INIT and FINALIZE issues

Madrid, September 2013

Hybrid WG

Jeff Squyres

v0.7 – 12 Sep 2013

Main goal

- Overcome limitations while using MPI with stacked and threaded libraries in a single MPI process
  – Stacked: single INIT/FINALIZE limitation requires each library to have global knowledge of who else is using MPI
  – Threaded: there are race conditions

How to accomplish the main goal

- Any thread can call MPI_INIT at any time
  - If a thread wants to use MPI, just call INIT
  - Avoids issues of invoking MPI API before INIT

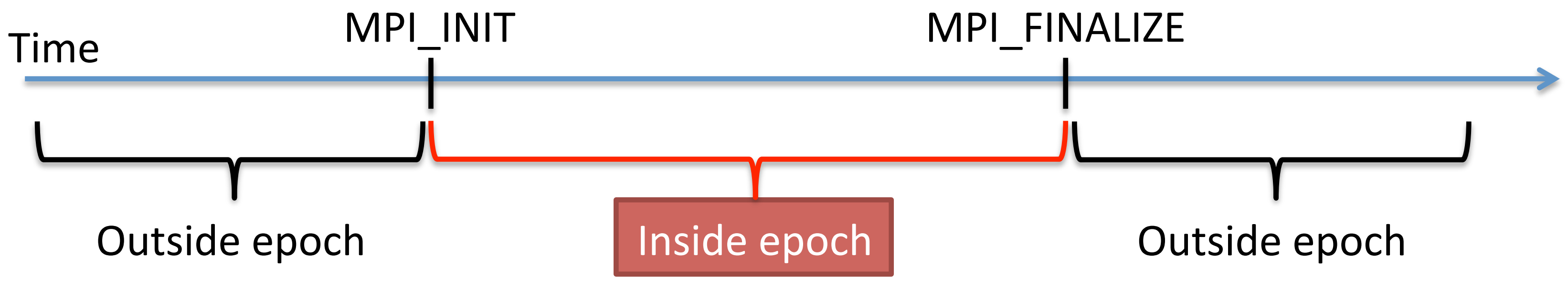
Encompasses three related topics

1. Thread safe INIT / FINALIZE
2. Nested INIT / FINALIZE
3. Re-initialization of MPI

(for brevity, only mentioning INIT in these slides, but everything also applies to INIT_THREAD)

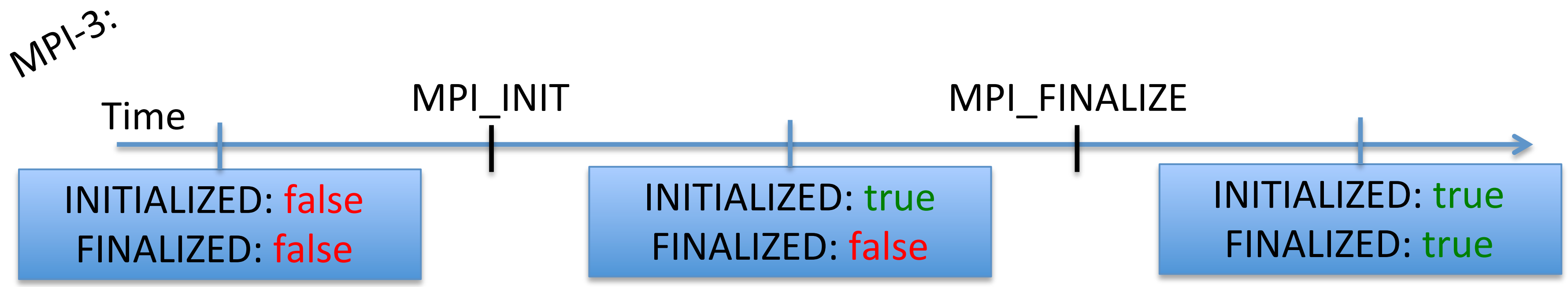
New definition: MPI epoch

- Current text phrasing:
  - "Before MPI_INIT is called…"
  - "After MPI_FINALIZE is called…"
- Need to re-spin this language
  - MPI "epoch": between initialization and finalization
  - You are either inside or outside of an epoch



Deprecate INITIALIZED and FINALIZED

- Based on the *first* initialization / finalization
  - Also: Inherently racy with multiple threads
  - Bottom line: libraries need their own state to know if MPI *and* their own local state is valid



New INIT Behavior

- Defined to be thread safe
  - Any thread can call INIT at any time
  - …regardless of resulting MPI thread level
  - …regardless of whether in MPI epoch or not

New INIT Behavior: Consequences

- Behave as if all calls to INIT increment internal ref count
- If MPI is not currently initialized
  - Initialize MPI / start a new MPI epoch
  - This can happen multiple times in a process
- Local-only operation if MPI is already initialized (i.e., ref count increment)

New INIT Behavior: Consequences

- If multiple threads call INIT simultaneously
  - One thread will actually initialize MPI
  - Rest will block until MPI is actually initialized
- High quality implementation will allow different thread levels in different epochs

New INIT Behavior: Consequences

- Thread level determination
  - Set by the initializing call to INIT (at the beginning of the epoch)
  - "Requested" level *may* be ignored by subsequent calls to INIT in the same epoch
  - "Provided" may not be decreased during an epoch

New FINALIZE Behavior

- Must call FINALIZE as many times as INIT was called
- Erroneous to call FINALIZE outside of MPI epoch

- Must obey MPI thread level
  - In THREAD_MULTIPLE, FINALIZE must be thread safe
  - …just like all other MPI calls

New FINALIZE Behavior: Consequences

- All calls to FINALIZE behave as if they decrement internal ref count
  - Will actually finalize / close the epoch once ref count reaches zero

