FACHHOCHSCHULE TRIER
FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

Migration von AUTOSAR-basierten Echtzeitanwendungen auf Multicore-Systeme

Michael Bohn / Jörn Schneider /
Christian Eltges / Robert Rößger

Fachhochschule Trier

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

News zur Fachhochschule Trier

FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik

30. Mai 2011: Sieg in Shell Eco Marathon Europe sowie CO_2-Award

18. Mai 2011: Promotionsmöglichkeit für FH Trier: nur 7 FHs bundesweit erhielten Forschungskolleg

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

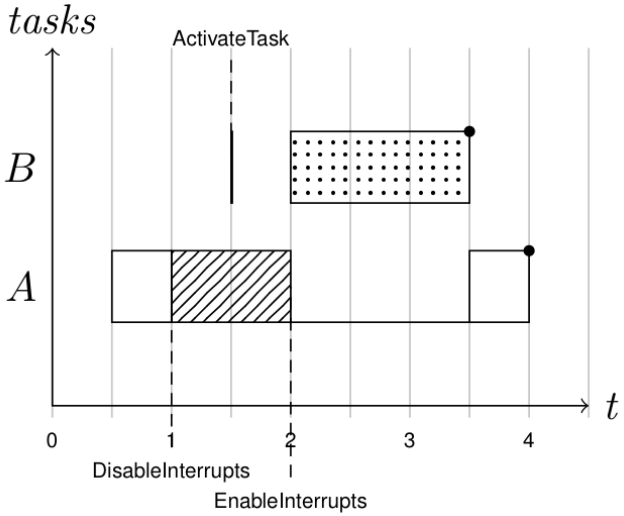
Ausgangssituation

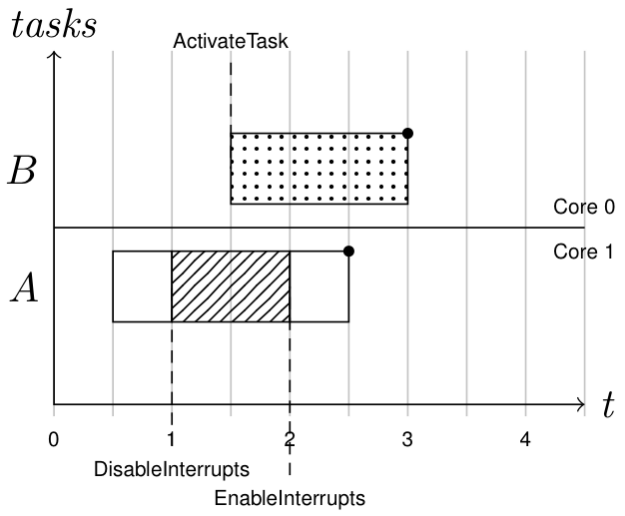
- Existierende Steuergeräte-Software im Kfz
  - Entwickelt für Einzelprozessorsysteme
  - Verwendet entsprechende Synchronisationsmechanismen
    - z.B. Interruptsperren oder Taskprioritäten
  - Betriebssystem
    - OSEK
    - AUTOSAR
- Migration auf Multicore-Systeme
  - Jetzt echte Nebenläufigkeit
    - ➔ Neue Race Conditions möglich

Migration

- Standardlösung: Schützen kritischer Abschnitte durch wechselseitigen Ausschluss
  - Sind im Einzelprozessorsystem auch geschützt
    - Warum funktioniert das nicht im Multicore Fall?

Beispiel Interruptsperre:





Lösungsidee

- Das Betriebssystem/die Laufzeitumgebung erledigen das

- Probleme:
  1. Wie werden wechselseitige Ausschlüsse auf Multicore Systemen realisiert?
  2. Welche wechselseitigen Ausschlüsse müssen gewahrt bleiben?

Seitenblick: Realisierung auf Multicore Systemen

- AUTOSAR: nur Spinlocks
  - Potentielle Deadlocks
  - Potentielle Verschwendung von Rechenzeit
  - Priority Inversion
  - Unbounded Remote Blocking
- Untersuchung echtzeitfähiger Alternativen in Arbeit
  - Simulator: SimTrOS

Welche wechselseitigen Ausschlüsse müssen gewahrt bleiben?

- Alle?
  - Problem: Viele wechselseitige Ausschlüsse ergeben sich auf Einzelprozessoren ungewollt
  - Konsequenz: Parallelität nicht nutzbar!

➔ Keine sinnvolle Lösung

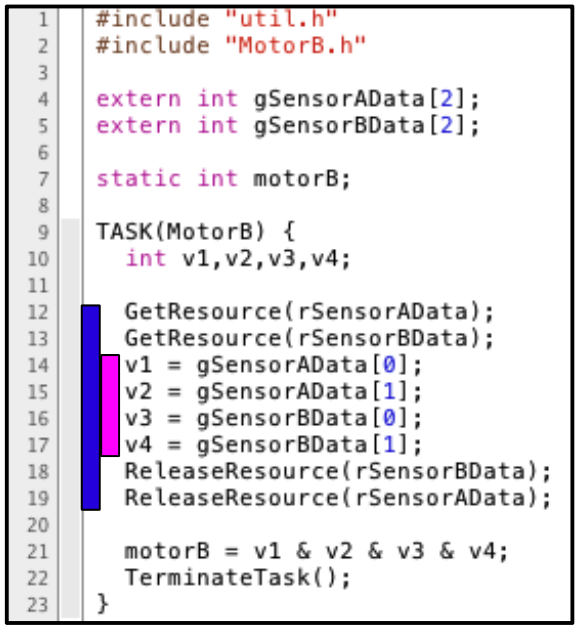
- Also nur die Notwendigen!
  - Welche sind notwendig?
  - Wie wird das ermittelt?

Unsere Lösung:

Automatisierung über zweistufiges Analysewerkzeug

Phase I: Ermitteln kritischer Abschnitte

- Mittels statischer Programm-Analyse
  - Bsp.
    - Interruptsperre
    - Get-/ReleaseResource
  - Eigens entwickelte flow-sensitive, generische API-Call Paar Analyse auf Hochsprachenebene



```
1  #include "util.h"
2  #include "MotorB.h"
3
4  extern int gSensorAData[2];
5  extern int gSensorBData[2];
6
7  static int motorB;
8
9  TASK(MotorB) {
10   int v1,v2,v3,v4;
11
12   GetResource(rSensorAData);
13   GetResource(rSensorBData);
14   v1 = gSensorAData[0];
15   v2 = gSensorAData[1];
16   v3 = gSensorBData[0];
17   v4 = gSensorBData[1];
18   ReleaseResource(rSensorBData);
19   ReleaseResource(rSensorAData);
20
21   motorB = v1 & v2 & v3 & v4;
22   TerminateTask();
23 }
```

Analyseeigenschaften

- Analyse findet mindestens alle vorhandenen wechselseitigen Ausschlüsse (WA)
- Zwei Arten von Pessimismus:
  - Vermeintliche WA
    - Überapproximation der Analyse (False Positives)
  - Vorhandene aber unnötige WA
    - Kollateralschaden, z.B. Interruptsperre
- Wie Pessimismus reduzieren?

