APPM 4720/5720 — week 14:

Structured matrix computations

Gunnar Martinsson
The University of Colorado at Boulder

What is the cost of a matrix-vector multiply $\mathbf{b} = \mathbf{A}\mathbf{x}$ with an $N \times N$ matrix $\mathbf{A}$?

- If $\mathbf{A}$ is a general matrix, then the cost is $O(N^2)$.
- If $\mathbf{A}$ is sparse with say k elements per row, then the cost is $O(k\,N)$.
- If $\mathbf{A}$ is circulant (so that $\mathbf{A}(i,j) = \mathbf{a}(i-j)$), then the FFT renders the cost $O(N \log N)$. (Similar statements hold for Toeplitz matrices, Hankel matrices, etc.)
- If $\mathbf{A}$ has rank k (so that $\mathbf{A} = \mathbf{B}\mathbf{C}^*$ where $\mathbf{B}$ and $\mathbf{C}$ are $N \times k$), then the cost is $O(kN)$.

In general, we say that a matrix is *structured* if it admits algorithms for matrix-vector multiplication, that have lower complexity than that of a general matrix.

Example: Let $\{\boldsymbol{x}_i\}_{i=1}^N$ be a collection of points in $\mathbb{R}^2$ and set $\mathbf{A}(i,j) = \log|\boldsymbol{x}_i - \boldsymbol{x}_j|$. Then $\mathbf{q} \mapsto \mathbf{A}\mathbf{q}$ can be evaluated in $O(N)$ operations, and $\mathbf{A}$ is "structured."

The matrix $\mathbf{A}$ is a particular example of what we call a *rank-structured matrix.* These have the property that their off-diagonal blocks have numerically low rank.

Many rank structured matrices allow fast operations not only for matrix-vector multiply, but also for matrix inversion, LU-factorization, matrix-matrix multiply, etc.

This lecture describes a particularly simple class of structured matrices.

Review of the SVD and numerical rank:

Every $m \times n$ matrix $\mathbf{A}$ admits a factorization (with $r = \min(m,n)$):

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^* = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_r] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^* \\ \mathbf{v}_2^* \\ \vdots \\ \mathbf{v}_r^* \end{bmatrix} = \mathbf{U}\mathbf{D}\mathbf{V}^*.$$

The *singular values* σ_j are ordered so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$.

The *left singular vectors* $\mathbf{u}_j$ are orthonormal.

The *right singular vectors* $\mathbf{v}_j$ are orthonormal.

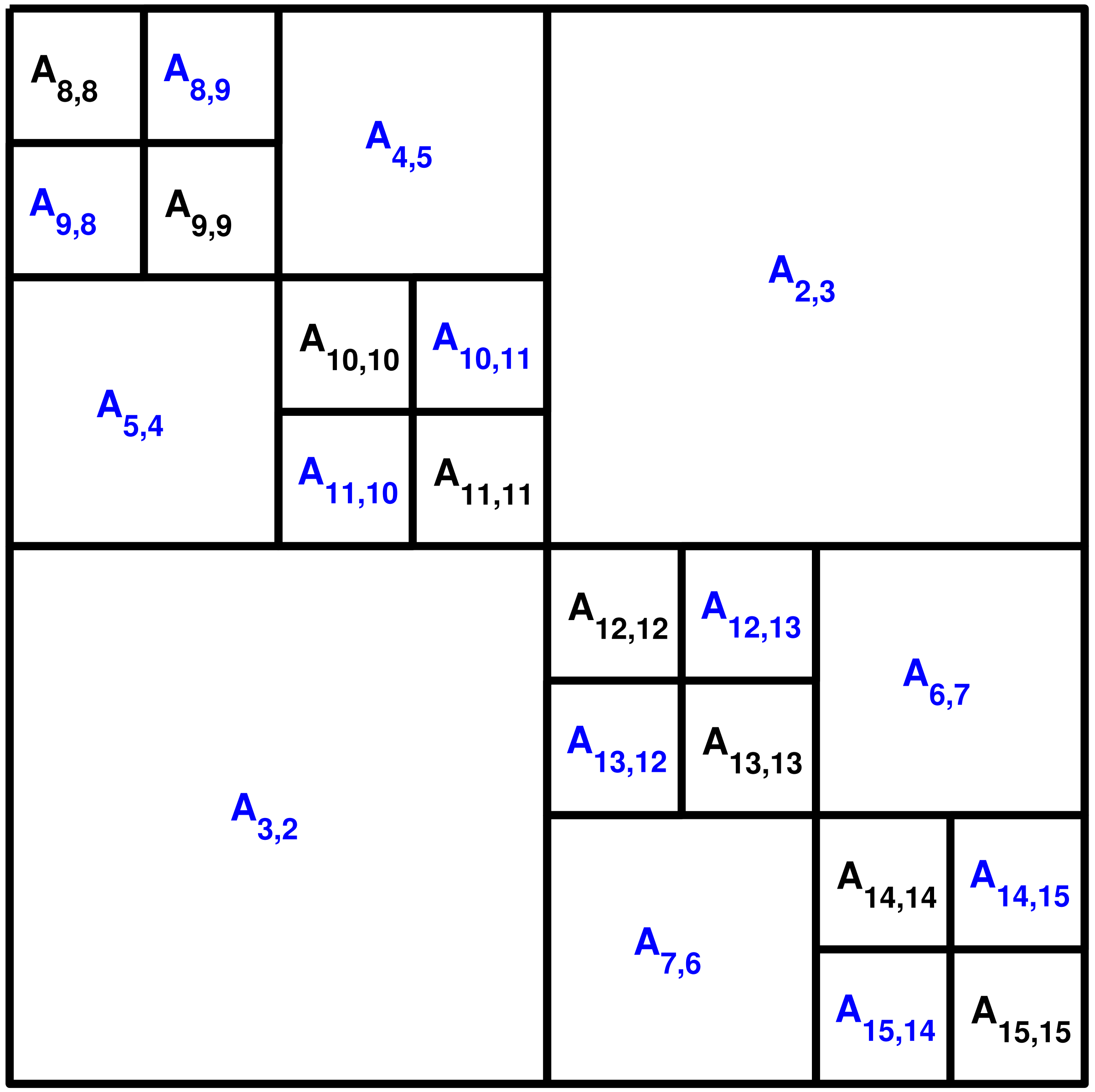
The SVD provides the exact answer to the low rank approximation problem:

$$\begin{aligned} \sigma_{j+1} &= \min\{||\mathbf{A} - \mathbf{B}|| :\ \mathbf{B} \text{ has rank } j\}, \\ \sum_{i=1}^{j} \sigma_i \mathbf{u}_i \mathbf{v}_i^* &= \operatorname{argmin}\{||\mathbf{A} - \mathbf{B}|| :\ \mathbf{B} \text{ has rank } j\}. \end{aligned}$$

Definition: We say that $\mathbf{A}$ has ε-rank (at most) k if $\sigma_{k+1} \leq \varepsilon$.

A very simple family of rank structured matrices

We informally say that a matrix is in $\mathcal{S}$-format if it can be tesselated "like this":



We require that

- the diagonal blocks are of size at most $2k \times 2k$
- the off-diagonal blocks (in blue in the figure) have rank at most k.

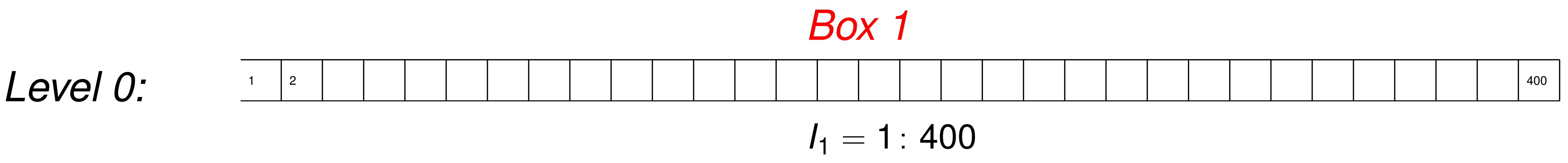
The cost of performing a matvec is then

$$\underbrace{2 \times \frac{N}{2} k + 4 \times \frac{N}{4} k + 8 \times \frac{N}{8} k + \cdots}_{\log N \text{ terms}} \sim N \log(N)\, k.$$

Note: The "S" in "$\mathcal{S}$-matrix" is for Simple — the term is not standard by any means ...

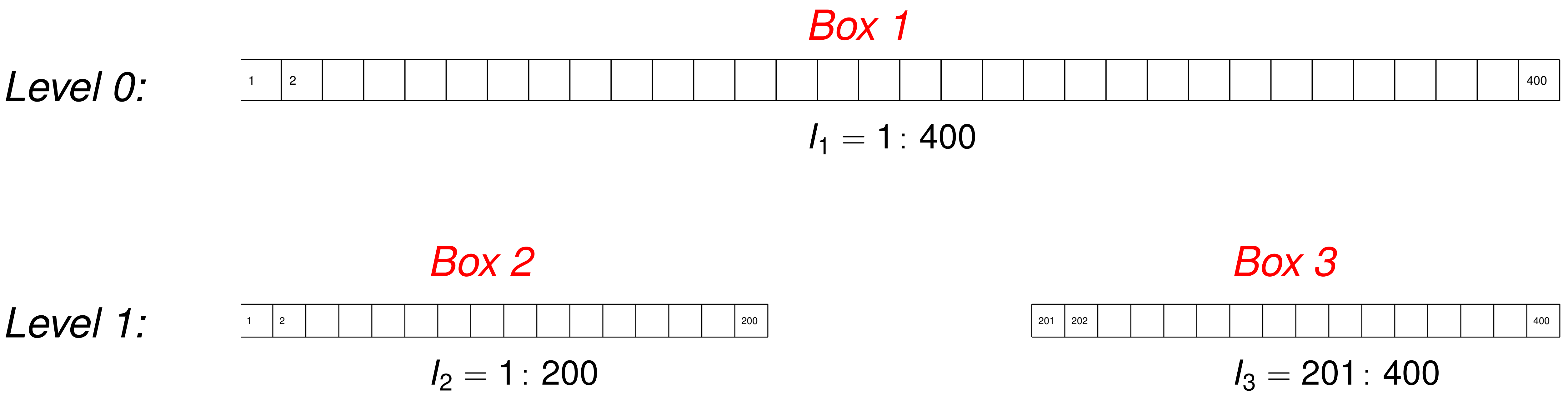
Notation: Let **A** be an $N \times N$ matrix. To properly define an $\mathcal{S}$-matrix, we first need to define the concept of an *index tree* on the index vector $I = [1, 2, 3, \ldots, N]$.

