

Universität Karlsruhe (TH)

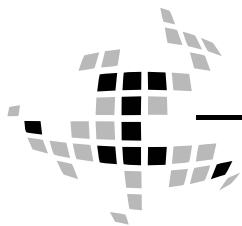
Forschungsuniversität · gegründet 1825

Optimierungstechniken für Hochgeschwindigkeitskommunikation

Prof. Dr. Walter F. Tichy

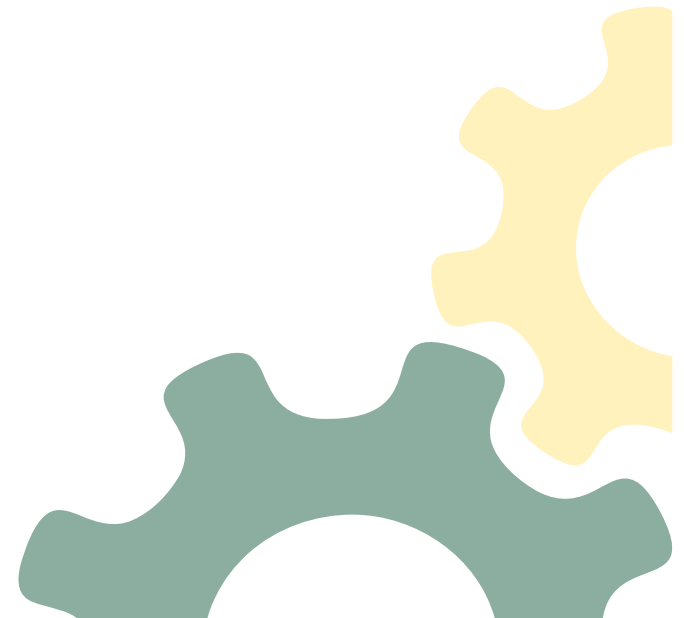
Dr. Victor Pankratius

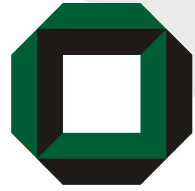
Ali Jannesari



Fakultät für **Informatik**

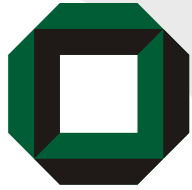
Lehrstuhl für Programmiersysteme



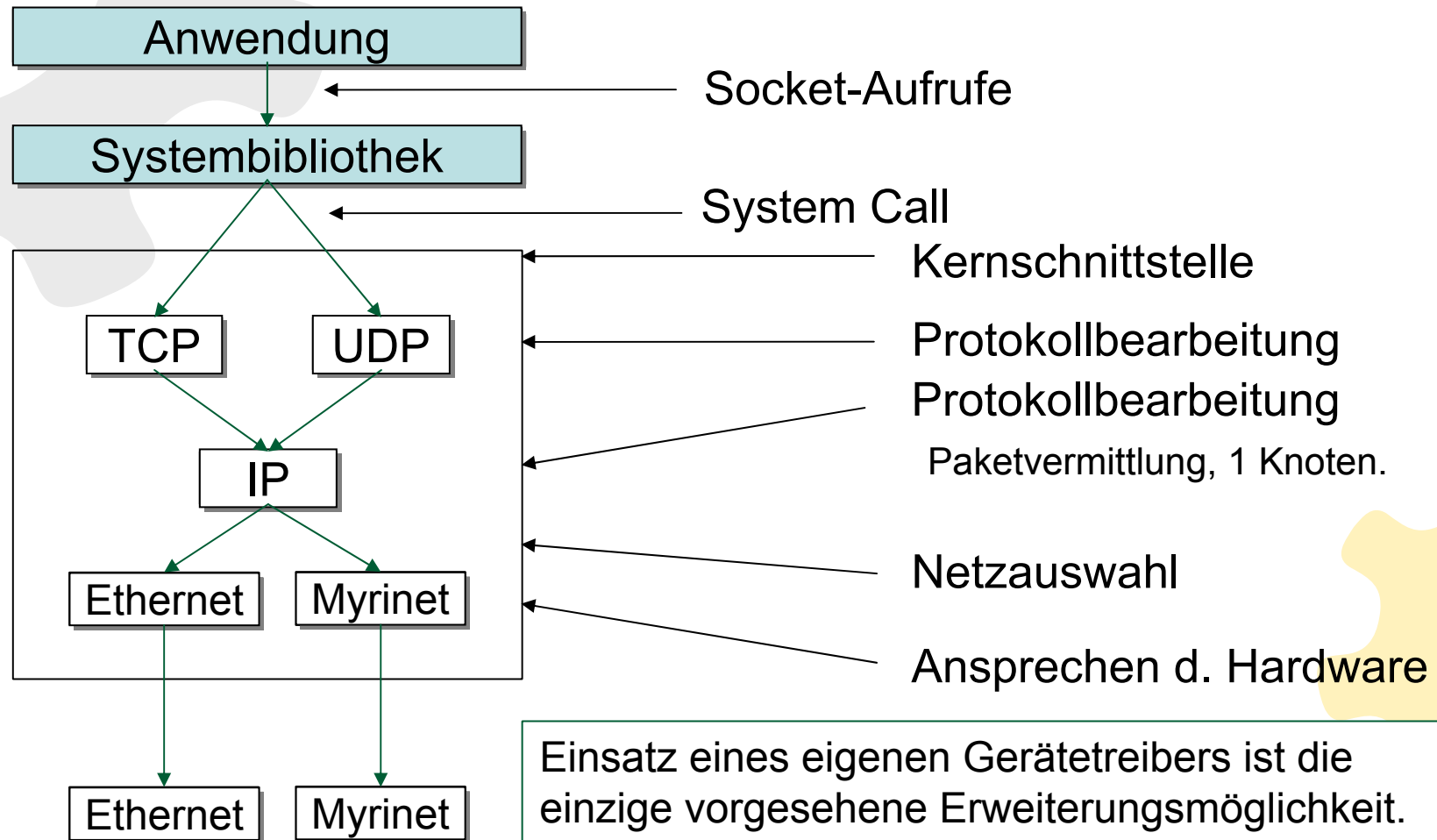


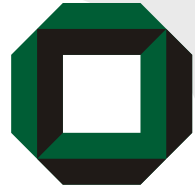
Vorlesung Rechnerbündel

- Architektur von Rechnerbündeln
 - Hochgeschwindigkeitsnetzwerke
 - Hochgeschwindigkeitskommunikation
 - TCP/IP-UDP/IP-Probleme
 - Optimierungstechniken
 - Kommunikationsmodelle
 - Beispielsysteme



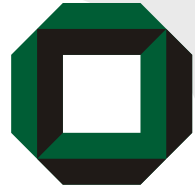
Kommunikationspfad in Unix





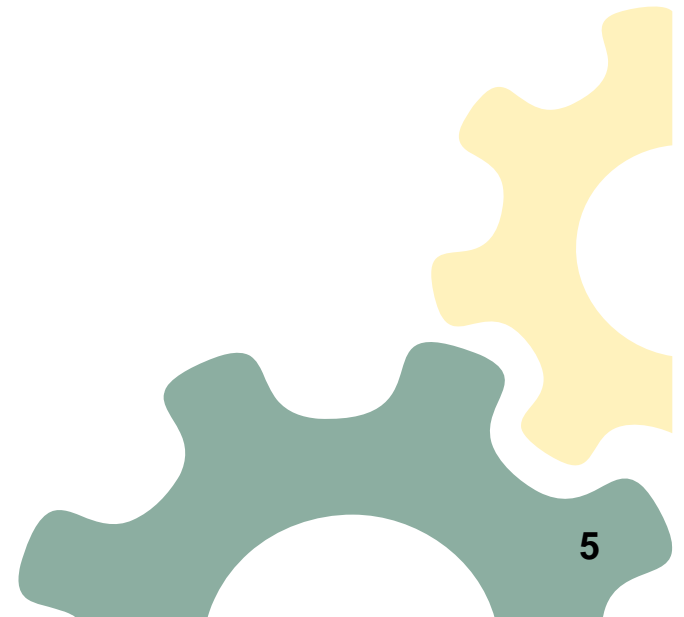
Nachteile der Treiberlösung

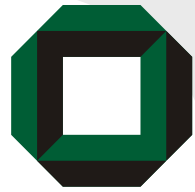
- Software-Overhead bedingt Leistungsgrenze.
- Es gibt kaum Optimierungsmöglichkeiten.
- Beste FastEthernet-Implementierungen:
 - ~75 μ s Latenz (ca. 10 μ s Hardware-Latenz)
 - ~10 Mbyte/s Bandbreite (85% Hardware-Bandbreite)
- Gründe:
 - Alles passiert im Kern: Eintritt/Austritt, Kontextwechsel, **Kopieraufwand**.
 - keine Vereinfachungen/Optimierungen im Protokollturm
 - keine Ausnutzung von Hardware-Eigenschaften
 - Empfangen ausgelöst durch Unterbrechungen



