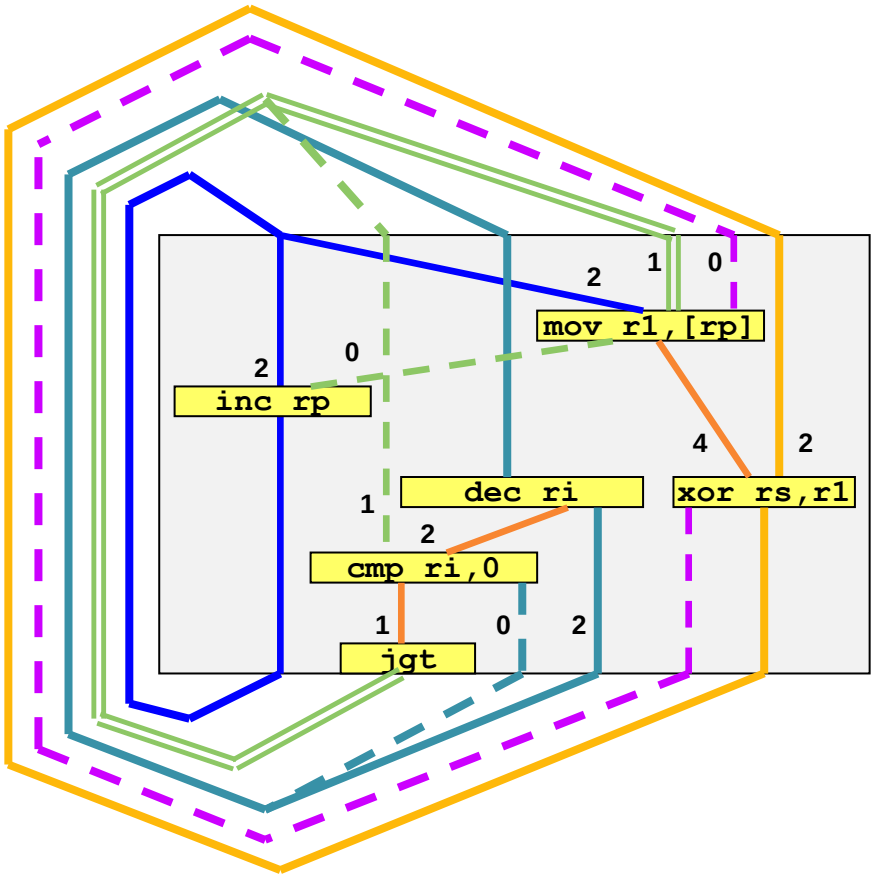
Optimization of nested loops

Dependency analysis

- Dependency
  - The need to execute one instruction after another
    - Associated with a latency = the minimal time difference between the instructions
  - Partial ordering of instructions
- Data dependency – passing a value through a (temporary) variable
  - Write-Read
- Anti-dependency – no value passed but protecting the effect
  - Read-Write
  - Write-Write
- Control dependency
  - Condition-Operation: Waiting to confirm that the operation is requested
    - For operations that cannot be undone – writing memory, possible faults, ...

- Usually analyzed over a loop

Example




```
char chksum(
  char * rp, int ri)
{
  char rs = 0;
  while ( ri > 0 )
  {
    char r1 = *rp++;
    rs ^= r1;
    --ri;
  }
  return rs;
}
```

