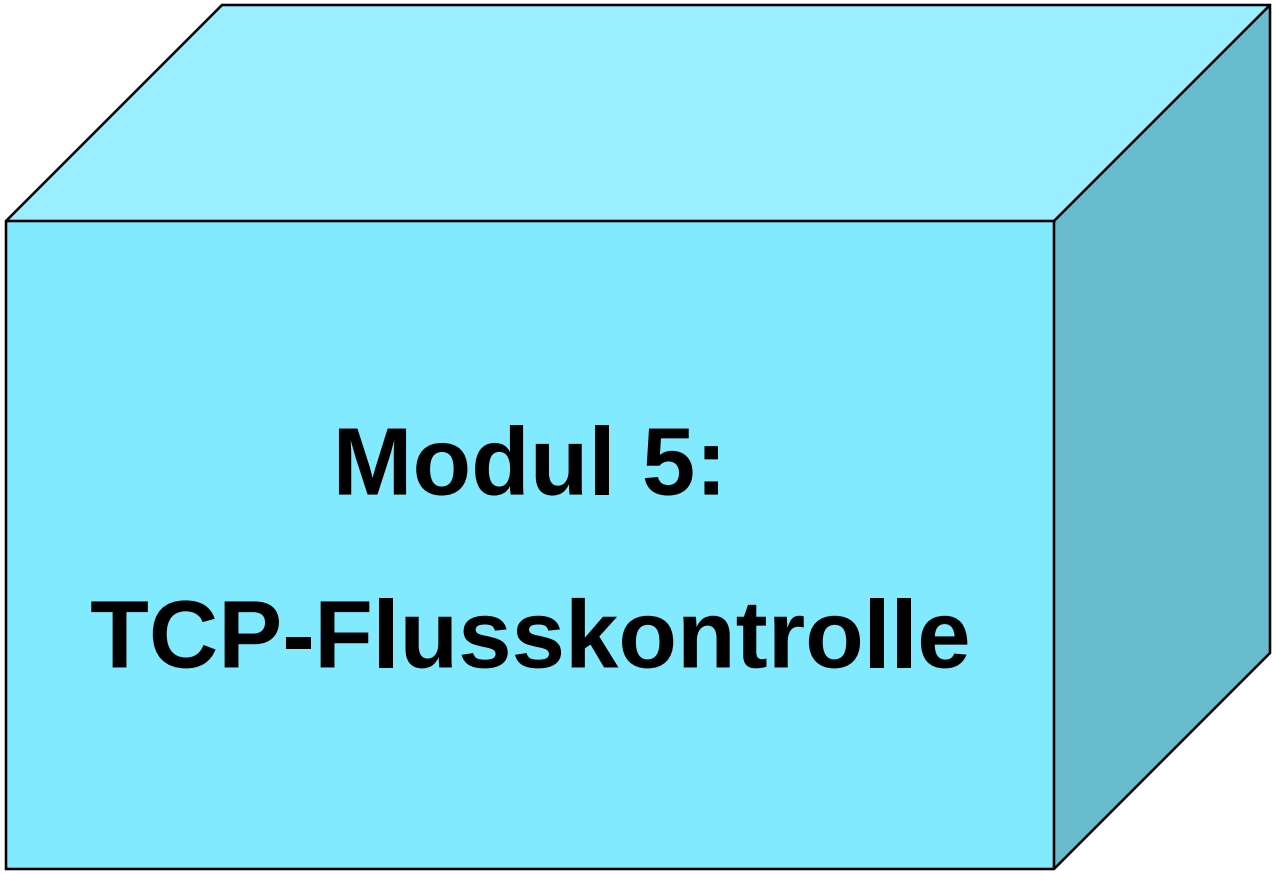
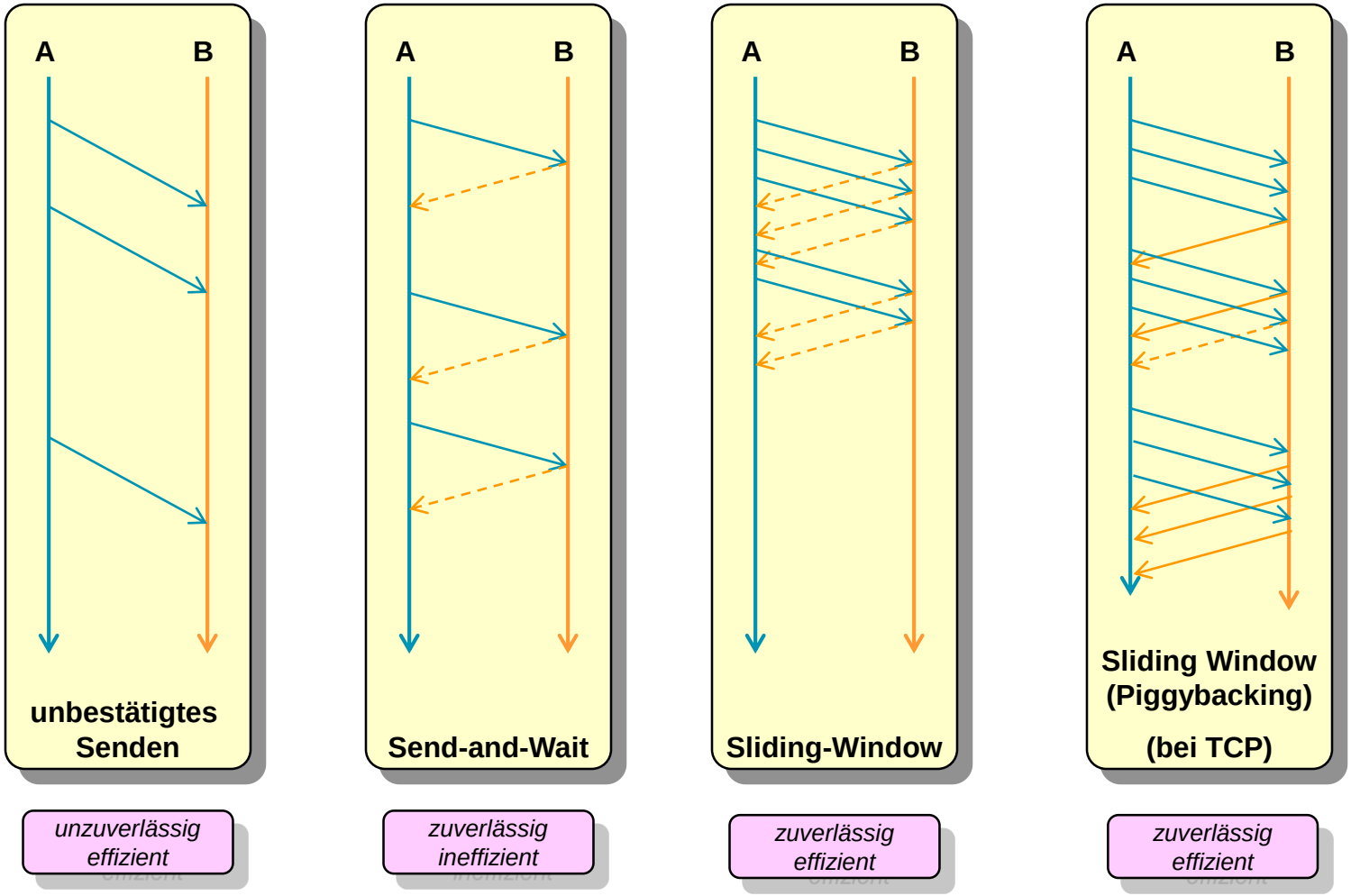
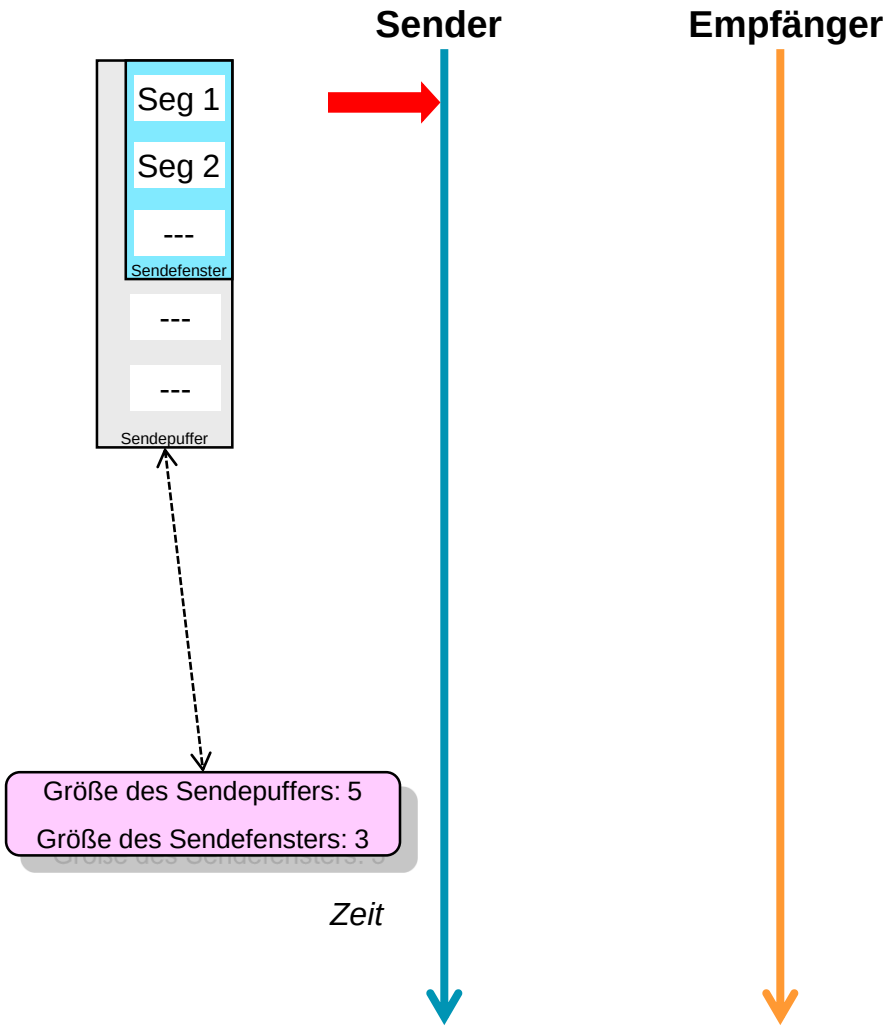


