8.7 Bisection for computing eigenvalues of a tridiagonal matrix

Observation: The characteristic polynomial of a tridiagonal matrix can be evaluated via the matrix entries in form of a sequence of polynomials with increasing degree:

$$p(\lambda) = \det(T - \lambda I) = \det \begin{pmatrix} \delta_1 - \lambda & \gamma_2 & 0 & \cdots & 0 \\ \gamma_2 & \delta_2 - \lambda & \gamma_3 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \gamma_{n-1} & \delta_{n-1} - \lambda & \gamma_n \\ 0 & \cdots & 0 & \gamma_n & \delta_n - \lambda \end{pmatrix}$$

$$p_0(\lambda) = 1$$

$$p_1(\lambda) = \delta_1 - \lambda$$

$$p_2(\lambda) = (\delta_2 - \lambda)p_1(\lambda) - \gamma_2^2 p_0(\lambda)$$

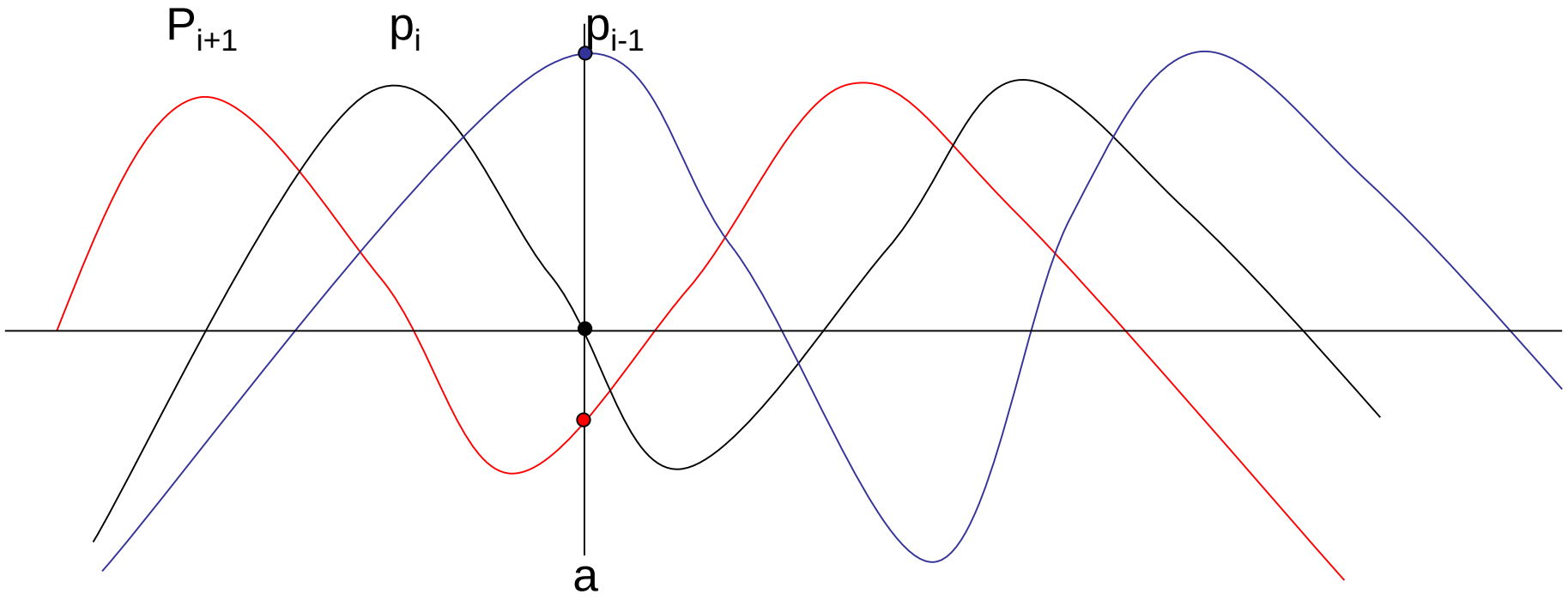
$$p_i(\lambda) = (\delta_i - \lambda)p_{i-1}(\lambda) - \gamma_i^2 p_{i-2}(\lambda), \qquad i = 3,4,\ldots,n$$

$$p(\lambda) = p_n(\lambda)$$

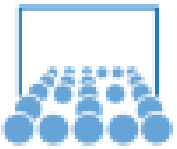
The sequence of polynomials is a Sturm chain:

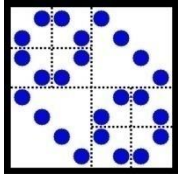
1. All p_i have only single zeros
2. $\text{sign}(p_{n-1}(a)) = -\text{sign}(p'_n(a))$ for all real zeros of $p_n(x)$
3. For i=1,2,…,n-1: $p_{i+1}(a)p_{i-1}(a) < 0$ for all real zeros of $p_i(x)$
4. The polynomial $p_0(x)$ does not change ist sign

Proof by induction.



At all zeros of p_i the neighbors p_{i-1} and p_{i+1} must have different sign.



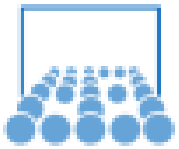


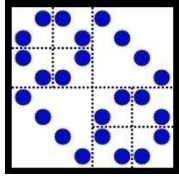
Define w(a):= # sign changes in $p_i(a)$, i=1,…,n .

It holds: w(a) = # zeros of $p_n(x)$ for x<a.

Consider eigenvalues ordered $\lambda_1 < \lambda_2 < \ldots < \lambda_{n-1} < \lambda_n$.
We want to find λ_i, the i-th zero of $p_n(x)$.

It holds: $\lambda_i < a \rightarrow$ w(a) = [# zeros left of a] >= i





Bisection Algorithm:

Choose an interval $I=[a_0, b_0]$ which contains λ_i.
Therefore: $w(b_0) >= i$ and $w(a_0) < i$.
Evaluate the polynomial sequence for $a=(a_0+b_0)/2$ and count the sign changes in the sequence $p_i(a) \rightarrow w(a)$.

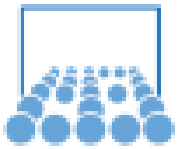
If $w(a) >= i$: Replace in interval I b_0 by a
Otherwise: Replace in interval I a_0 by a.

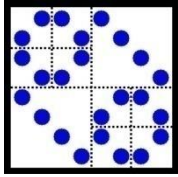
Generates converging sequence of smaller and smaller intervals that contain the eigenvalue λ_i certainly.

Advantages:

- can be easily parallelized on top level
- can be used with high or low accuracy
- gives only eigenvalues

Allows also Newton method for zeros of p_n





8.8 MRRR for eigenvectors

Multiple Relatively Robust Representations

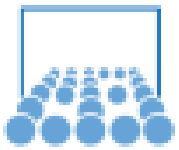
Idea: Given tridiagonal matrix.
Assume the eigenvalues to be computed fast in parallel e.g. via Bisection to high accuracy.

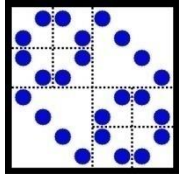
Use inverse iteration for computing the eigenvectors to high accuracy.

Observations:

Inverse iteration is cheap, because of tridiagonal form! (??)

But eigenvalue cluster lead to low accuracy in eigenvectors!





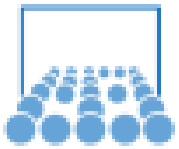
Outline

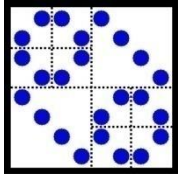
1. Choose μ such that $T + \mu I$ is positive definite.
2. Compute Cholesky factorization $T + \mu I = L D L^T$.
3. Compute eigenvalues of $L D L^T$ to high relative accuracy (Bisection)
4. Group eigenvalues according to their relative gap

   a) Isolated eigenvalue: Compute eigenvector via twisted factorization and Inverse Iteration

   b) Cluster:

   - Choose μ near cluster and compute $L D L^T - \mu I = L_1 D_1 L_1^T$.
   - Refine eigenvalues in cluster to high relative accuracy
   - Set $L \leftarrow L_1$, $D \leftarrow D_1$.
   - Repeat step 4 for eigenvalues in this cluster.





Twisted factorization: $T - \hat{\lambda} I = N_r D_r N_r^T$

$$N_r = \begin{pmatrix} x & & & & & & \\ x & x & & & & & \\ & . & . & & & & \\ & & x & \gamma_r & x & & \\ & & & & . & . & \\ & & & & & x & x \\ & & & & & & x \end{pmatrix}$$

r is chosen to minimize $|\gamma_r|$.
Guarantuees fast convergence of Inverse Iteration.

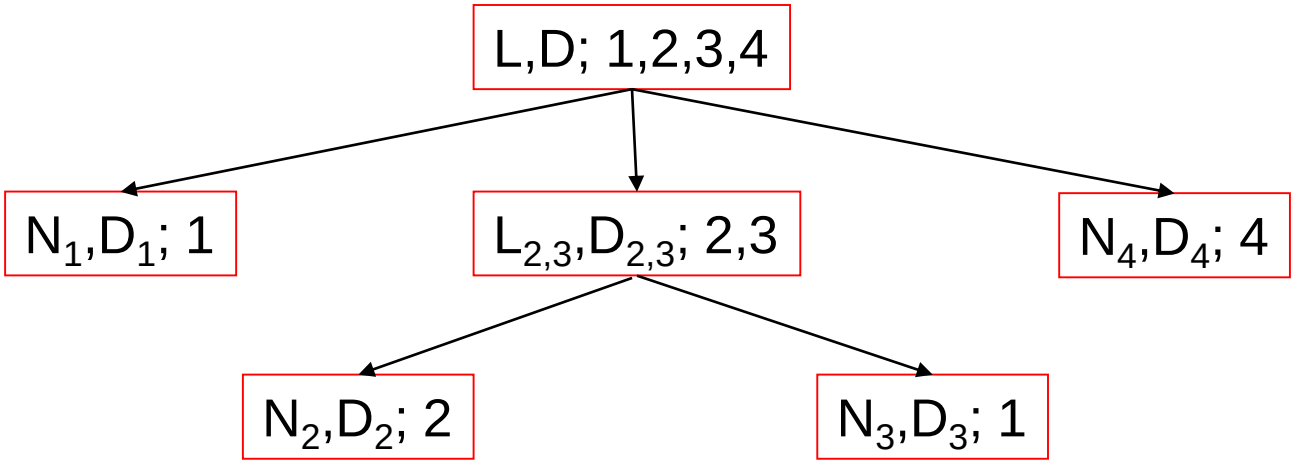
Conclusion

MR^3 allows the computation of eigenvectors with high accuracy

(also for clustered eigenvalues)

using only small number of inverse iteration steps.

Results in tree relative to clusters:



Use fixed point iteration (ILU) for Cholesky and twisted systems.

8.9 Sequential QR Algorithm for computing all Eigenvalues:

Standard algorithm for computing eigenpairs: QR-algorithm

Prestep: Transform A by Givens or Householder matrices to tridiagonal form.

$$G_{2,3} * \left(\begin{array}{ccc|cc} a_{11} & a_{12} & \boxed{a_{13}} & * & * \\ \hline a_{21} & a_{22} & a_{23} & * & * \\ \boxed{a_{31}} & a_{32} & a_{33} & * & * \\ \hline * & * & * & * & * \\ * & * & * & * & * \end{array}\right) * G_{2,3}^{H}$$

to eliminate a_{31} and a_{13}

Main difference to QR-factorization:

- Use subdiagonal entry for eliminating elements
- Apply Q from both sides
- Gives tridiagonal matrix (or upper Hessenberg for nonsymymetric A) .

For better parallelism use block Householder like in the QR-decomposition.

QR-Algorithm

First step:

By Householder matrices transform A by equivalence transformations on tridiagonal (upper Hessenberg) form: $A \rightarrow H \cdot A \cdot H^T = T$

For the following we assume A already tridiagonal (upper Hessenberg)

Second step: Compute QR-decomposition of A, $A = QR$ and

replace $A = A_{old}$ by $A_{new} = RQ$

$$A_{new} = RQ = (Q^T A)Q = Q^T AQ$$

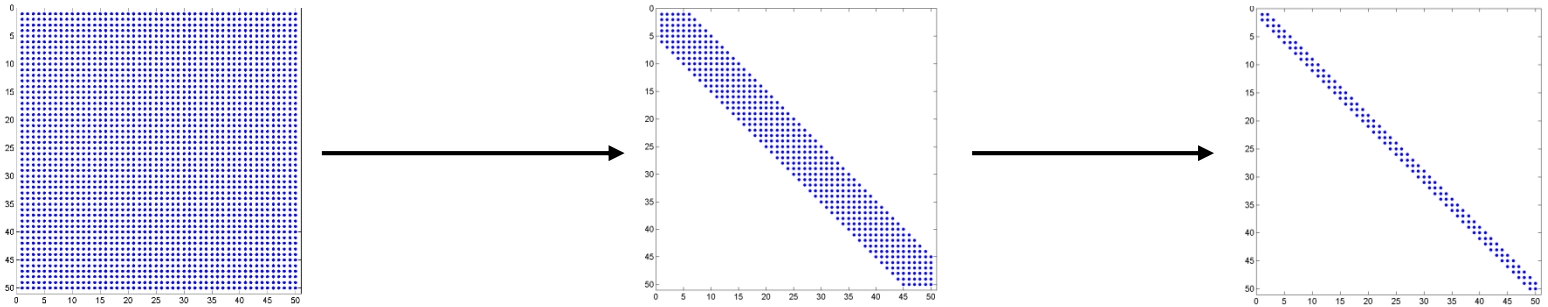
Therefore A and A_{new} have the same eigenvalues

Repeat these QR-steps until convergence against diagonal (upper triangular) matrix.
Use last diagonal entry r as shift A-rI, apply QR step on shifted matrix.

