Efficient local low-rank updates for $\mathcal{H}^2$-Matrices

Steffen Börm
(with Knut Reimer)
funded by DFG grant BO 3289-4/1

Christian-Albrechts-Universität zu Kiel

INS Bonn, 15th of November, 2013

Data-sparse representations

Challenge: Non-local operators appear naturally in many applications.

- Solution operators of partial differential equations.
- Boundary integral method.
- Population dynamics.
- Control theory.

Discretization schemes lead to highly non-sparse matrices.

Data-sparse representations

Challenge: Non-local operators appear naturally in many applications.

- Solution operators of partial differential equations.
- Boundary integral method.
- Population dynamics.
- Control theory.

Discretization schemes lead to highly non-sparse matrices.

Idea: Represent these matrices in a data-sparse representation, i.e., by $\mathcal{O}(n \log^{\alpha}(n))$ coefficients instead of n^2.

Examples: Panel clustering, multipole expansion, wavelets, hierarchical matrices.

Algebraic operations

Problem: Using a data-sparse representation severely restricts the number of operations that can be carried out efficiently.

Goal: Find a representation and matching efficient algorithms for

- evaluating matrix-vector products,
- computing the inverse,
- finding an LR or Cholesky factorization and
- constructing good preconditioners for Krylov methods.

Algebraic operations

Problem: Using a data-sparse representation severely restricts the number of operations that can be carried out efficiently.

Goal: Find a representation and matching efficient algorithms for

- evaluating matrix-vector products,
- computing the inverse,
- finding an LR or Cholesky factorization and
- constructing good preconditioners for Krylov methods.

Hierarchical matrices can be used to solve all of these problems, but still require a large amount of storage.
$\rightarrow$ Hackbusch 1999, Hackbusch/Khoromskij 2000, Hackbusch/Grasedyck 2003

Algebraic operations

Problem: Using a data-sparse representation severely restricts the number of operations that can be carried out efficiently.

Goal: Find a representation and matching efficient algorithms for

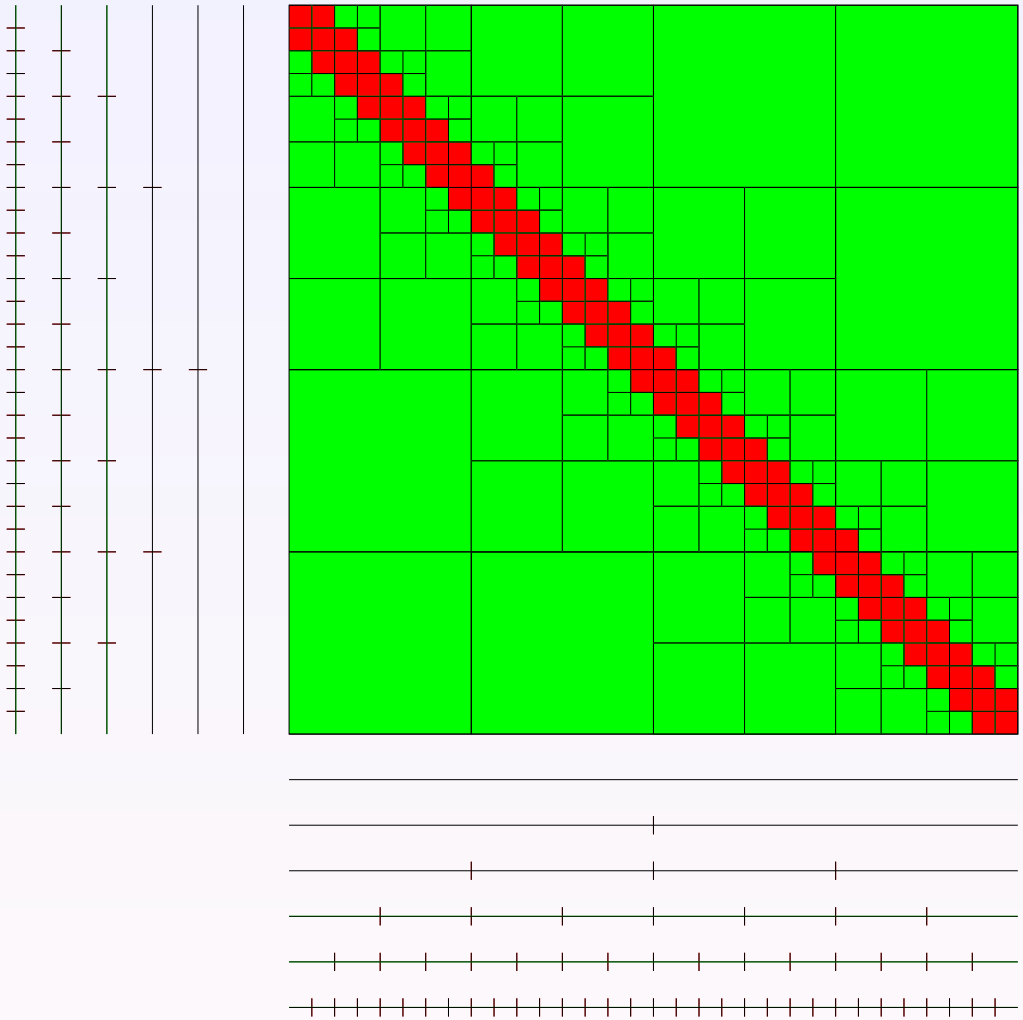
- evaluating matrix-vector products,
- computing the inverse,
- finding an LR or Cholesky factorization and
- constructing good preconditioners for Krylov methods.

Hierarchical matrices can be used to solve all of these problems, but still require a large amount of storage.
$\rightarrow$ Hackbusch 1999, Hackbusch/Khoromskij 2000, Hackbusch/Grasedyck 2003

Question: Can we do better?

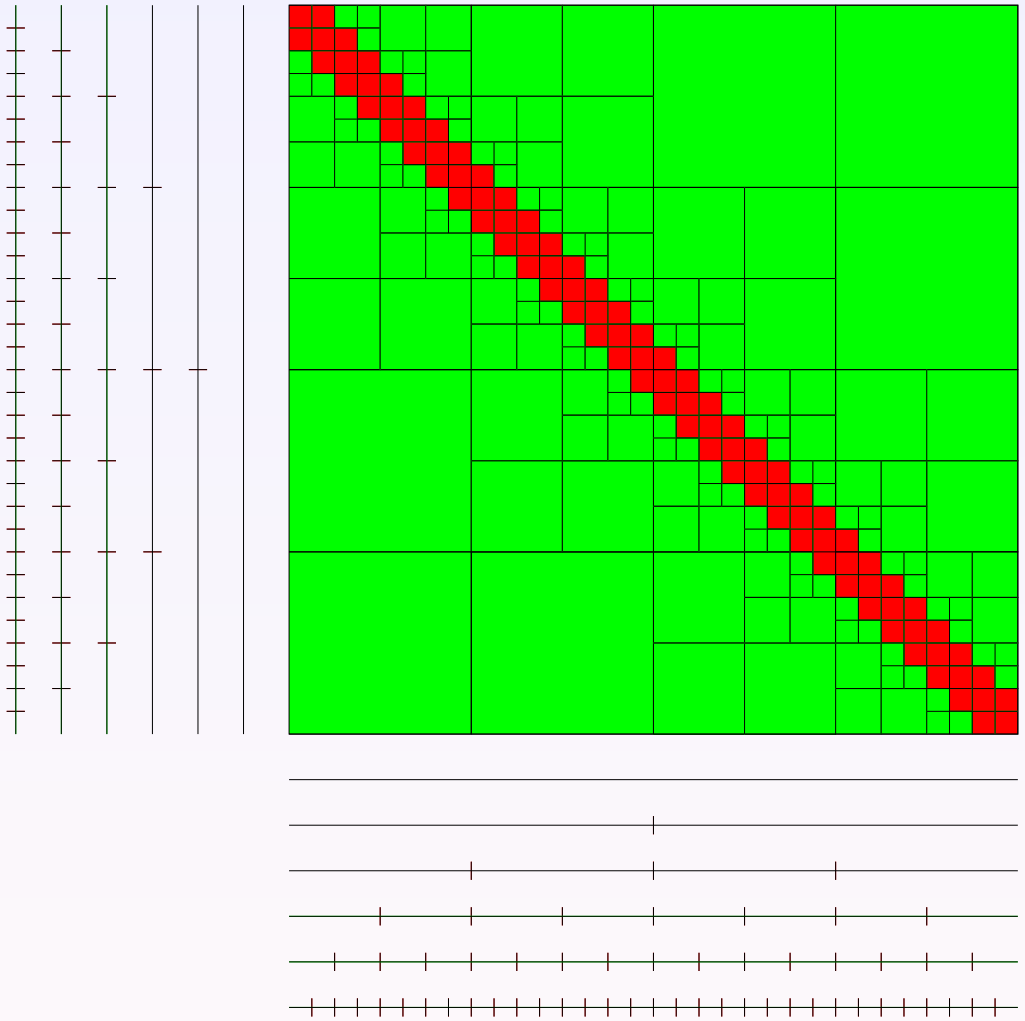
Overview

Hierarchical matrix



Matrix split into blocks $t \times s$ consisting of clusters t, s.

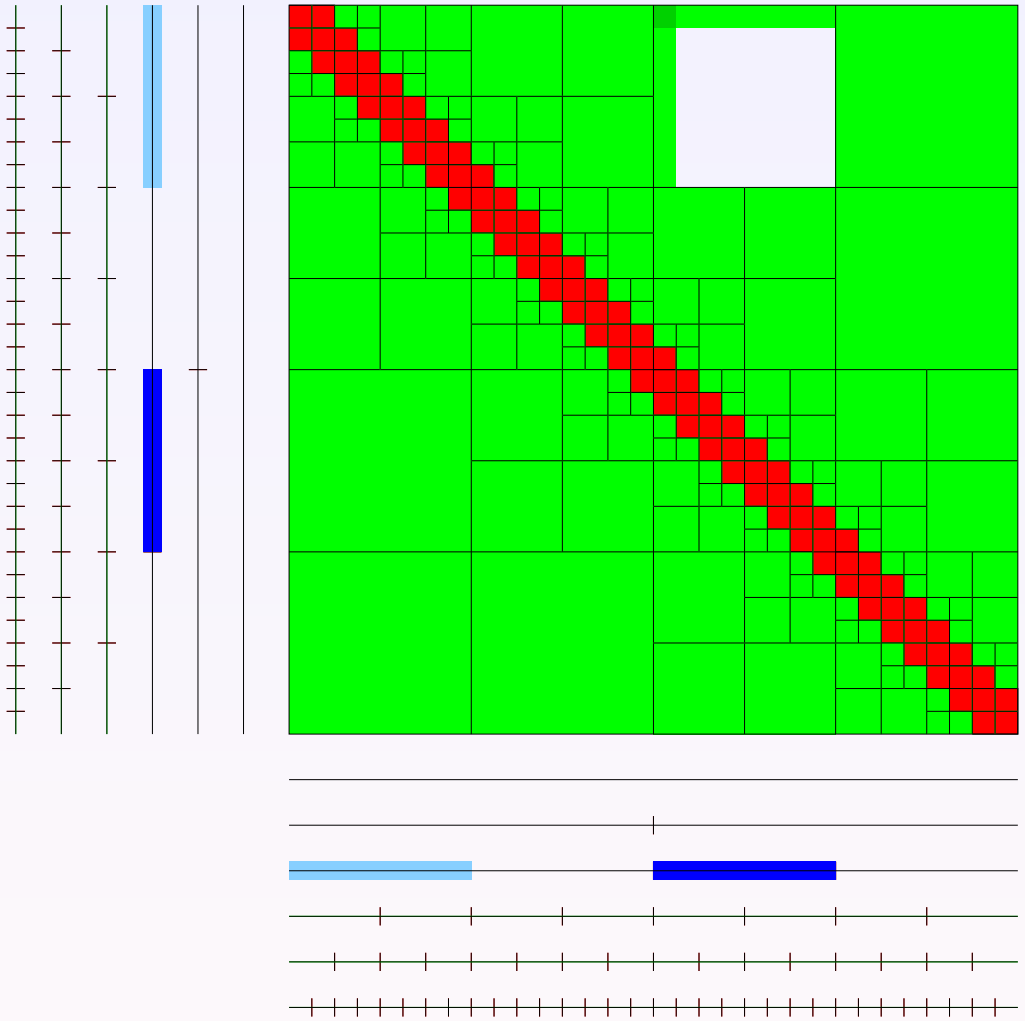
Hierarchical matrix



Matrix split into blocks $t \times s$ consisting of clusters t, s.

Clusters chosen from a hierarchical cluster tree.

Hierarchical matrix



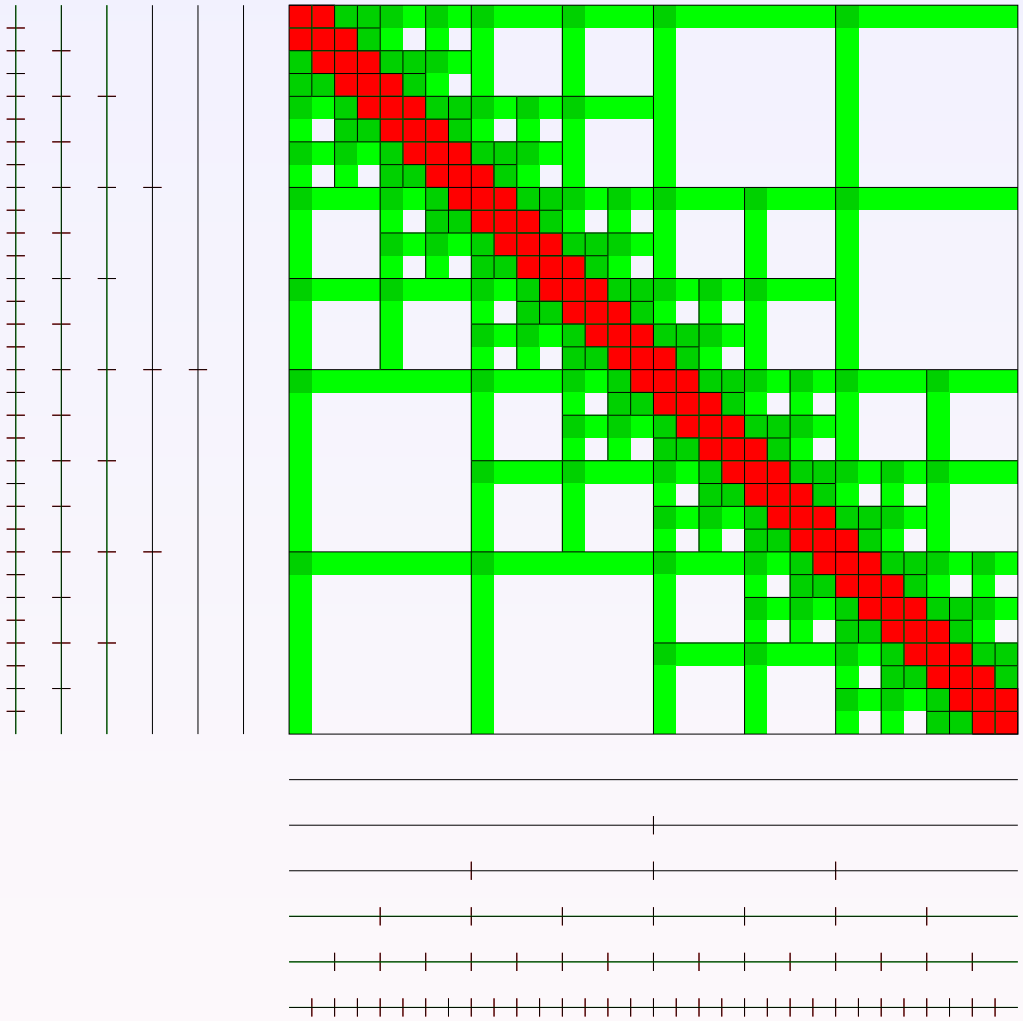
Matrix split into blocks $t \times s$ consisting of clusters t, s.

Clusters chosen from a hierarchical cluster tree.

Blocks represented by low-rank factorizations

$$G|_{t\times s} \approx AB^*.$$

Hierarchical matrix



Matrix split into blocks $t \times s$ consisting of clusters t, s.

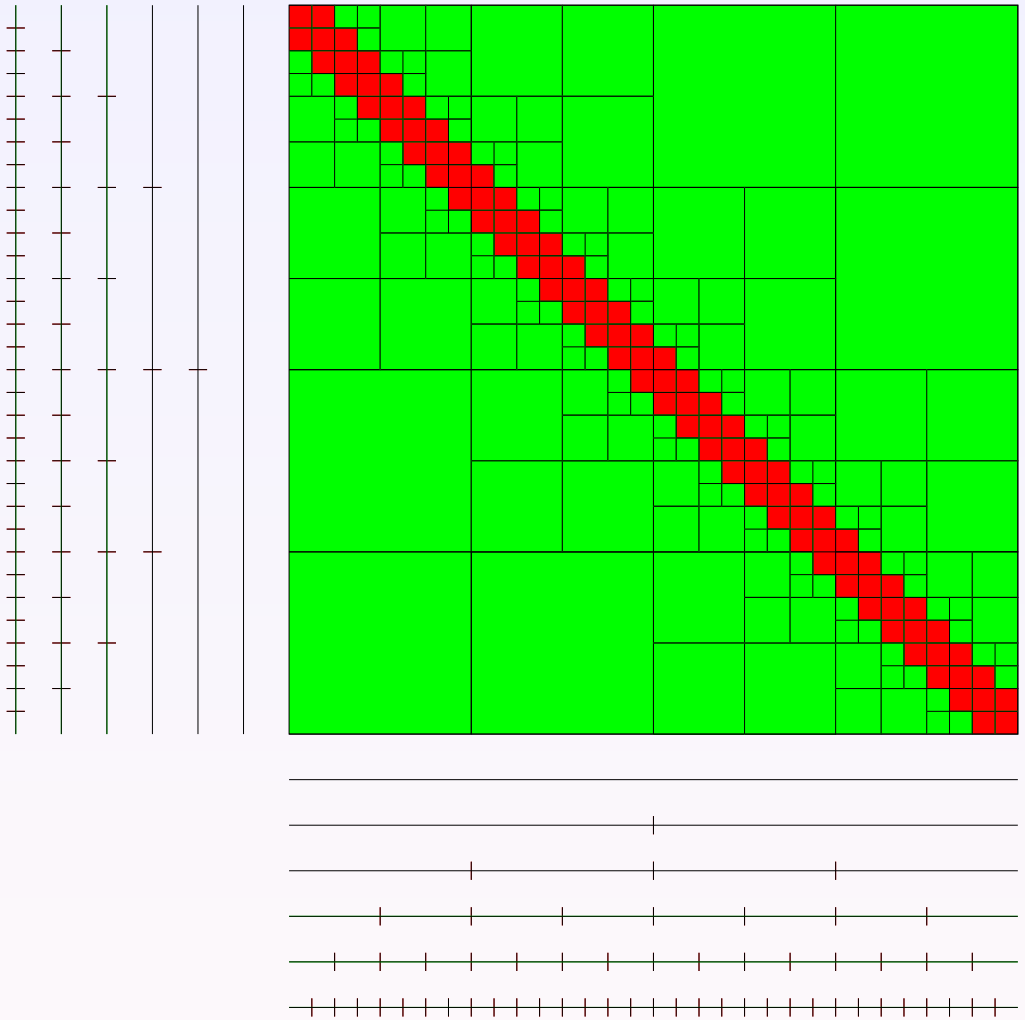
Clusters chosen from a hierarchical cluster tree.

Blocks represented by low-rank factorizations

$$G|_{t\times s} \approx AB^*.$$

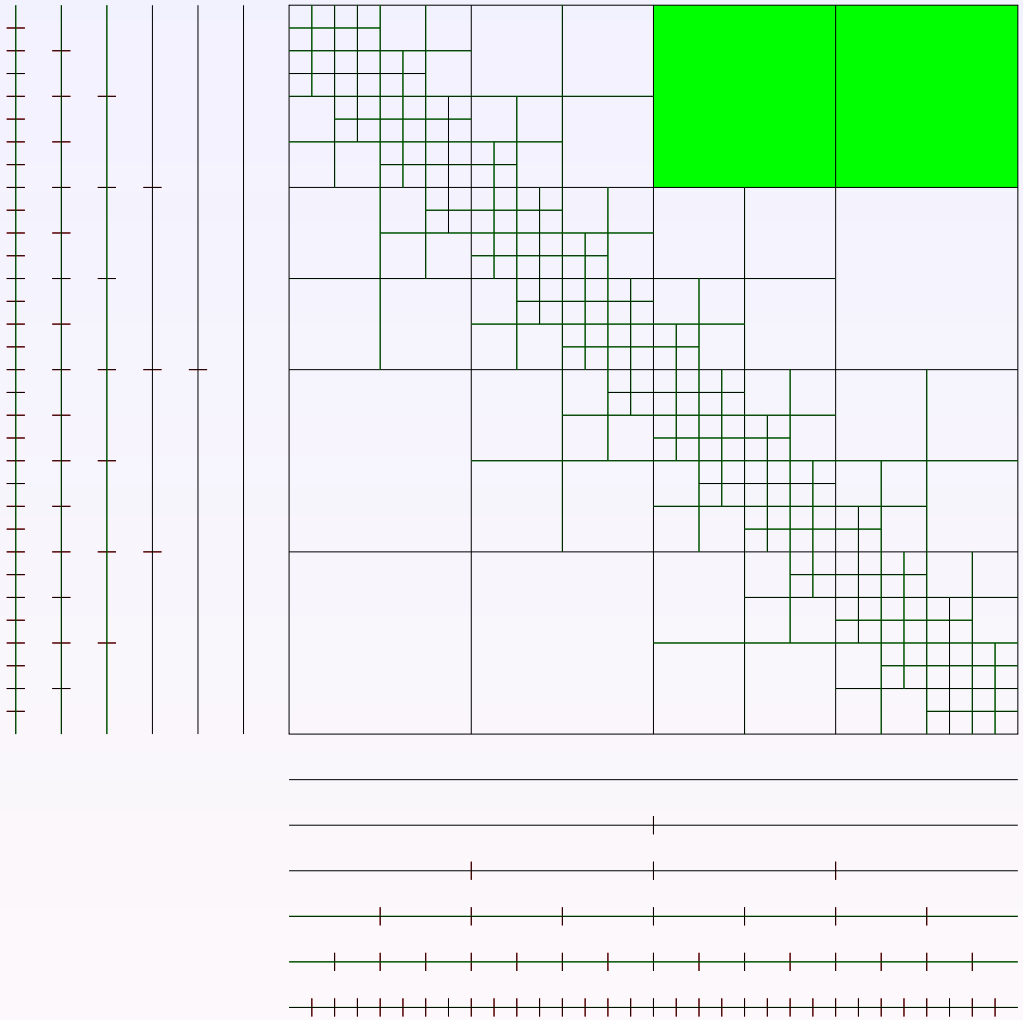
Result: $\mathcal{O}(nk \log n)$ storage, $\mathcal{O}(nk^\alpha \log^\beta n)$ runtime.