Prinzip des Sliding-Window: Zuverlässigkeit + Effizienz



Sliding-Window - detaillierter

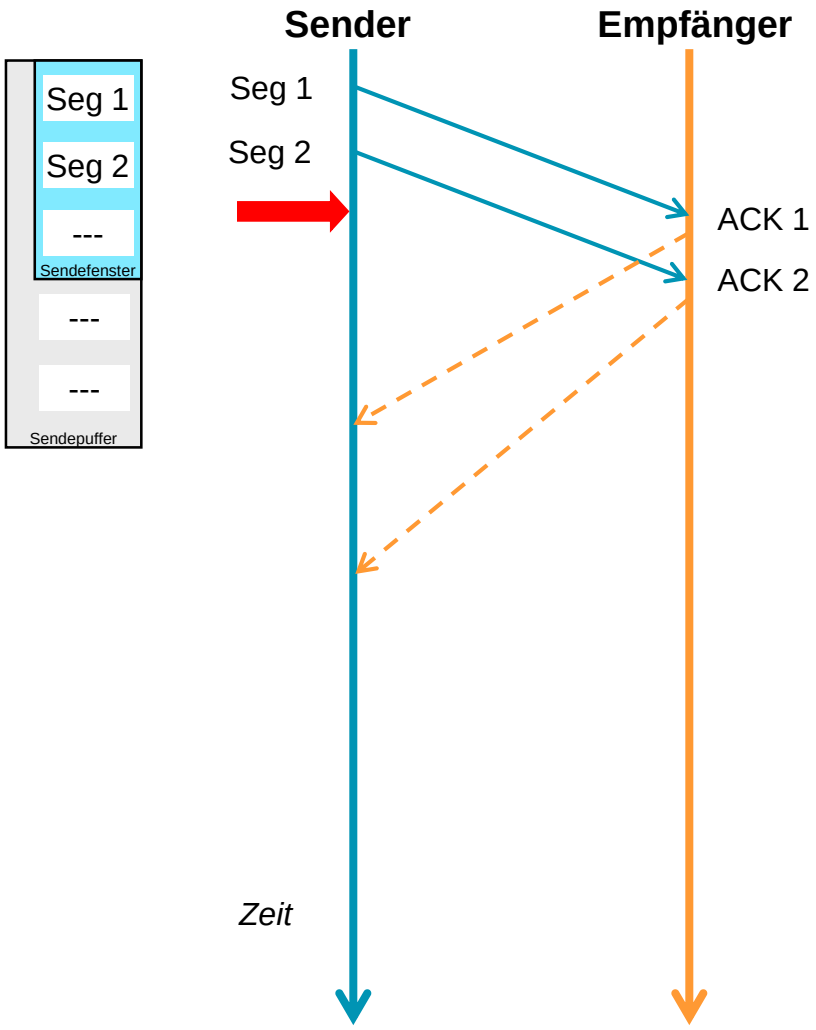


- **Anwendung produziert 2 Segmente**
- **Diese Segmente liegen im Sendepuffer und im Sendefenster**

Was ist der Unterschied zwischen Sendepuffer und Sendefenster?

