Computing eigenvalues in parallel

Daniel Kleine-Albers, 31.03.2009
JASS09, St. Petersburg

Computing eigenvalues in parallel.
Contents.

Daniel Kleine-Albers

Eigenvalues.
How they are defined.

- $Av = \lambda v$
  - λ is the eigenvalue
  - v is the corresponding eigenvector
    - no change of direction
- Solutions to characteristic polynomial $\det(A-\lambda I) = 0$
- Eigenvalues stay the same for similar matrices

Eigenvalues.
What they are good for.

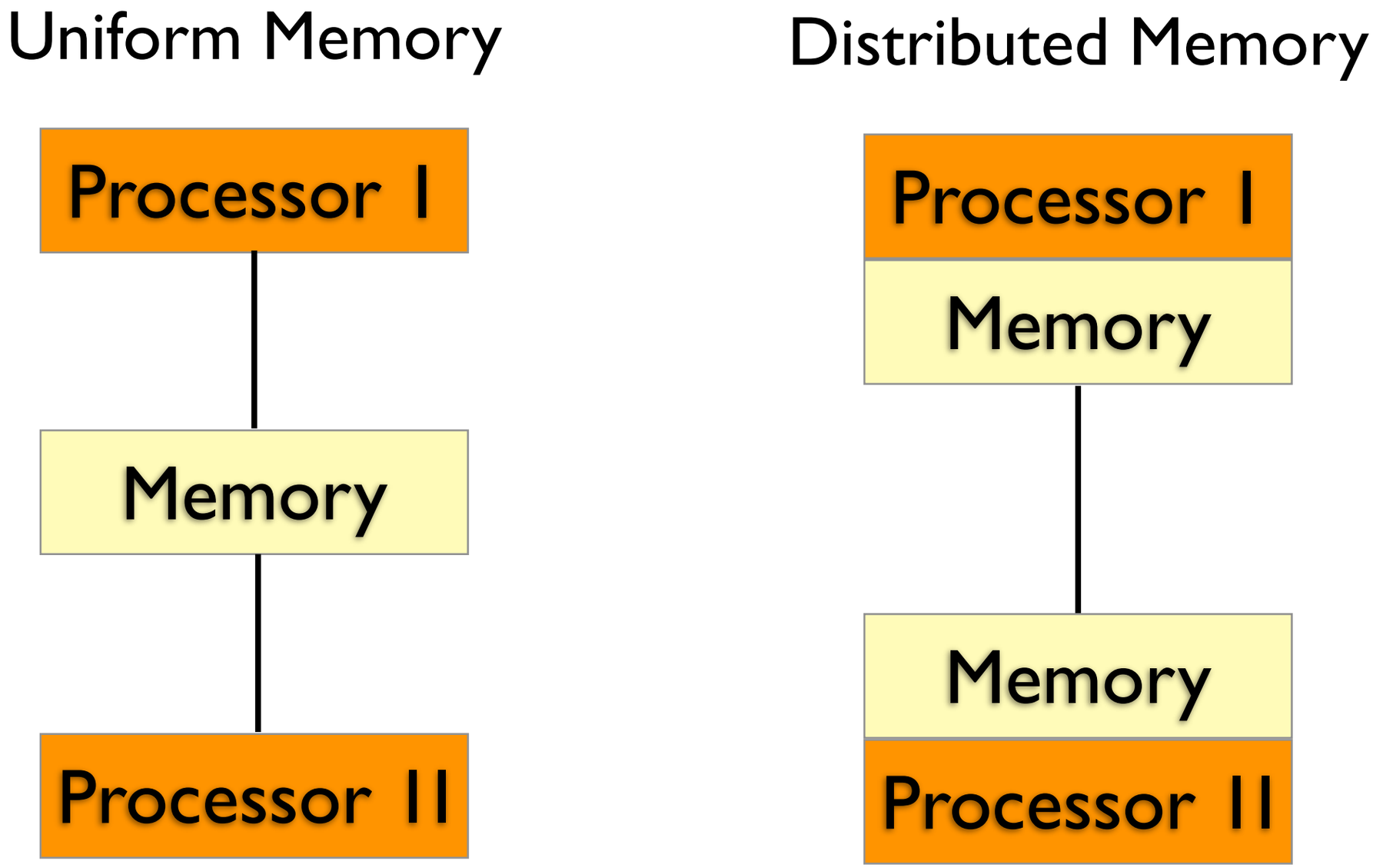
- Quantum mechanics
- Compression algorithms
- Eigenfrequency
- Further physical applications in stiffness calculations, load analysis, etc.

Parallelisation.
Why it is important.

- High performance computers
- Single core clock speeds are close to the physical limits
- Multi core systems are standard
- Even mobile phones will become multi core processors soon
- Flexibility: Parallel programs can easily be run serial, but not the other way round

Daniel Kleine-Albers

Parallelisation.
Basic parallel architectures.



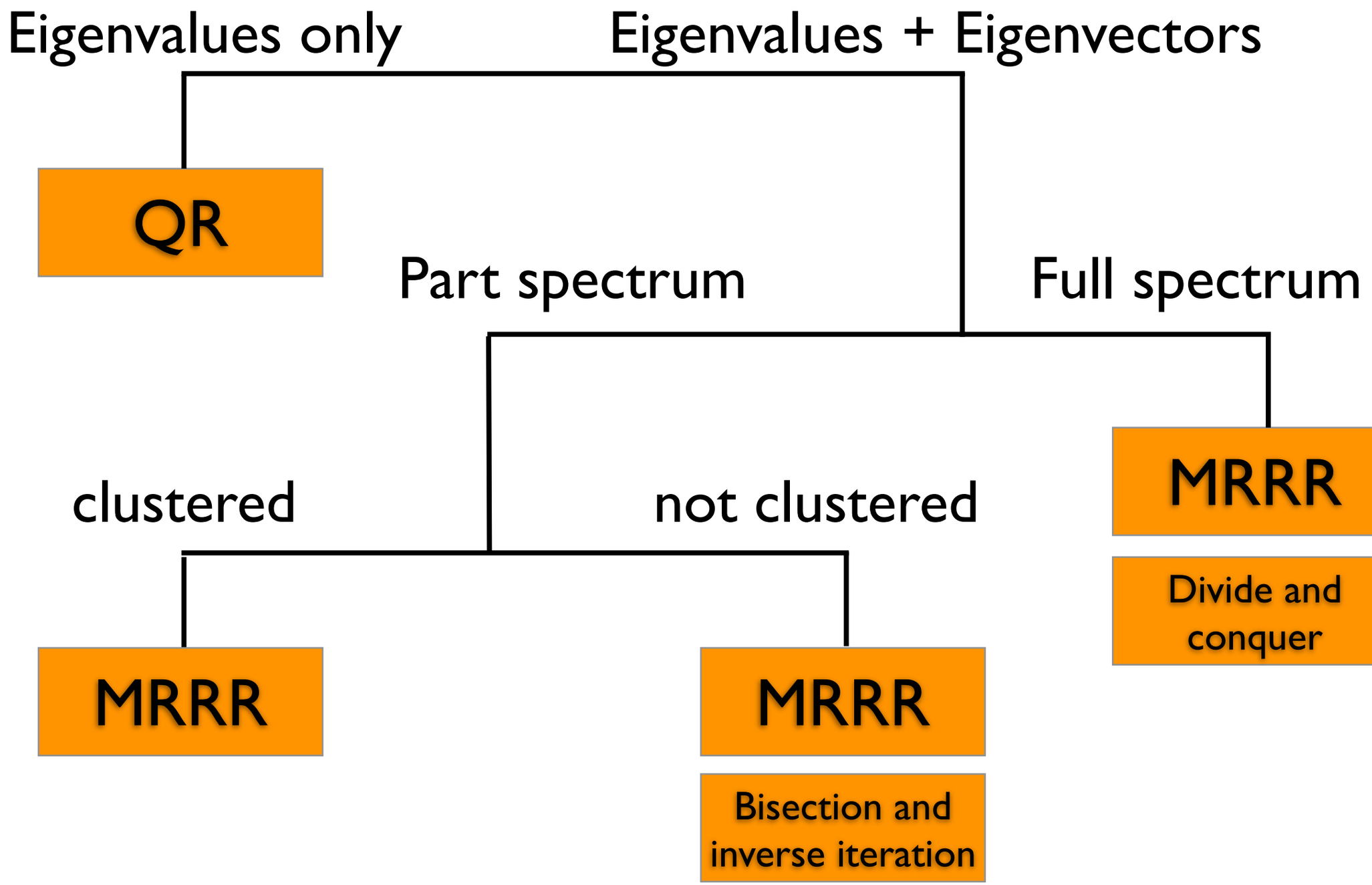
Parallelisation.
Criteria for parallelization.

