Schedulingverfahren für zeitgesteuerte Anwendungen in verteilten embedded Systemen

Dirk Ewerlin
INF-M3 - Seminar/Ringvorlesung - Wintersemester 2007/2008
14. Dezember 2007

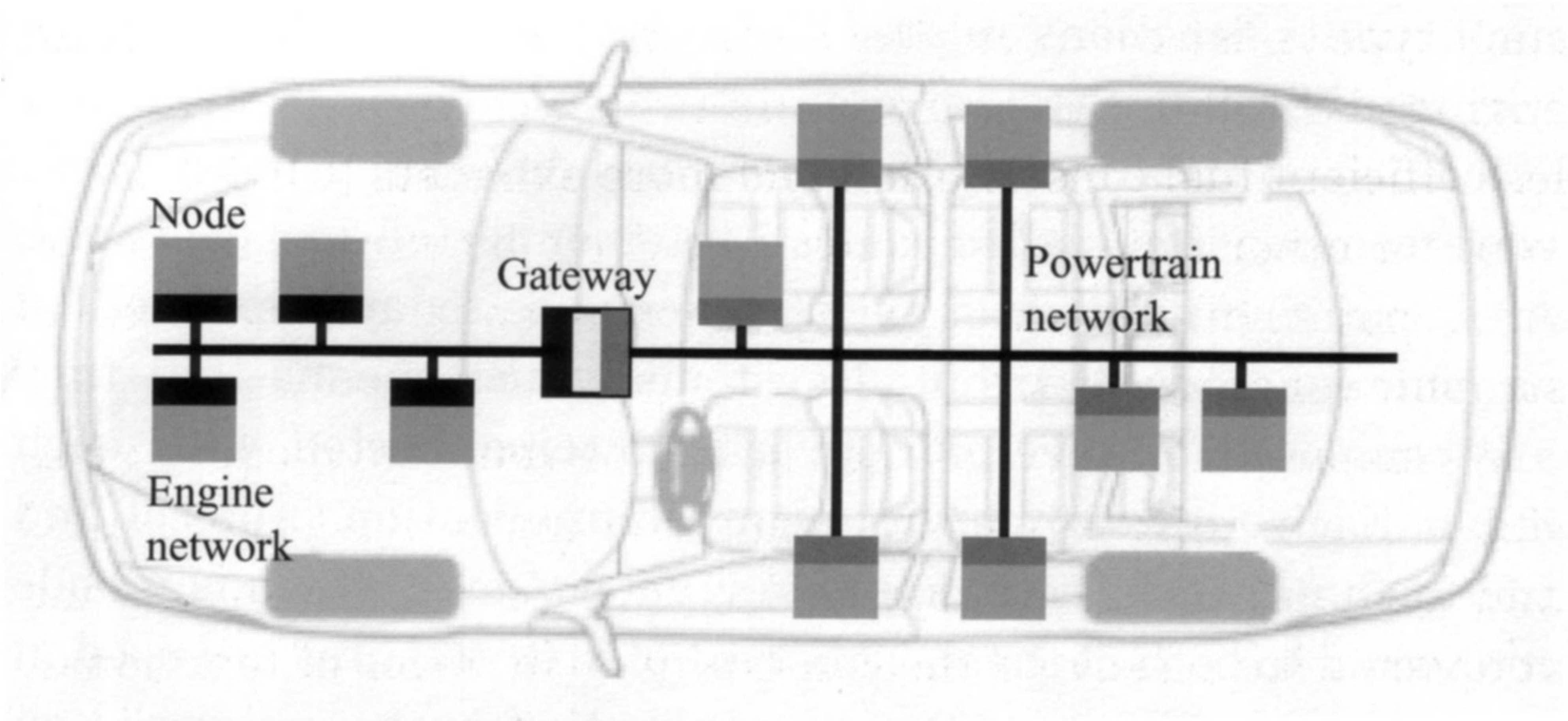
Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- Zusammenfassung
- Literatur

Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- Zusammenfassung
- Literatur

Verteiltes Embedded System im Automobil



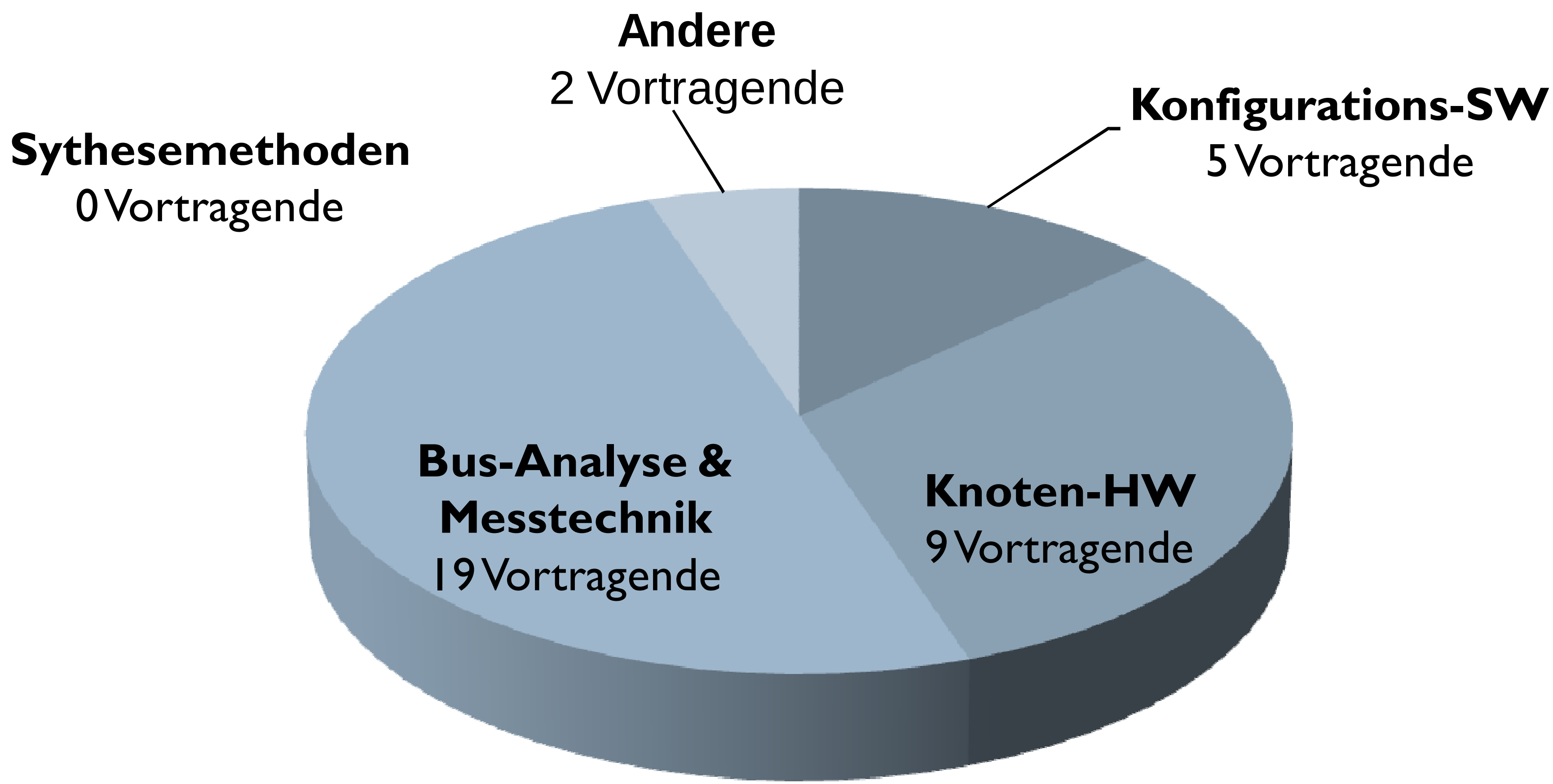
Verteilte embedded Systeme

- Steigende Anzahl an ereignisgesteuerten Systemen
  - Führt zu hoher Buslast
  - Antwortverhalten nicht vorhersagbar,
    führt zu keinem Determinismus
  - Nicht beherrschbare Folgen

