Ordnung
in Verteilten Systemen

Wozu Ordnung?

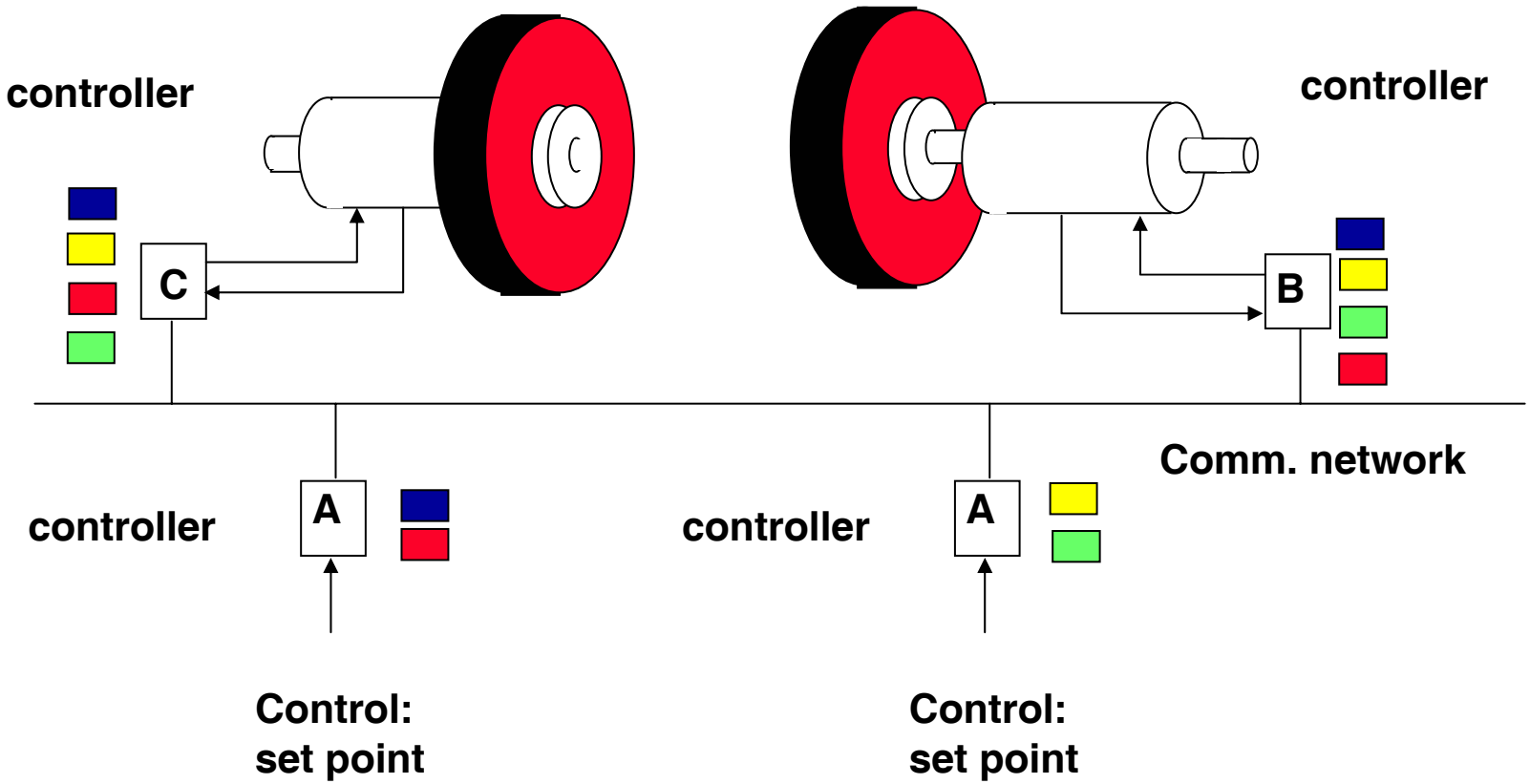
Bestimmen der Reihenfolge, in der Ereignisse auftreten *können* !

- **Ermitteln von Ursache und Wirkung (Kausalität) in einer verteilten Berechnung (Debugging).**

Durchsetzen einer bestimmten *Ordnungsstrategie*, d.h. einer a priori festgelegten Reihenfolge von Ereignissen.

- **Koordination gemeinsamer Aktivitäten.**

Ordnung muss sein!



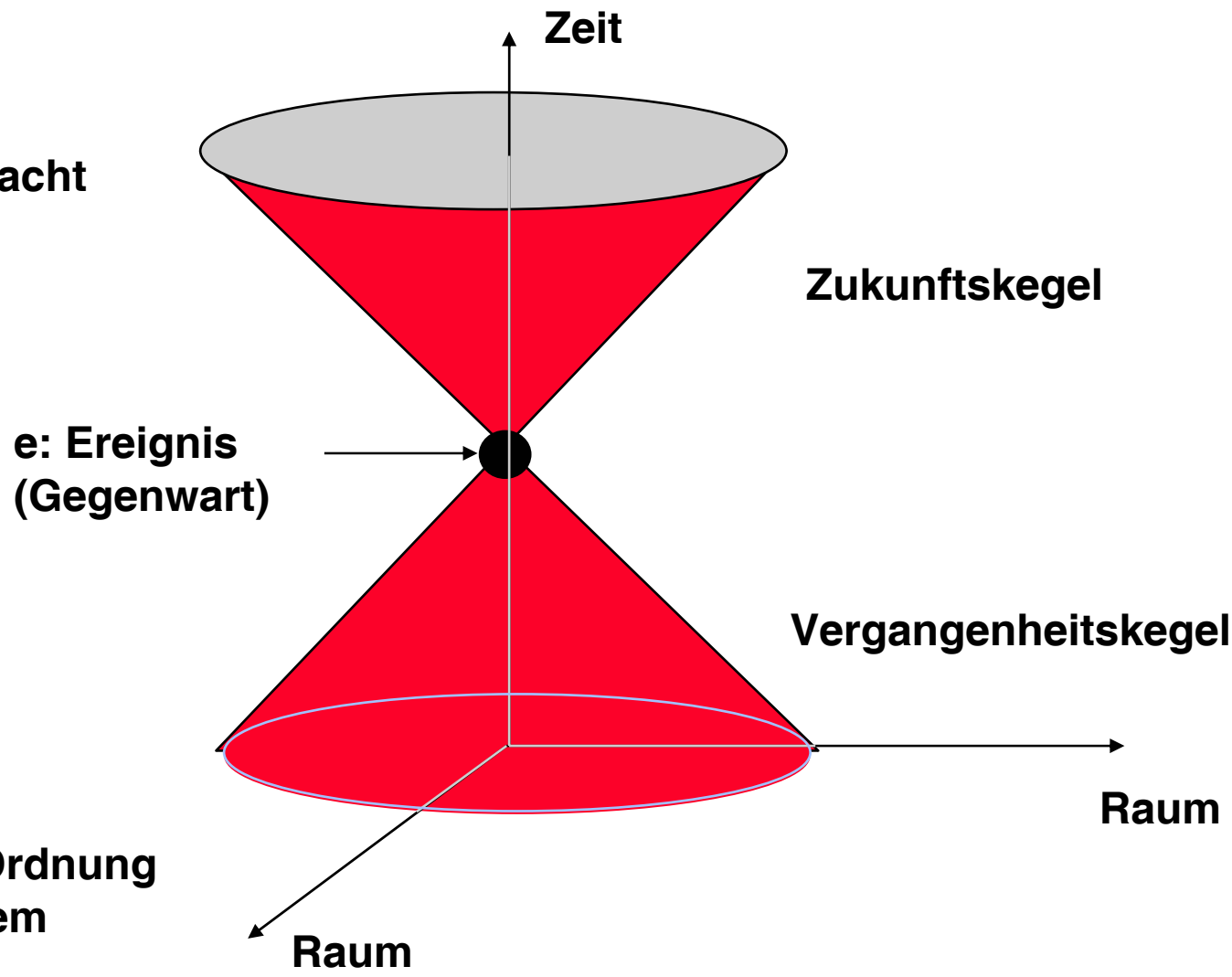
I.

Was kann geordnet werden?

Wodurch entsteht Ordnung in einem verteilten System?

Was kann geordnet werden?

e kann Auswirkungen auf Ereignisse haben, die im Zukunftskegel liegen und kann nur durch Ereignisse verursacht worden sein, die im Vergangenheitskegel liegen.



Ist eine totale zeitliche Ordnung in einem verteilten System prinzipiell herstellbar ?

Vorrang-Relation (Precedence)

In einem System können Ereignisse aufgrund der kausalen Relation zwischen Ursache und Wirkung geordnet werden (happens before ! (Lamport 78)).

Def.: Vorrang-Relation $\rightarrow$

1. Für alle e_i^k , $e_i^l \in h_i$ mit $k < l$: $e_i^k \rightarrow e_i^l$ (h_i ist die "Vergangenheit" von Prozess i)

2. Wenn e_i = send (m) und e_j = receive (m) : $e_i \rightarrow e_j$

3. Wenn $e \rightarrow e'$ und $e' \rightarrow e''$ gilt: $e \rightarrow e''$

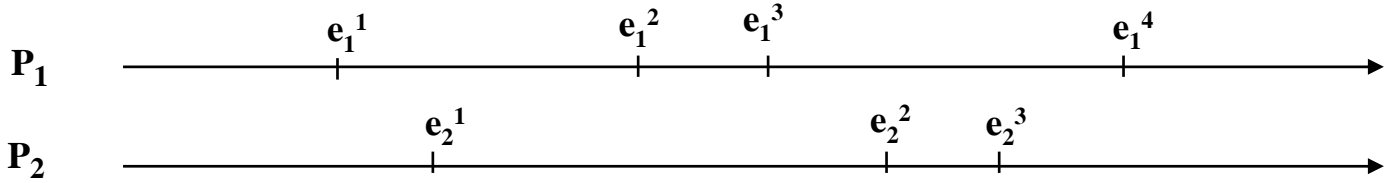
Nebenläufige Ereignisse sind solche, die in keiner kausalen Abhängigkeit zueinander stehen, d.h. es gilt weder $e \rightarrow e'$ noch gilt $e' \rightarrow e$. Schreibweise: $e \parallel e'$

Eine verteilte Berechnung kann formal als eine partiell geordnete Menge, definiert durch das Tupel $(H, \rightarrow)$, aufgefaßt werden.

Berechnungsmodell

Eine verteilte Berechnung wird durch die gemeinsame Aktivität lokaler, sequentieller Prozesse erbracht.

Die Aktivität eines lokalen sequentiellen Prozesses wird als Folge von Ereignissen modelliert.



Ein Ereignis ist entweder *prozeßlokal,* d.h. verursacht eine interne, lokale Zustandsänderung, oder

Ein Ereignis beinhaltet die Kommunikation mit einem anderen Prozeß. Das wird durch ein Sende- oder Empfangsereignis modelliert.

Nachrichten sind eindeutige, einmal vorhandene Ereignisse, d.h. zwei Nachrichten von demselben Prozess mit demselben Inhalt werden als zwei Ereignisse modelliert.

Alle Modelle des Data Sharing werden auf der gewählten Abstraktionsebene als Kommunikation aufgefaßt.

