

Betriebssysteme (BS)

VL 6.1 – Unterbrechungen, Synchronisation – Motivation

Volkmar Sieh / Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

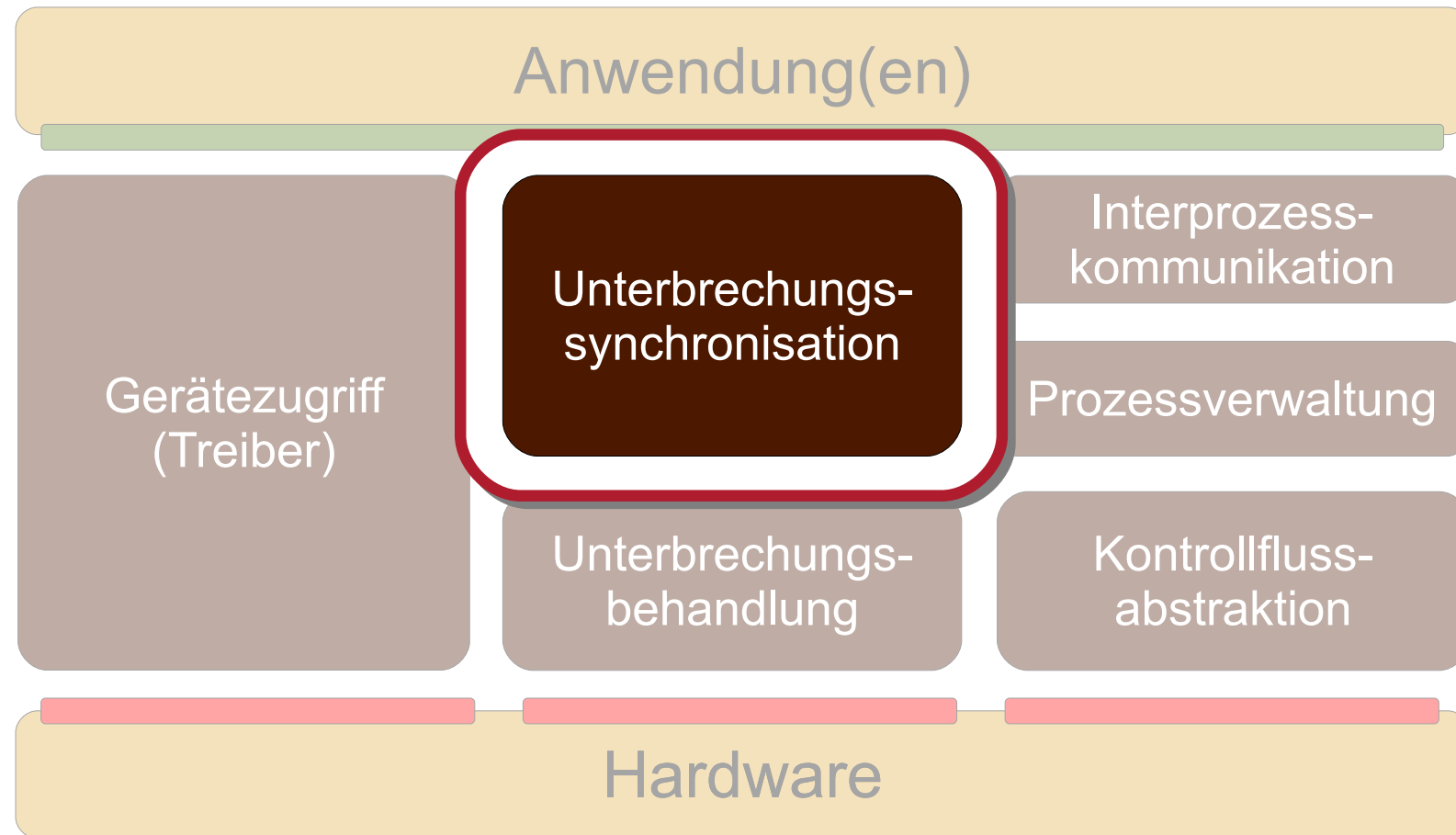
Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 21 – 29. November 2021



https://www4.cs.fau.de/Lehre/WS21/V_BS

Überblick: Einordnung dieser VL



Betriebssystementwicklung



Agenda

Einleitung
Prioritätsebenenmodell
Harte Synchronisation
Weiche Synchronisation
Prolog/Epilog-Modell
Zusammenfassung
Referenzen



Agenda

Einleitung

Motivation

Erstes Fazit

Prioritätsebenenmodell

Harte Synchronisation

Weiche Synchronisation

Prolog/Epilog-Modell

Zusammenfassung

Referenzen



Zustandsänderungen ...

- sind Sinn und Zweck der Unterbrechungsbehandlung
 - Gerätetreiber müssen über den Abschluss einer E/A Operation informiert werden
 - der Scheduler muss erfahren, dass eine Zeitscheibe abgelaufen ist
- müssen mit Vorsicht durchgeführt werden
 - Unterbrechungen können zu jeder Zeit auftreten
 - kritisch sind Daten/Datenstrukturen, die der normale Kontrollfluss die Unterbrechungsbehandlung sich teilen

Beispiele aus der letzten Vorlesung

Beispiel 2: Ringpuffer

auch die Pufferimplementierung ist kritisch ...

```
// Pufferklasse in C++
class BoundedBuffer {
    char buf[SIZE]; int occupied; int nextin, nextout;
public:
    BoundedBuffer(): occupied(0), nextin(0), nextout(0) {}
    void produce(char data) { // Unterbrechungsbehandlung:
        int elements = occupied; // Elementzähler merken
        if (elements == SIZE) return; // Element verloren
        buf[nextin] = data; // Element schreiben
        nextin++; nextin %= SIZE; // Zeiger weitersetzen
        occupied = elements + 1; // Zähler erhöhen
    }
    char consume() { // normaler Kontrollfluss:
        int elements = occupied; // Elementzähler merken
        if (elements == 0) return 0; // Puffer leer, kein Ergebnis
        char result = buf[nextout]; // Element lesen
        nextout++; nextout %= SIZE; // Lesezeiger weitersetzen
        occupied = elements - 1; // Zähler erniedrigen
        return result; // Ergebnis zurückliefern
    }
};
```

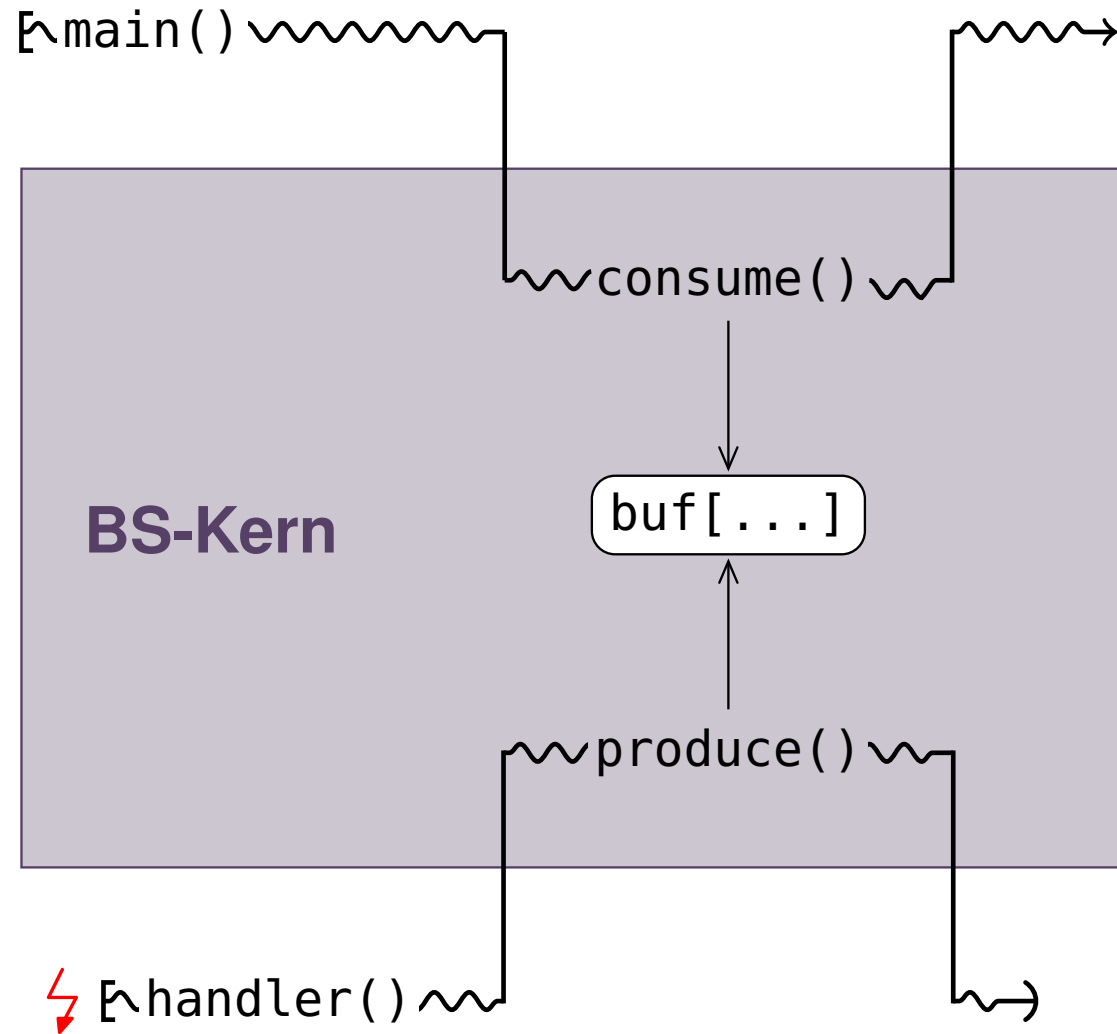
Motivation: Ursache

Kontrollflüsse
“von oben”

“begegnen”
sich im Kern

und “von unten”

Anwendungskontrollfluss (A)



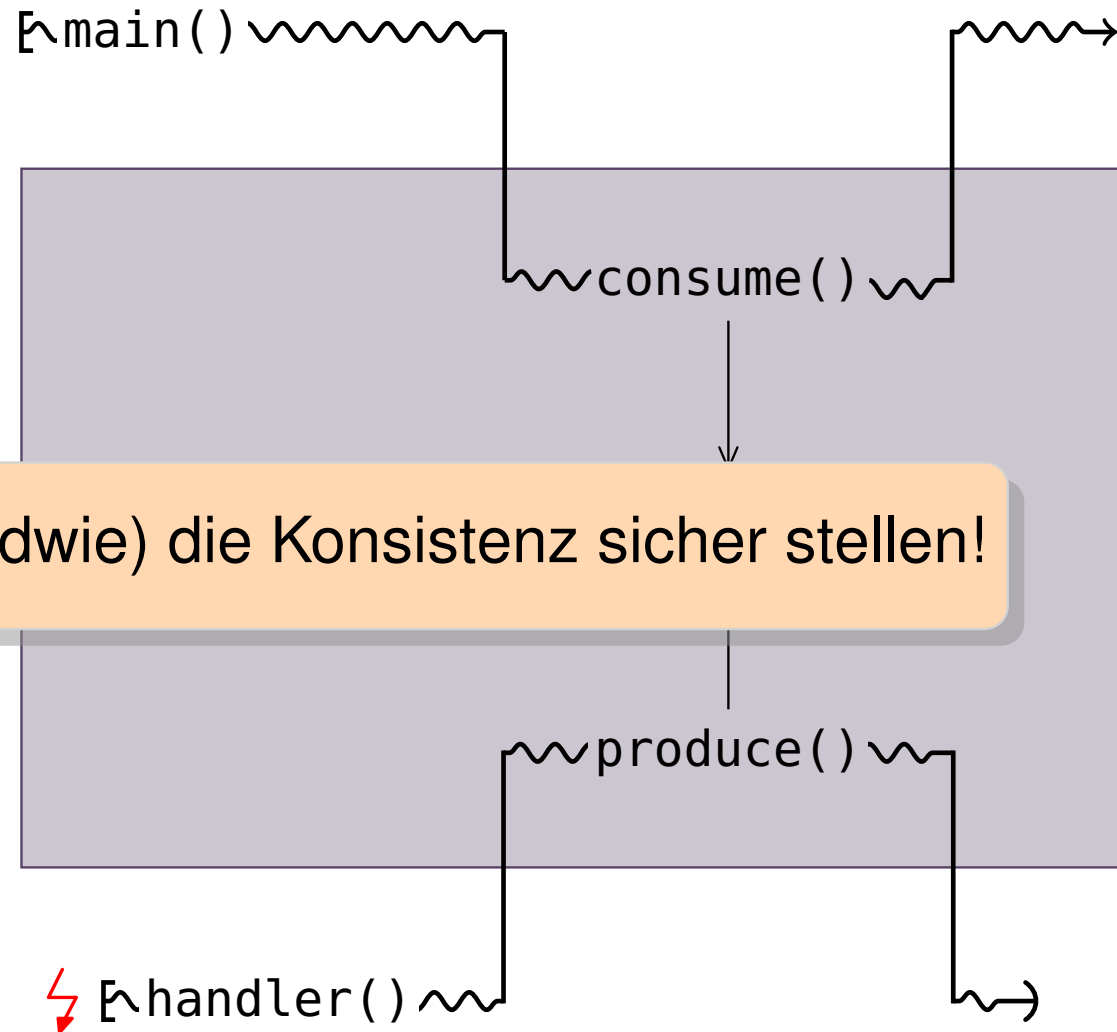
Unterbrechungskontrollfluss (UB)



Motivation: Ursache

Kontrollflüsse
“von oben”

Anwendungskontrollfluss (A)



“b
sic

Wir müssen (irgendwie) die Konsistenz sicher stellen!

und “von unten”

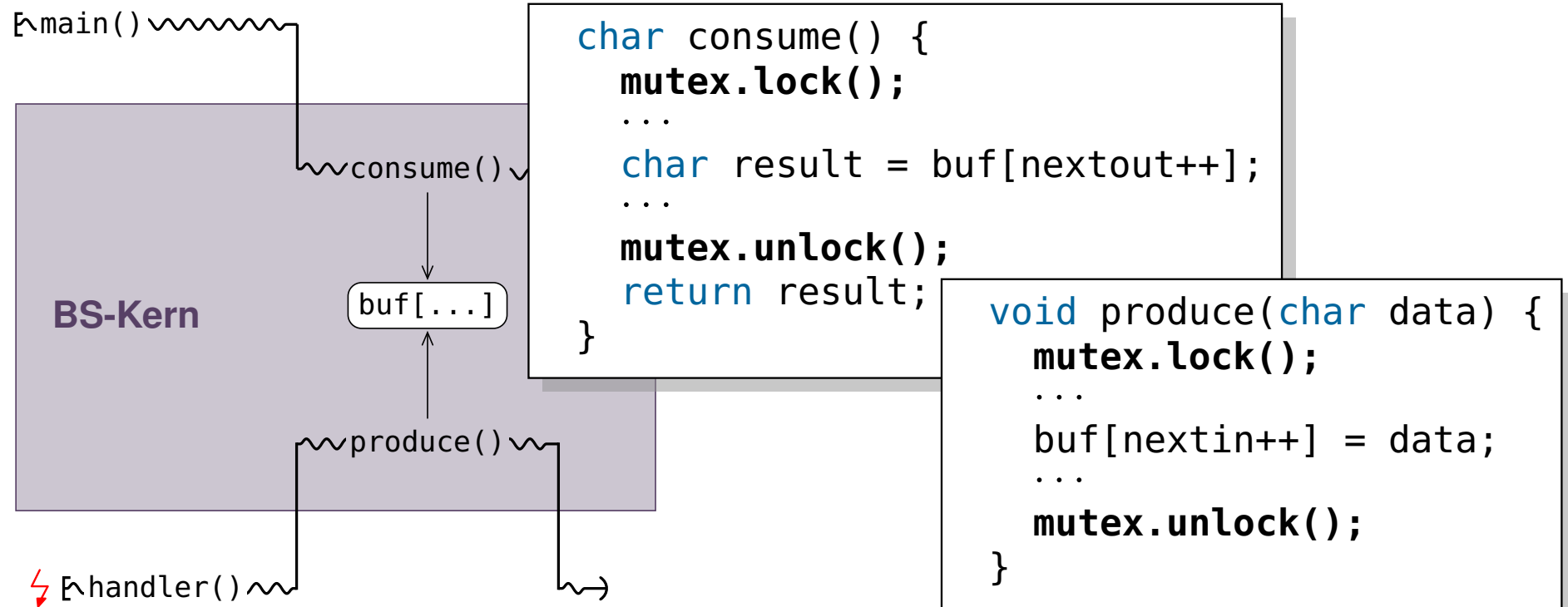
Unterbrechungskontrollfuss (UB)



Naiver Lösungsansatz

- Zweiseitige Synchronisation
 - gegenseitiger Ausschluss durch Mutex, *Spin-Lock*, ... (vgl. [SP])
 - wie zwischen zwei Prozessen

Anwendungskontrollfluss (A)



Unterbrechungskontrollfuss (UB)

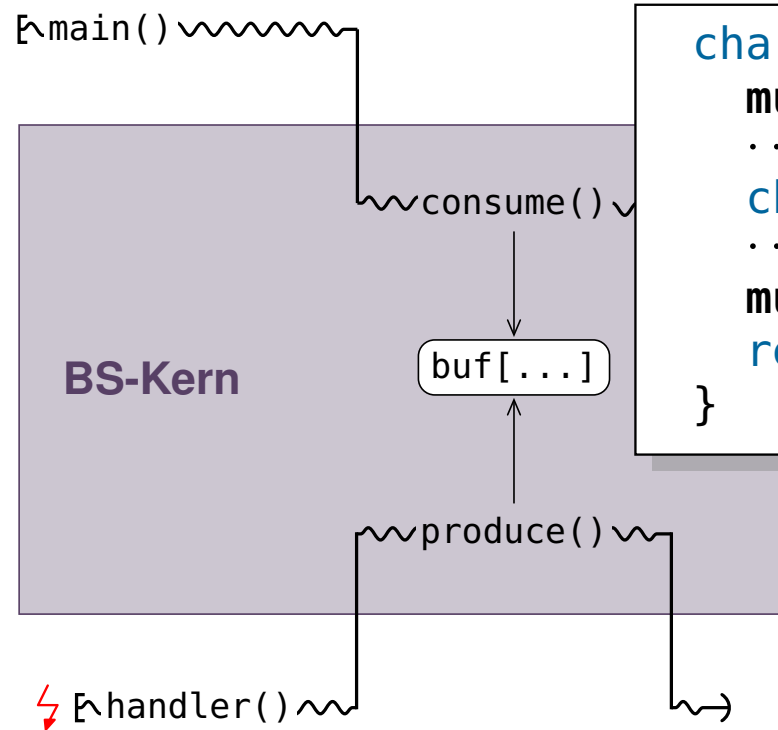


Naiver Lösungsansatz

- Zweiseitige Synchronisation
 - gegenseitiger Ausschluss durch M
 - wie zwischen zwei Prozessen

Zweiseitige Synchronisation funktioniert **natürlich nicht!**

Anwendungskontrollfluss (A)



```
char consume() {  
    mutex.lock();  
    ...  
    char result = buf[nextout++];  
    ...  
    mutex.unlock();  
    return result;  
}
```

```
void produce(char data) {  
    mutex.lock();  
    ...  
    buf[nextin++] = data;  
    ...  
    mutex.unlock();  
}
```

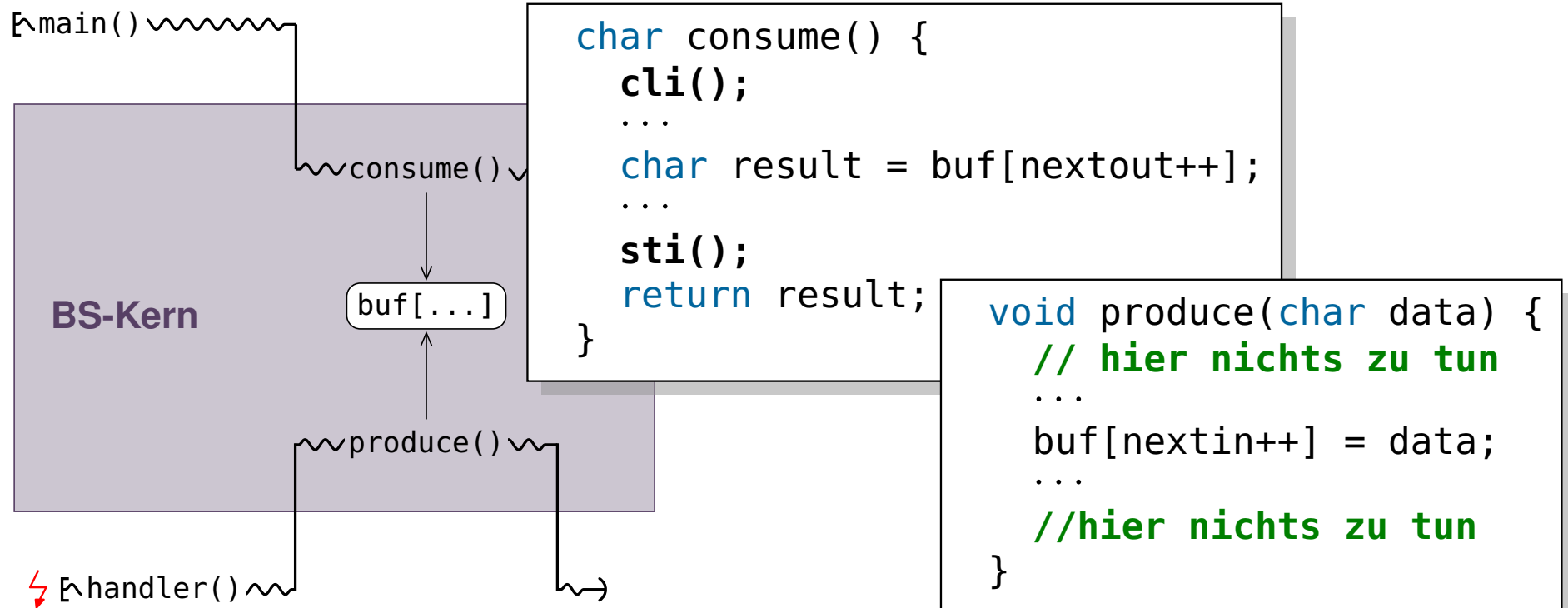
Unterbrechungskontrollfuss (UB)



Besserer Lösungsansatz

- Einseitige Synchronisation
 - Unterdrückung der Unterbrechungsbehandlung im Verbraucher
 - Operationen `disable_interrupts()` `enable_interrupts()`
(im Folgenden o. B. d. A. in „Intel“-Schreibweise: `cli()` / `sti()`)

Anwendungskontrollfluss (A)



Unterbrechungskontrollfluss (UB)



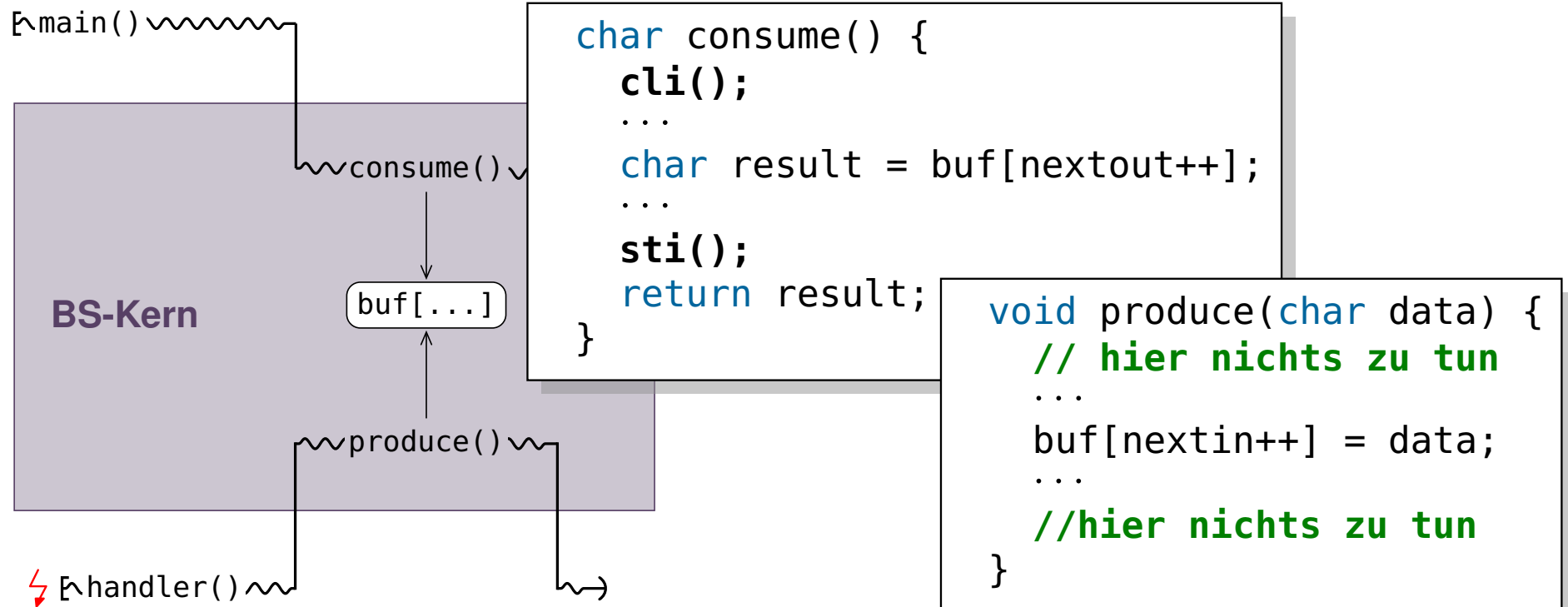
Besserer Lösungsansatz

■ Einseitige Synchronisation

- Unterdrückung der Unterbrechungen
- Operationen `disable_interrupts()` (im Folgenden o. B. d. A. in „Intel“-Schreibweise)

Einseitige Synchronisation funktioniert. [Warum?]

Anwendungskontrollfluss (A)



Unterbrechungskontrollfluss (UB)



Erstes Fazit

- Konsistenzsicherung zwischen
 - Anwendungskontrollfluss (A) und
 - Unterbrechungsbehandlung (UB)muss **anders erfolgen** als zwischen Prozessen
- Die Beziehung zwischen A und UB ist **asymmetrisch**
 - Es handelt sich um „verschiedene Arten“ von Kontrollflüssen
 - UB **unterbricht** Anwendungskontrollfluss
 - implizit, an beliebiger Stelle
 - hat immer Priorität, läuft durch (*run-to-completion*)
 - A kann UB **unterdrücken** (besser: **verzögern**)
 - explizit, mit `cli/sti` (Grundannahme 5 aus VL 4)
- Synchronisation / Konsistenzsicherung erfolgt **einseitig**



Erstes Fazit

- Konsistenzsicherung zwischen
 - Anwendungskontrollfluss (A) und
 - Unterbrechungsbehandlung (UB)muss **anders erfolgen** als zwischen Prozessen
- Die Beziehung zwischen A und UB ist **asymmetrisch**
 - Es handelt sich um „verschiedene Arten“ von Kontrollflüssen
 - UB **unterbricht** Anwendungskontrollfluss
 - implizit, an beliebiger Stelle
 - hat immer Priorität, läuft durch (*run-to-completion*)
 - A kann UB **unterdrücken** (besser: **verzögern**)
 - explizit, mit `cli/sti` (Grundannahme 5 aus VL 4)
- Synchronisation / Konsistenzsicherung erfolgt **einseitig**

Diese Tatsachen müssen wir **beachten!**

(Das heißt aber auch: Wir können sie **ausnutzen**)

