Rayleigh Quotient Iteration

Rayleigh Quotient iteration: Start with vector y and real $\rho = y^T A y / y^T y$ and repeat:

$$v = y / \|y\|; \qquad y = (A - \rho I)^{-1} v; \qquad \rho = \rho + y^T v / \|y\|^2;$$

Inverse Iteration with replacing the shift σ by the newest eigenvalue estimate.

y: new eigenvector estimate → leads to new eigenvalue estimate:

$$\rho_{new} = \frac{y^T A y}{y^T y} = \frac{y^T (A - \rho I + \rho I) y}{y^T y} = \frac{y^T (A - \rho I) y}{y^T y} + \rho = \frac{y^T v}{\|y\|^2} + \rho$$

Fast convergence, but uncertain to which eigenvalue we will converge.
Expensive! Ill-conditioned!

8.5 Arnoldi (Lanczos) for sparse A

Use the transformation on Hessenberg (tridiagonal) form described for GMRES.
Compute the eigenvalues of the small Hessenberg matrix and use them as approximations for the eigenvalues of the original matrix.

By Arnoldi Orthogonalization of the Krylov subspace (b,Ab,A^2b,…) we get the relation

$$Au_{j-1} = \sum_{k=1}^{j-1} h_{k,j-1} u_k + \tilde{u}_j = \sum_{k=1}^{j} h_{k,j-1} u_k$$

$$A \cdot U_m = A\begin{pmatrix} u_1 & \dots & u_m \end{pmatrix} = \begin{pmatrix} u_1 & \dots & u_{m+1} \end{pmatrix} \cdot \tilde{H}_{m+1,m} = U_m H_{m,m} + h_{m+1,m} u_{m+1}$$

Eigenvalues of $H_{m,m}$ as approximations for A.
(Small $h_{m+1,m}$ → good approximation).
Good approximation for extreme eigenvalues.
For symmetric A, H is tridiagonal.

The same approach can be applied on: f(A)b

8.6 Jacobi-Davidson for sparse A

Idea: - No Krylov subspace, more related to Rayleigh quotient and subspace it.
- choose subspace relative to eigenvalue we are looking for
- include preconditioning;

Starting point:
Consider eigenvalue approximations derived by $V_m^H A V_m$
for subspace relative to V_m.
The eigenpairs of $V_m^H A V_m$ are used as approximations to some eigenvalues of A

How to choose new subspace V_{m+1} with additional vector u such that the new approximation for special eigenvalues is strongly improved?

For first eigenpair approximation u_m and $t_m=(u_m^H A u_m)/(u_m^H u_m)$, we try to improve these approximations by small corrections u and t to get better estimates u_m+u and t_m+t

$$A(u_m+u)\overset{!}{=}(t_m+t)(u_m+u), \qquad u \perp u_m,\; u_m^H u_m = 1$$

Jacobi-Davidson II

$$A(u_m + u) = (t_m + t)(u_m + u), \qquad u \perp u_m$$

$$\Downarrow$$

$$(A - t_m I)u = t u_m - (A - t_m I)u_m + \cancel{tu}$$

$\downarrow$

ignore correction t u of second order

Use orthogonal projection with $I - u_m u_m^H$ from the left.
This leads to

$$(I - u_m u_m^H)(A - t_m I)(I - u_m u_m^H)\, u = -(I - u_m u_m^H)(A - t_m I)u_m$$

$$(I - u_m u_m^H)(A - t_m I)(I - u_m u_m^H)\, u = -(A - t_m I)u_m + \cancel{u_m(u_m^H A u_m) - t_m u_m (u_m^H u_m)}$$

$= 0$

Jacobi-Davidson III

For new approximation we have to solve

$$(I - u_m u_m^H)(A - t_m I)(I - u_m u_m^H)\, u = -(A - t_m I) u_m$$

$$P(A - t_m I) P u = -r_m \qquad or \qquad \boxed{\tilde{A} u = -r_m}$$

$A - t_m I$ gets ill-conditioned for t_m near eigenvalue, but P is a projection orthogonal to the near singular vector!

$\tilde{A}$ is singular, but linear system is still solvable.

Compared to Inverse Iteration/RQI:
Replace ill-conditioned by singular system.

Jacobi-Davidson IV

New eigenvector estimate $u_m+u \rightarrow u_{m+1}$ also leads to new eigenvalue estimate $t_{m+1} \rightarrow (u_{m+1}^H A u_{m+1})/(u_{m+1}^H u_{m+1})$.

Choose the new estimate u_{m+1} to enlarge the subspace V_m by the new vector u to V_{m+1}.

Compute eigenpairs of $V_{m+1}^H A V_{m+1}$.

Repeat this step a few times.

Restart the whole process with last best approximation as starting vector u_1, resp. 1-dim subspace V_1.

Advantages: Allows to compute also inner eigenvalues without solving more and more ill-conditioned problems like Rayleigh QI.

Jacobi-Davidson V

Main step: Solve linear system

$$P(A - t_m I)Pu = -r_m \qquad or \qquad P(A - t_m I)\tilde{u} = -r_m$$

approximately.

Therefore, we use a few steps of preconditioned cg or GMRES.

Preconditioner: M^{-1} preconditioner for A $\rightarrow$ $PM^{-1}P$ preconditioner for PAP

In each iteration step we have to multiply with A, with P, and solve in M.

