Domänenorientierte Softwarearchitektur mit CÉU und RUST am Beispiel eines Heizungsgateways zur Fernüberwachung und Fernparametrisierung

Matthias Terber

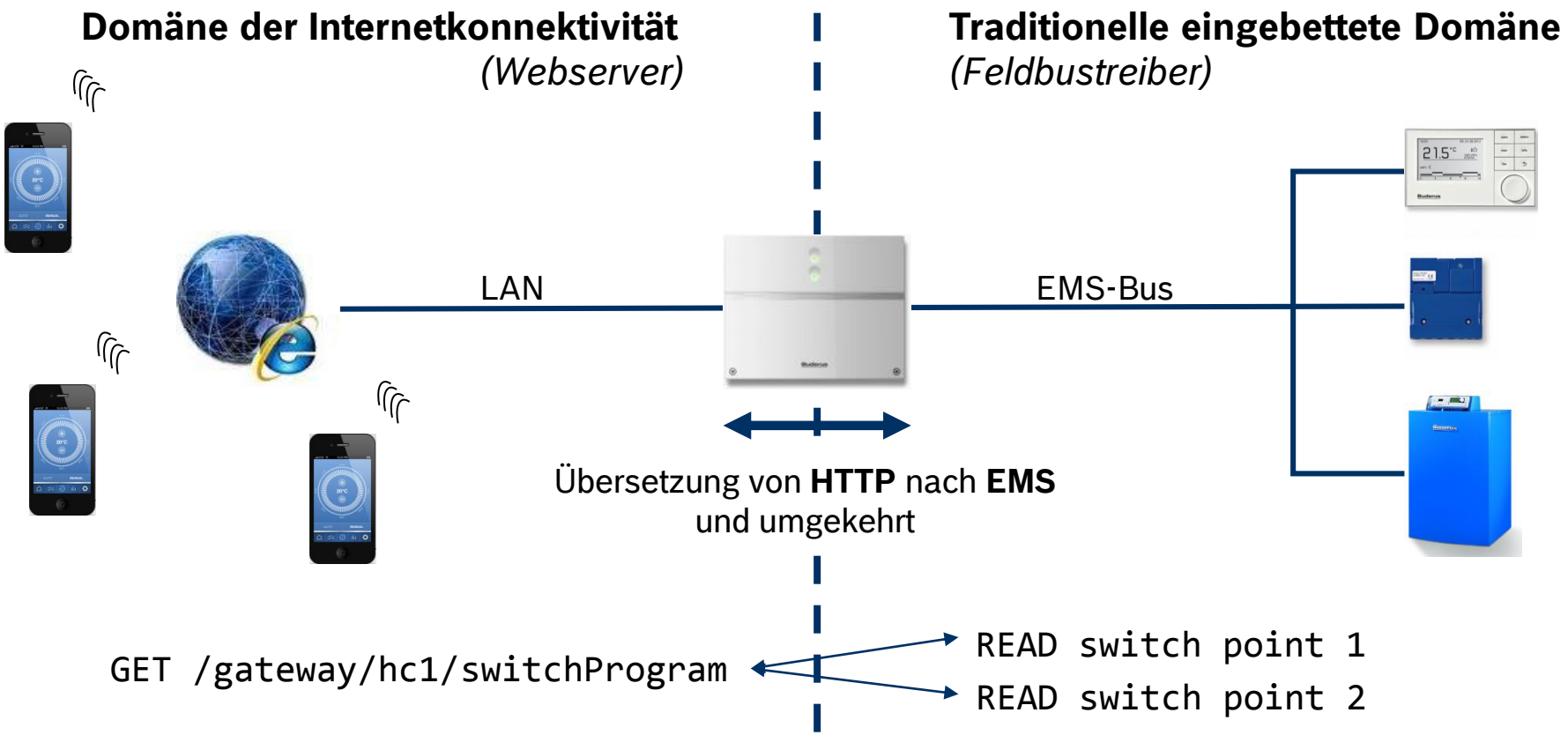
18. November 2016

BOSCH

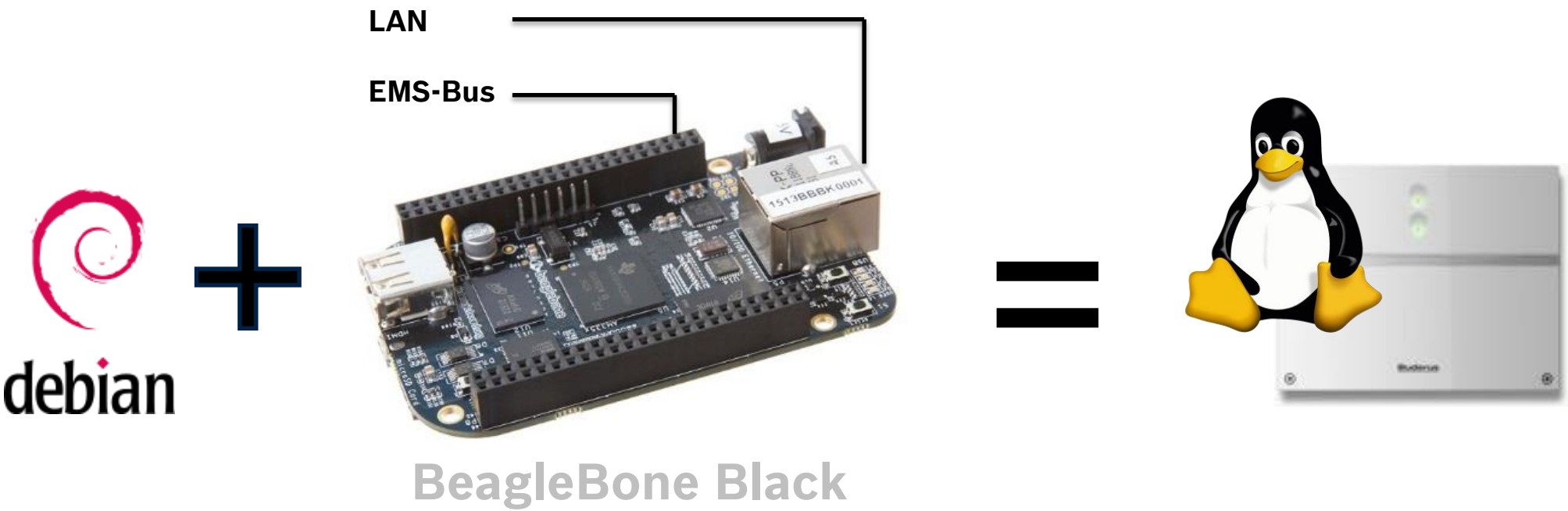
Agenda

1. Motivation
   - Gateway Systemüberblick
   - Ziel: Neuentwurf mit Linux
2. Domänenanalyse
   - Berechnungscharakteristiken
   - Berechnungsmodelle
3. Neuentwurf mit CÉU & RUST
   - Geschichtete Softwarearchitektur
   - Anwendungsbeispiele
4. Zusammenfassung

