Performance Engineering
Case study: “Jacobi” stencil

The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores

Prof. Dr. G. Wellein[a,b] , Dr. G. Hager[a]
[a] Erlangen National High Performance Computing Center (NHR@FAU)
[b] Department für Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg, Sommersemester 2026

Stencil schemes

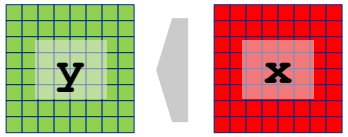
- Stencil schemes frequently occur in PDE solvers on regular lattice structures
- Basically it is a sparse matrix vector multiply (spMVM) embedded in an iterative scheme (outer loop)
- but the regular access structure allows for matrix free coding

```
do iter = 1, maxit

   Perform sweep over regular grid: y ← x

   Swap y ←→ x

enddo
```



- Complexity of implementation and performance depends on
  - update scheme, e.g. Jacobi-type, Gauss-Seidel-type,…
  - spatial (stencil) extent, e.g. 7-pt or 25-pt in 3D,…
  - …

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Case study: Jacobi stencil

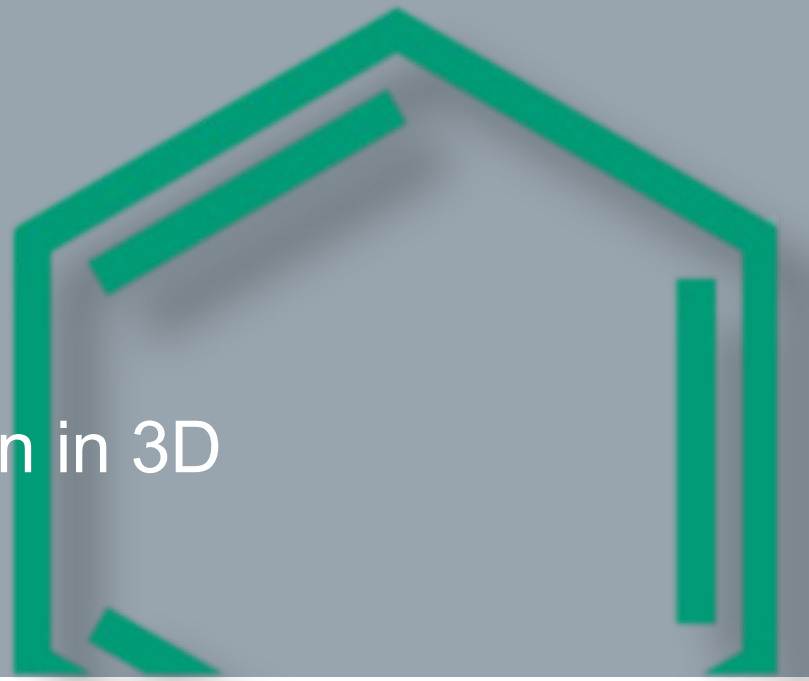
The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

OpenMP parallelization strategies and layer condition in 3D

NT stores



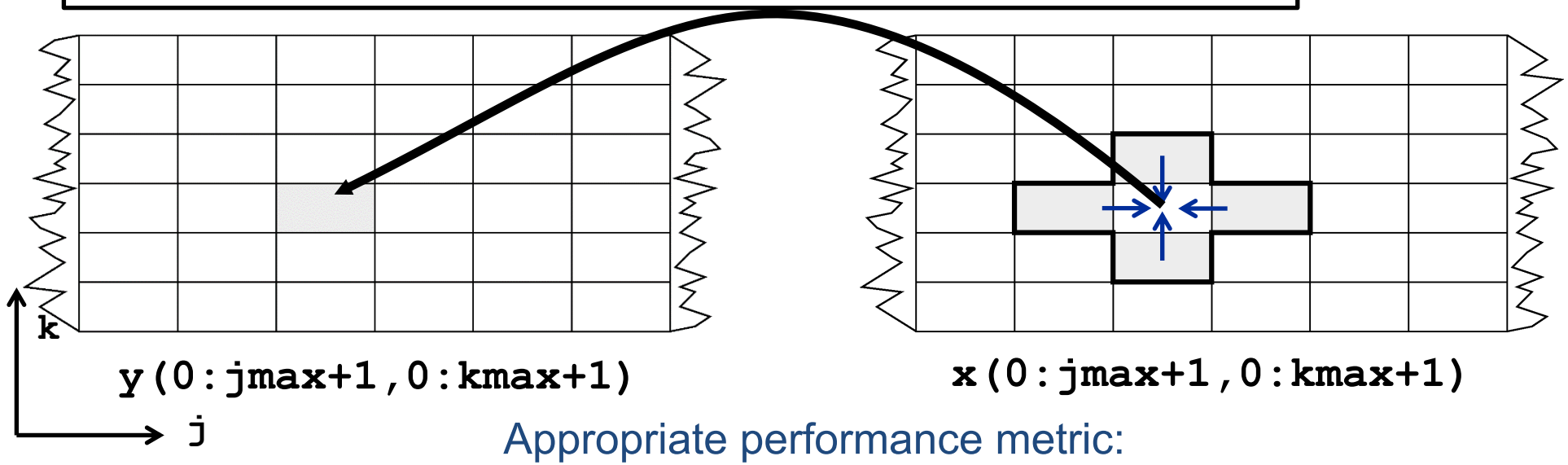
Jacobi-type 5-pt stencil in 2D

sweep

```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (  x(j-1,k) + x(j+1,k) &
