Concurrent Distributed Systems

Zusammenfassung

Joel von Rotz / Quelldateien

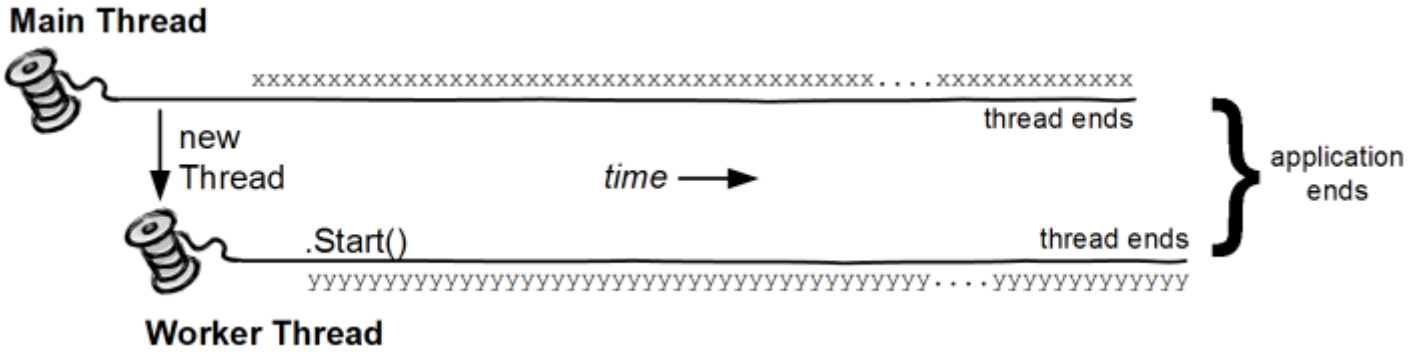
Table of Contents

C#/.NET

Garbage Collector

C# verfügt über einen Garbage Collector, welcher nicht verwendete (& referenzlose) Objekte automatisch löscht und somit Speicher freigibt.

Threads `System.Threading`



- Erhält eigenen Stack für lokale Variablen
- Mehrere Threads können auf gemeinsame Variablen zugreifen
  - **Deadlock** und **Race Condition** beachten!

Erstellen

Erstellen mit `new Thread(...)` auf zwei Varianten: `ThreadStart` & `ParameterizedThreadStart`

```
Thread t  = new Thread(new ThreadStart(f1));
Thread t2 = new Thread(new ParameterizedThreadStart(f2));

static void f1(void) {/* ... */};
static void f2(object value){/* ... */};
```

Starten

Starten mit `.start(param)`:

```
t.start();
t2.start(true);
```

Priorität konfigurieren

Priorität setzen mit `.Priority`

```
t.Priority = ThreadPriority.Lowest;
```

`Highest` (4), `AboveNormal` (3), `Normal` (2), `BelowNormal` (1), `Lowest` (0)

Beenden

1. Methode ohne Felder beendet
2. Auftreten einer Exception

```
static void Main() {
  try {
    new Thread(Go).Start();
  } catch(Exception ex) {
    Console.Writeline("Exception!");
  }
}
static void Go() {
  throw null; // exception will NOT be caught
  Console.Writeline("uups");
}
```

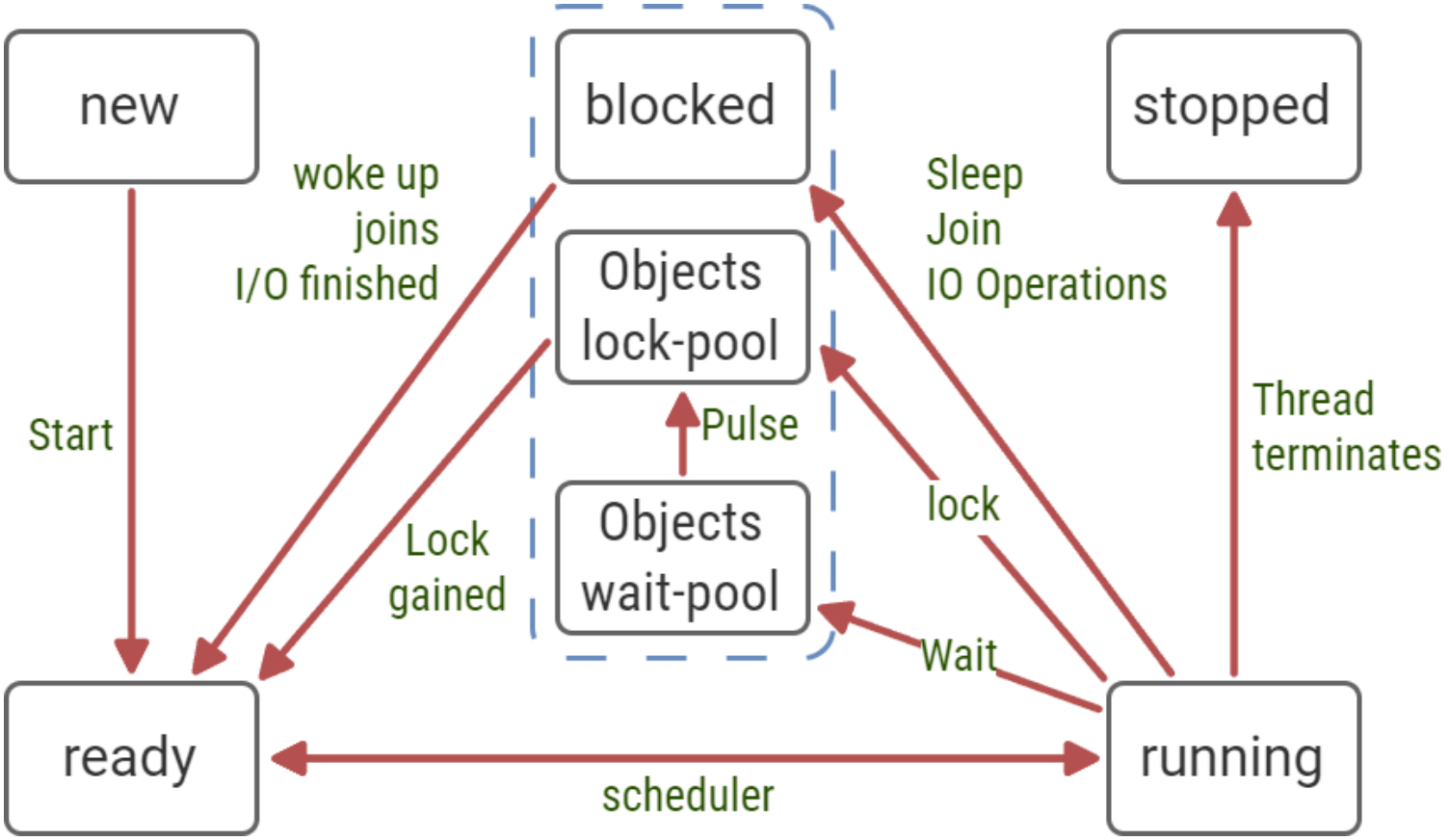
Thread sind (fast) isoliert

Beispiel oben `try/catch` ist nur im Bezug zur Erstellung des Threads brauchbar, denn es wird die Exception **NICHT** abfangen, da diese **IM** Thread ausgelöst wurde.

Die Funktion `.Abort()` „killed“ den Thread à la:

Scenario You want to turn off your computer
Solution You strap dynamite to your computer, light it, and run.

Lebenszyklen



Legende

`new` Thread-Objekt erstellt, aber noch nicht gestartet

`ready` gestartet, lokaler Speicher (Stack) zugeteilt, wartet auf Zuweisung des Prozessors

`running` Thread läuft

`blocked` Thread wartet bis eine Bedingung erfüllt wird **/** Aufruf einer Betriebssystemroutine, z.B. File-Operationen

`stopped` Thread existiert nicht mehr, Objekt schon.

`Object lock-pool` Bei der Verwendung vom *lock*-Konstrukt, erhält der Threads Lebenszyklus diesen zusätzlichen Zustand. Jedes Object hat genau einen lock-pool.

`Object wait-pool` Menge von Threads, die vom Scheduler unterbrochen wurden und auf ein Ereignis warten, um fortgesetzt wreden zu können.

⚠ Wichtig

Der Objects lock-pool und der Objects wait-pool müssen zum gleichen Objekt gehören.

```
object synch = new object();

lock (synch) {
  Monitor.Wait(synch);
}
```

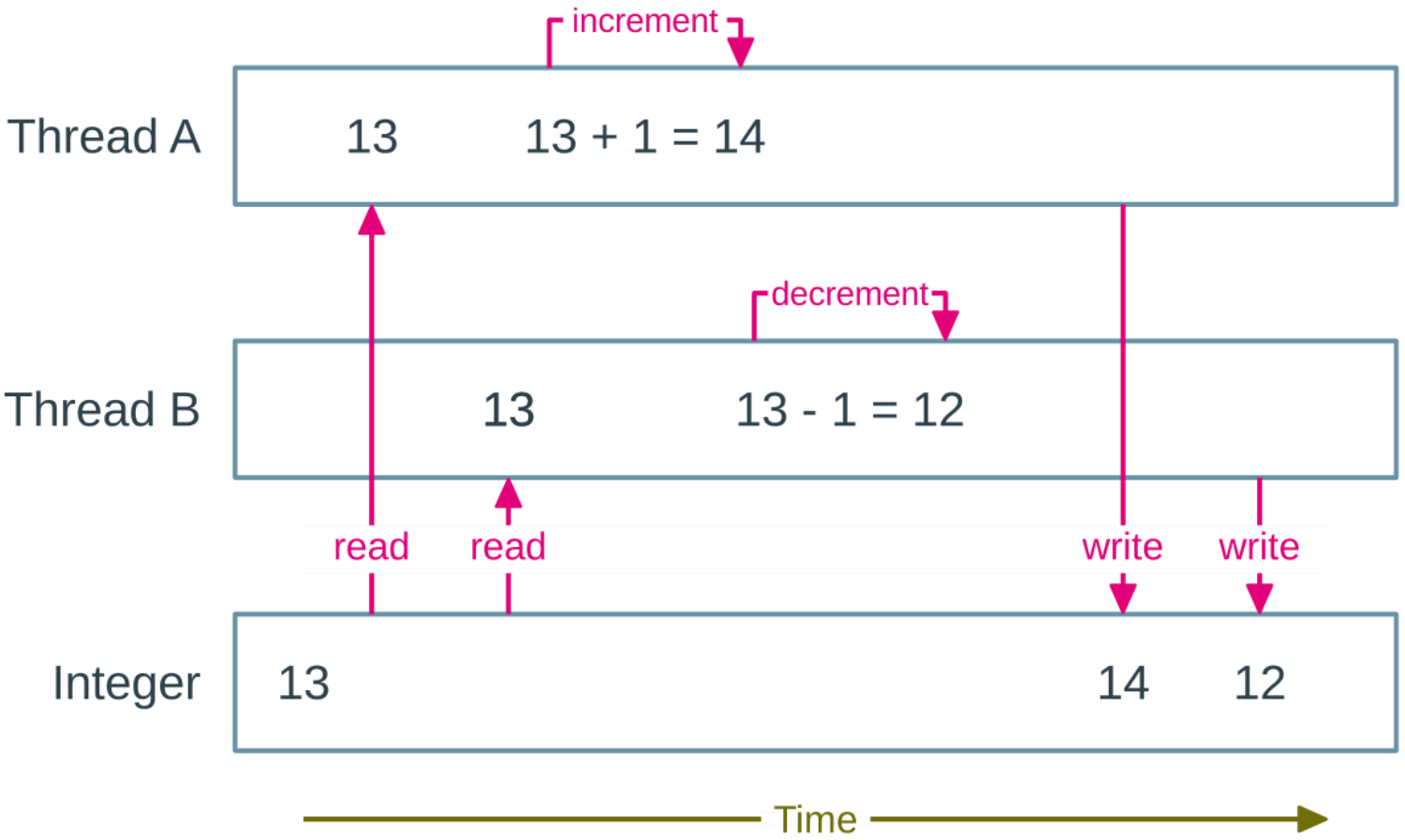
Bei nicht Einhaltung wird die Exception ausgelöst:

`System.Threading.SynchronizationLockException`

Thread Synchronisation

Race Conditions

…ist eine Konstellation, in denen das Ergebnis einer Operation vom zeitlichen Verhalten bestimmter Einzeloperationen abhängt.

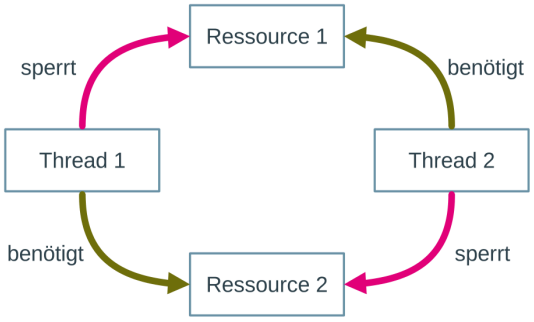


Gute Lösung dazu muss vier Bedingungen erfüllen:

1. Nur ein Thread in kritischen Abschnitten
2. Keine Annahmen zur zugrundeliegenden Hardware treffen.
3. Ein Thread darf andere Threads nicht blockieren, ausser in kritischen Bereichen
4. Thread sollte nicht unendlich lange warten, bis dieser in den kritischen Bereich eintreten kann.

Deadlocks

Entsteht, wenn Threads auf Resourcen warten, welche sie gegenseitig sperren und somit kein Thread sich befreien kann.



Join-Funktion

Mit `.Join()` können Threads auf den Abschluss eines anderen Threads warten (z.B. Öffnung einer Datei bevor Schreibung). Beim Aufruf wird der aktuelle Thread blockiert, bis die Join-Kondition erreicht wurde.

```
// in Thread t
t2.Join(); // wait for Thread t2's completion
```

`lock` Konstrukt

Mit Locks können Threads Code Bereiche reservieren. Dies wird mit…

```
lock(<object>) {
  /* do stuff with <object> or use it as a flag*/
}
```

… gemacht. Als `<object>` kann eine Flag (z.B. ein `object` Objekt) oder eine Ressource (z.B. File Objekt) verwendet werden.