- Data locality
- Data dependence
- Communication overhead
- Speedup achieved

Computing eigenvalues in parallel. Contents.

Daniel
Kleine-Albers

Algorithms.
Choose depending on your needs.



Computing eigenvalues in parallel.
Contents.

Daniel Kleine-Albers

QR Iteration.
Basic algorithm.

```
n = size(A)
while (n > 1) {
    Factorize A = QR using QR decomposition
    A = RQ
    if (a_{n,n-1} < tolerance) {
        output λ_n = a_{n,n}
        remove row and column n
        n--
    }
}
output λ_n = a_{n,n}
```

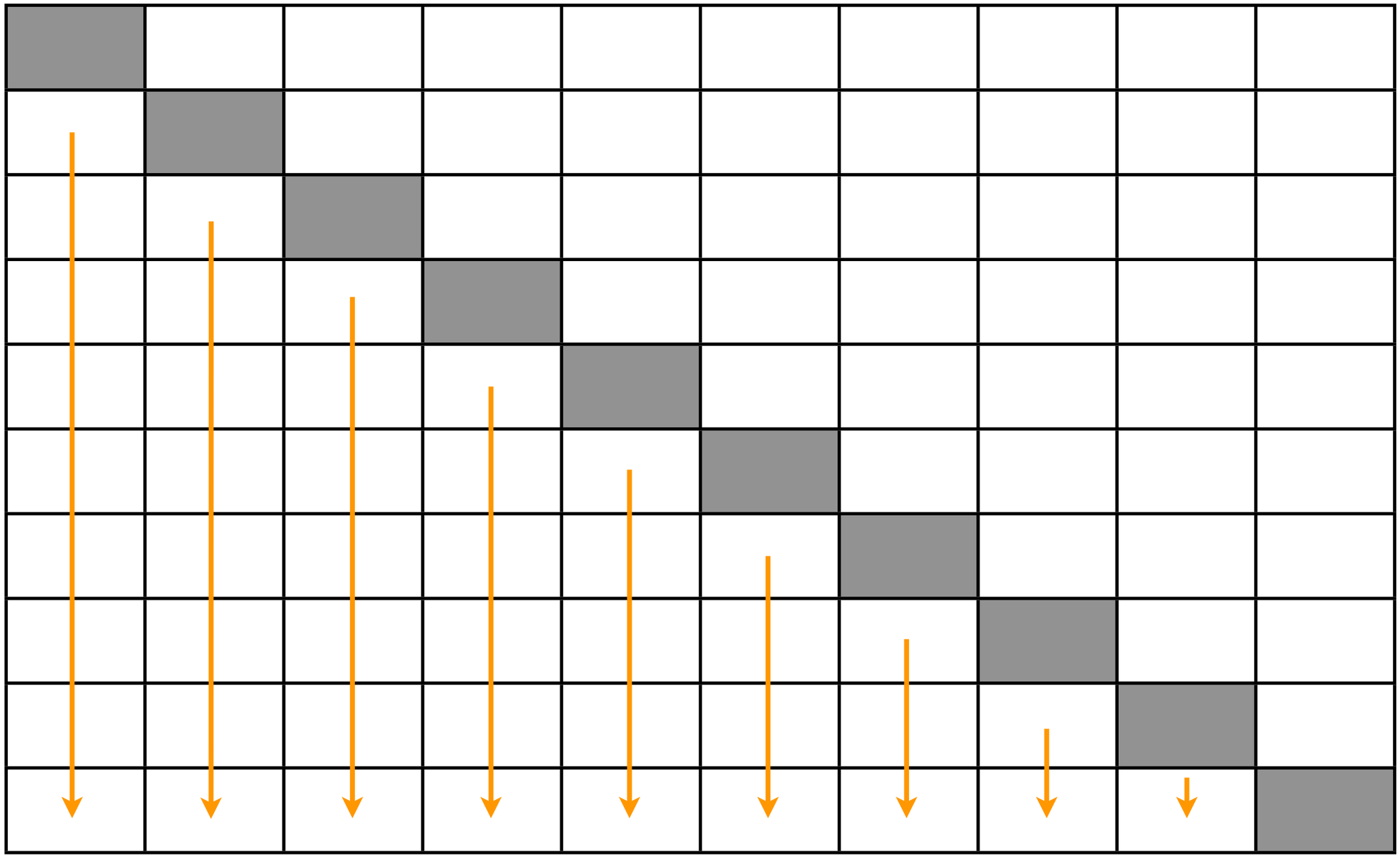
Daniel Kleine-Albers

QR Iteration.
Notes.

