Concurrent Reading and Writing

Motivation

```
Int32 _zaehler, _temp = 0;

Thread _thread1;
Thread _thread2;
```

```
_temp = _zaehler;
_temp = _temp + 1;
_zaehler = _temp;
```

Motivation

_thread1	_zaehler	_thread2
	0	
_temp = _zaehler = 0 _temp = 1 _zaehler = _temp = 1	1	
	2	_temp = _zaehler = 1 _temp = 2 _zaehler = _temp = 2
		_temp = _zaehler = 2 _temp = 3
_temp = _zaehler = 2 _temp = 3 _zaehler = _temp = 3	3	
	3	_zaehler = _temp = 3

!=4

Motivation

Gleichzeitige Änderung geteilter Daten/Variablen:

-> Wettlaufsituation

Verschiedene Datenversionen

Nur **eine** bestimmte Version

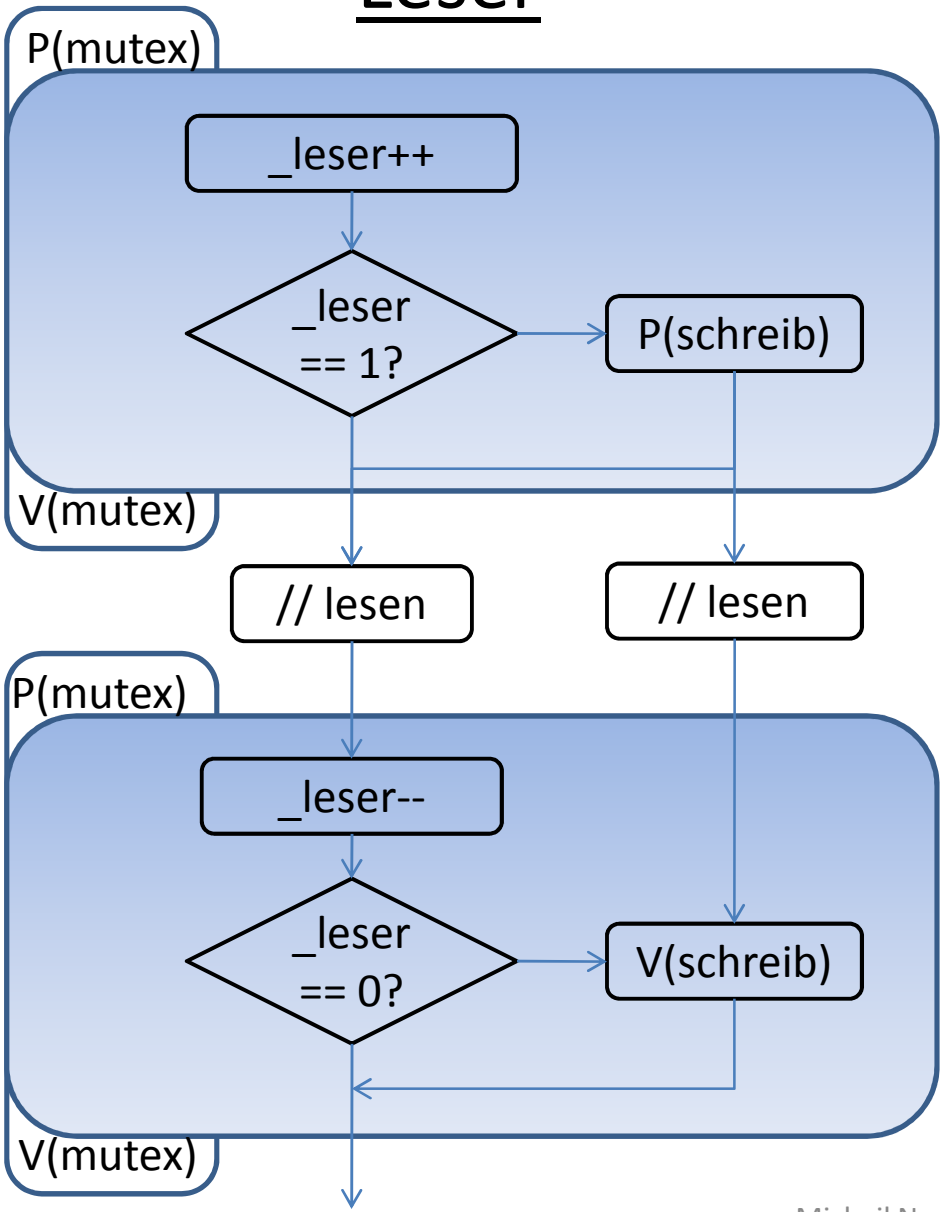
=> Blockierungslösung

Motivation

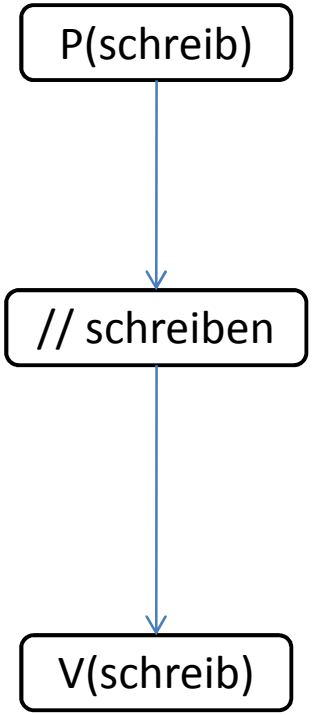
_thread1	_zaehler	_thread2
	0	
_temp = _zaehler = 0 _temp = 1 _zaehler = _temp = 1	1	
