

# Linearisierbarkeit als Korrektheitseigenschaft für nicht-blockierende Datenstrukturen

Jakob Krainz

[jakob@hawo-net.de](mailto:jakob@hawo-net.de)

AKSS 2009, Vortrag am 14. 5.

# Gliederung

- 1 Einleitung
- 2 Definition der Linearisierbarkeit
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Einleitung

Generelles Themengebiet:

Datenstrukturen



in einer Multiprozessorumgebung

# Zielsetzung

Datenstrukturen als Hilfsmittel bei

- Entwurf
- Implementierung
- Nachweis der Korrektheit

Vorteile:

- hohes Abstraktionsniveau
- Abkapselung von Code

Vorgehensweise im sequentiellen Fall:

- Festlegung der zulässigen Operationen auf der Datenstruktur (“Interface”)
- Beschreibung des Verhaltens der Operationen (z. B. durch Axiome)
- Implementierung des Verhaltens

## Problem: Nebenläufige Programme?

- Abstrakte Beschreibung nur für sequentiellen Fall gültig
- Zustände / Werte eines abstrakten Datentyps nur zwischen Zugriffen definiert, nicht während eines Zugriffs.
- Wirkung von nebenläufigen Zugriffen nicht definiert.

## Was ist zu tun?

- Sinnvoll: Weiterverwendung des Altbekanntes

abstrakte Spezifikation  
eines Datentyps für  
sequentielle Programme

→

Spezifikation für  
nebenläufige  
Programme.

- Zusätzliche Eigenschaften oder Bedingungen erforderlich...

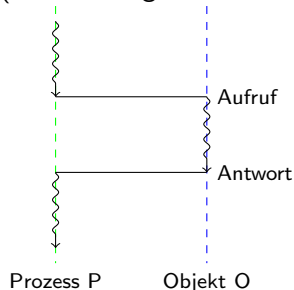
# Linearisierbarkeit als Eigenschaft

Gegeben:

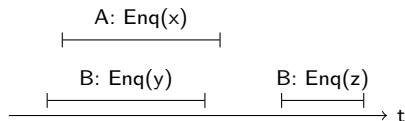
- Eine abstrakte (sequentielle) Spezifikation des Datentyps
- Die Linearisierbarkeit der Implementierung

Damit ist die Implementierung in nebenläufigen Programmen verwendbar.

Ein Zugriff auf ein Objekt:  
(z. B. Einfügen in eine Liste)



Mehrere nebenläufige Zugriffe:



Nach diesen 3 Aufrufen:

- x, y und z in der Liste
- z hinter x und y

# Anforderungen an einen Datentyp

- 2 nebenläufige Zugriffe sollen effektiv hintereinander passieren
- 2 sequentielle Zugriffe sollen in ihrer Reihenfolge passieren



- 1 Einleitung
- 2 Definition der Linearisierbarkeit**
  - Informelle Definition
  - Formale Definition
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Informelle Definition der Linearisierbarkeit

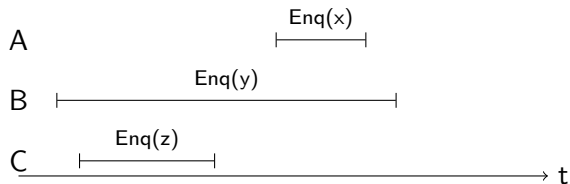
## Linearisierbarkeit

Eine Datenstruktur ist linearisierbar, wenn:

- eine Ablauffolge von z. T. nebenläufigen Zugriffen umgeformt werden kann in eine Ablauffolge von sequentiellen Zugriffen
- Zugriffe, die nicht nebenläufig zueinander passieren, ihre Reihenfolge behalten.

