
Rechnernetze II

SoSe 2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 14. Juli 2020

Rechnernetze II

SoSe 2020

9 Leistungssteigerung von Netzen



Inhalt

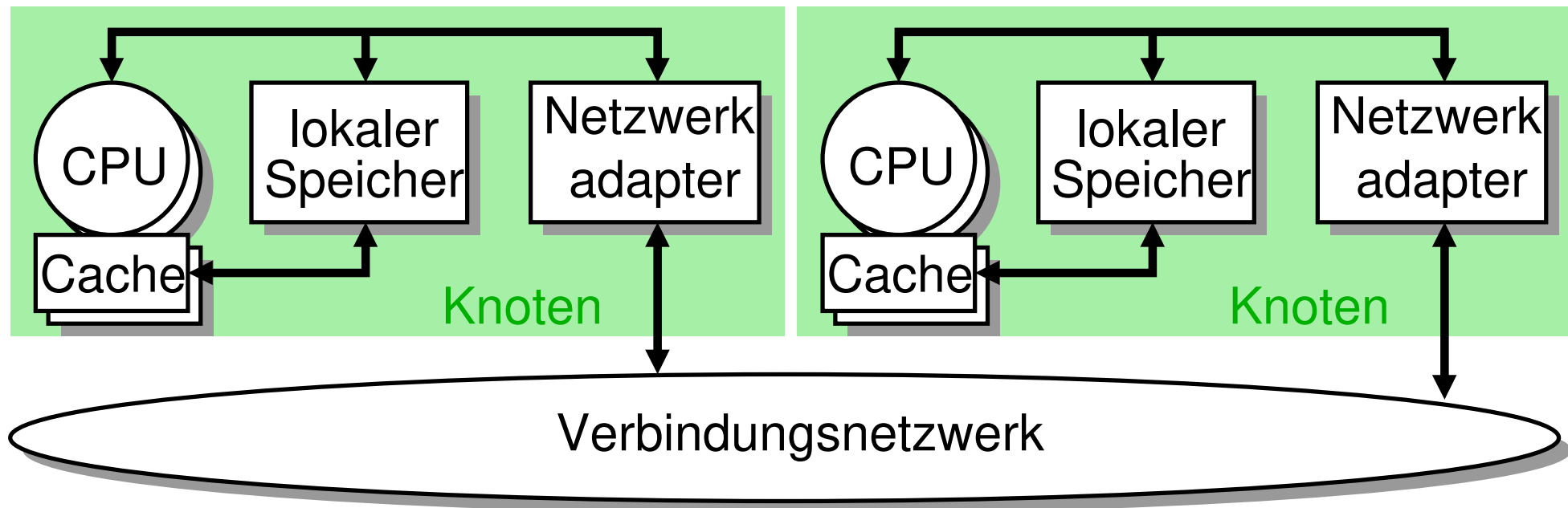
- ➔ Motivation: Hochleistungsrechner
- ➔ Maßnahmen zur Leistungssteigerung
- ➔ Beispiel: Infiniband

- ➔ Harald Richter: Verbindungsnetzwerke für parallele und verteilte Systeme. Spektrum Akademischer Verlag, 1997
- ➔ Weitere Angaben auf der WWW-Seite

Hochleistungsrechner

- ➔ Für rechenintensive Anwendungen
 - ➔ Klimasilimulation, Genomanalyse, Arzneimittel-Design, ...
- ➔ Als Server
 - ➔ WWW (Suchmaschinen), Datenbanken, ...
- ➔ Multiprozessorsysteme: mehrere gekoppelte Prozessoren
 - ➔ mit gemeinsamem Speicher (SMP, ccNUMA)
 - ➔ mit verteiltem Speicher (MPP)
- ➔ Beispiel: Fugaku (Japan, schnellster Rechner der Welt!)
 - ➔ 7.299.072 CPU-Kerne
 - ➔ 415 TFlop/s Rechenleistung

Multiprozessorsysteme mit verteiltem Speicher (MPPs)



- ➔ Sehr gut skalierbar (bis mehrere 100000 Knoten)
- ➔ Kommunikation/Synchronisation durch Nachrichtenaustausch
- ➔ Architektur entspricht Cluster von Rechnern



Programmierung von MPPs

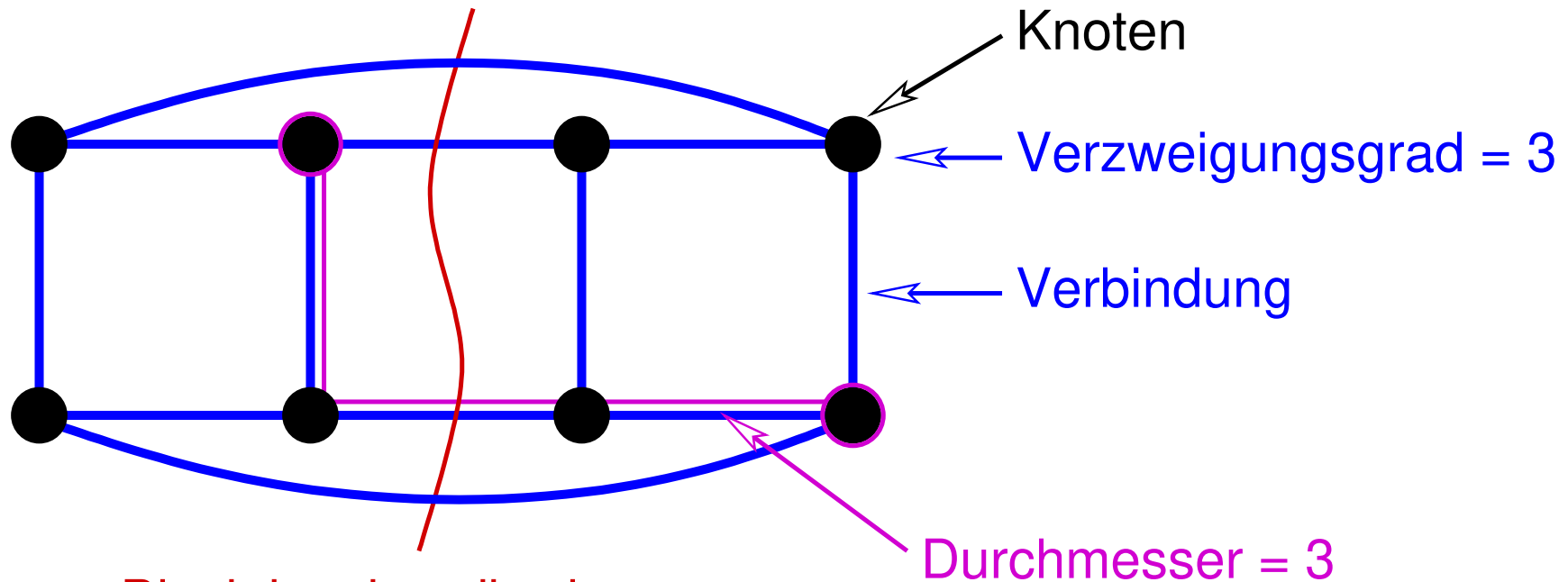
- ➔ Parallele Prozesse (meist ein Prozeß pro Knoten)
 - ➔ jeder Prozeß bearbeitet ein Teilproblem
- ➔ Kooperation durch Nachrichtenaustausch
 - ➔ Punkt-zu-Punkt: *send*, *receive*
 - ➔ global: *broadcast* (1-zu-N), *reduce* (N-zu-1), ...
 - ➔ oft sehr große Nachrichten
- ➔ Wichtig für hohe Rechenleistung:
 - ➔ Kommunikationszeit \ll Rechenzeit
 - ➔ Überlappung von Kommunikation und Berechnung
 - ➔ gleichzeitige Kommunikation vieler Knoten



Maßgebliche Eigenschaften der Verbindungsnetze

- ➔ Routing: i.a. mehrere mögliche Pfade
 - ➔ **Blockierungsfreiheit**: Kommunikation zweier Knoten blockiert andere Kommunikationen nicht
 - ➔ Ausfallsicherheit
- ➔ **Bisektionsbandbreite**: Gesamtbandbreite, wenn eine Hälfte der Knoten an die anderen sendet (*worst case*)
- ➔ **Durchmesser** des Netzes: Pfadlänge zwischen zwei maximal entfernten Knoten
- ➔ **Verbindungsgrad** eines Knotens: Anzahl der Links
- ➔ **Skalierbarkeit**: „Eigenschaften des Netzes bleiben bei Erweiterung erhalten“

Beispiel



Bisektionsbandbreite =
4 * Bandbreite der Einzelverbindungen



➔ Ziele:

- ➔ hohe Bisektionsbandbreite
- ➔ niedrige Latenz
 - ➔ schnelle Weiterleitung in den Zwischenknoten
 - ➔ geringe Latenz durch Softwareschichten

➔ Mechanismen:

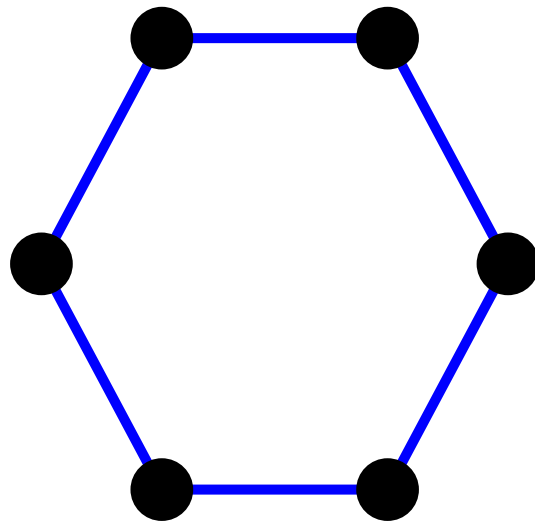
- ➔ Netztopologien
- ➔ Weiterleitungstechniken
- ➔ Protokolle der Anwendungsschicht
 - ➔ Vermeidung von Kopiervorgängen
- ➔ *Remote DMA* und *OS bypass*
 - ➔ Vermeidung von Betriebssystem-Einsprünge



