**S11.1** **Satz: Der Algorithmus von Burns und Neiger löst das Einigungsproblem. Spätestens nach** $\min(f + 2, m + 1)$ **Runden haben sich alle Prozessoren entschieden.**

**Beweis:**

**Terminierung: trivial**

**Richtigkeit: Unmittelbare Folgerung aus c)**

**Einigung: Da es** $m + 1$ **Gruppen gibt, aber höchstens m fehlerhafte Prozessoren, muß wenigstens eine Gruppe fehlerfrei sein. Damit folgt aus a) die Einigung spätestens in der (m+1)-ten Runde.**

**Ist** $f < m$**, so enthält** $G_1, \ldots, G_{f+2}$ **wenigstens zwei fehlerfreie Gruppen, woraus nach b) die Behauptung folgt.**







## 11.4 PAXOS

### 11.4.1 Aufgabenstellung

❏ **Modellannahmen**

- **Prozessoren**
  - **Mögliches Fehlverhalten: Fail Stop**
    **d. h. Prozessoren sind entweder *lebendig* oder *ausgefallen***
  - **Wiederanlauf erlaubt**
- **Kommunikationssystem**
  - **vollständig vermascht**
  - **zuverlässige FIFO-Verbindungen**

❏ **Aufgabenstellung**

- **Einigung**
  - **Wertevorrat für Anfangswerte beliebig**
  - **Einigung auf den Anfangswert eines der Prozessoren**

❑ **Synchronisation**

- **Die Ausführung von Zustandsübergängen erfolgt atomar**
- **L maximale Ausführungsdauer der (atomaren) Durchführung eines Zustandsübergangs**
- **D maximale Übertragungsdauer von Nachrichten**

❑ **Architektur**

| |
|---|
| Multi-Paxos (Folge von Einigungsvorgängen) |
| Basic Paxos (ein einzelner Einigungsvorgang) |
| Wahl eines Anführers |
| Ausfallerkennung |
| Nachrichtensystem |
| Hardware |

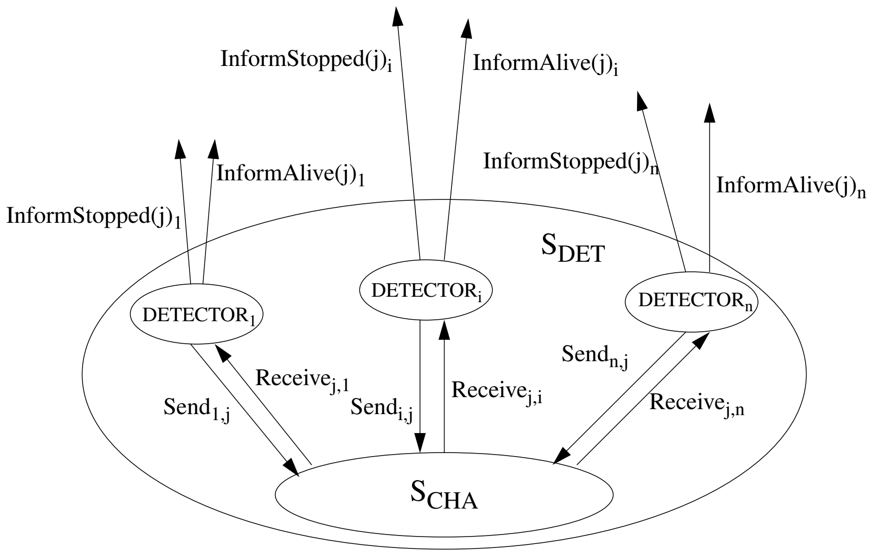






## 11.4.2 Ausfallerkennung

❑ **Systemstruktur beschrieben mit asynchron gekoppelten E/A-Automaten**



## Formale Beschreibung der Komponenten

**DETECTOR(z, c)$_i$**

**Signature:**

Input: Receive(m)$_{j,i}$, Stop$_i$, Recover$_i$

Internal: Check(j)$_i$

Output: InformStopped(j)$_i$, InformAlive(j)$_i$, Send(m)$_{i,j}$

**State:**

| | |
|---|---|
| Clock $\in$ R | initially arbitrary |
| Status $\in$ {alive, stopped} | initially alive |
| Alive $\in 2^I$ | initially I |
| for all j $\in$ I | |
| Prevrec(j) $\in R^{\geq 0}$ | initially arbitrary |
| Lastinform(j) $\in R^{\geq 0}$ | initially *Clock* |
| Lastsend(j) $\in R^{\geq 0}$ | initially *Clock* |
| Lastcheck(j) $\in R^{\geq 0}$ | initially *Clock* |





