

$$c : \begin{pmatrix} c_{gerade} \\ c_{ungerade} \end{pmatrix} \Rightarrow \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

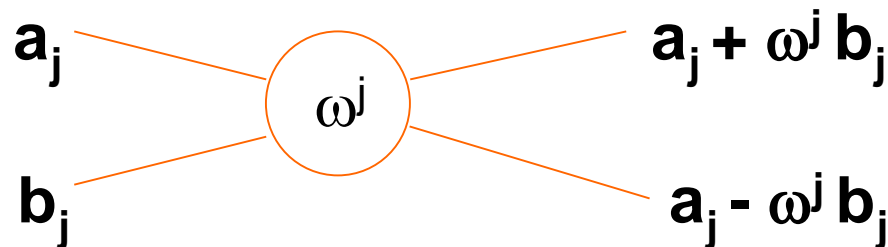
$$v = \begin{pmatrix} v_{oben} \\ v_{unten} \end{pmatrix} = \begin{pmatrix} v_1 + \omega^j v_2 \\ v_1 - \omega^j v_2 \end{pmatrix}$$

Anstelle einer IDFT Länge n sind also nun vier IDFT's der Länge $n/4$ zu berechnen.

Allgemein dazu notwendig: $n=2^p$ ist Zweierpotenz;
 Dann kann dieses Verfahren rekursiv durchgeführt werden,
 bis man *eine IDFT der Länge n* ausgedrückt hat durch
 n IDFT's der Länge 1.

Grundidee:

Rekursive Aufteilung des aktuellen Vektors in geraden und ungeraden Anteil; falls die Ergebnisse der beiden IDFT's halber Länge vorliegen, werden die beiden Teile zum Ergebnis doppelter Länge kombiniert mittels

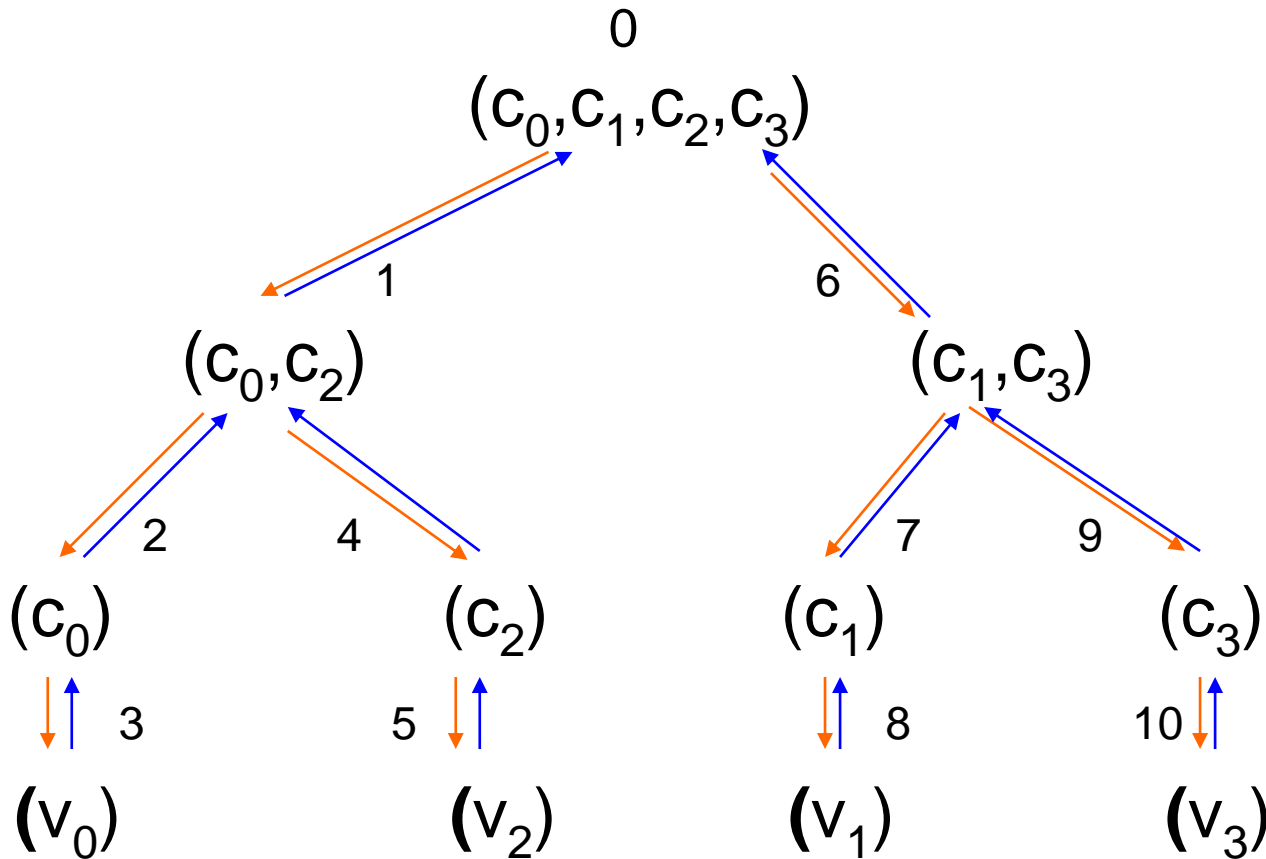


```

FUNCTION(v0,...,vn-1) = IDFT(c0,...,cn-1,n)
IF n==1
    v0 = c0 ;
ELSE
    m=n/2 ;
    (g0,...,gm-1) = IDFT(c0, c2,..., cn-2,m) ;
    (u0,...,um-1) = IDFT(c1, c3,..., cn-1,m) ;
    ω = exp(2iπ/n) ;
    FOR j=0:m-1
        vj = gj + ωj uj ;
        vj+m = gj - ωj uj ;
    END
END
END
    
```

5.2.2 Analyse des rekursiven Programms:

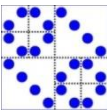
Aufrufabfolge als binärer Baum (hier $n=4$):



Ziffern 0 bis 10 bezeichnen die unterschiedlichen rekursiven Aufrufe des Programms IDFT mit neuen Parametern

(**roter Pfeil: Aufruf**, **blauer Pfeil: Werterückgabe und Ende**).

Insgesamt $3n-1$ Programmaufrufe IDFT.

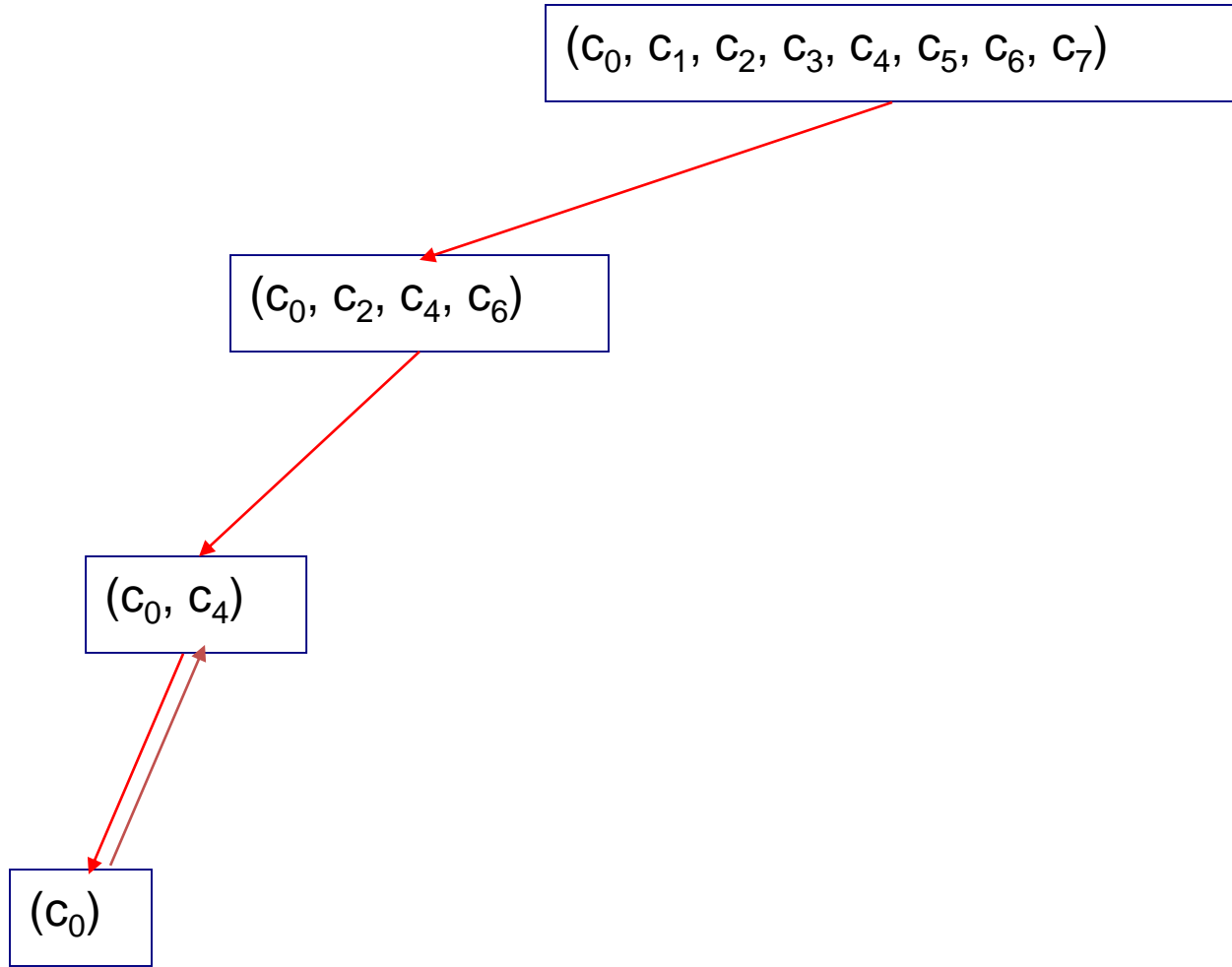


$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$

(c_0, c_2, c_4, c_6)

(c_0, c_4)

(c_0)



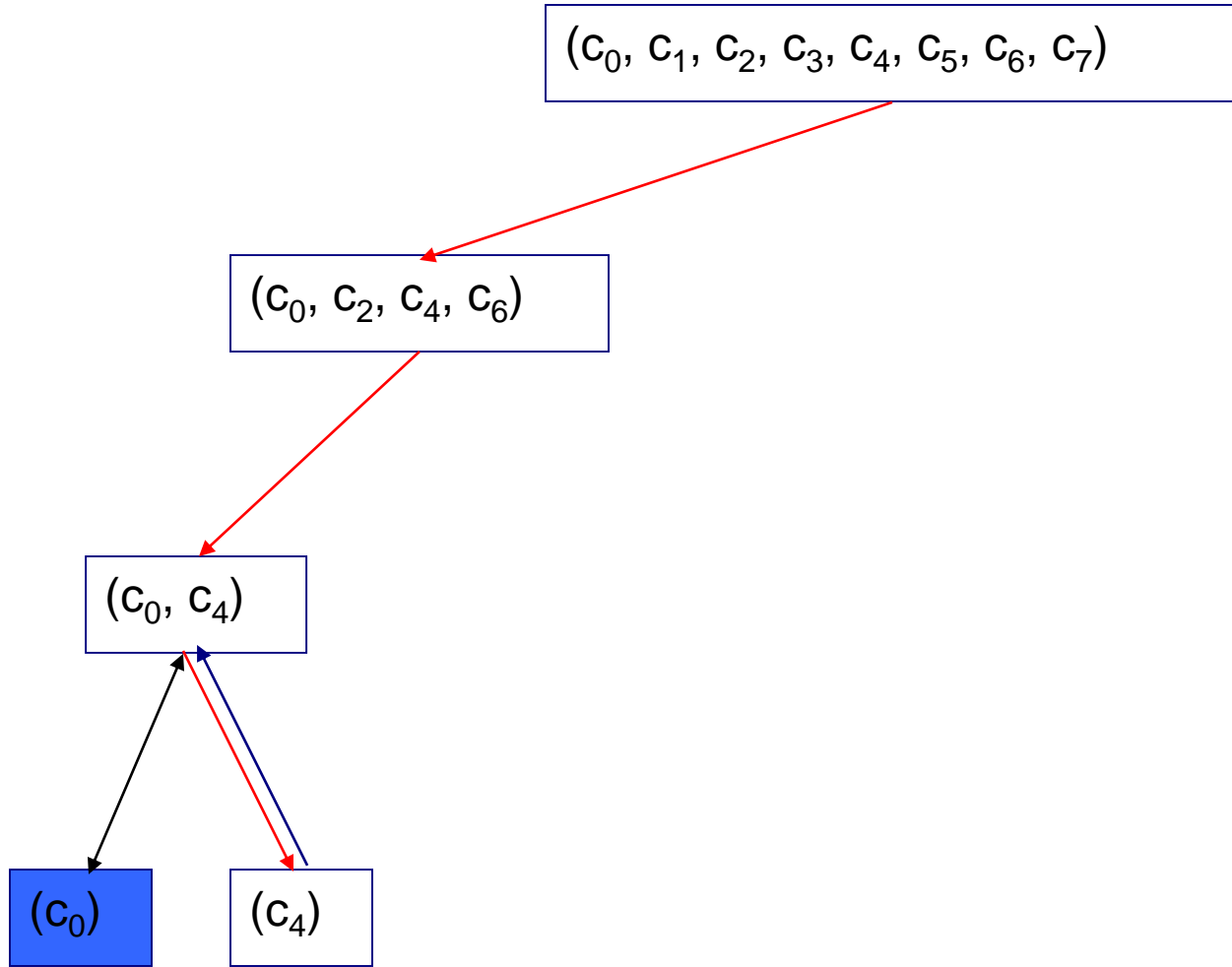
$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$

(c_0, c_2, c_4, c_6)

(c_0, c_4)

(c_0)

(c_4)



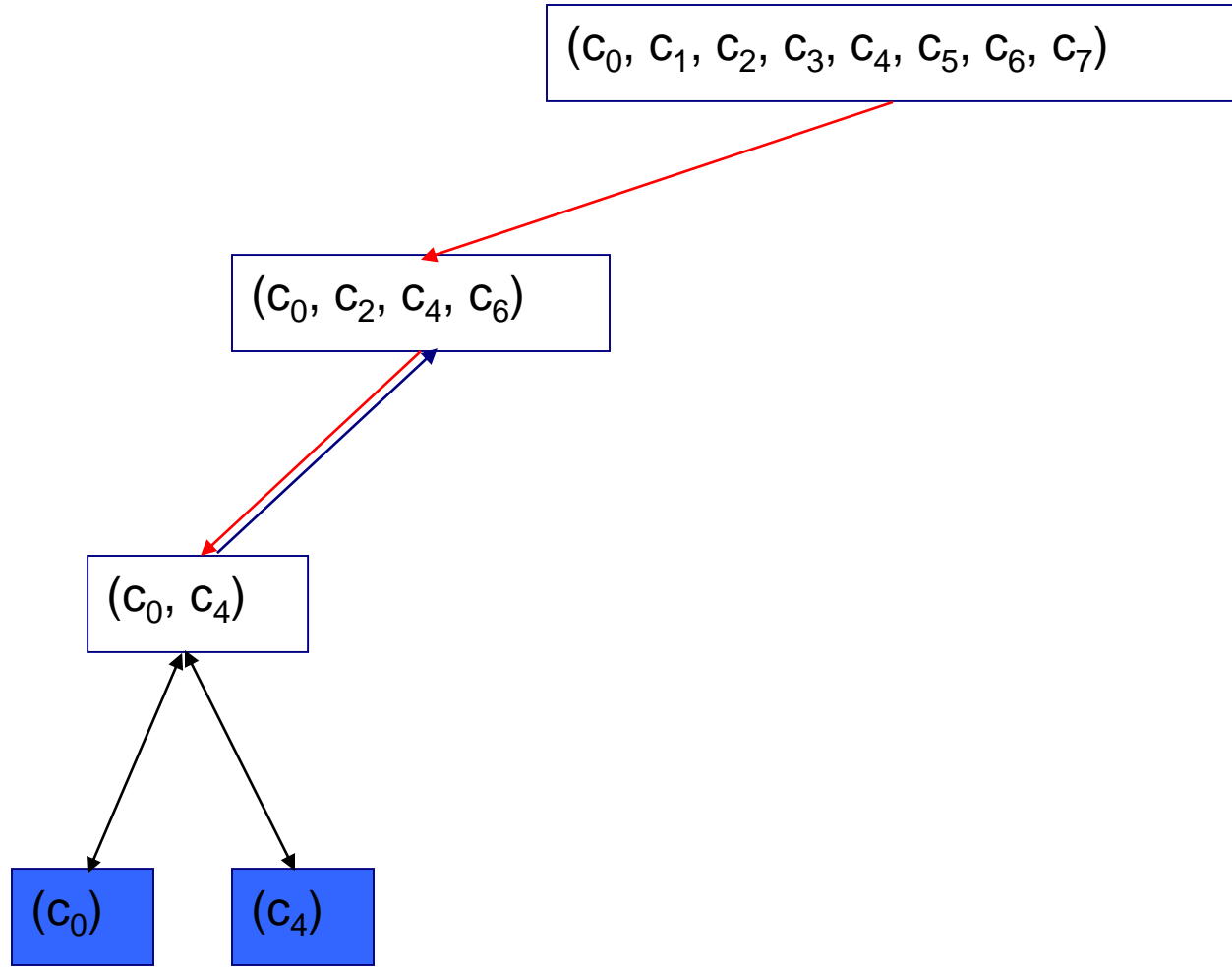
$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$