Uniform hierarchical matrix



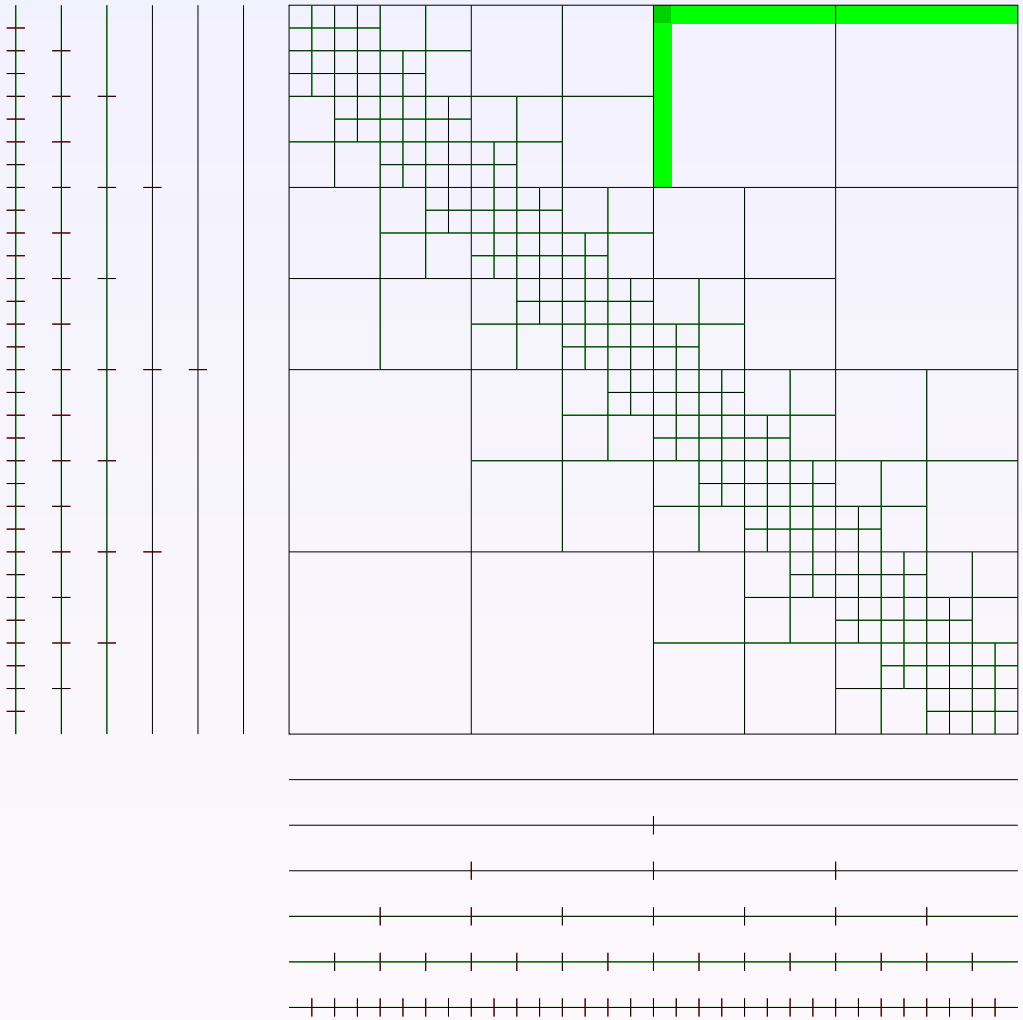
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



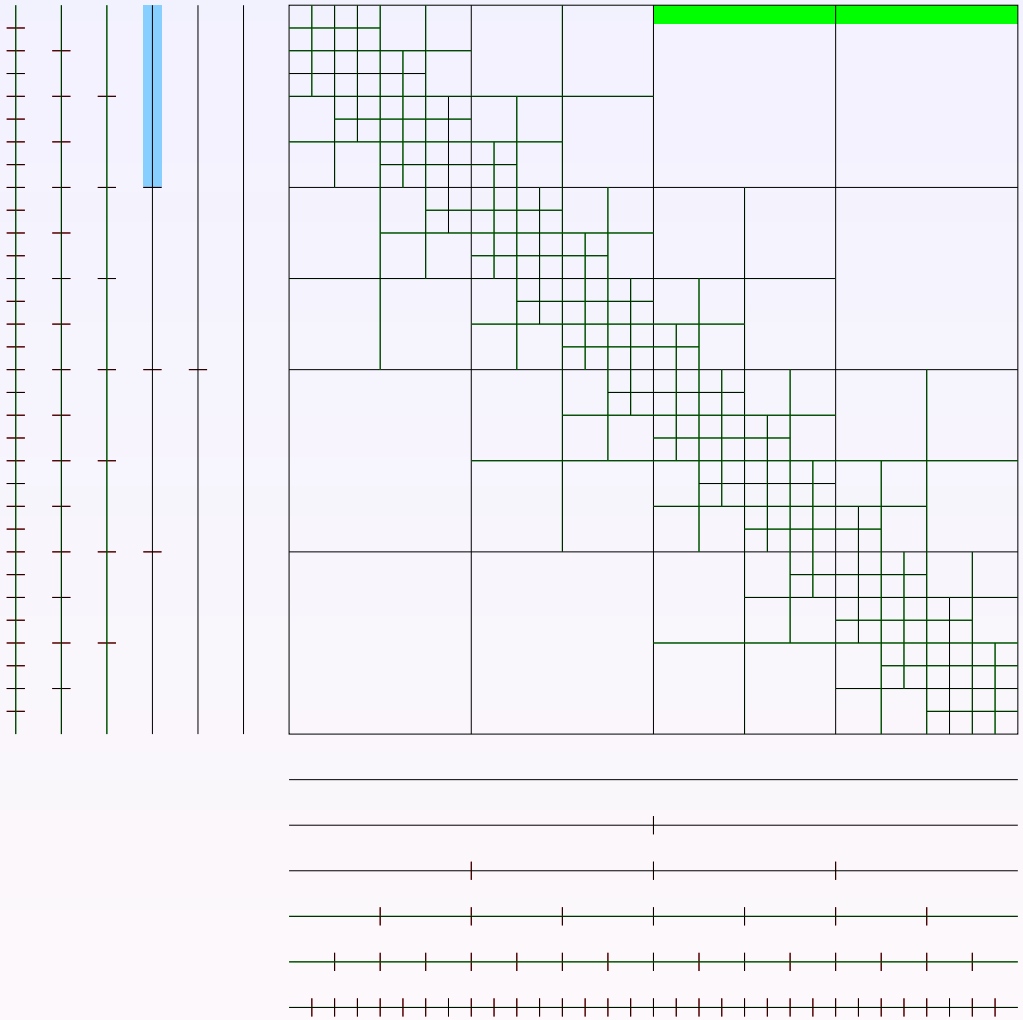
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



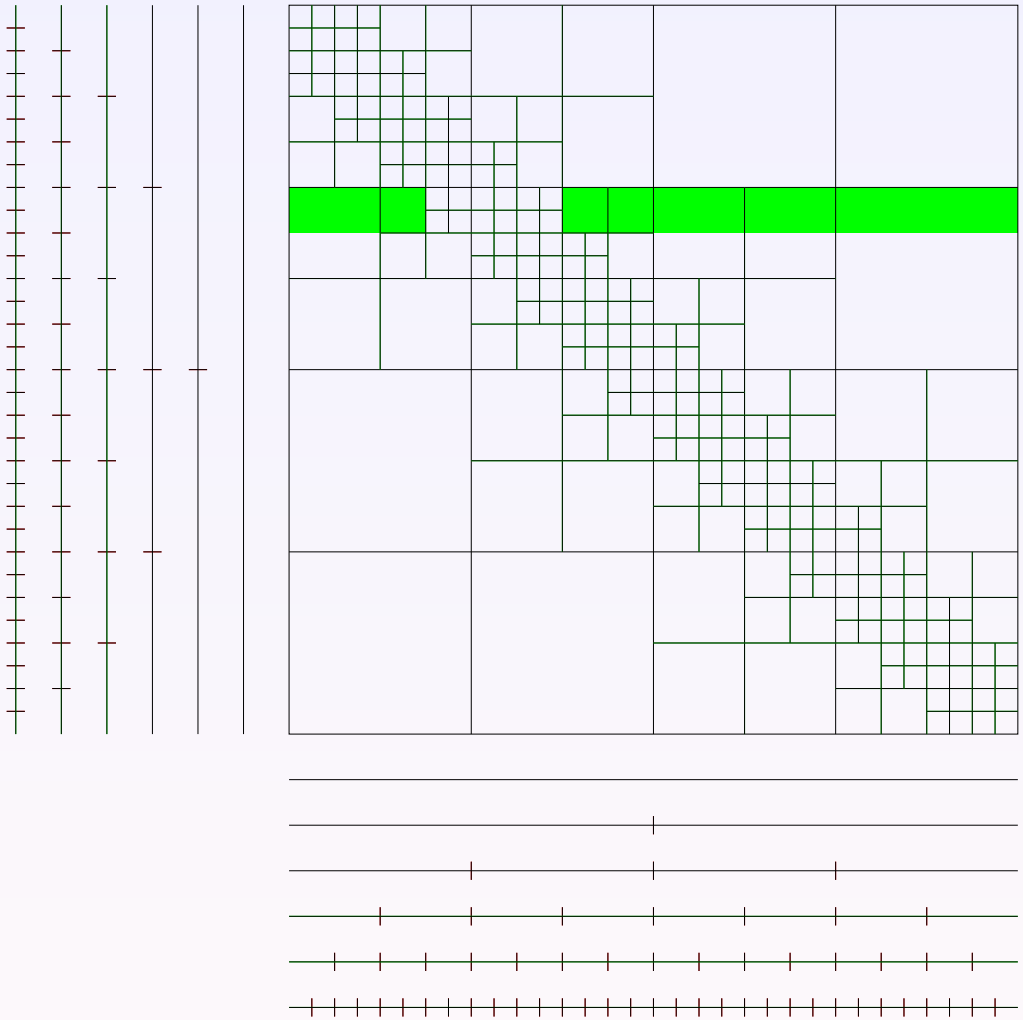
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



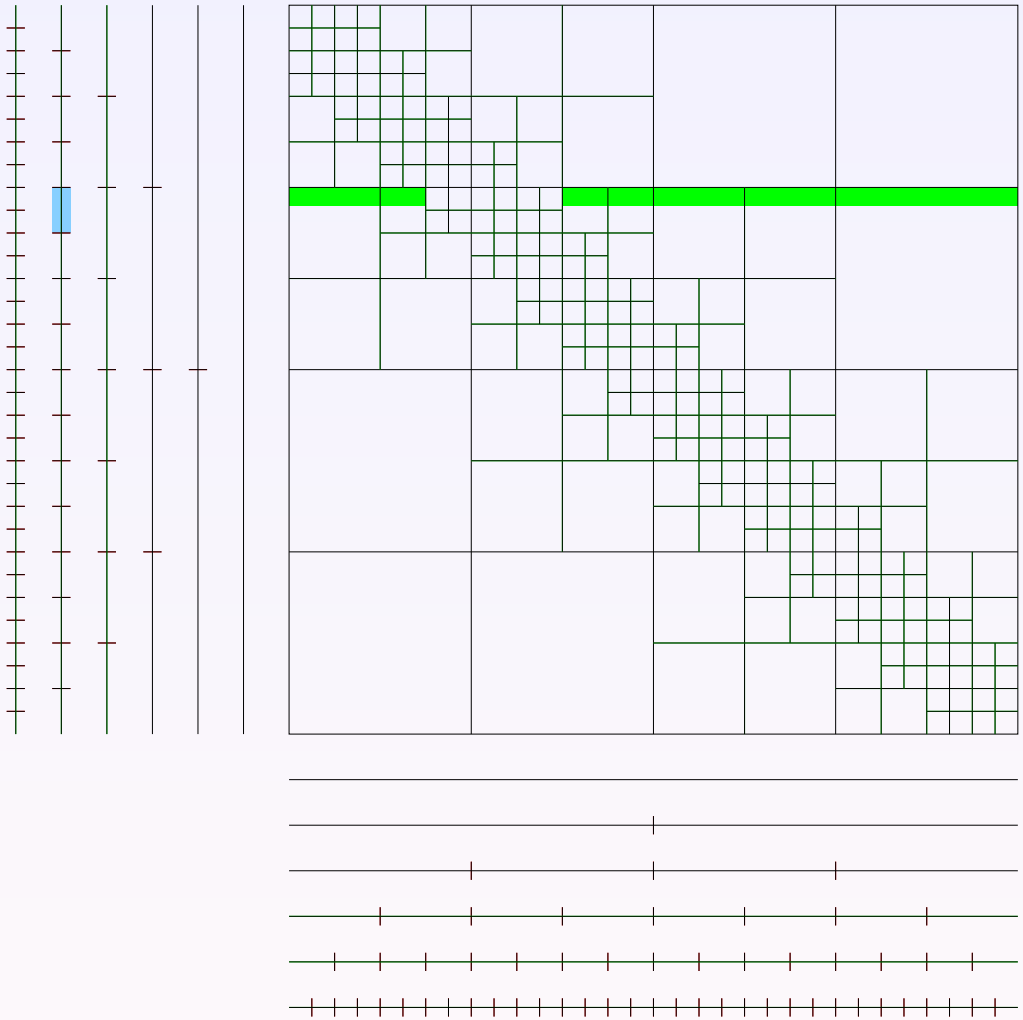
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



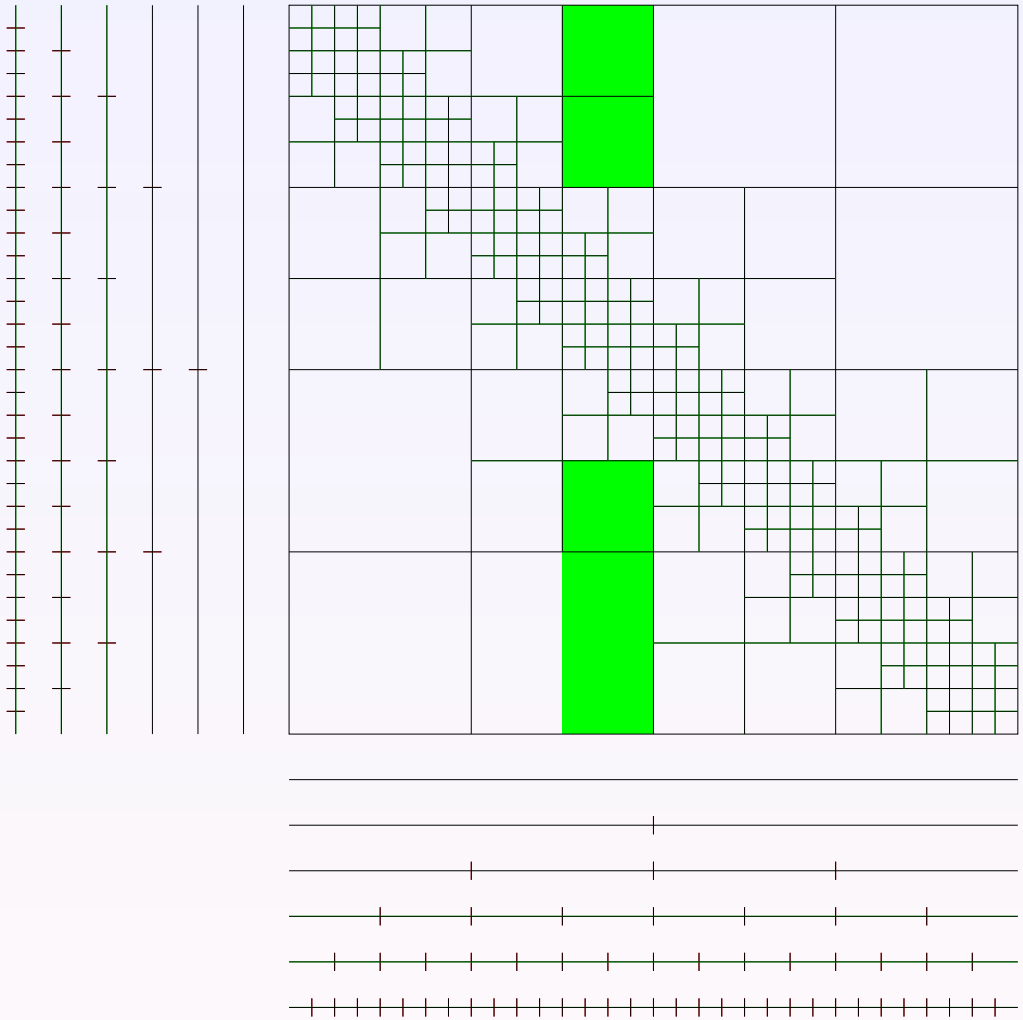
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



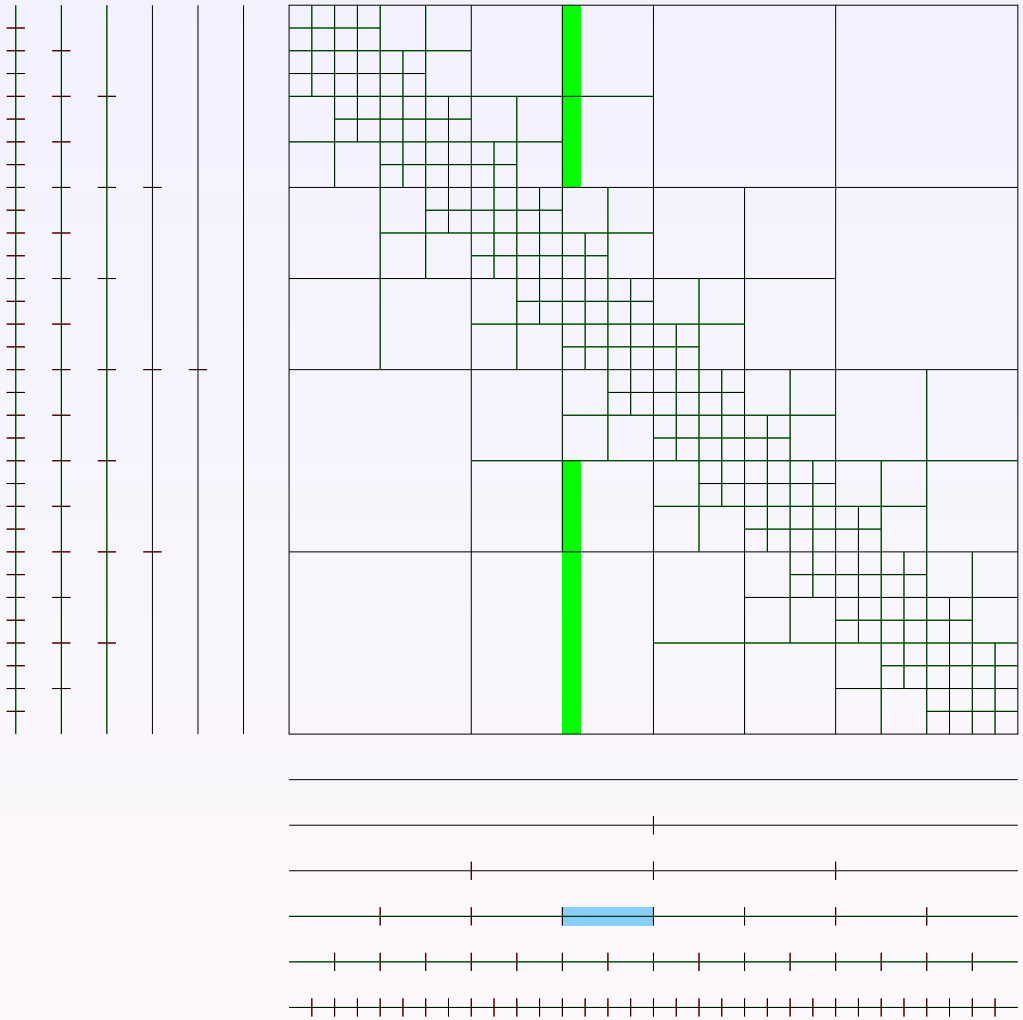
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



