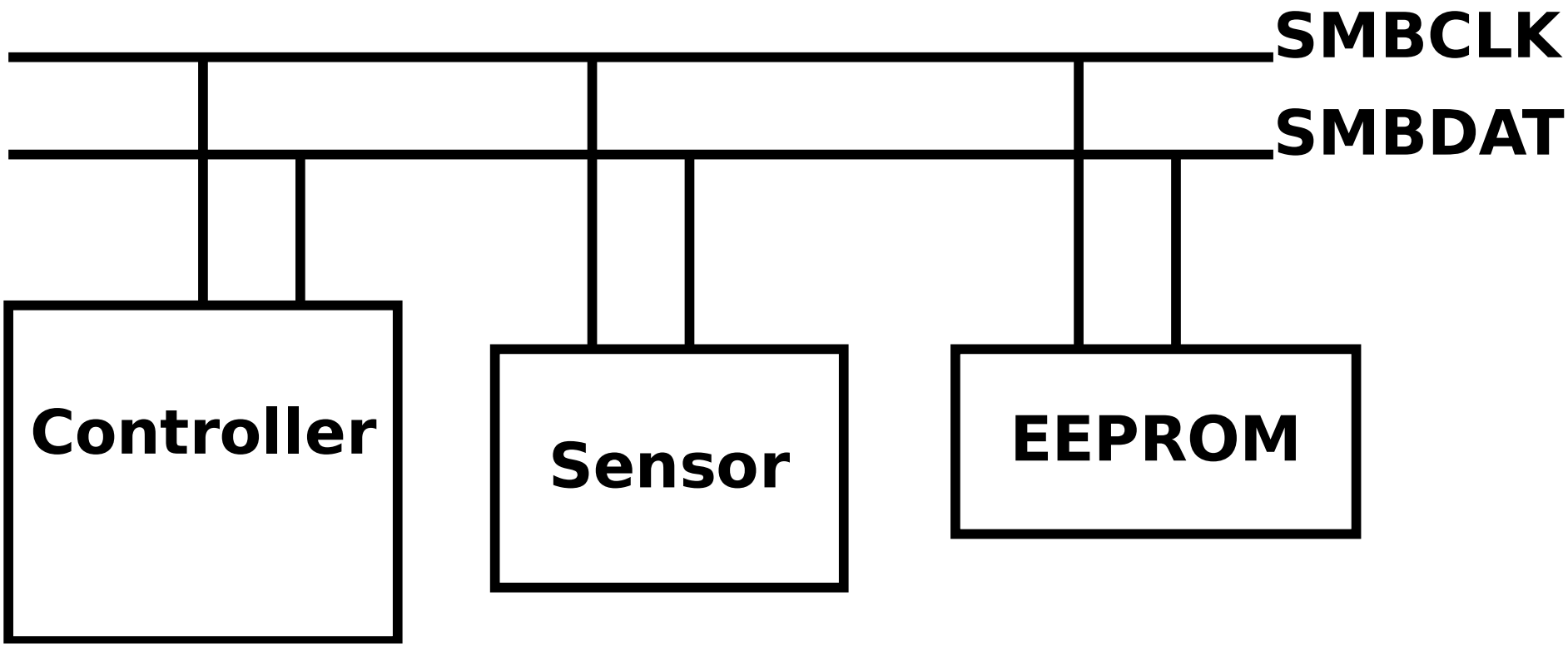
Never ever break userspace - was das in der Praxis bedeutet

Wolfram Sang, Consultant / Renesas

16.03.2024, CLT2024

SMBus Spezifikation

SMBus Beispiel



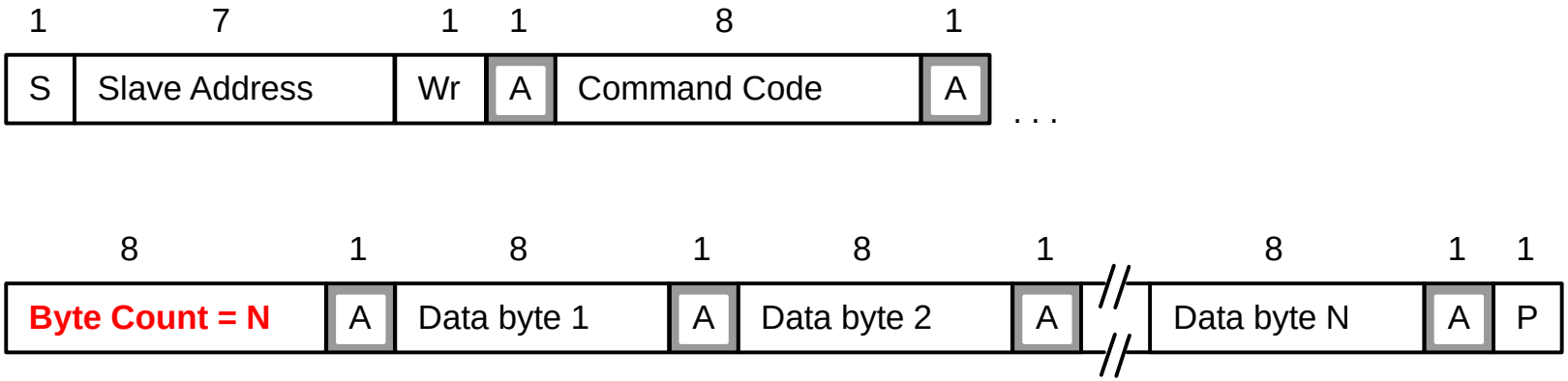
SMBus ist quasi eine ''Untermenge'' von I2C

SMBus Transaction Types

SMBus hat genauer spezifizierte Transaktionen

- read_byte / write_byte
- read_word / write_word
- **read_block / write_block**
- process_call
- ...