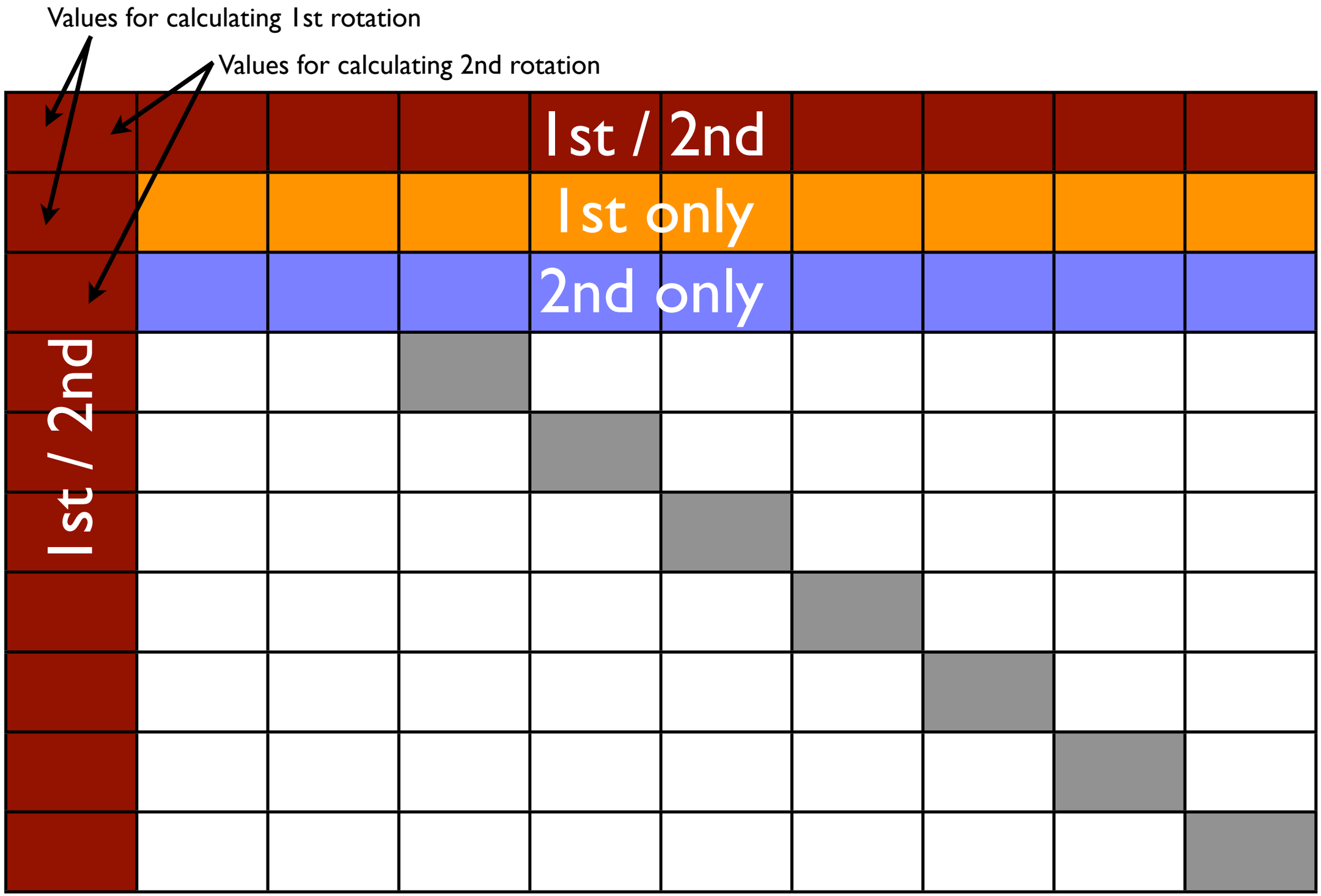
- Based on Givens rotations to zero out the sub-diagonal elements:
  $O(n^2)$ rotations per step
- QR decomposition does not need to be explicitly calculated:
  $A = \ldots G_2 * (G_1 * A * G_1^T) * G_2^T \ldots$
- Performance depends mainly on the number of non-zero elements in the matrix

Daniel
Kleine-Albers

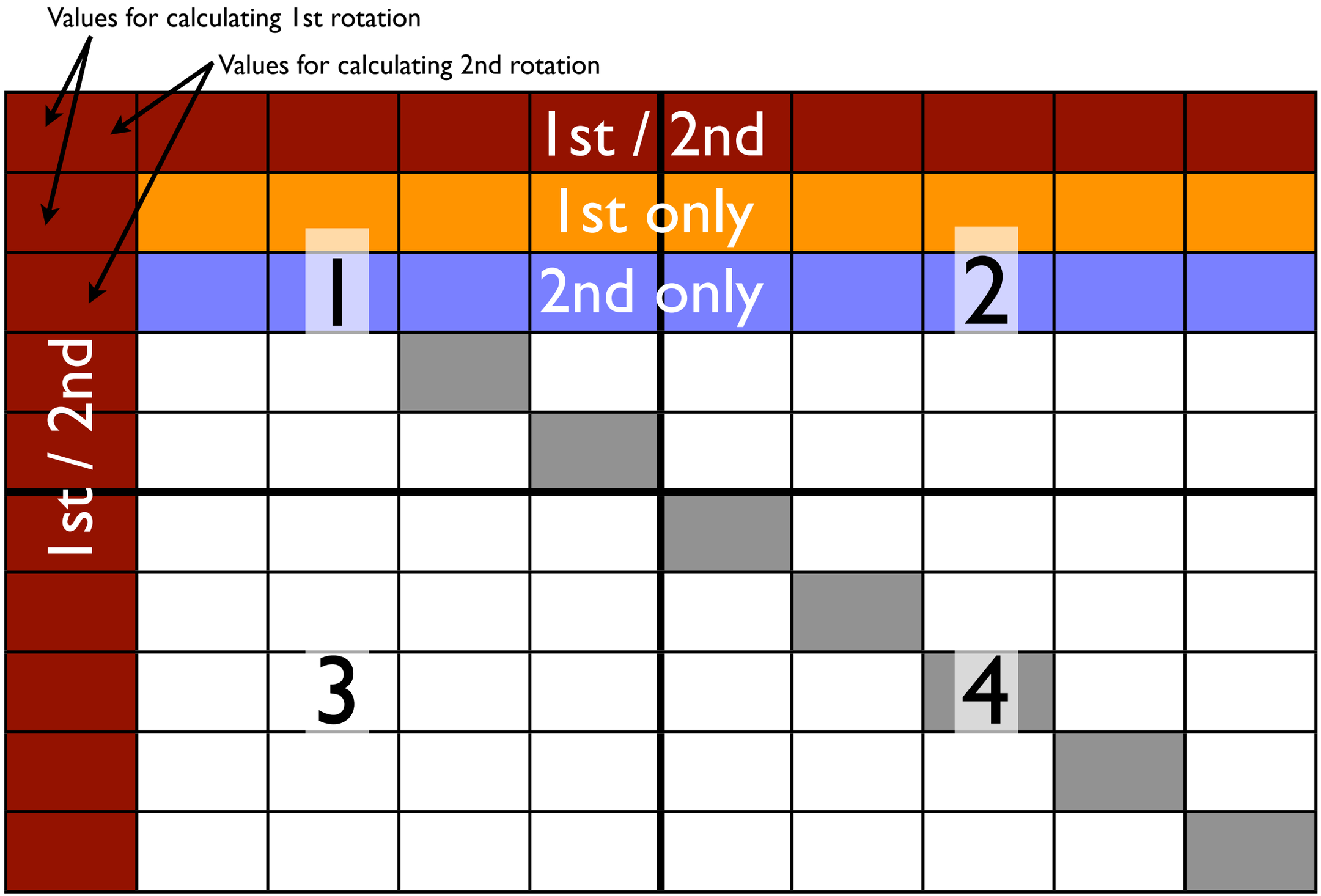
QR Iteration.
Order of Givens rotations.



QR Iteration.
Changed values.



QR Iteration.
Pipeline parallelization.



QR Iteration.
Parallelization.

