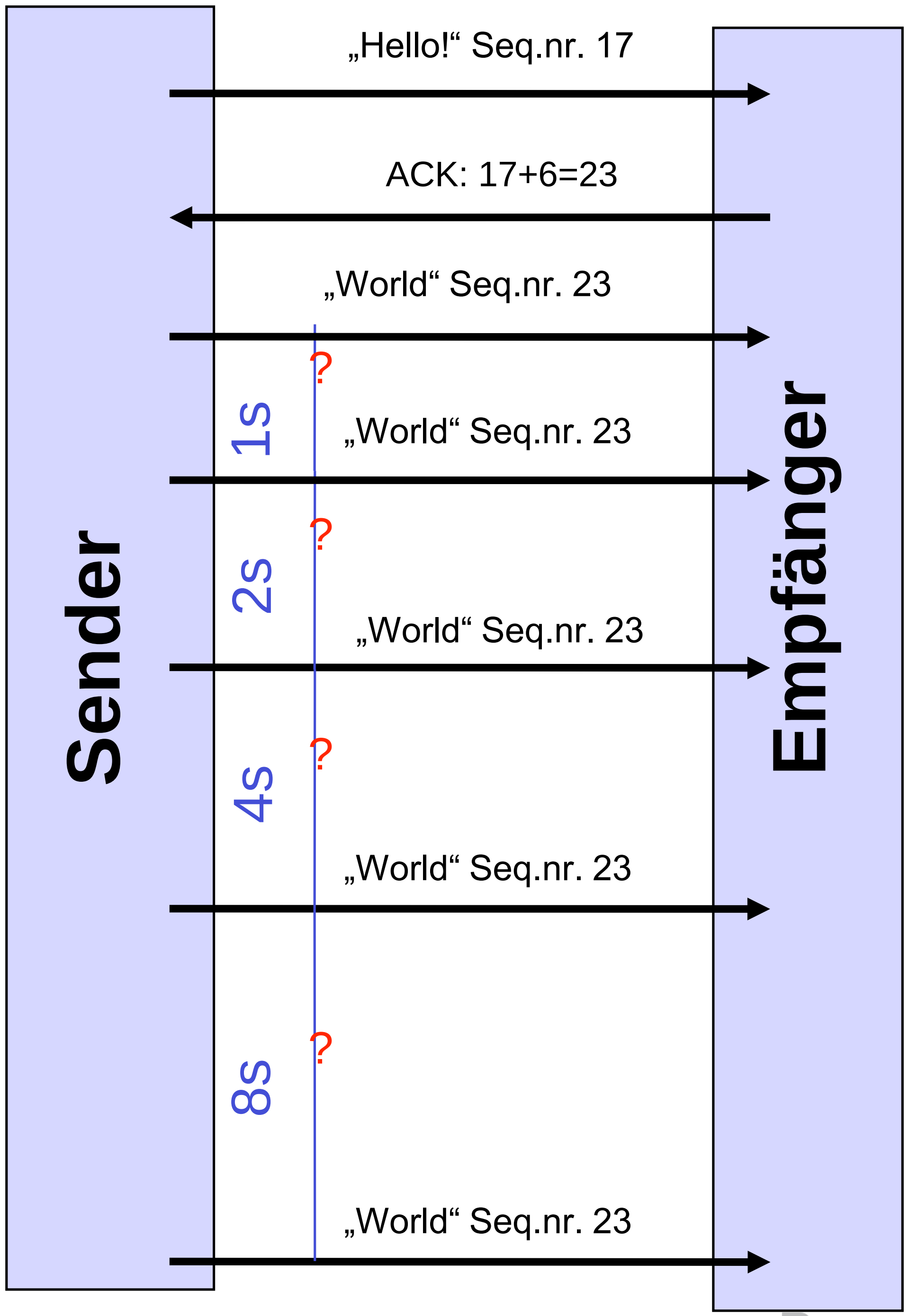
Exponentielles Zurückweichen

- Retransmission Timout (RTO)
  - regelt Zeitraum zwischen Senden von Datenduplikaten, falls Bestätigung ausbleibt
- Wann wird ein TCP-Paket nicht bestätigt?
  - Wenn die Bestätigung wesentlich länger benötigt, als die durchschnittliche Umlaufzeit (RTT/round trip time)
    - 1. Problem: Messung der RTT
    - 2. Problem: Bestätigung kommt, nur spät
  - Sender
    - Wartet Zeitraum gemäß RTO
    - Sendet Paket nochmal und setzt
    - RTO $\leftarrow$ 2 RTO   (bis RTO = 64 Sek.)
- Neuberechnung von RTO, wenn Pakete bestätigt werden



Schätzung der Umlaufzeit (RTT/Round Trip Time)

1: $R = M_4 \quad M_3 \quad M_2 \quad M_1 \quad M_0$

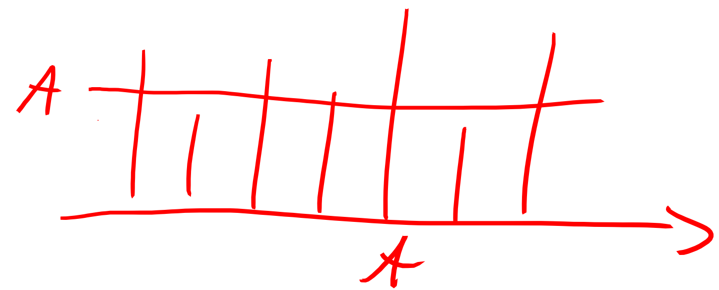
2: $R = 0{,}9 M_4 + 0{,}1 M_3$

3: $R = \cancel{0{,}1 M}\ 0{,}9^2 M_4 + 0{,}9 \cdot 0{,}1 M_3 + 0{,}1 M_2$

4: $R = 0{,}9^3 M_4 + 0{,}9^2 \cdot 0{,}1 M_3 + 0{,}9 \cdot 0{,}1 M_2 + 0{,}1 M_1$

$R = \sum_i 0{,}9^{i-1}\, 0{,}1\, M_i$

- TCP-Paket gilt als nicht bestätigt, wenn Bestätigung „wesentlich“ länger dauert als RTO
  - RTT nicht on-line berechenbar (nur rückblickend)
  - RTT schwankt stark
- Daher: Retransmission Timeout Value aus großzügiger Schätzung:
  - RFC 793: (M := letzte gemessene RTT)
    - $R \leftarrow \alpha R + (1-\alpha) M$, wobei $\alpha = 0{,}9$
    - $RTO \leftarrow \beta R$, wobei $\beta = 2$
  - Jacobson 88: Schätzung nicht robust genug, daher
    - $A \leftarrow A + g\,(M - A)$, wobei $g = 1/8$ $\qquad A \leftarrow (1-g)A + g M$
    - $D \leftarrow D + h\,(|M - A| - D)$, wobei $h = 1/4$
    - $RTO \leftarrow A + 4D$
- Aktualisierung nicht bei mehrfach versandten Pakete



TCP - Algorithmus von Nagle

- Wie kann man sicherstellen,
  - dass kleine Pakete zeitnah ausgeliefert werden
  - und bei vielen Daten große Pakete bevorzugt werden?
- Algorithmus von Nagle:
  - Kleine Pakete werden nicht versendet, solange Bestätigungen noch ausstehen.
    - Paket ist klein, wenn Datenlänge < MSS
  - Trifft die Bestätigung des zuvor gesendeten Pakets ein, so wird das nächste verschickt.
- Beispiel:
  - Telnet versus ftp
- Eigenschaften
  - Selbst-taktend: Schnelle Verbindung = viele kleine Pakete