- Lösung: zeitgesteuerter Buskommunikation z.B. FlexRay
  - Garantiertes Antwortverhalten liefert Determinismus
  - Aufgabe: Timing Tasksystem auf Kommunikationssystem

Entwicklungsschwerpunkte FlexRay

- FlexRay Productday 29.11.2007



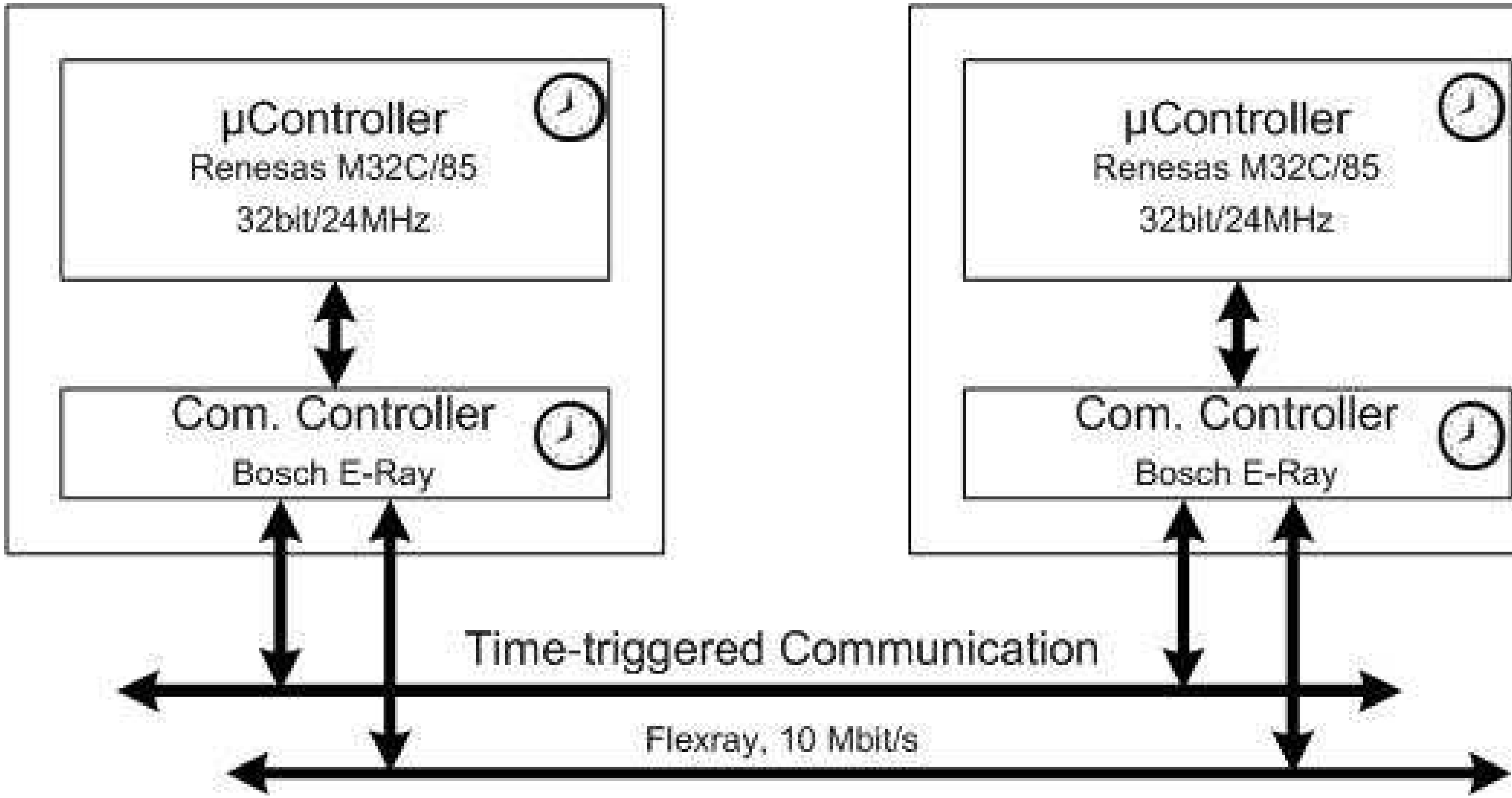
Ziele zum Einstieg in das Marktsegment

- Kenntnisse zur Anwendung der Konfigurationssoftware
- Kenntnisse zur Verwendung der Knoten-Hardware
- Durchdringung wissenschaftlicher Methoden zum Timing der Taskaktivierung und Abstimmung auf das Kommunikationssystems

Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- Zusammenfassung
- Literatur

Aufbau eines verteilten Hard Real-Time Systems



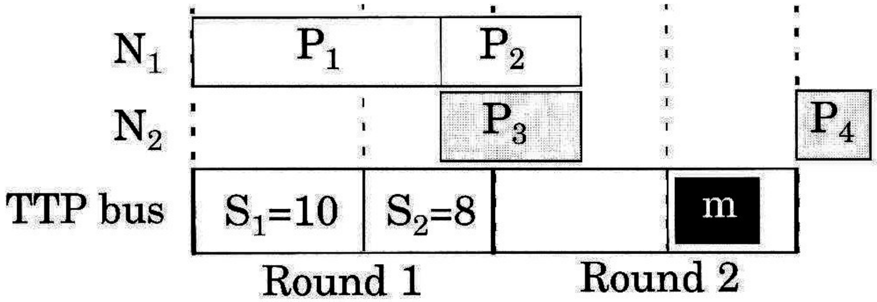
Aufgabe in verteilten Hard Real-Time Systemen

- Buskonfiguration
- Zeitgesteuerte Taskaktivierung

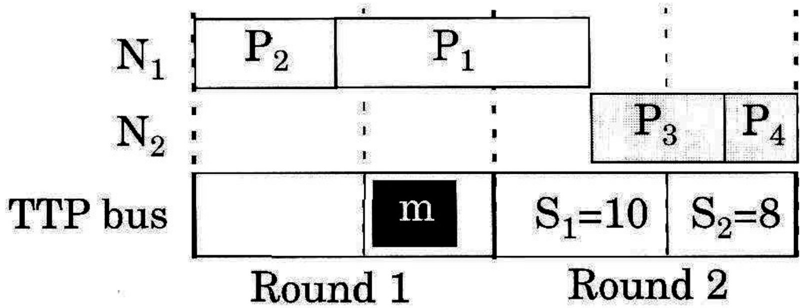
- Aufgabe: Systemsynchronisation

- Lösung
  - Com.-Controller synchronisiert über Protokoll
  - Com.-Controller startet Taskaktivierung