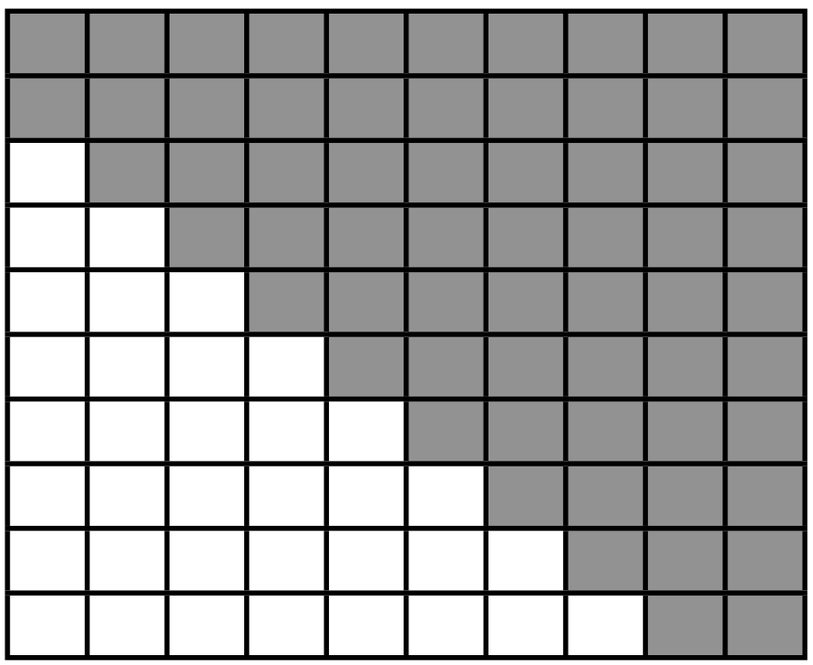
- Hard to parallelize the algorithm itself because steps need to be executed sequentially
  -> however a pipeline scheme is possible
- Idea:
  Transform the matrix to a compact form to reduce number of required Givens rotations

Finding eigenvalues efficiently.
General approach.

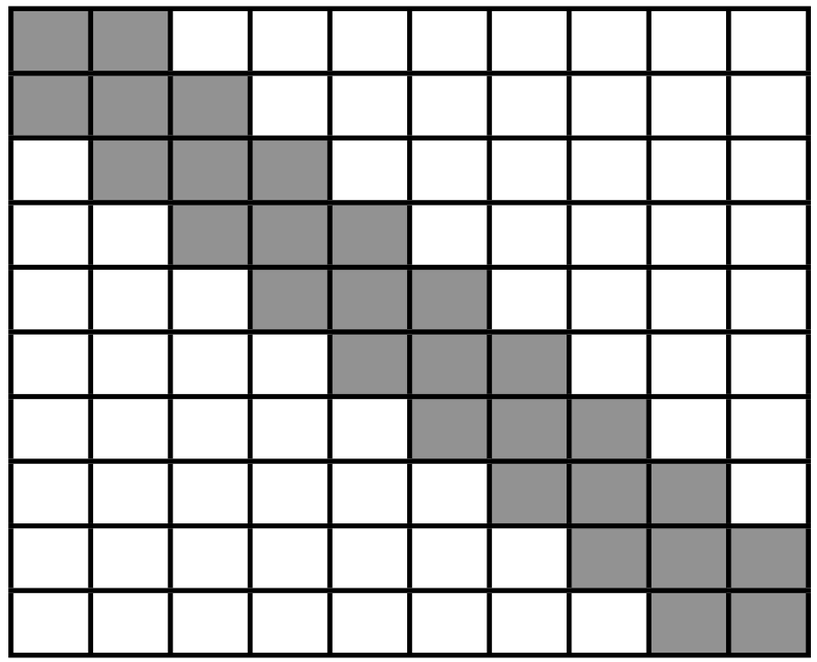
- Transform the matrix to compact form
  Upper hessenberg for non-symmetric,
  Tridiagonal for symmetric matrices
- Find eigenvalues of reduced matrix, e.g. by using QR iteration
- Backtransform, if eigenvectors are required

QR Iteration.
Compact matrix forms.

Full matrix → Householder transformations → Upper Hessenberg

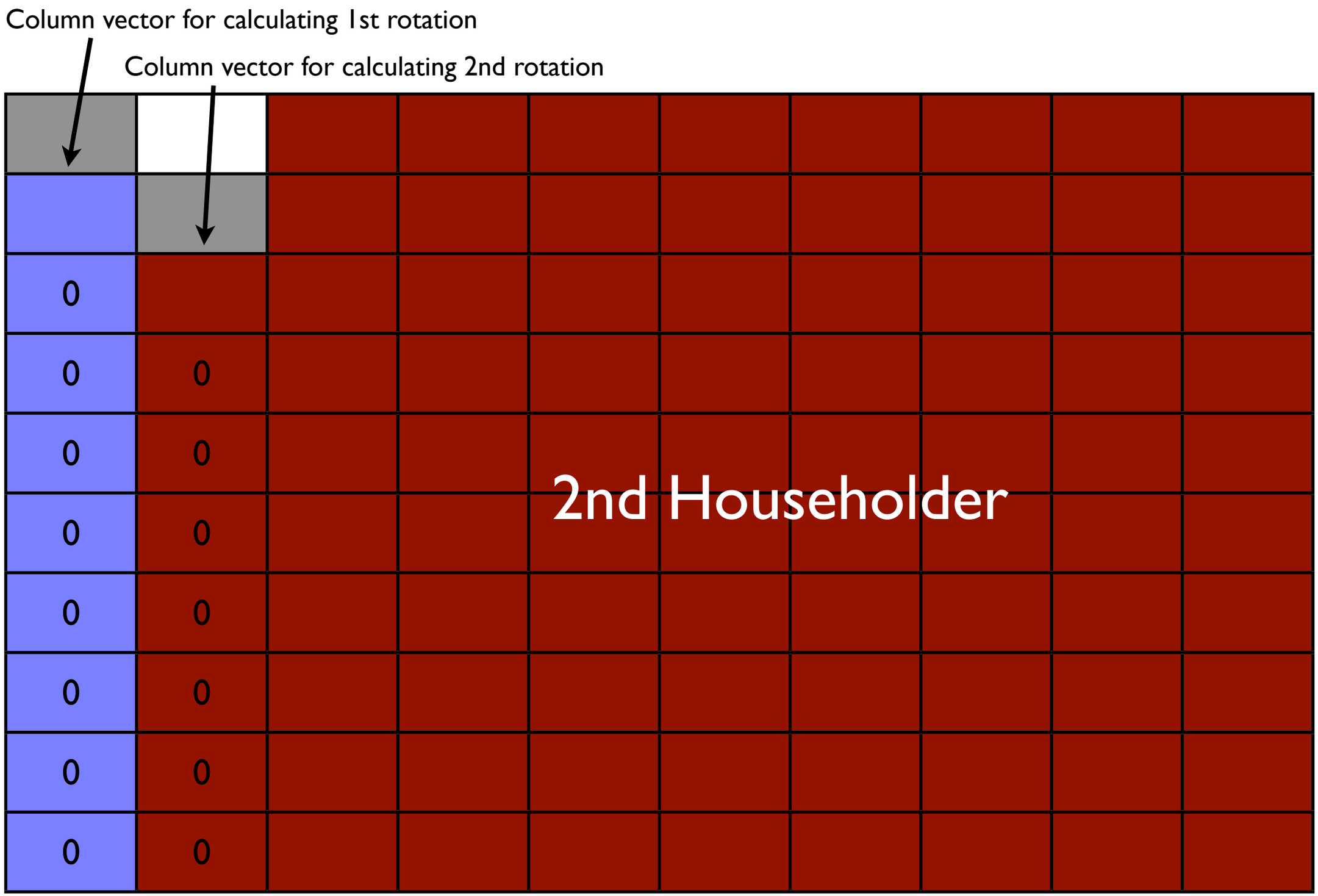


Symmetric matrix → Householder transformations → Tridiagonal



=> only O(n) Givens rotations required in each step

Householder Transformation.
Changed values at 2nd rotation.

