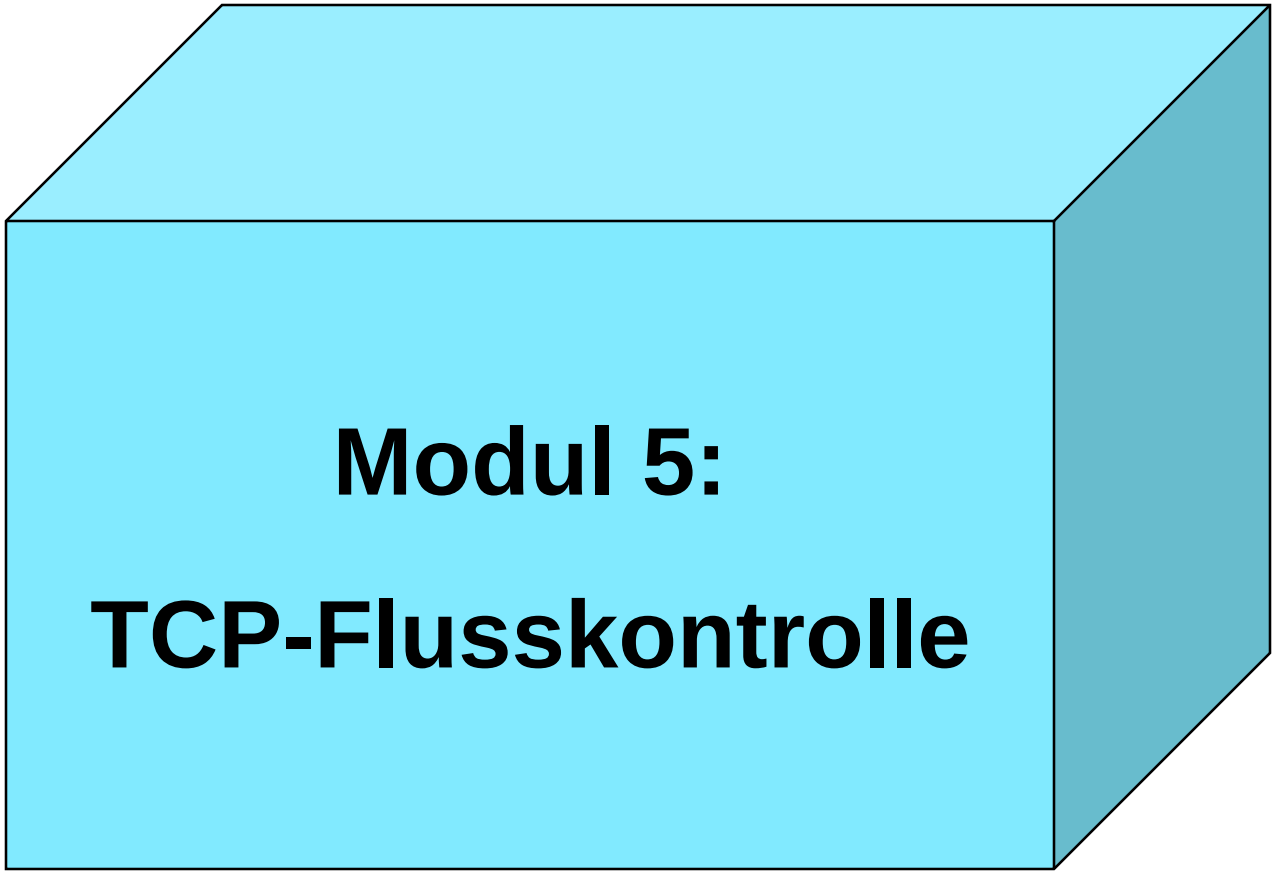
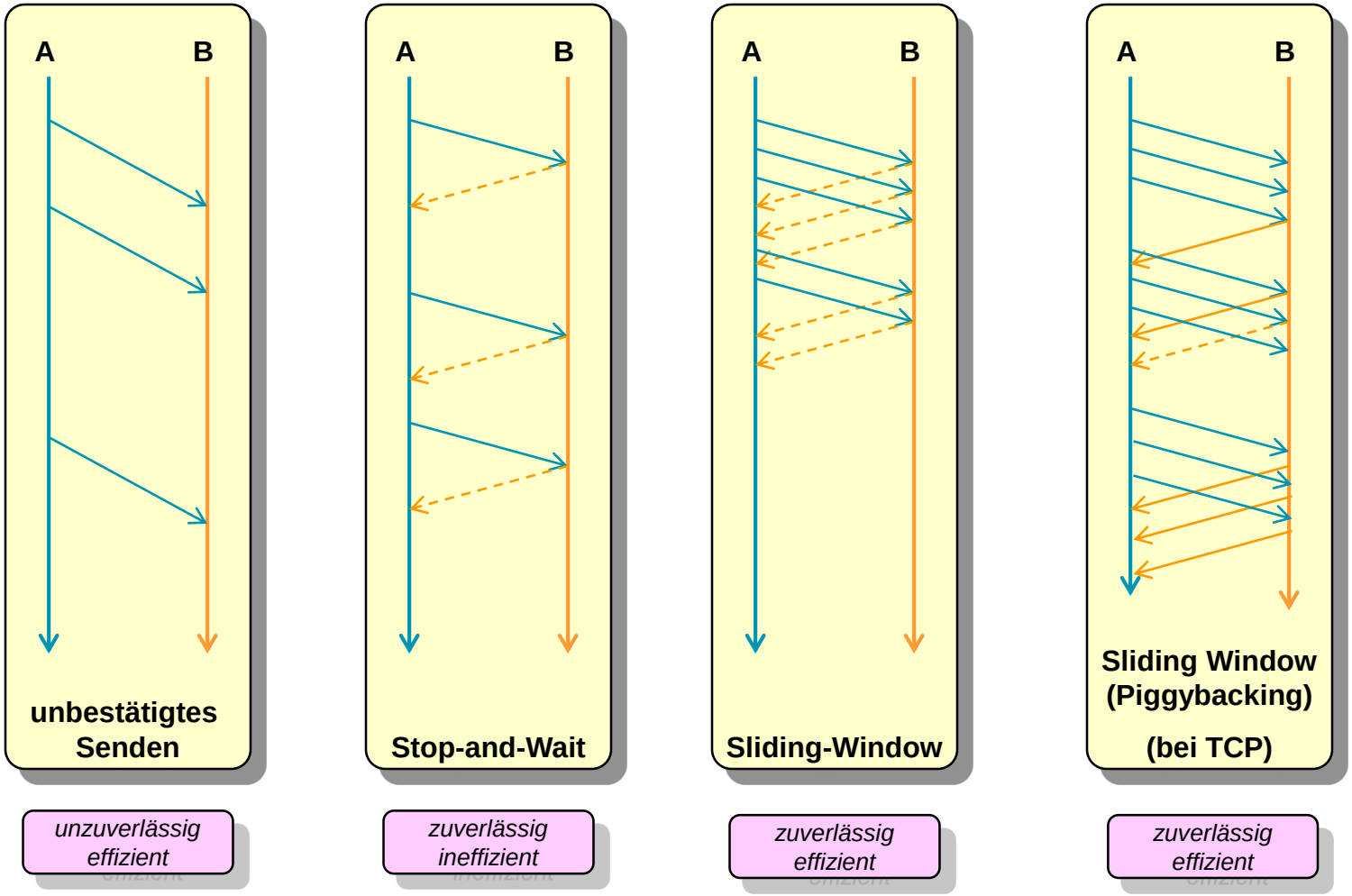