BP 1 Einigungsprotokolle: PAXOS

**Actions:**

input Stop$_i$

Eff: Status = stopped;

input Recover$_i$

Eff: Status = alive;

input Receive(“Alive”)$_{j,i}$

```
Eff: if (Status == alive) {
  Prevrec(j) = Clock;
  if (j ∉ Alive) {
    Alive = Alive ∪ {j};
    Lastcheck(j) = Clock + c;
  }
}
```


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

**internal Check(j)$_i$**

Pre: Status == alive && j $\in$ Alive

Eff: Lastcheck(j) = Clock + c;

if (Clock > Prevrec(j) + z + D) { Alive = Allive - {j}; }

**output Send("Alive")$_{i,j}$**

Pre: Status == alive

Eff: Lastsend(j) = Clock + z;

**output InformStopped(j)$_i$**

Pre: Status == alive && j $\notin$ Alive

Eff: Lastinform(j) = Clock + L;

**output InformAlive(j)$_i$**

Pre: Status == alive && j $\in$ Alive

Eff: Lastinform(j) = Clock + L;







**time_passage $\nu$(t)**

Pre: Status == alive

Eff: Let t' be such that {

$\forall$j, Clock + t' $\leq$ Lastinform(j);

$\forall$j, Clock + t' $\leq$ Lastsend(j);

$\forall$j, Clock + t' $\leq$ Lastcheck(j);

}

Clock = Clock + t'

## 11.4.3 Wahl eines Anführers

❏ **Anführer fungiert als Leiter von Einigungsrunden**

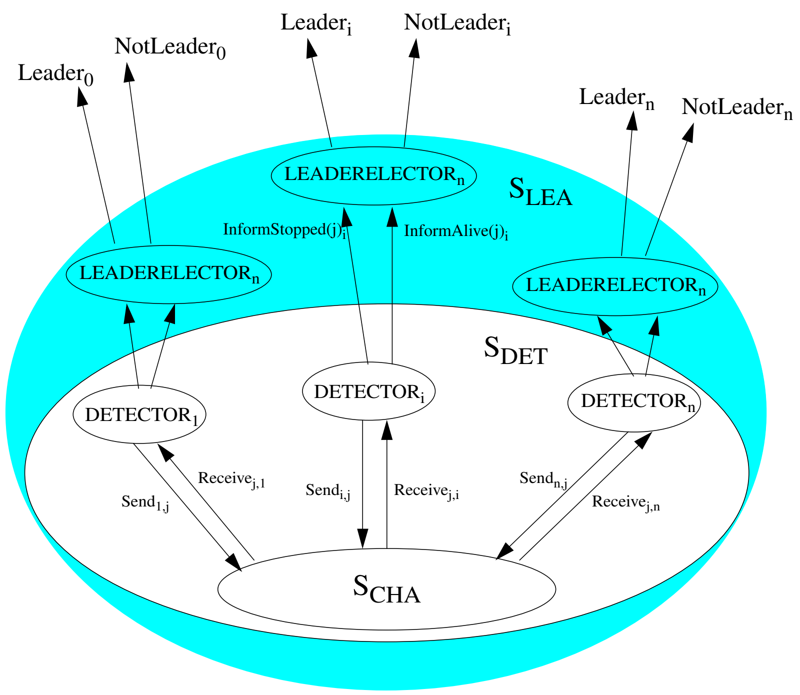
- **Zur Vereinfachung wird ein Verfahren ausgewählt, das eventuell zu mehreren Anführern führt**
- **Durch Zuteilung von logischen Zeitstempeln kann gewährleistet werden, daß sich schließlich einer durchsetzt (ähnlich Wahl bei allgemeinen Kommunikationsgraphen)**