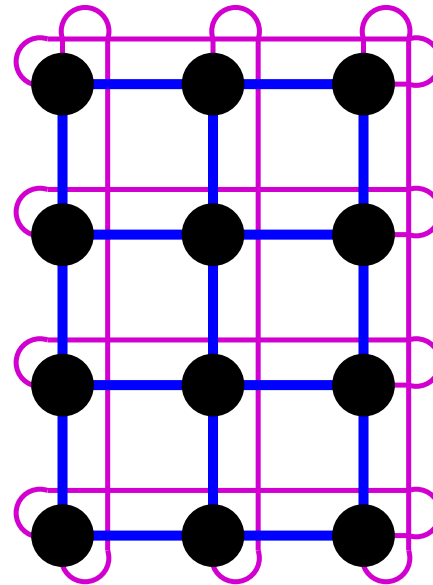
9.2.1 Netztopologien

- ➔ „Statische“ Verbindungsnetze
 - ➔ direkte Verbindungen zwischen Paaren von Knoten
 - ➔ Forwarding erfolgt durch in die Knoten eingebaute Switches
 - ➔ z.B. Ring, Gitter, Hyperwürfel
- ➔ Dynamische Verbindungsnetze
 - ➔ Knoten sind indirekt über einen oder mehrere Switches verbunden
 - ➔ z.B. Kreuzschienenverteiler, Clos-Netz

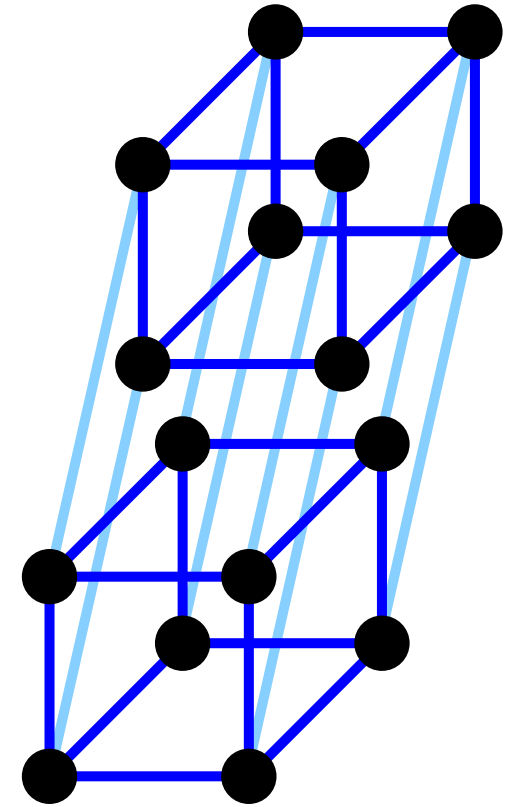
Statische Verbindungsnetze: Beispiele



Ring



Gitter / Torus



4D Hyperwürfel

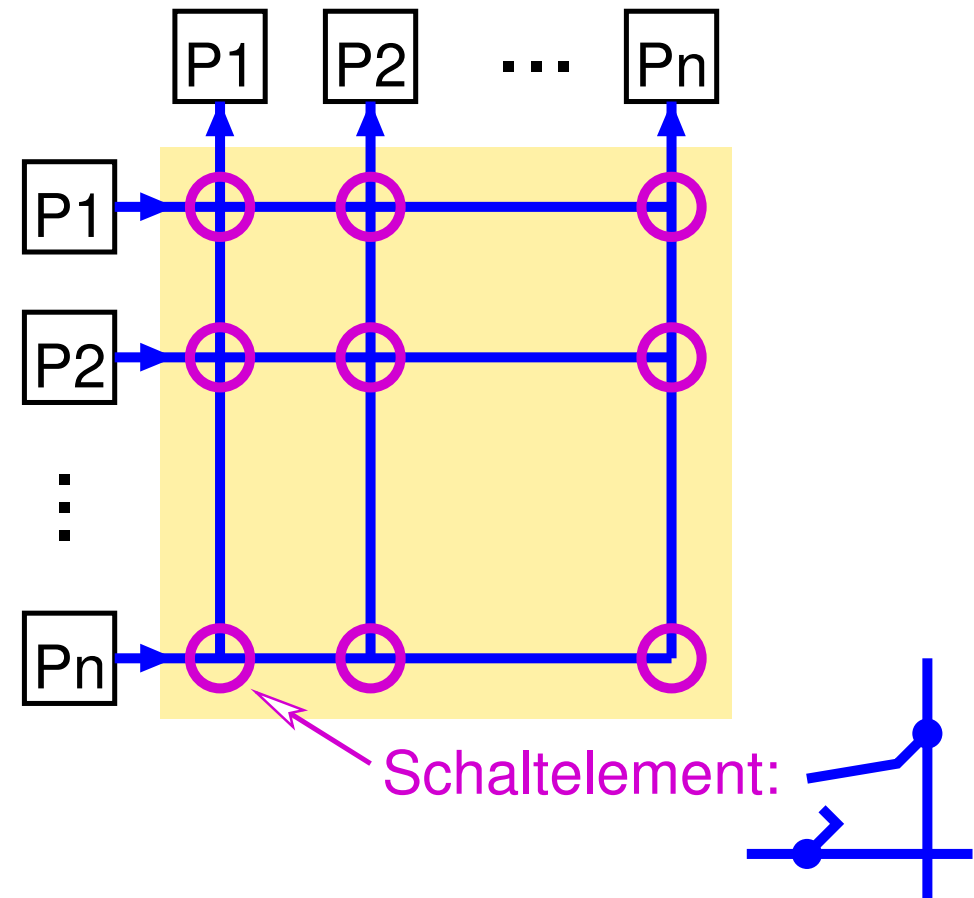
Durchmesser $O(N)$
Verzweigungsgrad 2
Bisektionsbandbr. $2 * \text{Link}$

$O(\sqrt{N})$
4
 $O(\sqrt{N}) * \text{Link}$

$O(\log(N))$
 $O(\log(N))$
 $O(N) * \text{Link}$

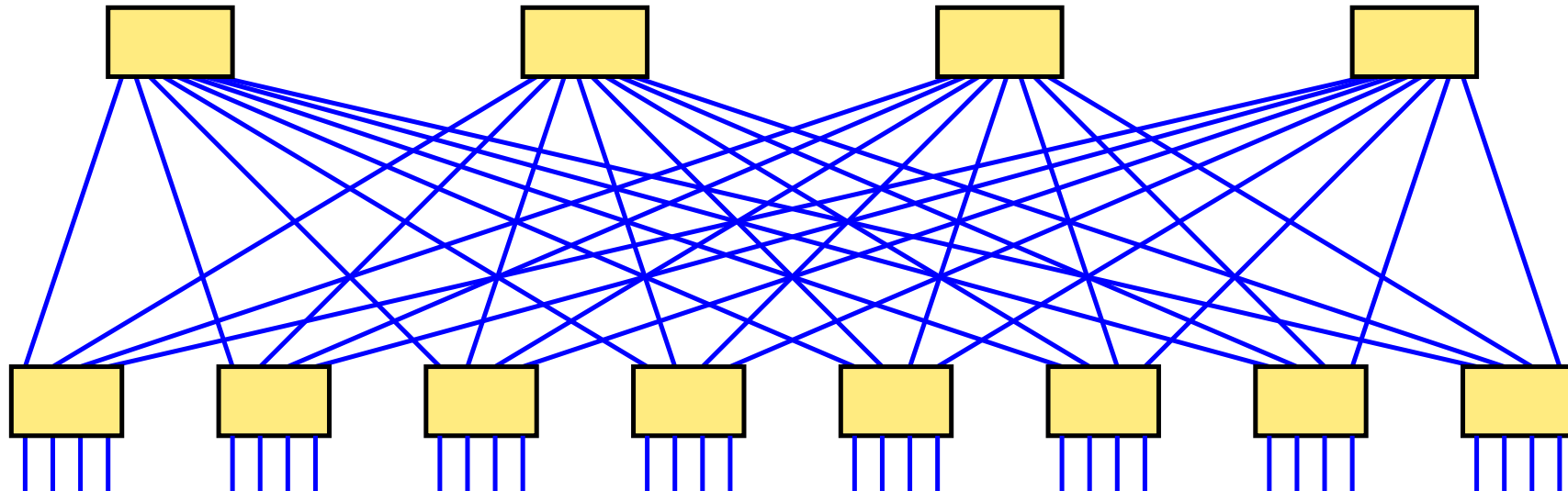
Dynamische Verbindungsnetze: Kreuzschienenverteiler (*Crossbar Switch*)

- ➔ Knoten hat getrennte Links pro Richtung
- ➔ Alle möglichen, disjunkten Knotenpaare können gleichzeitig verbunden werden
 - ➔ blockierungsfreies Netz
- ➔ Hoher Hardwareaufwand: n^2 Schaltelemente bei n Knoten
- ➔ I.a. nicht erweiterbar