Collective Behavior

- INIT and FINALIZE are still collective
  - *…but only when they open / close an epoch*
  - Otherwise, they are local-only operations
    - A use-case / example coming later (be patient)
  - Behave as if they are increments / decrements on ref count (advice to implementers)

Outside of the MPI epoch

- All MPI handles go stale when epoch completes

Main thread

- Definition essentially stays the same
  - Main thread = thread that initialized MPI in this epoch
  - Concept cannot be deprecated because of THREAD_FUNNELED

  …more on this topic later…
- In MPI-3, main thread must finalize
  - Only relevant in THREAD_MULTIPLE case
  - Open questions:
    - Does this still matter? We don't think so.
    - Does anyone know why this restriction exists?
      (need to poll implementers)

QUERY_THREAD / IS_THREAD_MAIN

- Must always be thread safe
  - Any thread can call these functions at any time
  - …regardless of MPI thread level

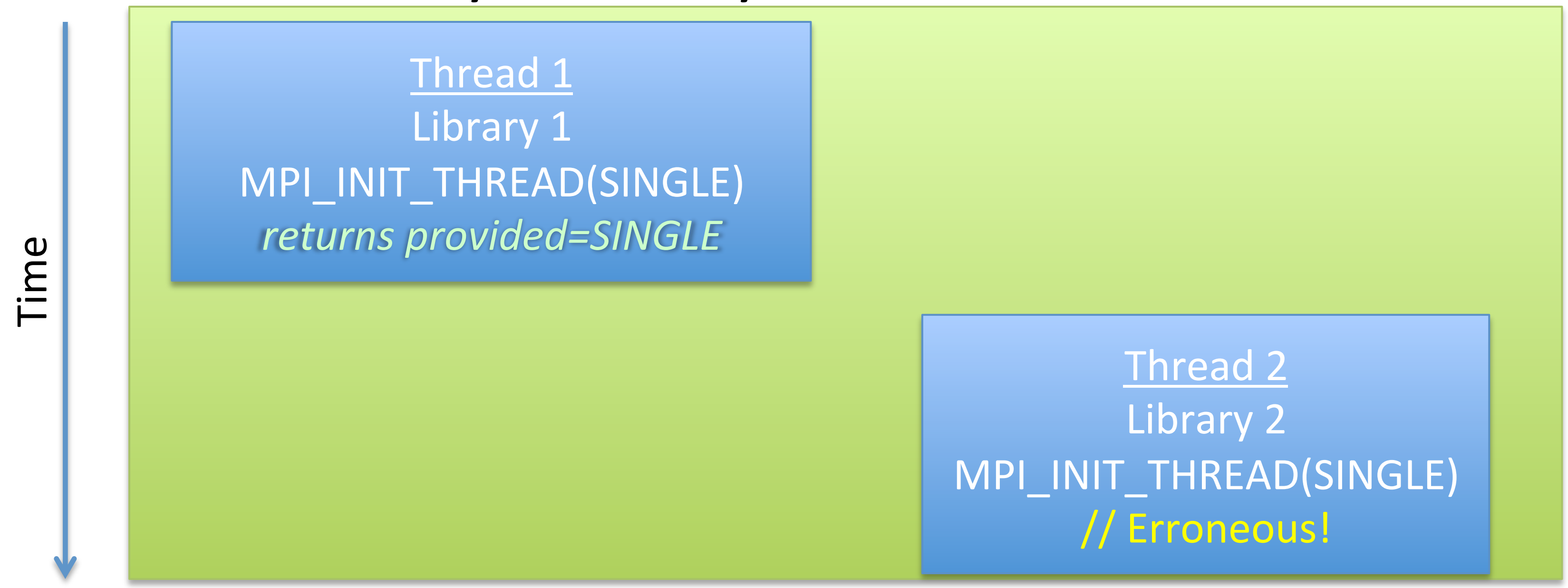
```
MPI_QUERY_THREAD();
if SERIALIZED:
    THREAD_LOCK
    MPI_FOO
    THREAD_UNLOCK
else
    MPI_FOO
endif
```

INITIALIZED / FINALIZED

- Do not change these functions at all
  - I.e., keep the current (non-thread safe) definitions
- In fact, deprecate them!
  - Rationale for why they are being deprecated: inherent race condition between multiple threads calling MPI_INIT and MPI_INITIALIZED

