Data-Link Protokolle

Vortrag im Rahmen des Seminars “Verteilte Algorithmen”

von Jochen Koberstein

Einleitung

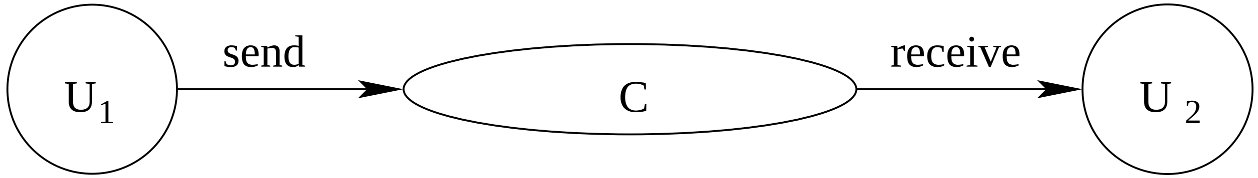
Der Vortrag basiert auf dem Buch “Distributed Algorithms” von *Nancy Lynch*

(erschienen bei *Morgan Kaufmann Publishers*).

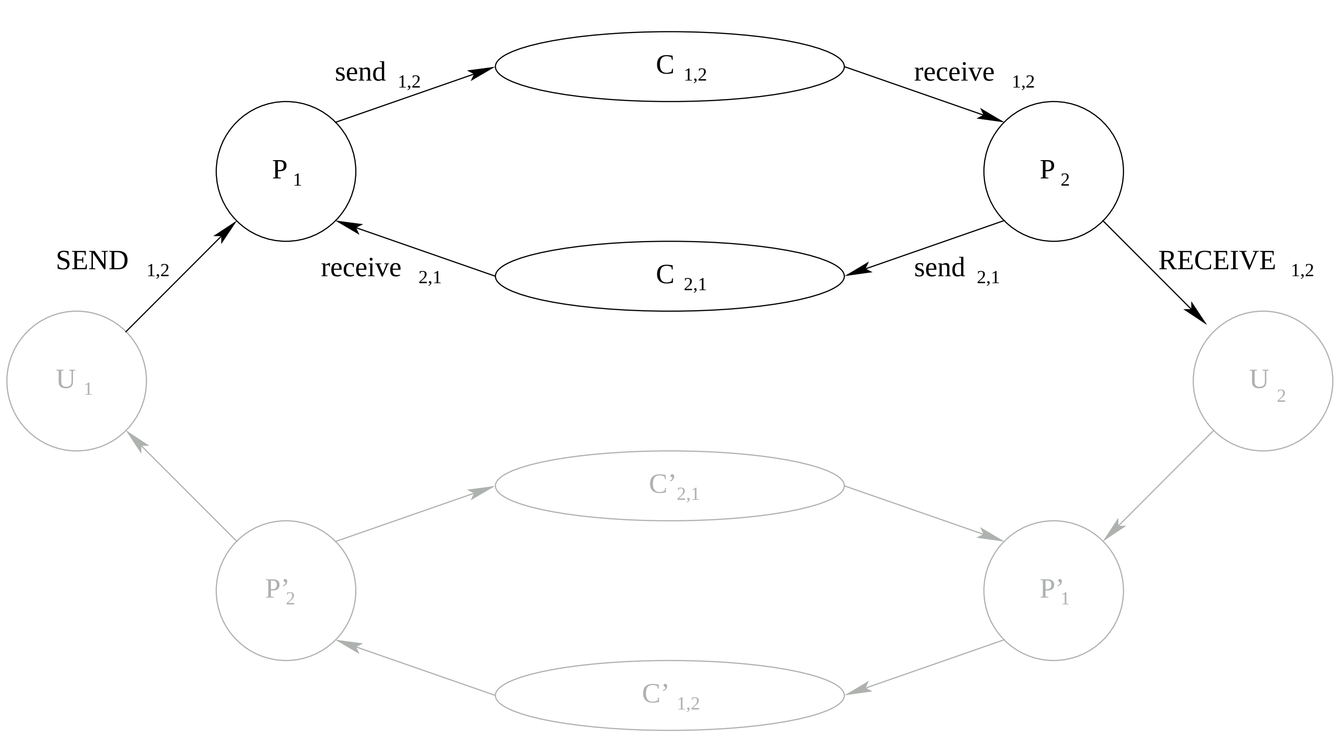
Was sind Data-Link Protokolle?

