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

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

Leibniz Universität Hannover
Universität Osnabrück

2020-11-05

Dynamic Software Updating

Apply updates during the run time

Dynamic Software Updating

Apply updates during the run time

- High Availability *service quality must not decrease*
- Expensive Reboot *e.g., applications with large run-time state*

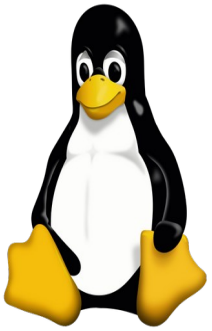
Dynamic Software Updating

Apply updates during the run time

- High Availability *service quality must not decrease*
- Expensive Reboot *e.g., applications with large run-time state*

Prime Example: Operating Systems

→ Linux Kernel *(Ksplice, kGraft)*



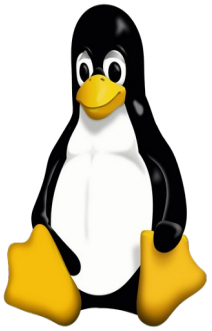
Dynamic Software Updating

Apply updates during the run time

- **High Availability** *service quality must not decrease*
- **Expensive Reboot** *e.g., applications with large run-time state*

Prime Example: Operating Systems

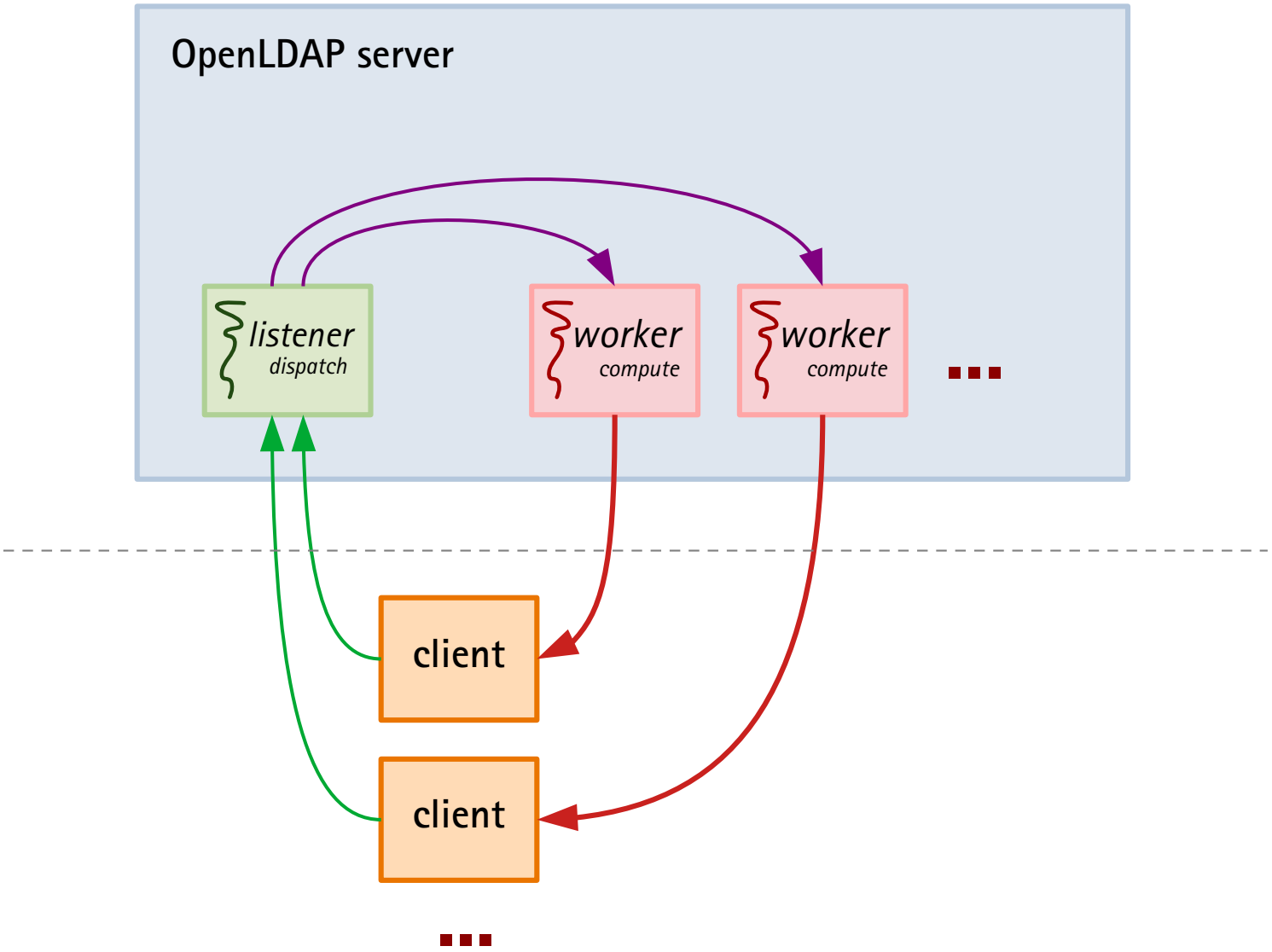
→ Linux Kernel *(Ksplice, kGraft)*



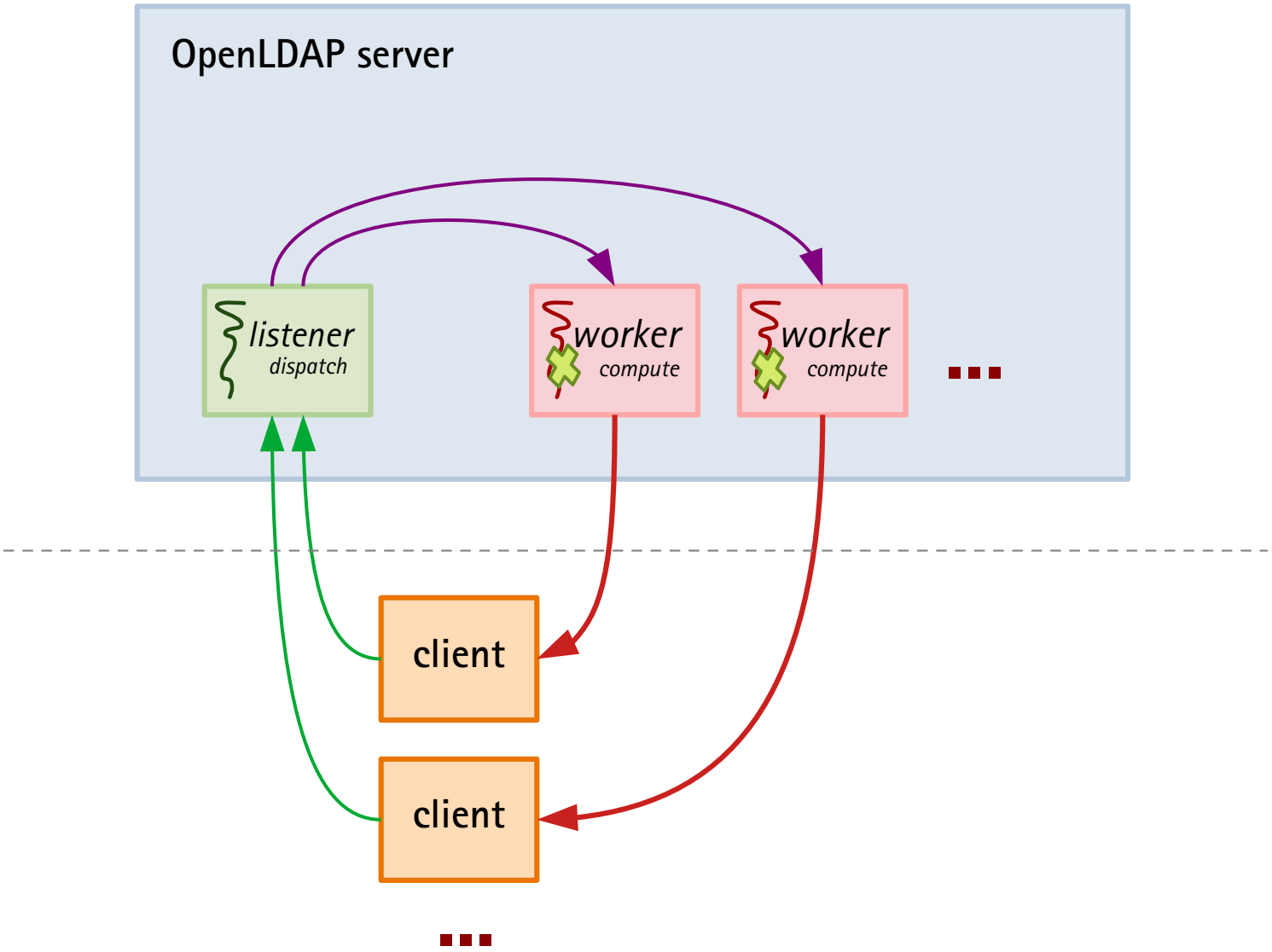
Userspace Applications?

→ DSU rarely used in practice

Example: OpenLDAP



Example: OpenLDAP



Example: OpenLDAP

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

Example: OpenLDAP

`do_work()`

```
void worker_thread() {
 while (1) {
  wait_for_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

Example: OpenLDAP

`do_work()`

```
void worker_thread() {
 while (1) {
  wait_for_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

Example: OpenLDAP

`do_work()`

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

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

```
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

Example: OpenLDAP

`do_work()`

