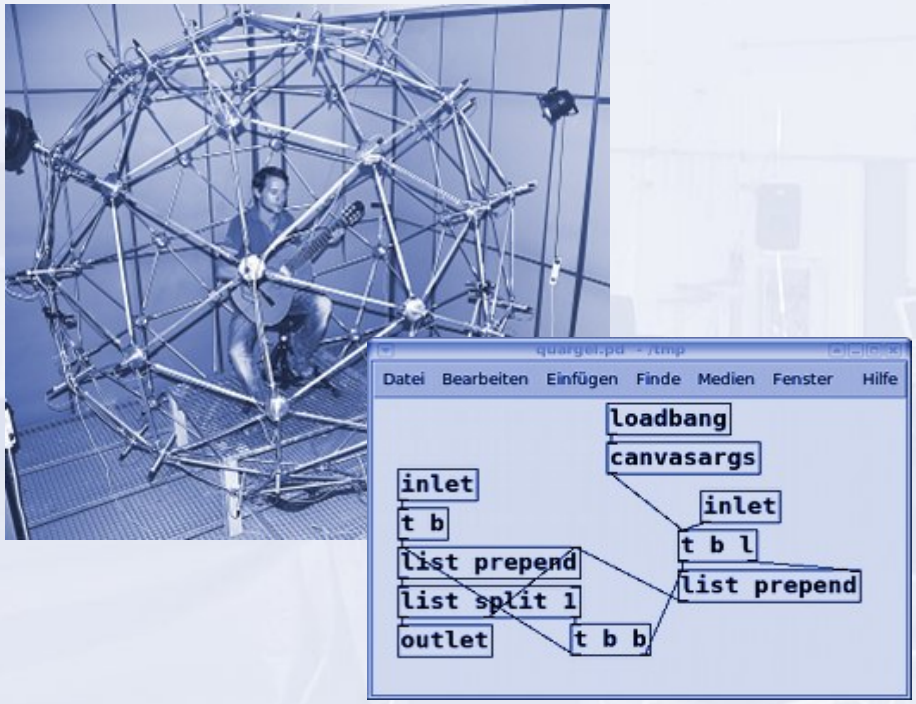
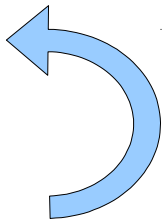
Implementierung von Algorithmen
Pure Data: Signal Processing



