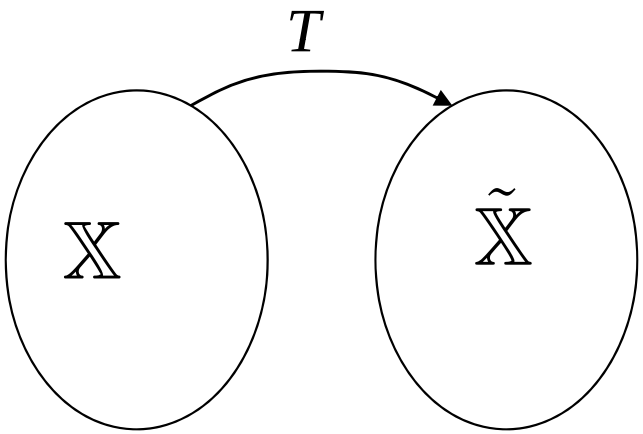
5. Schnelle Algorithmen zur Signal- und Bildverarbeitung

Aufwand einer allgemeinen Abbildung vom Vektorraum $\mathbb{R}^N \rightarrow \mathbb{R}^N$ (linear oder nichtlinear)

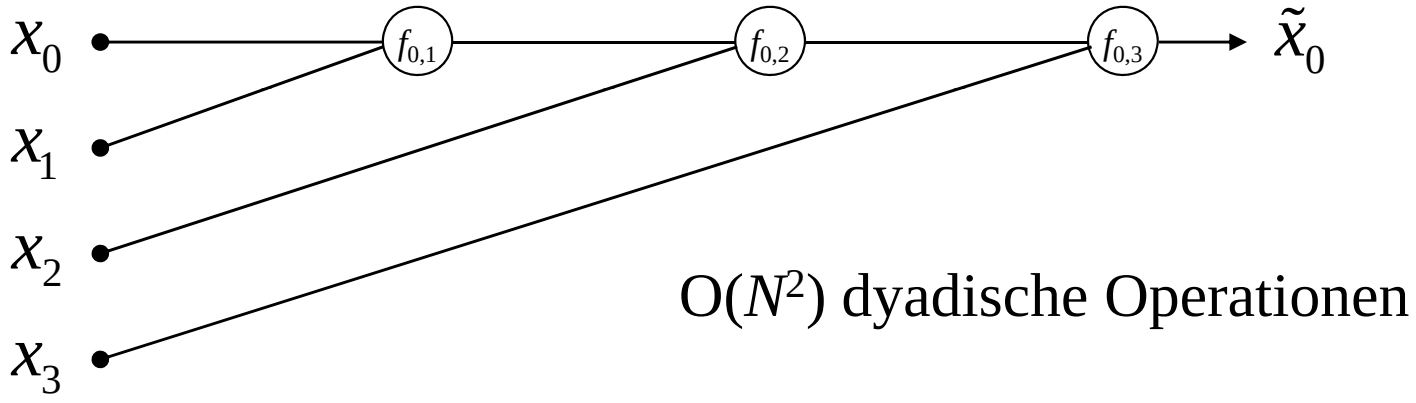
$$\tilde{\mathbf{x}} = T(\mathbf{x}) \quad \text{mit:} \quad \dim(\mathbf{x}) = \dim(\tilde{\mathbf{x}}) = N$$



Für die Abbildung T werden N^2 zweistellige Verknüpfungen benötigt, wenn jeder Eingangswert in aller Allgemeinheit in die Berechnung eines jeden Ausgangswertes eingehen soll.

Dies wird z.B. in dem folgenden Berechnungsschema deutlich:

$$\tilde{x}_j = f_{j,N-1}(\cdots f_{j,3}(f_{j,2}(f_{j,1}(x_0, x_1), x_2), x_3)\cdots, x_{N-1})$$



O(N^2) dyadische Operationen

Z.B. Die lineare Vektorraumoperation:

$$\boxed{\tilde{\mathbf{x}} = \mathbf{W}\mathbf{x}}$$

$$\begin{bmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\tilde{x}_0 = \overbrace{\underbrace{\underbrace{(w_{00} \cdot x_0 + w_{01} \cdot x_1)}_{f_{01}(x_0, x_1)} + w_{02} \cdot x_2}_{f_{02}(\bullet, x_2) = (\bullet) + w_{02} \cdot x_2}}^{f_{03}(\bullet, x_3)} + w_{03} \cdot x_3$$

Dabei wird eine Addition+Multiplikation als eine Operation gezählt.

Eine Klasse schneller, Transformationen durch rekursive Faktorisierung der Transformation

Lässt sich die Transformation hingegen faktorisieren, d.h. kann man die Transformation der Dimension *N* auf zwei Transformationen der halben Dimension und einem Verschmelzungsschritt mit linearem Aufwand zurückführen, so erhält man:

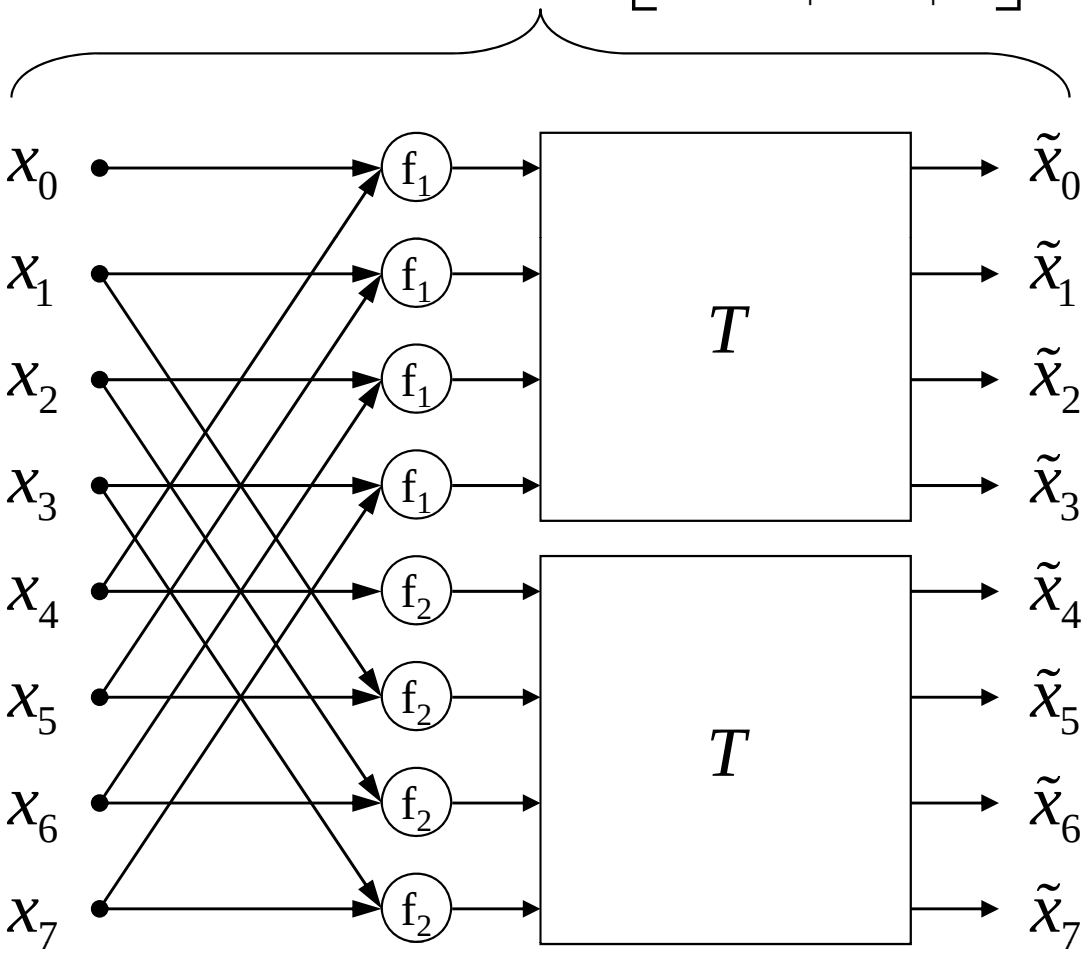
$$\tilde{\mathbf{x}} = T(\mathbf{x}) = \begin{bmatrix} \widetilde{f_1(\mathbf{x}_{1|2}, \mathbf{x}_{2|2})} \\ \widetilde{f_2(\mathbf{x}_{1|2}, \mathbf{x}_{2|2})} \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{x}^{(1)}_{1|2}} \\ \widetilde{\mathbf{x}^{(1)}_{2|2}} \end{bmatrix}$$

Dabei bedeutet $f(\mathbf{x},\mathbf{y})$ die Anwendung der zweistelligen Verknüpfung f auf korrespondierende Elemente der beiden Vektoren **x** und **y**.

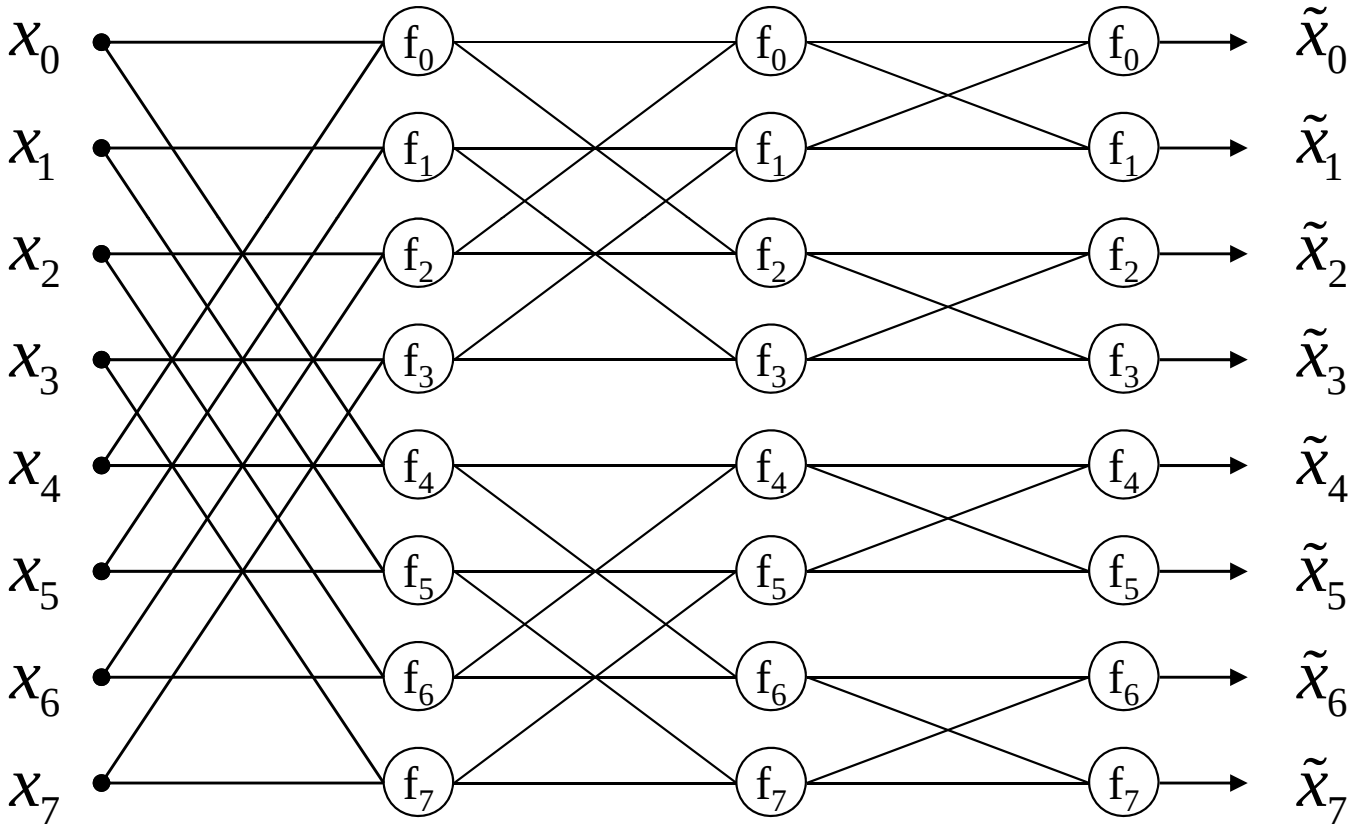
Nach mindestens (ld *N)* Schichten, geht jedes Eingangselement x_i in die Berechnung eines jeden Ausgangselementes $\tilde{x}_j$ ein!