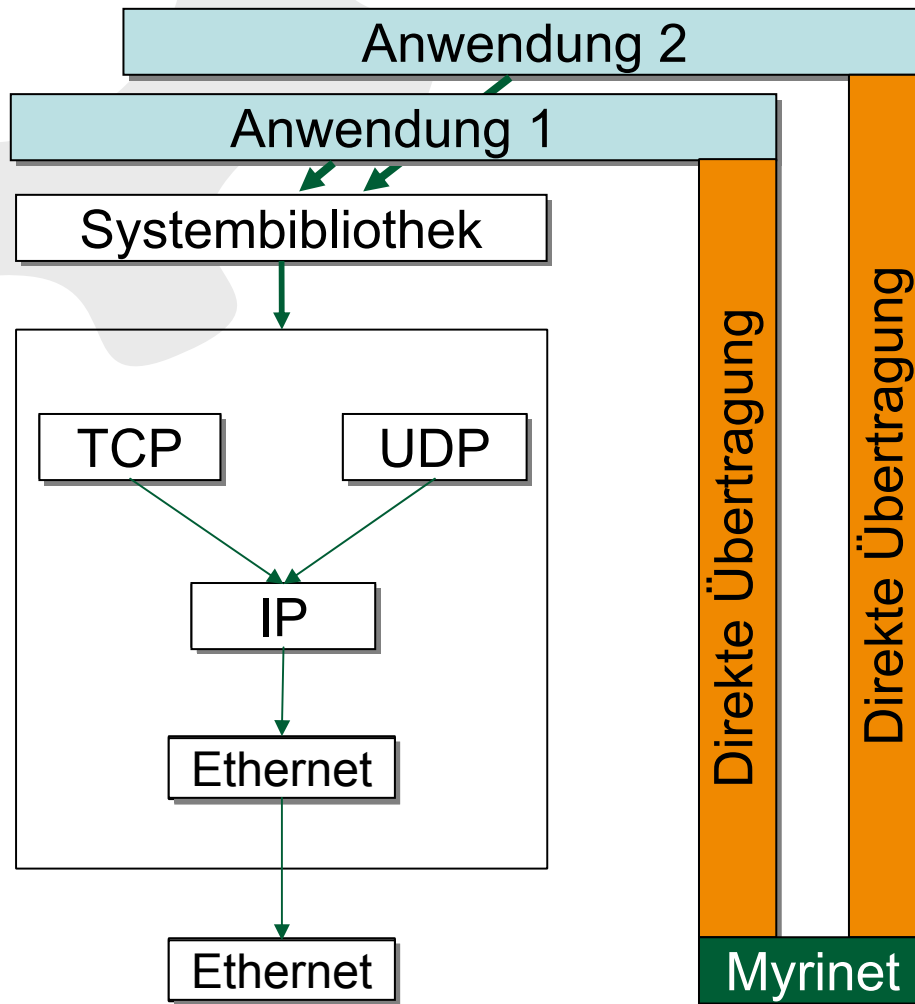
Vorlesung „Rechnerbündel“

- Architektur von Rechnerbündeln
 - Hochgeschwindigkeitsnetzwerke
 - Hochgeschwindigkeitskommunikation
 - TCP/IP-UDP/IP-Probleme
 - Optimierungstechniken
 - Kommunikationsmodelle
 - Beispielsysteme

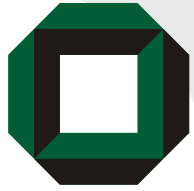




Umgehung des normalen Protokollstapels



- Ansteuerung der Kommunikations-Hardware direkt aus dem Adressraum der Applikation unter Umgehung des Betriebssystems.
- Pufferung und Steuerung getrennt pro Anwendung im Adapter und Hauptspeicher.



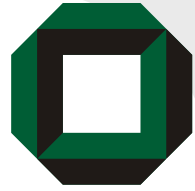
Umgehung des normalen Protokollstapels

Kontra:

- Aufgaben/Funktionen, die sonst von Systembibliotheken und dem Betriebssystemkern bereitgestellt wurden, entfallen oder müssen von der Kommunikationsbibliothek (also Bibliotheken oder der Kommunikationshardware) übernommen/bereitgestellt werden:
 - Standardisierte Kommunikationsschnittstellen (z.B. Sockets),
 - Schutzmaßnahmen der Prozesse untereinander und gegenüber der Hardware,
 - Koordination konkurrierender Prozesse,
 - Protokollverarbeitung: Verbindungsauf- und -abbau,
 - Zuordnung von Datenpaketen zu Prozessen/Empfangspuffern.

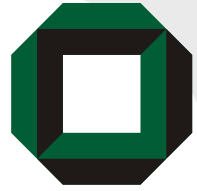
Pro:

- Für spezifische Kommunikationsbedürfnisse und für spezielle Hardware **kann** optimiert werden.



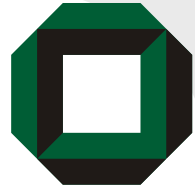
Optimierungstechniken

- Vereinfachte Protokolle
- Vermeidung von Systemaufrufen
- Schnelle Systemaufrufe
- Schnelle Unterbrechungsroutrinen
- Abfrage statt Unterbrechung
- Vermeidung von Kopieroperationen



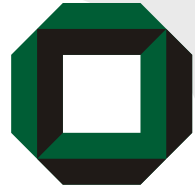
Vereinfachte Protokolle

- Abstimmung auf Fähigkeiten der Hardware
- Adressierung (Knoten / Prozess / Endpunkt)
- Datenaustausch (Strom / paketorientiert)
- Fragmentierung (intern / extern)
- Reihenfolgetreue (ja / nein)
- Verbindungsaufbau (ja / nein, wenn ja: wie?)
- optimistische Protokolle (Fehlerfall ist je nach Hardware unwahrscheinlich.)
- Fehlerbehandlung



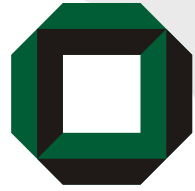
Vermeidung von Systemaufrufen

- Einblenden der Kommunikationshardware in den Adressraum der Anwendung ermöglicht Hardwarezugriff außerhalb des Kerns.
- **ABER:** verschiedene Operationen, wie z.B. das Einblenden der Adressräume selbst, die Manipulation der Adressumsetzungstabellen und Sicherheitsmechanismen können nur im Kern realisiert werden.
- **Ziel:** so wenige Kerneintritte wie möglich, nur so viele wie nötig.



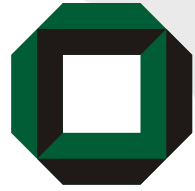
Schnelle Systemaufrufe

- Verwenden eines zweiten Einsprungpunkts
 - mit etwas anderen Garantien seitens des Kerns,
 - mit etwas anderen Vorbedingungen,
- insbesondere:
- Sichern nur bestimmter CPU-Register.
 - Kein Kopieren von Übergabedaten.
 - Kein Überprüfen von Parametern.
 - Direkte Rückkehr zum aufrufenden Prozess (normalerweise wird nach einem Systemaufruf immer der Prozessabwickler aktiviert).



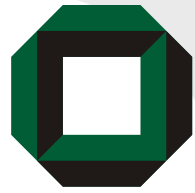
Schnelle Unterbrechungsroutrinen

- Optimierung der Unterbrechungsfunktionen innerhalb des Kerns.
 - Beispiel: Granularität von Sperren erhöhen.
- Optimierung der Unterbrechungsroutrine für die spezielle Netzwerkkarte.
- Verzögerung der Unterbrechungsauslösung für kurze Zeit oder bis eine bestimmte Anzahl Pakete eingetroffen ist, um mehrere empfangenen Pakete in einer Unterbrechung behandeln zu können.



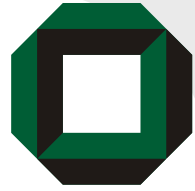
Abfrage statt Unterbrechung

- Anwendung überprüft selbst (periodisch), ob neue Pakete empfangen wurden.
 - Einblenden der Statusregister des Netzwerkes in den Adressraum der Anwendung.
 - Ereignis setzt ein Status-Bit, nun sichtbar für die Anwendung.
 - Effekt: Sowohl Unterbrechung als auch Kerneintritt werden eingespart.
- Problem: verzögerte (keine?) Reaktion auf eingehende Nachricht, weil die Anwendung gerade nicht ausgeführt wird.
- Problem: Aktives Warten (bei leeren Puffern) verbraucht Rechenzeit.
- Offene Frage: Wie gelangen die Daten selbst aus dem Netzwerkadapter in den Speicherbereich der Anwendung?