Block Transfer Write (1998)

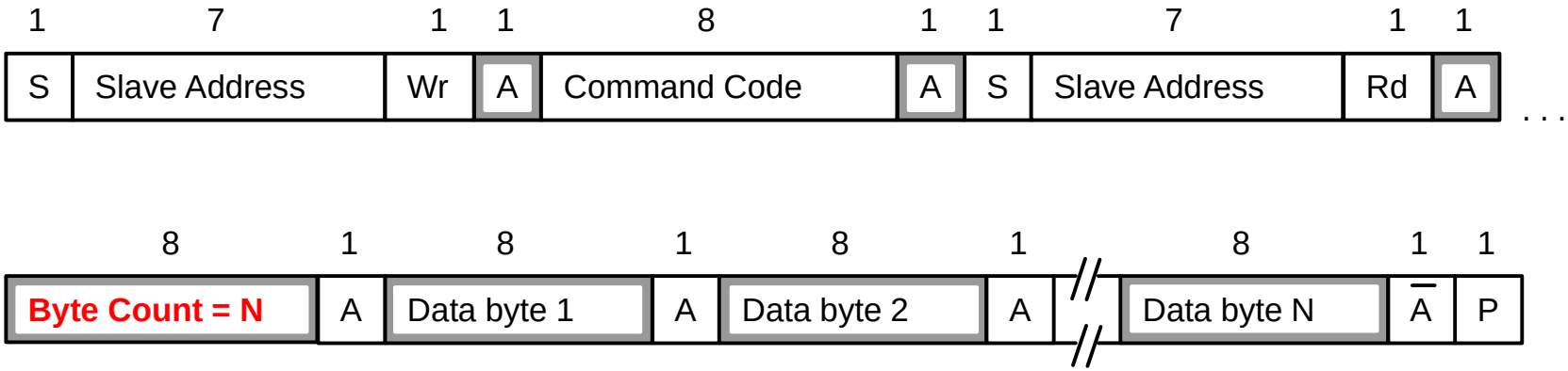


Block Write

The byte count may not be 0. A Block Read or Write is allowed to transfer a maximum of 32 data bytes.

System Management Bus Specification, Revision 1.1, December 11, 1998, Section 7.5.7, Page 25f

Block Transfer Read (1998)



Block Read

Die Blockgröße kommt vom Client!

Block Transfer Update (2014)

The byte count may be 0.[1] *A Block Read or Block Write is allowed to transfer a maximum of 255 data bytes.*

System Management Bus (SMBus) Specification, Version 3.0, 20 Dec 2014, Section 6.5.7, Page 42

[1] Was heißt das?

Warum?

- bessere / volle Kompatibilität mit
  - SMBus 3.x
  - PMBus
  - ...
- Unterstützung für nicht-abwärtskompatible Geräte
- Strom sparen durch geringere Ressourcen-Nutzung

  Sultan Alsawaf berichtete von 1.6W(!) weniger Verbrauch des Touchpads
- keine Buffer Overflows mehr

  Maintainers Favorit
- fast alle Controller unterstützen das im Prinzip schon

  sofort verfügbar
- schneller ist es marginal, aber weniger Verwaltungsaufwand

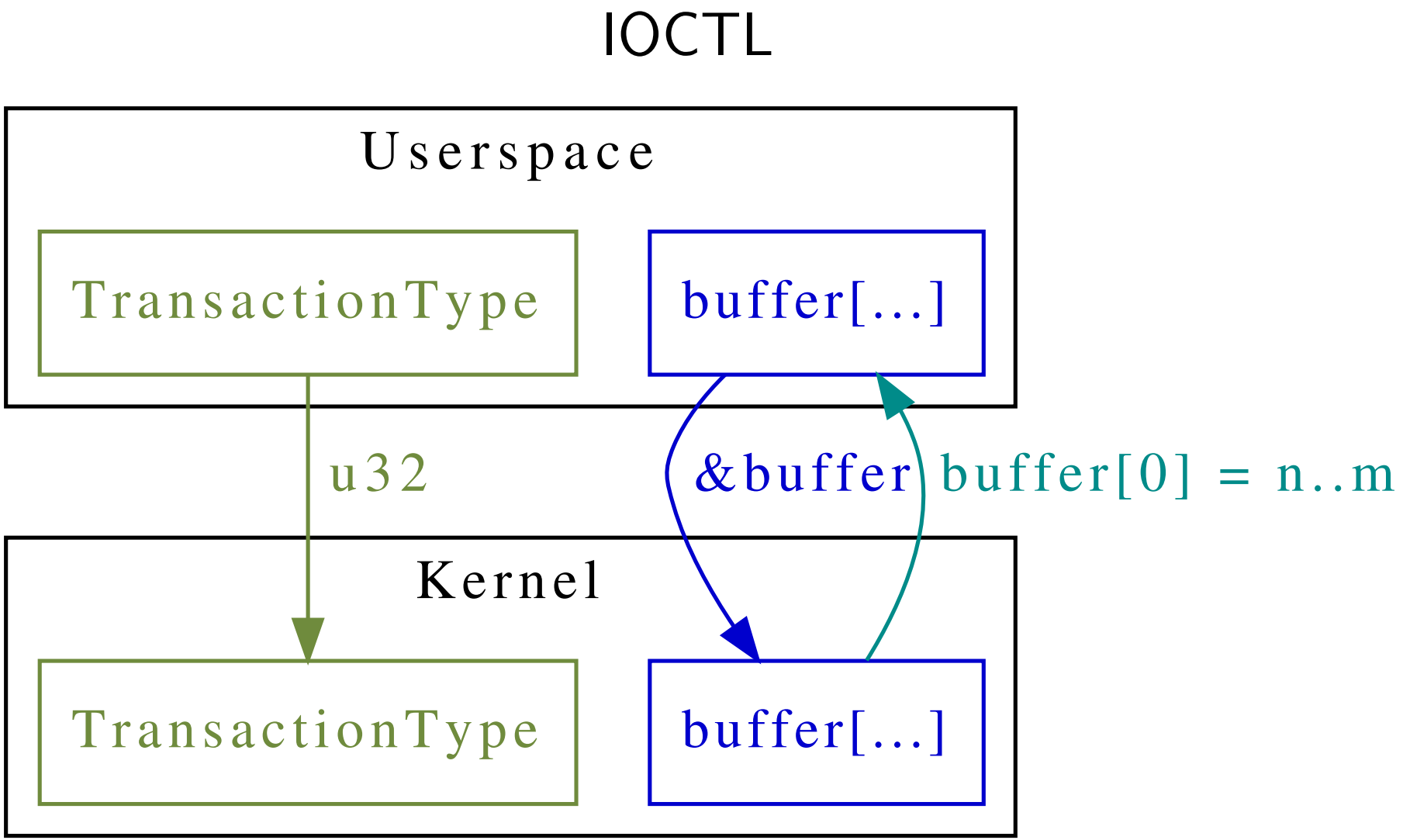
Zusammenfassung

- SMBus hat verschiedene Typen von Transaktionen
- Block-Transaktionen haben Limitationen
- diese Limitationen wurden 2014 angehoben

  sonst hat sich am Block-Transfer nichts geändert
- die meisten Controller unterstützen größere Blöcke eh schon

SMBus, Linux & Userspace

Was braucht's?