8.10 Twostep Tridiagonalization

Reduce full matrix to tridiagonal (upper Hessenberg)
Sequential BLAS1!

For allowing better parallelism reduce matrix A to
block-banded form, and then in a second step to tridiagonal form.



Advantage:
First step allows block/BLAS3 operations and is good in parallel.
second step is cheap; can be implemented e.g. by MR^3 or D&C.

Disadvantage: For eigenvectors 2 transformations necessary

Bothsided Householder for Tridiagonalization

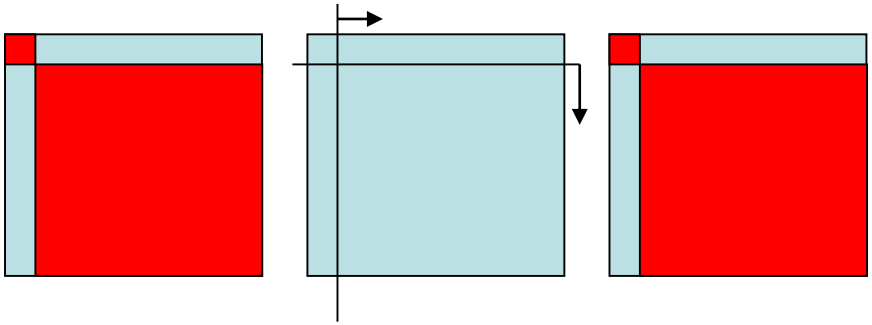
Compute Householder vector u in order to eliminate subtridiagonal entries in the first column/row.

Apply

$$A \rightarrow (I-2uu^H)A(I-2uu^H) = A - 2u(u^HA) - 2(Au)u^H + 4\,uu^H(u^HAu) =$$

$$= A - 2u(u^HA+ru^H) - 2(Au+ru)u^H =$$

$$= A - uy^H - yu^H$$



Two matrix update steps

To reduce BLAS2 operations work blockwise,

$A \rightarrow A - UY^H - YU^H$ (BLAS3)

but still first Au is needed (BLAS2).

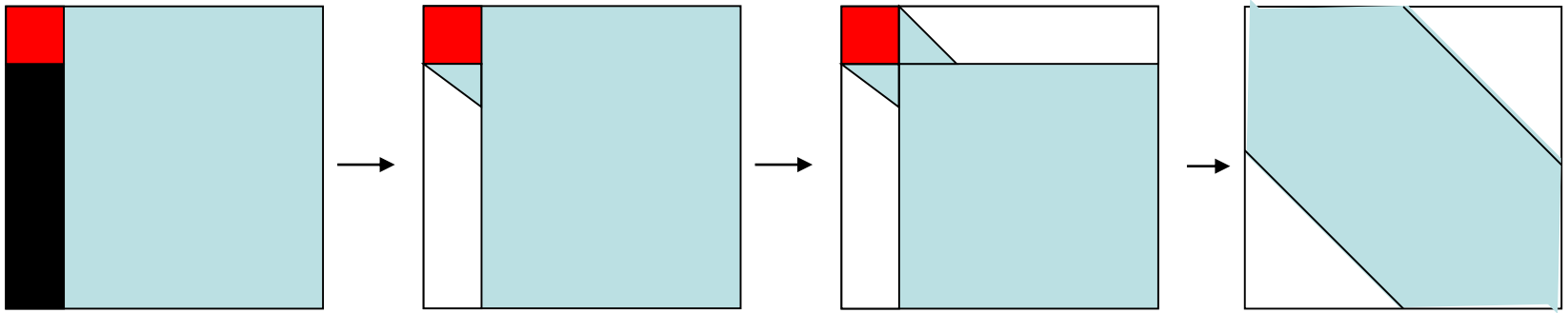
Block-Band reduction

In the first step find QR decomposition of subblock $A(1 + b : n, 1 : n_b) = A_1$ where b is the bandwidth and n_b is a block size.

Compute QR decomposition of black part A_1:

Applying (I , Q^H) from the left leads to triangular form of black part.

Applying from both sides: Band structure.

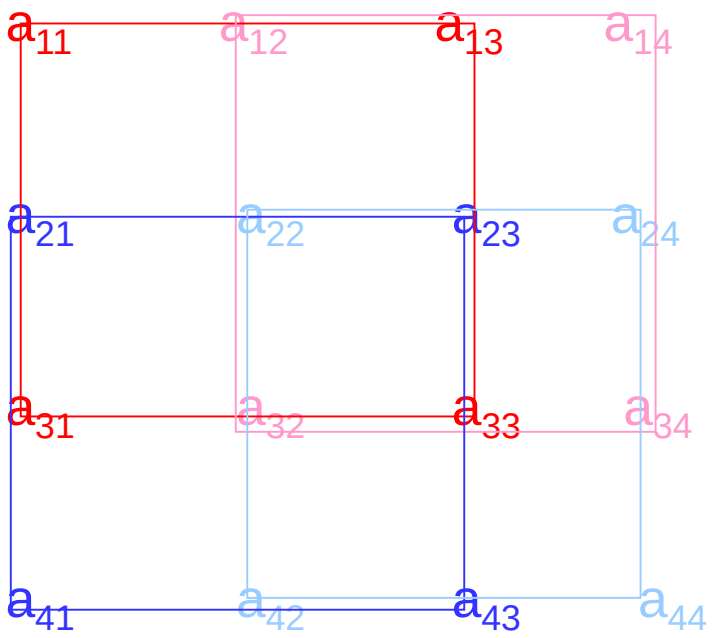


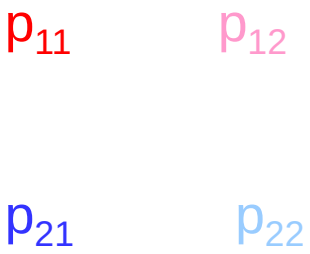
Store Householder vectors on positions of new generated zeros.

Use Tall&Skinny or Cholesky QR.

2D-Cyclic Data Distribution

4 x 4 – Matrix on 2 x 2 processor array

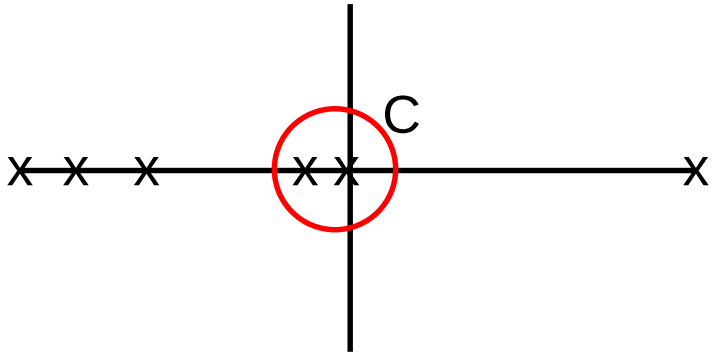




Advantage: better load balancing because matrices and Householder vectors are getting smaller.

8.11 FEAST

Use integration over closed curve C in complex plane in order to derive an approximation to the subspace built by the eigenvectors related to the eigenvalues inclosed by the curve.



Closed curve contains 2 eigenvalues with 2 (orthogonal) eigenvectors → 2-dim subspace

FEAST c't

$$U := \frac{1}{2\pi i}\int_C (zI - A)^{-1} Y \quad dz$$

For rank 2 matrix Y, the computed matrix U contains the span of the 2 eigenvectors in C.

With U computed build the small matrices

$$A_U = U^H A U, \qquad B_U = U^H U$$

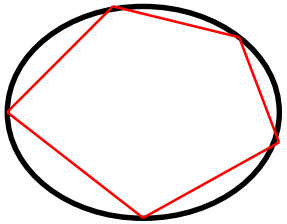
and solve the small eigenvalue problem

$$A_U W = B_U W \cdot \Lambda$$

Repeat with Y = X = U*W until convergence.

FEAST c't

Main work: Use quadrature rule with discretization points z_j , j=1,…,p, in C to compute the integral.



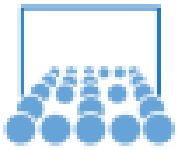
Therefore, we need to solve

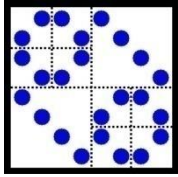
$$\left(z_j I - A\right) U_j = Y$$

for different z_j and blocks $Y=(y_1,\ldots,y_m)$

$$U = \frac{1}{2\pi i}\sum_{j=1}^{p} \omega_j \varphi(t_j)\left(\varphi(t_j) I - A\right)^{-1} Y =$$

$$= \frac{1}{2\pi i}\sum_{j=1}^{p} \omega_j z_j \left(z_j I - A\right)^{-1} Y$$





Advantages:

First level of parallelism:

Choose different curves containing all wanted eigenvalues.

Problem: Gap

Second level of parallelism:

Solve linear equations for different z_j and

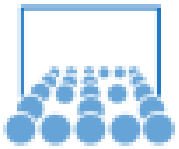
for one z_j for different columns of Y

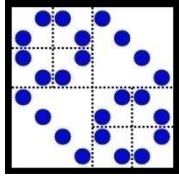
Problem: Might be near singular for z_j close to eigenvalue

Third level of parallelism:

Parallelize iterative solver.

Problem: Convergence for sparse iterative solver





Problem

Linear equations are extremely ill-conditioned if eigenvalues are close to curve and therefore $z_j I - A$ very ill-conditioned.

Slow convergence of iterative solver!

Preconditioning?

Large eigenvalue clusters. How to separate?

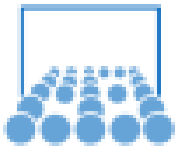
Generalized Eigenvalue Problem:

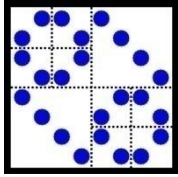
$$Ax = \lambda Bx, A = A^H, B = B^H > 0, spd$$

Compute Cholesky decomposition $B=L^HL$ and solve

$$Ax = \lambda L^H Lx \Leftrightarrow \left(L^{-H} AL^{-1}\right)Lx = Lx \Leftrightarrow$$

$$\Leftrightarrow \left(L^{-H} AL^{-1}\right)y = y, y = Lx$$





Different Questions:

Sparse or dense?

Eigenvectors or not?

All eigenvalues or only a few?

Interior eigenvalues?

Clustered eigenvalues?

Real or complex?

Symmetric or not?

Generalized or not?

Fixed point iteration for Cholesky factorization of a tridiagonal matrix

$$\begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & b_2 & \\ & b_2 & a_3 & b_3 \\ & & b_3 & a_4 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ c_2 & 1 & & \\ & c_3 & 1 & \\ & & c_4 & 1 \end{pmatrix} \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & d_3 & \\ & & & d_4 \end{pmatrix} \begin{pmatrix} 1 & c_2 & & \\ & 1 & c_3 & \\ & & 1 & c_4 \\ & & & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} d_1 & d_1 c_2 & & \\ d_1 c_2 & d_2 + d_1 c_2^2 & d_2 c_3 & \\ & d_2 c_3 & d_3 + d_2 c_3^2 & d_3 c_4 \\ & & d_3 c_4 & d_4 + d_3 c_3^2 \end{pmatrix}$$

Function with zero:

$$0 = f(d_1, d_2, d_3, d_4, c_2, c_3, c_4) = \begin{pmatrix} d_1 - a_1 \\ d_2 + d_1 c_2^2 - a_2 \\ d_3 + d_2 c_3^2 - a_3 \\ d_4 + d_3 c_4^2 - a_4 \\ d_1 c_2 - b_1 \\ d_2 c_3 - b_2 \\ d_3 c_4 - b_3 \end{pmatrix}$$

Derivative J of f:

$$J = \left(\begin{array}{cccc|ccc} 1 & & & & 0 & & \\ c_2^2 & 1 & & & 2d_1c_2 & & \\ & c_3^2 & 1 & & & 2d_2c_3 & \\ & & c_4^2 & 1 & & & 2d_3c_4 \\ \hline c_2 & & & & d_1 & & \\ & c_3 & & & & d_2 & \\ & & c_4 & 0 & & & d_3 \end{array}\right) = \left(\begin{array}{cccc|ccc} 1 & & & & 0 & & \\ c_2^2 & 1 & & & 2c_2 & & \\ & c_3^2 & 1 & & & 2c_3 & \\ & & c_4^2 & 1 & & & 2c_4 \\ \hline c_2 & & & & 1 & & \\ & c_3 & & & & 1 & \\ & & c_4 & 0 & & & 1 \end{array}\right) \left(\begin{array}{ccccccc} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & d_1 & & \\ & & & & & d_2 & \\ & & & & & & d_3 \end{array}\right)$$

To solve in every Newton step:

$$\tilde{J} = \left(\begin{array}{cccc|cccc} 1 & & & & 0 & & & \\ c_2^2 & 1 & & & 2c_2 & & & \\ & c_3^2 & 1 & & & 2c_3 & & \\ & & c_4^2 & 1 & & & & 2c_4 \\ \hline c_2 & & & & 1 & & & \\ & c_3 & & & & 1 & & \\ & & c_4 & 0 & & & & 1 \end{array}\right) = \begin{pmatrix} E & C \\ D & I \end{pmatrix}$$

with Schur complement S

$$S = E - CD = \begin{pmatrix} 1 & & & \\ c_2^2 & 1 & & \\ & c_3^2 & 1 & \\ & & c_4^2 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ 2c_2 & & \\ & 2c_3 & \\ & & 2c_4 \end{pmatrix} \begin{pmatrix} c_2 & & 0 \\ & c_3 & 0 \\ & & c_4 \\ & & 0 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ -c_2^2 & 1 & & \\ & -c_3^2 & 1 & \\ & & -c_4^2 & 1 \end{pmatrix}$$

Bidiagonal Equations to solve in every Newton step in E and S: $x_{k+1} = x_k - inv(J)\, f(x_k)$

$$\begin{pmatrix} 1 & & & \\ -c_2^2 & 1 & & \\ & -c_3^2 & 1 & \\ & & -c_4^2 & 1 \end{pmatrix} x = g$$

and a second trivial linear system in I.

