technische universität
dortmund

fakultät für informatik
informatik 12

Synthese Eingebetteter Systeme

Sommersemester 2011

17 – Abbildung von Anwendungen: Entwurfsraumerkundung und Beispiele

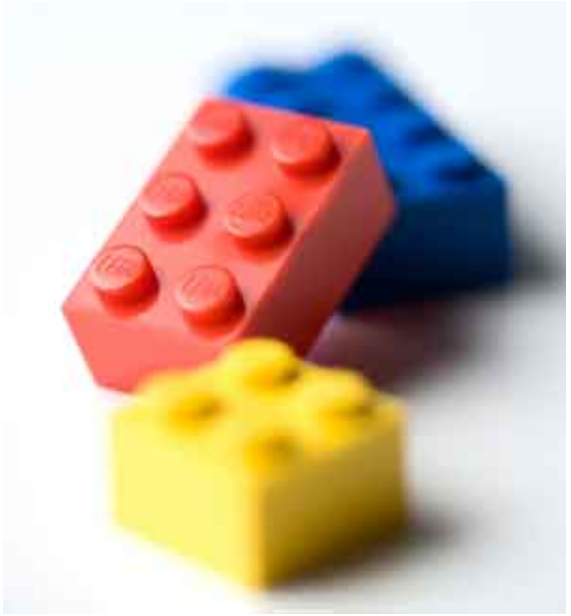
Michael Engel
Informatik 12
TU Dortmund

2011/06/29

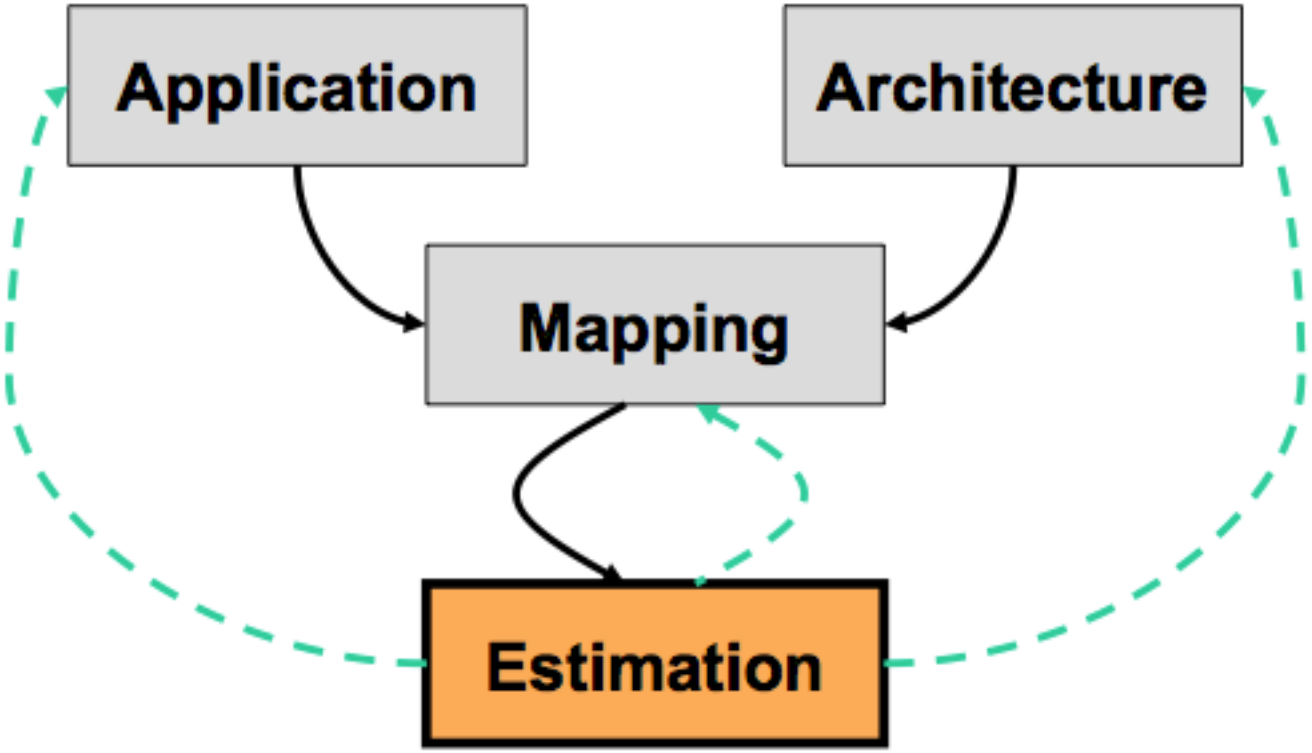
– unter Verwendung von Foliensätzen von Prof. Lothar Thiele und Dr. Iuliana Bacivarov, ETH Zürich –

DOL: Entwurfsraumerkungung & Beispiele

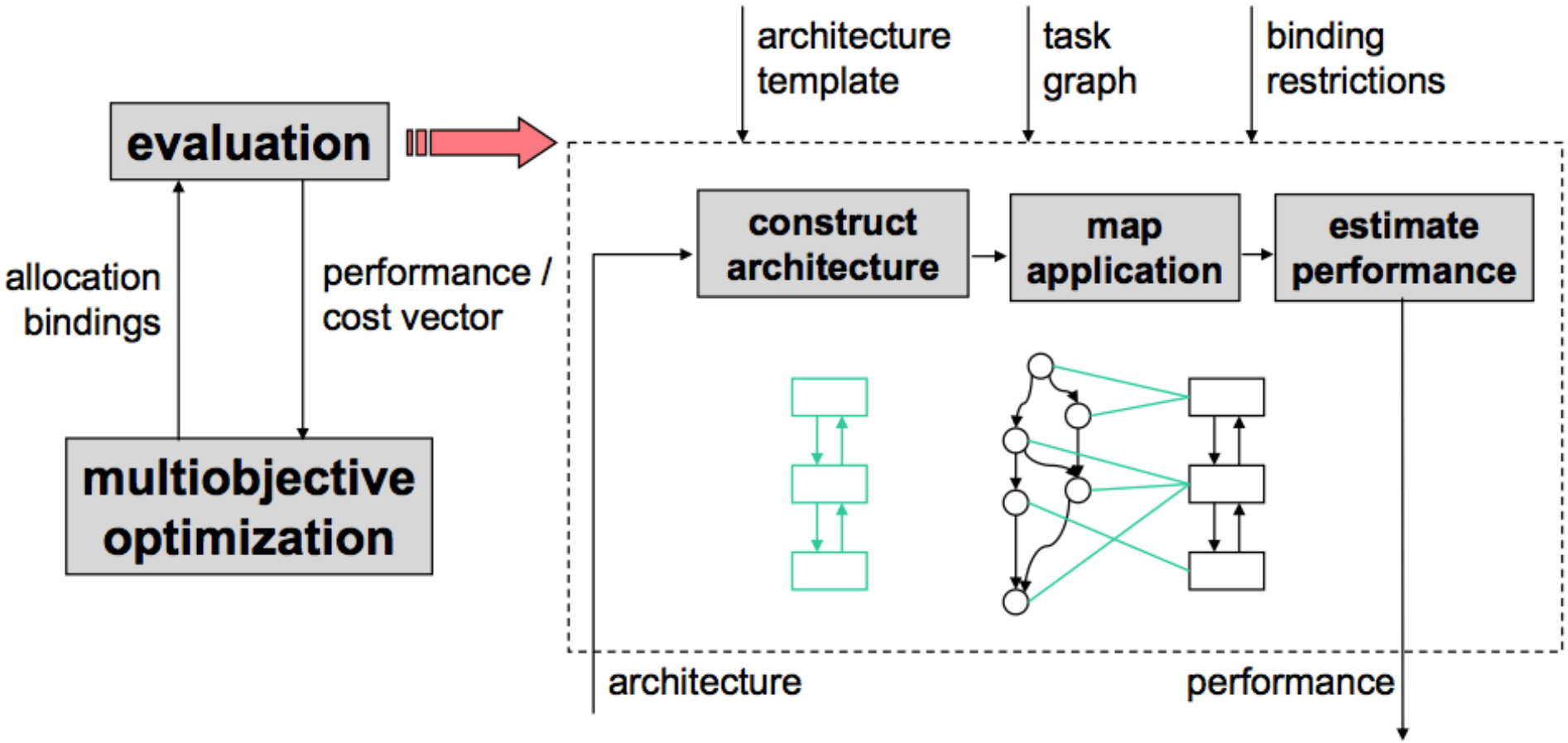
- Entwurfsraumerkundung
- Einfache Beispiele
- MPEG2-Decoder



Entwurfsraumerkundung



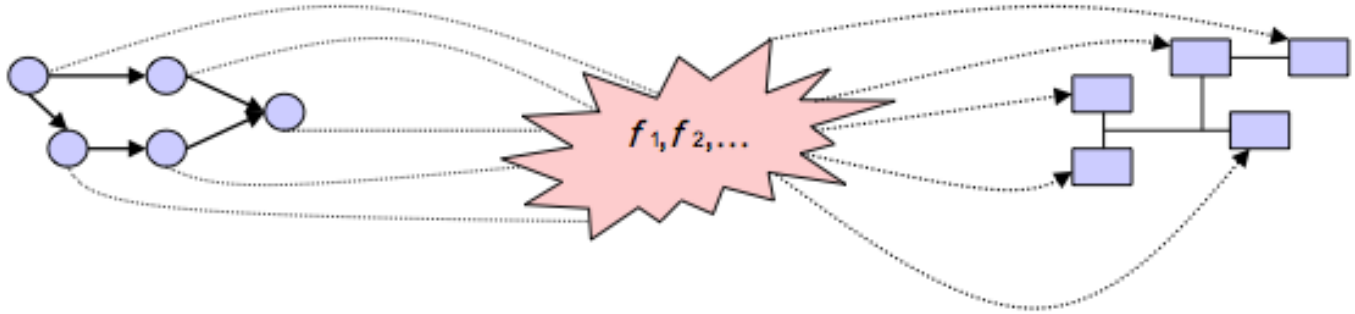
Entwurfsraumerkundung im Detail



Beispiel 1: Einfaches Modell

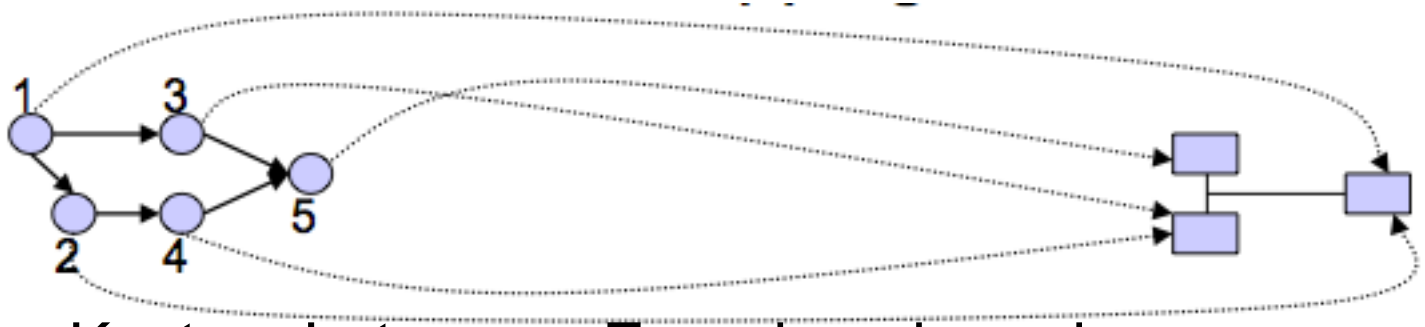
Gegeben:

Algorithmus Abbildungen Architektur



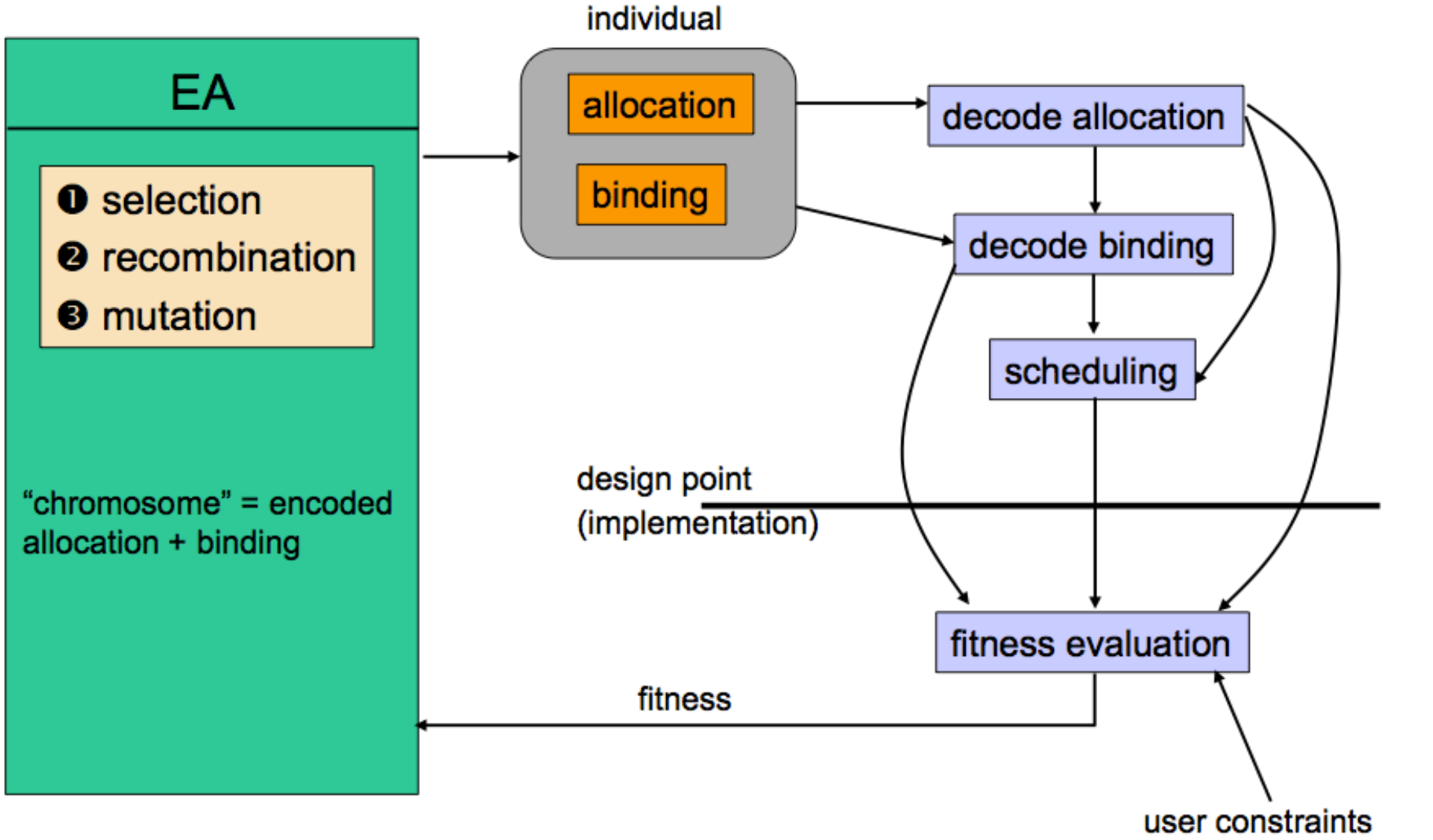
Gesucht:

Schedule Abbildung Architektur



Ziele: Kosten, Latenzen, Energieverbrauch

Beispiel 1: Evolutionärer Algorithmus für Entwurfsraumerkundung



Beispiel 1: Basismodell

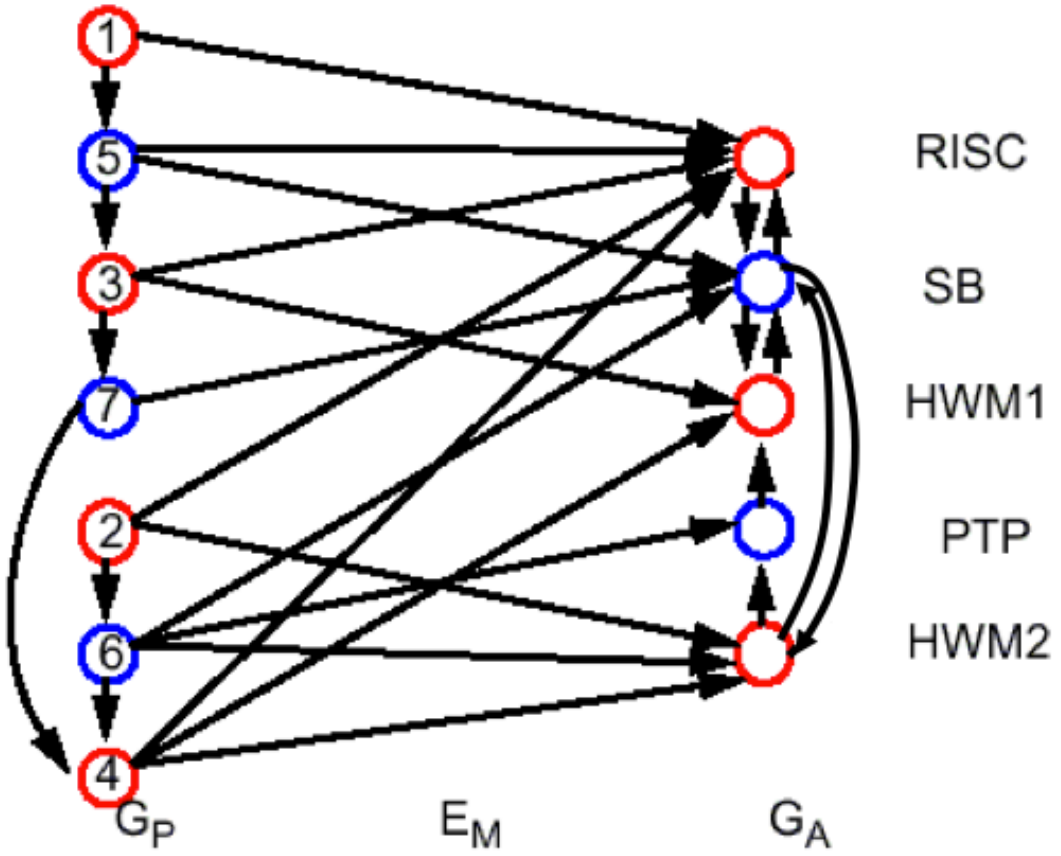
- Definition:

Ein *Spezifikationsgraph* ist ein Graph $G_S=(V_S,E_S)$ bestehend aus einem *Datenflussgraphen* G_P, einem *Architekturgraphen* G_A und *Kanten* E_M.

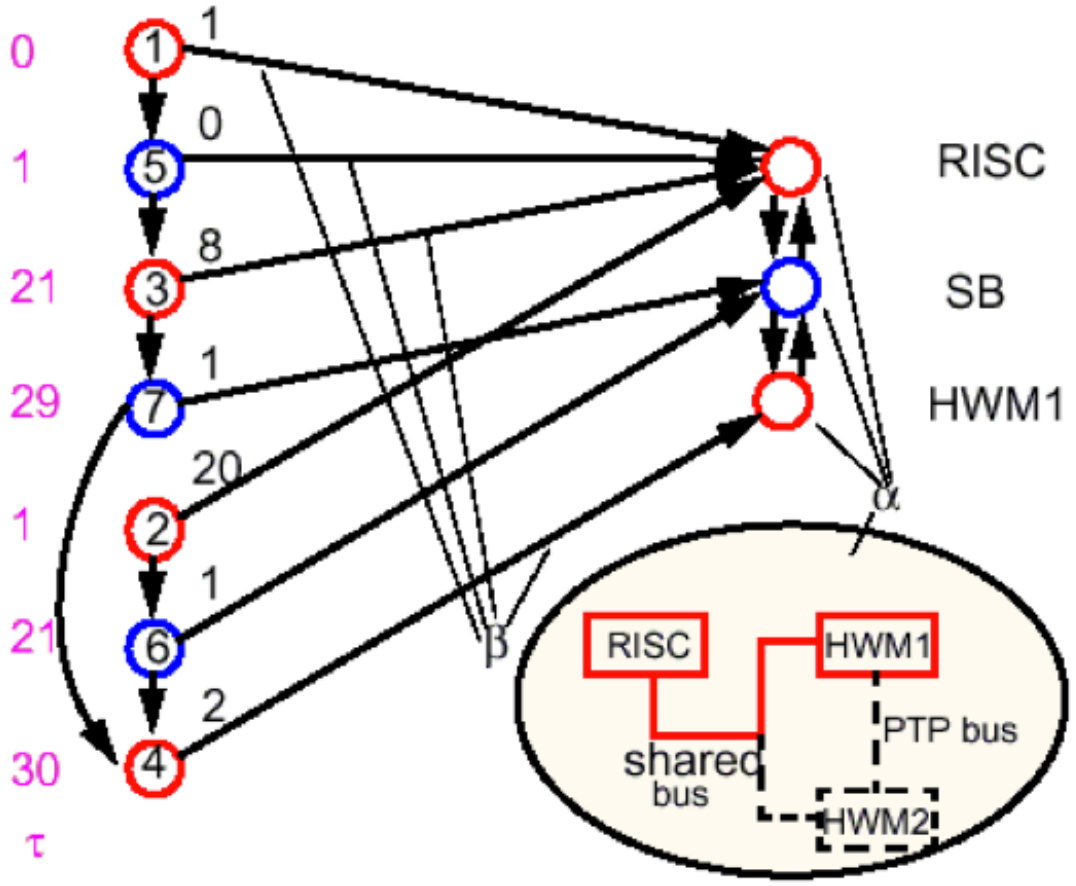
Es gilt:

$V_S = V_S \cup V_A$

$E_S = E_P \cup E_A \cup E_M$



Beispiel 1: Abbildung



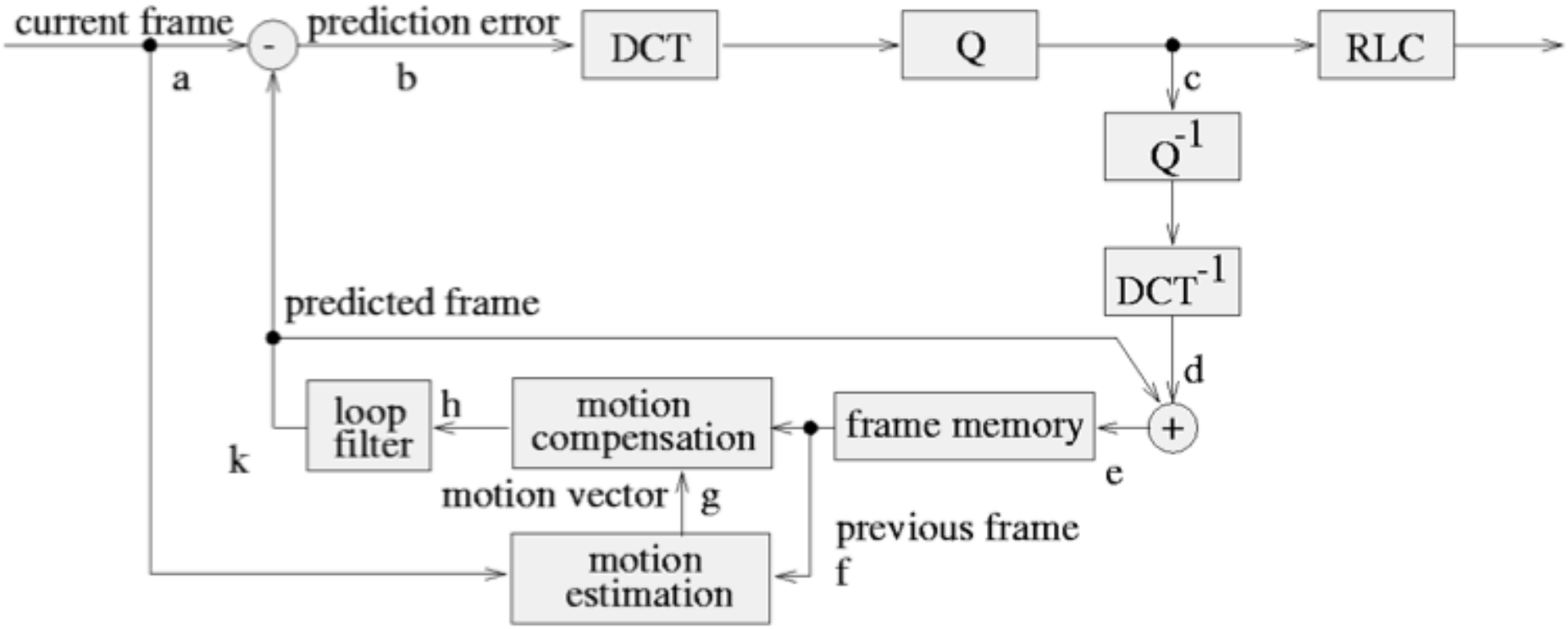
Beispiel 1: Aufgaben

Codierung von (Allokation + Bindung)

- Einfache Codierung
  - Z.B. 1 Bit pro Ressource, eine Variable pro Bindung
  - Einfach implementierbar
  - Viele nicht realisierbare Lösungen
- Codierung und *Reparatur*
  - Z.B. Einfache Codierung, abgeändert, so dass für jedes $v_p \in V_P$ mindestens ein $v_A \in V_A$ existiert mit einem $\beta(v_P) = v_A$
  - Reduziert Anzahl nicht realisierbarer Lösungen
- Erzeugung der Ausgangspopulation, Mutation
- Rekombination

