4 Gegenseitiger Ausschluß

4.1 Erlaubnisbasierte Algorithmen

4.1.1 Der Algorithmus von Lamport

Ein Algorithmus basierend auf vollständiger Replikation des relevanten Systemzustandes

4.1.2 Der Algorithmus von Ricart und Agrawala

Reduktion der Nachrichtenzahl

4.1.3 Der Algorithmus von Maekawa

Maekawa, M.: A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems. ACM Transactions on Computer Systems, Vol. 3, No. 2, May 1985, pp. 145-159.

Verteilte Darstellung des Systemzustandes

4.1.4 Der Algorithmus von Sanders

Sanders, B. A.: The Information Structure of Distributed Mutual Exclusion Algorithms. ACM Transactions on Computer Systems, Vol. 5, No. 3, Aug. 1987, pp. 284-299.

Universeller erlaubnisbasierter Algorithmus

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

4.2 Token-basierte Algorithmen

4.2.1 Der Algorithmus von Suzuki und Kasami

4.2.2 Der Algorithmus von Singhal

Singhal, M.: A Heuristically-Aided Algorithm for Mutual Exclusion in Distributed Systems. IEEE Transactions on Computers, Vol. 38, No. 5, May 1989, pp. 651-662.

4.2.3 Der Algorithmus von Raymond

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

❑ **Systemmodell**

- **Prozesse (Knoten, 'sites') wickeln zeitlich sequentiell eine Folge von Aktionen ab**
- **Es gibt drei Arten von Aktionen:**
  - lokale, die keinerlei Auswirkungen auf andere Prozesse haben,
  - Aktionen zum Senden von Nachrichten,
  - Aktionen zum Empfangen von Nachrichten.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

❑ **Logische Uhren nach Lamport**

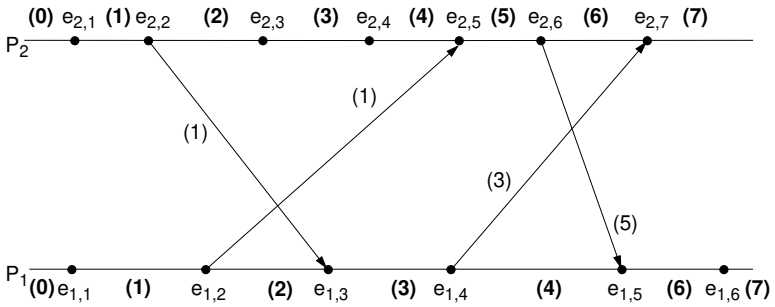
- **Jeder Prozeß verfügt über eine Uhr C_i , die in folgender Weise zählt:**
  - Bei Ausführung einer lokalen Aktion und bei Ausführung einer Sende-Aktion durch Prozeß **i**, wird seine Uhr C_i um 1 erhöht.
  - Jede Nachricht **m** trägt als Zeitstempel t_m den Stand der Uhr des Senders mit sich, den sie unmittelbar vor der Sendeaktion hatte.
  - Nach Ausführung einer Empfangsaktion durch Prozeß **i** bei Uhrenstand C_i und empfangener Nachricht **m** wird die lokale Uhr auf $\max(C_i, t_m) + 1$ gestellt.
- **Ergänzung zu vollständiger Ordnung**

  Zeitstempel bestehend aus Prozessornummer i und lokaler Zeit C_i.

  Anordnung: $(C_i, i) < (C_k, k)$ **genau dann, wenn** $C_i < C_k$ **oder** $C_i = C_k$ **und** $i < k$.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

• Beispiel



Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

❑ **Wesentliche Gesichtspunkte**

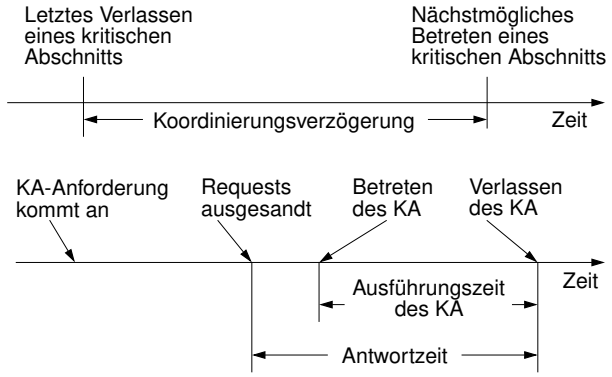
- **Gewährleistung des gegenseitigen Ausschlusses**
- **Verklemmungsfrei**
- **Aushungerungsfrei**

  D. h. kein Teilnehmer wird auf Dauer am Betreten des k. A. gehindert, während andere Teilnehmer wiederholt ihren k. A. ausführen.
- **Fair**

  D. h. die Zulassung erfolgt in der Reihenfolge der Anforderungen.
- **'livelock'-frei**
- **Fehlertolerant**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

• Erzielte Performanz



Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

4.1.1 Der Algorithmus von Lamport

- **Nachrichtenkanal: zuverlässig, FIFO**
- **Eintrittswunsch des Knotens S_i (site i)**

  REQUEST(ts, i) an alle
- **Verlassen eines kritischen Abschnitts**

  RELEASE(ts, i) an alle
- **Bearbeitung von REQUESTs:**
  - **Jeder Prozeß sammelt bei ihm angekommene REQUEST-Nachrichten und ordnet sie sortiert nach steigendem Zeitstempel in lokaler REQUEST-Warteschlange.**
  - **Wenn S_j REQUEST-Nachricht vonS_i erhält, sendet S_j eine REPLY-Nachricht an S_i.**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Betreten kritischer Abschnitte**

  S_i kann k. A. betreten, wenn folgende Bedingungen gemeinsam erfüllt sind:

  - S_i hat von allen anderen eine Nachricht mit Zeitstempel größer (ts, i) erhalten.
  - S_i's REQUEST ist am Kopf der Warteschlange.

- **Verlassen kritischer Abschnitte**

  - S_i entfernt bei Verlassen seines k. A. die eigene Anforderung aus der lokalen Warteschlange und sendet RELEASE-Nachricht an alle Knoten.
  - Wenn S_j eine RELEASE-Nachricht von S_i bekommt, entfernt es die zugehörige REQUEST-Nachricht aus seiner Warteschlange

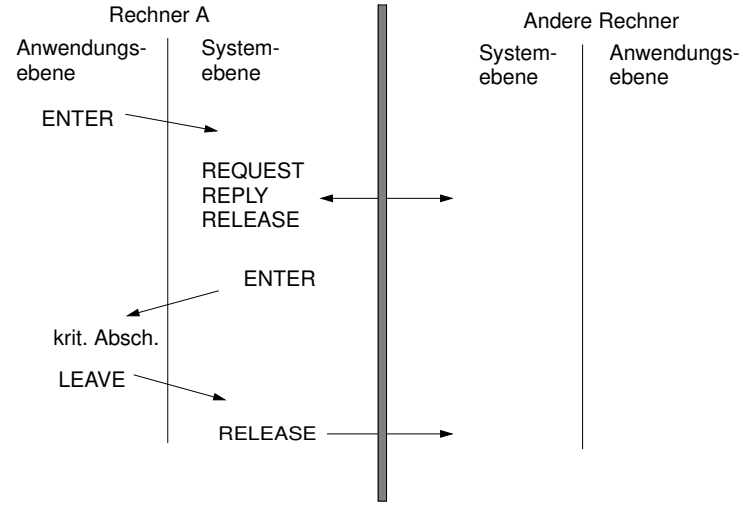
- **Verhalten**

  T mittlere Nachrichtenlaufzeit; E mittlere Ausführungszeit für k. A.