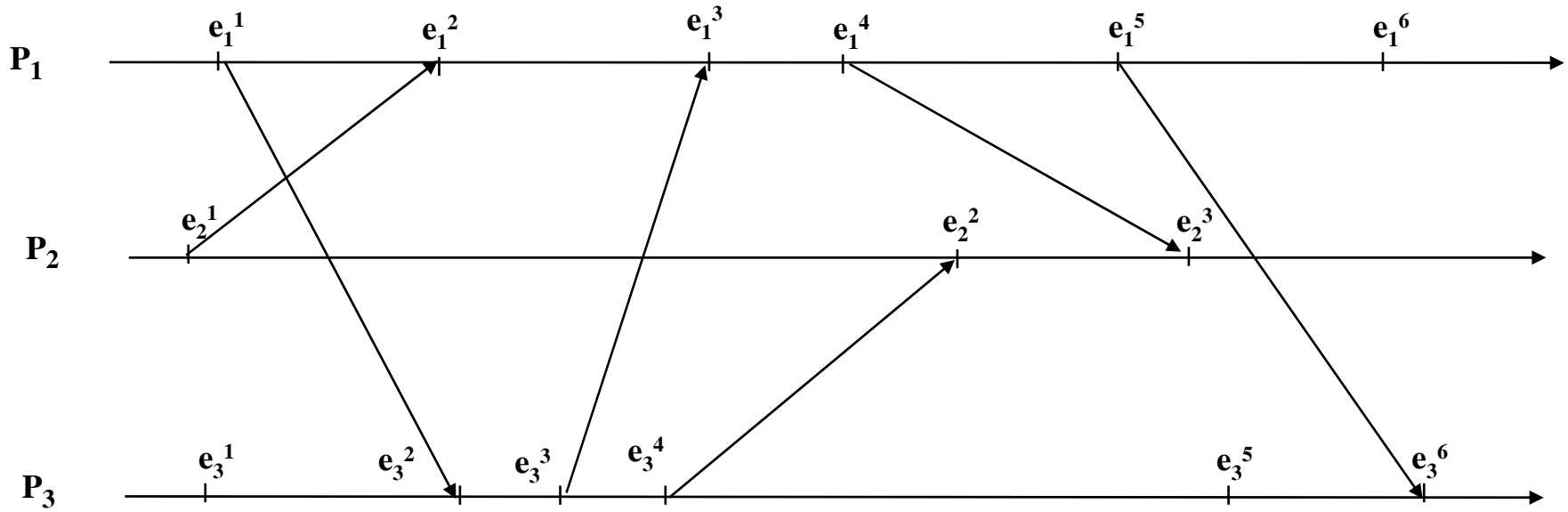
Berechnungsmodell (cont.)

Def.:
Die lokale Historie des Prozesses p_i ist eine (möglicherweise unendliche) Folge von Ereignissen $h_i = e_i^1\ e_i^2\ e_i^3\ \ldots\ e_i^n\ \ldots$ (kanonische Aufzählung).
Sie definiert eine totale Ordnung der lokalen Ereignisse.

Def.:
Die globale Historie ist die Menge $H = h_1 \cup h_2 \cup h_3 \cup \ldots\ldots\ldots \cup h_n$.

Bem.: die globale Historie spezifiziert keinerlei relative Zeit oder Ordnung zwischen Ereignissen.

Raum-Zeit-Diagramm



$e_1^2 \rightarrow e_3^6$

$e_3^1 \parallel e_1^2$

Was bedeutet Konsistenz in einem verteilten System

Ein Zustand, der durch irgendeine mögliche Ausführung von Prozessen zustande kommen kann. Insbesondere muss die Kausalität gewährleistet werden.

Globaler Zustand und Schnitte

Lokaler Zustand:
z_i^k : lokaler Zustand von p_i nach Ausführung von e_i^k

z_i^0 : Initialzustand von p_i

Globaler Zustand: $\Sigma = (z_1 , z_2 ,. . . . , z_n)$

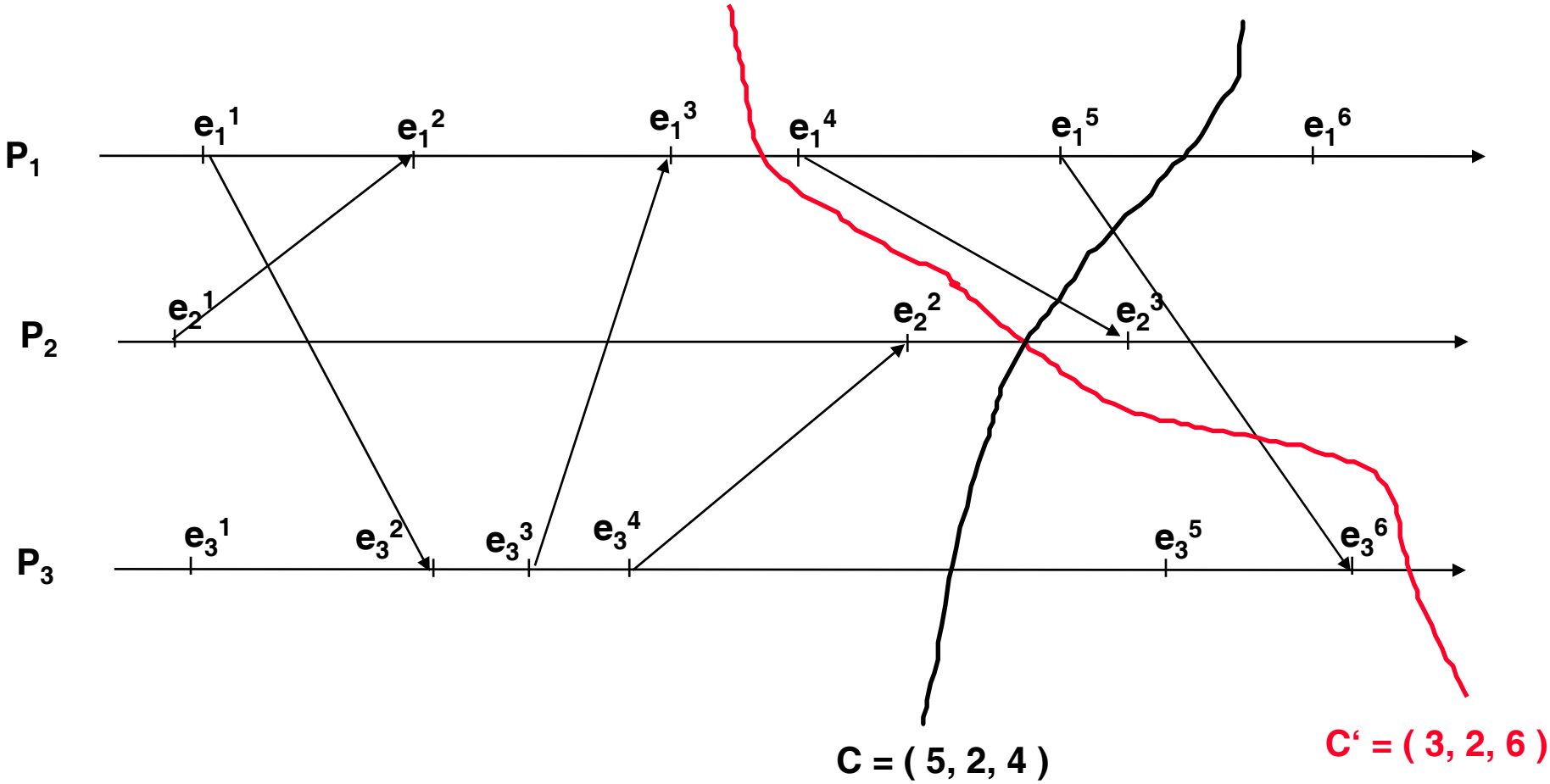
Def.: Der *Schnitt C* einer verteilten Berechnung ist die Untermenge C der globalen Historie H mit:

$$C = h_1^{C_1} \cup h_2^{C_2} \cup h_3^{C_3} \cup . . . . . \cup h_n^{C_n} .$$

Das Tupel natürlicher Zahlen $(c_1 , c_2 ,. . . . , c_n)$ repräsentiert dabei den jeweiligen Index des letzten Ereignisses, das für jeden Prozeß berücksichtigt wurde. Die Menge der letzten Ereignisse $\{ e_1^{C_1}, e_2^{C_2}, e_3^{C_3}. . . . . ., e_n^{C_n} \}$ wird die *Front* des Schnitts genannt.

Dem Schnitt, definiert durch $(c_1 , c_2 ,. . . . , c_n)$ ist ein globaler Zustand $(z_1^{C_1}, z_2^{C_2}, z_3^{C_3}. . . . . ., z_n^{C_n})$ zugeordnet.

Globaler Zustand und Schnitte



*Ausführung*en

Def.: Ausführung (Run) einer verteilten Berechnung:

Die Ausführung einer verteilten Berechnung ist eine total geordnete Folge R, die alle Ereignisse der globalen Historie enthält und mit jeder lokalen Historie übereinstimmt.

Konsistenz

1.) Ein Schnitt ist konsistent, wenn gilt:
für alle e, e‘ gilt: (e $\in$ C) und (e‘ $\rightarrow$ e) $\Rightarrow$ e‘ $\in$ C

2.) Ein konsistenter Zustand wird durch einen konsistenten Schnitt repräsentiert.

Analogien zwischen sequentiellen und verteilten Berechnungen:

I. **Zeitpunkt eines Ereignisses in einer sequentiellen Berechnung.**
Front eines konsistenten Schnitts in einer verteilten Berechnung.

II. **e liegt *vor* oder *nach* einem bestimmten Zeitpunkt in einer sequentiellen Berechnung**
e liegt *vor* oder *nach* einem Schnitt C, wenn es *links* oder *rechts* der Front von C liegt.

Die Ausführung R einer verteilten Berechnung heißt konsistent, wenn für alle Ereignisse gilt:
e $\rightarrow$ e‘ impliziert, daß e vor e‘ erscheint. (R definiert eine totale Ordnung)

Eine Ausführung R = e_1 , e_2 , . . . resultiert in einer Folge globaler Zustände Σ^0, Σ^1,Σ^2 . . .
wobei Σ^0 den globalen Anfangszustand (z_1^0, z_2^0, z_3^0. . . . . ., z_n^0 ; z_i^k : lokaler Zustand von p_i nach Ausführung von e_i^k) bezeichnet.

⇨ **Eine konsistente Ausführung durchläuft nur konsistente globale Zustände**
Eine Ausführung bezeichnet die Folge von Ereignissen *UND* die Folge konsistenter globaler Zustände

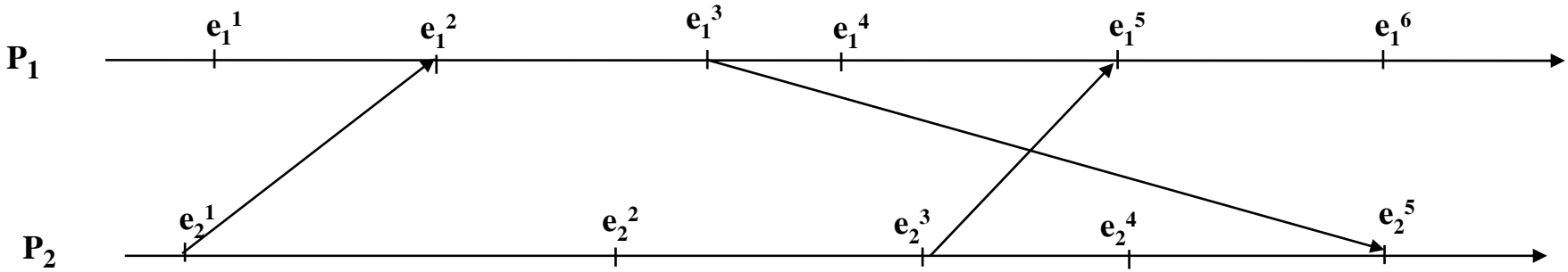
Erreichbarkeit:

1.) Jeder konsistente globale Zustand Σ^i wird aus einem vorhergehenden Zustand Σ^{i-1} durch das Auftreten eines einzelnen Ereignisses e^i hergeleitet . Für zwei solche konsistenten globalen Zustände einer Ausführung R sagt man Σ^{i-1} führt zu Σ^i in R.

2.) Sei $\rightarrow_R$ die transistive Hülle der "führt zu" Relation (d.h. die Menge aller von Σ^i direkt oder über Zwischenzustände erreichbaren Zustände) in einer gegebenen Ausführung R.

Σ' heißt erreichbar von Σ in der Ausführung R genau dann wenn $\Sigma \rightarrow_R \Sigma'$.

Beisp. Konstruktion des Gitters erreichbarer konsistenter Zustände



$\Sigma^{00} = (z_1^0, z_2^0)$, $\Sigma^{01} = (z_1^0, z_2^1)$, $\Sigma^{10} = (z_1^1, z_2^0)$, $\Sigma^{11} = (z_1^1, z_2^1)$, ...

I.) Jeder globale Zustand ist von Σ^{00} erreichbar.

II.) Ein *Pfad* durch das Gitter ist eine Folge globaler Zustände mit aufsteigenden Index, wobei sich der Index zweier aufeinanderfolgenden Zustände um 1 unterscheidet.