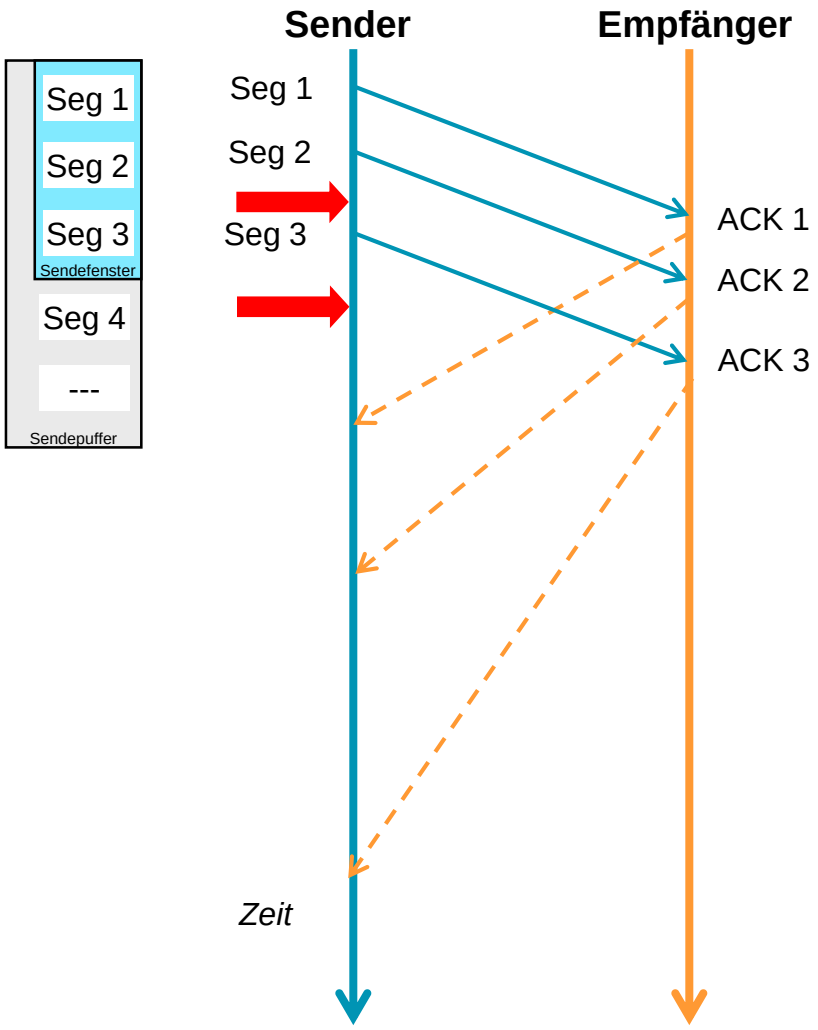
Was ist die Aufgabe von Sendepuffer und Sendefenster?

Sliding-Window - detaillierter



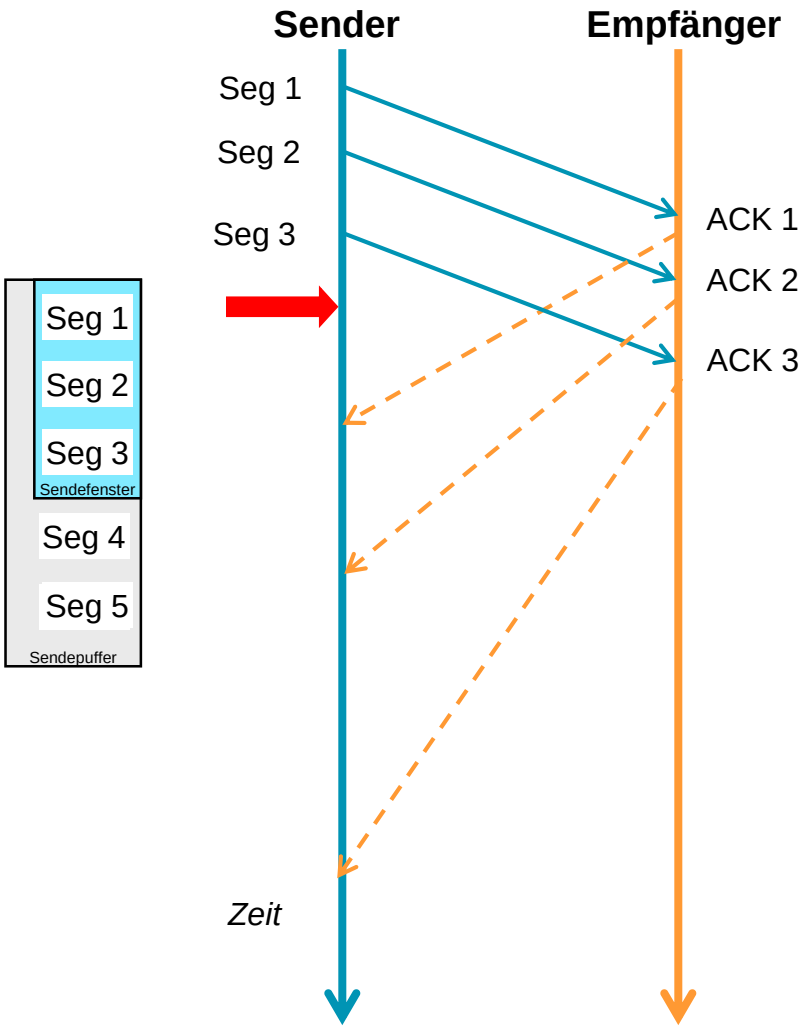
- **Was passiert jetzt??**
- **Beide Segmente können gesendet werden**
- **Auswirkungen auf den Sendepuffer und das Sendefenster?**