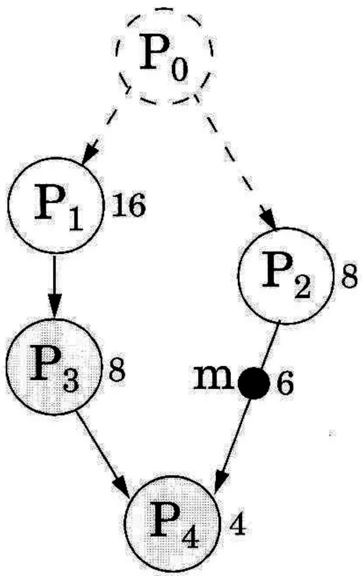
Beispiel



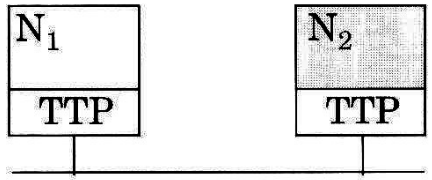
a) Schedule length of 40 ms



b) Schedule length of 36 ms

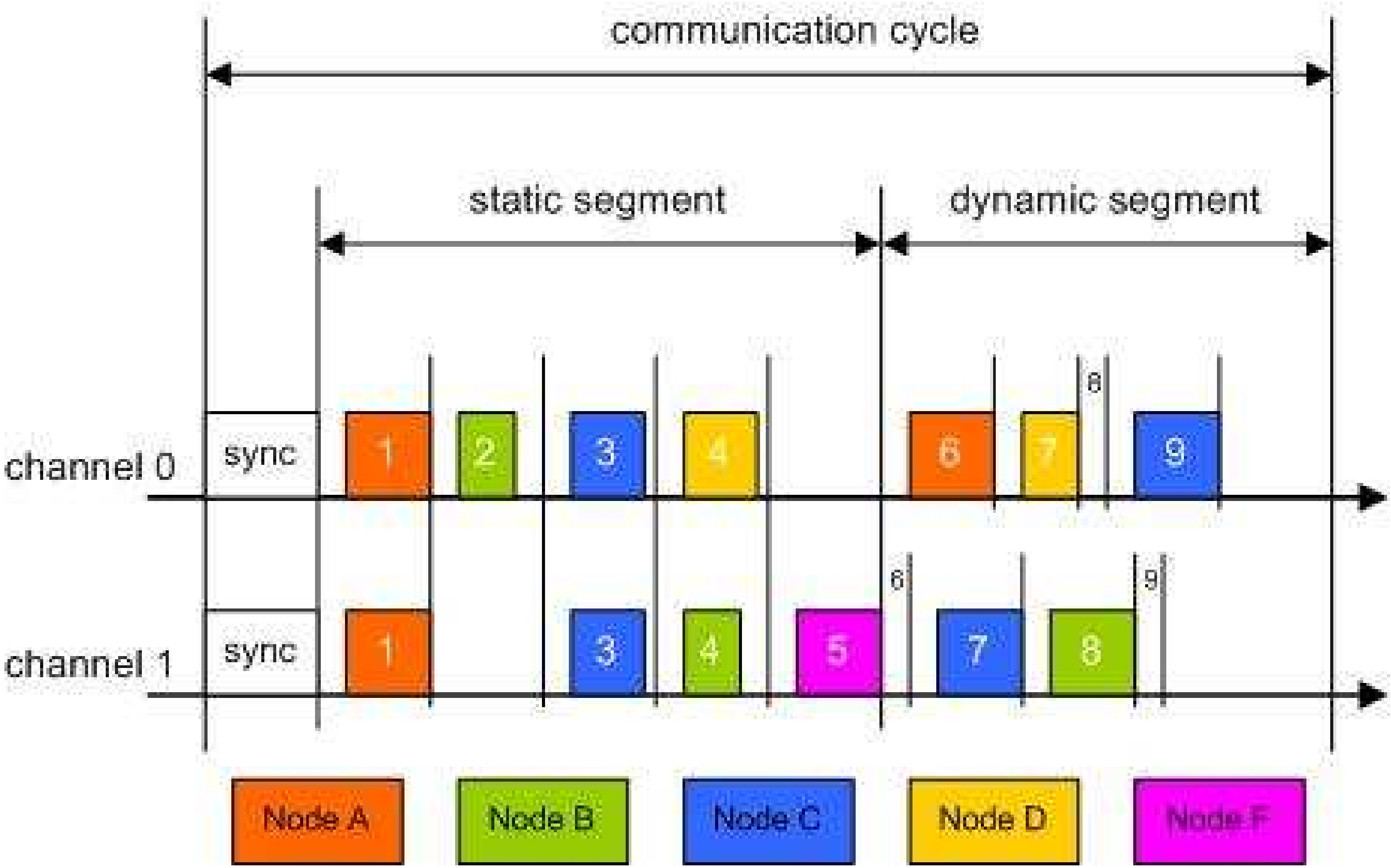


c) Application



d) Architecture

FlexRay TDMA-Runde



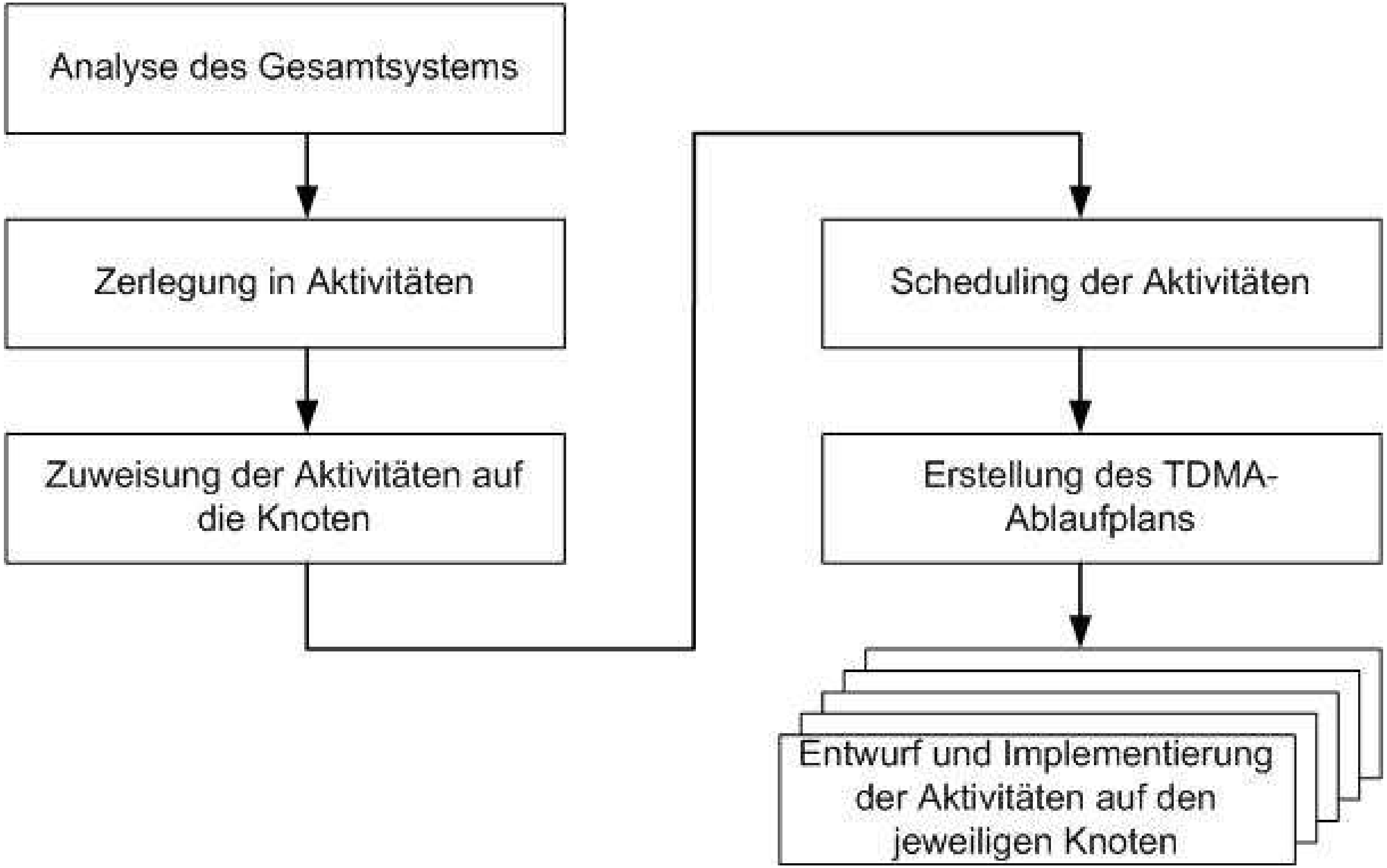
Praxisbeispiele

- BMW X5
  - aktives Fahrwerk
- SCV
  - Verteilte Geschwindigkeits- und Lenkwinkelregelung
  - Koordininierung
- HAWKS Racing
  - FlexRay als Bussystem
  - Modell 2008