Bidiagonal Solves: (1) Iteratively, stationary with preconditioner ISAI or Block Jacobi:

$$Lx = b \Rightarrow PLx = Pb \Rightarrow x_{j+1} = P(b - Lx_j) + x_j$$

or

$$Lx = b \Rightarrow LPP^{-1}x = b \Rightarrow LPy = b, x = Py \Rightarrow y_{j+1} = (b - LPy_j) + y_j$$

$P1_k$ is either ISAI to bidiagonal L with pattern L^k or lower band width k

or

$P2_k$ is block bidiagonal, setting every k-th subdiagonal entry in L to zero.

ISAI-Preconditioner (A-LL‘)_S=0_S

$$\begin{pmatrix} 1 & & & \\ -c_2^2 & 1 & & \\ & -c_3^2 & 1 & \\ & & -c_4^2 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & & \\ c_2^2 & 1 & & \\ -c_2^2 c_3^2 & c_3^2 & 1 & \\ c_2^2 c_3^2 c_4^2 & -c_3^2 c_4^2 & c_4^2 & 1 \end{pmatrix}$$

Bidiagonal direct solve with Sherman-Morrisson Woodbury:

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & L_3 & \\ & & & L_4 \end{pmatrix} + c_{k+1} e_{k+1} e_k^T + c_{2k+1} e_{2k+1} e_{2k}^T + c_{3k+1} e_{3k+1} e_{3k}^T =$$

$$= \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & L_3 & \\ & & & L_4 \end{pmatrix} + \begin{pmatrix} c_{k+1} e_{k+1} & c_{2k+1} e_{2k+1} & c_{3k+1} e_{3k+1} \end{pmatrix} \begin{pmatrix} e_k & e_{2k} & e_{3k} \end{pmatrix}^T =$$

$$= \tilde{L} + CE$$

Inverse of Sherman-Morrisson-Woodbury:

$$L^{-1}b = \tilde{L}^{-1}b - \tilde{L}^{-1}C\left(I_3 + E\tilde{L}^{-1}C\right)^{-1}E\tilde{L}^{-1}b$$

To solve linear systems in Ltilde with b and C.

7. Discrete Fourier Transform

$$\bar{\omega} = conj(\omega) = \exp(-2\pi i / n)$$

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \bar{\omega} & \bar{\omega}^2 & \cdots & \bar{\omega}^{n-1} \\ 1 & \bar{\omega}^2 & \bar{\omega}^4 & \cdots & \bar{\omega}^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \bar{\omega}^{n-1} & \bar{\omega}^{2(n-1)} & \cdots & \bar{\omega}^{(n-1)(n-1)} \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-1} \end{pmatrix}$$

$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} v_j \bar{\omega}^{jk}, \qquad k = 0,1,\ldots,n-1$$

c = DFT (v). Vector c is the Discrete Fourier Transform of vector v.

Important application: values → coefficients, frequency analysis

Discrete Fourier Transform

$$c_k = \frac{1}{n}\sum_{j=0}^{n-1} v_j \overline{\omega}^{jk}, \qquad k = 0,1,\ldots,n-1$$

DFT is nothing else than matrix times vector or n inner products.

Therefore, costs sequentially: $O(n^2)$
in parallel: n*log(n) processors, log(n) time steps by n independent fan-in processes.

DFT is very important in many applications.

Therefore, fast algorithms have been developed by divide-and-conquer

→ FFT = Fast Fourier Transform reduces costs O(n log(n)) sequentially.

Odd – even Partitioning

$$v_j = \sum_{k=0}^{n-1} c_k \cdot \exp\left(\frac{2\pi i j k}{n}\right) =$$

$$= \sum_{k=0}^{n/2-1} c_{2k} \cdot \exp\left(\frac{2\pi i j\, 2k}{n}\right) + \sum_{k=0}^{n/2-1} c_{2k+1} \cdot \exp\left(\frac{2\pi i j (2k+1)}{n}\right)$$

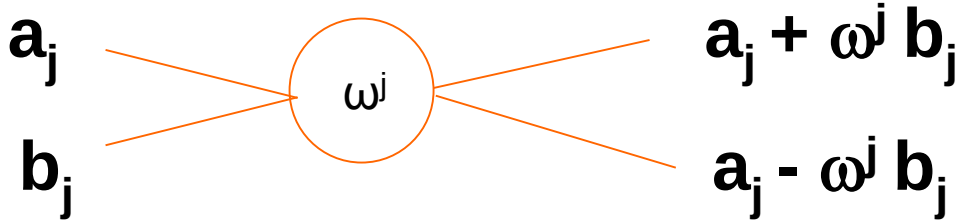
$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2\pi i j k}{m}\right) + \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2\pi i j k}{m}\right)$$

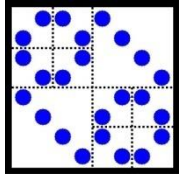
$$v_{m+j} =$$

$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2i\pi (j+m) k}{m}\right) + \omega^{j+m} \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2i\pi (j+m) k}{m}\right)$$

$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2ijk\pi}{m}\right) - \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2ijk\pi}{m}\right)$$

Butterfly:

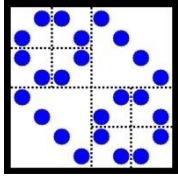


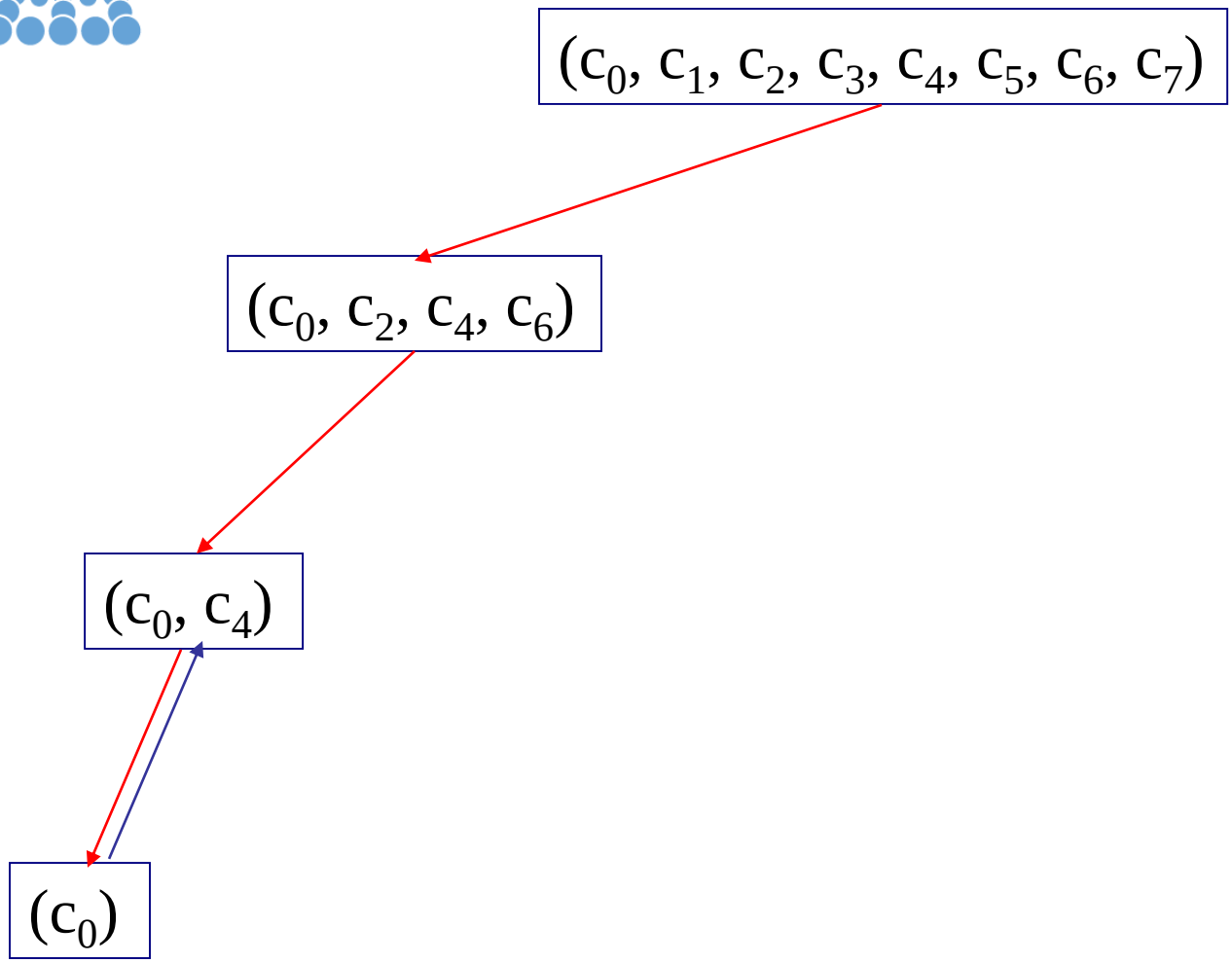


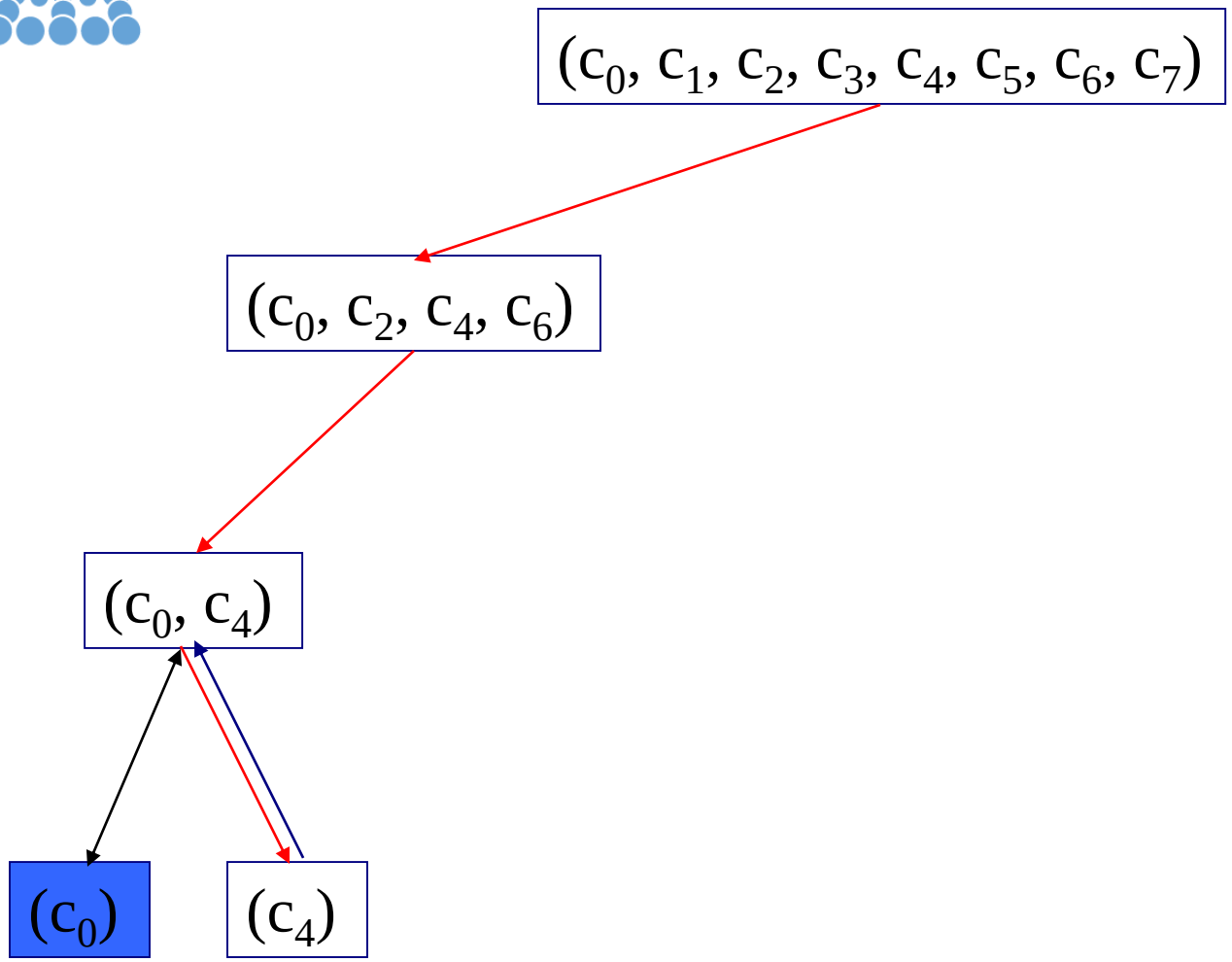
FFT: Recursive Algorithm

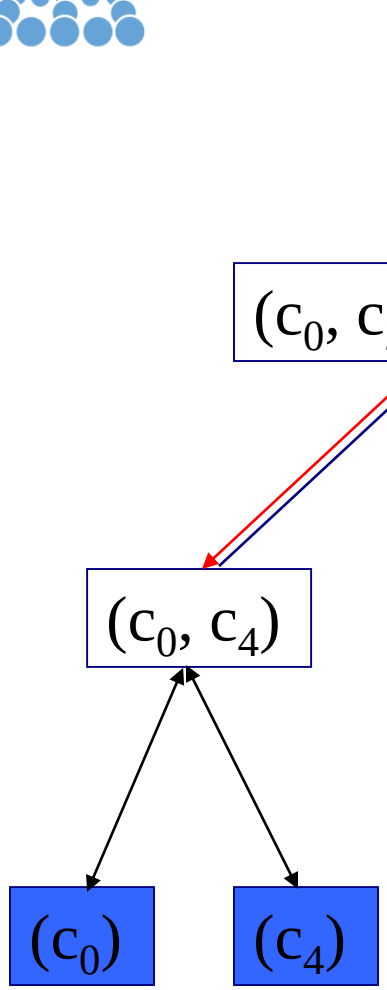
Partitioning the sums in the IDFT in odd and even coefficients delivers the first and the second part of v = IDFT(c):