- Dependencies
  - inside an iteration
  - across iterations
- Cyclic graph

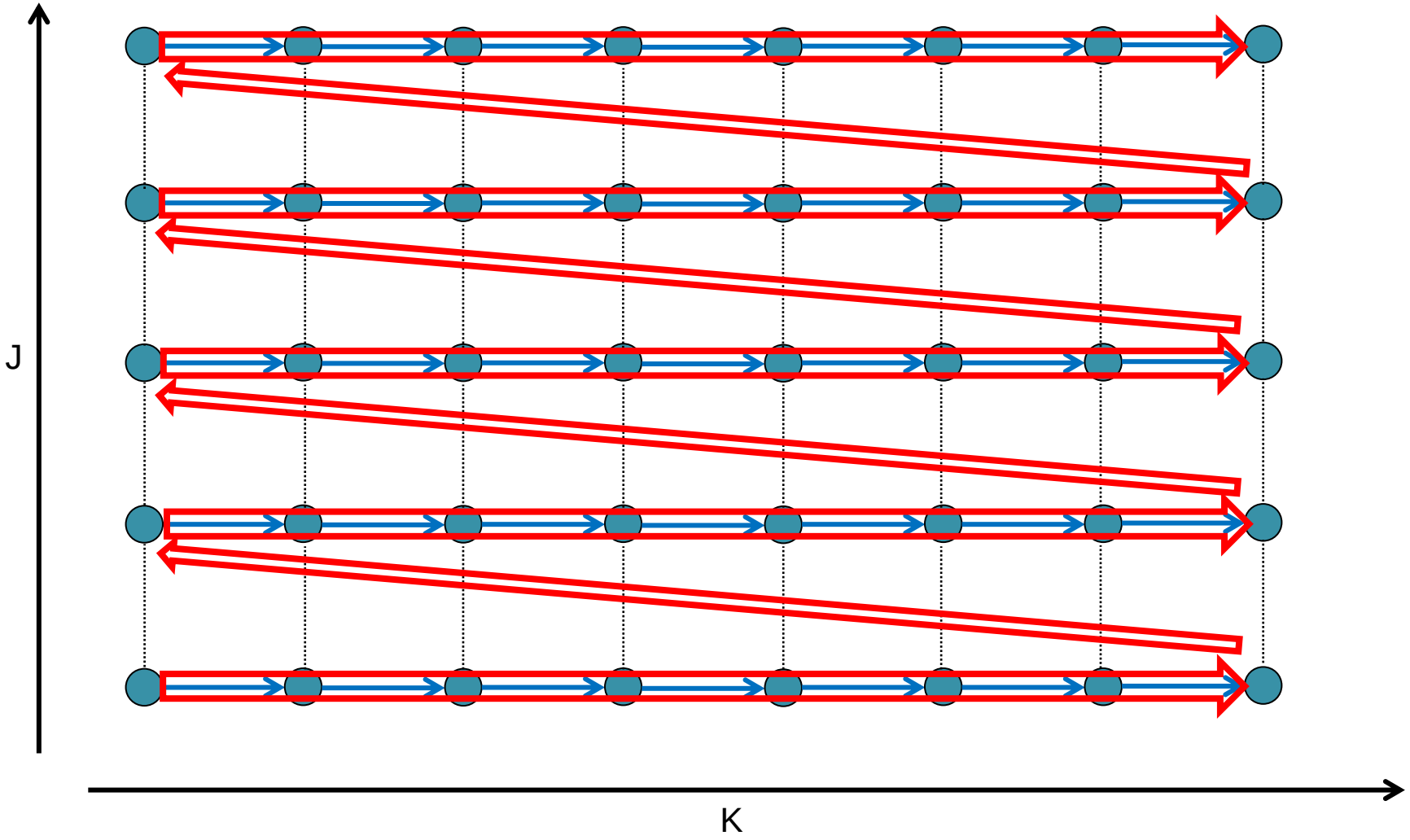
Example – vector-by-matrix multiplication

```
for J := 1 to M do
  for K := 1 to N do
    C[J] := C[J] + A[J]*B[J,K]

for J := 1 to M do
begin
  S := C[J]
  for K := 1 to N do
    S := S + A[J]*B[J,K]
  C[J] := S
end

for K := 1 to N do
  for J := 1 to M do
    C[J] := C[J] + A[J]*B[J,K]
```

- Critical dependency cycle
  - The cycle with the greatest latency
    - Alternating reading and writing of the same element C[J]
  - An inner loop iteration can never be faster than the latency of the read operation
    - The latency of writes is usually 0
- Transformation to a local variable placed in a register
    - Note: this is not an equivalent transformation if C and A may be aliased
  - Improves latency but does not remove the critical dependency cycle
- Loop reversal
    - Not equivalent in the presence of aliasing
  - Removes the dependency cycle completely
    - It is now present in the outer loop