Dynamische Verbindungsnetze: Clos-Netz

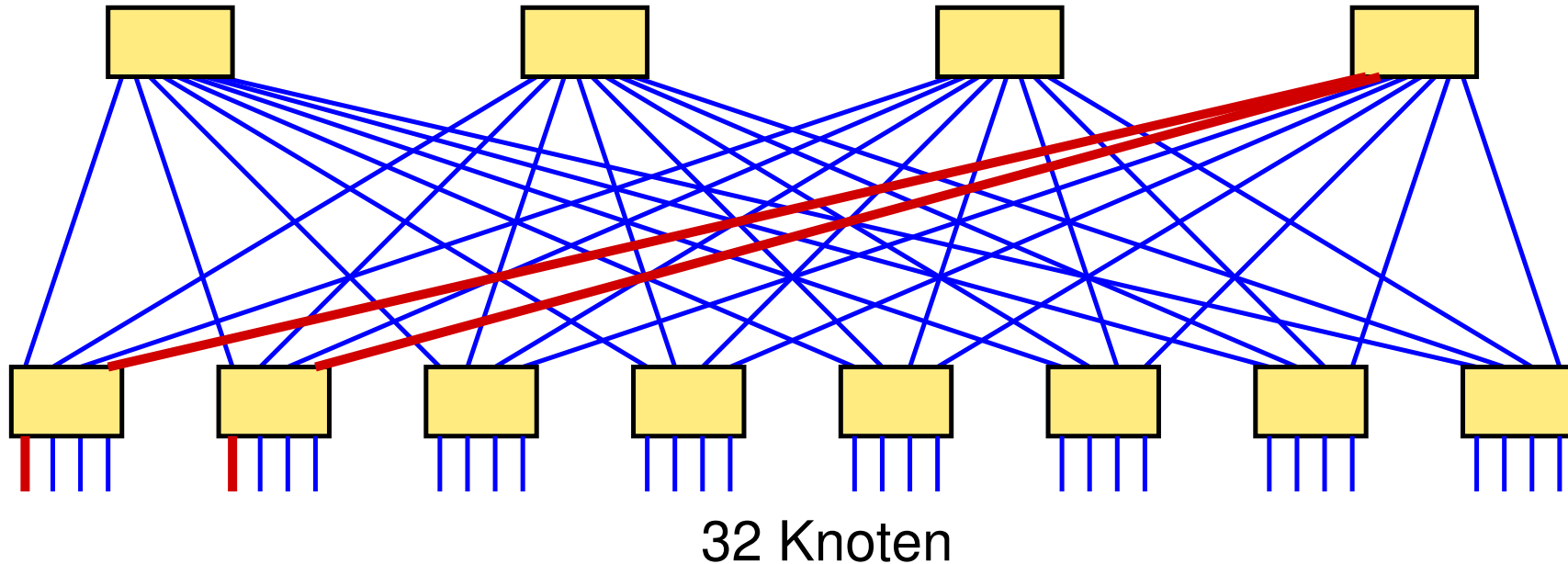
- ➔ Erweiterbares Netz auf Basis (kleiner) Kreuzschienenverteiler
 - ➔ maximal mögliche Bisektionsbandbreite
 - ➔ blockierungsfrei, wenn ex. Routen geändert werden dürfen
- ➔ Beispiel: Vernetzung 32 Knoten mit 8x8 Crossbars



32 Knoten

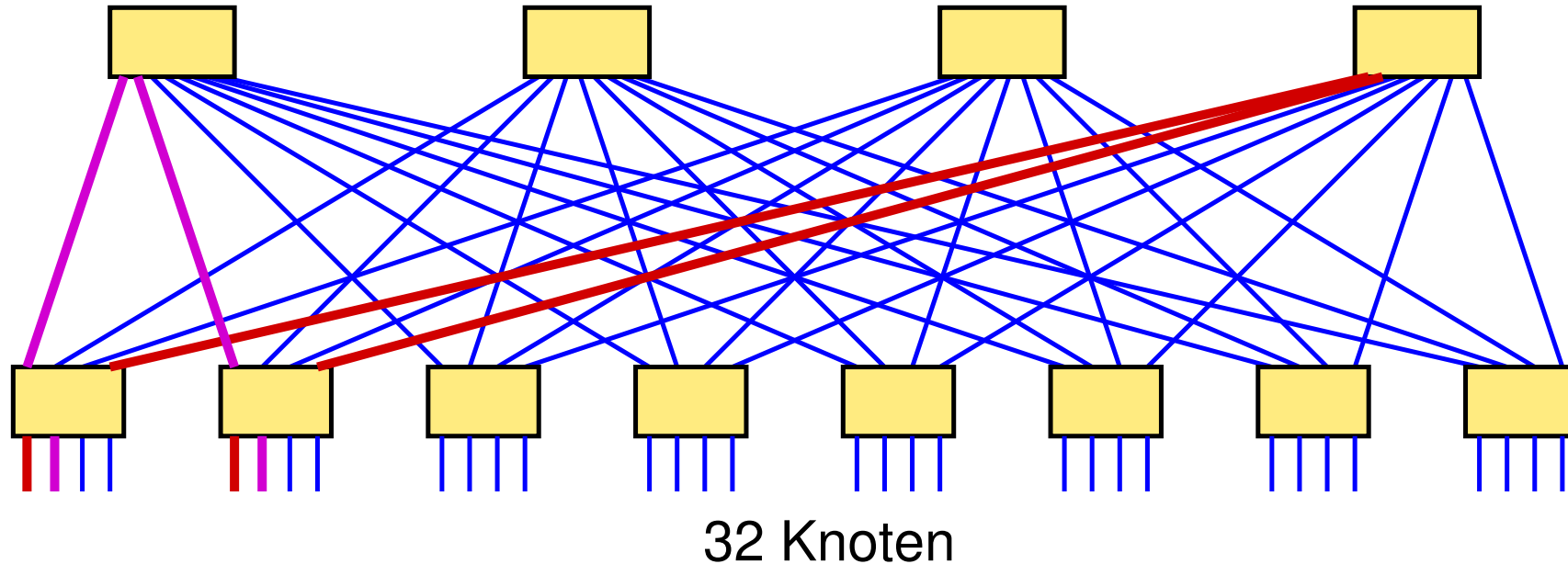
Dynamische Verbindungsnetze: Clos-Netz

- ➔ Erweiterbares Netz auf Basis (kleiner) Kreuzschienenverteiler
 - ➔ maximal mögliche Bisektionsbandbreite
 - ➔ blockierungsfrei, wenn ex. Routen geändert werden dürfen
- ➔ Beispiel: Vernetzung 32 Knoten mit 8x8 Crossbars



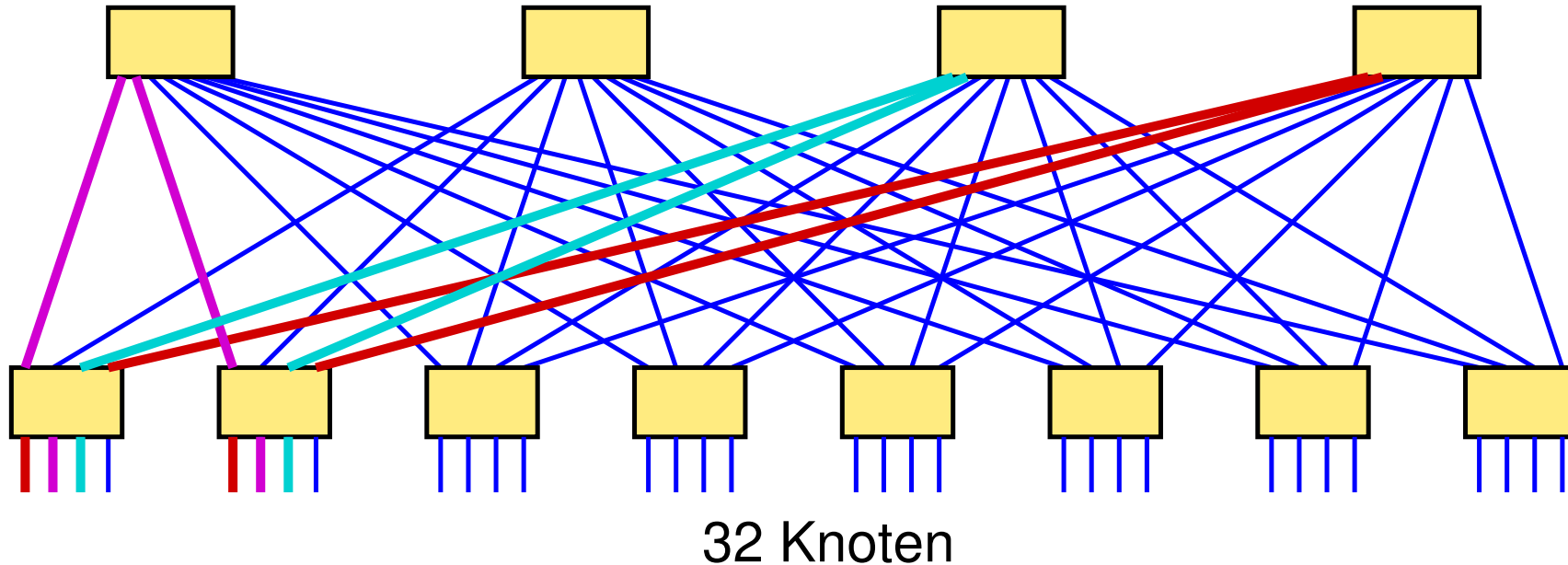
Dynamische Verbindungsnetze: Clos-Netz

- ➔ Erweiterbares Netz auf Basis (kleiner) Kreuzschienenverteiler
 - ➔ maximal mögliche Bisektionsbandbreite
 - ➔ blockierungsfrei, wenn ex. Routen geändert werden dürfen
- ➔ Beispiel: Vernetzung 32 Knoten mit 8x8 Crossbars