  - Antwortzeit: $2T + E$
  - Nachrichten: $3(N-1)$ pro k. A.
  - Koordinierungsverzögerung: T

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Implementationsarchitektur**



Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Die Ablaufumgebung

```
class BasicMessage {
  static final int DATALEN = 10;

  SimProcess source, destination;
  String type;
  double timestamp;
  int creationTime, deliveringTime; // For simulation purposes only

  public BasicMessage(SimProcess f_source,
                      SimProcess f_destination,
                      String f_type,
                      int latency) { ... }

  public void print(String string) { ... }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig


```
public abstract class SimProcess extends Thread {
  Scheduler scheduler;
  int uniqueID = 1;
  int nodeNumber;
  Vector outgoingConnections = new Vector();
  Vector incomingConnections = new Vector();
  Semaphore messageArrived;
  int receiveTurn;     // Number of next connection to be searched
                          for messages
  double time;         // Local time

  public SimProcess(String name, Scheduler scheduler,
                    int nodeNumber) {
   ...
    time = nodeNumber / 100.0;
  ...
 }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

```
  public void addConnection(SimProcess destination) { ... }
  // Add outgoing channel; at destination add as incoming channel

  Message receive() { ... }
  // Waiting for any message

  public void run () { ... }

  public abstract void runProcess();

  public double incrementTime() { return time++; }

  public double getTime() { return time; }

  public void setTime(double f_time) { time = f_time; }
}
```


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


```
public class BasicConnection {
  SimProcess source, destination;
  Semaphore empty, full, mutex;
  Semaphore messageArrived = null;
  Vector buffer = new Vector();   // Message buffer

  public BasicConnection(SimProcess f_source,
                         SimProcess f_destination) { ... }

  public BasicConnection(SimProcess f_source,
                         SimProcess f_destination,
                         Semaphore messageArrived) { ... }

  public void send(String type) { ... }

  public BasicMessage receive() { ... }

  public synchronized void xmit(BasicMessage message) { ... }

}
```


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

Der Algorithmus von Lamport

```
public class Message extends BasicMessage {
  double requestTimestamp;

  public Message(SimProcess source, SimProcess destination,
                 String type, double requestTimestamp, int latency) {
    super(source, destination, type, latency);
    this.requestTimestamp = requestTimestamp;
  }
}
```


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


```
public class Connection extends BasicConnection {

  public Connection(SimProcess source, SimProcess destination) {
    super(source, destination);
  }
  public Connection(SimProcess source, SimProcess destination,
                    Semaphore messageArrived) {
    super(source, destination, messageArrived);
  }
  public void send(String type) {
    Message message
      = new  Message(source, destination, type, 0, 2);
    xmit(message);
  }
  public void send(String type, double requestTimestamp) {
    Message message = new  Message(source, destination, type,
                                   requestTimestamp, 2);
    xmit(message);
  }
}
```


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

Gegenseitiger Ausschluß: Der Algorithmus von Lamport

```
class QueueEntry {
  String type;              // REQUEST, REPLY, OUTSIDE, INSIDE
  double requestTimestamp;
  double latestTimestamp;

  public QueueEntry() {
    type = "OUTSIDE";
    requestTimestamp = latestTimestamp = -1;
  }

  public QueueEntry(Message packet) {
    type = packet.type;
    requestTimestamp = packet.timestamp;
    latestTimestamp = packet.timestamp;
  }
}
```


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

Gegenseitiger Ausschluß: Der Algorithmus von Lamport

```
public class SystemThread extends SimProcess {
  QueueEntry[] queue = new QueueEntry[Lamport.NUMBEROFPROCESSES];

  public SystemThread(String name, Scheduler scheduler,
                      int nodeNumber) {
    super(name, scheduler, nodeNumber);
    for (int i = 0; i < Lamport.NUMBEROFPROCESSES; i++) {
      queue[i] = new QueueEntry();
    }
  }
```


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

Gegenseitiger Ausschluß: Der Algorithmus von Lamport

```
  public void runProcess() {
    int i;

    scheduler.sleep(2000);

    // Process S1 wants to enter critical section
    if (Thread.currentThread().getName().equals("S1")) {
      ((Connection)(outgoingConnections.elementAt(nodeNumber)))
        .send("ENTER");

    while (true) {
      Message message = receive();
      int sender = message.source.nodeNumber;
      queue[sender].latestTimestamp = message.timestamp;
```


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

Gegenseitiger Ausschluß: Der Algorithmus von Lamport

```
      if (message.type.equals("ENTER")) {
        for (i = 0; i < Lamport.NUMBEROFPROCESSES; i++) {
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("REQUEST", message.timestamp);
        }

      } else if (message.type.equals("LEAVE")) {
        for (i = 0; i < Lamport.NUMBEROFPROCESSES; i++) {
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("RELEASE");
        }
```


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

```
      } else if (message.type.equals("REQUEST")) {
        queue[sender] = new QueueEntry(message);
        for (i = 0; i < Lamport.NUMBEROFPROCESSES; i++) {
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("REPLY");
        }

      } else if (message.type.equals("REPLY")) {
        if (check()) {
        // Process enters critical section
          scheduler.sleep(10000);
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("LEAVE");
        }
```


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

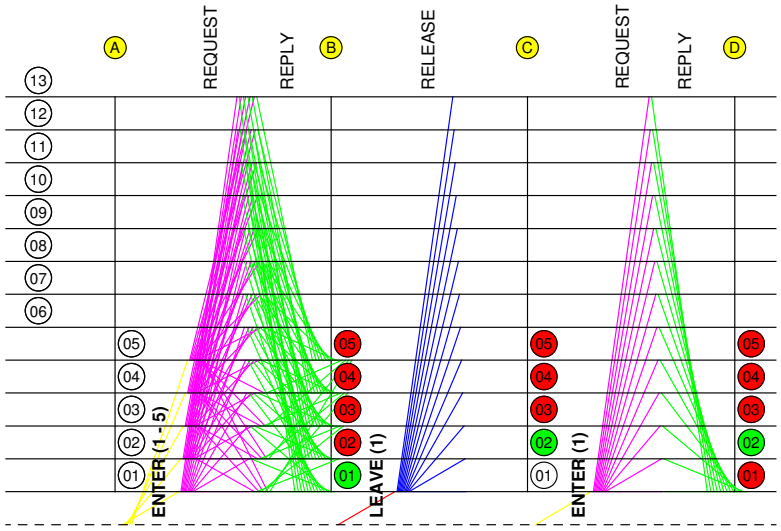

```
      } else if (message.type.equals("RELEASE")) {
        queue[sender].type = "OUTSIDE";
        if (check()) {
        // Process enters critical section
          scheduler.sleep(10000);
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("LEAVE");
        }

      } else {
        SystemPrinter.println("Error: unknown message type");
        System.exit(0);
      }
    }
  }
```


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


```
  boolean check() {
  // checking if I can enter
    if (queue[nodeNumber].type.equals("REQUEST")) {
      boolean allowed = true;
      for (int i = 0; i < Lamport.NUMBEROFPROCESSES; i++) {
        if (queue[i].latestTimestamp
              < queue[nodeNumber].requestTimestamp
            || (queue[i].type.equals("REQUEST")
                && queue[i].requestTimestamp
                     < queue[nodeNumber].requestTimestamp)) {
          allowed = false; break;
        }
      }
      if (allowed) { queue[nodeNumber].type = "INSIDE"; }
      return allowed;
    } else {
      return false;
    }
  }
}
```


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



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

4.1.2 **Der Algorithmus von Ricart und Agrawala**

- **Nachrichtenkanal: zuverlässig**
  (FIFO-Eigenschaft nicht notwendig!)
- **Eintrittswunsch des Knotens S_i (site i)**
  REQUEST(ts, i) an alle
- **Bearbeitung von REQUESTs:**
  - **Wenn S_j eine REQUEST-Nachricht von S_i erhält, sendet er eine REPLY-Nachricht an S_i, falls er**
    - weder selbst seinen kritischen Abschnitt betreten will noch ihn ausführt, oder
    - Knoten S_j seinerseits eine Anforderung gestellt hat und der Zeitstempel von S_i's Anforderung kleiner ist als derjenige der Anforderung von S_j.
  - **In allen anderen Fällen wird das Senden der REPLY-Nachricht aufgeschoben.**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Betreten kritischer Abschnitte**
  S_i kann k. A. betreten, wenn er auf seine Anforderung von allen anderen eine REPLY-Nachricht erhalten hat.
- **Verlassen des kritischen Abschnitts durch S_i**
  - **S_i versendet zu allen aufgeschobenen REQUESTs eine REPLY-Nachricht.**

❑ **Verhalten**

T mittlere Nachrichtenlaufzeit;
E mittlere Ausführungszeit für k. A.