❏ **Systemstruktur**



- **Formale Beschreibung**

**$LEADERELECTOR_i$**

**Signature:**

Input: $InformStopped(j)_i$, $InformAlive_i$, $Stop_i$, $Recover_i$

Output: $Leader_i$, $NotLeader_i$

**State:**

| | |
|---|---|
| Status $\in$ {alive, stopped} | initially alive |
| Pool $\in 2^I$ | initially {i} |
| Leader $\in$ I | initially i |







**Actions:**

**input $Stop_i$**

Eff: Status = stopped;

**input $Recover_i$**

Eff: Status = alive;

**input $InformStopped(j)_i$**

```
Eff: if (Status == alive) {
        Pool = Pool - {j};
        Leader = max(Pool);
     }
```

**input $InformAlive(j)_i$**

```
Eff: if (Status == alive) {
        Pool = Pool ∪ {j};
        Leader = max(Pool);
     }
```

**output Leader$_i$**

Pre: Status == alive && i = Leader

Eff: skip;

**output NotLeader$_i$**

Pre: Status == alive && i Leader

Eff: none;


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann





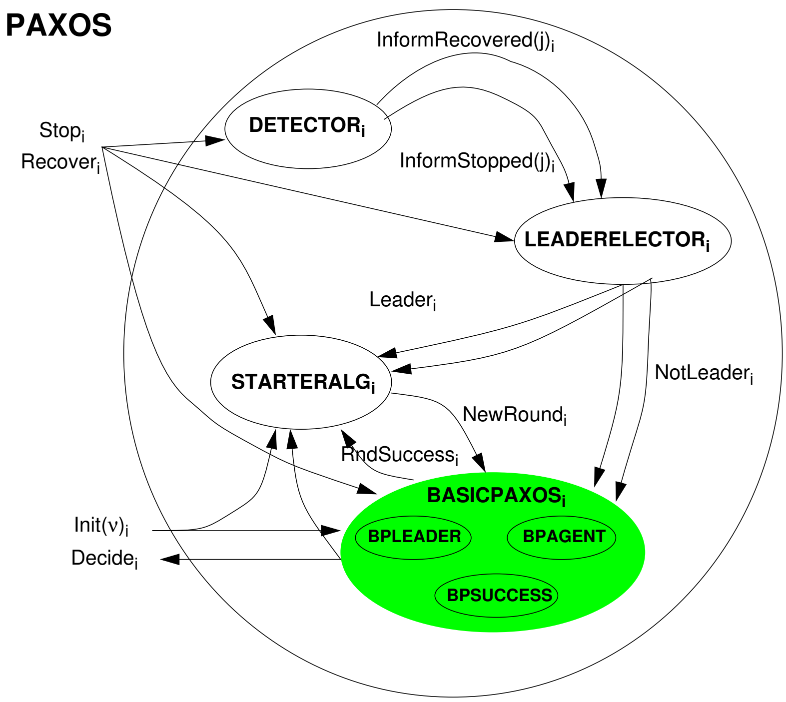
## 11.4.4 Paxos

- **Modularisierung**
  - **BASICPAXOS** versucht eine Entscheidungsrunde durchzuführen, an deren Ende sich alle Teilnehmer der Runde entschieden haben.
  - Runden können möglicherweise wegen Teilausfällen nicht beendet werden. Entscheidungsrunde wird deshalb von **STARTERALG** überwacht. Diese Komponente startet nach einer gewissen Zeit eine neue Runde mit dem gleichen Anführer.
  - **BPSUCCESS** verbreitet das Ergebnis einer erfolgreich beendeten Entscheidungsrunde.