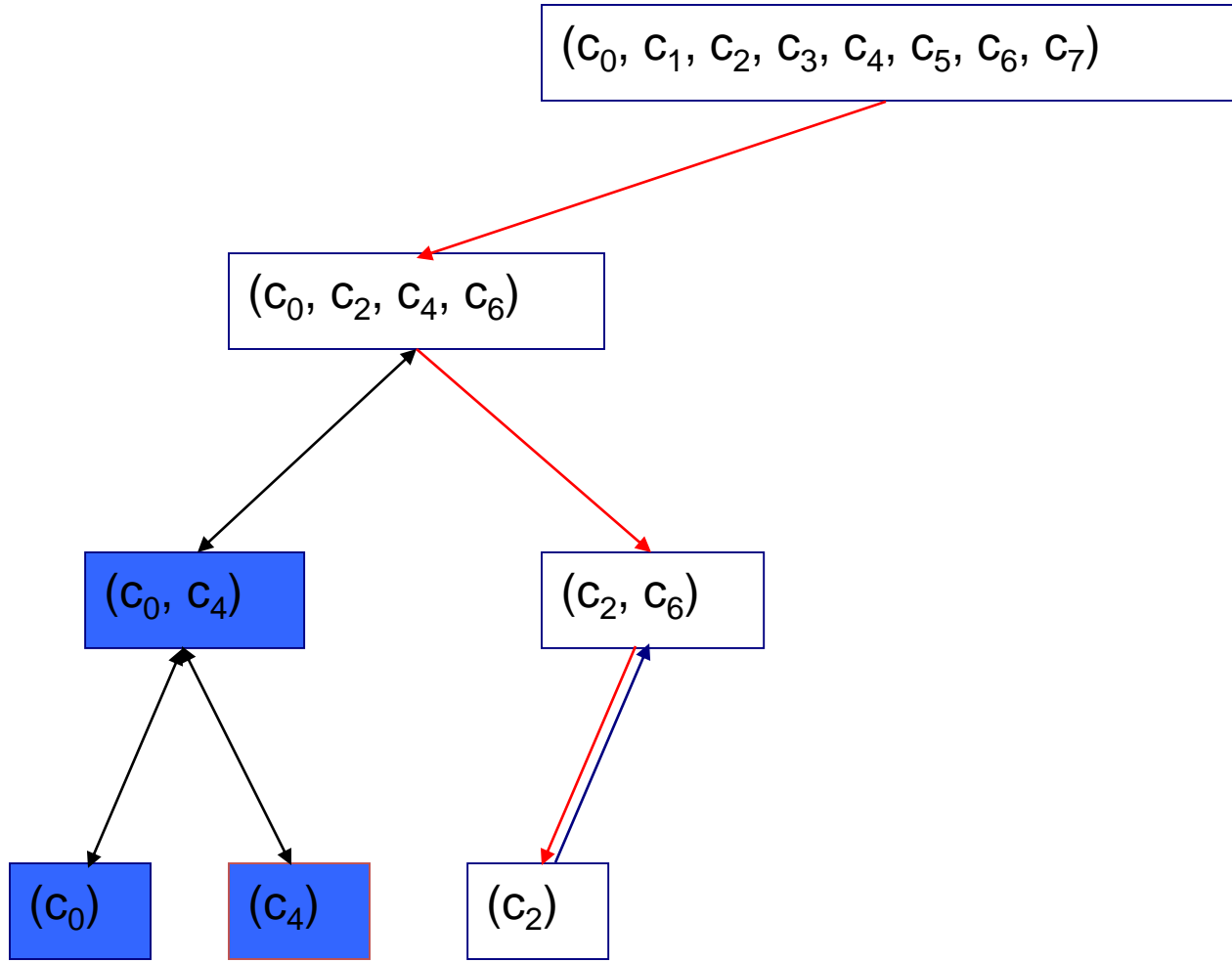
(c_0, c_2, c_4, c_6)

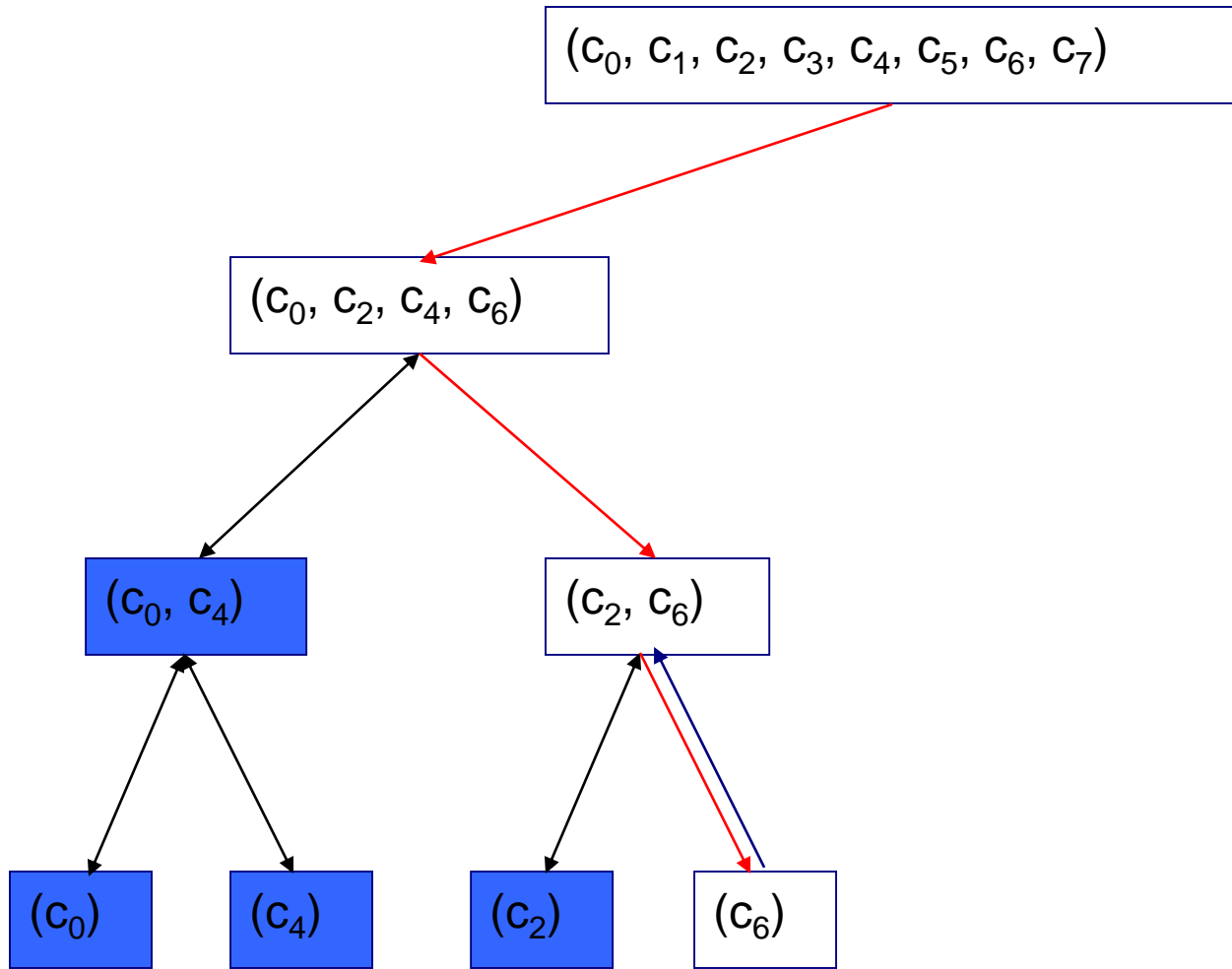
(c_0, c_4)

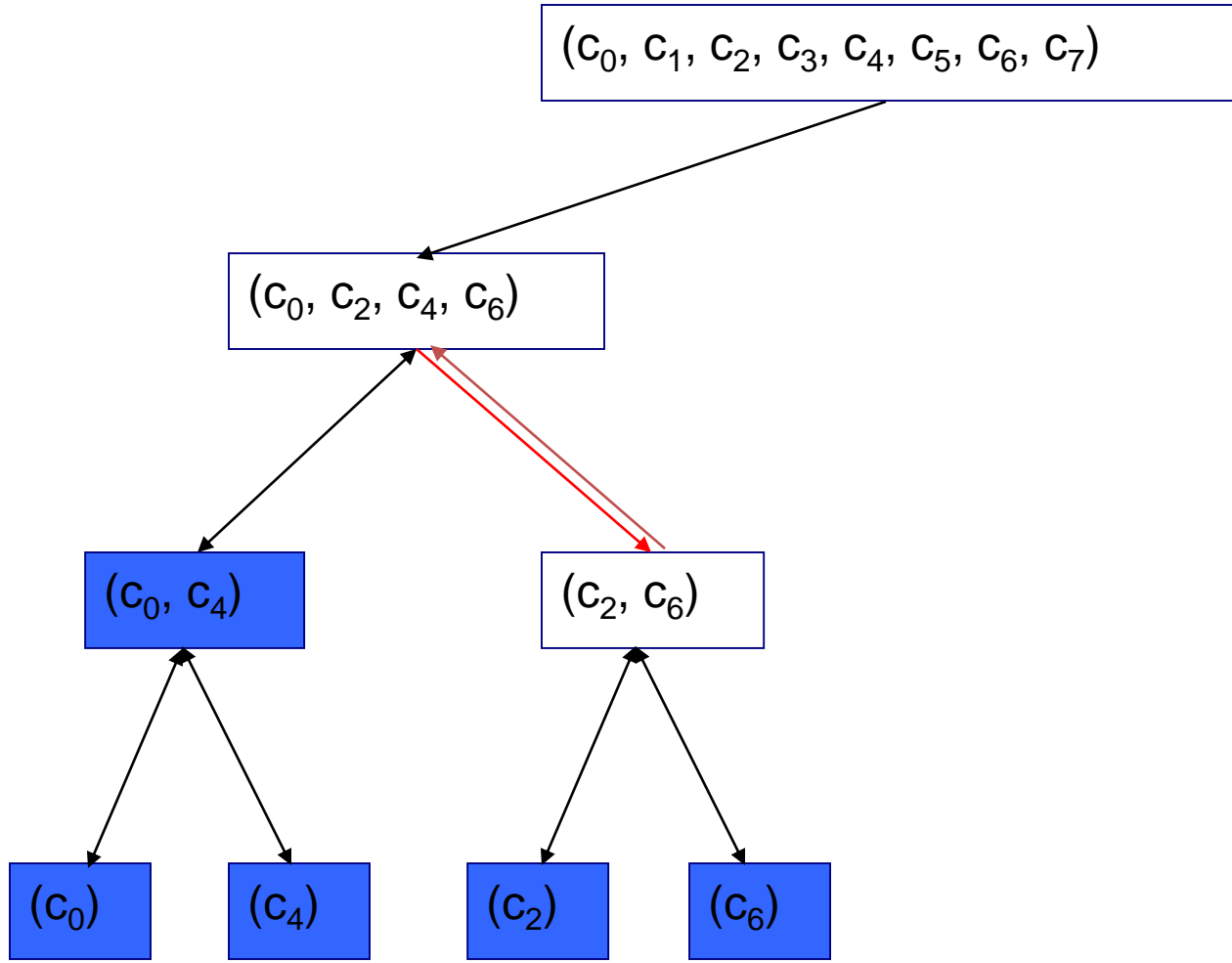
(c_0)

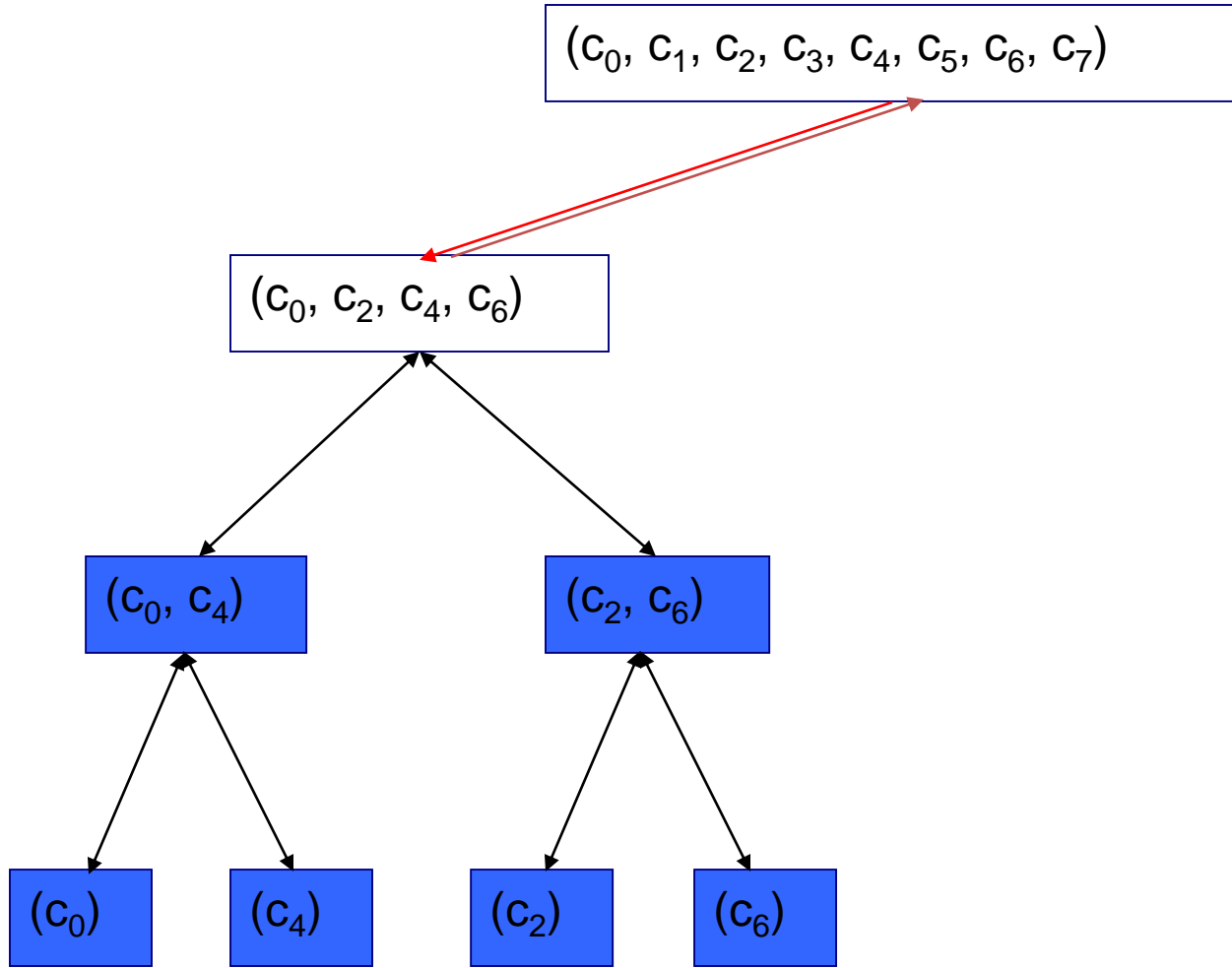
(c_4)