`lock` ist die Kurzform für…

```
Monitor.Enter(<object>);
try{
  /* critical section */
}
finally { Monitor.Exit(<object>) }
```

EventWaitHandle

Threads warten an einem inaktiven Event-Objekt, bis dieses aktiv (frei) geschaltet wird. Es gibt zwei Arten:

`AutoResetEvent` Threadaktivierung durch das Event setzt das Eventsignal automatisch zurück zu *inaktiv* (nur `.Set()`)

`ManualResetEvent` Eventsignal muss manuel zurückgesetzt werden (`.Set()` → dann `.Reset()`)

```
private static
  EventWaitHandle wh = new AutoResetEvent(false);

static void Main() {
  new Thread(Waiter).Start();
  Thread.Sleep(1000);
  wh.Set();
}

static void Waiter() {
  Console.Writeline("Waiting ... ");
  wh.WaitOne();
}
```

Pulse/Wait

Wenn der Zugang zu einem kritischen Abschnitt nur von bestimmten Bedingungen oder Zuständen abhängt, so reicht das Konzept der einfachen Synchronisation mit *lock* nicht aus

```
/* Monitor.<func> */
bool Wait(object obj);
bool Wait(object obj, int timeout_ms);
bool Wait(object obj, TimeSpan timeout);

void Pulse(object obj);
void PulseAll(object obj);
```

Es führen zwei Wege aus dem Warte-Zustand:

1. Anderer Thread signalisiert Zustandswechsel
2. Angegebene Zeit ist abgelaufen (dabei wird der aktuelle Lock wieder genommen und `Wait` gibt `false` zurück)

⚠ Nur in kritischen Bereichen

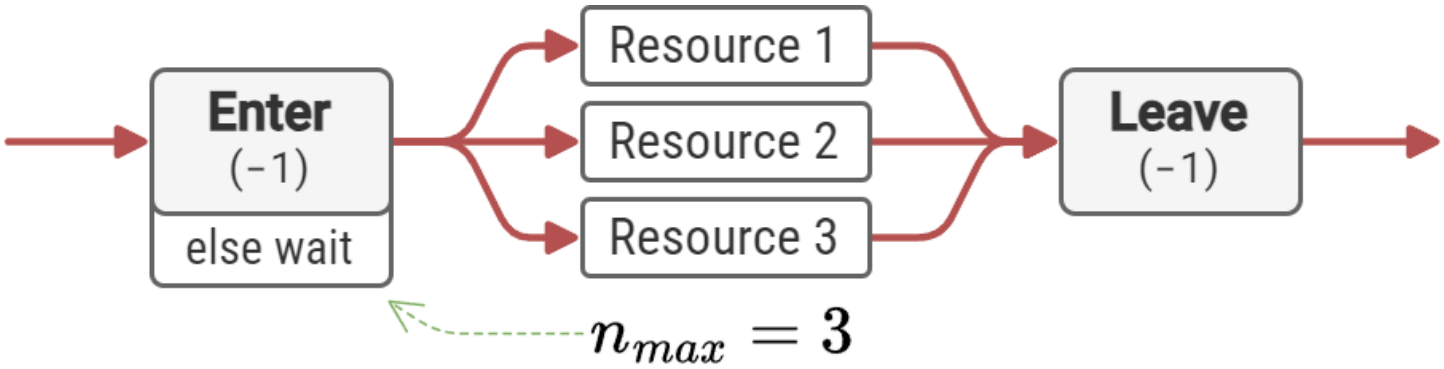
`Wait`, `Pulse` und `PulseAll` dürfen nur innerhalb eines kritischen Bereichs ausgeführt werden!

Ruft ein Thread `Wait` auf, wird der Lock für diesen Abschnitt freigeben! Nach Erhalt eines Pulses wartet der Thread auf den Lock seines Abschnittest.

```
private static object synch = new object();

static void Main() {
  new Thread(Waiter).Start();
  Thread.Sleep(1000);
  lock (synch) {
    Monitor.Pulse(synch);
  }
}

static void Waiter() {
  lock (synch) {
    Console.WriteLine("Waiting ... ");
    if (Monitor.Wait(synch, 1000))
      Console.WriteLine("Notified");
    else
      Console.WriteLine("Timeout");
  }
}
```

Semaphore

(Signalmasten, Leuchttürme)



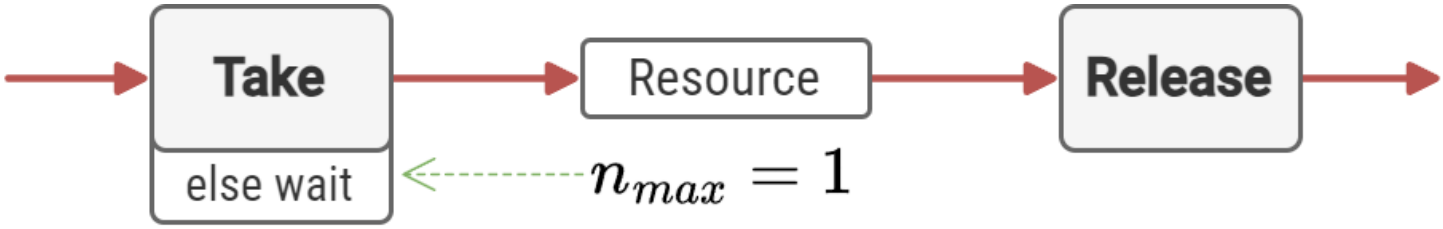
Mit Semaphoren können an vorbestimmte Anzahl *Teilnehmer* Ressourcen erlaubt werden, bevor der Ressourcen-Zugang gesperrt wird.

`.WaitOne()` **($sema.P()$)** Eintritt (Passieren) in einen synchronisierten Bereich, wobei mitgezählt wird, der wievielte Bereich es ist.

`.Release()` **($sema.V()$)** Verlassen (Freigeben) eines synchronisierten Bereichs, wobei mitgezählt wird, wie oft der Bereich verlassen wird.

```
// (initialCount: 3, maximumCount: 3)
private static Semaphore s = new Semaphore(3, 3);

static void Main() {
  for (int i = 0; i < 10; i++) {
    new Thread(Go).Start(i);
  }
}

static void Go(object number) {
  while (true) {
    s.WaitOne(); // thread X waits

    // thread X enters critical section
    Thread.Sleep(1000); // entries limited to 3

    s.Release(); // Thread X leaves
  }
}
```

Mutex



Nützlich, wenn eine Ressource (z.B. Ethernet-Schnittstelle) von mehreren Threads verwendet werden möchte.

> **Freigabe-Scope**
>
> Im Vergleich zu Semaphoren, welches erlaubt von anderen Aktivitätsträgern freizugeben, muss die Mutex vom Mutex-Besitzer freigegeben werden!

```
private static      //   (initially owned, name)
  Mutex mu = new Mutex(false, "CoolName");

static void Main() {
  if (!mu.WaitOne(TimeSpan.FromSeconds(5),false)) {
    Console.WriteLine("Another app instance exists");
    return;
  } try {
    Console.WriteLine("Running - Enter to exit");
    Console.ReadLine();
  } finally {
    mu.ReleaseMutex();
  }
}
```

Streams

Streams dienen dazu, drei elementare Operationen ausführen zu können:

Schreiben Dateninformationen müssen in einem Stream geschrieben werden. Das Format hängt vom Stream ab.

Lesen Aus dem Datenstrom muss gelesen werden, ansonsten könnte man die Daten nicht weiterverarbeiten.

Wahlfreien Zugriff Nicht immer ist es erforderlich, den Datenstrom vom ersten bis zum letzten Byte auszuwerten. Manchmal reicht es, erst ab einer bestimmten Position zu lesen.

C# implementiert Stream-Klassen mit sequentielle Ein-/Ausgabe auf verschiedene Datentypen:

Zeichenorientiert (`StreamReader/-Writer`, `StringReader /-Writer`) mit Wandlung zwischen interner Binärdarstellung und externer Textdarstellung. Grundlage ist die byteorientierte Ein- und Ausgabe mit den Klassen `TextReader` und `TextWriter`

Binär (`BinaryReader/-Writer`, Unterklassen von `Stream`) ohne Wandlung der Binärdarstellung

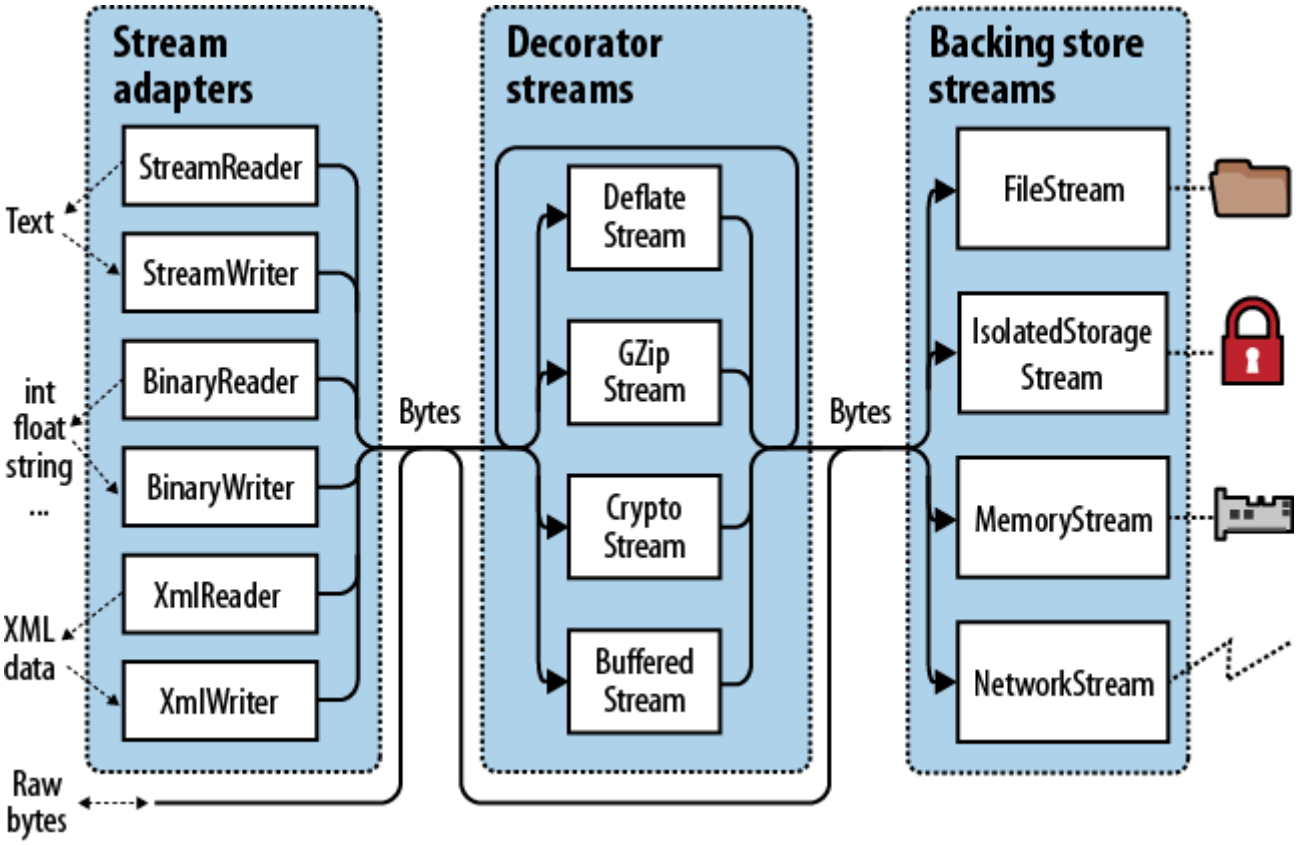
Stream Architektur

.NET-Stream-Architektur konzentriert sich auf drei Konzepte:

Adapter formen Daten (Strings, elementare Datentypen, etc.) aus Programmen um. (siehe Adapter Entwurfsmusters)

Dekorator fügen neue Eigenschaften zu dem Stream hinzu. (siehe Dekorator Entwurfsmusters)

Sicherungsspeicher ist ein Speichermedium, wie etwa ein Datenträger oder Arbeitsspeicher.



Lesen

Lesen aus einem Netzwerk TCP-Socket (`SR` = `StreamReader`)::

```
TcpClient client = new TcpClient("192.53.1.103",13);
SR inStream = new SR(client.GetStream());
Console.WriteLine(inStream.ReadLine());
client.Close();
```

Lesen aus einer Datei (`SR` = `StreamReader`):

```
try {
  using (SR sr = new SR("t.txt")) {
  string line;
    while ((line = sr.ReadLine()) ≠ null) {
      Console.WriteLine(line);
    }
  }
} catch (Exception e) {
  Console.WriteLine(e);
}
```

Lesen aus einer Datei mit einem Pass-Through-Stream:

```
Stream stm = new FileStream("Daten.txt",
                    FileMode.Open,
                    FileAccess.Read);
ICryptoTransform ict = new ToBase64Transform();
CryptoStream cs = new CryptoStream(stm,
                      ict,
                      CryptoStreamMode.Read);
TextReader tr = new StreamReader(cs);
string s = tr.ReadToEnd();
Console.WriteLine(s);
```

Schreiben

Lesen von einer Tastatur und Schreiben auf den Bildschirm:

```
string line;
Console.Write("Bitte Eingabe: ");
while ((line = Console.ReadLine()) ≠ null) {
  Console.WriteLine("Eingabe war: " + line);
  Console.Write("Bitte Eingabe: ");
}
```

Schreiben in eine Daten mit implizitem FileStream:

```
try {
  using (StreamWriter sw = new StreamWriter("Daten.txt")) {
    string[] text = { "Titel", "Köln", "4711" };
for (int i = 0; i < text.Length; i++)
sw.WriteLine(text[i]);
}
Console.WriteLine("fertig.");
}
catch (Exception e) { Console.WriteLine(e); }
```

```
StreamWriter sw = new StreamWriter("Daten.txt")
// equals to
FileStream fs = new FileStream("Daten.txt",
                      FileMode.Create);
StreamWriter sw = new StreamWriter(fs);
```

Socket Kommunikation `System.Net.Sockets.Socket`

Sockets werden für *Interprozesskommunikation* verwendet, also zwischen zwei oder mehrere Prozesse (z.B. Applikation). Damit zwei Prozesse sich verstehen, müssen beide die selbe Sprache (Protokoll) sprechen: *TCP/IP*, *UDP*, *Datagram-Sockets*, *Multicast-Sockets*, etc.

Sockets können…

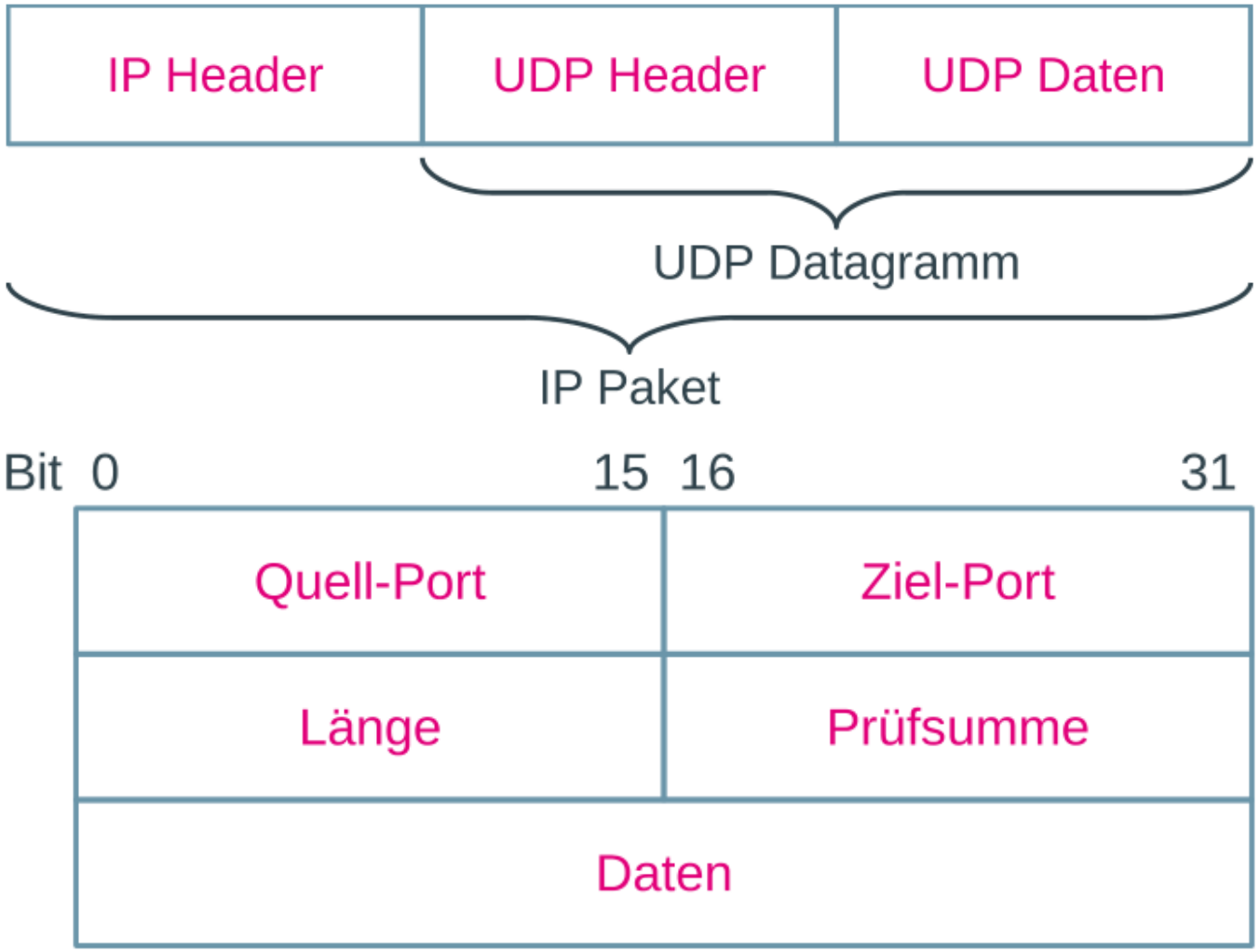
- …Einen Port binden
- …An einem Port auf Verbindungsanfragen hören
- …Verbindung zu entfernten Prozess aufbauen
- …Verbindungsanfragen akzeptieren
- …Daten an entfernten Prozess senden

UDP Protokoll

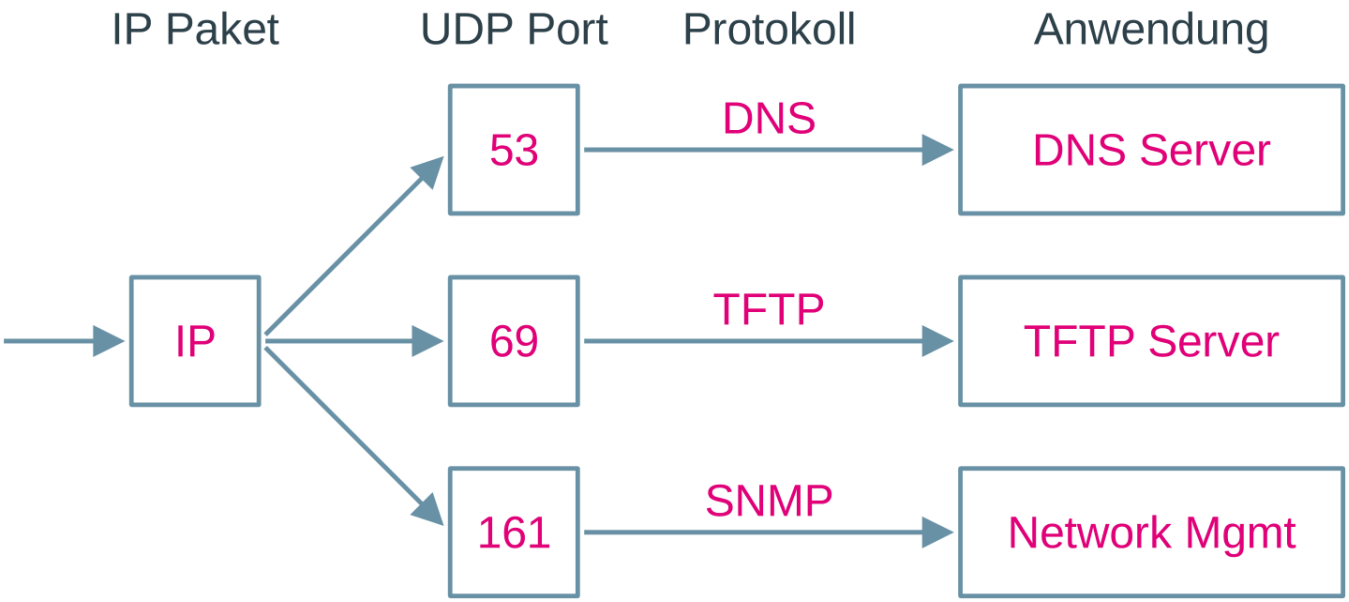
User Data Protocol

Ermöglicht das Senden von gekapselte rohe IP Datagramme zu übertragen, ohne Verbindungsaufbau.

⇒ *Verbindungslos* bedeutet keine Garantie, dass das gesendete Paket beim Empfänger ankommen.



UDP Header besteht aus 8 Byte. Die *Länge* entspricht Header Bytes + Daten Bytes. Die Prüfsumme wird über das gesamte Frame berechnet (IP Paket).



Der Ziel-Port bestimmt, für welche Anwendung ein Datenpaket bestimmt ist.

TODO Add Code Snippets here?

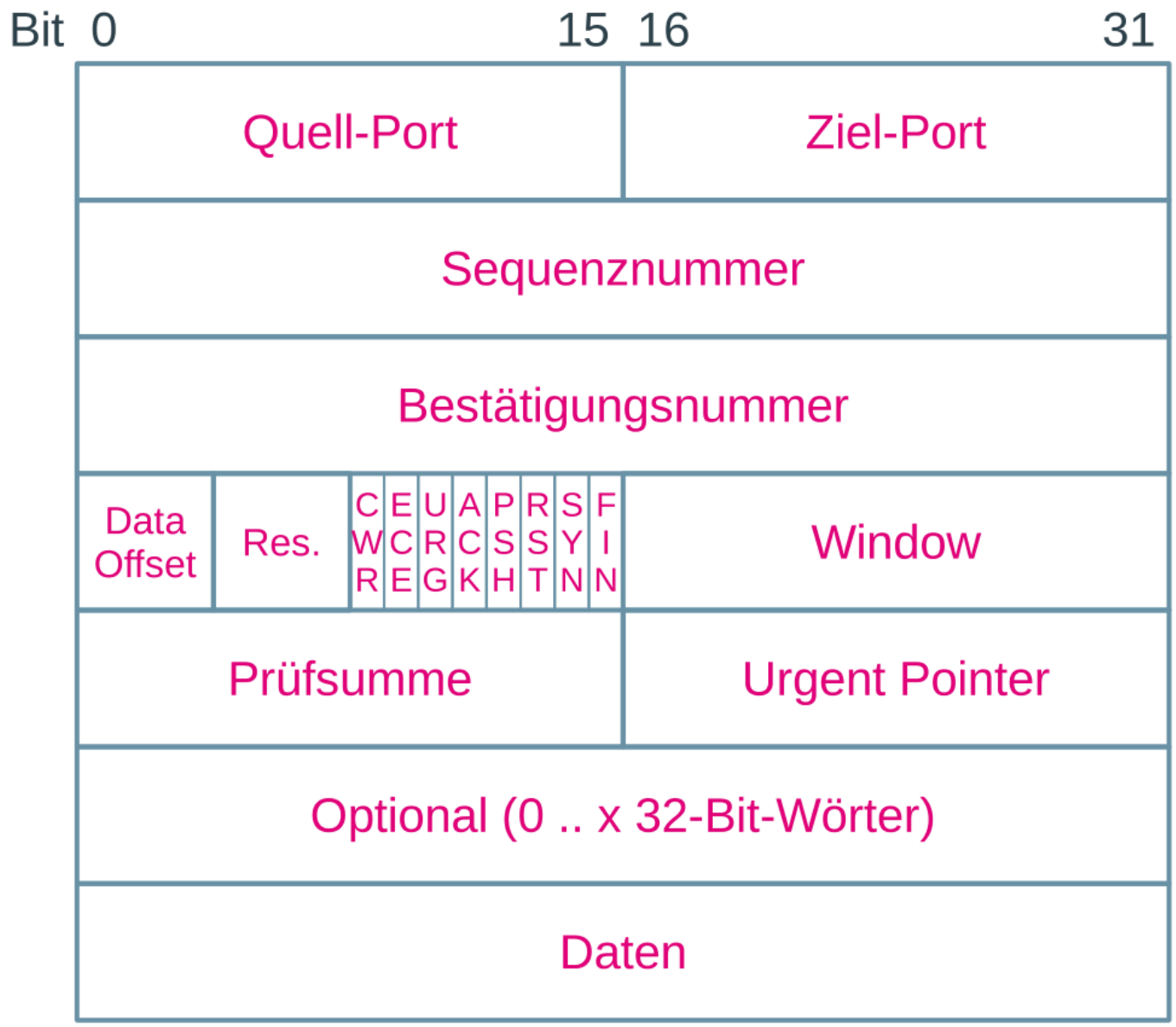
TCP Protokoll

Transmission Control Protocol

Datagram ist ähnlich wie bei UDP.

IP sorgt dafür, dass die Pakete von Knoten zu Knoten gelangen; **TCP** behandelt den Inhalt der Pakete und korrigiert dies (durch erneutes Senden)

TCP kann als End-to-End Verbindung in Vollduplex betrachtet werden → Möglich mit separierten Sende- & Empfangs-Counter.



Wichtige Merkmale:

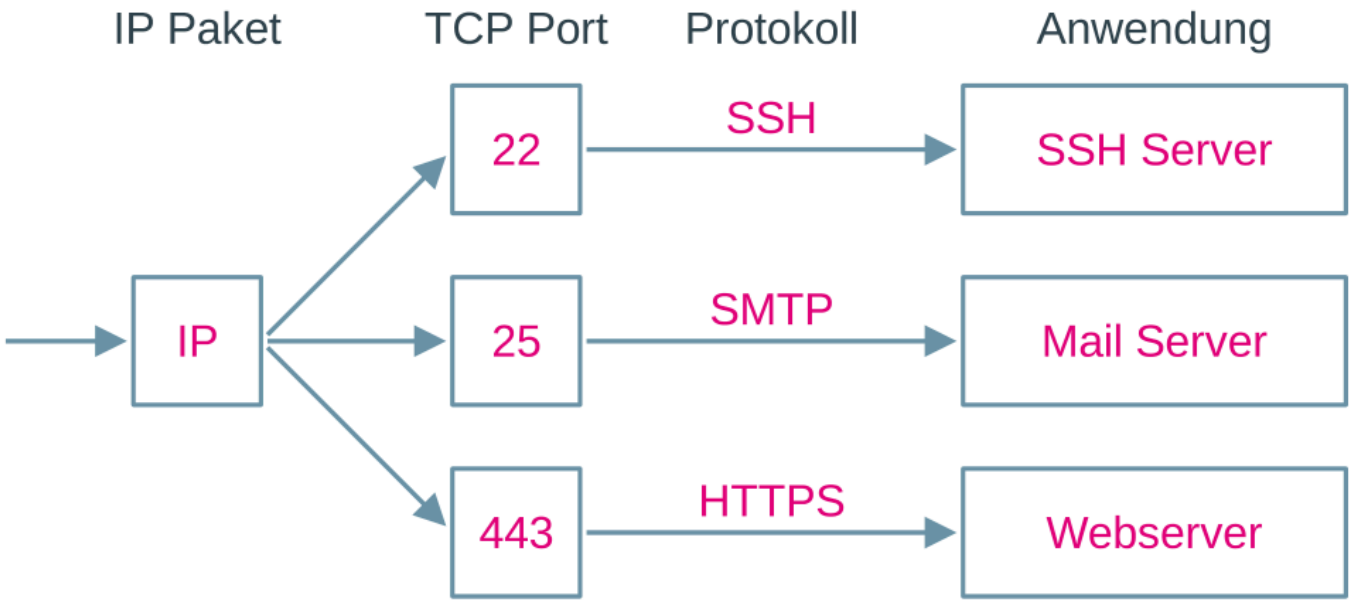
Verbindungsorientiert Vor Datenübertragung, wird eine Verbindung aufgebaut (Threeway Handshake)

Zuverlässige Datenübertragung Sicherstellung, dass alle gesendeten Daten korrekt beim Empfänger ankommen (Sequenz-Counter, ACK, Fehlerkorrektur [z.B. Prüfsummen])

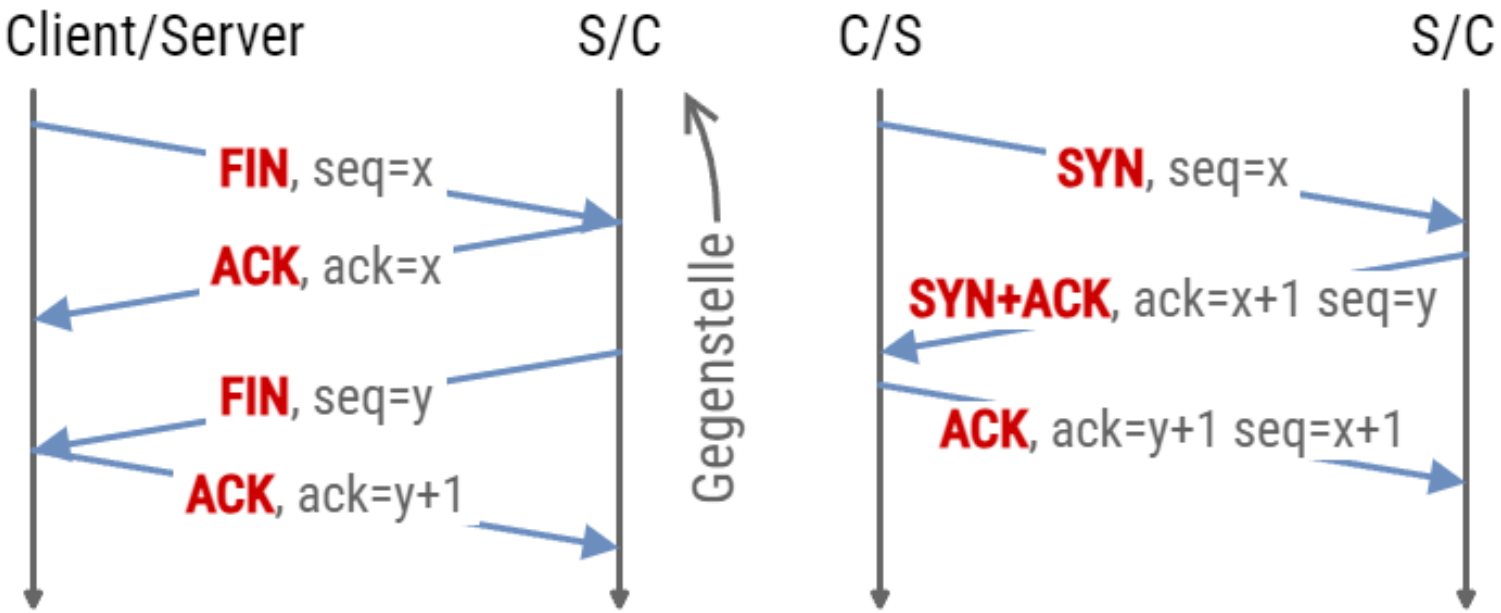
Segmentierung und Reassemblierung Grosse Datenmengen werden in kleinere Segmente (65535 bytes [64KB]) aufgeteilt und entsprechend beim Empfänger wieder zusammengesetzt.

Flow Control damit Sender den Empfänger nicht mit mehr Daten überfordert.

Congestion Control Dynamische Datenübertragungsrate anhand Netzwerkauslastung



Verbindungsaufbau wird via *Three-Way* Handshake gemacht (folgendes Bild rechts). Der Abbau mit einem *Four-Way* Handshake (folgendes Bild links).

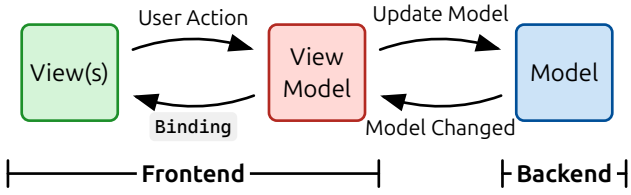


TODO Add Code Snippets here?

Interfaces

TODO Hello

MVVM



Model-View-ViewModel (MVVM) ist ein Entwurfsmuster und eine Variante des **Model-View-Controller**-Musters (MVC).

Darstellung und Logik wird getrennt in UI und Backend.

Vorteile

- ViewModel kann unabhängig von der Darstellung bearbeitet werden
- Testbarkeit keine UI-Tests nötig
- Weniger *Glue Code* zwischen Model & View
- Views kann separat von Model & ViewModel implementiert werden
- Verschiedene Views mit dem selben ViewModel.

⊗ Nachteile

- Höherer Rechenaufwand wegen bi-direktionalen „Beobachters“
- Overkill für simple Applikationen
- Datenbindung kann grosse Speicher einnehmen

Link 1, Link 2

View

What to display, Flow of interaction

Ist das User Interface des Programmes und ist via `Binding` und `Command` and das ViewModel gebunden.

View Model

Business Logic, Data Objects

Bildet den Zustand der View(s) ab. Es können verschieden Views mit dem selben ViewModel verbunden werden.

Model

How to display information

Beschreibt den Zustand für das Backend und kommuniziert mit anderen Prozessen (z.B. Betriebssystemroutinen)

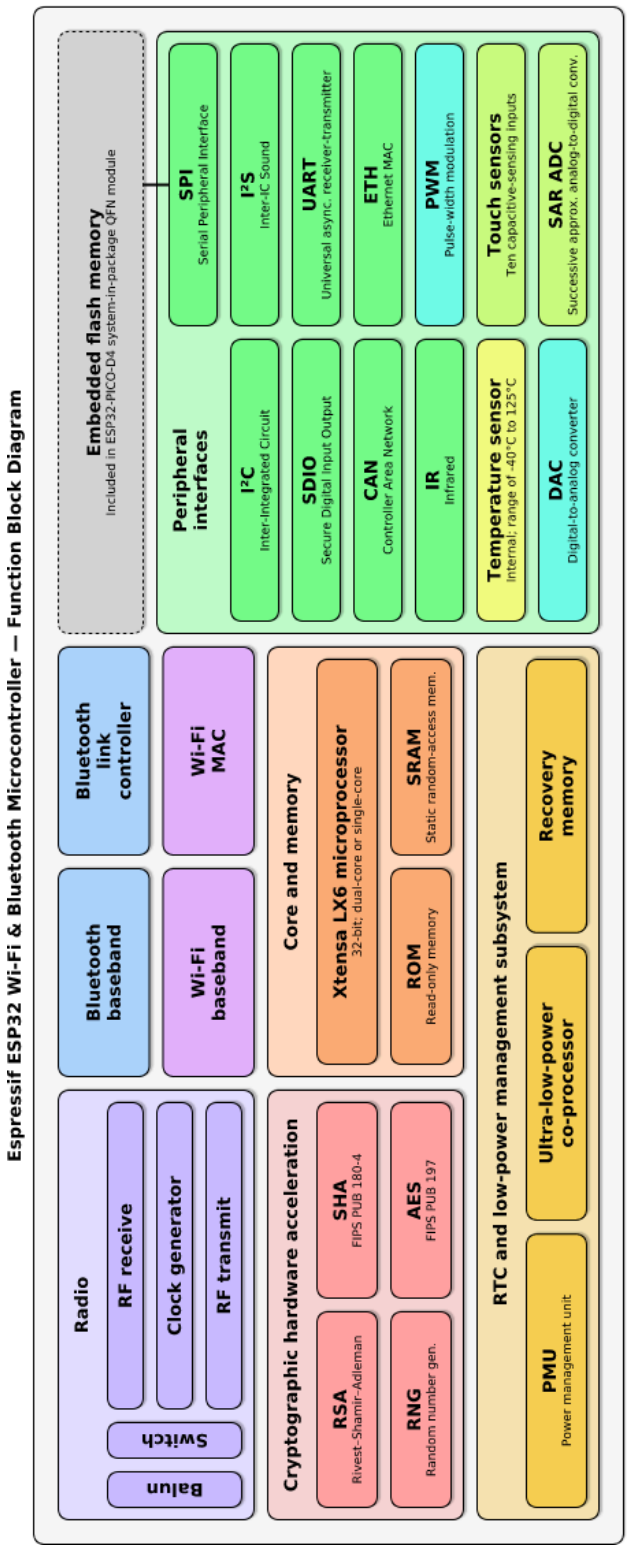
Werkzeuge & Entwicklung

Espressif

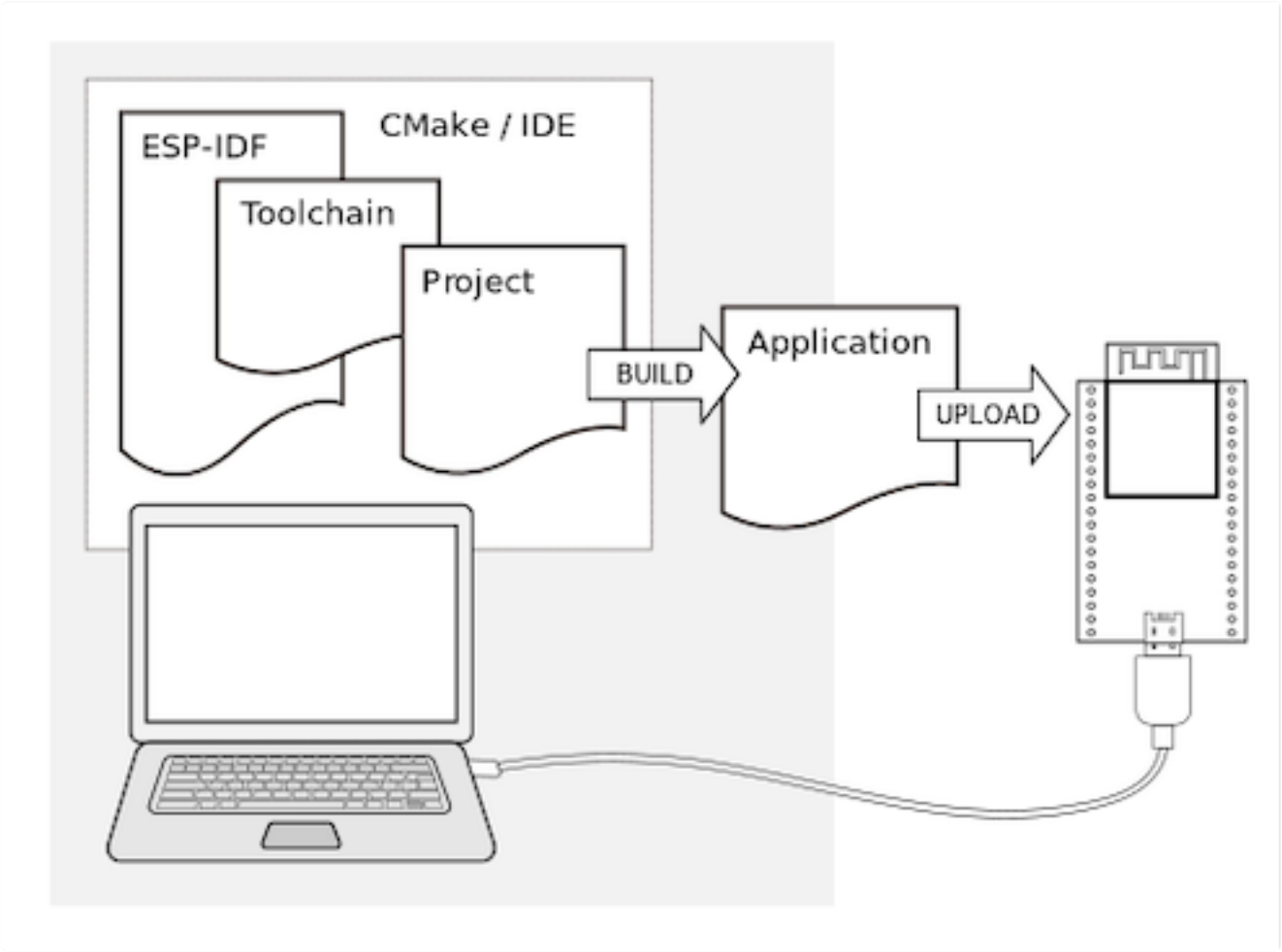
Chinesische Firma in Shanghai (Gründung 2008). Halbleiter-Chips werden bei TSMC hergestellt (*fabless* Herstellung).

Mikrocontroller

ESP8266 (2014) Tensilica Xtensa LX106, 64KB iRAM, 96KB DRAM, WiFi, ext. SPI Flash
ESP32 (2016) Wi-Fi + BLE, Single/Dual Core Xtensa LX6 @240 Mhz
ESP32-S2 (2019) Single-Core, Security, WiFi, keine FPU, kein Bluetooth, Xtensa LX7 @240 MHz
ESP32-S3 (2019) (FPU, WiFi+BLE, Dual-Xtensa LX7 @240 MHz, + RISC-V)
ESP32-C3 (2020) (Single-Core RISC-V @160 MHz, Security, WiFi+BLE)
ESP32-C6 (2021) (RISC-V @160 MHz, RISC-V @160 MHz + LP RISC-V @20 MHz, WiFi, Bluetooth/BLE, IEEE 802.15.4)
ESP32-H2 (2023) (Single-core RISC-V @96 MHz, IEEE802.15.4, BLE, no WiFi)

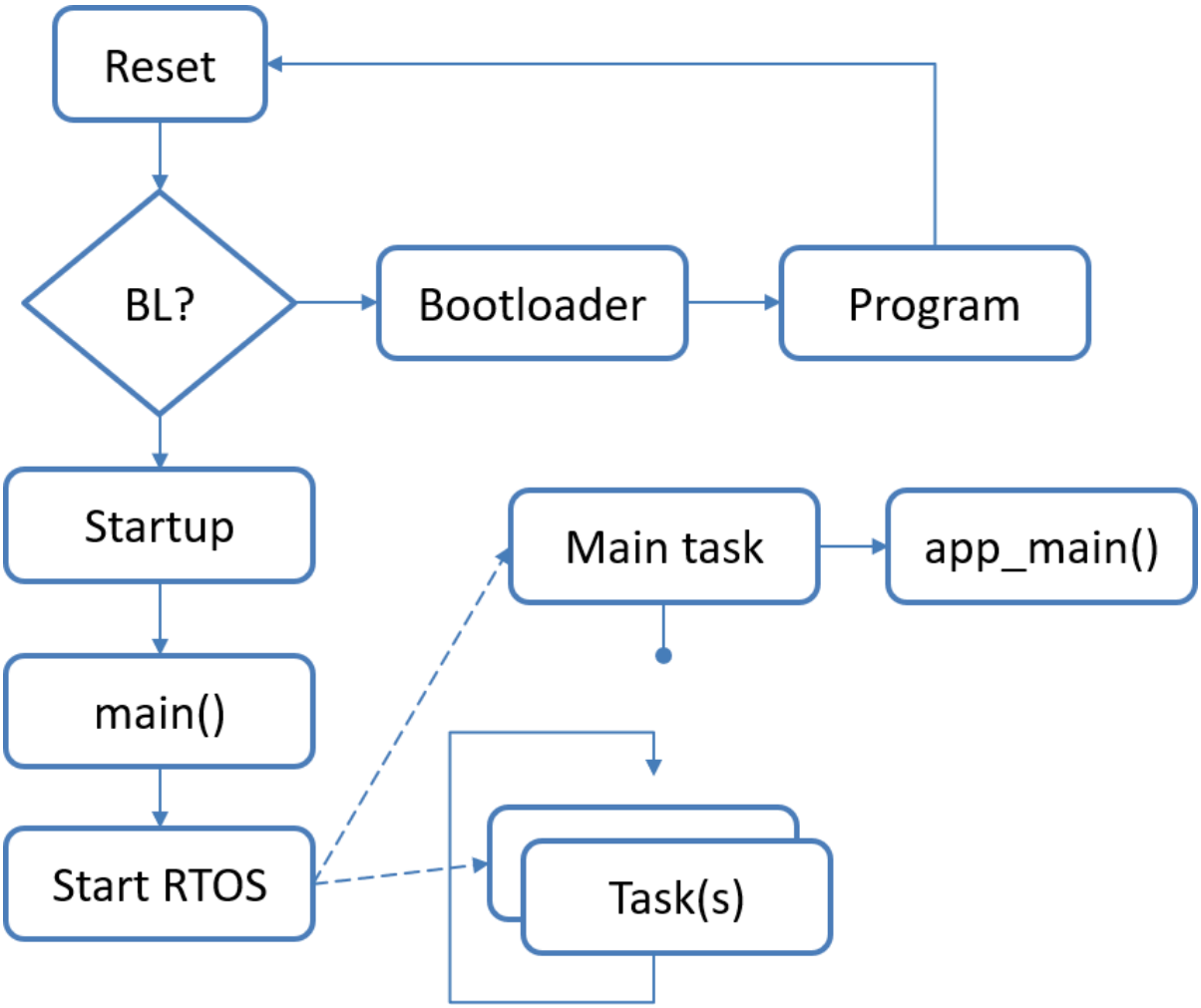

Espressif ESP32 Wi-Fi & Bluetooth Microcontroller — Function Block Diagram
Embedded flash memory
Included in ESP32-PICO-D4 system-in-package QFN module
Peripheral interfaces
SPI
Serial Peripheral Interface
I²S
Inter-IC Sound
UART
Universal async. receiver-transmitter
ETH
Ethernet MAC
PWM
Pulse-width modulation
Touch sensors
Ten capacitive-sensing inputs
SAR ADC
Successive approx. analog-to-digital conv.
I²C
Inter-Integrated Circuit
SDIO
Secure Digital Input Output
CAN
Controller Area Network
IR
Infrared
Temperature sensor
Internal; range of -40°C to 125°C
DAC
Digital-to-analog converter
Bluetooth baseband
Bluetooth link controller
Wi-Fi baseband
Wi-Fi MAC
Radio
Balun
Switch
RF receive
Clock generator
RF transmit
Core and memory
Xtensa LX6 microprocessor
32-bit; dual-core or single-core
ROM
Read-only memory
SRAM
Static random-access mem.
Cryptographic hardware acceleration
RSA
Rivest-Shamir-Adleman
SHA
FIPS PUB 180-4
RNG
Random number gen.
AES
FIPS PUB 197
RTC and low-power management subsystem
PMU
Power management unit
Ultra-low-power co-processor
Recovery memory

ESP-IDF



ESP32 Startup

1. Power-On oder Reset
2. Bootloader
3. EN Low? → Neue Anwendung über UART laden
4. Startup IDF (Initialisierung, Memory,…)
5. Starten FreeRTOS
6. ‚main' Task → ruft `app_main()` Funktion auf
7. Anwendung läuft in `app_main()`, oder started neue Tasks



ESP app_main()

- `app_main()` wird von einem Task gerufen → Task hat tiefste Priorität `0` (`tskIDLE_PRIORITY`)
- **FreeRTOS**: Preemptive, Highest Priority Task läuft
- `printf` wird auf Konsole (UART) umgeleitet

OpenOCD

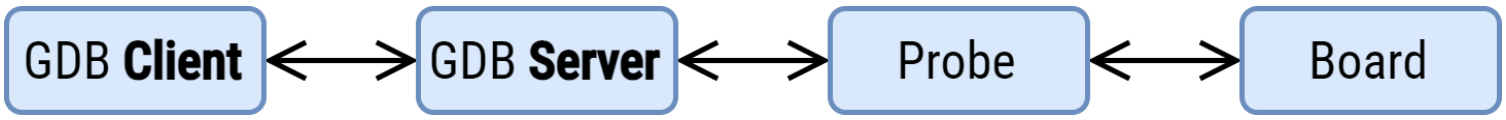
Open On-Chip Debugging

Ein **GDB-Server** für Debugging, In-System Programming und *boundry-scan* testing für Mikrocontroller-Systeme. Was eigentlich zwischen GBD und Mikrocontroller eingesetzt wird.

(ESP-IDF hat eine eigene modifizierte Version)

GBD Client-Server Architektur

GNU Debugger

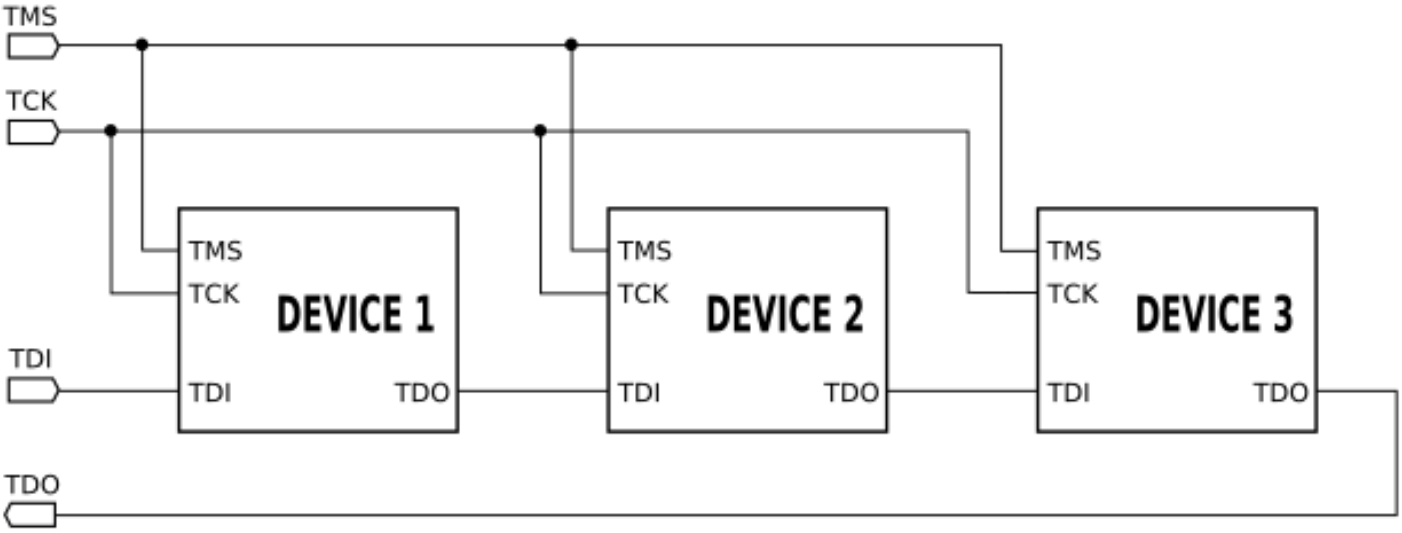


- IDE ⇔ GDB MI Protokoll ⇔ GDB
- IDE verbindet sich via GDB-Client zum GDB-Server (z.B. OpenOCD, Jlink GDB Server)
- Server übersetzt Befehle in **Debug** Signale (JTAG, SWD)

JTAG

Joint Test Action Group

- Shift Register Protokoll, für Design-Verifikation und Testen von Halbleitern.
- Daisy-Chain möglich!



⇒ **cJTAG** Variante mit weniger Pins: *TMS* → *TMSC*, ~~TDI~~

CMake

Löst das Problem der Abhängigkeit von Host & Toolchain von `make` → `cmake` ist ein **Generator**, welches dann mit `make` oder `ninja` weiterverarbeitet wird.

Verteilte Entwicklung

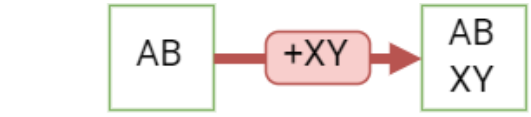
TODO

Git

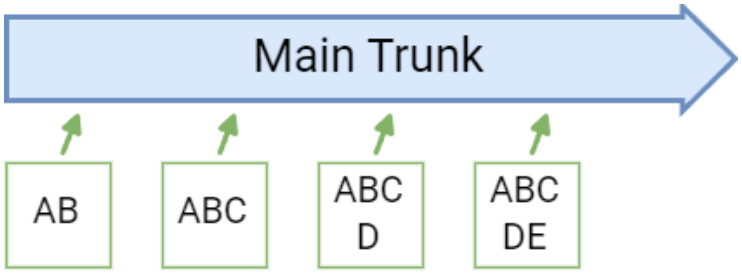
Git ist eine Versionsverwaltungssoftware!

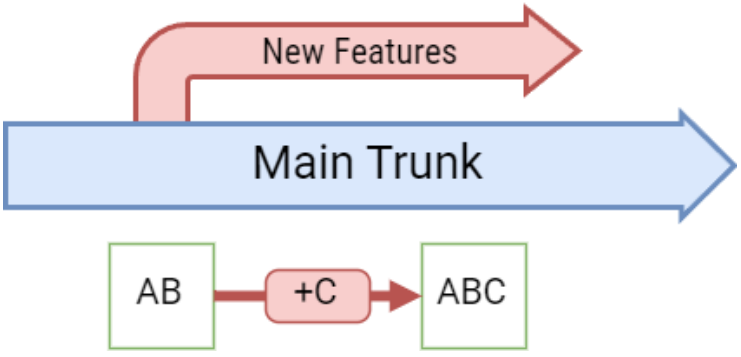
Konzepte

Following shows the most basic concepts used in version control systems such as Git.

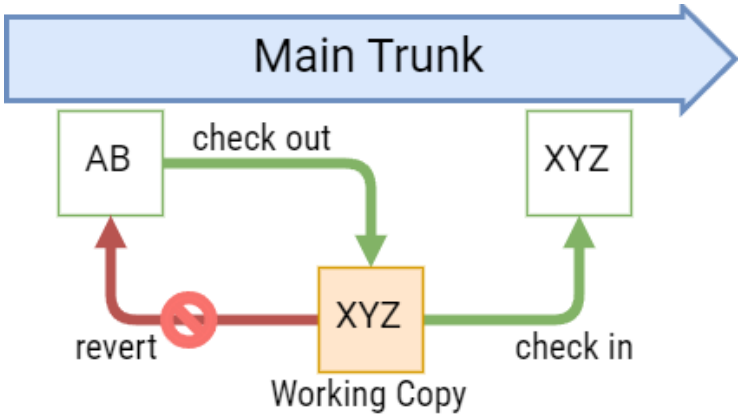


Basic Checkins



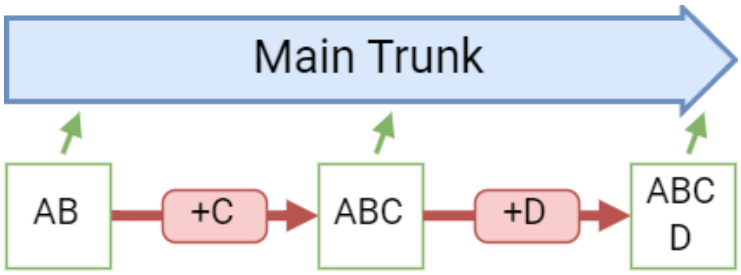


Checkout and Edit

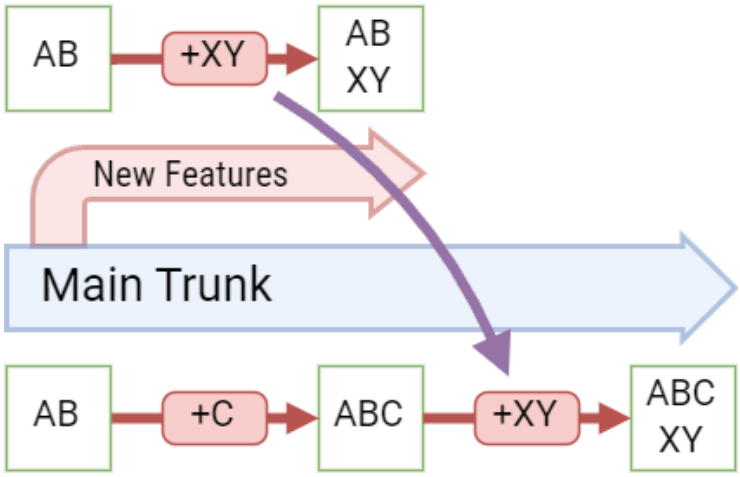


Branching

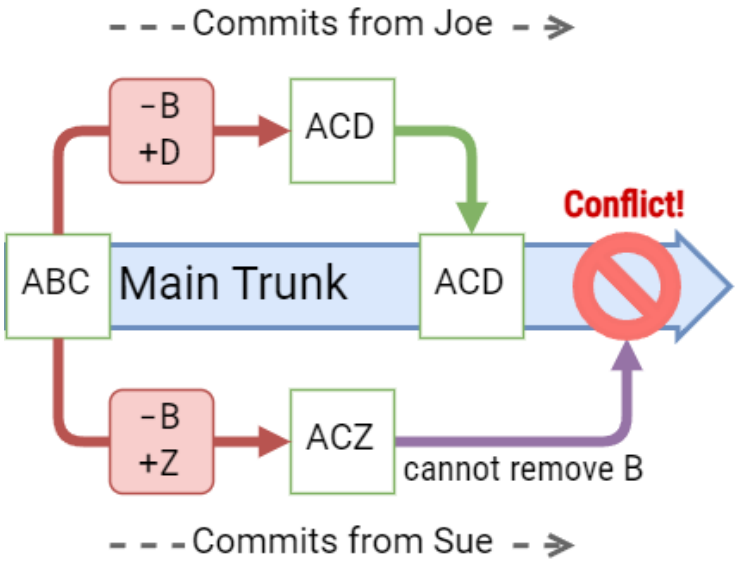
Diffs



Merging



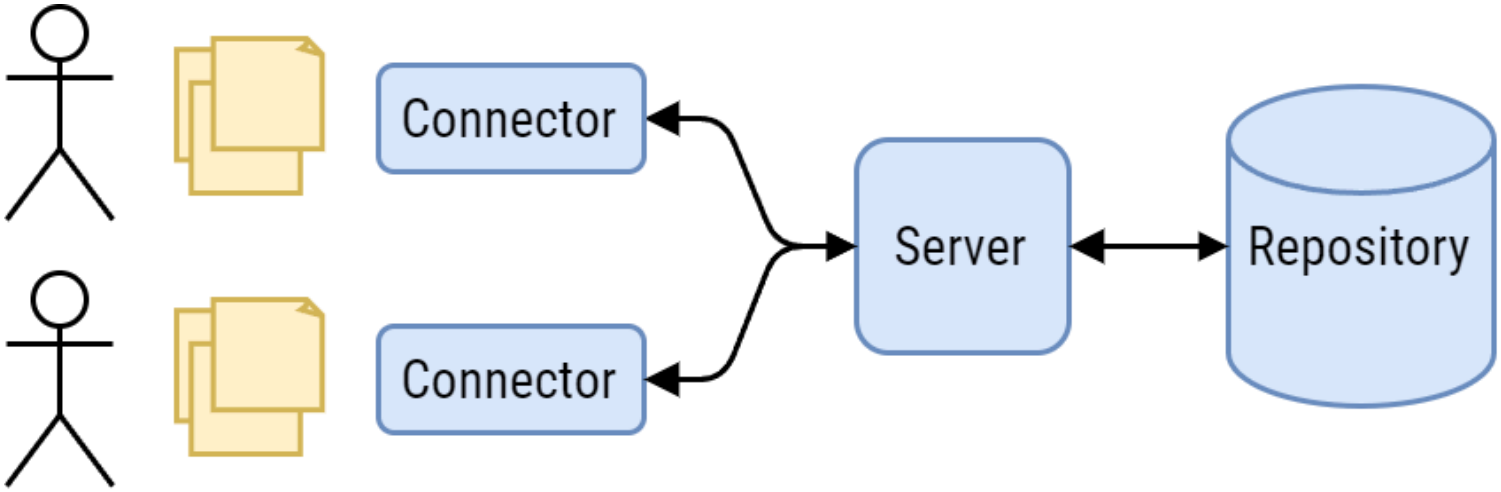
Conflicts



Was gehört in ein VSC

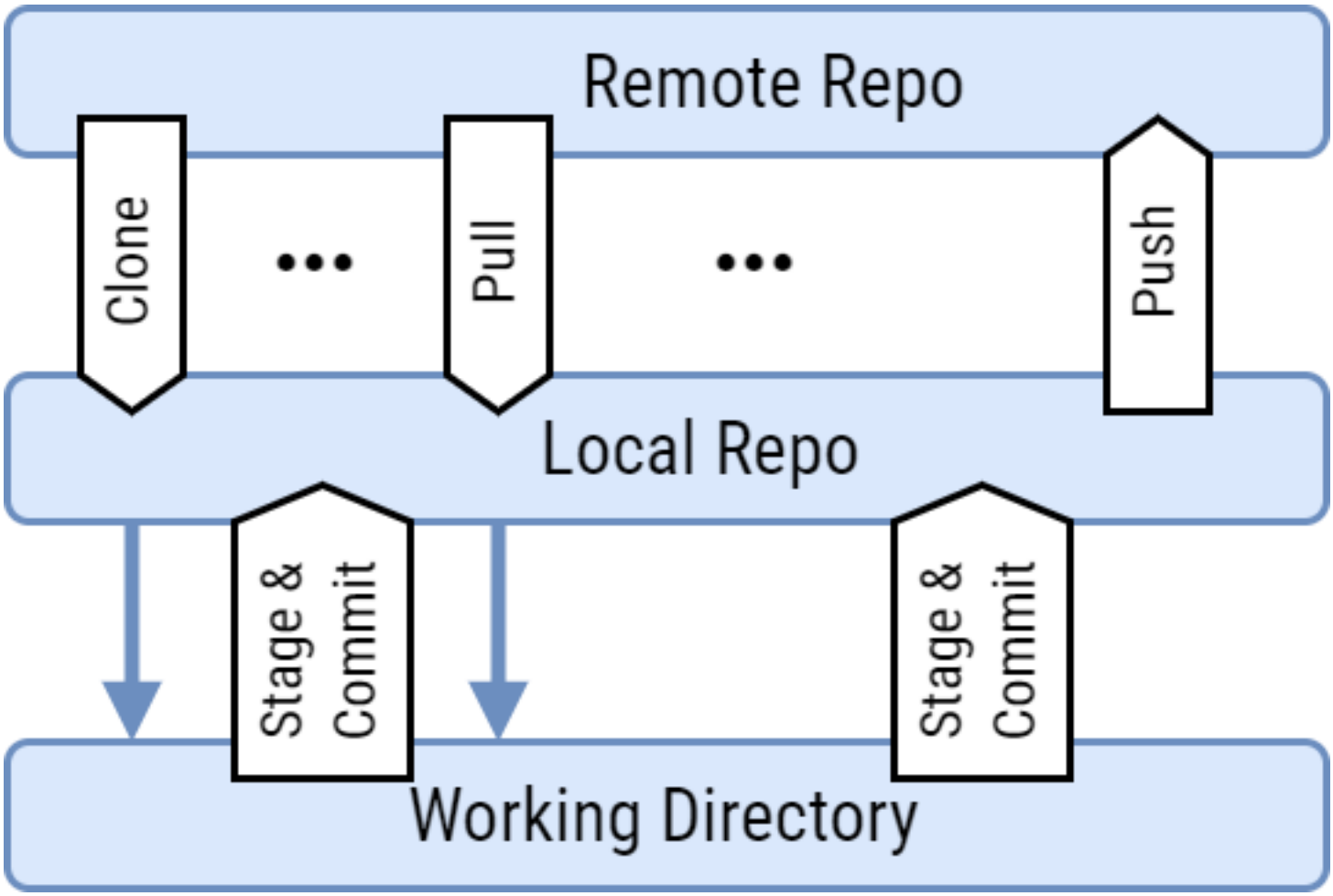
- **Kein** Backup: Source, Derived, Other
- `.gitignore`
- Stufen: Repository, Verzeichnis, rekursiv
- Empfehlungen

Modell



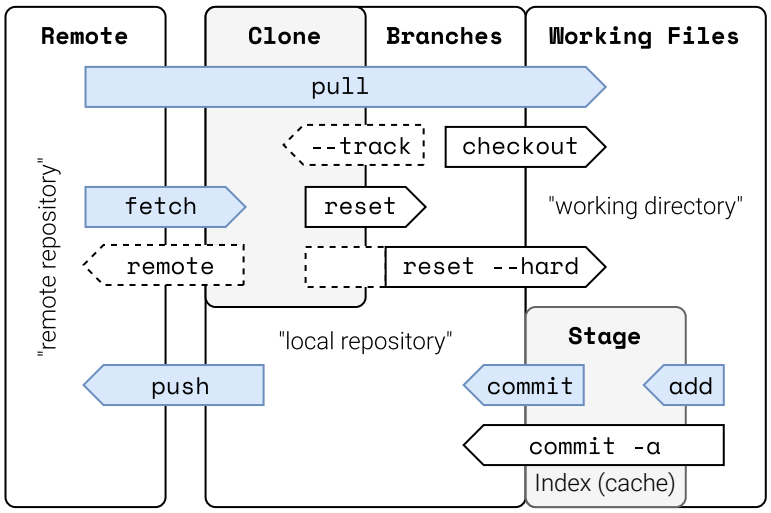
- **Connector**: git bash, Client
- **Server**: git (local und als Server (z.B. GitHub))
- Server & Repository: local, remote, verteilt
  - Zentralisiert (z.B. SVN) oder verteilt (z.B. git)
- Überlegungen: Platz, Übertragung, Arbeitsfluss

Workflow



„Commit Early, commit often“

Befehle



- Optionales **Remote Repository**
- **Local Repository** (clone)
- Lokale Datenbank existiert als **Working Directory** auf Disk
- **Index**: Cache, Stage, Sammlung von Änderungen, welche in die Datenbank überführt (**commit**) werden
- **fetch**, **pull**, **push**
- **checkout**, **add**

Setup & Konfiguration

Mit `git init` wird der aktuelle Arbeitsordner zu einem Git Repo konvertiert (rückgängig durch löschen von `.git` Ordner)

Möchte man ein Remote Repo *herunterladen* kann dies mit `git clone [url]` gemacht werden.

Danach muss das Repo konfiguriert werden.
Services wie GitLab & GitHub nutzen diese Information um die entsprechenden Profile anzugeben.

```
# local
git config user.name "[name]"
git config user.email "[email]"

# global
git config user.name "[name]"
git config user.email "[email]"
```

Add & Commit

```
# check current state of the repo
git status
# add/stage files based on pattern (or path)
git add [pattern]
# commit staged files with message
git commit -m "[msg]"
# push changes to the remote repo
git push
# unstage staged files
git reset -- [pattern]
# compare changes of the file
git diff [file]
# compare changes of the staged file
git diff --staged [file]
```

Branch & Merge

Branches sind nützlich für separate Entwicklungen von Funktionen, welche nach Testen in den Hauptbranch gemerget werden.

Branches werden erstellt mit folgendem Befehl (+ weitere nützliche Befehle)

```
git branch [new_branch] # create
git branch -m [old_branch] [new_branch] # rename
git branch -c [old_branch] [new_branch] # copy
git branch -d [branch] # delete
git switch [branch] # switch to branch
```

Branch Merging verläuft mit dem Prinzip:

*Merge commits **FROM** the stated branch*

Also muss man sich im Destinations-Branch befinden und von dort aus die Änderungen von Merge-Branch reinmergen!

1. Änderungen im Branch `dummy` **commiten** und **pushen**
2. Branch zu `main` wechseln

```
git switch main
```

3. Merge Operation ausführen

```
git merge [-m "[msg]"] dummy
# No automatic merge commit (used for inspection)
git merge --no-commit dummy
```

Merge-Konflikten

Merge-Konflikte treten in der Regel in den folgenden Szenarien auf:

Simultaneous Edits Zwei Entwickler ändern dieselbe Codezeile in verschiedenen Branches
Conflicting Changes Eine Datei wird in einem Branch gelöscht und im anderen geändert
Complex Merges Mehrere Branches werden gemerget und es entstehen Änderungen über mehrere Dateien und Zeilen

Bei Konflikten, kann das Mergetool verwendet werden

```
git mergetool
```

Möchte man den Merge rückgängig machen

```
git merge --abort # revert to pre-merge state
```

ESP32

WiFi

Erich Styper Implementation

[`initialise_wifi`]:
- FreeRTOS Event Group erstellt für Connections/Disconnections/etc. verwendet

```
s_wifi_event_group = xEventGroupCreate();
esp_netif_init();
esp_event_loop_create_default();
APP_WiFi_NetIf = esp_netif_create_default_wifi_sta();
```

- WiFi Konfiguration basierend auf MAC holen

```
config = ESP32_GetDeviceConfig();
esp_netif_set_hostname(APP_WiFi_NetIf, config->hostName);
```

- Standard WiFi Konfiguration initialisieren

```
config = ESP32_GetDeviceConfig();
esp_netif_set_hostname(APP_WiFi_NetIf, config->hostName);
```

- Event Handler registrieren
  - Callback, wo Event Group bits gesetzt werden

```
esp_event_handler_register(WIFI_EVENT,
    ESP_EVENT_ANY_ID, &event_handler, NULL);
esp_event_handler_register(IP_EVENT,
    IP_EVENT_STA_GOT_IP, &event_handler, NULL);
```

⇒ Falls eine Verbindung nicht geht, wird die alternative Verbindung (z.B. Home-WiFi) genommen.

UDP

User Data Protocol

- UDP-Datagramm:

$\texttt{SrcPort}_{16b} + \texttt{DstPort}_{16b} + \texttt{Length}_{16b} + \texttt{CRC}_{16b} + \texttt{Data}$

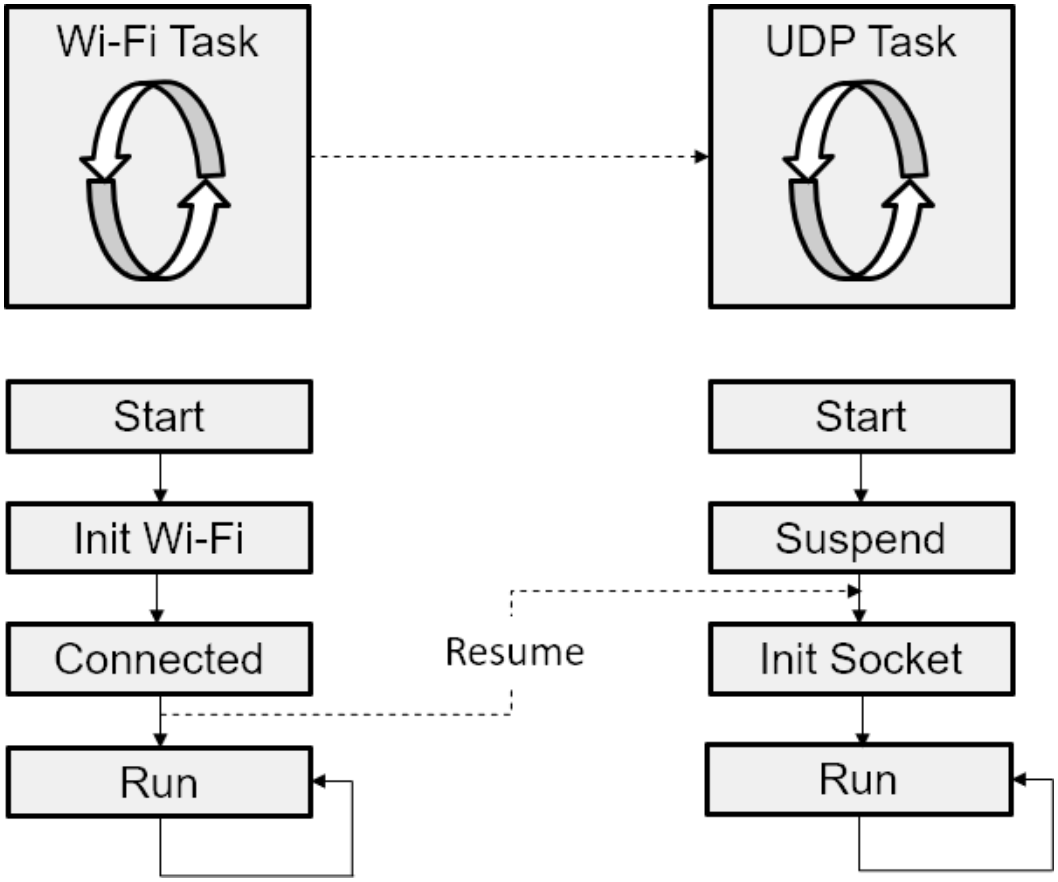
- IP-Datagramm:

$\texttt{IP-Header}_{96b\ (IPv4)} + \texttt{UDP-Datagramm}$

- IPv4 Header:

$\texttt{SrcIP}_{32b}$ + $\texttt{DstIP}_{32b}$ + 0_{8b} + $\texttt{P-ID}_{8b}$ + $\texttt{Length}_{16b}$

UDP hat einen kleineren Header als TCP (8-Byte vs 20 Byte)!

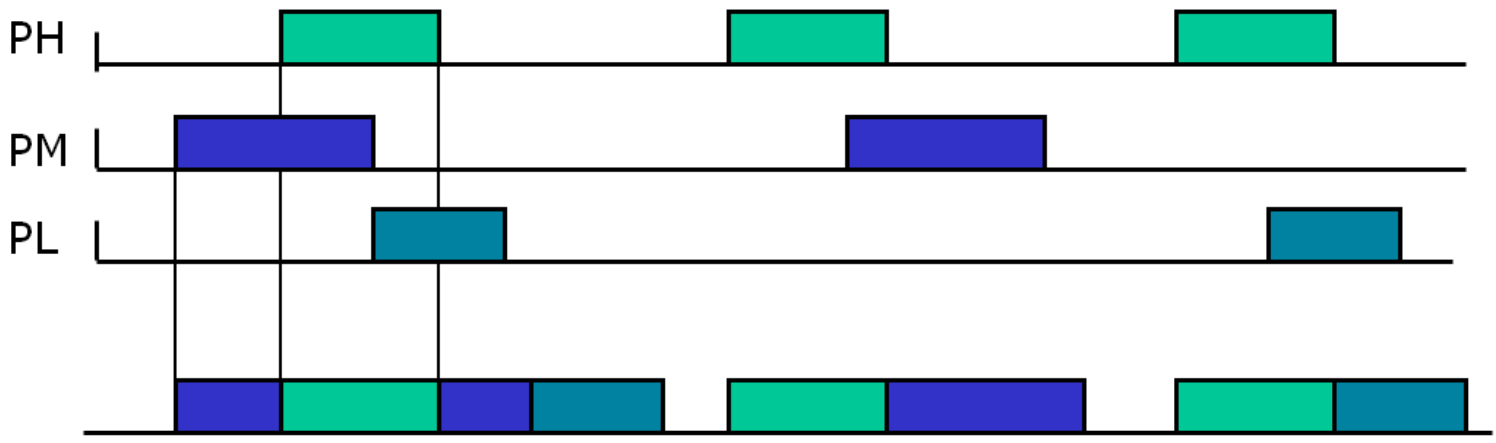


(Espressif) FreeRTOS SMP

Symmetric Multiprocessing

- **RTOS**: Skalierbarkeit, Erweiterbarkeit, Synchronisation
- SMP wurde von Espressif entwickelt (gleiche MIT Lizenz)
  - Dual-Core Tensilica, gemeinsamer Speicher
  - `CPU0` → `PRO_CPU` Protocol
  - `CPU1` → `APP_CPU` Application (`app_main()`)
- Tasks können an spezifischen Task gepinnt werden

