

$GHC_{(STG, Cmm, asm)}$ illustrated

for hardware person

exploring some mental models and implementations

Takenobu T.

"Any sufficiently advanced technology is
indistinguishable from **magic**."

Arthur C. Clarke

NOTE

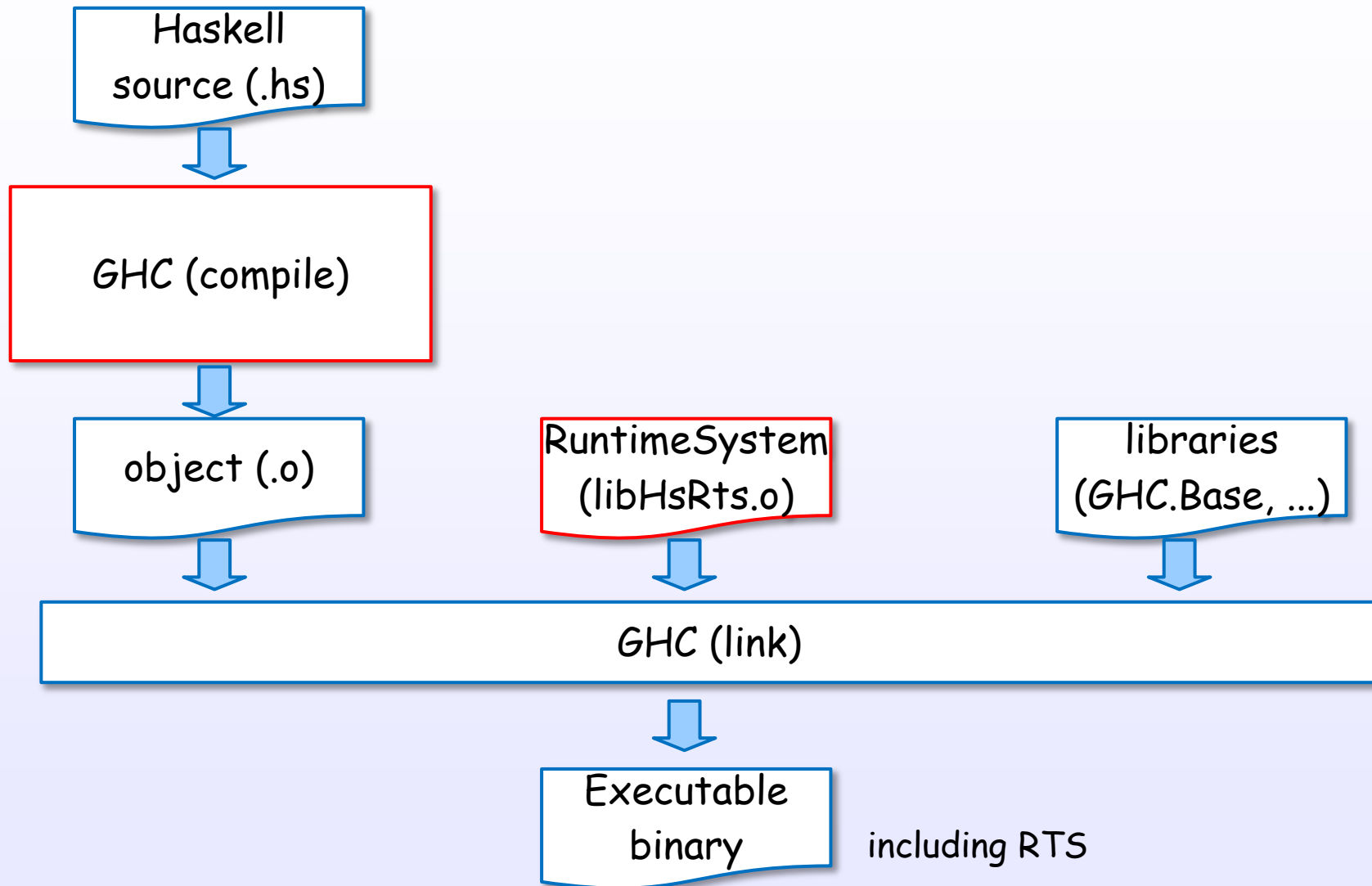
- This is not official document by ghc development team.
- Please don't forget "semantics". It's very important.
- This is written for ghc 7.8.

Contents

- Executable binary
- Compile steps
- Runtime System
- Development languages
- Machine layer/models
- STG-machine
- Heap object in STG-machine
- STG-machine evaluation
- Pointer tagging
- Thunk and update
- Allocate and free heap objects
- STG - C land interface
- Thread
- Thread context switch
- Creating main and sub threads
- Thread migration
- Heap and Threads
- Threads and GC
- Spark
- Mvar
- Software transactional memory
- IO and FFI
- IO manager
- Bootstrap
- References

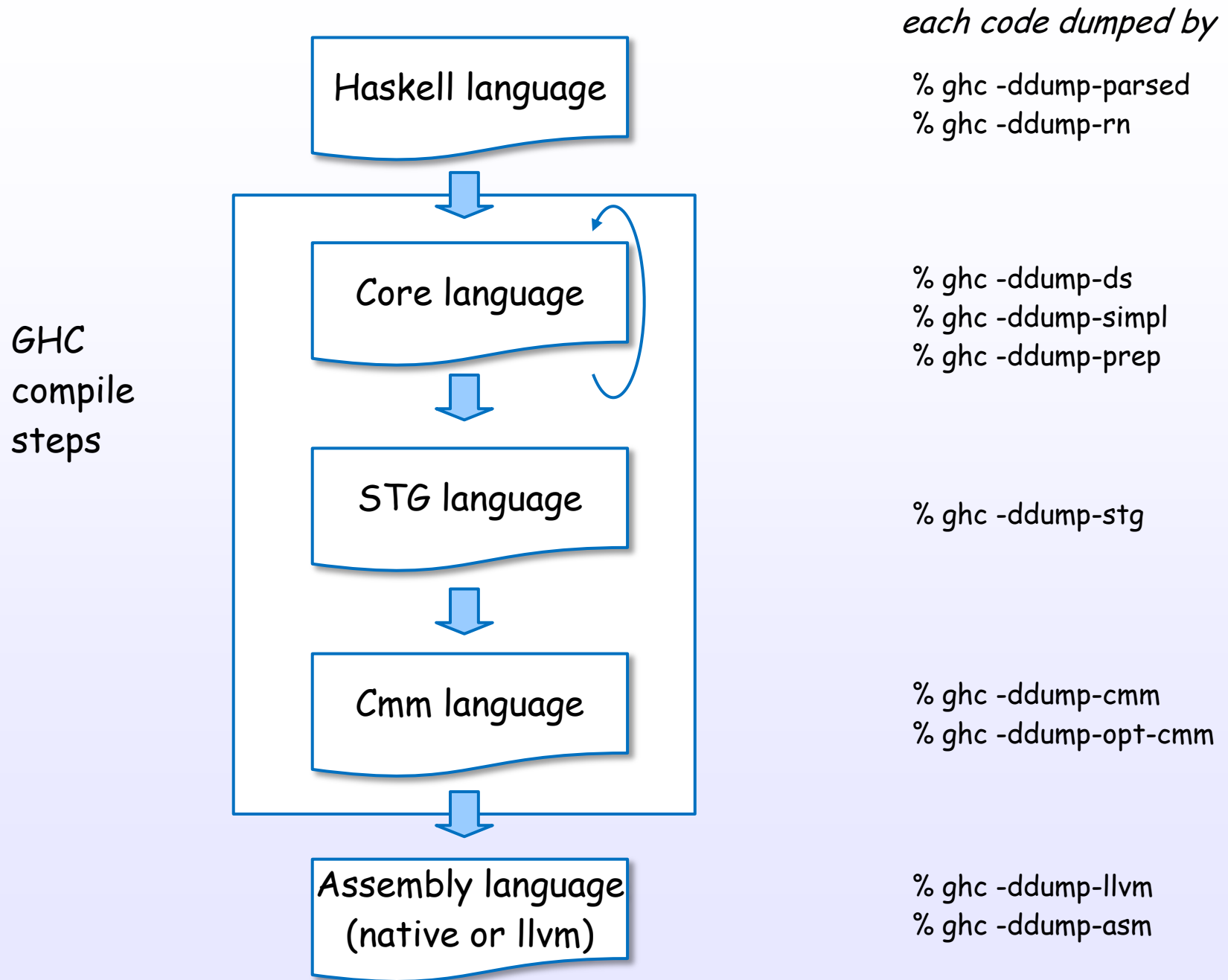
Executable binary

GHC = Compiler + Runtime System (RTS)



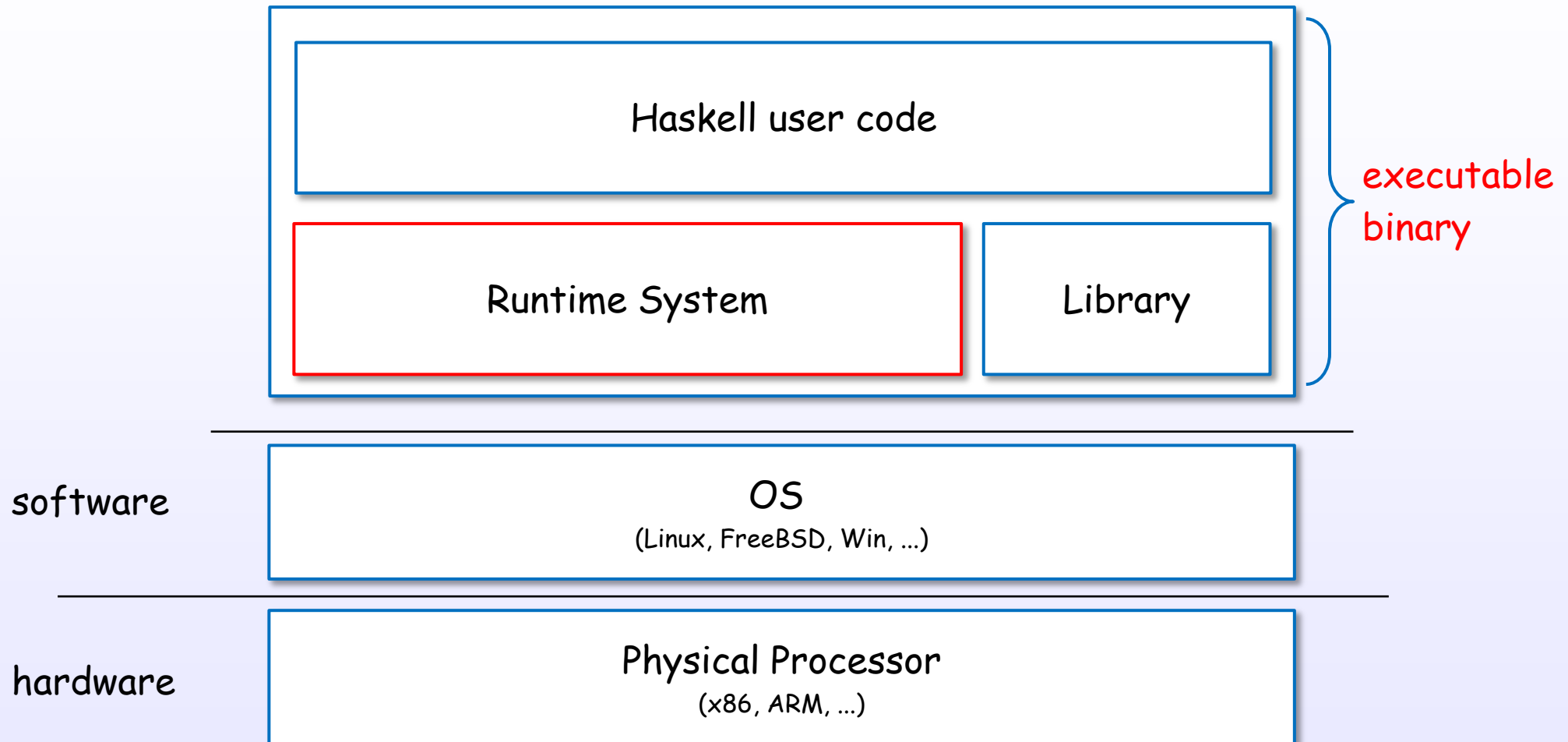
Compile steps

GHC transitions between five representations



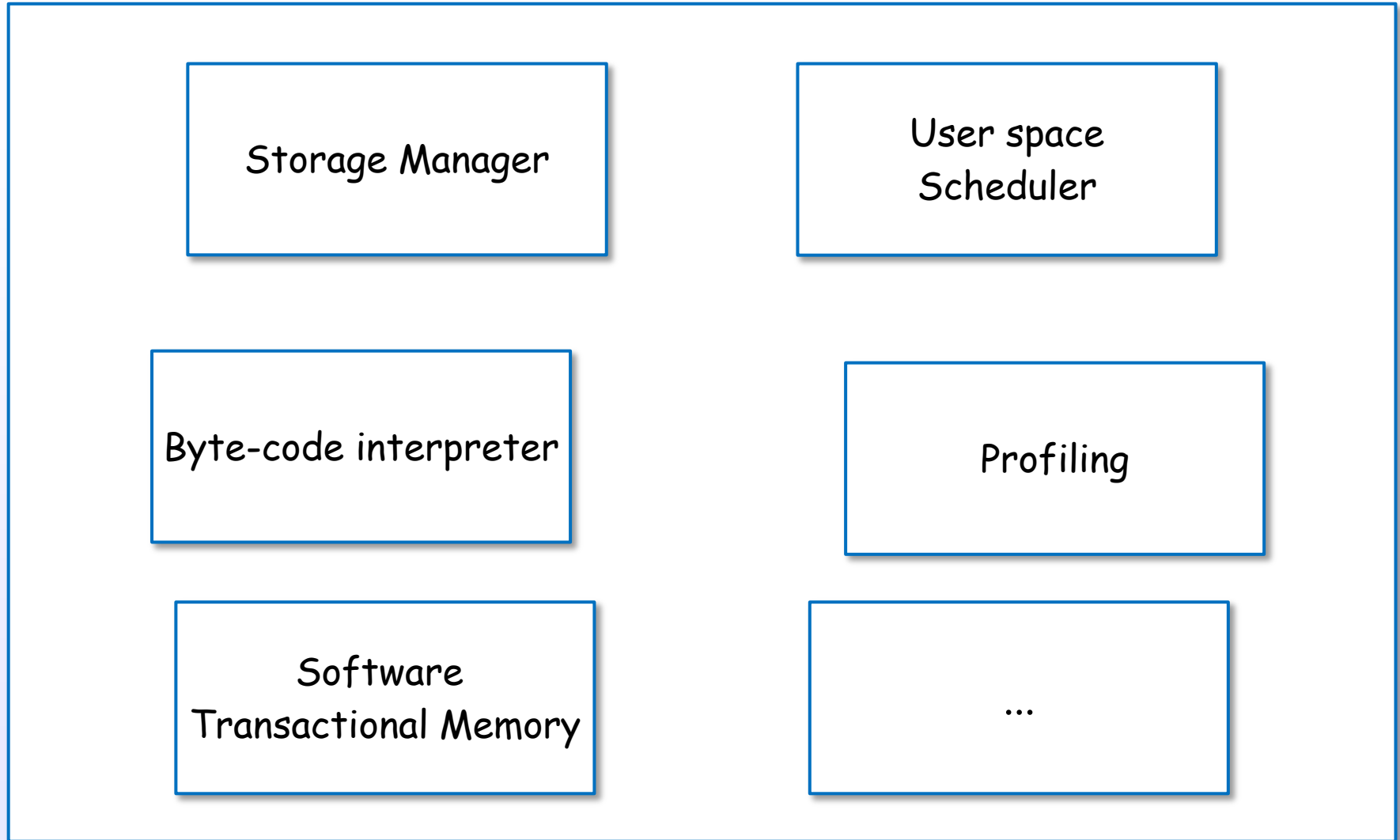
Runtime System

Generated binary includes RTS



Runtime System includes ...

Runtime System



Development languages

GHC developed by some languages

compiler

(\$(TOP)/**compiler**/*)

Haskell

+

Alex (lex)

Happy (yacc)

Cmm (C--)

Assembly

runtime system

(\$(TOP)/**rts**/*)

C

+

Cmm

Assembly

library

(\$(TOP)/**libraries**/*)

Haskell

+

C

Machine layer/models

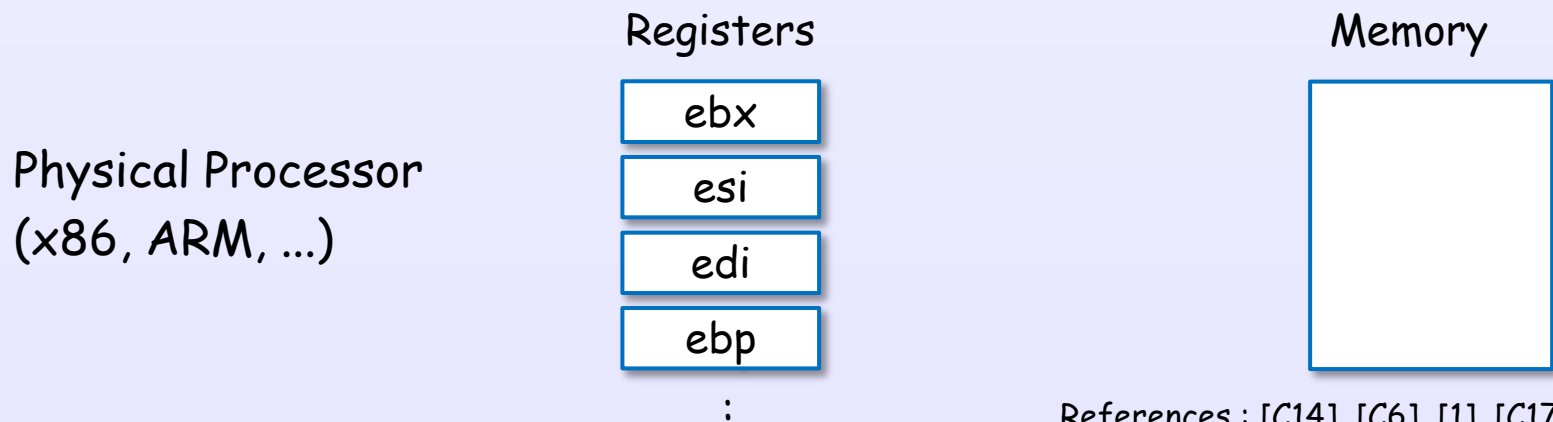
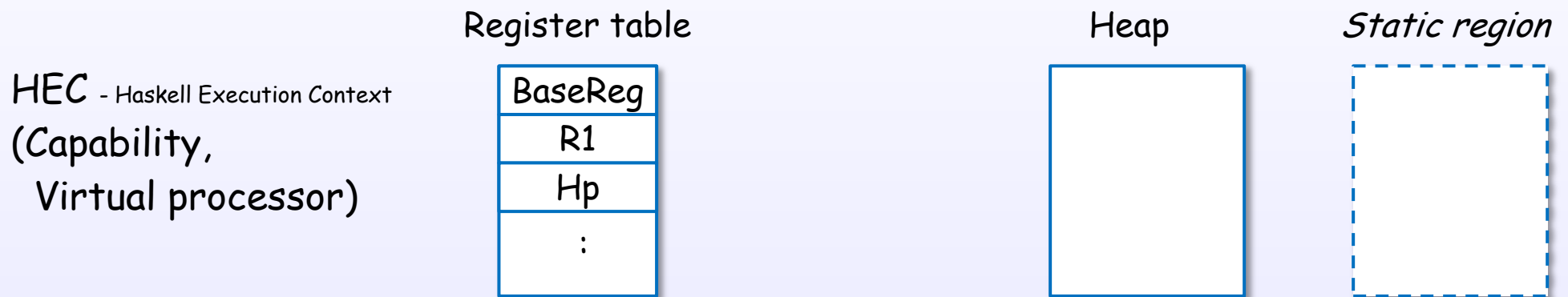
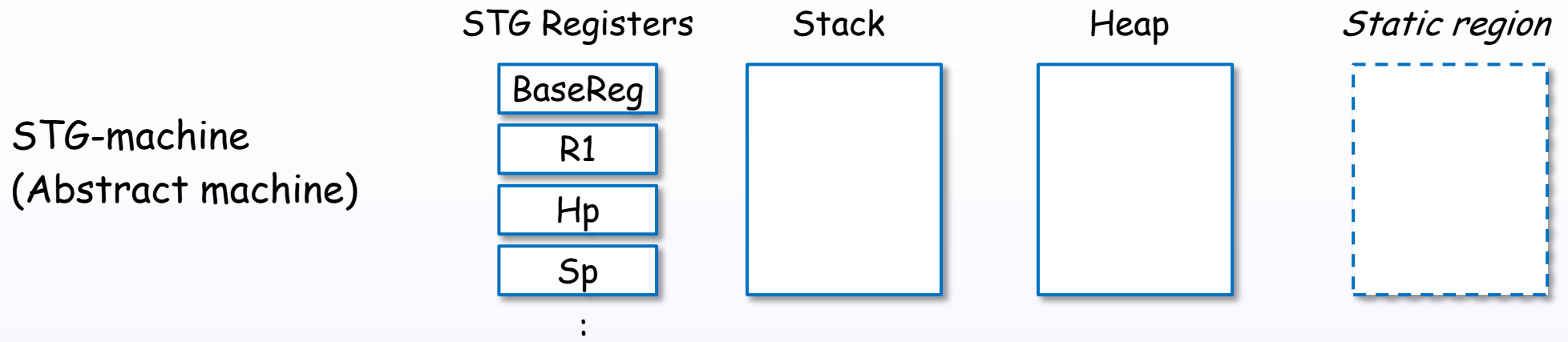
Machine layer

STG-machine
(Abstract machine)

HEC - Haskell Execution Context
(Capability, Virtual processor)

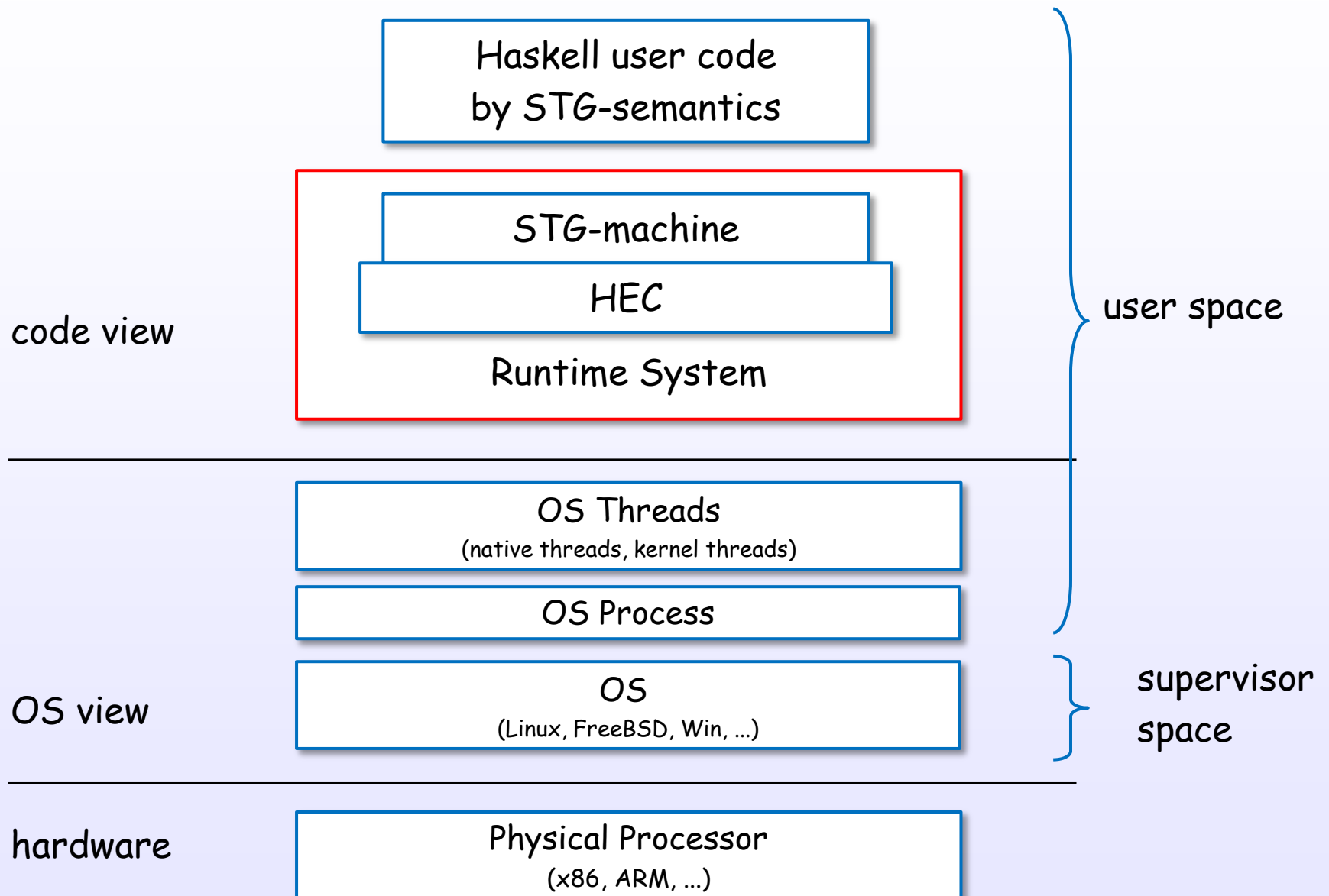
Physical Processor
(x86, ARM, ...)

Machine layer



References : [C14], [C6], [1], [C17], [7], [S15], [S16], [S11]

Runtime system and HEC

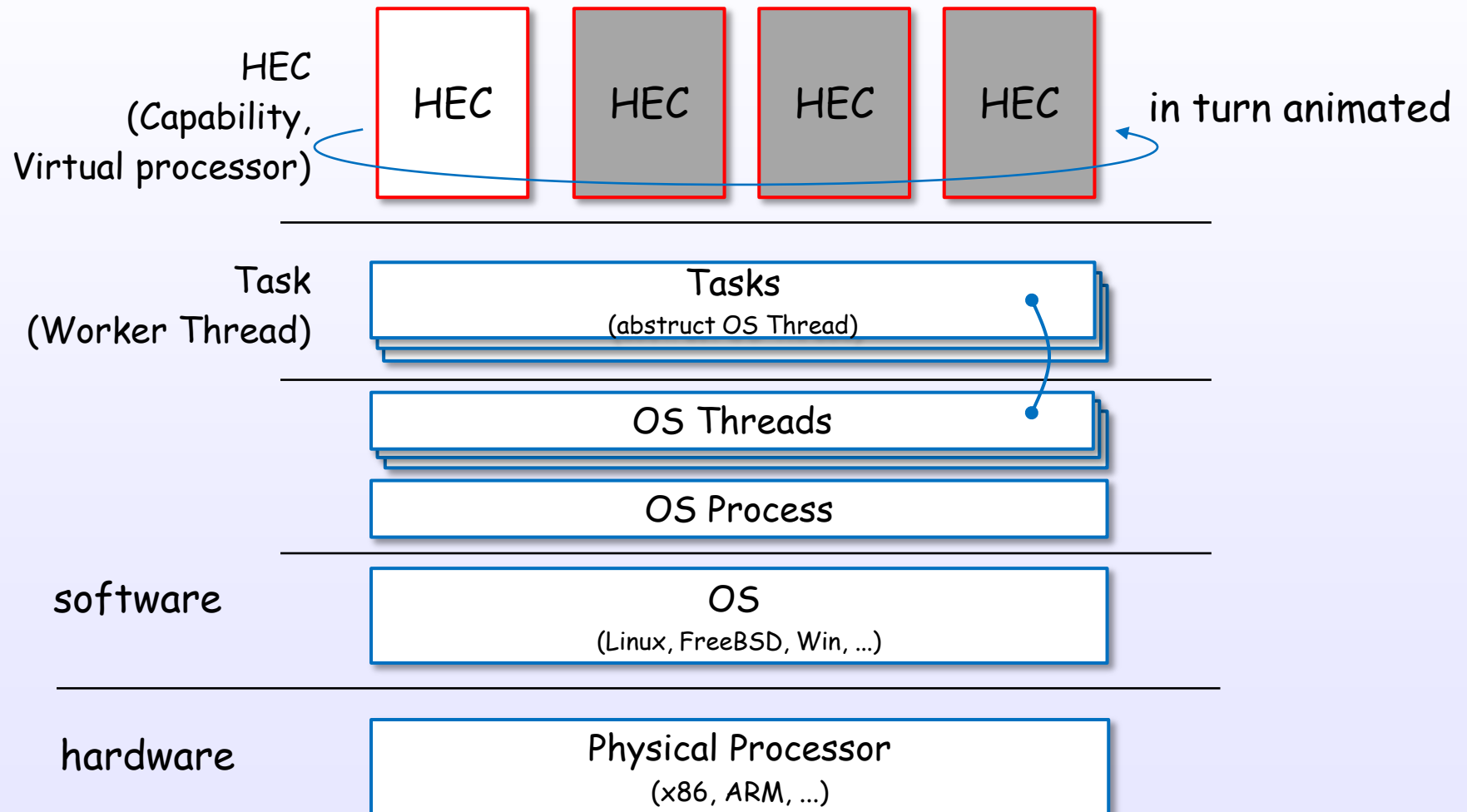


many HECs

multi HEC generated by compile and runtime options :

```
% ghc -rtsopts -threaded
```