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

## Einigungsprotokolle: PAXOS

❑ **Systemstruktur**



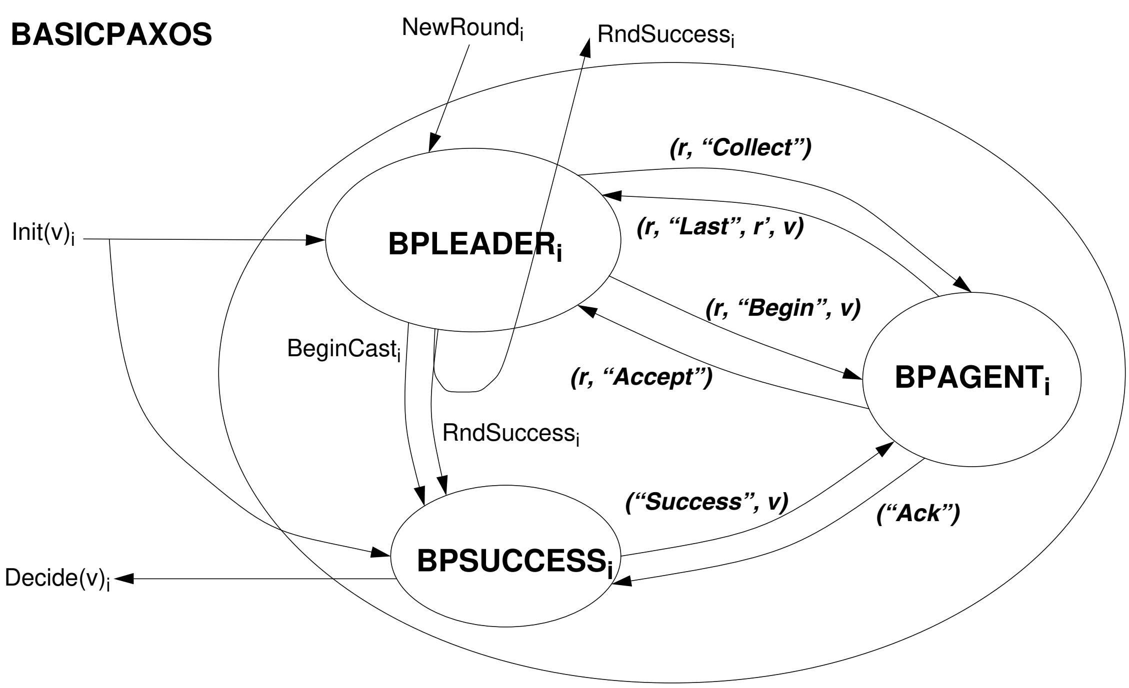


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann





## Einigungsprotokolle: PAXOS





Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

## 11.4.5 BASICPAXOS

❑ **Informelle Beschreibung**

- **"Collect"-Nachrichten (r, "Collect"):**
  **Werden vom Rundenanführer als Zeichen des Beginns der Runde r an alle versandt.**

- **"Last"-Nachrichten (r, "Last", r', v'):**
  **Werden bei Empfang einer "Collect"-Nachricht der Runde r an den Rundenanführer gesandt. Dabei ist r' die letzte Runde, in der der Sender einen Wert, nämlich v', als zustimmungsfähig akzeptiert hat. Existiert keine solche Runde wird r' = (0,i) gesetzt und als Wert der Anfangswert zurückgegeben bzw. nil.**

- **"Begin"-Nachricht (r, "Begin", v):**
  **Wird vom Rundenanführer als Vorschlag für den Einigungswert v versandt, der anhand der "Last"-Nachrichten ermittelt wird.**