Sliding-Window - detaillierter



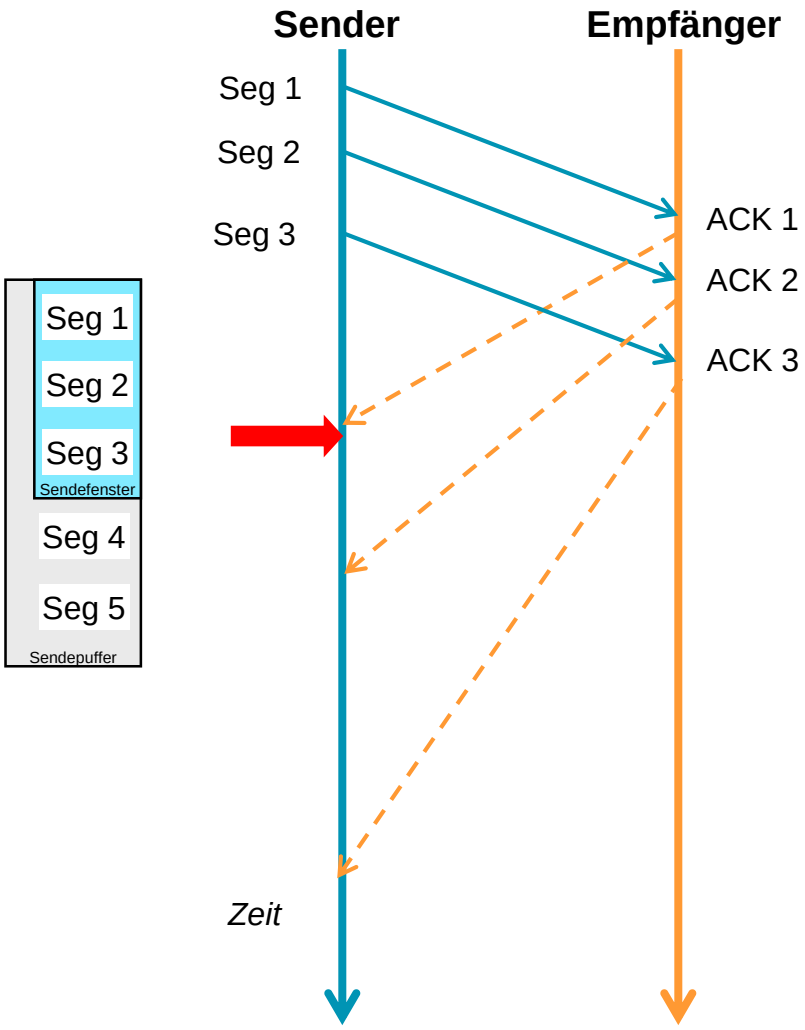
- **Anwendung produziert zwei weitere Segmente**
- **Was macht der Sender?**

Sliding-Window - detaillierter



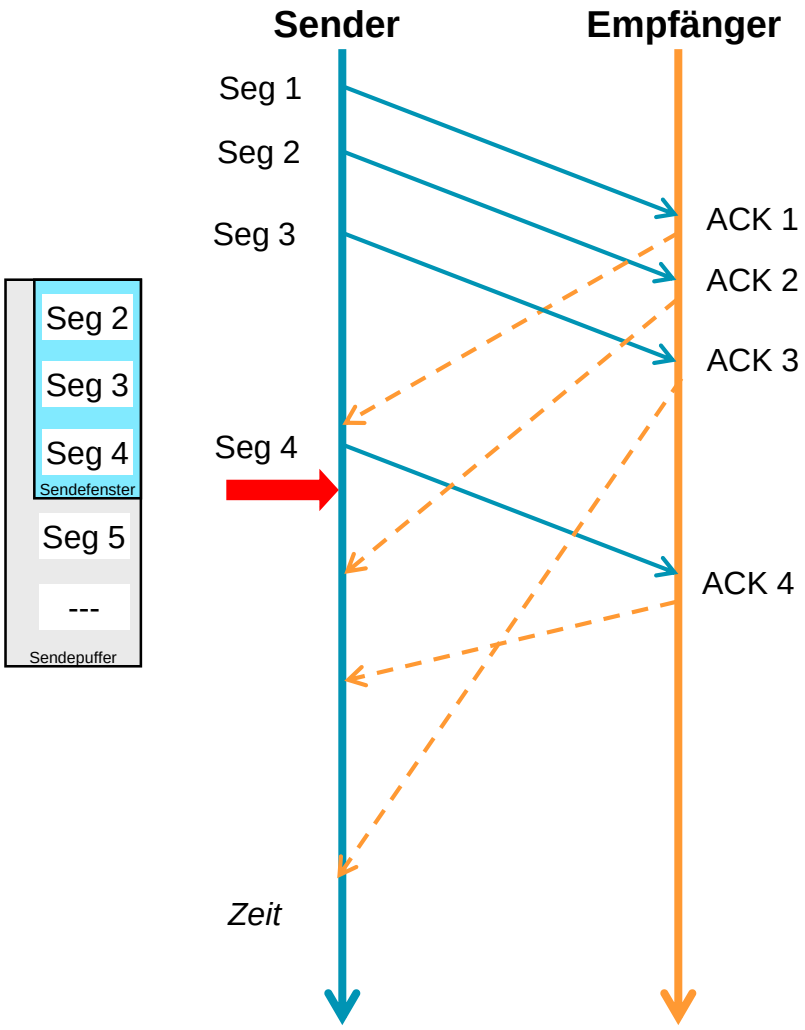
- **Anwendung produziert ein weiteres Segment**
- **Was passiert?**
- **Was wäre, wenn sie zwei weitere Segmente produziert?**