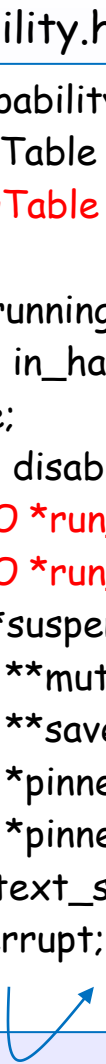
```
% ./xxx +RTS -N4
```



HEC (Capability) data structure

[rts/Capability.h]

```
struct Capability_ {  
    StgFunTable f;  
    StgRegTable r;  
    nat no;  
    Task *running_task;  
    rtsBool in_haskell;  
    nat idle;  
    rtsBool disabled;  
    StgTSO *run_queue_hd;  
    StgTSO *run_queue_tl;  
    InCall *suspended_ccalls;  
    bdescr **mut_lists;  
    bdescr **saved_mut_lists;  
    bdescr *pinned_object_block;  
    bdescr *pinned_object_blocks;  
    int context_switch;  
    int interrupt;  
  
#if defined(THREADED_RTS)  
    Task *spare_workers;  
    nat n_spare_workers;  
    Mutex lock;  
    Task *returning_tasks_hd;  
    Task *returning_tasks_tl;  
    Message *inbox;  
    SparkPool *sparks;  
    SparkCounters spark_stats;  
#endif  
  
    W_ total_allocated;  
    StgTVarWatchQueue *free_tvar_watch_queues;  
    StgInvariantCheckQueue *free_invariant_check_queues;  
    StgTRecChunk *free_trec_chunks;  
    StgTRecHeader *free_trec_headers;  
    nat transaction_tokens;  
}
```

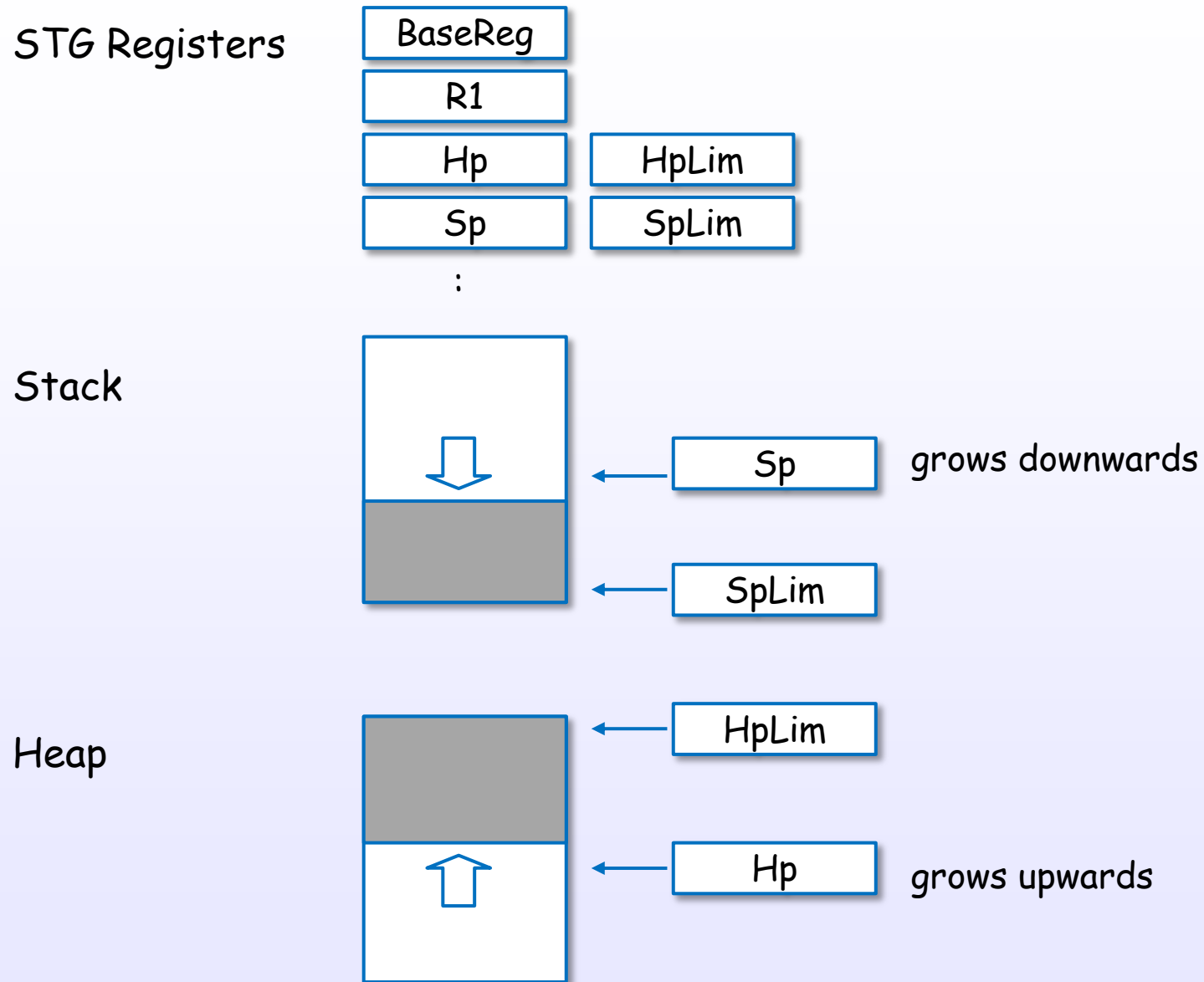


HEC (Capability) has Register table and Run queue and ...

HEC (Capability) is initialized in `initCapabilities ()` [rts/rts/Capability.c]

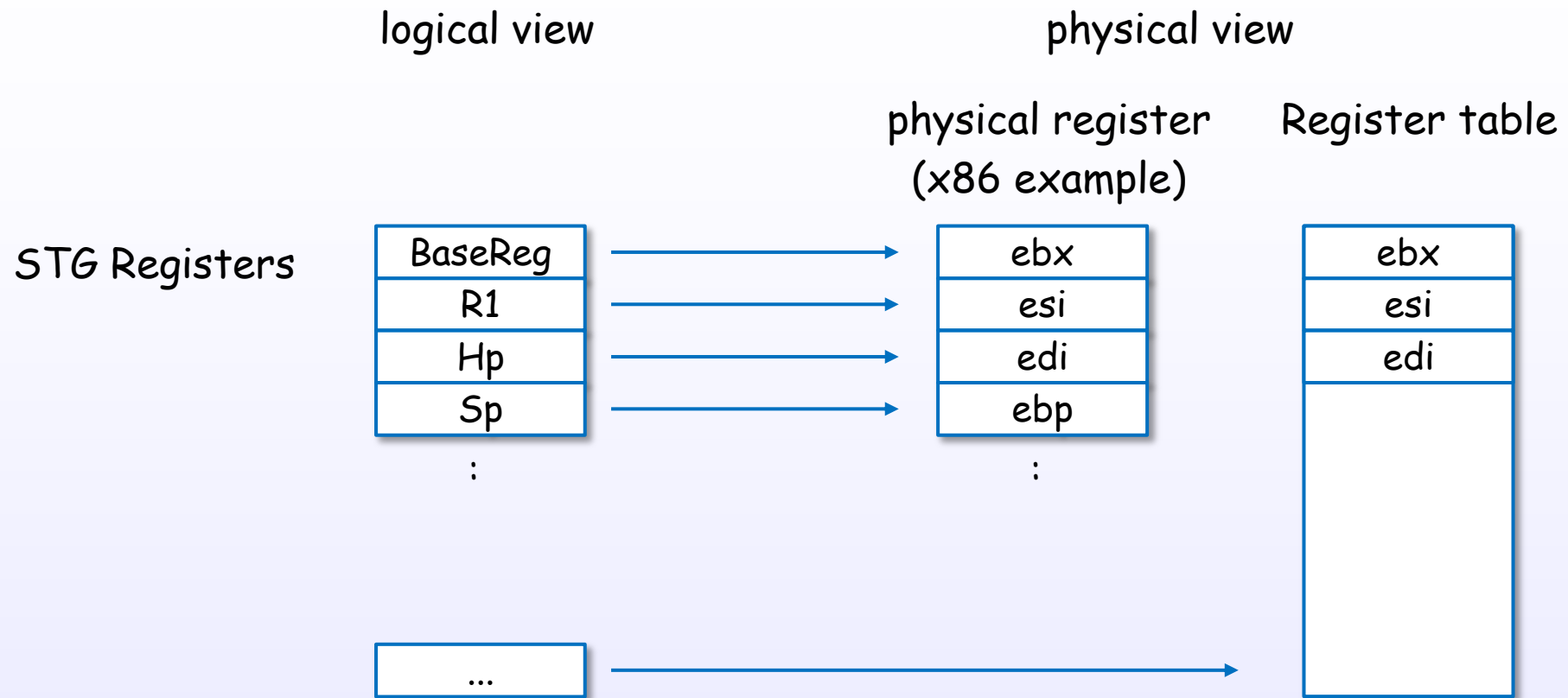
STG-machine

STG-machine

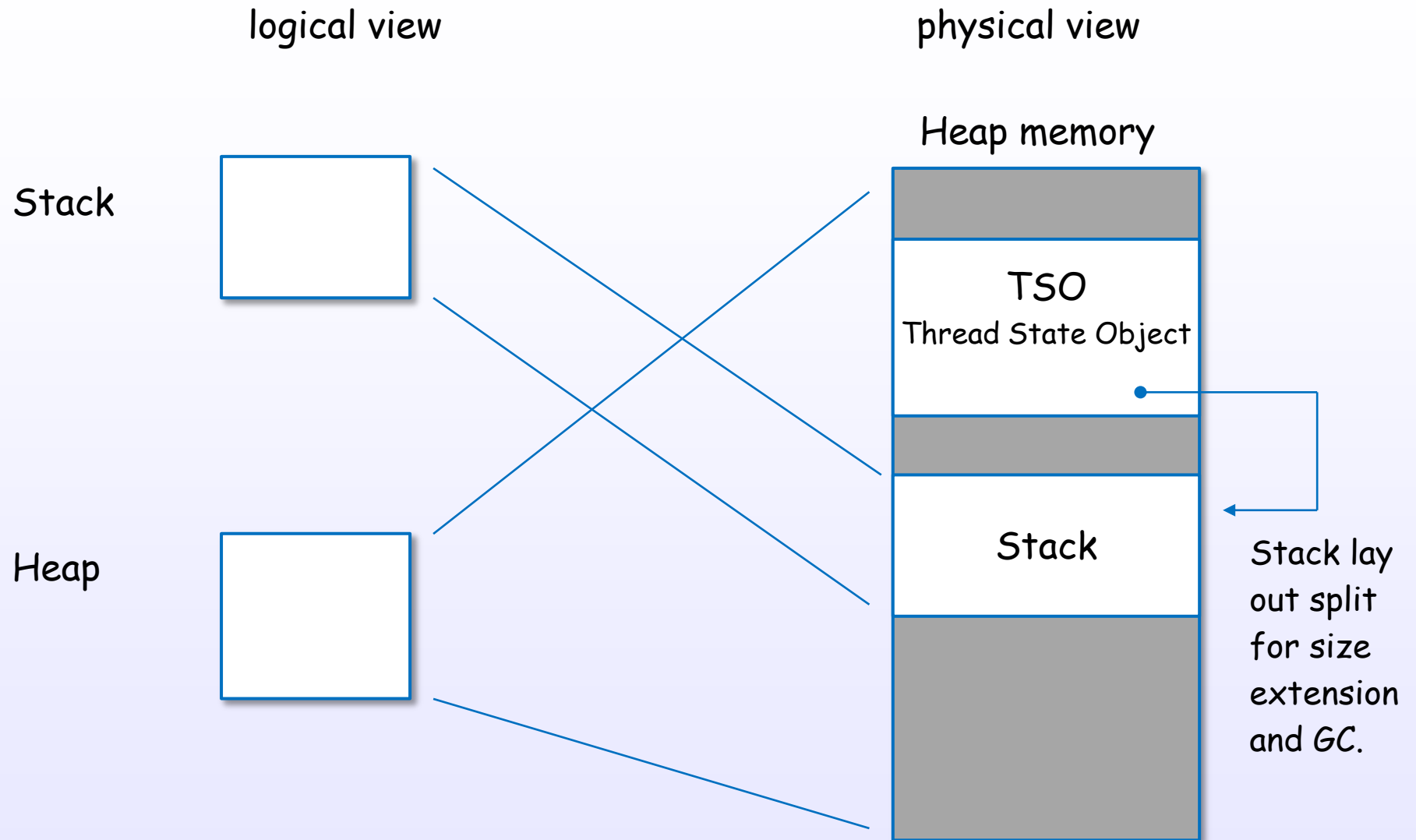


Real Haskell code executed in STG semantics.

STG-machine mapped to physical processor



STG-machine mapped to physical processor



Stack is in TSO object.
TSO object in heap.

TSO data structure

[includes/rts/storage/TSO.h]

```
typedef struct StgTSO_  
  StgHeader          header;  
  struct StgTSO_*     _link;  
  struct StgTSO_*     global_link;  
  struct StgStack_*   *stackobj;  
  StgWord16           what_next;  
  StgWord16           why_blocked;  
  StgWord32           flags;  
  StgTSOBlockInfo     block_info;  
  StgThreadID         id;  
  StgWord32           saved_errno;  
  StgWord32           dirty;  
  struct InCall_*      bound;  
  struct Capability_*  cap;  
  struct StgTRecHeader_* trec;  
  struct MessageThrowTo_* blocked_exceptions;  
  struct StgBlockingQueue_* bq;  
  StgWord32 tot_stack_size;  
} *StgTSOPtr;
```

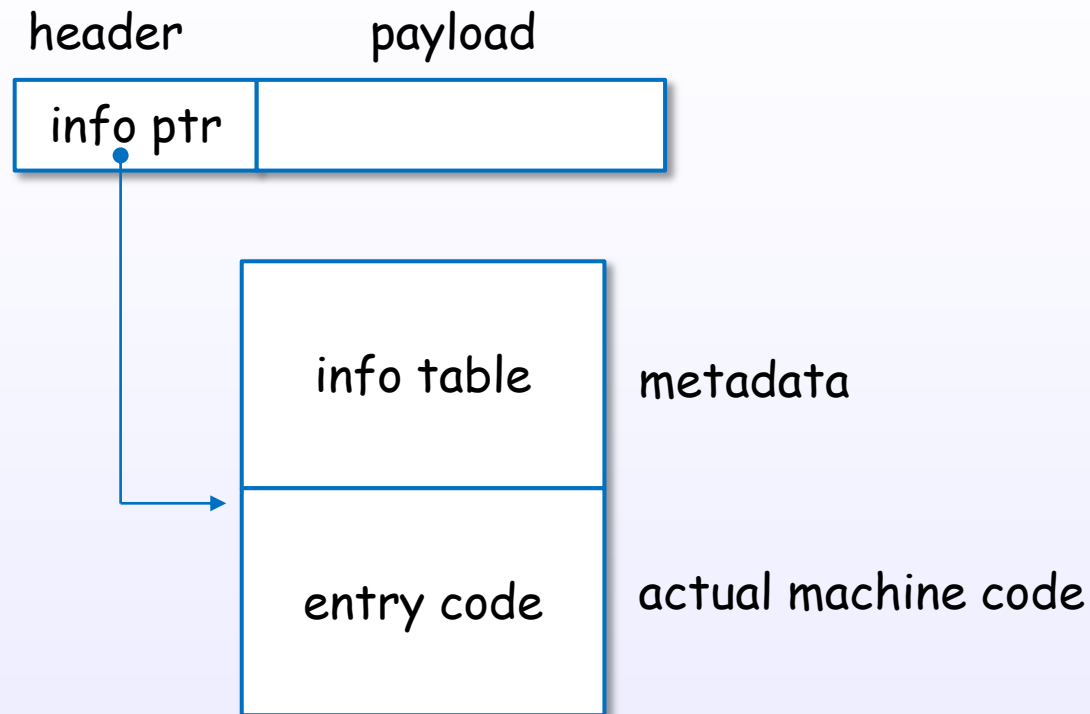
← link to stack object

TSO object is **only ~17words + stack**. Lightweight.

Heap object in STG-machine

Heap object (closure)

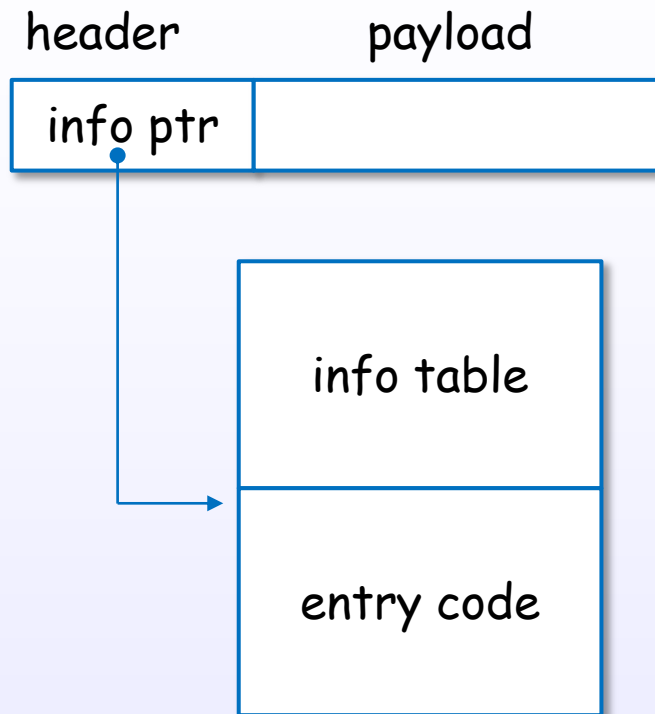
logical view



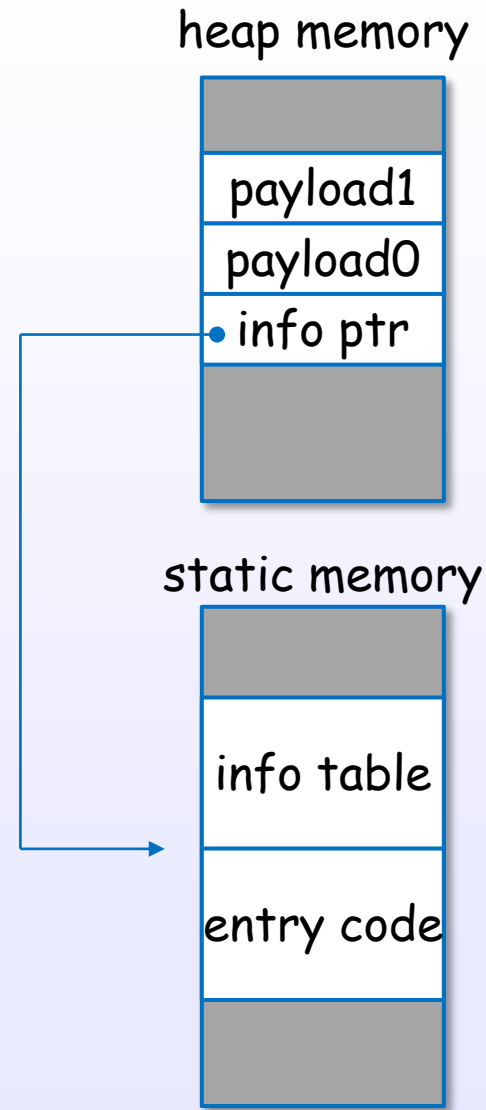
Closure (header + payload) + Info Table + Entry Code

Heap object (closure)

logical view

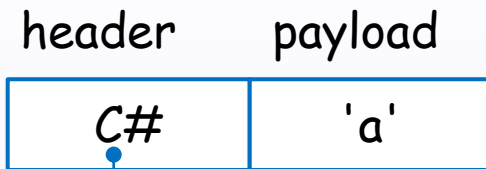


physical view



Closure examples : Char, Int

'a' :: Char



info
ptr

GHC.Types.C#_static_info

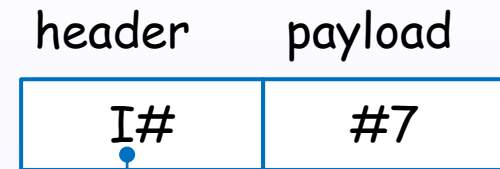
layout : 0_1
type : CONSTR
bitmap :

info table

inc %esi
jmp *0x0(%ebp)

entry code

7 :: Int



info
ptr

GHC.Types.I#_static_info

layout : 0_1
type : CONSTR
bitmap :

info table

inc %esi
jmp *0x0(%ebp)

entry code

Closure example code

[Example.hs]

```
module Example where
value1 :: Int
value1 = 7
```

STG

[ghc -O -ddump-stg Example.hs]

```
Example.value1 :: GHC.Types.Int
[GblId, Caf=NoCafRefs, Str=DmdType m, Unf=OtherCon []] =
  NO_CCS GHC.Types.I#! [7];
```

Cmm

[ghc -O -ddump-opt-cmm Example.hs]

```
section "data" { __stginit_main:Example:
}
```

```
section "data" {
  Example.value1 closure:
    const GHC.Types.I#_static_info;
    const 7;
}
```

asm

[ghc -O -ddump-asm Example.hs]

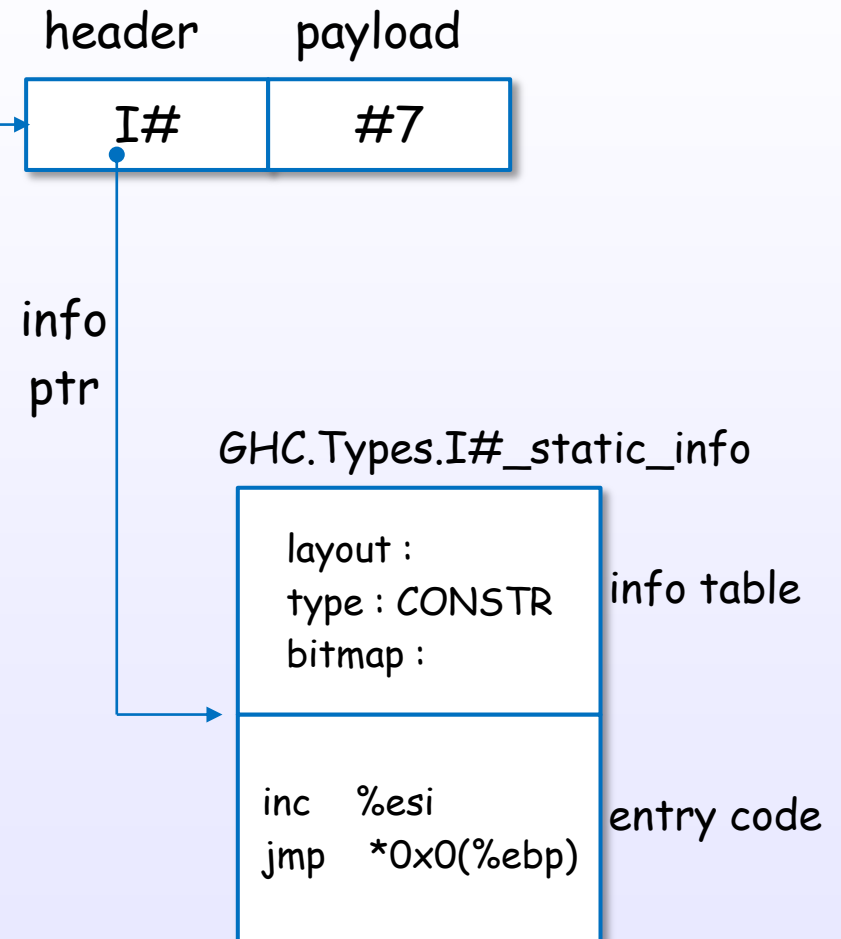
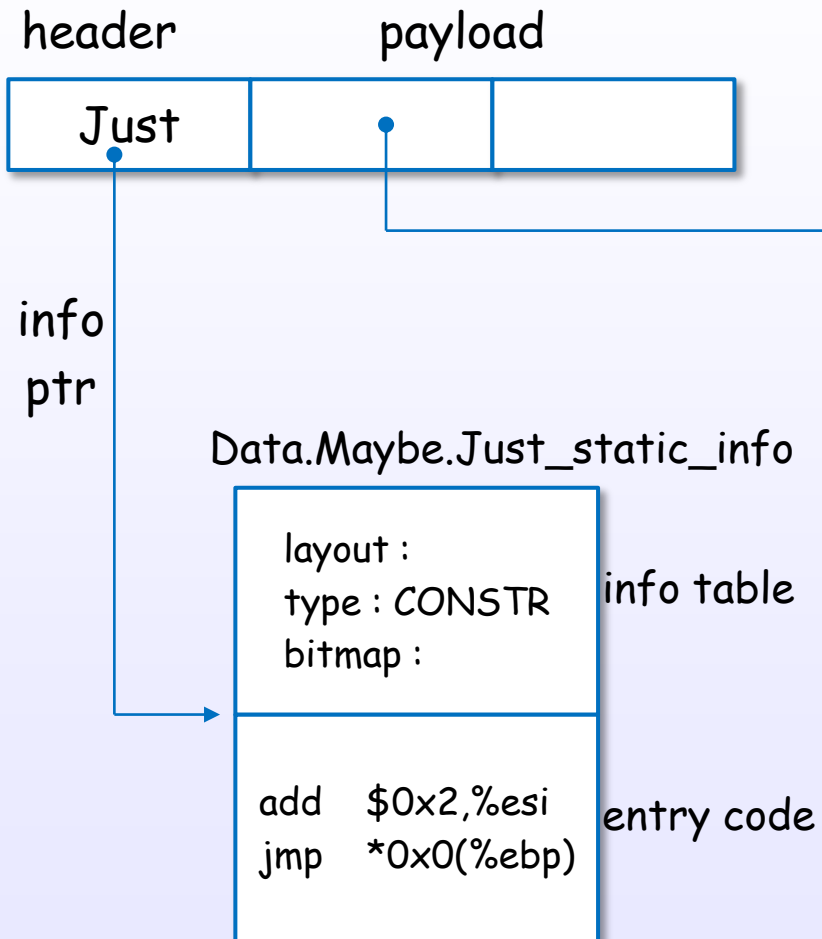
```
.data
    .align 4
    .align 1
    .globl __stginit_main:Example
    __stginit_main:Example:
    .data
        .align 4
        .align 1
        .globl Example.value1_closure
    Example.value1_closure:
        .long GHC.Types.I#_static_info
        long 7
```

```
.section .data
    .align 4
    .align 1
    SMd_srt:
```