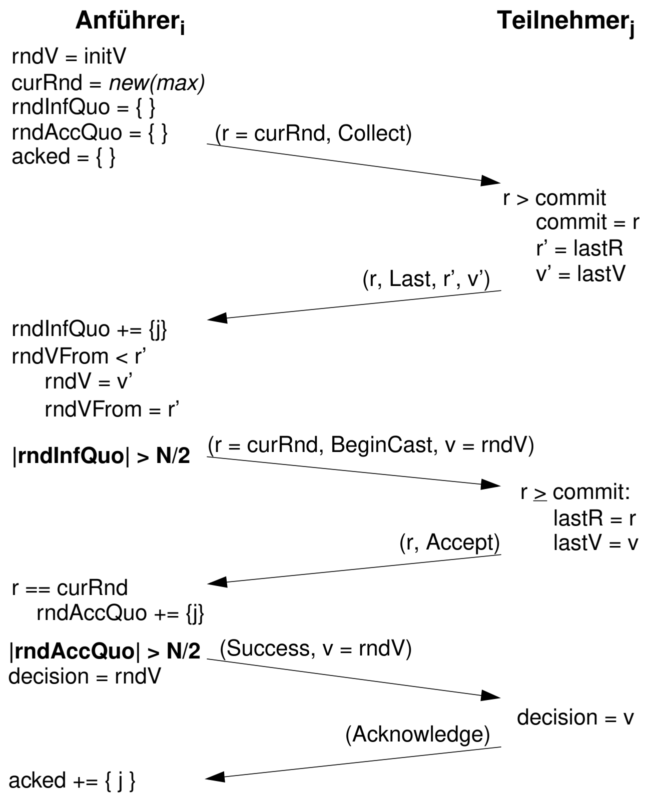
- **"Accept"-Nachricht (r, "Accept"):**
  **Wird als Antwort auf "Begin"-Nachricht zurückgesandt zum Zeichen, daß der vorgeschlagene Wert akzeptiert wird.**
- **"OldRound"-Nachricht (r, "OldRound", r'):**
  **Wird als Antwort auf eine "Collect"- oder "Begin"-Nachricht gesandt zum Zeichen, daß der Sender in der späteren Runde r' einen Einigungswert akzeptiert hat.**

- **"Success"-Nachricht ("Success", v):**
  **Wird vom Rundenanführer am Ende einer erfolgreichen Runde als Entscheidungswert gesandt.**

- **"Ack"-Nachricht ("Ack"):**
  **Wird an den Rundenanführer gesandt, als Zeichen des Empfangs einer "Success"-Nachricht, d. h. der Kenntnis des Entscheidungswertes.**

## Einigungsprotokolle: PAXOS

| | |
|---|---|
| rndV | value of this round |
| rndVFrom | round from which rndV is taken |
| curRnd | number of current round |
| rndInfQuo | set of processes for which "Last" has been received |
| rndAccQuo | set of processes for which "Accept" has been received |
| acked | set of processes for which "Acknowledge" has been received |
| commit | number of committed round |
| lastR | number of latest round |
| lastV | value for round lastR |
| decision | value for which processor decided |