Simple preconditioner: M = diag(A)
Better preconditioner: SPAI or MSPAI

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:

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

$$p_0(\lambda) = 1$$

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

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

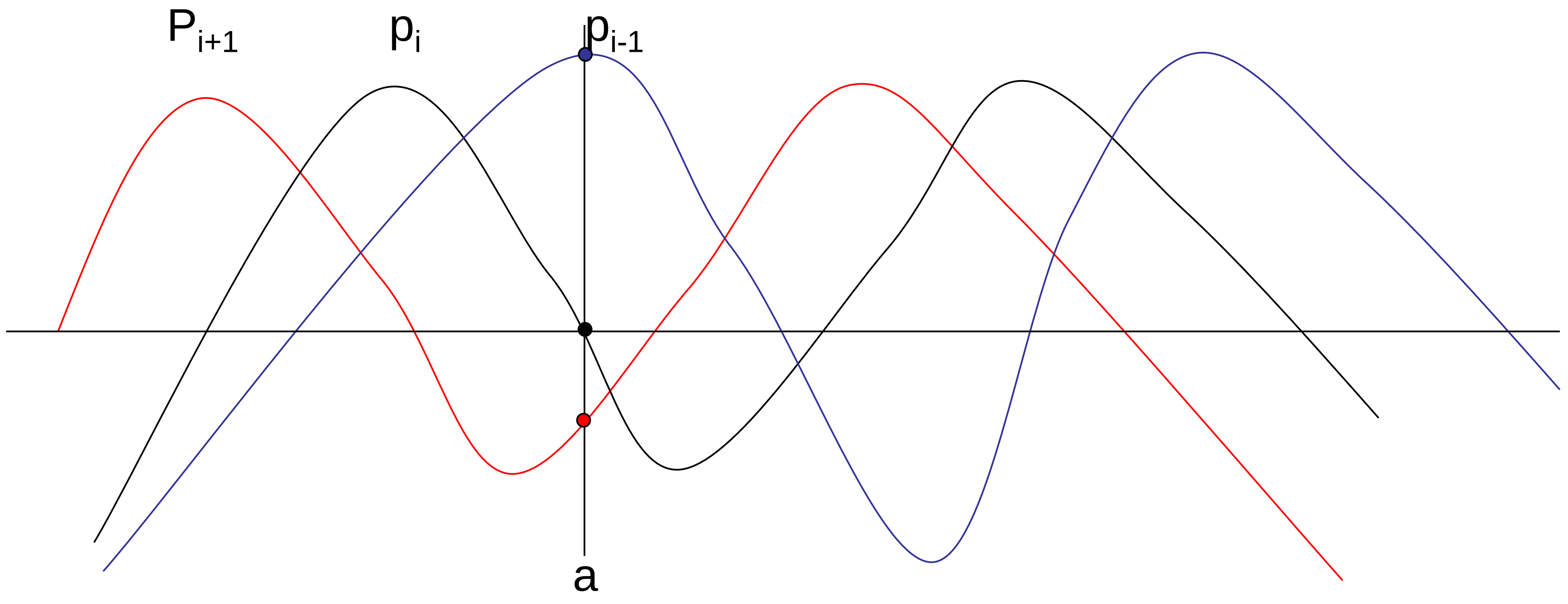
$$p_i(\lambda) = (\delta_i - \lambda)p_{i-1}(\lambda) - \gamma_i^2 p_{i-2}(\lambda), \qquad i = 3,4,\dots,n$$

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

The sequence of polynomials is a Sturm chain:

1. All p_i have only single zeros
2. sign($p_{n-1}(a)$) = - 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 $p_i(a)$, $i=1,\ldots,n$.

$\boxed{\text{It holds: } w(a) = \text{\# zeros of } p_n(x) \text{ for } x<a.}$

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: $\lambda_i < a \rightarrow w(a) = [\text{ \# zeros left of } a\ ] >= i$

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
- can be used with high or low accuracy

8.8 MR^3 for tridiagonal matrices

Idea: Use inverse iteration for computing the eigenvectors of a tridiagonal matrix.
In prestep the eigenvalues have to be computed!

Observations:
Inverse iteration is cheap, because of tridiagonal form

Parallel and independent Inverse Iteration for different eigenvalues.
High accuracy inspite of (near) singular linear system!

Find a good starting vector such that we need only small number of iterations!

Multiple Relatively Robust Representations = MRRR

Outline of the algorithm:

Compute eigenvalue approximation λ with high relative accuracy (e.g. Bisection)

Find the column number r of $(T - \lambda I)^{-1}$ with largest norm
Use bidiagonal factorizations $T = LDL^T$.

Perform one step of inverse iteration $(T - \lambda I)\, z = e_r$

MR^3 allows the computation of eigenvectors with high accuracy (also for small or close together eigenvalues) using factorizations: $L_+D_+L_+^T = LDL^T - \sigma I$.

Find optimal column k:

Consider $$\left(LDL^T - \lambda I\right)v_k = e_k \gamma_k \qquad with \quad v_k(k) = 1$$

Then $$\frac{1}{\gamma_k} = \frac{e_k^T v_k}{\gamma_k} = e_k^T \left(\frac{v_k}{\gamma_k}\right) = e_k^T \left(\left(LDL^T - \lambda I\right)^{-1} e_k\right) = \left(\left(LDL^T - \lambda I\right)^{-1}\right)_{k,k}$$

There exist O(n) algorithms for computing all γ_k based on factorizations $LDL^T - \lambda I = L_+ D_+ L_+^T = U_- D_- U_-^T$.

For finding the column r of largest norm choose column with minimum γ_k.

Problems: Fast convergence of Inverse Iteration
Separation of very close eigenvalues

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.

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

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 $A_{new} = RQ$

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

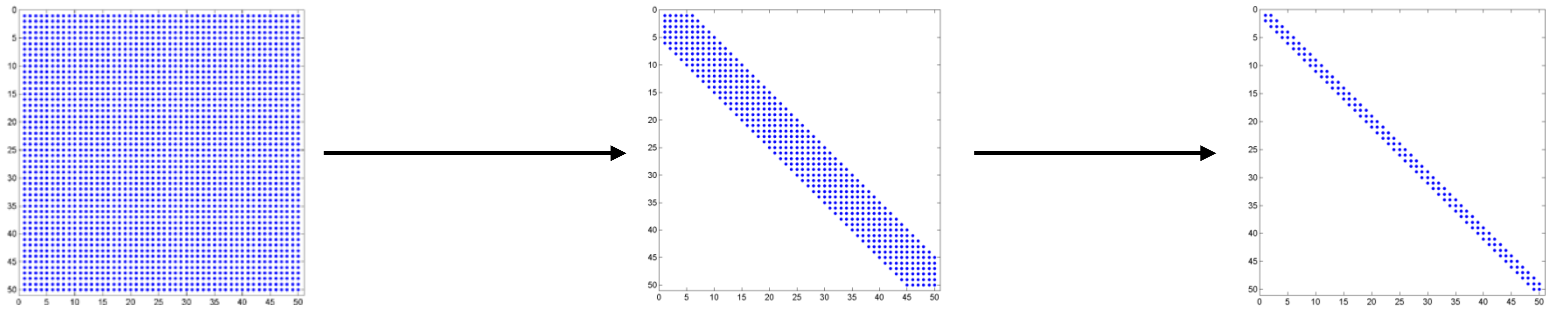
Therefore A and A_{new} have the same eigenvalues

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!