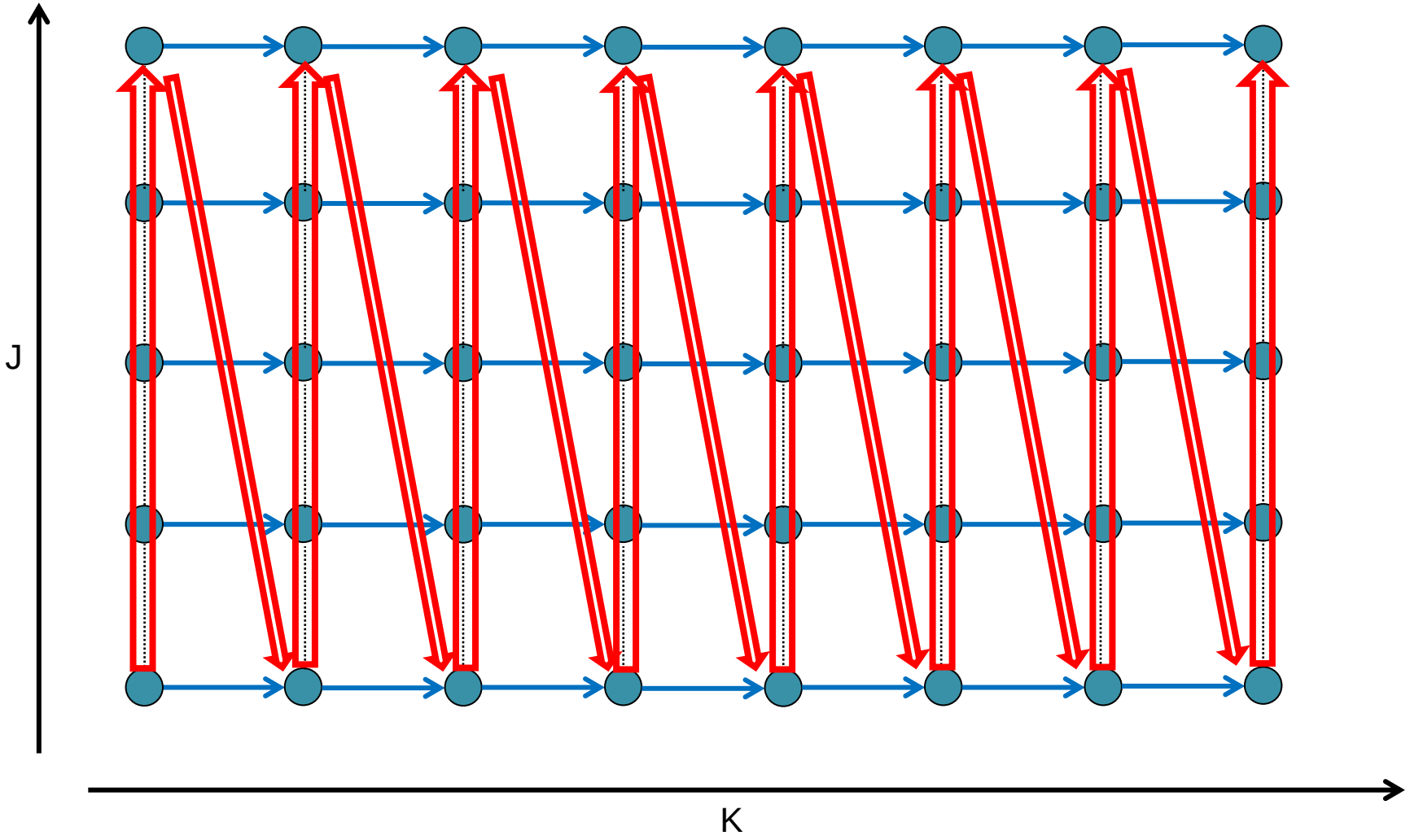
Loop reversal

- The original pass through the iteration space
  - Iteration space = possible combinations of control-variable values
  - Most neighbors are dependent



