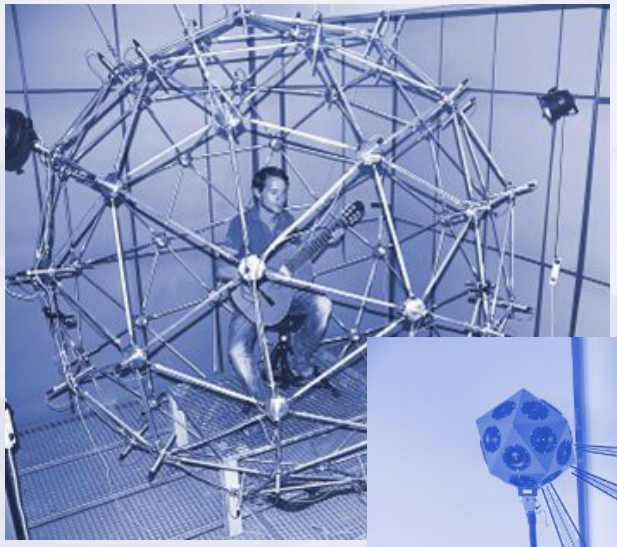
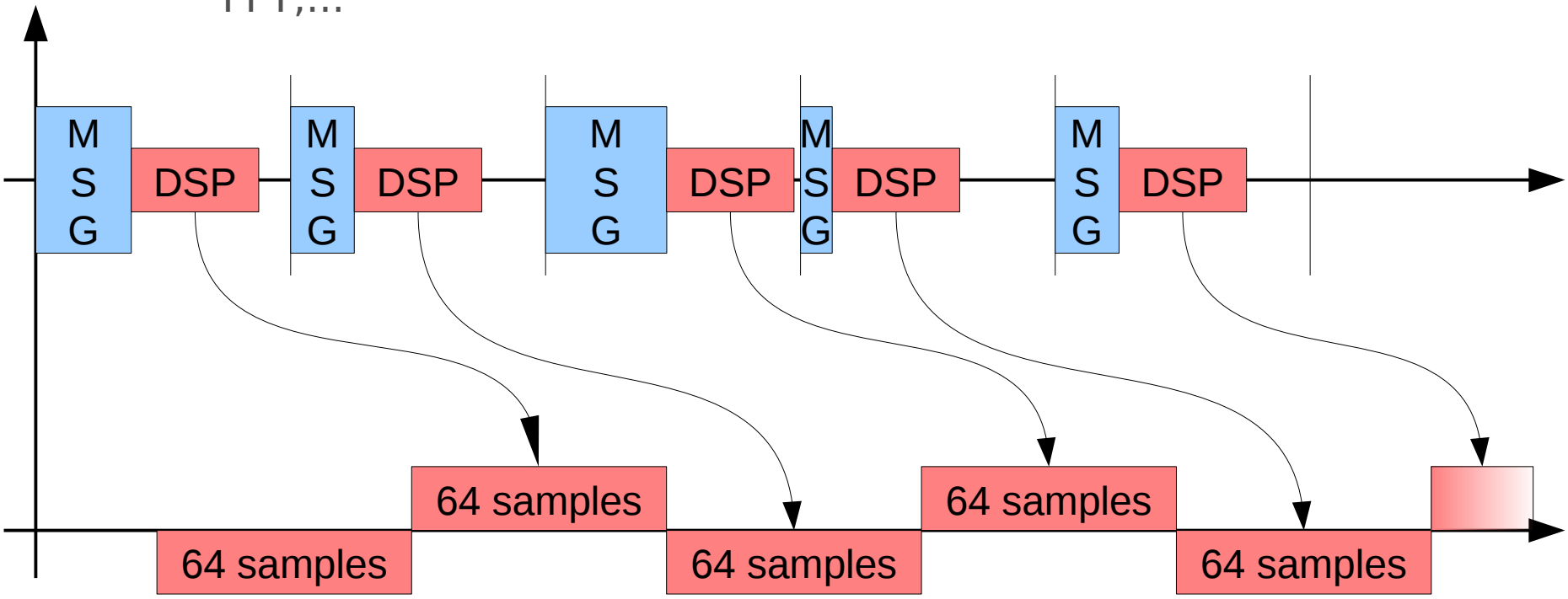
Implementierung von Algorithmen Pure Data: Signal Processing