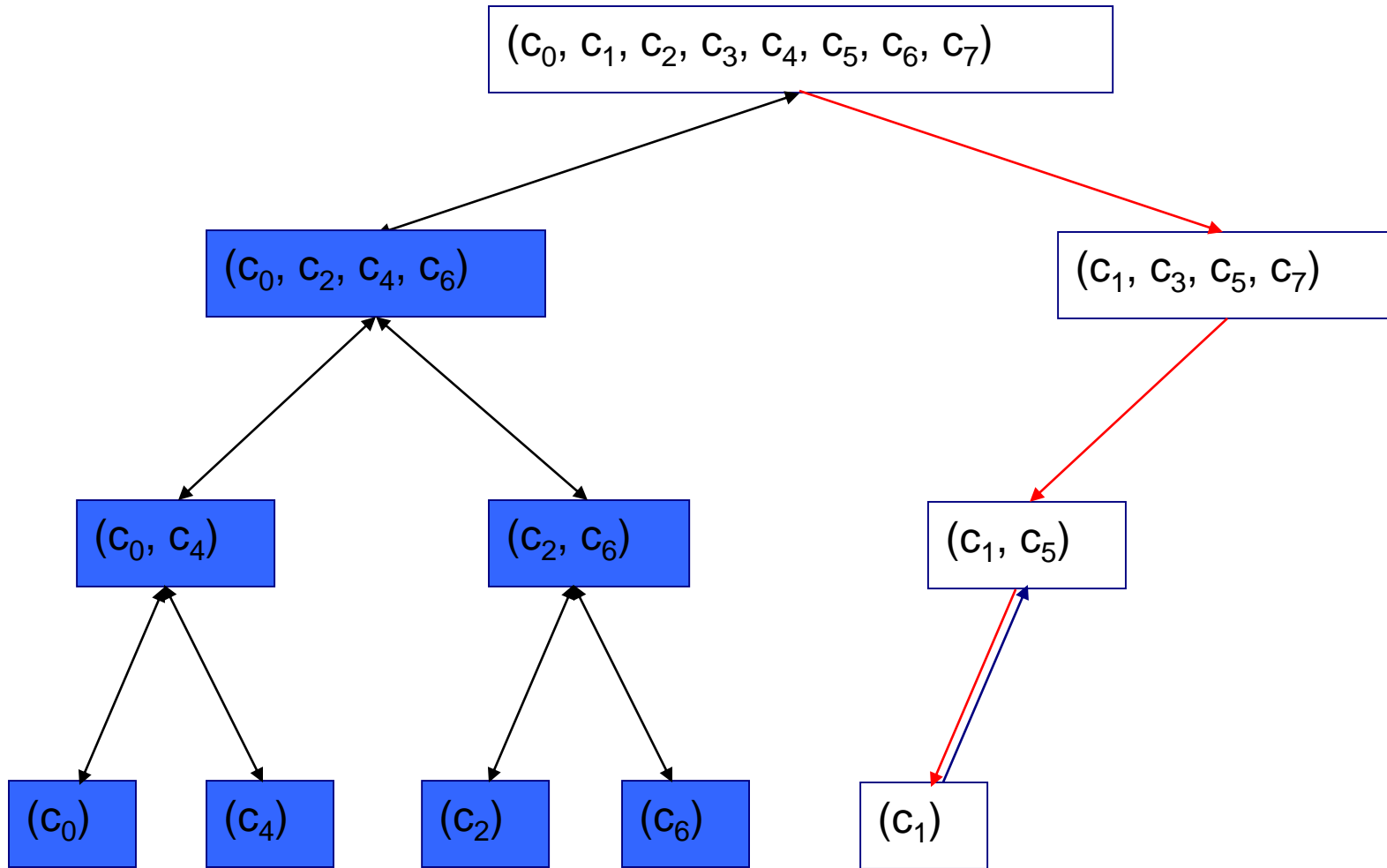


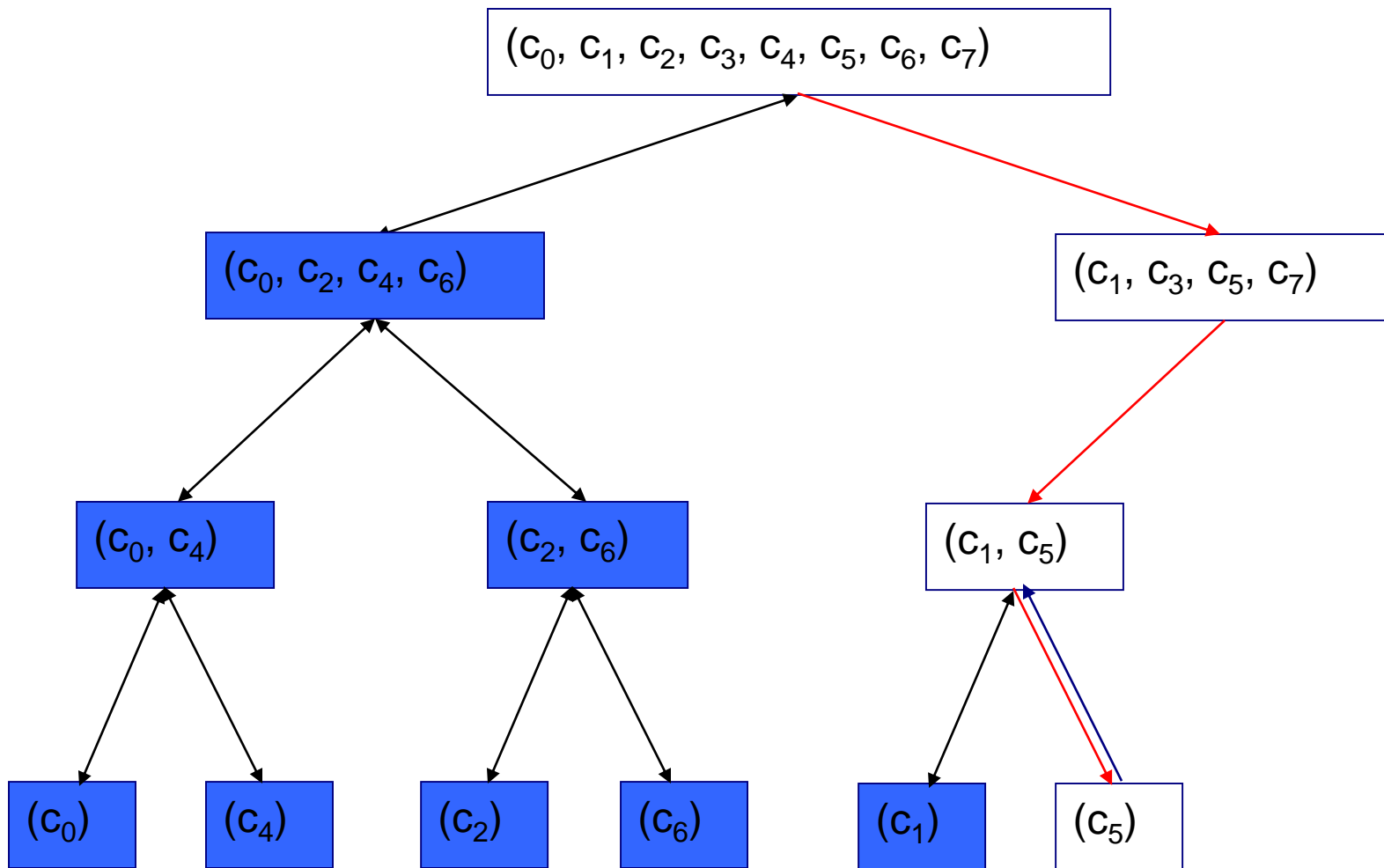


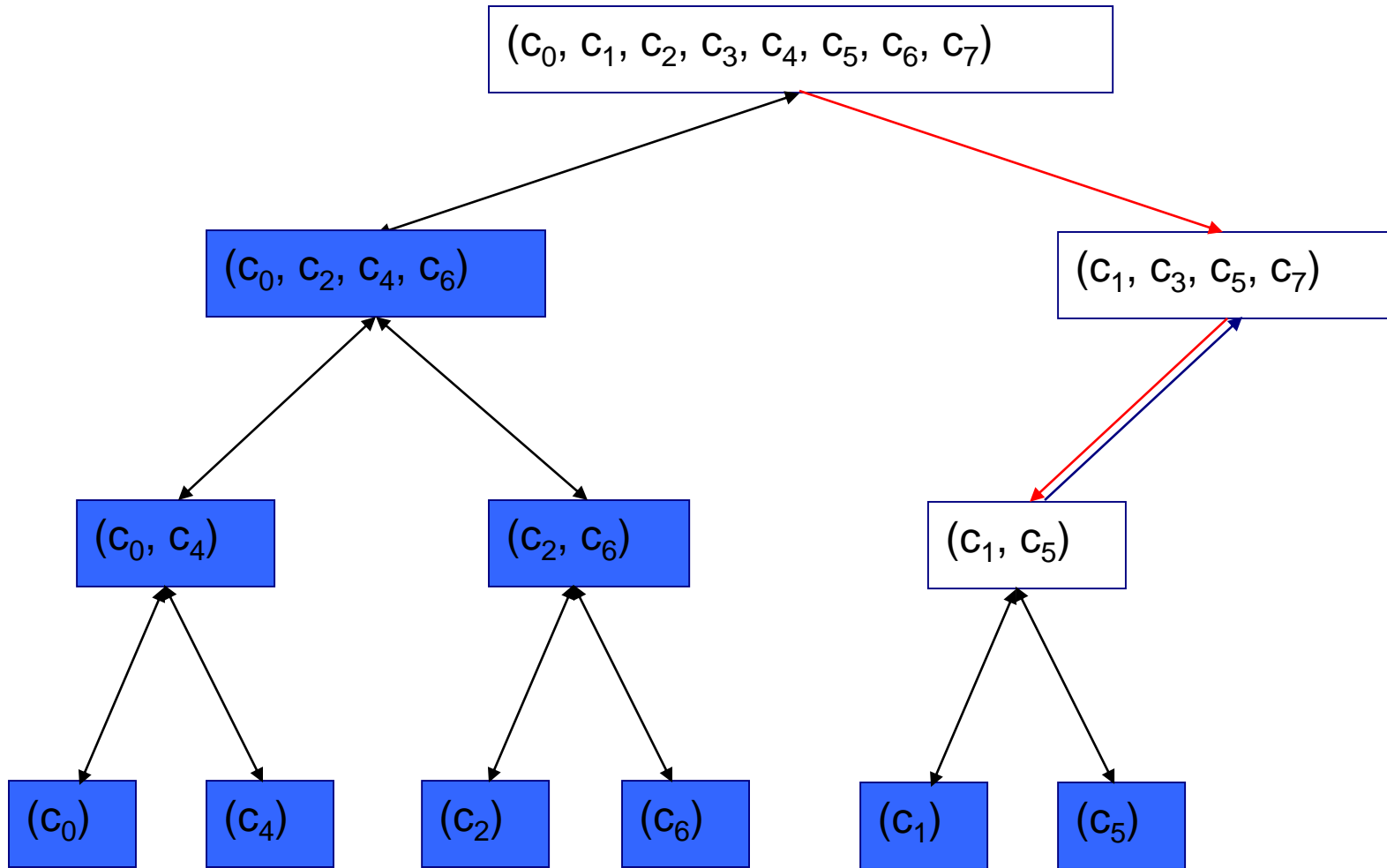


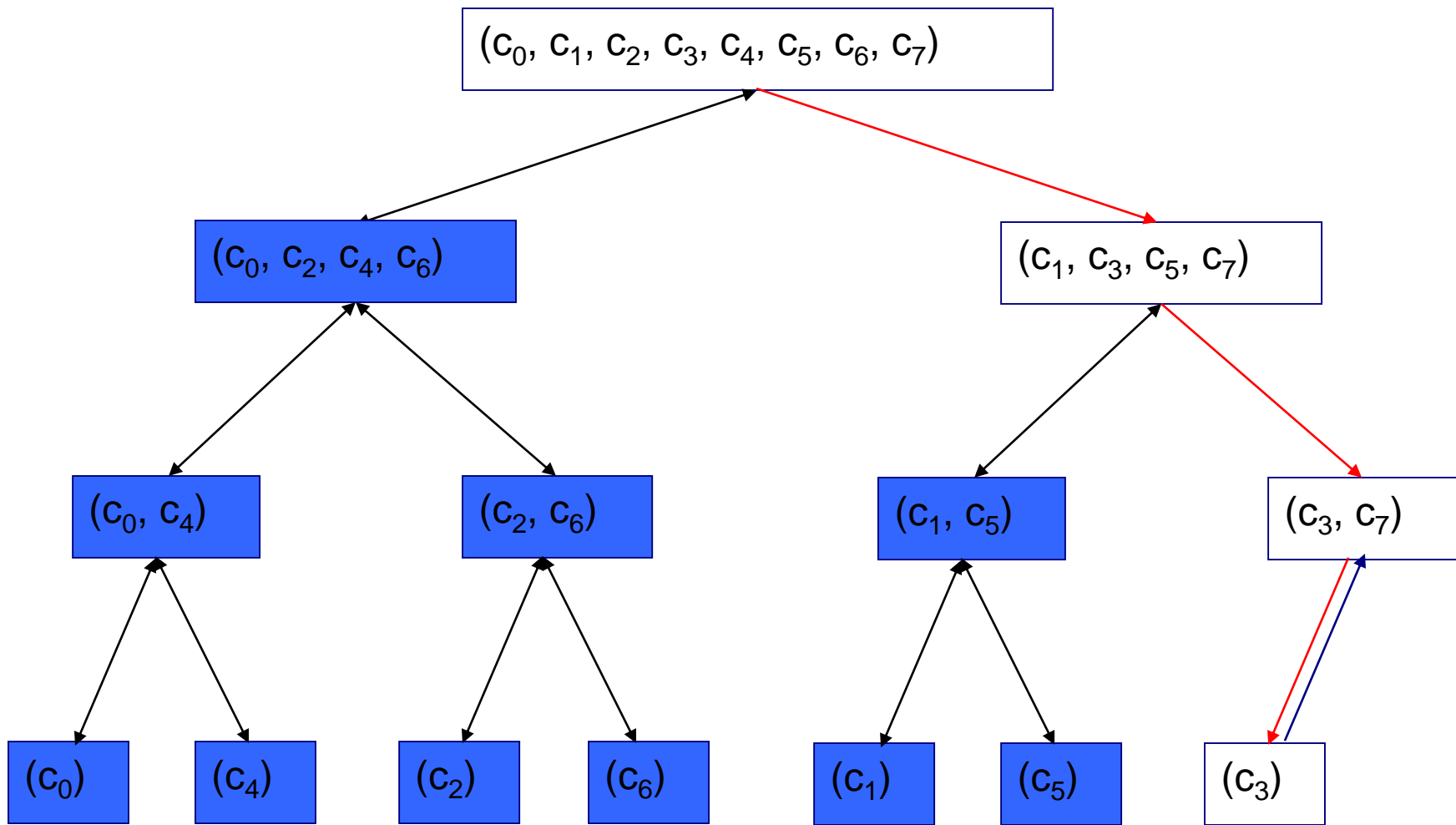


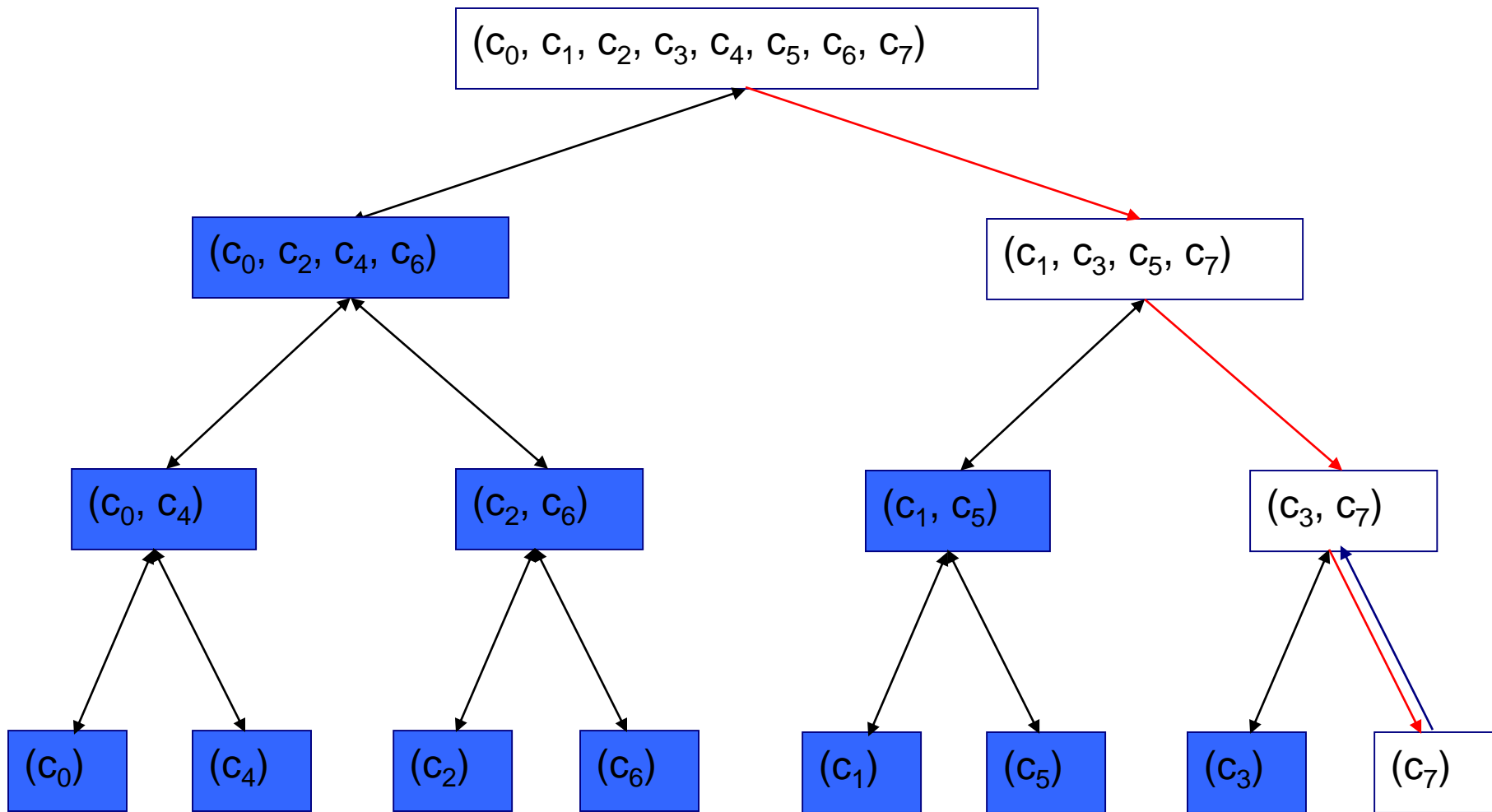


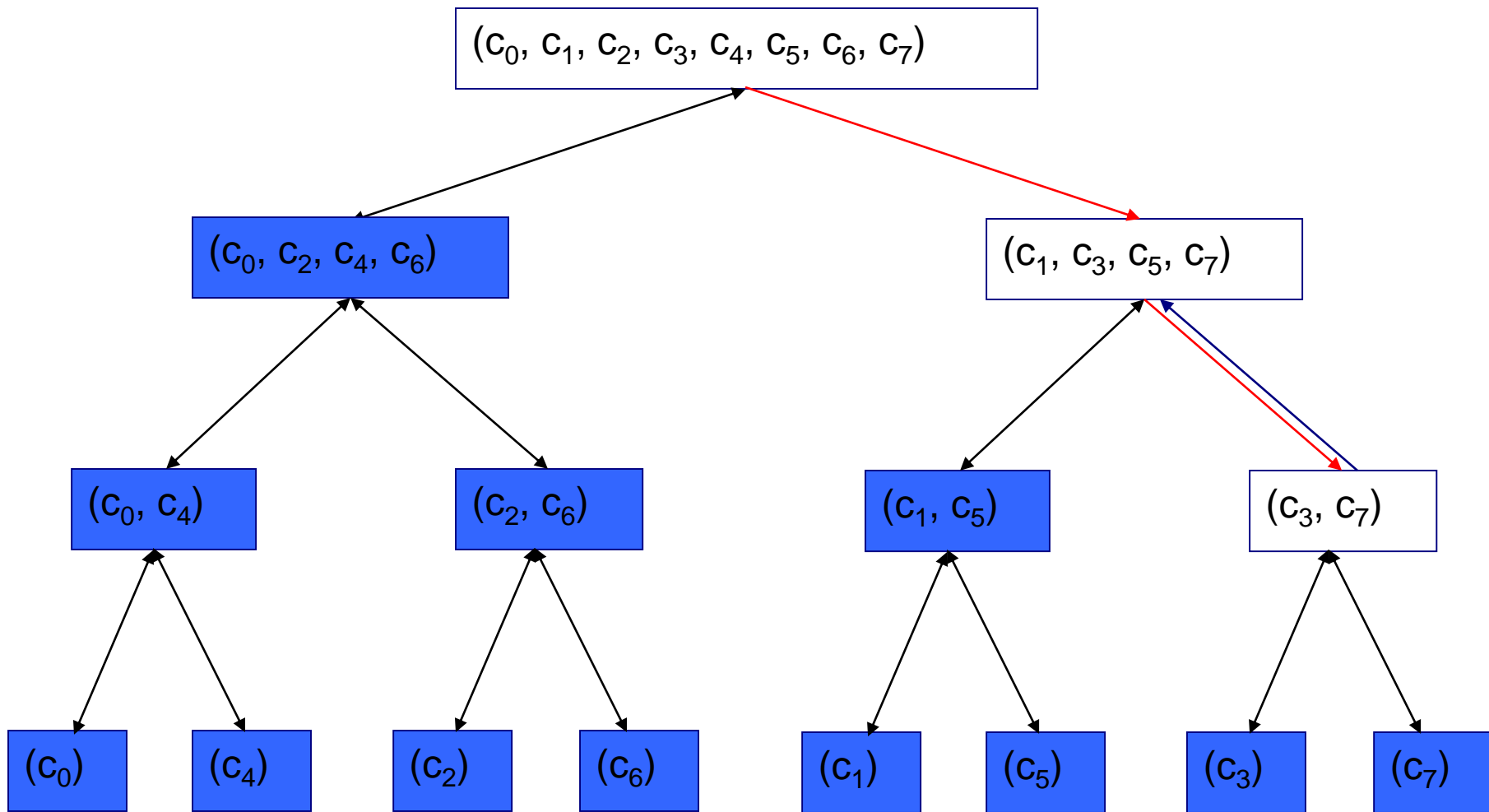


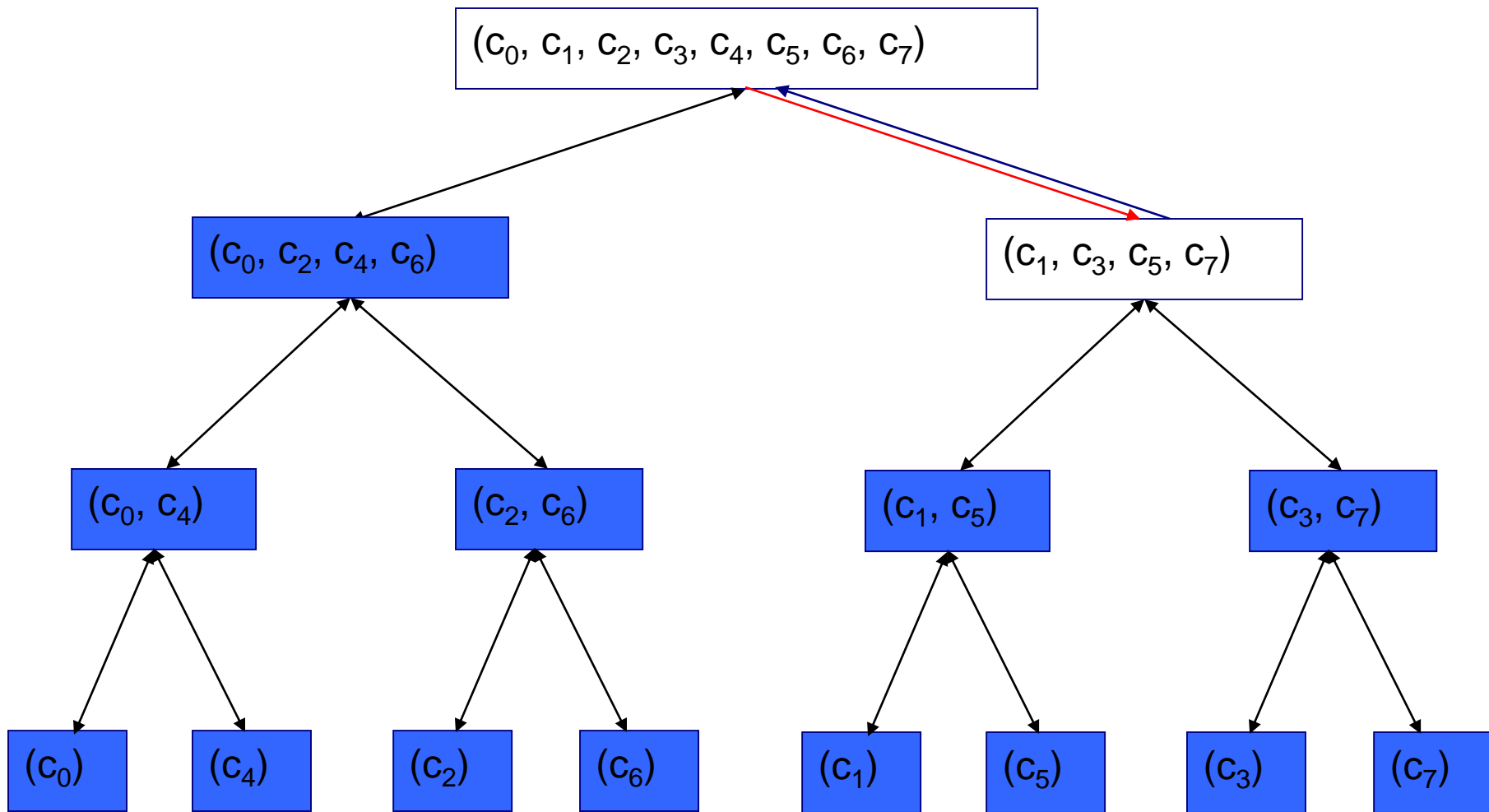


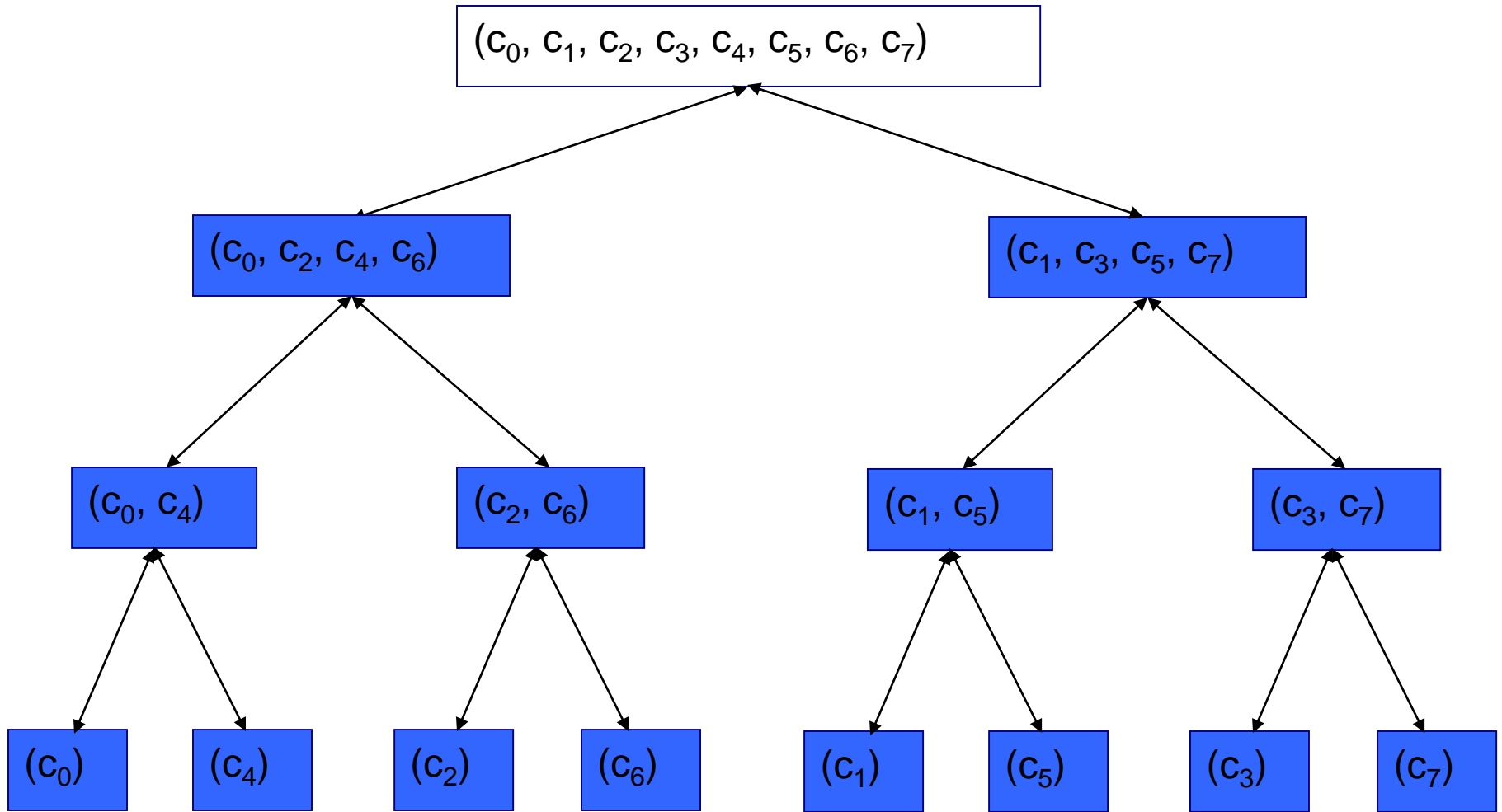


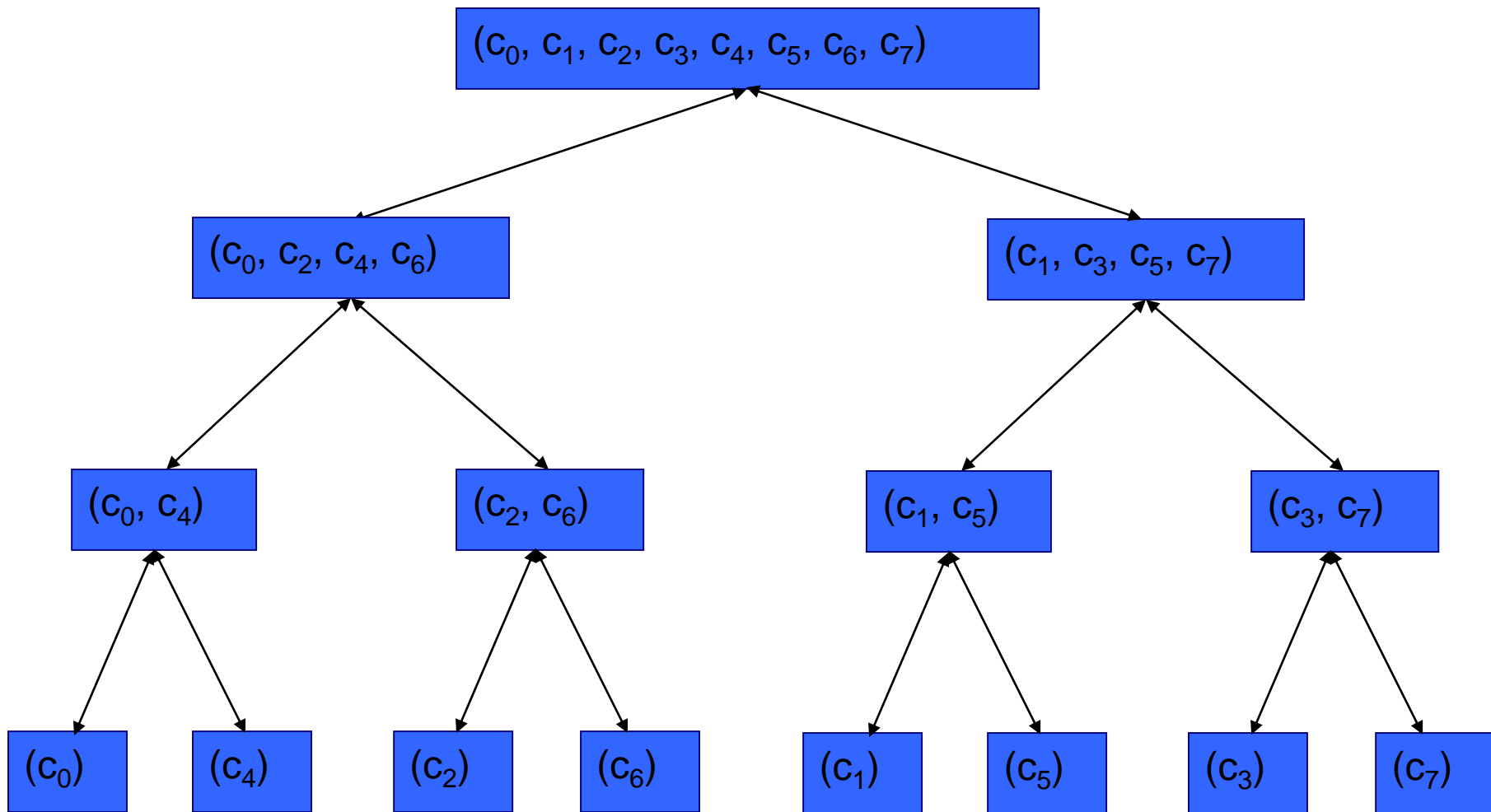


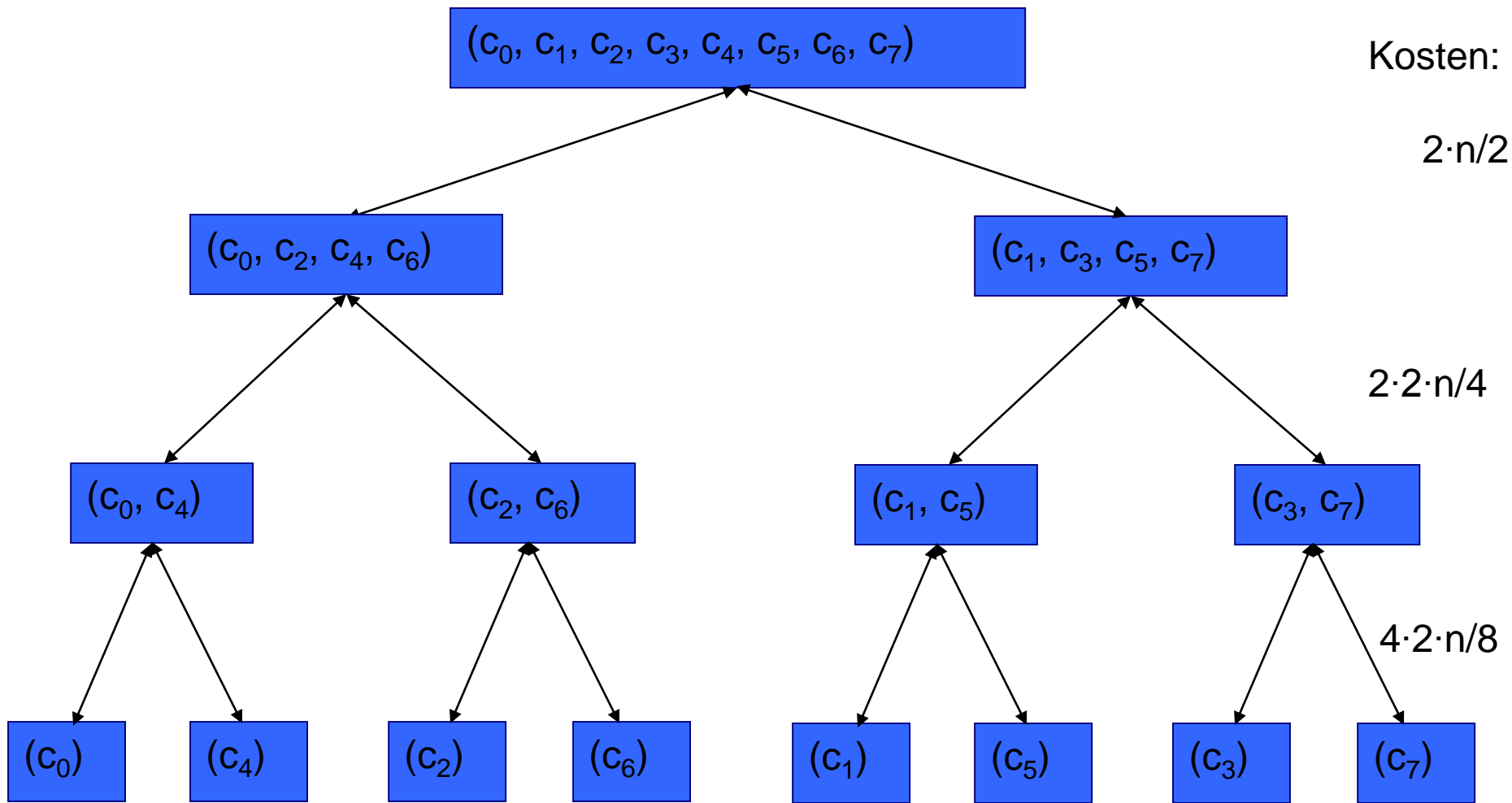














