automotive engineering iav

Cache-Kohärenz für embedded Multicore-Mikrocontroller mit harter Echtzeitanforderung

Echtzeit 2019
Philipp Jungklass, Boppard am Rhein, November 2019

Motivation

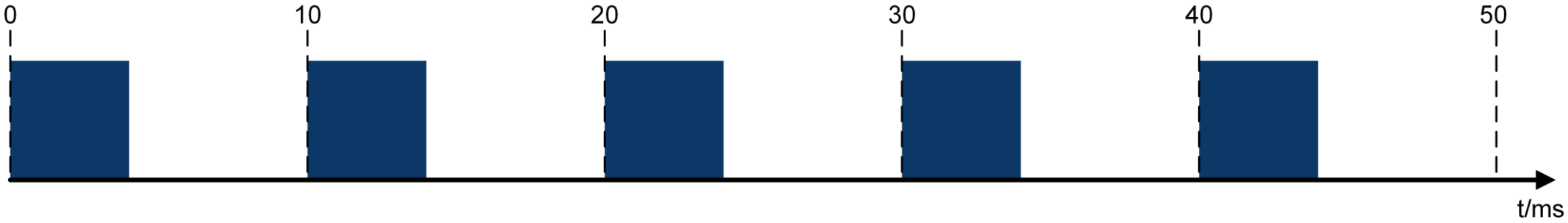
- **Multicore-Mikrocontroller werden immer beliebter in Anwendungen mit harten Echtzeitanforderungen**
  - Höhere Performance bei relativ niedrigen Kosten
  - Höheres Integrationsniveau
- **Embedded Multicore und harte Echtzeitanforderungen sind schwer kombinierbar**
  - Jede Multicore Architektur hat geteilte Ressourcen → Flaschenhals
- **Intercore-Kommunikation → Zugriff auf geteilten Speicher**
- **Konkurrierende Zugriffe → potenzielle Wartezyklen**
- **Echtzeitsystem → maximale Anzahl an Wartezyklen**
- **Pessimistische Worst-Case-Execution-Time**
  - Reduzierung der nutzbaren Rechenzeit
  - Potentiell höhere Unkosten