- Point-to-Point Kommunikation
- fehlerfreie Übertragung über fehlerbehaftete Verbindung
- Schicht 2 der OSI-Modells

$\leadsto$ Ziel: Realisierung eines *reliable FIFO channel*



1. Es werden nur korrekte Nachrichten übermittelt
2. Jedem *send* folgt ein *receive* Ereignis
3. Ein *send* Ereignis resultiert in (genau) einem *receive* Ereignis
4. Keine Umsortierung der Nachrichten im Kanal



- Externe Transitionen sind in Großbuchstaben (z.B. SEND), interne in Kleinbuchstaben (z.B. send) geschrieben.
- Es wird unterschieden zwischen *high-level* und *low-level* Nachrichten.
- Die (endliche) Menge der *high-level* Nachrichten wird mit M bezeichnet, die einzelne Nachricht mit $m \in M$
- Die Menge der *low-level* Nachrichten wird mit M' bezeichnet, die einzelne Nachricht mit $m' \in M'$

Rahmenbedingungen

- Es gibt nur zwei Knoten
- Folgende Fehlerarten sind zugelassen:
  - Verlust (loss)
  - Duplizierung (duplication)
  - Umsortierung (reordering)
- Nicht betrachtet werden Verfälschungen der Nachrichten

Die Duplizierung sei endlich.

Strong Loss Limitation (SLL)

Unendlich viele *send(m)* Ereignisse führen zu unendlich vielen *receive(m)* Ereignissen.

Weak Loss Limitation (WLL)

Unendlich viele *send* Ereignisse führen zu unendlich vielen *receive* Ereignissen.

- Einleitung
- Behandlung aller Fehler
  $\rightsquigarrow$ ggf. unendlich große *low-level* Nachrichten
- *low-level* Nachrichten endlicher Größe
  $\rightsquigarrow$ man betrachte drei Kombinationen der Fehlerarten
- Was geschieht, wenn Prozesse ausfallen?

Das Stenning Protokoll

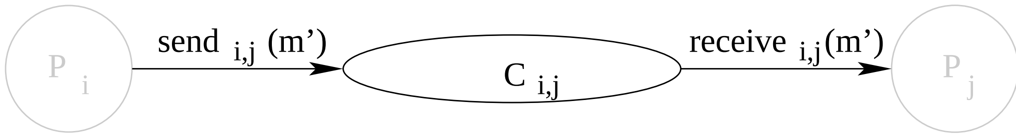
Erlaubte Fehler:

- (begrenzter) Verlust (WLL),
- (endliche) Duplizierung und
- Umsortierung

von Nachrichten

ABER ggf. unendlich große *low-level* Nachrichten

$\leadsto$ Kanäle als *lossy reordering FIFO channel* (siehe Kapitel 14, S. 463)



Signature:

Input:

send(m’)$_{i,j}$

Output:

receive(m’)$_{i,j}$

States:

in-transit, a multiset of elements of M’, initially empty

Transitions:

send(m’)$_{i,j}$

Effect:

add any finite number of copies of *m’* to *in-transit*

receive(m’)$_{i,j}$

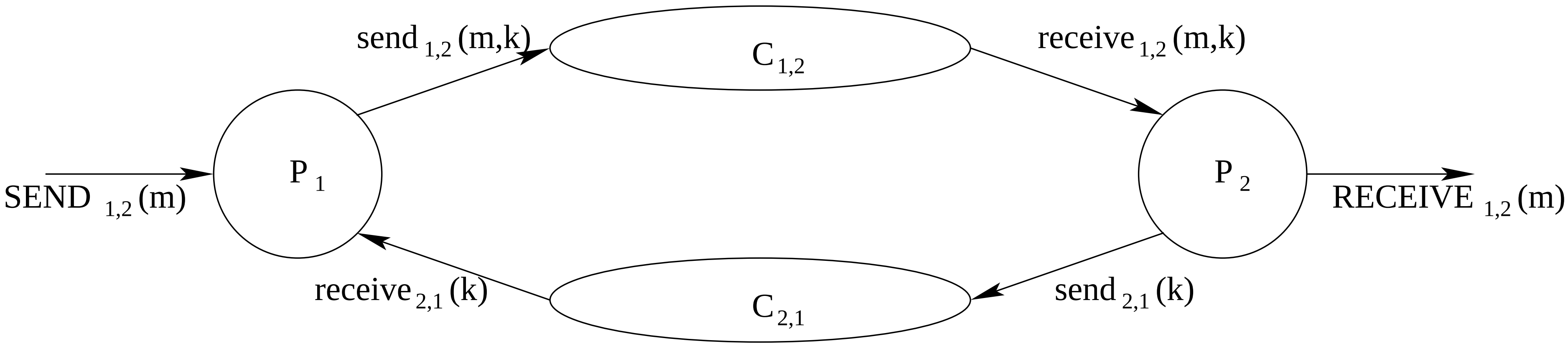
Precondition:

m’ $\in$ *in-transit*

Effect:

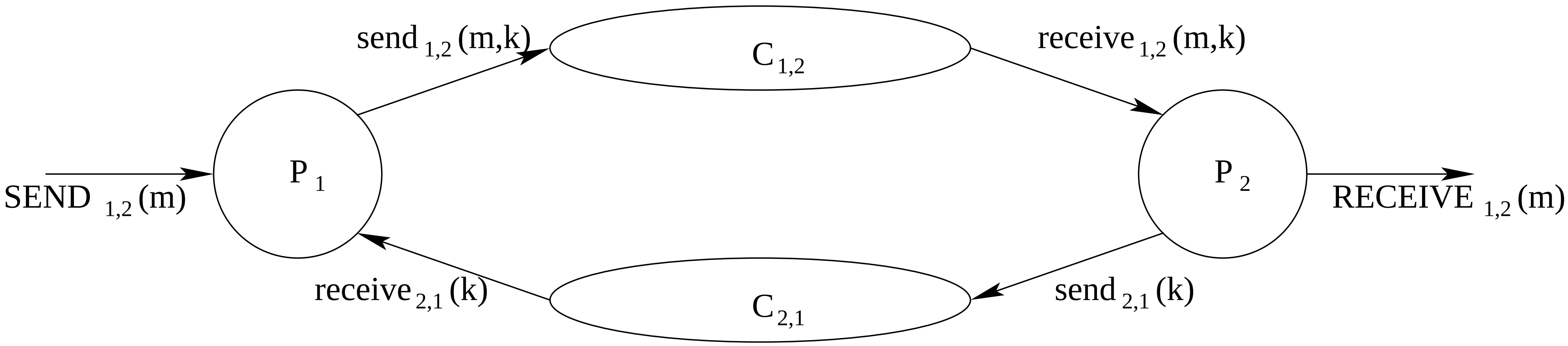
remove one copy of *m’* from *in-transit*

Tasks: {**receive(m’)**$_{i,j}$: m $\in$M’}



Signature P_1:

Input:	Output:
SEND$(m)_{1,2}$, $m \in M$	send$(m,k)_{1,2}$, $m \in M$, $k \in \mathbb{N}$
receive$(k)_{2,1}$, $k \in \mathbb{N}$	



Signature P_2:

Input:	Output:
$\text{receive(m,k)}_{1,2}$, $m \in M$, $k \in \mathbb{N}$	$\text{RECEIVE(m)}_{1,2}$, $m \in M$
	$\text{send(k)}_{2,1}$, $k \in \mathbb{N}$

States:

`buffer`, a FIFO queue of elements of M, initially empty

`tag` $\in \mathbb{N}$, initially 1

Transitions:

SEND(m)$_{1,2}$

Effect:

add m to `buffer`

send(m,k)$_{1,2}$

Precondition:

m is first in buffer

$k = tag$

Effect:

none

receive(k)$_{2,1}$

Effect:

if ($k = tag$) then

remove first element (if any) of buffer

$tag := tag + 1$

Tasks: $\{send(m,k)_{1,2} : m \in M, k \in \mathbb{N}\}$

States:

`buffer`, a FIFO queue of elements of M, initially empty

`tag` $\in \mathbb{N}$, initially 0

Transitions:

$\mathbf{RECEIVE}_{1,2}\mathbf{(m)}$

Precondition:

m is first in buffer

Effect:

remove first element of buffer

$\mathbf{receive(m,k)}_{1,2}$

Effect:

if (k = tag +1) then

add m to buffer

tag := tag +1

$\mathbf{send(k)}_{2,1}$

Precondition:

k = tag

Effect:

none

Tasks:

$\{\text{send}(k)_{2,1} : k \in \mathbb{N}\}$

$\{\text{RECEIVE}(m)_{1,2}: m \in M\}$

$\mathbf{P}_1$	$\mathbf{P}_2$
$\mathbf{SEND(m)}_{1,2}$ Effect: add m to `buffer` $\mathbf{send(m,k)}_{1,2}$ Precondition: m is first in buffer k = tag Effect: none $\mathbf{receive(k)}_{2,1}$ Effect: if (k = tag) then remove first element (if any) of buffer tag := tag + 1	$\mathbf{RECEIVE}_{1,2}\mathbf{(m)}$ Precondition: m is first in buffer Effect: remove first element of buffer $\mathbf{receive(m,k)}_{1,2}$ Effect: if (k = tag +1) then add m to buffer tag := tag + 1 $\mathbf{send(k)}_{2,1}$ Precondition: k = tag Effect: none

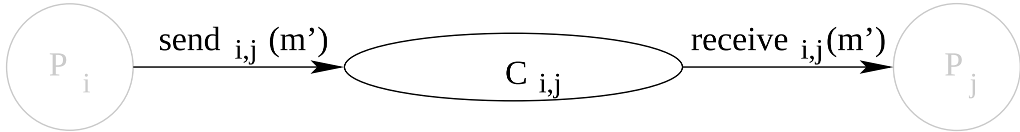
- Das Stenning Protokoll implementiert einen FIFO unter der Bedingung der WLL
- Duplizierung und Umsortieren von Nachrichten erlaubt
- Sehr kompakter Algorithmus

Wesentlicher Nachteil:

- Die genutzten Tags sind aus $\mathbb{N}$ und können beliebig lang werden

Alternating Bit Protocol (ABP)

- basiert auf dem Stenning Protokoll
- Tags fester Länge (ein Bit)
- Höhere Anforderungen an die zu Grunde liegenden Kanäle:
  kein Umsortieren mehr!
  $\leadsto$ Kanäle als *lossy FIFO channel* (siehe Kapitel 14, S. 462)



Signature:

Input:	Output:
send(m')$_{i,j}$	**receive(m')**$_{i,j}$

States:

queue, a FIFO queue of elements of M', initially empty

Transitions:

send(m')$_{i,j}$

Effect:

add any finite number of copies of *m'* to *queue*

receive(m')$_{i,j}$

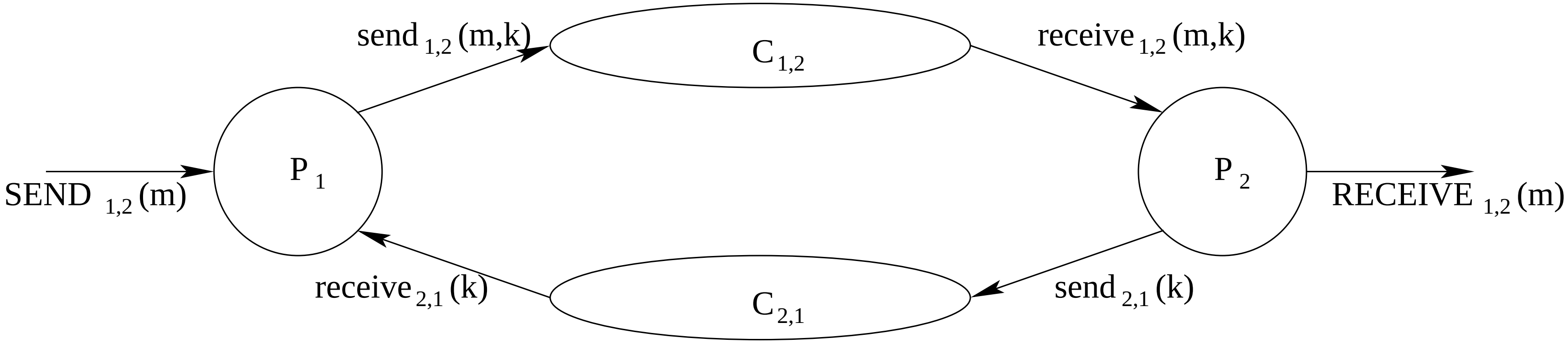
Precondition:

m' is first on *queue*

Effect:

remove first element of *queue*

Tasks: {**receive(m')**$_{i,j}$: m' $\in$M'}



$$k \in \{0,1\},\ m \in M$$

Signature $\mathbf{P}_1$:

Input:	Output:
$SEND(m)_{1,2}$, $m \in M$	$send(m,k)_{1,2}$, $m \in M$, $k \in \{0,1\}$
$receive(k)_{2,1}$, $k \in \{0,1\}$	

States P_1:

`buffer`, a FIFO queue of elements of M, initially empty

`tag` $\in \{0,1\}$, initially 1

Transitions P_1:

SEND(m)$_{1,2}$

Effect:

add m to `buffer`

send(m,k)$_{1,2}$

Precondition:

m is first on buffer

$k = tag$

Effect:

none

receive(k)$_{2,1}$

Effect:

if $(k = tag)$ then

remove first element (if any) of buffer

$tag := tag + 1 \bmod 2$

Tasks P_1: $\{send(m,k)_{1,2} : m \in M, k \in \{0,1\}\}$

Signature P_2:

Input:	Output:
$\text{receive}(m,k)_{1,2}$, $m \in M$, $k \in \{0,1\}$	$\text{RECEIVE}(m)_{1,2}$, $m \in M$
	$\text{send}(k)_{2,1}$, $k \in \{0,1\}$

States P_2:

`buffer`, a FIFO queue of elements of M, initially empty
`tag` $\in \{0,1\}$, initially 0

Transitions $\mathbf{P_2}$:

RECEIVE$_{1,2}$

Precondition:

m is first on buffer

Effect:

remove first element of buffer

receive(m,k)$_{1,2}$

Effect:

if ($k \neq tag$) then

add m to buffer

$tag := tag + 1 \bmod 2$

send(k)$_{2,1}$

Precondition:

$k = tag$

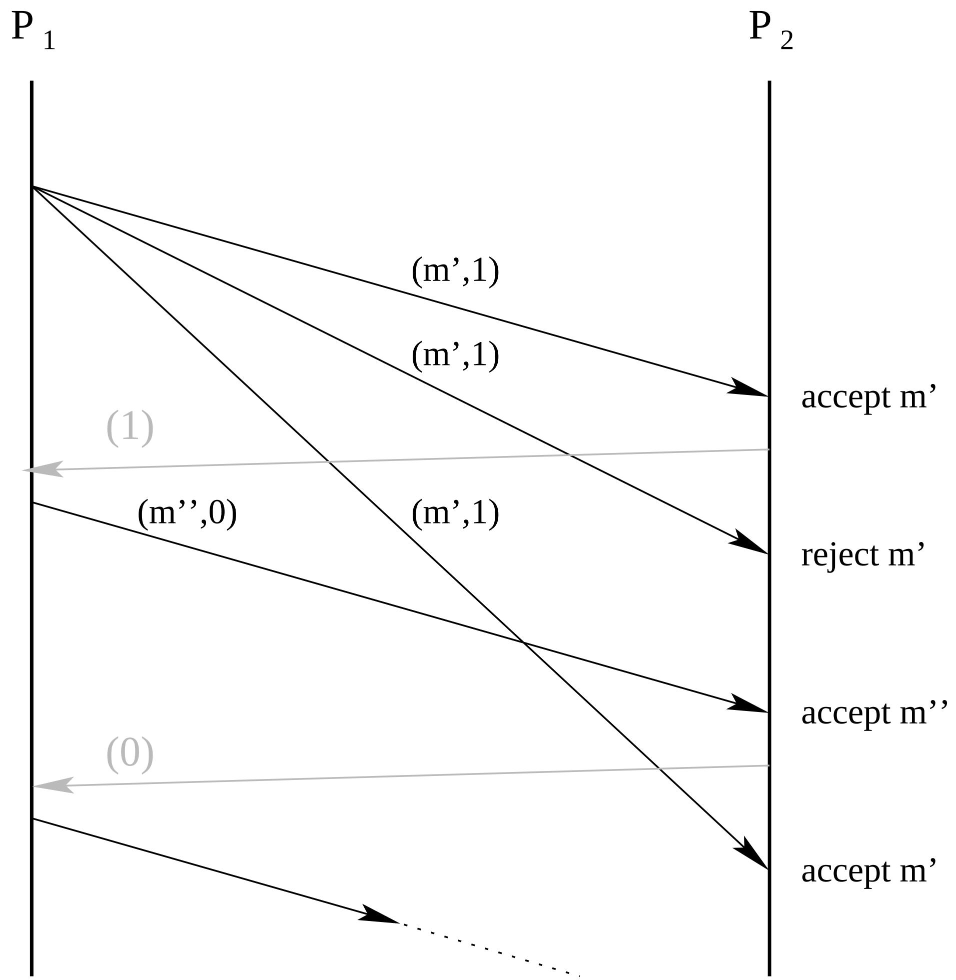
Effect:

none

Tasks $\mathbf{P_2}$:

$\{send(k)_{2,1} : k \in \{0,1\}\}$

$\{RECEIVE(m)_{1,2}: m \in M\}$



Prozeß P_2 akzeptiert die Nachricht m' zweimal, dies widerspricht aber der Eigenschaft eines sicheren FIFO Kanals!