                      +    x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

Lattice site update (LUP)

2 arrays

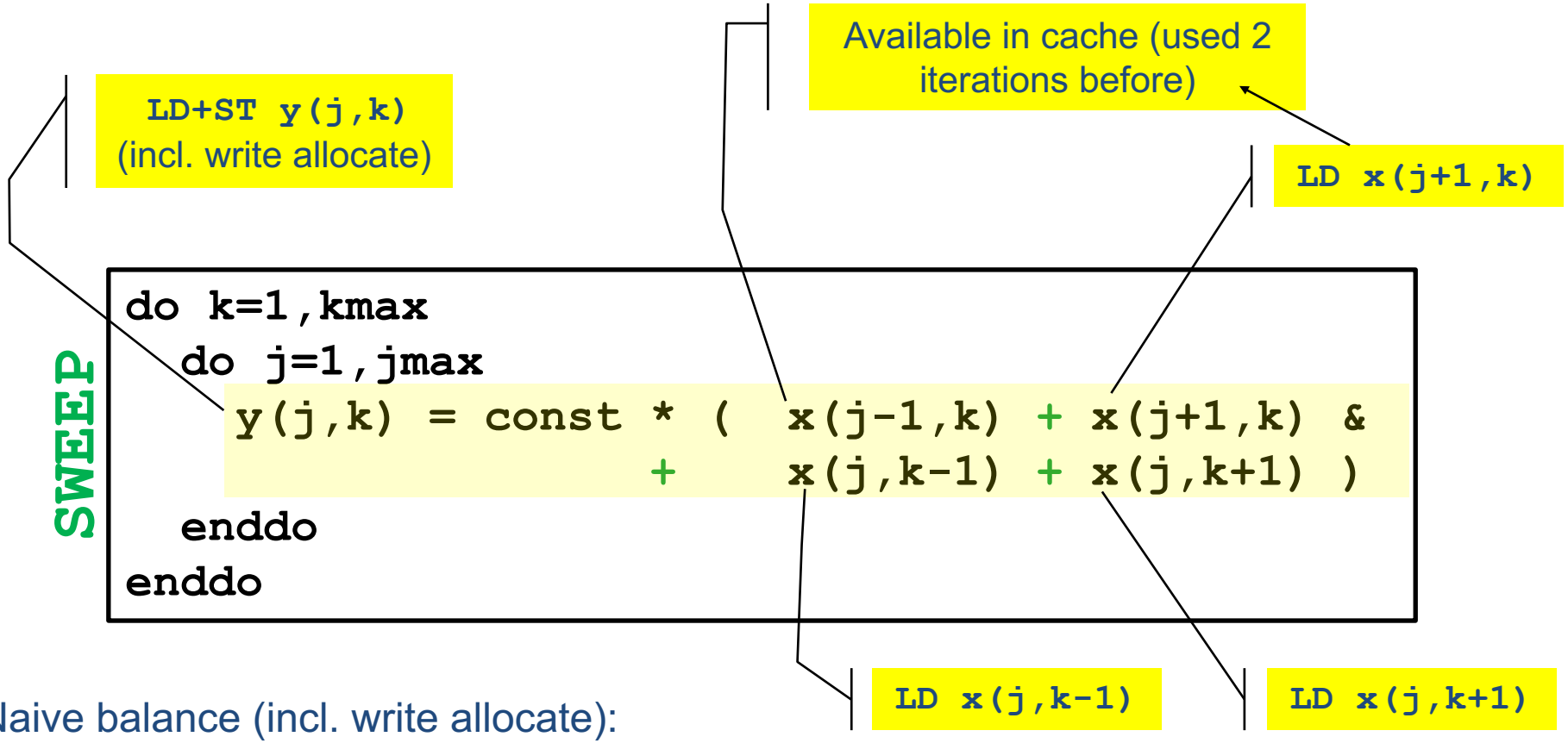


Appropriate performance metric:

"Lattice site updates per second" [LUP/s]

(here: Multiply by 4 FLOP/LUP to get FLOP/s rate)

Jacobi 5-pt stencil in 2D: data transfer analysis

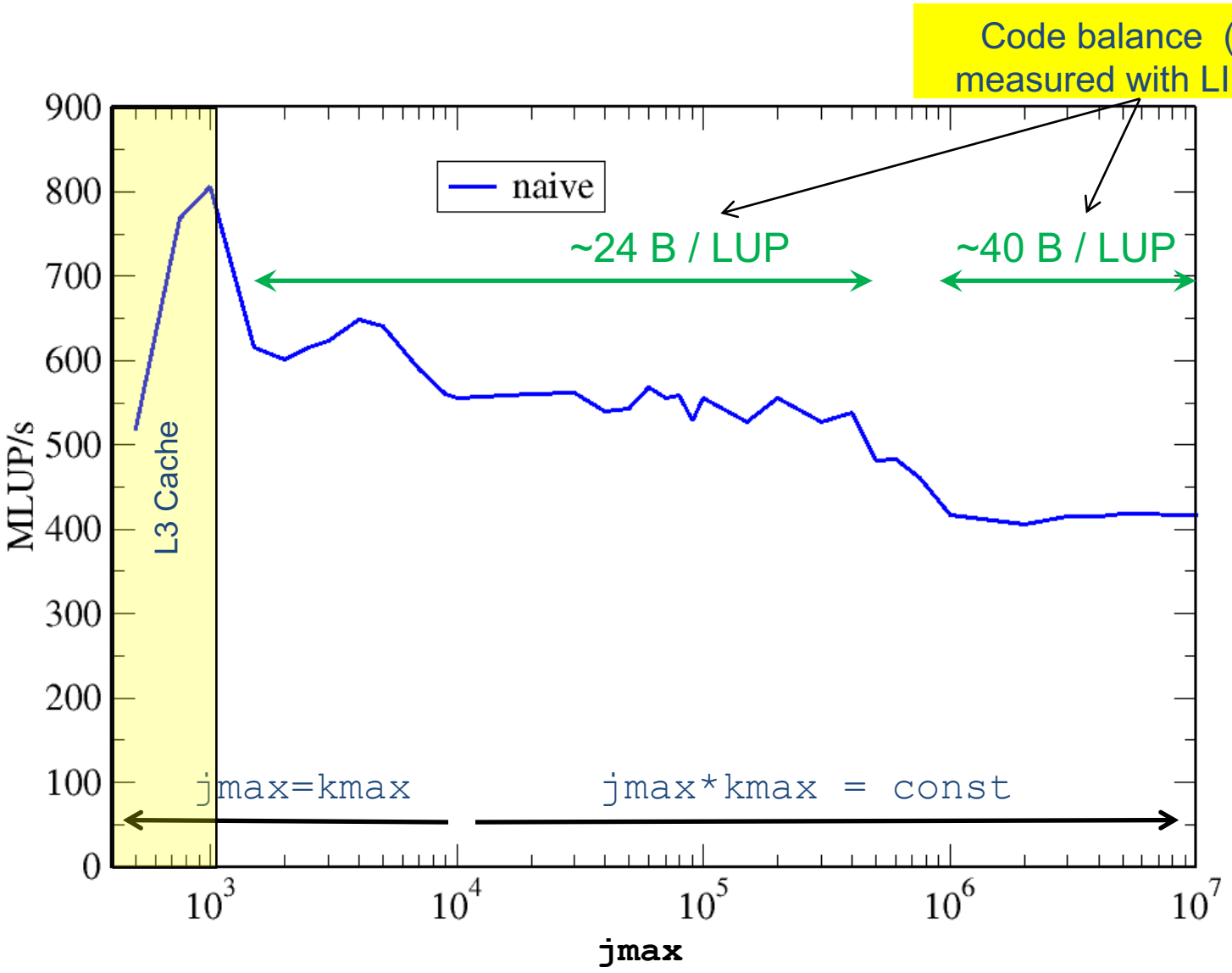


Naive balance (incl. write allocate):

`x( :, :)` : 3 LD +
`y( :, :)` : 1 ST+ 1LD

→ B_C = 5 Words / LUP → B_C = 40 B / LUP (assuming double precision)

Jacobi 5-pt stencil in 2D: Single core performance



Questions:

1. How to achieve 24 B/LUP also for large **`jmax`**?
2. How to sustain >600 MLUP/s for **`jmax > 10⁴`** ?
3. Why 24 B/LUP anyway???

Intel Compiler: ifort V13.1

Intel Xeon E5-2690 v2 ("IvyBridge"@3 GHz)

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

Case study: Jacobi stencil

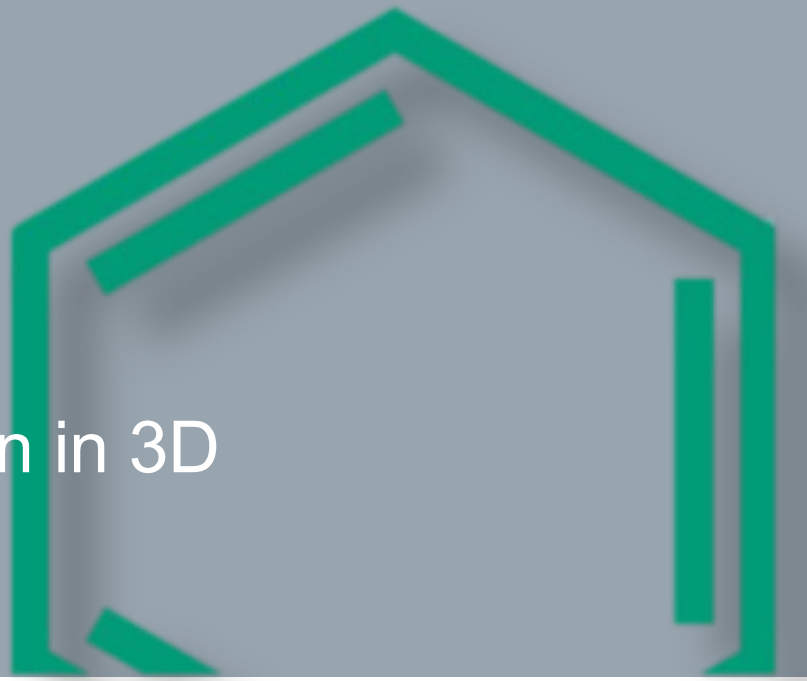
The basics in two dimensions (2D)

Layer condition in 2D

From 2D to 3D

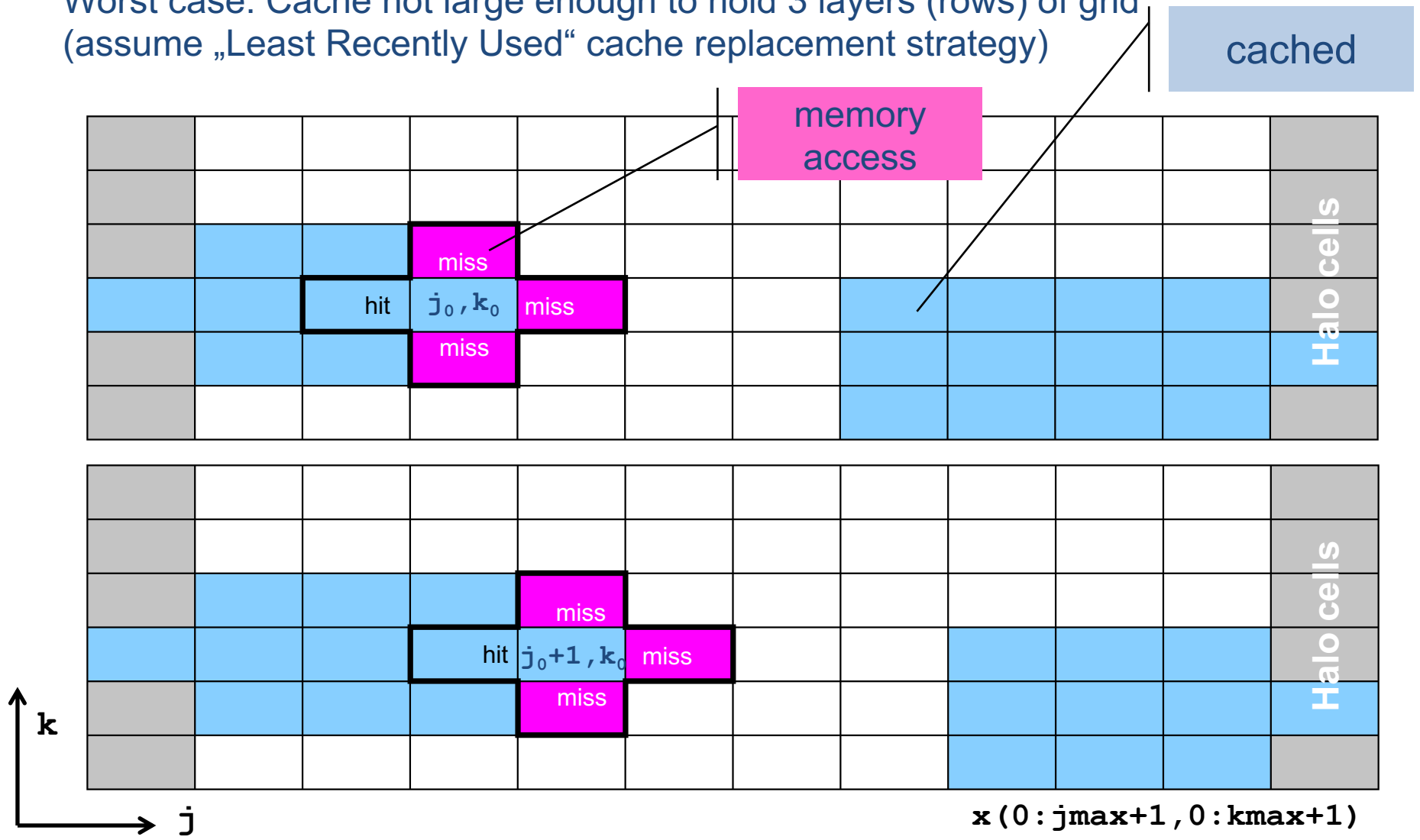
OpenMP parallelization strategies and layer condition in 3D

NT stores



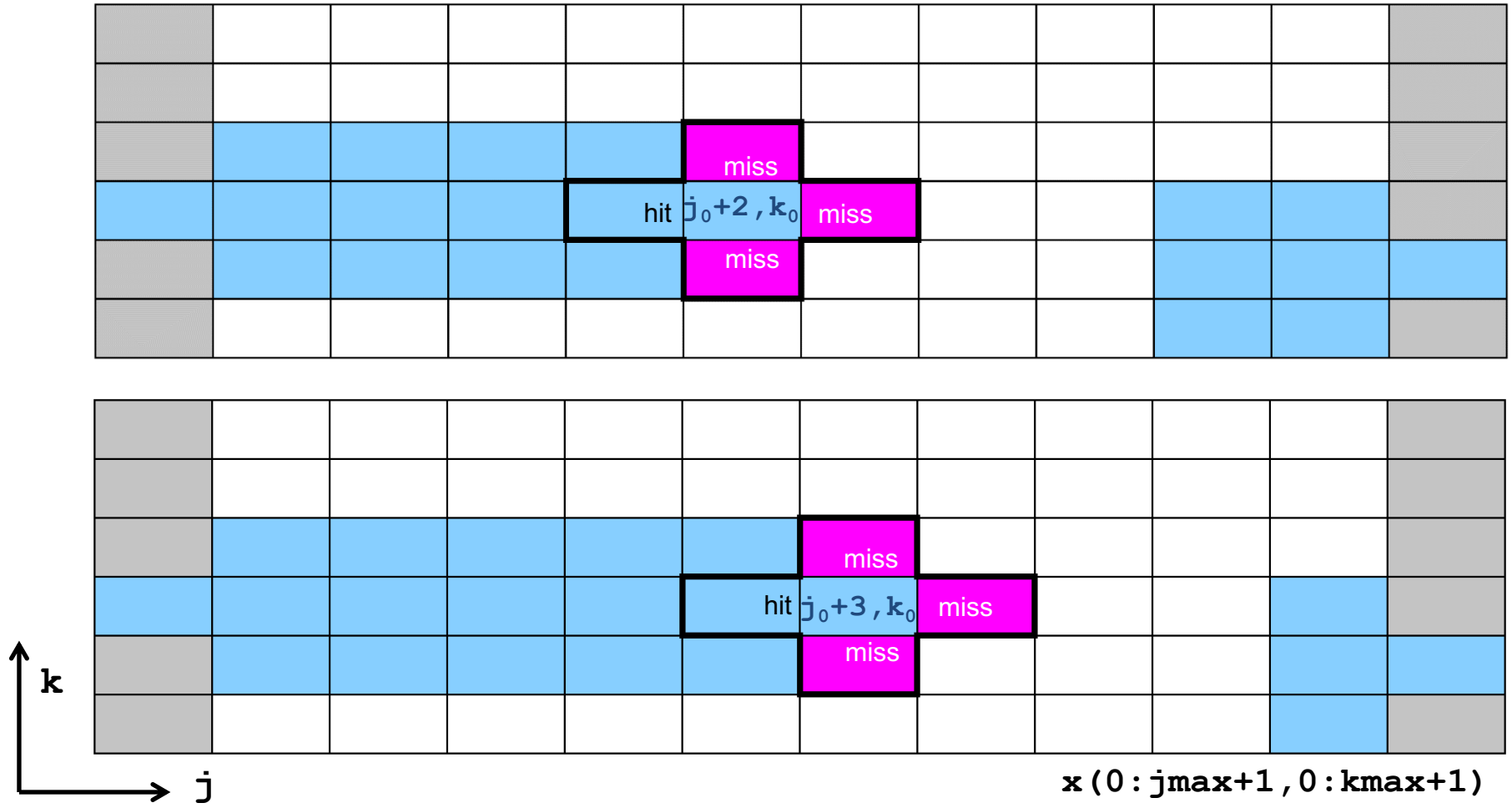
Analyzing the data flow