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.

Bothsided Householder for Tridiagonalization

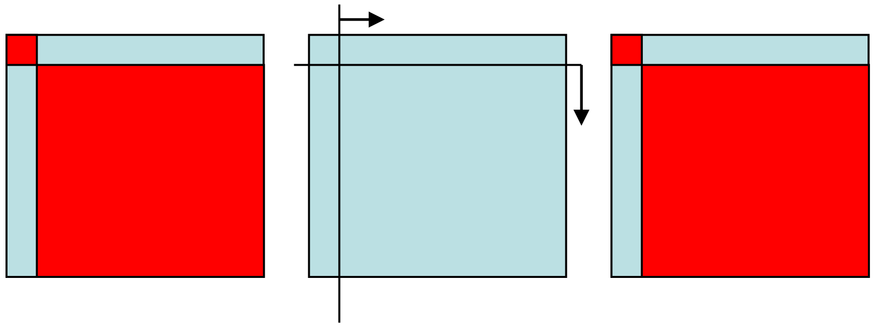
Compute Householder vector u in order to eliminate subtridiagonal entries in the first column/row.

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).

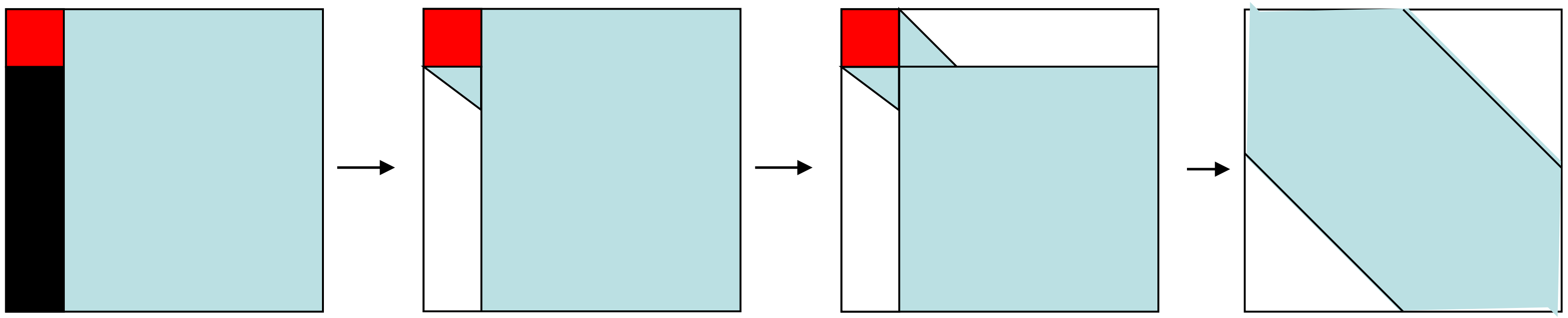
Block-Band reduction

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.

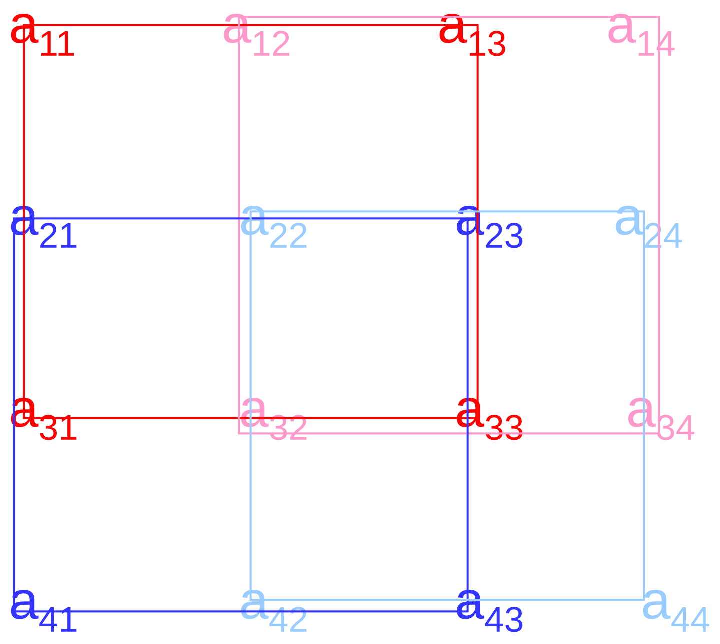


Store Householder vectors on positions of new generated zeros.

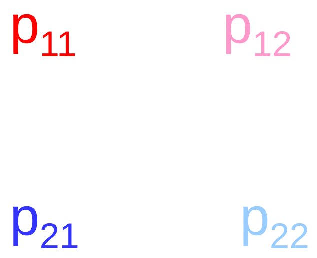
Use Cholesky QR.

2D-Cyclic Data Distribution

4 x 4 – Matrix on 2 x 2 processor array



a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}



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

7. Discrete Fourier Transform

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

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

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

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 fan-in processes.

DFT is very important in many applications.

