ISE SRA

Leibniz Universität Hannover

From Global to Local Quiescence: Wait-Free Code Patching of Multi-Threaded Processes

Florian Rommel, Christian Dietrich, Daniel Friesel,
Marcel Köppen, Michael Müller, Olaf Spinczyk, Daniel Lohmann

Leibniz Universität Hannover
Universität Osnabrück

25.09.2020

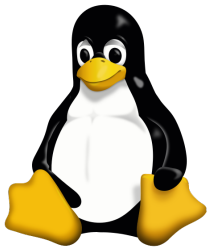
Dynamic Software Updating

→ Einspielen von Updates während der Laufzeit eines Programms

- **Hochverfügbarkeit:**
  Servicequalität darf nicht einbrechen
- **Neustart ist teuer:**
  Großer Laufzeitzustand,
  oder anderweitig teurer Startup

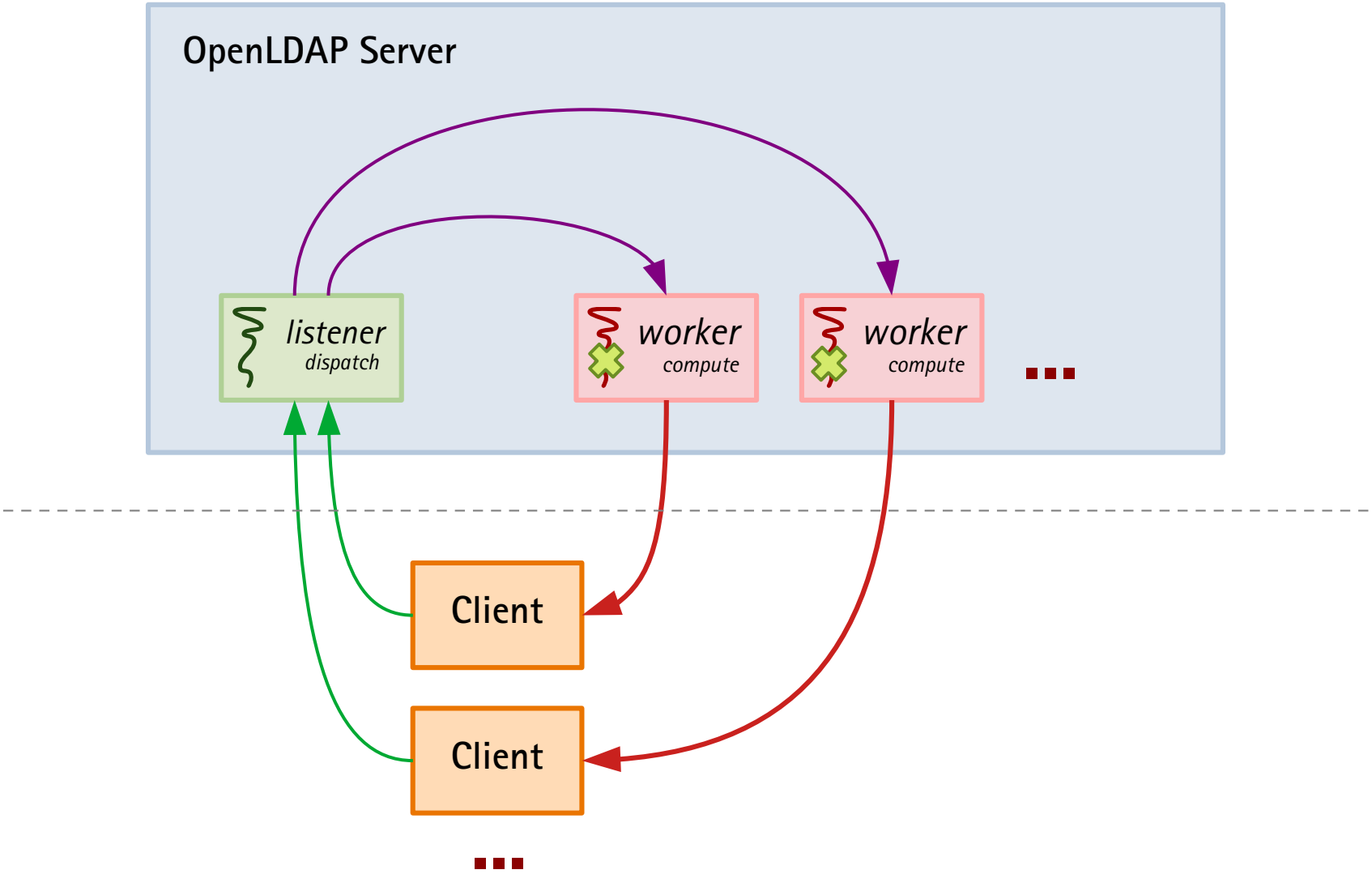
Beispiel: Linux-Kernel, von Distributoren angeboten

→ Kpatch, Ksplice, kGraft



Keine tatsächlich verwendete Lösung für Userspace-Applikationen

Beispiel: OpenLDAP



Beispiel: OpenLDAP

```
void worker_thread() {
 while (1) {
  wait_for_work();
  do_work();
 }
}
```

do_work()

```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  op->o_tmpfree( ros->mapped_attrs, op->o_tmpmemctx );
  filter_free_x( op, op->ors_filter, 1 );
  op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
  op->ors_attrs = ros->ors_attrs;
  op->ors_filter = ros->ors_filter;
  op->ors_filterstr = ros->ors_filterstr;
  ...
}
```

buggy

```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  if ( op->ors_filter != ros->ors_filter ) {
    filter_free_x( op, op->ors_filter, 1 );
    op->ors_filter = ros->ors_filter;
  }
  if ( op->ors_filterstr.bv_val != ros->ors_filterstr.bv_val ) {
    op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
    op->ors_filterstr = ros->ors_filterstr;
  }
  ...
}
```

patched

Beispiel: OpenLDAP

```
void worker_thread( ) {
 while (1) {
  wait_for_work( );
  do_work( );

  // quiescence point
  if (patch_pending( )) {
    barrier( );
    wait_for_patch( );
  }
 }
}
```

```
void patcher_thread( ) {
 while (1) {
  wait_for_patch_request( );
  set_patch_pending( );
  barrier( );
  apply_patch( );
  reset_patch_pending( );
  resume_workers( );
 }
}
```

do_work( )

```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  op->o_tmpfree( ros->mapped_attrs, op->o_tmpmemctx );
  filter_free_x( op, op->ors_filter, 1 );
  op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
  op->ors_attrs = ros->ors_attrs;
  op->ors_filter = ros->ors_filter;
  op->ors_filterstr = ros->ors_filterstr;
  ...
}
```

buggy

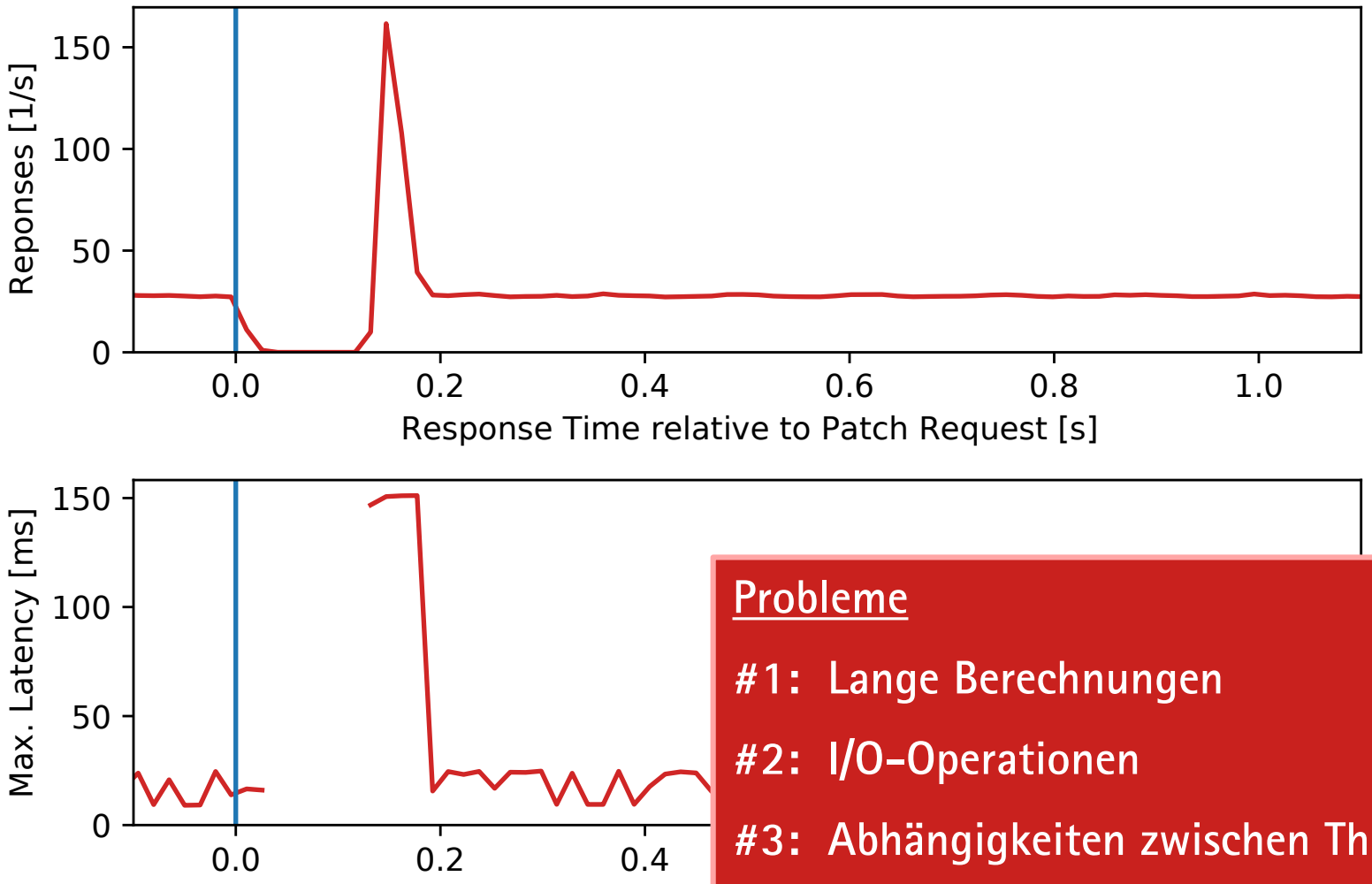
```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  if ( op->ors_filter != ros->ors_filter ) {
    filter_free_x( op, op->ors_filter, 1 );
    op->ors_filter = ros->ors_filter;
  }
  if ( op->ors_filterstr.bv_val != ros->ors_filterstr.bv_val ) {
    op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
    op->ors_filterstr = ros->ors_filterstr;
  }
  ...
}
```

patched

Alle Worker müssen in der Barriere sein, bevor der Patch eingespielt werden kann.