The idea is to execute recursive bijection:



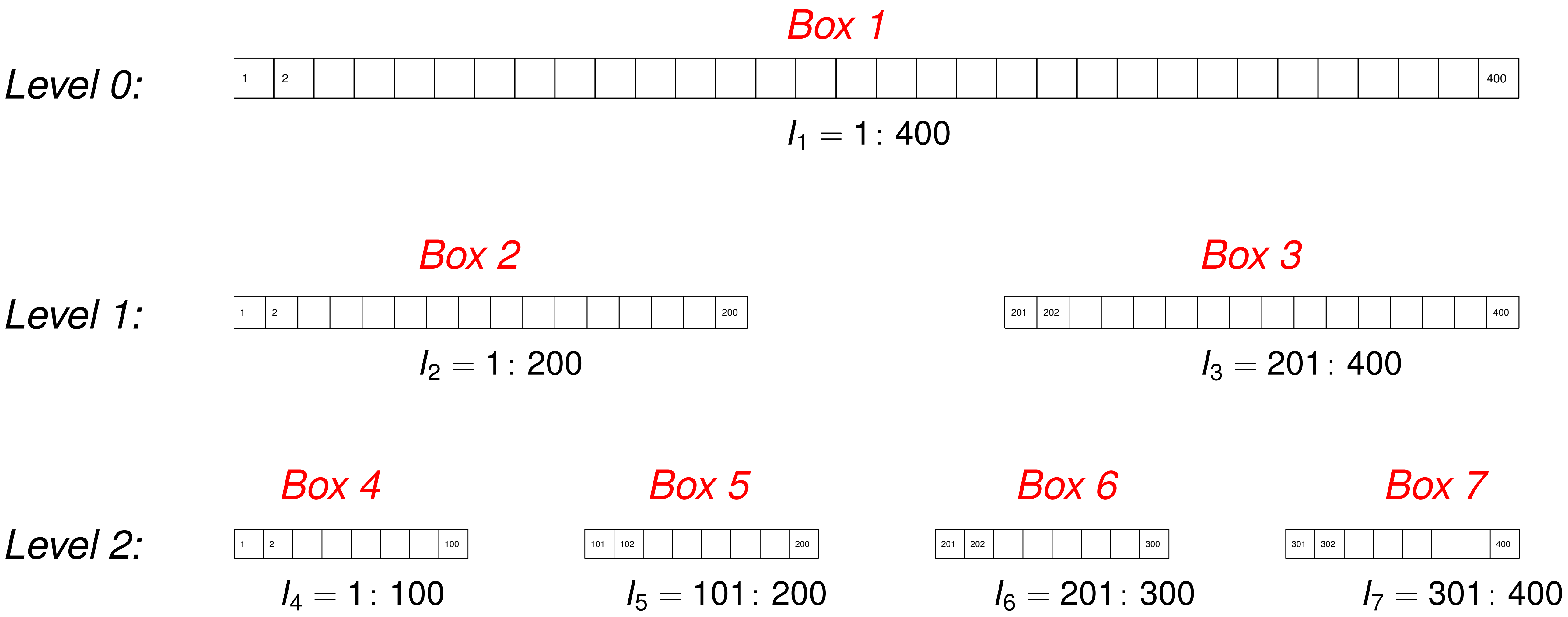
Notation: Let $\mathbf{A}$ be an $N \times N$ matrix. To properly define an $\mathcal{S}$-matrix, we first need to define the concept of an *index tree* on the index vector $I = [1, 2, 3, \dots, N]$.

The idea is to execute recursive bijection:



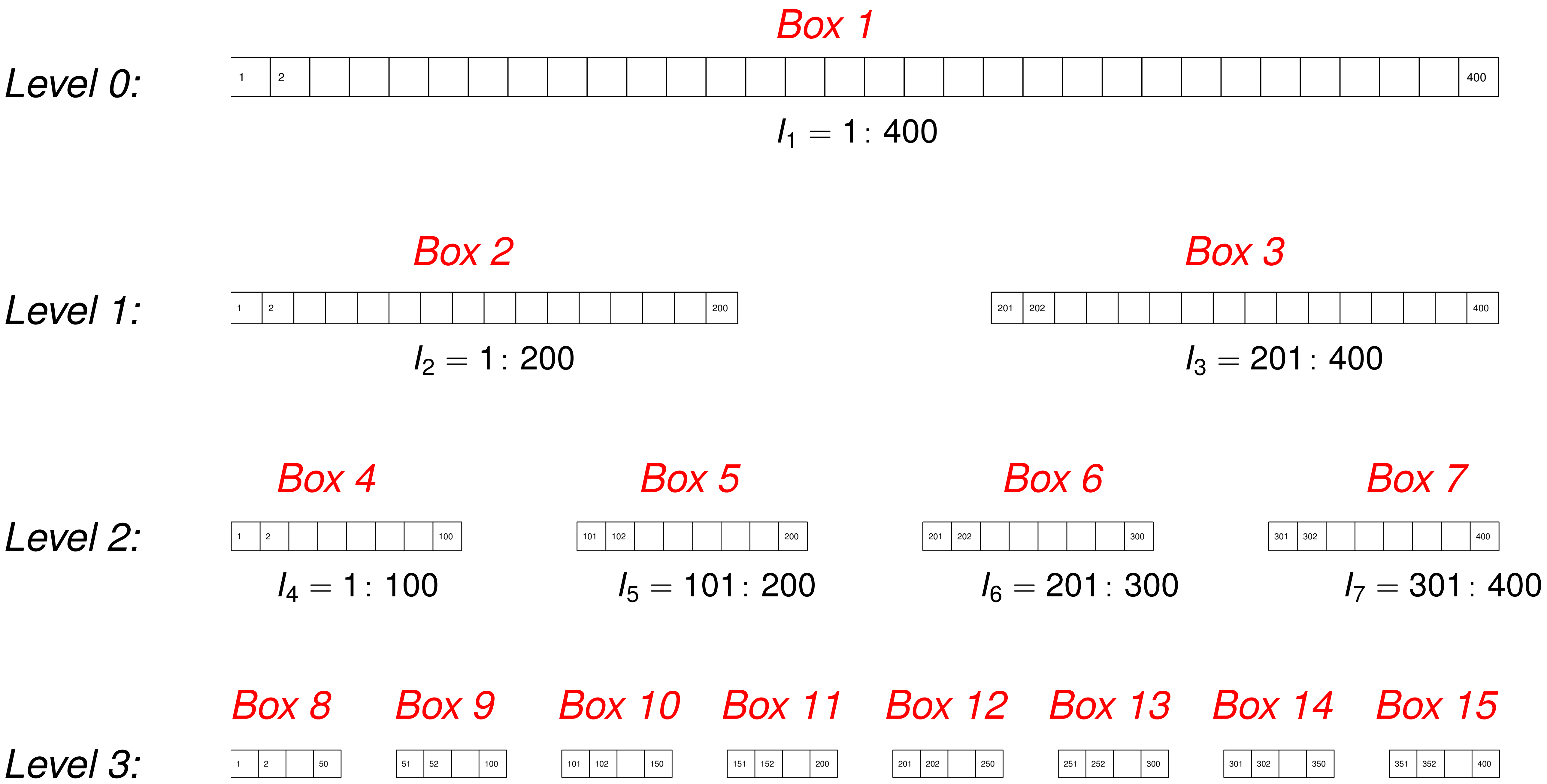
Notation: Let $\mathbf{A}$ be an $N \times N$ matrix. To properly define an $\mathcal{S}$-matrix, we first need to define the concept of an *index tree* on the index vector $I = [1, 2, 3, \ldots, N]$.

The idea is to execute recursive bijection:



Notation: Let $\mathbf{A}$ be an $N \times N$ matrix. To properly define an $\mathcal{S}$-matrix, we first need to define the concept of an *index tree* on the index vector $I = [1, 2, 3, \dots, N]$.

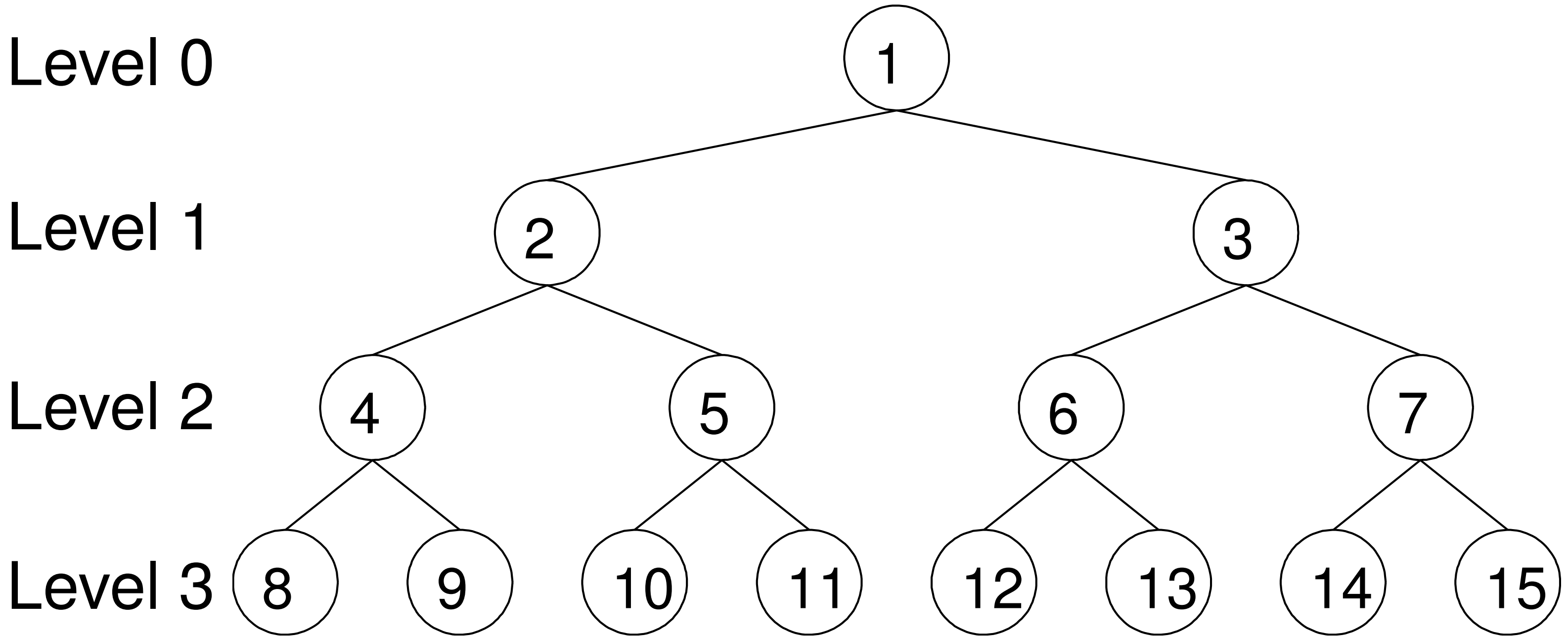
The idea is to execute recursive bijection:



Notation: Let $\mathbf{A}$ be an $\mathcal{S}$-matrix of size $N \times N$.

Suppose $\mathcal{T}$ is a binary tree on the index vector $I = [1, 2, 3, \dots, N]$.

For a node τ in the tree, let I_τ denote the corresponding index vector.