Worst case: Cache not large enough to hold 3 layers (rows) of grid
(assume „Least Recently Used“ cache replacement strategy)



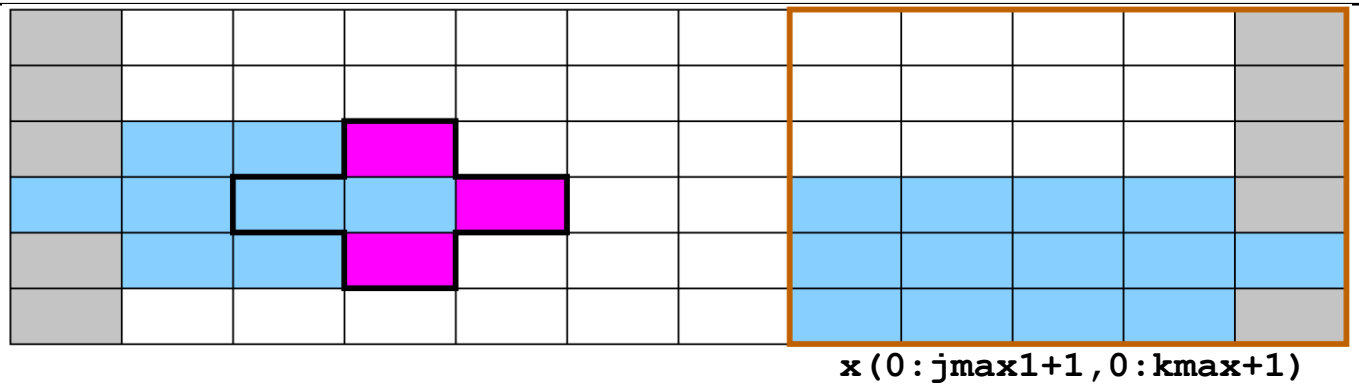
Analyzing the data flow

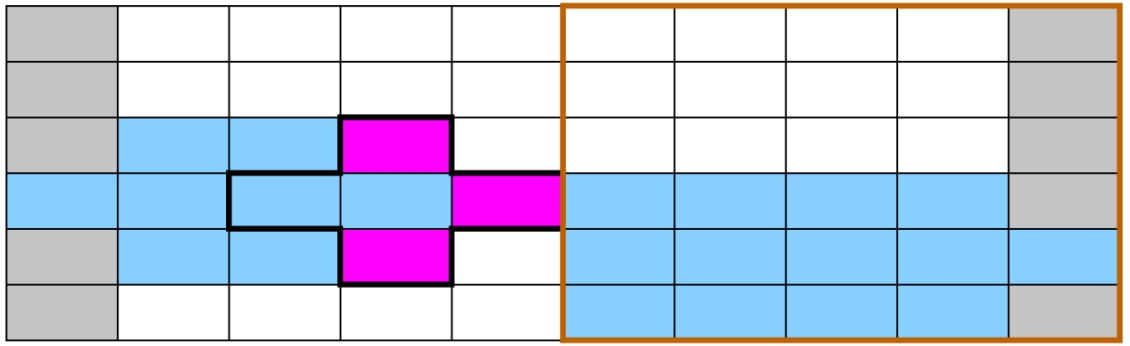
Worst case: Cache not large enough to hold 3 layers (rows) of grid