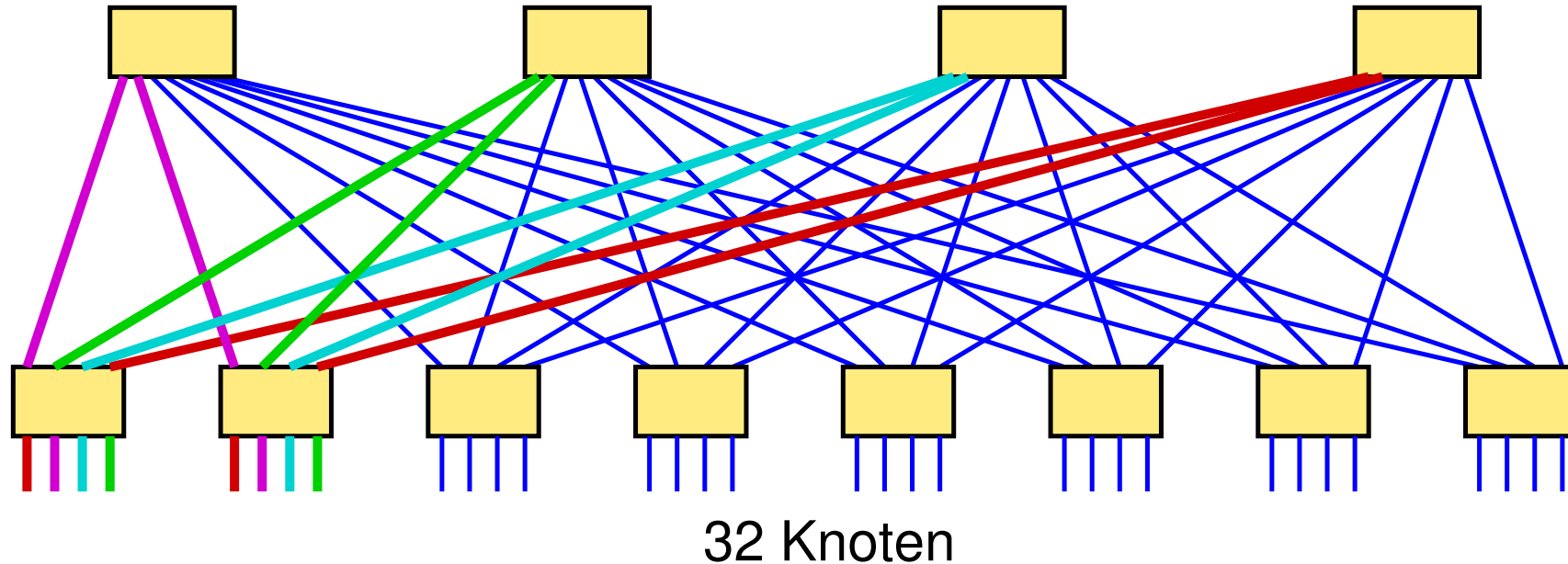
Dynamische Verbindungsnetze: Clos-Netz

- ➔ Erweiterbares Netz auf Basis (kleiner) Kreuzschienenverteiler
 - ➔ maximal mögliche Bisektionsbandbreite
 - ➔ blockierungsfrei, wenn ex. Routen geändert werden dürfen
- ➔ Beispiel: Vernetzung 32 Knoten mit 8x8 Crossbars



Dynamische Verbindungsnetze: Clos-Netz

- ➔ Erweiterbares Netz auf Basis (kleiner) Kreuzschienenverteiler
 - ➔ maximal mögliche Bisektionsbandbreite
 - ➔ blockierungsfrei, wenn ex. Routen geändert werden dürfen
- ➔ Beispiel: Vernetzung 32 Knoten mit 8x8 Crossbars





9.2.2 Weiterleitungstechniken

- ➔ Ziel: Schnelle Weiterleitung in den Zwischenknoten
- ➔ Randbedingung: Netze mit hoher Bandbreite und großen Paketen
 - ➔ hoher Bedarf an Pufferplatz in den Switches
 - ➔ daher ggf. Flusskontrolle zwischen Switches
- ➔ Randbedingung: mehrere mögliche Wege im Netz
 - ➔ schnelle (einfache) Routing-Verfahren erforderlich
- ➔ Trend: Zentralisierung der Steuerung des Netzverkehrs
 - ➔ *Software Defined Networking*



➔ *Store-and-Forward Routing*

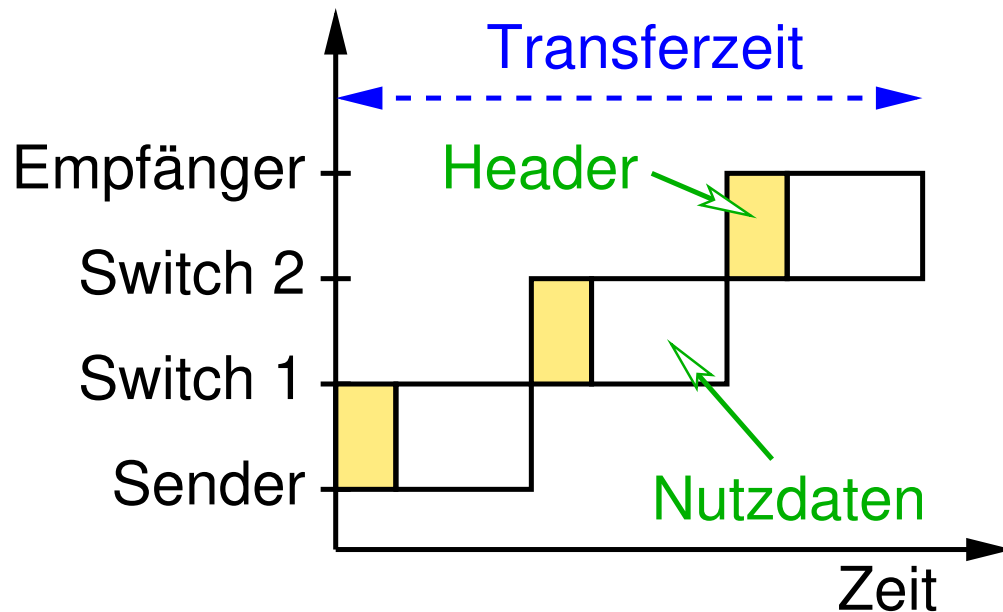
- ➔ Switch empfängt Paket vollständig, analysiert Header, gibt Paket an entsprechenden Ausgangsport weiter
- ➔ Puffer für mindestens ein Paket notwendig
- ➔ paketorientierte Flußkontrolle
- ➔ Problem: hohe Latenz

➔ *Virtual-Cut-Through Routing*

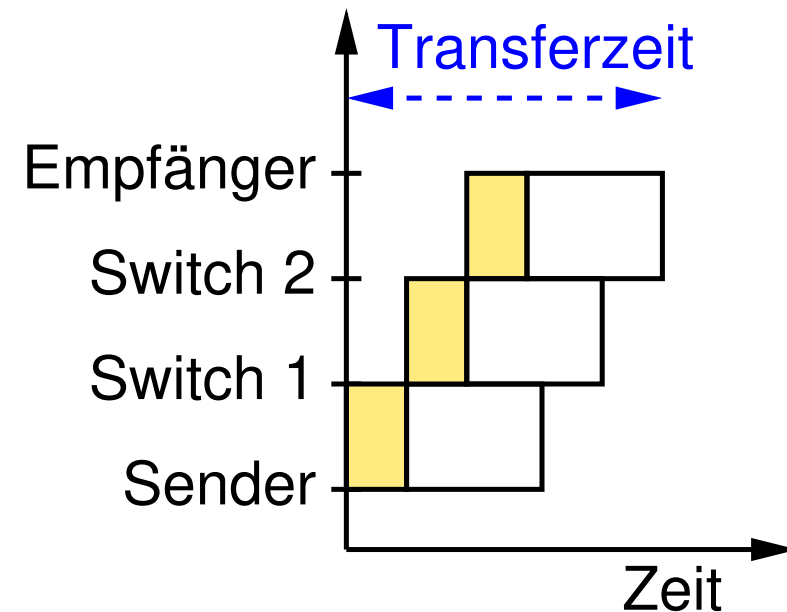
- ➔ Switch schaltet Weg zum passenden Ausgangsport bereits nach Empfang des Headers durch
 - ➔ benötigt schnelle Forwarding-Logik
- ➔ falls Ausgangsport belegt: Puffern des gesamten Pakets
- ➔ paketorientierte Flußkontrolle

Vergleich von *Store-and-Forward* und *Virtual-Cut-Through*

Store-and-Forward-Routing



Virtual-Cut-Through-Routing



➔ Anmerkung: auch schnelle Ethernet-Switches verwenden heute *Virtual-Cut-Through*



➔ ***Wormhole-Routing***

- ➔ Weiterleitung wie bei *Virtual-Cut-Through*
- ➔ feingranulare Flußkontrolle auf Bitübertragungsebene
 - ➔ Flits (*flow control digits*), typ. 1-2 Bytes
- ➔ in Switches nur Puffer für wenige Flits (abhängig von RTT)
- ➔ bei belegtem Ausgangsport: Rückstau des Pakets
 - ➔ Flits enthalten keine Zieladresse \Rightarrow Links bleiben blockiert
- ➔ Vorteil: geringe Latenz, wenig Pufferbedarf
- ➔ Nachteil: (größere) Gefahr von Deadlocks
 - ➔ Vermeidung durch entsprechende Routing-Algorithmen



Software Defined Networking (SDN)

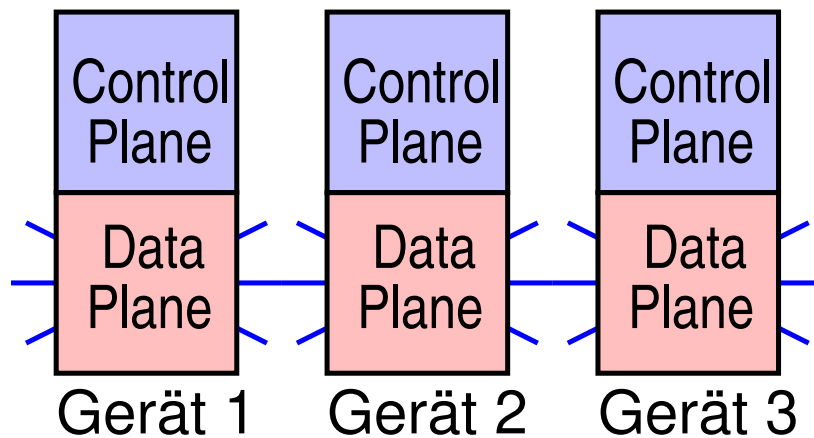
- ➔ Schichtenaufbau eines Weiterleitungsknotens:
 - ➔ *Data Plane*: regelbasierte, tabellengesteuerte Weiterleitung von Frames/Paketen
 - ➔ *Control Plane*: Erstellung der Regeln bzw. Weiterleitungstabellen (z.B. durch Routing-Protokolle)

- ➔ Grundideen von SDN:
 - ➔ Trennung von *Data Plane* und *Control Plane*
 - ➔ Zentralisierung der *Control Plane*
 - ➔ Kommunikation zw. zentralem Controller und Weiterleitungsknoten über das Netzwerk
 - ➔ Protokoll z.B. OpenFlow (über TLS / TCP)