Sliding-Window - detaillierter



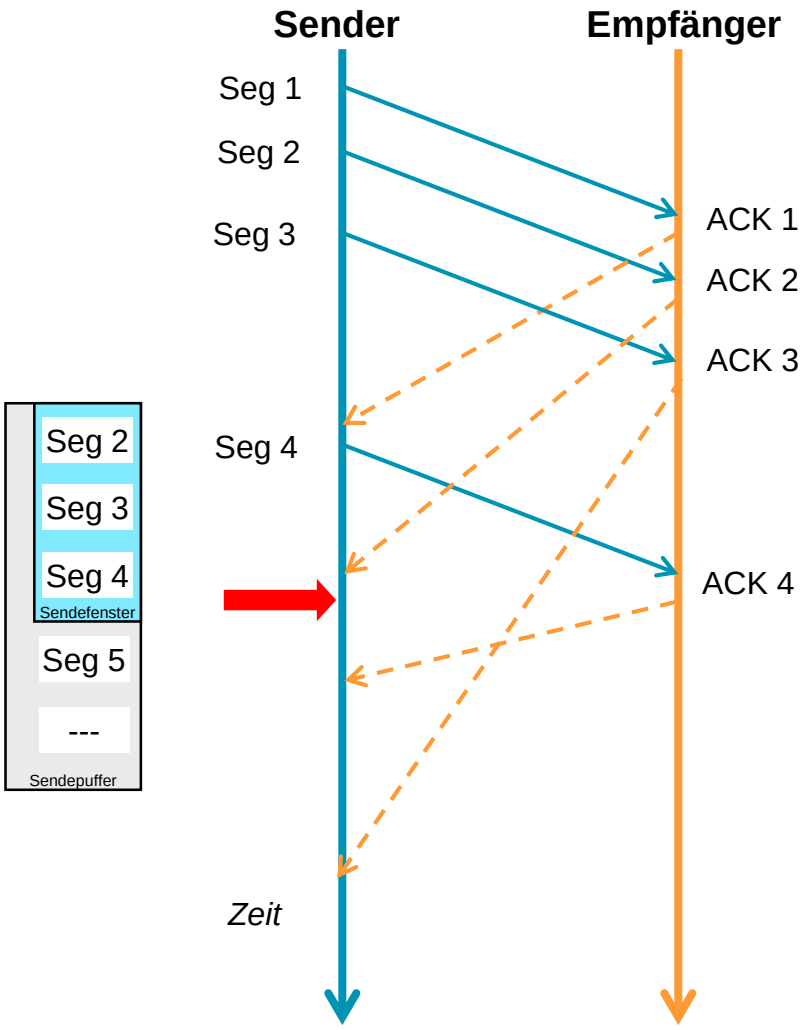
- **ACK 1 kommt nun beim Sender an**
- **Was passiert?**

Sliding-Window - detaillierter



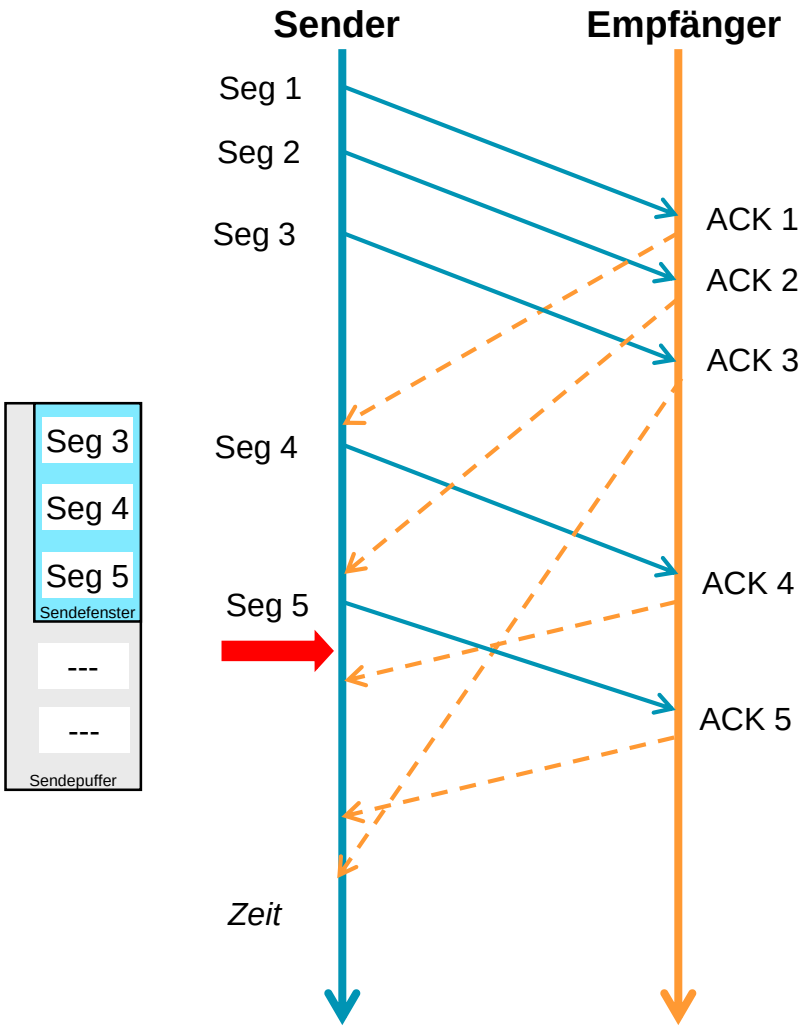
- **Schieben des Fensters**
- **Versenden von Segment 4**

Sliding-Window - detaillierter



- **Nun kommt ACK 2**
- **Was passiert?**

Sliding-Window - detaillierter



- **Schieben des Fensters**
- **Versenden von Seg 5**

Für Fortgeschrittene:

- **Was passiert, wenn ACK 4 und ACK 5 vor ACK 3 kommen?**

Frage an alle:

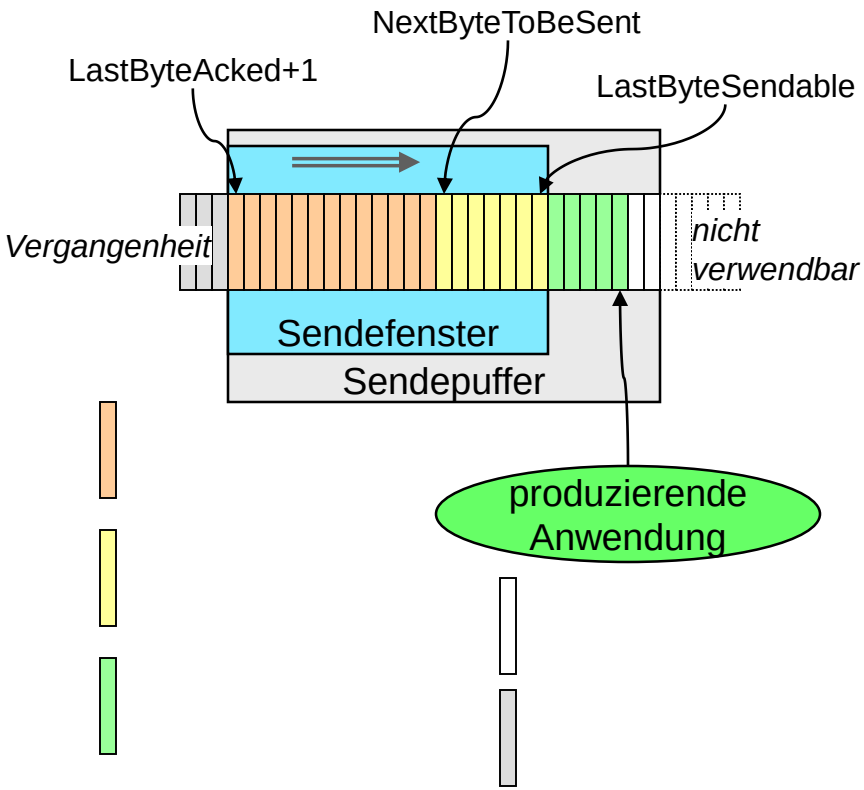
- **Wie groß soll das Sendefenster sein?**
- **Wer legt das Sendefenster fest?**