- **Antwortzeit: $2T + E$**
- **Nachrichten: $2(N-1)$ pro k. A.**
- **Koordinierungsverzögerung: T**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Der Algorithmus von Ricart und Agrawala

```
public class Message extends BasicMessage {
  double requestTimestamp;

  public Message(SimProcess source, SimProcess destination,
                 String type, double requestTimestamp, int latency) {
    super(source, destination, type, latency);
    this.requestTimestamp = requestTimestamp;
  }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig


```
public class Connection extends BasicConnection {

  public Connection(SimProcess source, SimProcess destination) {
    super(source, destination);
  }
  public Connection(SimProcess source, SimProcess destination,
                    Semaphore messageArrived) {
    super(source, destination, messageArrived);
  }
  public void send(String type) {
    Message message
      = new  Message(source, destination, type, 0, 2);
    xmit(message);
  }
  public void send(String type, double requestTimestamp) {
    Message message = new  Message(source, destination, type,
                                   requestTimestamp, 2);
    xmit(message);
  }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Gegenseitiger Ausschluß: Algorithmus von Ricart und Agrawala

```
public class SystemThread extends SimProcess {
  String state;  // OUTSIDE, INSIDE, REQUEST
  double requestTimestamp;
  boolean[] deferredReply;
  int numberOfReplies;

  public SystemThread(String name, Scheduler scheduler,
    int nodeNumber) {
    super(name, scheduler, nodeNumber);
    deferredReply
      = new boolean[scheduler.system.getNumberOfProcesses()];
    requestTimestamp = -1;
    for (int i=0; i<scheduler.system.getNumberOfProcesses(); i++)
      deferredReply[i] = false;
    state = "OUTSIDE";
  }
```


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

Gegenseitiger Ausschluß: Algorithmus von Ricart und Agrawala

```
  public void runProcess() {
    int i;

    scheduler.sleep(2000);

    // Process S1 wants to enter critical section
    if (Thread.currentThread().getName().equals("S0")
        || Thread.currentThread().getName().equals("S1")) {
      ((Connection)(outgoingConnections.elementAt(nodeNumber)))
        .send("ENTER");
    }

    while (true) {
      Message message = receive();
      int sender = message.source.nodeNumber;
```


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

Gegenseitiger Ausschluß: Algorithmus von Ricart und Agrawala

```
      if (message.type.equals("ENTER")) {
        state = "REQUEST";
        requestTimestamp = message.timestamp;
        for (i=0; i<scheduler.system.getNumberOfProcesses(); i++){
          ((Connection)(outgoingConnections.elementAt(i)))
            .send("REQUEST", message.timestamp);
        }
        numberOfReplies = 0;

      } else if (message.type.equals("LEAVE")) {
        state = "OUTSIDE";
        for (i=0; i<scheduler.system.getNumberOfProcesses(); i++){
          if (deferredReply[i])
            ((Connection)(outgoingConnections.elementAt(i)))
              .send("REPLY");
        }
```


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

Gegenseitiger Ausschluß: Algorithmus von Ricart und Agrawala

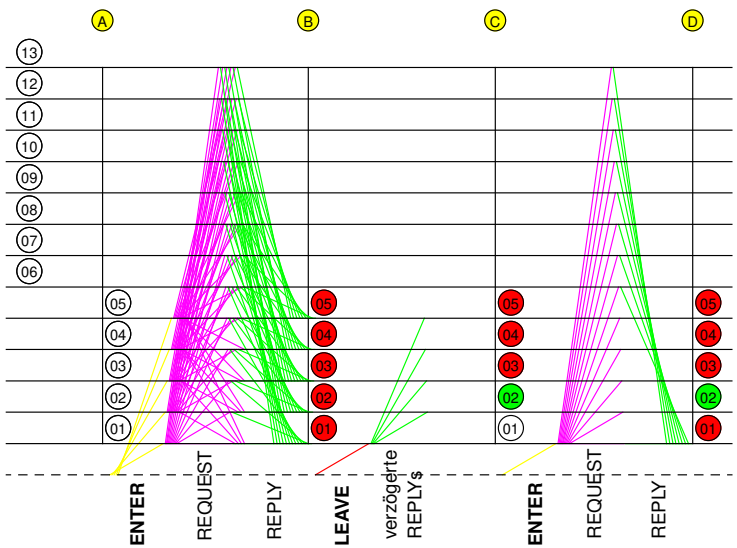
```
      } else if (message.type.equals("REQUEST")) {
        if (state.equals("OUTSIDE")
            || (state.equals("REQUEST")
                && message.requestTimestamp<=requestTimestamp)) {
          ((Connection)(outgoingConnections.elementAt(sender)))
            .send("REPLY");
        } else {
          deferredReply[sender] = true;
        }

      } else if (message.type.equals("REPLY")) {
        numberOfReplies++;
        check();

      } else {
        SystemPrinter.println("Error: Unknown message type");
        System.exit(0);
      }
    }
  }
```


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

```
void check() {
// checking if I can enter
  if (state.equals("REQUEST")) {
    if (numberOfReplies
        == scheduler.system.getNumberOfProcesses()) {
      state = "INSIDE";
      scheduler.sleep(10000);
      ((Connection)(outgoingConnections.elementAt(nodeNumber)))
        .send("LEAVE");
    }
  }
 }
}
```

4.1.3 Algorithmus Maekawa

- **Vorstellung: Die Verantwortung soll auf alle möglichst gleichmäßig verteilt werden, wobei jeder nur soviel Verantwortung wie unbedingt nötig tragen soll.**
- **Lösungsidee: Bildung von Quorum-Mengen, wobei jeder Beteiligte zum Eintritt in den kritischen Abschnitt die Zustimmung seines Quorums benötigt.**

◆ **Bedingungen an Quorum-Mengen**

Jedem Prozeß i wird ein Quorum S_i $(1 \le i \le n)$ zugeordnet derart, daß folgende Eigenschaften erfüllt sind:

1. $\forall i, j (1 \le i < j \le n \Rightarrow (S_i \cap S_j \neq \emptyset))$
2. $\forall i (1 \le i \le n \Rightarrow i \in S_i)$
3. $\forall i (1 \le i \le n \Rightarrow (|S_i| = \text{Konstante 1}))$
4. $\forall i (1 \le i \le n \Rightarrow (|\{j | (i \in S_j)\}| = \text{Konstante 2}))$

❑ **Satz (aus der endlichen projektiven Geometrie)**
Eine Quorum-Menge, die obige Bedingungen erfüllt, existiert, wenn sich n darstellen läßt in der Form $n = p^m(p^m + 1) + 1$,
wobei p eine Primzahl ist.

Beispiel: $n = 13 = 3^1(3^1 + 1) + 1$

In anderen Fällen müssen die Bedingungen (3) und (4) gelockert werden.

Gegenseitiger Ausschluß: Algorithmus von Maekawa

- Nachrichtenformat: (source, target, timestamp, message)
- Übergangsfunktion

1. Zugang anfordern durch Knoten i:

   send(i, s, T, REQUEST), an alle $s \in S_i$

2. Bei Empfang von Anforderung r = (i, s, T', REQUEST) durch s:

   frei: r in Anforderungsmenge aufnehmen
   belegt für r notieren;
   send(s, i, T, LOCKED);

   belegt für r': r in Anforderungsmenge aufnehmen;
   r nicht ältestes Element der Anf.-Menge:
   send(s, i, T, FAILED);
   r ältestes Element der Anf.-Menge:
   send(s, r'.source, T, INQUIRE),
   falls noch nicht geschehen;

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

Gegenseitiger Ausschluß: Algorithmus von Maekawa

3. Bei Empfang von (s, i, T', INQUIRE) durch i:

   Eine der letzten Anforderungen von i wurde von einem Quorums-Mitglied mit FAILED beantwortet:
   send(i, s, T, RELINQUISH);
   LOCKED(s) aus der Erlaubnismenge entfernen;
   sonst: ignorieren, aber vormerken!!!!;

4. Bei Empfang von RELINQUISH:

   Bestehende Belegung (*belegt für r*) aufheben;
   Für älteste Anforderung r' aus der Anforderungsmenge
   belegt für r' notieren;
   send(s, r'.source, T, LOCKED);

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

Gegenseitiger Ausschluß: Algorithmus von Maekawa

5. Bei Empfang von (s, i, T', LOCKED)

   LOCKED(s) in die Erlaubnismenge aufnehmen
   Von allen Mitgliedern des Quorums Erlaubnis (LOCKED) erhalten:
   Kritischen Abschnitt bearbeiten;

6. Beenden des kritischen Abschnitts

   Erlaubnismenge leeren
   send(i, s, T, RELEASE) für alle $s \in S_i$

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

Gegenseitiger Ausschluß: Algorithmus von Maekawa

7. Bei Empfang von RELEASE:

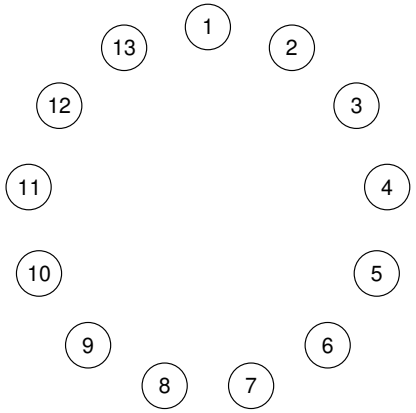
   belegt für r:
   r aus Anforderungsmenge entfernen
   Anforderungsmenge enthält neben r weitere Anforderungen:
   Für älteste Anforderung r' $\neq$ r aus der Anforderungsmenge
   belegt für r' notieren;
   send(s, r'.source, T, LOCKED);

8. Bei Empfang von (s, i, T', FAILED) durch i

   Eine der letzten Anforderungen von i wurde vom Quorums-Mitglied t mit INQUIRE beantwortet:
   send(i, t, T, RELINQUISH);
   LOCKED(t) aus der Erlaubnismenge entfernen;
   sonst: ignorieren, aber vormerken!!!!