1. Stand der Technik

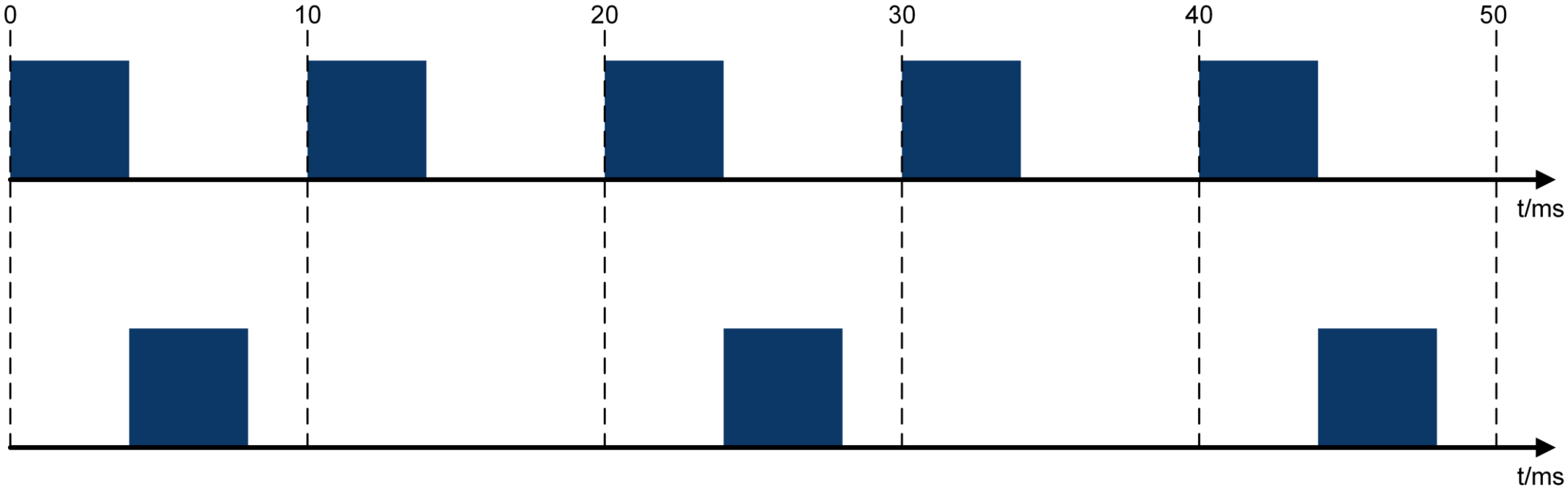
1. Stand der Technik

1. Stand der Technik

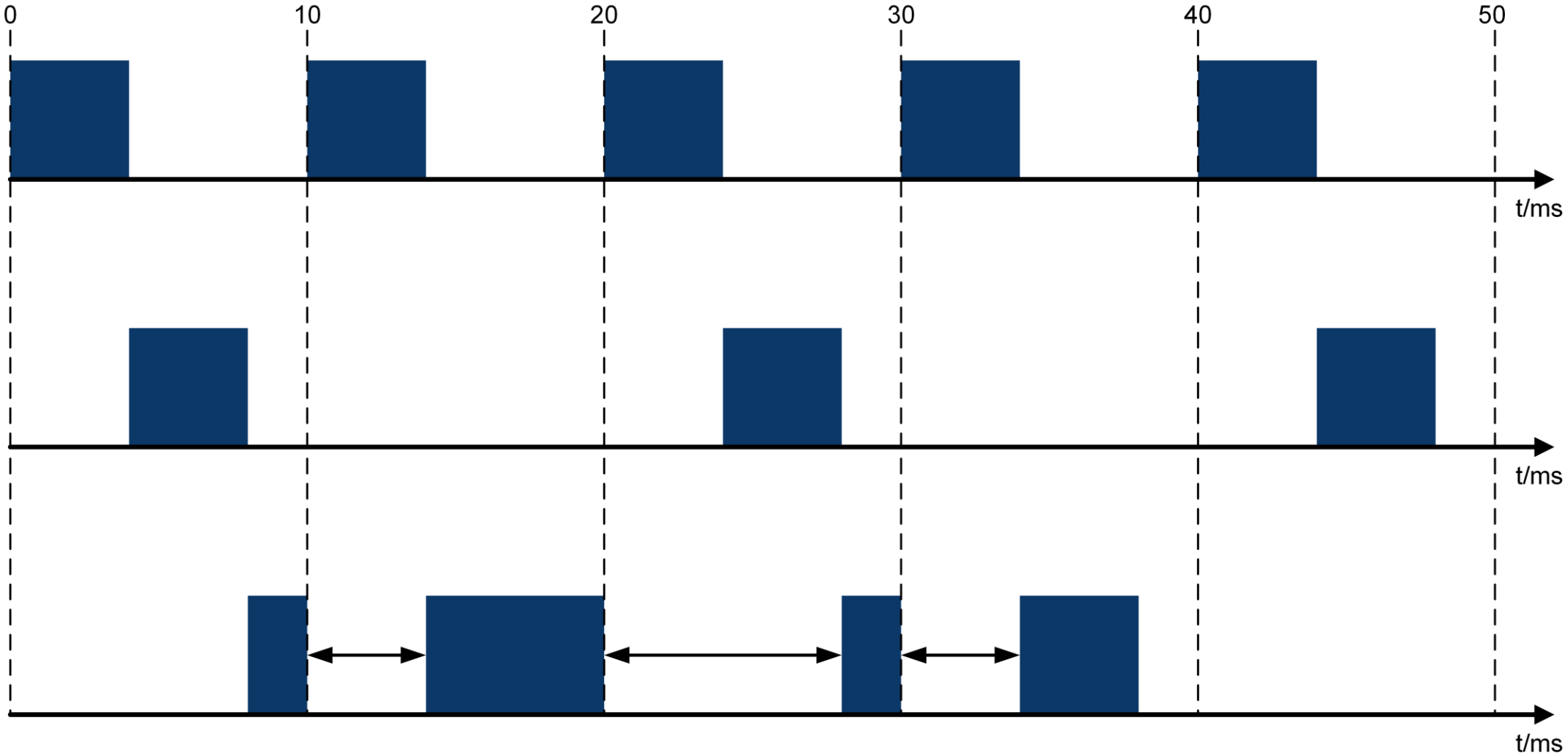
1. Stand der Technik



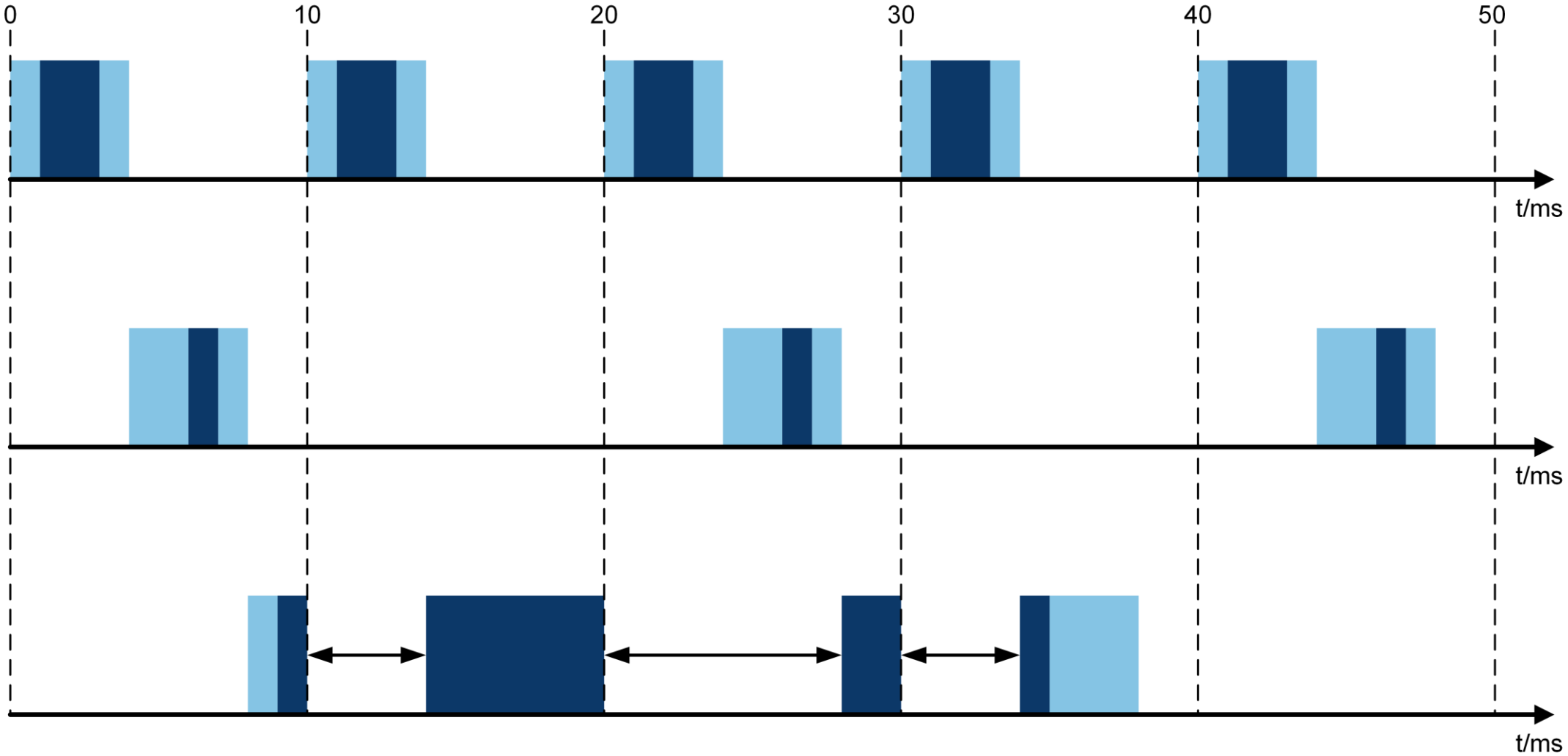
1. Stand der Technik



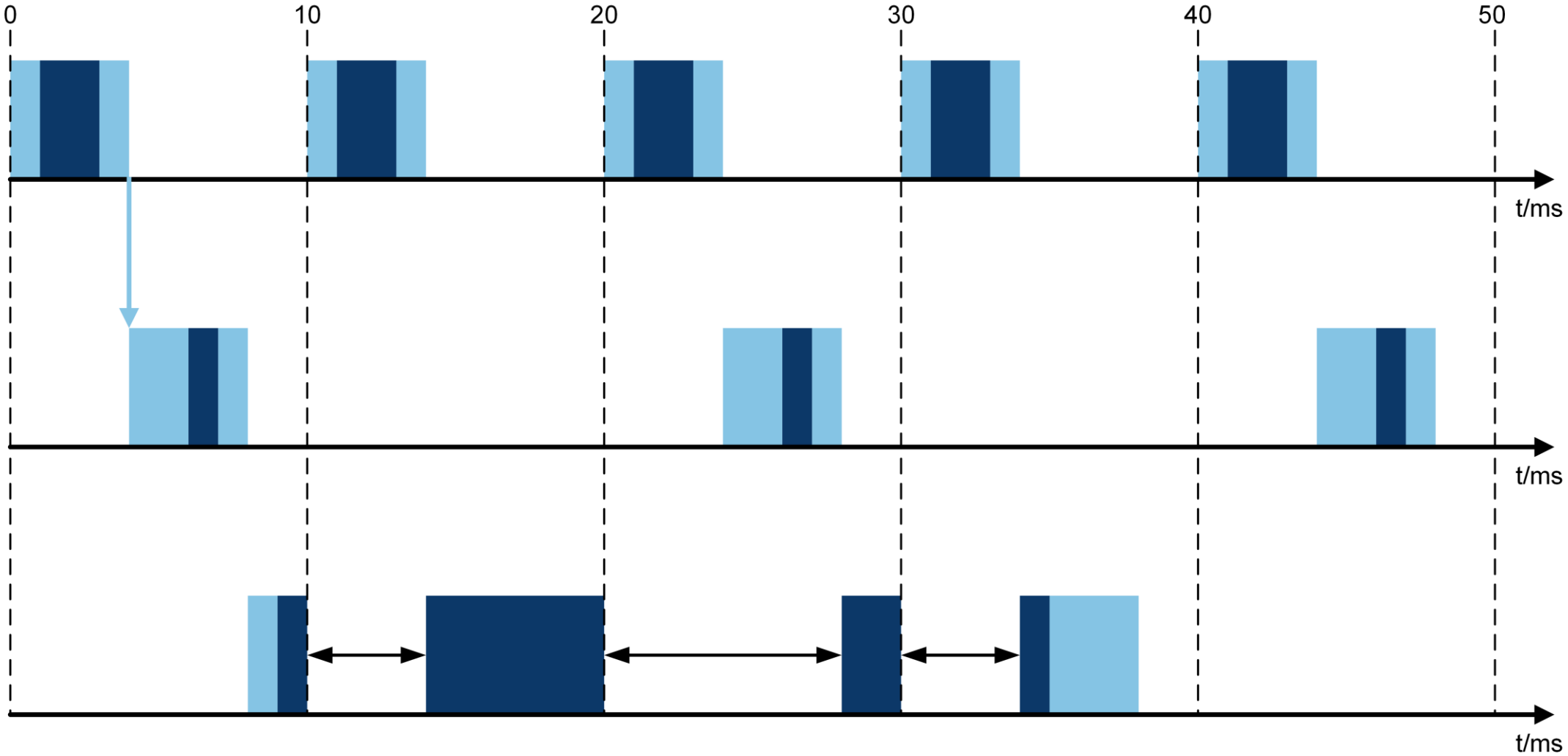
1. Stand der Technik



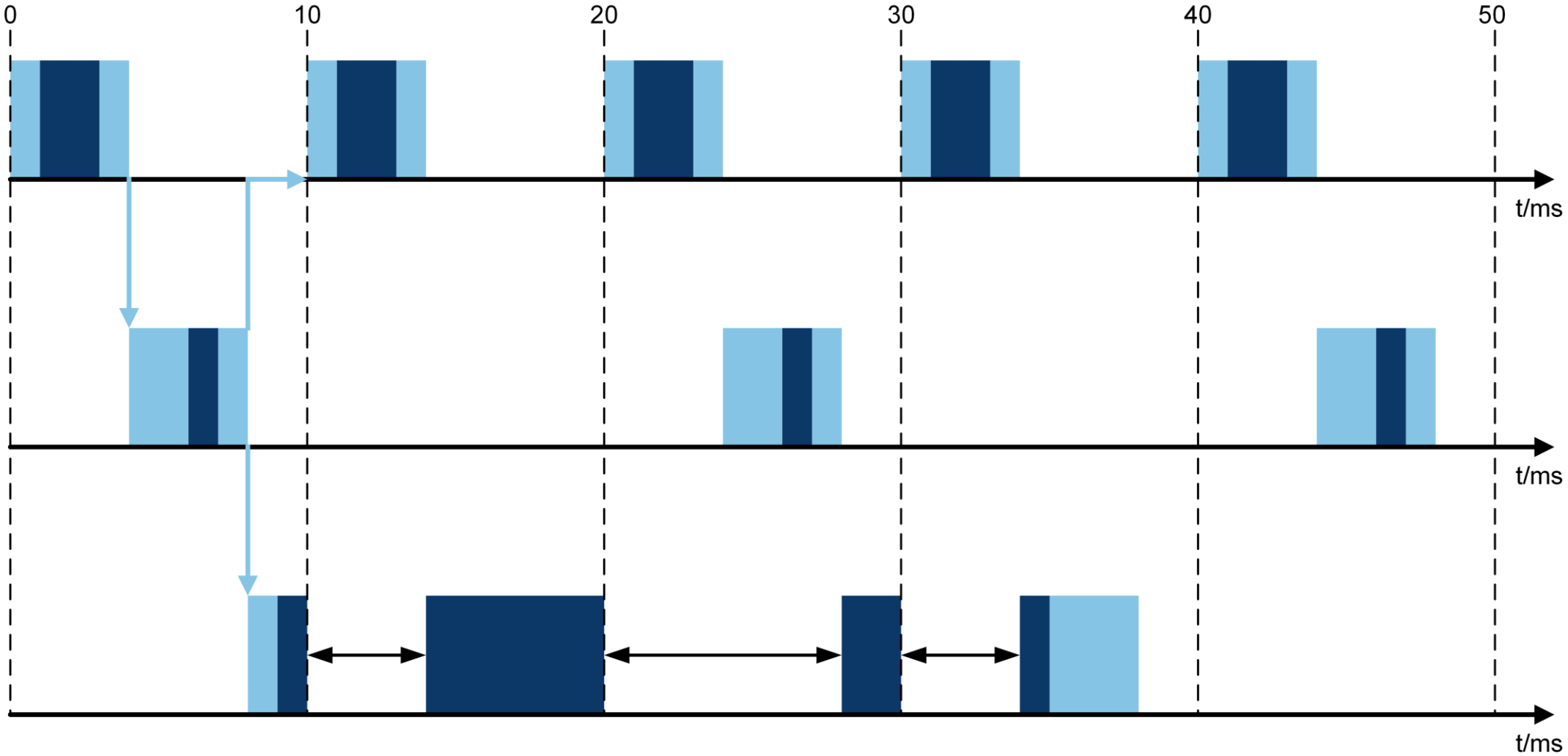
1. Stand der Technik



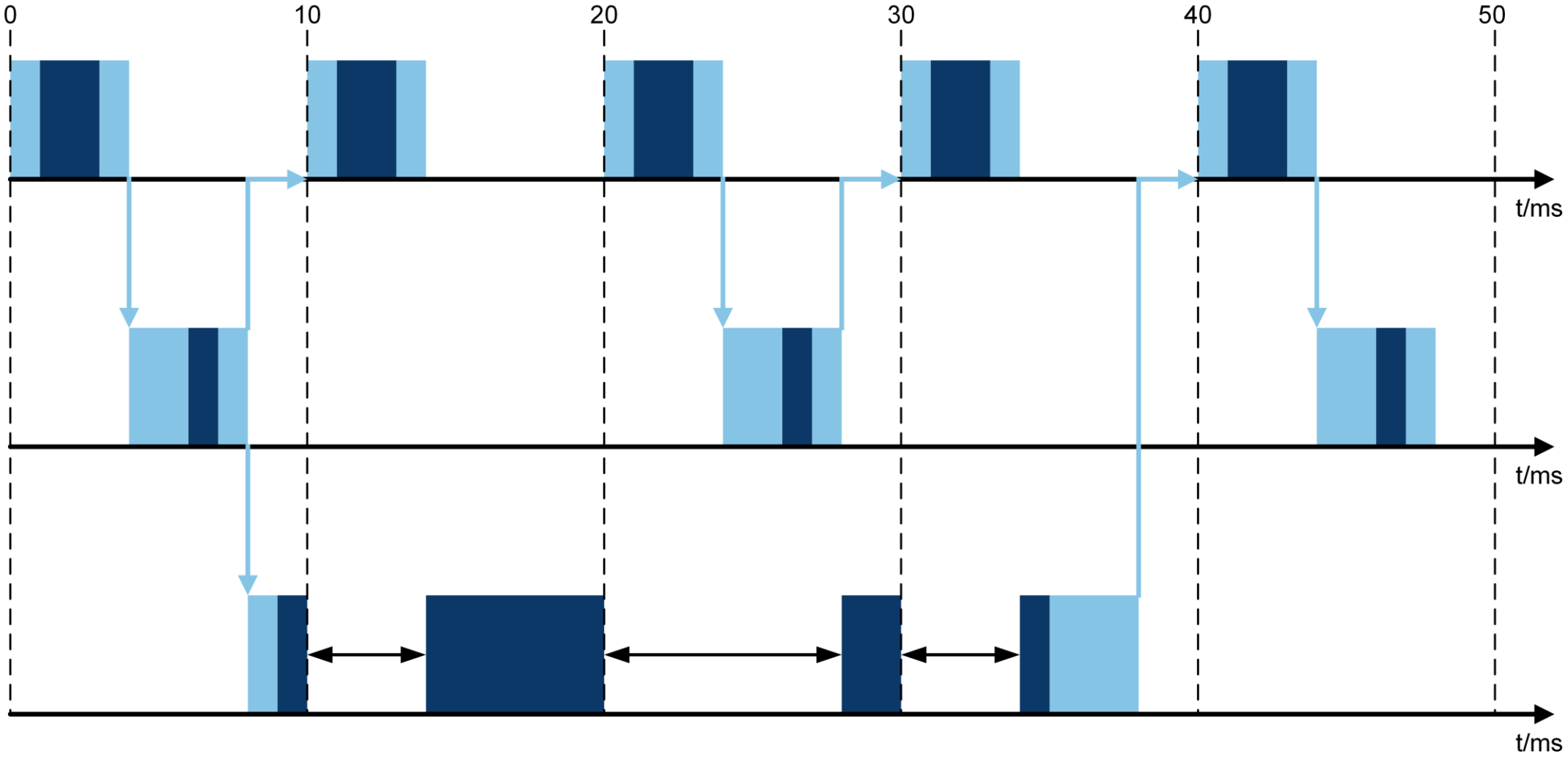
1. Stand der Technik



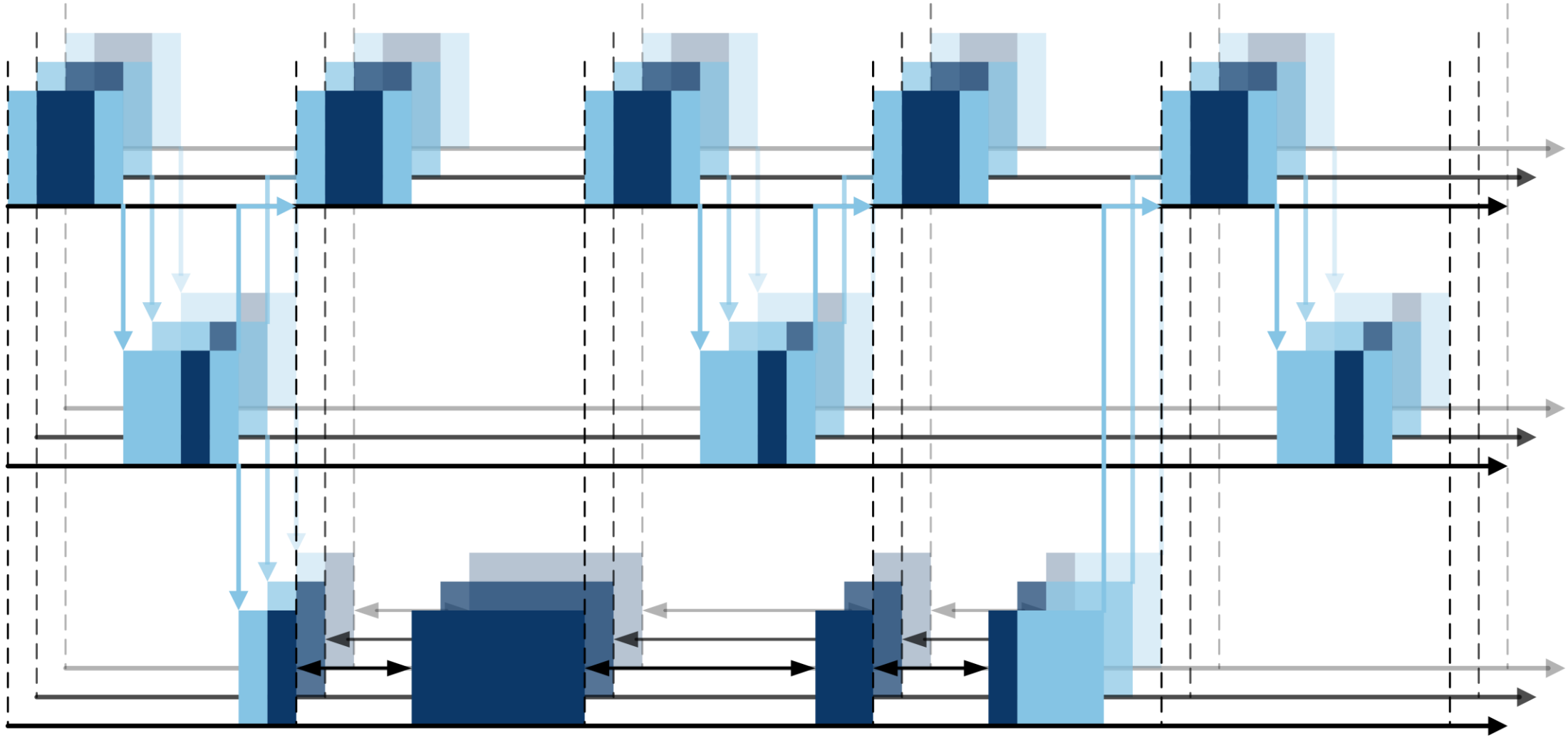
1. Stand der Technik



1. Stand der Technik



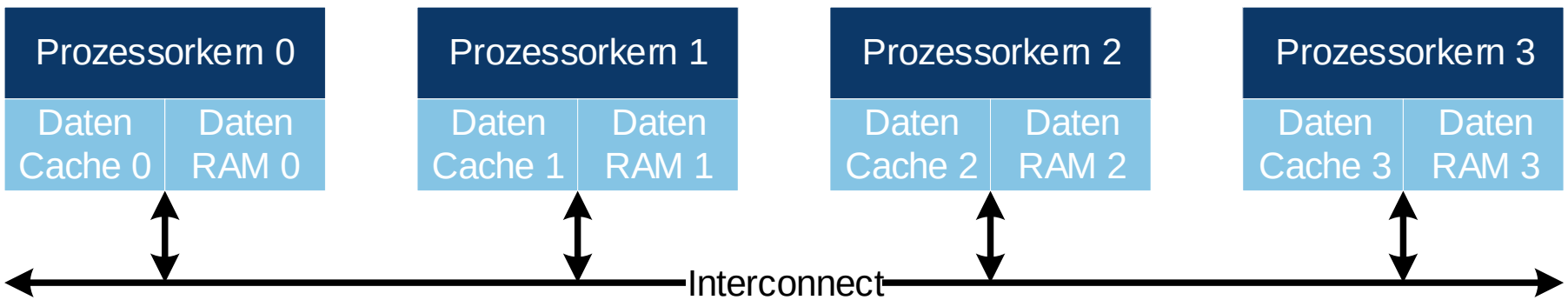
1. Stand der Technik



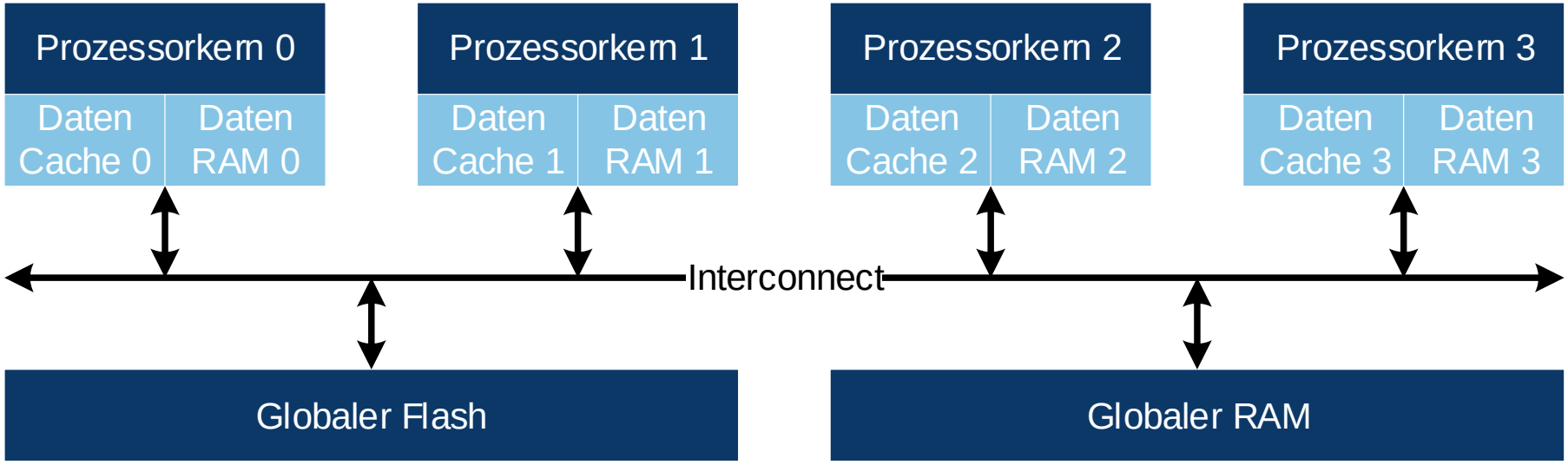
1. Stand der Technik

1. Stand der Technik

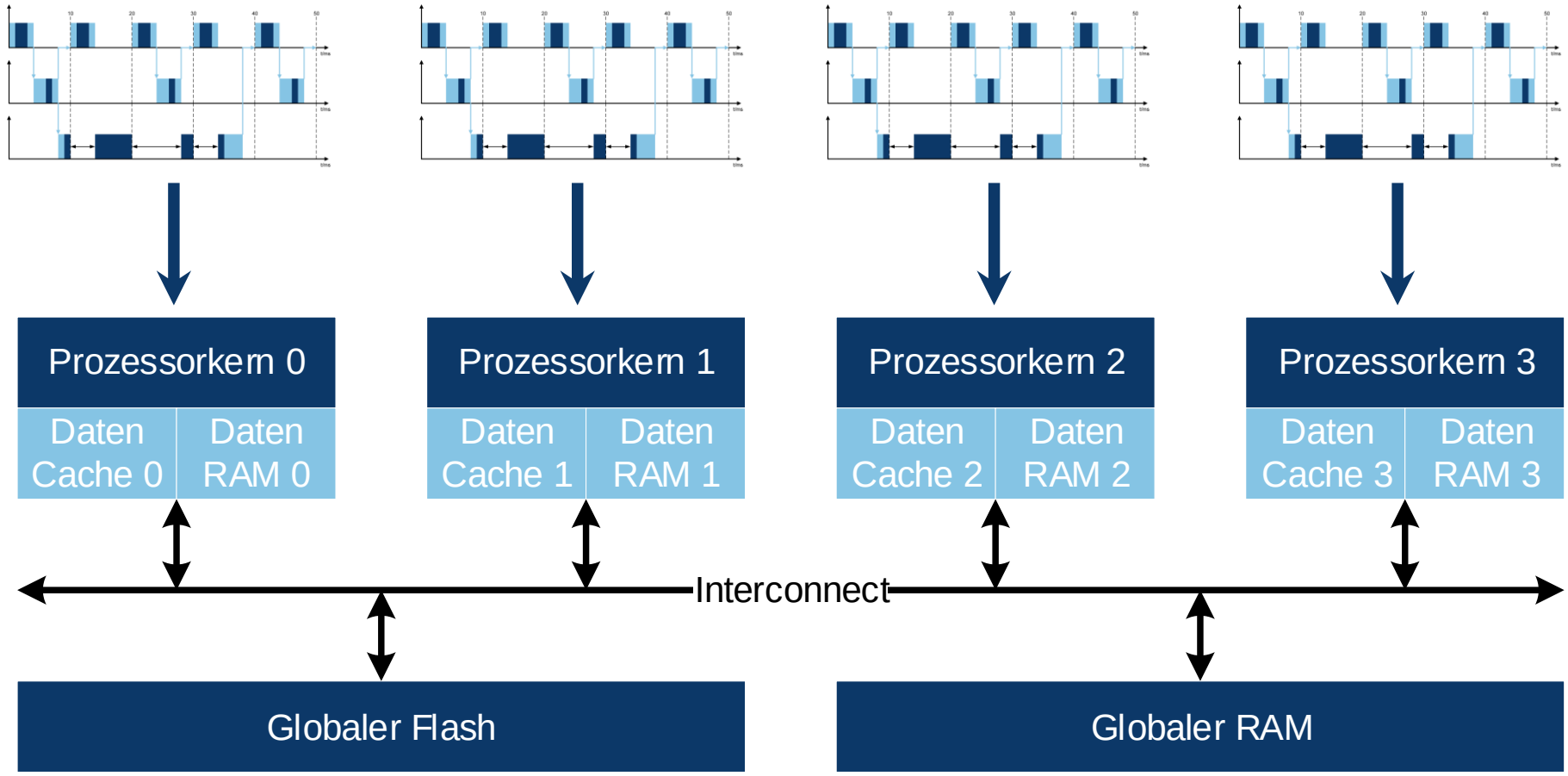
1. Stand der Technik



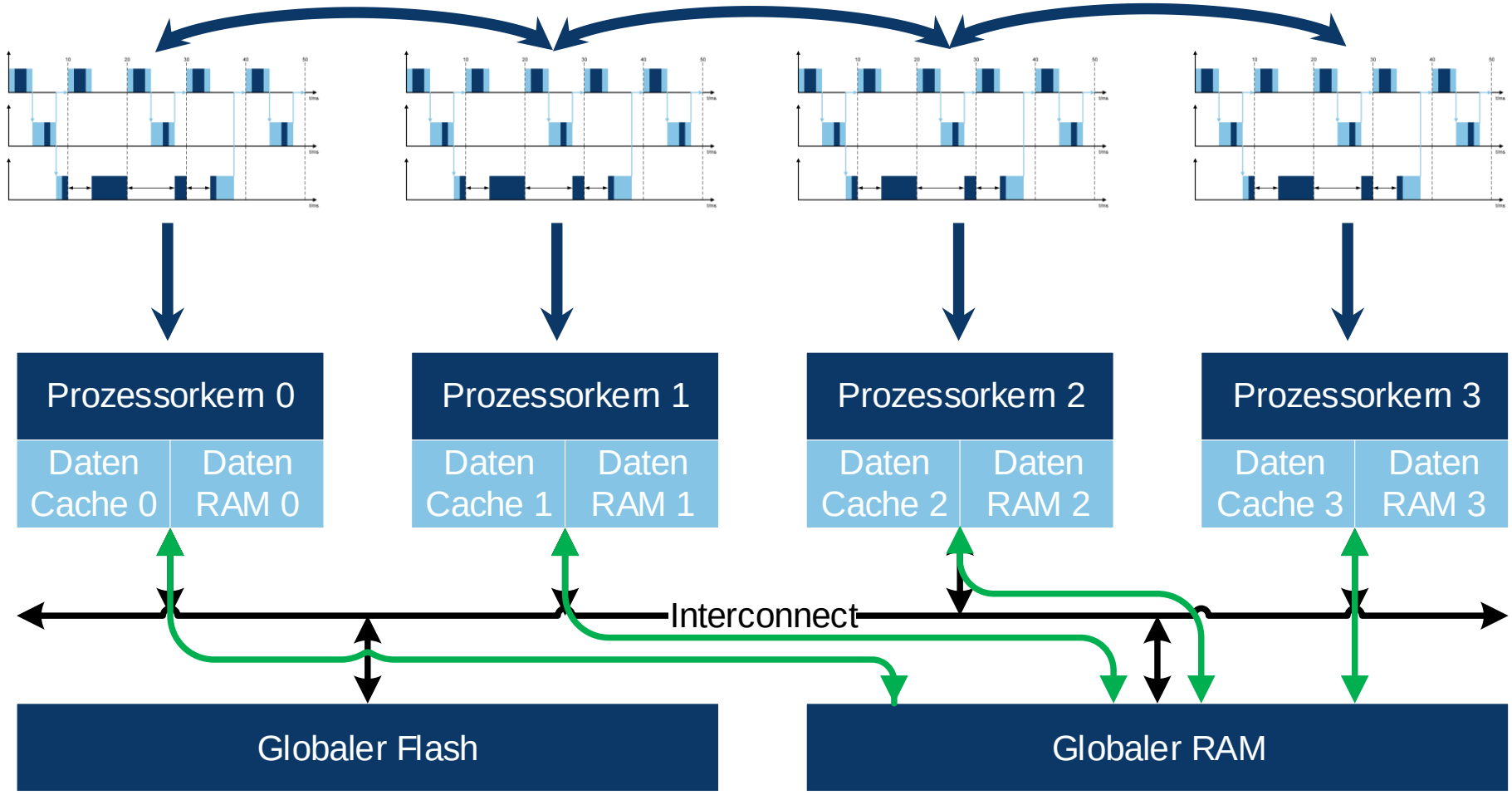