Agenda

- Motivation
- Zeitgesteuerte Systeme
- **Anwendungsmodellierung**
- Thesis Outline
- Zusammenfassung
- Literatur

Entwicklungsphasen



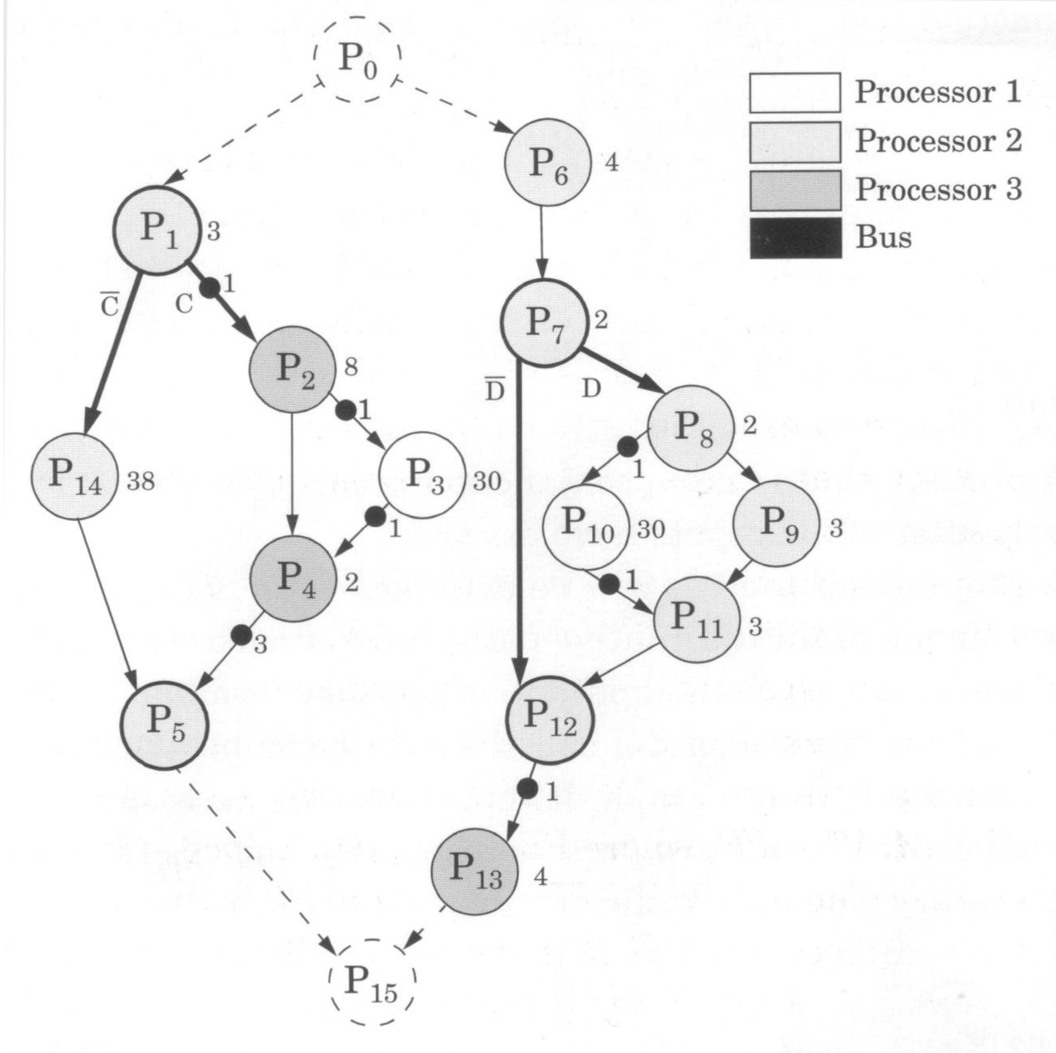
Entwurfsansatz

- Embedded Systems Lab der Universität Linköping

- Ziel:
  - Generieren eines statischen Schedules für ein Tasksystem
  - Erzeugung eines Schedules mit minimaler Ausführungszeit
  - Optimierung der Parameter des Kommunikationsprotokolls
    - Dimensionierung der Slot-Größen
    - Festlegung der Slot-Reihenfolge

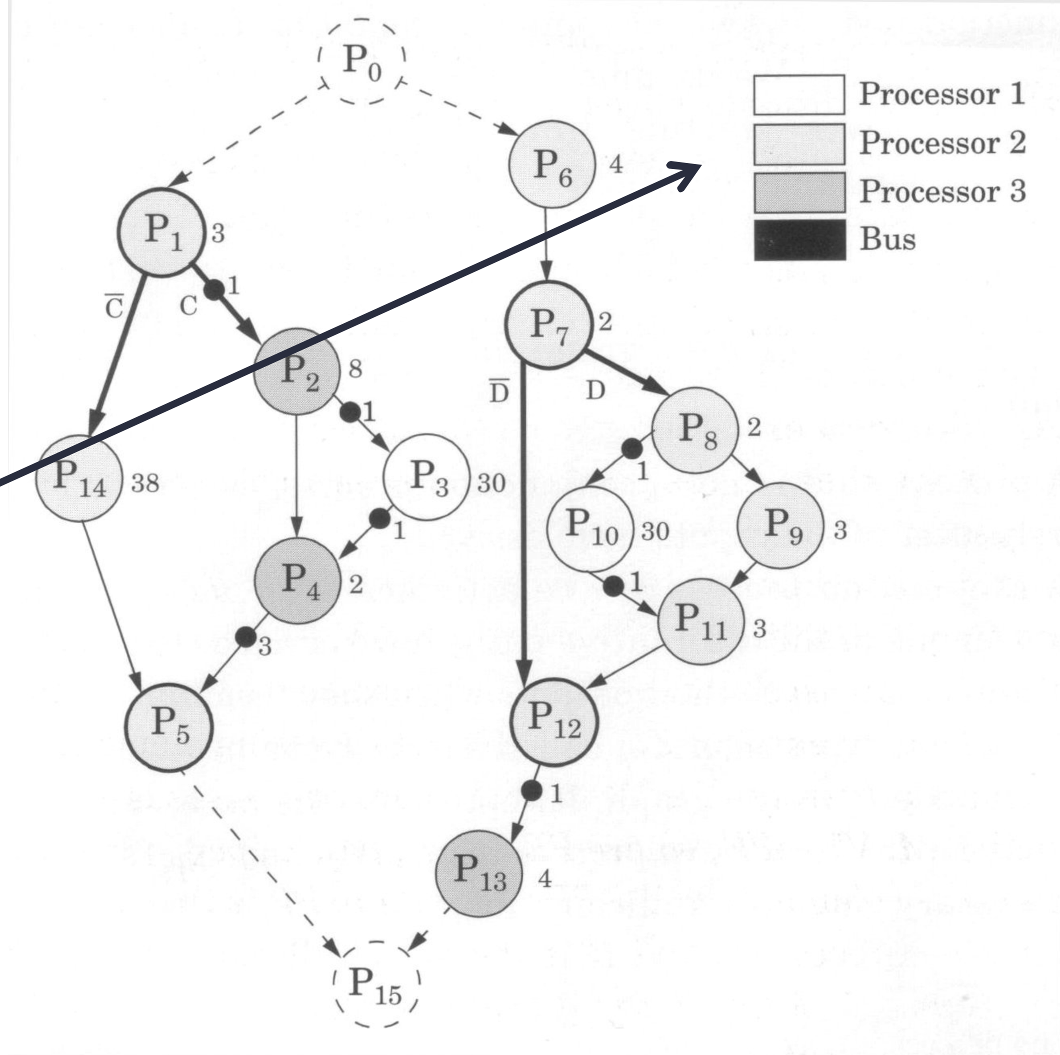
Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)



Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

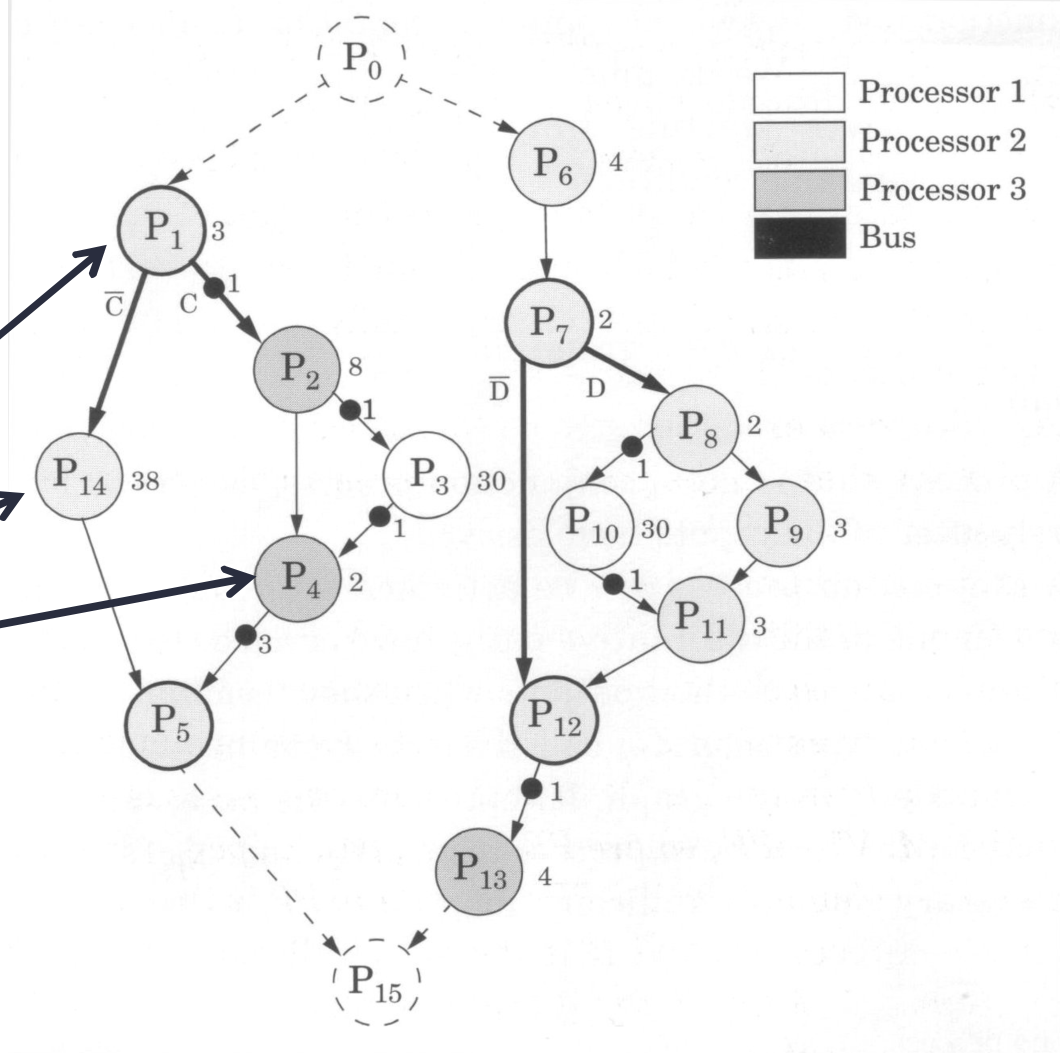


Ressourcen des Systems

Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

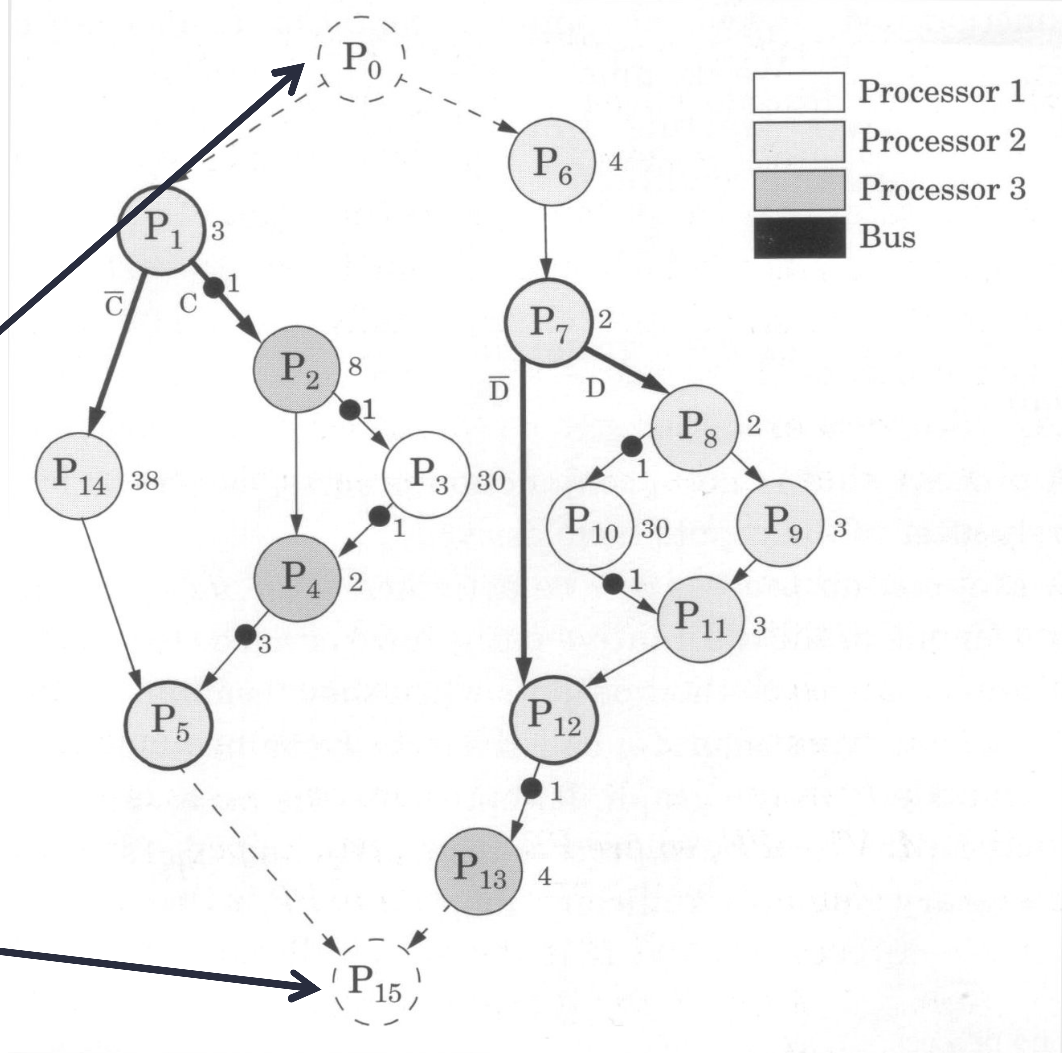
Prozesse:
hier bereits Ressourcen zugeordnet



Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

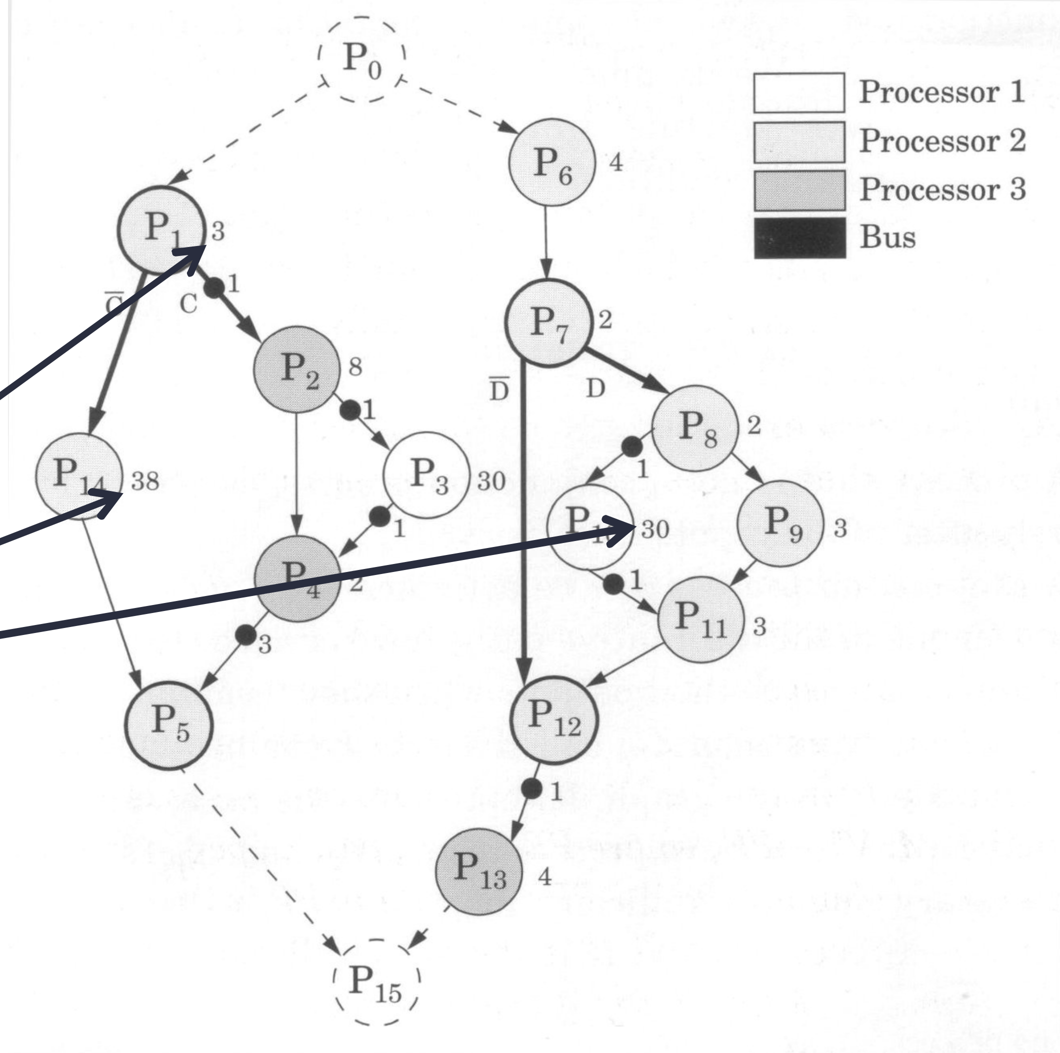
Polarer Graph:
Quelle und Senke



Modellierungsansatz mit CPG

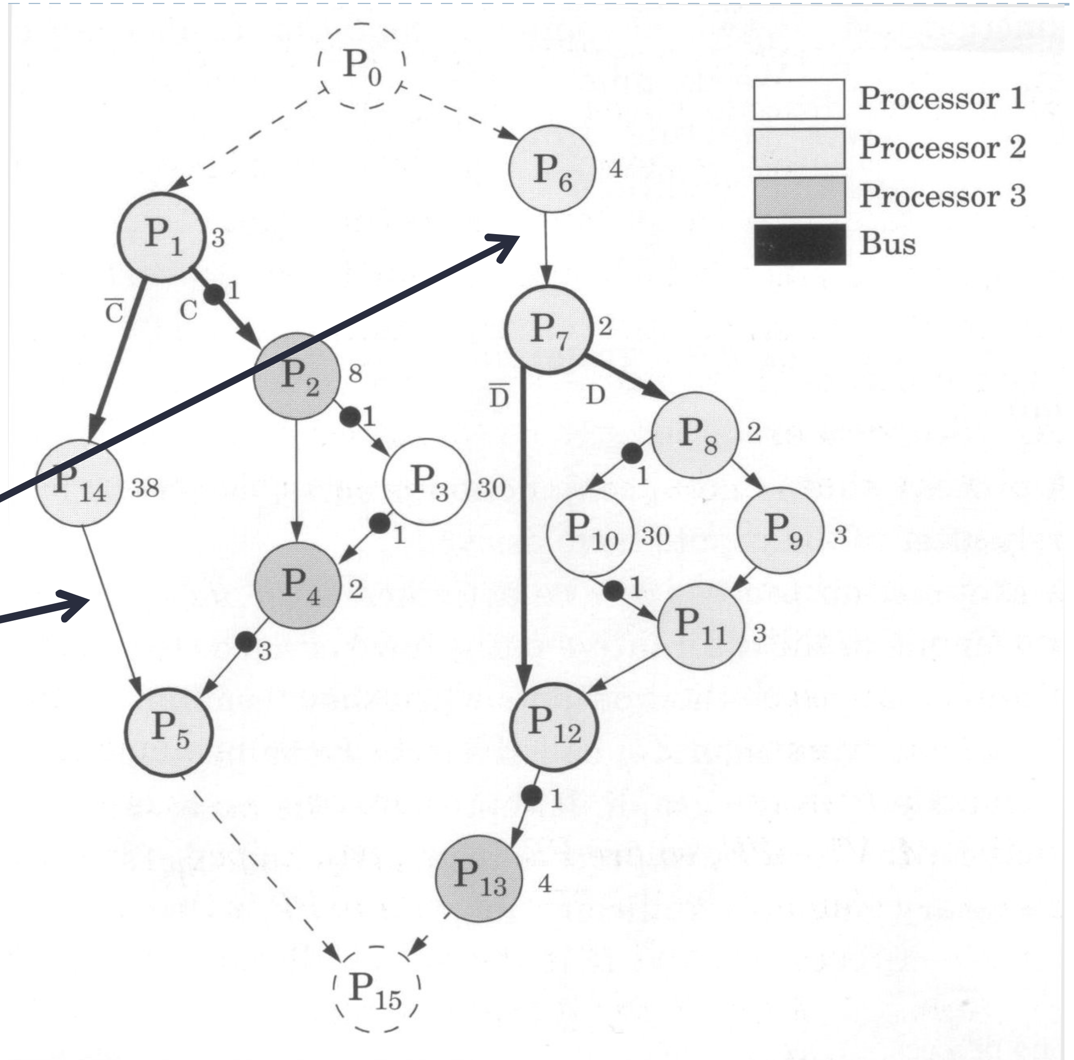
- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