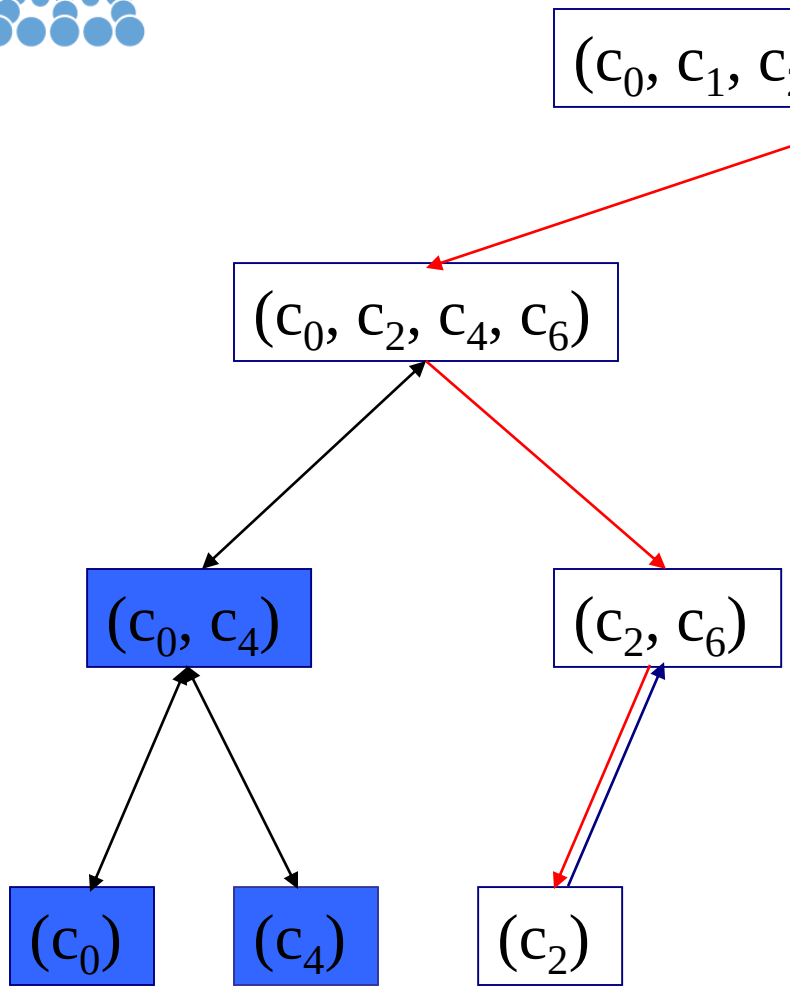
FUNCTION(v_0,...,v_{n-1}) = IDFT(c_0,…,c_{n-1},n)
    IF n==1
        $v_0 = c_0$;
    ELSE
        m=n/2 ;
        (g_0,…, g_{m-1}) = IDFT(c_0, c_2,…, c_{n-2},m) ;
        (u_0,…, u_{m-1}) = IDFT(c_1, c_3,…, c_{n-1},m) ;
        $\omega = \exp(2i\pi/n)$;
        FOR j=0:m-1
            $v_j = g_j + \omega^j u_j$;
            $v_{j+m} = g_j - \omega^j u_j$;
        END
    END

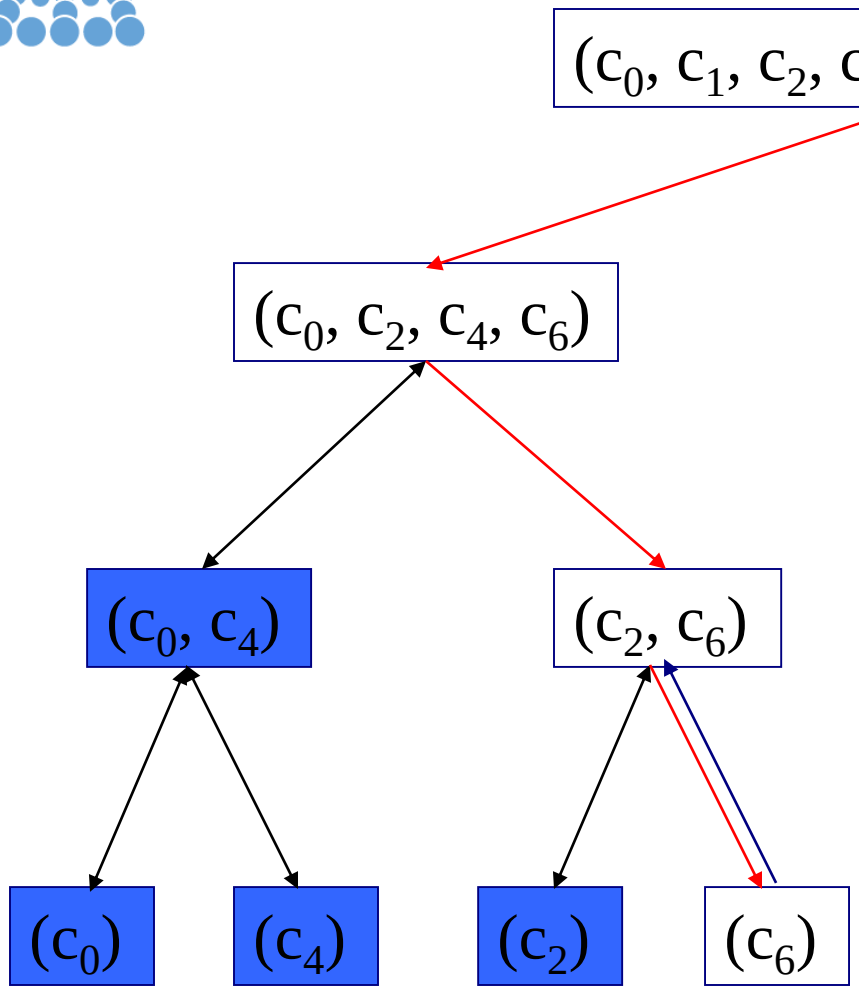


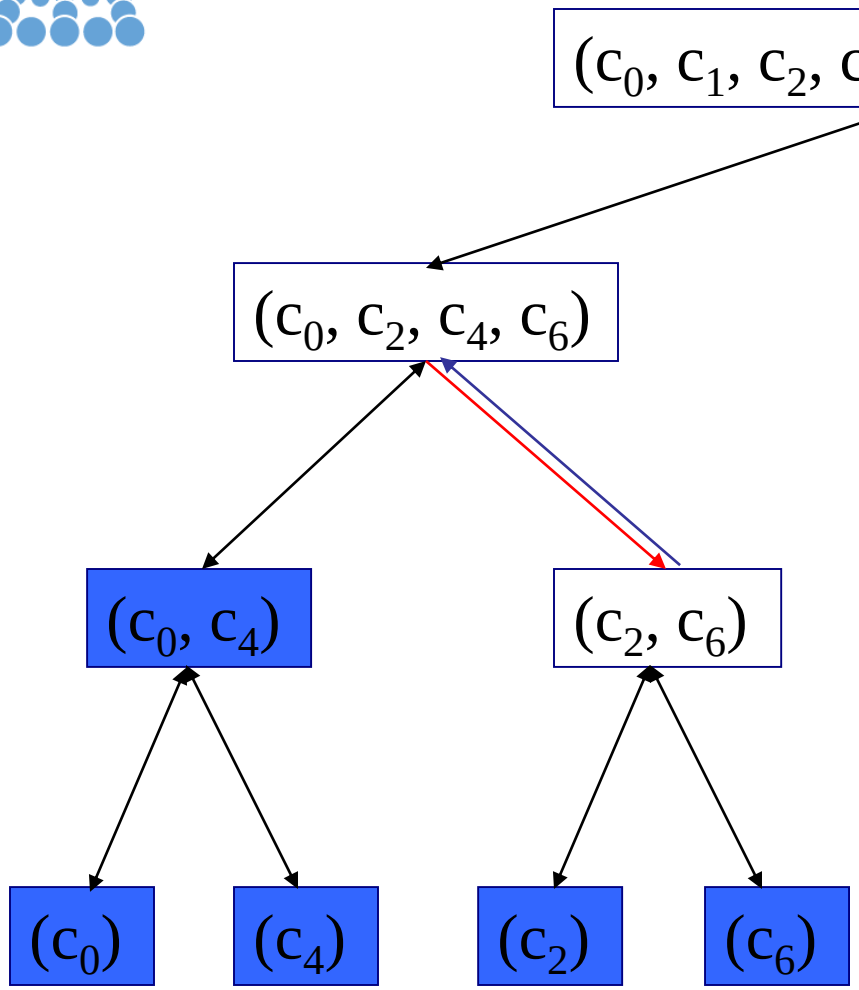

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_0)

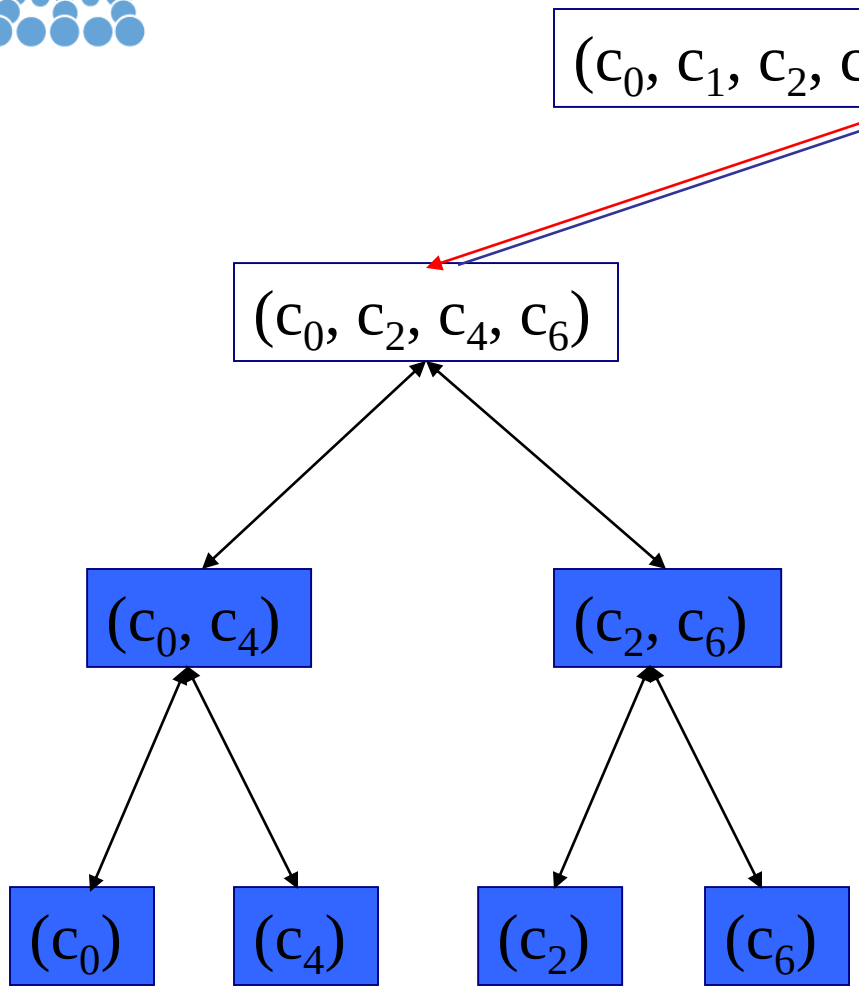

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_0)
(c_4)

