

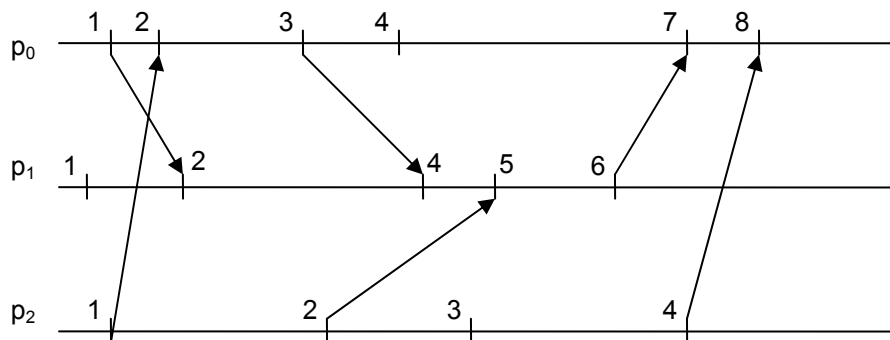
# Übung SA Gruppe Hosebei

20.06.2002

Michele Luongo	migi@luongo.org	s99-713-190
Franziska Zumsteg	F.Zumsteg@access.unizh.ch	s99-717-084
Philip Iezzi	pipo@iezzi.ch	s99-714-354
Raphael Bianchi	saint.ch@dte.ch	s95-662-003

## Aufgabe 5.1

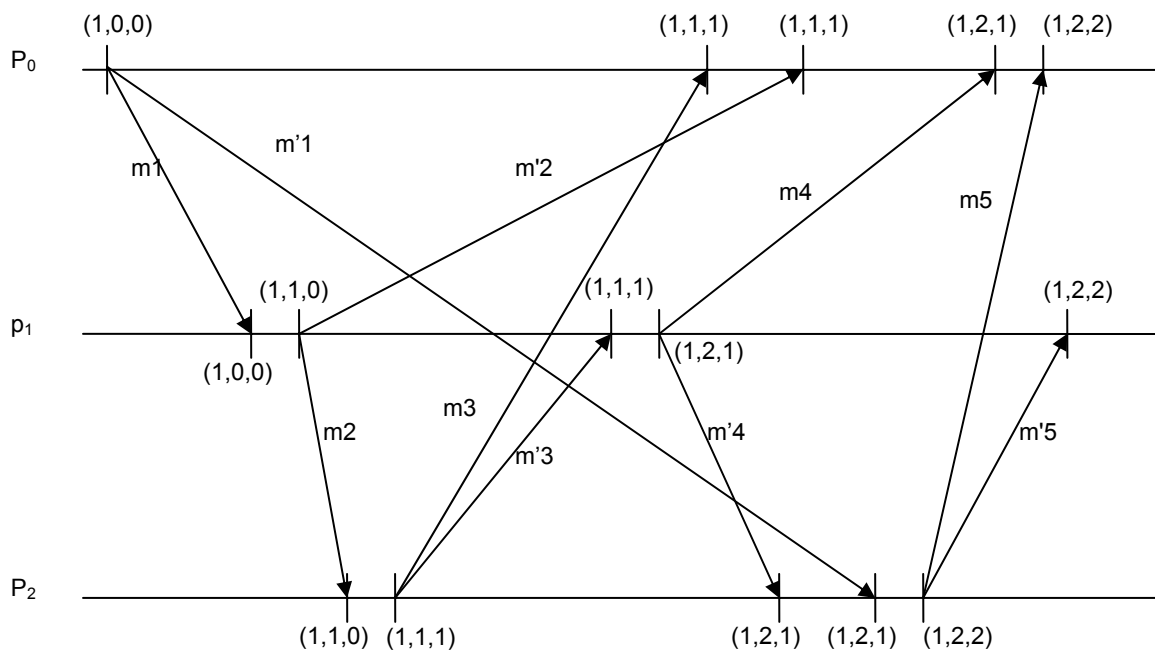
**Wert der logischen Uhren (LC):**



## Aufgabe 5.2

**a) Wert der Vektoruhren**

lokale Komponente der Uhr wird nur dann inkrementiert, wenn eine Nachricht gesendet wird!:

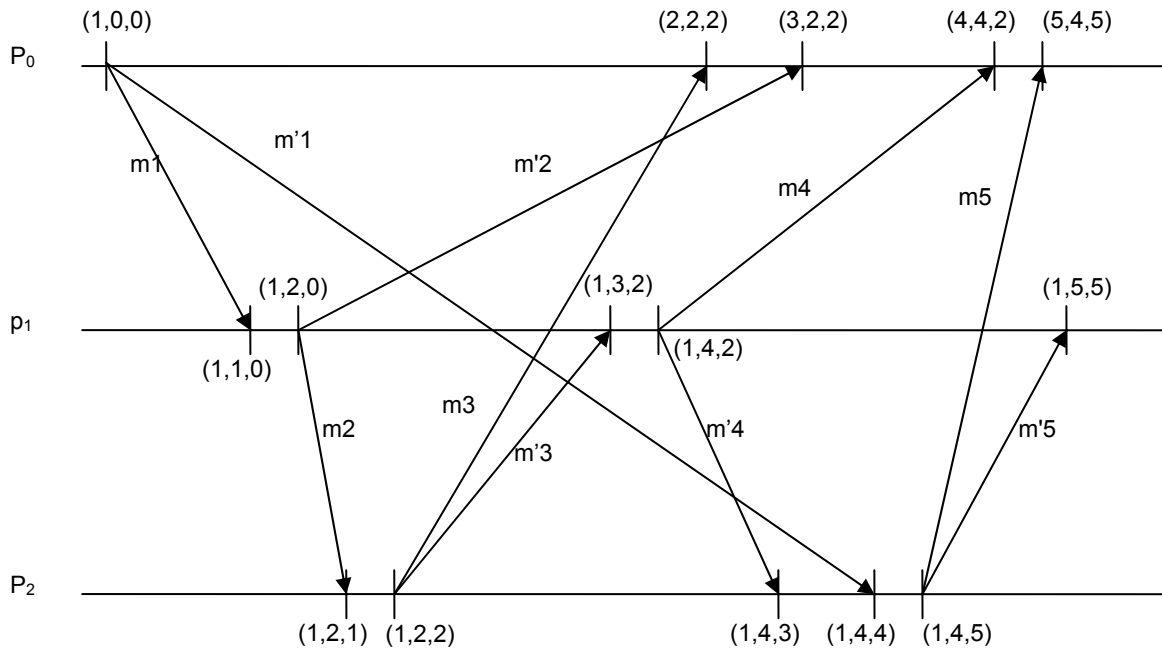


**Mullender - VC**

Lösung gemäss Mullender 2nd Ed., S. 72:

```

VC(ei) [i] := VC[i]+1      if ei is an internal or send event
VC(ei) := max{VC, TS(m)}  if ei = receive(m)
VC(ei) [i] := VC[i]+1
    
```