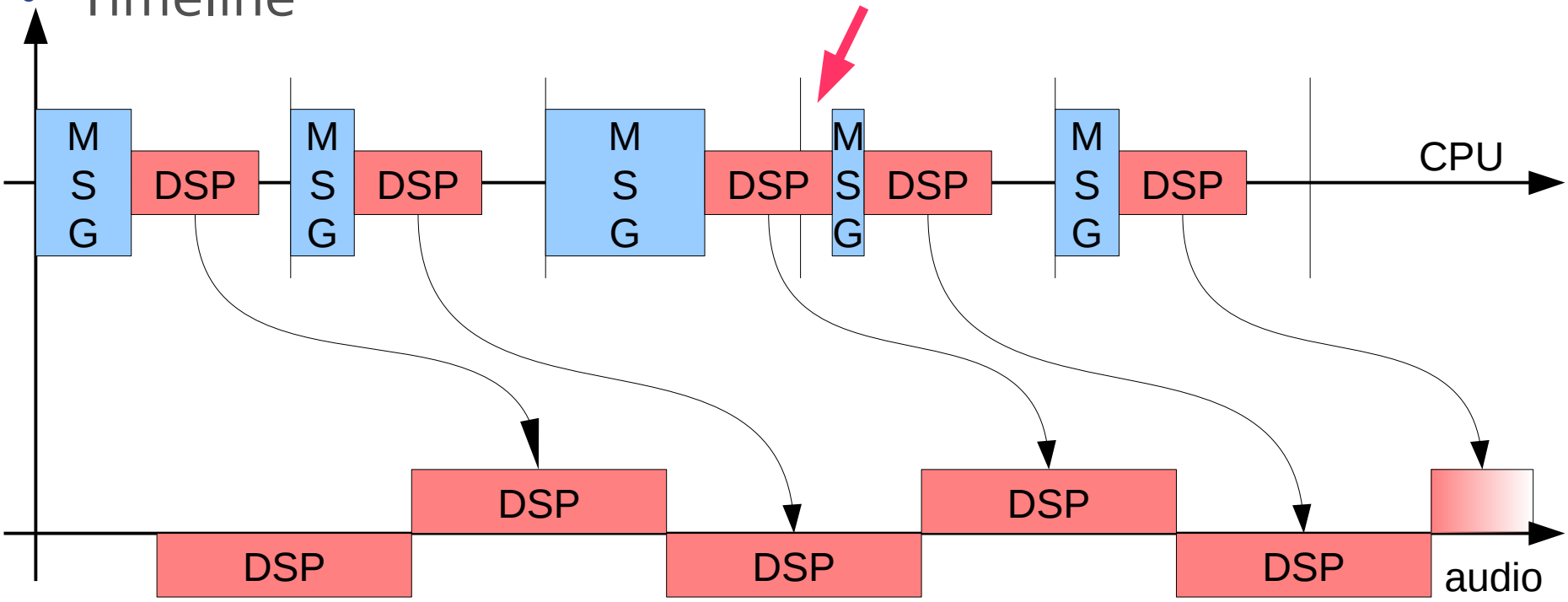
IOhannes m zmölnig

Scheduler

- Loop
  - messages (var.Dauer)
  - dsp (rel.const.Dauer)
  - *sleep*

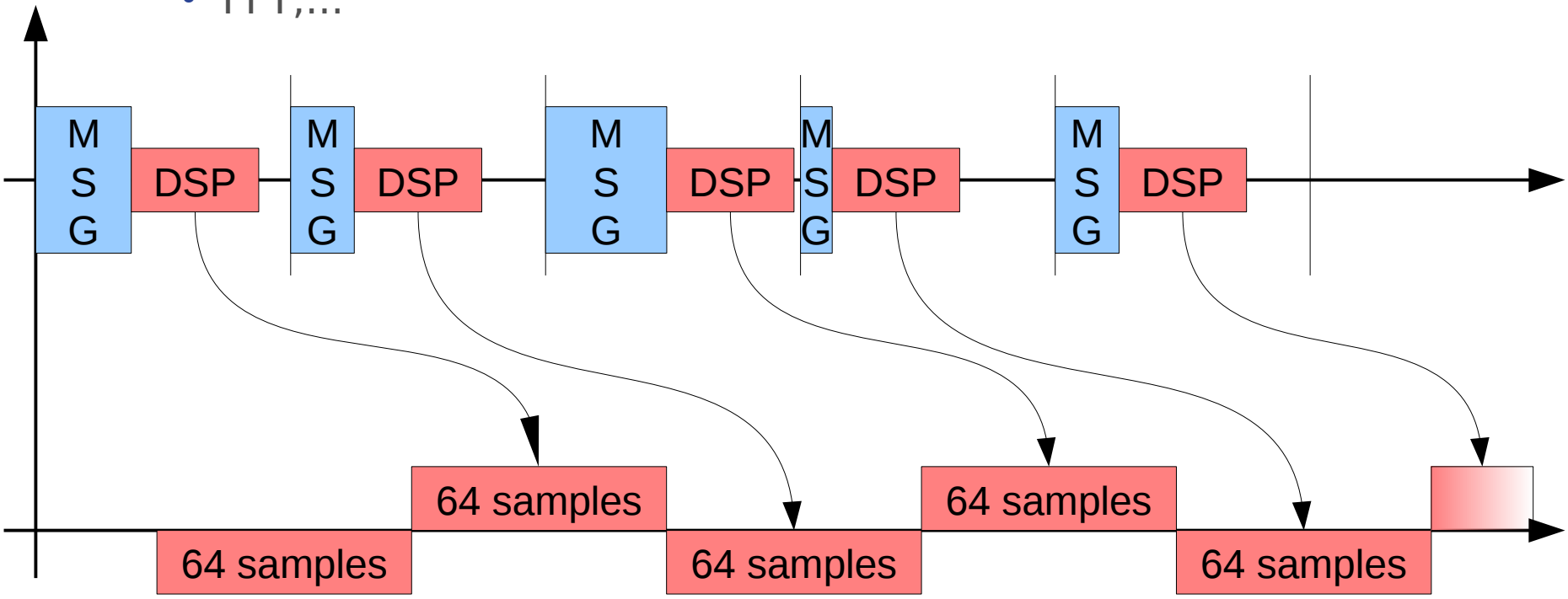


- Timeline



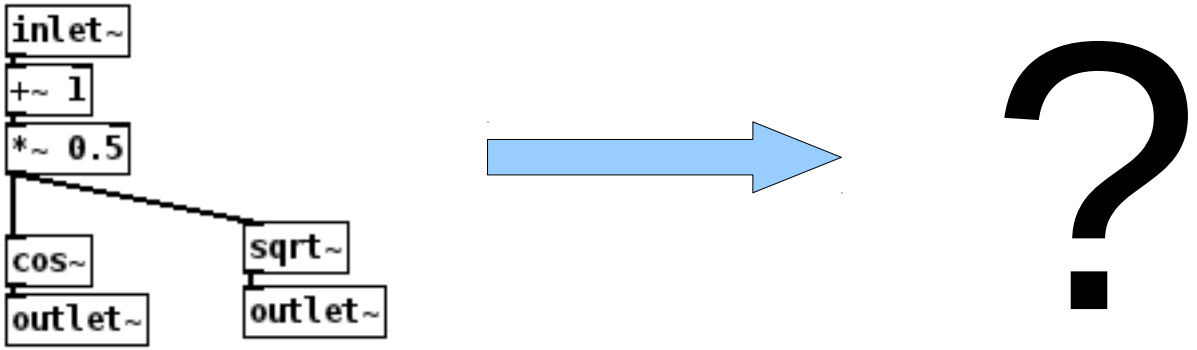
DSP

- Blockverarbeitung
  - Latenz ↔ Performance
  - Nachteil
    - rekursive Filter
  - Vorteil
    - FFT,...



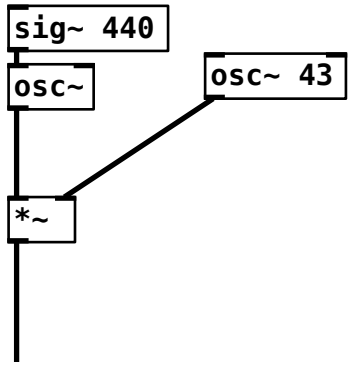