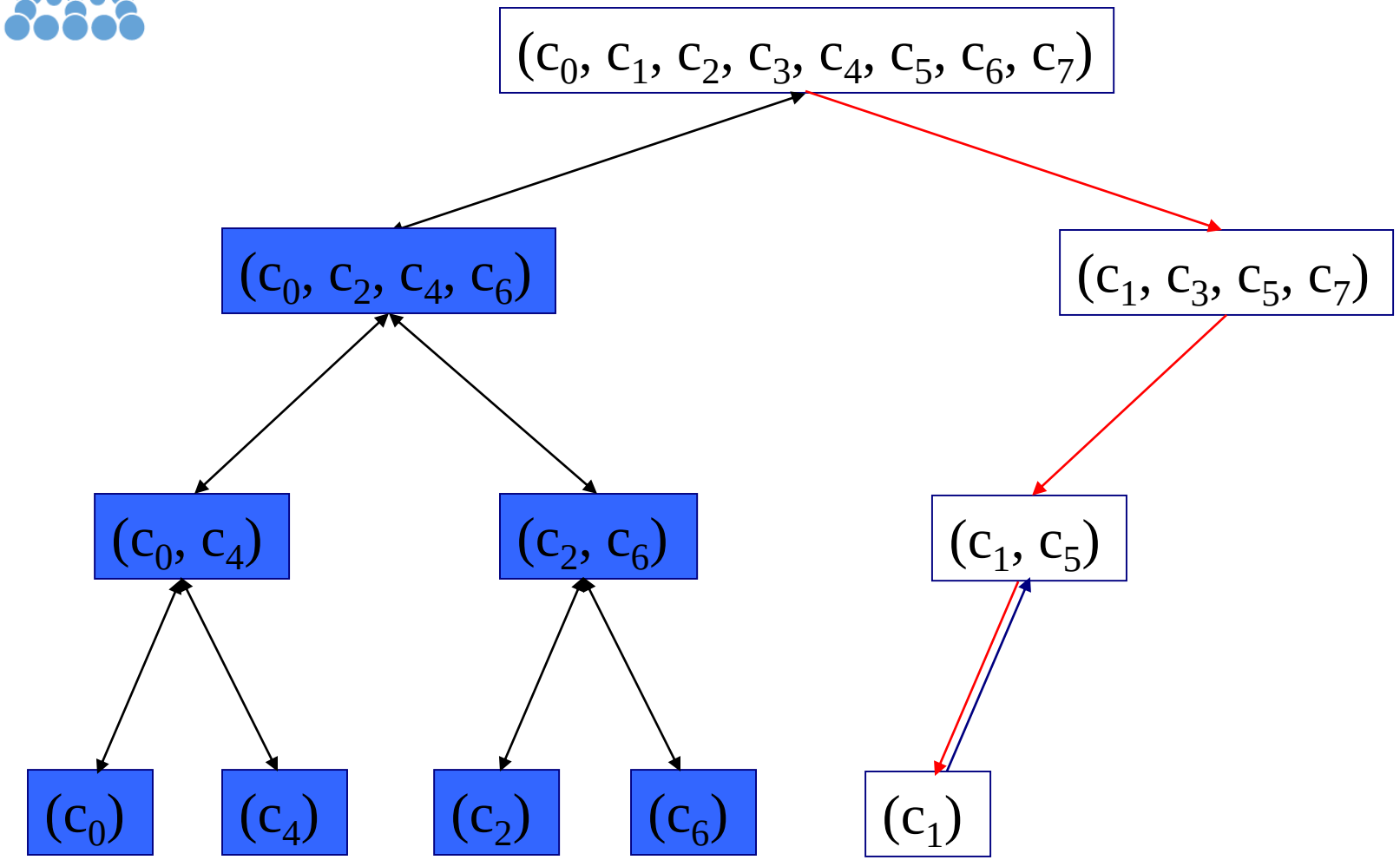

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_0)
(c_4)

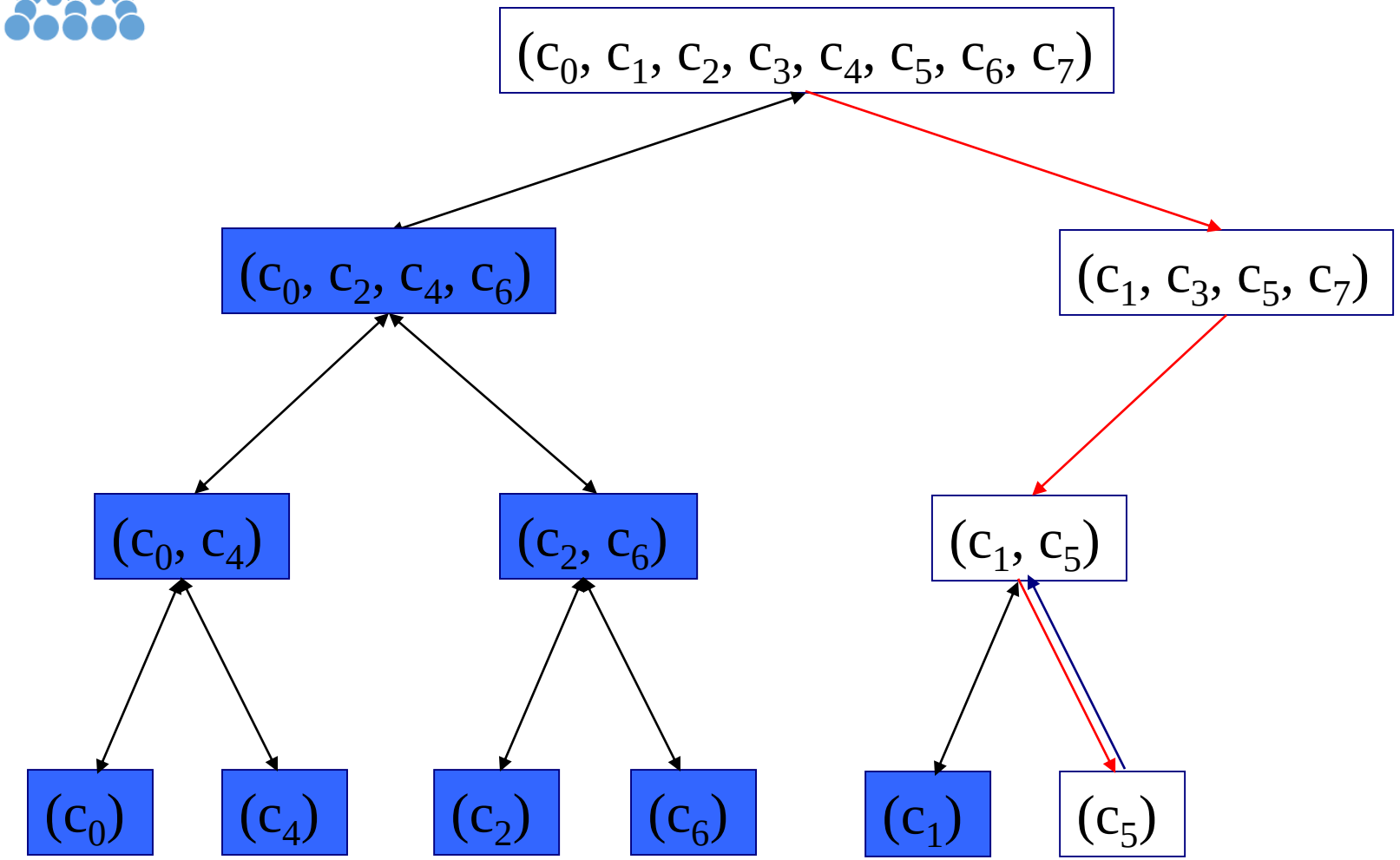

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_2, c_6)
(c_0)
(c_4)
(c_2)

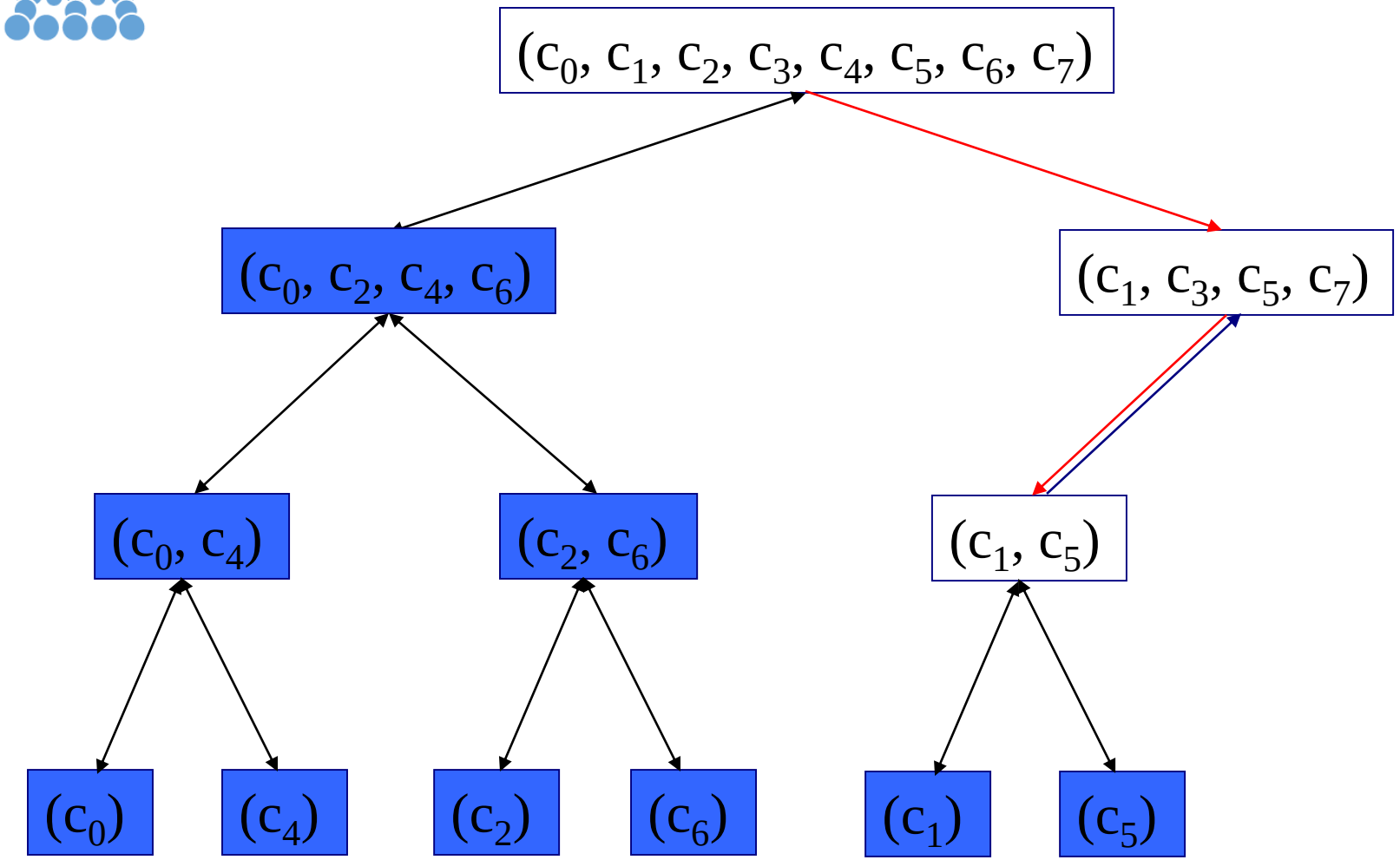

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_2, c_6)
(c_0)
(c_4)
(c_2)
(c_6)

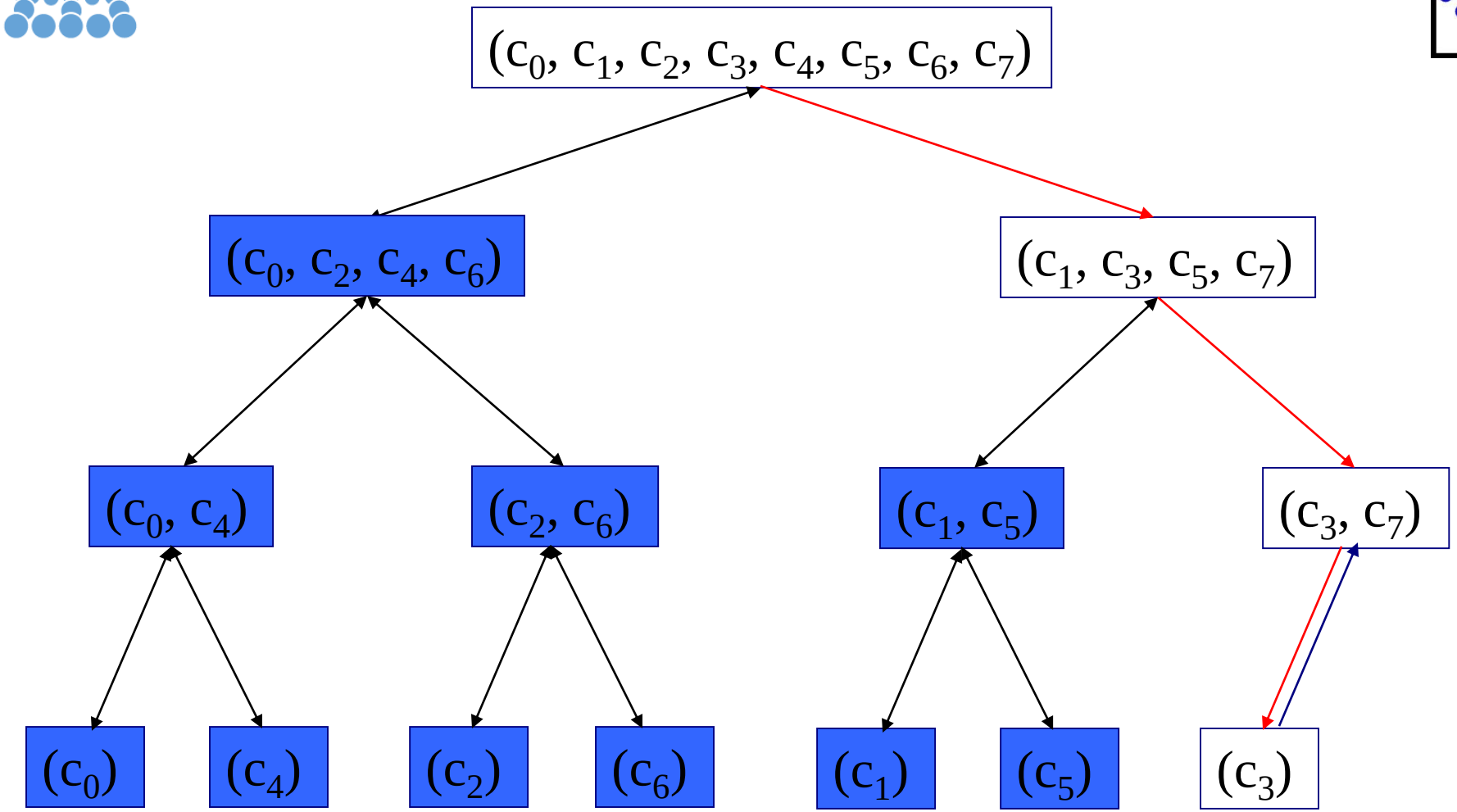

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_2, c_6)
(c_0)
(c_4)
(c_2)
(c_6)