Die Aufrufe auf der untersten Ebene sind trivial, da mit Vektoren der Länge 1 als Eingabe.

Kosten für $n=2^p$ ($p=\log(n)$):

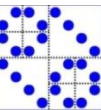
Auf jeder Ebene sind jeweils 2^k Vektoren der Länge 2^{n-k} zu bearbeiten (umsortieren und zusammenkombinieren):

	<i>Kosten</i>
Ebene 1: 2 Vektoren der Länge $n/2$	$O(n)$
Ebene 2: 4 Vektoren der Länge $n/4$	$O(n)$
....	
Ebene p : n Vektoren der Länge 1	$O(n)$

Gesamtkosten daher:

$$\mathbf{O(p \cdot n) = O(p \cdot 2^p) = O(\log_2(n) \cdot n)}$$

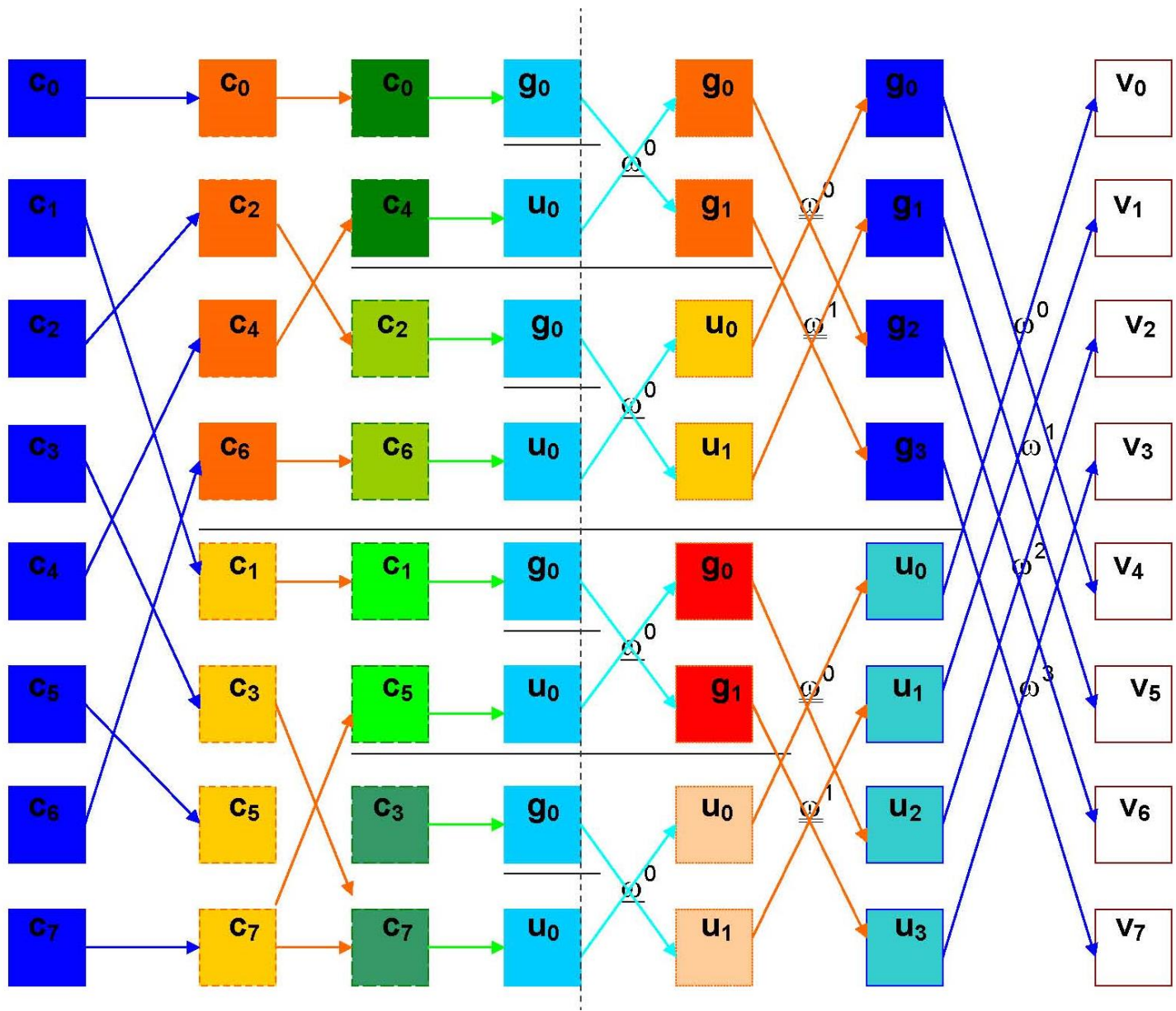
anstatt $O(n^2)$, wie sonst für Matrix-Vektor-Multiplikation.



Rekursives Programm zwar in $O(pn)$ sehr schnell,
hat aber großen Overhead für Aufrufe:

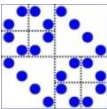
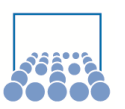
Es muss dynamisch laufend neuer Speicherplatz
angelegt werden.

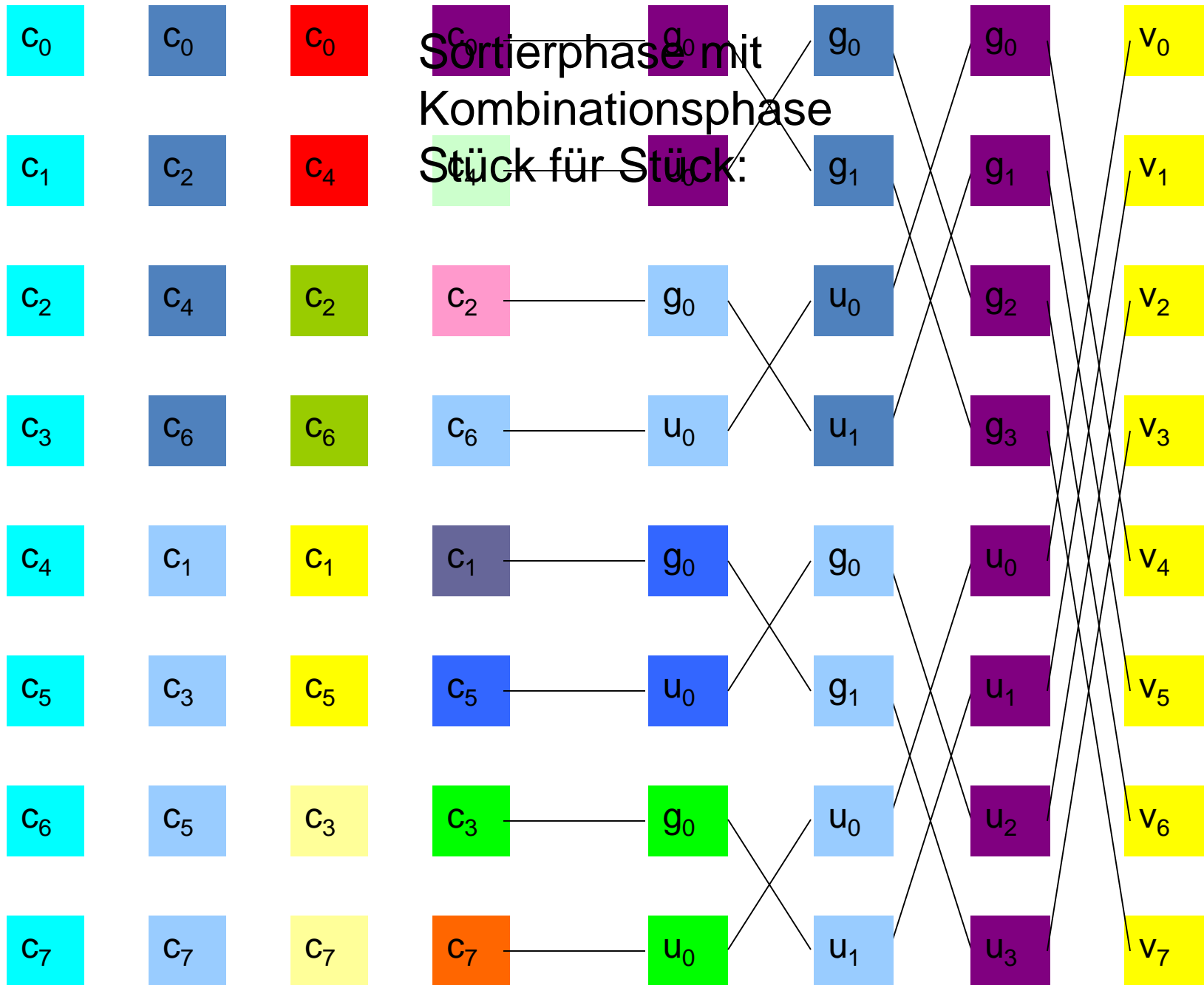
Ziel ist es im Folgenden, die Rekursion besser durch
einfache For-Loops darzustellen.