```
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();
 }
}
```

```
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

Example: 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

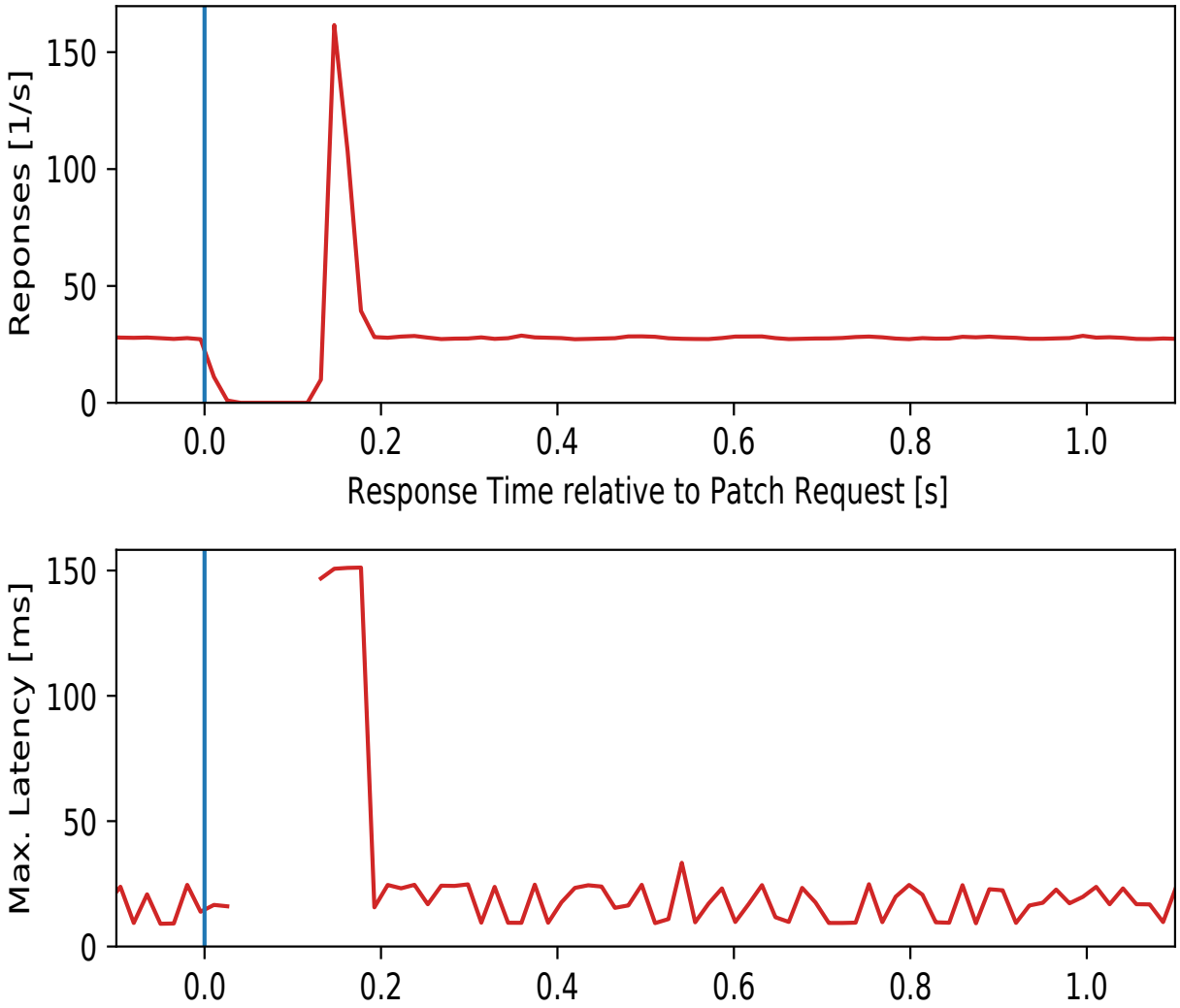
```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  if ( op->ors_filter != ros->ors_filter ) {
    filter_free_x( op, op->ors_filter, 1 );
    op->ors_filter = ros->ors_filter;
  }
  if ( op->ors_filterstr.bv_val != ros->ors_filterstr.bv_val ) {
    op->o_tmpfree( op->ors_filterstr.bv_val,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

patched

Global Quiescence:
All workers must be in the barrier before patching

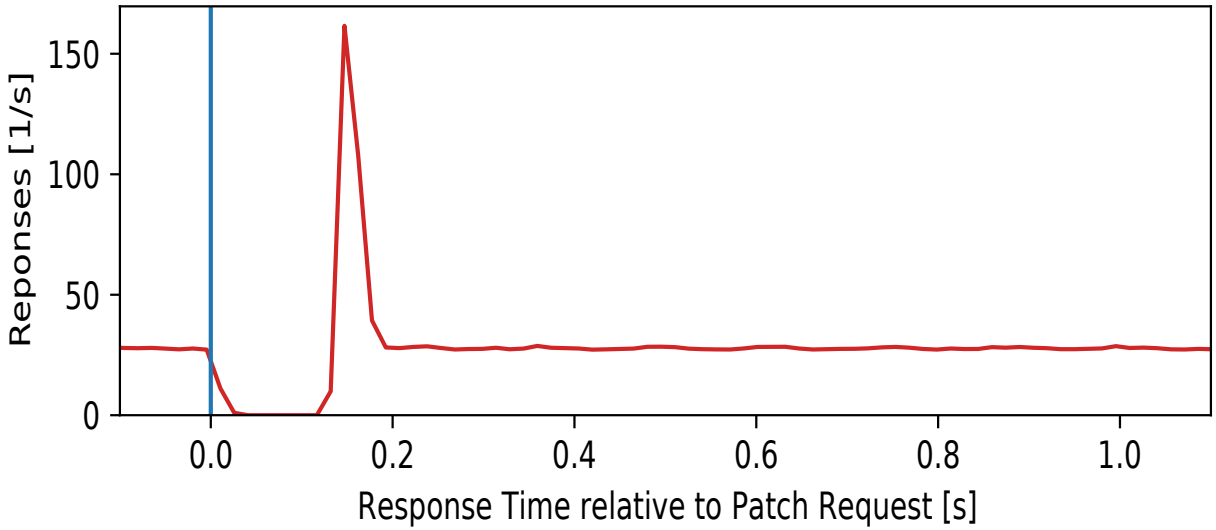
Global Quiescence

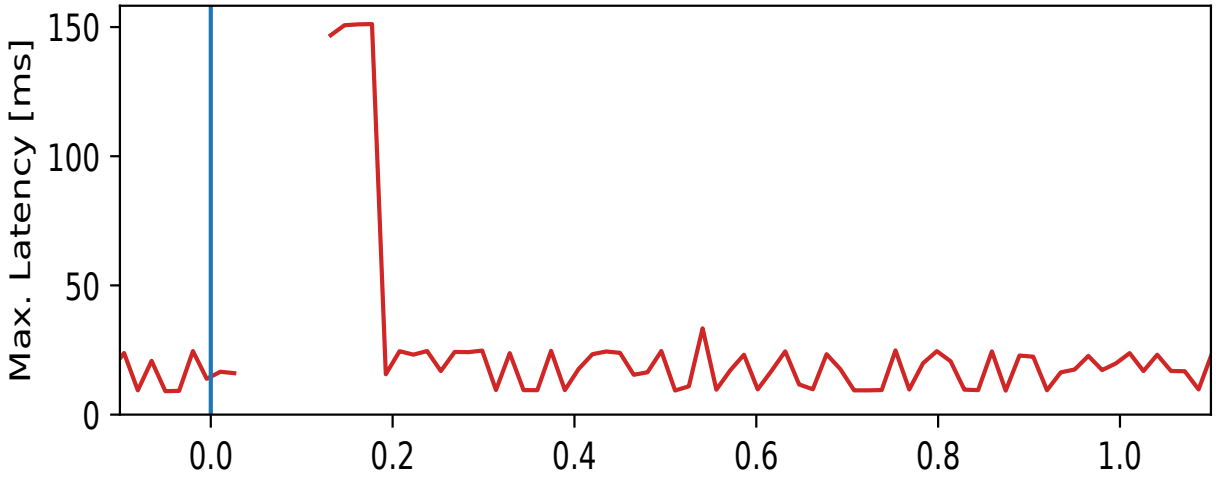
The to-be-patched code is not active in any thread



Global Quiescence

The to-be-patched code is not active in any thread





Problems

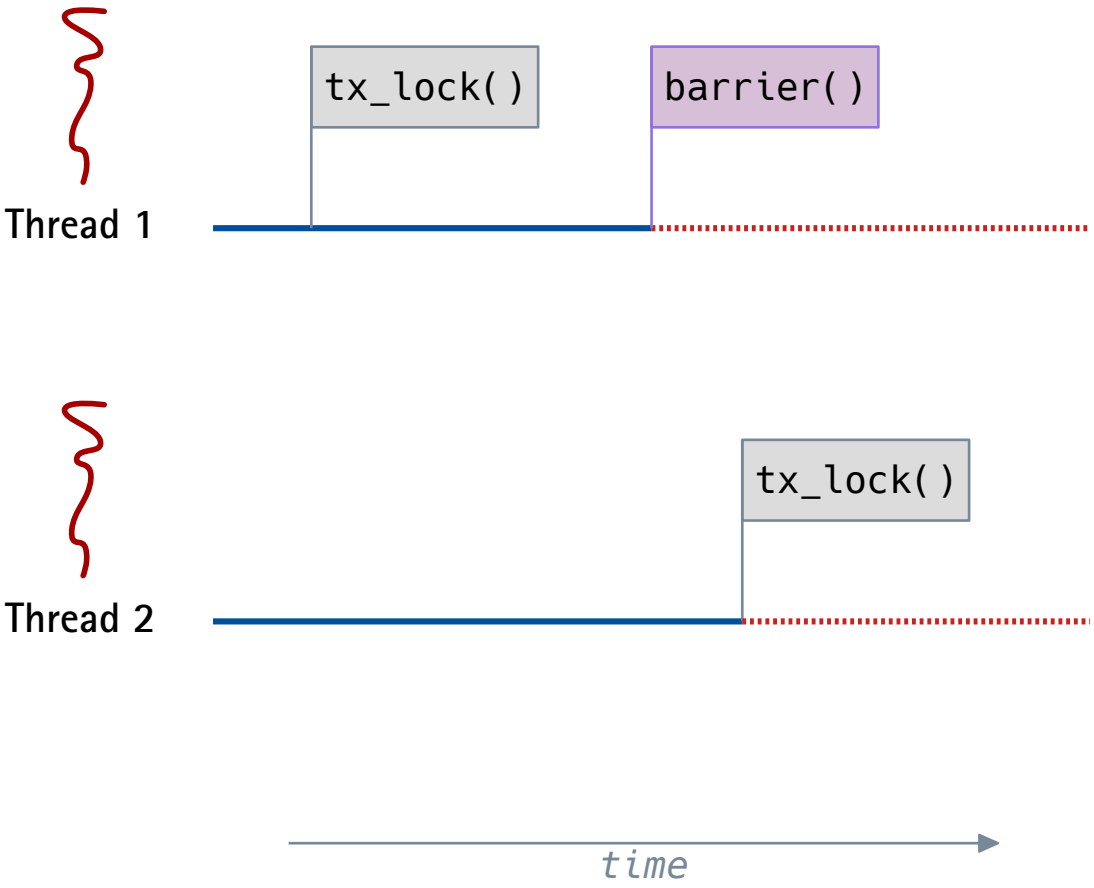
#1: Long Calculations

#2: I/O Operations

#3: Inter-Thread Dependencies

Global Quiescence

→ MariaDB: Transaction Locks



Problems

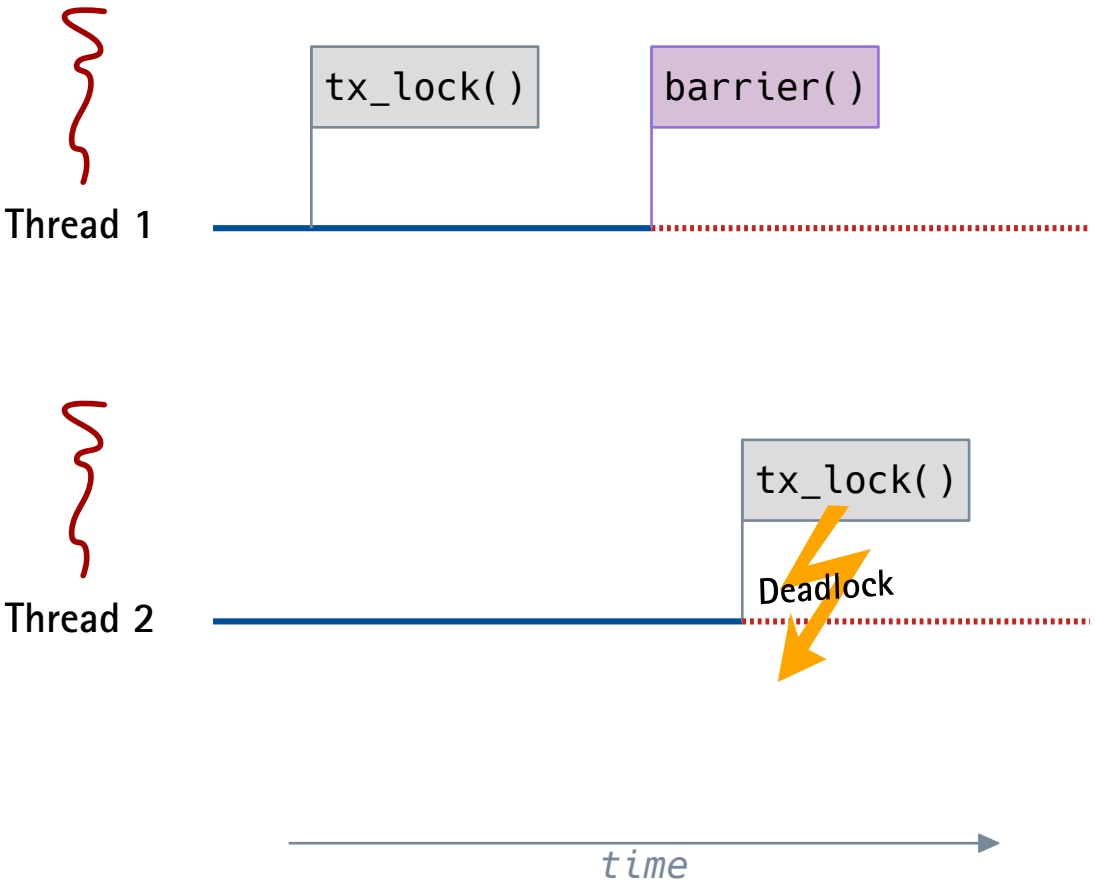
#1: Long Calculations

#2: I/O Operations

#3: Inter-Thread Dependencies

Global Quiescence

→ MariaDB: Transaction Locks



Problems

#1: Long Calculations

#2: I/O Operations

#3: Inter-Thread Dependencies