# Beispiel

Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:

# Beispiel

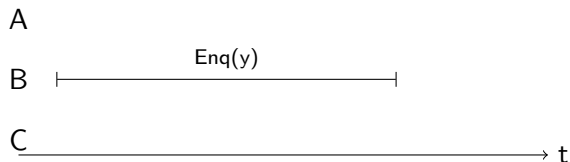
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:

# Beispiel

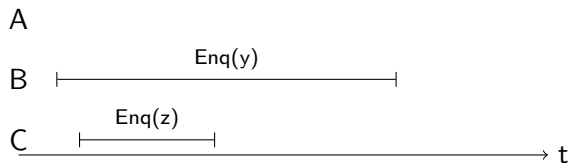
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:

# Beispiel

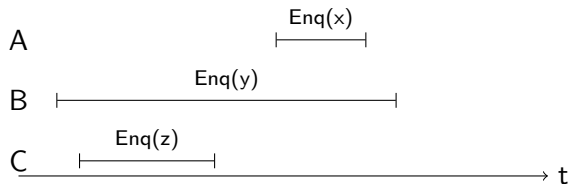
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:

# Beispiel

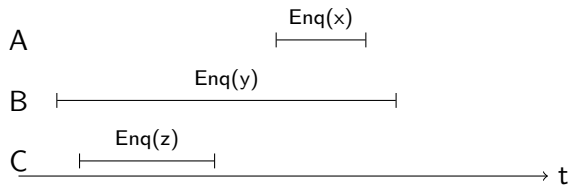
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:

# Beispiel

Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,



# Beispiel

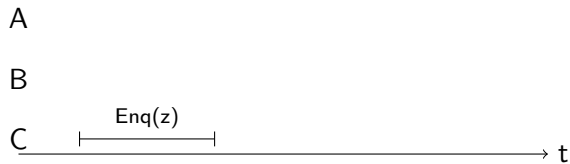
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,

# Beispiel

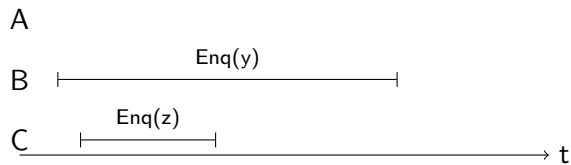
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,

# Beispiel

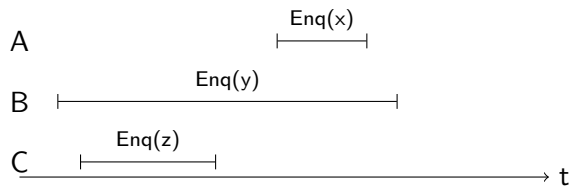
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,

# Beispiel

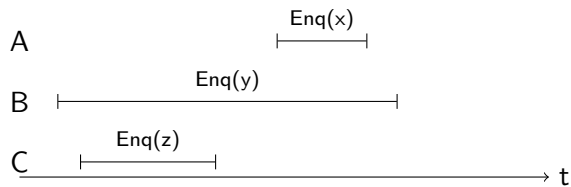
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,

# Beispiel

Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,

# Beispiel

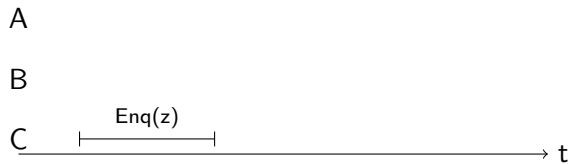
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,

# Beispiel

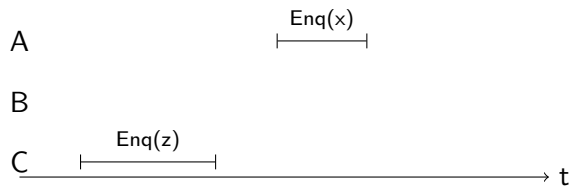
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,

# Beispiel

Die folgende Ablauffolge ist mehrfach linearisierbar:

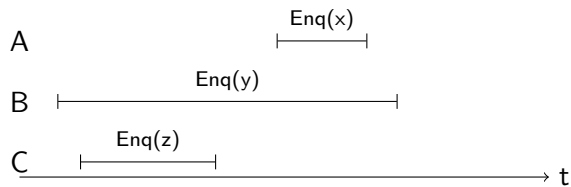


Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,



# Beispiel

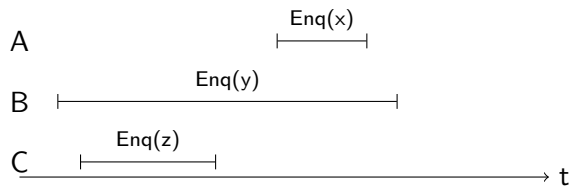
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,

# Beispiel

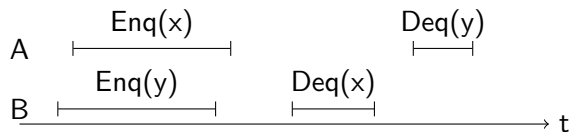
Die folgende Ablauffolge ist mehrfach linearisierbar:



Die dabei entstehenden Listen:  $[y, z, x]$ ,  $[z, y, x]$ ,  $[z, x, y]$

## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



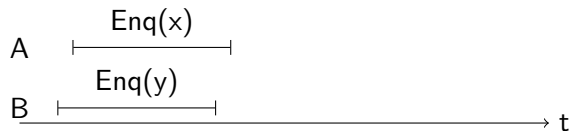
## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



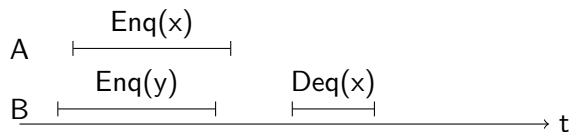
## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



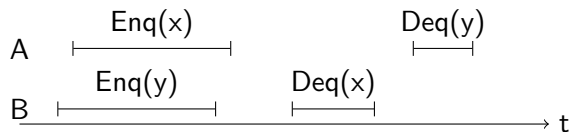
## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



## Weiteres Beispiel

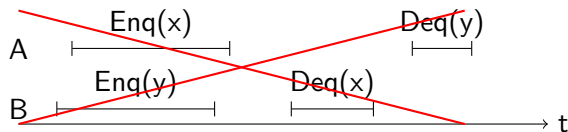
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:





## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da  $x$  vor  $y$  entfernt wird, muss  $x$  vor  $y$  eingefügt worden sein

## Weiteres Beispiel

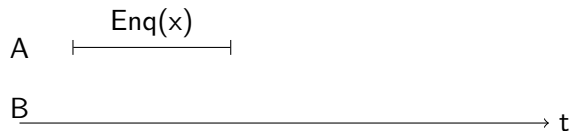
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:

A  
B \_\_\_\_\_ → t

Da x vor y entfernt wird, muss x vor y eingefügt worden sein

## Weiteres Beispiel

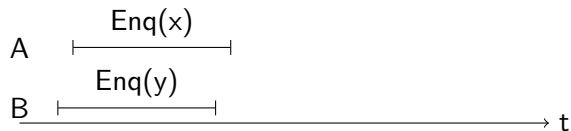
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da x vor y entfernt wird, muss x vor y eingefügt worden sein

## Weiteres Beispiel

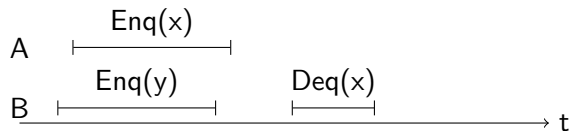
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da  $x$  vor  $y$  entfernt wird, muss  $x$  vor  $y$  eingefügt worden sein

## Weiteres Beispiel

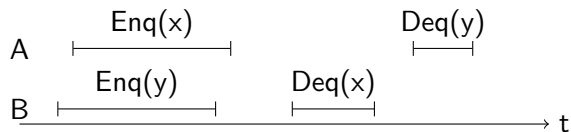
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da  $x$  vor  $y$  entfernt wird, muss  $x$  vor  $y$  eingefügt worden sein

## Weiteres Beispiel

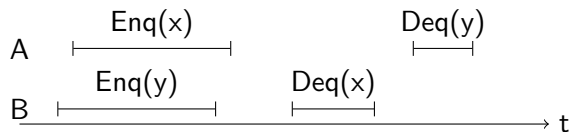
Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da  $x$  vor  $y$  entfernt wird, muss  $x$  vor  $y$  eingefügt worden sein

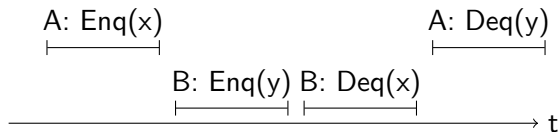
## Weiteres Beispiel

Die folgende Ablauffolge ist nur auf eine Art linearisierbar:



Da  $x$  vor  $y$  entfernt wird, muss  $x$  vor  $y$  eingefügt worden sein

Diese Ausführungsreihenfolge ist die einzig Mögliche:



# Linearisierungspunkt

Eine linearisierbare Datenstruktur bietet die Illusion, dass die Zugriffe zwar beliebig lange dauern, ihre Wirkung allerdings atomar während des Zugriffs eintritt.