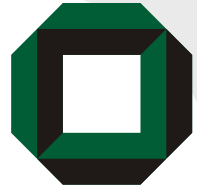
Klingel („Doorbell“)-Region

- Idee der vorhergehenden Folie, nun angewendet auf den Sendevorgang:
 - Die Anwendung informiert den Netzwerkadapter durch Schreiben in ein bestimmtes, eingeblendetes Register darüber, dass eine Nachricht verschickt werden soll.
 - Die Hardware des Adapters könnte z.B. so ausgelegt sein, dass ein Zugriff auf bestimmte Register eine Unterbrechung für den adaptereigenen Prozessor bewirkt.
- Vorteil: Kerneintritt gespart.
- Offene Frage auch hier: Wie kommen die Daten selbst aus dem Speicher der Anwendung zum Netzwerkadapter?



Vermeidung von Kopieroperationen

- Anwendung ↔ Kern ↔ Netzadapter
- innerhalb der Programmierumgebung
 - **Verbesserung:** Anwendungsdaten möglichst ohne Umweg über den Kern zum Netzadapter bringen.
- innerhalb der Protokollverarbeitung
 - **Verbesserung:** Puffer mit Reserve für übergeordnete Protokolldaten,
 - **Verbesserung:** möglichst wenige Schichten (Referenzübergabe statt Kopieroperationen).
- unterschiedliche Strategien für
 - Sender/Empfänger
 - kleine/große Pakete
- Bandbreite von Zero-Copy (=One-Copy), Two-Copy bis N-Copy.



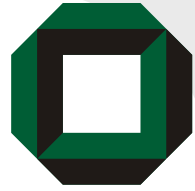
Mögliche Verbesserung des Datentransfers

Wie kommen Daten aus den Puffern der Anwendung in den Netzadapter und zurück?

Möglichkeit 1: **Programmierte E/A (Programmed I/O, PIO)**

CPU kopiert Daten Wort für Wort direkt in den Netzadapter. Zwar hohe CPU-Last, aber kein Kerneintritt. Nur für kleine Pakete sinnvoll.

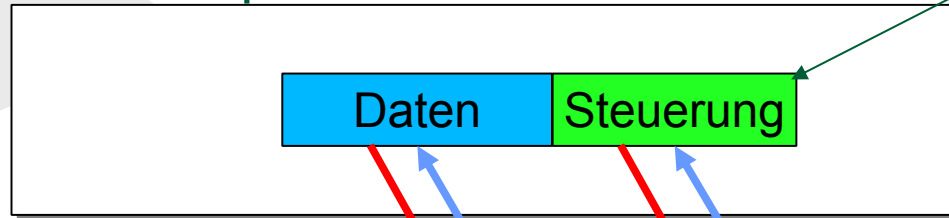
Asymmetrie: PIO nur CPU-seitig



PIO (programmed I/O)

Netzadapter

„Doorbell-Region“

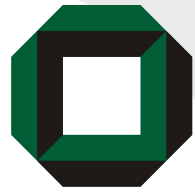


Senden
Empfangen

PIO

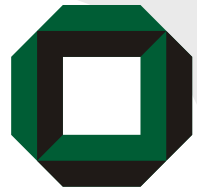
Hauptspeicher





PIO (Bewertung)

- + Einfach (Kopieroperationen)
- + Definierte Semantik:
 - Senden kehrt erst dann zurück, wenn Daten sicher im Netzadapter angekommen sind.
 - Empfang wird erst dann ausgeführt, wenn Daten im Netzadapter empfangen wurden.
- + Sehr wenig Aufwand, kein Kerneintritt. Daher schnelles Senden kurzer Nachrichten
- Starke CPU Belastung bei größeren Datenmengen
- Unakzeptable Leistung beim Empfangen
 - Empfangstest per Abfrage im Adressbereich des Netzadapters ist langsamer als Abfrage im Hauptspeicher.
 - Bei erfolgreichem Empfangstest müssen anschließend noch die Daten gelesen werden.

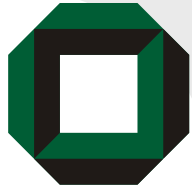


Mögliche Verbesserung des Datentransfers

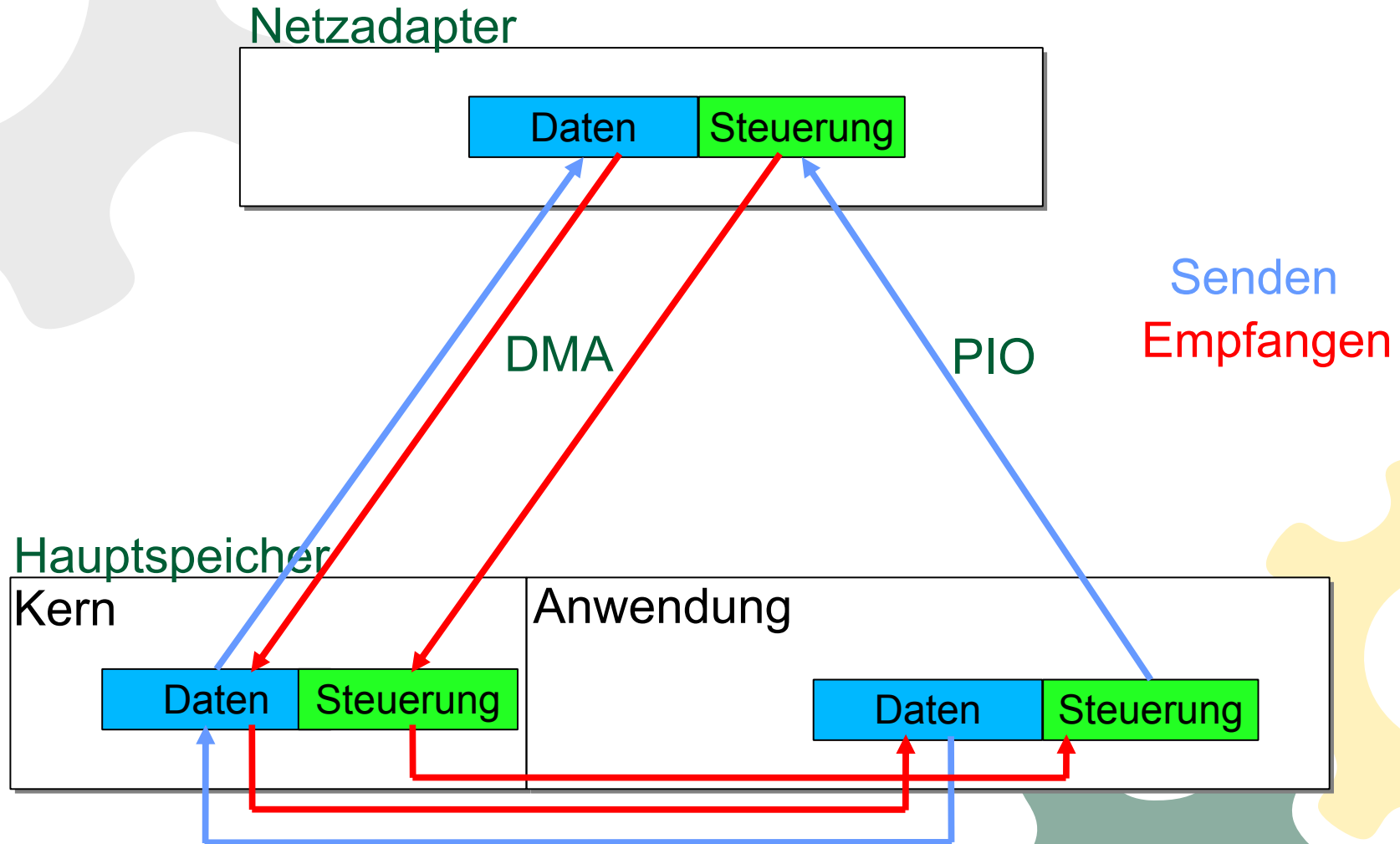
Möglichkeit 2: Direct Memory Access (DMA)

Datenblock wird durch einem separaten „DMA-Treiber“ von einer physikalischen Speicheradresse zu einer anderen kopiert. (Annahme: DMA-Treiber befindet sich auf dem Netzadapter.)

- Kern-Eintritt nötig,
 - da physikalische Adressen notwendig, aber Anwendung keine physikalischen Adressen kennt, und
 - da Anwendungsspeicher ausgelagert sein könnte.
- DMA-Puffer müssen vorab zugewiesen werden.
- Asynchrone DMA-Ausführung erlaubt Weiterarbeit der CPU.
- Aber Semantik-Probleme durch asynchrone Ausführung:
Man muss berücksichtigen/herausfinden, wann der DMA-Puffer vollständig gelesen bzw. komplett geschrieben wurde.