III.) Jeder Pfad entspricht einer konsistenten Ausführung einer Berechnung.

(z.B. Σ^{00}, Σ^{01}, Σ^{11}, Σ^{21}, Σ^{31}, Σ^{32}, Σ^{42}, Σ^{43}, Σ^{44}, Σ^{54}, Σ^{64}, Σ^{65})

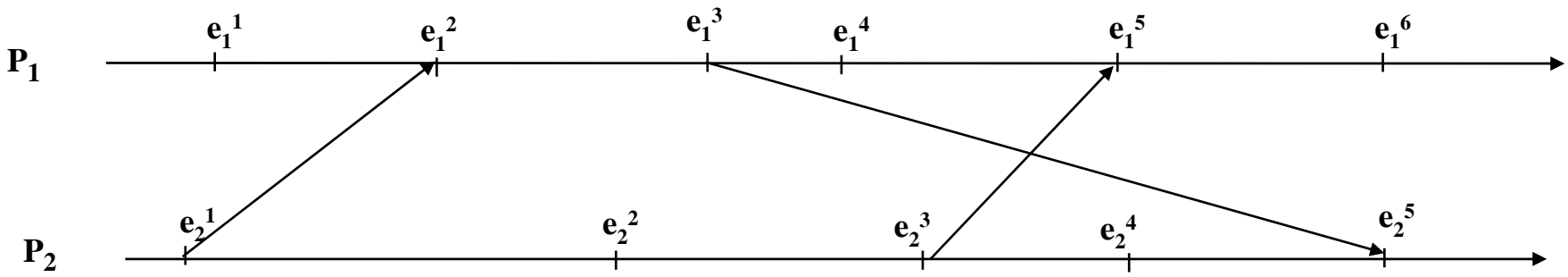
Bemerkungen:

Normalerweise ist das Gitter der konsistenten Zustände nicht bekannt.

Auch die tatsächlichen globalen Zustände, die während einer Berechnung durchlaufen werden, können nicht ohne zusätzliche Hilfsmittel bestimmt werden. (z.B. ob e_2^2 vor e_1^3 erfolgt).

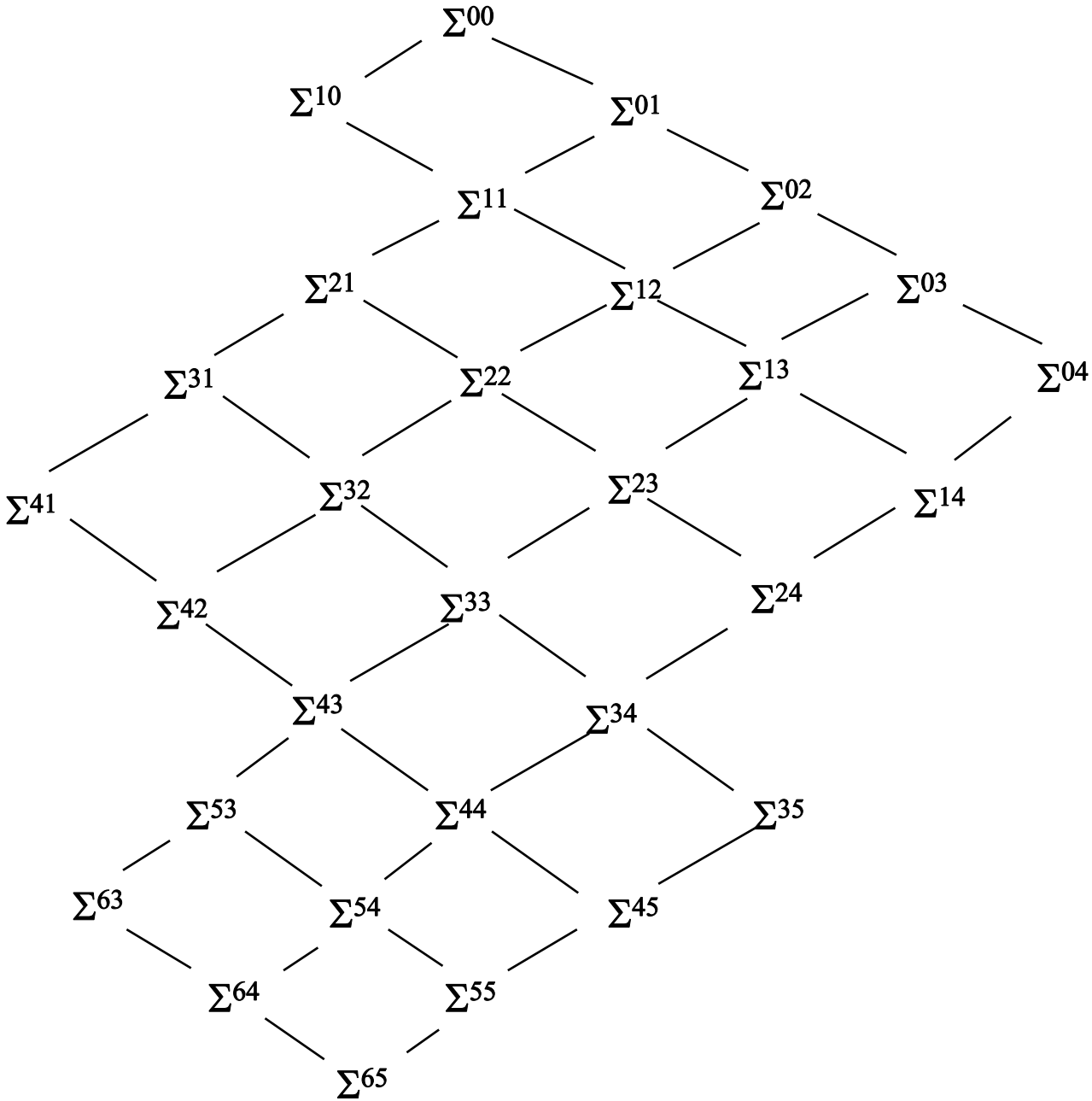
Nur ein allwissender externer Beobachter ist in der Lage, die Folge globaler Zustände zu identifizeieren, die eine Ausführung durchläuft.

Beisp. Konstruktion des Gitters erreichbarer konsistenter Zustände



$\Sigma^{00} = (z_1^0, z_2^0)$, $\Sigma^{01} = (z_1^0, z_2^1)$, $\Sigma^{10} = (z_1^1, z_2^0)$, $\Sigma^{11} = (z_1^1, z_2^1)$, ...

Gitter der konsistenten Zustände



II.

Nach welchen Gesichtspunkten kann man Ereignisse ordnen?

Wie kann man die Ordnung bestimmen und durchsetzen?

Algorithmen zur Konstruktion einer konsistenten Sicht

Bisher: **Existenz eines allwissender Beobachter, d.h. keine Annahmen darüber, *wie* Informationen gewonnen werden.**

Problem: **Nachrichten können verzögert werden, so dass sie in beliebiger Reihenfolge beim Empfänger ankommen.**

Ziel: **Konstruktion einer konsistenten Reihenfolge durch den Empfänger von Nachrichten.**

Ansatz: **Umordnung der Nachrichten im Empfangsprozess anhand der durchzusetzenden Ordnungsstrategie.**

Verfahren zur Ordnung von Nachrichten

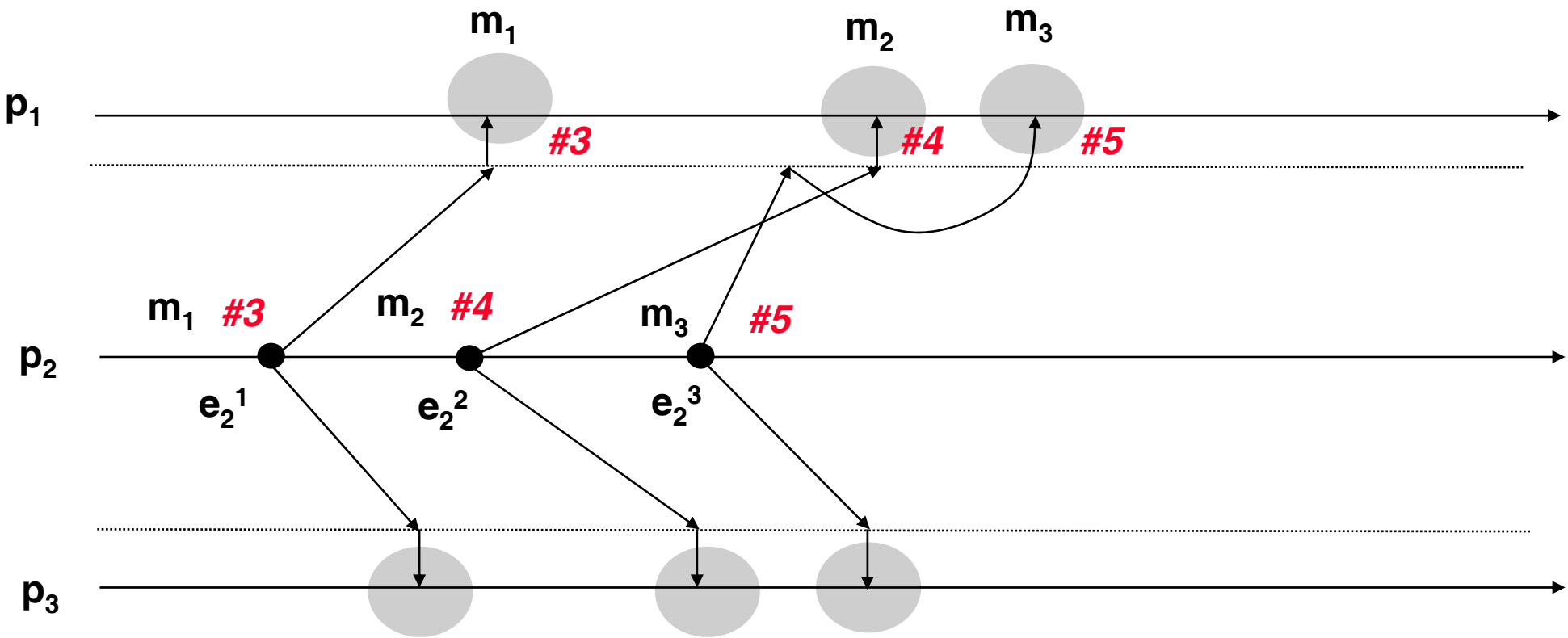
Zeitliche Ordnung: Nachrichten sind so geordnet, dass die Nachricht m_1, die zeitlich vor der Nachricht m_2 gesendet wird, auch vor m_2 ankommt.

Problem: Nachrichten können verzögert werden, so dass sie in beliebiger Reihenfolge beim Empfänger ankommen.

Ziel: Konstruktion einer konsistenten Reihenfolge durch den Empfänger von Nachrichten.

Ansatz: Umordnung der Nachrichten im Empfangsprozess anhand der durchzusetzenden Ordnungsstrategie.

FIFO-Empfangsordnung für Prozeßpaare



lokale Nachrichtenempfangswarteschlange

FIFO-Empfangsordnung für Prozeßpaare

Umordnung der Nachrichten im Empfangsprozeß.

Vorgehensweise: Unterscheidung des *Zustellens* einer Nachricht an den Anwendungsprozeß vom *Empfangen* einer Nachricht.

FIFO-Zustellung **(delivery): $send_i(m) \rightarrow send_i(m') \Rightarrow deliver_j(m) \rightarrow deliver_j(m')$**

FIFO-Z. verhindert, daß eine Nachricht eine später gesendete Nachricht deselben Prozesses überholt.