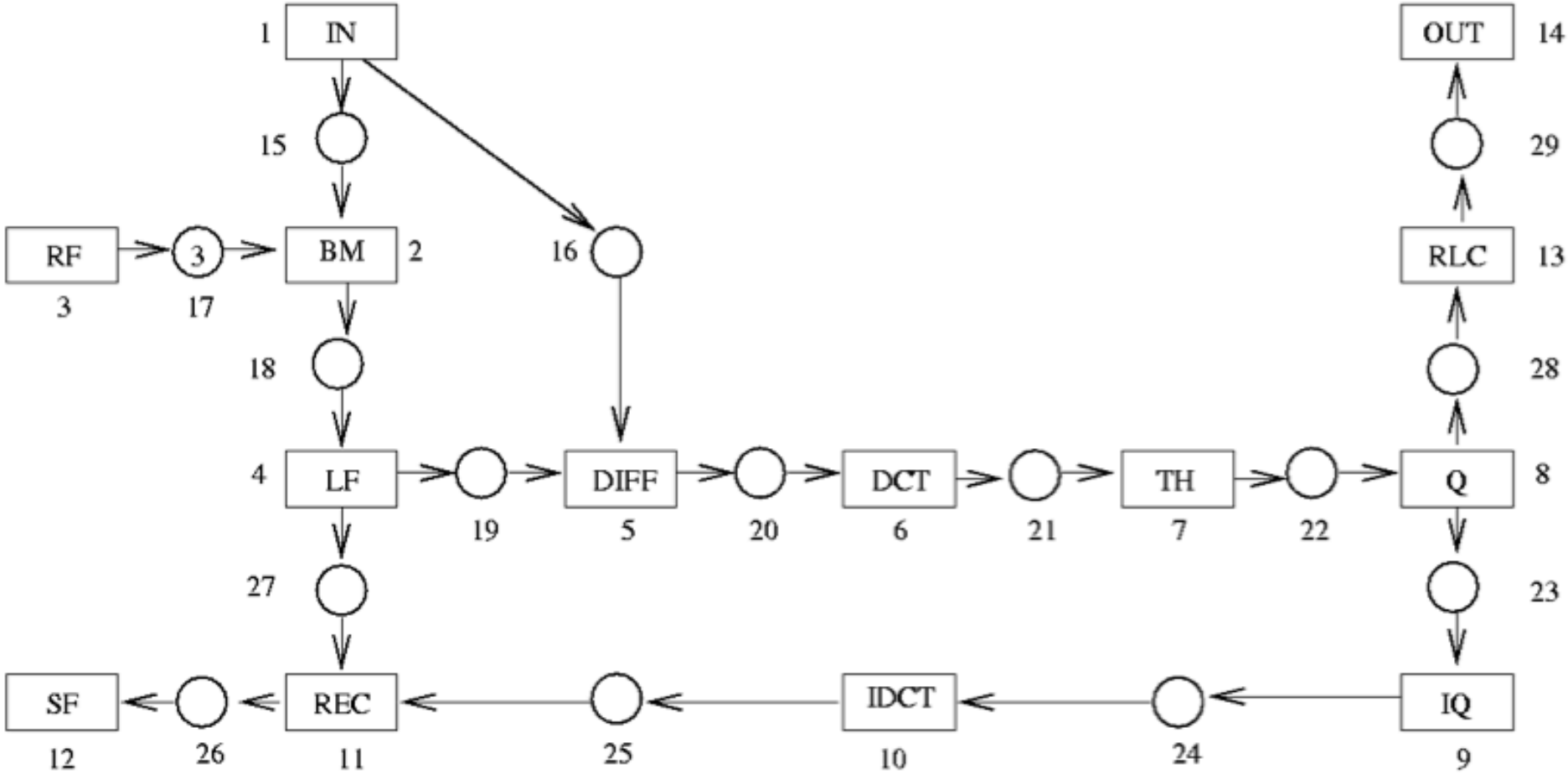
Beispiel 1: Spezifikation

Verhaltensspezifikation eines Videocodes für Videokompression

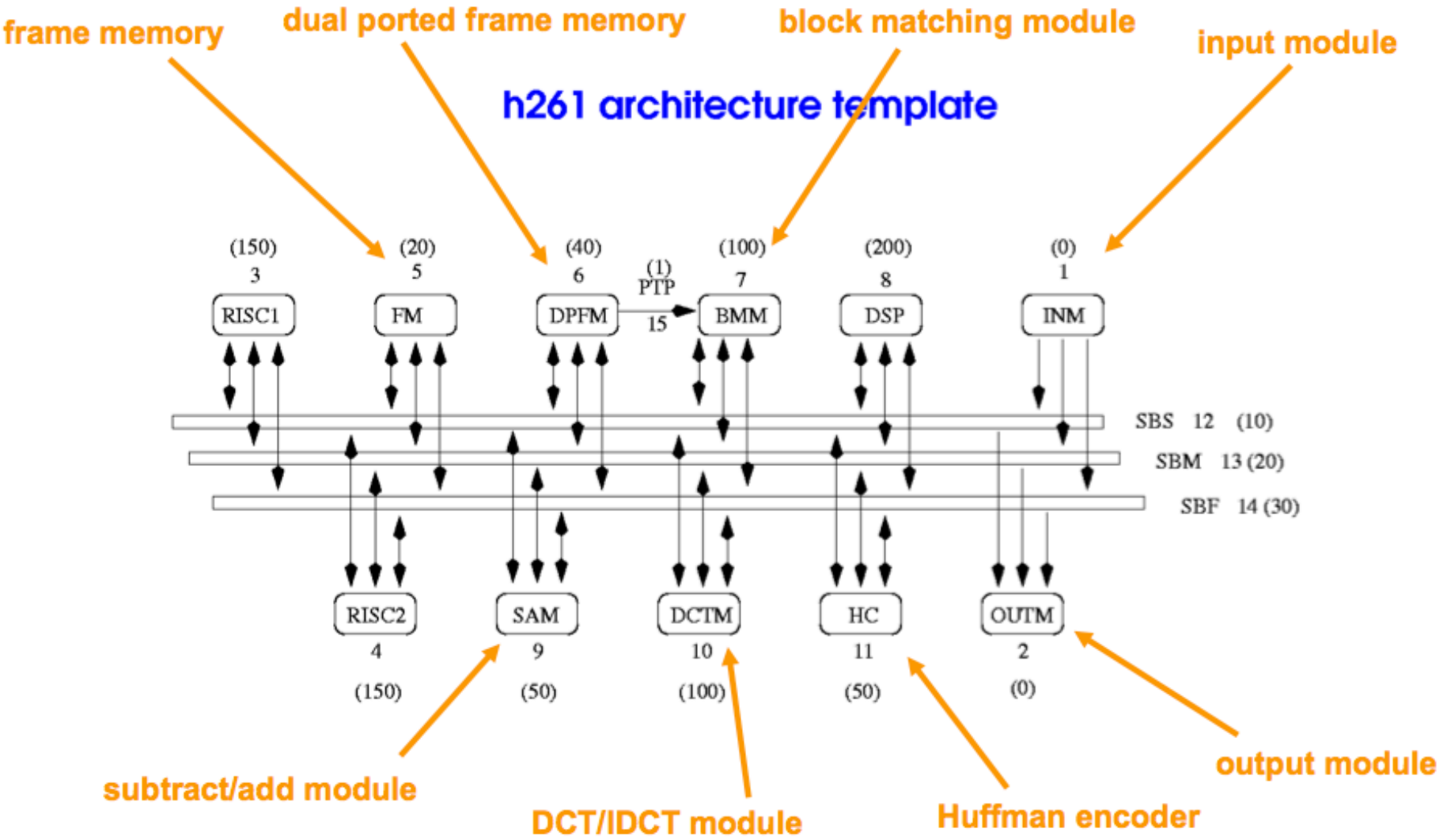


Beispiel 1: Spezifikation

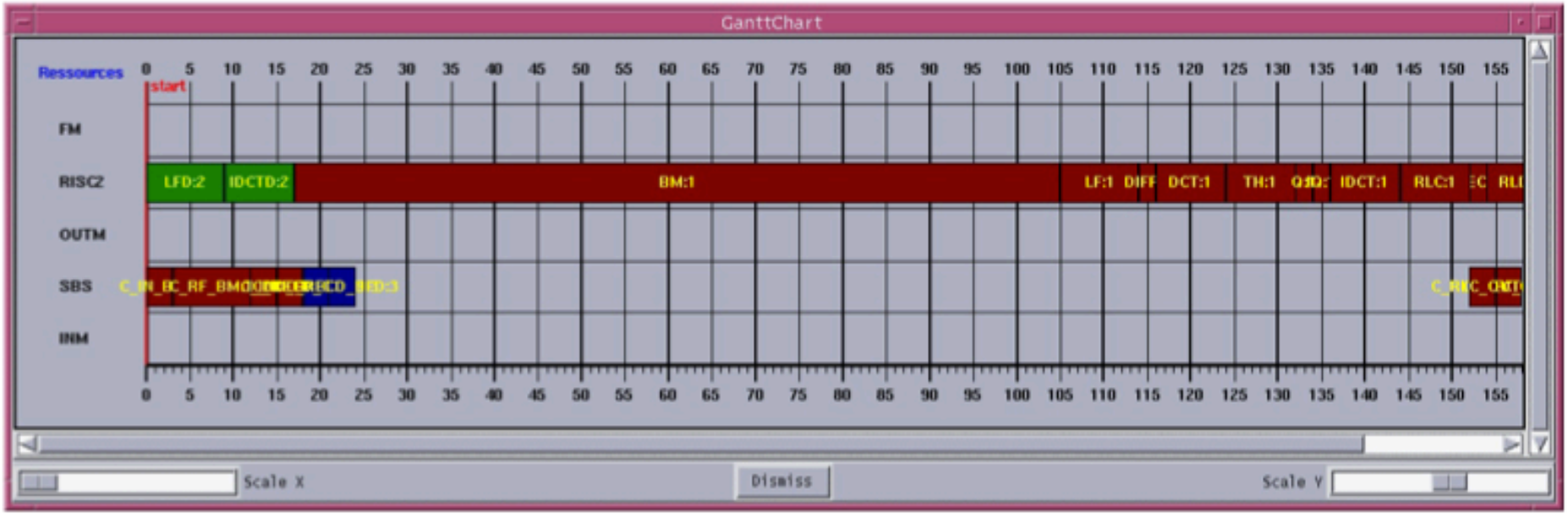
Problemgraph des Videocodierers

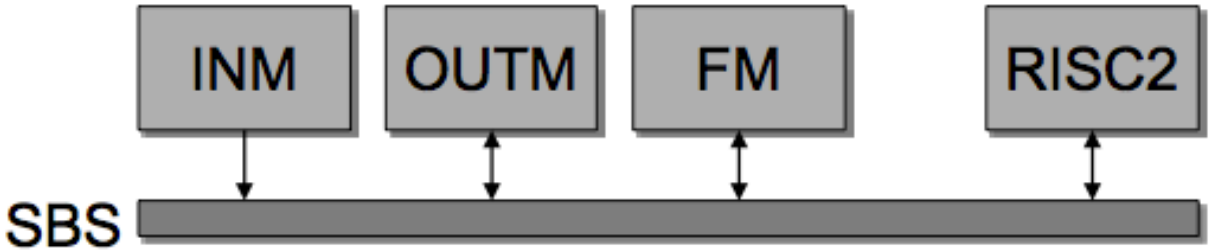


Beispiel 1: Spezifikation

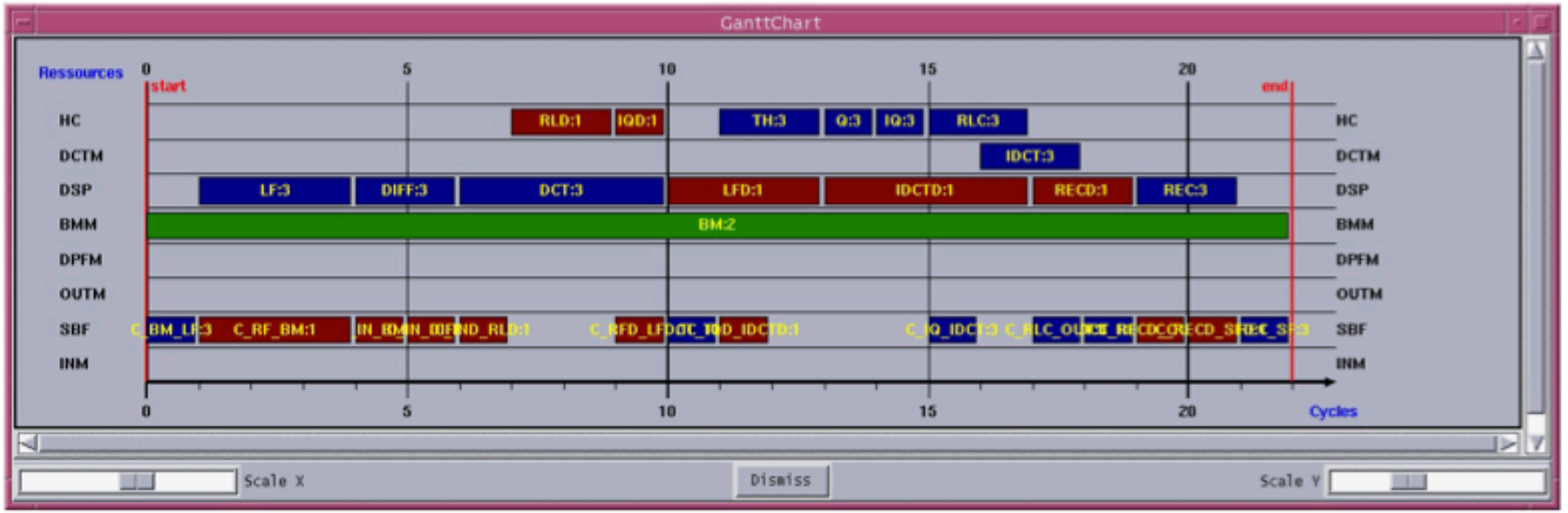


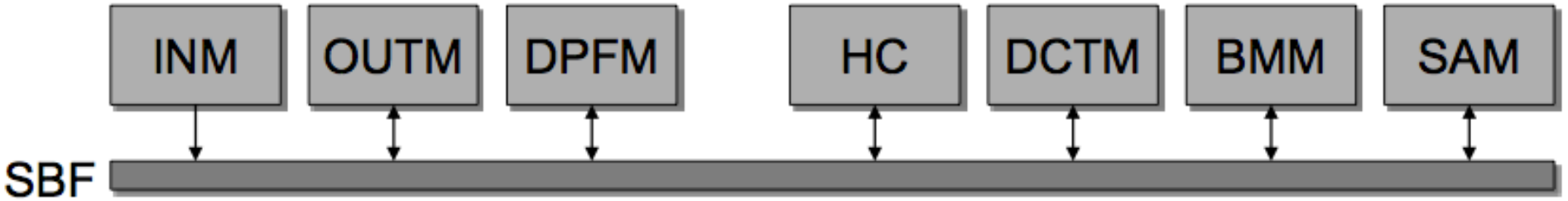
Beispiel 1: Lösung 1





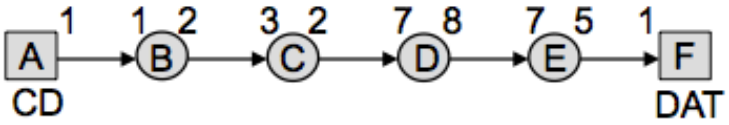
Beispiel 1: Lösung 2





Beispiel 2: Softwaresynthese

synchronous data flow graph



transformation

software implementation

Decisions:

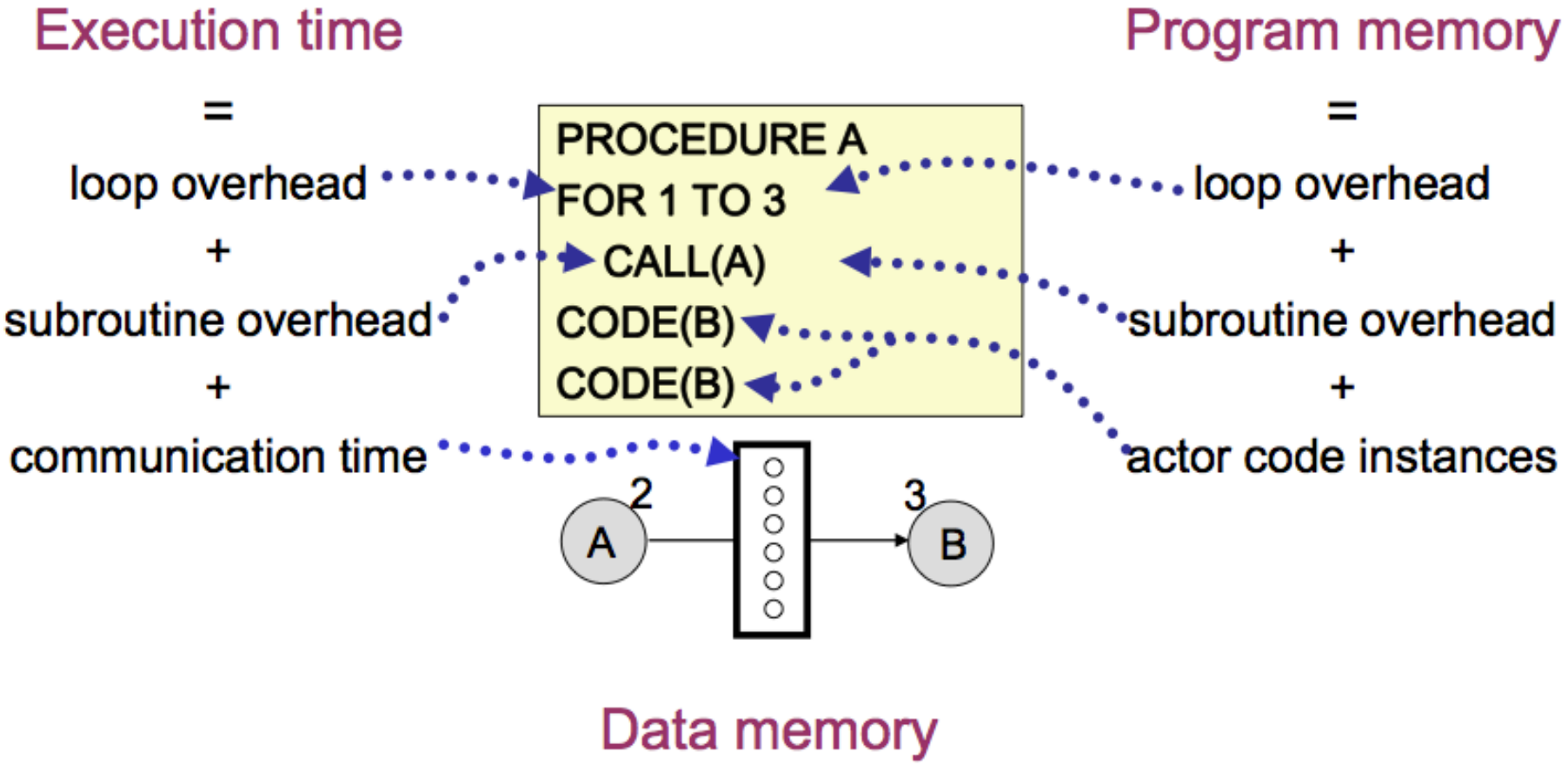
❶ Schedule

ABABABCCABABA…

❷ Code generation model

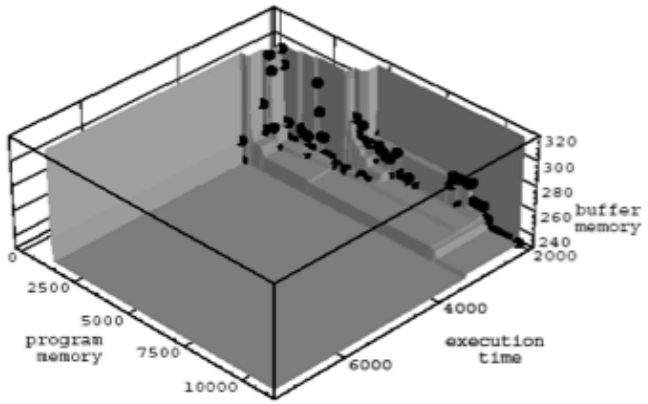
Inlining	subroutines	looping
CODE(A) CODE(B) CODE(A) CODE(B) CODE(C)	CALL(A) CALL(B) CALL(A) CALL(B) CALL(C)	FOR 1 TO 2 CODE(A) CALL(B) CODE(C) CODE(A)