Gateway-Systemüberblick



Ziel: Neuentwurf mit Linux

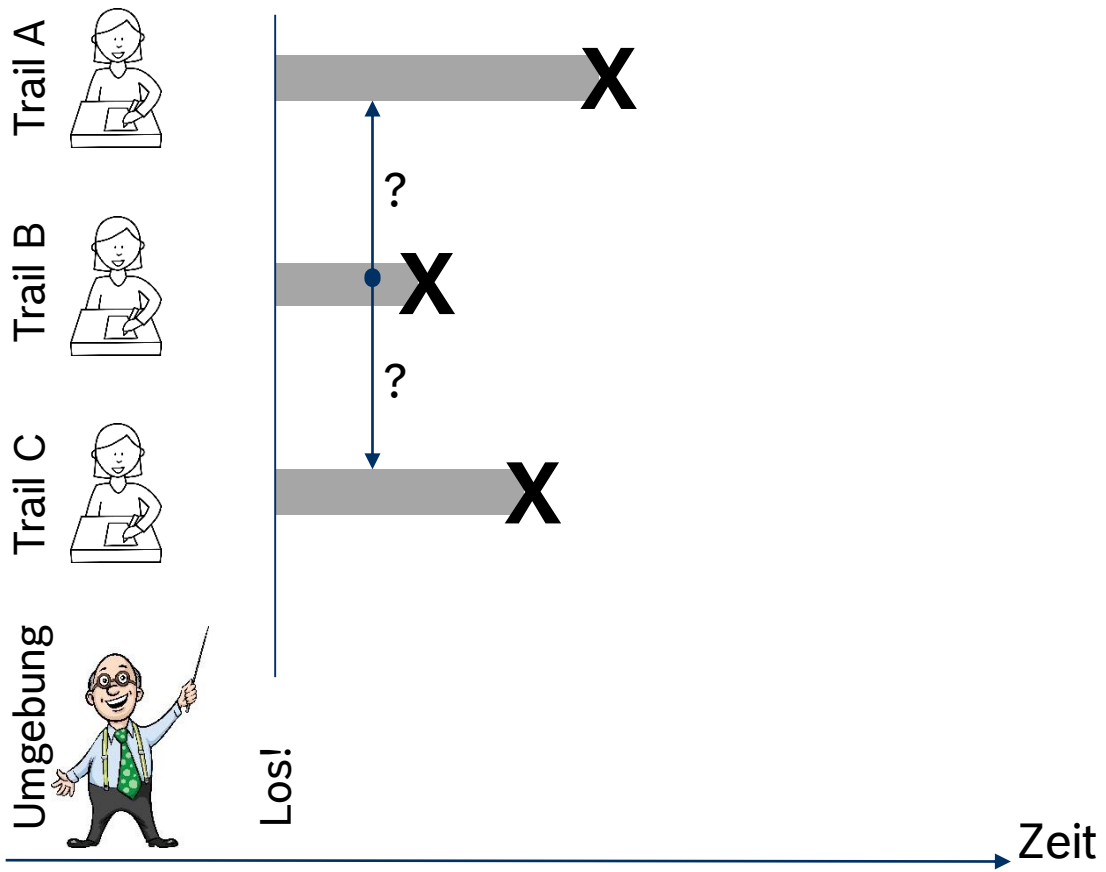


Berechnungscharakteristiken: Internet vs. Feldbus

- ✓ Unabhängig + zustandslos
- ✓ Verschachtelt + nicht-deterministisch
- ✓ Langlebig + nicht-echtzeitkritisch

- ✓ Abhängig + zustandsgebunden
- ✓ Streng sequentiell + deterministisch
- ✓ Kurzlebig + echtzeitkritisch