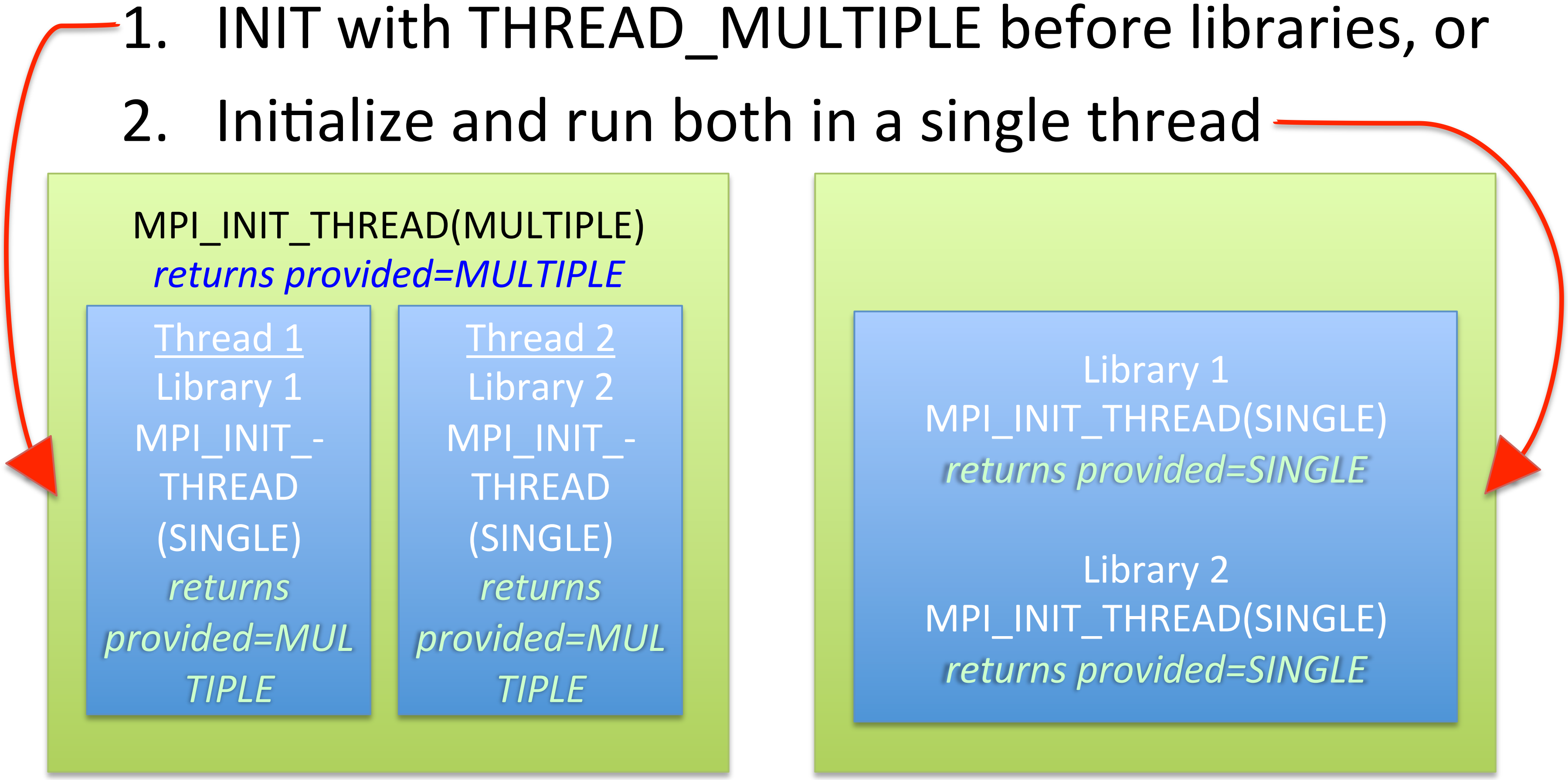
Corner case 1

- Multiple libraries in a process INIT with THREAD_SINGLE in different threads
  - Erroneous; MPI doesn't define what happens
  - → But likely, the only reasonable choice is to abort



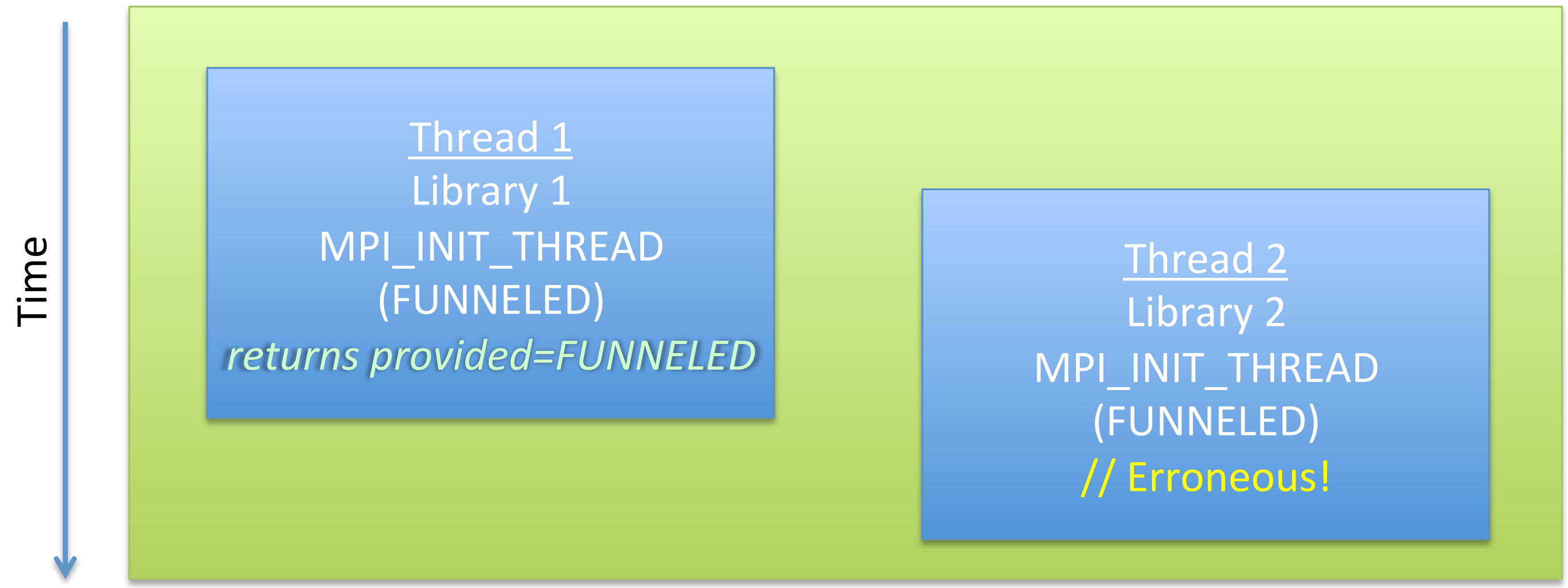
Corner case 1

- Workarounds:
  1. INIT with THREAD_MULTIPLE before libraries, or
  2. Initialize and run both in a single thread