Transaktionstypen

include/**uapi**/linux/i2c.h:

```
#define I2C_SMBUS_QUICK             0
#define I2C_SMBUS_BYTE              1
#define I2C_SMBUS_BYTE_DATA         2
#define I2C_SMBUS_WORD_DATA         3
#define I2C_SMBUS_PROC_CALL         4
#define I2C_SMBUS_BLOCK_DATA        5 <= hierum gehts
#define I2C_SMBUS_I2C_BLOCK_BROKEN  6 <= Zombie
#define I2C_SMBUS_BLOCK_PROC_CALL   7
#define I2C_SMBUS_I2C_BLOCK_DATA    8
```

Und die Daten?

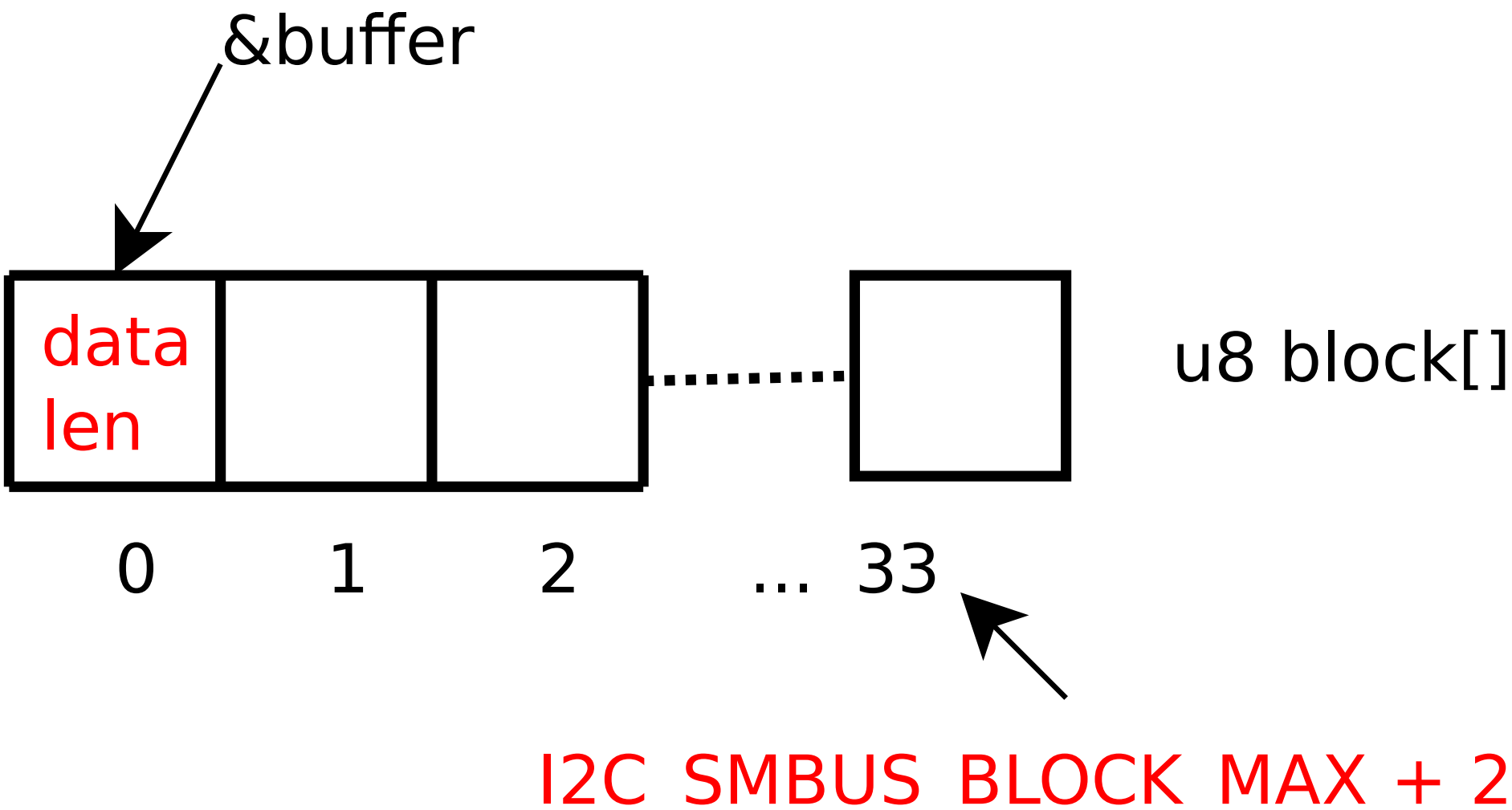
include/**uapi**/linux/i2c.h[2]:

```
#define I2C_SMBUS_BLOCK_MAX     32
      /* As specified in SMBus standard */

union i2c_smbus_data {
        ...
        __u8 block[I2C_SMBUS_BLOCK_MAX + 2];
            /* block[0] is used for length */
            /* and one more for user-space compatibility */
};
```