Rekursive Faktorisierung der Transformation T

$$\tilde{\mathbf{x}} = T(\mathbf{x}) = \begin{bmatrix} \widetilde{f_1(\mathbf{x}_{1|2}, \mathbf{x}_{2|2})} \\ \widetilde{f_2(\mathbf{x}_{1|2}, \mathbf{x}_{2|2})} \end{bmatrix}$$



Butterfly- oder In-Place-Signalflußgraph der schnellen Transformation T



Berechnungskomplexität

Durch die Faktorisierung ergibt sich ein Aufwand für $N=2^n$ von:

$$(1 \cdot N + 2 \cdot N/2 + 4 \cdot N/4 + \cdots N/2 \cdot 2) = N \cdot ld(N) = N \cdot n \quad \text{Operationen}$$

D.h. ein Aufwandsgewinn von: $$\frac{N^2}{N \log_2 N} = \frac{N}{\log_2 N}$$

Für $N=2^{10}=1024$ ergibt sich bereits ein Gewinn von $1024/10 \approx 100$.

Laufzeitgewinn

N	N^2	$N \operatorname{ld} N$	Gewinn: $\frac{N^2}{N \cdot \operatorname{ld} N} = \frac{N}{\operatorname{ld} N}$
100	10.000	664	15
500	250.000	4.483	55
1.000	10^6	10^4	100
$10^3 \cdot 10^3 = 10^6$	10^{12}	$2 \cdot 10^6$	50.000

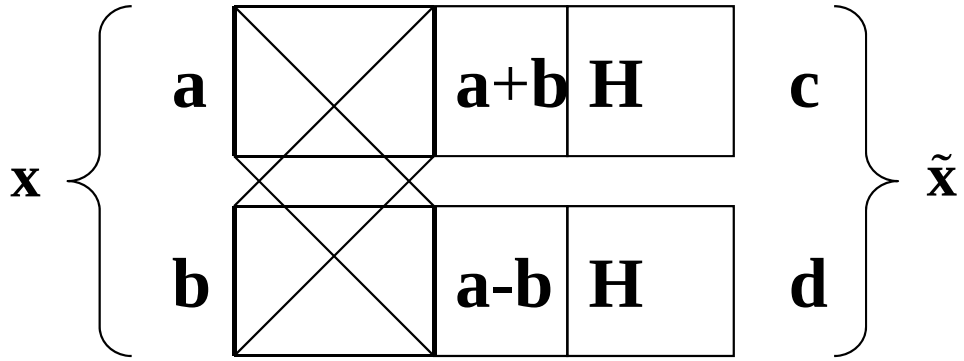
Konsequenzen der Faktorisierung

1. Schneller Algorithmus mit ($N \cdot \log_2 N$) zweistelligen Verknüpfungen
2. In-Place-Algorithmus
3. Modulare Nutzung
   - Hardware: modularer Aufbau aus Bausteinen kleinerer Dimension
   - Software: modulare Nutzung kleinerer Teiltransformationen z.Bsp. bei beschränktem Haupspeicher
4. Rekursion sehr leistungsfähig für Beweisführung (vollständige Induktion)

Die schnelle Walsh-Hadamard-Transformation

Rekursive Definition von **H**: $\tilde{\mathbf{x}} = \mathbf{H}\mathbf{x}$ mit: $\mathbf{H}_N = \begin{bmatrix} \mathbf{H}_{N/2} & \mathbf{H}_{N/2} \\ \mathbf{H}_{N/2} & -\mathbf{H}_{N/2} \end{bmatrix}$

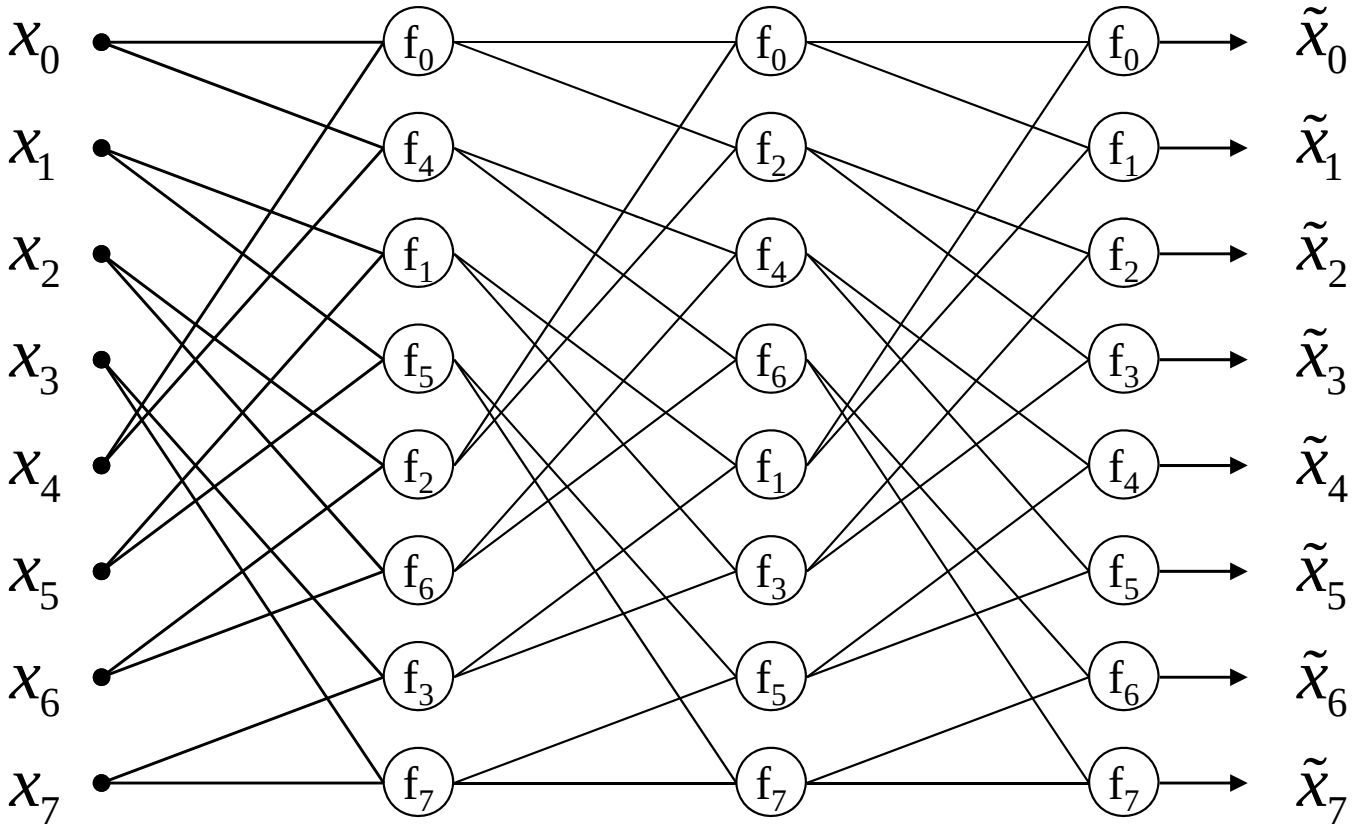
$$\Rightarrow \quad \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{H} \\ \mathbf{H} & -\mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{Ha}+\mathbf{Hb} \\ \mathbf{Ha}-\mathbf{Hb} \end{bmatrix} = \begin{bmatrix} \mathbf{H}(\mathbf{a}+\mathbf{b}) \\ \mathbf{H}(\mathbf{a}-\mathbf{b}) \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{a}+\mathbf{b} \\ \mathbf{a}-\mathbf{b} \end{bmatrix}$$



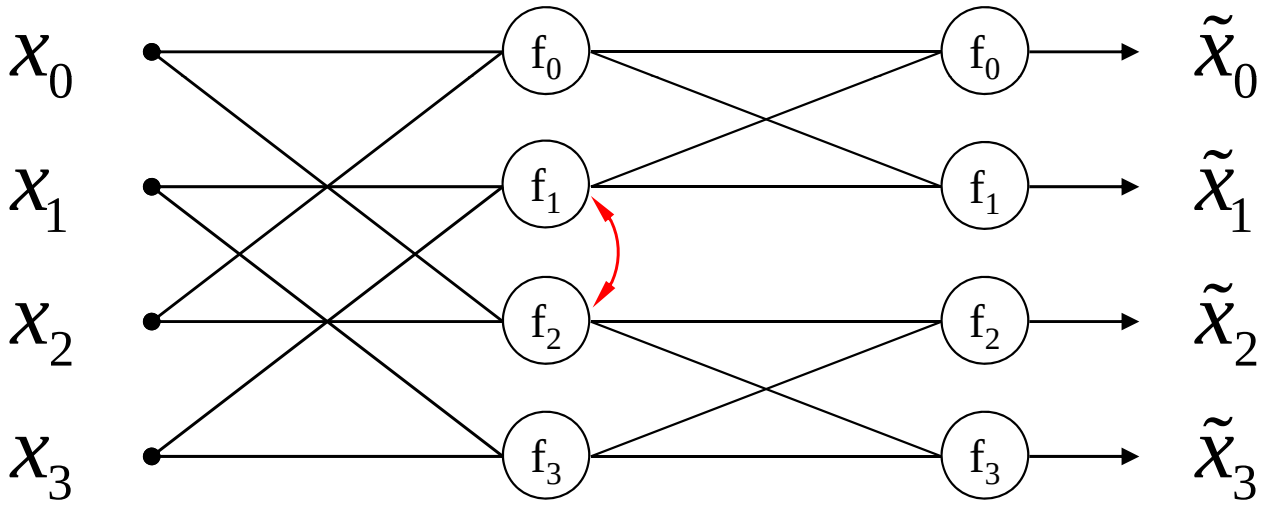
Bei Fortsetzung der Rekursion wird die H-Matrix auf Kosten linearer Verschmelzungsschritte auf Diagonalform reduziert!

$t(N) = 2 \cdot t(N/2) + N \Rightarrow t(N) = O(N \operatorname{ld} N)$ Rekursion mit linearem Verschmelzungsaufwand (Folie 27 mit $a = c = 2$)

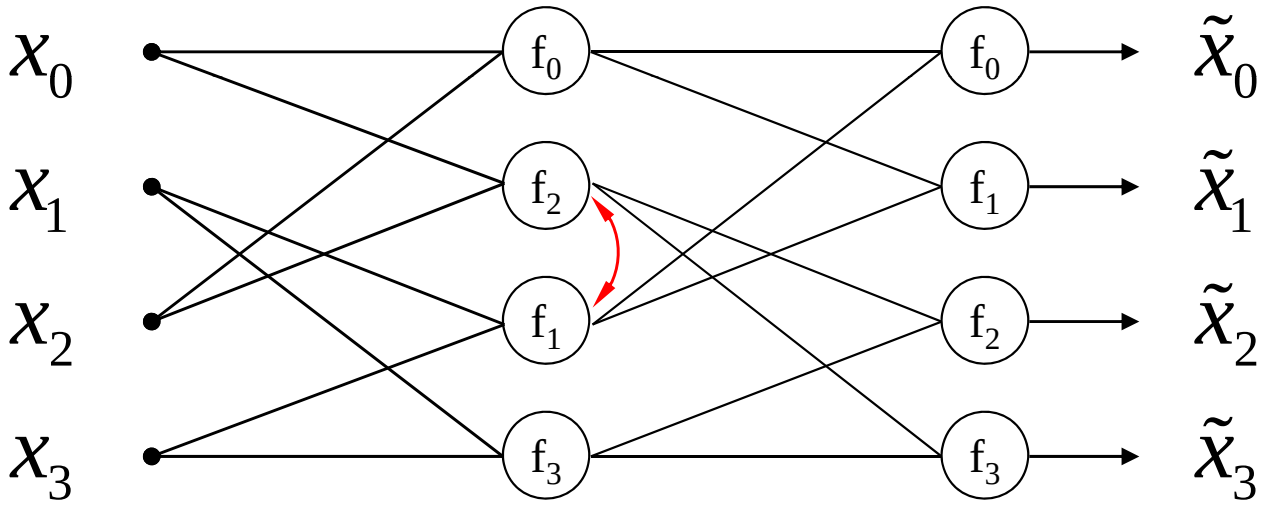
Homogener oder de Bruijn-Signalflußgraph der schnellen Transformation T



