Protected User-Level DMA in SCI Shared Memory Umgebungen

Mario Trams

University of Technology Chemnitz,
Chair of Computer Architecture

6. Halle — Chemnitz Seminar zu Parallelverarbeitung und Programmiersprachen
December 20th 2001, Chemnitz

Outline

- Background
- Speichermodelle
- SCI Basics
- Message Passing auf SCI?
- SCI und DMA–Engine
- Erweiterung der “herkömmlichen” SCI Architektur

Background

Allgemeines Arbeitsgebiet:

- zielt auf Message–Passing Applikationen ab
- Shared Memory bzw. Distributed Shared Memory wird als Kommunikationsgrundlage “missbraucht”

Ziel:

- Verbesserung der Architektur heutiger PCI–SCI Kommunikationshardware, um
  - ⋆ möglichst das gesamte Leistungspotential bis zur Anwendung zu bringen
  - ⋆ möglichst gute Basis für effiziente Message–Passing Bibliotheken zu schaffen

Speichermodelle

- **Shared Memory:**
  Mehrere Prozesse/Prozessoren einer parallelen Applikation arbeiten auf gleichen Speicherbereichen und können dadurch Informationen austauschen (z.B. auf einfachen Multiprozessorsystemen)

$\Rightarrow$ mehr oder weniger implizite Kommunikation

$\Rightarrow$ Bandbreitenproblem mit Hauptspeicher

Speichermodelle (cont.)

- Gegenteil: **Distributed Memory**
  Jeder Prozess(or) hat seinen eigenen Speicher und muss auf andere Art und Weise mit anderen Prozessen/Prozessoren kommunizieren (wie auch immer das geht).

$\Rightarrow$ expliziter Nachrichtenaustausch
$\Rightarrow$ kein Speicherbandbreitenproblem
$\Rightarrow$ dafür Kommunikationsbandbreitenproblem

Speichermodelle (cont.)

- Distributed Shared Memory (DSM):

  ⋆ Mischung aus beidem

  ⋆ Prozessoren besitzen eigenen Speicher, jedoch besteht die Möglichkeit auf Speicher anderer Prozessoren zuzugreifen

  ⇒ im Vergleich zu Distributed Memory schnelle Kommunikation, da keine Intervention durch Betriebssystem notwendig
  (User–Level Kommunikation wie auch bei Shared Memory)

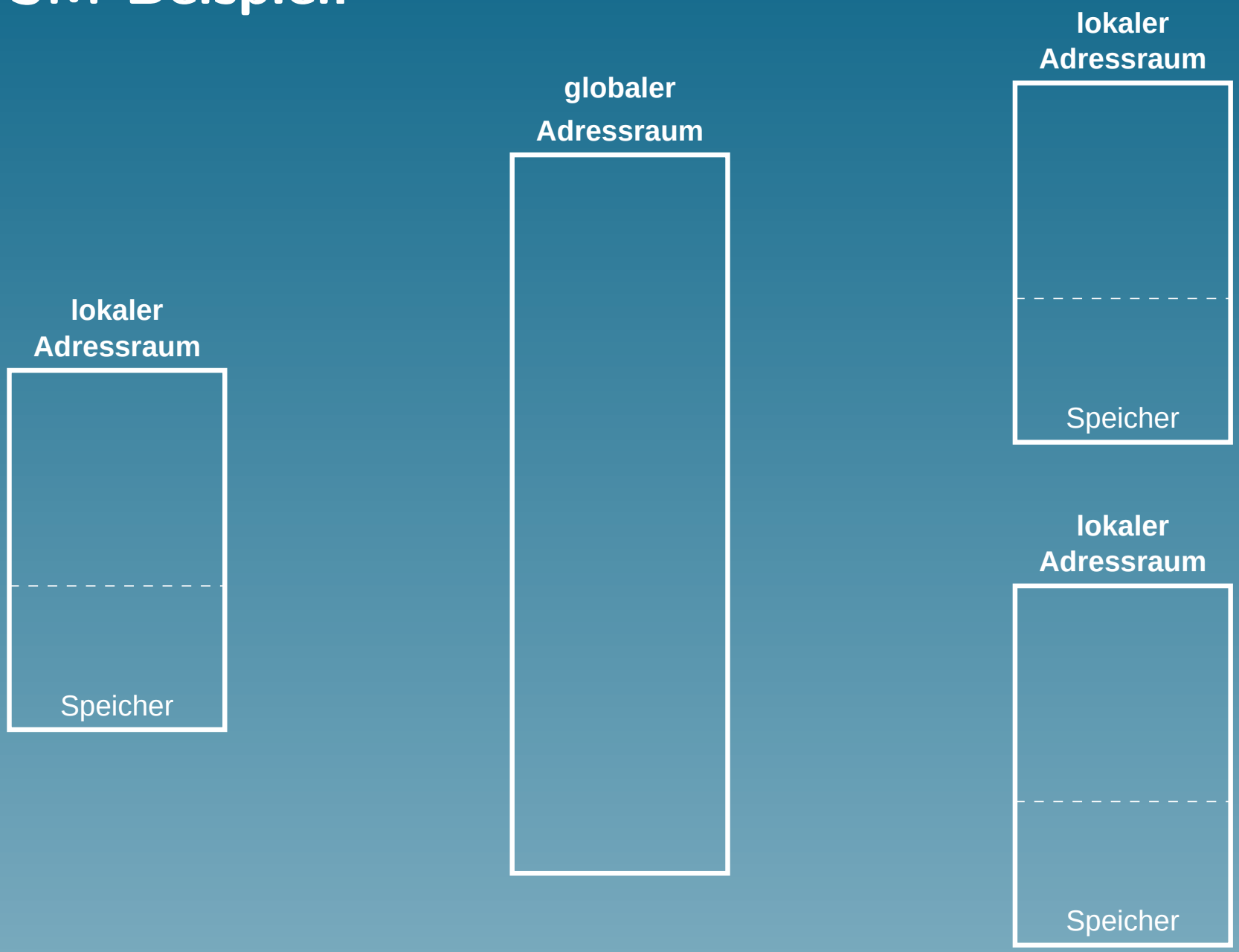
SCI Basics (cont.)