DSP-Graph

- „Kompilieren“ des Datenfluss-Diagramms in ausführbares „Programm“



DSP-Graph

- Auflösen von Abhängigkeiten
  - **Senke** kann erst ausgeführt werden, wenn *alle* Eingänge befüllt sind
  - → d.h. nachdem jede (verbundene) **Quelle** ausgeführt wurde

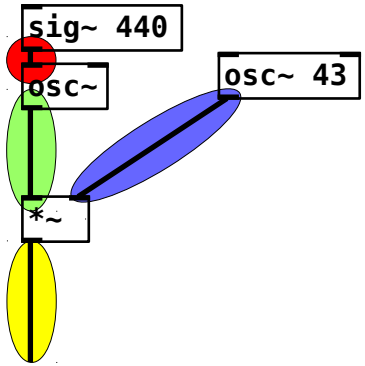


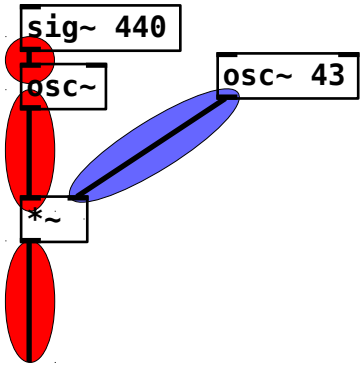
1. [sig~ 440]
2. [osc~]
3. [osc~ 43]
4. [*~]