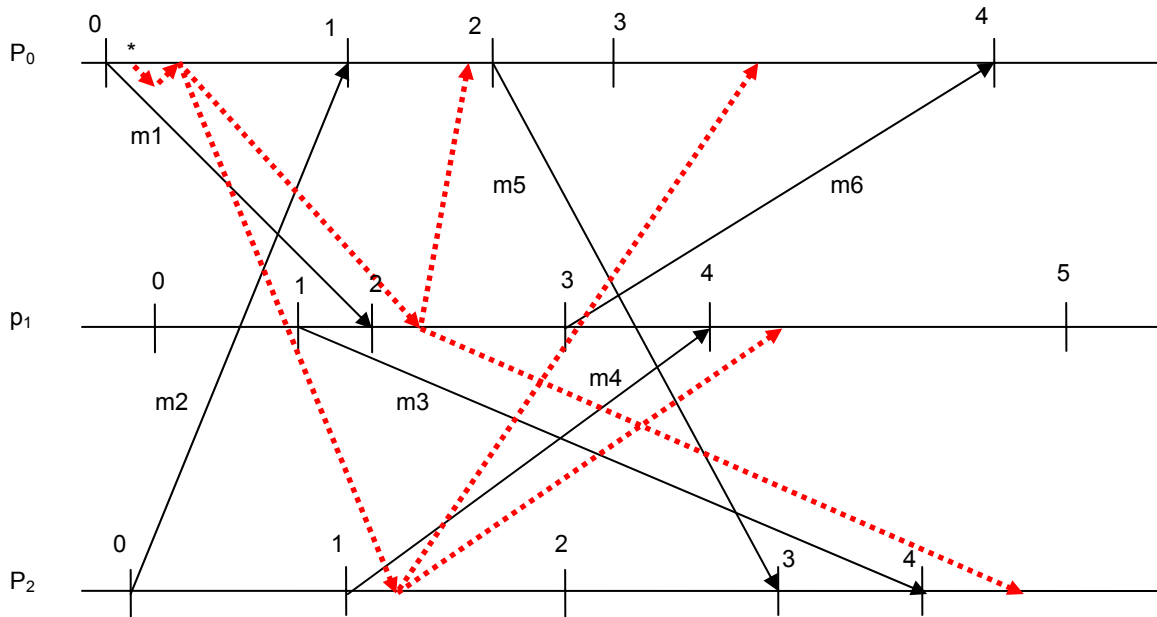


**b) kausale Lieferungsregel (DR3)**

D[1...3] vorher	Empfangene Nachricht mit Zeitstempel	gelieferte Nachricht	empfangene aber noch nicht gelieferte Nachrichten	D[1...3] nachher
[0,0,0]	m3(1,1,1)	-	{m3}	[0,0,0]
[0,0,0]	m'2(1,1,0)	m'2	{m3}	[0,1,0]
[0,1,0]	-	m3	{}	[0,1,1]
[0,1,1]	m4(1,2,1)	m4	{}	[0,2,1]
[0,2,1]	m5(1,2,2)	m5	{}	[0,2,2]

### Aufgabe 5.3

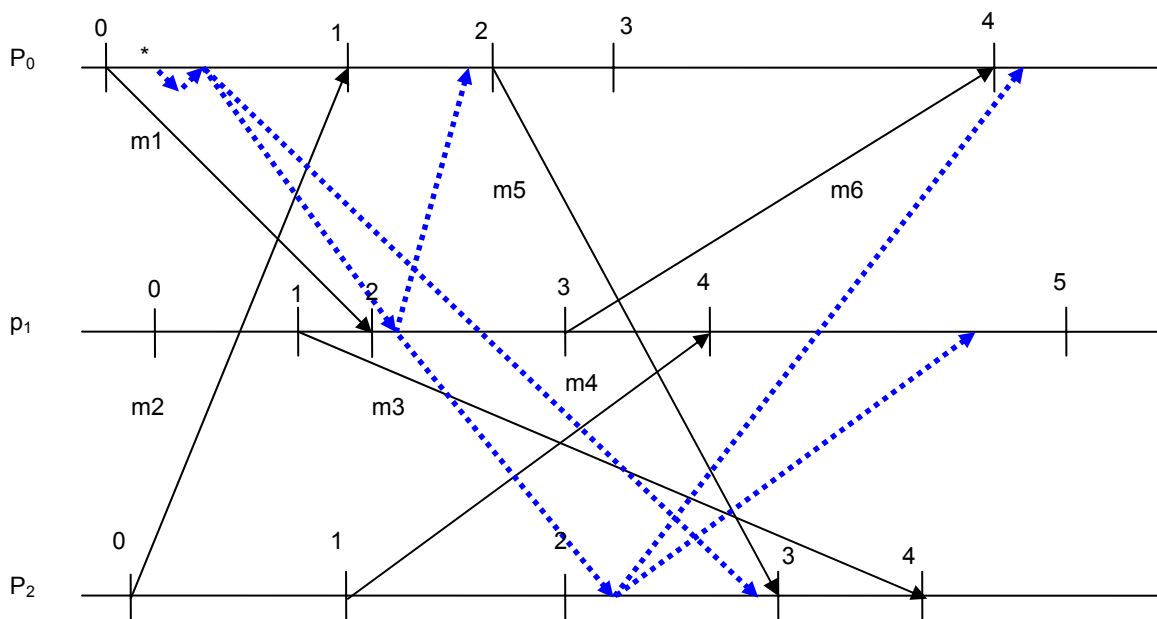
#### 1. Ablauf:



globaler Zustand:  $\sum^{021}$

$p_0$	$p_1$	$p_2$
$\sigma_0 = 0$	$\sigma_1 = 2$	$\sigma_2 = 1$
$\chi_{10} = \{\}$	$\chi_{01} = \{\}$	$\chi_{02} = \{\}$
$\chi_{20} = \{m_2\}$	$\chi_{21} = \{m_4\}$	$\chi_{12} = \{m_3\}$