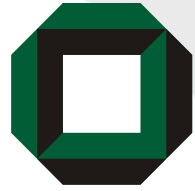


Copy DMA – Kopier-DMA



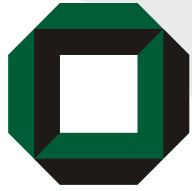
PIO Copy (evtl. DMA)

Prof. W. F. Tichy, Dr. V. Pankratius, A. Jannesari



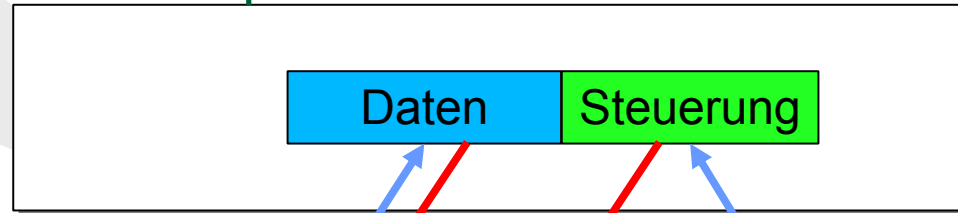
Kopier-DMA (Bewertung)

- + Relativ einfach (bei statischer Allokation von DMA Puffern)
- + Definierte Semantik:
 - Senden kehrt erst dann zurück, wenn Daten sicher im Kern angekommen sind.
 - Empfangen wird erst dann ausgeführt, wenn Daten im Kern empfangen wurden.
- + Kopieren auf Speicherbus schneller als auf PCI
- 2 zusätzliche Kopieroperationen (Senden & Empf.)
- Verwaltung/Aufsetzen der DMA-Puffer im Kern

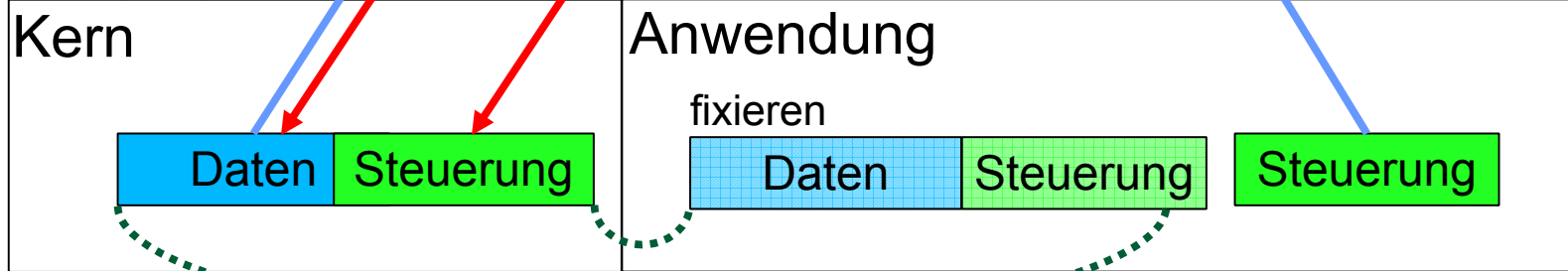


Direktes DMA

Netzadapter



Hauptspeicher

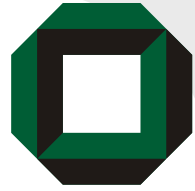


Senden
Empfangen

DMA

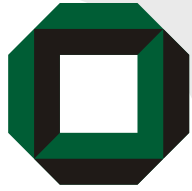
PIO

MMAP

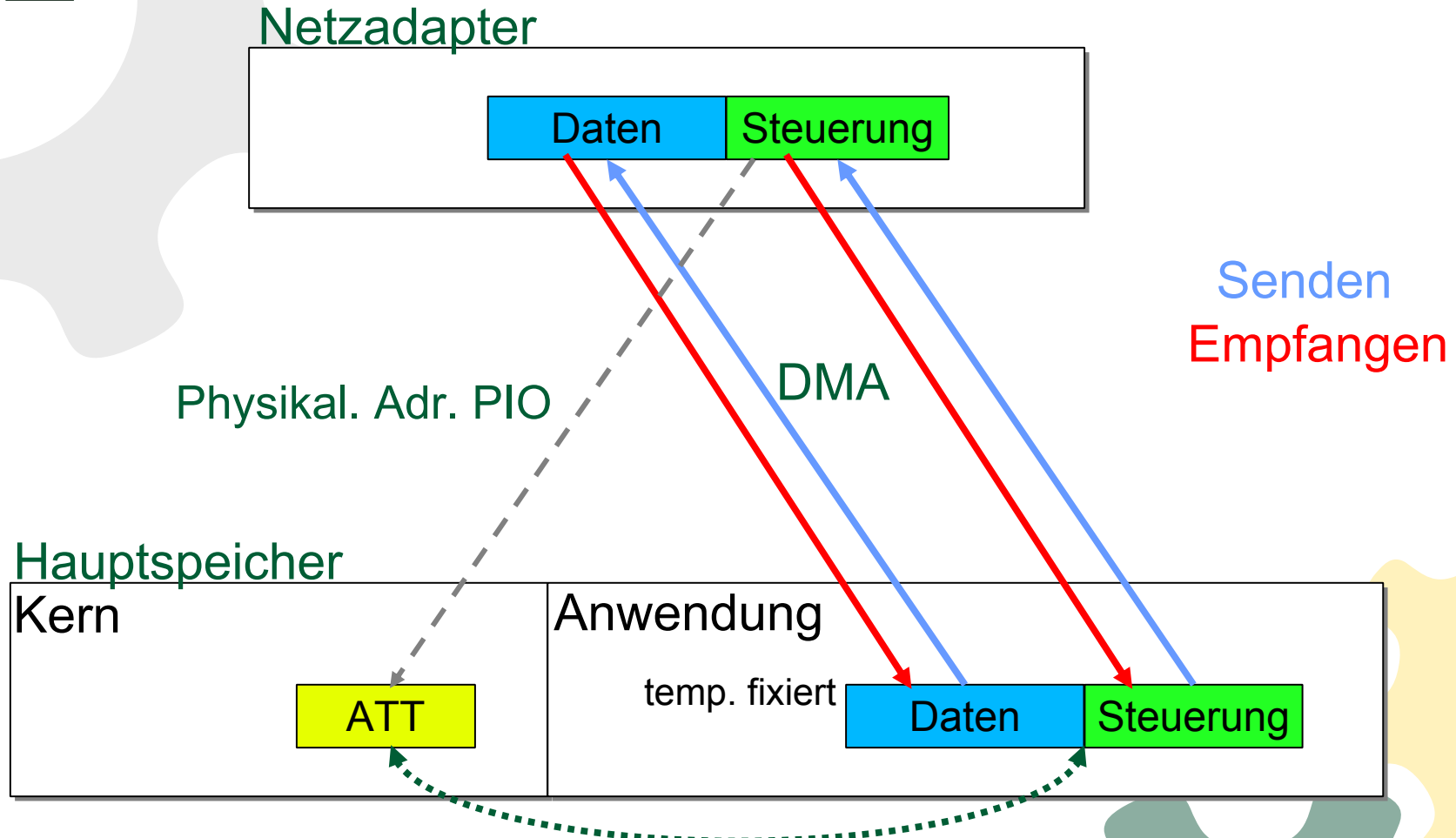


Direktes DMA (Bewertung)

- + Wenig CPU Belastung
- + Weniger Kopier-Operationen (als Kopier-DMA)
- Schwierige Semantik:
 - Das Senden kehrt schon zurück, wenn Daten noch zum Adapter unterwegs sind. Gegenmaßnahmen zur Pufferwiederverwendung nötig.
 - Sondermaßnahmen für Empfang nötig: Netzadapter muss signalisieren, wenn DMA komplett ist.
 - DMA-Puffer (und Zuordnung) müssen bei Start der Applikation statisch festgelegt werden!
Dynamisch erzeugte Kommunikationspuffer (wie in MPI erforderlich) sind kaum möglich
- Beschränkte Anzahl an DMA-Puffern



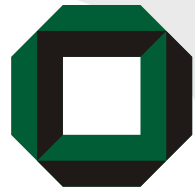
V2P (virtual to physical) DMA



Dynamic Address Translation

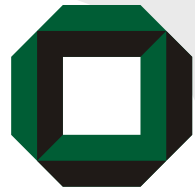
(Anwendung erhält physik. Adressen des Puffers vom Kern)