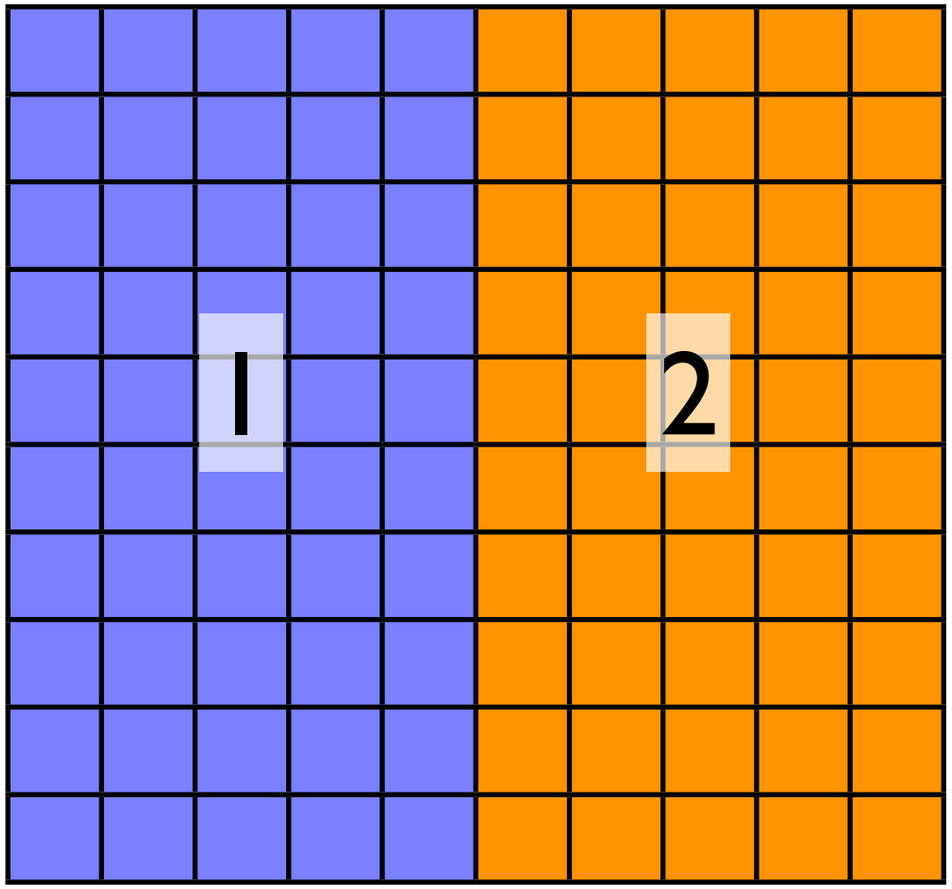


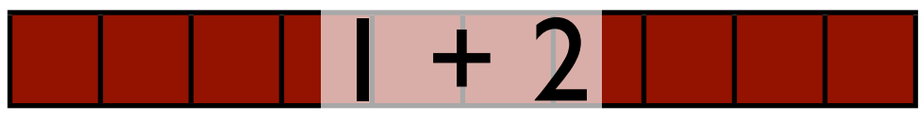
Householder Transformation.
Parallelization.

- Can we parellelize this? Yes we can!
- Application of Householder similarity transform consists of two matrix-vector products
- Matrix vector products can be parallelized easily (but inefficient)

Daniel
Kleine-Albers

Householder Transformation.
Matrix vector product parallelization.



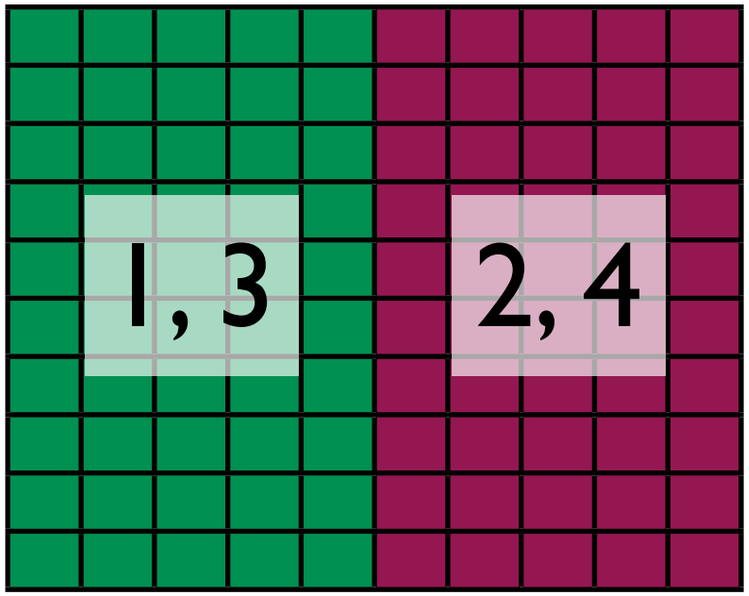


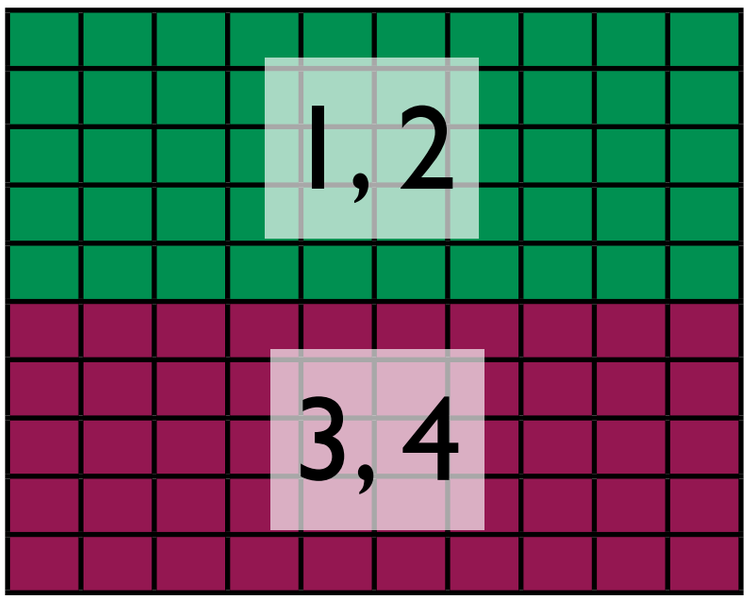
Blocked Householder Transformation. Parallelization.