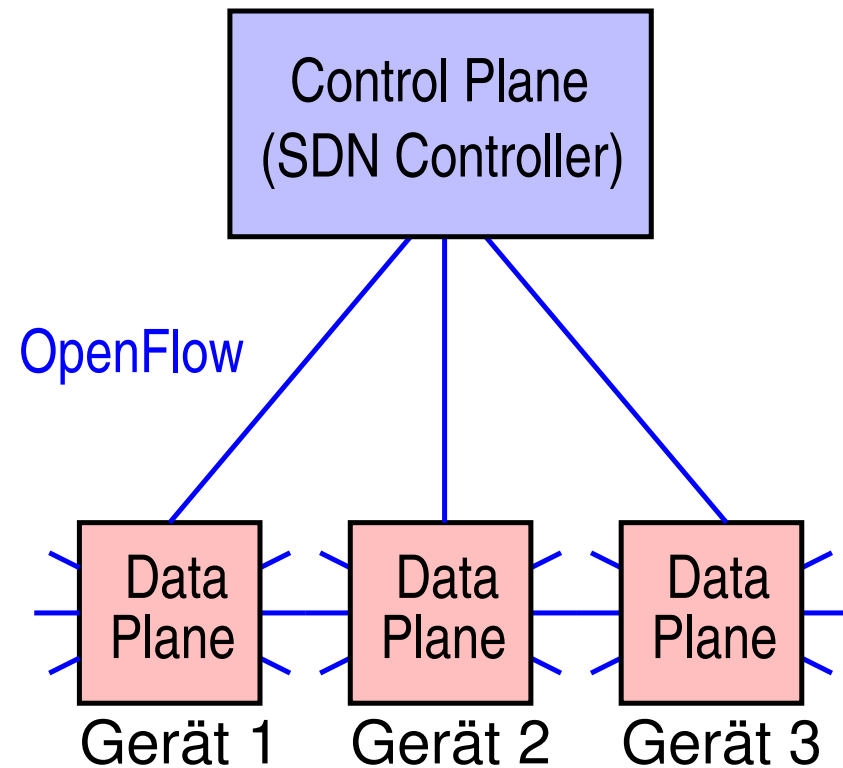
Software Defined Networking (SDN) ...

➔ Vergleich der Architekturen:

Übliche Netzarchitektur



SDN Architektur



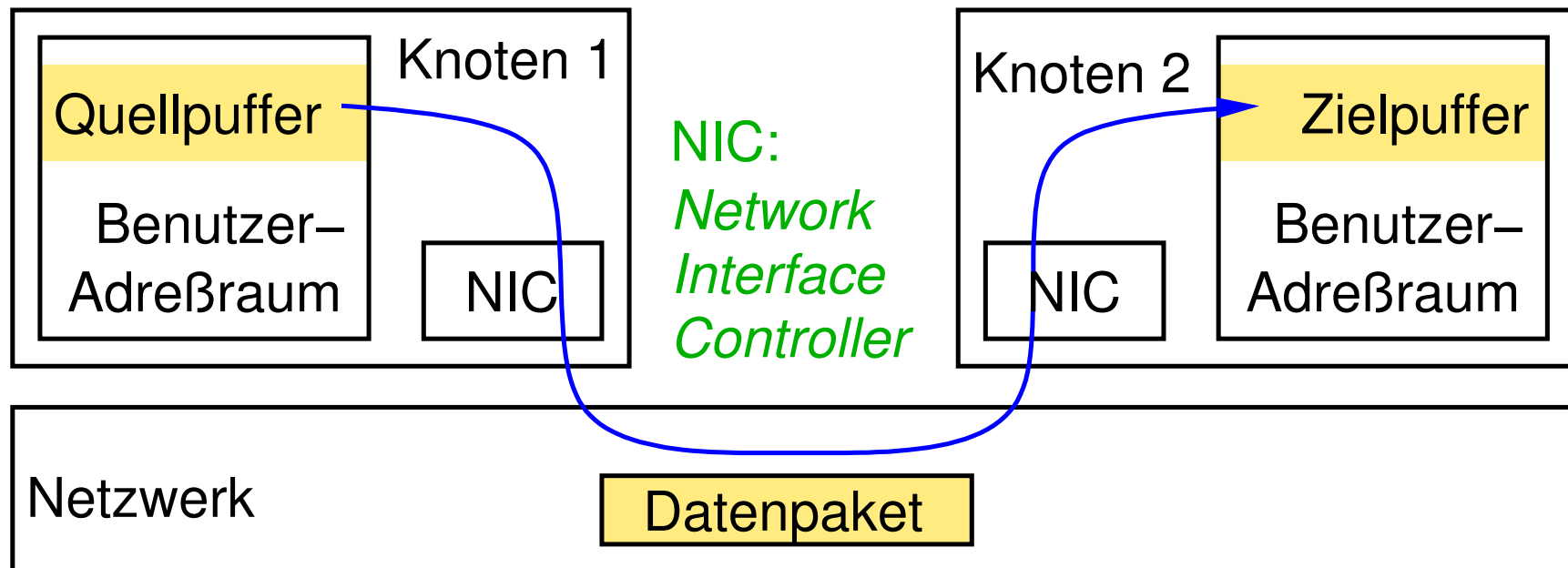


Vorteile von SDN:

- ➔ Flexiblere Weiterleitungsregeln
 - ➔ bei OpenFlow u.a. basierend auf:
 - ➔ Quell-/Ziel-MAC, VLAN-ID, VLAN Priorität
 - ➔ MPLS Label
 - ➔ Quell-/Ziel-IP-Adresse, Flow-Label, Protokoll
 - ➔ Quell-/Ziel-Port
 - ➔ Weiterleitung kann flussbasiert erfolgen
 - ➔ damit insbes. bessere QoS Unterstützung
- ➔ Intelligenter Controller möglich
 - ➔ z.B. können Weiterleitungsregeln automatisch aufgrund vorgegebener Policies erstellt werden

9.2.3 Protokolle der Anwendungsschicht

- ➔ Nachrichtenaustausch auf Anwendungsebene:
 - ➔ Kopieren von Daten aus dem Adreßraum des Senders in den Adreßraum des Empfängers



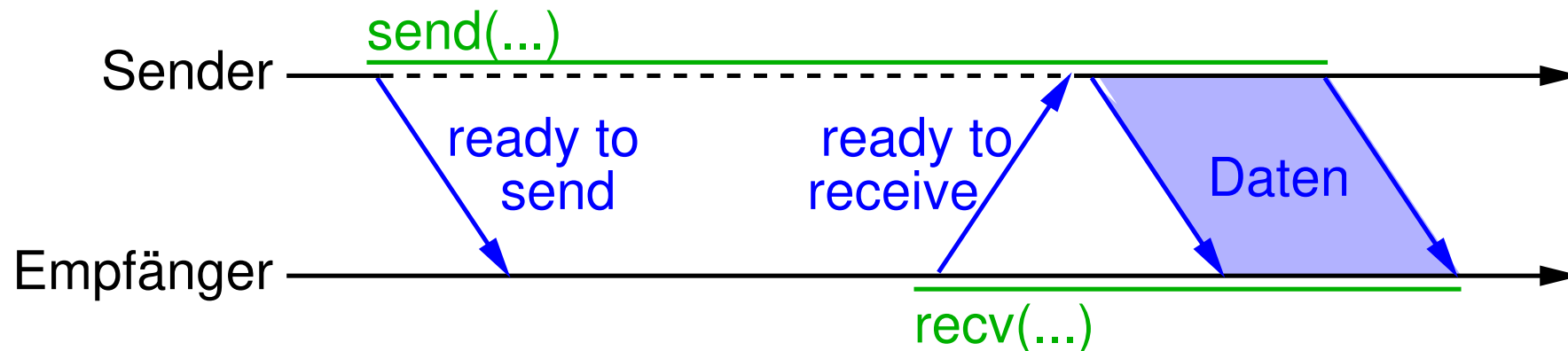
- ➔ Problem: Adresse des Zielpuffers erst bekannt, wenn Empfänger bereits auf Nachricht wartet



Asynchrones, optimistisches Protokoll

- ➔ Senderprozeß verschickt Nachricht ohne Wissen über den Zustand des Empfängerprozesses
- ➔ Falls Empfangsoperation noch nicht gestartet: Kommunikationsbibliothek muß Nachricht zwischenspeichern
 - ➔ Probleme:
 - ➔ Speicherplatzbedarf
 - ➔ Zusätzliche Kopie bei Empfangsoperation
- ➔ Daher nur für kurze Nachrichten eingesetzt
 - ➔ lange Nachrichten mit synchronem Protokoll (\sim RTS/CTS) (oder Zusicherung des Senders, daß Empfänger bereit ist)

