

Case study: A Jacobi smoother

The basics in two dimensions

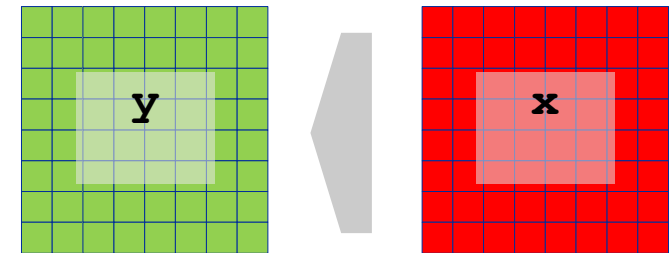


Stencil schemes

- Stencil schemes frequently occur in PDE solvers on regular lattice structures
- Basically 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, max_iterations  
    Perform sweep over regular grid:  $y(:) \leftarrow x(:)$   
    Swap  $y \leftrightarrow x$   
enddo
```

- Complexity of implementation and performance depends on
 - stencil operator, e.g. Jacobi-type, Gauss-Seidel-type, ...
 - discretization, e.g. 7-pt or 27-pt in 3D,...

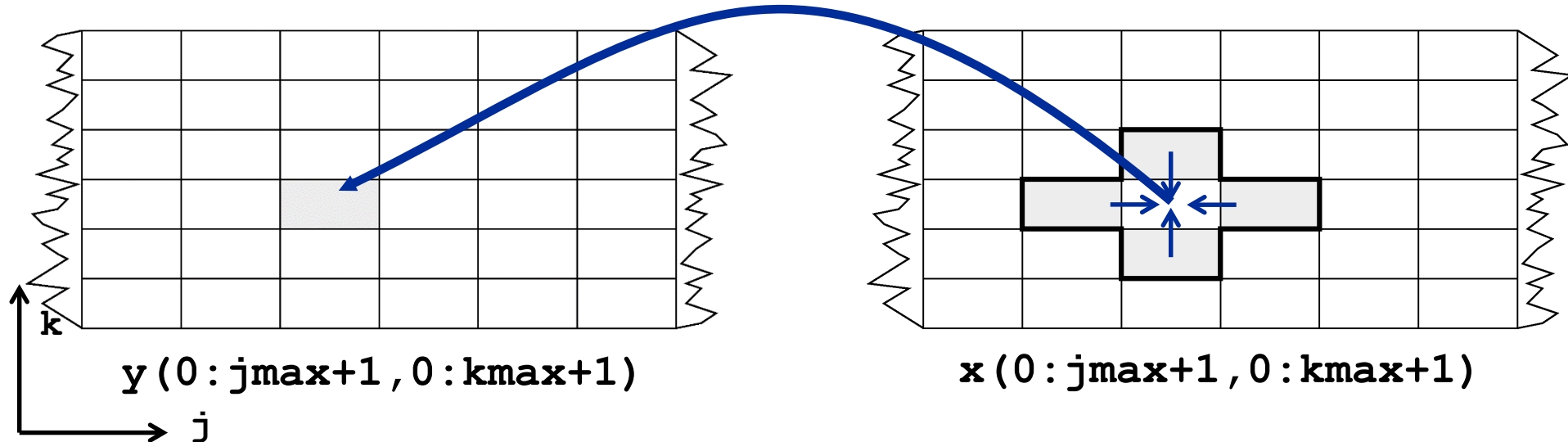


Jacobi-type 5-pt stencil in 2D