SCI (Scalable Coherent Interface):

- eine Hardware–Implementierung von DSM
- realisiert einen großen globalen Adressraum
- Prozesse/Rechenknoten können lokalen Speicher dort verfügbar machen
- Prozesse/Rechenknoten können auf globalen verfügbare Speicherbereiche zugreifen
  (wie auf lokalen Speicher, nur langsamer $\Rightarrow$ NUMA)

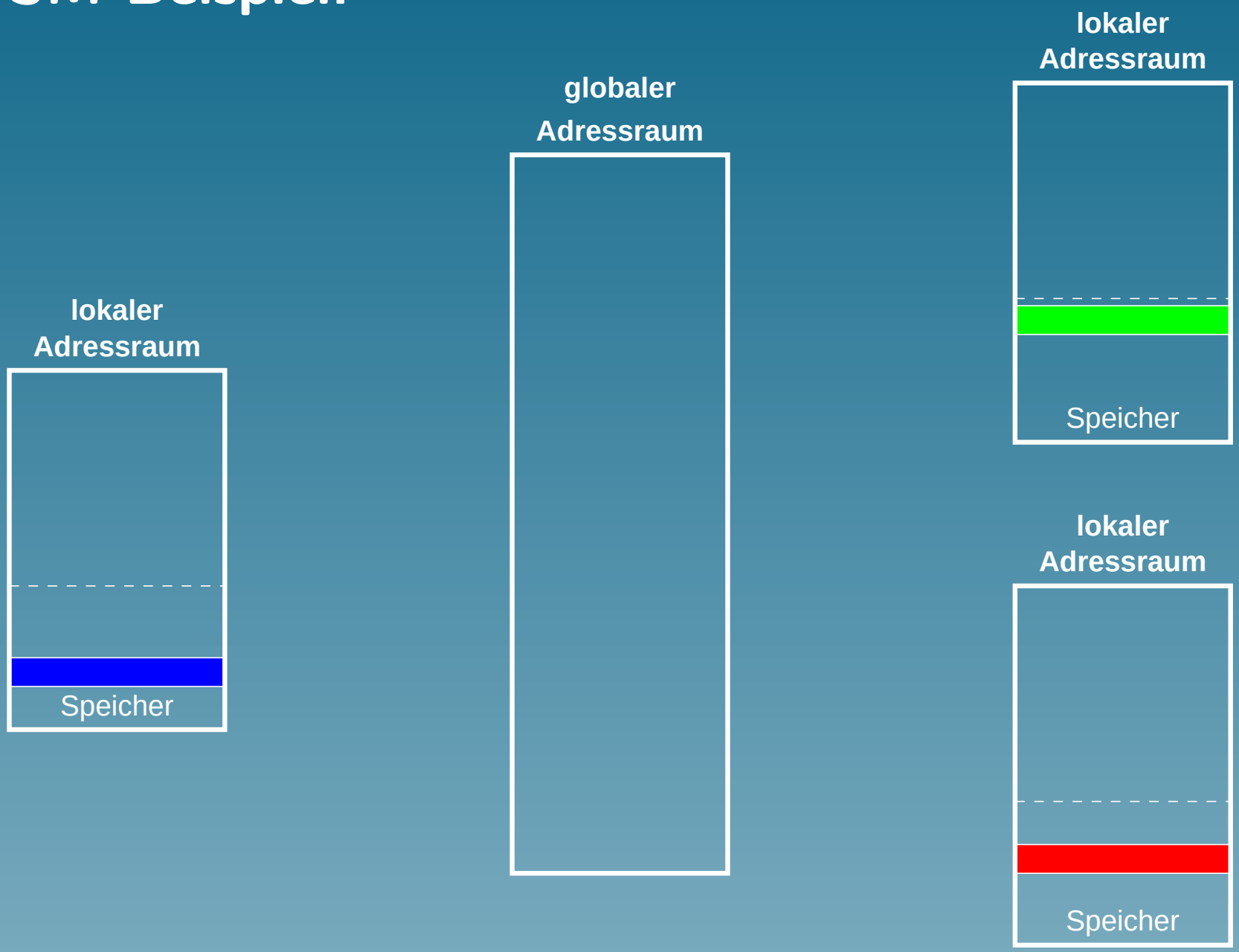
SCI Basics (cont.)

