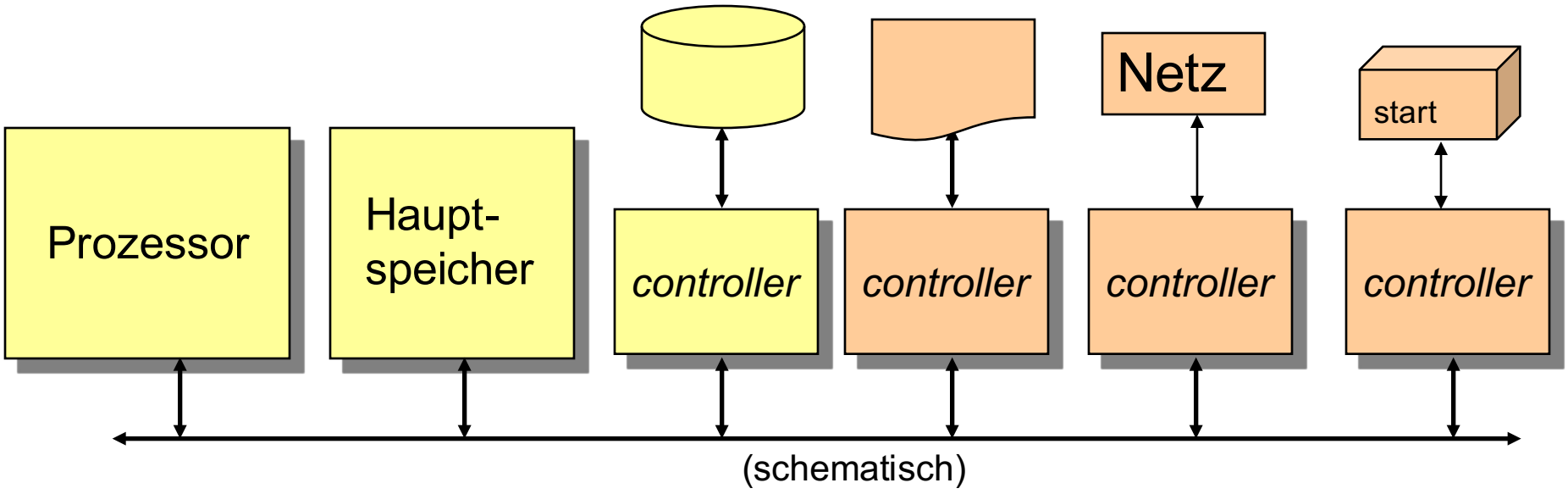
Rechnerstrukturen, Teil 2

Vorlesung 4 SWS WS 19/20

2.5 Kommunikation, Ein-/Ausgabe

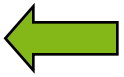
Prof. Dr. Jian-Jia Chen
Fakultät für Informatik – Technische Universität Dortmund
jian-jia.chen@cs.uni-dortmund.de
http://ls12-www.cs.tu-dortmund.de

Kontext



Die Wissenschaft Informatik befasst sich mit der Darstellung, Speicherung, Übertragung und Verarbeitung von Information. [Gesellschaft für Informatik]

Wie sieht das interne Verbindungsnetzwerk aus?
☞ Bussysteme, Bustopologien

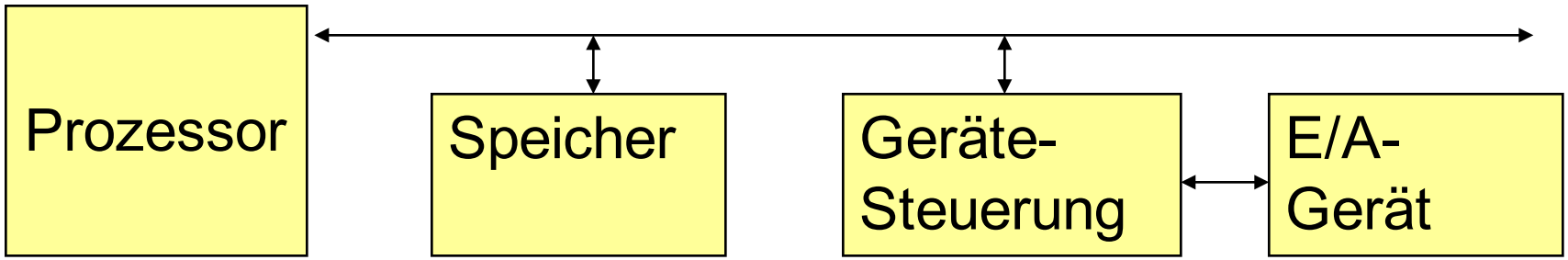
- Wie sieht das Verbindungsnetzwerk aus? 
- Wie adressiert man die E/A-Geräte und deren Steuerungen?
- Busse mit oder ohne Bestätigung?
- Wie stimmt man die Geschwindigkeit der E/A-Geräte mit der des Prozessors ab?

Viele Entwurfsbeschränkungen:

- Geschwindigkeit
- Kosten
- Kompatibilität
- ...

☞ viele Bus-Topologien.

1. Ein einziger Bus



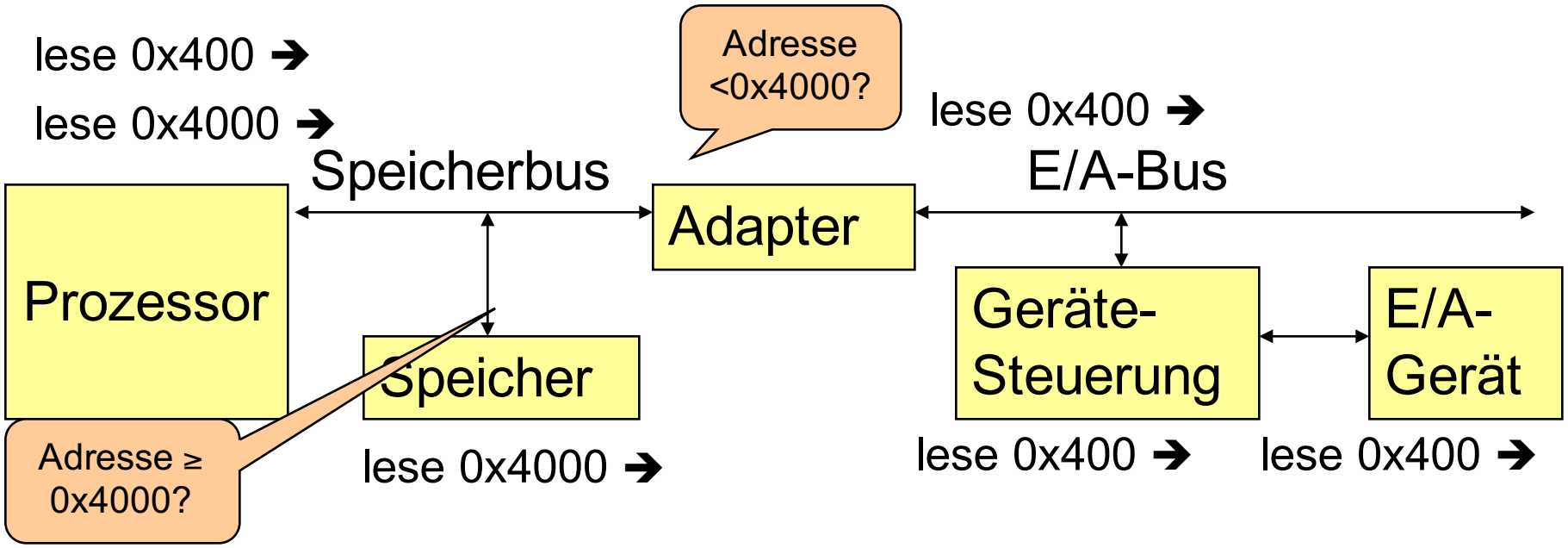
Sehr einfach;

Motiviert u.a. durch Pinbeschränkungen;

Enge Kopplung von Speicher- und E/A-Spezifikationen;

Eingeschränkte Parallelarbeit.

2. Einstufiger Adapter zum Übergang auf E/A-Bus

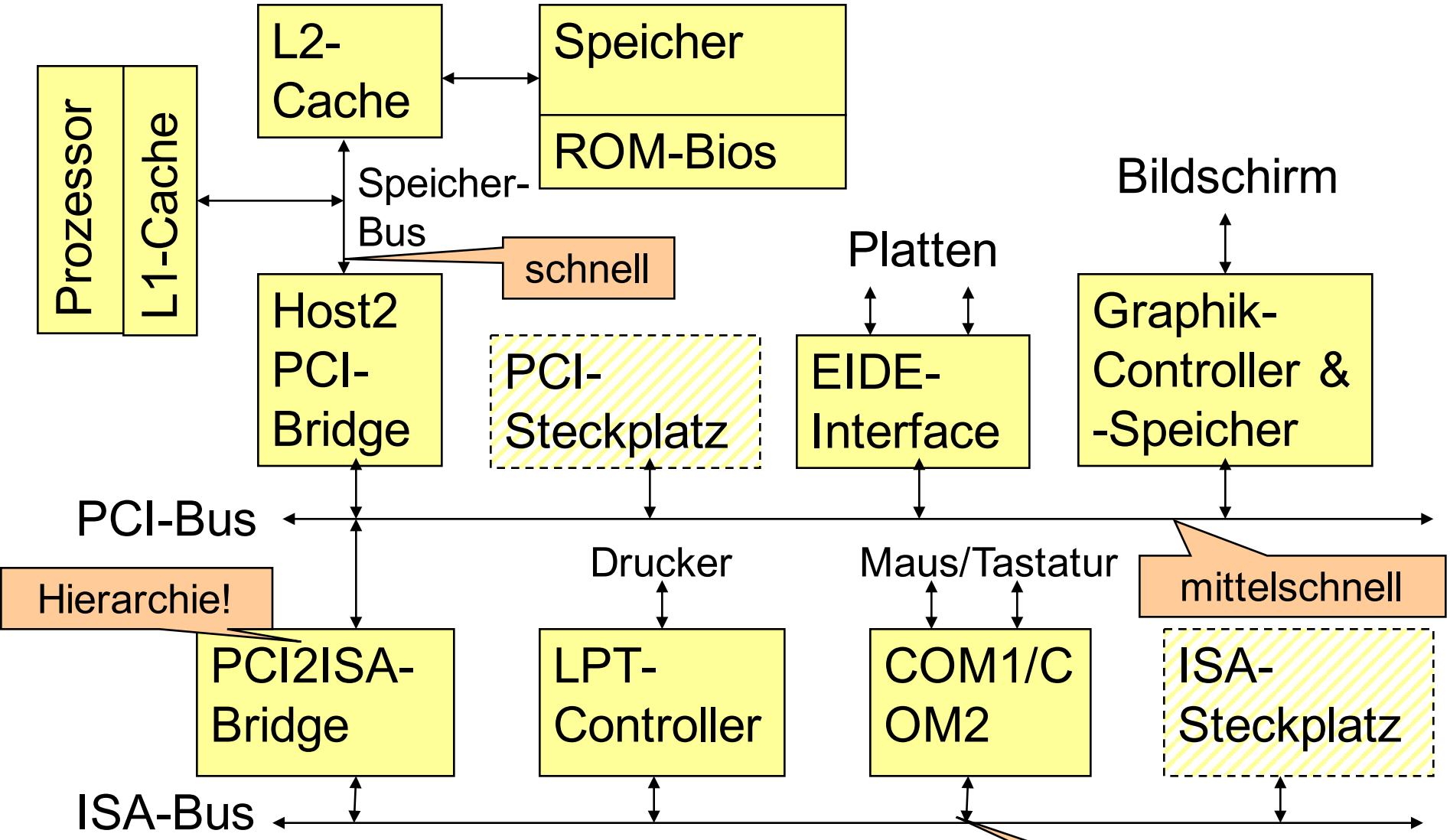


Spezielle Speicheradressen oder Kontrollleitung *IO/Memory.*

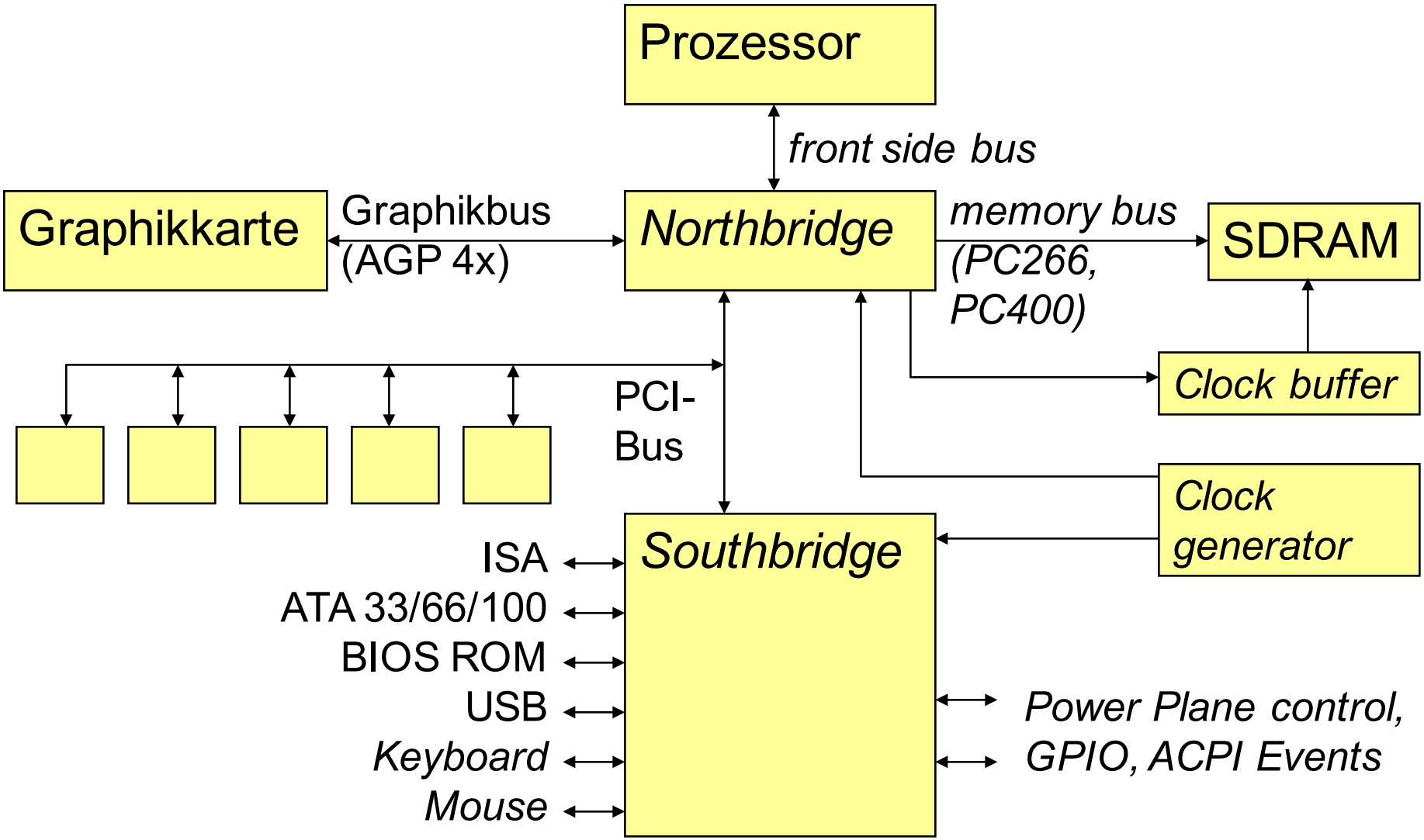
Speicherbus ggf. mit größerer Wortbreite;

höhere Geschwindigkeit.