IOhannes m zmölnig

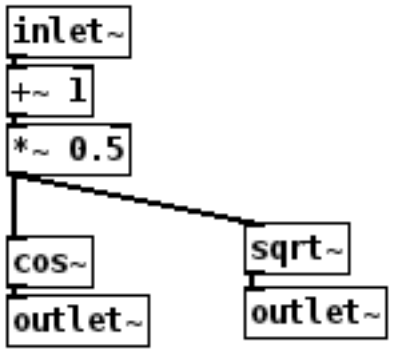
DSP

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



DSP-Graph

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

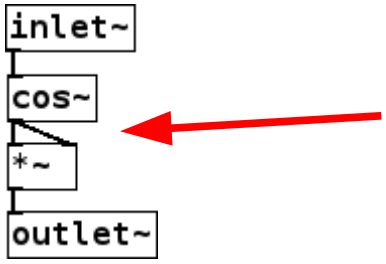


+~	vec1	vec1
*~	vec1	vec1
cos~	vec1	vec2
outlet~	vec2	–
sqrt~	vec1	vec3
outlet~	vec3	–

```
cos~::perform(t_signal input, t_signal output);
```

DSP: Order of Execution

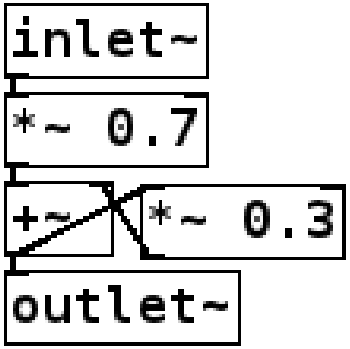
- Dataflow
  - Alle Eingänge müssen befüllt sein, bevor Ausgabe generiert werden kann
  - synchron
    - bei jedem Zyklus müssen *immer alle* Eingänge befüllt werden
    - hot/cold wird nicht benötigt!



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

DSP: Rekursion

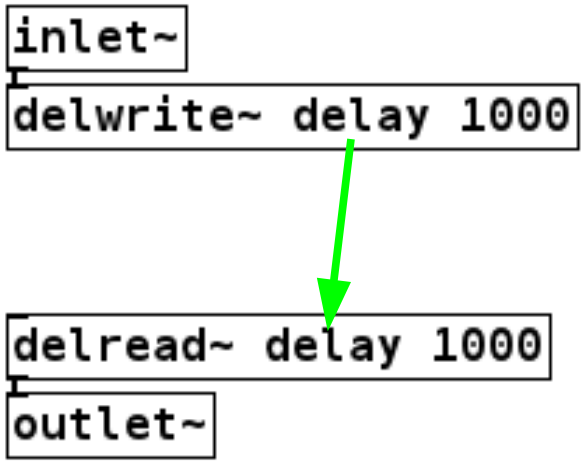
- Dataflow-Bedingung (alle Eingänge befüllt) kann nicht immer eingehalten werden
- → Verzögerung
  - um einen ganzen Block(!)

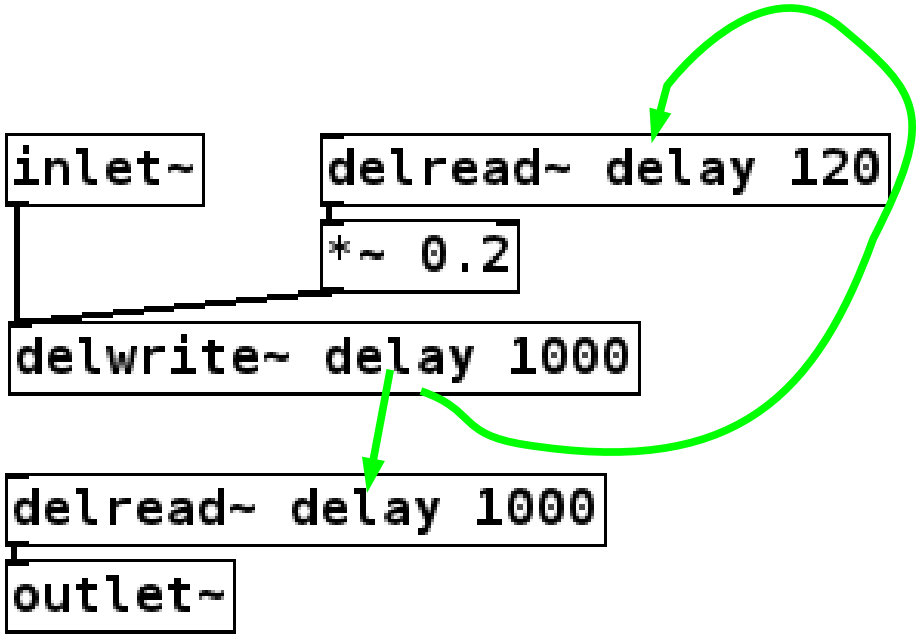


(don't try this at home...)