DSM Beispiel:



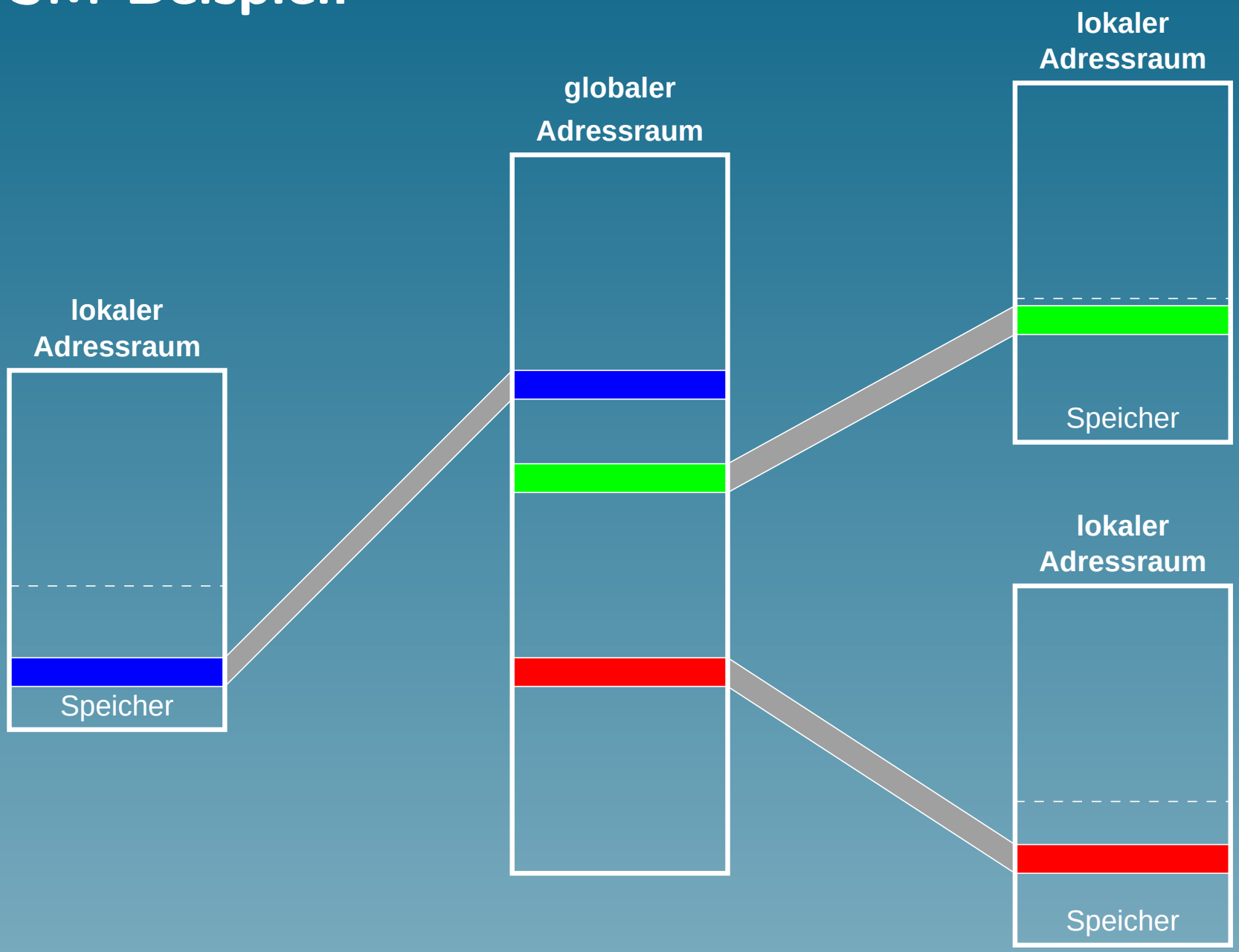
SCI Basics (cont.)

DSM Beispiel:



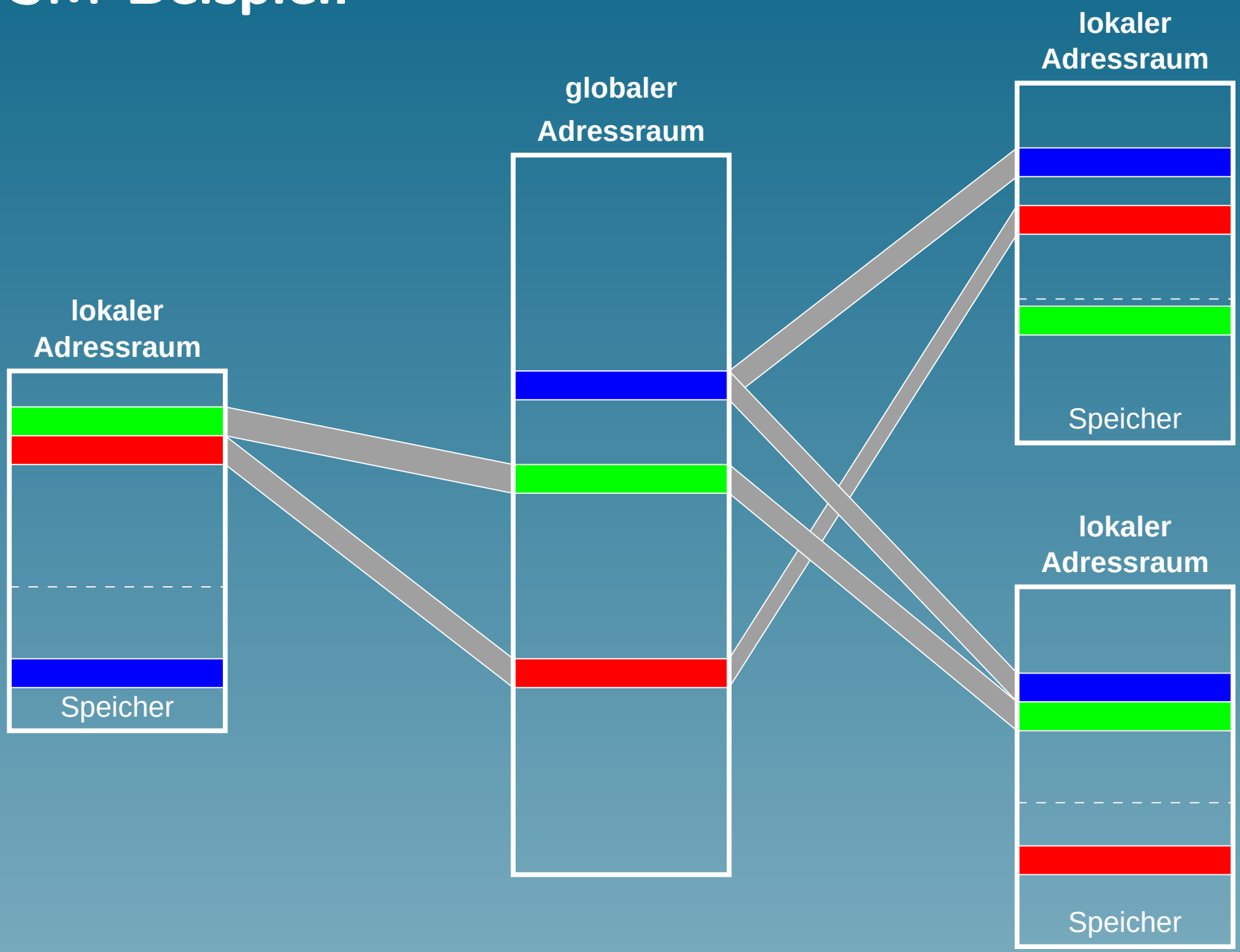
SCI Basics (cont.)

DSM Beispiel:



SCI Basics (cont.)

DSM Beispiel:



SCI Eckdaten

Eigenschaften der neuesten Generation:
(Dolphin PCI-64/66 PCI–SCI Karte)

- nutzbare Bandbreite $>$ 300MByte/s (remote Write)
- Latenzzeit $< 2\mu$s (Application to Application)
- Verbindungstopologien:
  - ⋆ Ring (kleine Systeme)
  - ⋆ Torus (2D, jetzt auch 3D)
  - ⋆ Switch–basiert (auch mit Ring gemischt)
- momentan **keine** Cache Kohärenz wegen PCI

SCI und Message–Passing

- SCI bietet hohe remote–Write Bandbreite sowie sehr geringe Latenzzeit, letzteres durch Einfachheit des Datentransfers begründet (simple Speicherreferenz)
- fehlende Cache–Kohärenz bei PCI–SCI Adaptern und geringe remote–Read Bandbreite macht PCI–SCI problematisch für Shared Memory Applikationen

$\Rightarrow$ die meisten SCI Systeme werden als Basis für Message–Passing Applikationen eingesetzt

SCI und Message–Passing (cont.)

Problem bei Message–Passing auf SCI:

- das Versenden von Messages geht zwar schnell, nimmt aber den Prozessor vollständig in Anspruch

$\Rightarrow$ hohe Bandbreite ist Makulatur, da für die CPU keine Zeit für die eigentliche Berechnung übrig bleibt

SCI und Message–Passing (cont.)

Problem bei Message–Passing auf SCI:

- das Versenden von Messages geht zwar schnell, nimmt aber den Prozessor vollständig in Anspruch

$\Rightarrow$ hohe Bandbreite ist Makulatur, da für die CPU keine Zeit für die eigentliche Berechnung übrig bleibt