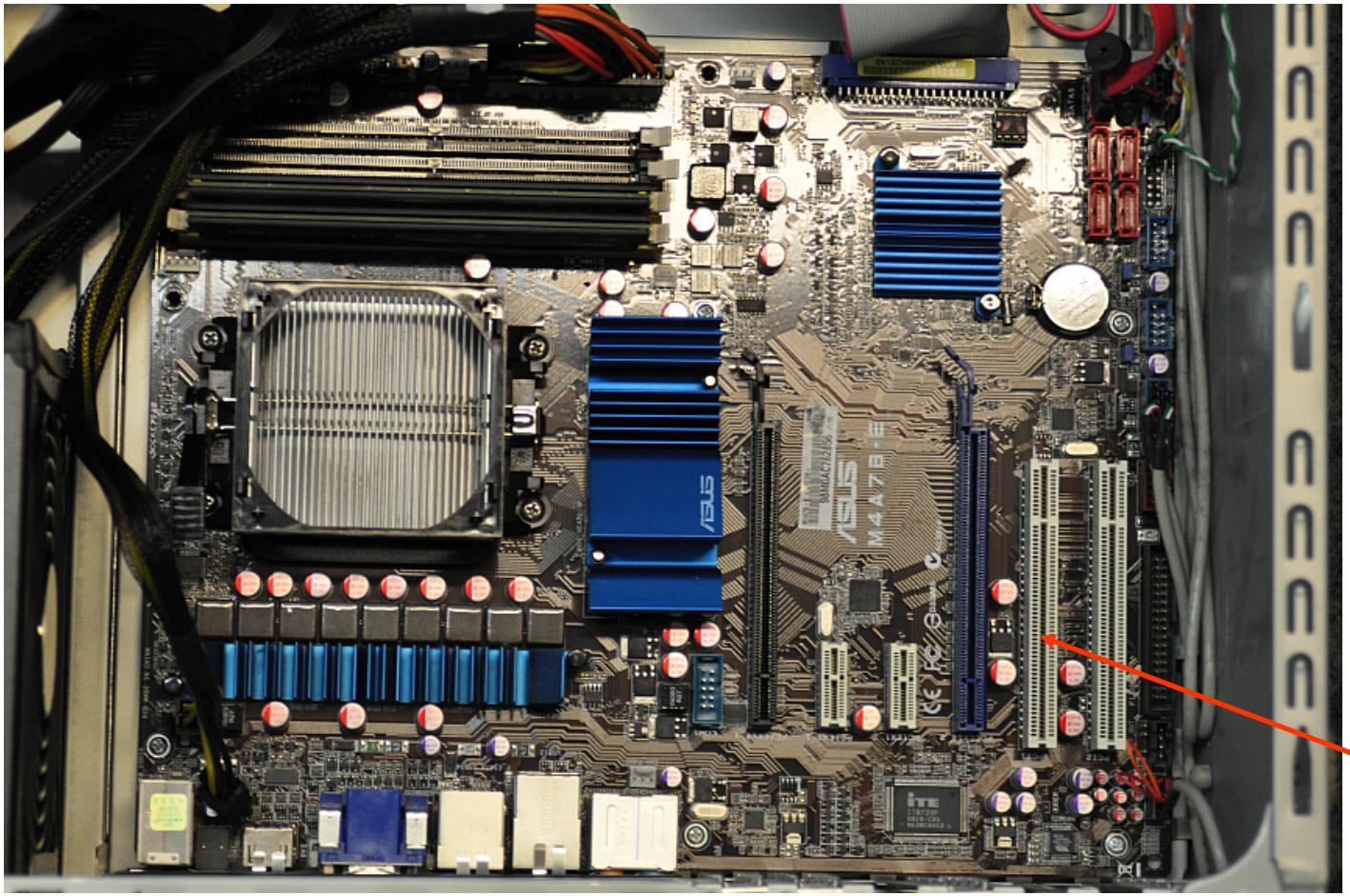
3. Grundstruktur der ersten PCI-basierten Systeme



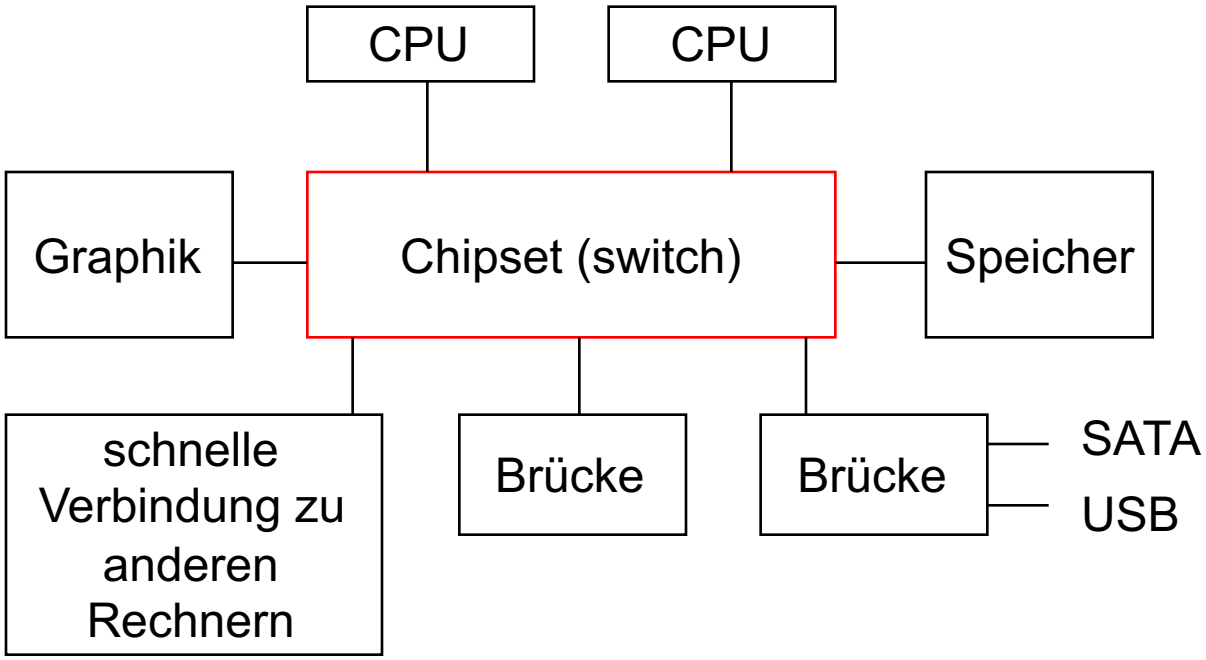
4. Grundstruktur späterer PCI-basierter Systeme



Wo sind diese Komponenten auf dem Board?

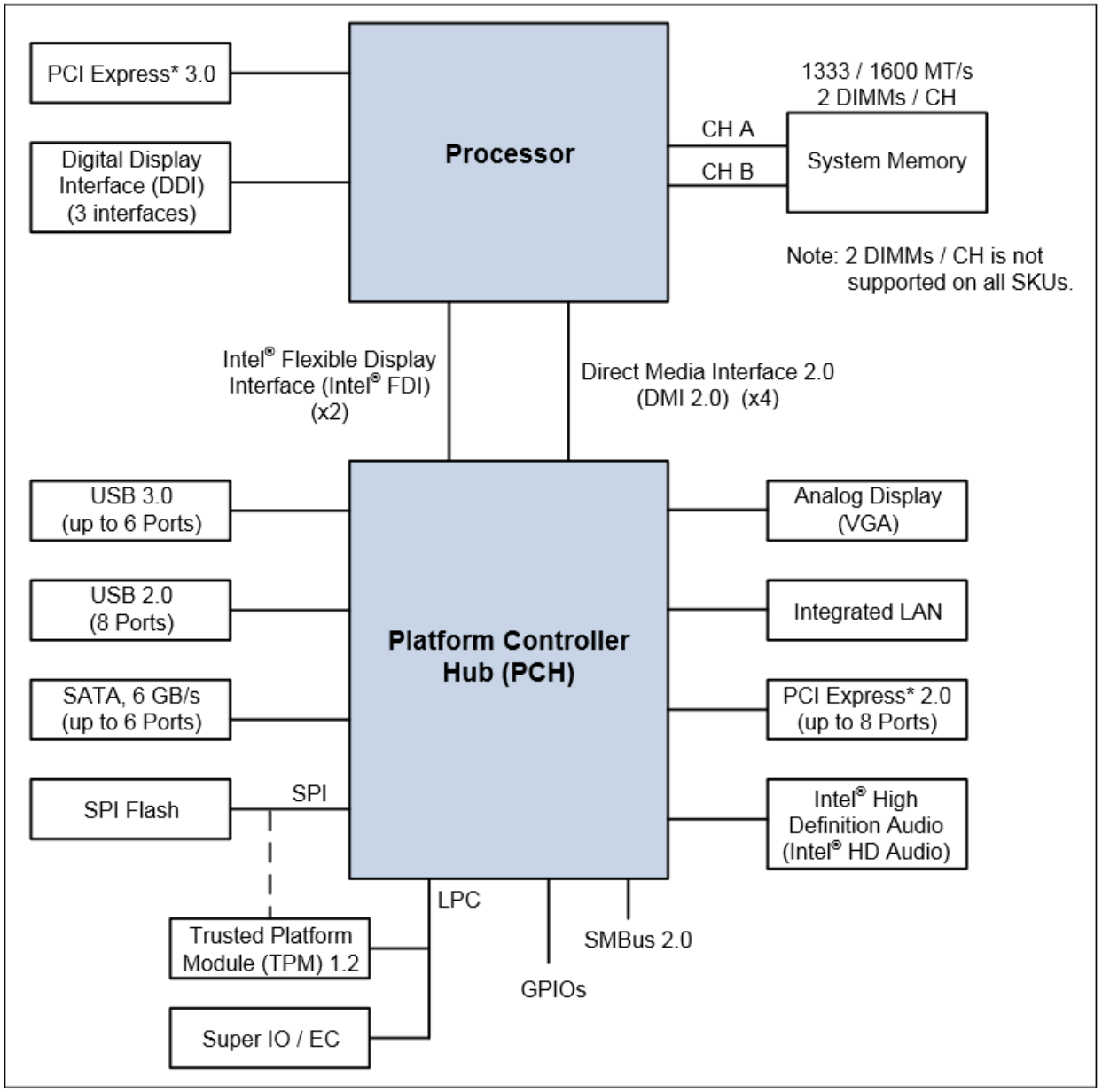


5. PCI-Express



Möglichkeit der Vermeidung eines Busses, an den mehrere Karten angeschlossen sind und die gegenseitig die Geschwindigkeit begrenzen.

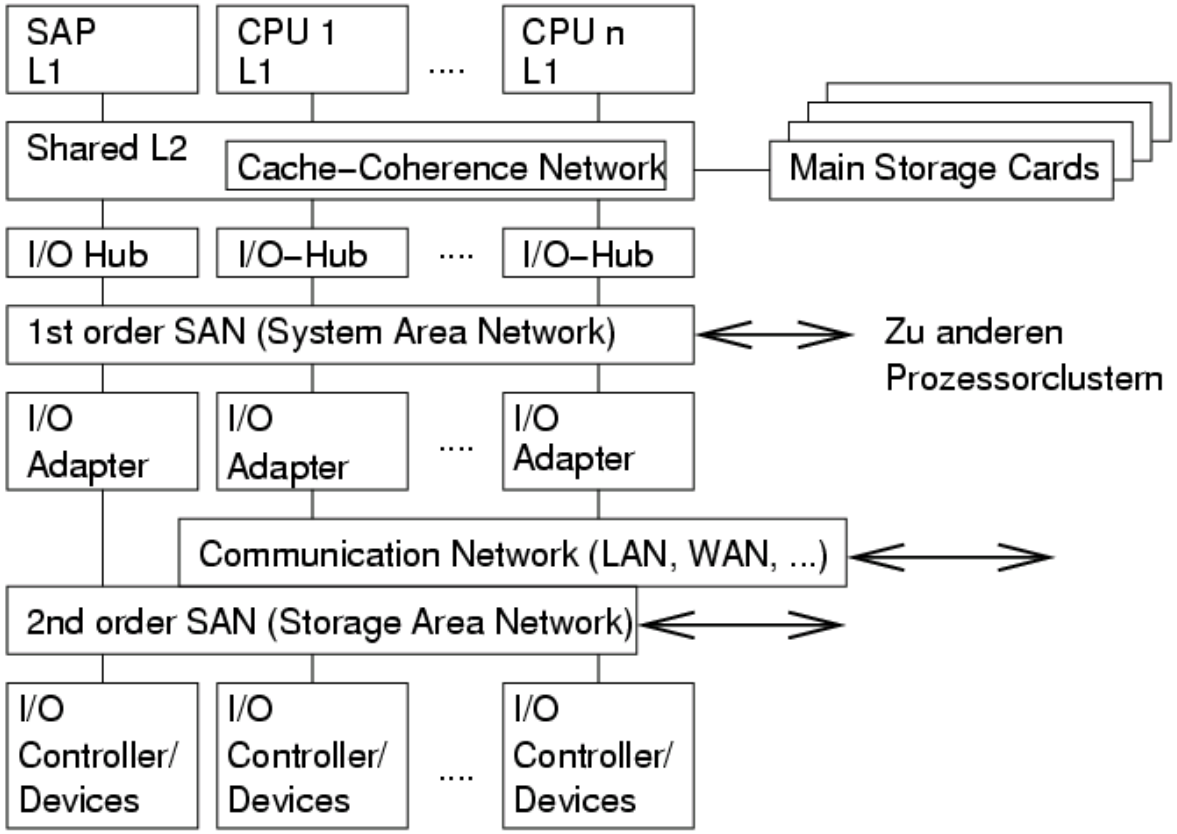
Aktueller Intel Prozessor



http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf

6. Vollständig separate Speicher- und E/A-Busse (Großrechner)

Der Speicher ist Multiport-Speicher, und es ex. Kanäle, welche ohne CPU-Belastung Daten zwischen Speicher und E/A-Steuerungen transportieren & eigene Programme ausführen.