[2] hinzugefügt 2002

Layout in C

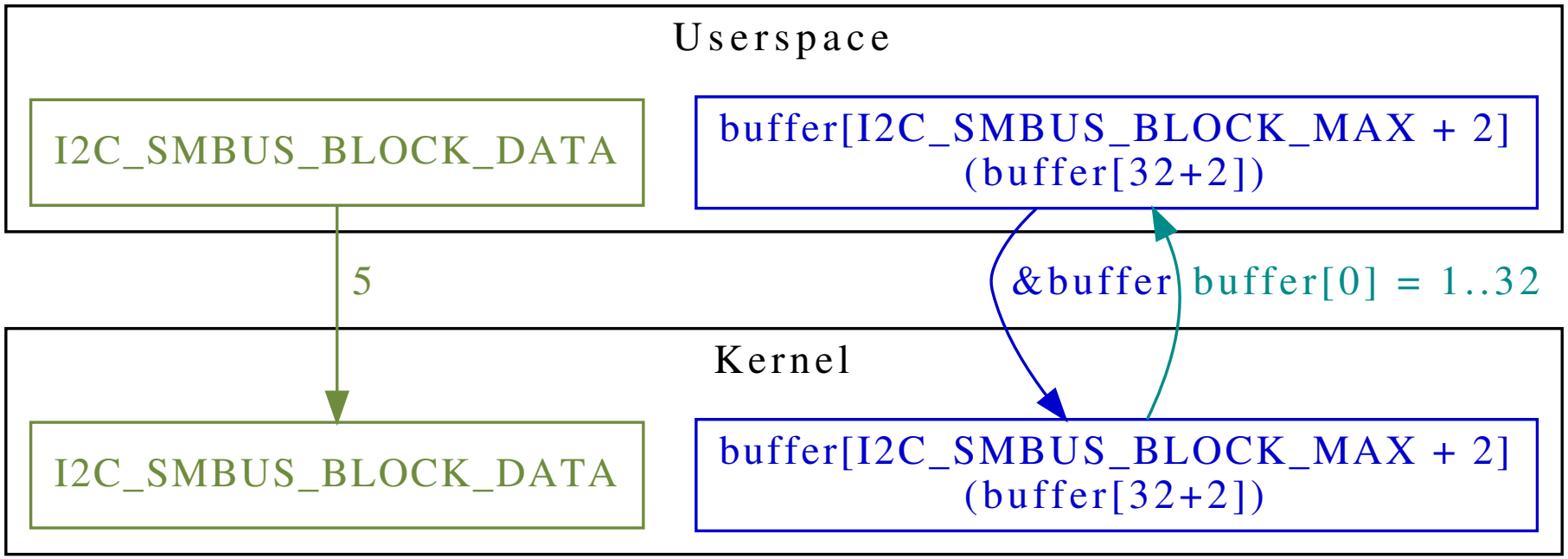


Braucht Schutz gegen Buffer Overflow

drivers/i2c/busses/i2c-imx.c:

```
if ((len == 0) || (len > I2C_SMBUS_BLOCK_MAX))
        return -EPROTO;
msgs->len += len;
```

Das funktioniert bislang gut



Wie jetzt das höhere Limit?

Der einfachste Ansatz

```
-#define I2C_SMBUS_BLOCK_MAX     32    /* "alt" */
+#define I2C_SMBUS_BLOCK_MAX     255   /* "neu" */
```

Vier Fälle zu berücksichtigen

1. alter Userspace - alter Kernel

   geht immer!

2. neuer Userspace - neuer Kernel

   geht immer!

