State of Multicore OCaml

KC Sivaramakrishnan

University of Cambridge

OCaml Labs

Outline

- Overview of the multicore OCaml project
- Multicore OCaml runtime design
- Future directions

Multicore OCaml

Multicore OCaml

- Add native support for concurrency and (shared-memory) parallelism to OCaml

Multicore OCaml

- Add native support for concurrency and (shared-memory) parallelism to OCaml

- History
  - ★ Jan 2014: Initiated by Stephen Dolan and Leo White
  - ★ Sep 2014: Multicore OCaml design @ OCaml workshop
  - ★ Jan 2015: KC joins the project at OCaml Labs
  - ★ Sep 2015: Effect handlers @ OCaml workshop
  - ★ Jan 2016: Native code backend for Amd64 on Linux and OSX
  - ★ Jun 2016: Multicore rebased to 4.02.2 from 4.00.0
  - ★ Sep 2016: Reagents library, *Multicore backend for Links* @ OCaml workshop
  - ★ Apr 2017: ARM64 backend

Multicore OCaml

Multicore OCaml

- History continued…
  - ★ Jun 2017: Handlers for Concurrent System Programming @ TFP
  - ★ Sep 2017: Memory model proposal @ OCaml workshop
  - ★ Sep 2017: CPS translation for handlers @ FSCD
  - ★ Apr 2018: Multicore rebased to 4.06.1 (*will track releases going forward*)
  - ★ Jun 2018: Memory model @ PLDI

Multicore OCaml

- History continued…
  - ★ Jun 2017: Handlers for Concurrent System Programming @ TFP
  - ★ Sep 2017: Memory model proposal @ OCaml workshop
  - ★ Sep 2017: CPS translation for handlers @ FSCD
  - ★ Apr 2018: Multicore rebased to 4.06.1 (*will track releases going forward*)
  - ★ Jun 2018: Memory model @ PLDI
- Looking forward…
  - ★ Q3'18 — Q4'18: Implement missing features, upstream prerequisites to trunk
  - ★ Q1'19 — Q2'19: Submit feature-based PRs to upstream

Components

Multicore Runtime
+
Domains

Effect Handlers

Effect System

Components

Multicore Runtime + Domains	Effect Handlers	Effect System

- Multicore Runtime
    - ★ Multicore GC + Domains (creating and managing parallel threads)

Components

Multicore Runtime + Domains	Effect Handlers	Effect System

- Multicore Runtime
  - ★ Multicore GC + Domains (creating and managing parallel threads)
- Effect handlers
  - ★ *Fibers:* Runtime system support for *linear* delimited continuations

Components

Multicore Runtime
+
Domains

Effect Handlers

Effect System

- Multicore Runtime
  - ★ Multicore GC + Domains (creating and managing parallel threads)
- Effect handlers
  - ★ *Fibers:* Runtime system support for *linear* delimited continuations
- Effect system
  - ★ Track user-defined effects in the type system
  - ★ Statically rule out the possibility of *unhandled* effects

Components

Multicore Runtime
+
Domains

Effect Handlers

Effect System

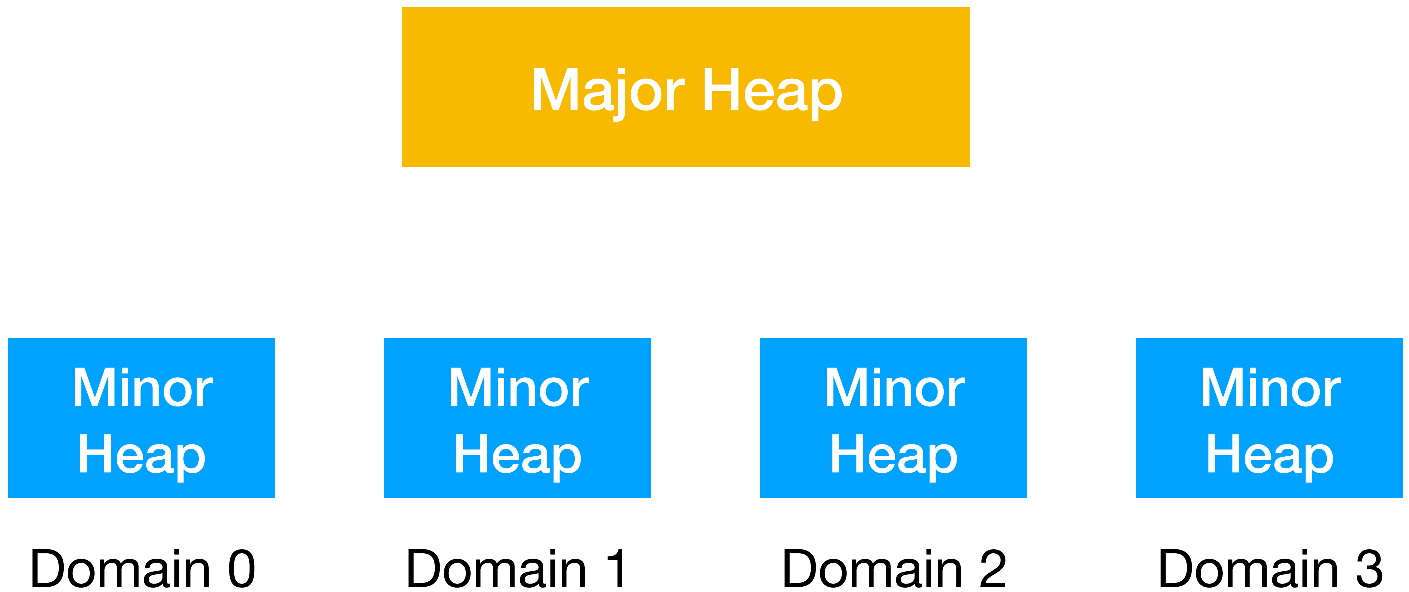
Current implementation

- Multicore Runtime
  - ★ Multicore GC + Domains (creati rallel threads)
- Effect handlers
  - ★ *Fibers:* Runtime system support for *linear* delimited continuations
- Effect system
  - ★ Track user-defined effects in the type system
  - ★ Statically rule out the possibility of *unhandled* effects