Global Quiescence

Der zu patchende Code ist in keinem Thread aktiv



Probleme

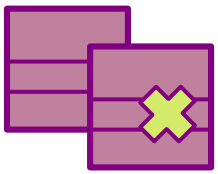
#1: Lange Berechnungen

#2: I/O-Operationen

#3: Abhängigkeiten zwischen Threads

Local Quiescence

Idee: Threads werden unabhängig voneinander gepatcht.



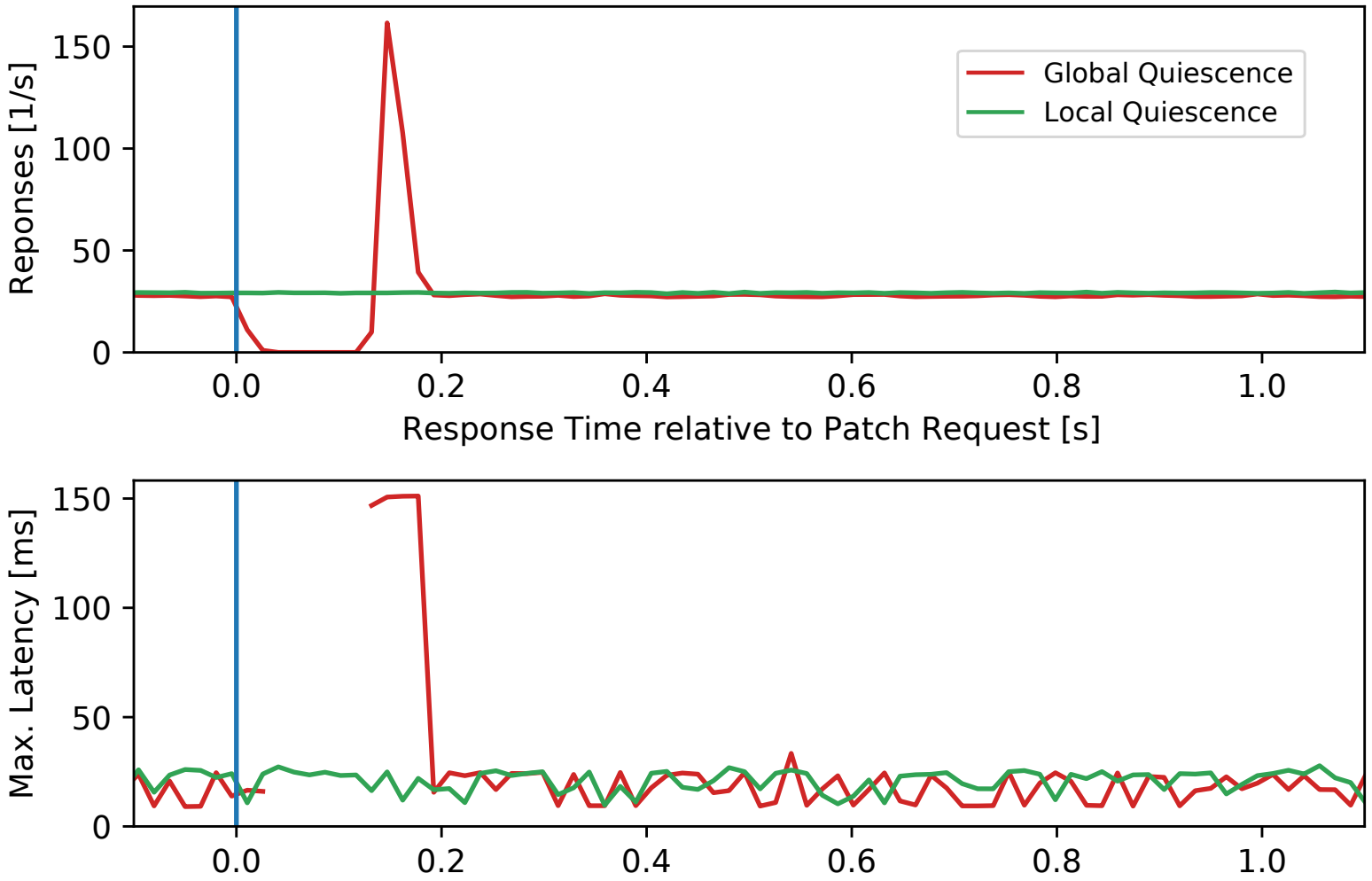
Wait-Free Code Patching
via Address-Space Generations

OS-Erweiterung für Live-Patching für multithreaded Anwendungen

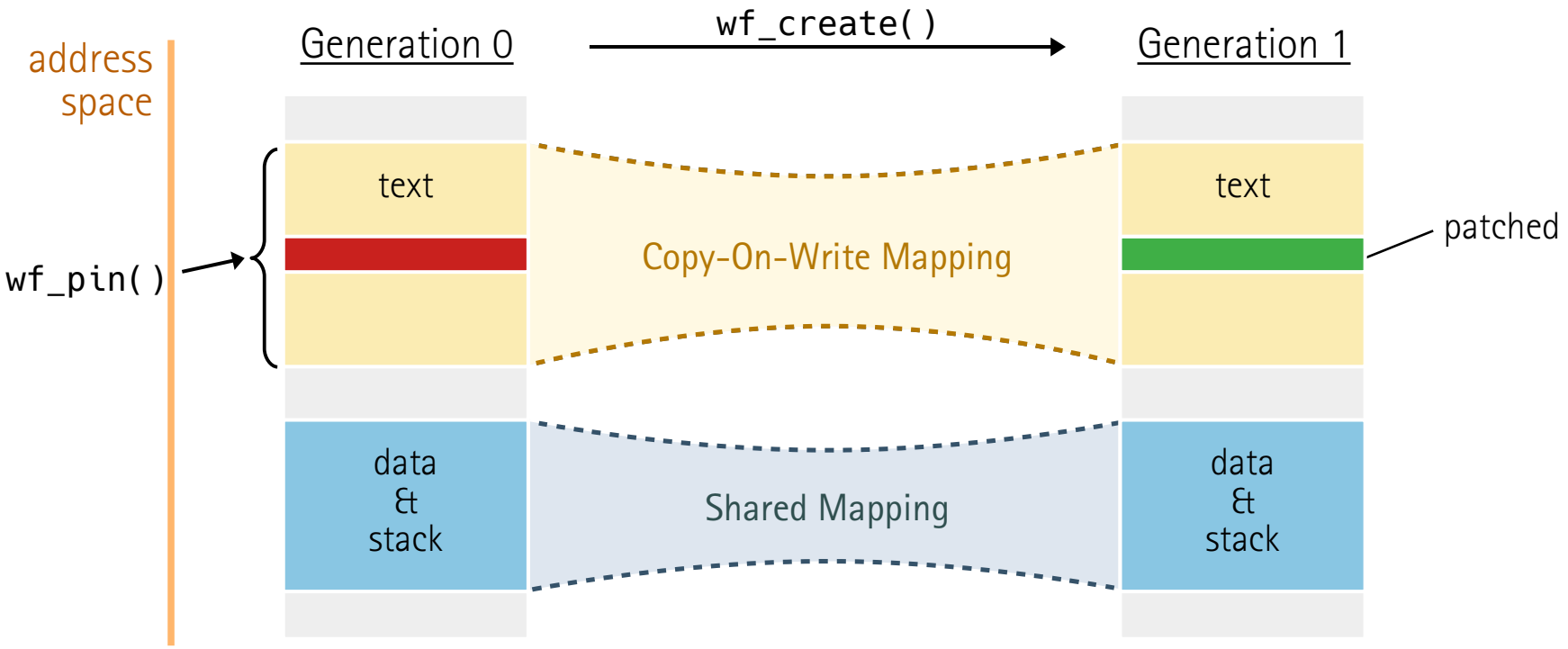
- Mehrere Sichten auf einen Adressraum (Adressraum-Generationen)
- (Thread-) Local Quiescence
- Thread-weise Migration zwischen den Adressraum-Generationen

→ Implementierung für den Linux-Kernel

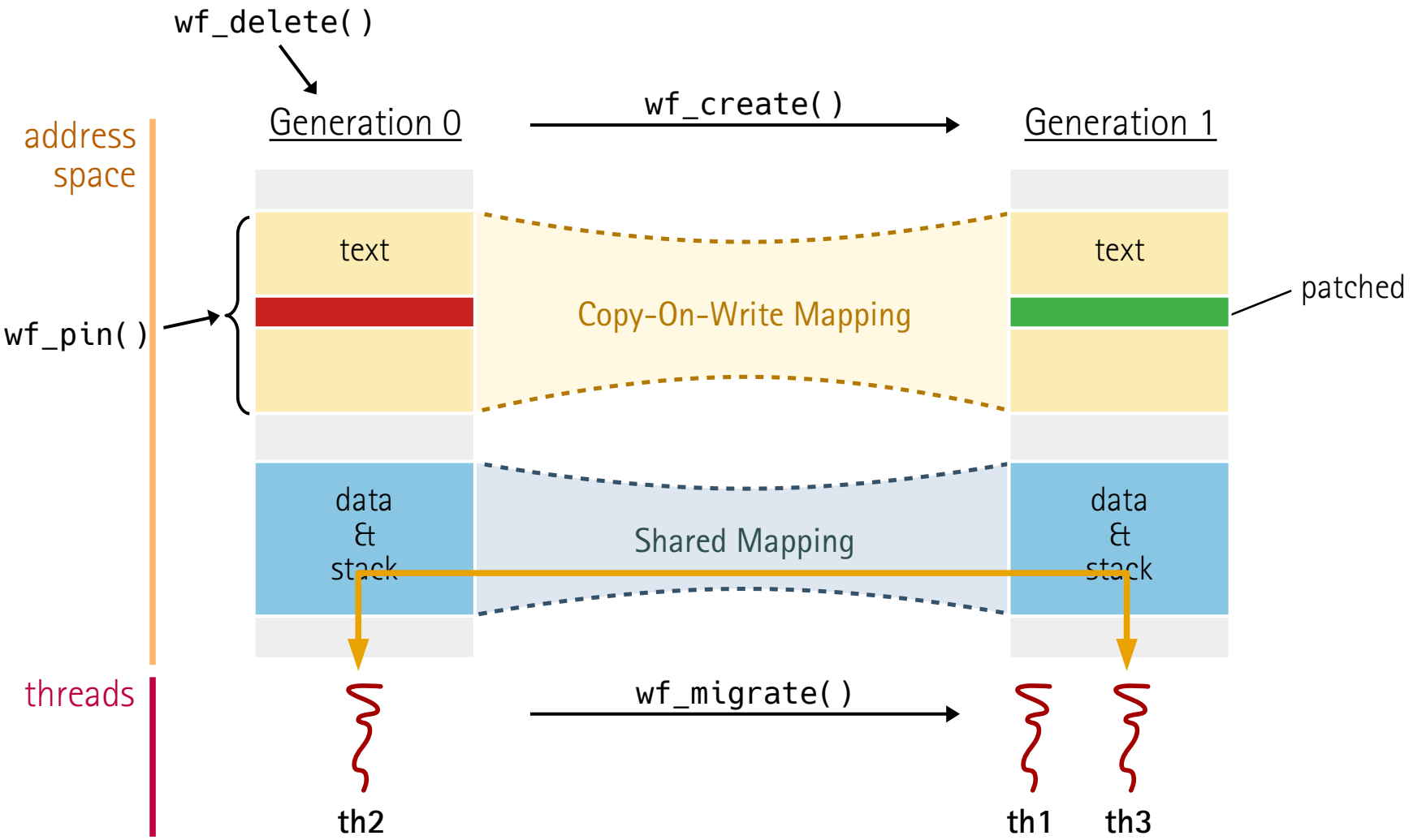
Local Quiescence



Adressraum-Generationen



Adressraum-Generationen



Zurück zu OpenLDAP

```
void worker_thread() {
 while (1) {
  wait_for_work();
  do_work();

  // quiescence point
  if (migration_pending()) {
    wf_migrate();
  }

 }
}
```

```
void patcher_thread() {
 while (1) {
  wait_for_patch_request();
  wf_create();
  wf_migrate();
  apply_patch();
  set_migration_pending();
 }
}
```

do_work()

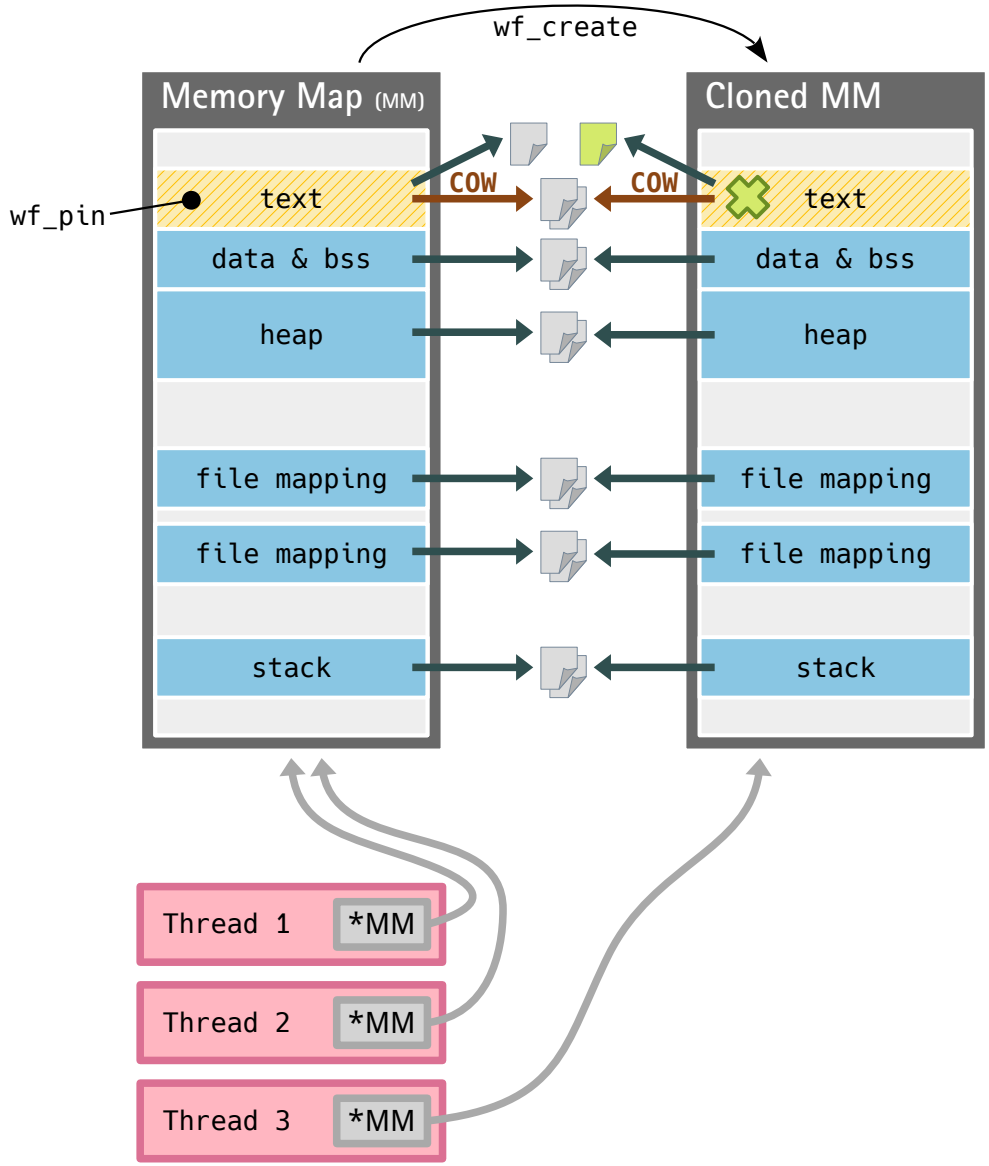
```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  op->o_tmpfree( ros->mapped_attrs, op->o_tmpmemctx );
  filter_free_x( op, op->ors_filter, 1 );
  op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
  op->ors_attrs = ros->ors_attrs;
  op->ors_filter = ros->ors_filter;
  op->ors_filterstr = ros->ors_filterstr;
  ...
}
```

buggy

```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  if ( op->ors_filter != ros->ors_filter ) {
    filter_free_x( op, op->ors_filter, 1 );
    op->ors_filter = ros->ors_filter;
  }
  if ( op->ors_filterstr.bv_val != ros->ors_filterstr.bv_val ) {
    op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
    op->ors_filterstr = ros->ors_filterstr;
  }
  ...
}
```

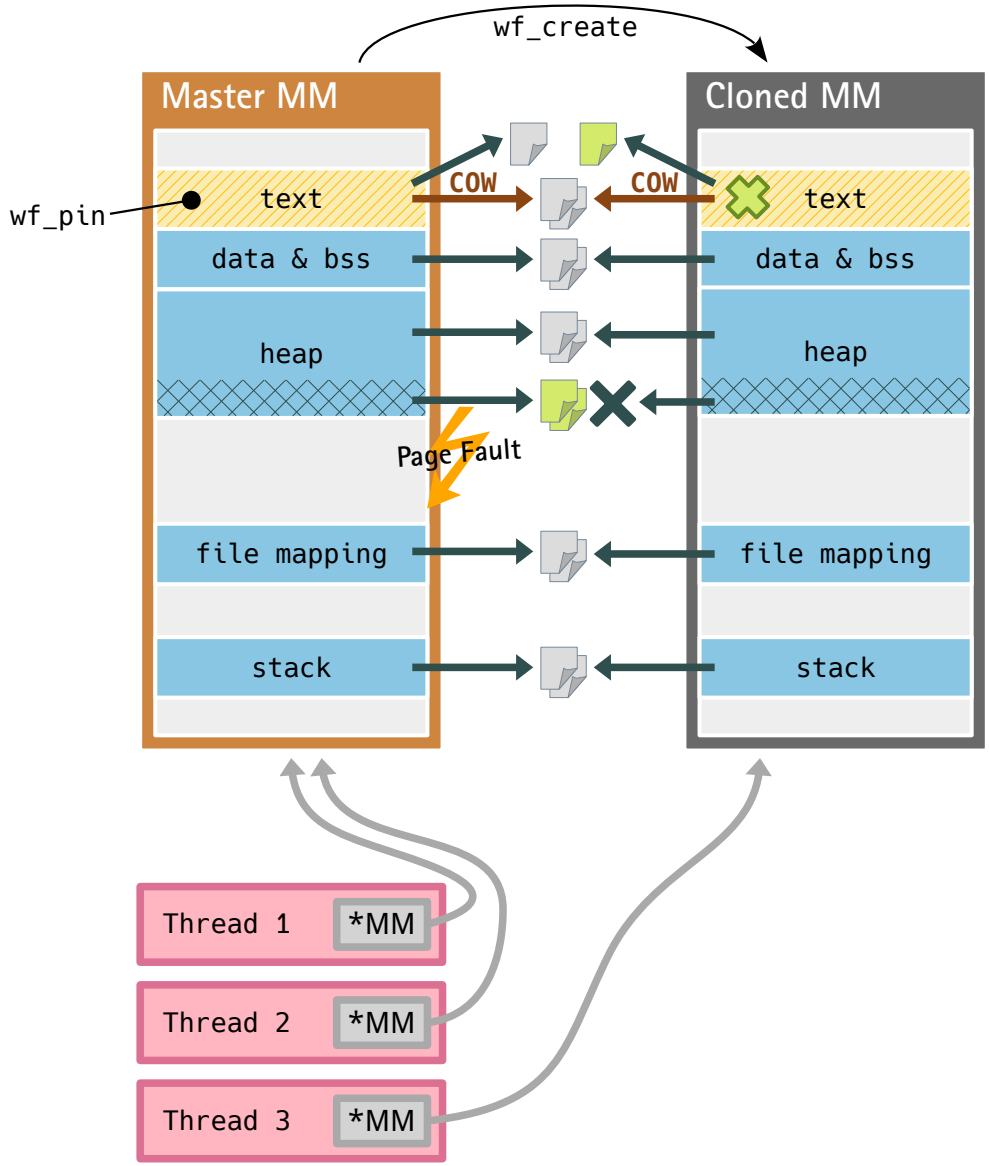
patched

Implementierung im Linux-Kernel



- **wf_create**
  - Erstellt Klon der Memory-Map (= Generation) wie bei `fork()` nur ohne COW
  - Gepinnte Mappings haben COW-Verhalten
- **wf_migrate**
  - Anpassung des MM-Zeigers des Threads
  - Kontextwechsel

Implementierung im Linux-Kernel



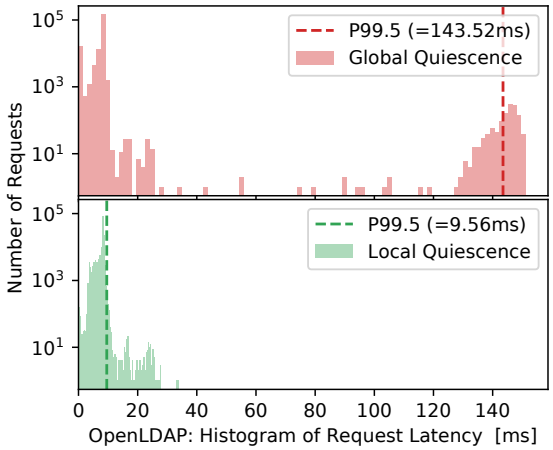
- **wf_create**
  - Erstellt Klon der Memory-Map (= Generation) wie bei `fork()` nur ohne COW
  - Gepinnte Mappings haben COW-Verhalten
- **wf_migrate**
  - Anpassung des MM-Zeigers des Threads
  - Kontextwechsel
- **Mapping-Änderungen** (mmap, munmap, ...)
  - Synchronisiert auf allen MMs
  - Master-MM: Lazy-Page-Initialization, und „Locking-Proxy" für alle MM-Klone

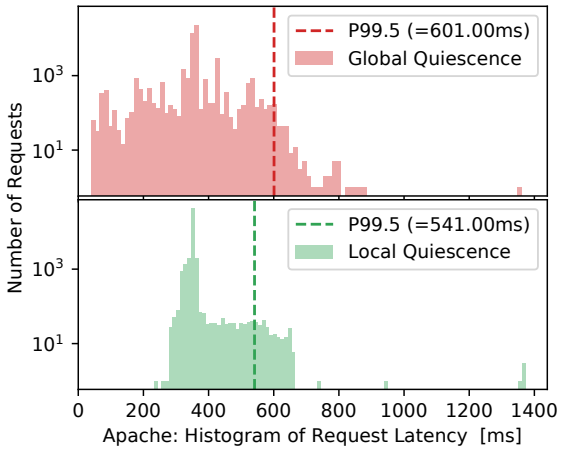
Evaluation: Patches

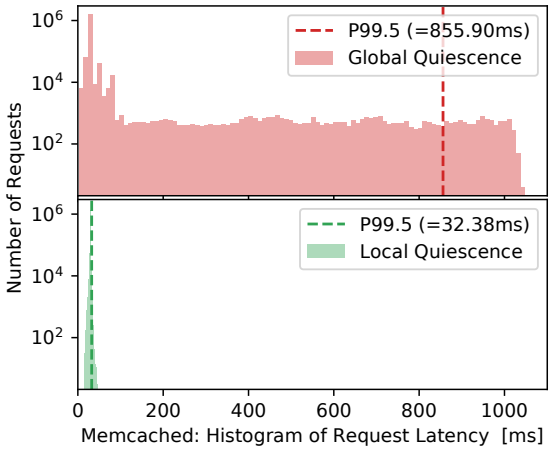
Debian 10.0 Packages und Debian Patches (außer MariaDB)

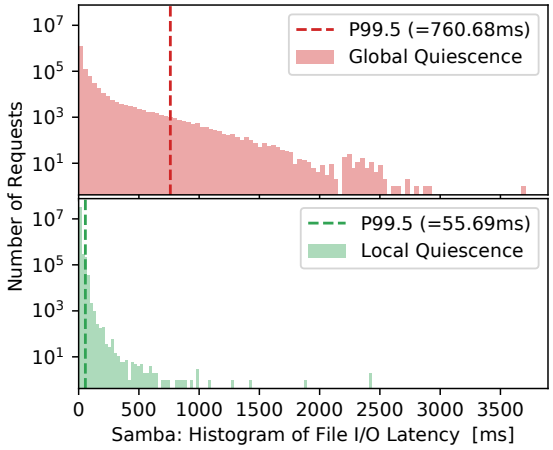
	OpenLDAP	Apache	Memcached	Samba	MariaDB*	Node.js
Patches (CVE)	13 (2)	10 (10)	1 (1)	2 (2)	74 (26)	4 (0)
↓ **Beschränkung auf code-only Patches**						87% (88%)
text-only	9 (2)	7 (7)	1 (1)	2 (2)	67 (24)	4 (0)
↓ **Einspielen der Patches mit modifiziertem Kpatch**						39% (47%)
kpatch'able	9 (2)	7 (7)	1 (1)	2 (2)	16 (5)	0 (0)