Prof. W. F. Tichy, Dr. V. Pankratius, A. Jannesari

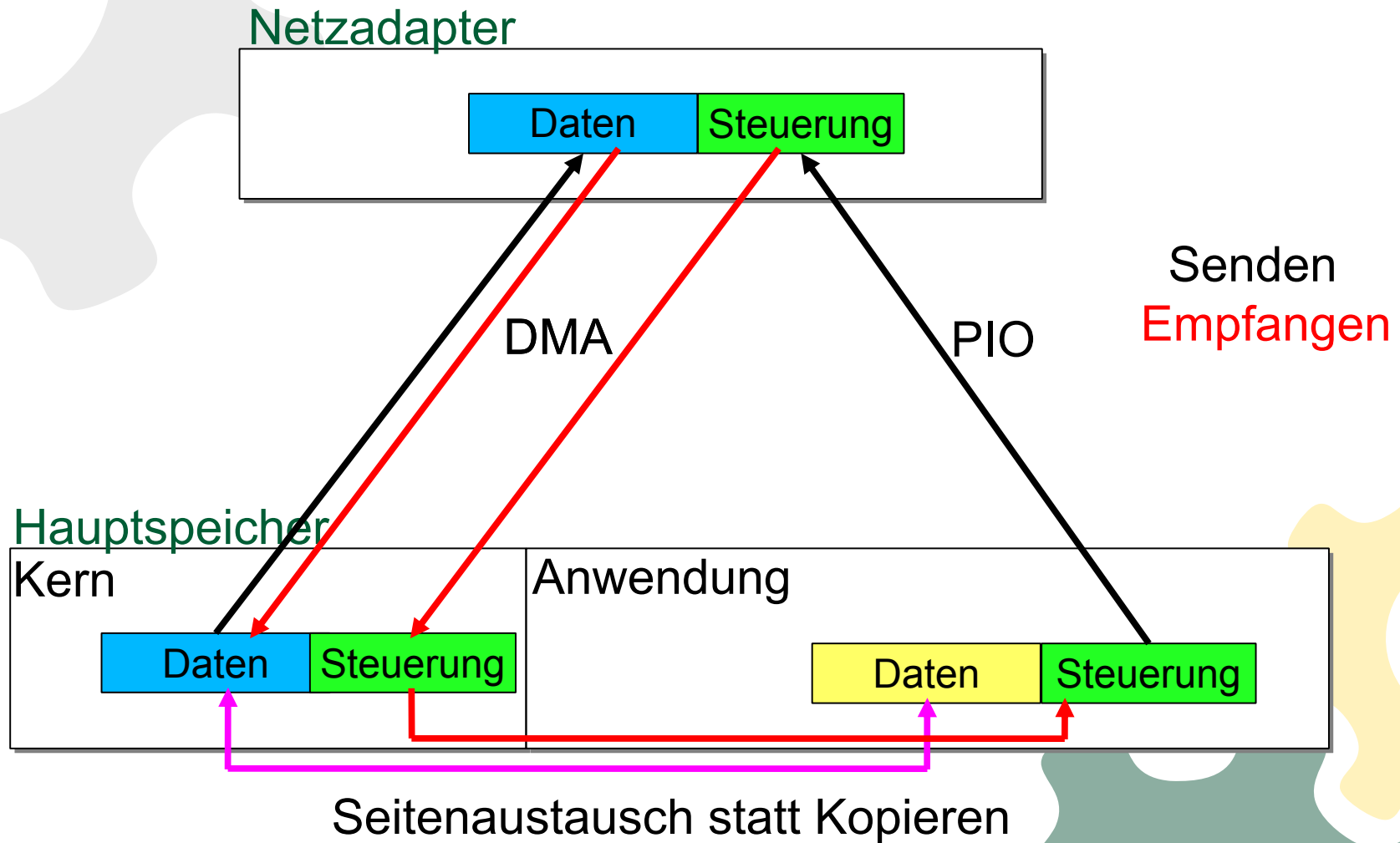


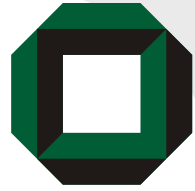
V2P-DMA (Bewertung)

- + Wenig CPU Belastung
- + Weniger Kopier-Operationen (als Kopier-DMA)
- + Dynamische Pufferdefinition möglich
- schwierige Semantik:
 - Das Senden kehrt schon zurück, wenn Daten noch zum Adapter unterwegs sind. Gegenmaßnahmen zur Pufferwiederverwendung nötig.
 - Sondermaßnahmen für Empfang nötig: Netzadapter muss signalisieren, wenn DMA komplett ist.
 - Zum Empfangen muss ein Puffer vorgeplant sein. Was geschieht, wenn kein Puffer angemeldet ist?
- Extreme Eingriffe in Speicherverwaltung (Festzurren von Speicherseiten). Schritthalten mit Kern-Release.



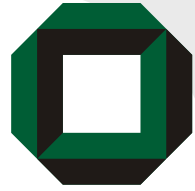
PEX (page exchange) DMA Seitenaustausch-DMA)





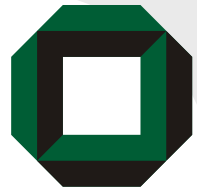
PEXDMA (Bewertung)

- + Kein Overhead
- + wenig CPU Belastung
- + definierte Semantik:
 - Senden kehrt erst dann zurück, wenn Daten sicher im Kern angekommen sind.
 - Empfangen wird erst dann ausgeführt, wenn Daten im Kern empfangen wurden.
- Extreme Eingriffe in die Speicherverwaltung (dynamisches Ändern der ATT's, dynamisches Festzurren / Entzurren von Speicherseiten;
- Daten müssen auf Seitengrenzen ausgerichtet sein, das sind sie aber in der Anwendung normalerweise nicht. (daher hat dieses Verfahren keine prakt. Bedeutung)



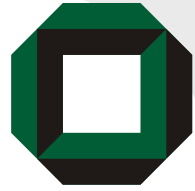
Durchsatzbeispiele

	Alpha 21164a 600 MHz	P-II 350 MHz
PIO	55,0 MB/s	50,0 MB/s
Kopier-DMA	43,0 MB/s	62,0 MB/s
Direktes DMA	67,0 MB/s	124,0 MB/s
V2P DMA	66,2 MB/s	120,5 MB/s
PEX DMA	66,2 MB/s	120,5 MB/s
Zeit f. Kerneintritt	1,5 μ s	1,8 μ s



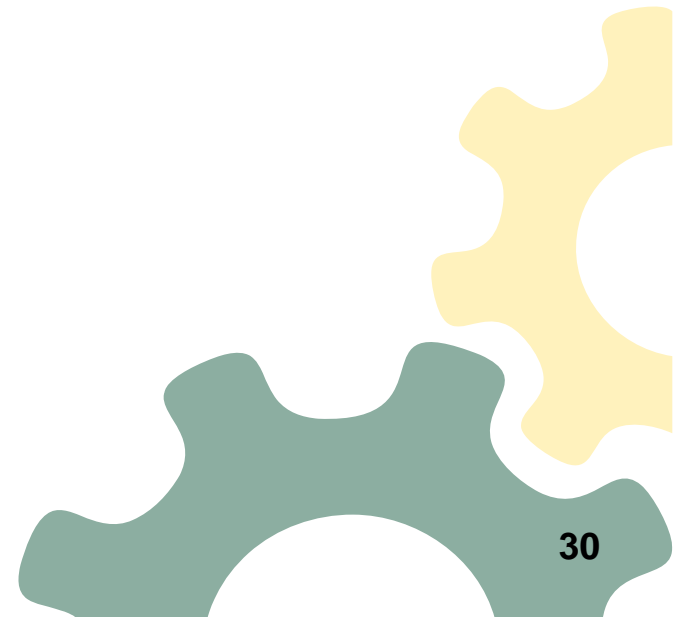
Zusammenfassung Optimierungstechniken

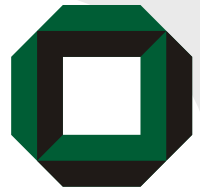
- Vereinfachte Protokolle
- Vermeidung von Systemaufrufen
- Schnelle Systemaufrufe
- Schnelle Unterbrechungsroutrinen
- Abfrage statt Unterbrechung
- Vermeidung von Kopieroperationen



Vorlesung „Rechnerbündel“

- Architektur von Rechnerbündeln
 - Hochgeschwindigkeitsnetzwerke
 - Hochgeschwindigkeitskommunikation
 - TCP/IP-UDP/IP-Probleme
 - Optimierungstechniken
 - Kommunikationsmodelle
 - Beispielsysteme