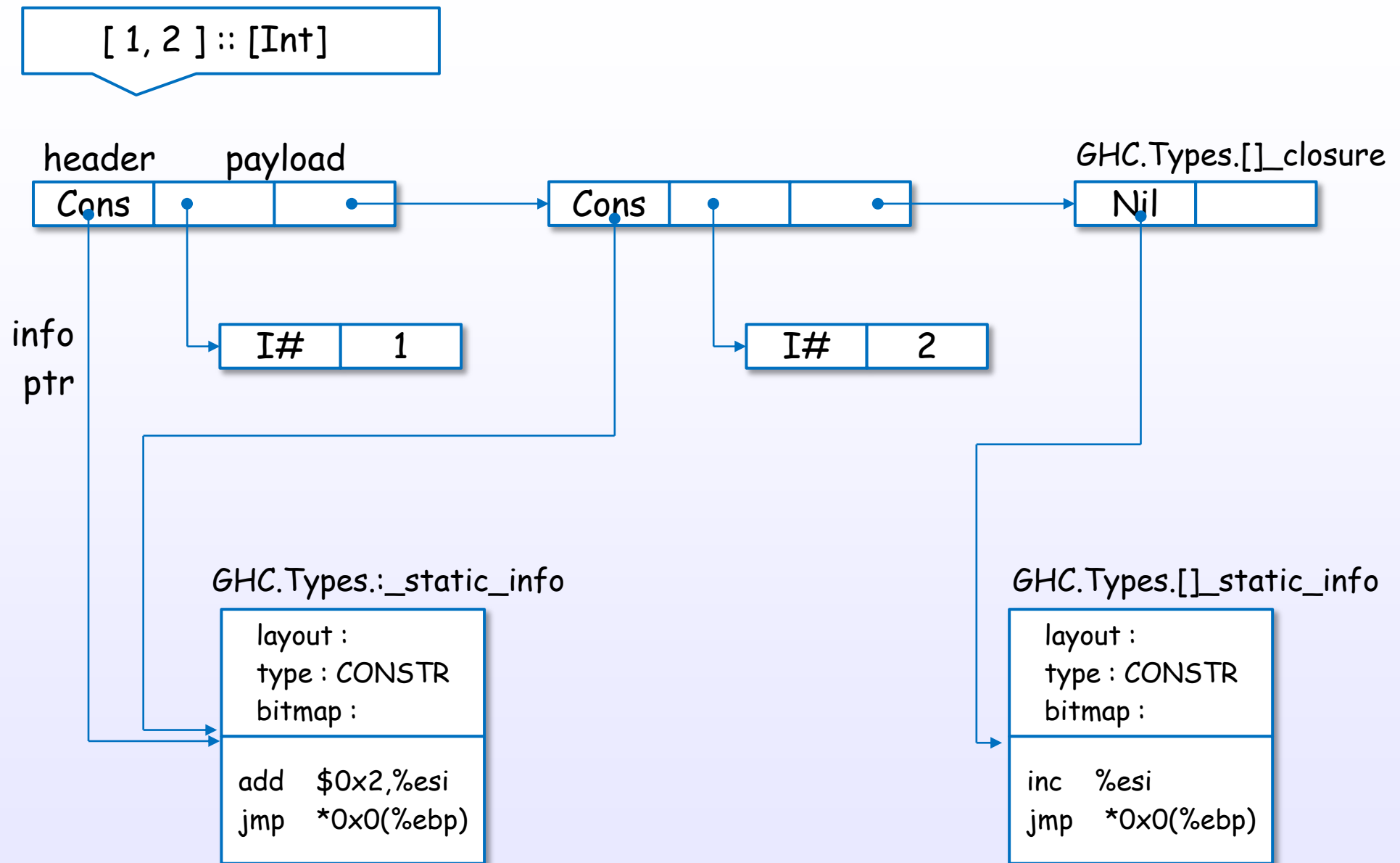
header	payload
I#	#7

Closure examples : Maybe

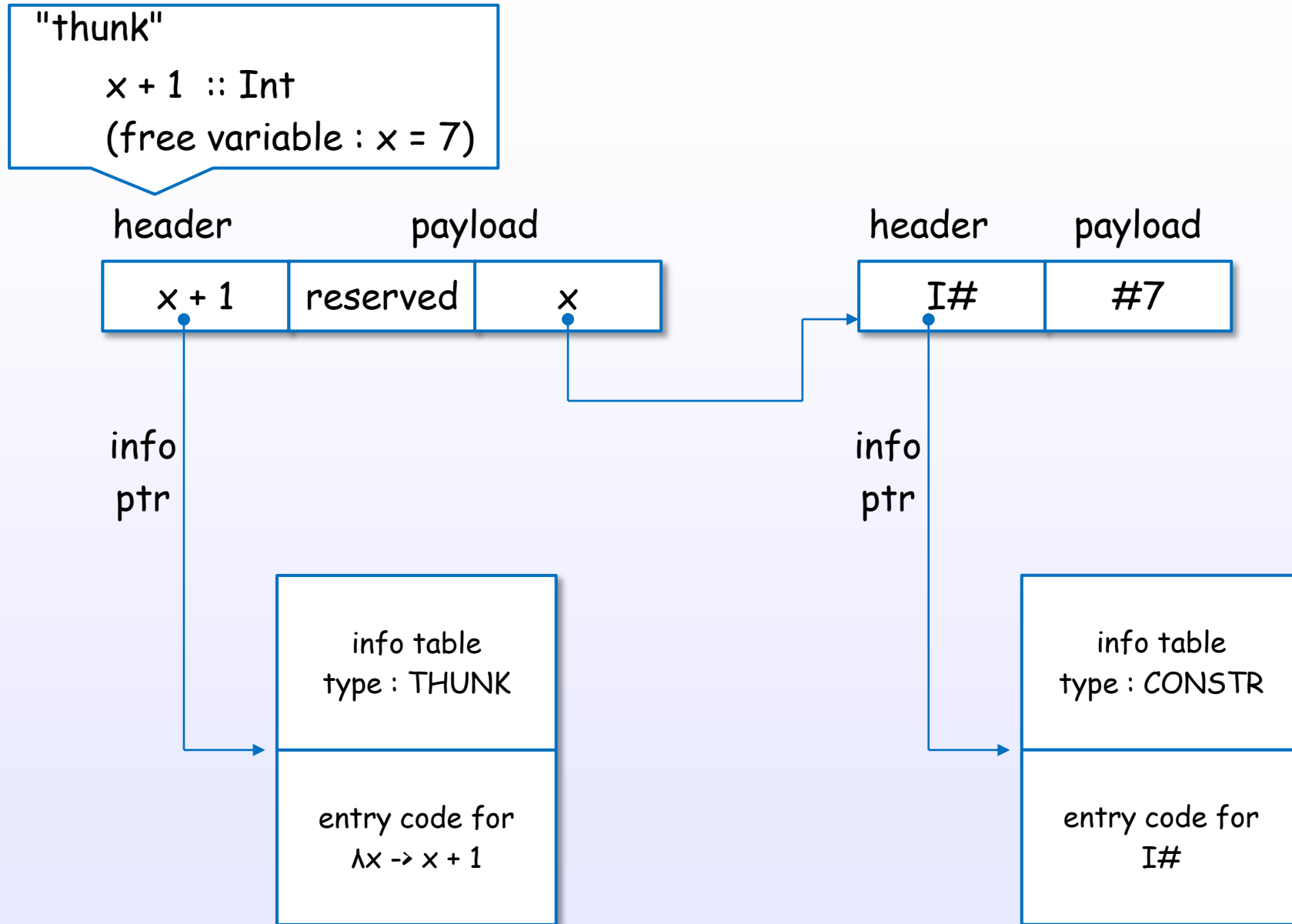
Just 7 :: Maybe Int



Closure examples : List

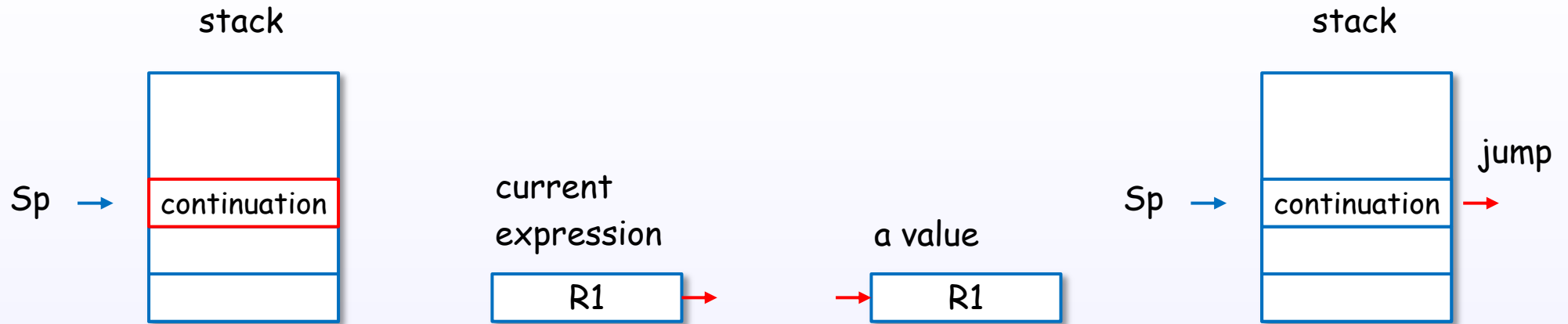


Closure examples : Thunk



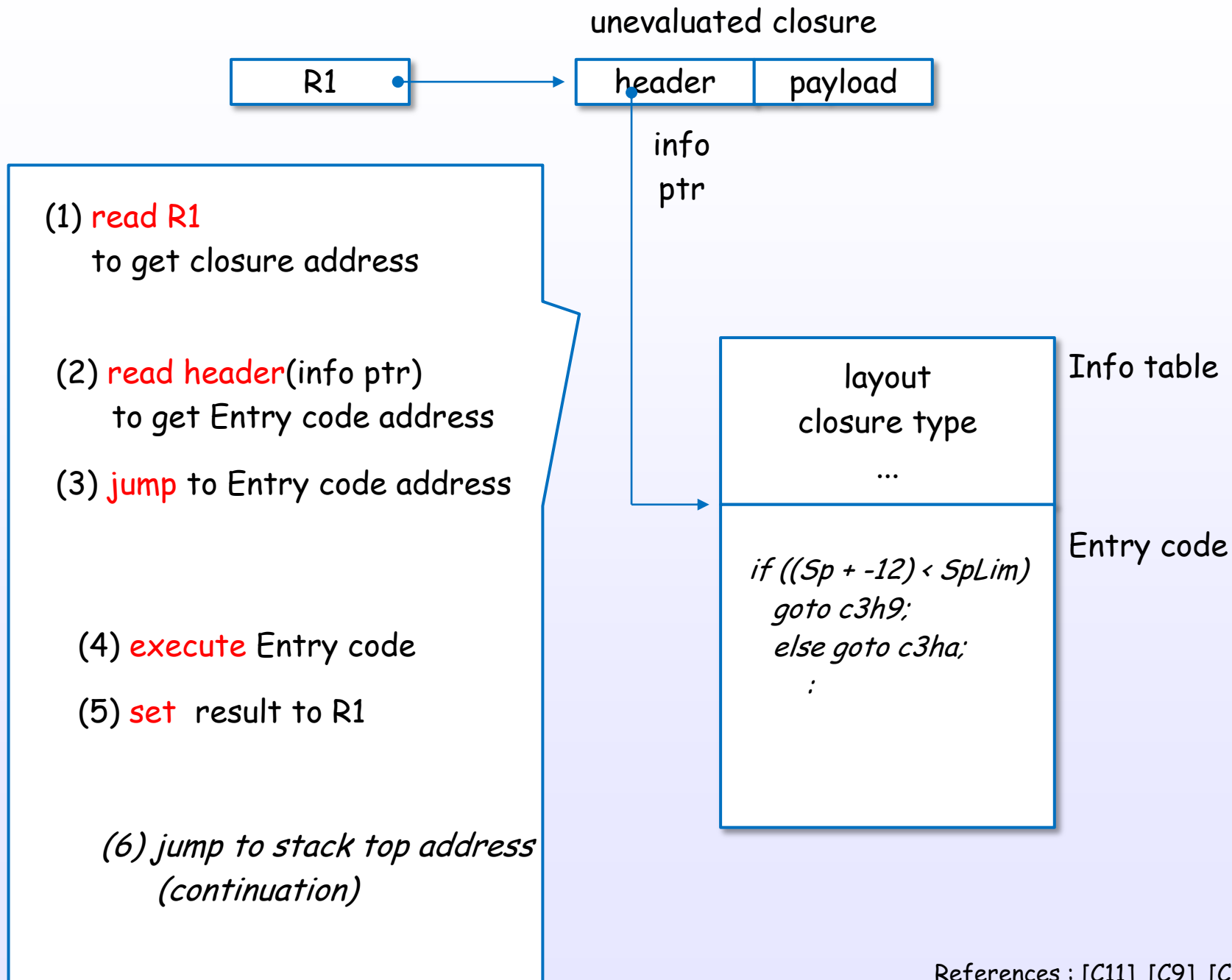
STG-machine evaluation

STG evaluation flow



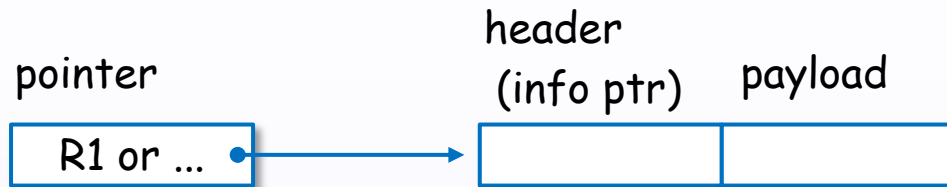
- (1) push continuation code (next code) to stack top
- (2) enter to R1 closure
- (3) set result to R1
- (4) jump (return) to stack top code
- (5) repeat from (1)

Enter to closure



Pointer tagging

Pointer tagging



pointer



... This closure is unevaluated.



... evaluated closure;
1st constructor value or evaluated.
(for instance: "Nothing")



... evaluated closure; 2nd constructor value.
(for instance: "Just xx")



... evaluated closure; 3rd constructor value.

* 32bit machine case

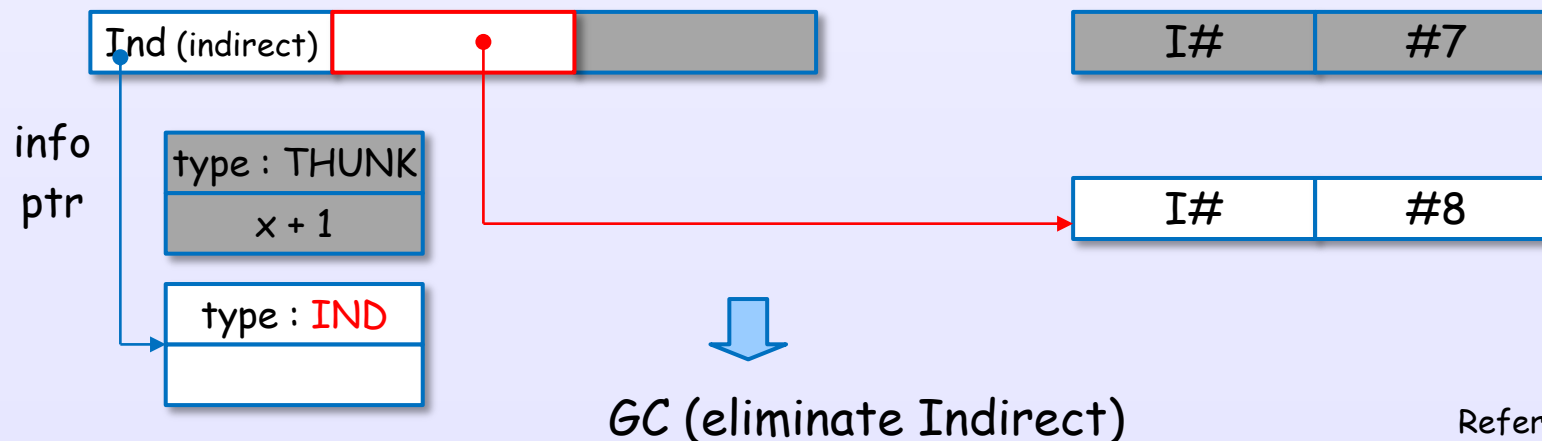
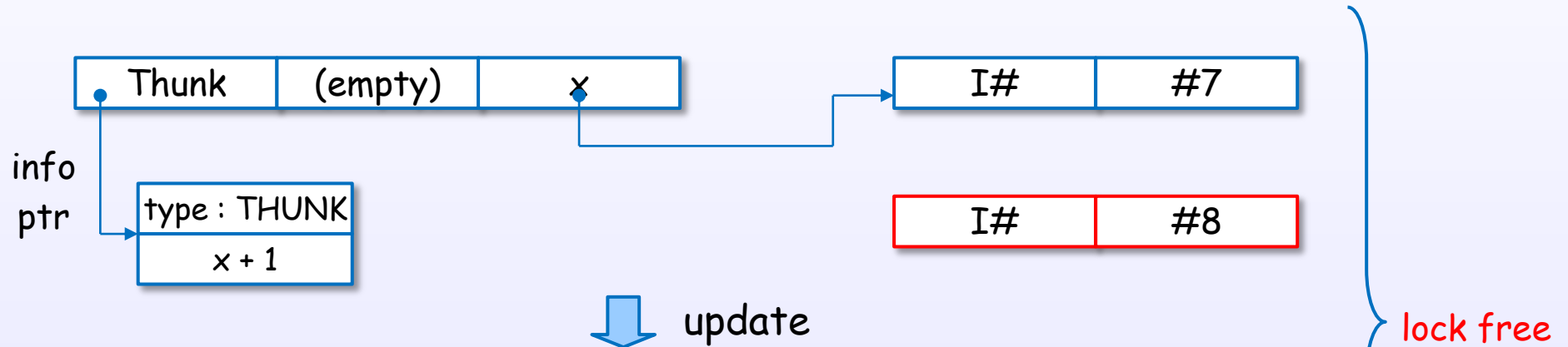
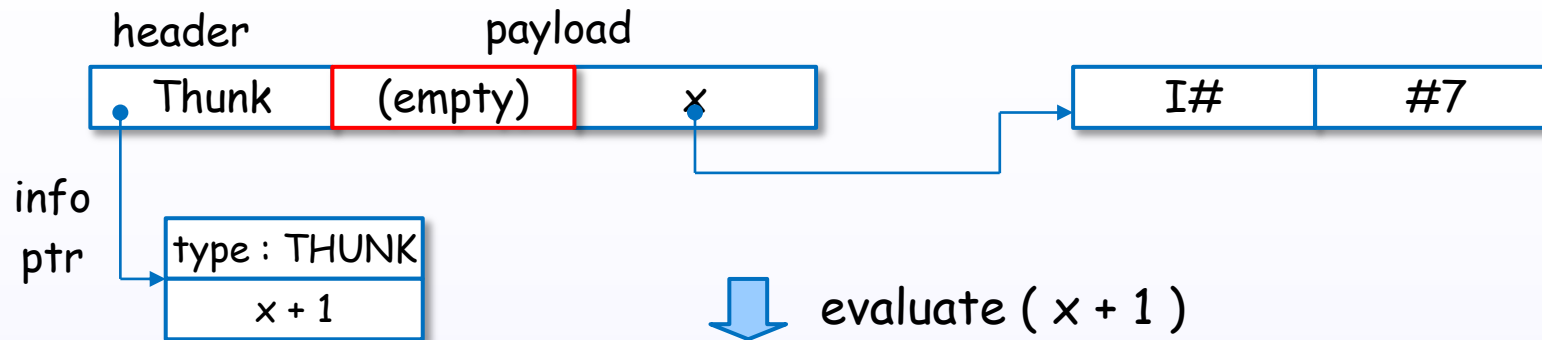
quick judgment!

check only pointer's lower bits without evaluating the closure.

Think and update

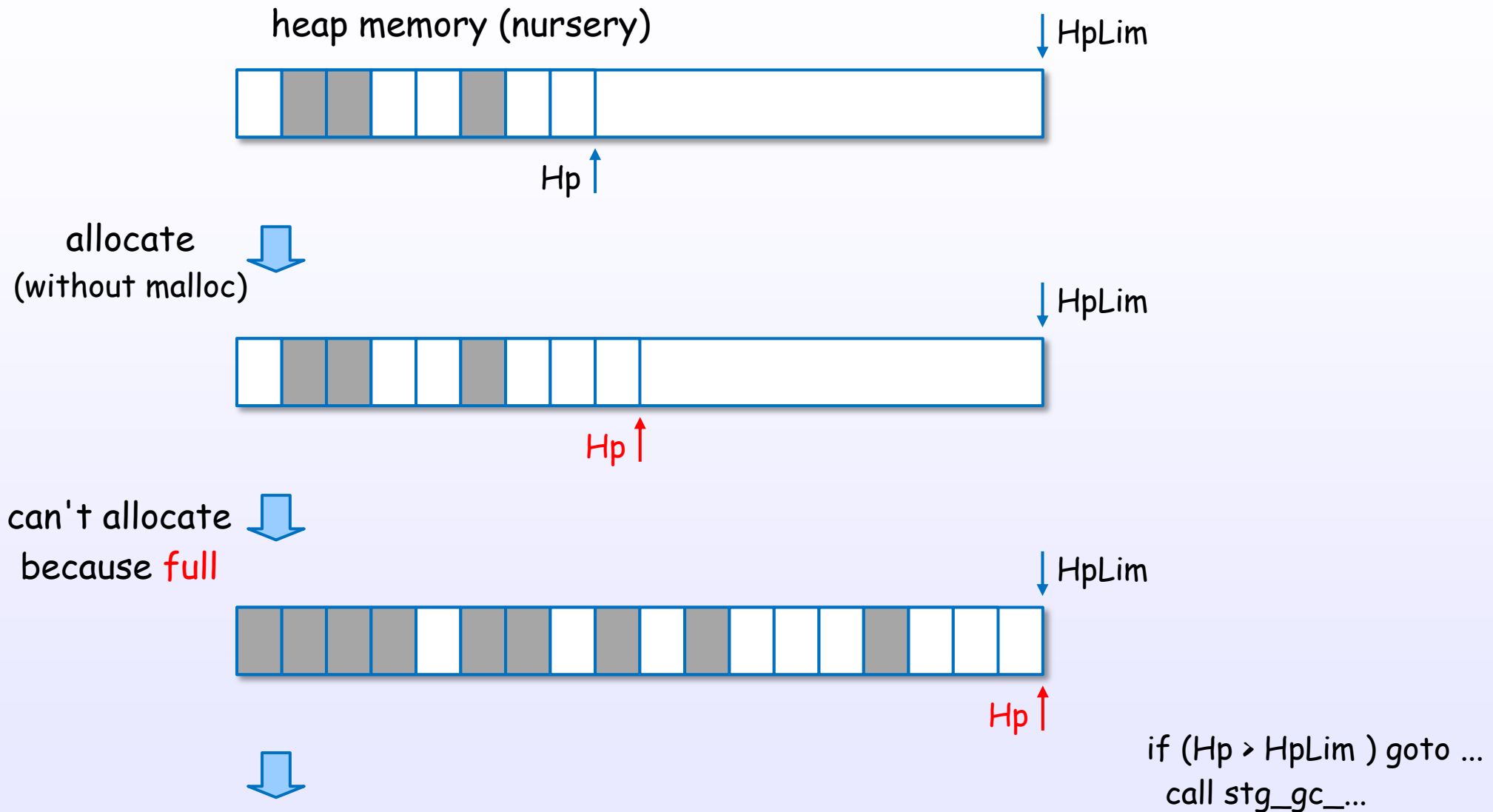
Thunk and update

"thunk" $x + 1 :: \text{Int}$ (free variable : $x = 7$)

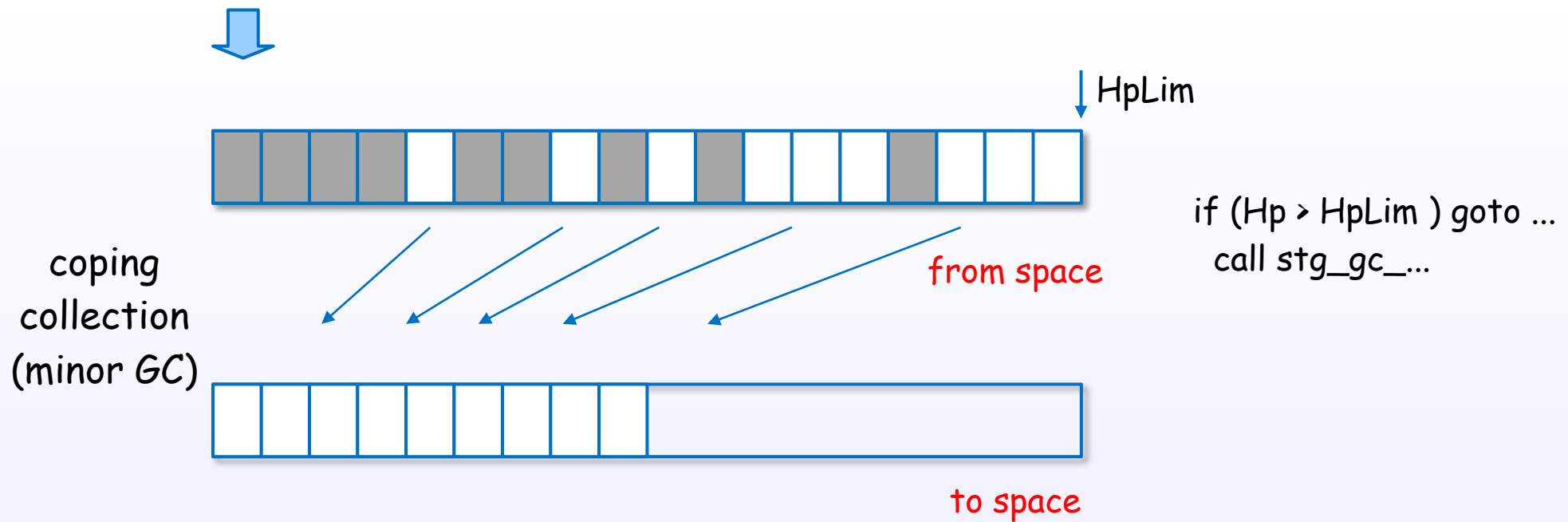


Allocate and free heap objects

Allocate heap objects

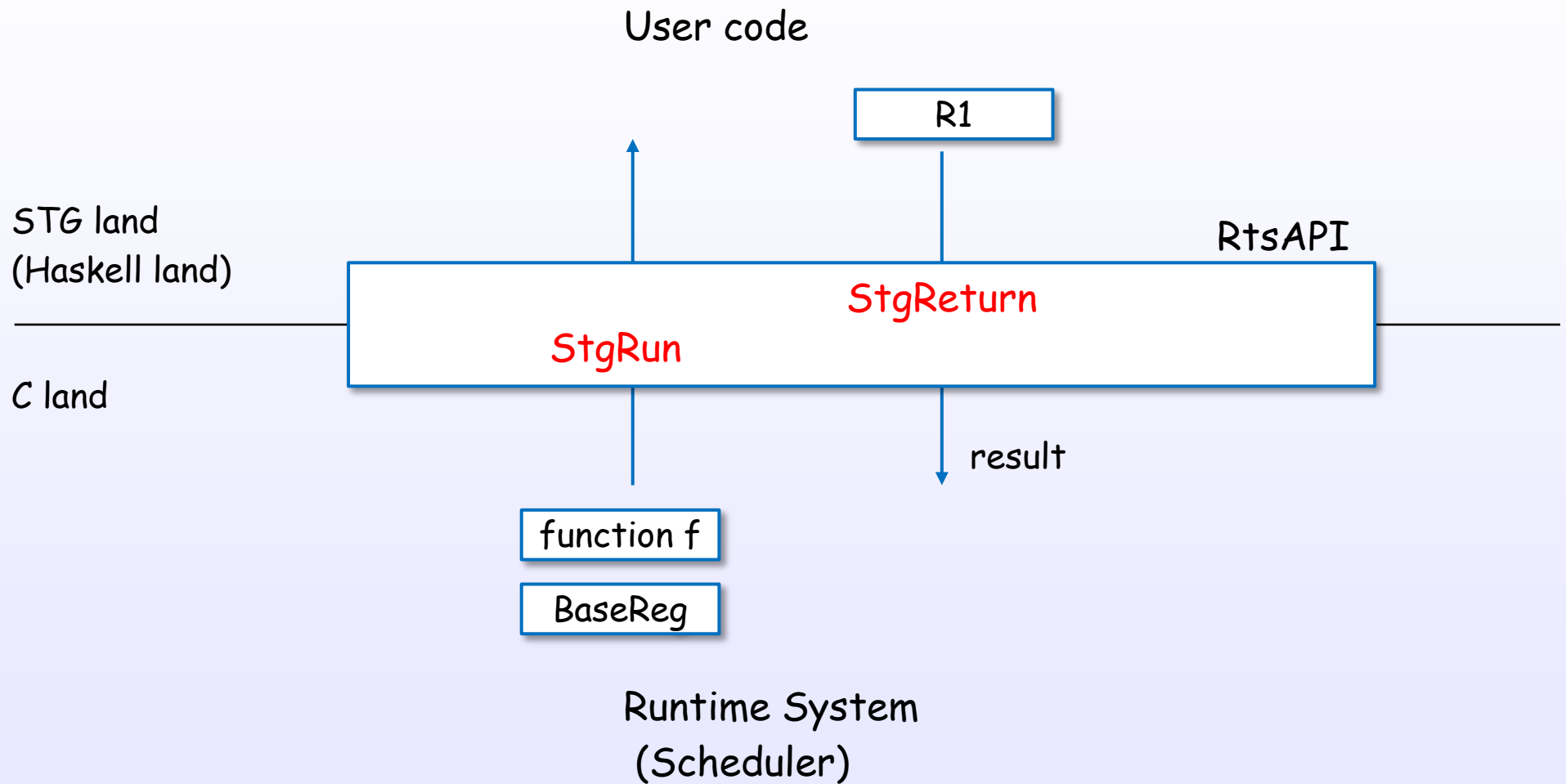


free and collect heap objects



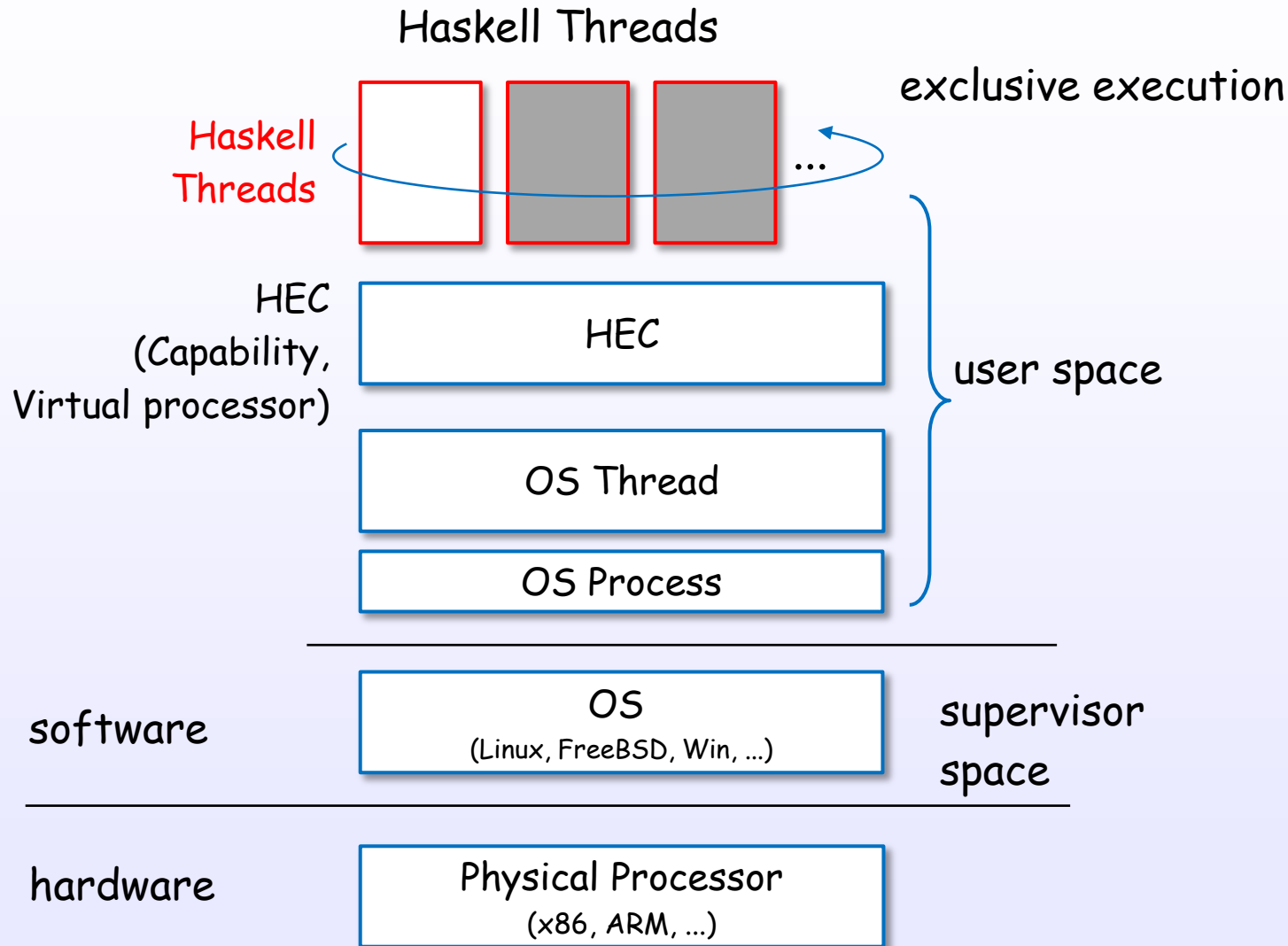
STG - C land interface

STG (Haskell) land - C land interface

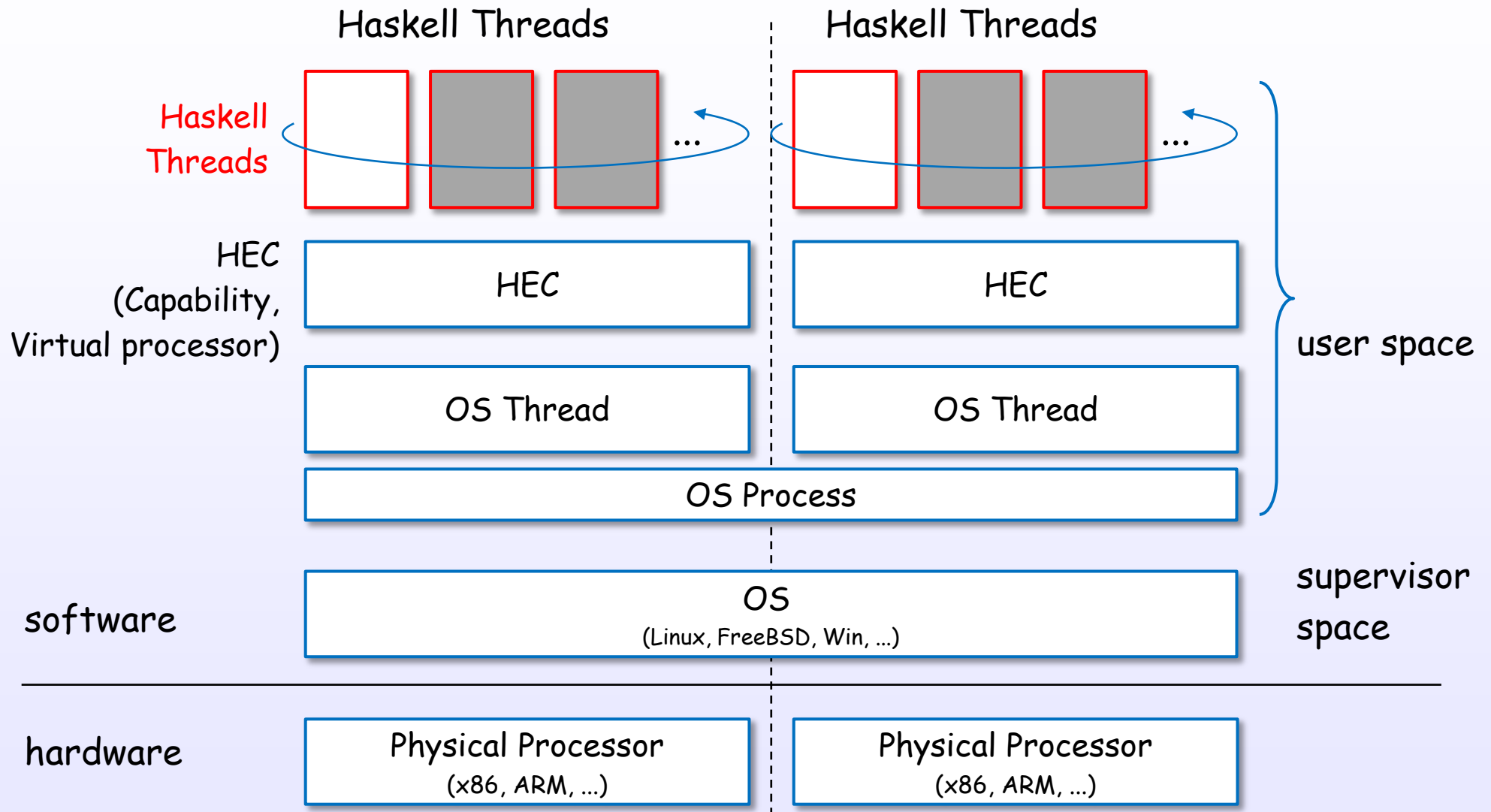


Thread

Thread layer (single core)