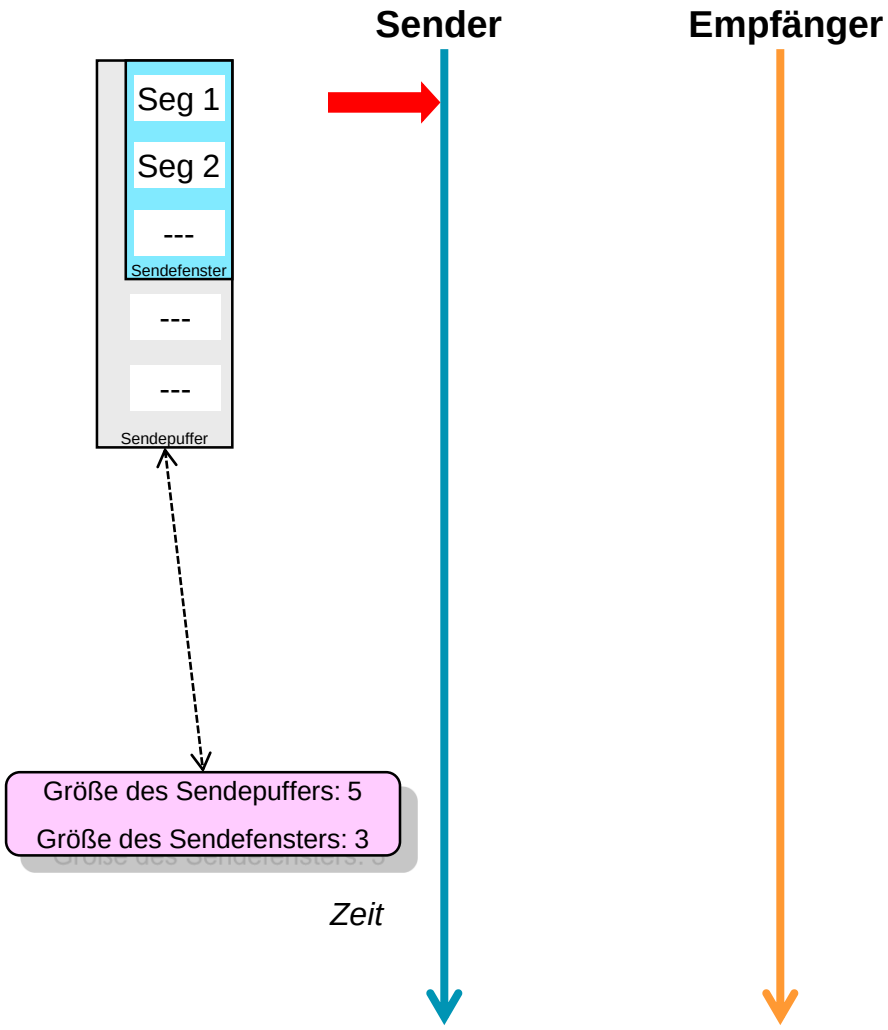