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

→ FFT = Fast Fourier Transform

Odd – even Partitioning

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

$$= \sum_{k=0}^{n/2-1} c_{2k} \cdot \exp\left(\frac{2\pi ij2k}{n}\right) + \sum_{k=0}^{n/2-1} c_{2k+1} \cdot \exp\left(\frac{2\pi ij(2k+1)}{n}\right)$$

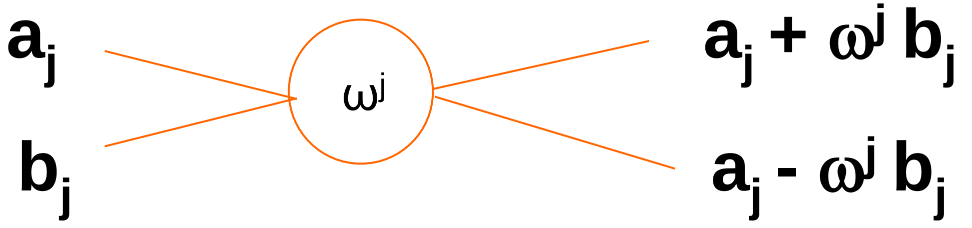
$$= \sum_{k=0}^{m-1} c_{2k} \boxed{\cdot \exp\left(\frac{2\pi ijk}{m}\right)} + \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \boxed{\cdot \exp\left(\frac{2\pi ijk}{m}\right)}$$

$$v_{m+j} =$$

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

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

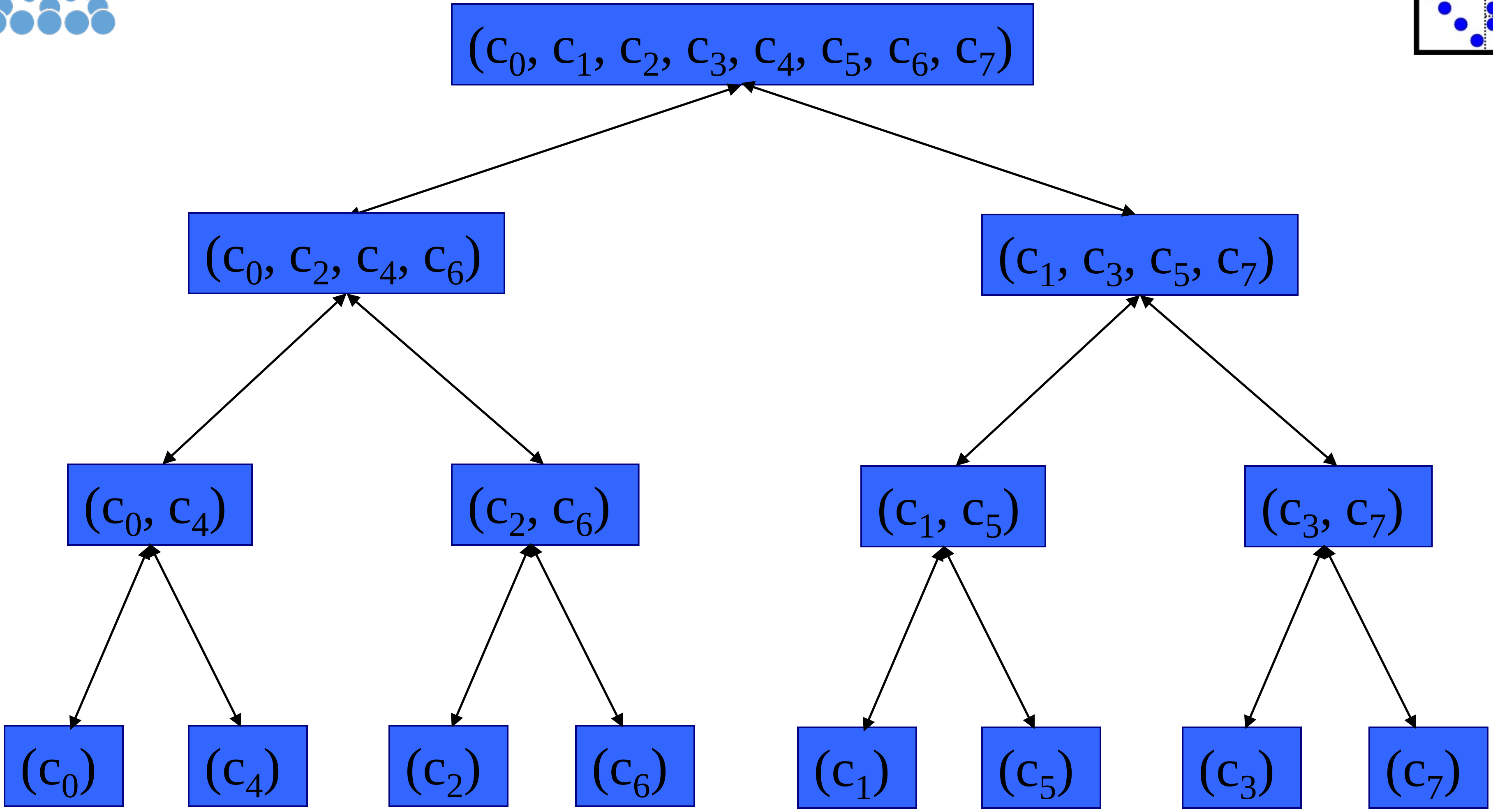
Butterfly:



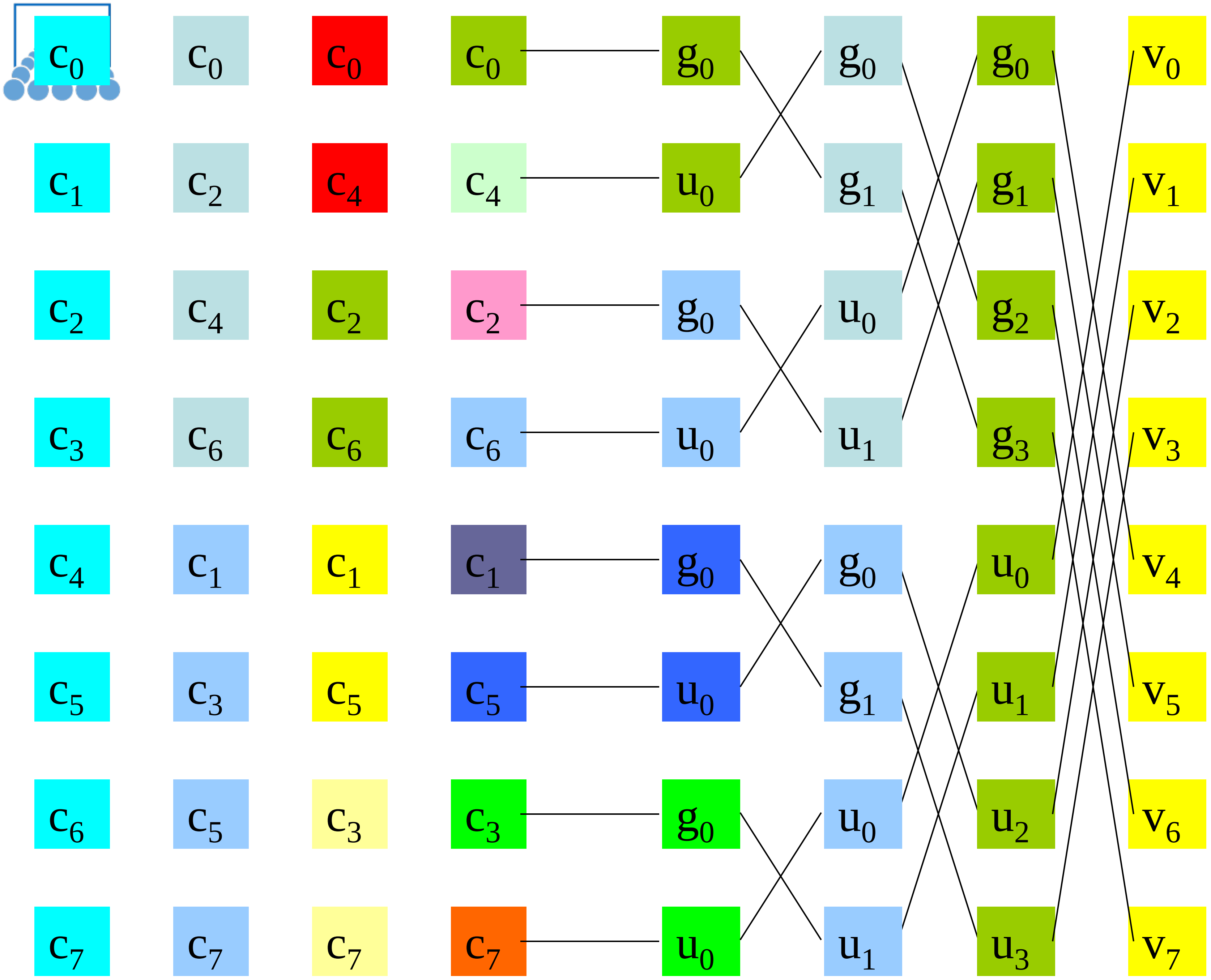
FFT: Recursive Algorithm

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

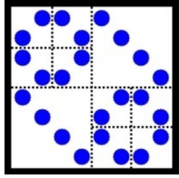
```
FUNCTION($v_0$,...,$v_{n-1}$) = IDFT($c_0$,…,$c_{n-1}$,n)
    IF n==1
        $v_0 = c_0$ ;
    ELSE
        m=n/2 ;
        ($g_0$,…, $g_{m-1}$) = IDFT($c_0$, $c_2$,…, $c_{n-2}$,m) ;
        ($u_0$,…, $u_{m-1}$) = IDFT($c_1$, $c_3$,…, $c_{n-1}$,m) ;
        $\omega = \exp(2i\pi/n)$ ;
        FOR 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_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_0 c_0 c_0 g_0 g_0 g_0 v_0
c_1 c_2 c_4 c_4 u_0 g_1 g_1 v_1
c_2 c_4 c_2 c_2 g_0 u_0 g_2 v_2