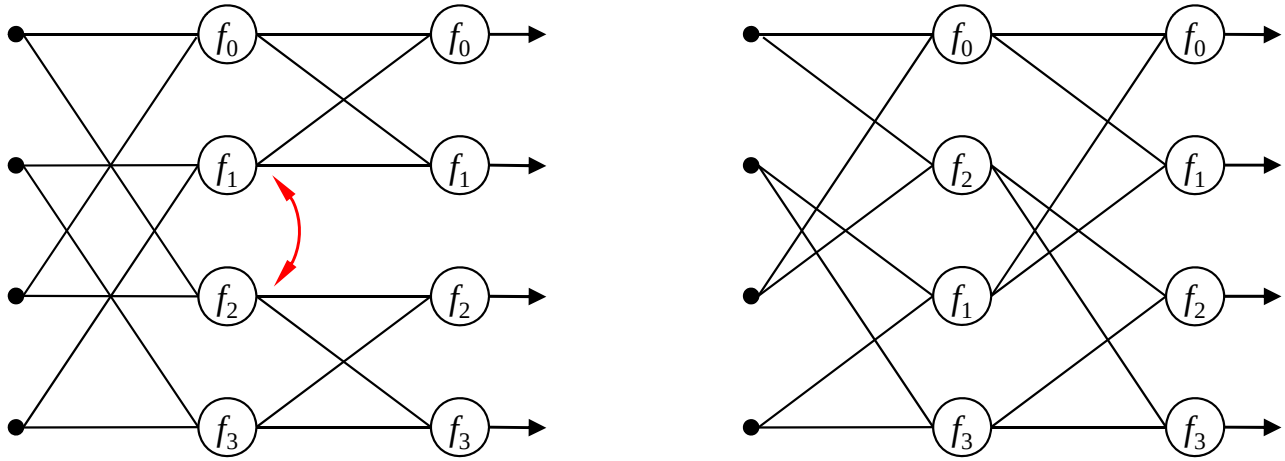
Übergang zum homogenen Graphen durch Permutation der Knoten



Übergang zum homogenen Graphen durch Permutation der Knoten



Übergang vom De Bruijn-Graph zum homogenen Graphen



Knoten verschieben!

Verbindungen werden mitgenommen

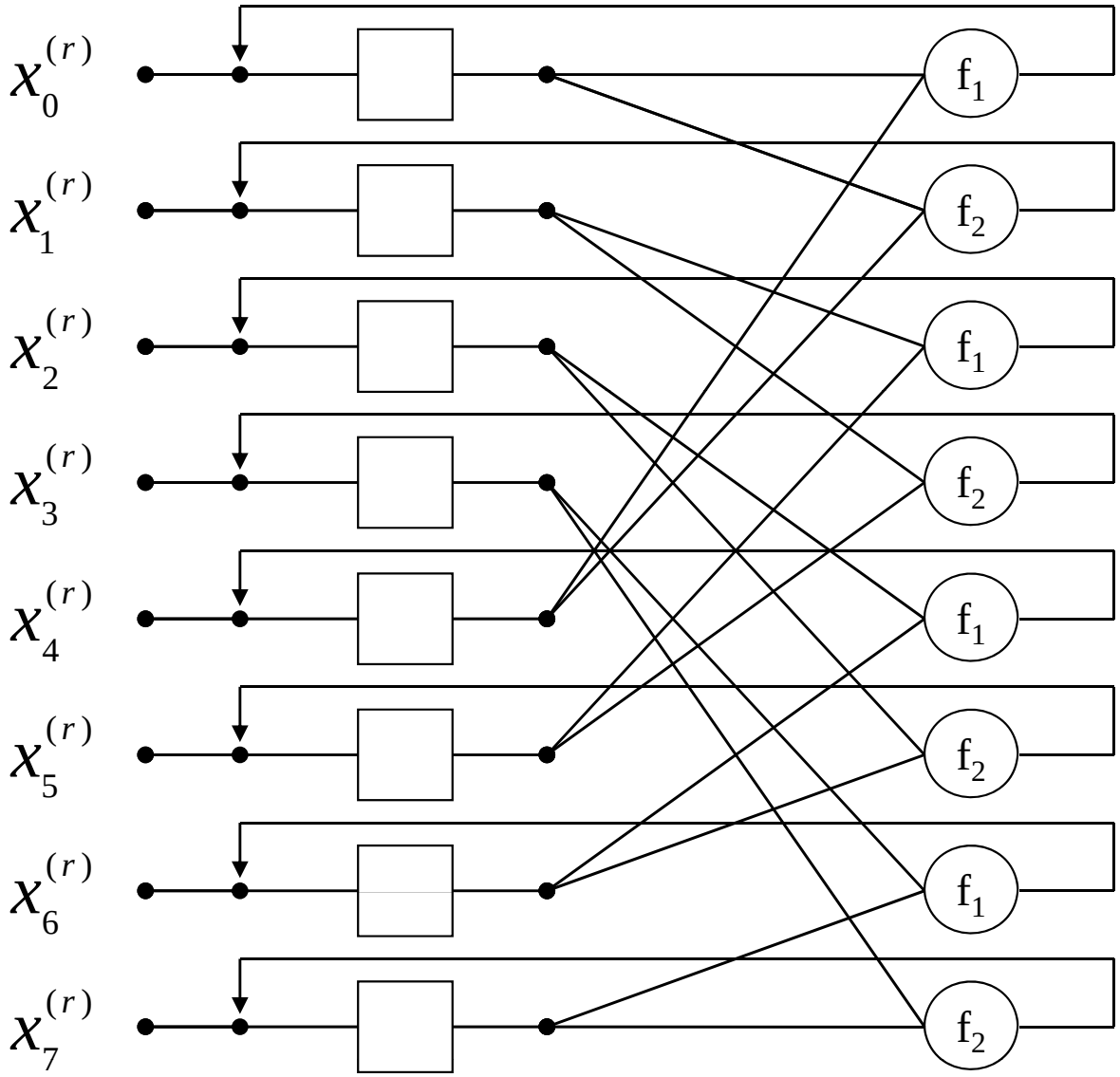
Die Umwandlung des Butterfly- in den homogenen Graphen für eine allgemeine Basis-B Faktorisierung durch Permutation der Knoten

Die Verknüpfung f_r von Schicht j des Butterfly-Graphen wandert an die Stelle r', mit:

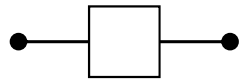
$$r'^{(j)} = \tau_j(r_B^{(0)}) = \tau_1(r_B^{(j-1)})$$

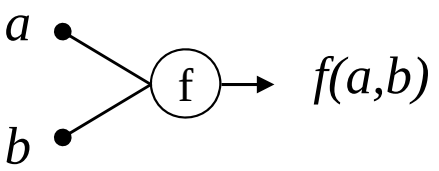
τ_j bezeichnet j zyklische Verschiebungen nach links von r, dargestellt im Basis-B-Zahlensystem mit $n=\log_B N$ Stellen

Allgemeiner paralleler Signalprozessor (N=8)



Speicherzelle





arithm.
Verknüpfung

Die Transformation T in APL

```
[0] Z←T X;N
[1] ⍝ T REKURSIV (mit Bit-Reversal)
[2] →(1>N←(,⍴Z←X)÷2)/0
[3] Z←,(T((N↑X) F1 (N↓X))),[1.5](T((N↑X) F2 (N↓X)))
```

```
[0] Z←T X;I;LN;NH
[1] ⍝ T ITERATIV
[2] LN←1+2⊛NH←(⍴Z←X)÷2
[3] I←1
[4] M:Z←,((NH↑Z) F1 (NH↓Z)),[1.5]((NH↑Z) F2 (NH↓Z))
[5] →(LN≥I←I+1)/M
```

Schnelle Fouriertransformation FFT

```
[0] Z←FFTR X;N;W
[1] ⍝ FFT REKURSIVE DEFINITION
[2] ⍝ MIT BIT-REVERSAL
[3] →(1>N←(⍴Z←X)÷2)/0
[4] W←CEXP○(¯1+⍳N)÷N
[5] Z←,(FFTR(N↑X)+(N↓X)),[1.5](FFTR W×(N↑X)-(N↓X))
```

```
[0] Z←CEXP X
[1] Z←¯12○X
```

Schneller CT-in-place-Algorithmus

(in Matlab-Notation)

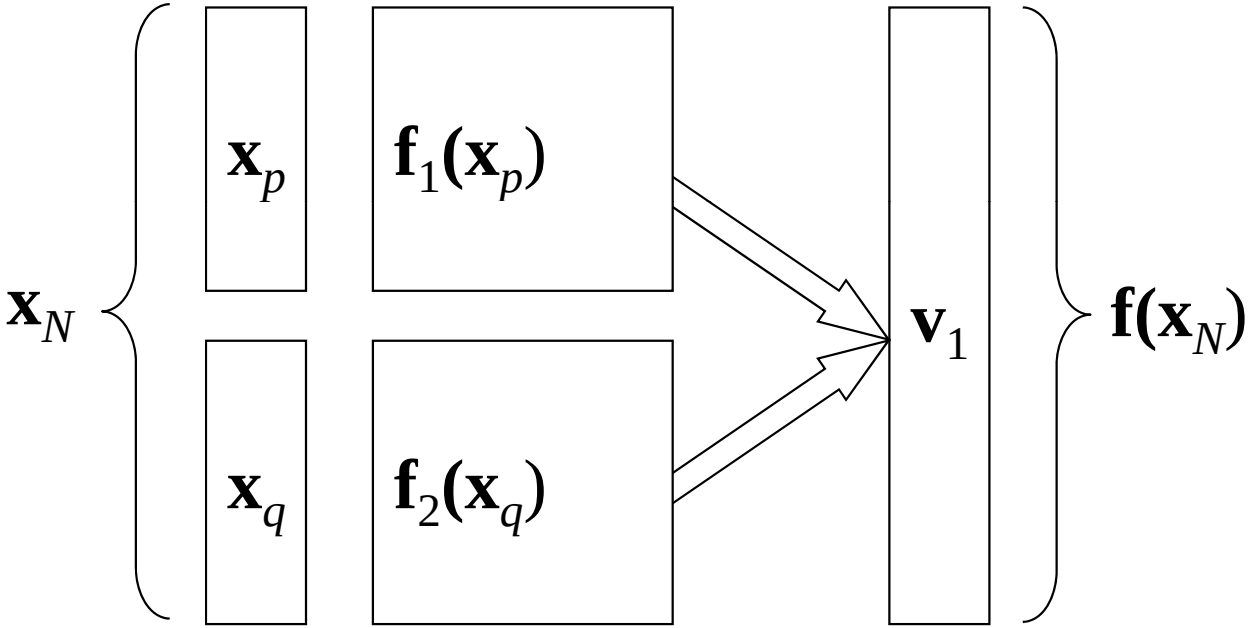
```
function X = ict( X )
%  ICT   Schneller ct-in-place-Algorithmus
%  X = ICT(X) berechnet allgemeinen schnellen Basis-2-Algorithmus.
%  Operatorwahl durch externe Funktionen f1(a,b) und f2(a,b)

n = length( X );       % Dimension des Eingabevektors

ln = log2( n );
np2 = n;

for i = 1:ln,          % fuer jede Schicht
  np = np2;
  np2 = np/2;
  for m =1:np2,        % fuer jeden Knoten eines Teilbaumes
    for j1 = m:np:n,   % fuer jeden Teilbaum
      j2 = j1 + np2;
      t = f2( X(j1), X(j2) );
      X(j1) = f1( X(j1), X(j2) );
      X(j2) = t;
    end
  end
end
```

z.B. Walsh-Transf:

```
function
f1(a,b) = a+b;
```

```
function
f2(a,b) = a-b;
```