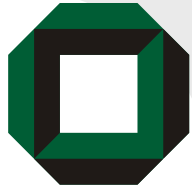




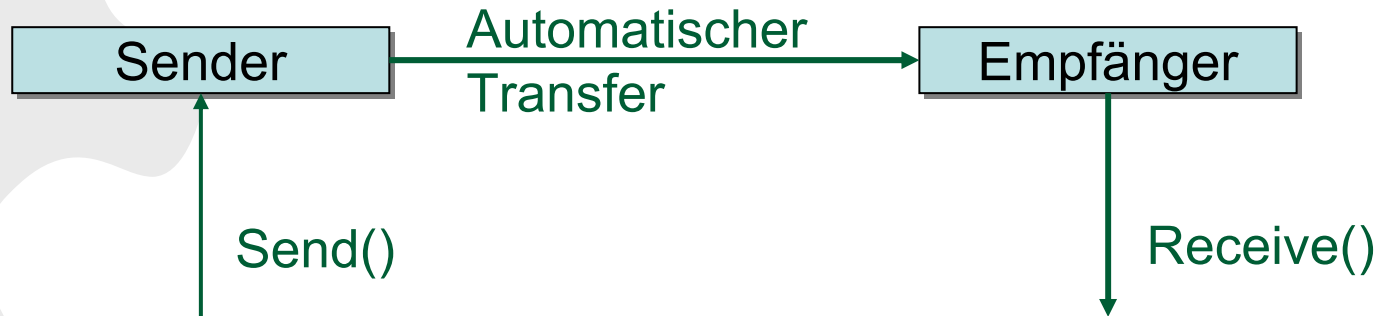
Grundlegende Kommunikationsparadigmen

- Klassisches Send/Receive
- Erweitertes Send/Receive
- (Virtueller) gemeinsamer Speicher, Shared Memory (SHM)
- Methodenfernaufruf, Remote Procedure Call (RPC)
- Active Messages (AM)
- Bulk Synchronous Parallel (BSP)

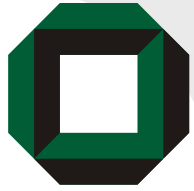
Je nach verwendetem Paradigma sind andere Optimierungen möglich (oder eben nicht).



Klassisches Send/Receive



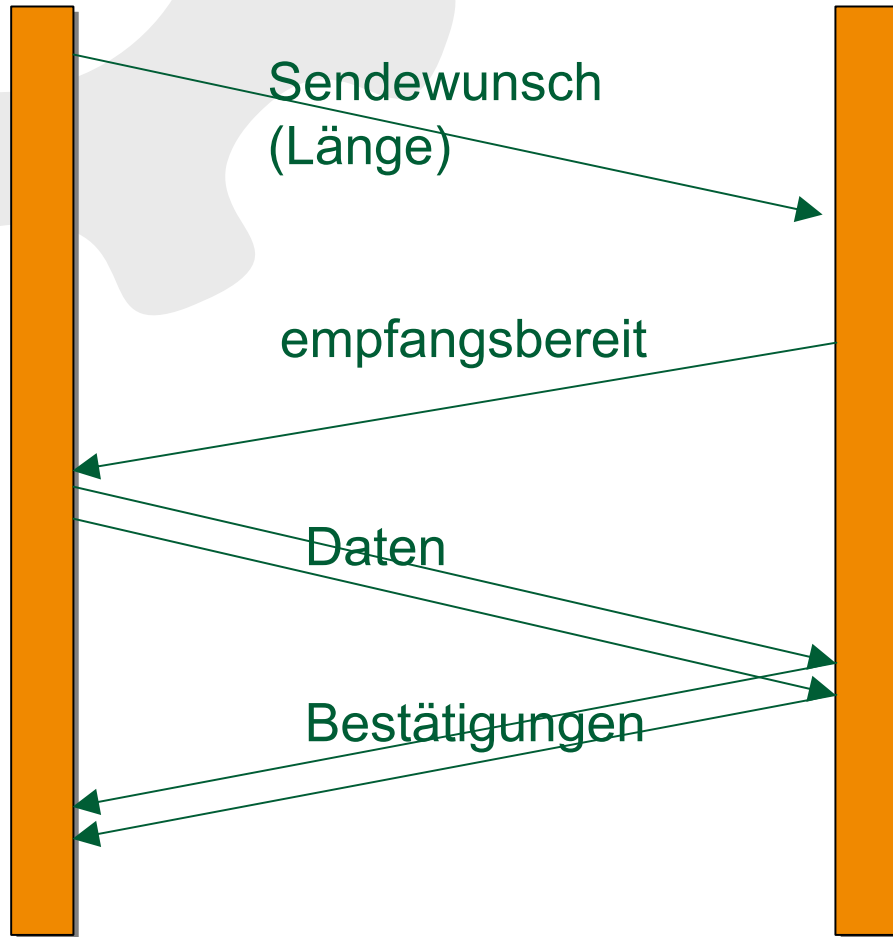
- Explizite Kommunikationsaufrufe
- Zweiseitige Kommunikation (Sender und Empfänger müssen beide aktiv werden.)
- Auch einseitige Variante: Sender kann Paket in Empfängerbereich schreiben ohne explizite Empfangsoperation (Lesen analog). Empfänger muss Bereich vorher für entfernten Zugriff freigeben.
- Synchrone und asynchrone Version
- Bei synchroner Version kann der Sender die Laufzeit der Kommunikation nicht anderweitig verwenden.
- Pufferung
- Vorwiegend nur Kommunikation, lediglich Rendezvous von Sender- und Empfängerprozess



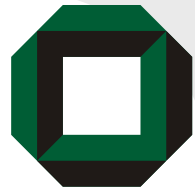
Synchrones Send/Receive: Einzelschritte

Sender

Empfänger

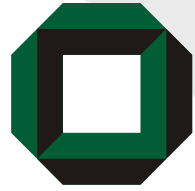


- Sender muss zwischenzeitlich warten.
- Keine Überlappung von Kommunikation und Berechnung möglich.



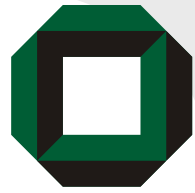
Zu bedenken

- Adressierung beim Sender: Endpunkt („Port“), Prozess, Knoten
- Pufferung beim Empfänger: Wie viele Daten können gesendet werden, bis zum ersten Mal Receive durchgeführt wird? Flusskontrolle: wie sorgt man dafür, dass der Sender wartet, wenn Puffer voll sind?
- Empfänger blockiert, falls keine Daten vorhanden. Soll Möglichkeit zum nicht-blockierenden Abfragen angeboten werden?
- Kann der Empfänger auf Pakete von bestimmten Sendern warten? Auf einen bestimmten Pakettyp?

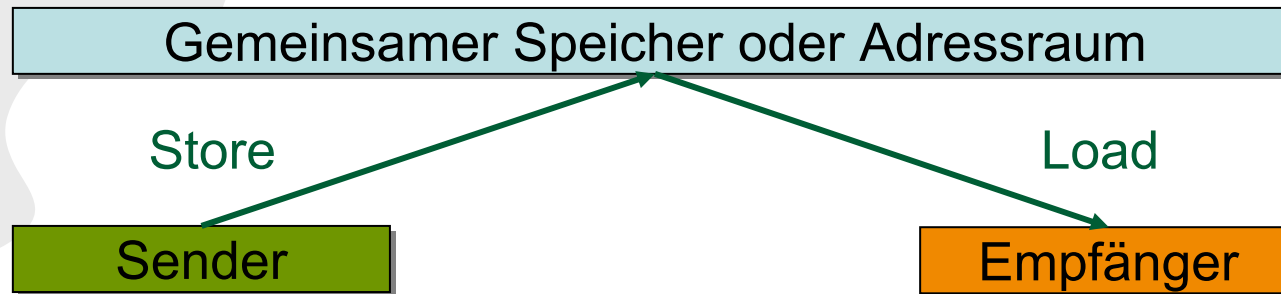


Erweitertes Send/Receive

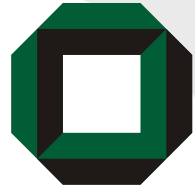
- Virtuelle Maschine aus logischen Knoten
 - Spezielle Gruppenkommunikationsoperationen
 - Rundruf (broadcast)
 - Synchronisation
 - Verteilen (scatter)
 - Einsammeln (gather)
- jeweils an alle Knoten
oder an Knotengruppe
- Bestandteil der MPI-Bibliothek.
 - Viel verwendet für wissenschaftliches Rechnern auf Rechnern mit verteiltem Speicher. Später mehr.