```
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
```

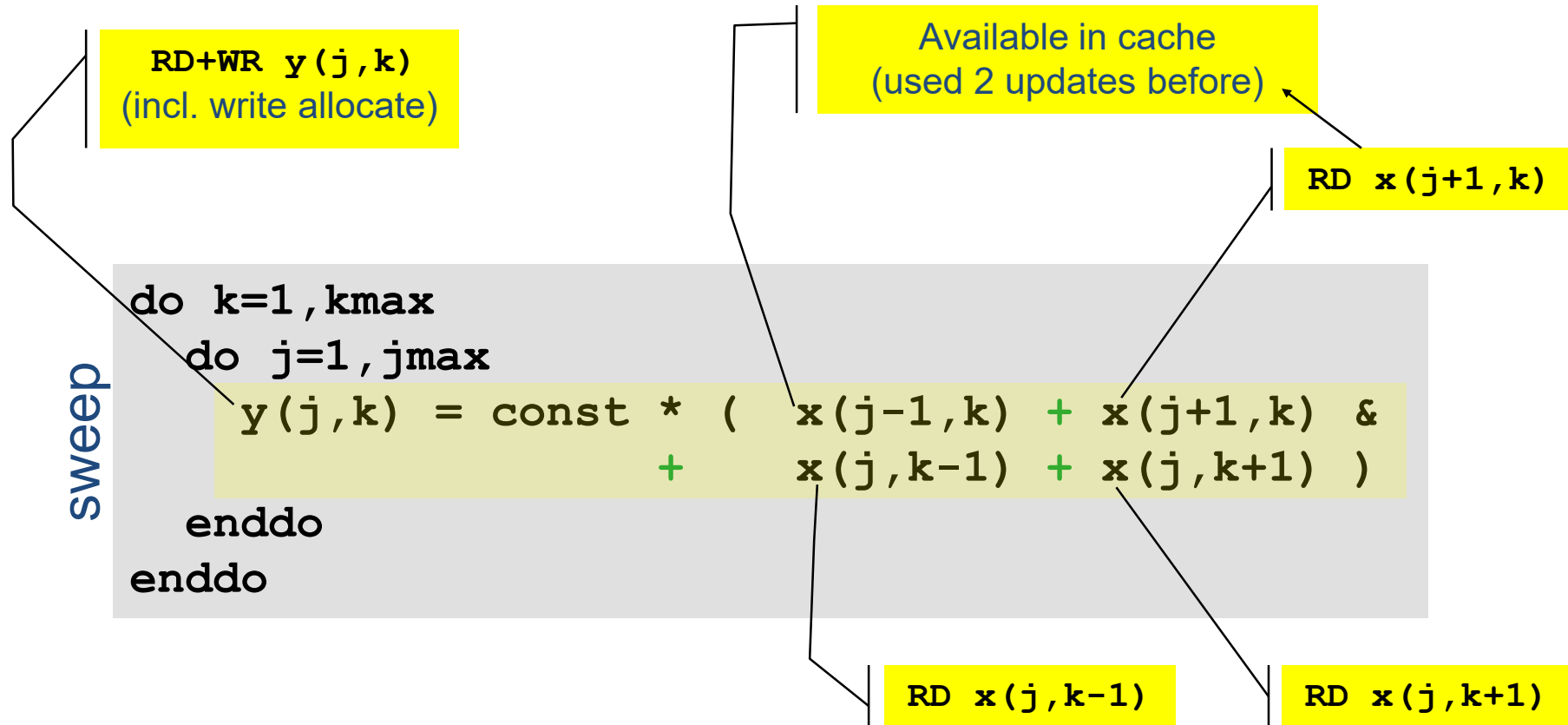
sweep

Lattice site update (LUP)



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 2D: data transfer analysis



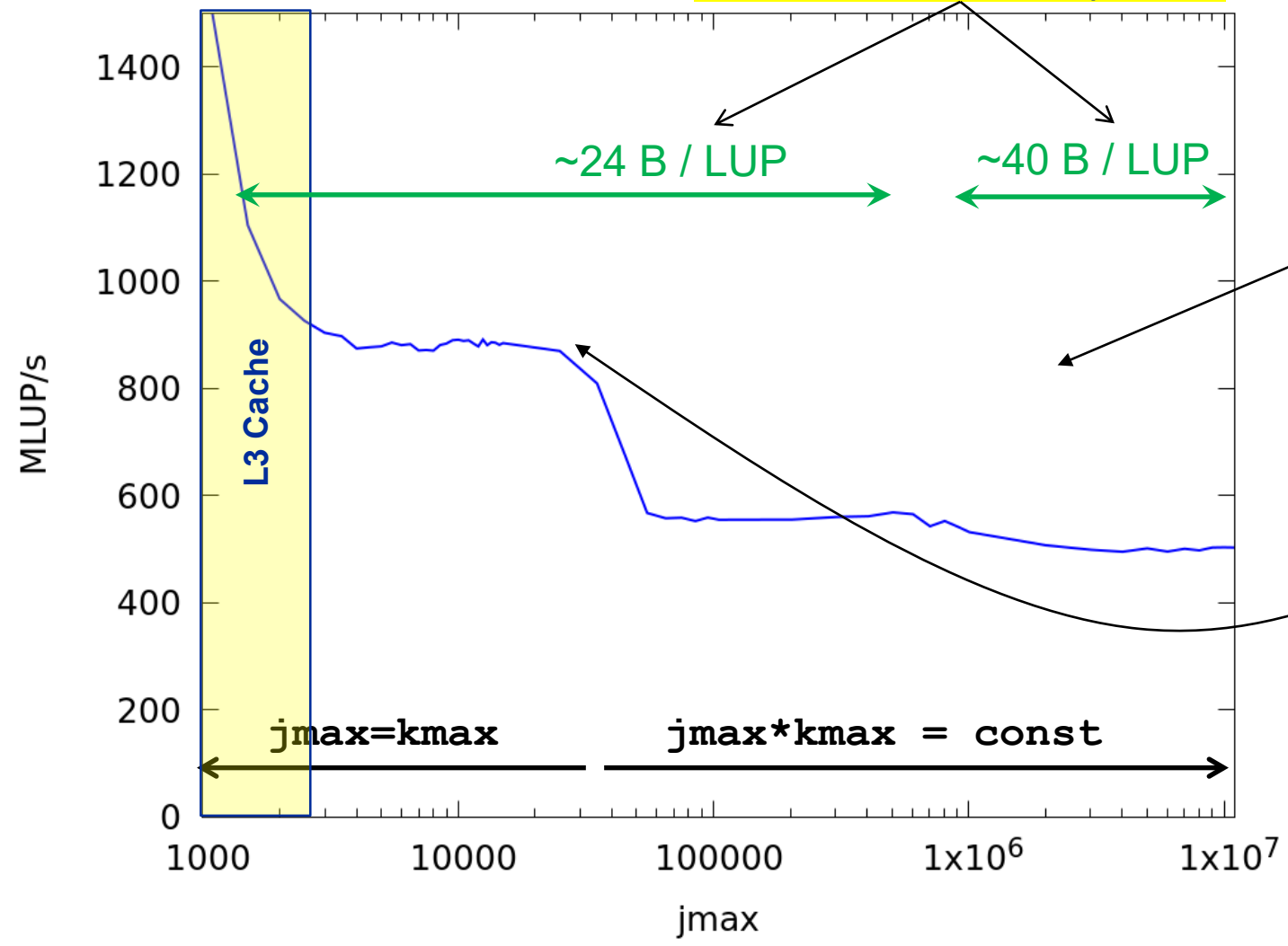
Naive balance (incl. write allocate):

$\mathbf{x}(\cdot, \cdot)$: 3 RD +
 $\mathbf{y}(\cdot, \cdot)$: 1 WR+ 1 RD

→ $B_C = 5 \text{ Words} / \text{LUP} = 40 \text{ B} / \text{LUP}$ (assuming double precision)

Jacobi 5-pt stencil 2D: Single core performance

Code balance (B_C)
measured with likwid-perfctr



Questions:

1. How to achieve 24 B/LUP also for large j_{max} ?
2. How to sustain >800 MLUP/s for $j_{max} > 10^4$?

Intel Compiler 2022.1.0
Intel Xeon Platinum 8360Y
("IcelakeSP"@2.4 GHz)

