



# Wartefreie Synchronisation

---

Martha Willhaug

[Martha.Willhaug@informatik.stud.uni-erlangen.de](mailto:Martha.Willhaug@informatik.stud.uni-erlangen.de)



# Gliederung

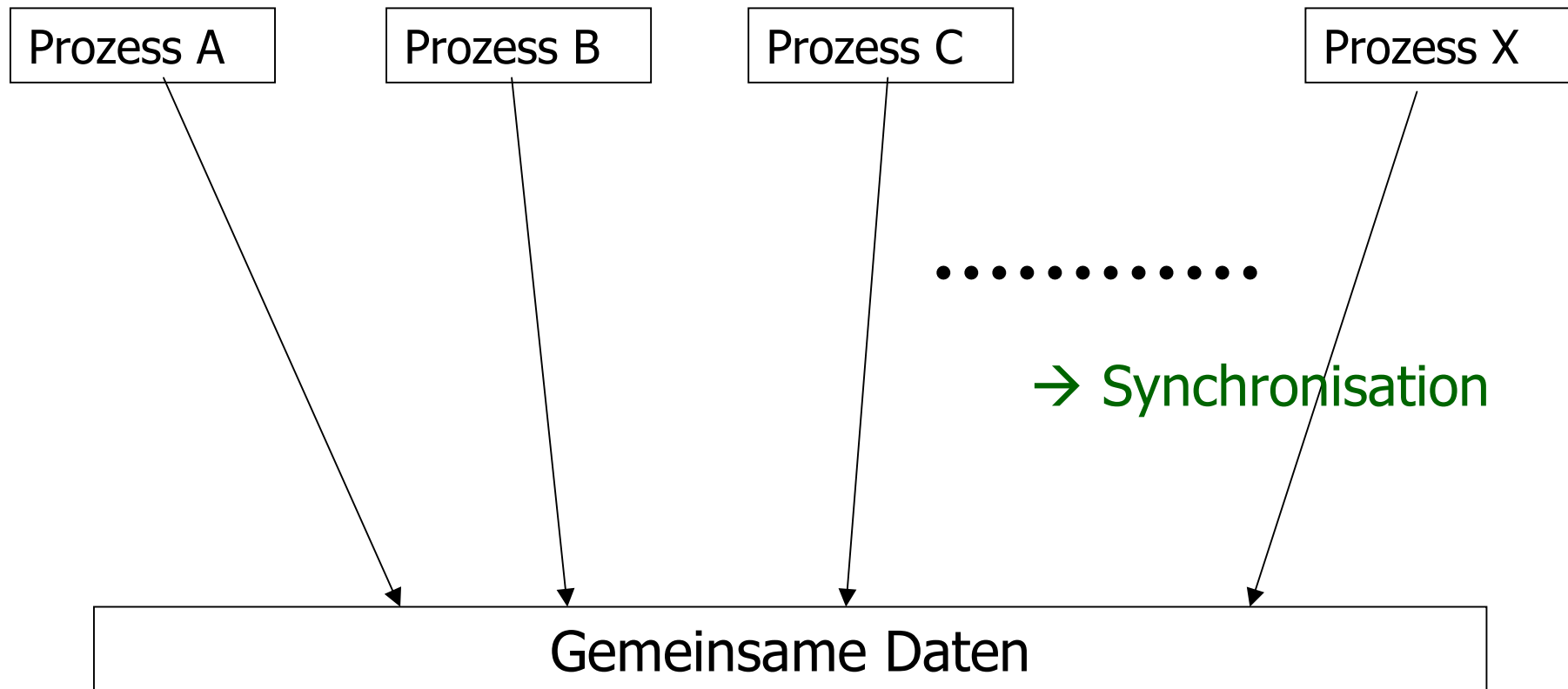
---

1. Einleitung
2. Definition
3. Unmöglichkeitsresultate
4. Ein universelles wartefreies Konstrukt