Flusskontrolle

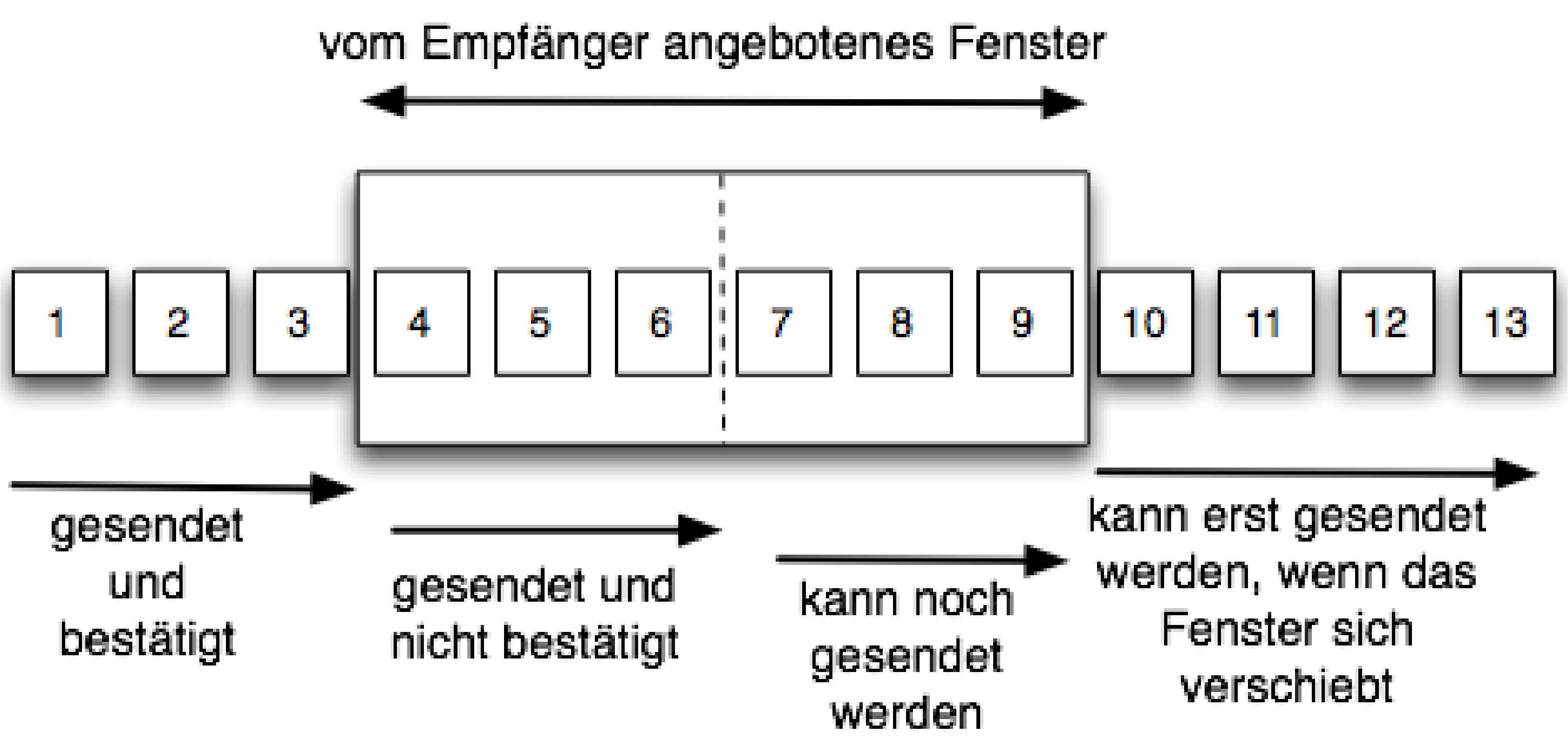
- Problem: Schneller Sender und langsamer Empfänger
  - Der Sender lässt den Empfangspuffer des Empfängers überlaufen
  - Übertragungsbandweite wird durch sinnlosen Mehrfachversand (nach Fehlerkontrolle) verschwendet
- Anpassung der Frame-Sende-Rate an dem Empfänger notwendig

Langsamer Empfänger

Schneller Sender

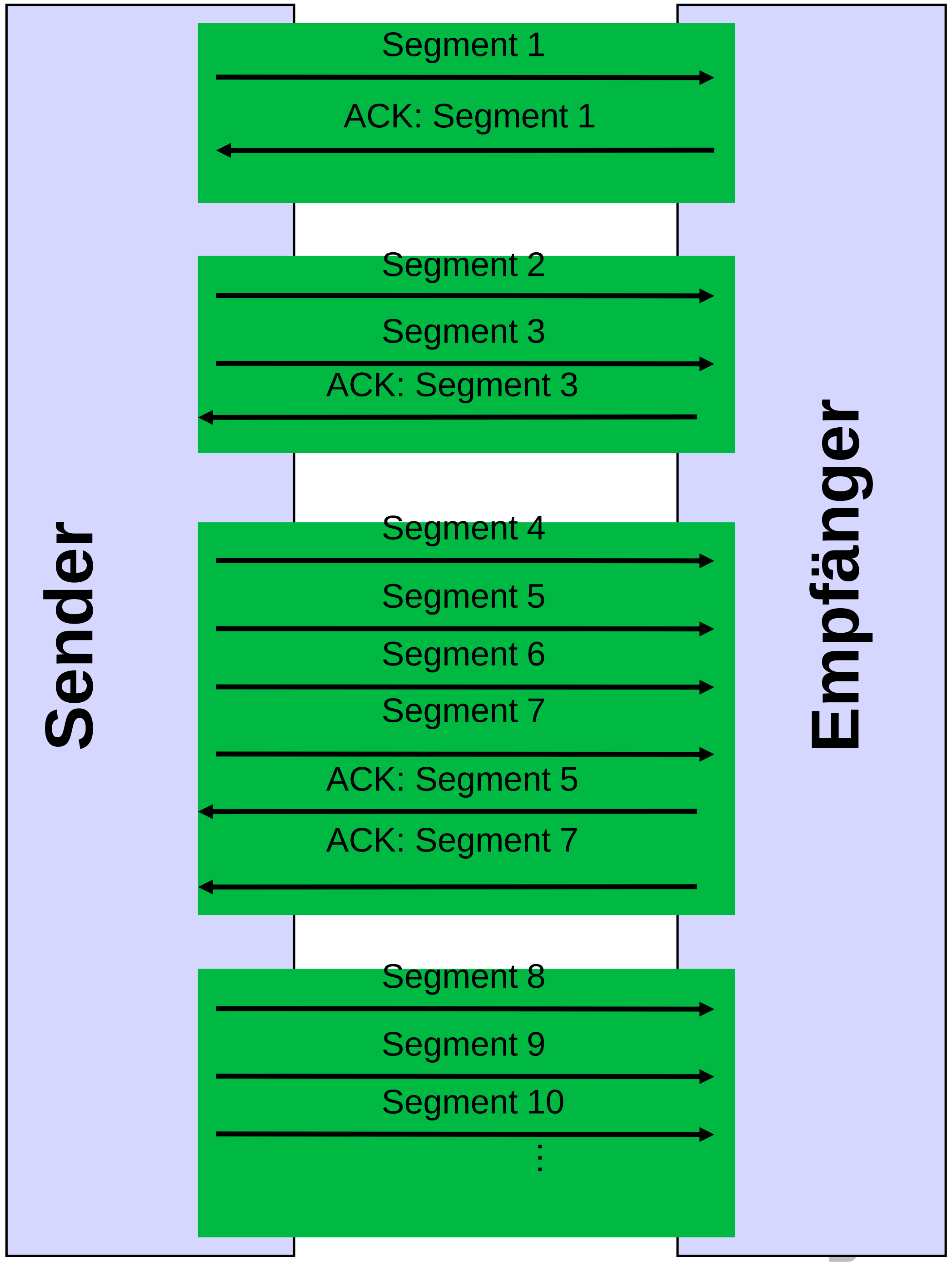
Gleitende Fenster (sliding windows)

