GHC(STG,Cmm,asm) illustrated

for hardware persons

exploring some mental models and implementations

Takenobu T.

Rev. 0.03.0
WIP

“Any sufficiently advanced technology is indistinguishable from magic.”

Arthur C. Clarke

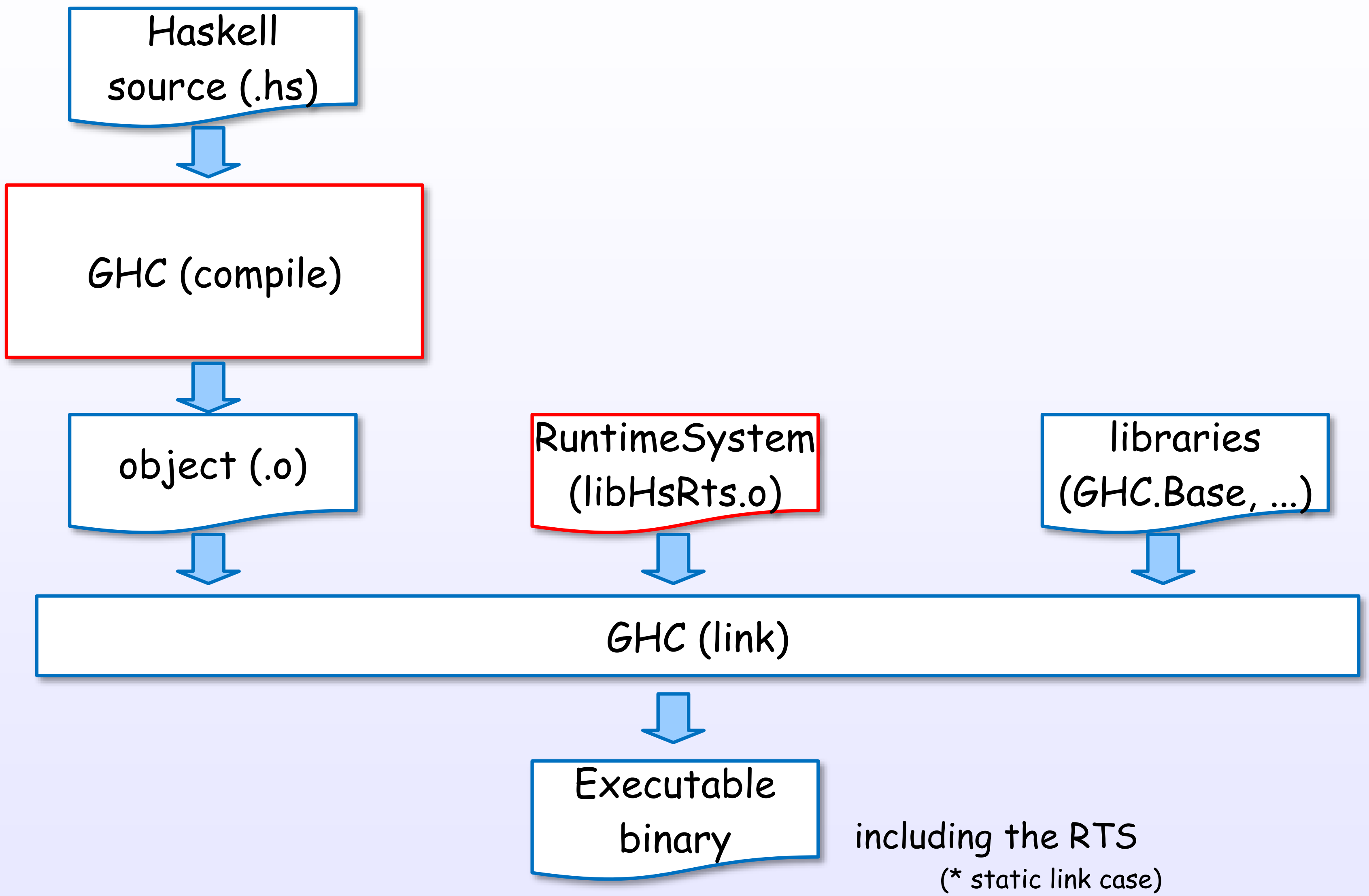
NOTE

- This is not an official document by the ghc development team.
- Please don't forget “semantics”. It's very important.
- This is written for ghc 8.6.

Contents

Executable binary

The GHC = Compiler + Runtime System (RTS)

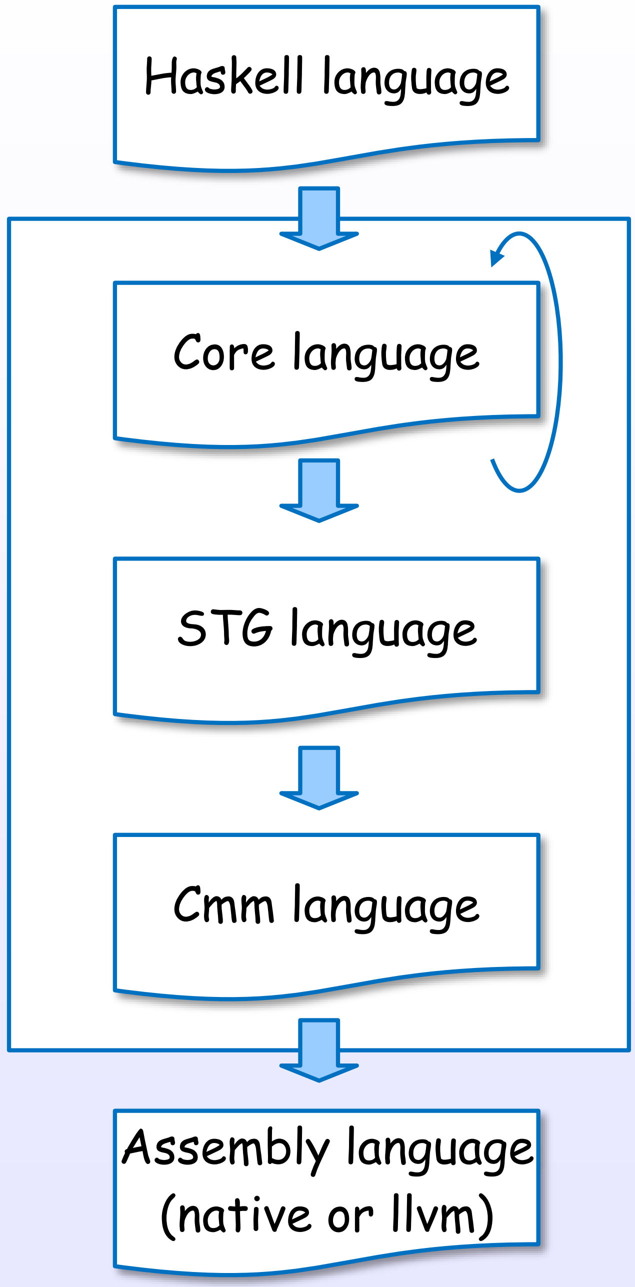


References : [1], [C1], [C3], [C12], [C21], [S7], [21], [22]

Compile steps

GHC transitions between five representations

GHC
compile
steps



each intermediate code can be dumped by :

$ ghc -ddump-parsed
$ ghc -ddump-rn

$ ghc -ddump-ds
$ ghc -ddump-simpl
$ ghc -ddump-prep

$ ghc -ddump-stg

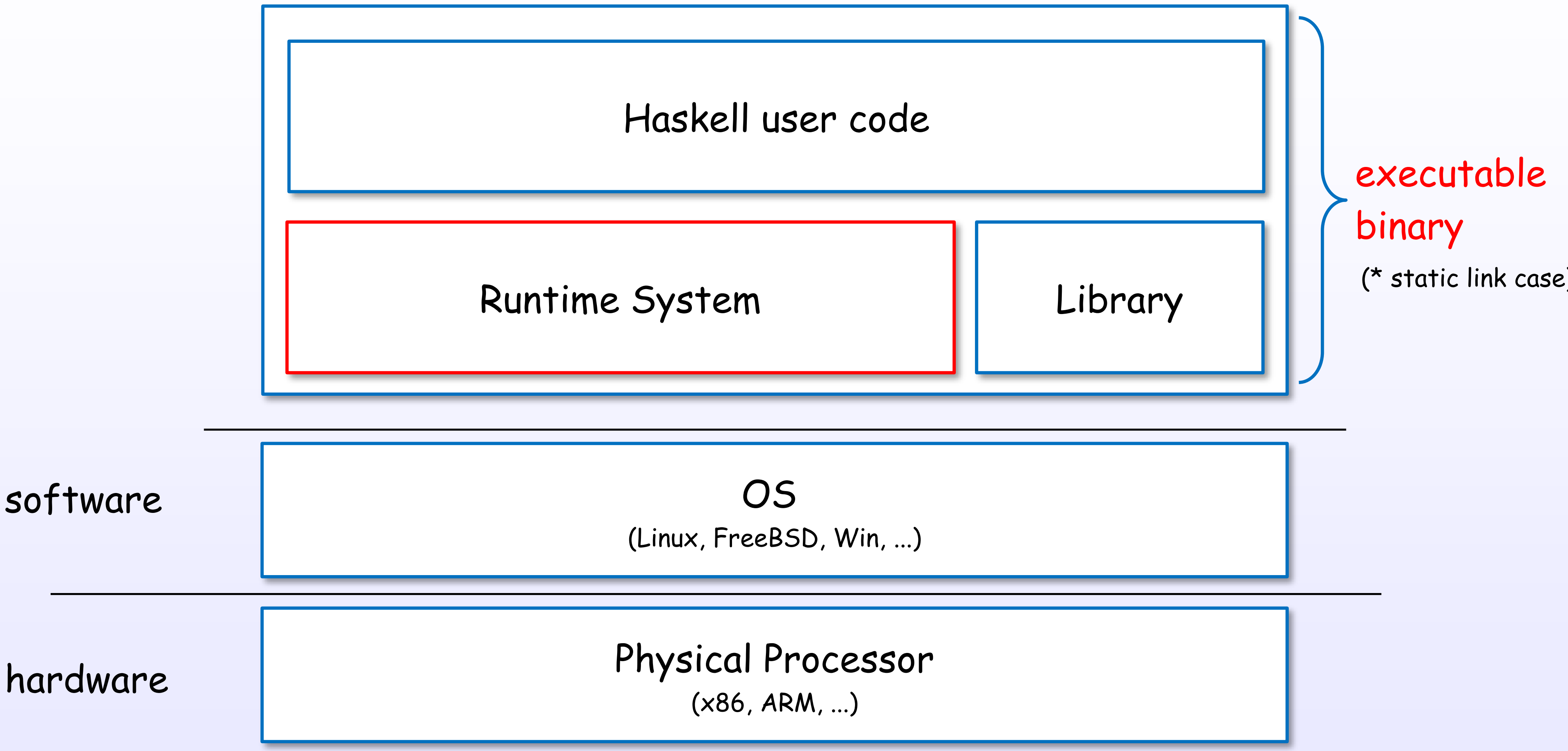
$ ghc -ddump-cmm
$ ghc -ddump-opt-cmm

$ ghc -ddump-llvm
$ ghc -ddump-asm

References : [1], [C3], [C4], [9], [C5], [C6], [C7], [C8], [S7], [S8], [21], [22]

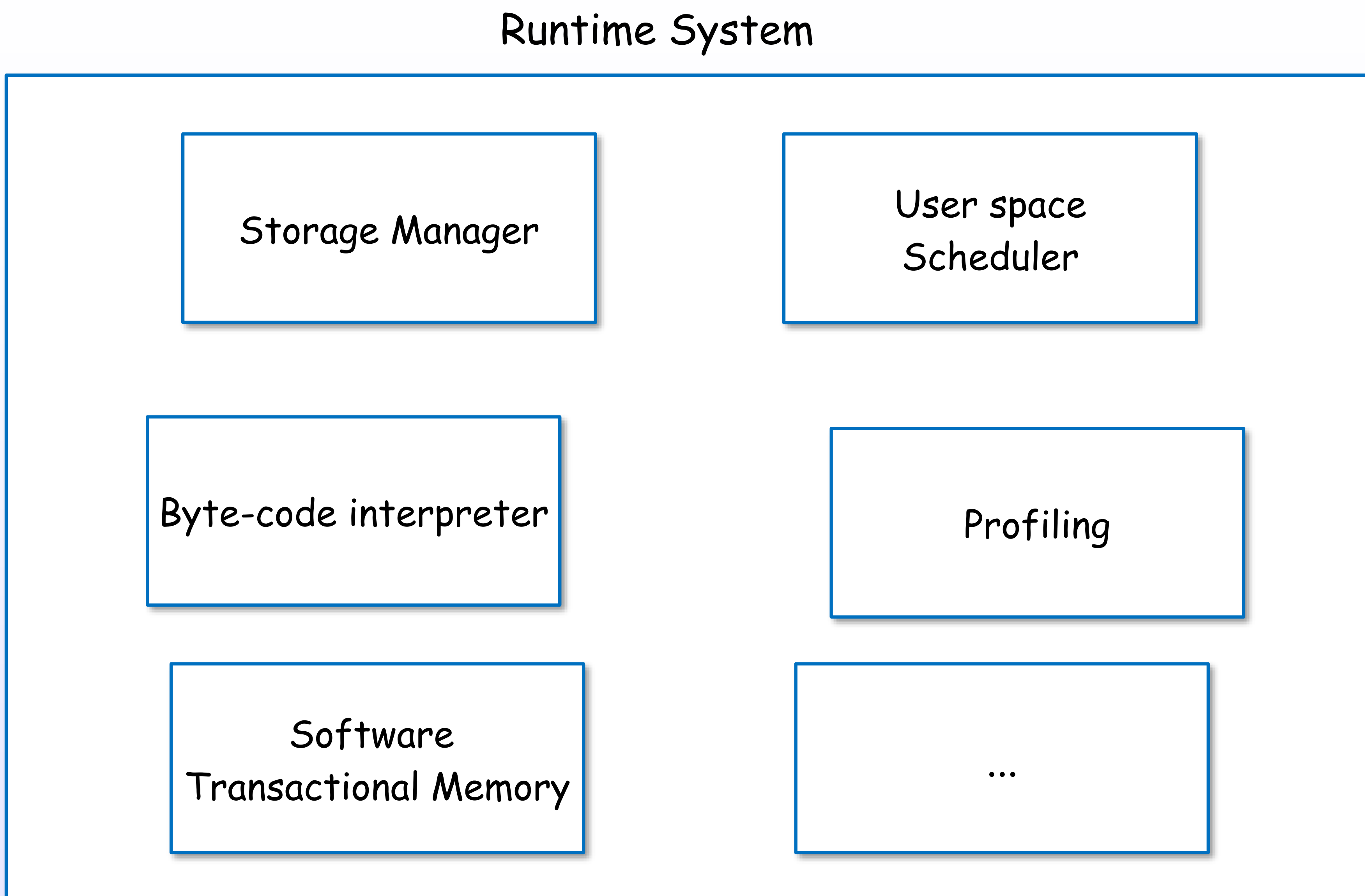
Runtime System

Generated binary includes the RTS



References : [C12], [9]

Runtime System includes ...



References : [C12], [8], [9], [5], [17], [S13]

Development languages

The GHC is 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

References : [C2], [22]

Machine layer/models

Machine layer

STG-machine
(Abstract machine)

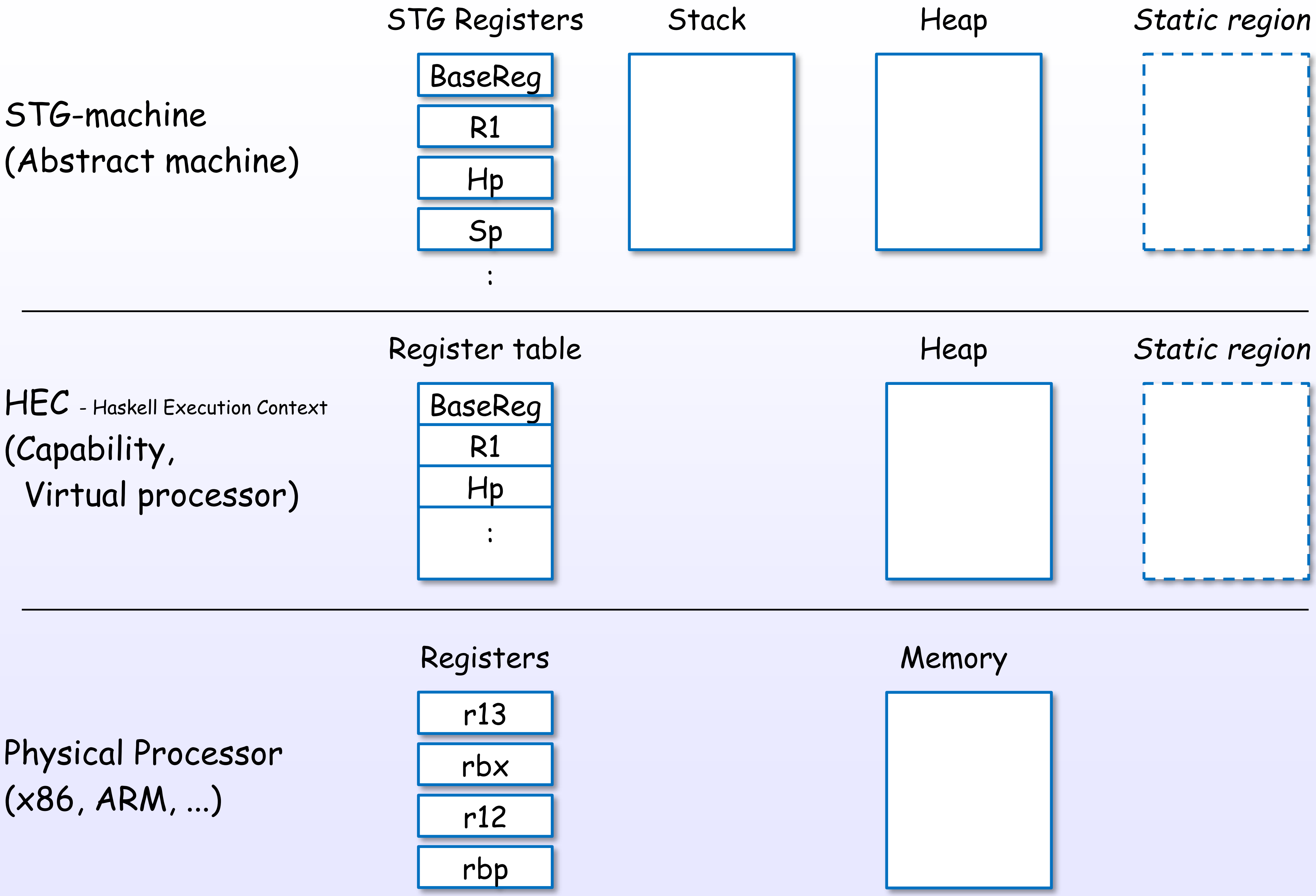
HEC - Haskell Execution Context
(Capability, Virtual processor)

Physical Processor
(x86, ARM, ...)

Each Haskell code is executed in STG semantics.

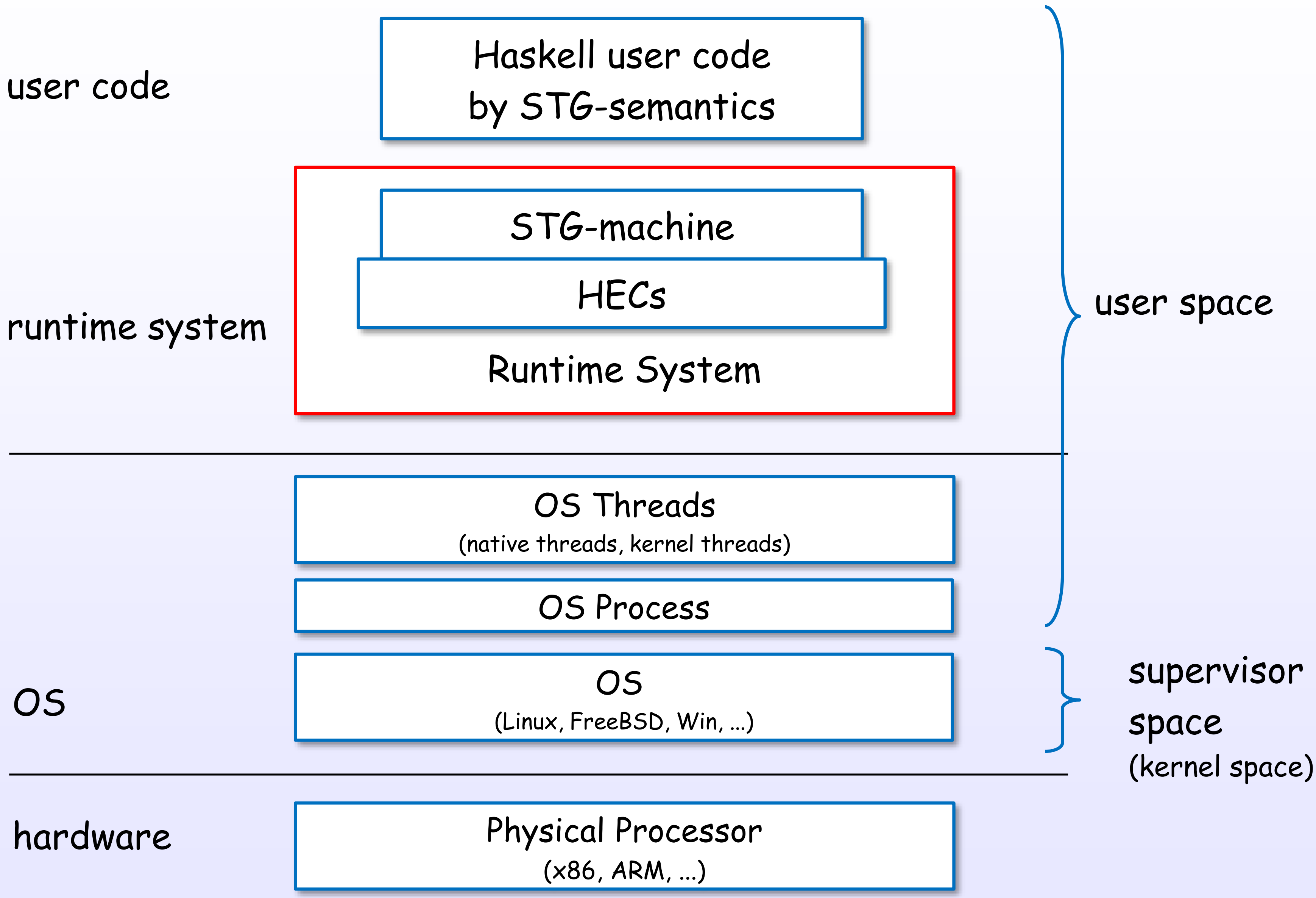
References : [C16], [C6], [2], [C19], [8], [S15], [S16], [S11]

Machine layer



References : [C16], [C6], [2], [C19], [8], [S15], [S16], [S11]

Runtime system and HEC

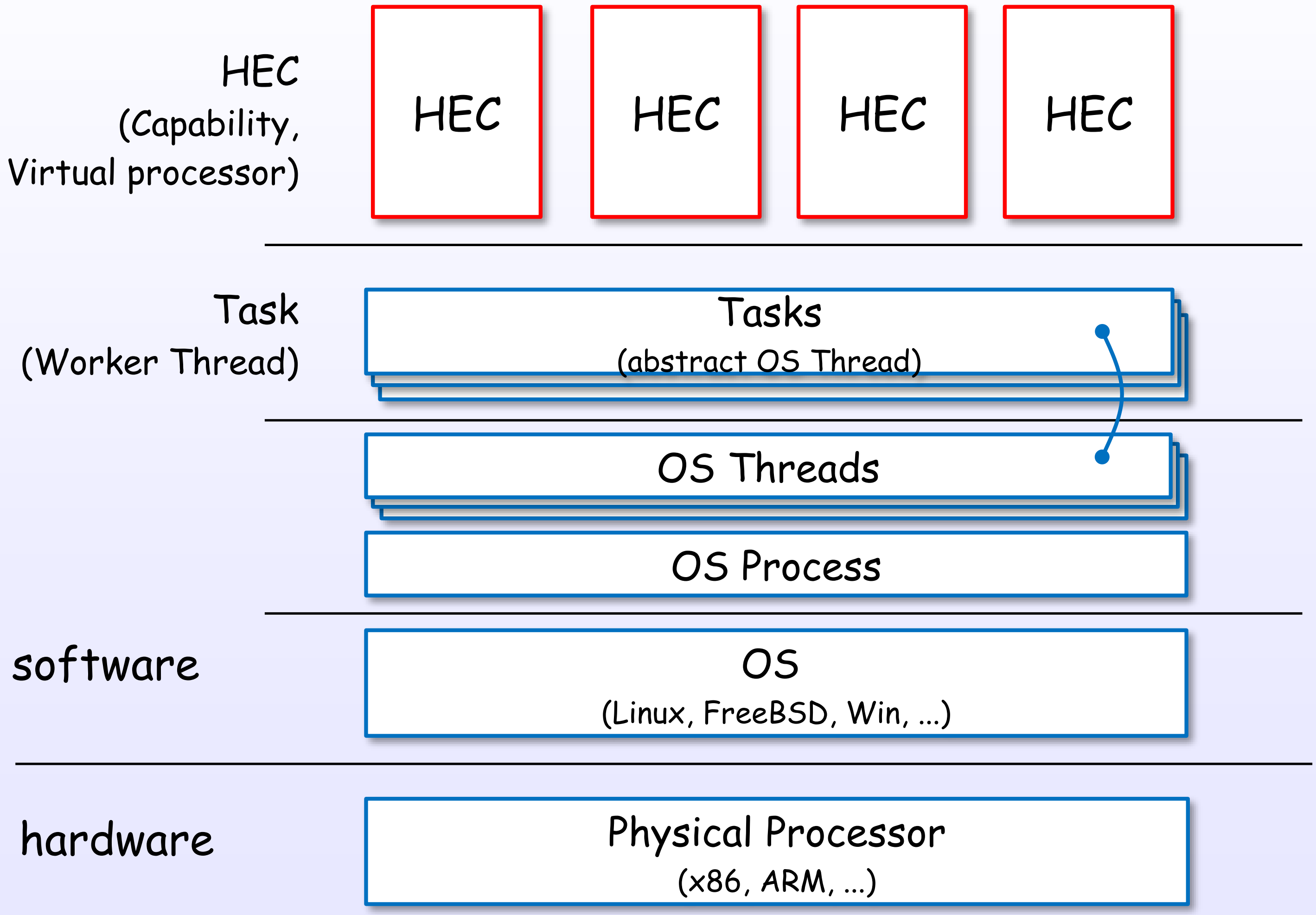


References : [C16], [C6], [2], [C19], [8], [S15], [S16], [S11]

many HECs

Multi HECs can be generated by compile and runtime options :

```
$ ghc -rtsopts -threaded
$ ./xxx +RTS -N4
```



References : [1], [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

HEC (Capability) data structure

[rts/Capability.h] (ghc 8.6)

```
struct Capability_ {
    StgFunTable f;
    StgRegTable r;
    uint32_t no;
    uint32_t node;
    Task *running_task;
    bool in_haskell;
    uint32_t idle;
    bool disabled;
    StgTSO *run_queue_hd;
    StgTSO *run_queue_tl;
    uint32_t n_run_queue;
    InCall *suspended_ccalls;
    uint32_t n_suspended_ccalls;
    bdescr **mut_lists;
    bdescr **saved_mut_lists;
    bdescr *pinned_object_block;
    bdescr *pinned_object_blocks;
    StgWeak *weak_ptr_list_hd;
    StgWeak *weak_ptr_list_tl;
    int context_switch;
    int interrupt;
    W_ total_allocated;

#if defined(THREADED_RTS)
    Task *spare_workers;
    uint32_t n_spare_workers;
    Mutex lock;
    Task *returning_tasks_hd;
    Task *returning_tasks_tl;
    uint32_t n_returning_tasks;
    Message *inbox;
    struct PutMVar_ *putMVars;
    SparkPool *sparks;
    SparkCounters spark_stats;
#endif

    StgTVarWatchQueue *free_tvar_watch_queues;
    StgTRecChunk *free_trec_chunks;
    StgTRecHeader *free_trec_headers;
    uint32_t transaction_tokens;
}
```

register table

run queue

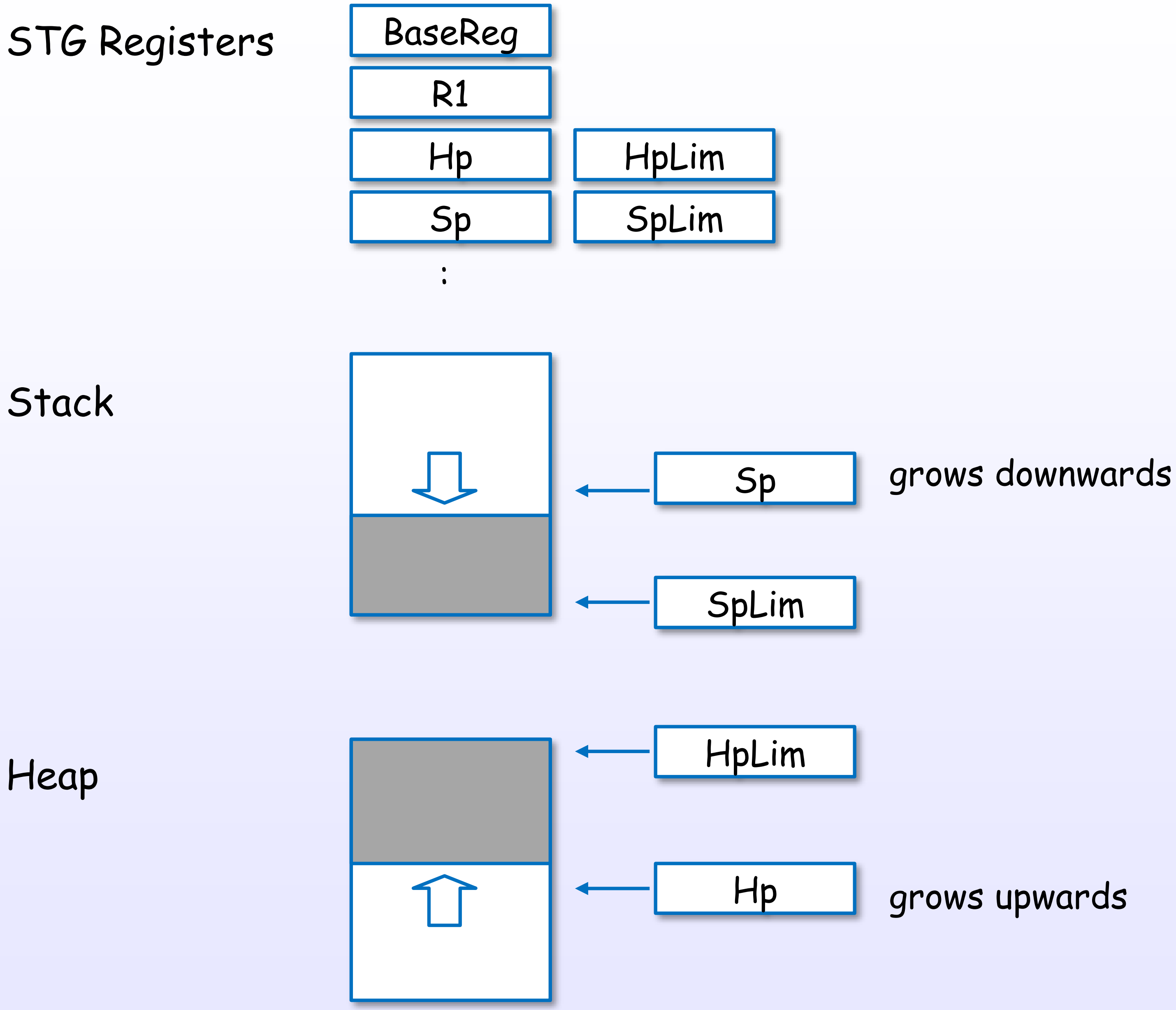
Each HEC (Capability) has a register table and a run queue and ...