c_3 c_6 c_6 c_6 u_0 u_1 g_3 v_3
c_4 c_1 c_1 c_1 g_0 g_0 u_0 v_4
c_5 c_3 c_5 c_5 u_0 g_1 u_1 v_5
c_6 c_5 c_3 c_3 g_0 u_0 u_2 v_6
c_7 c_7 c_7 c_7 u_0 u_1 u_3 v_7



FFT sequentially

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

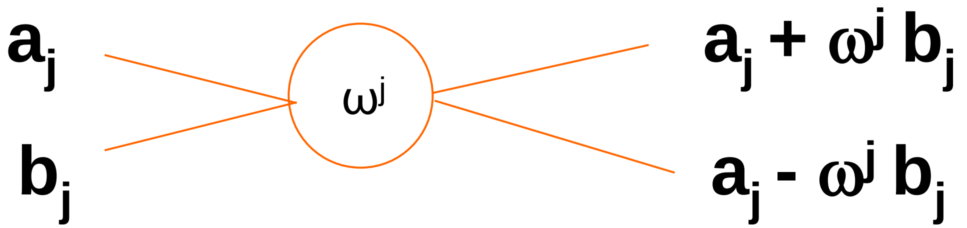
Thereby, the first step is the reordering of the variables → Bitreversal

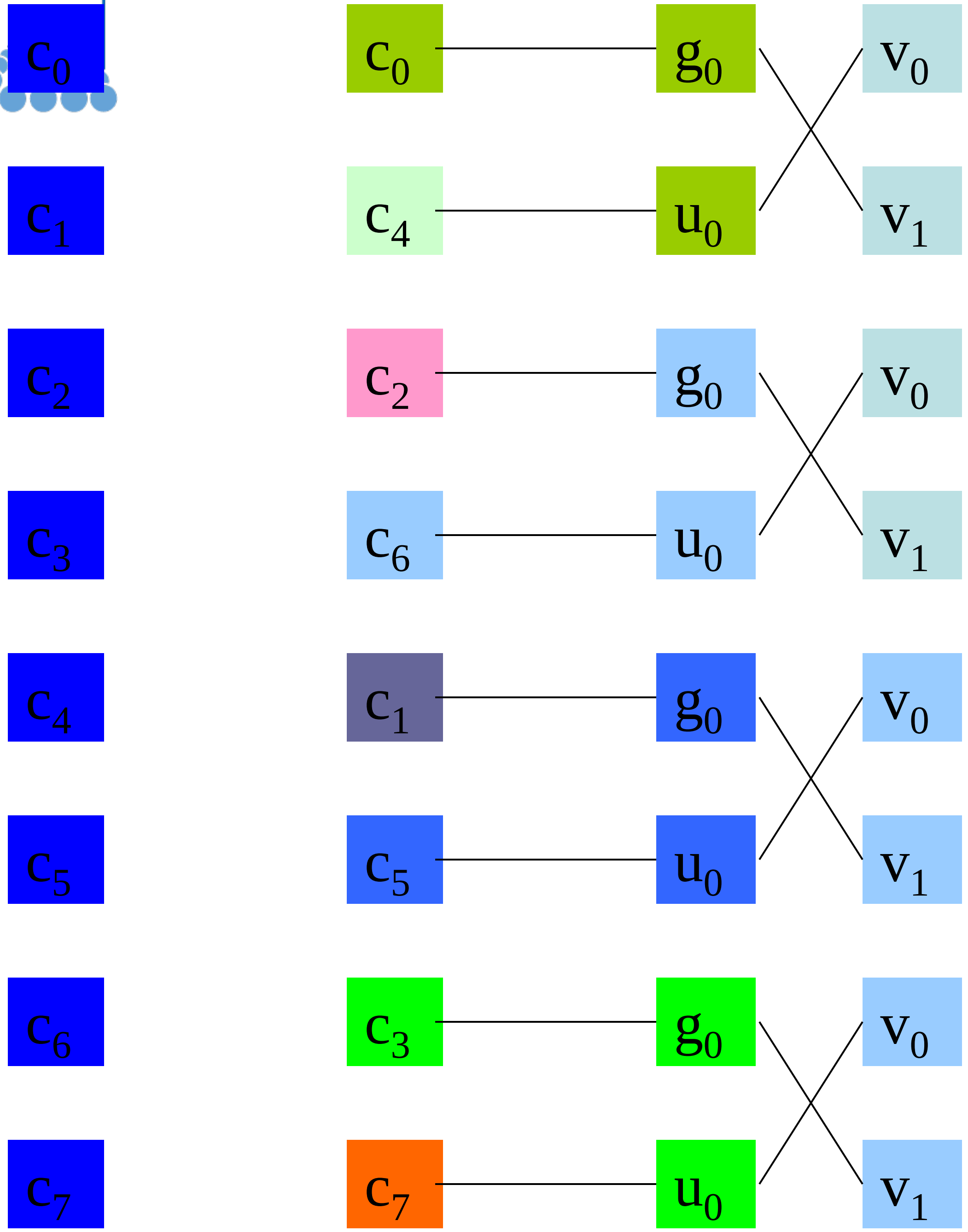
Index $k = (k_p, \dots , k_1)_2 \rightarrow (k_1, \dots , k_p)_2$,

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

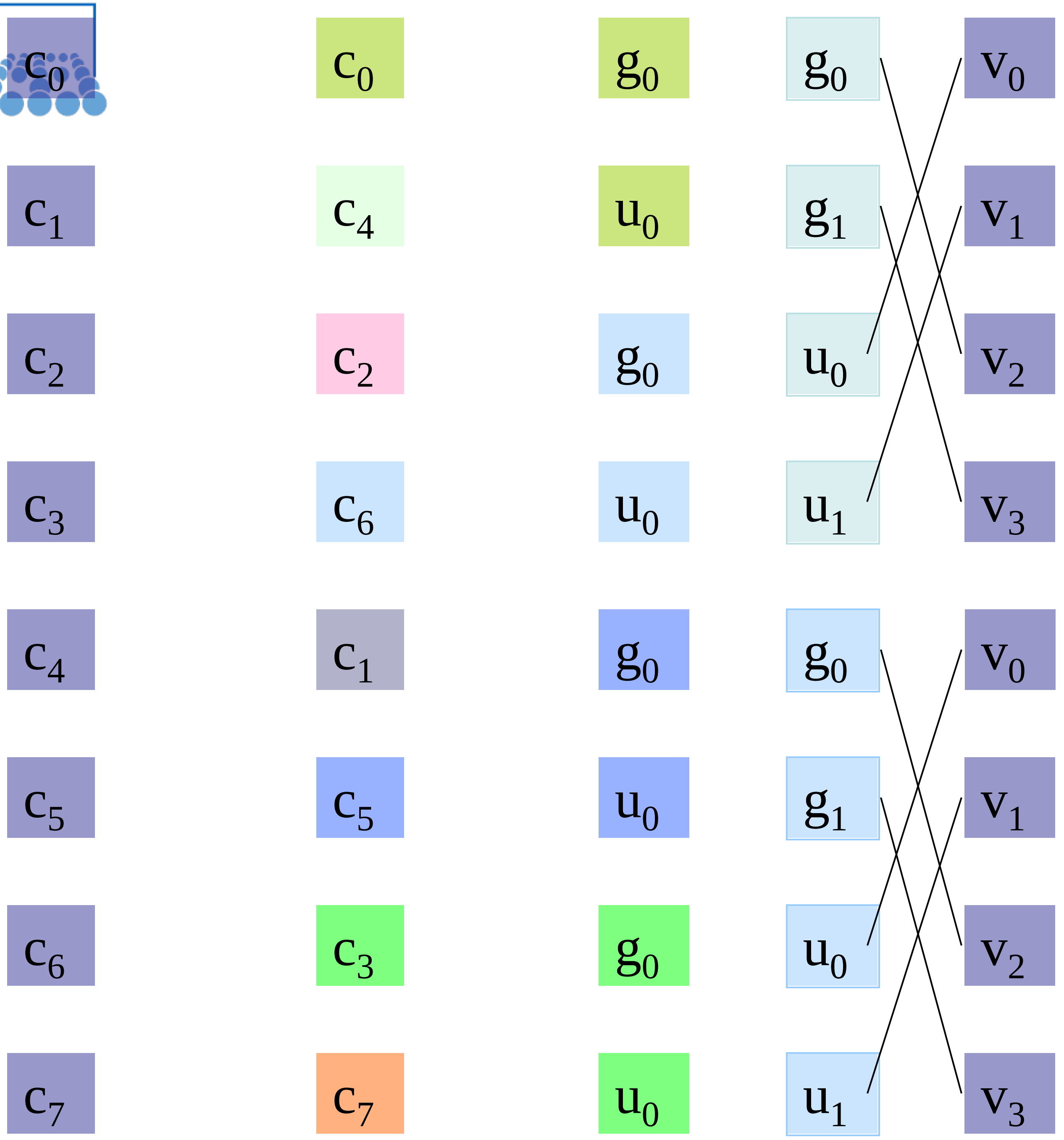
After permutation, the butterflies have to be applied between elements of certain distance.