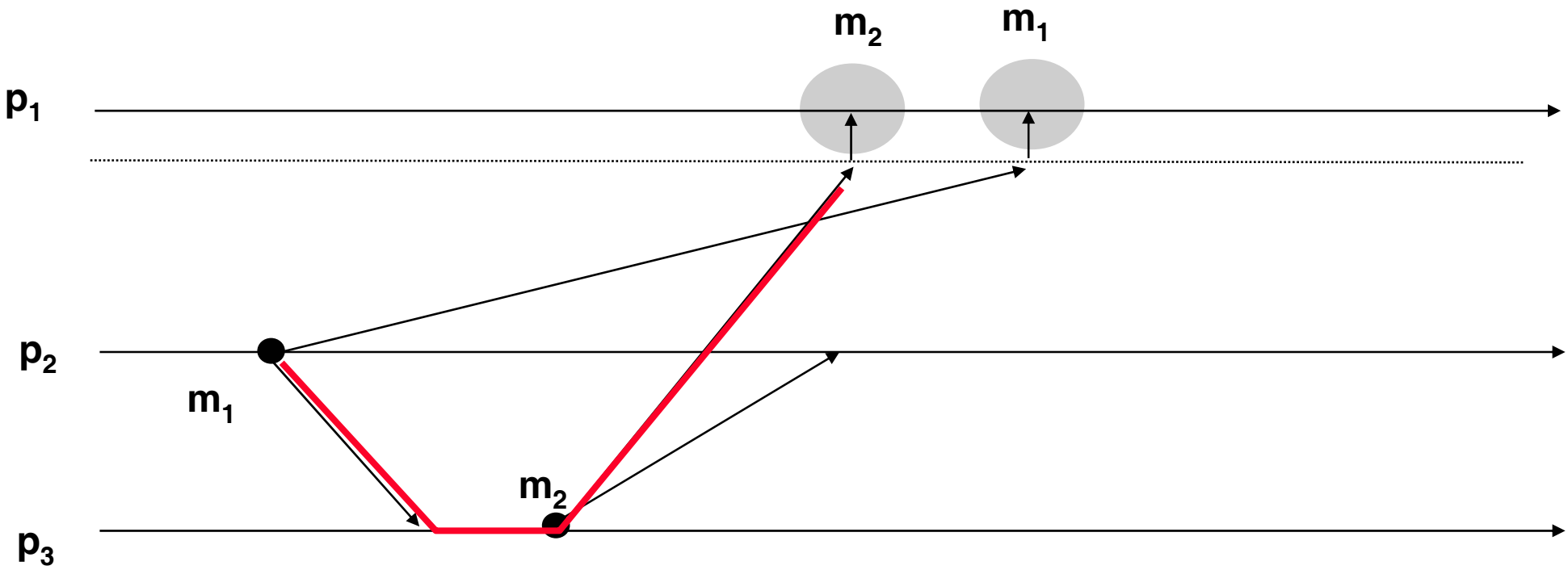
Zusätzlicher Aufwand: Prozeß muß eine Sequenznr. zu den Nachrichten hinzufügen

FIFO-Z. ist hinreichend, um zu garantieren, daß eine Beobachtung mit (irgend) einer Ausführung übereinstimmt, weil
FIFO-Z. garantiert, daß die Ordnung der lokalen Ereignisse erhalten bleibt

ABER: Da FIFO-Z. nur zwischen Prozeßpaaren definiert ist, ist es nicht hinreichend, um zu garantieren, daß die Beobachtung einer konsistenten Ausführung entspricht.

Ordnung
in Verteilten Systemen

FIFO-Empfangsordnung ist unzureichend



Die Reihenfolge der Ereignisse, die p_1 aufgrund der Nachrichtenfolge konstruiert, ist inkonsistent

Kausale Zustellung

Def. Kausale Zustellung:
Für alle Nachrichten m, m' und alle Prozesse p_i , p_j (sendende Prozesse) und p_k (empfangender Prozeß) gilt:

K-Zustellung **(CD): $send_i(m) \rightarrow send_j(m') \Rightarrow deliver_k(m) \rightarrow deliver_k(m')$**

CD erhält die globale kausale Ordnung aller Nachrichten im verteilten System.

Kausale Zustellung

Gegeben die in kausaler Abhängigkeit stehenden Ereignisse e und e'.

Um kausale Zustellung zu realisieren müssen wir in der Lage sein zu entscheiden:

Gibt es ein Ereignis e'' so daß gilt:

$$e \rightarrow e'' \rightarrow e'$$

?

Es ist notwendig, die Ereignisse entlang der kausalen Abhängigkeiten zu ordnen. Dabei definiert der rein zeitliche Vorrang nur eine potentielle Abhängigkeit.

Forderung:

1. **Die Ordnung der Nachrichten soll auf allen Knoten gleich sein**

2. **Die Ordnung der Nachrichten soll kausale Abhängigkeiten korrekt wiedergeben.**

Wie realisieren?

Ziel: Beobachter, der lokale Ereignisse in einen konsistenten globalen Ereignisstrom einordnet $\Rightarrow$ eine *totale Ordnung* herstellt.

Intuitive Lösung:
Herstellung der Ordnung über globale Zeit.

Annahmen:
Alle Prozesse haben Zugriff auf eine globale Echtzeituhr.
Die Kommunikationsverzögerung ist durch *d* begrenzt.

RC(e) ist der Wert der globalen Uhr, wenn das Ereignis e auftritt.
RC(e) wird als Zeitstempel (Timestamp) TS zur Nachricht an den Monitor-Prozeß hinzugefügt.

Zustellregel (delivery rule):

DR 1 : Zum Zeitpunkt t stelle alle empfangenen Nachrichten in aufsteigender Folge der TS mit TS = t - d zu.

(I.) **Durch die Begrenzung d ist man sicher, daß alle Nachrichten, die vor t-d abgesendet wurden, mittlerweile auch empfangen wurden - oder, anders herum ausgedrückt - (I) daß keine frühere Nachricht noch unterwegs ist, die erst nach dem Zeitpunkt t empfangen wird.**

Warum ist durch DR 1 globale Konsistenz der Beobachtung gesichert ?

1. durch Bedingung (I)
2. Die Beobachtung O ist konsistent gdw.
die Clock Condition (CC) gilt : $e \rightarrow e'$ $\Rightarrow$ $RC(e) < RC(e')$.
Diese Bedingung ist sicher durch globale Zeit gewährleistet.

Nachteil dieses Ansatzes: sehr starke Annahmen wie globale Zeit und Begrenzung der Nachrichtenverzögerung.

Frage: Kann man globale Ordnung auch ohne globale Echtzeituhr herstellen ?

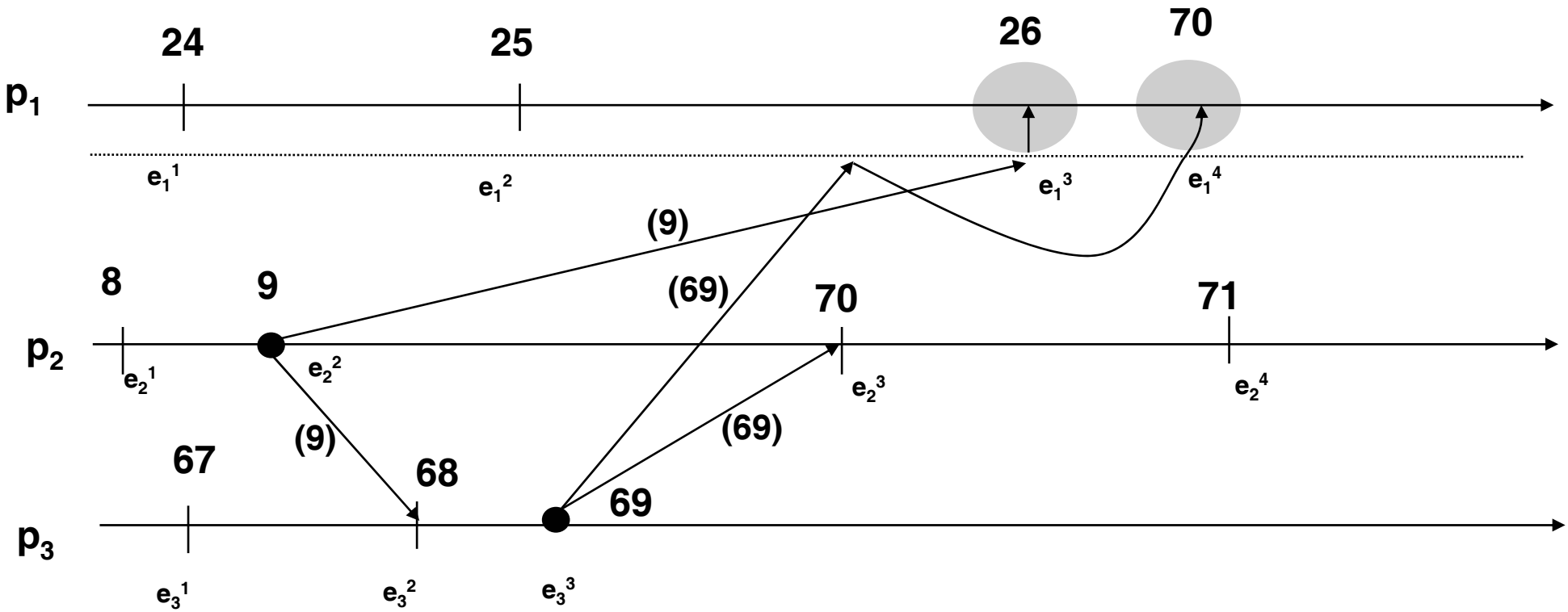
Logische Uhren (Lamport 1978)

Grundgedanke: Um eine konsistente Sicht des verteilten Systems zu erhalten, müssen *nur* kausale Abhängigkeiten berücksichtigt werden,

d.h.

Die Ordnung der Ereignisse basierend auf aufsteigenden Zeitwerten muß mit der kausalen Ordnung übereinstimmen.

Totale Ordnung durch logische Uhren



Logische Uhren (cont.)

- Jeder Prozeß unterhält eine lokale Variable LC, die seine individuelle logische Uhr darstellt. Die logische Uhr bildet Ereignisse auf positive, ganze Zahlen ab.
- $LC(e_i)$: Wert der logischen Uhr von Prozeß p_i, wenn das Ereignis e_i erzeugt wird.
- Jede Nachricht m, die gesendet wird, enthält den Zeitstempel TS(m), der den Wert der logische Uhr des sendenden Prozesses darstellt.
- Initialisierung: Bevor irgendein Ereignis erzeugt wird, werden alle Uhren auf "0" gesetzt.
- Die folgende "Update-Regel" definiert, wie die logische Uhr durch ihren Prozeß p_i beim Auftreten des Ereignisses e_i modifiziert wird:

$$LC(e_i) := \begin{cases} LC + 1 & \text{wenn } e_i \text{ intern oder ein Sende-Ereignis ist} \\ \max\{LC, TS(m)\} + 1 & \text{wenn } e_i \text{ ein Empfangs-Ereignis ist} \end{cases}$$

Logische Uhren (cont.)

Eigenschaften Logischer Uhren:

- **Lokale Uhren erzeugen immer aufsteigende Werte**

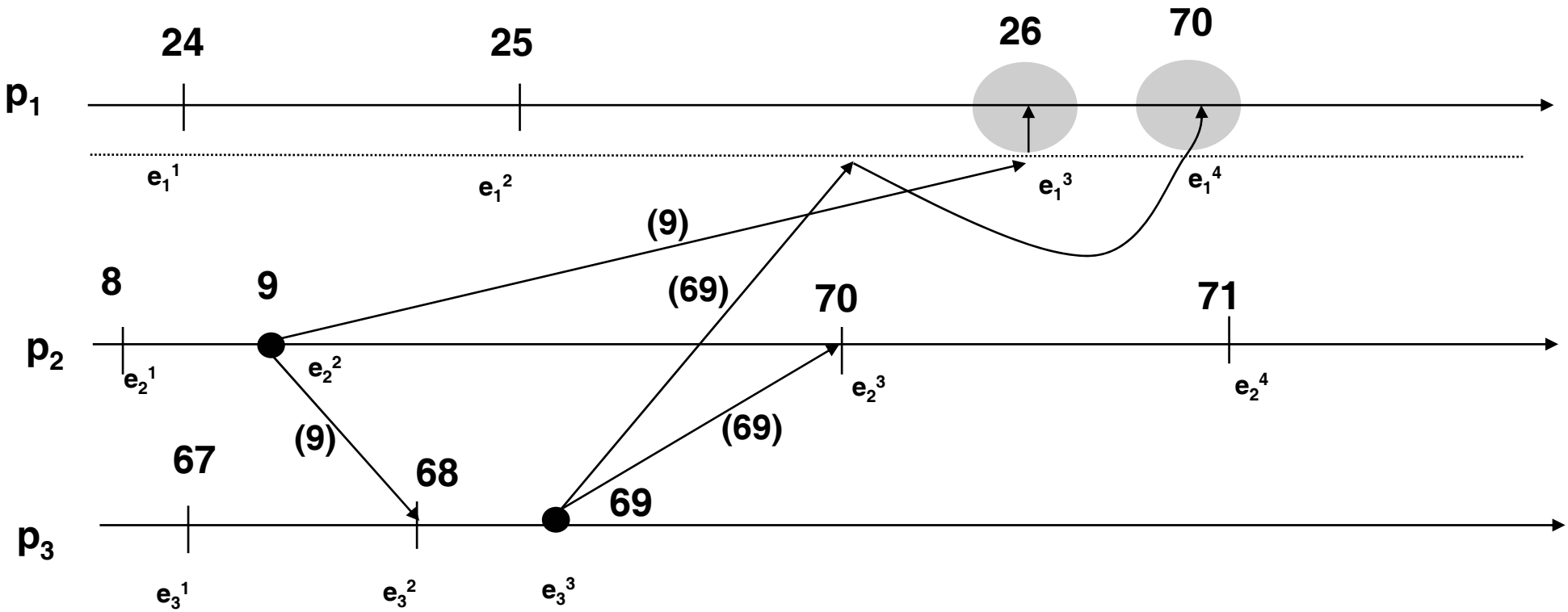
- **Logische Uhrenwerte sind aufsteigend im Hinblick auf kausale Ordnung**

- **Logische Uhren erfüllen die Bedingung : $e \rightarrow e' \Rightarrow LC(e) < LC(e')$.**

  schwache Clock Condition weil: $LC(e) < LC(e') \not\Rightarrow e \rightarrow e'$

Frage: Sind Logische Uhren hinreichend für konsistente Beobachtungen ?