	2	_temp = _zaehler = 1 _temp = 2 _zaehler = _temp = 2
	3	_temp = _zaehler = 2 _temp = 3 _zaehler = _temp = 3
_temp = _zaehler = 3 _temp = 3 _zaehler = _temp = 4	4	

Semaphorblockierung

Leser



Schreiber

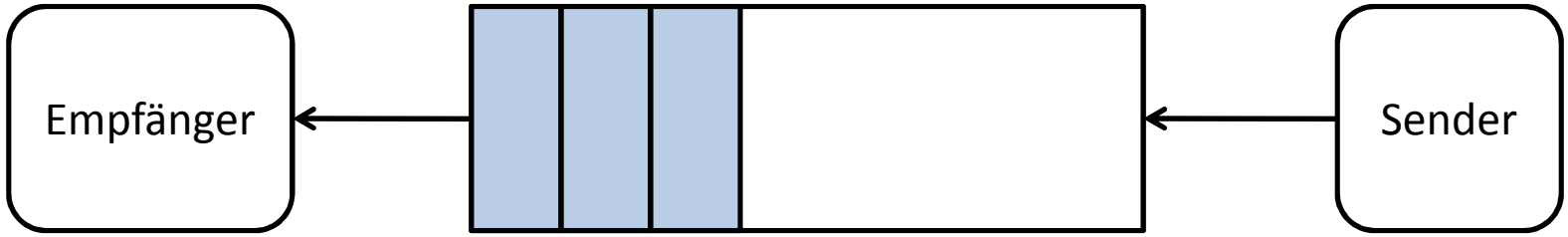


Semaphorblockierung

- Erste Leser ruft P(schreib)
- Solange diese Semaphore in Platz ist können Leser frei lesen
- Letzte Leser ruft V(schreib)

- **Nachteil**: Schreiber können für lange Zeit blockiert werden

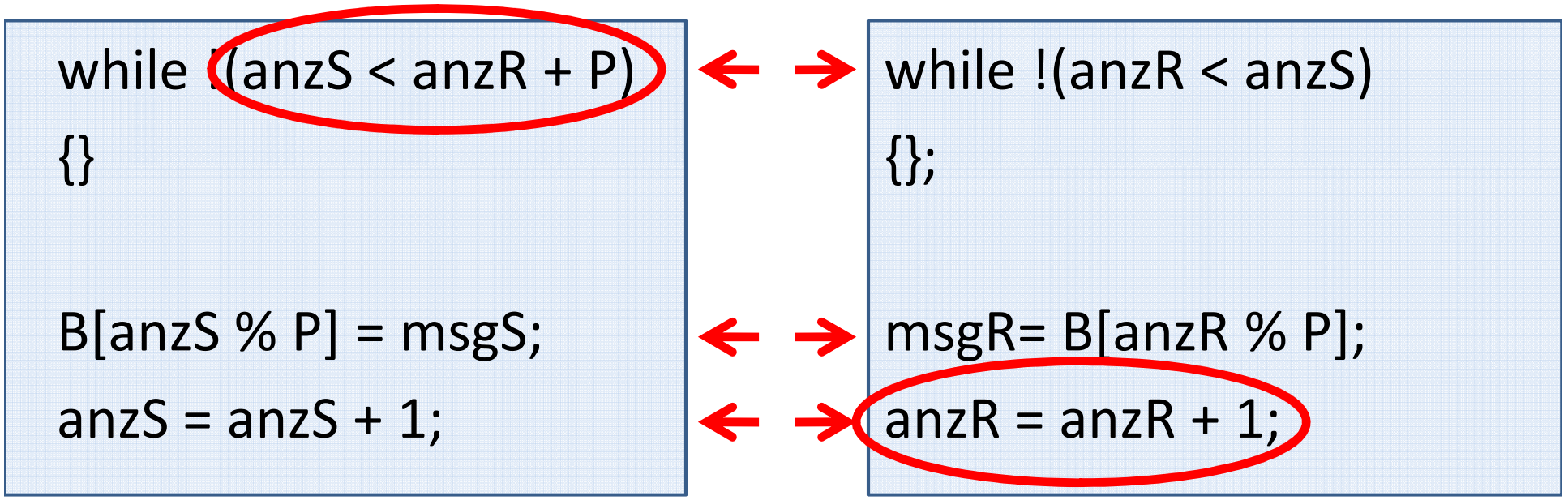
Message Buffer



Message Buffer

- **P** = Puffergröße
- **anzS** = Anzahl der gesendeten Nachrichten
- **anzR** = Anzahl der gelesenen Nachrichten
- **msgS** = Zu sendende Nachricht
- **msgR** = Zu lesende Nachricht
- Puffer leer: Empfänger muss warten
- Puffer voll: Sender muss warten

Message Buffer



Theorem

Reihenfolge vermeiden:

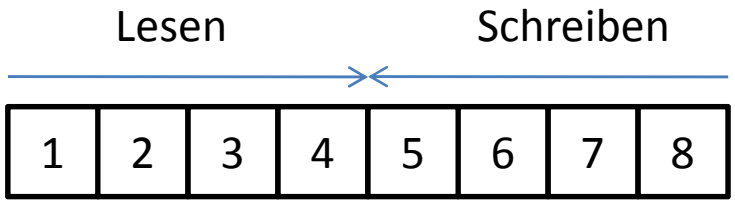
– Ein Lesezugriff darf **keine** Spuren von **mehreren** Daten bekommen

Spuren:

- Anfang des Lesens kommt vor dem Ende des Schreibens der neuen Daten
- Ende des Lesens kommt nach dem Anfang des Schreibens der alten Daten

Theorem

- Spuren umgehen:
  - Lesen von links nach rechts der Daten
  - Schreiben von rechts nach links der Daten



Theorem

- Wenn v von ⟵ geschrieben wird, dann erhält lesen von ⟶

  v[k,l] <= v[l].

- Wenn v von ⟶ geschrieben wird, dann erhält lesen von ⟵

  v[k,l] >= v[k]

“General Readers Writers Solution”

Versionsnummern v1 und v2 verwalten Synchronisation

Schreiber: (1 mal)

```
v1 = v1 + 1;
// schreiben
v2 = v2 + 1;
```

Leser: (x mal)

```
do
{
    temp = v2;
    // lesen
}
while (v1 != temp)
```

General Readers Writers Solution

- Wird verwendet wenn:
  - Das warten des Schreibers ungewollt ist
  - Wenn die Chance sehr gering ist, dass eine Leseoperation wiederholt werden muss

Message Buffer

Sender:

```
while !(anzS < anzR + P)
{}

B[anzS % P] = msgS;
anzS = anzS + 1;
```

Empfänger:

```
while !(anzR < anzS)
{};

msgR= B[anzR % P];
anzR = anzR + 1;
```

Message Buffer

Sender:

```
while !(anzS < anzR + P)
{}

B[anzS % P] = msgS;
anzS = anzS + 1;
```

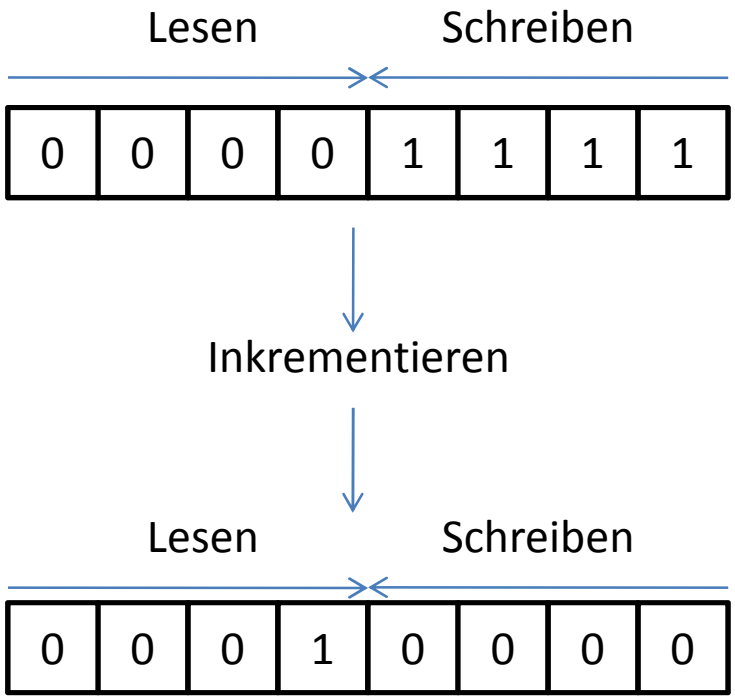
Empfänger:

```
while !(anzR < anzS)
{};

msgR= B[anzR % P];
anzR = anzR + 1;
```

Vielen Dank!

Theorem


Lesen
Schreiben
0 0 0 0 1 1 1 1
Inkrementieren
Lesen
Schreiben
0 0 0 1 0 0 0 0