MPI_INIT_THREAD(MULTIPLE)
returns provided=MULTIPLE
Thread 1
Library 1
MPI_INIT_-THREAD (SINGLE)
returns provided=MULTIPLE
Thread 2
Library 2
MPI_INIT_-THREAD (SINGLE)
returns provided=MULTIPLE
Library 1
MPI_INIT_THREAD(SINGLE)
returns provided=SINGLE
Library 2
MPI_INIT_THREAD(SINGLE)
returns provided=SINGLE

Corner case 2

- Multiple libraries in a process INIT with FUNNELED or SERIALIZED in different threads
  - Similar to SINGLE (slide 18): this is erroneous



Low quality implementations

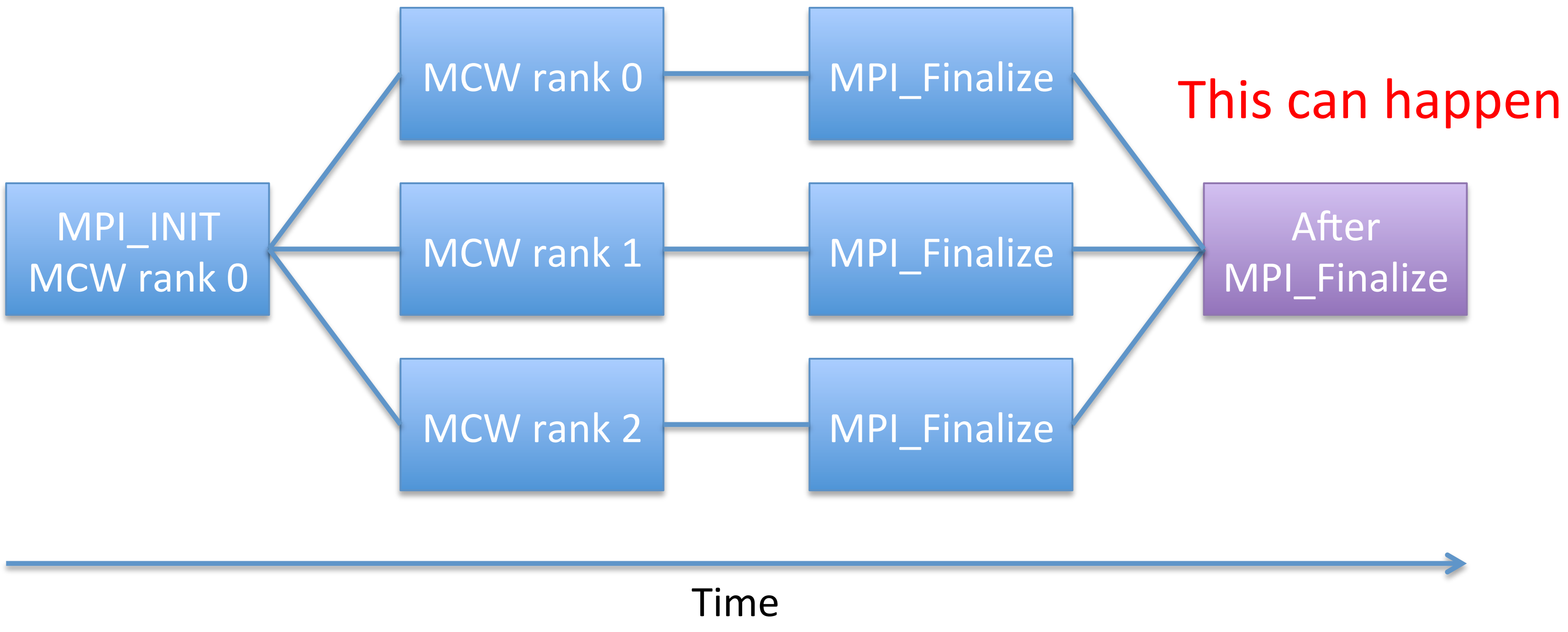
- Never actually finalize MPI, even when ref count decrements to 0
  - Maybe finalize via atexit() handler, or somesuch
  - *But still must behave as if finalized when ref count decrements to 0* (e.g., MPI handles go stale)
- Never change thread level after first MPI epoch

High quality implementations

- Actually finalizes MPI when ref count gets to 0
  - Release internal resources, etc.
- Re-initializes at next MPI epoch
  - Allow new thread level for new epoch

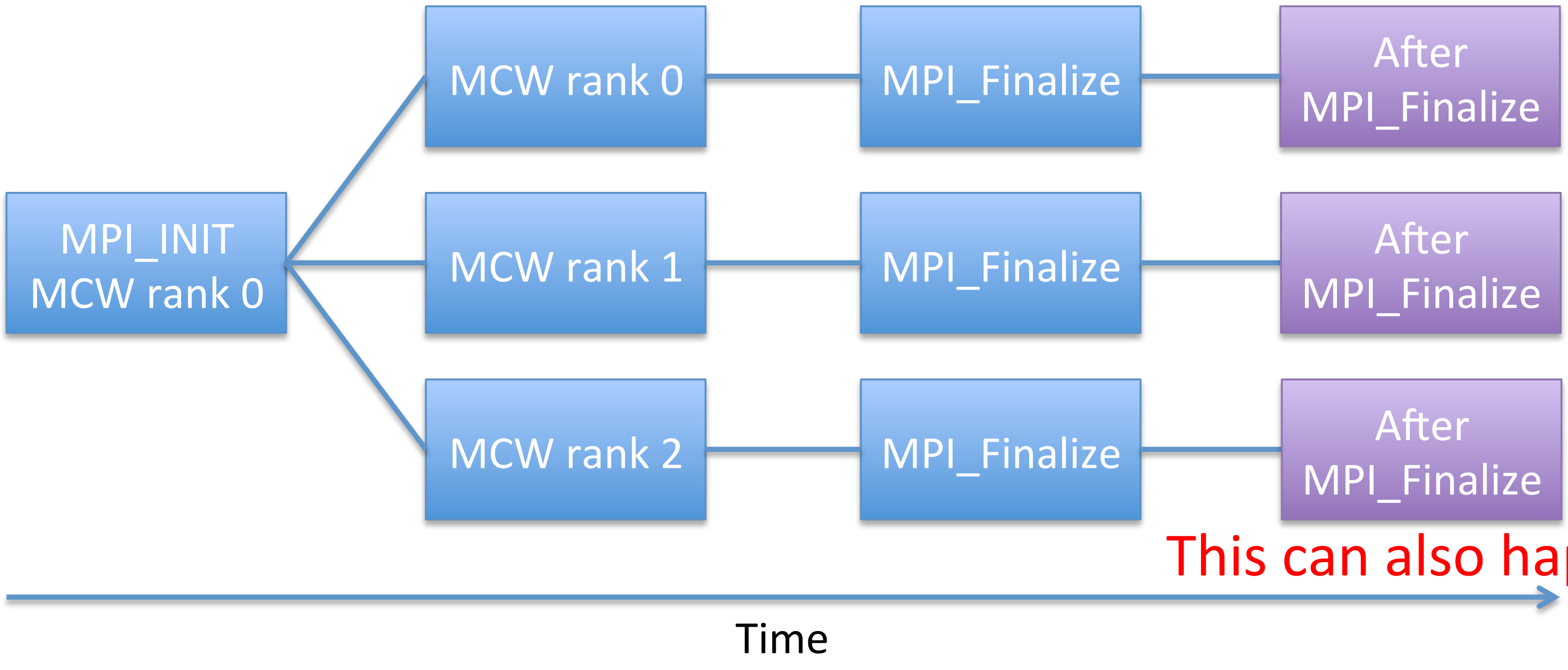
Corner case 3

- No guarantee which threads / OS processes continue to exist outside of MPI epoch
  - Threads or processes may die during finalization



Corner case 3

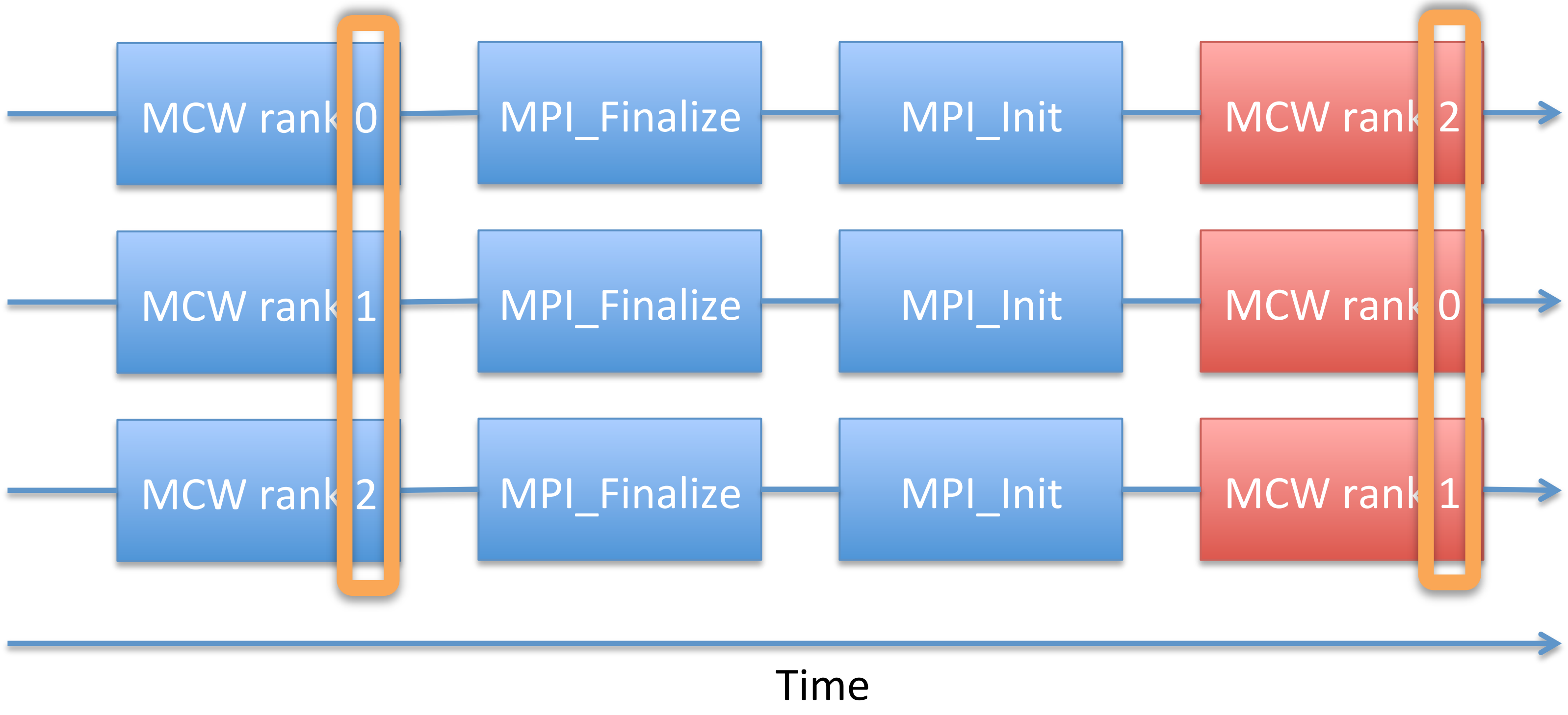
- No guarantee which threads / OS processes continue to exist outside of MPI epoch
  - Threads or processes may die during finalization



This can also happen

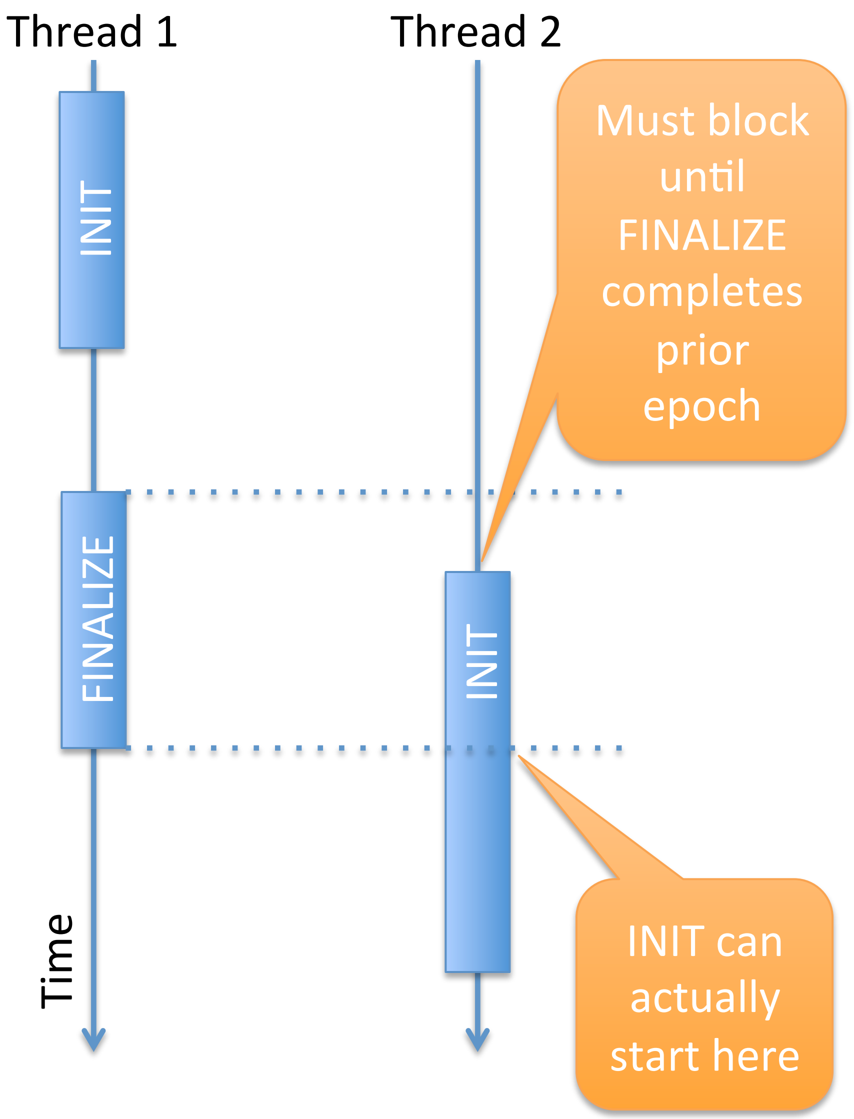
Corner case 3

- COMM_WORLD rank may change
  - If an MPI process exists outside of MPI epoch, its COMM_WORLD rank may change @ next epoch



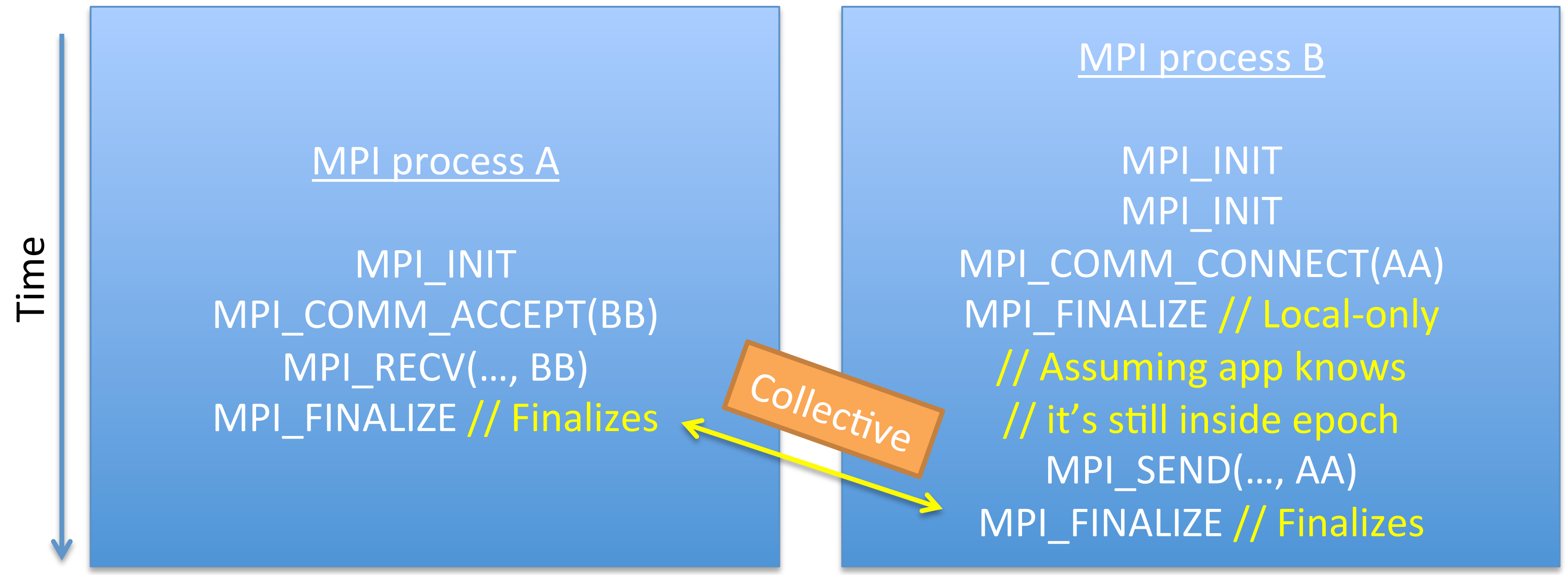
Corner case 4

- INIT may block until FINALIZE completes
  - If FINALIZE is in the process closing an epoch when INIT is invoked
  - In this case, INIT will re-initialize MPI