## Einigungsprotokolle: PAXOS

- **Abstrakte Darstellung**

### Signature of BPLEADER

**Input:** Receive(m)$_{j,i}$, m ∈ {"Last", "Accept", "OldRound"}

**Internal:** Collect$_i$, GatherLast$_i$, GatherAccept$_i$, GatherOldRound$_i$

**Output:** Send(m)$_{i,j}$, m ∈ {"Collect", "Begin"}, BeginCast$_i$, RndSuccess(v)$_i$

### Signature of BPAGENT

**Input:** Receive(m)$_{j,i}$, m ∈ {"Collect", "Begin"}

**Internal:** LastAccept$_i$, Accept$_i$

**Output:** Send(m)$_{i,j}$, m ∈ {"Last", "Accept", "OldRound"}

### Signature of BPSUCCESS

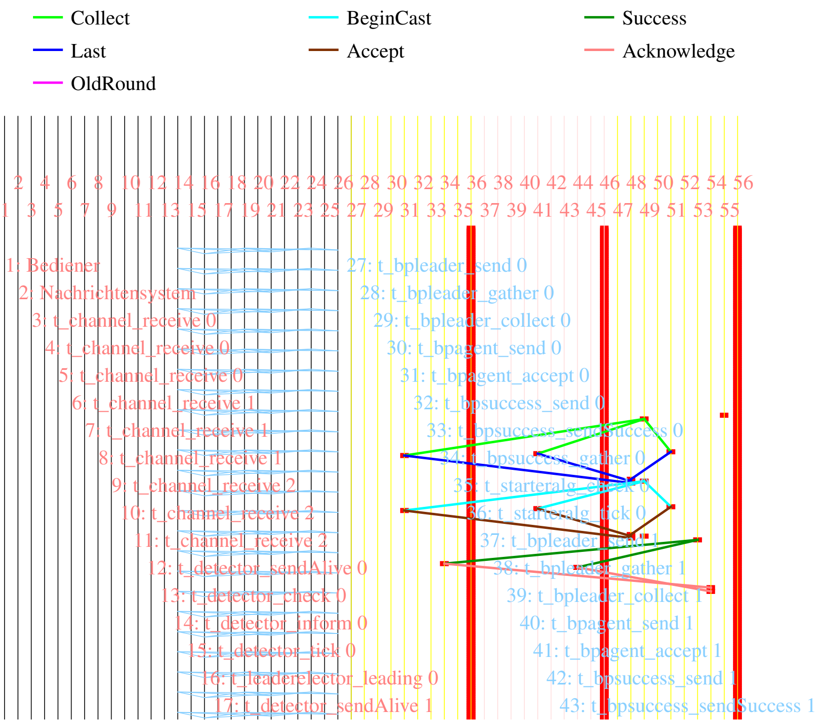
**Input:** Receive(m)$_{j,i}$, m ∈ {"Ack", "Success"}, RndSuccess(v)$_i$