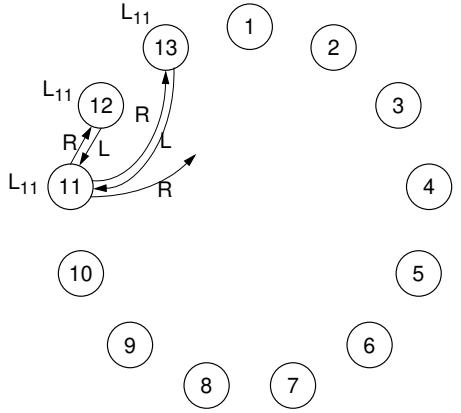
Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

(A)



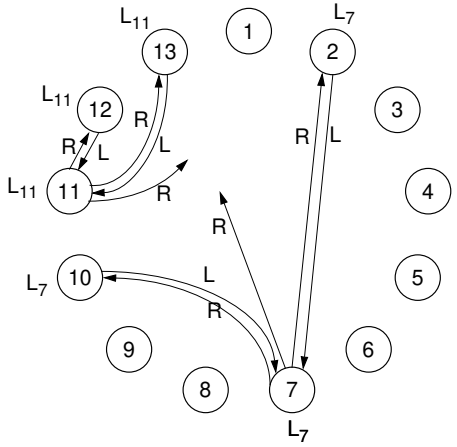
$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$

(B)



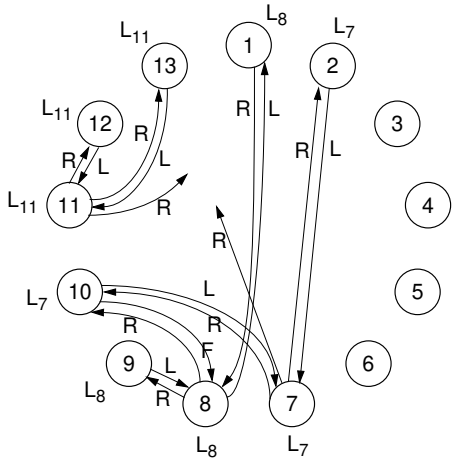
$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$

(C)

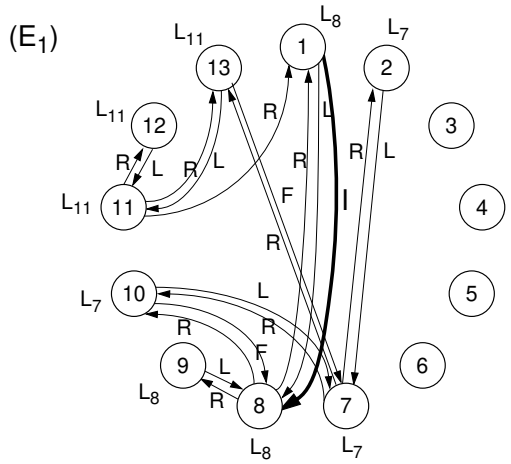


$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$

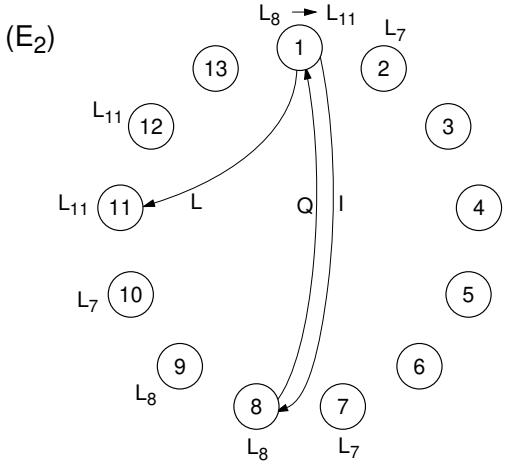
(D)



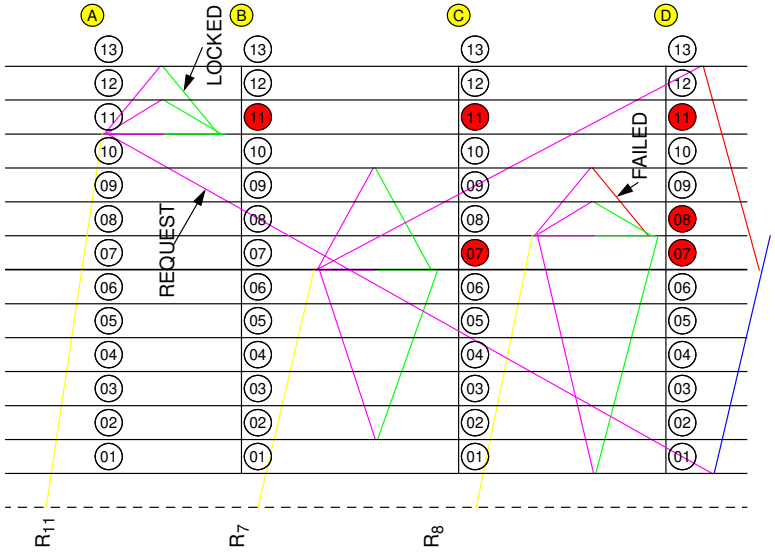
$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$

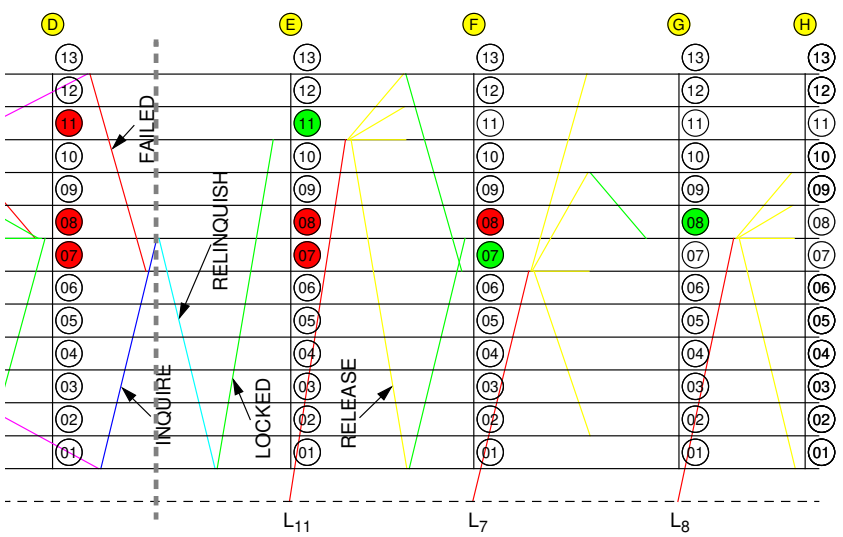


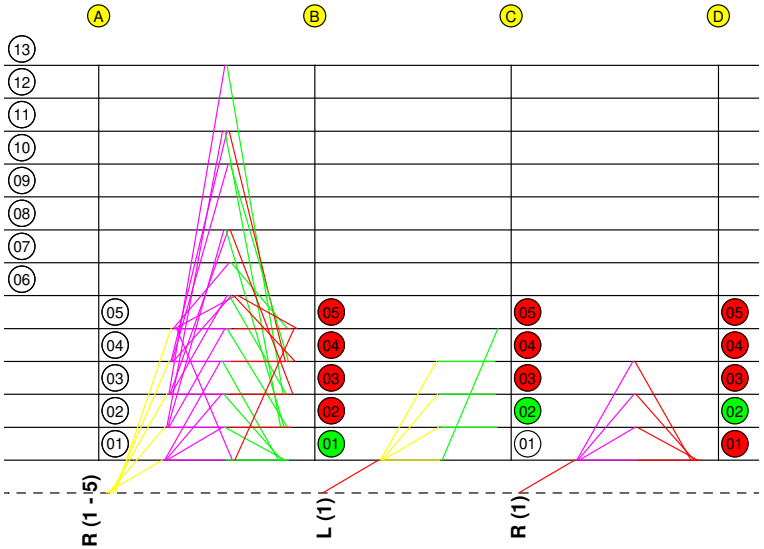
$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$



$S_1 = \{ 1, 2, 3, 4 \}$
$S_2 = \{ 2, 5, 8, 11 \}$
$S_3 = \{ 3, 6, 8, 13 \}$
$S_4 = \{ 4, 6, 10, 11 \}$
$S_5 = \{ 1, 5, 6, 7 \}$
$S_6 = \{ 2, 6, 9, 12 \}$
$S_7 = \{ 2, 7, 10, 13 \}$
$S_8 = \{ 1, 8, 9, 10 \}$
$S_9 = \{ 3, 7, 9, 11 \}$
$S_{10} = \{ 3, 5, 10, 12 \}$
$S_{11} = \{ 1, 11, 12, 13 \}$
$S_{12} = \{ 4, 7, 8, 12 \}$
$S_{13} = \{ 4, 5, 9, 13 \}$







4.1.4 **Verallgemeinerung: Algorithmus von Sanders**

R_i Menge der Prozesse, bei denen Prozeß i Erlaubnis einholen muß, wenn er seinen kritischen Abschnitt betreten will.