Beispiel 2: Optimierungskriterien

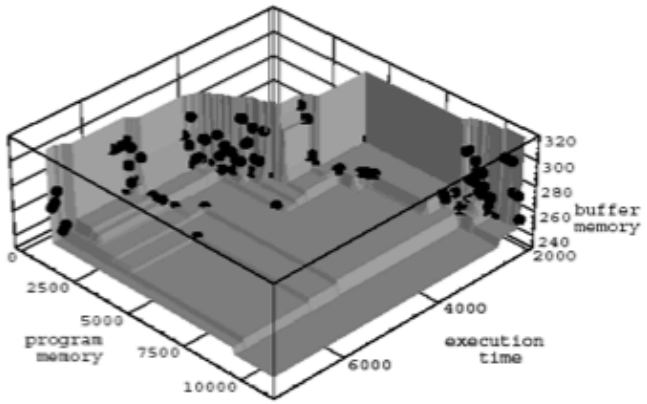


Beispiel 2: Trade-Offs im Pareto-Raum

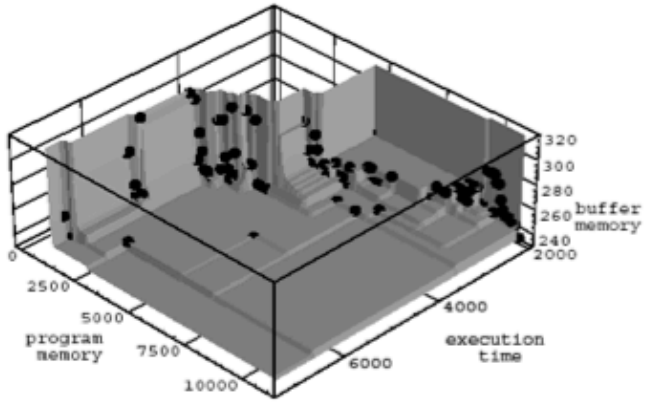
ADSP 2106x



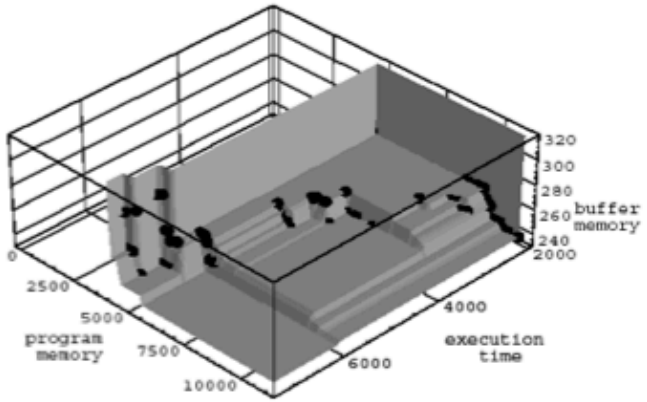
TI TMS320C40



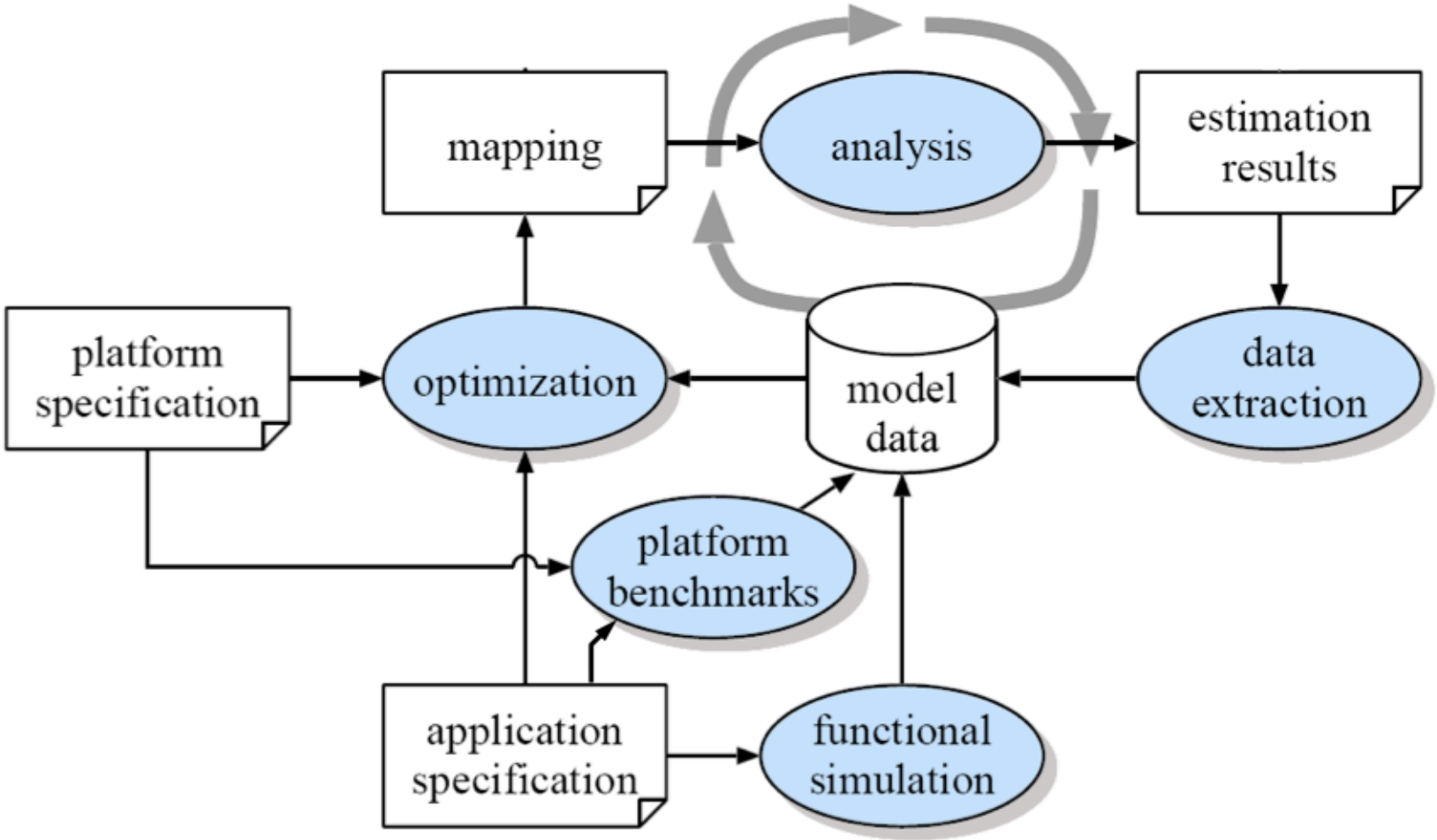
Motorola DSP56k



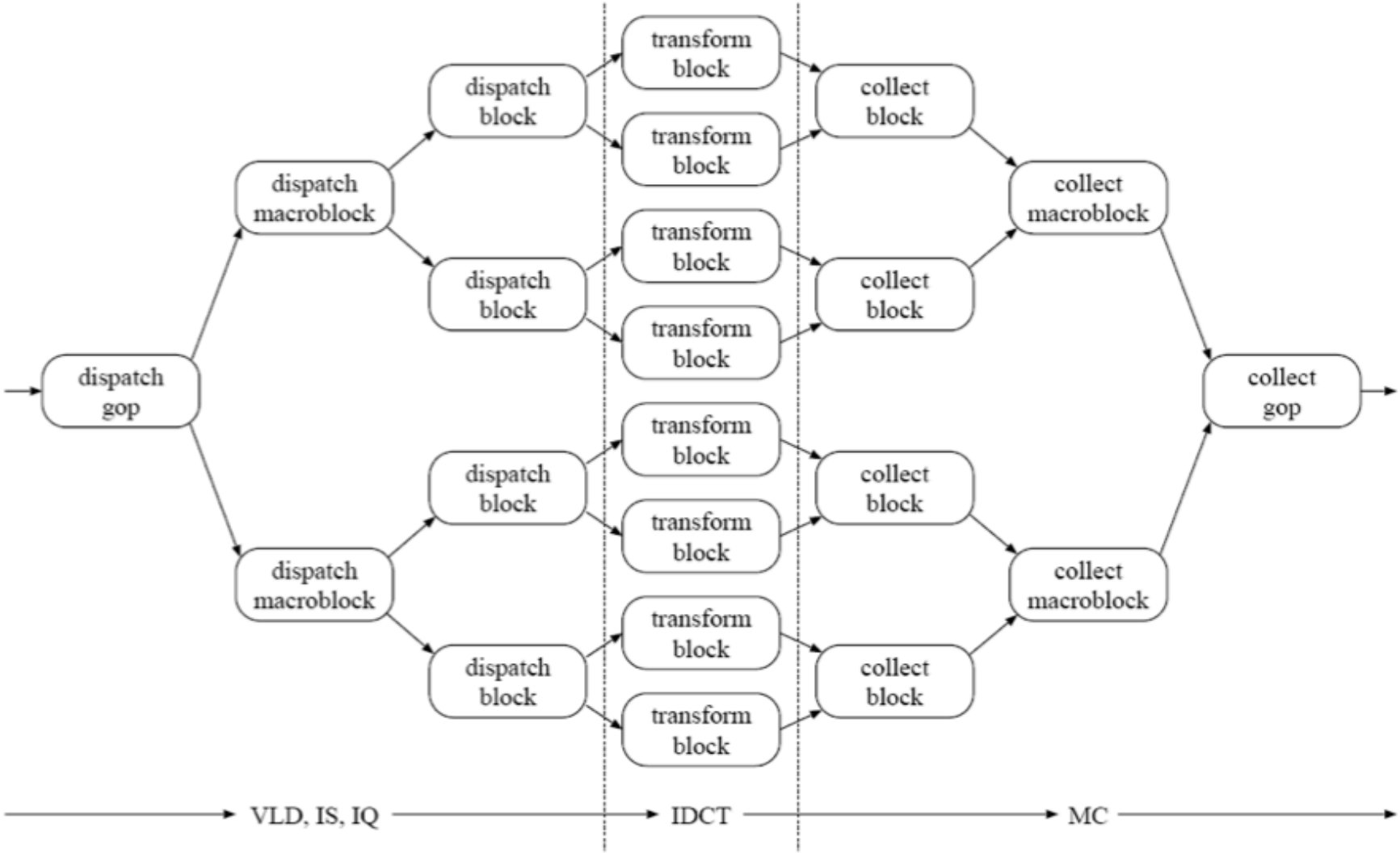
Hypothetical Processor



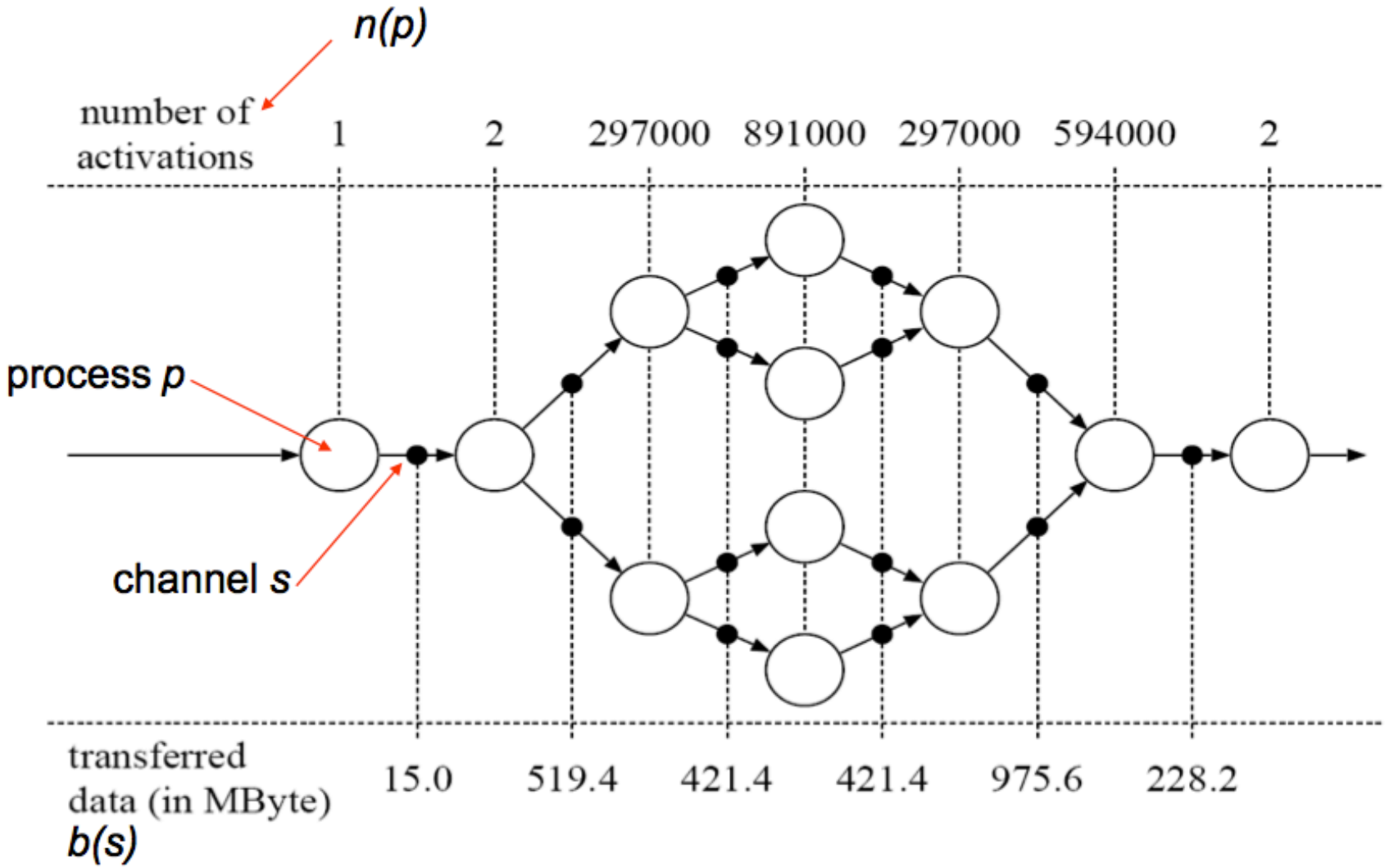
Beispiel 3: Erkundungsstrategie



Beispiel 3: KPN für Motion JPEG Decoder



Beispiel 3: Ergebnis der funktionalen Simulation



Beispiel 3: Ergebnis von Plattform-Benchmarks

- Plattform-Benchmarks
  - Kommunikationsbandbreite *t(g)* der Verbindung *g*
- Simulation einzelner Tasks

processor *c*

runtime *r(p, c)*

Process *p*	Runtime on RISC	Runtime on DSP
dispatch gop	0.13	0.20
dispatch macroblock	6.68	8.52
dispatch block	0.06	0.04
transform block	2.00	1.25
collect block	0.05	0.04
collect macroblock	12.33	8.51
collect gop	0.18	0.30