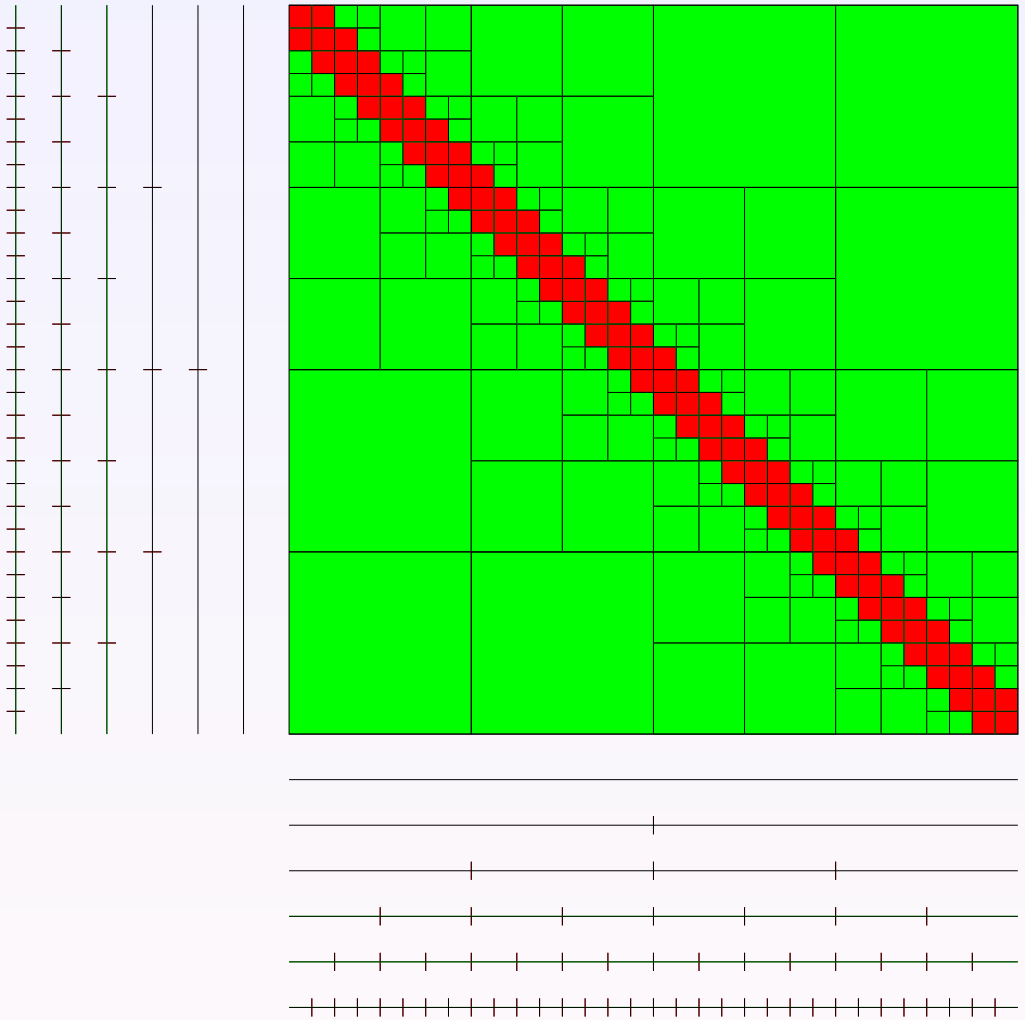
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix



Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Uniform hierarchical matrix

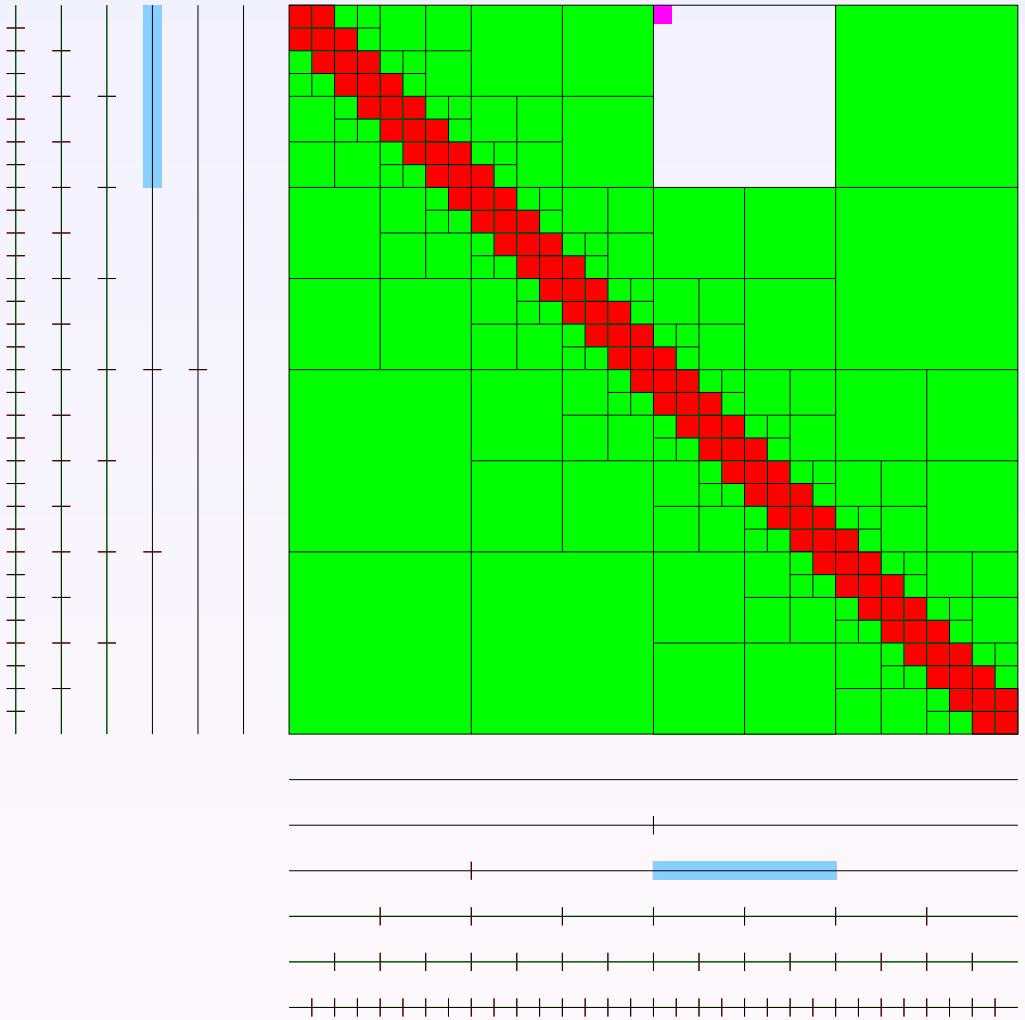


Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Idea: Use three factors.

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*.$$

Uniform hierarchical matrix

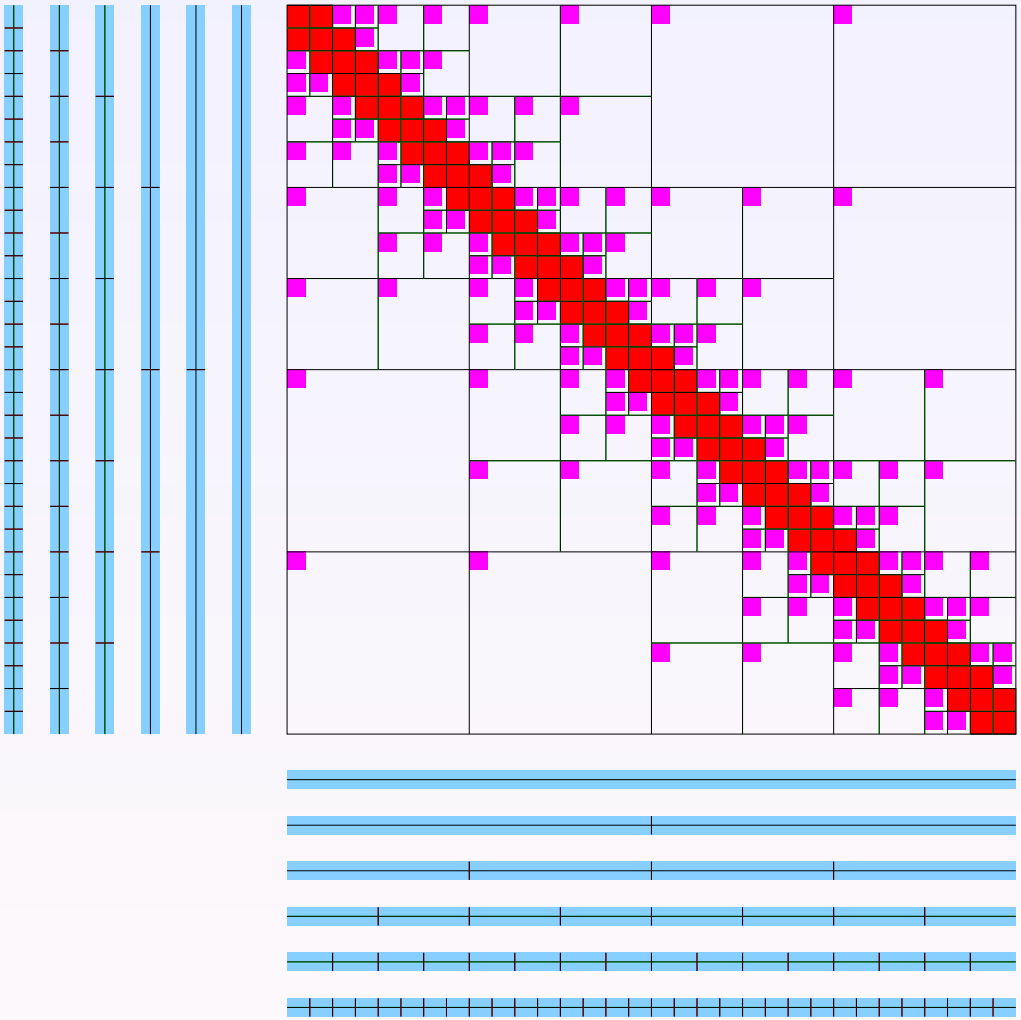


Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

Idea: Use three factors.

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*.$$

Uniform hierarchical matrix



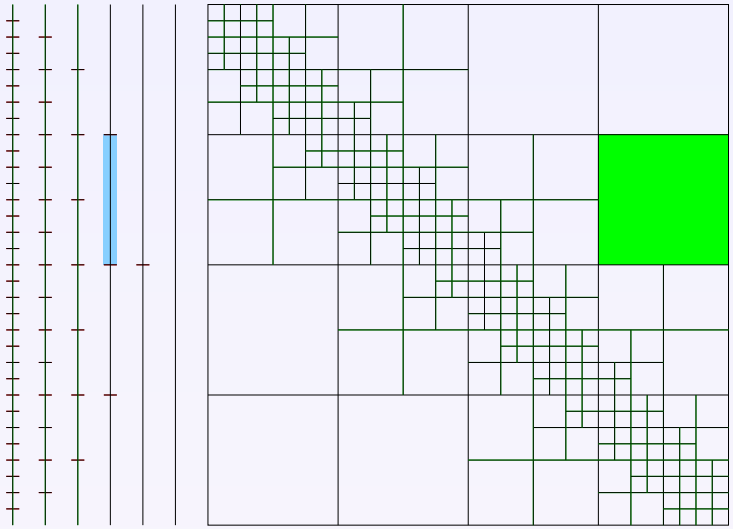
Observation: As long as we stay away from the diagonal, we find approximations for large blocks.

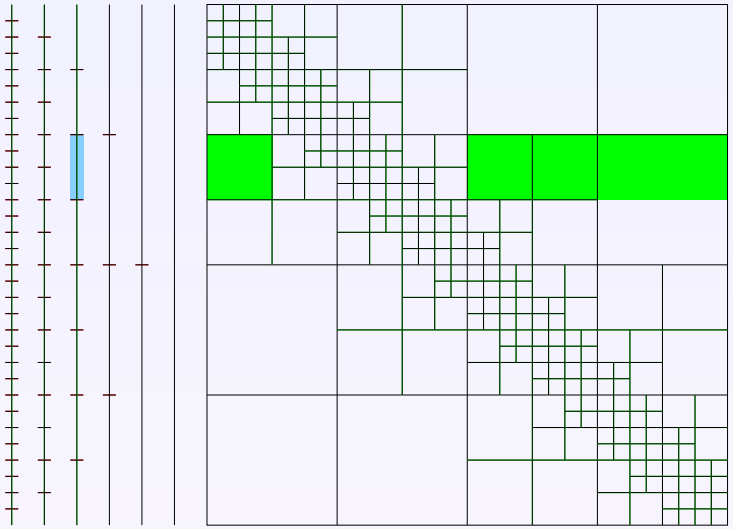
Idea: Use three factors.

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*.$$

Result: Only $\mathcal{O}(nk)$ storage for coupling matrices $S_{t,s}$, but still $\mathcal{O}(nk \log n)$ for cluster bases V_t and W_s.

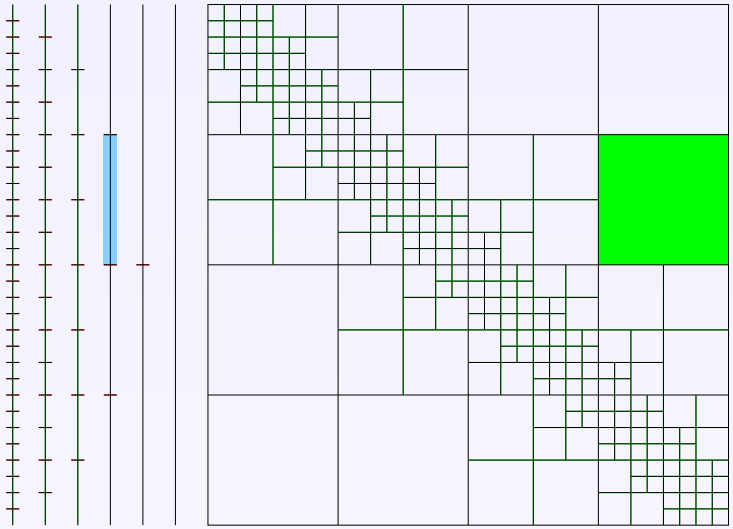
Nested cluster basis

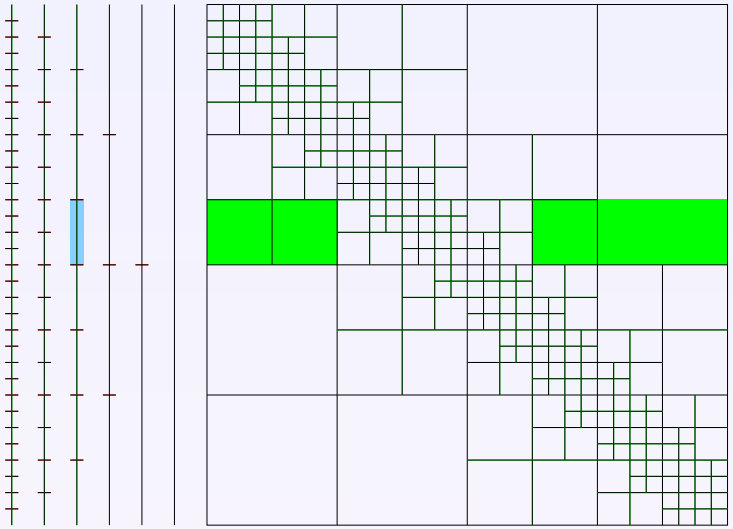




Observation: Cluster bases of sons approximate more blocks than their father.

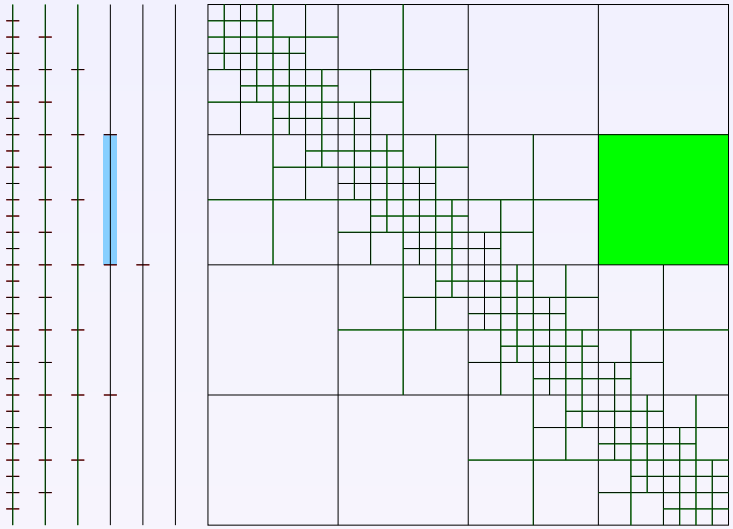
Nested cluster basis

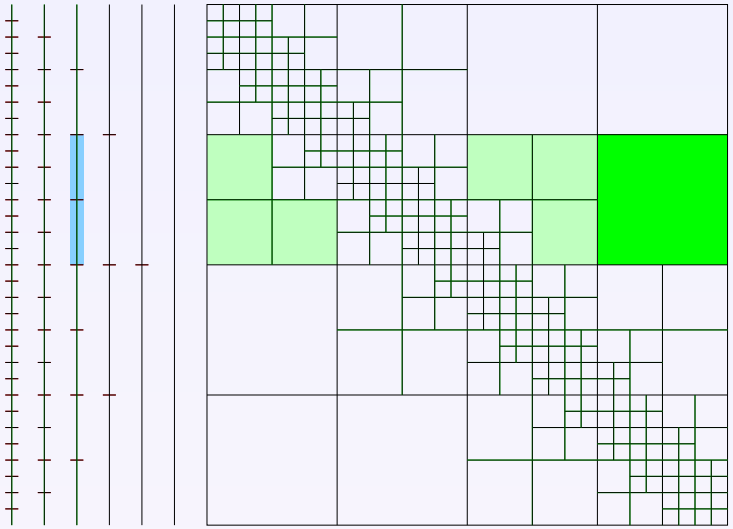




Observation: Cluster bases of sons approximate more blocks than their father.

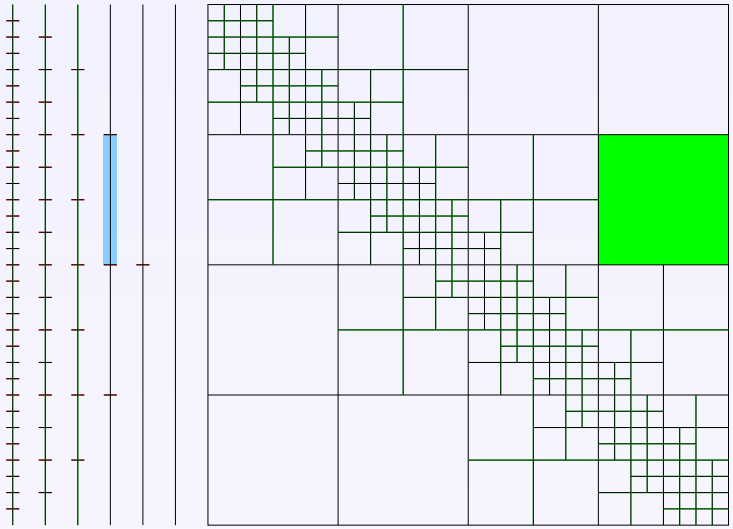
Nested cluster basis

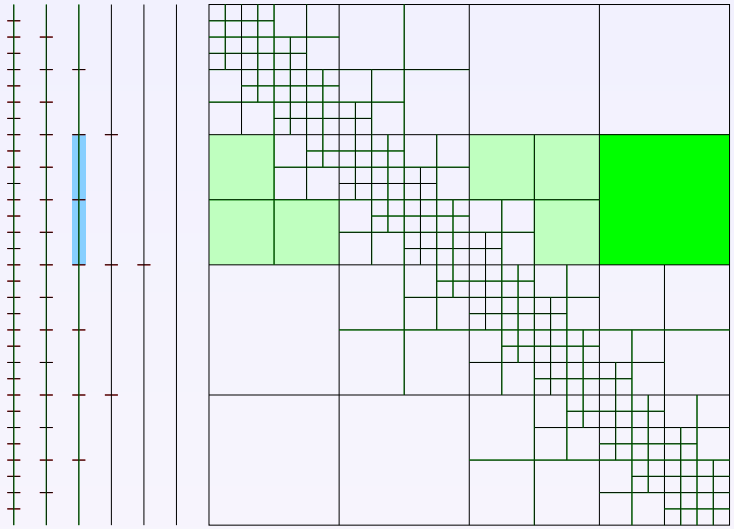




Observation: Cluster bases of sons approximate more blocks than their father. $\rightarrow$ Should also approximate father's cluster basis.

Nested cluster basis

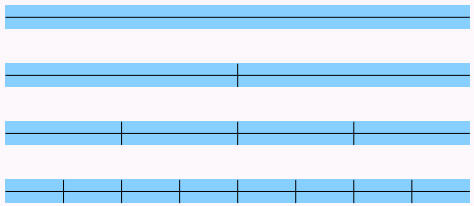




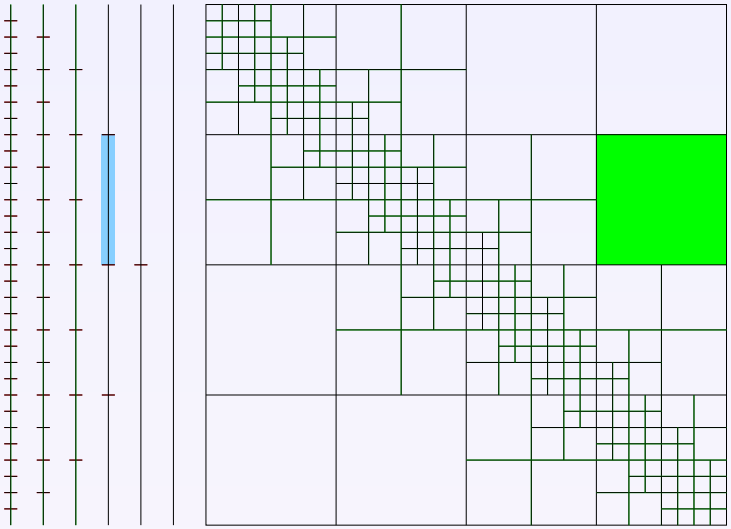
Observation: Cluster bases of sons approximate more blocks than their father. $\rightarrow$ Should also approximate father's cluster basis.