## Definition Linearisierungspunkt

Ein Linearisierungspunkt eines Zugriffs ist ein Zeitpunkt während dieses Zugriffs, zu dem der Zugriff eine atomare Zustandstransition der Datenstruktur bewirkt.

## Alternative Definition der Linearisierbarkeit

Eine Ablauffolge ist linearisierbar, wenn jeder Zugriff einen Linearisierungspunkt besitzt.



- 1 Einleitung
- 2 **Definition der Linearisierbarkeit**
  - Informelle Definition
  - Formale Definition
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Grundbegriffe

## Nebenläufig

Zwei Zugriffe einer Ablauffolge sind *nebenläufig*, wenn sich die zwei Zeitintervalle der Zugriffe überlappen.

Zwei nicht nebenläufige Zugriffe sind *zueinander sequentiell*.

## Zeitliche Ordnung von Zugriffen

wenn  $Z_1$  und  $Z_2$  nicht nebenläufig sind und  $Z_1$  vor  $Z_2$  passiert, schreibt man  $Z_1 <_H Z_2$

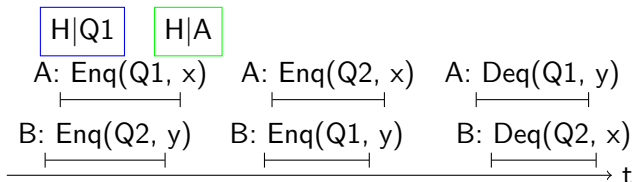
$<_H$  definiert eine partielle Ordnung auf den Zugriffen

## Teilablauffolgen

Zu einer Ablauffolge  $H$ , einem Prozess  $P$  und einem Objekt  $O$ :

$H|P$  ist die Teilablauffolge der Zugriffe von  $P$

$H|O$  ist die Teilablauffolge der Zugriffe auf  $O$

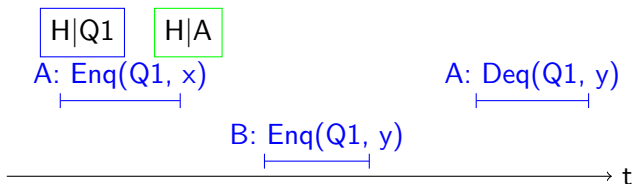


## Teilablauffolgen

Zu einer Ablauffolge  $H$ , einem Prozess  $P$  und einem Objekt  $O$ :

$H|P$  ist die Teilablauffolge der Zugriffe von  $P$

$H|O$  ist die Teilablauffolge der Zugriffe auf  $O$

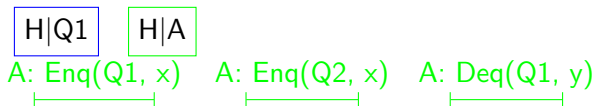


## Teilablauffolgen

Zu einer Ablauffolge  $H$ , einem Prozess  $P$  und einem Objekt  $O$ :

$H|P$  ist die Teilablauffolge der Zugriffe von  $P$

$H|O$  ist die Teilablauffolge der Zugriffe auf  $O$



\_\_\_\_\_ → t

# Unvollständige Zugriffe

In einer Ablauffolge kann es Zugriffe geben, deren Antwortereignis noch nicht passiert ist; diese Zugriffe sind *unvollständig*.

$\text{complete}(H)$  ist die Teilablauffolge von  $H$ , die aus allen vollständigen Zugriffen besteht

# Äquivalenz von Ablauffolgen

Zwei Ablauffolgen  $H$  und  $H'$  über der gleichen Menge von Prozessen und Objekten sind *äquivalent* wenn für alle beteiligten Prozesse  $P$  gilt:  $H|P$  und  $H'|P$  bestehen aus Zugriffen, die

- in der gleichen Reihenfolge
- auf die gleichen Objekte
- mit den gleichen Parametern und Ergebnissen

stattfinden.

Der zeitliche Abstand zwischen zwei Zugriffen oder die Dauer der einzelnen Zugriffe darf sich ändern.