Prinzip des Sliding-Window: Zuverlässigkeit + Effizienz



Sliding-Window - detaillierter

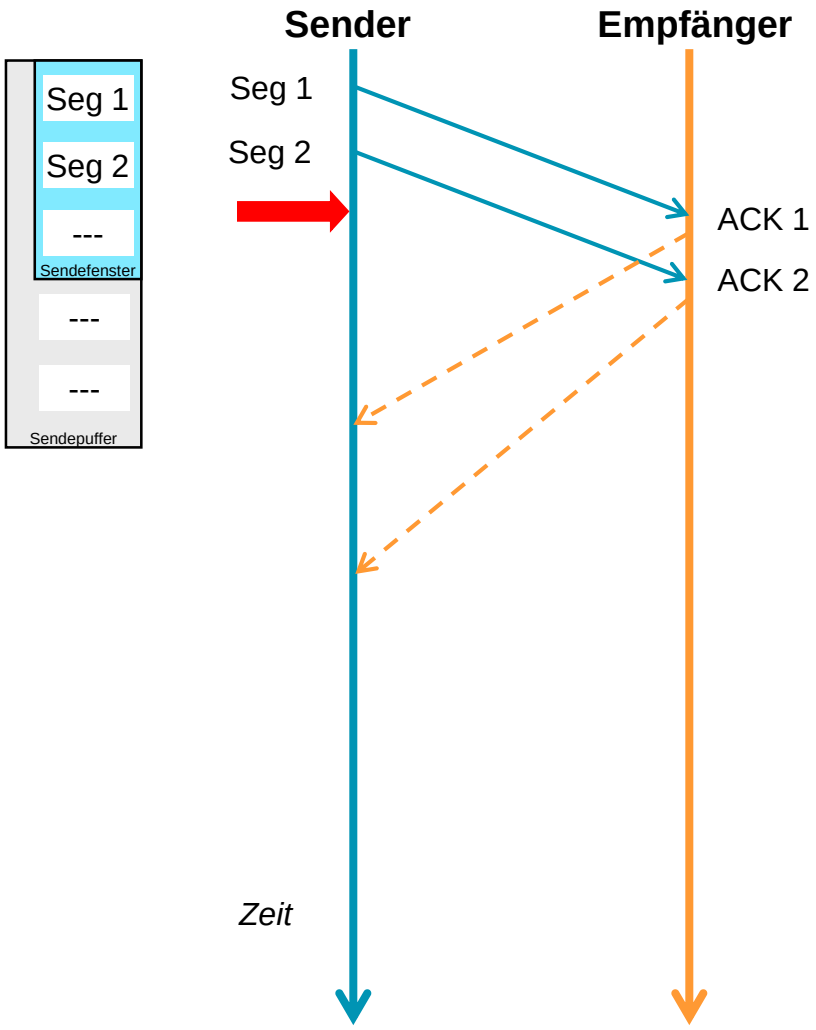


- **Anwendung produziert 2 Segmente**
- **Diese Segmente liegen im Sendepuffer und im Sendefenster**

Was ist der Unterschied zwischen Sendepuffer und Sendefenster?

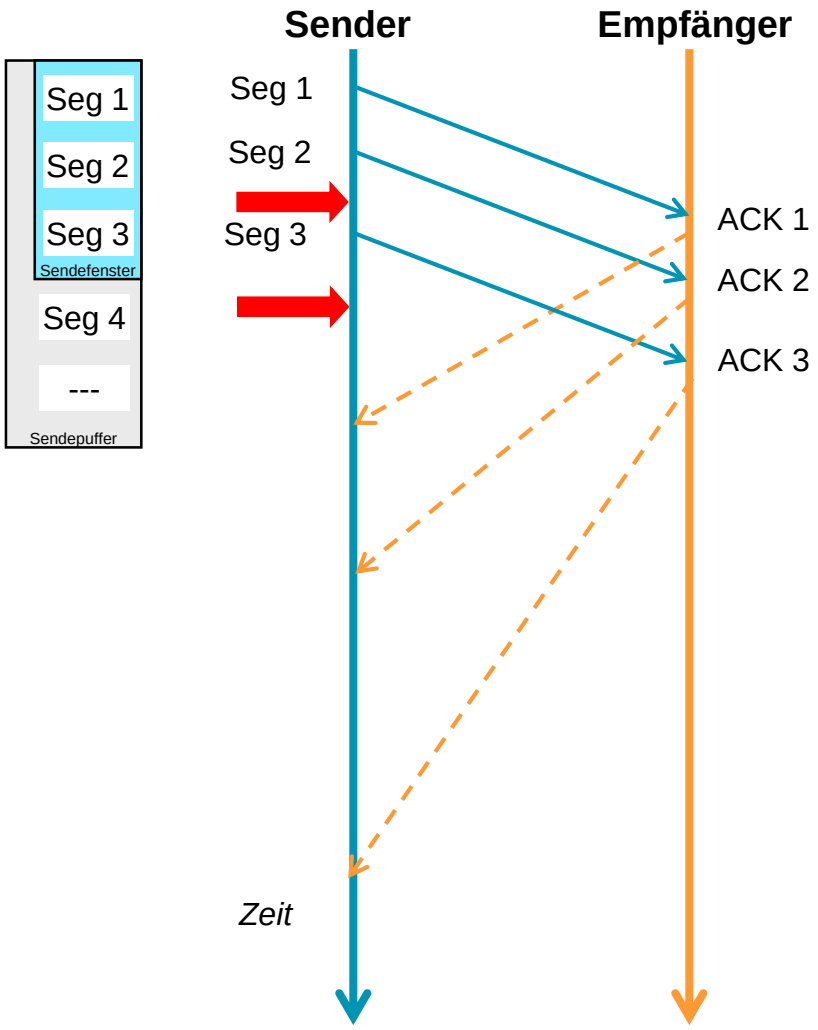
Was ist die Aufgabe von Sendepuffer und Sendefenster?

