Implementierung von Algorithmen Pure Data: Scheduler

IOhannes m zmölnig

Logische Zeit

- innerhalb von Pd
  - alle Events finden zur „richtigen“ logischen Zeit statt (timestamp)
  - Folge-Message
    - finden zum gleichen logischen Zeitpunkt statt!
    - Messages passieren in „Null-Zeit“
  - Gleichzeitigkeit
    - *semantisch*: Messages mit gleichem Timestamp
    - *logisch*: Events werden immer sequentiell abgearbeitet!
      - deterministisch!

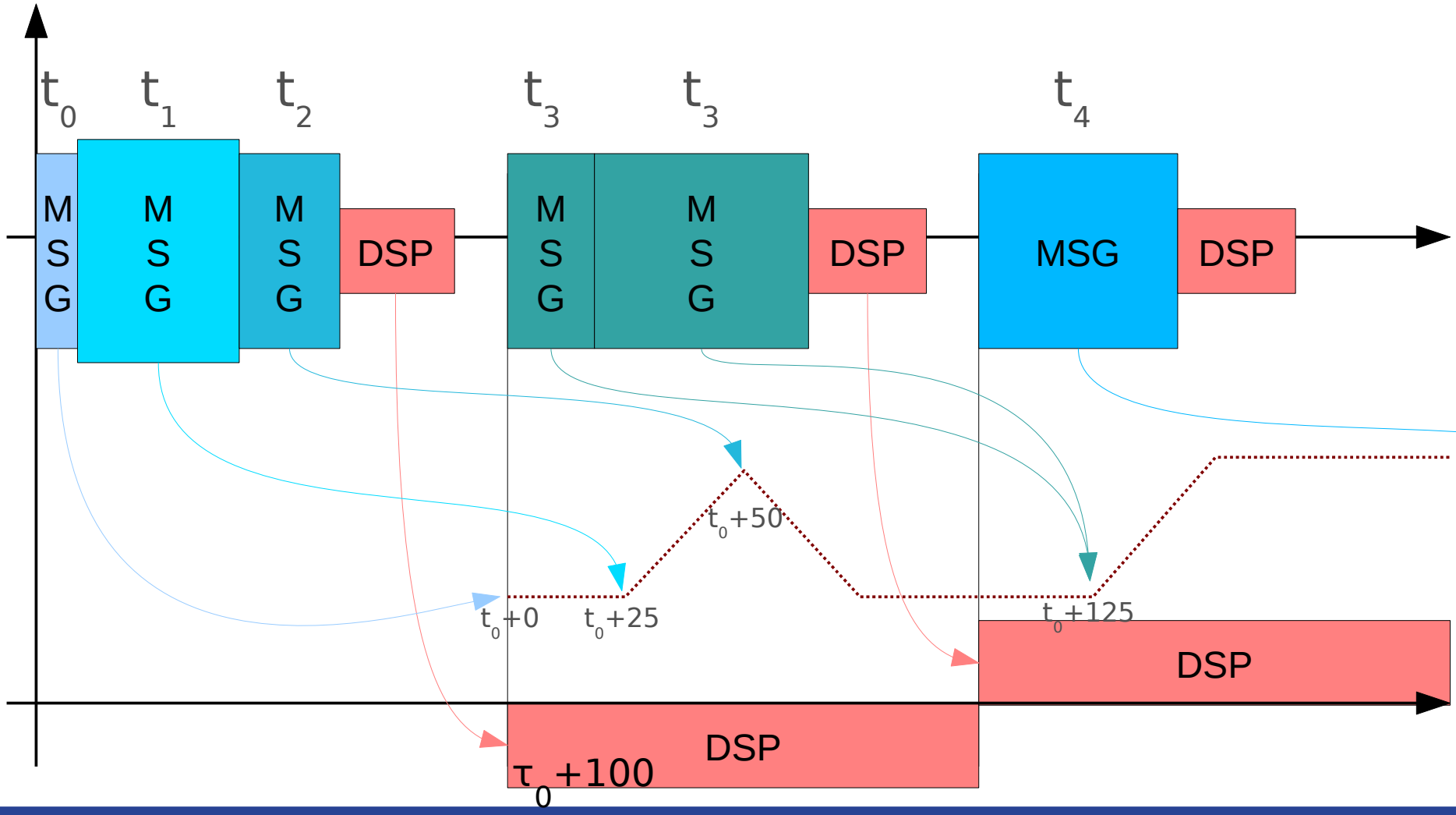
Echt(e) Zeit

- Messages werden in Bursts abgearbeitet
  - am Beginn des Tick-Zyklus
- Tick-Zyklen schwimmen innerhalb eines Buffers
- Message-Verarbeitung braucht Zeit (CPU-Zyklen)