Totale Ordnung durch logische Uhren kann korrekt hergestellt werden . . .



ABER: wenn keine max. Verzögerung durchgesetzt wird, kann man nie feststellen, ob eine Nachricht ausgeliefert werden darf !!

Gap-Detection Property (GDP):

Gegeben die Ereignisse e und e' mit den Uhrenwerten LC(e) und LC(e').
Es gilt LC(e) < LC(e').

Die GDP bezeichnet die Fähigkeit zu entscheiden, ob es ein Ereignis e'' gibt, so dass gilt: LC(e) < LC(e'') < LC(e')

Die GDP wird zur Garantie der Lebendigkeit benötigt.

Problem: Realisierung von Algorithmen mit den folgenden Eigenschaften:

1. **Alle Ereignisse sind total geordnet**
2. **Es kann aufgrund der Ereignisse entschieden werden, wann ein Ereignis ausgeliefert werden kann.**

Hinweis: Echtzeituhren allein lösen das Problem nicht !

Wie kann man die (kausale) Vergangenheit darstellen?

1. Ansatz: Jede Nachricht enthält alle Nachrichten ihrer Vergangenheit.

Vorteil: Beim Empfang einer Nachricht können alle Nachrichten ausgeliefert werden.

Nachteil: Offensichtlich

2. Ansatz: Jede Nachricht enthält die IDs aller Nachrichten ihrer Vergangenheit.

Vorteil: Beim Empfang einer Nachricht kann überprüft werden, ob alle kausal abhängigen Nachrichten empfangen wurden.

Nachteil: Nachrichten müssen in irgendeiner Instanz gespeichert und bei Bedarf zugestellt werden.

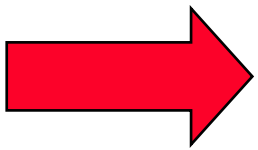
Problem:

Kausale Historien wachsen schnell. Mechanismus zum Löschen nicht mehr benötigter Nachrichten wird gesucht.

Frage:

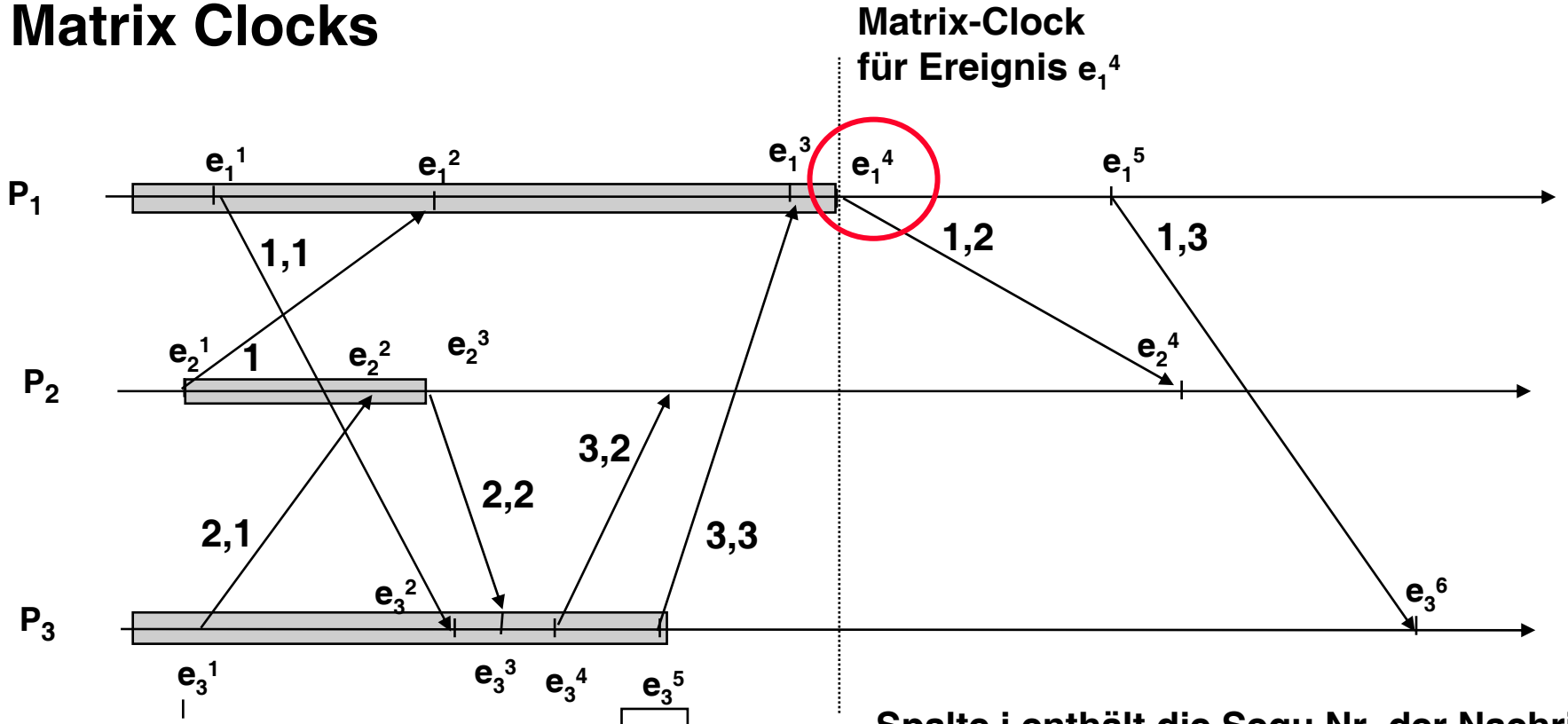
Was ist die minimale Menge an Information, um die kausale Historie repräsentieren zu können ?

Im allgemeinen Fall ist die minimale Kontrollinformation zur Darstellung der kausalen Historie n^2 wobei n die Anzahl der Prozesse ist.



Matrix-Clocks für Punkt-zu-Punkt Komm.
Vektor-Clocks für Broadcast Komm.

Matrix Clocks



	P_1	P_2	P_3
P_1	0	0	1
P_2	1	0	1
P_3	1	2	0

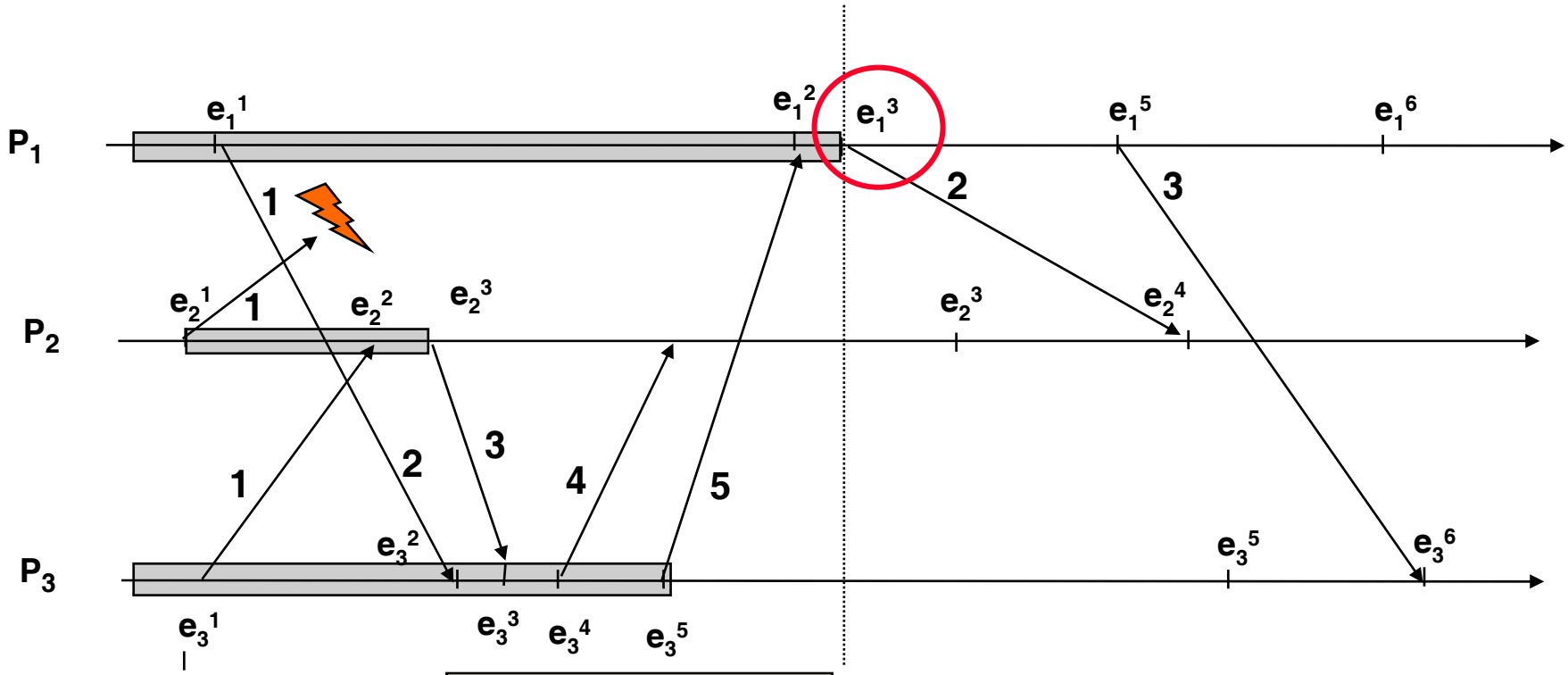
Zeile i enthält die Sequ.Nr. der Nachrichten, die von Prozess i an die Prozesse j gesendet wurden.

Spalte j enthält die Sequ.Nr. der Nachrichten, die von den Prozessen i an Prozess j gesendet wurden.

Es werden nur Sende-ereignisse betrachtet.

Mit jeder Nachricht wird eine Matrix-Clock verschickt, welche die lokale Sicht des Senders enthält.