Phase II: Reduktion der Einschränkungen

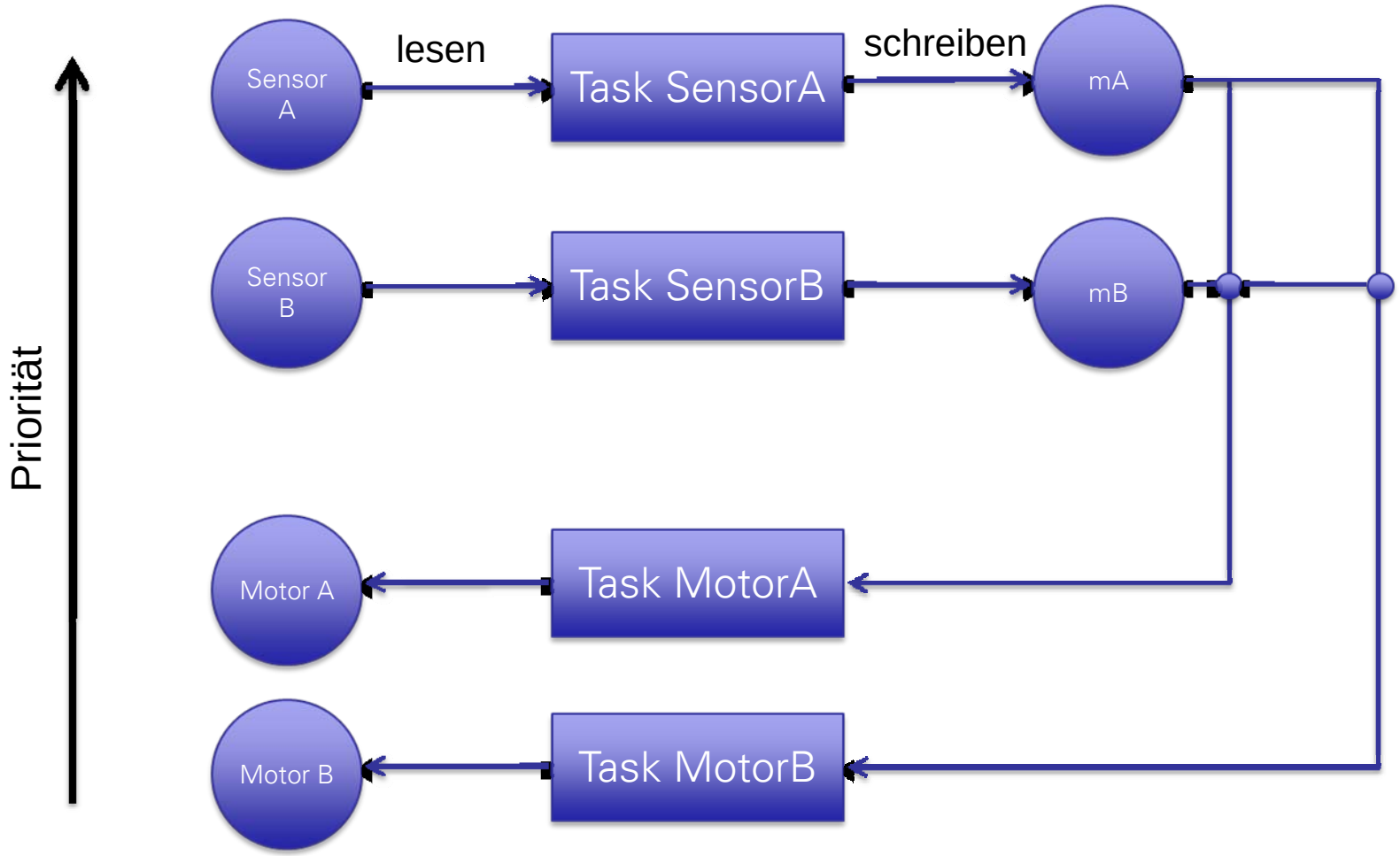
- Speicherzugriffsanalyse
  - Value Analyzer von AbsInt GmbH

- Wenn kein geteilter Speicher zwischen Programmabschnitten
  ➔ kein Grund für WA aus Programmsemantik erkennbar

- Achtung: Gründe abseits der Semantik sind möglich
  - Bsp. 2 Aktoren dürfen sich nicht gleichzeitig bewegen

Analyseablauf im Beispiel

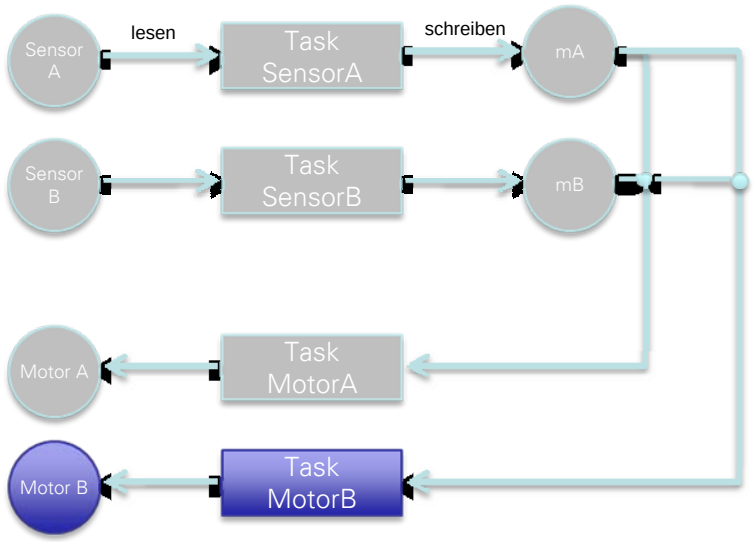
Beispielanwendung



Beispielanwendung

- Wechselseitiger Ausschluss – Task MotorB

```
1  #include "util.h"
2  #include "MotorB.h"
3
4  extern int gSensorAData[2];
5  extern int gSensorBData[2];
6
7  static int motorB;
8
9  TASK(MotorB) {
10   int v1,v2,v3,v4;
11
12   GetResource(rSensorAData);
13   GetResource(rSensorBData);
14   v1 = gSensorAData[0];
15   v2 = gSensorAData[1];
16   v3 = gSensorBData[0];
17   v4 = gSensorBData[1];
18   ReleaseResource(rSensorBData);
19   ReleaseResource(rSensorAData);
20
21   motorB = v1 & v2 & v3 & v4;
22   TerminateTask();
23 }
```

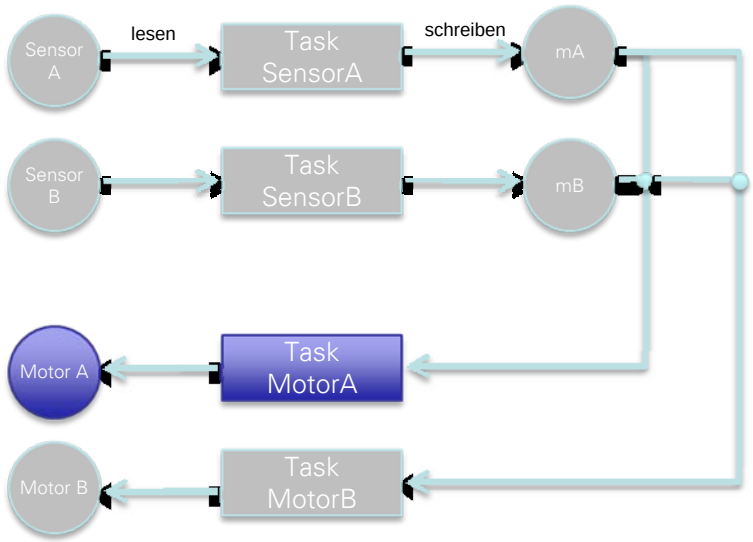


Beispielanwendung

- Wechselseitiger Ausschluss – Task MotorA

Non-Preemptive

```
1  #include "util.h"
2  #include "MotorA.h"
3
4  extern int gSensorAData[2];
5  extern int gSensorBData[2];
6
7  static int motorA;
8
9  TASK(MotorA) {
10   int v1 = gSensorAData[0];
11   int v2 = gSensorAData[1];
12   int v3 = gSensorBData[0];
13   int v4 = gSensorBData[1];
14
15   motorA = v1 & v2 & v3 & v4;
16   TerminateTask();
17 }
18
```