Sortierphase

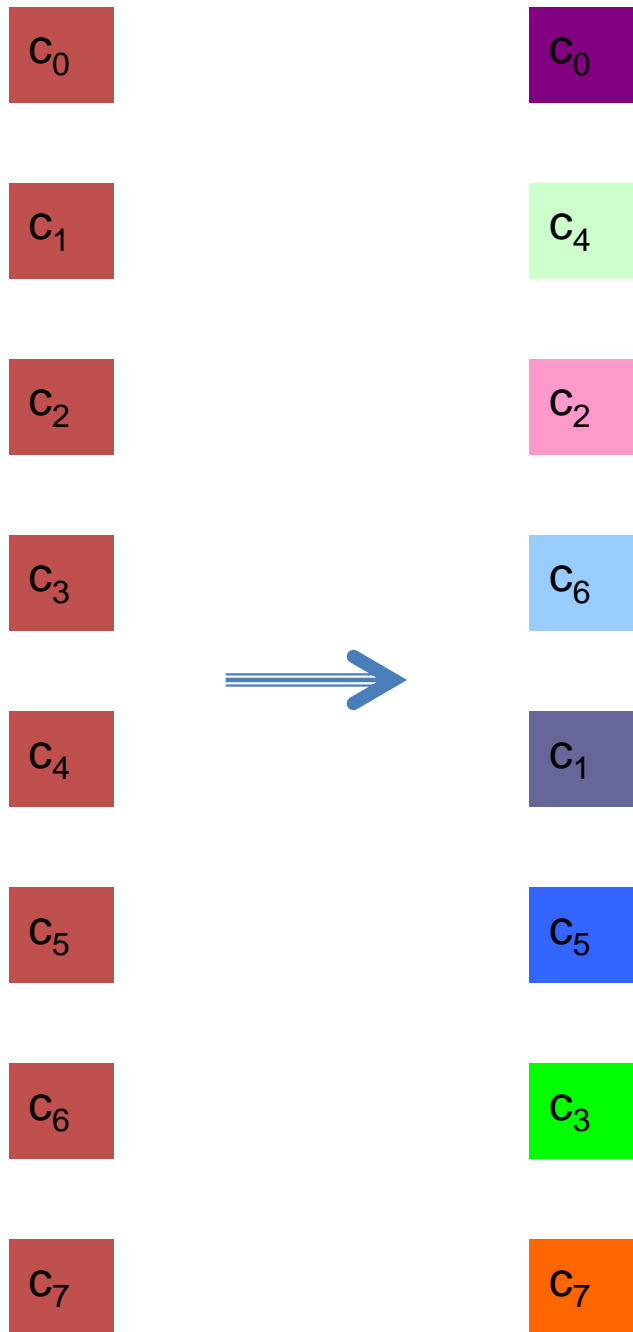
Kombinationsphase



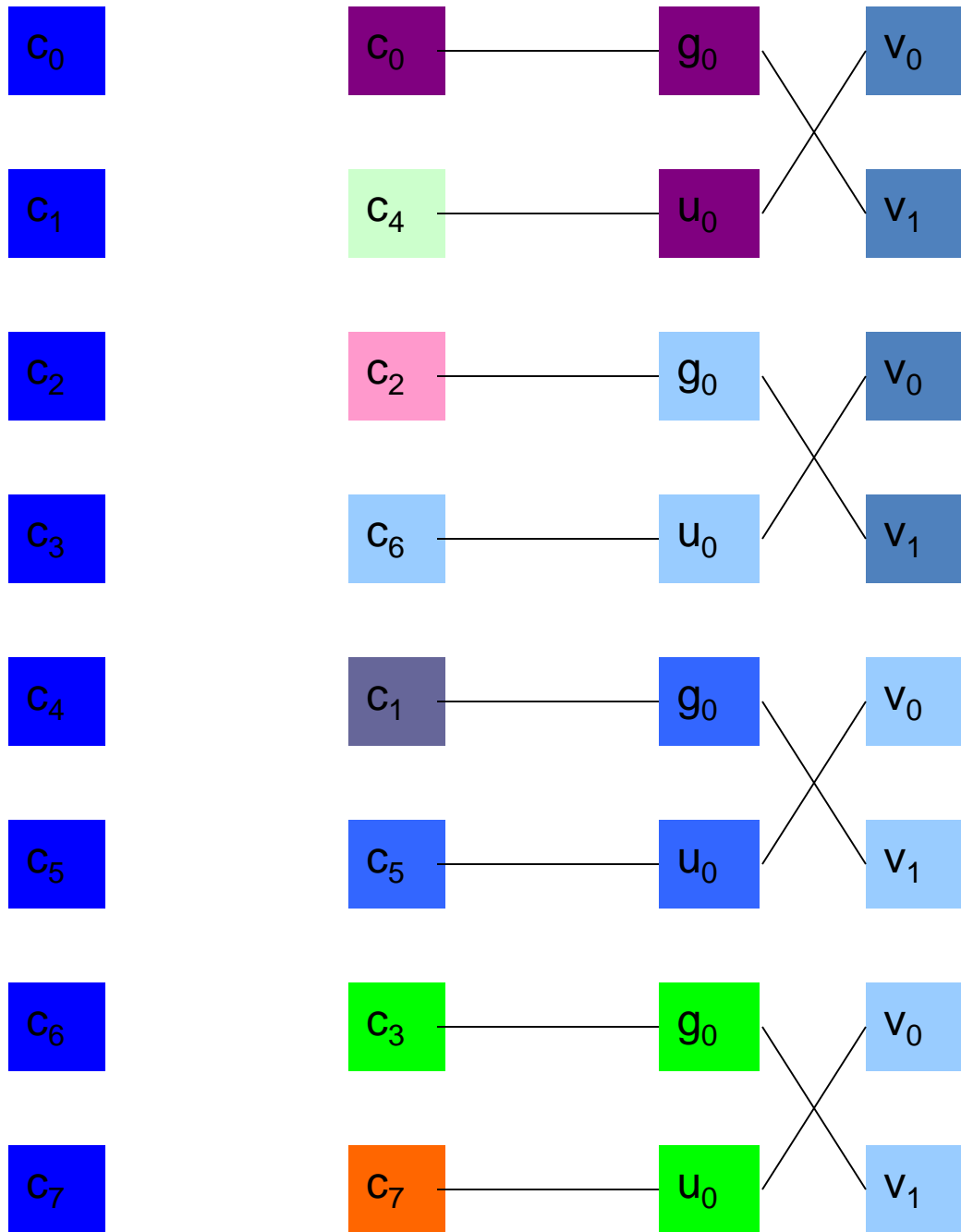


C ₀	C ₀	C ₀	C ₀
C ₁	C ₂	C ₄	C ₄
C ₂	C ₄	C ₂	C ₂
C ₃	C ₆	C ₆	C ₆
C ₄	C ₁	C ₁	C ₁
C ₅	C ₃	C ₅	C ₅
C ₆	C ₅	C ₃	C ₃
C ₇	C ₇	C ₇	C ₇

Sortierphase =
Aufrufphase:

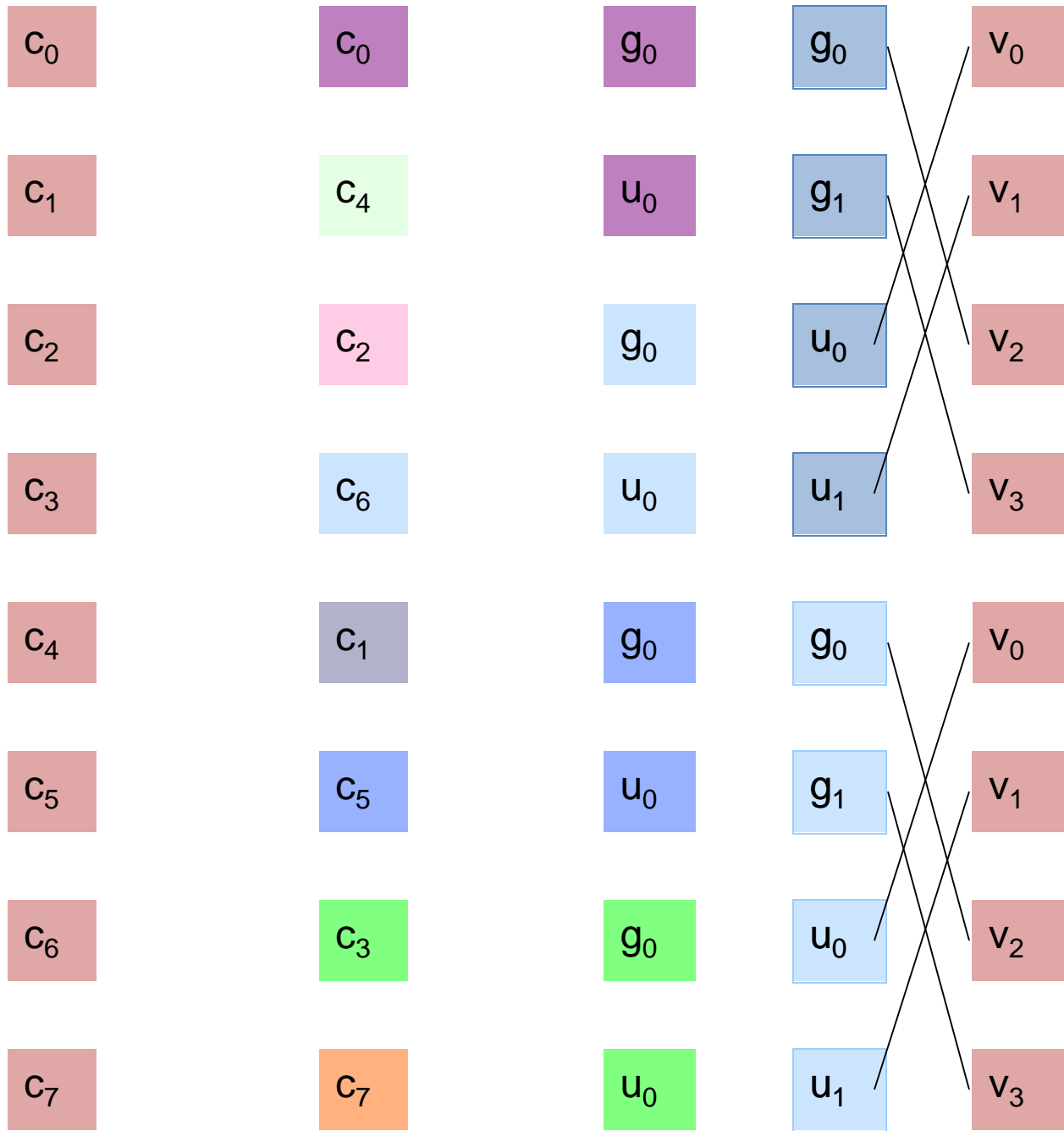


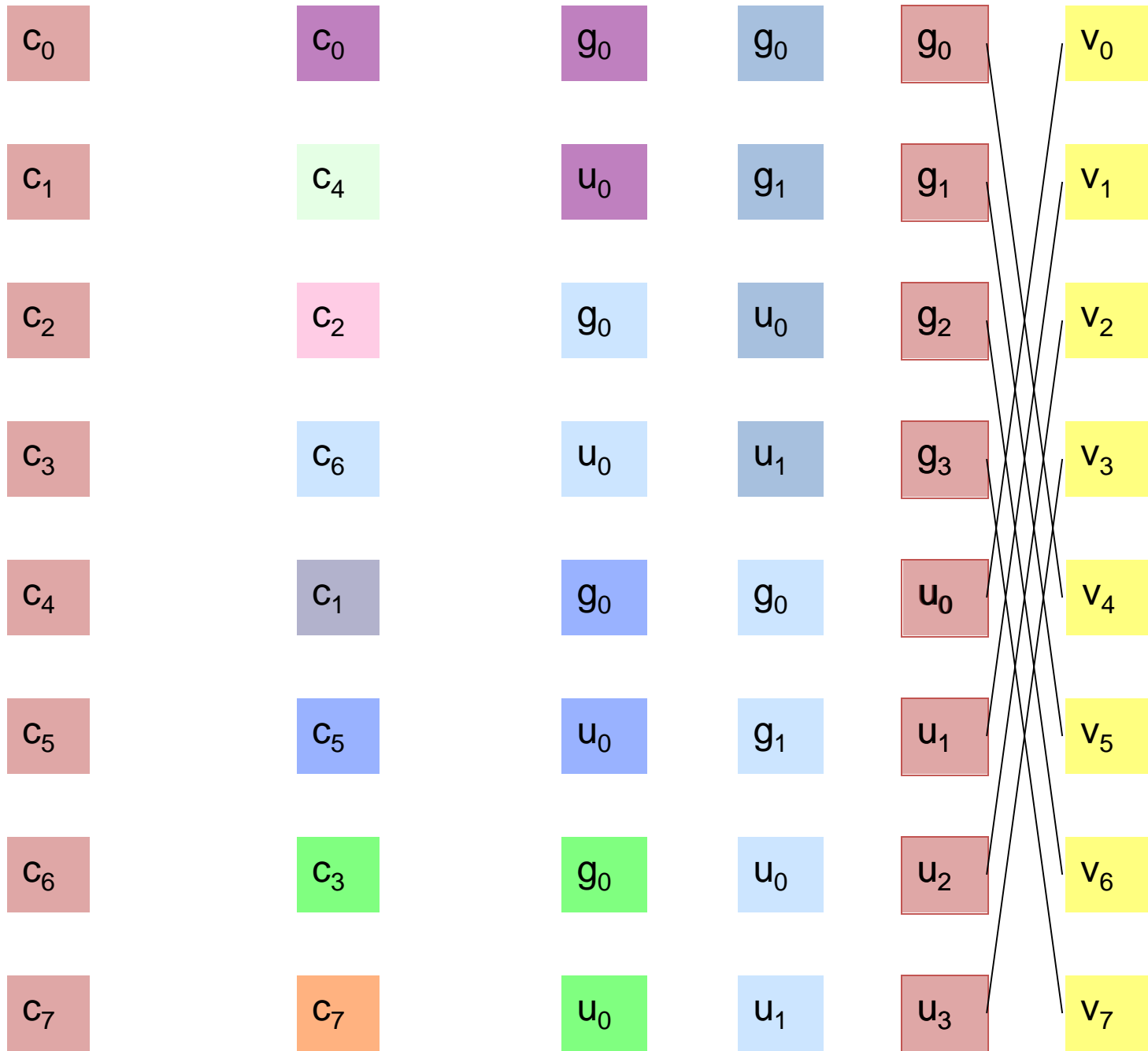
Sortierphase =
Aufrufphase
in einem Schritt:



Kombinationsphase

spaltenweise





Passendes $\omega = \exp(i\pi/m)$:

ω zu $n=2$: $\exp(i\pi)$, $j=0$; ω zu $n=4$: $\exp(i\pi/2)$, $j=0,1$

ω zu $n=8$: $\exp(i\pi/4)$, für Potenzen $j=0,1,2,3$

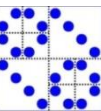
Der Faktor, der im Butterfly bei einem Kombinationsschritt verwendet werden muss, hängt ab

- von der aktuellen Länge der zu kombinierenden Vektoren
- und vom Index j , der die Stelle der Komponenten angibt, die gerade kombiniert werden.

In jeder Spalte werden gleiche Komponenten von g und u zu Index j neu kombiniert zu v (das dann auch wieder g oder u für die nächsten Spalte ist).

Idee:

Schreibe zwei einfache Loop-Programme, eines für die *Sortierphase* und eines für die *Kombinationsphase*.



5.2.3. Sortierphase

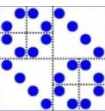
Betrachte Indizes k von Eingabevektor c_k im Binärsystem:

$$k = \left(b_{p-1} \dots b_1 b_0 \right)_2 \quad \text{mit} \quad n = 2^p$$

In jedem Schritt wird nach gerade/ungerade sortiert, also im ersten Schritt nach der **letzten** Binär-Stelle b_0 :
 Alle Komponenten, mit $b_0=0$ kommen nach vorne,
 alle mit $b_0=1$ nach hinten.

Im zweiten Schritt wird wieder nach gerade/ungerade sortiert in den beiden Teilvektoren, d.h.
 in den zwei Teilvektoren $b_1=0$ nach vorne, $b_1=1$ nach hinten,
 also die mit Zweierstelle 1 nach hinten

usw.





Dies entspricht insgesamt einer Sortierung von hinten!

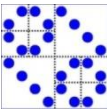
Bei der üblichen Sortierung stehen Zahlen mit erstem Bit $b_{p-1}=1$ hinten und mit $b_{p-1} = 0$ vorne. Danach wird bei der üblichen Sortierung dasselbe bezüglich b_{p-2} wiederholt. ...
So entsteht die üblich sortierte Reihenfolge.

In unserer neuen Sortierphase passiert dasselbe, aber von hinten her, beginnend mit letztem Bit b_0 .

Insgesamt wird durch die rekursiven Aufrufe also die Folge $0, 1, \dots, n-1$ nun aufsteigend sortiert, aber von hinten her! Daher stehen die Elemente nach den rekursiven Aufrufen in dieser sortierten Reihenfolge, die Indizes bitmäßig umgekehrt.

Neuer Index = neue Position: Drehe Bitreihenfolge um.

Wir bezeichnen mit $\pi(k) = \pi((b_{p-1} \dots b_1 b_0)_2) = (b_0 \dots b_{p-1})_2$
das Ergebnis dieser Vertauschungen \rightarrow Bitreversal.



Beispiel:

	binär		reverse	
0	→	000	→	000
1	→	001	→	100
2	→	010	→	010
3	→	011	→	110
4	→	100	→	001
5	→	101	→	101
6	→	110	→	011
7	→	111	→	111

	dezimal	
→		0
→		4
→		2
→		6
→		1
→		5
→		3
→		7

Also: $6 = (110)_2 \rightarrow (011)_2 \rightarrow 3$, 6 wandert an Position 3.

k		→		$\pi(k)$
0	000	000	000	0
1	001	010	100	4
2	010	100	010	2
3	011	110	110	6
4	100	001	001	1
5	101	011	101	5
6	110	101	011	3
7	111	111	111	7

Letzte Spalte mit Bitwerte in ‚reverser‘ Reihenfolge, sortiert nach Stellen in der Binärdarstellung in umgekehrter Reihenfolge.

Beispiel: $p=3, n=8$:

$$k = 5 = (101)_2 \rightarrow \pi(k) = (101)_2 = 5$$

$$k = 3 = (011)_2 \rightarrow \pi(k) = (110)_2 = 6$$

$$k = 1 = (001)_2 \rightarrow \pi(k) = (100)_2 = 4$$

Programmidee für Bit-Reversal:

Zähle für $k=0,1,\dots,n-1$ beginnend mit $\pi(0)=0$ mit $\pi(k)$ ‚hoch‘, also berechne aus $\pi(k-1)$ jeweils das nächste $\pi(k)$.

Sei $k-1 = (b_{p-1}b_{p-2} \dots \mathbf{011\dots11})_2$, und daher

$$\pi(k-1) = (\mathbf{11\dots110} \dots b_{p-2}b_{p-1})_2 ;$$

Daher ist dann $k = (b_{p-1}b_{p-2} \dots \mathbf{100\dots00})_2$, und daher

$$\pi(k) = (\mathbf{00\dots001} \dots b_{p-2}b_{p-1})_2 ;$$

Beim Hochzählen: ... 0 1 1 1 1 \rightarrow ... 1 0 0 0

Von hinten: 1 1 1 1 0 ... \rightarrow 0 0 0 0 1 ...