5. Fazit



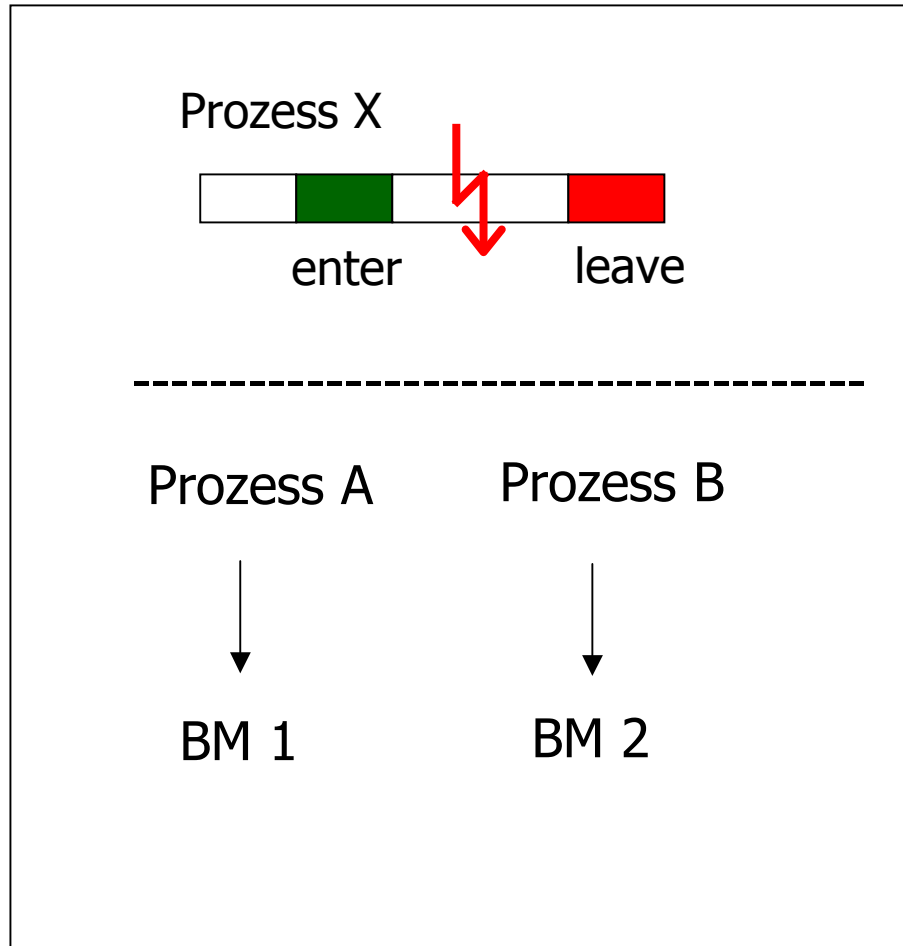
# Synchronisation

---

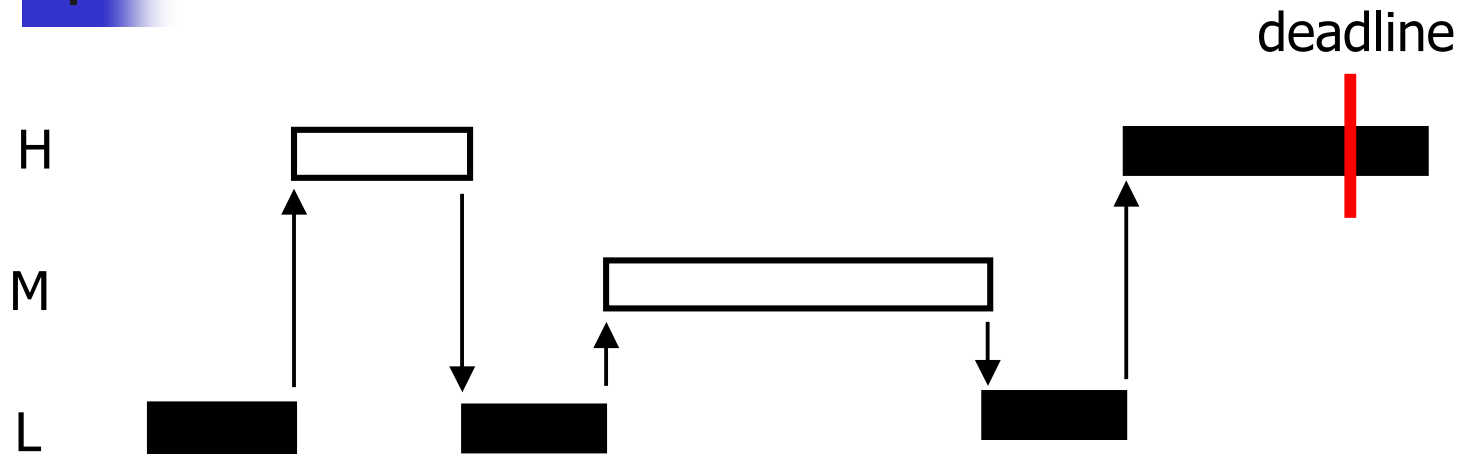


# Blockierende Verfahren

- Kritischer Abschnitt:  
Verklemmung (deadlock, livelock)
- Philosophenproblem:  
Verhungering
- Prioritätenumkehr



# Prioritätenumkehr



Abhilfe:

H gibt L seine Priorität ab

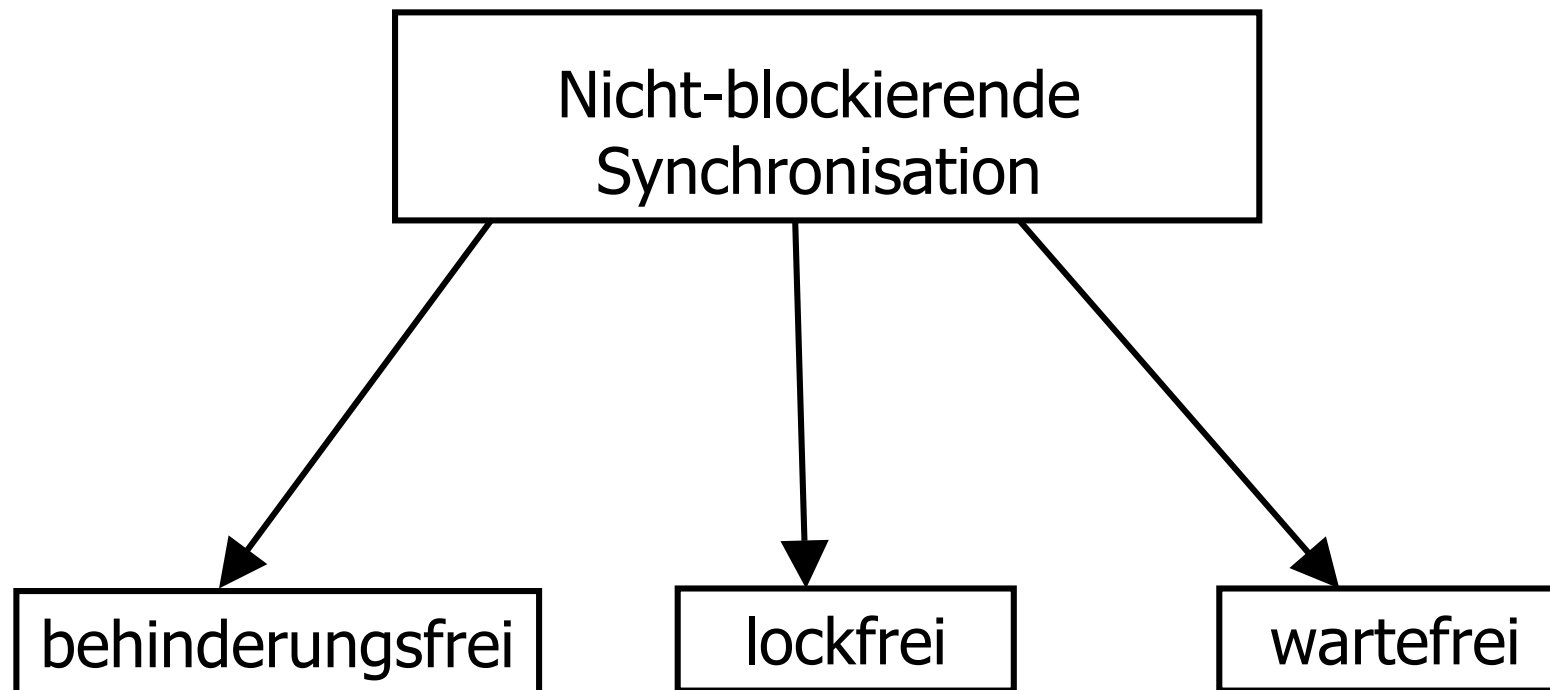
L kann mit höhere Priorität nicht verdrängt werden

→ wird schneller fertig

→ H kommt schneller wieder dran



# Nicht-blockierende Verfahren





# Nicht-blockierende Verfahren

---

- **Behinderungsfrei:**

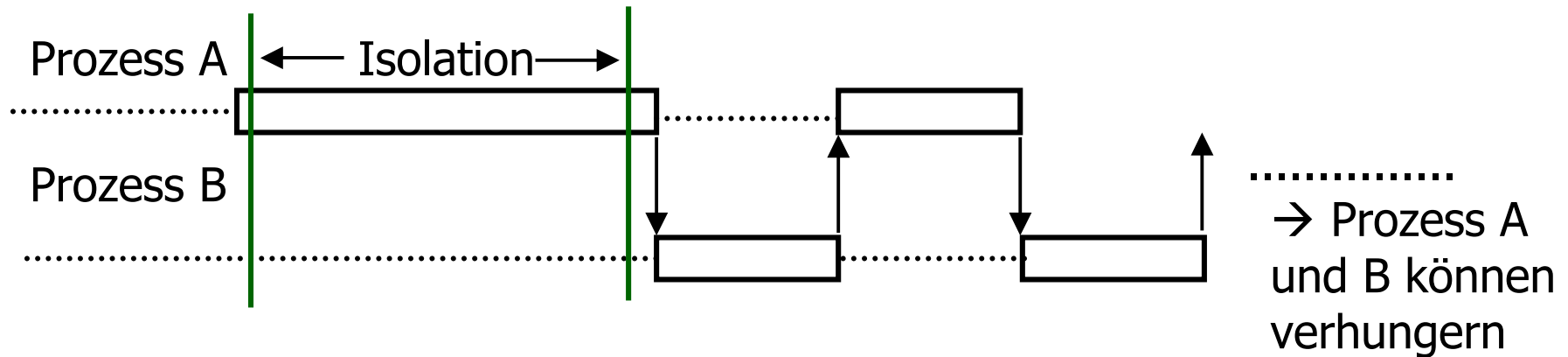
- Ein Prozess in „Isolation“ beendet seine Operation
- Gegenseitige Beeinflussung möglich
  - Alle Prozesse können verhungern