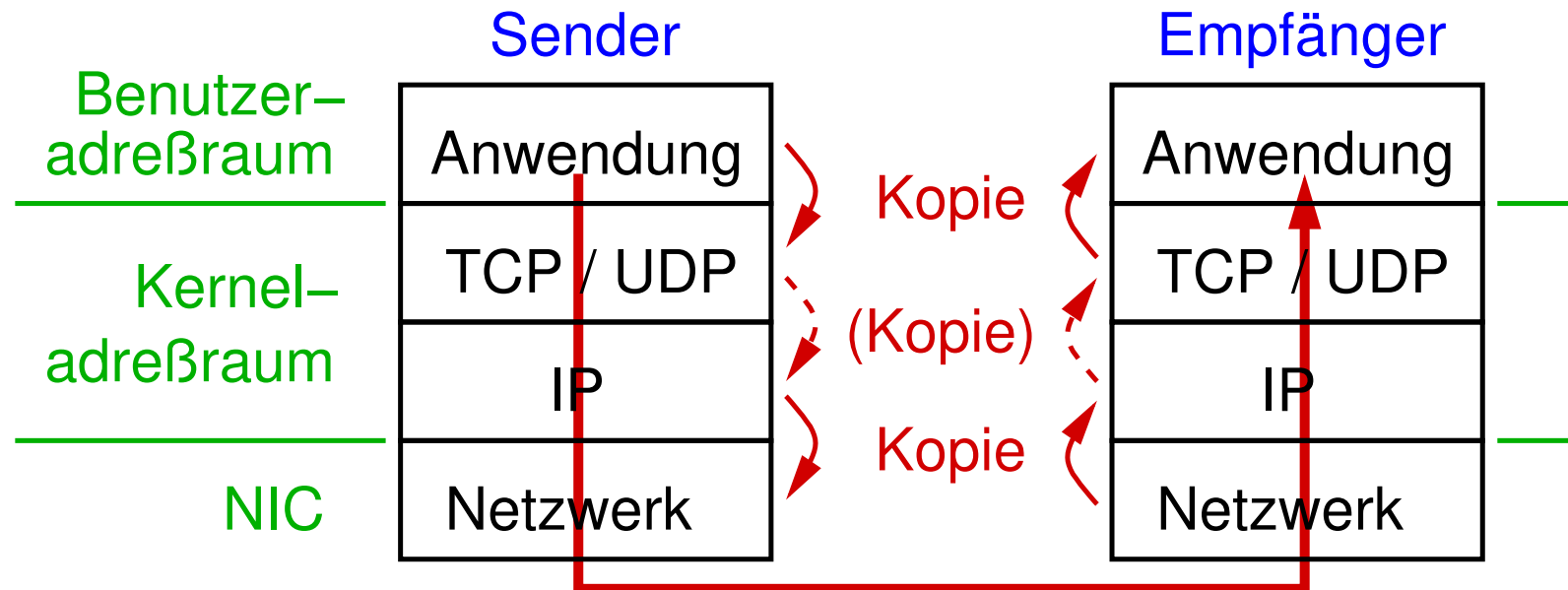
Synchrones Protokoll



- ➔ Sender wird blockiert, bis Empfang gestartet wurde
- ➔ *ready-to-send*-Nachricht enthält Länge der Daten
- ➔ Vorteil:
 - ➔ kein zusätzlicher Pufferspeicher notwendig
 - ➔ Vermeidung einer Kopieroperation beim Empfänger
- ➔ Nachteil: evtl. höhere Latenz und Blockierung des Senders

9.2.4 Remote DMA und OS Bypass

➔ Kommunikation in geschichteten Systemen (hier IP):



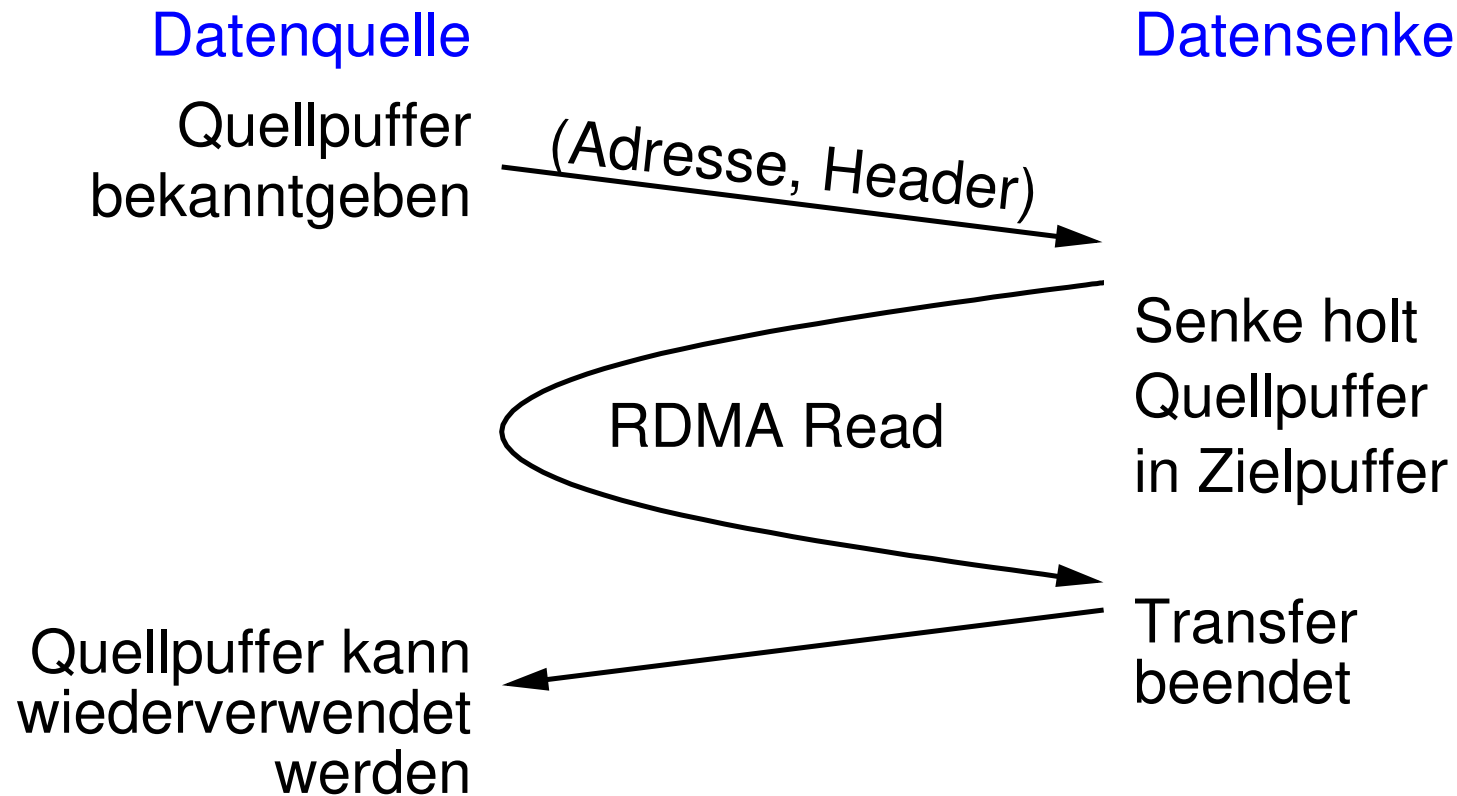
- ➔ Kopieroperationen limitieren Bandbreite, erhöhen Latenz, belasten CPU
- ➔ Ziel: Kopieren möglichst vermeiden („zero copy“)



Remote DMA

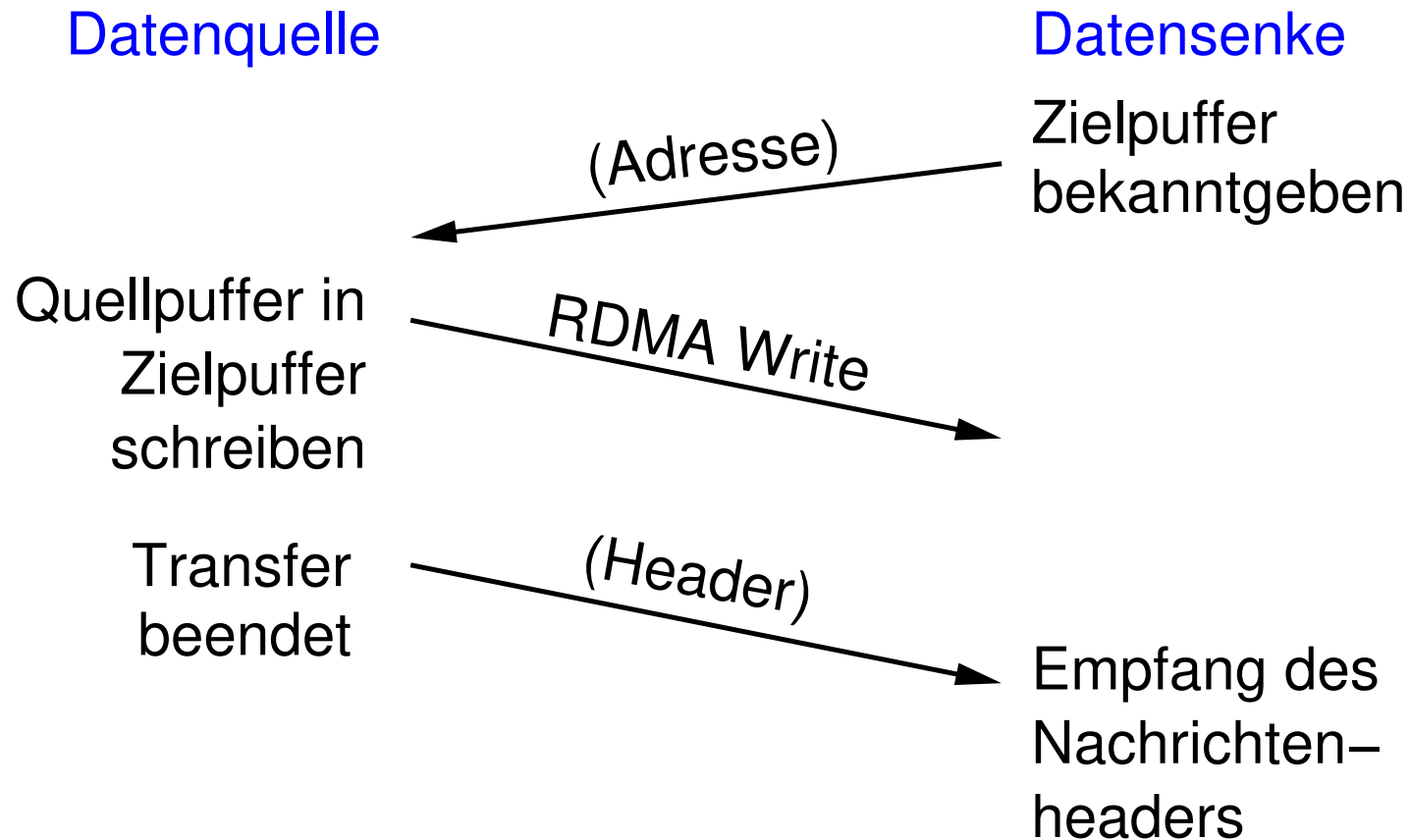
- ➔ DMA (*Direct Memory Access*)
 - ➔ DMA-Controller führt selbständig Datentransfers zwischen zwei Speicherbereichen durch (parallel zu CPU-Aktivität)
 - ➔ Auftrag durch die CPU: Quell- und Zieladresse, Länge
- ➔ *Remote DMA*
 - ➔ DMA-Transfer zwischen (Benutzer-)Speicherbereichen auf **verschiedenen** Rechnern über ein Netz
 - ➔ Operationen: DMA Read, DMA Write
 - ➔ Protokolle werden vollständig durch NIC realisiert
 - ➔ Betriebssystem muß Transfer nur noch anstoßen