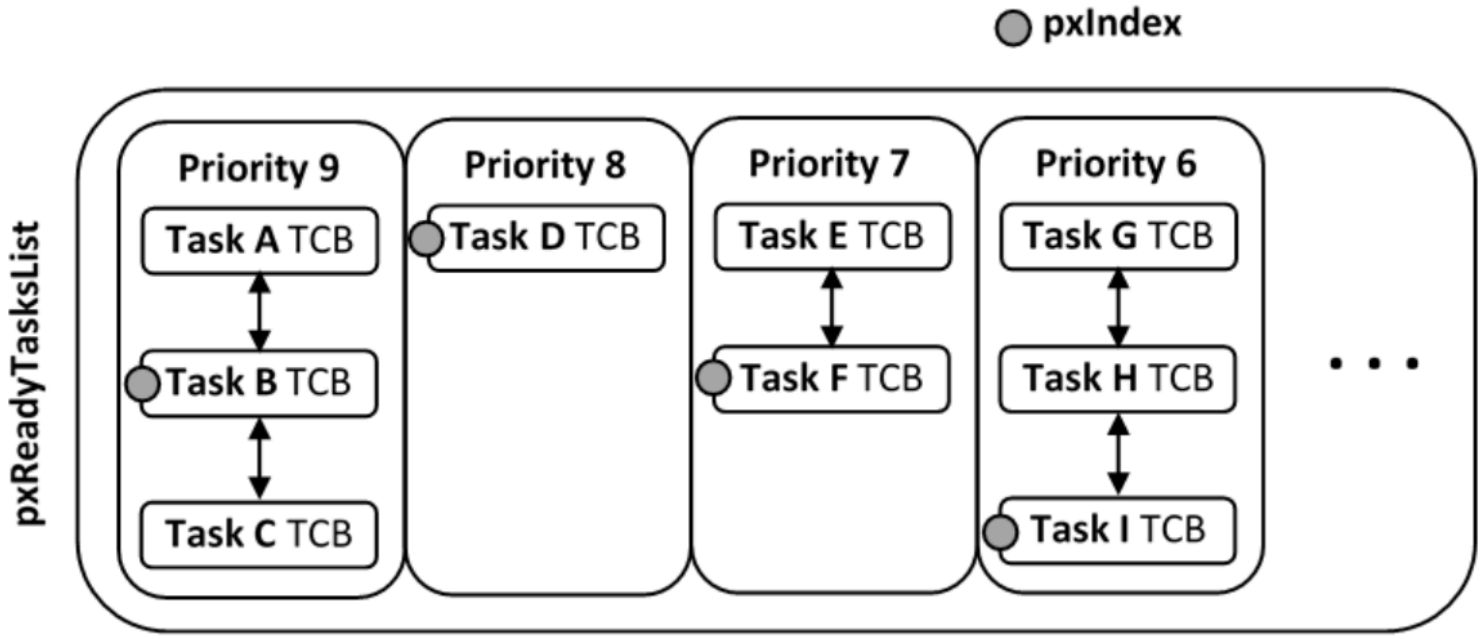
Prioritäten



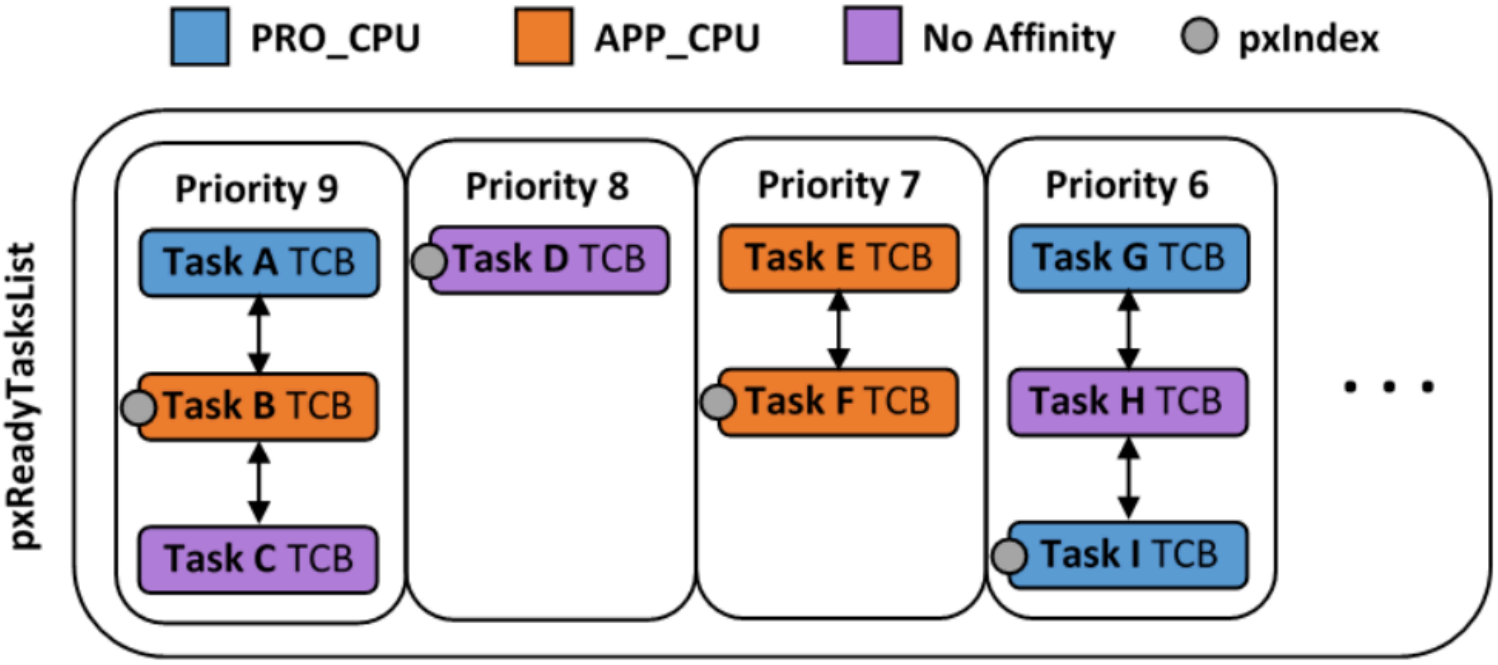
- 0 (`tskIDLE_PRIORITY`) ist tiefste Dringlichkeit (FreeRTOS: Val↑Prio↑, ARM: V↑P↓)
- Es läuft „ready" Task mit höchster Prio
- preemptive: Scheduler unterbricht Tasks

SMP Round Robin (RR) Scheduling

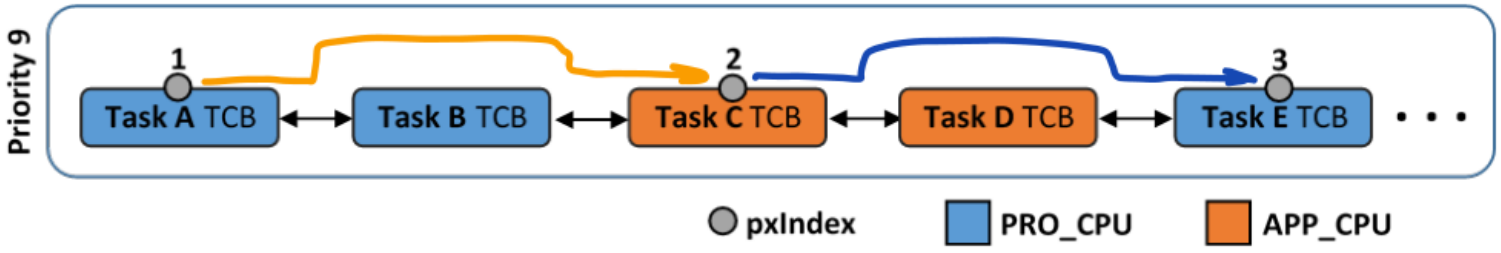
- Standard FreeRTOS: *RR* für Tasks gleicher Prio



- FreeRTOS SMP: *Best-Effort RR*
  - Scheduler auf **jedem** Core! → behandelt nur eigene Tasks, tick interrupt **NICHT** synchronisiert
  - **Gemeinsame** Task Liste



- Core-Scheduler überspringt Tasks gleicher Prioritäten des anderen Cores!



Tasks

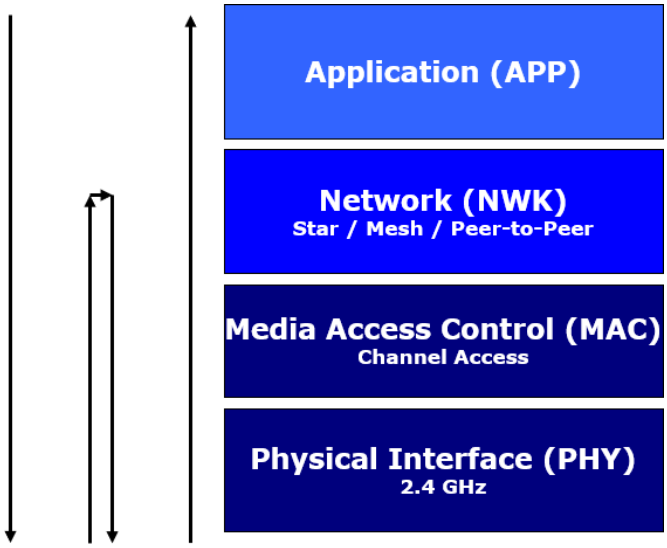
Erstellen eines Tasks ist ähnlich wie beim Standard-FreeRTOS, ausser die Zuweisung auf einen Core (wird mit wrapper umgangen!)

```
// Standard & Wrapper
BaseType_t xTaskCreate(
```

```
  TaskFunction_t pvTaskCode, const char *const pcName,
  const uint32_t usStackSize, void *const pvParam,
  UBaseType_t uxPrio, TaskHandle_t *const pvTaskHndl);
// SMP
xTaskCreatePinnedToCore(
  ...,
  tskNO_AFFINITY)
```

`0`: auf `PRO_CPU` ; `1`: auf `APP_CPU` ; `tskNO_AFFINITY`: auf beiden

RNet



APP: SendMessage(0x12,„hello“)

NWK: resend(msg), route(msg,dst), connect(addr, port), etc.

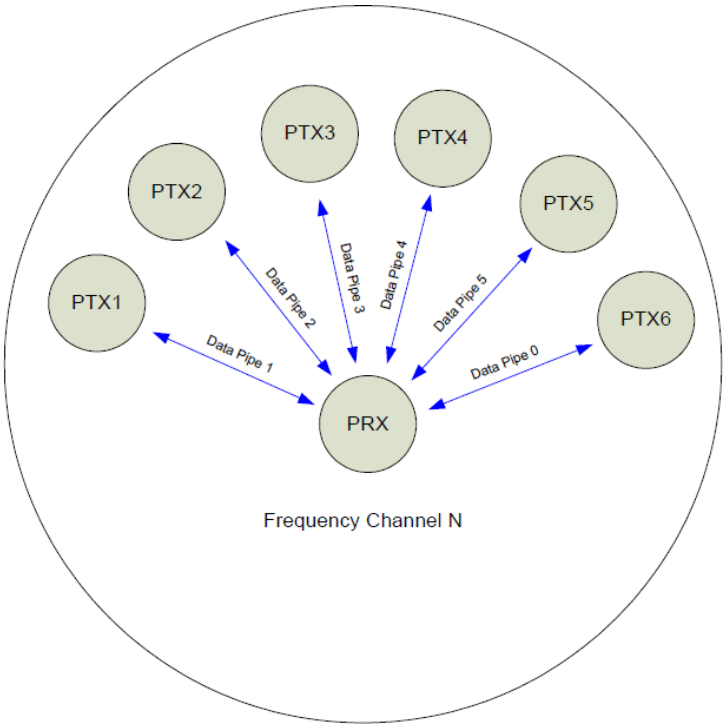
MAC: SelectChannel(6), ScanChannels(all)

PYH: GetLQI(), SetRxMode(), SetTxMode(), SPITransmit(buf)

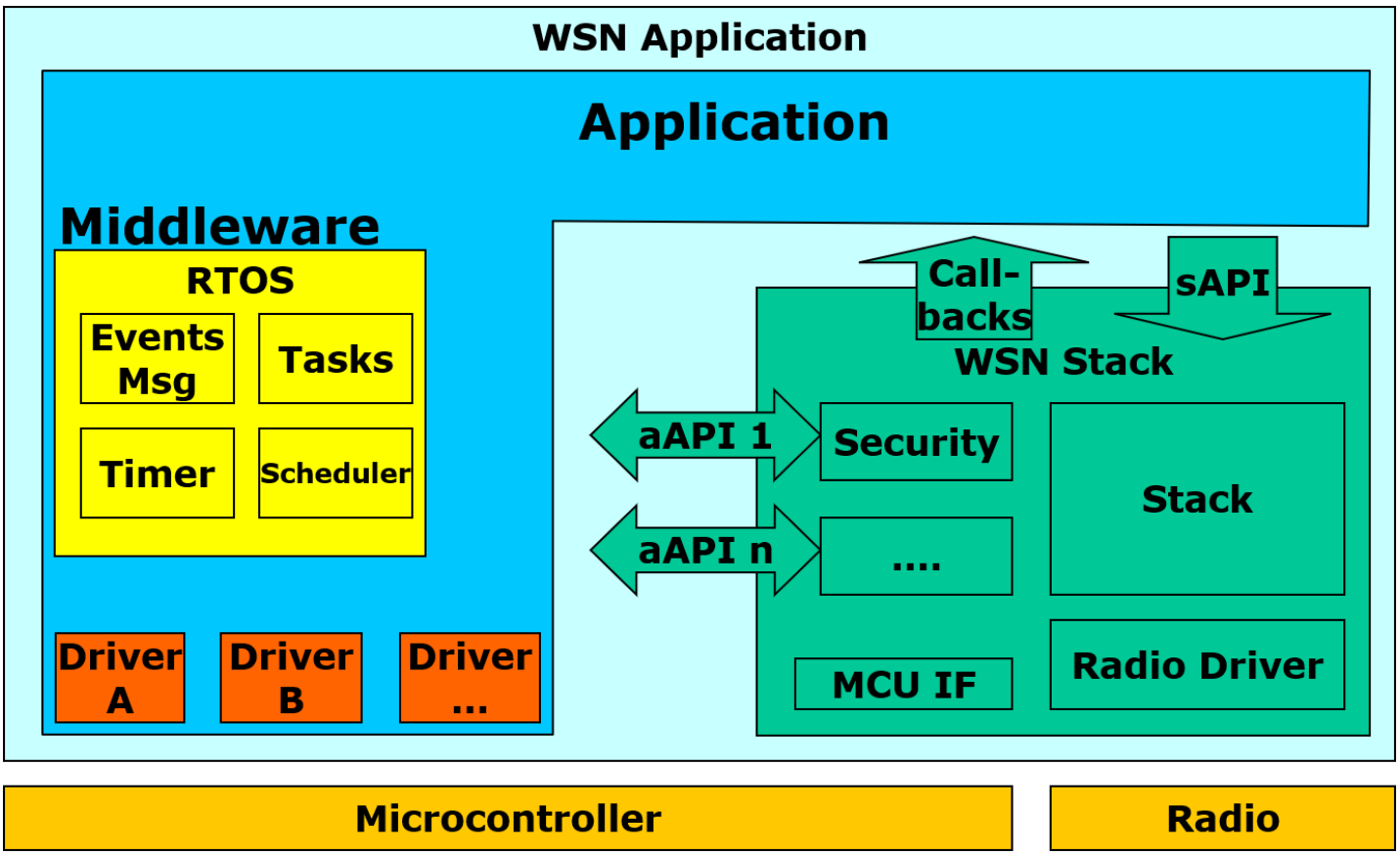
nRF24L01+

- Nordic nRF24L01+
- Pins: SPI, CE, CSN, IRQ
- 2.4 GHz ISM
- 250 kbps, 1 Mbps, 2 Mbps
- Enhanced ShockBurst: auto ACK & retry
- Payload: max 32 Bytes
- 6 data pipe MulitCeiver