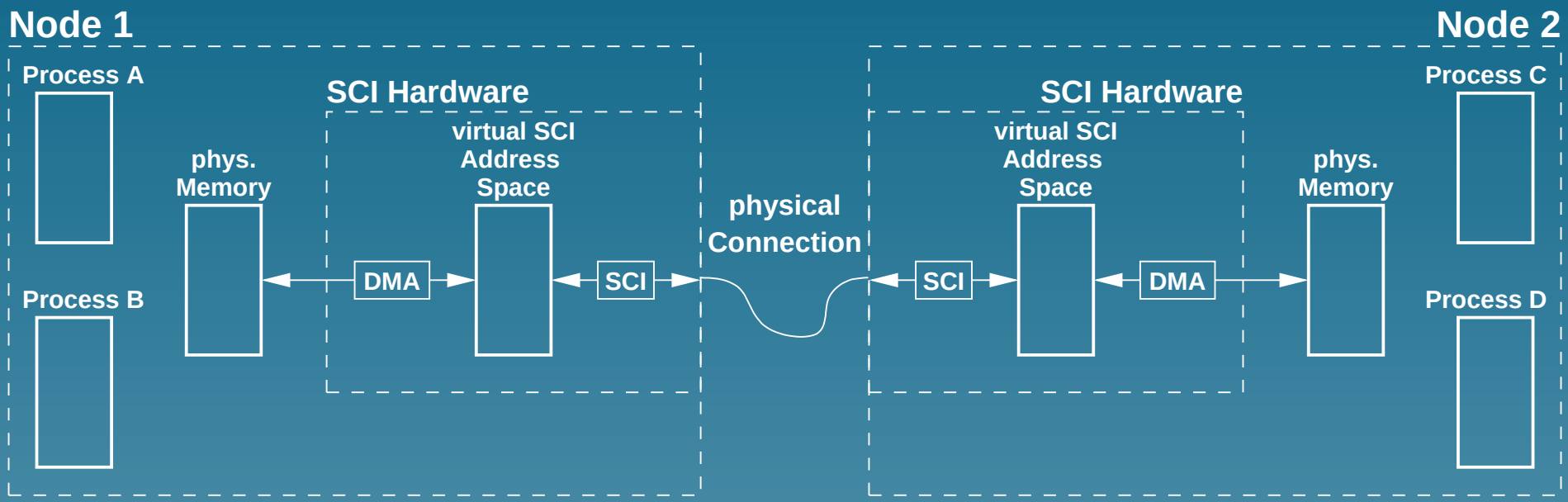
Abhilfe:

- eine sog. DMA–Engine in der Hardware, welche Daten blockweise im Hintergrund kopiert

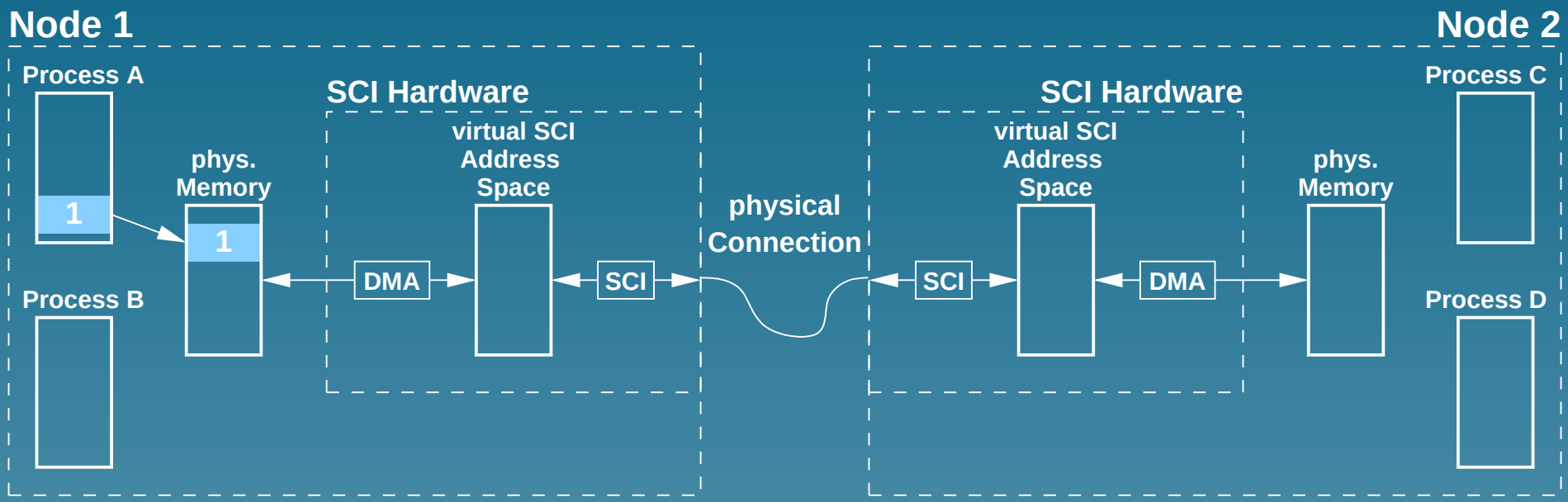
$\Rightarrow$ Überlappung von Berechnung und Datentransfer