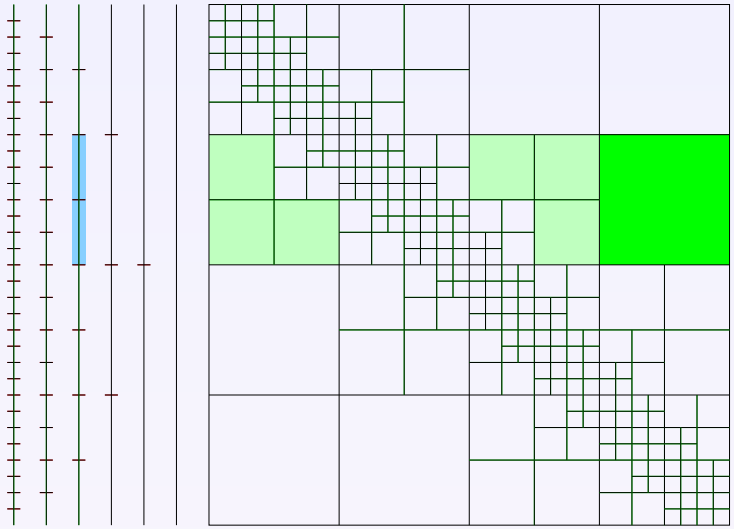
Idea: Represent V_t by son's matrices and small transfer matrices.

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$



Nested cluster basis

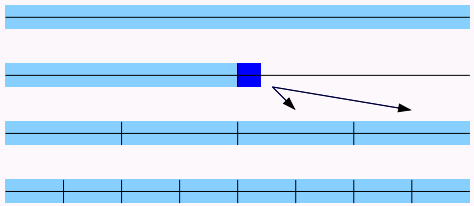




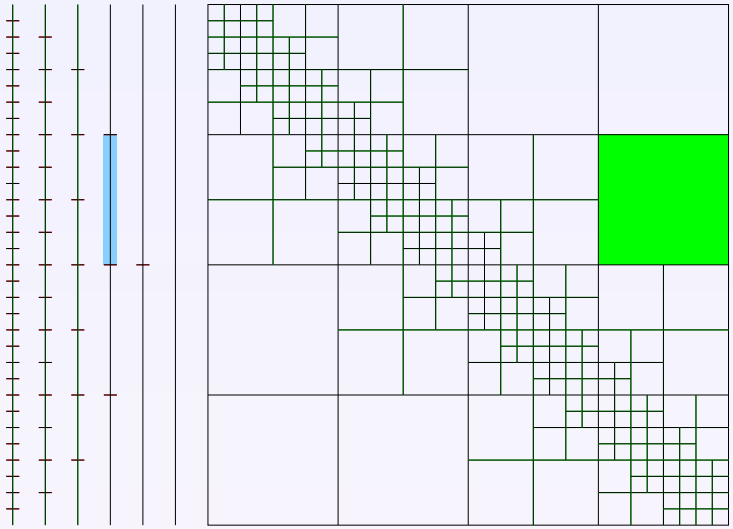
Observation: Cluster bases of sons approximate more blocks than their father. $\rightarrow$ Should also approximate father's cluster basis.

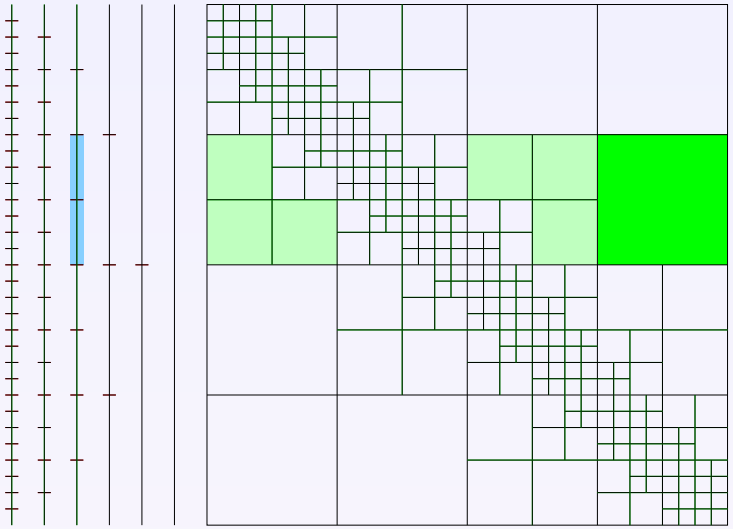
Idea: Represent V_t by son's matrices and small transfer matrices.

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$



Nested cluster basis

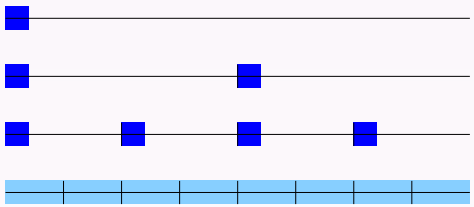




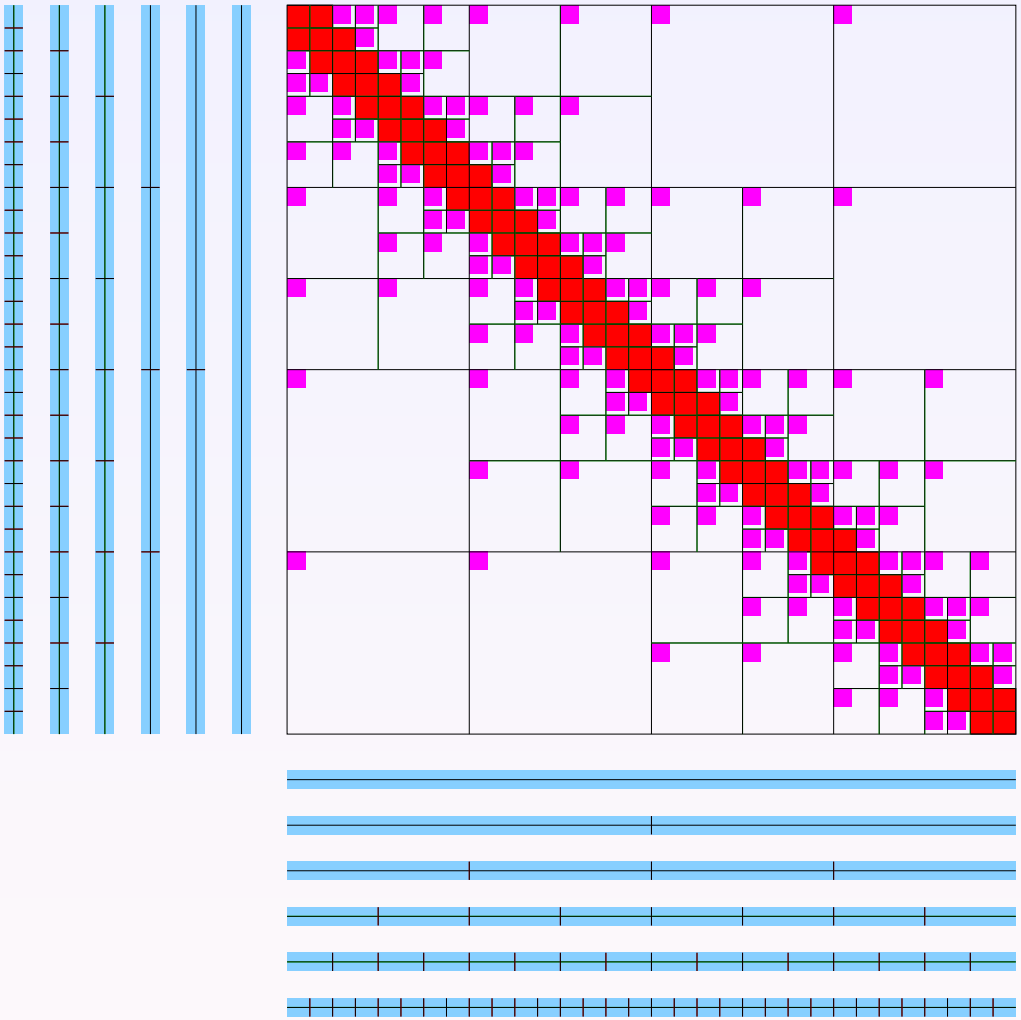
Observation: Cluster bases of sons approximate more blocks than their father. $\rightarrow$ Should also approximate father's cluster basis.

Idea: Represent V_t by son's matrices and small transfer matrices.

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$



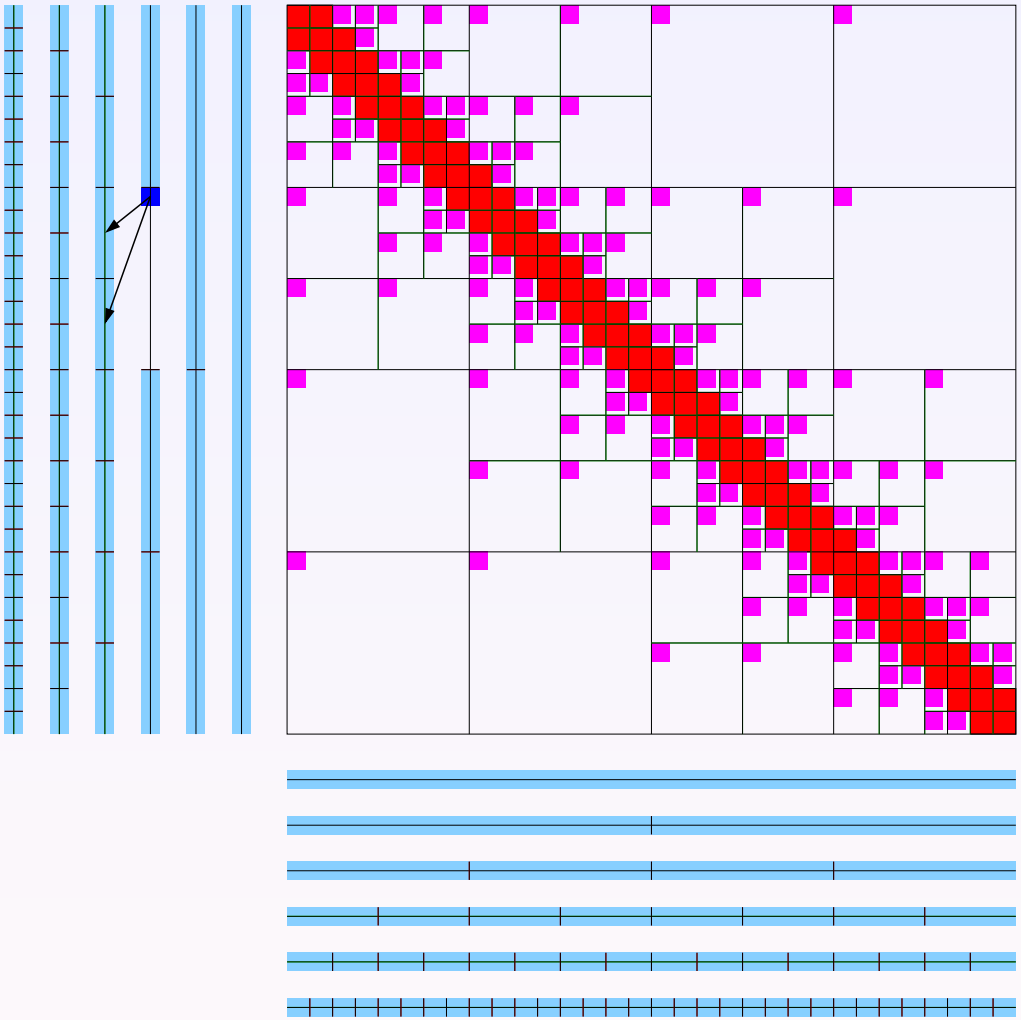
$\mathcal{H}^2$-matrix



Blocks approximated by

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*$$

$\mathcal{H}^2$-matrix



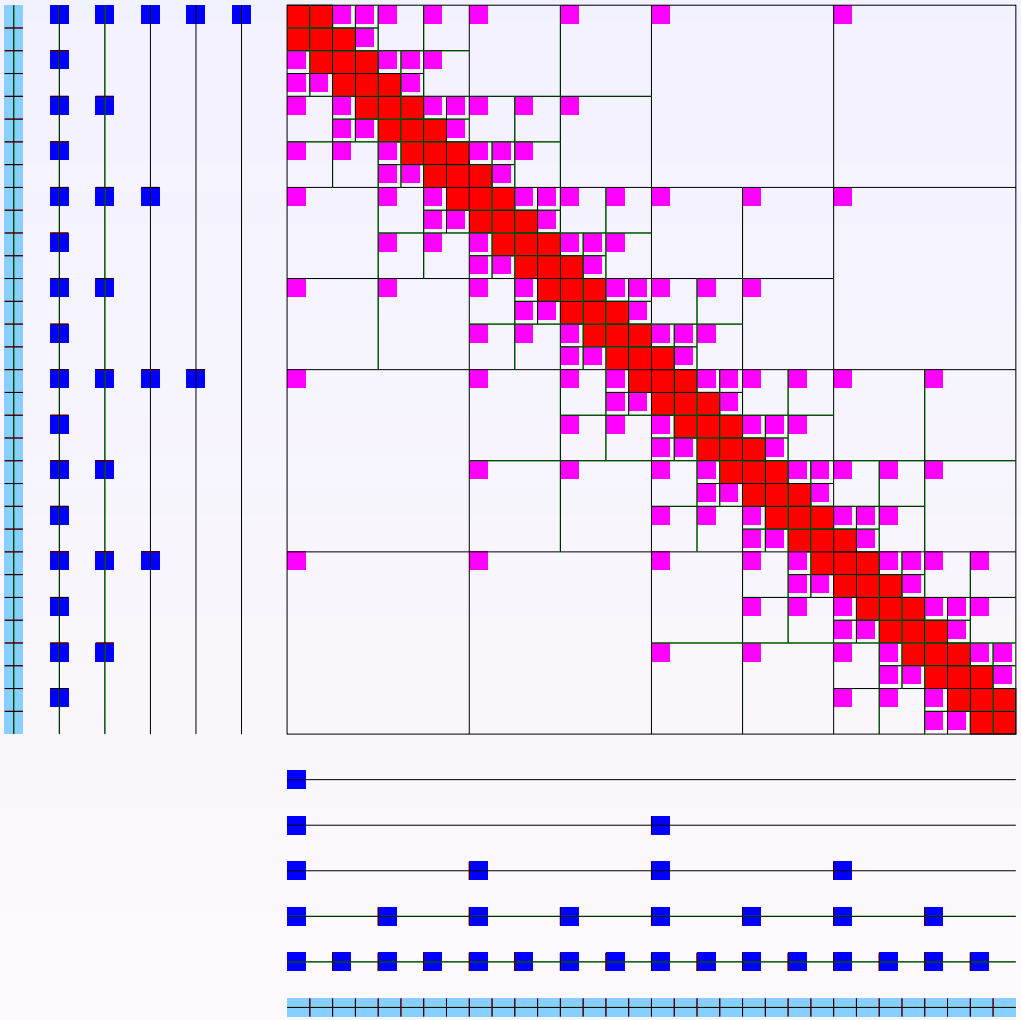
Blocks approximated by

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*$$

Cluster basis nested:

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$

$\mathcal{H}^2$-matrix



Blocks approximated by

$$G|_{t\times s} \approx V_t S_{t,s} W_s^*$$

Cluster basis nested:

$$V_t = \begin{pmatrix} V_{t_1} E_{t_1} \\ V_{t_2} E_{t_2} \end{pmatrix}$$

Result: Complexity $\mathcal{O}(nk)$

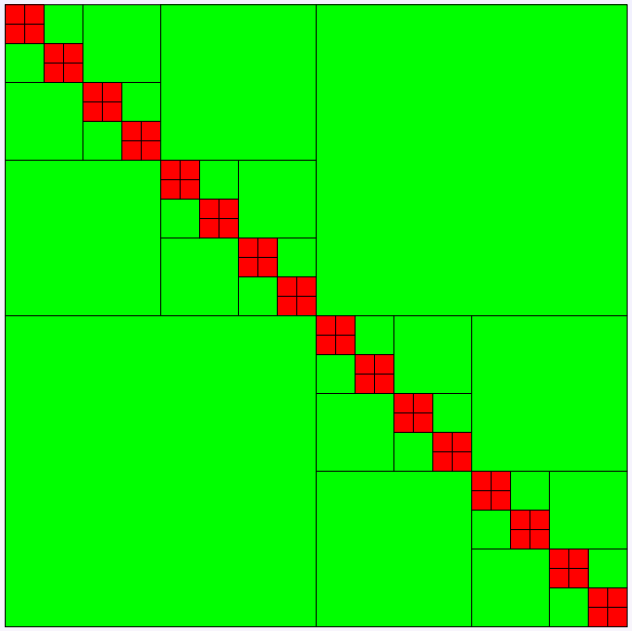
Prior work

Integral operators: Discretization of integral operators with asymptotically smooth kernel function leads immediately to $\mathcal{H}^2$-matrices.
$\rightarrow$ Hackbusch/Khoromskij/Sauter 2000, B./Hackbusch 2002, B./Löhndorf/Melenk 2005, Bebendorf/Venn 2012

Compression: Any $n \times n$ matrix G can be approximated by a quasi-optimal $\mathcal{H}^2$-matrix in $\mathcal{O}(n^2 k)$ operations.
Faster algorithm for $\mathcal{H}$-matrices takes $\mathcal{O}(nk^2 \log n)$ operations.
$\rightarrow$ B./Hackbusch 2002, B. 2005

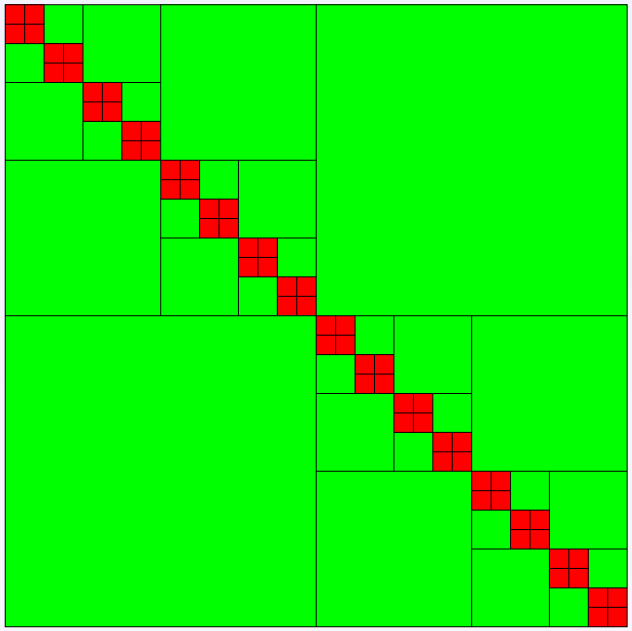
Algebraic operations: For fixed cluster bases, best approximation of the product of two $\mathcal{H}^2$-matrices can be computed in $\mathcal{O}(nk^2)$ operations.
$\rightarrow$ B. 2006

HSS-matrices



HSS-matrix: $\mathcal{H}^2$-matrix with very simple block structure.
$\rightarrow$ Eidelman/Gohberg 1999, Hackbusch/Khoromskij/Kriemann 2004, Chandrasekaran/Gu/Pals 2004, Chandrasekaran/Gu/Lyons 2005

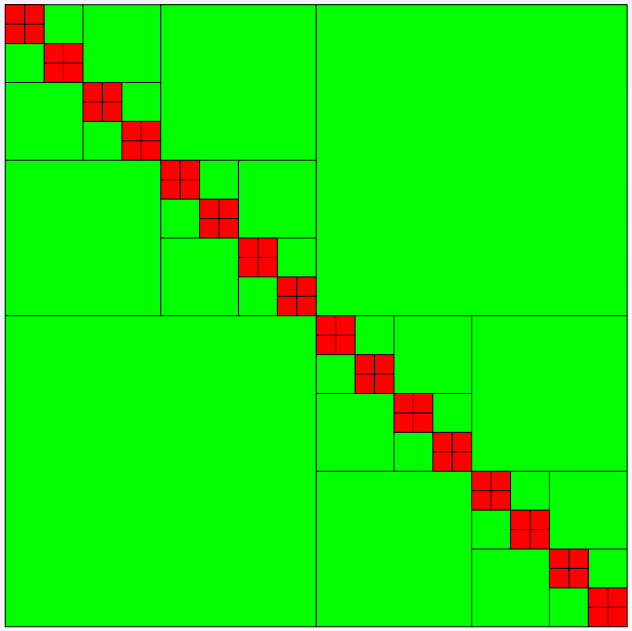
HSS-matrices

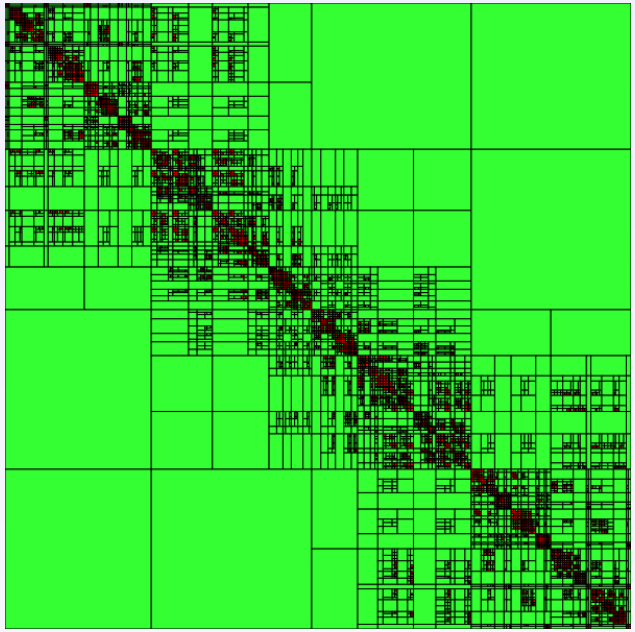


HSS-matrix: $\mathcal{H}^2$-matrix with very simple block structure.
$\rightarrow$ Eidelman/Gohberg 1999, Hackbusch/Khoromskij/Kriemann 2004, Chandrasekaran/Gu/Pals 2004, Chandrasekaran/Gu/Lyons 2005

Advantage: Many operations in $\mathcal{O}(nk^2)$, no approximation required.