- Datenratenanpassung durch Fenster
  - Empfänger bestimmt Fenstergröße (wnd) im TCP-Header der ACK-Segmente
  - Ist Empfangspuffer des Empfängers voll, sendet er wnd=0
  - Andernfalls sendet Empfänger wnd>0
- Sender beachtet:
  - Anzahl unbestätigter gesender Daten ≤ Fenstergröße



Slow Start
Congestion Fenster

- Sender darf vom Empfänger angebotene Fenstergröße nicht von Anfang wahrnehmen
- 2. Fenster: Congestion-Fenster (cwnd/Congestion window)
  - Von Sender gewählt (FSK)
  - Sendefenster: min {wnd,cwnd}
  - S: Segmentgröße
  - Am Anfang:
    - cwnd $\leftarrow$ S
  - Für jede empfangene Bestätigung:
    - cwnd $\leftarrow$ cwnd + S
  - Solange bis einmal Bestätigung ausbleibt
- „Slow Start“ = Exponentielles Wachstum



TCP Tahoe: Congestion Avoidance

x: Anzahl Pakete pro RTT

- Jacobson 88:
  - Parameter: cwnd und Slow-Start-Schwellwert (ssthresh=slow start threshold)
  - S = Datensegmentgröße = maximale Segmentgröße
- Verbindungsaufbau:
  - cwnd $\leftarrow$ S ssthresh $\leftarrow$ 65535

  x $\leftarrow$ 1 **y $\leftarrow$ max**

- Bei Paketverlust, d.h. Bestätigungsdauer > RTO,
  - multiplicatively decreasing
    cwnd $\leftarrow$ S ssthresh $\leftarrow$ $\max\left\{2S, \frac{1}{2}\min\{\mathrm{cwnd}, \mathrm{wnd}\}\right\}$

  x $\leftarrow$ 1 **y $\leftarrow$ x/2**

- Werden Segmente bestätigt und cwnd ≤ ssthresh, dann
  - slow start: cwnd $\leftarrow$ cwnd + S

  x $\leftarrow$ 2⊕x, bis x = y

- Werden Segmente bestätigt und cwnd > ssthresh, dann additively increasing

$$\mathrm{cwnd} \leftarrow \mathrm{cwnd} + S\,\frac{\cancel{S}}{\cancel{cwnd}} \cdot \frac{\cancel{cwnd}}{\cancel{S}}$$

pro RTT $\frac{cwnd}{S}$

$= cwnd + S$

x $\leftarrow$ x +1

TCP Tahoe

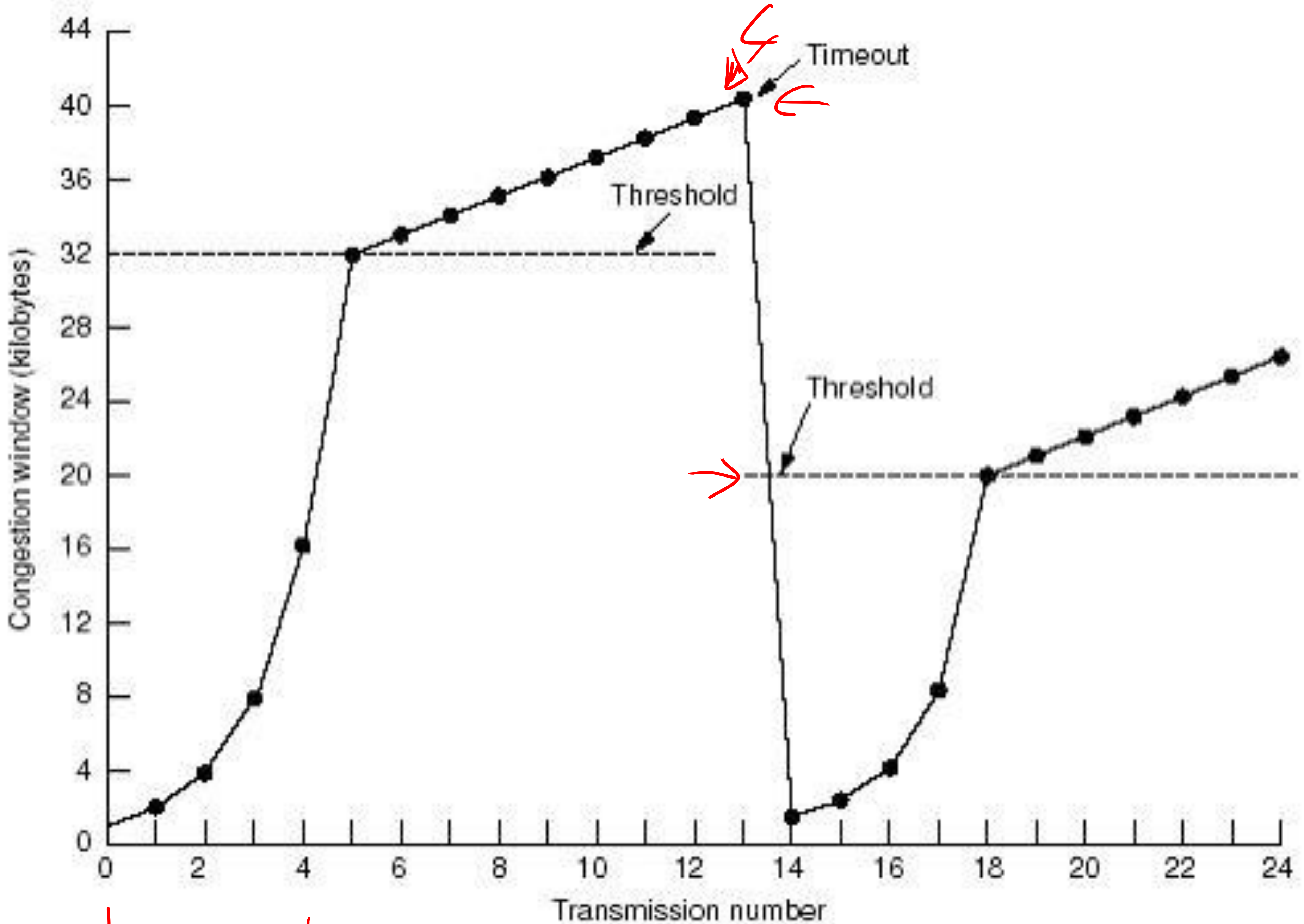
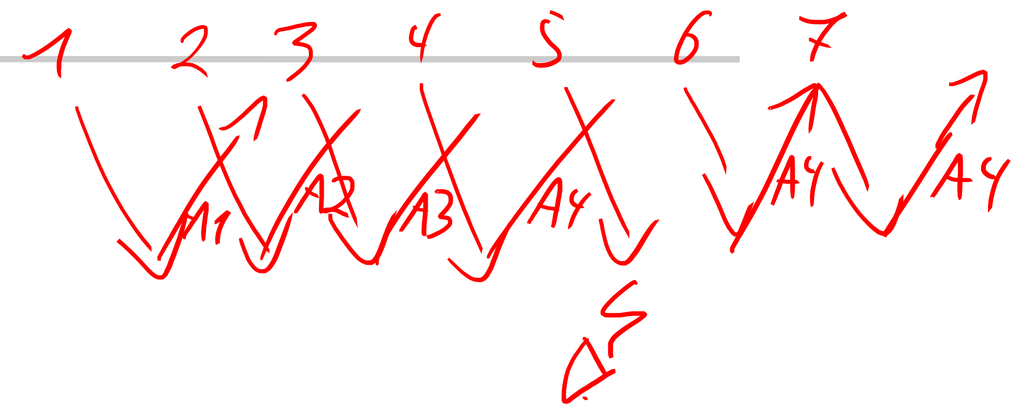