## Linearisierbarkeit

Eine Ablauffolge  $H$  ist linearisierbar, wenn zwei weitere Ablauffolgen  $H'$  und  $S$  existieren, die folgenden Bedingungen genügen:

- $H'$  entsteht, indem in  $H$  einige unvollständige Zugriffe vervollständigt werden.
- $S$  ist zu  $\text{complete}(H')$  äquivalent.
- $S$  ist sequentiell.
- Die Spezifikation aller Objekte erlaubt  $S$ .
- Für alle Zugriffe  $Z_1$  und  $Z_2$  in  $H$  mit:  $Z_1 <_H Z_2$  gilt  $Z_1 <_S Z_2$ .



Um eine Ablauffolge zu linearisieren:

- Für nebenläufige Zugriffe eine sequentielle Reihenfolge wählen
- Reihenfolge nichtnebenläufiger Zugriffe gleich belassen
- Ergebnis darf die sequentielle Spezifikation der beteiligten Objekte nicht verletzen

- 1 Einleitung
- 2 Definition der Linearisierbarkeit
- 3 Anwendung der Linearisierbarkeit**
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Allgemeine Folgerungen

- Eine linearisierbare Datenstruktur ist in nebenläufigen Programmen verwendbar.
- “Bösartiges” Verhalten einer einzigen Datenstruktur wird ausgeschlossen
- Wettlaufsituationen im restlichen Programm werden nicht ausgeschlossen:
  - ▶ Linearisierungen können mehrdeutig sein
  - ▶ Abhängigkeiten müssen ggf. explizit dargestellt werden

## Beispiel: Verhalten einer FIFO–Warteschlange

Aus der Linearisierbarkeit einer Warteschlange lässt sich Folgendes herleiten:

- Wenn  $x$  vor  $y$  eingefügt wird, so wird es nicht nach  $y$  entfernt.
- Wenn  $x$  vor  $y$  eingefügt wird und  $y$  entnommen wurde, so wurde  $x$  vor  $y$  entnommen oder die zwei Entnahmen sind nebenläufig.
- Wenn  $x$  entnommen wird, so wurde es eingefügt; das Einfügen war vor der Entnahme oder nebenläufig zur ihr.

- 1 Einleitung
- 2 Definition der Linearisierbarkeit
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit**
  - Blockierungsfreiheit
  - Lokalität
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Blockierungsfreiheit

- Linearisierbare Datenstrukturen erzwingen keine Blockade von Operationen.
- Zu jeder Ablauffolge mit einem unvollständigen Zugriff existiert eine Ablauffolge in der dieser Zugriff vollständig ist (wenn der Zugriff durchführbar ist).
- Ein unvollständiger Zugriff muss damit nicht auf Zugriffe warten, die noch nicht stattgefunden haben.
- Ein Objekt muss nicht zwischen zwei Zugriffen für totale Operationen blockiert sein.

# Anmerkungen

- Die Implementierung *kann* Zugriffe verzögern und dennoch linearisierbar bleiben - sie *muss* nicht.
- Undurchführbare Operationen dürfen blockieren – die Blockierungsfreiheit gilt nur für durchführbare Operationen
- Es ist nicht spezifiziert, auf wieviele nebenläufige Operationen eine Operation warten muss – die Gefahr einer Verhungering des Prozesses ist immer noch gegeben.

# Lokalität

Wie weist man die Linearisierbarkeit eines Systems von Objekten nach?

- Betrachtung des Gesamtsystems nicht nötig
- Nachweis der Linearisierbarkeit der einzelnen Objekte reicht aus für die Linearisierbarkeit des gesamten Systems
- Enorme Verringerung des Beweisaufwands



- 1 Einleitung
- 2 Definition der Linearisierbarkeit
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit**
  - Atomare Zustandsänderungen
  - Atomarität durch Locking
  - Allgemeine Beweismethode
- 6 Vergleich mit anderen Korrektheitseigenschaften
- 7 Fazit

# Atomare Zustandsänderungen

Einfachste Möglichkeit:

Jede Operation hat einen Linearisierungspunkt

⇒ Linearisierbarkeit

Wie?