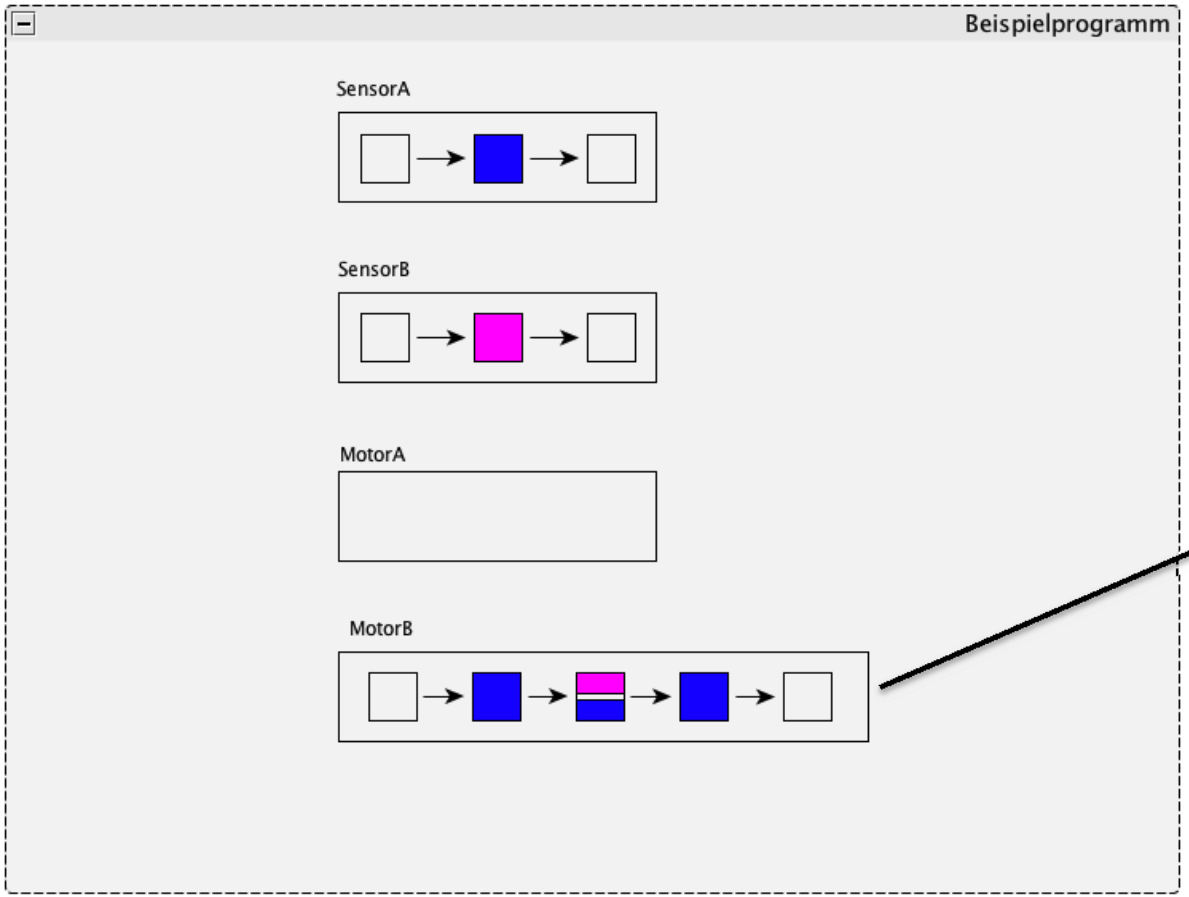
Phase I - Extraktion

Phase I: Extraktion

- Ermittlung *aller* vorhandenen *einseitigen* Ausschlüsse
- OSEK/AUTOSAR-Mechanismen:
  - höhere Priorität ➔ OIL/XML
    - Basispriorität
    - Priority-Ceiling
  - Nichtunterbrechbarkeit ➔ OIL/XML
  - gleiche Taskpriorität ➔ OIL/XML
  - Ressourcenmechanismus ➔ statische Programm-analyse + OIL/XML
  - Interruptsperre ➔ statische Programm-analyse

Phase I - Extraktion

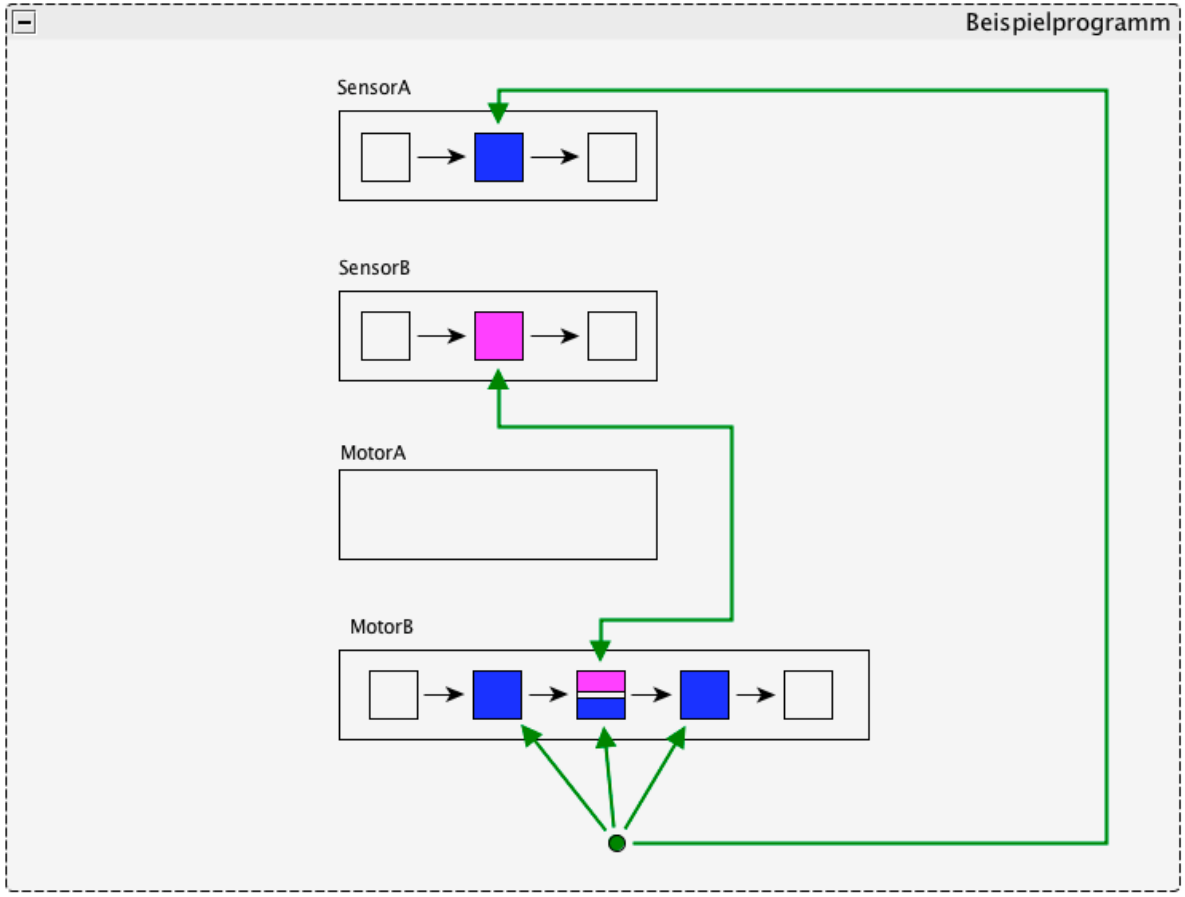
- Einseitiger Ausschlussgraph (EAG) - Basis




```
1  #include "util.h"
2  #include "MotorB.h"
3
4  extern int gSensorAData[2];
5  extern int gSensorBData[2];
6
7  static int motorB;
8
9  TASK(MotorB) {
10   int v1,v2,v3,v4;
11
12   GetResource(rSensorAData);
13   GetResource(rSensorBData);
14   v1 = gSensorAData[0];
15   v2 = gSensorAData[1];
16   v3 = gSensorBData[0];
17   v4 = gSensorBData[1];
18   ReleaseResource(rSensorBData);
19   ReleaseResource(rSensorAData);
20
21   motorB = v1 & v2 & v3 & v4;
22   TerminateTask();
23 }
```