1. [osc~ 43]
2. [sig~ 440]
3. [osc~]
4. [*~]

DSP-Graph

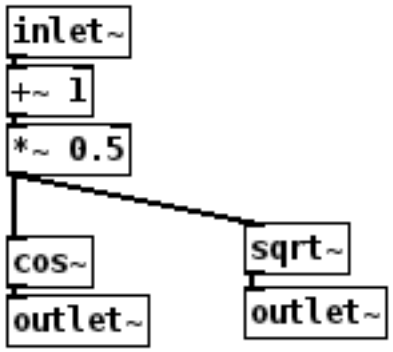
- Bufferverwaltung
  - Copy Optimierung
    - Quelle schreibt direkt in den Eingangsbuffer einer Senke
  - Cache Optimierung
    - Wiederverwendung von Buffern
      - Eingangsbuffer = Ausgangsbuffer
      - Mehrere Objekte verwenden den gleichen Buffer





DSP-Graph

- „Kompilieren“ des Datenfluss-Diagramms in linearisierten DSP-Graphen
  - Auflösen von Abhängigkeiten
  - Cache-Optimierung
- ~~linked list~~ → array
  - „perform“ routinen + speicher für I/O



+~	vec1	vec1
*~	vec1	vec1
cos~	vec1	vec2
outlet~	vec2	–
sqrt~	vec1	vec3
outlet~	vec3	–

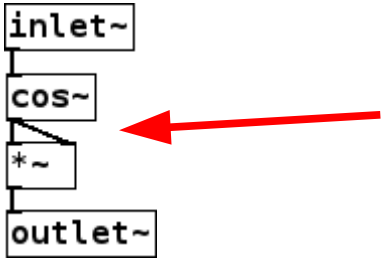
DSP-Perform Routine

- wird für jeden DSP-tick aufgerufen
- generiert aus einem Block Eingangssamples einen Block Ausgangssamples
- Input- und Outputbuffer können sich überlappen!

```
cos~::perform(int samples, t_signal input, t_signal output)
{
    for i in samples:
        output[i]=cos(input[i]);
}
```

DSP: Order of Execution

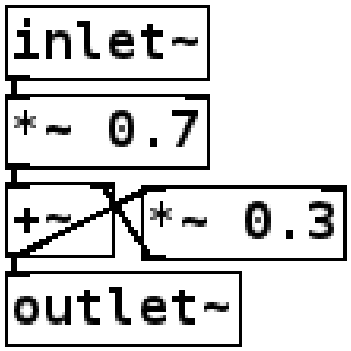
- Dataflow
  - Alle Eingänge müssen befüllt sein, bevor Ausgabe generiert werden kann
  - **synchron**
    - **bei jedem Zyklus müssen *immer alle* Eingänge befüllt werden**
    - **hot/cold wird nicht benötigt**!



`signalmul::perform(in1, in1, out1);`
wird erst aufgerufen, nachdem [cos~] den „in1“ Vector geschrieben hat