Um aus $\pi(k-1)$ also $\pi(k)$ zu erhalten, muss das erste (von links) zusammenhängende Einserpaket in $\pi(k-1)$ durch ein Nullerpaket ersetzt werden, und die in $\pi(k-1)$ daran anschließende erste 0 durch eine 1 in $\pi(k)$ ersetzt werden.

Der Rest bleibt unverändert.

Programm Bitreversal:

```

 $\pi(0) = 0$  ;
FOR k = 1:n-1
     $\pi(k) = \pi(k-1)$  ;
    FOR j = 1 : p
        IF j-tes Bit von  $\pi(k)$  ist 0:
            Setze j-tes Bit von  $\pi(k)$  gleich 1;
                                                    % 0  $\rightarrow$  1
            break ;                               % k fertig
        ELSE
            Setze j-tes Bit von  $\pi(k)$  gleich 0 ;
                                                    % 1  $\rightarrow$  0
        ENDIF
    ENDFOR
ENDFOR
ENDFOR

```

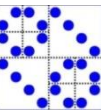
Für jedes k :

Frage von links her ab, ob aktuelle Stelle zu erstem Einserpaket gehört:

- Wenn nein, dann ersetze durch 1, und die Bearbeitung für dieses k ist fertig; k wird um 1 erhöht
- Wenn ja, dann ersetze diese 1 durch 0, und gehe zur nächsten Stelle eine Position nach rechts

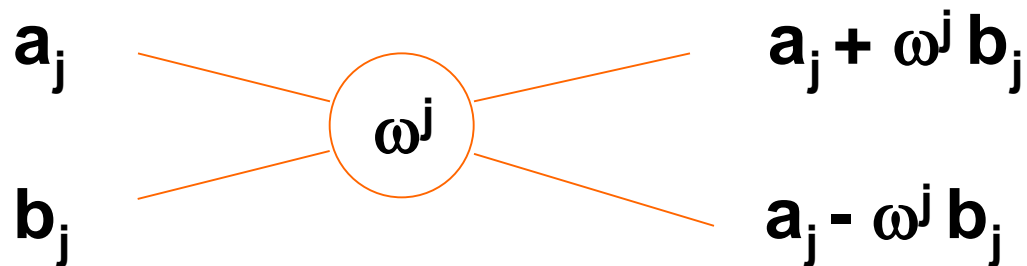
Anstatt des rekursiven Durchlaufs durch das Berechnungstableau führen wir die Umsortierung der Vektorkomponenten in einem Schritt durch.

Genauso sollen die Schmetterlinge jetzt auch spaltenweise ausgeführt werden!



5.2.4. Kombinationsphase

Die umsortierten Komponenten sind nun paarweise in der Form (als Butterfly)



zu kombinieren, und zwar mit wechselndem $\omega^j = \exp(2ij\pi/n)$, je nach Länge n des durch die Kombination entstehenden Gesamtvektors und je nach Stelle j im Teilvektor.

ω ist dabei also abhängig von der Spalte, in dem Tableau (rechte Spalte bezieht sich auf Maximallänge n , usw.).

j ist davon abhängig, die wievielte Komponente in den beiden Vektoren gerade mittels Butterfly kombiniert wird.

Der Abstand zwischen zwei zu kombinierenden Einträgen ist die aktuelle Vektorlänge.

Im Folgenden bezeichnet

k : die jeweilige zu bearbeitende Spalte.

$\omega_j \leftarrow \omega^j$

j : Komponente j in den zu kombinierenden Teilvektoren

s : die in Spalte k auftretenden Vektor-Paar-Kombinationen

Programm Kombinationsphase

```

r = 1 ;
FOR k = 1:p
  m = r ;
  r = 2m ;
  FOR j = 0:m-1
     $\omega_j = \cos(\pi j/m) + i \sin(\pi j/m) ;$ 
    FOR s = 0:r:n-r
      g = vj+s ;
      h =  $\omega_j \cdot v_{j+s+m}$  ;
      vj+s = g + h ;
      vj+s+m = g - h ;
    ENDFOR
  ENDFOR
ENDFOR

```

5.2.5. Ergänzung Kosten:

Kosten für IDFT der Länge $2n =$
 $(2 * \text{Kosten für IDFT der Länge } n) + a*n$

also

$$T_{2n} = 2*T_n + a*n$$

Setze $f_p := T_{2^p} = T_n$, $n=2^p$ oder $p=\log_2(n)$

Dann gilt

$$f_{p+1} = 2*f_p + a*n = 2*f_p + a*2^p, \quad * x^p, \quad \Sigma$$

$$\sum_{p=0}^{\infty} f_{p+1} x^p = 2 \cdot \underbrace{\sum_{p=0}^{\infty} f_p x^p}_{\mathbf{F(x)}} + a \cdot \sum_{p=0}^{\infty} 2^p x^p$$

$$(F(x) - f_0) / x = 2 \cdot F(x) + a / (1 - 2x)$$

$$F(x) \cdot (1 - 2x) = ax / (1 - 2x) + f_0$$

$$F(x) = ax / (1 - 2x)^2 + f_0 / (1 - 2x)$$

heißt 'Erzeugende Funktion' der Folge f_p .

Reihenentwicklung von $F(x)$:

$$a \cdot \sum_{p=0}^{\infty} \frac{(p+1)!}{p!} 2^p x^{p+1} + f_0 \cdot \sum_{p=0}^{\infty} 2^p \cdot x^p =$$

$$= f_0 + \sum_{p=1}^{\infty} \left(a \frac{p!}{(p-1)!} 2^{p-1} + f_0 2^p \right) \cdot x^p =$$

$$= f_0 + \sum_{p=1}^{\infty} (pa + 2f_0) 2^{p-1} x^p$$

und daher $f_p = (ap + 2f_0)2^{p-1}$ für $p=1,2,\dots$ oder

$$T_n = T_{2^p} = (ap + 2f_0)2^{p-1} = \frac{a}{2}n \cdot \log_2(n) + f_0n = O(n \cdot \log_2(n))$$

$f_0 = T_1$: Kosten für IDFT der Länge 1

a : Kosten für Operationen in **einem** Butterfly

Durchführbarkeit der Divide & Conquer – Idee
ist so nur möglich, wenn n in viele Faktoren zerfällt
(z.B. wenn n eine Zweier-Potenz ist).

Beispiel: FFT für $n=3^q$

$$\begin{aligned}
 v_j &= \sum_{k=0}^n c_k e^{2\pi i j k / n} = \sum_{k=0}^{n/3-1} c_{3k} e^{2\pi i j 3k / n} + \\
 &+ \sum_{k=0}^{n/3-1} c_{3k+1} e^{2\pi i j (3k+1) / n} + \sum_{k=0}^{n/3-1} c_{3k+2} e^{2\pi i j (3k+2) / n} = \\
 &= F_0 + e^{2\pi i j / n} F_1 + e^{2\pi i 2j / n} F_2
 \end{aligned}$$

Dazu benötigt

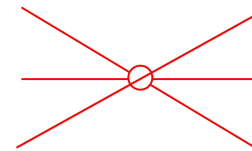
- IDFT(c_0, c_3, c_6, \dots)
- IDFT(c_1, c_4, c_7, \dots)
- IDFT(c_2, c_5, c_8, \dots)

Sortieren modulo 3: Bitreversal in ternärer Darstellung
z.B. dreistellig für $n = 3^3 = 27$:

$$5 = 1 \cdot 3 + 2 \cdot 1 = (012)_3 \rightarrow (210)_3 = 2 \cdot 9 + 1 \cdot 3 = 21$$

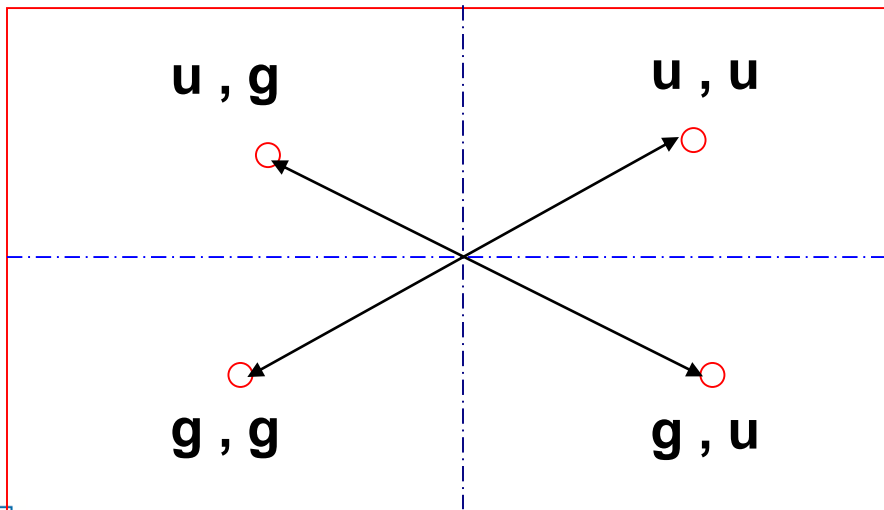
In Sortierphase wandert c_5 an die Position c_{21} .

„Dreier-Butterfly“

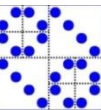


Beispiel: FFT in 2D

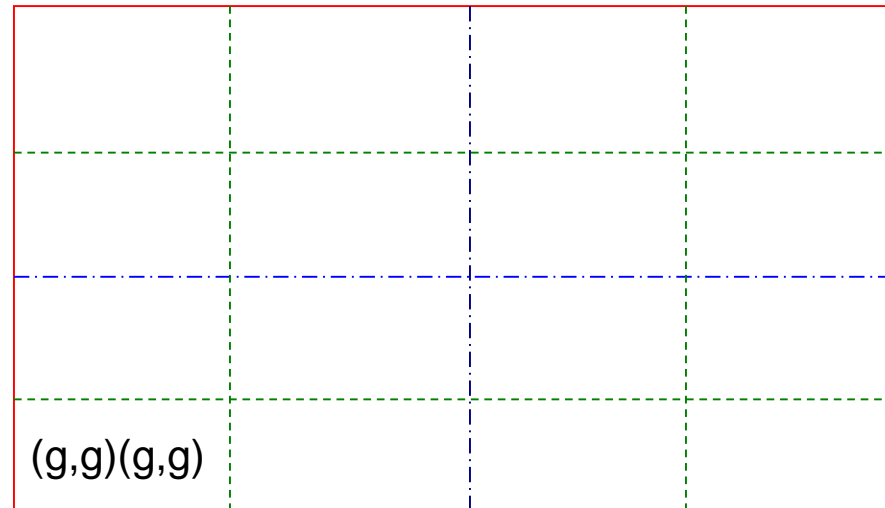
$$\begin{aligned}
 v_{j,s} &= \sum_{k=0}^N \sum_{r=0}^M c_{k,r} e^{2\pi ijk/N} e^{2\pi irs/M} = \\
 &= \sum_{k=0}^{N/2-1} \sum_{r=0}^{M/2-1} c_{2k,2r} e^{2\pi ij2k/N} e^{2\pi i2rs/M} + c_{2k,2r+1} e^{2\pi ij2k/N} e^{2\pi i(2r+1)s/M} \\
 &+ c_{2k+1,2r} e^{2\pi ij(2k+1)/N} e^{2\pi i2rs/M} + c_{2k+1,2r+1} e^{2\pi ij(2k+1)/N} e^{2\pi i(2r+1)s/M} = \\
 &= F_{00} + e^{2\pi is/M} F_{10} + e^{2\pi ij/N} F_{01} + e^{2\pi ij/N} e^{2\pi is/M} F_{11}
 \end{aligned}$$



Butterfly kombiniert 4
Komponenten zu 4 neuen
Komponenten



Rekursiv weiter aufspalten:



Doppelsumme erlaubt unterschiedliches Abarbeiten:

Entweder 1-dim. FFT in Zeilen + 1-dim. FFT in Spalten,
 oder
 rekursiv 2-dim. FFT in Blöcken.



1-dim. FFT für allgemeines n :

Allgemein kann man bei einer Primfaktorzerlegung der Form $n = n_1 \dots n_p$ eine Aufspaltung in n_1 Teilsummen im ersten Schritt, \dots , in n_p Teilsummen im letzten Schritt ausführen.

Ist z.B. $n=8$, so lässt sich die IDFT-Summe aufspalten in 2-2-2, 2-4, 4-2, oder 8 in 3, 2, oder 1 Level!

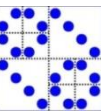
Ist n eine Primzahl \rightarrow FFT geht so nicht!

Durch gruppentheoretische Überlegungen lässt sich dann aber z.B. dieser Fall auf die Berechnung der FFT zu $n-1$ und $n-2$ zurückführen, die dann keine Primzahlen sind.

Andere Idee: Zero Padding oder Einbettung in Zweier-Potenz.



Verallgemeinerung auf allgemeine Stützstellen: NFFT





FFTW = Fastest Fourier Transform in the West: *Meta-programm*



Teste für CPU, Cache, usw. die verschiedenen Faktorzerlegungen von n .

Wähle diejenige mit schnellster Laufzeit aus und erstelle automatisch FFT-Routine.

$$n = 8 = 1 * 8 = 2 * 4 = 2 * 2 * 2 = 4 * 2 = 8 * 1$$

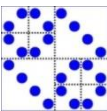
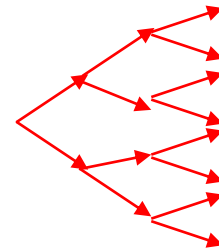
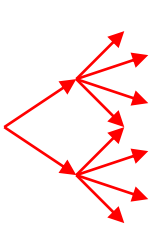
IDFT Länge 8 oder

IDFT Länge 2 gefolgt von IDFT 4 oder

IDFT Länge 2, IDFT Länge 2, IDFT Länge 2 oder

IDFT Länge 4 gefolgt von IDFT 2

ohne IDFT, als Matrixmultiplikation



5.3. Anwendungen

5.3.1. Fourierreentwicklung:

Stückweise stetige, 2π -periodische (in $[-\pi, \pi]$) Funktion $f(x)$ lässt sich als Fourier-Reihe darstellen (vgl. Taylor/Potenz-Reihe):

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx))$$
$$\left\{ \equiv \sum_{k=-\infty}^{k=\infty} c_k e^{ikx} = \sum_{k=-\infty}^{k=\infty} c_k (e^{ix})^k = \sum c_k z^k \right\}$$

a_k und b_k heißen Fourier-Koeffizienten von $f(x)$ und berechnen sich (wegen der Orthogonalität der Funktionen $\cos(kx)$ und $\sin(kx)$) aus

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx) dx, \quad b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(kx) dx$$

Koeffizienten geben die Größe der Anteile von Vielfachen der Grundfrequenz ($1/2\pi$) an, aus denen sich f zusammensetzt

Wellenzahl: k

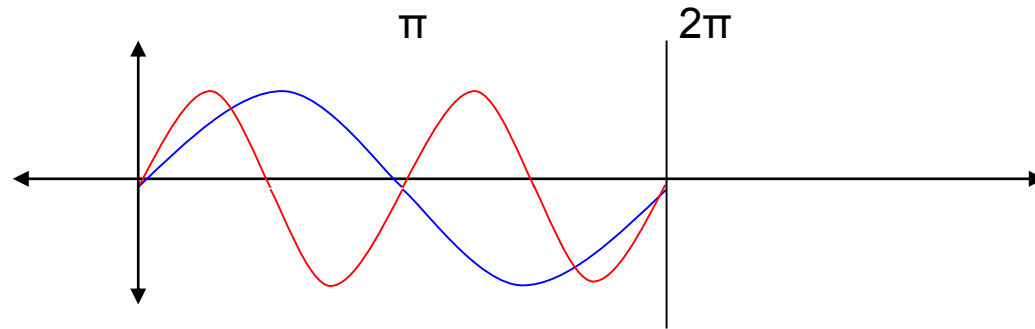
Periode: $2\pi/k$

Frequenz: $k/(2\pi)$

Grundfrequenz für $k=1$, $\cos(x)$:

Periode 2π , Frequenz $1/(2\pi)$

**(a_k, b_k) messen Anteil von f zur Frequenz $k / (2\pi)$,
(bzw. Periode $2\pi/k$, oder Wellenzahl k).**



Schwingung mit Wellenzahl **$k=1$** und **$k=2$**

Wellenzahl **$k=2$** entspricht Periode π , bzw. Frequenz $1/\pi$

$\cos(kx)$, $\sin(kx)$ bilden Orthonormalsystem (Basis)!