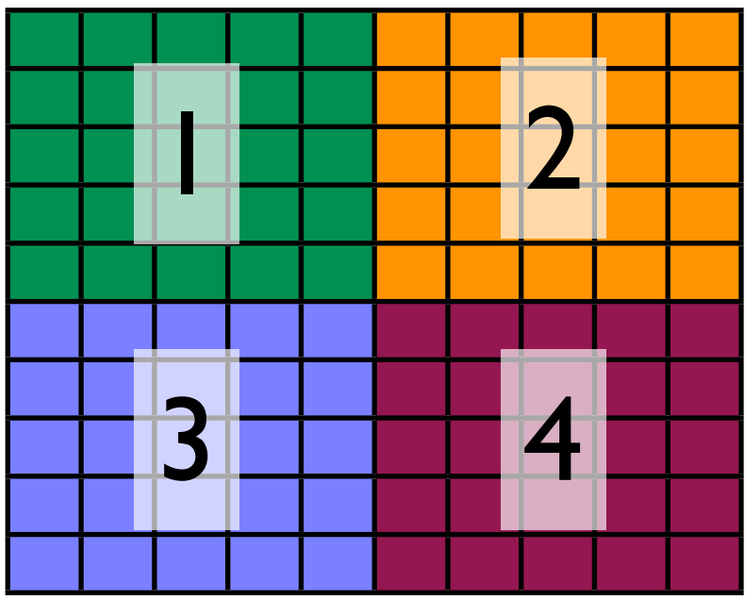
- For a blocked householder transformation the transformations are accumulated
- Application as matrix-matrix products, which can be parallelized good
- However - still a lot of matrix-vector operations required

Daniel
Kleine-Albers

Blocked Householder Transformation.
Matrix matrix product parallelization.







Daniel Kleine-Albers

QR Iteration.
Some timings.

Variant	Complexity	Time
Full QR	$O(n^3)$ per step	1467 ms
QR + non-block. transformation	$O(n^3)$ for redu. $O(n^2)$ per step	693 ms
QR + blocked transformation	$O(n^3)$ for redu. $O(n^2)$ per step	1017 ms
QR + blocked parallel transf.	$O(n^3)$ for redu. $O(n^2)$ per step	1109 ms

Measurement done with self-written Groovy code on Core 2 Duo 2GHz on a 10x10 Lotkin Matrix

Householder Reduction.
Some timings.

Variant	Complexity	Time
Non-Blocked	$O(n^3)$	256 ms
Blocked	$O(n^3)$	975 ms
Blocked Parallel	$O(n^3)$	552 ms

Measurement done with self-written Groovy code on Core 2 Duo 2GHz on a 10x10 Lotkin Matrix

Daniel
Kleine-Albers

QR Iteration.
Review.

- Parallelization of reduction is possible
- QR Iteration can not be parallelized good enough
- More optimization can be applied on symmetric matrices
- Reduction can be optimized using 2-step process
- Next step:
  Replace QR Iteration with something else

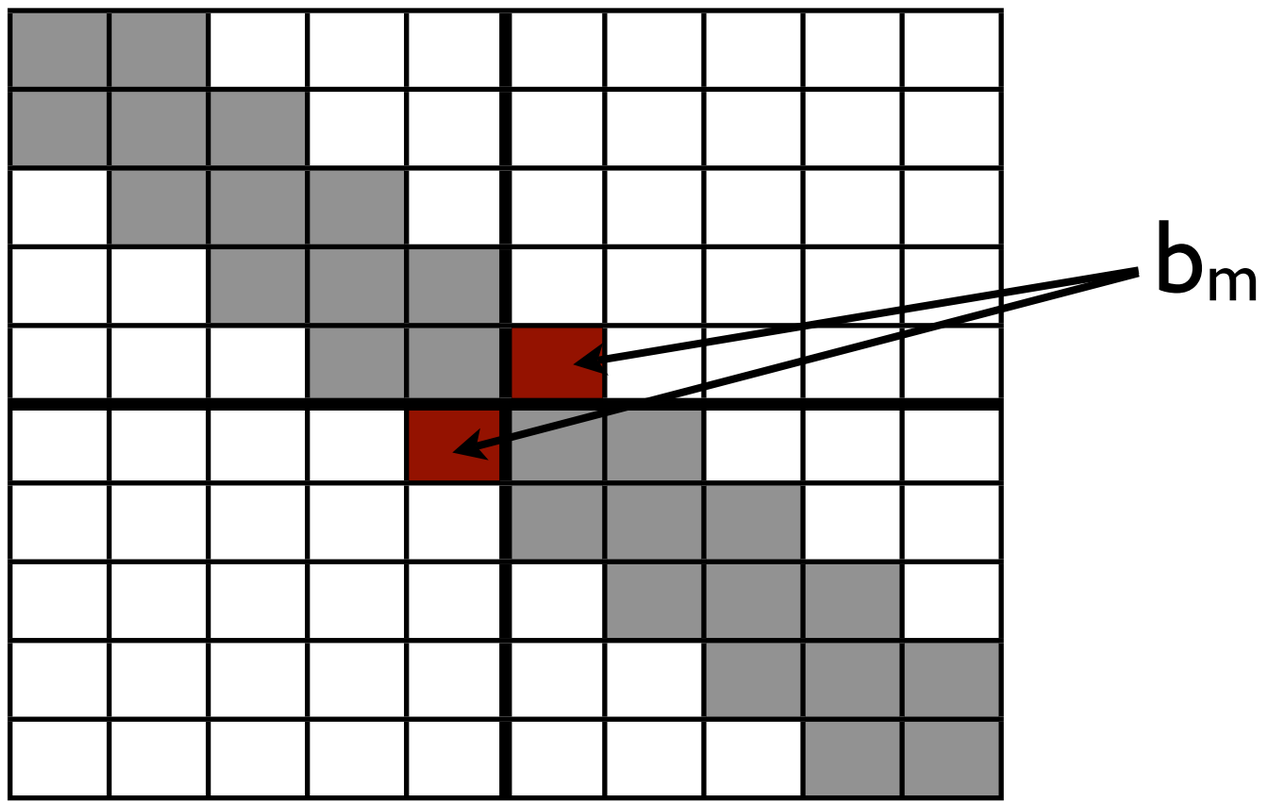
Daniel
Kleine-Albers

Computing eigenvalues in parallel.
Contents.

Daniel
Kleine-Albers

Divide and Conquer.
General idea.

- Only applicable on symmetric matrices
- Divide the (reduced) matrix into submatrices to calculate eigenvalues and -vectors



Daniel
Kleine-Albers

Divide and Conquer.

Divide.

$$\left[\begin{array}{cc|cc} \ddots & \ddots & & \\ \ddots & a_{m-1} & b_{m-1} & \\ & b_{m-1} & a_m - |b_m| & \\ \hline & & a_{m+1} - |b_m| & b_{m+1} \\ & & b_{m+1} & a_{m+2} \quad \ddots \\ & & \ddots & \ddots \end{array}\right] + \left[\begin{array}{cc|cc} & & & \\ & & |b_m| & b_m \\ \hline & & b_m & |b_m| \\ & & & \end{array}\right]$$

$$\left[\begin{array}{cc} T_1 & \\ & T_2 \end{array}\right] + |b_m| \, v \, v^T$$

Divide and Conquer.
Conquer.

- Mathematical formulation leads to seculiar equation
- Roots are the eigenvalues
  -> Zero finders used, e.g. Newton's method
- Deflation leads to fast results
- Eigenvectors can be computed easily when eigenvalues are known

Divide and Conquer.
Algorithm.