**Internal:** SendSuccess, GatherSuccess$_i$, GatherAck$_i$

**Output:** Send(m)$_{i,j}$, m ∈ {"Success"}, Decide(v)$_i$

## BP 1 Einigungsprotokolle: PAXOS

### Ungestörter Ablauf



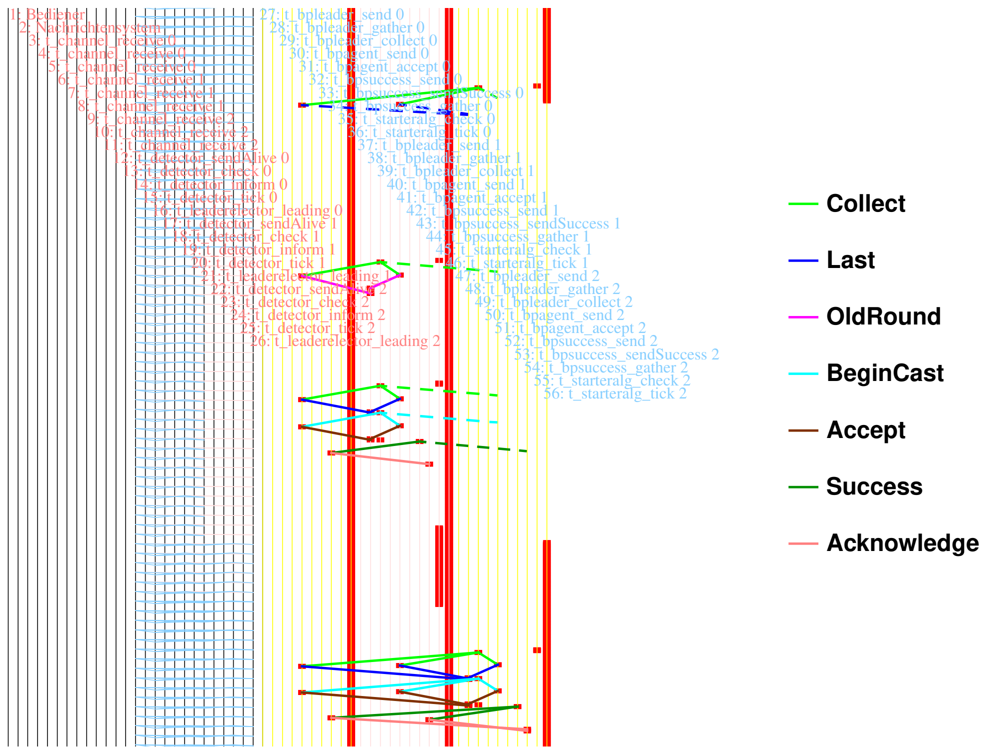


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann



## BP 1 Einigungsprotokolle: PAXOS

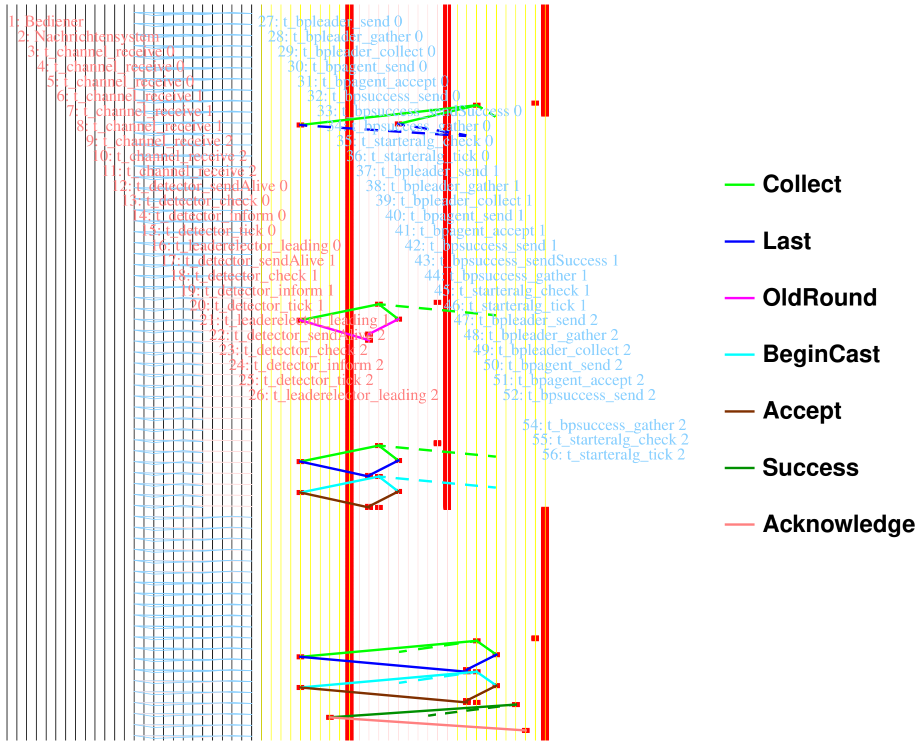
### Zeitweiser Ausfall von Prozessor 2





Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

## Zeitweiser Ausfall der Prozessoren 1 und 2





Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann





- **Gültigkeit**
  - **Anreicherung mit “Historien”-Variablen**
  - **Invariante 1: In jedem Zustand der Ausführung gilt für jede Runde r mit Hleader[r] = nil:**

| | | |
|---|---|---|
| Hvalue[r] | = | nil |
| Hfrom[r] | = | nil |
| HinfQuo[r] | = | { } |
| HaccQuo[r] | = | { } |

**D11.1 Eine Runde r heißt tot, wenn |Hreject[r]| ≥ n/2 ist.**

**Anführer$_i$** | **Teilnehmer$_j$**

rndV = initV
curRnd = *new(max)*
rndInfQuo = { }
rndAccQuo = { }
acked = { }
Hleader[curRnd] = i

(r = curRnd, Collect) →

r > commit
commit = r
r’ = lastR
v’ = lastV
lastR < r" < r
Hreject[r"] += { j }

← (r, Last, r’, v’)

rndInfQuo += {j}
rndVFrom < r’
rndV = v’
rndVFrom = r’

**|rndInfQuo| > N/2**
HinfQuo[curRnd] = rndInfQuo
Hfrom[curRnd] = rndVFrom
Hvalue[curRnd] = rndV

(r = curRnd, BeginCast, v = rndV) →

r ≥ commit:
lastR = r
lastV = v

← (r, Accept)

r == curRnd
rndAccQuo += {j}

**|rndAccQuo| > N/2**
decision = rndV
HaccQuo[curRnd] = rndAccQuo

(Success, v = rndV) →

decision = v

(Acknowledge)


Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

D4.2 R bezeichne den Wertevorrat für Rundennummern.

$R_S$ bezeichne die Menge $\{ r \in R \mid Hleader_i[r] \neq nil \}$.