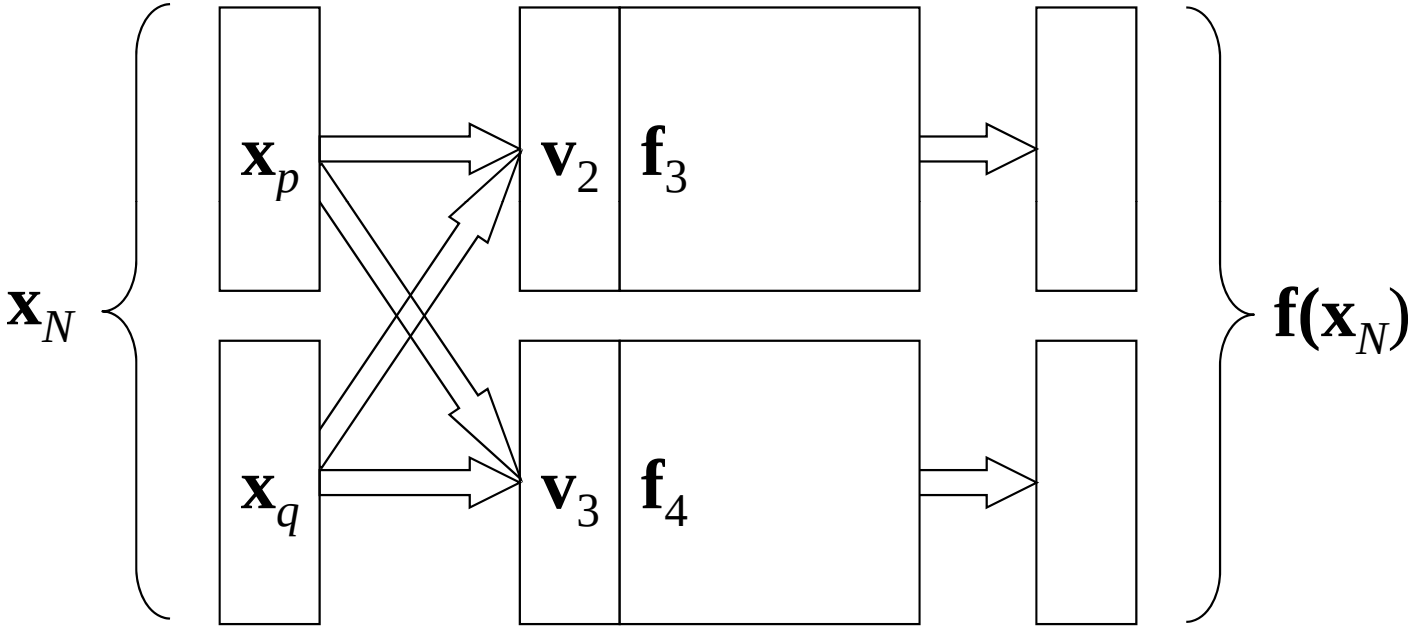
Erste kanonische Zerlegungsstrategie:
separat auswerten und Verschmelzung am Schluss



$$\mathbf{f}(\mathbf{x}_N) = \mathbf{v}_1(\mathbf{f}_1(\mathbf{x}_p), \mathbf{f}_2(\mathbf{x}_q))$$

Zweite kanonische Zerlegungsstrategie

Verschmelzung zu Beginn, dann separat auswerten



$$\mathbf{f}(\mathbf{x}_N) = \mathbf{f}_3(\mathbf{v}_2(\mathbf{x}_p, \mathbf{x}_q)), \mathbf{f}_4(\mathbf{v}_3(\mathbf{x}_p, \mathbf{x}_q))$$

Beispiel: Zwei Zerlegungsstrategien zum Sortieren von N Zahlen

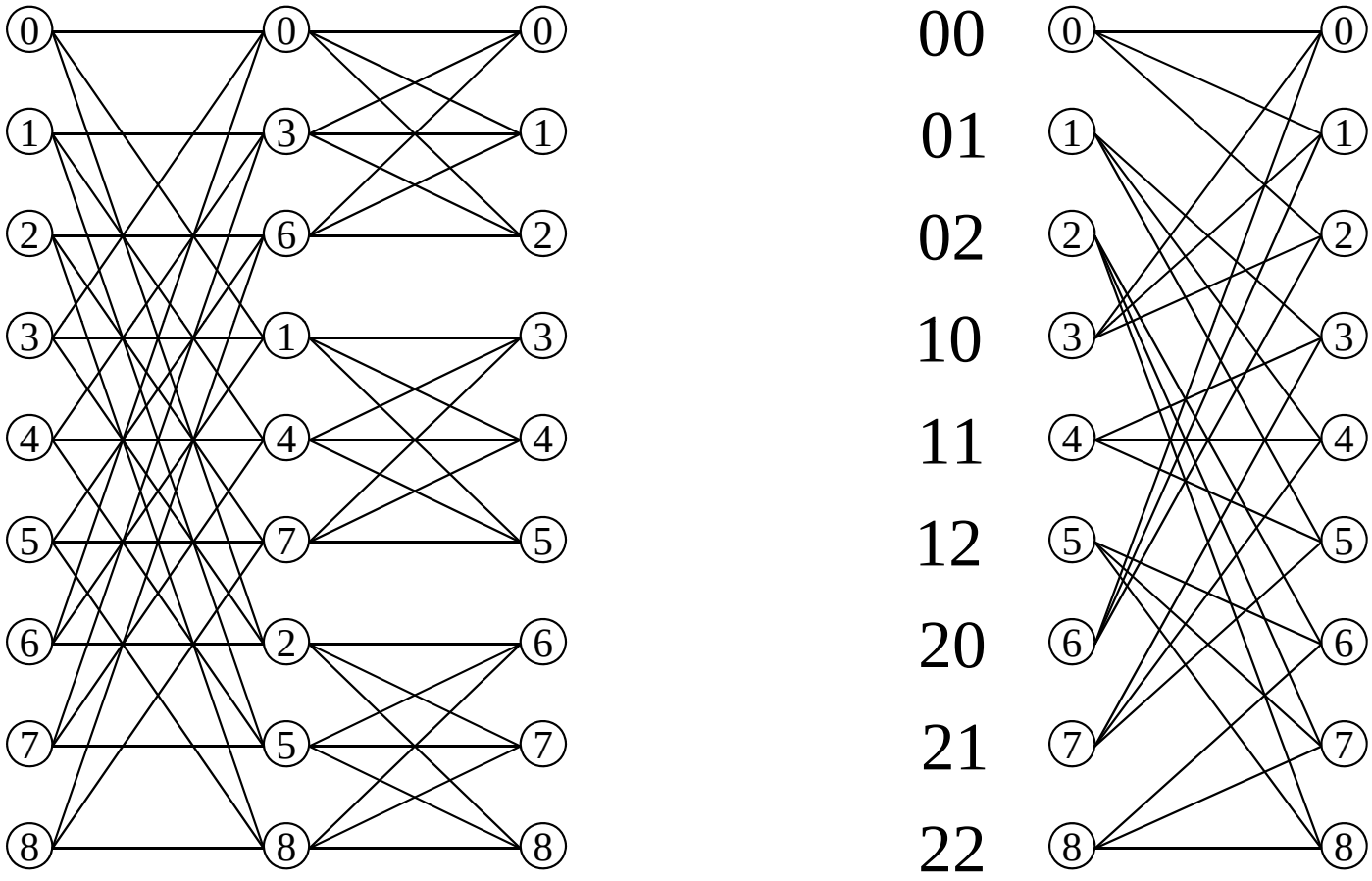
- Zu I (Merge-Sort):
  - Trenne die Daten in zwei Häften (ungeordnet trennen)
  - Sortiere beide Hälften (entkoppelt bearbeiten)
  - Verschmelze die sortierten Hälften zu einem geordneten Datensatz der Länge *N* (geordnet zusammenführen)
- Zu II (Quicksort):
  - Separiere die Ausgangsdaten in zwei Hälften derart, dass alle Elemente der ersten Hälfte kleiner sind als die der zweiten Hälfte (geordnet trennen)
  - Sortiere beide Hälften (entkoppelt bearbeiten)
  - Erstelle ein Gesamtergebnis durch einfaches Aneinanderreihen der Teilergebnisse (einfach, da unabhängig voneinander zusammenfügen)

Rekursive Basis-3 Faktorisierung der Transformation T

$$\tilde{\mathbf{x}} = T(\mathbf{x}) = \begin{bmatrix} \overbrace{f_1(\mathbf{x}_{1|3}, \mathbf{x}_{2|3}, \mathbf{x}_{3|3})} \\ \overbrace{f_2(\mathbf{x}_{1|3}, \mathbf{x}_{2|3}, \mathbf{x}_{3|3})} \\ \overbrace{f_3(\mathbf{x}_{1|3}, \mathbf{x}_{2|3}, \mathbf{x}_{3|3})} \end{bmatrix}$$

Dabei bedeutet $f(\mathbf{x},\mathbf{y},\mathbf{z})$ die Anwendung der dreistelligen Verknüpfung f auf korrespondierende Elemete der drei Vektoren $\mathbf{x}$,$\mathbf{y}$ und $\mathbf{z}$.

Die beiden kanonischen Verarbeitungsgraphen für eine Basis-3-Faktorisierung (Butterfly u. De Bruijn)



Nach mindestens ($\log_B N$) Schichten, geht jedes Eingangselement $\boldsymbol{x}_i$ in die Berechnung eines jeden Ausgangselementes $\tilde{x}_j$ ein!

Anzahl der dyadischen Operationen bei einem Basis-B-Algorithmus

$(B-1)N \cdot \log_B N$ dyadische Operationen

$\log_B N$ - Schichten

$N \cdot (B-1)$ - Operationen/Schicht

Zum Beispiel:

- *B*=2, *N*=1024:
  10 Schichten
  1024 Operationen/Schicht => 10 240≈10^4 Operationen
- *B*=4, *N*=1024:
  5 Schichten
  3·1024=3072 Operationen/Schicht => 15 360 Operationen

- *B*=*N*, *N*=1024:
  1 Schicht
  N2 Operationen/Schicht => $N^2=1024^2=1\,048\,576 \approx 10^6$ Operationen

Analyse und Rechenregeln rekursiver Zerlegungen

Bei vollständiger Auflösung einer Rekursion erhält man die folgenden asymptotischen Analysen für die Berechnungskomplexität:

I. Konstanter Verschmelzungsaufwand d:

$$t(N) = \begin{cases} b & \text{für } N = 1 \quad \text{mit: } N = c^n \\ a \cdot t(\frac{N}{c}) + d & \text{für } N > 1 \quad \text{mit: } a, b > 0 \end{cases}$$

Asymptotische Komplexität:

$$t(N) = \begin{cases} O(N^{\log_c a}) & \text{für } a \neq 1 \\ O(\log_c N) & \text{für } a = 1 \end{cases}$$

II. Linear wachsender Verschmelzungsaufwand $b \cdot N$:

$$t(N) = \begin{cases} b & \text{für } N = 1 \quad \text{mit: } N = c^n \\ a \cdot t(\frac{N}{c}) + b \cdot N & \text{für } N > 1 \quad \text{mit: } a, b > 0 \end{cases}$$