Fourierreihe entspricht Taylorreihe

Für periodische/auf Intervall definierte Funktion

Satz: Das trigonometrische Polynom

$$f_n(x) = \frac{a_0}{2} + \sum_{k=1}^n (a_k \cos(kx) + b_k \sin(kx))$$

stellt die optimale Approximation an die Funktion f dar aus dem Vektorraum der trigonometrischen Polynome vom Grad n :

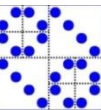
$$\|f_n(x) - f(x)\|_2^2 = \int_{-\pi}^{\pi} (f_n(x) - f(x))^2 dx \quad \text{ist minimal.}$$

f_n ist die Orthogonalprojektion von f bzgl. \int auf $[-\pi, \pi]$ auf den Raum der trigonometrischen Polynome vom Grad n

Näherungsweise Berechnung der Fourierkoeffizienten aus dem Integral mittels Trapezregel und äquidistanten Stützstellen

$$x_0 = -\pi, \quad x_j = -\pi + j \frac{2\pi}{n}, \quad x_n = \pi.$$

$$\begin{aligned} a_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx) dx \approx \\ &\approx \frac{1}{\pi} \cdot \frac{2\pi}{n} \cdot \left(\frac{1}{2} f(-\pi) \cos(-k\pi) + f(x_1) \cos(kx_1) + \dots \right. \\ &\quad \left. \dots + f(x_{n-1}) \cos(kx_{n-1}) + \frac{1}{2} f(\pi) \cos(k\pi) \right) = \\ &= \frac{2}{n} \sum_{j=0}^{n-1} f(x_j) \cos\left(k\left(-\pi + j \frac{2\pi}{n}\right)\right) = \\ &= \frac{2}{n} \cos(k\pi) \sum_{j=0}^{n-1} f(x_j) \cos\left(\frac{2\pi k j}{n}\right) + \frac{2}{n} \sin(k\pi) \sum_{j=0}^{n-1} f(x_j) \sin\left(\frac{2\pi k j}{n}\right) \end{aligned}$$



Daher ergeben sich die a_k näherungsweise aus den Real- und Imaginärteilen von

$$\sum_{j=0}^{n-1} f(x_j) \exp\left(\frac{2i\pi kj}{n}\right) = \sum_{j=0}^{n-1} f(x_j) \cos\left(\frac{2\pi kj}{n}\right) + i \cdot \sum_{j=0}^{n-1} f(x_j) \sin\left(\frac{2\pi kj}{n}\right)$$

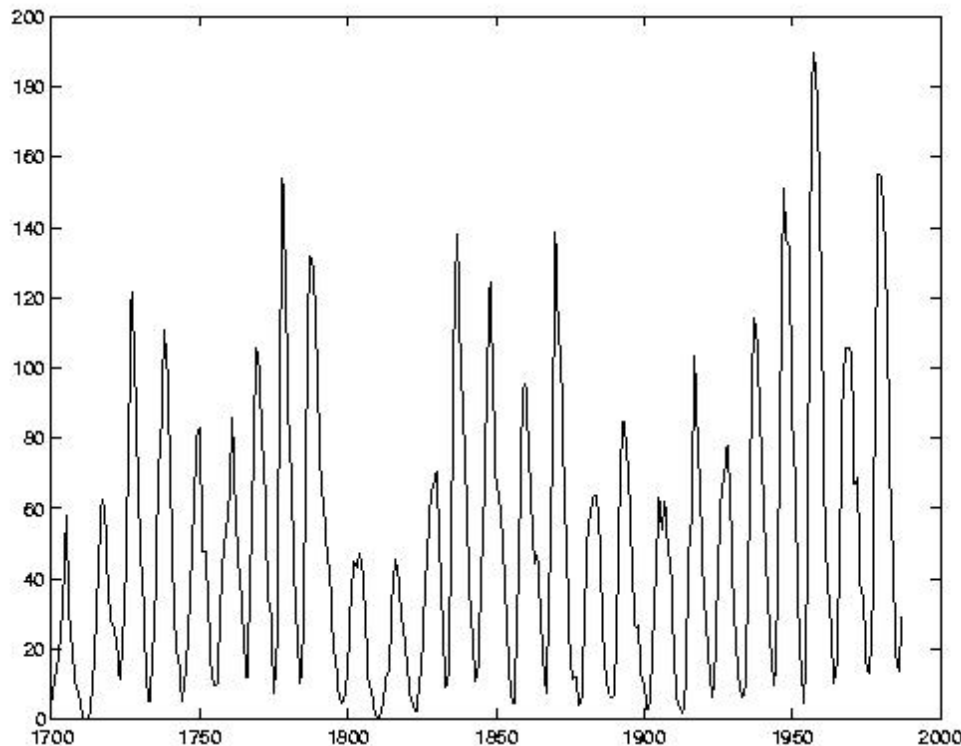
Diese Werte erhält man aus der IDFT angewendet auf die Funktionswerte an den Stützstellen.

Genauso lassen sich die b_k annähern.

Also können mittels DFT auf den Vektor der Funktionswerte die Frequenzanteile von f näherungsweise bestimmen werden.

DFT transformiert Funktionswerte in Koeffizienten

Sonnenflecken treten alle 11 Jahre verstärkt auf.
 Seit 1700 wird die jährliche Sonnenfleckenaktivität beschrieben durch die sog. Wolfer-Zahl (Größe und Anzahl der Flecken) (Rudolf Wolf ca. 1850).



**Wolferzahlen
für die Jahre
1700 bis 2000**

Wolferzahlen in Vektor v als Funktionswerte einer unbekannt periodischen Funktion g mit den Jahreszahlen als Stützstellen.

Gesamtbeobachtungszeitraum T besteht aus 300 Jahren.

Ersetze daher Intervall $[0, 2\pi]$ durch das Intervall $[0, T]$ durch den Übergang von

$$\exp(ikx) \rightarrow \exp(ikx \cdot 2\pi/300)$$

Zeitintervall $\Delta = 1$ Jahr, in dem eine Wolferzahl bestimmt wurde;

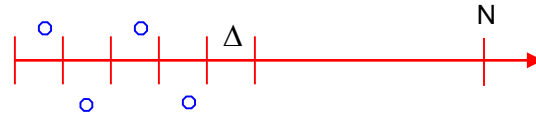
$N = T / \Delta$ ist Anzahl der Beobachtungsintervalle =

= Länge des Vektors v

= Anzahl der Stützstellen;

Wellenzahl wieder k , Periode $T/k = 300 \text{ J.}/k$, Frequenz k/T ;

Grundfrequenz also $1/T$.



Gibt es Schwingungen, die nicht erkannt werden können?

Es können nur periodische Vorgänge erfasst werden mit einer Periode $> 2\Delta = 2$ Jahre, da bei kleineren Perioden die ‚Schwingung‘ feiner als die feinste Unterteilung (1 Jahr) wäre, und daher als solche nicht erkennbar ist.

Daher ist die größte beobachtbare Wellenzahl k gleich $N/2=150$, entspricht der Frequenz $(N/2)(1/T)=1/(2\Delta)$, der sog. Nyquist-Frequenz

Berechne $y = \text{FFT}(v)$

$y(0)$ ist die Gesamtsumme aller Wolferzahlen und wird gleich 0 gesetzt (enthält keine Angabe über Periodizität).

Die anderen Koeffizienten von y enthalten die Größe der verschiedenen Frequenzanteile von g (aber als komplexe Zahlen).

Berechne $p(k) = |y(k)|^2$ für $k = 1, \dots, N/2$

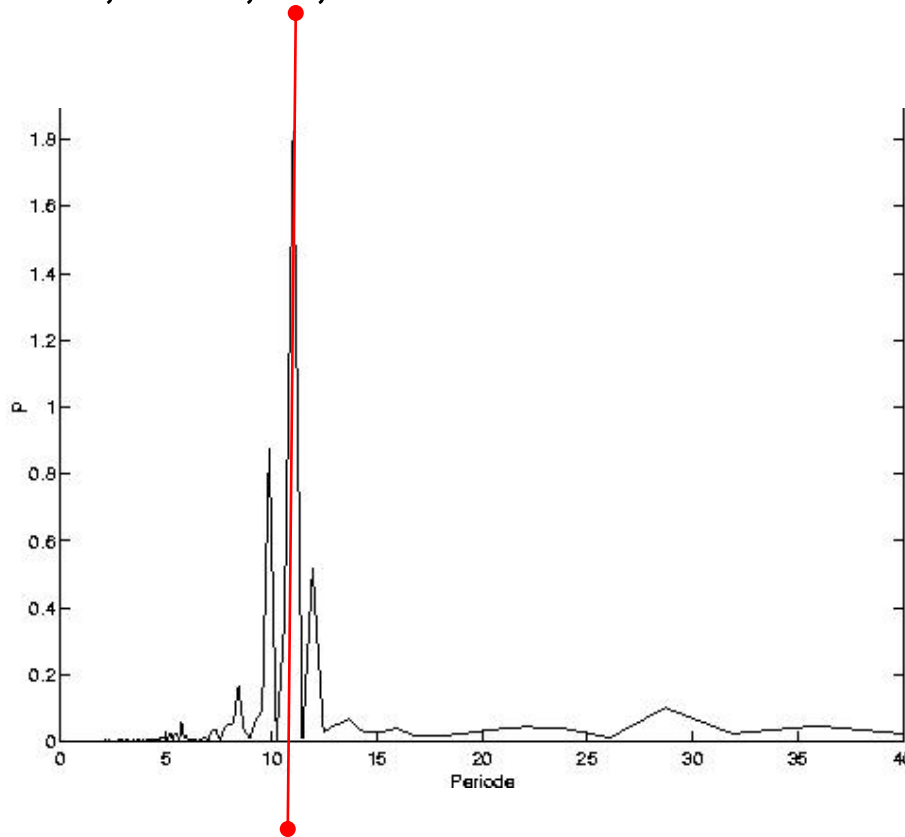
also untersuche nur die Beträge der Frequenzanteile bis zur Nyquist-Wellenzahl $k = (N/2)$.

$p(k)$ misst die Größe der k -ten Vielfachen der Grundfrequenz, also der Frequenz

$$f(k) = k / T = k / (\Delta N) \quad \text{für } k=1,2,\dots,N/2$$

Anders ausgedrückt bestimmt $p(k)$ das Gewicht der Periode $1 / f(k) = T / k$ in der Funktion f .

Wir tragen nun den Vektor p auf gegen die dazugehörigen Perioden T/k , $k=1, \dots, N/2$



Betragsquadrate der Fourierkoeffizienten, aufgetragen über die dazugehörigen Perioden.

Wir können den Zyklus (Periode) von 11 Jahren direkt ablesen.

Allgemeines Problem:

In welcher Form stellt man numerische Daten dar?

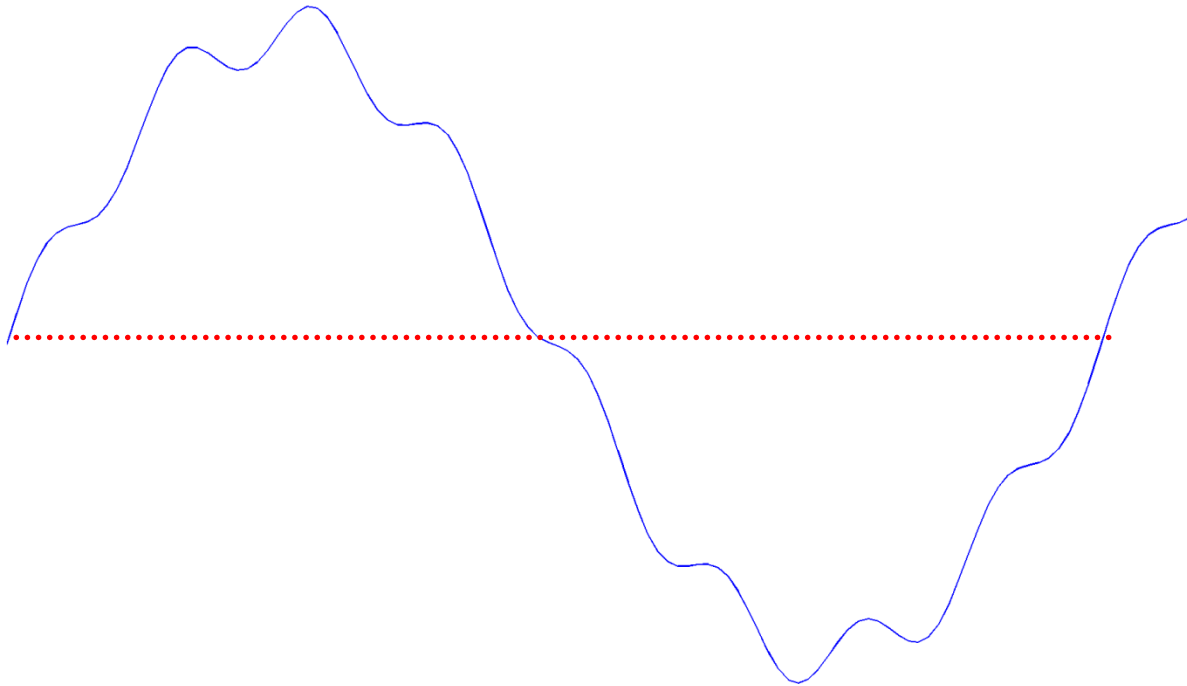
Normalerweise aus diskreten Messwerten: Samples $f(x_j)$
Dies entspricht einer Darstellung im Ortsraum.

Nachteil: Viele Daten, schlecht komprimierbar, keine Analyse!

Besser: Übergang zu einer Darstellung bzgl. geeigneter
Basisfunktionen (z.B. x^j , $\exp(jx)$, $\cos(jx)$,...)

Dann reichen ev. wenige Koeffizienten aus, um das Signal (fast) vollständig zu beschreiben. Außerdem kann man aus den Koeffizienten Schlüsse über das Verhalten der Funktion ziehen (Frequenzanalyse).

Beispiel: $f(x) = 0.5 \cdot \sin(x) + 0.1 \cdot \sin(10x)$



Zwei Koeffizienten reichen aus zur Beschreibung

Die Koeffizienten bezeichnet man auch als Beschreibung der Funktion im Phasenraum (Frequenzraum)

Bilder werden üblicherweise als Matrix gespeichert, deren Einträge pixelweise den Grauwert des entsprechend nummerierten Pixels enthalten, bzw. bei Farbbildern den Anteil einer der drei Farbkomponenten Rot, Grün oder Blau (= RGB).

RGB wird meist in der YUV-Form umgewandelt, bei der die Matrix Y die Helligkeit beschreibt, und U und V den Farbton (U, V lassen sich stärker komprimieren!).

JPEG zerlegt das Bild in 8 x 8 – Pixel große Blöcke, die zunächst getrennt bearbeitet werden. Auf jede dieser Teilmatrizen wird eine zweidimensionale Cosinus-Transformation angewandt: DCT

Warum DCT, warum nicht DFT?

Aus Anwendung der DCT erhält man für die Teilmatrizen wieder die Frequenzanteil-koeffizienten a_k und b_k in Teilmatrizen.

*Die Teilmatrix mit den Frequenzanteilen wird nun quantisiert, d.h. die darin stehenden reellen Zahlen werden bestimmten Zahlenintervallen zugeordnet und Intervallweise jeweils durch eine Zahl angenähert, die für ein ganzes Intervall gilt.
Maschinenzahlen!*

Dadurch werden auch automatisch alle kleinen Komponenten, die in dem ersten Intervall mit den kleinsten Werten liegen, durch Null ersetzt.

Dies erzeugt einen Qualitätsverlust!

*Die entstandene Gesamtmatrix aus den diskreten Zahlenwerten wird codiert – **Huffman-Codierung***

Nachteile:

Fourier-Methoden haben Schwierigkeiten mit Kanten in Bildern
Kanten \leftrightarrow Unstetigkeiten

Durch stetige Funktionen $\cos(kx)$, $\sin(kx)$ sind Kanten schlecht darstellbar! Differenzierbarkeit!

Kosten der DCT: $O(n \log(n))$ bei n Pixel. Zu teuer.

Daher DCT auf 8×8 -Blöcke.

JPEG2000:

Anstatt Cosinus-Transformation auf 8×8 wendet man eine Wavelet-Transformation auf größere Teilmatrizen an.

Wavelet-Ansatzfunktionen haben keine Problem mit Kanten, vgl. B-Splines. Genauere Untersuchung folgt später.

Kosten für Wavelet-Transformation $O(n)$

Beispiel: MATLAB Gibbs-Phänomen (recht.m)

Fourierkoeffizienten für Rechtecksignal;
Überschwingen an Kante unvermeidbar bei Darstellung durch
endliches trigonometrisches Polynom!