Nachrichtenübertragung mit RDMA Read





Nachrichtenübertragung mit RDMA Write





VIA: Virtual Interface Architecture

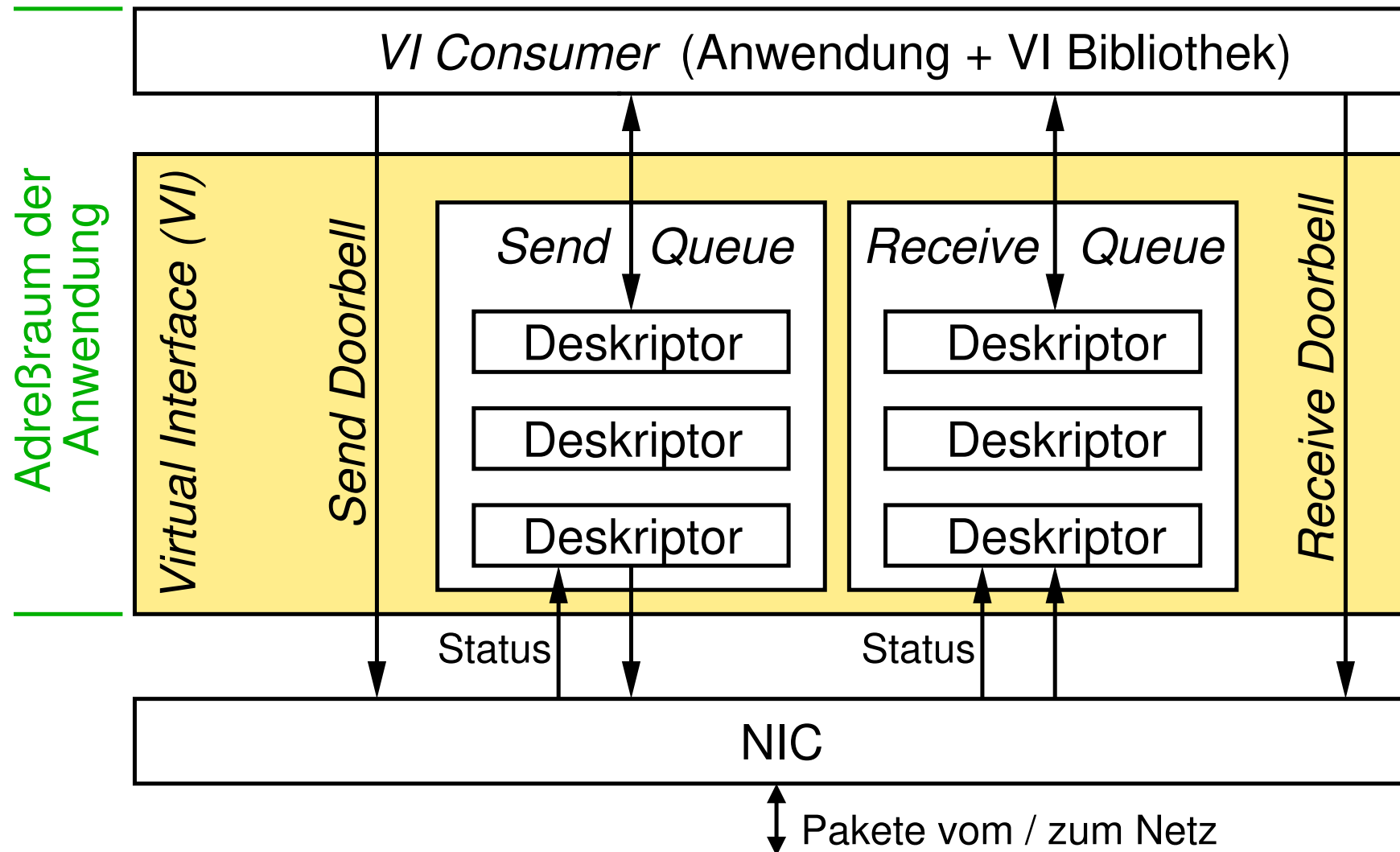
- ➔ Spezifikation von Intel, Compaq und Microsoft (1997)
- ➔ Ziel: Betriebssystem (BS) vollständig aus kritischem Pfad der Kommunikation entfernen
- ➔ Trennung von Steuerung und Daten
 - ➔ BS nur noch zum Aufsetzen von Verbindungen nötig (verbindungsorientiertes Protokoll)
 - ➔ Kommunikation ohne Einbeziehung des BS möglich
- ➔ Datenaustausch wird vollständig durch NIC realisiert, insbesondere Multiplexing und Demultiplexing
 - ➔ jeder Prozeß bekommt eigene virtuelle Schnittstelle zum NIC



VIA Operationen

- ➔ Initialisierung und Terminierung (über BS)
 - ➔ erzeugen der virtuellen Schnittstelle
- ➔ Registrierung und Deregistrierung von Puffern (über BS)
 - ➔ Puffer werden im physischen Adreßraum fixiert
- ➔ Verbindungsauf- und -abbau (über BS)
 - ➔ BS programmiert NIC und erzeugt Zugriffsschlüssel
- ➔ Datentransfer (direkt über virtuelle Schnittstelle)
 - ➔ Senden, Empfangen, RDMA Write, RDMA Read (optional)

VIA: Aufbau der virtuellen Schnittstelle





VIA: Ablauf eines Datentransfers

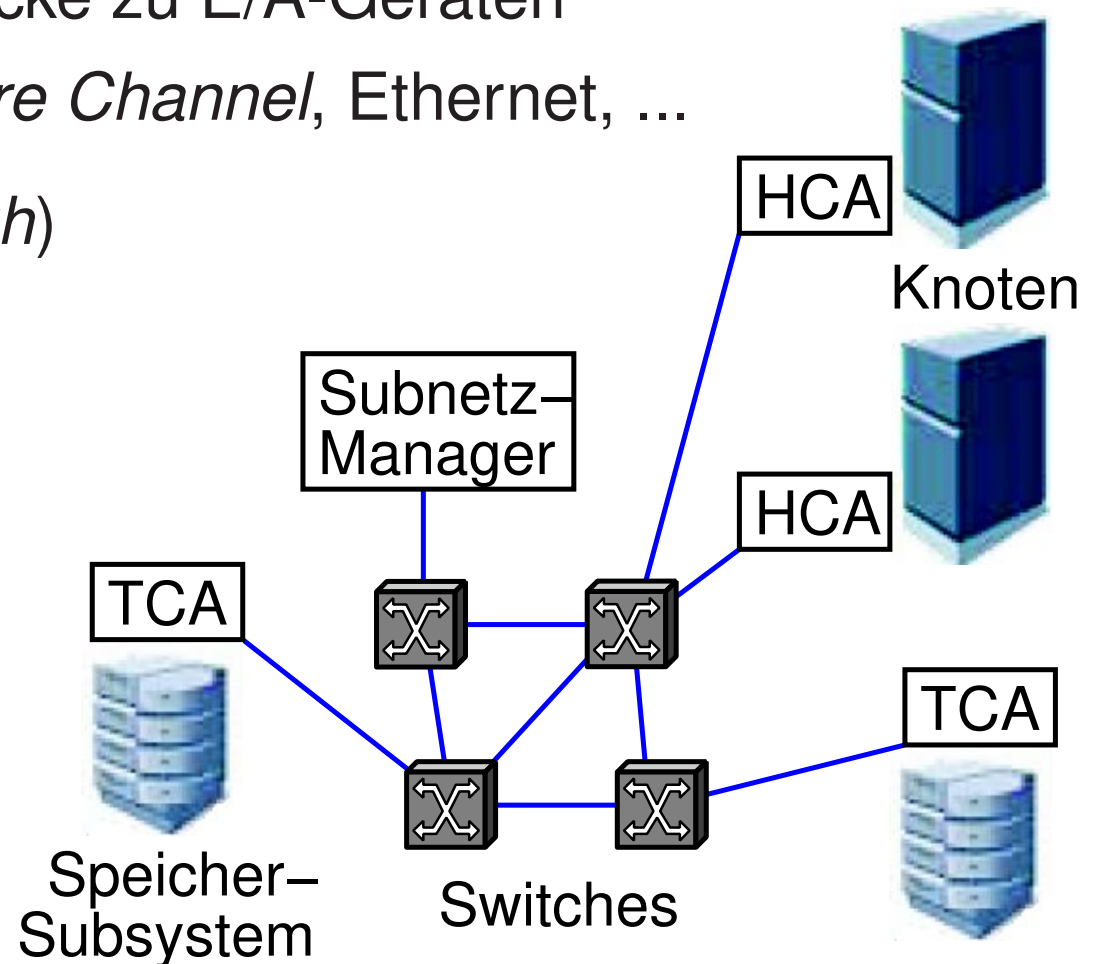
- ➔ Senden:
 - ➔ Deskriptor erzeugen (Sendepuffer, Adresse, Länge) und in *Send Queue* einreihen
 - ➔ *Send Doorbell* läuten
 - ➔ Statusbit im Deskriptor zeigt Ende der Operation an, Polling durch Anwendung (oder: blockierender BS-Aufruf)
- ➔ Empfangen: analog
 - ➔ eingehende Pakete ohne passenden Deskriptor werden verworfen
- ➔ RDMA Write / Read:
 - ➔ Initiator gibt auch Pufferadresse im anderen Knoten an