Loop reversal

- The order after the loop reversal
  - Most neighbors are independent

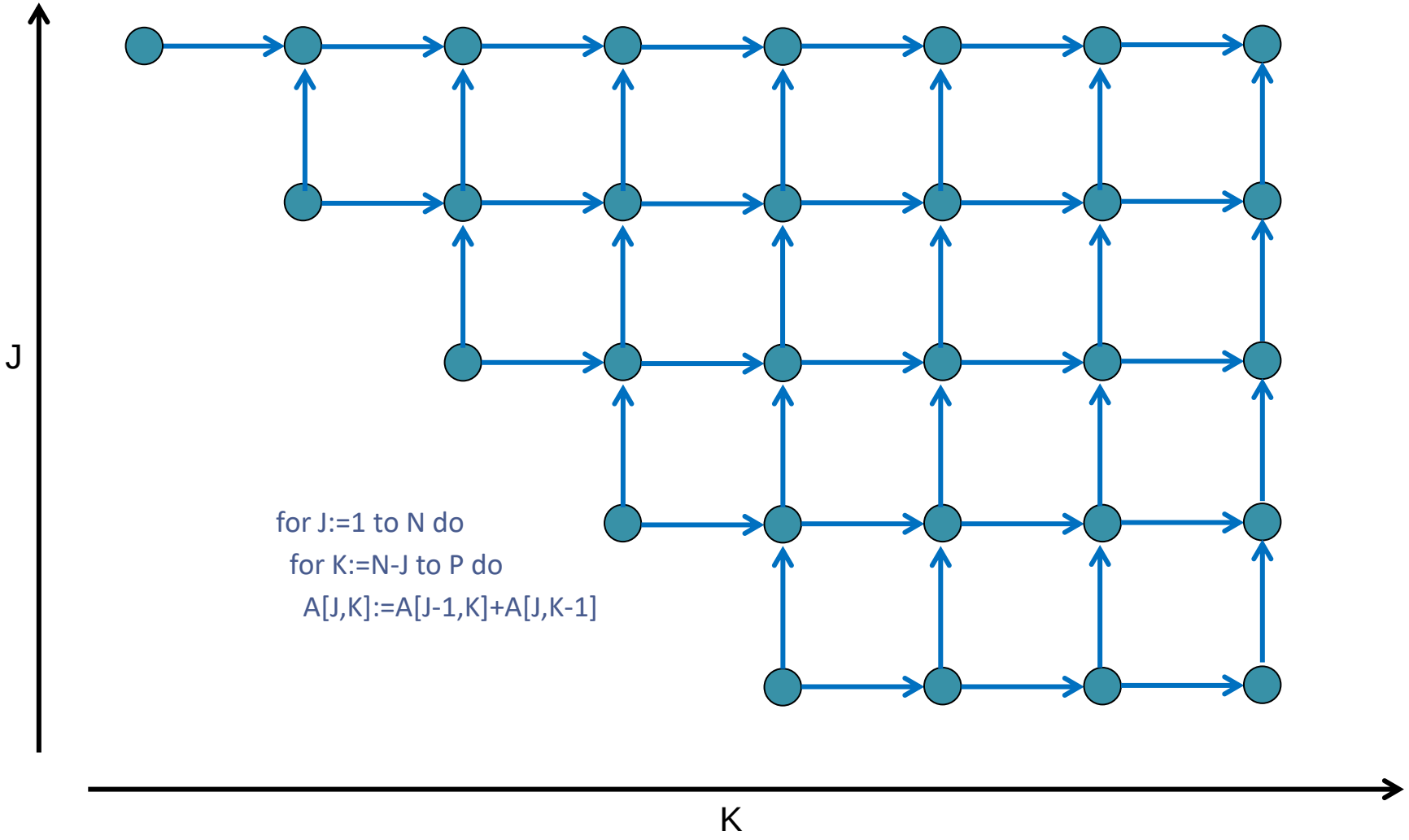


“Parallel” bsearch

```
for ( i = 0; i < N; ++ i )
   bsearch( a, M, b[ i]);

void bsearch( a, M, x)
{
   while ( /*...*/ )
   {
     if ( a[ j] > x )
          j = /*...*/;
     else
          j = /*...*/;
   }
}
```

```
bsearch_many( a, M, b, N);

void bsearch_many( a, M, b, N)
{
   while ( /*???*/ )
     for ( i = 0; i < N; ++ i )
     {
          if ( a[ j[ i]] > b[ i] )
            j[ i] = /*...*/;
          else
            j[ i] = /*...*/;
     }
}
```

A more general example



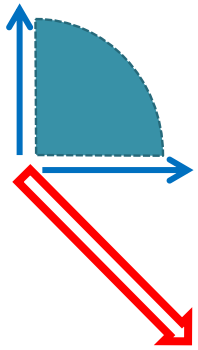
Loop skewing

- *Polyhedral compilation* (generalized Loop skewing)