1. Stand der Technik



1. Stand der Technik



1. Stand der Technik



- **Nutzung des globalen RAMs zur Intercore-Kommunikation**

- **Vorteile:**
  - Einheitliche Zugriffszeit für alle Prozessorkerne
  - Einfachere Speicherverwaltung
  - Exklusive Nutzung der lokalen RAM-Speicher durch den jeweiligen Prozessorkern

- **Nachteile:**
  - Konkurrierender Zugriff durch alle Prozessorkerne
  - RAM-Speicher mit langsamster Zugriffszeit

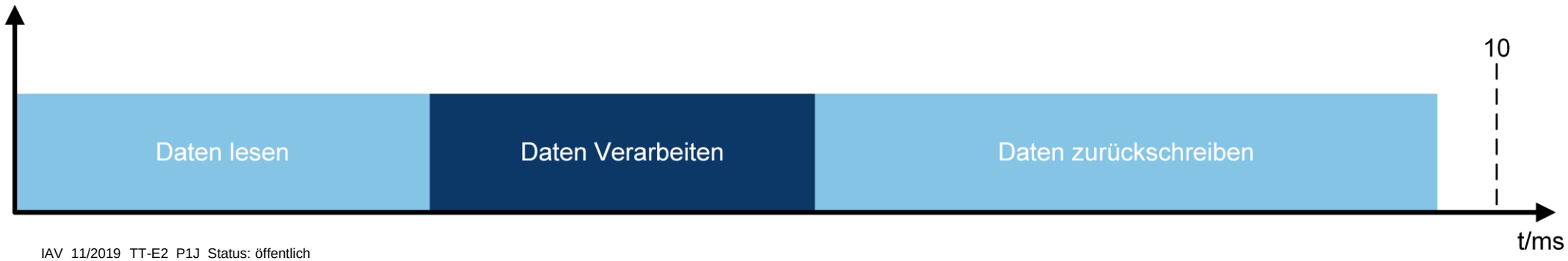
1. Stand der Technik

- **Nutzung des globalen RAMs zur Intercore-Kommunikation**

- **Vorteile:**
  - Einheitliche Zugriffszeit für alle Prozessorkerne
  - Einfachere Speicherverwaltung
  - Exklusive Nutzung der lokalen RAM-Speicher durch den jeweiligen Prozessorkern