#### 2. Ablauf:



globaler Zustand:  $\sum^{022}$

$p_0$	$p_1$	$p_2$
$\sigma_0 = 0$	$\sigma_1 = 2$	$\sigma_2 = 2$
$\chi_{10} = \{\}$	$\chi_{01} = \{\}$	$\chi_{02} = \{\}$
$\chi_{20} = \{m_2\}$	$\chi_{21} = \{m_4\}$	$\chi_{12} = \{\}$

## Aufgabe 5.4

### Qualitätseigenschaften Broadcast

**zuverlässig (reliable) = validity + agreement + integrity**

**validity:** Wenn ein korrekter Prozess broadcast(m) durchführt, führen (irgendwann) alle korrekten Prozesse deliver(m) aus.

**agreement:** Wenn ein korrekter Prozess eine deliver(m) durchführt, dann führen irgendwann alle korrekten Prozesse deliver(m) aus

**integrity:** deliver(m) wird von einem korrekten Prozess höchstens ein Mal ausgeführt, und nur, wenn ein Prozess ein broadcast(m) durchgeführt hat.

### Satz 1

In einem asynchronen System, in dem jedes Paar von fehlerfreien Prozessen durch einen fehlerfrei funktionierenden Kommunikationspfad verbunden sind, garantiert B1 einen zuverlässigen Broadcast.

### Beweis:

- *nicht validity => nicht (B1 + Annahmen)*  
 $\neg(\text{validity}) \Rightarrow \neg(\text{alle korrekten Prozesse sind zuverlässig}) \Rightarrow \text{Nachrichten müssen verloren gehen} \Rightarrow \neg(\text{B1} + \text{Annahmen})$
- *nicht agreement => nicht (B1 + Annahmen)*  
 $\neg(\text{agreement}) \Rightarrow \text{ein korrekter Prozess führt deliver(m) aus} \wedge \neg(\text{alle anderen korrekten Prozesse führen deliver(m) aus}) \Rightarrow \neg(\text{alle Nachrichten kommen irgendwann an}) \Rightarrow \neg(\text{B1} + \text{Annahmen})$
- *nicht integrity => nicht (B1 + Annahmen)*  
 $\neg(\text{integrity}) \Rightarrow \neg(\text{ein Prozess liefert maximal einmal}) \Rightarrow \neg(\text{B1} + \text{Annahmen})$

### Satz 2

Wenn in einem asynchronen System Prozesse nur receive omissions Fehler haben und jeder Prozess mit jedem korrekten Prozess über einen fehlerfrei funktionierenden Kommunikationspfad verbunden ist, garantiert B1 gleichmässig zuverlässigen Broadcast (i.e. zusätzl. uniform agreement und uniform integrity).

### definitions:

*receive omissions:* Prozess empfängt Nachricht nicht (z.B. Crash)

*uniform agreement:* Führt ein (korrekter oder fehlerhafter) Prozess ein deliver(m) aus, dann führen irgendwann alle korrekten Prozesse ein deliver(m) aus.

*uniform integrity:* Jeder (korrekte oder fehlerhafte) Prozess führt deliver(m) höchstens 1 mal aus, und nur, wenn ein Prozess ein broadcast(m) durchgeführt hat.

**Beweis:**

*Uniform Agreement:*

Prozesse mit receive omissions sind keine korrekten Prozesse => nicht in der Definition von *Agreement* enthalten. Die Definition für *Agreement* gilt weiterhin für korrekte Prozesse und wurde bereits oben (Satz 1) bewiesen.

Der Prozess, der *broadcast(m)* ausführt, muss jedoch nicht korrekt sein (d.h. darf receive omissions haben, da diese das Senden ja nicht beeinträchtigen). => Alle korrekten Prozesse führen irgendwann *deliver(m)* durch

*Uniform Integrity:*

ein nicht korrekter Prozess (mit receive omissions) kann gar kein *deliver(m)* ausführen, da er ja gar nie erst eine Nachricht erhalten hat => *deliver(m)* wird höchstens 1 mal (d.h. in diesem Fall 0 mal) ausgeführt!

*Validity:*

Die Definition von *Validity* beinhaltet nur korrekte Prozesse → siehe Beweis oben (Satz 1)