Adressierung

- Wie sieht das Verbindungsnetzwerk aus?
- Wie adressiert man die E/A-Geräte und deren Steuerungen? 
- Busse mit oder ohne Bestätigung?
- Wie stimmt man die Geschwindigkeit der E/A-Geräte mit der des Prozessors ab?

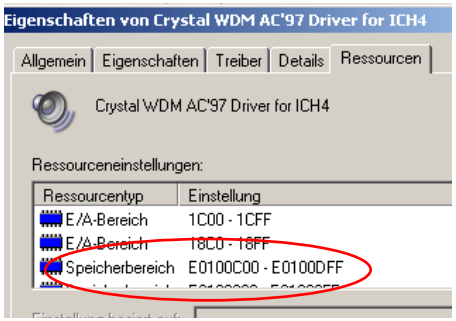
2 Techniken:

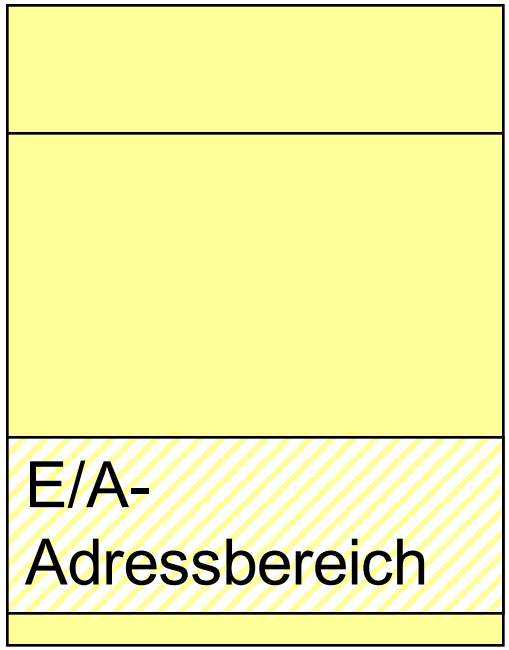
1. Speicherbezogene Adressierung
2. Eigene E/A-Adressen

1. Speicherbezogene Adressierung (*memory mapped I/O*)

Geräte erhalten spezielle Speicheradressen,
Kommunikation mittels *load*- und *store*-Befehlen,
Prozessoren ohne separate Speicher- und E/A-Schnittstelle;
Trennung in E/A- und Speicherbus nur über Busadapter

Adressraum:





load = Eingabe;
store = Ausgabe

Muss von *Caching* ausgenommen werden.

Beispiel: MIPS-Maschine wie bei SPIM realisiert

Eigenschaften speicherbezogener E/A-Adressierung

Vorteile:

- Großer E/A-Adressraum aufwandsarm,
- Wenig Festlegung beim Prozessorentwurf,
- Keine separaten E/A-Befehle notwendig;

  kleiner Befehlssatz.

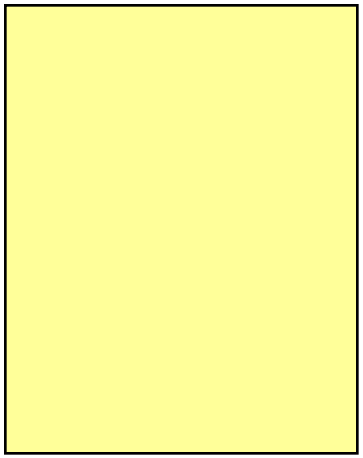
Nachteile:

- Besonderes E/A-Timing nur über Busadapter möglich,
- Zugriffsschutz auf E/A-Geräte nur über

  Speicherverwaltung.

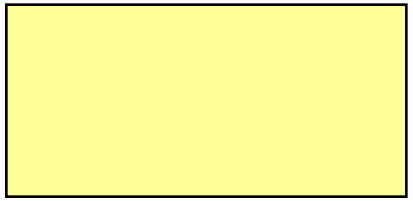
2. Separate E/A-Adressen

Speicheradressraum



spezieller E/A-Befehle

E/A-Adressraum

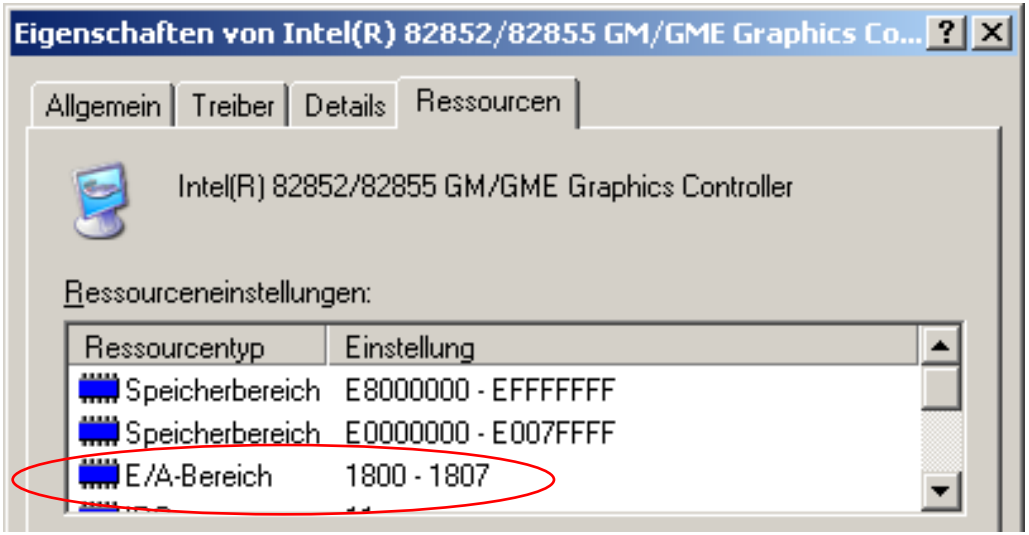


in = Eingabe;
out = Ausgabe

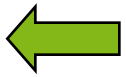
Umkehrung von Vor- und Nachteilen der speicherbezogenen Adressierung.

Prozessoren mit E/A-Adressen können auch speicherbezogene Adressierung nutzen.

Beispiel: Intel x86/Pentium-Familie.



2.5.1.3 Synchrone und asynchrone Busse

- Wie sieht das Verbindungsnetzwerk aus?
- Wie adressiert man die E/A-Geräte und deren Steuerungen?
- Busse mit oder ohne Bestätigung? 
- Wie stimmt man die Geschwindigkeit der E/A-Geräte mit der des Prozessors ab?

Unterscheidung zwischen:

- unidirektionalem Timing (bei synchronen Bussen) und
- bidirektionalem Timing (bei asynchronen Bussen).

Synchrone und asynchrone Busse

Unterscheidung basiert auf Unterscheidung zwischen unidirektionalem Timing (bei synchronen Bussen) und bidirektionalem Timing (bei asynchronen Bussen).

- **Unidirektionales Timing:**
  Kommunikationspartner verlassen sich darauf, dass Partner innerhalb festgelegter Zeit reagieren.
- **Bidirektionales Timing:**
  Kommunikationspartner bestätigen per Kontrollsignal (senden ein ***acknowledgement***), dass sie in der erwarteten Weise reagiert haben.

Vergleich der beiden Methoden

	unidirektional	**bidirektional**
Vorteile	einfach; bei konstanten Antwortzeiten schnell	passt sich unterschiedlichen Geschwindigkeiten an
Nachteile	Kommunikationspartner muss in bestimmter Zeit antworten	komplexer; Zeitüberwachung notwendig; evtl. langsam
	synchrone Busse, Speicherbusse	asynchrone Busse, E/A- und Peripheriebusse

Sender und Empfänger als kommunizierende Automaten

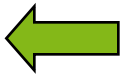
- Sender und Empfänger stellen jeweils einen Automaten dar, der die Bussignale erzeugt.
- Diese Automaten kommunizieren bzw. synchronisieren sich über die Steuersignale wie *address strobe* (AS) und *acknowledge (ACK).*

Zusammenfassung

- Struktur der Verbindungsnetzwerke richtet sich nach einer Vielzahl von Randbedingungen; Spannbreite von
  - einfachen Leitungen
  - bis zu Kommunikationsprozessoren
- Adressierung:
  - *Memory-Mapped I/O*
  - Separate E/A-Adressen
- Busse mit oder ohne Bestätigung?
  - Unidirektionale Busse
  - Bidirektionale
- Wie stimmt man die Geschwindigkeit der E/A-Geräte mit der des Prozessors ab?

Ablauf der Kommunikation zwischen CPU und Gerätesteuerungen

- Wie sieht das Verbindungsnetzwerk aus?
- Wie adressiert man die E/A-Geräte und deren Steuerungen?
- Busse mit oder ohne Bestätigung?
- Wie stimmt man die Geschwindigkeit der E/A-Geräte mit der des Prozessors ab?



Techniken:

1. Direkte Ein-/Ausgabe (*Immediate devices*)
2. Status-Methode (*busy waiting*)
3. *Polling*
4. *Interrupts*
5. *Direct memory access* (DMA)

Programmed I/O

1. Direkte Ein-/Ausgabe (*immediate devices*)

In manchen Fällen: keine Synchronisation erforderlich, weil die Geräte immer zu Ein- und Ausgaben bereit sind und weil die Häufigkeit des Lesens bzw. Schreibens unwichtig ist.

Beispiele:

- Setzen von Schaltern/Lämpchen
- Ausgabe an Digital/Analog-Wandler
- Schreiben in Puffer
- Setzen von Kontrollregistern

Programmieren:

- **`load`**- bzw. **`input`**-Befehl zur Eingabe
- **`store`** bzw. **`output`**-Befehl zur Ausgabe

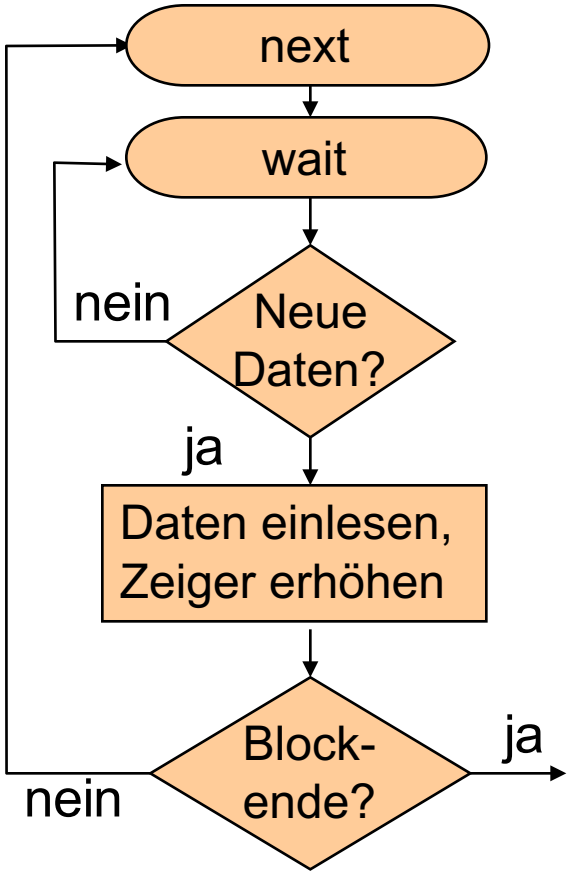
2. Status-Methode (*Busy waiting*) - Prinzip der Warteschleife -

Das „Bereit“-Bit wird in einer Warteschleife abgeprüft, bis nach Anzeige eines entsprechenden Werts die Ein- bzw. Ausgabe erfolgen kann.

Prinzip:

```
REPEAT
   REPEAT
      lese Statuswort des Gerätes
   UNTIL  Bereit-Bit im Statuswort ist gesetzt;
   lese (bzw. schreibe) Datenwort;
   erhöhe Blockzeiger;
UNTIL  Ende des Blocks ist erreicht;
```

Prozessor bleibt während des Wartens beschäftigt (*busy*).