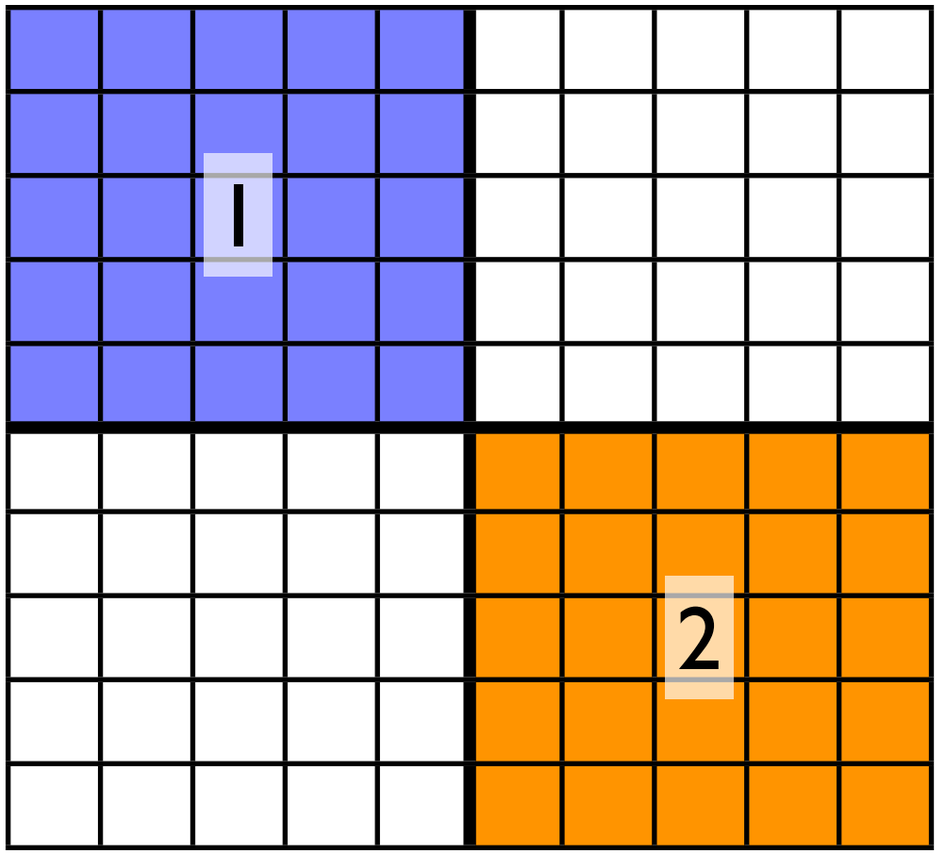
```
function [ Q,A ] = dc_eig(T)
      if T is 1x1
          return Q=1,A=T
      else
          split T to T1,T2
          [Q1,A1] = dc_eig(T1)
          [Q2,A2] = dc_eig(T2)
          based on A1,A2,Q1,Q2 find combined
                                  eigenvalues A and eigenvectors Q
          return Q,A
    endif
end
```

Can be done in parallel

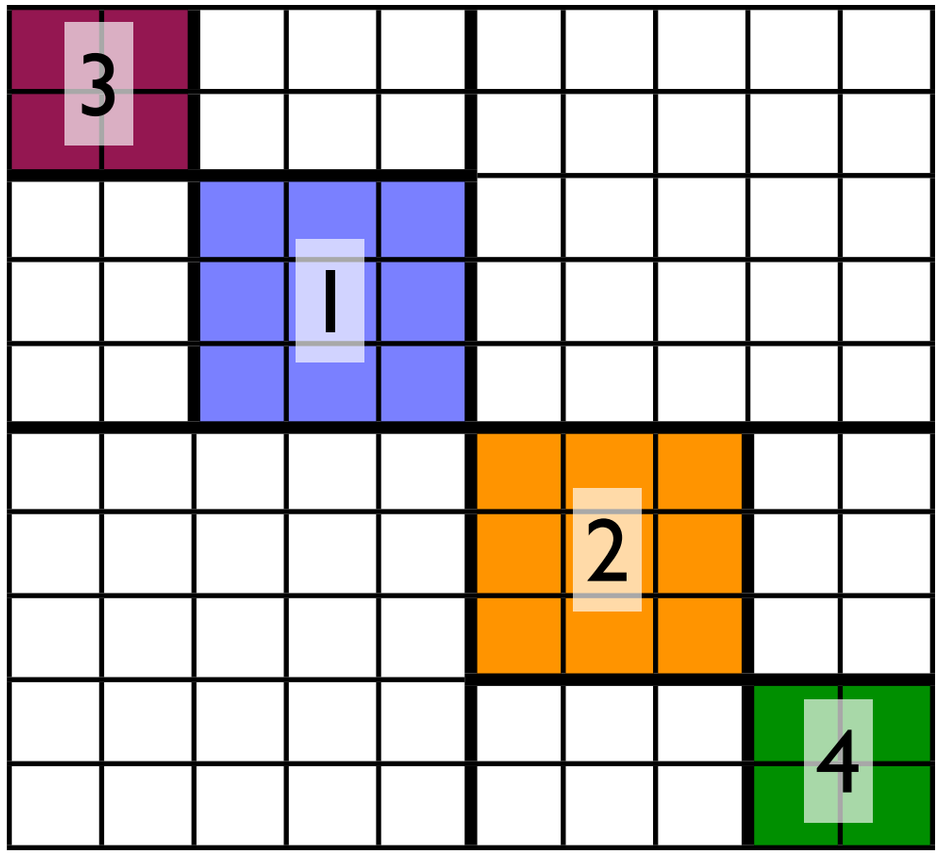
Daniel
Kleine-Albers

Divide and Conquer.
Parallelization.

Depth = 1



Depth = 2



Divide and Conquer.
Review.

- Parallelization of reduction as before
- Divide and Conquer can be parallelized naturally and very easy – “embarrassingly parallel”
- Also computes eigenvectors -> sometimes slower if you only want eigenvalues
- In practice: Used for n > 25
  smaller n: switch to QR iteration
- Extra memory required

Daniel
Kleine-Albers

Computing eigenvalues in parallel.
Contents.

Daniel
Kleine-Albers

MRRR.
Multiple relatively robust representations.