Matrix Clocks

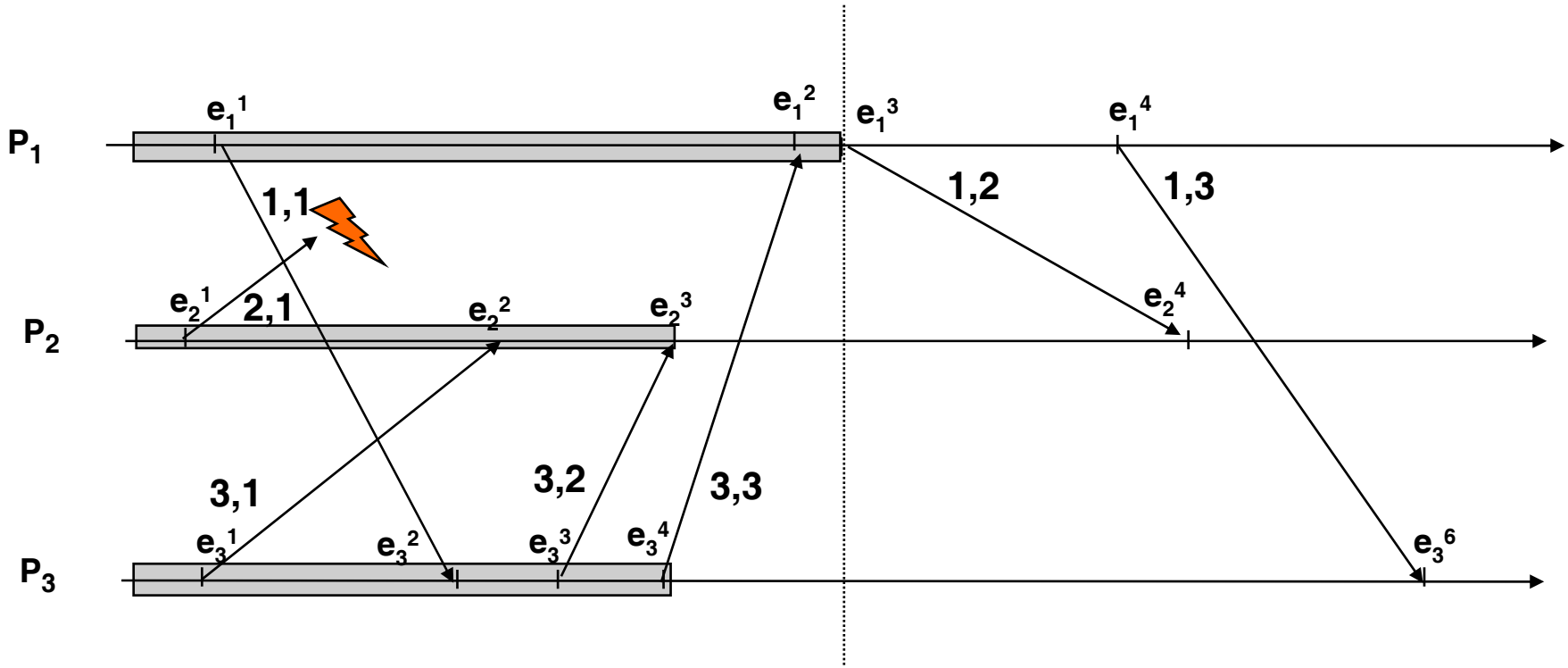


Matrix-Clock für Ereignis e_1^3

	P_1	P_2	P_3
P_1	0	0	1
P_2	1	0	1
P_3	1	2	0

Mit jeder Nachricht wird eine Matrix-Clock verschickt, welche die lokale Sicht des Senders enthält.

Matrix Clocks



Matrix-Clock für Ereignis e_1^4

	P_1	P_2	P_3
P_1	0	0	1
P_2	0	0	0
P_3	1	2	0

Fehlende Nchricht wird nicht erkannt, es besteht aber auch keine kausale Abhängigkeit.

Matrix Clocks (MC)

Interpretation der MC:

Zeile i repräsentiert die Sequenznr. der letzten Nachrichten, die von Prozess i an Prozess j gesendet wurden.

Spalte j repräsentiert die Sequenznr. der Nachrichten die Prozess j von Prozess i erhalten haben müsste.

Update Regeln:
Bei einem neuen Ereignis e_i^k von Prozeß p_i wird die MC folgendermaßen erhöht:

Sendeereignis: Prozess P_i sendet Nachricht (e_k) an Prozess P_j
MC [i,j] := MC [i,j] + 1

Empfangsereignis:
MC[i,j] := max {MC [i,j], TS(m)}

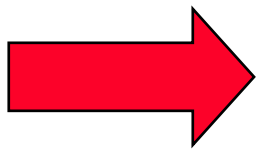
Matrix-Clocks

Problem:
Obwohl minimal, immer noch zu viel Kontrollinformation

Ziel:
Zusammenfassung der Kontrollinformation

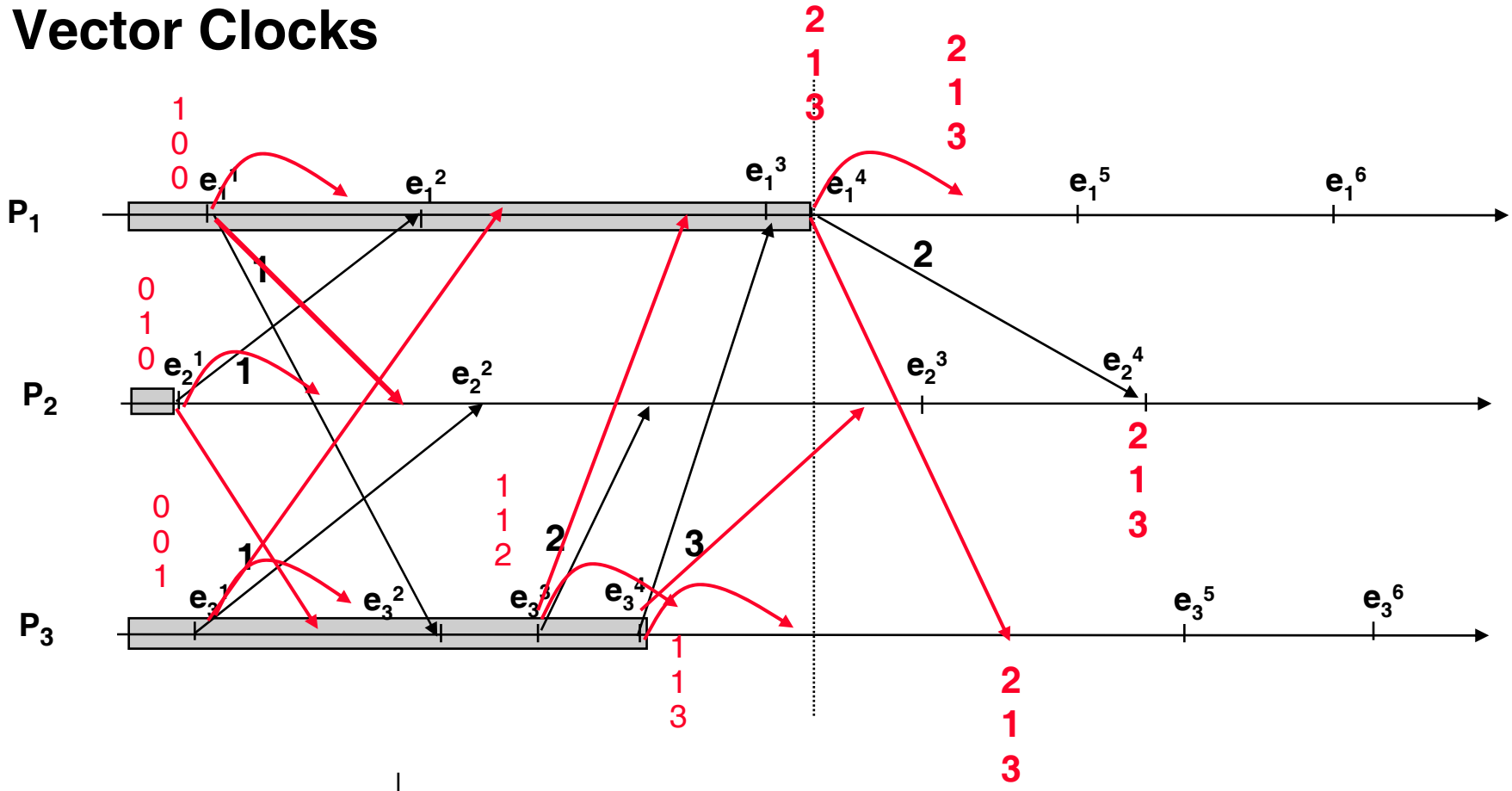
Voraussetzung:
Alle Prozesse senden Broadcast-Nachrichten

Folgerung:
Eine Zeile in der Matrix läßt sich in einem Wert zusammenfassen



Vektor-Clocks

Vector Clocks



Matrix-Clock für Ereignis e_1^4

	P_1	P_2	P_3
P_1	2	2	2
P_2	1	1	1
P_3	3	3	3

Es werden nur Sendeereignisse betrachtet.

Die Zeilen der Matrix können zu einem Wert zusammengefasst werden.

Vektor Clocks (VC)

Update Regeln:

Bei einem neuen Ereignis e_i von Prozeß p_i wird die VC folgendermaßen erhöht:

Sendeereignis:

$VC(e_i)[i] := VC[i] + 1$

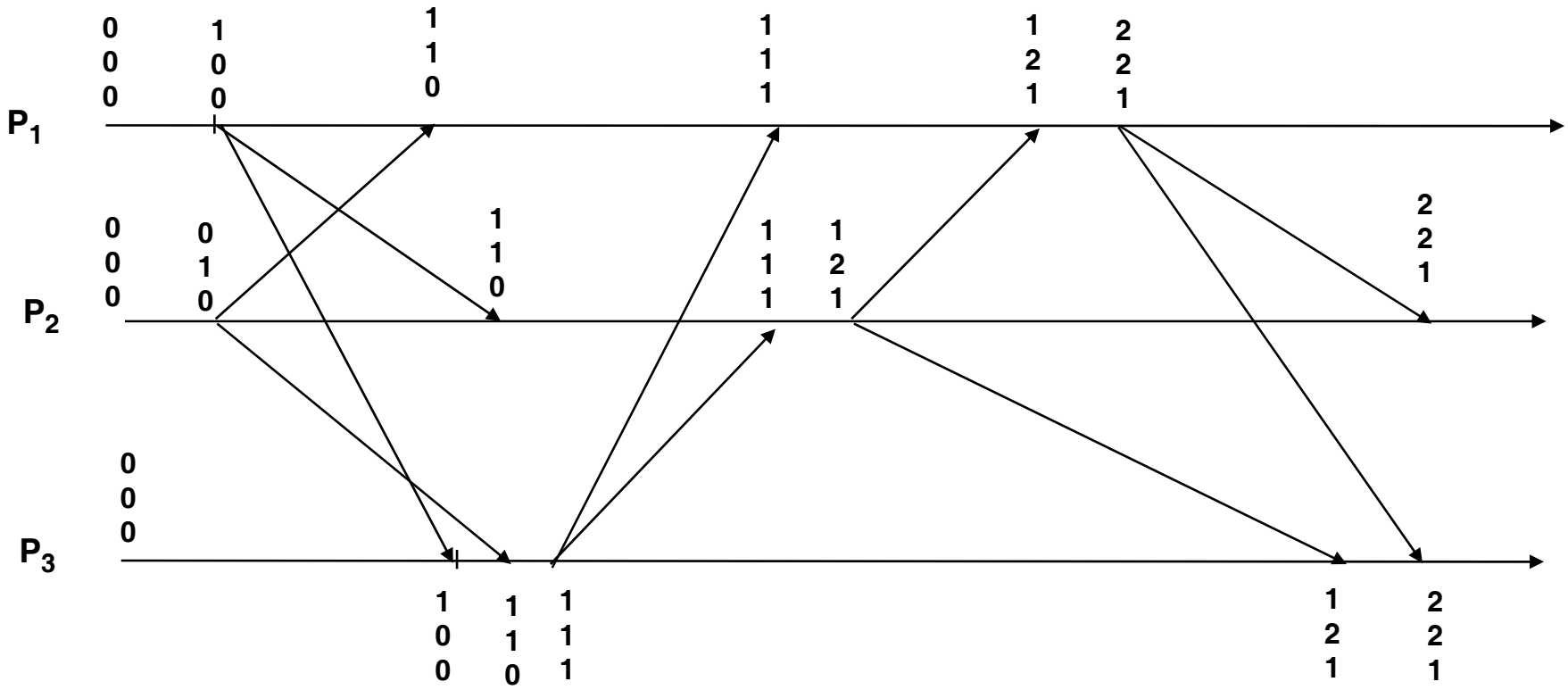
Empfangsereignis:

$VC(e_i) := \max\{VC, TS(m)\}$

Interpretation der VC:

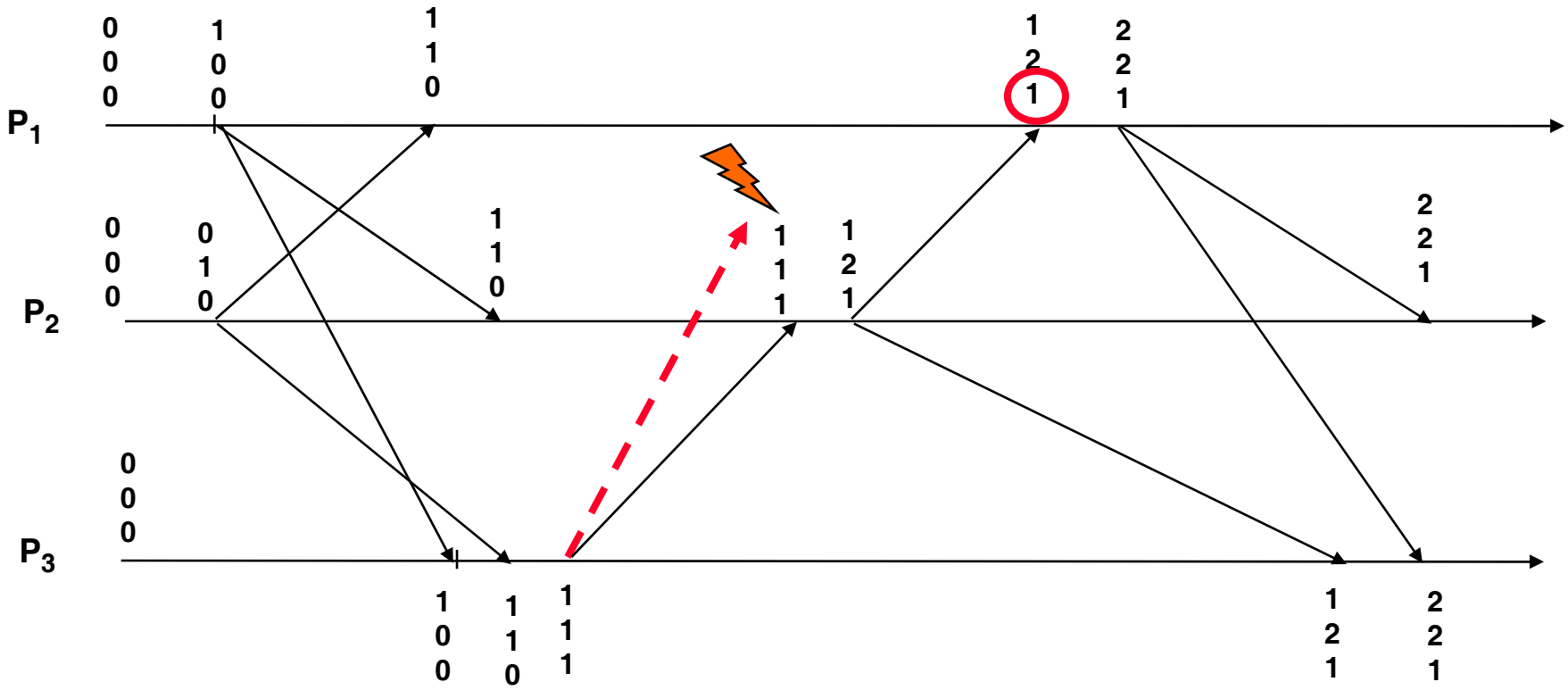
$VC(e_i)[j] \equiv$ Anzahl der Ereignisse von p_j, die Ereignis e_i von p_i kausal vorangehen.

Vektor Clocks



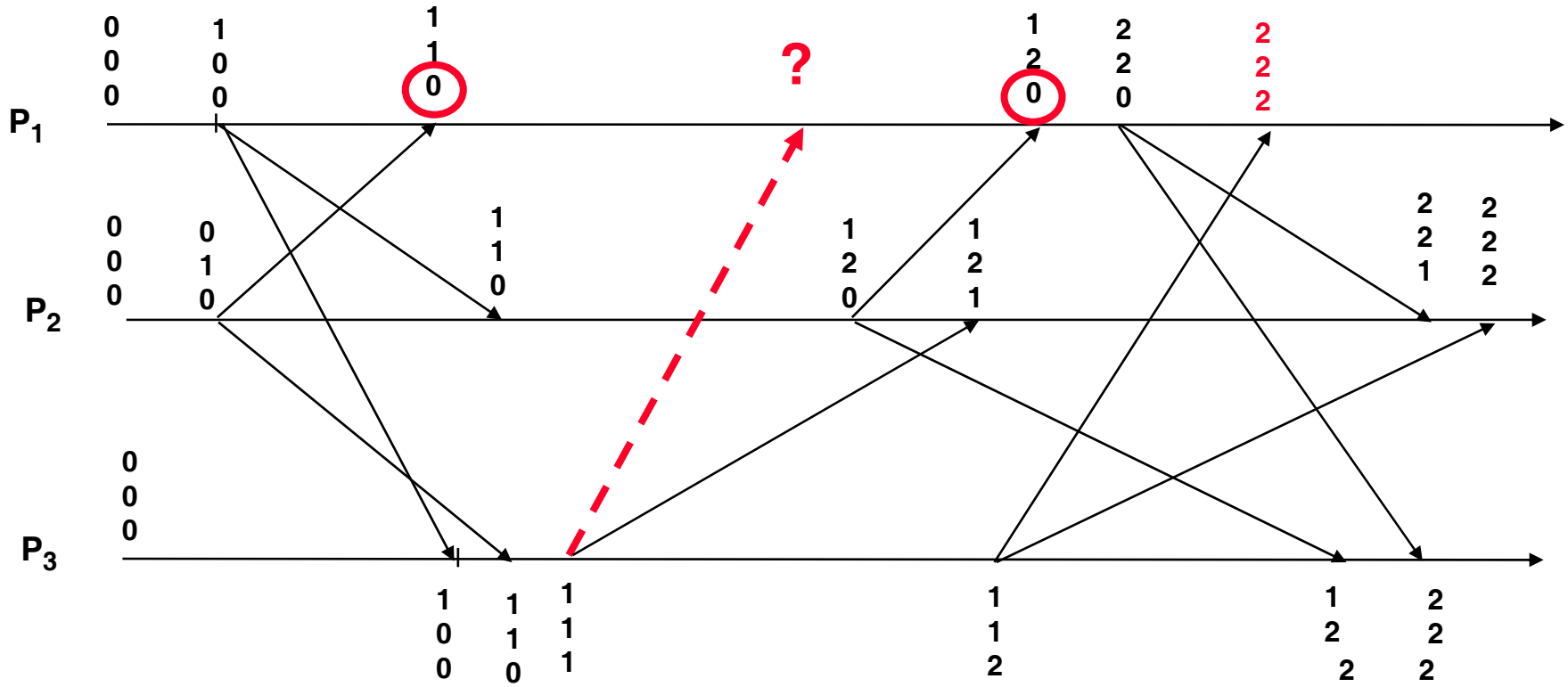
Für jedes Ereignis wird die lokale Sequenznr. mitgeführt und die Sequenznr. der Ereignisse anderer Prozesse, von denen das lokale Ereignis abhängt.

Vektor Clocks



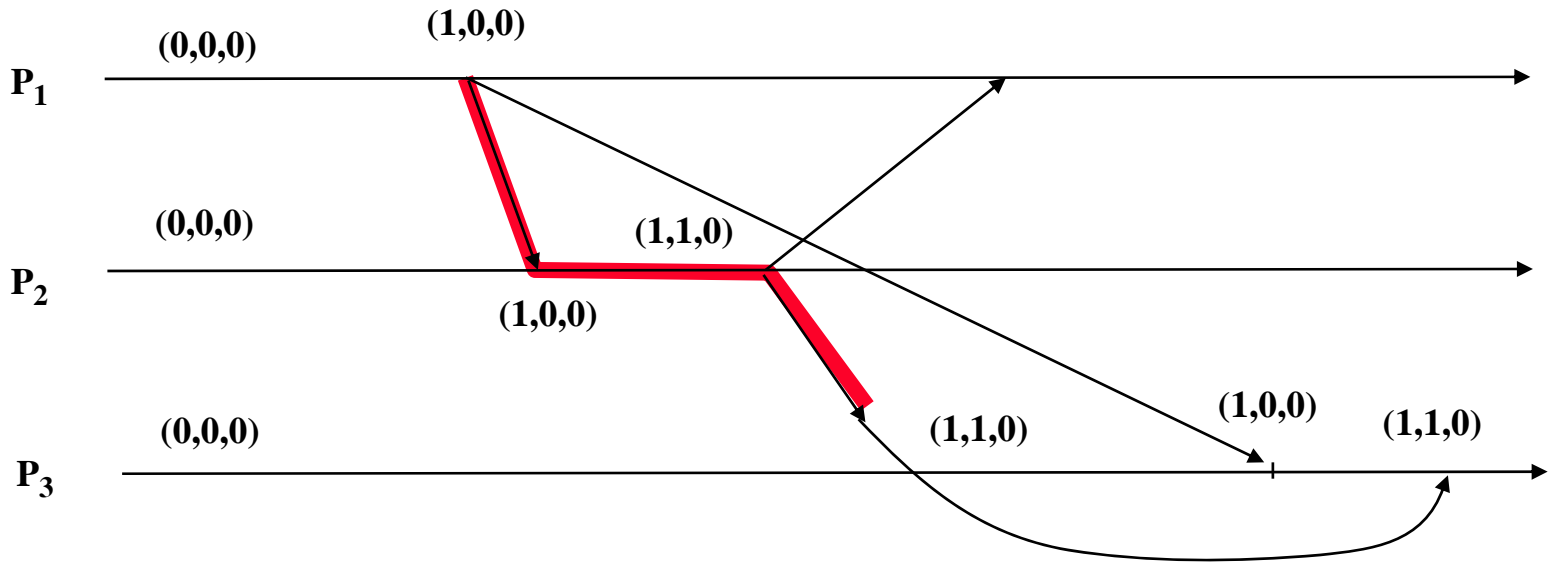
Nachricht geht verloren

Vektor Clocks



Wann darf eine Nachricht ausgeliefert werden?

Vektor-Clocks zur Durchsetzung der kausalen Abhängigkeit



Vektor Clocks

1.) **“Kleiner Relation “ :**

$$V < V' \Leftrightarrow (V \neq V') \text{ UND } (\text{für alle } k: 1 \leq k \leq n : V[k] \leq V'[k])$$

2.) **Starke Clock-Condition :**

$$e \rightarrow e' \Leftrightarrow VC(e) < VC(e')$$

3.) **Nebenläufigkeit:**

Gegeben: e_i **von** p_i
e_j **von** p_j

$$e_i \parallel e_j \Leftrightarrow (VC(e_i)[i] > VC(e_j)[i]) \text{ UND } (VC(e_j)[j] > VC(e_i)[j])$$

Vektor Clocks (cont.)

4.) **Komponentenweise Numerierung:**

Für alle $j \neq i$, gibt $VC(e_i)[j]$ die Anzahl der Ereignisse von P_i an, die dem Ereignis e_i von P_i vorangehen.