Sliding-Window - detaillierter



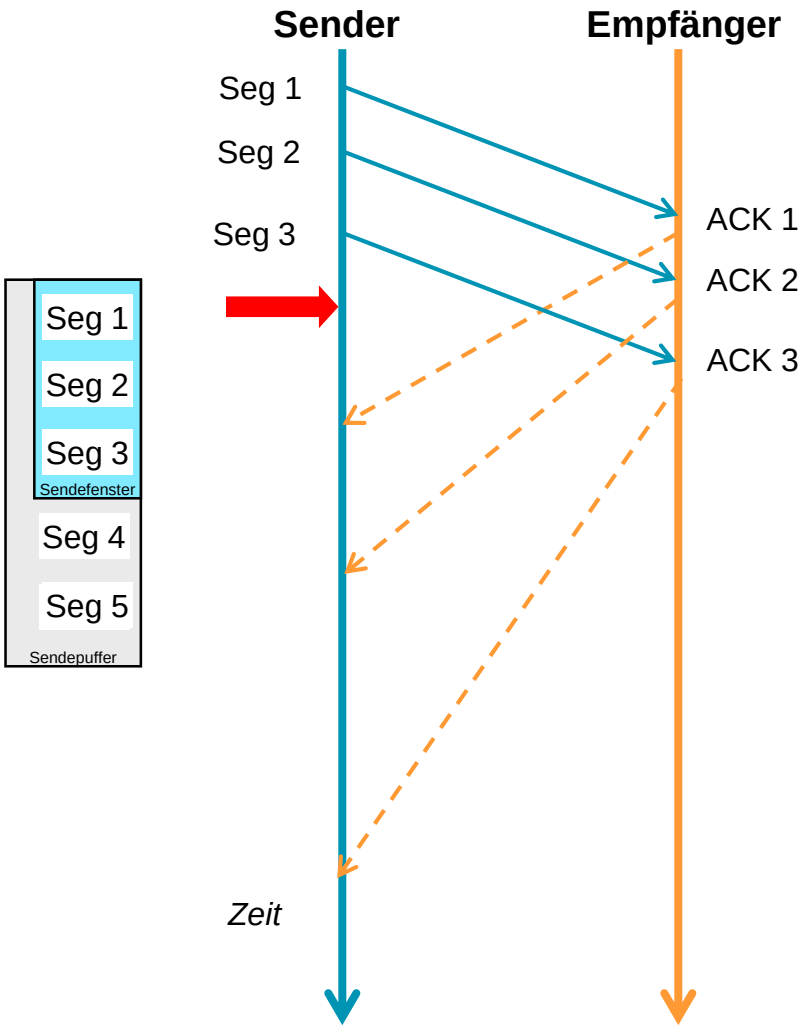
- **Was passiert jetzt??**
- **Beide Segmente können gesendet werden**
- **Auswirkungen auf den Sendepuffer und das Sendefenster?**

Sliding-Window - detaillierter



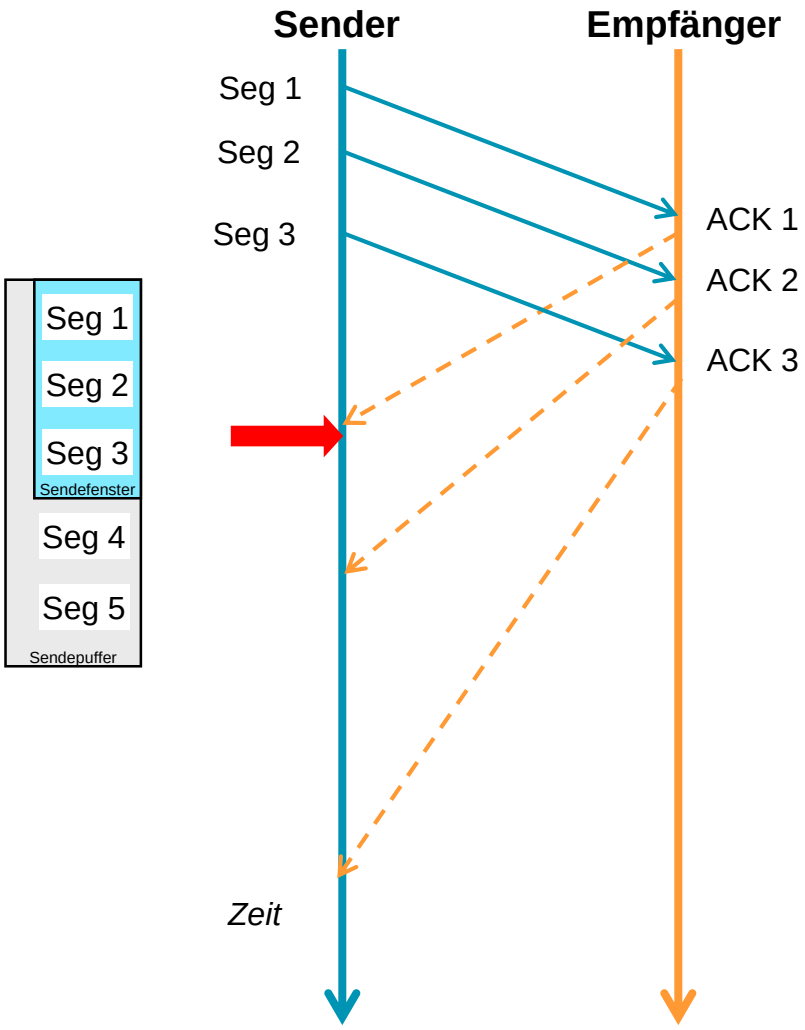
- **Anwendung produziert zwei weitere Segmente**
- **Was macht der Sender?**

Sliding-Window - detaillierter



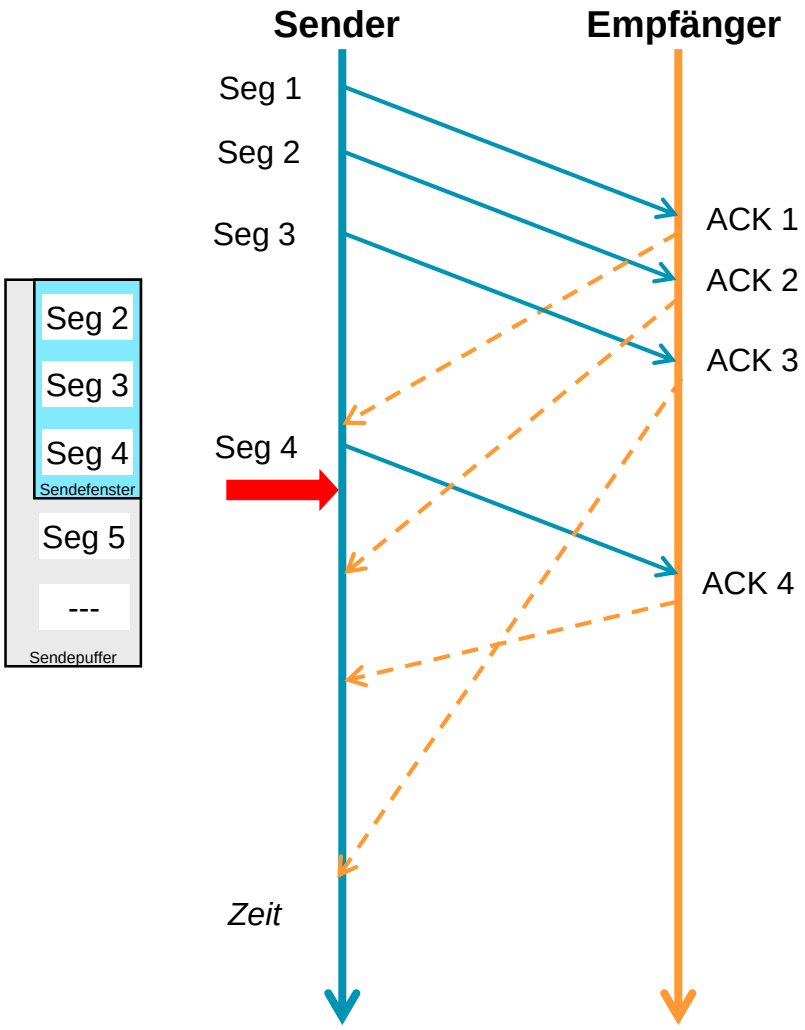
- **Anwendung produziert ein weiteres Segment**
- **Was passiert?**
- **Was wäre, wenn sie zwei weitere Segmente produziert?**