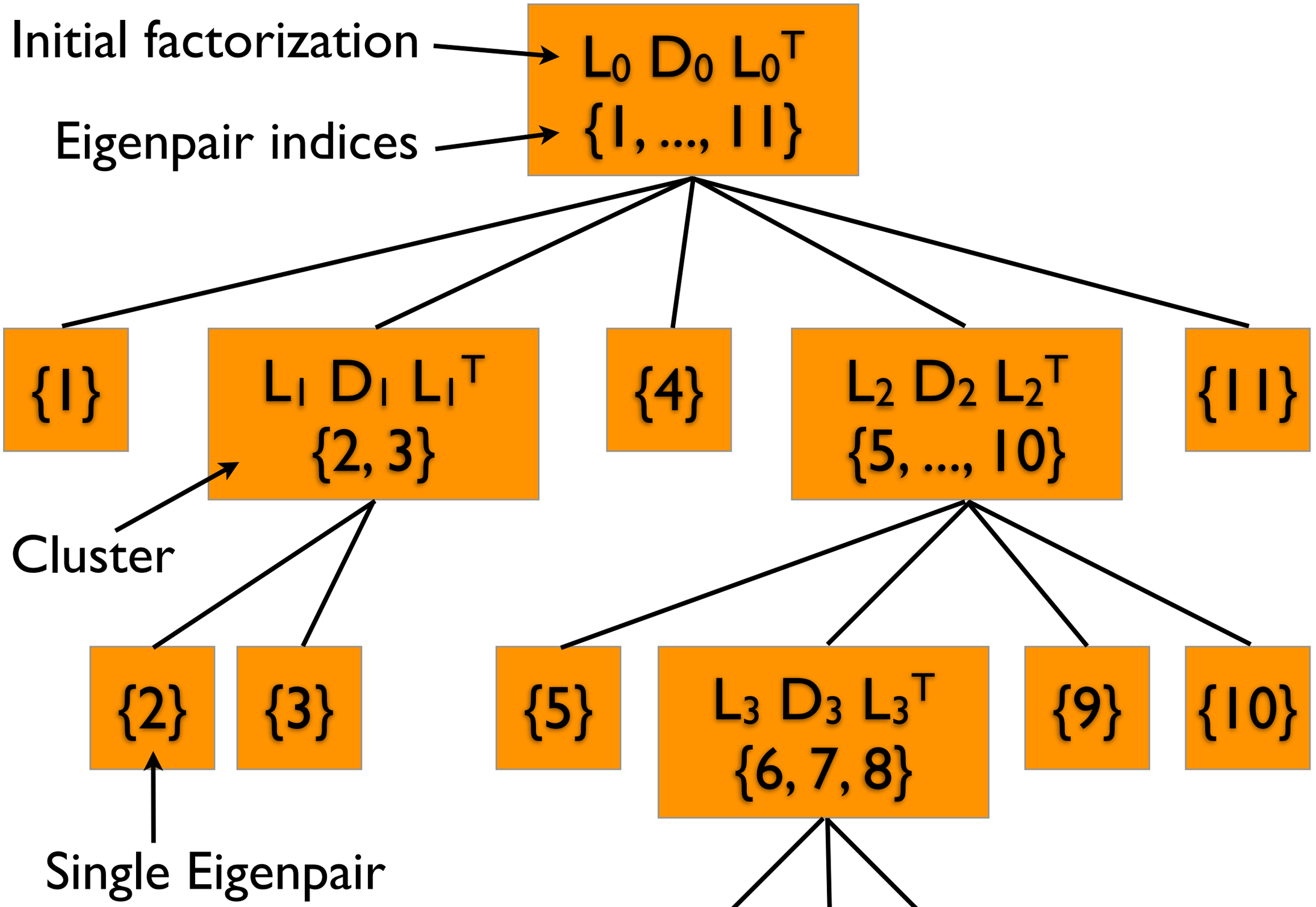
- Modern algorithm
- Applicable to symmetric matrices
- O(nk)

- Based on multiple LDL^T representations
- Calculated eigenvectors are numerically orthogonal without reorthogonalization
- Very high accuracy

MRRR.
New Representations.

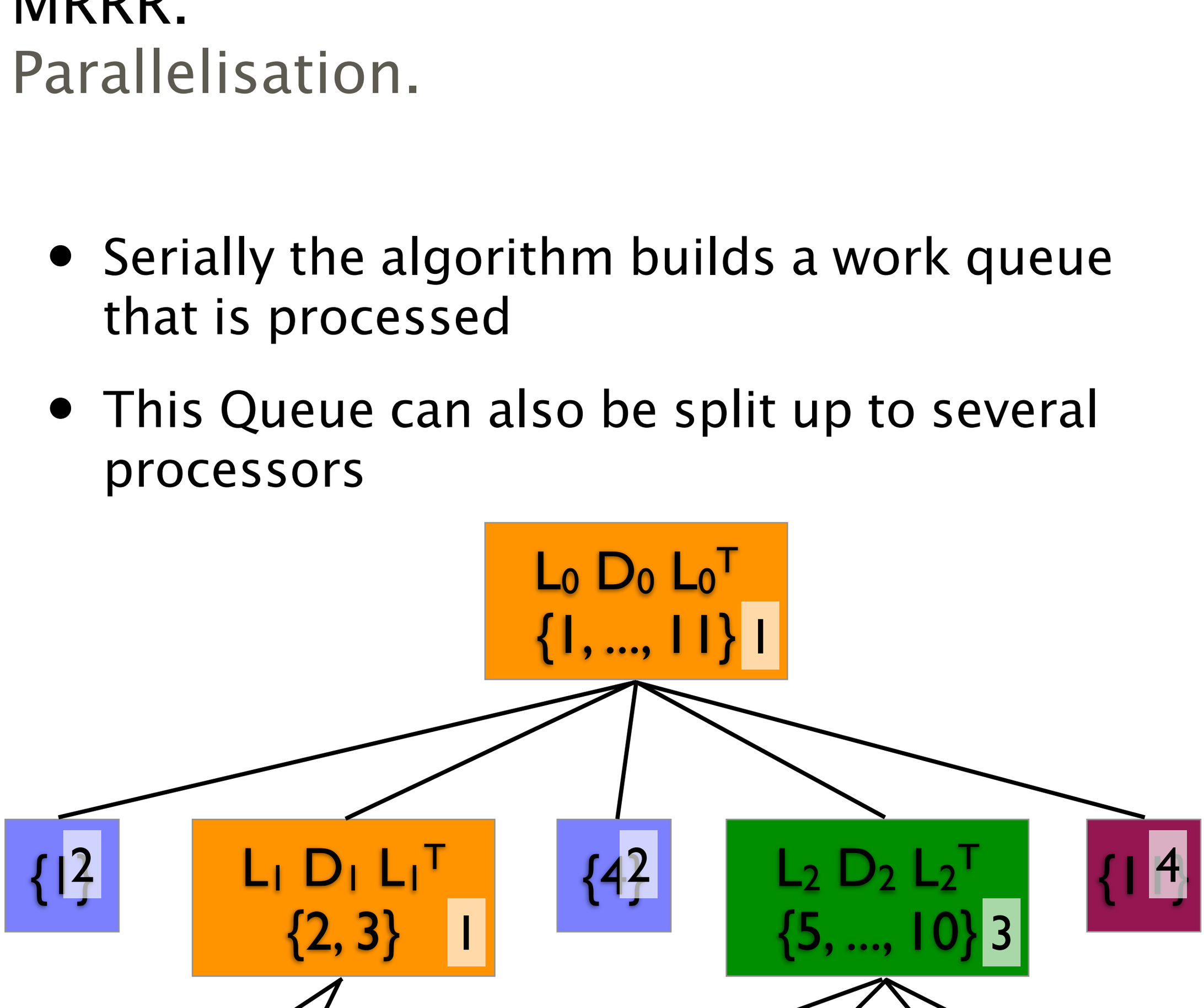
- For each cluster of eigenvalues a new LDL^T representation is build
  -> eigenvalues need to be “estimated”
- The new representation is a “child” of its “parent” representation
- $L_C D_C L_C^T = LDL^T - \tau I$
- τ is chosen to shift into the cluster

MRRR.
Representation tree.



MRRR.
Parallelisation.

- Serially the algorithm builds a work queue that is processed
- This Queue can also be split up to several processors



MRRR.
Review.

- O(nk)
- Calculated eigenvectors are numerically orthogonal without reorthogonalization
- Can be used for subsets of eigenpairs
- Very high accuracy

Computing eigenvalues in parallel. Conclusion.

- Three steps:
  1. Transformation to compact form
  2. Calculation of eigenvalues/-vectors
  3. Back-Transformation
- Transformation and Back-Transformation parallelizable using Matrix-Matrix-Products
- Three algorithms:
  - QR iteration (for all kinds of matrices)
  - Divide and Conquer (symmetric matrices)
  - MRRR (symmetric matrices)

Thank you for your attention.