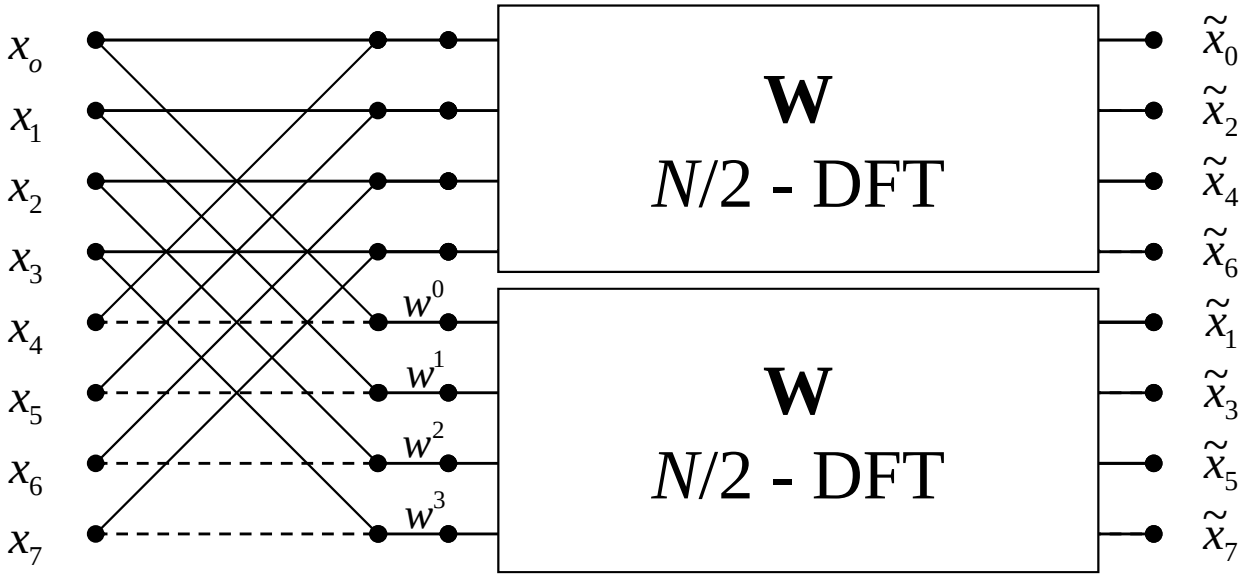
Asymptotische Komplexität:

$$t(N) = \begin{cases} O(N) & \text{für } a < c \\ O(N \log_c N) & \text{für } a = c \\ O(N^{\log_c a}) & \text{für } a > c \end{cases}$$

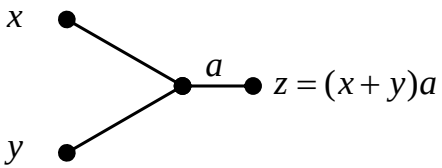
Faktorisierung der DFT => FFT
(Fast Fourier Transform)

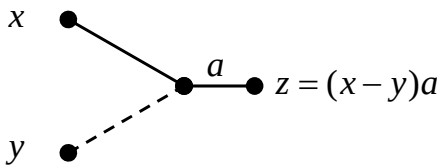
$$f_1(a,b) = a + b$$
$$f_2(a,b) = (a - b)w^i$$

FFT nach dem Sande-Tukey-Algorithmus (decimation in frequency) Originaldaten in natürlicher Ordnung

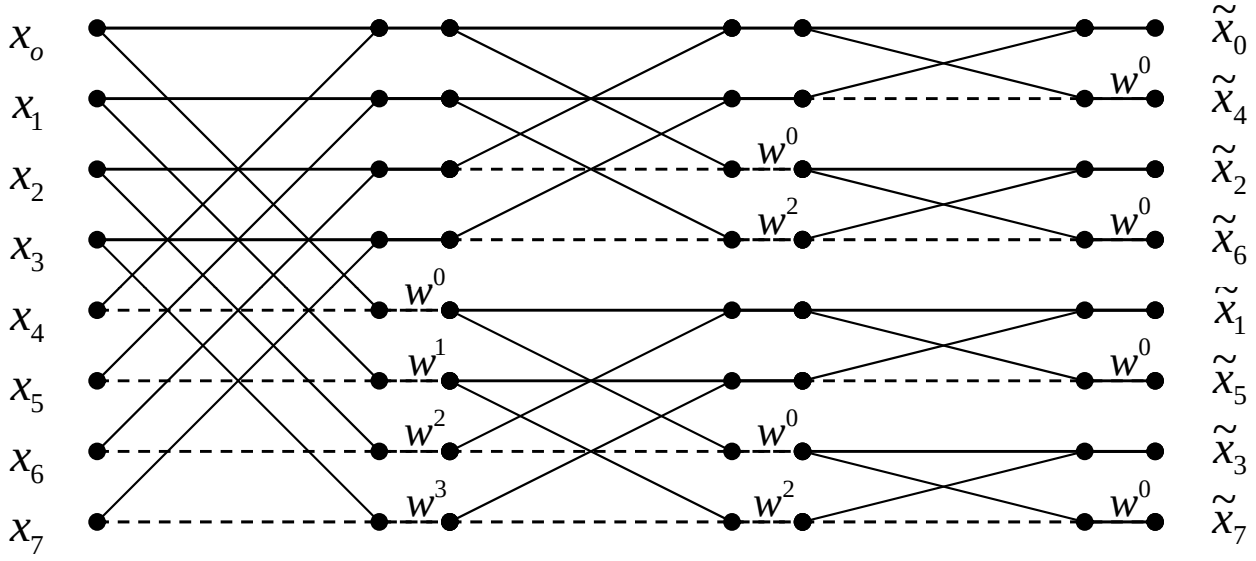


$$w = e^{-j2\pi/N}$$

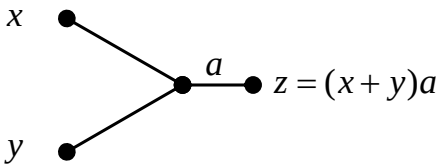


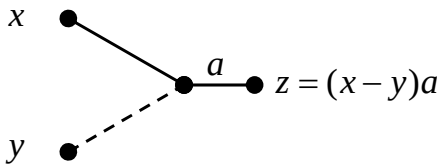


FFT nach dem Sande-Tukey-Algorithmus

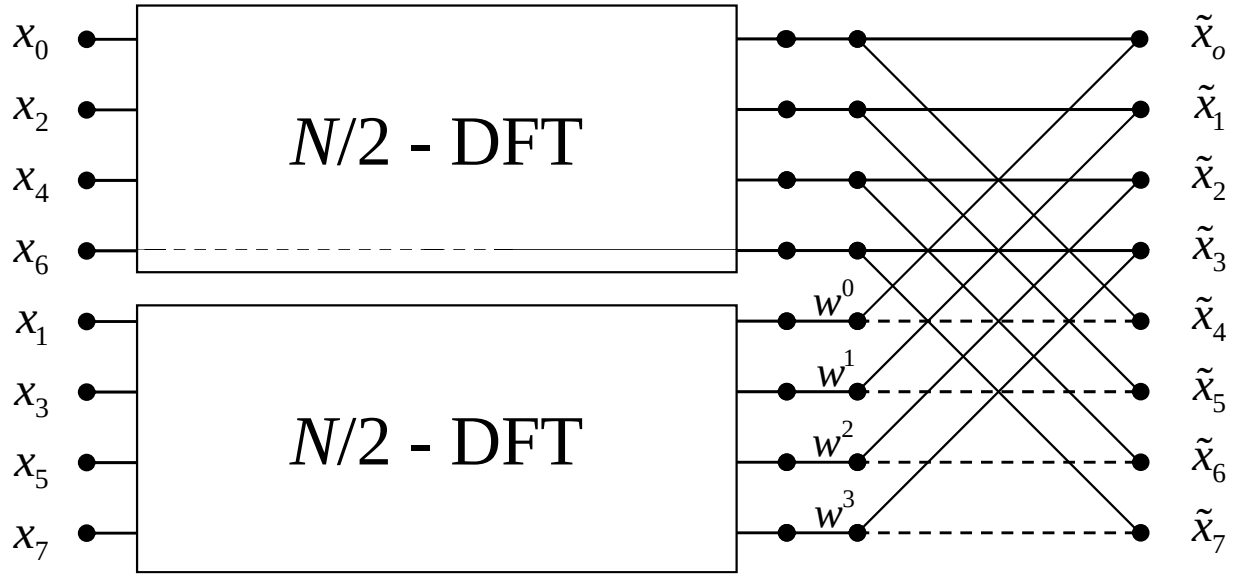


$$w = e^{-j2\pi/N}$$

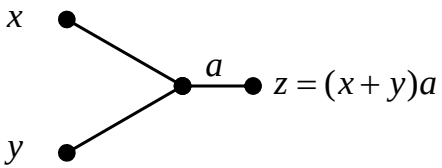


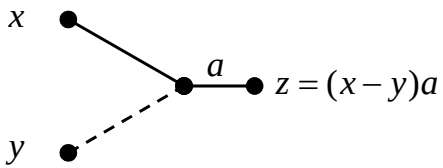


FFT nach dem Cooley-Tukey-Algorithmus (decimation in time) Ergebnisse in natürlicher Ordnung

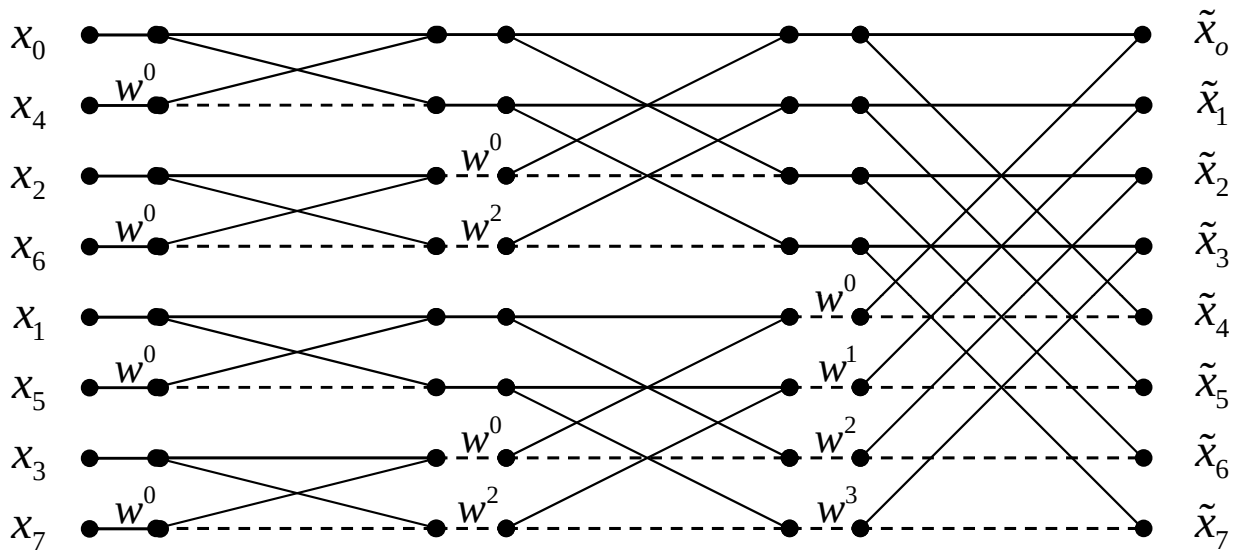


$$w = e^{-j2\pi/N}$$

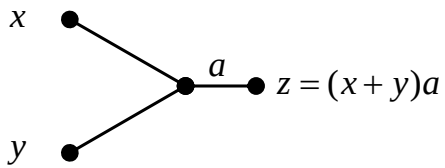


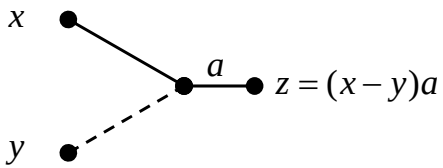


FFT nach dem Cooley-Tukey-Algorithmus



$$w = e^{-j2\pi/N}$$





FFT nach dem Sande-Tukey-Algorithmus
(in Matlab-Notation)

```
function result = FFT(X, N)
% result = FFT(X, N)
%   Eindimensionale FFT (Sande-Tukey-In-Place-Algorithmus)
LN = floor(log2(N));
NP2 = N;
for I=1:LN,
  NP = NP2;
  NP2 = NP/2;
  U = 1.0+ 0i;
  w = exp(i * (-pi/NP2));
  for M=1:NP2,
    for J1=M:NP:N,
      J2 = J1 + NP2;
      T = X(J1) - X(J2);
      X(J1) = X(J1) + X(J2);
      X(J2) = T * U;
    end;
    U=U*w;
  end;
end;
result = BR(X,N);
```

Bit-reversal rekursiv

Gerade und ungerade Zahlen rekursiv entmischen (inverse perfect shuffle)