I_i Menge der Prozesse, die informiert werden müssen, wenn Prozeß i seinen kritischen Abschnitt verläßt.

$St_i = \{ j \mid i \in I_j \}$; Menge aller Prozesse, die Prozeß i bei Verlassen ihres kritischen Abschnitts informieren müssen.

S4.1 **Satz**

Wenn $\forall i(1 \le i \le N \Rightarrow i \in I_i)$ gilt, dann genügen die beiden folgenden Bedingungen zur Sicherstellung des gegenseitigen Ausschlusses:

1. $\forall i(1 \le i \le N \Rightarrow I_i \subseteq R_i)$
2. $\forall i, j(1 \le i, j \le N \Rightarrow (I_i \cap I_j \neq \emptyset) \vee (i \in R_j \wedge j \in R_i))$

- **Algorithmus**
  - **Zugang anfordern durch Prozeß i**

    send(i, j, T, REQUEST) an alle $j \in R_i$
    warten_auf_LOCKED von allen $j \in R_i$

  - **Verlassen des kritischen Abschnitts durch Prozeß i**

    send(i, j, -, RELEASE) an alle $j \in I_i$

  - **Empfang von r = (i, s, T, REQUEST)**

    r in Anforderungsmenge aufnehmen;
    status == frei:
    send(s, r'.source,-,LOCKED)
    für älteste bestehende Anforderung r'
    status = belegt(r'), falls r'.source $\in St_s$
    r' aus der Anforderungsmenge entfernen

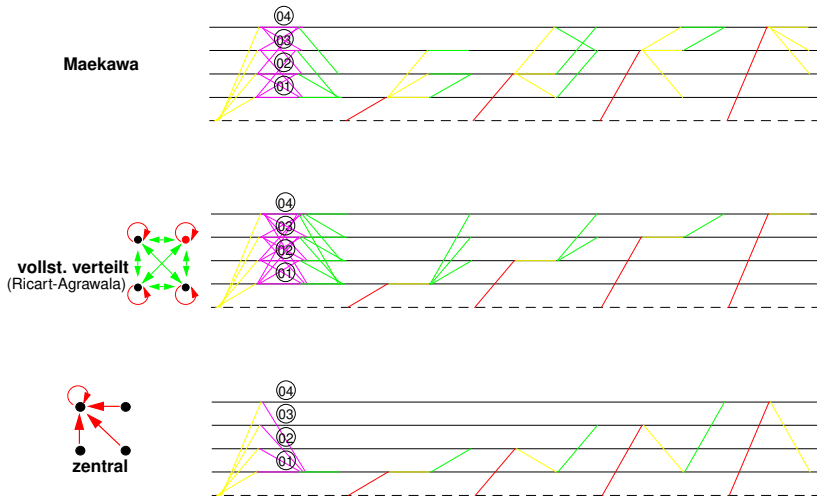
- **Empfang von r = (r, s, -, RELEASE)**

  status = frei
  solange status == frei und Anforderungsmenge != ∅
  send(s, r'.source,-,LOCKED)
  für älteste bestehende Anforderung r'
  status = belegt(r'), falls r'.source $\in St_s$
  r' aus der Anforderungsmenge entfernen

Nicht verklemmungsfrei! Lösung analog Maekawa

- **Graphische Darstellung**

  i → j $j \in I_i$
  i → j $j \in R_I, j \notin I_i$



❑ **Zahl der Nachrichten pro kritischem Abschnitt**

NM_i sei die Zahl der Nachrichten pro kritischem Abschnitt des Prozesses P_i. Dann ist

$$|I_i - \{i\}| + 2(|R_i - \{i\}|) \le NM_i$$

Der erste Summand ist die Zahl der Informations-nachrichten beim Verlassen eines k. A., der zweite die der Request- und Locked-Nachrichten.

Weiter ist

$$NM_i \le |I_i - \{i\}| + 2(|R_i - \{i\}|) + DM \quad \text{mit}$$

$$DM = \sum_{j \in R_i} r_j \text{ und } r_j = \begin{cases} 1 & \text{falls } St_j = \{j\} \text{ oder } = \{i, j\} \quad (\text{FAIL}) \\ 4 & \text{sonst} \quad (\text{INQUIRE, RELIN., LOCKED, FAIL}) \end{cases}$$

❑ **Spezialisierungen**

◆ **Lamport**

$$I_i = R_i = \text{alle}, \; |I_i| = |R_i| = N$$

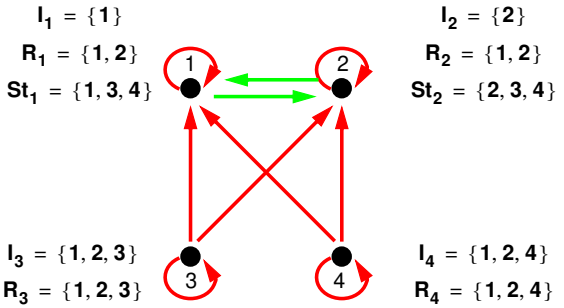
$$3(N-1) \le NM_i \le 3(N-1) + 4N$$

◆ **Ricart-Agrawala**

$$I_i = \{i\}, \; R_i = \text{alle}$$

$$2(N-1) \le NM_i \le 2(N-1) + N$$

❑ **Ein neuer Algorithmus**



x → y x muß y fragen, aber nicht informieren

x ⇒ y x muß y fragen und informieren

- **Zahl auszutauschender Nachrichten**

  $2 \le NM_i \le 2 + 8$ für $i \in \{1, 2\}$

  $6 \le NM_i < 6 + 9$ für $i \in \{3, 4\}$

- **Koordinierungsverzögerung**

$$D_{i,j} = \begin{cases} 2 & \text{falls } I_i \cap I_j \neq \emptyset \\ 0 & \text{falls } i = j \\ 1 & \text{sonst} \end{cases}$$

i \ j	1	2	3	4
1	0	1	2	2
2	1	0	2	2
3	2	2	0	2
4	2	2	2	0

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

4.2 Token-basierte Algorithmen

- **Prinzip**
  - Im System existiert genau ein Token
  - Ein Prozeß kann einen kritischen Abschnitt nur betreten, wenn er über das Token verfügt.
  - Ein Prozeß, der in seinen kritischen Abschnitt eintreten will, aber nicht über das Token verfügt, versendet REQUEST-Anforderungen.
  - Ein Prozeß, der REQUEST-Anforderungen erhält, übergibt das Token, sobald es frei ist, an einen Anforderer.
- **Probleme**
  - Erzeugung genau eines Tokens
  - Erkennung veralteter REQUEST-Anforderungen
  - Bestimmung des Prozesses, dem das Token übergeben werden soll.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

4.2.1 Algorithmus von Suzuki-Kasami für n kooperierende Prozesse P_i (0 <= i < n)

- **Datenstrukturen**
  - **Lokale Datenstruktur des Prozesses P_i**

    int RN[n] // Sequenznummern, soweit bekannt
  - **Datenstruktur des Tokens**

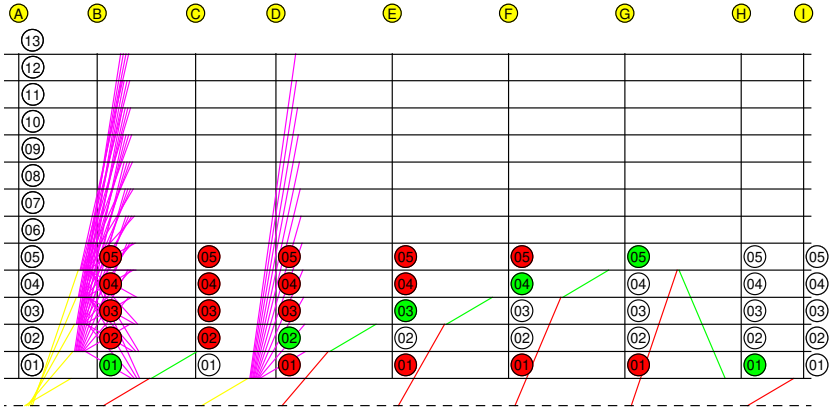
    struct {int LN[n]; int_fifo queue(n); }