Fig3

Slow Start | lineares Wachstum | Slow Start

pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

Fast Retransmit und Fast Recovery



- TCP Tahoe [Jacobson 1988]:
  - Geht nur ein Paket verloren, dann
    - Wiederversand Paket + Restfenster
    - Und gleichzeitig Slow Start
  - Fast retransmit
    - Nach drei Bestätigungen desselben Pakets (triple duplicate ACK),
    - sende Paket nochmal, starte mit Slow Start
- TCP Reno [Stevens 1994]
  - Nach Fast retransmit:
    - ssthresh $\leftarrow$ min(wnd,cwnd)/2
    - cwnd $\leftarrow$ ssthresh + 3 S
  - Fast recovery nach Fast retransmit
    - Erhöhe Paketrate mit jeder weiteren Bestätigung
    - cwnd $\leftarrow$ cwnd + S
  - Congestion avoidance: Trifft Bestätigung von P+x ein:
    - cwnd $\leftarrow$ ssthresh

y $\leftarrow$ x/2

x $\leftarrow$ y + 3

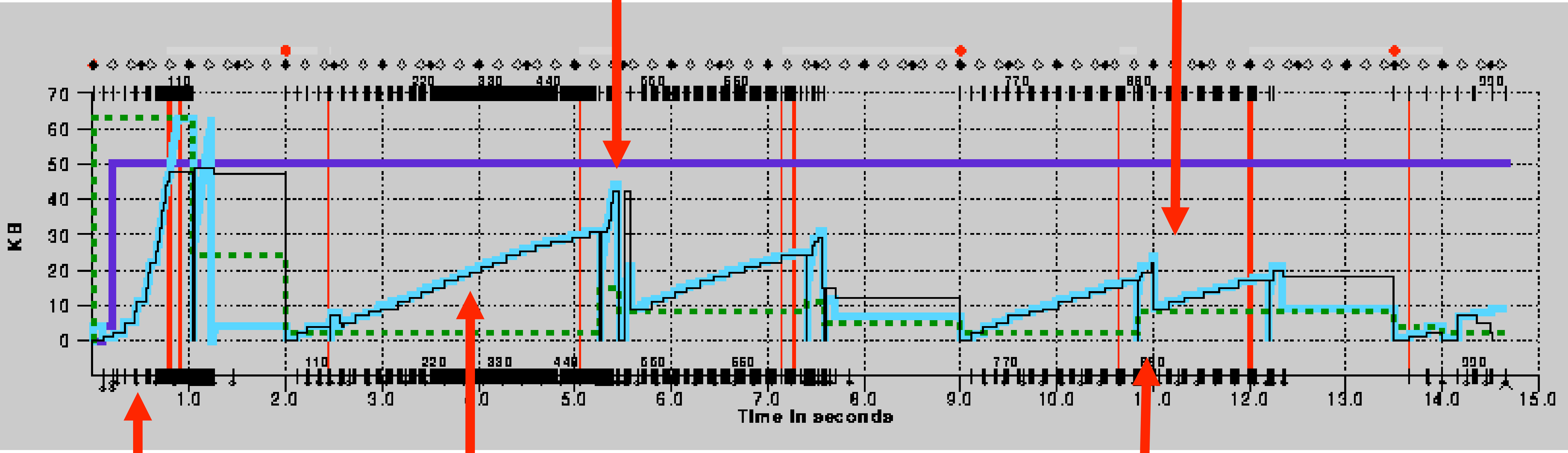
Stauvermeidungsprinzip: AIMD

- Kombination von TCP und Fast Recovery verhält sich im wesentlichen wie folgt:
  - Verbindungsaufbau: $\boxed{x \leftarrow 1}$
  - Bei Paketverlust, MD:multiplicative decreasing $\boxed{x \leftarrow x/2}$
  - Werden Segmente bestätigt, AI: additive increasing $\boxed{x \leftarrow x+1}$

Beispiel: TCP Reno in Aktion

Fast Retransmit

Fast Recovery



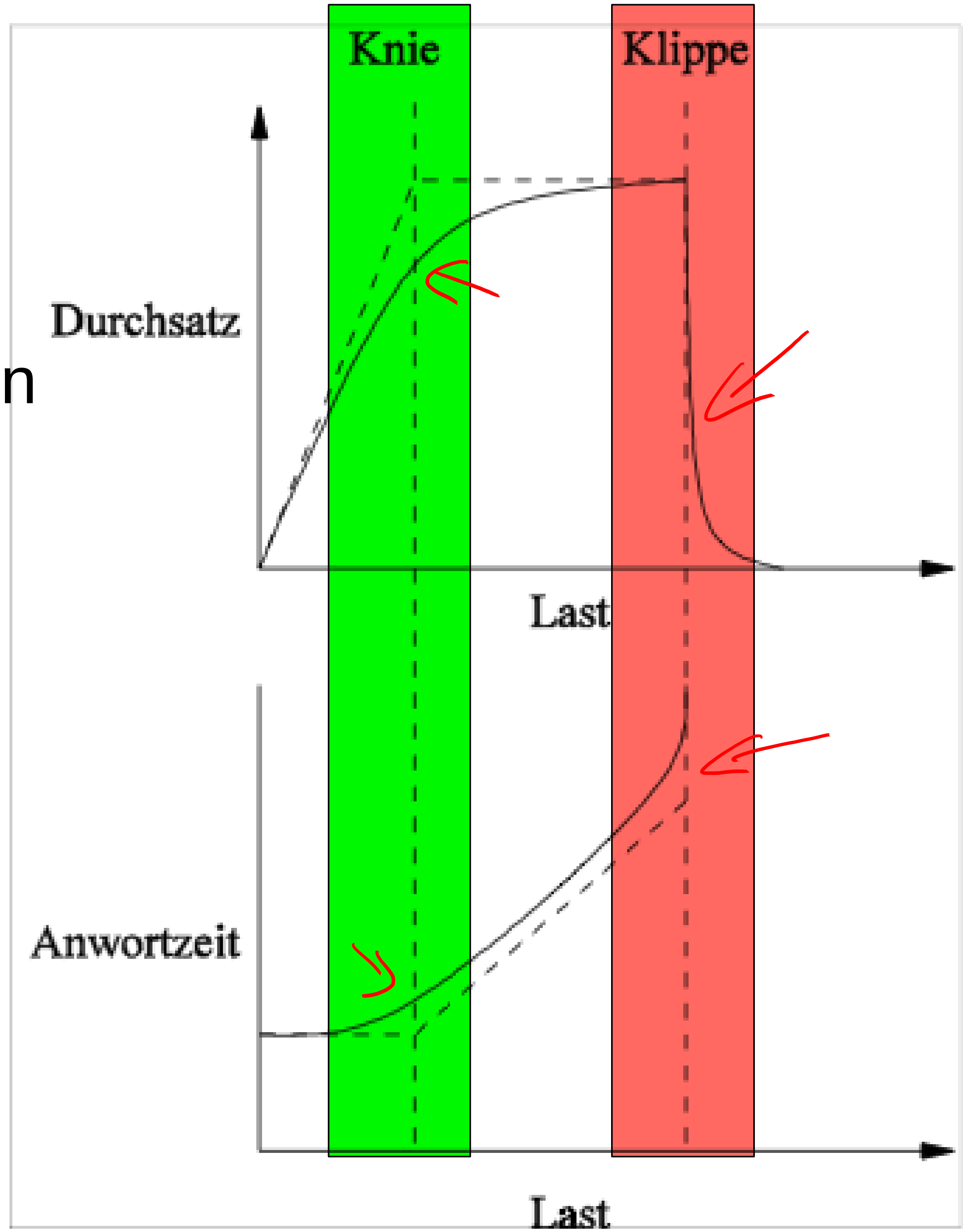
Additively Increase

Slow Start

Multiplicatively Decrease

Durchsatz und Antwortzeit

- Klippe:
  - Hohe Last
  - Geringer Durchsatz
  - Praktisch alle Daten gehen verloren
- Knie:
  - Hohe Last
  - Hoher Durchsatz
  - Einzelne Daten gehen verloren