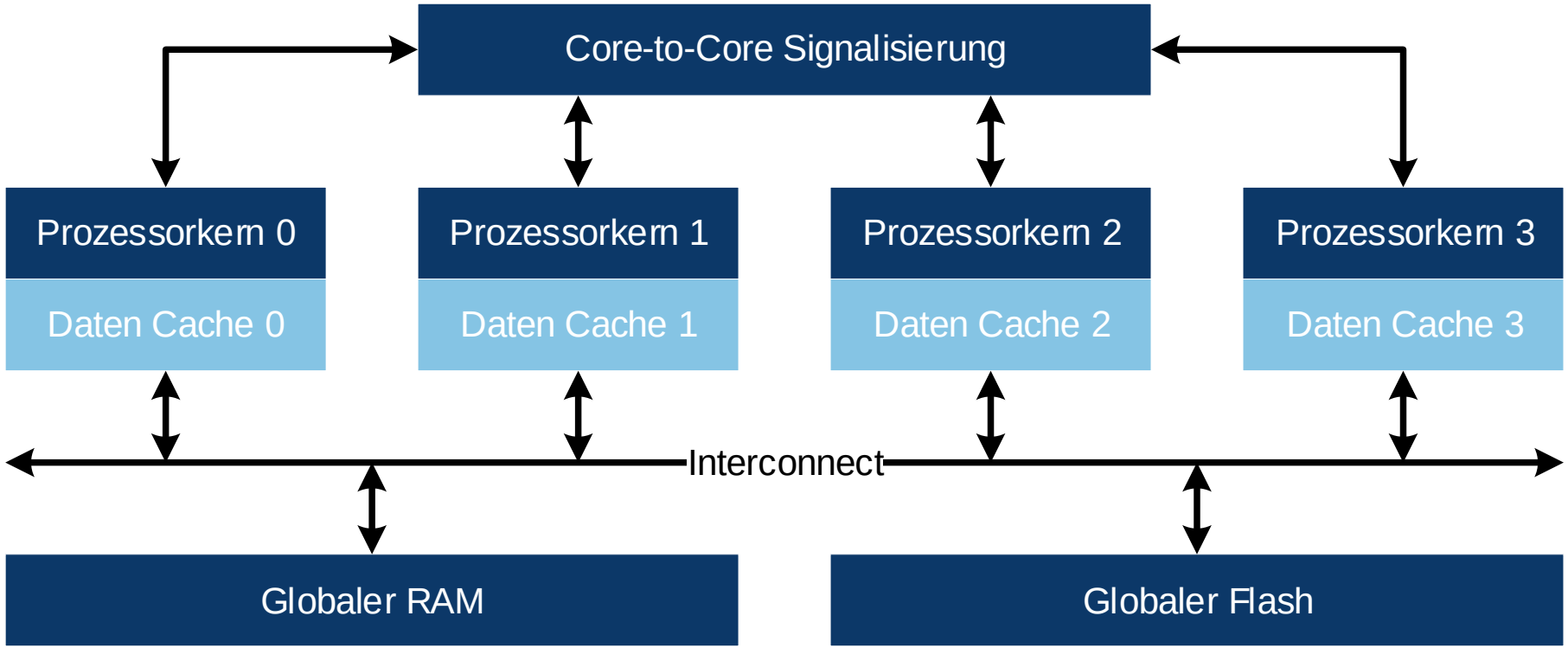
- **Nachteile:**
  - Konkurrierender Zugriff durch alle Prozessorkerne
  - RAM-Speicher mit langsamster Zugriffszeit

- **Zugriffszeiten werden maximal → Pessimistische Worst-Case-Execution-Time**



2. Konzept

- **Grundkonzept: Nutzung der lokalen Caches zur Intercore-Kommunikation**
  - Voraussetzung: echtzeitfähiges Cache-Kohärenzprotokoll



2. Konzept

- **Kategorisierung der Daten nach der Kritikalität:**
  - Kategorie 1: Zeitunkritisch
  - Kategorie 2: Zeitneutral
  - Kategorie 3: Zeitkritisch

2. Konzept

- **Kategorisierung der Daten nach der Kritikalität:**
  - Kategorie 1: Zeitunkritisch
  - Kategorie 2: Zeitneutral
  - Kategorie 3: Zeitkritisch

- **Kategorie 1:** Aktualisierung des Datums: Nutzung des veralteten Datums im Cache

- **Kategorie 2:** Aktualisierung des Datums: Nutzung des aktualisiertem Datums im globalen RAM

- **Kategorie 3:** Aktualisierung des Datums: Aktualisierung der betroffenen Cacheline

2. Konzept

- **Kategorisierung der Daten nach der Kritikalität:**
  - Kategorie 1: Zeitunkritisch
  - Kategorie 2: Zeitneutral
  - Kategorie 3: Zeitkritisch

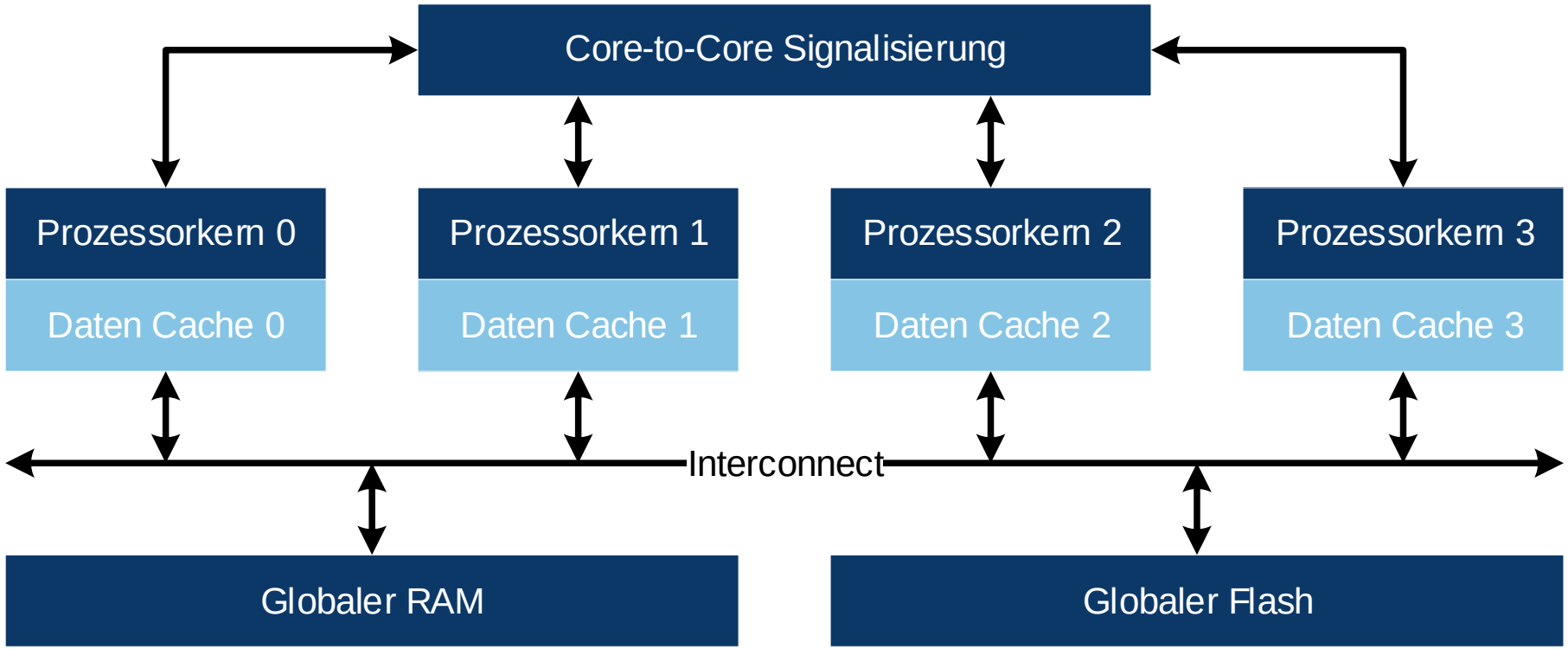
- **Kategorie 1:** Aktualisierung des Datums: Nutzung des veralteten Datums im Cache

- **Kategorie 2:** Aktualisierung des Datums: Nutzung des aktualisiertem Datums im globalen RAM

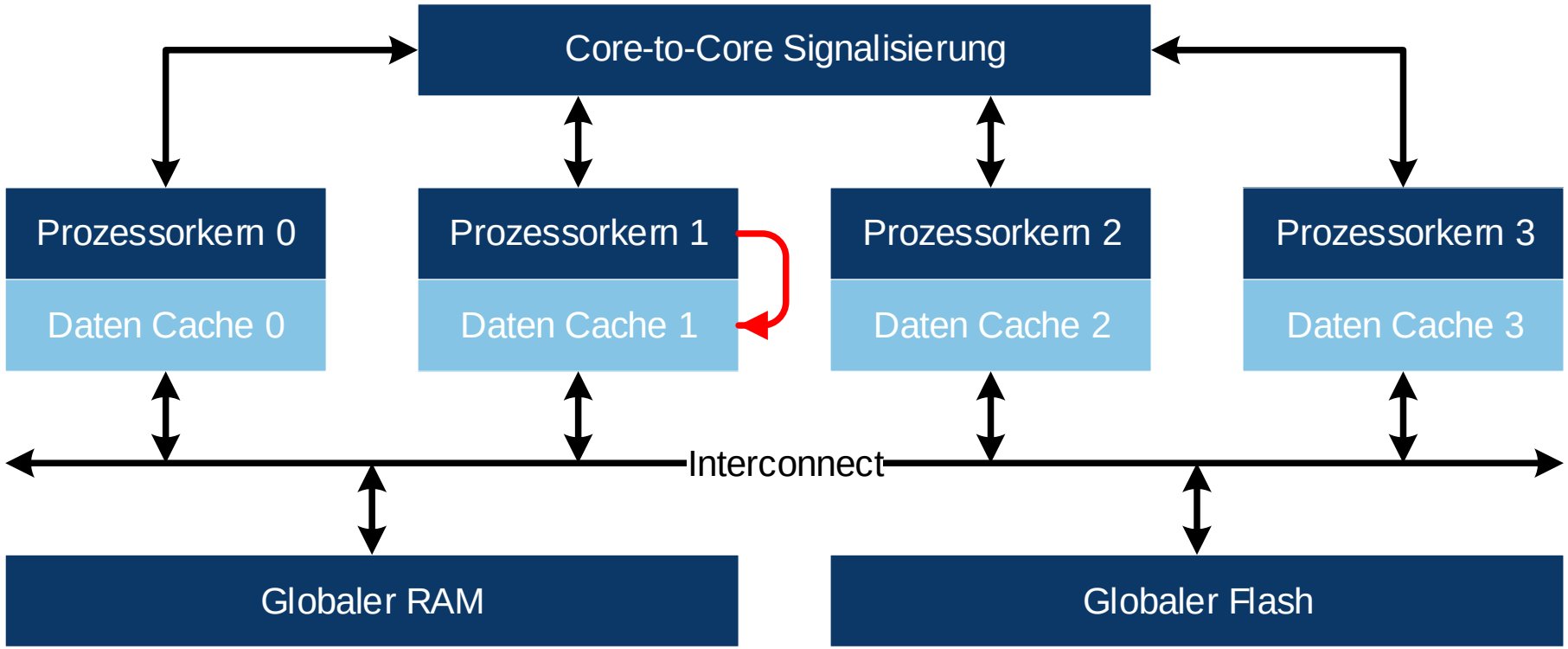
- **Kategorie 3:** Aktualisierung des Datums: Aktualisierung der betroffenen Cacheline

- **Randbedingungen:**
  - 1 produzierender Prozessorkern
  - n konsumierende Prozessorkerne
  - Task-basiertes System