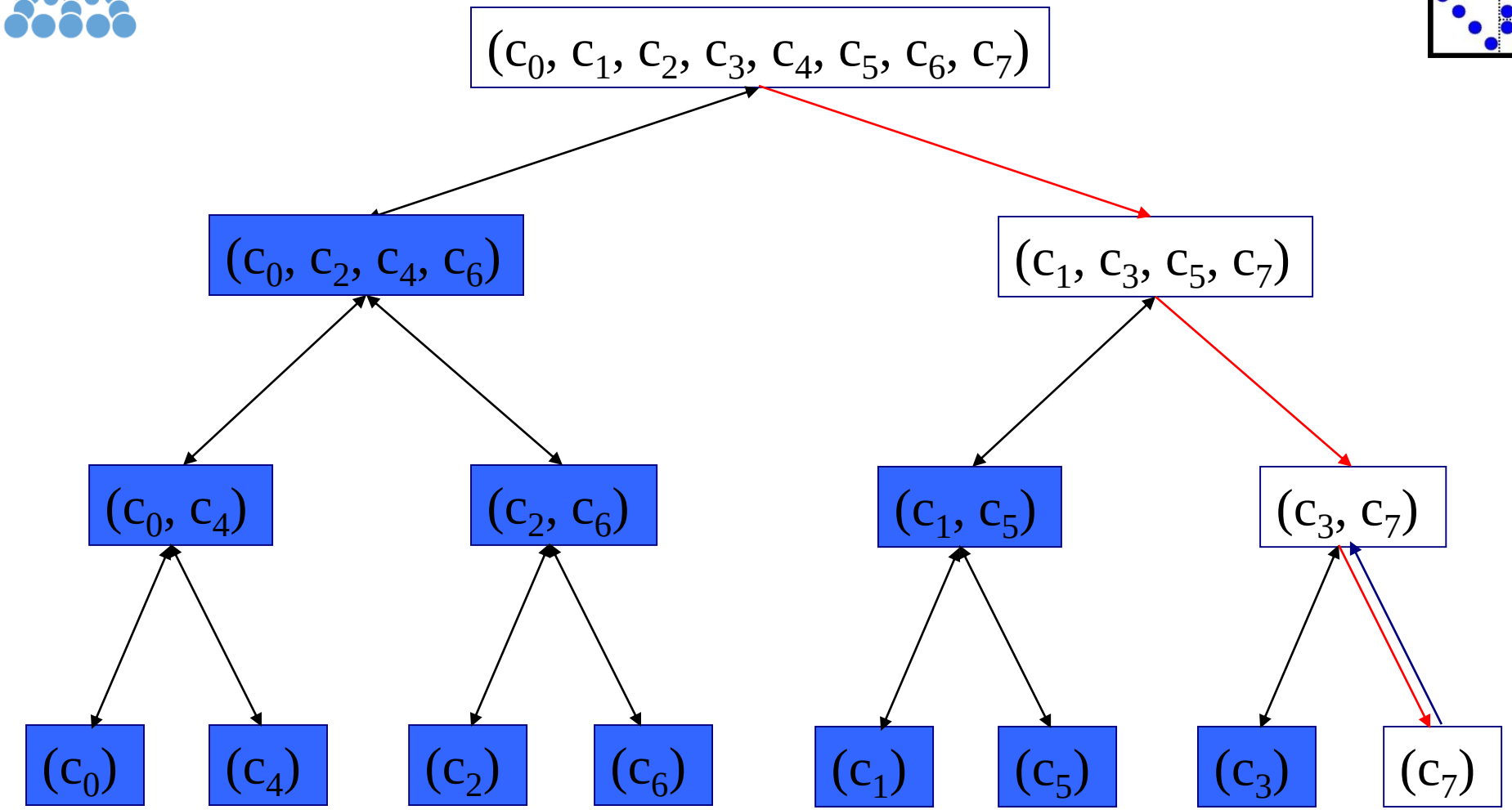

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_0, c_4)
(c_2, c_6)
(c_0)
(c_4)
(c_2)
(c_6)

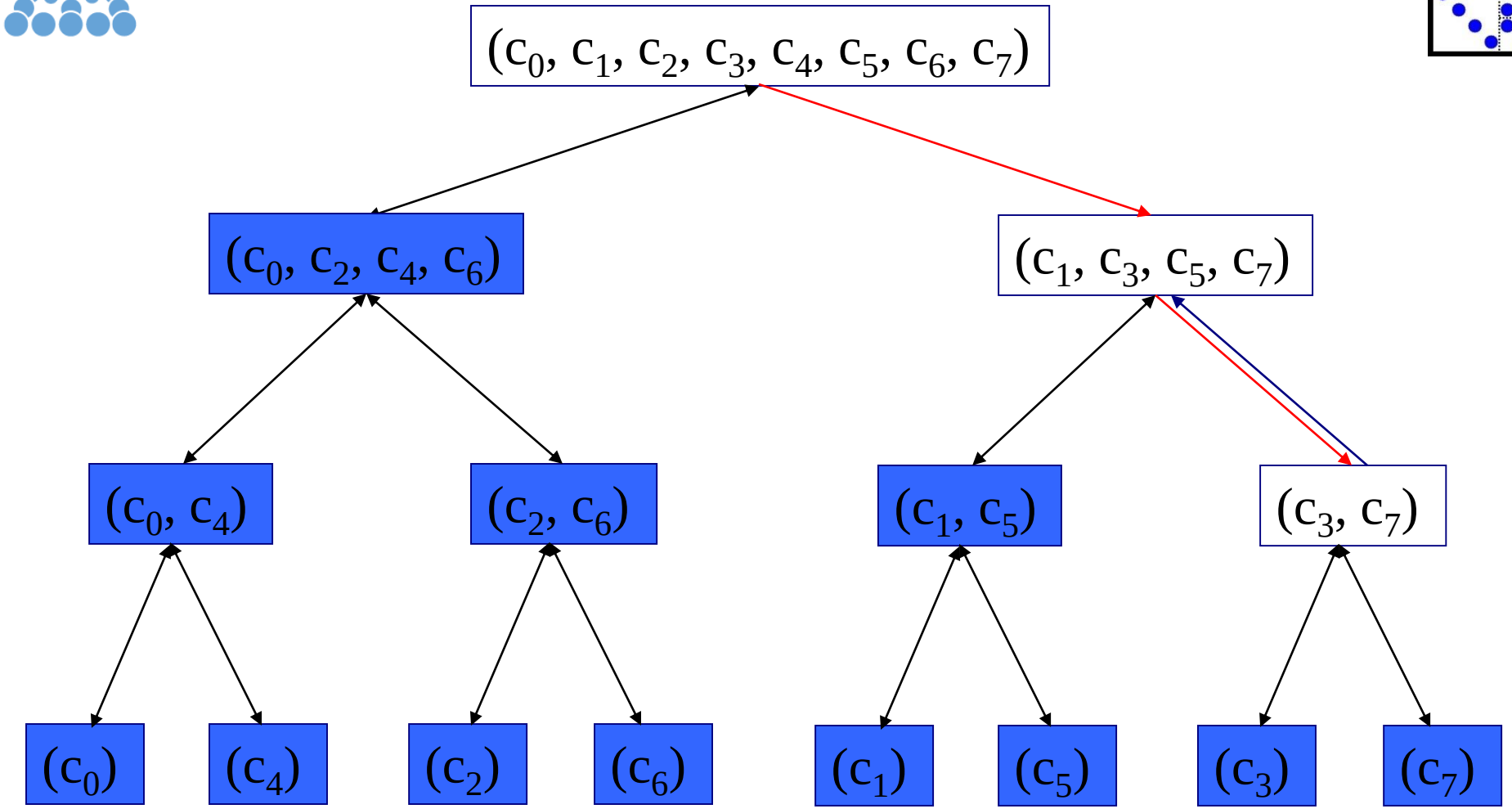

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)

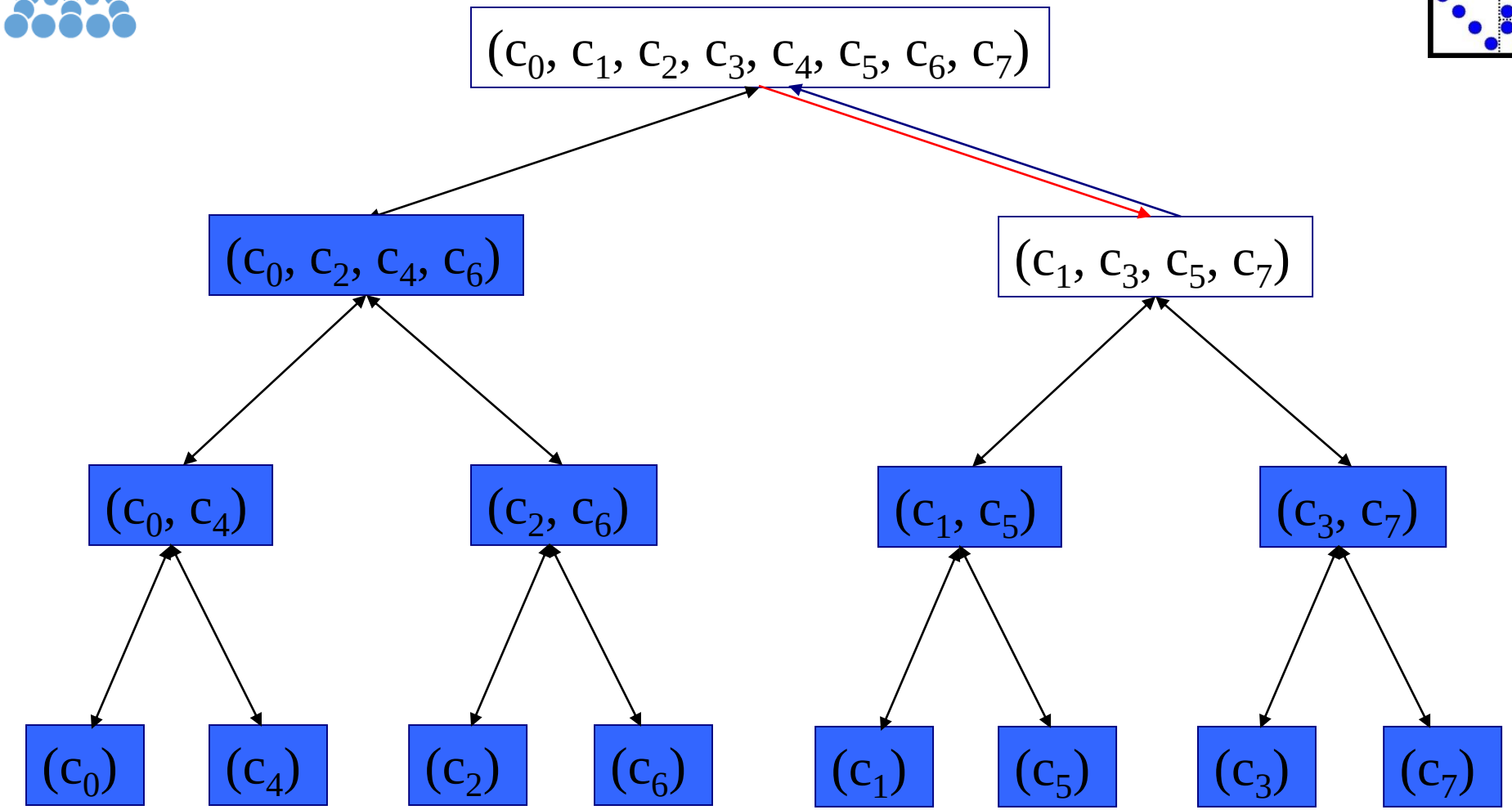

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)