Worst Case Execution Time (WCET) der Prozesse



Modellierungsansatz mit CPG

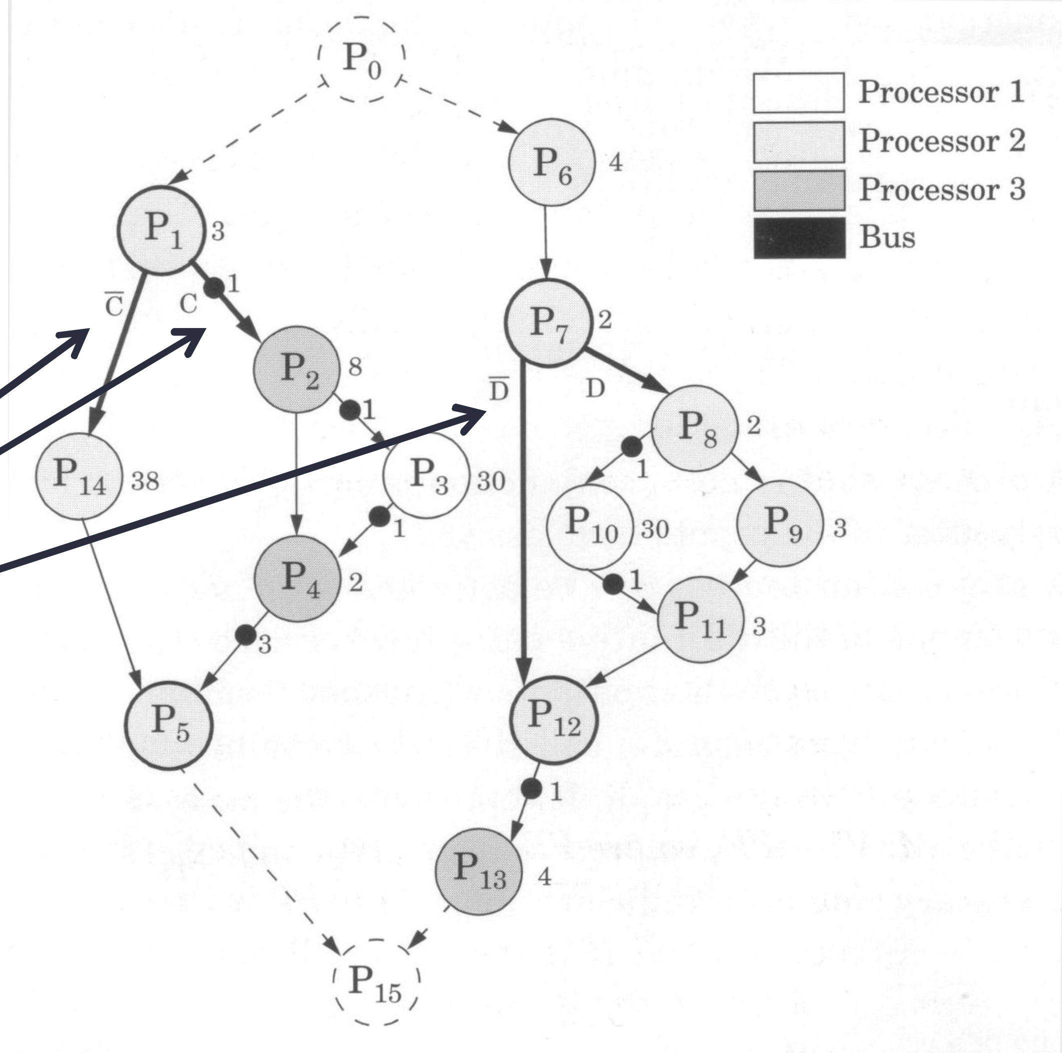
- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)



Gerichtete Kanten: Abhängigkeiten zwischen Prozessen

Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

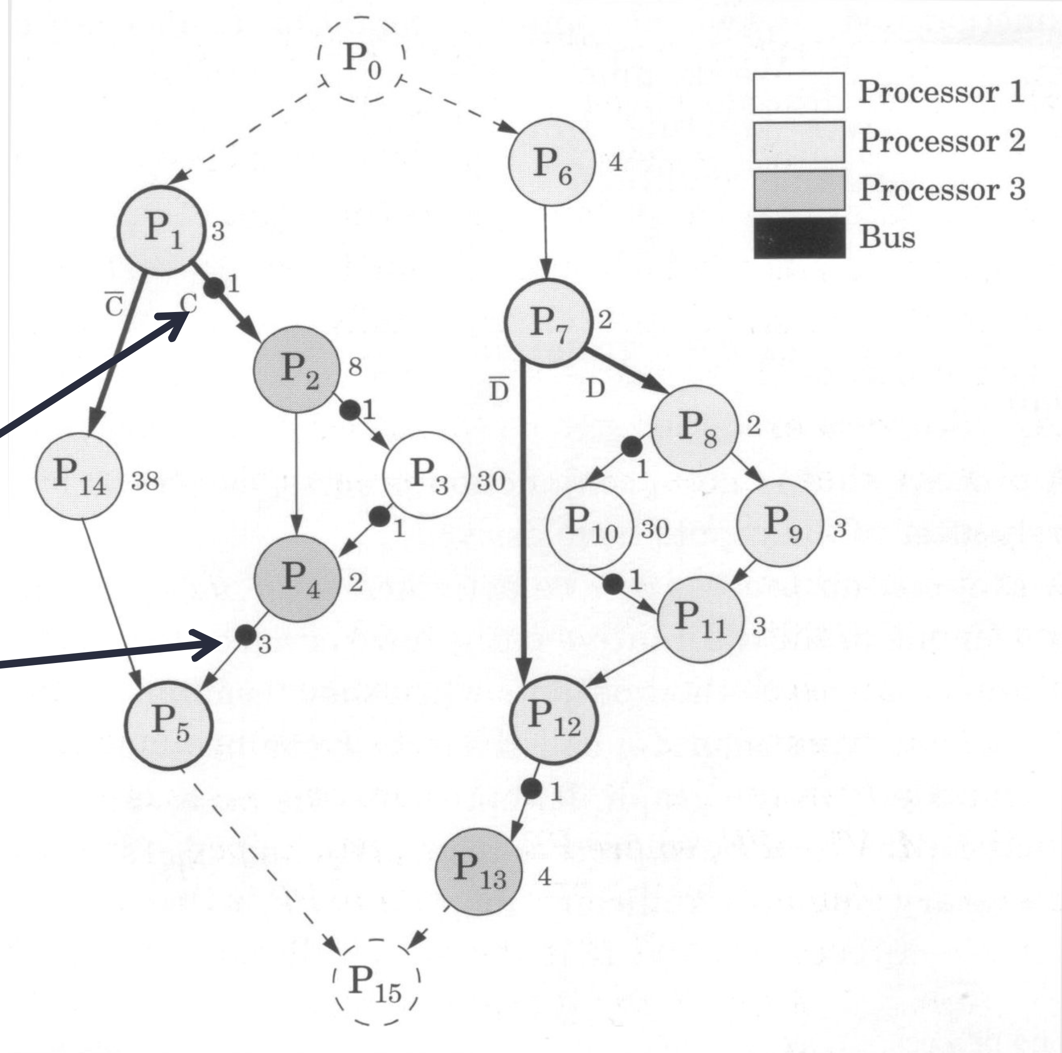


Konditionierte Kanten: Alternative Pfade

Modellierungsansatz mit CPG

- Modellierung interagierender Prozesse in Conditional Process Graphs (CPG)

Datenaustausch über den Bus



List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

CurrentTime: Aktueller Ausführungs-zeitpunkt

List Scheduling Funktion

ReadyList: Enthält ausführbare Prozesse

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

KnownConditions: Derzeit bekannte Konditionswerte

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

Update Funktion: Aktualisiert Liste der ausführbaren Prozesse

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

GetReadyProcess Funktion:
Liefert Prozess mit höchster Priorität

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

Insert Funktion: Schreibt den ausgewählten Prozess in eine Schedule Table

List Scheduling Funktion

```
ListScheduling(CurrentTime, ReadyList, KnownConditions)
repeat
    Update(ReadyList)
    for each processing element PE do
        if PE is free at CurrentTime then
            P_i = GetReadyProcess(ReadyList)
            if there exists a P_i then
                Insert(P_i, ScheduleTable, CurrentTime, KnownConds)
                if P_i is a disjunction process then
                    C_i = condition calculated by P_i
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the true branch,
                        KnownConditions ∪ true C_i)
                    ListScheduling(CurrentTime,
                        ReadyList ∪ ready nodes from the false branch,
                        KnownConditions ∪ false C_i)
                end if
            end if
        end if
    end for
    CurrentTime = earliest time when a scheduled process terminates
until all processes of this alternative path are scheduled
end ListScheduling
```