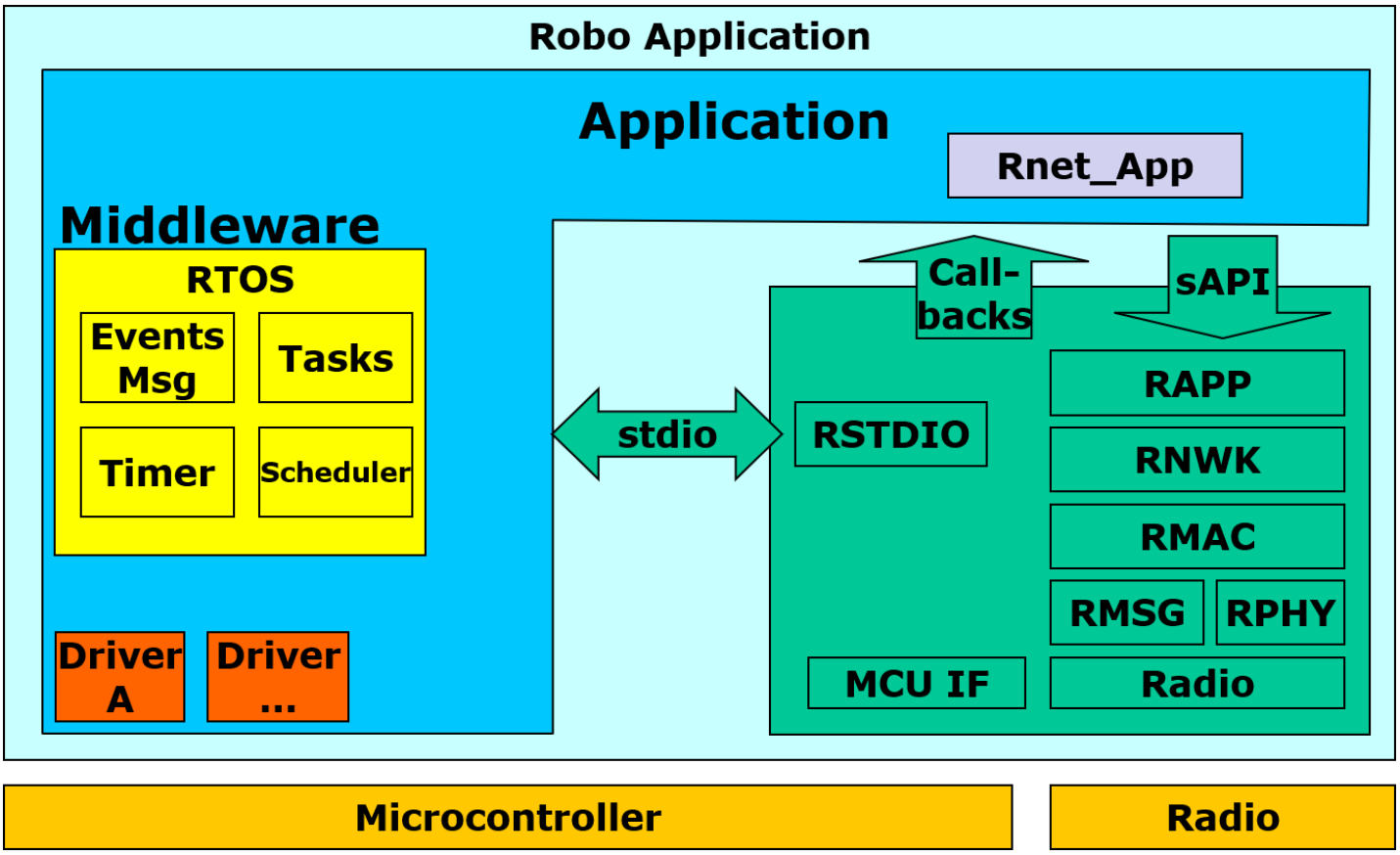
Receive Data Pipes: Empfangs-„Kanäle“ ⟹



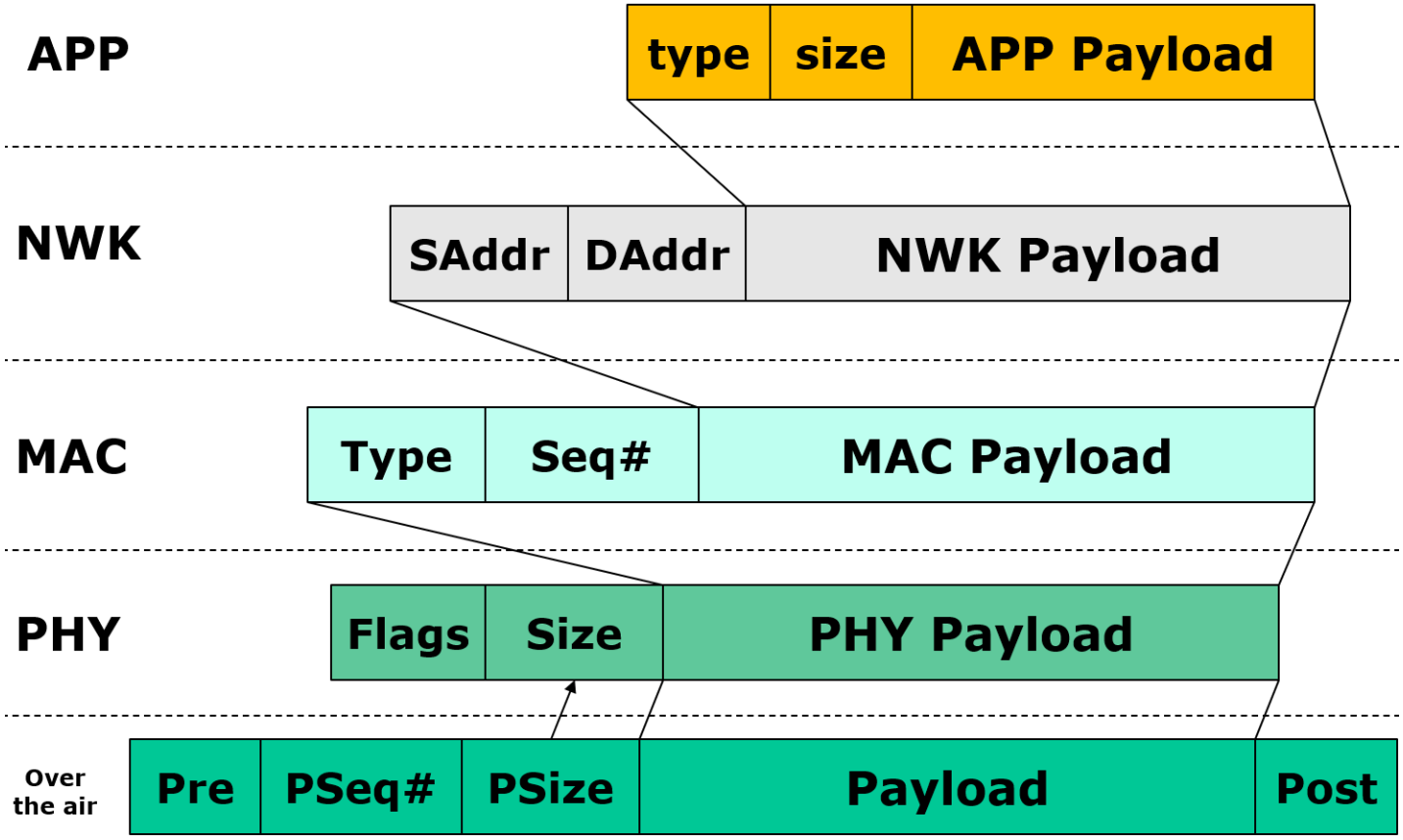
Übliche WSN Anwendung und Stack



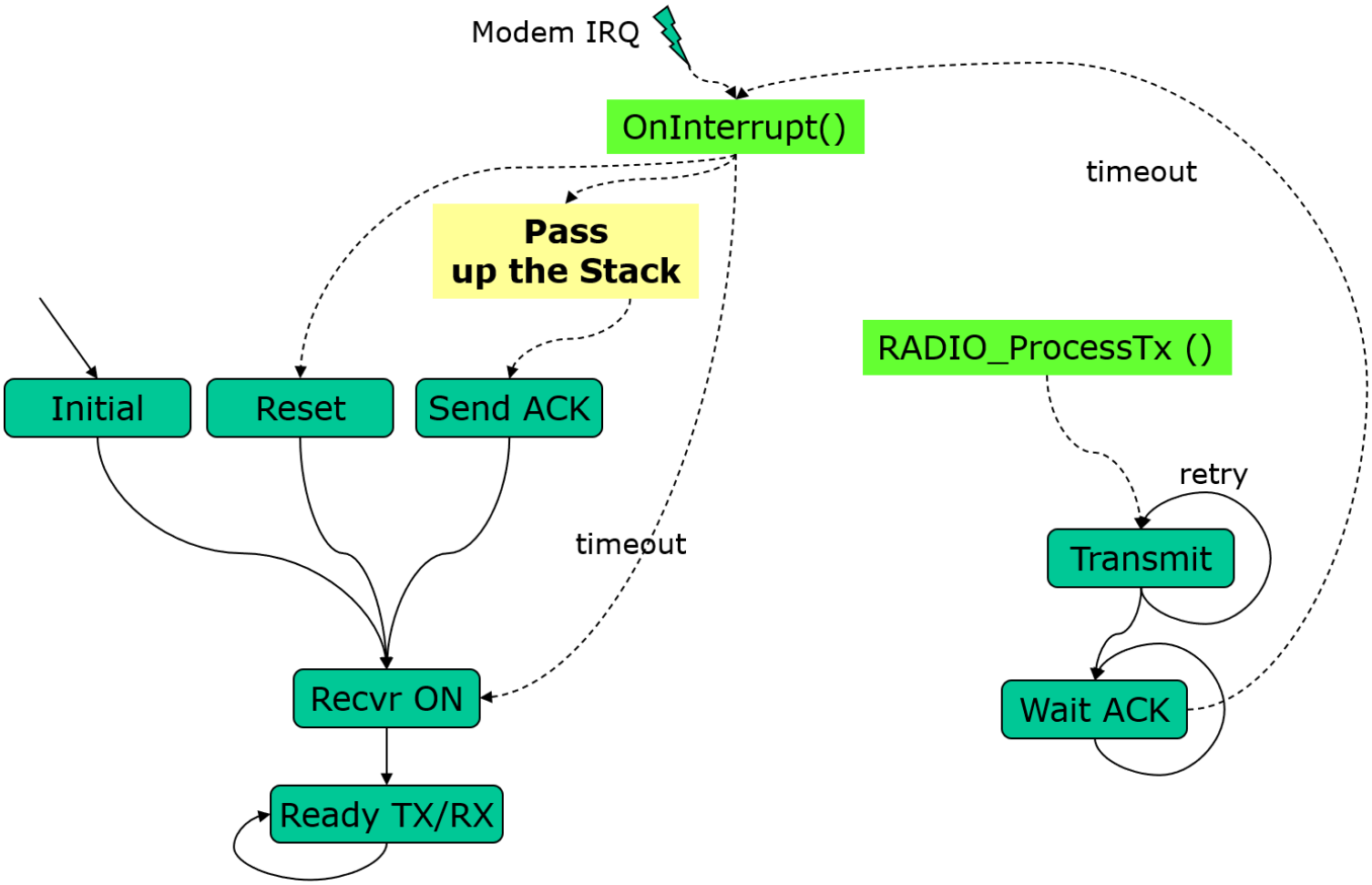
RNet Stack Anwendung

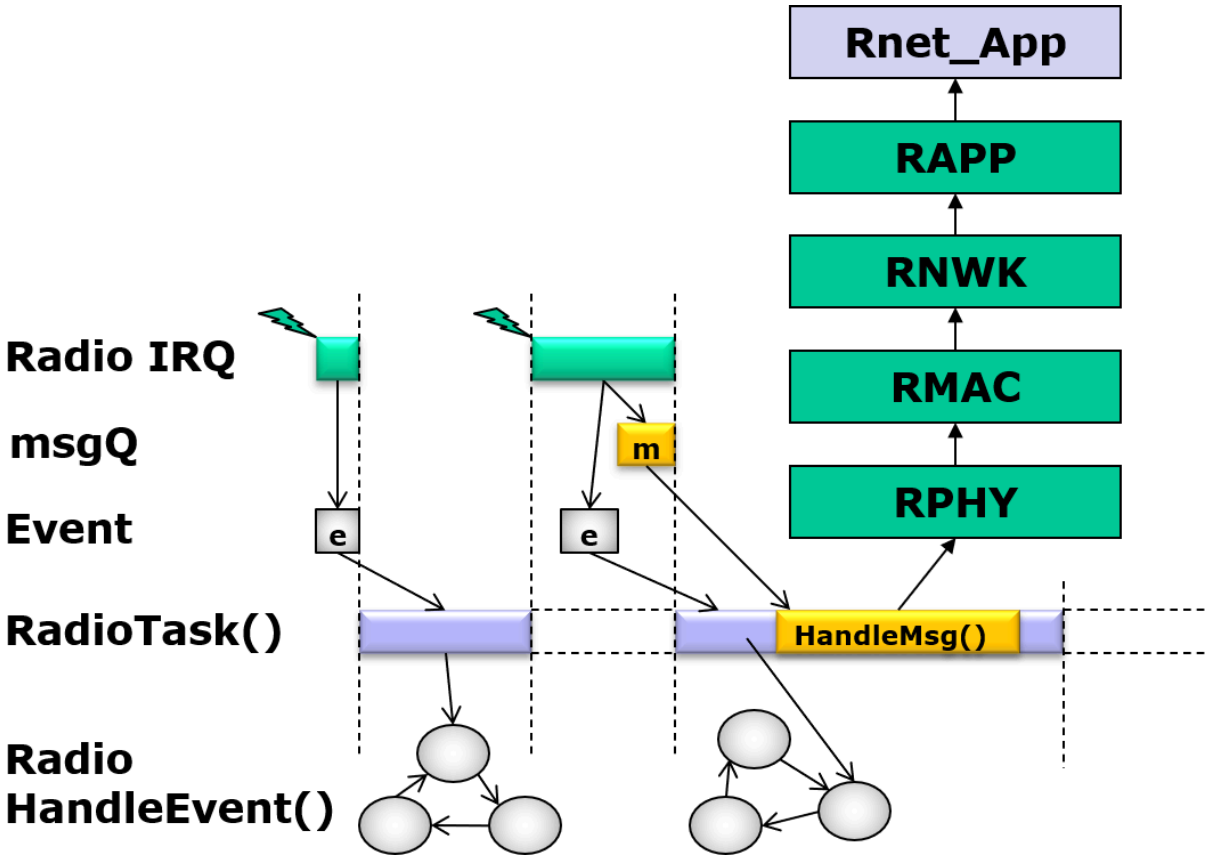


Payload Packaging



Radio States & Processing



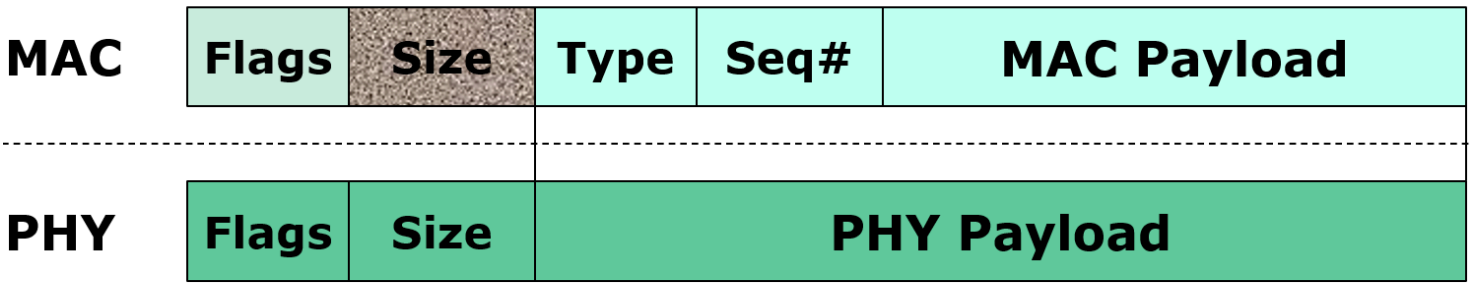


Der Radio-Stack behandlet die Payloads via Queues (mit einem Task)! Senden wird direkt in die Queue geschrieben via Wrapper Funktion. Lesen wird über ein `OnPackt` -Event ausgelöst.

Copy-Less Stack Operation

Pakete werden durch den Stack gereicht, damit nicht alles kopiert werden muss! Am Schluss werden Dateien hinzugefügt.

Beispiel: **MAC** & **PHY** sind inhaltlich gleich, nur die Betrachtung anders



Verteile Architekturen

Multicore

Konfiguration

Feldbus

Drahtlos

Remote Access

FreeRTOS Crash Kurs

FreeRTOS SMP

Symmetric **M**ulti**P**rocessing

Critical Sections, Reentrancy

Semaphore

Mutex

Nachrichten

Semaphore

Event Flags

Queues

Direct Task Notification

Stream Buffer

Message Buffer

CI/CD

Continuous **I**ntegration and **C**ontinuous **D**elivery

Pipeline

Ausführung von Jobs & Stages überspringen

```
when: manual        # .gitlab-ci.yml
[ci skip]           # commit message
git push -o ci.skip # command line
```