- atomare Zustandsänderungen
- implementierbar durch atomare Operationen (CAS, LL/SC etc.)
- Beispiel: Mutex

# Beispiel

## Einfaches Beispiel: Mutex

- Zustände des Mutex:
  - ▶ gesperrt: "l"
  - ▶ frei: "u"
- drei Operationen auf dem Mutex:
  - ▶ try\_lock:  
return CAS(m, "u", "l")
  - ▶ unlock  
CAS(m, "l", "u")
  - ▶ lock  
while not try\_lock(m) do nothing

- Die Implementierung ist linearisierbar:  
Linearisierungspunkt  $\leftrightarrow$  atomare Instruktion
- lock blockiert bei “l”, trotzdem nichtblockierend:  
lock ist keine totale Operation
- Anwendbar bei einfachen Datenstrukturen; bei komplexeren Datenstrukturen gibt z. T. keine hinreichenden atomaren Instruktionen...

# Atomarität durch Sperren

- Pro Objekt ein Mutex
- Zu Beginn des Zugriffs: Mutex sperren, falls schon gesperrt: warten bis Mutex frei
- Am Ende des Zugriffs: Mutex freigeben
- Triviale Implementierung
- So nur für totale Operationen geeignet
- wenig performant
- linearisierbar:  
Linearisierungspunkt  $\iff$  Mutexfreigabe

# Allgemeine Beweismethode

Ziel: Linearisierbarkeit ohne Sperren

Problem: atomare Instruktionen nicht ausreichend

- bieten die Möglichkeit, wenige Bytes atomar zu manipulieren
- nicht ausreichend bei größeren Datenstrukturen und verteilten Änderungen

Allgemeine Beweismethode nötig!

## Lösungsansatz: Invariante

- Bedingung, die die Datenstruktur zu jedem Zeitpunkt erfüllt
- Wird zu Beginn verifiziert
- Zerlegung der Operationen in atomare Instruktionen
- Für jede atomare Instruktion zu zeigen:  
Invariante gilt vorher  $\Rightarrow$  Invariante gilt nachher
- Mit der Invariante das korrekte Funktionieren des Objekts beweisen:
  - ▶ im sequentiellen Fall:  
ein interner Zustand = ein abstrakter Zustand
  - ▶ hier:  
ein interner Zustand = eine Menge von abstrakten Zuständen
  - ▶ bei einer Instruktion ändert sich diese Menge
  - ▶ korrektes Funktionieren  $\iff$  Menge ändert sich passend zur Spezifikation

- 1 Einleitung
- 2 Definition der Linearisierbarkeit
- 3 Anwendung der Linearisierbarkeit
- 4 Eigenschaften der Linearisierbarkeit
- 5 Nachweis der Linearisierbarkeit
- 6 Vergleich mit anderen Korrektheitseigenschaften**
  - Serialisierbarkeit
  - Sequentielle Konsistenz
- 7 Fazit



# Vergleich mit anderen Korrektheitseigenschaften

Zur besseren Einschätzung der Linearisierbarkeit:

- Vergleich mit Serialisierbarkeit
- Vergleich mit sequentieller Konsistenz

# Serialisierbarkeit

Vorgehensweise:

- Organisation der Zugriffe in Transaktionen
- Prozess
  - ▶ startet Transaktion
  - ▶ macht einige Zugriffe (lesend oder schreibend)
  - ▶ beendet Transaktion
- Zugriffe nehmen ihre Wirkung atomar mit der Beendigung der Transaktion an
- Transaktion ist atomarer Zustandsübergang des gesamten Systems

## Definition Serialisierbarkeit

Eine Ablauffolge ist serialisierbar, wenn sie äquivalent ist zu einer Ablauffolge, in der die Transaktionen sequentiell passieren.

# Serialisierbarkeit ist blockierend

2 Transaktionen:

1.:  $X := Y + 1$

Implementierung:

- Lese Y nach tmp
- Berechne  $tmp = tmp + 1$
- Schreibe tmp nach X

2.:  $Y := X + 1$

Implementierung:

- Lese X nach tmp
- Berechne  $tmp = tmp + 1$
- Schreibe tmp nach Y

Problem:

- 1. Transaktion startet mit "Lese Y nach tmp"
- 2. Transaktion startet mit "Lese X nach tmp"
- 1. Transaktion schreibt tmp+1 nach X
- 2. Transaktion schreibt tmp+1 nach Y

→ Falsche Werte in X und Y!