- **Lockfrei:**

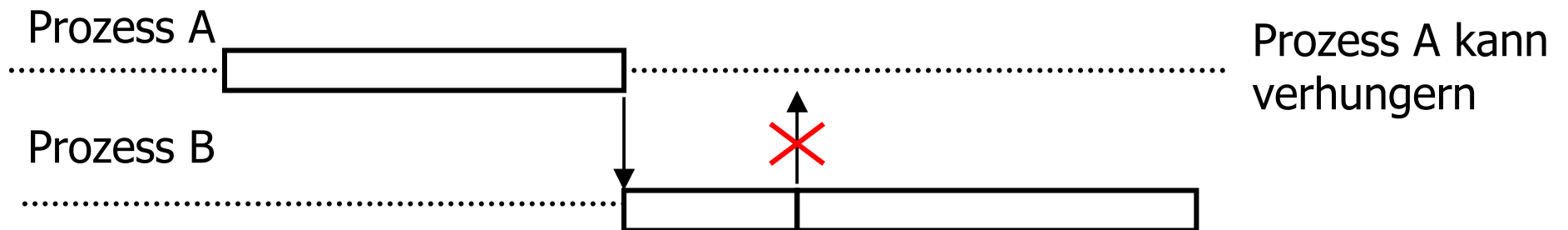
- Garantiert globalen Fortschritt des Systems
- Einzelne Prozesse können verhungern

# Nicht-blockierende Verfahren

## Behinderungsfrei:



## Lockfrei:





## Definition: wartefreie Synchronisation

---

- Jeder Prozess beendet seine Operation in einer endlichen Anzahl an Schritten
- k-bedingt wartefrei:
  - Es existiert ein  $k > 0$
  - Prozesse beenden Operationen in k Schritten



# Konsensusnummer

---

- Größte Zahl  $n$  für die ein Objekt den  $n$ -Prozess-Konsensus löst
- Falls  $n$  nicht existiert  $\rightarrow$  unendlich
- Wartefreie Implementierung von  $n$  Prozessen
- Objekte auf Ebene  $n$ :
  - keine Nachbildung von Objekten aus Ebene  $n-1$
  - Nachbildung aus gleicher Ebene?



# Konsensusnummer

---

Consensus Number	Object
1	read/write registers
2	test&set, swap, fetch&add, queue, stack
⋮	⋮
$2n-2$	n-register assignment
⋮	⋮
$\infty$	memory-to-memory, move and swap, augmented queue, compare&swap



# Konsensus Protokoll

---

- Prozesse arbeiten auf gemeinsamen Objekten
- Starten mit einem Eingabewert
- Kommunizieren miteinander
- Einigen sich auf gemeinsamen Eingabewert
- Eigenschaften:
  - Konsistent
  - Wartefrei
  - Valid
- Zustände: univalent  $\longleftrightarrow$  bivalent



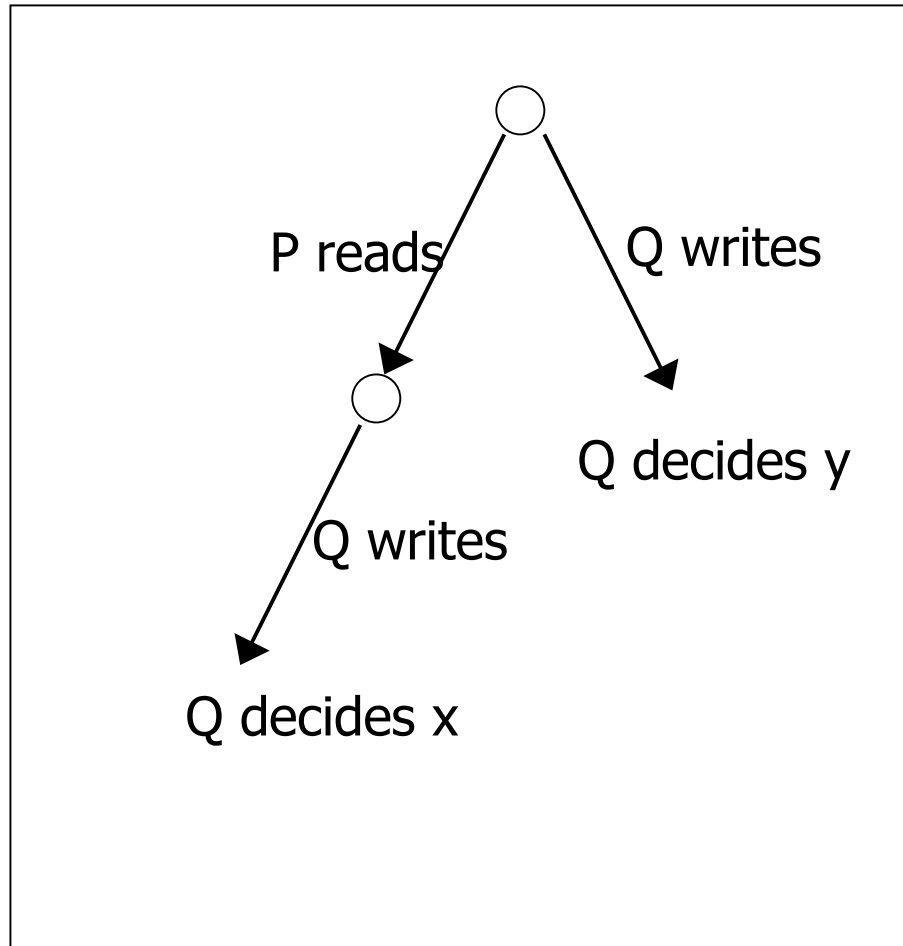
# Atomare Lese-/Schreibregister

---

- Haben Konsensusnummer 1
- Wartefreies Objekt für einen Prozess
- Können nicht Objekte mit höherer Konsensusnummer nachbilden
- Konsensusbildung:
  - Es darf kein bivalenter Zustand erreicht werden
  - Alle Prozesse sind an Konsensusbildung beteiligt