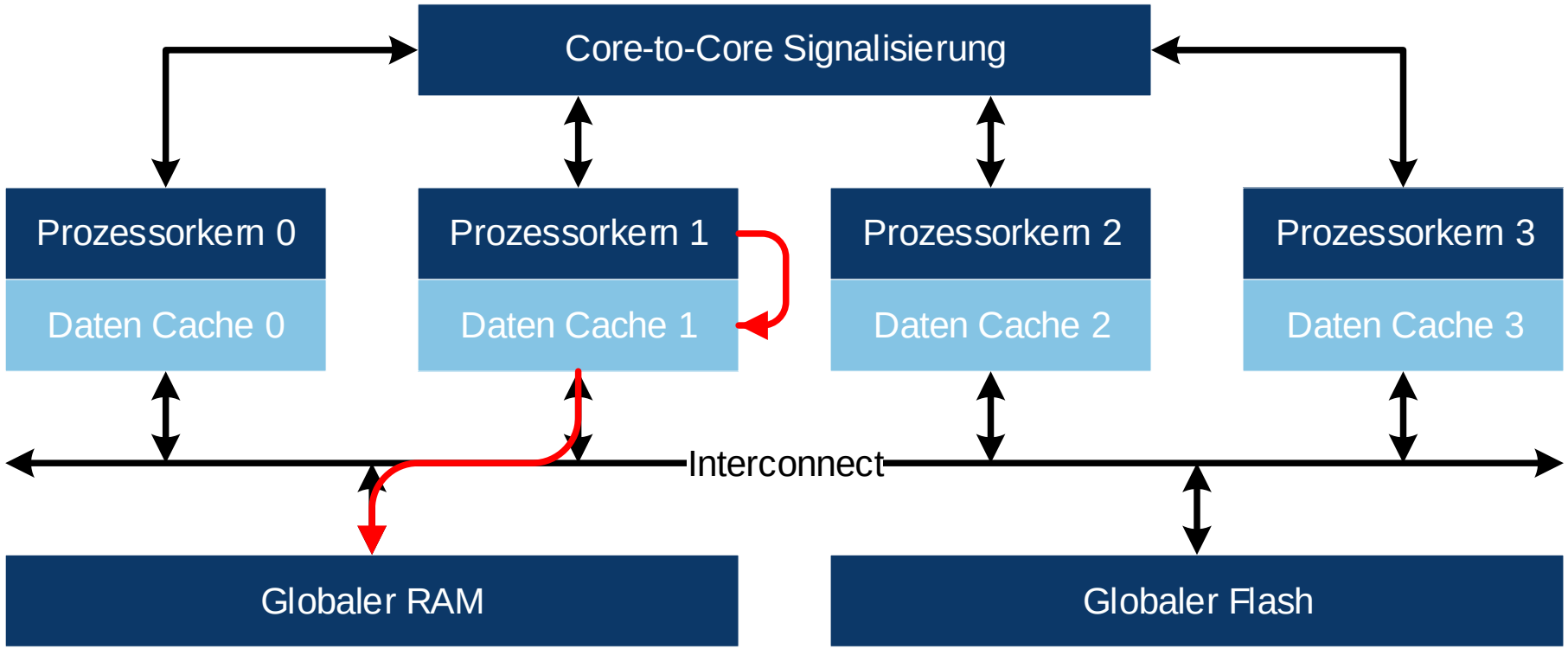
- **Beispiel:**
  - Kategorie 1



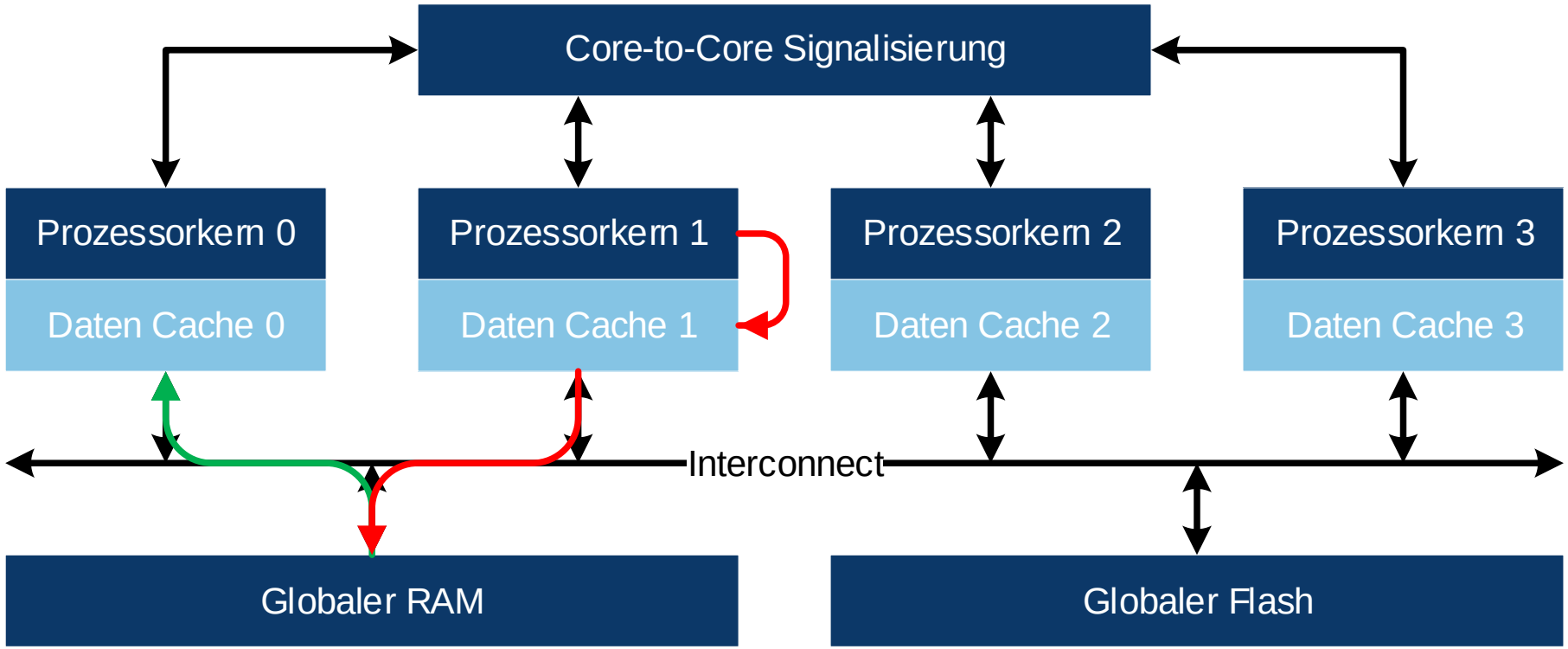
- **Beispiel:**
  - Kategorie 1
  - Produzent: Core 1 (Ende der Task-Ausführung)



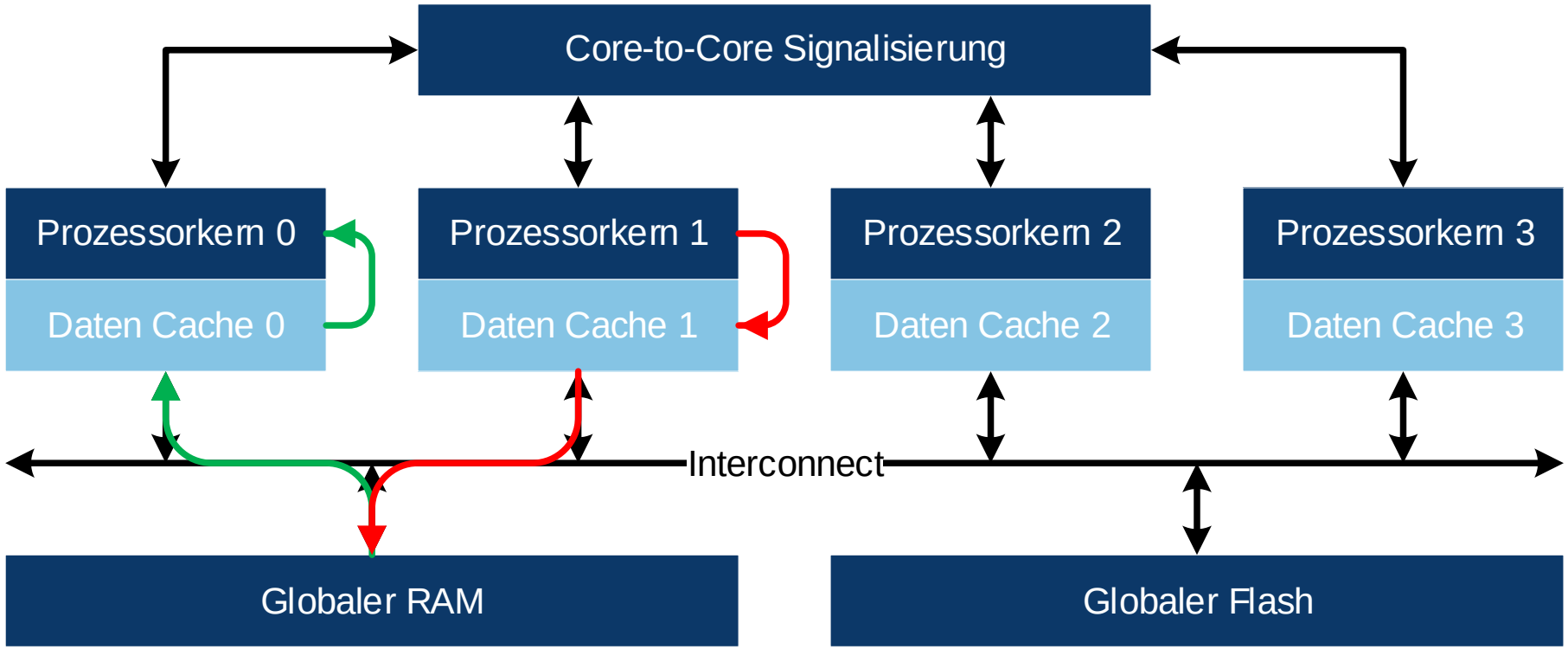
- **Beispiel:**
  - Kategorie 1
  - Produzent: Core 1 (Ende der Task-Ausführung)



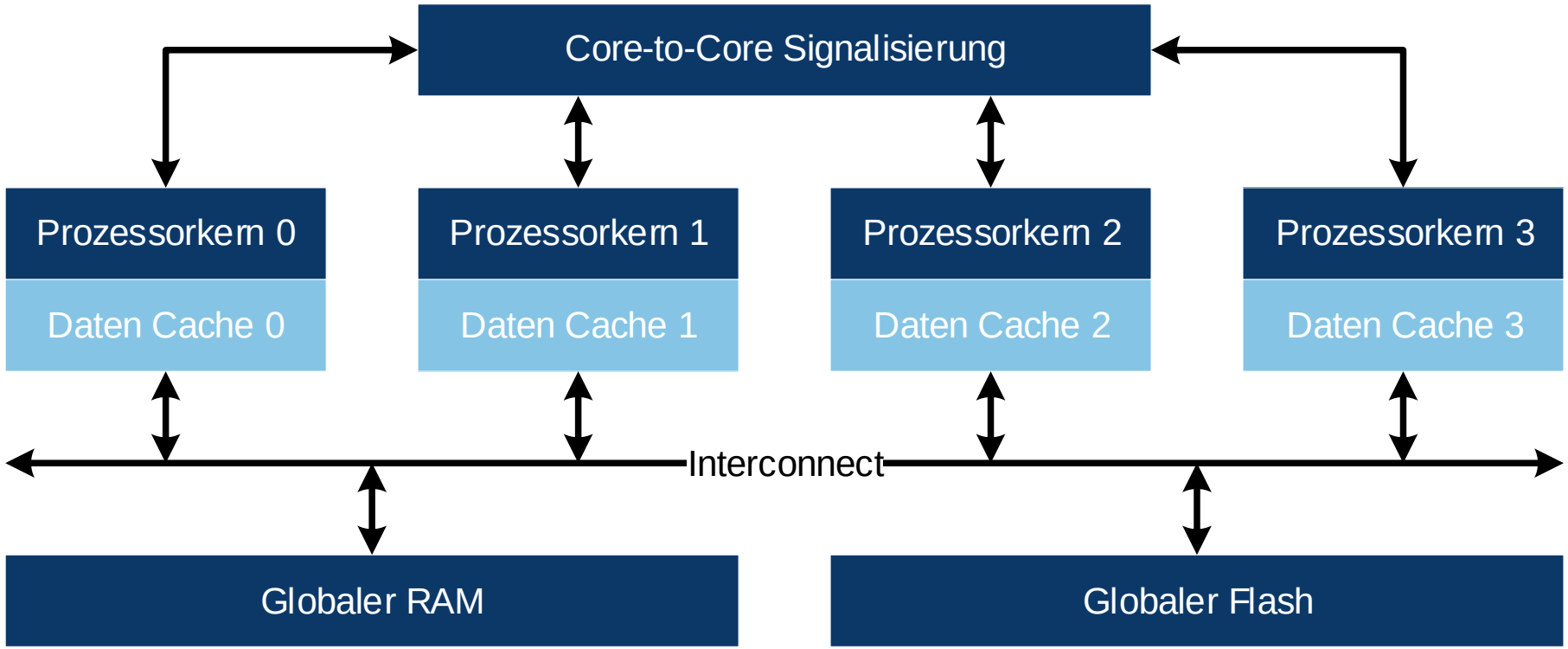
- **Beispiel:**
  - Kategorie 1
  - Produzent: Core 1 (Ende der Task-Ausführung)
  - Konsument: Core 0 (Anfang der Task-Ausführung)



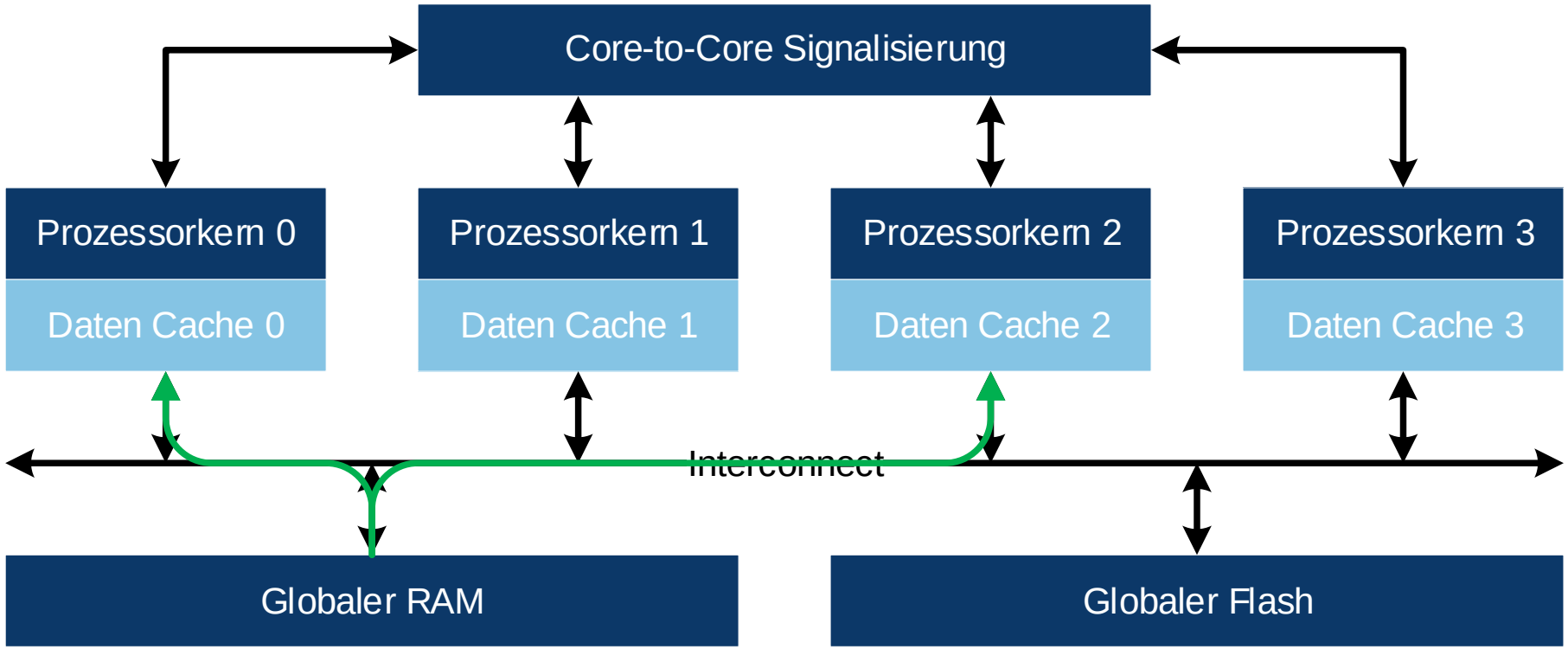
- **Beispiel:**
  - Kategorie 1
  - Produzent: Core 1 (Ende der Task-Ausführung)
  - Konsument: Core 0 (Anfang der Task-Ausführung)