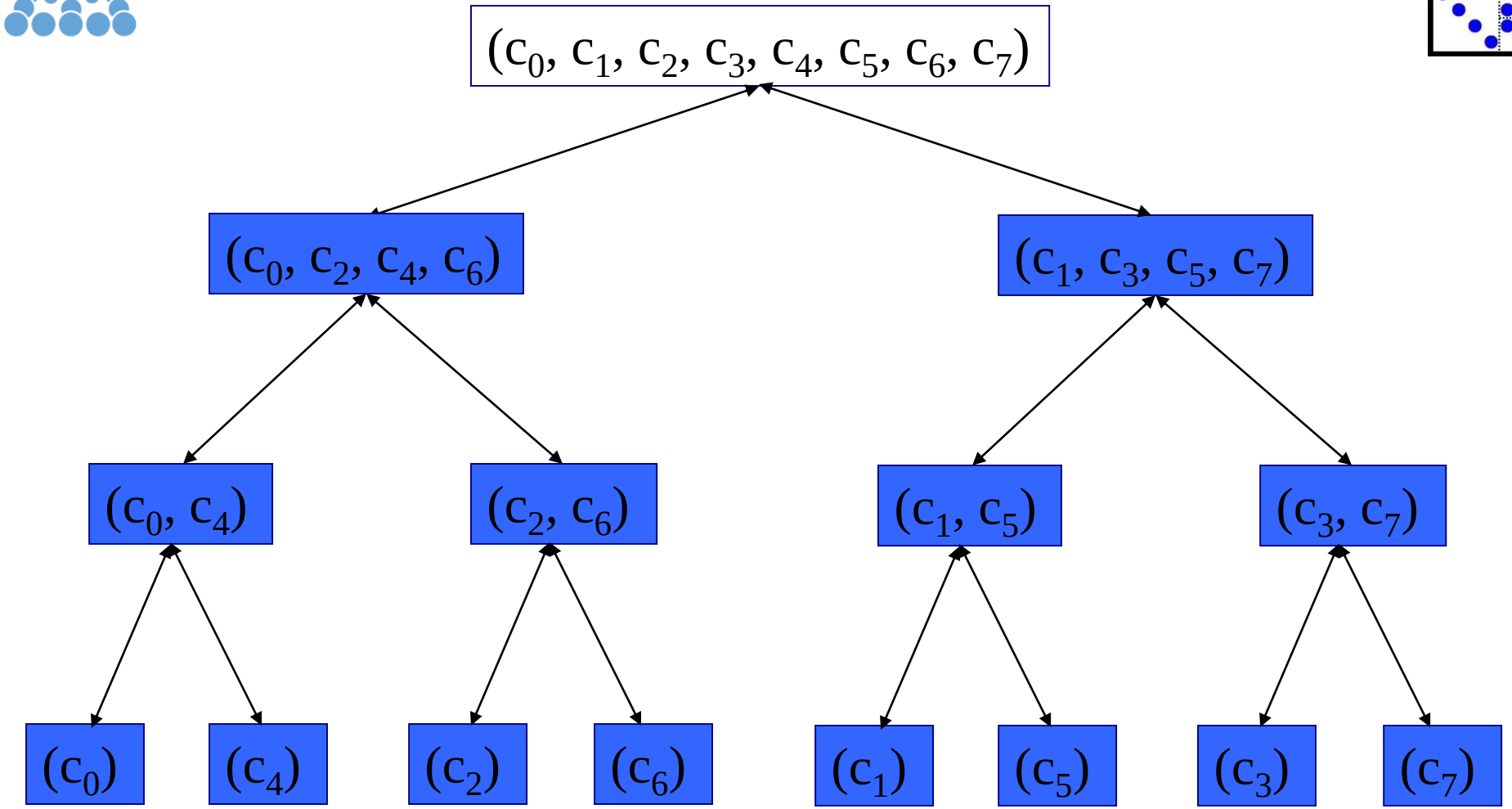

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)

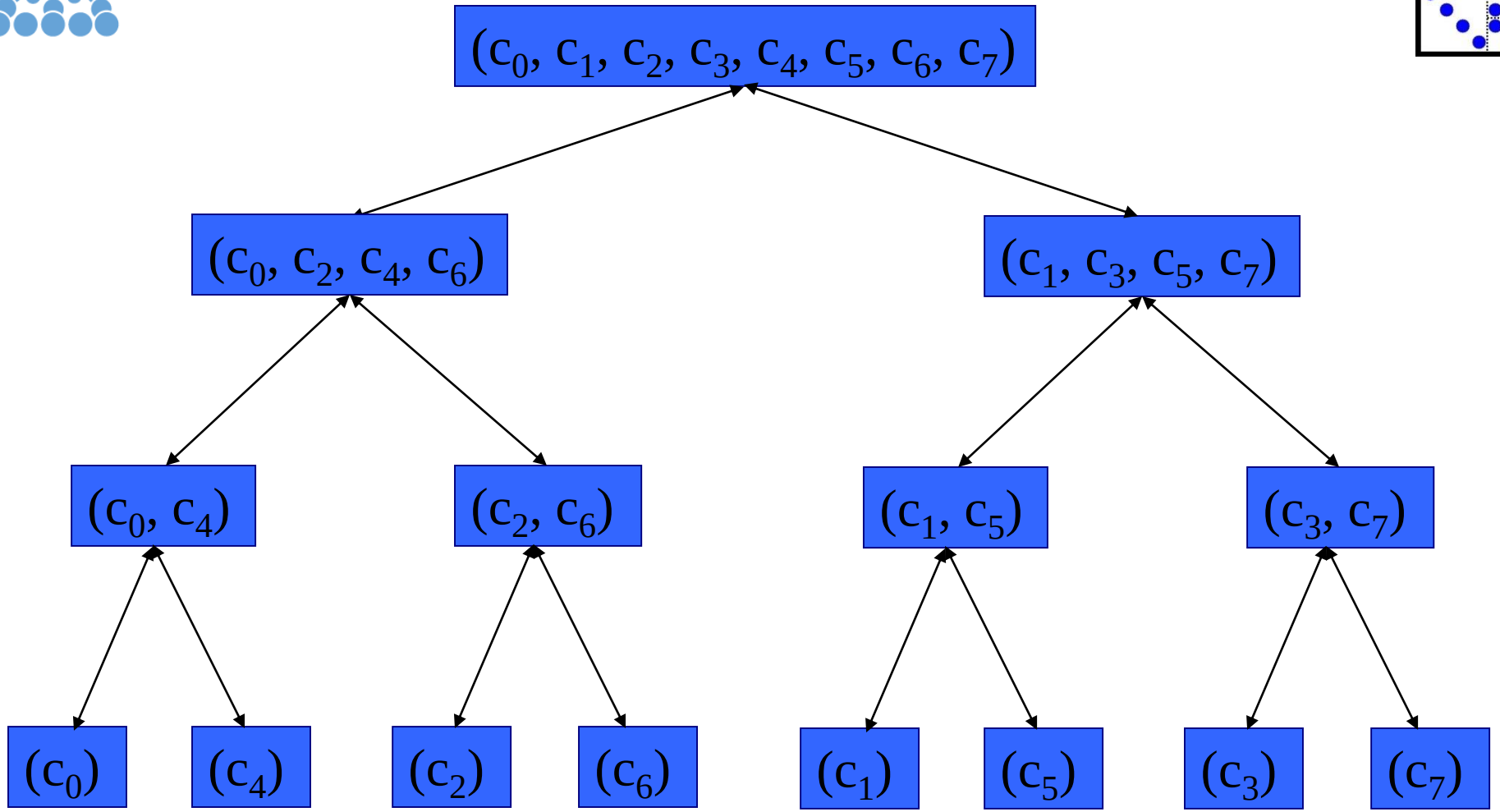

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)

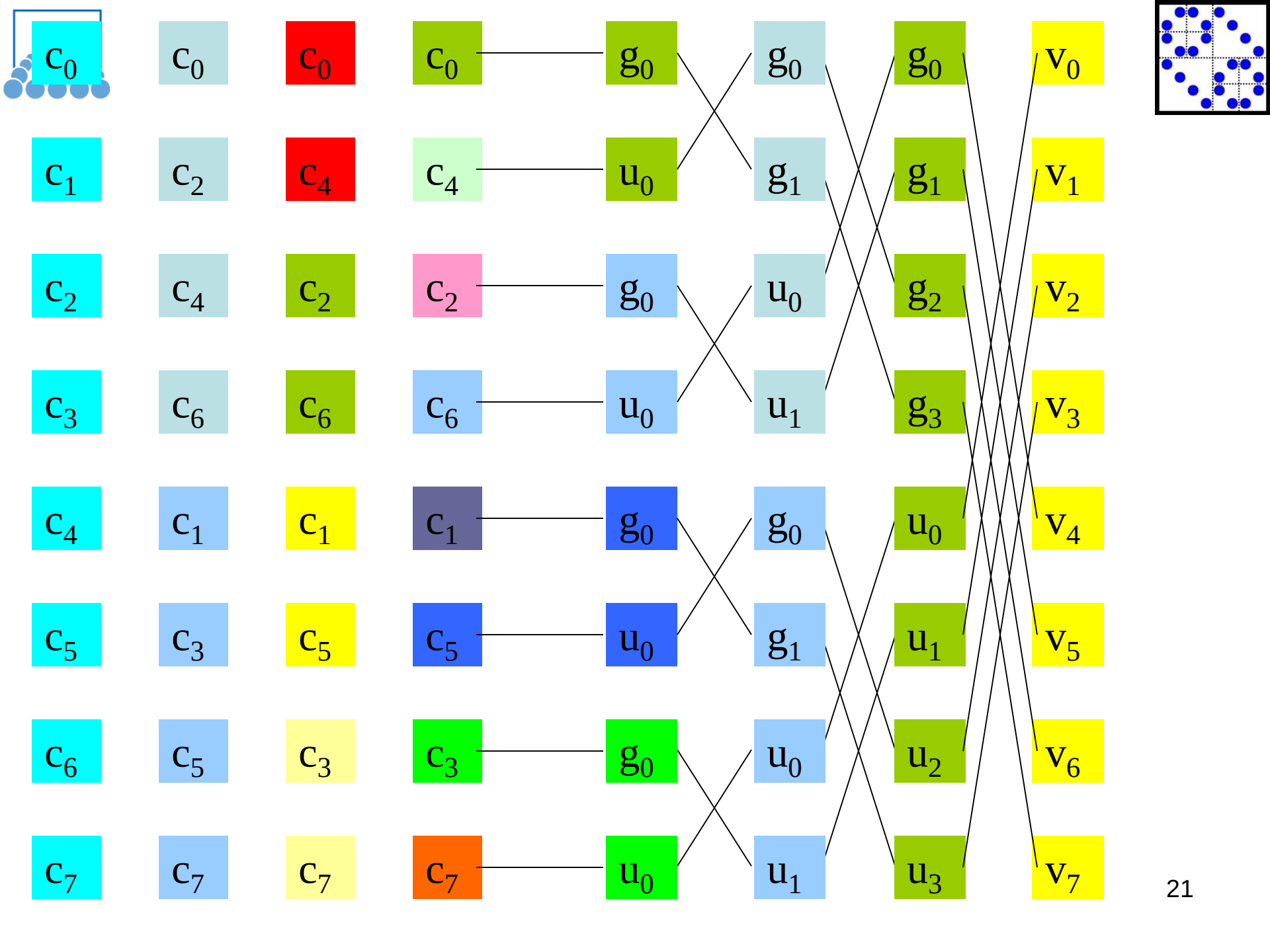

$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)
(c_7)


$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)
(c_7)


$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)
(c_7)


$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)
(c_7)


$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$
(c_0, c_2, c_4, c_6)
(c_1, c_3, c_5, c_7)
(c_0, c_4)
(c_2, c_6)
(c_1, c_5)
(c_3, c_7)
(c_0)
(c_4)
(c_2)
(c_6)
(c_1)
(c_5)
(c_3)
(c_7)


c0 c0 c0 c0 g0 g0 g0 v0
c1 c2 c4 c4 u0 g1 g1 v1
c2 c4 c2 c2 g0 u0 g2 v2
c3 c6 c6 c6 u0 u1 g3 v3
c4 c1 c1 c1 g0 g0 u0 v4
c5 c3 c5 c5 u0 g1 u1 v5
c6 c5 c3 c3 g0 u0 u2 v6
c7 c7 c7 c7 u0 u1 u3 v7

FFT sequentially

The recursive formulation of the FFT can be written by log(n) simple loops.