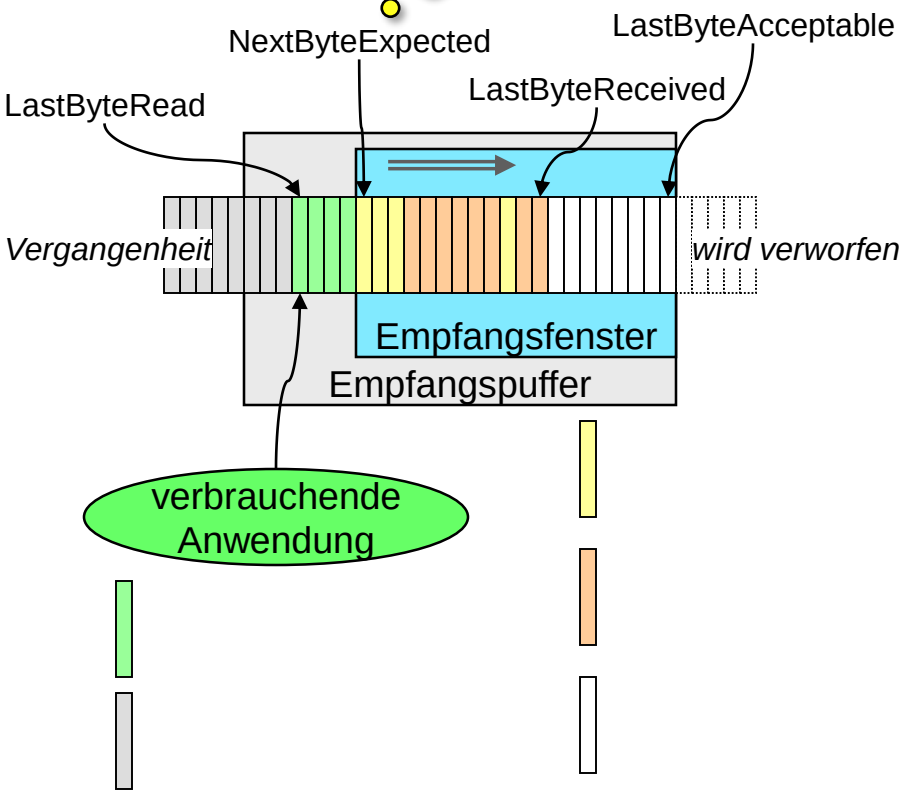
TCP-Sliding-Window – Sende- und Empfangspuffer

Jetzt alles auf Byte-Niveau!

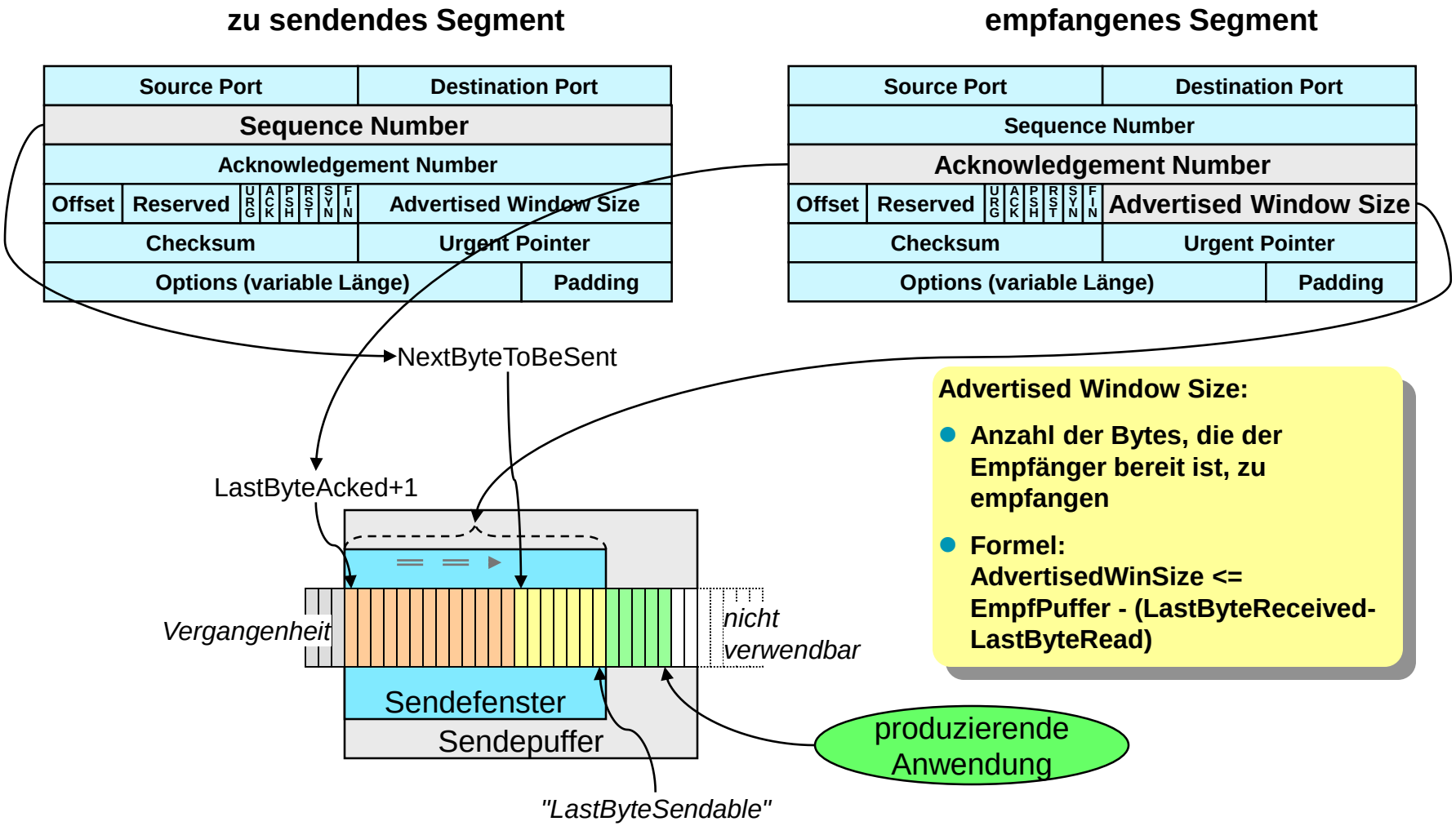
Spezielle Variante des Sliding-Window-Grundalgorithmus