Beispiel 3: Überschlagsrechnung

processor c with worst total runtime

number of activations of process p

runtime of process p on processor c

$$obj_1 = \max_{c \in \mathcal{C}} \left\{ \sum_{\forall p \text{ mapped to } c} n(p) \cdot r(p,c) \right\}$$

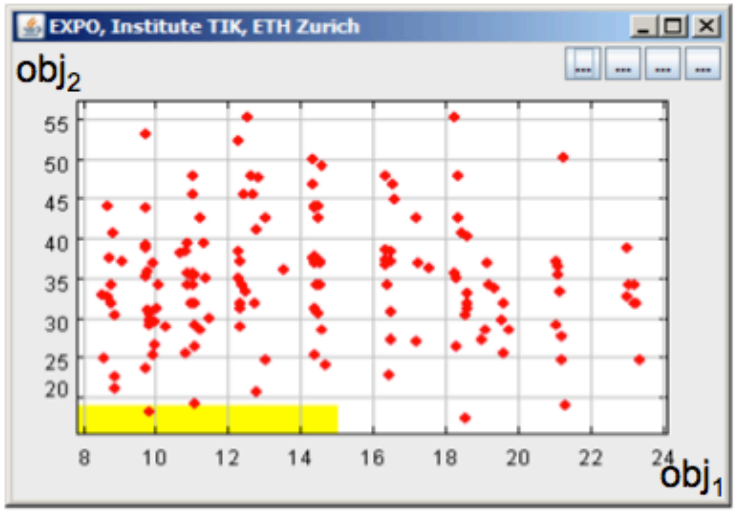
communication link with worst load

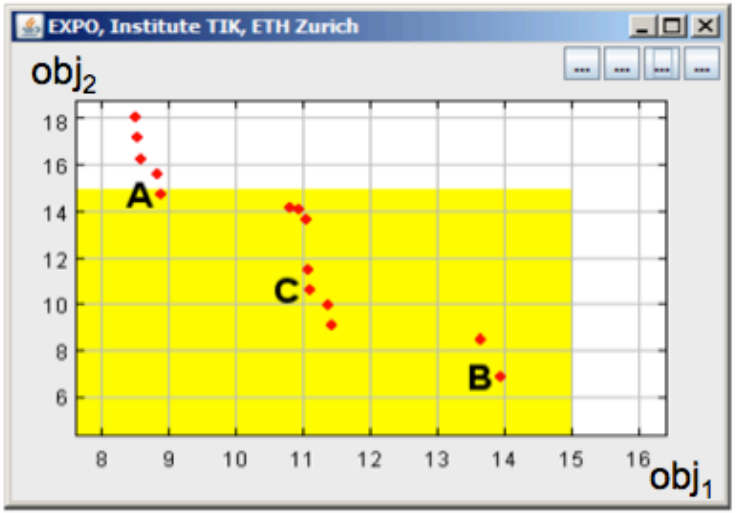
communication request from channel s

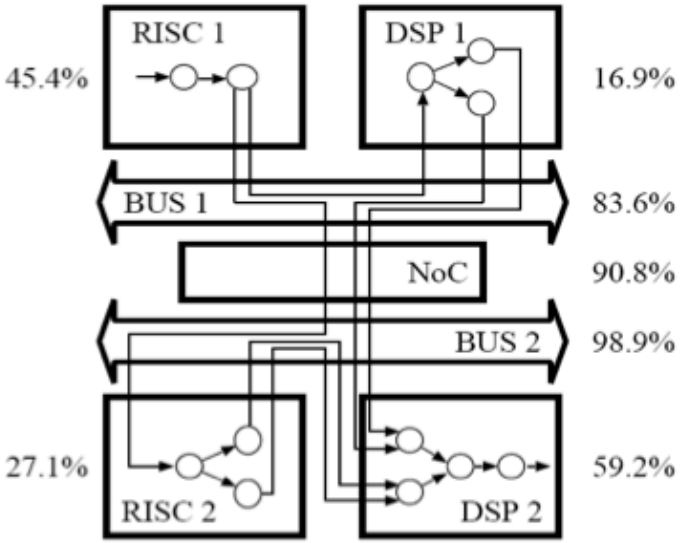
bandwidth of communication link g

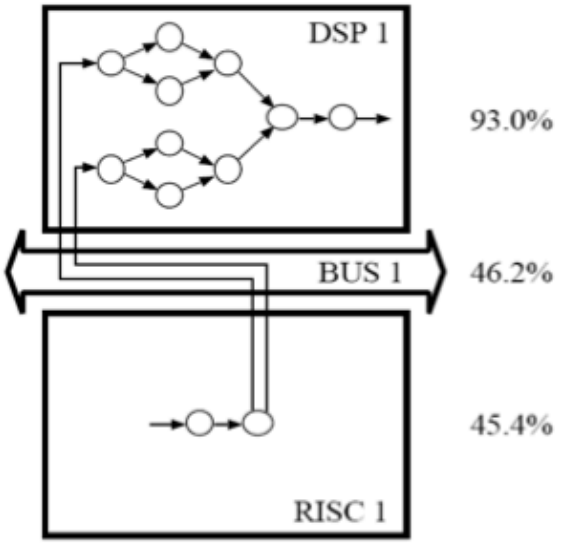
$$obj_2 = \max_{g \in \mathcal{G}} \left\{ \sum_{\forall s \text{ mapped onto } g} \frac{b(s)}{t(g)} \right\}$$

Beispiel 3: Ergebnis der Erkundung



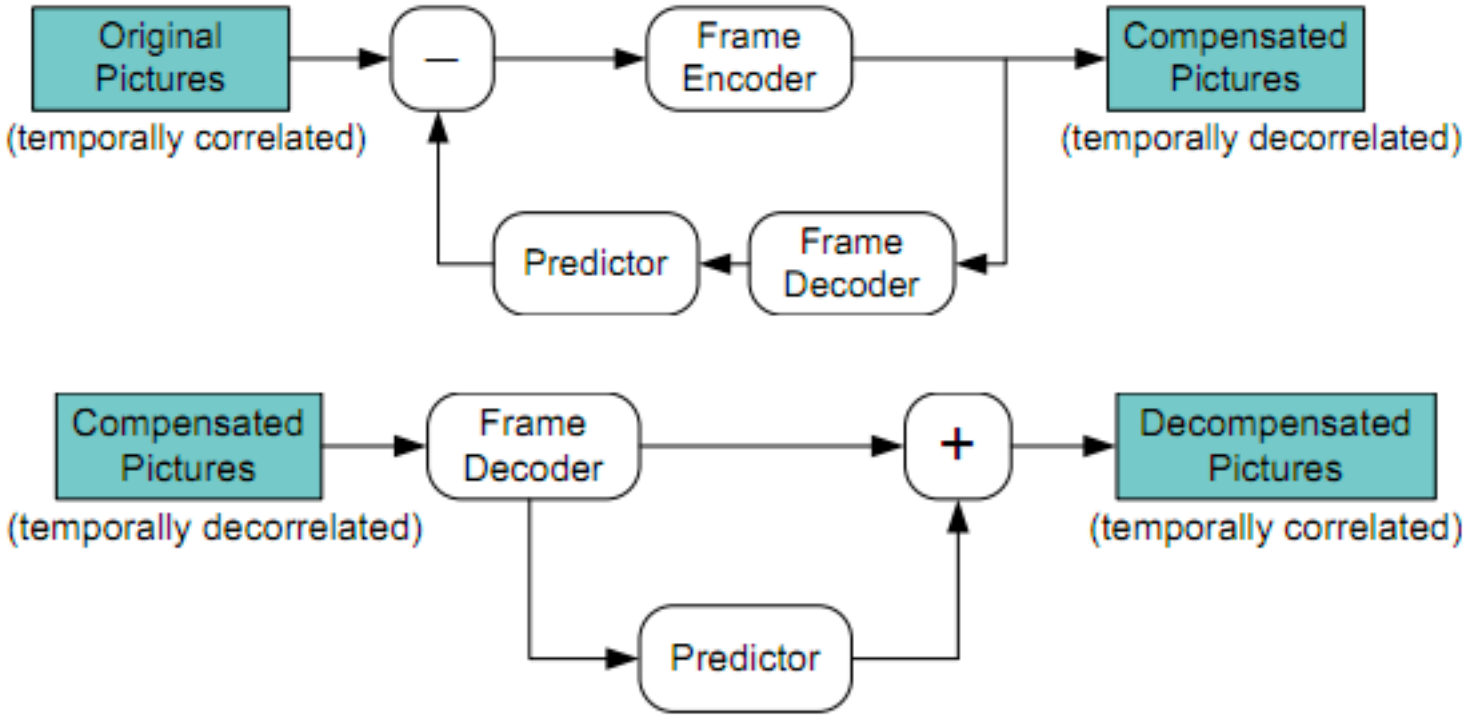






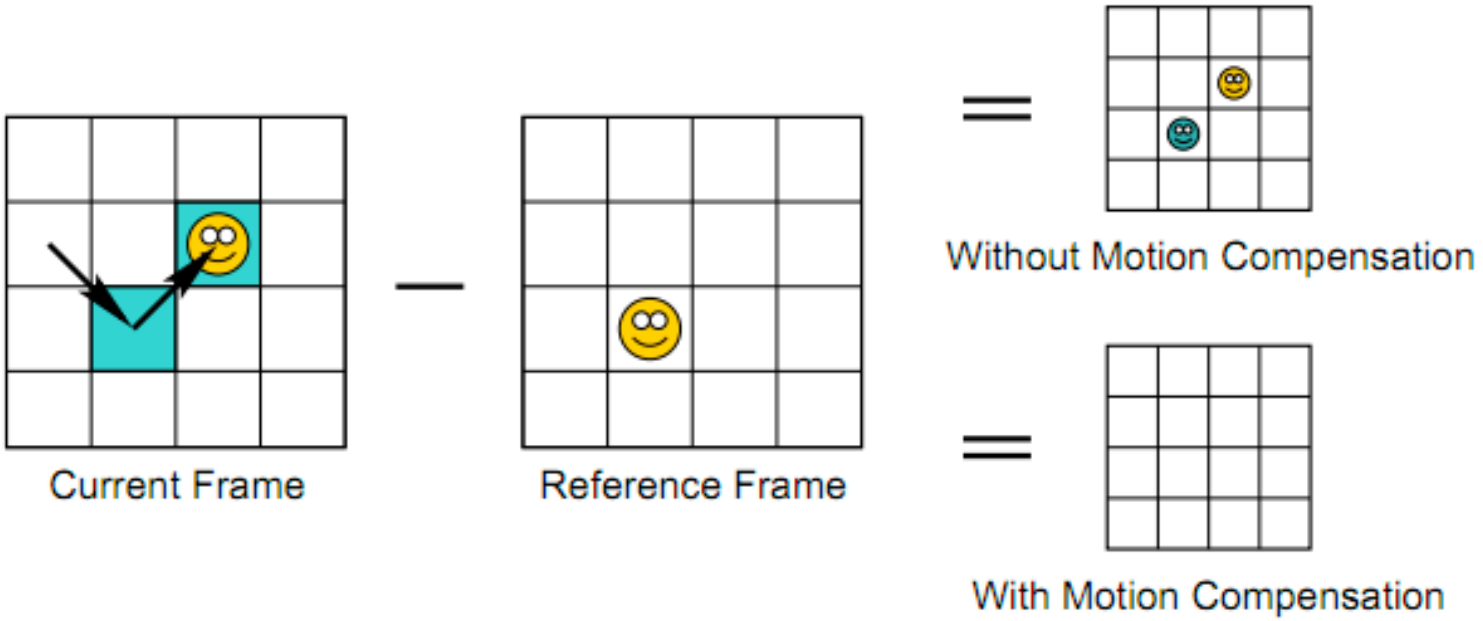
Beispiel 4: MPEG2-Decoder

- Verbreitetes Format für Videoinformationen
- Codierung und Decodierung:



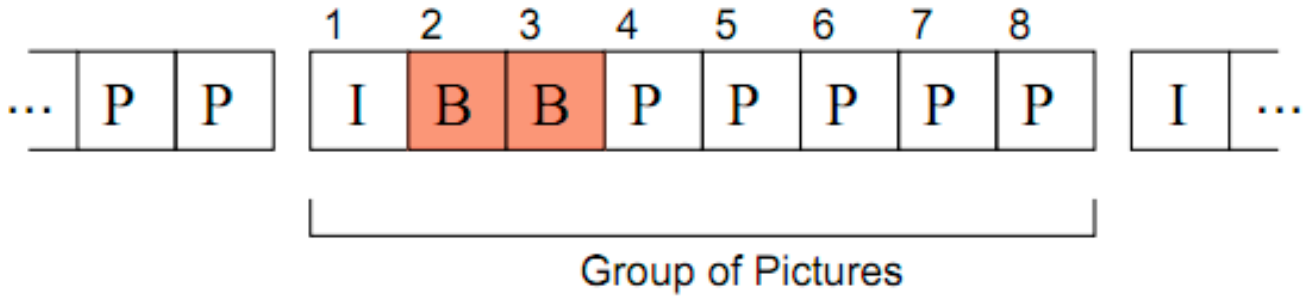
Beispiel 4: MPEG2-Decoder

- Ausnutzung temporaler Redundanzen (Folgeframes)
  - Wenig nützlich bei schnellen Bewegungen
- Ausnutzung von Bewegungsinformationen (motion compensation) zwischen Folgeframes