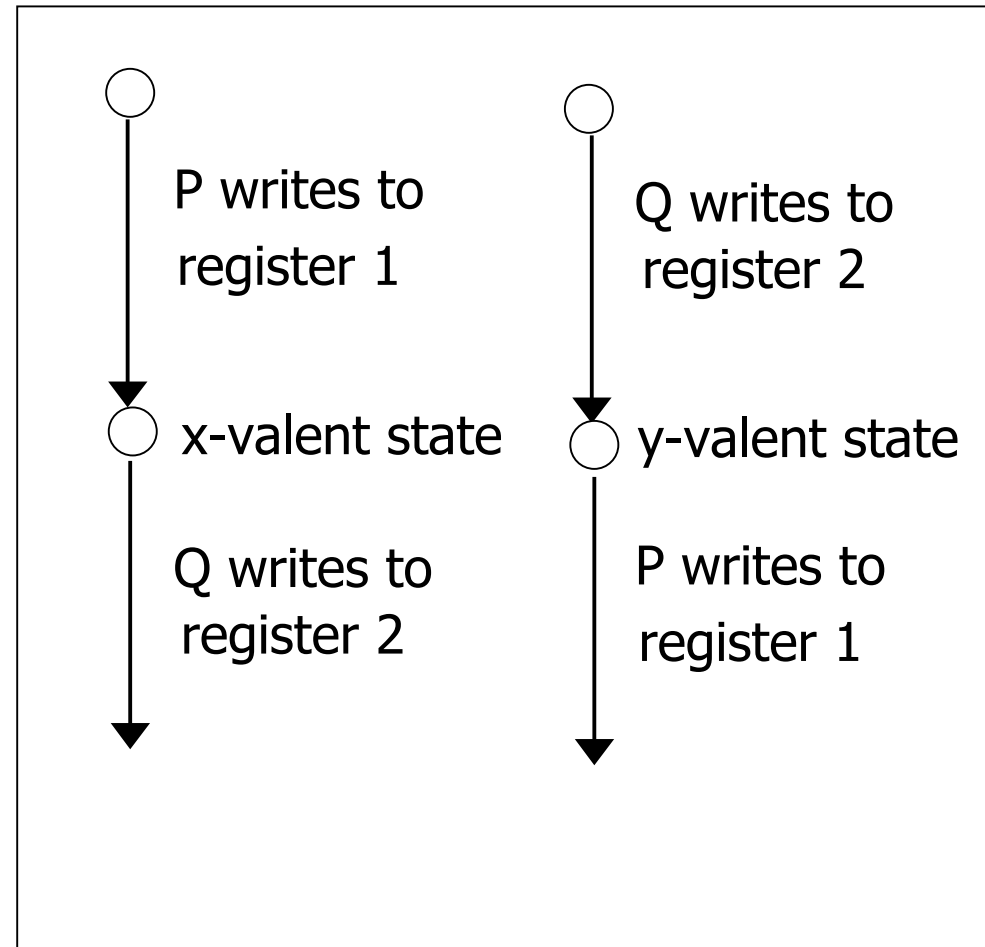
# Atomare Lese-/Schreibregister

- 1. Fall: P liest und Q schreibt in das Register
- P führt zu einem x-wertigen Zustand
- Q zu einem y-wertigen Zustand