$I_1 = [1, 2, \dots, 400]$

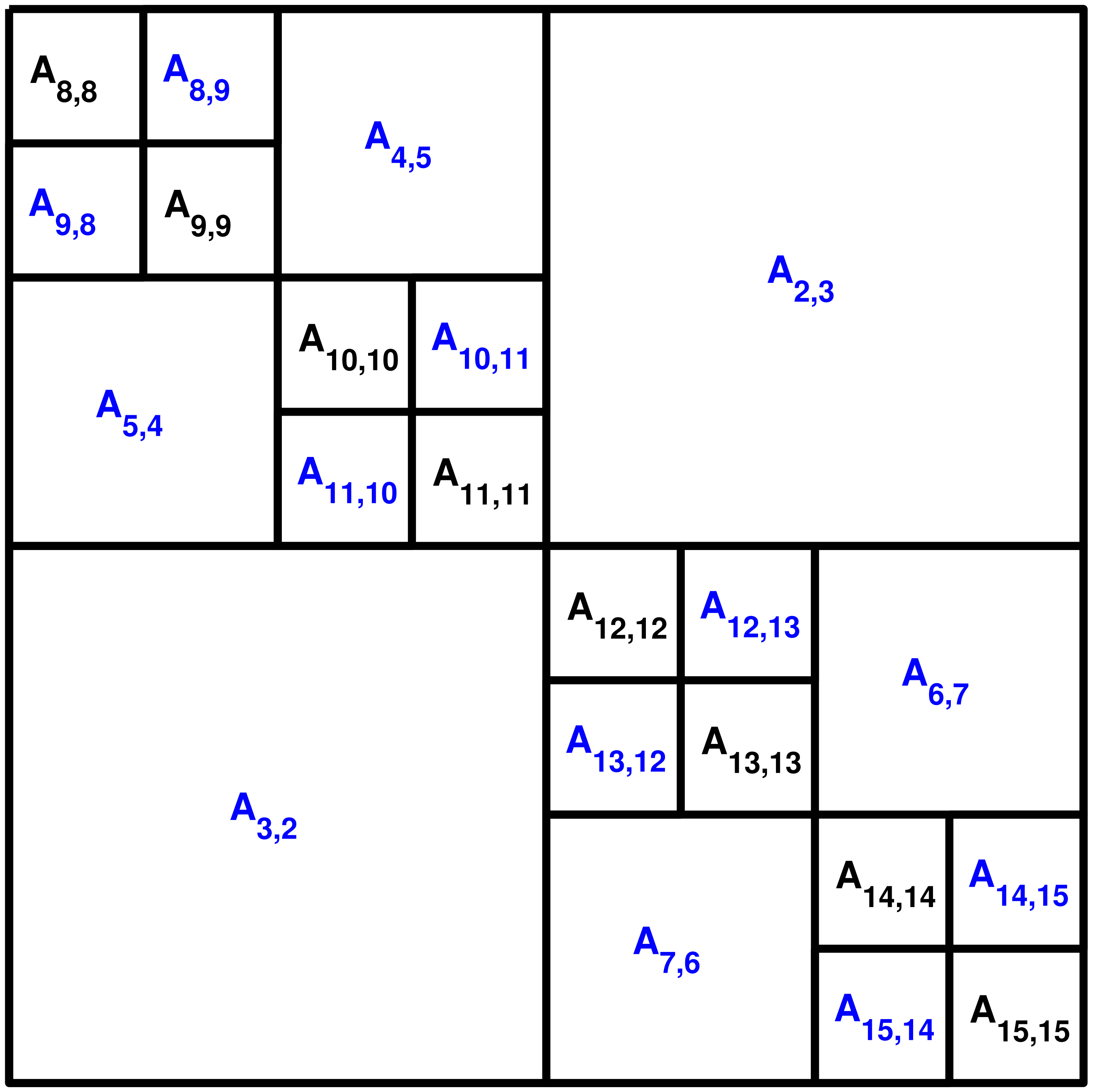
$I_2 = [1, 2, \dots, 200]$, $I_3 = [201, 202, \dots, 400]$

$I_4 = [1, 2, \dots, 100]$, $I_5 = [101, 102, \dots, 200]$, $\dots$

$I_8 = [1, 2, \dots, 50]$, $I_9 = [51, 52, \dots, 100]$, $\dots$

For nodes σ and τ on the same level, set $\mathbf{A}_{\sigma,\tau} = \mathbf{A}(I_\sigma, I_\tau)$.

With the binary tree of indices $\{I_\tau\}_\tau$, and with $\mathbf{A}_{\sigma,\tau} = \mathbf{A}(I_\sigma,\, I_\tau)$, we get

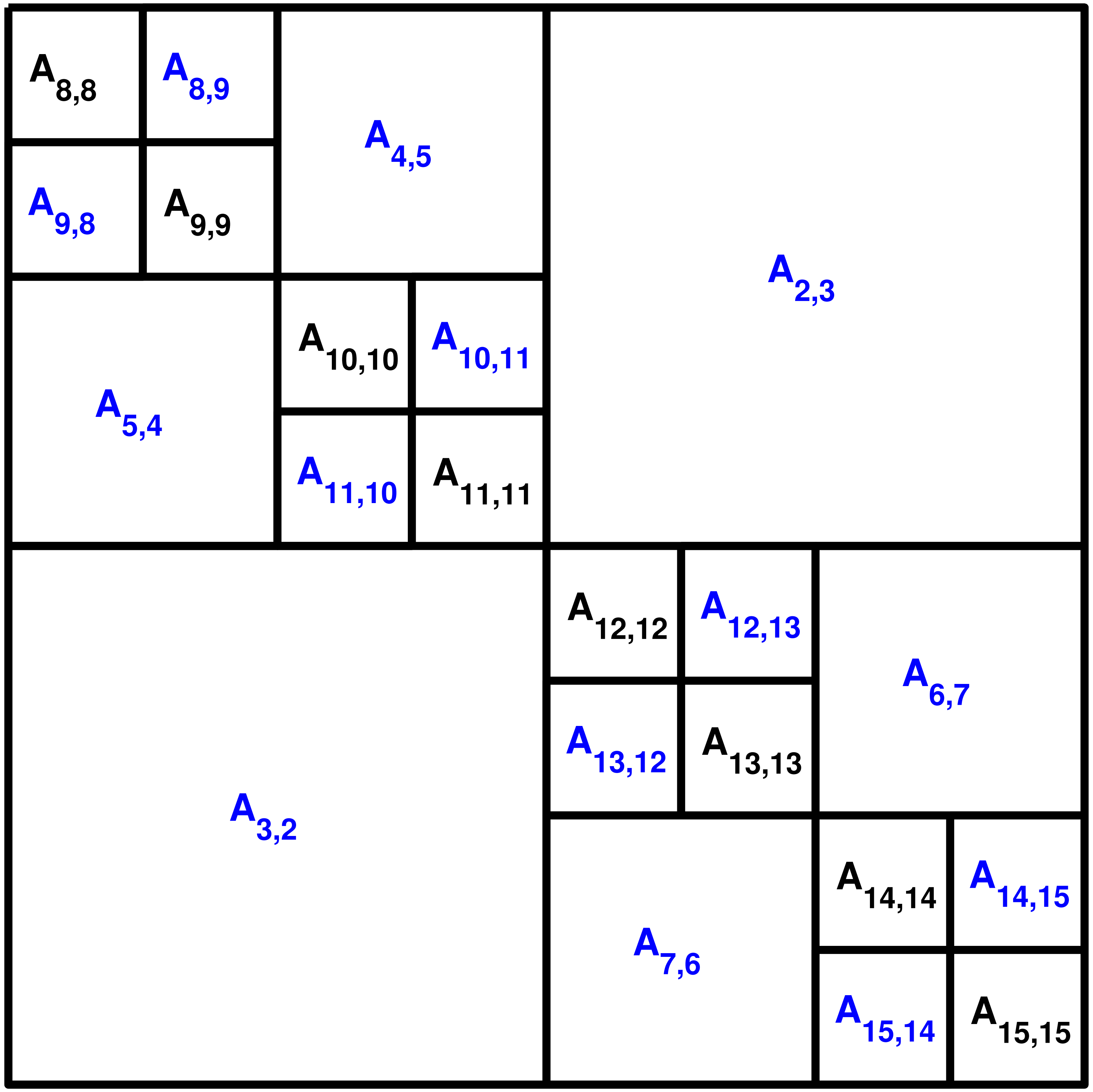


Every "blue" matrix has numerically low rank and admits a factorization

$$\begin{array}{ccccc} \mathbf{A}_{\sigma,\tau} & = & \mathbf{U}_\sigma & \tilde{\mathbf{A}}_{\sigma,\tau} & \mathbf{V}_\tau \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

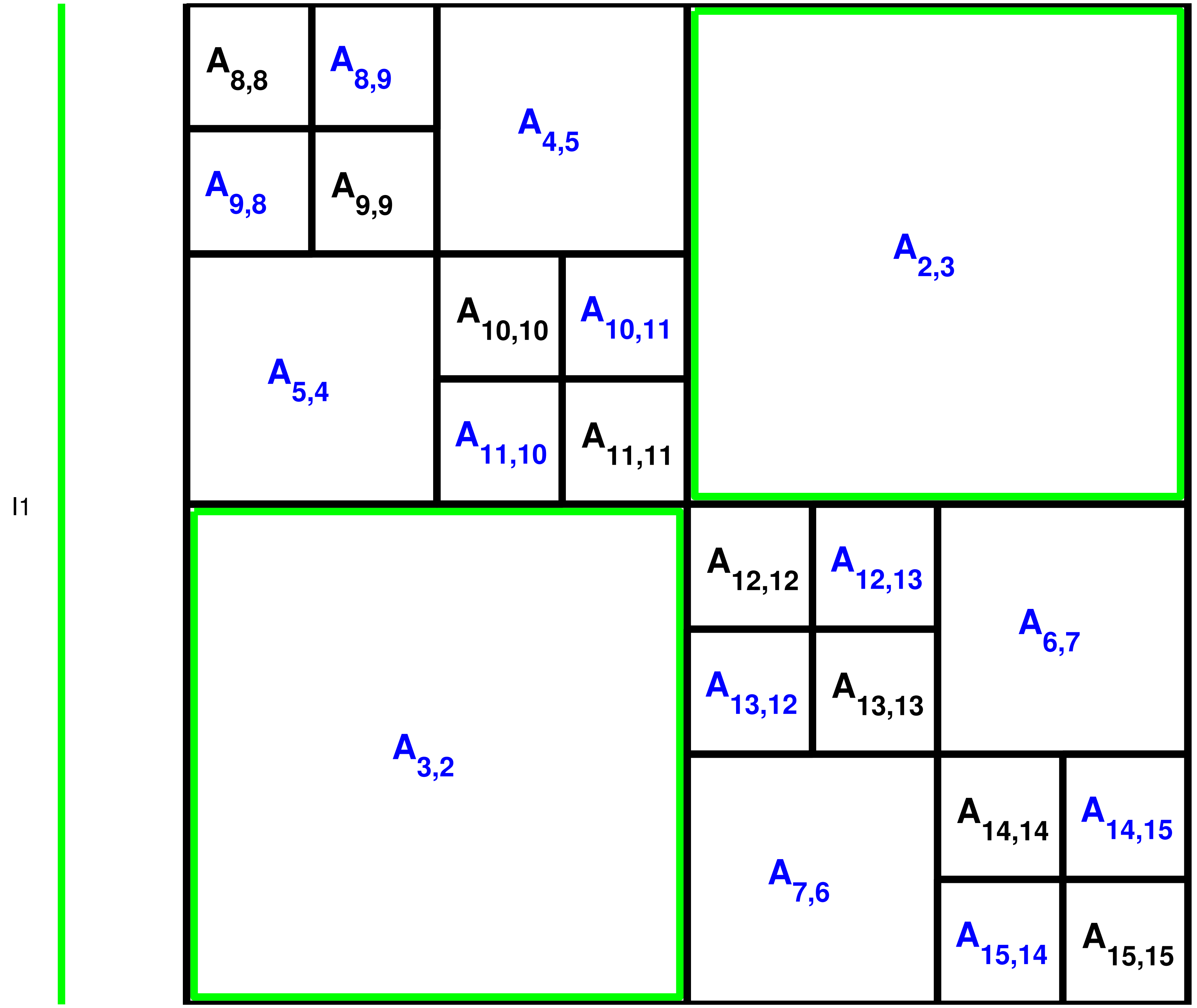
Operations involving a blue matrix are executed using its compact representation.

Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* INITIALIZATION



$$\mathbf{b} = \mathbf{0}$$

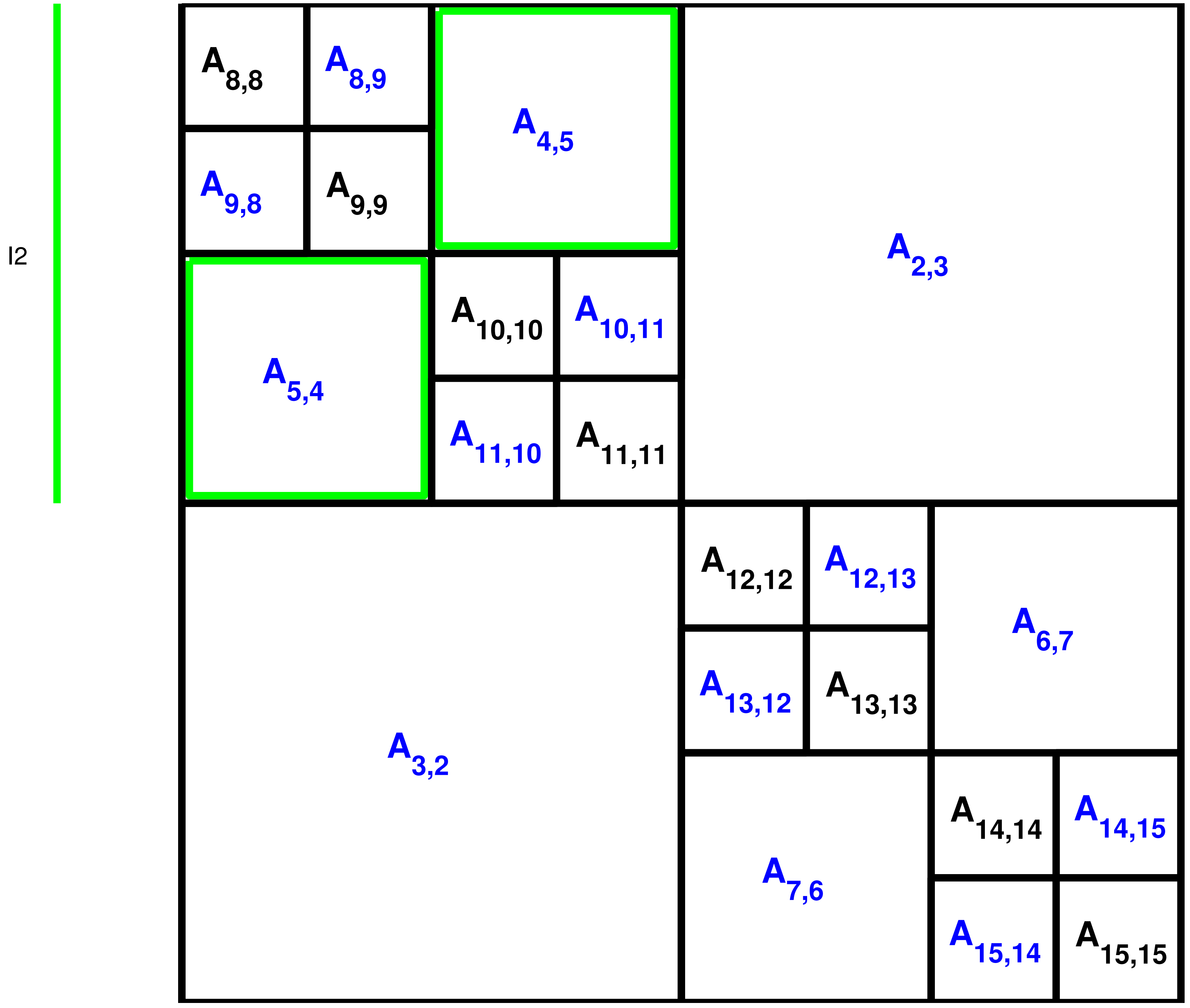
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$: Process node 1



The children of node 1 are $\{2, 3\}$;

$$\begin{bmatrix} \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{2,3} \\ \mathbf{A}_{3,2} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}$$

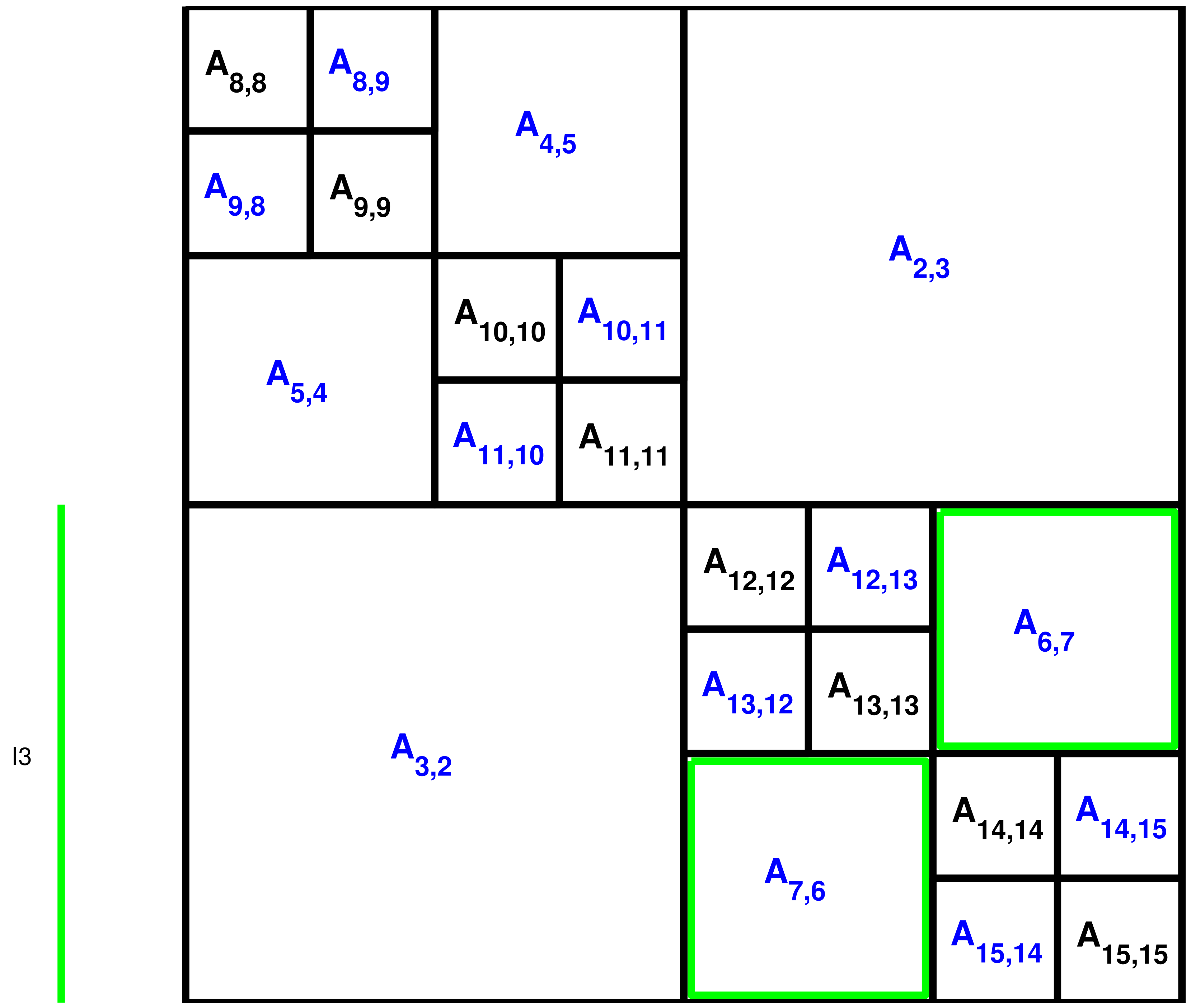
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 2



The children of node 2 are $\{4, 5\}$;

$$\begin{bmatrix} \mathbf{b}_4 \\ \mathbf{b}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_4 \\ \mathbf{b}_5 \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{4,5} \\ \mathbf{A}_{5,4} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix}$$