0	0	0	0	0	0	0
1	0	0	0	1	8	2
2	0	0	1	0	4	4
3	0	0	1	1	12	6
4	0	1	0	0	2	8
5	0	1	0	1	10	10
6	0	1	1	0	6	12
7	0	1	1	1	14	14
8	1	0	0	0	1	1
9	1	0	0	1	9	3
10	1	0	1	0	5	5
11	1	0	1	1	13	7
12	1	1	0	0	3	9
13	1	1	0	1	11	11
14	1	1	1	0	7	13
15	1	1	1	1	15	15

Bit-reversal rekursiv

Gerade und ungerade Zahlen rekursiv entmischen (inverse perfect shuffle)

0	0	0	0	0	0	0	0
1	0	0	0	1	8	2	4
2	0	0	1	0	4	4	8
3	0	0	1	1	12	6	12
4	0	1	0	0	2	8	2
5	0	1	0	1	10	10	6
6	0	1	1	0	6	12	10
7	0	1	1	1	14	14	14
8	1	0	0	0	1	1	1
9	1	0	0	1	9	3	5
10	1	0	1	0	5	5	9
11	1	0	1	1	13	7	13
12	1	1	0	0	3	9	3
13	1	1	0	1	11	11	7
14	1	1	1	0	7	13	11
15	1	1	1	1	15	15	15

Bit-Reversal rekursiv

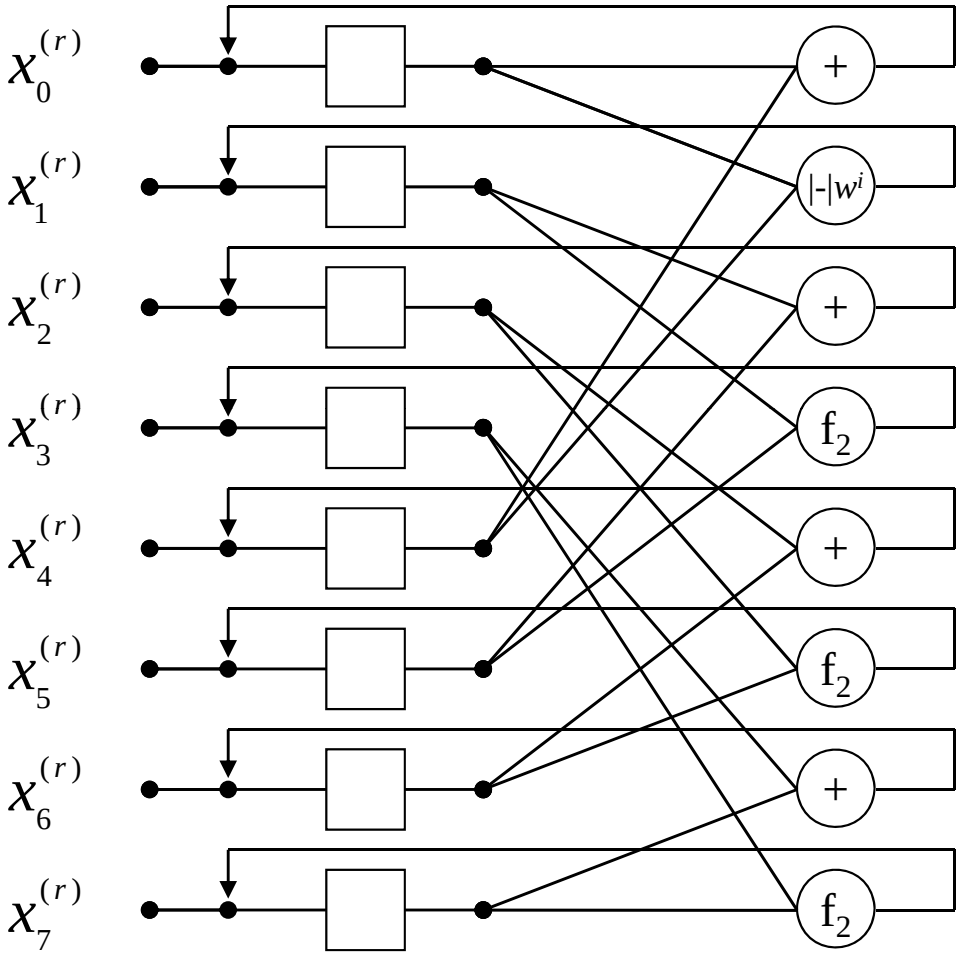
Gerade und ungerade Zahlen rekursiv entmischen (inverse perfect shuffle)

0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	8	2	4	8	
2	0	0	1	0	4	4	8	4	
3	0	0	1	1	12	6	12	12	
4	0	1	0	0	2	8	2	2	
5	0	1	0	1	10	10	6	10	
6	0	1	1	0	6	12	10	6	
7	0	1	1	1	14	14	14	14	
8	1	0	0	0	1	1	1	1	
9	1	0	0	1	9	3	5	9	
10	1	0	1	0	5	5	9	5	
11	1	0	1	1	13	7	13	13	
12	1	1	0	0	3	9	3	3	
13	1	1	0	1	11	11	7	11	
14	1	1	1	0	7	13	11	7	
15	1	1	1	1	15	15	15	15	

Bit-Reversal

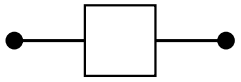
```
function result = BR(X, N)
% result = BR(X, N)
%   Fuehrt ein Bit-reversal auf dem Vektor X der Laenge N durch
NH = N / 2;
J = 1;
for I=1:N-1,
  if (I < J),
    T = X(J);
    X(J) = X(I);
    X(I) = T;
  end;
  K = NH;
  while (K < J),
    J = J-K;
    K = K/2;
  end;
  J=J+K;
end;
result = X;
return;
```

Paralleler FFT-Prozessor (N=8)

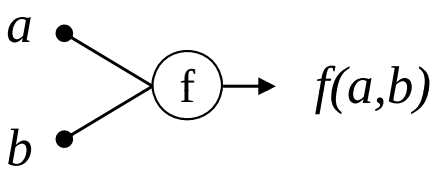


$j=1$	$j=2$	$j=3$
w^0	w^0	w^0
w^1	w^0	w^0
w^2	w^2	w^0
w^3	w^2	w^0

Speicherzelle

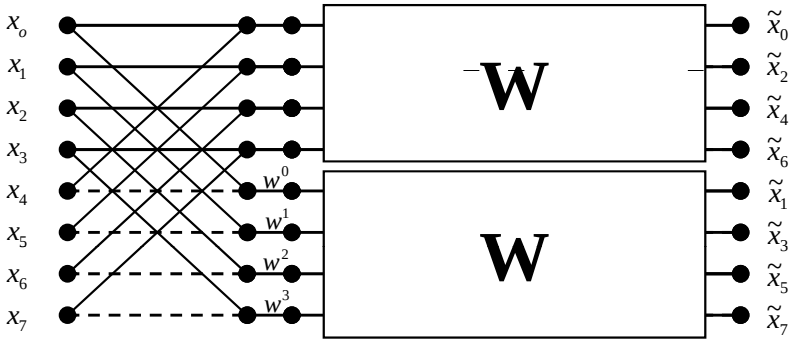


arithm. Verknüpfung



Faktorisierung der Fourier-Matrix **W**

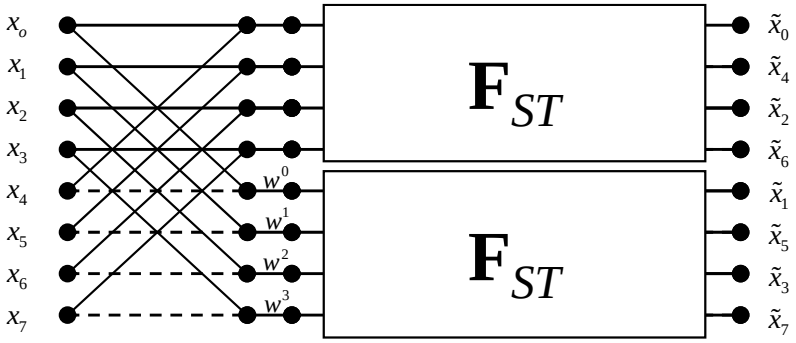
Rekursive Definition des Sande-Tukey-Algorithmus:



$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \mathbf{P}_{ug} \begin{bmatrix} \mathbf{W} & \mathbf{W} \\ \mathbf{WD} & -\mathbf{WD} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$$

mit: $\mathbf{D} = diag(w^0, w^1, \ldots, w^{N/2-1}), \quad w = e^{j\frac{2\pi}{N}}$

Die Permutation $\mathbf{P}_{ug}$ ergibt sich aus der rekursiven Zerlegung des Bit-Reversal in der ersten Stufe.



$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \mathbf{P}_{br} \begin{bmatrix} \mathbf{F}_{ST} & \mathbf{F}_{ST} \\ \mathbf{F}_{ST}\mathbf{D} & -\mathbf{F}_{ST}\mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$$

mit: $\mathbf{D} = diag(w^0, w^1, \ldots, w^{N/2-1}), \quad w = e^{j\frac{2\pi}{N}}$

Faktorisierung der Fourier-Matrix **W**

Die Fouriermatrix kann ähnlich wie die Walsh-Hadamard-Matrix faktorisiert werden.
Aufbauend auf die rekursive Definition des Sande-Tukey-Algorithmus erhält man:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \mathbf{P}_{br} \begin{bmatrix} \mathbf{F}_{ST} & \mathbf{F}_{ST} \\ \mathbf{F}_{ST}\mathbf{D} & -\mathbf{F}_{ST}\mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \mathbf{P}_{ug} \begin{bmatrix} \mathbf{W} & \mathbf{W} \\ \mathbf{WD} & -\mathbf{WD} \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$$

$$= \mathbf{P}_{ug} \begin{bmatrix} \mathbf{W}(\mathbf{a}+\mathbf{b}) \\ \mathbf{WD}(\mathbf{a}-\mathbf{b}) \end{bmatrix} = \mathbf{P}_{ug} \begin{bmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{WD} \end{bmatrix} \begin{bmatrix} \mathbf{a}+\mathbf{b} \\ \mathbf{a}-\mathbf{b} \end{bmatrix}$$

mit: $\mathbf{D} = diag(w^0, w^1, \ldots, w^{N/2-1}), \quad w = e^{j\frac{2\pi}{N}}$

Statt einer voll besetzten Matrix erhält man eine ausgedünnte Blockdiagonalmatrix auf Kosten eines linearen Verschmelzungsaufwandes.
Fourier-Matrix hier in Bit-Reversal zeilenpermutierter Form (nicht frequenzgeordnet), ähnlich wie beim Unterschied Hadamard-/Walsh-Matrix!
Bei Fortsetzung der Rekursion wird die F-Matrix auf Kosten linearer Verschmelzungsschritte auf Diagonalform reduziert!

$$t(N) = 2 \cdot t(N/2) + N \Rightarrow t(N) = O(N \operatorname{ld} N)$$

Rekursion mit linearem Verschmelzungsaufwand

Übergang von Cooley-Tukey zu Sande-Tukey-Algorithmus

Aus der rekursiven Definition von Cooley-Tukey und Sande-Tukey-Algorithmus ergibt sich:

$$\text{Cooley-Tukey:}\quad \tilde{\mathbf{x}} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \mathbf{F}_{CT} & \mathbf{D}\mathbf{F}_{CT} \\ \mathbf{F}_{CT} & -\mathbf{D}\mathbf{F}_{CT} \end{bmatrix} \mathbf{P}_{br}\mathbf{x} = \mathbf{F}_{CT}\mathbf{P}_{br}\mathbf{x}$$

$$\text{Sande-Tukey:}\quad \tilde{\mathbf{x}} = \mathbf{W}\mathbf{x} = \mathbf{P}_{br} \begin{bmatrix} \mathbf{F}_{ST} & \mathbf{F}_{ST} \\ \mathbf{F}_{ST}\mathbf{D} & -\mathbf{F}_{ST}\mathbf{D} \end{bmatrix} \mathbf{x} = \mathbf{P}_{br}\mathbf{F}_{ST}\mathbf{x}$$