Thread layer (multi core)



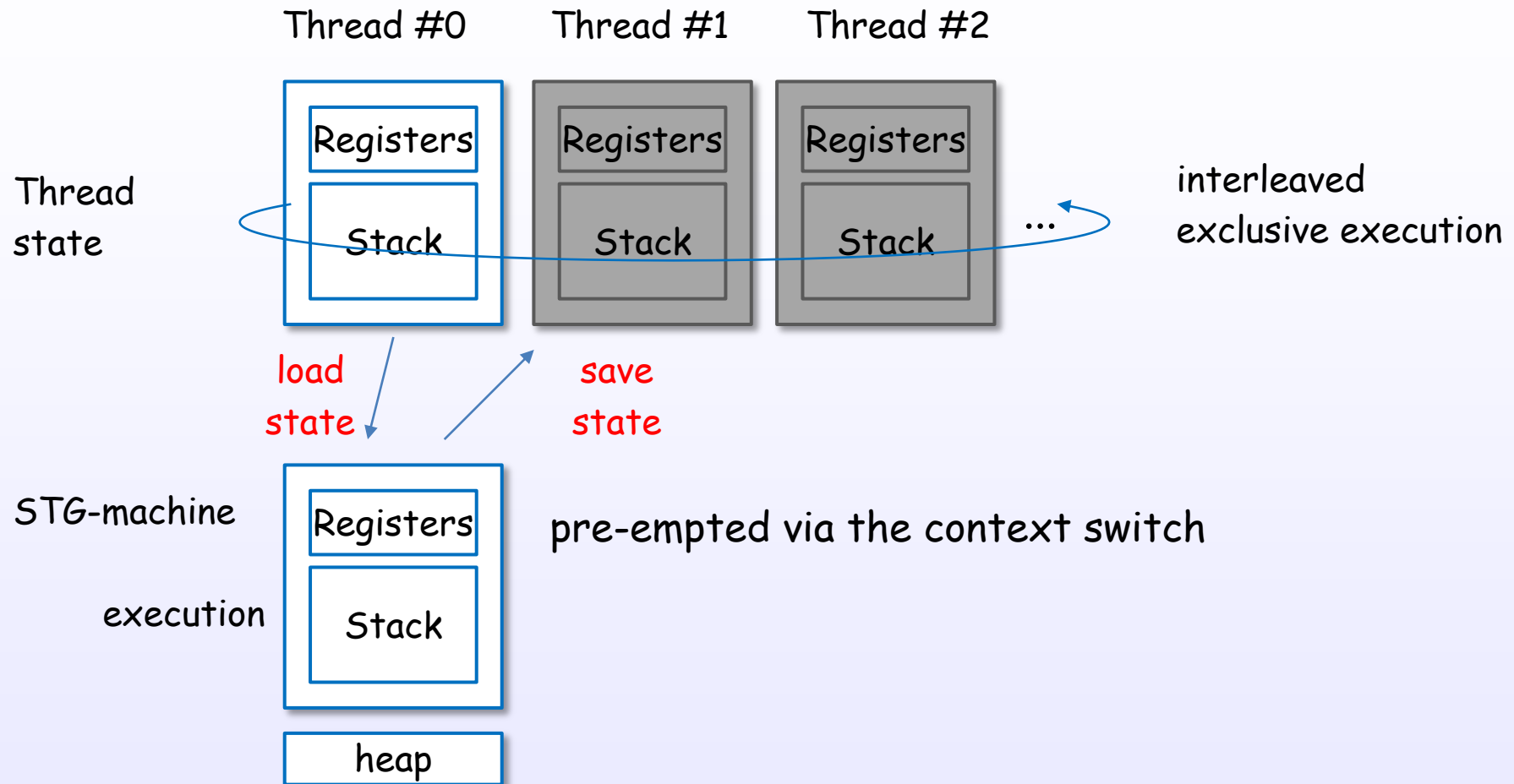
*Threaded option case (ghc -threaded)

References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14]

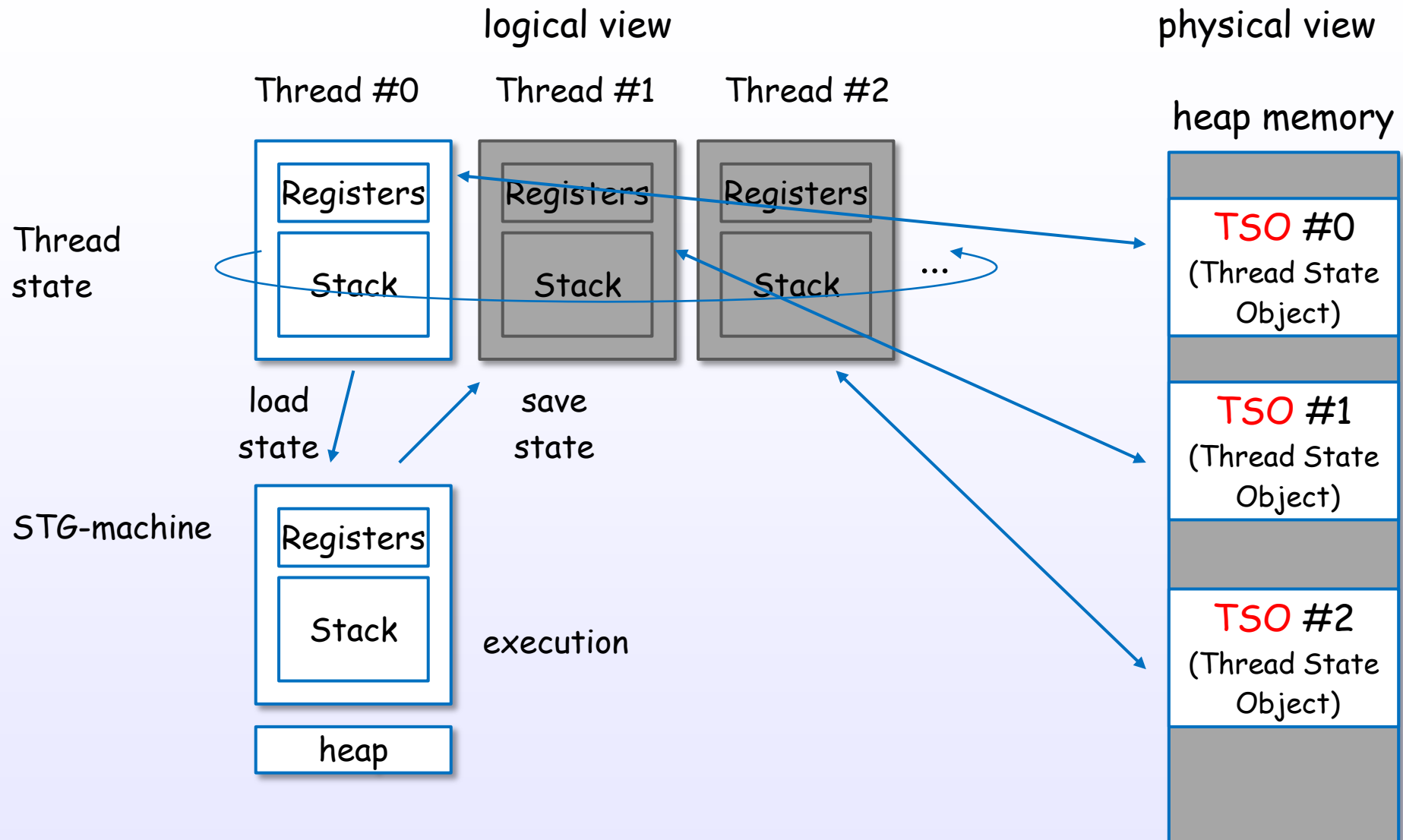
Thread context switch

Threads and context switch

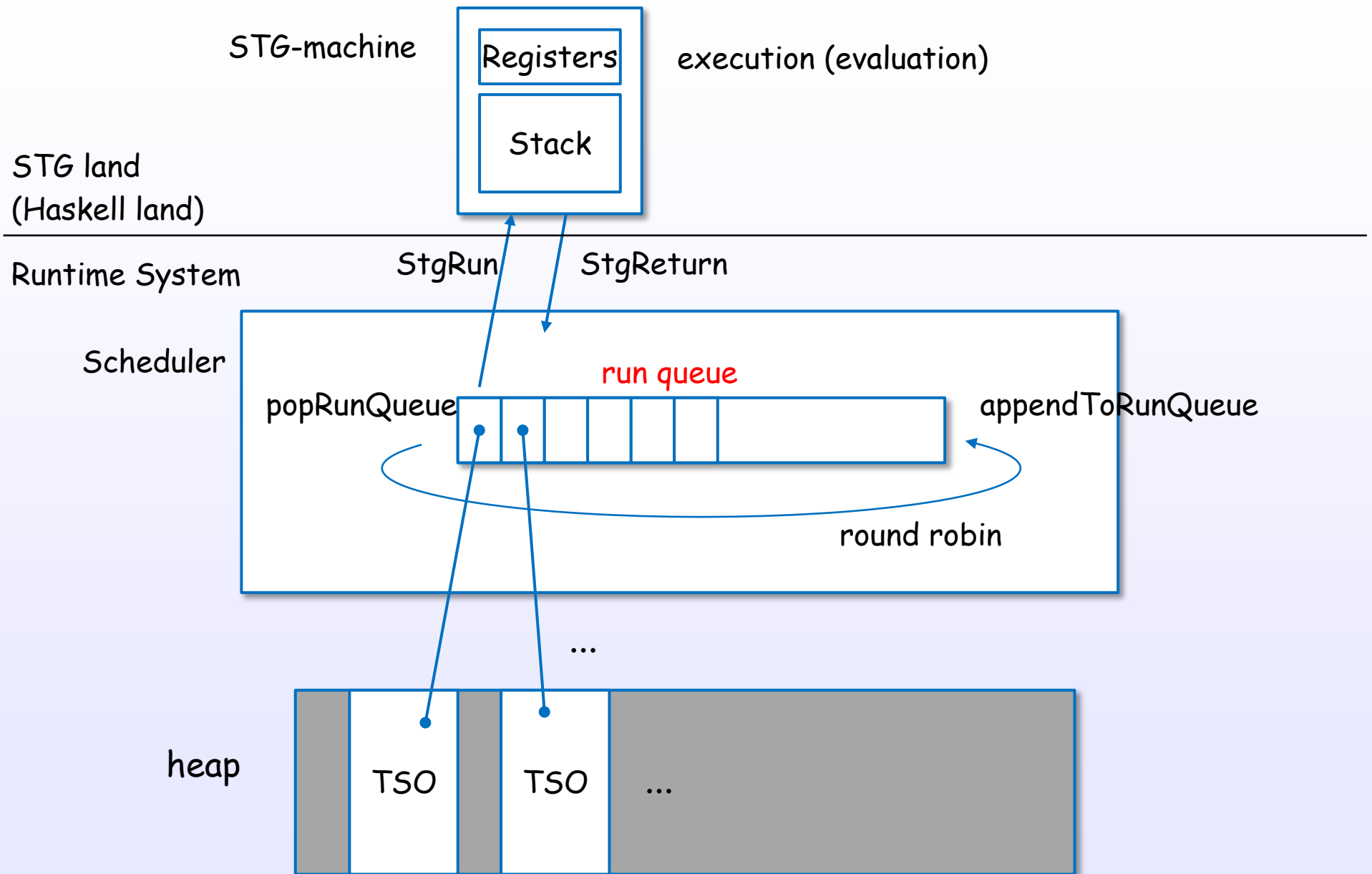
logical view



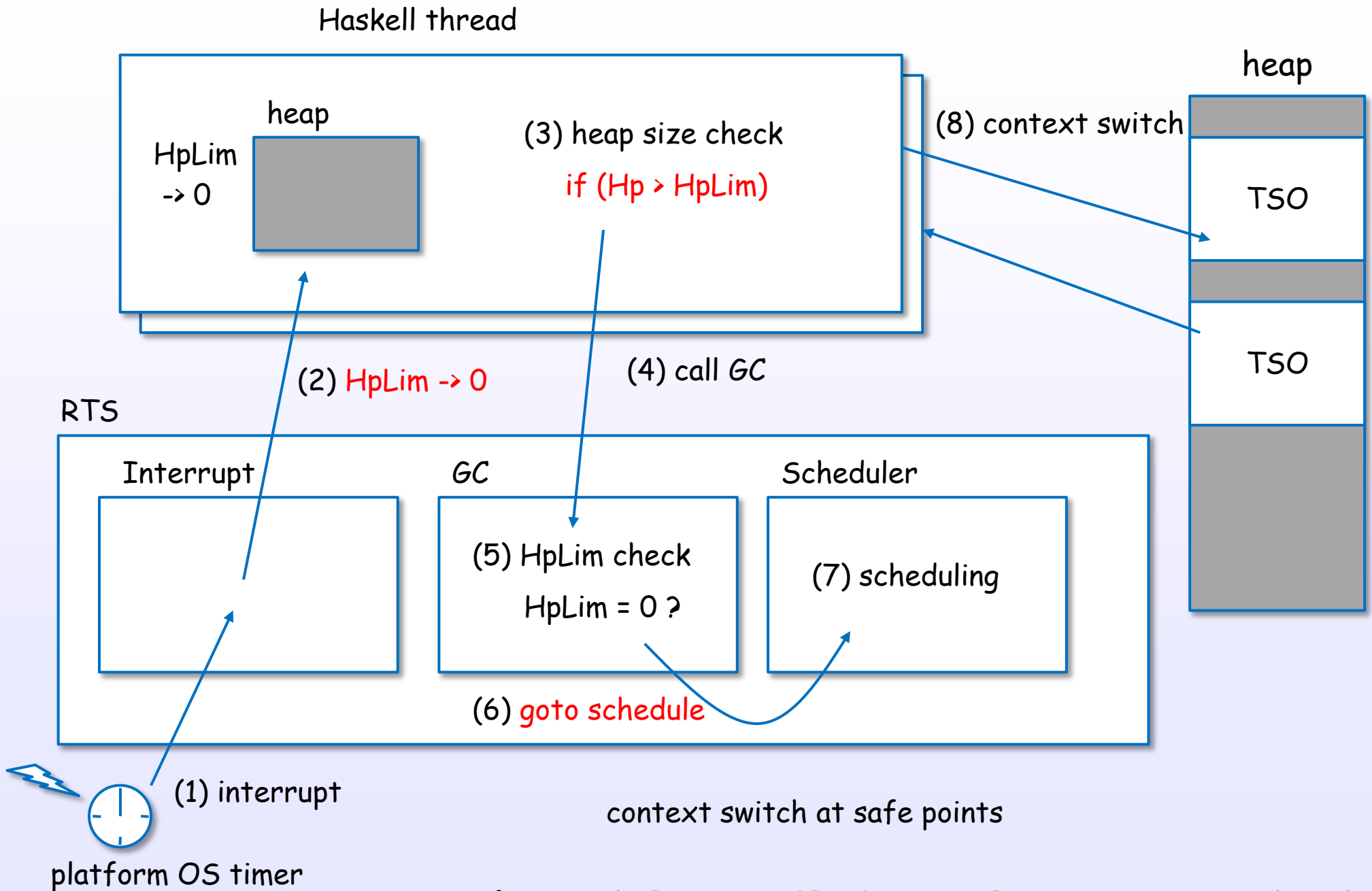
Threads and TSOs



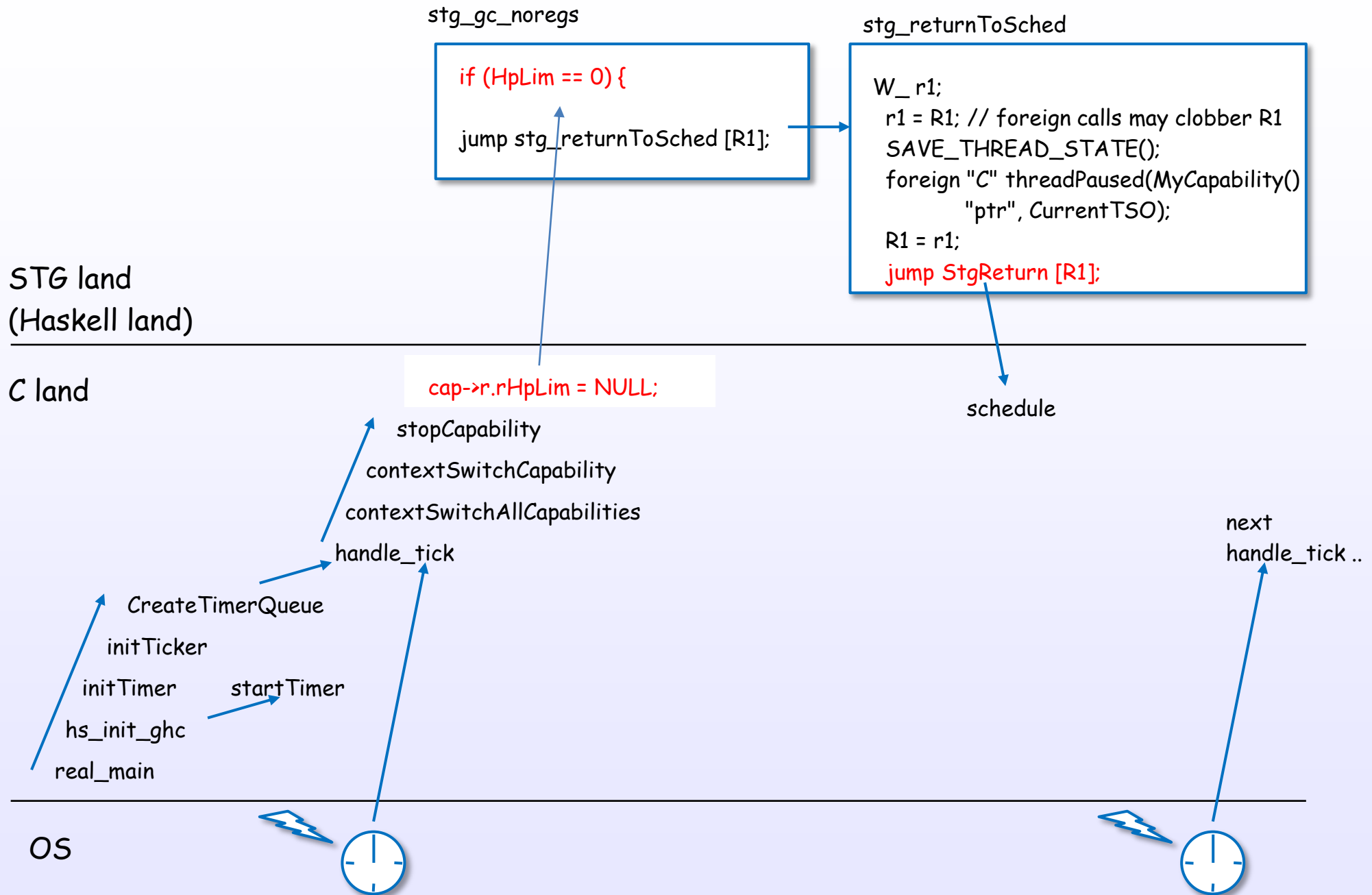
Scheduling threads



Context switch flow



Context switch flow (code)



*Windows case

References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14], [S24]

Creating main and sub threads

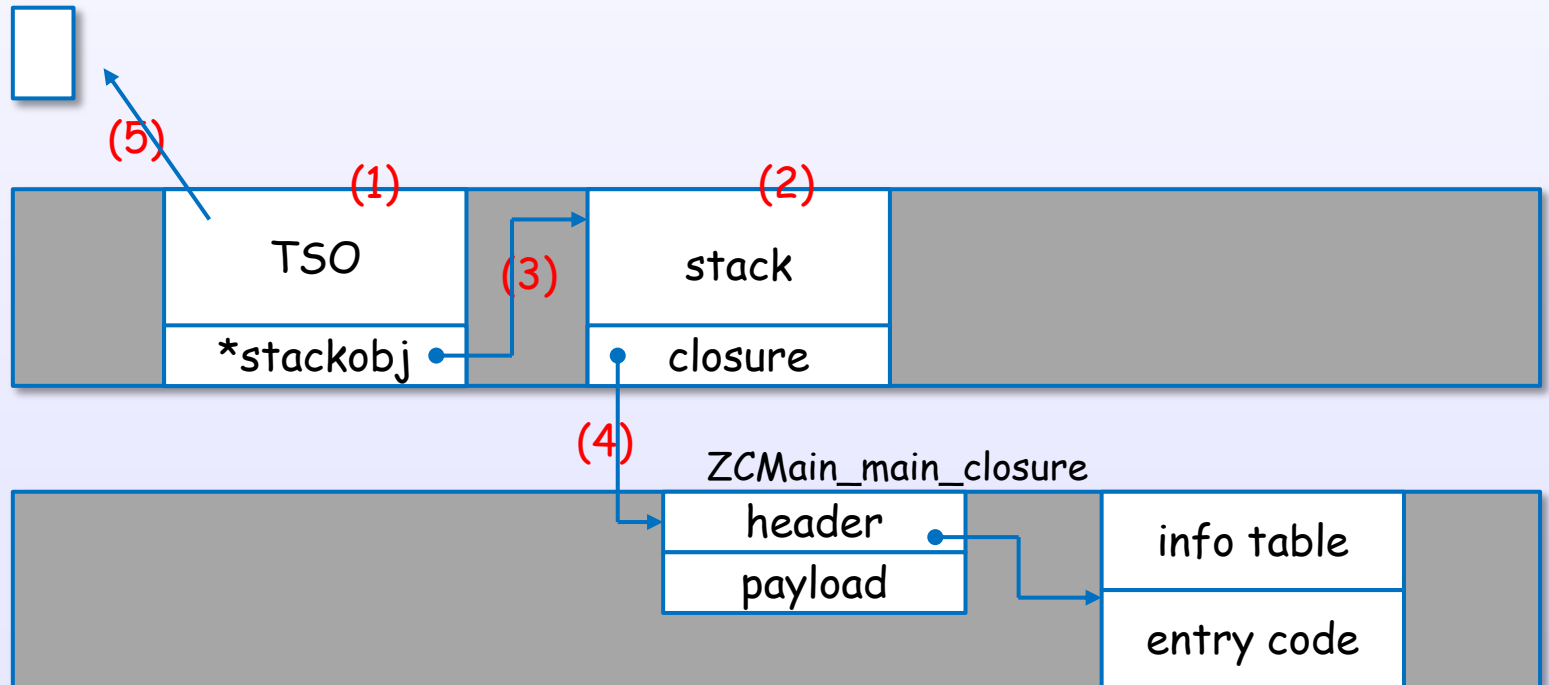
Create main thread

C land

Runtime system bootstrap code [rts/RtsAPI.c]

```
rts_evalLazyIO
  createIOThread
    createThread ... (1), (2), (3)
    pushClosure ... (4)
  scheduleWaitThread
    appendToRunQueue ... (5)
```

scheduler
run queue



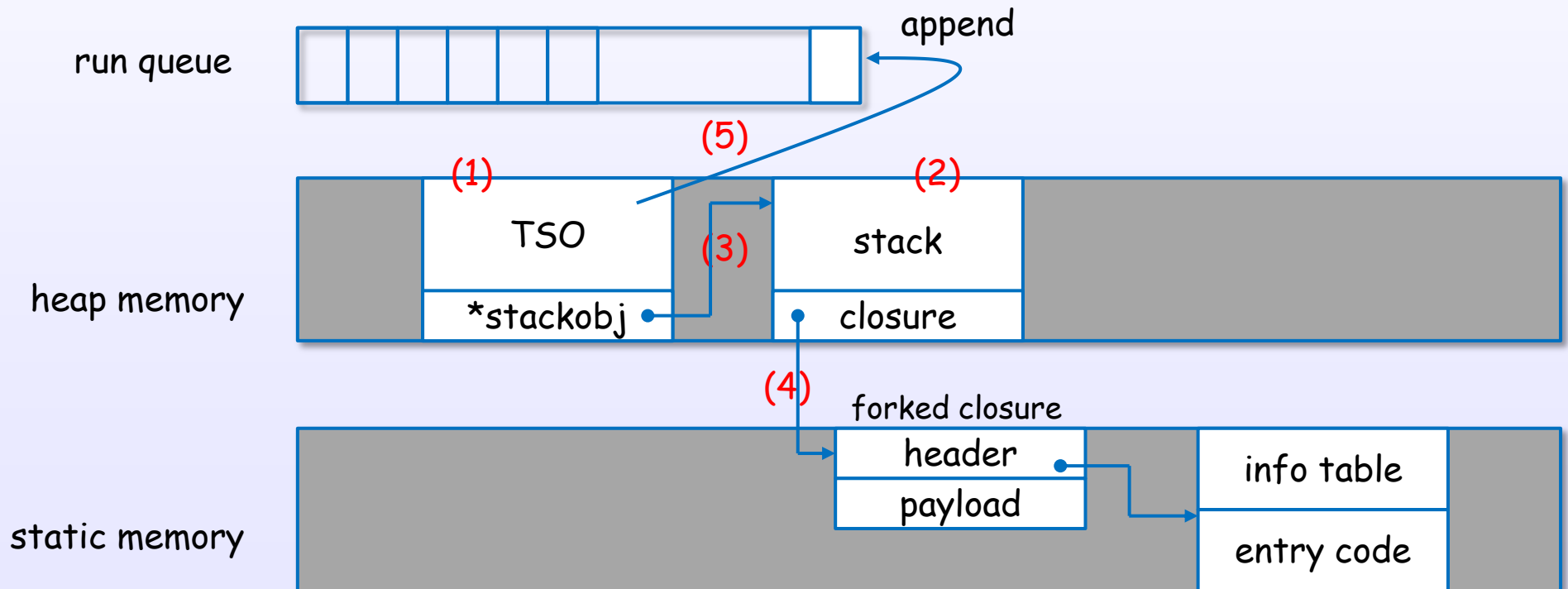
Create sub thread by forkIO

Haskell Threads

```
forkIO
  stg_forkzh
    ccall createIOThread ... (1), (2), (3), (4)
    ccall scheduleThread ... (5)
```