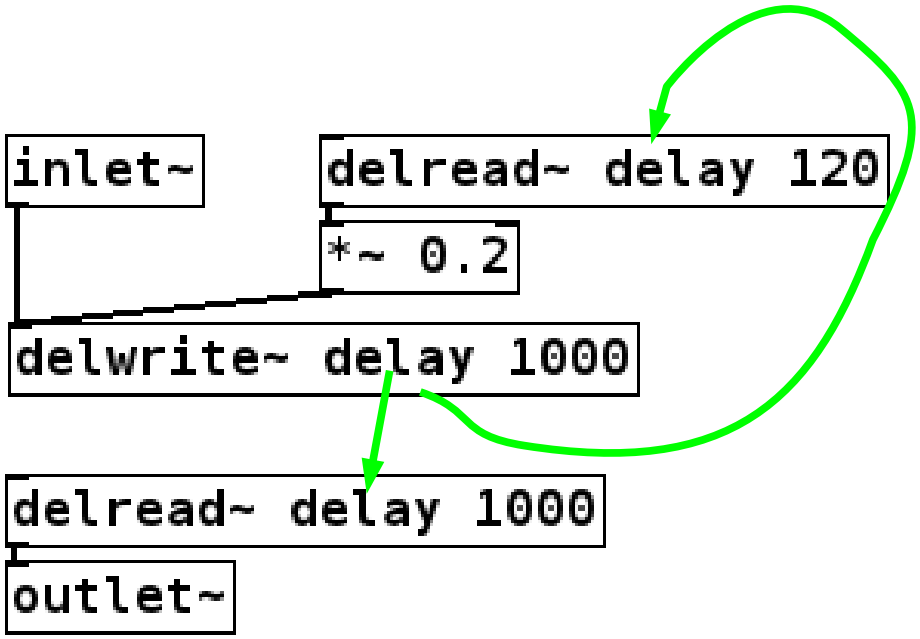
DSP: Rekursion

- Dataflow-Bedingung (alle Eingänge befüllt) kann **nicht** immer **eingehalten** werden



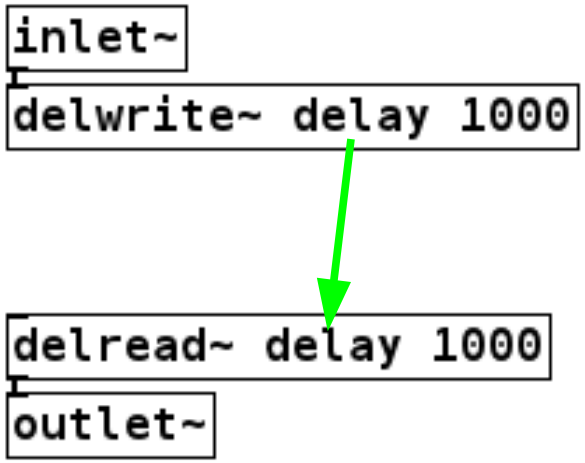
DSP: Rekursion

- → Verzögerung
  - Um (mindestens) einen ganzen Block(!)



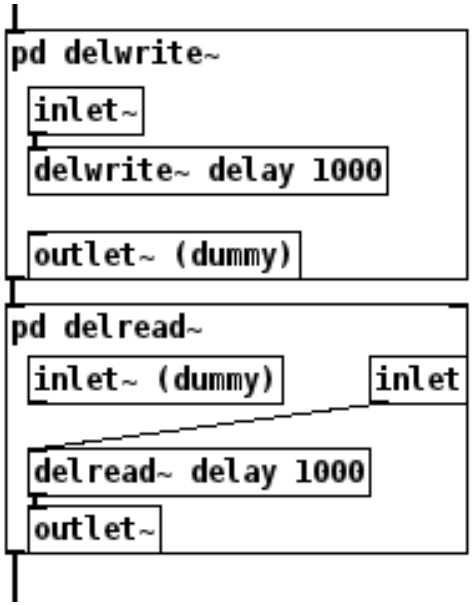
DSP: Implizite Verbindungen

- Unklar, welches Objekt zuerst Output generieren muss/soll!
  - wenn [delread~] vor [delwrite~] ausgeführt wird, gibt’s ein Delay von (mindestens) 1 Block
- „trigger für DSP“?