Gibt es ein Protokoll mit *tags* aus eine endlichen Menge (*bounded tags*), welches...

- Umsortierung **und**
- Vervielfältigung

...von Nachrichten erlaubt?

$\rightsquigarrow$ NEIN!

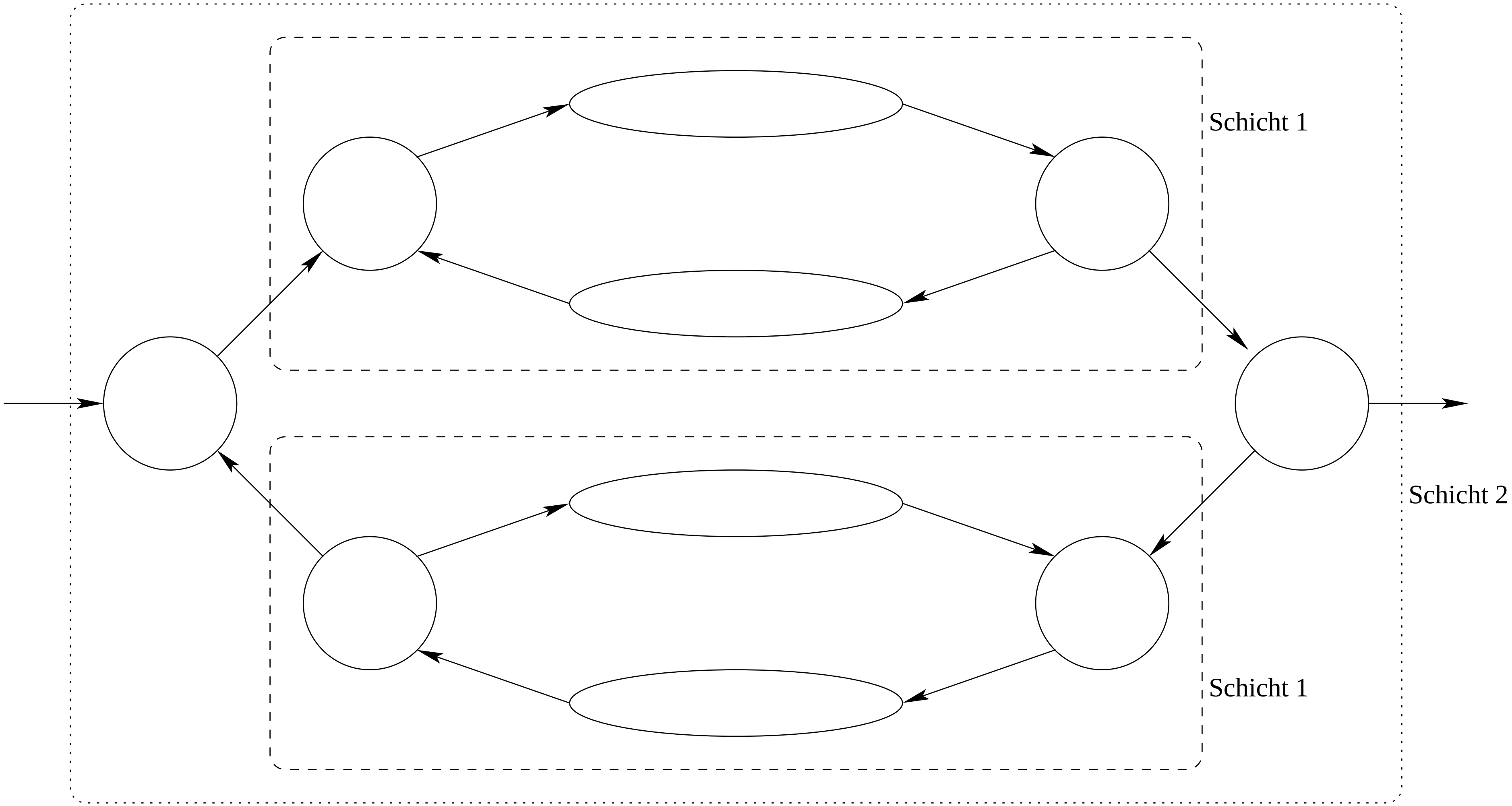
Begrifflichkeiten:

- Seien $Q_{1,2}$und $Q_{2,1}$*trace properties* für die Kanäle mit den Eigenschaften: kein Verlust aber Umsortierung und (endliche) Vervielfältigung von Nachrichten.
- Eine Ausführung α von $P_1 \times P_2$ sei konsistent mit $Q_{1,2}$wenn gilt: $\alpha|ext(Q_{1,2} \in traces(Q_{1,2})$. (analog für $Q_{2,1}$)
- Vereinfachung für *bounded tag*: man lege M und M' als endlich fest

Theorem 22.8: Es gibt kein *bounded tag* Protokoll (P_1,P_2), welches die FIFO Funktion mit Hilfe der durch $Q_{1,2}$und $Q_{2,1}$beschriebenen Kanäle implementiert (in dem Sinne, daß wenn α eine faire, konsistente Ausführung von $P_1 \times P_2$ ist, gilt: $\alpha|ext(FIFO) \in fairtraces(FIFO)$.)

Das Probe Protokoll

- erlaubt Verlust und Umsortierung aber keine Vervielfältigung von Nachrichten
- rein theoretisches Protokoll


Schicht 1
Schicht 2
Schicht 1

Signature:

Input:	Output:
SEND(m)$_{1,2}$, m $\in$M	**send(m)**$_{1,2}$, m $\in$M
receive("probe")$_{2,1}$	

States:

latest $\in$M $\cup$ $\{null\}$, initially *null*
unanswered $\in\mathbb{N}$, initially 0

Transitions:

SEND(m)$_{1,2}$
Effect:
latest := m

receive("probe")$_{2,1}$
Effect:
unanswered := unanswered + 1

send(m)$_{1,2}$
Precondition:
unanswered > 0
m = latest
Effect:
unanswered := unanswered -1

Tasks: $\{$**send(m)**$_{1,2}$: m $\in$M$\}$

Signature:

Input:	Output:
receive(m)$_{1,2}$**, m** $\in$**M**	**RECEIVE(m)**$_{1,2}$, m $\in$M
	send("probe")$_{2,1}$

States:

pending $\in \mathbb{N}$, initially 0
old $\in \mathbb{N}$, initially 0
for every m $\in$M:
  count(m) $\in \mathbb{N}$, initially 0

Transitions:

RECEIVE(m)$_{1,2}$	**send("probe")**$_{2,1}$
Precondition:	Precondition:
$count(m) \; > old$	$true$
Effect:	Effect:
for all (m') $\in$M do	$pending := pending + 1$
$count(m') := 0$	
$old := pending$	**receive(m)**$_{1,2}$
	Effect:
	$count(m) := count(m) + 1$

Tasks: $\{$**RECEIVE(m)**$_{1,2}$: m $\in$M$\}$

$\{$**send("probe")**$_{2,1}\}$

Für Schicht 1 gilt:

- Nachrichten werden nicht mehr umsortiert
- WLL
- Nachrichten werden ggf. unendlich oft dupliziert

Komplexität:

- Für eine *high-level* Nachricht werden mindestens $old + 1$ *low-level* Nachrichten benötigt, old ist monoton steigend!
- Gehen k *low-level* Nachrichten verloren, bedeutet dies, daß für jede danach übertragene *high-level* Nachricht mindestens k+1 *low-level* Nachrichten benötigt werden!

$\leadsto$ Es werden mehr und mehr *low-level* Nachrichten zur Übertragung einer *high-level* Nachricht benötigt, desto länger das Protokoll läuft.

Jedes Protokoll, welches einen verläßlichen FIFO Kanal garantiert und dabei auf den beschriebenen Kanälen basiert, hat eine ähnliche Komplexität!

Definition: Sei α eine vollständige Ausführung, $k \in \mathbb{N}^+$ und $m \in M$. Dann ist α' eine **k-extension** von α für m, falls folgendes gilt:

1. In dem Teil von α' nach α sind genau die beiden externen Ereignisse $SEND(m)_{1,2}$ und $RECEIVE(m)_{1,2}$ enthalten.

2. Alle *low-level* Nachrichten, die P_2 in α' empfängt, wurden nach α gesendet.

3. Die Anzahl der $receive_{1,2}$ Ereignisse in α' nach α ist $\leq k$.

Ein Protokoll heißt **k-message-bounded**, falls für jede vollständige Ausführung α und jedes $m \in M$ eine k-extension von α für m existiert.

Ein Protokoll heißt **message-bounded**, falls es k-message-bounded ist, für ein $k \in \mathbb{N}^+$.