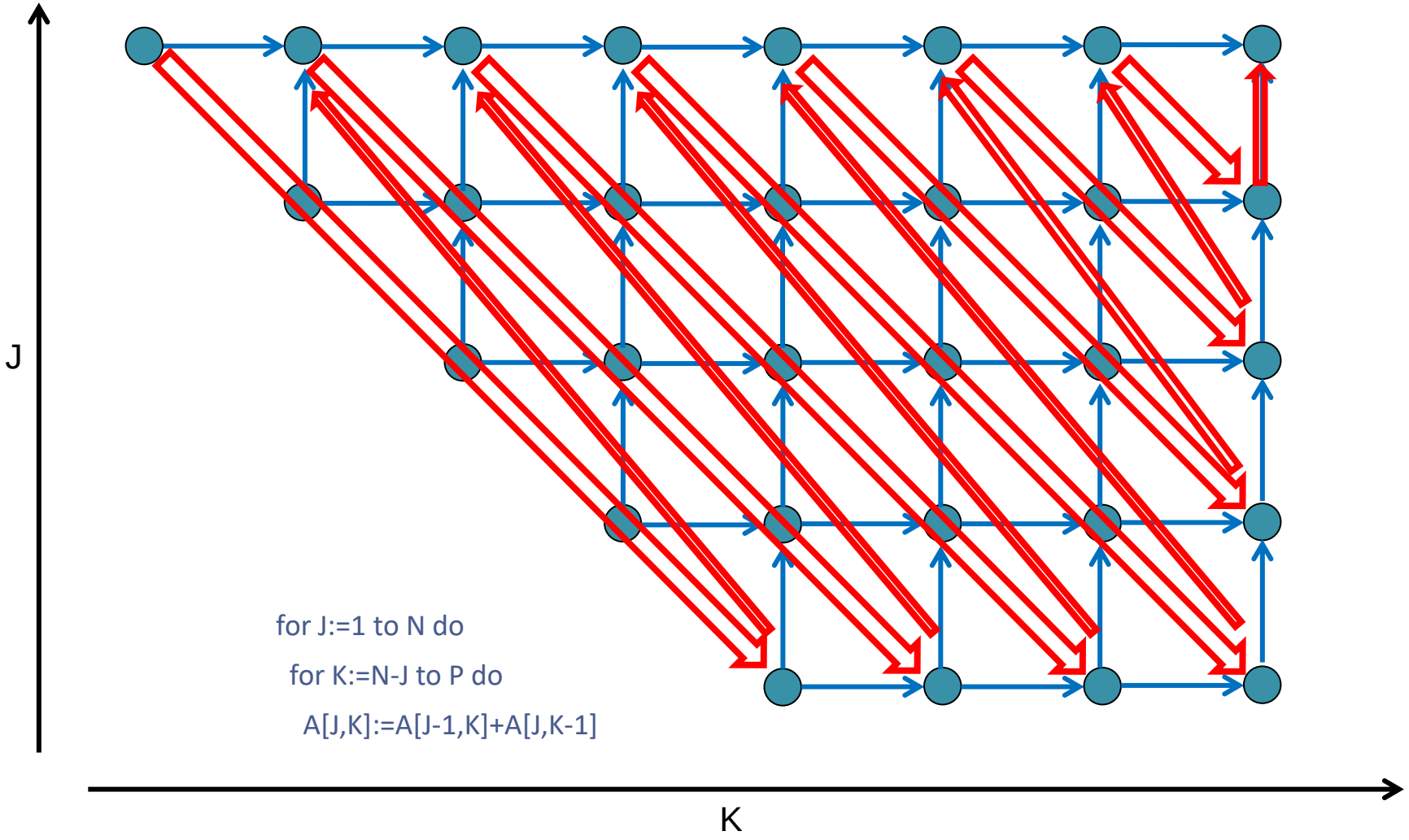
```
for J:=1 to N do
  for K:=N-J to P do
    A[J,K]:=A[J-1,K]+A[J,K-1]
```

- A loop nest is qualified for polyhedral optimization, if:
  - The borders of iteration space are linear inequality constraints on control variables
    - Control variables are normalized to have step = +1
  - All memory accesses are indexed by linear combinations of control variables
    - If the same array is accessed more than once, the multiplicative constants must be identical

- Determining cross-iteration dependencies
  - Each write-read, read-write, or write-write pair for the same array must be examined
    - The difference of indices determines the cases of dependency
      - A[J1,K1] === A[J2-1,K2] implies <J2,K2> - <J1,K1> = <1, 0>
      - The vector <1, 0> indicates the direction of the dependency in the iteration space
      - The other pair A[J1,K1] === A[J2,K2-1] in this example produces <0, 1>
    - The vectors are always oriented so that their leftmost nonzero element is positive
      - Because the orientation of the dependency is determined by the original order of iterations
    - The convex hull of dependency vectors determines transitively dependent iterations
- Optimizing for fine-grained parallelization (vectorization, ILP)
  - In the innermost loop, use an iteration direction outside the convex hull of dependencies
    - May require a linear combination of the original control variables
    - It requires the transformation of iteration boundaries (for-loop boundary expressions)
      - More complex cases: Divide the iteration space into simpler geometrical shapes



A more general example




```
for J:=1 to N do
  for K:=N-J to P do
    A[J,K]:=A[J-1,K]+A[J,K-1]
```