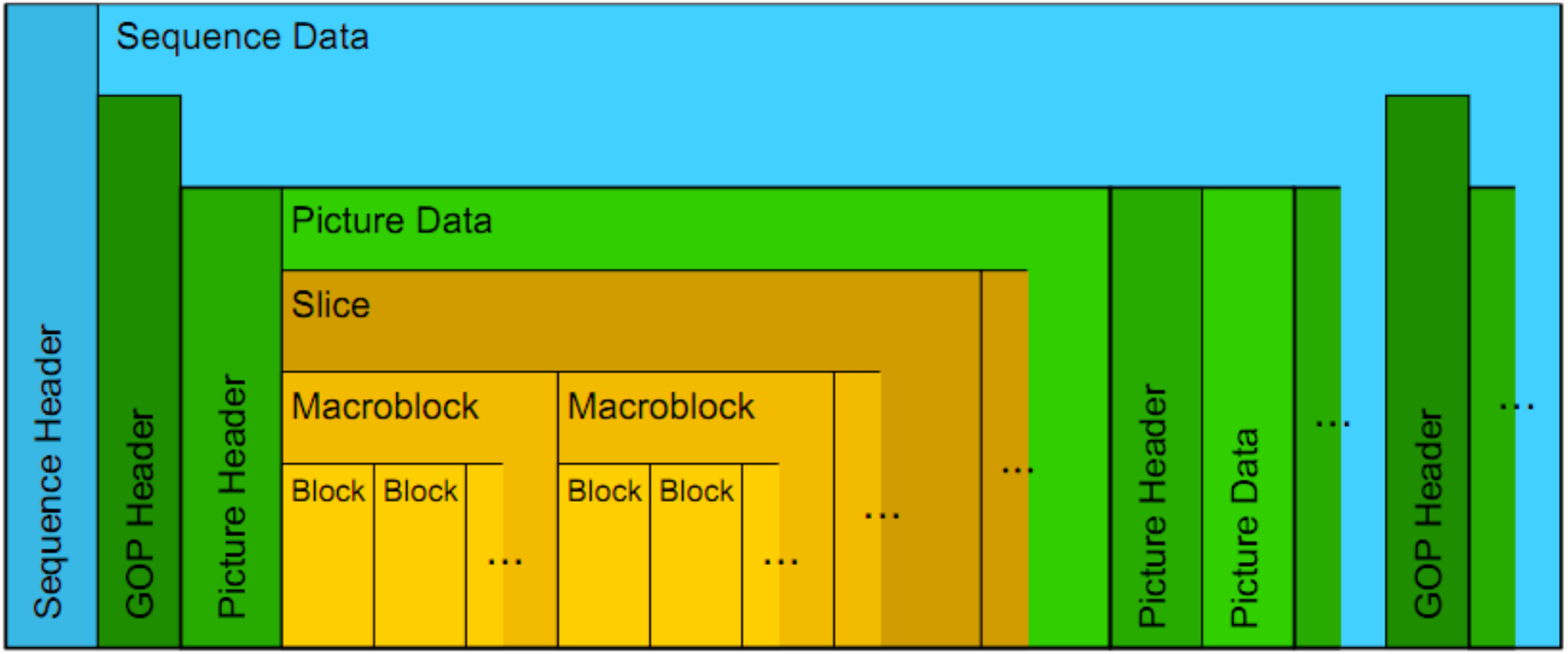
Beispiel 4: MPEG2-Decoder

- Verschiedene Frametypen
  - I-Frames: least compressible but don't require other video frames to decode
  - P-Frames: can use data from previous frames to decompress, more compressible than I-frames
  - B-Frames: can use both previous and forward frames for data reference to get the highest amount of data compression
- Group of Pictures (GOP): 1 I-Frame + n B-Frames + m P-Frames:



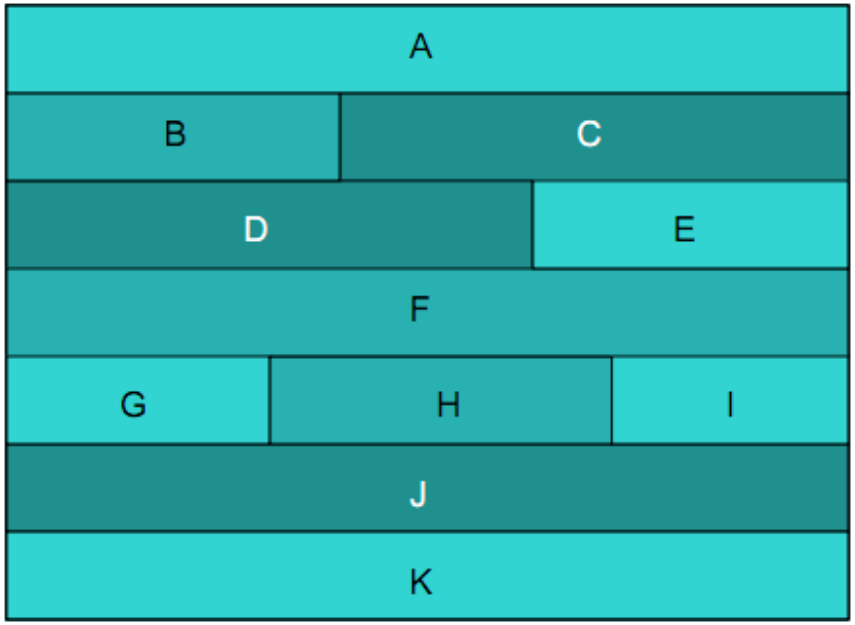
MPEG2-Decoder: Datenstruktur

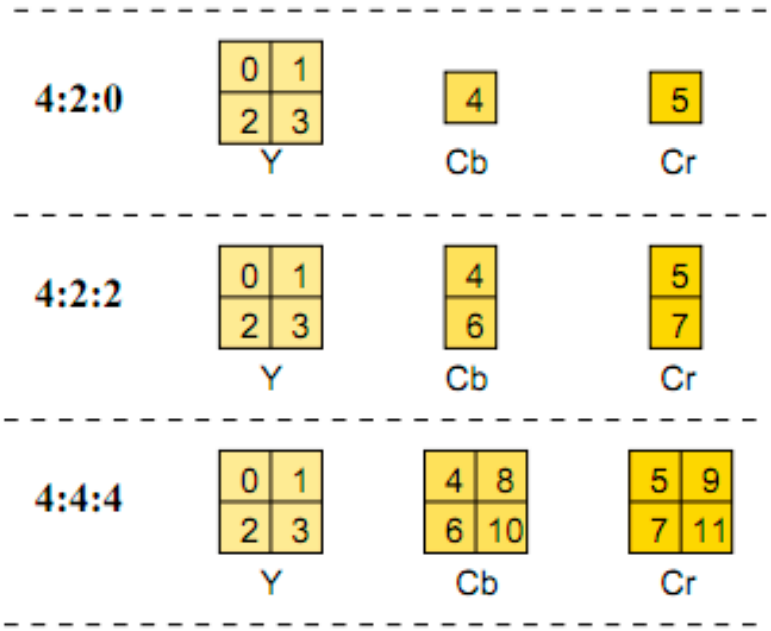
- Struktur des MPEG2-Bitstroms:



MPEG2-Decoder: Details des Datenstroms

- Slices: Menge von Makroblöcken einer Zeile
- Makroblock: Information für motion compensation





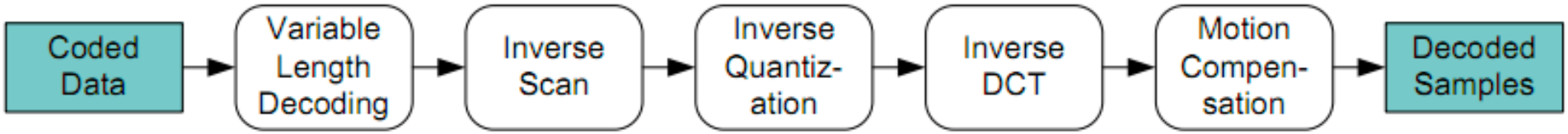
MPEG2-Decoder: Ablauf im Detail

Decodiervorgang: Frequenzbereich

- Konvertieren der Eingangsdaten (Lauflängencodiert) in Bewegungsvektoren und DCT-Koeffizienten
- Inverse Scan transformiert 1-dim. Strom in 2D-Array von Koeffizienten
- Inverse Quantization multipliziert Koeffizienten-Array mit Quantisierungsparameter und prüft Sättigung

Luminanz-/Chrominanzwerte

- Inverse DCT: Transformation Frequenz- zu Bildbereich
- Motion Compensation (aufwändigster Schritt)



MPEG2-Decoder: Dateistruktur des Quellcodes

Referenzimplementierung der MPEG Software Simulation Group (MSSG): mpeg2dec

Dateistruktur:

mpeg2dec.c	main() function, initialization, command-line option processing
getpic.c	picture decoding routines
motion.c	motion vector decoding routines
recon.c	motion compensation routines
gethdr.c	header decoding routines
getblk.c	DCT coefficient decoding routines
getbits.c	bit level routines
getvlc.c	variable length decoding routines
idct.c	fast inverse discrete cosine transform
idctref.c	double precision inverse discrete cosine transform
global.h	declaration of global variables

MPEG2-Decoder: Codestruktur

```
main() {
  <open input file>
  initialize_decoder();
  decode_bitstream() {
    for (all videosequences) {
      for (all pictures) {
        picture_data() {
          for (all slices) {
            for (all macroblocks) {
              <get macroblock mode>
              <decode motion vectors>
              <get macroblock pattern>
              for (all blocks) {
                <decode DCT coefficients>
              }

                                        motion_compensation() {
                                          <form predictions>
                                          for (all blocks) {
                                            <perform inverse DCT>
                                            <add prediction and coefficient data>
                                          }
                                        } /* end of motion_compensation() */
                                      }
                                    }
                                  } /* end of picture_data() */
                                  frame_reorder();
                                }
                              }
                            } /* end of decode_bitstream() */
                            <close input file>
                          }
```

MPEG2-Decoder: Parallelisierung

Generischer Ansatz:

- Partitionierung
- Kommunikationsanalyse
- Agglomeration (Granularitätskontrolle) und
- Abbildung

MPEG2-Decoder: Parallelisierung

Partitionierung

- Aufteilung von Kommunikation und Daten in kleinere Tasks
- *Domain decomposition:*
  - Erst Aufteilung der Daten, dann Zuweisung von Berechnungen
  - Ergibt Datenparallelität
- *Functional decomposition:*
  - Erst Aufteilung der Berechnung, dann Zuweisung der zugehörigen Daten
  - Ergibt Pipeline-Parallelität

MPEG2-Decoder: Parallelisierung

Kommunikationsanalyse

- Nach der Aufteilung muss die Kommunikation zwischen den einzelnen Tasks analysiert werden
- Zugehörige Datenstrukturen werden definiert

Agglomerationsphase

- Evaluation des Entwurfs
- Kombinierung von Tasks, wenn dies Leistung steigert oder Kosten senkt

Abbildung

- Abbildung von Tasks auf Prozessoren
- Maximierung der Prozessorauslastung
- Minimierung der Kommunikationskosten

MPEG2-Decoder: Parallelisierung

Modifikationen für DOL-basierte Parallelisierung des MPEG2-Decoders

- Manuelle Abbildungsstufe fällt weg: Aufgabe von DOL
- Komplexität und Verschachtelung des MPEG2-Decoders erfordert verbesserten Parallelisierungsansatz
  - Aufteilung des Algorithmus in *Abstraktionsebenen*: Dateneinheit, Recheneinheit und erzeugtes Berechnungsergebnis
  - Untersuchung von *domain decomposition* und *functional decomposition* für jede Abstraktionsebene
  - Evaluation durch Kommunikationsanalyse
    - Je weniger Inter-Prozess-Kommunikation, desto besser

MPEG2-Decoder: Abstraktionsebenen

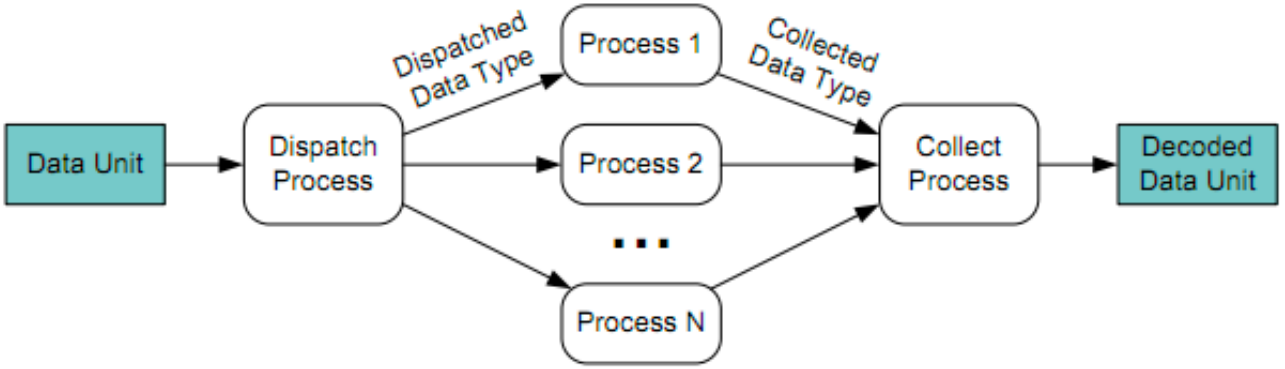
1	**System Level**
	Data unit: MPEG-2 bitstream
	Demultiplex and decode video, audio and data stream
	Decoded data unit: stream of decoded pictures, audio samples and data
2	**Videosequence Level**
	Data unit: Coded video sequence bitstream
	Extract header information, decode GOPs (if available) and pictures
	Decoded data unit: Decoded videosequence (a number of decoded pictures)
3	**Picture Level**
	Data unit: Coded picture bitstream
	Extract header information and decode slices
	Decoded data unit: Decoded picture (raw 8-bit RGB or YUV data)

MPEG2-Decoder: Abstraktionsebenen

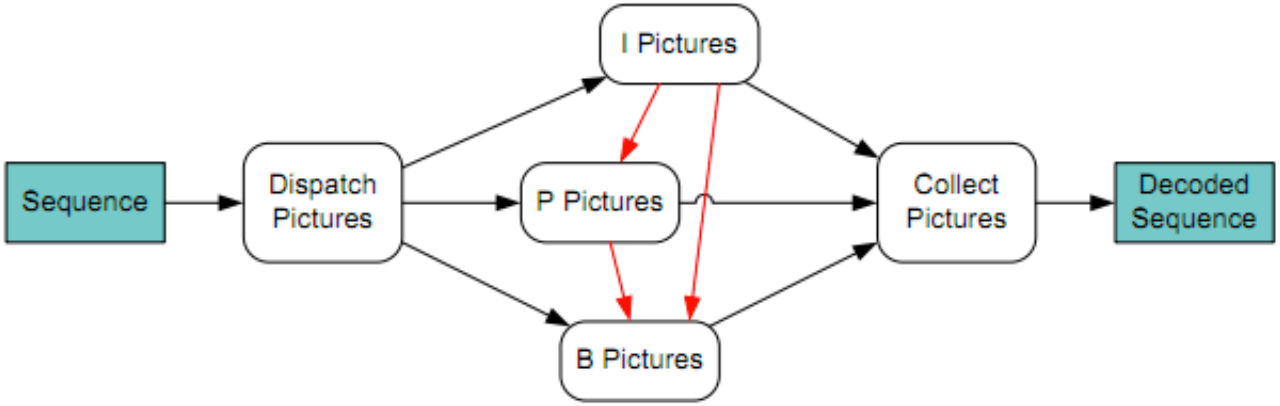
4	**Slice Level**
	Data unit: Coded slice bitstream
	Extract header information and decode macroblocks
	Decoded data unit: Decoded slice (one 16 pels wide row of a picture)
5	**Macroblock Level**
	Data unit: Coded macroblock bitstream
	Variable length decoding, block decoding and motion compensation
	Decoded data unit: Decoded MB (a 16×16 pels large section of a picture)
6	**Block Level**
	Data unit: Coded block bitstream (decoded DCT coefficients)
	Inverse Scan, Inverse Quantization, Inverse DCT
	Decoded data unit: Decoded block (8×8 chrominance or luminance values)

MPEG2-Decoder: Domain decomposition

- Genereller Ablauf der *domain decomposition*:



- Realisierung auf Bildebene:

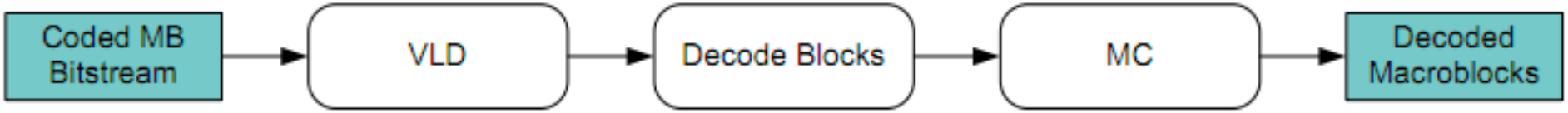


MPEG2-Decoder: Domain decomposition

Makroblock-Ebene:

- Domain decomposition
  - Anzahl der Blöcke in Makroblock bekannt
  - Abhängig von Formatinformation im Header: 6, 8, 12
- Functional decomposition
  - Berechnung in drei Stufen: variable length decoding (VLD) , block decoding und motion compensation (MC)

- Resultierende Pipeline auf Makroblock-Ebene:

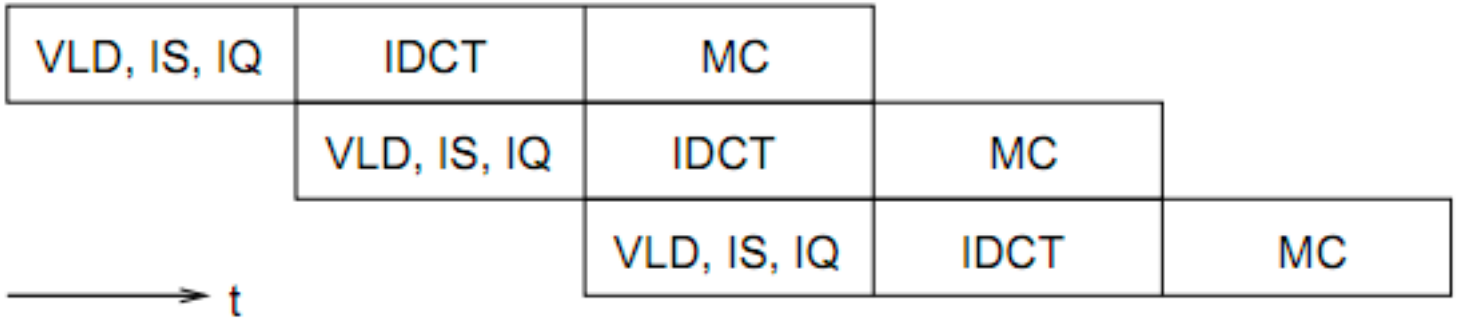


MPEG2-Decoder: Domain decomposition

Gewählte domain decompositions:

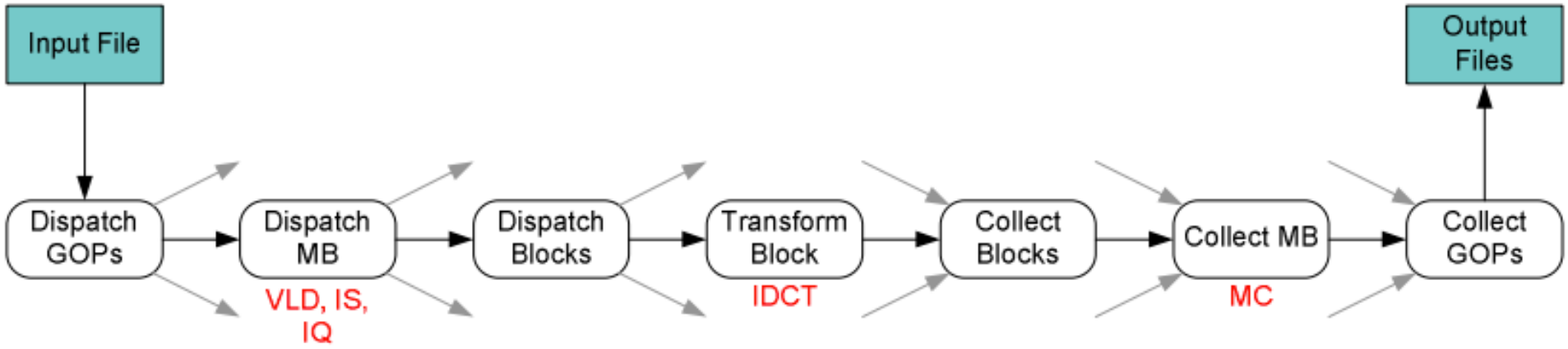
1	**GOP**	N_1 groups of pictures are processed in parallel
2	**Macroblock**	N_2 macroblocks are processed in parallel
3	**Block**	N_3 blocks are processed in parallel

Struktur der Pipeline:



MPEG2-Decoder: Prozessnetzwerk

Resultierendes Prozessnetzwerk:



MPEG2-Decoder: Prozessnetzwerk

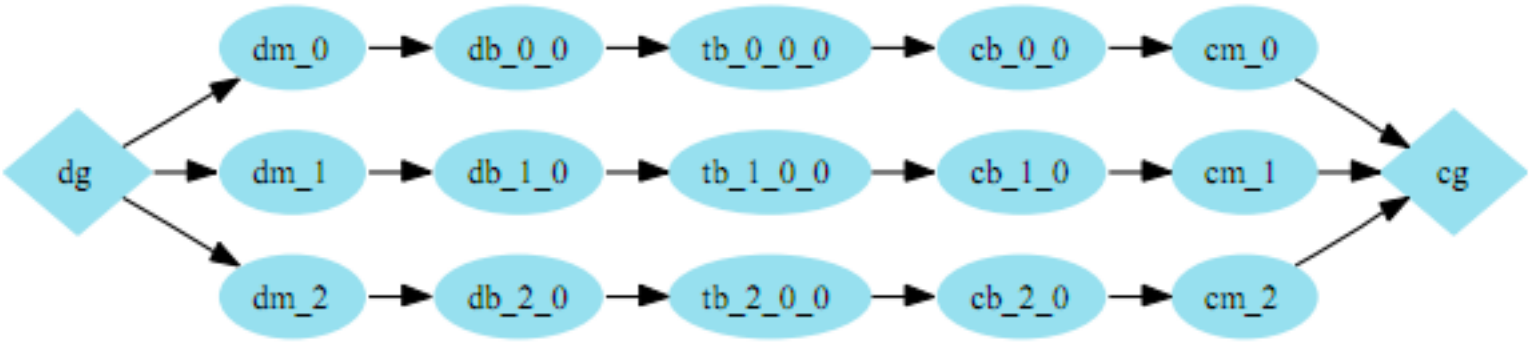
Ausschnitt der XML-Definition des Prozessnetzwerks:

```
<!-- N1 is the number of GOPs processed in parallel -->
<variable name="N1" value="2" />

<!-- N2 is the number of macroblocks processed in parallel -->
<variable name="N2" value="2" />

<!-- N3 is the number of blocks processed in parallel -->
<variable name="N3" value="2" />

<!-- instantiate processes -->
<process name="dispatch_gops">
  <iterator variable="i" range="N1">
    <port type="output" name="out">
      <append function="i" />
    </port>
  </iterator>
  <source type="c" location="dispatch_gops.c" />
</process>
(...)
```

MPEG2-Decoder: Parameter

Parametrisierung

1	**GOP**	N_1 groups of pictures are processed in parallel
2	**Macroblock**	N_2 macroblocks are processed in parallel
3	**Block**	N_3 blocks are processed in parallel

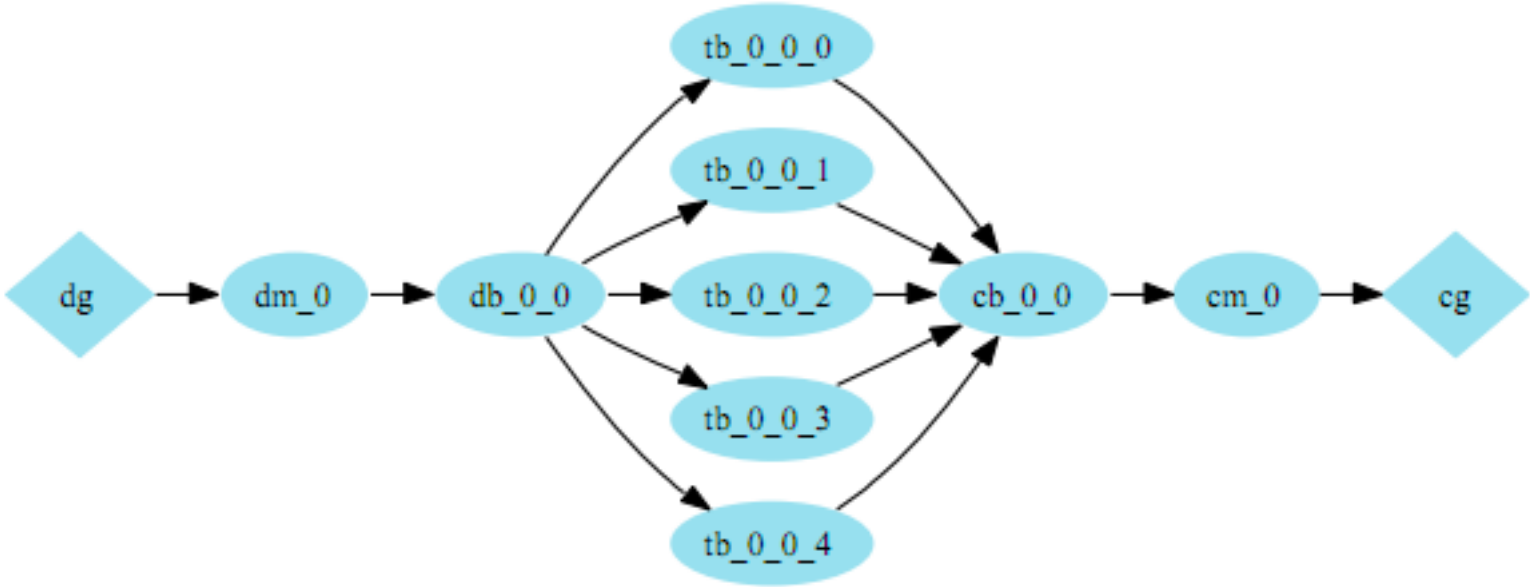


a) $N_1 = 3,\ N_2 = 1,\ N_3 = 1$

MPEG2-Decoder: Parameter

Parametrisierung

1	**GOP**	N_1 groups of pictures are processed in parallel
2	**Macroblock**	N_2 macroblocks are processed in parallel
3	**Block**	N_3 blocks are processed in parallel

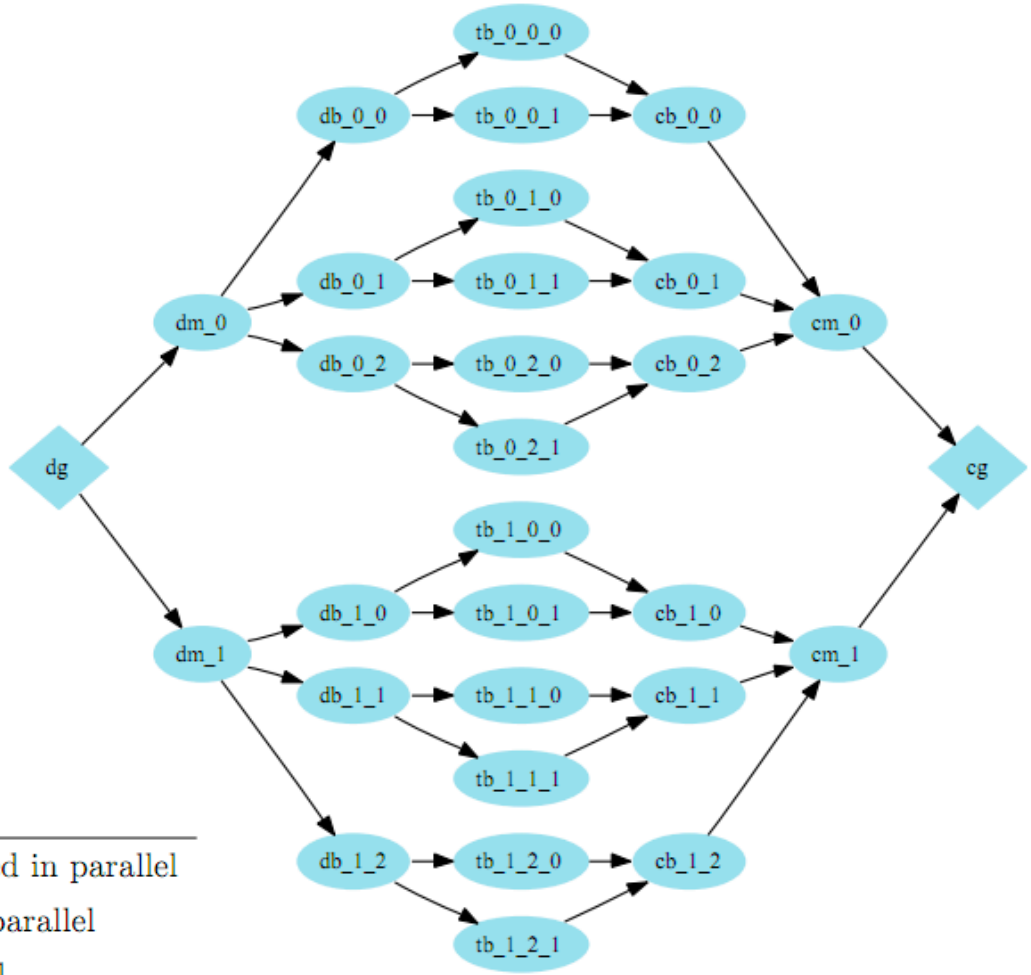


b) $N_1 = 1,\ N_2 = 1,\ N_3 = 5$

MPEG2-Decoder: Parameter

Parametrisierung

1	**GOP**	N_1 groups of pictures are processed in parallel
2	**Macroblock**	N_2 macroblocks are processed in parallel
3	**Block**	N_3 blocks are processed in parallel



c) $N_1 = 2,\ N_2 = 3,\ N_3 = 2$

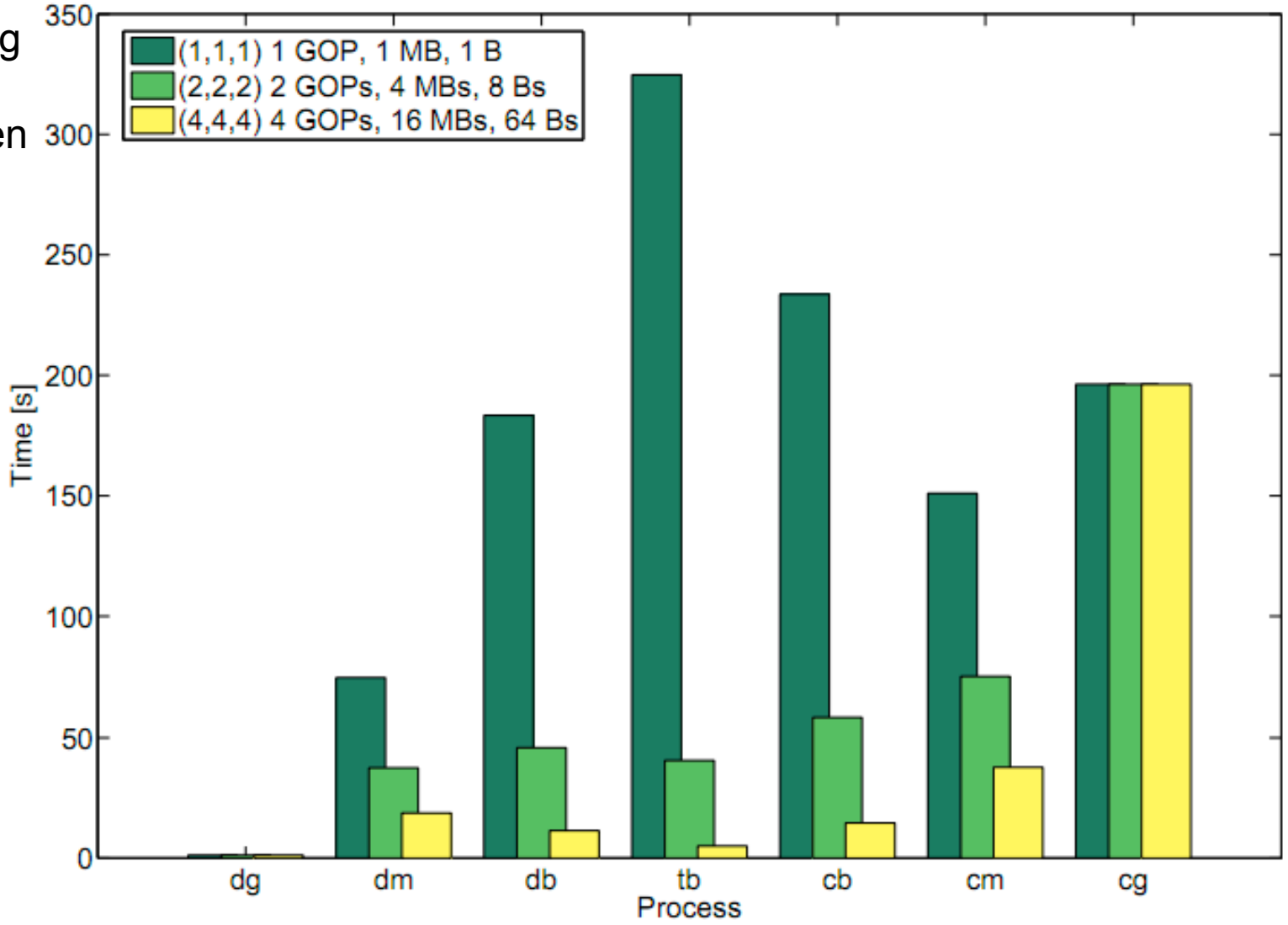
MPEG2-Decoder: Profiling

- total ms per call: time spent in the fire function, including all sub–functions
- calls: total number of calls
- Laufzeit: ms per call * calls

```
cumulative    self              self    total
  seconds   seconds    calls  ms/call  ms/call  name
  (...)
  2455.78     0.48  1910212     0.00     0.17  transform_block_fire__FP8_process
  2457.83     0.28   359487     0.00     0.42  collect_mb_fire__FP8_process
  2458.98     0.21   359487     0.00     0.65  collect_blocks_fire__FP8_process
  2461.92     0.10   359487     0.00     0.51  dispatch_blocks_fire__FP8_process
  2464.10     0.00       30     0.00    41.58  dispatch_gops_fire__FP8_process
  2464.10     0.00       30     0.00  2500.27  dispatch_mb_fire__FP8_process
  2464.10     0.00       27     0.00  7269.91  collect_gops_fire__FP8_process
  (...)
```

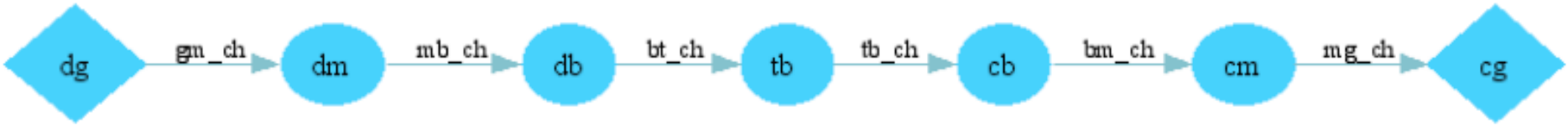
MPEG2-Decoder: Profiling

Parallelisierung für drei Konfigurationen (N1, N2, N3)



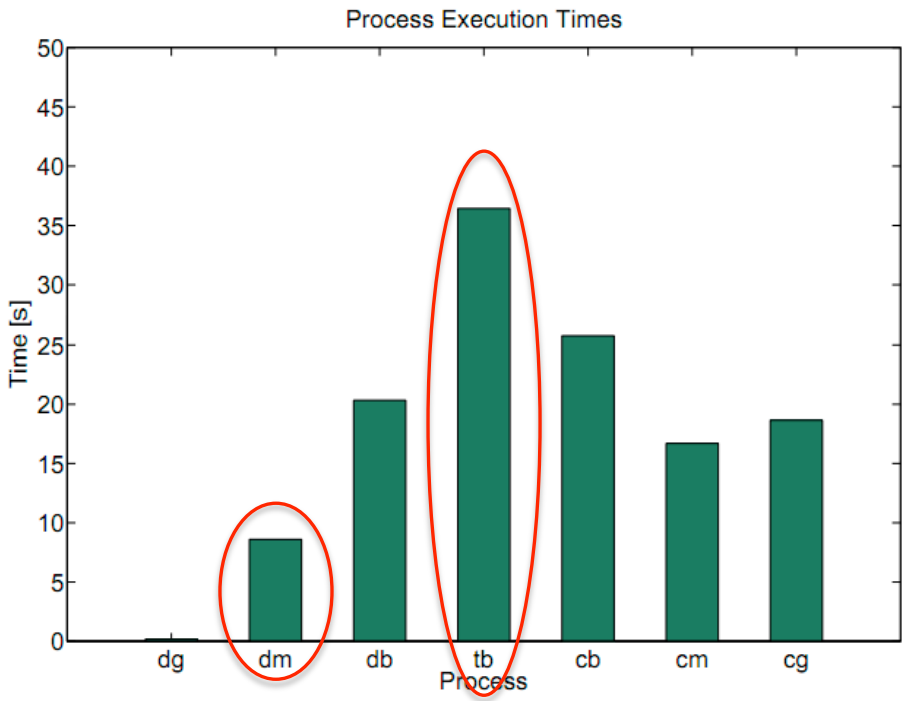
MPEG2-Decoder: Ergebnisse

- Beispielarchitektur: 2 RISC-CPUs, 2 DSPs
- Konfiguration (1, 1, 1)
- Pipeline:



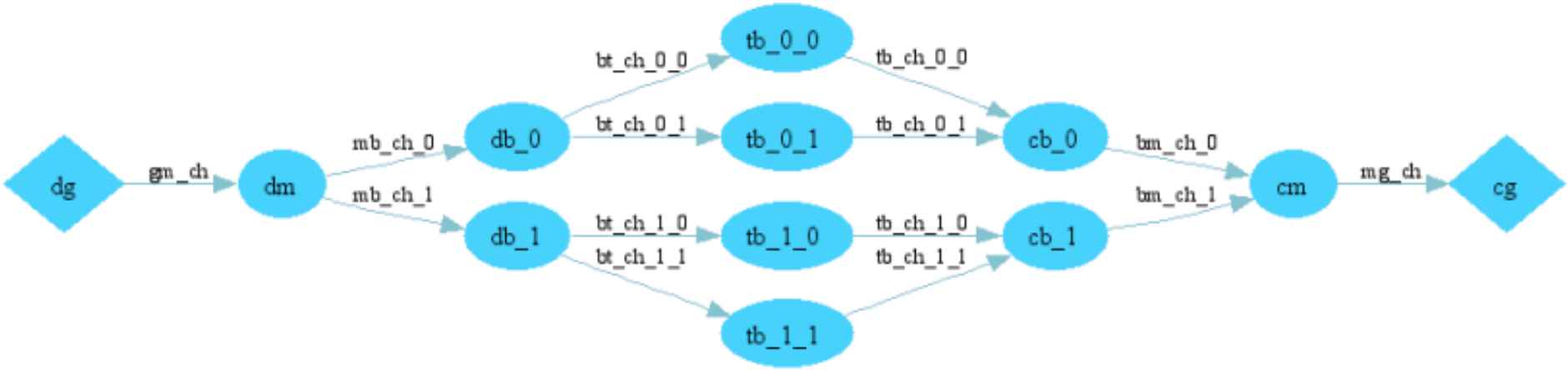
Profiling:

- Schlecht balancierte Pipeline
- Große Variation der Ausführungszeiten:
  t(tb) ≈ 2* t(db) ≈ 4*t(dm)



MPEG2-Decoder: Ergebnisse

- Balancierung der Pipeline:
  - 2 dispatch_block-, 4 transform_block-Tasks: (2, 2, 2)

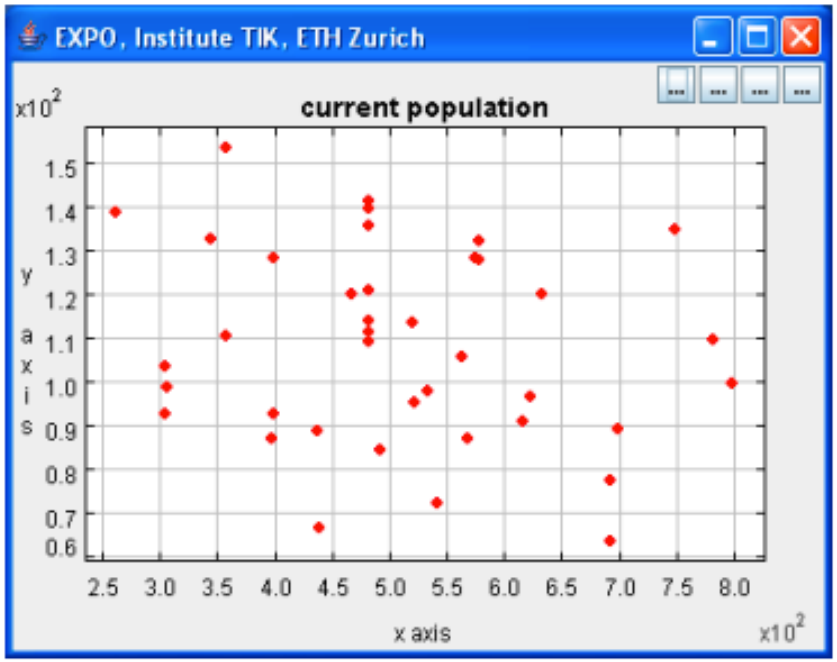


MPEG2-Decoder: Ergebnisse

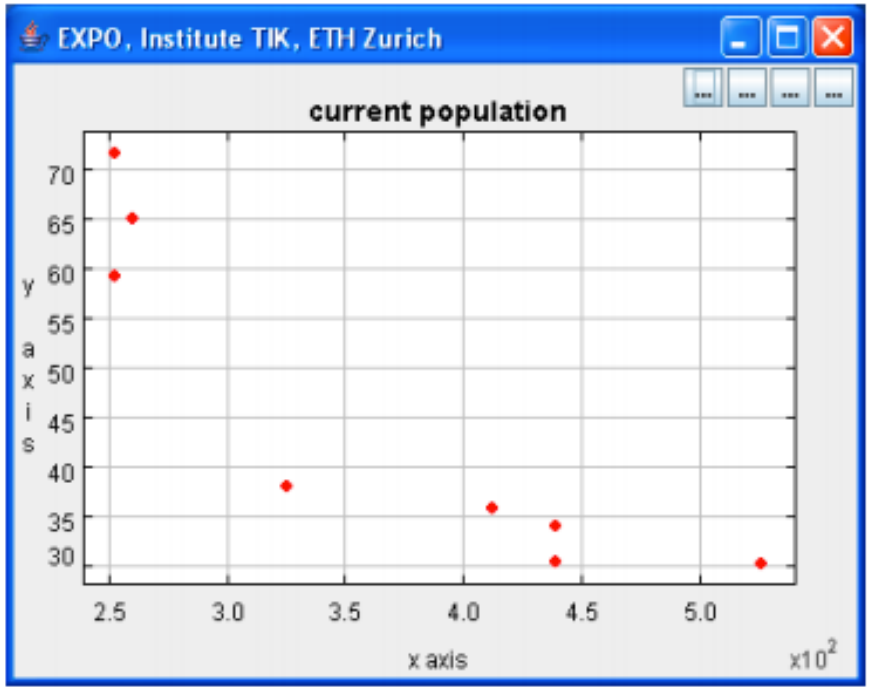
- Profiling für (1, 1, 1) und (2, 2, 2):

MPEG2-Decoder: Multikriterielle Optimierung

- Rechenlast (x) und Kommunikationsaufwand (y)



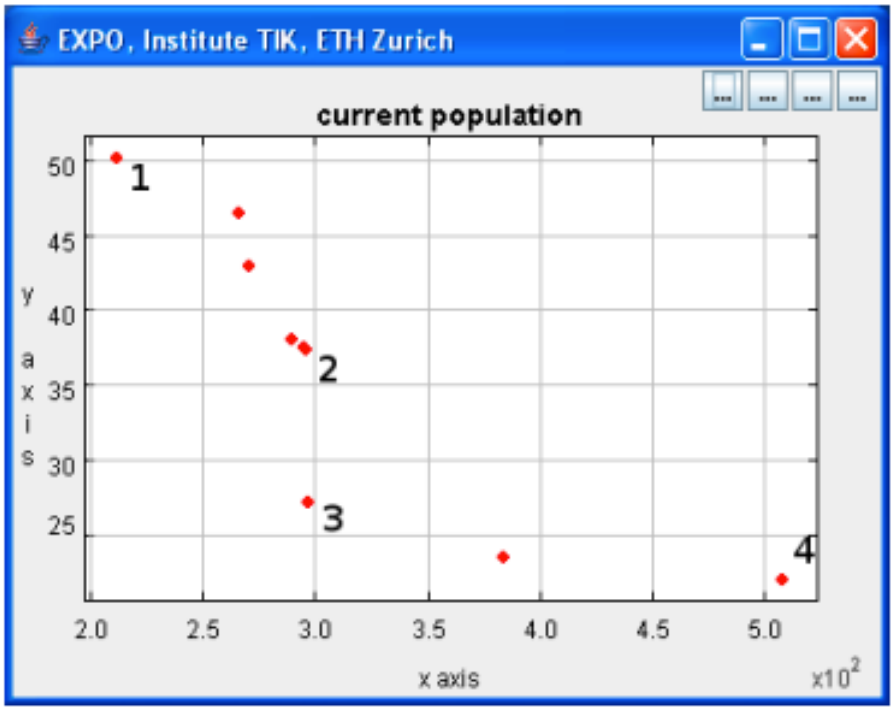
(a) Initial population



(b) Generation 20

MPEG2-Decoder: Multikriterielle Optimierung

- Rechenlast (x) und Kommunikationsaufwand (y):
  - 4 Pareto-optimale Lösungen in Generation 200 des evolutionären Algorithmus



(c) Generation 200

Zusammenfassung

- Entwurfsraumerkundung
- Beispiele