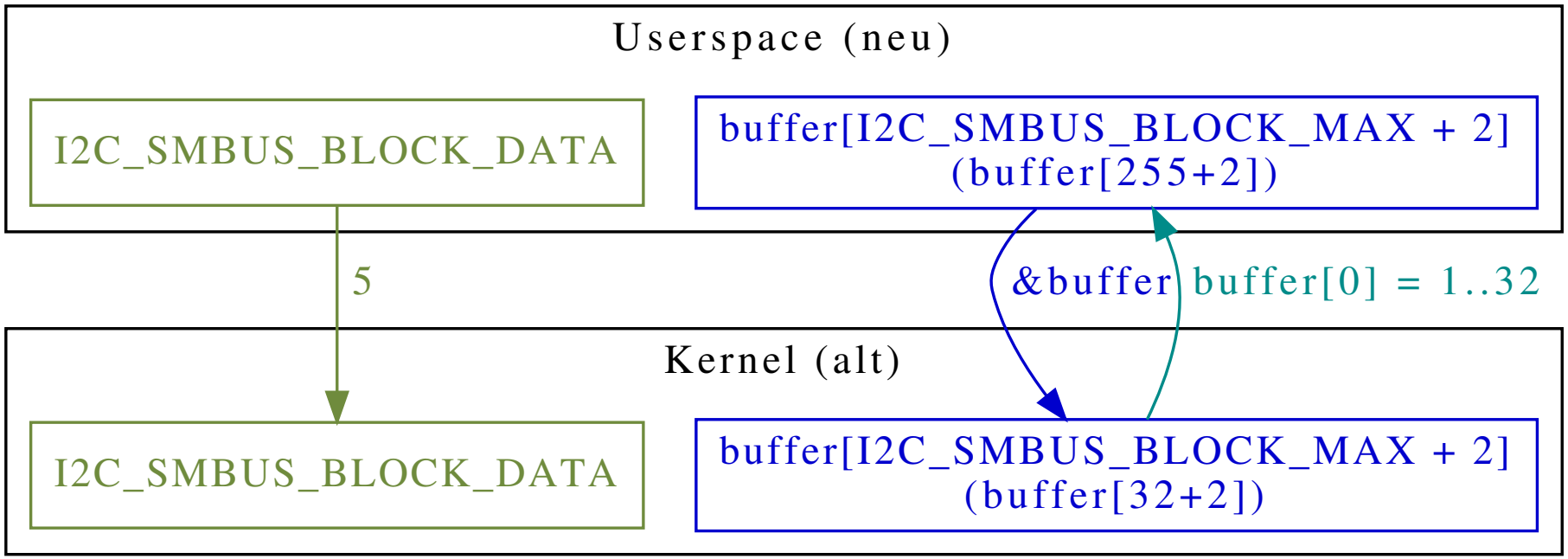
3. alter Userspace - neuer Kernel

   durch Kernel-Updates

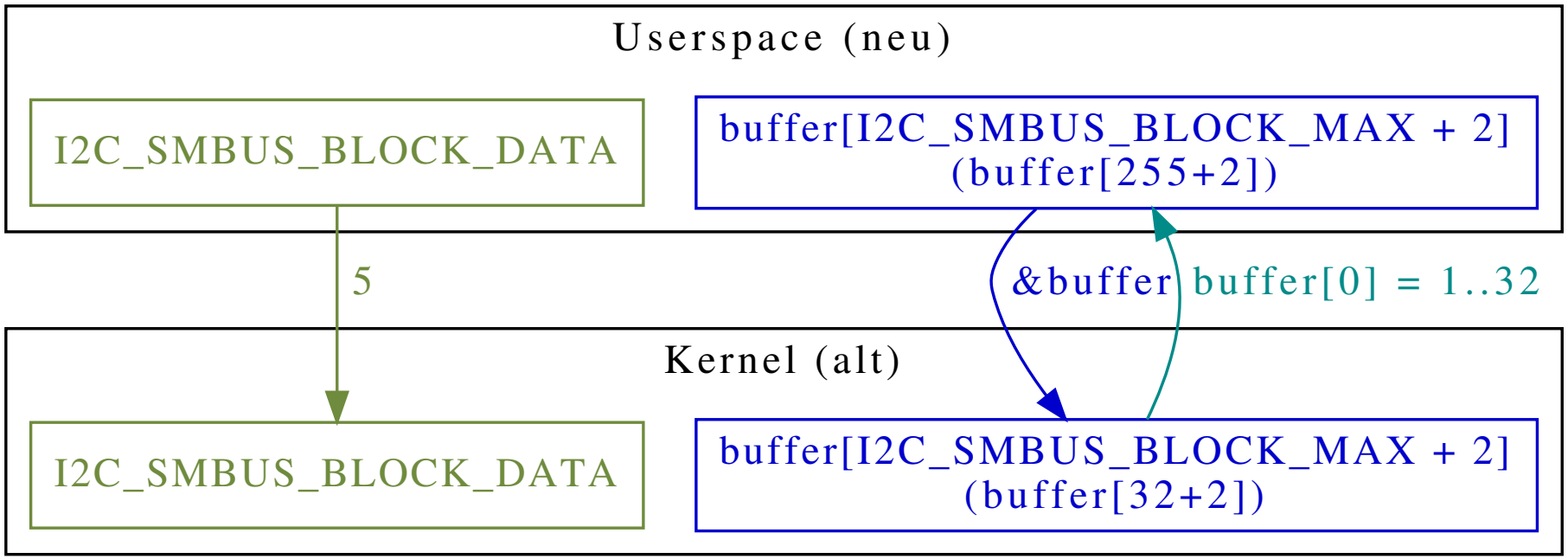
4. neuer Userspace - alter Kernel

   durch Kernel-Downgrades

Problem: neuer Userspace - alter Kernel

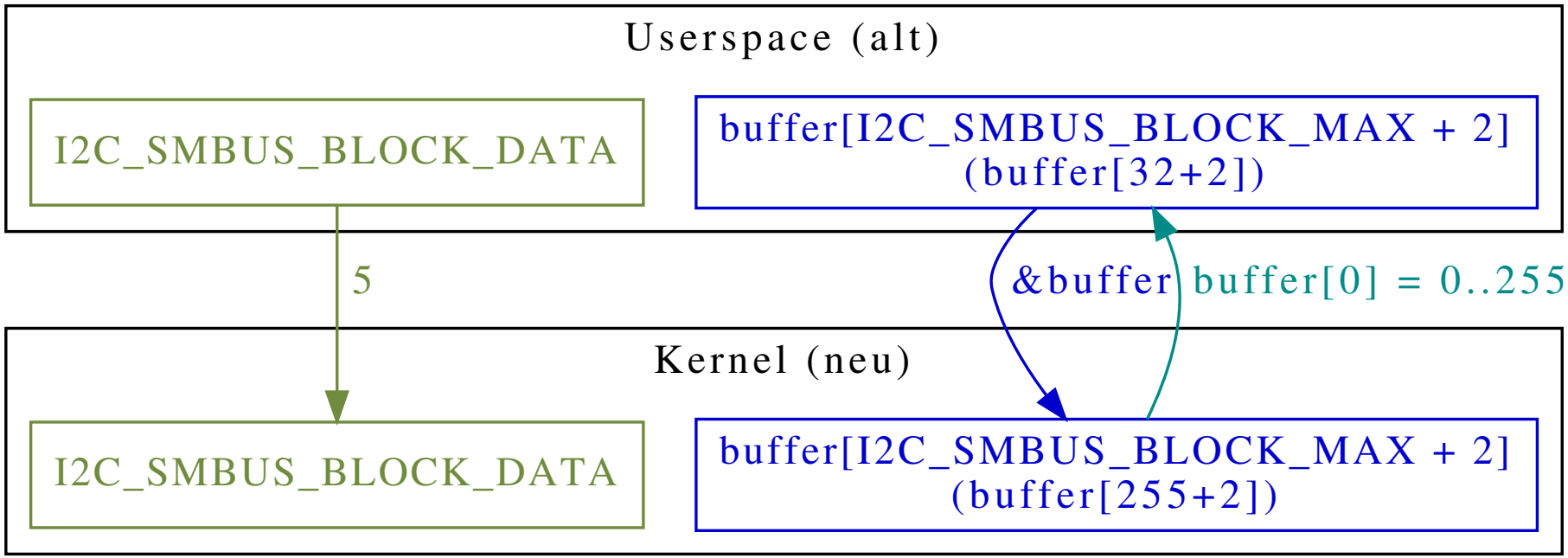


Problem: neuer Userspace - alter Kernel

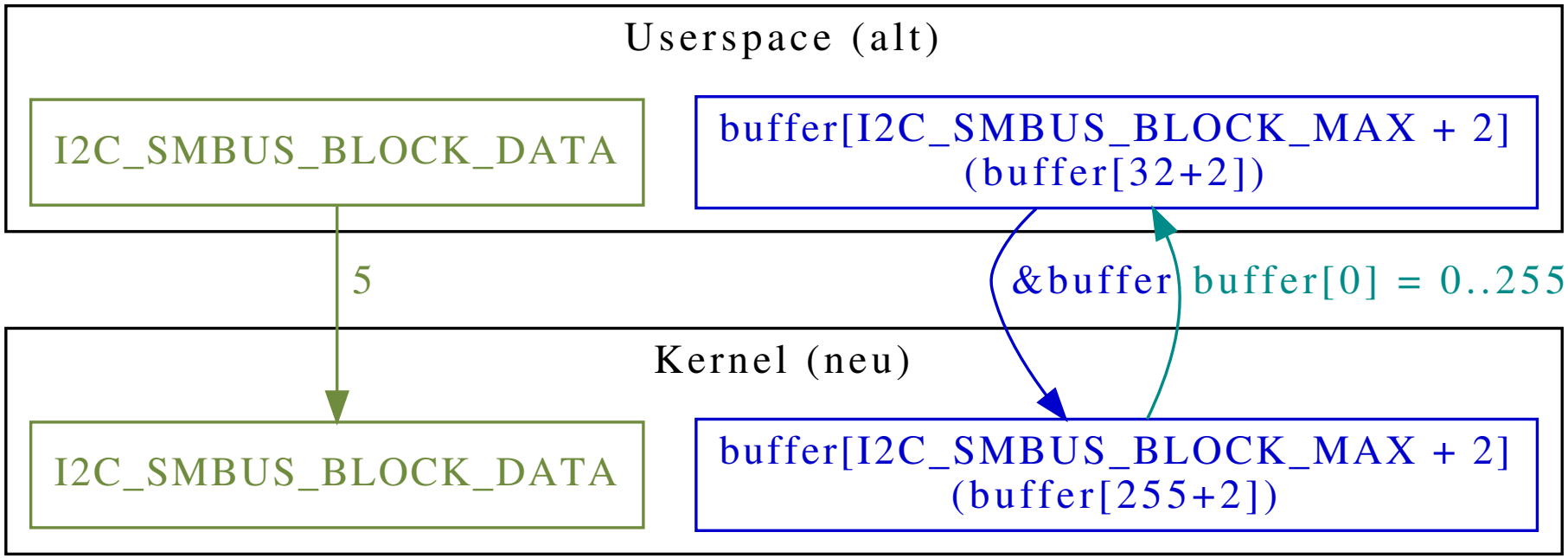


buffer[0] unerwartet klein

Problem: alter Userspace - neuer Kernel