Each HEC (Capability) is initialized at initCapabilities [rts/Capability.c]

References : [S15], [S16], [C13], [C19]

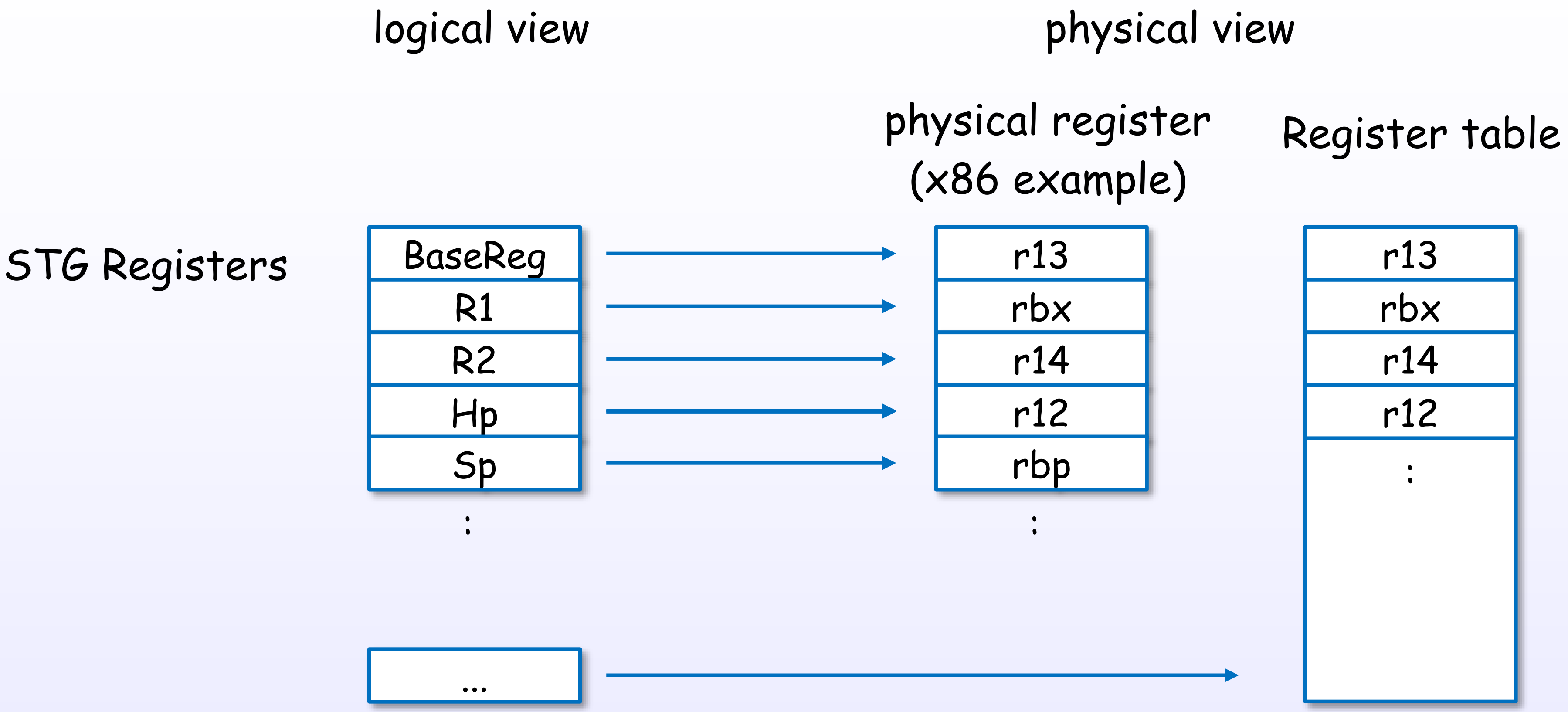
STG-machine

The STG-machine consists of three parts



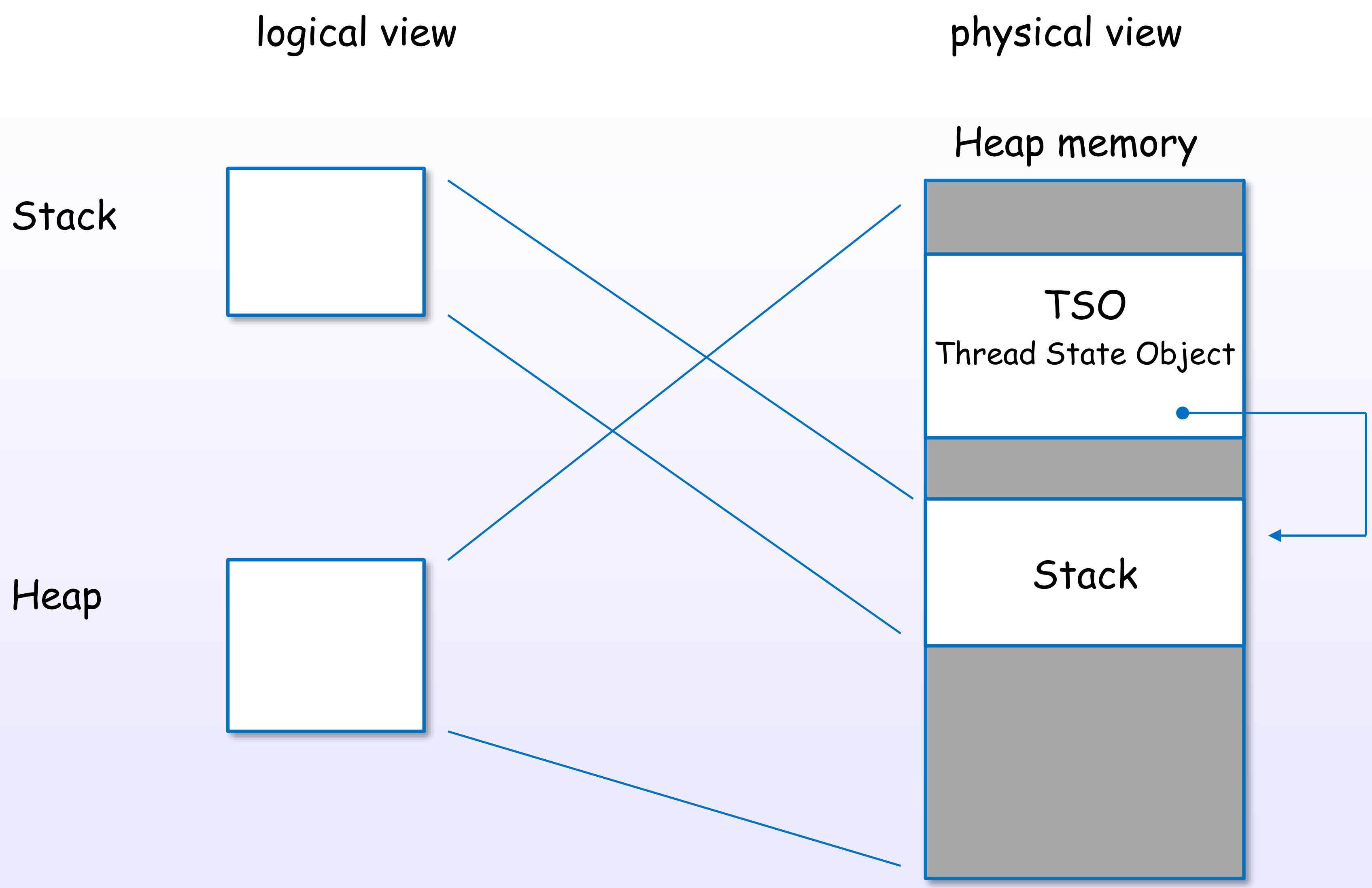
References : [2], [C17], [C13], [C14]

STG-machine is mapped to physical processor



References : [C17], [S1], [S2]

STG-machine is mapped to physical processor



A stack and a TSO object are in the heap.
The stack is stored separately from the TSO for size extension and GC.

References : [C13], [C14], [S16], [S5]

TSO data structure

[includes/rts/storage/TSO.h] (ghc 8.6)

```
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;
    StgInt64  alloc_limit;
    StgWord32  tot_stack_size;
} *StgTSOPtr;
```

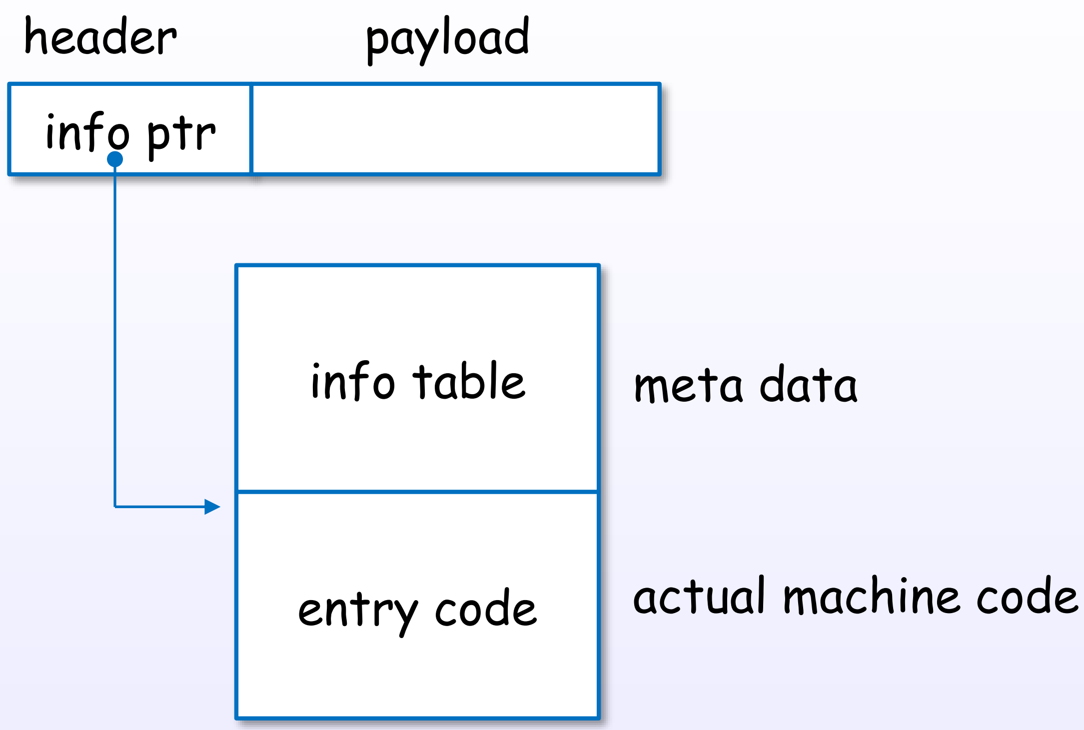
link to stack object

A TSO object is only ~18words + stack. Lightweight!

References : [S5]

Heap objects in STG-machine

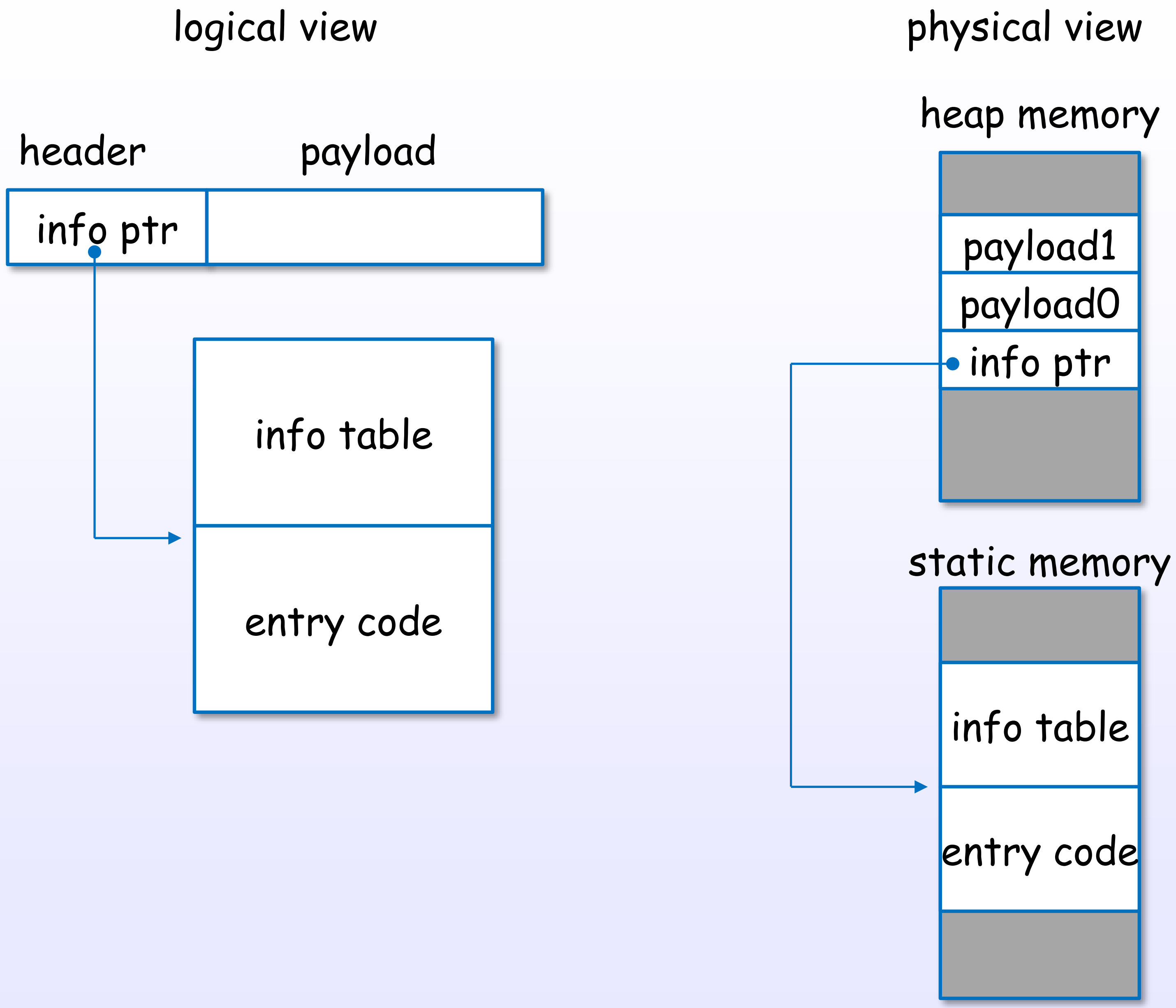
Every heap object is represented uniformly



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

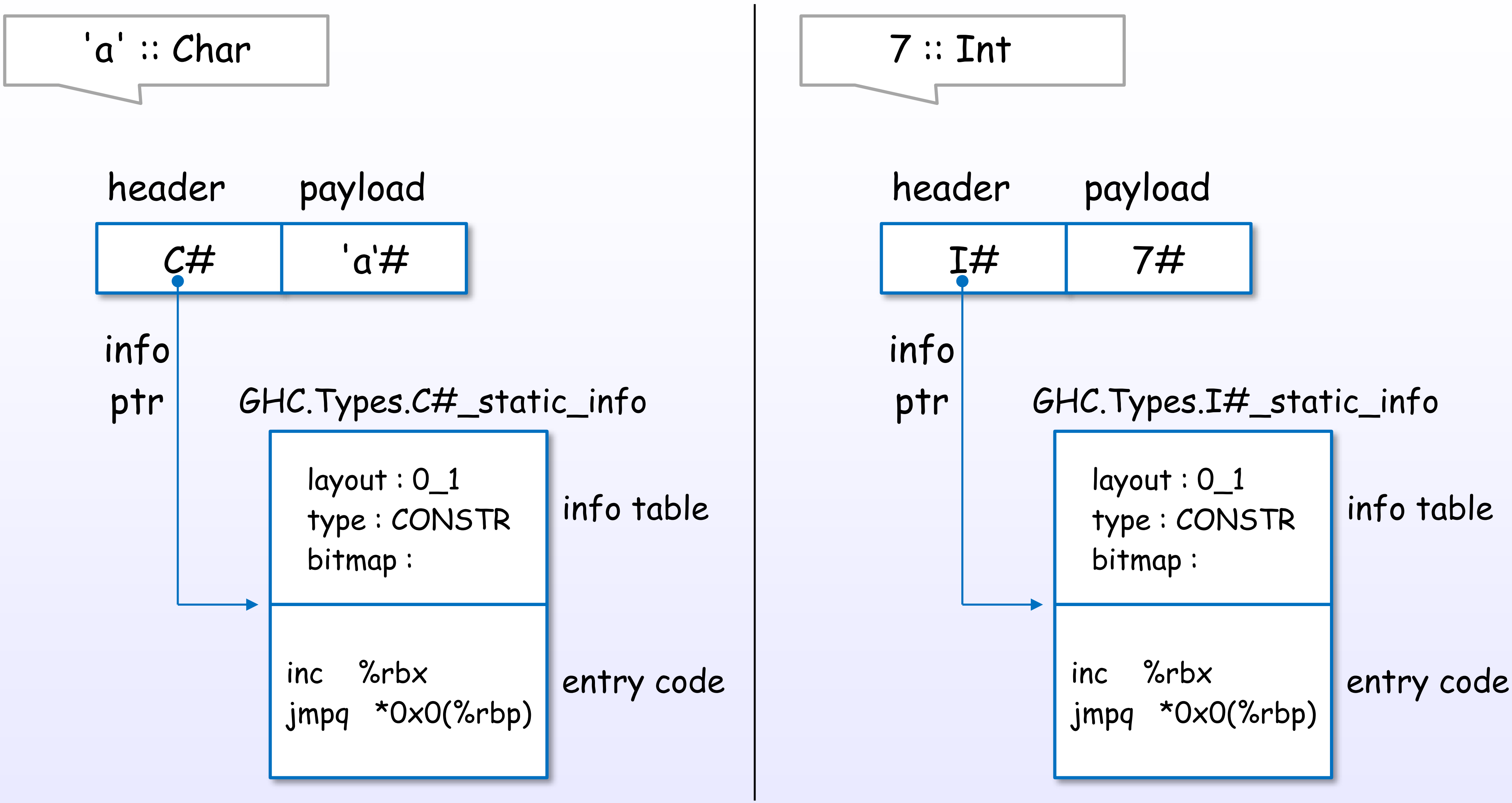
References : [C13], [S3], [S4], [S6], [2]

Heap object (closure)



References : [C13], [S3], [C9], [C8], [2], [12], [13]

Closure examples : Char, Int



References : [C13], [S3], [C9], [C8], [2], [S20], [13]

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=m, Unf=OtherCon []] =
    CCS_DONT_CARE GHC.Types.I#! [7#];
```

Cmm

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

```
section ""data" . Example.value1_closure" {
    Example.value1_closure:
        const GHC.Types.I#_con_info;
        const 7;
}
```

asm

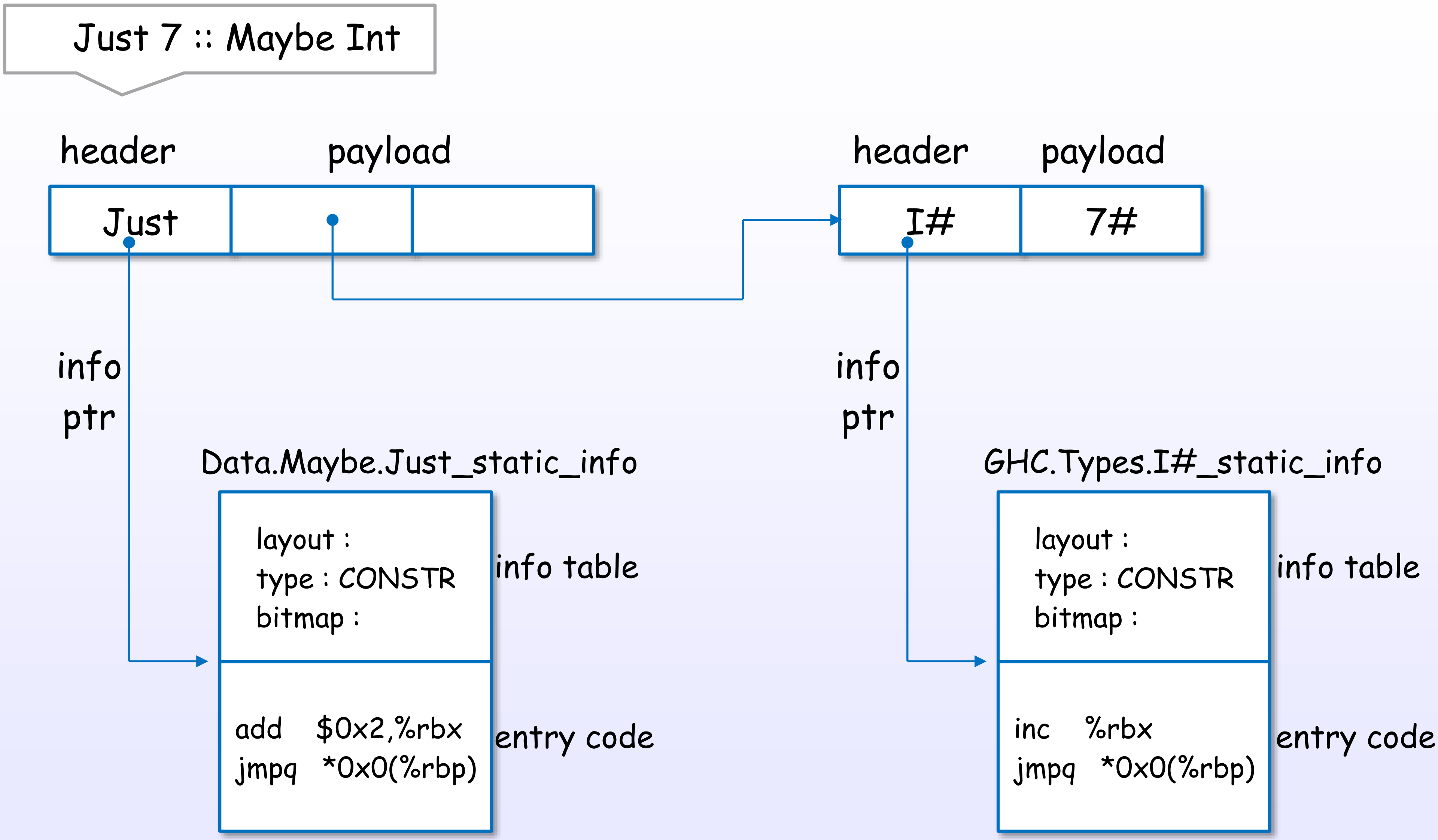
[ghc -O -ddump-asm Example.hs]