(+assume „Least Recently Used“ replacement strategy)



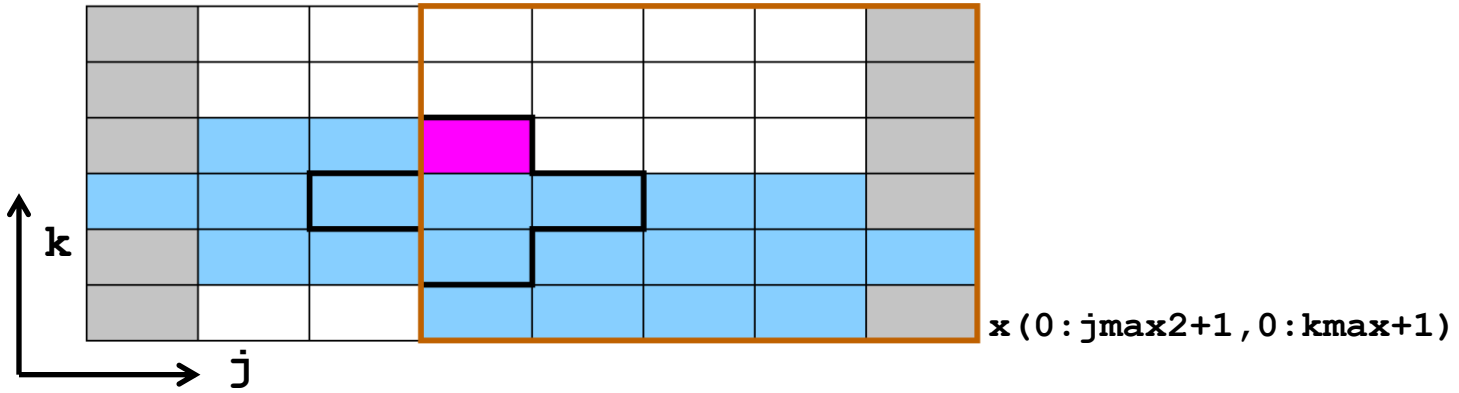
Analyzing the data flow

Reduce inner (j-) loop dimension successively



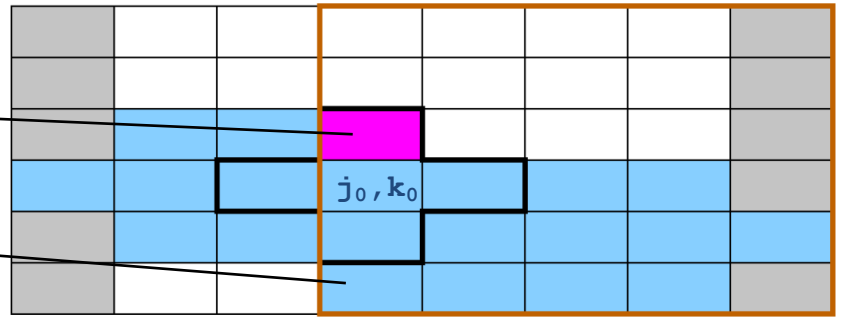


Best case: 3 „layers“ of grid fit into the cache!