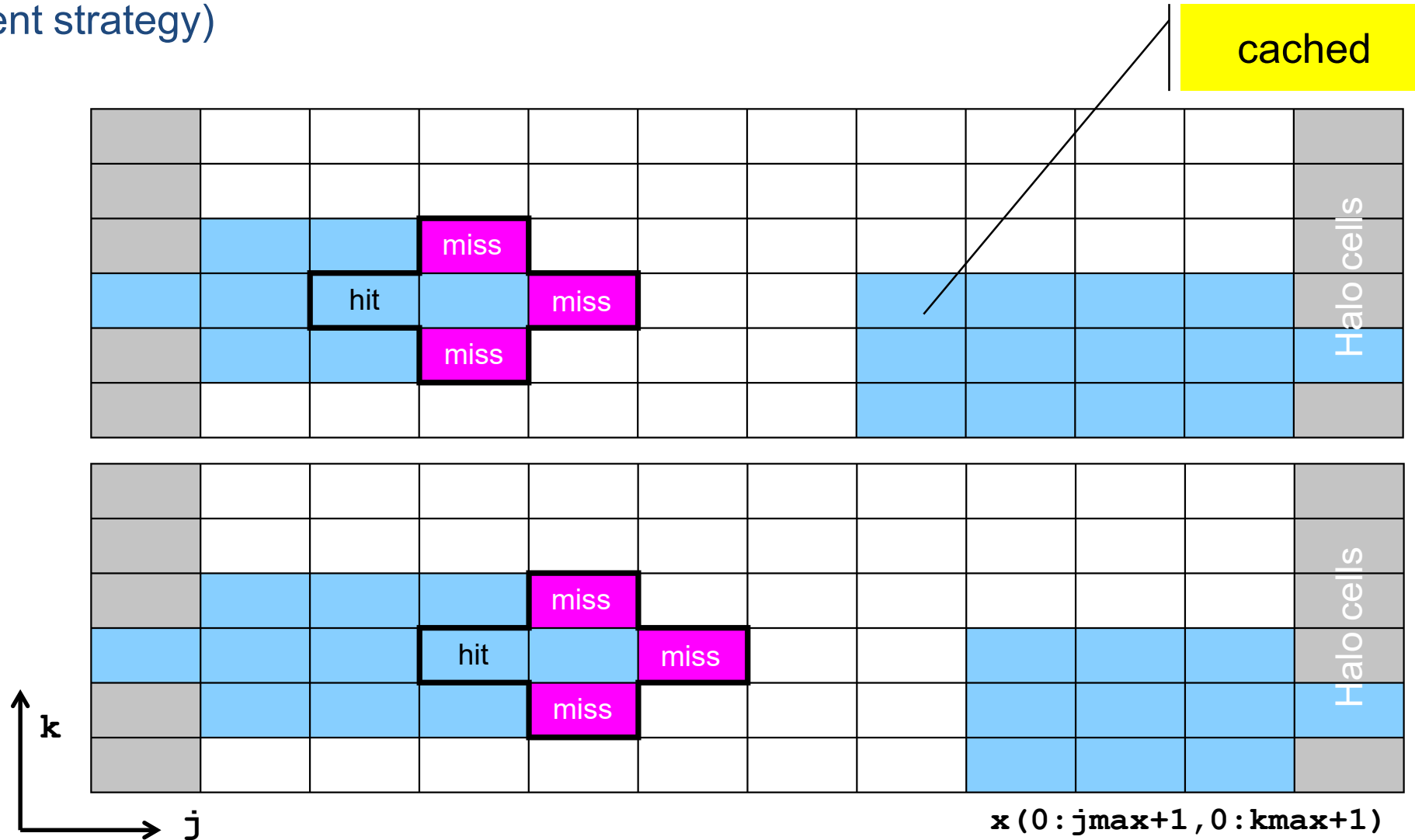
Case study: A Jacobi smoother

Layer conditions



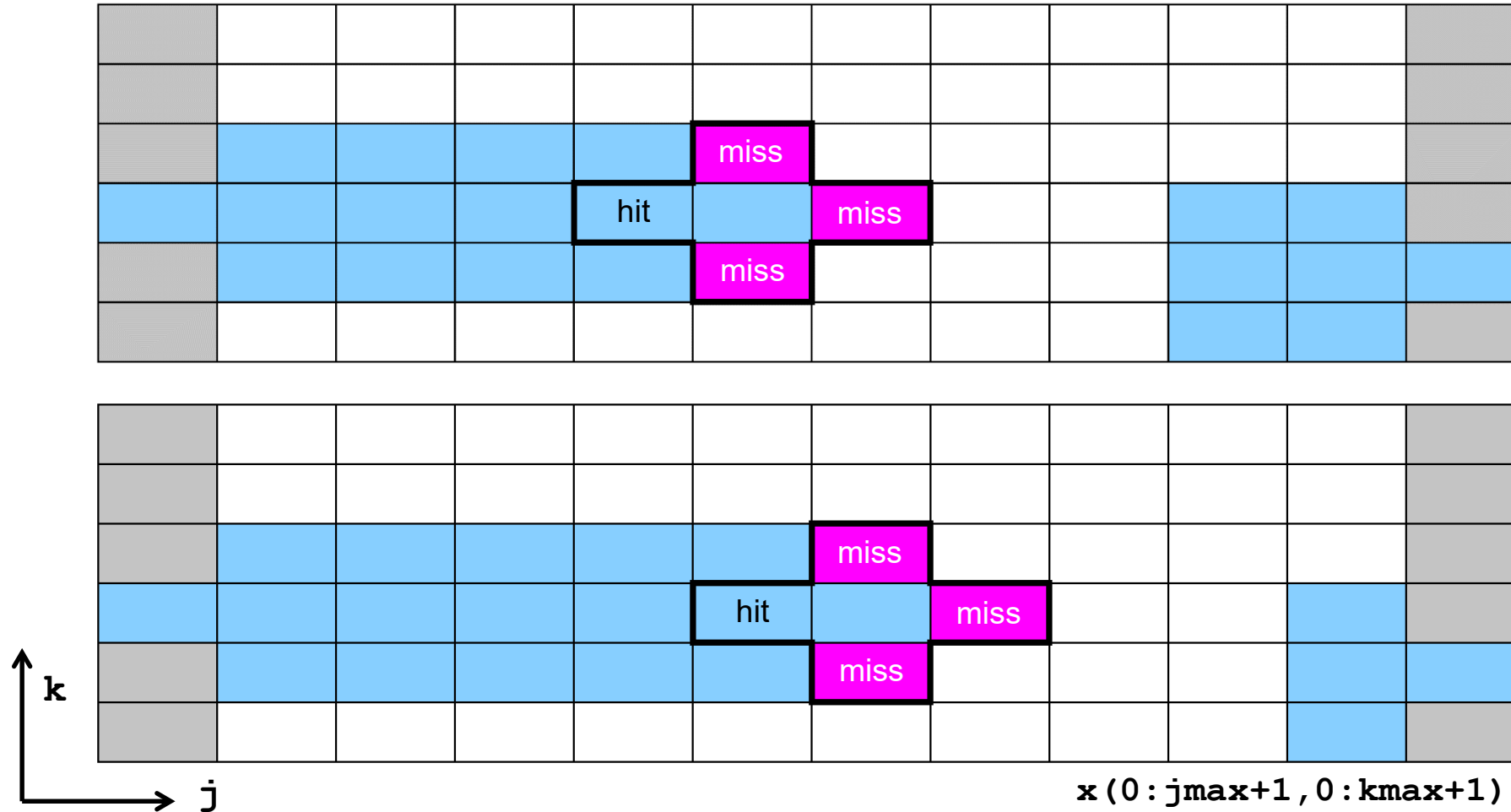
Analyzing the data flow

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



Analyzing the data flow

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

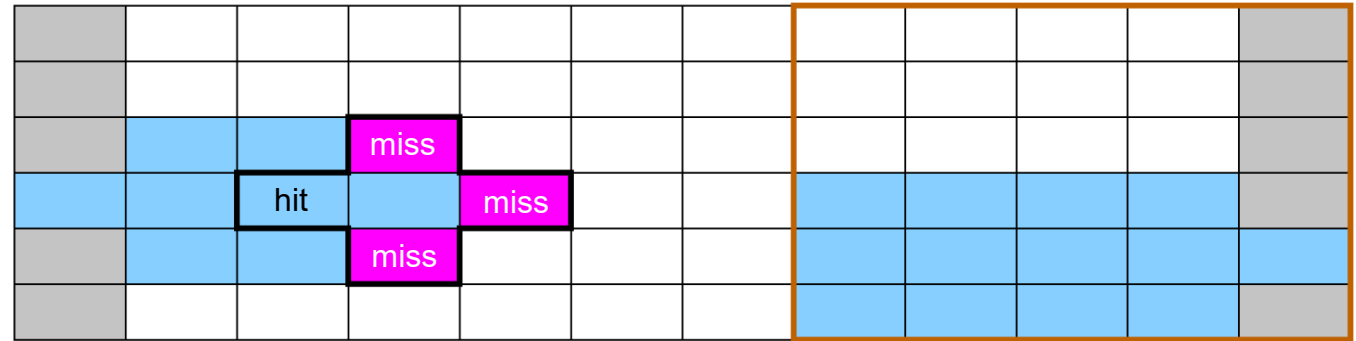
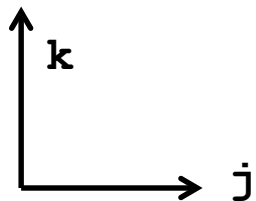


Analyzing the data flow

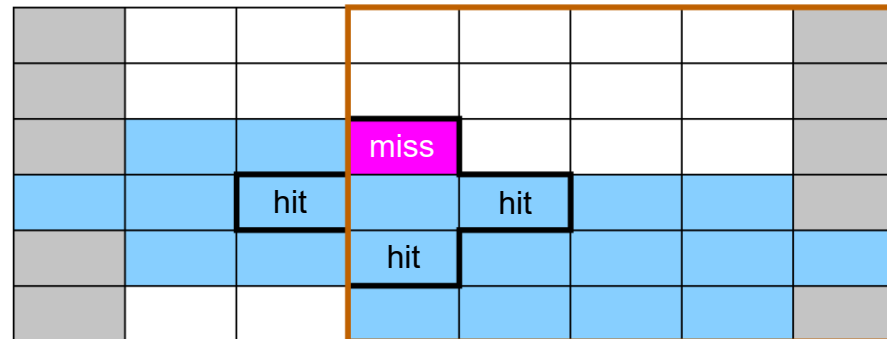
Reduce inner (j-) loop dimension successively



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



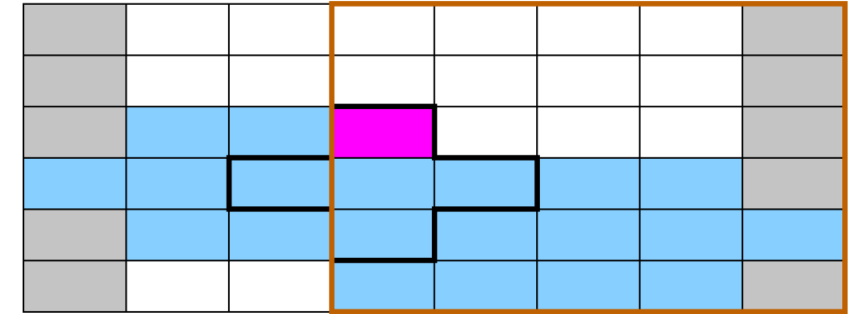
$x(0:j_{\max}1+1, 0:k_{\max}+1)$



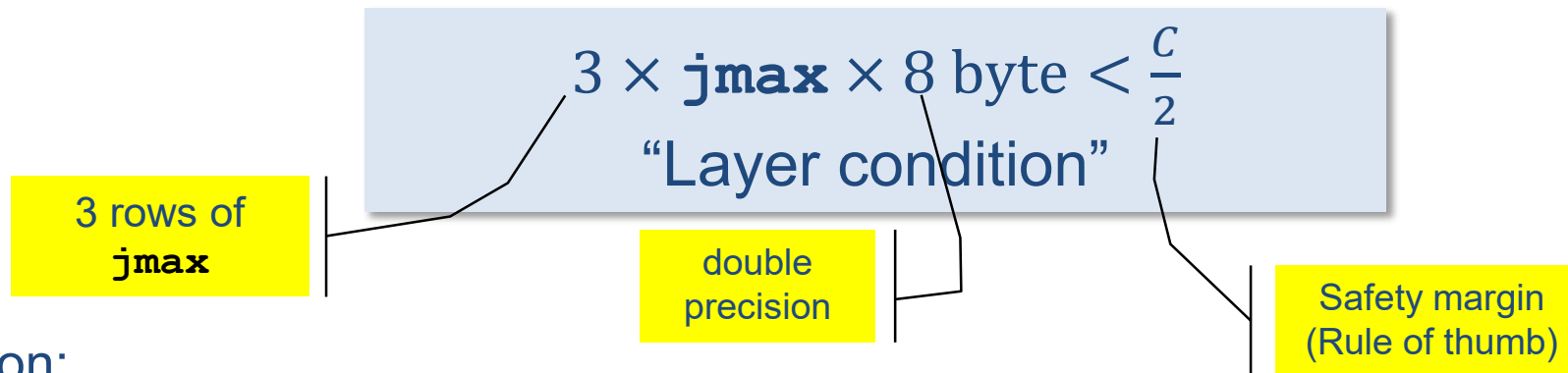
$x(0:j_{\max}2+1, 0:k_{\max}+1)$

Analyzing the data flow: Layer condition