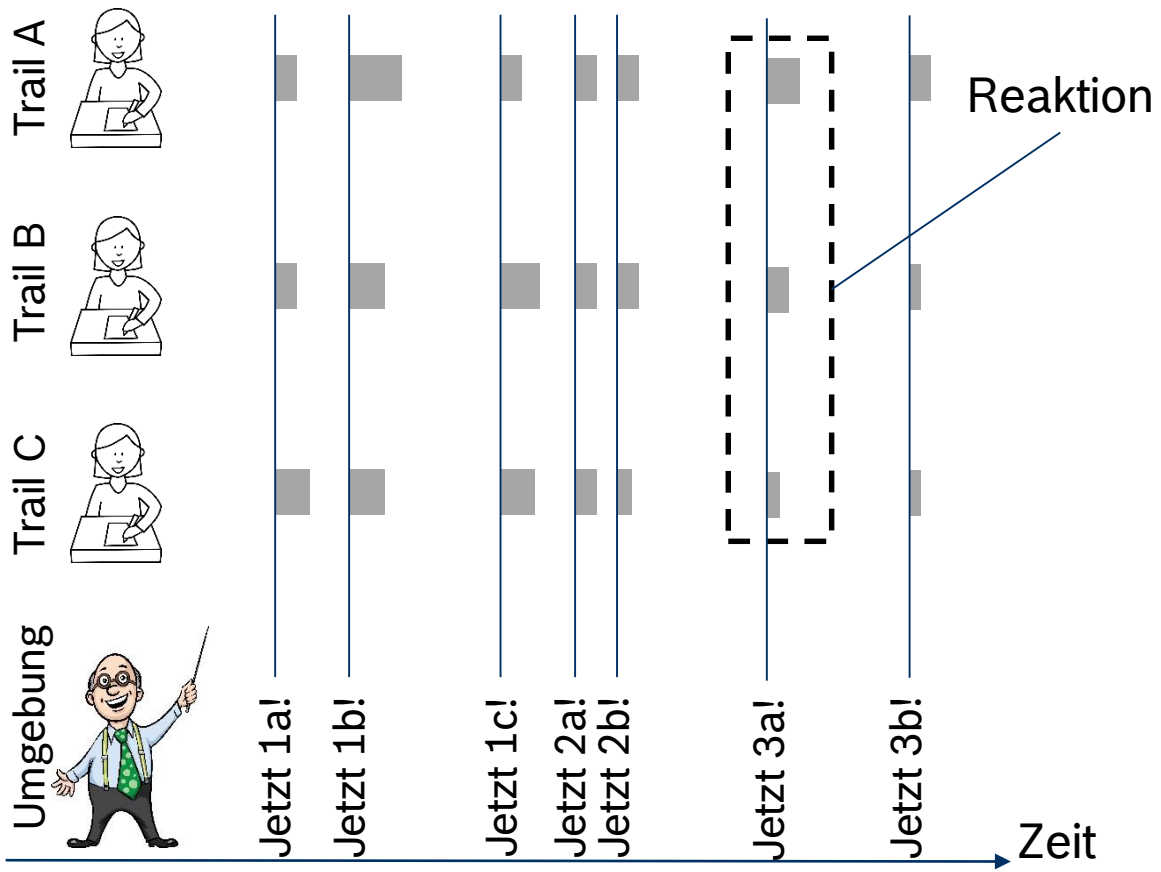
Berechnungsmodell: Asynchrone Ausführung



- Ausführung bestimmt durch: **Trails**
- Trails
  - Laufen unabhängig
  - Haben keine gemeinsame Sicht auf die Umgebung oder untereinander
  - Bestimmen selbst die Zeitpunkte für Synchronisation oder Kommunikation

