# Decoupling User and Kernel Space:

## A System Call Framework for OctoPOS

May 29, 2020

Eva Dengler

Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

# Motivation

**Aktuell:** OctoPOS, das Betriebssystem für InvasIC, ohne Trennung zwischen *user* und *kernel space*

# Motivation

**Aktuell:** OctoPOS, das Betriebssystem für InvasIC, ohne Trennung zwischen *user* und *kernel space*

+ schnell durch direkten Funktionsaufruf

- Anwendung hat vollen Zugriff auf das laufende System

- starke Koppelung von Anwendung und Betriebssystem

# Motivation

**Aktuell:** OctoPOS, das Betriebssystem für InvasIC, ohne Trennung zwischen *user* und *kernel space*

+ schnell durch direkten Funktionsaufruf

- Anwendung hat vollen Zugriff auf das laufende System

- starke Koppelung von Anwendung und Betriebssystem

**Aufgabe:**
teile OctoPOS in *user* und *kernel space* zur späteren Realisierung eines Schutzkonzeptes

# Motivation

**Ziel:**
eine Schnittstelle zwischen *user* und *kernel space*, ...

- ... um Betriebssystemfunktionen anzufordern
- ... die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- ... die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- ... die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Inhalt

# Systemaufrufe

# Systemaufrufe

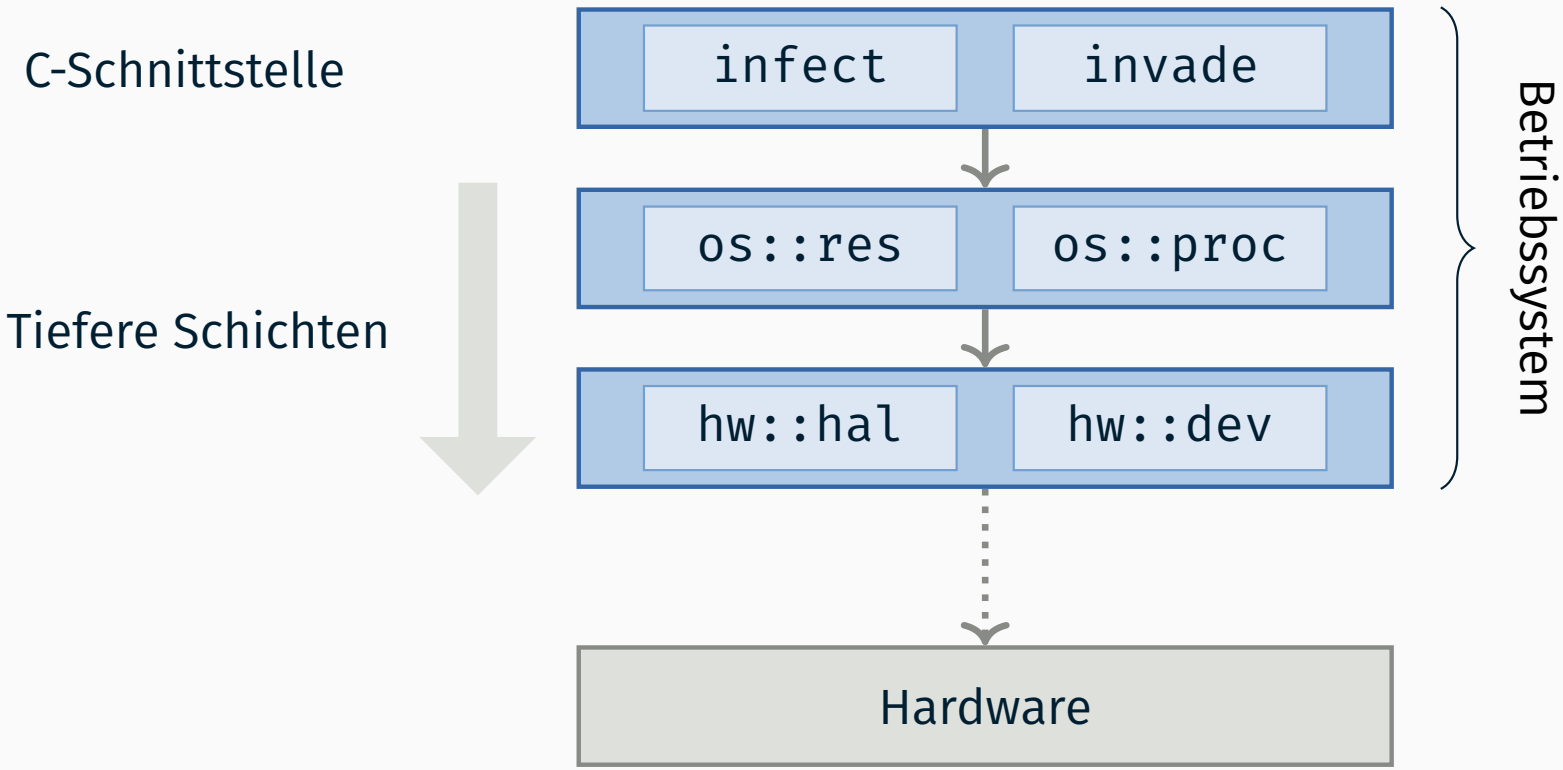
Aus dem Sachwortverzeichnis
*Systemprogrammierung - Grundlage von Betriebssystemen*

**Systemaufruf** *(en.) system call*.
Aufruf von einem im Betriebssystem liegenden Unterprogramm, initiiert durch eine in einem Maschinenprogramm kodierte Aktion. Letztere kommt durch einen Systemaufrufbefehl zum Ausdruck.

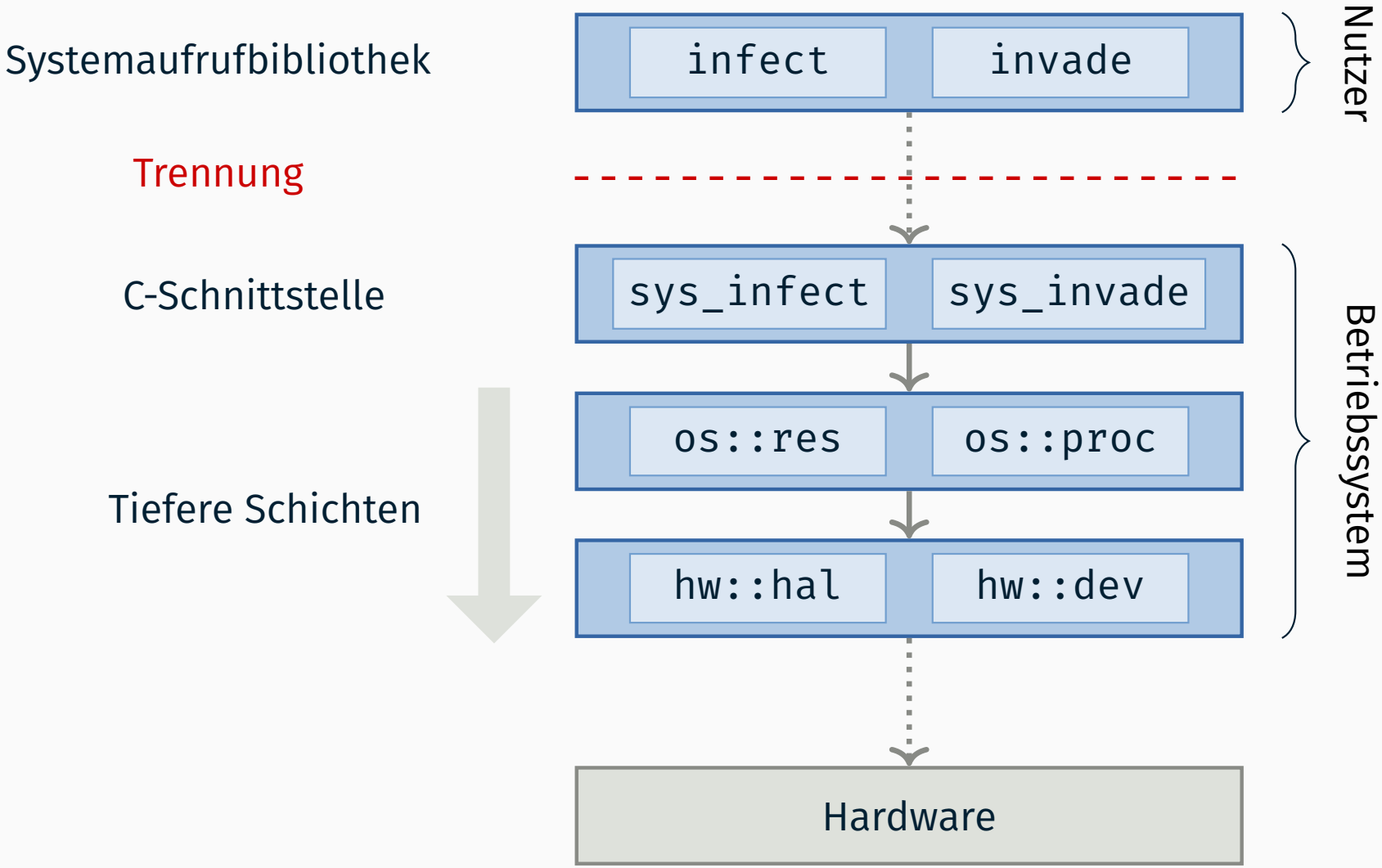
# Identifizierung der Systemaufrufe

- Jede Funktion mit eindeutiger Nummer gekennzeichnet
- Systemaufruftabelle für Zuordnung von Nummer zu Funktion
- Tabelle zur Laufzeit (re-)konfigurierbar

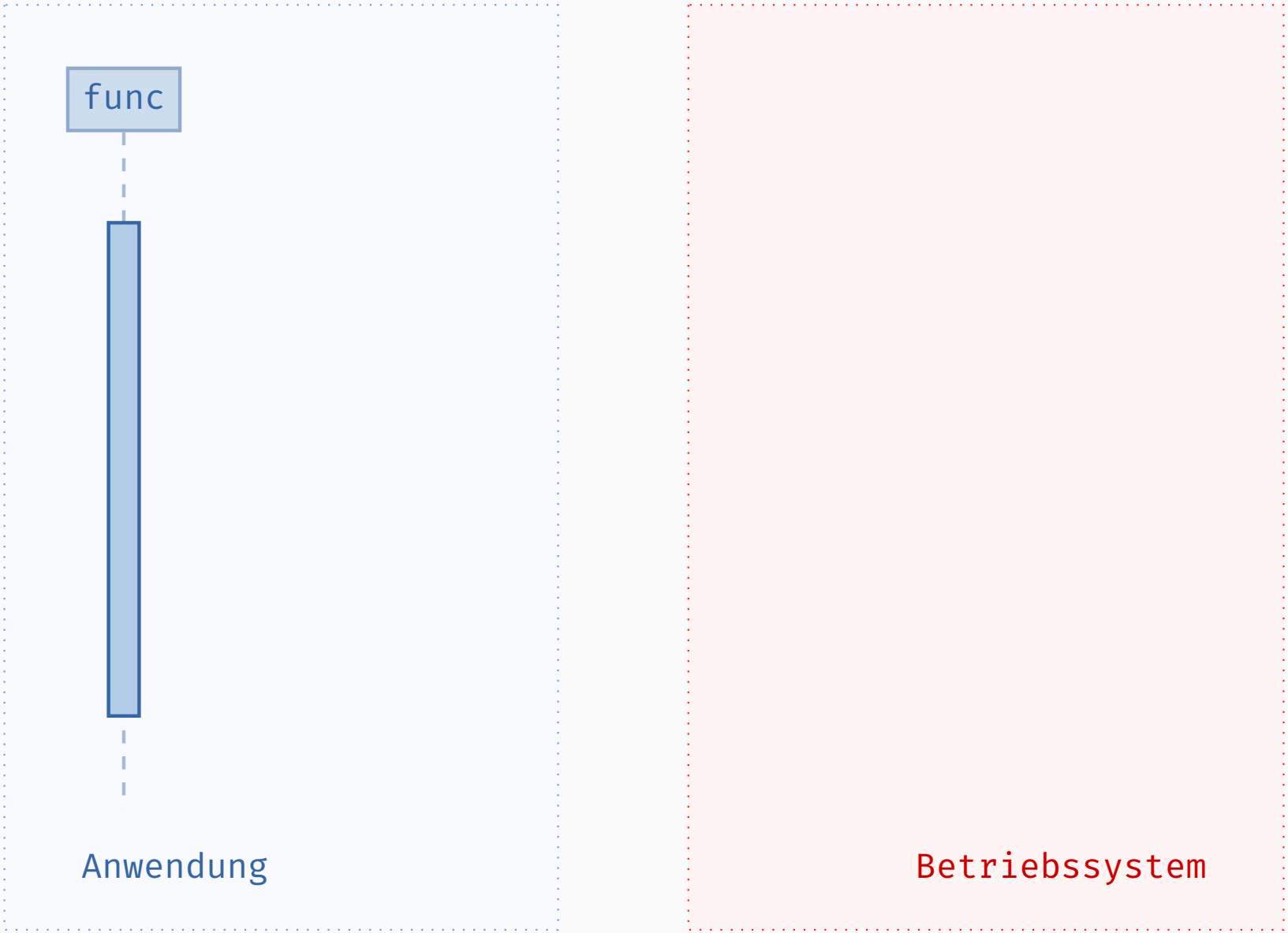
# Systemaufruffunktionen



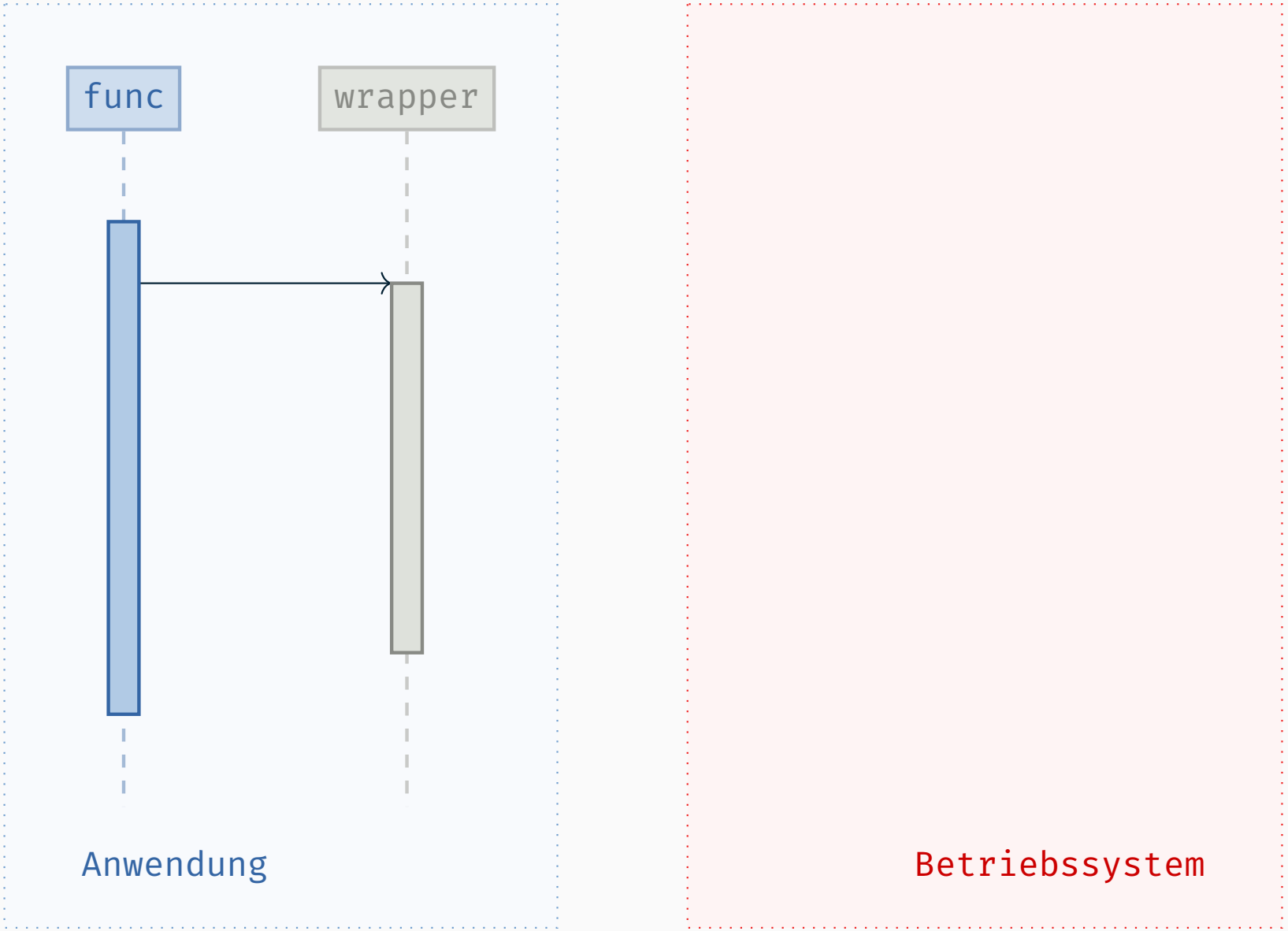
# Systemaufruffunktionen



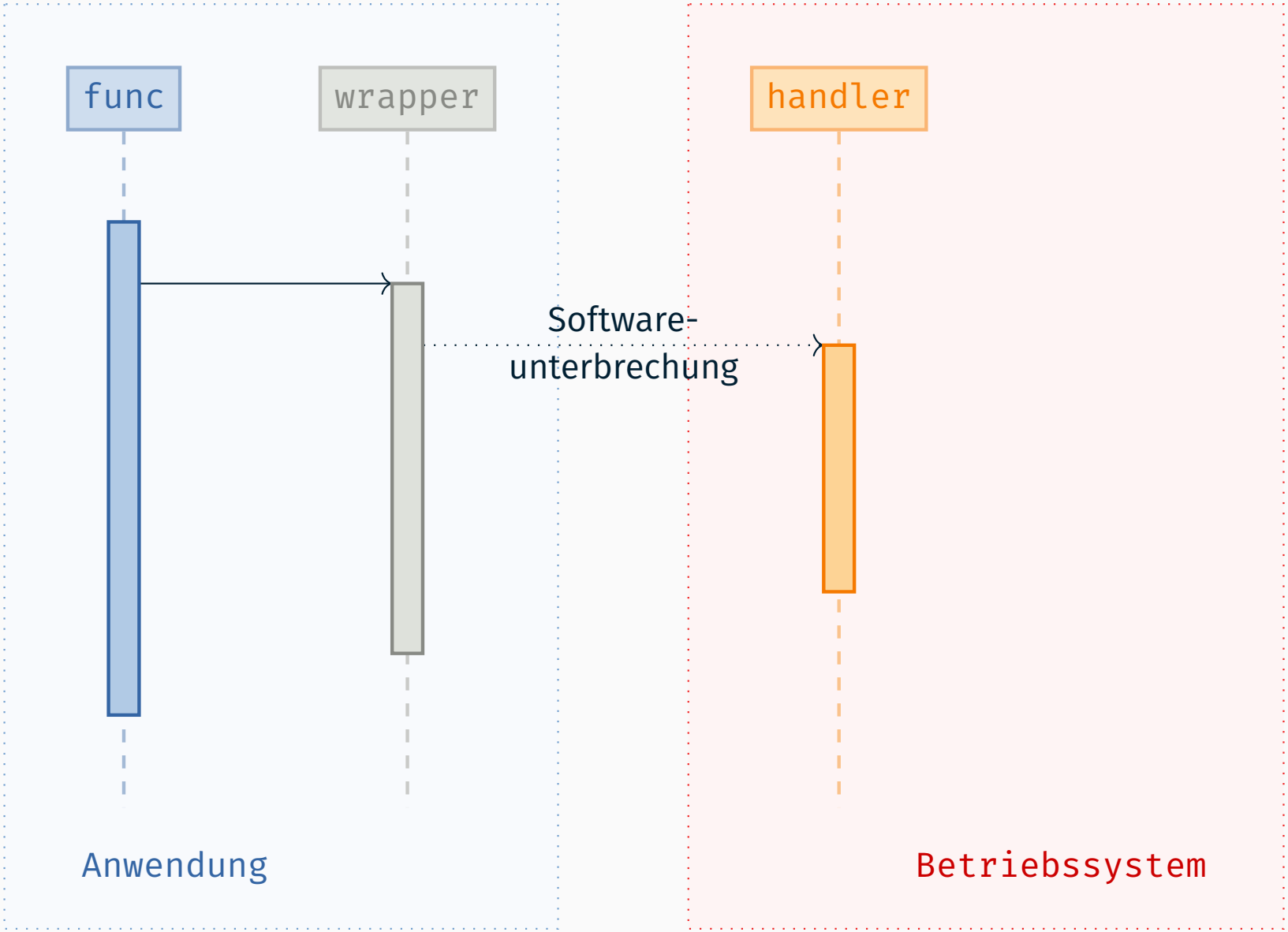
# Ablauf eines Systemaufrufes



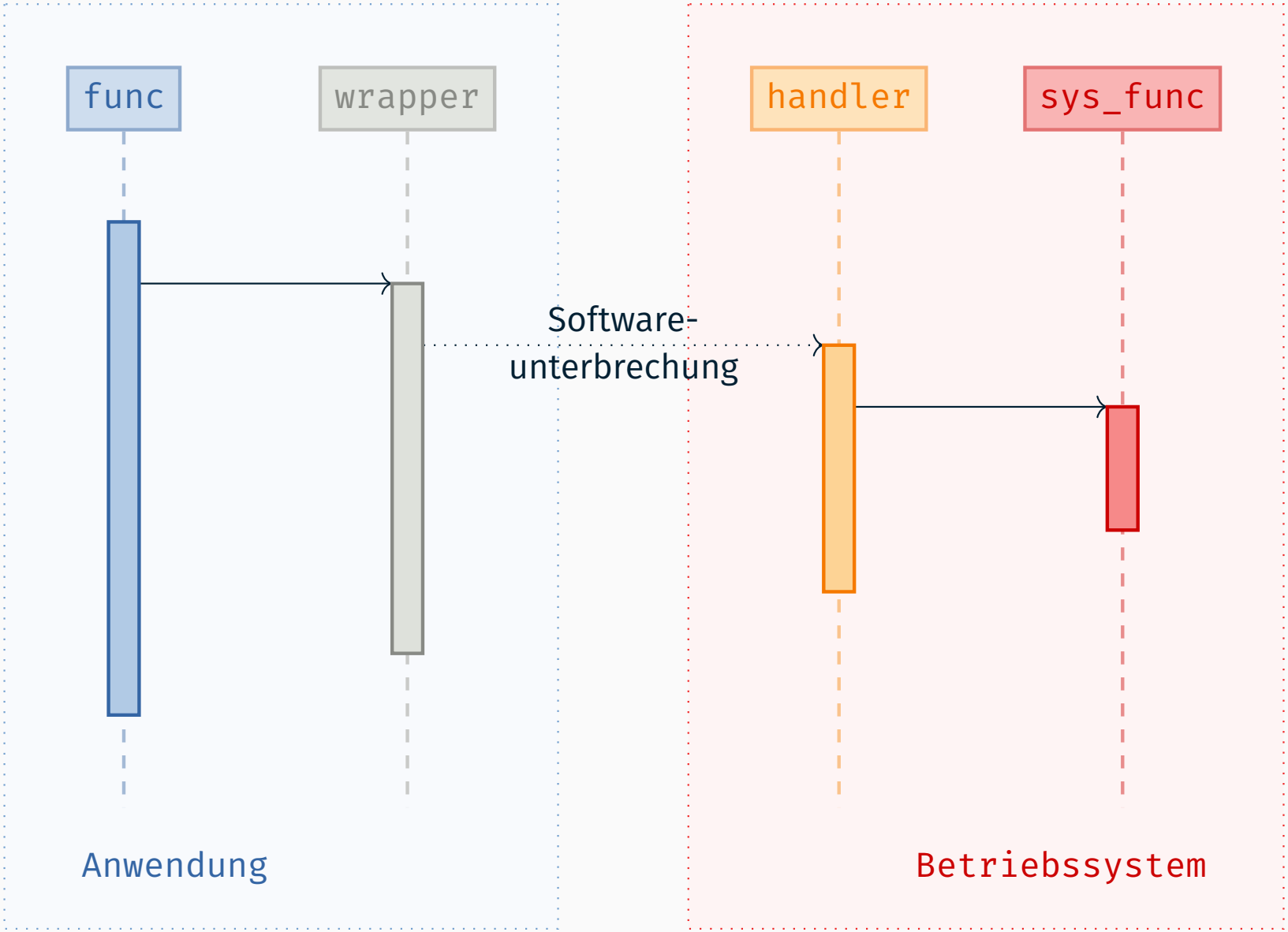
# Ablauf eines Systemaufrufes



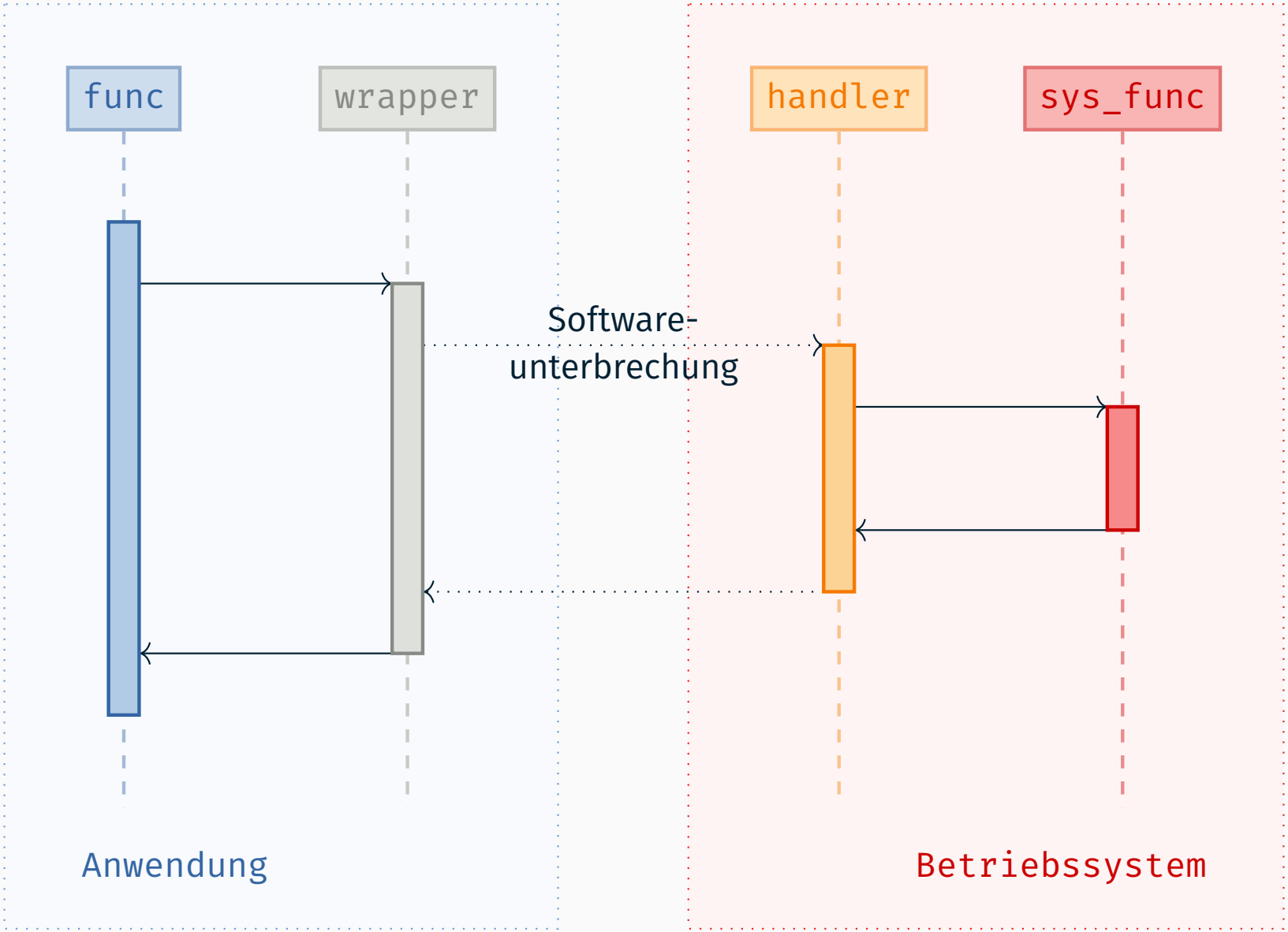
# Ablauf eines Systemaufrufes



# Ablauf eines Systemaufrufes



# Ablauf eines Systemaufrufes



# Ablauf eines Systemaufrufes...?

- Wie werden die Parameter verpackt?
- Wie wechselt man in das Betriebssystem?
- Wie empfängt und verarbeitet das Betriebssystem die Anfrage?
- Wie gibt man das Ergebnis zurück an die Anwendung?