- Objekte die mit der Real World synchronisiert sind, können Timestamps verwenden um die logische Zeit in echte Zeit zu übersetzen

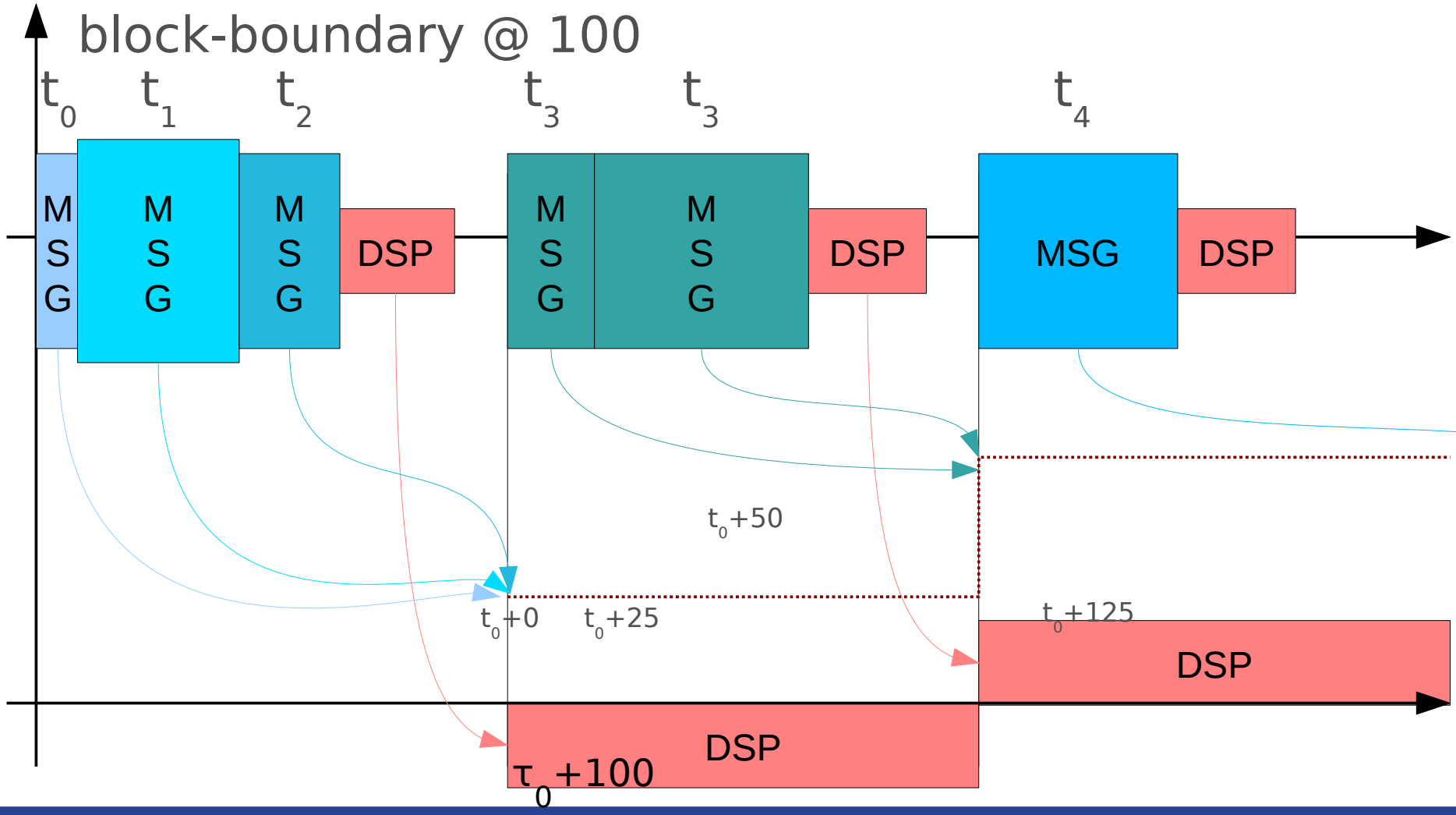
logical time vs real time

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



logical time vs real time

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



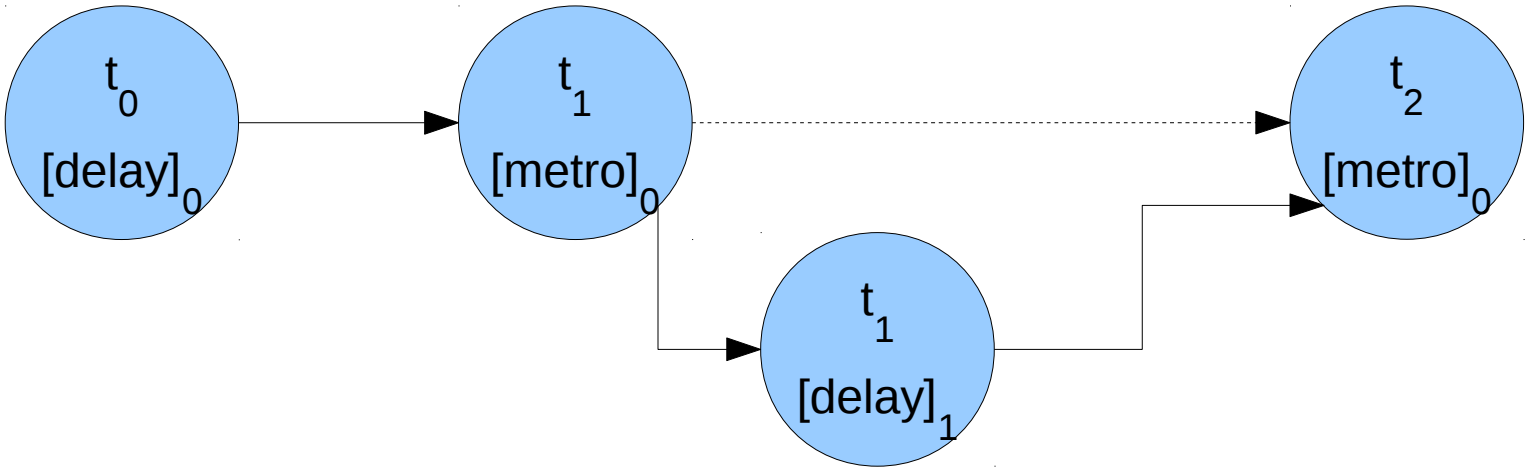
Message vs DSP

- Messages
  - asynchron
  - logische NULL-Dauer
  - variable Evaluierungszeit
- DSP
  - synchron zu Real World
  - fixe Dauer (Blocksize)
  - ~const. Evaluierungszeit

- Messages werden **vor** DSP-Berechnungen abgearbeitet
- Messages werden **immer alle** abgearbeitet