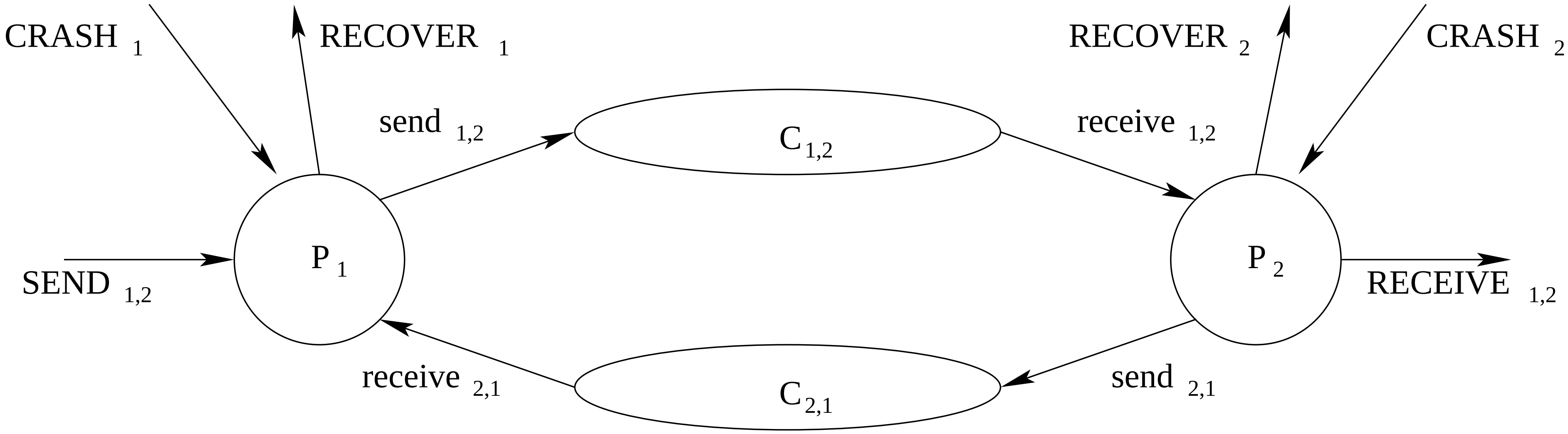
Theorem 22.11:
Es gibt kein message-bounded bounded-tag Protokoll (P_1,P_2), das die FIFO Funktion mit Kanälen realisiert, die Nachrichten verlieren und umsortieren.

Ausfälle von Prozessen

Fällt ein Prozeß aus, ohne wieder zu erwachen, ist keine weitere Kommunikation möglich.

Fällt ein Prozeß aus, um darauf wieder zu erwachen - **ohne** daß dabei Informationen bzgl. seines Zustandes verloren gehen, entspricht dies nur einem Pausieren.

Fällt ein Prozeß aus und erwacht darauf wieder, kann dieser aber auch mit einem definierten Start-Zustand fortfahren und verliert dabei sämtliche Information bzgl. seine Zustandes vor dem Ausfall.



Theorem 22.13: Es gibt kein Protokoll für das gegebene Ausfall-Modell, welches auf Basis von *reliable FIFO channels* die *FIFO* Funktionalität implementiert.

Gelockerte Eigenschaft B für die externe Schnittstelle:

- Keine Vervielfältigung, aber Umsortierung
- Verlust darf auftreten, falls das entsprechende SEND Ereignis einem RECOVER vorausgeht

Eigenschaften der Kanäle:

- Keine Vervielfältigung oder Umsortierung
- Verlust im Rahmen der SSL

($\rightsquigarrow$ *lossy FIFO channels*)

Es läßt sich zeigen, daß es auch für diese Bedingungen kein Protokoll gibt, welches einen zuverlässigen FIFO Kanal garantiert.

Die Lösung: Einsatz von *unique identifiers*, die “niemals” mehrfach UID genutzt werden - auch nicht nach Ausfällen!

Bemerkung zu UIDs:

In der Praxis sind die verfügbaren UIDs sehr groß aber **nicht** unendlich.

Deshalb ist eine Strategie zur erneuten Nutzung der UIDs notwendig, die sicherstellt, daß keine alten Packete mit gleicher UID mehr unterwegs sind.

$\leadsto$ Zeit zur Nachrichtenübermittlung, lokale Bearbeitungszeiten, Anzahl der *high-level* Nachrichten pro Zeiteinheit, explizite Richtlinien zum Verwerfen alter Packete

Die Folien finden sich unter:

http://www.koberstein.de/seminar