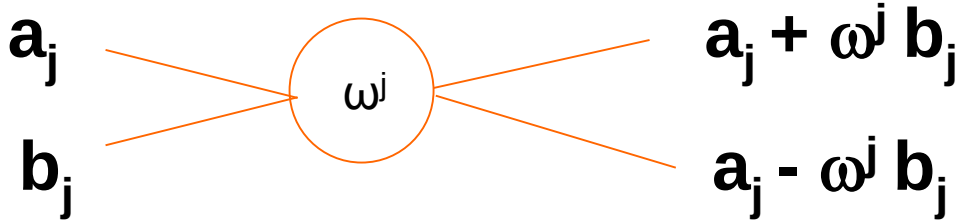
Thereby, the first step is the reordering of the variables → Bitreversal

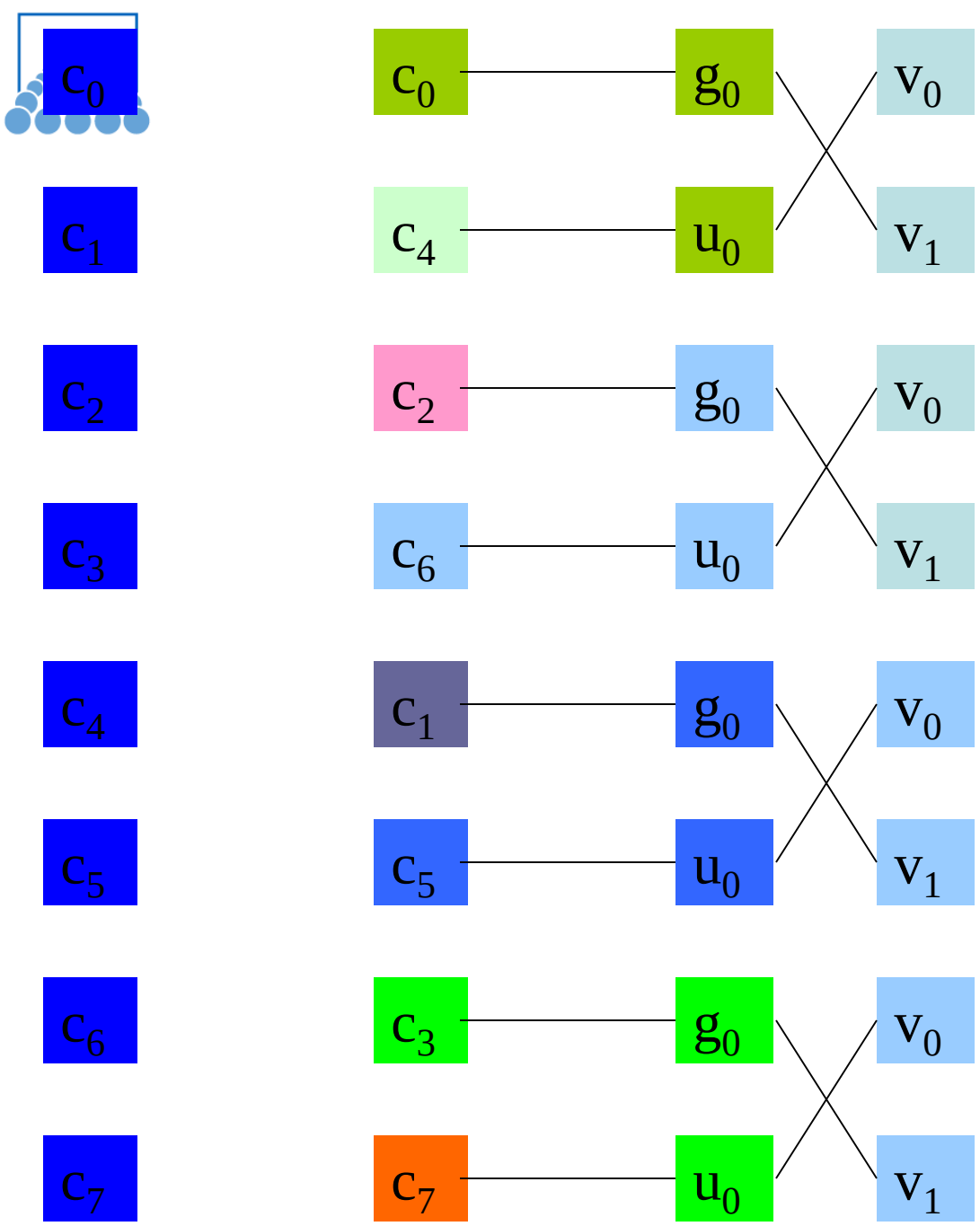
Index $k = (k_p, \dots, k_1)_2 \rightarrow (k_1, \dots, k_p)_2$,

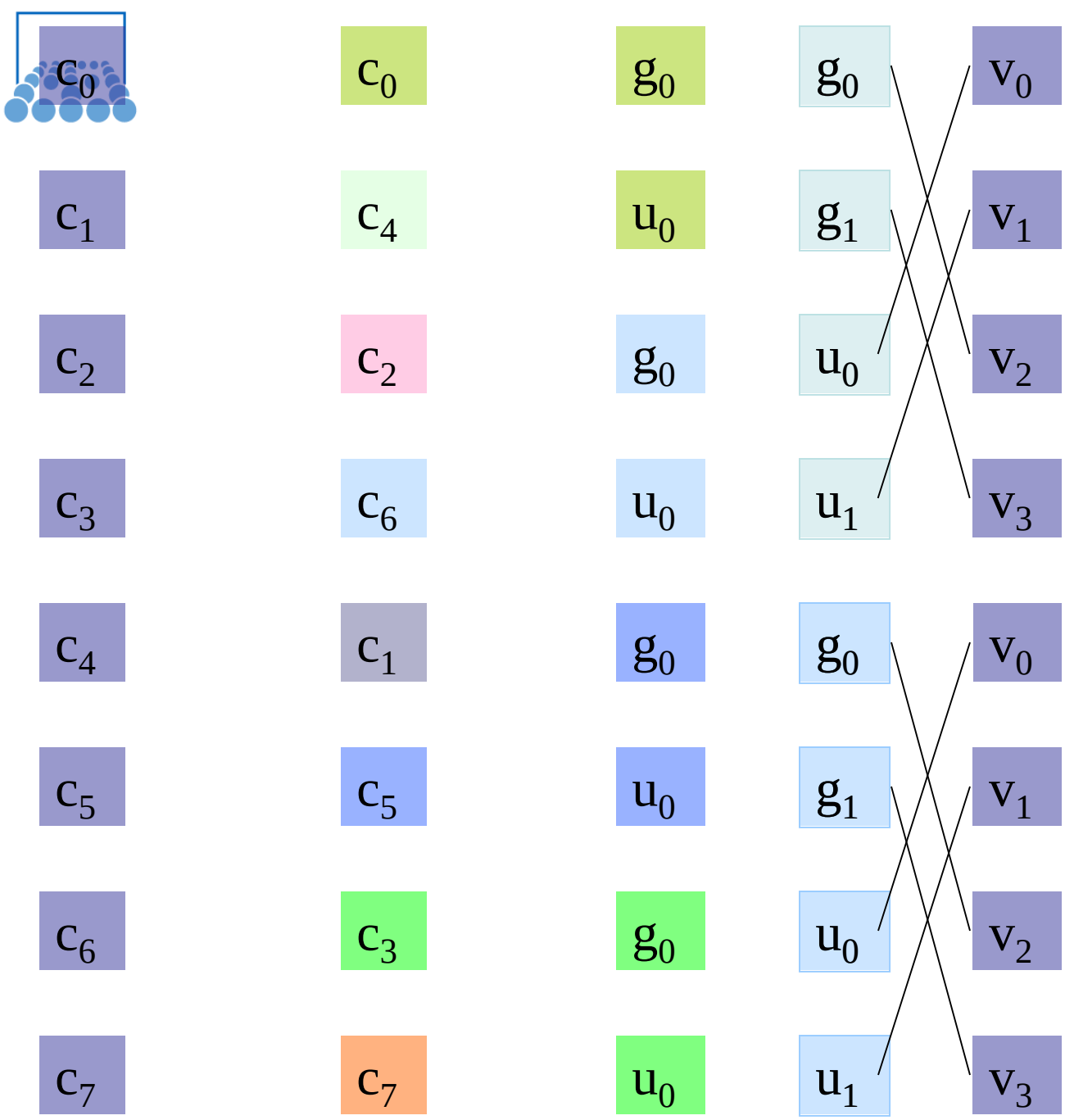
e.g. $5 = (0\ 0\ 1\ 0\ 1)_2 \rightarrow (1\ 0\ 1\ 0\ 0)_2 = 16 + 4 = 20$, $c_5 \rightarrow c_{20}$

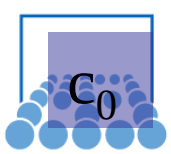
After permutation, the butterflies have to be applied between elements of certain distance.

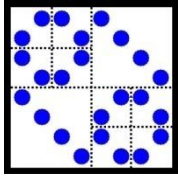
Butterfly:



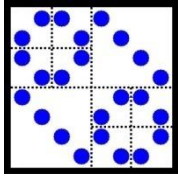

c0
c1
c2
c3
c4
c5
c6
c7
c0
g0
v0
c4
u0
v1
c2
g0
v0
c6
u0
v1
c1
g0
v0
c5
u0
v1
c3
g0
v0
c7
u0
v1


c_0
c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_0
c_4
c_2
c_6
c_1
c_5
c_3
c_7
g_0
u_0
g_0
u_0
g_0
u_0
g_0
u_0
g_0
g_1
u_0
u_1
g_0
g_1
u_0
u_1
v_0
v_1
v_2
v_3
v_0
v_1
v_2
v_3





c_0	c_0	g_0	g_0	g_0	v_0
c_1	c_4	u_0	g_1	g_1	v_1
c_2	c_2	g_0	u_0	g_2	v_2
c_3	c_6	u_0	u_1	g_3	v_3
c_4	c_1	g_0	g_0	u_0	v_4
c_5	c_5	u_0	g_1	u_1	v_5
c_6	c_3	g_0	u_0	u_2	v_6
c_7	c_7	u_0	u_1	u_3	v_7



FFT in Parallel

Costs in parallel: n processors $k = 0, 1, 2, \ldots, 2^p$

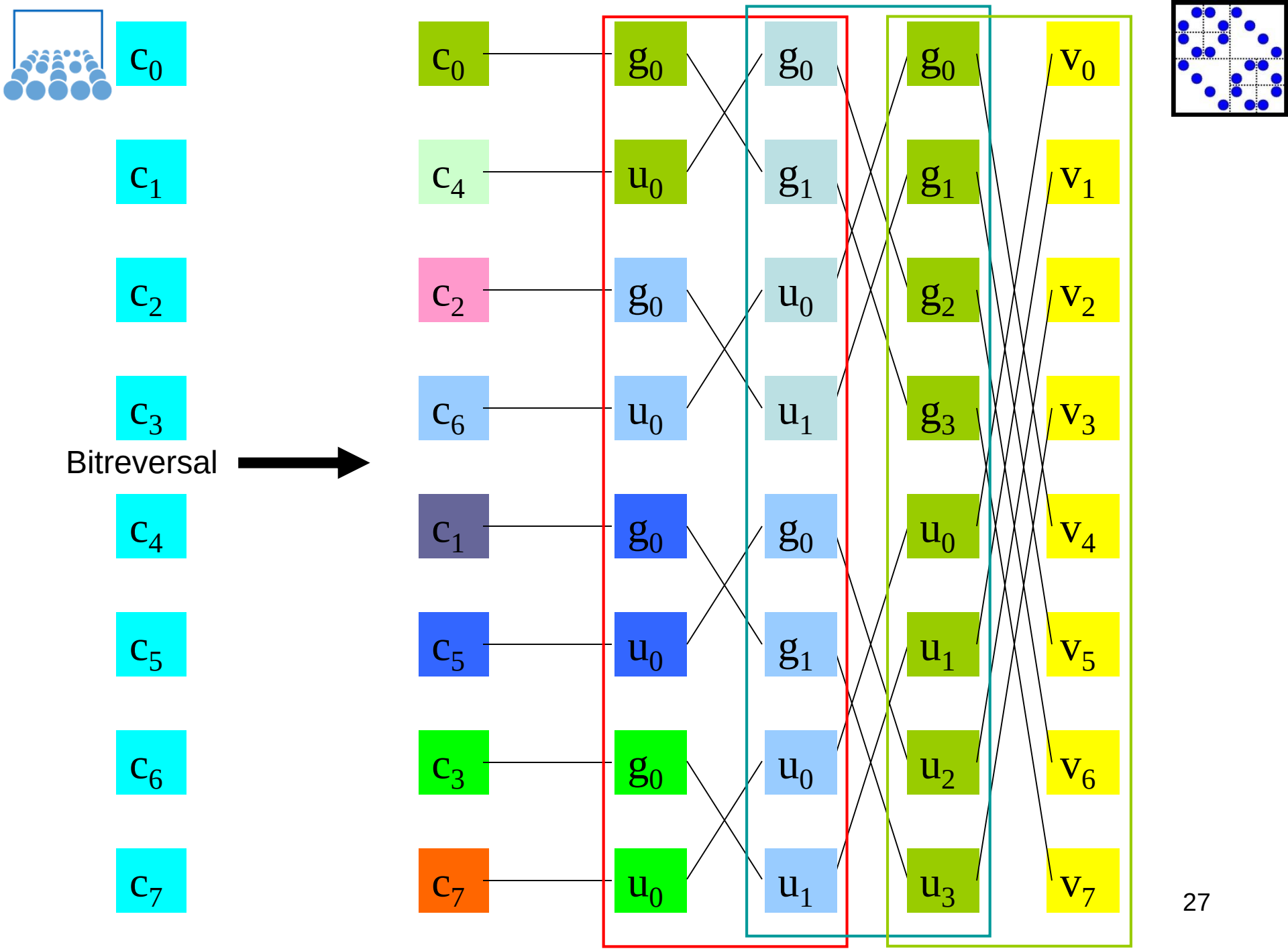
each processor computes Bitreversal k and sends its entry to the resulting processor $\pi(k)$

Then each two neighbouring processors compute butterfly,

log(n) times with growing distance

Only n processors, but still log(n) time steps.

Advantage over trivial parallelization only in number of processors


c0
c1
c2
c3
Bitreversal
c4
c5
c6
c7
c0
c4
c2
c6
c1
c5
c3
c7
g0
u0
g0
u0
g0
u0
g0
u0
g0
g1
u0
u1
g0
g1
u0
u1
g0
g1
g2
g3
u0
u1
u2
u3
v0
v1
v2
v3
v4
v5
v6
v7

FFT on Hypercube

Distribute entries on vertices of hypercube.

Butterfly has to be applied always between neighbors in distance 1,2,4,…

Hence, the binary indices differ only at one position.

Therefore, butterflies have to computed only between neighboring vertices