Ein einfaches Datenratenmodell

- n Teilnehmer, Rundenmodell
  - Teilnehmer i hat Datenrate $x_i(t)$
  - Anfangsdatenrate $x_1(0), \ldots, x_n(0)$ gegeben
- Feedback nach Runde t:
  - y(t) = 0, falls $\sum_{i=1}^{n} x_i(t) \leq K$
  - y(t) = 1, falls $\sum_{i=1}^{n} x_i(t) > K$
  - wobei K ist Knielast
- Jeder Teilnehmer aktualisiert in Runde t+1:
  - $x_i(t+1) = f(x_i(t), y(t))$
  - Increase-Strategie $f_0(x) = f(x,0)$
  - Decrease-Strategie $f_1(x) = f(x,1)$
- Wir betrachten lineare Funktionen:

$$f_0(x) = a_I + b_I x \qquad \text{und} \qquad f_1(x) = a_D + b_D x \ .$$

Lineare Datenratenanpassung

- Interessante Spezialfälle:
  - AIAD: Additive Increase Additive Decrease

    $$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = a_D + x\ ,$$

    wobei $a_I > 0$ und $a_D < 0$.

  - MIMD: Multiplicative Increase/Multiplicative Decrease

    $$f_0(x) = b_I x \quad \text{und} \quad f_1(x) = b_D x\ ,$$

    wobei $b_I > 1$ und $b_D < 1$.

  - AIMD: Additive Increase Multiplicative Decrease

    $$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = b_D x\ ,$$

    wobei $a_I > 0$ und $b_D < 1$.

Fairness und Effizienz

- Effizienz
  - Last: $$X(t) := \sum_{i=1}^{n} x_i(t)$$
  - Maß $$|X(t) - K|$$

0 = gut
>0 = schlecht

- Fairness: Für $x=(x_1, \ldots, x_n)$:

$$F(x) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} (x_i)^2} .$$

  - $1/n \leq F(x) \leq 1$
  - $F(x) = 1 \leftrightarrow$ absolute Fairness
  - Skalierungsunabhängig
  - Kontinuierlich, stetig, differenzierbar
  - Falls k von n fair, Rest 0, dann $F(x) = k/n$

1. $x_1 = x_2 = \ldots = x_n = x$

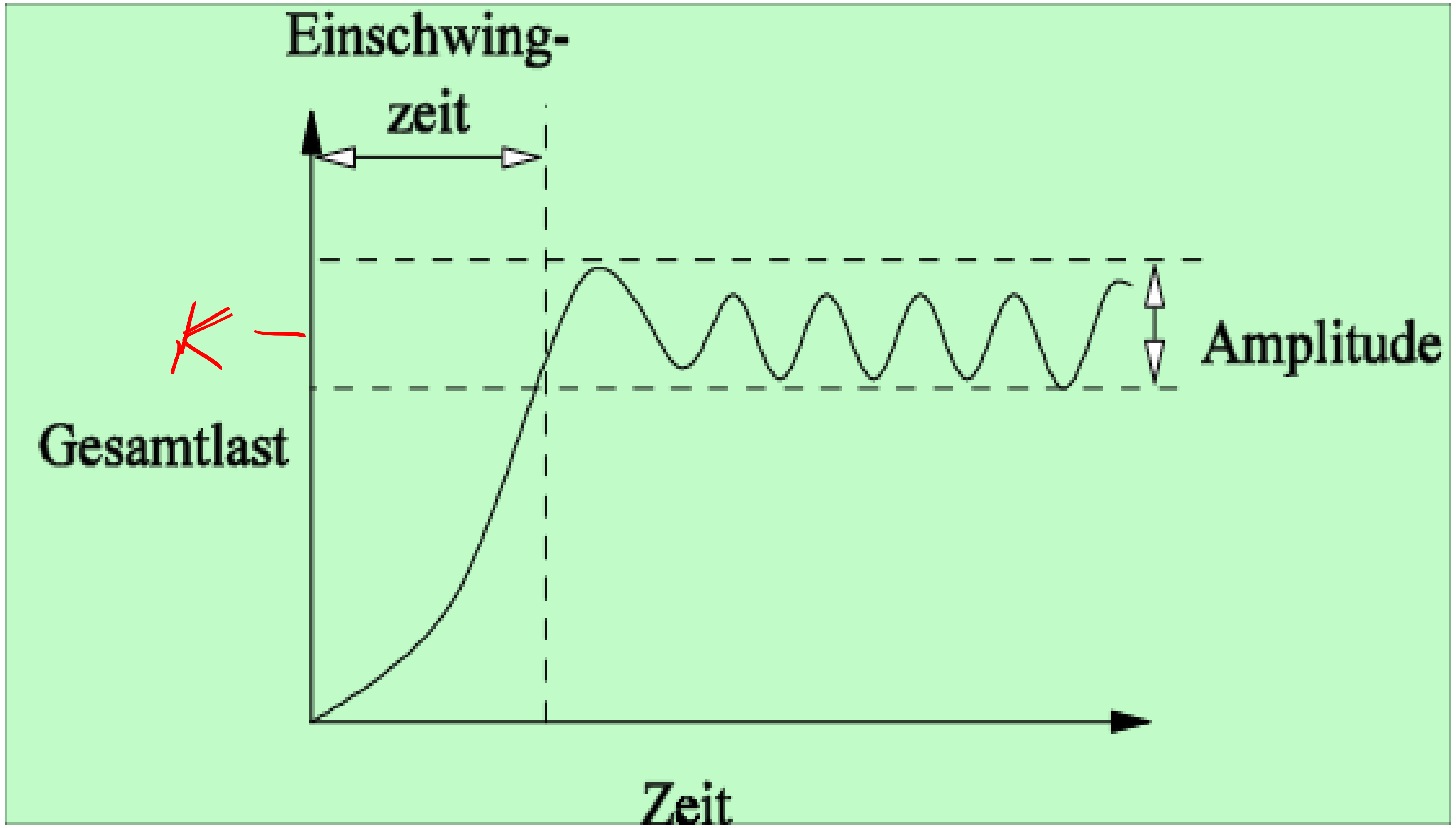
$F(x) = \frac{(n \cdot x)^2}{n \cdot n \cdot x^2} = 1$