Die Permutationsmatrix $\mathbf{P}_{br}\mathbf{x}$ realisiert die Bit-Reversal-Permutation und es gilt:

$$\mathbf{P}_{br}^{-1} = \mathbf{P}_{br}^{T} = \mathbf{P}_{br}$$

Vergleicht man die beiden rekursiven Definitionen, so erhält man unmittelbar:

$\mathbf{F}_{ST} = \mathbf{P}_{br}\mathbf{W}$ BR-Permutation der Zeilen von $\mathbf{W}$

$\mathbf{F}_{CT} = \mathbf{W}\mathbf{P}_{br}$ BR-Permutation der Spalten von $\mathbf{W}$

$$\Rightarrow \boxed{\mathbf{F}_{CT}^{T} = \left(\mathbf{W}\mathbf{P}_{br}\right)^{T} = \mathbf{P}_{br}^{T}\mathbf{W}^{T} = \mathbf{P}_{br}\mathbf{W} = \mathbf{F}_{ST}}$$

Es gilt: $(\mathbf{A}\mathbf{B})^{T} = \mathbf{B}^{T}\mathbf{A}^{T}$

Übergang von Cooley-Tukey zu Sande-Tukey-Algorithmus

Die frequenzgeordnete Fouriermatrix ergibt sich somit gemäß:

$$\mathbf{W} = \mathbf{P}_{br}\mathbf{F}_{ST} = \mathbf{F}_{CT}\mathbf{P}_{br}$$

und damit auch:

$$\mathbf{F}_{ST} = \mathbf{P}_{br}^{T}\mathbf{F}_{CT}\mathbf{P}_{br} = \mathbf{P}_{br}\mathbf{F}_{CT}\mathbf{P}_{br}$$

Und somit kann man beide Transformationen beschreiben als:

$$\tilde{\mathbf{x}} = \mathbf{W}\mathbf{x} = \underbrace{\mathbf{P}_{br}\mathbf{F}_{ST}\mathbf{x}}_{\substack{\text{zuerst Transf.}\\ \text{dann BR}}} = \underbrace{\mathbf{F}_{CT}\mathbf{P}_{br}\mathbf{x}}_{\substack{\text{zuerst BR}\\ \text{dann Transf}}}$$

Dabei bedeutet:

$\mathbf{F}\mathbf{P}_{br}$ eine BR-Permutation der Spalten

$\mathbf{P}_{br}\mathbf{F}$ eine BR-Permutation der Zeilen

Transformation zweier *reeller* Folgen **a** und **b** mit einer komplexen FFT

(z.B. zur schnellen Faltung zwischen **a** und **b**)

Man verteilt die beiden reellen Folgen **a** und **b** auf den Real- und Imaginärteil einer Folge **x** und transformiert komplex:

$$\mathbf{x}(n) = \mathbf{a}(n) + j\mathbf{b}(n) \overbrace{\Rightarrow}^{\text{Linearität}} \tilde{\mathbf{x}}(k) = \tilde{\mathbf{a}}(k) + j\tilde{\mathbf{b}}(k)$$

Aus der Tatsache, dass die Fouriertransformierte einer *reellen* Sequenz konj. symmetrisch und die einer imaginären Sequenz konj. antisymmetrisch ist folgt:

$$\tilde{\mathbf{a}}(k) = \tilde{\mathbf{x}}_e(k) = \tfrac{1}{2}(\tilde{\mathbf{x}}(k) + \tilde{\mathbf{x}}^*(-k)) \quad \text{konj. symmetrisch}$$

$$j\tilde{\mathbf{b}}(k) = \tilde{\mathbf{x}}_o(k) = \tfrac{1}{2}(\tilde{\mathbf{x}}(k) - \tilde{\mathbf{x}}^*(-k)) \quad \text{konj. antisymmetrisch}$$

Transformation von *reellen* Folgen der Länge $2N$ mit *einer* komplexen FFT der Länge N

Man verteilt die Werte mit geradem Index auf den Realteil und die ungeraden Elemente auf den Imaginärteil und verfährt so wie in der Folie zuvor angegeben:

$$\left.\begin{aligned} \mathbf{x}_g &\Rightarrow \text{Realteil} \\ \mathbf{x}_u &\Rightarrow \text{Imaginärteil} \end{aligned}\right\} \text{Transformation gemäß Folie zuvor}$$

Sodann erhält man unter Anwendung des Cooley-Tukey-Algorithmus:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{\mathbf{x}}_g + \mathbf{d} \circ \tilde{\mathbf{x}}_u \\ \tilde{\mathbf{x}}_g - \mathbf{d} \circ \tilde{\mathbf{x}}_u \end{bmatrix}$$

mit:

$$\mathbf{d} = \left[w^0, w^1, w^2, \ldots, w^{N/2-1} \right]; \quad w = e^{-j2\pi/N}$$

Komplexität zweidimensionaler diskreter unitärer Transformationen

Gegeben: Bildmatrizen **A** der Dimension $N \times N = 2^n \times 2^n$.

1) Berechnung eines Fourierkoeffizenten über Innenprodukt z.B. bei Walsh-Transf.:

$$\widetilde{A_{ij}} = \langle \mathbf{A}, \mathbf{W}_{ij} \rangle \;\Rightarrow\; O(N^2)$$

Für alle Elemente von $\tilde{\mathbf{A}} = \left\{ \widetilde{A_{ij}} \right\} \;\Rightarrow\; O(N^2 \cdot N^2) = O(N^4)$

2) Walsh-Transformation mit separierbarem Kern:

$$\tilde{\mathbf{A}} = \underbrace{\mathbf{W} \cdot \underbrace{\mathbf{A} \cdot \mathbf{W}}_{N^3}}_{2N^3}$$

Damit bereits eine Reduktion auf $O(N^3)$!

3) Walsh-Transformation mit schneller 1D-Transformation:

$$\tilde{\mathbf{A}} = \underbrace{\mathbf{W} \cdot \underbrace{\mathbf{A} \cdot \mathbf{W}}_{N \cdot (N \operatorname{ld} N) = N^2 \cdot \operatorname{ld} N}}_{2N^2 \cdot \operatorname{ld} N}$$

Damit Reduktion auf $O(N^2 \cdot \operatorname{ld} N)$

Eindimensionale Realisierung der zweidimensionalen FFT

$$\mathbf{X} = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \end{bmatrix}$$