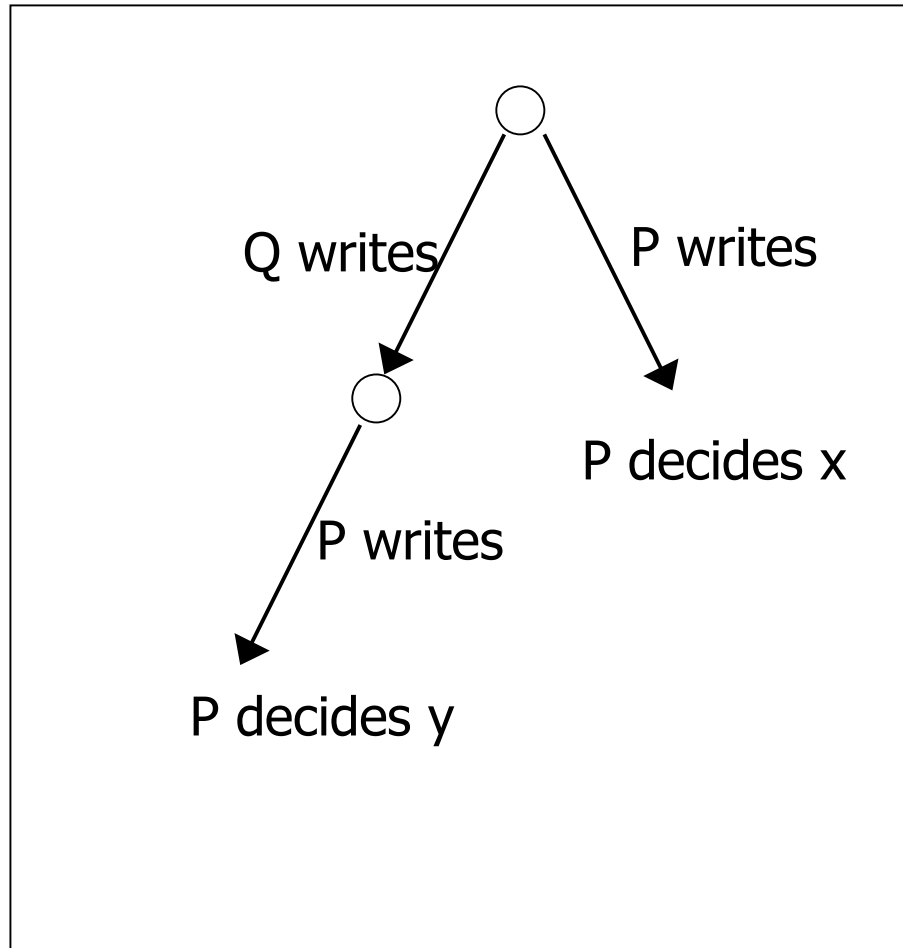
# Atomare Lese-/Schreibregister

- 2. Fall: P und Q schreiben in verschiedene Register



# Atomare Lese-/Schreibregister

- 3. Fall: P und Q schreiben in das gleiche Register





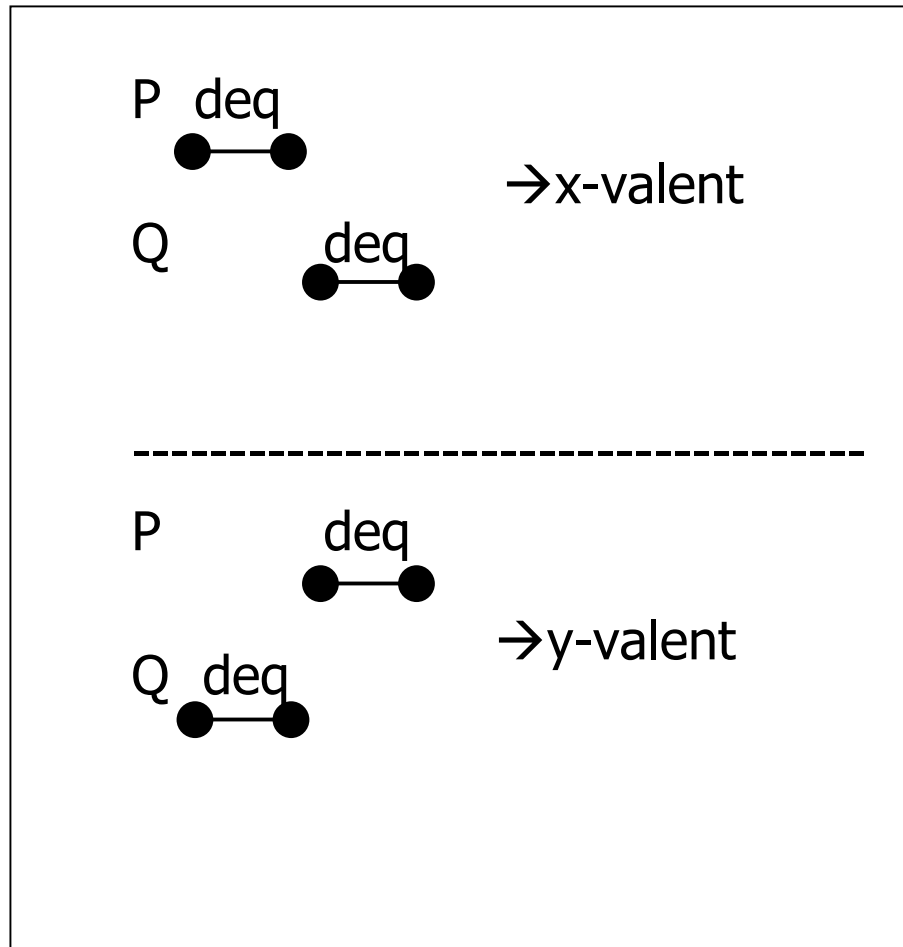
# FIFO-Queue

---

- Konsensusnummer 2
- Objekt: Konsensus wird gebildet
- Konsensusbildung
  - Alle Prozesse sind daran beteiligt
  - Prozesse müssen wissen was auf Queue passiert ist