Problem: alter Userspace - neuer Kernel



Buffer overflow!

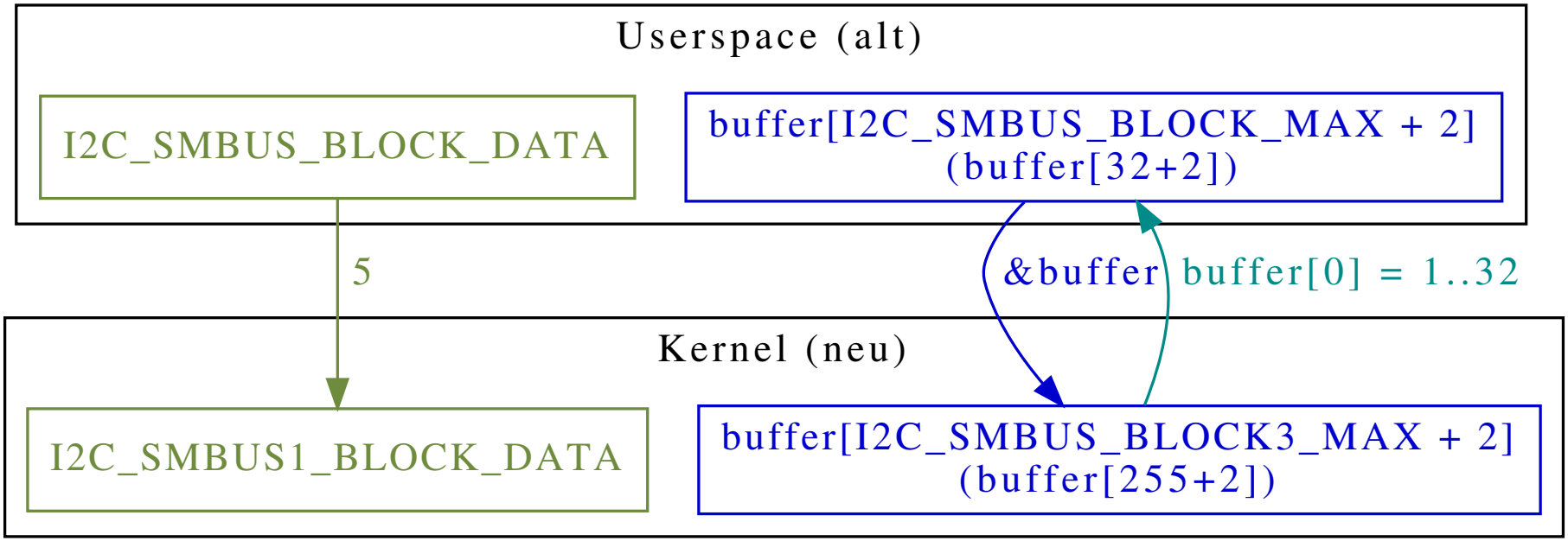
Proposal by Daniel Stodden, part 1

```
+#define I2C_SMBUS3_BLOCK_MAX    255
+
 union i2c_smbus_data {
         ...
-        __u8 block[I2C_SMBUS_BLOCK_MAX + 2];
+        __u8 block[I2C_SMBUS3_BLOCK_MAX + 2];
     /* block[0] is used for length */
     /* and one more for user-space compatibility */
 };
```

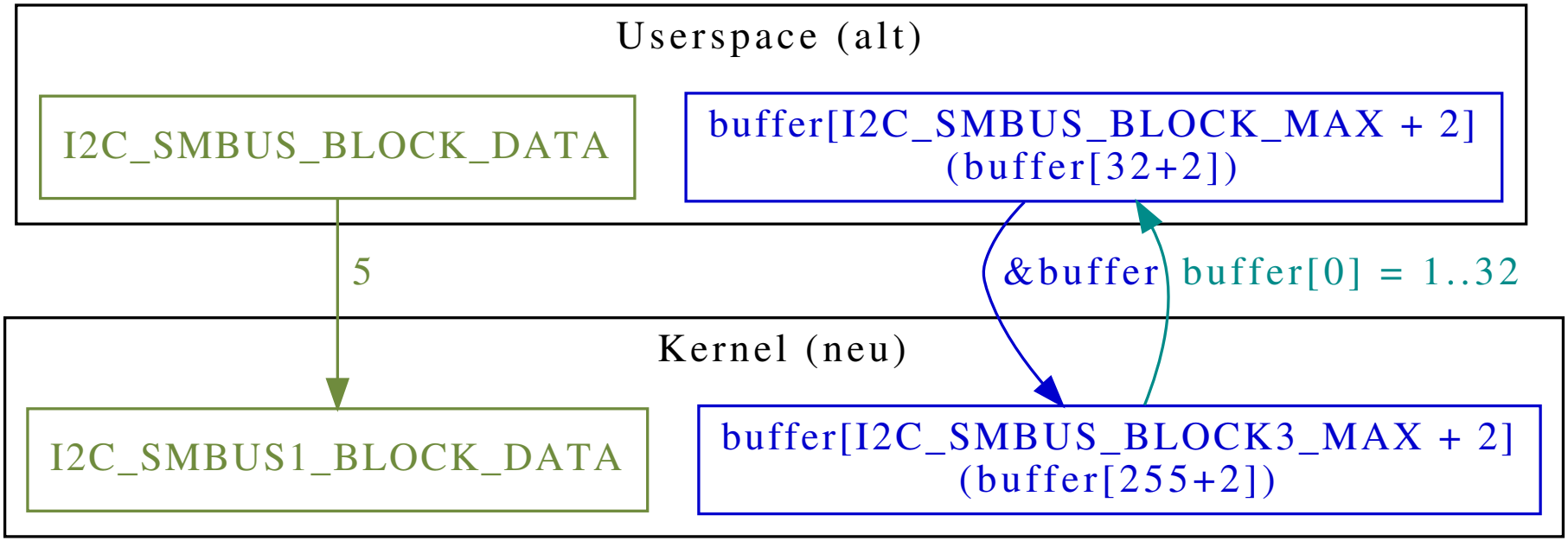
Proposal by Daniel Stodden, part 2

```
-#define I2C_SMBUS_BLOCK_DATA        5
+ /* Legacy 32 byte block limit */
+#define I2C_SMBUS1_BLOCK_DATA       5
+ /* Smbus 3.0, 255 byte block limit */
+#define I2C_SMBUS_BLOCK_DATA        9
```