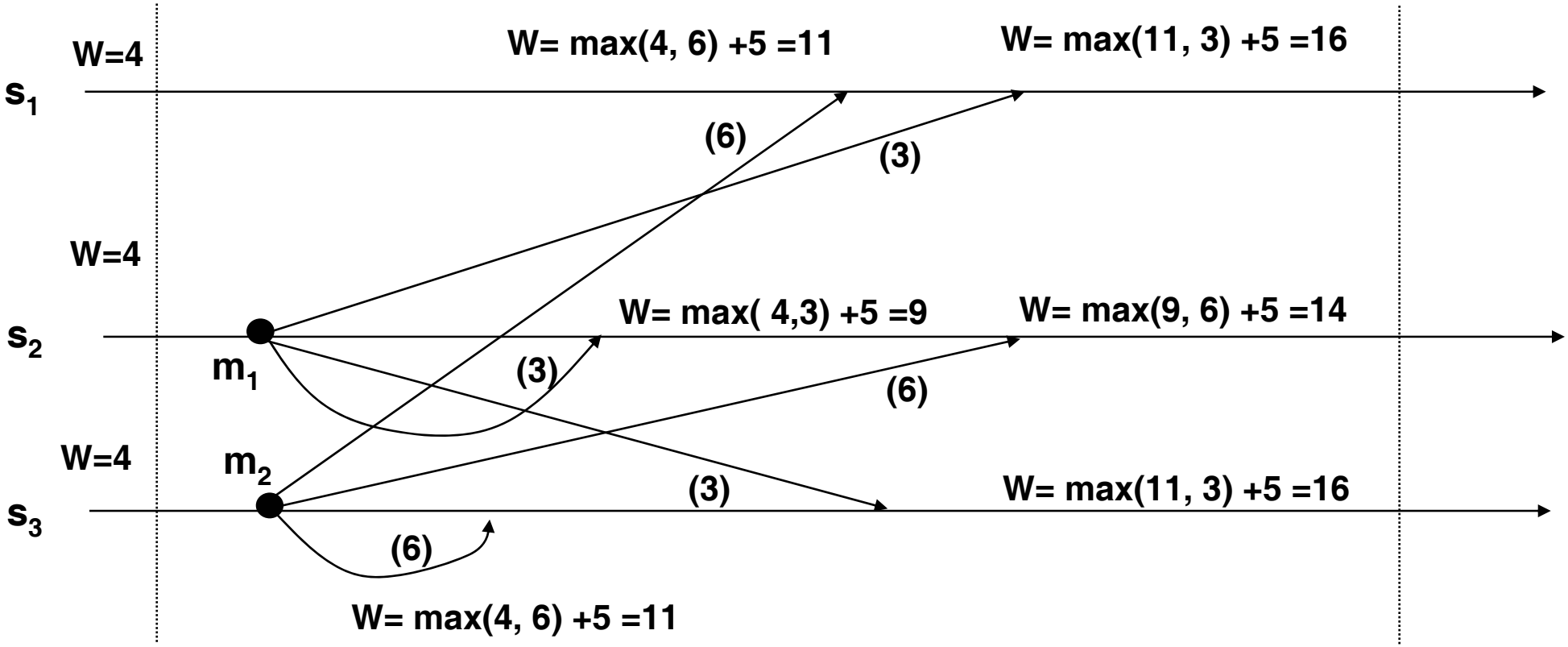
5.) **(Komponentenweise)Lücke zwischen zwei Ereignissen:**

Gegeben sei: $VC(e_i)[i] < VC(e_j)[i]$ für $j \neq i$
Es gilt: $e_i \rightarrow e_j$ und $VC(e_i)[i] - VC(e_j)[i]$ gibt die Anzahl der Ereignisse an, die bzgl. Prozeß P_i zwischen e_i und e_j liegen.

6.) **Lückenerkennung (Gap Detection) zwischen zwei Ereignissen:**

Wenn $VC(e_i)[i] - VC(e_j)[i] > 1$ für $j \neq i$
gilt: $e_i \rightarrow e_j$ und es gibt ein e_i' , so daß gilt:
$e_i \rightarrow e_i' \rightarrow e_j$, d.h. ein Ereignis in Prozeß P_i , das zwischen e_i und e_j liegt.

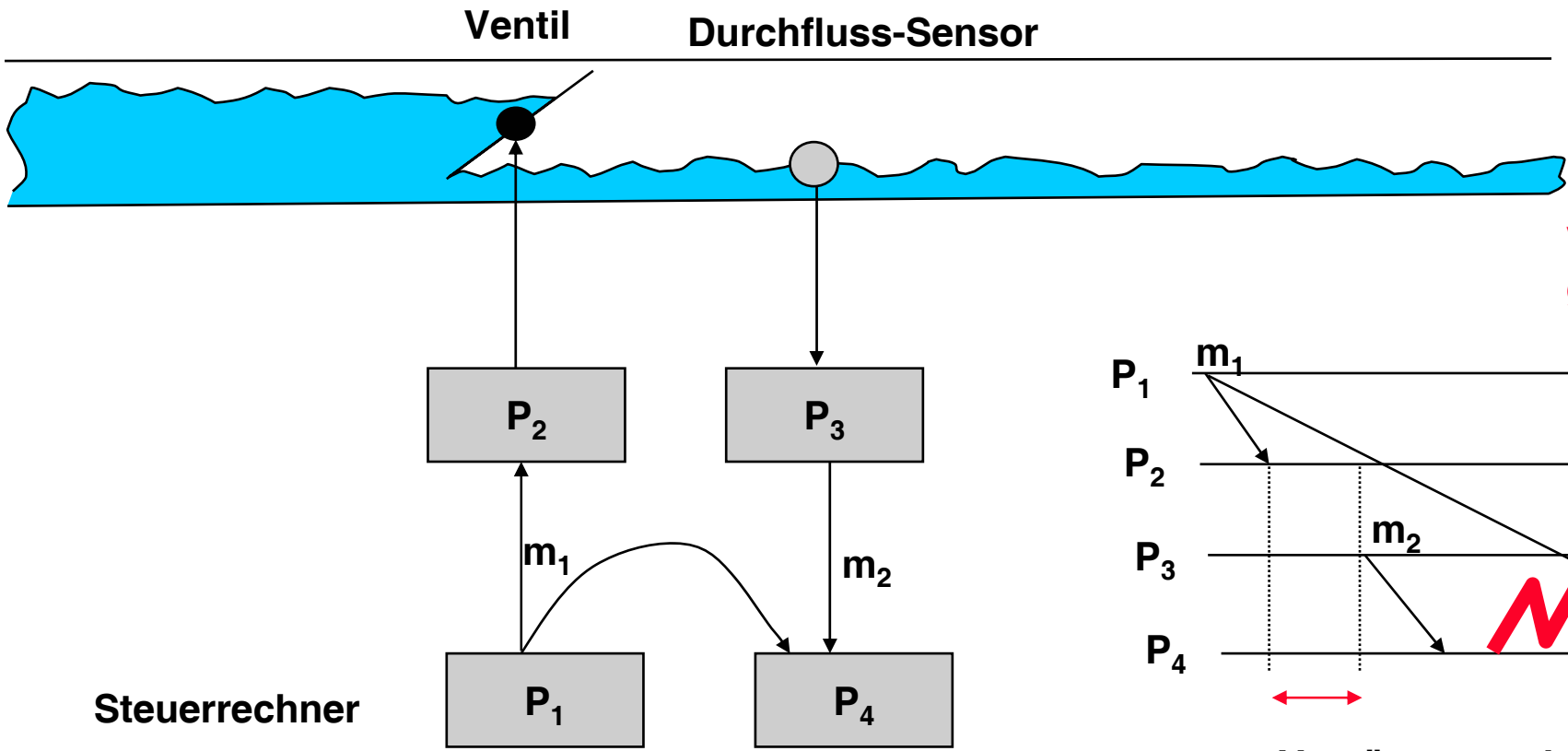
Was ordnet die logische Ordnung ?



Jeder Sensorprozeß s_i unterhält eine Variable W, die einen globalen Zustand z.B. der Umgebung darstellen soll. Ein neuer Wert wird bestimmt aus dem alten Wert von W und der Nachricht von den anderen Sensoren:
W_t = max (W_{t-1} , Sensornachricht) + 5

Was ordnet die logische Ordnung?

Das Problem der verdeckten physische Kanäle



Synchrone Systeme

Das Kommunikationssystem hat eine bekannte und garantierte maximale Nachrichtenverzögerung *d*.

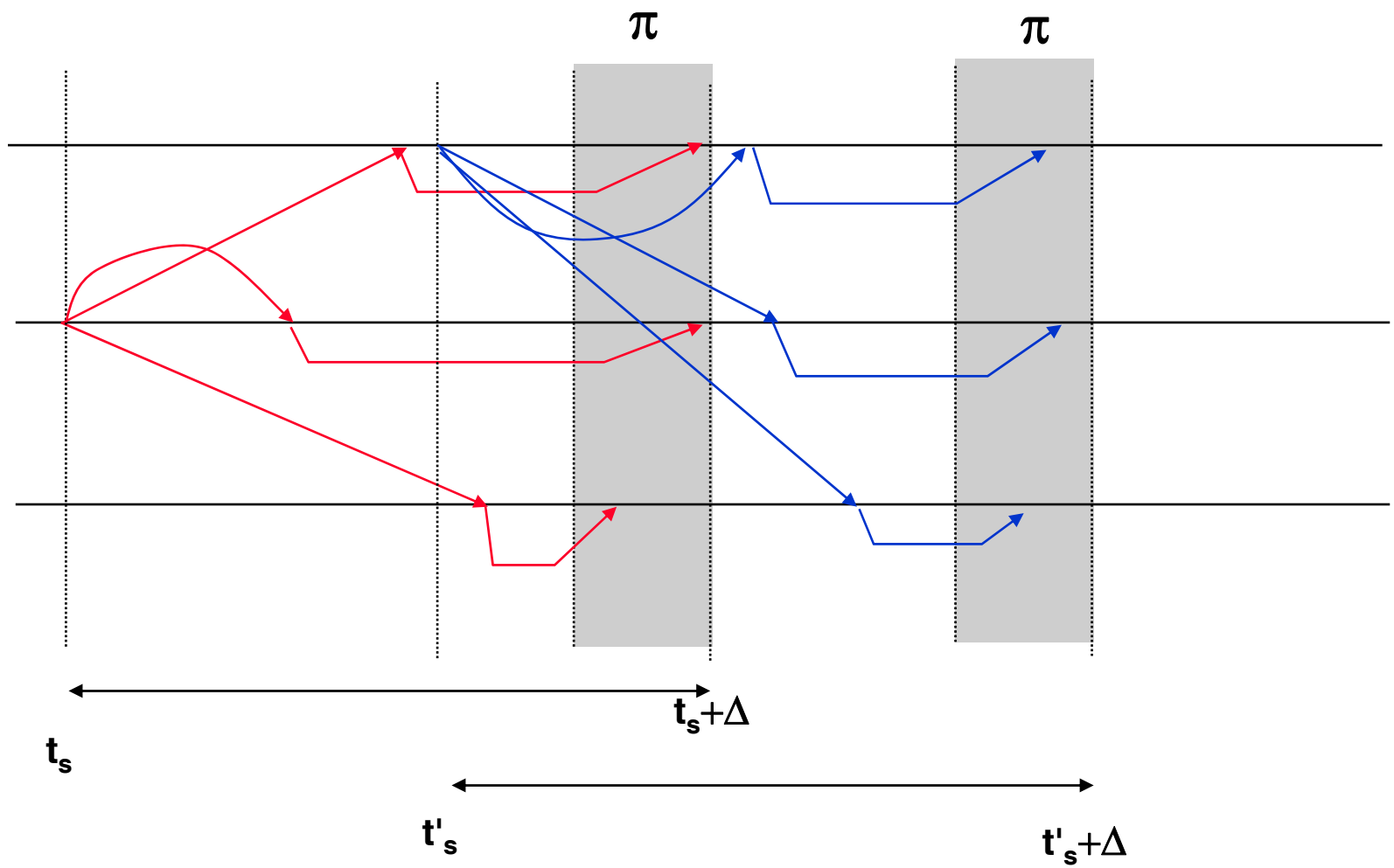
Alle Prozesse haben Zugriff auf eine globale Echtzeituhr (RC).

RC(e) ist der Wert der globalen Uhr, wenn das Ereignis e auftritt.
RC(e) wird als Zeitstempel (Timestamp) TS zur Nachricht an den Monitor-Prozeß hinzugefügt.

Zustellregel (delivery rule):

Zum Zeitpunkt t stelle alle empfangenen Nachrichten in aufsteigender Folge der TS mit TS = t - d zu.

Δ - Protokolle



Zeitliche Ordnung

Eine Nachricht m_1 wird als *zeitlich vorangehend* auf eine Nachricht m_2 bezeichnet, wenn m_1 mindestens δ vor m_2 gesendet wurde, d.h. gilt:

$$t(send(m_1)) - t(send(m_2)) > \delta$$

Nach dieser Definition garantiert ein Protokoll, das Nachrichten in zeitlicher Ordnung ausliefert auch die kausale Ordnung.