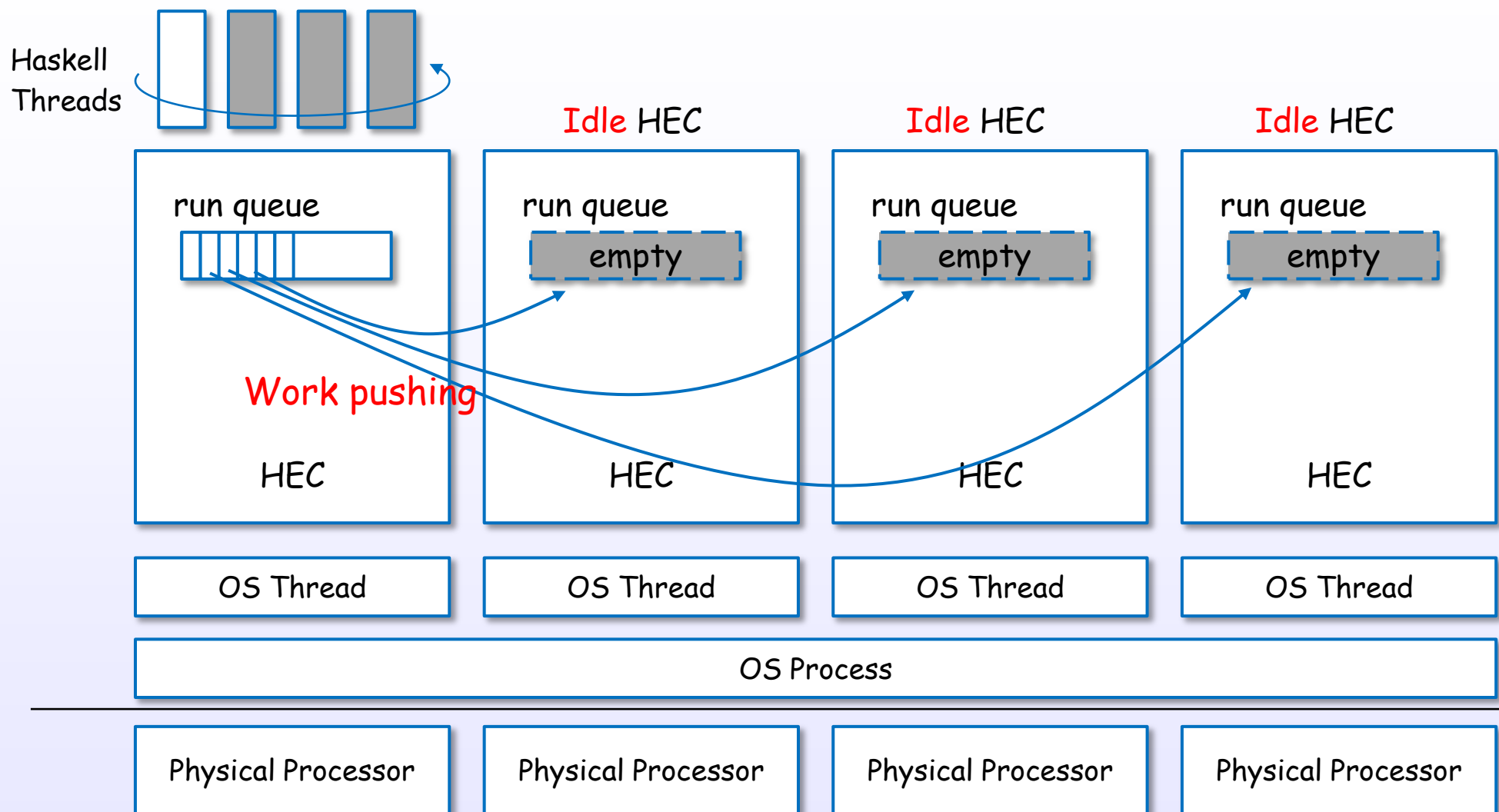
STG land
(Haskell land)

C land



Thread migration

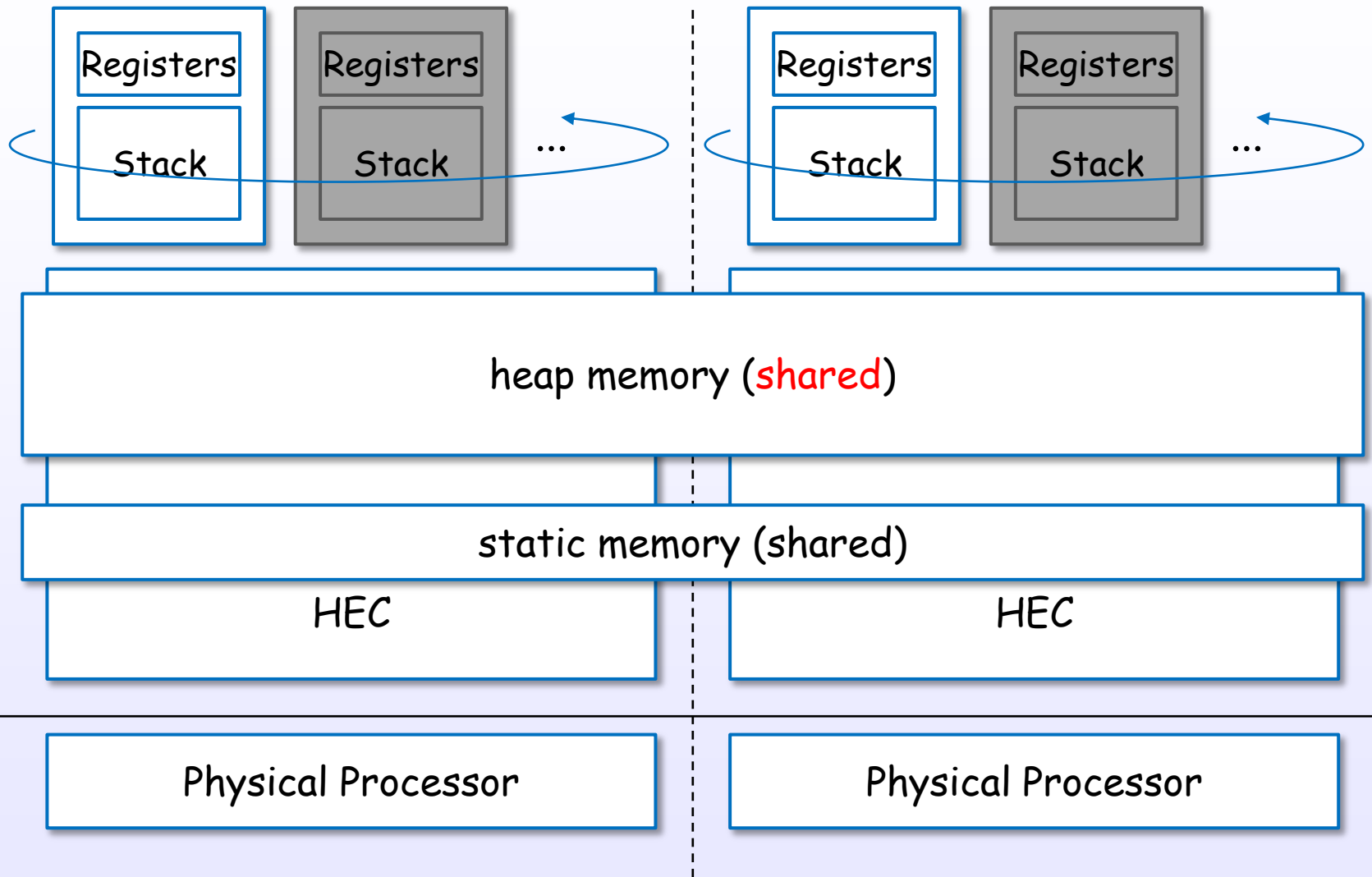
Threads migration



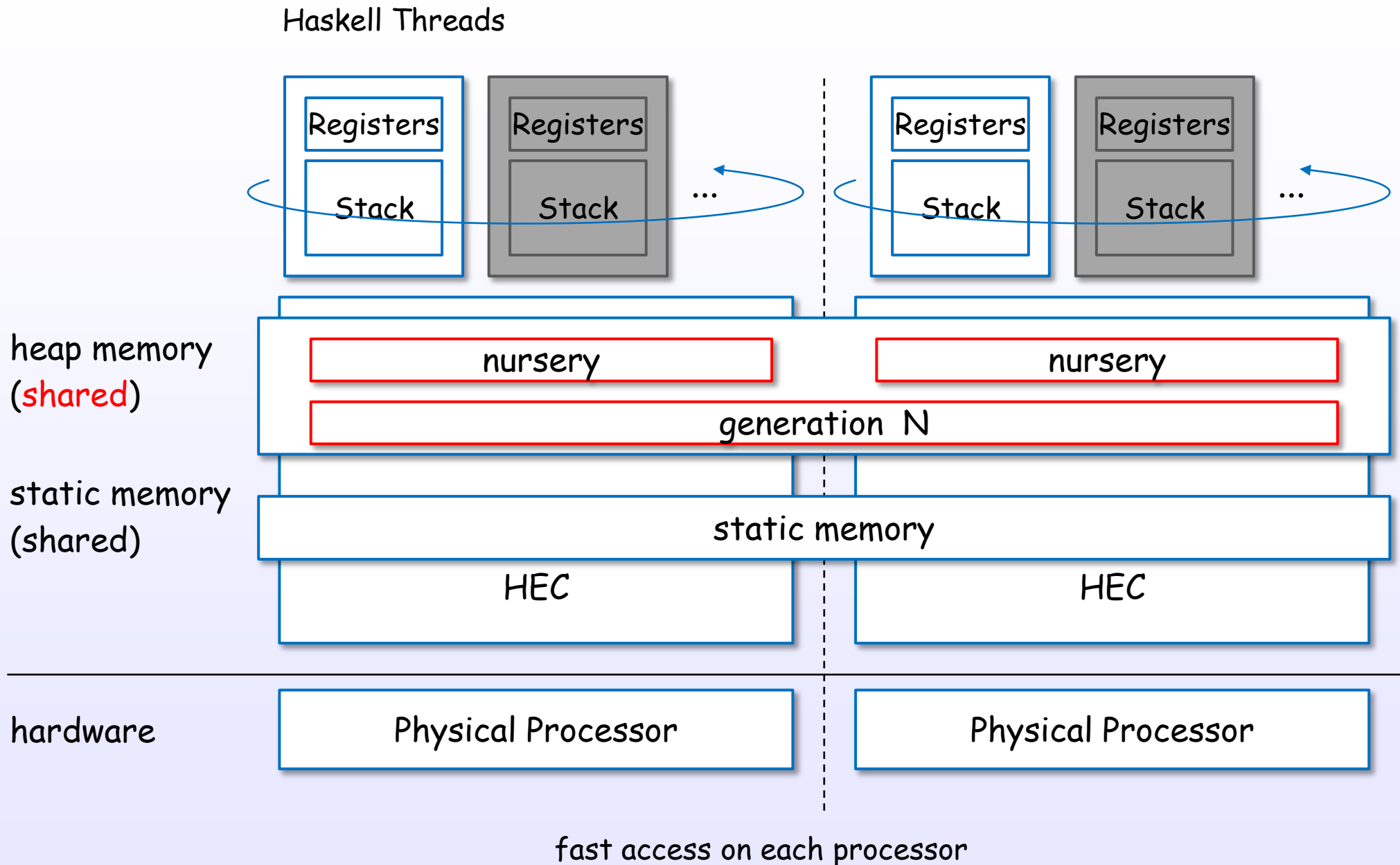
Heap and Threads

Threads and shared heap

Haskell Threads

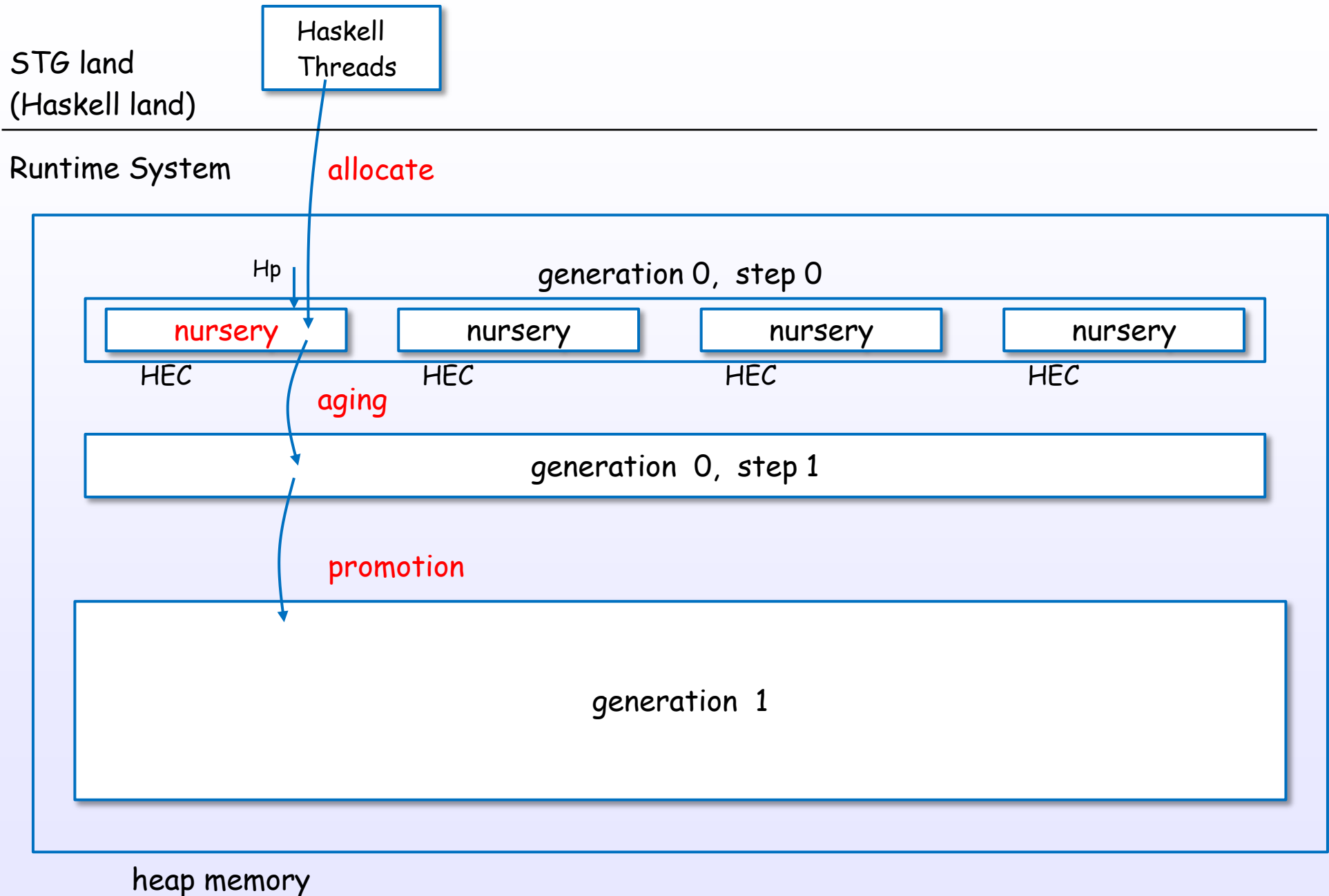


Local allocation area (nursery)



Threads and GC

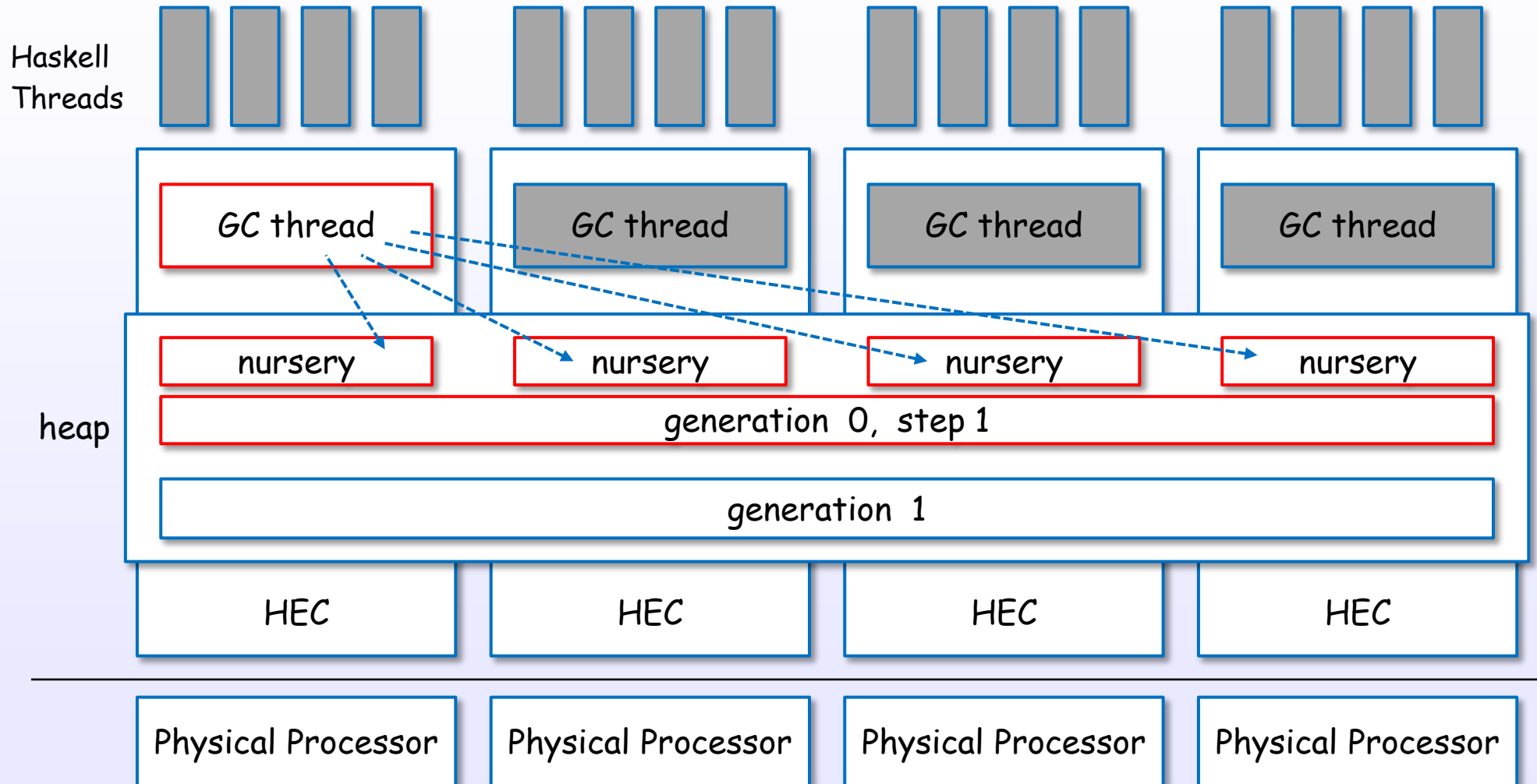
GC, nursery, generation, aging, promotion



Threads and minor GC

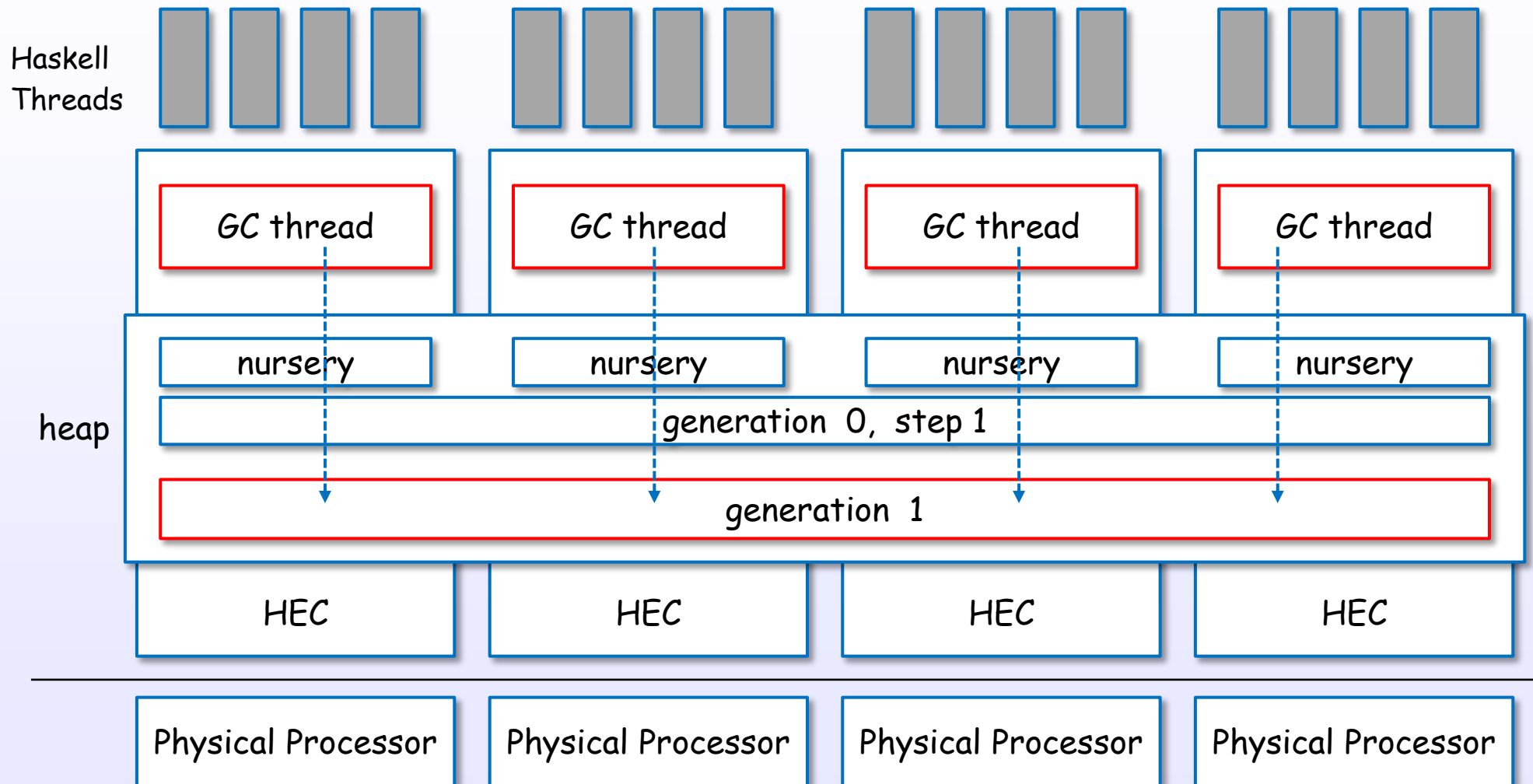
sequential GC for young generation (minor GC)

"stop-the-world" GC



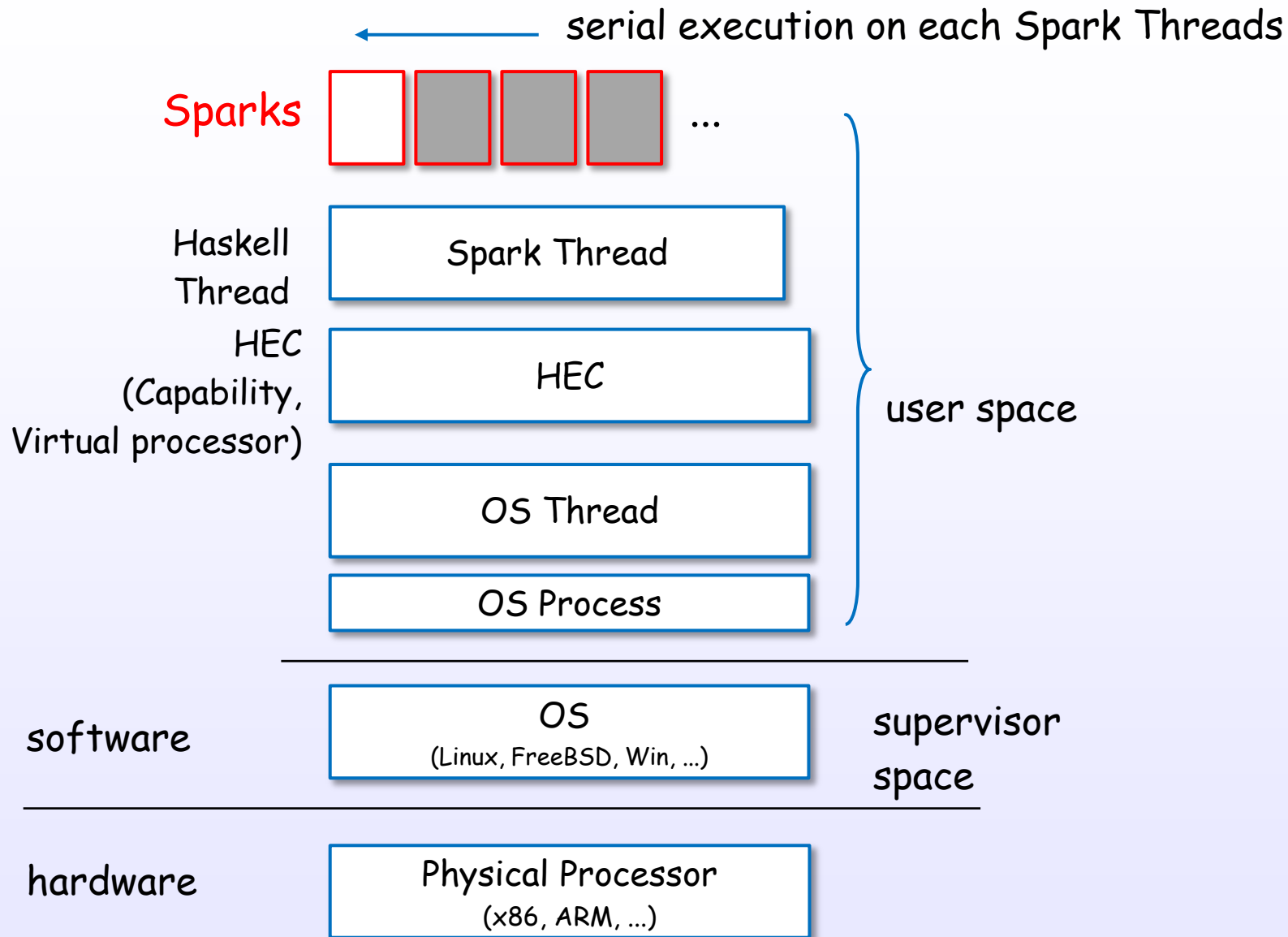
Threads and major GC

parallel GC for oldest generation (major GC)
"stop-the-world" GC



Spark

Spark layer

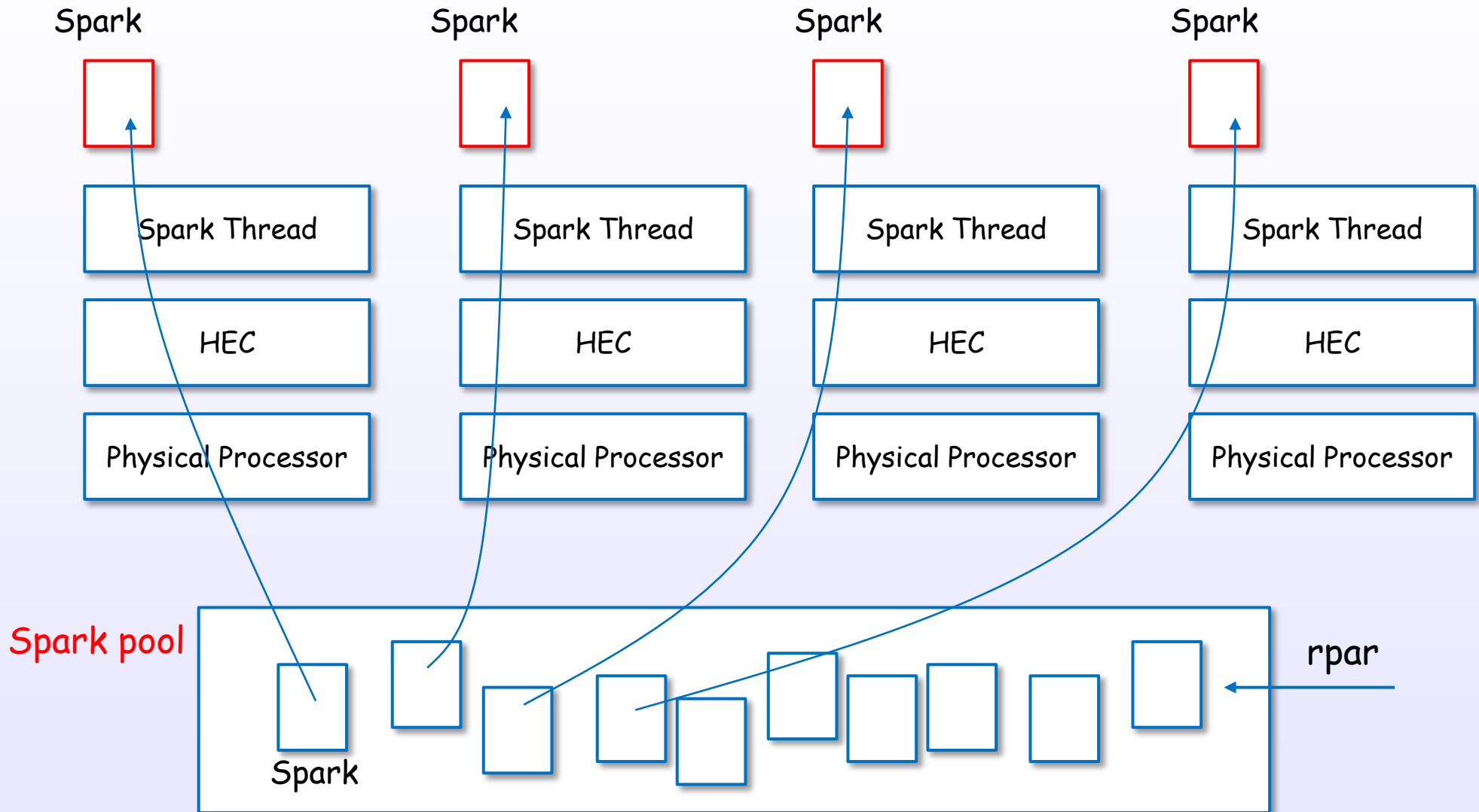


Spark Threads are generated on idle HECs.

References : [C17], [18], [S17], [S26], [S27], [S33], [S12]

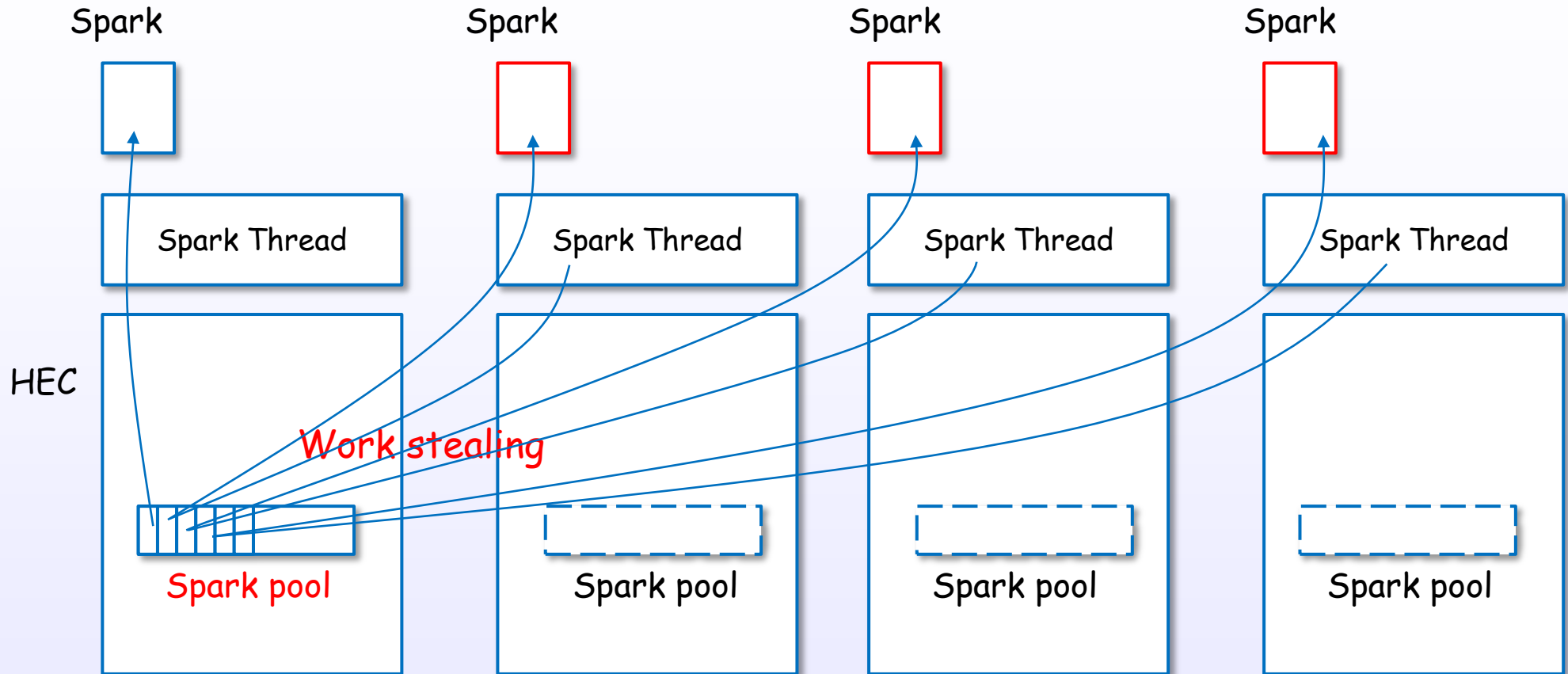
Sparks and Spark pool

logical view

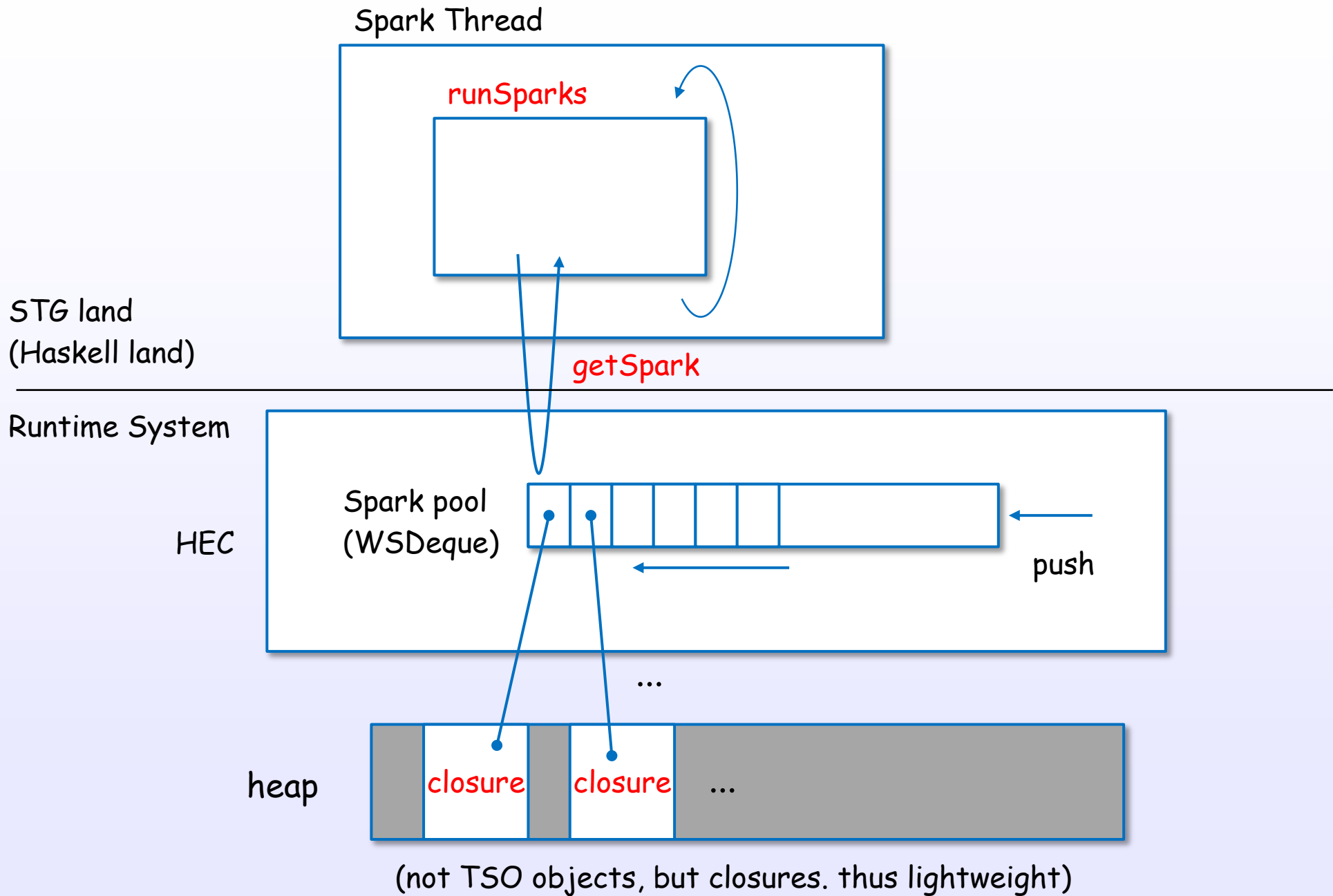


Spark pool and work stealing

physical view



Sparks and closures



MVar

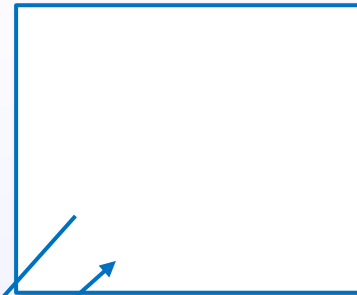
MVar

Haskell Thread #0

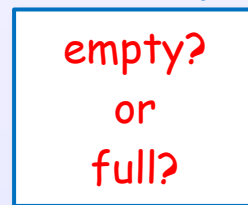


putMVar

Haskell Thread #1



takeMVar



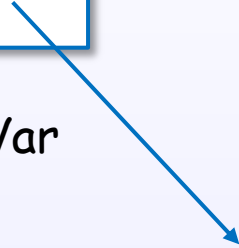
MVar

MVar

Haskell Thread



putMVar



BLOCKED
if full

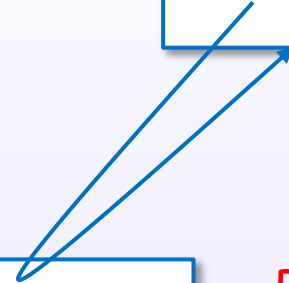


MVar

Haskell Thread



takeMVar



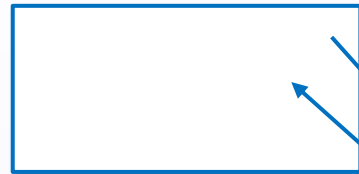
BLOCKED
if empty



MVar

MVar example

Haskell Thread #0

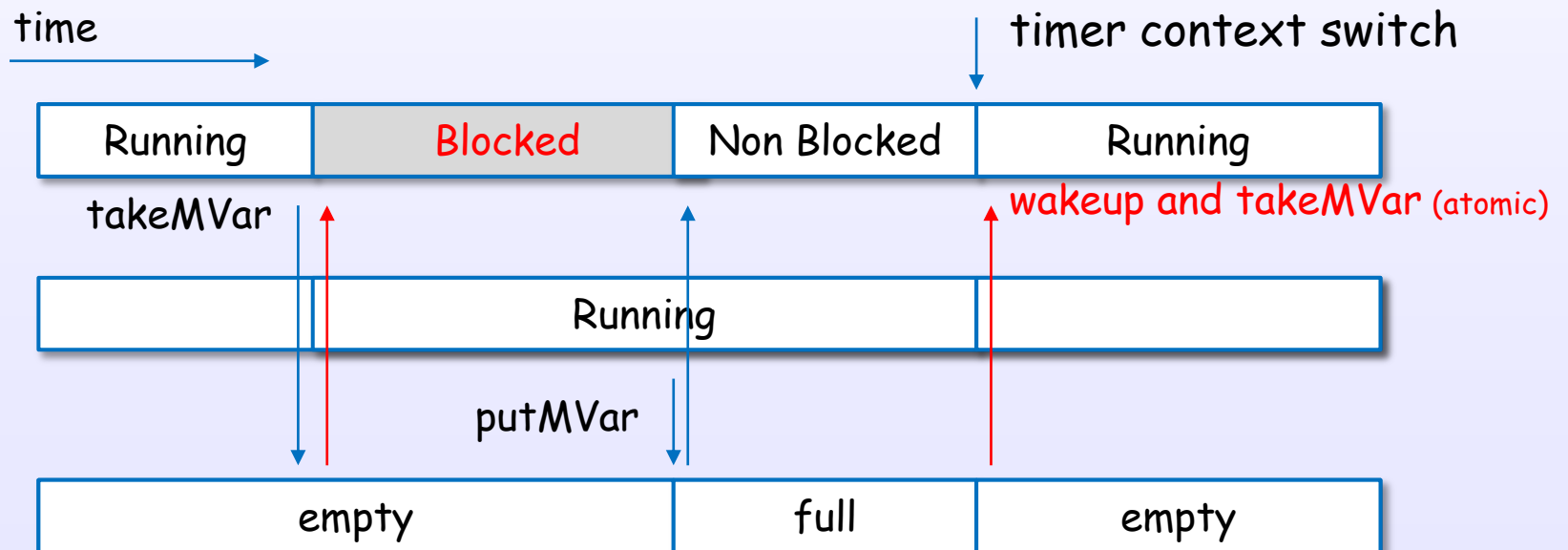


takeMVar

Haskell Thread #1



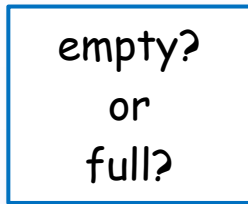
putMVar



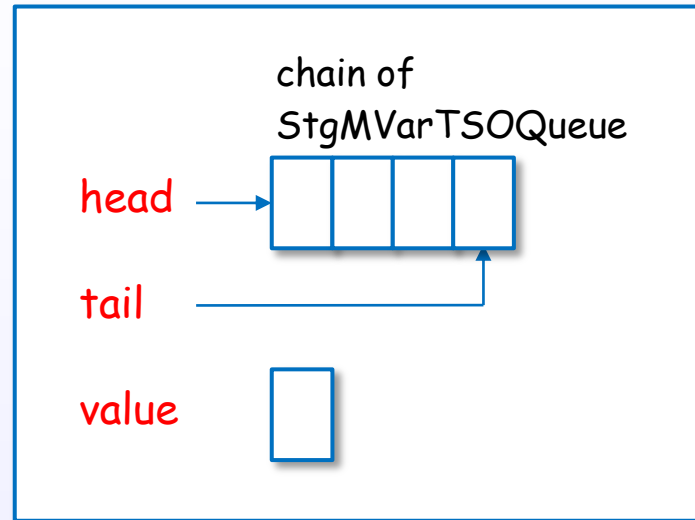
MVar view

User view

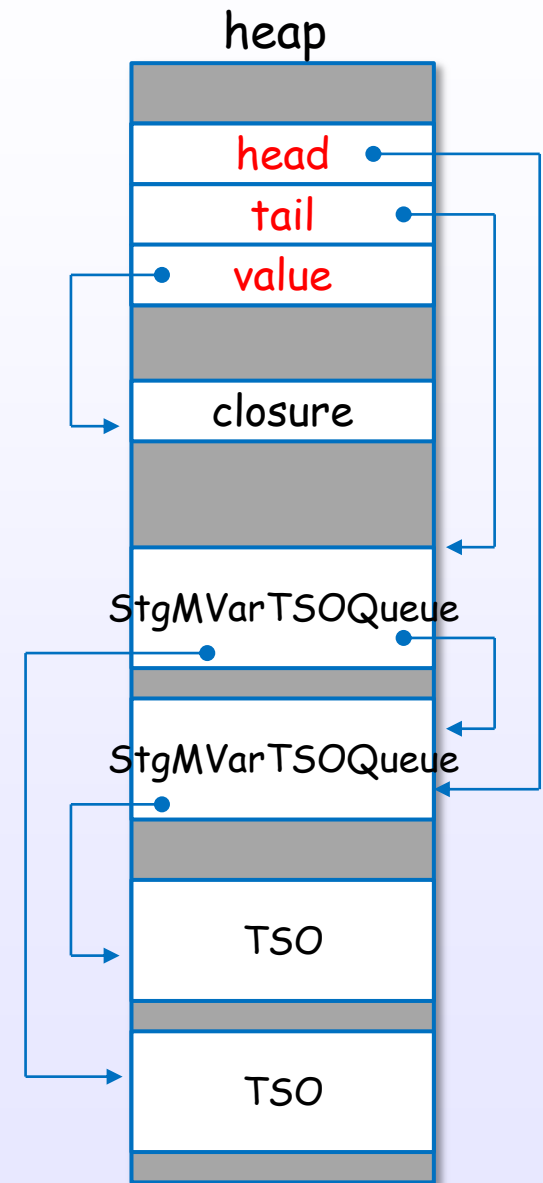
MVar



logical MVar object



physical MVar object



newEmptyMVar

Haskell Threads

```
newEmptyMVar  
newMVar#
```

(1) **call** Runtime primitive

Runtime System

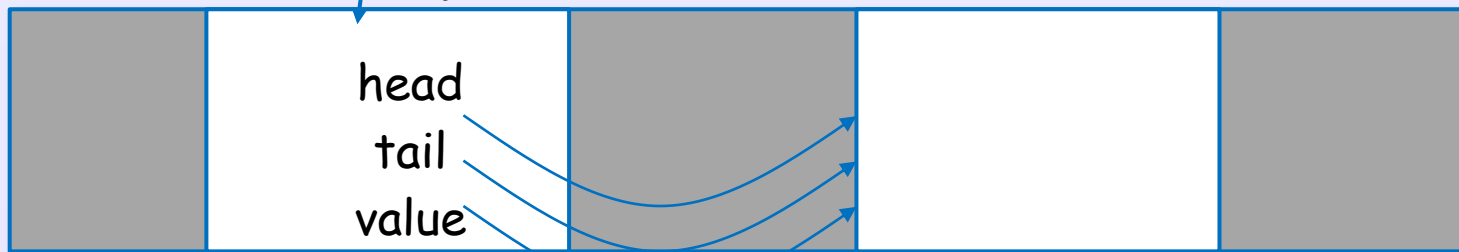
```
stg_newMVarzh  
  ALLOC_PRIM_  
  SET_HDR  
  StgMVar_head  
  StgMVar_tail  
  StgMVar_value
```

(2) **create MVar** object

MVar object

stg_END_TSO_QUEUE_closure

heap



(3) **link** each field

References : [15], [17], [18], [S31], [S12]

takeMVar (empty case)

Haskell Threads

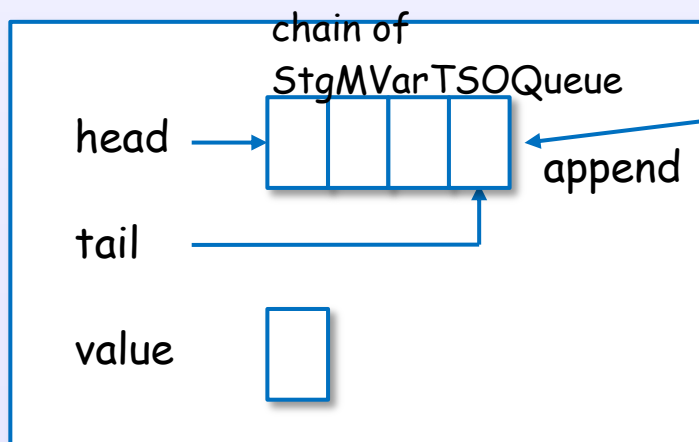
takeMVar
takeMVar#

Runtime System

stg_takeMVarzh

- (1) **create** StgMVarTSOQueue
- (2) **append**
- (3) **StgReturn**

return to scheduler



MVar object

StgMVarTSOQueue

stg_END_TSO_QUEUE

CurrentTSO

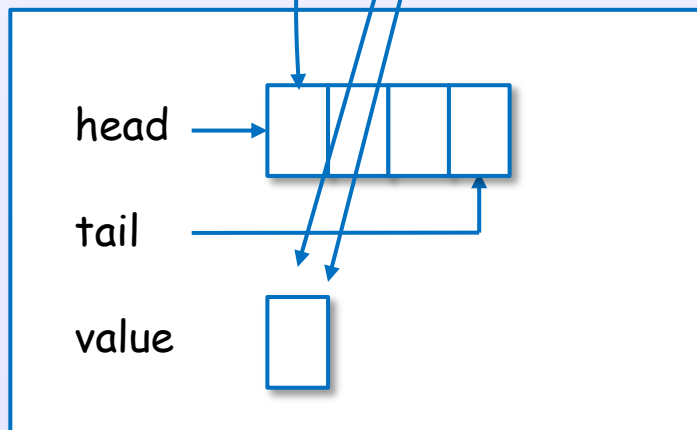
takeMVar (full case)

Haskell Threads

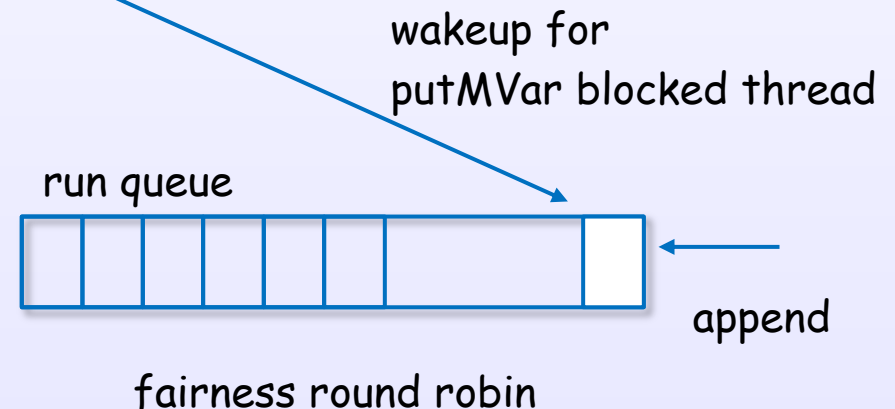
```
takeMVar  
takeMVar#
```

Runtime System

```
stg_takeMVarzh  
(1) get value  
(2) set empty  
(3) remove head  
(4) tryWakeupThread
```



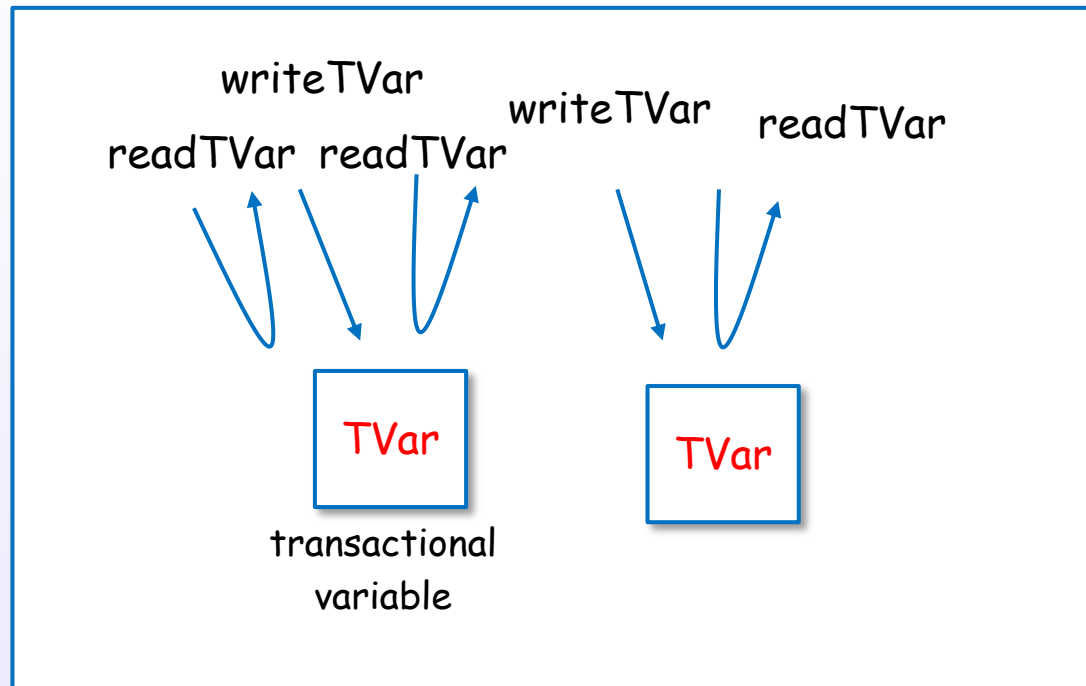
MVar object



Software transactional memory

Create atomic block by atomically

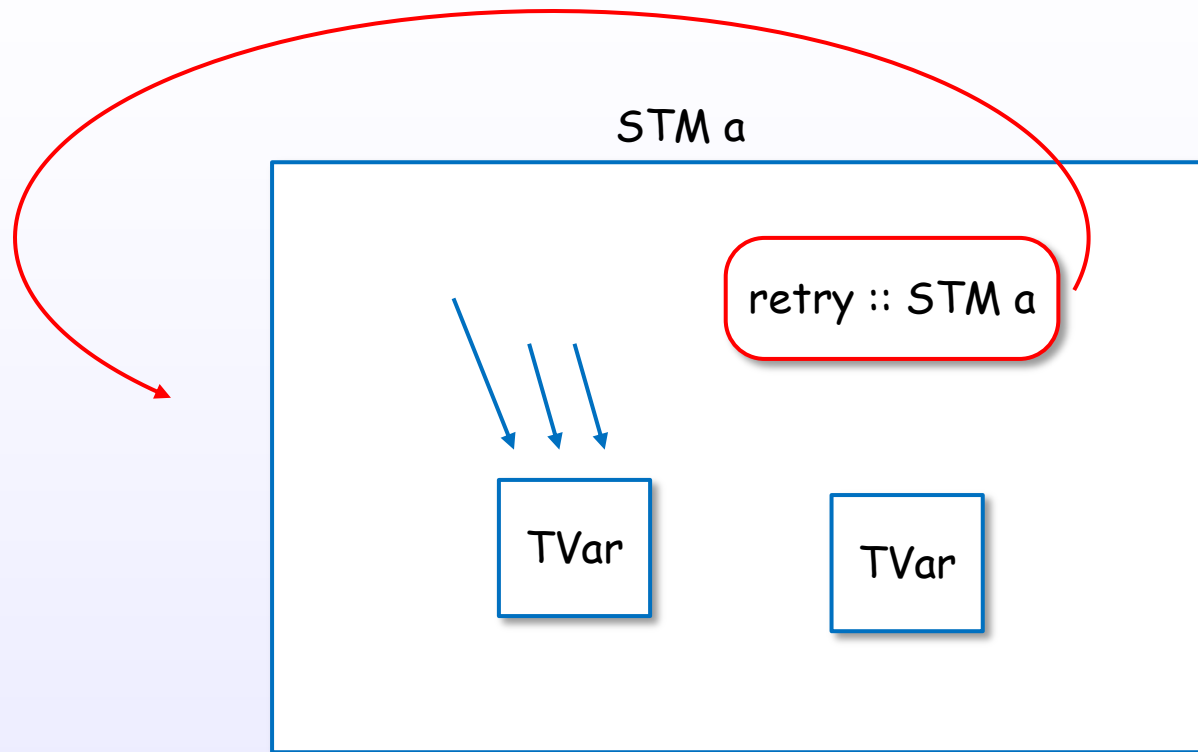
atomically :: STM a -> IO a



Create "atomic block"

All or Nothing

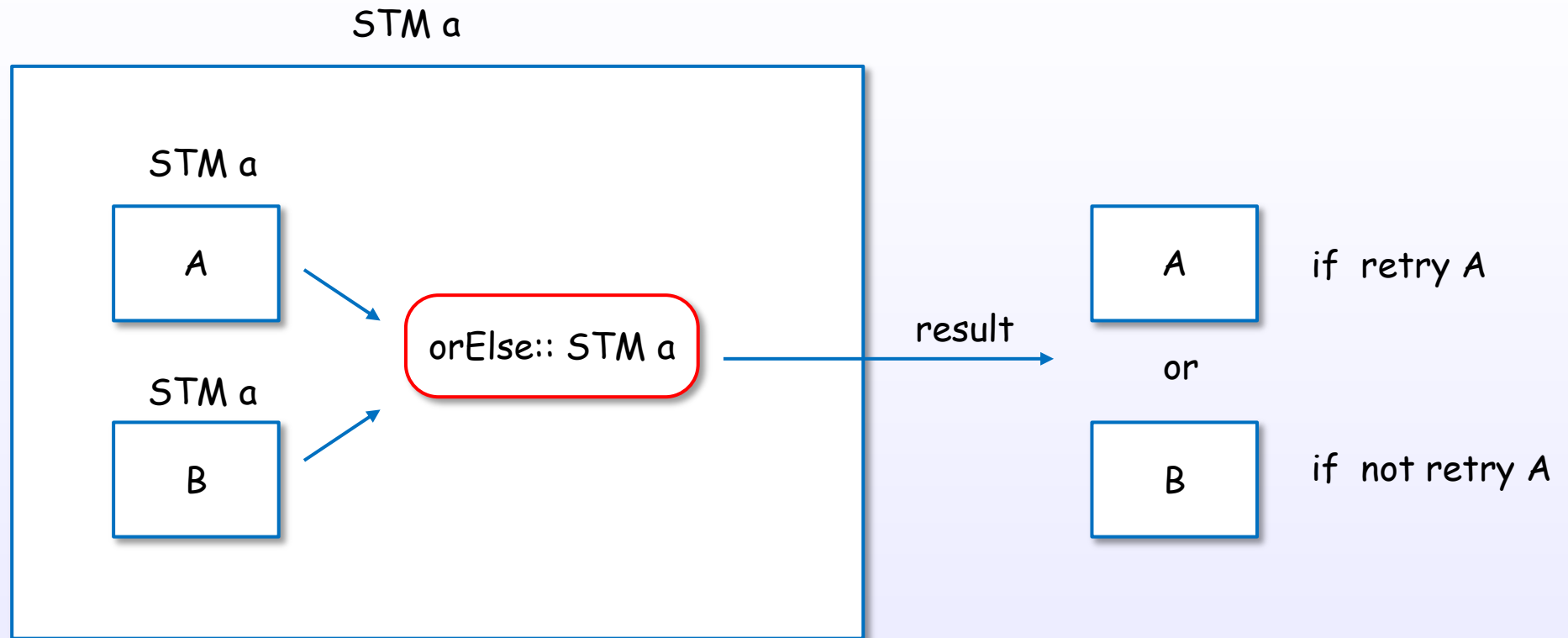
Rollback and blocking control by retry



Safety retry

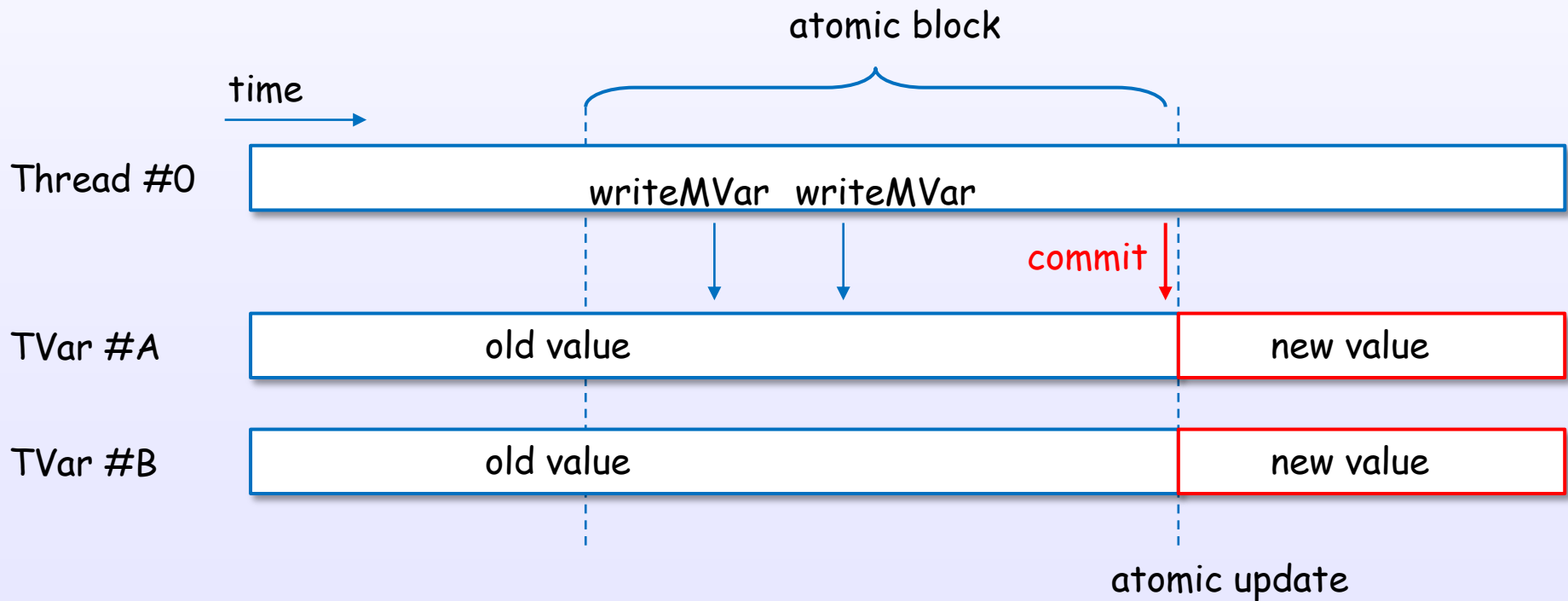
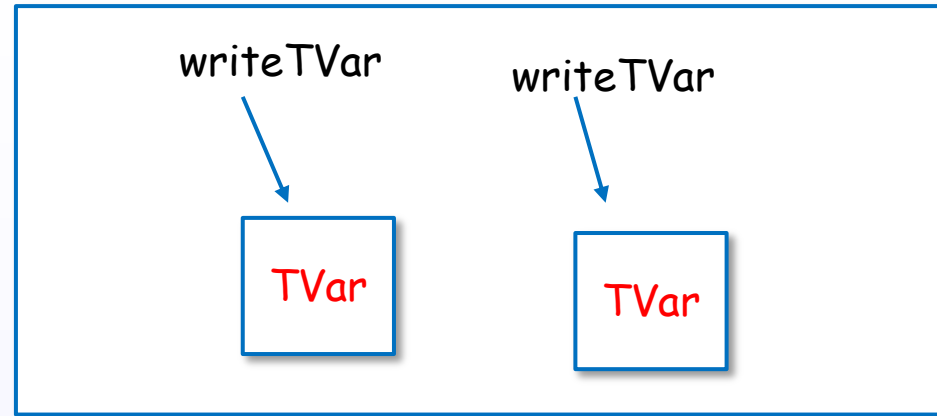
All or Nothing

