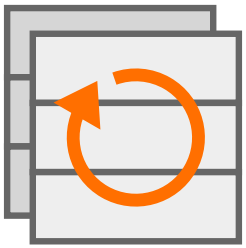


Operating
System
Group

TUHH
Technische Universität Hamburg

Semi-Dynamic Variability

Between Static Configuration and Dynamic Reconfiguration

Christian Dietrich

13. September 2021

Special-Purpose Systems

Different Plattforms

Security-Updates

Dynamic Scalability

Debugging & Introspection

Extendable and/or Reducible Feature Set

Special-Purpose Systems

Different Plattforms

Security-Updates

Dynamic Scalability

Debugging & Introspection

Extendable and/or Reducible Feature Set

Special-Purpose Systems

Different Plattforms

Security-Updates

Dynamic Scalability

Debugging & Introspection

Extendable and/or Reducible Feature Set

Special-Purpose Systems

Different Plattforms

Security-Updates

Dynamic Scalability

Debugging & Introspection

Extendable and/or Reducible Feature Set

Special-Purpose Systems

Different Plattforms

Security-Updates

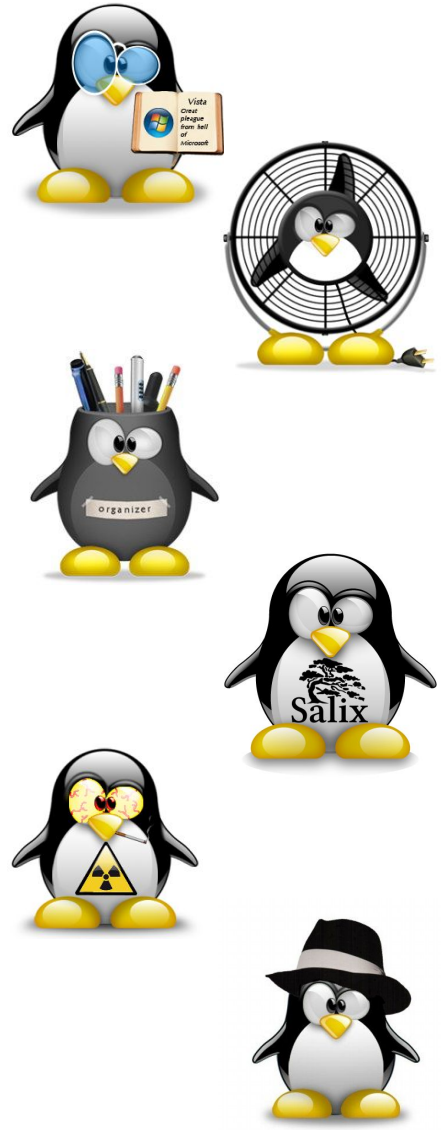
Dynamic Scalability

Debugging & Introspection

Extendable and/or Reducible Feature Set

Variable System Software is Adaptable System Software

Software Product Line



Variable System Software is Adaptable System Software

Software Product Line

Static and Dynamic Variability

TUHH
Technische Universität Hamburg

Creation: of a variant's binary form
Binding: Integration of Variant into Program

Static and Dynamic Variability

TUHH
Technische Universität Hamburg

Creation: of a variant's binary form
Binding: Integration of Variant into Program

```
#ifdef CONFIG_SMP
irq_disable();
spin_acquire(&lock);
#else
irq_disable();
#endif
```

Static Variability

- Creating: Compile-Time
- Binding: Compile-Time

\+ No Run-Time Overhead
− Each combination: A different binary

Static and Dynamic Variability

Technische Universität Hamburg

Creation: of a variant's binary form

Binding: Integration of Variant into Program

```
#ifdef CONFIG_SMP
irq_disable();
spin_acquire(&lock);
#else
irq_disable();
#endif
```

```
if (config_smp) {
  irq_disable();
  spin_acquire(&lock);
} else {
  irq_disable();
}
```

Static Variability

- Creating: Compile-Time
- Binding: Compile-Time

\+ No Run-Time Overhead

− Each combination: A different binary

Dynamic Variability

- Creation: Hybrid Variants
- Binding: Time-of-Use

− Repeated binding costs

\+ Flexible and reconfigurable

Static and Dynamic Variability

TUHH
Technische Universität Hamburg

Creation: of a variant's binary form
Binding: Integration of Variant into Program

```
#ifdef CONFIG_SMP
irq_disable();
spin_acquire(&lock);
#else
irq_disable();
#endif
```

```
if (config_smp) {
  irq_disable();
  spin_acquire(&lock);
} else {
  irq_disable();
}
```

Static Variability

- Creating: Compile-Time
- Binding: Compile-Time

\+ No Run-Time Overhead
− Each combination: A different binary

Dynamic Variability

- Creation: Hybrid Variants
- Binding: Time-of-Use

− Repeated binding costs
\+ Flexible and reconfigurable

⇝ Semi-Dynamic Variability

Static Creation and **efficient** dynamic (re-)binding

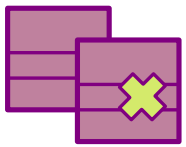
Agenda

TUHH
Technische Universität Hamburg

Multiverse
Efficient Semi-Dynamic Variability

EuroSys'19

- Preparing variants at compile time
- Efficient binding by run-time code patching



Wait-free code patching
Run-time code patching without interruption

OSDI'20

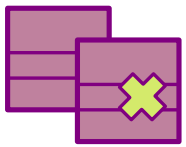
- Preparing changes in a separate address space
- Incremental migration of threads

Agenda

Multiverse
Efficient Semi-Dynamic Variability

EuroSys'19

- Preparing variants at compile time
- Efficient binding by run-time code patching



Wait-free code patching
Run-time code patching without interruption

OSDI'20

- Preparing changes in a separate address space
- Incremental migration of threads

Creation: Compiler Plugin

Technische Universität Hamburg

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();
    //...
}
```

Creation: Compiler Plugin

TUHH
Technische Universität Hamburg

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();
    //...
}
```

GCC

Code Segment

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

Creation: Compiler Plugin

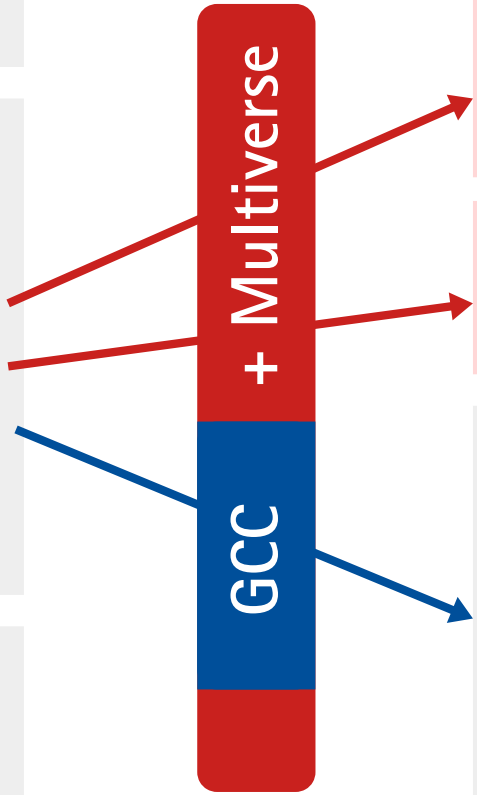
TUHH
Technische Universität Hamburg

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
```

Creation: Compiler Plugin

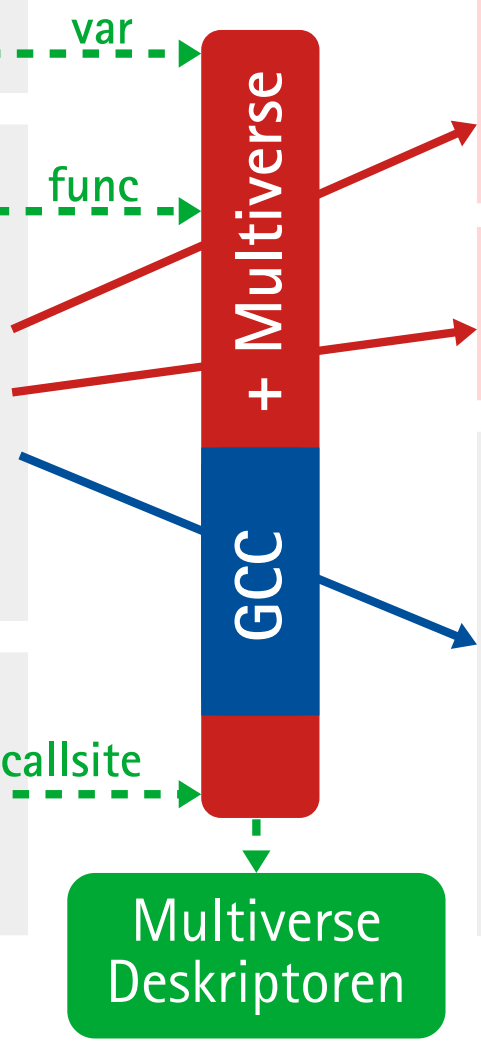
TUHH
Technische Universität Hamburg

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
```

Bindung: Run-Time Code Patching

TUHH
Technische Universität Hamburg

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

Bindung: Run-Time Code Patching

Initial geladenes Code Segment

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

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

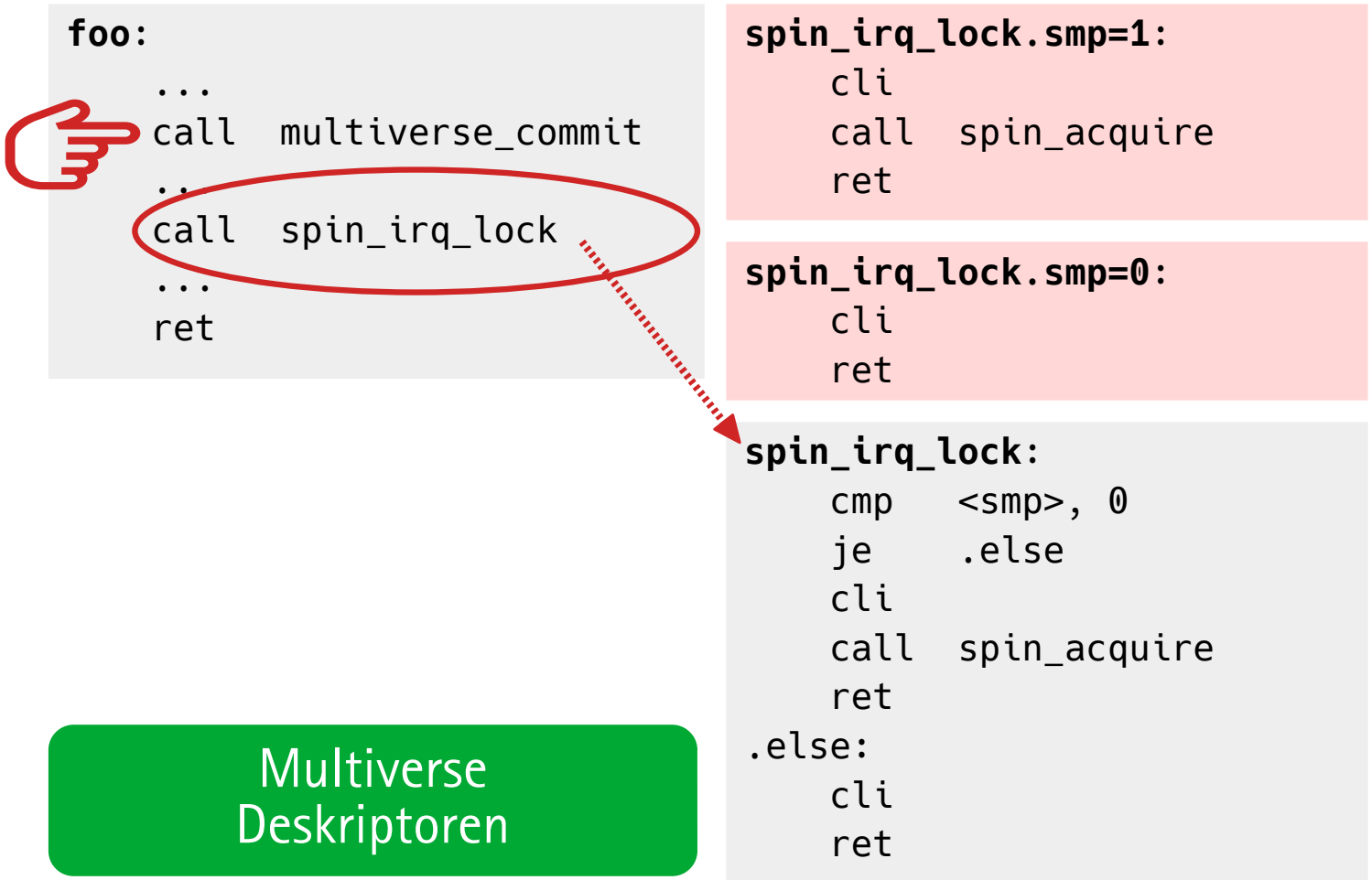
```
spin_irq_lock.smp=0:
    cli
    ret
```

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

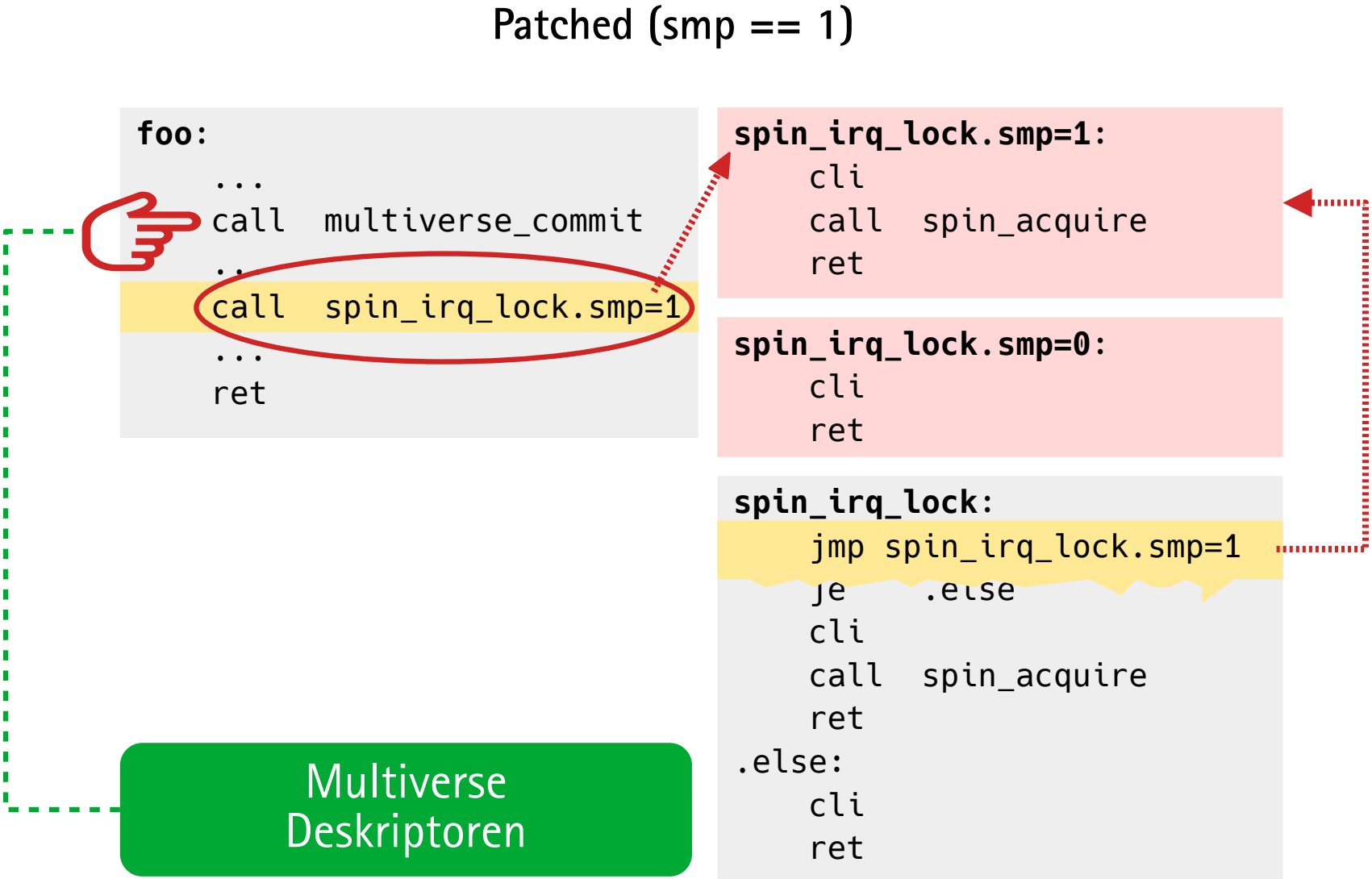
Multiverse Deskriptoren

Bindung: Run-Time Code Patching

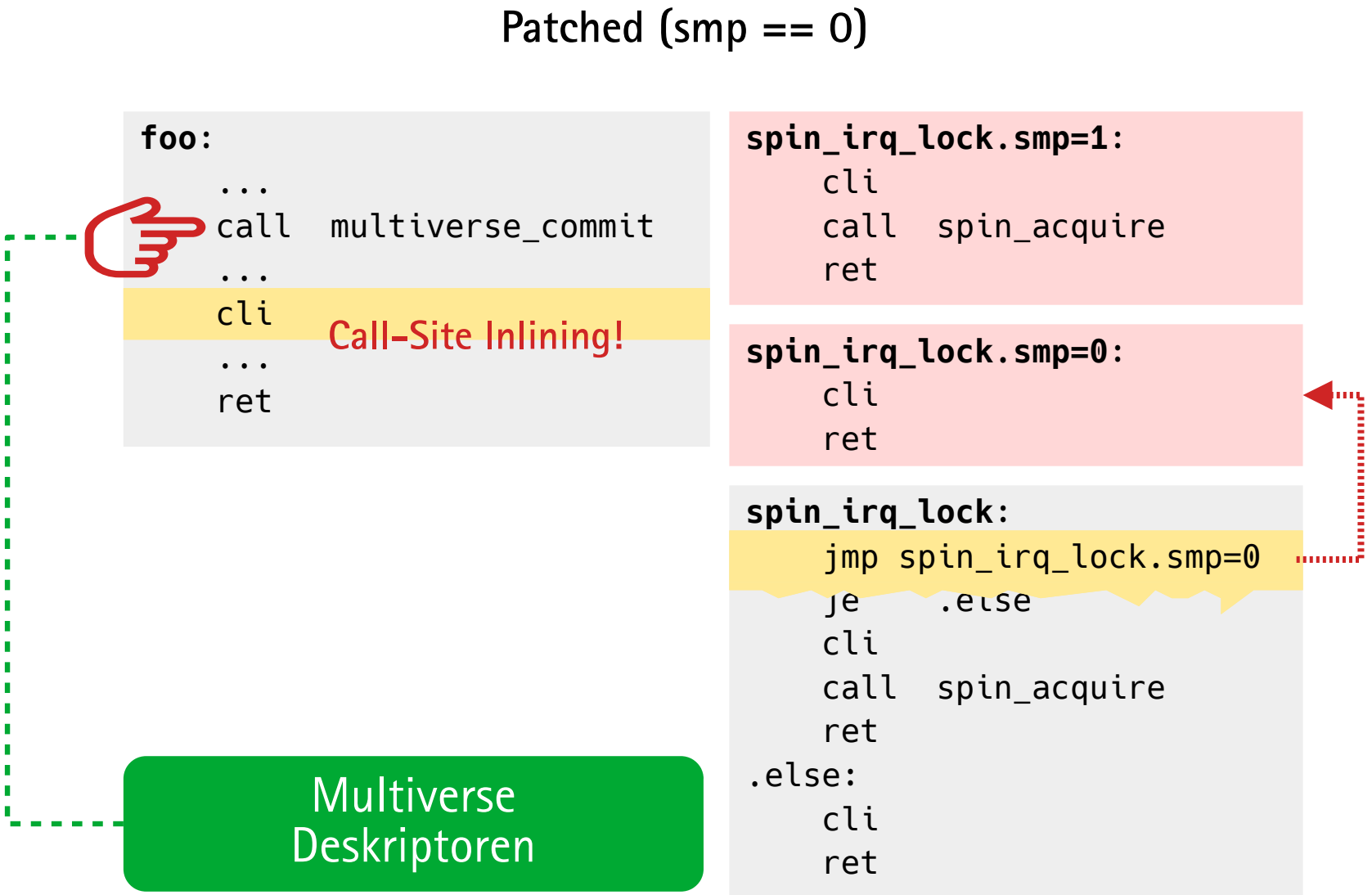
Initial geladenes Code Segment



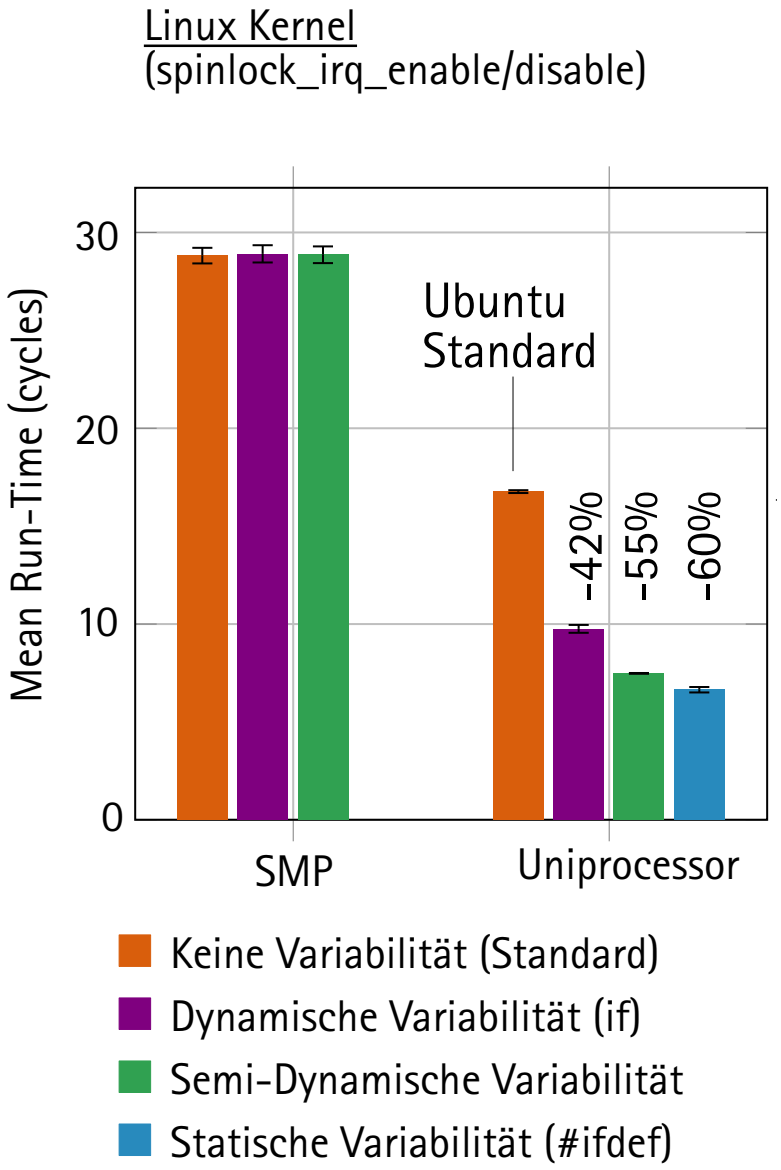
Bindung: Run-Time Code Patching



Bindung: Run-Time Code Patching

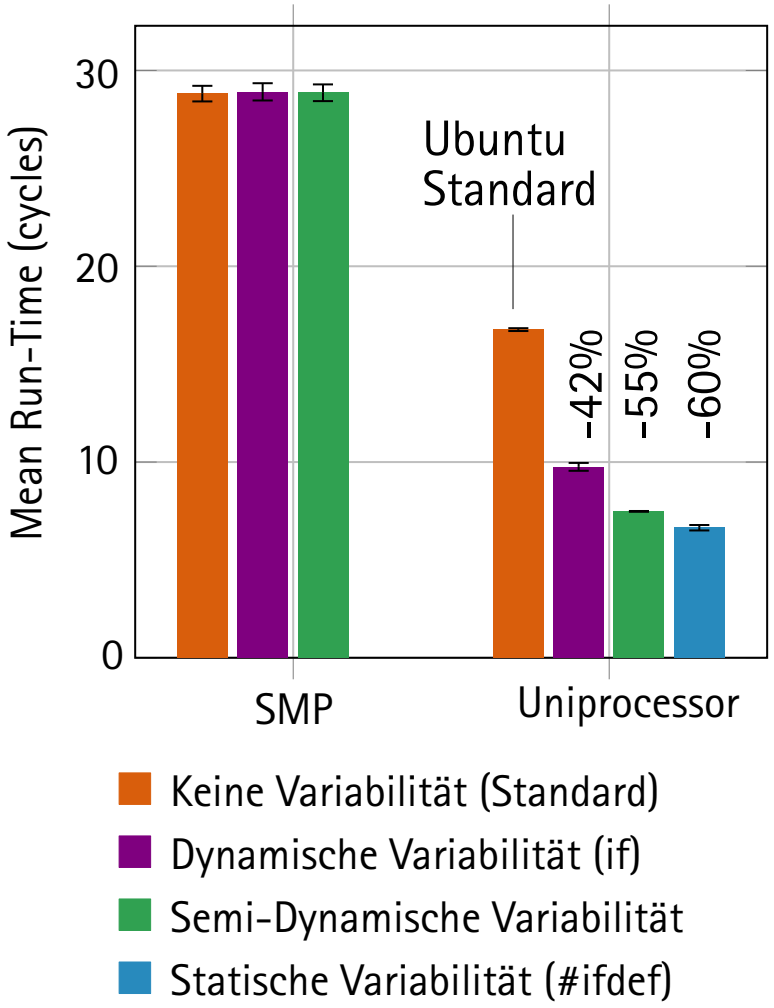


Semi-Dynamic Locks in Kernel/Userspace

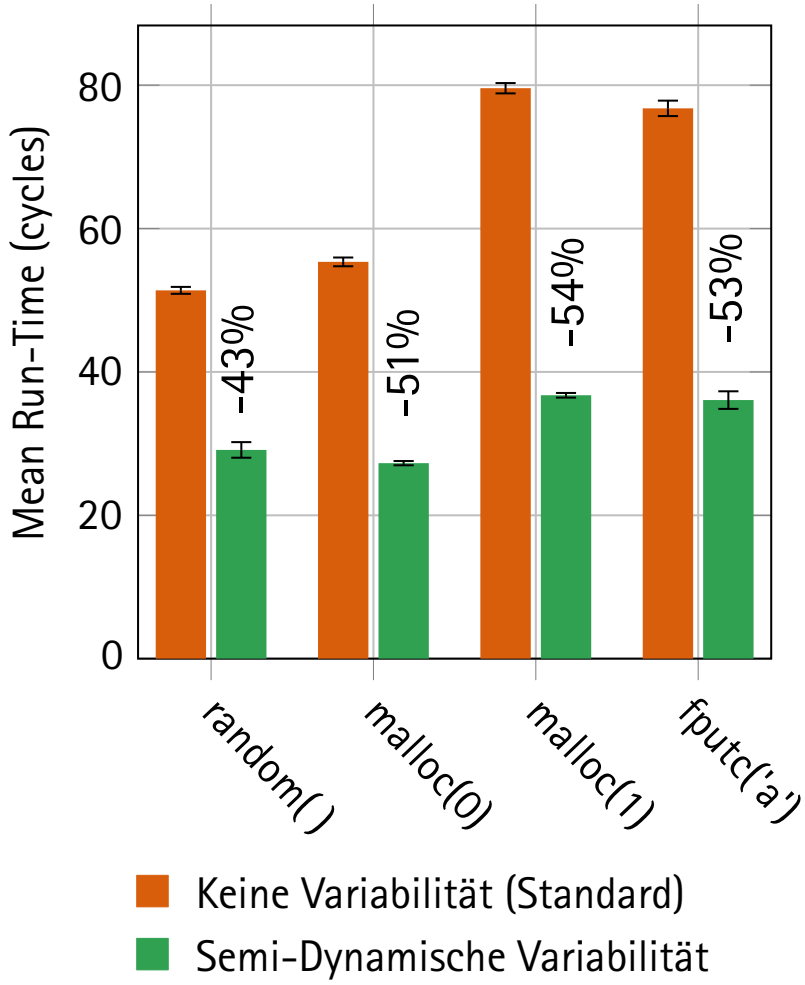


Semi-Dynamic Locks in Kernel/Userspace

Linux Kernel
(spinlock_irq_enable/disable)

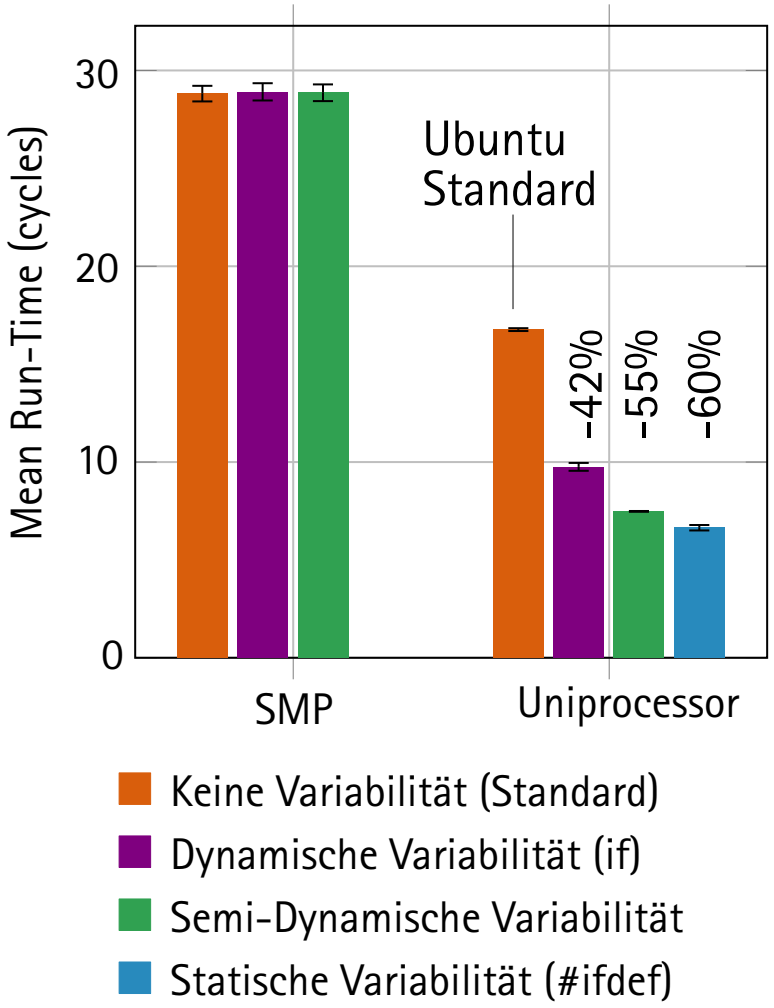


Musl C Bibliothek
(Single Threaded Modus)

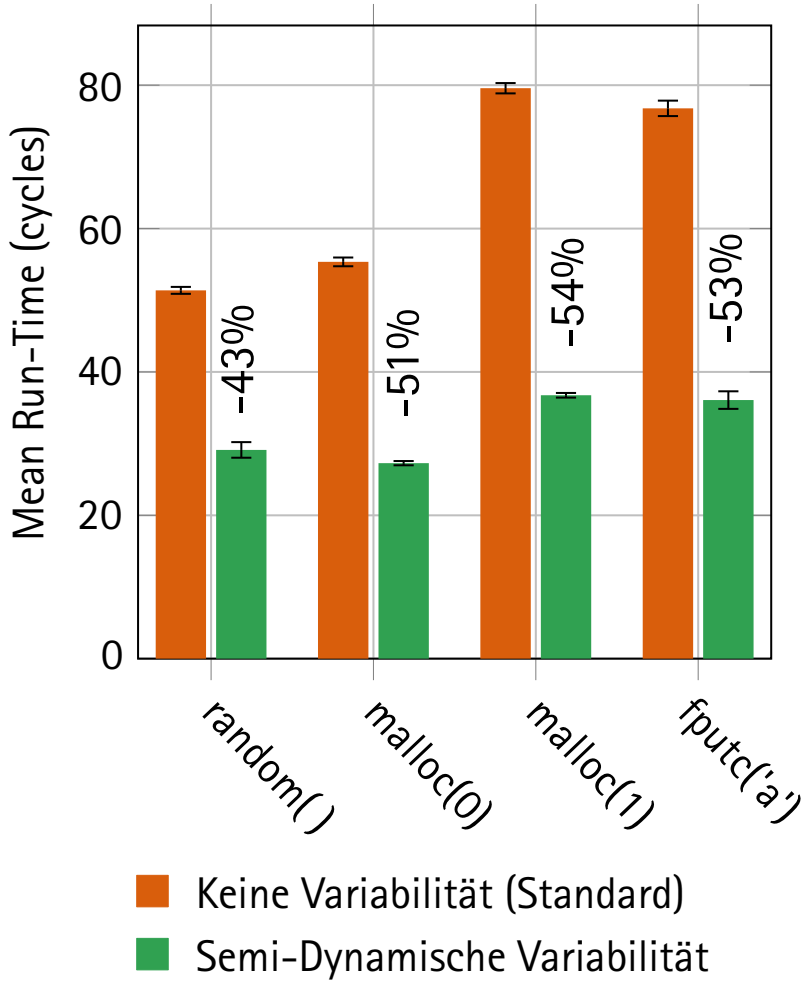


Semi-Dynamic Locks in Kernel/Userspace

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



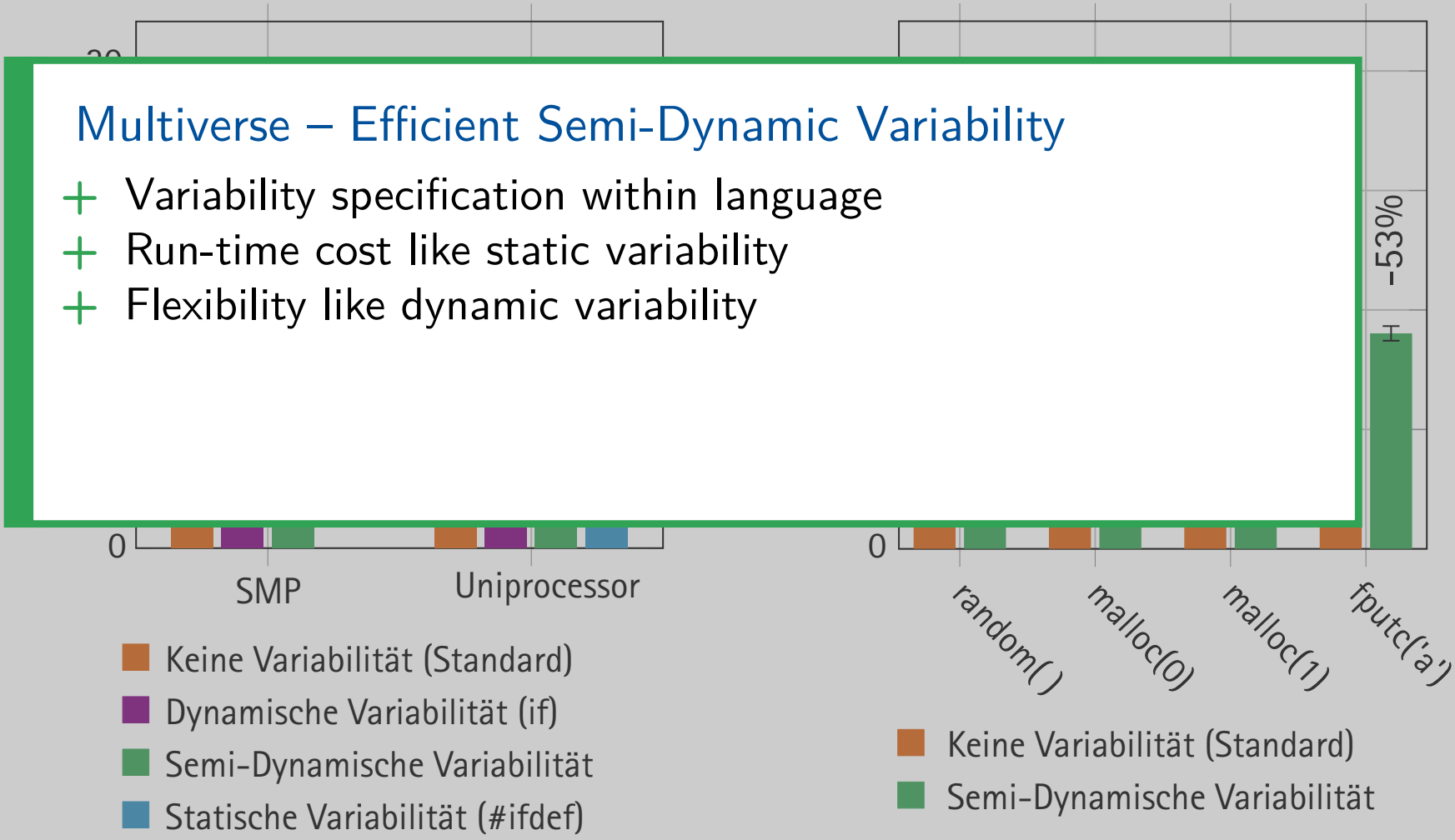
Musl C Bibliothek
(Single Threaded Modus)



Semi-Dynamic Locks in Kernel/Userspace

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

Musl C Bibliothek
(Single Threaded Modus)



Multiverse – Efficient Semi-Dynamic Variability

+ Variability specification within language
+ Run-time cost like static variability
+ Flexibility like dynamic variability

Semi-Dynamic Locks in Kernel/Userspace

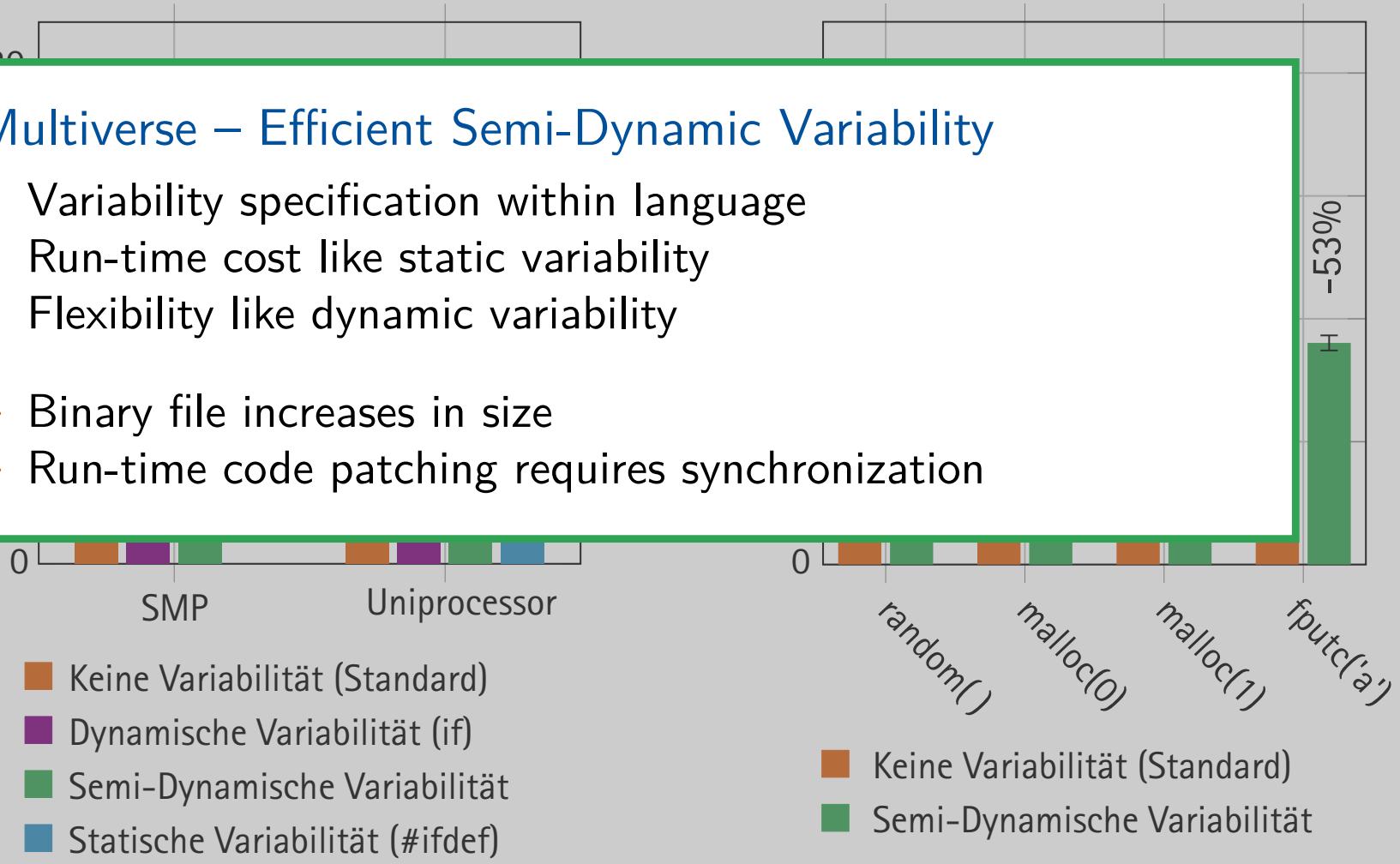
Lock Elision in Kernel 4.16:
→ 1161 spin-lock call sites
→ +40 KiB size (zipped total: 10 MiB)

Musl C Bibliothek
(Single Threaded Modus)

Multiverse – Efficient Semi-Dynamic Variability

+ Variability specification within language
+ Run-time cost like static variability
+ Flexibility like dynamic variability

– Binary file increases in size
– Run-time code patching requires synchronization



Tutorial: Virtual Machine and SSH Host

TUHH
Technische Universität Hamburg

- Example Source Code
  - `git clone https://github.com/luhsra/multiverse-atlas-demo.git`
  - `./init` => Clone dependent Libraries
  - You need a mmview-enabled kernel for example 02/03
- Virtual Machine Image
  - Download:
    `https://lab.sra.uni-hannover.de/Publications/2021/netsys-tutorial/`
  - Start: `qemu-system-x86_64 -m 4G -enable-kvm -hda hda.qcow2 -smp 4`
  - User/Passwort: `root/root`, `user/user`
- Running Virtual Machine (temporary)
  - `ssh user@lab.sra.uni-hannover.de -p 2250`
  - Password: `FIXME`
  - Machine will be available only for the time of the tutorial.

Tutorial: 01-multiverse

TUHH
Technische Universität Hamburg

Goal

Demonstrate Semi-Dynamic Lock-Elision with Multiverse

- Static Creation with `__attribute__((multiverse))`
  - Multiverse Variables (MV) are „features“. Default Range: {0, 1}
  - Attributed functions are multiversed at MV usages.
  - Cross-Product of MV ranges.
- Binary File: `01-multiverse`
  - Regular functions: `lock()`, `unlock()`
  - Use `objdump -d 01-multiverse` to inspect assembler
  - Search for `lock.multiverse.smp_0`
- Multiverse Metadata

```
fn: lock 0x55bd78dc92e0, 2 variants
  mvfn: 0x55bd78dc9410 (vars 1)
    assign: smp in [0, 0]
  mvfn: 0x55bd78dc9420 (vars 1)
    assign: smp in [1, 1]
```

Tutorial: 01-multiverse


```
Without Multiverse (smp=0)
 work() -> 83.376870 ms
 work() -> 70.550493 ms
 work() -> 70.670475 ms
 work() -> 70.578593 ms
 work() -> 73.273835 ms
Average (n=5): 73.690053

Without Multiverse (smp=1)
 work() -> 511.101093 ms
 work() -> 511.533894 ms
 work() -> 512.103367 ms
 work() -> 519.490820 ms
 work() -> 511.268393 ms
Average (n=5): 513.099513
```

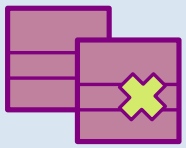
```
With Multiverse (smp=0)
 work() -> 18.856244 ms
 work() -> 18.820774 ms
 work() -> 18.878794 ms
 work() -> 18.757183 ms
 work() -> 18.795324 ms
Average (n=5): 18.821664


With Multiverse (smp=1)
 work() -> 509.799176 ms
 work() -> 509.639876 ms
 work() -> 518.221404 ms
 work() -> 511.009342 ms
 work() -> 510.144087 ms
Average (n=5): 511.762777
```

Agenda

Multiverse
Efficient Semi-Dynamic Variability

EuroSys'19

- Preparing variants at compile time
- Efficient binding by run-time code patching



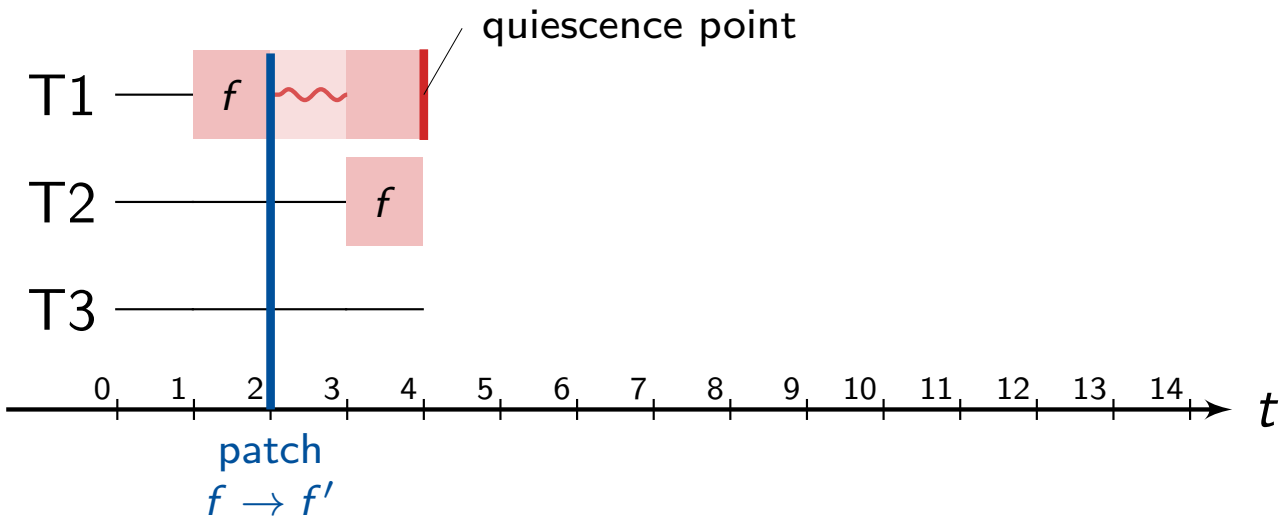
Wait-free code patching
Run-time code patching without interruption

OSDI'20

- Preparing changes in a separate address space
- Incremental migration of threads

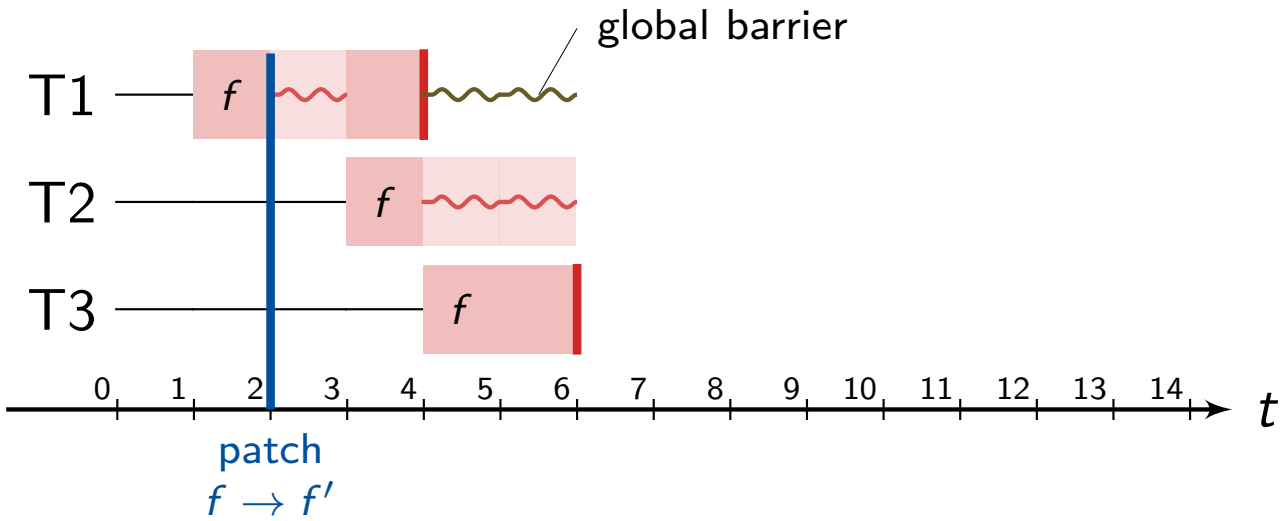
Global Quiescence

Run-Time Code Patching requires Quiescence!



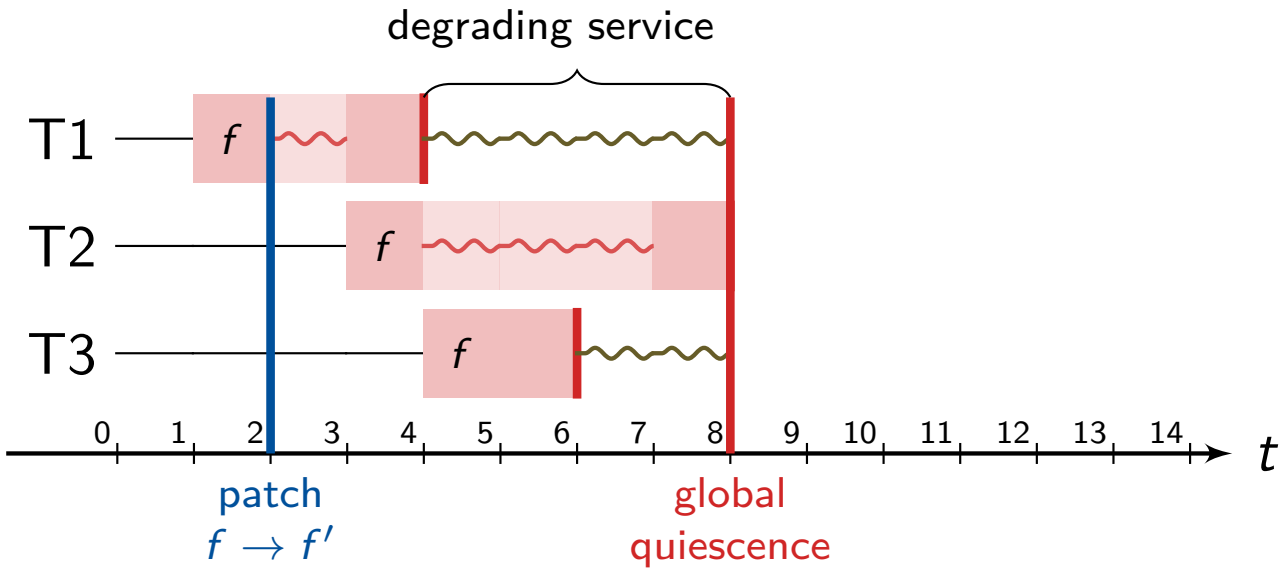
Global Quiescence

Run-Time Code Patching requires Quiescence!



Global Quiescence

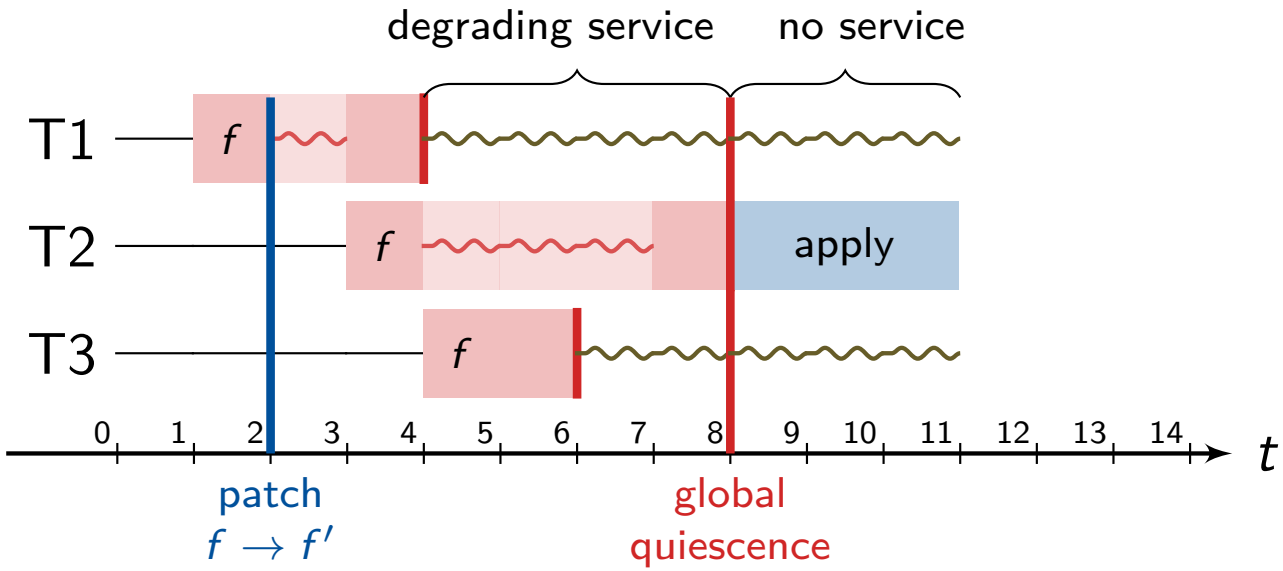
Run-Time Code Patching requires Quiescence!



- Global barrier bears risk of dead-lock
- Impact on service quality

Global Quiescence

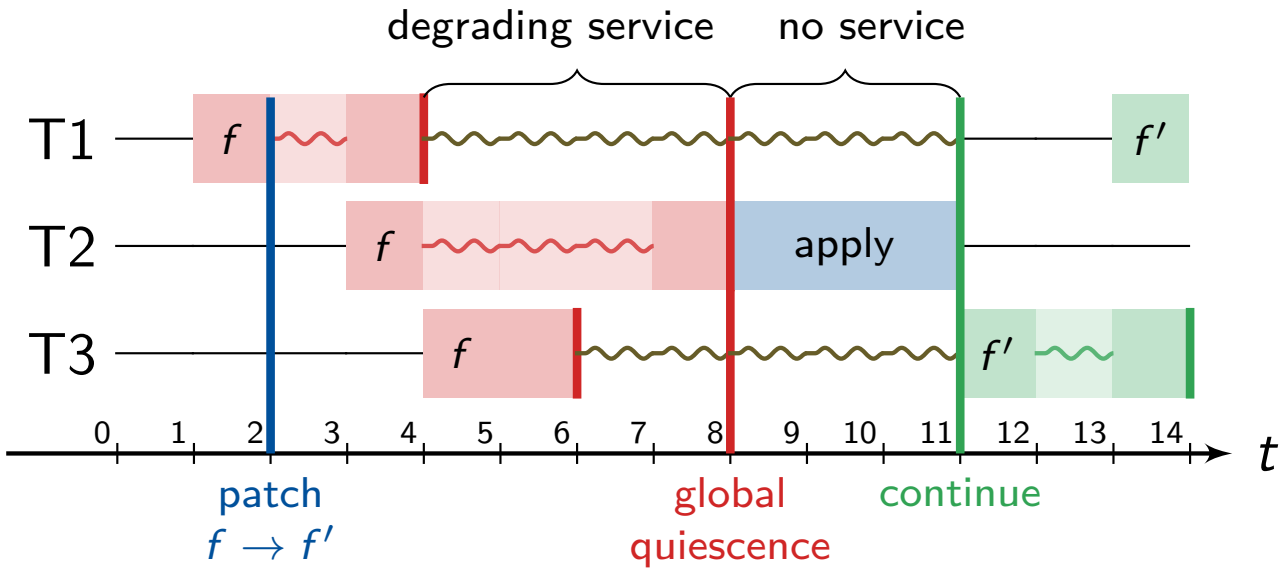
Run-Time Code Patching requires Quiescence!



– Global barrier bears risk of dead-lock
– Impact on service quality
\+ Strong consistency through global quiescence

Global Quiescence

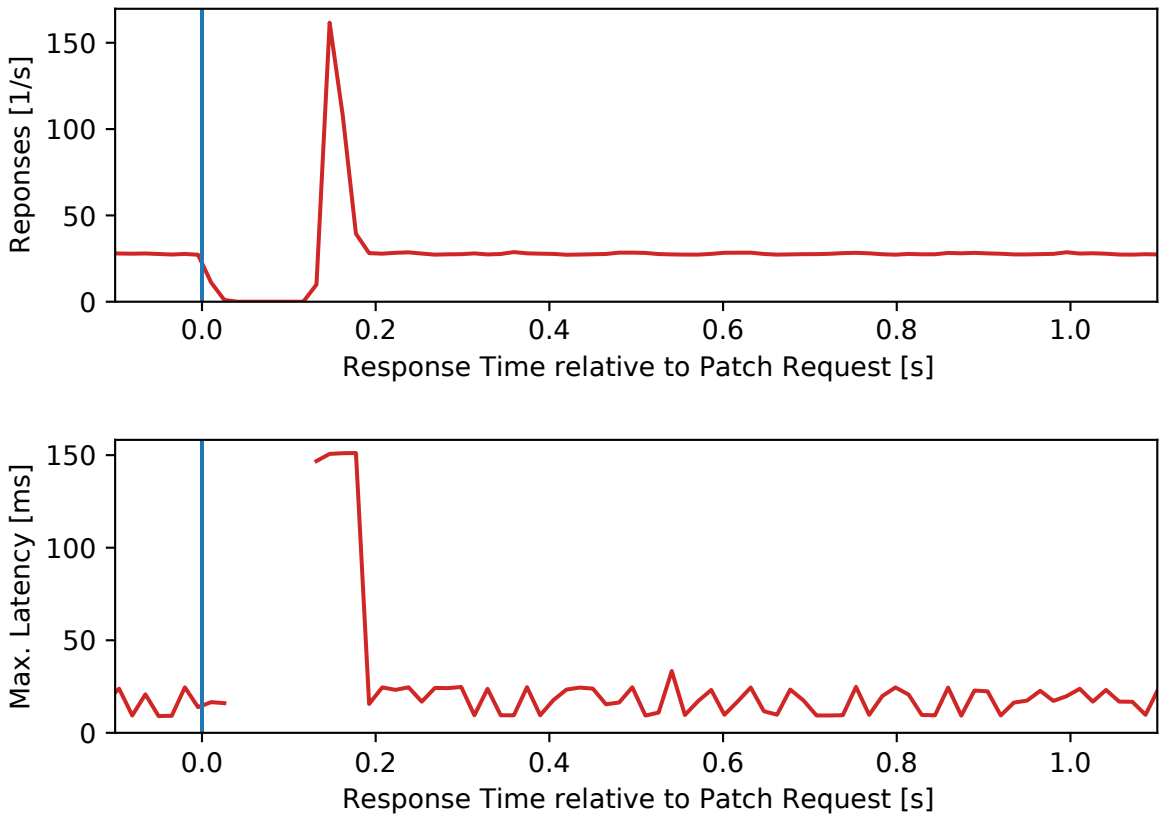
Run-Time Code Patching requires Quiescence!



- Global barrier bears risk of dead-lock
- Impact on service quality
+ Strong consistency through global quiescence

Global Quiescence

OpenLDAP

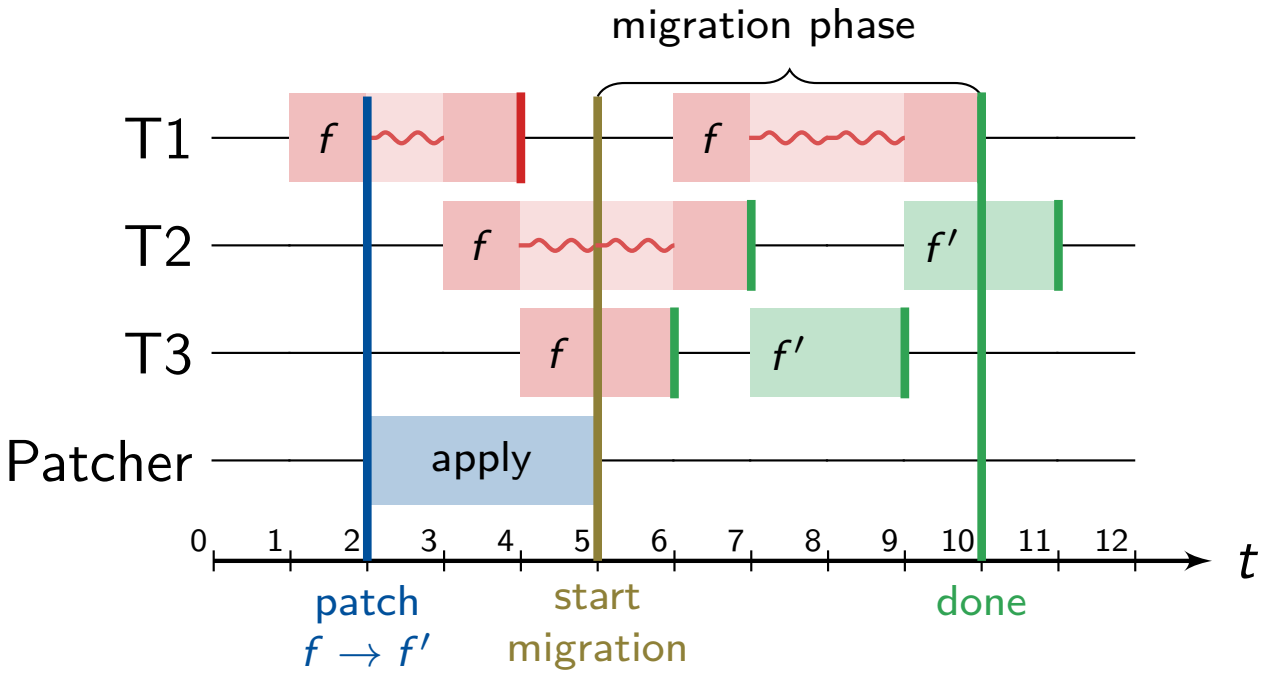


– Glo

– Imp

\+ Strong consistency through global quiescence

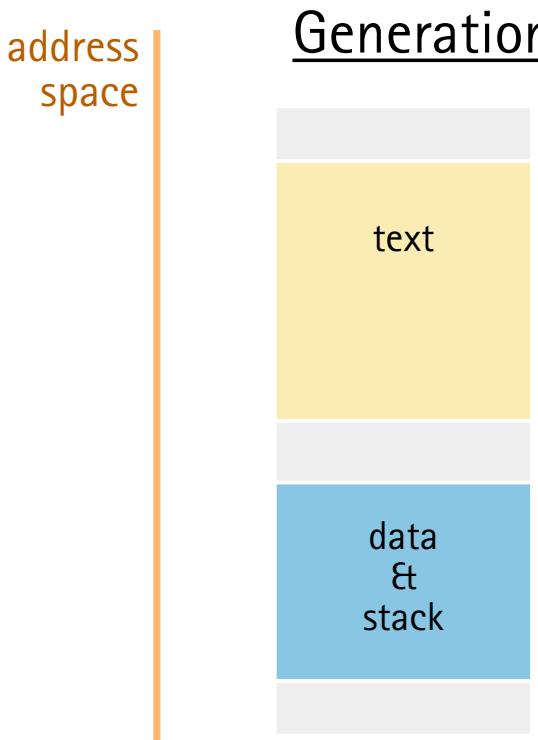
Global Quiescence → Local Quiescence



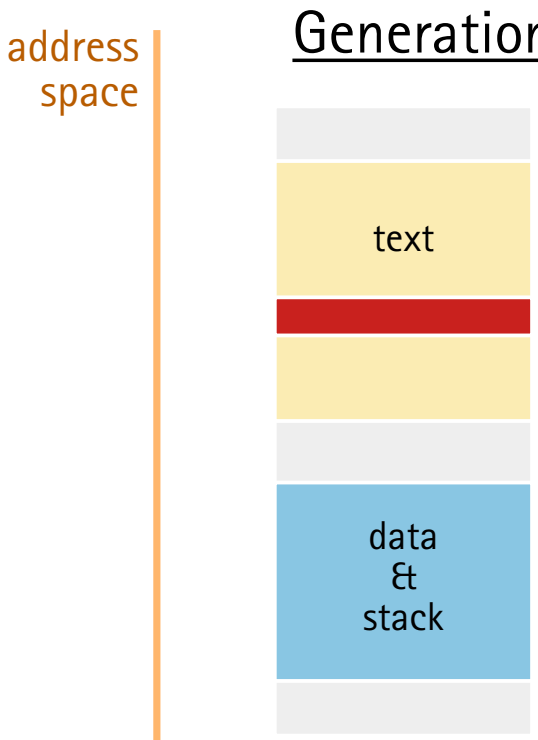
Wait-Free Code Patching — *Incremental Application of Patch*

- Prepare the modified code segment
- Threads migrate themselves and without synchronization
- Old and new variants co-exist (for shorter or longer periods of time)

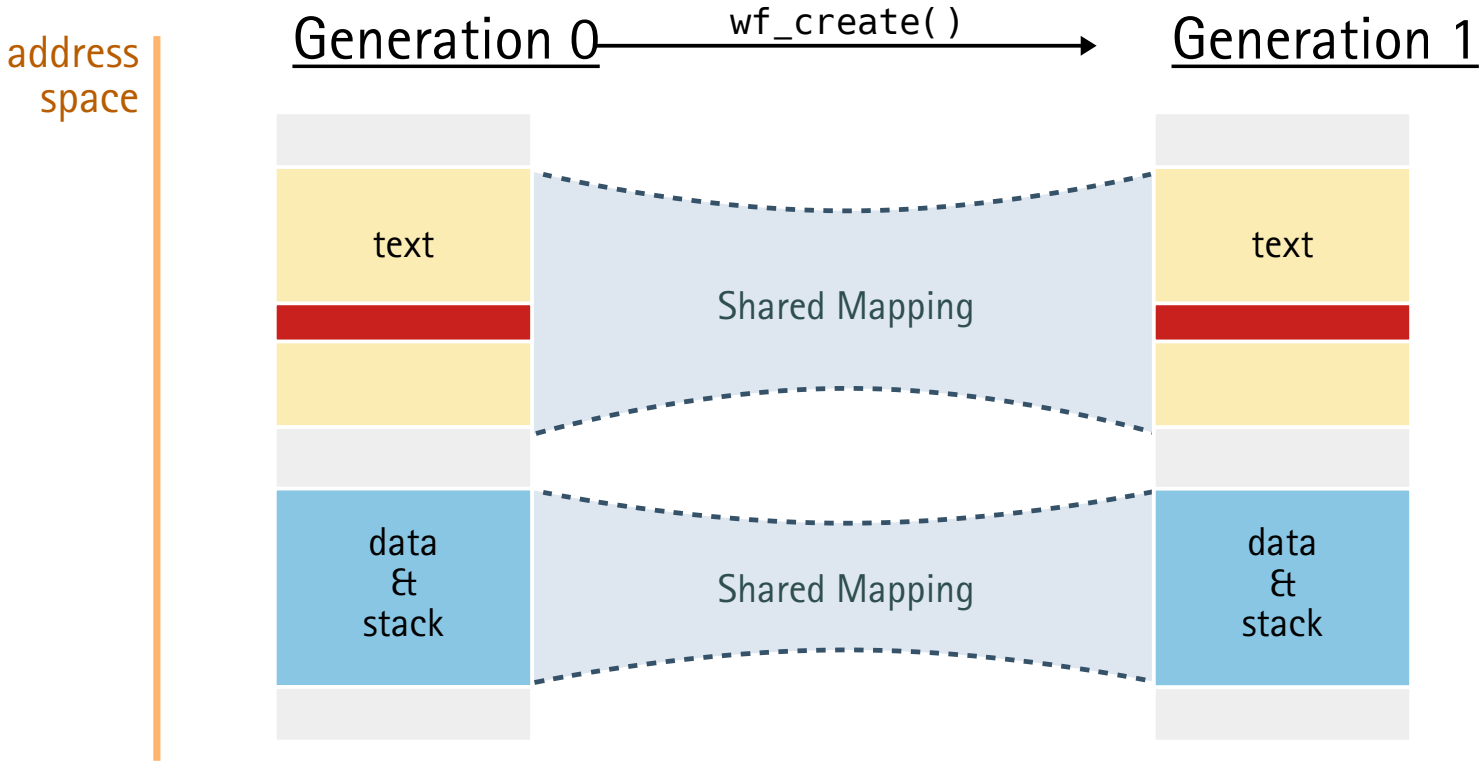
Address-Space Views



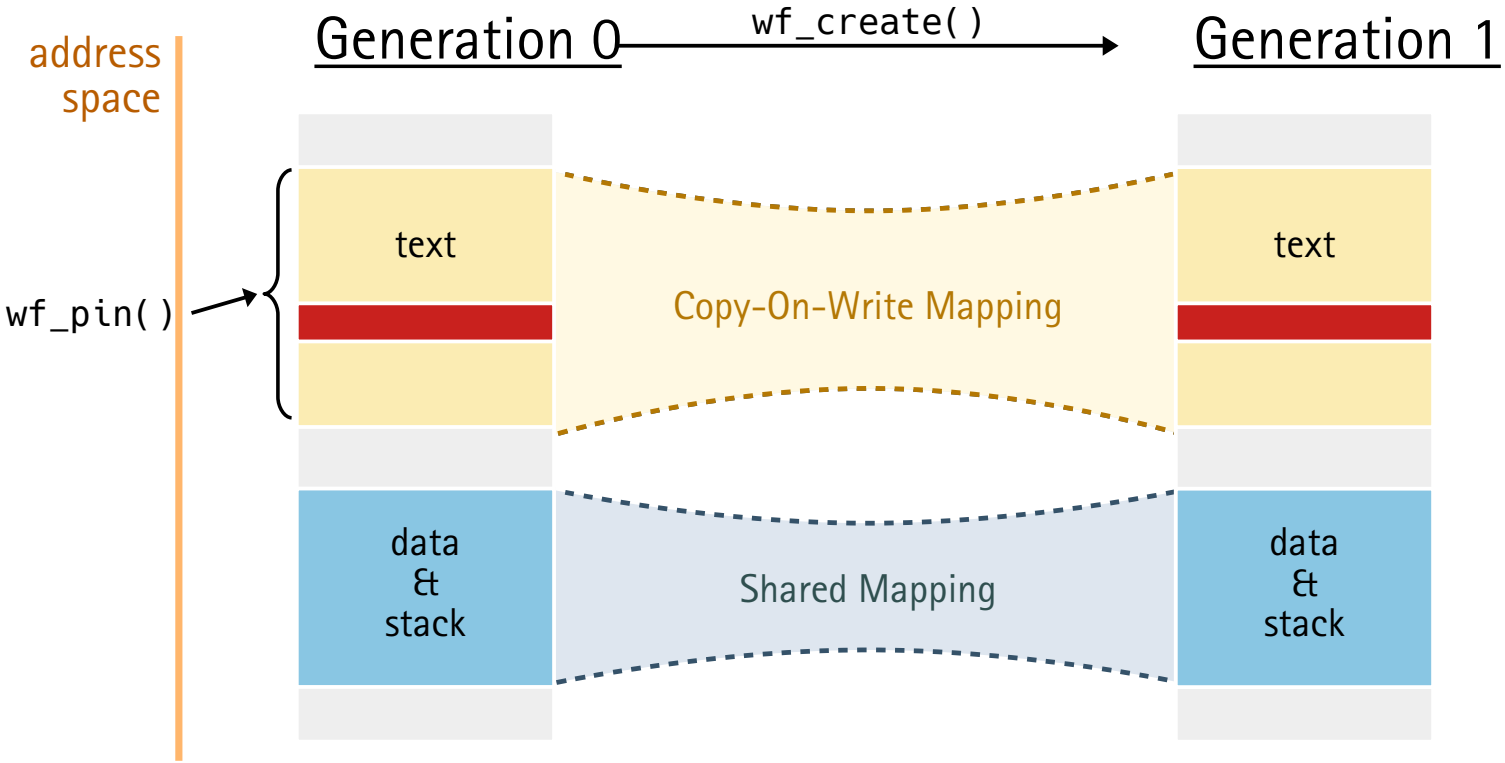
Address-Space Views



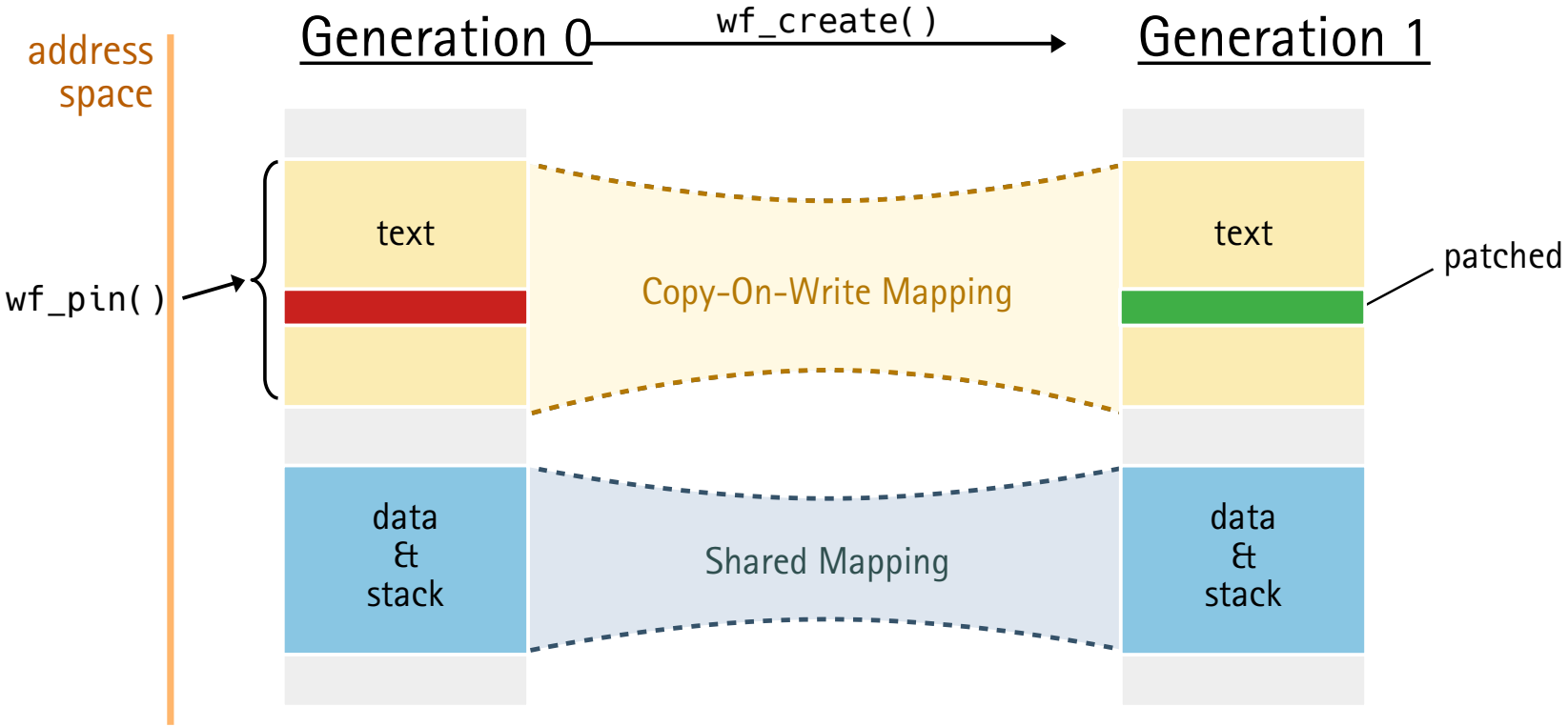
Address-Space Views



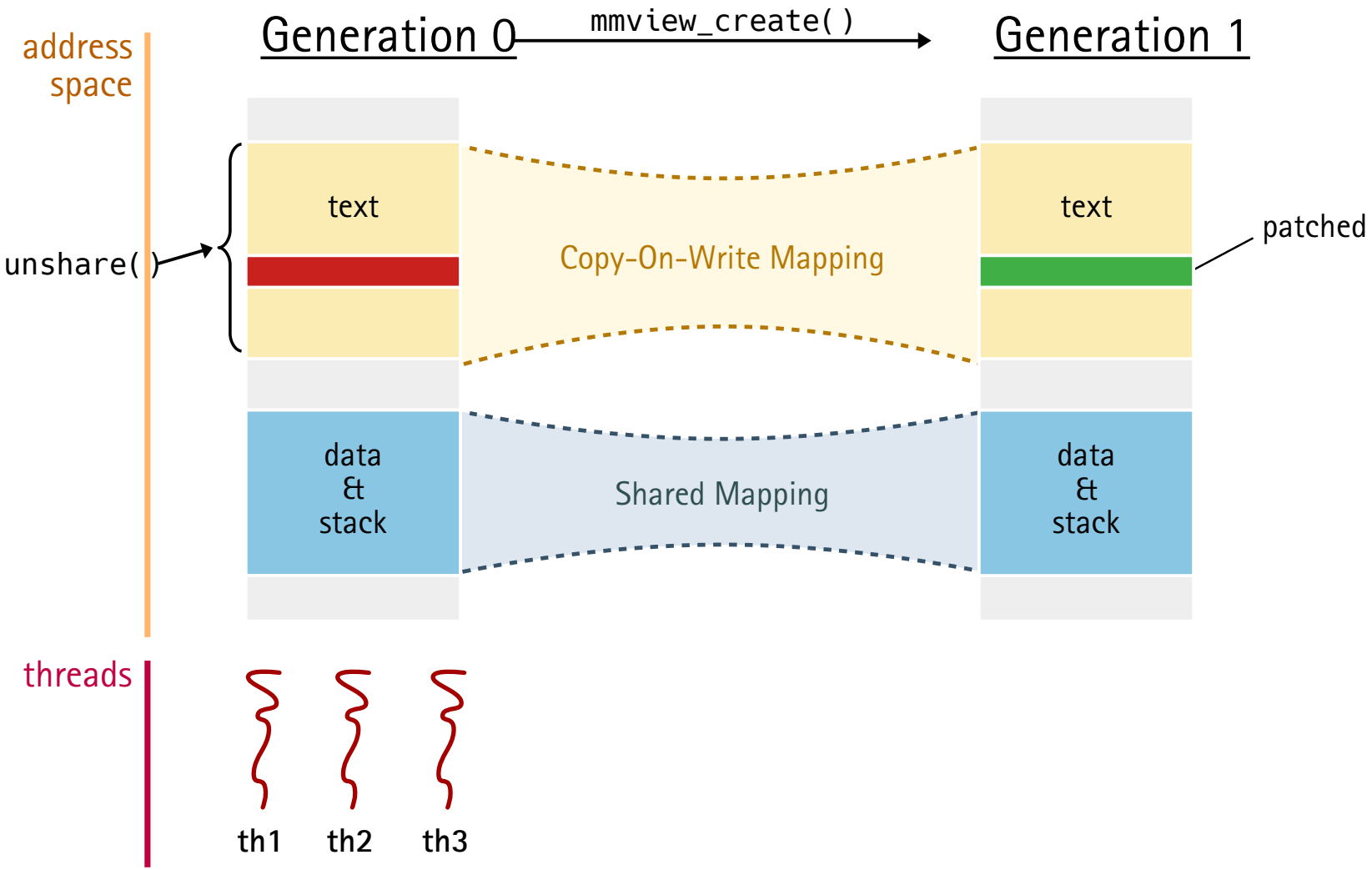
Address-Space Views



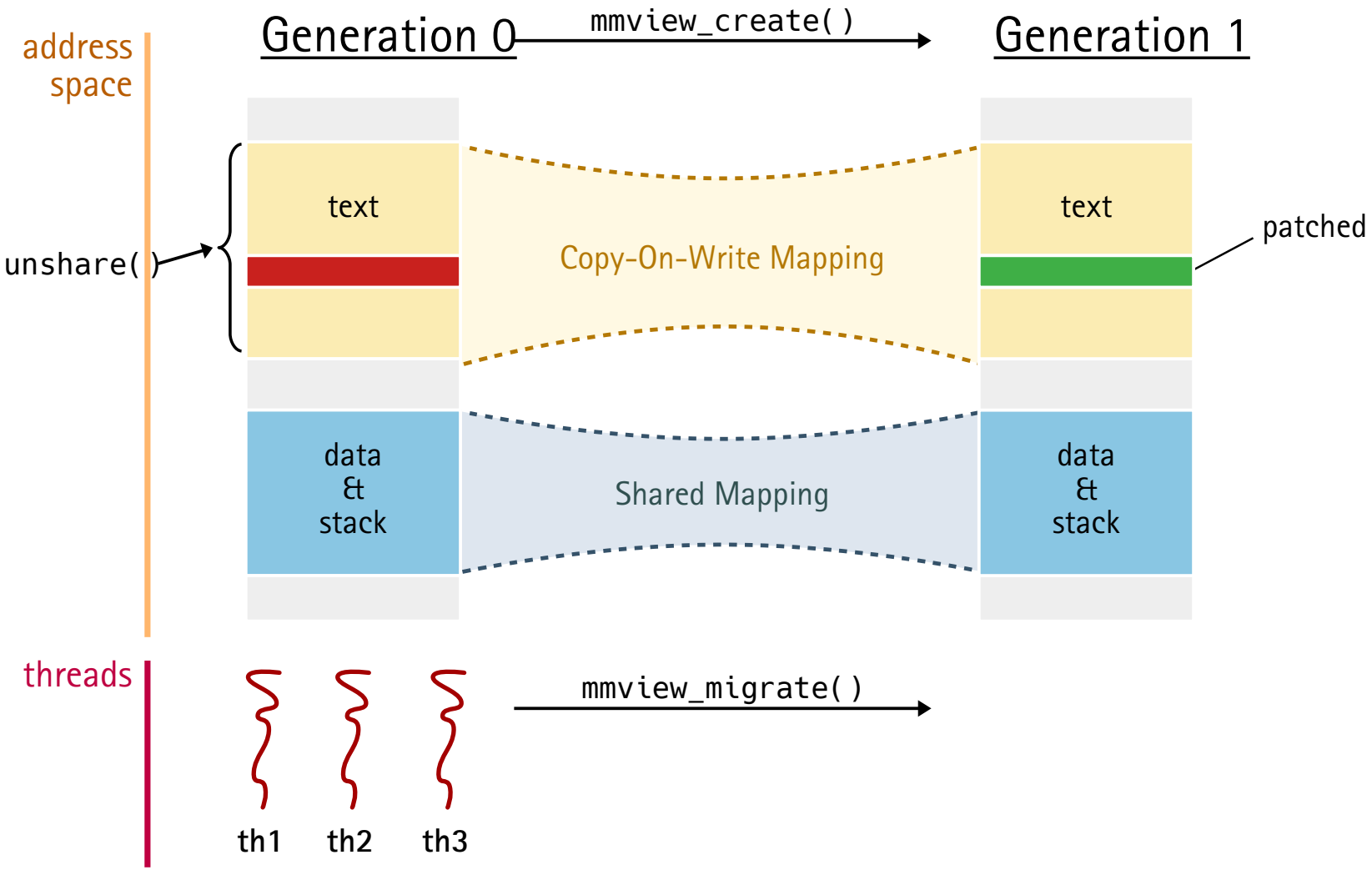
Address-Space Views



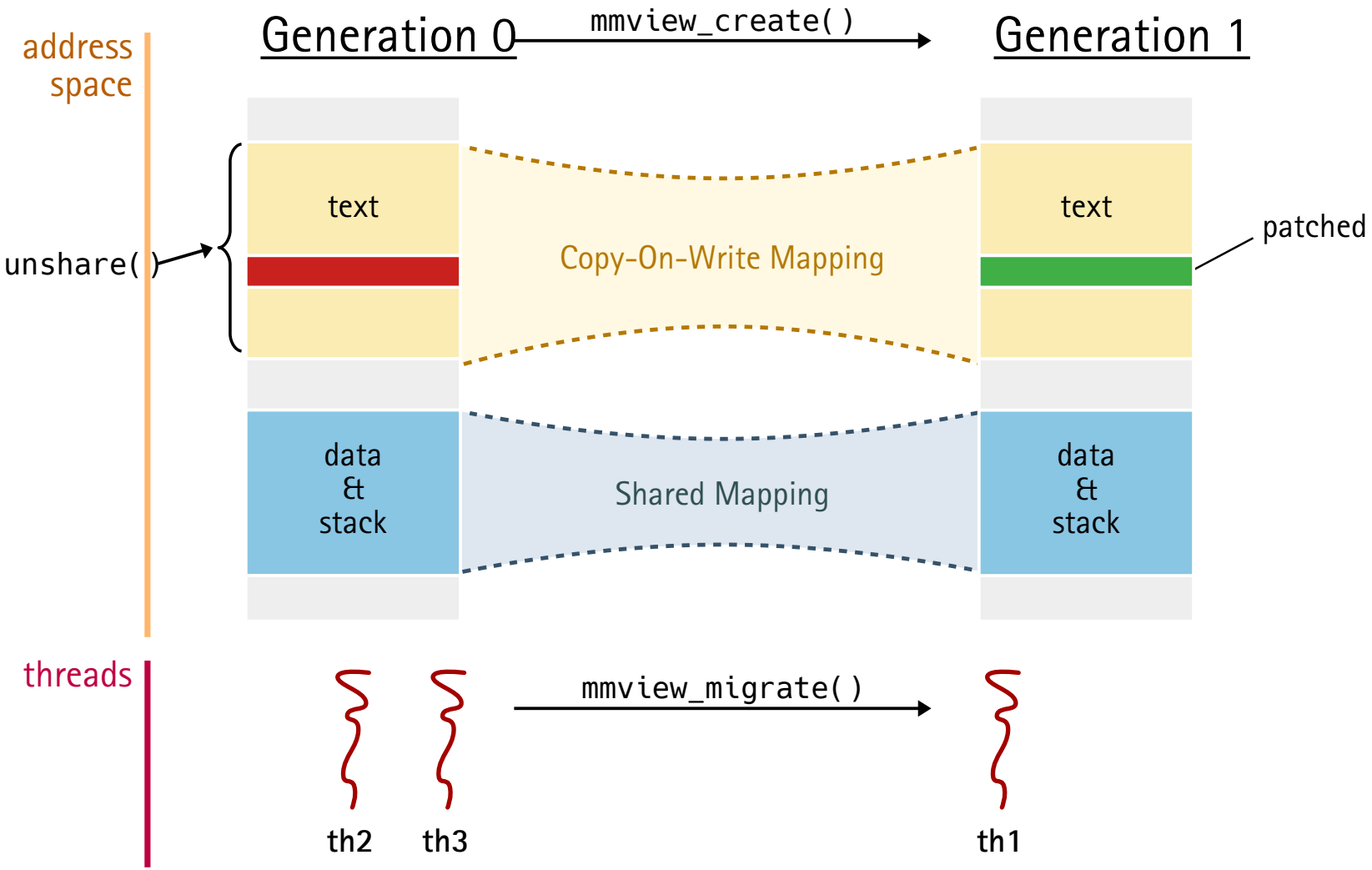
Address-Space Views



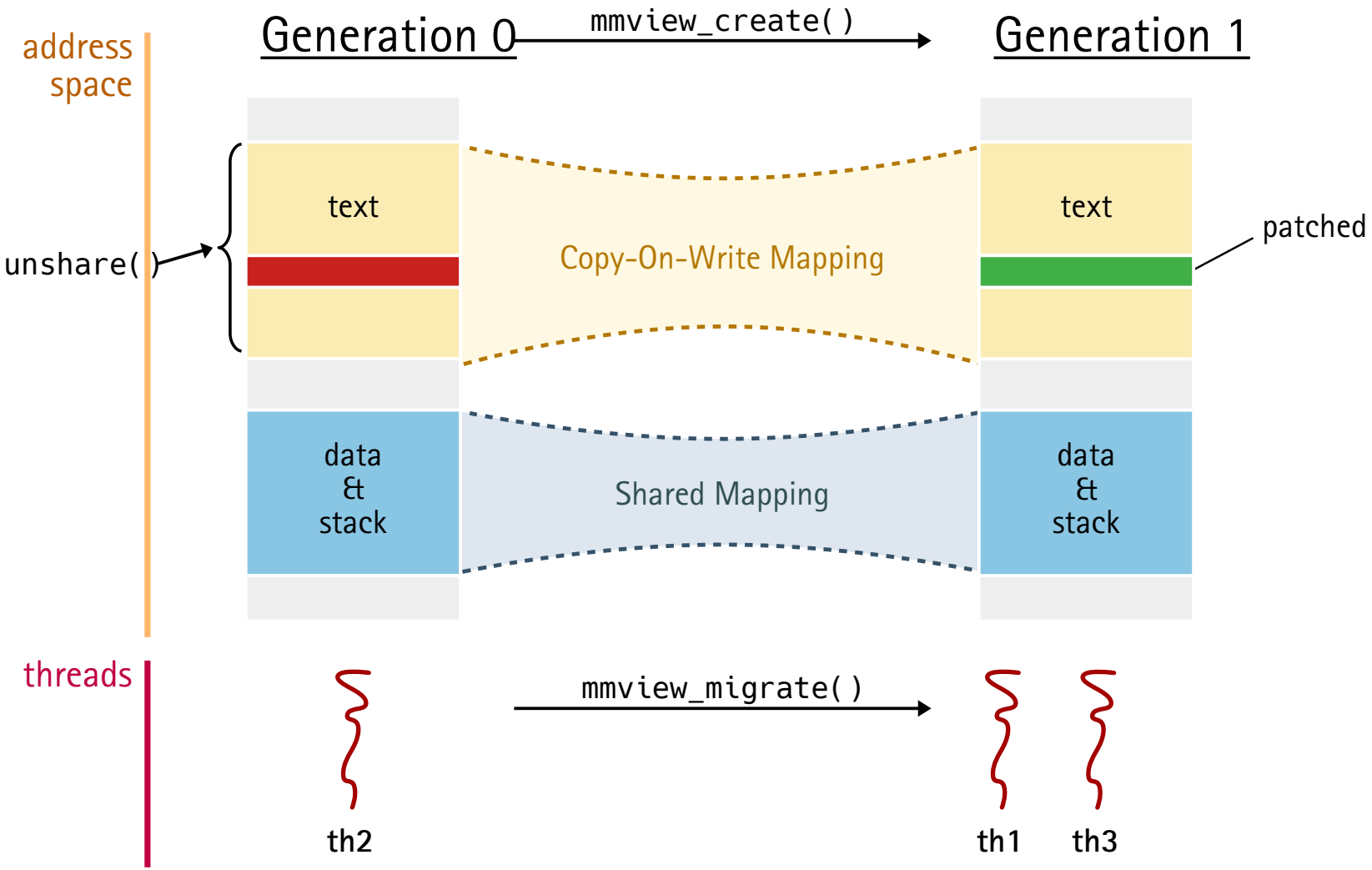
Address-Space Views



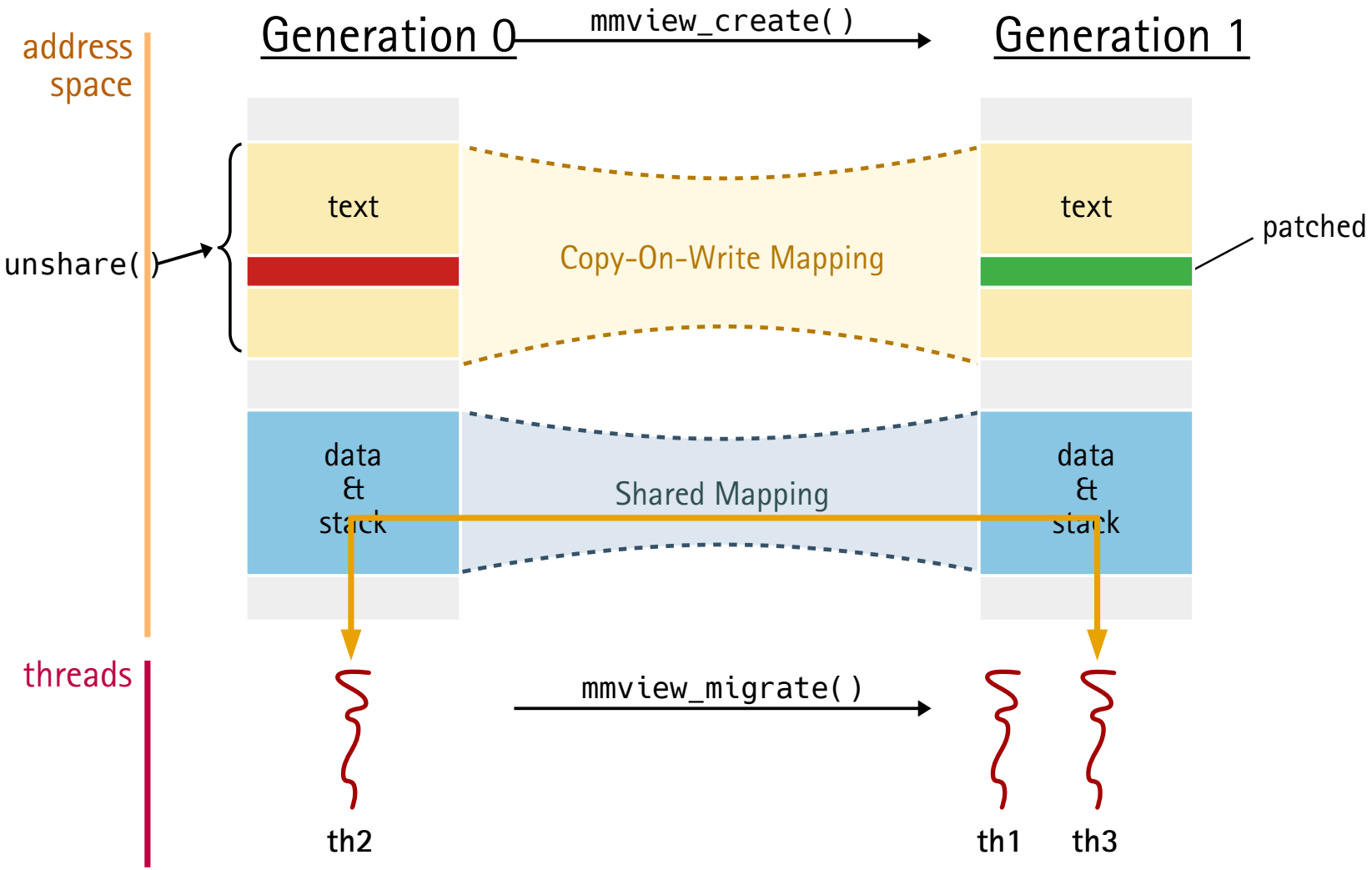
Address-Space Views



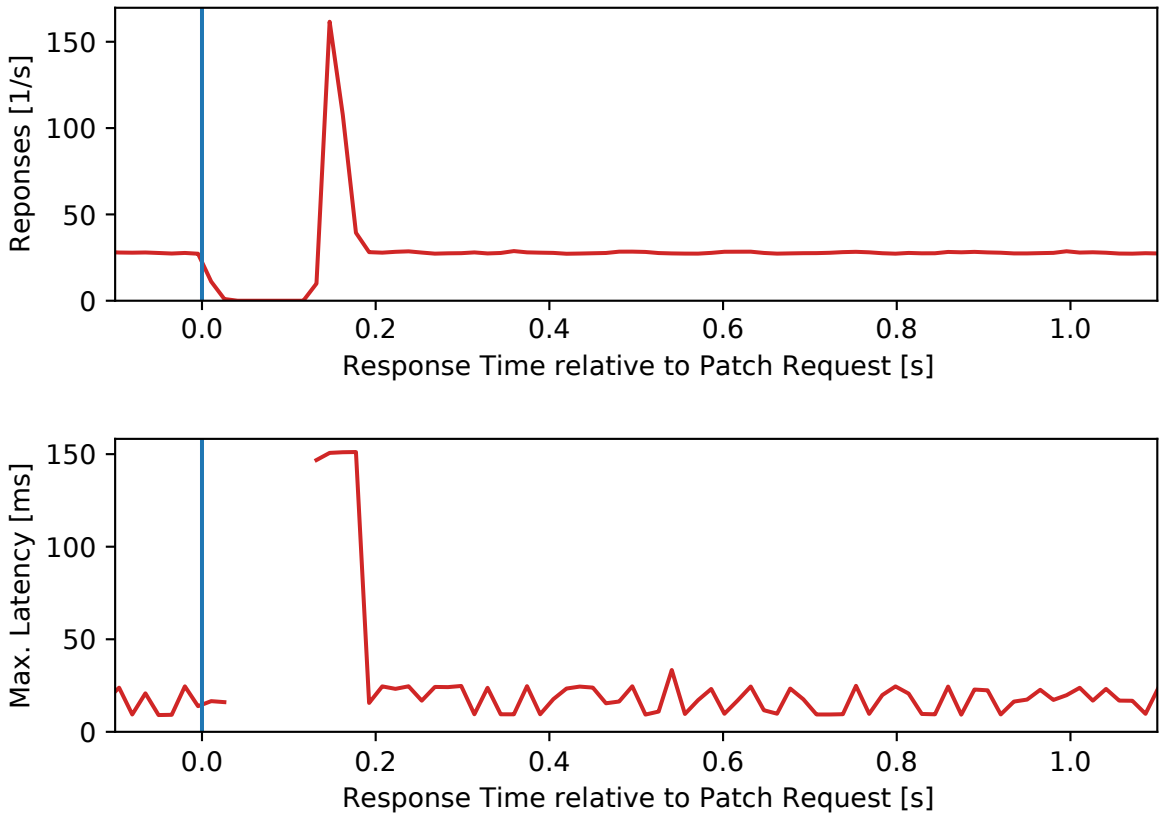
Address-Space Views



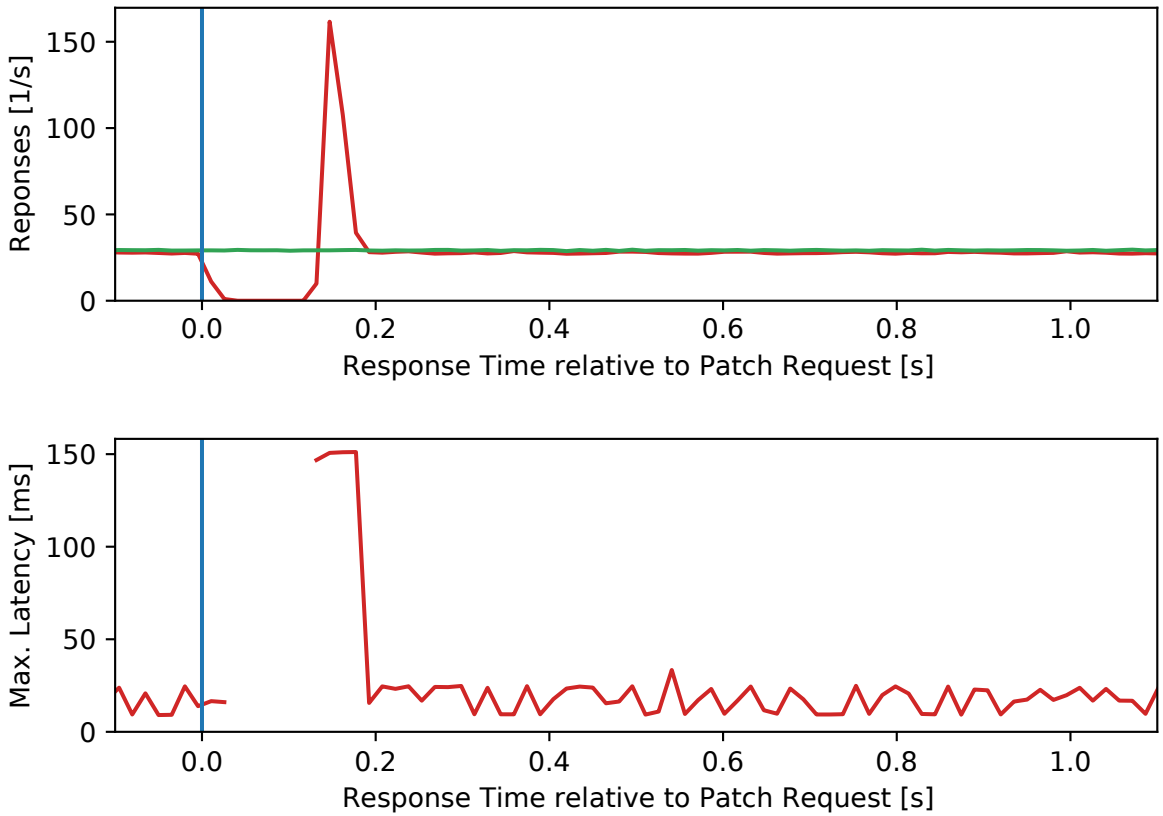
Address-Space Views



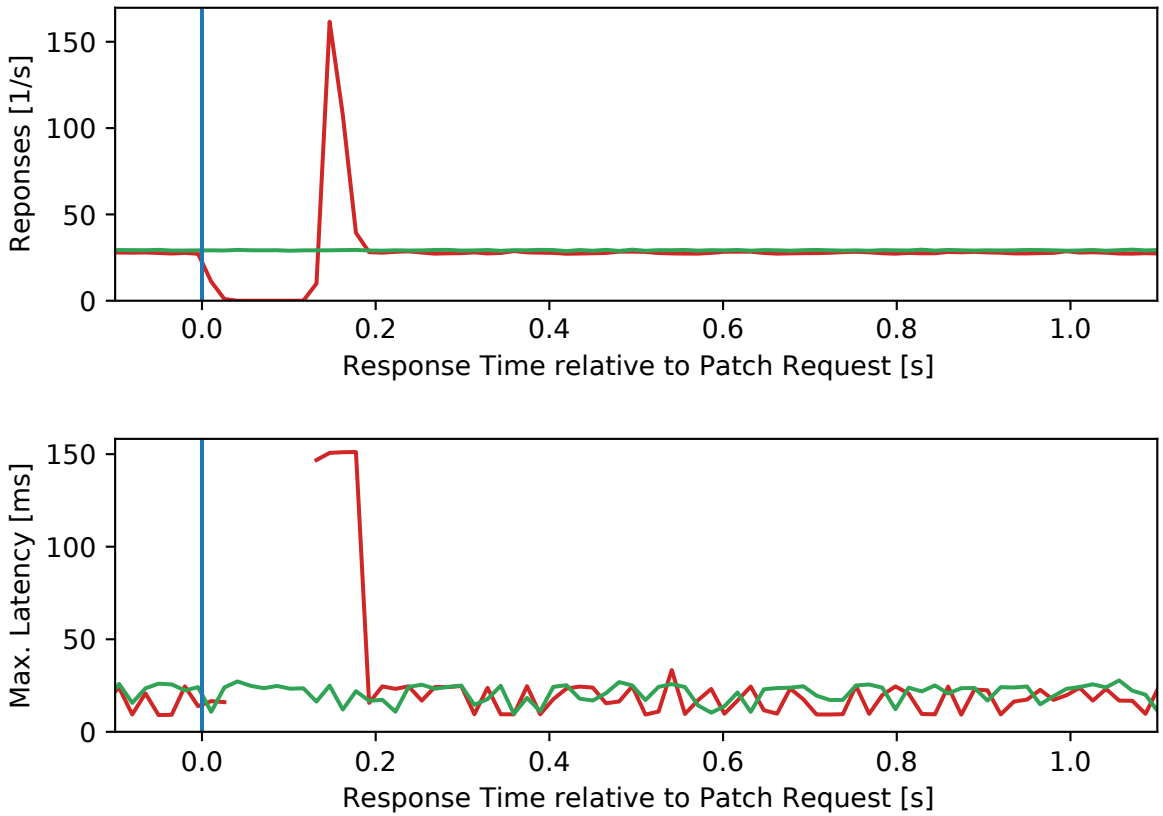
OpenLDAP


Reponses [1/s]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]
Max. Latency [ms]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]

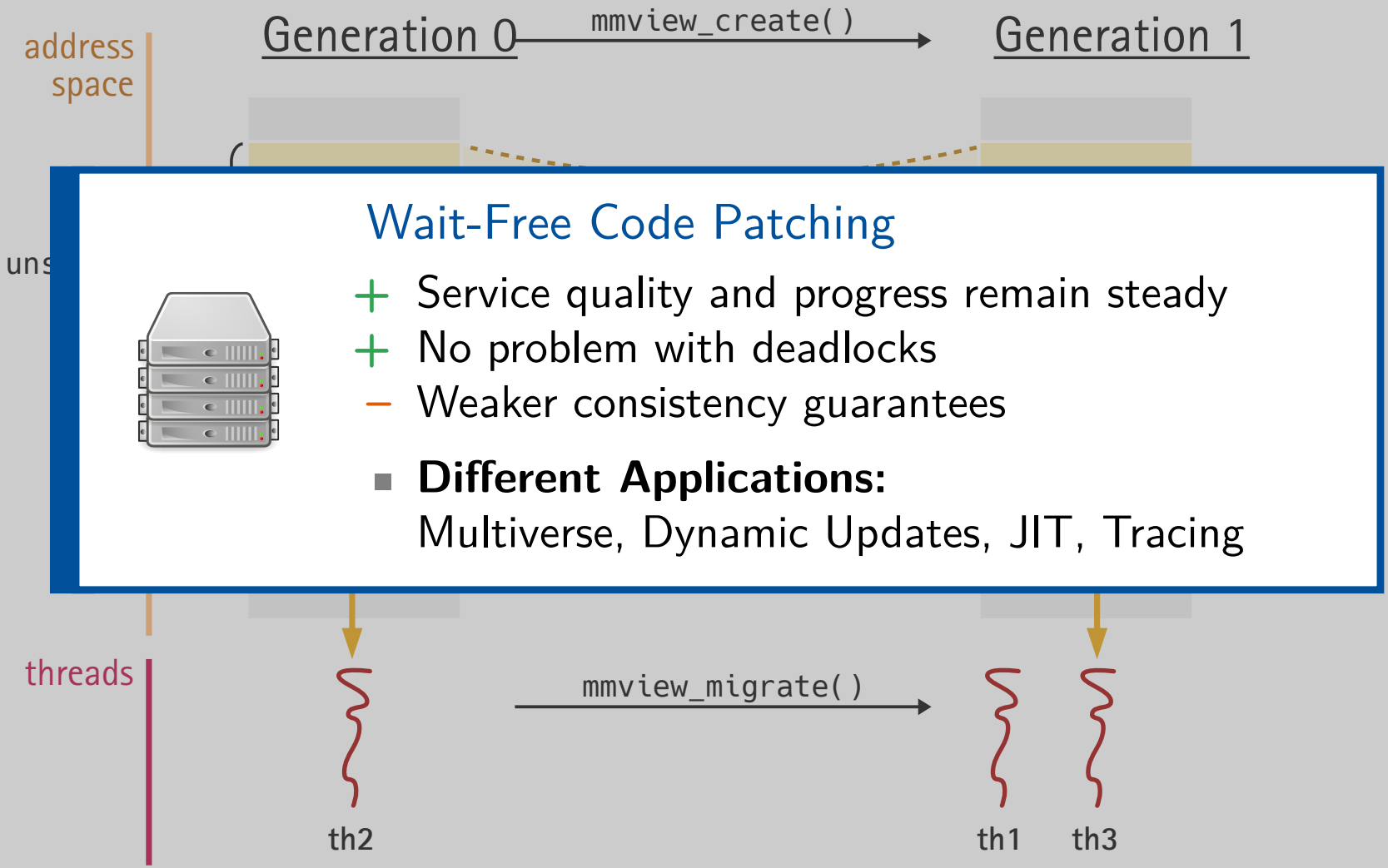
OpenLDAP


Reponses [1/s]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]
Max. Latency [ms]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]

OpenLDAP


Reponses [1/s]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]
Max. Latency [ms]
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
Response Time relative to Patch Request [s]

Response Time Latencies



Tutorial: 02-migrate

TUHH
Technische Universität Hamburg

Goal

Demonstrate Adress-Space View Migration with Multiverse

```
mmview_t view = mmview_create();
mmview_migrate(view);

multiversed_var = 1;
multiverse_commit();
desired_view = view;
```

- Create new address-space view and migrate the current thread.
- Commit current MV state into text segment.
- Warning: `multiverse_var` is shared!

```
if (mmview_current() != desired_view) {
    mmview_migrate(desired_mmview);
}
```

Each thread checks at its local-quiesence point, whether it is already in the correct view. If not
→ `mmview_migrate();`

Tutorial: 02-migrate


```
thread2  ->  view B
thread0  ->  view B
mmview_create() -> 1
mmview_migrate(1) -> 0
thread1  ->  view B
thread1 migrate: mmview_migrate(1) -> 0
thread2  ->  view B
thread2 migrate: mmview_migrate(1) -> 0
thread1  ->  view A
thread0  ->  view B
thread0 migrate: mmview_migrate(1) -> 0
thread1  ->  view A
thread0  ->  view A
thread2  ->  view A
thread1  ->  view A
thread0  ->  view A
thread1  ->  view A
thread1  ->  view A
```

Tutorial: 03-concurrent

TUHH
Technische Universität Hamburg

Goal

Demonstrate concurrent existence of multiversed address-space views.

- **Problem**: Profiling in multi-threaded processes is slow!
  - One (shared!) counter per profiling point.
  - Massive cache line bouncing between threads.
  - Your Choice: Racy counter updates OR run-time impact of atomic operations
- **Approach**: Let only one thread profile at once
  - Two Views with Multiverse: Profiling and Non-Profiling View
  - Let N threads (probabilistically) switch to the profiling view
  - Control parallelism in profiling with semaphore

Tutorial: 03-concurrent

- 4 threads calculating, no profiling

```
./03-concurrent 4 0
Avg. time for fib(30) = 0.956386 ms; -nan branches/call 400
```

- 4 threads calculating, 4 threads profiling

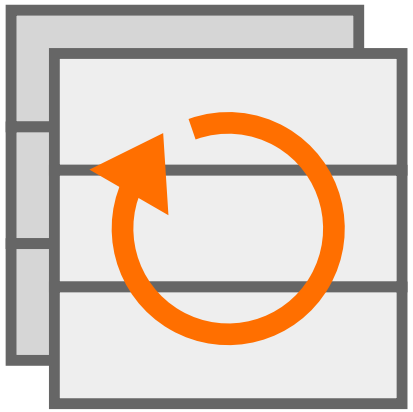
```
./03-concurrent 4 4
Avg. time for fib(30) = 106.137128 ms; 0.509816 branches/call 400
```

- 4 threads calculating, 1 threads profiling

```
./03-concurrent 4 1
Avg. time for fib(30) = 1.088290 ms; 0.500000 branches/call 400
```

Shameless Self-Plug: Open PhD Position

TUHH
Technische Universität Hamburg



***O**perating*
***S**ystem*
***G**roup*

- Operating System Group: Young Basic Research Group
  - Topics: Non-Volatile Memory, Variability, Dependable Real-Time Systems
  - Techniques: Virtual memory, Compilers, Fault Injection
  - Open PhD Position: 100% E13, 3 years
- Technical University Hamburg
  - ~8000 students, ~90 professors
  - Strong systems-oriented computer science
  - South of the Elbe

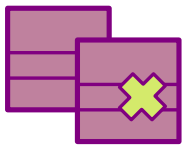
$\Rightarrow$ Get in contact, if interested

Agenda

Multiverse
Efficient Semi-Dynamic Variability

EuroSys'19

- Preparing variants at compile time
- Efficient binding by run-time code patching



Wait-free code patching
Run-time code patching without interruption

OSDI'20

- Preparing changes in a separate address space
- Incremental migration of threads

Referenzen I

TUHH
Technische Universität Hamburg

[1] Florian Rommel, Christian Dietrich, Daniel Friesel u. a. „From Global to Local Quiescence: Wait-Free Code Patching of Multi-Threaded Processes“. In: *14th Symposium on Operating System Design and Implementation (OSDI '20)*. Nov. 2020, S. 651–666. URL: https://www.usenix.org/conference/osdi20/presentation/rommel.

[2] Florian Rommel, Christian Dietrich, Michael Rodin u. a. „Multiverse: Compiler-Assisted Management of Dynamic Variability in Low-Level System Software“. In: *Fourteenth EuroSys Conference 2019 (EuroSys '19)* (Dresden, Germany). New York, NY, USA: ACM Press, 2019. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303959.

[3] Florian Rommel, Lennart Glauer, Christian Dietrich u. a. „Wait-Free Code Patching of Multi-Threaded Processes“. In: *Proceedings of the 10th SOSP Workshop on Programming Languages and Operating Systems (PLOS '19)*. Huntsville, ON, Canada: ACM, 2019, S. 23–29. ISBN: 978-1-4503-7017-2. DOI: 10.1145/3365137.3365404.

[4] Valentin Rothberg, Christian Dietrich, Alexander Graf u. a. „Function Multiverses for Dynamic Variability“. In: *Foundations and Applications of Self* Systems*. Hrsg. von Jesper Andersson, Rafael Capilla und Holger Eichelberger. Augsburg, 2016. URL: https://www4.cs.fau.de/Publications/2016/rothberg_16_dspl.pdf.