```
.section .data
.align 8
.align 1
.globl Example.value1_closure
.type Example.value1_closure, @object
Example.value1_closure:
        .quad   GHC.Types.I#_con_info
        .quad   7
```

header	payload
I#	7#

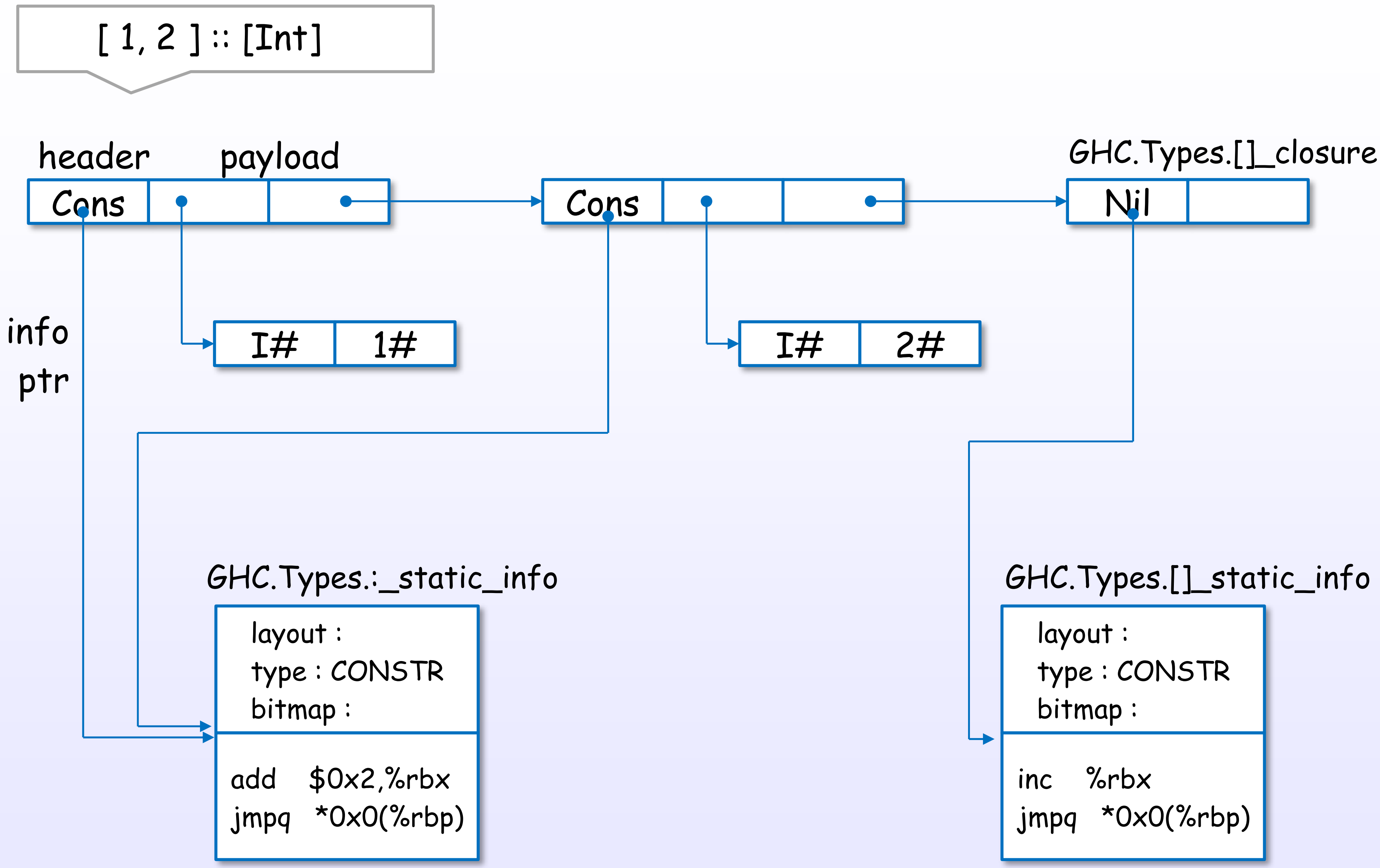
References : [C13], [S3], [C9], [C8], [2], [S20]

Closure examples : Maybe



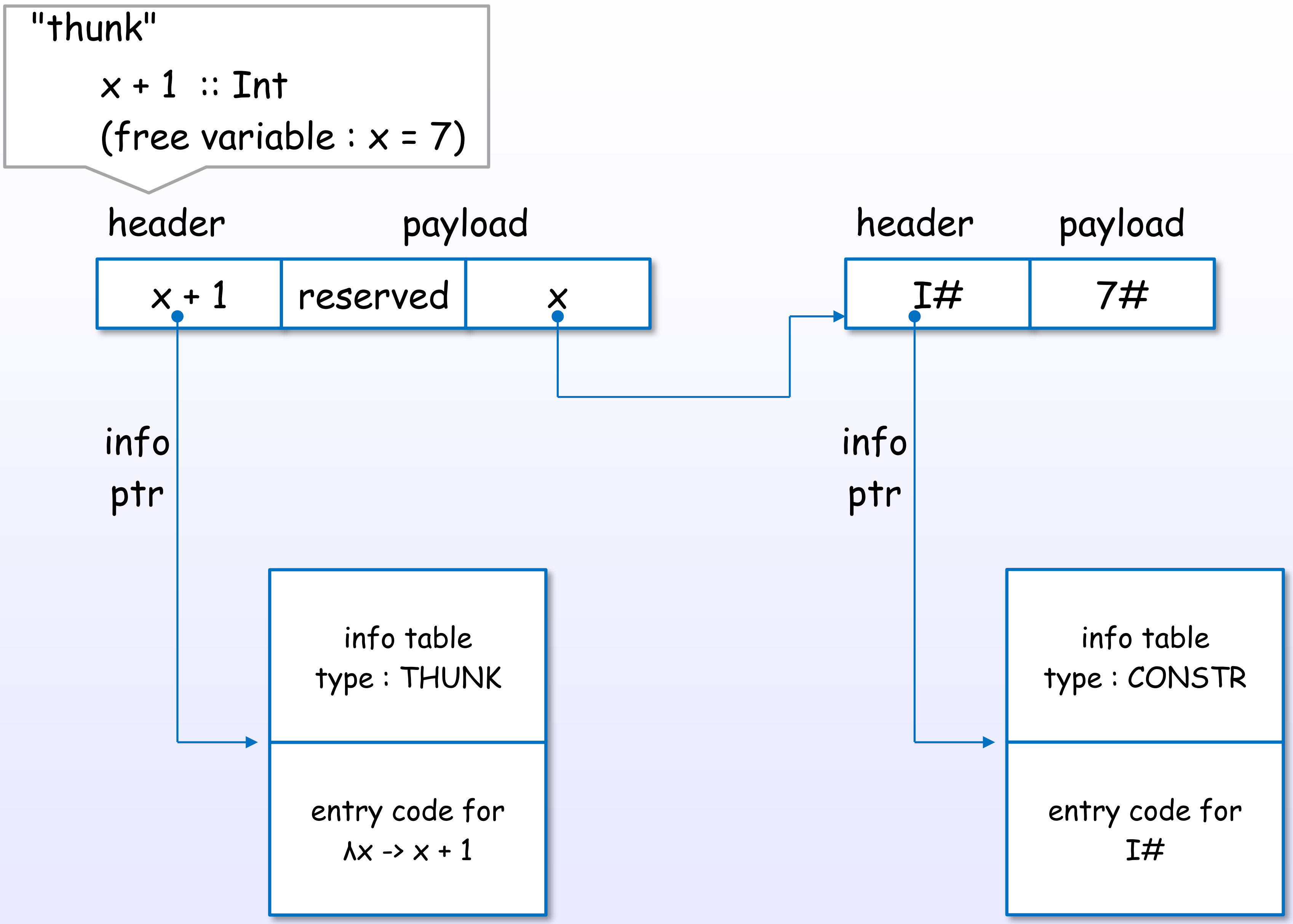
References : [C13], [S3], [C9], [C8], [2], [S20]

Closure examples : List



References : [C13], [S3], [C9], [C8], [2], [S20]

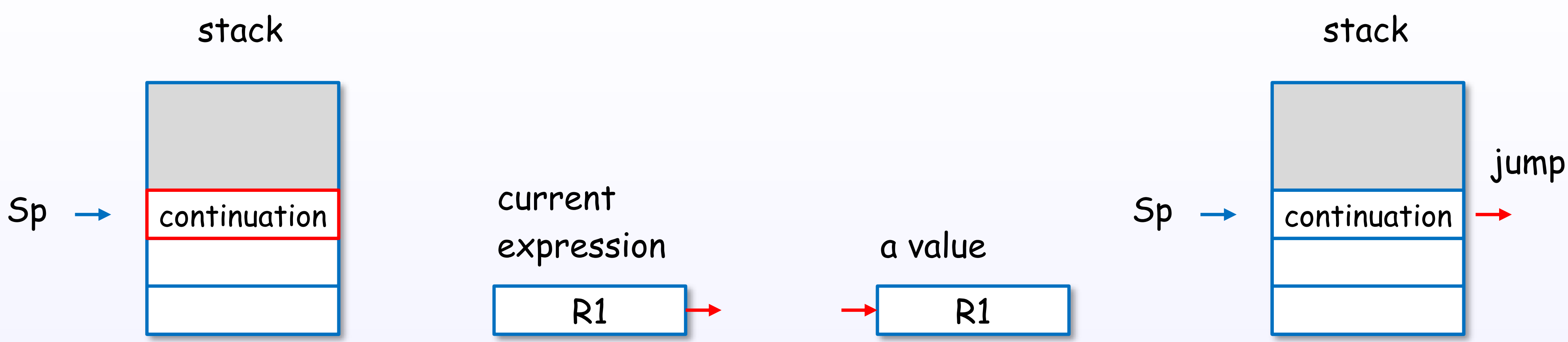
Closure examples : Thunk



References : [C13], [S3], [C9], [C8], [2], [S20]

STG-machine evaluation

STG evaluation flow



(1)
push a continuation code (next code) to the stack top

(2)
enter to R1 closure

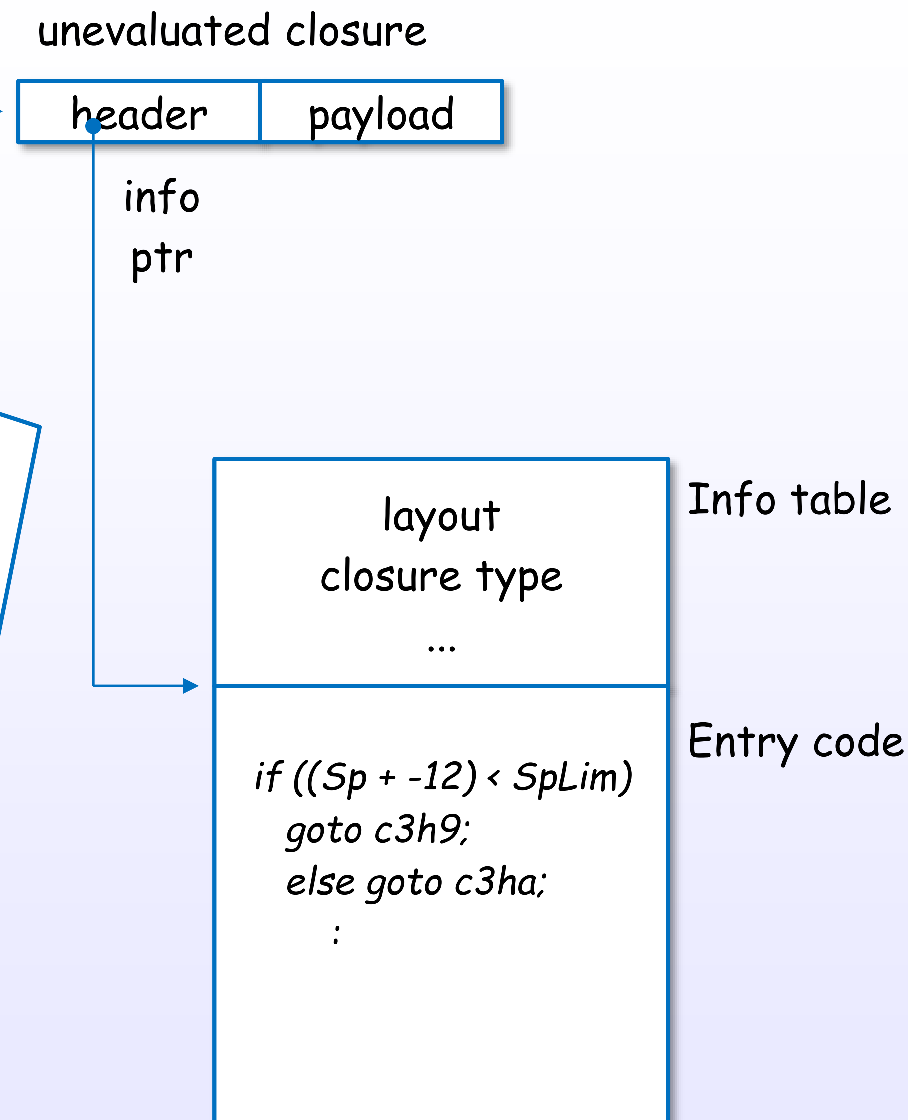
(3)
set a result to R1

(4)
jump (return) to the stack top code

(5)
repeat from (1)

References : [C8], [3], [12], [13]

Enter to a closure



(1) read R1
to get a closure address

(2) read header(info ptr)
to get a Entry code address

(3) jump to the Entry code address

(4) execute the Entry code

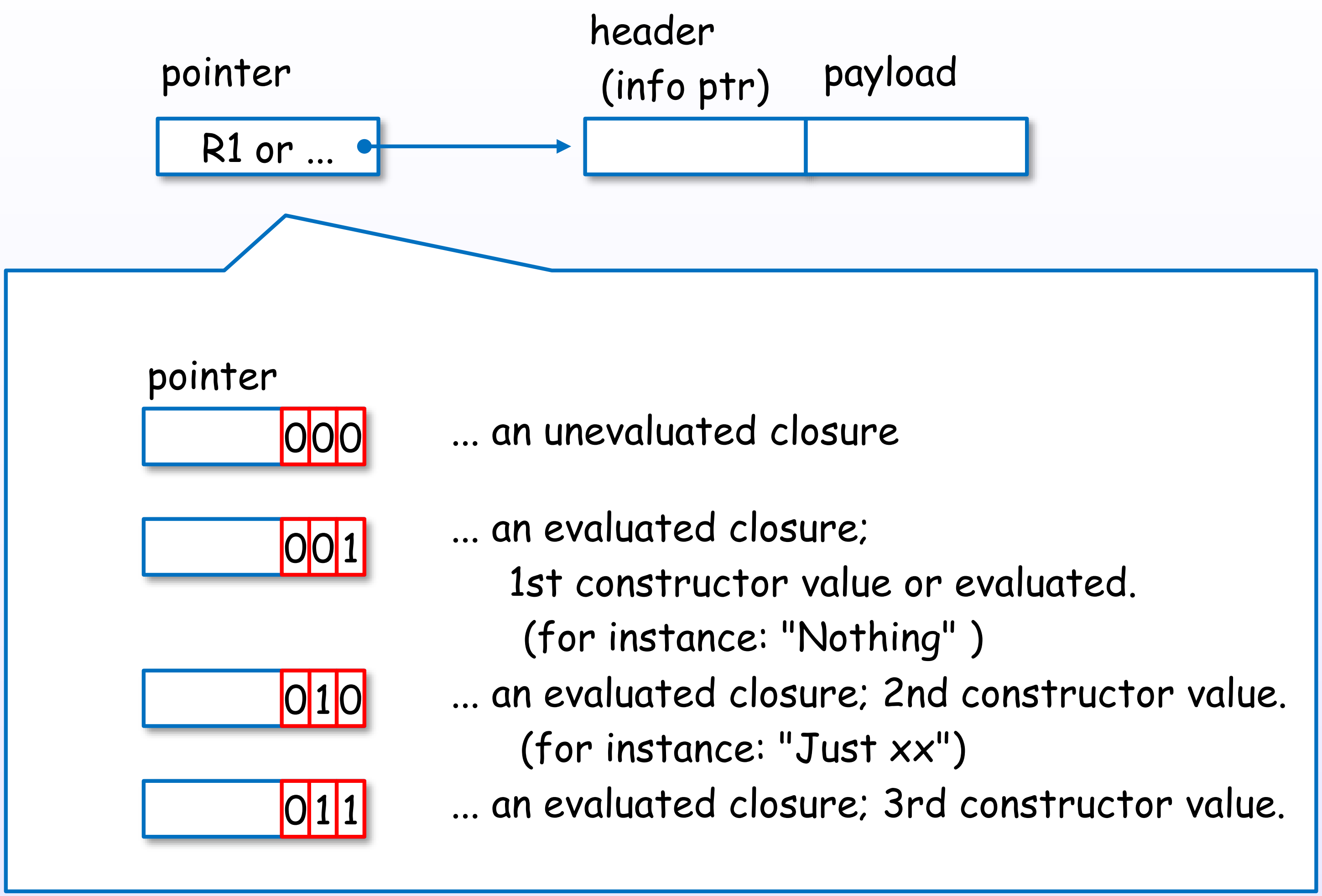
(5) set a result to R1

(6) jump to the stack top address
(continuation)

References : [C13], [C9], [C8], [10], [3], [2], [12], [13]

Pointer tagging

Pointer tagging



* 64bit machine case

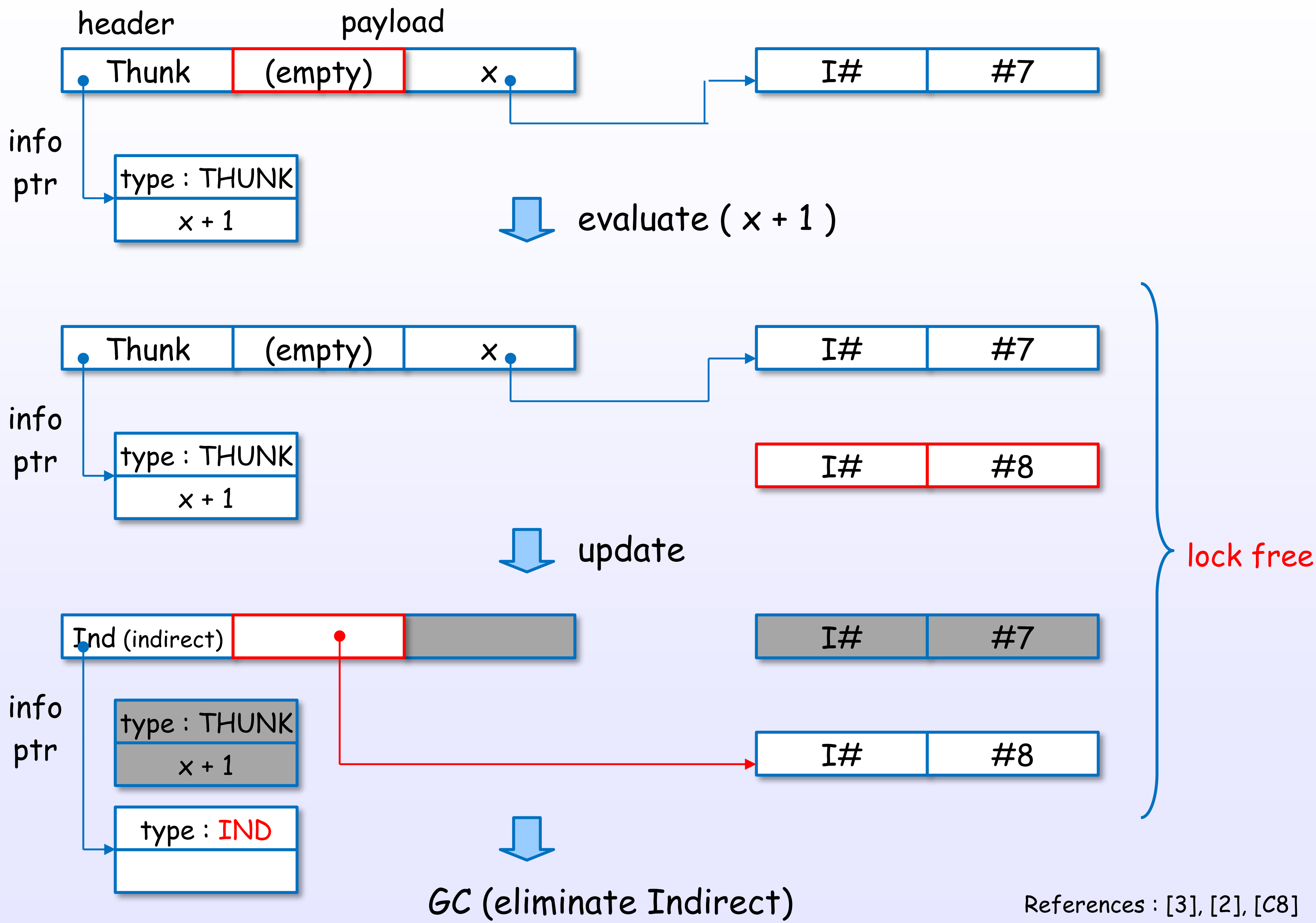
fast judgment!
check only pointer's lower bits without evaluating the closure.

References : [4], [2], [C18], [12], [13]

Thunk and update

Thunk and update

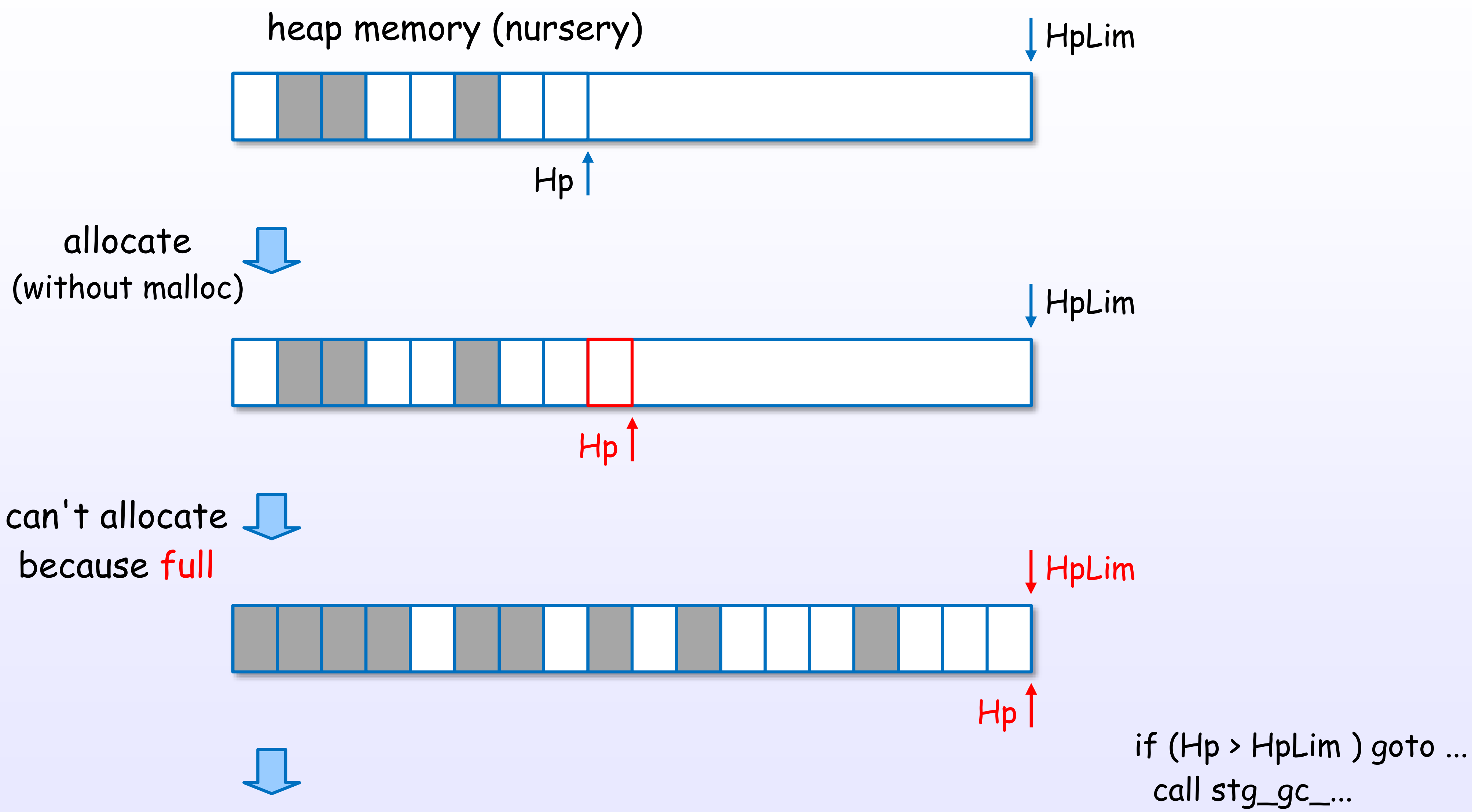
"thunk" x + 1 :: Int (free variable : x = 7)



References : [3], [2], [C8]

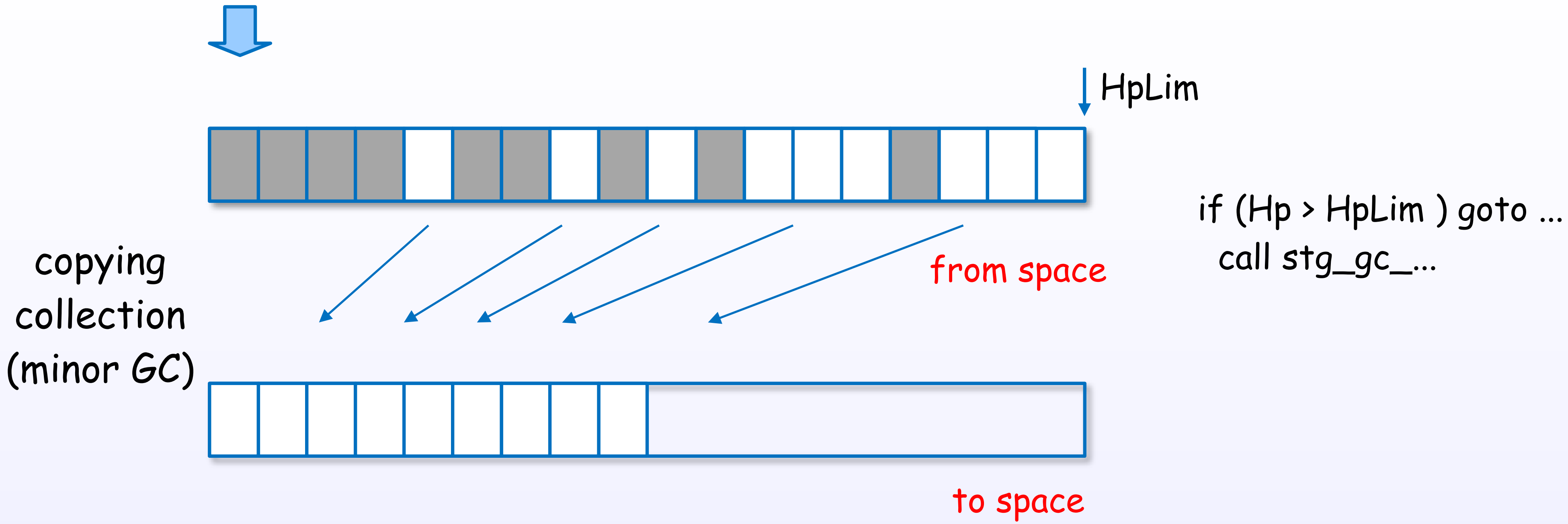
Allocate and free heap objects

Allocate heap objects



References : [C13], [C15], [8], [9], [5], [15], [12], [13], [19], [S25]

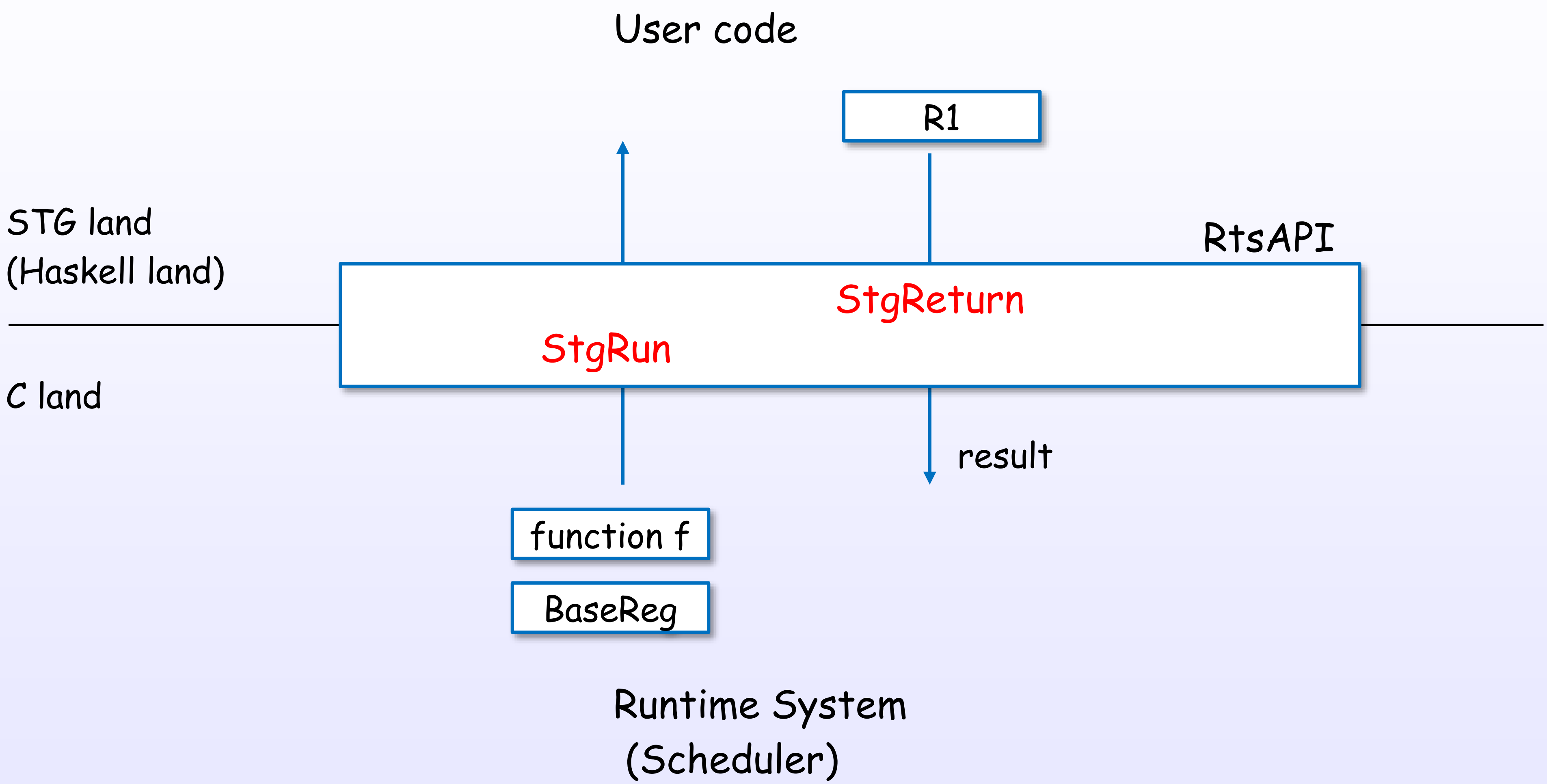
free and collect heap objects



References : [C13], [C15], [8], [9], [5], [15], [12], [13], [19], [S25]

STG - C land interface

STG (Haskell) land - C land interface

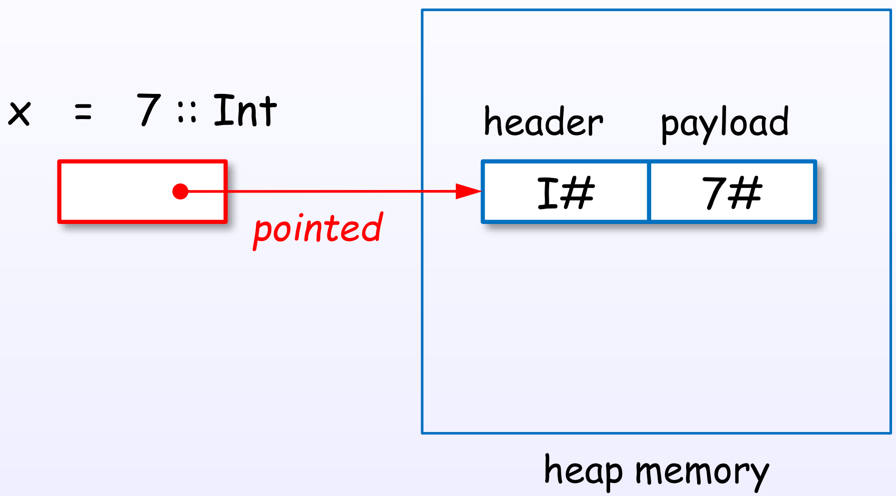


References : [S18], [S17], [S19], [S21]

Boxity : boxed and unboxed

Boxed and unboxed types

A boxed type



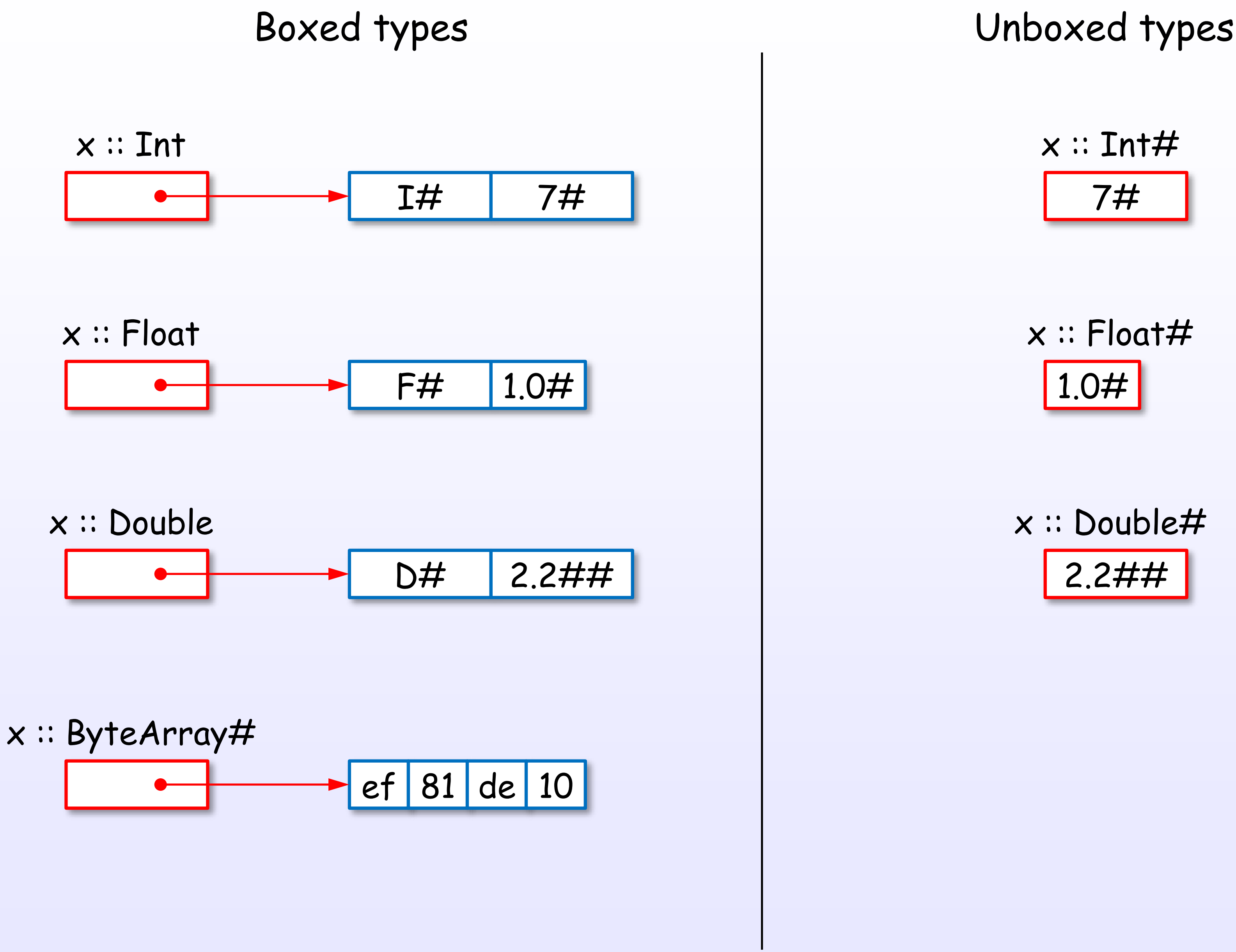
A boxed type is represented by a pointer to a heap-allocated object.

An unboxed type

An unboxed type is represented by the value itself.

References : [C10], [C13], [S31], [23], [24]

Boxity examples

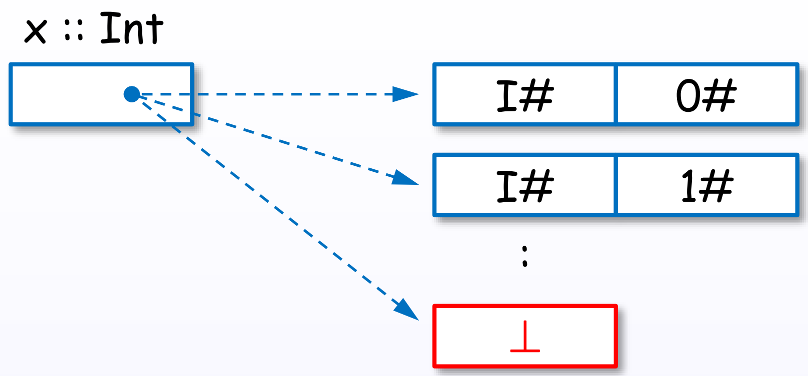


References : [C10], [C13], [S31], [23], [24]

Levity : lifted and unlifted

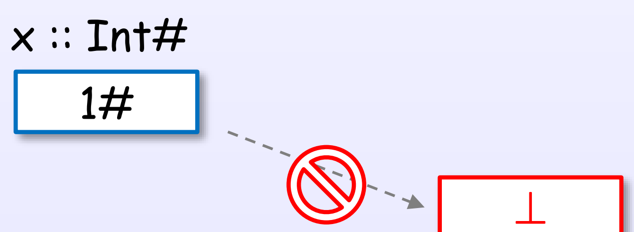
Lifted and unlifted types

A lifted type



A lifted type has one extra element
representing a non-terminating computation (bottom, $\perp$).

An unlifted type

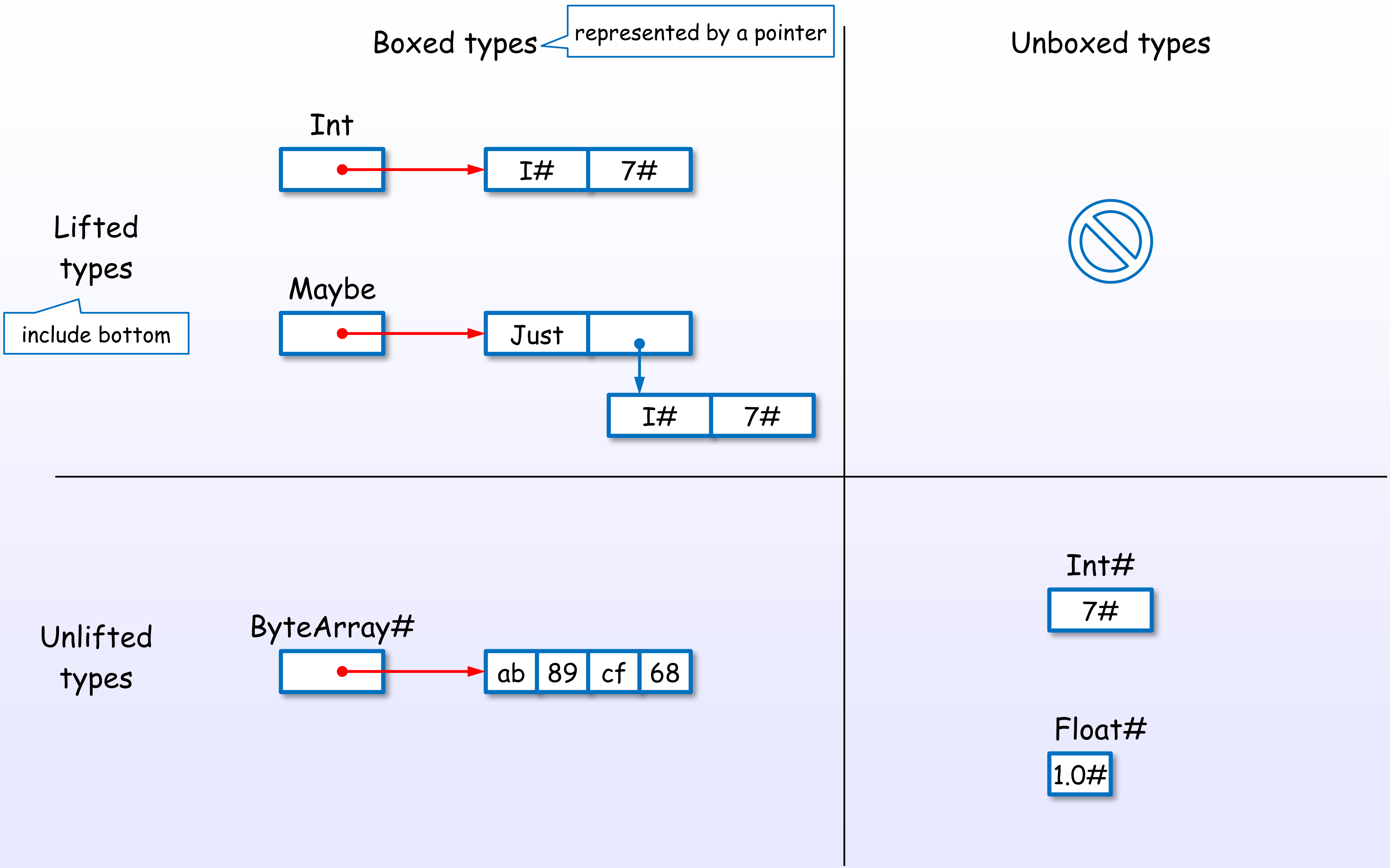


An unlifted type has no bottom.

References : [C11], [C10], [C13], [S31], [24], [23]

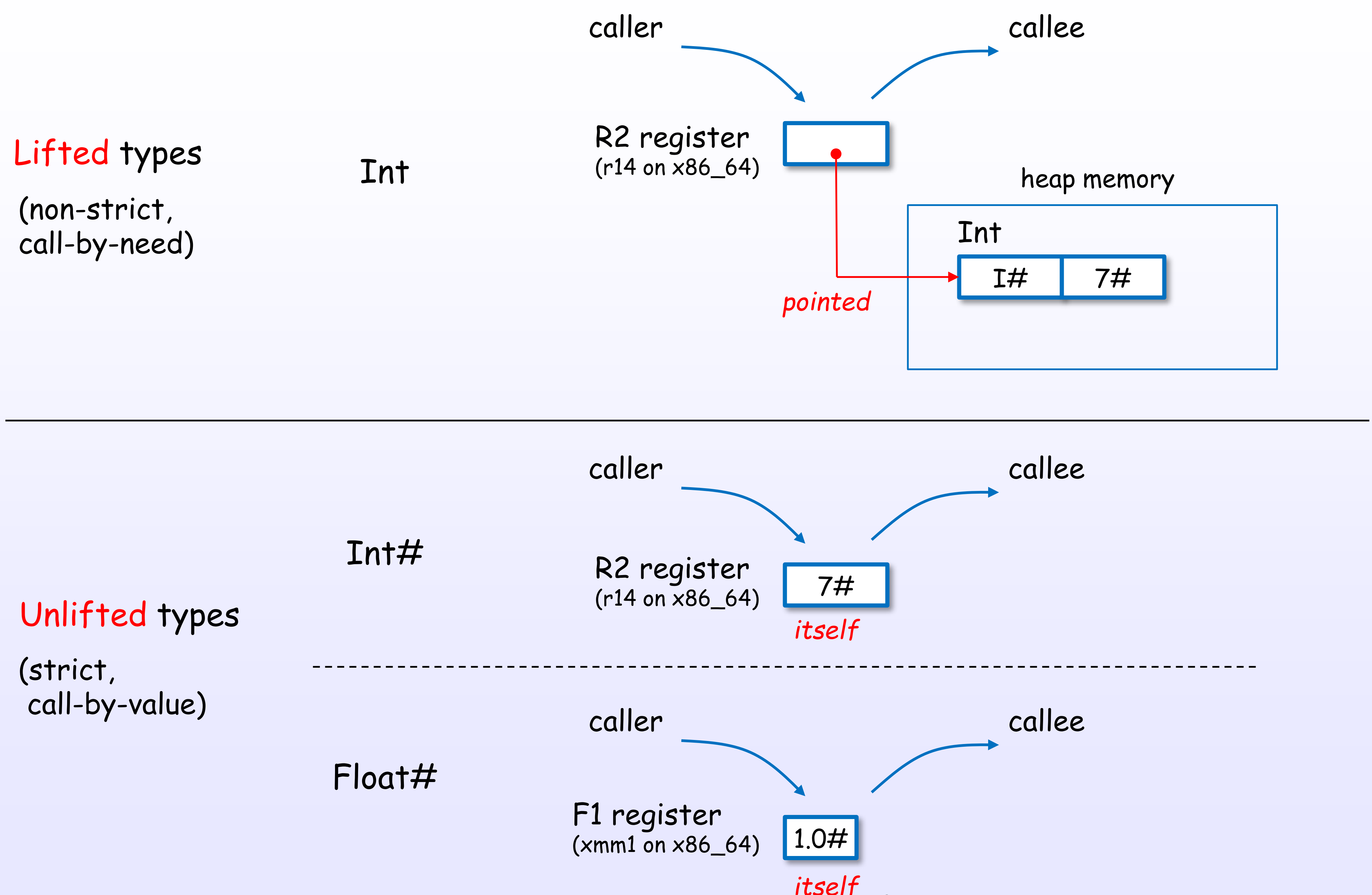
Boxity and levity

Boxity and levity examples



References : [24], [C11], [C10], [C13], [S31], [23]

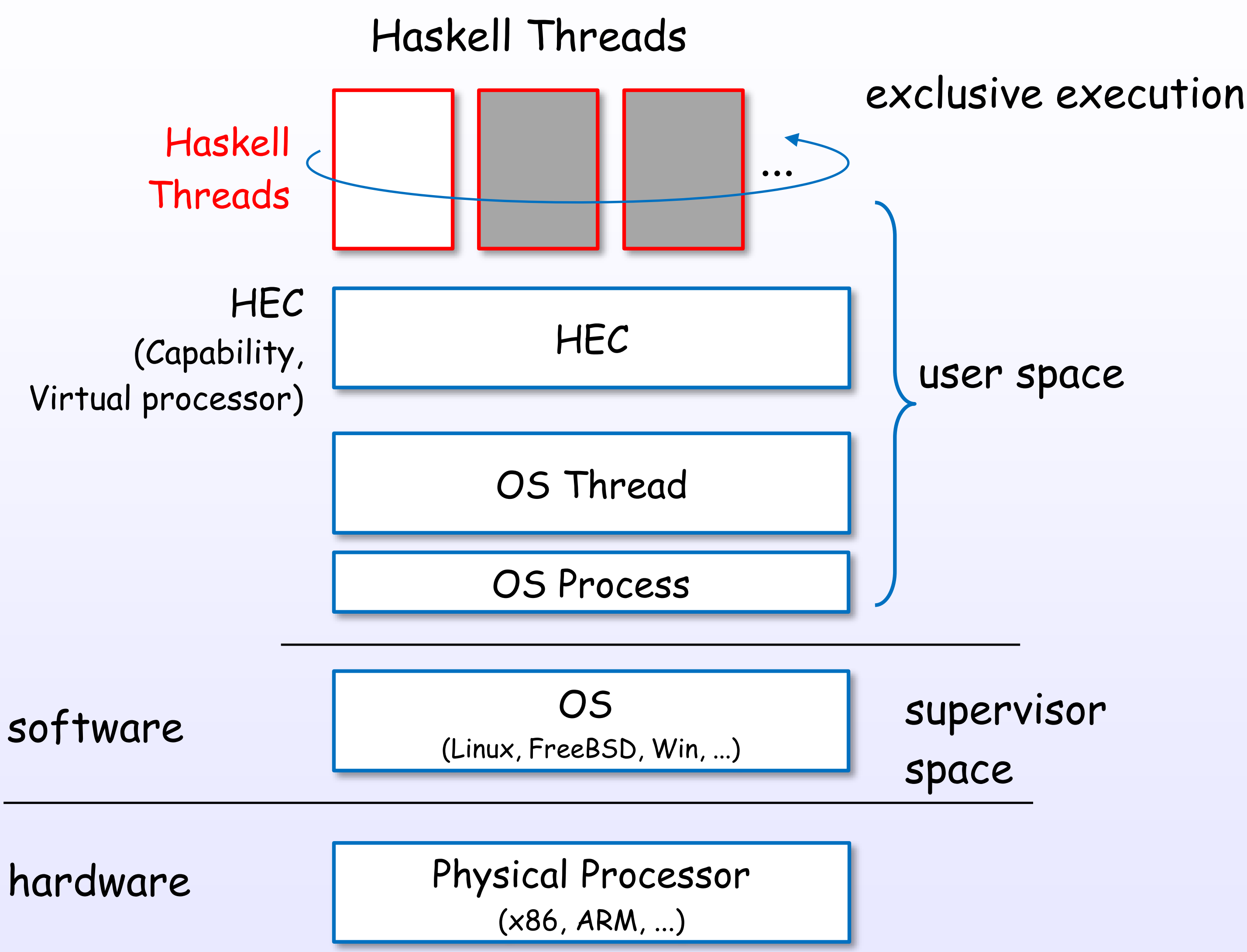
Calling convention examples



References : [24], [C11], [C10], [C13], [S31], [23]

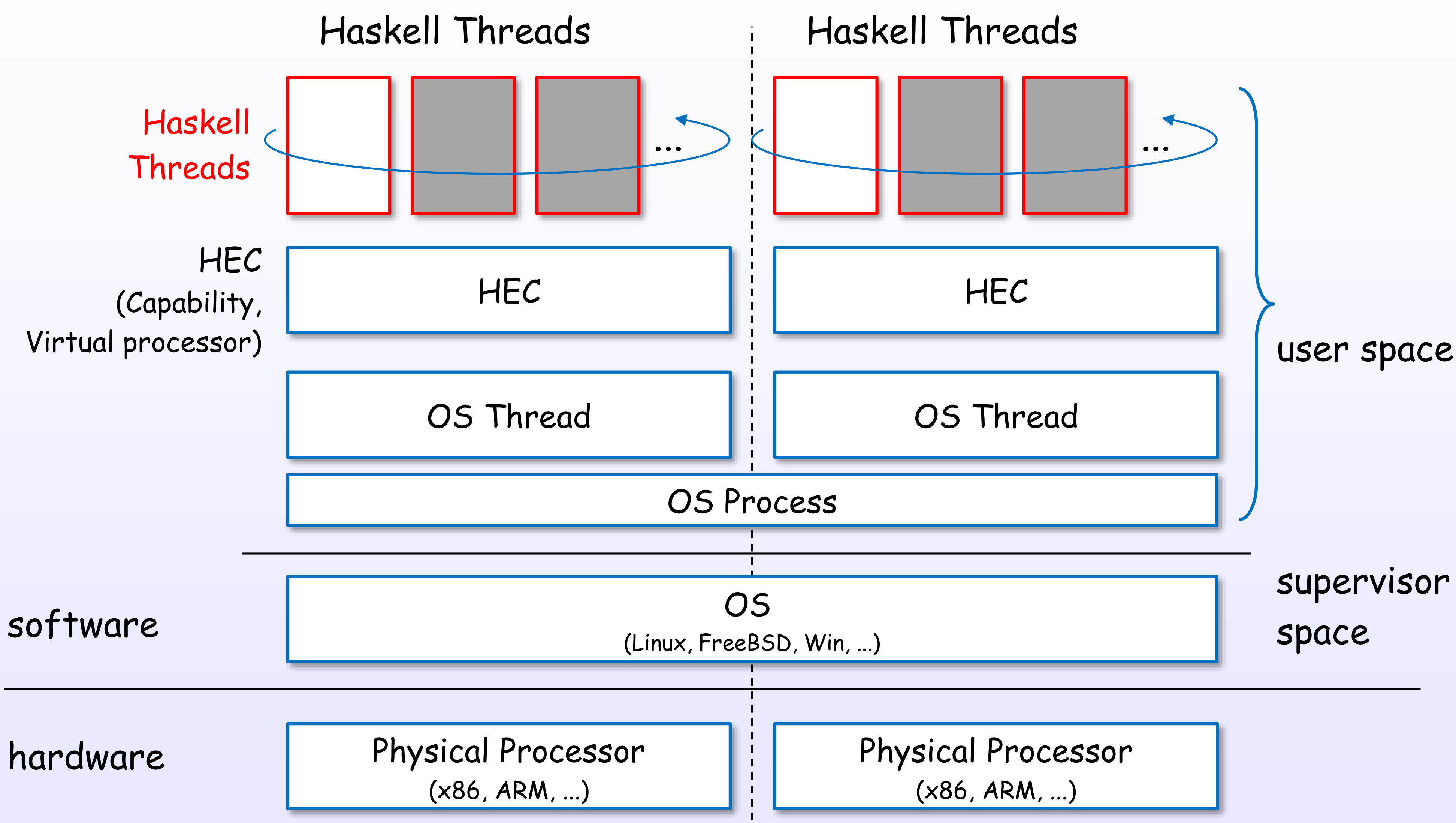
Thread

Thread layer (single core)



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

Thread layer (multi core)

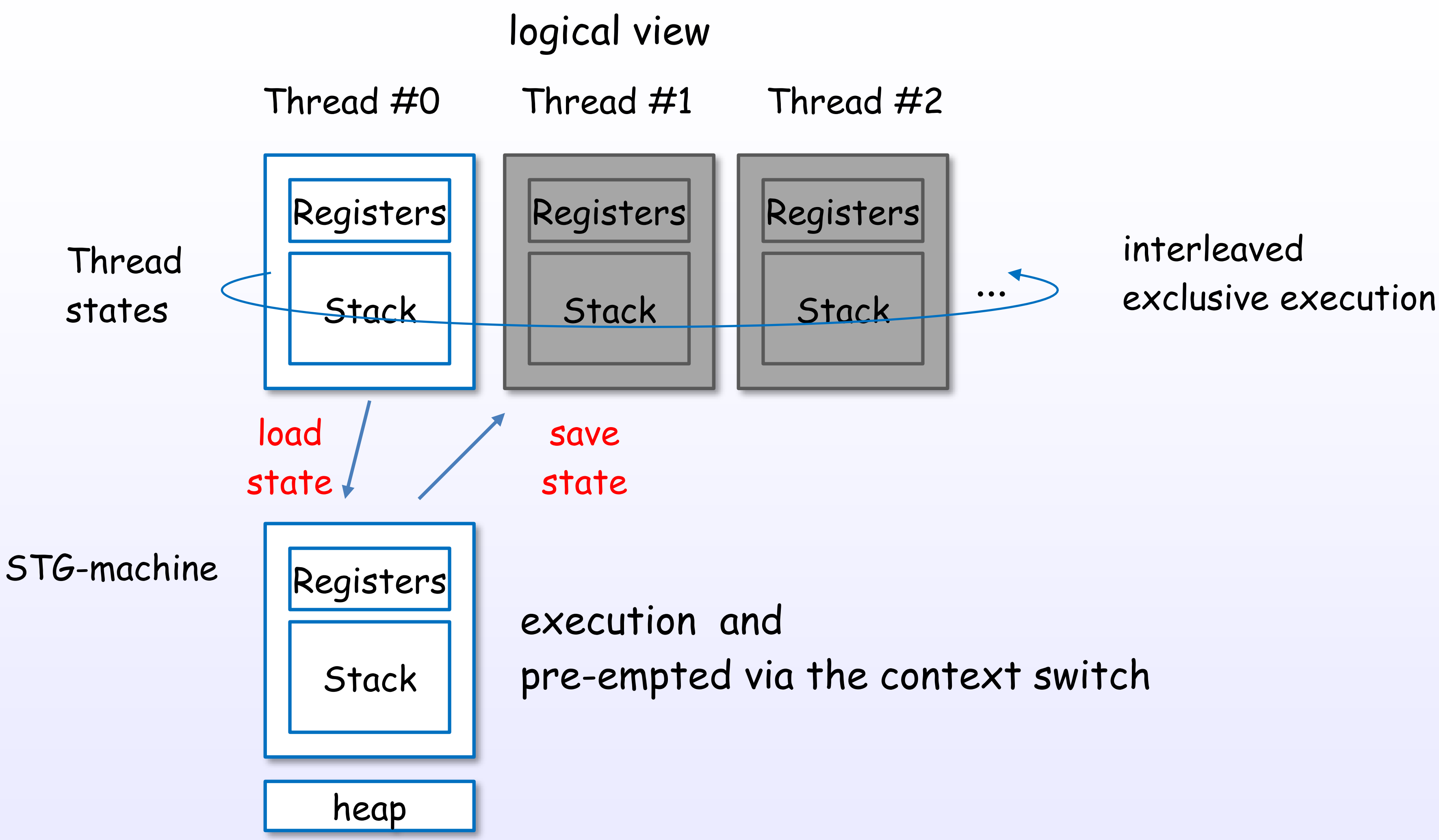


*Threaded option case (ghc -threaded)

References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

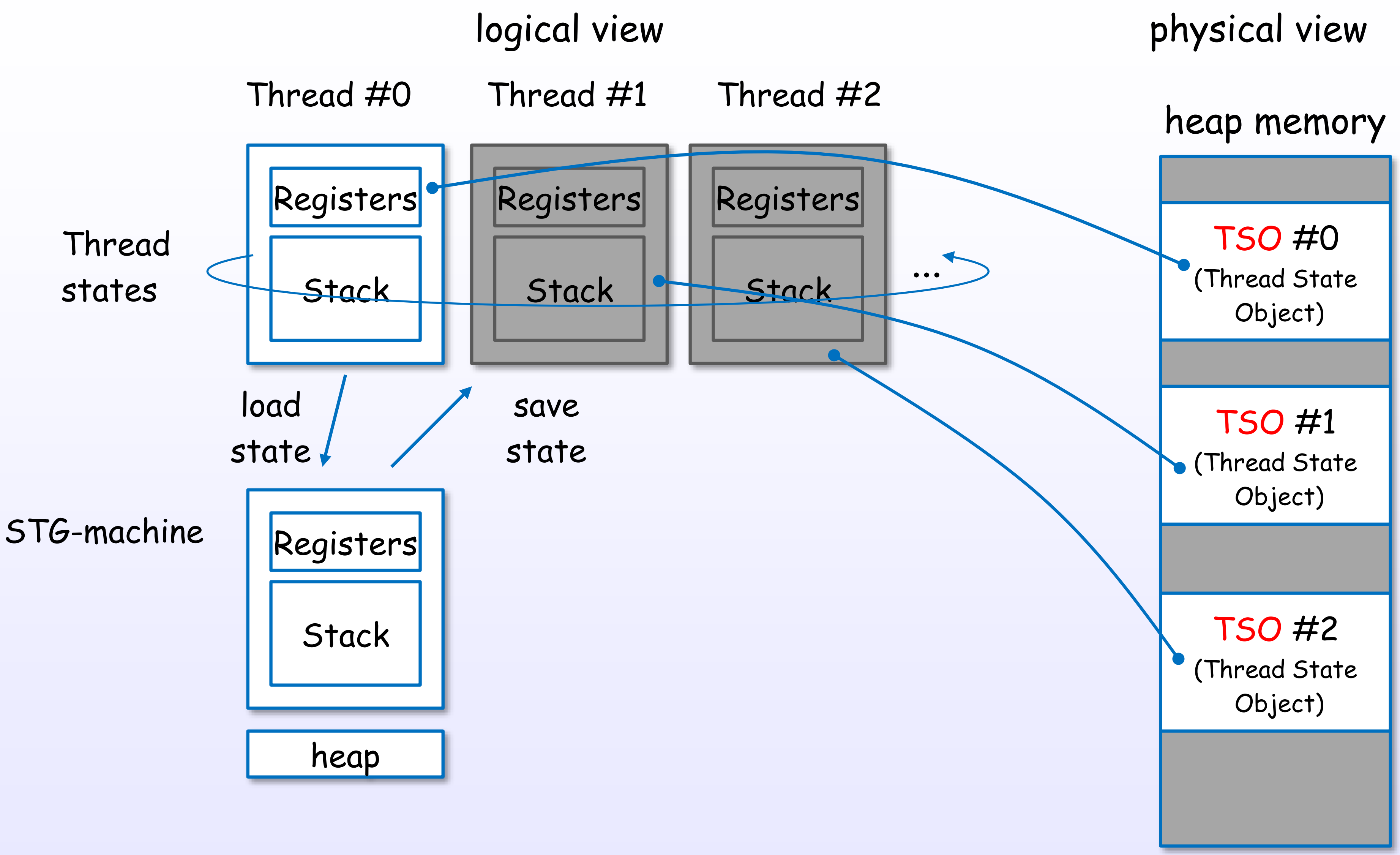
Thread context switch

Threads and context switch



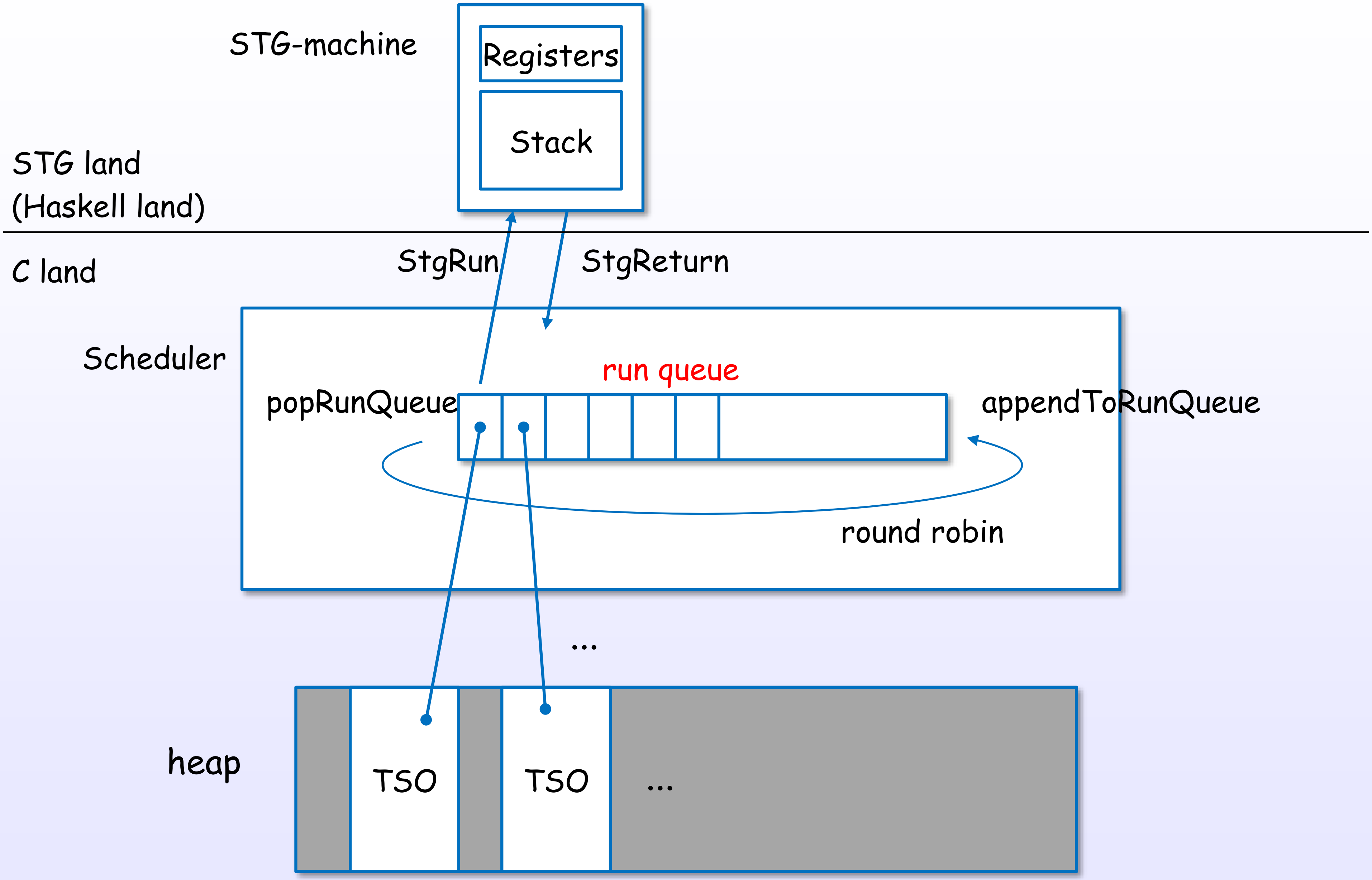
References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

Threads and TSOs



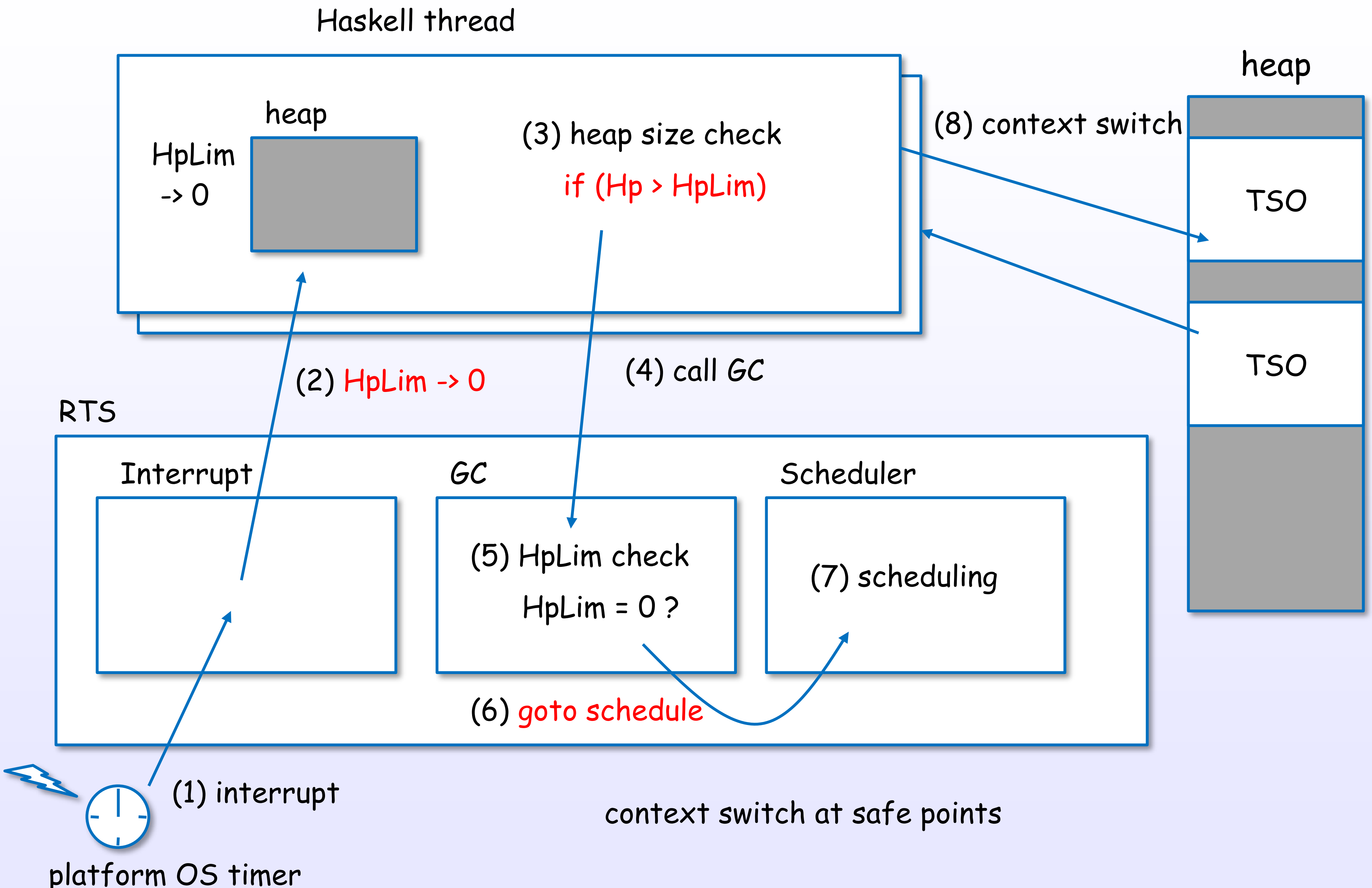
References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

Scheduling by run queue



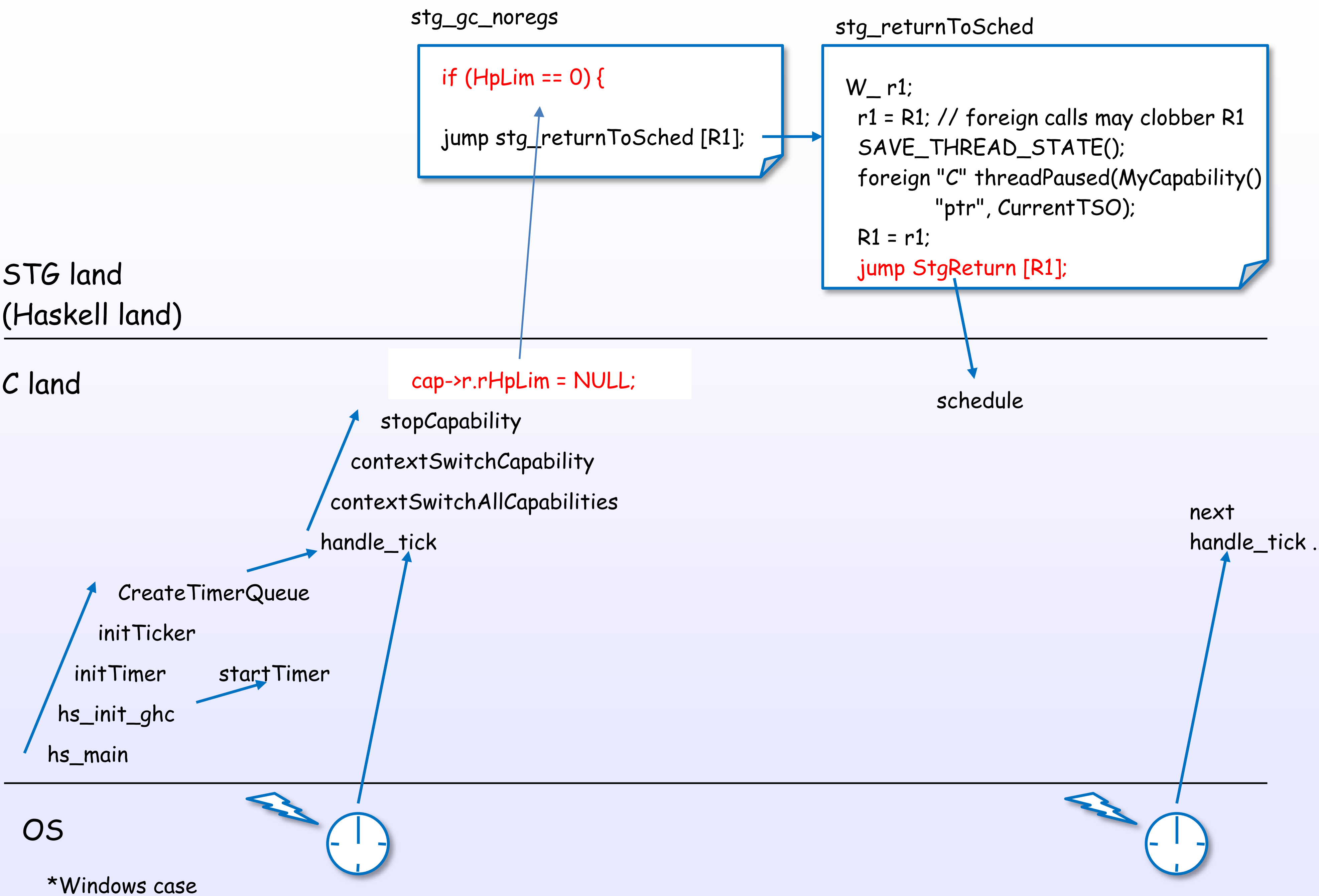
References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14]

Context switch flow



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14], [S24]

Context switch flow (code)



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S21], [S23], [S22], [S14], [S24]

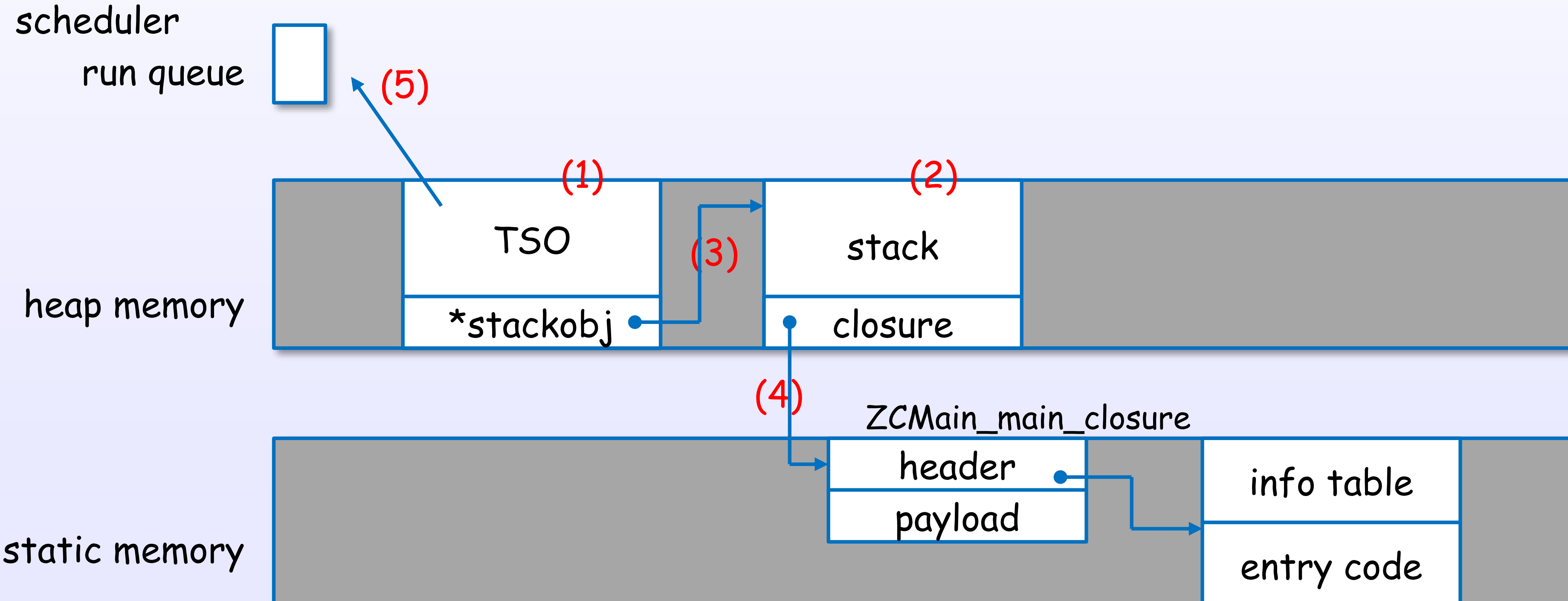
Creating main and sub threads

Create a main thread

Runtime System

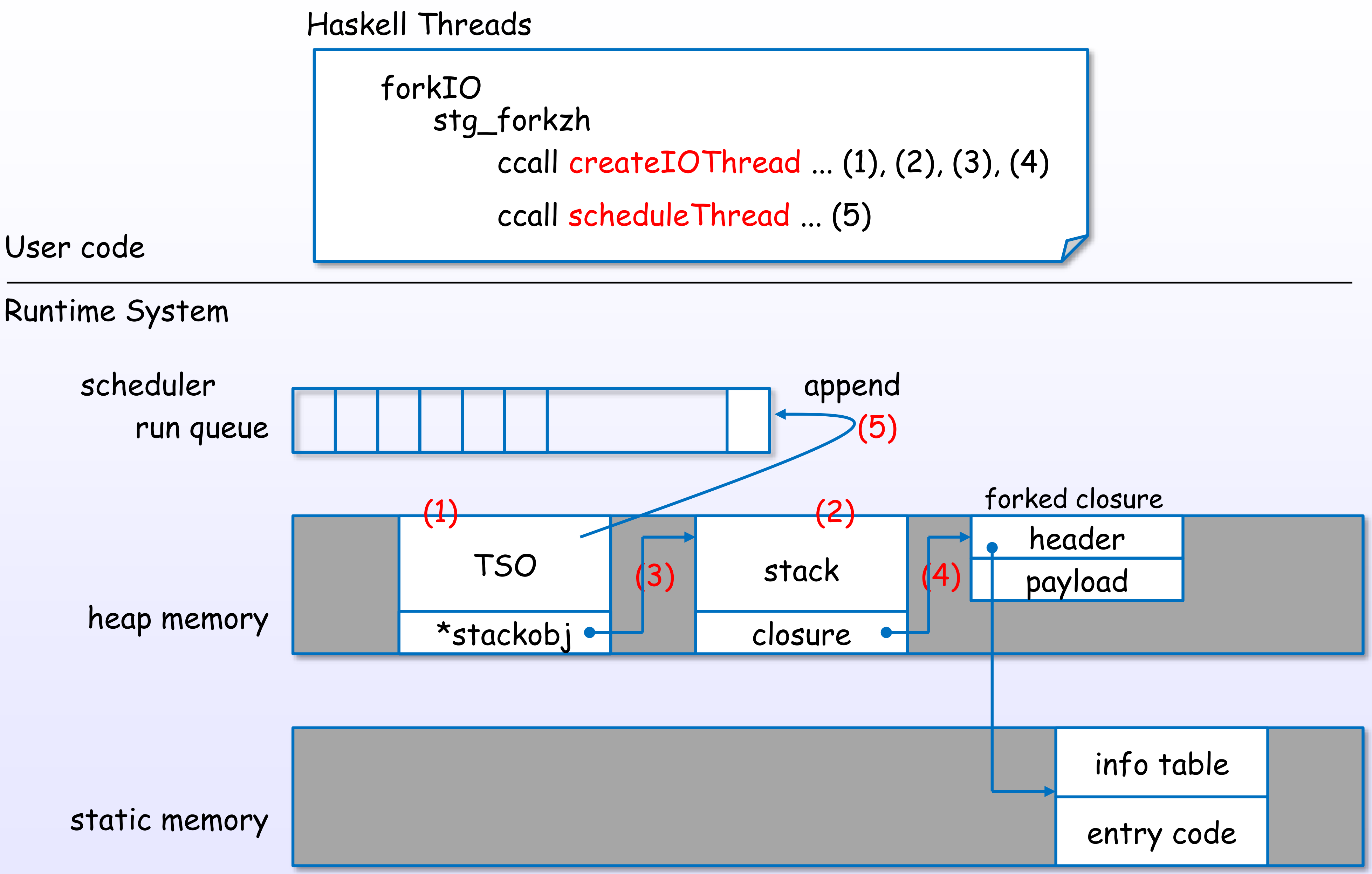
Runtime system bootstrap code [rts/RtsAPI.c]

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



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14], [S24]

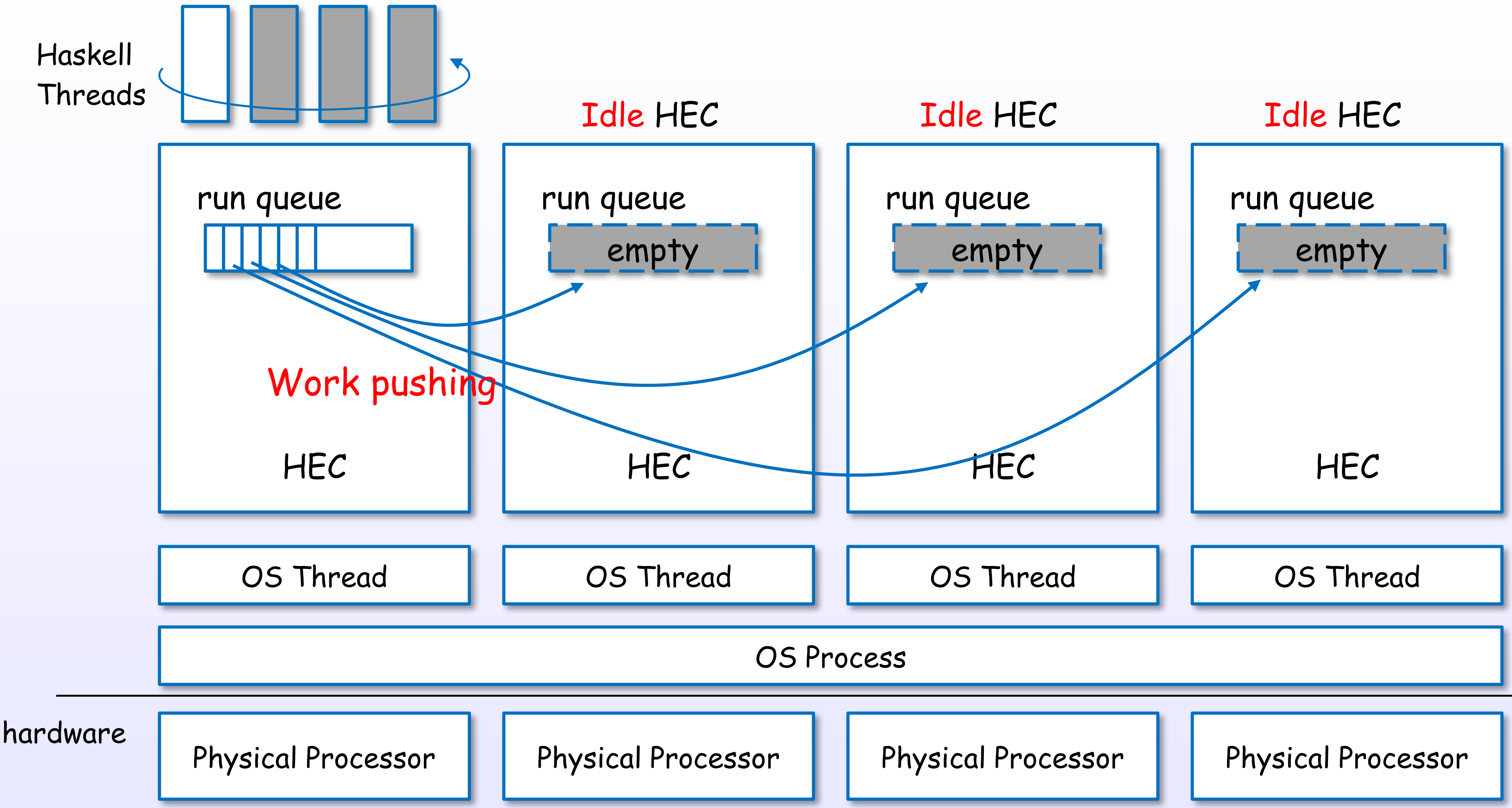
Create a sub thread using forkIO



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14], [S24]

Thread migration

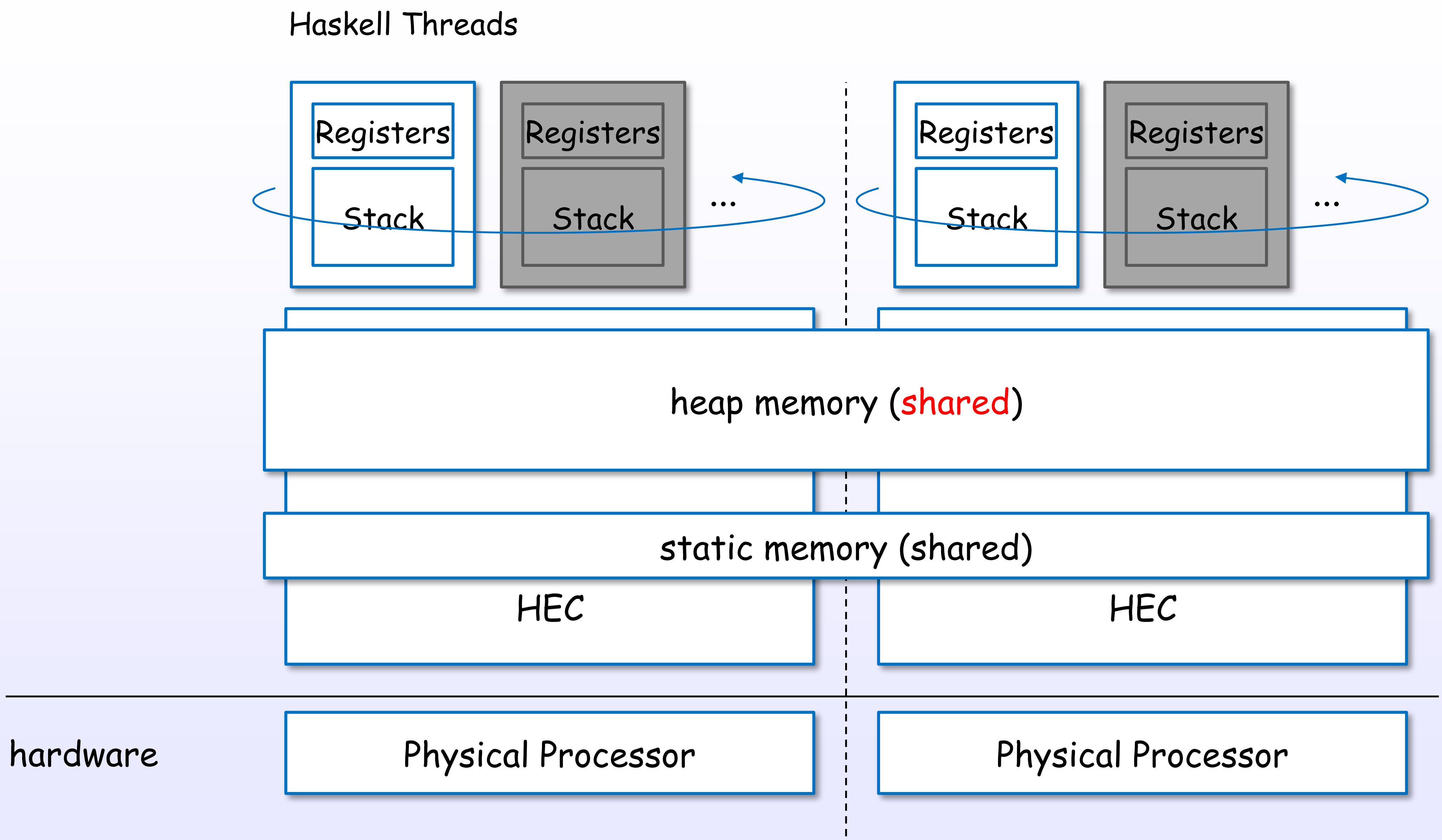
Threads are migrated to idle HECs



References : [5], [8], [9], [14], [C19], [C20], [S17], [S16], [S23], [S24]

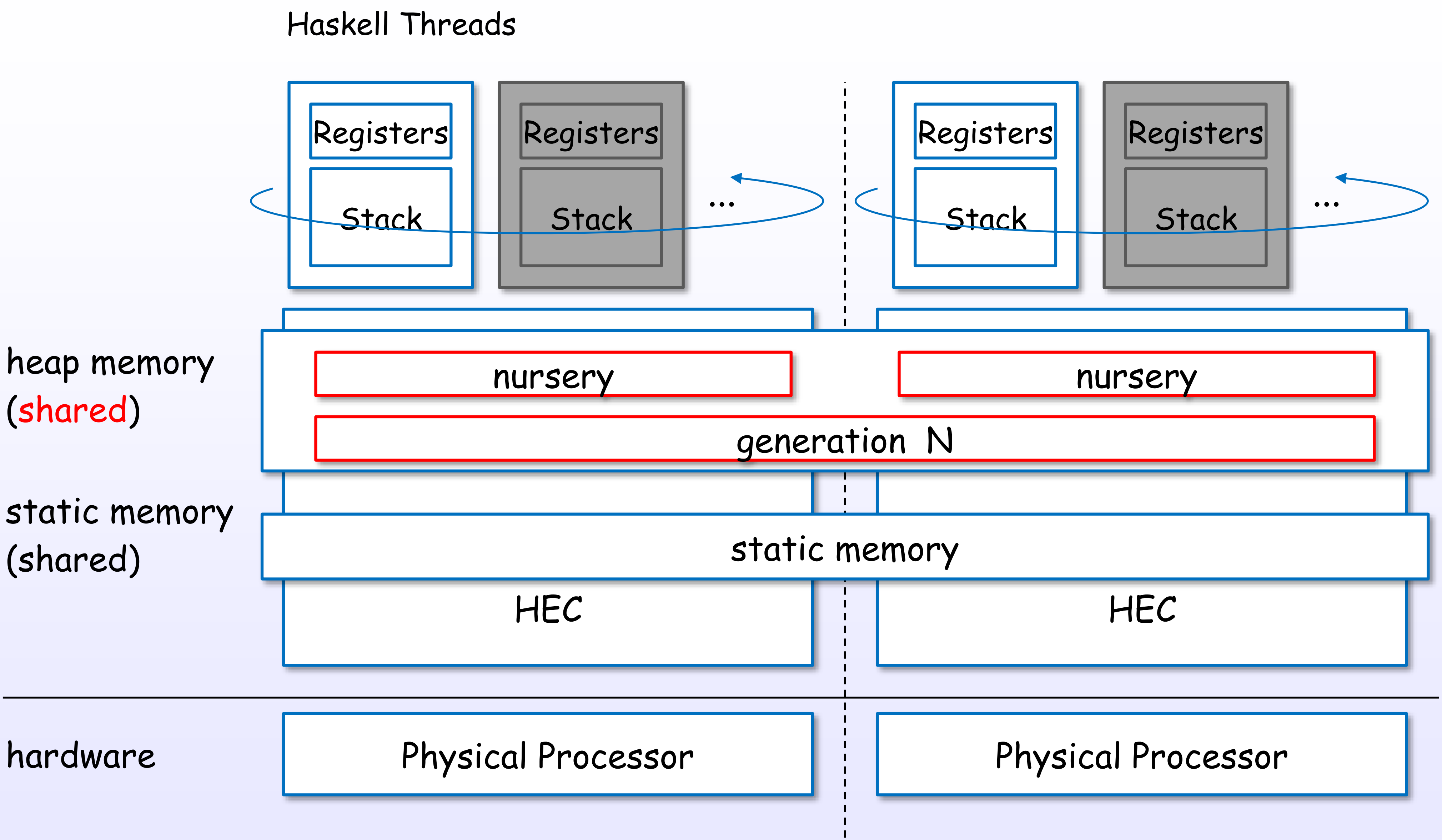
Heap and Threads

Threads share a heap



References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14], [S17], [S16], [S25]

Local allocation area (nursery)

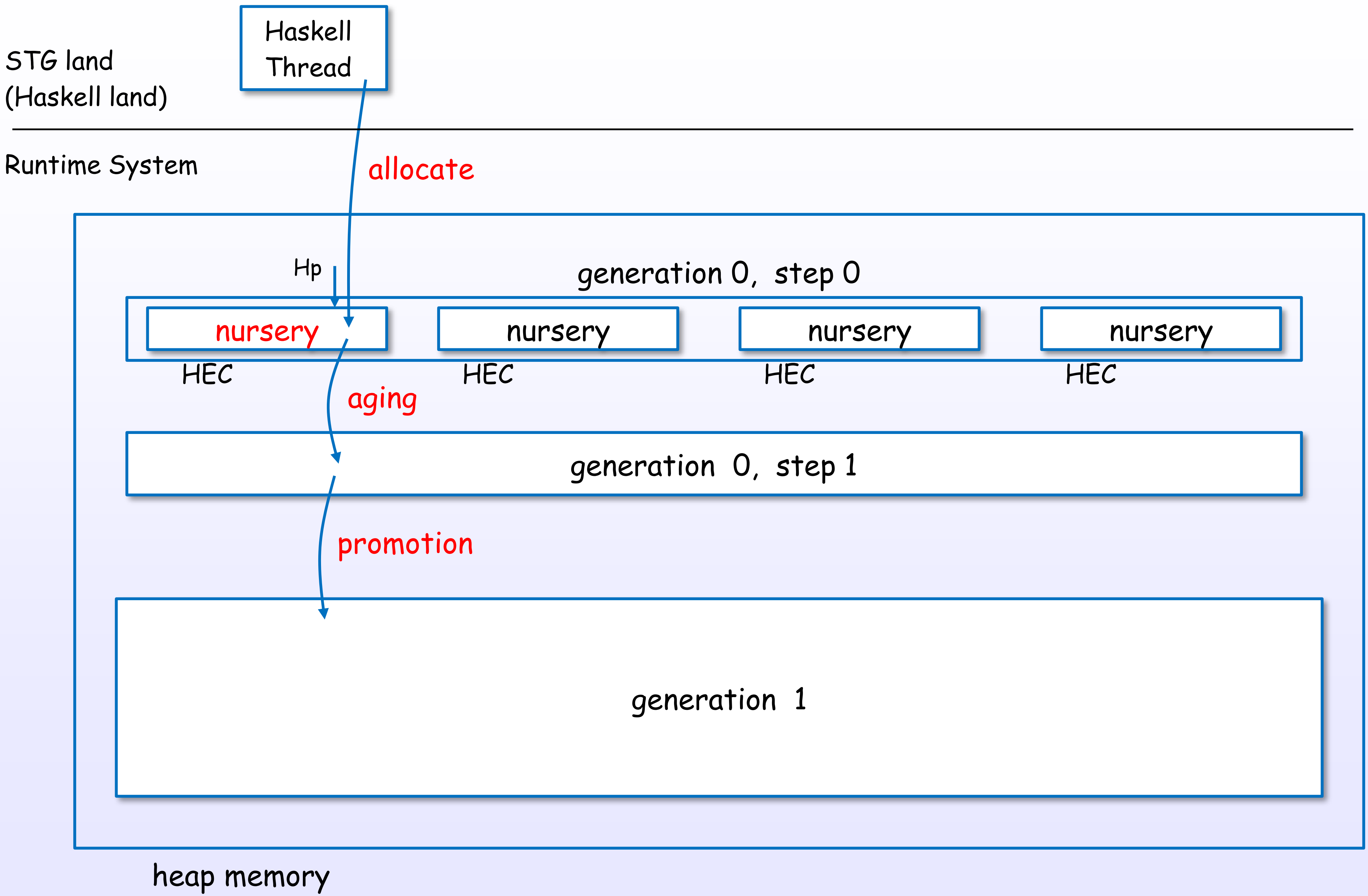


fast access using nursery for each processors

References : [5], [8], [9], [14], [C19], [C13], [19], [S17], [S16], [S23], [S22], [S14], [S17], [S16], [S25]

Threads and GC

GC, nursery, generation, aging, promotion

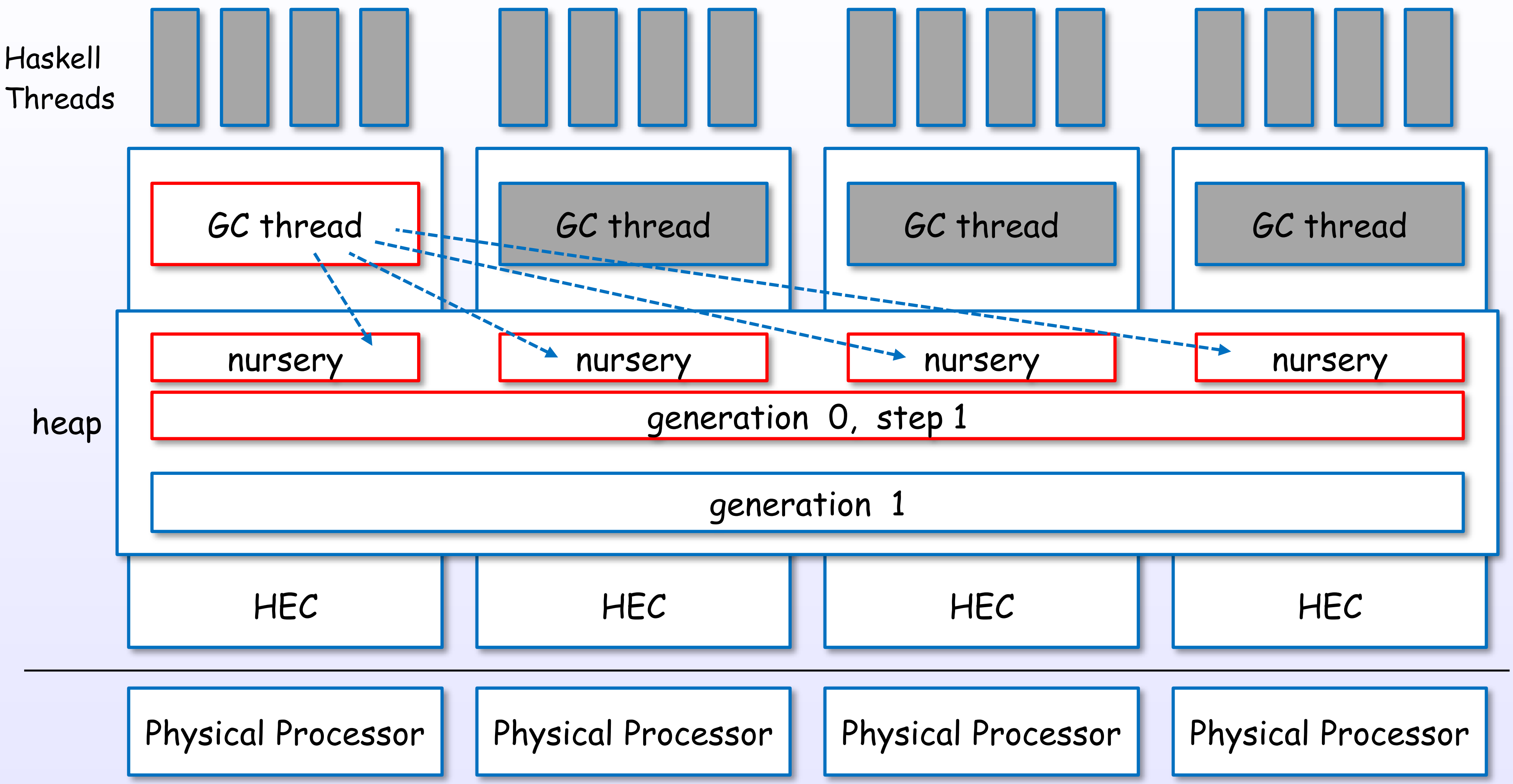


References : [8], [9], [15], [C15], [C13], [S25]

Threads and minor GC

sequential GC for young generation (minor GC)

"stop-the-world" GC

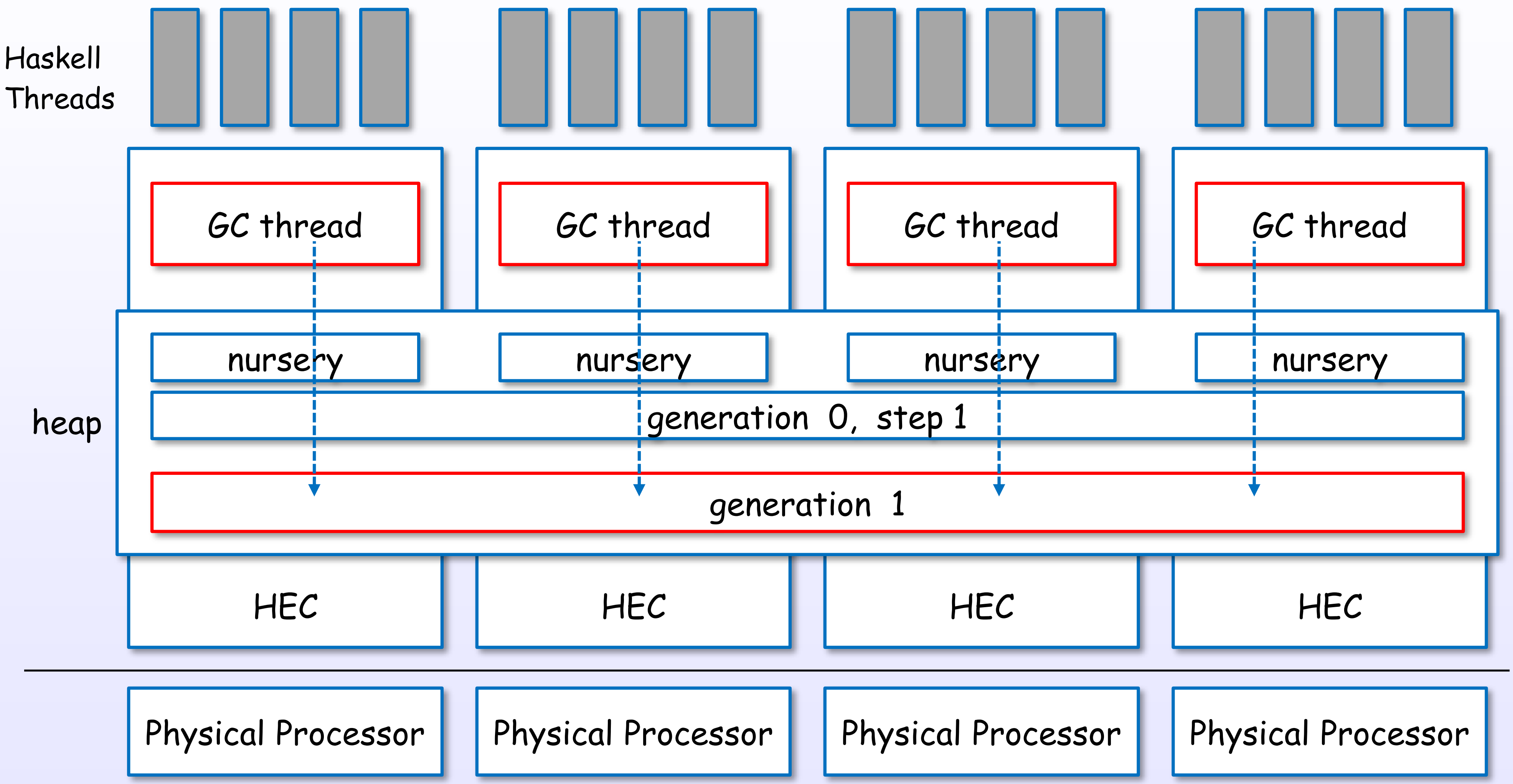


References : [8], [9], [15], [C15], [C13], [S25]

Threads and major GC

parallel GC for oldest generation (major GC)

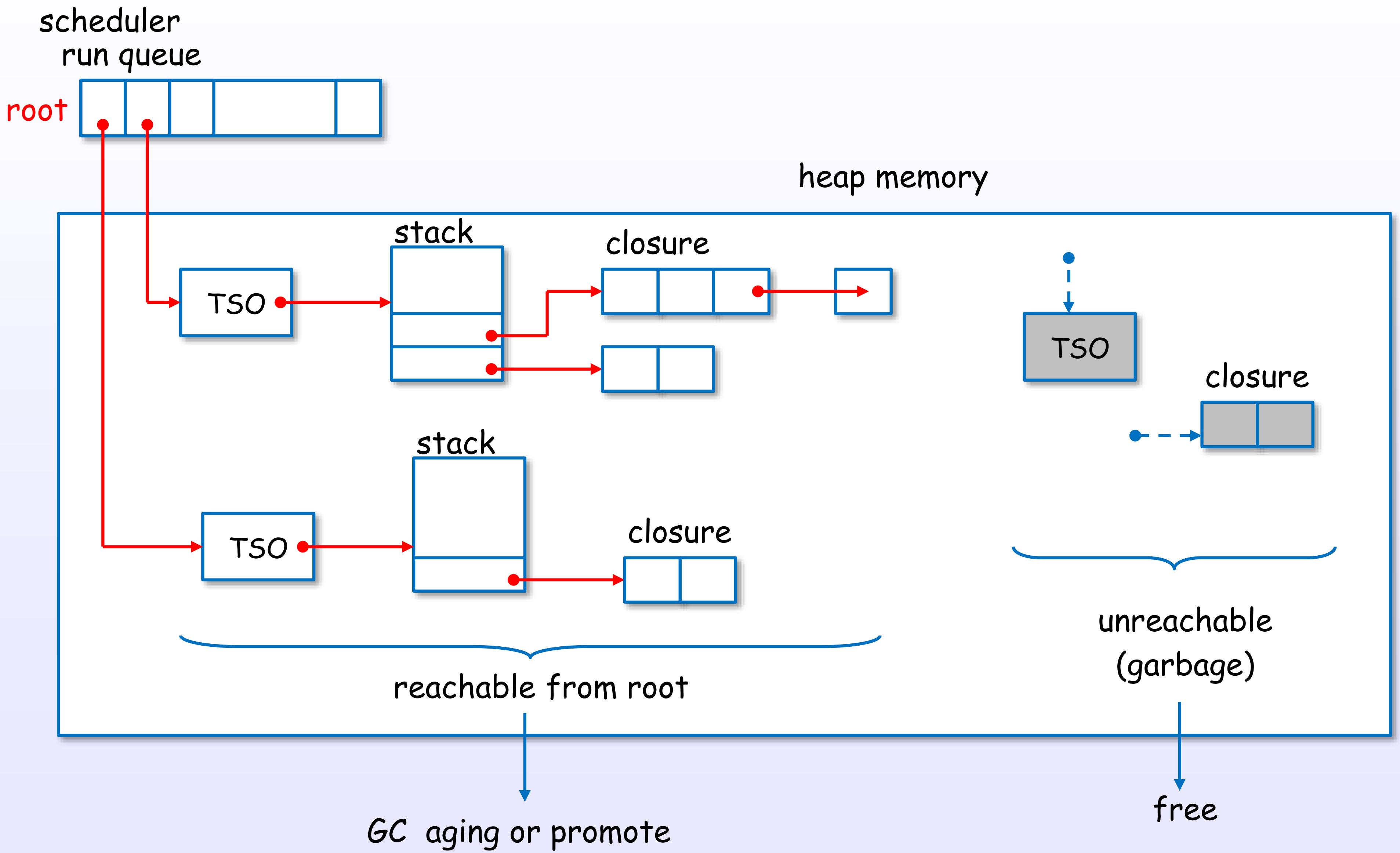
"stop-the-world" GC



References : [8], [9], [15], [C15], [C13], [S25]

GC discover live objects from the root

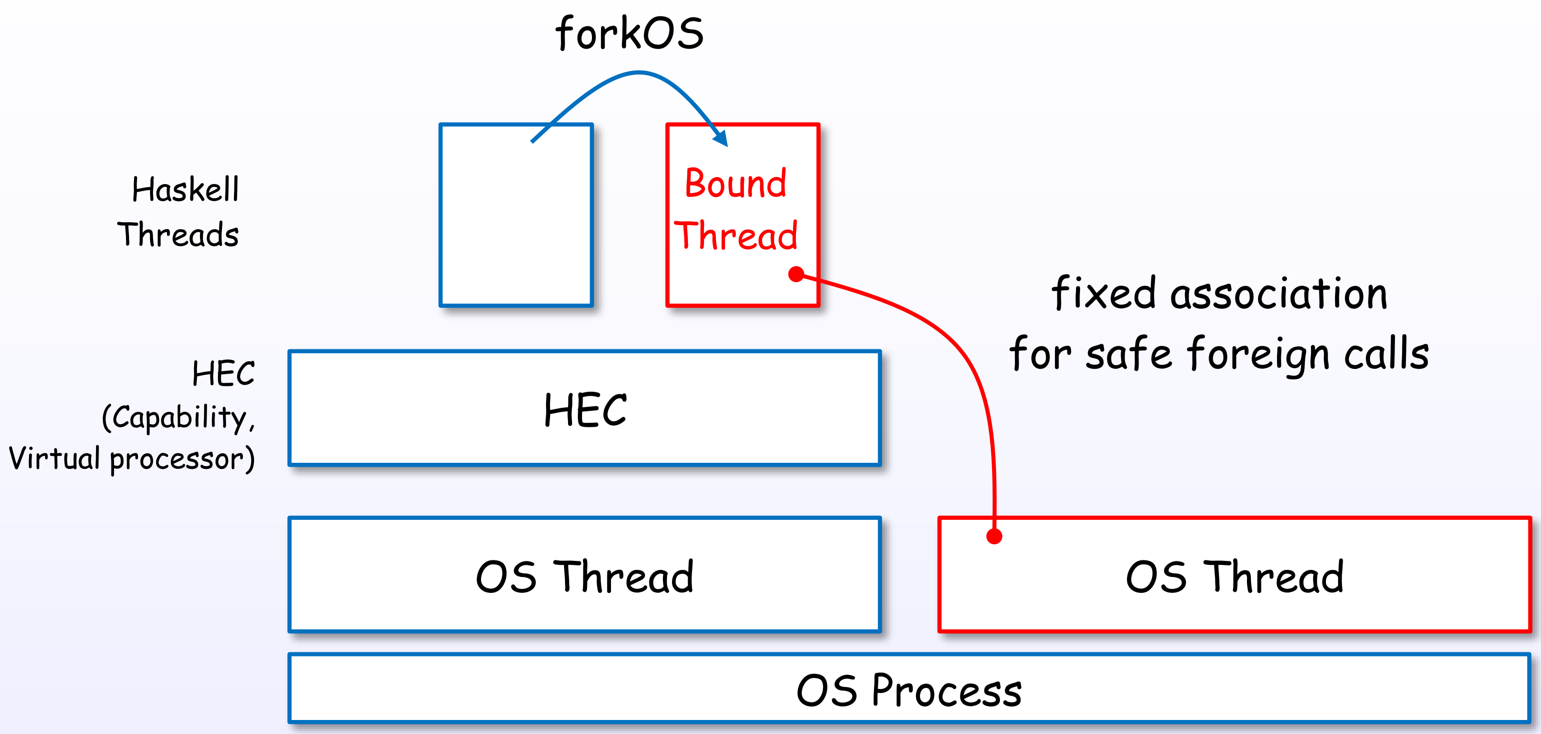
Runtime System



References : [8], [9], [15], [C15], [C13], [S25]

Bound thread

A bound thread has a fixed associated OS Thread

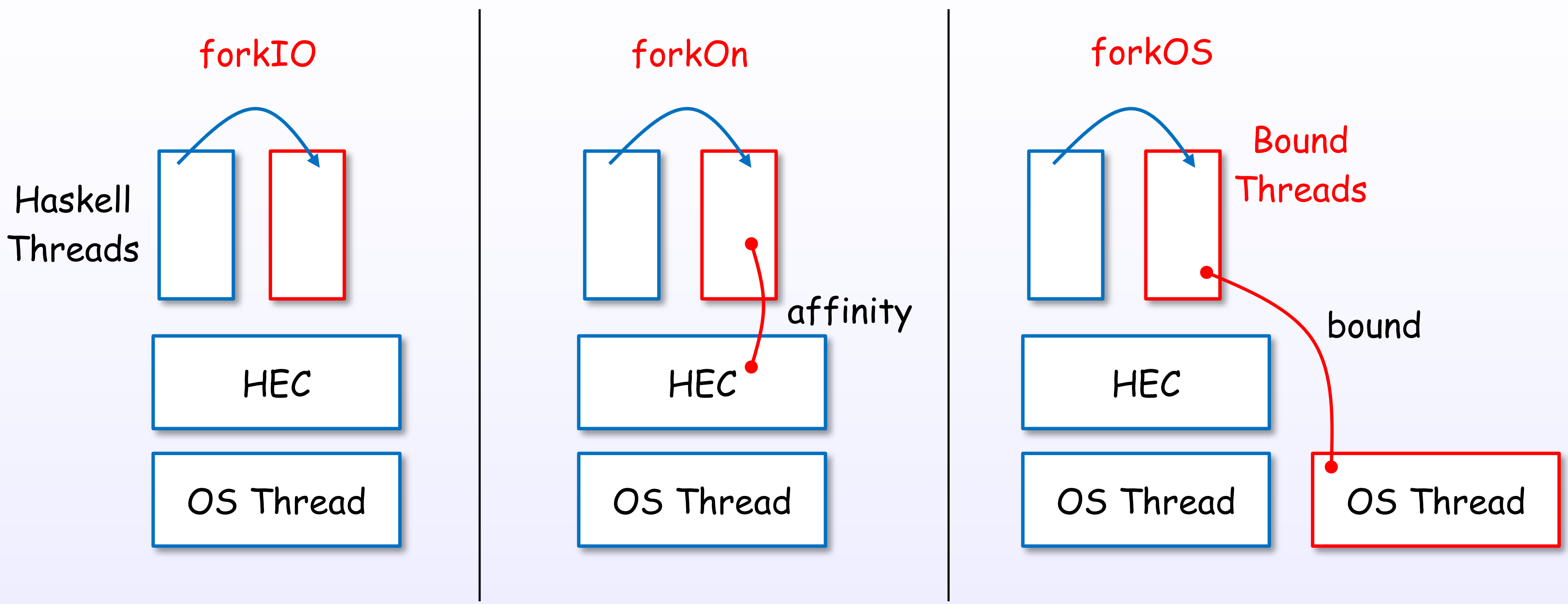


Foreign calls from a bound thread are all made by the same OS thread.
A bound thread is created using forkOS.

The main thread is bound thread.

References : [6], [5], [8], [9], [14], [C19], [19], [S17], [S16], [S23], [S22]

forkIO, forkOn, forkOS



create a haskell unbound thread

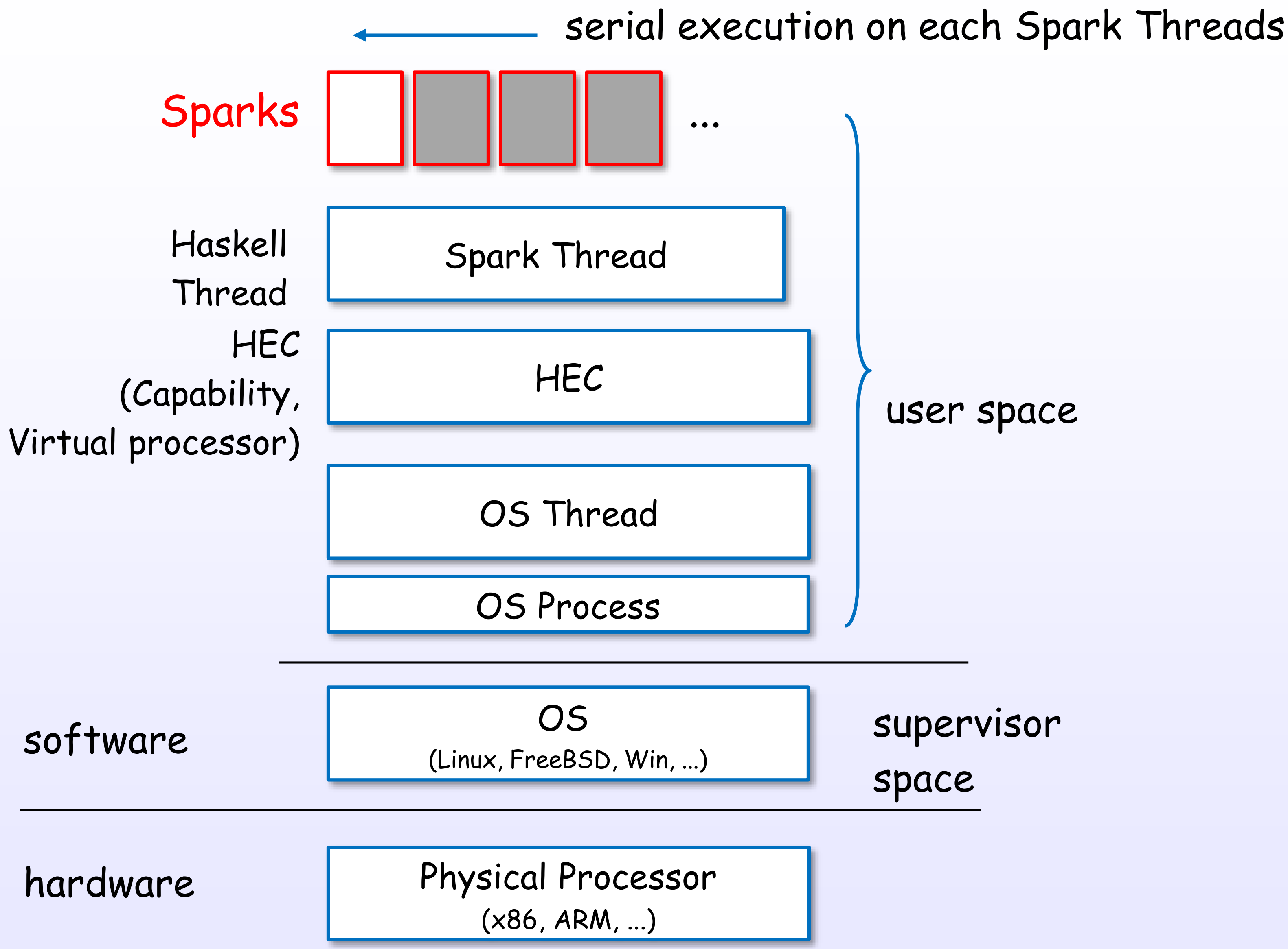
create a haskell unbound thread
on the specified HEC

create a haskell bound thread and an OS thread

References : [6], [5], [8], [9], [14], [C19], [19], [S17], [S16], [S23], [S22]

Spark

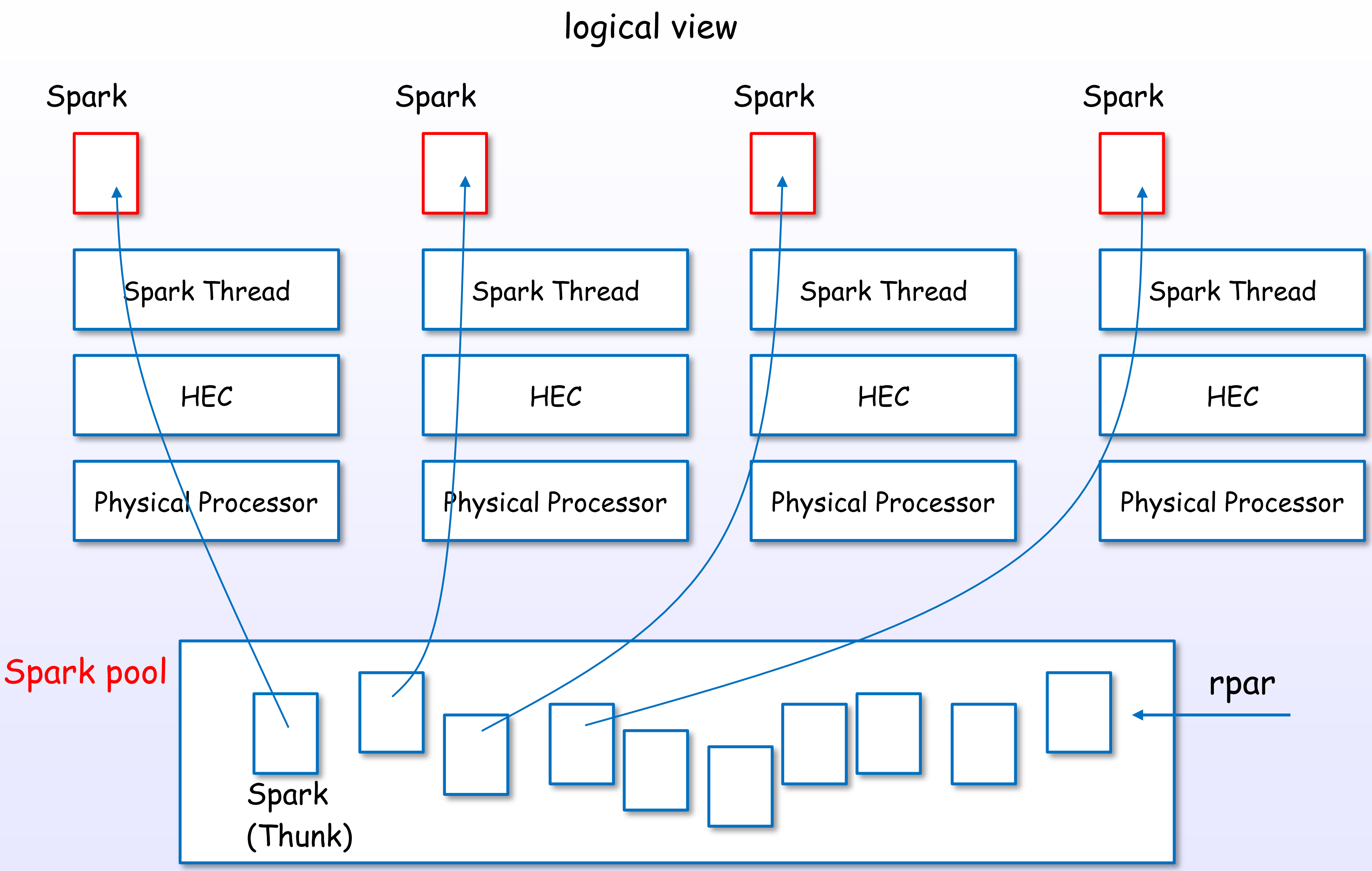
Spark layer



Spark Threads are generated on idle HECs.

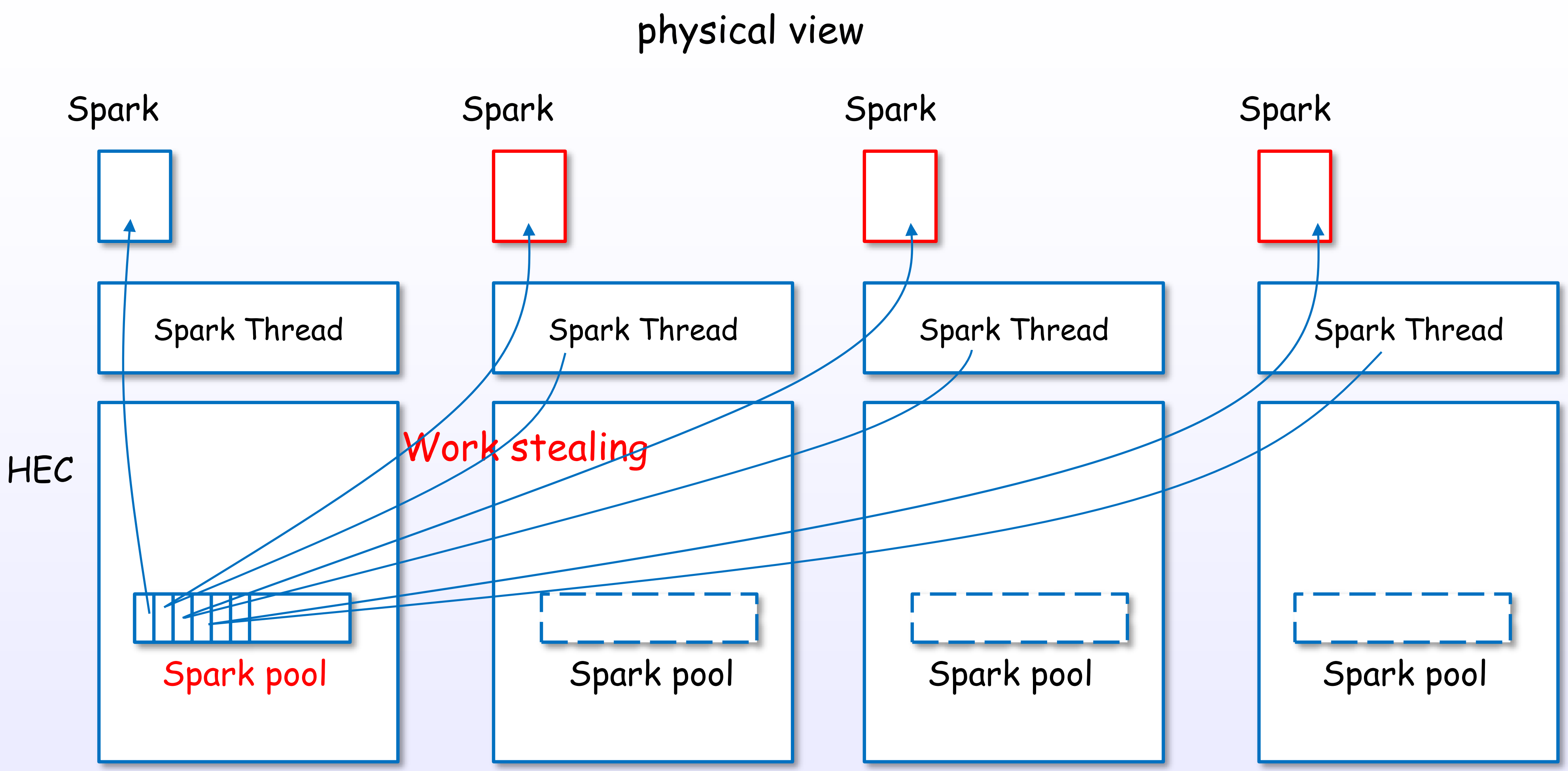
References : [C19], [19], [S17], [S26], [S27], [S34], [S12]

Sparks and Spark pool



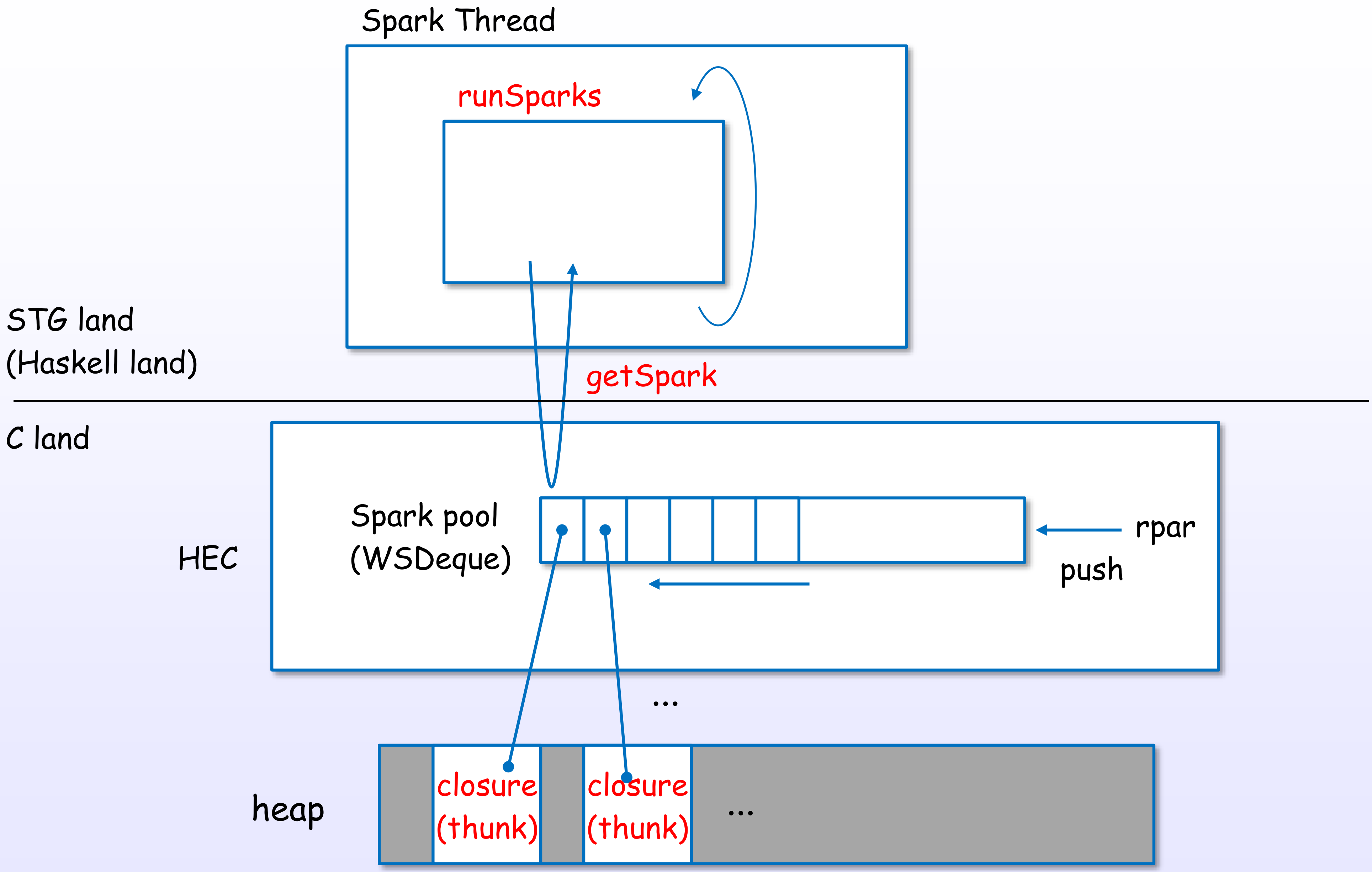
References : [C19], [19], [S17], [S26], [S27], [S34], [S12]

Spark pool and work stealing



References : [C19], [19], [S17], [S26], [S27], [S34], [S12]

Sparks and closures

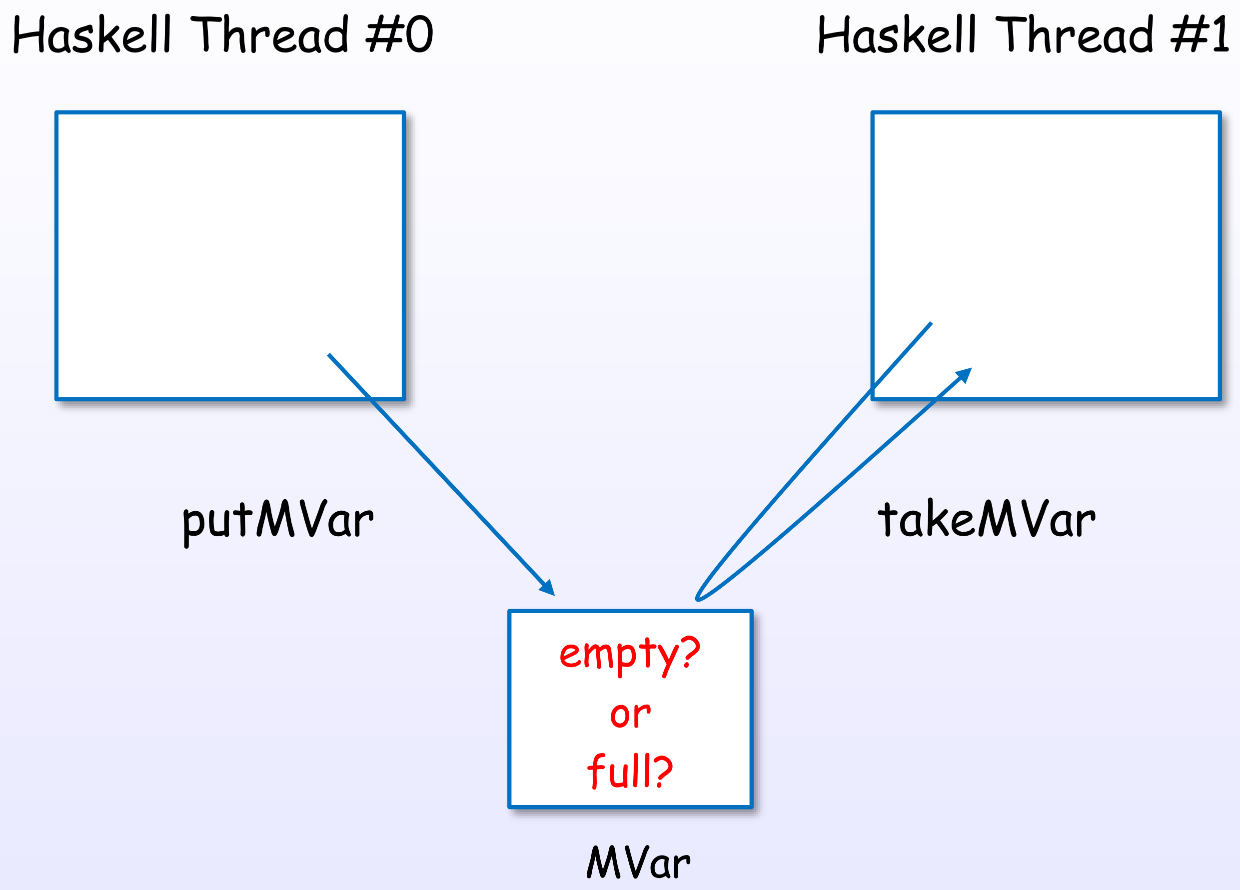


(not TSO objects, but closures. therefore very lightweight)

References : [C19], [19], [S17], [S26], [S27], [S34], [S12]

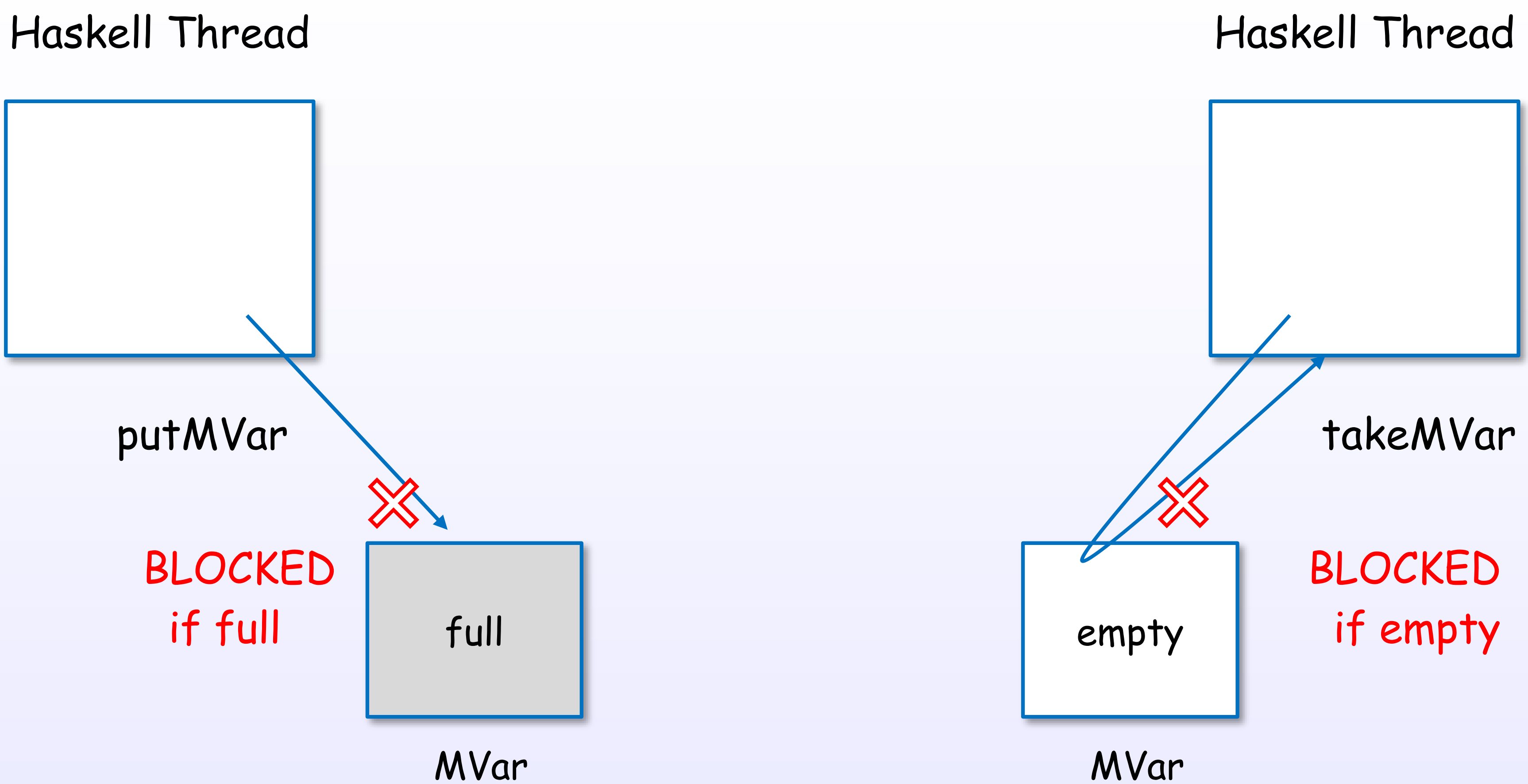
MVar

MVar



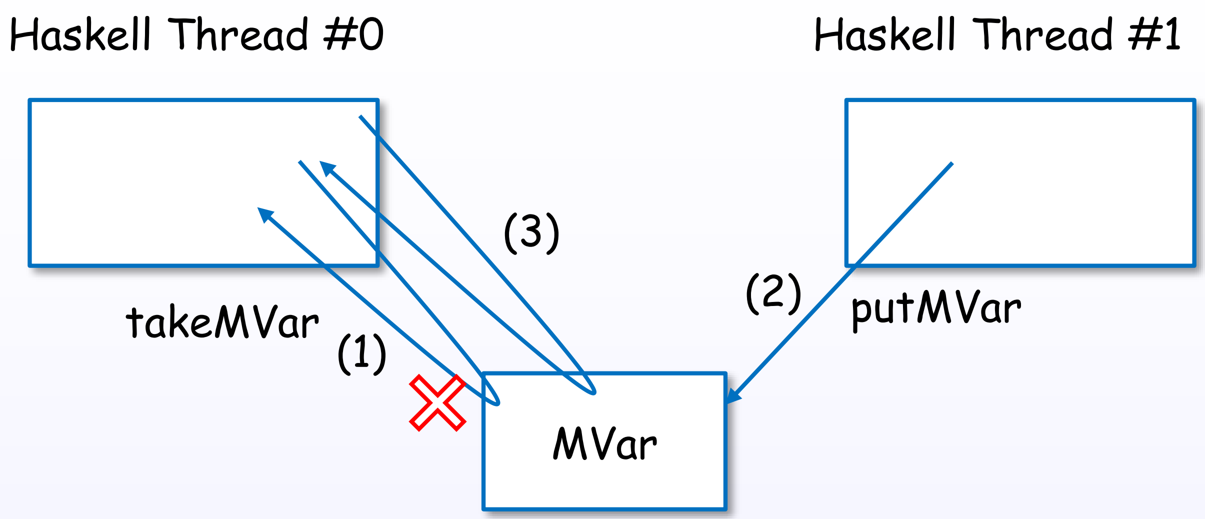
References : [16], [18], [19], [S32], [S12]

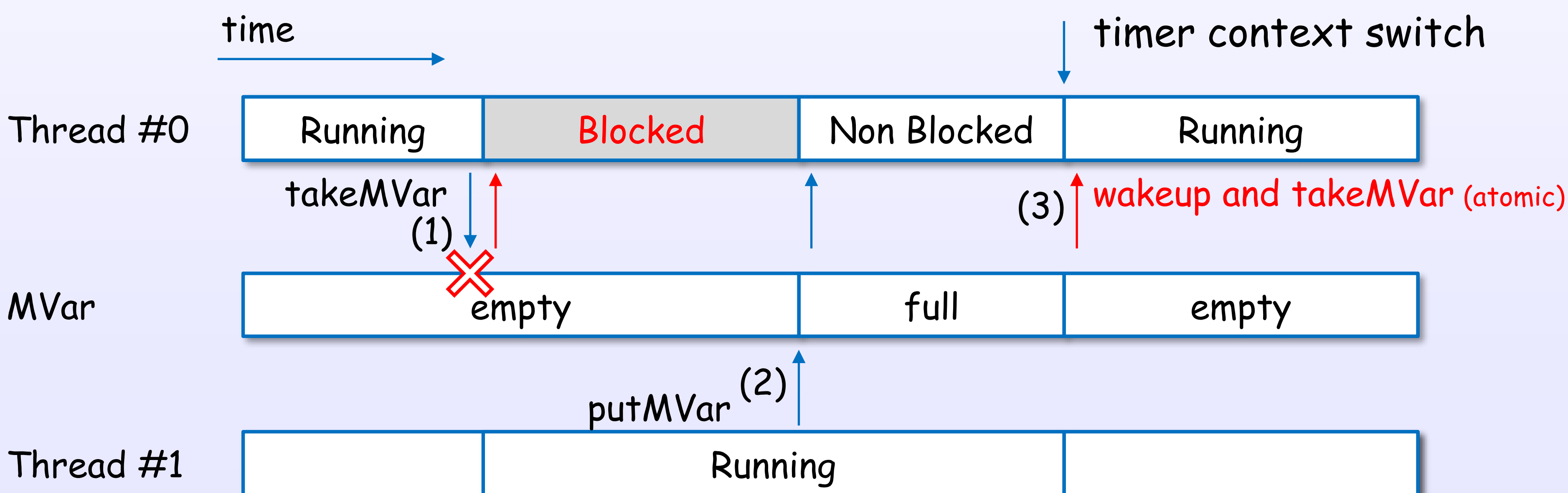
MVar and blocking



References : [16], [18], [19], [S32], [S12]

MVar example





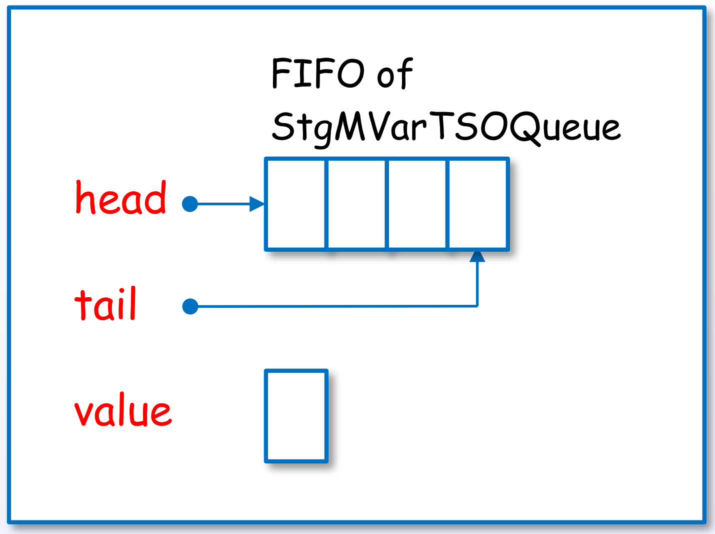
* single core case

References : [16], [18], [19], [S32], [S12]

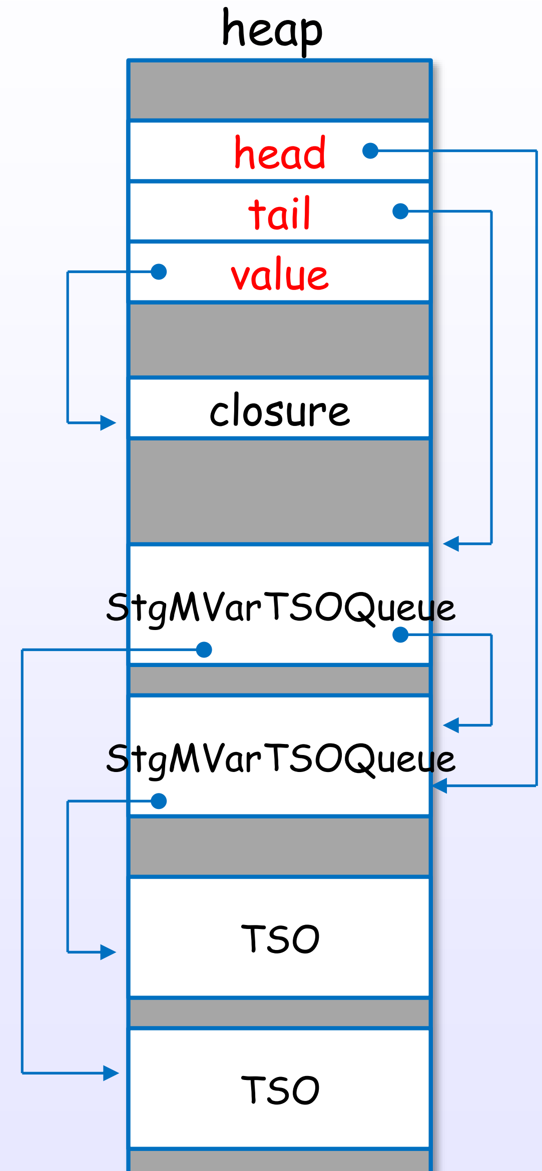
MVar object view

User view