$$\dim(\mathbf{X}) = M \times N = 2^m \times 2^n = 2 \times 4$$

2D - Bit - Reversal

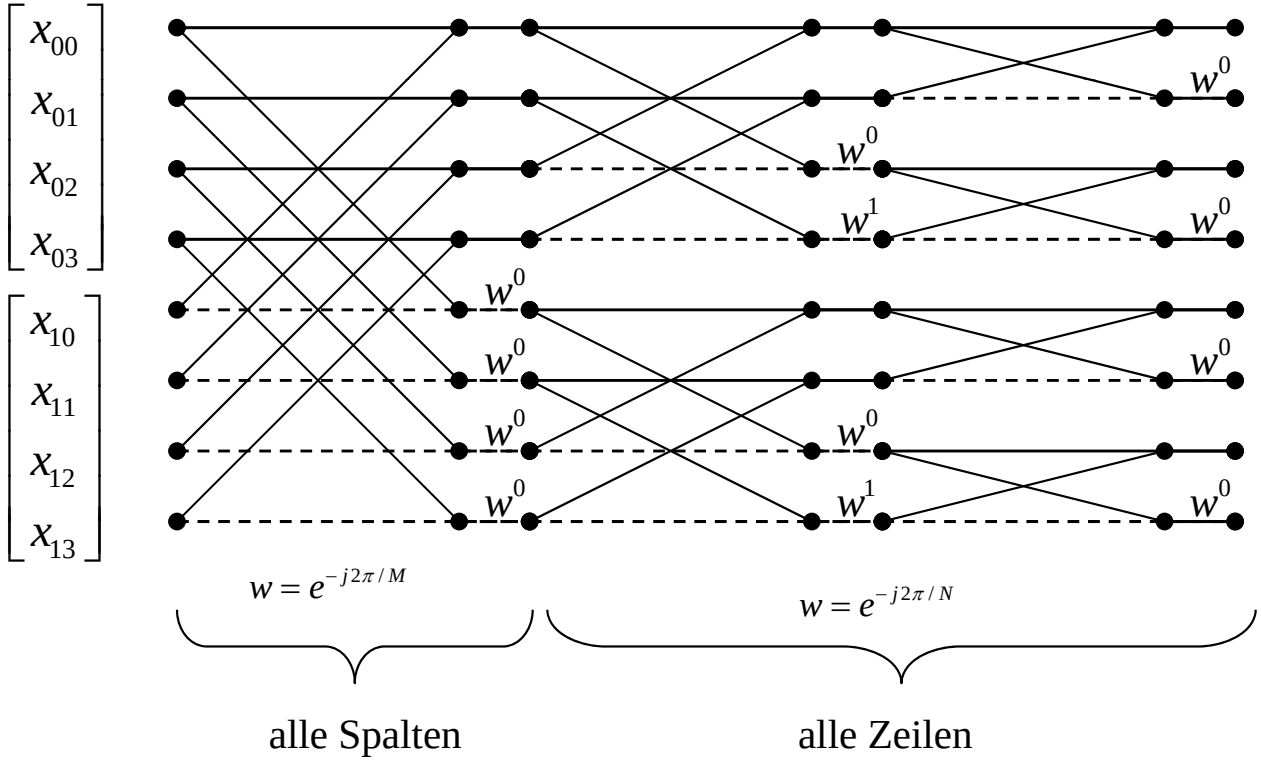
linearer Index : $i = k \cdot N + l$

m Bit | n Bit

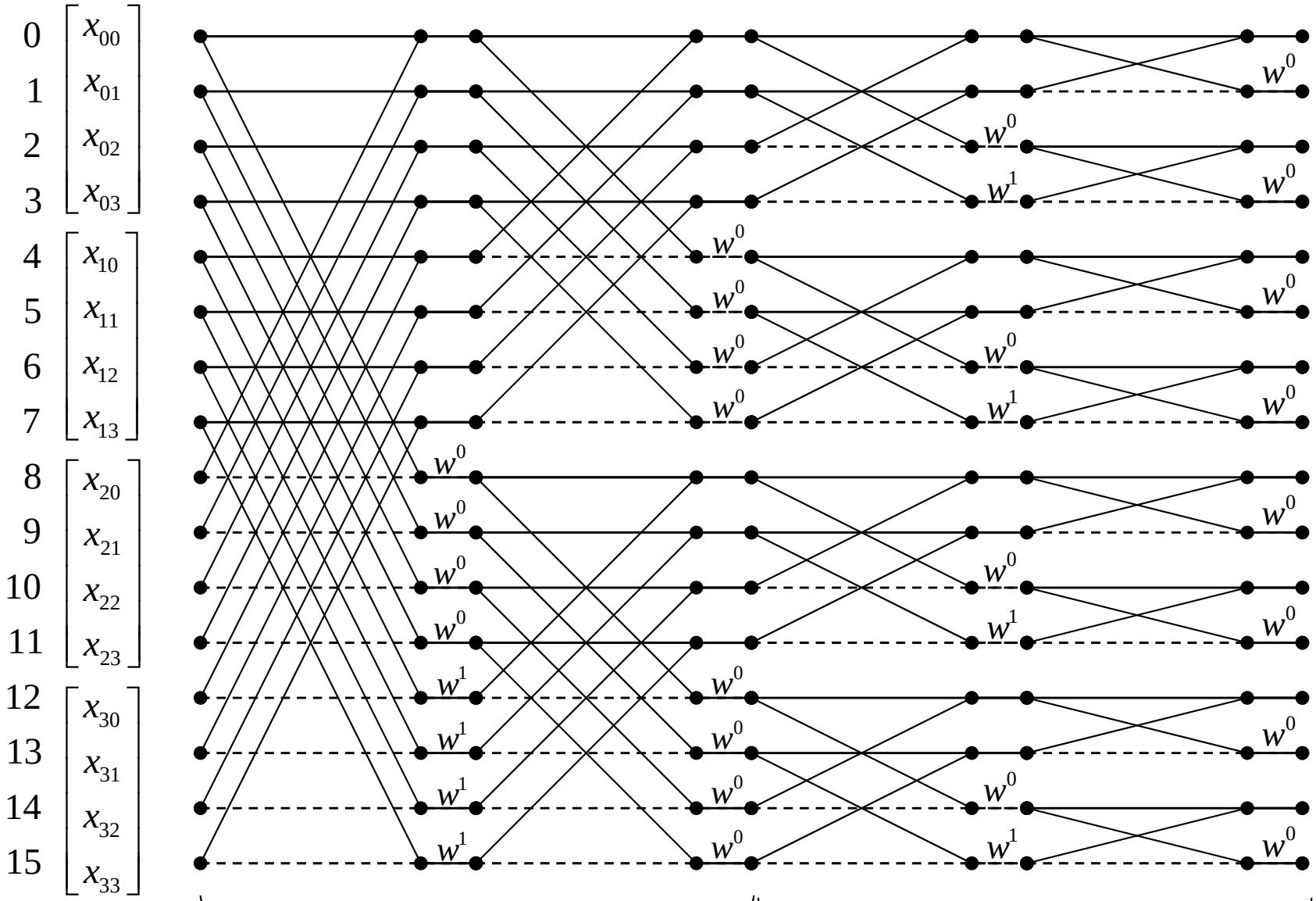
011 | 01101

BR | BR

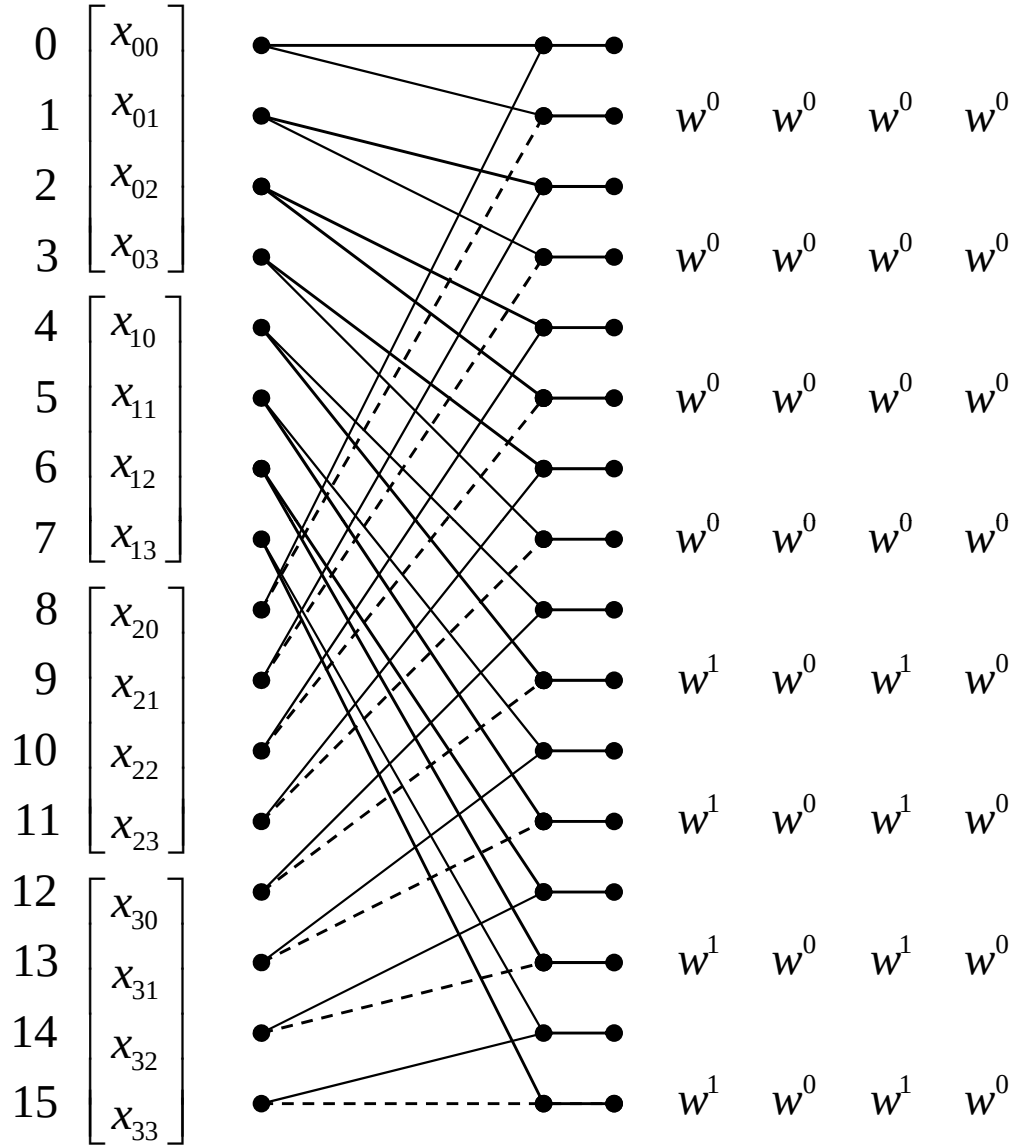
k | l



Eindimensionale Realisierung der zweidimensionalen FFT

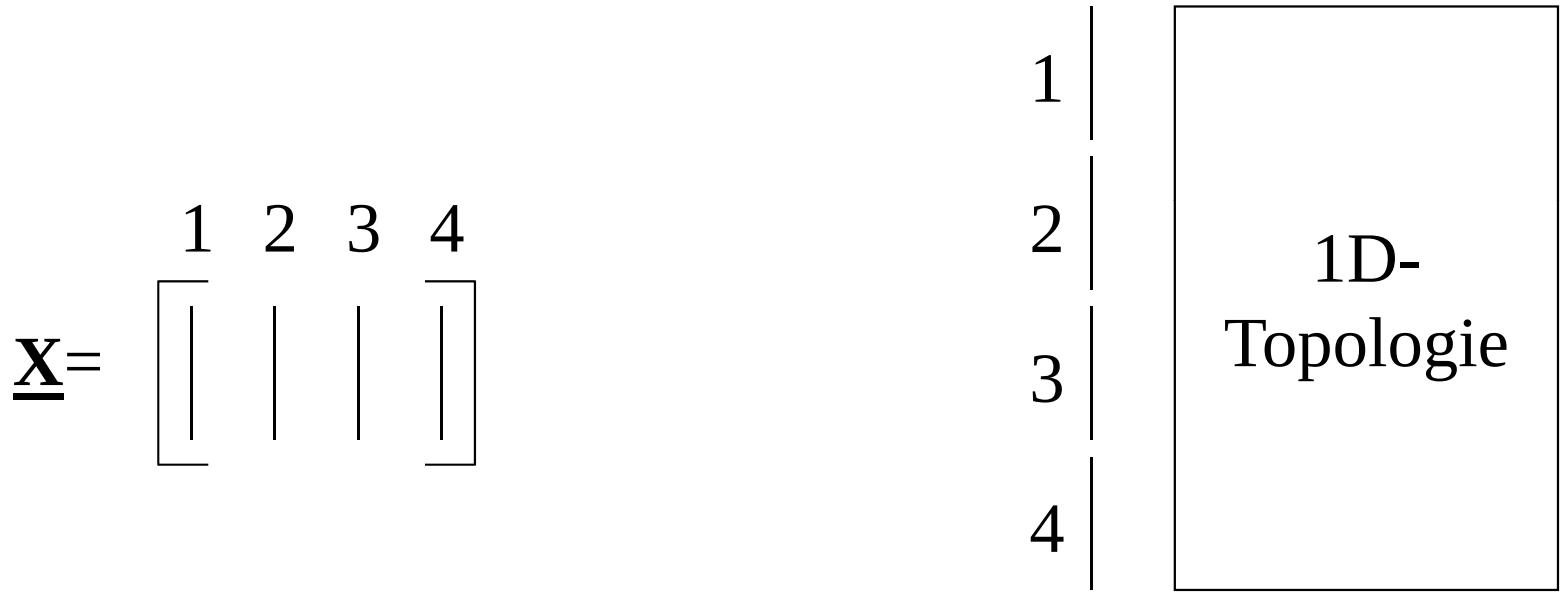


Homogene Struktur zur eindimensionalen Realisierung der zweidimensionalen FFT



$$w = e^{-j2\pi/4} = -j$$

Falls alle Spalten linear gestapelt werden:



Ergibt sich eine Zeilen/Spalten-Transformation
(zuerst Zeilen, dann Spalten)

Zahlenbeispiel

Bildmatrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 12 & 18 & 13 & 2 & 17 & 19 & 20 \\ 16 & 5 & 17 & 4 & 20 & 14 & 7 & 16 \\ 5 & 12 & 11 & 7 & 2 & 12 & 4 & 8 \\ 19 & 3 & 10 & 2 & 8 & 12 & 16 & 8 \end{bmatrix}$$

2D-DFT: (Fourier)

$$\tilde{\mathbf{A}} = \begin{bmatrix} (340,0) & (11.1,-24.6) & (-29,9) & (6.88,-44.6) & (10,0) & (6.88,44.6) & (-29,-9) & (11.1,24.6) \\ (41,21) & (-15.1,-26.5) & (-20,-2) & (15.5,-19) & (-5,-7) & (-24.9,-19.5) & (-32,24) & (8.46,5.02) \\ (-14,0) & (-2.88,18.3) & (-55,1) & (-7.12,14.3) & (-88,0) & (-7.12,-14.3) & (-55,-1) & (-2.88,-18.3) \\ (41,-21) & (8.46,-5.02) & (-32,-24) & (-24.9,19.5) & (-5,7) & (15.5,19) & (-20,2) & (-15.1,26.5) \end{bmatrix}$$

$$= \begin{bmatrix} 340 & 11.1 & -29 & 6.88 & 10 & 6.88 & -29 & 11.1 \\ 41 & -15.1 & -20 & 15.5 & -5 & -24.9 & -32 & 8.46 \\ -14 & -2.88 & -55 & -7.12 & -88 & -7.12 & -55 & -2.88 \\ 41 & 8.46 & -32 & -24.9 & -5 & 15.5 & -20 & -15.1 \end{bmatrix} + j \begin{bmatrix} 0 & -24.6 & 9 & -44.6 & 0 & 44.6 & -9 & 24.6 \\ 21 & -26.5 & -2 & -19 & -7 & -19.5 & 24 & 5.02 \\ 0 & 18.3 & 1 & 14.3 & 0 & -14.3 & -1 & -18.3 \\ -21 & -5.02 & -24 & 19.5 & 7 & 19 & 2 & 26.5 \end{bmatrix}$$

2D-DWT: (Walsh)

$$\tilde{\mathbf{A}} = \begin{bmatrix} 340 & -30 & 2 & -20 & -38 & -4 & 68 & 10 \\ 62 & -28 & -20 & -34 & 4 & -34 & 6 & -12 \\ 20 & -18 & 30 & -44 & -30 & 40 & -8 & 2 \\ -14 & 20 & -4 & -54 & -56 & -10 & -26 & -88 \end{bmatrix}$$

Zusätzliche Anmerkungen zu FFT-Algorithmen

- Unter zusätzlicher Ausnutzung von Symmetrien der Exponentialfunktion ergeben sich schnelle Basis-4 und Basis-8 Algorithmen, welche weniger Multiplikationen auf Kosten zusätzlicher Additionen enthalten.
- Es existieren auch Prime-FFT-Algorithmen für Primzahldimensionen (keine Faktorisierung!), welche auf eine schnelle Faltung zurückgeführt werden können.
- Es existieren auch Mixed-Radix-Algorithmen, basierend auf einer gemischten Faktorisierung: $N=N_1 \cdot N_2 \cdot N_3$
- Der Winograd-Algorithmus garantiert das absolute Minimum an notwendigen Multiplikationen auf Kosten von Additionen. Er ist jedoch sehr unregelmäßig und seine Struktur ist von der Dimension abhängig.

Weitere Anwendungen der Faktorisierung

- Schnelles paralleles Sortieren aufbauend auf Bitonen Sortierer (~(ld N)^2 Zyklen)
- Dynamische Programmierung (DBV-II)
- Parallele Berechnung von Skalarprodukten
- Lageinvariante Mustererkennung (Kursvorlesung)
- Schnelle Polynomberechnung