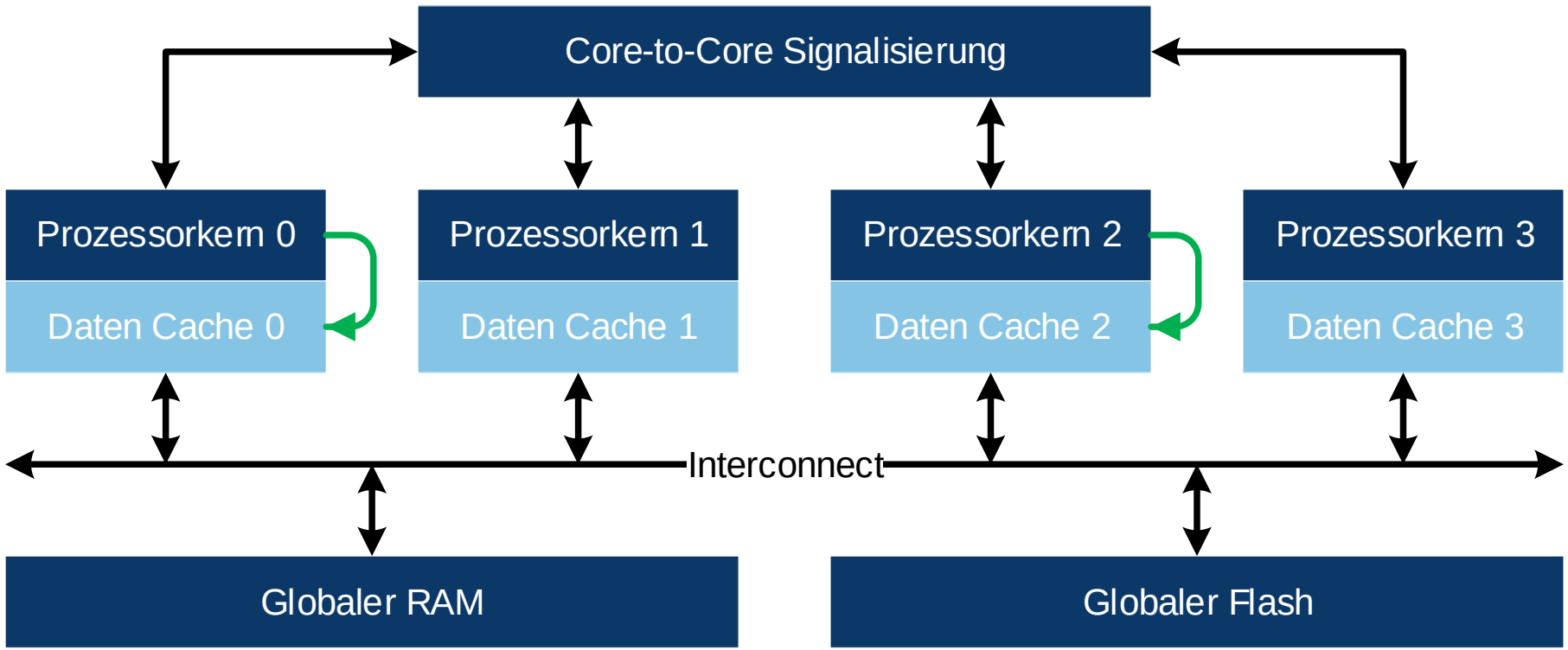
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



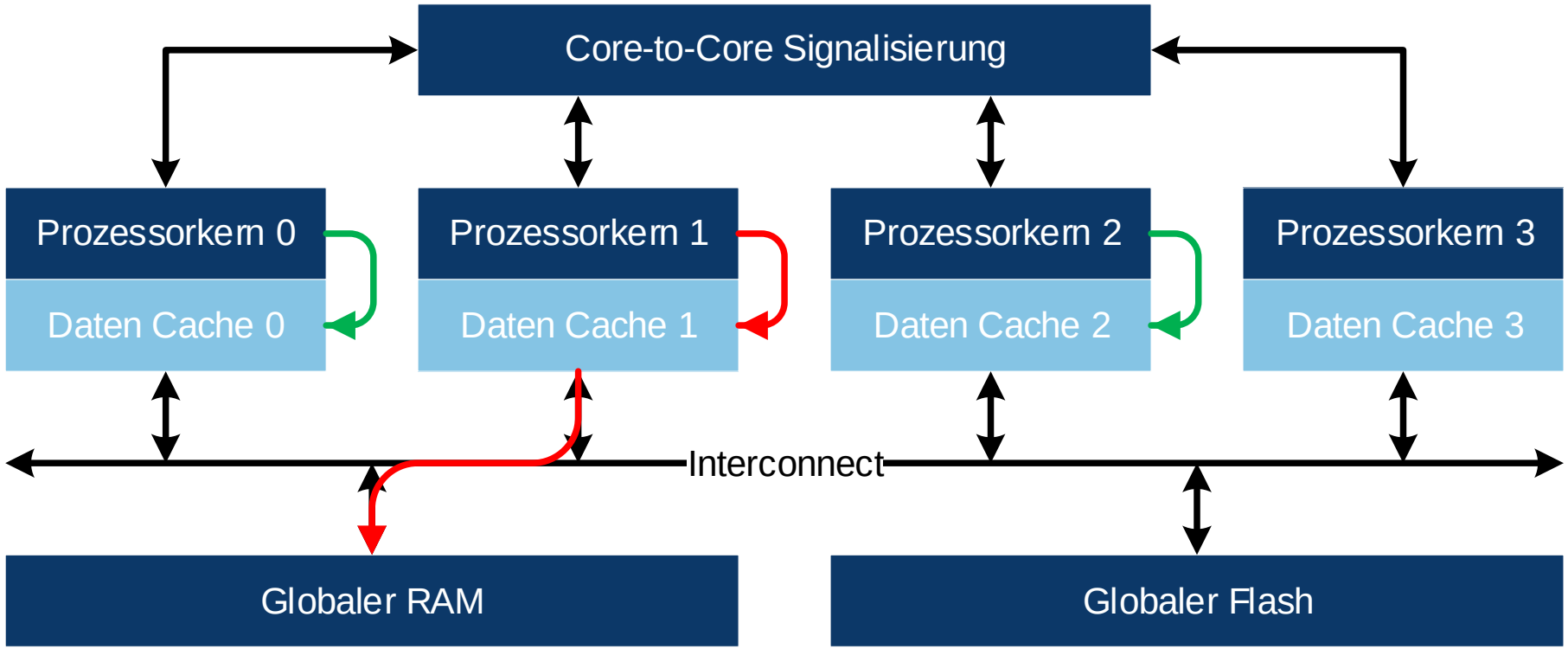
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



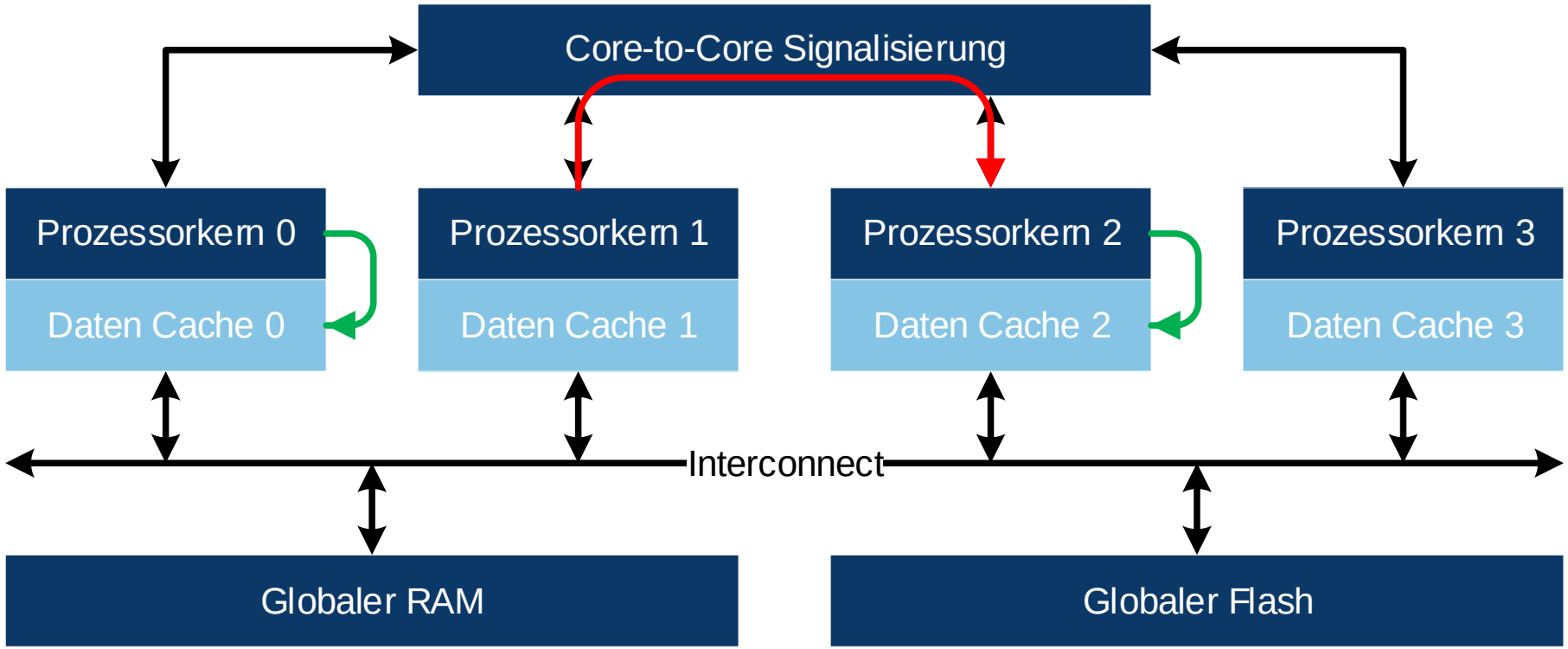
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



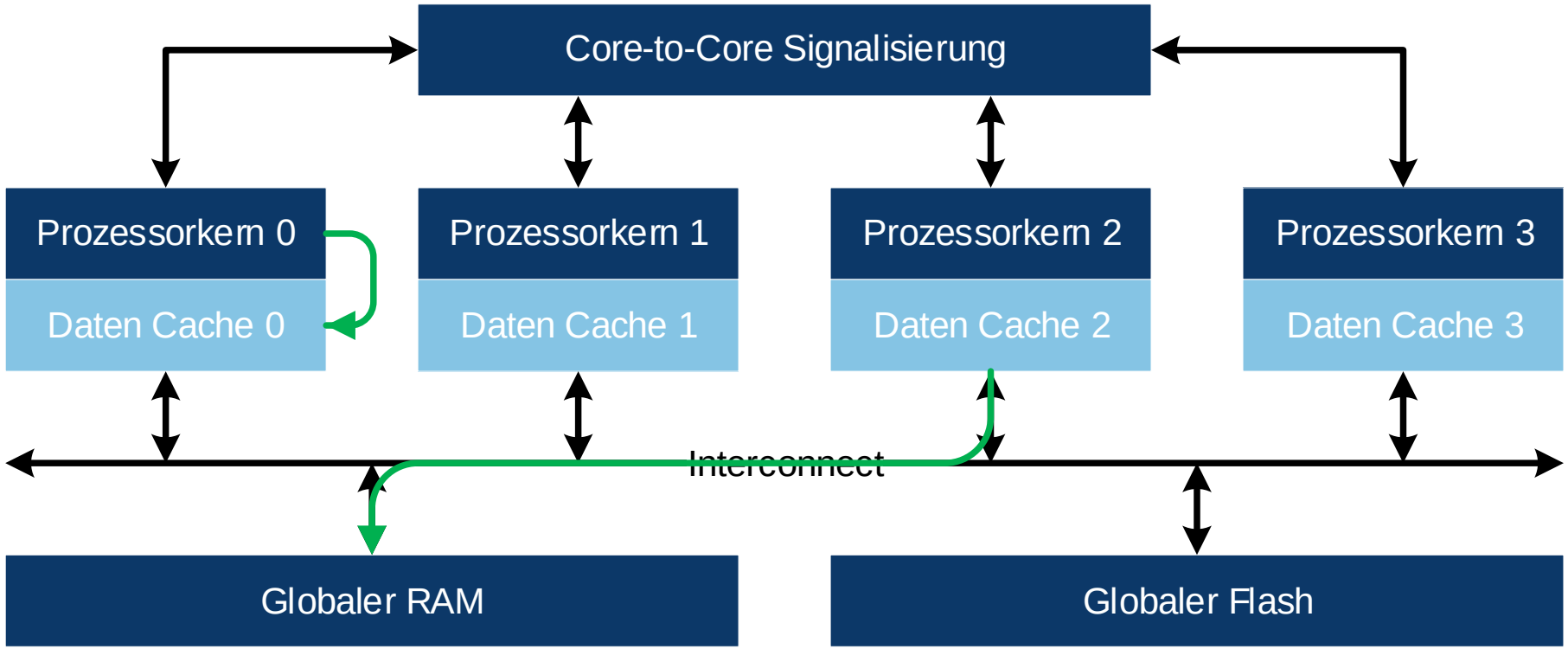
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