Phase I - Extraktion

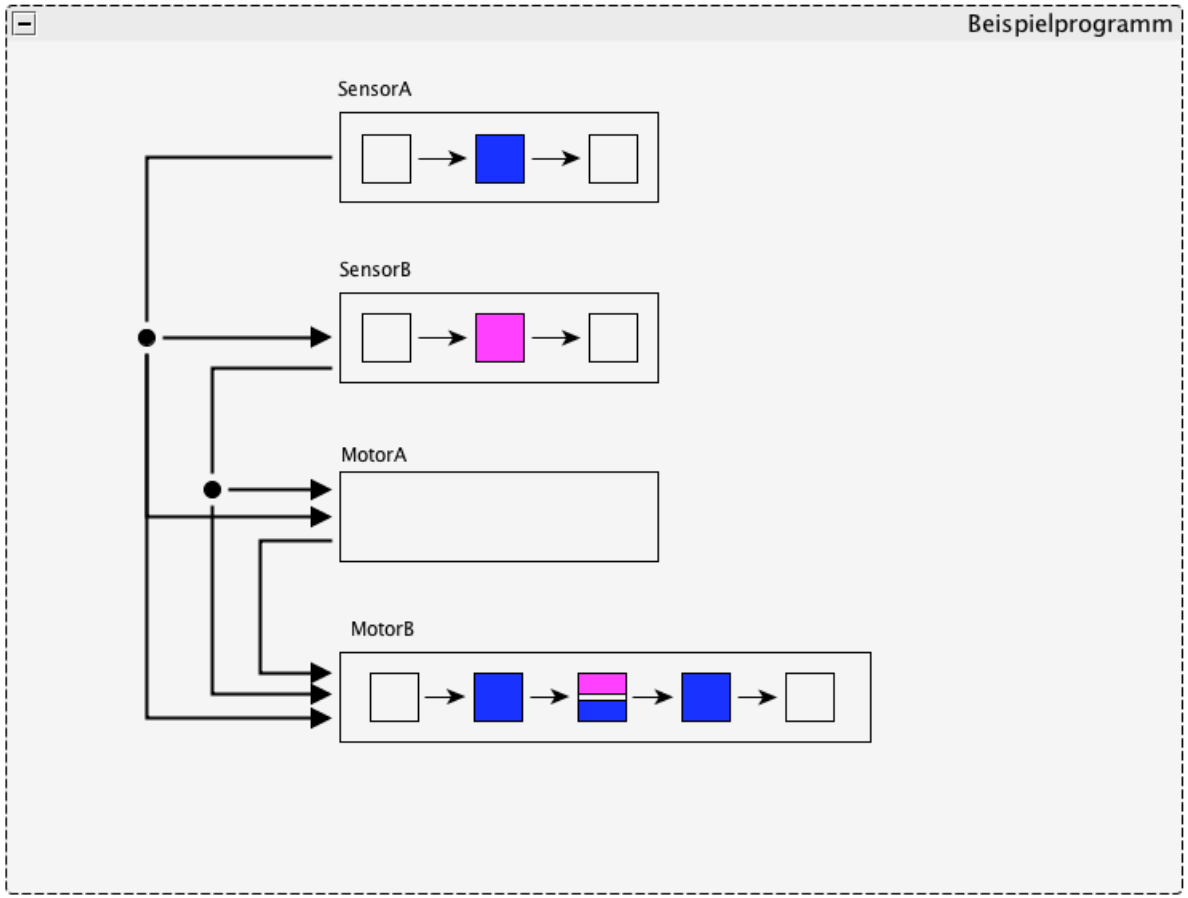
- Einseitiger Ausschlussgraph (EAG) - Ressourcen



Wechselseitiger Ausschluss über Ressourcen-mechanismus

Phase I - Extraktion

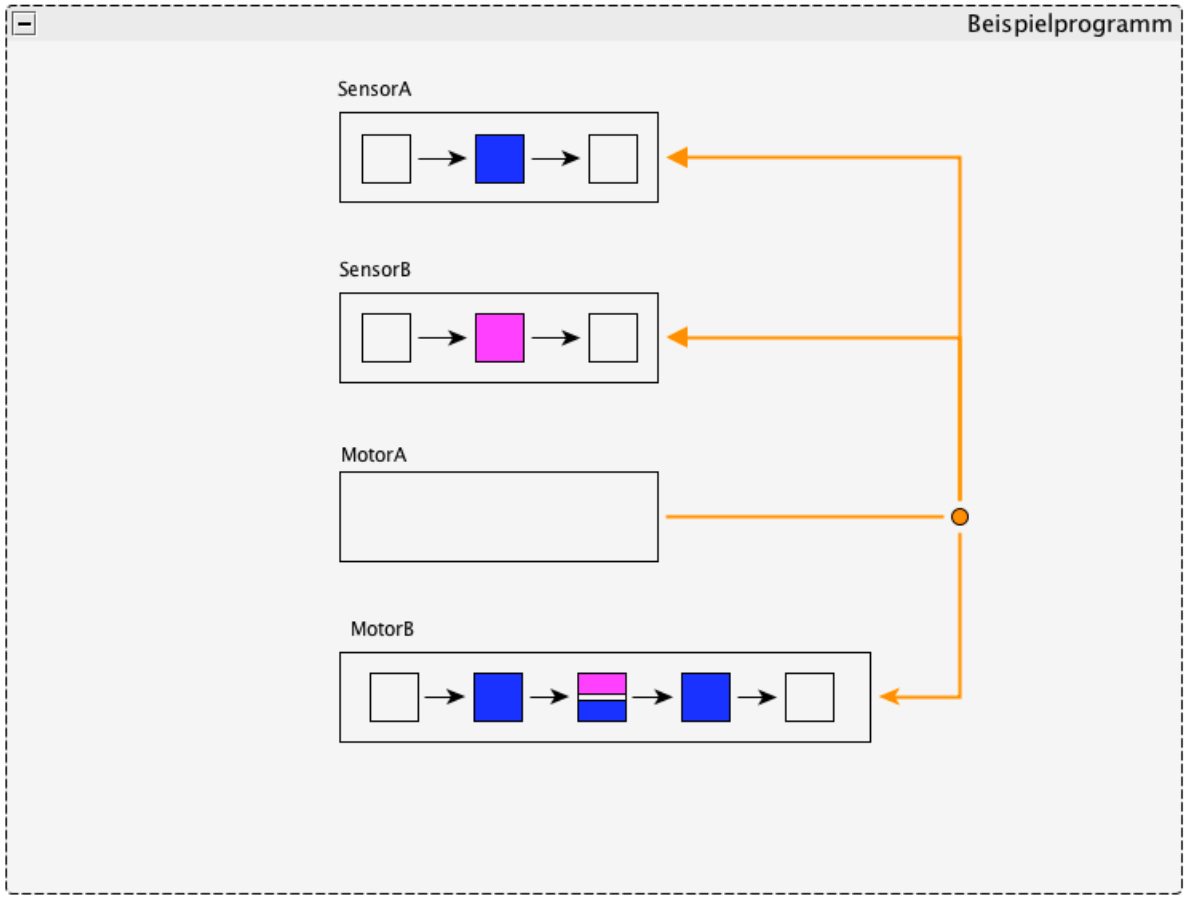
- Einseitiger Ausschlussgraph (EAG) – höhere Priorität