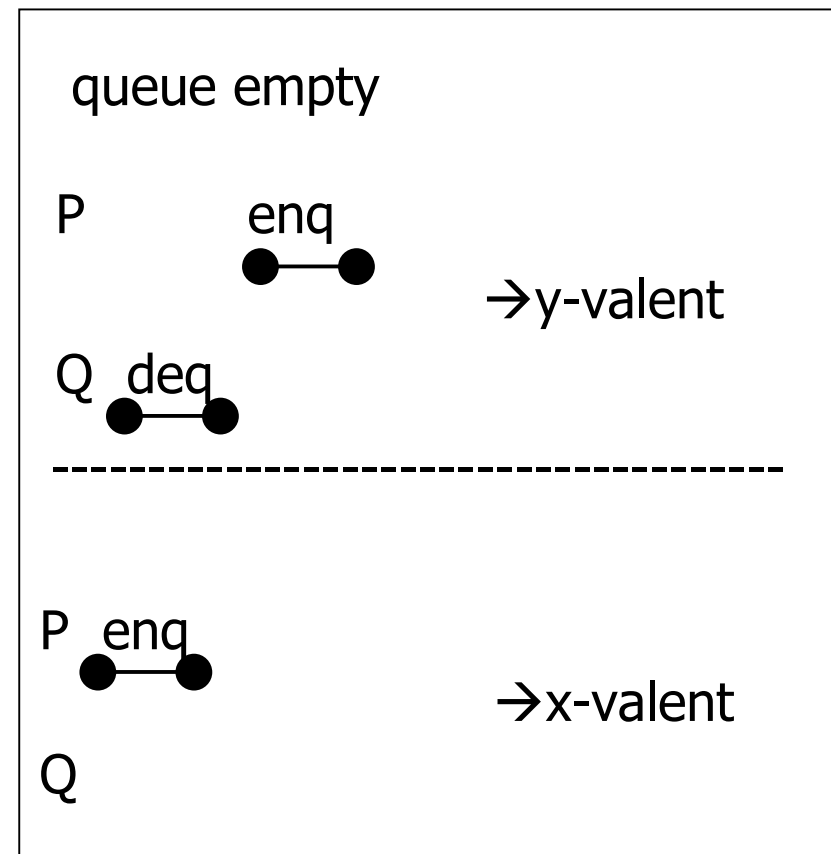
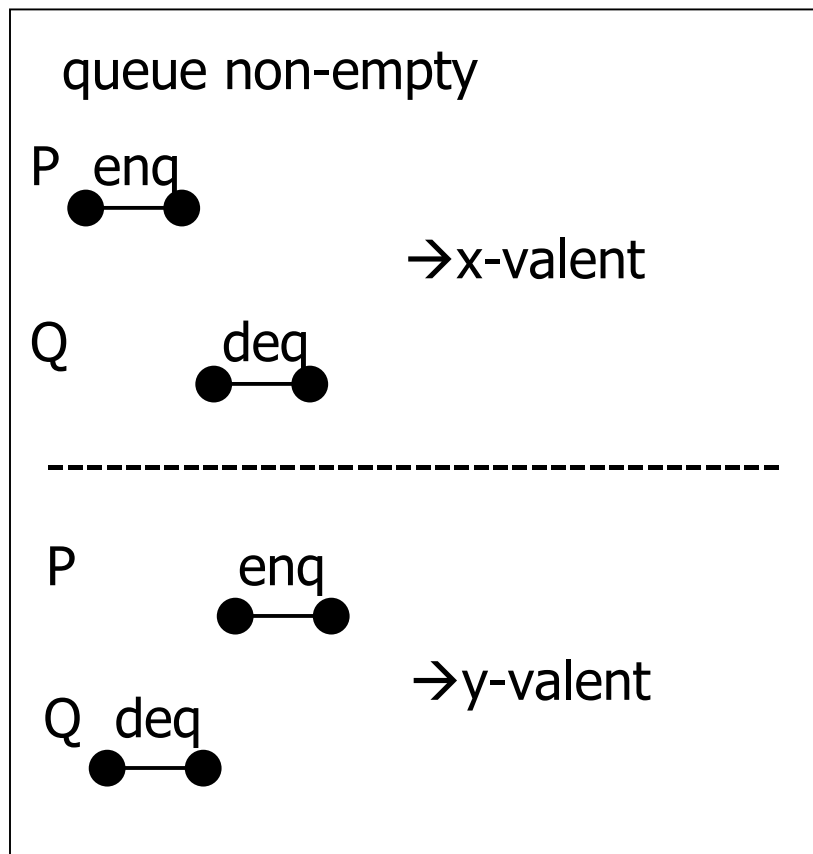
# FIFO-Queue: Konsensusbildung

- 3 Prozesse P, Q und R
- P führt in x-wertigen Zustand
- Q führt in y-wertigen Zustand
- 1. Fall: P und Q führen dequeue-Operationen aus