    LN[i] enthält die Sequenznummer der zuletzt durch P_i ausgeführten REQUEST-Operation.
- **Anforderung des Tokens**
  - **Wenn Prozeß P_i das Token benötigt, erhöht er seine Sequenznummer RN[i] und sendet eine Nachricht REQUEST(i, RN[i]) an alle anderen.**
  - **Wenn ein Prozeß P_j eine Nachricht REQUEST(i, sn) empfängt, setzt er RN[i] = max(RN[i], sn). Falls P_j über das Token verfügt und nicht im kritischen Abschnitt ist, sendet er es an P_i, wenn RN[i] == LN[i] + 1 ist.**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Ausführung eines kritischen Abschnitts**
  - **Prozeß P_i führt den kritischen Abschnitt aus, sobald er über das Token verfügt.**
- **Verlassen eines kritischen Abschnitts**
  - LN[i] = RN[i]
  - **Jeder Prozeß P_j, der nicht in queue vermerkt ist und für den RN[j] == LN[j] + 1 ist, wird an queue angefügt.**
  - **Falls queue nicht leer ist, wird der erste Prozeß aus queue entfernt und das Token an ihn verschickt.**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig



4.2.2 Algorithmus von Singhal

Datenstrukturen

- **Lokale Datenstruktur des Prozesses P_i**

  Zustände:

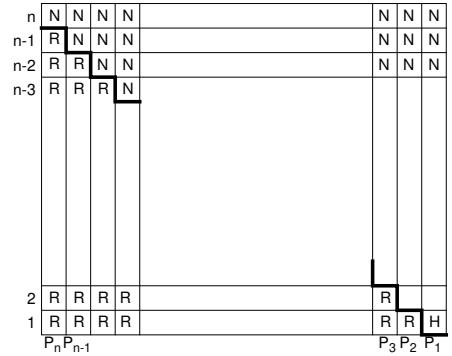
  - R Prozeß will kritischen Abschnitt betreten (requesting)
  - E Prozeß ist in kritischem Abschnitt (executing)
  - H Prozeß hält unbenutztes Token (holding free token)
  - N Prozeß ist unbeteiligt (none of the above)enum t_state {R, E, H, N};

```
/* 1 */  struct { t_state ST; // state
/* 2 */           int     SN; // sequence number
/* 3 */         } P[MAX_TEILNEHMERZAHL + 1];
```


Datenstruktur des Tokens

```
/* 4 */  struct {
/* 5 */    struct {
/* 6 */      t_state ST = N;
/* 7 */      int     SN = 0;
/* 8 */    } P[MAX_TEILNEHMERZAHL + 1];
/* 9 */  } Token;
```

Anfangsbesetzung der lokalen Zustände



Anforderung des Tokens durch Prozeß P_i

```
/*10*/  /*(1)*/  P[i].SN++;
/*11*/            if (P[i].ST != H) {
/*12*/              P[i].ST = R;
/*13*/              for (j = 1; j <= teilnehmerzahl; j++) {
/*14*/                if (P[j].ST == R && j != i) {
/*15*/                   send(j,         // Empfaenger
/*16*/                        REQUEST,   // Nachrichtentyp
/*17*/                        i,         // Sender
/*18*/                        P[i].SN)   // Daten
/*19*/                    );
/*20*/                }
/*21*/              }
/*22*/              wait_for_token();
/*23*/            }
```

Empfang einer REQUEST-Nachricht (i, REQUEST, j, sn)

i	**Adressat**
REQUEST	**Nachrichtentyp**
j	**Absender**
sn	**Sequenznummer**

```
/*24*/            if (P[j].SN < sn){
/*25*/              switch (P[i].ST) {
/*26*/  /*(8)*/       case N: P[j].ST = R;
/*27*/                        P[j].SN = sn;
/*28*/                        break;
/*29*/  /*(9)*/       case R: if (P[j].ST != R) {
/*30*/                          P[j].ST = R;
/*31*/                          send(j, REQUEST, i, P[i].SN);
/*32*/                        }
/*33*/                        P[j].SN = sn;
/*34*/                        break;
/*35*/  /*(10)*/      case E: P [j].ST = R;
/*36*/                        P[j].SN = sn;
/*37*/                        break;
/*38*/  /*(11)*/      case H: P[j].ST = R;
/*39*/                        P[j].SN = sn;
/*40*/                        T.P[j].ST = R;
/*41*/                        T.P[j].SN = sn;
/*42*/                        P[i].ST = N;
/*43*/                        send(j, TOKEN, i, token);
/*44*/                        break;
/*45*/              }
/*46*/            }
```


❑ Empfang des Tokens (i, TOKEN, j, token)

```
/*47*/  /*(2)*/   P[i].ST = E;
/*48*/             execute_critical_section();
```

❑ Verlassen eines kritischen Abschnitts

```
/*49*/  /*(3)*/   int tmp;
/*50*/             P[i].ST = N;            // wird fuer update benötigt
/*51*/             T.P[i].ST = N;
/*52*/             T.P[i].SN = P[i].SN;
/*53*/             update_ST_vectors();
/*54*/             P[i].ST = H;            // tatsaechlicher Zustand
/*55*/
/*56*/                                     // Arbitrierung nach round robin,
/*57*/                                     // koennte auch anders erfolgen.
/*58*/
/*59*/             for (tmp = 1; tmp <= teilnehmerzahl; tmp++) {
/*60*/               int j = ((i+tmp-1) % teilnehmerzahl) + 1;
/*61*/               if (P[j].ST == R) {
/*62*/                 P[i].ST = N;
/*63*/                 send(j, TOKEN, i, token);
/*64*/                 break;
/*65*/               }
/*66*/             }
```



```
/*67*/  update_ST_vectors() {
/*68*/    for (tmp = 1; tmp <= teilnehmerzahl; tmp++) {
/*69*/
/*70*/  /*(4)*/   if (P[tmp].ST == R && T.P[tmp].ST == N) {
/*71*/
/*72*/               if (P[tmp].SN == T.P[tmp].SN)
/*73*/               // Token hat die neuere Information
/*74*/                 P[tmp].ST = N;
/*75*/
/*76*/               else if (P[tmp].SN > T.P[tmp].SN) {
/*77*/               // Prozess hat die neuere Information
/*78*/                 T.P[tmp].ST = R;
/*79*/                 T.P[tmp].SN = P[tmp].SN;
/*80*/               } else /* if (P[tmp].SN < T.P[tmp].SN) */ {
/*81*/               // Token hat die neuere Information
/*82*/                 P[tmp].ST = N;
/*83*/                 P[tmp].SN = T.P[tmp].SN;
/*84*/               }
```



```
/*85*/             } else if (P[tmp].ST == N && T.P[tmp].ST == R) {
/*86*/  /*(5)*/       if (P[tmp].SN == T.P[tmp].SN)
/*87*/                   cout << "\n***** error 1\n" << flush;
/*88*/                 else if (P[tmp].SN > T.P[tmp].SN)
/*89*/                   cout << "\n***** error 2\n" << flush;
/*90*/                 else /* if (P[tmp].SN < T.P[tmp].SN) */{
/*91*/                 // Token hat die neuere Information
/*92*/                   P[tmp].ST = R;
/*93*/                   P[tmp].SN = T.P[tmp].SN;
/*94*/                 }
/*95*/             } else if (P[tmp].ST == N && T.P[tmp].ST == N){
/*96*/  /*(6)*/       if (P[tmp].SN == T.P[tmp].SN) ;
/*97*/                 // Token und Prozess haben gleiche Information
/*98*/                else if (P[tmp].SN > T.P[tmp].SN)
                          cout << "\n***** error 3\n";
/*99*/                 else /* if (P[tmp].SN < T.P[tmp].SN) */
/*100*/                // Token hat die neuere Information
/*101*/                 P[tmp].SN = T.P[tmp].SN;
```

```
/*102*/           } else /* if (P[tmp].ST == R && T.P[tmp].ST == R) */ {
/*103*/  /*(7)*/    if (P[tmp].SN == T.P[tmp].SN) ;
/*104*/              else if (P[tmp].SN > T.P[tmp].SN)
/*105*/              // Prozess hat die neuere Information
/*106*/               T.P[tmp].SN = P[tmp].SN;
/*107*/              else /* if (P[tmp].SN < T.P[tmp].SN) */
/*108*/              // Token hat die neuere Information
/*109*/                P[tmp].SN = T.P[tmp].SN;
/*110*/           }
/*111*/    }
/*112*/  }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Komprimierbar zu**

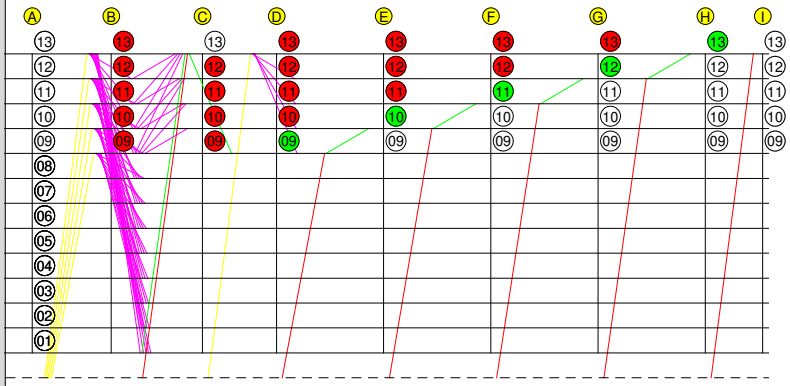
```
/*113*/  update_ST_vectors() {
/*114*/    for (tmp = 1; tmp <= teilnehmerzahl; tmp++) {
/*115*/      if (P[tmp].SN > T.P[tmp].SN) {
/*116*/      // Prozess hat neuere Information
/*117*/        T.P[tmp].ST = P[tmp].ST;
/*118*/        T.P[tmp].SN = P[tmp].SN;
/*119*/      } else {
/*120*/      // Token hat die neuere Information
/*121*/        P[tmp].ST = T.P[tmp].ST;
/*122*/        P[tmp].SN = T.P[tmp].SN;
/*123*/      }
/*124*/    }
/*125*/  }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Arbitrierung zur Erzielung von Fairneß**
  - **Auswahl nach kleinster Sequenznummer (zählt, wie oft der anfordernde Prozeß bislang den kritischen Abschnitt ausführte)**

