reason why there are so many page pulls is that once a page is swapped into physical memory, it
is not reused by another reference before the page is swapped out.

The array A can be mapped into memory by block-mapping as follows:

\[
\begin{align*}
\text{II} &= 0, 15 \quad \text{BY} \quad 4 \\
\text{JJ} &= 0, 15 \quad \text{BY} \quad 4 \\
\text{KK} &= 0, 15 \quad \text{BY} \quad 4 \\
\text{I} &= \text{II}, \min(\text{II} + \text{B}-1, 15) \\
\text{J} &= \text{JJ}, \min(\text{JJ} + \text{B}-1, 15) \\
\text{K} &= \text{KK}, \min(\text{KK} + \text{B}-1, 15) \\
\text{A}(\text{K}, \text{J}, \text{I}) &= \text{I} + \text{J} + \text{K}
\end{align*}
\]

The blocking factor we choose here is 4 for each of the three dimensions; thus, \(64 = 4 \times 4 \times 4\) elements form a block in real memory. The page reference sequence for indexed elements of A during the first 16 iterations now becomes:

<table>
<thead>
<tr>
<th>Indexed Element</th>
<th>Relative Address</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A(0, 0, 0))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(A(1, 0, 0))</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>(A(2, 0, 0))</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>(A(3, 0, 0))</td>
<td>48</td>
<td>6</td>
</tr>
<tr>
<td>(A(0, 1, 0))</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>(A(1, 1, 0))</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>(A(2, 1, 0))</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>(A(3, 1, 0))</td>
<td>52</td>
<td>6</td>
</tr>
<tr>
<td>(A(0, 2, 0))</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>(A(1, 2, 0))</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>(A(2, 2, 0))</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>(A(3, 2, 0))</td>
<td>56</td>
<td>7</td>
</tr>
<tr>
<td>(A(0, 3, 0))</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>(A(1, 3, 0))</td>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>(A(2, 3, 0))</td>
<td>44</td>
<td>5</td>
</tr>
<tr>
<td>(A(3, 3, 0))</td>
<td>60</td>
<td>7</td>
</tr>
</tbody>
</table>

From the above table list, within the first 16 iterations, all the referenced pages are relatively close
and can be reused for the next 48 iterations. Hence, by use of the block-mapping strategy, the I/O
behavior of a program with dense loop nests can be improved significantly.

### 3. The Mapping Algorithm

We present the algorithm that implements the block mapping we used in the above example.
Because of space limitations, we will not develop it step by step; for this we must refer to the
original thesis [5]. We simply formulate the algorithm in pseudo-code. We contrast it with the
corresponding pseudo-code for row- and column-major mapping.

Algorithm: **Row-Major Mapping**

Input:  \(k\): the dimension of the array to be mapped  
D: a vector of \(k\) index variables  
M: a vector of ranges, one for each dimension
Output: linear address mapped by row-major mapping strategy

Method:

```haskell
function address_row(var D: Array, M: Array, k: integer): integer;
    var address, temp, i: integer;
begin
    address := D[k-1]; temp := 1;
    for i := k-2 downto 0 do
        temp := temp * M[i+1]; address := address + D[i] * temp
    end; {for i}
    address_row := address
end; {address_row( )}
```

Algorithm: Column-Major Mapping

Input: k: the dimension of the array to be mapped  
D: a vector of k index variables  
M: a vector of ranges, one for each dimension

Output: linear address mapped by column-major mapping strategy

Method:

```haskell
function address_column(var D: Array, M: Array, k: integer): integer;
    var address, temp, i: integer;
begin
    address := D[0]; temp := 1;
    for i := 1 to k-1 do
        temp := temp * M[i-1]; address := address + D[i] * temp
    end; {for i}
    address_column := address
end; {address_column( )}
```

It should be clear that these are just the standard mapping functions used in all compilers for Fortran (column-major) and all other languages (row-major). In both cases, the algorithm takes the k indices that describe a reference into a k-dimensional array (element) and compute a single integer between 0 (inclusive) and the size of the entire array to be mapped (exclusive). As always, we assume C conventions, i. e., array ranges in all dimensions start with 0.

Below follows the algorithm for the block mapping. It follows along very similar lines as the previous two, except now we need also a vector of blocking factors. This permits us to have a different value in each dimension. in the above example where k=3, the vector of blocking factors is (4,4,4), i. e., the three blocking factors are the same. In general, they can be different. For a general development of the algorithm, we refer to [5].

Algorithm: Block-Mapping

Input: k: the dimension of the array to be mapped  
D: a vector of k index variables  
M: a vector of ranges, one for each dimension  
B: a vector of blocking factors