2. $x_1 = x, \quad x_2 = \ldots = x_n = 0$

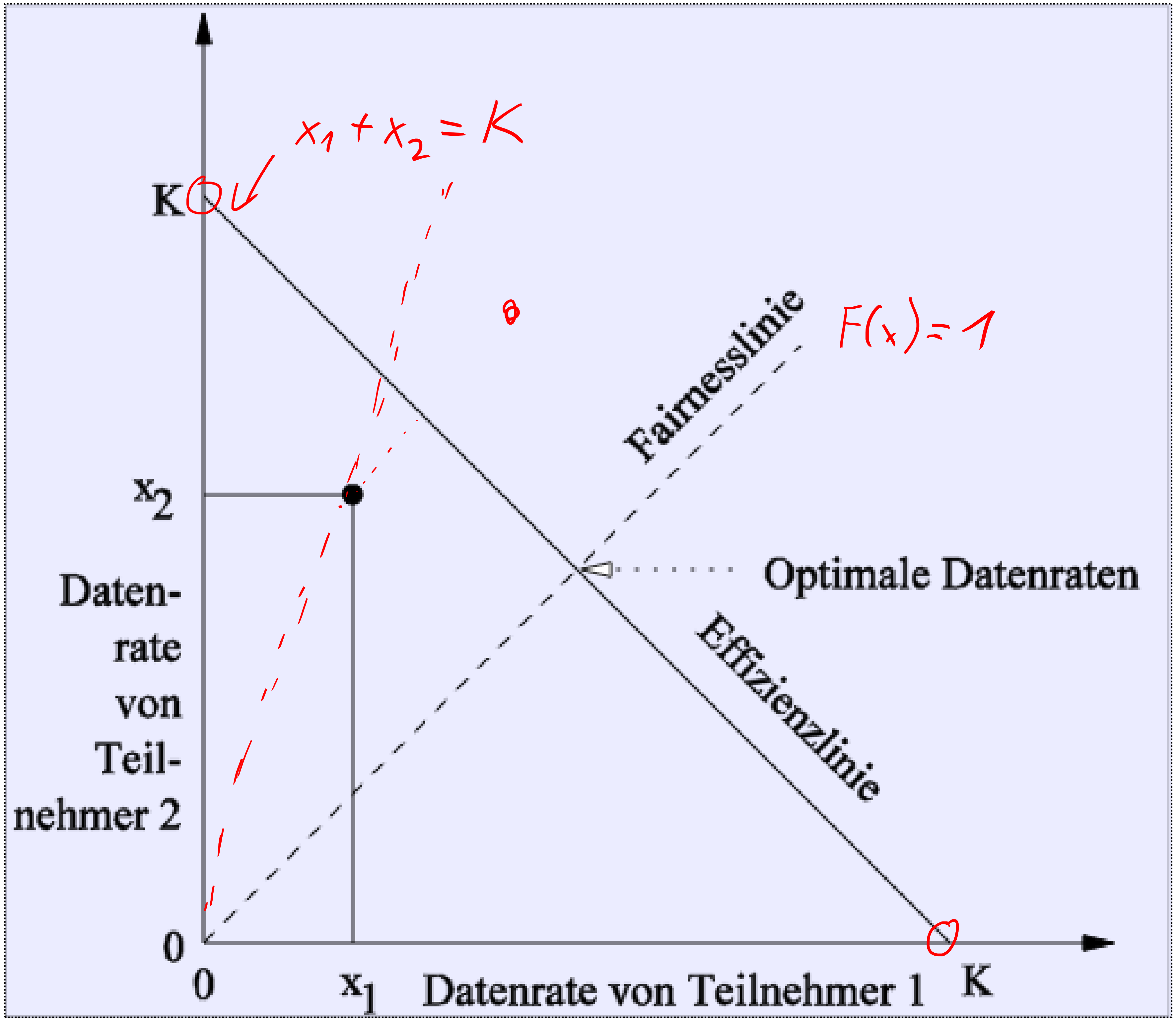
$F(x) = \frac{(x)^2}{n \cdot x^2} = \frac{1}{n}$

Konvergenz

- Konvergenz unmöglich
- Bestenfalls Oszillation um Optimalwert
  - Oszillationsamplitude A
  - Einschwingzeit T

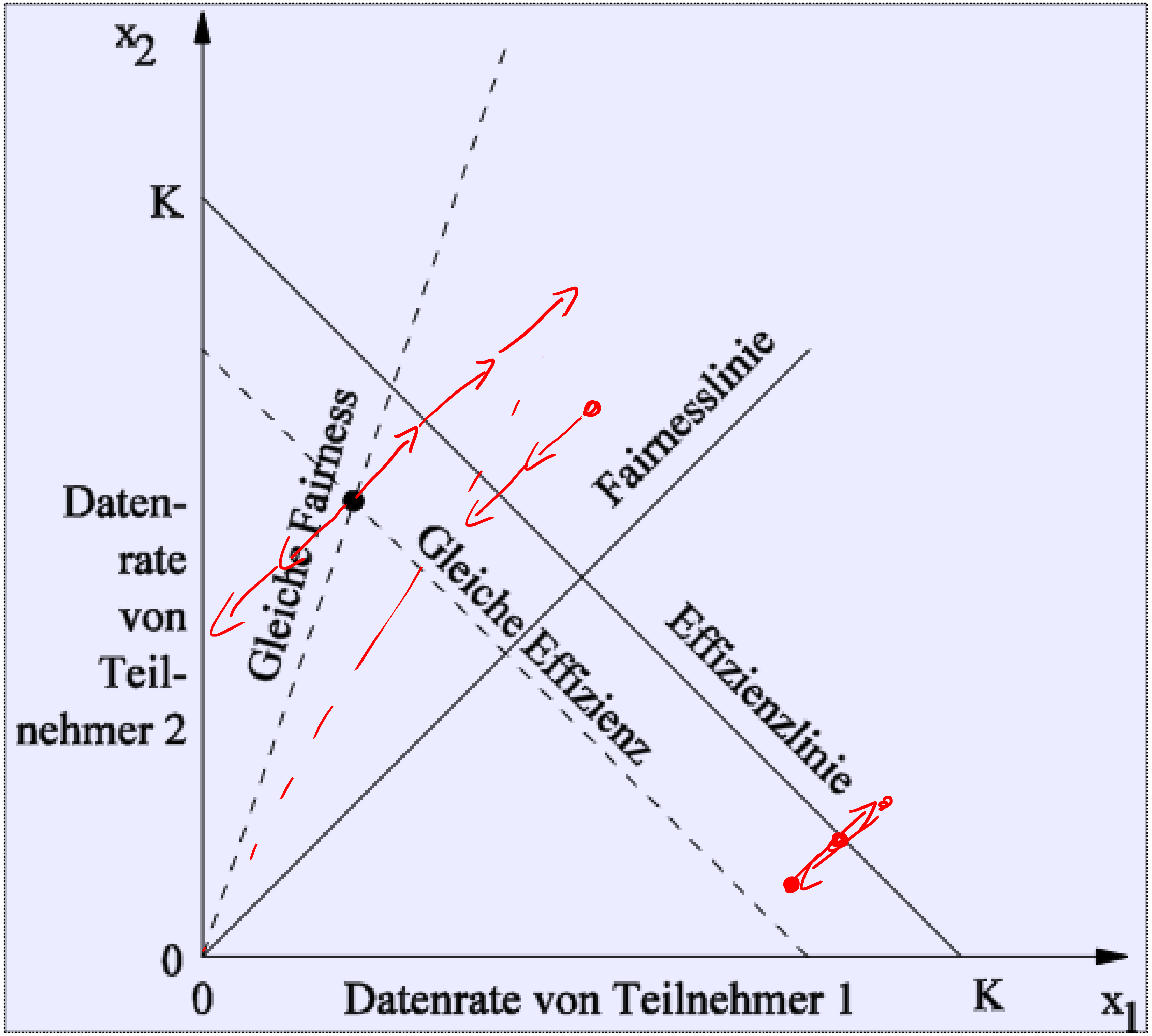


Vektordarstellung (I)

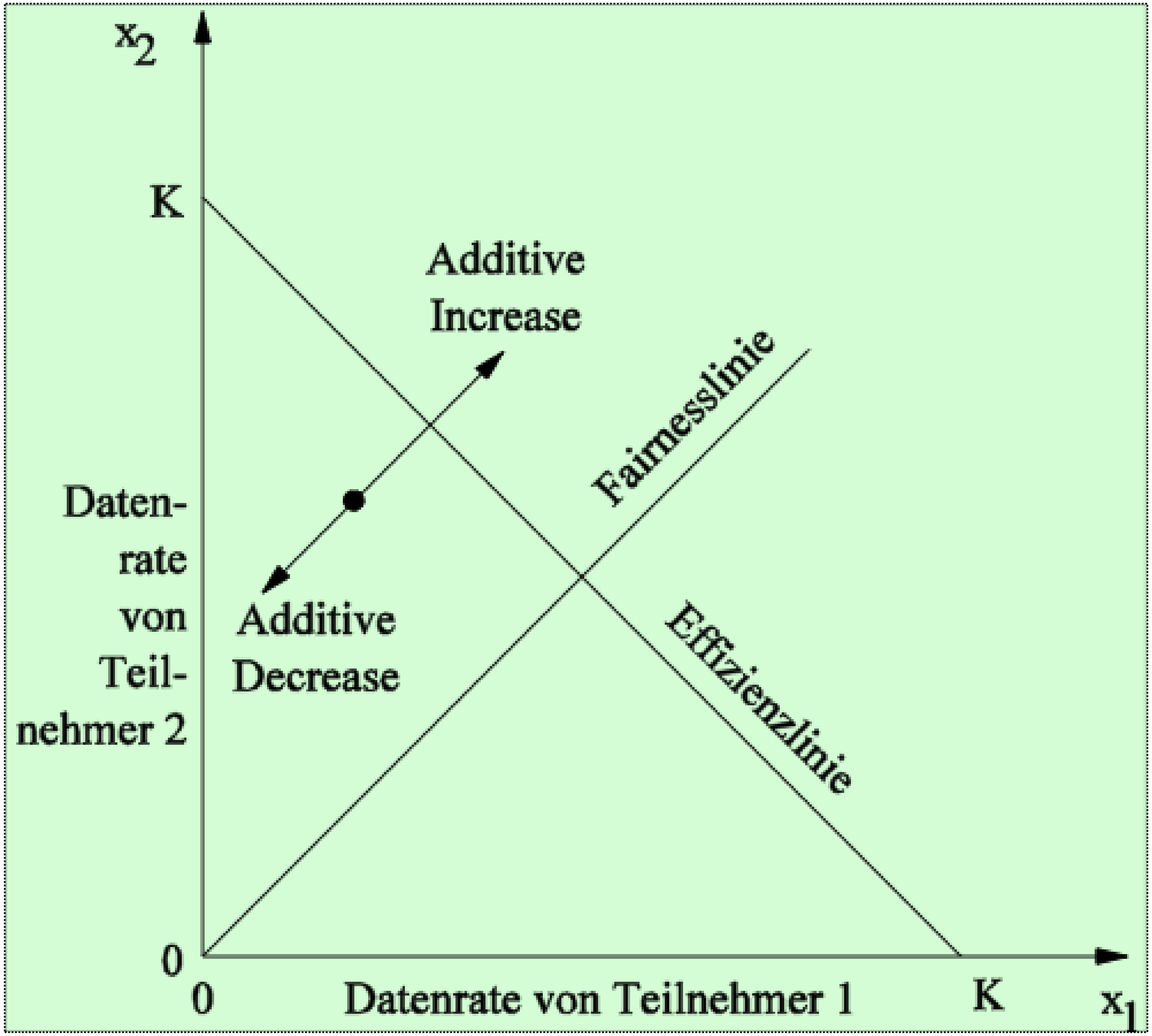


Vektordarstellung (II)

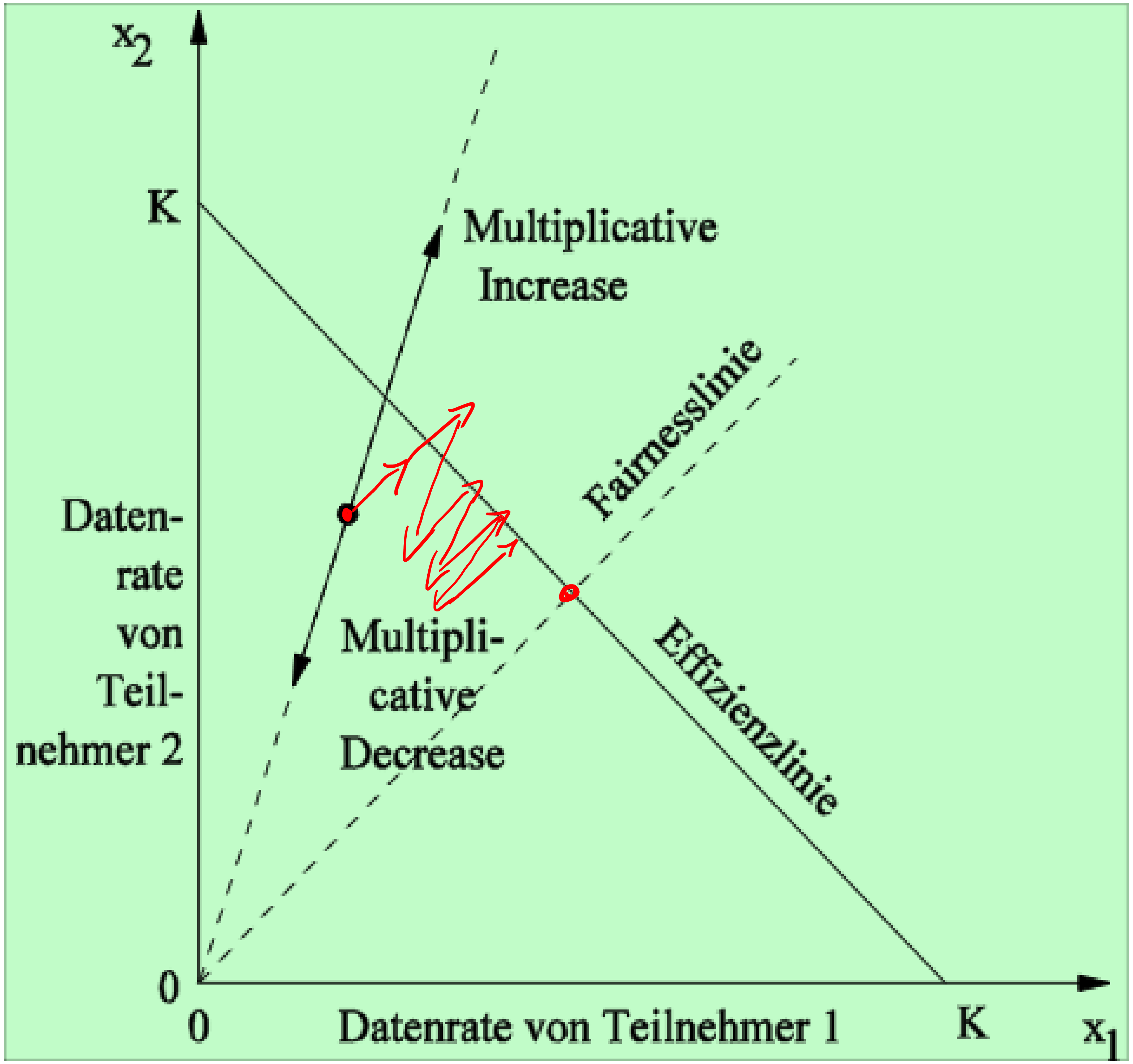
AIAD



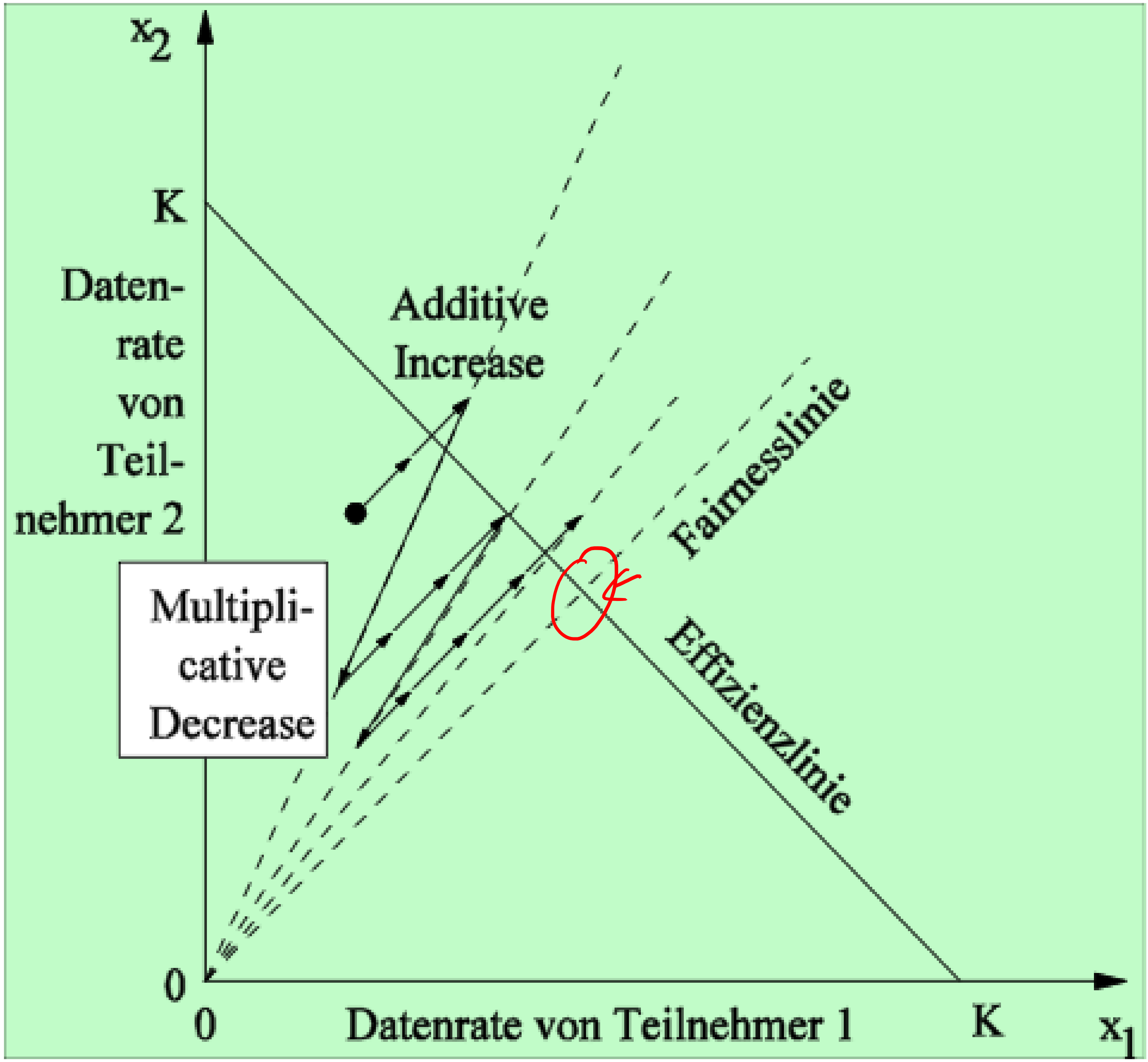
AIAD Additive Increase/ Additive Decrease

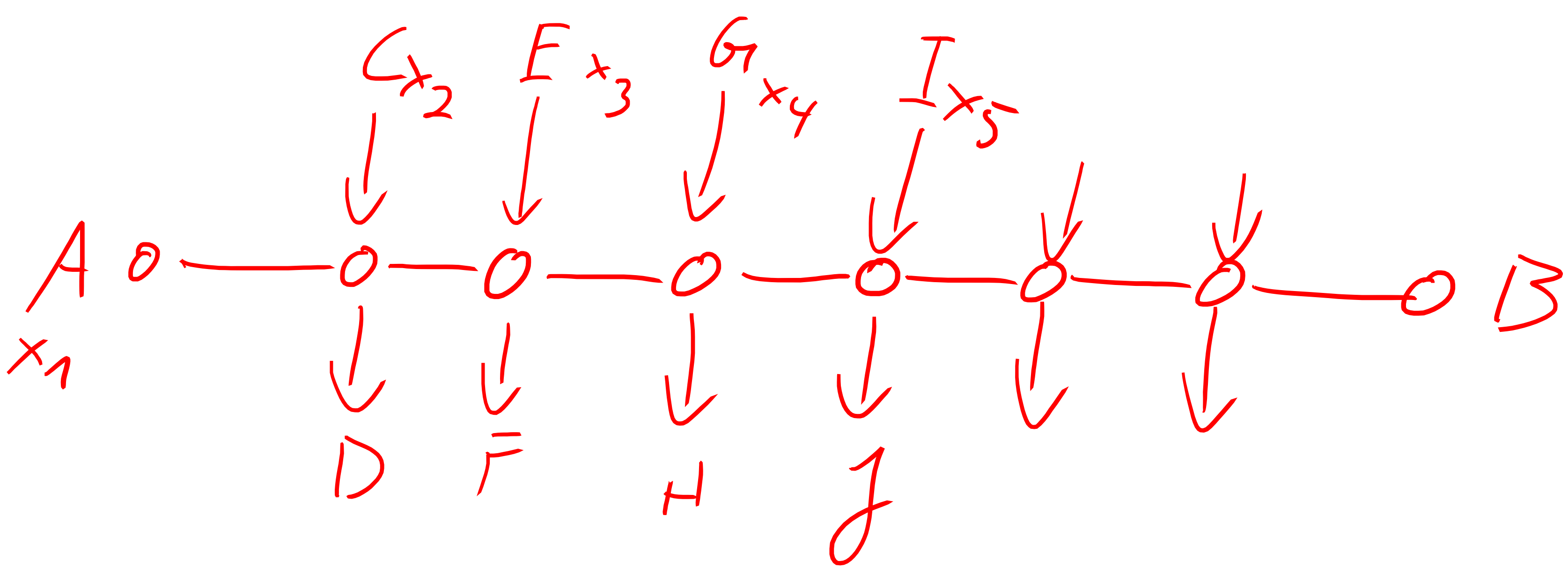


MIMD: Multiplicative Incr./ Multiplicative Decrease



AIMD: Additively Increase/ Multiplicatively Decrease





$K = 10$

x_1	x_2
1	8
2	9 4
1	4
2	5
1	6

x_1	x_3
1	3
2	4
1	5
2	6
1	7

x_1	x_4
1	5
2	6
1	7
2	8
1	9

x_1	x_5
1	6
2	7
1	8
2	9 5
1	4