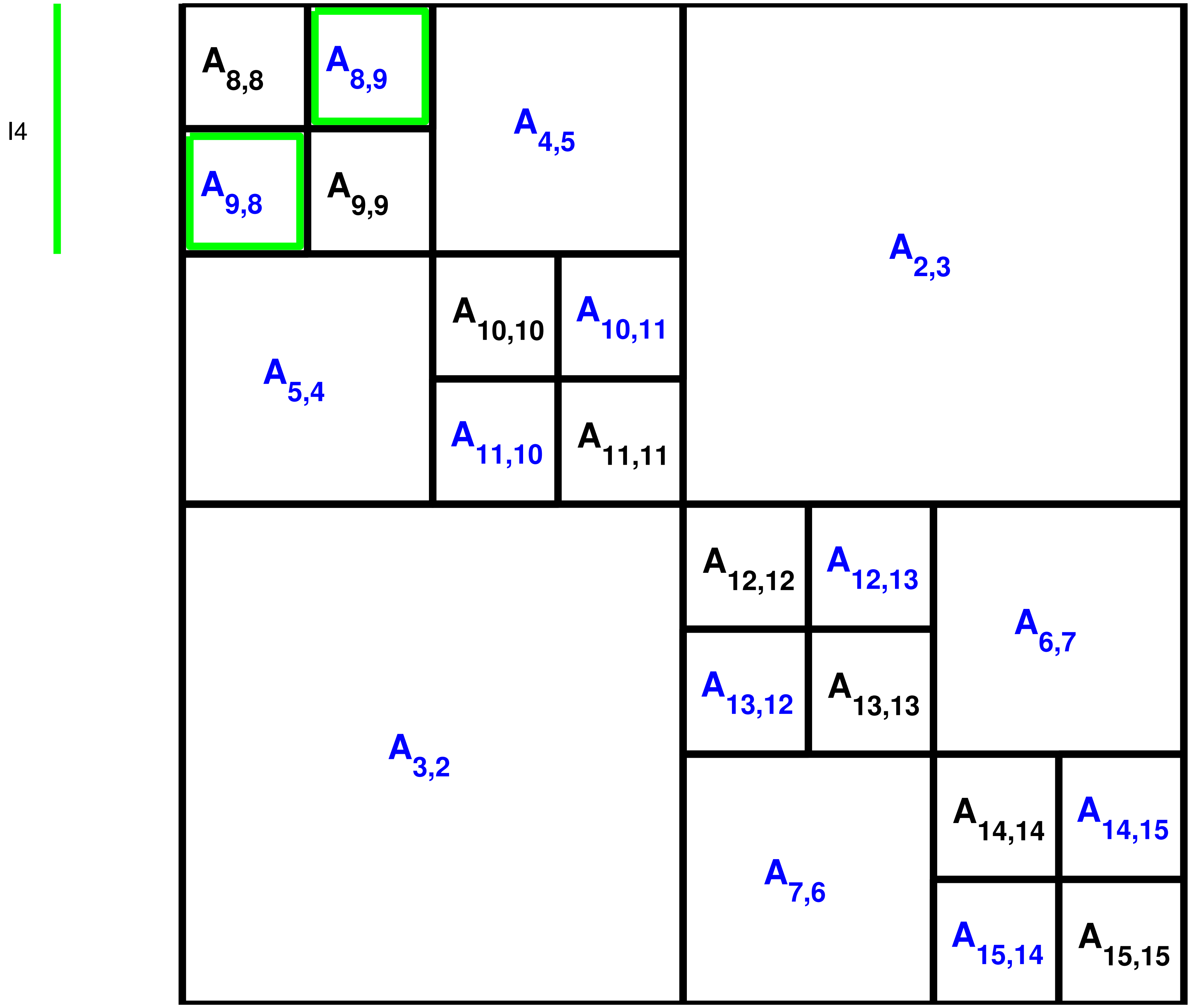
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 3



The children of node 3 are $\{6, 7\}$;

$$\begin{bmatrix} \mathbf{b}_6 \\ \mathbf{b}_7 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_6 \\ \mathbf{b}_7 \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{6,7} \\ \mathbf{A}_{7,6} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_6 \\ \mathbf{x}_7 \end{bmatrix}$$

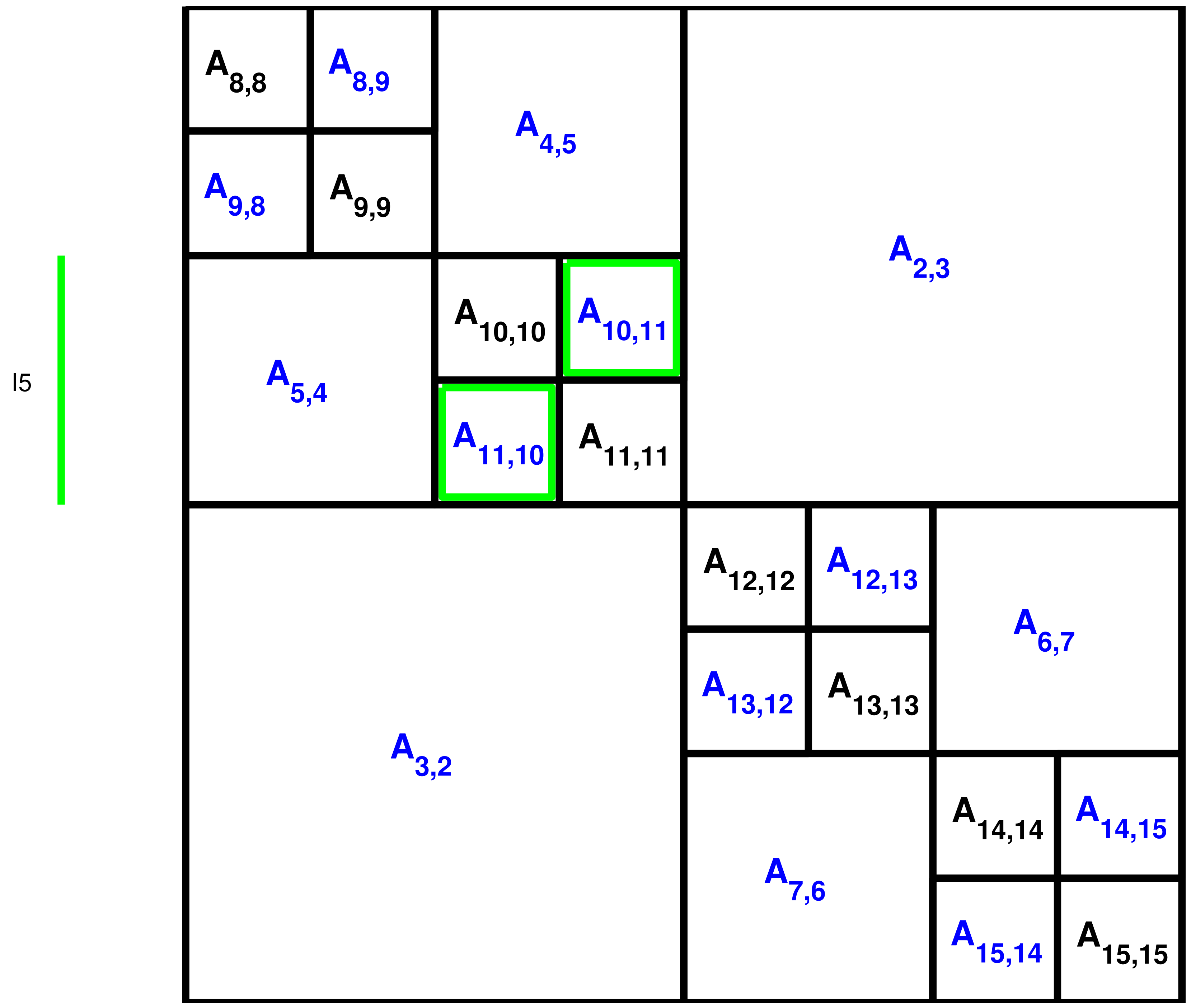
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 4



The children of node 4 are $\{8, 9\}$;

$$\begin{bmatrix} \mathbf{b}_8 \\ \mathbf{b}_9 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_8 \\ \mathbf{b}_9 \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{8,9} \\ \mathbf{A}_{9,8} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_8 \\ \mathbf{x}_9 \end{bmatrix}$$

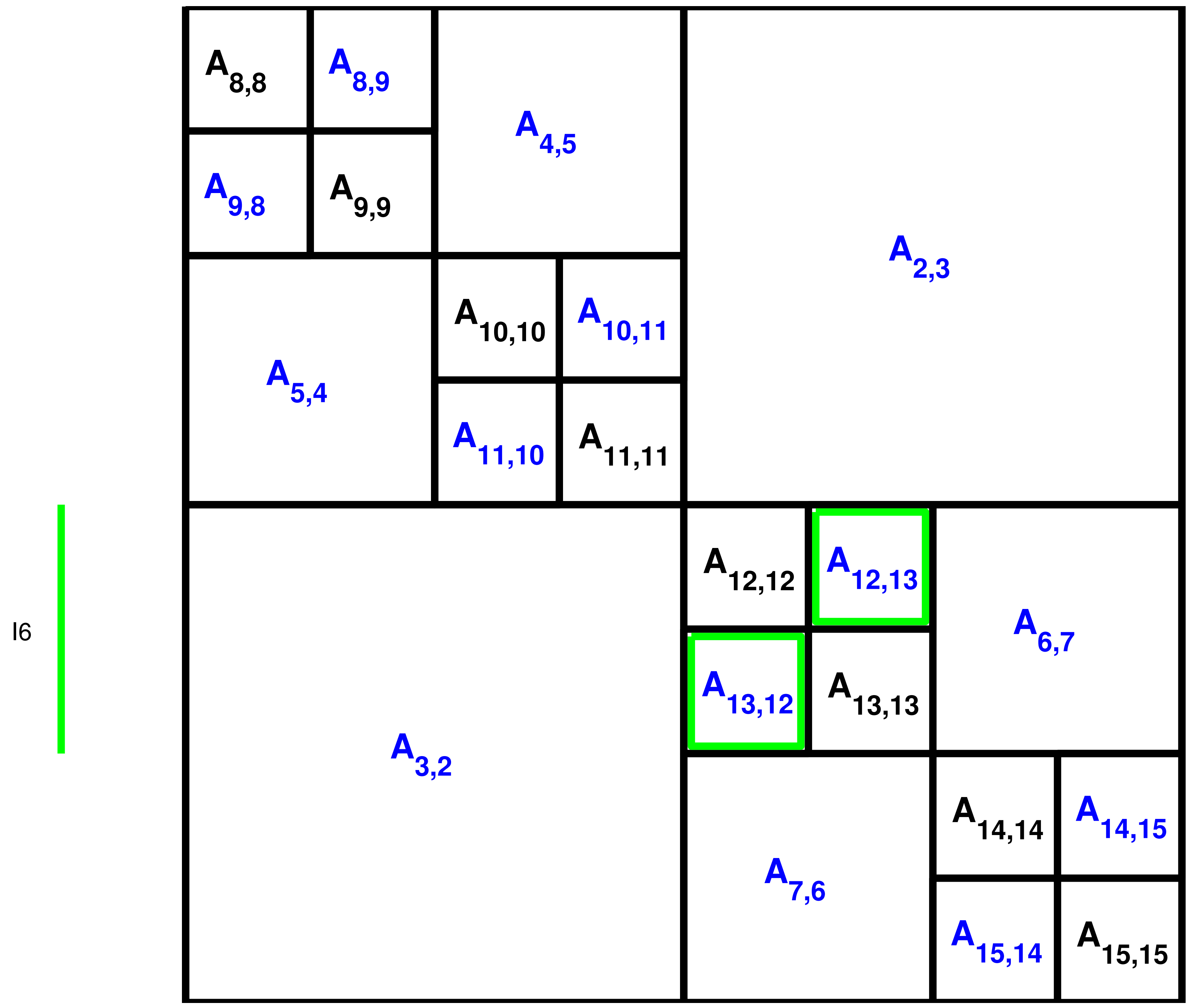
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 5



The children of node 5 are $\{10, 11\}$;

$$\begin{bmatrix} \mathbf{b}_{10} \\ \mathbf{b}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{10} \\ \mathbf{b}_{11} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{10,11} \\ \mathbf{A}_{11,10} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{10} \\ \mathbf{x}_{11} \end{bmatrix}$$

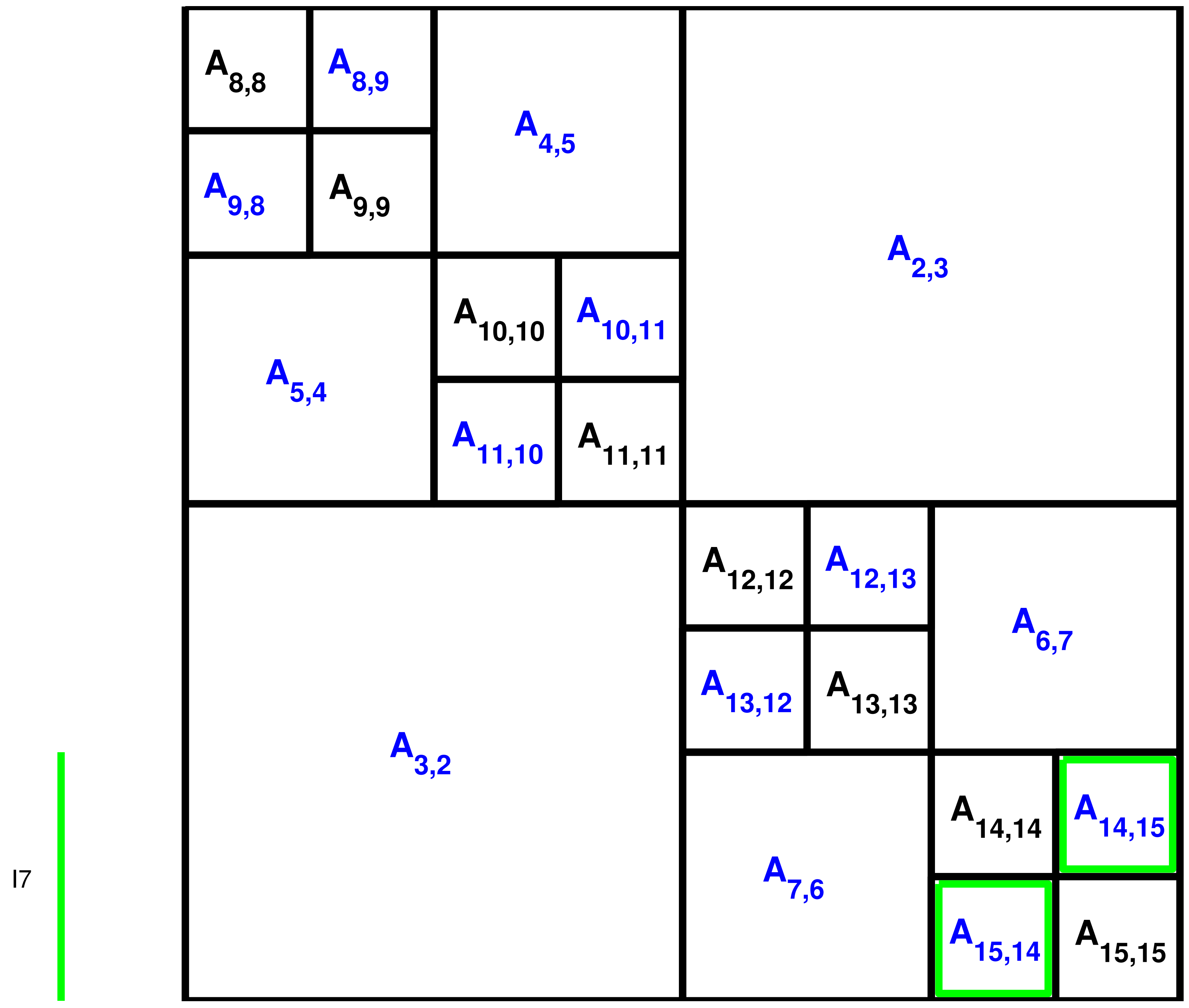
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 6



The children of node 6 are $\{12, 13\}$;

$$\begin{bmatrix} \mathbf{b}_{12} \\ \mathbf{b}_{13} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{12} \\ \mathbf{b}_{13} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{12,13} \\ \mathbf{A}_{13,12} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{12} \\ \mathbf{x}_{13} \end{bmatrix}$$

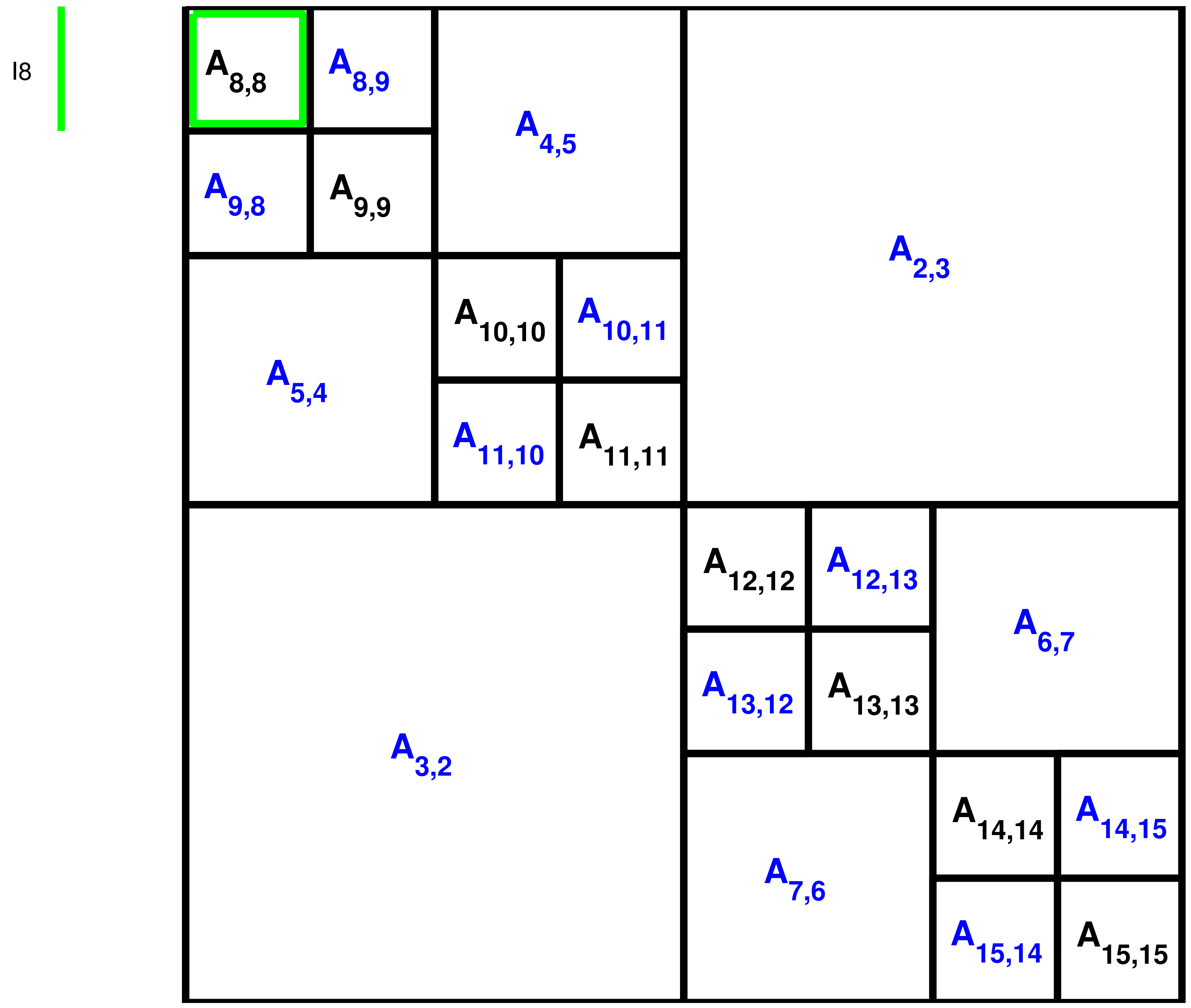
Matrix vector multiply $\mathbf{b} = \mathbf{A}\mathbf{x}$*:* Process node 7



The children of node 7 are $\{14, 15\}$;

$$\begin{bmatrix} \mathbf{b}_{14} \\ \mathbf{b}_{15} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{14} \\ \mathbf{b}_{15} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{14,15} \\ \mathbf{A}_{15,14} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{14} \\ \mathbf{x}_{15} \end{bmatrix}$$

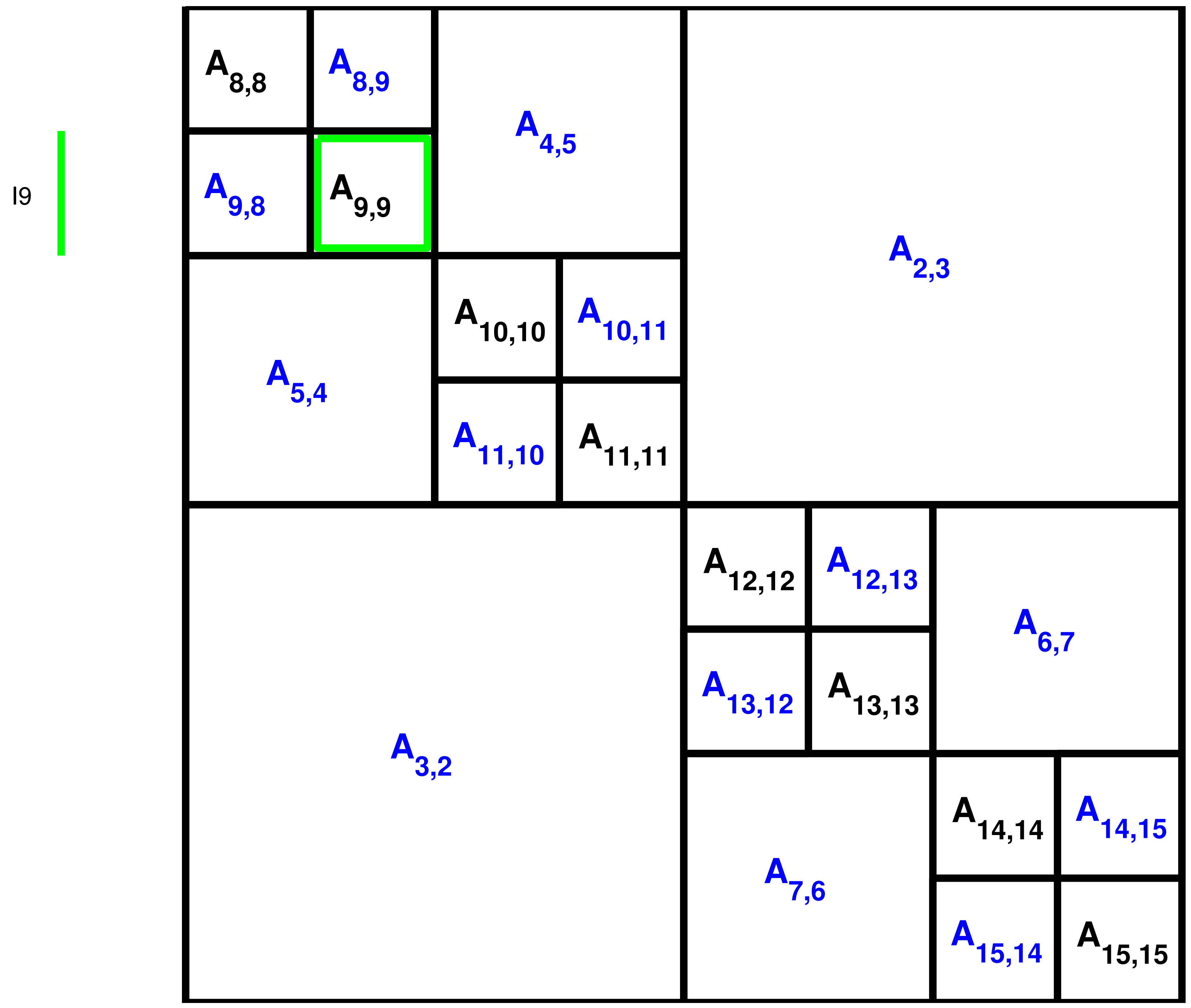
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 8



Node 8 is a leaf

$$\mathbf{b}_8 = \mathbf{b}_8 + \mathbf{A}_{8,8}\mathbf{x}_8$$

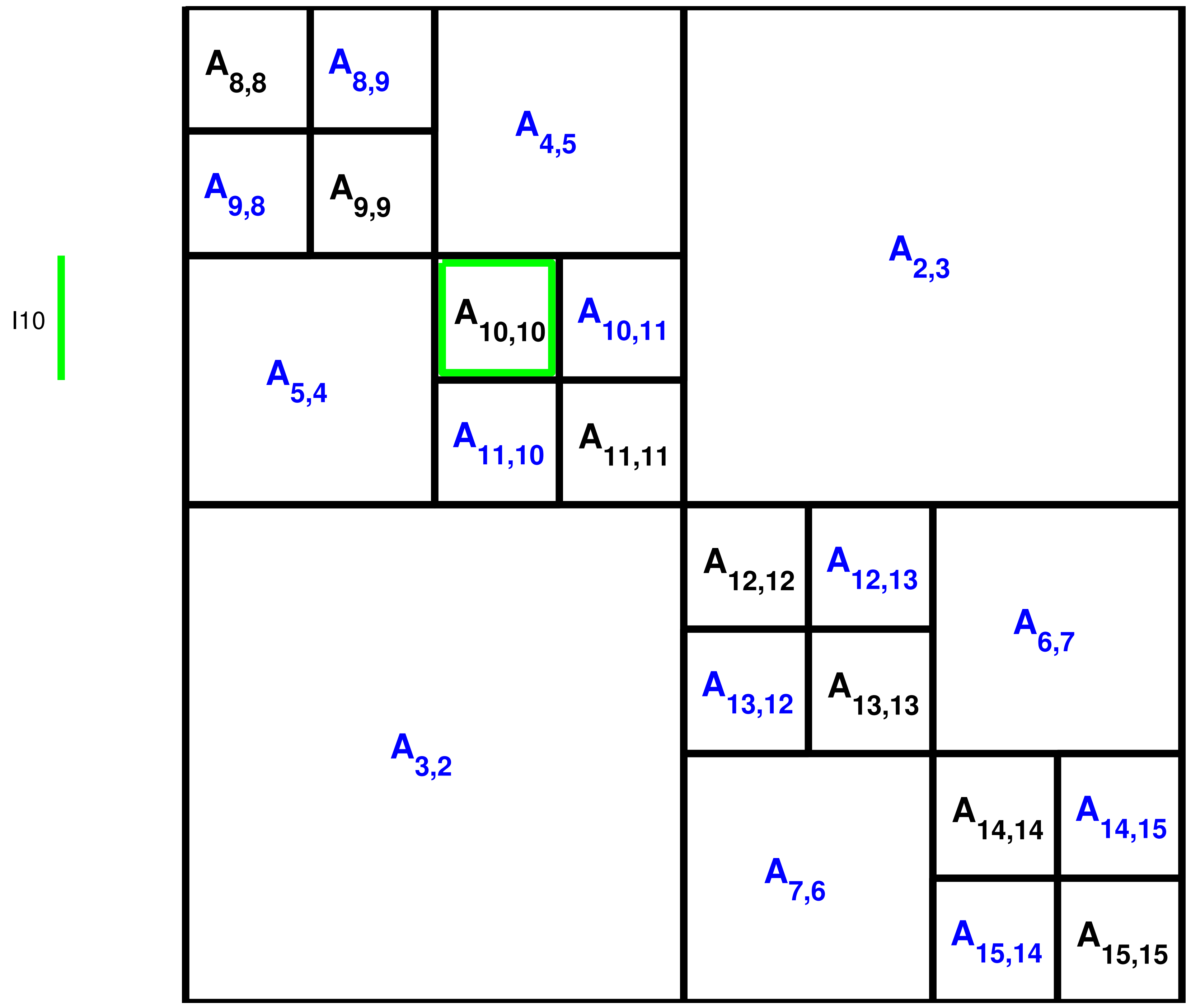
Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$*:* Process node 9



Node 9 is a leaf

$$\mathbf{b}_9 = \mathbf{b}_9 + \mathbf{A}_{9,9}\mathbf{x}_9$$

Matrix vector multiply $\mathbf{b} = \mathbf{Ax}$: Process node 10



Node 10 is a leaf

$$\mathbf{b}_{10} = \mathbf{b}_{10} + \mathbf{A}_{10,10}\mathbf{x}_{10}$$

ETC

MATRIX-VECTOR MULTIPLY FOR AN $\mathcal{S}$-MATRIX:

```
**b** $= \mathbf{0}$
**loop** $\tau$ is a node in the tree
    **if** $\tau$ is a leaf
        $\mathbf{b}(I_\tau) = \mathbf{b}(I_\tau) + \mathbf{A}_{\tau,\tau}\,\mathbf{x}(I_\tau)$
    **else**
        Let $\sigma_1$ and $\sigma_2$ denote the children of $\tau$.
        $\mathbf{b}(I_\tau) = \mathbf{b}(I_\tau) + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{\sigma_1,\sigma_2} \\ \mathbf{A}_{\sigma_2,\sigma_1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}(I_{\sigma_1}) \\ \mathbf{x}(I_{\sigma_2}) \end{bmatrix}.$
    **end if**
**end loop**
```

Note: The loop can be traversed in any order. This makes parallelization trivial.

Question: How do you find the factors in the sibling pairs?

In the context of direct solvers for finite element and finite difference problems, this turns out to not be an issue — the matrices are typically built up piecemeal.

In the context of integral equations (where the coefficient matrix will be dense), it is a major issue. Overcoming this was crucial to much of the recent progress in the field.

Recursive representations of rank-structured matrix algorithms:

We can define the $\mathcal{S}$-matrix format in recursive form by saying that **A** is a $\mathcal{S}$-matrix with internal rank k if either **A** is itself of dimension at most k, or if **A** admits a blocking

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix},$$

where $\mathbf{A}_1$ and $\mathbf{A}_2$ are $\mathcal{S}$-matrices, and $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ are of rank at most k.

The formula for the matrix-vector product can then be written as follows:

```
function b = matvec(A, x)

if A is dense
    $\mathbf{b} = \mathbf{A}\mathbf{x}$
else
    Split $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$.
    $\mathbf{b} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{11} \\ \mathbf{A}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} + \begin{bmatrix} \text{matvec}(\mathbf{A}_{11}, \mathbf{x}_1) \\ \text{matvec}(\mathbf{A}_{22}, \mathbf{x}_2) \end{bmatrix}$.
end
```

With the recursive representation, we can easily derive a formula for $\mathbf{A}^{-1}$.

First note that for any 2×2 block matrix $\mathbf{A}$ we have

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{S}_{11}^{-1} & -\mathbf{S}_{11}^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \\ -\mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{S}_{11}^{-1} & \mathbf{A}_{22}^{-1} + \mathbf{A}_{22}^{-1}\mathbf{A}_{21}\mathbf{S}_{11}^{-1}\mathbf{A}_{12}\mathbf{A}_{22}^{-1} \end{bmatrix}$$

where $\mathbf{S}_{11} = \mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}$ (provided that both $\mathbf{A}_{22}^{-1}$ and $\mathbf{S}_{11}^{-1}$ exist). This leads to :

```
function $\mathbf{B} = \text{matinv}(\mathbf{A})$
if $\mathbf{A}$ is dense
    $\mathbf{B} = \mathbf{A}^{-1}$
else
    Split $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$.
    $\mathbf{X}_{22} = \text{matinv}(\mathbf{A}_{22})$
    $\mathbf{T}_{11} = \text{matinv}(\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{X}_{22}\mathbf{A}_{21})$.
    $\mathbf{B} = \begin{bmatrix} \mathbf{T}_{11} & -\mathbf{T}_{11}\mathbf{A}_{12}\mathbf{X}_{22} \\ -\mathbf{T}_{11}\mathbf{A}_{21}\mathbf{X}_{22} & \mathbf{X}_{22} + \mathbf{X}_{22}\mathbf{A}_{21}\mathbf{T}_{11}\mathbf{A}_{21}\mathbf{X}_{22} \end{bmatrix}$.
end
```

Note: You need a low-rank update, recompression, etc.

There exist more "elegant" recursions. For instance, recall the Woodbury formula:

Suppose that an $n \times n$ matrix $\mathbf{A}$ can be split into

$$\begin{array}{ccccccc} \mathbf{A} & = & \mathbf{D} & + & \mathbf{U} & \tilde{\mathbf{A}} & \mathbf{V}^* \\ n \times n & & n \times n & & n \times k & k \times k & k \times n \end{array}$$

where $\mathbf{D}$ is a matrix that is for some reason easy to invert (diagonal, block-diagonal, ...), and $\mathbf{U}\tilde{\mathbf{A}}\mathbf{V}^*$ is a "rank-k" correction.

Then the Woodbury formula states that

$$\begin{array}{ccccccc} \big(\mathbf{D} + \mathbf{U}\tilde{\mathbf{A}}\mathbf{V}^*\big)^{-1} & = & \mathbf{D}^{-1} & - & \mathbf{D}^{-1}\mathbf{U} & \big(\tilde{\mathbf{A}} + \mathbf{V}^*\mathbf{D}^{-1}\mathbf{U}\big)^{-1} & \mathbf{V}^*\mathbf{D}^{-1}. \\ n \times n & & n \times n & & n \times k & k \times k & k \times n \end{array}$$

The point is that we can construct $\mathbf{A}^{-1}$ by executing:

1. Invert $\mathbf{D}$.
2. Invert the $k \times k$ matrix $\tilde{\mathbf{A}} + \mathbf{V}^*\mathbf{D}^{-1}\mathbf{U}$.
3. Perform a rank-k update to $\mathbf{D}^{-1}$.

Inversion of $\mathcal{S}$-matrix using Woodbury

Recall the Woodbury formula

$$(\mathbf{D} + \mathbf{U}\tilde{\mathbf{A}}\mathbf{V}^*)^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{U}(\tilde{\mathbf{A}} + \mathbf{V}^*\mathbf{D}^{-1}\mathbf{U})^{-1}\mathbf{V}^*\mathbf{D}^{-1}.$$

For an $\mathcal{S}$-matrix, we have

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix} + \begin{bmatrix} \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^* & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^* \end{bmatrix}.$$

Applying the Woodbury formula, we find, with $\mathbf{S}_{11} = \mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1$ and $\mathbf{S}_2 = \mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2$,

$$\begin{array}{ccccccc} \mathbf{A}^{-1} & = & \begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1} \end{bmatrix} & + & \begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix} & \begin{bmatrix} \mathbf{S}_1 & \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} & \mathbf{S}_2 \end{bmatrix}^{-1} & \begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1} \end{bmatrix}. \\ 2n \times 2n & & 2n \times 2n & & 2n \times 2k & 2k \times 2k & 2k \times 2n \end{array}$$

The recursion is now “cleaner,” as we can process $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ independently.
(In the previous formula, you first build $\mathbf{A}_{22}^{-1}$, then invert $\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}$.)

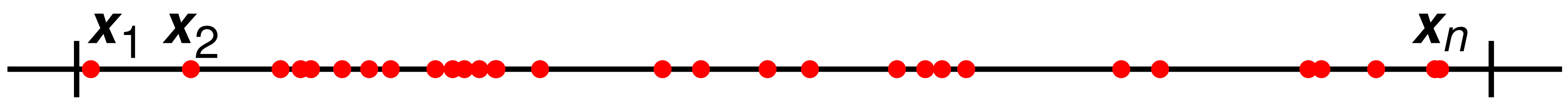
The “$\mathcal{S}$-matrix” format is very easy to use, but is not very efficient.

- Recursion tends to lead to simple formulas, but can be dicey to implement efficiently.
- The tree is traversed up and down many times.
- The off-diagonal blocks can be very large, which means that even though they have rank k, it becomes expensive to store and manipulate the factors.

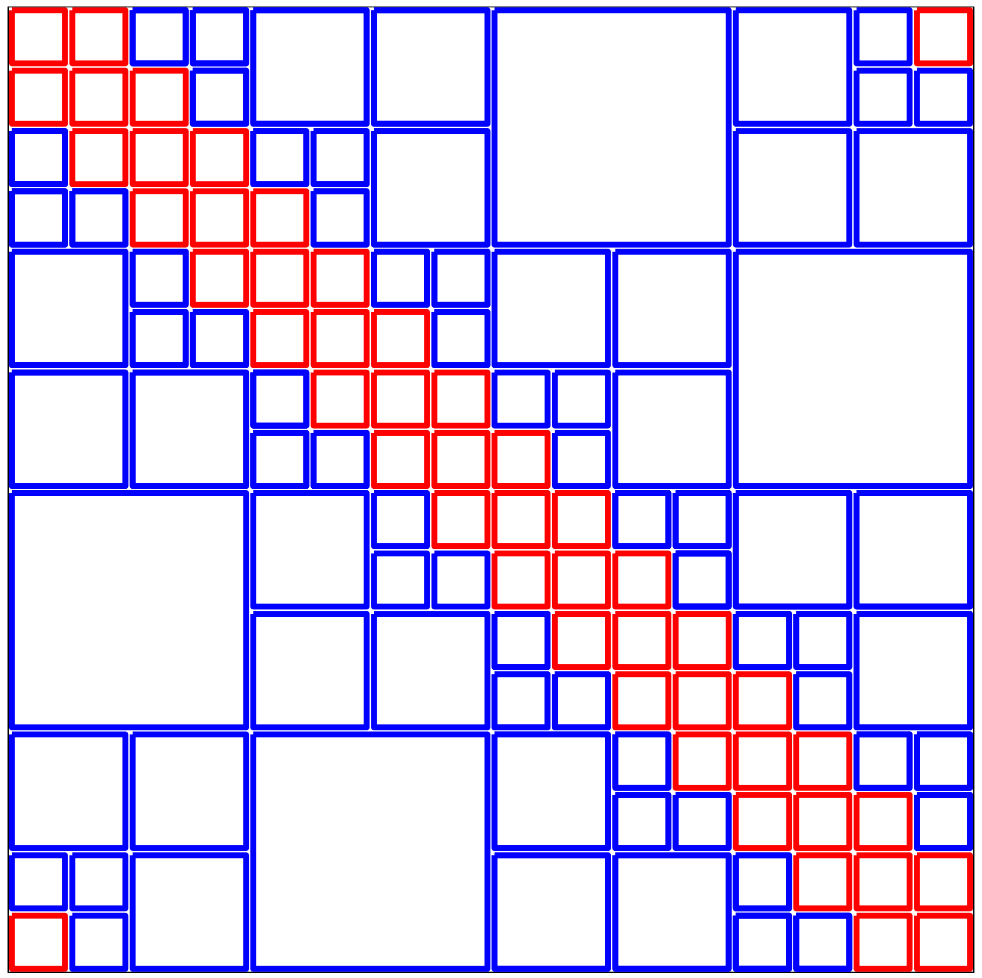
We will in Lecture 8 describe a more efficient format, called *Hierarchically Block Separable (HBS)* matrices (or “Hierarchically Semi-Separable (HSS)” matrices).

Question: Can you invert a matrix in the "FMM format"?

This is a little more complicated. The main problem is that the "tessellation pattern" is different. For instance, consider a set of n point charges along a line:



Then the matrix $\mathbf{A}$ with entries $\mathbf{A}(i,j) = \log|\boldsymbol{x}_i - \boldsymbol{x}_j|$ would be tessellated as



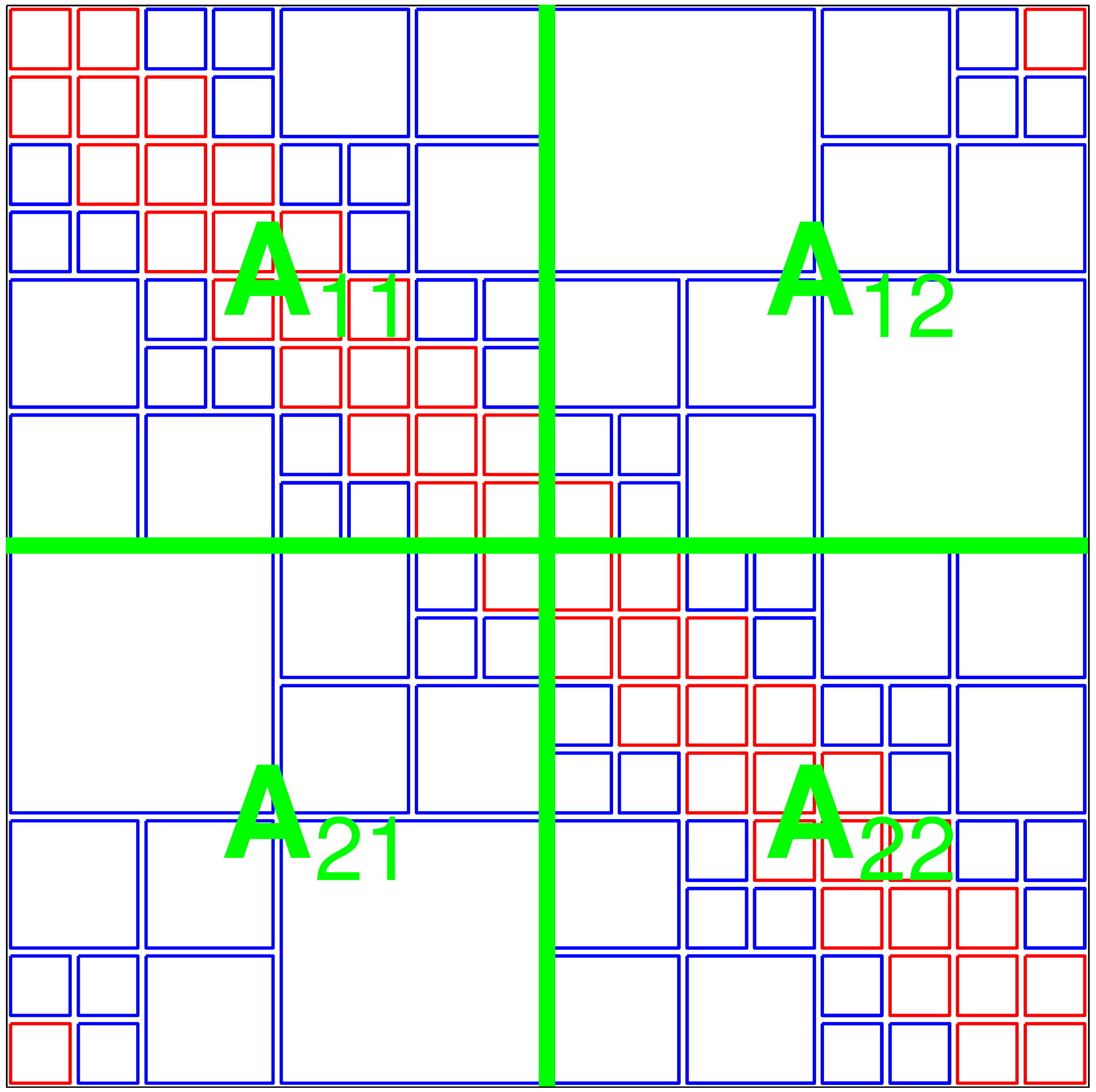
Red blocks are represented as dense matrices. Blue blocks are low rank.

Note how all low-rank blocks are "well-separated" from the diagonal.

Now suppose that we want to invert the FMM matrix using the formula

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1} \end{bmatrix} + \begin{bmatrix} \mathbf{A}_{11}^{-1}\mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1}\mathbf{U}_1 & \tilde{\mathbf{A}}_{12} \\ \tilde{\mathbf{A}}_{21} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1}\mathbf{U}_2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{V}_1^*\mathbf{A}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^*\mathbf{A}_{22}^{-1} \end{bmatrix}.$$

Well, look at what happens to the partitioning:



The matrices $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ now have much more complicated structure.

You *can* still invert it, but it is harder — will return to this point.

The key question here is *buffering* — do you compress directly touching blocks or not?

Schemes like the FMM, $\mathcal{H}$-matrices, etc., do use buffering. Advantages include:

- Lower ranks — sometimes *far* lower.
- Much easier to construct representations — can use smoothness, analytic expansions, etc.

Most of the direct solvers described in this lecture (based on $\mathcal{S}$-matrices, HBS matrices, etc), do not use buffering.

- Higher ranks — dense volume problems in 3D tend to get prohibitive.
- Harder to construct compressed representations.
- Much easier to use data-structures.

We will return to the question of buffering throughout the lectures.