$\Rightarrow$ lohnt sich jedoch nur ab bestimmter Blockgröße

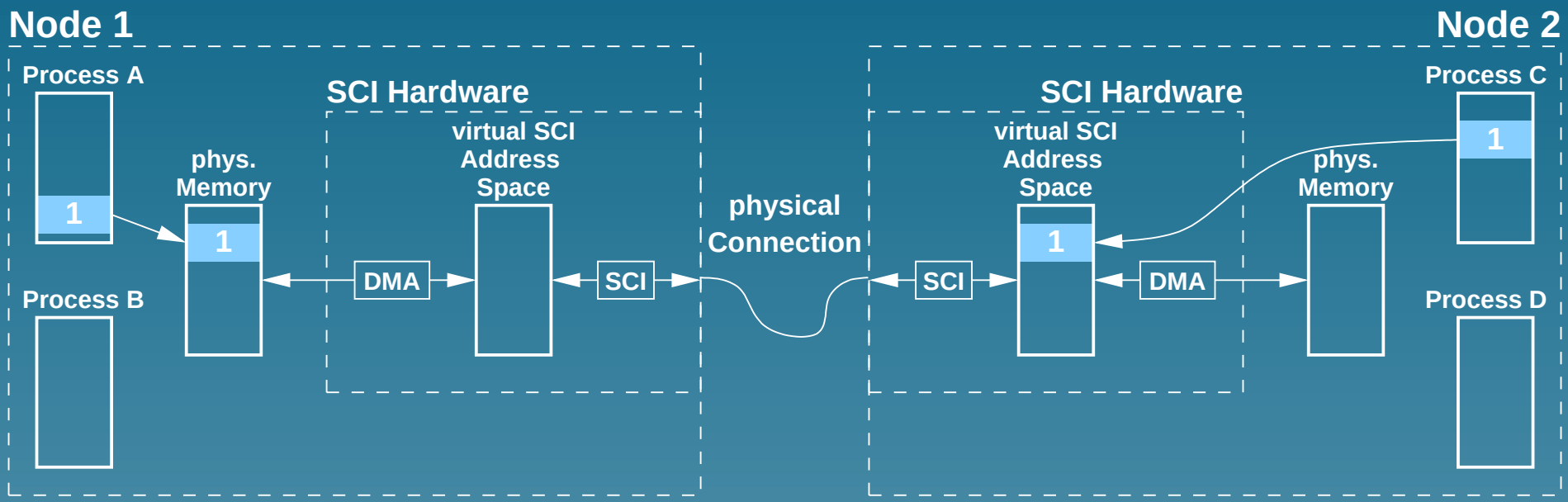
SCI mit DMA Engine



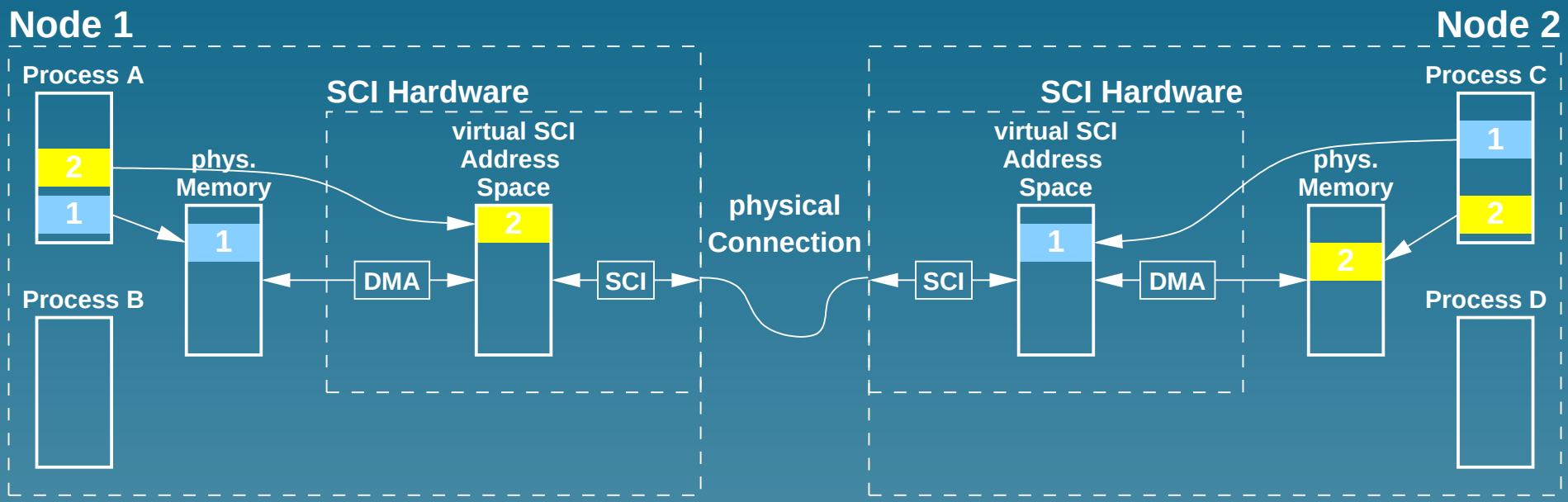
SCI mit DMA Engine



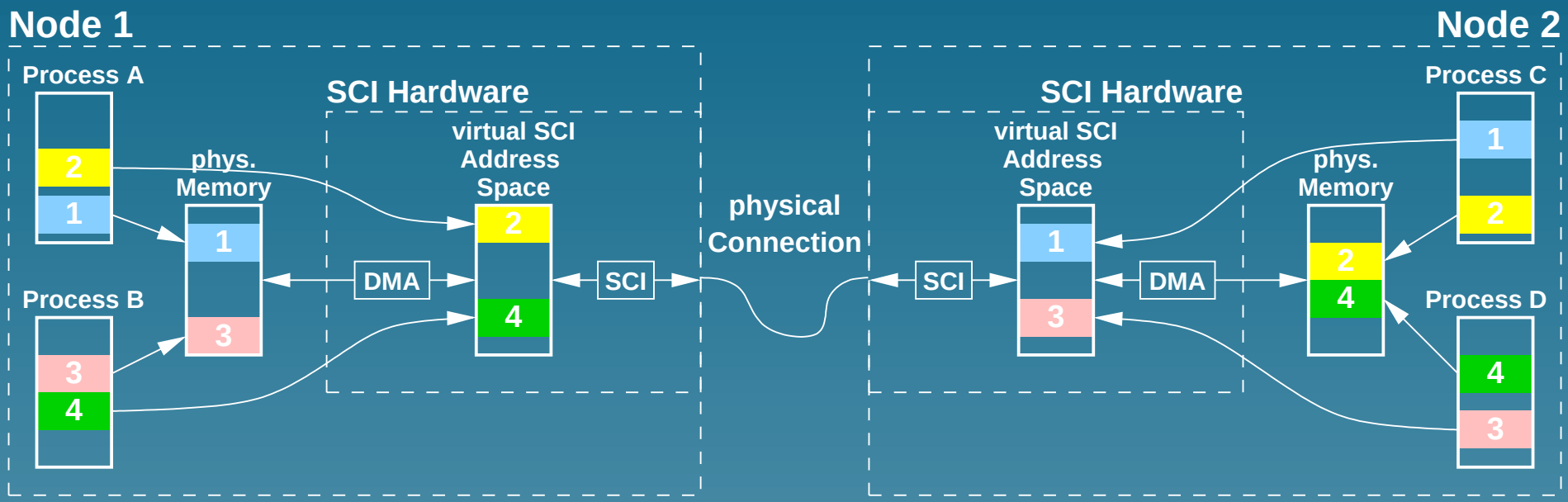
SCI mit DMA Engine



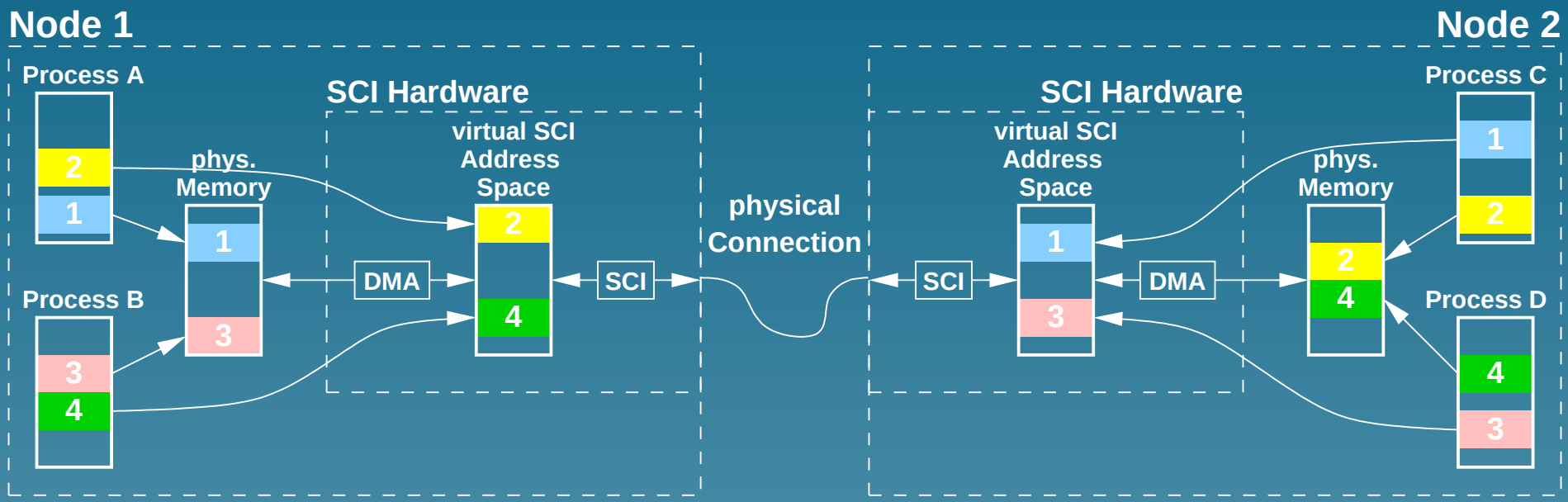
SCI mit DMA Engine



SCI mit DMA Engine



SCI mit DMA Engine



- Kopieren von Daten per CPU (PIO — Programmed IO) oder per DMA Engine ab bestimmter Blockgröße
- kleine Latenzzeit bleibt bei kurzen Nachrichten erhalten

SCI mit DMA Engine (cont.)

Aber:

- DMA–Engine darf aus Sicherheitsgründen nicht vom Anwendungsprogramm direkt programmiert werden
  - ⋆ SCI Hardware hat direkten Zugriff auf lokalen Speicher
  - ⋆ SCI Adressraum wird zwar “virtualisiert” (nur Subset sichtbar), jedoch ist momentan kein Schutzmechanismus vorhanden

SCI mit DMA Engine (cont.)