2D 5-pt Jacobi-type stencil, cache size C



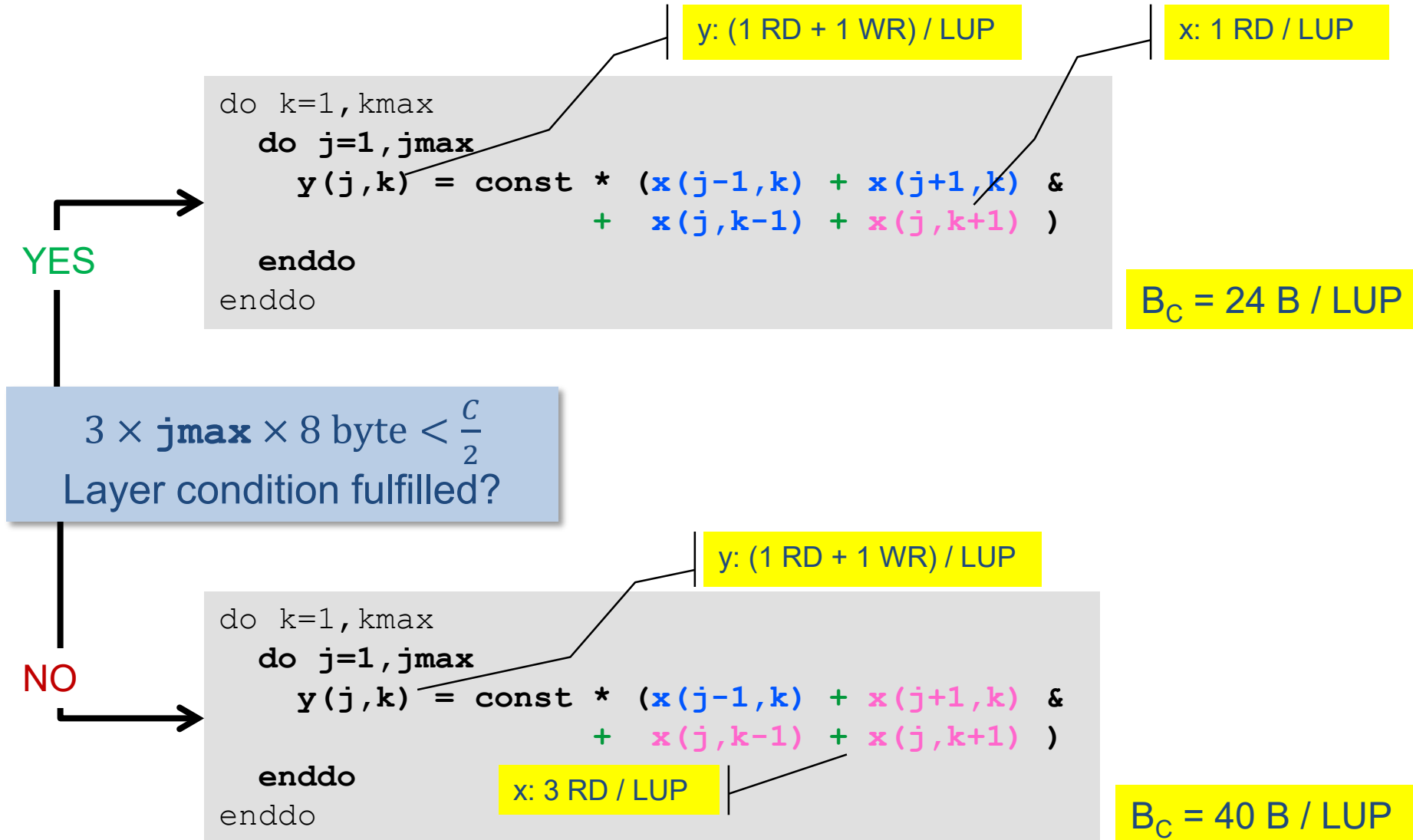
```
do k=1,kmax
  do j=1,jmax
    y(j,k) = const * (x(j-1,k) + x(j+1,k) &
                     + x(j,k-1) + x(j,k+1) )
  enddo
enddo
```



Layer condition:

- Does not depend on outer loop length (k_{\max})
- No strict guideline (cache associativity, data traffic for y not included)
- Needs to be adapted for other stencils (e.g., long-range stencils)

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



Case study: A Jacobi smoother

Optimization by spatial blocking



Enforcing a layer condition (2D 5-pt Jacobi)

- How can we enforce a layer condition for all domain sizes ?
- Idea: **Spatial blocking**
 - Reuse elements of $\mathbf{x}()$ 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, jbblock !