Eigenschaften der *busy waiting* Methode

Nachteile:

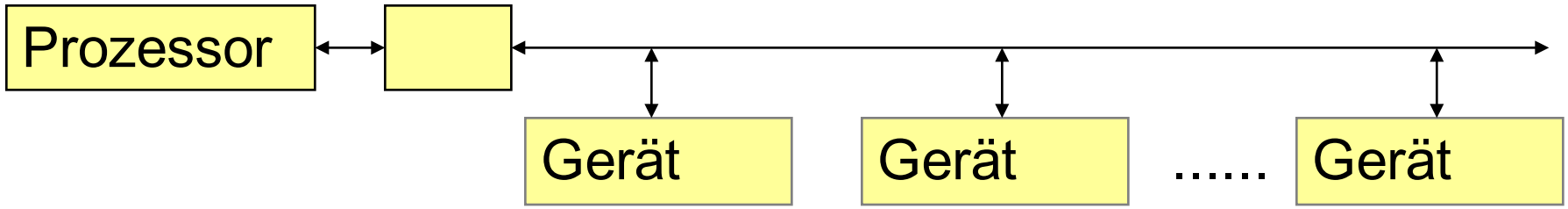
- Keine Möglichkeit der verzahnten Bearbeitung anderer Aufgaben.
- Geringe Übertragungsgeschwindigkeit.

Anwendung:

- Wenn keine anderen Aufgaben vorliegen.
- Wenn das Gerät so schnell ist, dass die Schleife selten durchlaufen wird und für die Umschaltung auf andere Aufgaben keine Zeit bleibt.
- Wenn das Gerät zu keiner anderen Methode fähig ist.

3. Polling

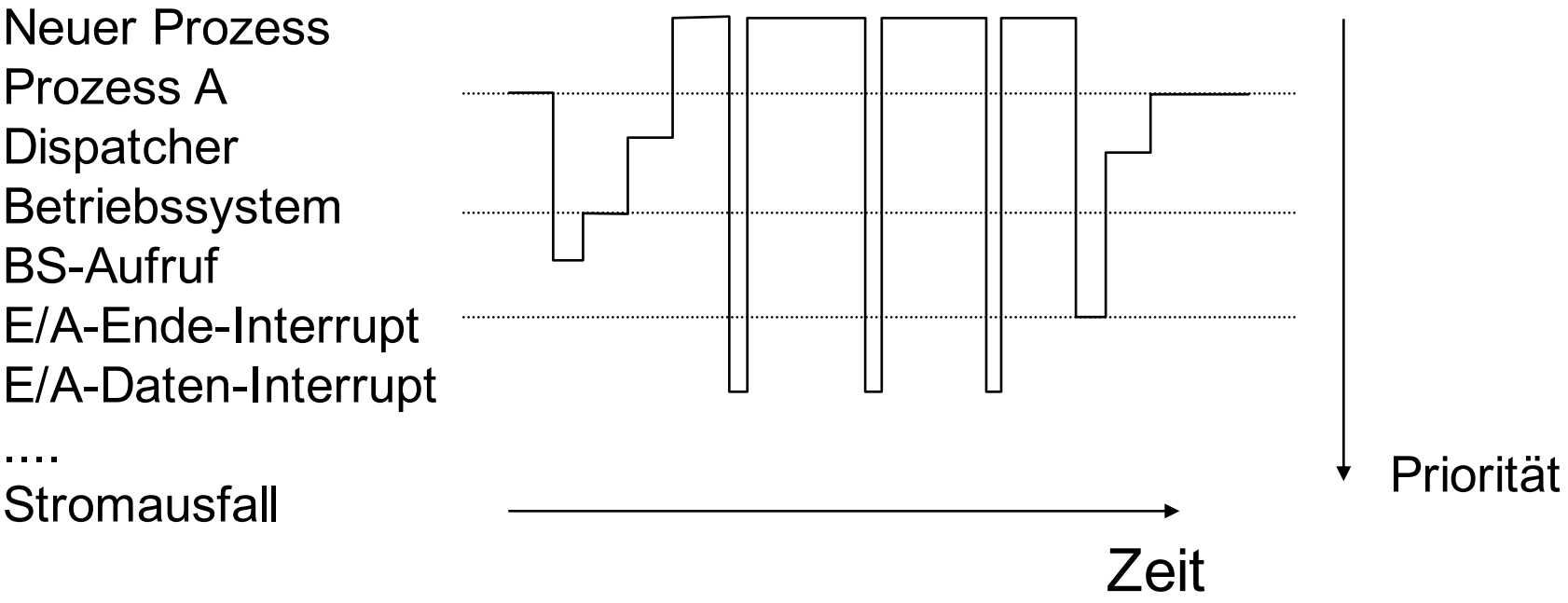
Polling = gelegentliches Prüfen des Bereit-Bits mit verzahnter Bearbeitung anderer Aufgaben.
Anwendungsbeispiel:
Erfassung von Übertragungswünschen einer großen Anzahl von Geräten:



Erlaubt eine gerechte Bedienung vieler Geräte (z.B. Terminals oder USB-Geräte). Vermeidet Überlastungen durch viele Übertragungswünsche; Vorteilhaft bei *denial-of-service*-Angriffen; unter Umständen große Reaktionszeit.

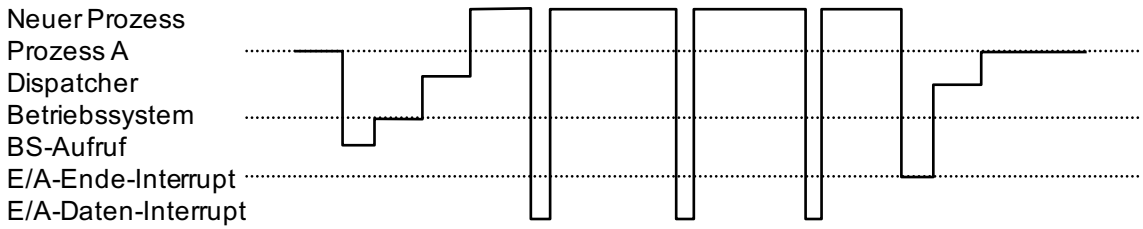
4. Unterbrechungen

Gerätesteuerung unterbricht den Prozessor, falls Datentransport(e) erforderlich werden oder falls Fehler auftreten.



Ablauf

- Prozess A überträgt E/A-Auftrag an das Betriebssystem.
- *Dispatcher* schaltet auf neuen Prozess um.
- Gerät sendet *Interrupt*.
- Falls *Interrupt* nicht gesperrt ist und die Priorität ausreichend hoch ist und der laufende Maschinenbefehl abgeschlossen ist:
- Sicherung des Zustandes (Software oder Mikroprogramm).
- Feststellen der *Interrupt*-Ursache (ggf. *vectored interrupt*).
- Datentransport: Ein-/Ausgabe, Lesen/Schreiben Speicher, Pufferzeiger erhöhen, Test auf Blockende.
- Restaurieren des Kontexts, *return from interrupt*.
- Prozess fährt in Bearbeitung fort.
- Nach einer Reihe von Datenübertragungsinterrupts erfolgt ein Blockende-*Interrup*t welcher den *Dispatcher* startet.

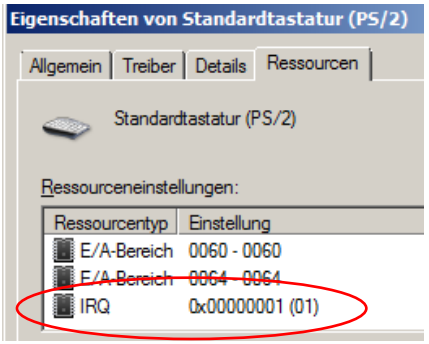


Eigenschaften

- Wegen des *Overheads* nicht für die schnelle Datenübertragung geeignet

Zu programmieren:
- Starten der Geräte vor der Übertragung.
- Übertragung per *Interrupt*-Routine.

Anwendungen:
- Häufig zur Übertragung zu langsamer Peripherie eingesetzt.



Für Rechner ohne *direct memory access* (s.u.) ist diese Methode erforderlich.

5. Direct memory access (DMA) & Buszuteilung

Beim ***direct memory access*** (DMA) werden die Datentransporte **direkt** zwischen Gerätesteuerung und Speicher durchgeführt, **ohne** Beteiligung des Prozessors.

Bislang: nur der Prozessor hat die volle Kontrolle über den Bus (legt Adressen an und erzeugt Kontrollsignale, ..),
andere Einheiten dürfen nur auf seine Anforderung tätig werden.

Diese Funktion des Prozessors heißt ***Bus-Master*-Funktion**.

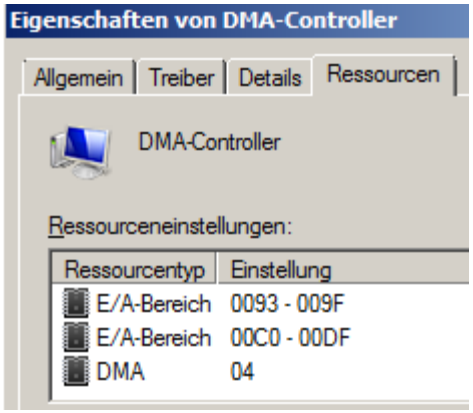
DMA macht es erforderlich, dass Gerätesteuerungen *Bus-Master* werden können.

Die Vergabe der *Bus-Master*-Funktion an einen Bewerber heisst **Buszuteilung (*bus arbitration*).**

Eigenschaften der DMA-Technik

Bessere Datenrate: mehrere Datenworte *im Ganzen* über den Bus übertragen (***block mode***), ohne neue Buszuteilung.

Besonders sinnvoll, falls die Geräte selbst Puffer haben (z.B. *track buffer* bei Plattenlaufwerken).



Zu programmieren:

- Initialisierung der DMA-Hardware
- Ende-Behandlung
- Fehlerbehandlung

☞ Aus Programmierersicht ist DMA die einfachste Technik.

Zusammenfassung

Ablauf der Kommunikation

Techniken:

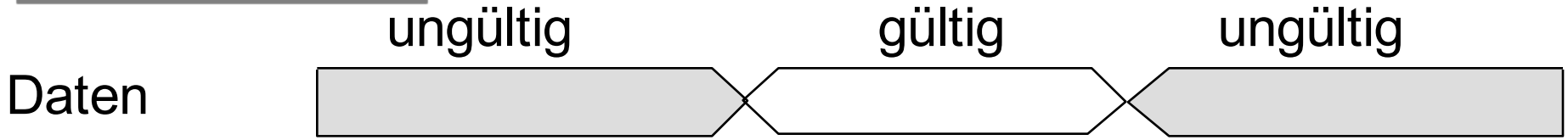
1. Direkte Ein-/Ausgabe (*Immediate devices*)
2. Status-Methode (*busy waiting*)
3. *Polling*
4. *Interrupts*
5. *Direct memory access* (DMA)

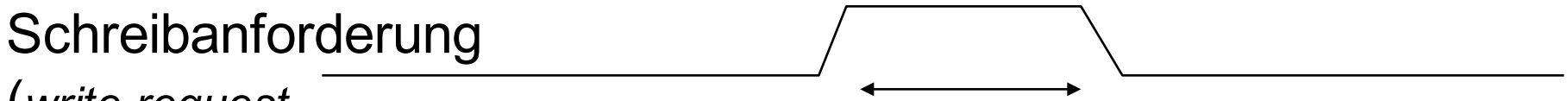
Anhang

Schreiben beim synchronen Bus

Sender → Empfänger

Prinzip:



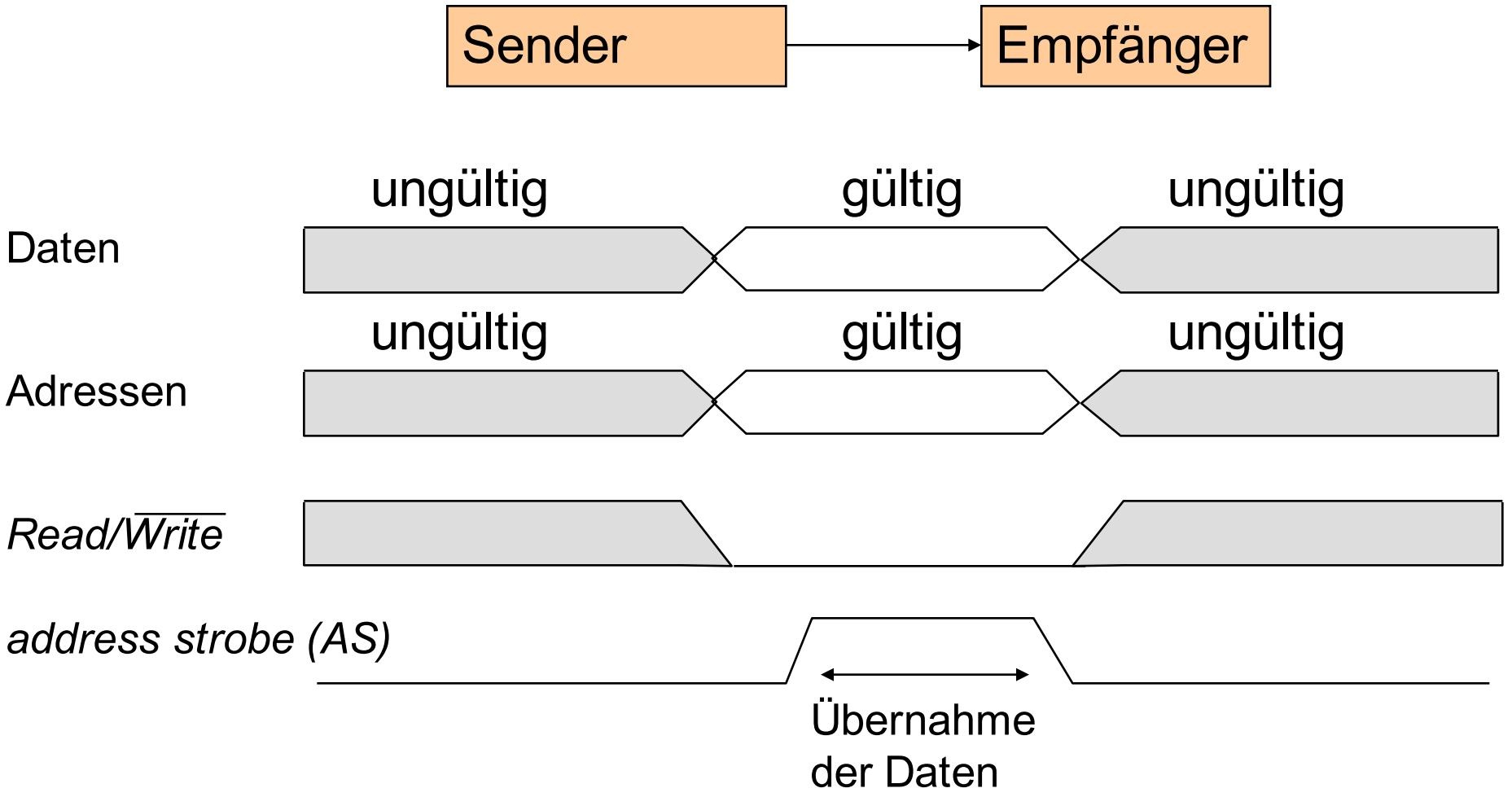


Übernahme der Daten innerhalb maximaler Zeit nach Flanke der Schreibanforderung

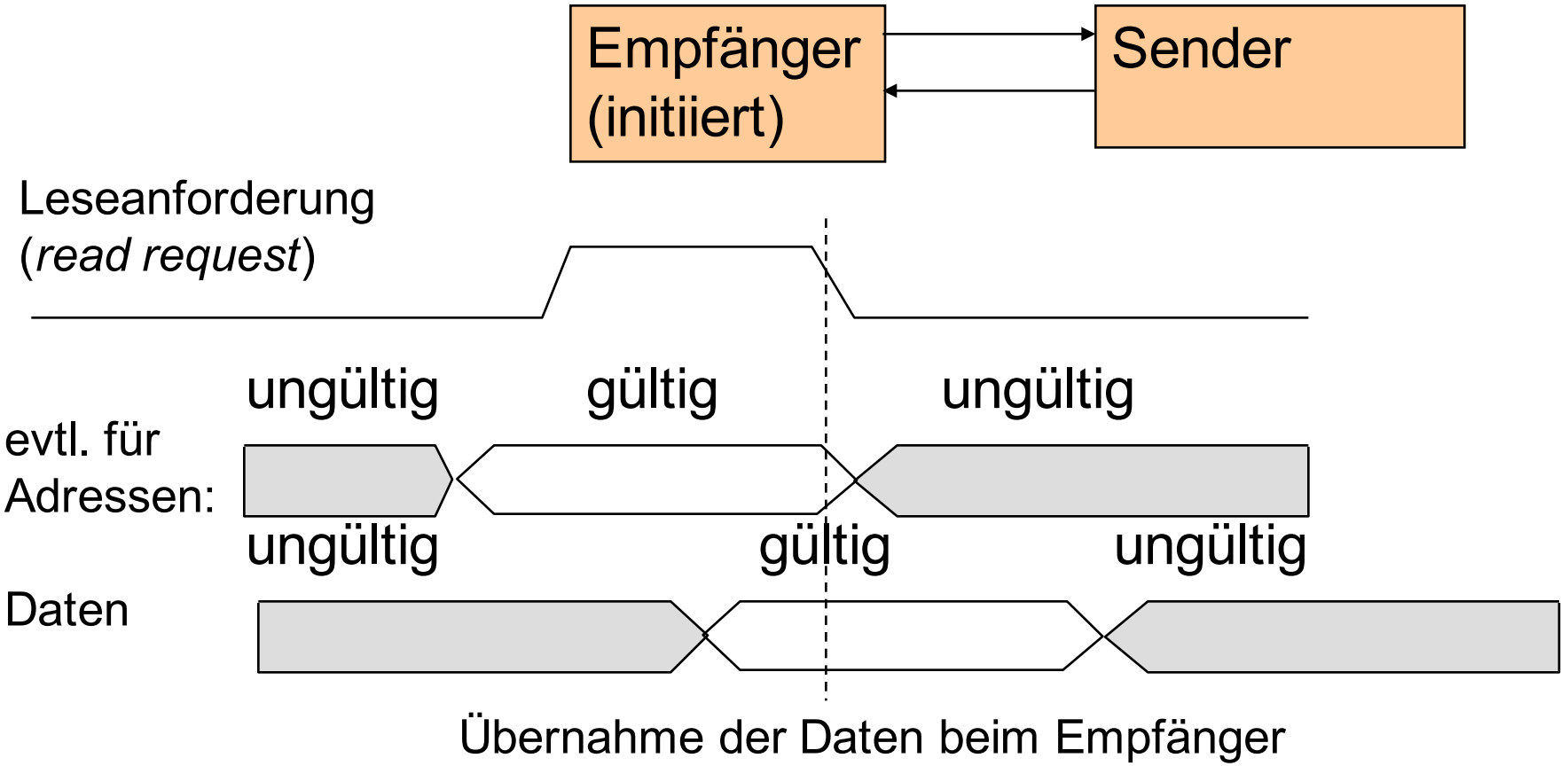
Sender verlässt sich darauf, dass der Empfänger die Daten annimmt.

Alle Signale laufen vom Sender zum Empfänger ☞ schnell.

Schreiben beim synchronen Bus
- mit Adressleitungen -

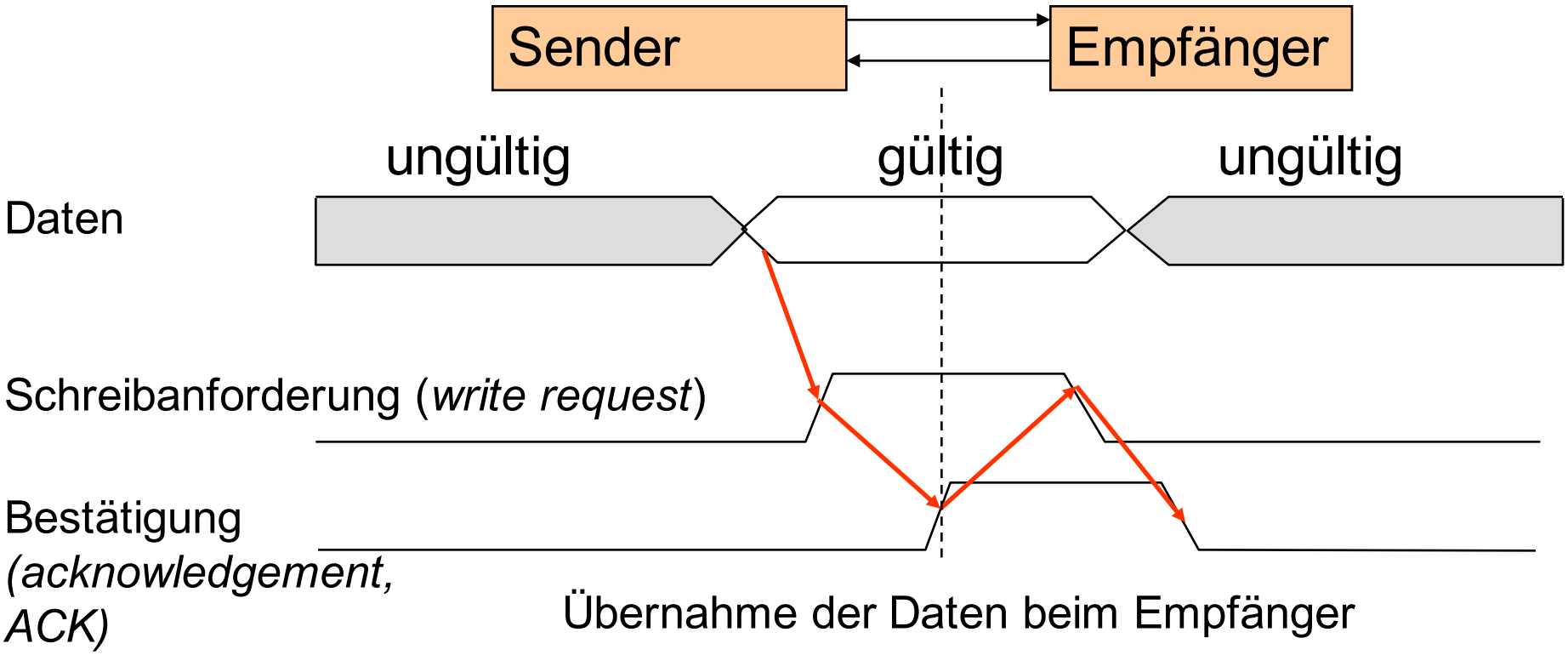


Lesen beim synchronen Bus



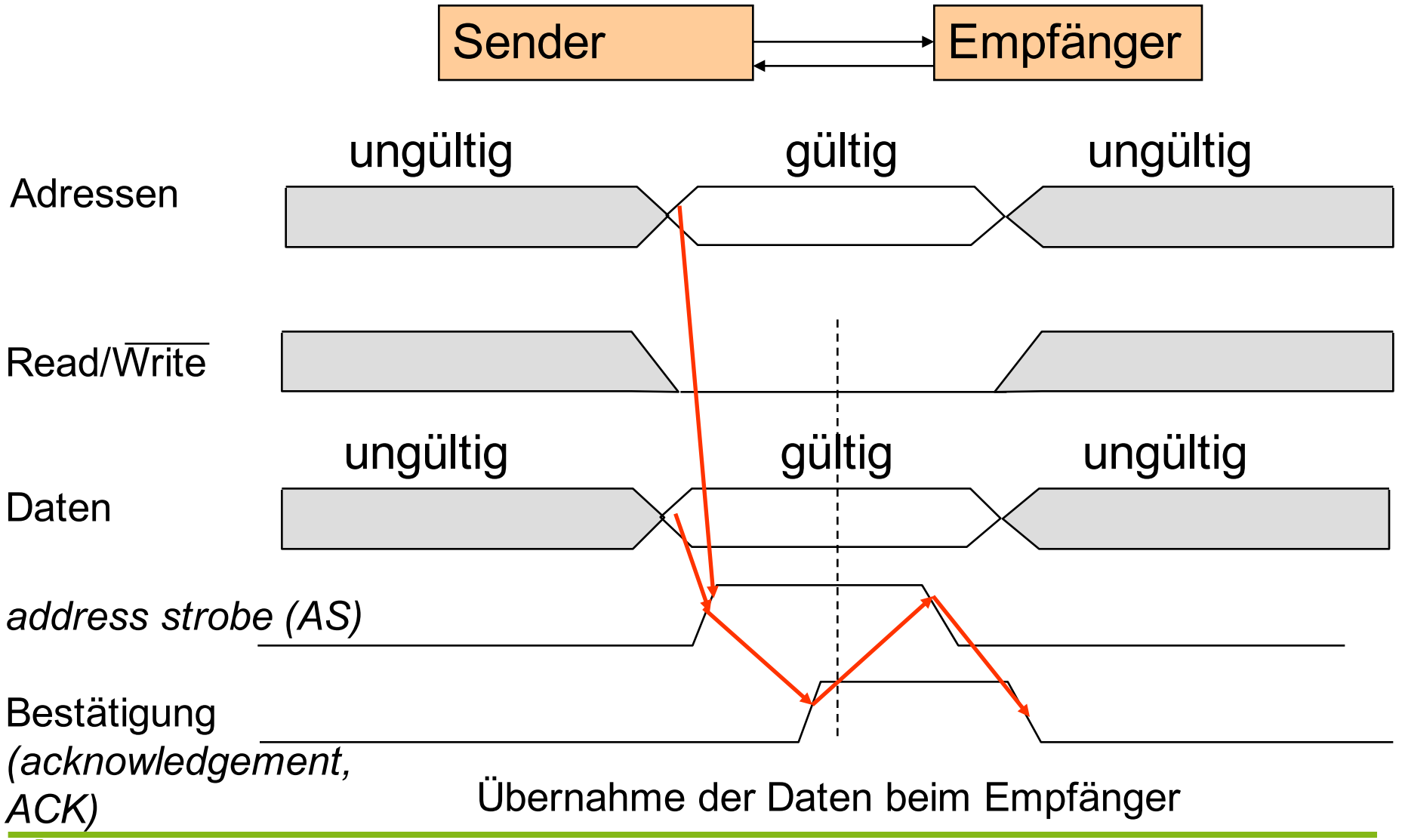
Empfänger und Sender verlassen sich auf verabredete Zeiten.
2 Signallaufzeiten.

Schreiben beim asynchronen Bus

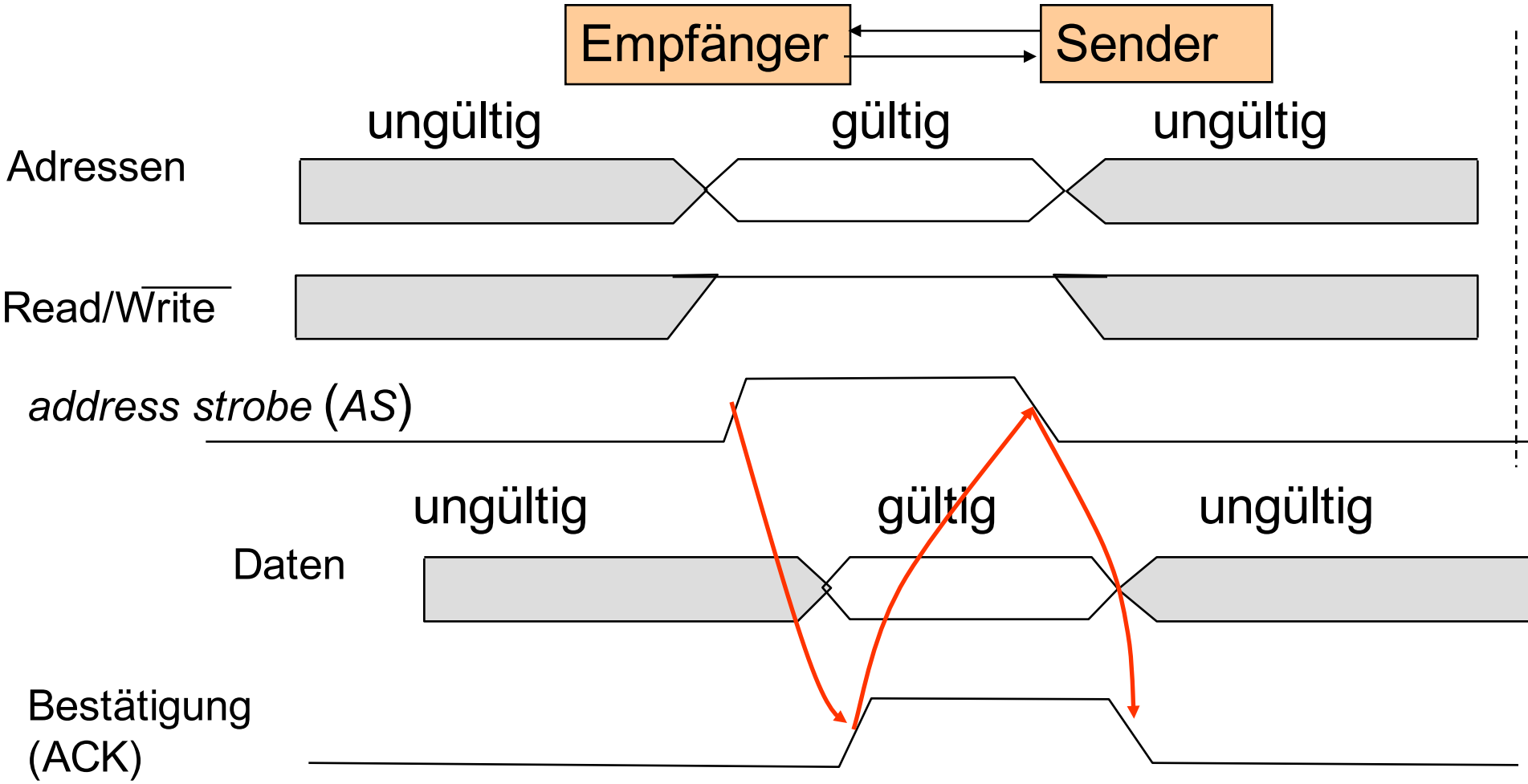


Der Sender muss Daten gültig halten, bis Bestätigung eintrifft.
Bis zu 4 Signallaufzeiten ☞ langsam.