Aber:

- DMA–Engine darf aus Sicherheitsgründen nicht vom Anwendungsprogramm direkt programmiert werden
  - ⋆ SCI Hardware hat direkten Zugriff auf lokalen Speicher
  - ⋆ SCI Adressraum wird zwar "virtualisiert" (nur Subset sichtbar), jedoch ist momentan kein Schutzmechanismus vorhanden
- Programmierung muss indirekt über Betriebssystem erfolgen
- unnötig hohe Latenzzeit für DMA–Transfers

Verbesserte SCI Architektur

Ausweg:

- "Virtualisierung" der Sicht der SCI–Hardware auf lokalen Speicher
  (d.h. eine zweite Adressumsetztungstabelle)

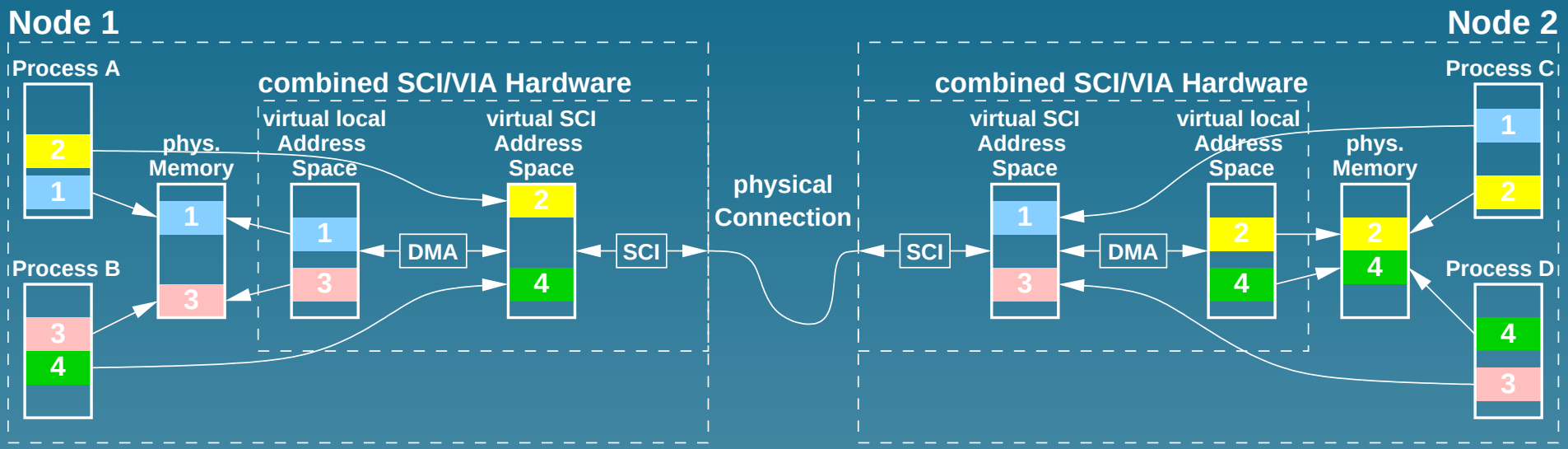
- gleichzeitige Integration von sog. *Protection Tags* in beide Tabellen

Verbesserte SCI Architektur

Ausweg:

- "Virtualisierung" der Sicht der SCI–Hardware auf lokalen Speicher
  (d.h. eine zweite Adressumsetztungstabelle)
- gleichzeitige Integration von sog. *Protection Tags* in beide Tabellen
- die Hardware überprüft (vergleicht) automatisch das Tag des initiierenden Prozesses mit dem Tag der betroffenen Adressbereiche
  - ⋆ prinzipielle Idee aus Virtual Interface Architecture (VIA) übernommen
  - ⋆ VIA sieht jedoch kein DSM vor

Verbesserte SCI Architektur (cont.)



⇒ DMA–Engine erlaubt nur die Zugriffe, wofür der zugehörige Prozess authorisiert ist

⇒ Speicherschutz ist gewährleistet und Betriebssystemaufruf ist nicht notwendig

⇒ Umschaltpunkt PIO/DMA verlagert sich nach unten

⇒ verbesserter Gesamtdurchsatz

Verbesserte SCI Architektur (cont.)

Nebeneffekt:

- virtuelle Sicht auf lokalen Adressraum vereinfacht das Speichermanagement wesentlich
- echte Zero–Copy Datentransfers (Versenden von Nachrichten ohne unnötiges Zwischenspeichern) können relativ einfach realisiert werden
- Zero–Copy mit heutiger kommerzieller PCI–SCI Hardware ist hingegen nur schwer realisierbar

Entwicklungsstatus

- FPGA–basierter Proof–of–Concept Prototyp
- grundlegende Architektur ist implementiert (d.h. beide Adressumsetzungen)
- Programmed IO ist fast vollständig implementiert mit vergleichsweise recht guten Leistungswerten: remote–Write Bandwidth 116MByte/s, PCI–to–PCI Latency 2.7μs
- Konzept der DMA–Engine sowie unterstützende Mechanismen zwar prinzipiell fertig, aber noch nicht implementiert

Zusammenfassung / Ausblick

- kurze Einführung in DSM Systeme und SCI
- DMA Engine in heutiger PCI–SCI Hardware und deren Probleme
- Darstellung eines prinzipiellen Ansatzes zur Verbesserung des DMA–Mechanismus bzw. der allg. SCI Architektur

Zusammenfassung / Ausblick

- kurze Einführung in DSM Systeme und SCI
- DMA Engine in heutiger PCI–SCI Hardware und deren Probleme
- Darstellung eines prinzipiellen Ansatzes zur Verbesserung des DMA–Mechanismus bzw. der allg. SCI Architektur
- **Was bringt die Zukunft?**
  Die vorgestellten Mechanismen werden in einer der nächsten Versionen der kommerziellen PCI–SCI Adapter von Dolphin realisiert sein.