## 2 Lösungsansätze:

- Vermeidung von kollidierenden Transaktionen
  - ▶ Feststellen welche Transaktionen kollidieren könnten
  - ▶ kollidierende Transaktionen werden verzögert
- Auflösung von kollidierenden Transaktionen
  - ▶ Feststellung von Kollisionen
  - ▶ kollidierende Transaktionen werden abgebrochen
  - ▶ ggf. Neustart der abgebrochenen Transaktionen

In jedem Fall: Während einer Transaktion können andere Prozesse nicht auf beteiligte Objekte zugreifen

→ Serialisierbarkeit ist blockierend

### Vorteile der Serialisierbarkeit:

- Konzeptionell einfach
- Transaktionen als Modellierungsmittel erwünscht
- Konsistenz des Systems einfach zu verifizieren

### Nachteile der Serialisierbarkeit:

- Während einer Transaktion sind andere Transaktionen blockiert
- alternativ müssen kollidierende Transaktionen abgebrochen werden
- weniger Nebenläufigkeit möglich

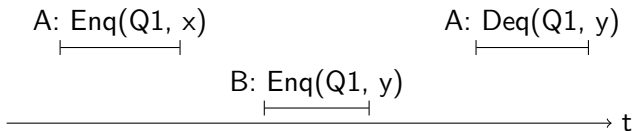
# Sequentielle Konsistenz

## Definition Sequentielle Konsistenz

Eine Ablauffolge ist sequentiell konsistent, wenn sie äquivalent ist zu einer sequentiellen Ablauffolge.

Sequentielle Konsistenz ist schwächer als Linearisierbarkeit:

- die Reihenfolge nichtnebenläufiger Zugriffe von unterschiedlichen Prozessen muss *nicht* erhalten werden
- damit leichter zu beweisen
- jede linearisierbare Ablauffolge ist auch sequentiell konsistent
- nicht jede sequentiell konsistente Ablauffolge ist linearisierbar



Die Teilablauffolge  $H|Q1$  ist sequentiell konsistent, aber nicht linearisierbar.

A: Enq(Q2, x)

B: Enq(Q2, y)

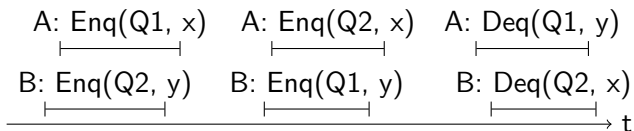
B: Deq(Q2, x)



Die Teilablauffolge  $H|Q1$  ist sequentiell konsistent, aber nicht linearisierbar.

Die Teilablauffolge  $H|Q2$  ist ebenfalls sequentiell konsistent, aber nicht linearisierbar.

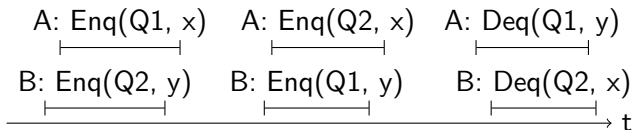




Die Teilablauffolge  $H|Q1$  ist sequentiell konsistent, aber nicht linearisierbar.

Die Teilablauffolge  $H|Q2$  ist ebenfalls sequentiell konsistent, aber nicht linearisierbar.

Die gesamte Ablauffolge  $H$  ist nicht sequentiell konsistent.



Die Teilablauffolge  $H|Q1$  ist sequentiell konsistent, aber nicht linearisierbar.

Die Teilablauffolge  $H|Q2$  ist ebenfalls sequentiell konsistent, aber nicht linearisierbar.

Die gesamte Ablauffolge  $H$  ist nicht sequentiell konsistent.  
 Sequentielle Konsistenz ist nichtlokal

## Vorteile der sequentiellen Konsistenz:

- leichter zu beweisen
- leichter zu implementieren
- niedrigere Laufzeitkosten

## Nachteile:

- Nichtlokal, damit höherer Aufwand bei Verifikation des korrekten Verhaltens.

# Fazit

- Linearisierbarkeit ist eine Bedingung für die Verwendbarkeit von Objekten in Multiprozessorsystemen
- Lokalität → skalierbar auf große Systeme
- Blockierungsfreiheit, damit:
  - ▶ höhere Parallelität, also höhere Leistung
  - ▶ potentiell gutes Echtzeitverhalten
- Nachteile:
  - ▶ Aussagen über den Zustand des gesamten Systems schwer zu treffen
  - ▶ In echten verteilten Systemen verspricht sequentielle Konsistenz höhere Leistung