Multiverse: Compiler-Assisted Management of Dynamic Variability in Low-Level System Software

Florian Rommel Christian Dietrich Michael Rodin Daniel Lohmann

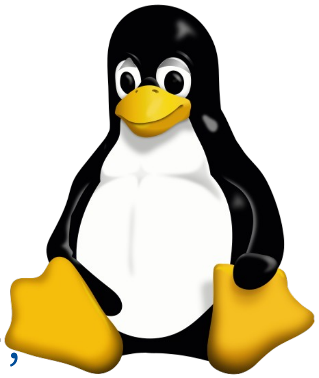
{rommel, dietrich, lohmann}@sra.uni-hannover.de michael@rodin.online

Leibniz Universität Hannover, Germany

2019-03-28

Static Variability in Linux

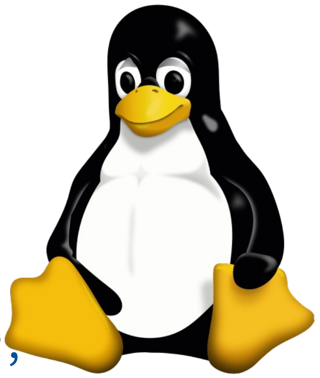
```
void __init sched_init(void)
{
    int i, j;
    unsigned long alloc_size = 0, ptr;

    wait_bit_init();

#ifdef CONFIG_FAIR_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
#endif
#ifdef CONFIG_RT_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
#endif
    if (alloc_size) {
        ptr = (unsigned long)kzalloc(alloc_size, GFP_NOWAIT);

#ifdef CONFIG_FAIR_GROUP_SCHED
        root_task_group.se = (struct sched_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);

        root_task_group.cfs_rq = (struct cfs_rq **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);

#endif /* CONFIG_FAIR_GROUP_SCHED */
#ifdef CONFIG_RT_GROUP_SCHED
        root_task_group.rt_se = (struct sched_rt_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);
```



Static Variability in Linux

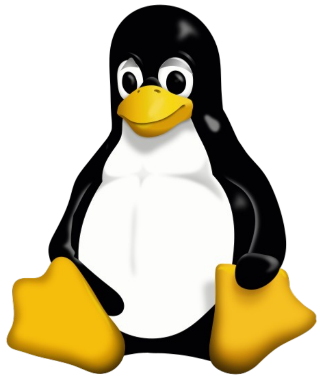
```
void __init sched_init(void)
{
    int i, j;
    unsigned long alloc_size = 0, ptr;

    wait_bit_init();

#ifdef CONFIG_FAIR_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
#endif
#ifdef CONFIG_RT_GROUP_SCHED
    alloc_size += 2 * nr_cpu_ids * sizeof(void **);
#endif
    if (alloc_size) {
        ptr = (unsigned long)kzalloc(alloc_size, GFP_NOWAIT);

#ifdef CONFIG_FAIR_GROUP_SCHED
        root_task_group.se = (struct sched_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);

        root_task_group.cfs_rq = (struct cfs_rq **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);

#endif /* CONFIG_FAIR_GROUP_SCHED */
#ifdef CONFIG_RT_GROUP_SCHED
        root_task_group.rt_se = (struct sched_rt_entity **)ptr;
        ptr += nr_cpu_ids * sizeof(void **);
```

Linux 5.0:
43918 #ifdefs



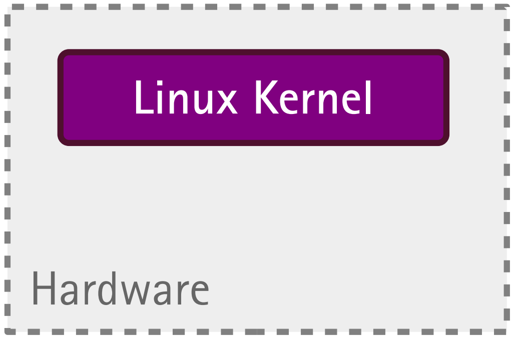
Dynamic Variability in Linux

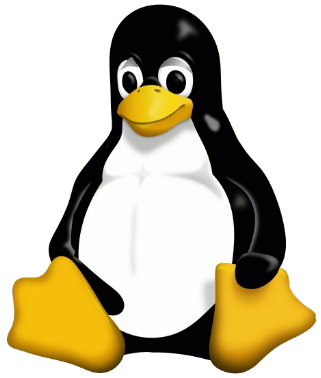
Example: Operations for Paravirtualized Kernels (PV-Ops)



Dynamic Variability in Linux

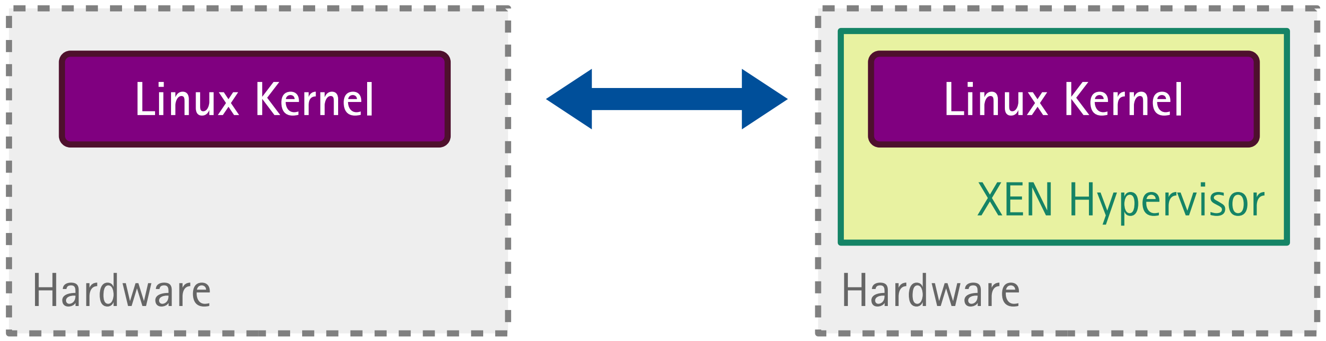
Example: Operations for Paravirtualized Kernels (PV-Ops)

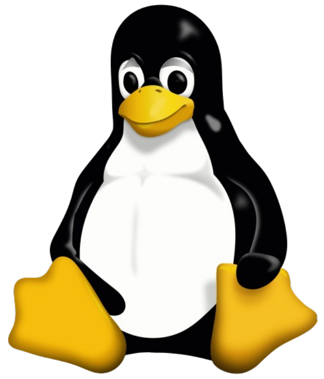




Dynamic Variability in Linux

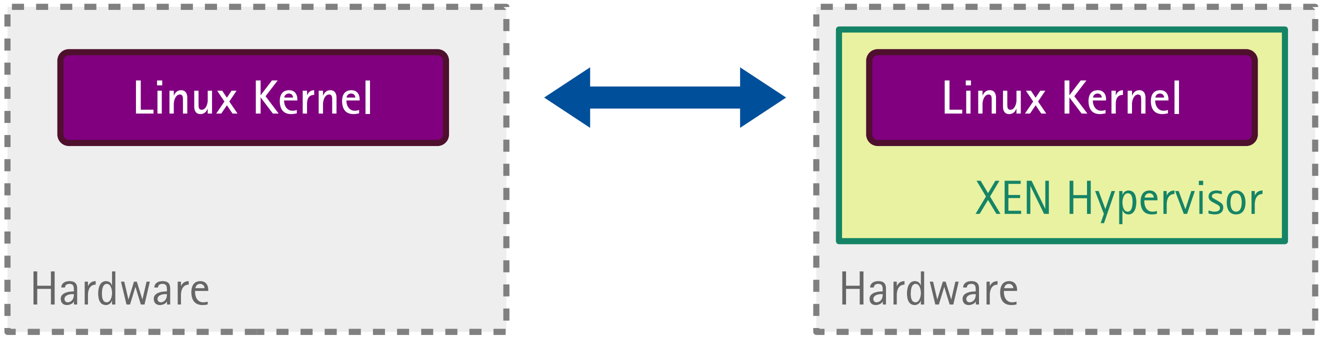
Example: Operations for Paravirtualized Kernels (PV-Ops)



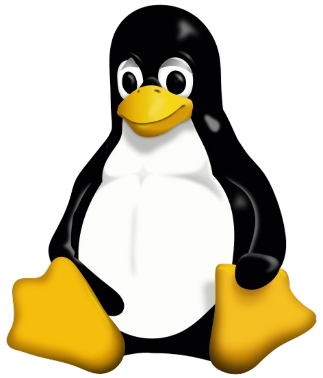


Dynamic Variability in Linux

Example: Operations for Paravirtualized Kernels (PV-Ops)

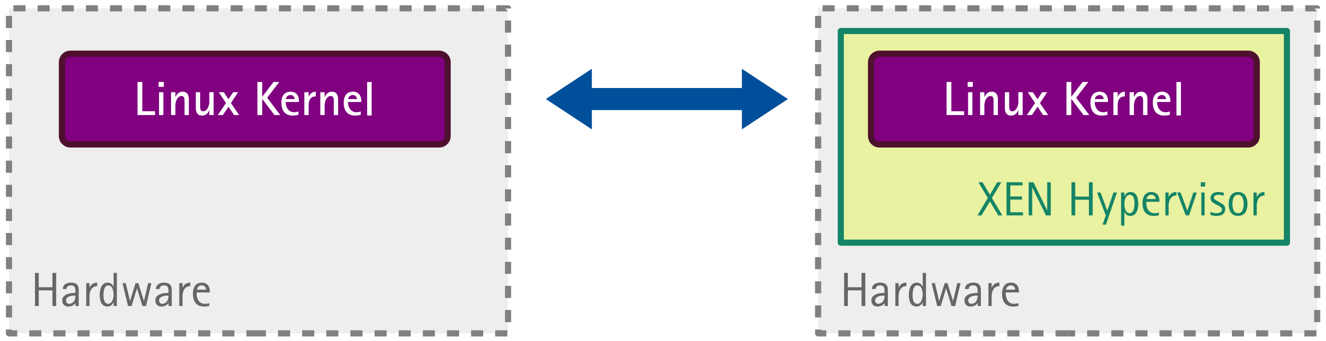


- Inside paravirtualization:
  Privileged operations must be replaced by calls to the hypervisor
  (e.g., enable/disable interrupts)



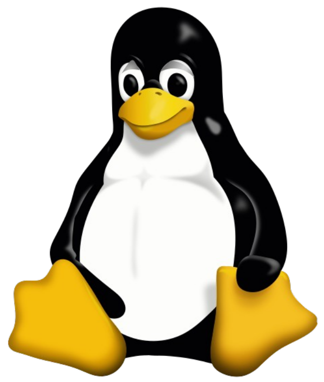
Dynamic Variability in Linux

Example: Operations for Paravirtualized Kernels (PV-Ops)



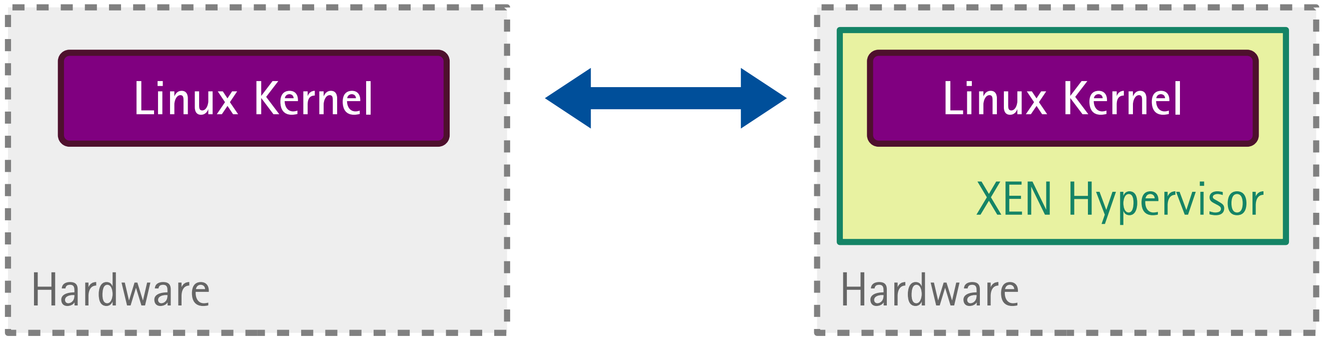
- Inside paravirtualization:
  Privileged operations must be replaced by calls to the hypervisor
  (e.g., enable/disable interrupts)

- Implemented by function pointers

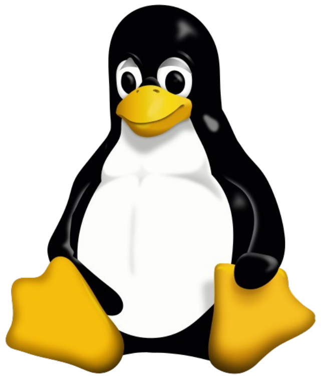


Dynamic Variability in Linux

Example: Operations for Paravirtualized Kernels (PV-Ops)

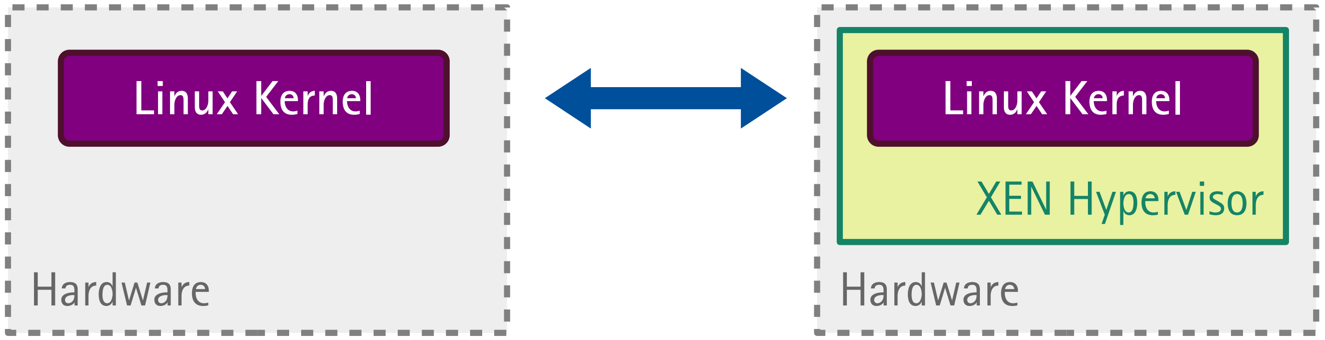


- Inside paravirtualization:
  Privileged operations must be replaced by calls to the hypervisor
  (e.g., enable/disable interrupts)
- Implemented by function pointers → **too much overhead**

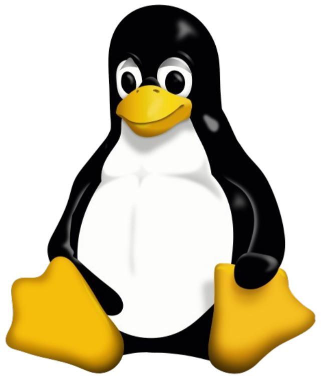


Dynamic Variability in Linux

Example: Operations for Paravirtualized Kernels (PV-Ops)



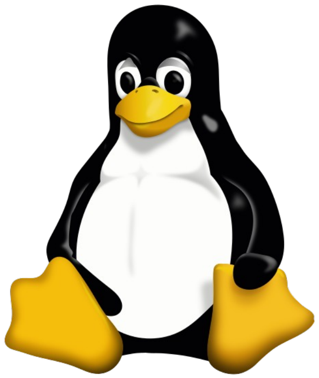
- Inside paravirtualization:
  Privileged operations must be replaced by calls to the hypervisor
  (e.g., enable/disable interrupts)
- Implemented by function pointers → **too much overhead**
- Run-time binary patching:
  Replace indirect calls by direct calls



Dynamic Variability in Linux

arch/x86/include/asm/paravirt_types.h

```
#define _paravirt_alt(insn_string, type, clobber) \
    "771:\n\t" insn_string "\n" "772:\n"          \
    ".pushsection .parainstructions,\"a\"\n"      \
    _ASM_ALIGN "\n"                               \
    _ASM_PTR " 771b\n"                            \
    "   .byte " type "\n"                         \
    "   .byte 772b-771b\n"                        \
    "   .short " clobber "\n"                     \
    ".popsection\n"
...
#define paravirt_alt(insn_string)                 \
    _paravirt_alt(insn_string, "%c[paravirt_typenum]",
↪                              "%c[paravirt_clobber]")
...
asm volatile(pre                                  \
             paravirt_alt(PARAVIRT_CALL)          \
             post                                 \
             : call_clbr, ASM_CALL_CONSTRAINT     \
             : paravirt_type(op),                 \
               paravirt_clobber(clbr),            \
               ##__VA_ARGS__                      \
             : "memory", "cc" extra_clbr);        \
```



Dynamic Variability in Linux

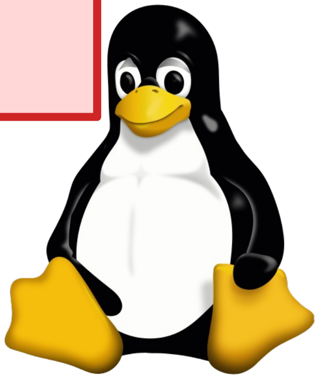
arch/x86/include/asm/paravirt_types.h

```
#define ____paravirt_alt(insn_string, type, clobber) \
```

- Complex implementation:

  PV-Ops: 7 files, ~2000 loc (for x86)

```
                 post                                    \
                 : call_clbr, ASM_CALL_CONSTRAINT        \
                 : paravirt_type(op),                    \
                   paravirt_clobber(clbr),               \
                   ##__VA_ARGS__                         \
                 : "memory", "cc" extra_clbr);           \
```



Dynamic Variability in Linux

arch/x86/include/asm/paravirt_types.h

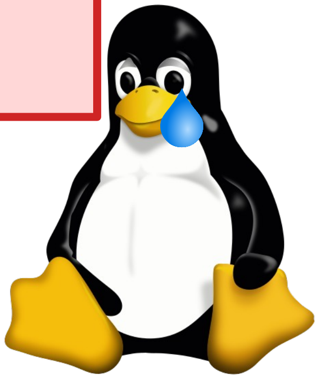
```
#define __paravirt_alt(insn_string, type, clobber) \
```

- Complex implementation:

  PV-Ops: 7 files, ~2000 loc (for x86)

- Highly architecture-dependent: Multiple implementations

```
                     post                                  \
                     : call_clbr, ASM_CALL_CONSTRAINT      \
                     : paravirt_type(op),                  \
                       paravirt_clobber(clbr),             \
                       ##__VA_ARGS__                       \
                     : "memory", "cc" extra_clbr);         \
```



Dynamic Variability in Linux

```
arch/x86/include/asm/paravirt_types.h

#define __paravirt_alt(insn_string, type, clobber) \
```

- Complex implementation:

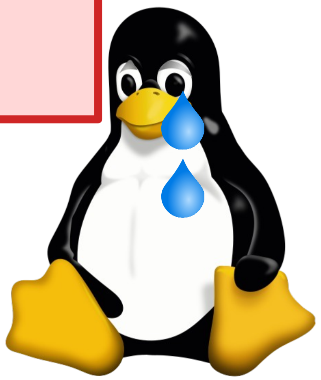
  PV-Ops: 7 files, ~2000 loc (for x86)

- Highly architecture-dependent: Multiple implementations

- Highly problem-specific: Multiple implementations

  e.g., alternative instructions, SMP alternatives

```
                post                                    \
                : call_clbr, ASM_CALL_CONSTRAINT        \
                : paravirt_type(op),                    \
                  paravirt_clobber(clbr),               \
                  ##__VA_ARGS__                         \
                : "memory", "cc" extra_clbr);           \
```



Dynamic Variability in Linux

arch/x86/include/asm/paravirt_types.h

```
#define ____paravirt_alt(insn_string, type, clobber) \
```

- Complex implementation:

  PV-Ops: 7 files, ~2000 loc (for x86)

- Highly architecture-dependent: Multiple implementations

- Highly problem-specific: Multiple implementations

  e.g., alternative instructions, SMP alternatives

→ Means for efficient dynamic variability are rarely used

```
                        post                                      \
                        : call_clbr, ASM_CALL_CONSTRAINT          \
                        : paravirt_type(op),                      \
                          paravirt_clobber(clbr),                 \
                          ##__VA_ARGS__                           \
                        : "memory", "cc" extra_clbr);             \
```

Efficient Dynamic Variability

Problems:

- Default approach: Performance costs (e.g., branches, function pointers)
- Binary patching: Code complexity → maintenance costs

Efficient Dynamic Variability

Problems:

- Default approach: Performance costs (e.g., branches, function pointers)
- Binary patching: Code complexity → maintenance costs

Multiverse

Compiler-Assisted Dynamic Variability via Binary Patching

Efficient Dynamic Variability

Problems:

- Default approach: Performance costs (e.g., branches, function pointers)
- Binary patching: Code complexity → maintenance costs

Multiverse

Compiler-Assisted Dynamic Variability via Binary Patching

Language extension to express efficient dynamic variability

- Express binary patching via standard control flow modification (if, ...)
- Generic mechanism for function-level run-time patching

Efficient Dynamic Variability

Problems:

- Default approach: Performance costs (e.g., branches, function pointers)
- Binary patching: Code complexity → maintenance costs

Multiverse

Compiler-Assisted Dynamic Variability via Binary Patching

Language extension to express efficient dynamic variability

- Express binary patching via standard control flow modification (if, ...)
- Generic mechanism for function-level run-time patching

→ *Compiler plugin + small run-time library*

Example: Linux Lock Elision

Linux Spinlock Implementation (simplified):

`CONFIG_SMP` set in the build system

```
void spin_irq_lock(raw_spinlock_t *lock) {
#ifdef CONFIG_SMP
    irq_disable();
    spin_acquire(&lock);
#else
    irq_disable();
#endif
}
```

Example: Linux Lock Elision

Linux Spinlock Implementation (simplified):

`CONFIG_SMP` set in the build system

```
void spin_irq_lock(raw_spinlock_t *lock) {
#ifdef CONFIG_SMP
    irq_disable();
    spin_acquire(&lock);
#else
    irq_disable();
#endif
}
```

Example: Linux Lock Elision

Linux Spinlock Implementation (simplified):

`CONFIG_SMP` set in the build system

```
void spin_irq_lock(raw_spinlock_t *lock) {
#ifdef CONFIG_SMP
    irq_disable();
    spin_acquire(&lock);
#else
    irq_disable();
#endif
}
```

Example: Linux Lock Elision

```
bool smp;


void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

Example: Linux Lock Elision

```
bool smp;


void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

branched control flow
run-time overhead

Example: Linux Lock Elision

```
bool smp;



void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

```
void foo(void) {
    smp = true;
```

Example: Linux Lock Elision

```
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

```
void foo(void) {
    smp = true;
```

Example: Linux Lock Elision

```
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

```
void foo(void) {
    smp = true;
    multiverse_commit();
    // …

    spin_irq_lock(lock);
}
```

Example: Linux Lock Elision

```
__attribute__((multiverse))
bool smp;


__attribute__((multiverse))
void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

```
void foo(void) {
    smp = true;
    multiverse_commit();
    // …


    spin_irq_lock(lock);
}
```

Example: Linux Lock Elision

```
__attribute__((multiverse))
bool smp;

__attribute__((multiverse))
void spin_irq_lock(…) {
  if (smp) {
    irq_disable();
    spin_acquire(&lock);
  } else {
    irq_disable();
  }
}
```

```
void foo(void) {
    smp = false;
    multiverse_commit();
    // …

    spin_irq_lock(lock);
}
```

Ahead-of-Time Variant Generation

Quellcode

```
__attribute__((multiverse))
bool smp;
```

```
__attribute__((multiverse))
void spin_irq_lock(…) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}
```

```
void foo() {
    //...
    spin_irq_lock();
    //...
}
```

Ahead-of-Time Variant Generation

Quellcode

```
__attribute__((multiverse))
bool smp;
```

```
__attribute__((multiverse))
void spin_irq_lock(…) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}
```

```
void foo() {
    //...
    spin_irq_lock();
    //...
}
```

Code Segment

```
spin_irq_lock:
    cmp   <smp>, 0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

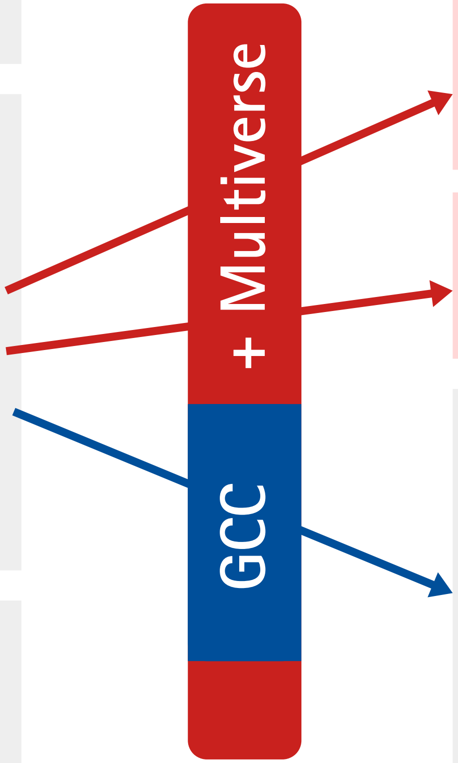
Ahead-of-Time Variant Generation

Quellcode

```
__attribute__((multiverse))
bool smp;
```

```
__attribute__((multiverse))
void spin_irq_lock(…) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}
```

```
void foo() {
    //...
    spin_irq_lock();
    //...
}
```



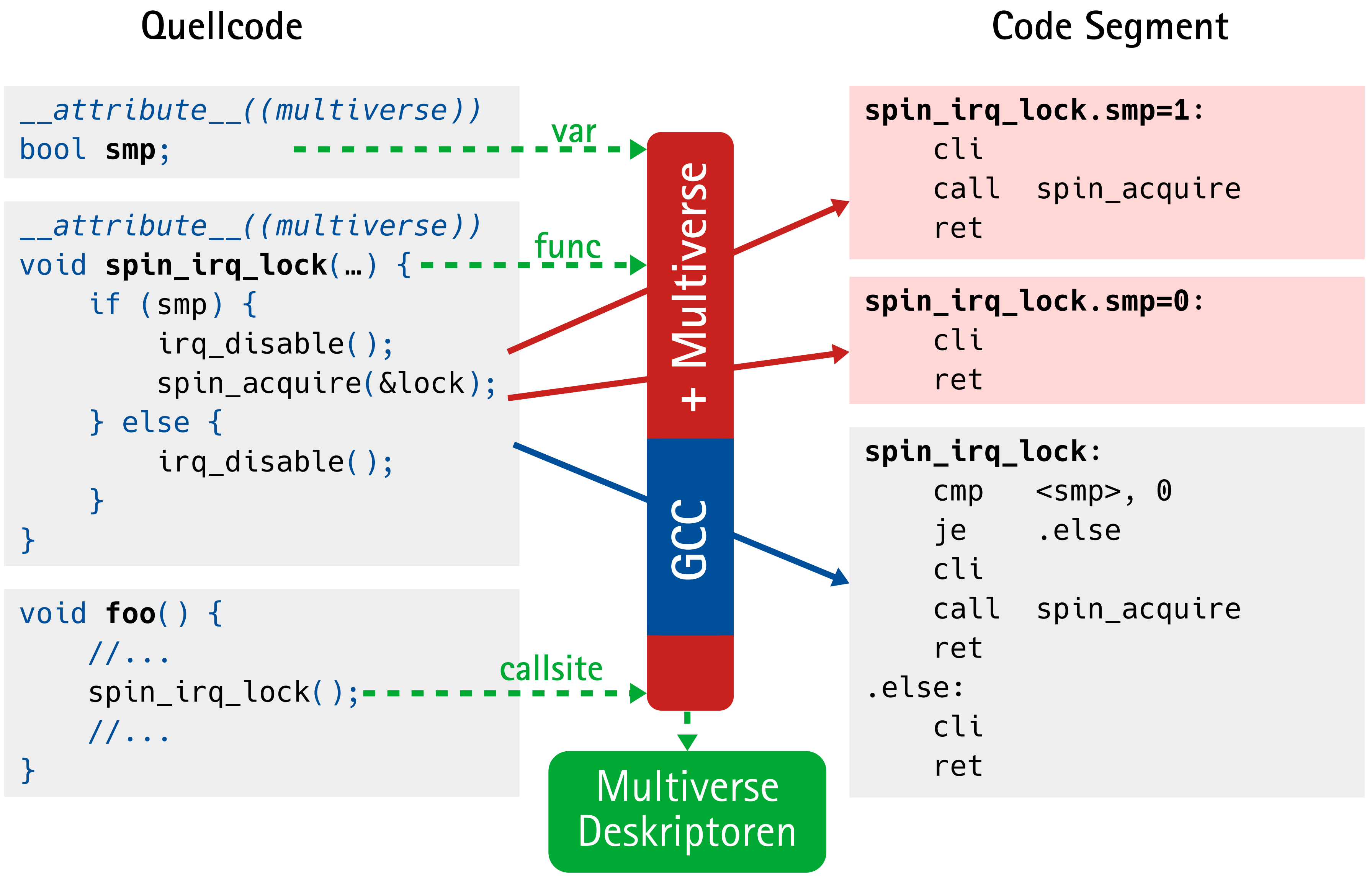
Code Segment

```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    cmp   <smp>, 0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Ahead-of-Time Variant Generation



Quellcode

```
__attribute__((multiverse))
bool smp;
```

```
__attribute__((multiverse))
void spin_irq_lock(…) {
    if (smp) {
        irq_disable();
        spin_acquire(&lock);
    } else {
        irq_disable();
    }
}
```

```
void foo() {
    //...
    spin_irq_lock();
    //...
}
```

Code Segment

```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    cmp   <smp>, 0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching

Initial geladenes Code Segment

```
foo:
    ...
    call  multiverse_commit
    ...
    call  spin_irq_lock
    ...
    ret
```

```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

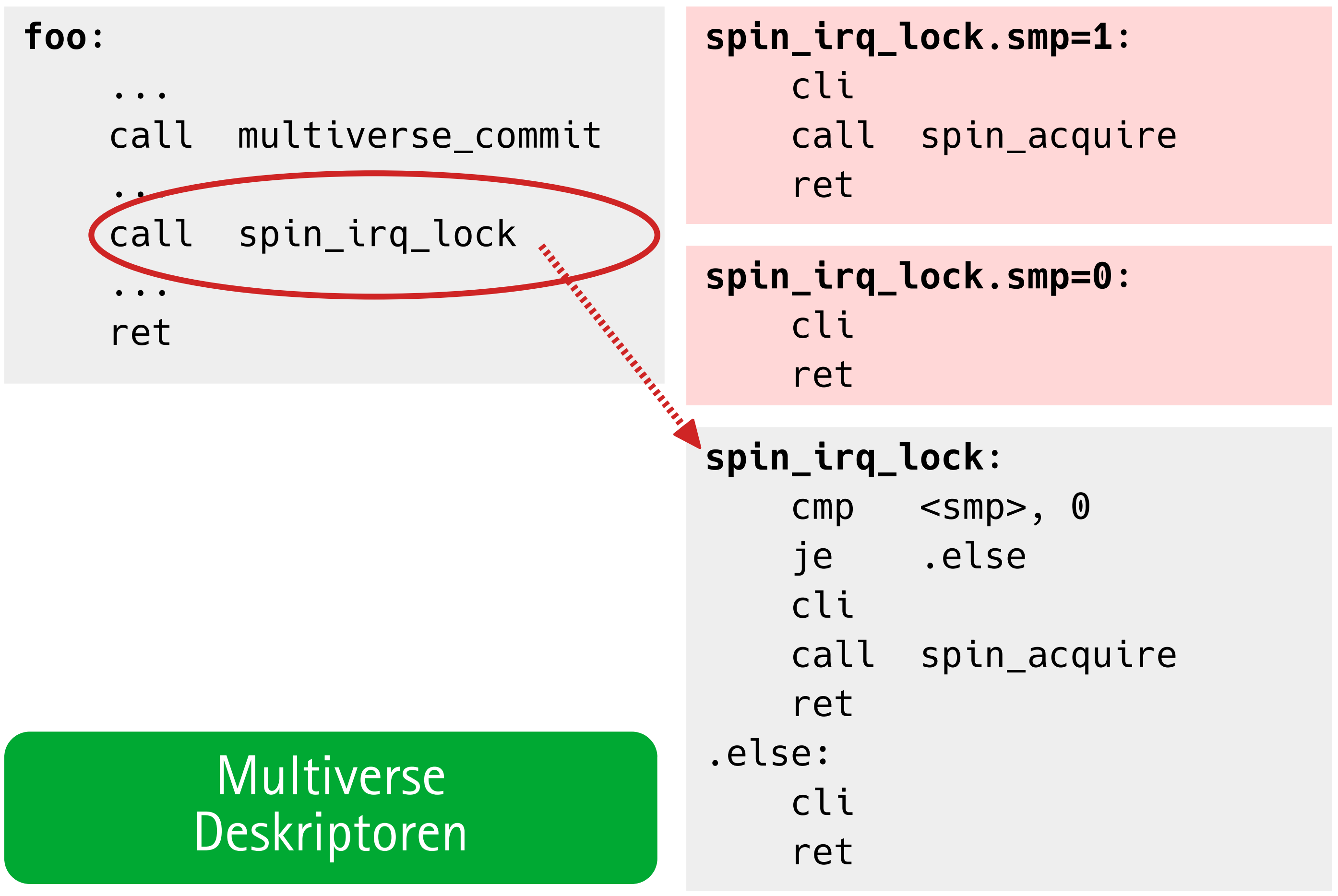
```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    cmp   <smp>, 0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Multiverse Deskriptoren

Run-Time Patching

Initial geladenes Code Segment



```
foo:
    ...
    call  multiverse_commit
    ...
    call  spin_irq_lock
    ...
    ret
```

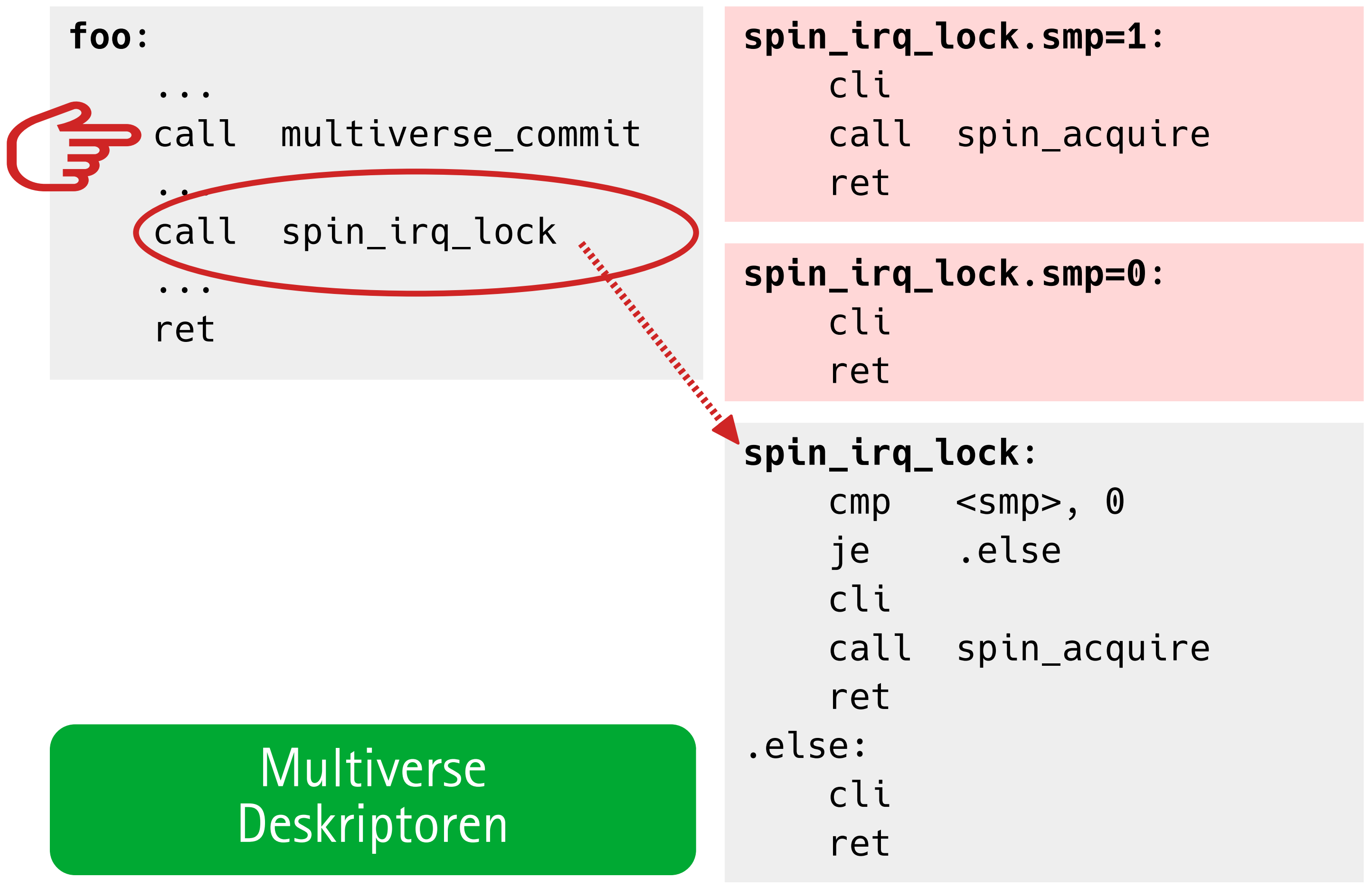
```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    cmp   <smp>, 0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching

Initial geladenes Code Segment



```
foo:
    ...
    call   multiverse_commit
    ...
    call   spin_irq_lock
    ...
    ret
```

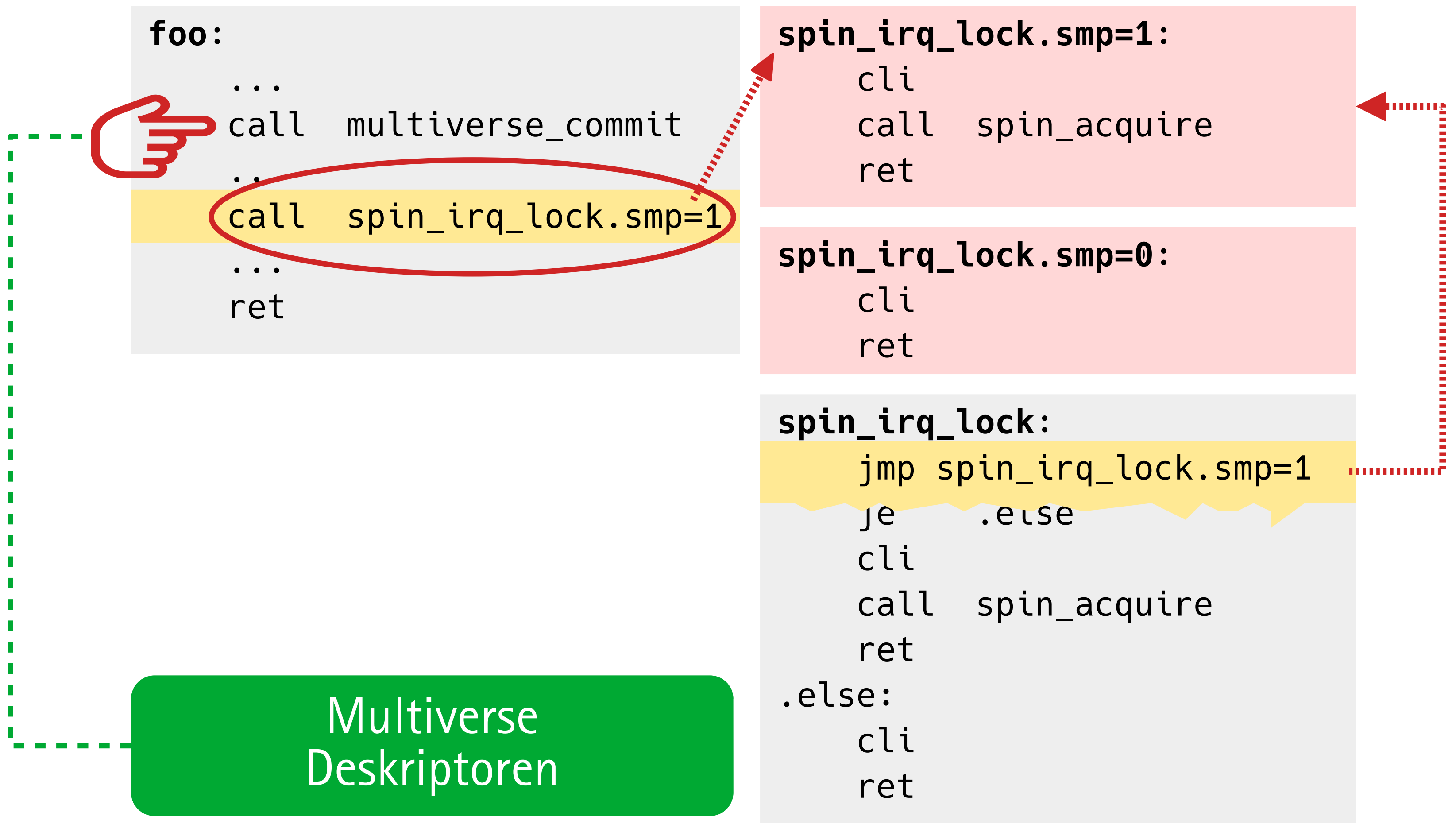
```
spin_irq_lock.smp=1:
    cli
    call   spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    cmp    <smp>, 0
    je     .else
    cli
    call   spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching

Patched (smp == 1)



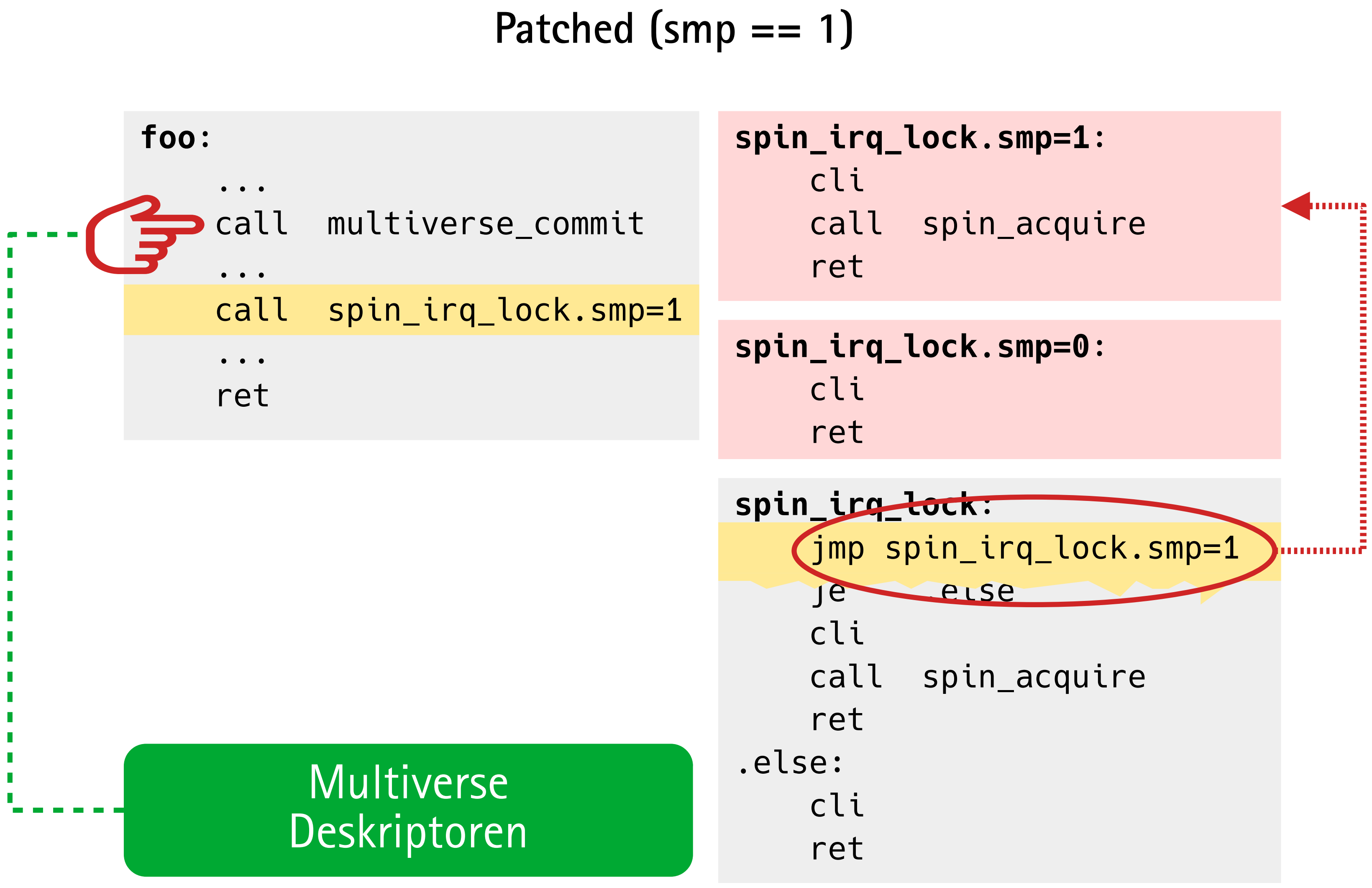
```
foo:
    ...
    call  multiverse_commit
    ...
    call  spin_irq_lock.smp=1
    ...
    ret
```

```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    jmp spin_irq_lock.smp=1
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching



Patched (smp == 1)

```
foo:
    ...
    call  multiverse_commit
    ...
    call  spin_irq_lock.smp=1
    ...
    ret
```

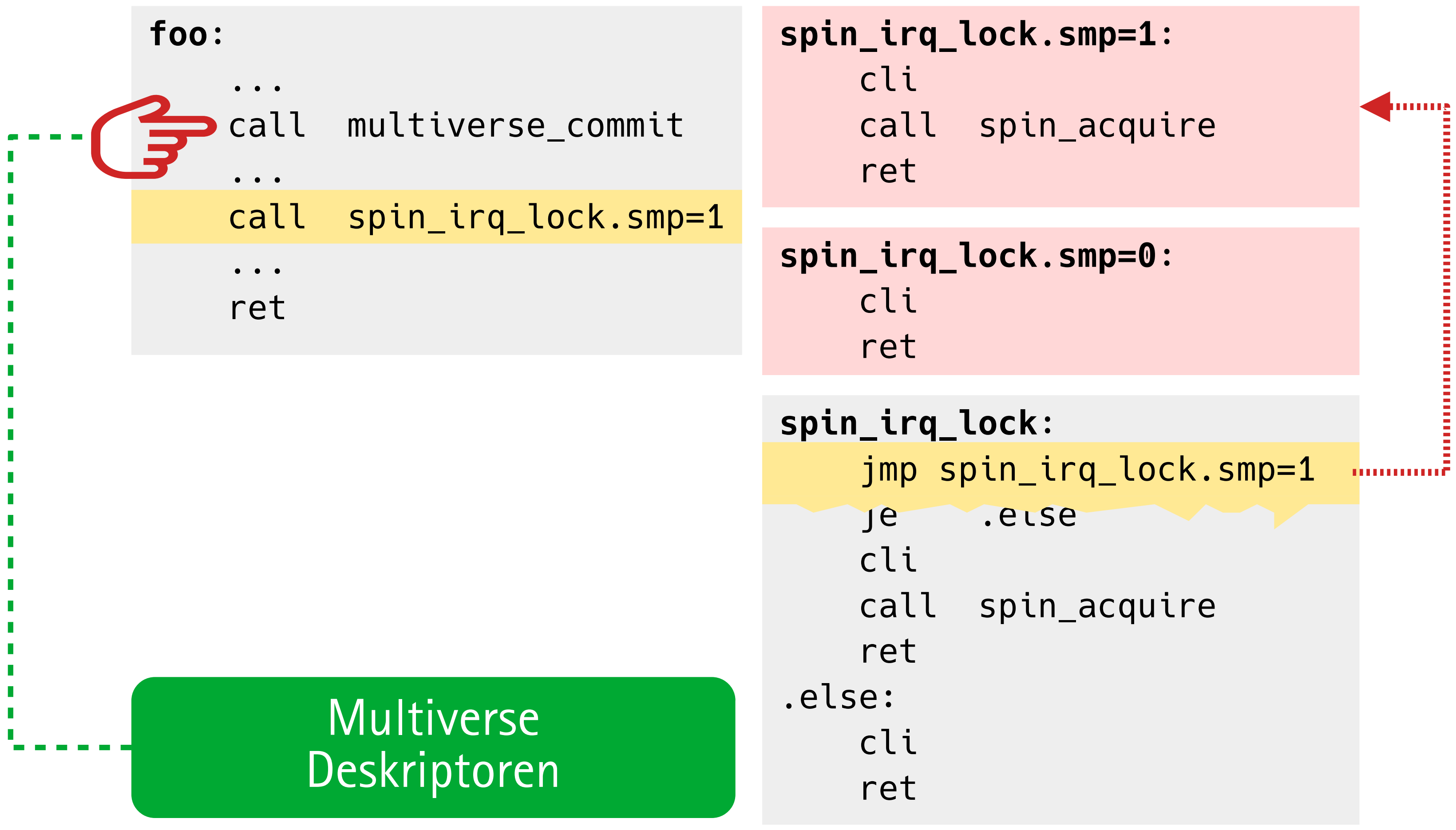
```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    jmp spin_irq_lock.smp=1
    je     .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching

Patched (smp == 1)



```
foo:
    ...
    call  multiverse_commit
    ...
    call  spin_irq_lock.smp=1
    ...
    ret
```

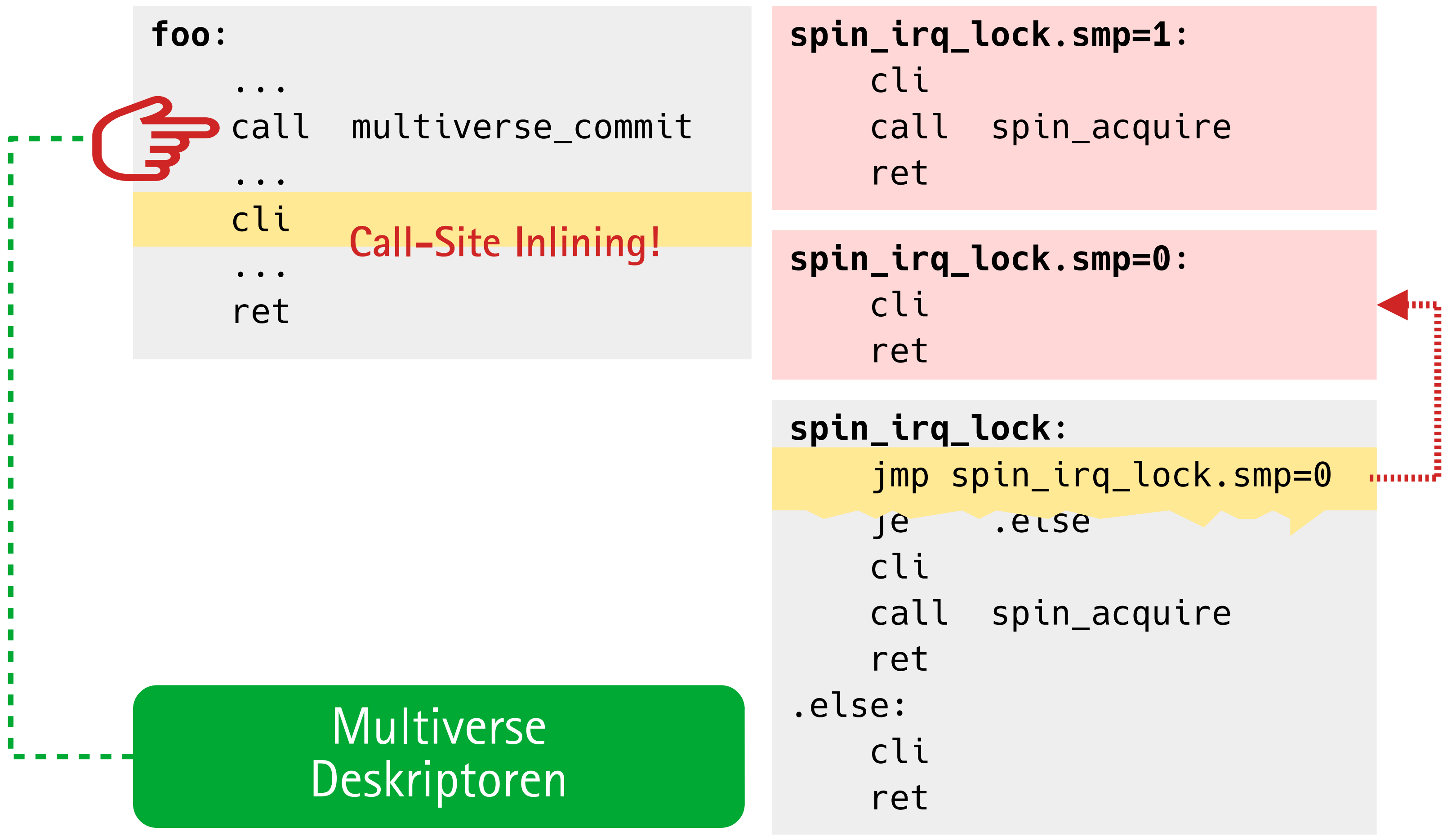
```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    jmp spin_irq_lock.smp=1
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

Run-Time Patching

Patched (smp == 0)



```
foo:
    ...
    call  multiverse_commit
    ...
    cli
    ...
    ret
```

Call-Site Inlining!

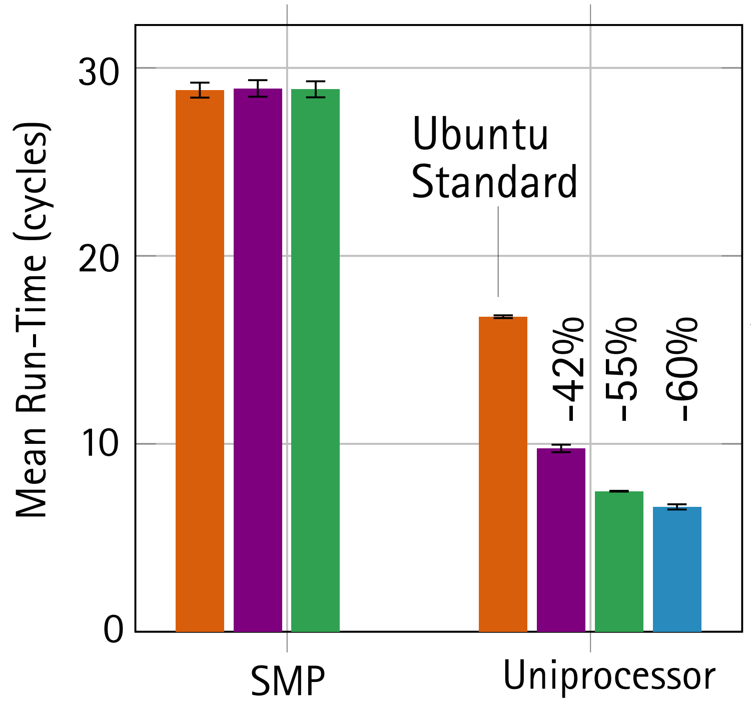
```
spin_irq_lock.smp=1:
    cli
    call  spin_acquire
    ret
```

```
spin_irq_lock.smp=0:
    cli
    ret
```

```
spin_irq_lock:
    jmp spin_irq_lock.smp=0
    je    .else
    cli
    call  spin_acquire
    ret
.else:
    cli
    ret
```

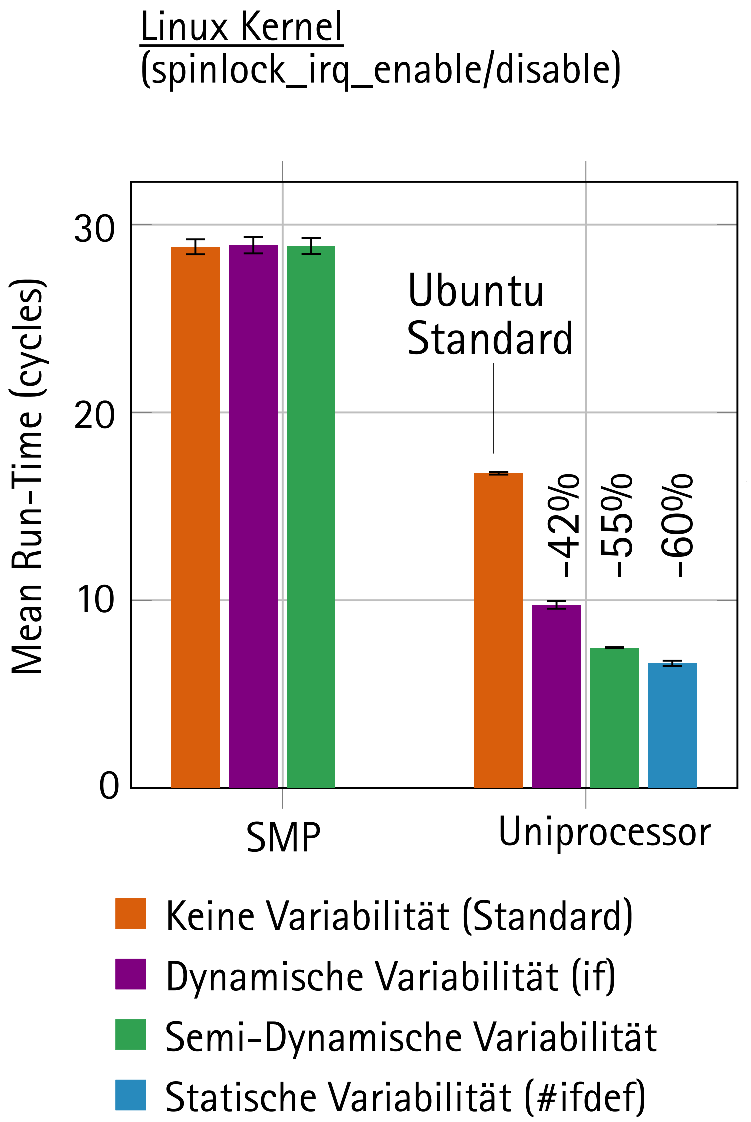
Evaluation: Lock Elision in Kernel und Userspace

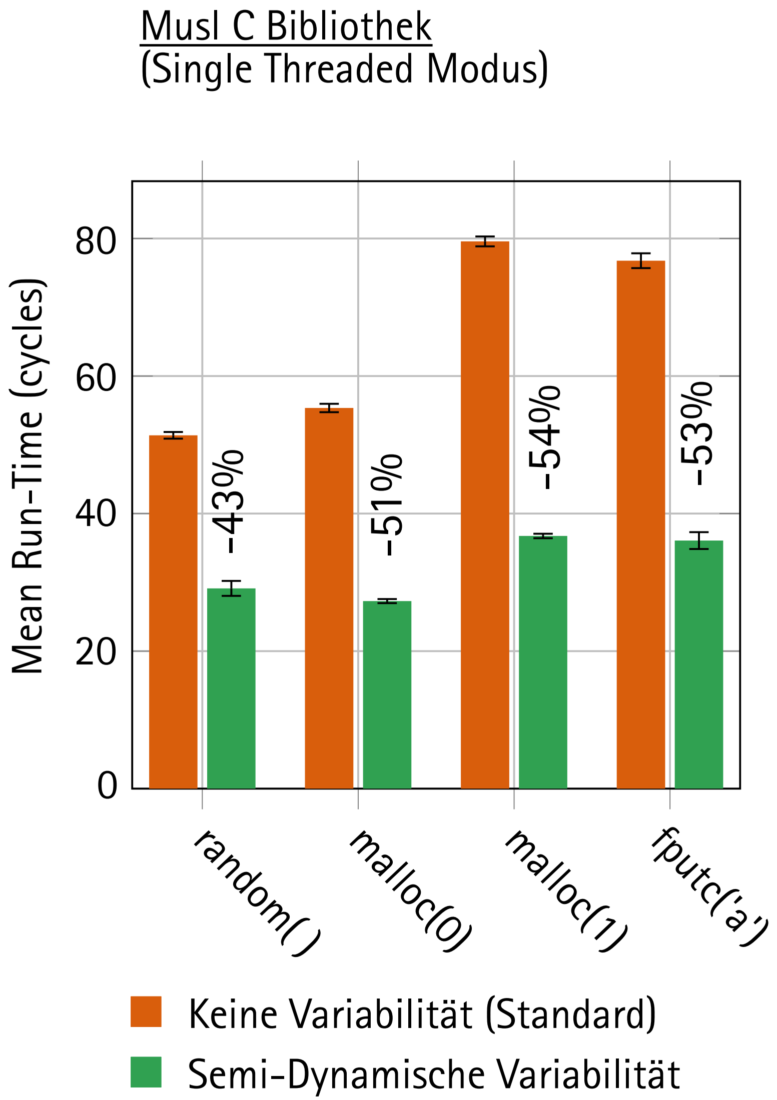
Linux Kernel
(spinlock_irq_enable/disable)



🟧 Keine Variabilität (Standard)
🟪 Dynamische Variabilität (if)
🟩 Semi-Dynamische Variabilität
🟦 Statische Variabilität (#ifdef)

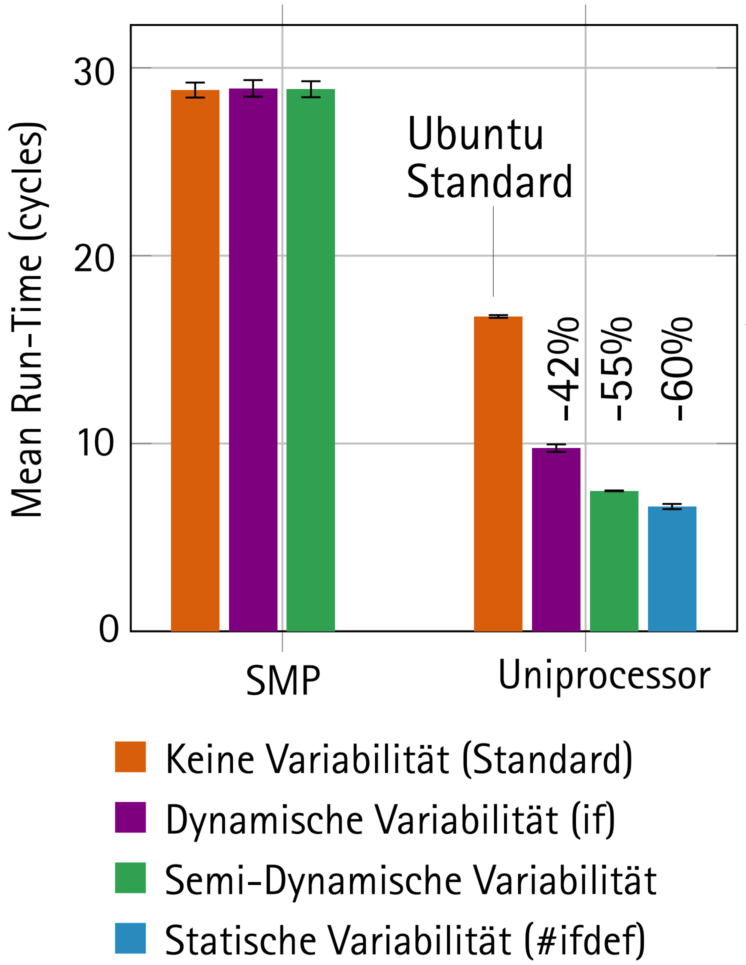
Evaluation: Lock Elision in Kernel und Userspace