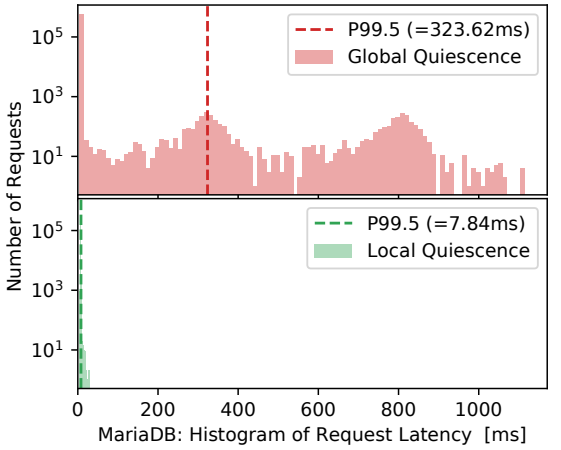
Evaluation: Request-Latenzen

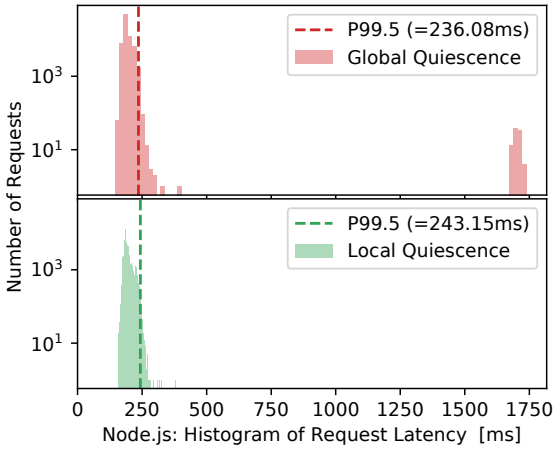




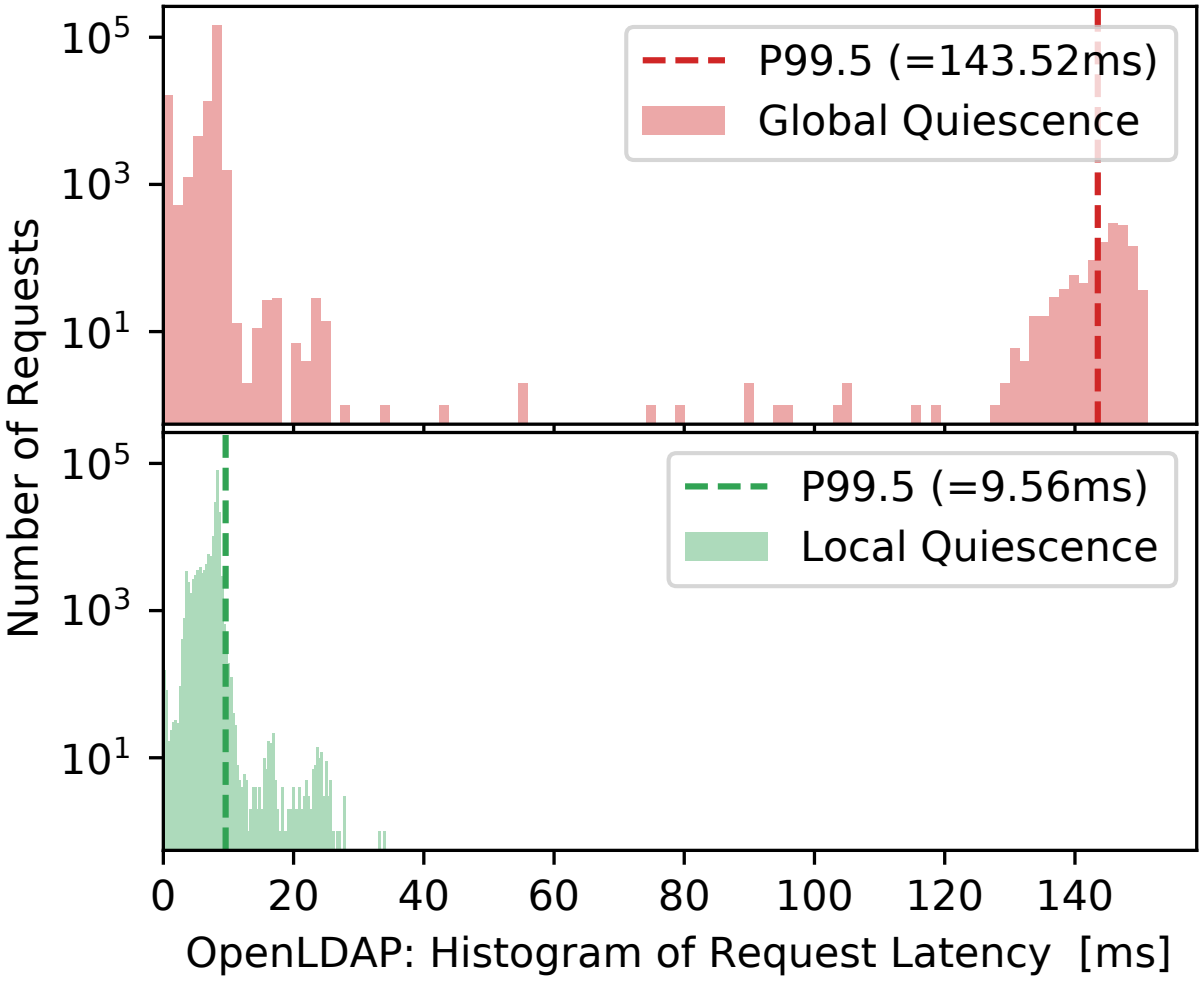








Evaluation: Request-Latenzen



Evaluation: Overhead

- Laufzeit (Microbenchmarks unter Last):
  - wf_create

    88±23 µs (Memcached) bis **2171±139 µs** (Node.js)
  - wf_migrate

    5±5 µs (Samba) bis **8±7 µs** (Node.js)
- Speicher (unter Last):

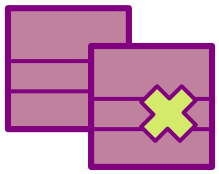
  132 KiB (Memcached) bis **1808 KiB** (Node.js)

Future Work

Zugrundeliegender Mechanismus:
Synchronisierte Adressraum-Klone mit partiellen Unterschieden

- Weitere Anwendungen
  - Kombination mit JIT-Compiler
  - Pfad-spezifische Kernel-Modifikationen (→ Synthesis)
  - Implementierung von dynamischer Variabilität (→ Multiverse)
  - Unterschiedliche Adressraum-Views für Daten (Isolation von Threads)
- Geplanter DFG-Antrag

Zusammenfassung



Wait-Free Code Patching
via Address-Space Generations

- Ziel: Reduzierung von Global Quiescence auf Local Quiescence
  - Vereinfachte Herstellung von Quiescence
  - Keine Einbußen bei der Quality of Service
- Umsetzung mit Hilfe von Adressraum-Generationen
  Implementiert als Kernel-Erweiterung für Linux
- Evaluation mit 6 Serveranwendungen
  - Erfolgreiche Anwendung von Code-Only-Patches während der Laufzeit
  - Signifikante Verbesserungen der Tail-Latency bei Wait-Free Patching