  do k=1, kmax
    do j= jb, min(jb+jbblock-1, jmax) !inner loop length jbblock
      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)

$$3 \times \mathbf{jb}block \times 8 \text{ byte} < \frac{C}{2}$$

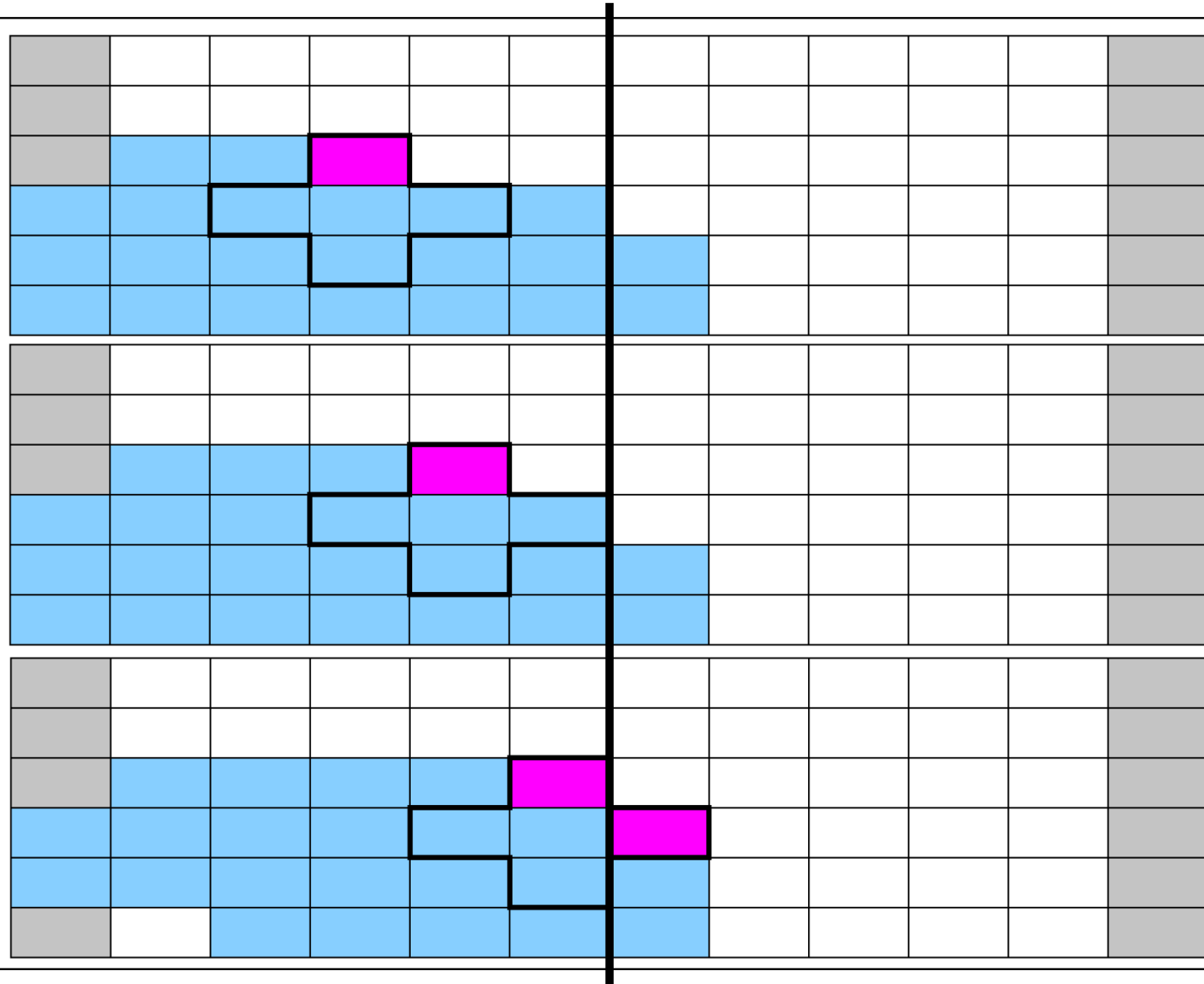
Determine for given C an appropriate **jb**block value:

$$\mathbf{jb}block < \frac{C}{48 \text{ byte}}$$

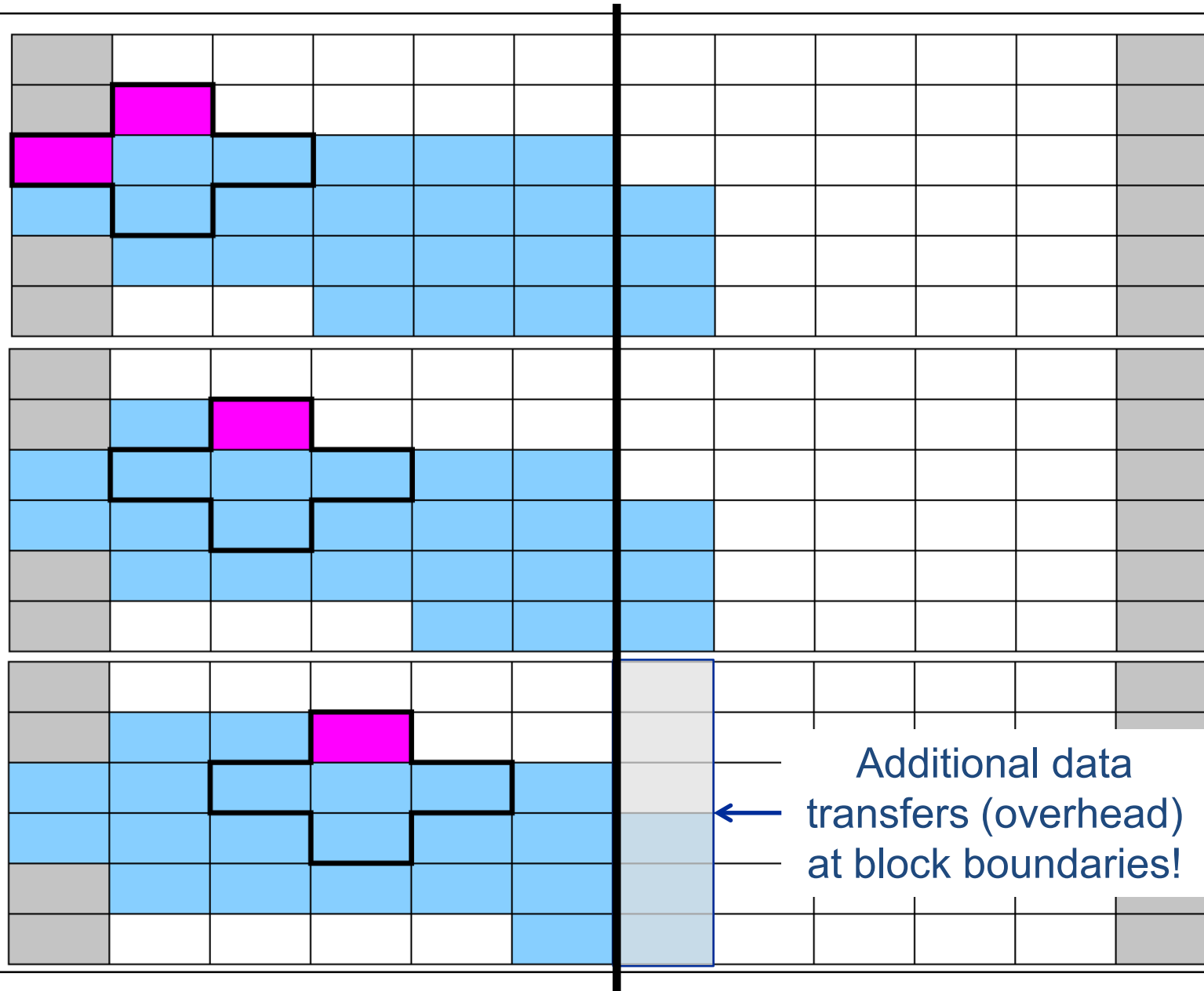
Establish the layer condition by blocking

Split domain into subblocks:

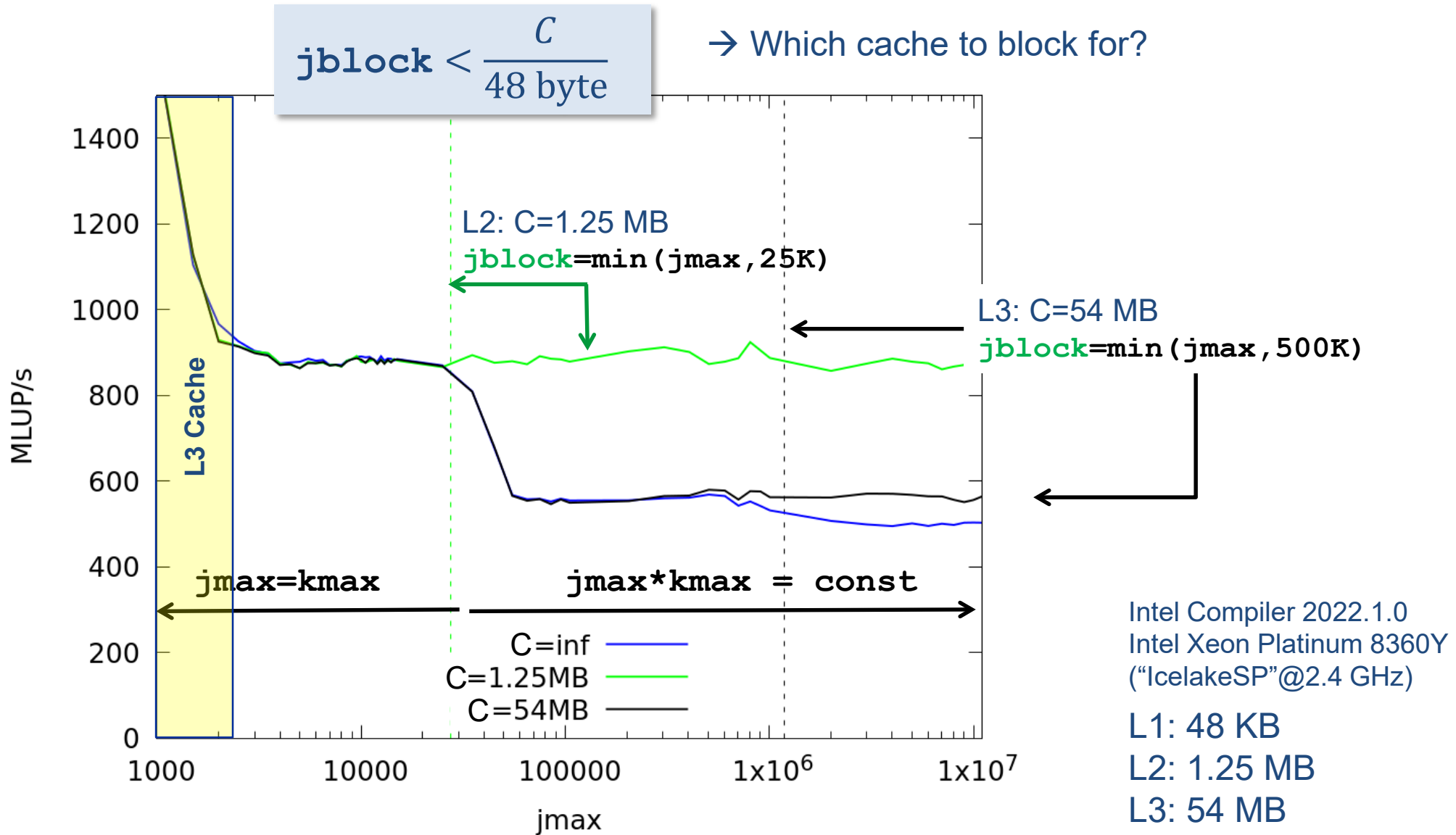
e.g. block size = 5



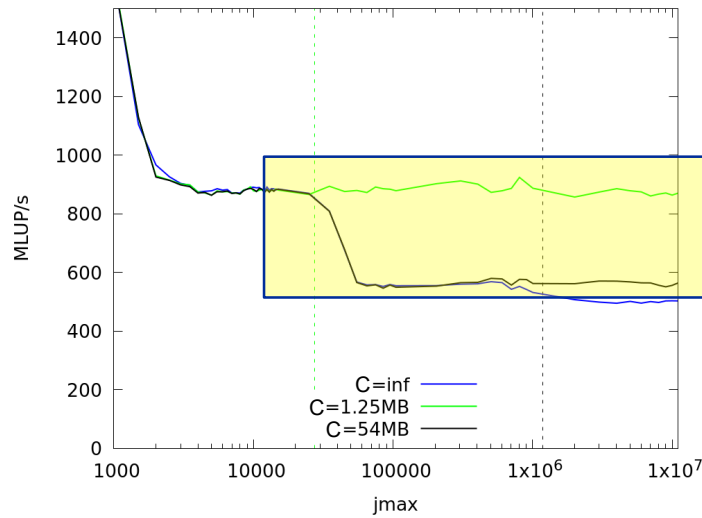
Establish the layer condition by blocking



Establish layer condition by spatial blocking

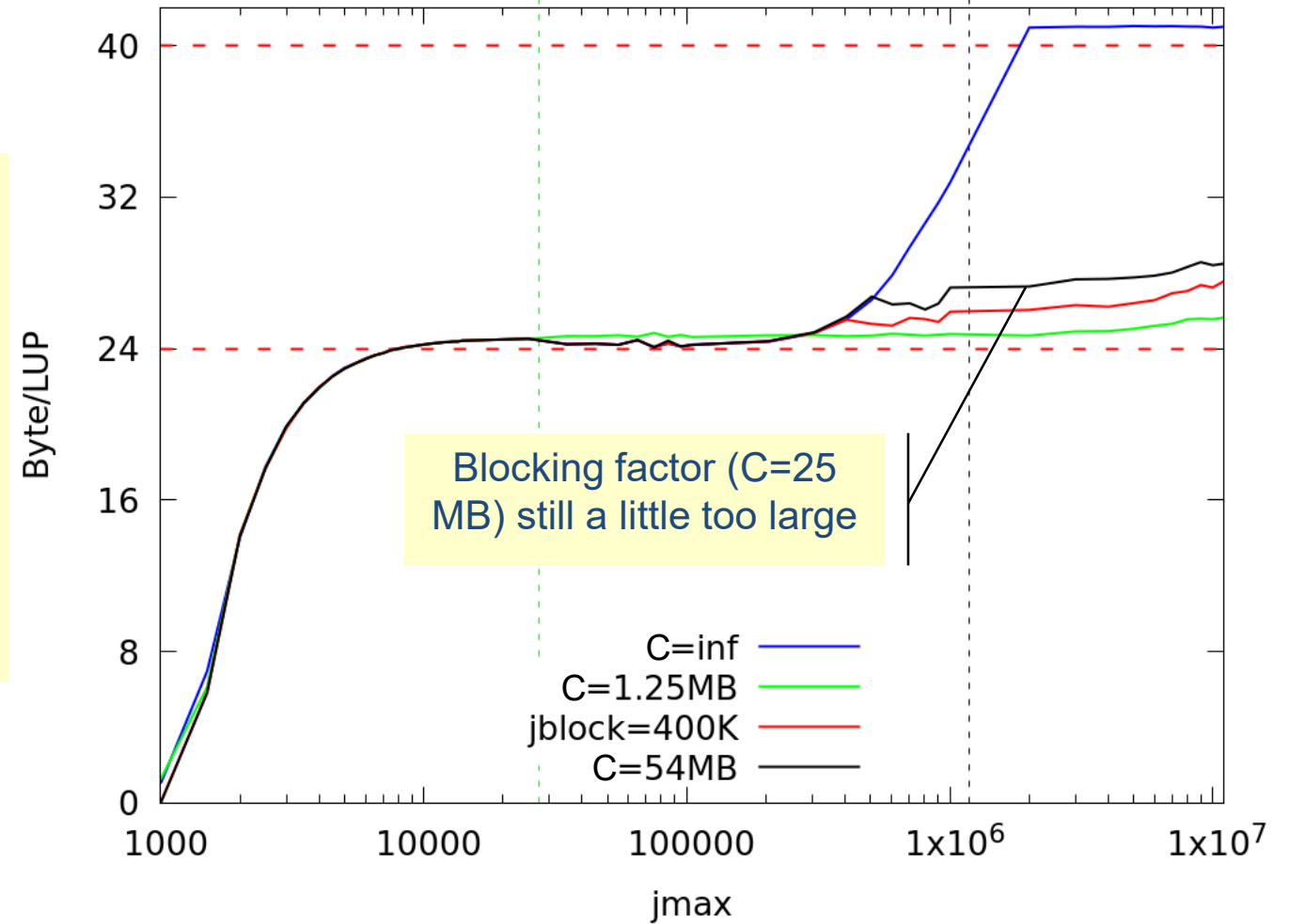