Einseitiger Ausschluss über höhere Priorität

Phase I - Extraktion

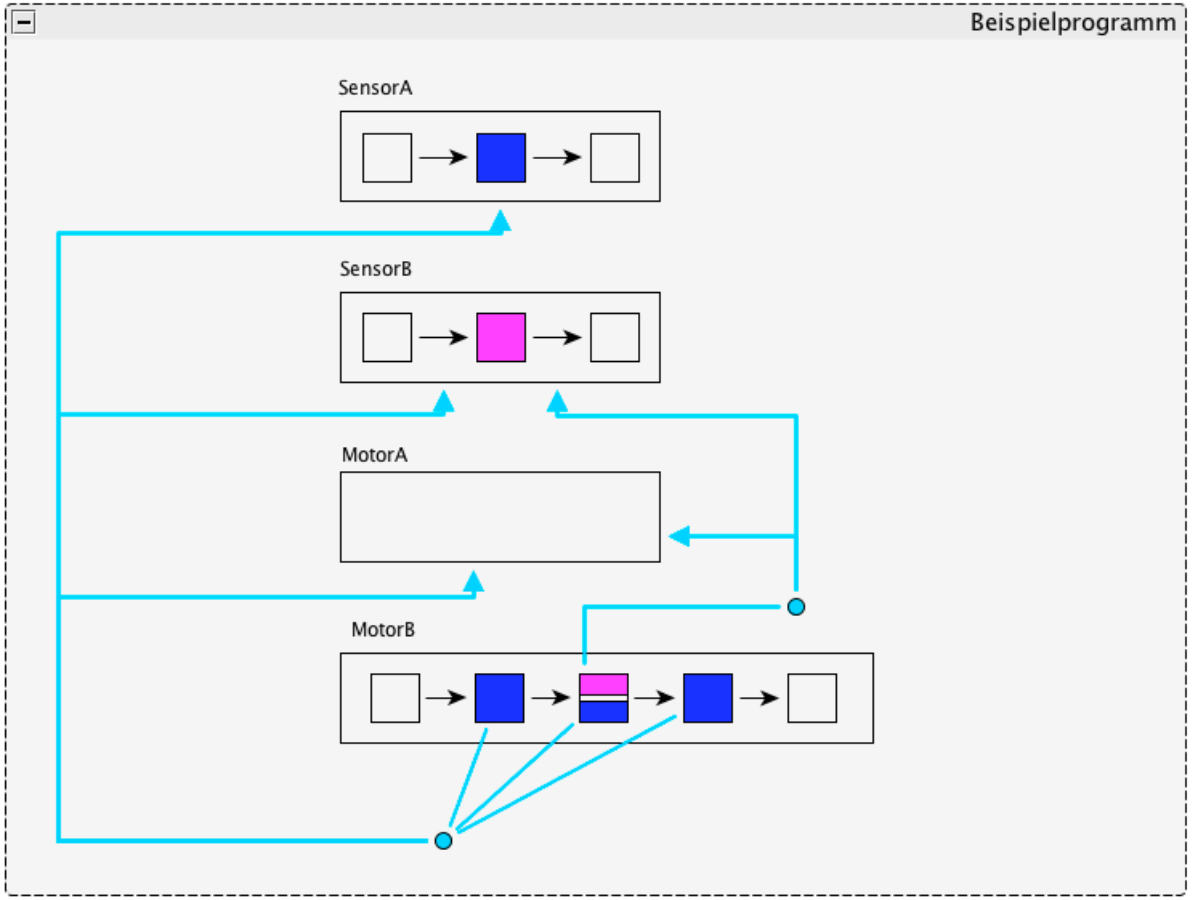
- Einseitiger Ausschlussgraph (EAG) - Nichtunterbrechbarkeit



Einseitiger Ausschluss über Nichtunter-brechbarkeit

Phase I - Extraktion

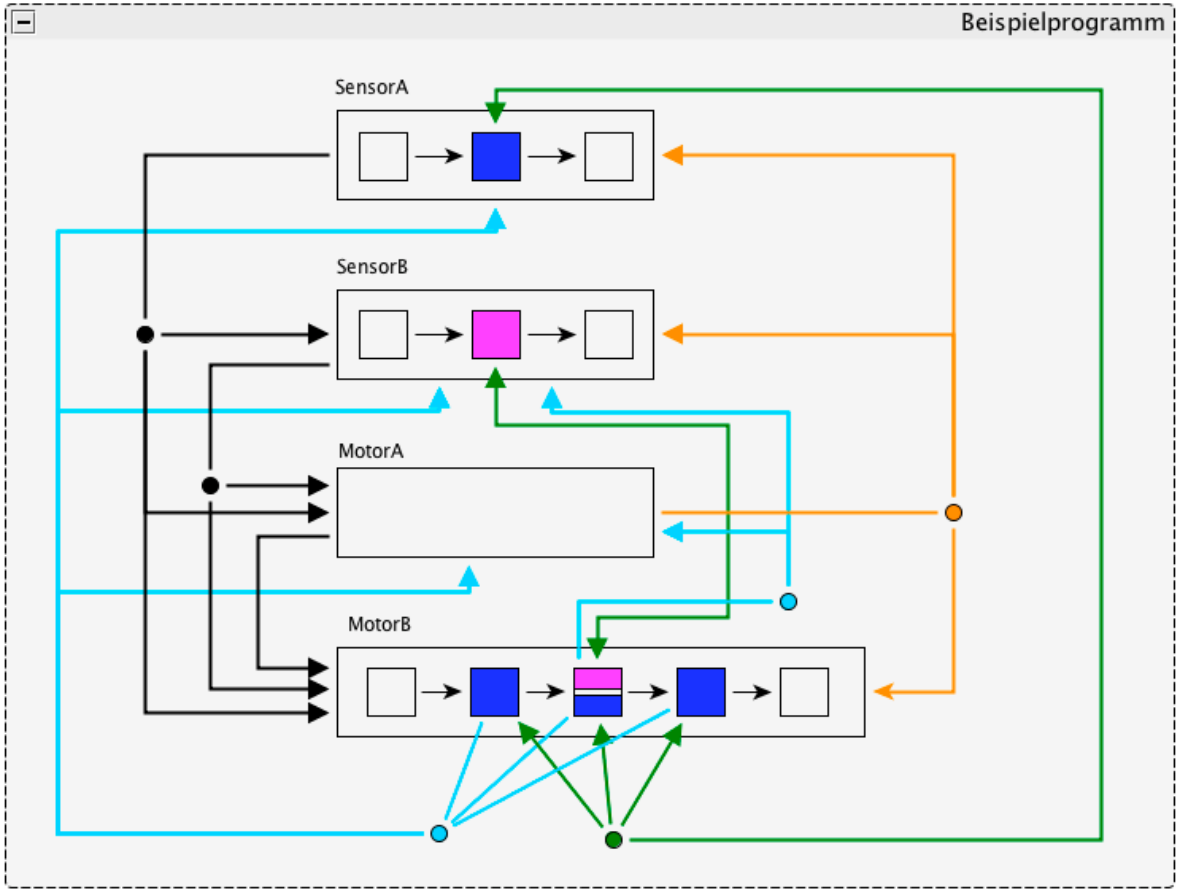
- Einseitiger Ausschlussgraph (EAG) – Priority Ceiling