Schreiben beim asynchronen Bus
- mit Adressleitungen -

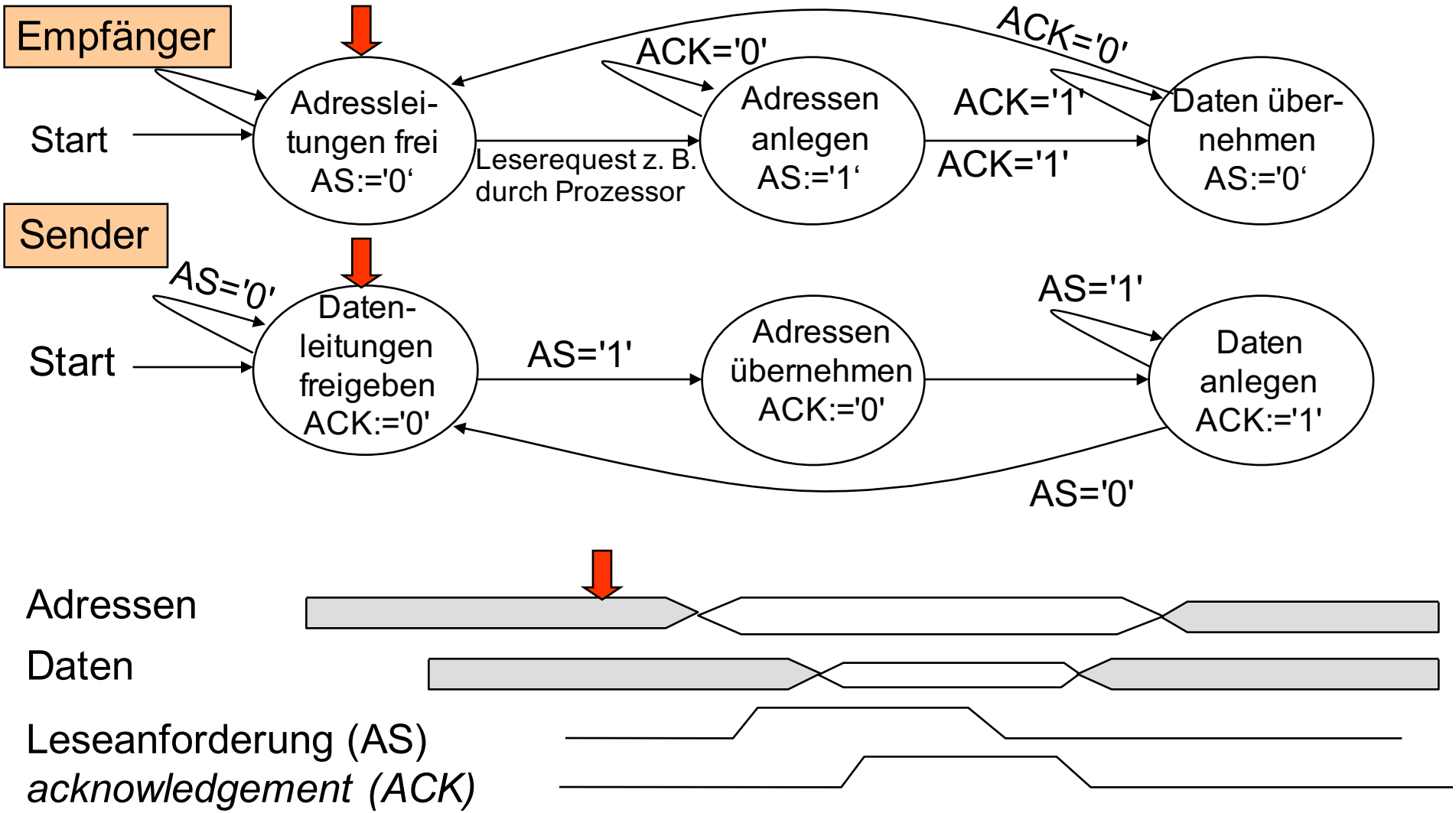


Lesen beim asynchronen Bus
- mit Adressleitungen -

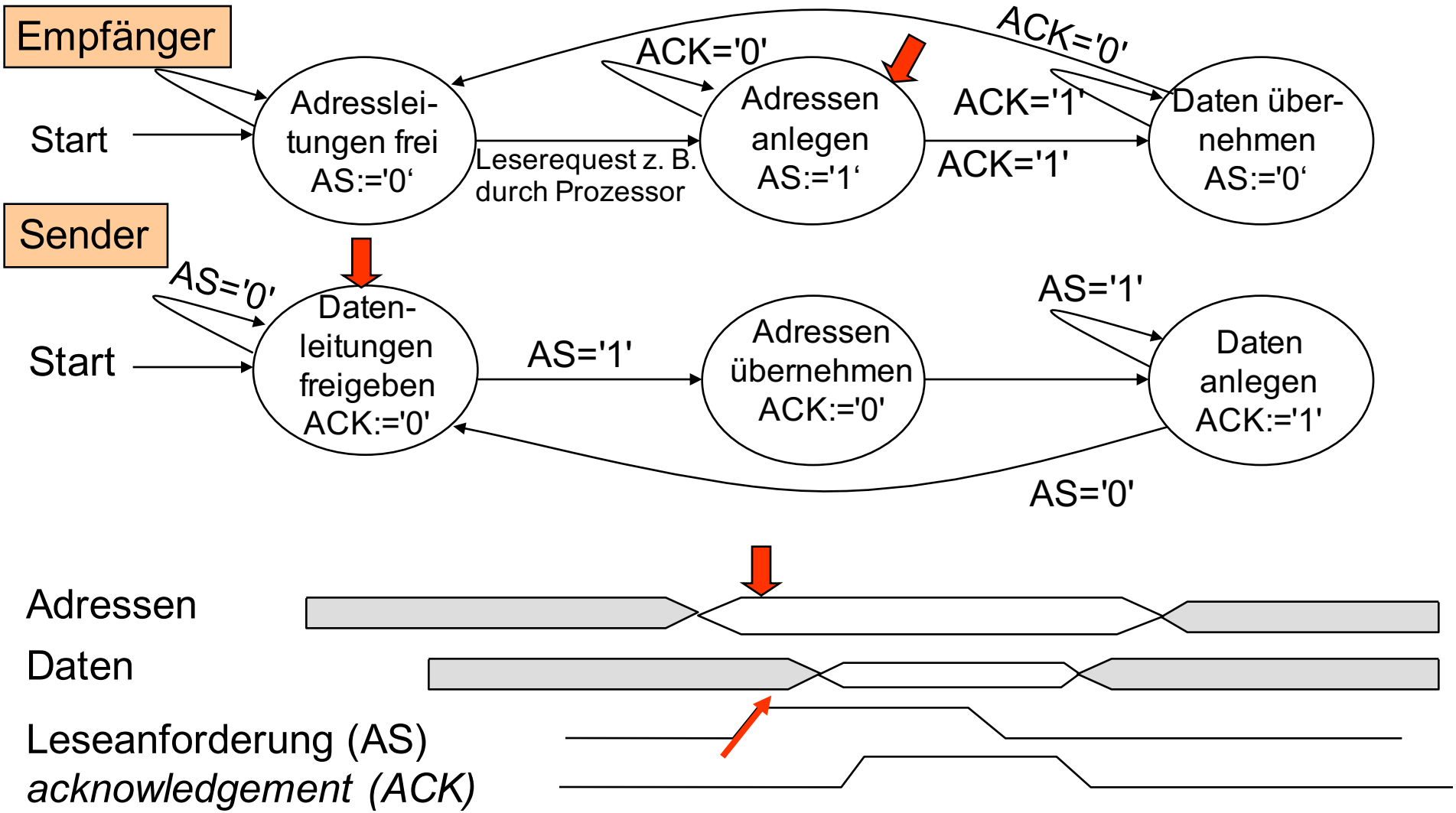


Sender muss Daten gültig halten, bis Bestätigung eintrifft. Mehrere Laufzeiten.

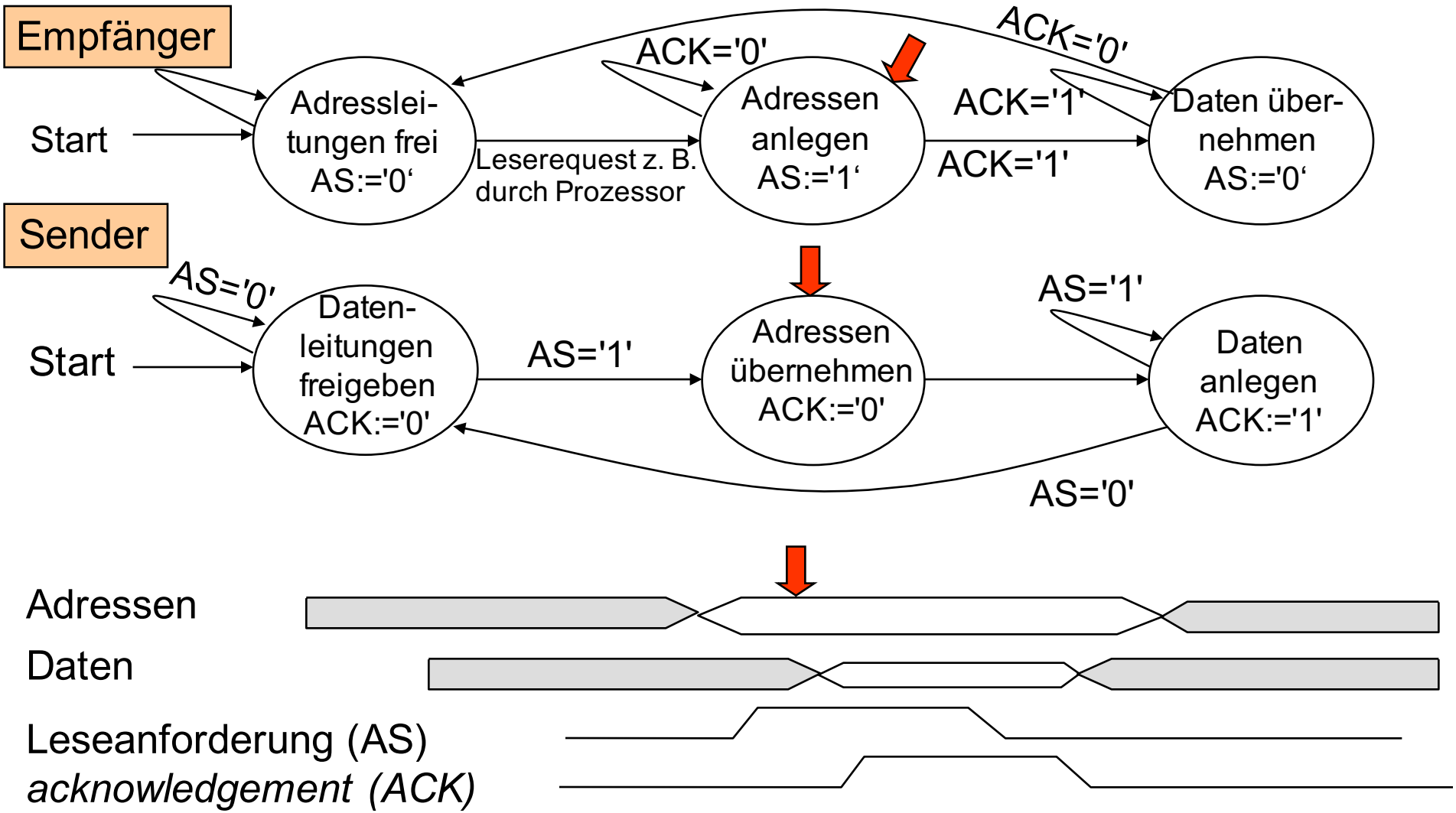
Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (1)