Components

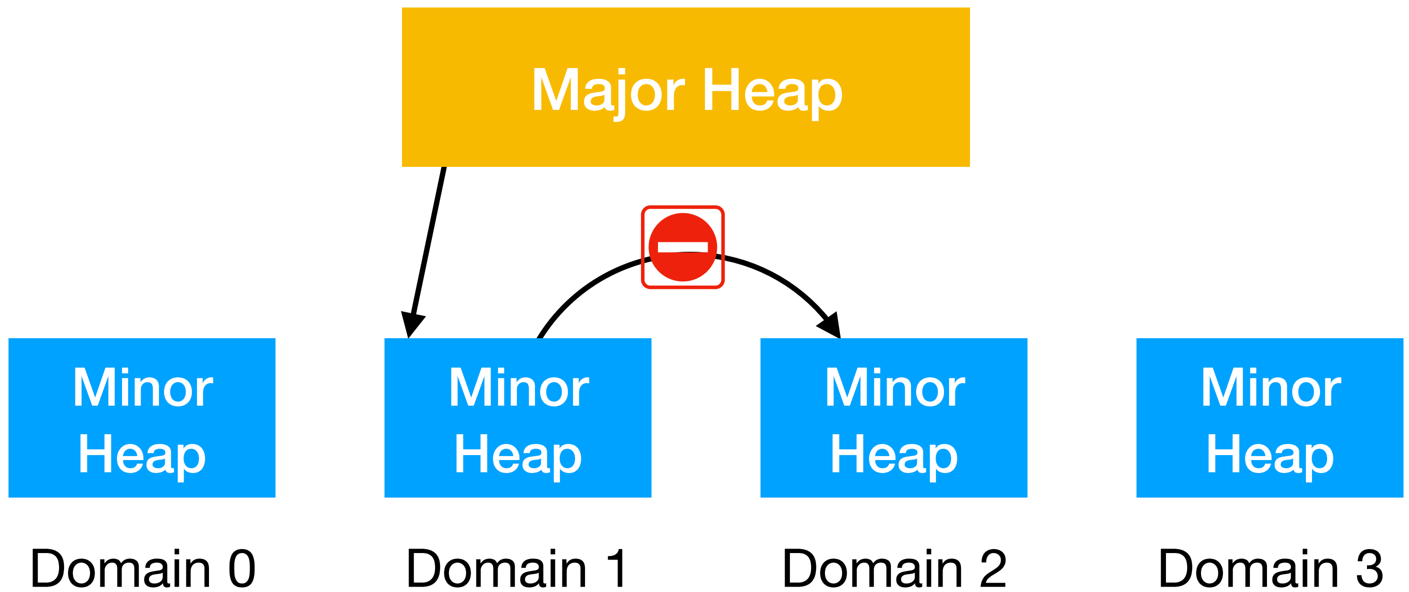
- Multicore Runtime
  - ★ Multicore GC + Domains (creati rallel threads)
- Effect handlers
  - ★ *Fibers:* Runtime system support for *linear* delimited continuations
- Effect system
  - ★ Track user-defined effects in the type system
  - ★ Statically rule out the possibility of *unhandled* effects

Multicore GC



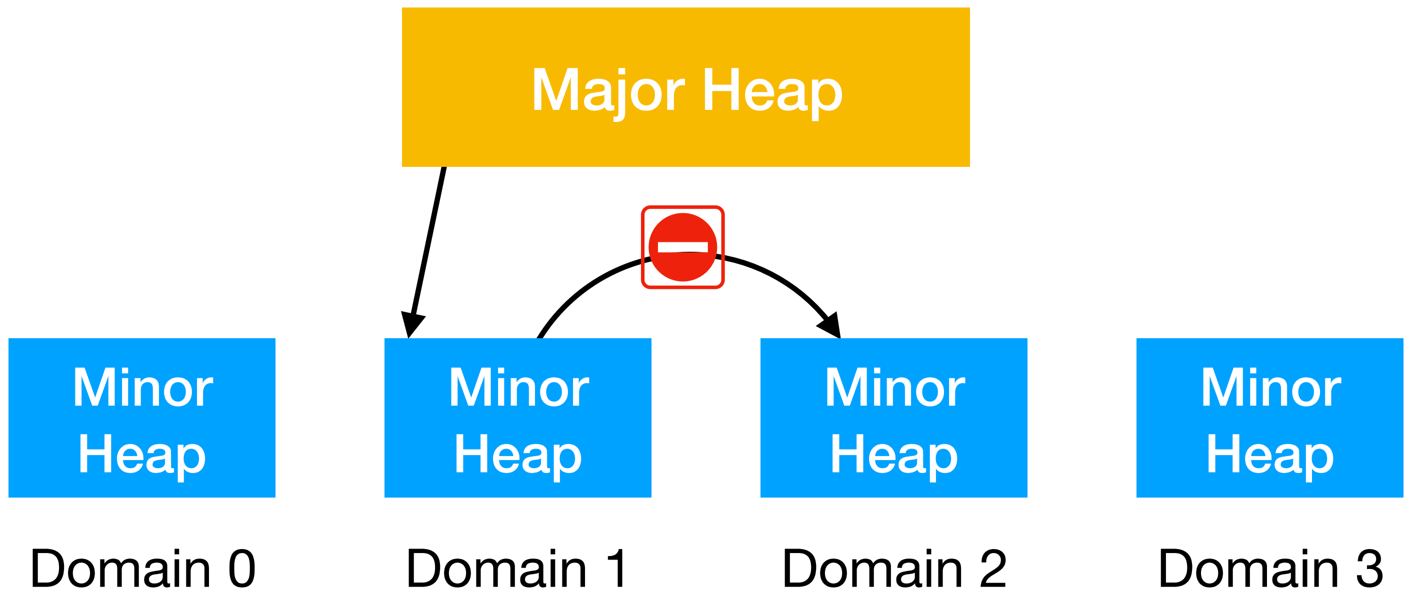
[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