logical MVar object

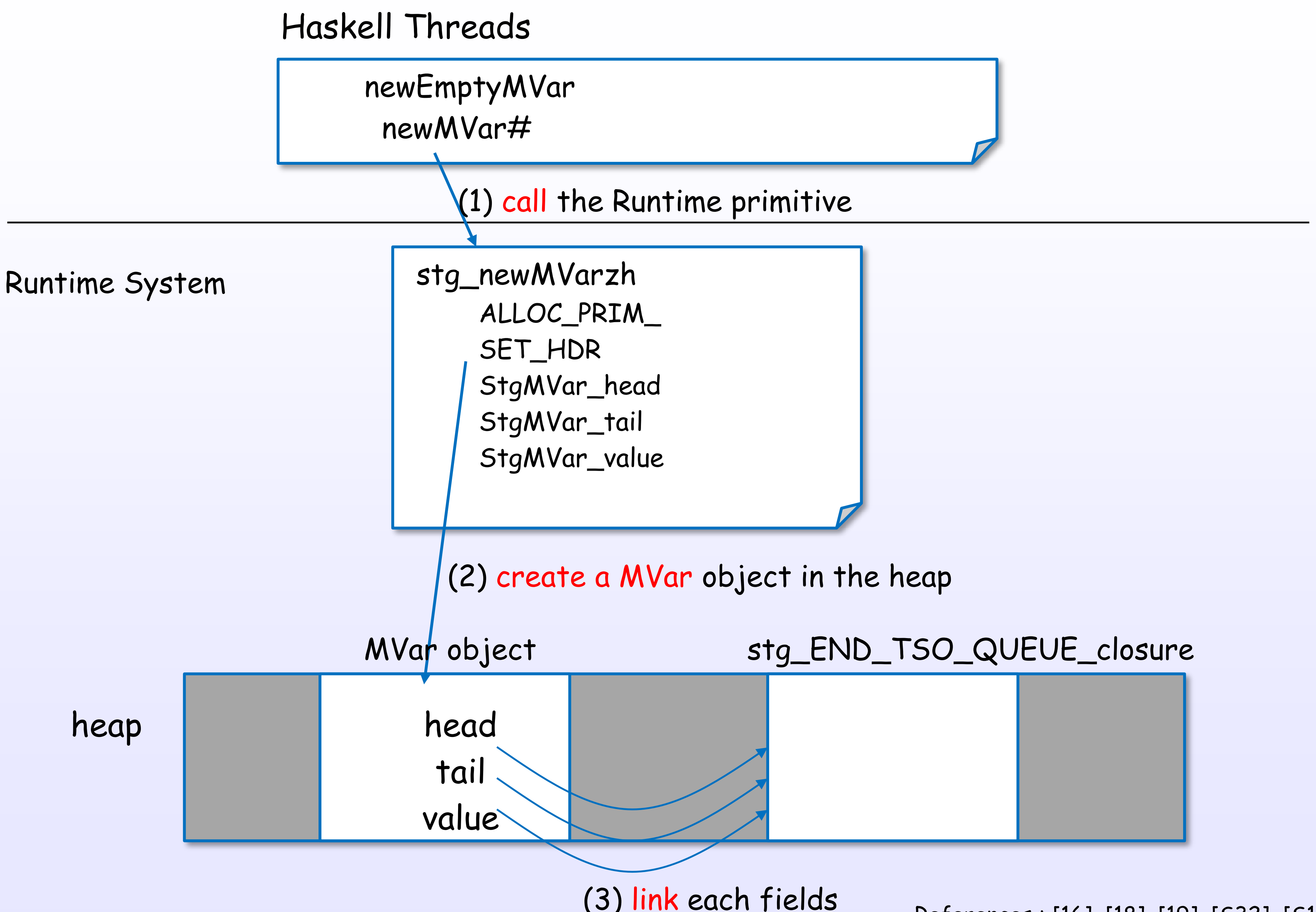


physical MVar object



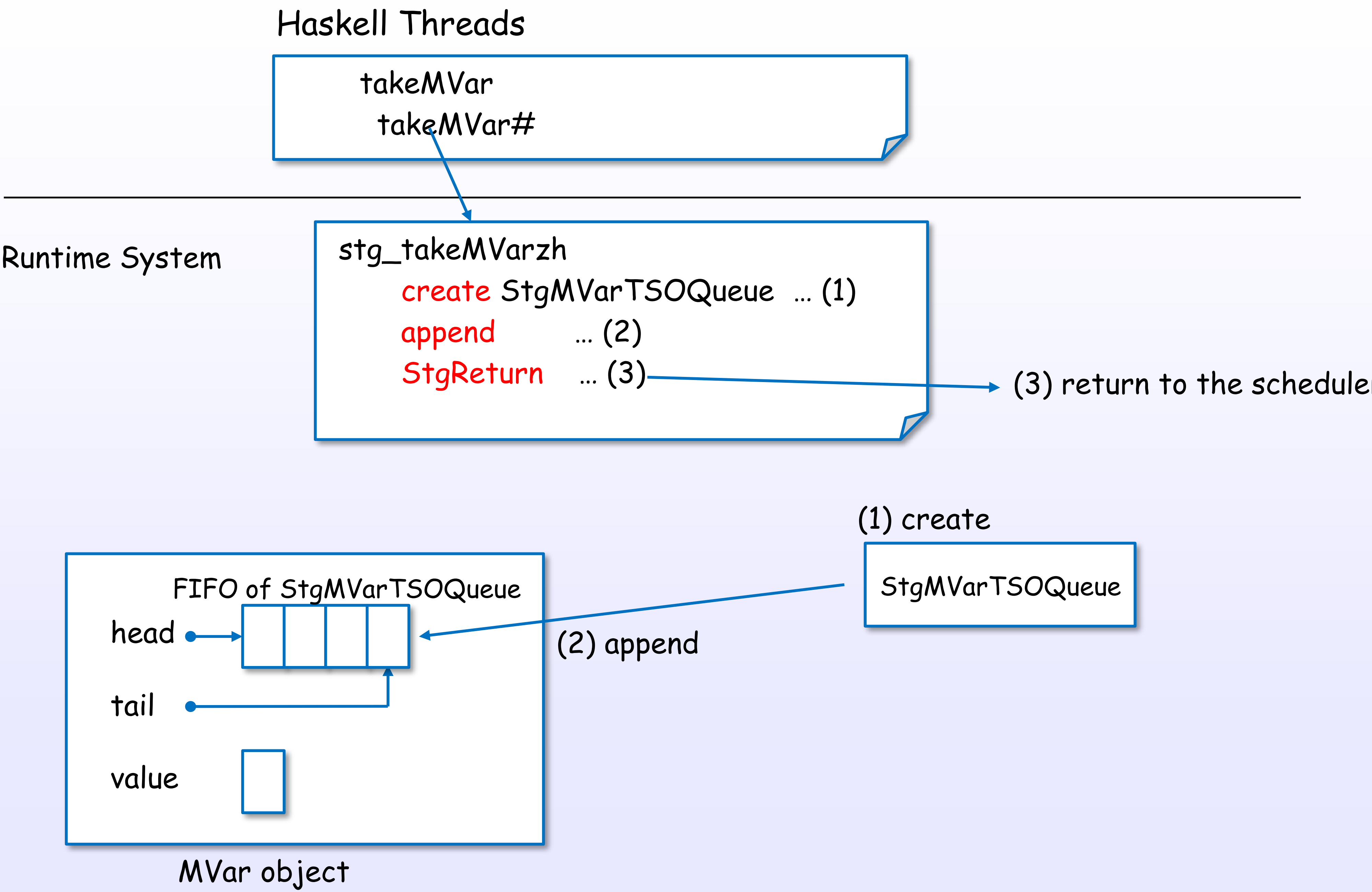
References : [16], [18], [19], [S32], [S12]

newEmptyMVar



References : [16], [18], [19], [S32], [S12]

takeMVar (empty case)



References : [16], [18], [19], [S32], [S12]

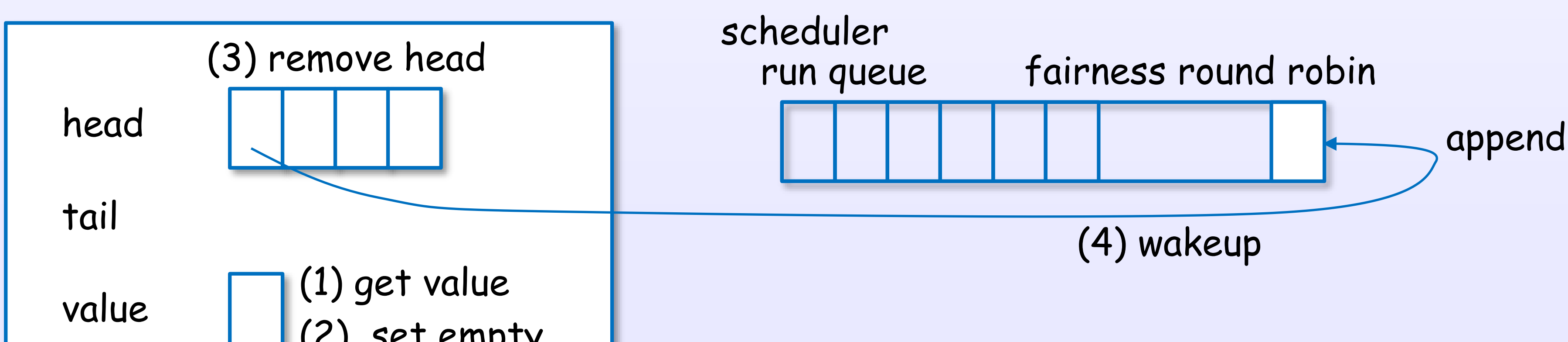
takeMVar (full case)

Haskell Threads

takeMVar
takeMVar#

Runtime System

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



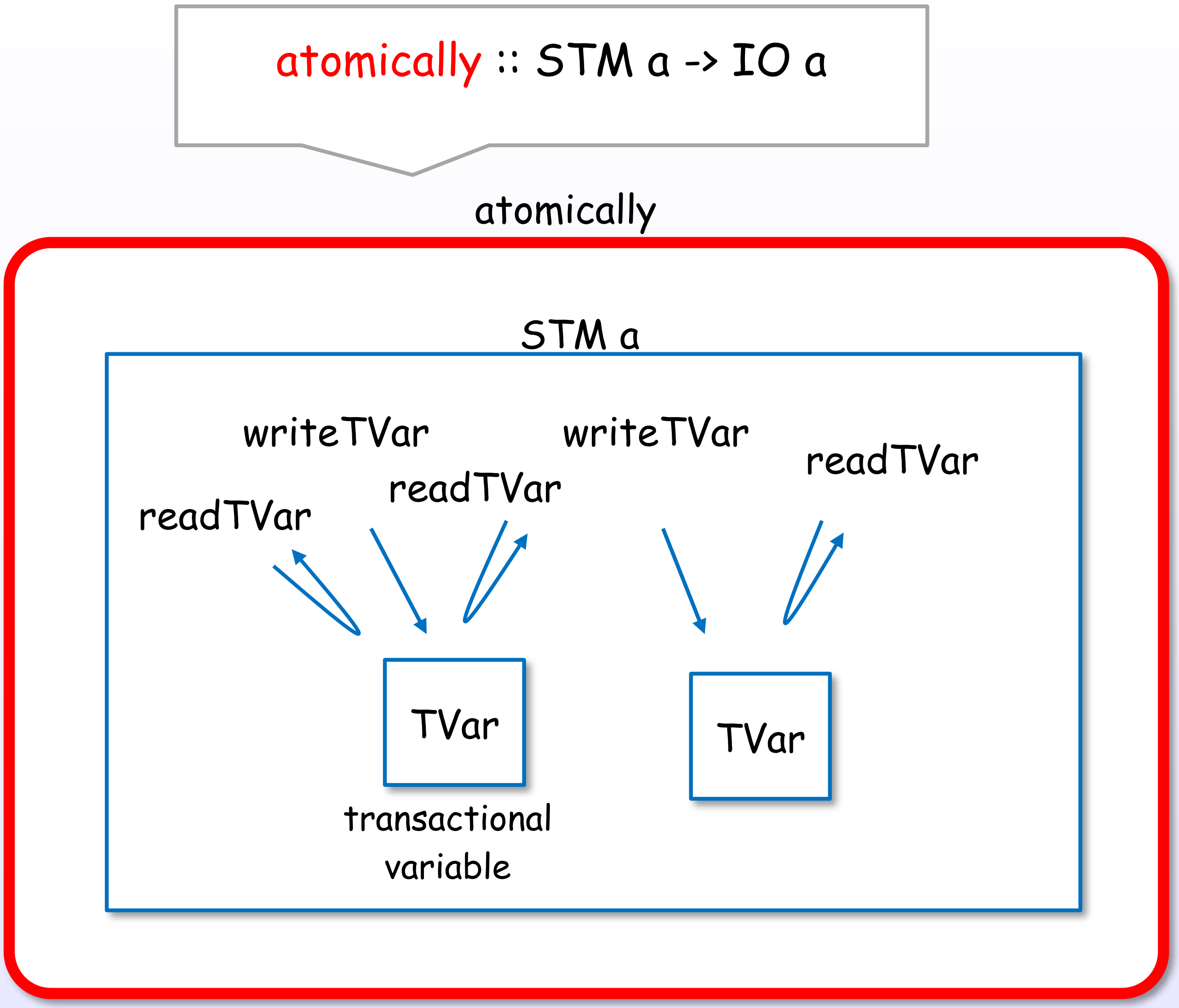
MVar object

Only one of the blocked threads becomes unblocked.

References : [16], [18], [19], [S32], [S12]

Software transactional memory

Create a atomic block using atomically

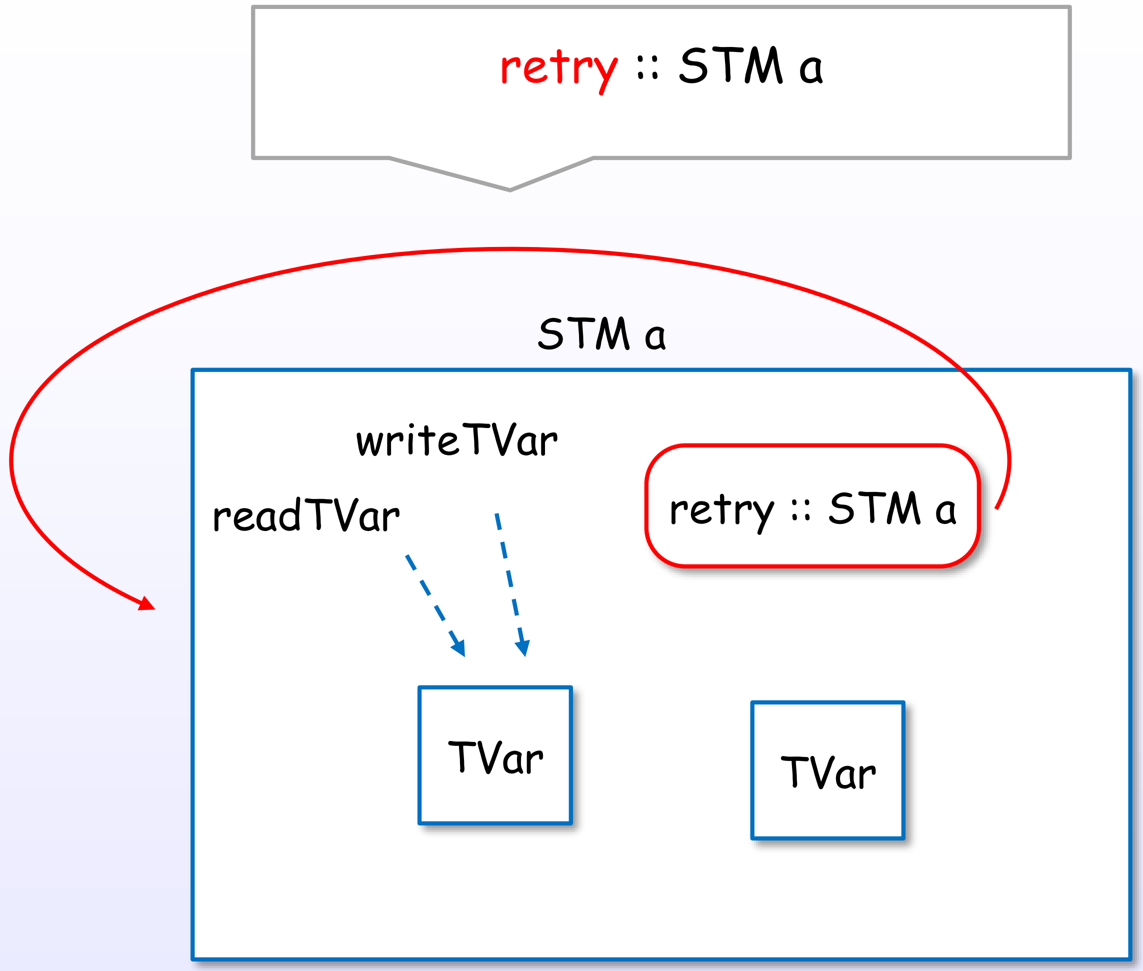


Create and evaluate a composable "atomic block"

Atomic block = All or Nothing

References : [17], [19], [20], [C20], [S12], [S28]

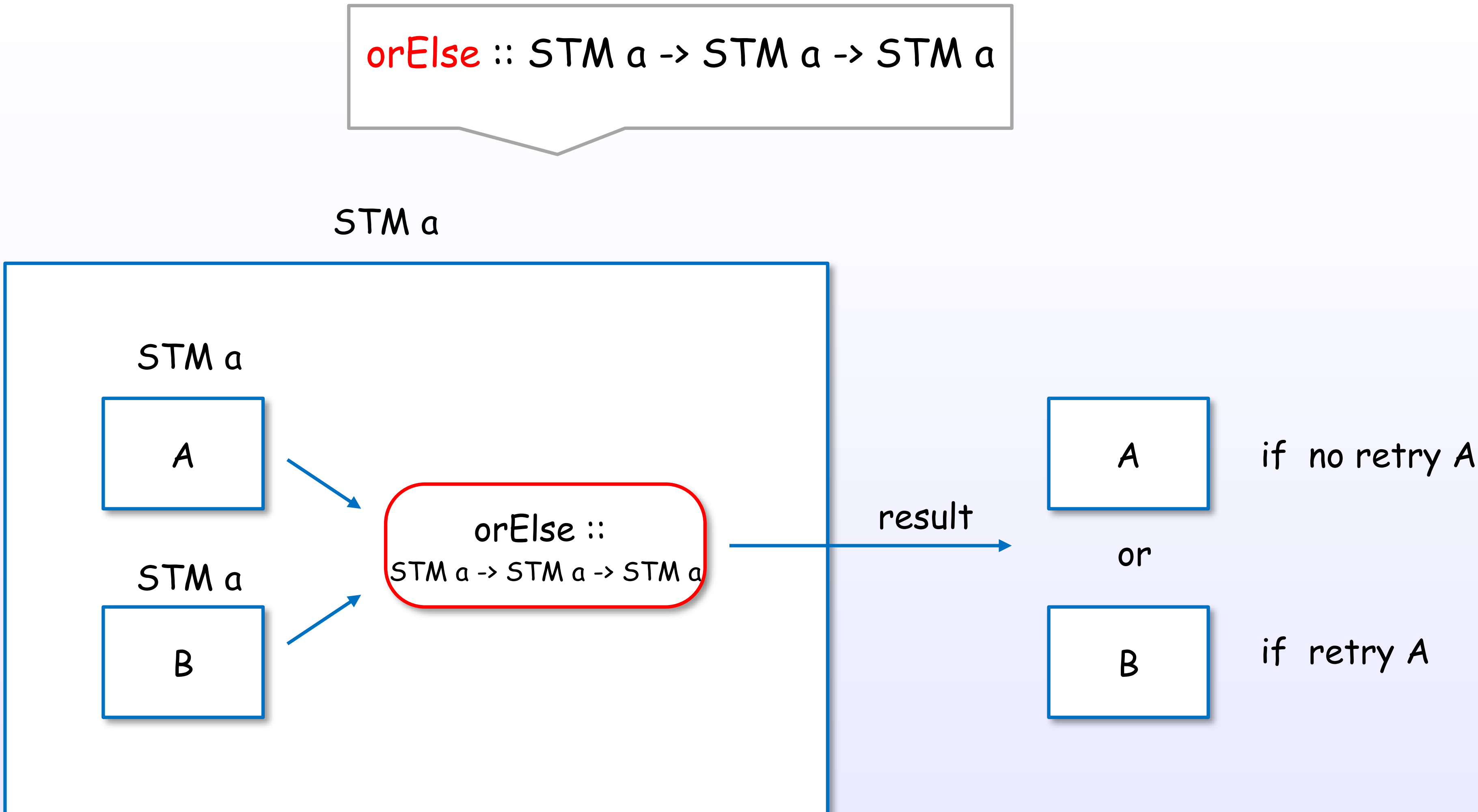
Rollback and blocking control using retry



Discard, blocking and try again

References : [17], [19], [20], [C20], [S12], [S28]

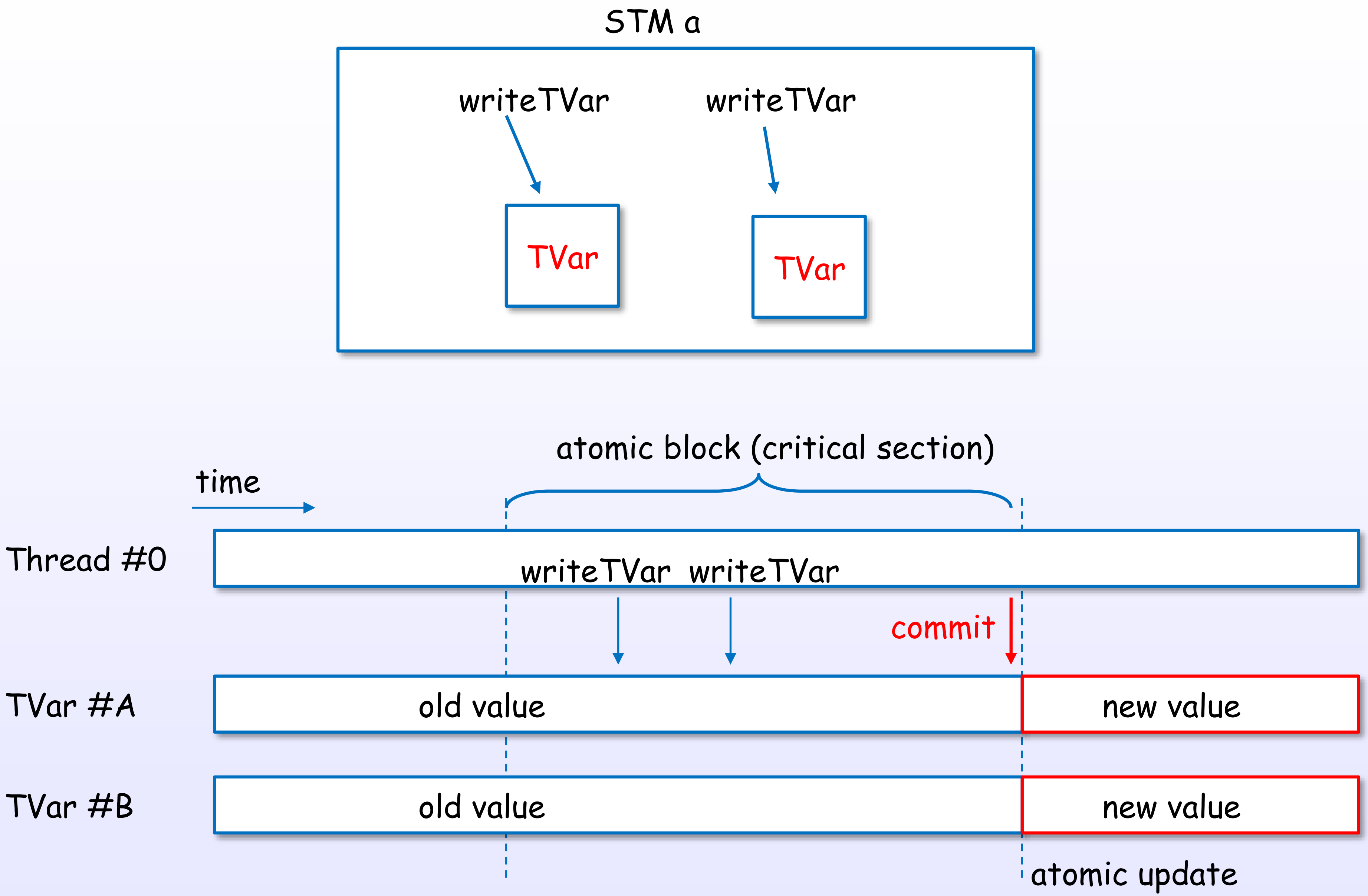
Compose OR case using orElse



A or B or Nothing

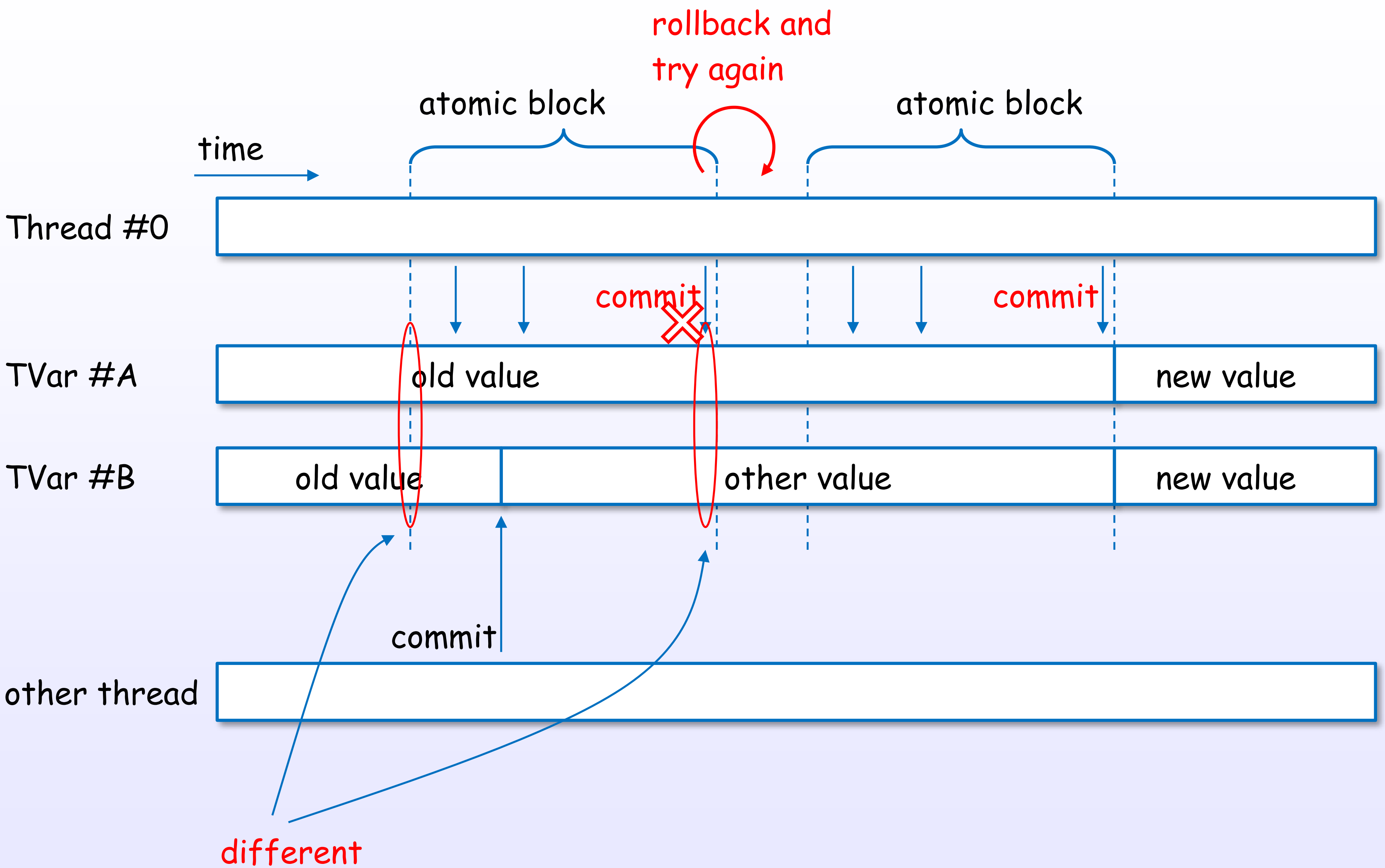
References : [17], [19], [20], [C20], [S12], [S28]

STM, TVar example (normal case)



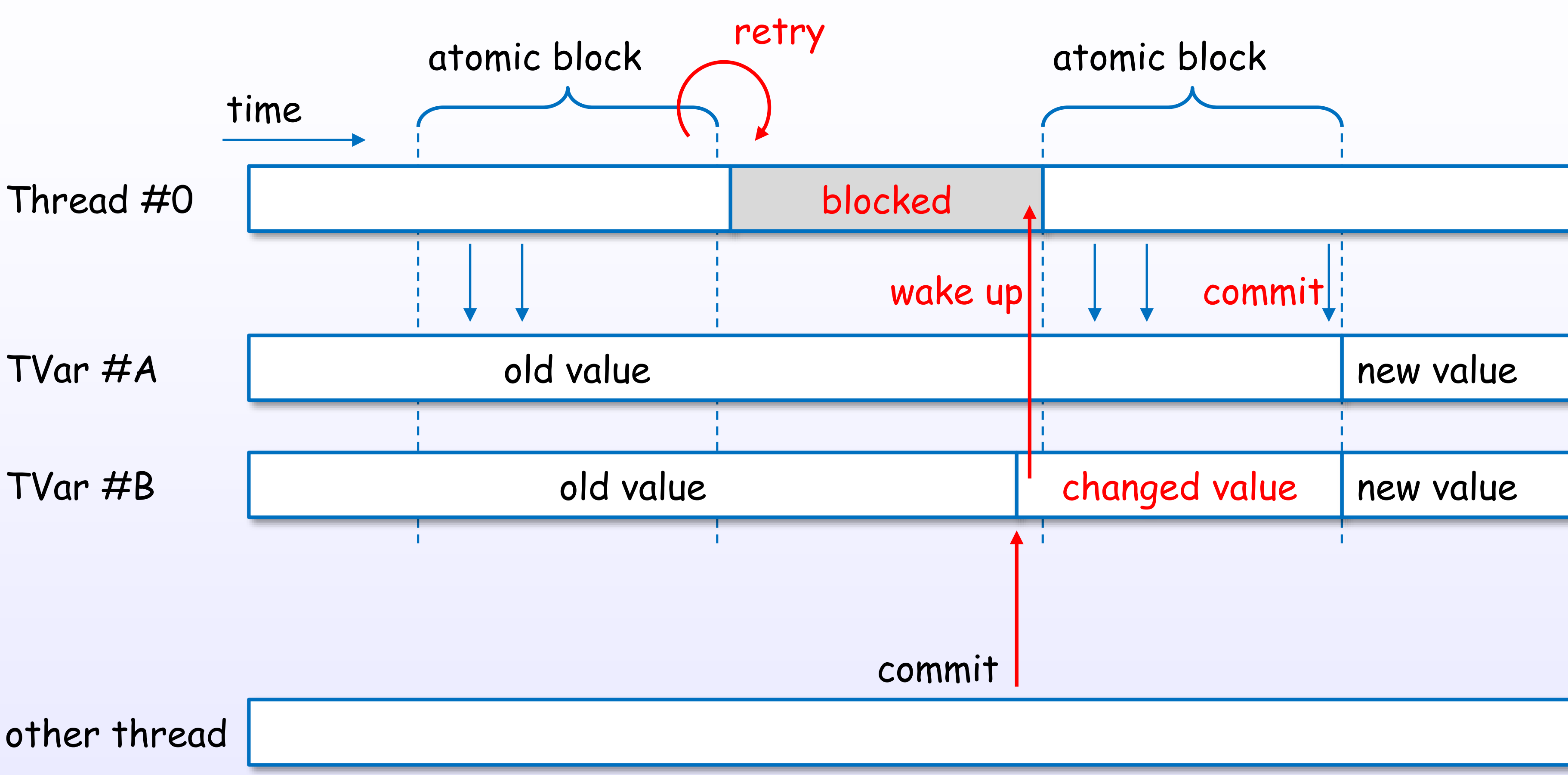
References : [17], [19], [20], [C20], [S12], [S28]

STM, TVar example (conflict case)



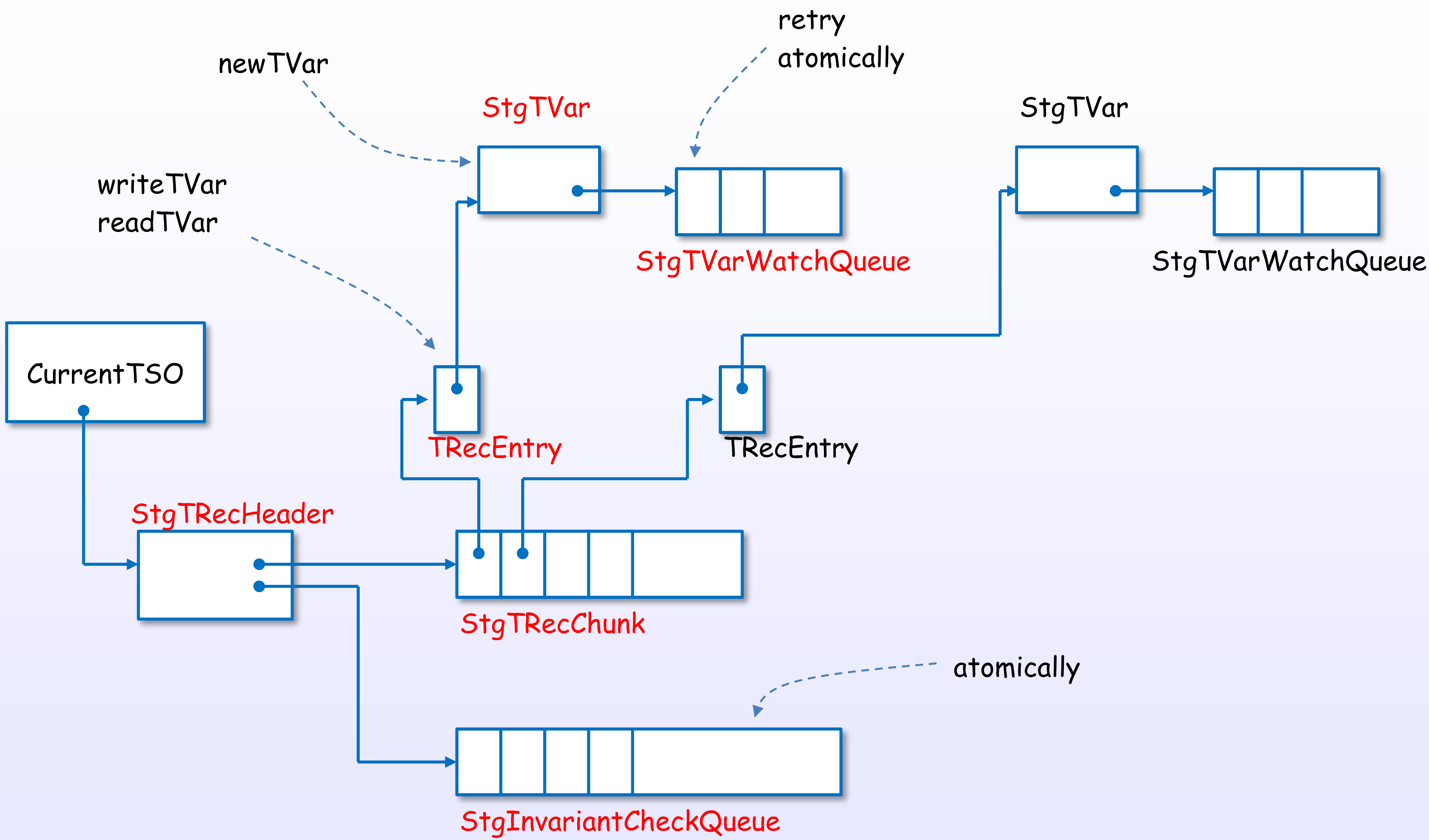
References : [17], [19], [20], [C20], [S12], [S28]

retry example



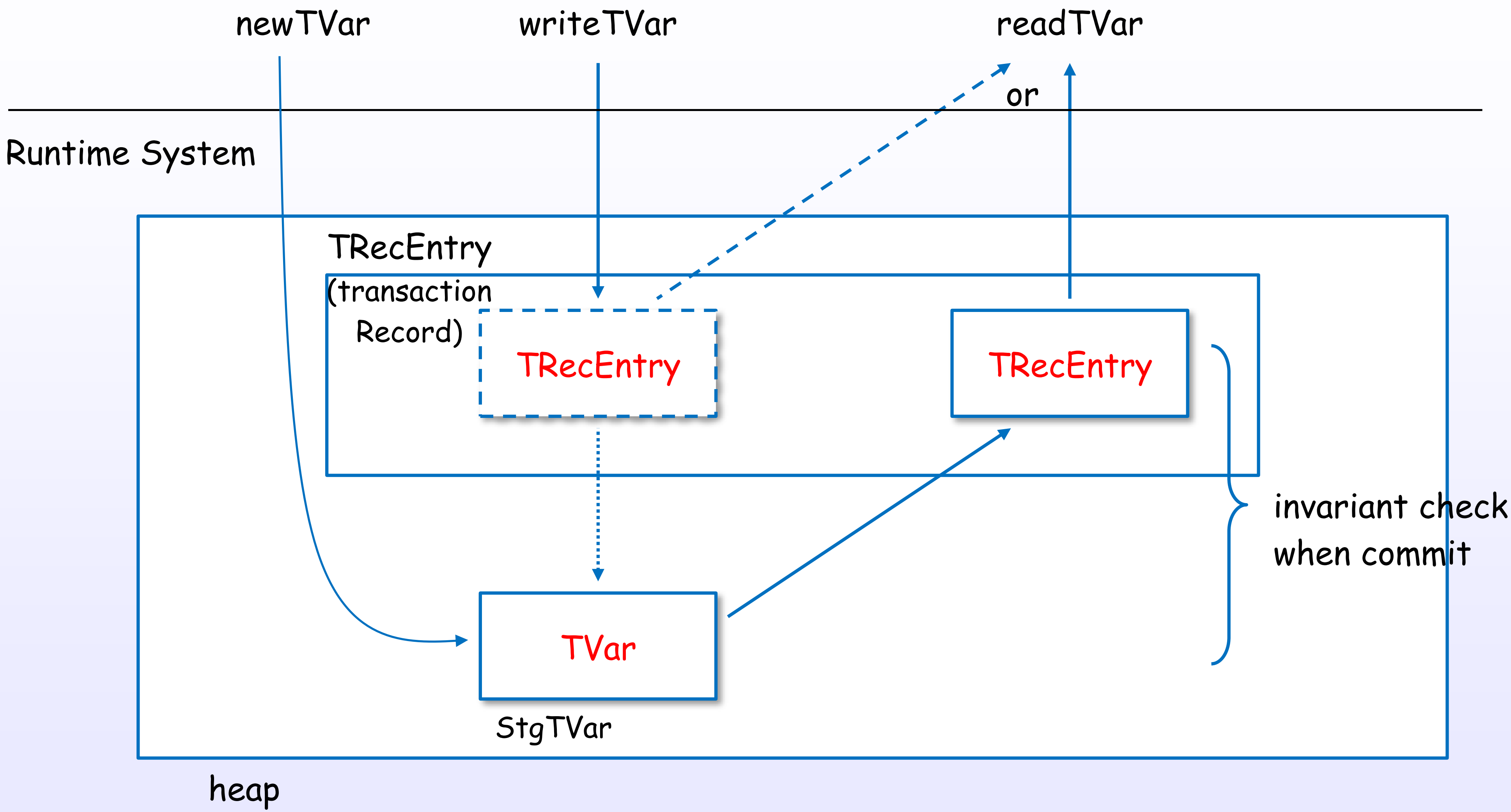
References : [17], [19], [20], [C20], [S12], [S28]

STM, TVar data structure



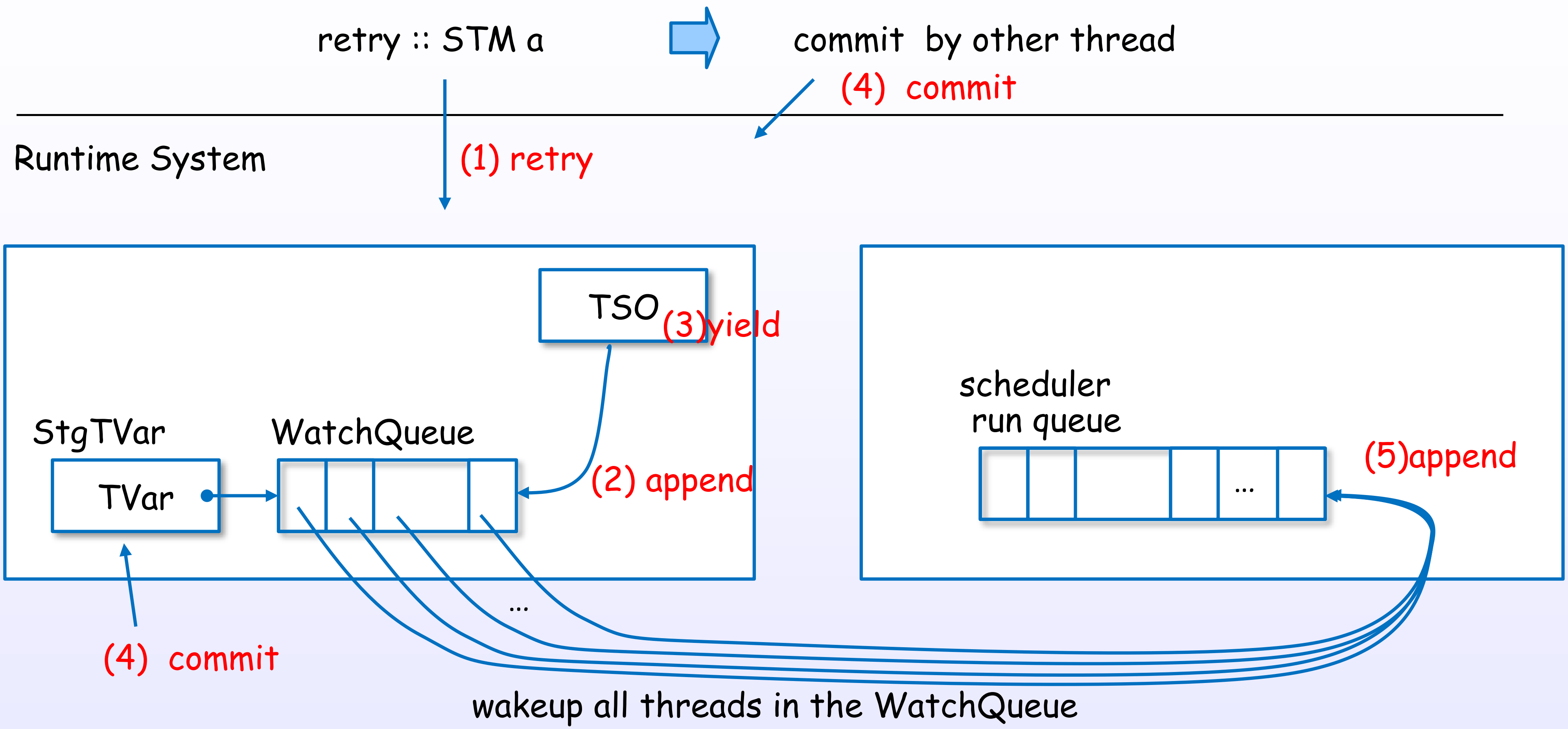
References : [17], [19], [20], [C20], [S12], [S28]

newTVar, writeTVar, readTVar



References : [17], [19], [20], [C20], [S12], [S28]

block by retry, wake up by commit

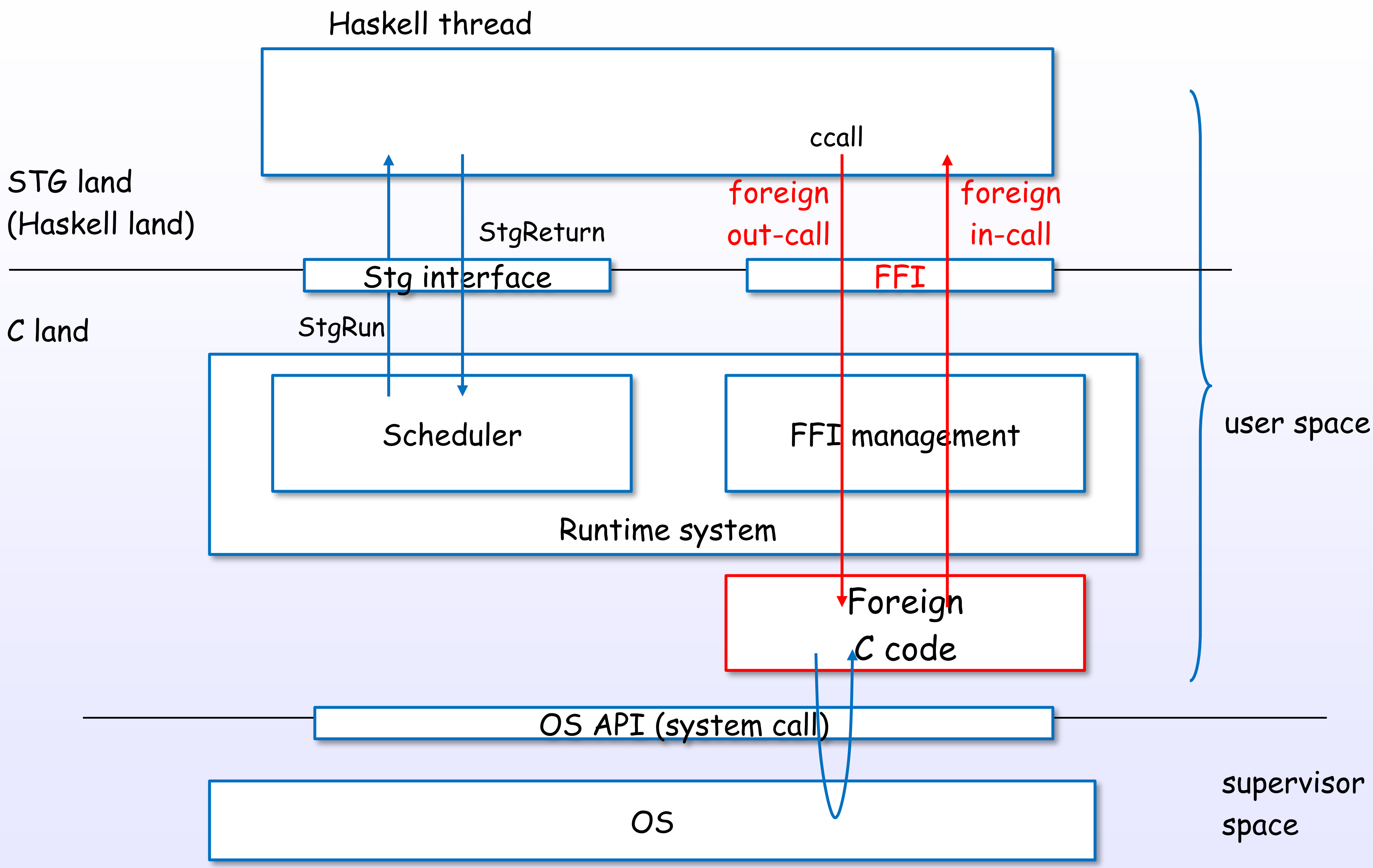


no guarantee of fairness,
because the RTS has to run all the blocked transaction.

References : [17], [19], [20], [C20], [S12], [S28]

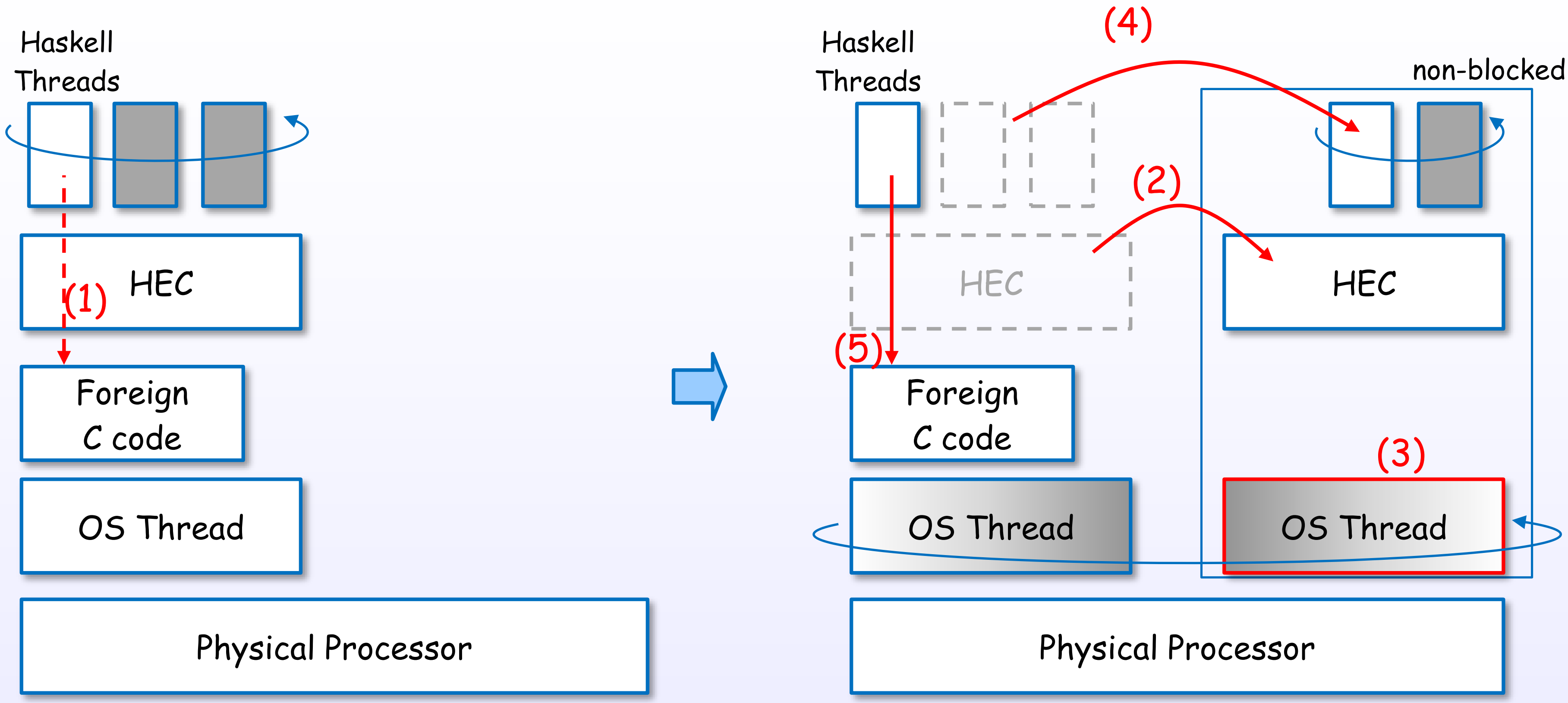
FFI

FFI (Foreign Function Interface)



References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

FFI and OS Threads



(1) a safe foreign call (FFI)

(2) move the HEC to other OS thread

(3) spawn or draw an OS thread

(4) move Haskell threads

(5) call the foreign C code

References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

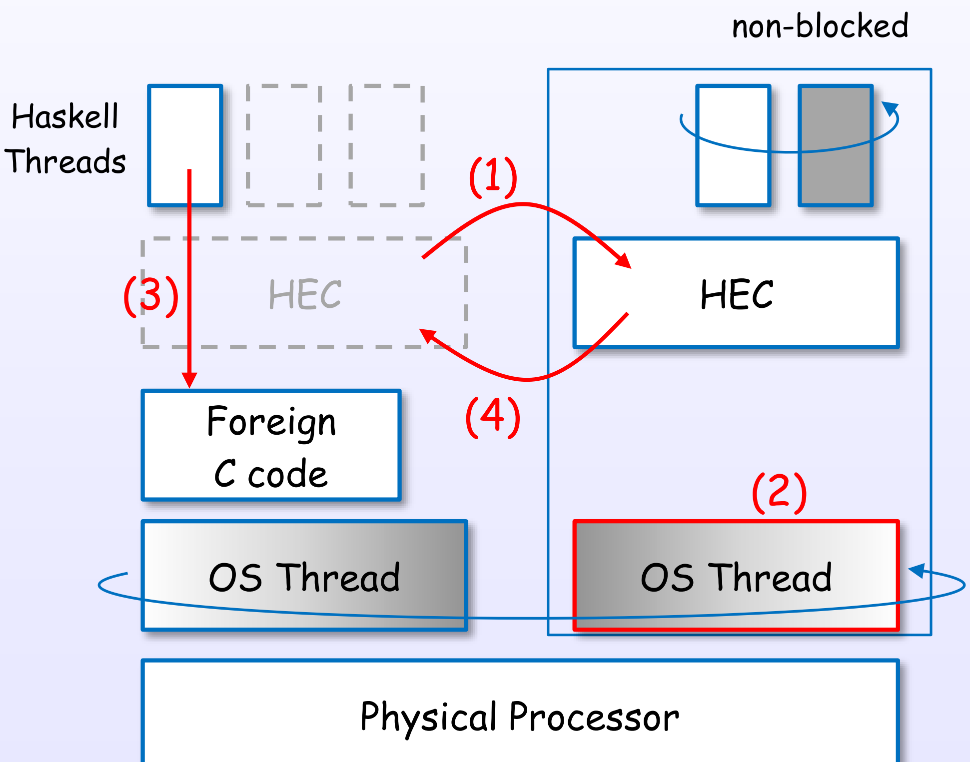
A safe foreign call (code)

Haskell Threads

```
ccall  suspendThread
ccall  FOREIGN_C_CODE    ... (3)
ccall  resumeThread
```

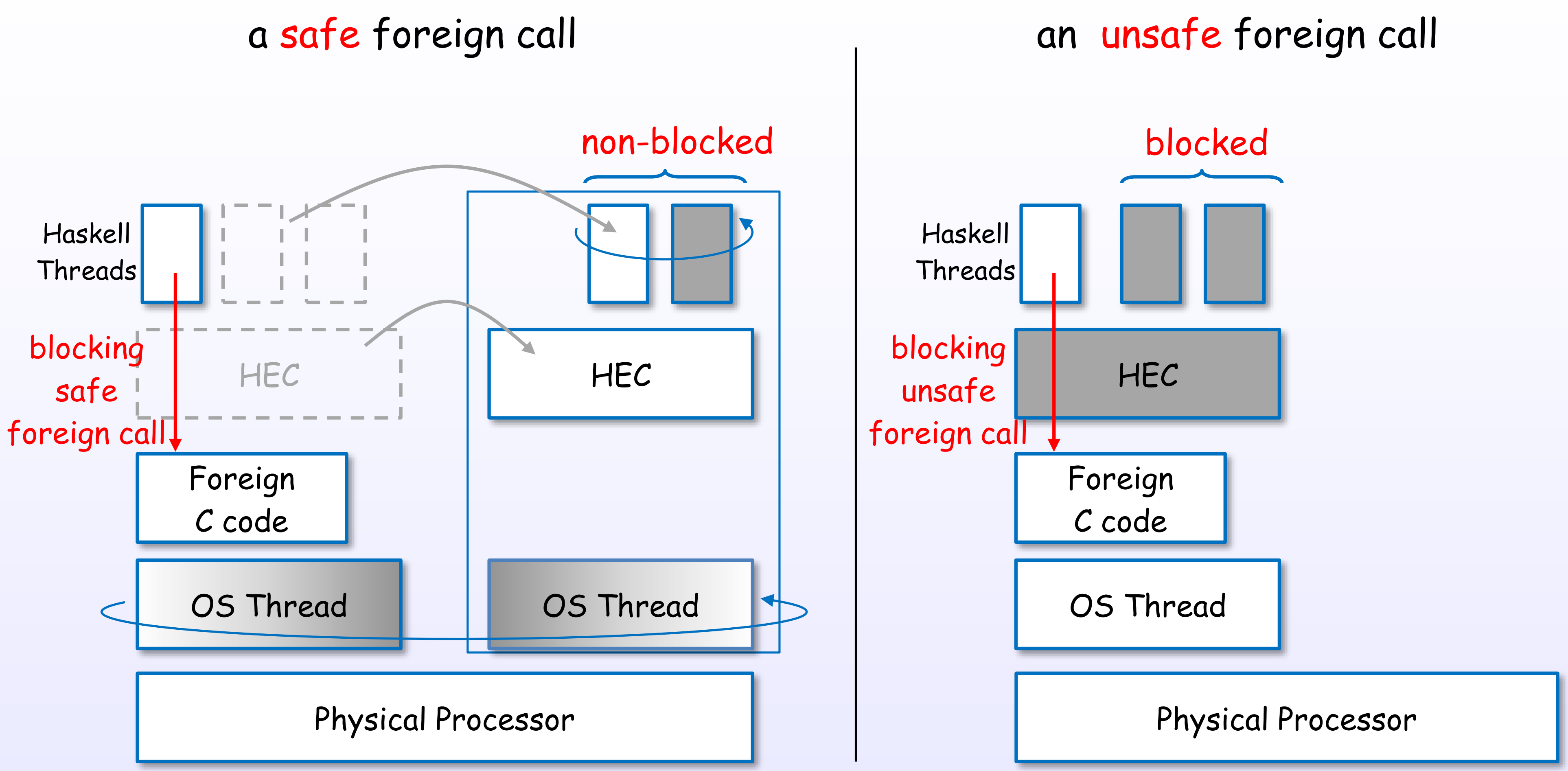
```
releaseCapability_
   giveCapabilityToTask    ... (1)
   startWorkerTask
      createOSThread       ... (2)
```