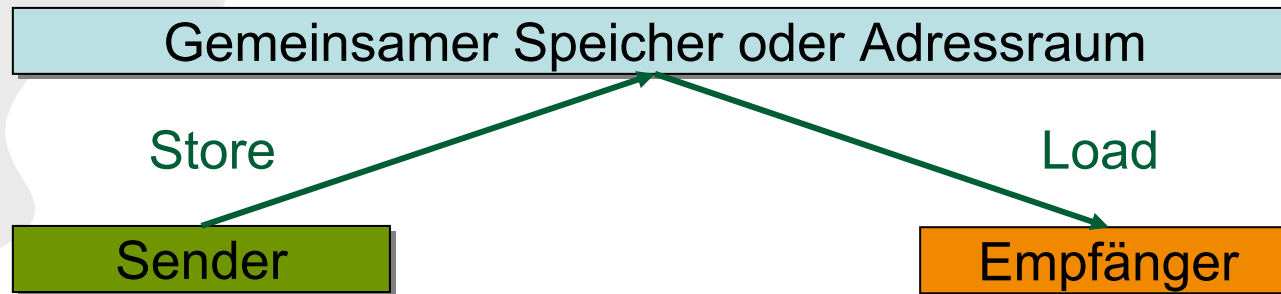
(Virtueller) Gemeinsamer Speicher, Shared Memory (SHM)



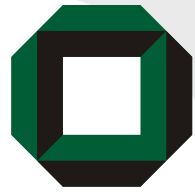
- Implizite Kommunikationsaufrufe
- Einseitige Kommunikation
- Synchronisationsprimitive (Semaphore, Mutex, ...) im Anwendungsprogramm nötig, um geordnete Zugriffe auf gemeinsam genutzte Speicherstellen zu gewährleisten.
- Konsistenzmodell (s.u.) für Kommunikationssystem wichtig.



(Virtueller) Gemeinsamer Speicher, Shared Memory (SHM)

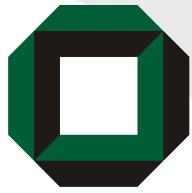


- Sender kann in der Regel weiterarbeiten, während Kommunikation zu Ende geführt wird.
- „reine“ Kommunikation: Sender und Empfänger laufen unabhängig.

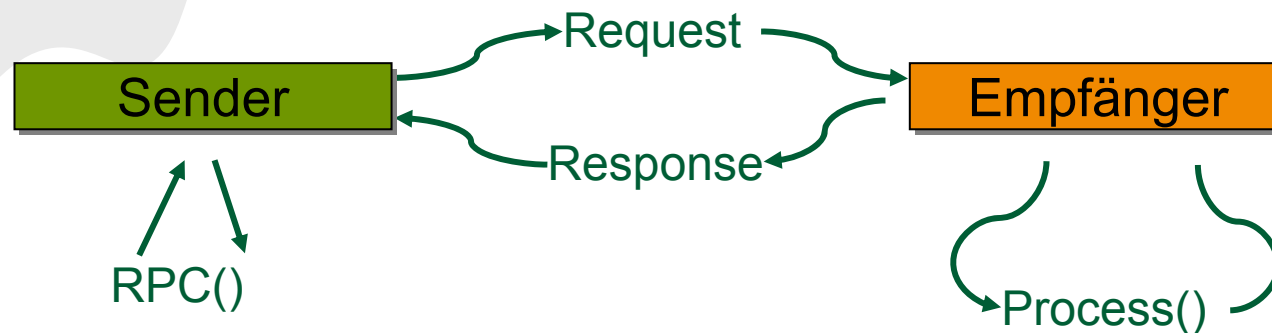


Zu bedenken

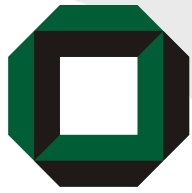
- Bei verteiltem Speicher muss empfängerseitig ein Vorrat an Kontrollfäden (Thread-Pool) vorgehalten werden, die die eingehende Pakete verarbeiten.
 - Diese Aufgabe kann auch von einer spezialisierten Hardware ausgeführt werden.
- Sync-Mechanismen/Monitore müssen effizient realisiert werden.
- Was passiert, wenn ein eingehendes Paket wegen einer Sperre nicht im Speicher abgelegt werden darf? Wie viel Pufferung ist möglich?
- Wie komfortabel/teuer soll das Cache-Konsistenz-Protokoll sein, das der Anwendung angeboten wird?



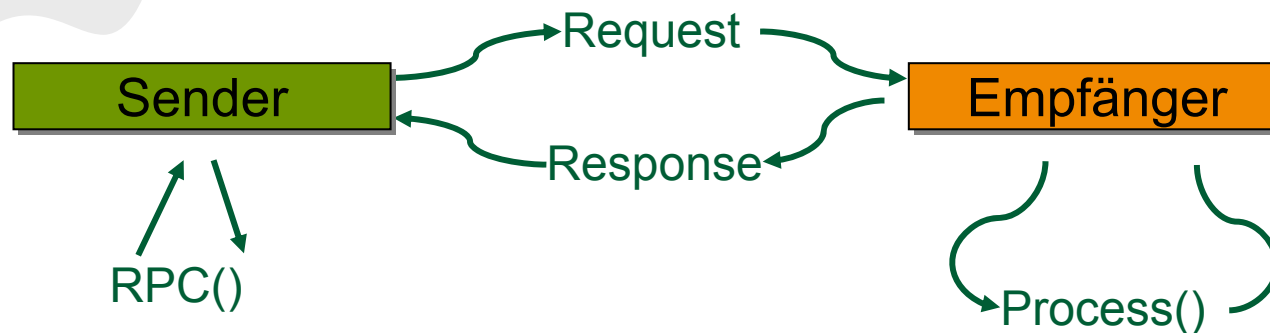
Methodenfernaufruf Remote Procedure Call (RPC)



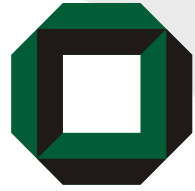
- Explizite Kommunikationsaufrufe
- Einseitige Kommunikation
- Bei synchroner Semantik wartet der Sender auf das Ergebnis. Es ist keine Parallelität möglich.
- Auch: asynchrone Variante
- Empfänger muss Kontrollfäden vorhalten, die mit jedem Aufruf (Methoden-Nummer, Parameter-Codierung) umgehen können.
- Zeiger können nur mit besonderen Maßnahmen gesendet werden („globaler Zeiger“, Fernzugriff).



Methodenfernaufruf Remote Procedure Call (RPC)

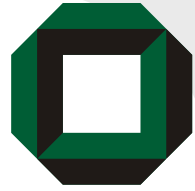


- Bei kurzen Methoden kann Sender ggf. aktiv auf die Antwort warten statt „sich schlafen zu legen“, d.h. die Kontrolle an das Betriebssystem zurückzugeben. Dies spart die Zeit für zwei Kontextwechsel, und ermöglicht die volle Ausnutzung der dem Sender zur Verfügung stehenden Zeitscheibe.
- Mit den kommunizierten Daten wechselt auch der Programmablauf vom Sender-Rechner auf den Empfänger-Rechner.

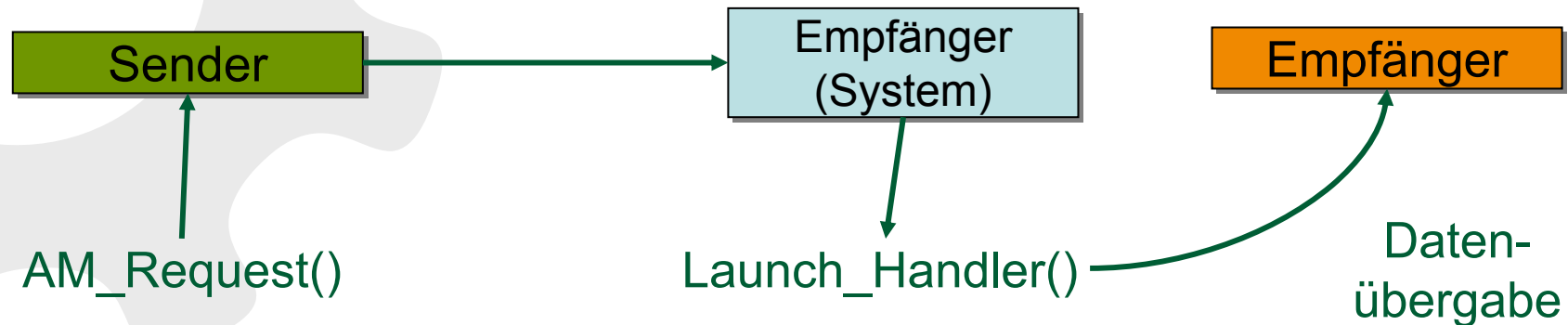


Zu bedenken

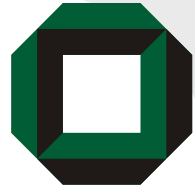
- Methodenadressierung: Funktionszeiger oder Tabelle?
- Im objektorientierten Bereich: Proxy-Objekte, die den entfernten Aufruf wie einen lokalen aussehen lassen.
- Leitungsprotokoll (Serialisierung) für Parameter.
- Empfänger muss einen Vorrat an Kontrollfäden vorhalten, die Pakete entgegen nehmen, dekodieren und dann die Funktion im Benutzerbereich aufrufen.
- Pool-Größe? Zu kleiner Pool führt dazu, dass nicht alle Aufrufe abgearbeitet werden können. Zu großer Pool = Ressourcen-Vergeudung.
- Wechsel der Kontrollfadenidentität kann (z.B. in Java) Probleme bei maschinenübergreifender Synchronisierung machen.
 - A wird gesperrt durch Monitor. Dann Fernaufruf an B. B ruft zurück an A, und sollte Zugriff erhalten, hat aber andere Identität (nämlich die eines der Kontrollfäden aus dem Empfangspool).