Einseitiger Ausschluss über Priority Ceiling

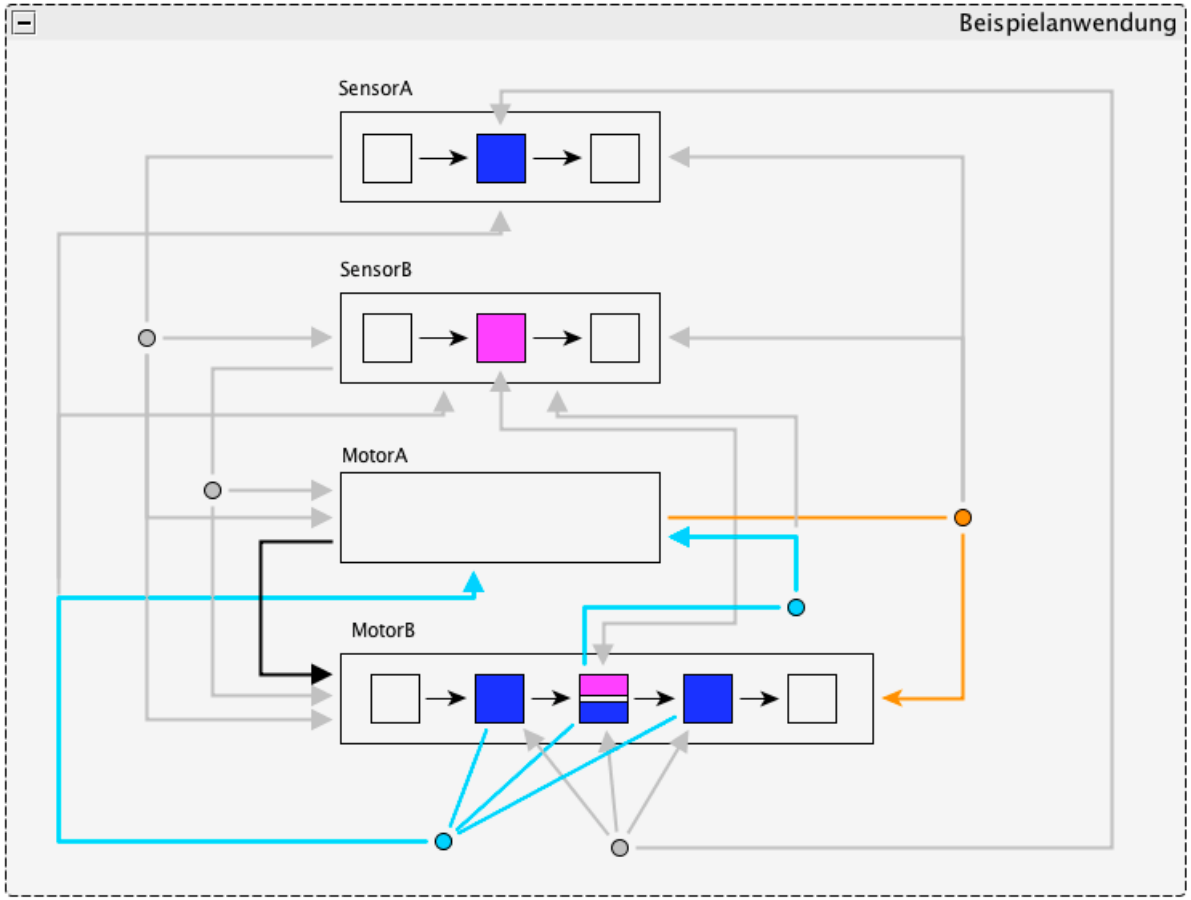
Phase I - Extraktion

- Einseitiger Ausschlussgraph (EAG) – Gesamtergebnis



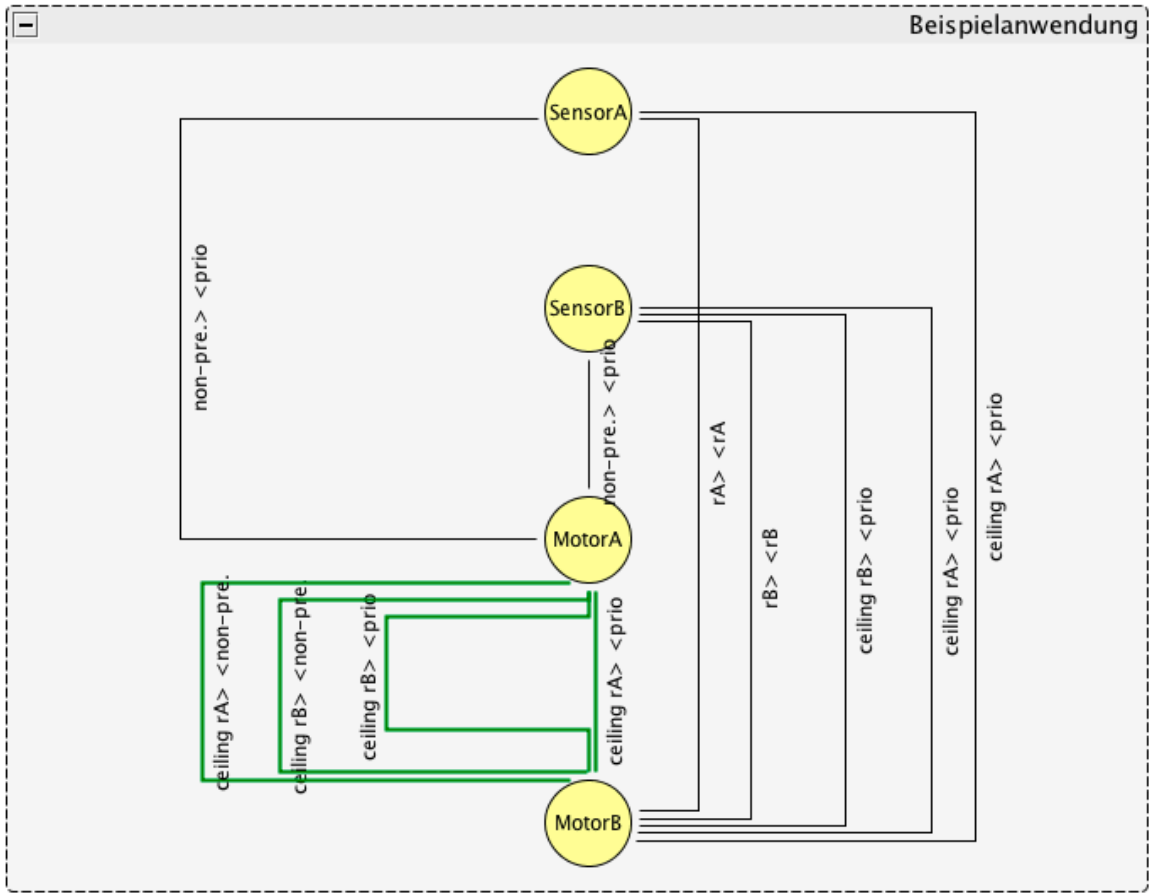
Phase I - Extraktion

- Ermittlung wechselseitiger Ausschlüsse



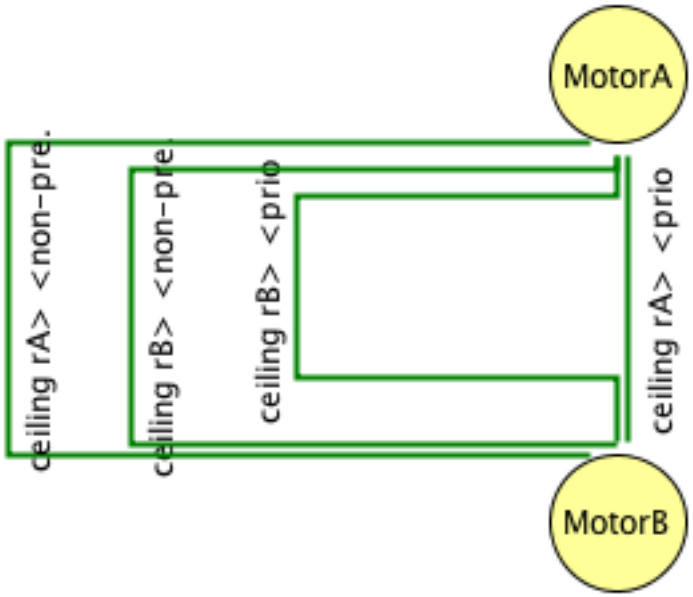
Phase I - Extraktion

- Constraint Graph (Extraktionsergebnis)



Phase II - Reduktion

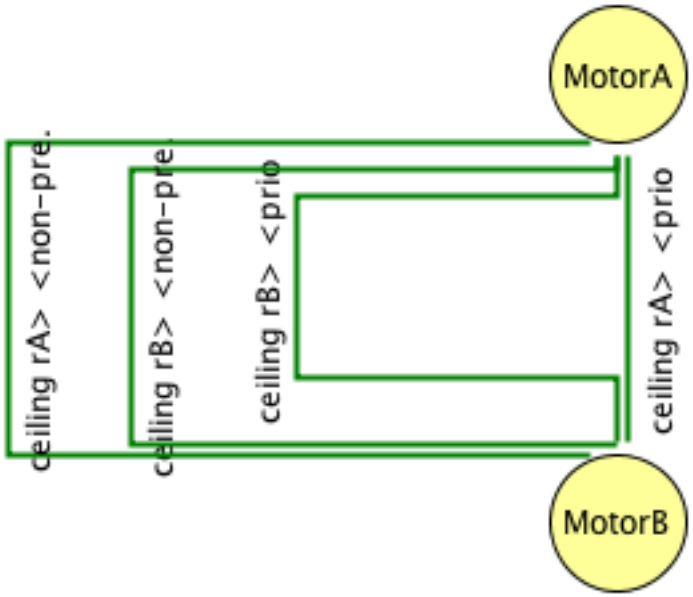
Phase II - Reduktion



Relevante Speicherzugriffe MotorA

Befehlsadr.		Datenadr.
0x400021c8	R	0x400060b8
0x400021cc	R	0x400060bc
0x400021d0	R	0x400060c8
0x400021d4	R	0x400060cc
0x400021e4	W	0x400060d0