HSS-matrices





HSS-matrix: $\mathcal{H}^2$-matrix with very simple block structure.
$\rightarrow$ Eidelman/Gohberg 1999, Hackbusch/Khoromskij/Kriemann 2004, Chandrasekaran/Gu/Pals 2004, Chandrasekaran/Gu/Lyons 2005

Advantage: Many operations in $\mathcal{O}(nk^2)$, no approximation required.

Disadvantage: Only appropriate for one-dimensional problems.

Overview

Algebraic compression

Goal: Turn any matrix into an $\mathcal{H}^2$-matrix with a given block structure and accuracy by a purely algebraic procedure.

Flexibility:

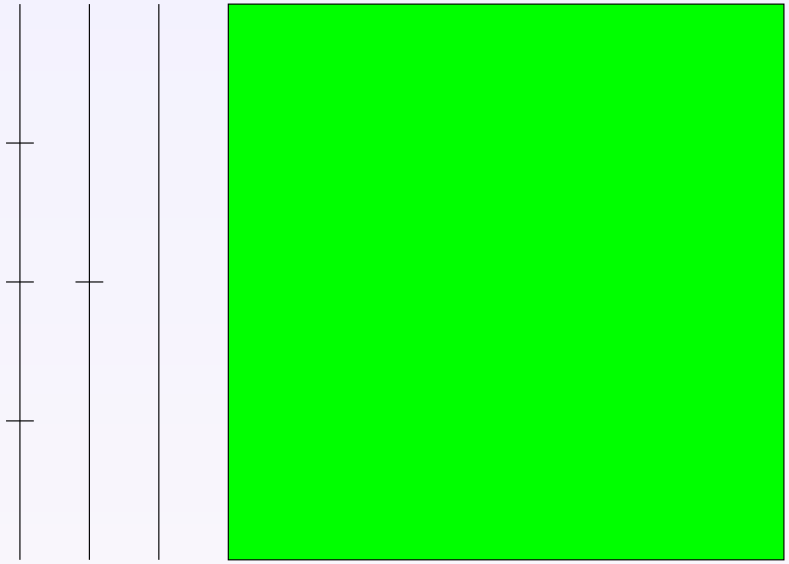
- Not limited to smooth kernel functions.
- Can be based on a variety of initial approximations. (interpolation, ACA, HCA, multipole, ...)

Robustness:

- Finds (almost) the best possible approximation.
- Numerically stable, based on orthogonal decompositions.

One block

Goal: Construct row cluster basis for one block.



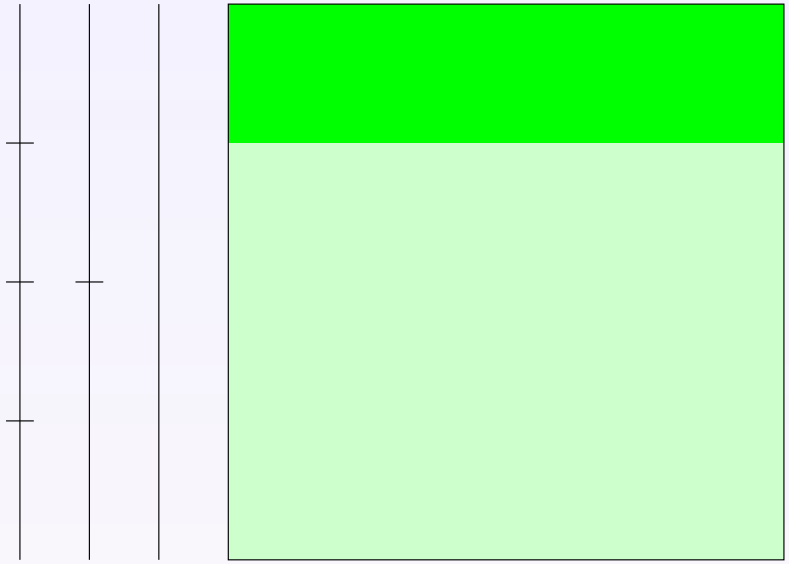
Leaves: Use SVD to find rank-k approximation

$$\widehat{G}_{t,s} := G|_{t\times s} \approx \widehat{V}_t B_{t,s}^*.$$

Matrix $\widehat{V}_t$ defines leaf matrix V_t.

One block

Goal: Construct row cluster basis for one block.



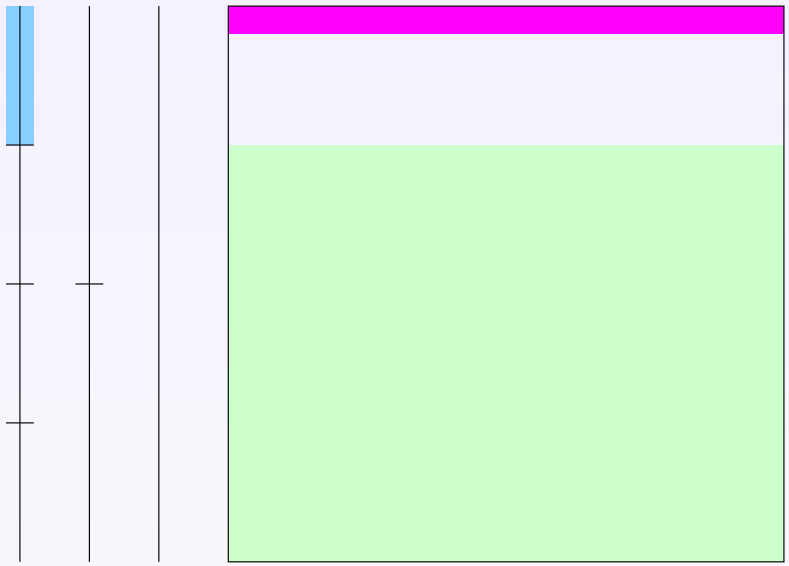
Leaves: Use SVD to find rank-k approximation

$$\widehat{G}_{t,s} := G|_{t\times s} \approx \widehat{V}_t B_{t,s}^*.$$

Matrix $\widehat{V}_t$ defines leaf matrix V_t.

One block

Goal: Construct row cluster basis for one block.



Leaves: Use SVD to find rank-k approximation

$$\widehat{G}_{t,s} := G|_{t\times s} \approx \widehat{V}_t B_{t,s}^*.$$

Matrix $\widehat{V}_t$ defines leaf matrix V_t.

One block

Goal: Construct row cluster basis for one block.

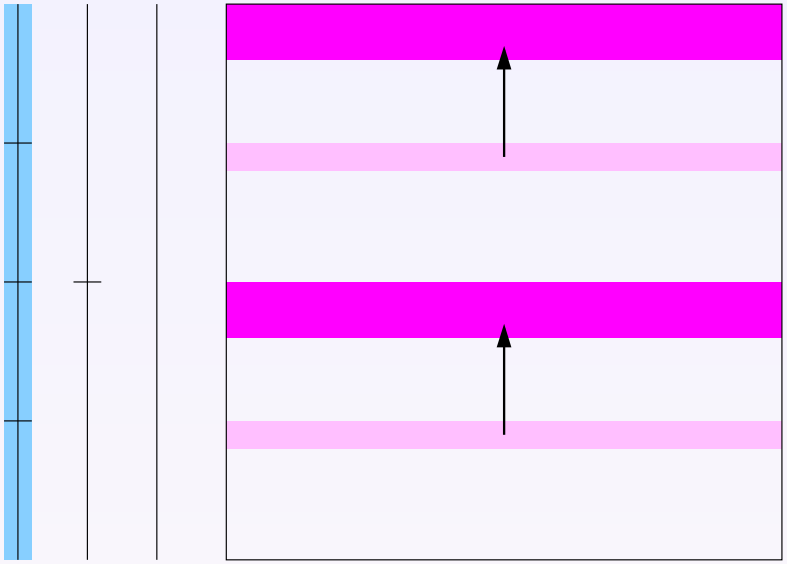
Leaves: Use SVD to find rank-k approximation

$$\widehat{G}_{t,s} := G|_{t\times s} \approx \widehat{V}_t B_{t,s}^*.$$

Matrix $\widehat{V}_t$ defines leaf matrix V_t.

One block

Goal: Construct row cluster basis for one block.

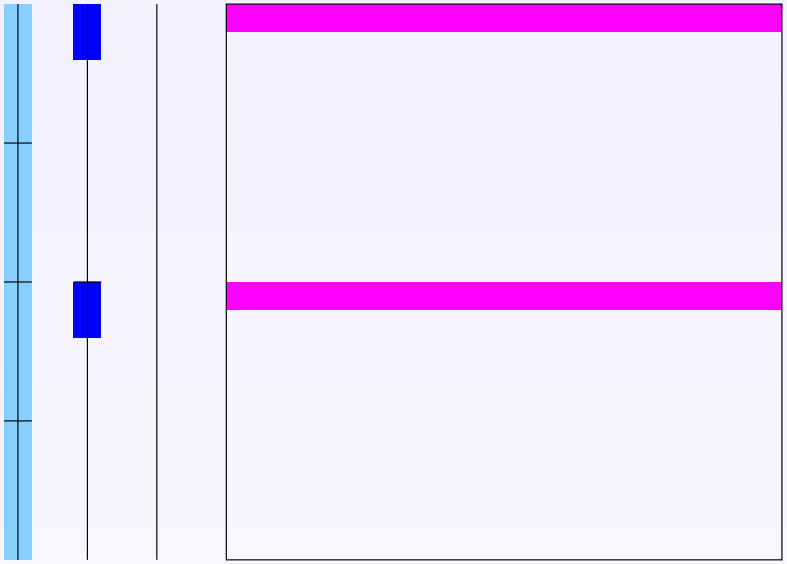


Non-leaves: Combine remainder matrices of sons

$$\widehat{G}_{t,s} = \begin{pmatrix} B^*_{t_1,s} \\ B^*_{t_2,s} \end{pmatrix}$$

One block

Goal: Construct row cluster basis for one block.



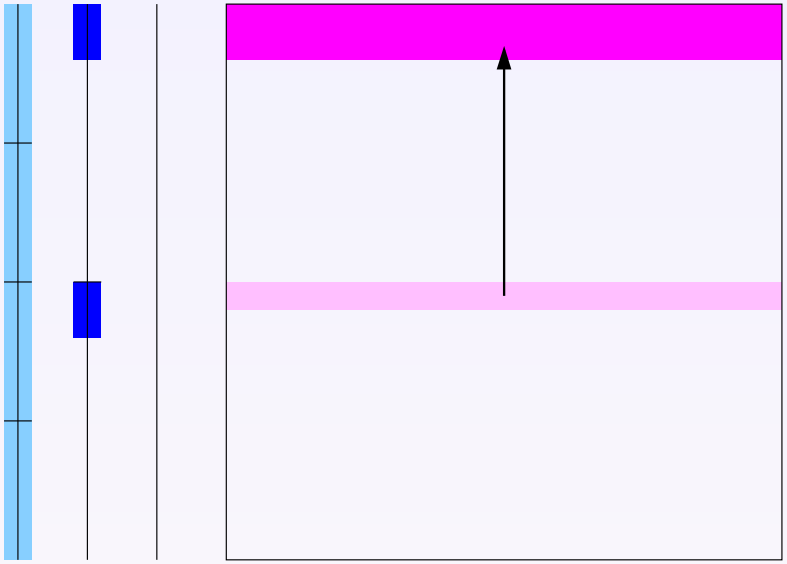
Non-leaves: Use SVD to find rank-k approximation

$$\widehat{G}_{t,s} \approx \widehat{V}_t B_{t,s}^*.$$

Upper and lower halves of $\widehat{V}_t$ define transfer matrices.

One block

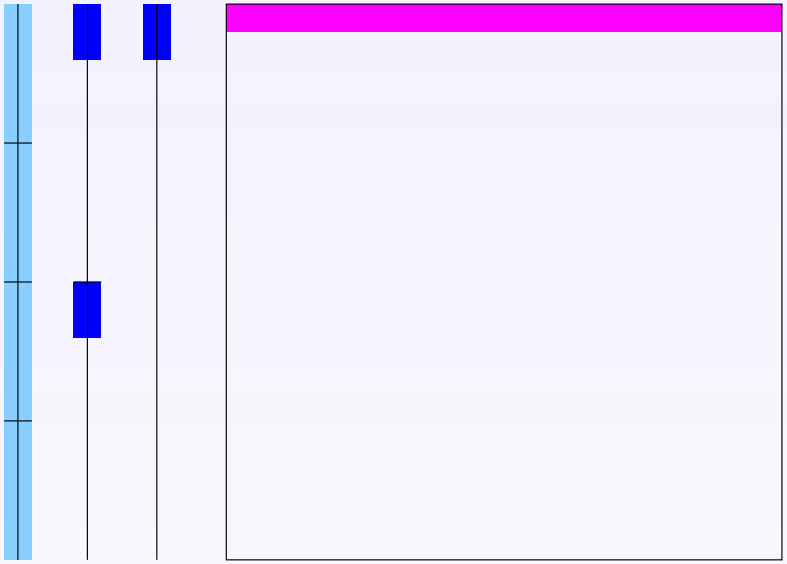
Goal: Construct row cluster basis for one block.



Work upwards through the tree until the block has been compressed.

One block

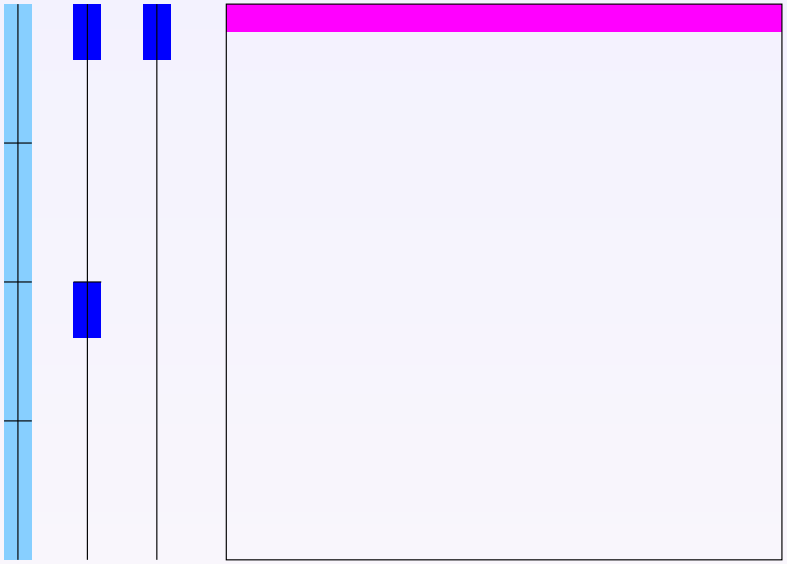
Goal: Construct row cluster basis for one block.



Work upwards through the tree until the block has been compressed.

One block

Goal: Construct row cluster basis for one block.

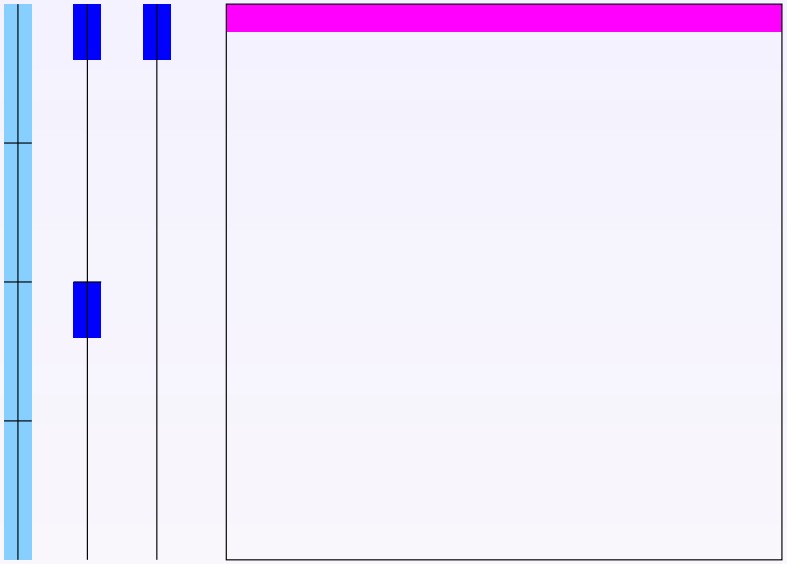


Result:

- Orthogonal cluster basis.
- Best-approximation properties.
- Algorithm uses only “thin” matrices, $\mathcal{O}(n^2 k)$ operations.

One block

Goal: Construct row cluster basis for one block.



Result:

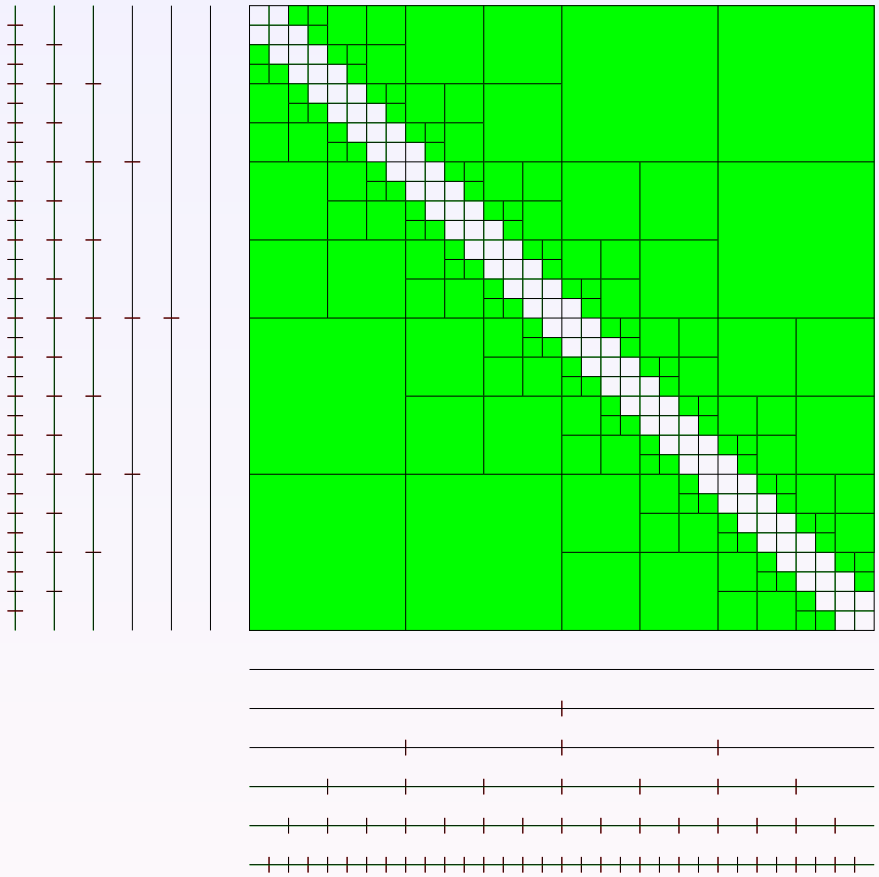
- Orthogonal cluster basis.
- Best-approximation properties.
- Algorithm uses only "thin" matrices, $\mathcal{O}(n^2 k)$ operations.

Approximation error given by sum of intermediate errors

$$\|(G|_{t\times s} - V_t B_{t,s}^*)x\|^2 = \sum_{t'\subseteq t} \|(\widehat{G}_{t,s} - \widehat{V}_t B_{t,s}^*)x\|^2.$$

Compression algorithm

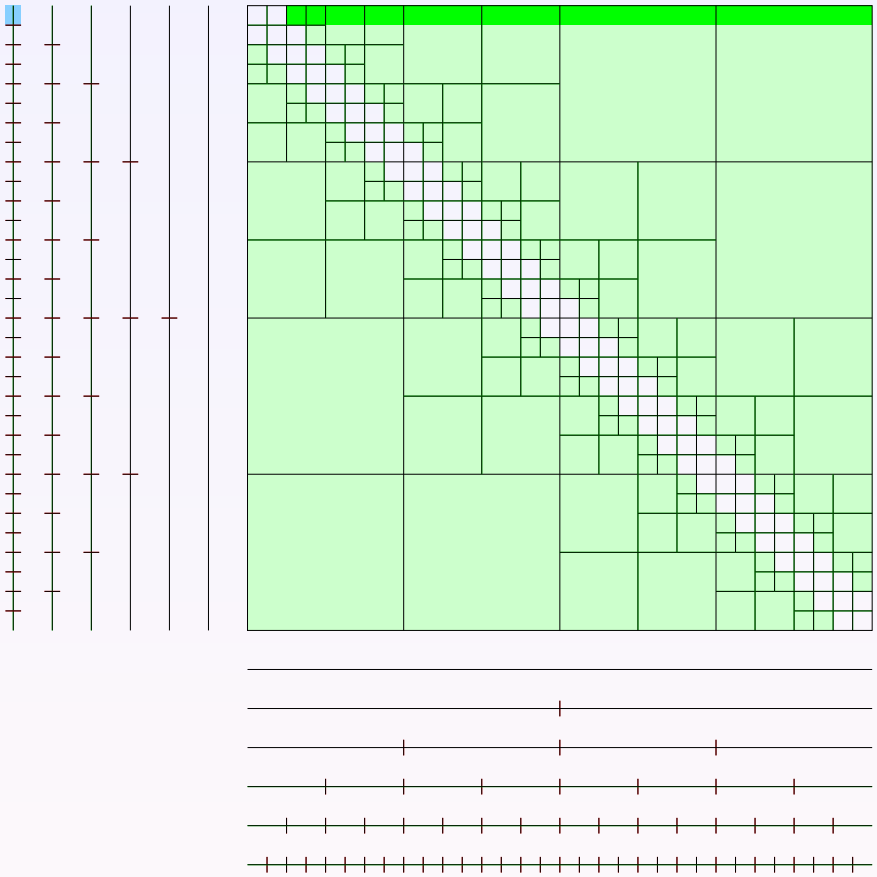
Idea: Treat all blocks connected to a cluster simultaneously.