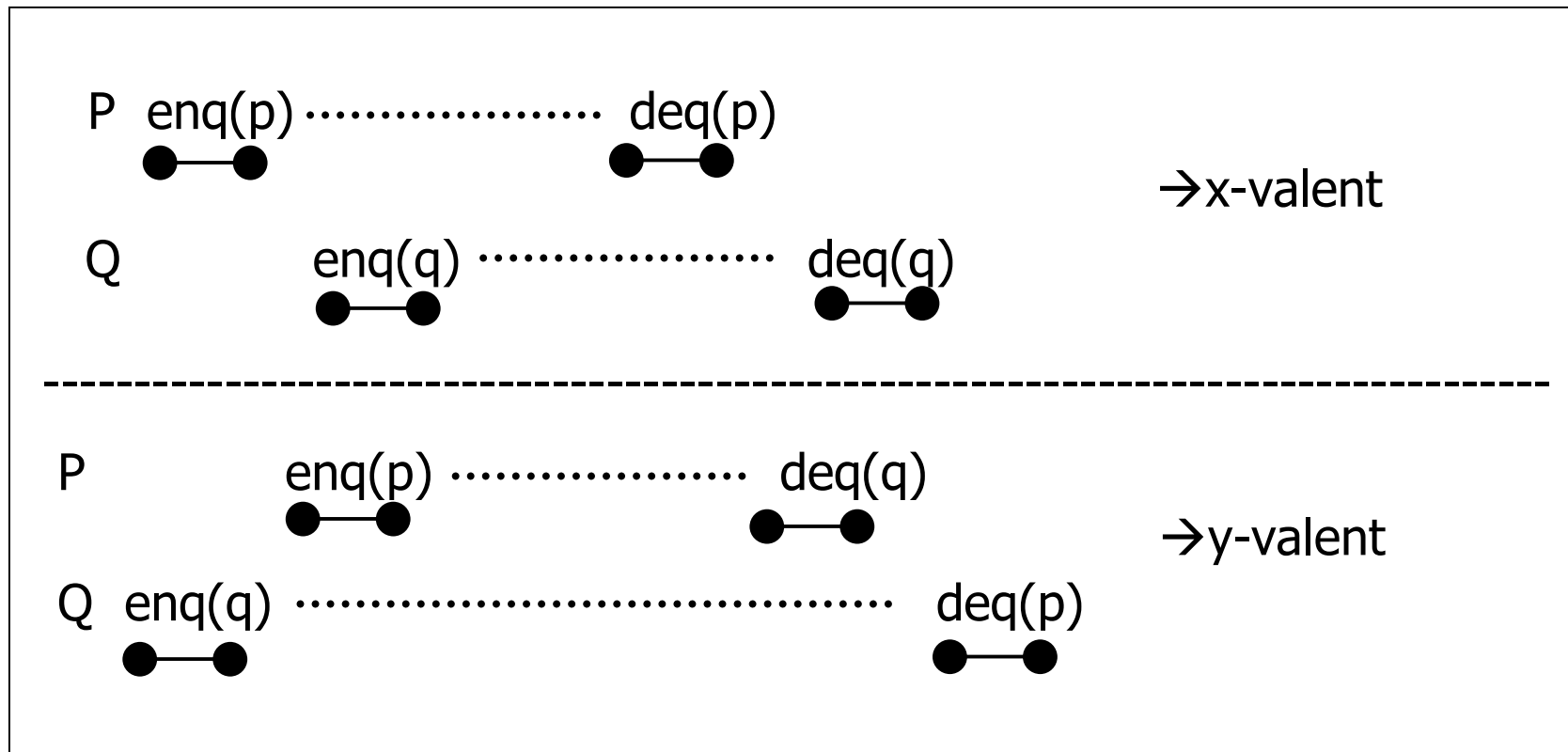
# FIFO-Queue: Konsensusbildung

## 2. Fall: P führt enqueue und Q dequeue-Operation aus



# FIFO-Queue: Konsensusbildung

3. Fall: P und Q fügen ein Element in die Queue ein





# Ein universelles wartefreies Konstrukt

---

- Bisher:
  - Konsensusnummer  $n \rightarrow$  wartefrei für  $n$  Prozesse
- Im Folgenden:
  - sequentielle Objekt  $\rightarrow$  wartefreies Objekt
  - Problem der Prioritätenumkehr wird gelöst



# Ein universelles wartefreies Konstrukt

---

## ■ Prozesse:

- `Announce`:  $i$ -tes Element ist Zeiger auf Speicherzelle  $i$
- `Head`:  $i$ -tes Element ist Zeiger auf letzte Speicherzele, die von  $i$  angeschaut wurde

## ■ Objekte (doppelt-verkettete Liste):

- `Seq`: Speicherzellenummer
- `Inv`: invocation
- `New`: Konsensusobjekt (`new.state`, `new.result`)
- `Before`, `After`

```

1 universal(what: INVOC) returns(RESULT)
2 mine: cell:=[seq: 0, inv: what,new: create(consensus_object),
           before: create(consensus_object), after: null]
3 for each process Q do
4     head[P] := max(head[P], head[Q])
5 end for
6 while announce[P].seq = 0 do
7     c: *cell := head[P]
8     help:*cell:= announce[(c.seq mod n)+1]
9     if help.seq = 0
10         then prefer := help
11         else prefer := announce[P]
12     end if
13     d := decide(c.after, prefer)
14     decide(c.new,apply(d.inv,c.new.state))
15     d.before := c
16     d.seq := c.seq + 1
17 end while
18 return (announce[P].new.result)
19 end universal

```



# Fazit: universelles wartefreies Konstrukt

---

- Vorteil

- Löst Problem der Prioritätenumkehr

- Nachteil

- Speicherplatzbedarf:  $O(n^3)$
- Laufzeit im Worst Case:  $O(n^3)$



# Zusammenfassung

---

- Wartefreie Synchronisation
  - Stärkste Fortschrittsbedingung von nicht-blockierender Synchronisation
  - Löst Probleme anderer Synchronisationsparadigmen
- Konsensusnummer
  - Aussage über wartefreie Implementierung
  - Konsensusprotokoll