    oder
  - **Auswahl der Anforderung des nächstgelegenen Prozesses**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig



Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Verifikation**
  - **Gegenseitiger Ausschluß**

Trivial, da ein Prozeß seinen kritischen Abschnitt nur dann betritt, wenn er über das Token verfügt und es erst nach Abschluß seines kritischen Abschnitts weitergibt.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

- **Verklemmungsfreiheit**

(Partielle) Verklemmung kann nur vorliegen, wenn die Prozesse in zwei Klassen zerfallen, so daß Prozesse einer Klasse keine Anforderungen an die der anderen richten und in beiden Anforderungen vorliegen.

Definitionen

P_i bezeichne (auch) den Zustandsvektor des i-ten Prozesses

$\mathbf{R_i^j}$ = **TRUE**, genau dann wenn $P_i[j].ST = R$

$\mathbf{N_i^j}$ = **TRUE**, genau dann wenn $P_i[j].ST = N$

$|\mathbf{P_i}|$ ist die Zahl der N-Einträge im Zustandsvektor des i-ten Prozesses

NP **ist die Menge der Prozesse, die seit Systembeginn noch keine Anforderung gestellt haben**

EP ist die Menge der Prozesse, die seit Systembeginn wenigstens einmal ihren kritischen Abschnitt durchlaufen haben

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

Im Anfangszustand gilt:

$\mathbf{R_i^j}$ ist erfüllt, für $\mathbf{1 \le i \le n}$ und $\mathbf{1 \le j < i}$

$\mathbf{N_i^j}$ ist erfüllt für $\mathbf{1 < i \le n}$ und $\mathbf{i \le j \le n}$

$\mathbf{N_1^j}$ ist erfüllt für $\mathbf{1 < j \le n}$

Eine Sequenz $P_n, P_{n-1}, ..., P_1$ ist "vollständig geordnet" wenn gilt:

$$|\mathbf{P_n}| < |\mathbf{P_{n-1}}| < \ldots < |\mathbf{P_1}|$$

L4.1.1 **Lemma**

Das Token hat niemals falsche R-Einträge.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

L4.1.2 **Lemma**

Wenn

- das System im Anfangszustand startet,
- alle Prozesse in **EP** ihren kritischen Abschnitt genau einmal durchlaufen haben und
- alle REQUEST-Nachrichten angekommen und bearbeitet sind,

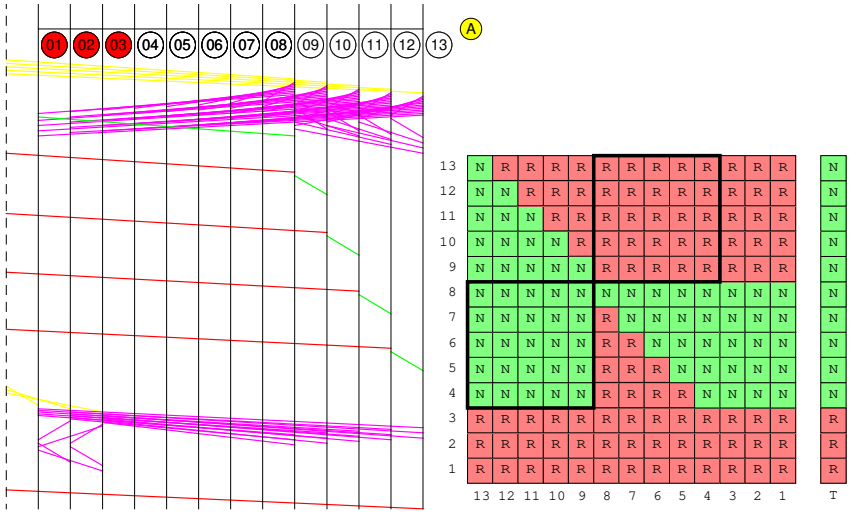
dann gilt:

$$\forall \mathbf{P_i}, \mathbf{P_j}(\mathbf{P_i} \in \mathbf{EP} \wedge \mathbf{P_j} \in \mathbf{NP} \Rightarrow \mathbf{R_j^i})$$

und

$$\forall \mathbf{P_i}, \mathbf{P_j}(\mathbf{P_i} \in \mathbf{EP} \wedge \mathbf{P_j} \in \mathbf{NP} \Rightarrow \mathbf{N_j^i})$$

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig



L4.1.3 Lemma

Wenn

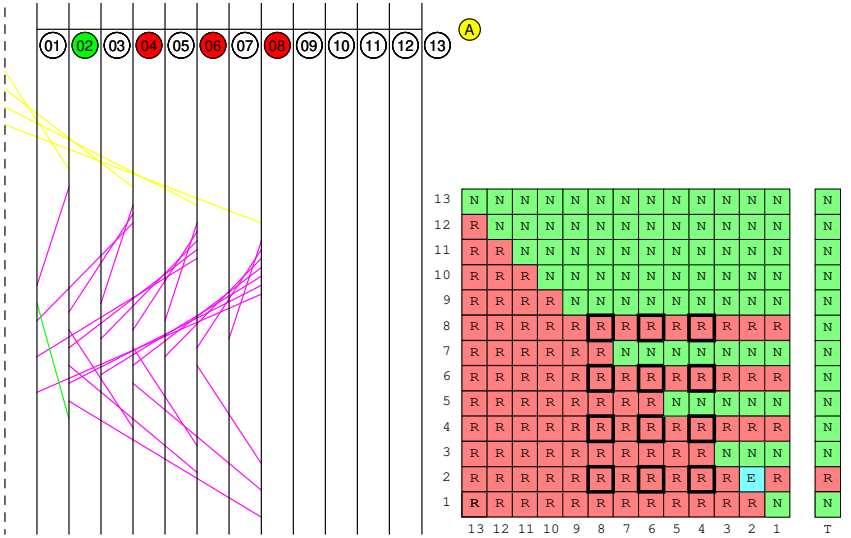
- das System im Anfangszustand startet,
- P_{i1}, P_{i2}, ..., P_{im} das Token anfordern,
- P_{ie} im kritischen Abschnitt ist und
- alle REQUEST-Nachrichten angekommen und bearbeitet sind,

dann gilt:

$$\forall \mathbf{i}, \mathbf{j}(\mathbf{i} \in \{\mathbf{i1}, \mathbf{i2}, \ldots, \mathbf{im}\} \wedge \mathbf{j} \in \{\mathbf{i1}, \mathbf{i2}, \ldots, \mathbf{im}, \mathbf{ie}\} \Rightarrow \mathbf{P_i}[\mathbf{j}] = \mathbf{R})$$

und

$$\forall \mathbf{j}((\mathbf{j} \in \{\mathbf{i1}, \mathbf{i2}, \ldots, \mathbf{im}\} \Rightarrow \mathbf{P_{ie}}[\mathbf{j}] = \mathbf{R}) \wedge (\mathbf{P_{ie}}[\mathbf{ie}] = \mathbf{E}))$$