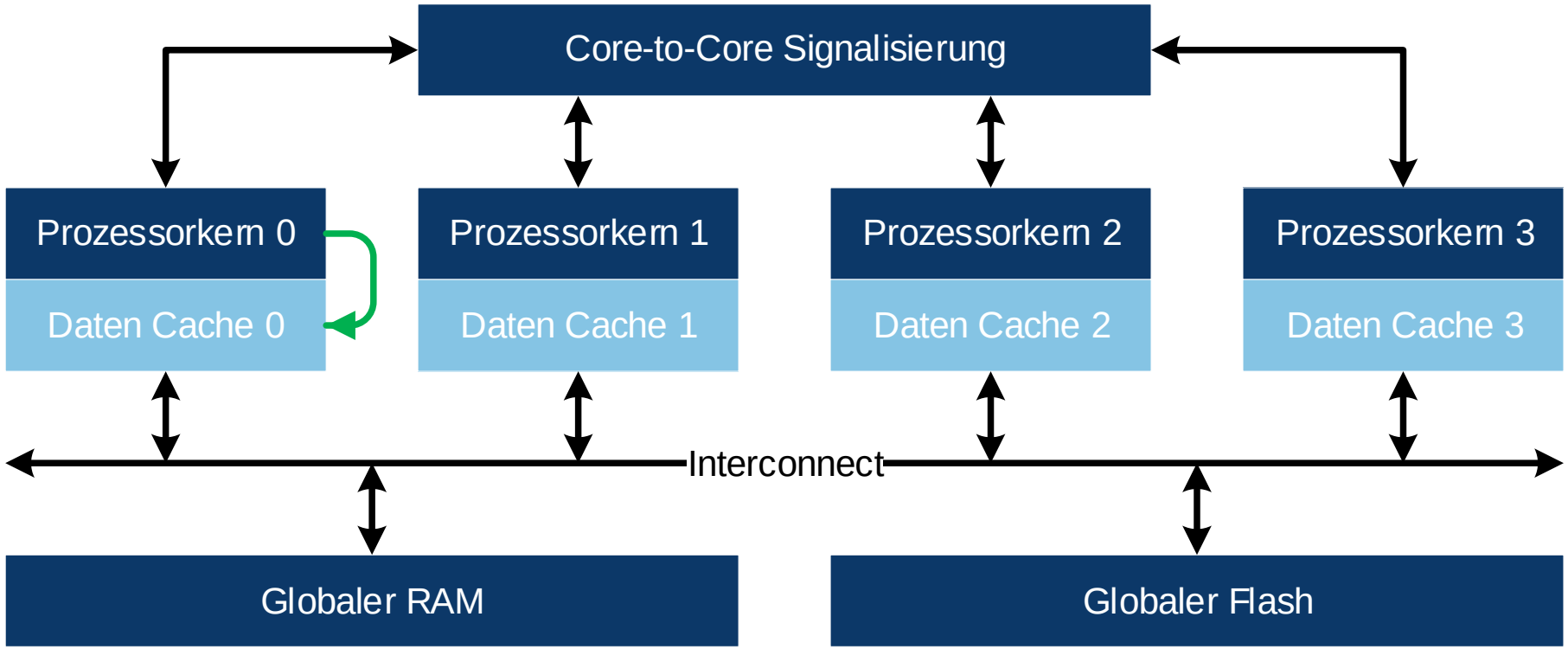
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



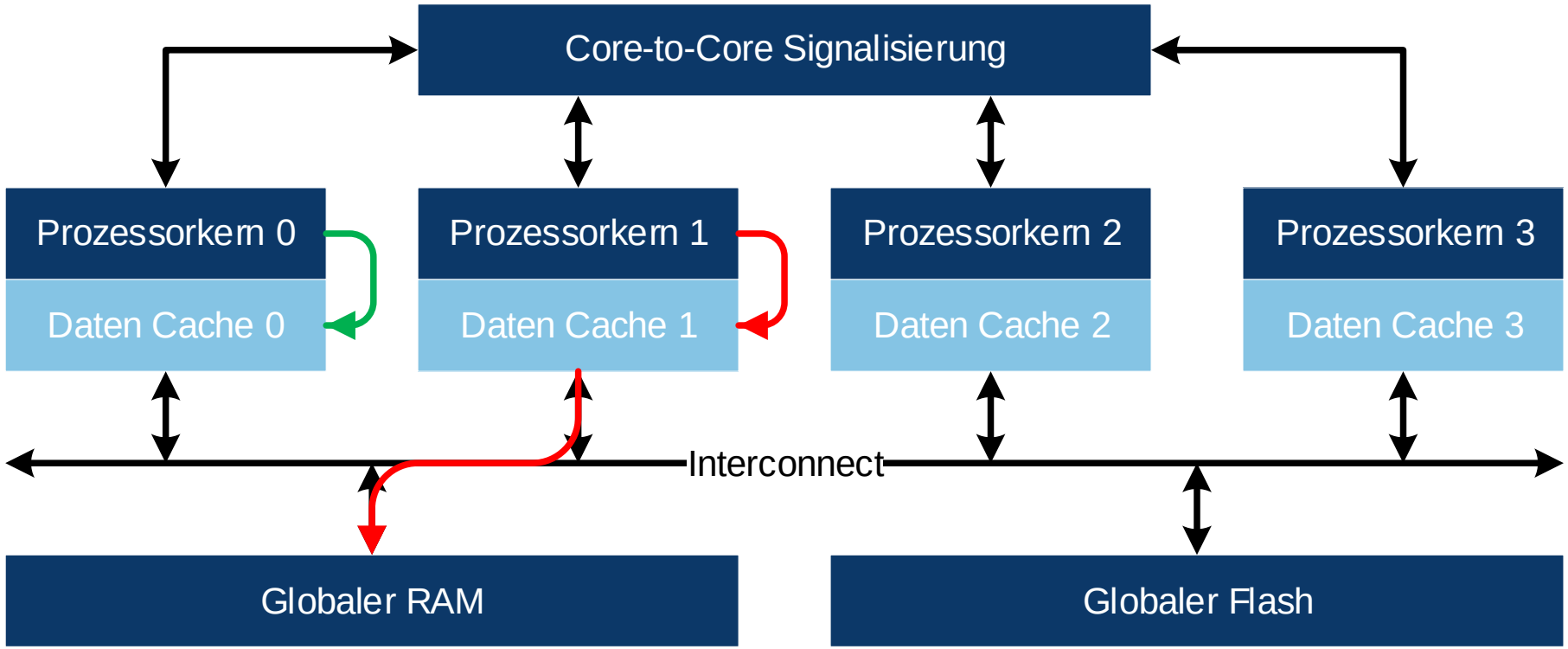
- **Beispiel:**
  - Kategorie 1/2
  - Produzent: Core 1 (Sofort)
  - Konsument: Core 0 (Kategorie 1); Core 2 (Kategorie 2)



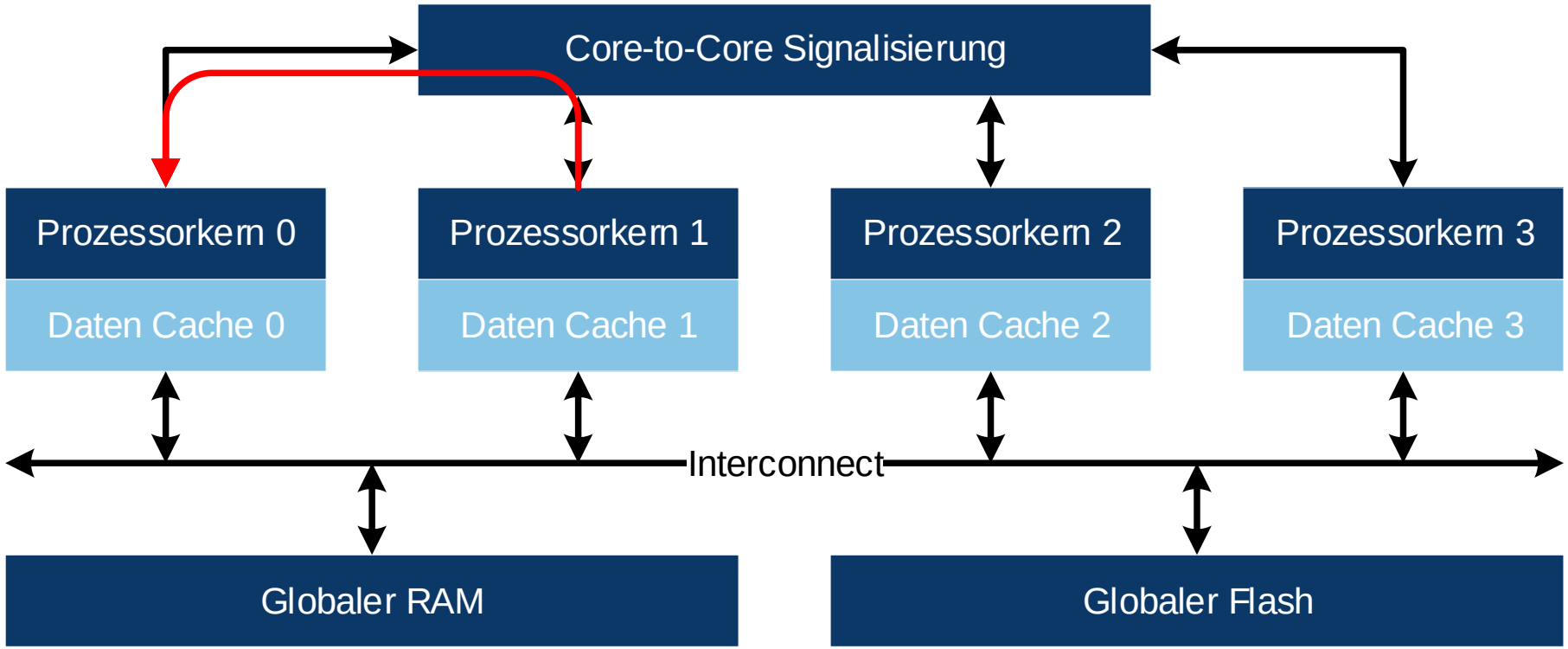
- **Beispiel:**
  - Kategorie 3
  - Produzent: Core 1
  - Konsument: Core 0 (Kategorie 3)



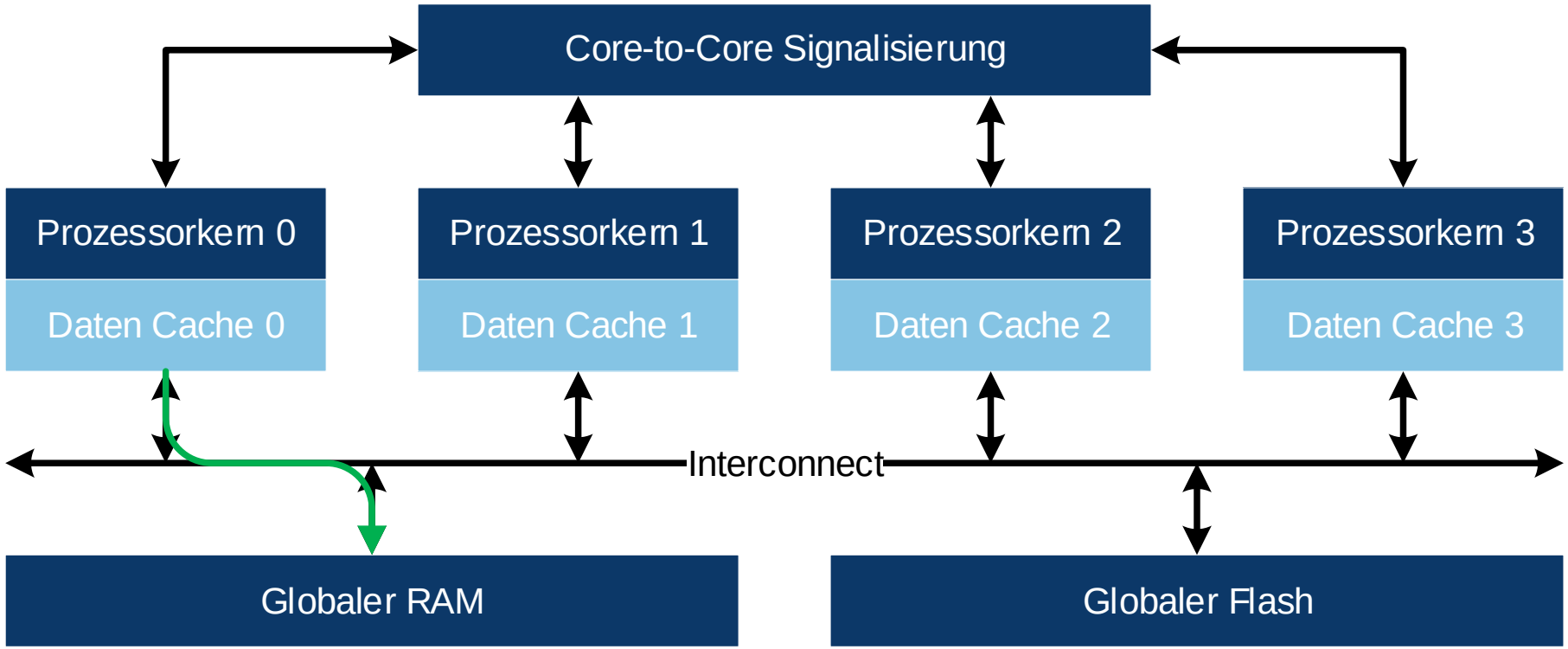
- **Beispiel:**
  - Kategorie 3
  - Produzent: Core 1
  - Konsument: Core 0 (Kategorie 3)