- **dynamische Fluss-Steuerung mittels "Advertised Window"**
- **Einbeziehung der verarbeitenden Anwendungen**





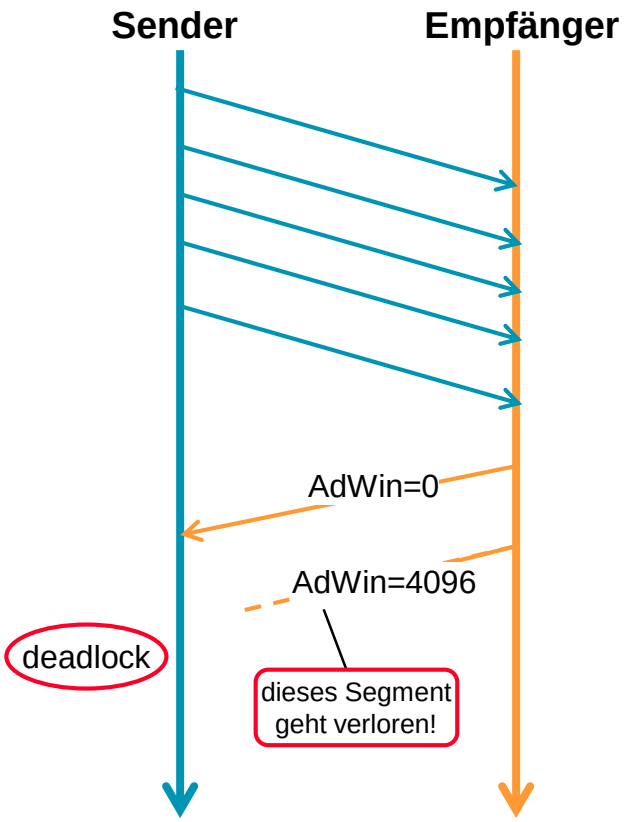
TCP-Flusskontrolle – Segmente auf Senderseite



Advertised Window Size:

- **Anzahl der Bytes, die der Empfänger bereit ist, zu empfangen**
- **Formel: AdvertisedWinSize <= EmpfPuffer - (LastByteReceived-LastByteRead)**

TCP-Flusskontrolle – "Zero advertised window size"



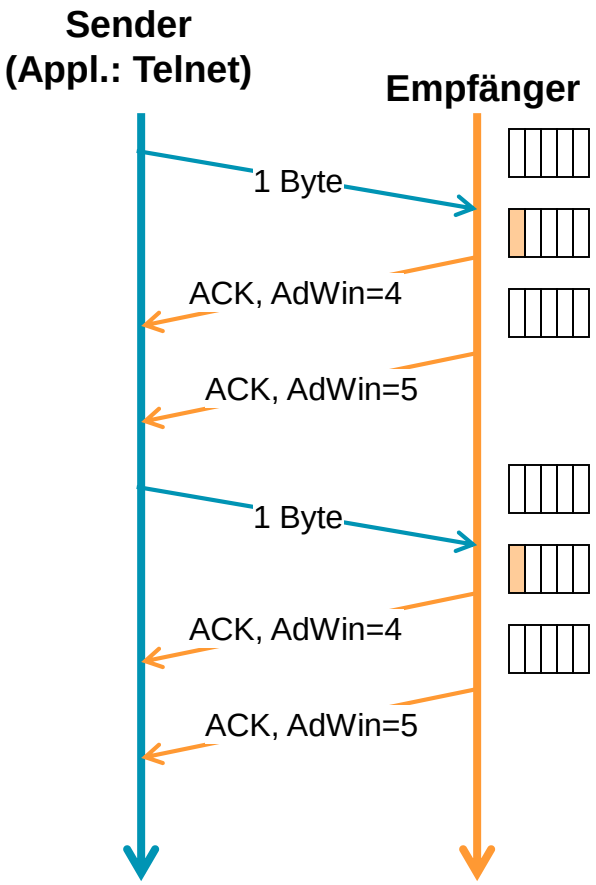
Dargestelltes Problem:

- **Wie kommt es zum Deadlock?**

Lösung des Problems:

- **Sender benutzt einen persistenten Timer, um periodisch ein Zero-Window-Probe-Segment zu senden, sobald das Empfängerfenster geschlossen ist.**
  (Algorithmus: Exponentieller Back-Off-Algorithmus: Startwert 1.5 Sek., Verdopplung nach jedem Ack bis Limit, z.B. 60 s, erreicht.
  → 1,5 3 6 12 24 48 60 60 60 60 60 ……..)
- **Probe-Segment enthält keine Nutzdaten. Spezifikation erlaubt explizit das Senden von Probe Segmenten auch nach geschlossenem Empfängerfenster.**
  (Beachte: solange Empfänger das Probe-Segment nicht verarbeiten kann, enthält das ACK die Sequenz-Nummer des letzten angenommen Bytes)

TCP-Flusskontrolle – "Small Packet Problem"



dargestelltes Problem:

- **Versenden von zwei Byte erzeugt 240 Byte Overhead**
  (2 Datensegmente á 40 Byte,
  2 Acknowledgements á 40 Byte,
  2 Fensteraktualisierungen á 40 Byte)