# Systemaufrufmechanismus

Mechanismus zum Wechsel von *user* zu *kernel space*:
Software-Unterbrechungen / Traps

# Systemaufrufmechanismus

Mechanismus zum Wechsel von *user* zu *kernel space*:
Software-Unterbrechungen / Traps

Ein Trap...

- stellt privilegierten Ausführungsmodus sicher
- wechselt mit der Ausführung in die Trap-Behandlungsroutine

# Systemaufrufmechanismus

Mechanismus zum Wechsel von *user* zu *kernel space*:
Software-Unterbrechungen / Traps

Ein Trap...

- stellt privilegierten Ausführungsmodus sicher
- wechselt mit der Ausführung in die Trap-Behandlungsroutine

Die Trap-Behandlungsroutine bestimmt mittels Systemaufrufnummer und Systemaufruftabelle die angeforderte Funktion und führt diese aus.

# Systemaufrufe auf Hardware?

OctoPOS unterstützt drei Platformen:

`x86guest`: 32-Bit-Anwendung auf Linux-Systemaufruf-API

`x64native`: Bare-Metal-Betriebssystem auf x86_64

`leon`: Bare-Metal-Betriebssystem für SPARC v8 LEON

# x86guest

Interrupts werden durch UNIX-Signale emuliert.

# x86guest

Interrupts werden durch UNIX-Signale emuliert.

Traps daher folgendermaßen implementiert:

- freies Systemaufrufsignal
- Registrierung eines neuen Signalhandlers in OctoPOS
- Verwenden des `rt_tgsigqueueinfo` Linux-Systemaufrufes zum Senden eines Signales mit Daten (für Parameterübergabe)

# x64native

Trap mithilfe von Interruptinstruktion: `int 0x30`

# x64native

Trap mithilfe von Interruptinstruktion: `int 0x30`

Warum kein `sysenter` / `sysleave` oder `syscall` / `sysret`?

- Long Mode für Intel und AMD: nur `syscall` / `sysret`
- `sysret` wechselt zu Privilegienlevel 3

# x64native

Trap mithilfe von Interruptinstruktion: `int 0x30`

Warum kein `sysenter` / `sysleave` oder `syscall` / `sysret`?

- Long Mode für Intel und AMD: nur `syscall` / `sysret`
- `sysret` wechselt zu Privilegienlevel 3
  - ↯ OctoPOS nutzt nur Privilegienlevel 0

# x64native

Trap mithilfe von Interruptinstruktion: `int 0x30`

Warum kein `sysenter` / `sysleave` oder `syscall` / `sysret`?

- Long Mode für Intel und AMD: nur `syscall` / `sysret`
- `sysret` wechselt zu Privilegienlevel 3
  - ↯ OctoPOS nutzt nur Privilegienlevel 0

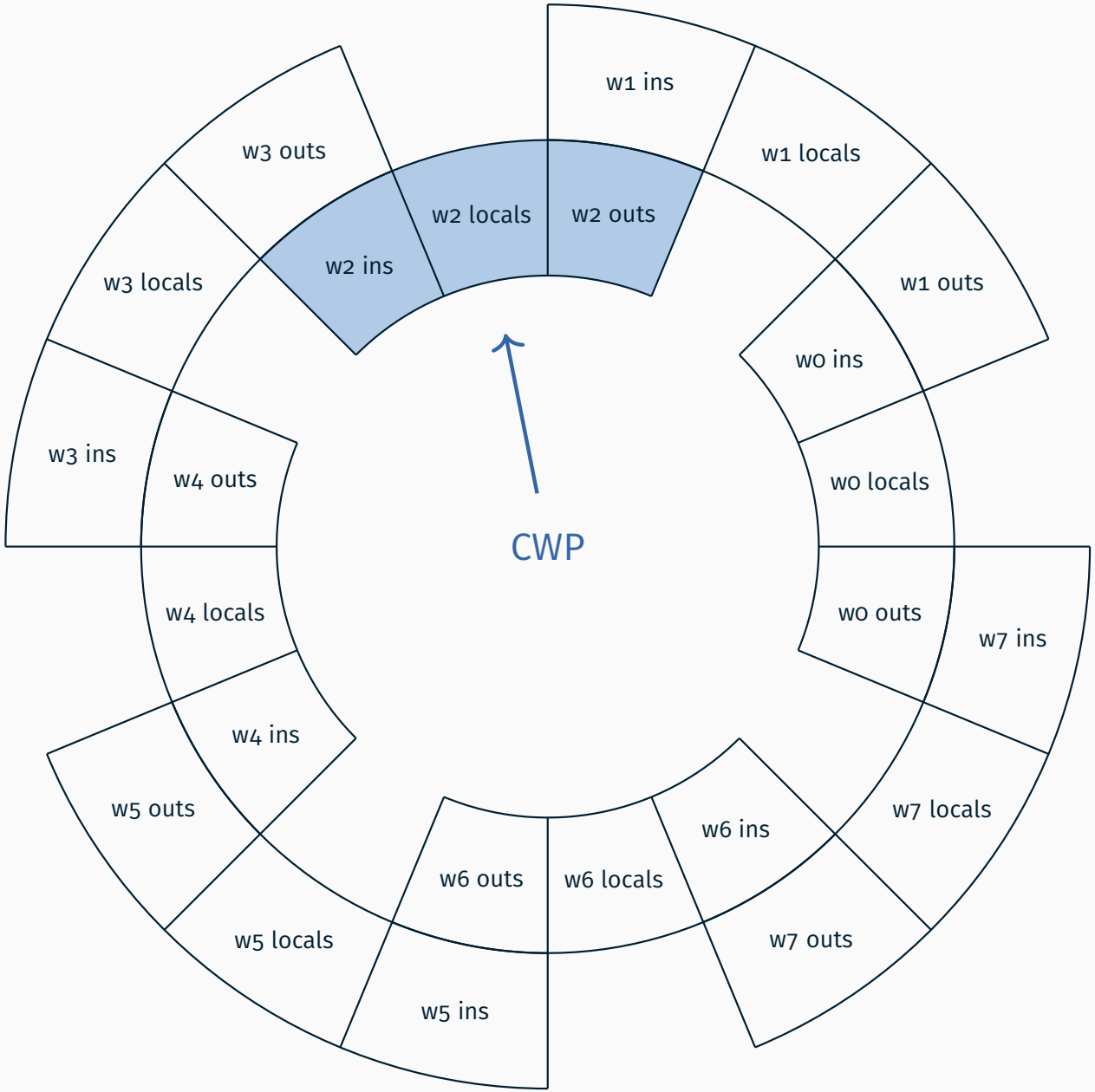
Traps daher folgendermaßen implementiert:

- neue Interruptbehandlung für `int 0x30`
- Parameter-/Ergebnisübergabe in Registern

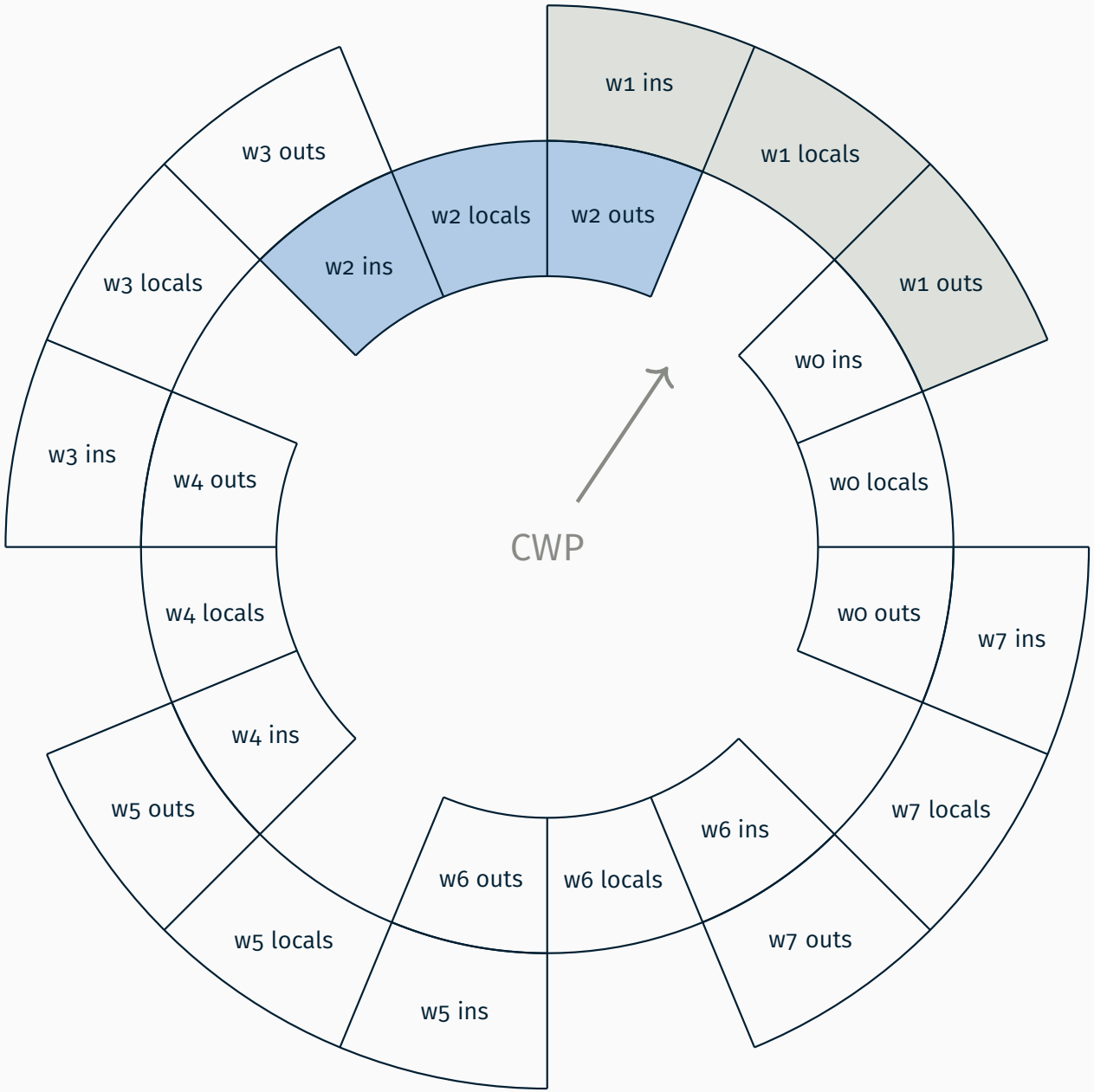
Trap mithilfe von Trapinstruktion: `ta 0x10`