- **Fokus liegt auf den Trails!**

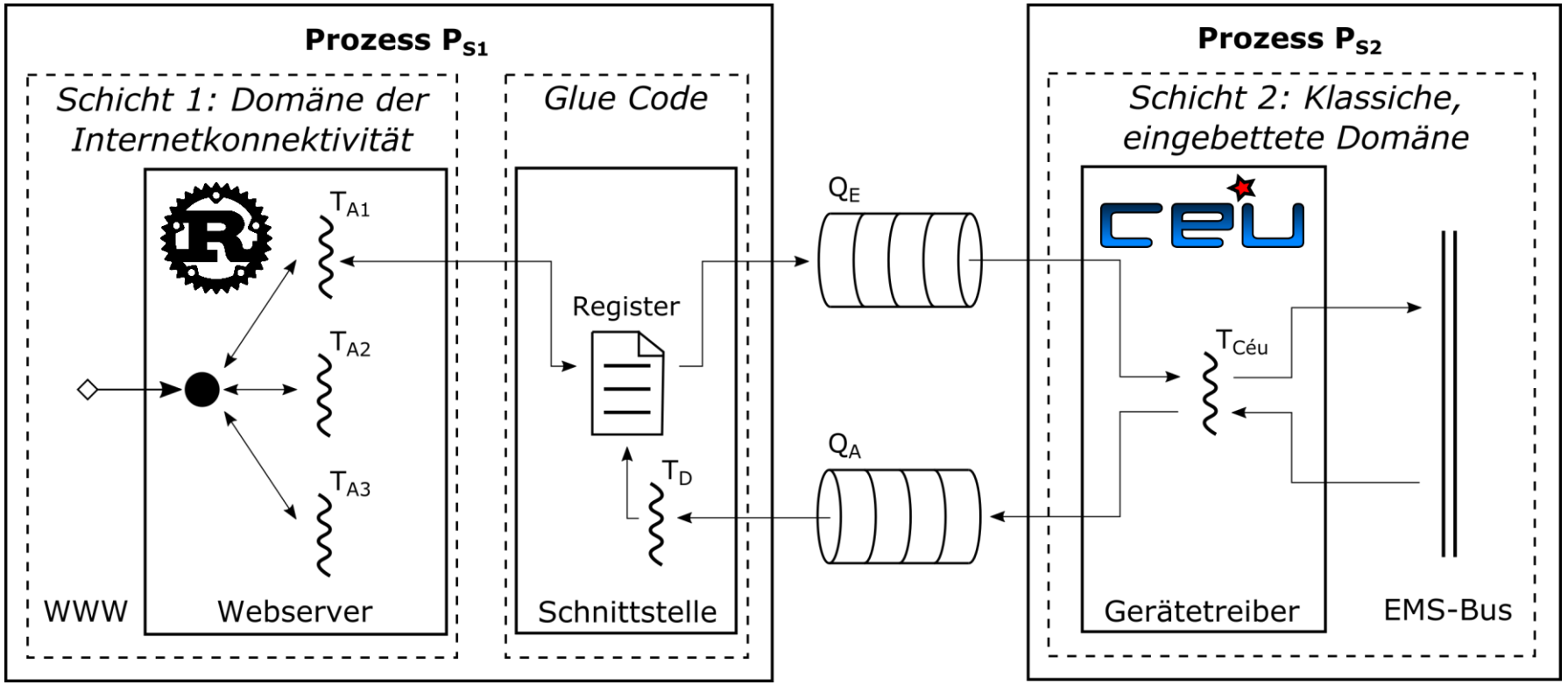
Berechnungsmodell: Synchrone Ausführung



- Ausführung bestimmt durch: **Umgebung**
- Umgebung
  - Initiiert die Ausführung
    - ereignisgesteuert
    - zeitgesteuert
  - Bestimmt die Geschwindigkeit
- Trails
  - Laufen im Gleichschritt
  - Sind kontinuierlich synchronisiert mit ihrer Umgebung und untereinander

- **Fokus liegt auf der Umgebung!**

Geschichtete Softwarearchitektur



Anwendungsbeispiel: Webserver

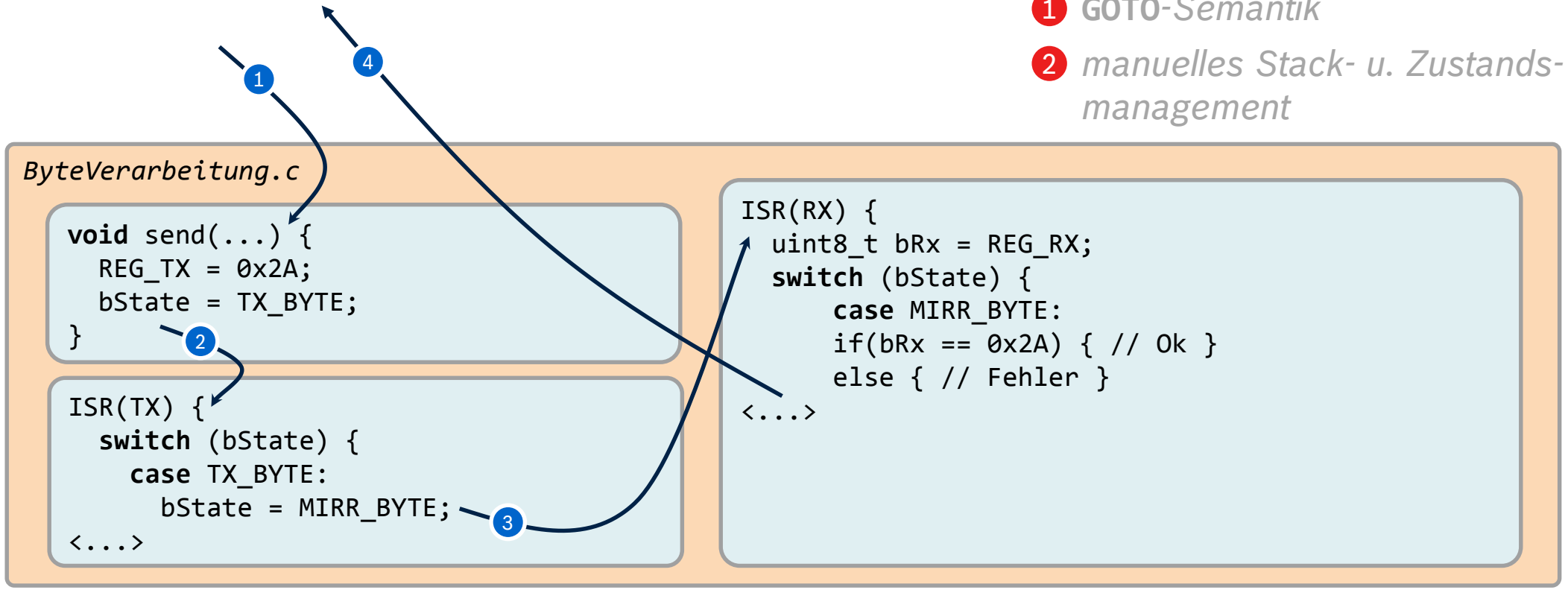
```
01: fn main() {
02:     let port = “443”;
03:     let addr = “0.0.0.0”.to_string() + “:” + port;
04:     Iron::new(|req: &mut Request| {
05:         match req.method {
06:             Method::Get => { handle_get(req) },
07:             Method::Put => { handle_put(req) },
08:             _ => Ok(Response::with(status::MethodNotAllowed))
09:         }
10:     }).https(addr,
11:         PathBuf::from("/etc/ssl/certs/ssl-cert-snakeoil.pem"),
12:         PathBuf::from("/etc/ssl/private/ssl-cert-snakeoil.key")).unwrap();
13: }
```