Validating the model: Memory code balance



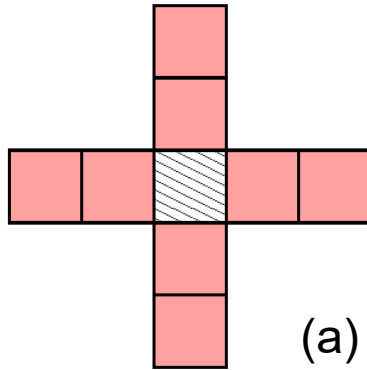
Main memory access is not reason for different performance (but L3 access is!)

Measured main memory code balance (BC)

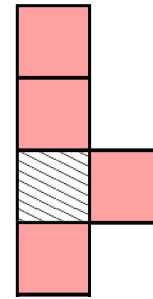


Intel Compiler 2022.1.0
Intel Xeon Platinum 8360Y
("IcelakeSP"@2.4 GHz)

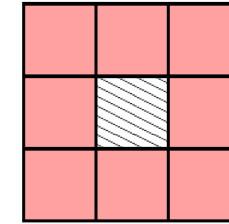
Stencil shapes and layer conditions in 2D



(a)



(b)



(c)

- a) Long-range $r = 2$: 5 layers ($2r + 1$)
- b) Asymmetric: 4 layers
- c) 2D box: 3 layers

Case study: A Jacobi smoother

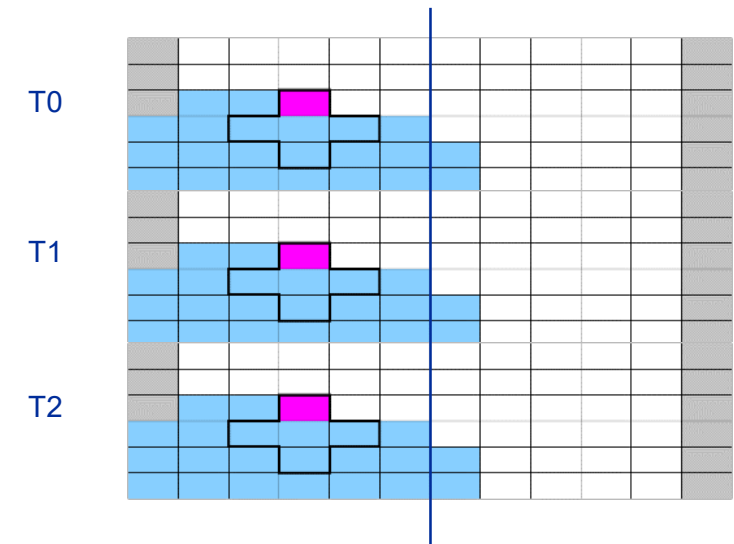
OpenMP parallelization



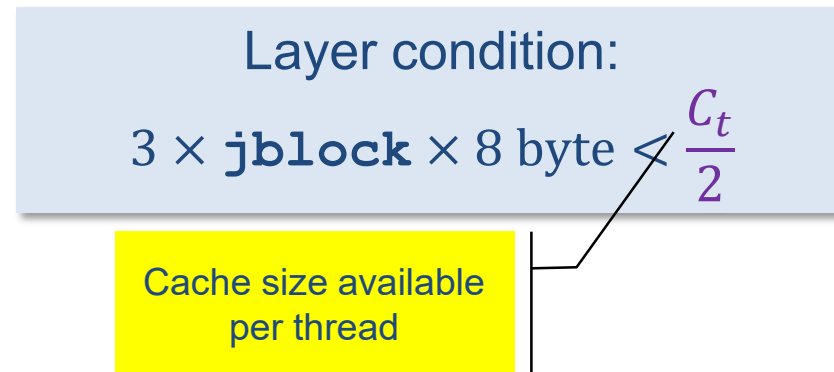
OpenMP parallelization of the blocked 2D stencil

Straightforward OpenMP work sharing:

```
do jb=1, jmax, jblock
!$OMP PARALLEL DO SCHEDULE(static)
  do k=1, kmax
    do j= jb, min(jb+jbblock-1, jmax)
      y(j,k) = const * (x(j-1,k) + x(j+1,k) &
        + x(j,k-1) + x(j,k+1) )
    enddo
  enddo
!$OMP END PARALLEL DO
enddo
```



Caveat: LC must be fulfilled **per thread** → shared cache causes smaller blocks!



OpenMP parallelization and blocking for a shared cache

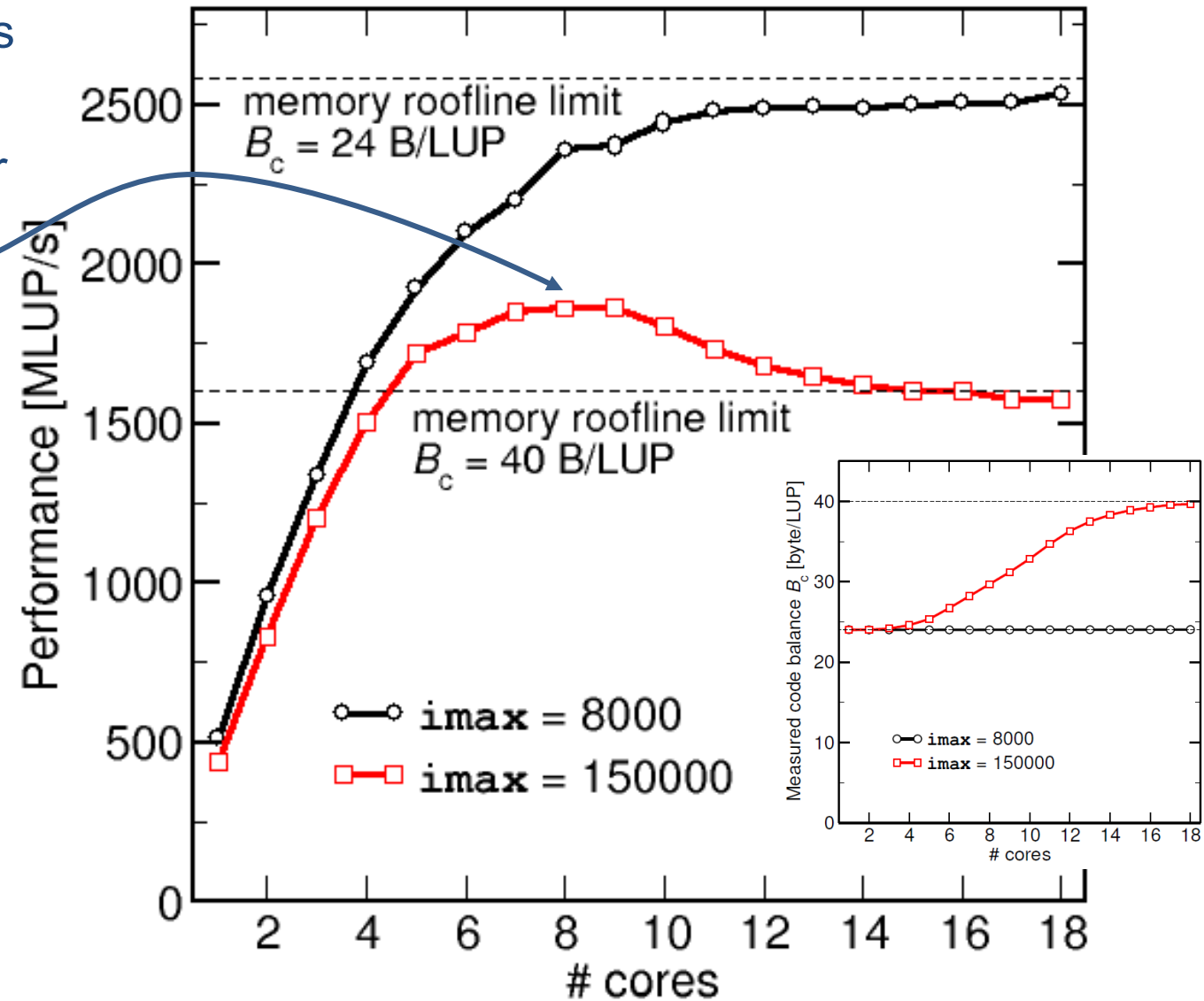
Layer conditions make for interesting effects

- Less and less shared cache available per thread as #threads goes up
- LC may break “along the way”

Solutions

1. Choose small enough block or domain size
 - Layers either small enough to fit in core-private caches *or*
 - Shared cache big enough to hold all layers for all threads
2. Adaptive blocking for shared cache:

$$\mathbf{jblock} = \frac{C}{\#threads \times 48 \text{ byte}}$$

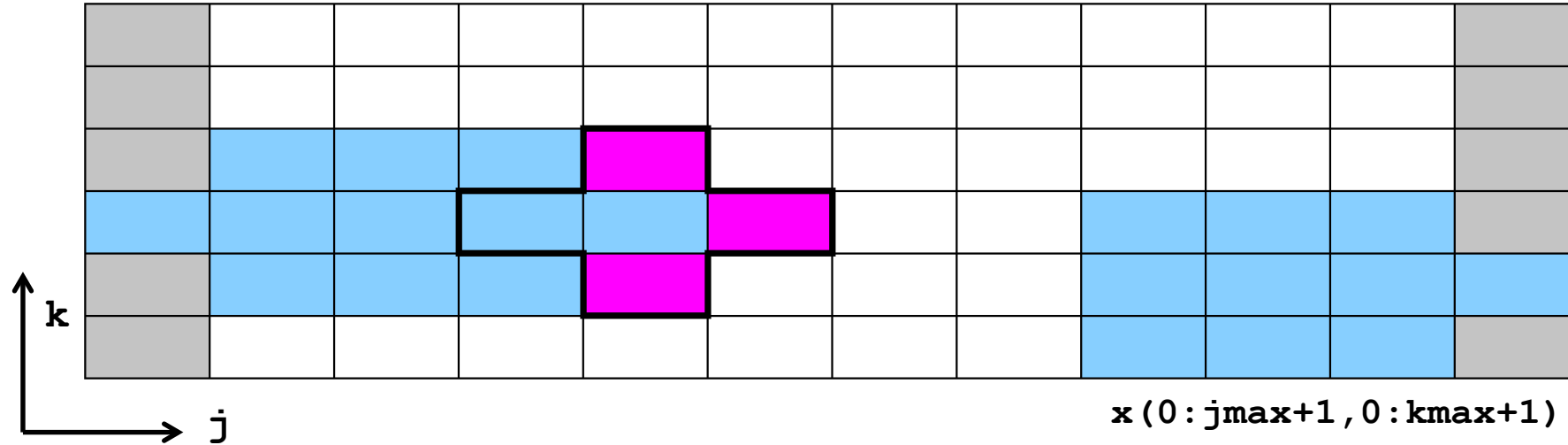


Case study: A Jacobi smoother

From 2D to 3D



2D:

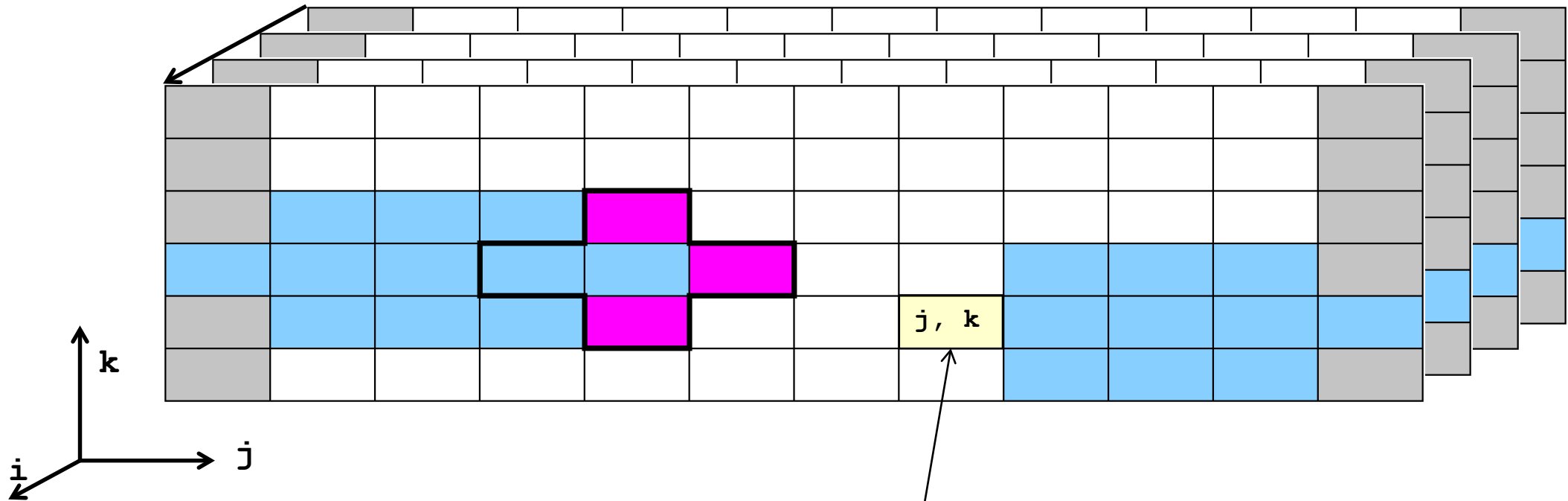


Towards 3D understanding

- Picture can be considered as 2D cut of 3D domain for a (new) fixed i index:

$$x(0:j_{\max}+1, 0:k_{\max}+1) \rightarrow x(i, 0:j_{\max}+1, 0:k_{\max}+1)$$

From 2D to 3D



- $\mathbf{x}(0:\mathbf{imax}+1, 0:\mathbf{jmax}+1, 0:\mathbf{kmax}+1)$ – Assume \mathbf{i} direction contiguous in main memory (Fortran notation)
- Stay at 2D picture and consider one cell of $\mathbf{j-k}$ plane as a contiguous slab of elements in \mathbf{i} direction: $\mathbf{x}(0:\mathbf{imax}, \mathbf{j}, \mathbf{k})$

Layer condition: From 2D 5-pt to 3D 7-pt 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
```