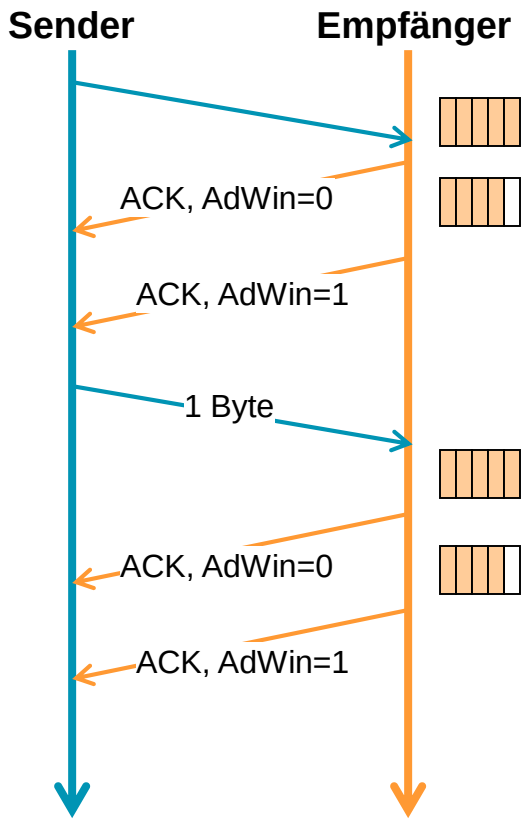
Lösungen:

- **Empfängerseitig: delayed Acknowledgement:**
  Verzögerung von Bestätigung und Fensteraktualisierungen um 200 ms
  (Idee dahinter: „Huckepack“, „Sammeln von ACKs“)
- **Senderseitig: Nagle's Algorithm (RFC 896, 1984):**
  Falls Daten byteweise von Anwendung kommen, sende nur das erste Byte, sammle Bytes auf und sende diese in einem Segment, sobald MSS erreicht oder das erste Byte bestätigt wird.

  Bei IP über LAN wenig Einfluss, aber bei WAN.

  Probleme bei interaktiven Anwendungen (Nagle's Algorithmus führt z.B. bei X-Window zu ruckeliger Arbeit). Daher ist Nagle's Alg. über Sockets abschaltbar.

TCP-Flusskontrolle – "Silly Window Syndrome (SWS) RFC 813"



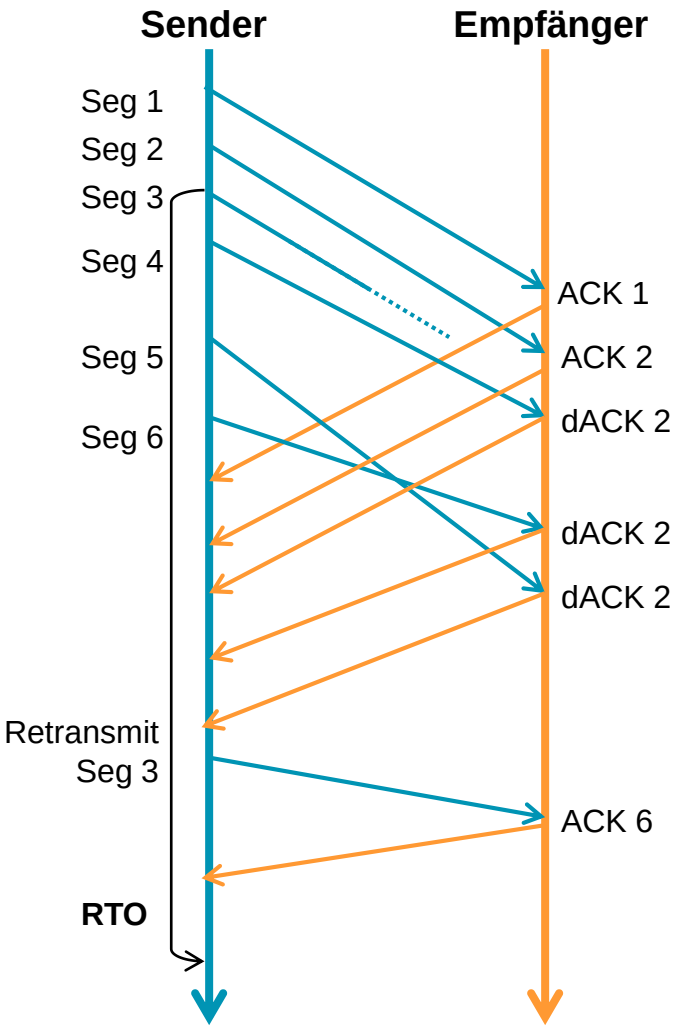
dargestelltes Problem:

- **Byteweise Verarbeitung auf Empfängerseite: Der Sender wird animiert, kleine Segmente zu senden.**

Lösungen:

- **Senderseitig:**
  (1) Versenden von kleinen Datenmengen vermeiden.
  (2) Nagle's Algorithm.
- **Empfängerseitig:**
  Fensteraktualisierungen nur bei größerem Betrag (z.B. wenn 30% vom Empfangspuffer oder 2 MSS frei)

Fast Retransmit (Grundprinzip)



dargestelltes Problem:

- **Grobe Einstellung des RTO führt zu Wartezeiten bei Paketverlust**

Lösung:

- **Bei dreifachem duplicate ACK Retransmission des fehlenden Pakets ohne auf Ablaufen des Retransmissiontimers zu warten**

Neues Problem:

- **Verlorenes Paket ist Zeichen für Überlastung des Netzes. Daher nach Fast Retransmit Reaktion auf Netzüberlast notwendig.**

–> Themenstellung Überlastkontrolle

Anmerkung:

Fast Recovery: Algorithmus, um nach Fast Retransmit Datenfluss zu erhalten
SACK (Selective Acknowledgement, RFC2018): Bestätigung der tatsächlich erhaltenen Segemente

TCP-Acknowledgements (gemäß RFC 1122, 10/89)

Ereignis	Reaktion TCP-Empfänger
Ankunft eines direkt nachfolgenden Segments, keine Lücke, alle vorhergehenden Segmente sind bereits bestätigt.	**delayed Acknowledgement:** Warte bis zu 200 ms, ob neues Segment kommt, wenn bis dahin keines kommt, muss ein ACK gesendet werden.
Ankunft von direkt nachfolgenden Segmenten, keine Lücke	**Senden eines kummulativen ACKs:** Bestätigen von mehreren Segmenten mit einem Acknowledgement
Ankunft eines Segments, das ganz oder teilweise eine Lücke füllt (so dass ein Teil des Bytestroms vervollständigt wird).	**Immediate Acknowledgement:** Sofortiges Senden eines ACKs.
Ankunft eines out-of-order Segments größer als NextByteExpected (es entsteht Lücke).	**Senden eines duplicate ACKs:** wiederholtes Senden des letzten ACKs (=Beginn der Lücke)