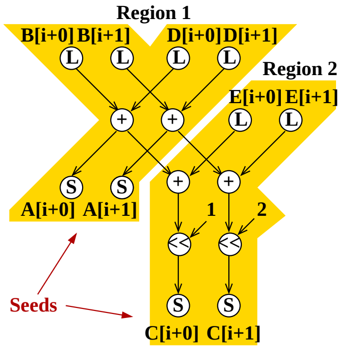


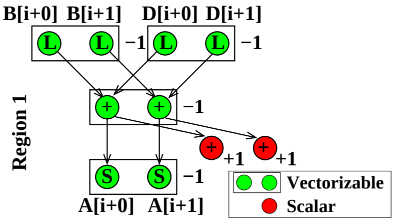
Total Cost: −2 VECTORIZED !



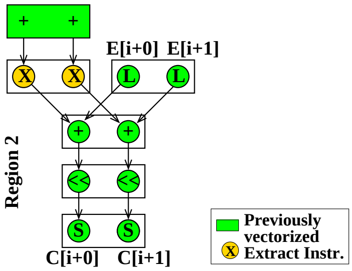
SLP is usually good enough

```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0]  = (tmp1 + E[i+0]) << 1;
C[i+1]  = (tmp2 + E[i+1]) << 2;
```





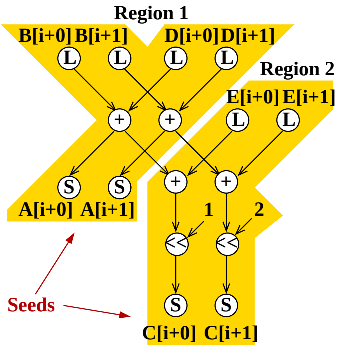
Total Cost: −2 VECTORIZED !

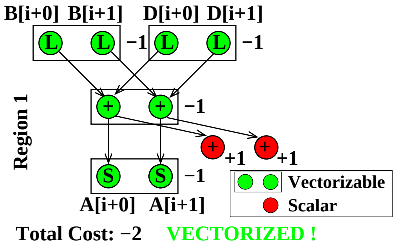


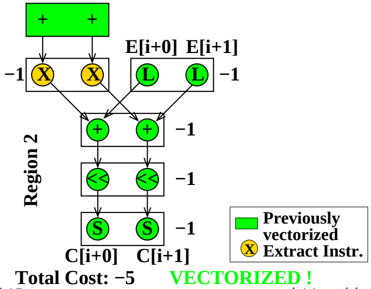
SLP is usually good enough

```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;

C[i+0] = (tmp1 + E[i+0]) << 1;
C[i+1] = (tmp2 + E[i+1]) << 2;
```

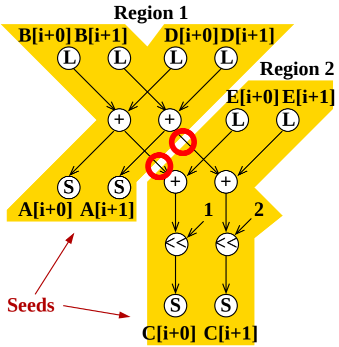


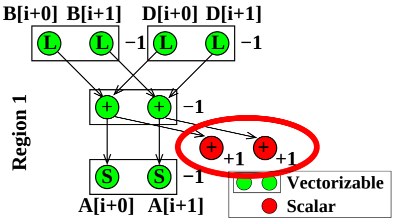




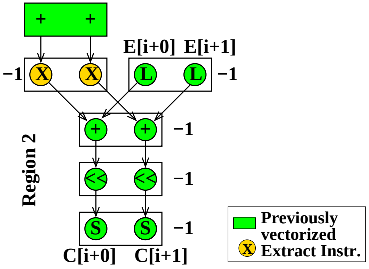
SLP is usually good enough

```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0]  = (tmp1 + E[i+0]) << 1;
C[i+1]  = (tmp2 + E[i+1]) << 2;
```





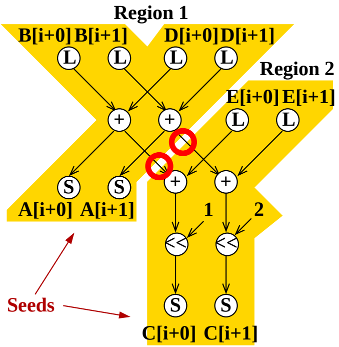
Total Cost: −2 VECTORIZED !



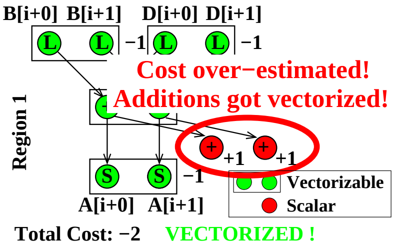
Total Cost: −5 VECTORIZED !

SLP is usually good enough

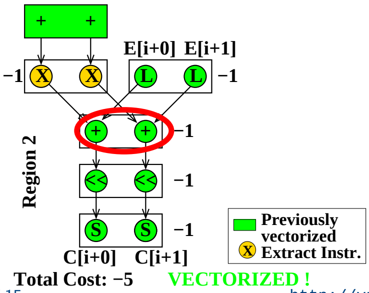
```
long tmp1, tmp2, A[], B[], C[], D[], E[];
tmp1 = B[i+0] + D[i+0];
tmp2 = B[i+1] + D[i+1];
A[i+0] = tmp1;
A[i+1] = tmp2;
C[i+0]  = (tmp1 + E[i+0]) << 1;
C[i+1]  = (tmp2 + E[i+1]) << 2;
```



Region 1
B[i+0] B[i+1]
D[i+0] D[i+1]
Region 2
E[i+0] E[i+1]
1
2
A[i+0] A[i+1]
C[i+0] C[i+1]
Seeds



B[i+0] B[i+1]
D[i+0] D[i+1]
−1
−1
Cost over−estimated!
Additions got vectorized!
Region 1
+1
+1
−1
A[i+0] A[i+1]
Vectorizable
Scalar
Total Cost: −2
VECTORIZED !



E[i+0] E[i+1]
−1
−1
−1
−1
−1
Region 2
C[i+0] C[i+1]
Previously vectorized
Extract Instr.
Total Cost: −5
VECTORIZED !