Phase II - Reduktion



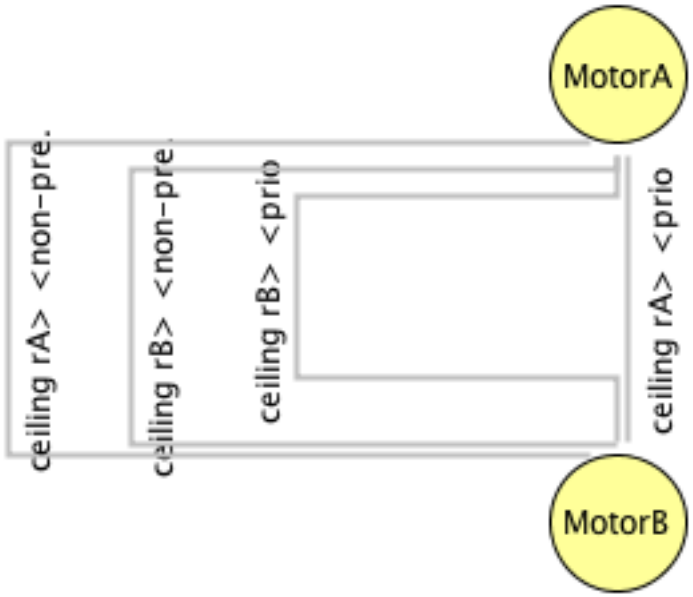
Relevante Speicherzugriffe MotorA

Befehlsadr.		Datenadr.
0x400021c8	R	0x400060b8
0x400021cc	R	0x400060bc
0x400021d0	R	0x400060c8
0x400021d4	R	0x400060cc
0x400021e4	W	0x400060d0

Relevante Speicherzugriffe MotorB

Befehlsadr.		Datenadr.
0x4000222c	R	0x400034a0
0x40002238	R	0x400034a4
0x40002240	R	0x400060b8
0x40002244	R	0x400060bc
0x40002248	R	0x400060c8
0x4000224c	R	0x400060cc
0x40002254	R	0x400034a4
0x40002260	R	0x400034a0
0x40002274	W	0x400060d4

Phase II - Reduktion



Überlappende Zugriffe nur lesend
➔
WA-Einschränkungen für Schutz Speicherzugriffe nicht notwendig!