Rekursiver Aufruf: Nachfolger der Prozesse, die Konditionen erzeugen, werden eingeplant

List Scheduling Funktion

- Prioritätsfunktion
  - Bestimmt die Taskreihenfolge konkurrierender Tasks
- Schedule Table
  - Enthält Aktivierungszeit der Prozesse unter bestimmten Konditionen
  - Prozessorspezifischer Teil wird im Knoten gespeichert
  - Dispatcher auf den Knoten zur Aktivierung der Prozesse

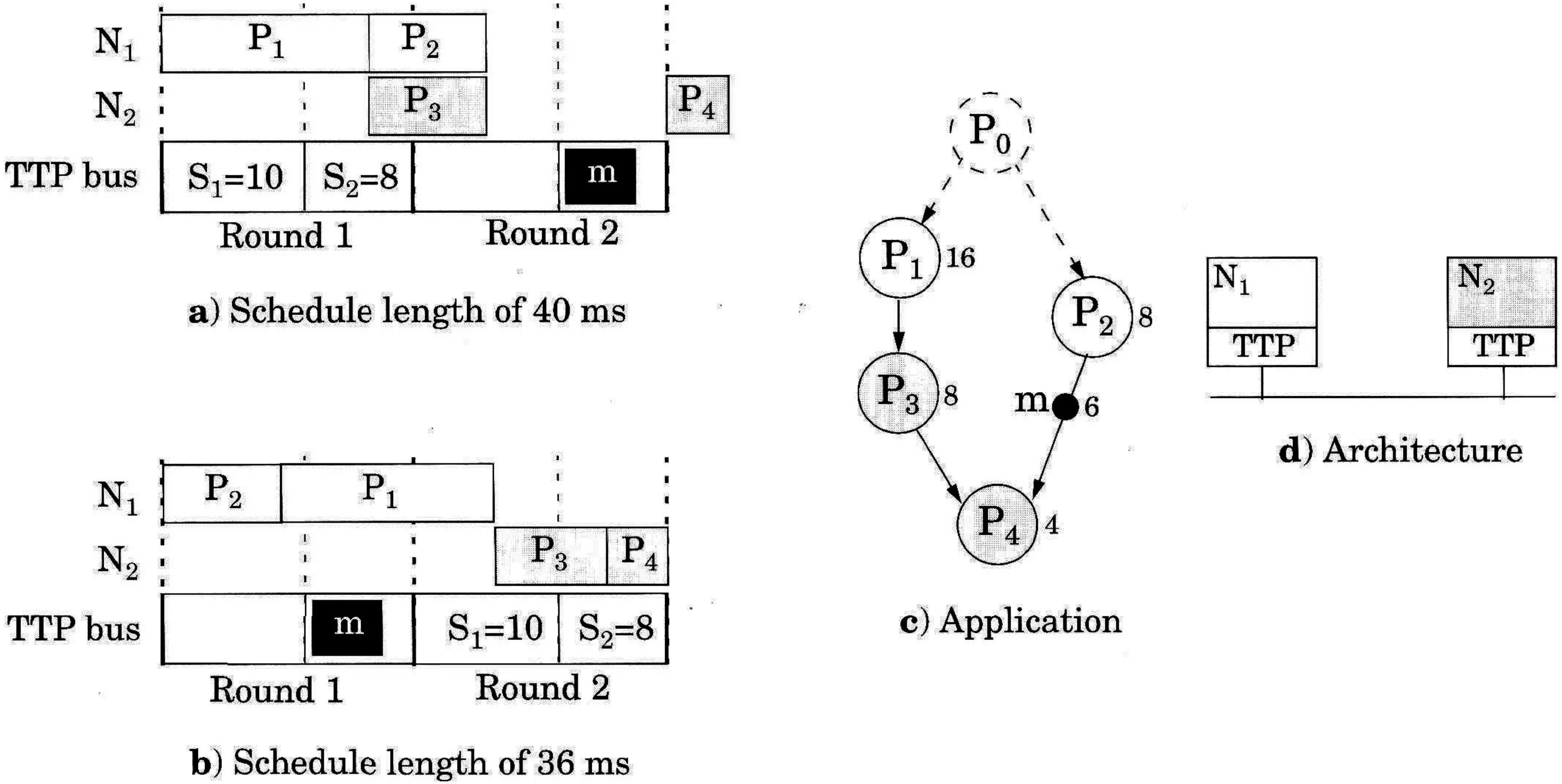
Nachrichten Scheduling

- Kommunikation wird als Prozess behandelt, der auf der Ressource Bus aktiviert wird
- Nachricht wird im frühestmöglichen Slot gesendet
- Beachtung des TDMA-Schemas
- Nachrichten-Scheduling findet während des Prozess-Schedulings statt

Optimierung der Bus-Konfiguration

- Ziel: Erzeugung eines kürzeren System-Delays
  - Dimensionierung der Slot-Größen
  - Spezifizieren der Slot-Zuordnungen

Beispiel



a) Schedule length of 40 ms

b) Schedule length of 36 ms

c) Application

d) Architecture

Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- Zusammenfassung
- Literatur

Thesis Outline

- Untersuchung des vorgestellten Ansatzes
- Komplettsicht des konkreten Algorithmus
- Vergleich mit anderen Verfahren,
  z.B. Bereich MPSoC, Verfahren in MA Sellentin
- Implementierung eines konkreten Beispiels, z.B. SCV: Einparkassistent, Kollisionsassistent
- Erweiterung der Untersuchung um den Mapping-Aspekt

Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- **Zusammenfassung**
- Literatur

Zusammenfassung

- Zeitgesteuerte Systeme ermöglichen Systeme mit definiertem Antwortverhalten
- FlexRay bietet ein Kommunikationsmedium
- Aufgaben
  - Timing der Prozesse
  - Konfiguration des Busses
  - Minimierung der Ausführungszeit der Anwendung

Agenda

- Motivation
- Zeitgesteuerte Systeme
- Anwendungsmodellierung
- Thesis Outline
- Zusammenfassung
- Literatur

Literatur

- G. C. Butazzo:
  Hard Real-Time Computing Systems
  Spinger, 2005
  ISBN: 0-387-23137-4
- P. Pop, P. Eles, Z. Peng:
  Analysis and Synthesis of Distributef Real-Time Embedded Systems
  Kluwer Academic, 2004
  ISBN: 1—4030-2872-5
- T. Pop, P. Pop, P. Eles, Z. Peng, A. Andrei:
  Timing Analysis of the FlexRay Communication Protocol
  Computer and Information Science Dept., Linköping University, Sweden
  Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06), 2006
- P. Eles, A. Doboli, P. Pop, Z. Peng
  Scheduling with Bus Access Optimization for Distributed Embedded Systems
  IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 8, NO. 5, OCTOBER 2000

Literatur

- W. Wolf, A. Jerraya:
  Multiproceessor Systems-on-Chip
  Morgan Kaufmann Series, 2005
  ISBN: 9780123852519
- P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, P. Pop
  1089-6503/98 , *1998 IEEE*
- J. Sellentin:
  Masterarbeit: Ein zeitgesteuertes, verteiltes SW-Konzept implementiert auf FlexRay-Komponenten für ein fahrerloses Transportsystem
  HAW Hamburg, 2006