1. *komfortable Verarbeitung von Zeichenketten*
2. *Ausführung in eigenem Thread pro Anfrage*
3. *Verschlüsselung*

Traditionell: Bytespiegelung in C

Zustand ist explizit,
Kontrollfluss implizit!

1. GOTO-*Semantik*
2. *manuelles Stack- u. Zustands-management*



Anwendungsbeispiel: Bytespiegelung

```
01: input u8 BYTE_RX;  // Eingangsereignis
02: output u8 BYTE_TX; // Ausgangsereignis
03: var u8 b = 0x2A;
04: var int ret = emit BYTE_TX => b;
05: if ret != 0 then
06:     escape -1;
07: end
08: par/or do //--- Trail 1 ---//
09:    var u8 mirror = await BYTE_RX;
10:    if mirror != b then
11:        escape -2;
12:    end
13: with      //--- Trail 2 ---//
14:    await 42ms;
15:    escape -3;
16: end
17: escape 0;
```

Kontrollfluss ist explizit,
Zustand implizit!

Essentielle Erweiterungen

1. *Warten auf Ereignisse (“blockierend”)*
2. *Parallelkomposition*
3. *Orthogonaler Abbruchmechanismus*

➔ “**S**tructured **S**ynchronous **R**eactive **P**rogramming”

Zusammenfassung

Internet ←→ eingebettete Domäne

- ✓ Geschichtete Softwarearchitektur
- ✓ Autarke Prozesse
- ✓ Schlanke Schnittstelle in C

Warum?

- ✓ Lose Kopplung / Unabhängigkeit
  - Robustheit
  - Einsatz geeigneter Sprachen
  - Einsatz existierender Lösungen
  - Selektive Priorisierung
- ✓ Ausführung auf verschiedenen Kernen
  - Erfüllung von Echtzeitanforderungen
- ✓ Einfacheres Testen und Bewerten

- ✓ Thread-basierte Nebenläufigkeit
- ✓ Hohes Abstraktionslevel + Effizienz
- ✓ Low-level Hardwarezugriff
- ✓ Webentwicklung komfortabler und sicherer

- ✓ Ereignisbehandlung + Zeit
- ✓ Strukturierte Nebenläufigkeit
- ✓ Expliziter, linearer Kontrollfluss
- ✓ Deterministisches Verhalten

Vielen Dank für Ihre Aufmerksamkeit!

Sind noch Fragen offen?