Compose OR case by orElse

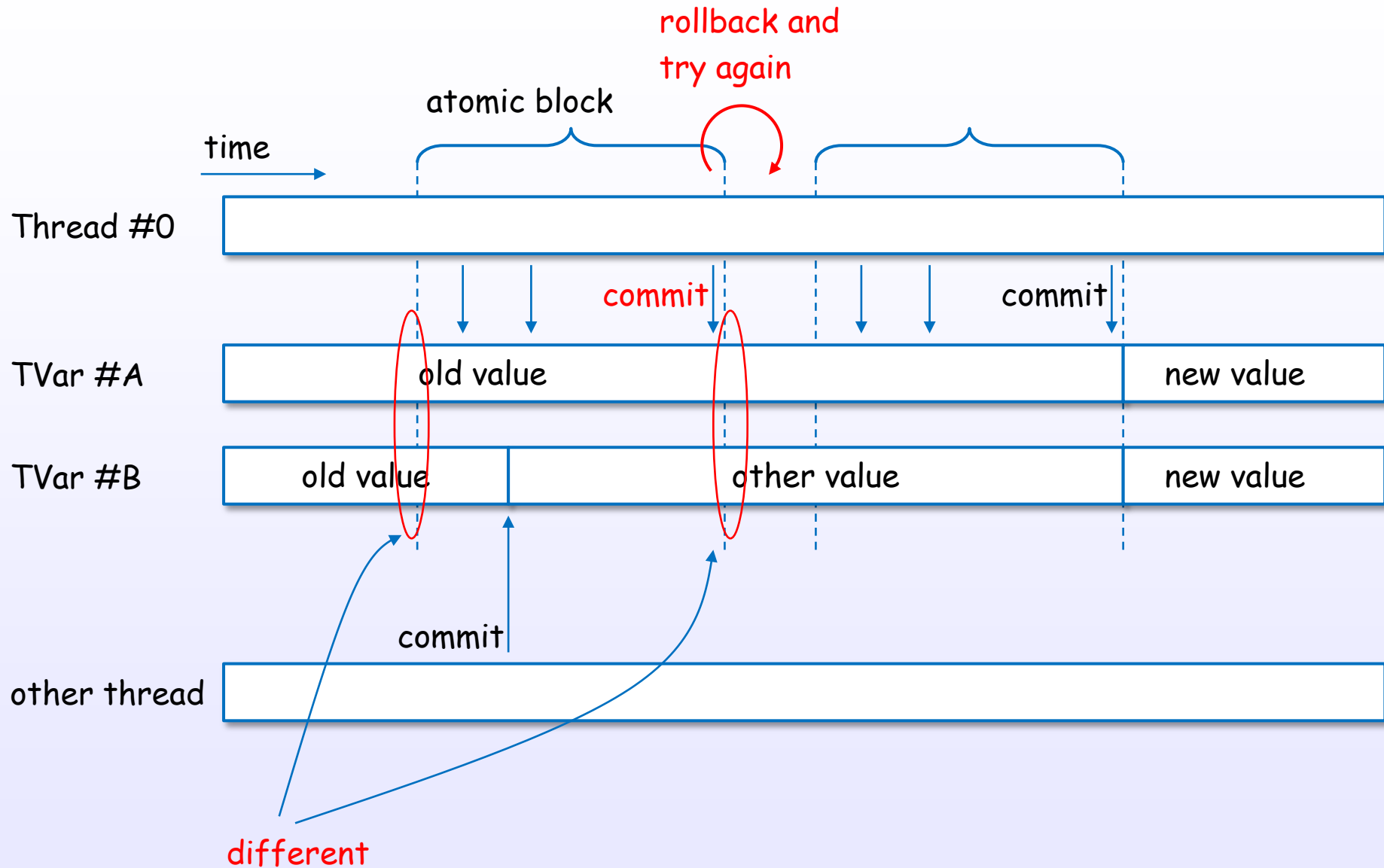


Compose "OR" case
A or B or Nothing

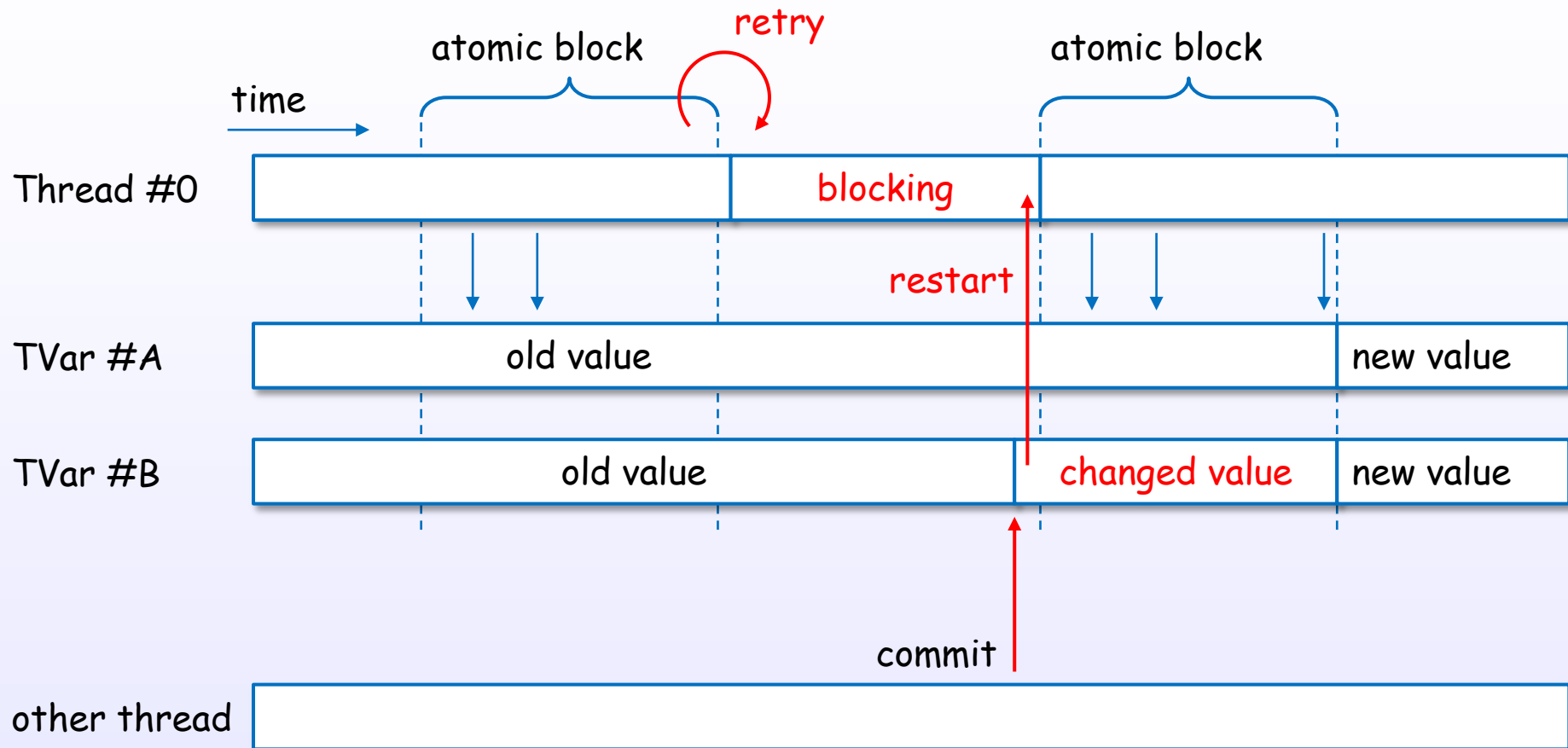
STM, TVar example (normal case)



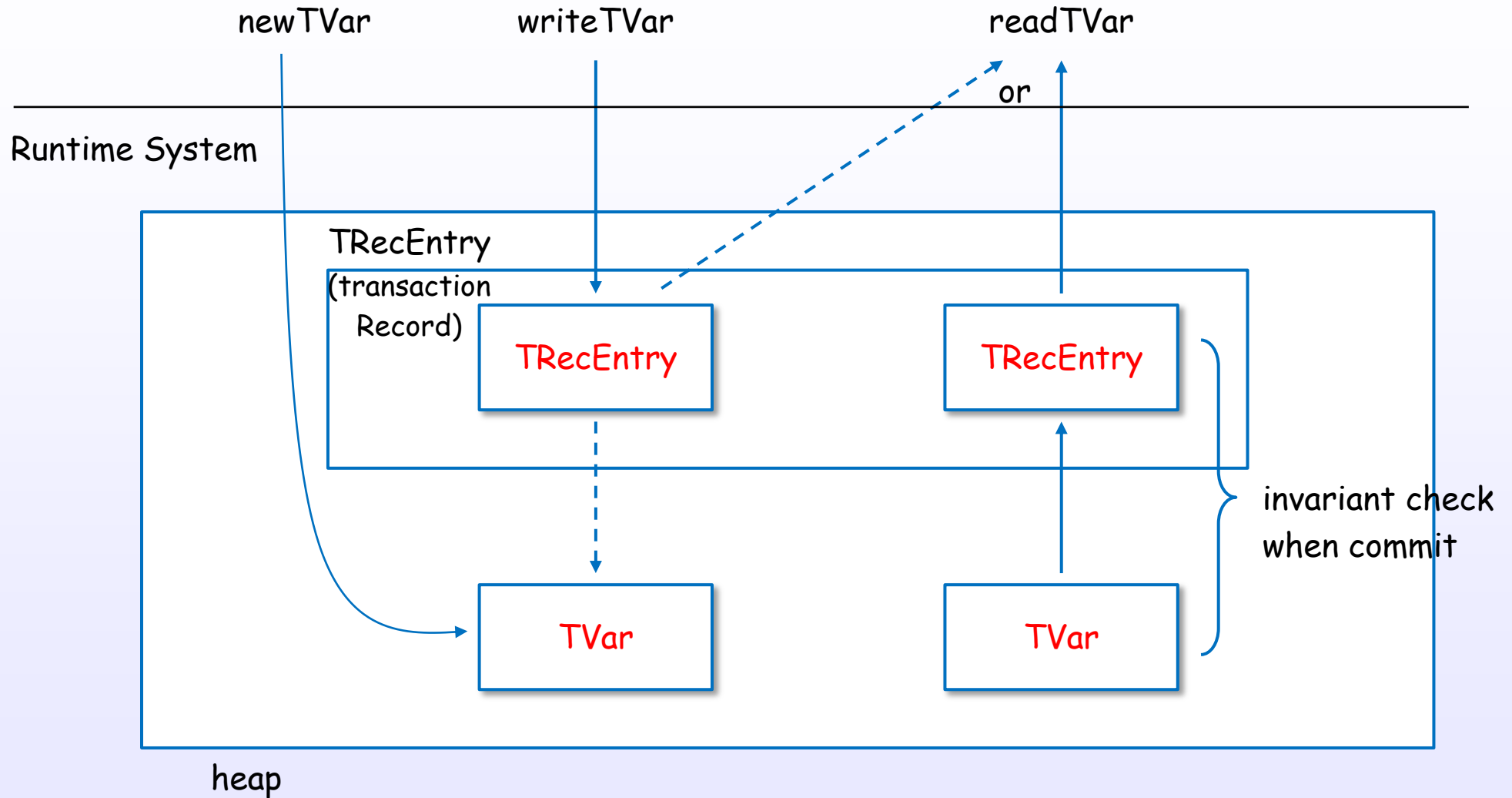
STM, TVar example (conflict case)