# SPARC

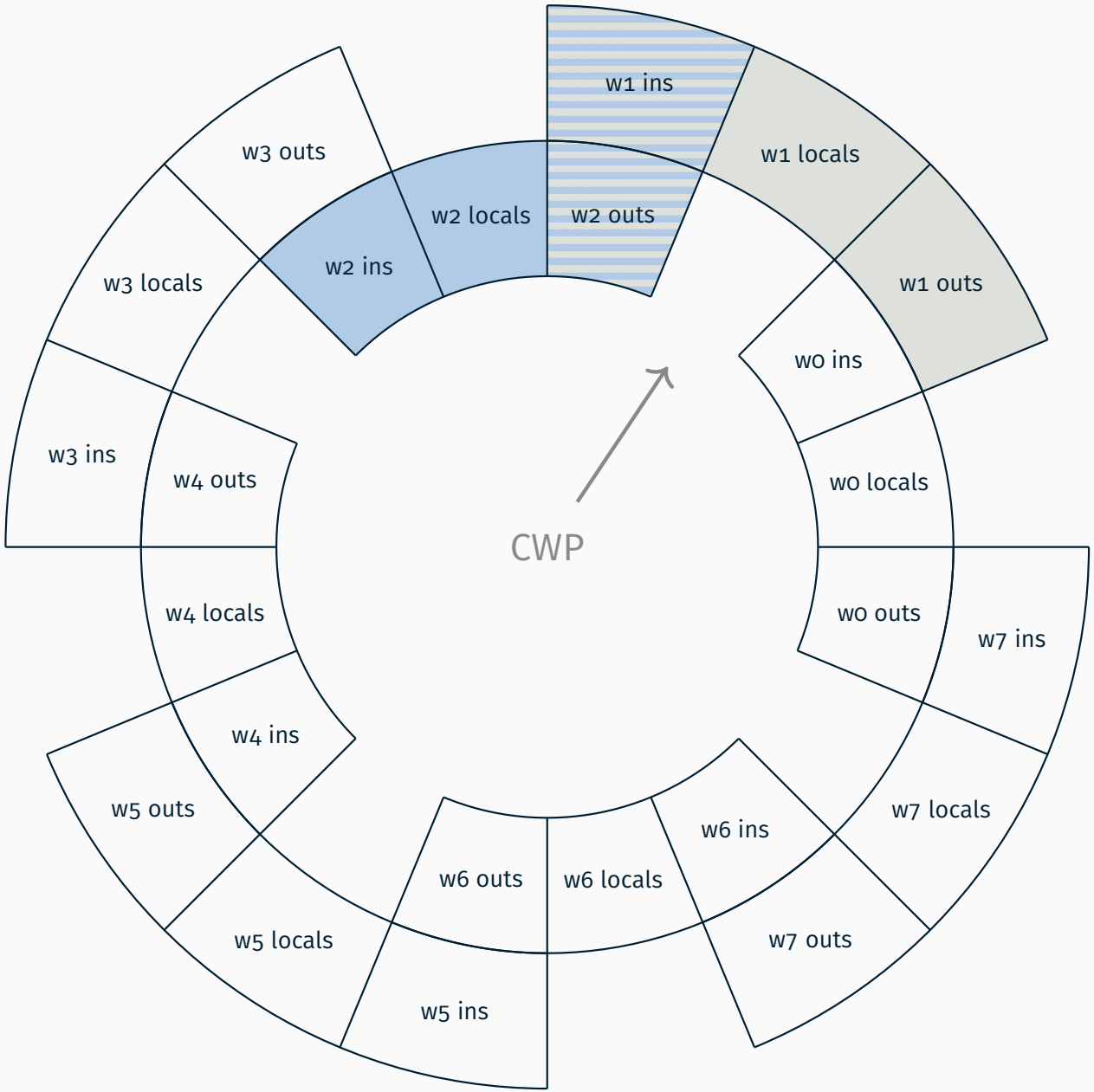
# SPARC und das Registerrad



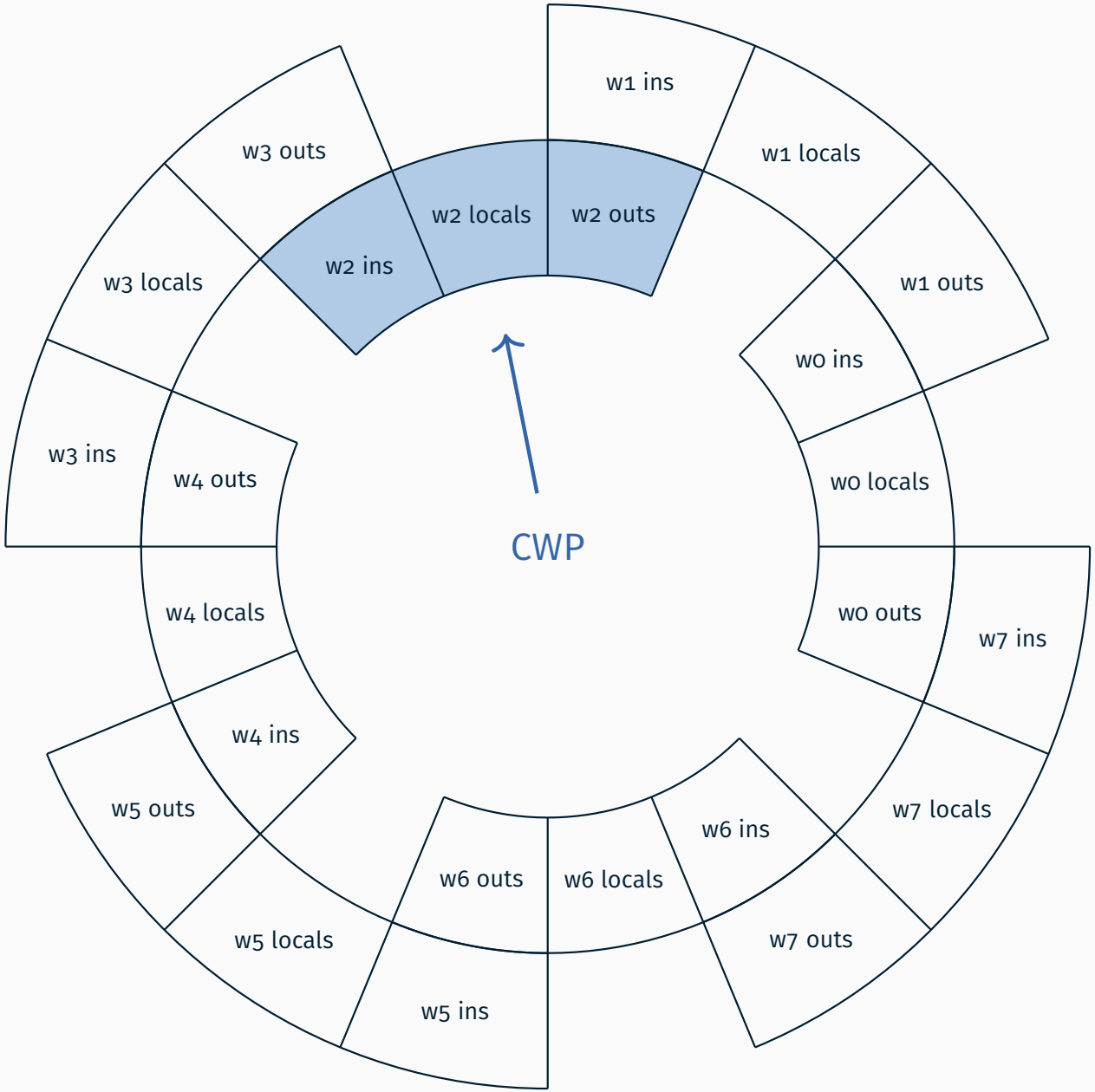
# SPARC und das Registerrad



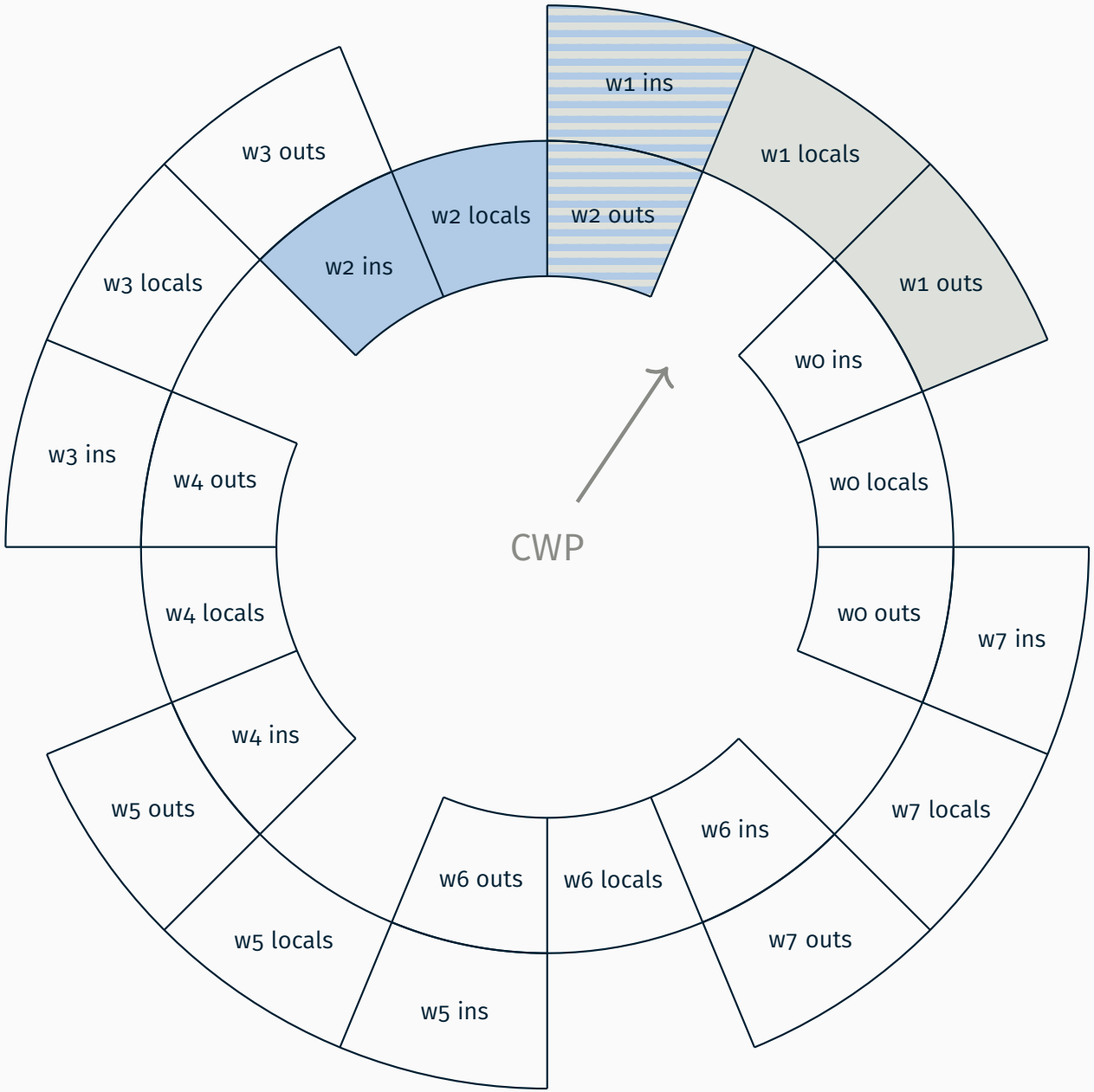
# SPARC und das Registerrad



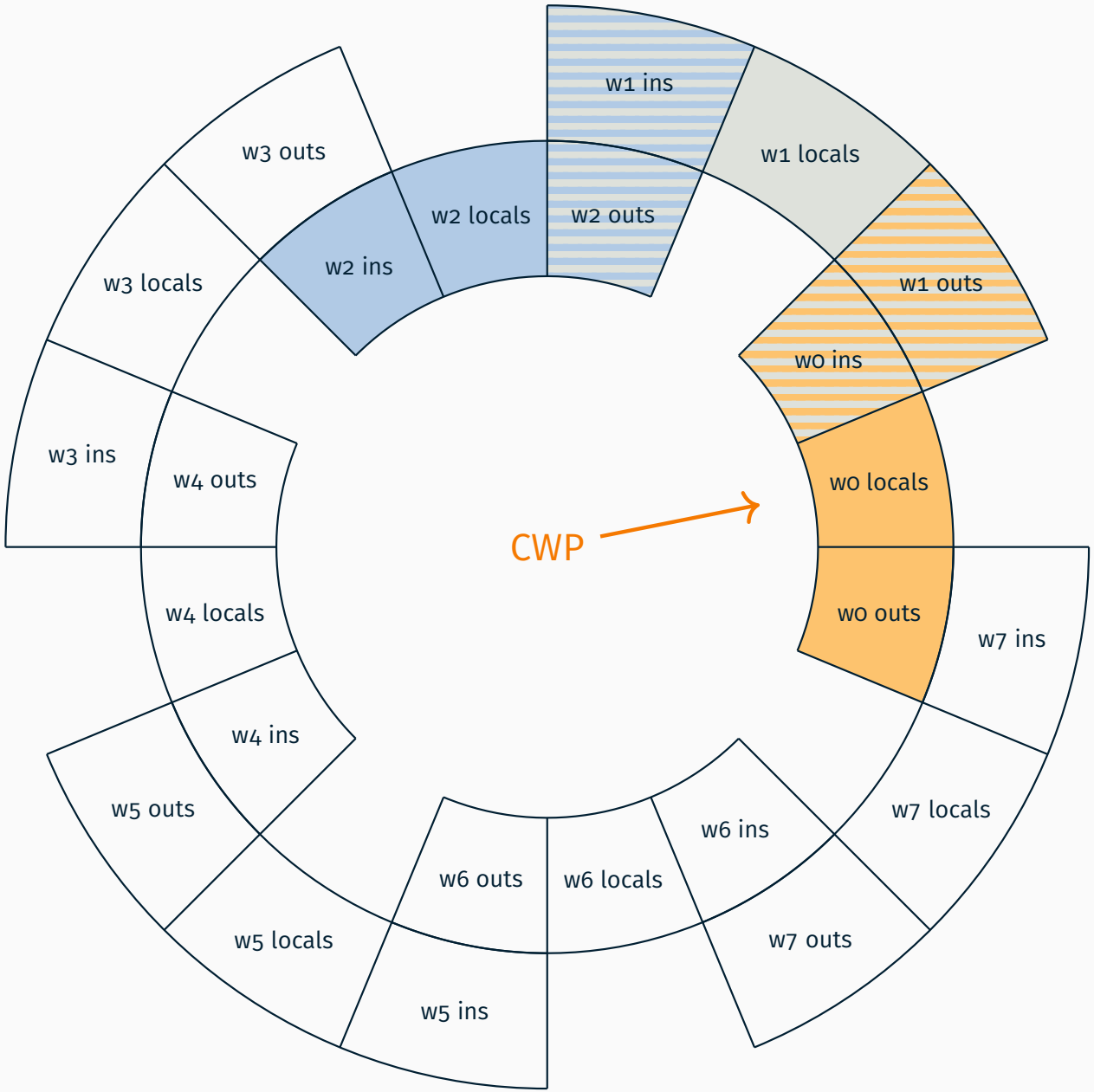
# SPARC und das Registerrad



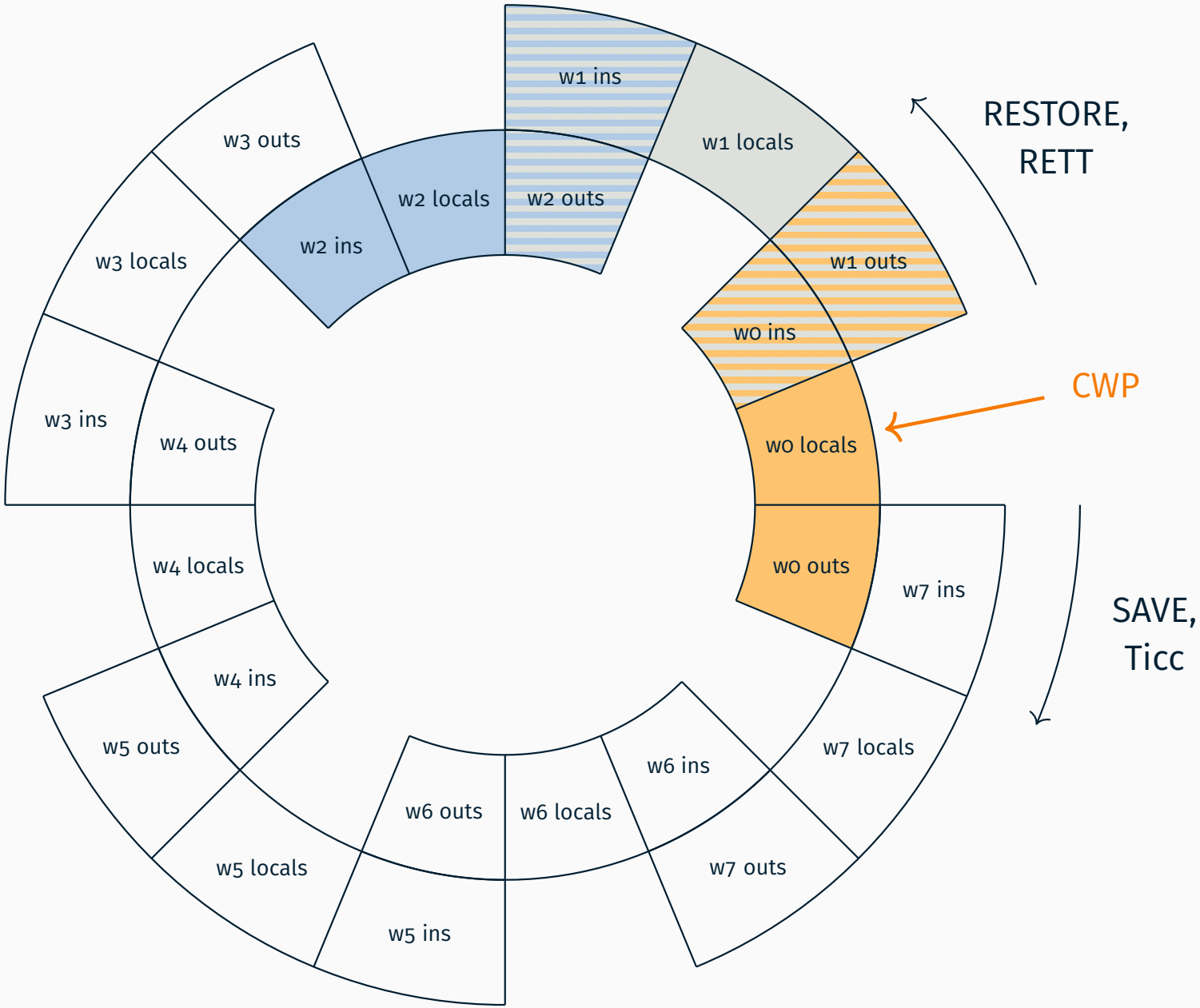
# SPARC und das Registerrad



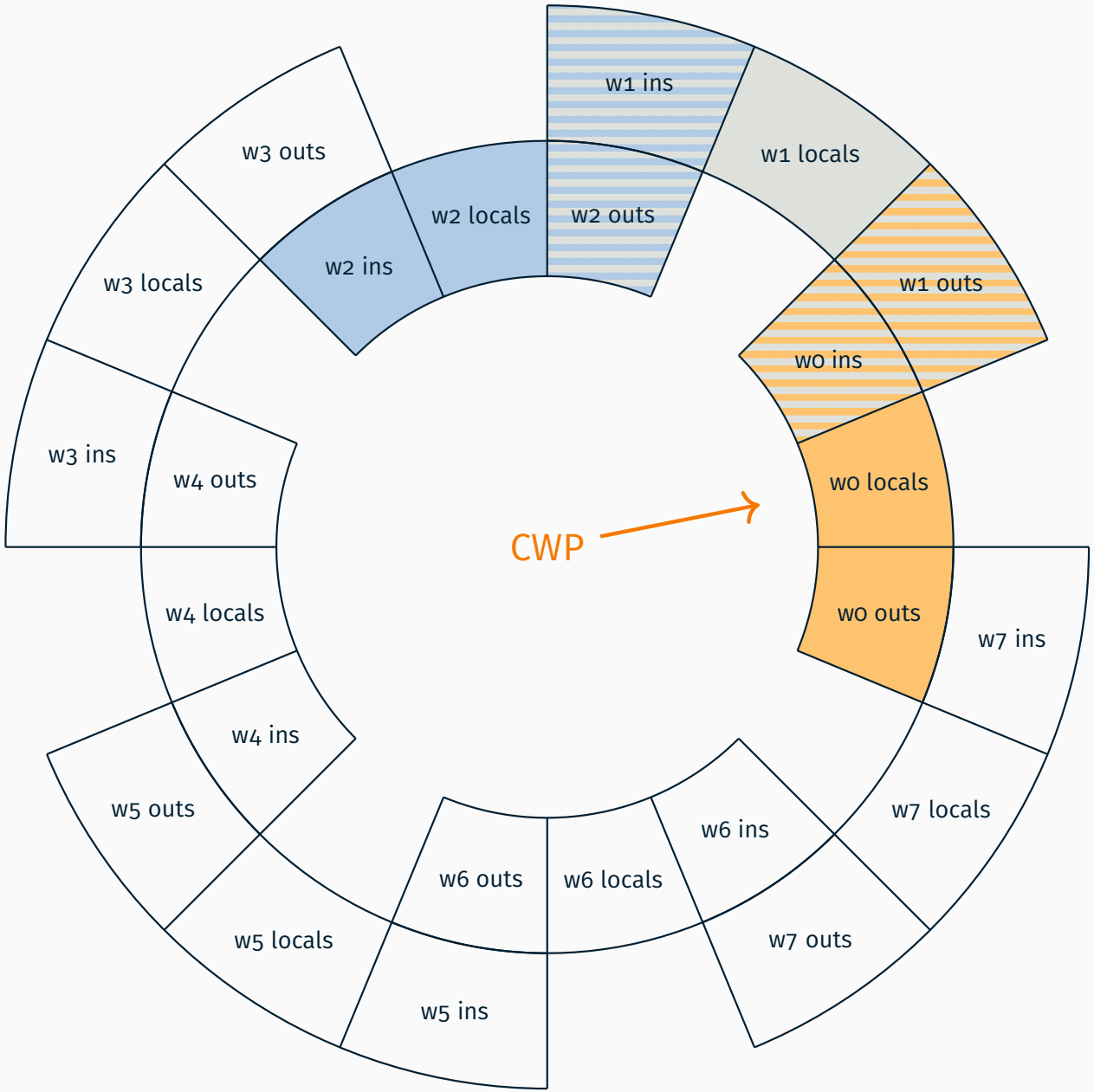
# SPARC und das Registerrad



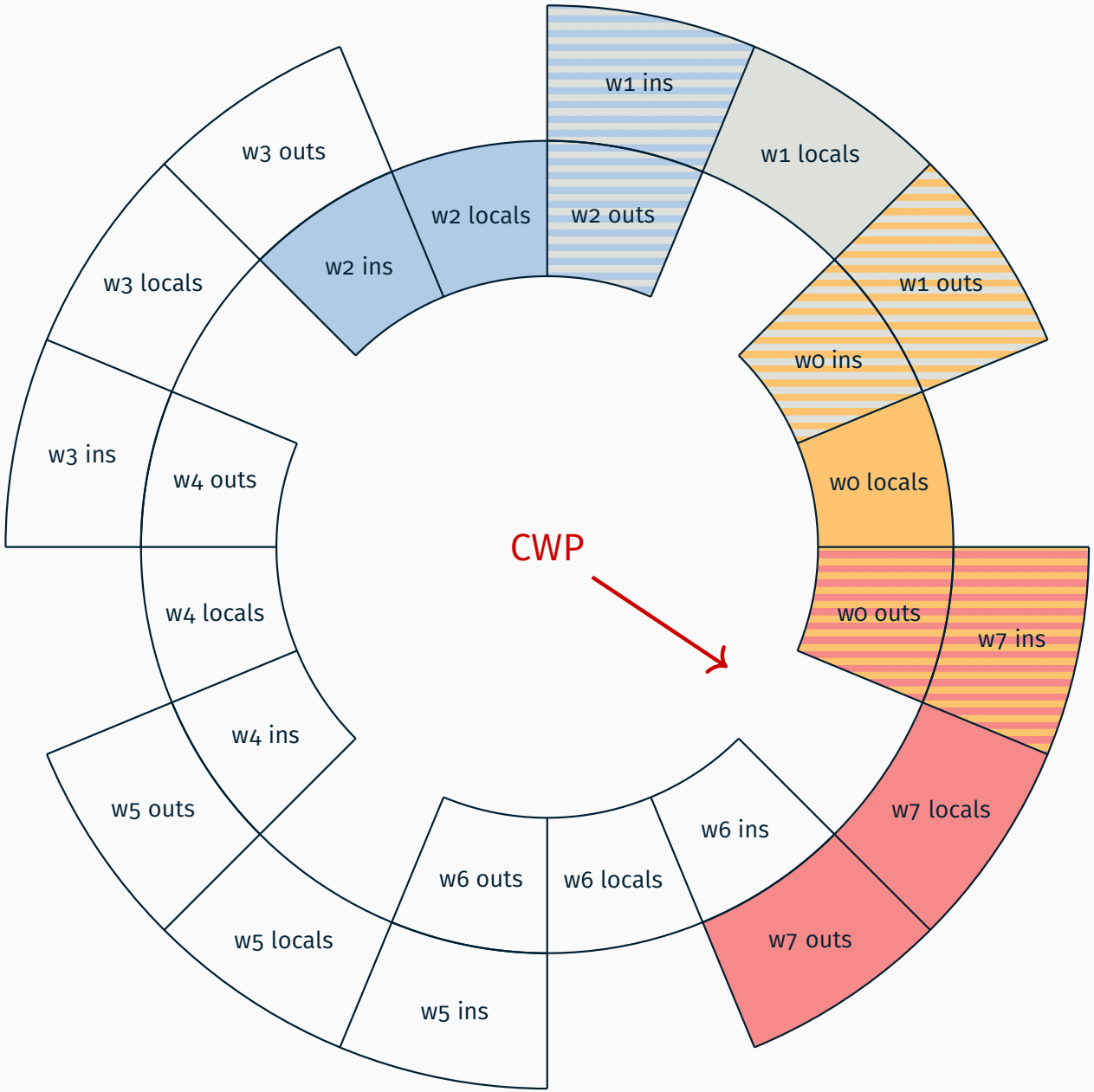
# SPARC und das Registerrad



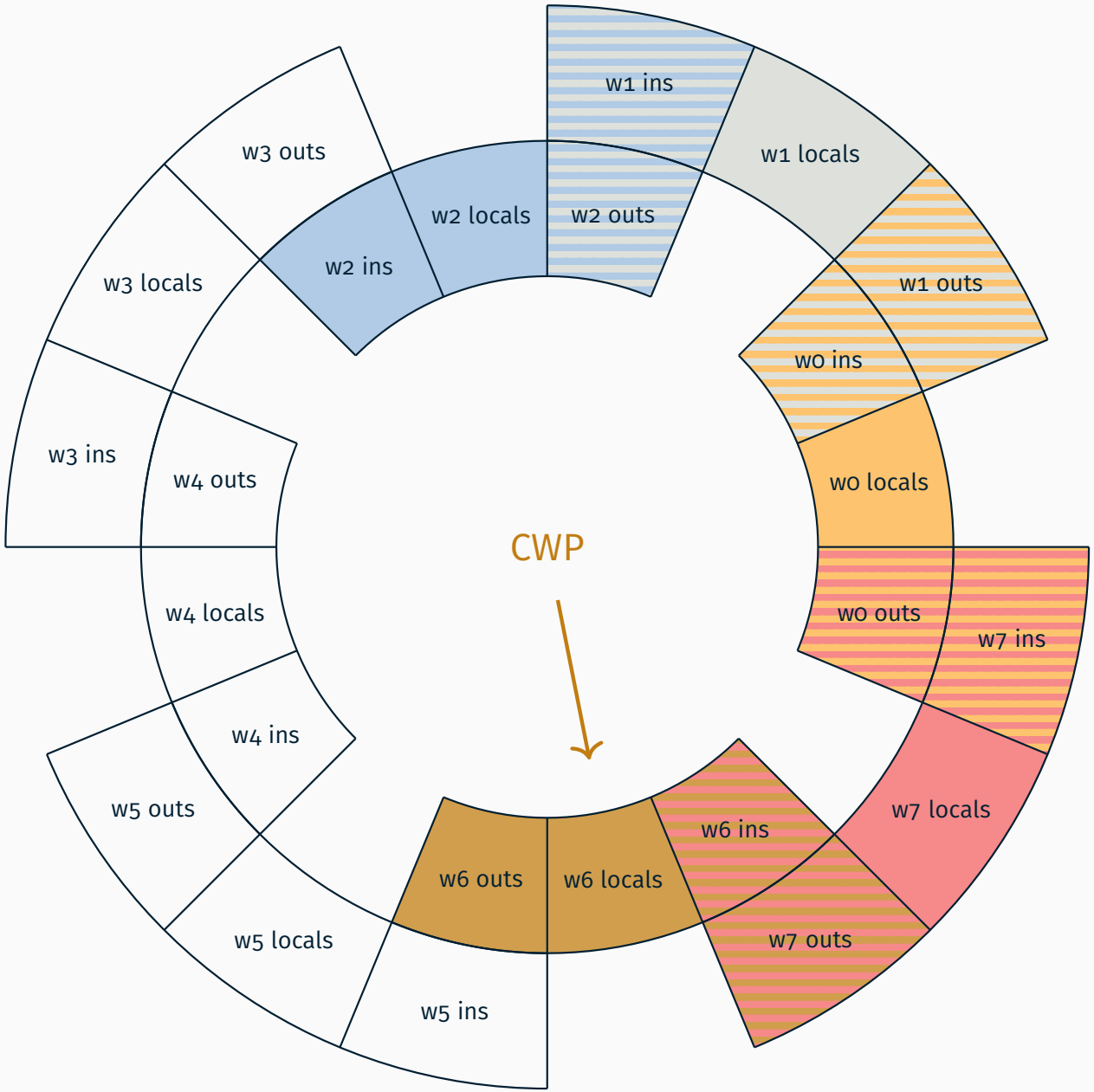
# SPARC und das Registerrad



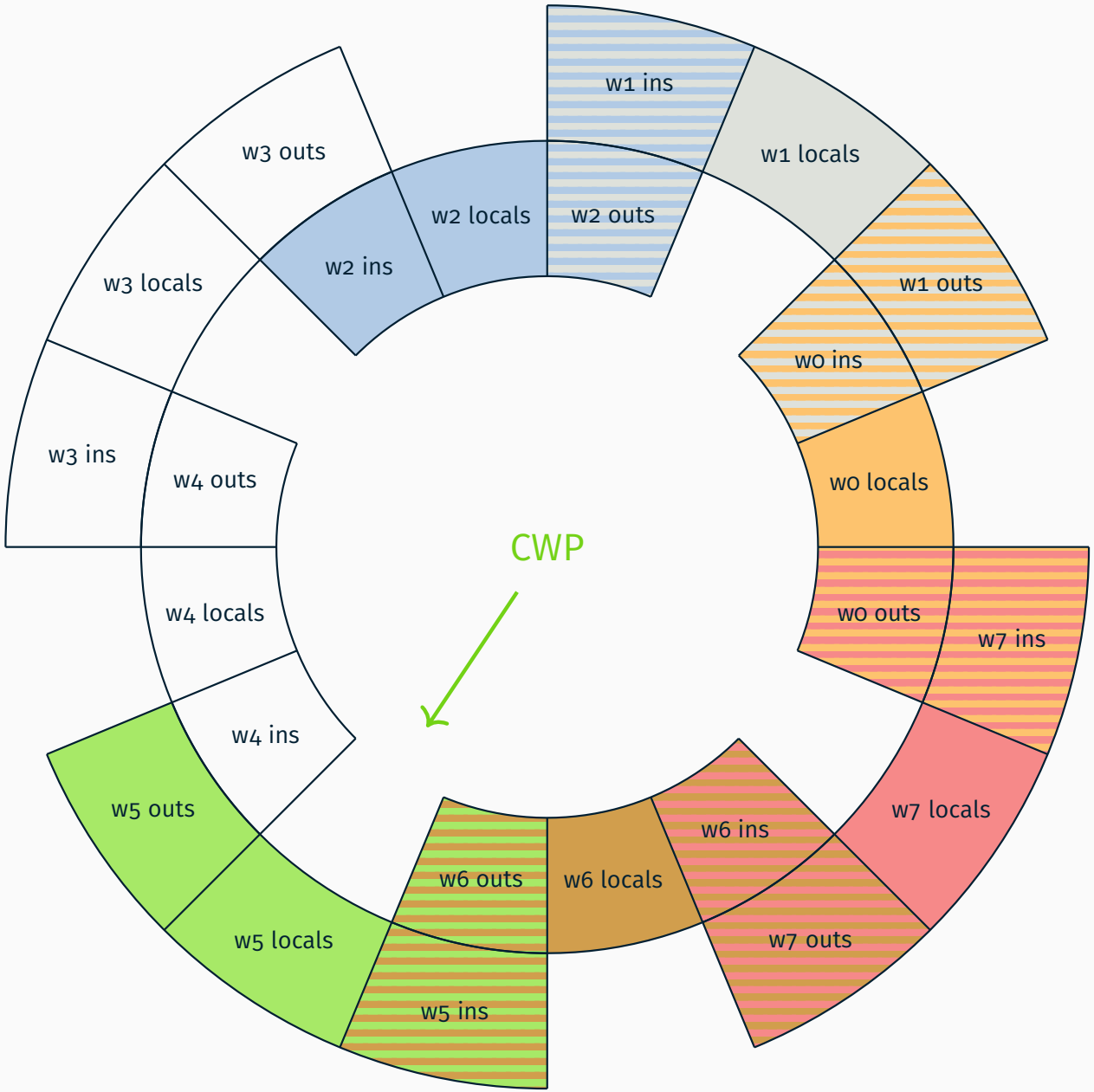
# SPARC und das Registerrad



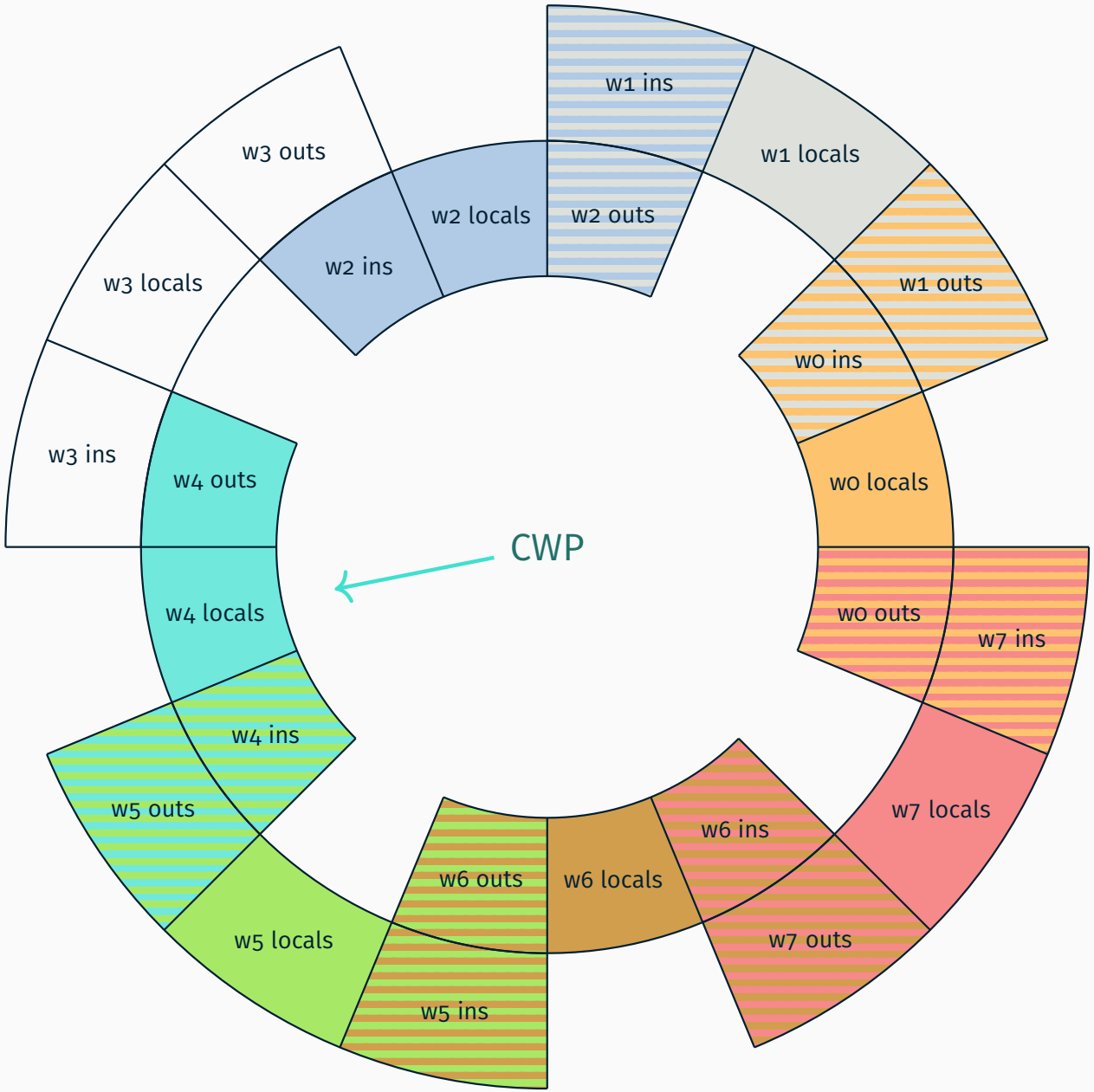
# SPARC und das Registerrad



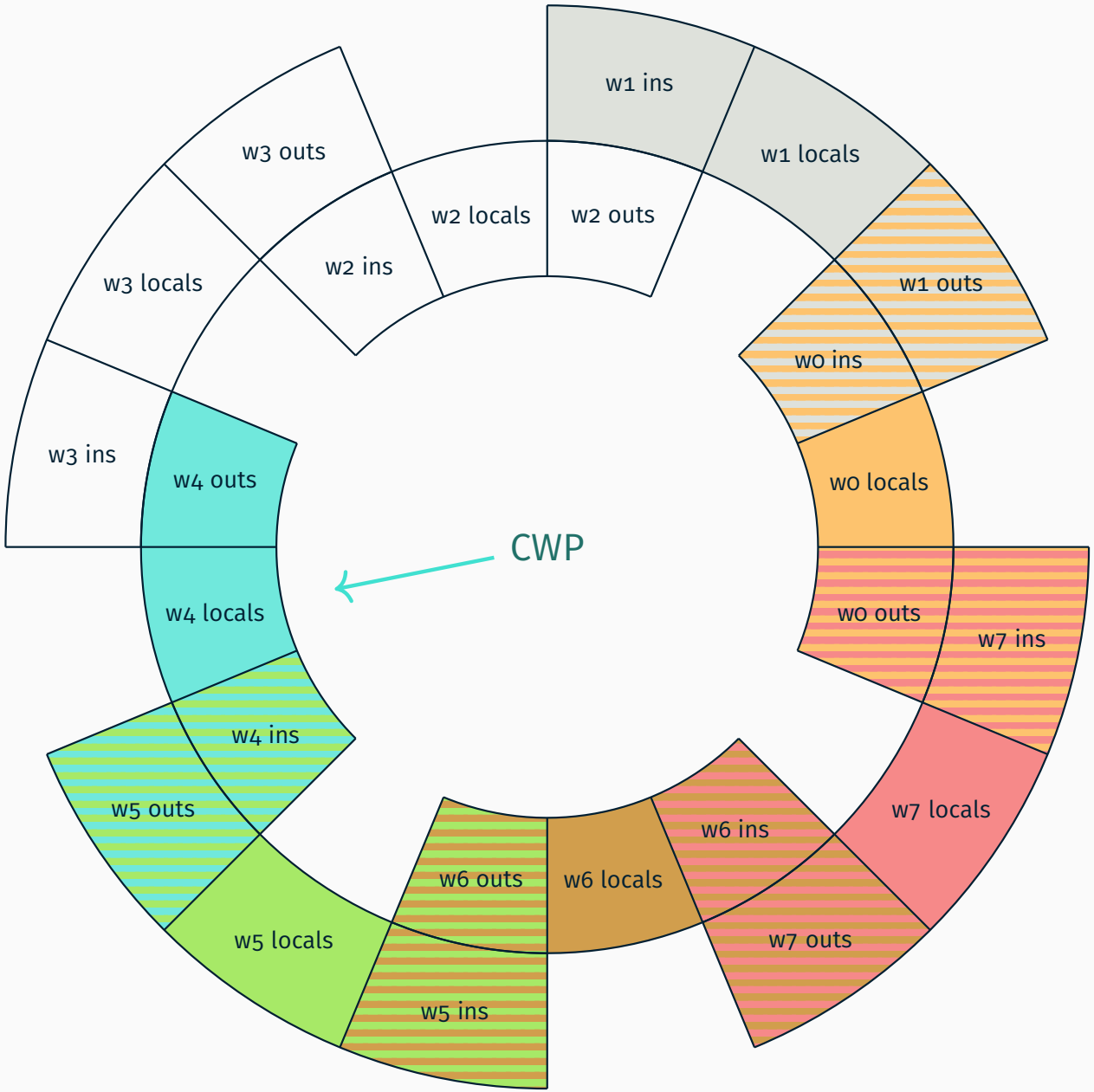
# SPARC und das Registerrad



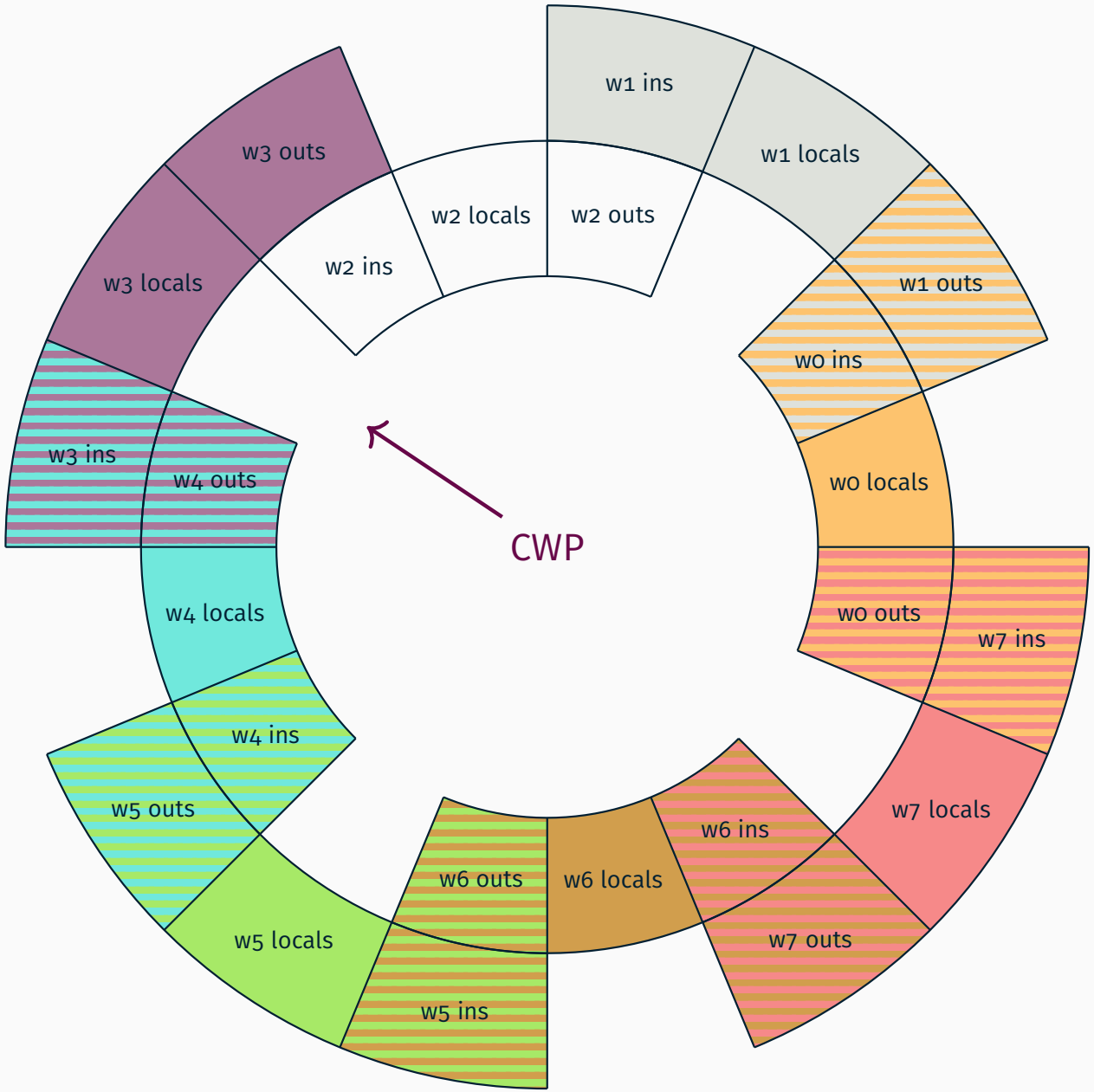
# Window Overflow



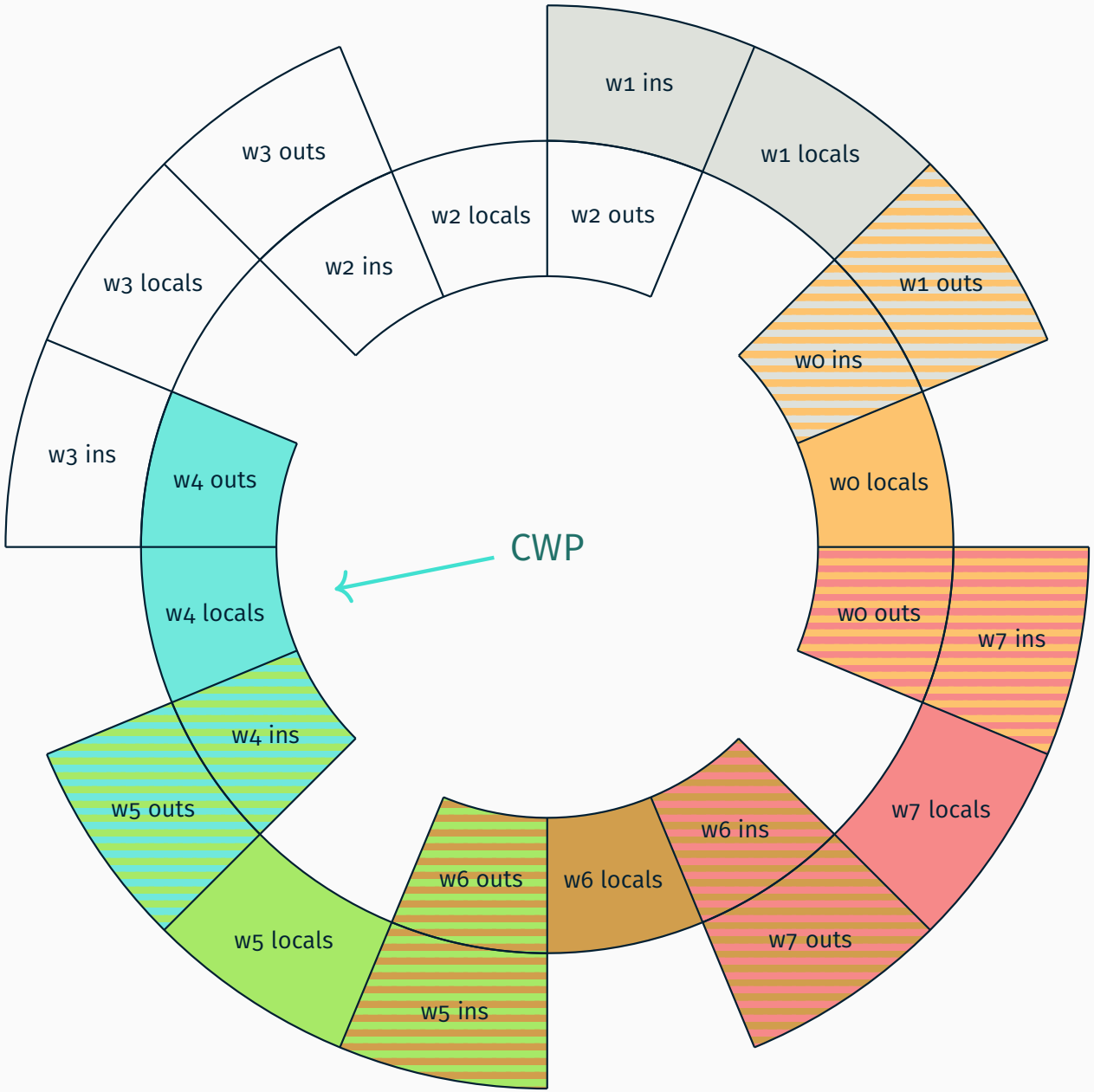
# Window Overflow



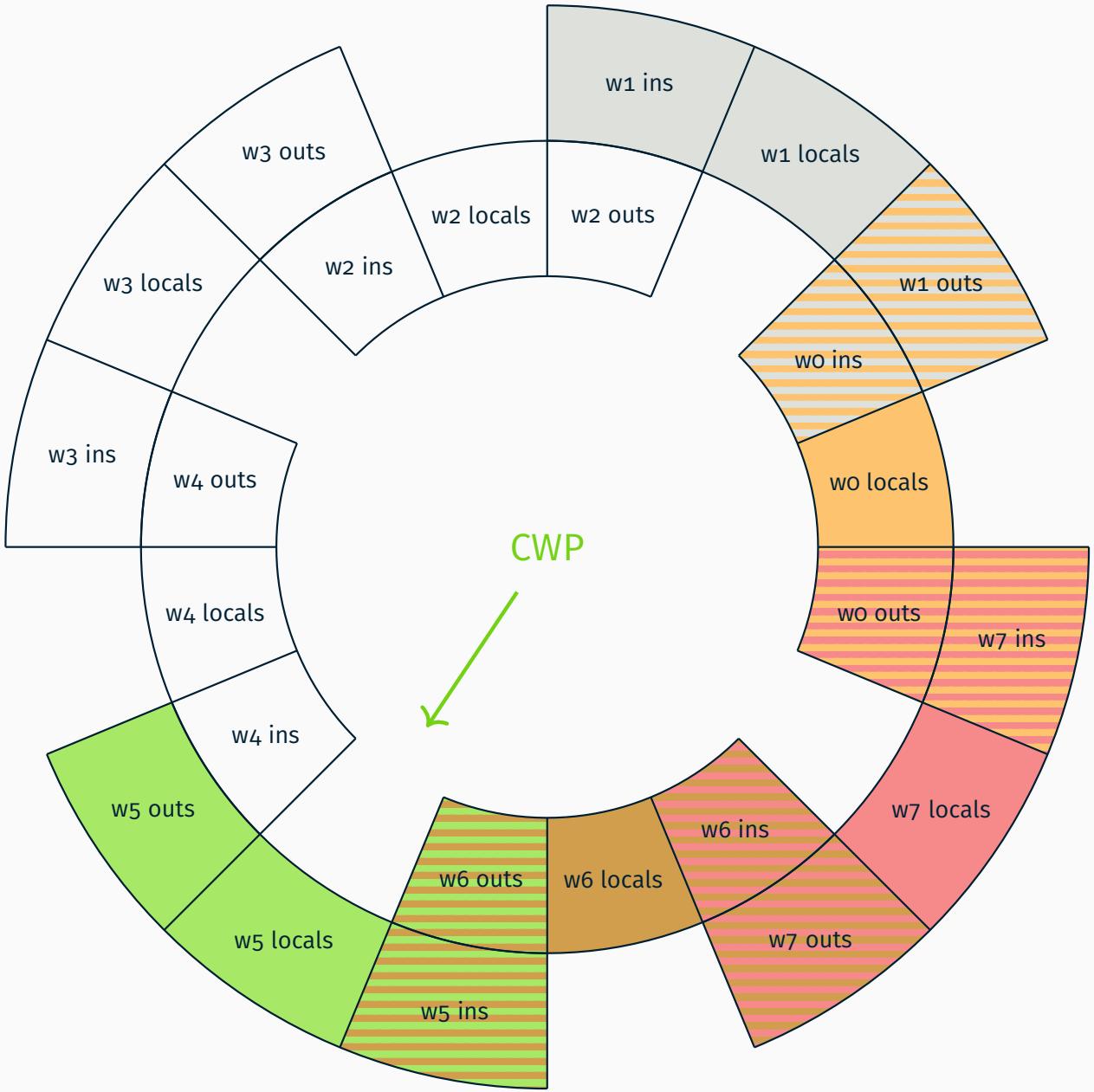
# Window Overflow



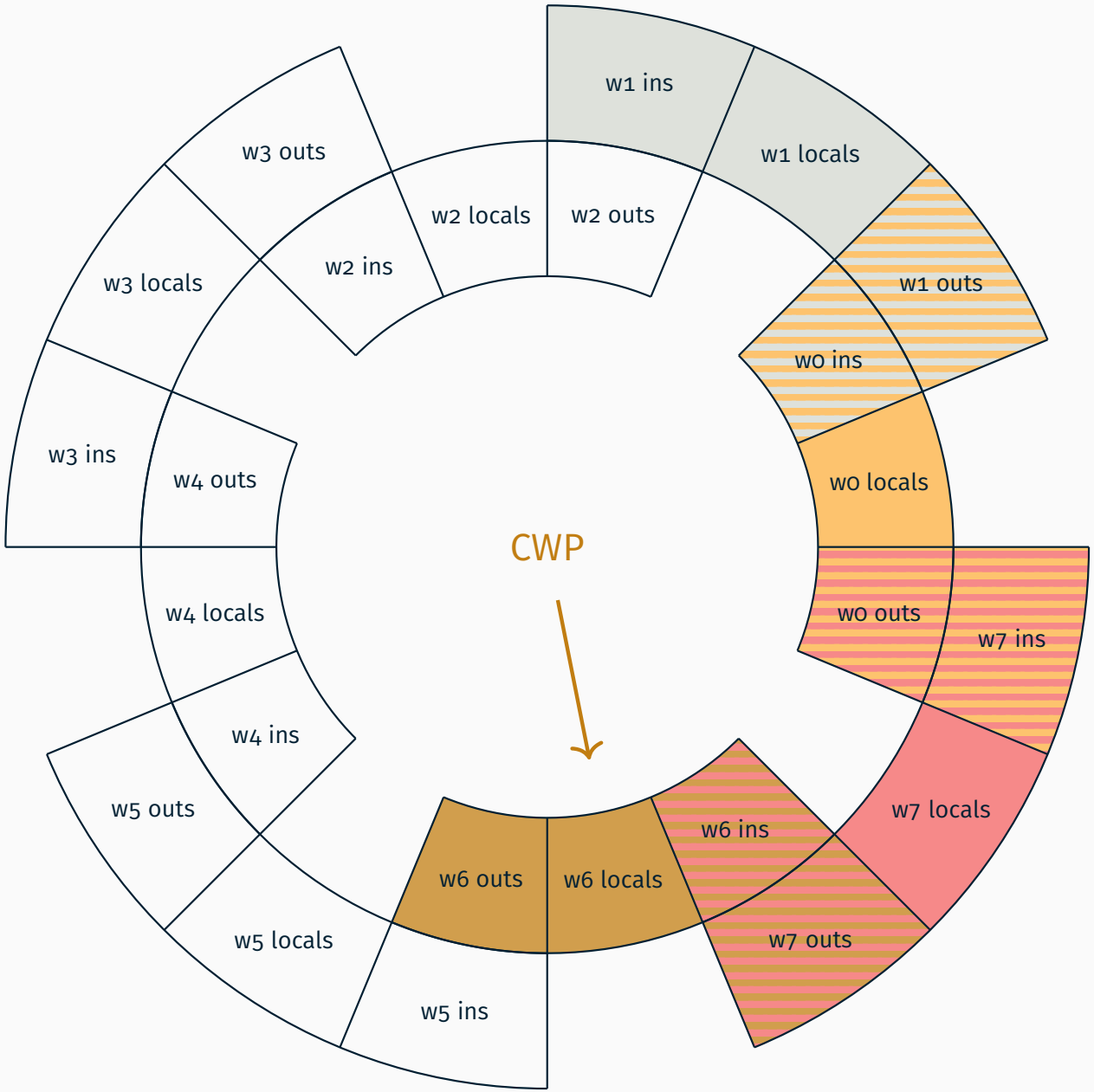
# SPARC und das Registerrad



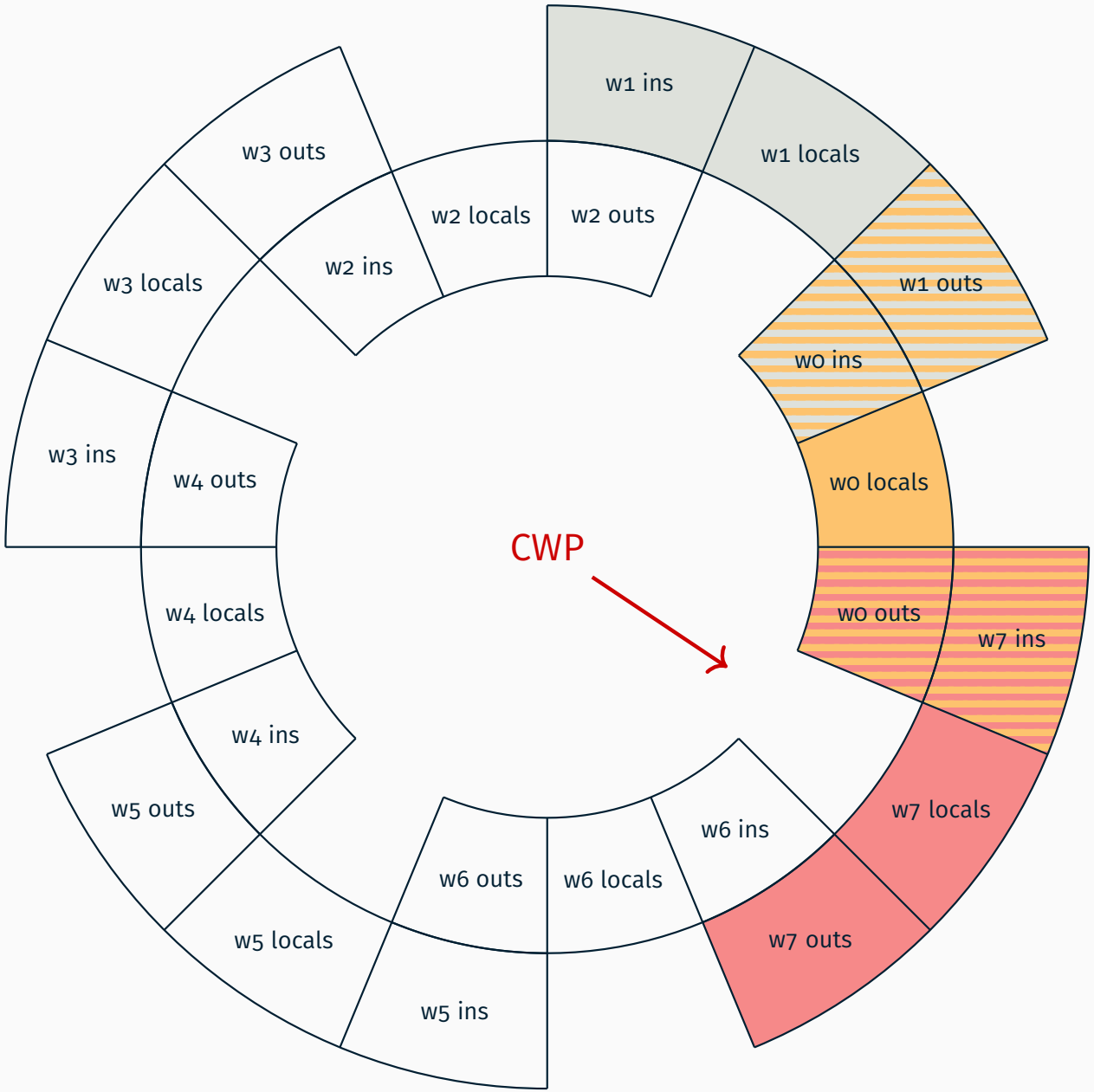
# SPARC und das Registerrad



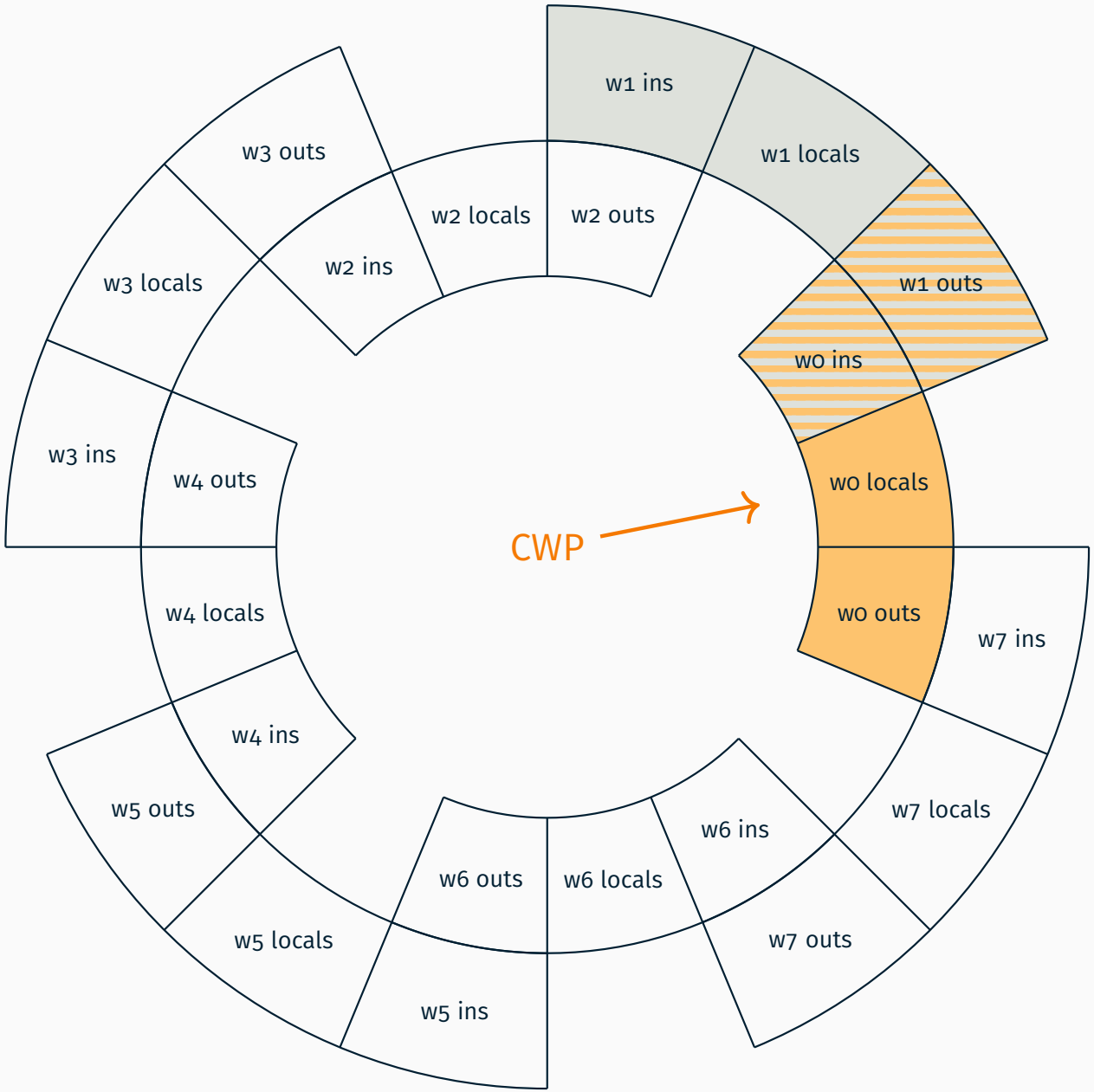
# SPARC und das Registerrad



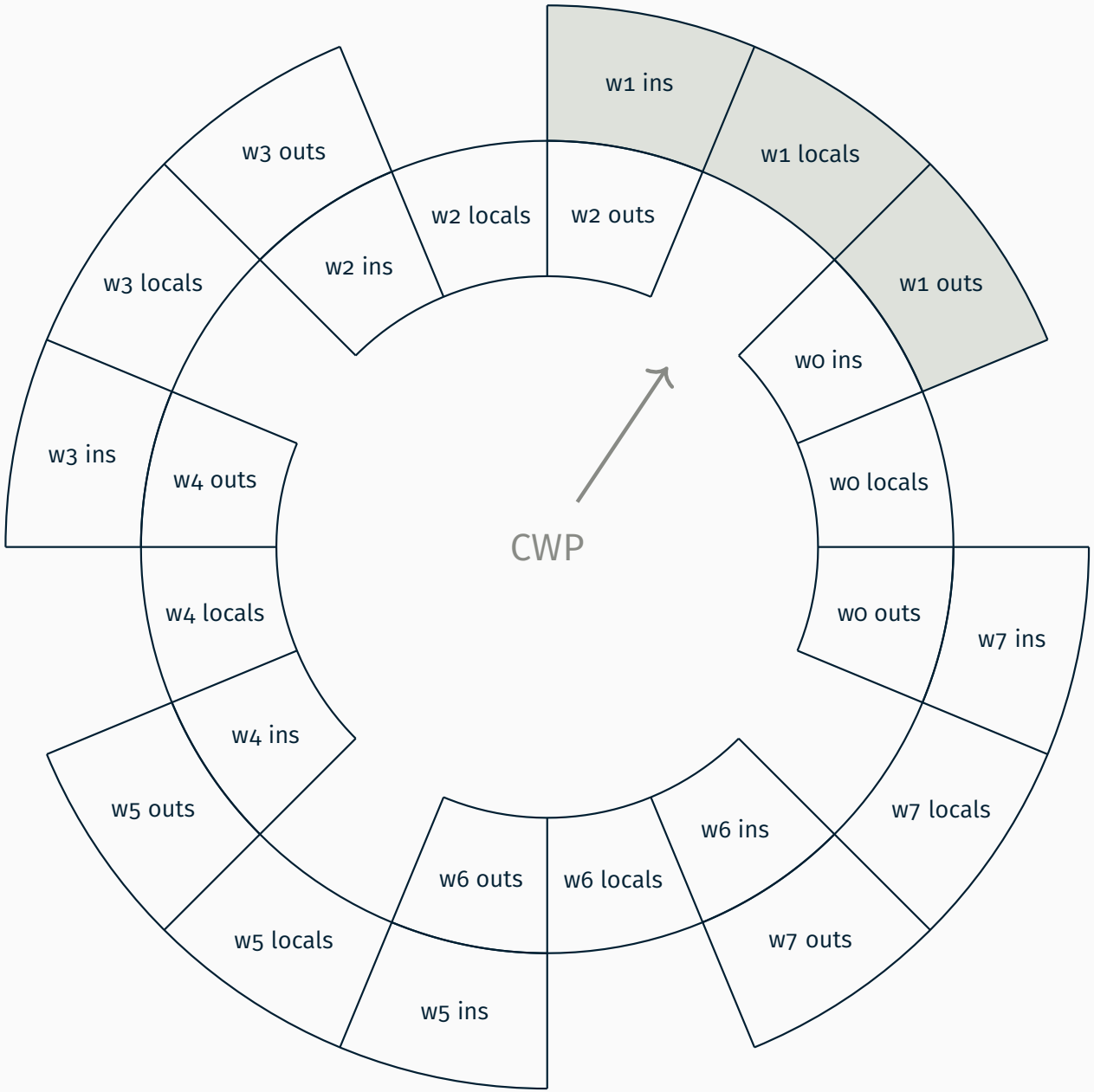
# SPARC und das Registerrad



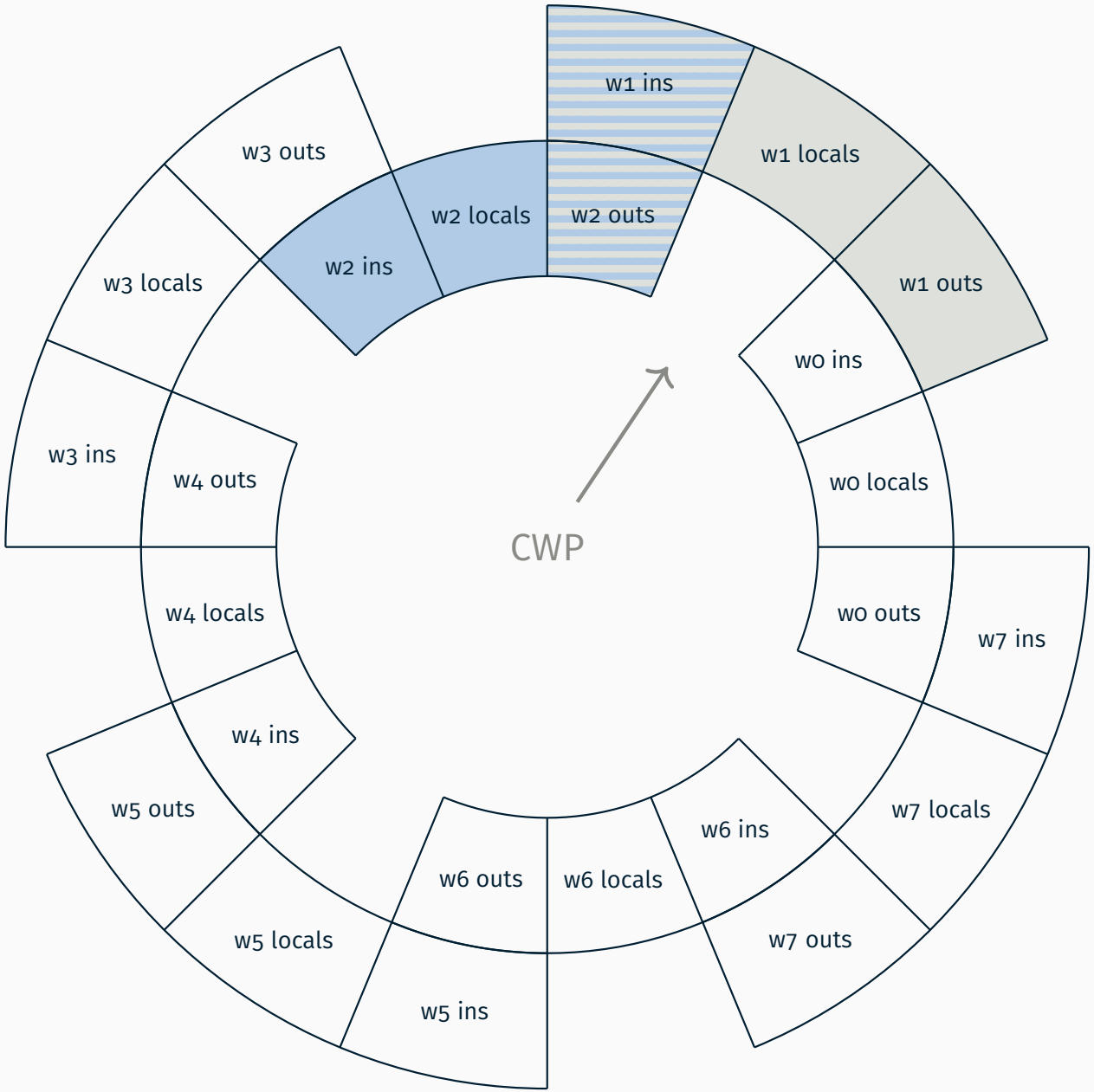
# SPARC und das Registerrad



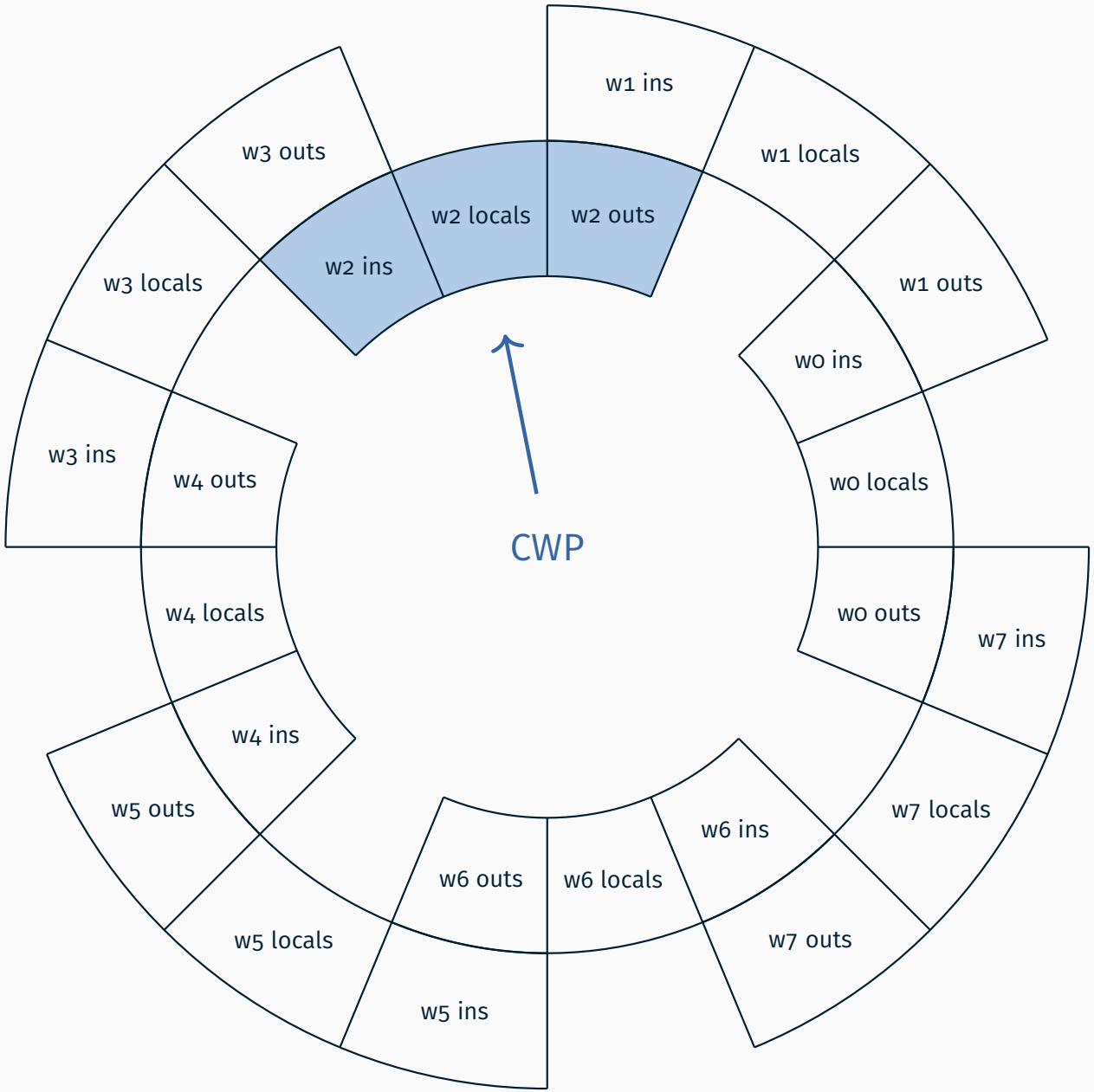
# Window Underflow



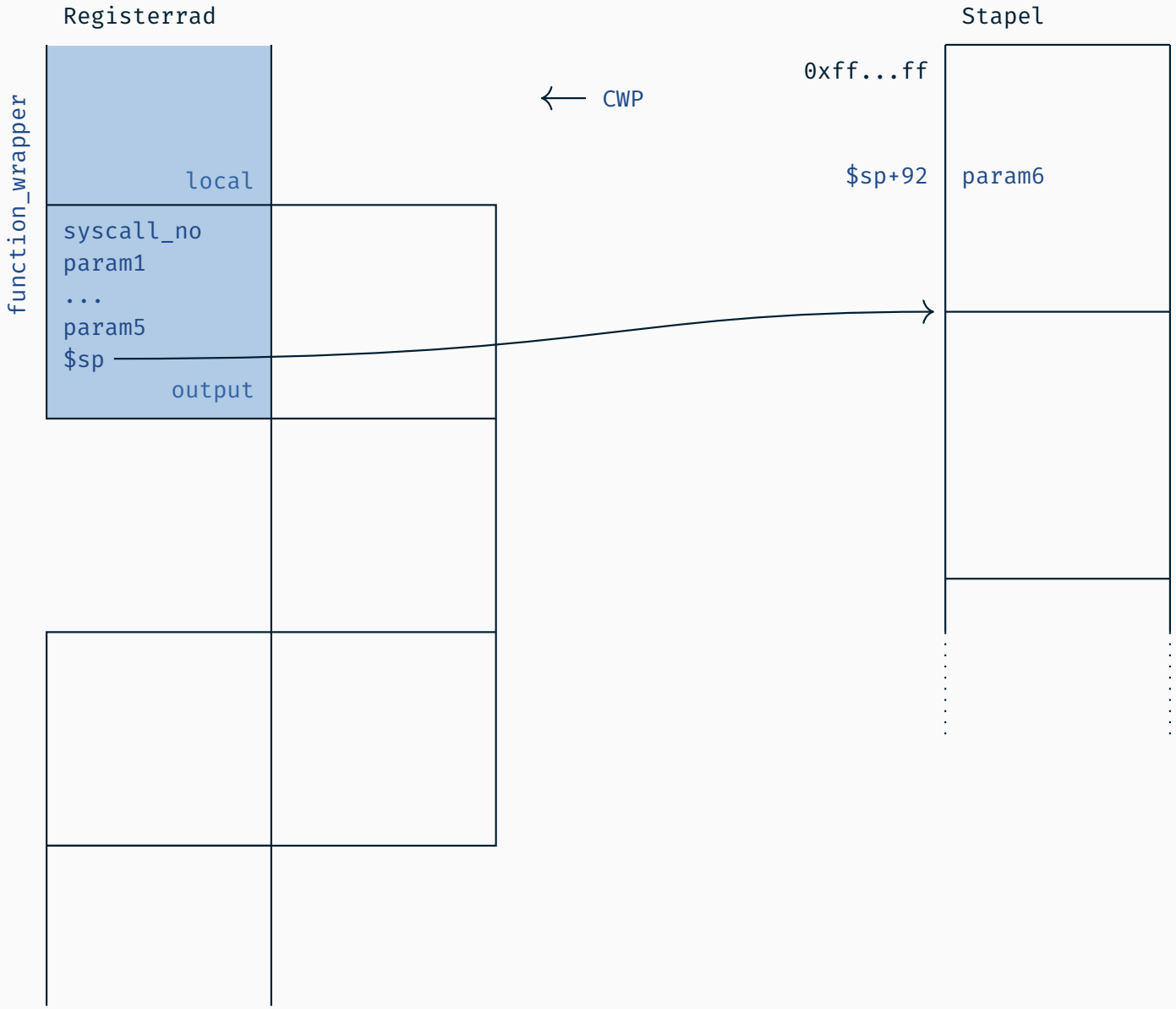
# Window Underflow



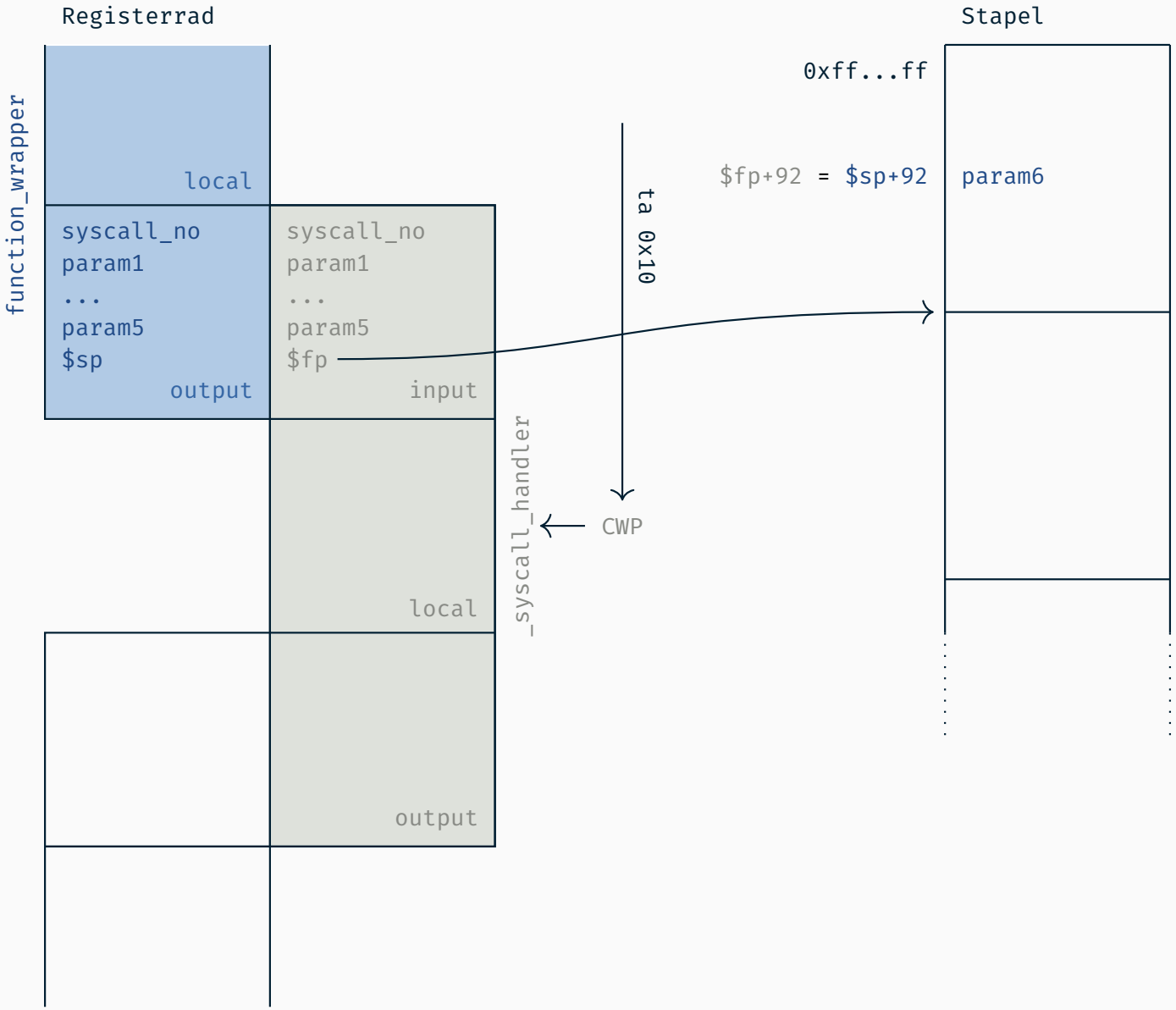
# Window Underflow



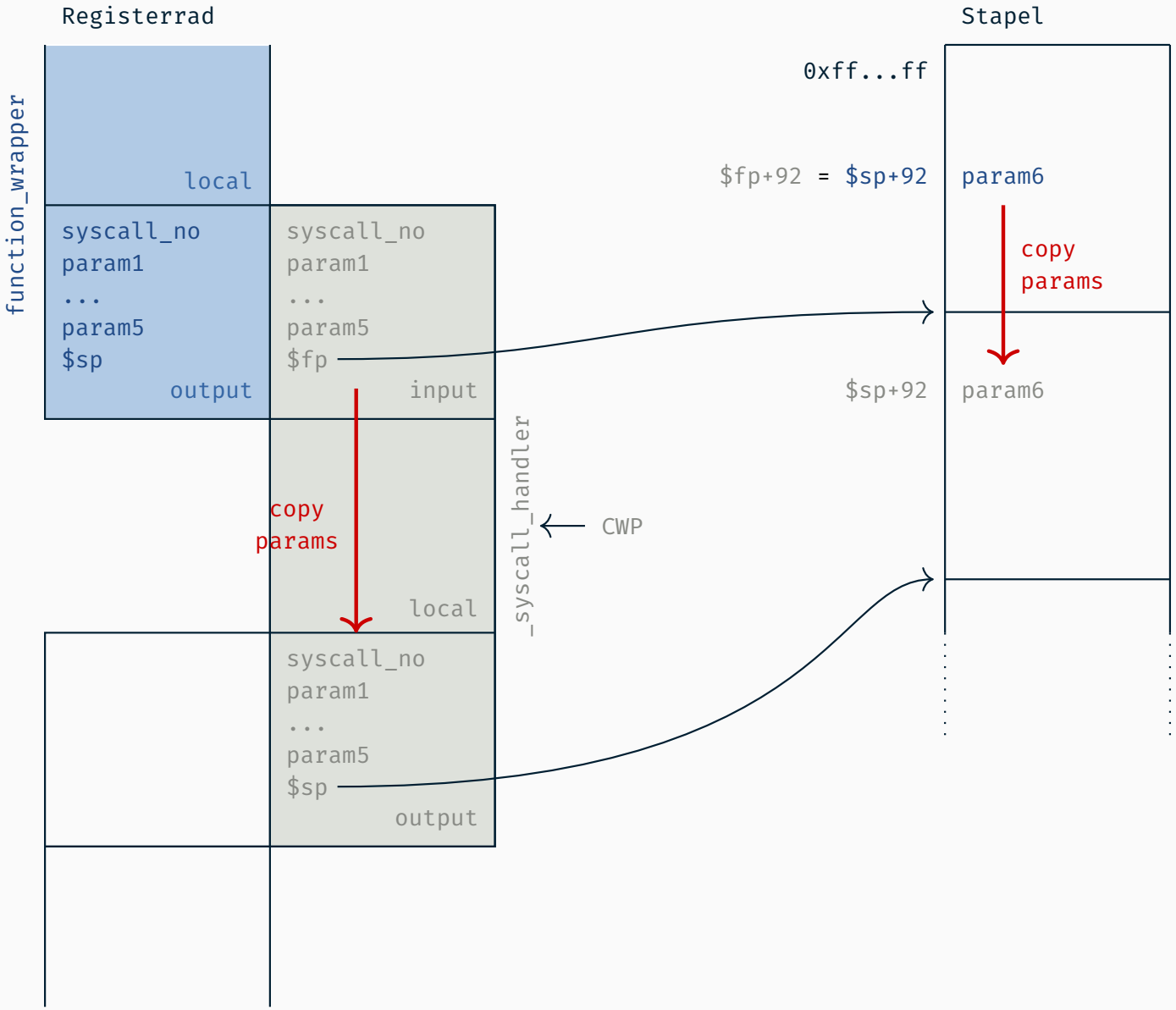
# SPARC und der Systemaufruf



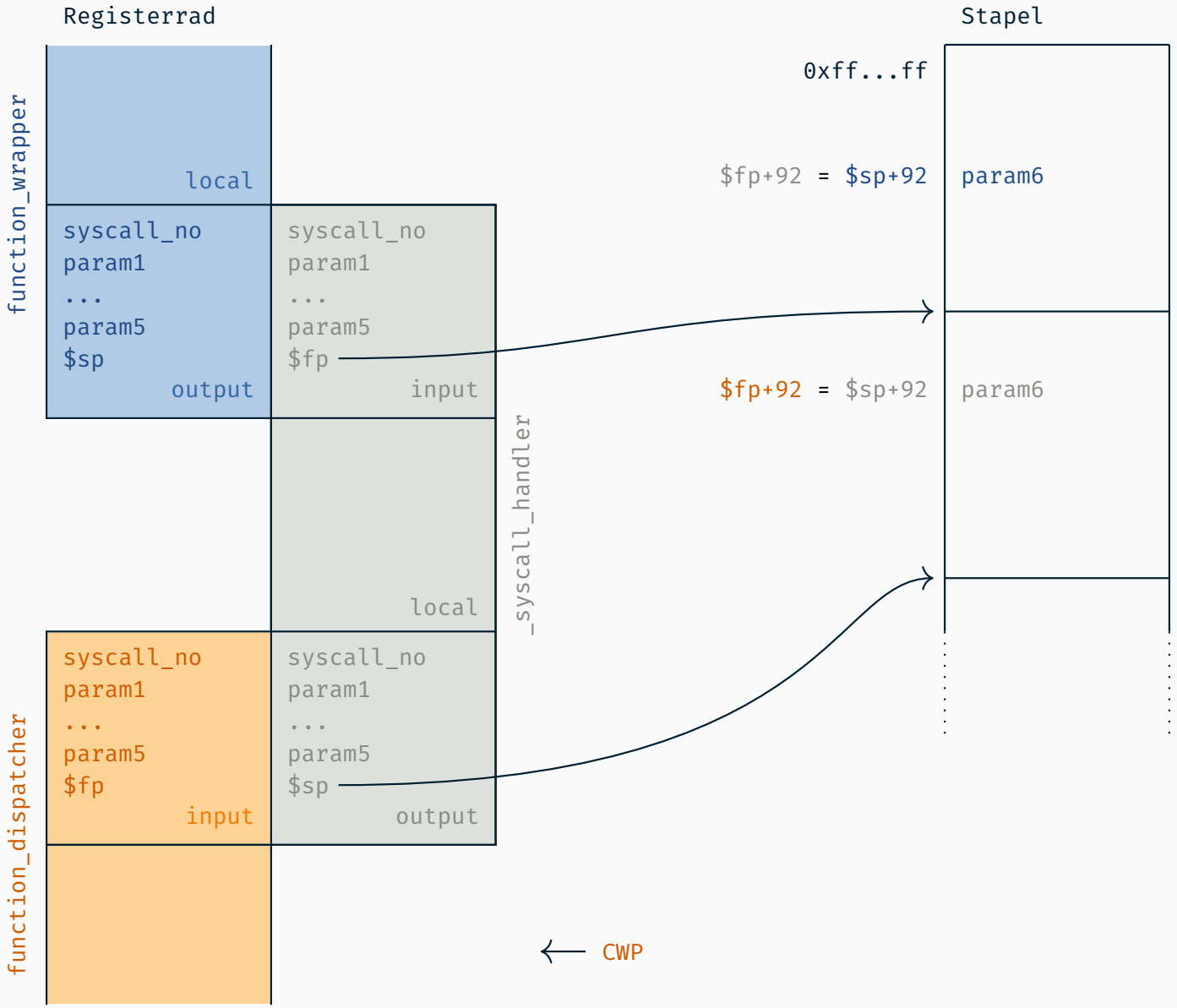
# SPARC und der Systemaufruf



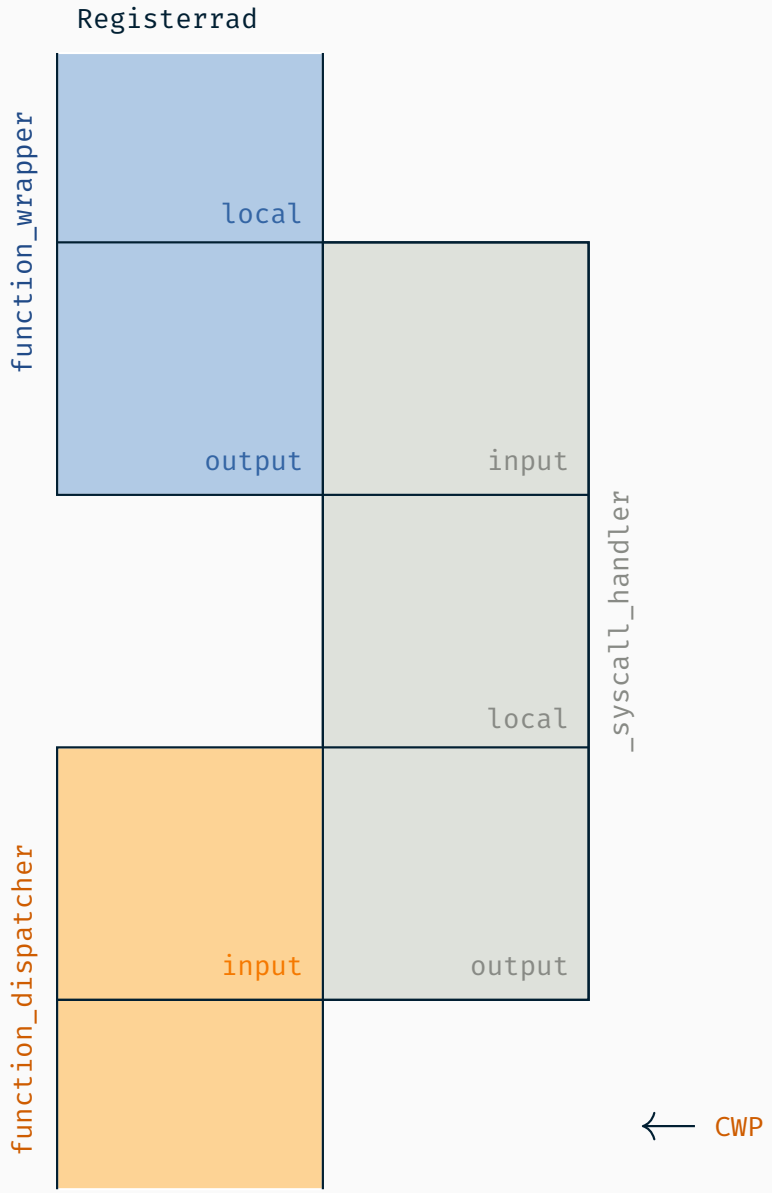
# SPARC und der Systemaufruf



# SPARC und der Systemaufruf

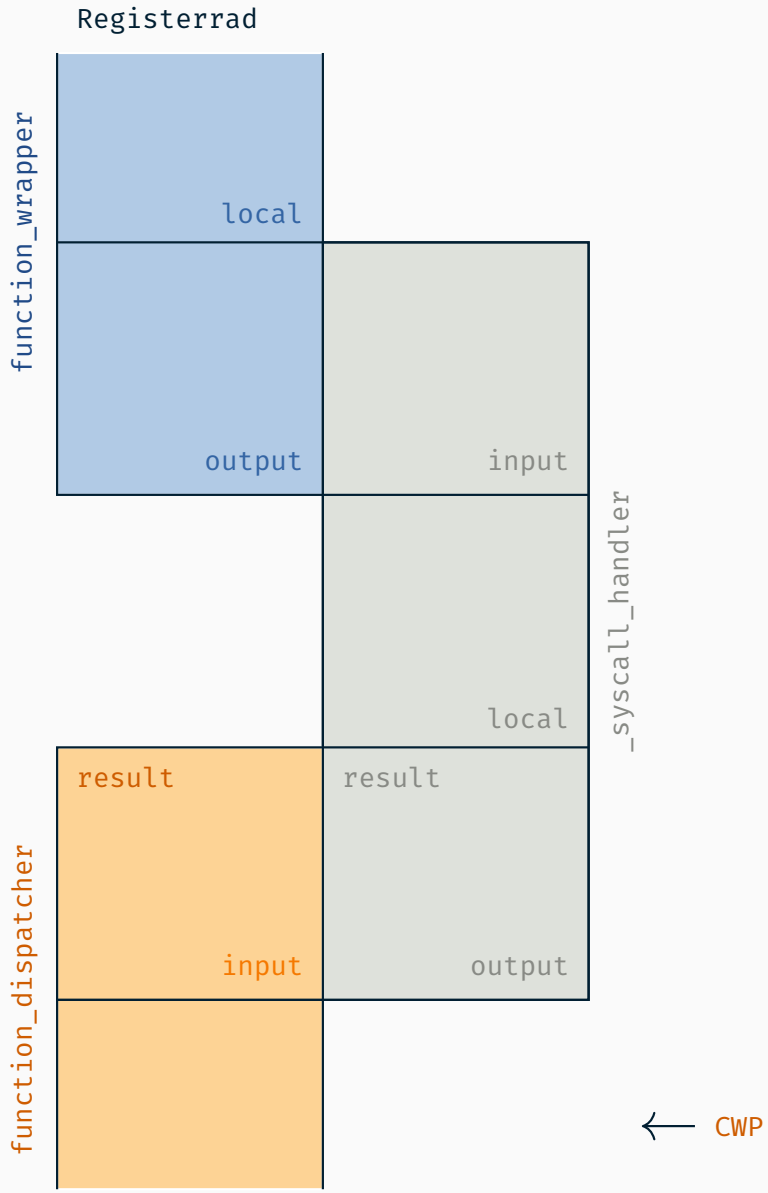


# SPARC und die Ergebnisrückgabe



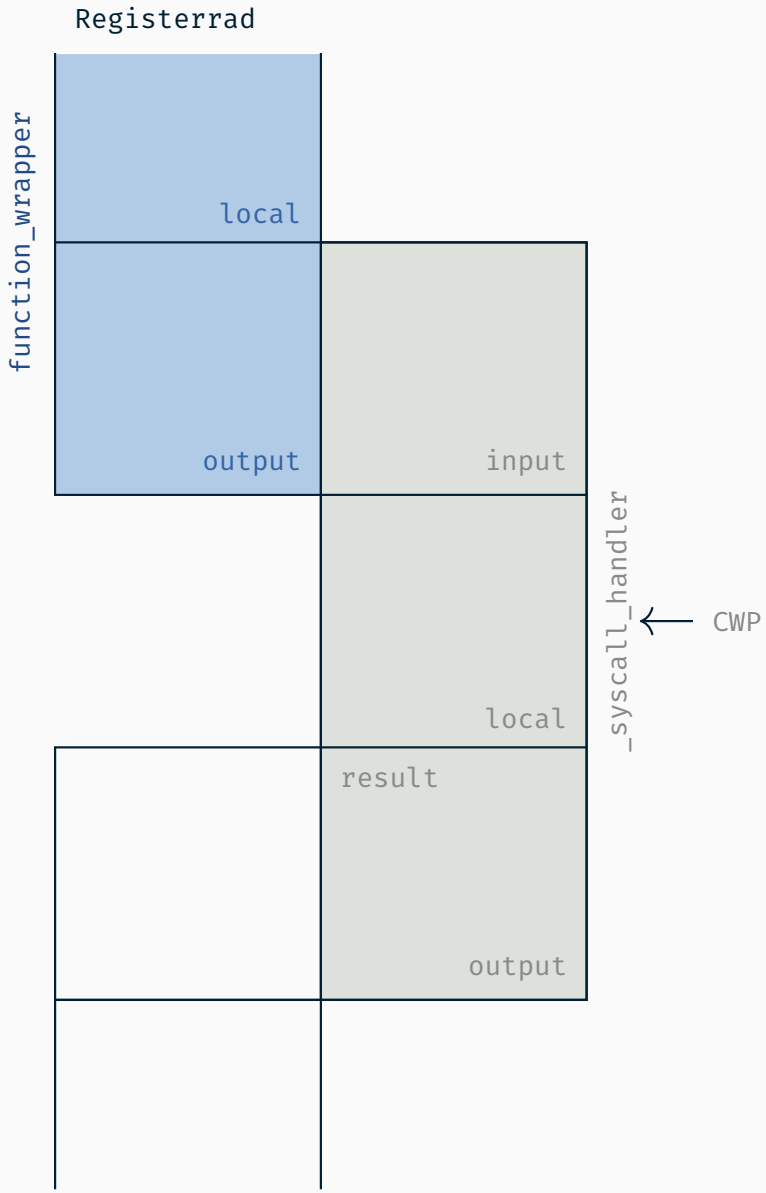

- Suchen der Funktion aus der Systemaufruftabelle mithilfe der übergebenen Nummer
- Ausführen der Funktion
- Rückgabe des Ergebnisses

# SPARC und die Ergebnisrückgabe



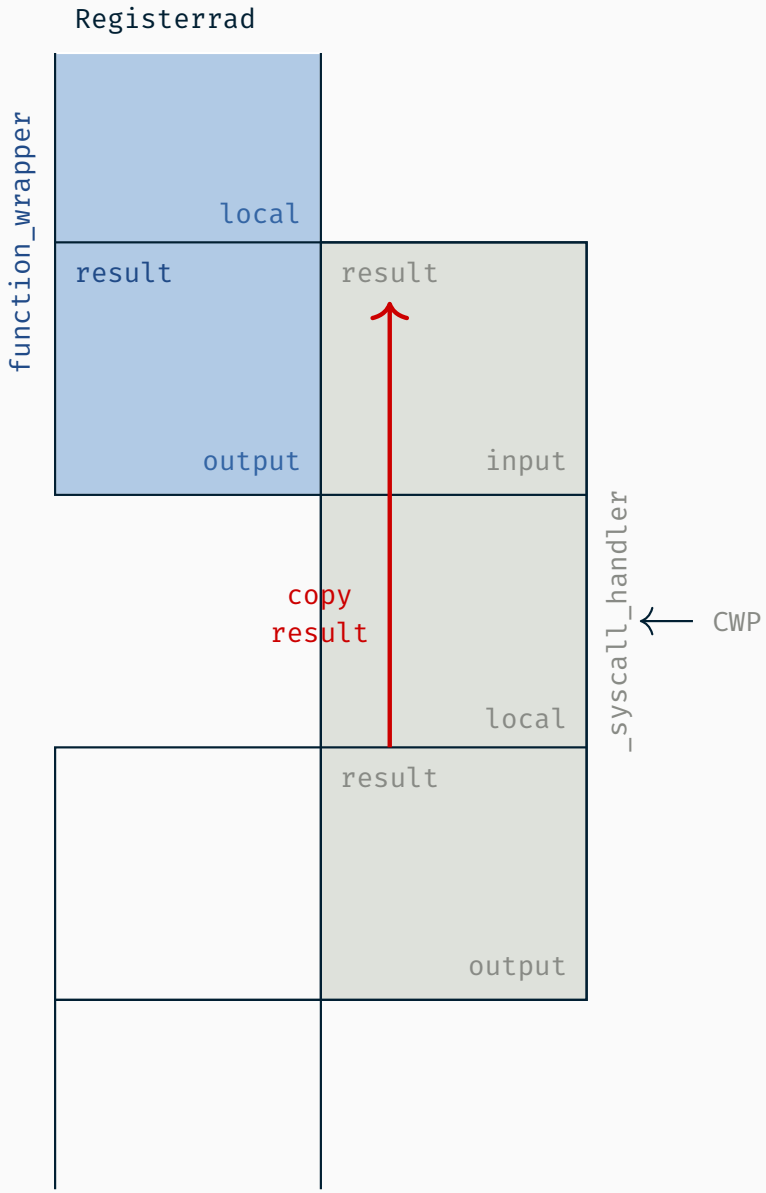

- Rückgabe des Ergebnisses

# SPARC und die Ergebnisrückgabe



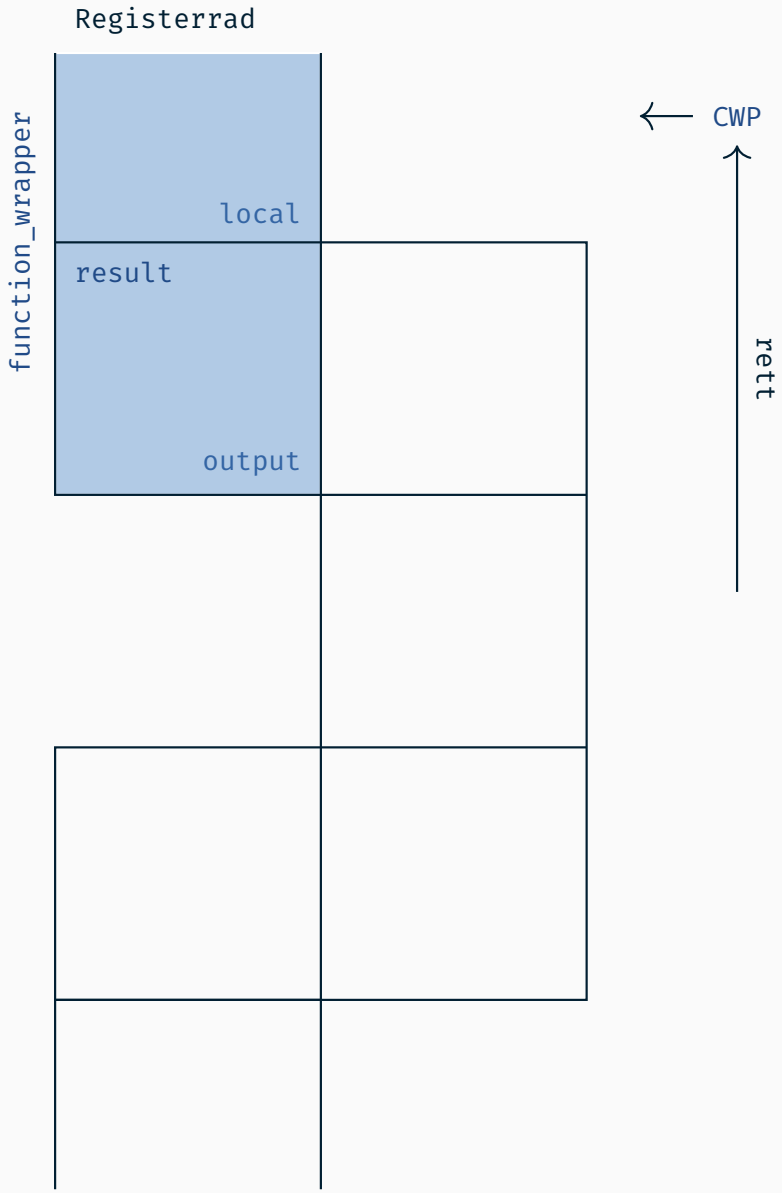

- Rückgabe des Ergebnisses

# SPARC und die Ergebnisrückgabe




- Rückgabe des Ergebnisses

# SPARC und die Ergebnisrückgabe




- Rückgabe des Ergebnisses

# Evaluation

# Evaluation

Evaluationsplatformen:

- `x86guest`: Ubuntu 18.04, Linux Kernel 4.15
- `x64native`: 4 Intel® Xeon® E7-4830 v3 Haswell Prozessoren mit insgesamt 48 phys. / 96 log. Kernen
- `leon`: proFPGA mit 16 Prozessoren, auf 50 MHz getaktet

# Evaluation

Evaluationsplatformen:

- `x86guest`: Ubuntu 18.04, Linux Kernel 4.15
- `x64native`: 4 Intel® Xeon® E7-4830 v3 Haswell Prozessoren mit insgesamt 48 phys. / 96 log. Kernen
- `leon`: proFPGA mit 16 Prozessoren, auf 50 MHz getaktet

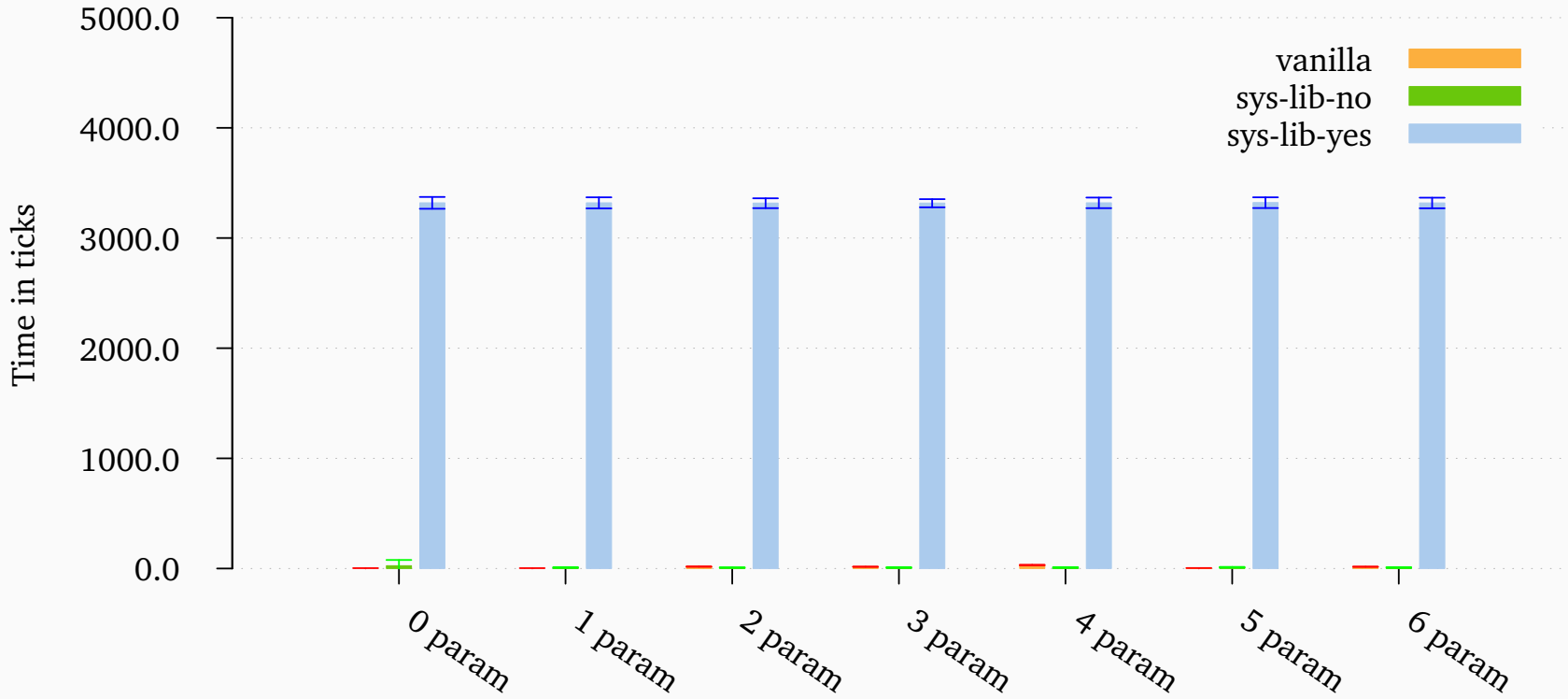
Einstellungen:

- `vanilla`: Basisversion von OctoPOS vor der Systemaufrufimplementierung
- `sys-lib-no`: Verwendung der Systemaufruftabelle
- `sys-lib-yes`: mit Systemaufruf

# Microbenchmarks

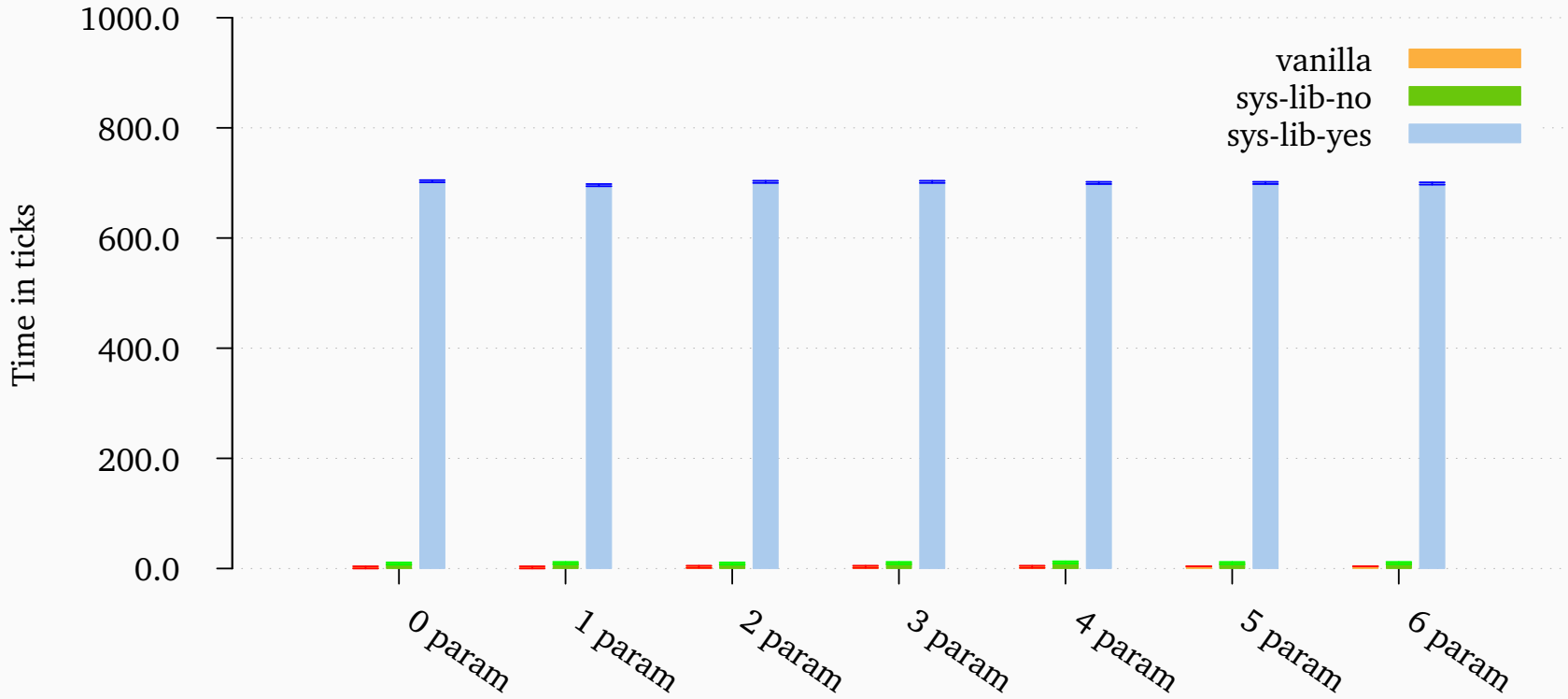
- leere Systemaufrufe und einfache Funktionen
- 100 000 Iterationen,
  davon die letzten 1000 zur Auswertung

# Leere Systemaufrufe unter `x86guest`



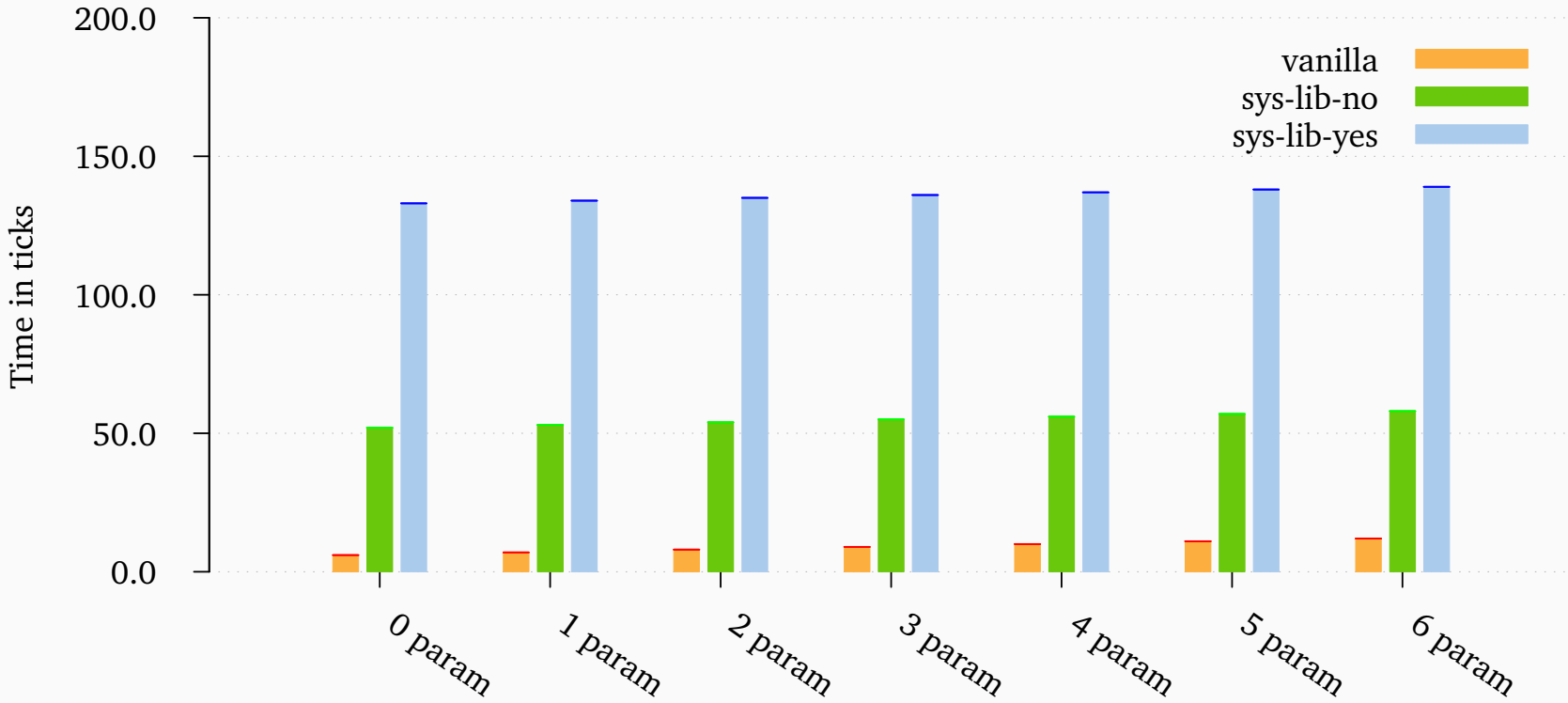

Systemaufruf: 3300 ticks = 3300 ns langsamer

# Leere Systemaufrufe unter `x64native`




Systemaufruf: 700 ticks = 330 ns langsamer

# Leere Systemaufrufe unter `leon`




Systemaufruf: 125 ticks = 2500 ns langsamer

# Makrobenchmarks

NAS Parallel Benchmarks:

- zum Einordnen der Leistung eines Supercomputers
- Anwendungen aus der Numerischen Strömungsmechanik (engl. computational fluid dynamics applications)
- pro Platform und Benchmark angepasste Benchmarkgrößen und Anzahl an MPI-Prozessen

# Anzahl an Systemaufrufen für NAS Parallel Benchmarks

| Architektur | BT | CG | EP | FT |
|---|---|---|---|---|
| x86guest | 2 683 091 | 1 486 877 | 9502 | 78 571 |
| x64native | 21 737 760 | 33 584 381 | 47 951 | 1 856 170 |
| leon | 367 503 | 649 244 | 4308 | - |

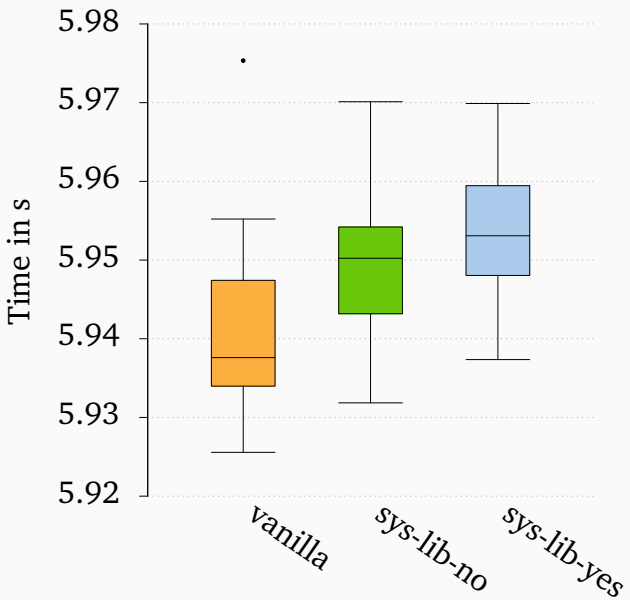
| Architektur | IS | LU | MG | SP |
|---|---|---|---|---|
| x86guest | 197 506 | 24 451 474 | 478 132 | 5 330 606 |
| x64native | 1 838 573 | - | 1 114 253 | 43 275 185 |
| leon | 85 977 | 324 066 | 161 197 | 600 838 |

# Anzahl an Systemaufrufen für NAS Parallel Benchmarks

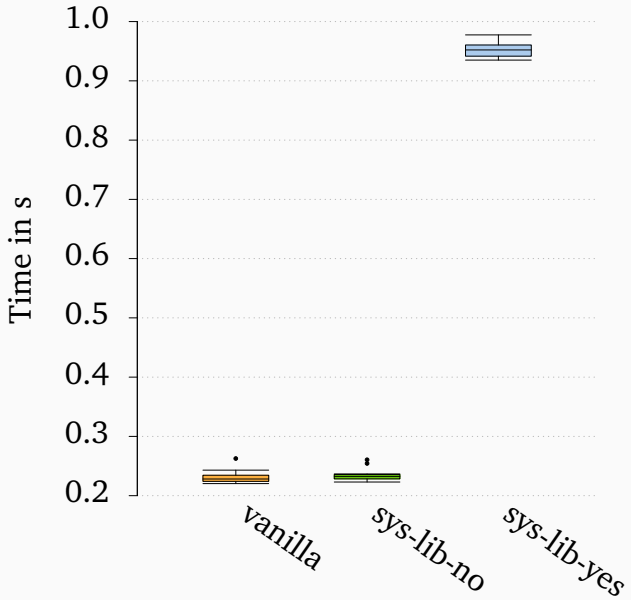
| Architektur | BT | CG | EP | FT |
|---|---|---|---|---|
| x86guest | 2 683 091 | 1 486 877 | 9502 | 78 571 |
| x64native | 21 737 760 | 33 584 381 | 47 951 | 1 856 170 |
| leon | 367 503 | 649 244 | 4308 | - |

| Architektur | IS | LU | MG | SP |
|---|---|---|---|---|
| x86guest | 197 506 | 24 451 474 | 478 132 | 5 330 606 |
| x64native | 1 838 573 | - | 1 114 253 | 43 275 185 |
| leon | 85 977 | 324 066 | 161 197 | 600 838 |

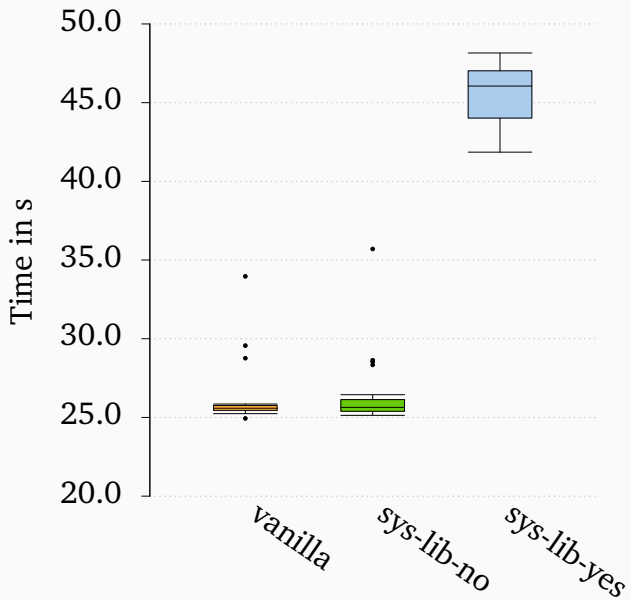
# NAS Parallel Benchmarks für x86guest




**(a)** EP Benchmark

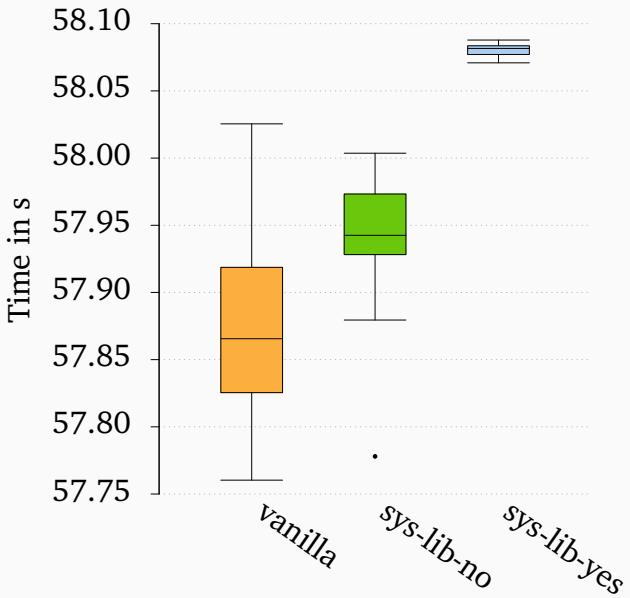



**(b)** MG Benchmark

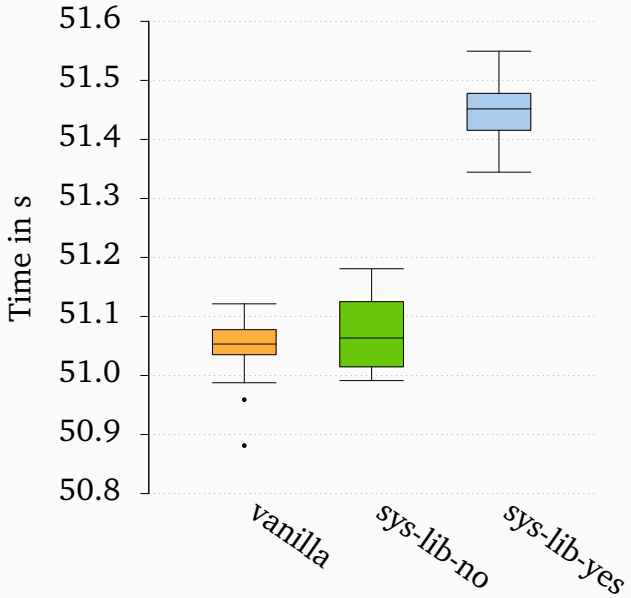



**(c)** SP Benchmark

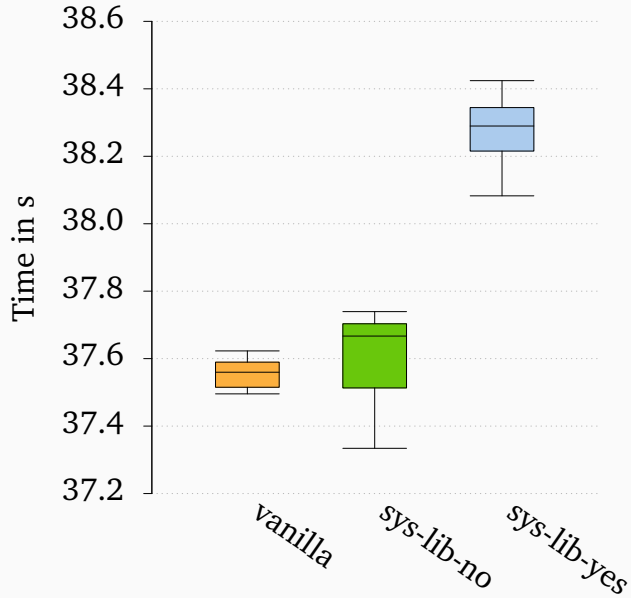
# NAS Parallel Benchmarks für `x64native`




**(a)** EP Benchmark

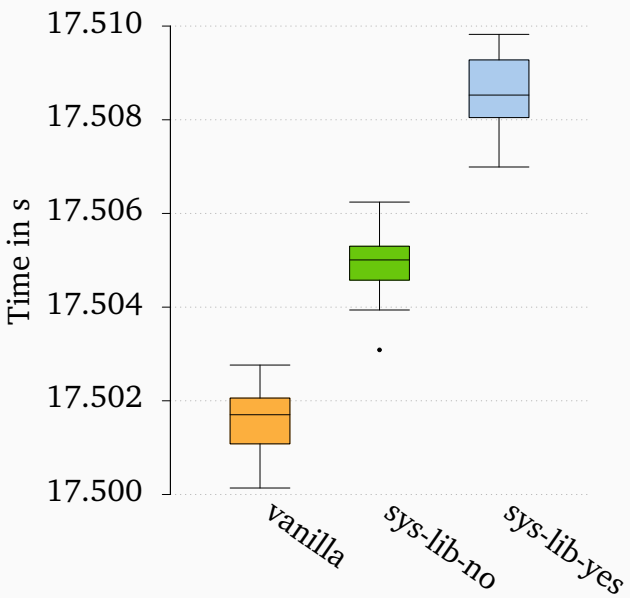



**(b)** MG Benchmark

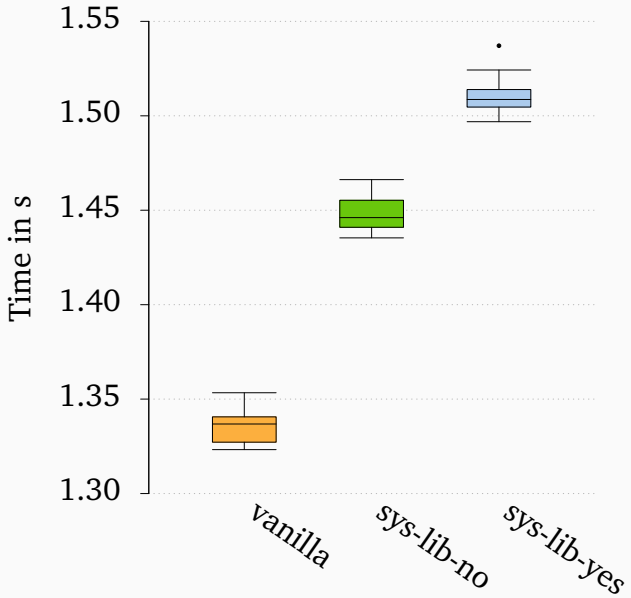



**(c)** SP Benchmark

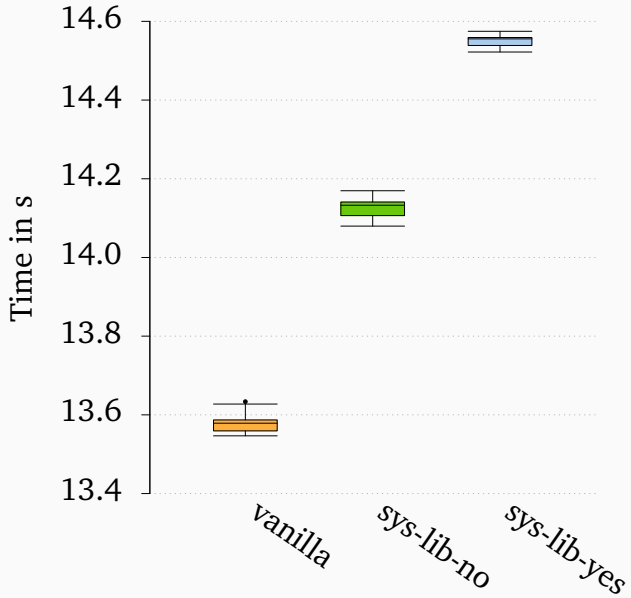
# NAS Parallel Benchmarks für `leon`




**(a)** EP Benchmark




**(b)** MG Benchmark




**(c)** SP Benchmark

# Zusammenfassung

# Zusammenfassung

**Ziel war:**
eine Schnittstelle zwischen *user* und *kernel space*, ...

- ... um Betriebssystemfunktionen anzufordern
- ... die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- ... die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- ... die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Zusammenfassung

**Ziel war:**
eine Schnittstelle zwischen *user* und *kernel space*, …

- ✓ … um Betriebssystemfunktionen anzufordern
- … die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- … die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- … die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Zusammenfassung

**Ziel war:**
eine Schnittstelle zwischen *user* und *kernel space*, …

- ✓ … um Betriebssystemfunktionen anzufordern
- ✓ … die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- … die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- … die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Zusammenfassung

**Ziel war:**
eine Schnittstelle zwischen *user* und *kernel space*, …

- ✓ … um Betriebssystemfunktionen anzufordern
- ✓ … die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- ✓ … die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- ■ … die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Zusammenfassung

**Ziel war:**
eine Schnittstelle zwischen *user* und *kernel space*, …

- ✓ … um Betriebssystemfunktionen anzufordern
- ✓ … die eine **lose Koppelung** zwischen Anwendung und Betriebssystem bereitstellt
- ✓ … die **einheitlich** und **architekturunabhängig** vom Anwendungsprogrammierer nutzbar ist
- ✓ … die **erweiterbar** und **dynamisch anpassbar** ist

für **alle drei** unterstützten Architekturen von OctoPOS.

# Ausblick

- Privilegientrennung durch Hardware-Mechanismen (Privilegienlevel mit RPL Bits für x86_64, S Bit für SPARC)

# Ausblick

- Privilegientrennung durch Hardware-Mechanismen (Privilegienlevel mit RPL Bits für x86_64, S Bit für SPARC)
- Schnelle Systemaufrufe unter `x64native` mit syscall / sysret

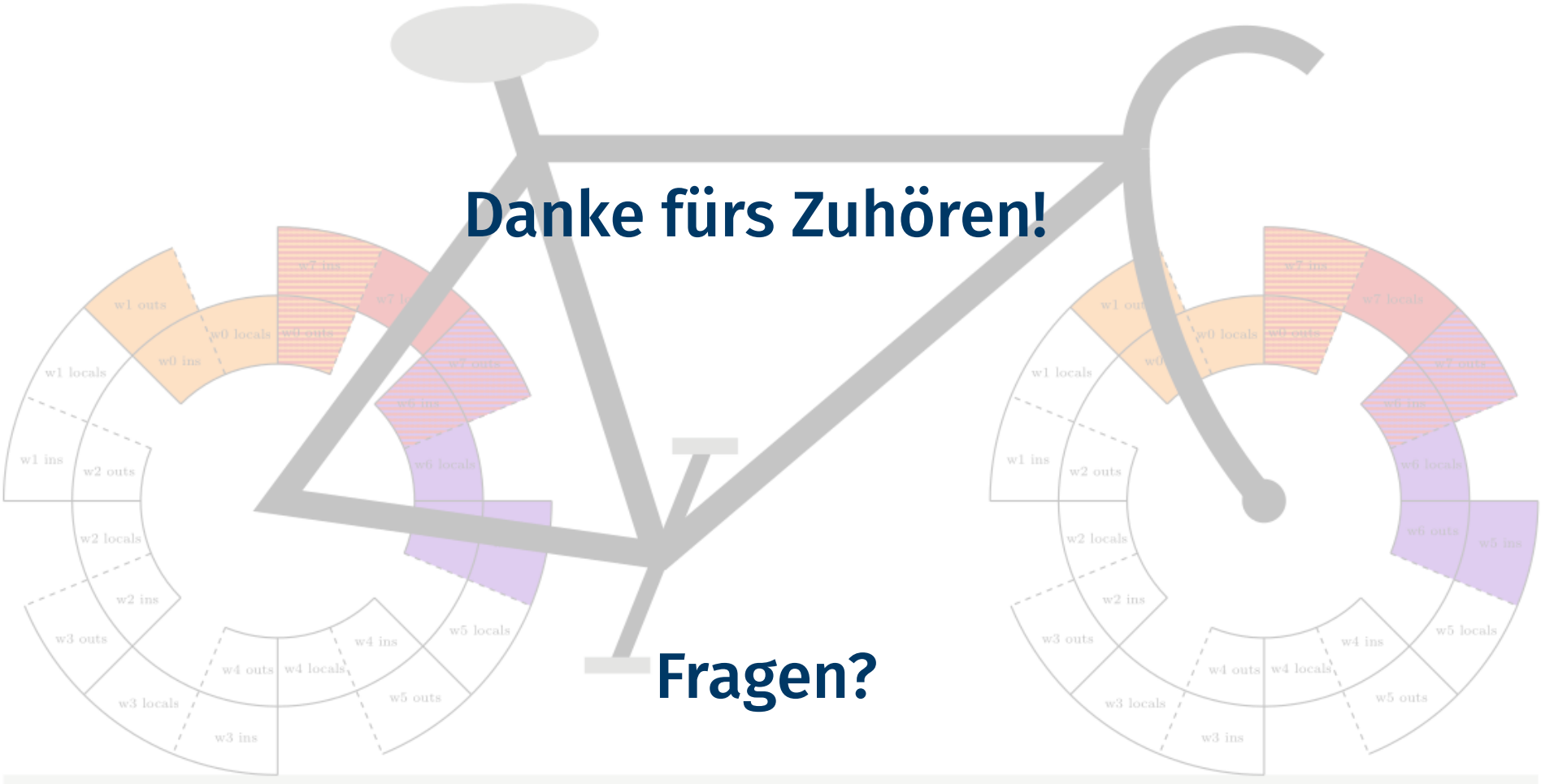
# Ausblick

- Privilegientrennung durch Hardware-Mechanismen (Privilegienlevel mit RPL Bits für x86_64, S Bit für SPARC)
- Schnelle Systemaufrufe unter `x64native` mit syscall / sysret
- Teile der C-Schnittstelle können auch im *user space* implementiert werden

# Ausblick

- Privilegientrennung durch Hardware-Mechanismen (Privilegienlevel mit RPL Bits für x86_64, S Bit für SPARC)
- Schnelle Systemaufrufe unter `x64native` mit syscall / sysret
- Teile der C-Schnittstelle können auch im *user space* implementiert werden
- asynchrone Implementierung des Systemaufrufmechanismus

# Danke fürs Zuhören!




# Fragen?

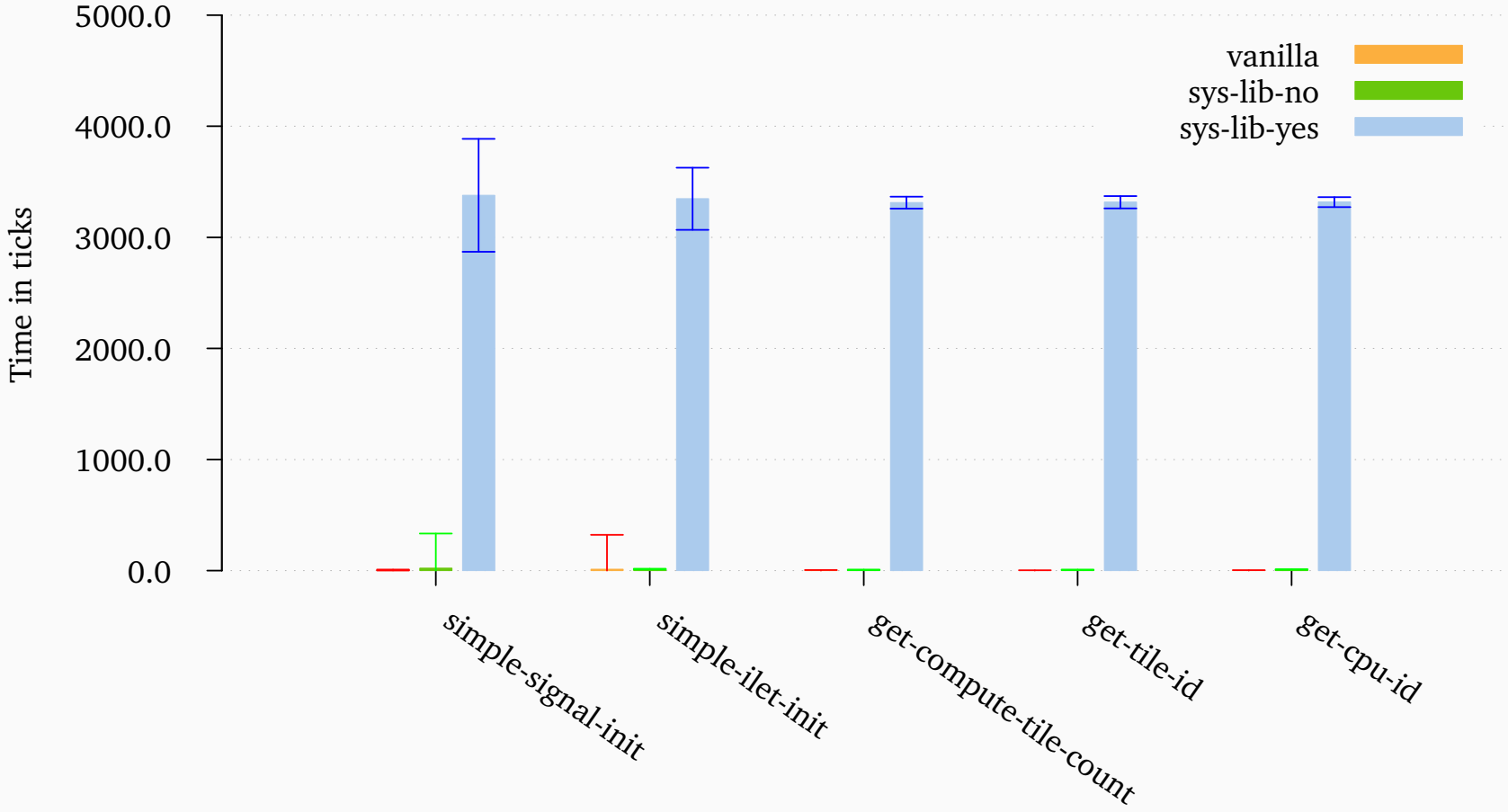
Danke fürs Zuhören!

Fragen?

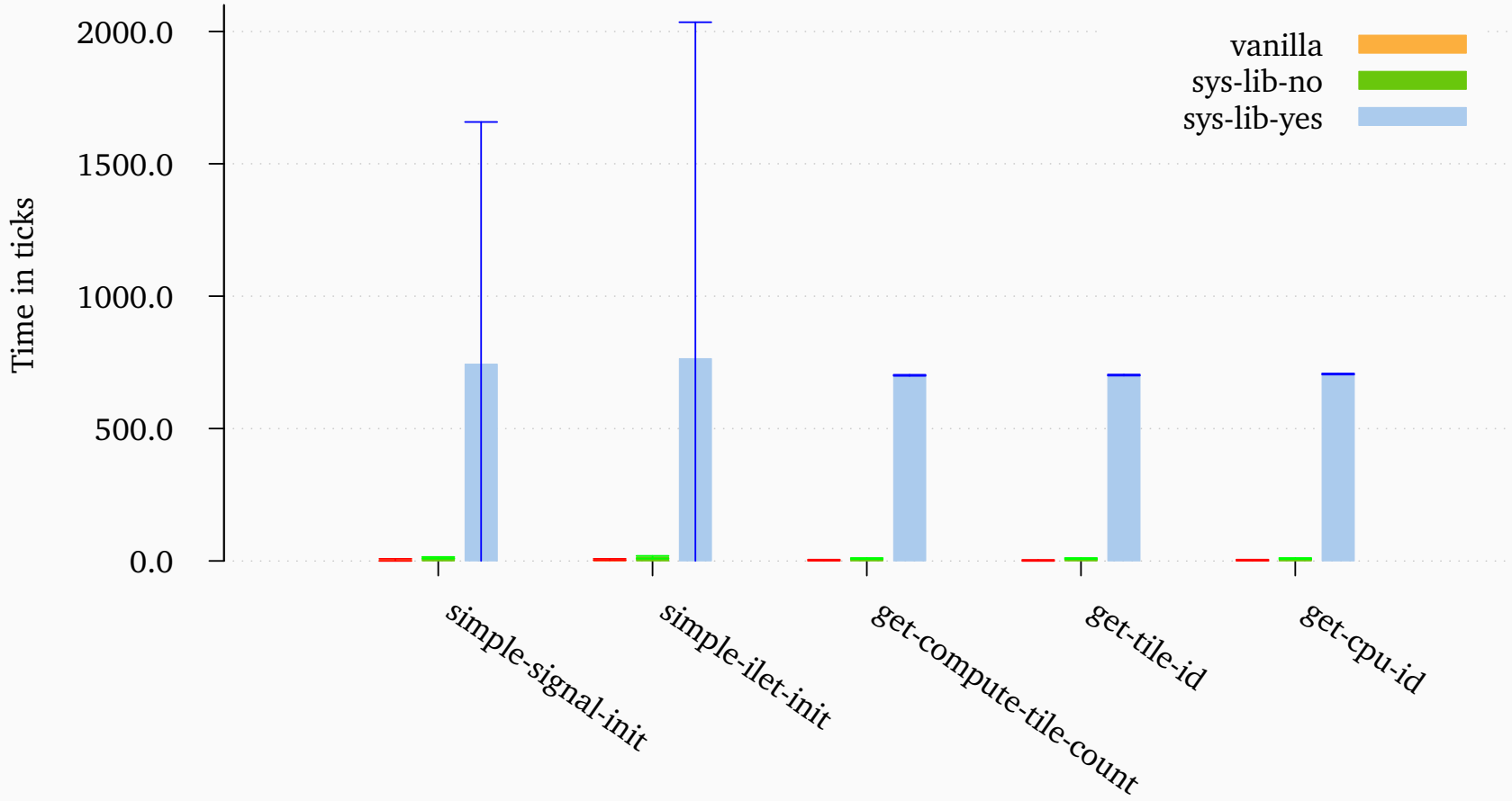
# leere Systemaufrufe: genaue Zeiten

| | version | 0 param | 1 param | 2 param | 3 param | 4 param | 5 param | 6 param |
|---|---|---|---|---|---|---|---|---|
| x86guest | vanilla | 0 ± 3 | 0 ± 3 | 17 ± 2 | 16 ± 1 | 31 ± 3 | 1 ± 3 | 16 ± 1 |
| | sys-lib-no | 24 ± 53 | 9 ± 3 | 8 ± 3 | 9 ± 3 | 9 ± 3 | 11 ± 3 | 9 ± 3 |
| | sys-lib-yes | 3319 ± 54 | 3319 ± 50 | 3316 ± 45 | 3316 ± 37 | 3319 ± 48 | 3321 ± 49 | 3318 ± 48 |
| x64native | vanilla | 2 ± 2 | 2 ± 2 | 3 ± 2 | 3 ± 2 | 3 ± 2 | 4 ± 0 | 4 ± 0 |
| | sys-lib-no | 9 ± 2 | 10 ± 2 | 9 ± 2 | 10 ± 2 | 11 ± 2 | 10 ± 2 | 10 ± 2 |
| | sys-lib-yes | 703 ± 2 | 696 ± 2 | 702 ± 2 | 702 ± 2 | 700 ± 2 | 700 ± 2 | 699 ± 2 |
| leon | vanilla | 6 ± 0 | 7 ± 0 | 8 ± 0 | 9 ± 0 | 10 ± 0 | 11 ± 0 | 12 ± 0 |
| | sys-lib-no | 52 ± 0 | 53 ± 0 | 54 ± 0 | 55 ± 0 | 56 ± 0 | 57 ± 0 | 58 ± 0 |
| | sys-lib-yes | 133 ± 0 | 134 ± 0 | 135 ± 0 | 136 ± 0 | 137 ± 0 | 138 ± 0 | 139 ± 0 |

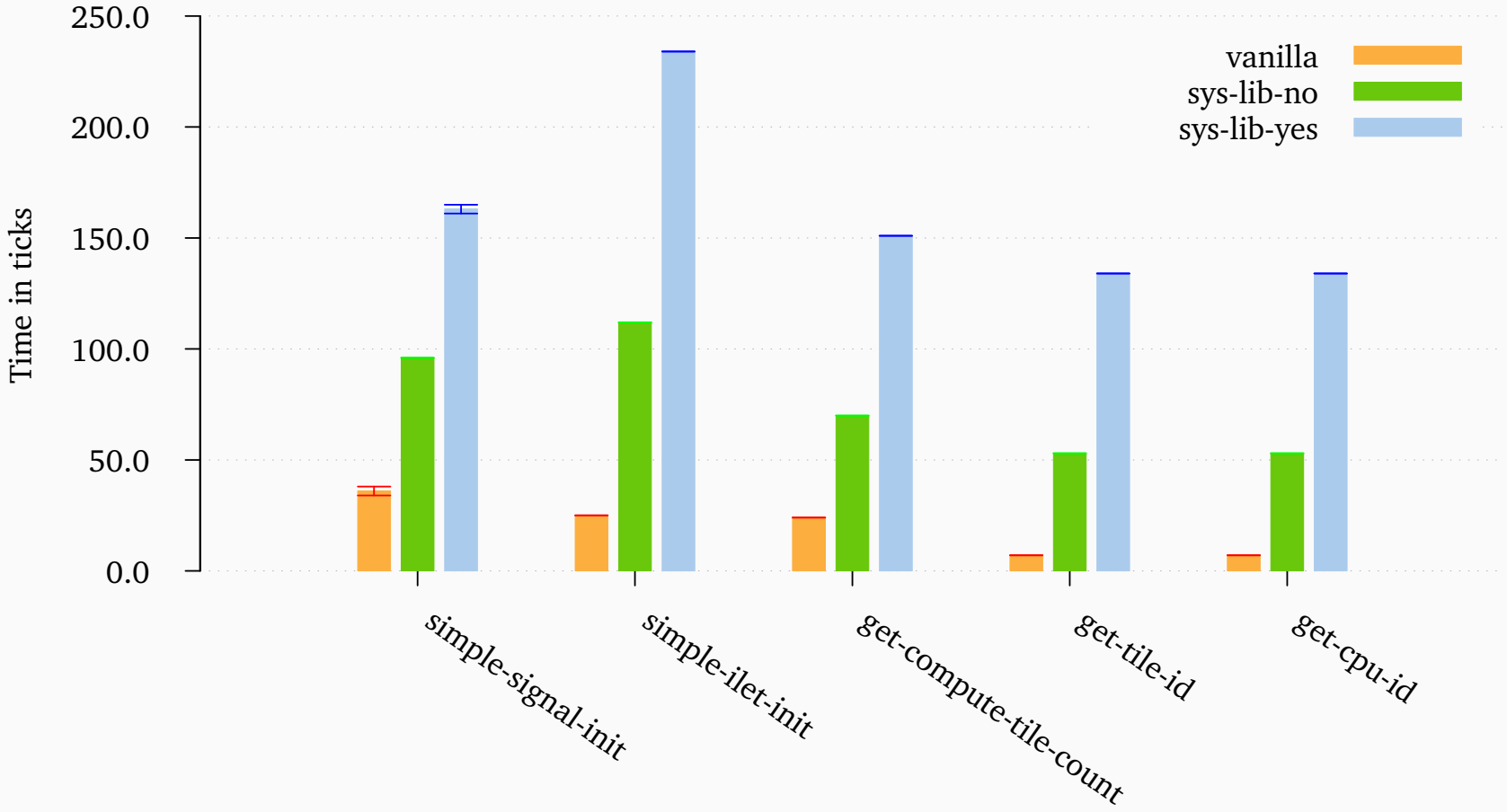
# Einfache Funktionen unter x86guest



# Einfache Funktionen unter `x64native`



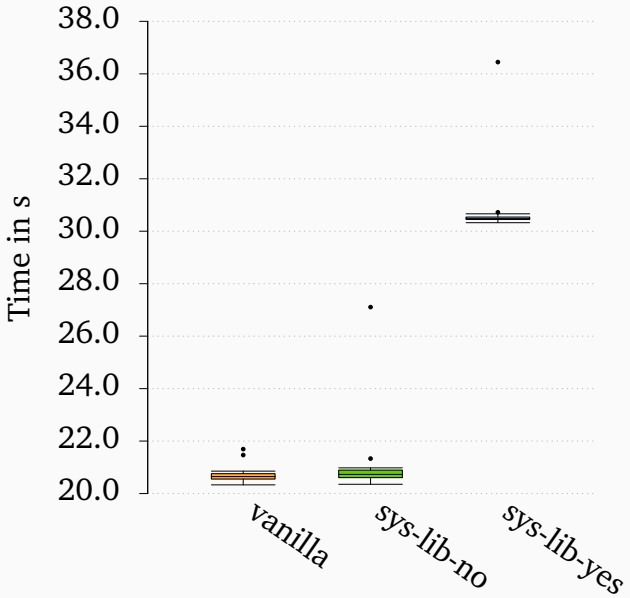
# Einfache Funktionen unter `leon`



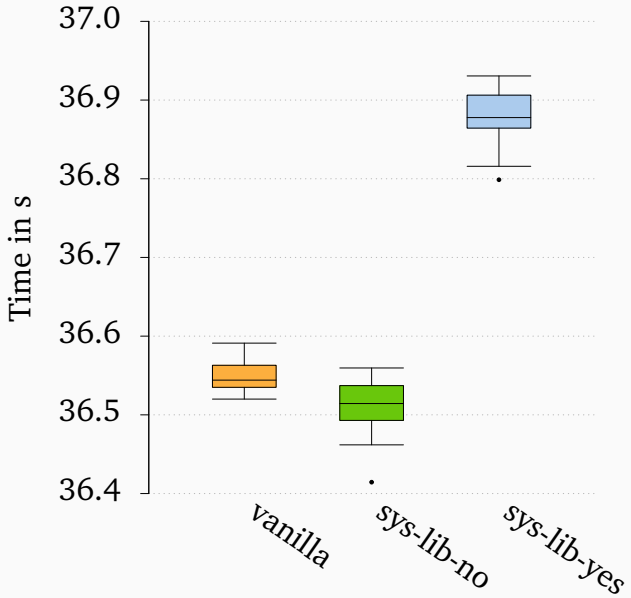
# Problemgrößen für NAS Parallel Benchmarks

| Architektur | | BT | CG | EP | FT | IS | LU | MG | SP |
|---|---|---|---|---|---|---|---|---|---|
| x86guest | class | A | A | A | W | A | A | W | A |
| | nprocs | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| x64native | class | C | D | D | C | D | - | D | C |
| | nprocs | 81 | 64 | 96 | 64 | 64 | - | 64 | 81 |
| leon | class | S | S | S | - | S | S | S | S |
| | nprocs | 16 | 16 | 16 | - | 16 | 16 | 16 | 16 |