Goal: Approximate entire matrix by $\mathcal{H}^2$-matrix.

Compression algorithm

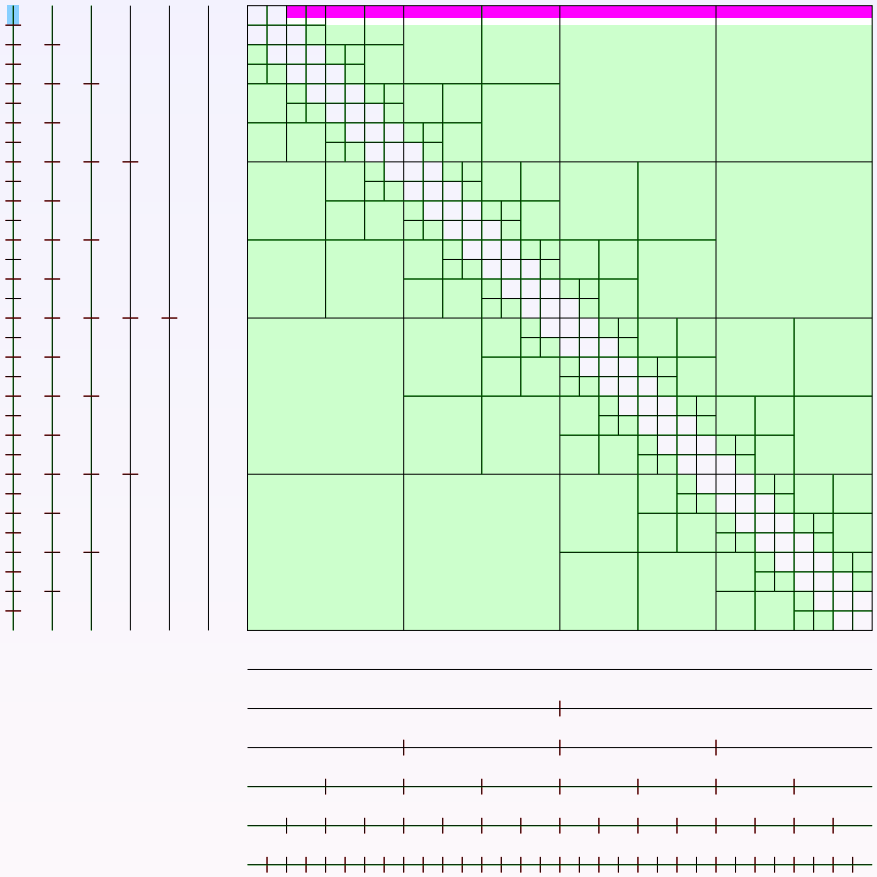
Idea: Treat all blocks connected to a cluster simultaneously.



Leaf clusters: Use SVD to compute best low-rank approximation.

Compression algorithm

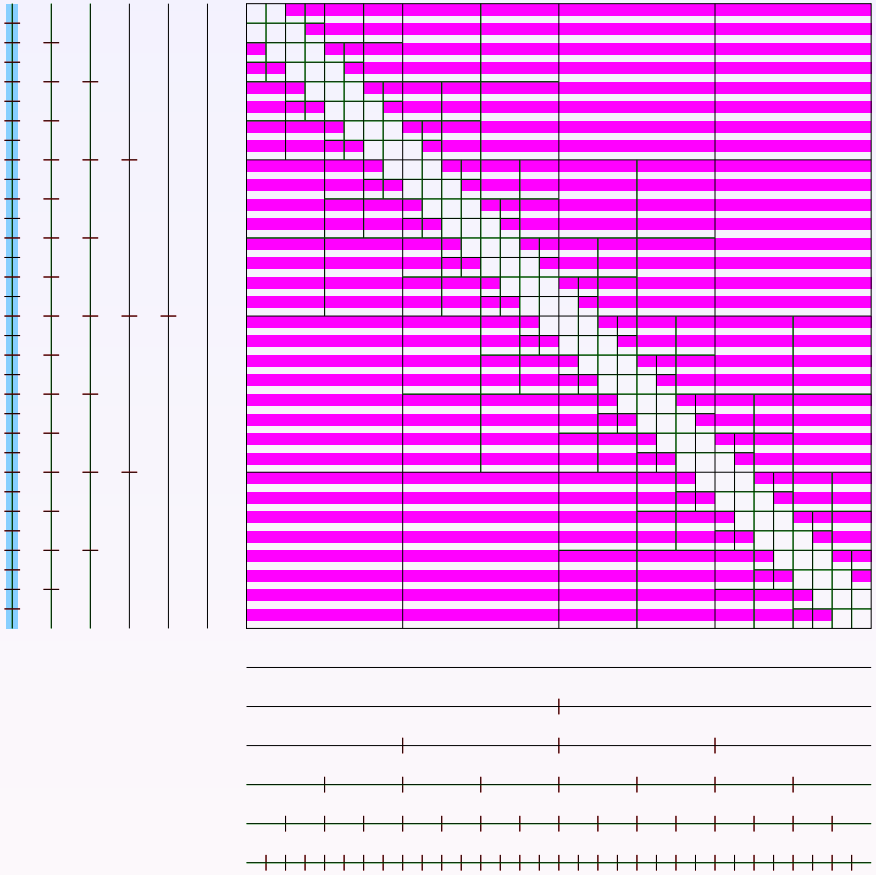
Idea: Treat all blocks connected to a cluster simultaneously.



Leaf clusters: Use SVD to compute best low-rank approximation.

Compression algorithm

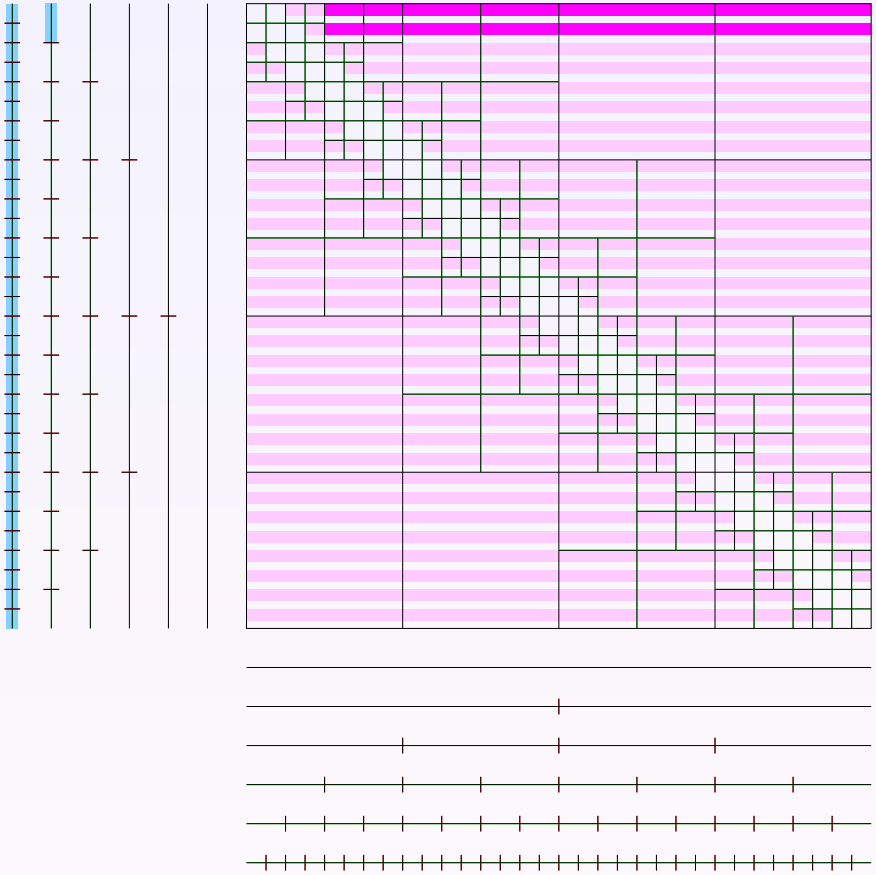
Idea: Treat all blocks connected to a cluster simultaneously.



Leaf clusters: Use SVD to compute best low-rank approximation.

Compression algorithm

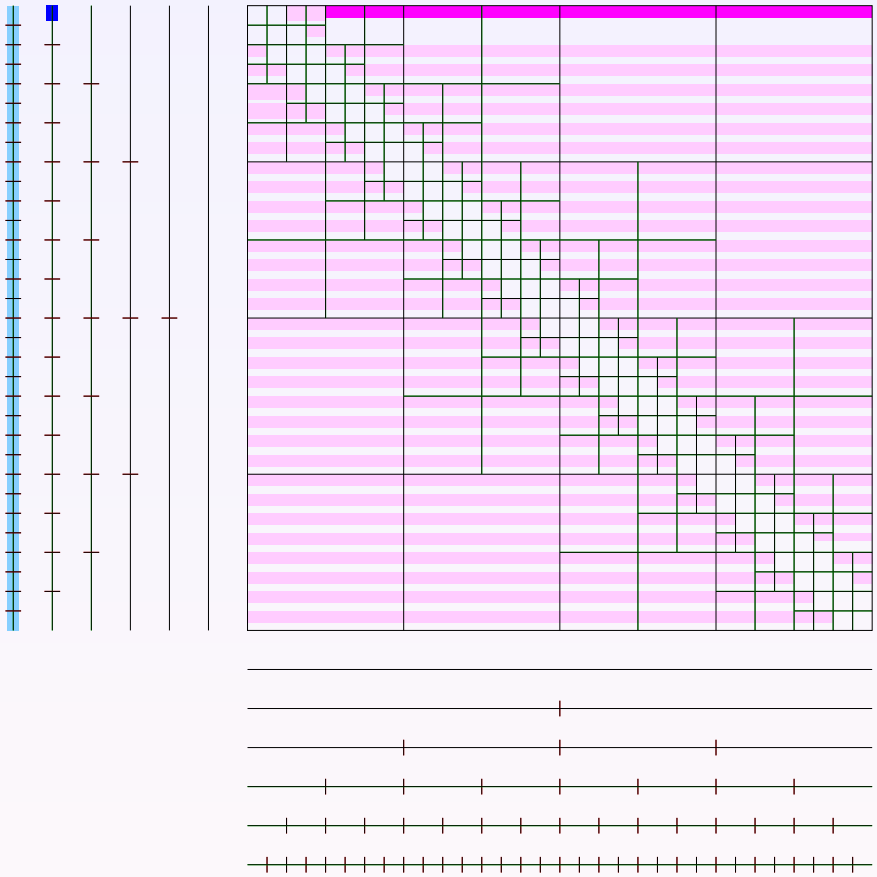
Idea: Treat all blocks connected to a cluster simultaneously.



Father clusters: Apply SVD to results of the sons' compression.

Compression algorithm

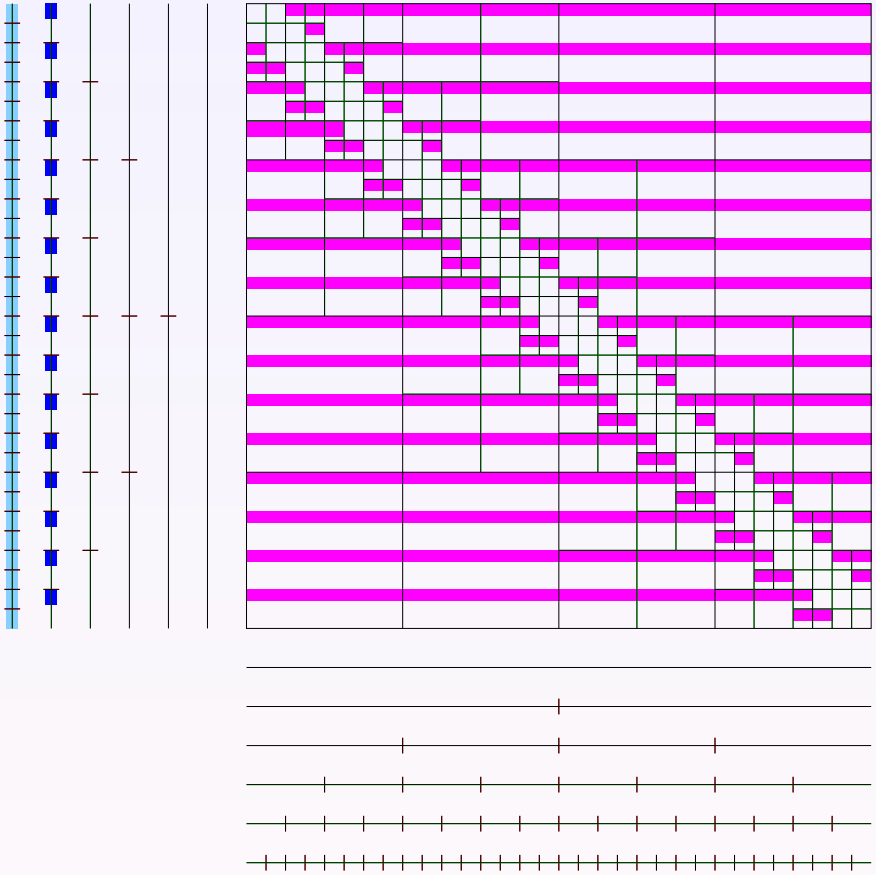
Idea: Treat all blocks connected to a cluster simultaneously.



Father clusters: Apply SVD to results of the sons' compression.

Compression algorithm

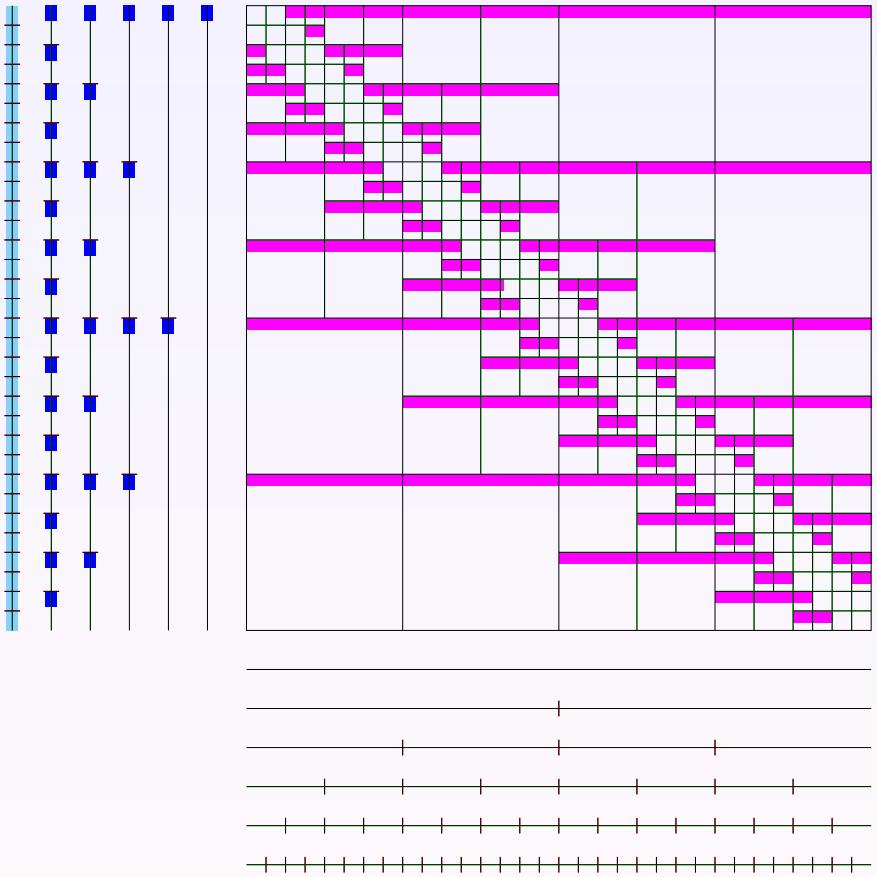
Idea: Treat all blocks connected to a cluster simultaneously.



Father clusters: Apply SVD to results of the sons' compression.

Compression algorithm

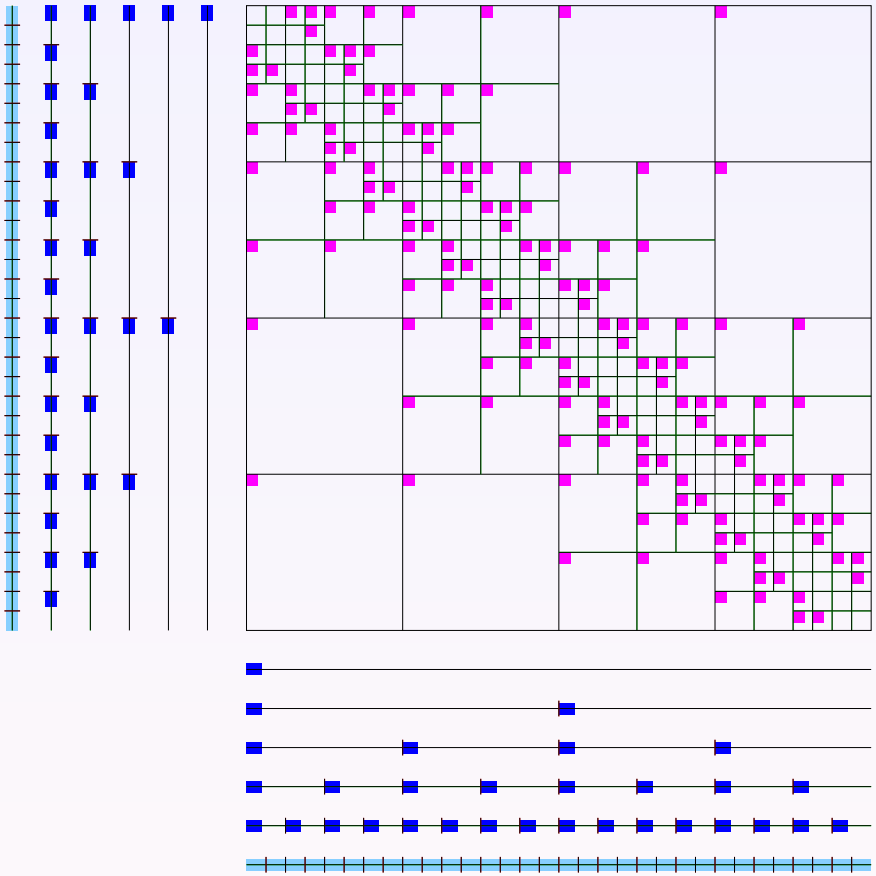
Idea: Treat all blocks connected to a cluster simultaneously.



Work towards the root until cluster basis $(V_t)_t$ has been computed.

Compression algorithm

Idea: Treat all blocks connected to a cluster simultaneously.



Apply procedure to columns.

Result: Compression of general $n \times n$ matrix in $\mathcal{O}(n^2 k)$ operations, rank k very close to minimum.

Improvement: Take advantage of data-sparse input matrix.
For $\mathcal{H}$-matrix: $\mathcal{O}(nk^2 \log n)$.
For $\mathcal{H}^2$-matrix: $\mathcal{O}(nk^2)$.

Overview

LR factorization

Goal: Given an $\mathcal{H}^2$-matrix G, compute its LR factorization $G = LR$.

Approach: If G is inadmissible, compute $G = LR$ directly.
Otherwise, use submatrices

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$$

LR factorization

Goal: Given an $\mathcal{H}^2$-matrix G, compute its LR factorization $G = LR$.

Approach: If G is inadmissible, compute $G = LR$ directly.
Otherwise, use submatrices

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$$

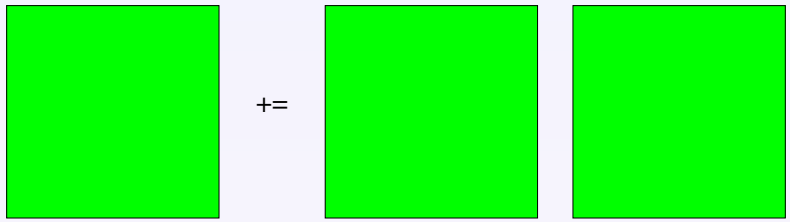
This is equivalent to

$$\begin{aligned} G_{11} &= L_{11} R_{11}, \\ G_{12} = L_{11} R_{12}, &\qquad G_{21} = L_{21} R_{11}, \\ G_{22} - L_{21} R_{12} &= L_{22} R_{22}. \end{aligned}$$

If we can compute $Z \leftarrow Z + \alpha XY$ efficiently, we can use recursion to perform matrix forward substitution and find the LR factorization.

Multiplication

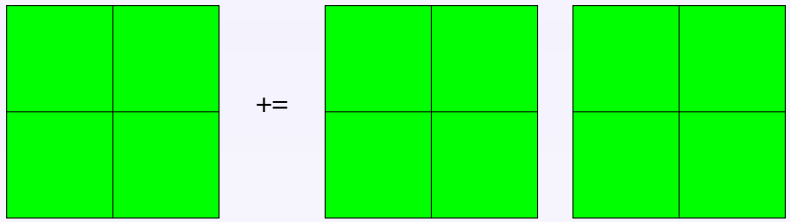
Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.



Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

Multiplication

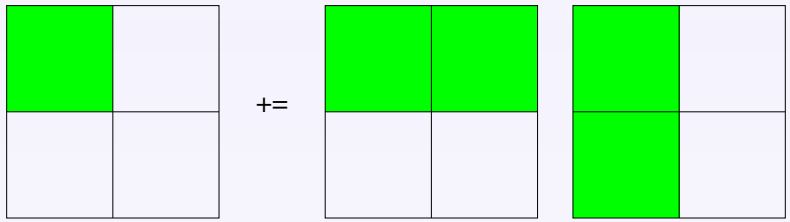
Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.



Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

Multiplication

Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.

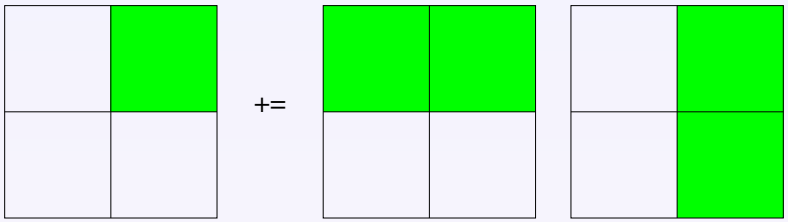


Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

- $Z|_{t_1\times r_1} \leftarrow Z|_{t_1\times r_1} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_1}$

Multiplication

Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.

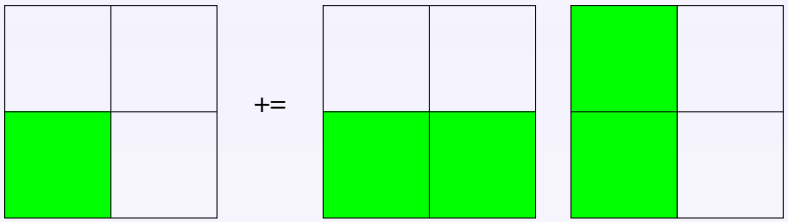


Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

- $Z|_{t_1\times r_1} \leftarrow Z|_{t_1\times r_1} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_1\times r_2} \leftarrow Z|_{t_1\times r_2} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_2}$

Multiplication

Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.

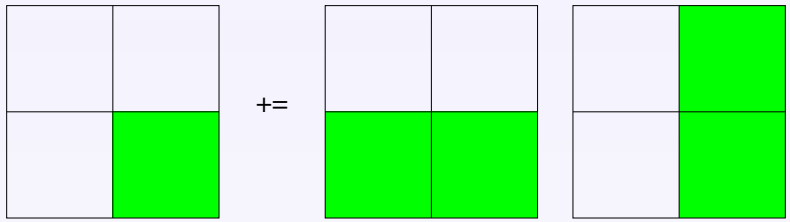


Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

- $Z|_{t_1\times r_1} \leftarrow Z|_{t_1\times r_1} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_1\times r_2} \leftarrow Z|_{t_1\times r_2} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_2}$
- $Z|_{t_2\times r_1} \leftarrow Z|_{t_2\times r_1} + \alpha X|_{t_2\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_2\times s_2} Y|_{s_2\times r_1}$

Multiplication

Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.

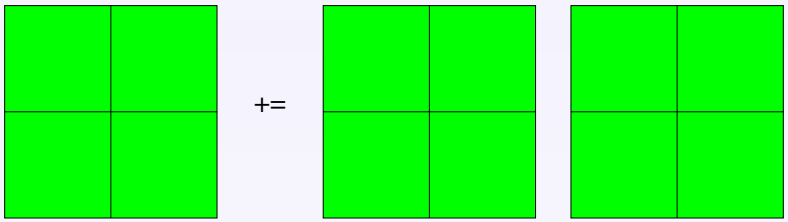


Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

- $Z|_{t_1\times r_1} \leftarrow Z|_{t_1\times r_1} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_1\times r_2} \leftarrow Z|_{t_1\times r_2} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_2}$
- $Z|_{t_2\times r_1} \leftarrow Z|_{t_2\times r_1} + \alpha X|_{t_2\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_2\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_2\times r_2} \leftarrow Z|_{t_2\times r_2} + \alpha X|_{t_2\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_2\times s_2} Y|_{s_2\times r_2}$

Multiplication

Goal: Perform update $Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha X|_{t\times s} Y|_{s\times r}$ efficiently.



Idea: If (t, s) and (s, r) are subdivided, treat them by recursion.

- $Z|_{t_1\times r_1} \leftarrow Z|_{t_1\times r_1} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_1\times r_2} \leftarrow Z|_{t_1\times r_2} + \alpha X|_{t_1\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_1\times s_2} Y|_{s_2\times r_2}$
- $Z|_{t_2\times r_1} \leftarrow Z|_{t_2\times r_1} + \alpha X|_{t_2\times s_1} Y|_{s_1\times r_1} + \alpha X|_{t_2\times s_2} Y|_{s_2\times r_1}$
- $Z|_{t_2\times r_2} \leftarrow Z|_{t_2\times r_2} + \alpha X|_{t_2\times s_1} Y|_{s_1\times r_2} + \alpha X|_{t_2\times s_2} Y|_{s_2\times r_2}$

Multiplication with low-rank block

Important case: What happens if one of the factors is not subdivided?

Multiplication with low-rank block

Important case: What happens if one of the factors is not subdivided?

Example: Let (s, r) be an admissible block of Y.

$$X|_{t\times s} Y|_{s\times r} = X|_{t\times s}(V_s S_{s,r} W_r^*)$$

Multiplication with low-rank block

Important case: What happens if one of the factors is not subdivided?

Example: Let (s, r) be an admissible block of Y.

$$X|_{t\times s} Y|_{s\times r} = X|_{t\times s}(V_s S_{s,r} W_r^*) = (X|_{t\times s} V_s S_{s,r}) W_r^*$$

Multiplication with low-rank block

Important case: What happens if one of the factors is not subdivided?

Example: Let (s, r) be an admissible block of Y.

$$X|_{t\times s} Y|_{s\times r} = X|_{t\times s}(V_s S_{s,r} W_r^*) = (X|_{t\times s} V_s S_{s,r}) W_r^* = A_{t,r} W_r^*.$$

Multiplication with low-rank block

Important case: What happens if one of the factors is not subdivided?

Example: Let (s, r) be an admissible block of Y.

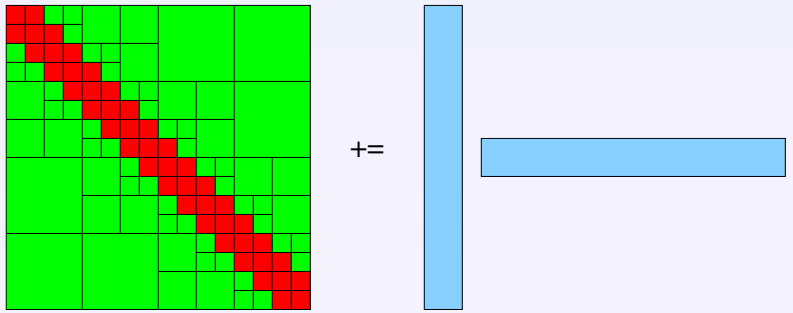
$$X|_{t\times s} Y|_{s\times r} = X|_{t\times s}(V_s S_{s,r} W_r^*) = (X|_{t\times s} V_s S_{s,r}) W_r^* = A_{t,r} W_r^*.$$

Idea: $A_{t,r}$ can be computed by fast matrix-vector multiplications.
We "only" need an efficient algorithm for low-rank updates

$$Z|_{t\times r} \leftarrow Z|_{t\times r} + \alpha A_{t,r} W_r^*.$$

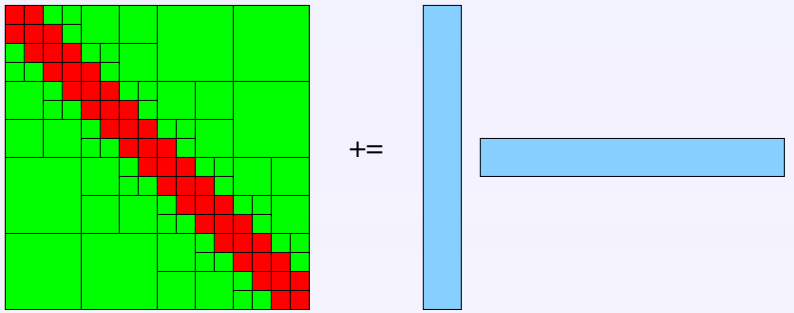
Low-rank update

Goal: Approximate $Z + AB^*$.



Low-rank update

Goal: Approximate $Z + AB^*$.



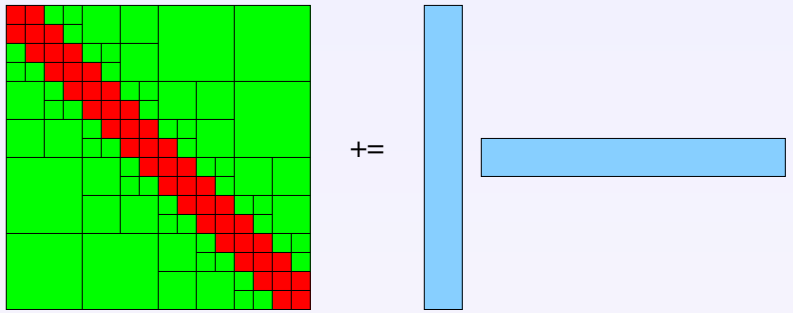
Idea: Result $Z + AB^*$ already is an $\mathcal{H}^2$-matrix:

$$(Z + AB^*)|_{t\times s} = V_t S_{t,s} W_s^* + A|_{t\times k} B|_{s\times k}^*$$

$$= \underbrace{\begin{pmatrix} V_t & A|_{t\times k} \end{pmatrix}}_{=:V_{\mathrm{new},t}} \underbrace{\begin{pmatrix} S_{t,s} & \\ & I \end{pmatrix}}_{=:S_{\mathrm{new},t,s}} \underbrace{\begin{pmatrix} W_s & B|_{s\times k} \end{pmatrix}^*}_{=:W_{\mathrm{new},s}^*}$$

Low-rank update

Goal: Approximate $Z + AB^*$.



Idea: Result $Z + AB^*$ already is an $\mathcal{H}^2$-matrix:

$$\begin{aligned}(Z + AB^*)|_{t\times s} &= V_t S_{t,s} W_s^* + A|_{t\times k} B|_{s\times k}^* \\ &= \underbrace{\begin{pmatrix} V_t & A|_{t\times k}\end{pmatrix}}_{=:V_{\mathrm{new},t}} \underbrace{\begin{pmatrix} S_{t,s} & \\ & I\end{pmatrix}}_{=:S_{\mathrm{new},t,s}} \underbrace{\begin{pmatrix} W_s & B|_{s\times k}\end{pmatrix}^*}_{=:W_{\mathrm{new},s}^*}\end{aligned}$$

Challenge: Each update increases storage requirements, although actual numerical rank may be low. $\rightarrow$ Recompression required.

Recompression

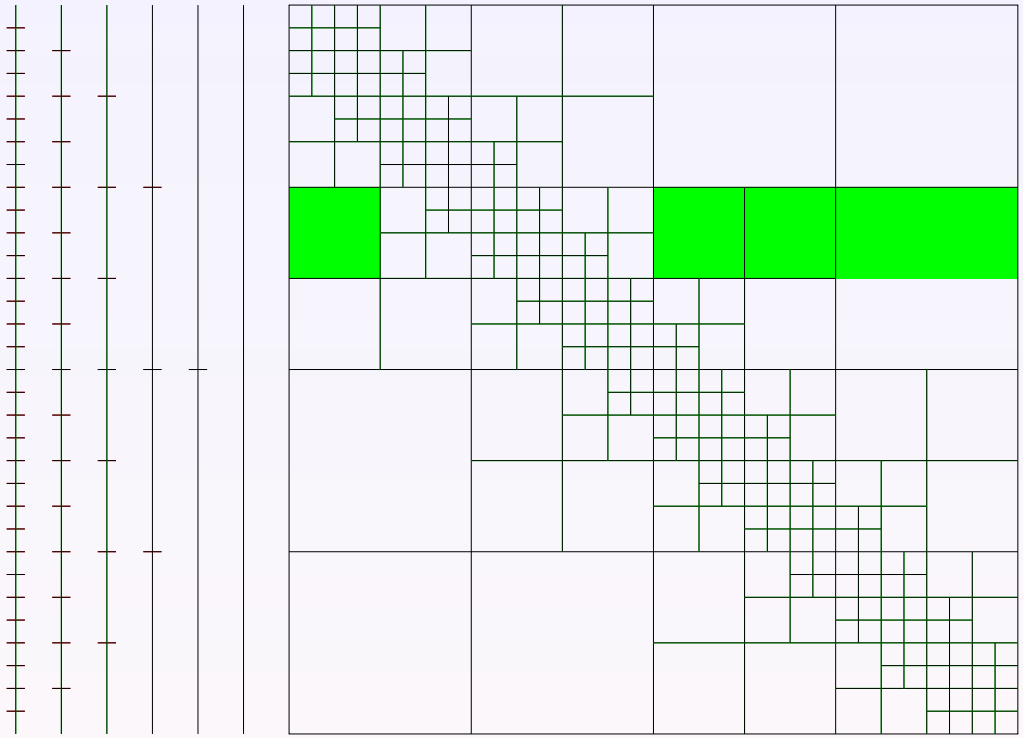
Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank.

Recompression

Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.

Recompression

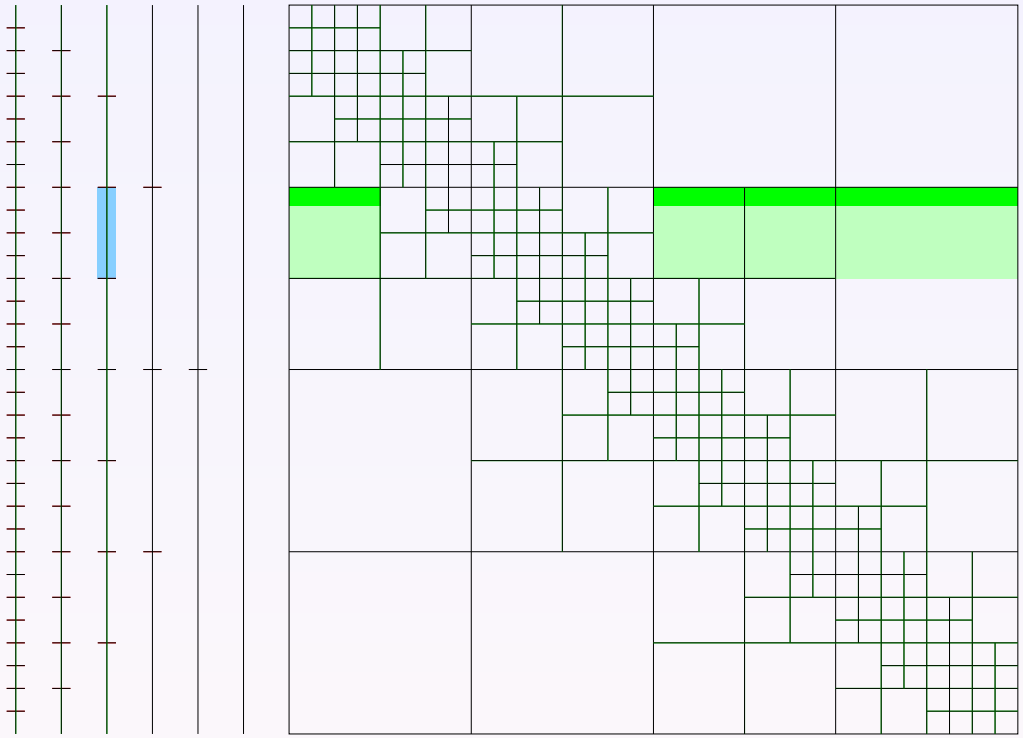
Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.



Goal: Find cluster basis approximating blocks $G|_{t \times R_t}$.

Recompression

Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.

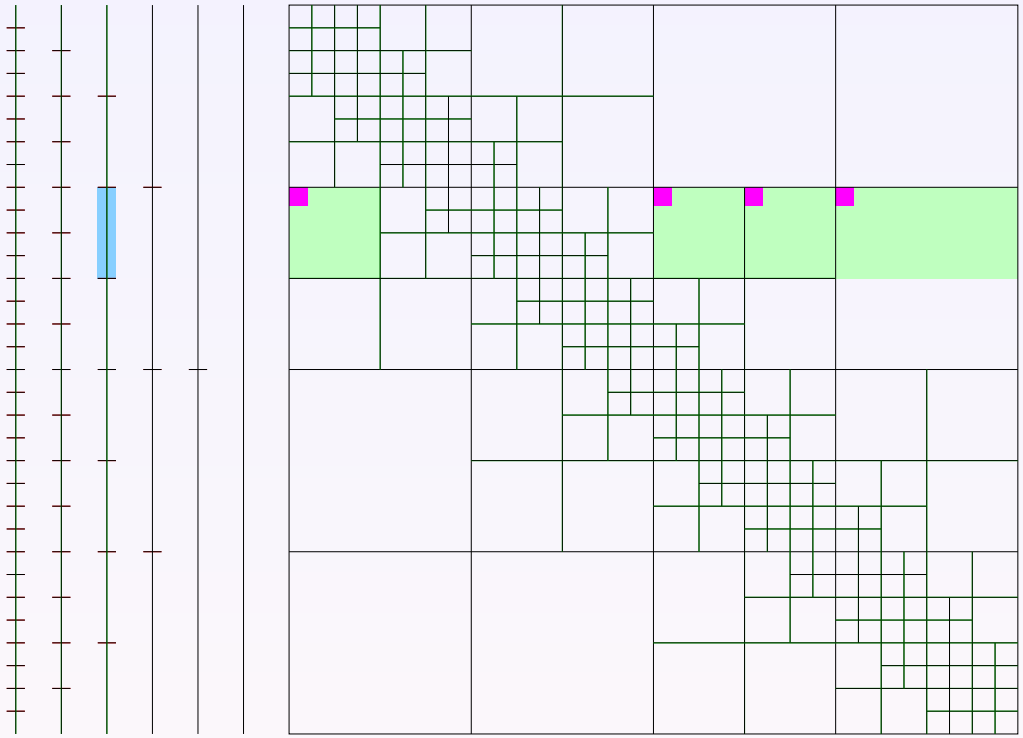


Goal: Find cluster basis approximating blocks $G|_{t \times R_t}$.

$\mathcal{H}^2$-matrix: $G|_{t \times R_t} = V_t B_t^*$

Recompression

Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.



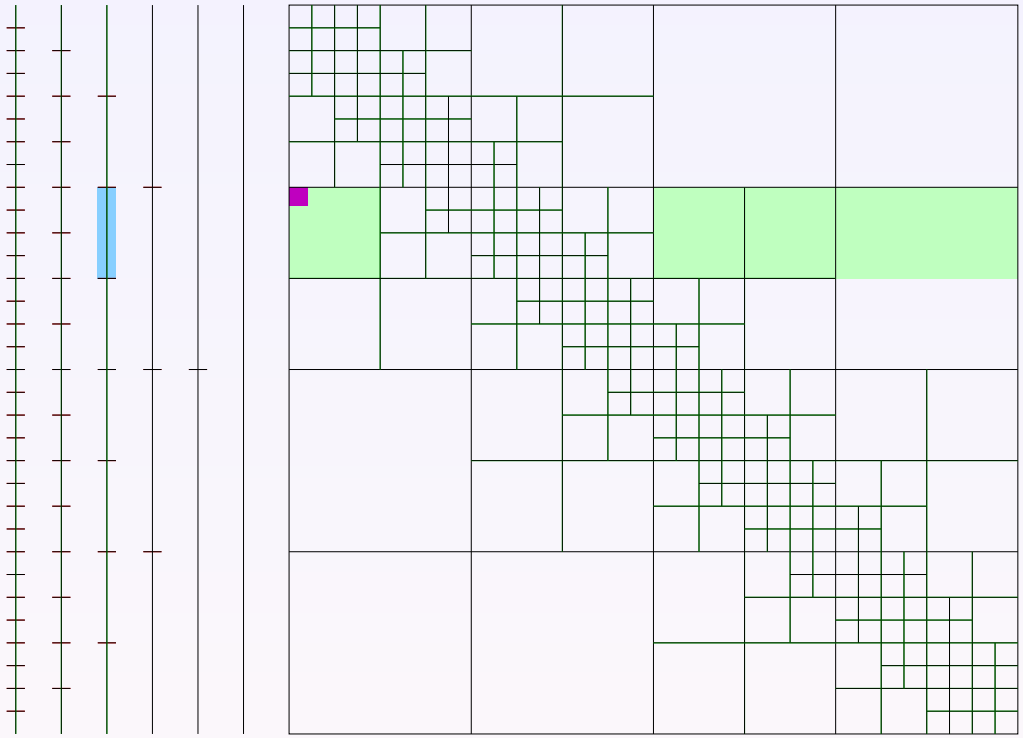
Goal: Find cluster basis approximating blocks $G|_{t\times R_t}$.

$\mathcal{H}^2$-matrix: $G|_{t\times R_t} = V_t B_t^*$

Need only singular values and left singular vectors.
$\rightarrow$ Apply QR-factorization.

Recompression

Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.



Goal: Find cluster basis approximating blocks $G|_{t\times R_t}$.