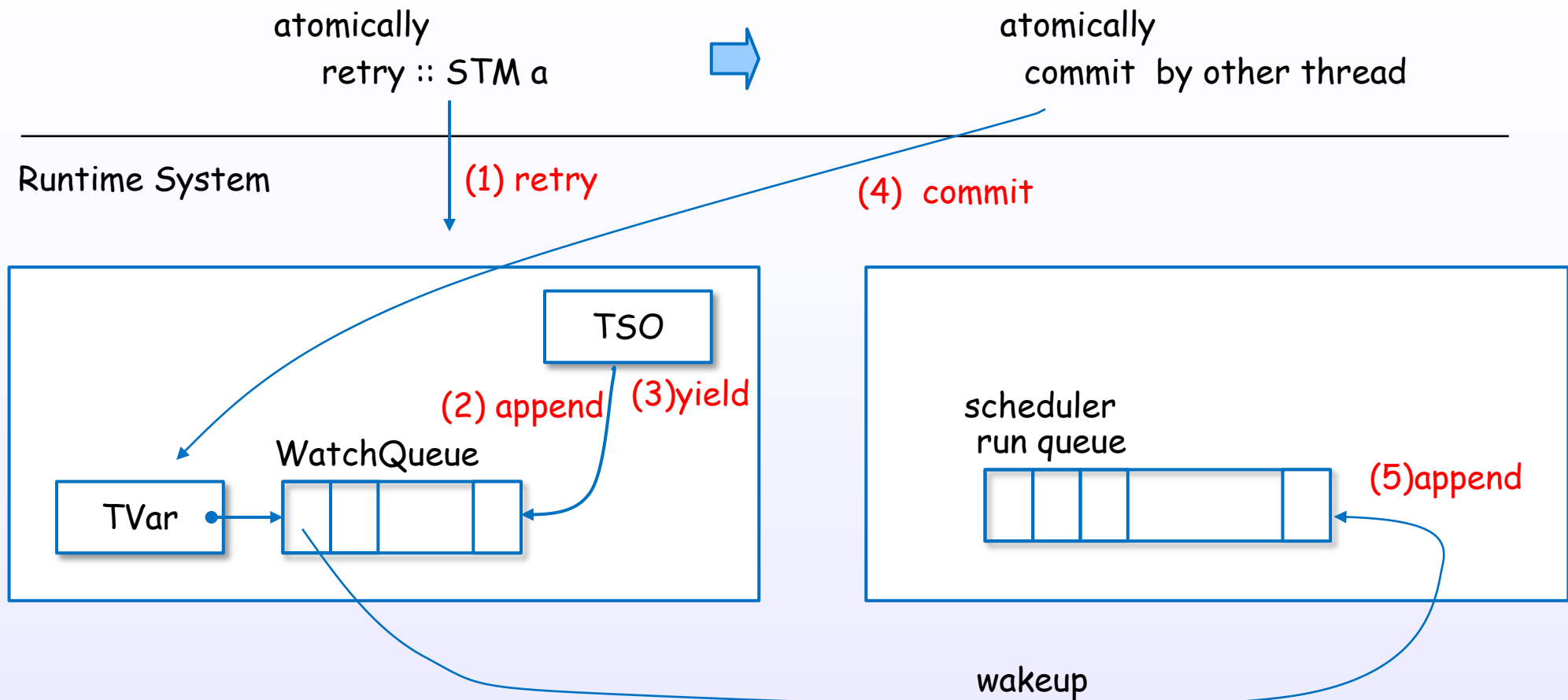
retry example



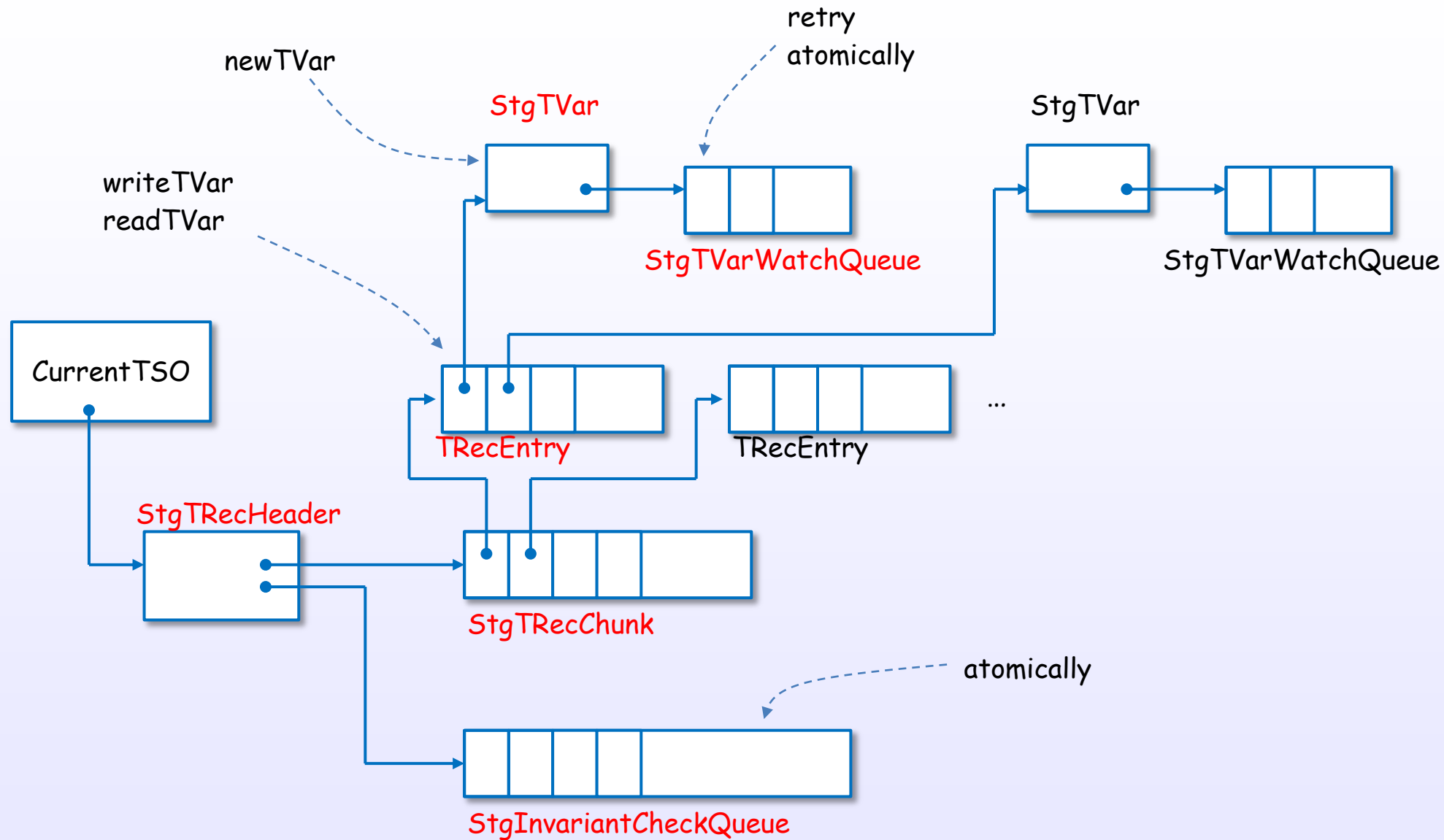
newTVar, writeTVar, readTVar



retry blocking and wake up



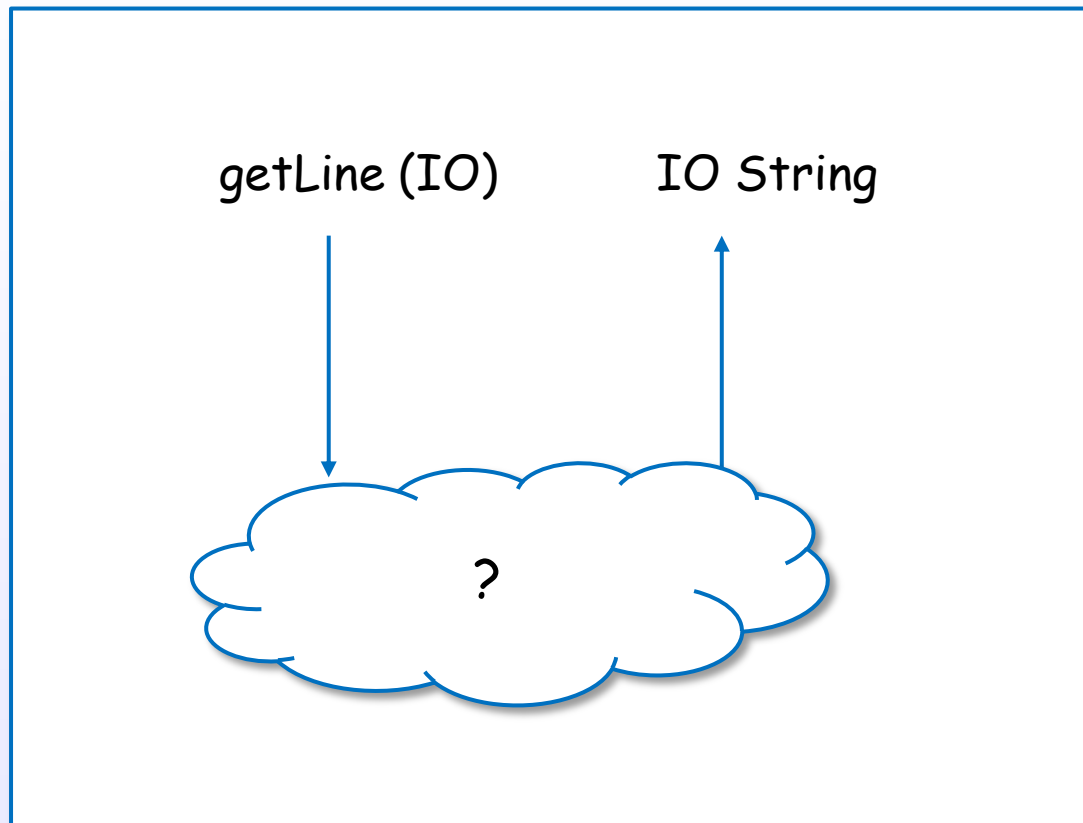
STM, TVar data structure



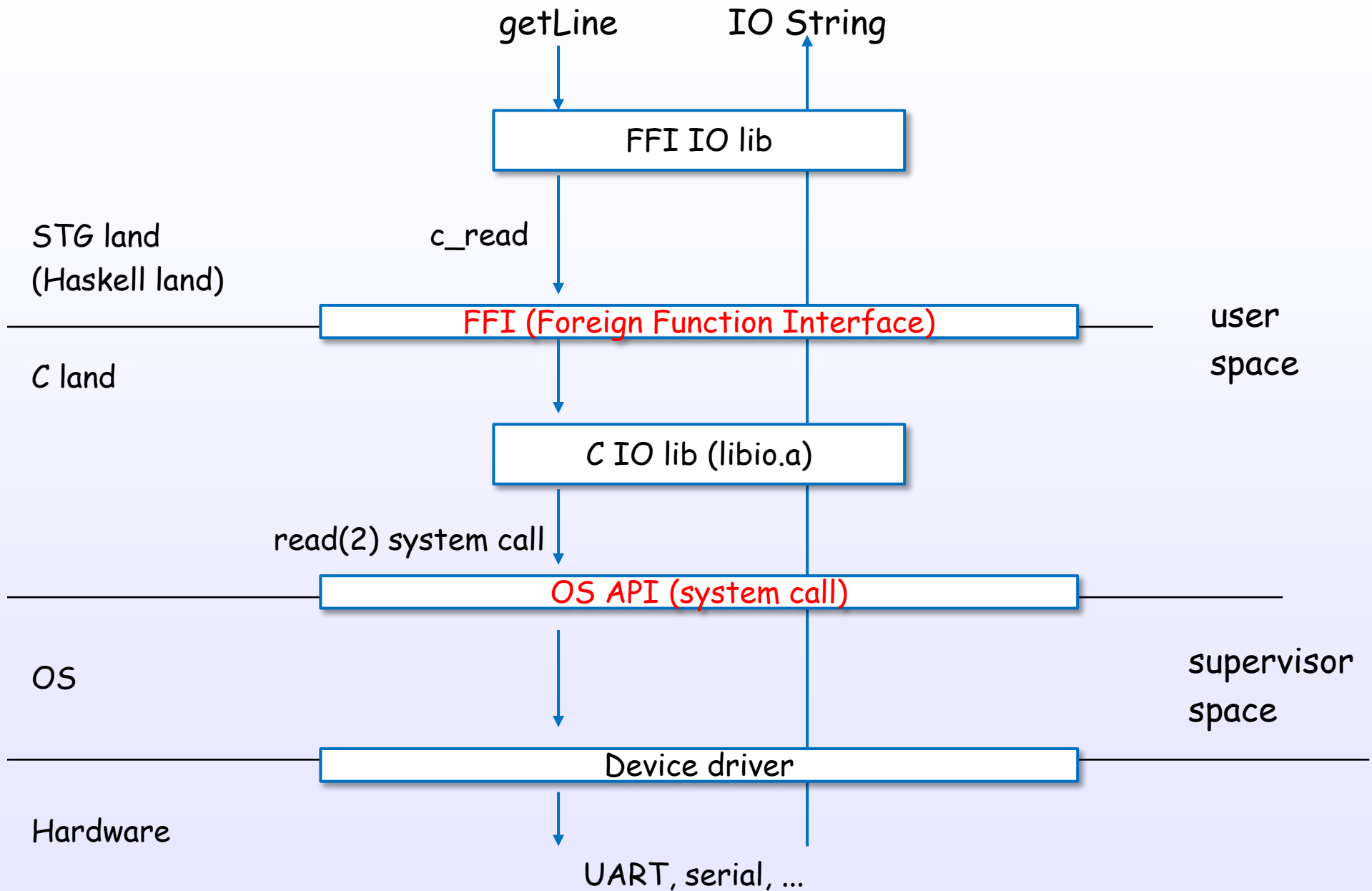
IO and FFI

IO

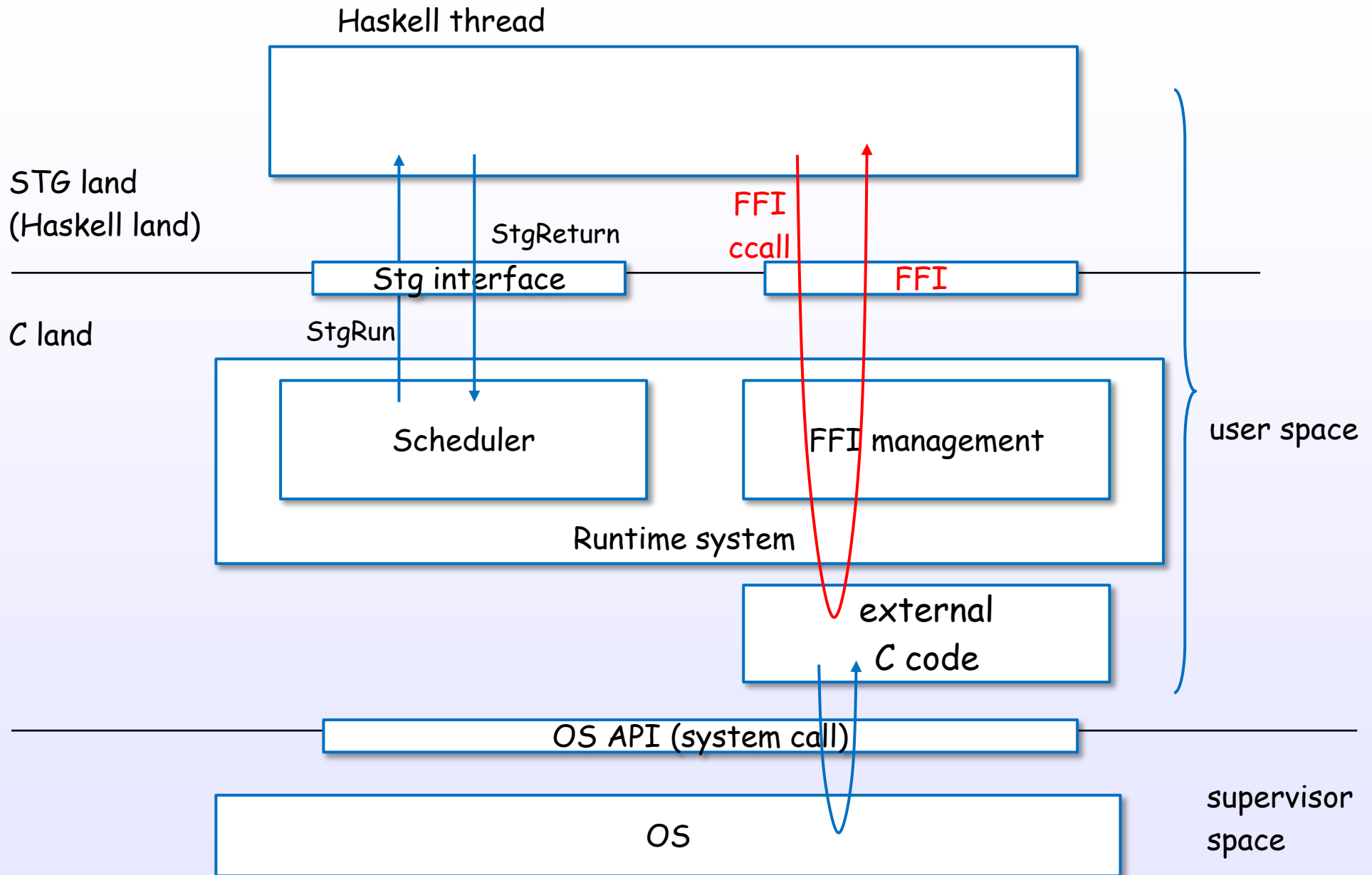
Haskell Thread



IO example: getLine

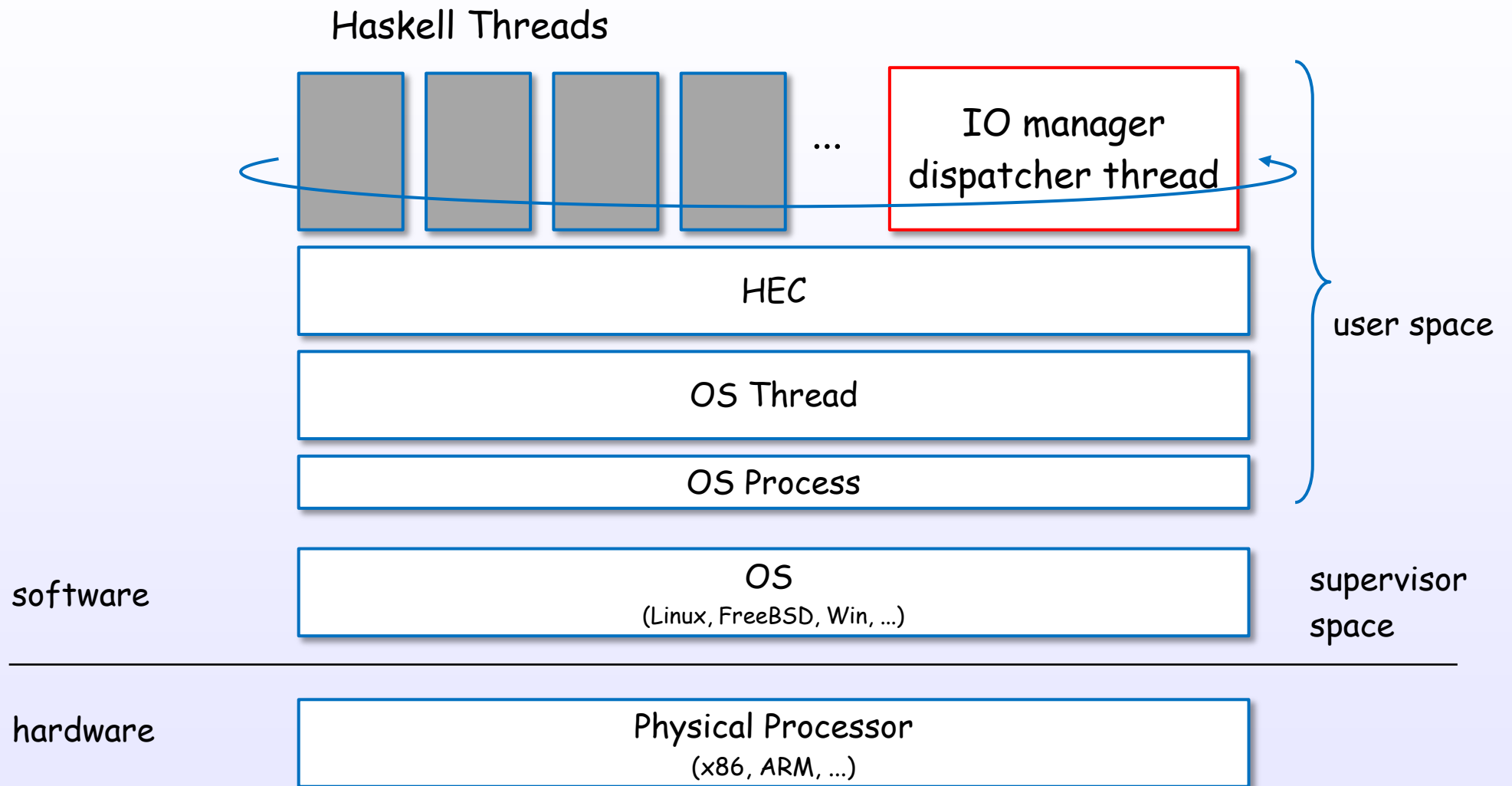


FFI (Foreign Function Interface)



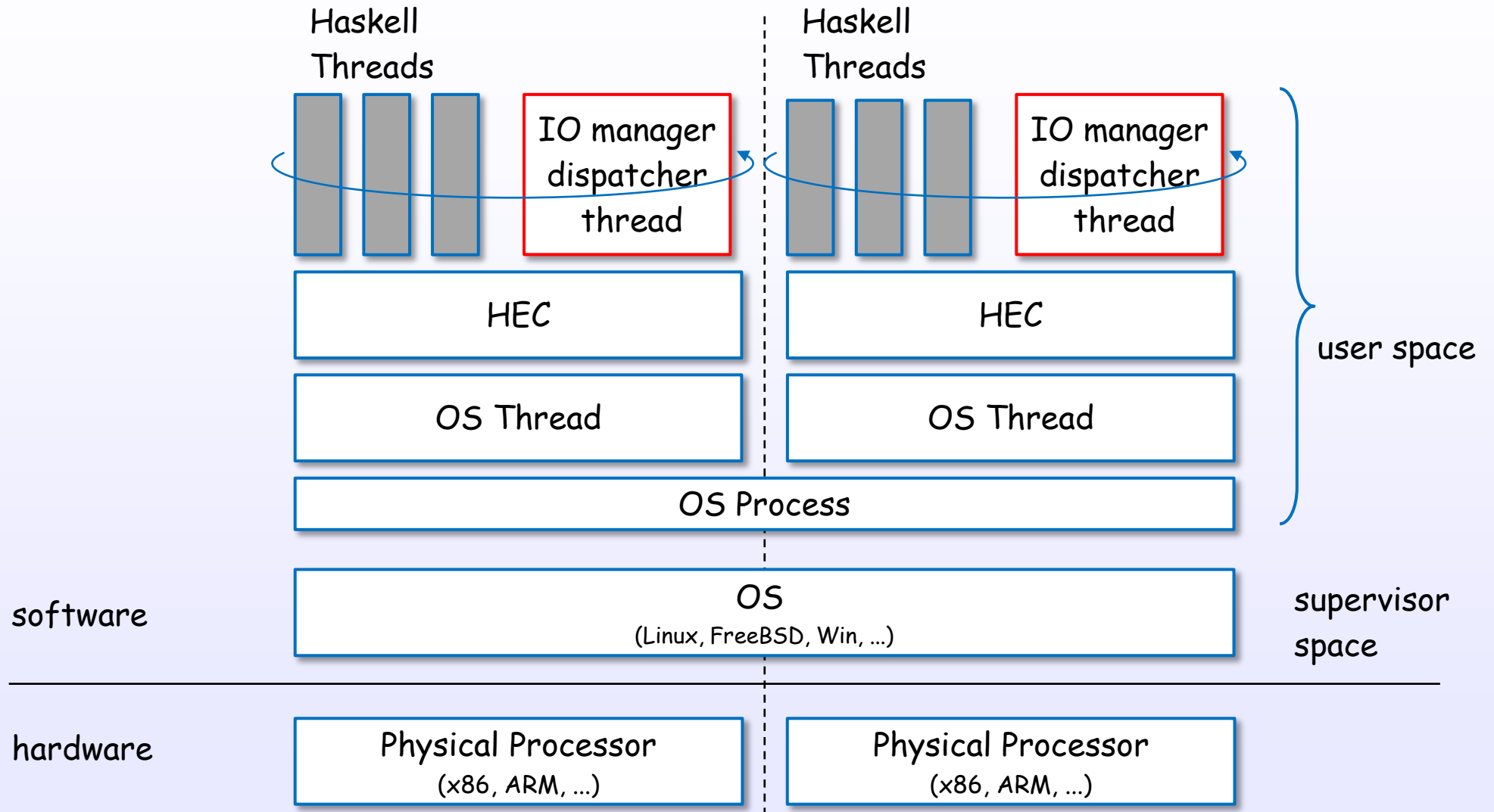
IO manager

IO manager (single core)



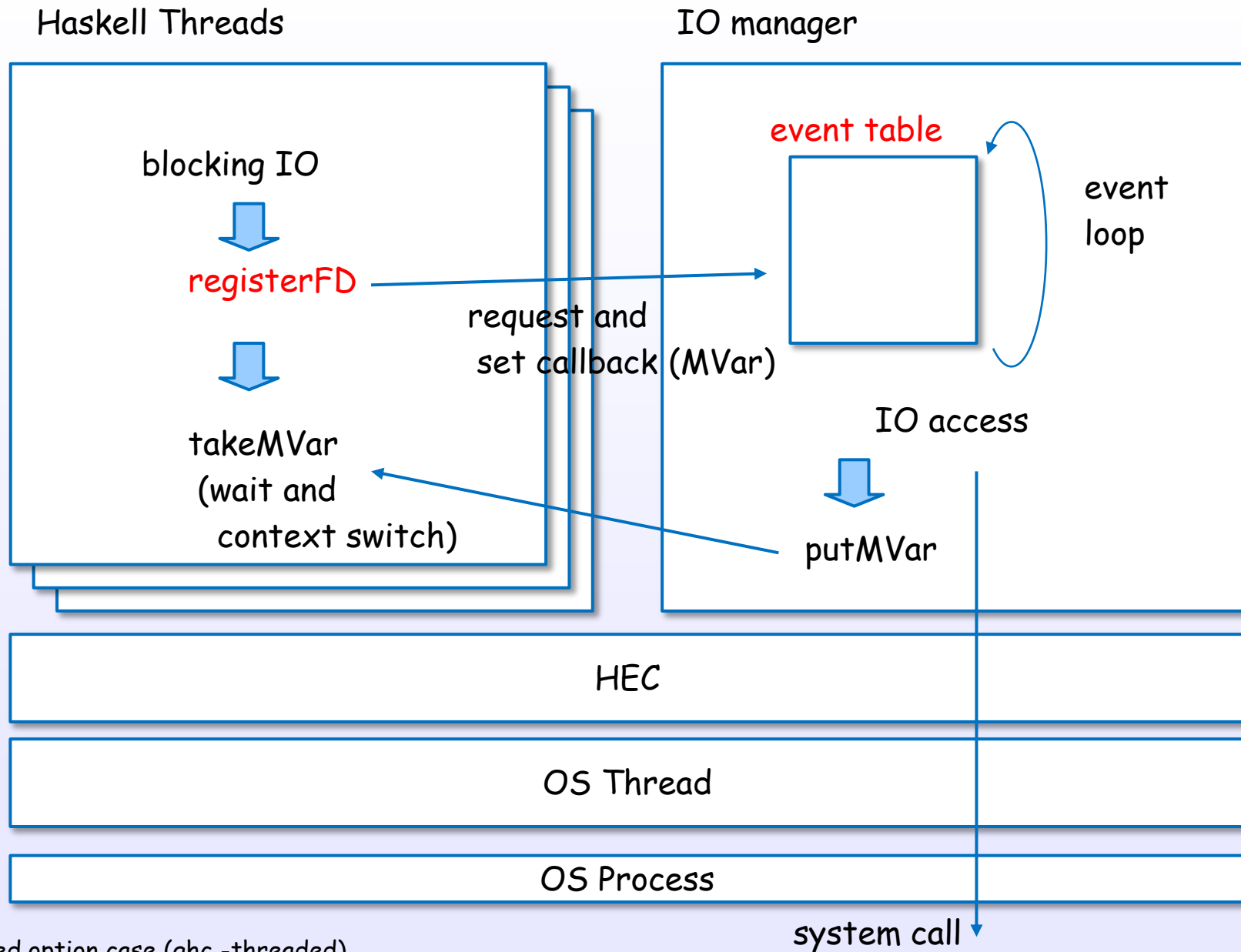
*Threaded option case (ghc -threaded)

IO manager (multi core)



*Threaded option case (ghc -threaded)

IO manager

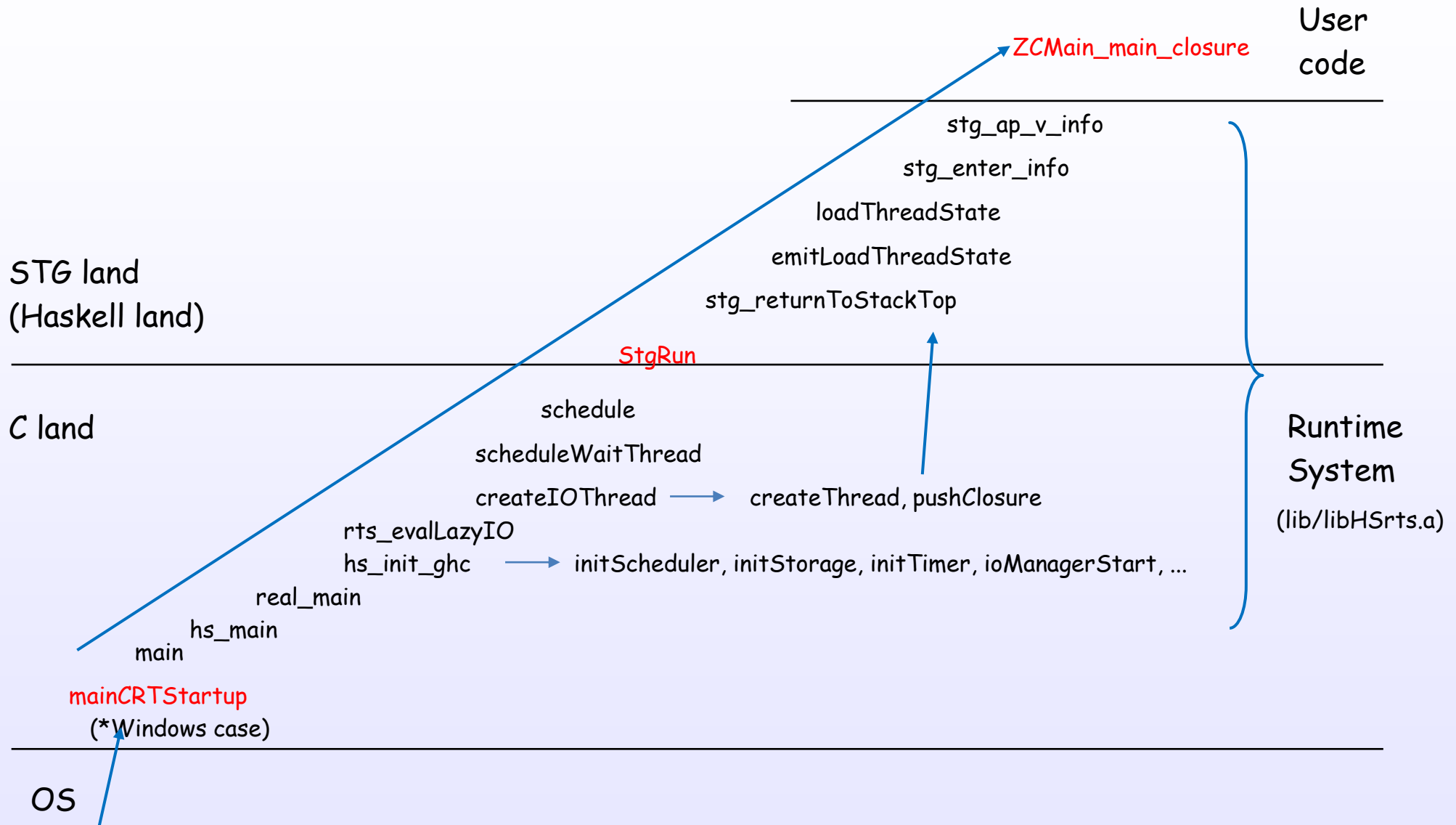


*Threaded option case (ghc -threaded)

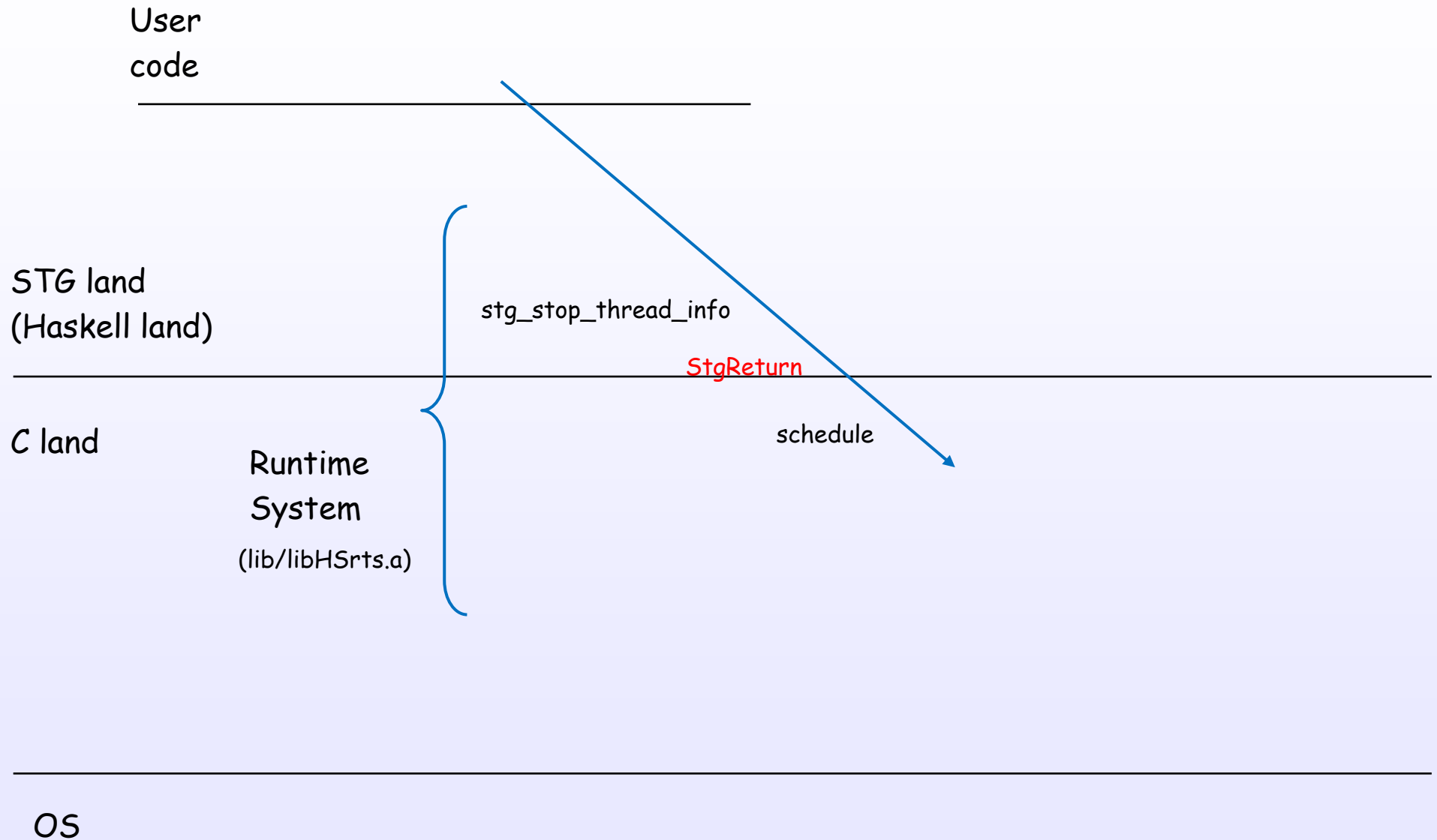
References : [6], [4], [7], [S29], [S30], [S32], [S37], [S35], [S3]

Bootstrap

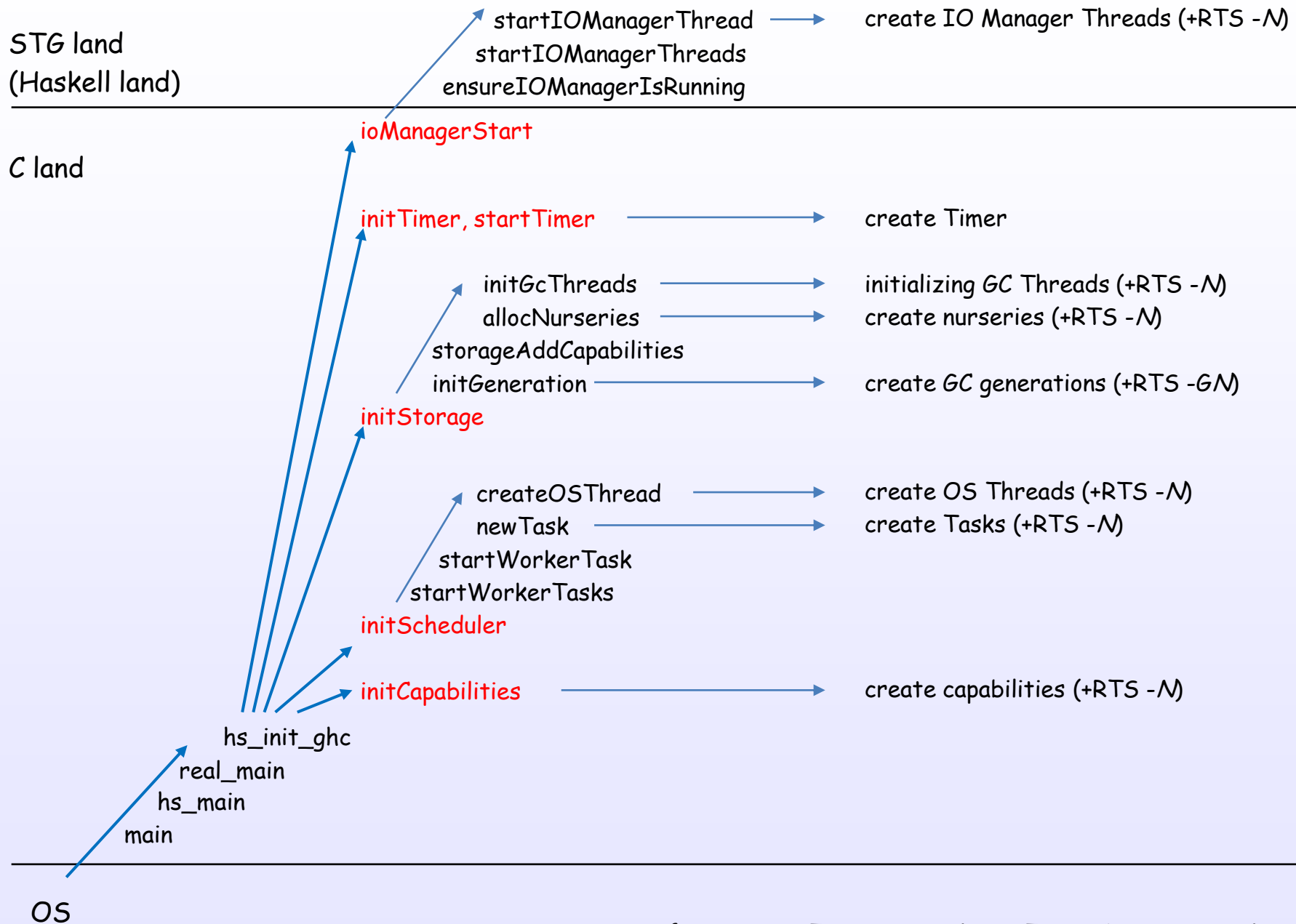
Bootstrap sequence



Exit sequence



Initializing



References

References

- [1] Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine Version 2.5
<http://research.microsoft.com/en-us/um/people/simonpj/Papers/spineless-tagless-gmachine.ps.gz>
- [2] Making a Fast Curry Push/Enter vs Eval/Apply for Higher-order Languages
<http://research.microsoft.com/en-us/um/people/simonpj/papers/eval-apply/>
- [3] Faster Laziness Using Dynamic Pointer Tagging
<http://research.microsoft.com/en-us/um/people/simonpj/papers/ptr-tag/ptr-tagging.pdf>
- [4] Runtime Support for Multicore Haskell
<http://research.microsoft.com/en-us/um/people/simonpj/papers/parallel/multicore-ghc.pdf>
- [5] Extending the Haskell Foreign Function Interface with Concurrency
<http://community.haskell.org/~simonmar/papers/conc-ffi.pdf>
- [6] Mio: A High-Performance Multicore IO Manager for GHC
<http://haskell.cs.yale.edu/wp-content/uploads/2013/08/hask035-voellmy.pdf>
- [7] The GHC Runtime System
web.mit.edu/~ezyang/Public/jfp-ghc-rts.pdf
- [8] The GHC Runtime System
<http://www.scs.stanford.edu/14sp-cs240h/slides/ghc-rts.pdf>
- [9] Evaluation on the Haskell Heap
<http://blog.ezyang.com/2011/04/evaluation-on-the-haskell-heap/>
- [10] IO evaluates the Haskell Heap
<http://blog.ezyang.com/2011/04/io-evaluates-the-haskell-heap/>

References

- [11] Understanding the Stack
<http://www.well-typed.com/blog/94/>
- [12] Understanding the RealWorld
<http://www.well-typed.com/blog/95/>
- [13] The GHC scheduler
<http://blog.ezyang.com/2013/01/the-ghc-scheduler/>
- [14] GHC's Garbage Collector
http://www.mm-net.org.uk/workshop190404/GHC's_Garbage_Collector.ppt
- [15] Concurrent Haskell
<http://www.haskell.org/ghc/docs/papers/concurrent-haskell.ps.gz>
- [16] Beautiful Concurrency
<https://www.fpcomplete.com/school/advanced-haskell/beautiful-concurrency>
- [17] Anatomy of an MVar operation
<http://blog.ezyang.com/2013/05/anatomy-of-an-mvar-operation/>
- [18] Parallel and Concurrent Programming in Haskell
<http://community.haskell.org/~simonmar/pcph/>
- [19] Real World Haskell
<http://book.realworldhaskell.org/>

References

The GHC Commentary

- [C1] <https://ghc.haskell.org/trac/ghc/wiki/Commentary>
- [C2] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/SourceTree>
- [C3] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler>
- [C4] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/HscMain>
- [C5] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/CoreSynType>
- [C6] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/StgSynType>
- [C7] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/CmmType>
- [C8] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/GeneratedCode>
- [C9] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/SymbolNames>
- [C10] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts>
- [C11] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/HeapObjects>
- [C12] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/Stack>
- [C13] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/GC>
- [C14] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution>
- [C15] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution/Registers>
- [C16] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution/PointerTagging>
- [C17] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Scheduler>
- [C18] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/STM>
- [C19] <https://ghc.haskell.org/trac/ghc/wiki/Commentary/Libraries>

References

Source code

- [S1] `includes/stg/Regs.h`
- [S2] `includes/stg/MachRegs.h`
- [S3] `includes/rts/storage/ClosureTypes.h`
- [S4] `includes/rts/storage/Closures.h`
- [S5] `includes/rts/storage/TSO.h`
- [S6] `includes/rts/storage/InfoTables.h`
- [S7] `compiler/main/DriverPipeline.hs`
- [S8] `compiler/main/HscMain.hs`
- [S9] `compiler/cmm/CmmParse.y.source`
- [S10] `compiler/codeGen/StgCmmForeign.hs`
- [S11] `compiler/codeGen/Stg*.hs`
- [S12] `rts/PrimOps.cmm`
- [S13] `rts/RtsMain.c`
- [S14] `rts/RtsAPI.c`
- [S15] `rts/Capability.h`
- [S16] `rts/Capability.c`
- [S17] `rts/Schedule.c`
- [S18] `rts/StgCRun.c`
- [S19] `rts/StgStartup.cmm`
- [S20] `rts/StgMiscClosures.cmm`
- [S21] `rts/HeapStackCheck.cmm`
- [S22] `rts/Threads.c`
- [S23] `rts/Task.c`
- [S24] `rts/Timer.c`
- [S25] `rts/sm/GC.c`
- [S26] `rts/Sparks.c`
- [S27] `rts/WSDeque.c`
- [S28] `rts/STM.h`
- [S29] `rts/posix/Signals.c`
- [S30] `rts/win32/ThrIOManager.c`
- [S31] `libraries/base/GHC/MVar.hs`
- [S32] `libraries/base/GHC/Conc/IO.hs`
- [S33] `libraries/base/GHC/Conc/Sync.lhs`
- [S34] `libraries/base/GHC/Event/Manager.hs`
- [S35] `libraries/base/GHC/Event/Thread.hs`
- [S36] `libraries/base/GHC/IO/BufferedIO.hs`
- [S37] `libraries/base/GHC/IO/FD.hs`
- [S38] `libraries/base/GHC/IO/Handle/Text.hs`
- [S39] `libraries/base/System/IO.hs`
- [S40] `libraries/base/System/Posix/Internals.hs`
- [S41] `AutoApply.o (utils/genapply/GenApply.hs)`

Connect the algorithm and transistor