Thread 1
Thread 2
INIT
FINALIZE
INIT
Time
Must block until FINALIZE completes prior epoch
INIT can actually start here

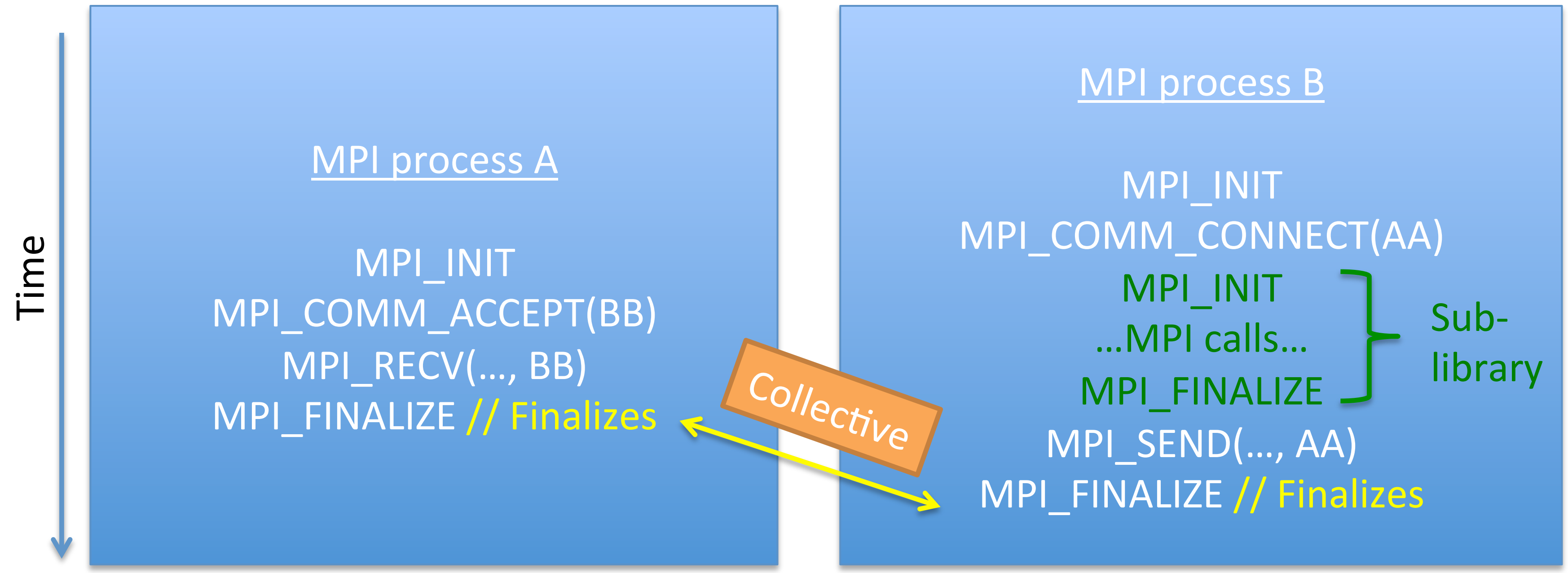
Corner case 5

- App must behave as if FINALIZE actually finalized
  - …unless it has a priori knowledge that the FINALIZE ref count is not yet zero