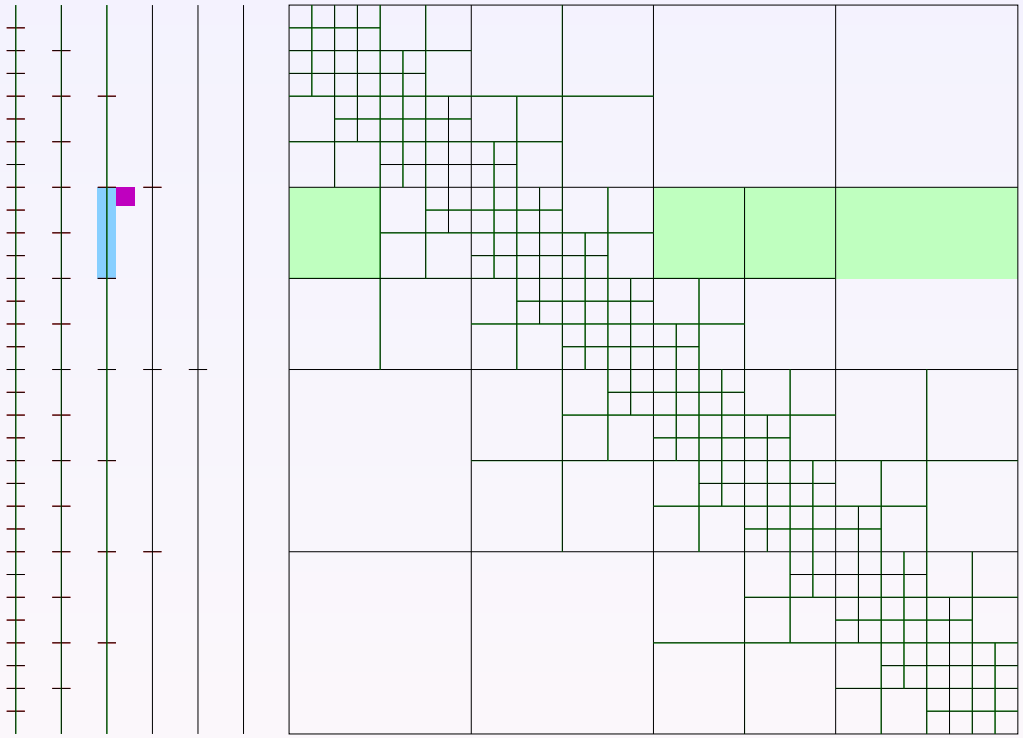
$\mathcal{H}^2$-matrix: $G|_{t\times R_t} = V_t B_t^*$

Need only singular values and left singular vectors.
$\rightarrow$ Apply QR-factorization.

$G|_{t\times R_t} = V_t R_t^* Q_t^*$, $R_t \in \mathbb{R}^{k\times k}$

Recompression

Goal: Given an $\mathcal{H}^2$-matrix G, find an $\mathcal{H}^2$-matrix approximation of sufficient accuracy but with lower rank. And do it efficiently.



Goal: Find cluster basis approximating blocks $G|_{t\times R_t}$.

$\mathcal{H}^2$-matrix: $G|_{t\times R_t} = V_t B_t^*$

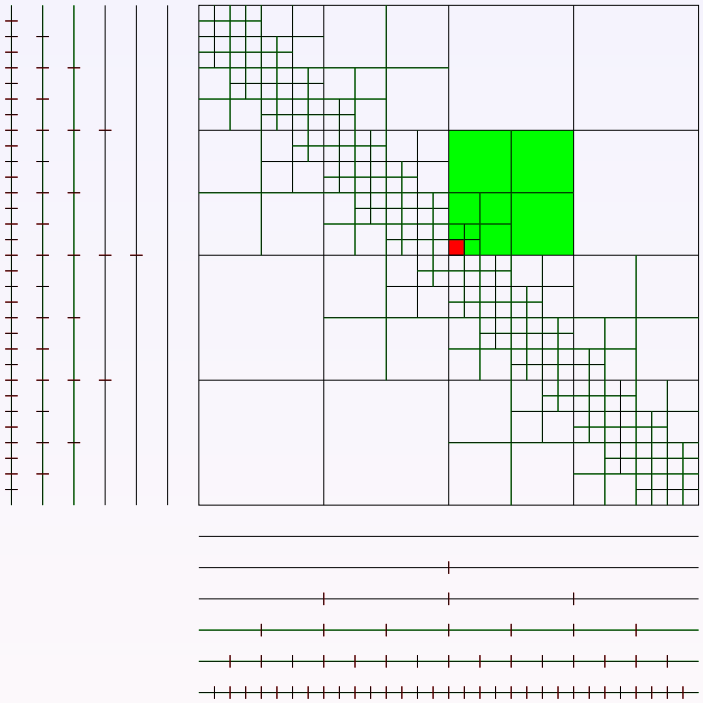
Need only singular values and left singular vectors.
$\rightarrow$ Apply QR-factorization.

$G|_{t\times R_t} = V_t R_t^* Q_t^*$, $R_t \in \mathbb{R}^{k\times k}$

Result: Weight matrices R_t can be computed in $\mathcal{O}(nk^2)$ operations.
$\rightarrow$ Singular value decompositions of $V_t R_t^*$ yield new cluster basis.

Local low-rank update

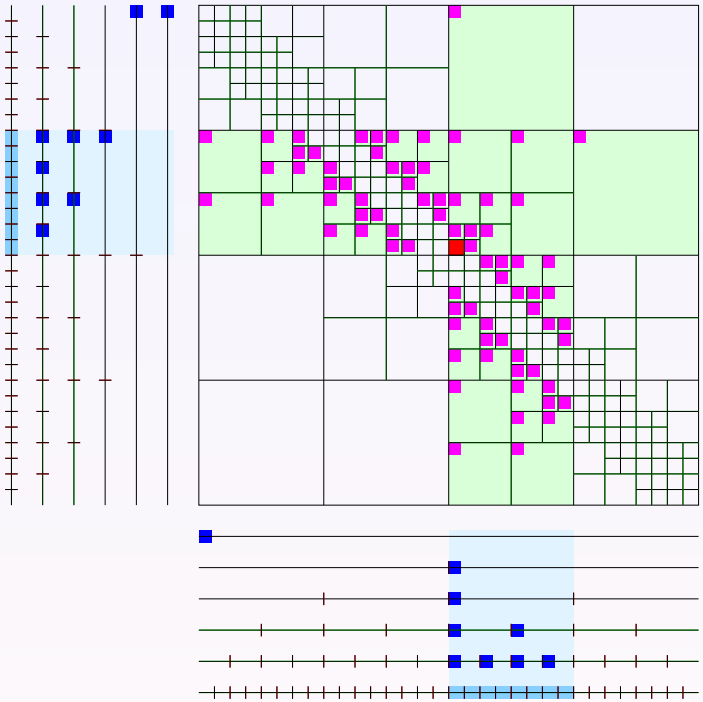
Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.



Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

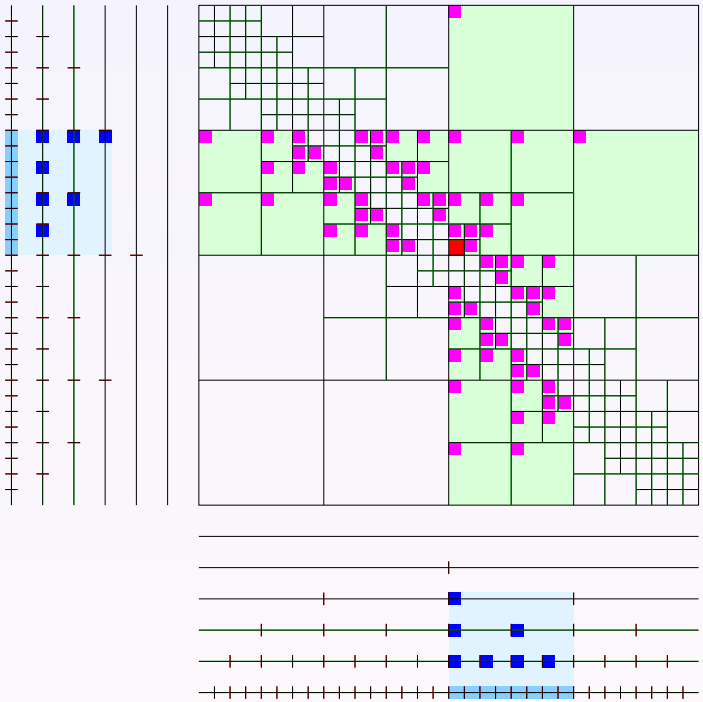
Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.



Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.

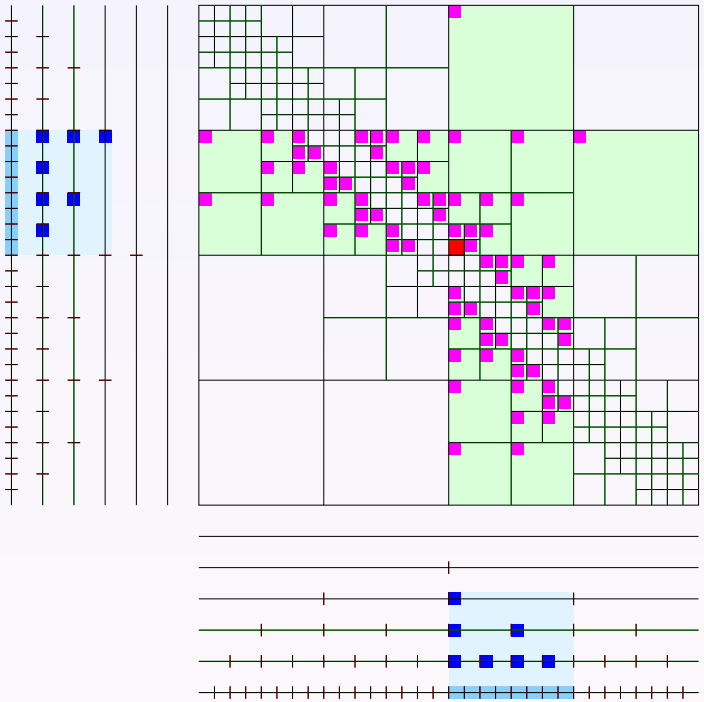


Observation: Nested cluster basis.
$\rightarrow$ Updating transfer matrix updates all ancestors of a cluster.

Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.



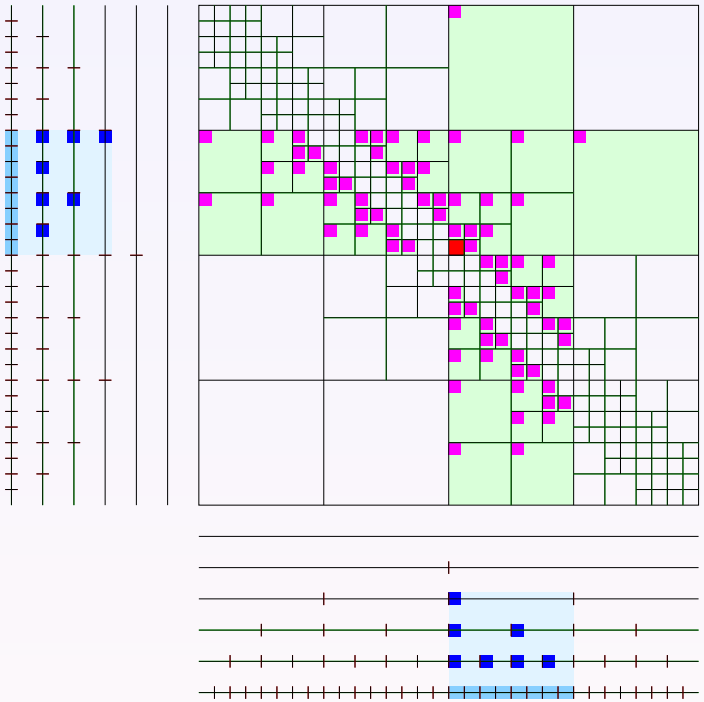
Observation: Nested cluster basis.
$\rightarrow$ Updating transfer matrix updates all ancestors of a cluster.

Important: Keep weights updated.
$\rightarrow$ No visits to ancestors required.

Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.



Observation: Nested cluster basis.
$\rightarrow$ Updating transfer matrix updates all ancestors of a cluster.

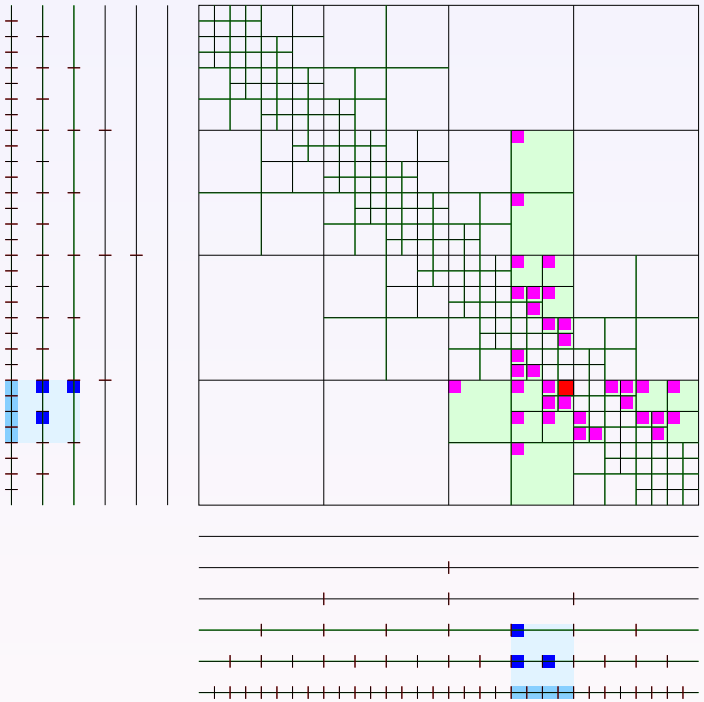
Important: Keep weights updated.
$\rightarrow$ No visits to ancestors required.

Result: Complexity $\mathcal{O}(k^2(\#t + \#s))$

Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.



Observation: Nested cluster basis.
$\rightarrow$ Updating transfer matrix updates all ancestors of a cluster.

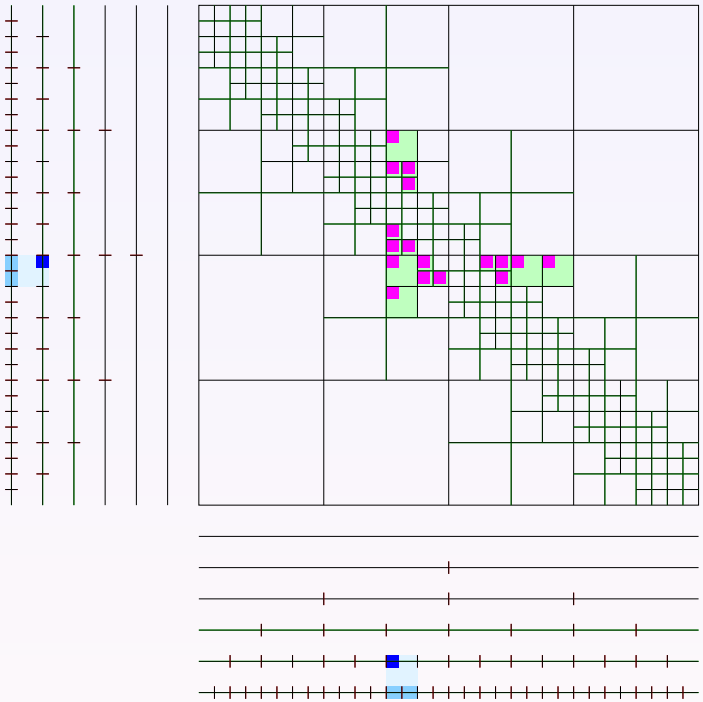
Important: Keep weights updated.
$\rightarrow$ No visits to ancestors required.

Result: Complexity $\mathcal{O}(k^2(\#t + \#s))$

Local low-rank update

Goal: Local update $Z|_{t\times r} \leftarrow Z|_{t\times r} + AB^*$.

Problem: Changing the cluster bases influences all blocks intersecting rows in t or columns in s.



Observation: Nested cluster basis.
$\rightarrow$ Updating transfer matrix updates all ancestors of a cluster.

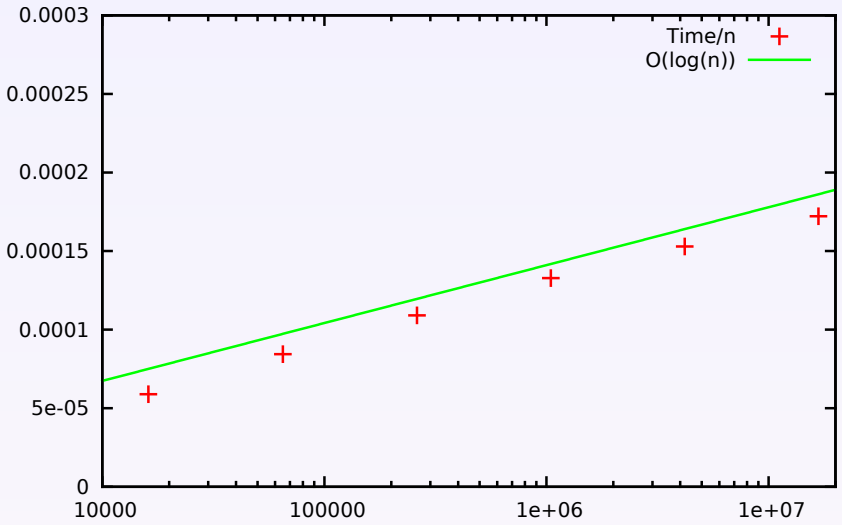
Important: Keep weights updated.
$\rightarrow$ No visits to ancestors required.

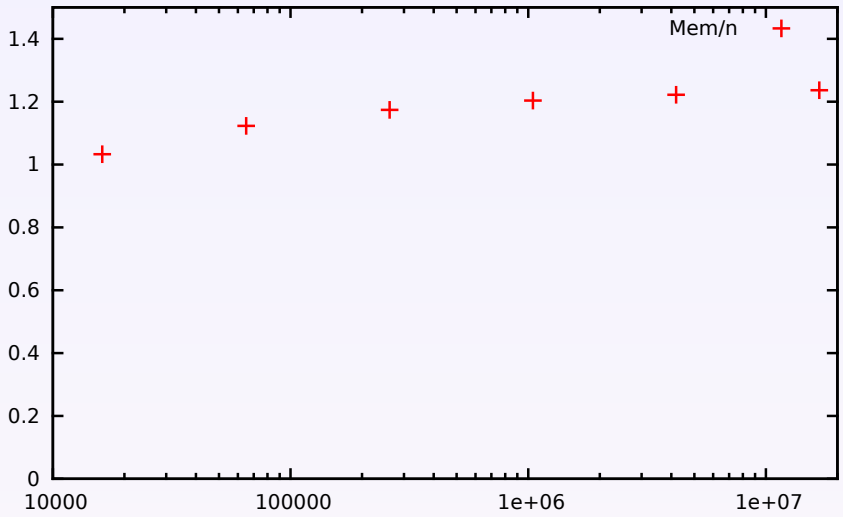
Result: Complexity $\mathcal{O}(k^2(\#t + \#s))$

Overview

Experiment: FEM Cholesky decomposition

Goal: Approximate Cholesky decomposition of a FEM stiffness matrix.



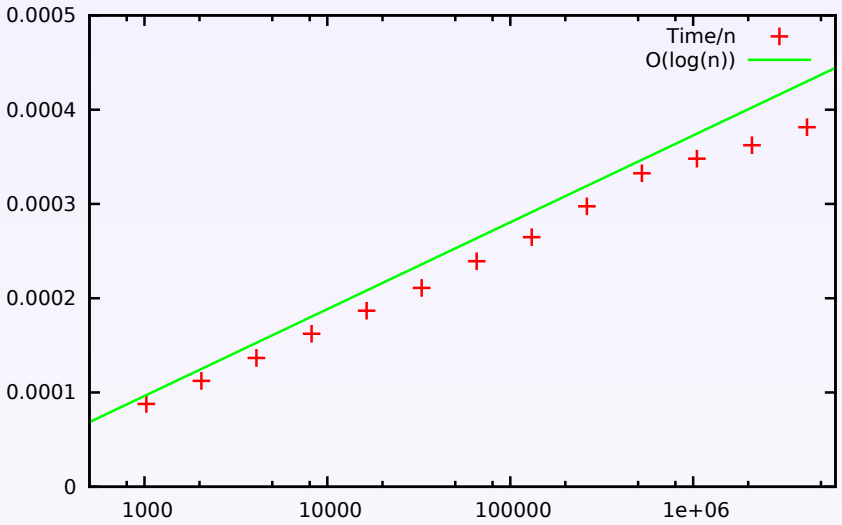


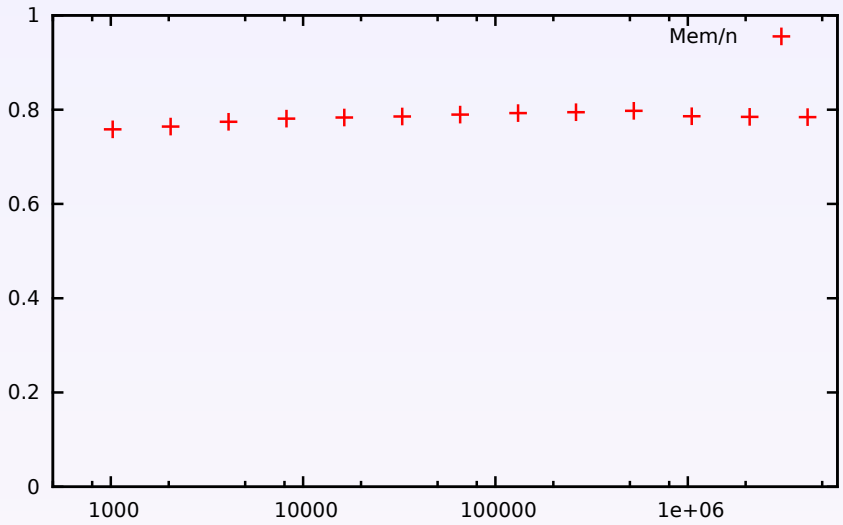
Results:

- Accuracy $\|I - \widetilde{L}^{-*}\widetilde{L}^{-1}A\|_2 \approx 0.1$.
- Factorization in $\sim n \log n$ operations.
- Storage requirements $\sim n$.

Experiment: BEM Cholesky decomposition

Goal: Approximate Cholesky decomposition of a BEM stiffness matrix.





Results:

- Accuracy $\|I - \widetilde{L}^{-*}\widetilde{L}^{-1}A\|_2 \approx 0.2$.
- Factorization in $\sim n \log n$ operations.
- Storage requirements $\sim n$.