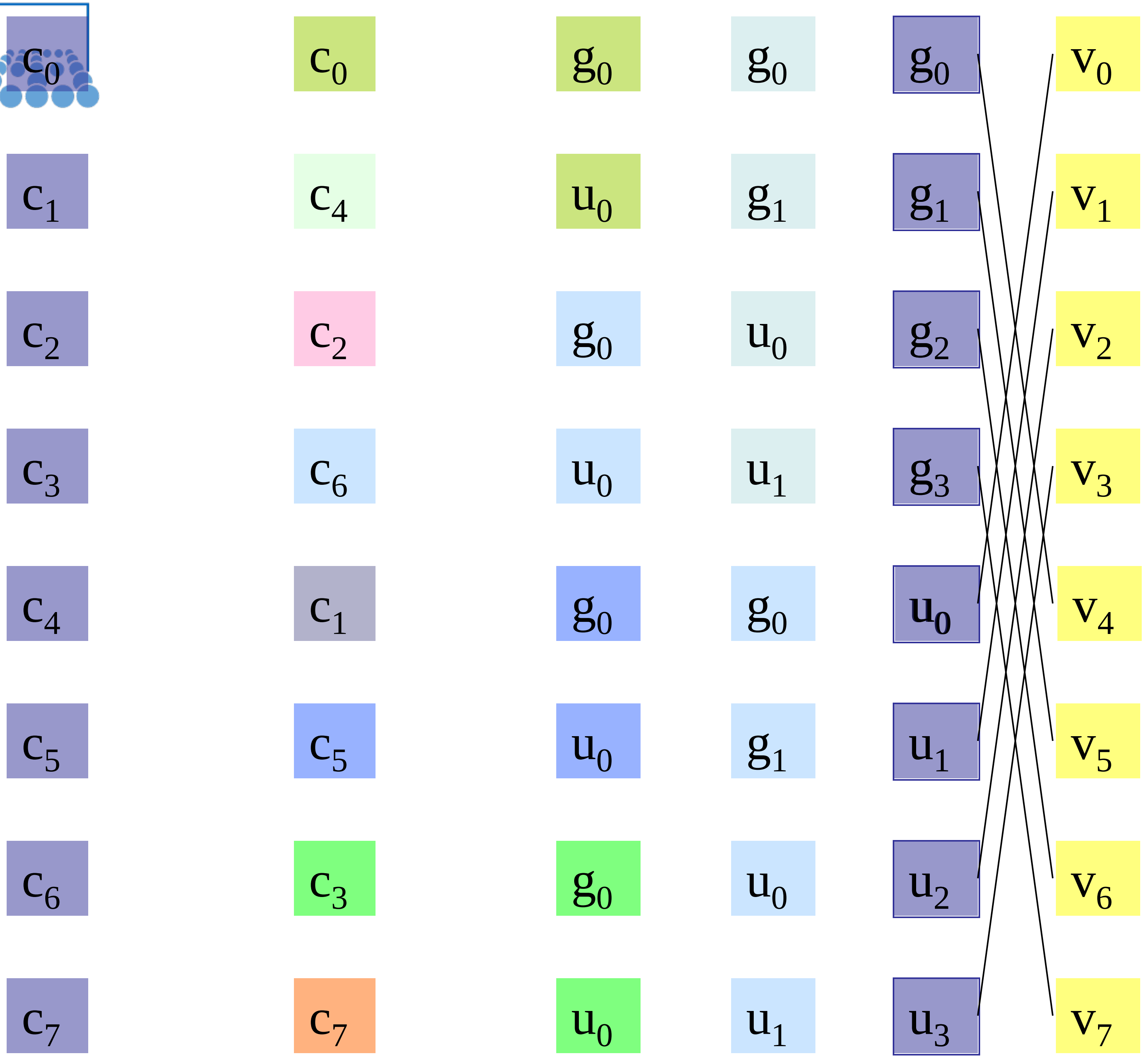
Butterfly:





c_0
c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_0 g_0 v_0
c_4 u_0 v_1
c_2 g_0 v_0
c_6 u_0 v_1
c_1 g_0 v_0
c_5 u_0 v_1
c_3 g_0 v_0
c_7 u_0 v_1


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_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
g_0
g_1
g_2
g_3
u_0
u_1
u_2
u_3
v_0
v_1
v_2
v_3
v_4
v_5
v_6
v_7

FFT in Parallel

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

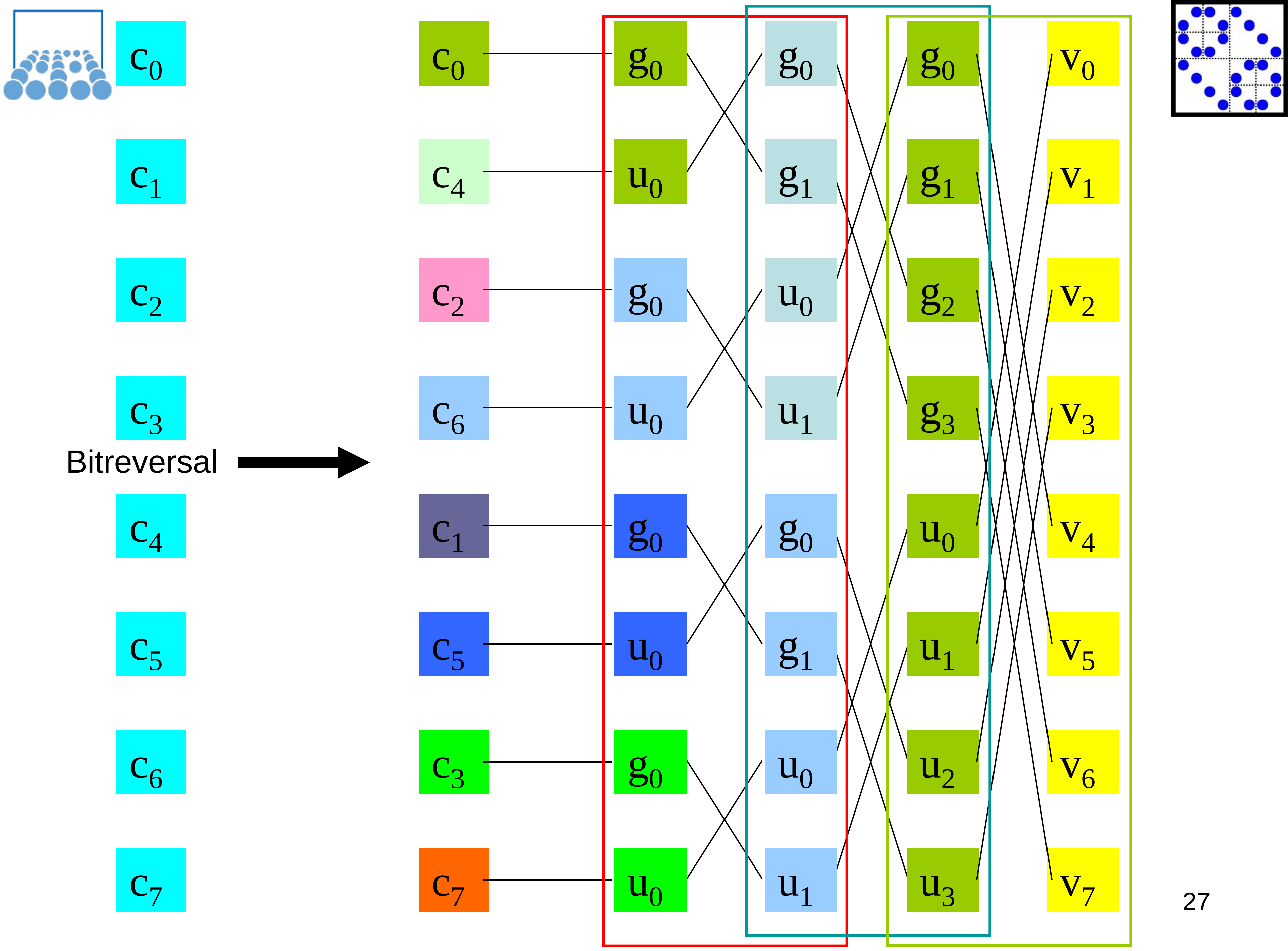
each processor computes Bitreversal k and sends
its entry to the resulting processor $\pi(k)$

Then each two neighbouring processors compute butterfly,