Multicore GC



[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

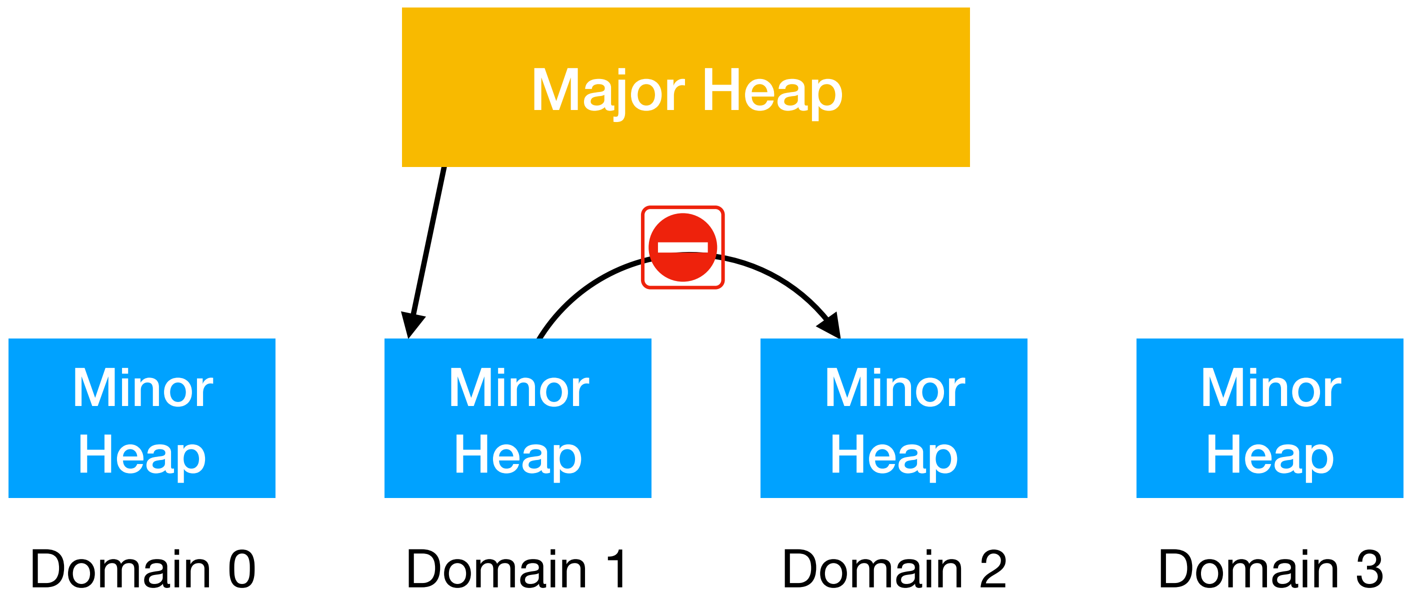
Multicore GC



- Independent per-domain minor collection

[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

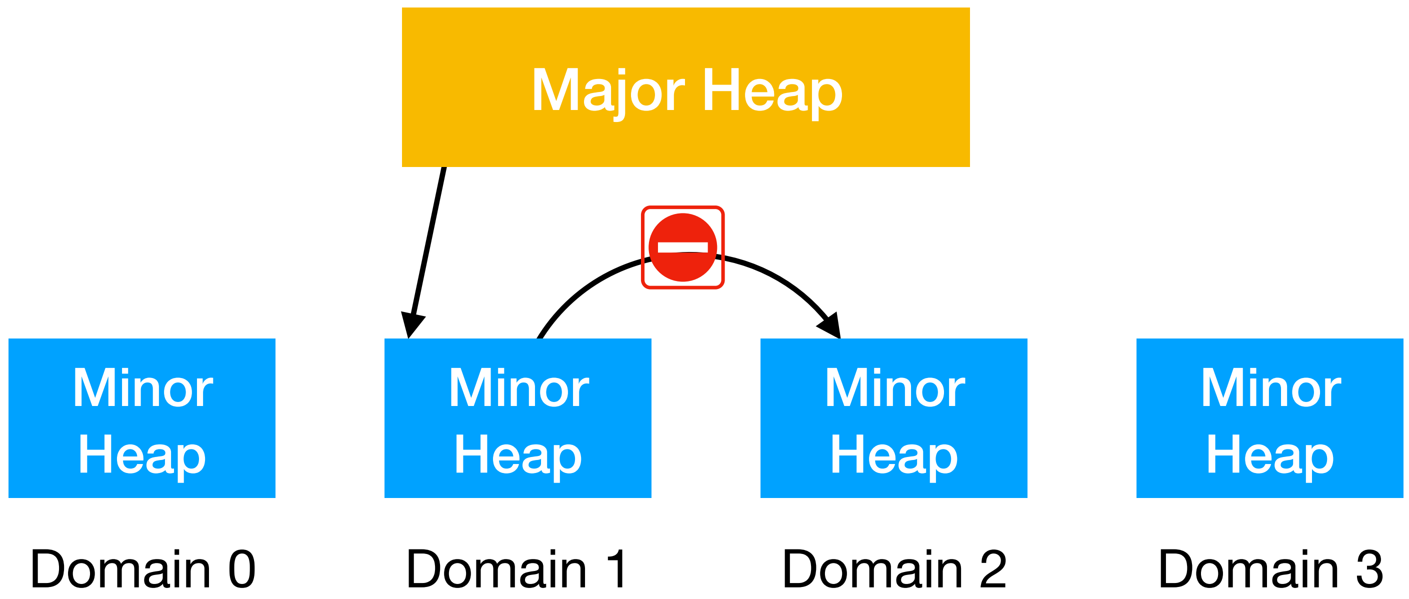
Multicore GC



- Independent per-domain minor collection
  - ★ Read barrier for mutable fields + promotion to major

[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

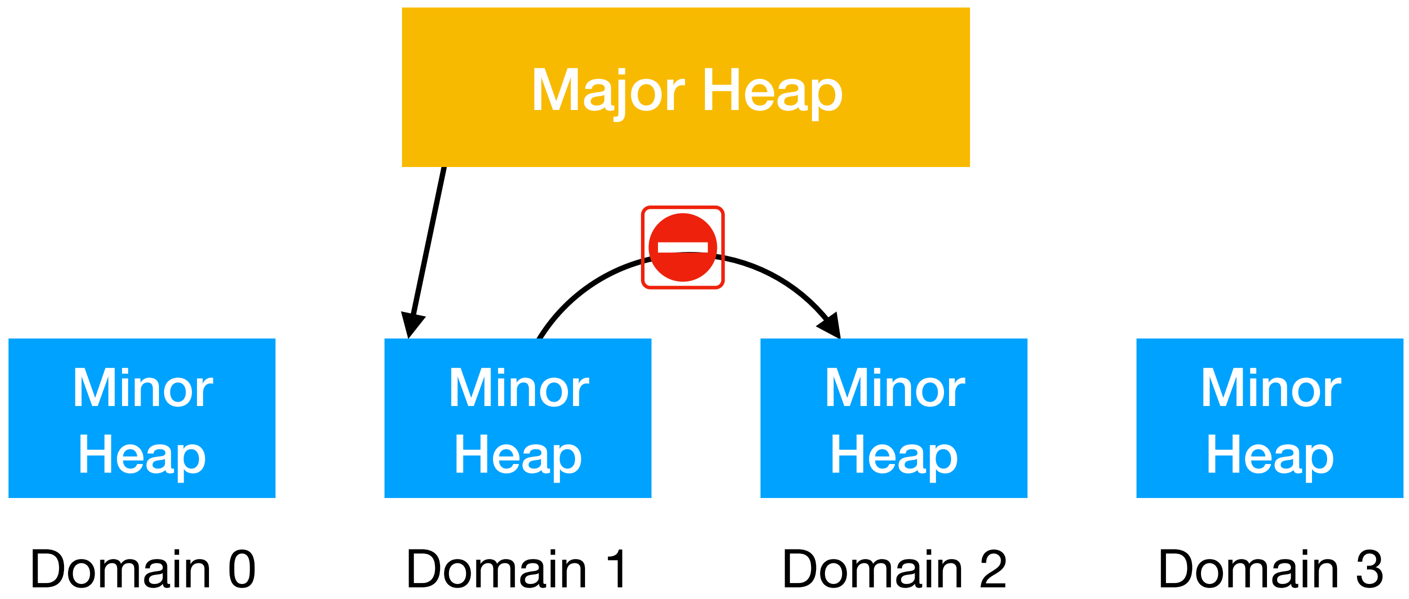
Multicore GC



- Independent per-domain minor collection
  - ★ Read barrier for mutable fields + promotion to major
- A new major allocator based on StreamFlow [1], lock-free multithreaded allocation

[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

Multicore GC



- Independent per-domain minor collection
  - ★ Read barrier for mutable fields + promotion to major
- A new major allocator based on StreamFlow [1], lock-free multithreaded allocation
- A new major GC based on VCGC [2] adapted to fibers, ephemerons, finalisers

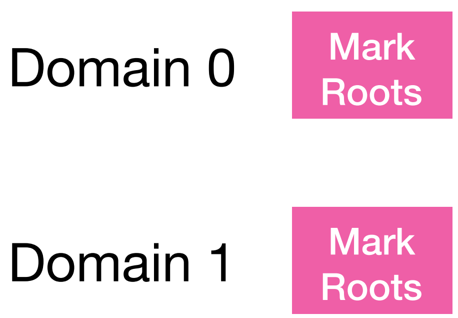
[1] Scott Schneider, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. "Scalable, locality-conscious multithreaded memory allocation." ISMM 2006.
[2] Lorenz Huelsbergen and Phil Winterbottom. "Very concurrent mark-&-sweep garbage collection without fine-grain synchronization." ISMM 1998.

Major GC

- Concurrent, incremental, mark and sweep
  - ★ Uses deletion/yuasa barrier
  - ★ Upper bound on marking work per cycle (not fixed due to weak refs)
- 3 phases:
  - ★ Sweep-and-mark-main
  - ★ Mark-final
  - ★ Sweep-ephe

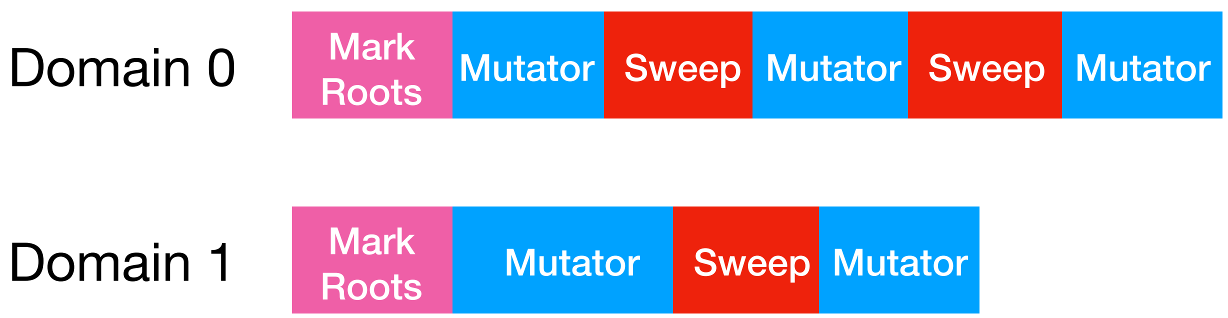
Major GC: Sweep-and-mark-main

Major GC: Sweep-and-mark-main



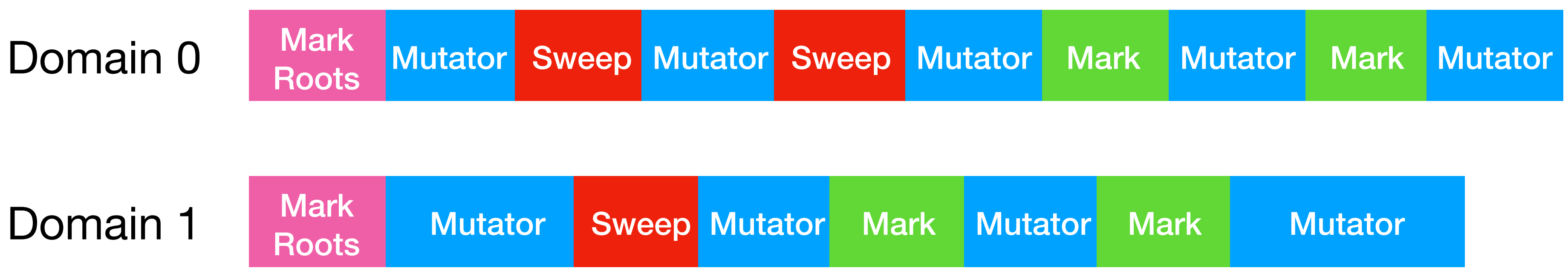
- Domains begin by marking roots

Major GC: Sweep-and-mark-main



- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator

Major GC: Sweep-and-mark-main

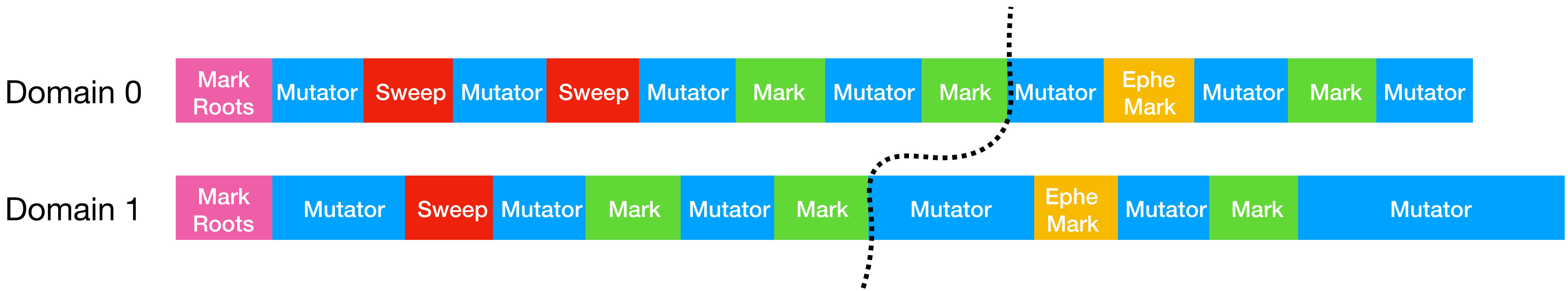


- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator

Major GC: Sweep-and-mark-main

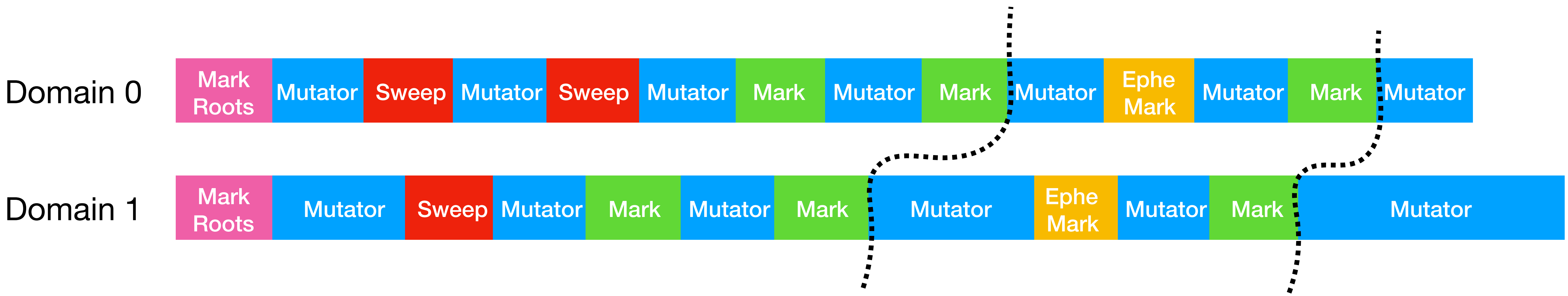
- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator

Major GC: Sweep-and-mark-main



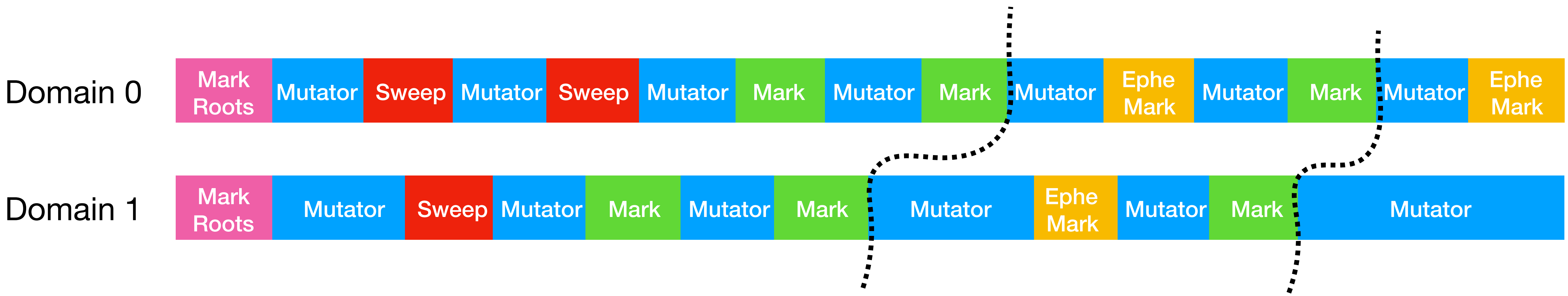
- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator
- Domains alternate between marking ephemerons, marking other objects and running mutator

Major GC: Sweep-and-mark-main



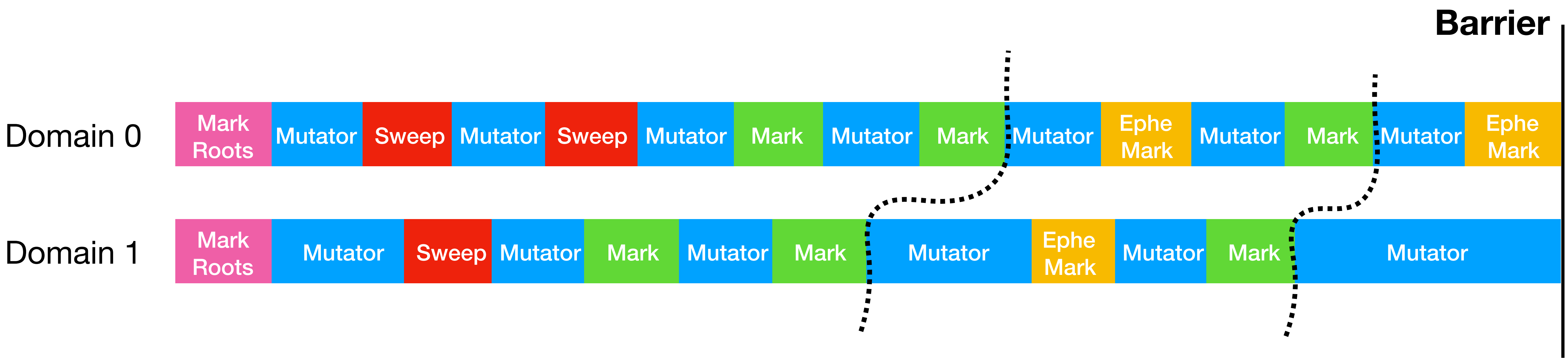
- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator
- Domains alternate between marking ephemerons, marking other objects and running mutator

Major GC: Sweep-and-mark-main



- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator
- Domains alternate between marking ephemerons, marking other objects and running mutator

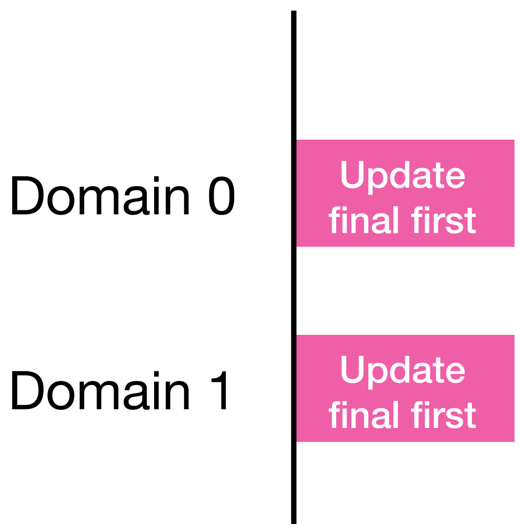
Major GC: Sweep-and-mark-main



- Domains begin by marking roots
- Domains alternate between sweeping own garbage and running mutator
- Domains alternate between marking objects and running mutator
- Domains alternate between marking ephemerons, marking other objects and running mutator
- Global barrier to switch to the next phase
  - ★ Reading weak keys may make unreachable objects reachable
  - ★ Verify that the phase termination conditions hold

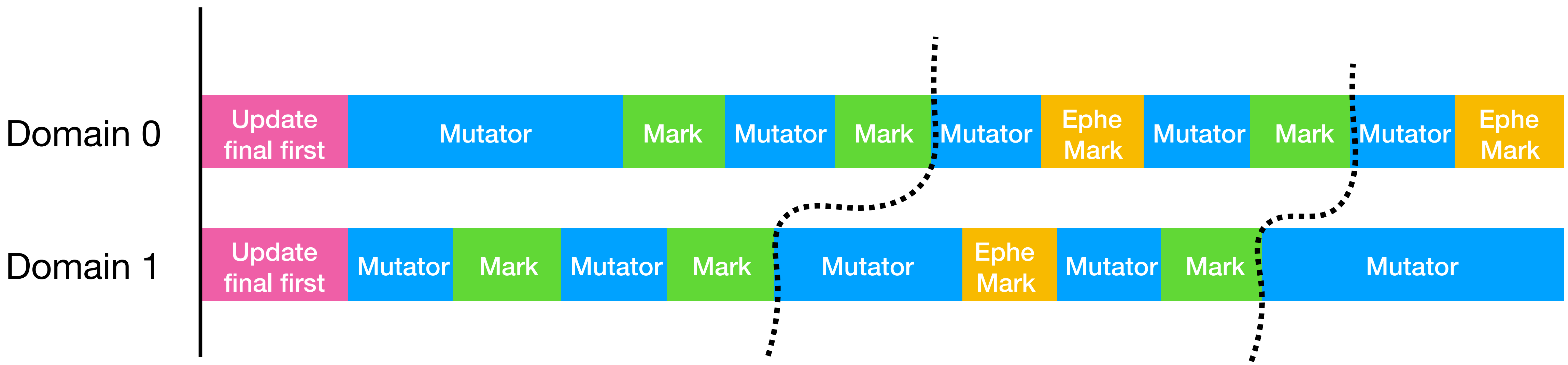
Major GC: mark-final

Major GC: mark-final



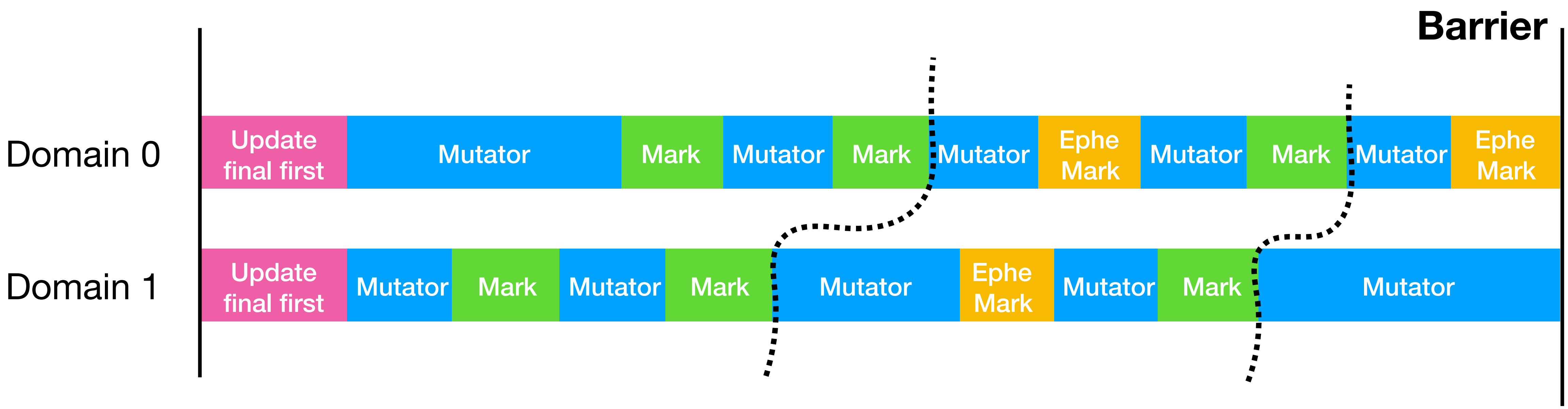
- Domains update `Gc.finalise` finalisers which take values and mark the values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk

Major GC: mark-final



- Domains update `Gc.finalise` finalisers which take values and mark the values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk

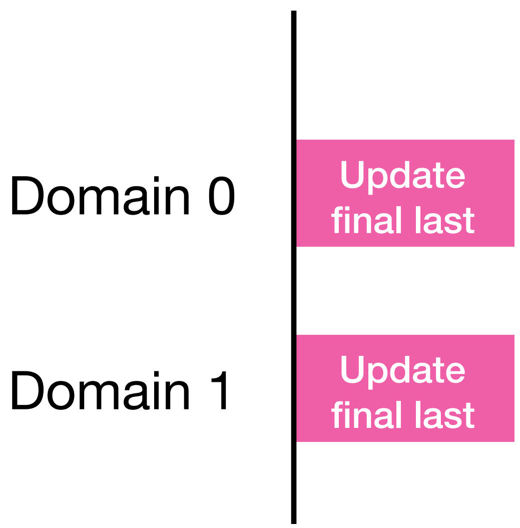
Major GC: mark-final



- Domains update `Gc.finalise` finalisers which take values and mark the values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk

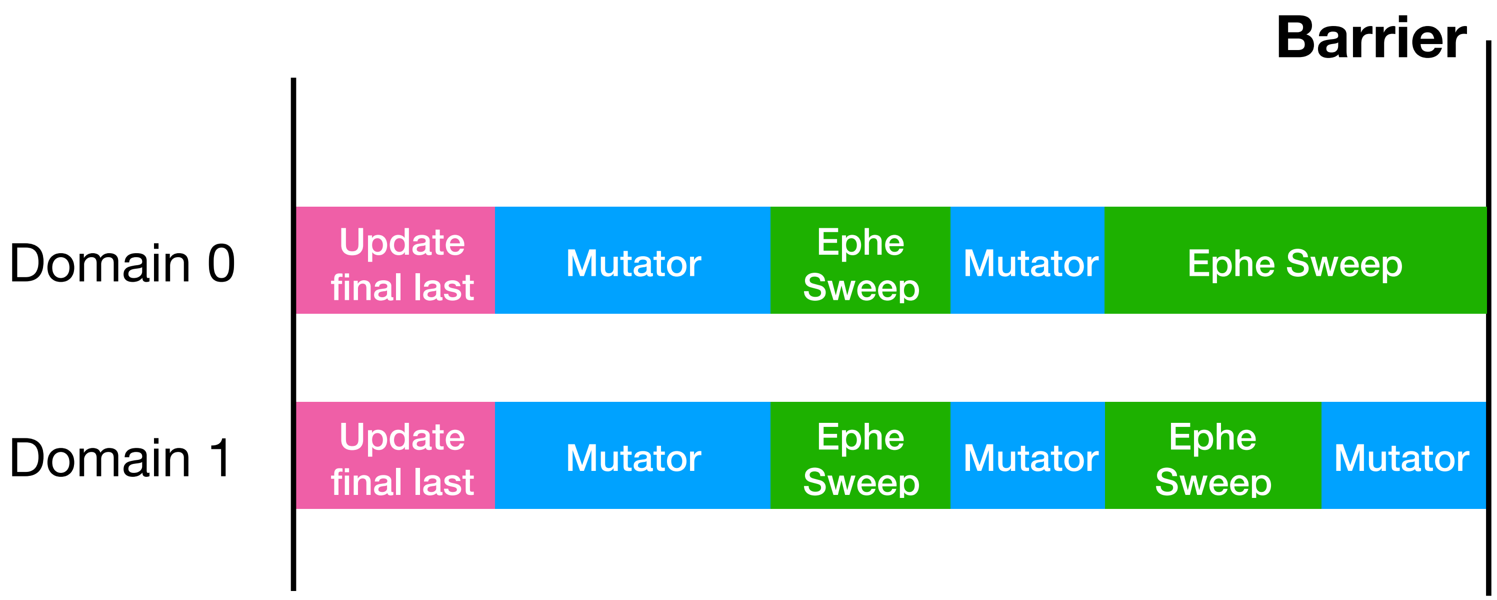
Major GC: sweep-ephe

Major GC: sweep-ephe



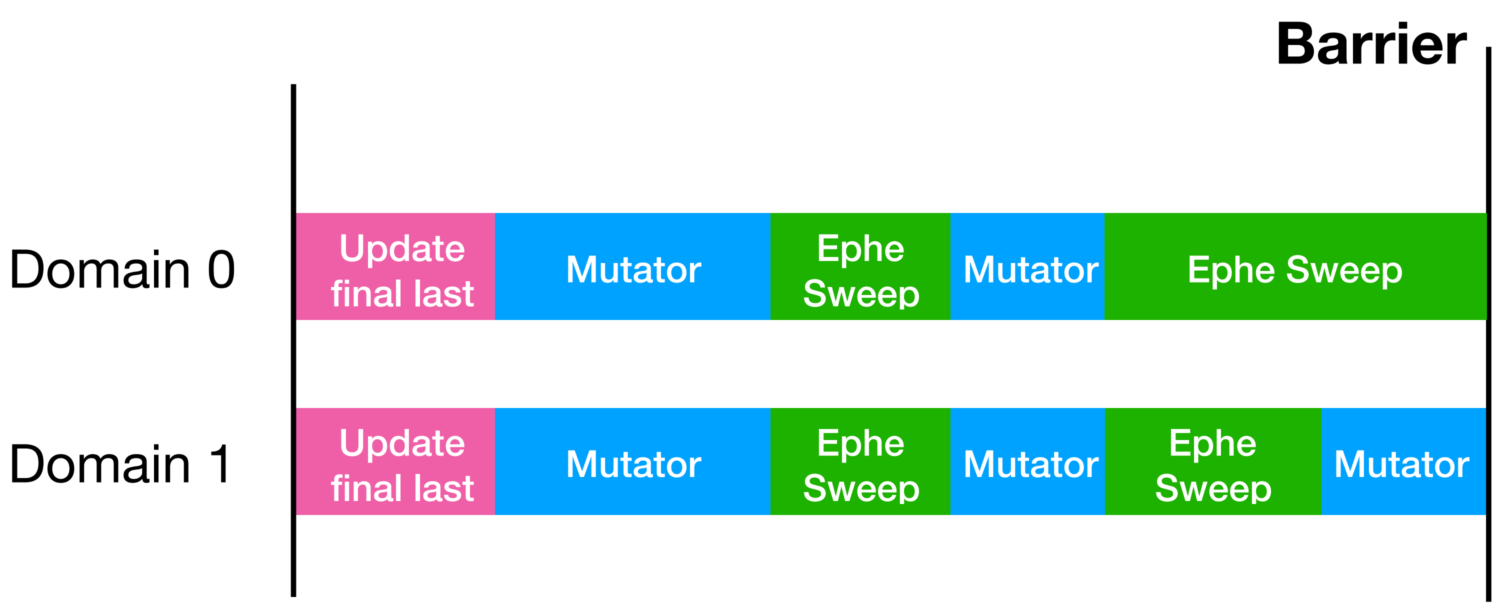
- Domains prepares the `Gc.finalise_last` finaliser list which do not take values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk

Major GC: sweep-ephe



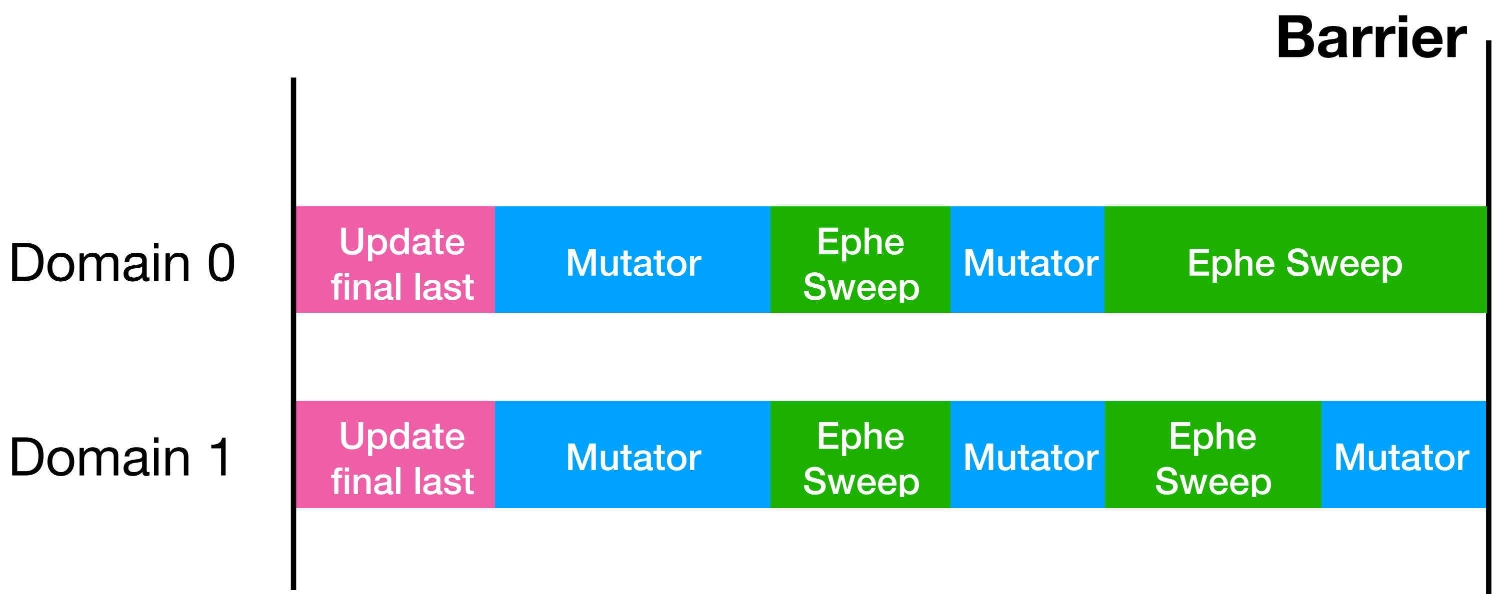
- Domains prepares the `Gc.finalise_last` finaliser list which do not take values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk

Major GC: sweep-ephe



- Domains prepares the `Gc.finalise_last` finaliser list which do not take values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk
- Swap the meaning of GC bits
  - ★ MARKED → UNMARKED
  - ★ UNMARKED → GARBAGE
  - ★ GARBAGE → MARKED

Major GC: sweep-ephe



- Domains prepares the `Gc.finalise_last` finaliser list which do not take values
  - ★ Preserves the order of evaluation of finalisers per domain c.f trunk
- Swap the meaning of GC bits
  - ★ MARKED → UNMARKED
  - ★ UNMARKED → GARBAGE
  - ★ GARBAGE → MARKED
- Major GC algorithm verified in SPIN model checker

Memory Model

Memory Model

- Goal: Balance comprehensibility and performance

Memory Model

- Goal: Balance comprehensibility and performance
- Generalise
  - ★ SC-DRF property
    - ✦ Data-race-free programs have sequential semantics
  - ★ to local DRF
    - ✦ Data-race-free *parts* of programs have sequential semantics

Memory Model

- Goal: Balance comprehensibility and performance
- Generalise
  - ★ SC-DRF property
    - ✦ Data-race-free programs have sequential semantics
  - ★ to local DRF
    - ✦ Data-race-free *parts* of programs have sequential semantics
- Bounds data races in *space* and *time*
  - ★ Data races on one location do not affect sequential semantics of another
  - ★ Dara races in the past or the future do no affect sequential semantics of non-racy accesses

Memory Model

Memory Model

- We have developed a memory model that has LDRF
  - ★ Atomic and non-atomic locations (no relaxed operations yet)
  - ★ Proven correct (on paper) compilation to x86 and ARMv8

Memory Model

- We have developed a memory model that has LDRF
  - ★ Atomic and non-atomic locations (no relaxed operations yet)
  - ★ Proven correct (on paper) compilation to x86 and ARMv8
- *Is it practical?*
  - ★ SC has LDRF and SRA is conjectured to have LDRF, but not practical due to performance impact

Memory Model

- We have developed a memory model that has LDRF
  - ★ Atomic and non-atomic locations (no relaxed operations yet)
  - ★ Proven correct (on paper) compilation to x86 and ARMv8
- *Is it practical?*
  - ★ SC has LDRF and SRA is conjectured to have LDRF, but not practical due to performance impact
- Must preserve load-store ordering
  - ★ Most compiler optimisations are valid (CSE, LICM).
    - ✦ No redundant store elimination across load.
  - ★ Free on x86, low-overhead on ARM (0.6% overhead) and POWER (2.9% overhead)

Runtime support for Effect handlers

Runtime support for Effect handlers

- Linear delimited continuations
  - ★ Linearity enforced by the runtime
  - ★ Raise exception when continuation resumed more than once
  - ★ Finaliser *discontinues* unresumed continuation

Runtime support for Effect handlers

- Linear delimited continuations
  - ★ Linearity enforced by the runtime
  - ★ Raise exception when continuation resumed more than once
  - ★ Finaliser *discontinues* unresumed continuation
- Fibers: Heap managed stack segments
  - ★ Requires stack-overflow checks at function entry
  - ★ Static analysis removes checks in small leaf functions

Runtime support for Effect handlers

- Linear delimited continuations
  - ★ Linearity enforced by the runtime
  - ★ Raise exception when continuation resumed more than once
  - ★ Finaliser *discontinues* unresumed continuation
- Fibers: Heap managed stack segments
  - ★ Requires stack-overflow checks at function entry
  - ★ Static analysis removes checks in small leaf functions
- C calls needs to be performed on C stack
  - ★ < 1% performance slowdown on average for this feature
  - ★ DWARF magic allows full backtrace across nested calls of handlers, C calls and callbacks.

Runtime support for Effect handlers

- Linear delimited continuations
  - ★ Linearity enforced by the runtime
  - ★ Raise exception when continuation resumed more than once
  - ★ Finaliser *discontinues* unresumed continuation
- Fibers: Heap managed stack segments
  - ★ Requires stack-overflow checks at function entry
  - ★ Static analysis removes checks in small leaf functions
- C calls needs to be performed on C stack
  - ★ < 1% performance slowdown on average for this feature
  - ★ DWARF magic allows full backtrace across nested calls of handlers, C calls and callbacks.
- WIP to support capturing continuations that include C frames c.f "Threads Yield Continuations"

Status

- Major GC and fiber implementations are stable modulo bugs
  - ★ *TODO: Effect System*
- Laundry list of minor features
  - ★ https://github.com/ocamllabs/ocaml-multicore/projects/3
- We need
  - ★ Benchmarks
  - ★ Benchmarking tools and infrastructure
  - ★ Performance tuning

Future Directions: Memory Model

Future Directions: Memory Model

- Memory model only supports atomic and non-atomic locations
    - ★ Extend memory model with *weaker* atomics and "new ref" while preserving LDRF theorem

Future Directions: Memory Model

- Memory model only supports atomic and non-atomic locations
  - ★ Extend memory model with *weaker* atomics and "new ref" while preserving LDRF theorem
- *Avoid become C++* — multiple weak atomics w/ subtle interactions
  - ★ Could we expose restricted APIs to the programmer?

Future Directions: Memory Model

- Memory model only supports atomic and non-atomic locations
  - ★ Extend memory model with *weaker* atomics and "new ref" while preserving LDRF theorem
- *Avoid become C++* — multiple weak atomics w/ subtle interactions
  - ★ Could we expose restricted APIs to the programmer?
- Verify multicore OCaml programs
  - ★ Explore (semi-)automated SMT-aided verification
  - ★ Challenge problem: verify k-CAS at the heart of Reagents library

Future Directions: Multicore MirageOS

Future Directions: Multicore MirageOS

- MirageOS rewrite to take advantage of typed effect handlers and multicore parallelism
  - ★ Typed effects for better *error handling* and *concurrency*

Future Directions: Multicore MirageOS

- MirageOS rewrite to take advantage of typed effect handlers and multicore parallelism
  - ★ Typed effects for better *error handling* and *concurrency*
- Better concurrency model over Xen block devices
  - ★ Extricate oneself from dependence on POSIX API
  - ★ Discriminate various concurrency levels (CPU, application, I/O) in the scheduler
  - ★ *Failure* and *Back pressure* as a first-class operation

Future Directions: Multicore MirageOS

- MirageOS rewrite to take advantage of typed effect handlers and multicore parallelism
  - ★ Typed effects for better *error handling* and *concurrency*
- Better concurrency model over Xen block devices
  - ★ Extricate oneself from dependence on POSIX API
  - ★ Discriminate various concurrency levels (CPU, application, I/O) in the scheduler
  - ★ *Failure* and *Back pressure* as a first-class operation
- Multicore-capable *Irmin*, a branch-consistent database library

Future Directions: Heterogeneous System

- Programming heterogenous, non Von Neumann architectures
  - ★ How do we capture computational model in richer type system?
  - ★ How do we compile efficiently to such a system?