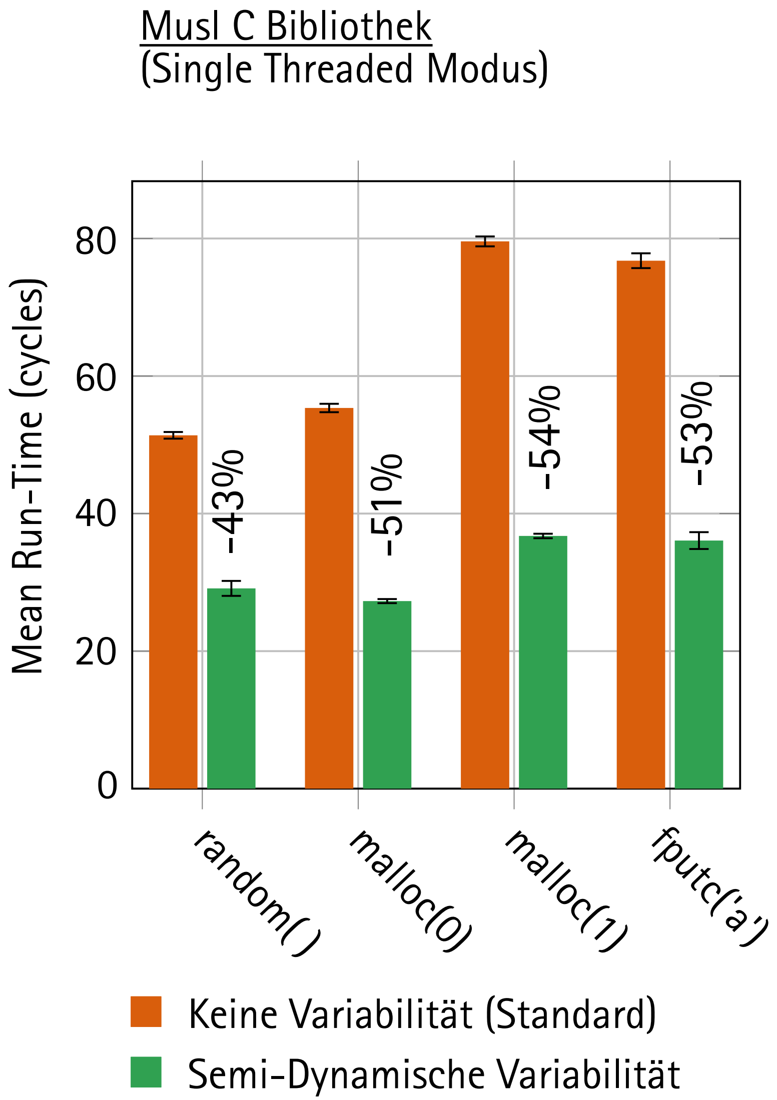




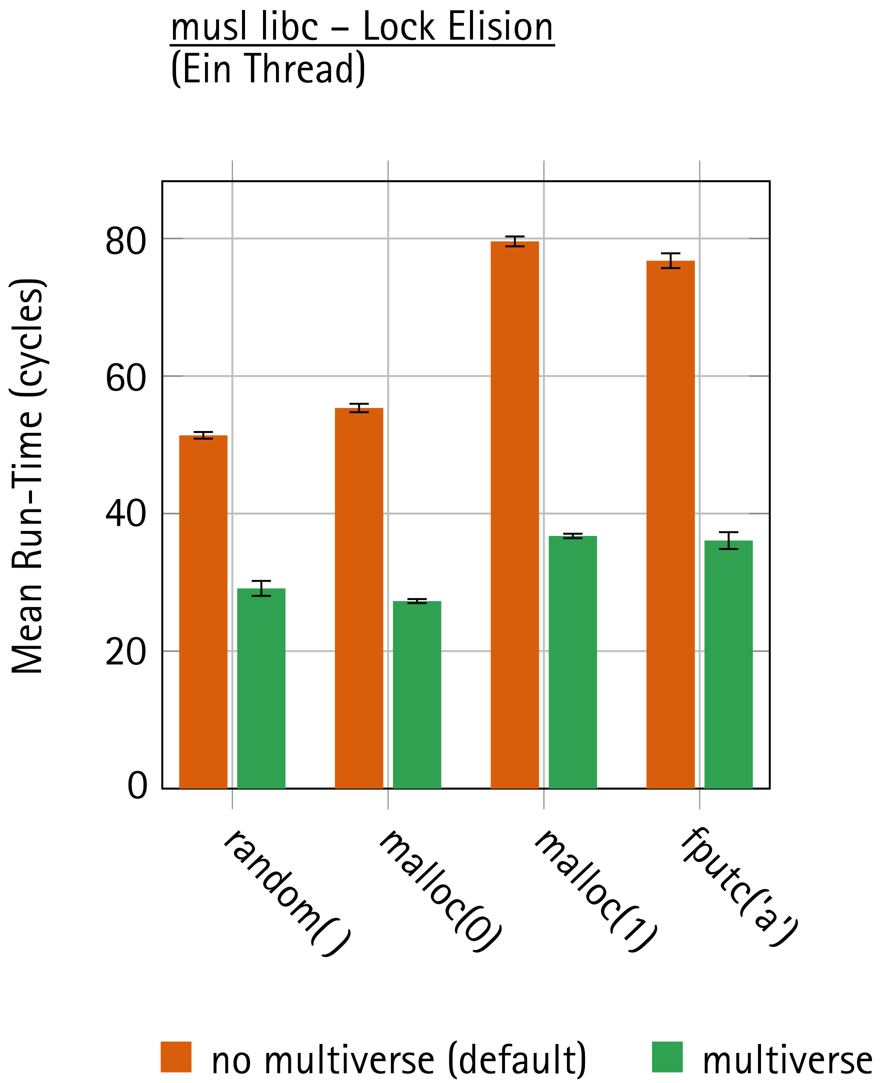
Evaluation: Lock Elision in Kernel und Userspace

Lock Elision in Kernel 4.16:
→ 1161 spin-lock call sites
→ +40 KiB size (zipped total: 10 MiB)

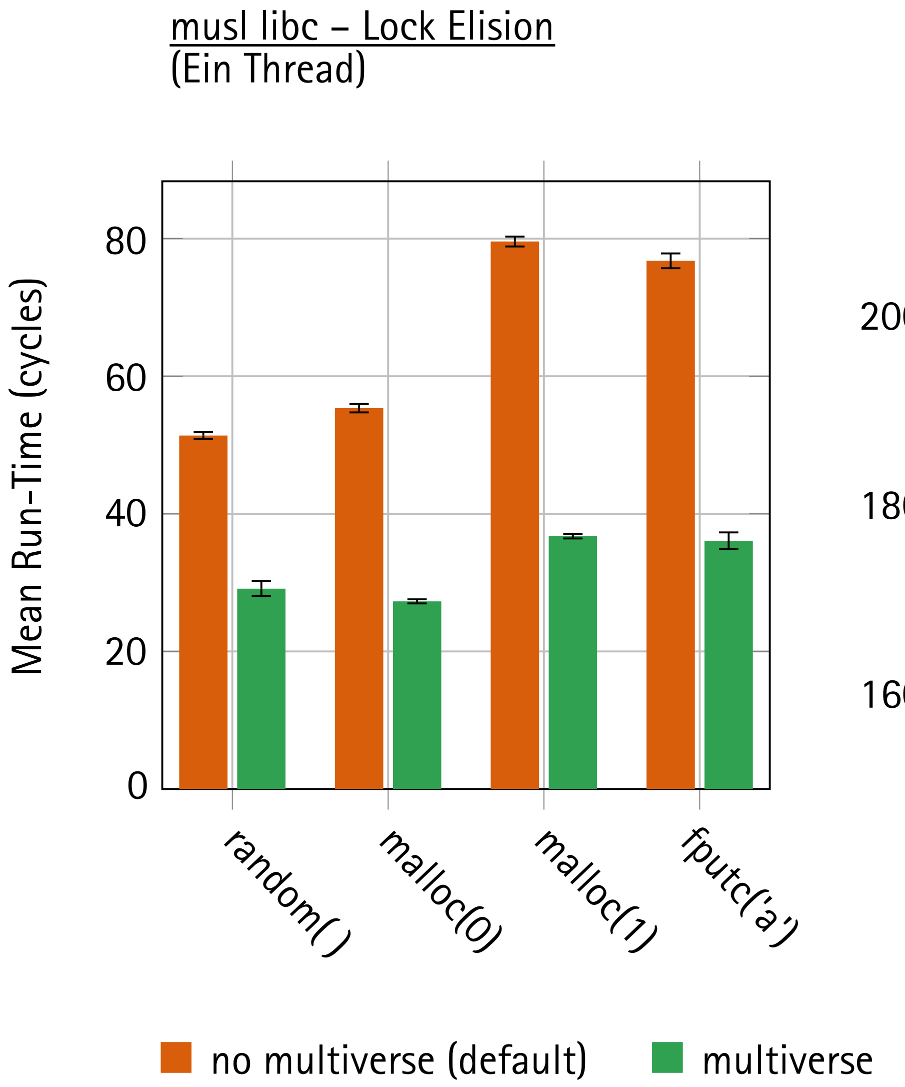




Evaluation – Userspace Microbenchmarks



Evaluation – Userspace Microbenchmarks



Evaluation

- **GNU Grep**

  → Optimized for more than 30 years

  Multiversed a conditional branch in the inner loop (recognition of multi-byte characters on/off)

Evaluation

- **GNU Grep**

  → Optimized for more than 30 years

  Multiversed a conditional branch in the inner loop (recognition of multi-byte characters on/off)

  → End-to-end measurement: **–2.73 % run-time**

  → Only **50** changed lines to use multiverse

Evaluation

- **GNU Grep**

  → Optimized for more than 30 years

  Multiversed a conditional branch in the inner loop (recognition of multi-byte characters on/off)

  → End-to-end measurement: **–2.73 % run-time**

  → Only **50** changed lines to use multiverse

- **Multiverse – Numbers**

<u>GCC Plugin</u>

Lines of Code:	<1200
+ compatibility headers:	~1300
GCC version:	6.3 and higher

<u>Run-Time Library</u>

Lines of Code:	< 850
Compiled:	6.5 KiB
Architectures:	IA-32, AMD64 *ARM [to come]*

Summary

Multiverse

Compiler-Assisted Dynamic Variability via Binary Patching

- Language extension for easy-to-use, efficient dynamic variability

GCC-Plugin:	Generates specialized function variants
Run-Time Library:	Function-level binary patching

- Evaluation
  - Consolidation of current patching mechanisms (PV-Ops)
  - Introduction of new dynamic variation points (Lock Elision, Grep)
- Try it: *https://github.com/luhsra/multiverse*