DSP: Order Forcing

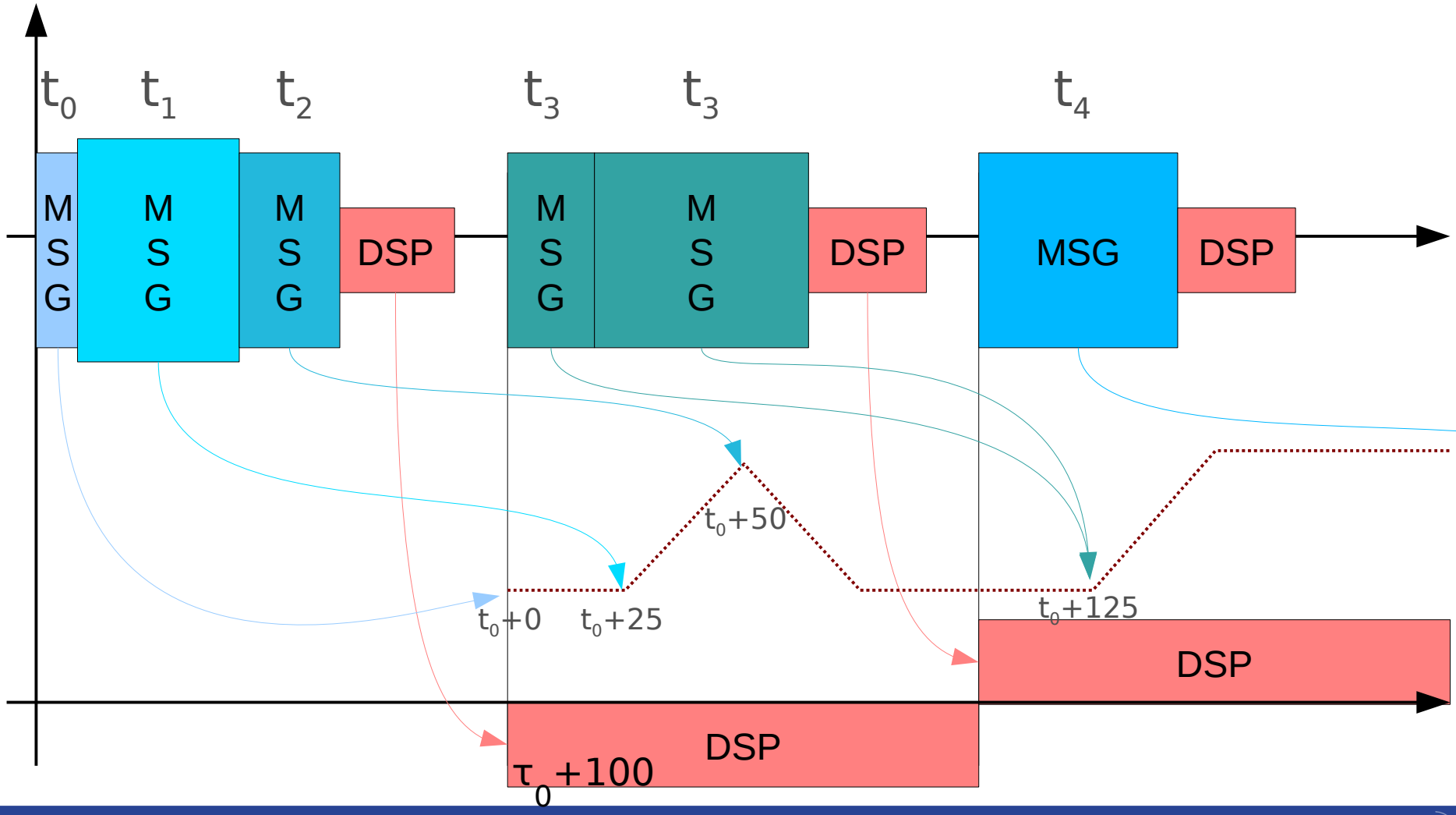
- Explizites Festlegen der Ausführungsreihenfolge
- Subpatches/Abstraktionen werden immer „gemeinsam“ ausgeführt
- Subpatches/Abstraktionen können mit iolet~s geordnet werden



„dummy“-outlet~→„dummy“ inlet~ garantiert, dass
[pd delwrite~] (und damit [delwrite~])
immer *vor*
[pd delread~] (und damit [delread~])
ausgeführt wird