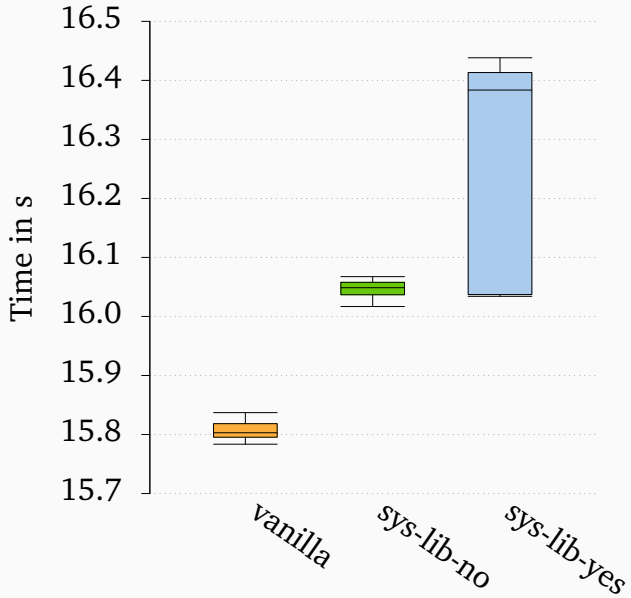
# BT Benchmark




(a) `x86guest`

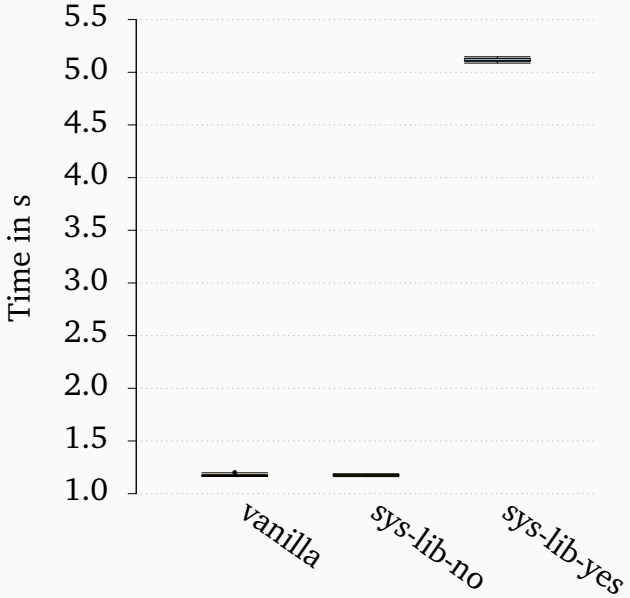



(b) `x64native`

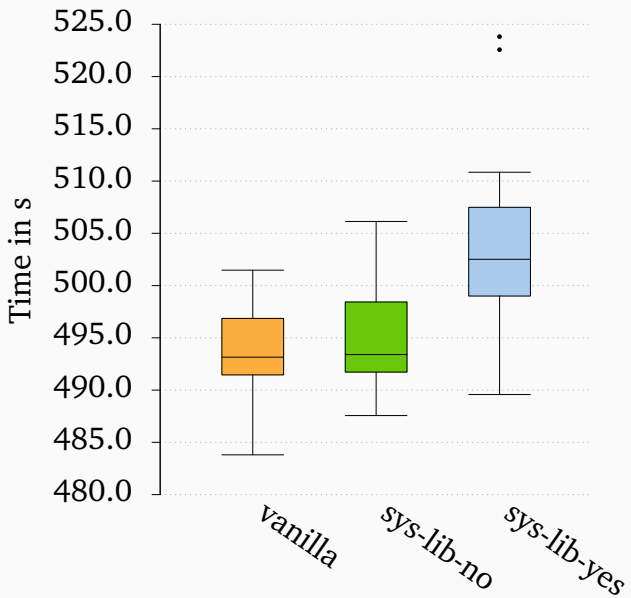



(c) `leon`

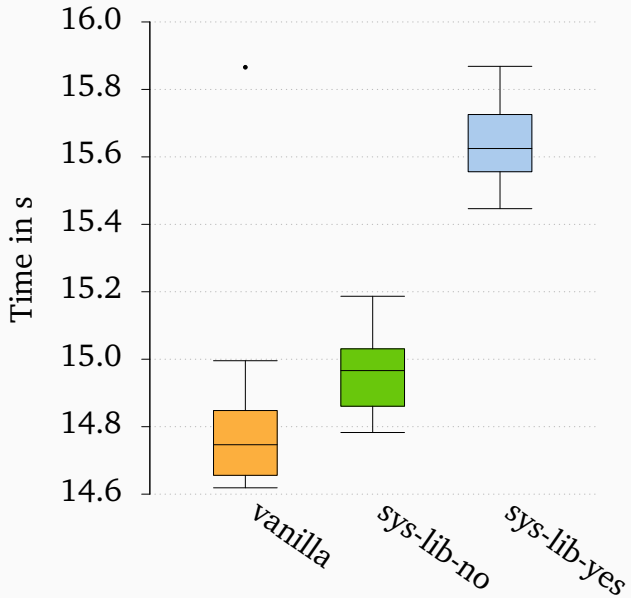
# CG Benchmark




(a) `x86guest`

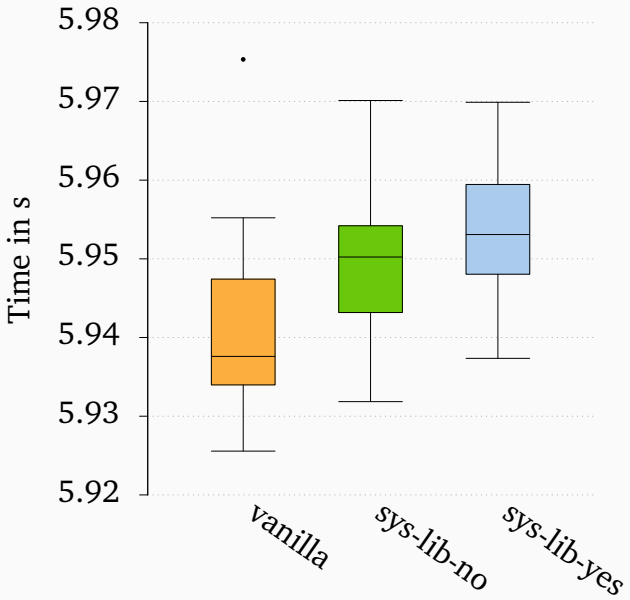



(b) `x64native`

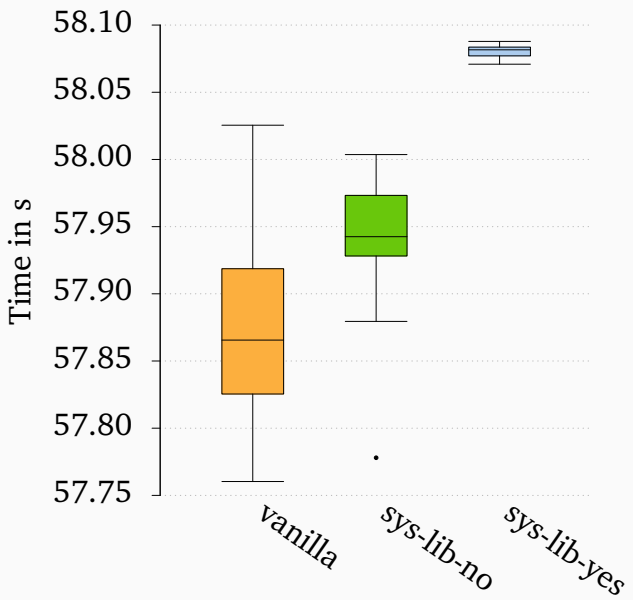



(c) `leon`

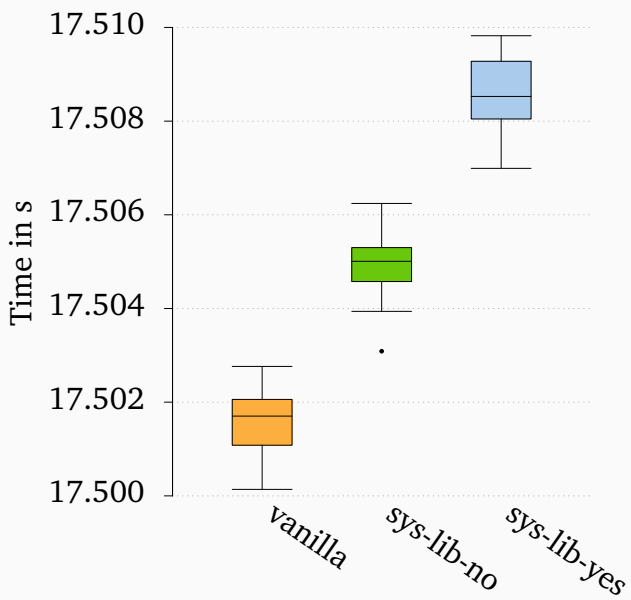
# EP Benchmark




(a) `x86guest`

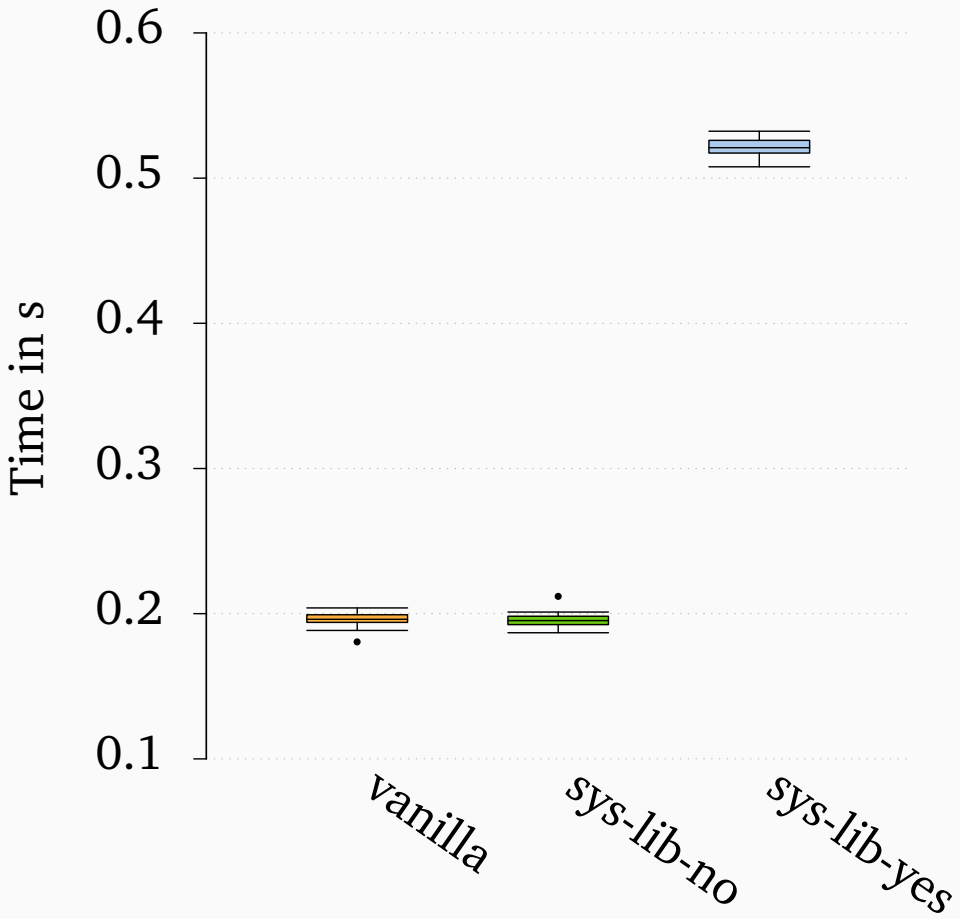



(b) `x64native`

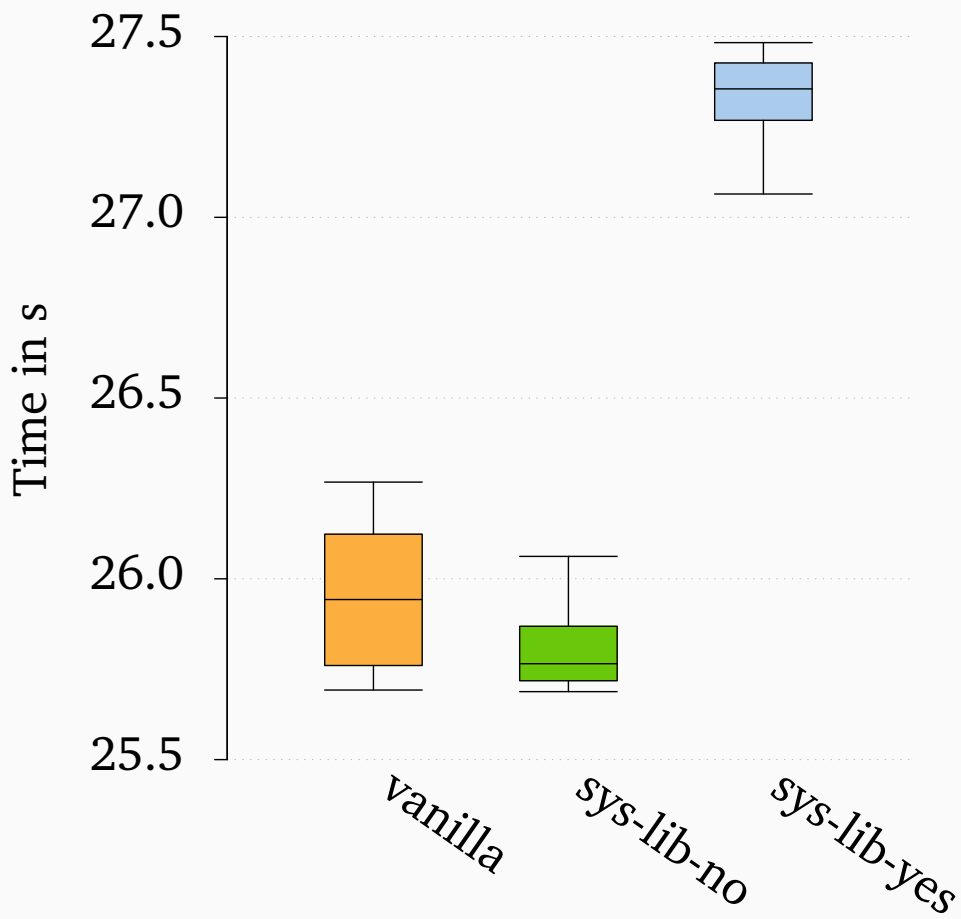



(c) `leon`

# FT Benchmark

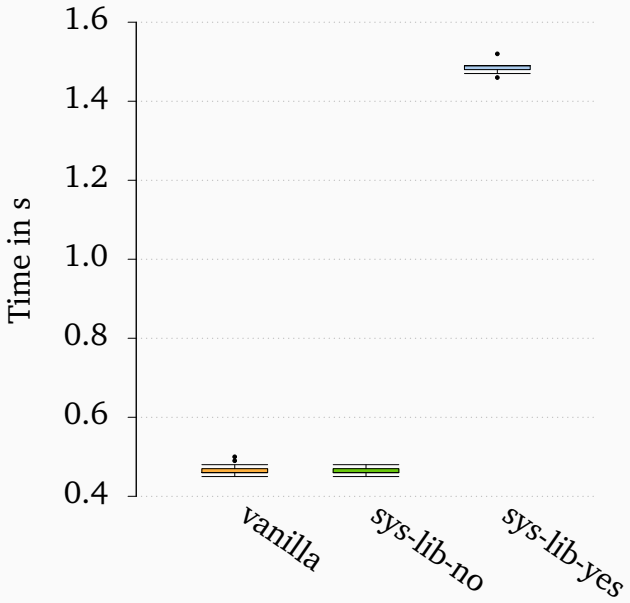



**(a)** `x86guest`

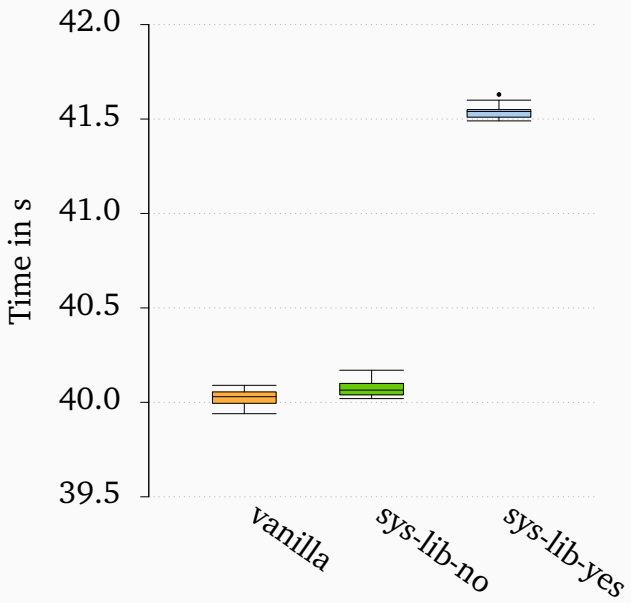



**(b)** `x64native`

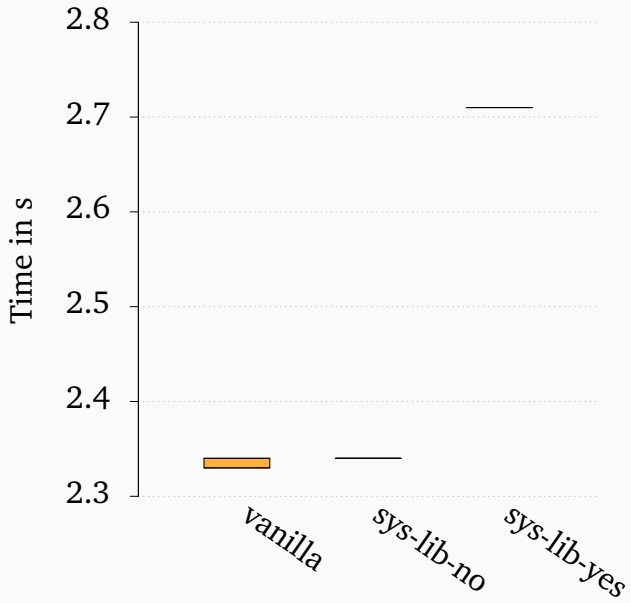
# IS Benchmark




(a) `x86guest`

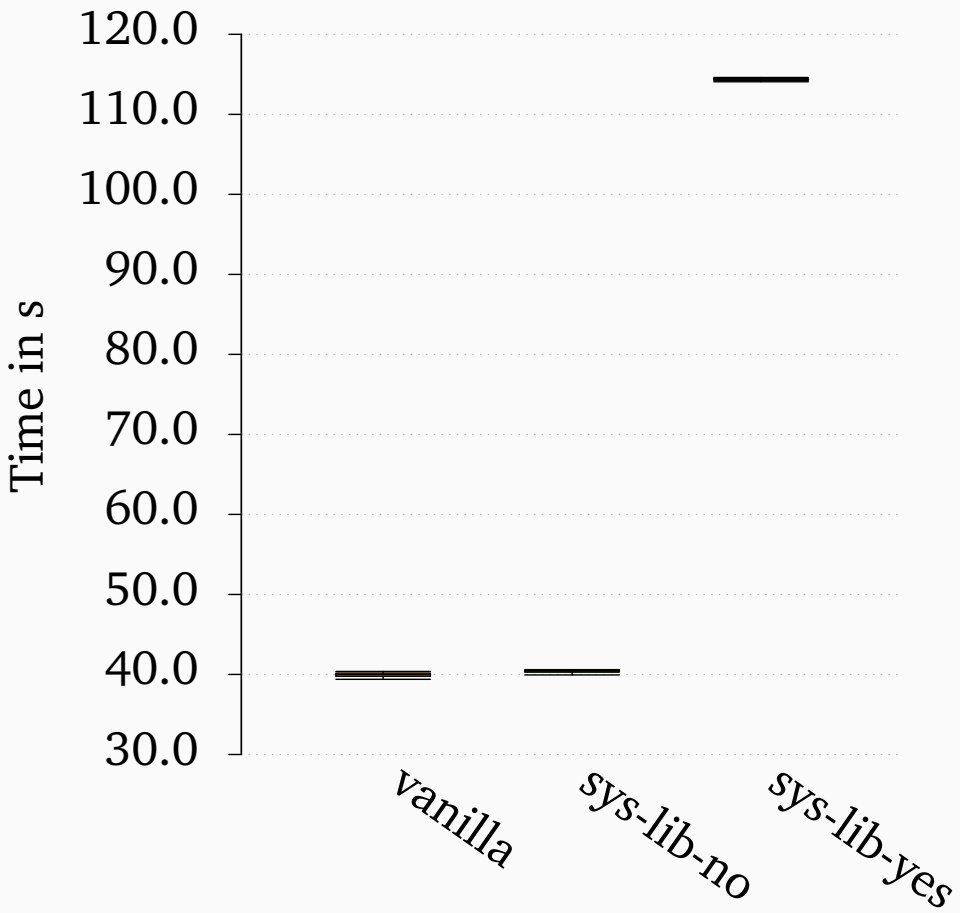



(b) `x64native`

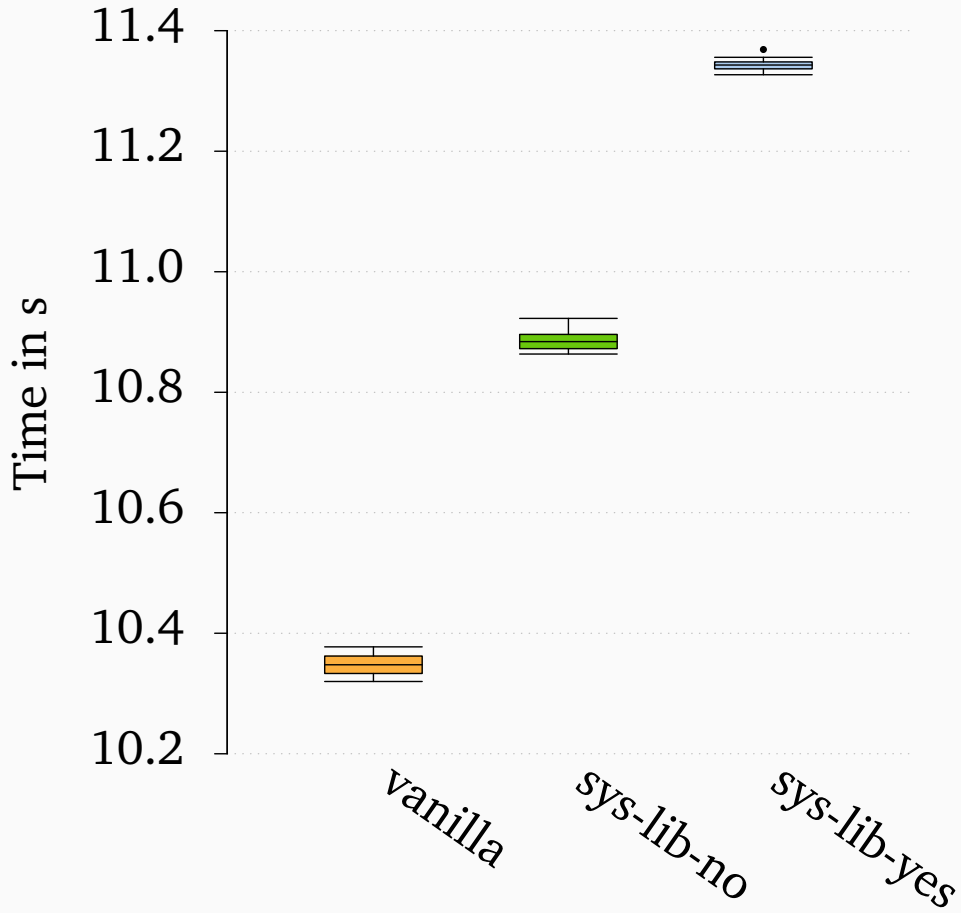



(c) `leon`

# LU Benchmark

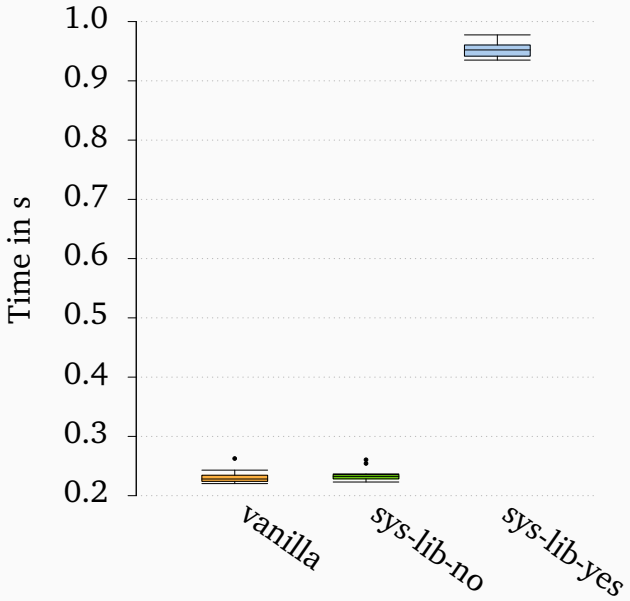



**(a)** `x86guest`

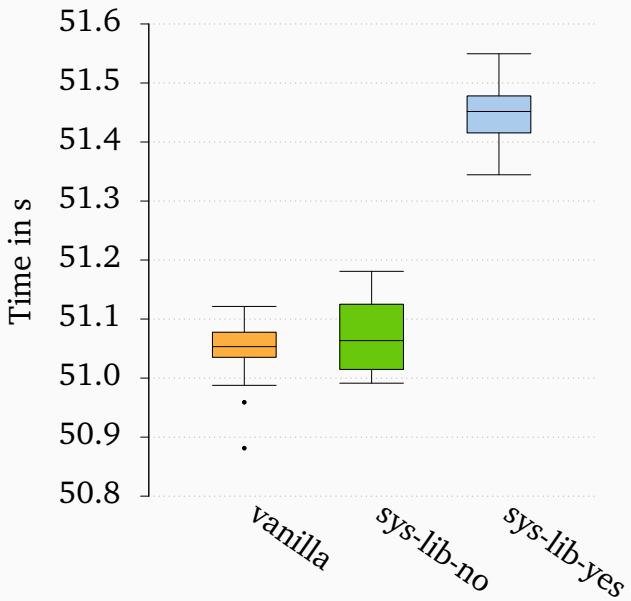



**(b)** `leon`

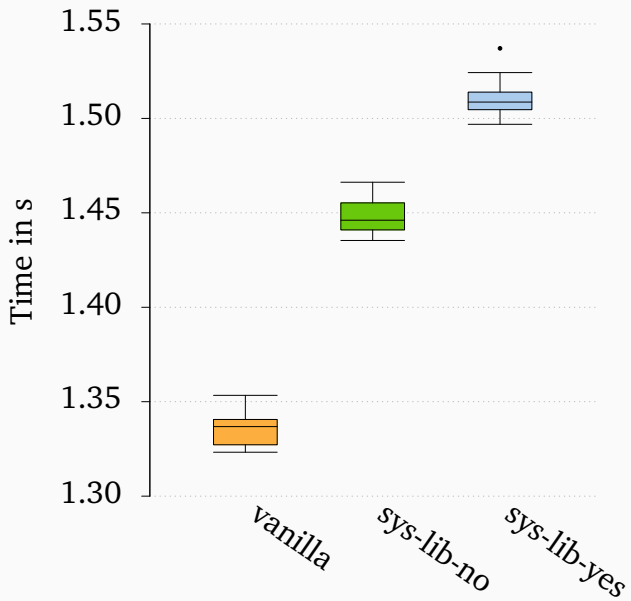
# MG Benchmark




(a) `x86guest`

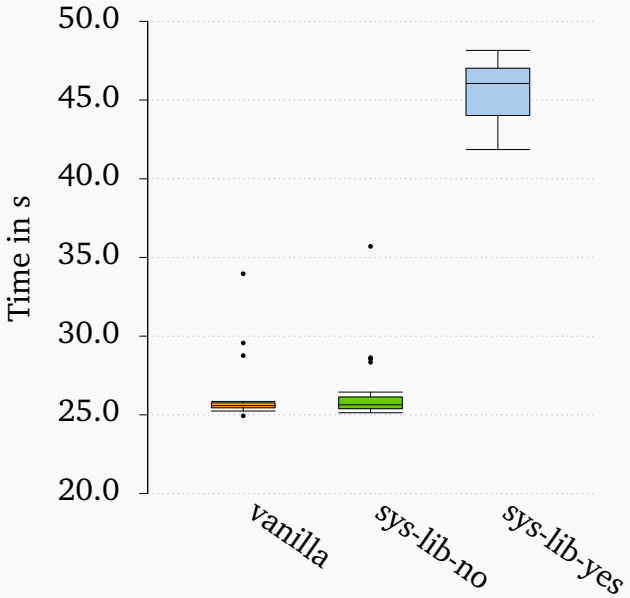



(b) `x64native`

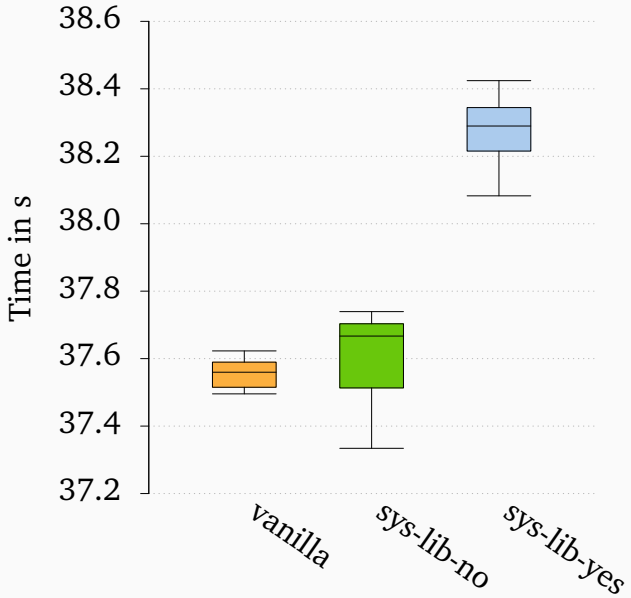



(c) `leon`

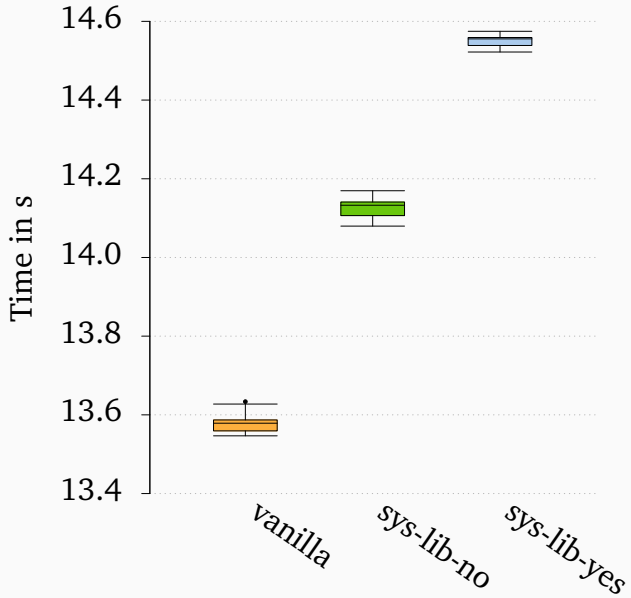
# SP Benchmark




**(a)** `x86guest`




**(b)** `x64native`




**(c)** `leon`