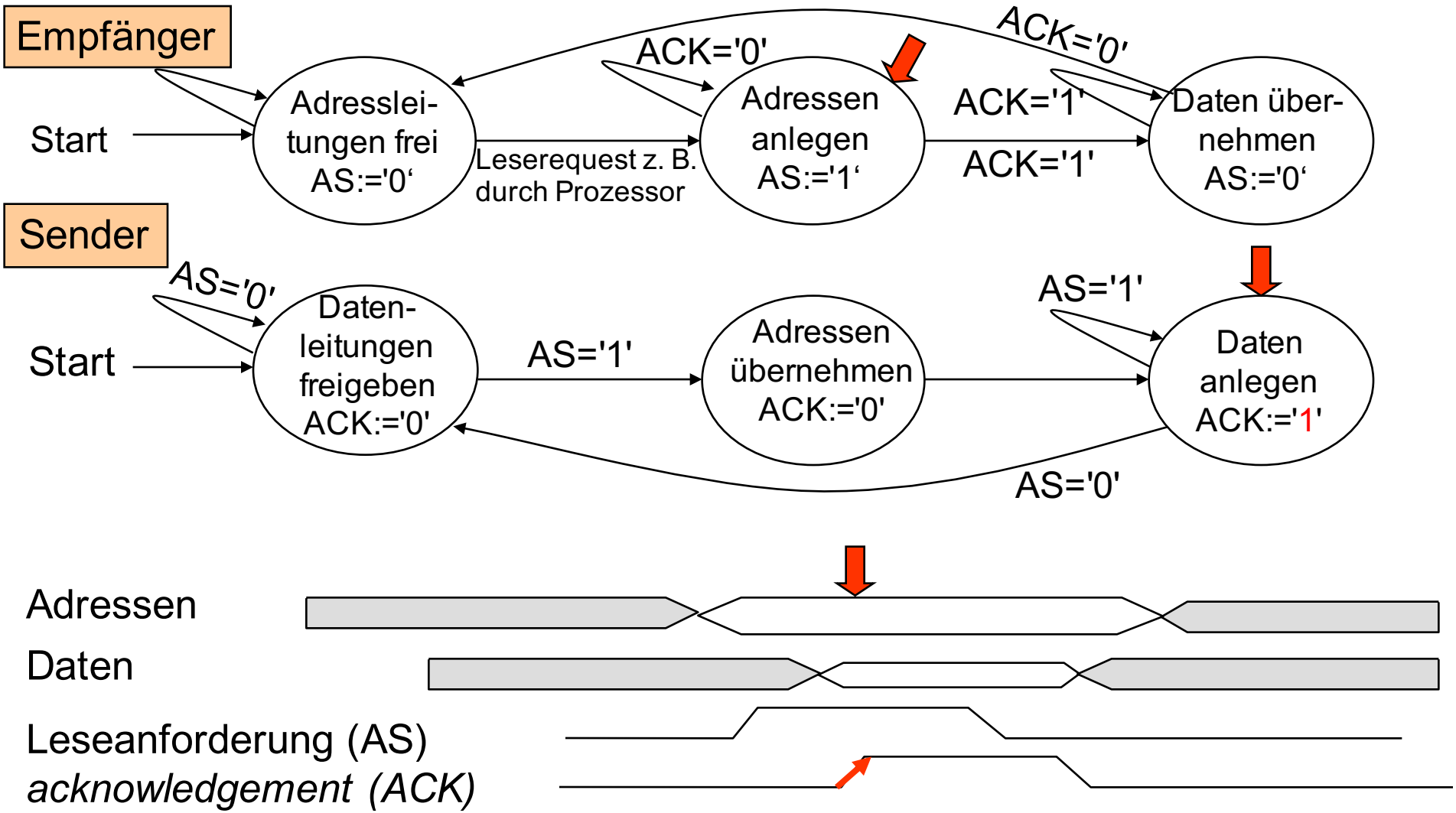
Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (2)



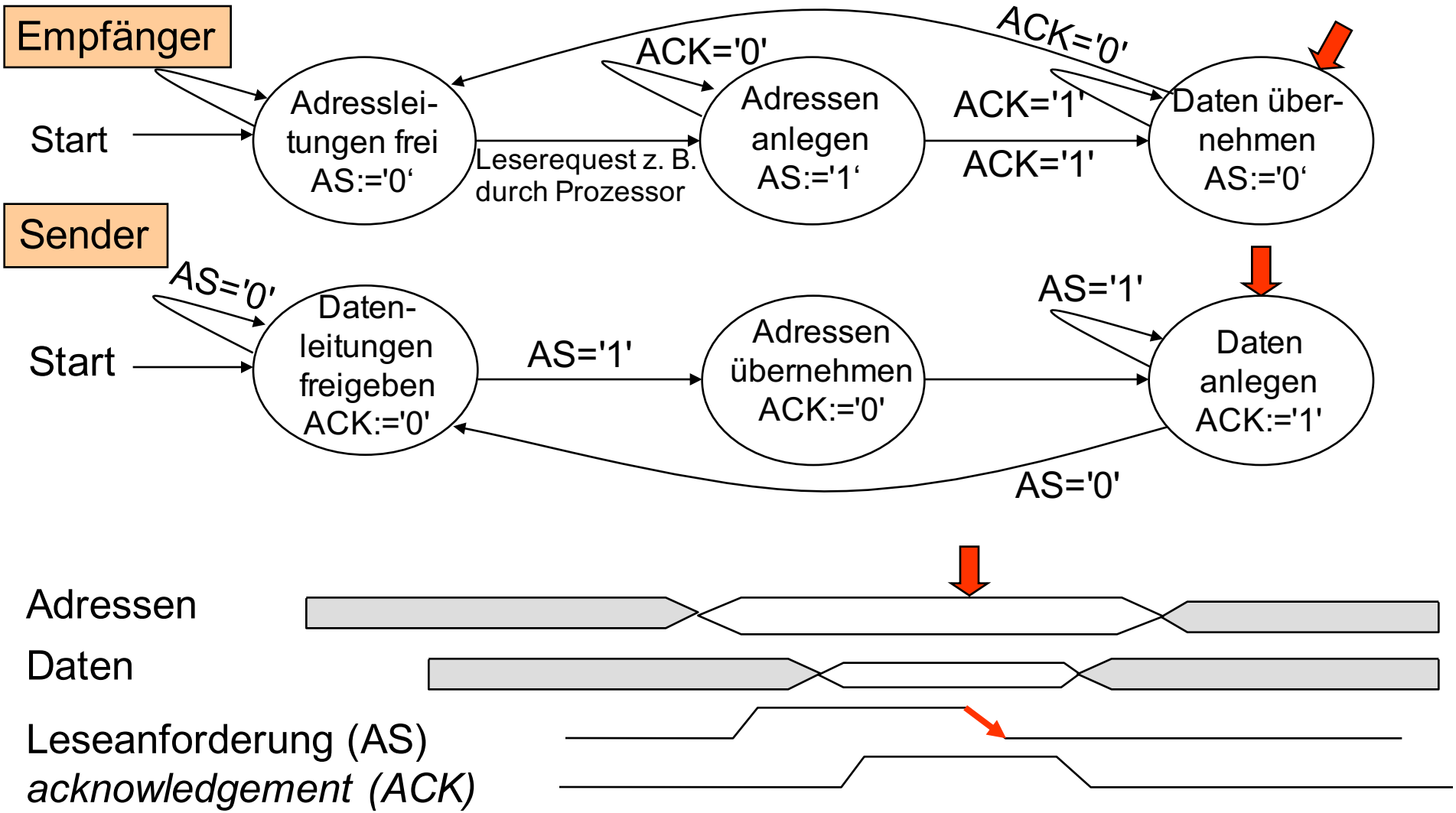
Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (3)



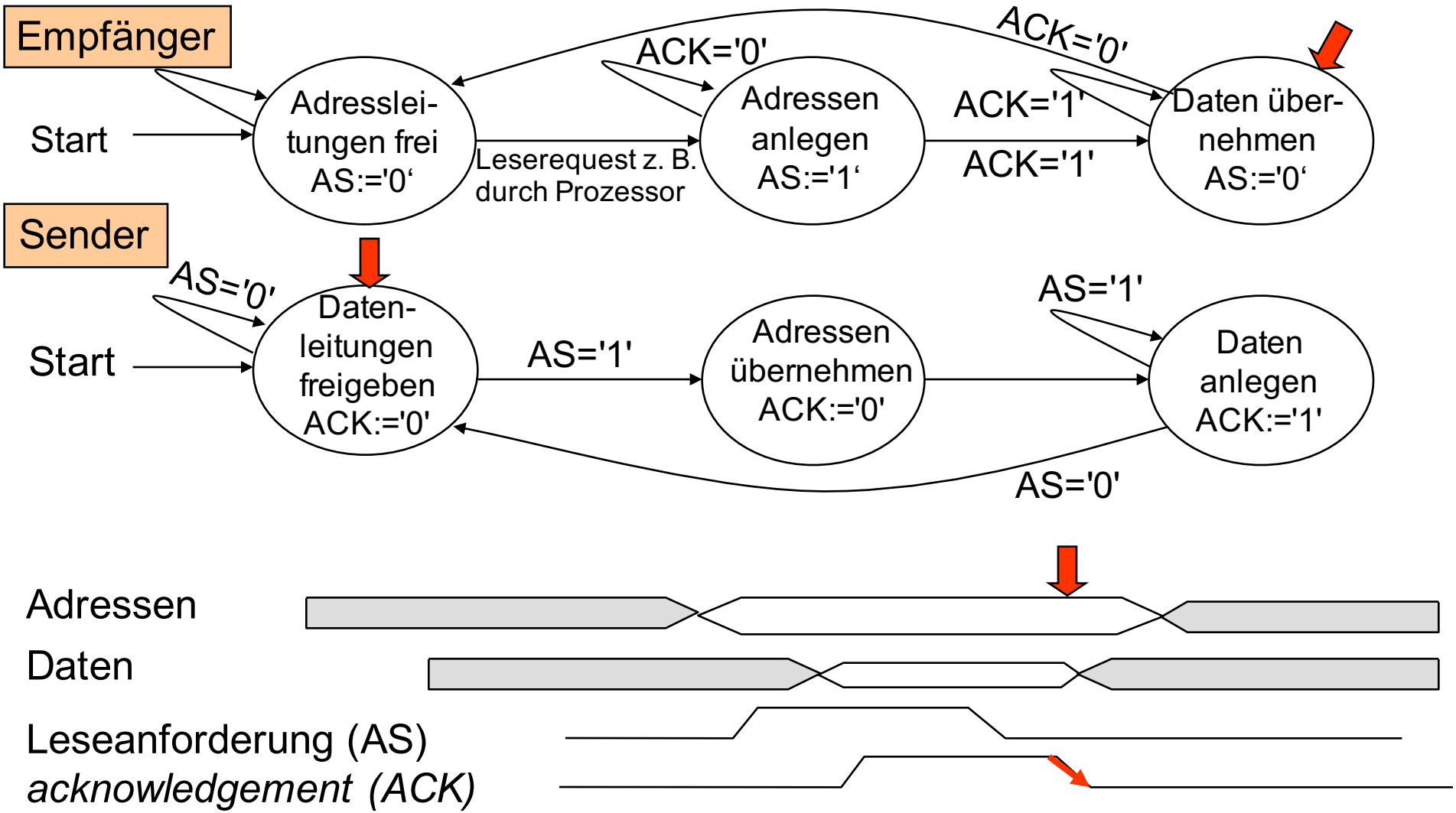
Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (4)



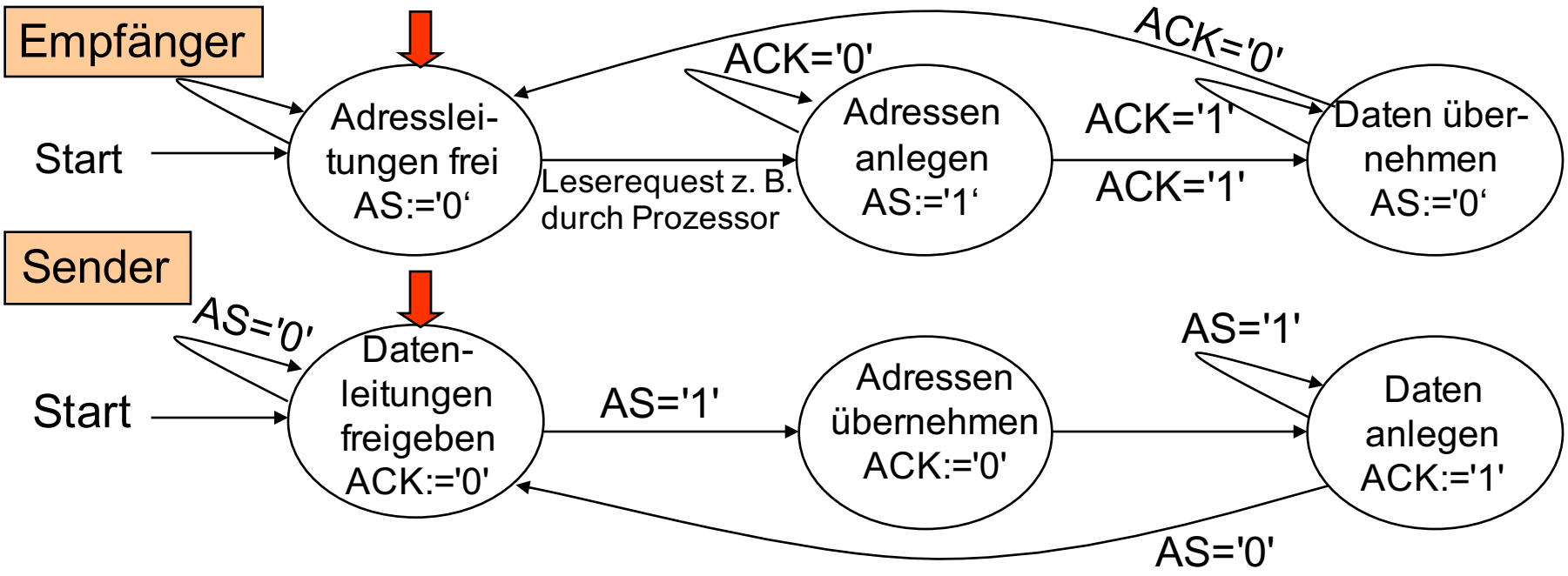
Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (5)

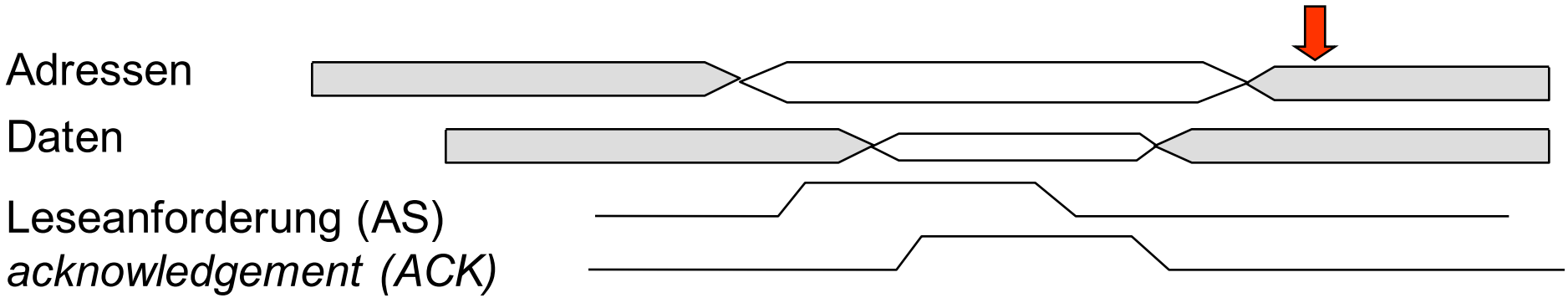


Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (6)



Beispielhaftes Zustandsdiagramm eines asynchronen Busses für das Lesen (7)





Buszuteilung

Wie entscheidet man bei der Busvergabe im Fall von mehreren Wettbewerbern?