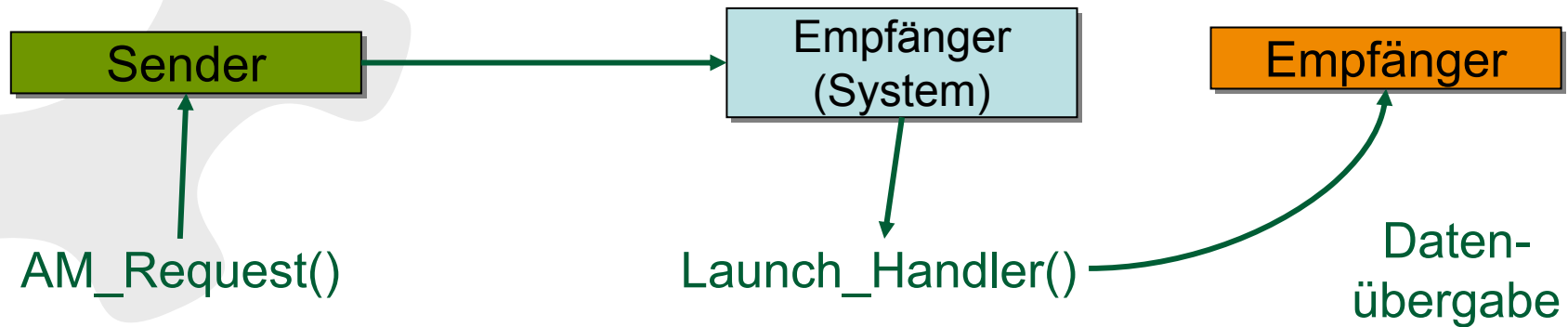
Active Messages (AM)



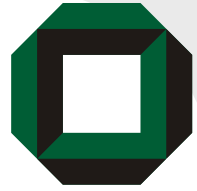
- Grundidee: Jeder Paketkopf enthält die Adresse eines kurzen Empfangsprogramms (Handler), das beim Empfänger im Benutzerbereich ausgeführt wird. (Ähnlich einer Unterbrechung, jedoch unprivilegiert.)
- Empfangsprogramm
 - wird durch Unterbrechung oder durch Abfrage gestartet.
 - liest die Paketdaten aus dem Netz und speichert sie in den Datenstrukturen der Anwendung.
- Explizite Kommunikationsaufrufe
- Einseitige Kommunikation



Active Messages (AM) (2)

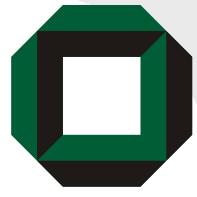


- Implizite Kommunikation auf Empfängerseite. Daten werden automatisch in die Datenstrukturen eingearbeitet.
- Empfangsprogramm muss Synchronisationsprimitive verwenden, um geordneten Zugriff auf Anwendungsdaten zu gewährleisten.
- Der Sender kann in der Regel weiterarbeiten, während Kommunikation zu Ende geführt wird.



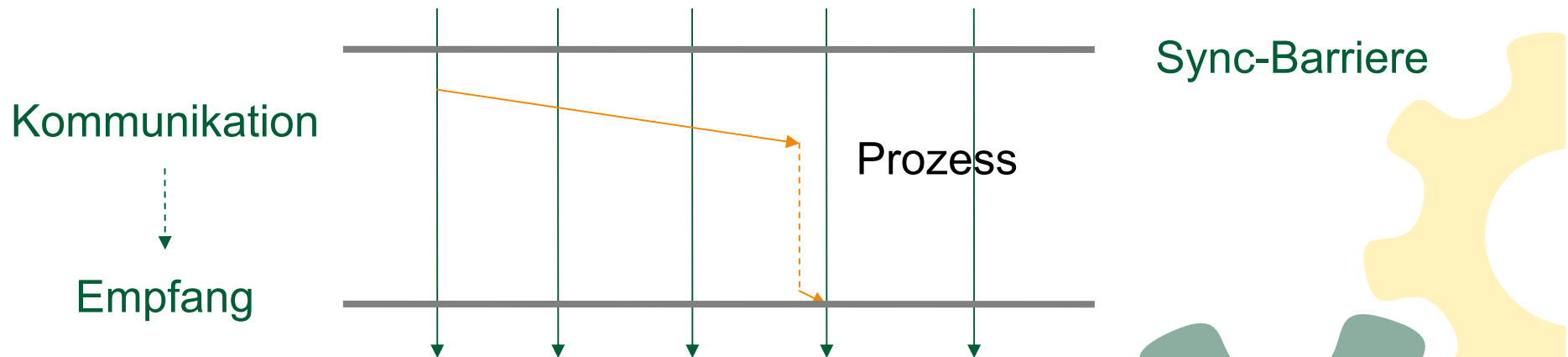
Zu bedenken

- Kleinerer Kontrollfaden-Pool.
- Benutzer-Funktionen greifen auf Netzwerk-Karte zu. Sicherheit/Schutz?
- Wie stellt man sicher, dass Benutzer-Funktionen die Pakete auch abnehmen? Sonst könnte die Kommunikation blockieren.
- Adressierung der Methoden: Funktionszeiger oder Tabelle



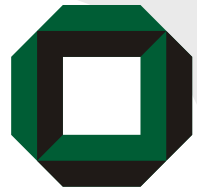
Bulk Synchronous Parallel (BSP)

- Ein BSP-Programm ist in Superschritte (**Supersteps**) gegliedert.
- In einem Superschritt rechnen die Prozessoren und stoßen Kommunikation an. Der Empfang wird aber erst zum Ende des Superschritts garantiert.
- Dazu am Superschritt-Ende eine Synchronisationsbarriere (für alle oder eine Gruppe der Prozessoren.) D.h., es wird auf die Ankunft aller gesendeten Botschaften gewartet.





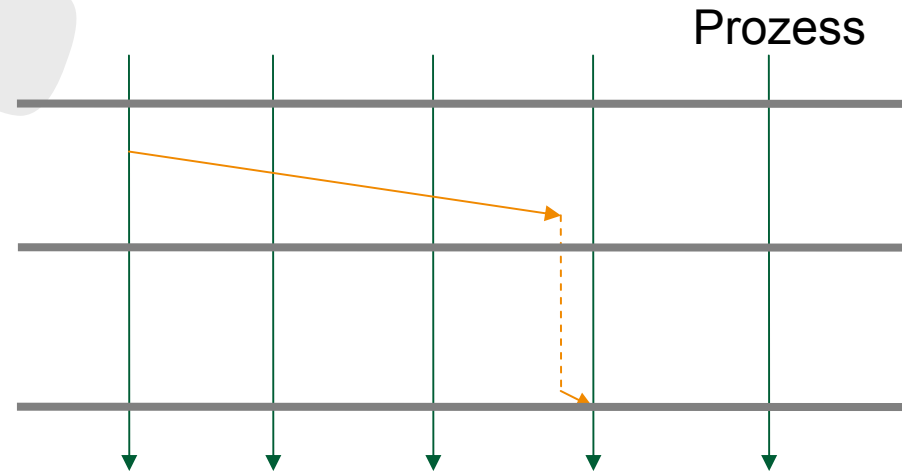
- BSP: Bulk Synchronous Parallel Computing
- Theoretischer Hintergrund
 - Die PRAM-Annahme, dass eine Kommunikationsoperation genau eine Zeiteinheit dauert ist unrealistisch.
 - **ABER:** die meisten Netzwerke können parallel zu den Prozessoren arbeiten.
 - d.h. Superschritte werden möglichst so konstruiert, dass sie gerade so viel Kommunikationszeit benötigen, wie für die Rechnung nötig ist.



Bulk Synchronous Parallel (BSP)

Kommunikation

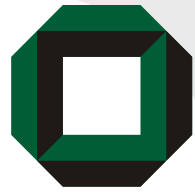
Empfang



Sync-Barriere

Nächste Sync-Barriere
besser schon hier!

- „reine“ Kommunikation. Sender und Empfänger laufen unabhängig (bis auf regelmäßige Synchronisationen.)



Zu bedenken

- Effiziente Synchronisierung nötig (per Baum).
- Pufferung der gesamten Kommunikation eines Superschritts erforderlich.
- Der Sender kann/soll (!) weiterarbeiten, während die Kommunikation zu Ende geführt wird.
- Effizienzeinbußen durch zu späte Synchronisationsbarrieren. Vermeidbar, falls die Rechenschritte sofort nach ihrem Abschluss selbst die Synchronisation starten.