- DSP-Berechnungen können **ausfallen**!

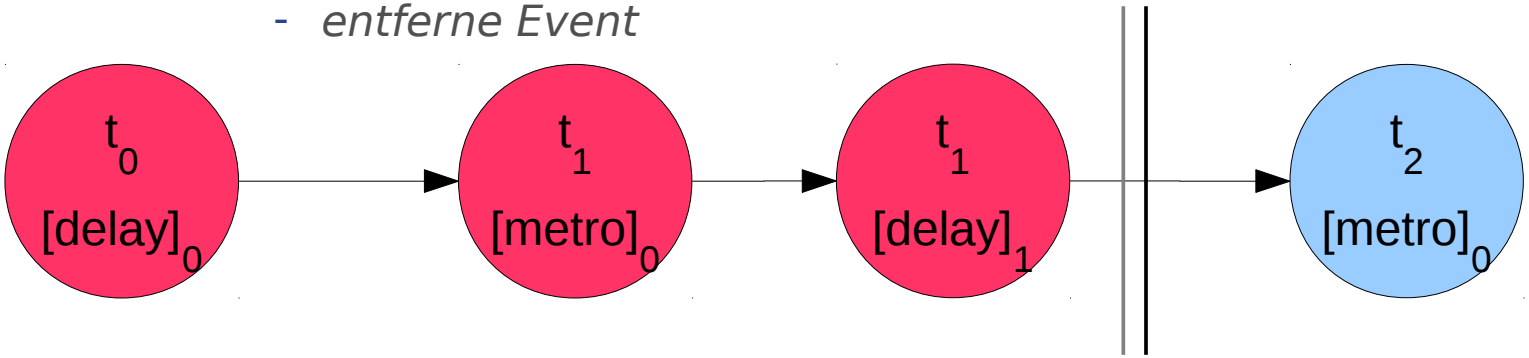
Message Scheduler

- geordnete Liste von Events (Timestamp+Objekt)
  - Events werden *am Ende* des jeweiligen Timestamps hinzugefügt
- Tick @ τ
  - für alle Events mit timestamp < τ:
    - *setze logische Zeit auf „timestamp“*
    - *rufe Objekt::tickMethod() auf*
    - *entferne Event*



Message Scheduler

- Tick @ τ
  - für alle Events mit timestamp < τ:
    - *setze logische Zeit auf „timestamp“*
    - *rufe Objekt::tickMethod() auf*
    - *entferne Event*



- τ: „bang“ event from scheduler
  - [print a]
  - schedule event at τ+0=τ
  - [print c]
- τ+0: new „bang“ event from scheduler
  - [print b]