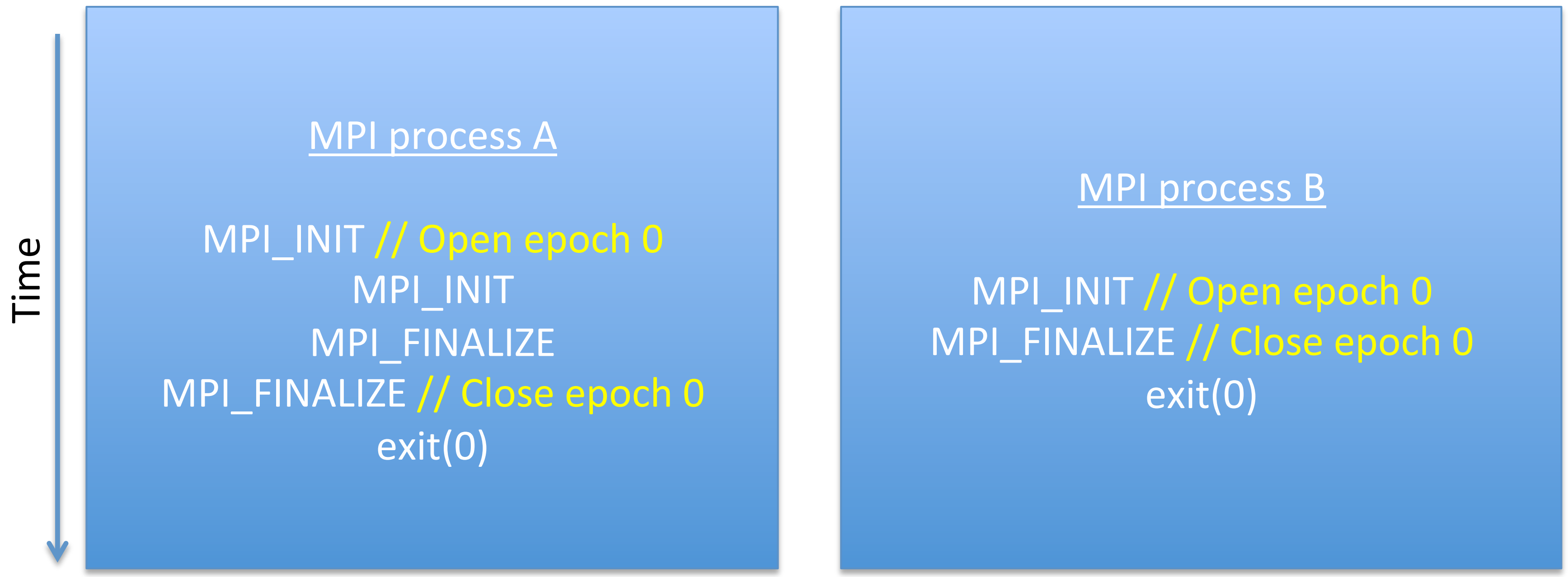
Corner case 5.1

- Notice the sub-library MPI_INIT / MPI_FINALIZE
  - Does not affect the end of the epoch
  - Specifically: does not affect A←→B connection



Corner case 6

- This is correct



MPI process A

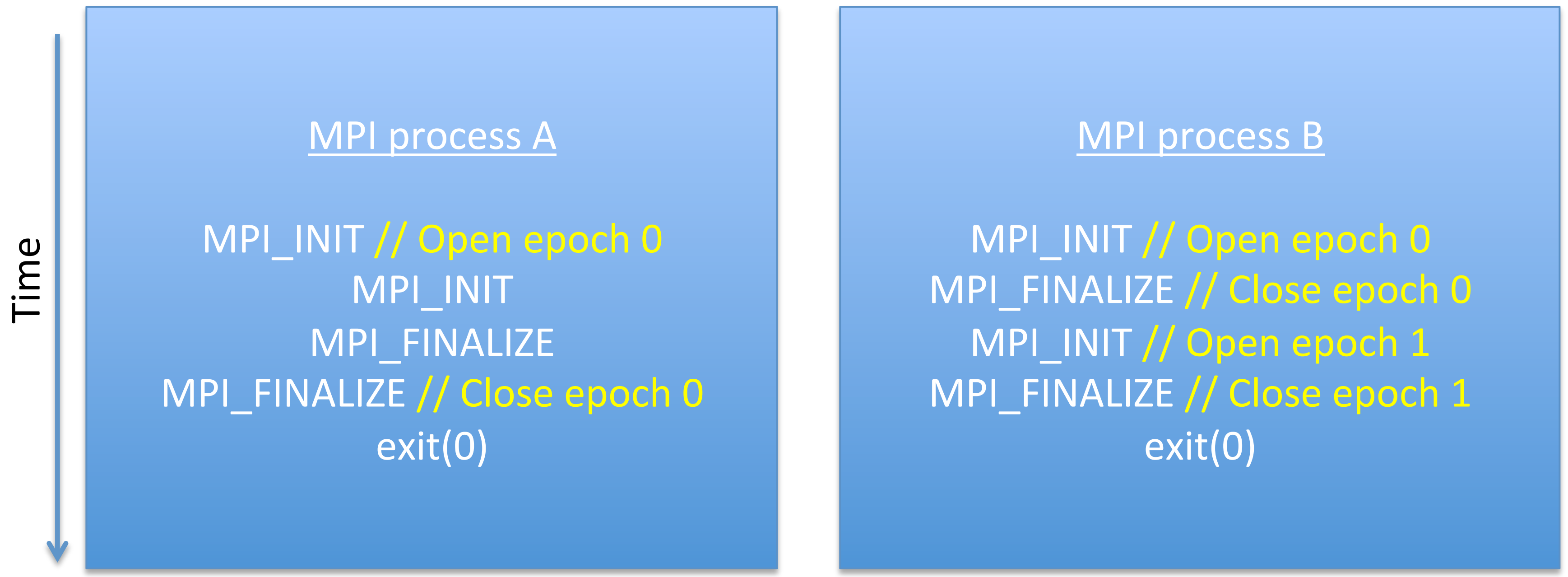
```
MPI_INIT // Open epoch 0
MPI_INIT
MPI_FINALIZE
MPI_FINALIZE // Close epoch 0
exit(0)
```

MPI process B

```
MPI_INIT // Open epoch 0
MPI_FINALIZE // Close epoch 0
exit(0)
```

Corner case 6

- This is not correct
  - "Doc, it hurts when I go like this…"



MPI process A

```
MPI_INIT // Open epoch 0
MPI_INIT
MPI_FINALIZE
MPI_FINALIZE // Close epoch 0
exit(0)
```

MPI process B

```
MPI_INIT // Open epoch 0
MPI_FINALIZE // Close epoch 0
MPI_INIT // Open epoch 1
MPI_FINALIZE // Close epoch 1
exit(0)
```

Summary

- MPI epoch concept
- INIT
  - Defined to be thread safe
  - (Behaves as if) Increment ref count
    - If ref count 1, collectively start new epoch
    - Otherwise, local-only operation
- QUERY_THREAD / IS_THREAD_MAIN
  - Defined to be thread safe

Summary

- Deprecate INITIALIZED and FINALIZED
  - Inherently racy, not enough for stacked libraries
- FINALIZE
  - Must be called as many times as INIT
  - (Behaves as if) Decrement ref-count
    - If went to 0, collectively close epoch
    - Otherwise, local-only operation

KTHXBYE