Kernelspace

- Ksplice: Probe for quiescence instead of waiting in a barrier

Kernelspace

- Ksplice: Probe for quiescence instead of waiting in a barrier

  → Patch may never get applied

Kernelspace

- Ksplice: Probe for quiescence instead of waiting in a barrier

  → Patch may never get applied

- kGraft, DynAMOS: Keep *patched and unpatched* functions in parallel

  Decide on per-thread-basis which version to use

Kernelspace

- Ksplice: Probe for quiescence instead of waiting in a barrier

  → Patch may never get applied

- kGraft, DynAMOS: Keep *patched and unpatched* functions in parallel

  Decide on per-thread-basis which version to use

  Global quiescence → local quiescence

Kernelspace

- Ksplice: Probe for quiescence instead of waiting in a barrier

  → Patch may never get applied

- kGraft, DynAMOS: Keep *patched and unpatched* functions in parallel

  Decide on per-thread-basis which version to use

  Global quiescence → local quiescence

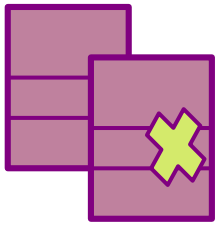
  → Problems: kernel-specific, performance penalty

Local Quiescence

Basic Idea: Patching threads independently from each other.

Local Quiescence

Basic Idea: Patching threads independently from each other.



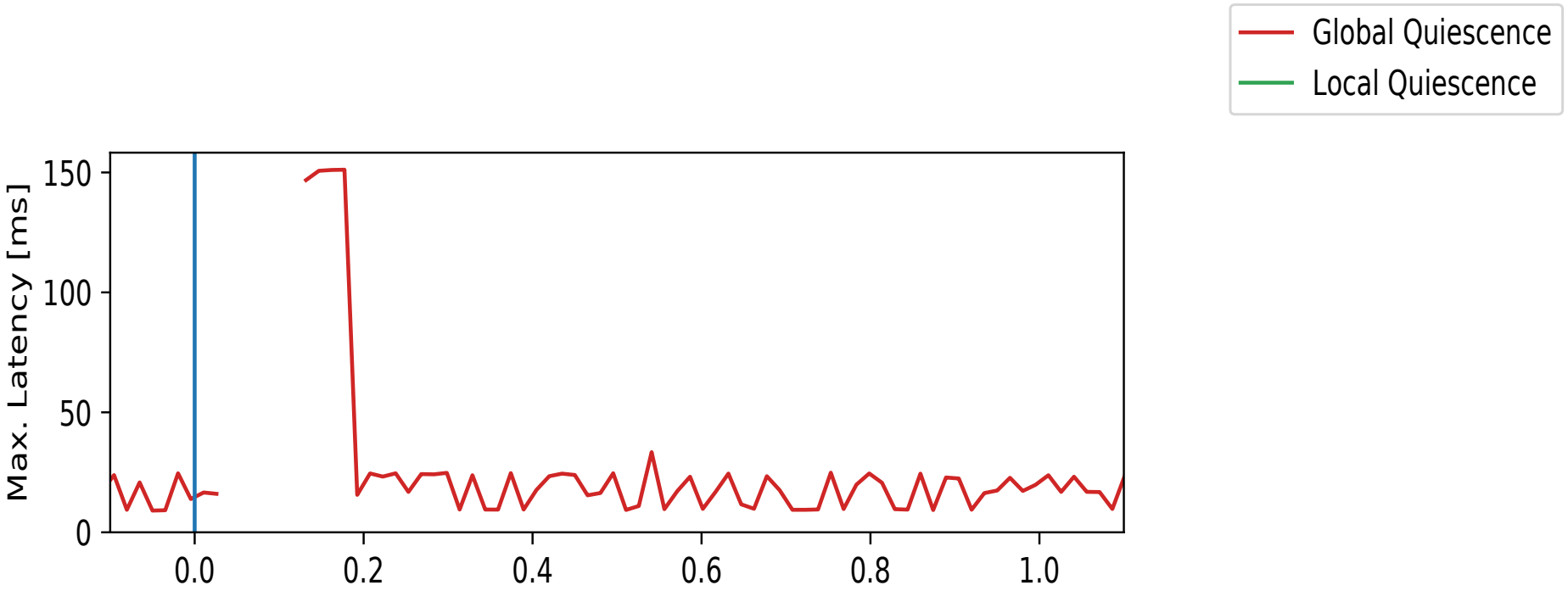
Wait-Free Code Patching
via Address-Space Generations

OS extension for run-time modification in multi-threaded processes

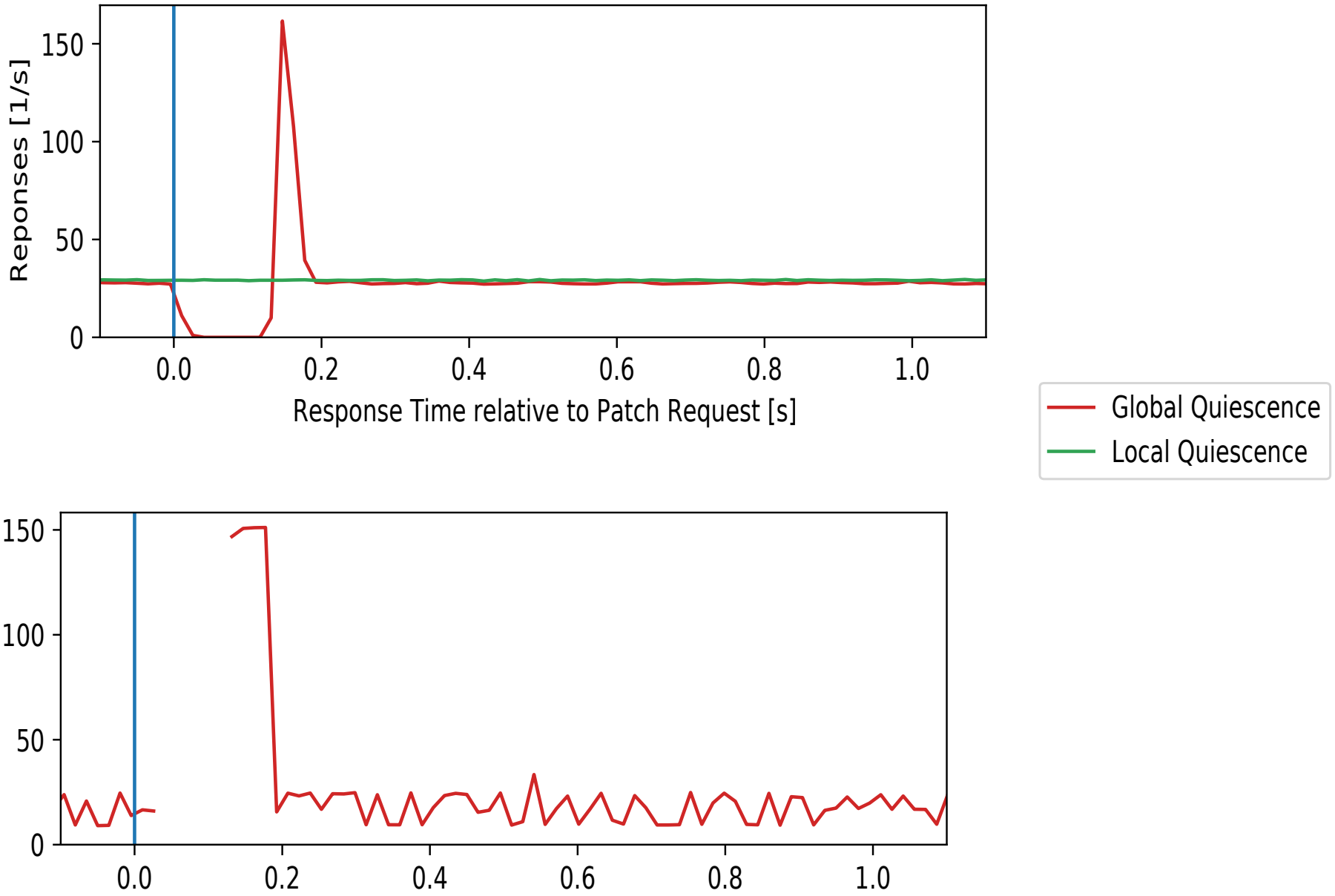
- AS generations: Multiple views of an address-space
- Thread-local quiescence
- Thread-by-thread migration between AS generations

→ Implementation in the Linux Kernel

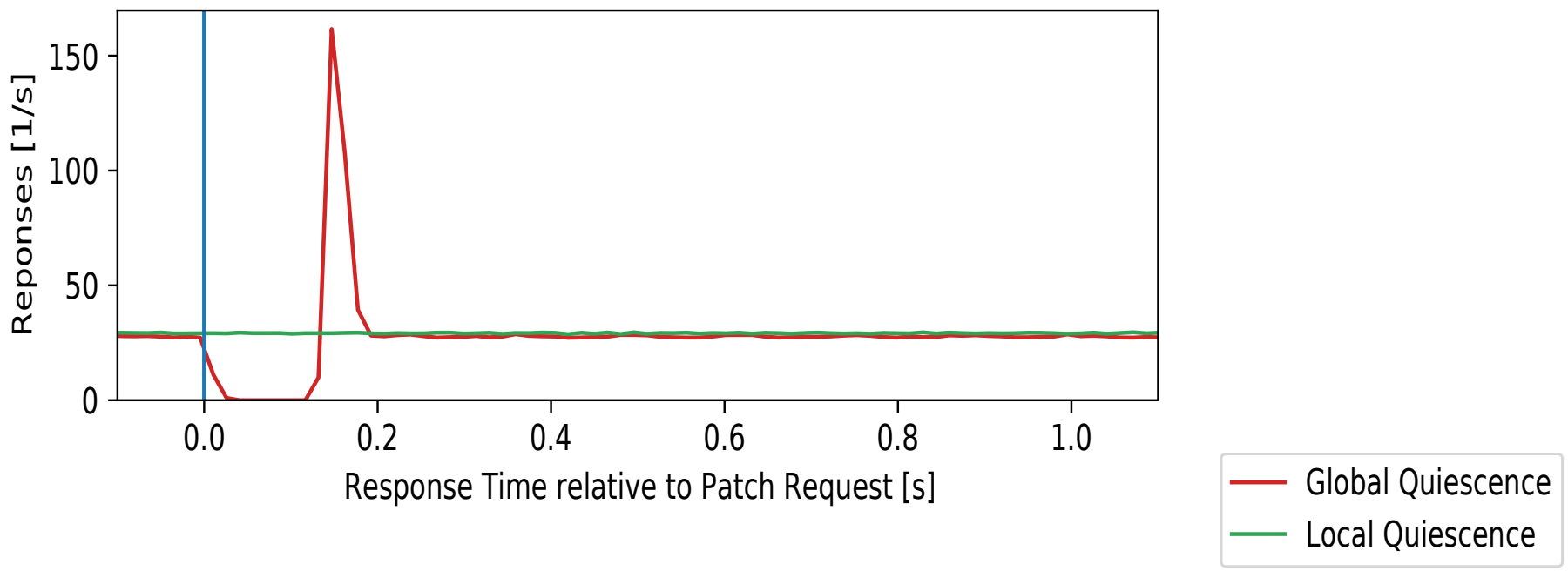
Local Quiescence



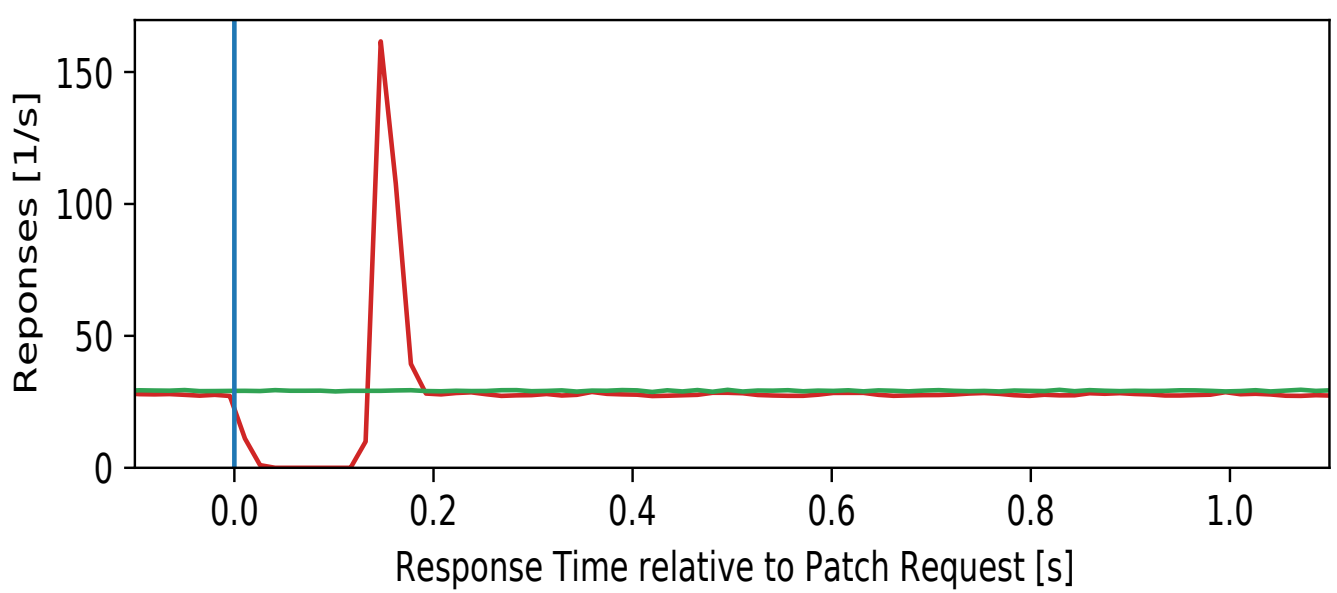
Local Quiescence



Local Quiescence



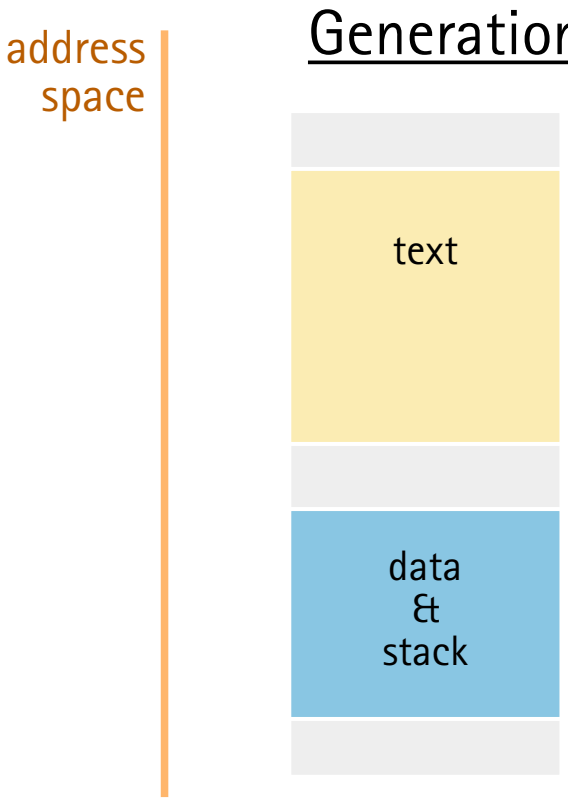
Local Quiescence



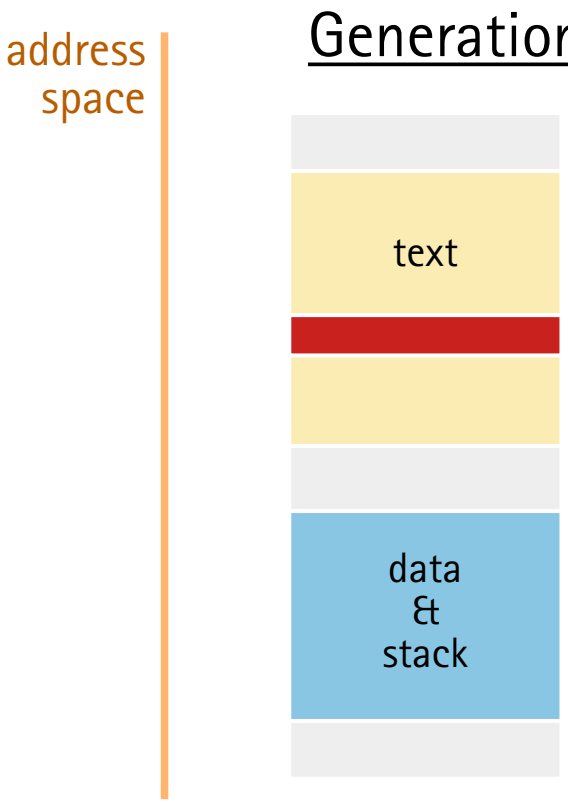
Inter-Thread Dependencies
→ No more deadlocks!

Global Quiescence
Local Quiescence

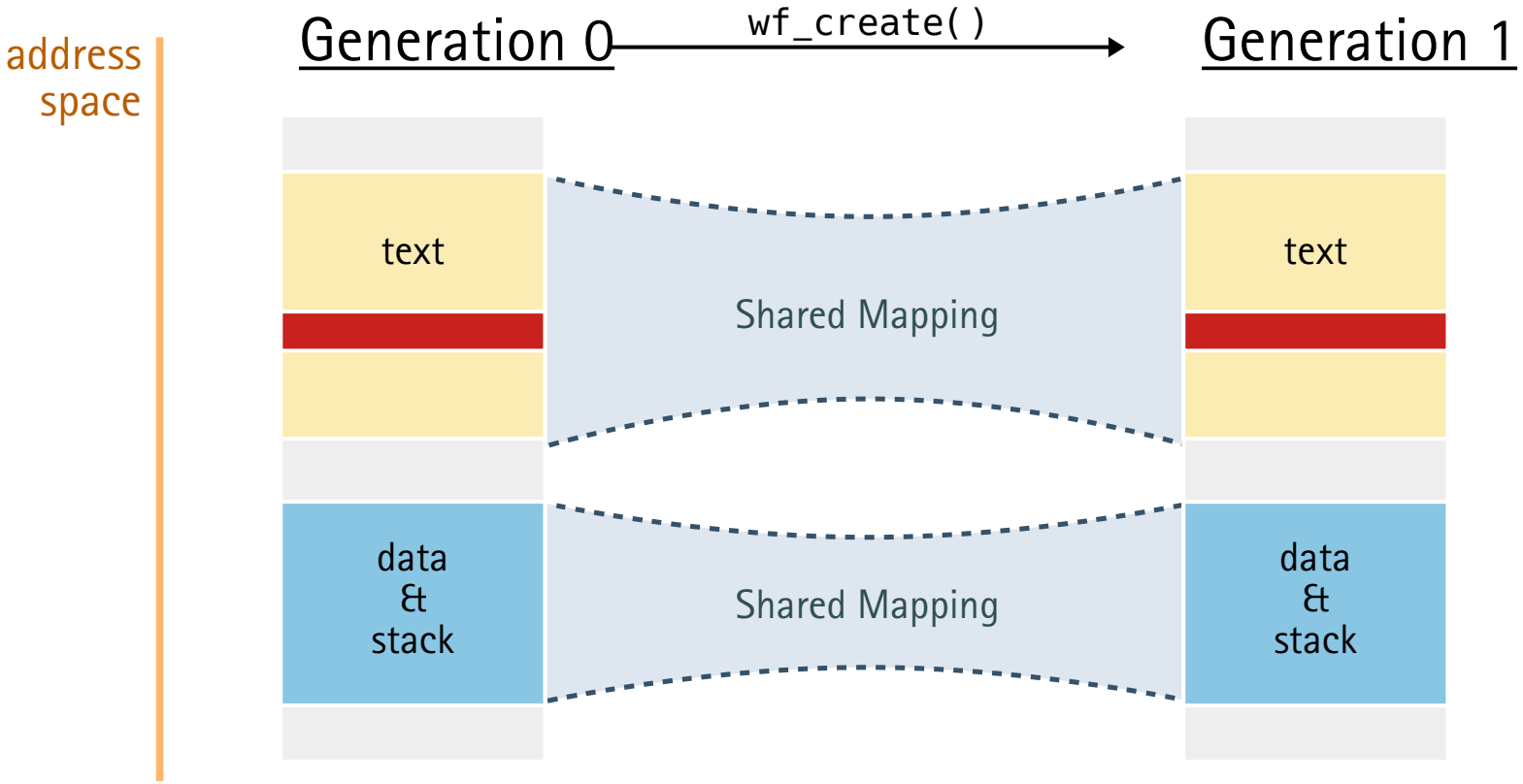
Address-Space Generations



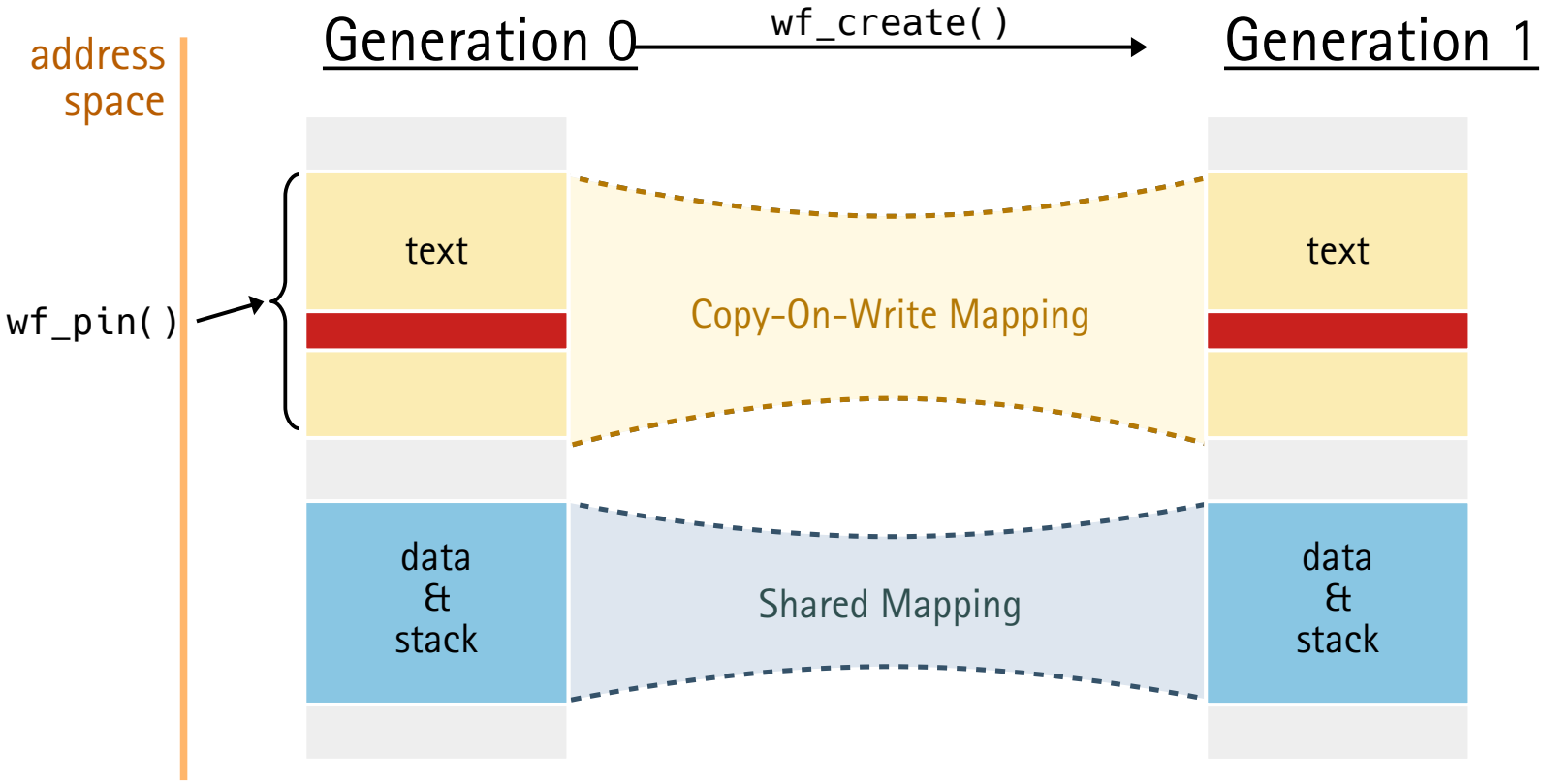
Address-Space Generations



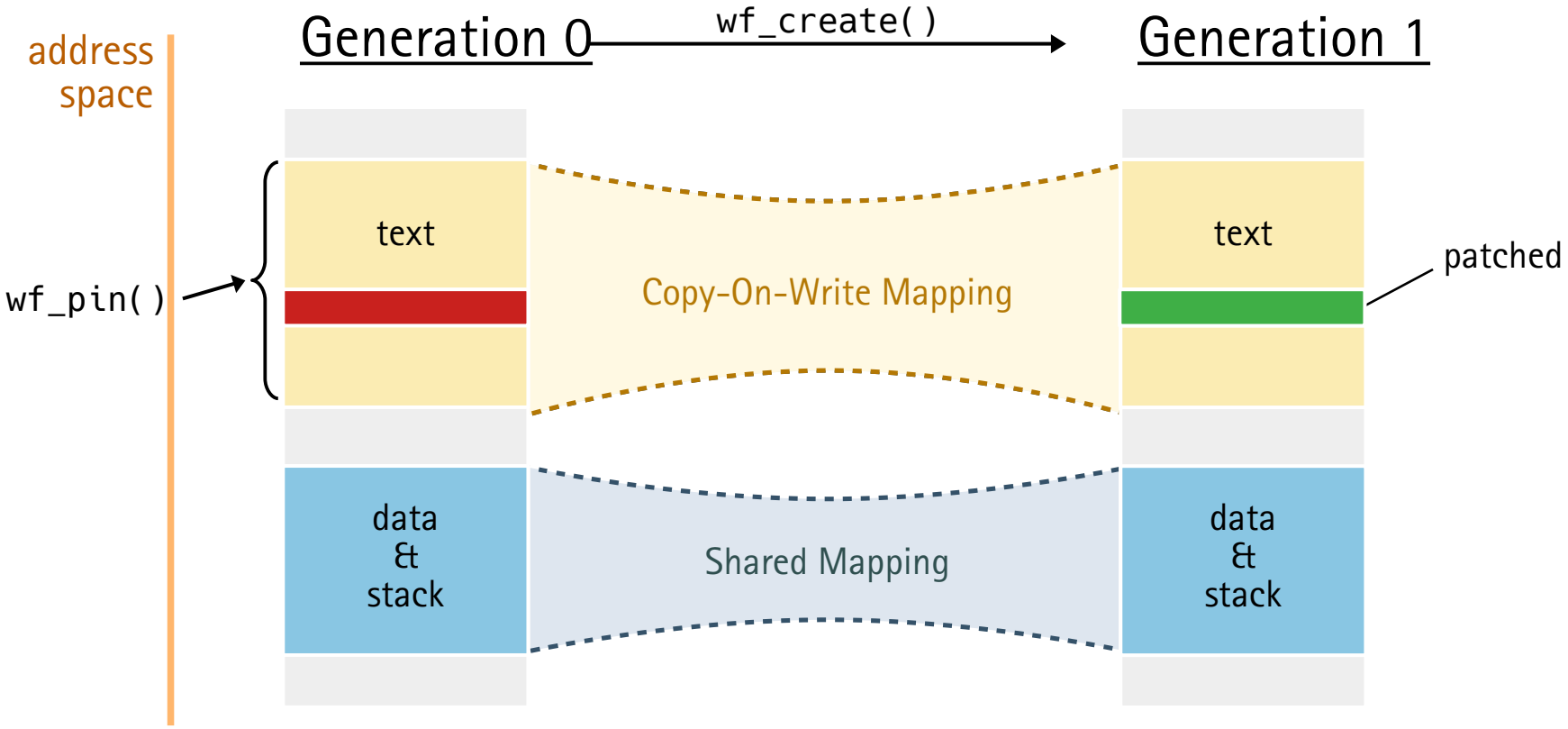
Address-Space Generations



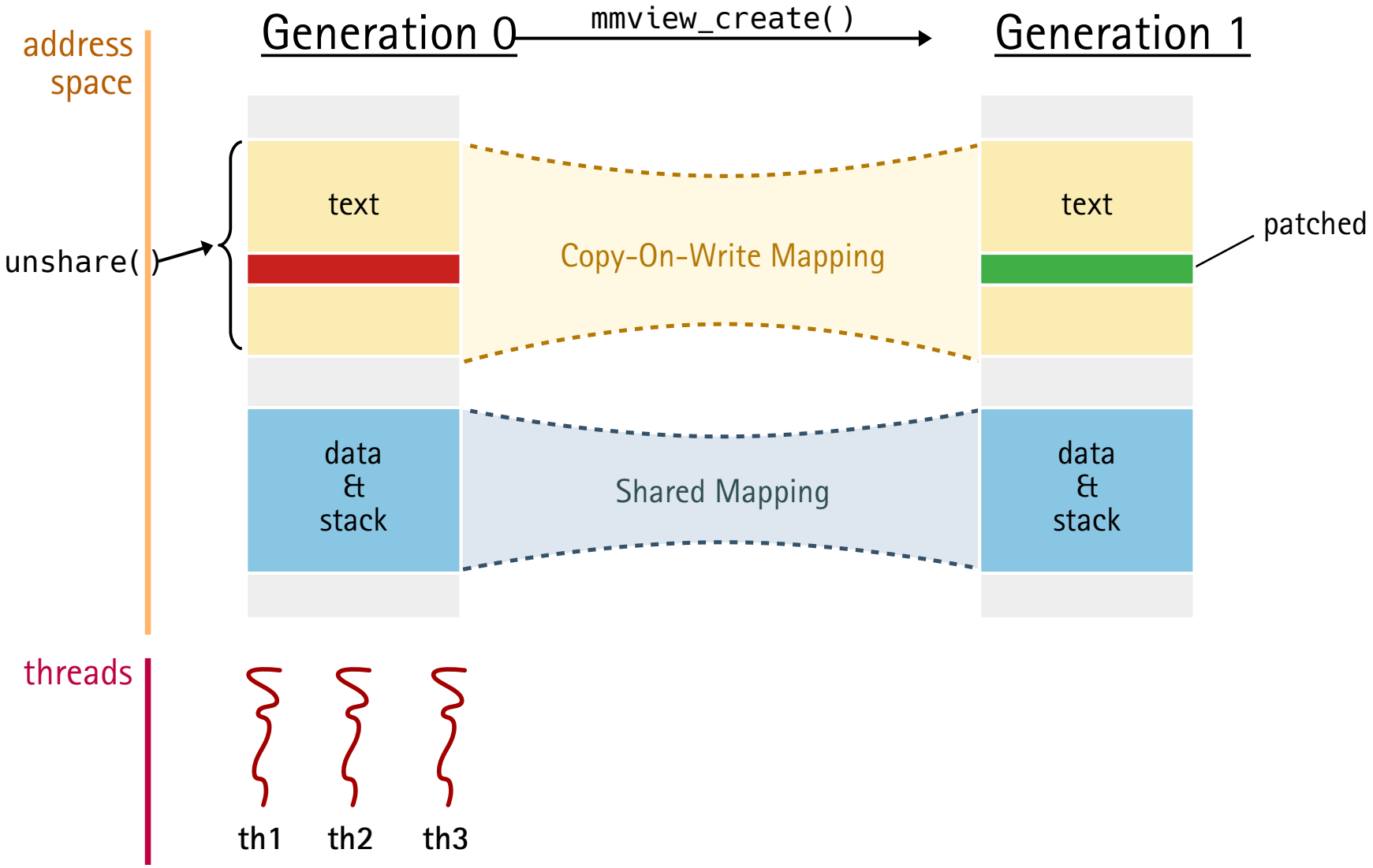
Address-Space Generations



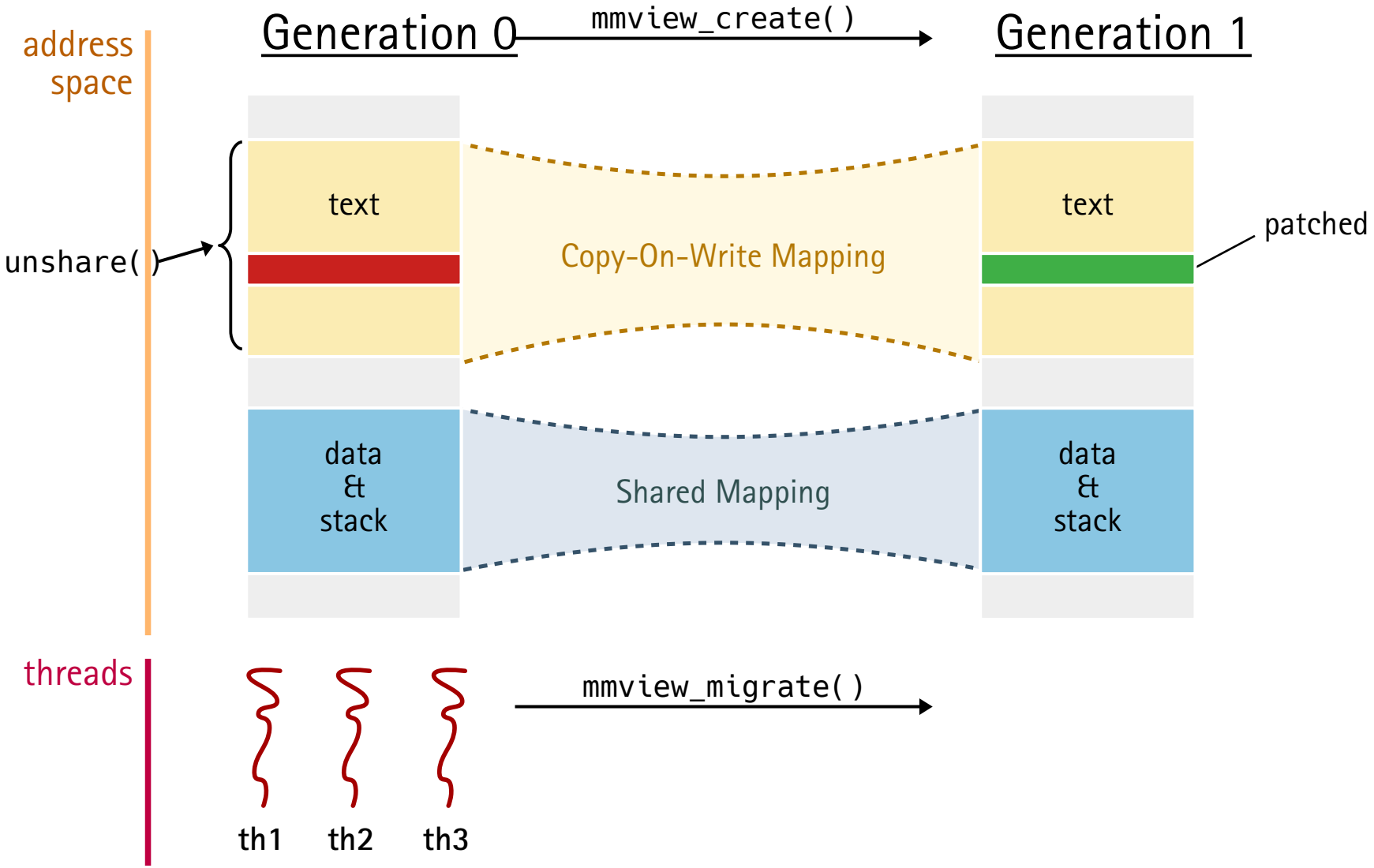
Address-Space Generations



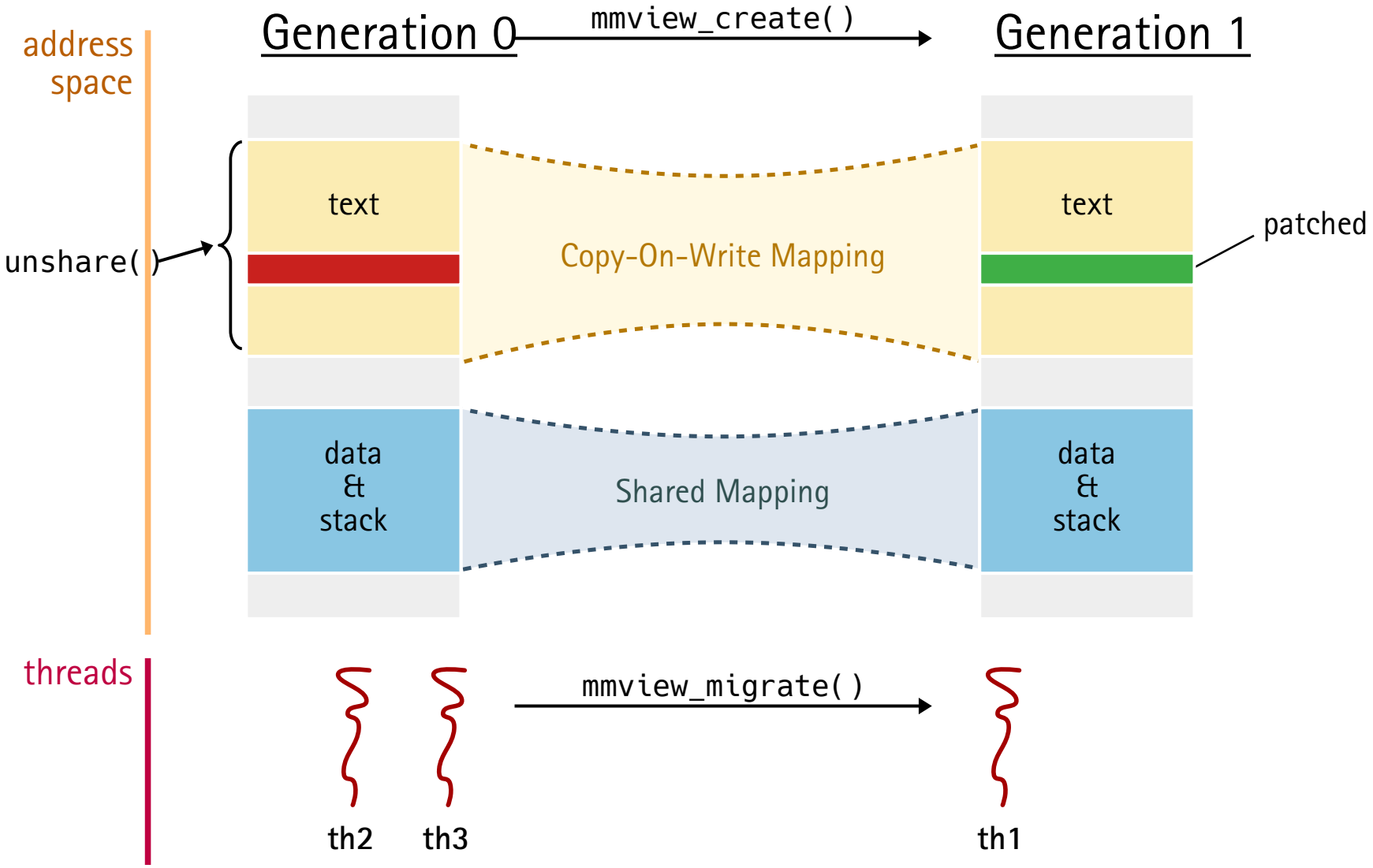
Address-Space Generations



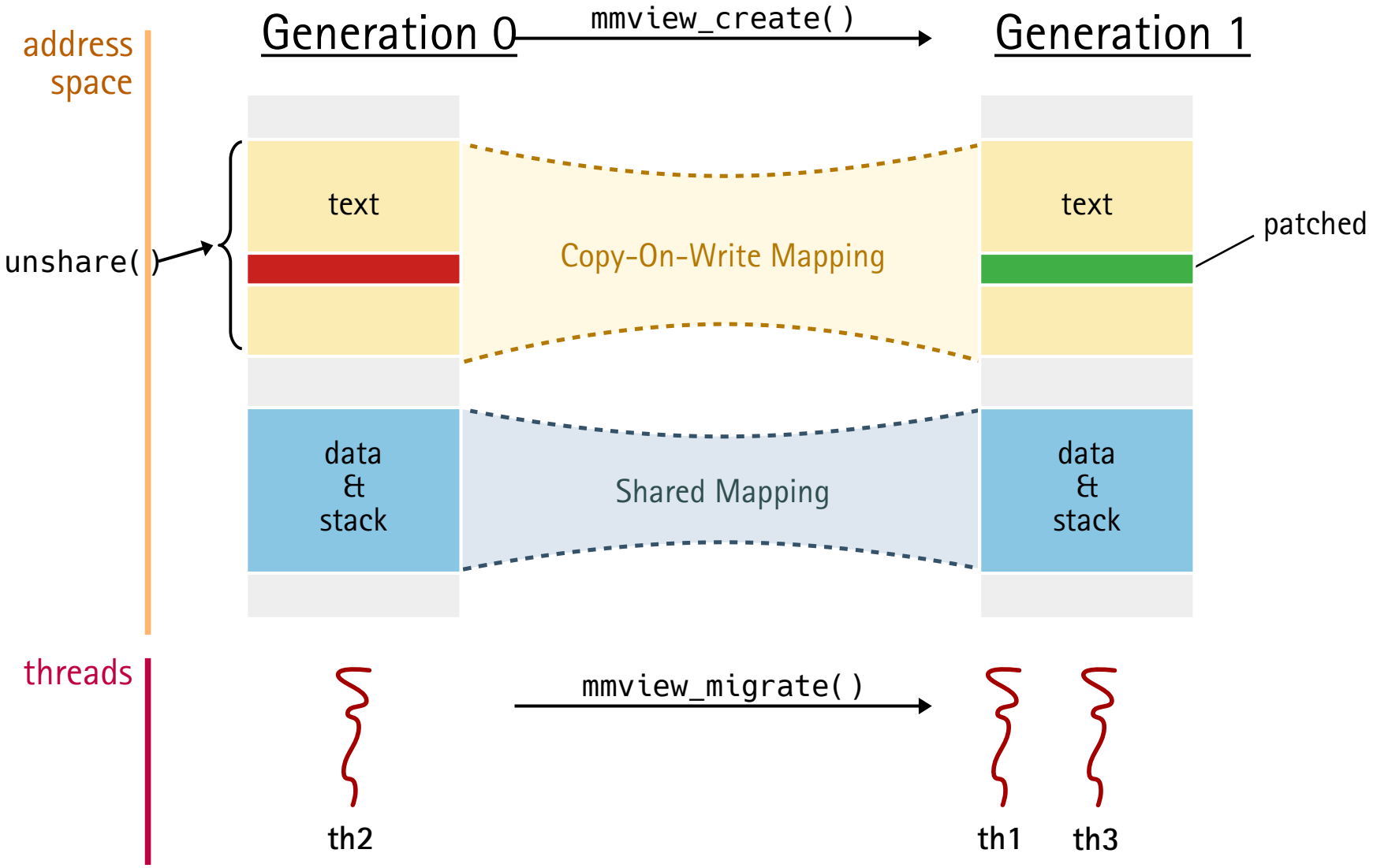
Address-Space Generations



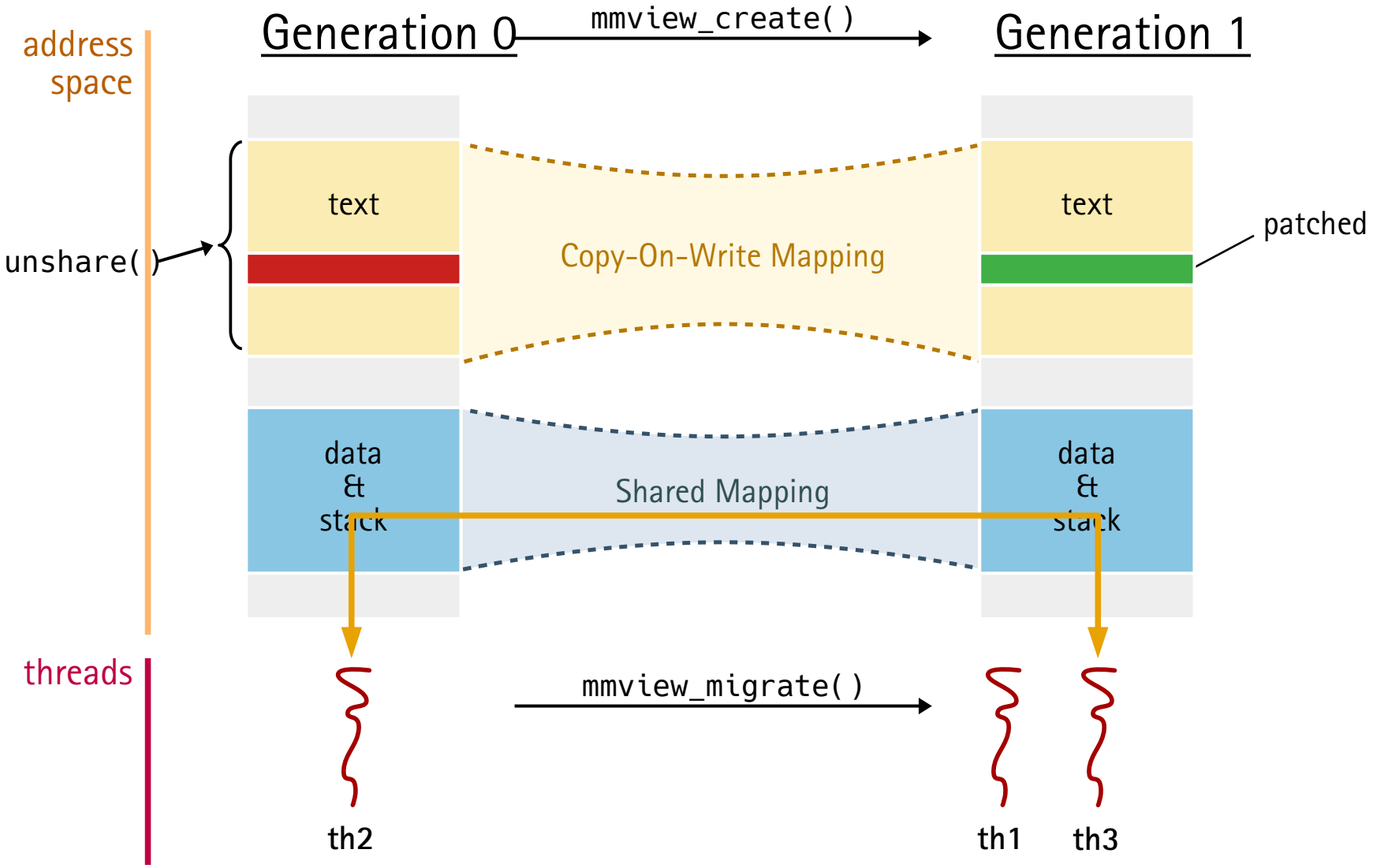
Address-Space Generations



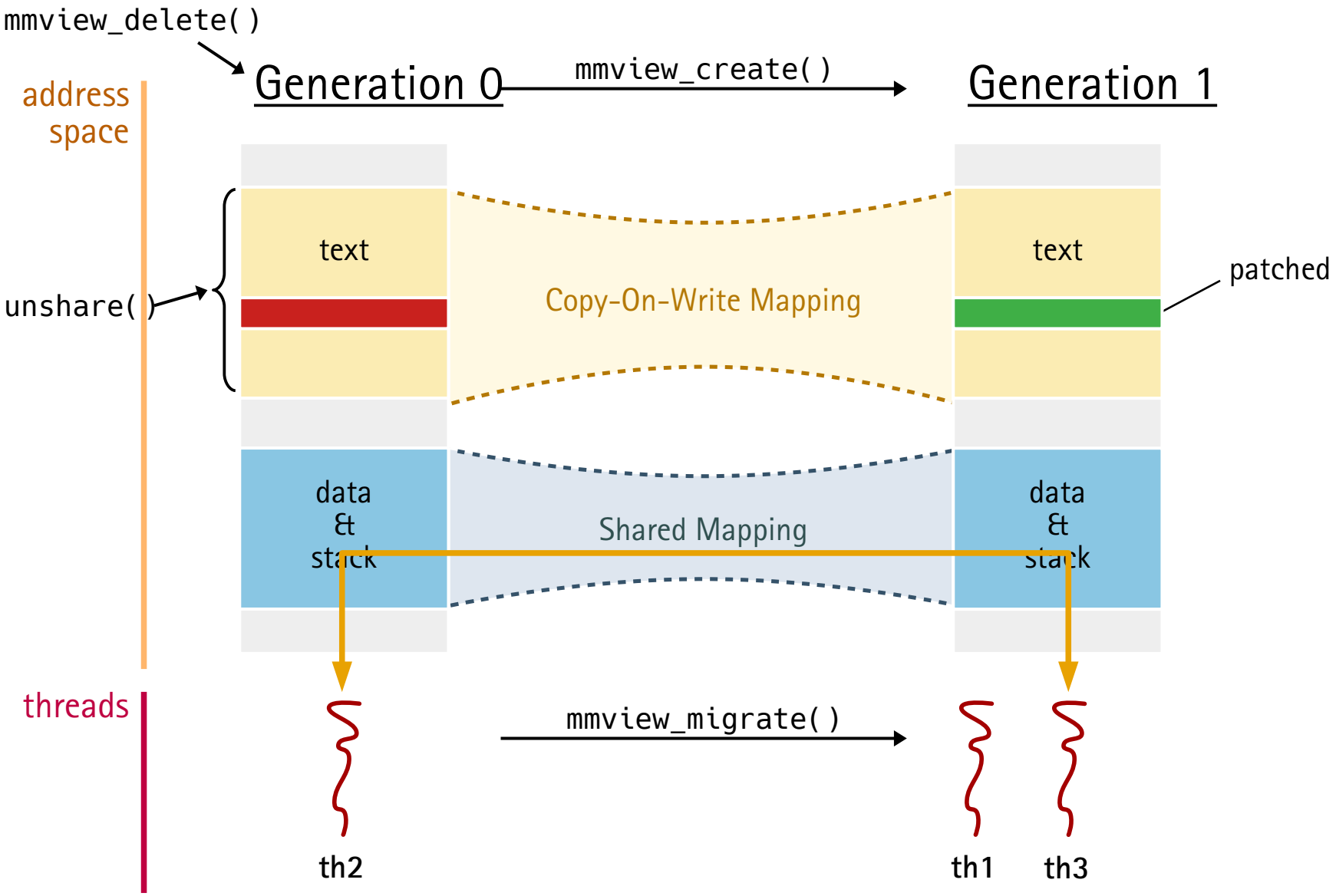
Address-Space Generations



Address-Space Generations



Address-Space Generations



Example: OpenLDAP

`do_work()`

```
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();
 }
}
```

```
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

Example: 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

Example: 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

Example: 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()

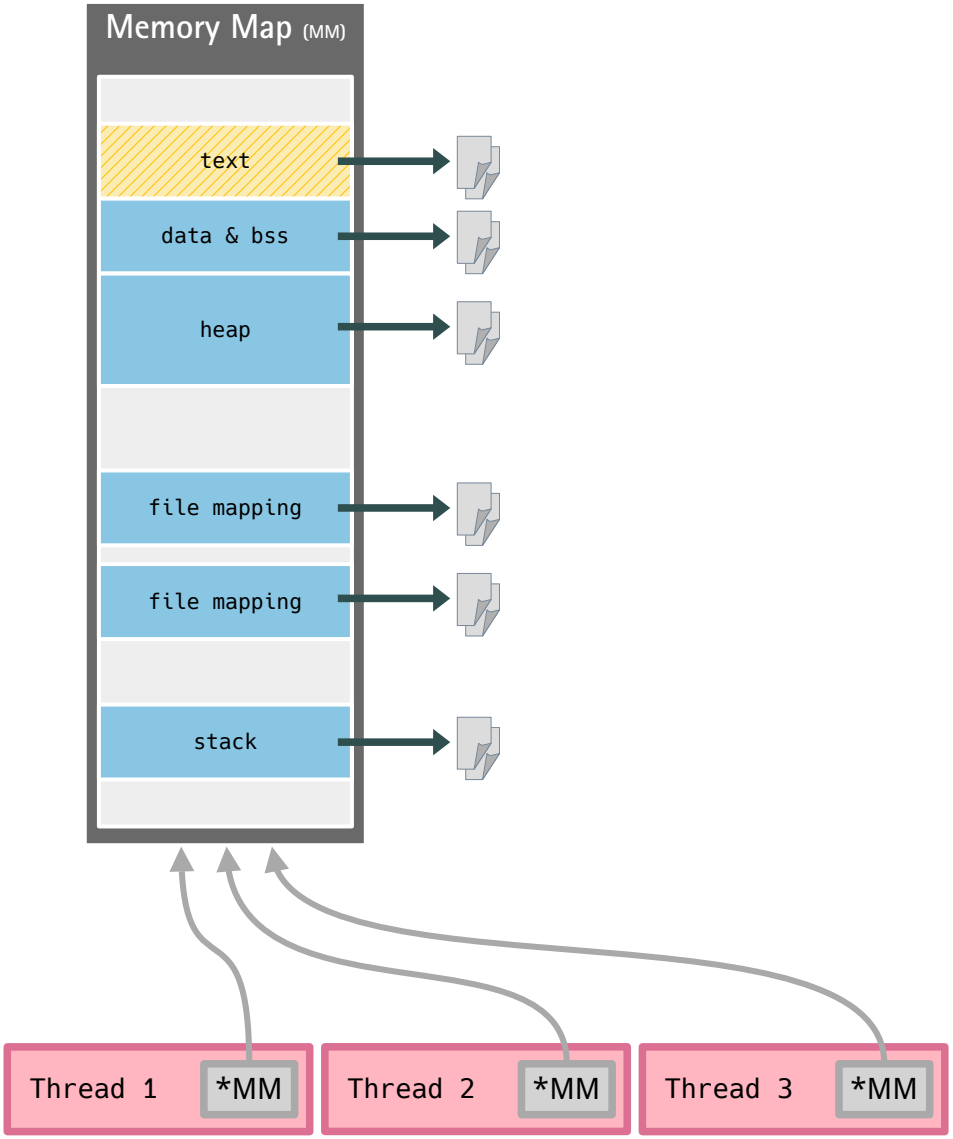
```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  op->o_tmpfree( ros->mapped_attrs, op->o_tmpmemctx );
  filter_free_x( op, op->ors_filter, 1 );
  op->o_tmpfree( op->ors_filterstr.bv_val, op->o_tmpmemctx );
  op->ors_attrs = ros->ors_attrs;
  op->ors_filter = ros->ors_filter;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

buggy

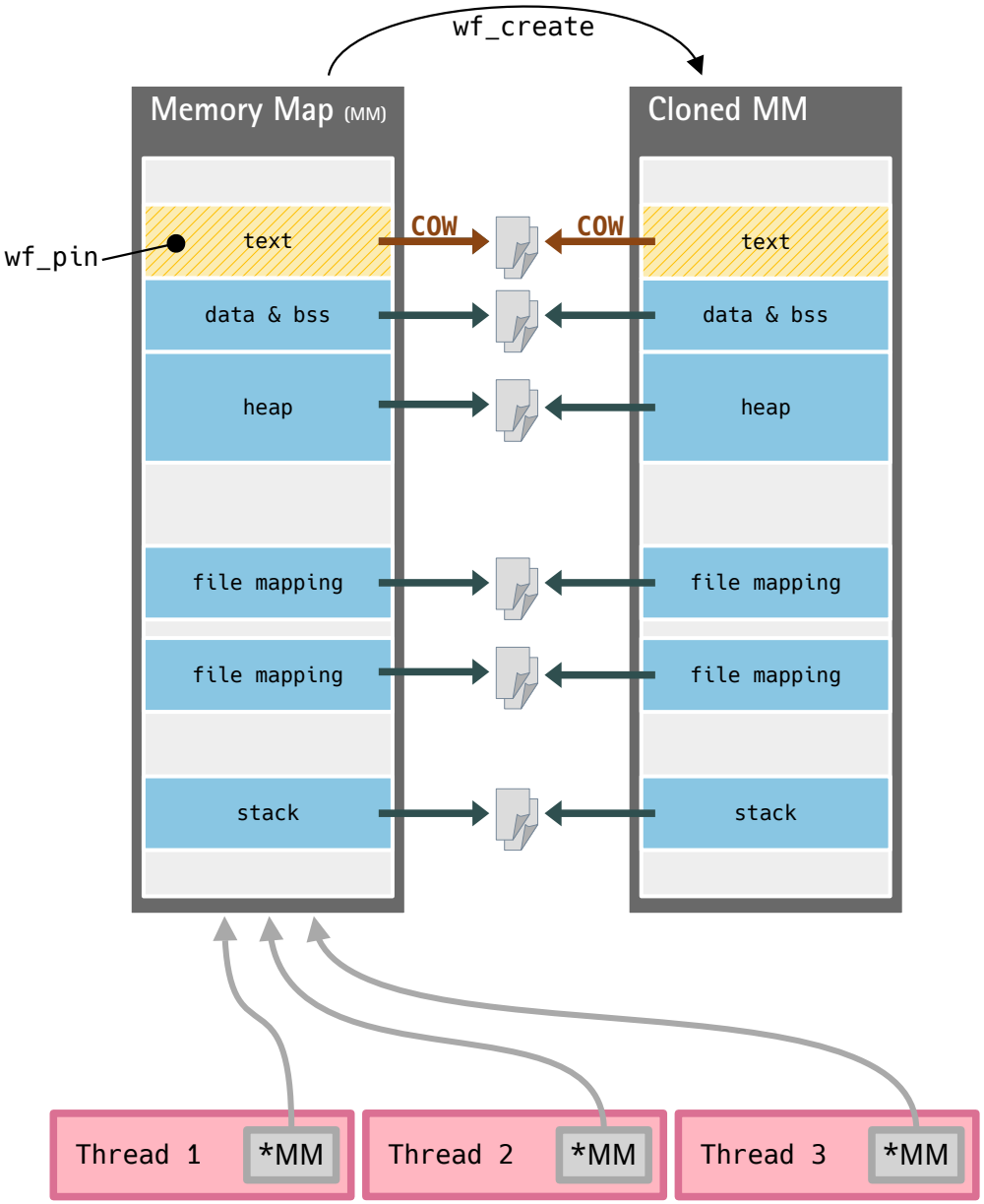
```
void rwm_op_rollback( Operation *op, SlapReply *rs, rwm_op_state *ros ) {
  ...
  if ( op->ors_filter != ros->ors_filter ) {
    filter_free_x( op, op->ors_filter, 1 );
    op->ors_filter = ros->ors_filter;
  }
  if ( op->ors_filterstr.bv_val != ros->ors_filterstr.bv_val ) {
    op->o_tmpfree( op->ors_filterstr.bv_val,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

patched

Implementation in the Linux Kernel

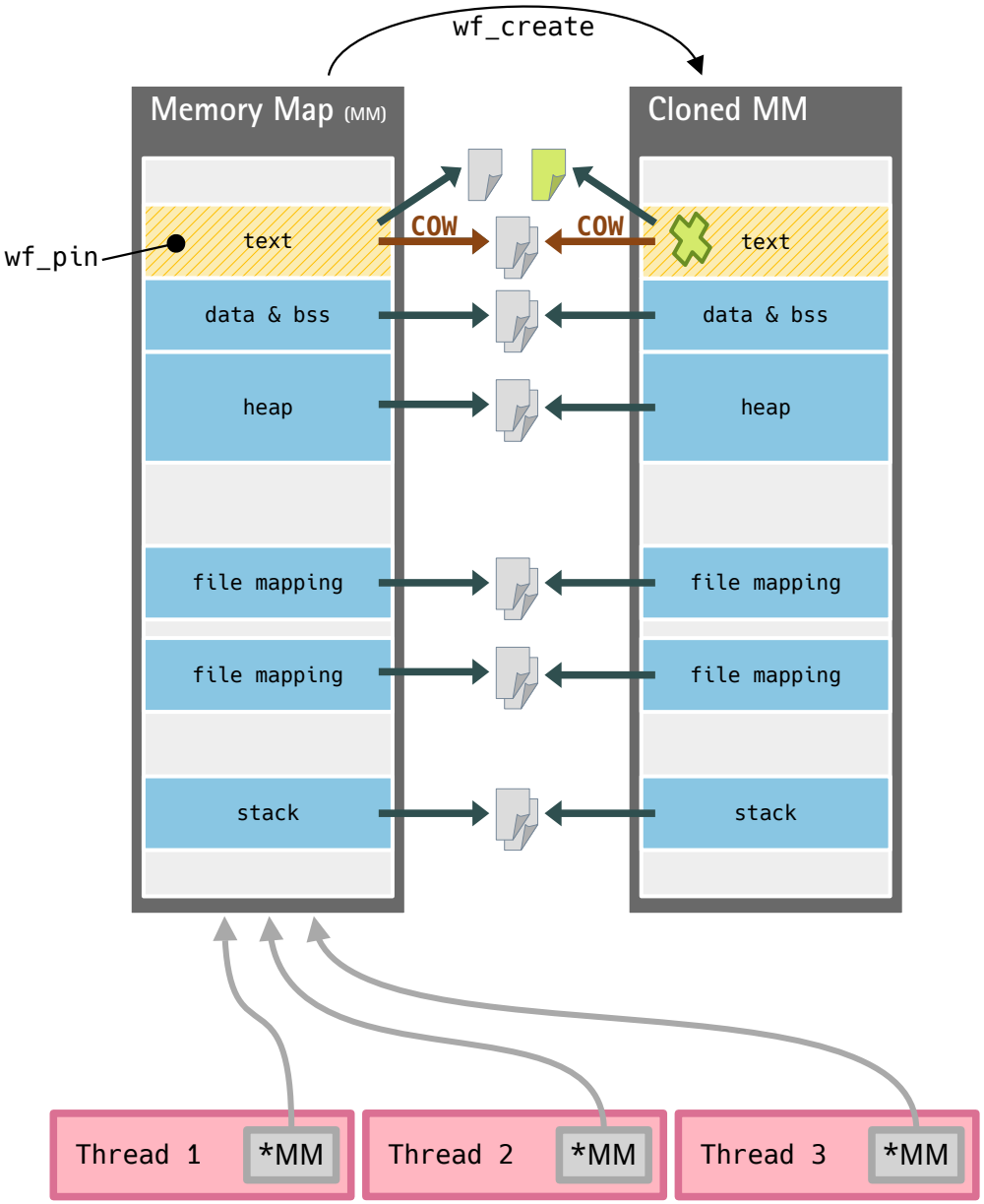


Implementation in the Linux Kernel



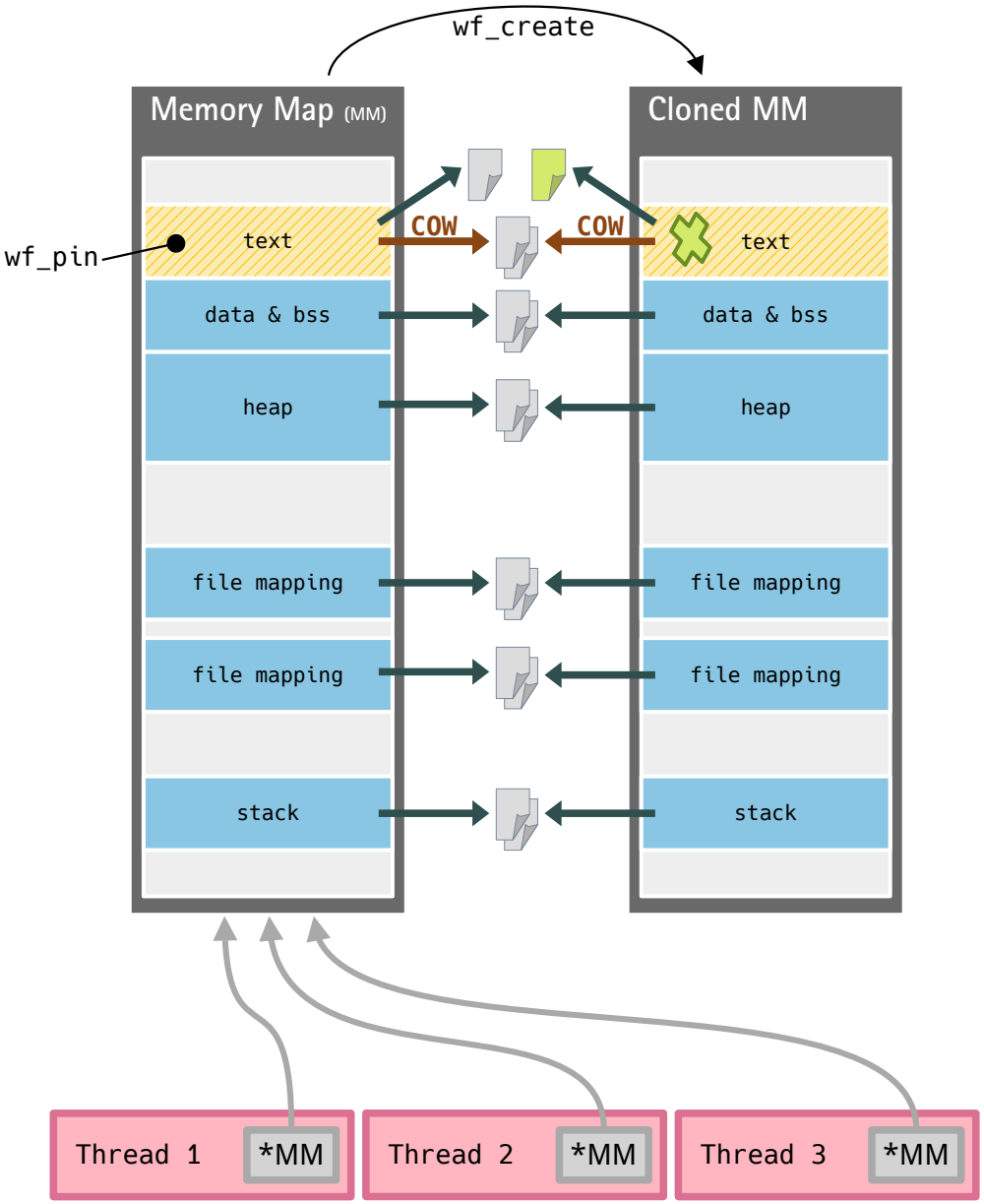
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW

Implementation in the Linux Kernel



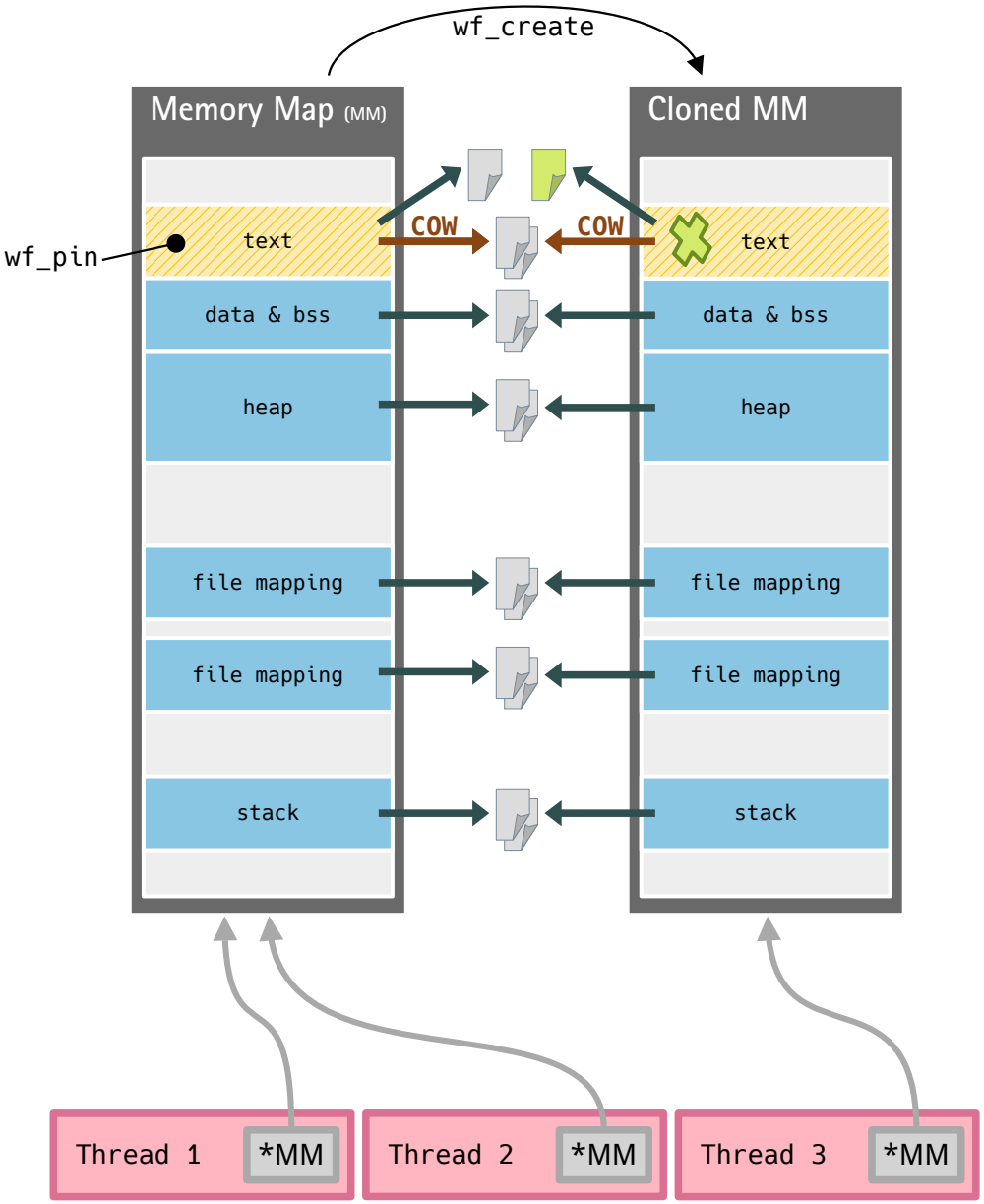
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW

Implementation in the Linux Kernel



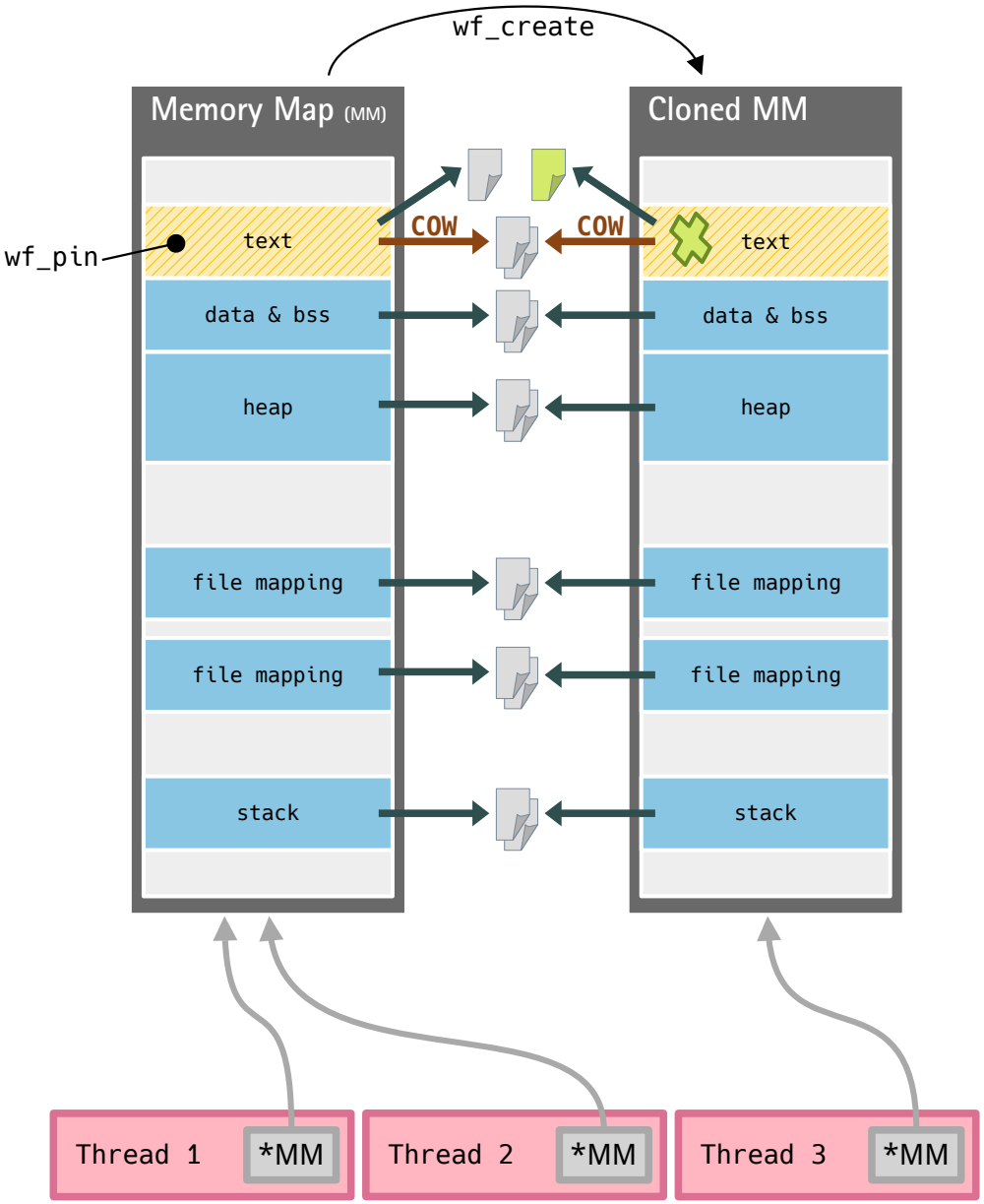
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch

Implementation in the Linux Kernel



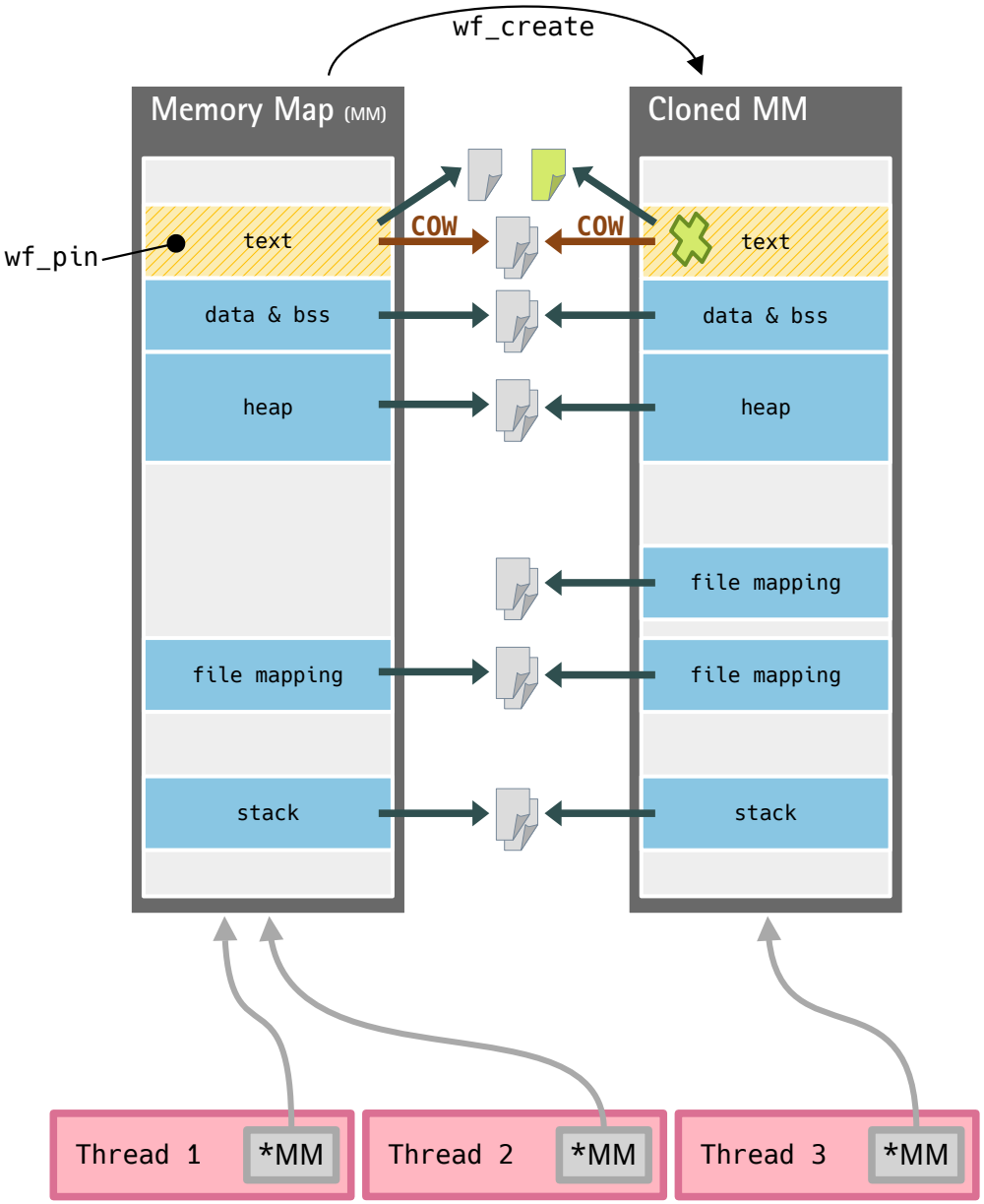
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch

Implementation in the Linux Kernel



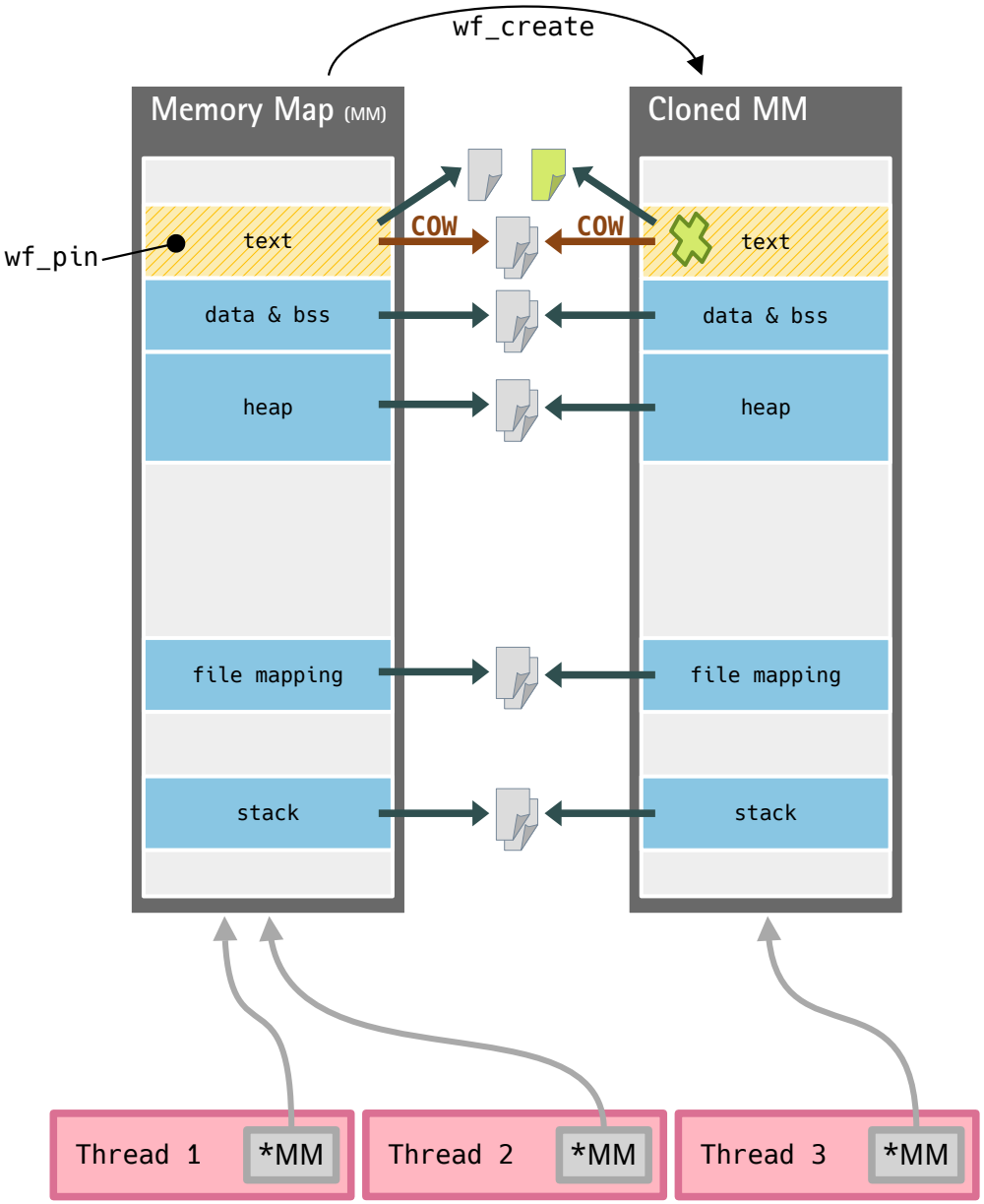
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



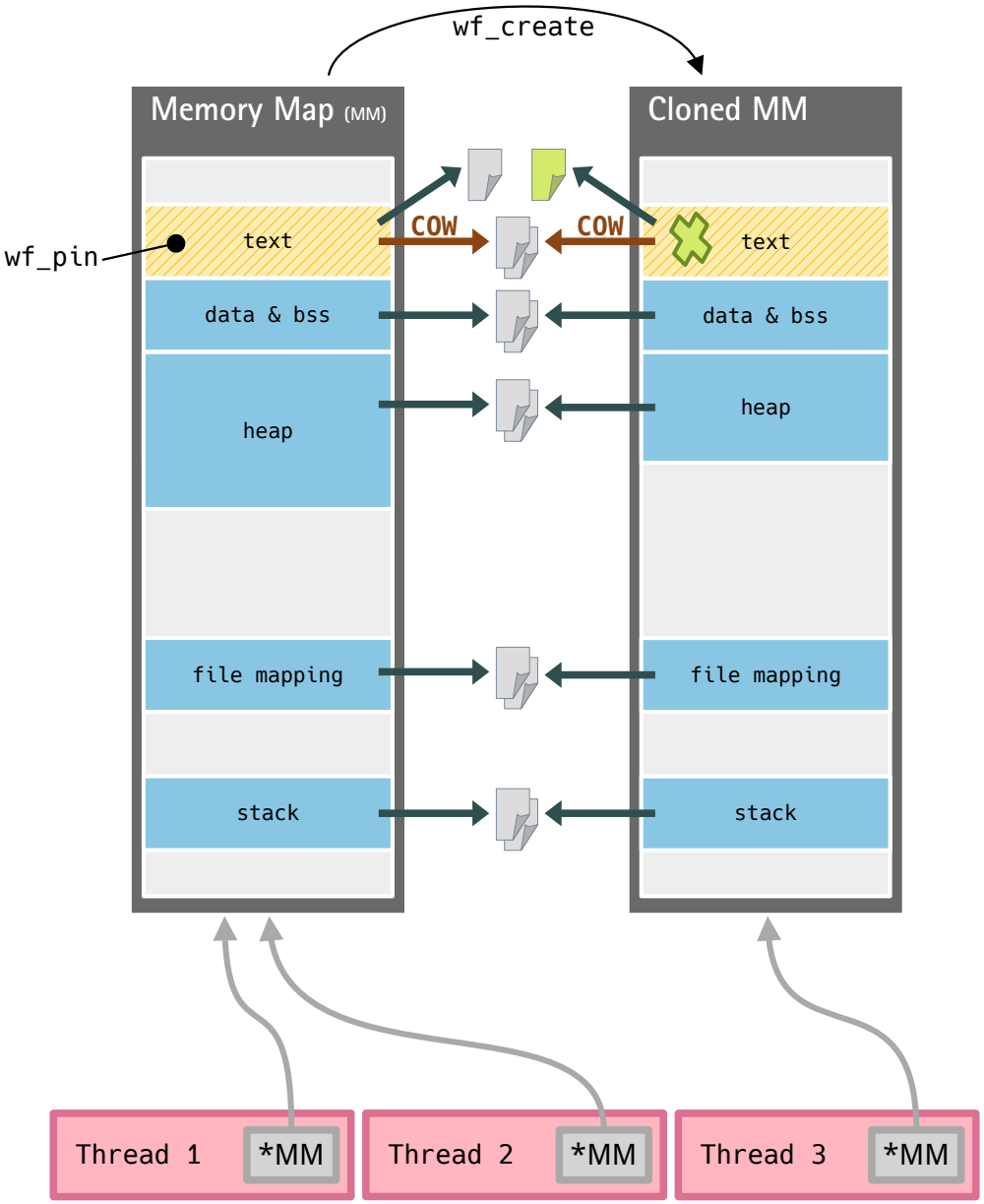
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



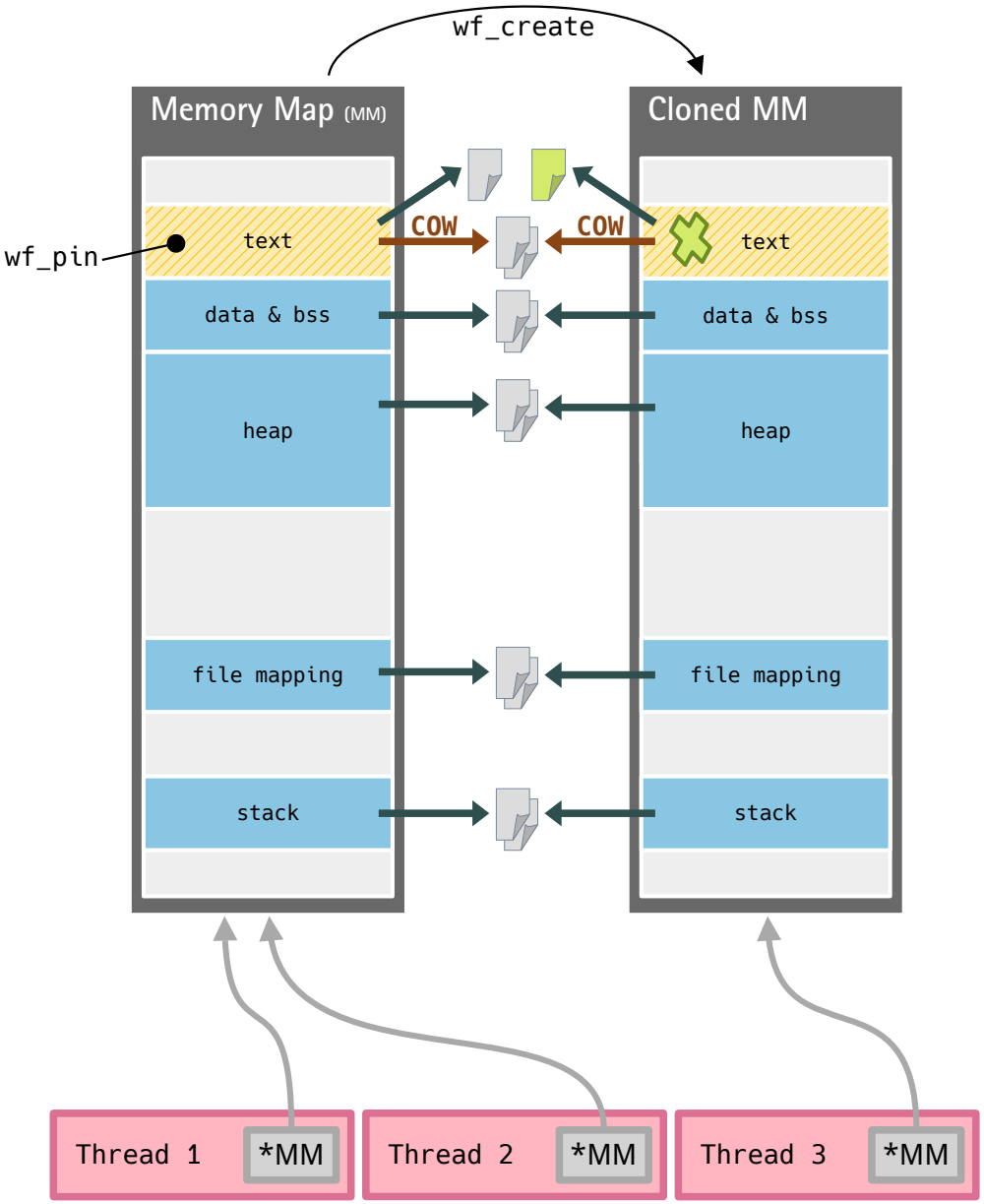
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



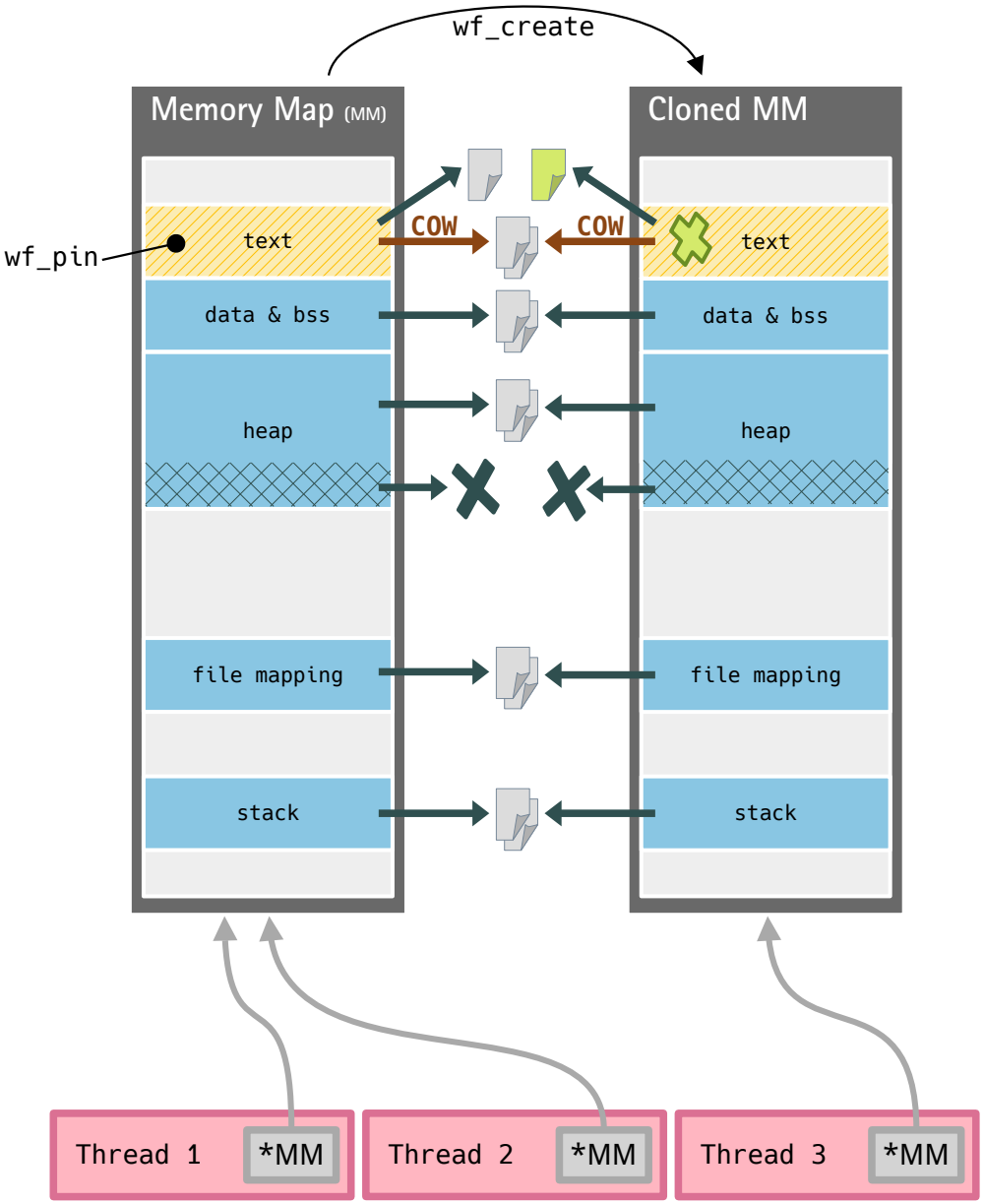
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



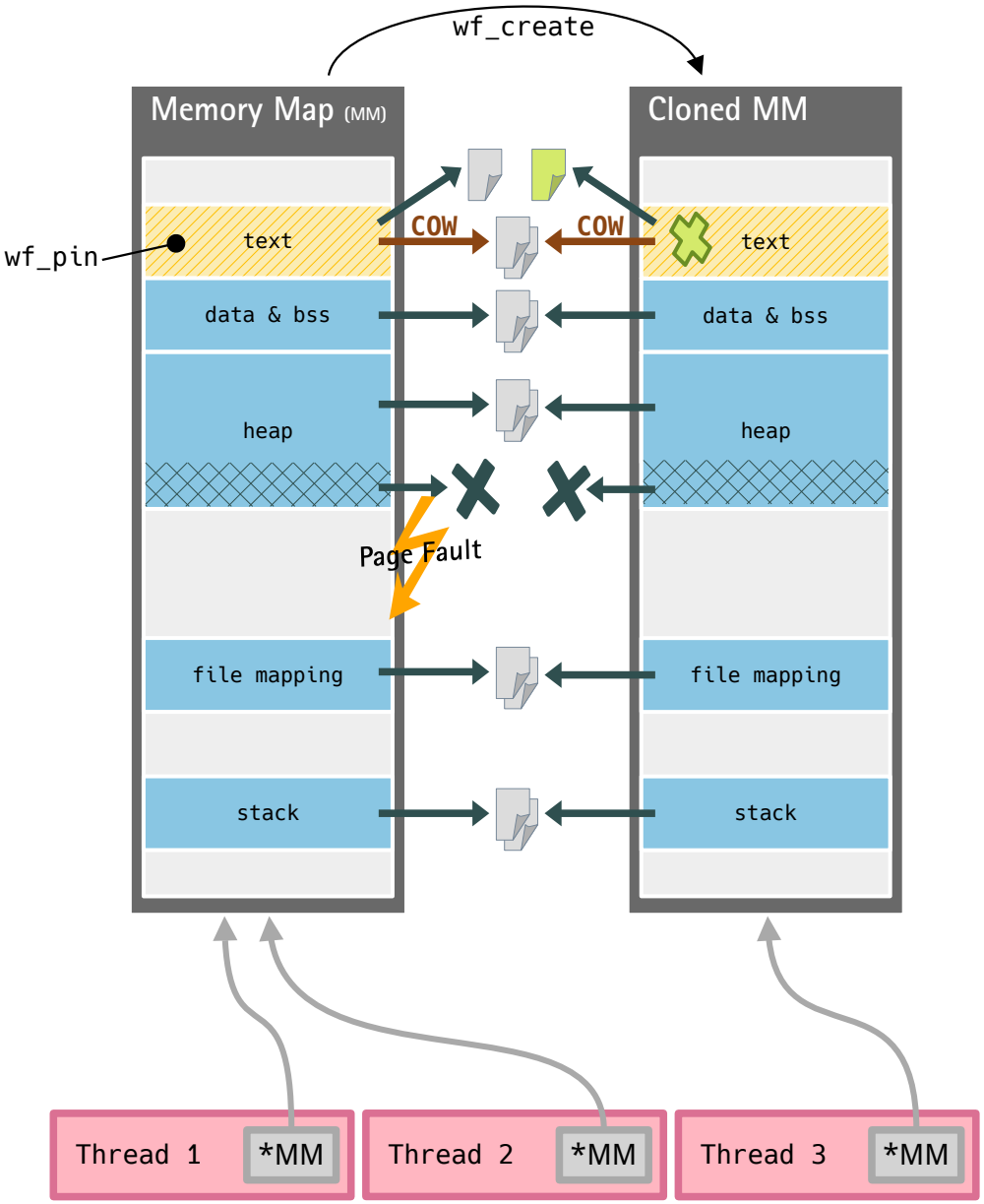
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



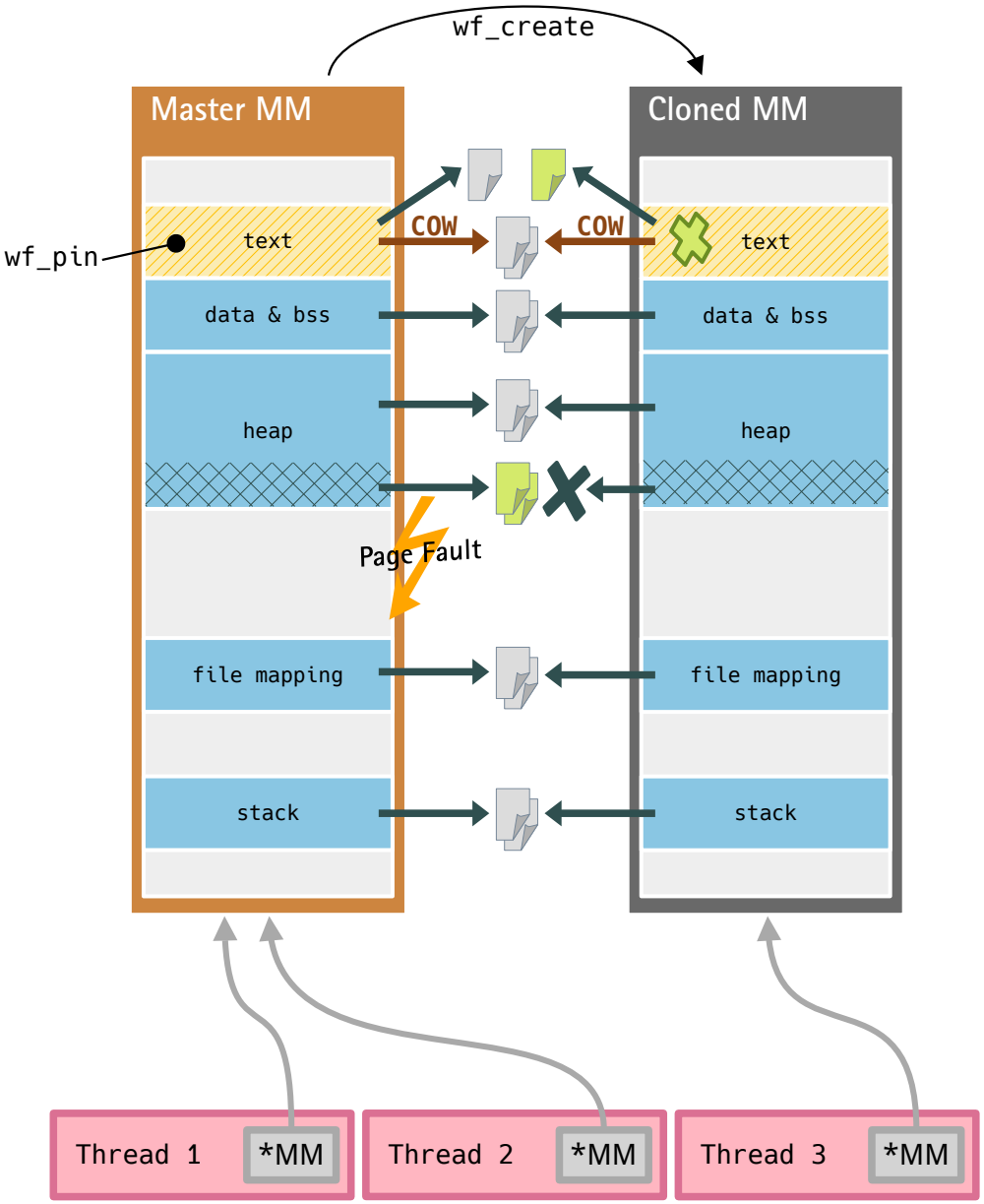
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



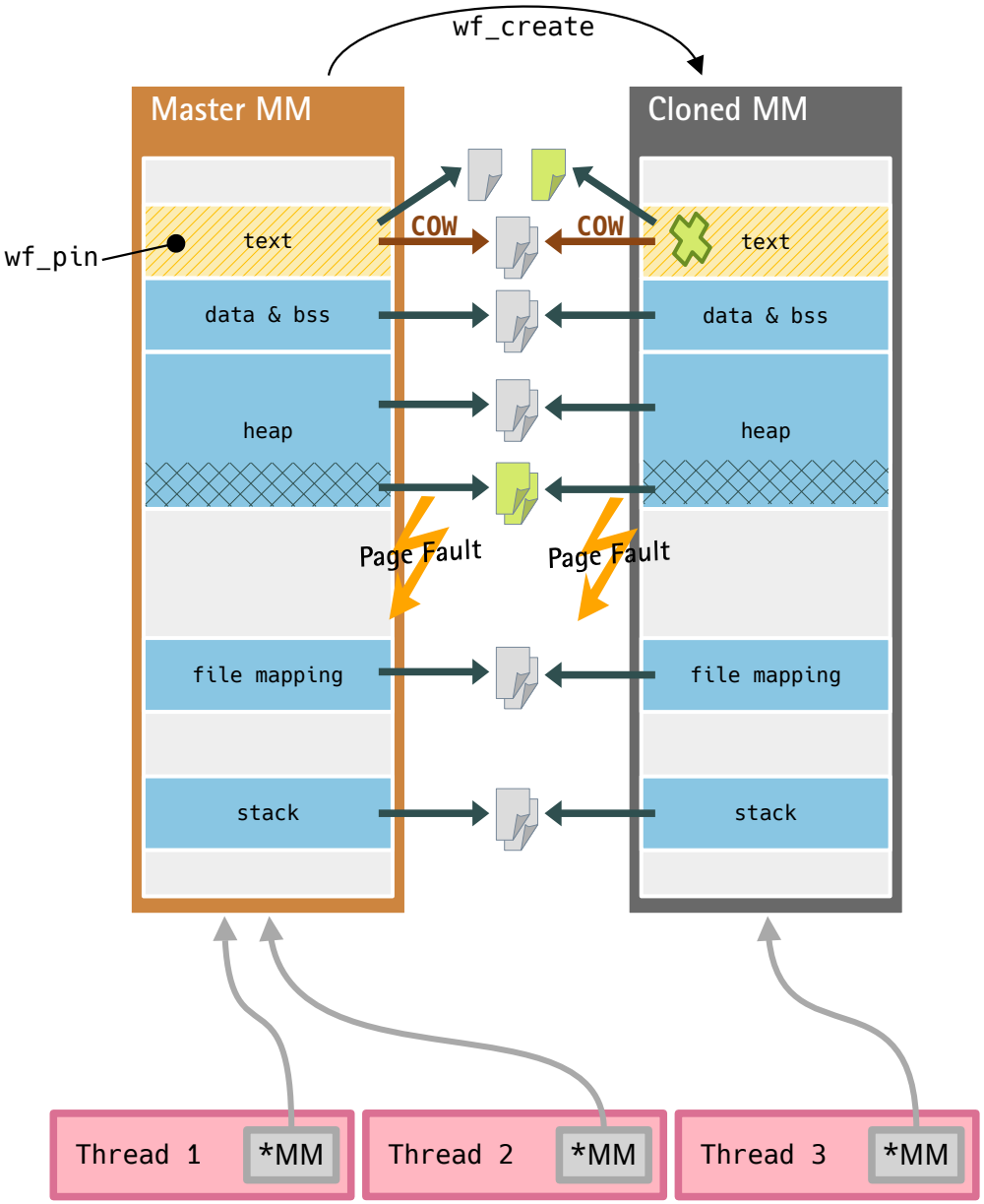
- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs

Implementation in the Linux Kernel



- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs
  - Master MM:
    Lazy page initialization,
    Locking proxy

Implementation in the Linux Kernel



- **wf_create**
  - Clones the memory map (MM) = AS generation like `fork()` but without COW
  - However, pinned mappings use COW
- **wf_migrate**
  - Changes the thread's MM pointer
  - Context switch
- **Mapping Changes**
  - Synchronized on all MMs
  - Master MM:
    Lazy page initialization,
    Locking proxy

Evaluation: Patches

Debian 10.0 packages and Debian patches (except MariaDB)

	OpenLDAP	Apache	Memcached	Samba	MariaDB	Node.js
Patches (CVE)	13 (2)	10 (10)	1 (1)	2 (2)	74 (26)	4 (0)

Evaluation: Patches

Debian 10.0 packages and Debian patches (except MariaDB)

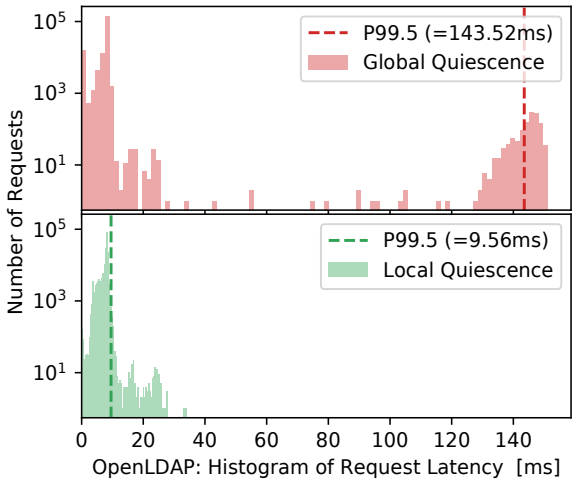
	OpenLDAP	Apache	Memcached	Samba	MariaDB	Node.js
Patches (CVE)	13 (2)	10 (10)	1 (1)	2 (2)	74 (26)	4 (0)
↓ **Restrict to code-only patches**					87% (88%)	
text-only	9 (2)	7 (7)	1 (1)	2 (2)	67 (24)	4 (0)

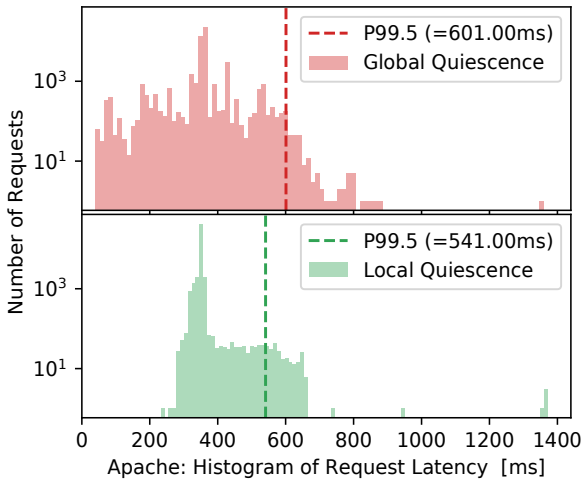
Evaluation: Patches

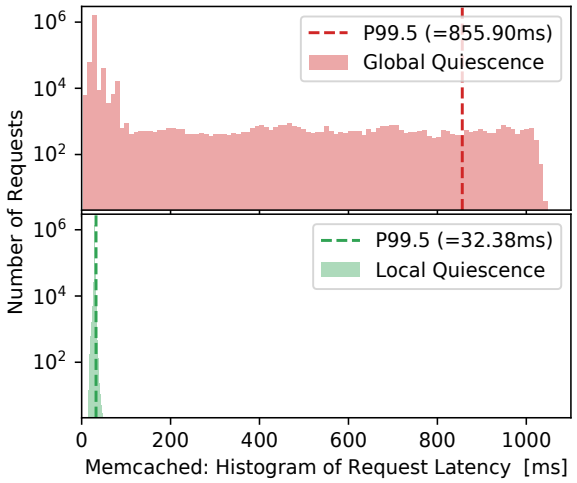
Debian 10.0 packages and Debian patches (except MariaDB)

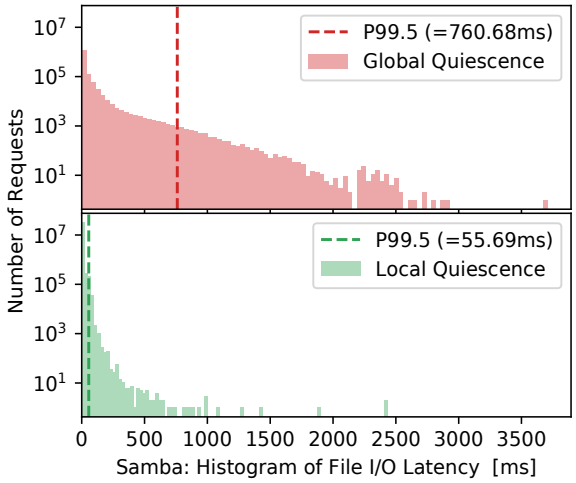
	OpenLDAP	Apache	Memcached	Samba	MariaDB	Node.js
Patches (CVE)	13 (2)	10 (10)	1 (1)	2 (2)	74 (26)	4 (0)
↓ **Restrict to code-only patches**					87% (88%)	
text-only	9 (2)	7 (7)	1 (1)	2 (2)	67 (24)	4 (0)
↓ **Generate patches via Kpatch**					39% (47%)	
kpatch'able	9 (2)	7 (7)	1 (1)	2 (2)	16 (5)	0 (0)

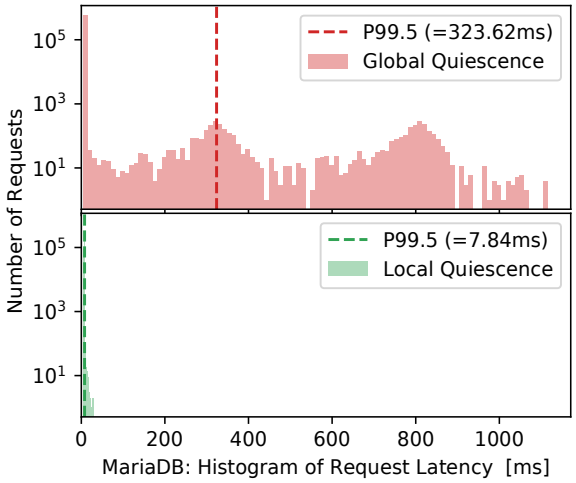
Evaluation: Request Latencies

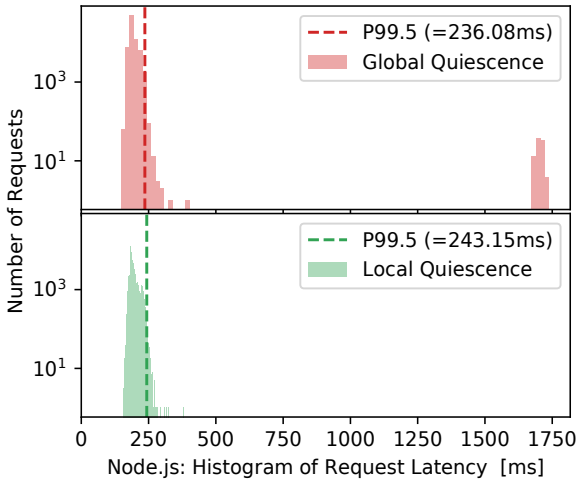




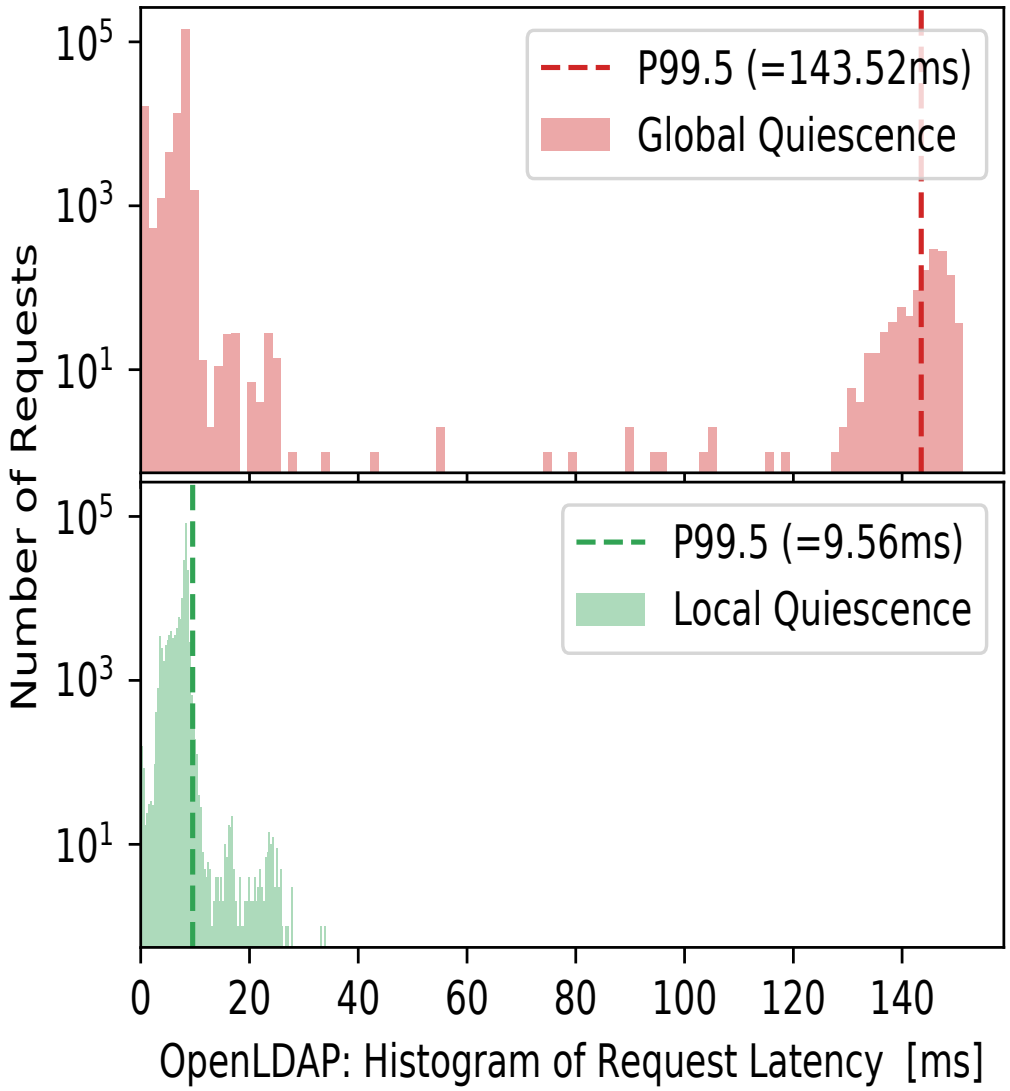




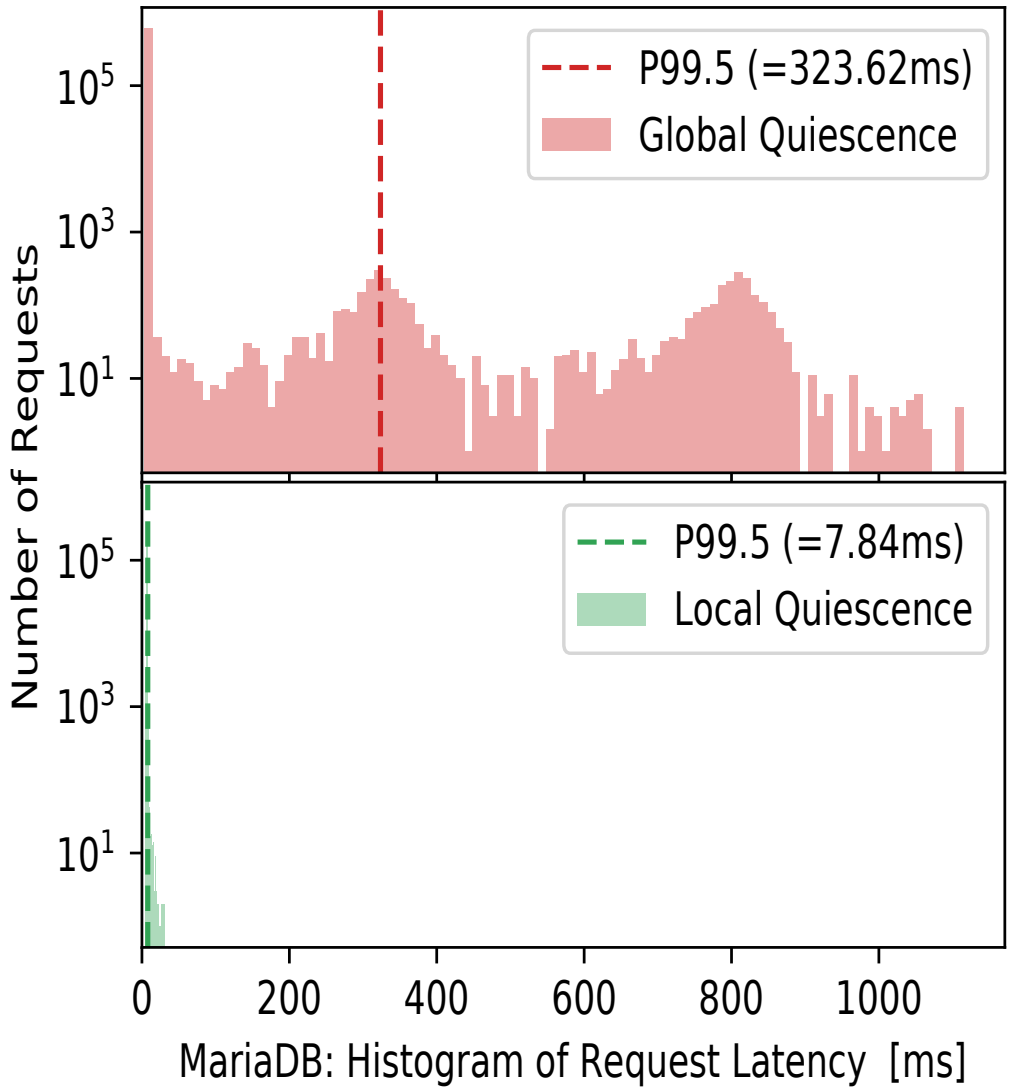




Evaluation: Request Latencies



Evaluation: Request Latencies



Evaluation: Overhead

- Run Time (microbenchmarks under load):
  - wf_create

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

    5±5 µs (Samba) to **8±7 µs** (Node.js)

Evaluation: Overhead

- Run Time (microbenchmarks under load):
  - wf_create

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

    5±5 µs (Samba) to **8±7 µs** (Node.js)

- Memory (under load):

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

Future Work

Basic mechanism

Synchronized address-space clones with partial differences

Future Work

Basic mechanism
Synchronized address-space clones with partial differences

Possible applications

- Combination with JIT compiler
- Path-specific kernel modification (→ Synthesis)
- Implementation of dynamic variability (→ Multiverse)
- Address-space views for Data (thread isolation)

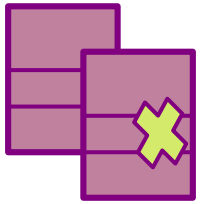
Future Work

Basic mechanism
Synchronized address-space clones with partial differences

Possible applications

- Combination with JIT compiler
- Path-specific kernel modification (→ Synthesis)
- Implementation of dynamic variability (→ Multiverse)
- Address-space views for Data (thread isolation)

Conclusion



Wait-Free Code Patching
via Address-Space Generations

- Goal: Reduce global quiescence to local quiescence
  - Easier to establish
  - Maintain quality of service
- Approach: Synchronized address-space clones
- Evaluation: 6 server applications
  - Successful application of code-only patches
  - Improved tail latencies during patching

Thank you for your attention.

Try it:
https://www.sra.uni-hannover.de/p/wfpatch

rommel@sra.uni-hannover.de