2D

$$3 \times j_{\max} \times 8 \text{ byte} < \frac{C}{2}$$

$$B_C = 24 \text{ B} / \text{LUP}$$

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

3D

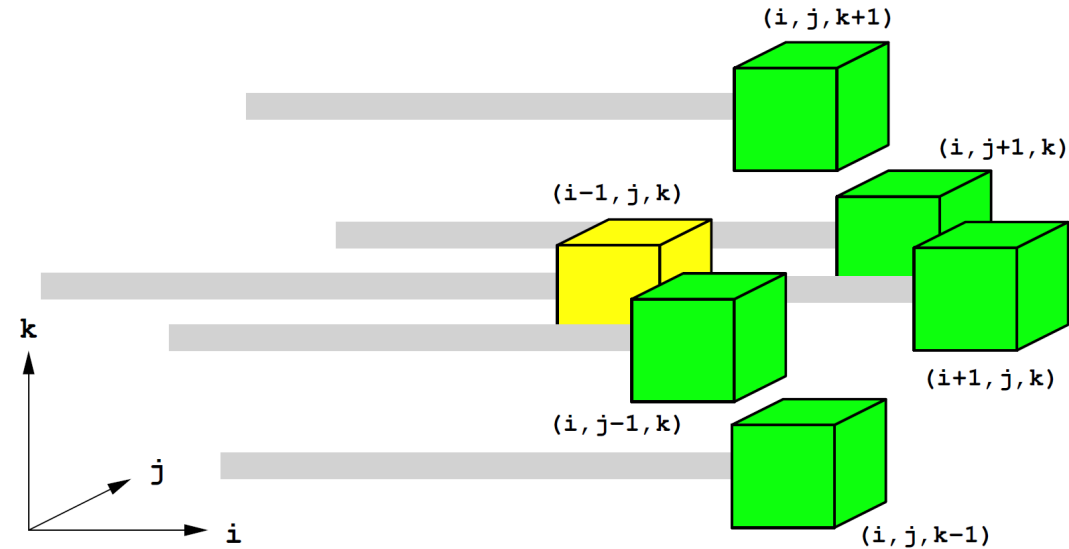
$$3 \times j_{\max} \times i_{\max} \times 8 \text{ byte} < \frac{C}{2}$$

$$B_C = 24 \text{ B} / \text{LUP}$$

3D 7-pt stencil

There is actually more than one layer condition in 3D

- “Outer” LC:
 $3 * i_{max} * j_{max}$
- “Inner” LC:
 $3 * i_{max}$ in the central layer



→ Code balance of

- 24 B/LUP if outer LC fulfilled
- 40 B/LUP if outer LC broken but inner LC fulfilled
- 56 B/LUP if inner & outer LC broken

Online Layer Condition calculator: <http://tiny.cc/LayerConditions>

Conclusions from the stencil example

- We have **made sense** of the memory-bound **performance** vs. problem size
 - “**Layer conditions**” lead to **predictions of code balance**
 - “**What part of the data comes from where**” is a crucial question
 - The model works only if the **bandwidth is “saturated”**
 - In-cache modeling is more involved
- **Avoiding slow data paths** == re-establishing the most favorable layer condition
- Improved code showed the **speedup predicted** by the model
- Optimal **blocking factor can be estimated**
 - Be guided by the cache size the **layer condition**
 - No need for exhaustive scan of “optimization space”
- **Food for thought**
 - Multi-dimensional loop blocking – would it make sense?
 - Can we choose a “better” OpenMP loop schedule?
 - What about temporal blocking?

Stencil references

- J. Hammer, G. Hager, J. Eitzinger, and G. Wellein: [Automatic Loop Kernel Analysis and Performance Modeling With Kerncraft](#). Proc. [PMBS15](#), the 6th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, in conjunction with ACM/IEEE Supercomputing 2015 ([SC15](#)), November 16, 2015, Austin, TX. [DOI: 10.1145/2832087.2832092](#), Preprint: [arXiv:1509.03778](#)
- H. Stengel, J. Treibig, G. Hager, and G. Wellein: [Quantifying performance bottlenecks of stencil computations using the Execution-Cache-Memory model](#). Proc. [ICS15](#), [DOI: 10.1145/2751205.2751240](#), Preprint: [arXiv:1410.5010](#)
- M. Wittmann, G. Hager, T. Zeiser, J. Treibig, and G. Wellein: [Chip-level and multi-node analysis of energy-optimized lattice-Boltzmann CFD simulations](#). Concurrency and Computation: Practice and Experience (2015). [DOI:10.1002/cpe.3489](#)
Preprint: [arXiv:1304.7664](#)
- J. Treibig, G. Wellein and G. Hager: [Efficient multicore-aware parallelization strategies for iterative stencil computations](#). Journal of Computational Science 2 (2), 130-137 (2011). [DOI 10.1016/j.jocs.2011.01.010](#)
- M. Wittmann, G. Hager, J. Treibig and G. Wellein: [Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters](#). Parallel Processing Letters 20 (4), 359-376 (2010).
- G. Wellein, G. Hager, T. Zeiser, M. Wittmann and H. Fehske: [Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization](#). Proc. COMPSAC 2009. [DOI: 10.1109/COMPSAC.2009.82](#)