DSP: Implizite Verbindungen

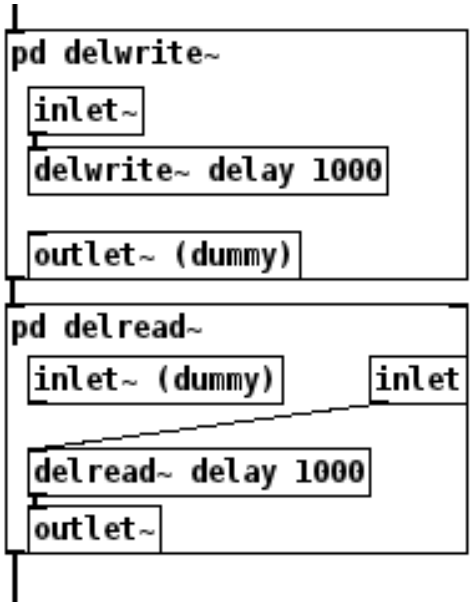
- Unklar, welches Objekt zuerst Output generieren muss/soll!
  - wenn [delread~] vor [delwrite~] ausgeführt wird, gibt's ein Delay von 1 Block
- „trigger für DSP"?





DSP: Order Forcing

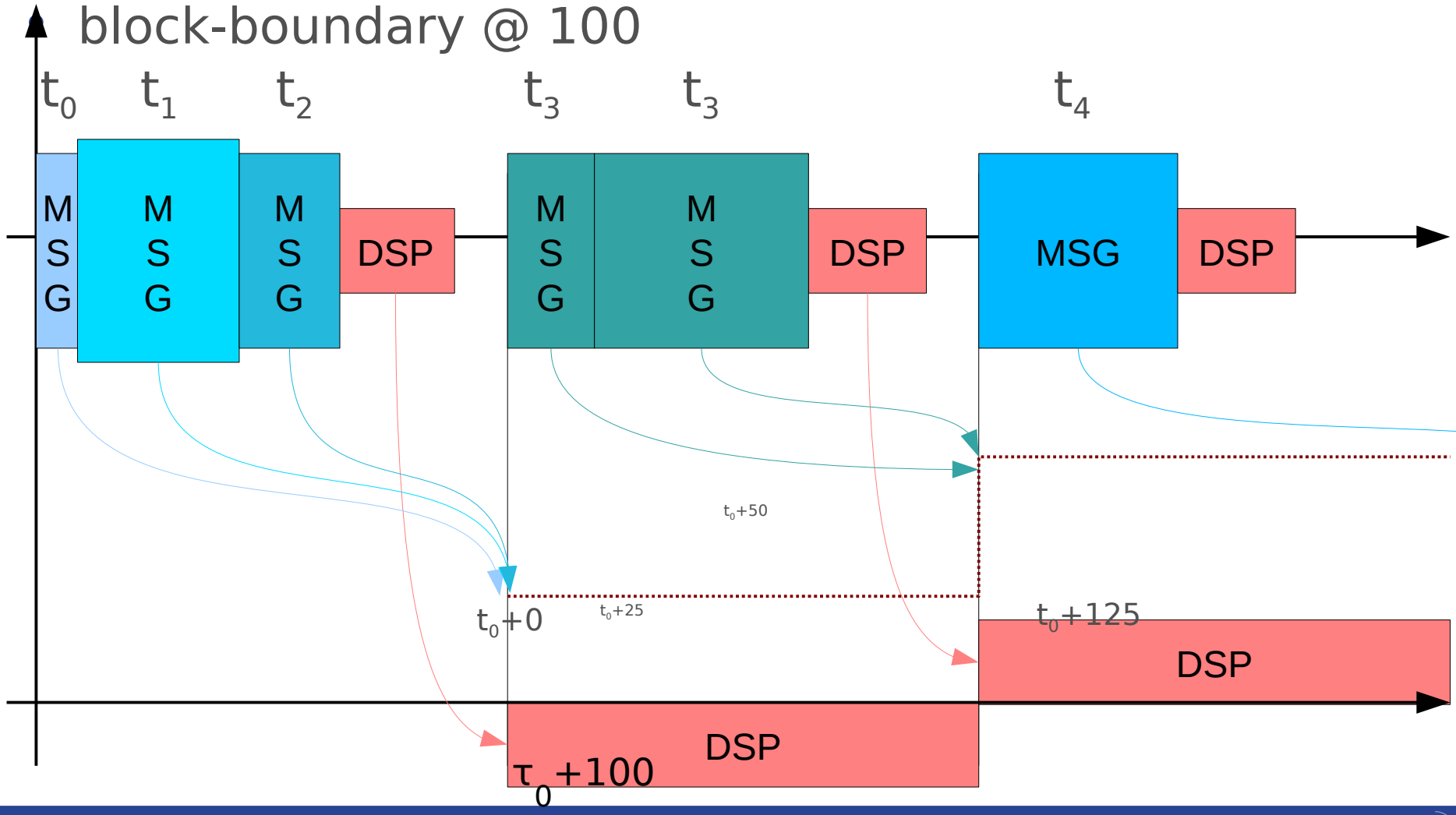
- Explizites Festlegen der Ausführungsreihenfolge
- Subpatches/Abstraktionen werden immer „gemeinsam“ ausgeführt
- Subpatches/Abstraktionen können mit iolet~s geordnet werden



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

logical time and DSP

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



(sub)-sample accurate timing

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