L4.1.4 Lemma

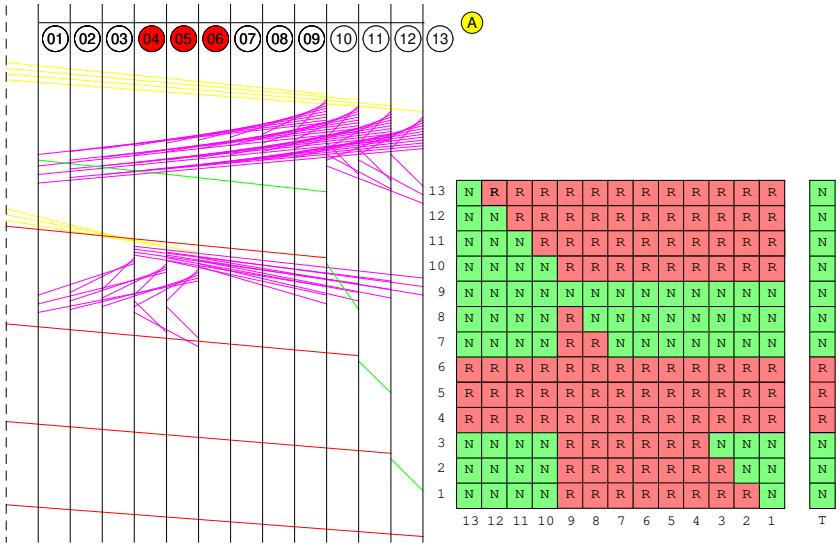
Wenn

- Prozesse P_{im}, P_{im-1}, ..., P_{i1} eben ihre kritischen Abschnitte in dieser Reihenfolge ausgeführt haben und
- alle von ihnen gesandten REQUEST-Nachrichten empfangen und verarbeitet wurden,

dann gilt:

$$\forall \mathbf{j}, \mathbf{k}(\mathbf{1} < \mathbf{j} \leq \mathbf{m} \wedge \mathbf{1} \leq \mathbf{k} < \mathbf{j} \Rightarrow \mathbf{R_{ij}^{ik}})$$

$$\forall \mathbf{j}, \mathbf{k}(\mathbf{1} \leq \mathbf{j} < \mathbf{m} \wedge \mathbf{j} < \mathbf{k} \leq \mathbf{m} \Rightarrow \mathbf{N_{ij}^{ik}})$$



L4.1.5 Lemma

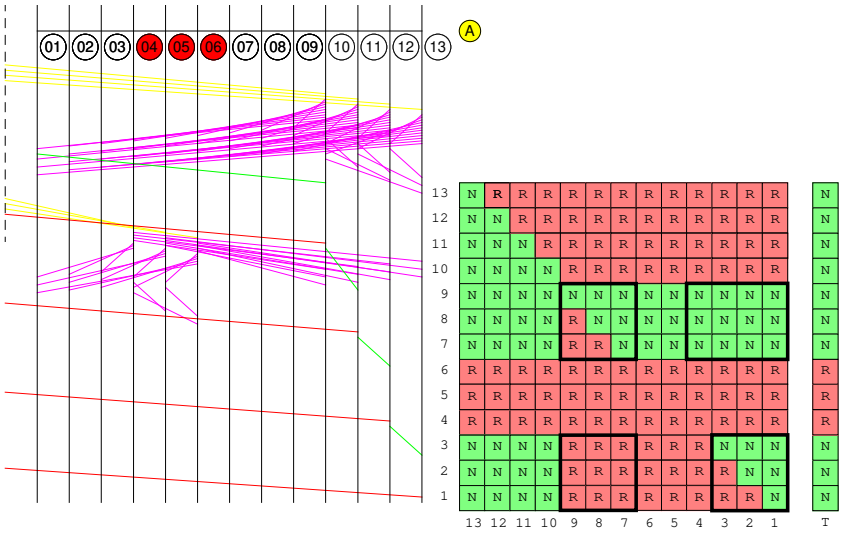
Wenn

- das System im Anfangszustand startet,
- alle Prozesse in NP = $\{P_{im}, P_{im-1}, ..., P_{i1}\}$ anfänglich $|\mathbf{P_{im}}| < |\mathbf{P_{im-1}}| < \ldots < |\mathbf{P_{i1}}|$ erfüllen und
- alle REQUEST-Nachrichten angekommen und bearbeitet sind,

dann gilt:

$\forall \mathbf{j}, \mathbf{k}(\mathbf{1} < \mathbf{j} \leq \mathbf{m} \wedge \mathbf{1} \leq \mathbf{k} < \mathbf{j} \Rightarrow \mathbf{R_{ij}^{ik}})$

$\forall \mathbf{j}, \mathbf{k}(\mathbf{1} < \mathbf{j} \leq \mathbf{m} \wedge \mathbf{j} < \mathbf{k} \leq \mathbf{m} \Rightarrow \mathbf{N_{ij}^{ik}})$



S4.2 Satz

Wenn das System im Anfangszustand startet und zu jedem Betrachtungszeitpunkt folgende Eigenschaften besitzt:

- Alle Prozesse $P_{im}, P_{im-1}, ..., P_{i1}$ haben ihre kritischen Abschnitte genau einmal und in dieser Reihenfolge ausgeführt.
- Alle REQUEST-Nachrichten sind angekommen und bearbeitet.
- Prozesse $P_{jr}, P_{jr-1}, ..., P_{j1}$ haben nie Anforderungen gestellt und $|\mathbf{P_{jr}}| < |\mathbf{P_{jr-1}}| < \ldots < |\mathbf{P_{j1}}|$.
- Kein Prozeß fordert das Token an.

Dann bilden $P_{jr}, P_{jr-1}, ..., P_{j1}, P_{im}, P_{im-1}, ..., P_{i1}$ eine vollständig geordnete Sequenz.

Beweis: Folgt unmittelbar aus Lemmata 2, 4 und 5.

S4.3 **Satz**

Wenn das System im Anfangszustand startet und zu jedem Zeitpunkt folgende Eigenschaften besitzt:

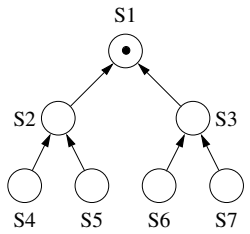
- Einige oder alle Prozesse haben ihren kritischen Abschnitt beliebige Male durchlaufen.
- Alle REQUEST-Nachrichten wurden empfangen und bearbeitet.
- Kein Prozeß hat eine Anforderung des Tokens ausstehen oder befindet sich im kritischen Abschnitt.

Dann bildet die Gesamtheit der Prozesse eine vollständig geordnete Sequenz.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

4.2.3 **Algorithmus von Raymond**

Die Prozesse sind als logischer Baum angeordnet



❑ **Datenstrukturen**

- Lokale Datenstruktur des Prozesses P_i

```
int holder;  // Nummer des Vorgängers!
             // Bei der Wurzel: eigene Nummer
int_fifo queue;  // Warteschlange von
                 // Anforderern
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

❑ **Anforderung des Tokens**

- **Wenn Prozeß P_i das Token benötigt und seine Warteschlange leer ist, sendet er eine REQUEST-Nachricht an seinen Vorgänger.**
  Er reiht sich in seine eigene Warteschlange ein.
- **Wenn Prozeß P_i von P_j eine REQUEST-Nachricht erhält, nimmt er P_j in seine Warteschlange auf. Hat er selbst keine ausstehen, so sendet er eine REQUEST-Nachricht an seinen Vorgänger.**
- **Wenn die Wurzel eine REQUEST-Nachricht erhält sendet sie das Token zu dem Anforderer und vermerkt ihn in der Variablen holder.**
- **Wenn ein Prozeß das Token erhält, tilgt der den ersten Eintrag seiner Warteschlange, sendet das Token dem entsprechenden Prozeß und vermerkt den Empfänger in seiner Variablen holder.**
  Wenn seine eigene Warteschlange dann noch nicht leer ist, sendet der Prozeß eine REQUEST-Nachricht an den in holder verzeichneten Prozeß.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

❑ **Ausführung eines kritischen Abschnitts**

- **Prozeß P_i führt den kritischen Abschnitt aus, sobald er über das Token verfügt und als erster in seiner Warteschlange steht. Sein Eintrag wird in diesem Fall aus der Warteschlange entfernt.**

❑ **Verlassen eines kritischen Abschnitts**

- **Falls die eigene Warteschlange nicht leer ist, wird das Token an den an erster Stelle stehenden Prozeß gesendet, der Empfänger in holder vermerkt und der Eintrag aus der Warteschlange entfernt.**
- **Wenn dann seine eigene Warteschlange noch nicht leer ist, wird eine REQUEST-Nachricht an den in holder vermerkten Prozeß gesandt.**

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig