Gruppen Di-T14 / Mi-T25

Tutorübung zu
Grundlagen: Rechnernetze und Verteilte Systeme
(SS 16)

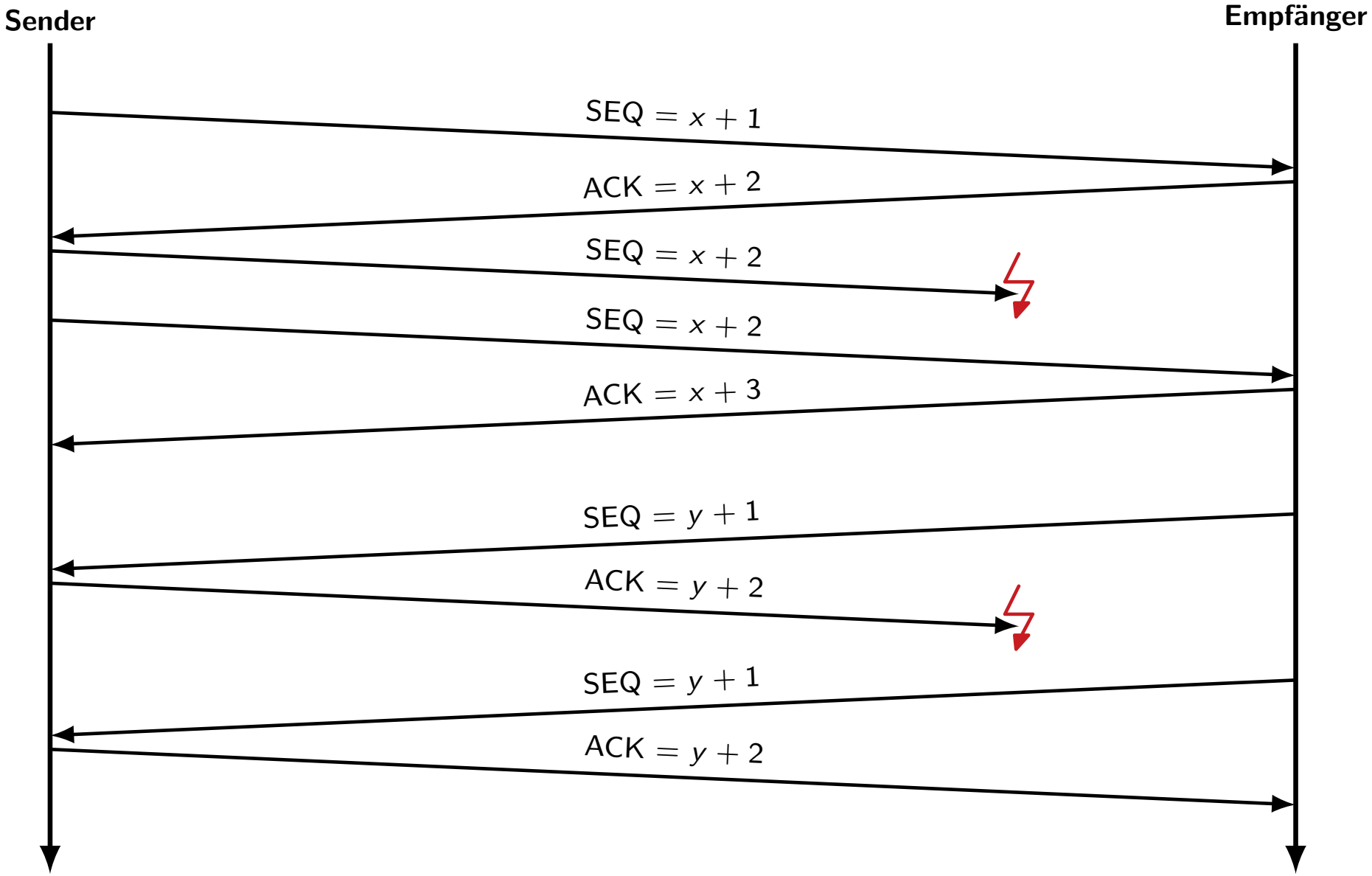
Michael Schwarz

Institut für Informatik
Technische Universität München

21.06 / 22.06.2016

Verbindungsorientierte Übertragung

Beispiel: Übertragungsphase



Sliding-Window-Verfahren

Bislang:

- Im vorherigen Beispiel hat der Sender stets nur ein Segment gesendet und dann auf eine Bestätigung gewartet
- Dieses Verfahren ist ineffizient, da abhängig von der Umlaufverzögerung (Round Trip Time, RTT) zwischen Sender und Empfänger viel Bandbreite ungenutzt bleibt („Stop and Wait“-Verfahren)

Idee: Teile dem Sender mit, wie viele Segmente **nach** dem letzten bestätigten Segment auf einmal übertragen werden dürfen, ohne dass der Sender auf eine Bestätigung warten muss.

Vorteile:

- Zeit zwischen dem Absenden eines Segments und dem Eintreffen einer Bestätigung kann effizienter genutzt werden
- Durch die Aushandlung dieser Fenstergrößen kann der Empfänger die Datenrate steuern → Flusskontrolle
- Durch algorithmische Anpassung der Fenstergröße kann die Datenrate an die verfügbare Datenrate auf dem Übertragungspfad zwischen Sender und Empfänger angepasst werden → Staukontrolle

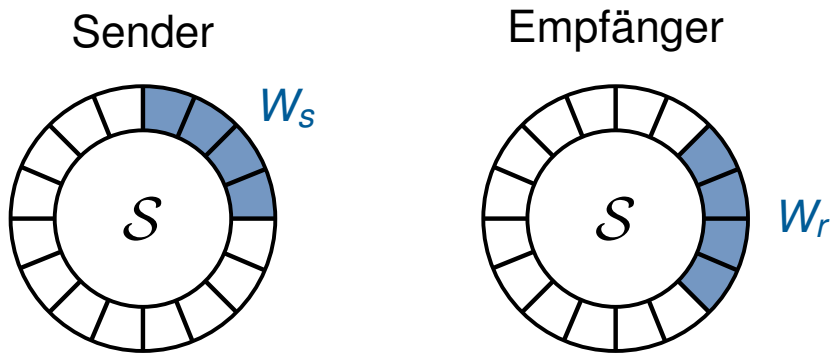
Probleme:

- Sender und Empfänger müssen mehr Zustand halten
  (Was wurde bereits empfangem? Was wird als nächstes erwartet?)
- Der Sequenznummernraum ist endlich → Wie werden Missverständnisse verhindert?

Zur Notation:

- Sender und Empfänger haben denselben Sequenznummernraum $\mathcal{S} = \{0, 1, 2, \ldots, N-1\}$.

  Beispiel: $N = 16$:



- Sendefenster (Send Window) $W_s \subset \mathcal{S}$, $|W_s| = w_s$:
  Es dürfen w_s Segmente nach dem letzten bestätigten Segment auf einmal gesendet werden.
- Empfangsfenster (Receive Window) $W_r \subset \mathcal{S}$, $|W_r| = w_r$:
  Sequenznummern der Segmente, die als nächstes akzeptiert werden.
- Sende- und Empfangsfenster „verschieben“ und überlappen sich während des Datenaustauschs.

Vereinbarungen:

- Eine Bestätigung ACK = $m + 1$ bestätigt alle Segmente mit SEQ $\leq m$. Dies wird als kumulative Bestätigung bezeichnet.
- Gewöhnlich löst jedes erfolgreich empfangene Segment das Senden einer Bestätigung aus, wobei stets das nächste erwartete Segment bestätigt wird. Dies wird als Forward Acknowledgement bezeichnet.

Wichtig:

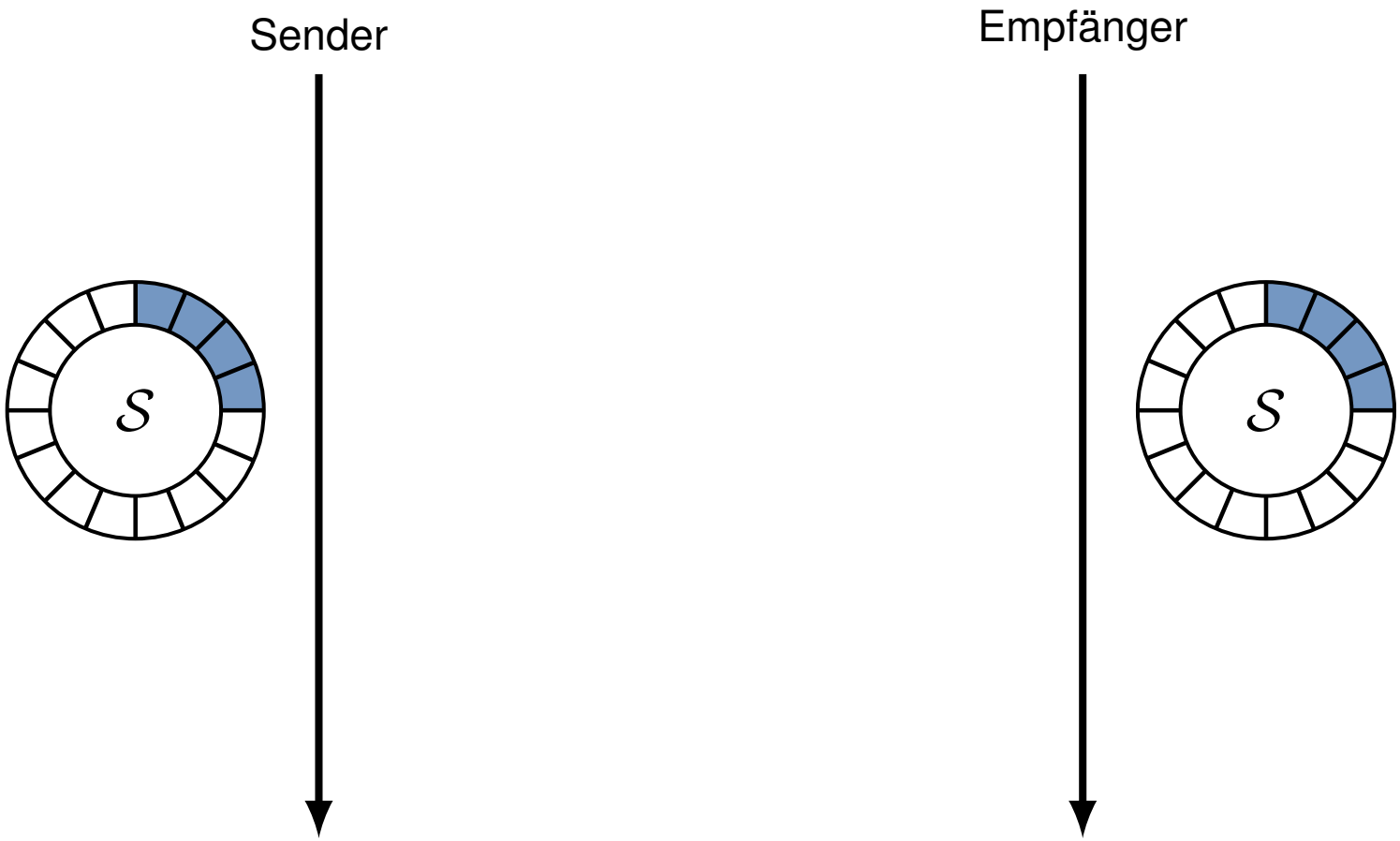
- In den folgenden Grafiken sind die meisten Bestätigungen zwecks Übersichtlichkeit nur angedeutet (graue Pfeile).
- Die Auswirkungen auf Sende- und Empfangsfenster beziehen sich nur auf den Erhalt der schwarz eingezeichneten Bestätigungen.
- Dies ist äquivalent zur Annahme, dass die angedeuteten Bestätigungen verloren gehen.

Sliding-Window-Verfahren

Sendefenster W_s bzw. Empfangsfenster W_r

gesendet aber noch nicht bestätigt

gesendet und bestätigt/empfangen

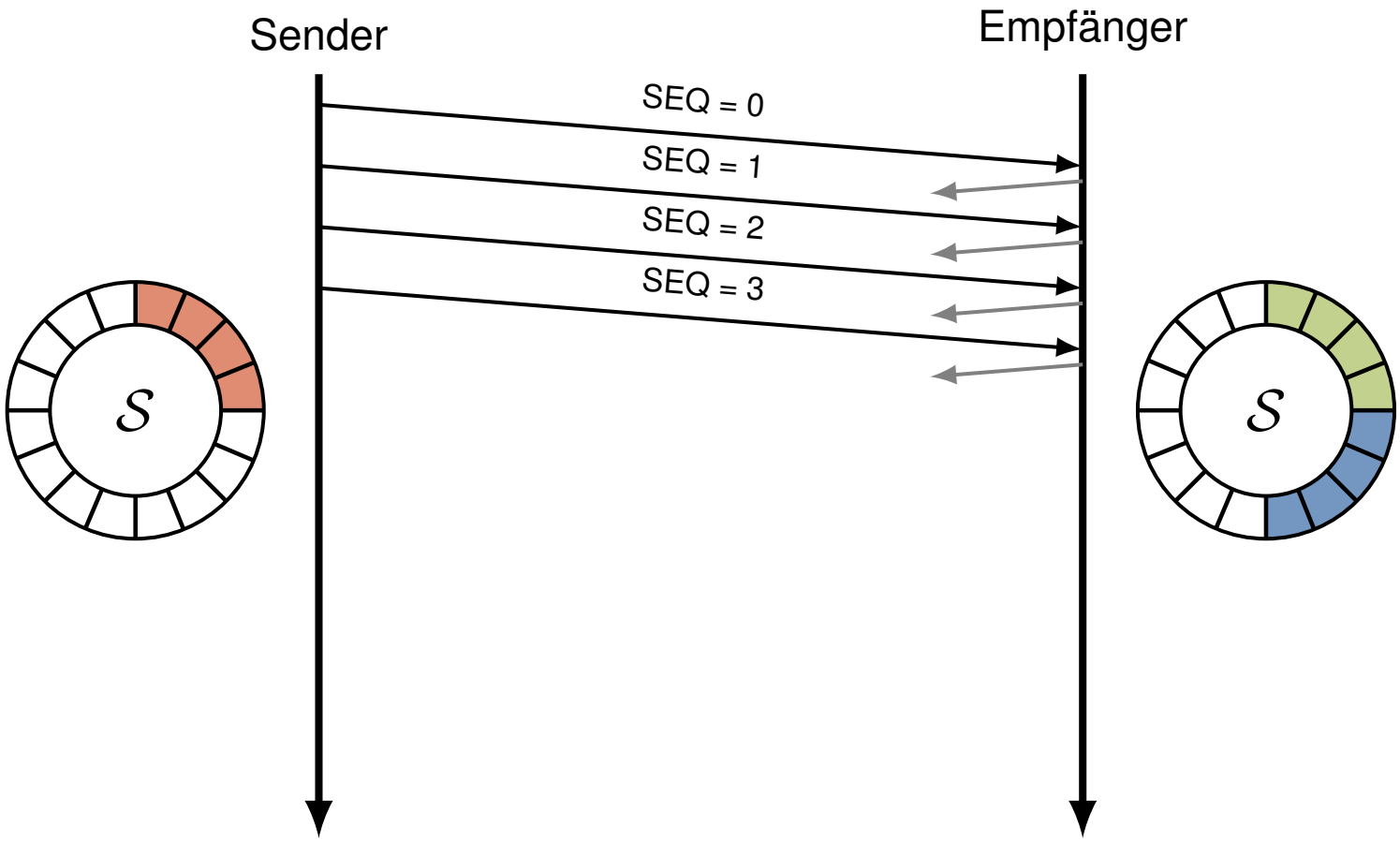


Sliding-Window-Verfahren

Sendefenster W_s bzw. Empfangsfenster W_r

gesendet aber noch nicht bestätigt

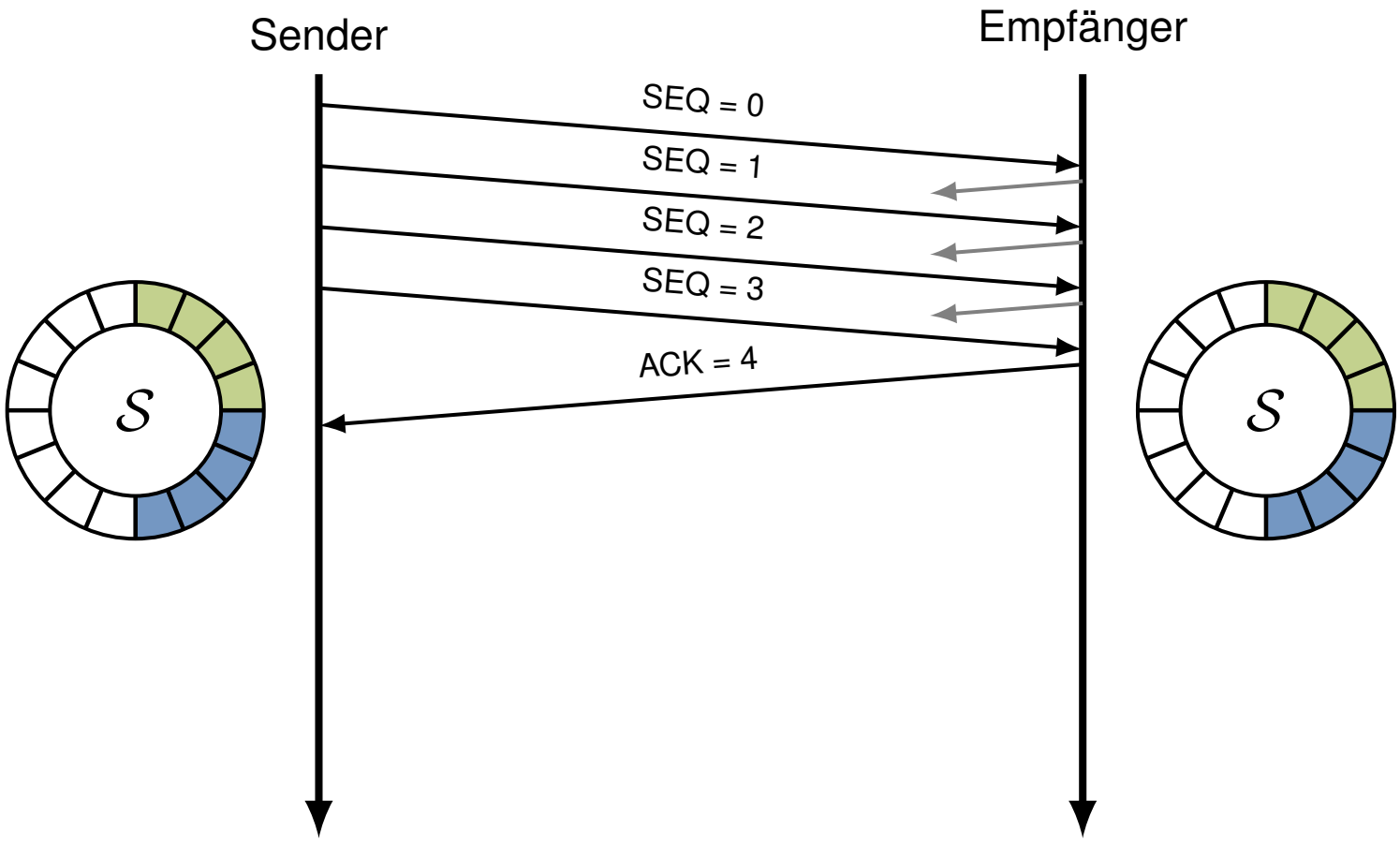
gesendet und bestätigt/empfangen



Sendefenster W_s bzw. Empfangsfenster W_r

gesendet aber noch nicht bestätigt

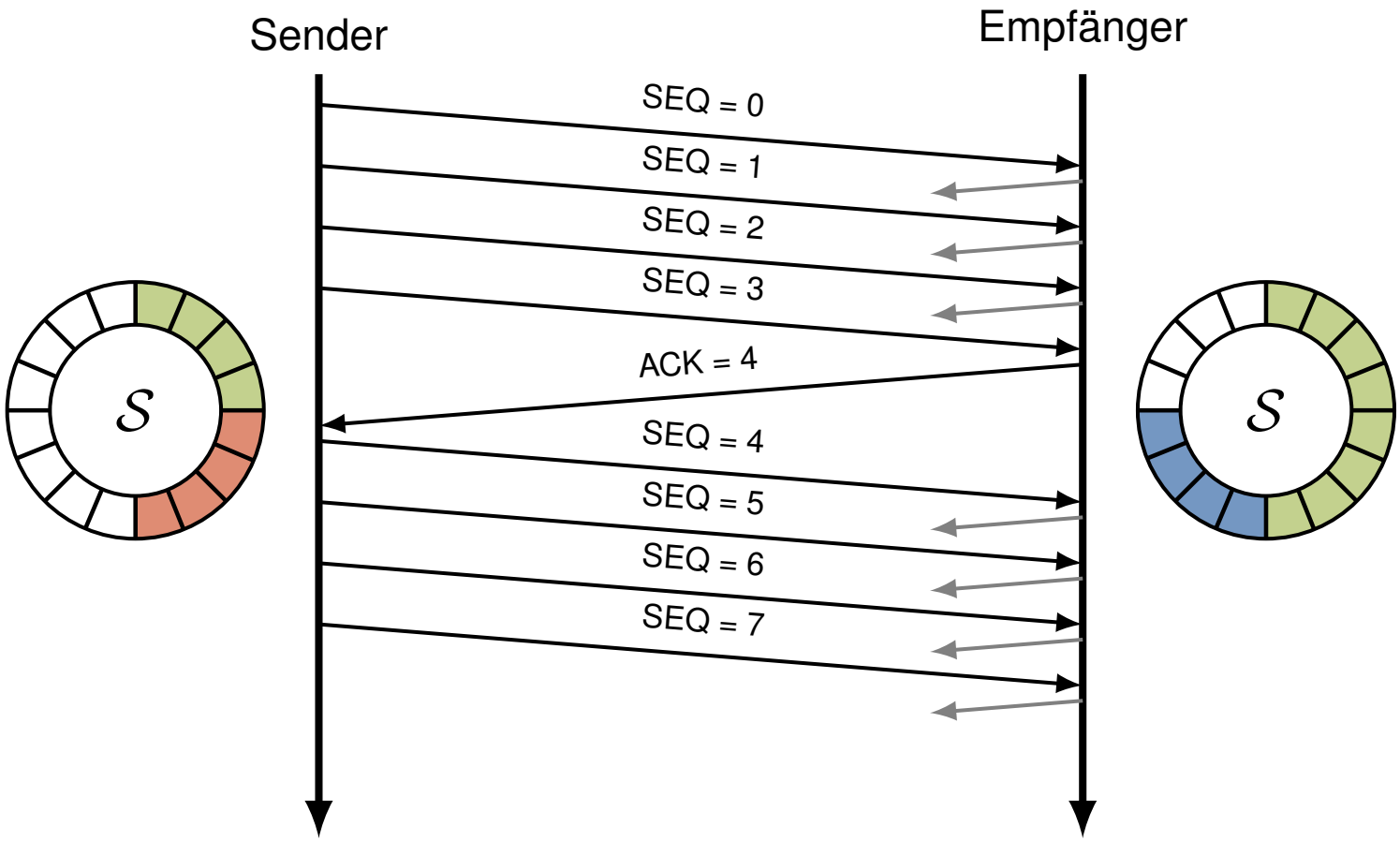
gesendet und bestätigt/empfangen



Sendefenster W_s bzw. Empfangsfenster W_r

gesendet aber noch nicht bestätigt

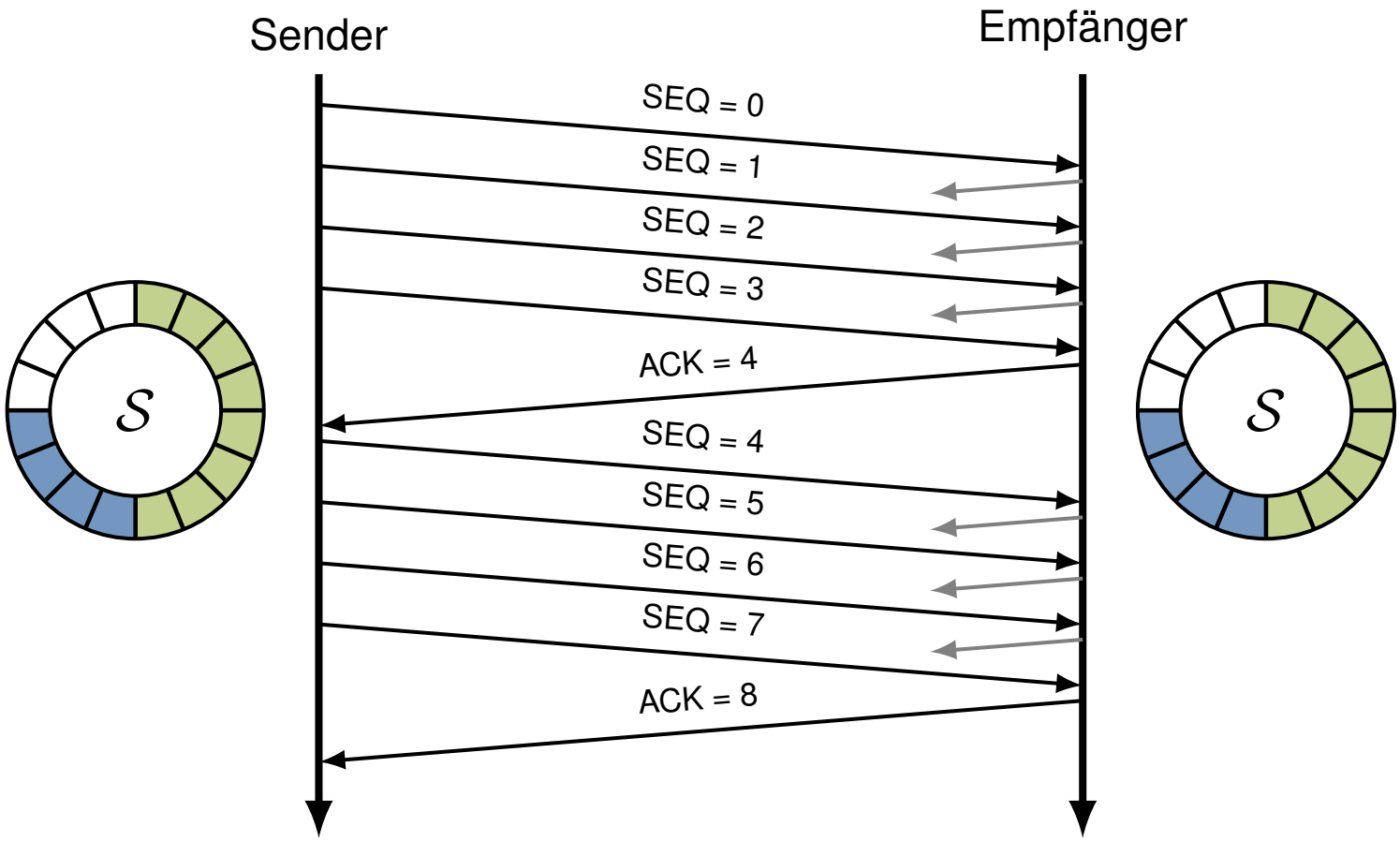
gesendet und bestätigt/empfangen



Sendefenster W_s bzw. Empfangsfenster W_r

gesendet aber noch nicht bestätigt

gesendet und bestätigt/empfangen



Sliding-Window-Verfahren

Neues Problem: Wie wird jetzt mit Segmentverlusten umgegangen?

Zwei Möglichkeiten:

1. Go-Back-N
   - Akzeptiere stets nur die nächste erwartete Sequenznummer
   - Alle anderen Segmente werden verworfen
2. Selective-Repeat
   - Akzeptiere alle Sequenznummern, die in das aktuelle Empfangsfenster fallen
   - Diese müssen gepuffert werden, bis fehlende Segmente erneut übertragen wurden

Wichtig:

- In beiden Fällen muss der Sequenznummernraum so gewählt werden, dass wiederholte Segmente eindeutig von neuen Segmenten unterschieden werden können.
- Andernfalls würde es zu Verwechslungen kommen
  $\rightarrow$ Auslieferung von Duplikaten an höhere Schichten, keine korrekte Reihenfolge.

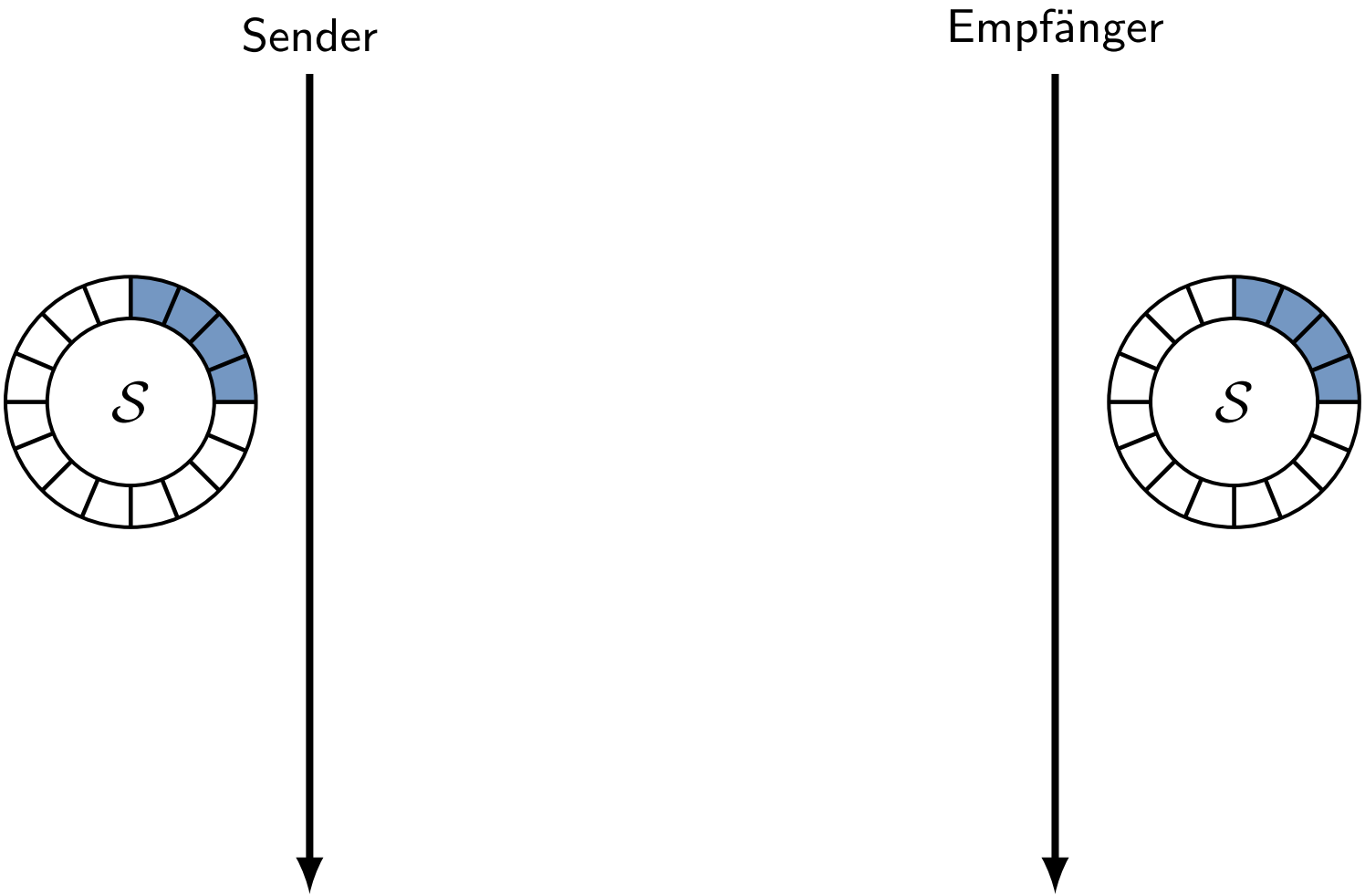
Frage: (siehe Übung)

Wie groß darf das Sendefenster W_s in Abhängigkeit des Sequenznummernraums $\mathcal{S}$ höchstens gewählt werden, so dass die Verfahren funktionieren?

Sliding-Window-Verfahren

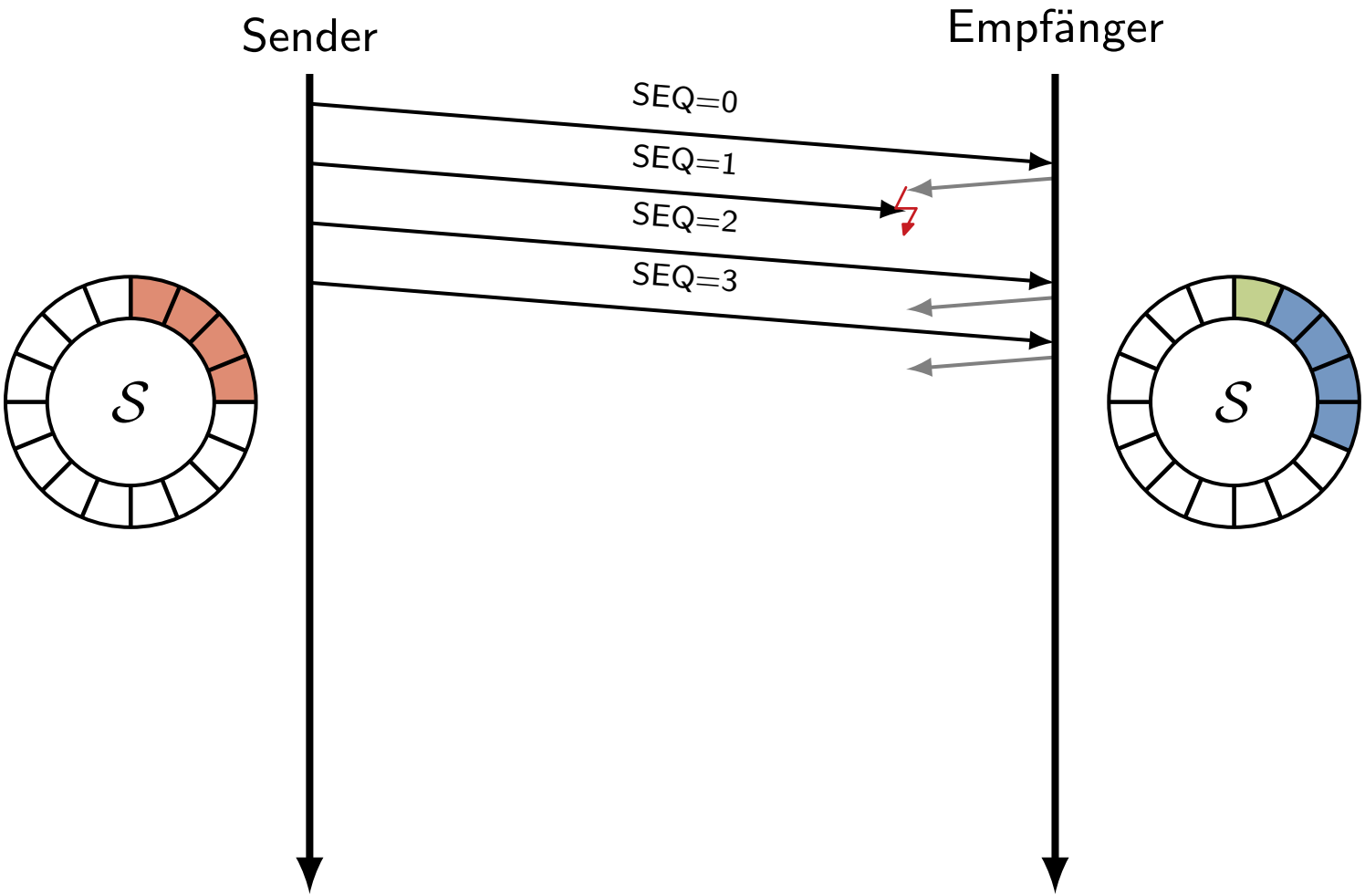
Go-Back-N:

$N = 16$, $w_s = 4$, $w_r = 4$



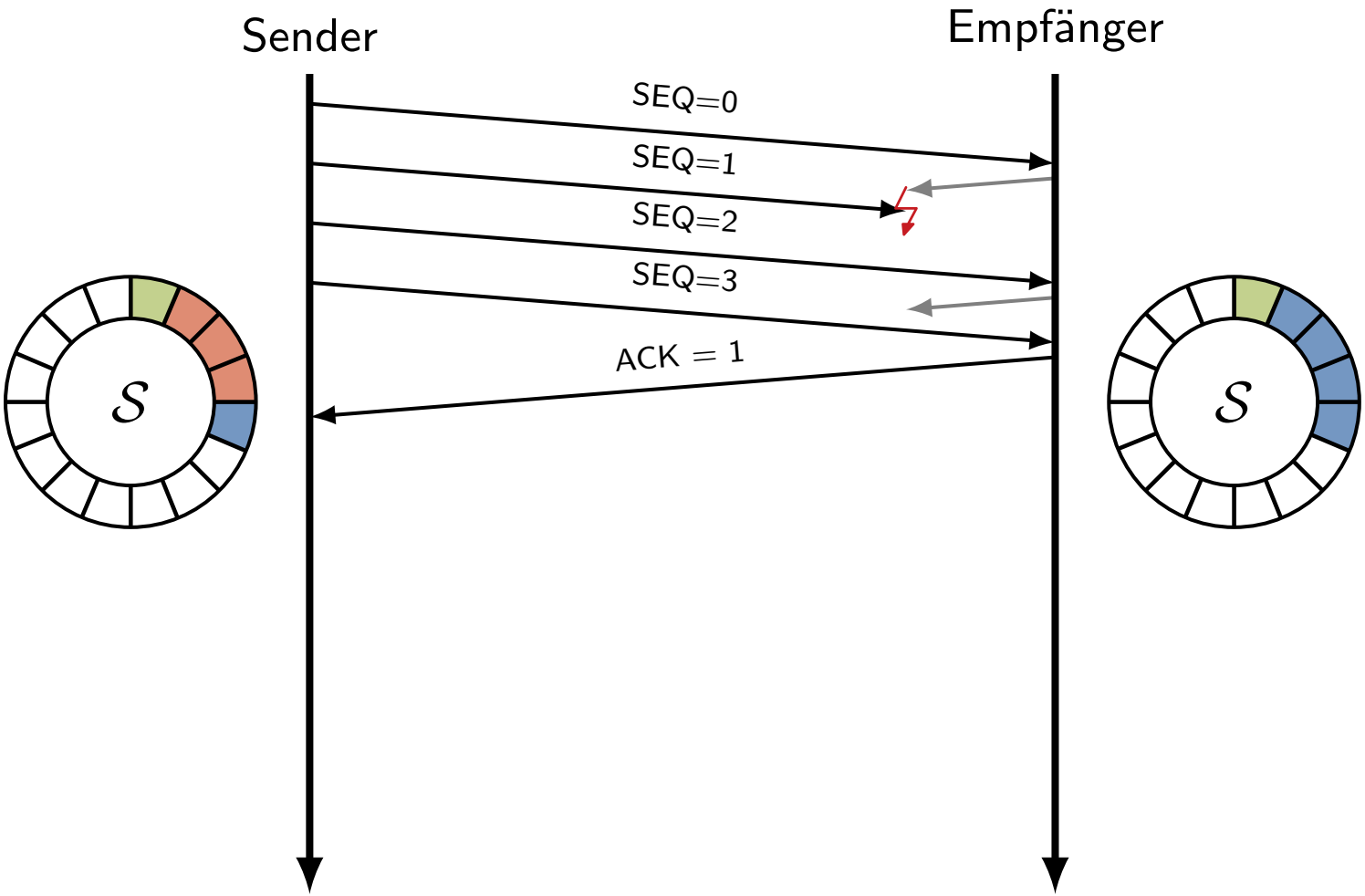
Go-Back-N:

$N = 16$, $w_s = 4$, $w_r = 4$



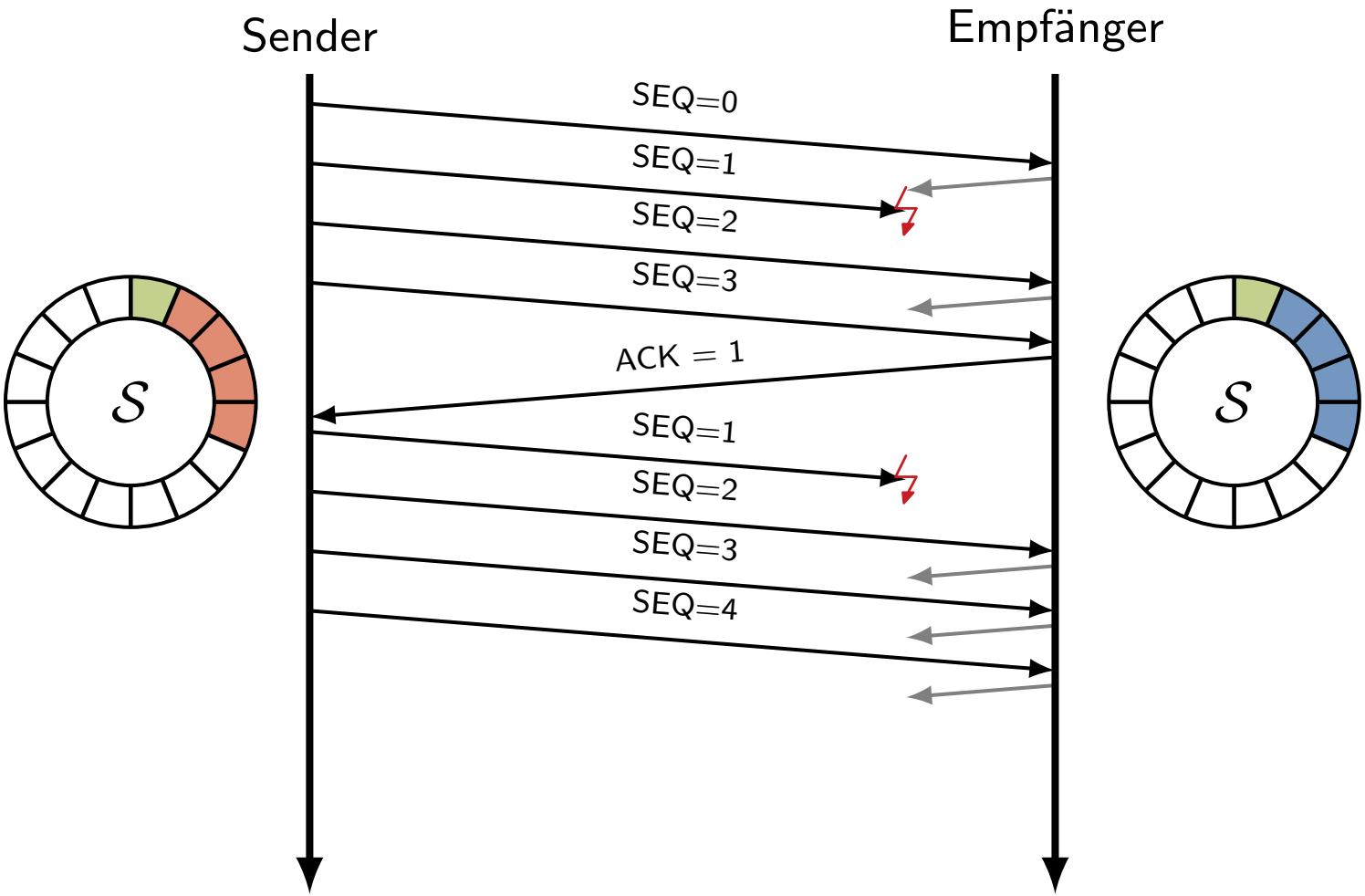
Go-Back-N:

$N = 16$, $w_s = 4$, $w_r = 4$



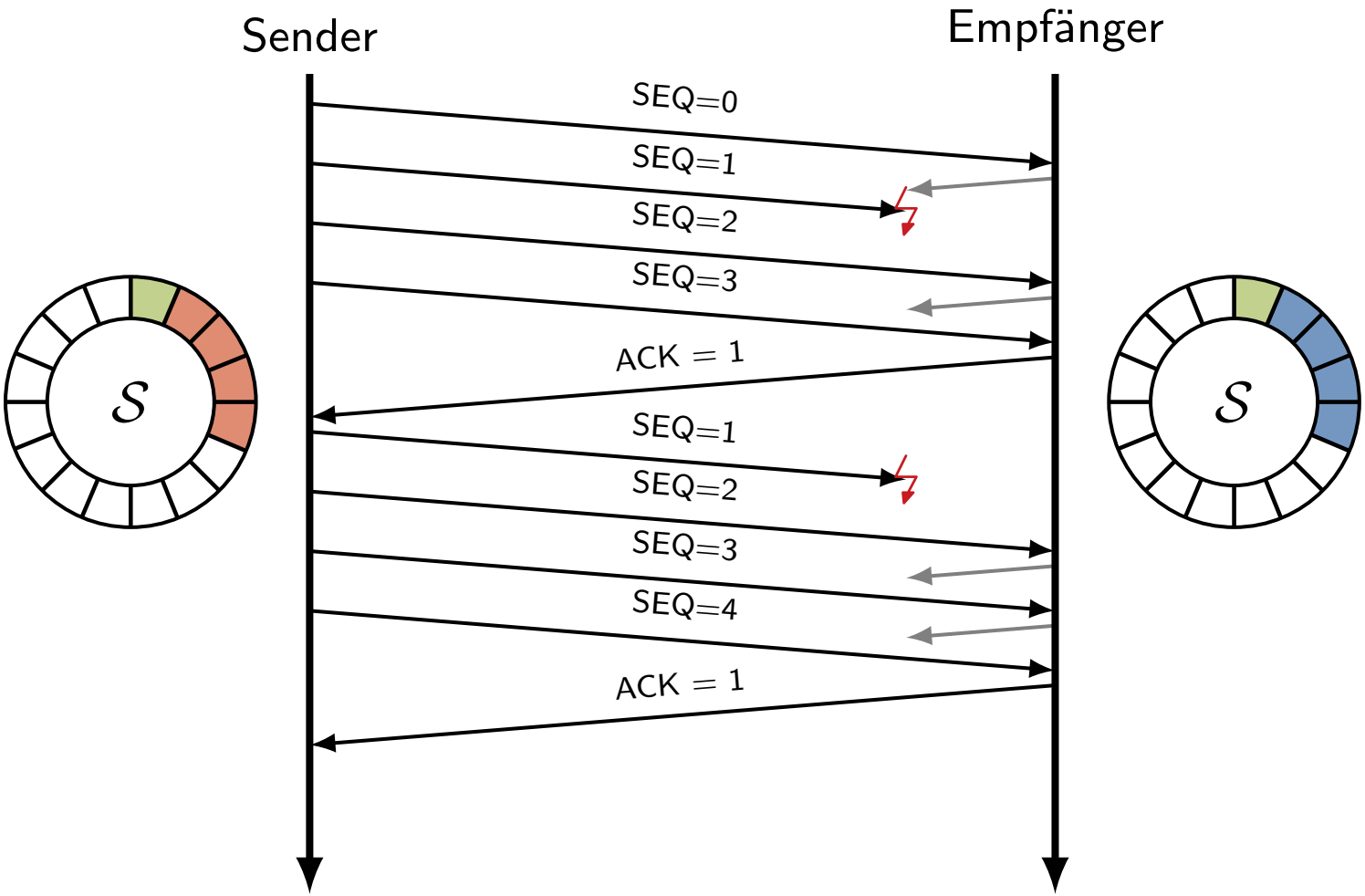
Go-Back-N:

$N = 16$, $w_s = 4$, $w_r = 4$



Go-Back-N:

$N = 16$, $w_s = 4$, $w_r = 4$



Anmerkungen zu Go-Back-N

Da der Empfänger stets nur das nächste erwartete Segment akzeptiert, reicht ein Empfangsfenster der Größe $w_r = 1$ prinzipiell aus. Unabhängig davon muss für praktische Implementierungen ein ausreichend großer Empfangspuffer verfügbar sein.

Bei einem Sequenznummernraum der Kardinalität N muss für das Sendefenster stets gelten:

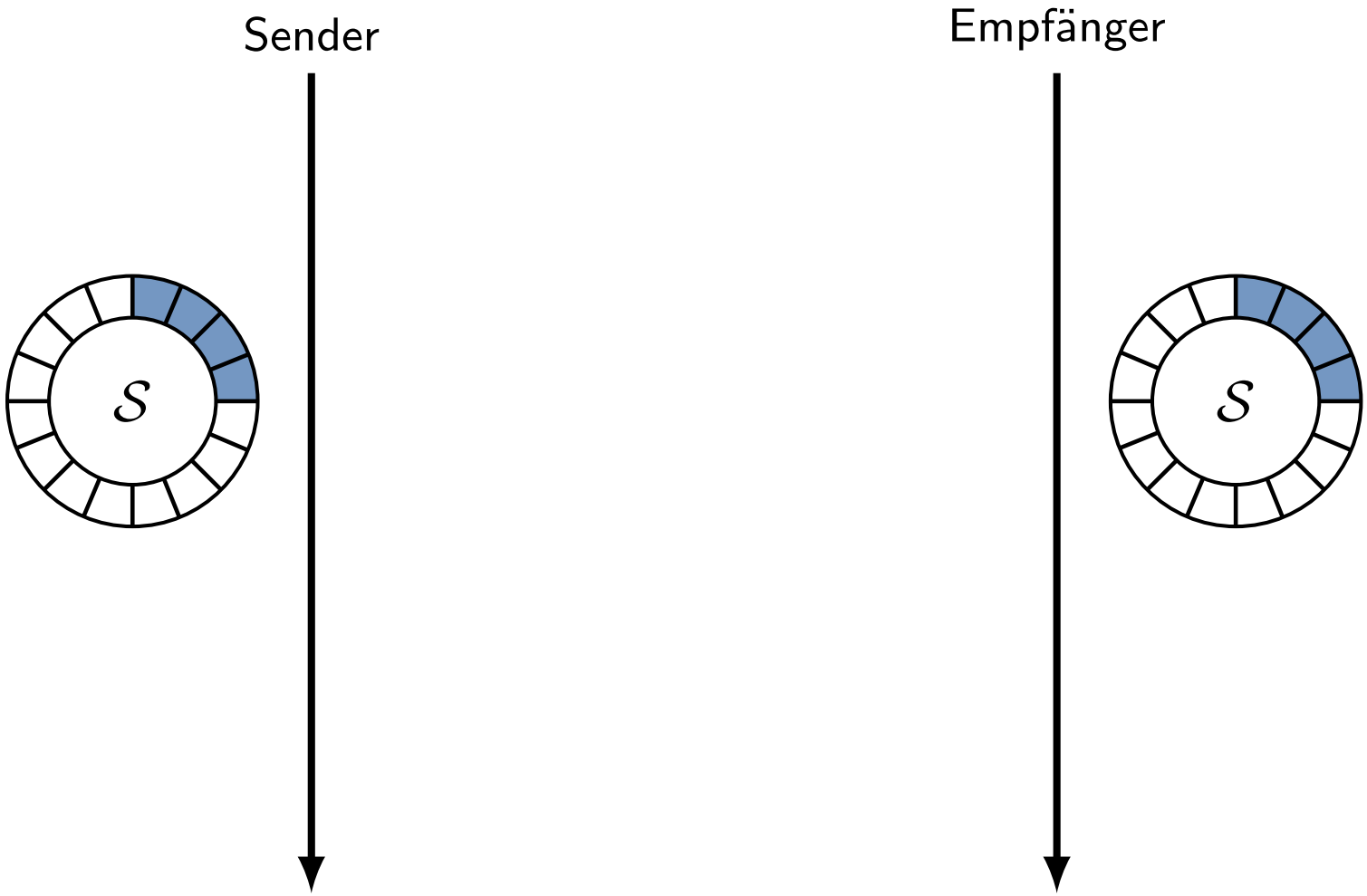
$$w_s \leq N - 1.$$

Andernfalls kann es zu Verwechslungen kommen (s. Übung).

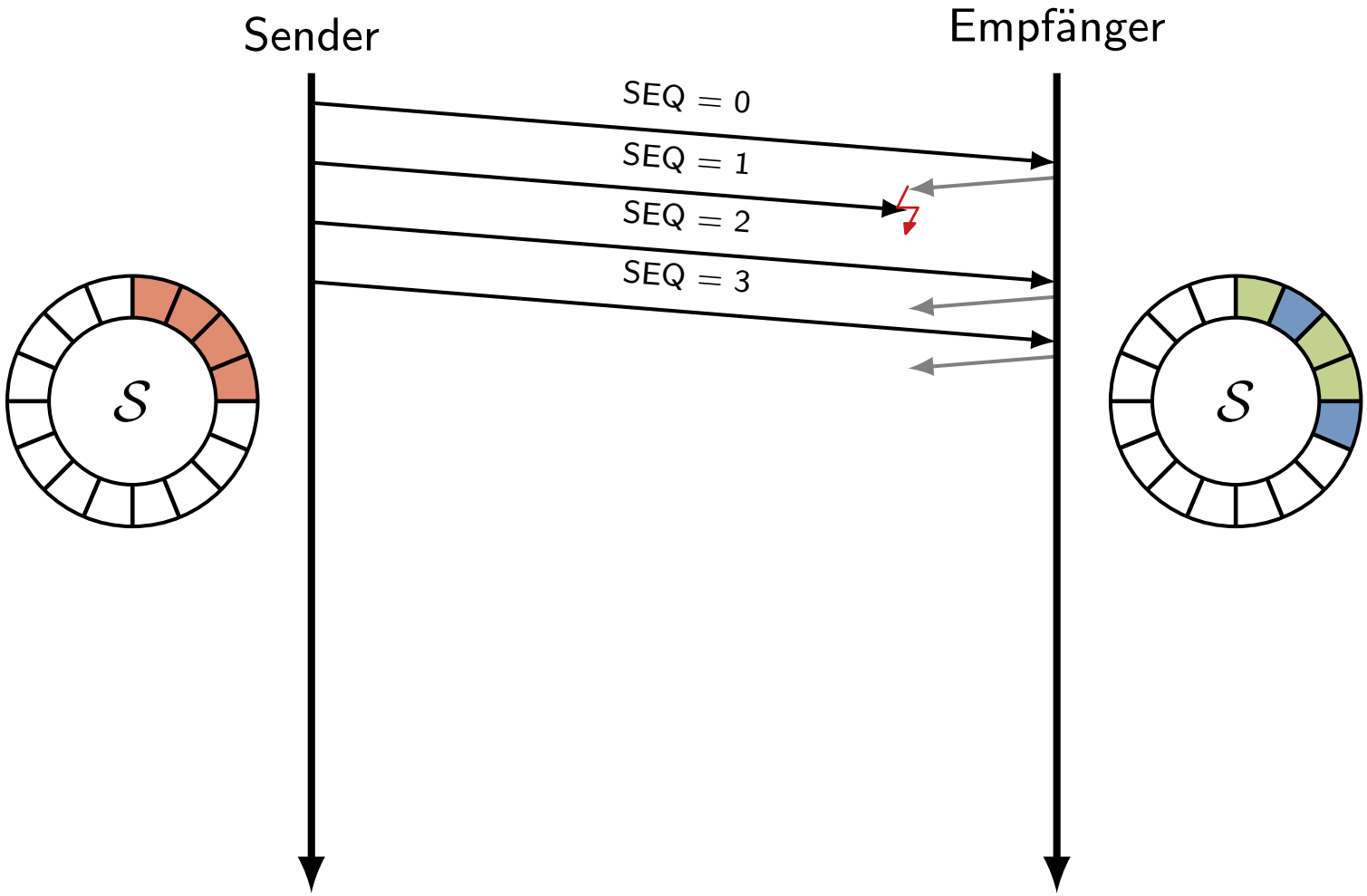
Das Verwerfen erfolgreich übertragener aber nicht in der erwarteten Reihenfolge eintreffender Segmente macht das Verfahren einfach zu implementieren aber weniger effizient.

Sliding-Window-Verfahren

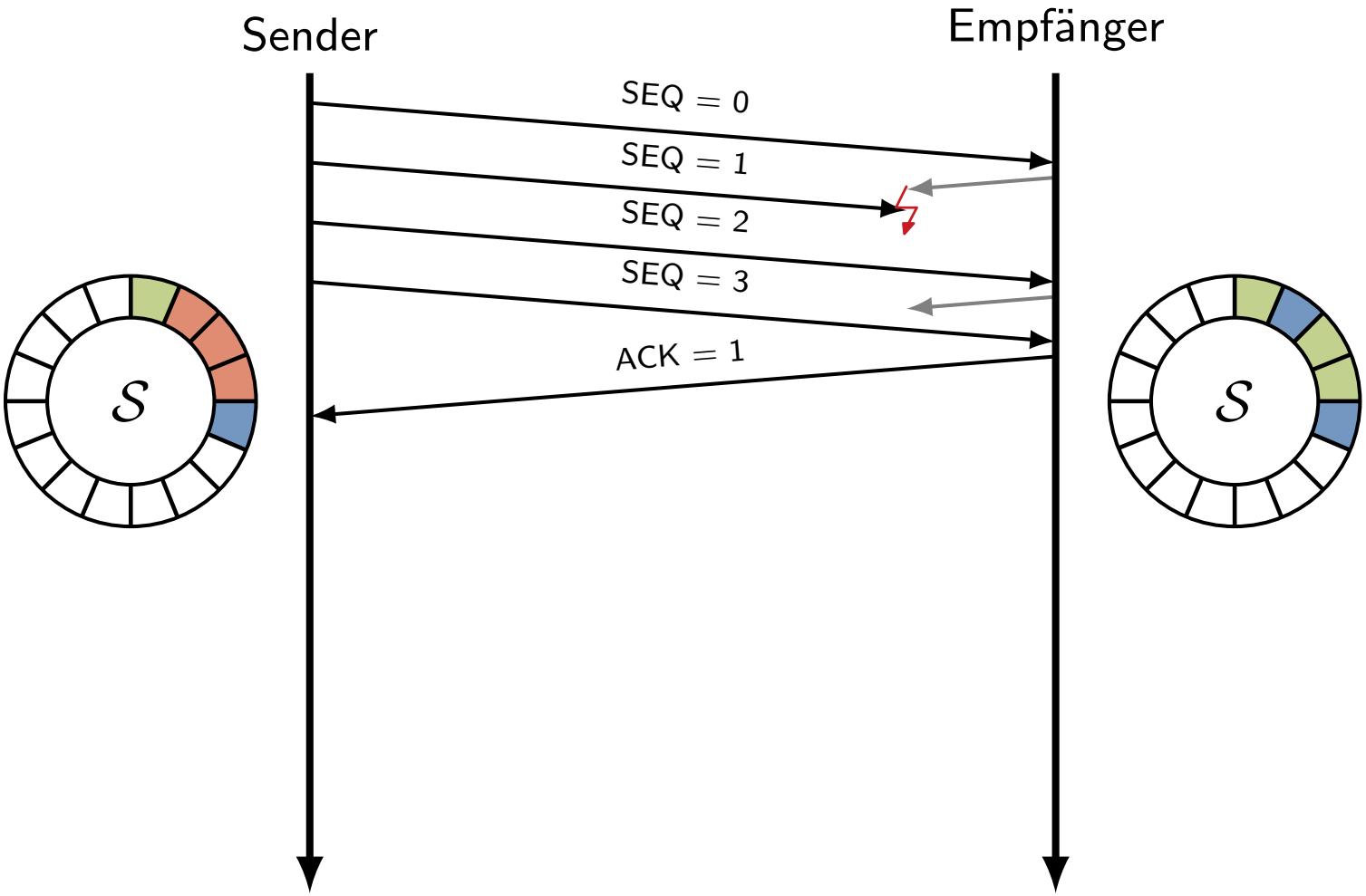
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



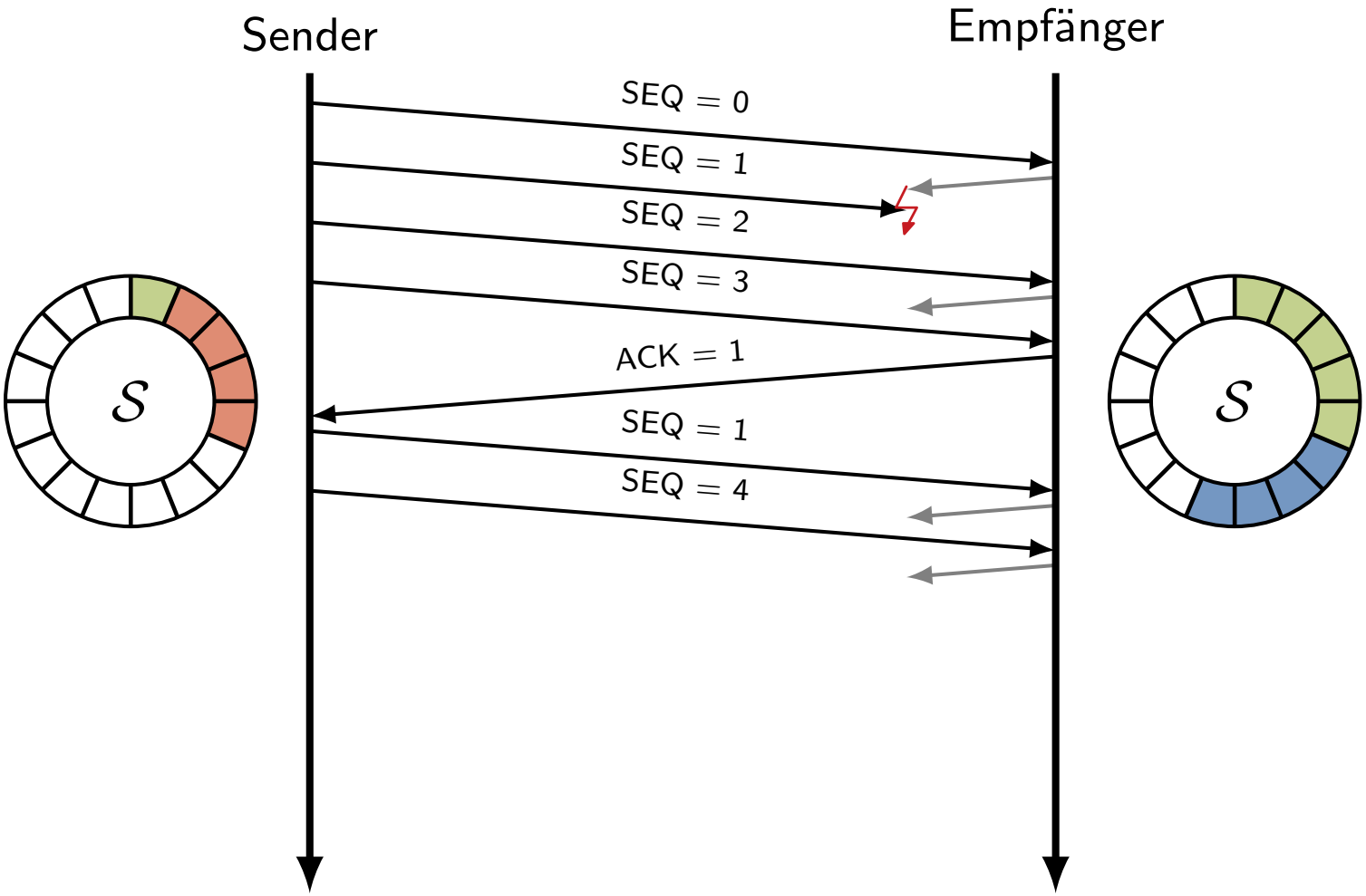
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



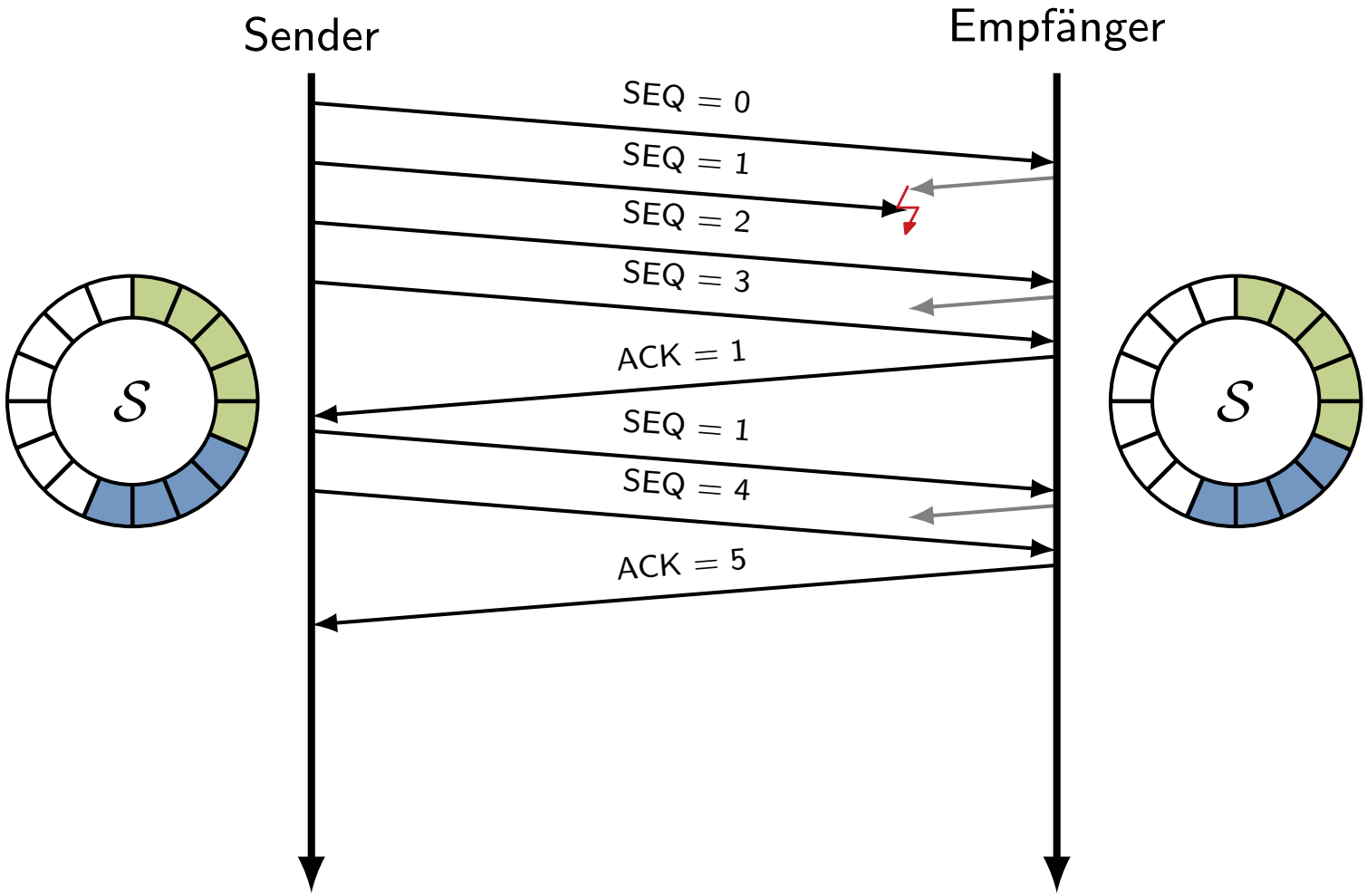
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Fluss- und Staukontrolle bei TCP

TCP-Flusskontrolle

Ziel der Flusskontrolle ist es, Überlastsituationen beim Empfänger zu vermeiden. Dies wird erreicht, indem der Empfänger eine Maximalgröße für das Sendefenster des Senders vorgibt.

- Empfänger teilt dem Sender über das Feld Receive Window im TCP-Header die aktuelle Größe des Empfangsfensters W_r mit.
- Der Sender interpretiert diesen Wert als die maximale Anzahl an Byte, die ohne Abwarten einer Bestätigung übertragen werden dürfen.
- Durch Herabsetzen des Wertes kann die Übertragungsrate des Senders gedrosselt werden, z. B. wenn sich der Empfangspuffer des Empfängers füllt.

TCP-Staukontrolle

Ziel der **Staukontrolle** ist es, Überlastsituationen im Netz zu vermeiden. Dazu muss der Sender Engpässe im Netz erkennen und die Größe des Sendefensters entsprechend anpassen.

Zu diesem Zweck wird beim Sender zusätzlich ein Staukontrollfenster (engl. Congestion Window) W_c eingeführt, dessen Größe wir mit w_c bezeichnen:

- W_c wird vergrößert, solange Daten verlustfrei übertragen werden.
- W_c wird verkleinert, wenn Verluste auftreten.
- Für das tatsächliche Sendefenster gilt stets $w_s = \min\{w_c, w_r\}$.

TCP-Staukontrolle

Man unterscheidet bei TCP grundsätzlich zwischen zwei Phasen der Staukontrolle:

1. Slow-Start:
   - Für jedes bestätigte Segment wird W_c um eine MSS vergrößert.
   - Dies führt zu exponentiellem Wachstum des Staukontrollfensters bis ein Schwellwert (engl. Congestion Threshold) erreicht ist.
   - Danach wird mit der Congestion-Avoidance-Phase fortgefahren.
2. Congestion Avoidance:
   - Für jedes bestätige Segment wird W_c lediglich um $(1/w_c)$ MSS vergrößert, d. h. nach Bestätigung eines vollständigen Staukontrollfensters um genau eine MSS.
   - Ein vollständiges Fenster kann frühestens nach 1 RTT bestätigt sein.
   - Dies führt zu linearem Wachstum des Staukontrollfensters in der RTT.

TCP-Varianten:

- Wir betrachten hier eine auf das Wesentliche reduzierte Implementierung von TCP, die auf TCP *Reno* basiert.
- Die einzelnen TCP-Version (*Tahoe, Reno, New Reno, Cubic, . . .*) unterscheiden sich in Details, sind aber alle zueinander kompatibel.
- Linux verwendet derzeit TCP *Cubic*, welches das Congestion Window schneller anwächsen lässt als andere TCP-Varianten.

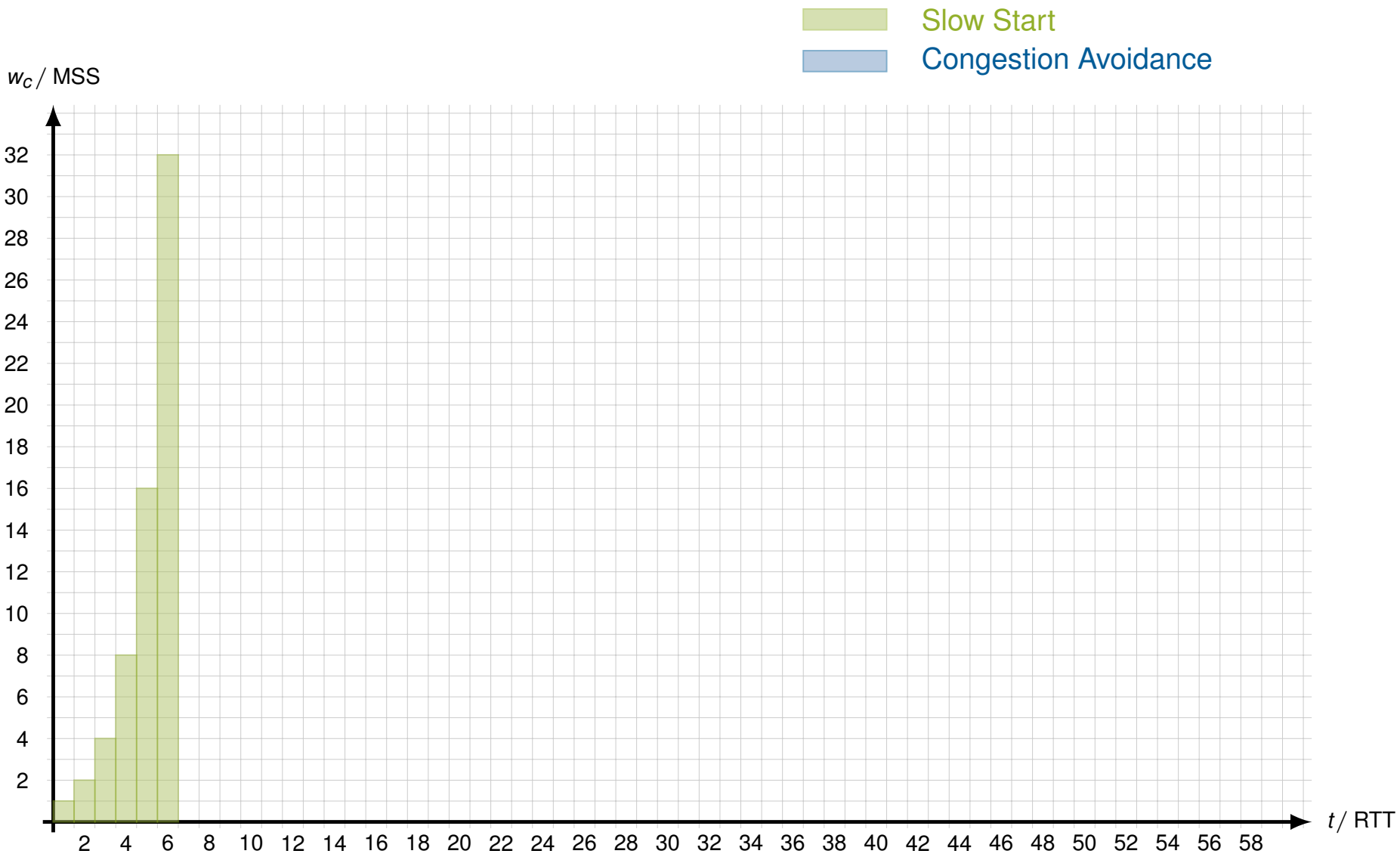
Fluss- und Staukontrolle bei TCP

Die folgende Beschreibung bezieht sich auf eine vereinfachte Implementierung von TCP *Reno*:

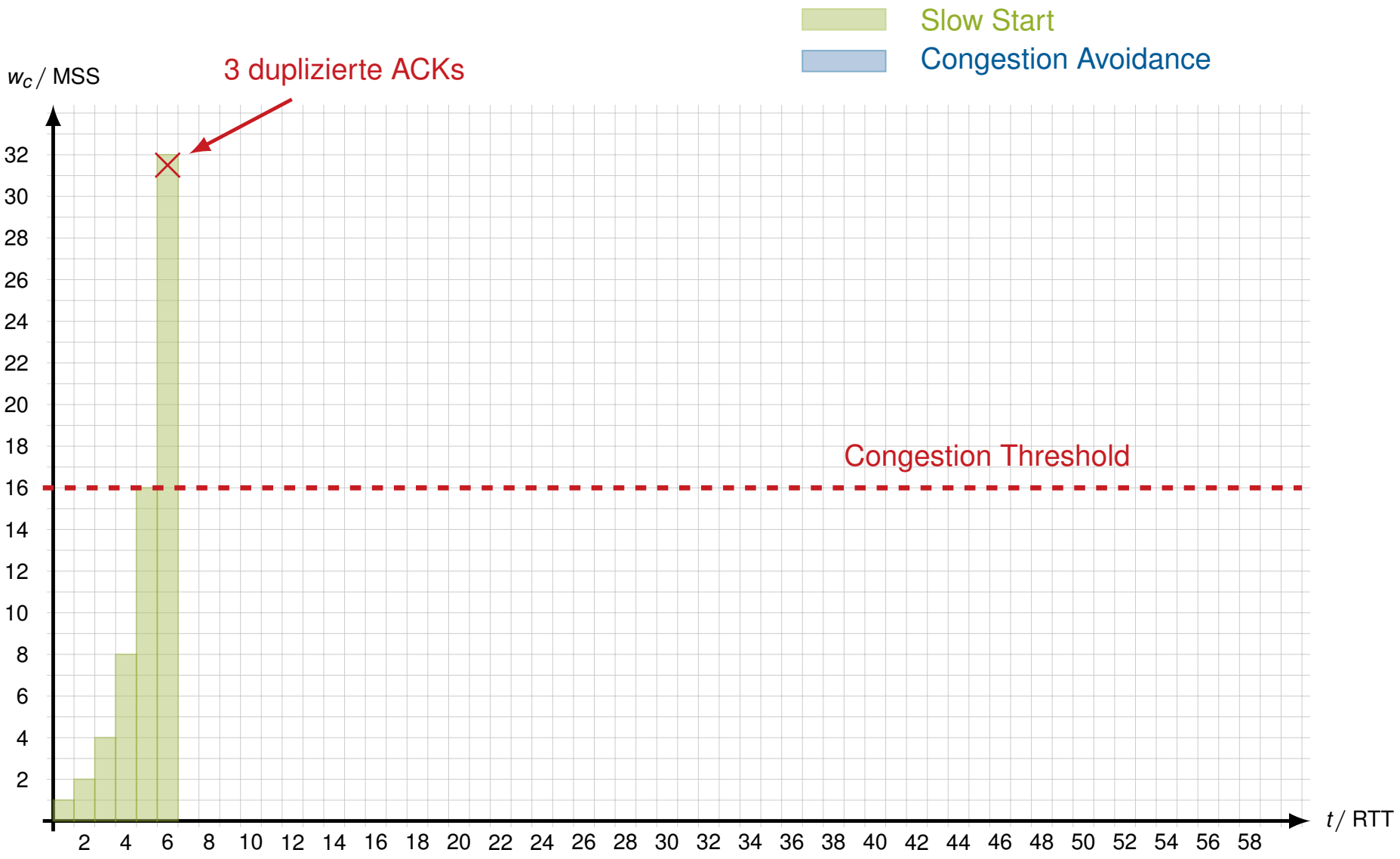
1. 3 duplizierte Bestätigungen (Duplicate ACKs)
   - Setze den Schwellwert für die Stauvermeidung auf $w_c/2$.
   - Reduziere W_c auf die Größe dieses Schwellwerts.
   - Beginne mit der Stauvermeidungsphase.
2. Timeout
   - Setze den Schwellwert für die Stauvermeidung auf $w_c/2$.
   - Setze $w_c = 1$ MSS.
   - Beginne mit einem neuen Slow-Start.

- Der Vorgänger TCP-Tahoe unterscheidet z. B. nicht zwischen diesen beiden Fällen und führt immer Fall 2 aus.
- Grundsätzlich sind alle TCP-Versionen kompatibel zueinander, allerdings können sich die unterschiedlichen Staukontrollverfahren gegenseitig nachteilig beeinflussen.

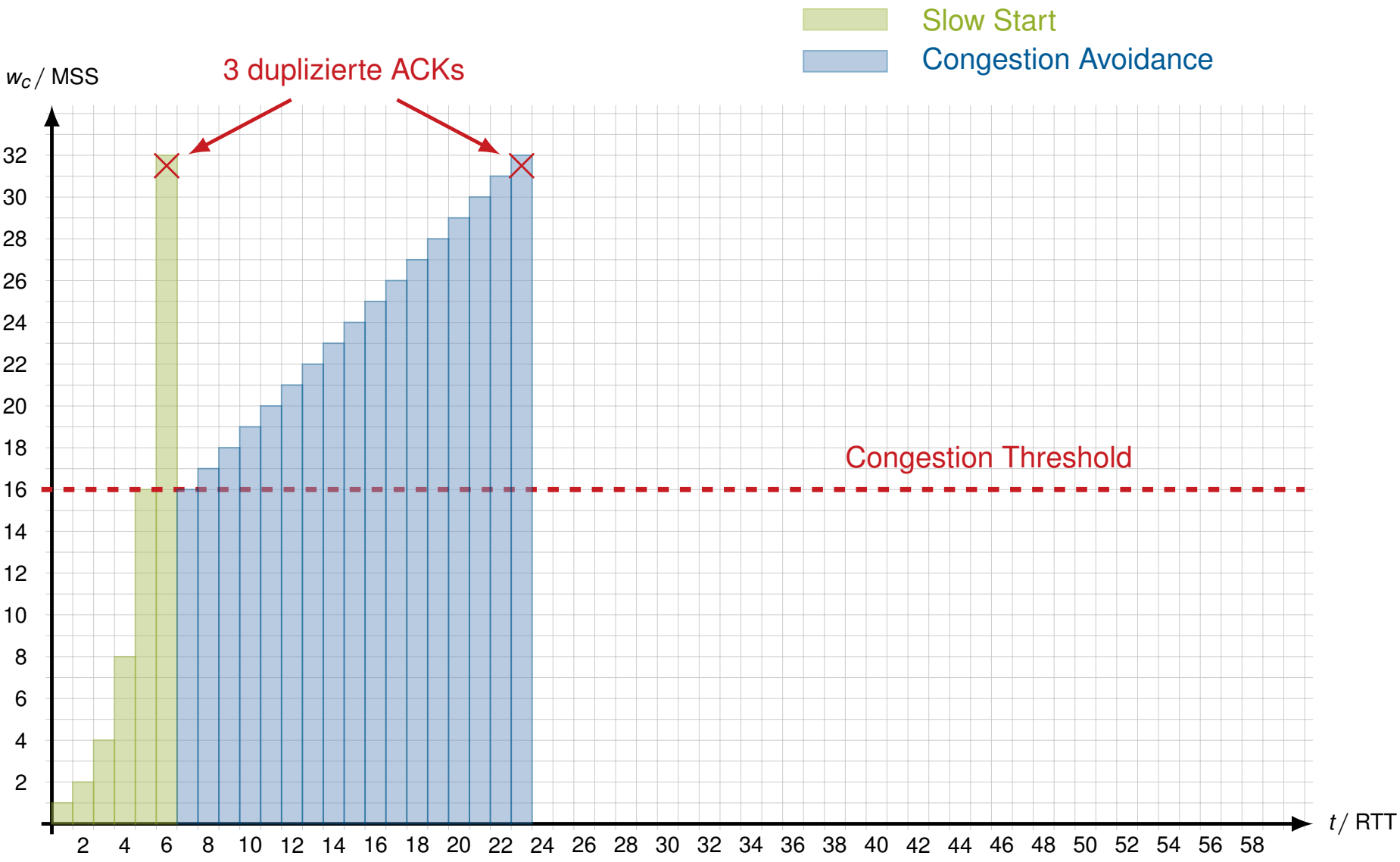
Beispiel: TCP-Reno (mit einigen Vereinfachungen)



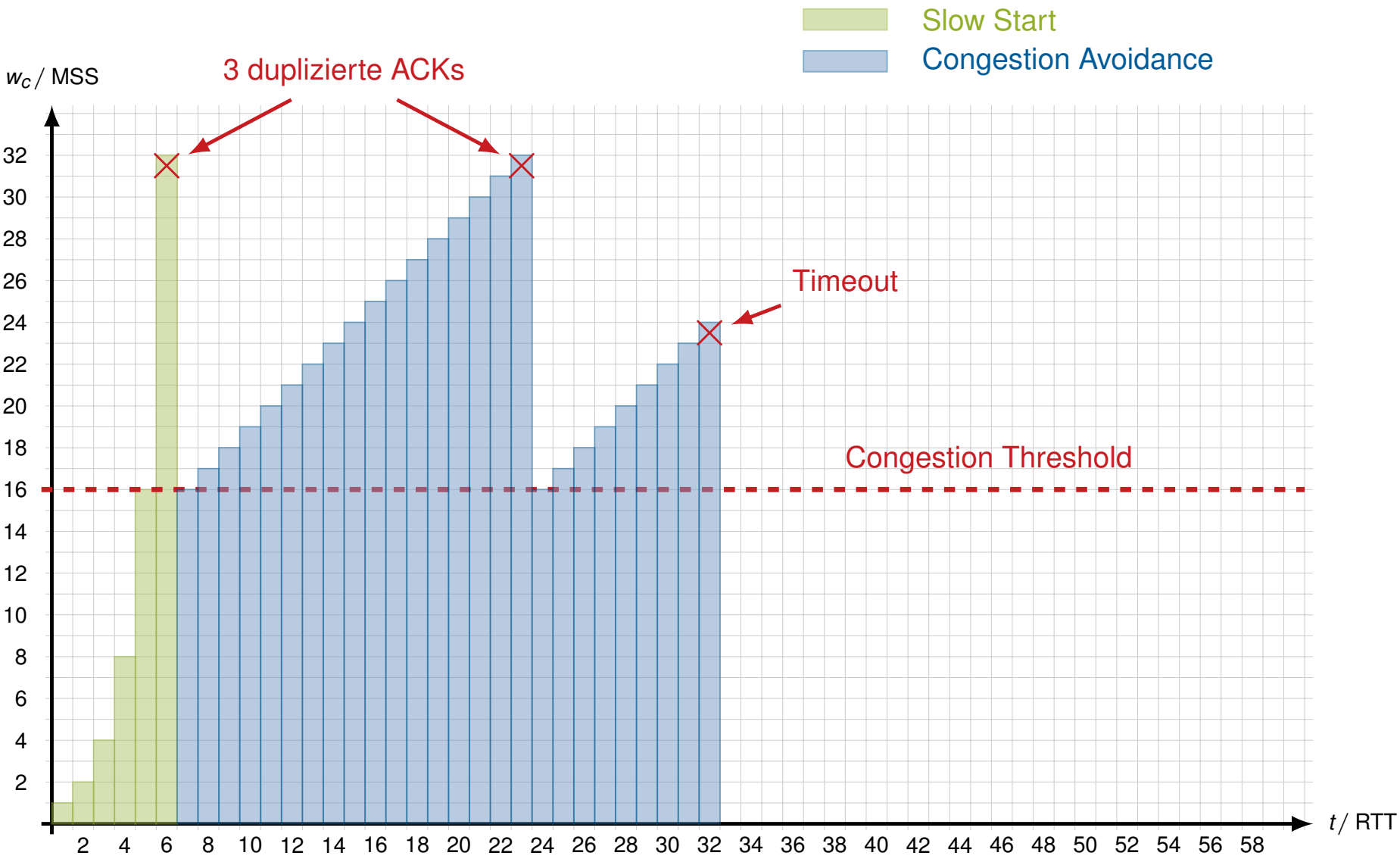
Beispiel: TCP-Reno (mit einigen Vereinfachungen)



Beispiel: TCP-Reno (mit einigen Vereinfachungen)



Beispiel: TCP-Reno (mit einigen Vereinfachungen)



Beispiel: TCP-Reno (mit einigen Vereinfachungen)