log(n) times with growing distance

n processors, log(n) time steps.

Advantage over trivial parallelization only in number of processors



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

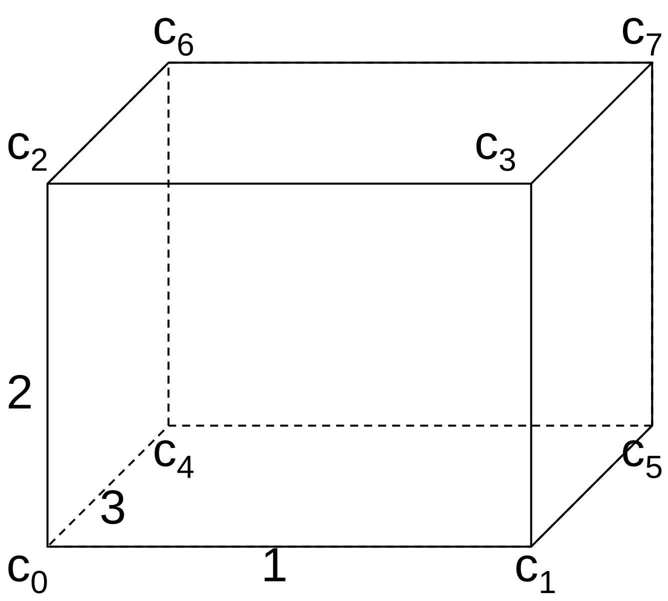
FFT on Hypercube

Distribute entries on vertices of hypercube.

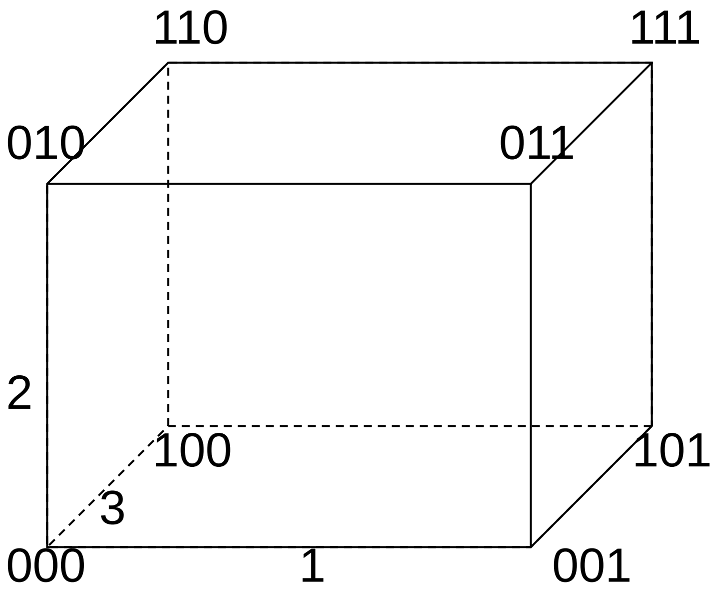
Butterfly has to be applied always between neighbors in distance 1,2,4,…

Hence, the binary indices differ only at one position.

Therefore, butterflies have to computed only between neighboring vertices



c_6
c_7
c_2
c_3
2
c_4
c_5
3
c_0
1
c_1



110
111
010
011
2
100
101
3
000
1
001