- ➔ Ziel: Hochgeschwindigkeitsnetz für Cluster
 - ➔ Kommunikation zwischen den Knoten
 - ➔ Anbindung von E/A-Geräten (Alternative zu E/A Bussen)
- ➔ Leistungsdaten:
 - ➔ Bandbreite bis zu 300 Gb/s
 - ➔ Hardware-Latenz pro Switch: ca. 100ns
 - ➔ zwischen Anwendungen: Latenz 1,32 μ s, Durchsatz 952 MB/s (mit 8 Gb/s Link)
- ➔ Infiniband umfasst die OSI-Schichten 1-4
 - ➔ Subnetze beliebiger Topologie, ggf. über Router verbunden
 - ➔ häufig: *Fat Tree*, Clos-Netzwerke, 3D-Torus, ...
- ➔ Unter den 500 schnellsten Rechnern der Welt: 25% mit Infiniband (54% mit 10G Ethernet)

Komponenten eines Infiniband-Subnetzes

- ➔ *Host Channel Adapter*: Netzwerkkarte
- ➔ *Target Channel Adapter*: Brücke zu E/A-Geräten
 - ➔ Schnittstelle zu SCSI, *Fibre Channel*, Ethernet, ...
- ➔ Switches (*Virtual-Cut-Through*)
- ➔ Subnetz Manager
 - ➔ zentrale (ggf. redundante) Komponente
 - ➔ Konfiguration der Switches (Weiterleitungstabellen!)
 - ➔ Netzwerkmonitoring





Adressierung

- ➔ LID (*Local ID*): 16-Bit Adresse innerhalb des Subnetzes
 - ➔ wird vom Subnetz Manager zugewiesen
- ➔ GUID (*Globally Unique ID*): weltweit eindeutige 64-Bit Adresse
 - ➔ analog zur MAC-Adresse bei Ethernet
- ➔ GID (*Global ID*): gültige IPv6 Adresse
 - ➔ z.B. gebildet aus Subnetz-Präfix und GUID

Paketformat

- ➔ Lokaler und globaler Header (Schicht 2 bzw. 3), ggf. weitere
- ➔ Zwei CRCs: für Ende-zu-Ende und *Hop-by-Hop* Fehlererkennung
- ➔ Wählbare MTU: 256 B, 1 KB, 2 KB, 4 KB

Bitübertragungsschicht

- ➔ Unterschiedliche Datenraten
 - ➔ SDR (2 Gb/s), DDR (4 Gb/s), QDR (8 Gb/s), FDR (14 Gb/s), EDR (25 Gb/s)
- ➔ Verbindungsleitungen: Kupfer oder Glasfaser
 - ➔ ein Kabelpaar (bzw. eine Faser) pro Übertragungsrichtung
 - ➔ Bündelung von 4 bzw. 12 Leitungen möglich (4x / 12x Link)
 - ➔ max. Länge ca. 10-17m (Kupfer), 125m-10km (Glasfaser)
- ➔ *Auto-negotiation* für Datenrate und Linkbreite
- ➔ Codierung:
 - ➔ 8B10B bei SDR, DDR und QDR; 64B66B bei FDR und EDR
 - ➔ Kontrollcodes für Framing, Auto-negotiation, Training, ...



Sicherungsschicht

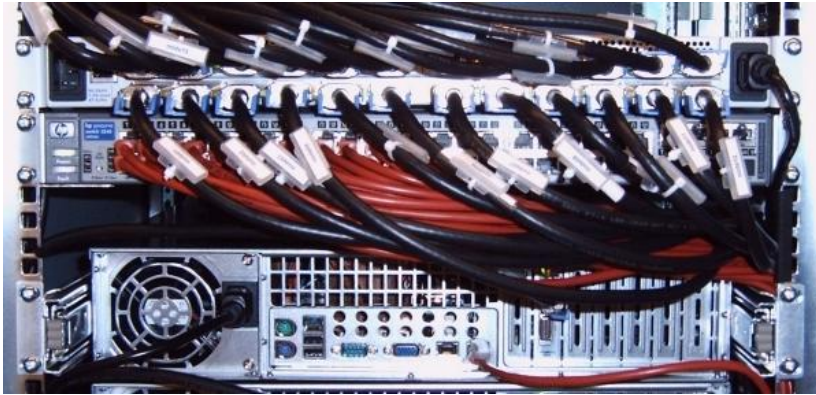
- ➔ Physische Links unterteilt in 2-16 virtuelle Verbindungen (*Virtual Lanes, VLs*)
 - ➔ VLs besitzen unterschiedliche Prioritäten (für QoS)
 - ➔ jeweils eigene Warteschlange, *Weighted RR* Scheduling
 - ➔ VL 15 (höchste Priorität) reserviert für Netzwerkmanagement
- ➔ Innerhalb jeder VL (außer VL 15)
 - ➔ paketweise Flußkontrolle
 - ➔ Überlastkontrolle (ähnlich DECbit, optional)



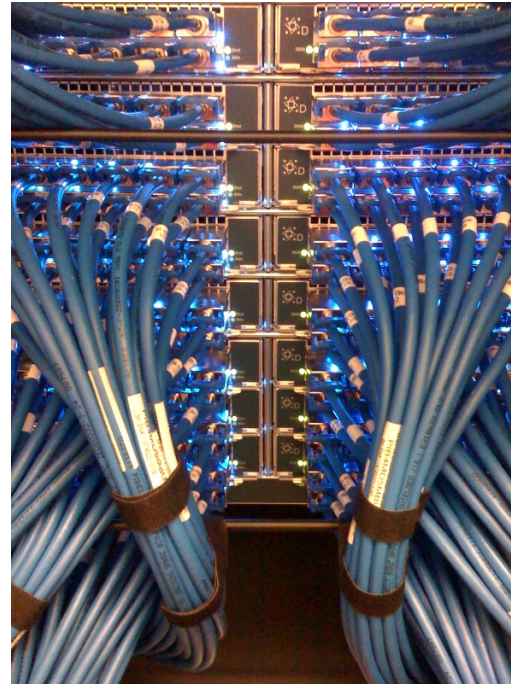
Transportschicht

- ➔ Verbindungslose und verbindungsorientierte Dienste
 - ➔ jeweils unzuverlässig und zuverlässig
 - ➔ zusätzlich *Raw-Modus* zum Transport von z.B. IPv6 Paketen
 - ➔ Paketierung und Sicherungsprotokoll in Hardware realisiert
- ➔ Verbindungsorientierte Dienste unterstützen Fehlertoleranz
 - ➔ Aufbau von zwei alternativen Verbindungen möglich
 - ➔ automatisches Umschalten im Fehlerfall
 - ➔ Anhalten („Leerlaufen lassen“) der Warteschlangen möglich
 - ➔ Verbindung wird dann durch Software umkonfiguriert
- ➔ Neben Send/Receive auch RDMA (Read/Write/Atomics)
 - ➔ jeweils mit *OS-Bypass* (ähnlich zu VIA)

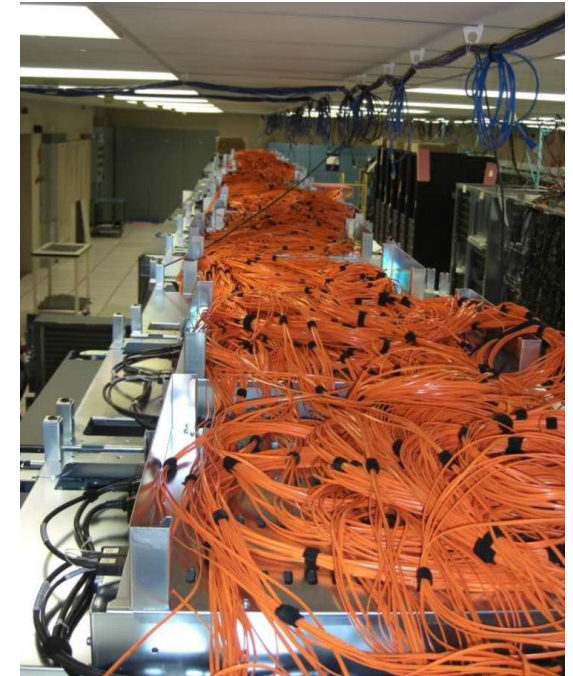
9.3 Beispiel: Infiniband ...



Infiniband-Switch
(Sternwarte Hamburg)



Infiniband-Switch
(Harvard Univ.)



HLRN_II (Glasfaser)



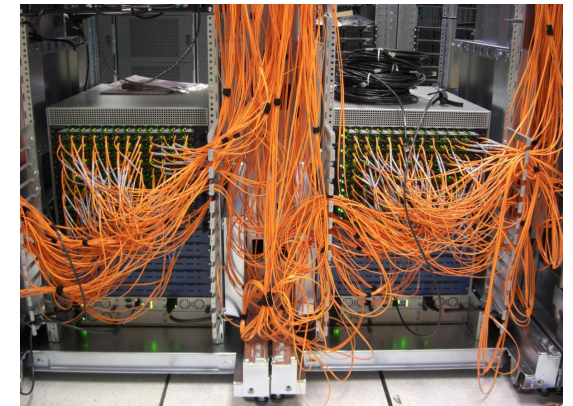
Stecker 4x



Stecker 12x



HCA (2 * 4x)





- ➔ Optimierung von (Bisektions-)Bandbreite und Latenz
 - ➔ Kosten (oft) sekundär
- ➔ Heute Vernetzung i.d.R. mit *Crossbar*-Switches
 - ➔ blockierungsfrei
 - ➔ *Virtual-Cut-Through* bzw. *Wormhole*-Routing
 - ➔ feingranulare Flußkontrolle auf Bitübertragungsebene
 - ➔ bei höherer Knotenzahl: z.B. Clos-Netze
- ➔ Trend: *Software Defined Networking*
- ➔ Zur Reduktion der Latenz:
 - ➔ Einsatz von RDMA („*zero copy*“)
 - ➔ Betriebssystem-*Bypass*, z.B. VIA