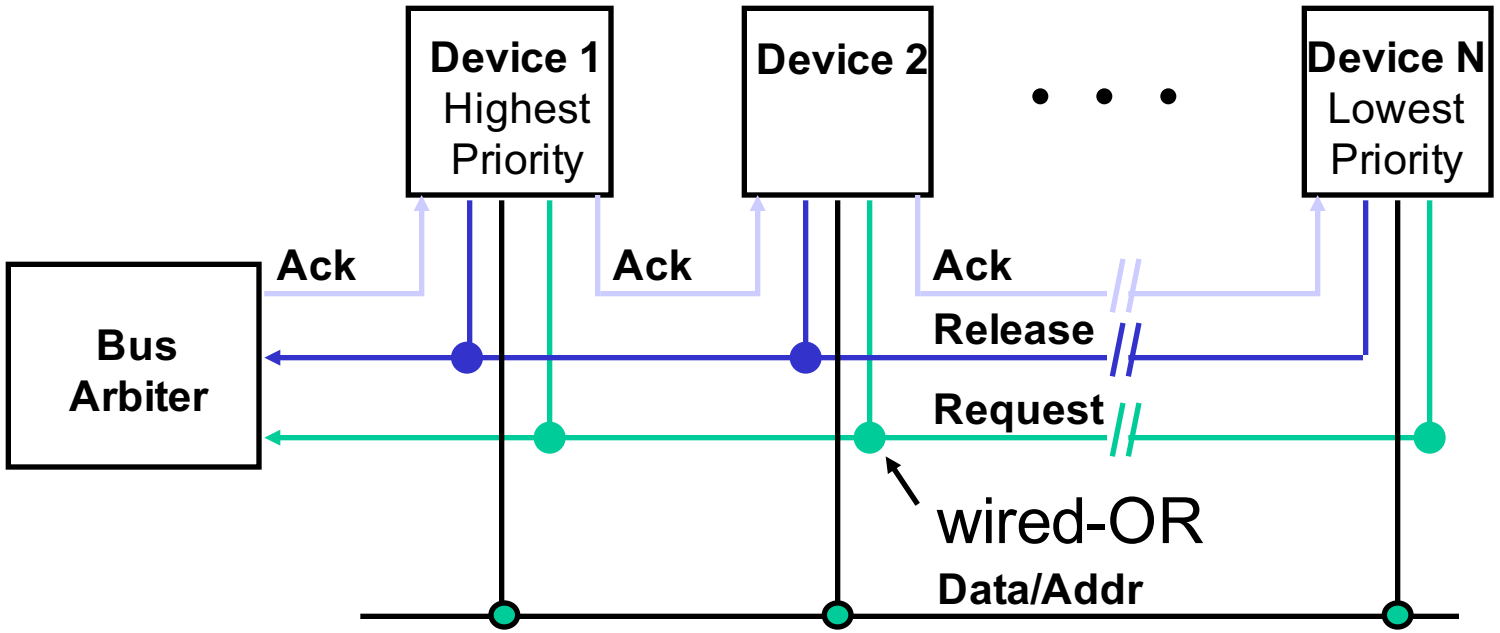
Methoden der Buszuteilung (*bus arbitration*):

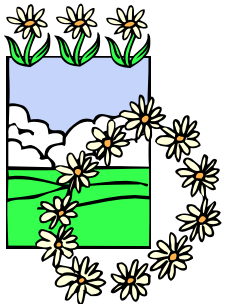
1. *Daisy chaining* (Gänseblümchen-Verkettung)
2. *Centralized, parallel arbitration*
3. *Distributed arbitration by self selection*
4. *Distributed arbitration by collision detection*
5. *Token ring, token bus*
   (Abschnitt 2.5.4.4 im Skript)

Siehe auch: Hennessy/ Patterson

Daisy chaining (Gänseblümchen-Verkettung)

Relative Position der *Controller* am Bus bestimmt Zuteilung.

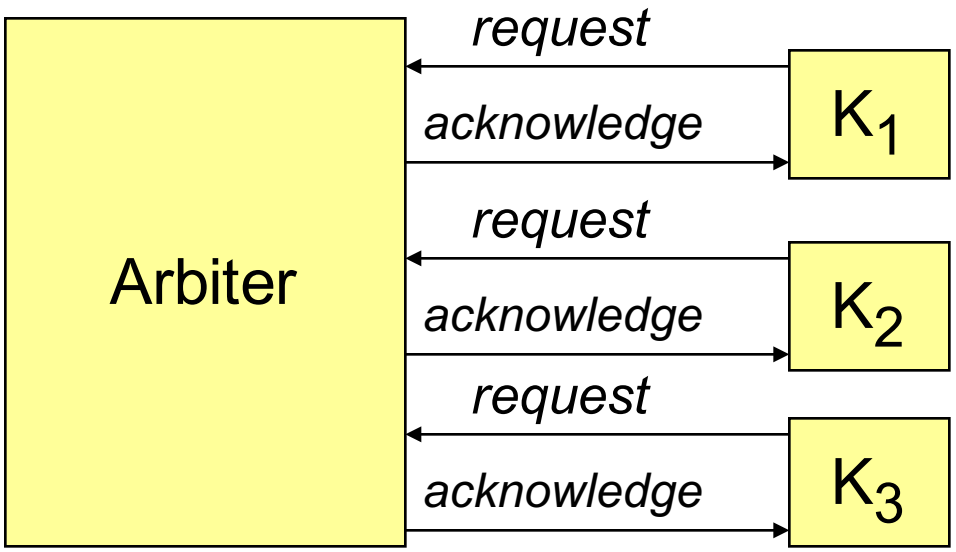




Daisy chaining: einfach, aber nicht fair (verhungern möglich)
Lange Busleitungen für hohe Performanz ungeeignet

Centralized, parallel arbitration

... multiple request lines,
... devices independently request the bus [HP95]

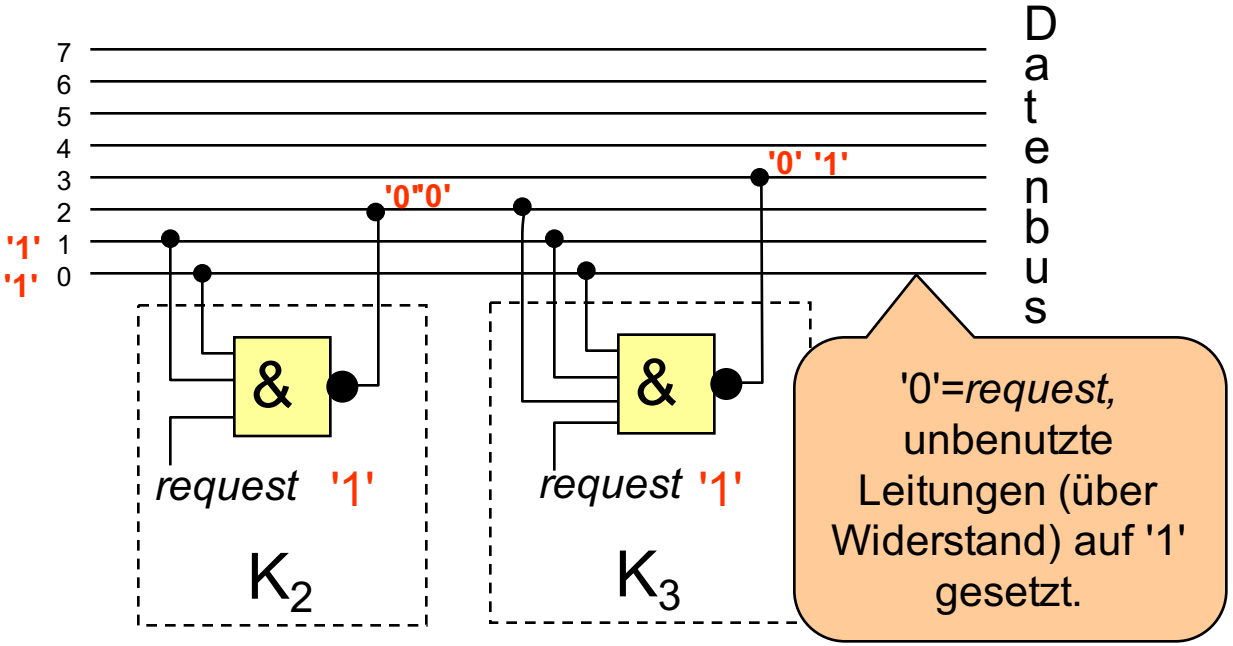


Central arbiter ... may become the bottleneck for bus usage

[HP95]

Distributed arbitration by self selection

These schemes also use multiple request lines, but the devices requesting bus access determine who will be granted access. [HP96]

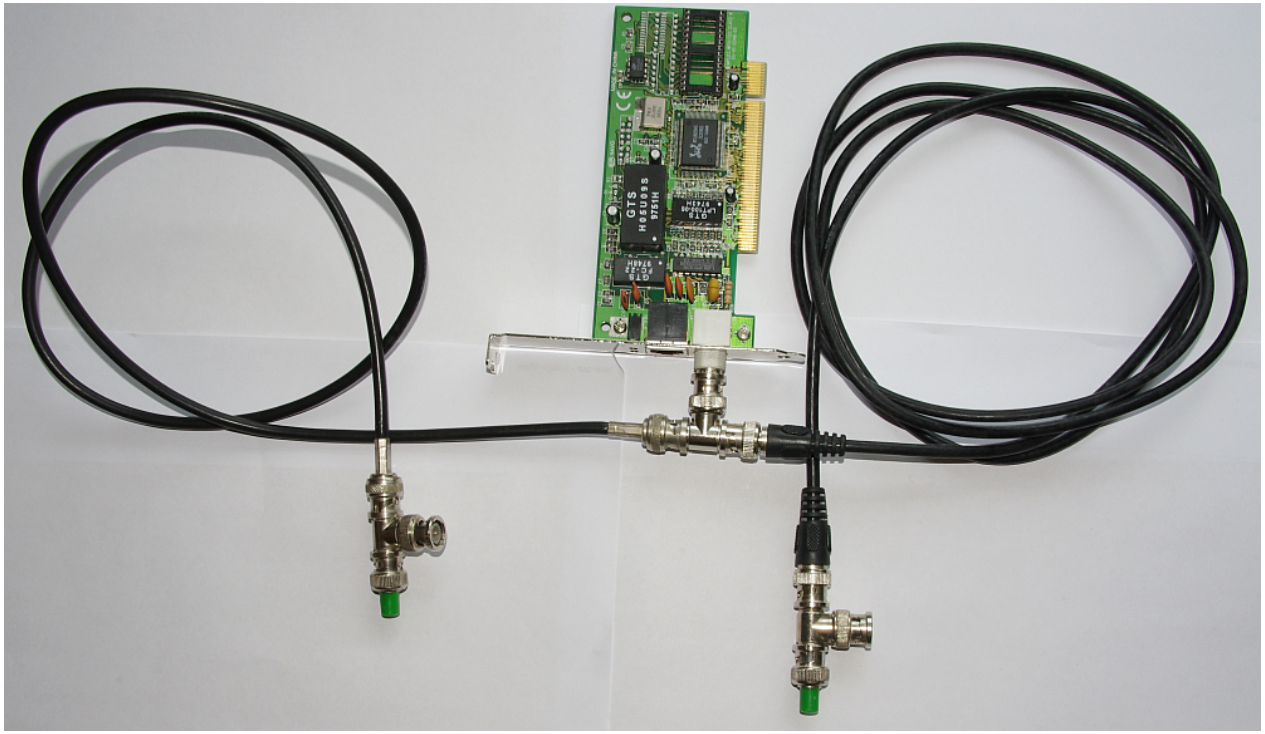


Phase, in der Datenbus für *arbitration* benutzt wird. Ein eigener *request* erscheint nur dann als '0' auf dem Bus, wenn kein Wunsch einer höheren Priorität auf dem Bus angemeldet ist.

Max. Anzahl der *controller* = Anzahl der Datenleitungen.
Anwendungsbeispiel: SCSI. Lange Busleitungen ☞ langsam

Distributed arbitration by collision detection

In this scheme, each device independently requests the bus. Multiple simultaneous requests result in a collision. The collision is detected and a scheme for selecting among the colliding parties is used. [HP95]



Zusammenfassung

Methoden der Buszuteilung (*bus arbitration*):

1. *Daisy chaining* (Gänseblümchen-Verkettung)
2. *Centralized, parallel arbitration*
3. *Distributed arbitration by self selection*
4. *Distributed arbitration by collision detection*
5. *Token ring, token bus,* weitere LAN-Verfahren (siehe Abschnitt 2.5.4.4)

Charakterisierung von Bussen durch

- Übertragungsrate, *split transactions*, Multiplexing, u.a.