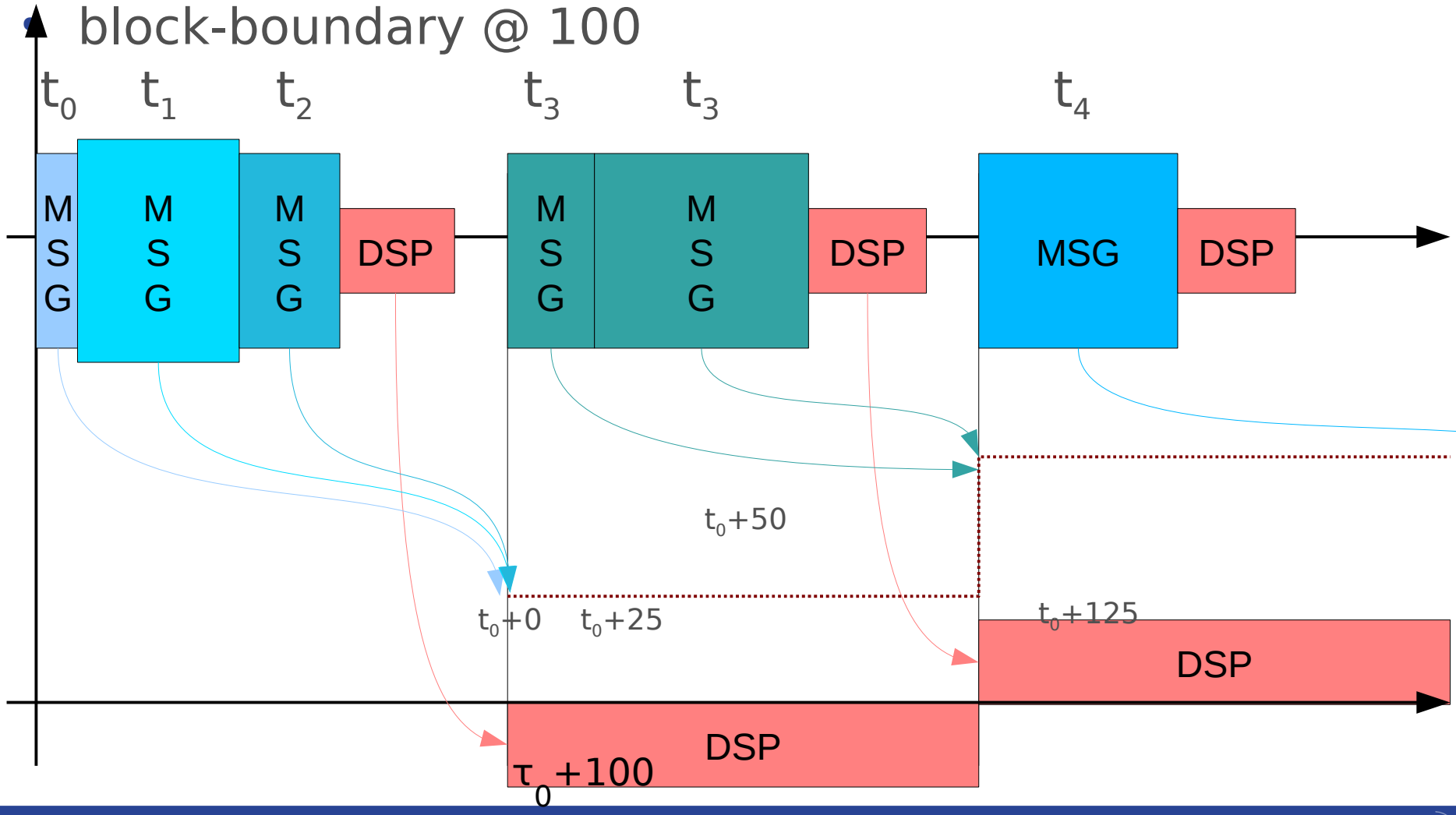
logical time and DSP time (ideal)

- $t_0 = t_1\text{-}25 = t_2\text{-}50 = t_3\text{-}125 = t_4\text{-}250$



logical time and DSP (block boundaries)

- $t_0 = t_1\text{-}25 = t_2\text{-}50 = t_3\text{-}125 = t_4\text{-}250$
- block-boundary @ 100



(sub)-sample accurate timing

- $t_0 = t_1\text{-}25 = t_2\text{-}50 = t_3\text{-}125 = t_4\text{-}250$