$R_V$ bezeichne die Menge $\{ r \in R \mid Hvalue_i[r] \neq nil \}$.

- Invariante 2: In jedem Systemzustand ist $R_V \subseteq R_S$.

D4.3 $r \in R_V$ heißt verankert, wenn für jede Runde $r' \in R_V$ mit $r' < r$ $r'$ tot oder Hvalue[r'] = Hvalue[r] ist.

- Invariante 3: In jedem Zustand, in dem Prozessor j an Prozessor i die Nachricht (r, "Last", r", v) gesendet hat, besitzen alle r' mit r" < r' < r die Eigenschaft $j \in$ Hreject(r').

- Invariante 4: In jedem Zustand, in dem Prozessor $j \in rndInfQuo_i$ und $curRnd_i = r$ ist, gilt für alle Runden r' mit $rndVFrom_i < r' < r$ die Beziehung $j \in$ Hreject(r').







- Invariante 5: In jedem Zustand, in dem Prozessor $j \in$ HinfQuo(r) ist, gilt für alle Runden r' mit Hfrom(r) < r' < r die Beziehung $j \in$ Hreject(r').

- Invariante 6: In jedem Zustand ist jede lebendige (nicht tote) Runde verankert.

- Invariante 7: In jedem Zustand besitzen alle Variablen $decision_i \neq nil$ den gleichen Wert. Dieser Wert ist Anfangswert mindestens eines Prozessors.

❑ Terminierung

D4.4 Unter einem ***Ausführungsfragment*** wird ein Ausschnitt aus einer Ausführung des Algorithmus verstanden, der während eines (gewählten) Zeitintervalls stattfand.

Ein Ausführungsfragment heißt *stabil*, wenn während des Zeitintervalls, in dem es ausgeführt wurde, Prozessoren weder ausgefallen noch wiederangelaufen sind.

Ein Ausführungsfragment heißt *gut*, wenn es stabil ist und eine Mehrheit von Prozessoren während seiner Ausführung lebendig ist.

L11.17 In einer Runde, die in einem stabilen Ausführungsfragment enthalten ist, werden maximal 6n Nachrichten versandt.







L11.18 Ein Ausführungsfragment a, in dem noch kein Prozessor sich für einen Wert entschieden hat, gelte folgendes:

1. a ist stabil,
2. In a existiert genau ein Anführer i,
3. a dauert mindestens 4 * L + 2 * n * L + 2 * D Zeiteinheiten,
4. Prozeß i ist in a Anführer der (geeignet gewählten) Runde r und
5. Runde r ist erfolgreich.

Dann wird rndSuccess spätestens zum Zeitpunkt 7 * L + 4 * n *L + 4 * D ab Beginn des Ausführungsfragmentes ausgeführt.

L11.19 Wenn ein Ausführungsfragment stabil ist, wenigstens 3 * L + 2 * n * L + 2 * D Zeiteinheiten dauert und genau ein Anführer sich bereits vor Beginn des Fragmentes entschieden hat, dann weiß er, daß jeder lebendige Prozessor sich entschieden hat und es wurden höchstens 2n Nachrichten versandt.

L11.20 Ein Ausführungsfragment a besitze folgende Eigenschaften:

1. a ist gut,
2. in a gibt es genau einen Anführer i und
3. a dauert mindestens 16 * L + 8 * n * L + 9 * D Zeiteinheiten.

Dann hat der Anführer zum Zeitpunkt 16 * L + 8 * n * L + 9 * D eine Entscheidung getroffen.

L11.21 Es sei a ein gutes Ausführungsfragment, das mindestens 24 * L + 10 * n * L + 13 * D Zeiteinheiten dauert. Dann gilt:

1. Der Anführer entscheidet sich nach höchstens 21 * L + 8 * n * L + 11 * D Zeiteinheiten und es werden höchstens 8n Nachrichten versandt.
2. Spätestens nach 24 * L + 10 * n * L + 13 * D Zeiteinheiten ab Beginn des Ausführungsfragments entscheidet sich jeder lebendige Prozessor und es wurden maximal 2n weitere Nachrichten versandt.







## 11.4.6 MULTIPAXOS