Analyzing the data flow: Layer condition

2D 5-pt Jacobi-type stencil




```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (x(j-1,k) + x(j+1,k) &
                   +   x(j,k-1) + x(j,k+1) )
  enddo
enddo
```

hit

3*jmax * 8B < CacheSize/2

"Layer condition"

3 rows of **jmax**

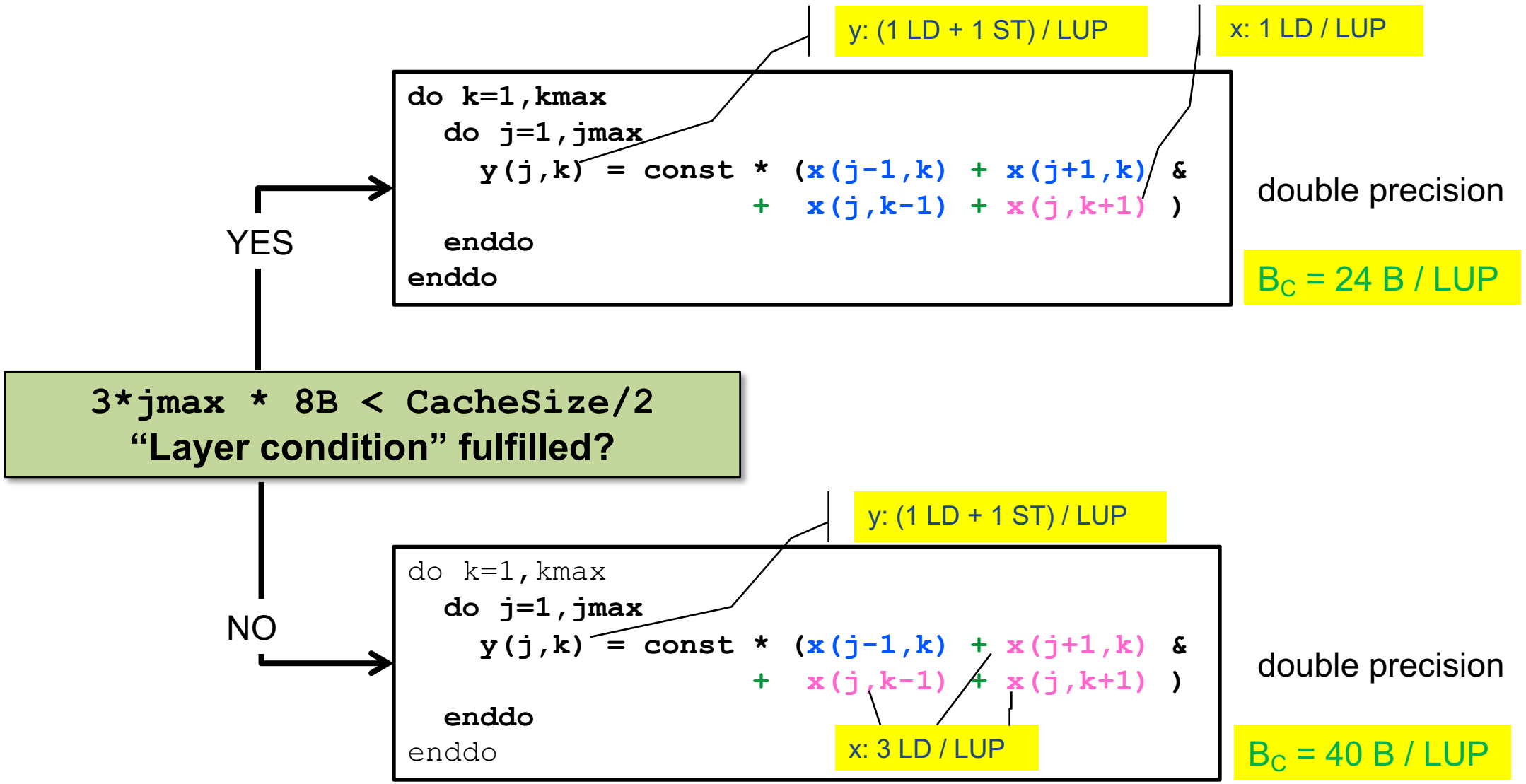
double precision

Safety margin (Rule of thumb)

Layer condition:

- No impact of outer loop length (**kmax**)
- No strict guideline (cache associativity – data traffic for **y()** not included)
- Needs to be adapted for other stencils, e.g., in 3D, long-range, multi-array,…

Analyzing the data flow: Layer condition (2D 5-pt Jacobi)



Analyzing the data flow: Layer condition (2D 5-pt Jacobi)

- How to establish the layer condition for all domain sizes ?
- Idea: Spatial blocking
  - Reuse elements of `x()` as long as they stay in cache
  - Sweep can be executed in any order, e.g., compute blocks in j-direction

→ "Spatial Blocking" of j-loop:

```
do jb=1,jmax,jblock !      Assume jmax is multiple of jblock
  do k=1,kmax
    do j= jb, (jb+jblock-1) ! Length of inner loop: jblock
      y(j,k) = const * (x(j-1,k) + x(j+1,k) &
                      +  x(j,k-1) + x(j,k+1) )
    enddo
  enddo
enddo
```

New layer condition (blocking)
`3* jblock * 8B < CacheSize/2`

→ Determine for given `CacheSize` an appropriate `jblock` value:

`jblock < CacheSize / 48 B`