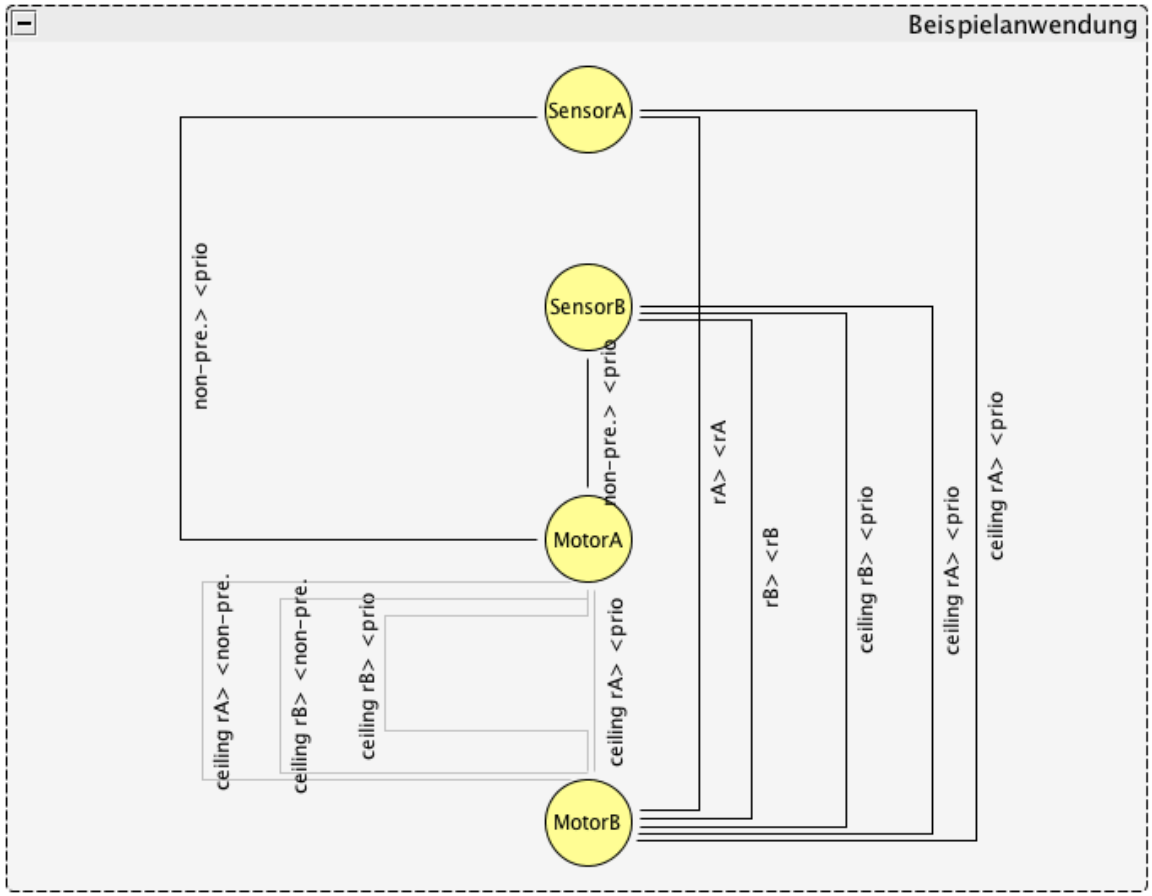
Relevante Speicherzugriffe MotorA

Befehlsadr.		Datenadr.
0x400021c8	R	0x400060b8
0x400021cc	R	0x400060bc
0x400021d0	R	0x400060c8
0x400021d4	R	0x400060cc
0x400021e4	W	0x400060d0

Relevante Speicherzugriffe MotorB

Befehlsadr.		Datenadr.
0x4000222c	R	0x400034a0
0x40002238	R	0x400034a4
0x40002240	R	0x400060b8
0x40002244	R	0x400060bc
0x40002248	R	0x400060c8
0x4000224c	R	0x400060cc
0x40002254	R	0x400034a4
0x40002260	R	0x400034a0
0x40002274	W	0x400060d4

Phase II - Reduktion

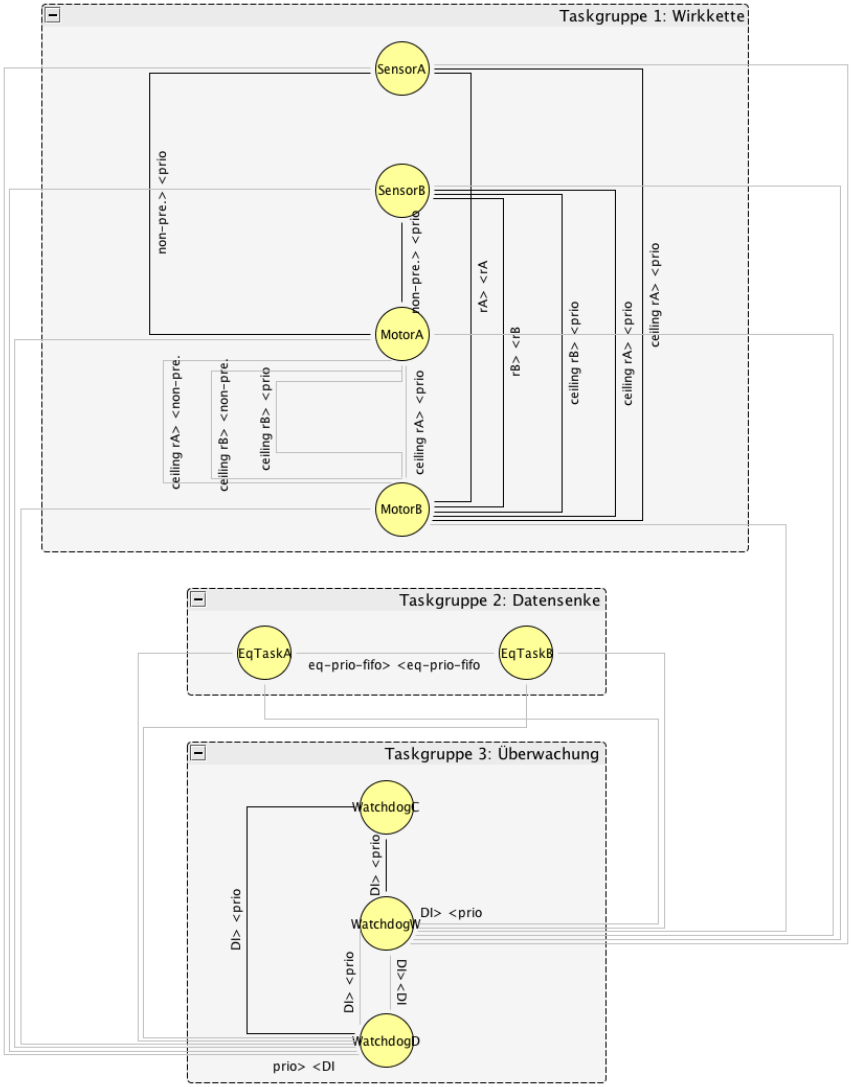


Wechselseitige Ausschlüsse:

Vor Reduktion: 11
Nach Reduktion: 7

Größeres Beispiel

FACHHOCHSCHULE TRIER
Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences
Informatik



Wechselseitige Ausschlüsse:

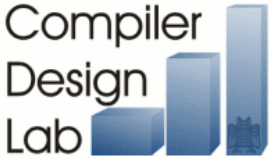
Vor Reduktion: 31
Nach Reduktion: 9

Fazit

- Migration auf Multicore führt zu neuen Race Conditions
- Abhilfe:
  - Finden aller wechselseitigen Ausschlüsse
  - Bewerten aller wechselseitigen Ausschlüsse
- Lösung:
  - Automatisierung durch zweistufigen Analyseansatz

Kooperationspartner:

AbsInt GmbH



Prof. Reinhard Wilhelm, Universität des Saarlandes