Alter Userspace - neuer Kernel

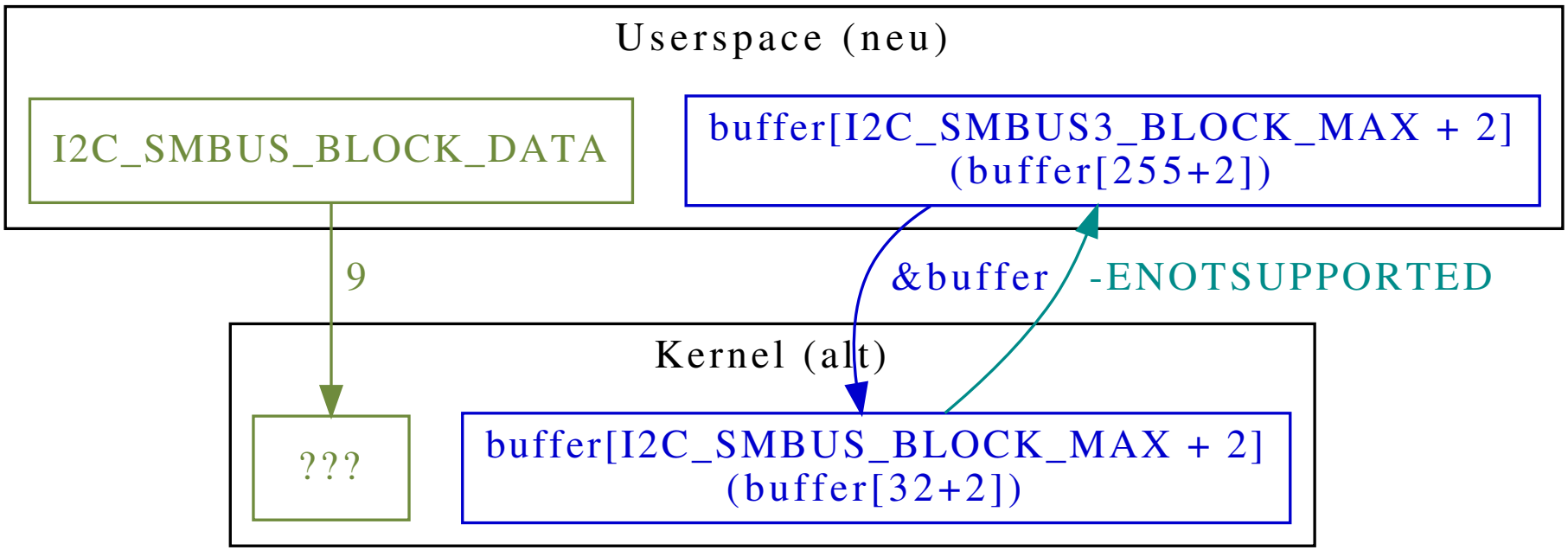


Alter Userspace - neuer Kernel

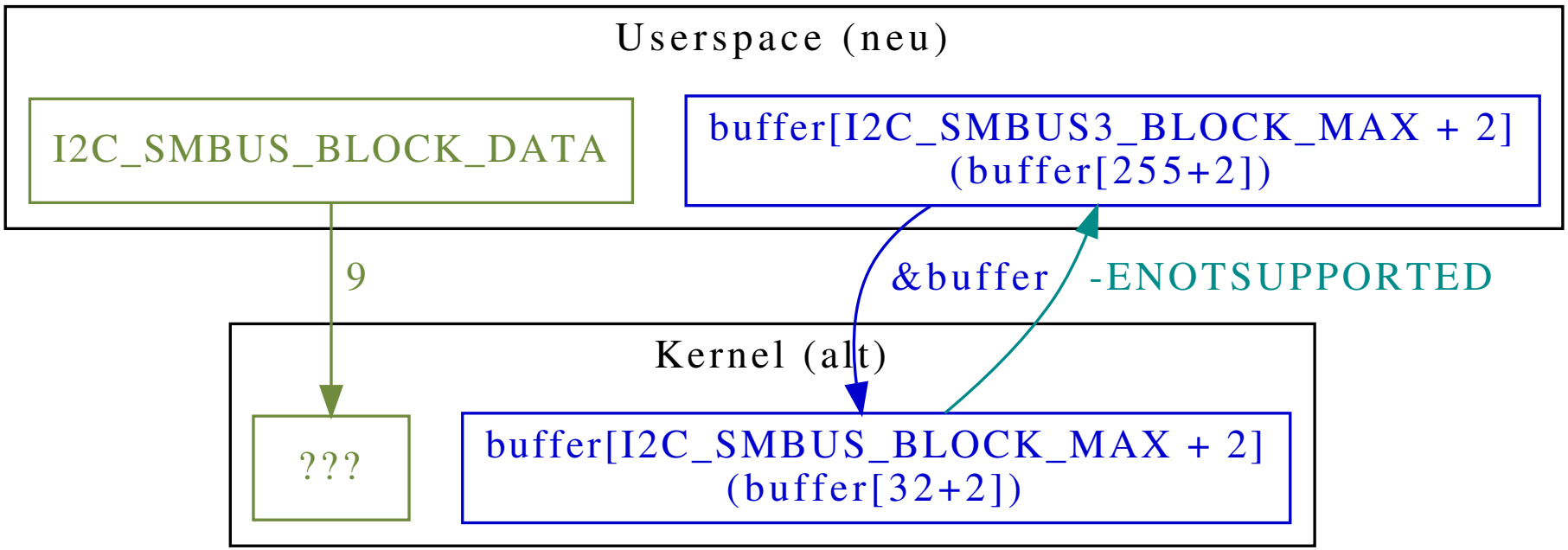


Geht!

Neuer Userspace - alter Kernel



Neuer Userspace - alter Kernel



Geht nicht!

Zwischenfazit

- neuer Code ist eh nötig
- dann aber:
  - Bestehendes nicht verändern
  - neue Features nur als Opt-In

Wie machen das die anderen?[3]

```
rfkill: make new event layout opt-in

Again new complaints surfaced that we had broken the ABI here,
although previously all the userspace tools had agreed that it
was their mistake and fixed it. Yet now there are cases (e.g.
RHEL) that want to run old userspace with newer kernels, and
thus are broken.

Since this is a bit of a whack-a-mole thing, change the whole
extensibility scheme of rfkill to no longer just rely on the
message lengths, but instead require userspace to opt in via a
new ioctl to a given maximum event size that it is willing to
understand.
```

[3] commit 54f586a9153201c6cff55e1f561990c78bd99aa7

Noch ein ioctl?

```
+#define I2C_SMBUS3_BLOCK_MAX   255
+
+union i2c_smbus3_data {
+       __u8 block[I2C_SMBUS3_BLOCK_MAX + 1];
+   /* block[0] is used for length */
+};

 #define I2C_SMBUS      0x0720  /* SMBus transfer */
+#define I2C_SMBUS3     0x0721  /* SMBus3 transfer */
```

Over-Engineered?

Eigentlich ist es nur:
Feature gegen Flag

Momentaner Ansatz, Teil 1

```
+ /* Legacy 32 byte block limit */
 #define I2C_SMBUS_BLOCK_DATA         5
+ /* Smbus 3.0, 255 byte block limit */
+#define I2C_SMBUS3_BLOCK_DATA        9
```

Momentaner Ansatz, Teil 2

```
+#define I2C_SMBUS3_BLOCK_MAX    255
+
 union i2c_smbus_data {
         ...
-        __u8 block[I2C_SMBUS_BLOCK_MAX + 2];
+        __u8 block[I2C_SMBUS3_BLOCK_MAX + 1];
     /* block[0] is used for length */
-    /* and one more for user-space compatibility */
 };
```

Bloß nicht nochmal!

```
-       __u8 block[I2C_SMBUS3_BLOCK_MAX + 1];
+       __u8 block[I2C_SMBUS3_BLOCK_MAX + 2];
    /* block[0] is used for length */
+   /* and one more to cover the full 8 bit range */
```

Warum nicht gleich so?

- erst sollten neue Transaktionstypen vermieden werden
- erst sollte Software nicht angepasst werden müssen
- die größere Datenstruktur sollte nicht aufgezwungen werden
- es gibt ein besseres Gefühl, wenn Alternativen auch durchdacht wurden

Und jetzt?

Es gibt nicht nur I2C_SMBUS_BLOCK_DATA

- `I2C_SMBUS_BLOCK_PROC_CALL`
- `I2C_SMBUS_I2C_BLOCK_DATA`
- `I2C_RDWR` ioctl
- und andere Nutzer von `I2C_M_RECV_LEN`

Arbeitsfelder

UAPI

- SMBus 3 Blockgröße und Typen defineren

Userspace

- Dokumentation
- libi2c updaten

Kernelspace

- neue Puffergröße als Standard etablieren
- nach Regressionen suchen
- die alten Limits aus den Controller-Treibern entfernen
- ggf. Geräte-spezifische Limits hinzufügen

Fazit 1

Opt-in ist das Mittel der Wahl

Fazit 2

Das erledigt sich nicht nebenbei

Danke für die Unterstützung!

Ende

Fragen? Anmerkungen?

Na klar doch...

- jetzt
- während der Konferenz
- wsa@kernel.org