Sliding-Window - detaillierter



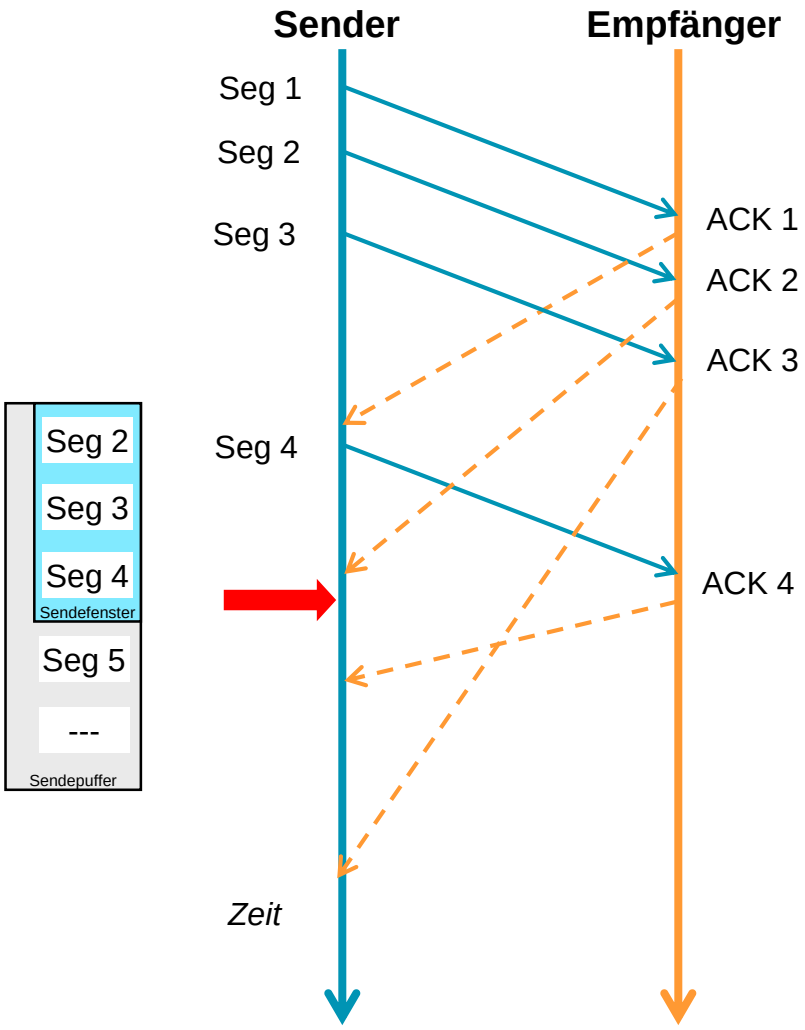
- **ACK 1 kommt nun beim Sender an**
- **Was passiert?**

Sliding-Window - detaillierter



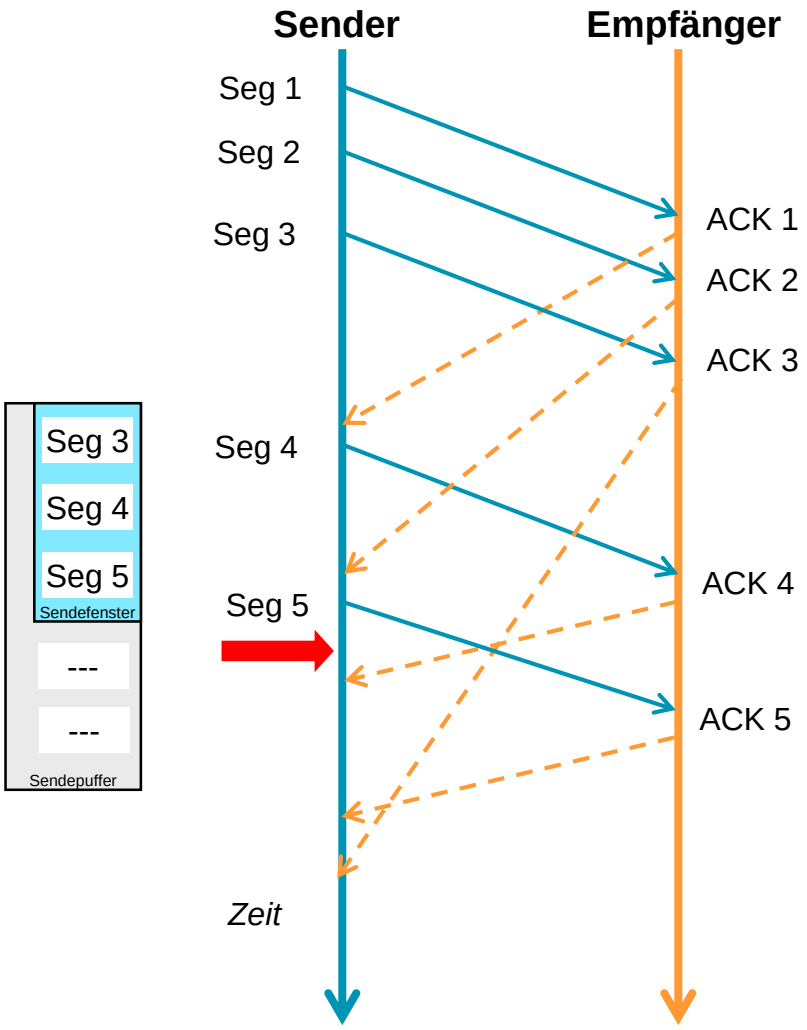
- **Schieben des Fensters**
- **Versenden von Segment 4**

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:

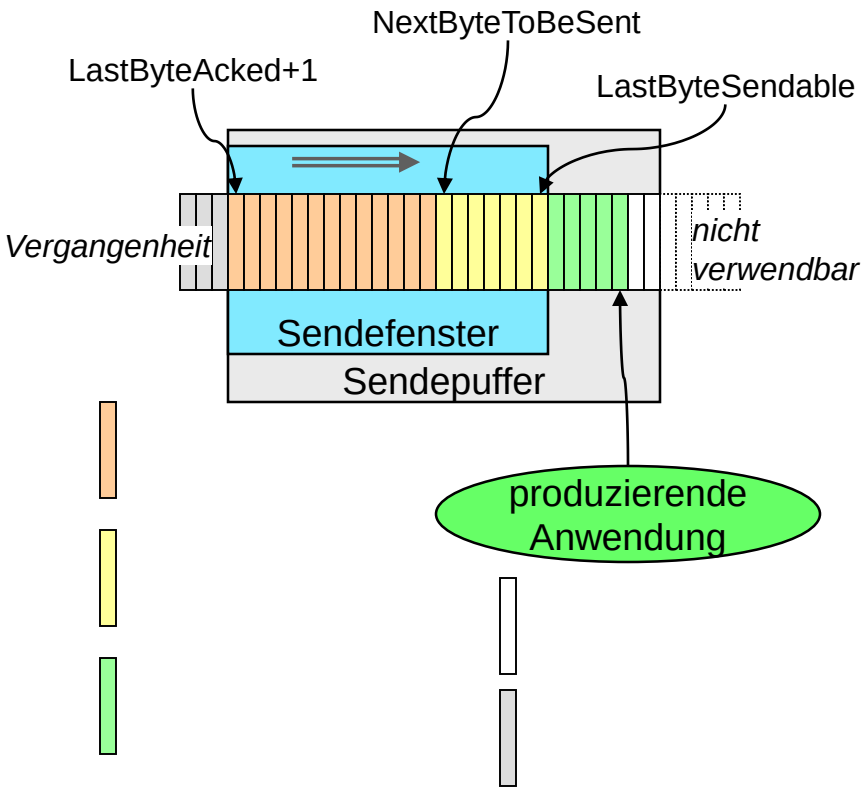
- **Wie groß soll das Sendefenster sein?**
- **Wer legt das Sendefenster fest?**

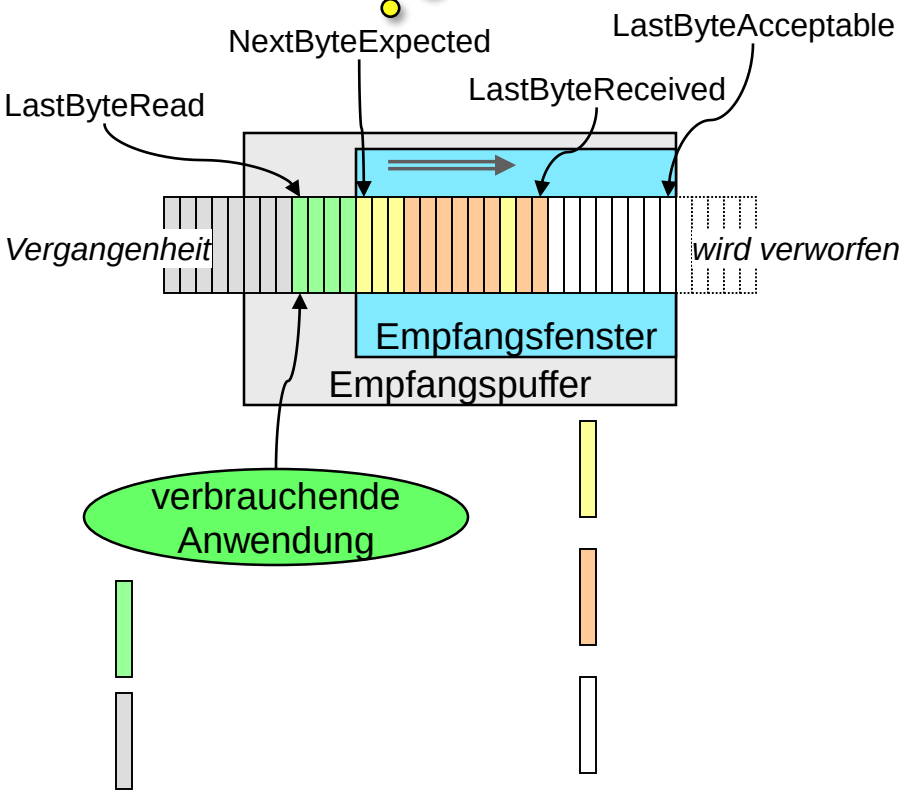
TCP-Sliding-Window – Sende- und Empfangspuffer

Jetzt alles auf Byte-Niveau!

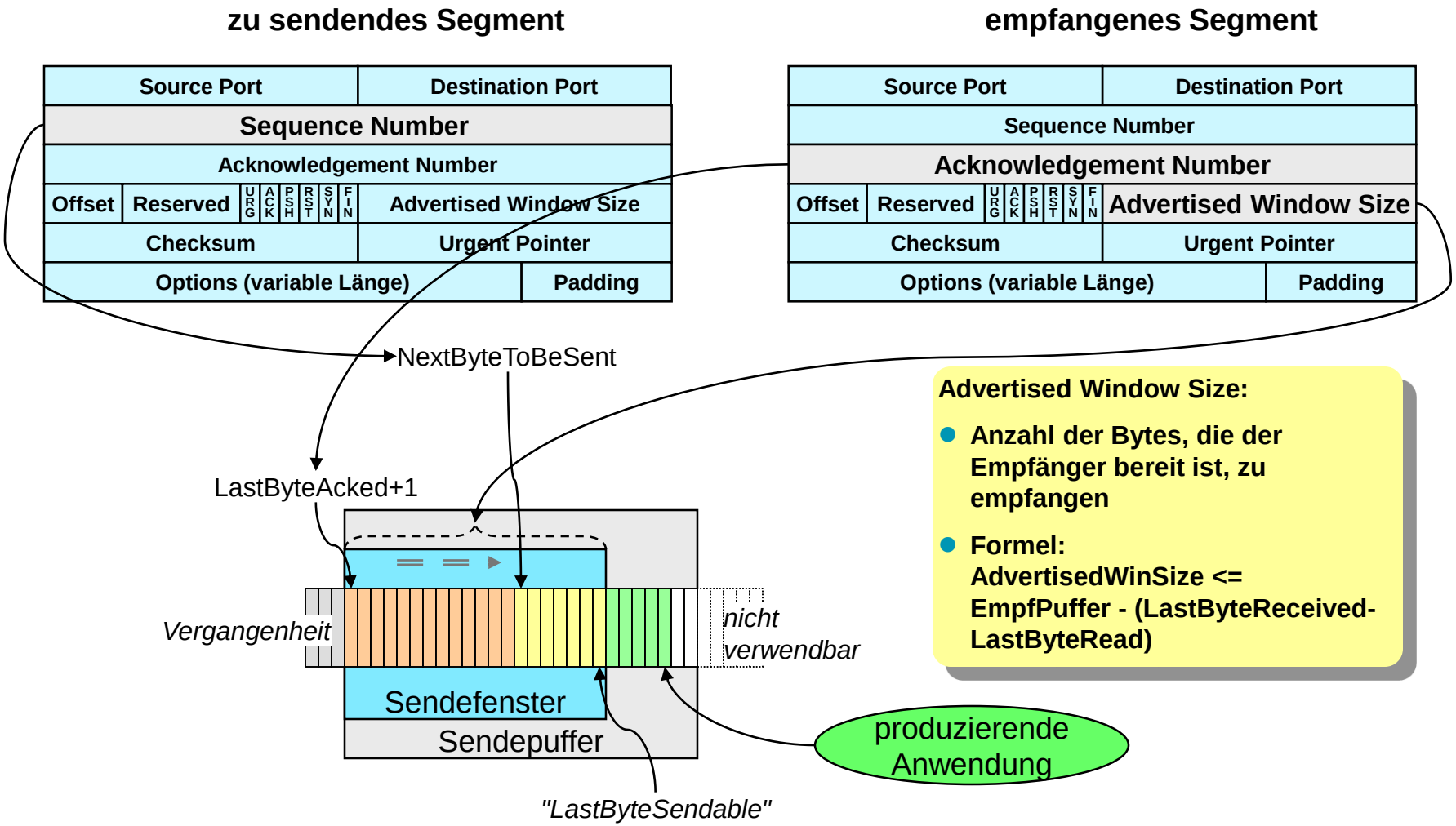
Spezielle Variante des Sliding-Window-Grundalgorithmus

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

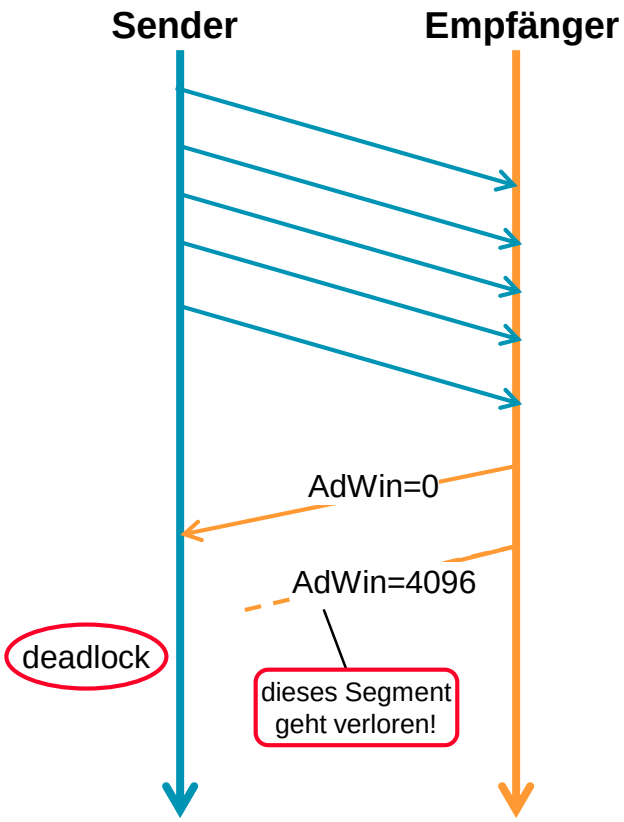




TCP-Flusskontrolle – Segmente auf Senderseite



TCP-Flusskontrolle – "Zero advertised window size"



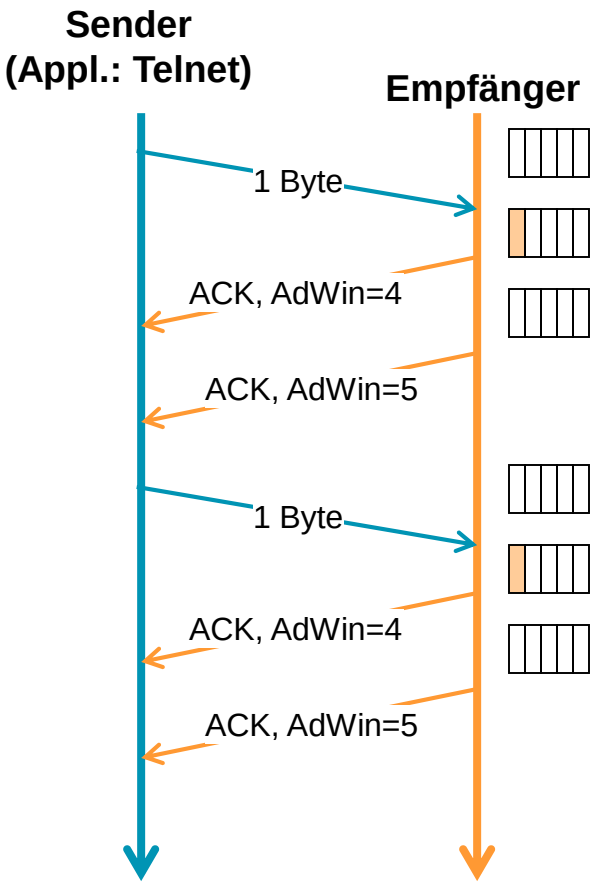
Dargestelltes Problem:

- **Wie kommt es zum Deadlock?**

Lösung des Problems:

- ?
-

TCP-Flusskontrolle – "Small Packet Problem"



dargestelltes Problem:

?

Lösungen:

- **<u>Empfängerseitig</u>: delayed Acknowledgement:**

  ?

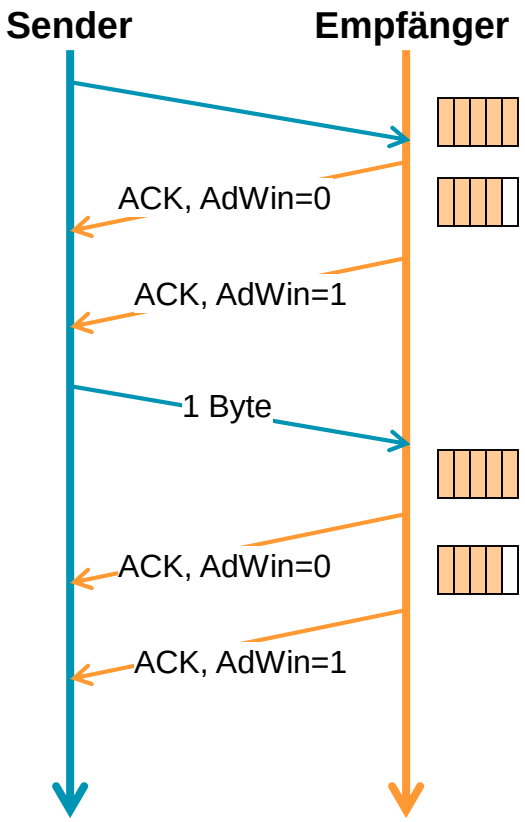
- **<u>Senderseitig</u>: Nagle's Algorithm (RFC 896, 1984):**

  ?

  ?

Alg. über Sockets abschaltbar.

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



dargestelltes Problem:

?

Lösungen:

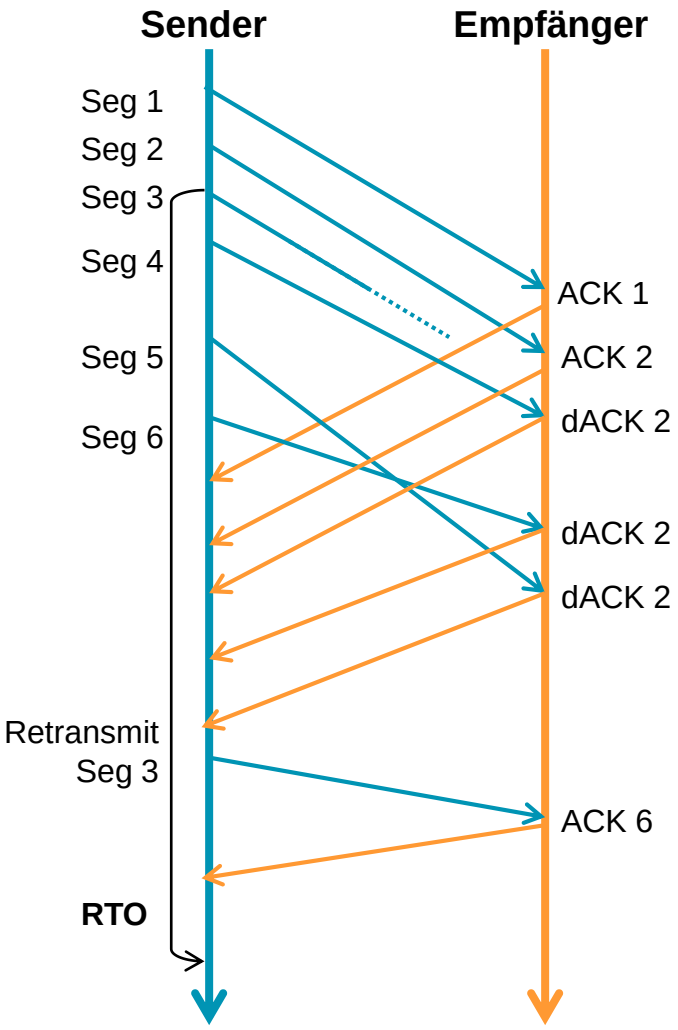
- **Senderseitig:**

?

- **Empfängerseitig:**

?

Fast Retransmit (Grundprinzip)



dargestelltes Problem:

?

Lösung:

- ?

Neues Problem:

- ?

–> 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)