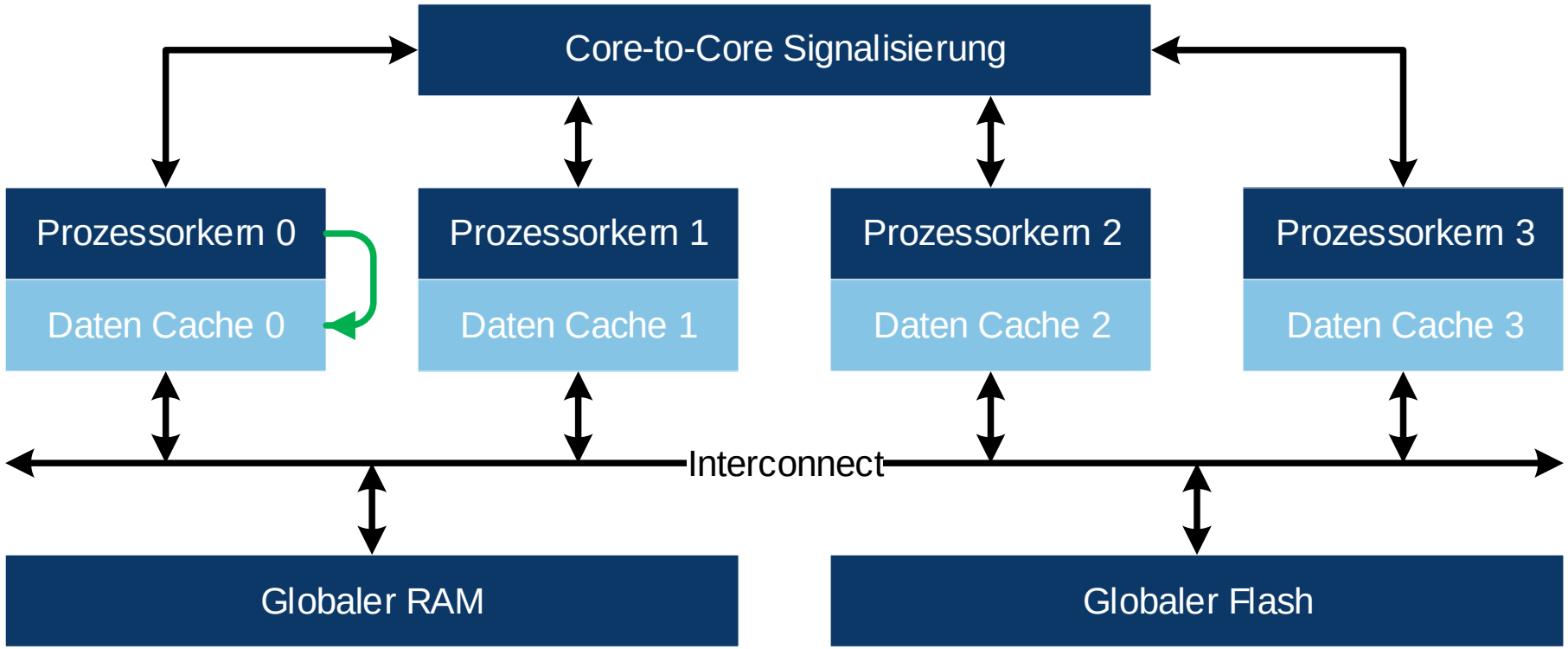
- **Beispiel:**
  - Kategorie 3
  - Produzent: Core 1
  - Konsument: Core 0 (Kategorie 3)



- **Beispiel:**
  - Kategorie 3
  - Produzent: Core 1
  - Konsument: Core 0 (Kategorie 3)

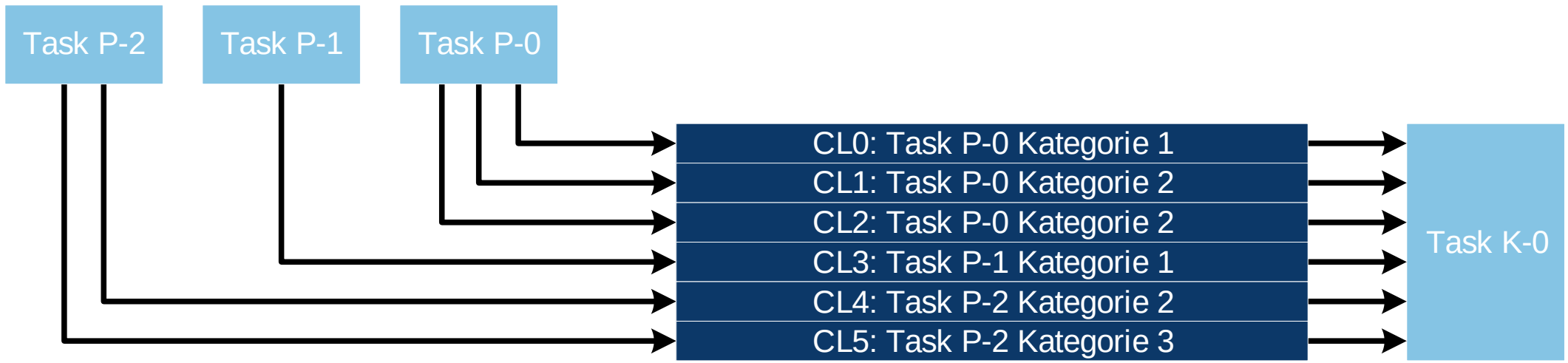


- **Beispiel:**
  - Kategorie 3
  - Produzent: Core 1
  - Konsument: Core 0 (Kategorie 3)



2. Konzept

- **Speicherallokation im Cache abhängig von:**
  - Speicherkapazität
  - Assoziation
  - Cacheline-Größe
- **Produzierende Tasks: P-0, P-1, P-2**
- **Konsumierende Tasks: K-0**
- **Je produzierende Task und Kategorie eine separate Cacheline**



3. Bewertung

3. Bewertung

- **Dauer eines Lesezugriff auf ein Datum:** $t_{read} = (t_{access} + t_{delay})$
- **Maximale Dauer aller Lesezugriffe auf ein Datum:** $t_{read,All} = n \cdot (t_{access} + t_{delay})$

3. Bewertung

- **Dauer eines Lesezugriff auf ein Datum:** $t_{read} = (t_{access} + t_{delay})$
- **Maximale Dauer aller Lesezugriffe auf ein Datum:** $t_{read,All} = n \cdot (t_{access} + t_{delay})$

- **Zugriffsdauer globaler RAM:** $t_{read,GRAM} = (t_{access,GRAM} + t_{delay,GRAM})$
- **Zugriffsdauer Cache:** $t_{read,Cache} = (t_{access,Cache})$
- **Aktualisierungsdauer Cache:** $t_{update,Cache} = (t_{read,GRAM} + \mathrm{t_{update}})$
- **Maximale Dauer aller Aktualisierungen Cache:** $t_{update,All} = m \cdot (t_{read,GRAM} + \mathrm{t_{update,Cache}})$

3. Bewertung

- **Dauer eines Lesezugriff auf ein Datum:** $t_{read} = (t_{access} + t_{delay})$
- **Maximale Dauer aller Lesezugriffe auf ein Datum:** $t_{read,All} = n \cdot (t_{access} + t_{delay})$

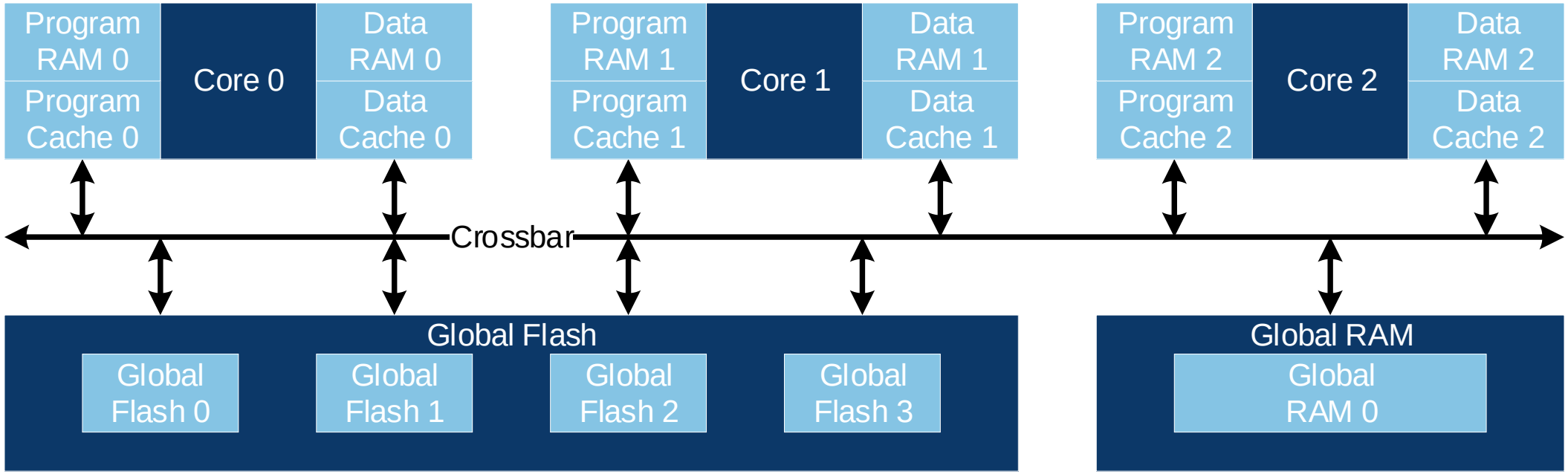
- **Zugriffsdauer globaler RAM:** $t_{read,GRAM} = (t_{access,GRAM} + t_{delay,GRAM})$
- **Zugriffsdauer Cache:** $t_{read,Cache} = (t_{access,Cache})$
- **Aktualisierungsdauer Cache:** $t_{update,Cache} = (t_{read,GRAM} + \mathrm{t_{update}})$
- **Maximale Dauer aller Aktualisierungen Cache:** $t_{update,All} = m \cdot (t_{read,GRAM} + \mathrm{t_{update,Cache}})$

- **Bewertung:** $n \cdot t_{read,GRAM} > n \cdot t_{read,Cache} + m \cdot (t_{read,GRAM} + t_{update,Cache})$

4. Resultate

4. Resultate

- **Mikrocontroller: Infineon AURIX TC298**
  - 3 Prozessorkerne
  - Zwei-Wege-Assoziativer Cache (8 KB) → 256 Bit Cacheline
  - Globaler RAM (32 KB)
- **Datensatz: 4KB Array**
  - 32 Bit-Werte



4. Resultate

- **Mikrocontroller: Infineon AURIX TC298**
  - 3 Prozessorkerne
  - Zwei-Wege-Assoziativer Cache (8 KB) → 256 Bit Cacheline
  - Globaler RAM (32 KB)
- **Datensatz: 4KB Array**
  - 32 Bit-Werte

Quelle	Ziel	Dauer (Ticks) Exklusiver Zugriff
Data Cache 1	Data Cache 1	2076
Global RAM	Data Cache 1	11287
Data Cache 1	Global RAM	12281
Global RAM	Global RAM	22529

4. Resultate

- **Mikrocontroller: Infineon AURIX TC298**
  - 3 Prozessorkerne
  - Zwei-Wege-Assoziativer Cache (8 KB) → 256 Bit Cacheline
  - Globaler RAM (32 KB)
- **Datensatz: 4KB Array**
  - 32 Bit-Werte

Quelle	Ziel	Dauer (Ticks) Exklusiver Zugriff	Dauer (Ticks) Konkurrierender Zugriff
Data Cache 1	Data Cache 1	2076	2076
Global RAM	Data Cache 1	11287	18410
Data Cache 1	Global RAM	12281	21959
Global RAM	Global RAM	22529	39870

5. Diskussion

5. Diskussion

- **Nutzung der Caches zur Intercore-Kommunikation**
  - Steigerung der Zugriffsgeschwindigkeit
  - Reduzierung der Anzahl der konkurrierenden Zugriffe
  - Reduzierung des Speicherbedarfs

- **Kategorisierung der Daten je Prozessorkern**
  - Verringerung der Aktualisierungshäufigkeit (Bedarfsabhängig)

- **Geplante Erweiterung:**
  - Automatisierte Analyse der Daten für die Intercore-Kommunikation
  - Automatisierte Verteilung der Daten für die Intercore-Kommunikation

Kontakt

Philipp Jungklass

IAV GmbH

Rockwellstraße 16, 38518 Gifhorn
Phone +49 5371 80 51141

Philipp.jungklass@iav.de

www.iav.com

Prof. Dr.-Ing. Mladen Berekovic

Universität zu Lübeck
Institut für Technische Informatik

Ratzeburger Allee 160, 23562 Lübeck

berekovic@iti.uni-luebeck.de

www.iti.uni-luebeck.de

Quellen

Alle für diesen Vortrag verwendeten Quellen sind in dem dazugehörigen Artikel aufgelistet!