```
waitForReturnCapability    ... (4)
```



References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

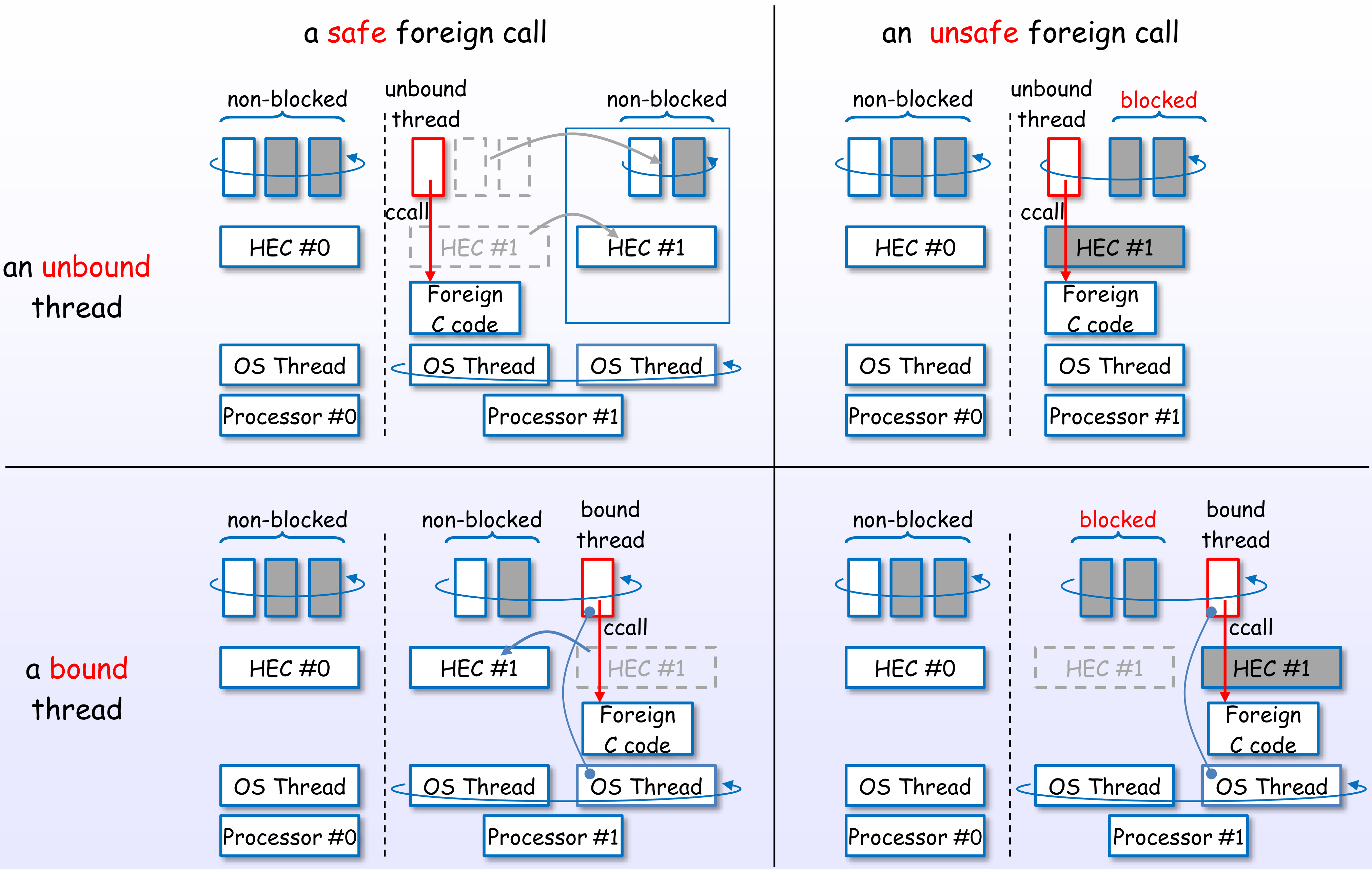
a safe and an unsafe foreign call



faster,
but blocking to the other Haskell threads

References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

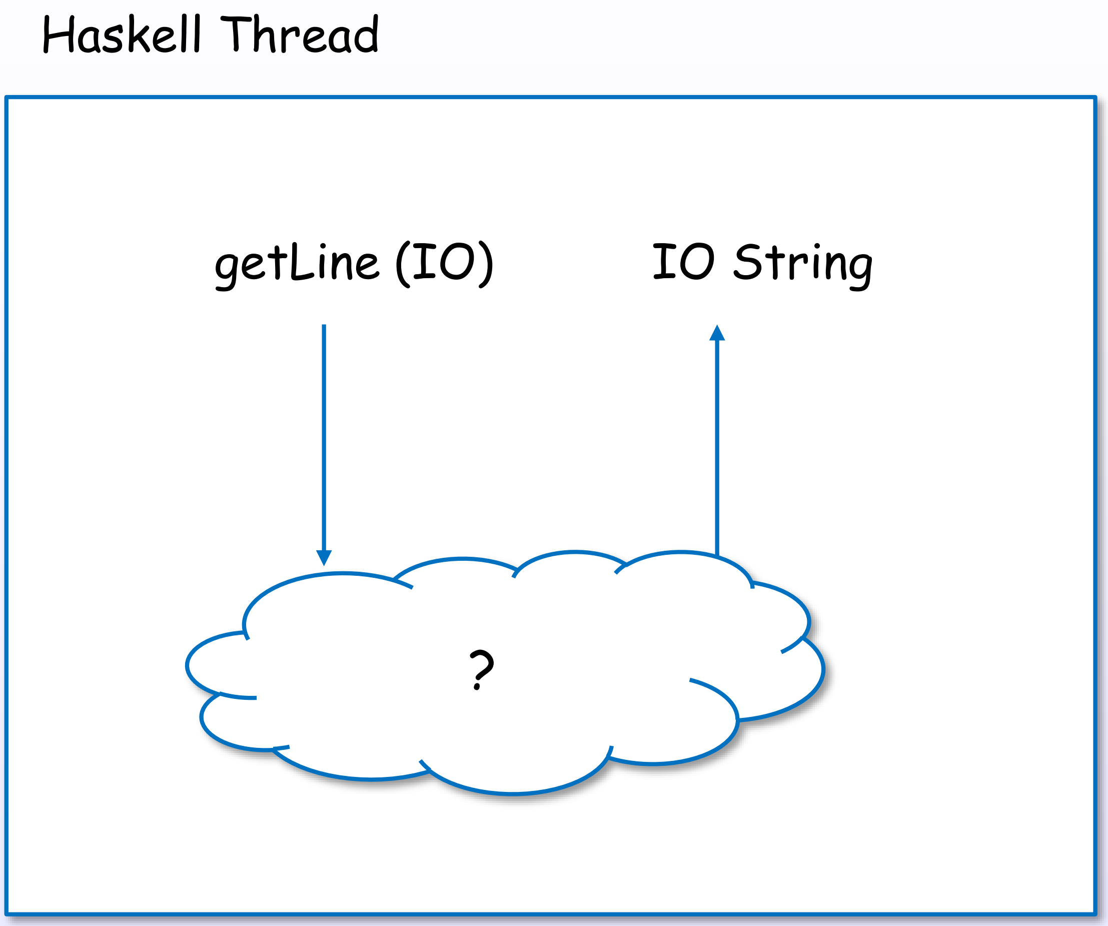
Safe/unsafe foreign call and bound/unbound thread



References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

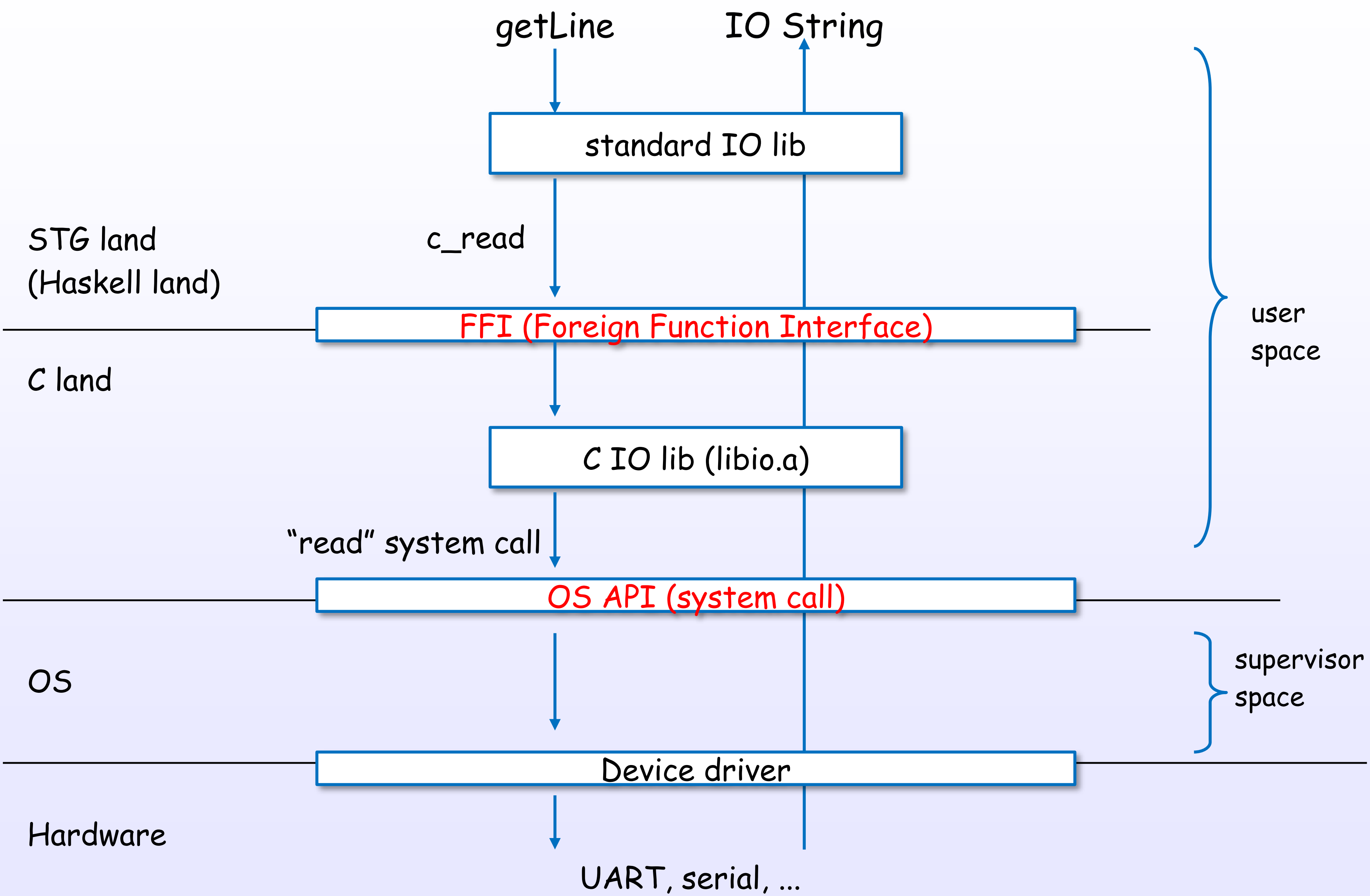
IO and FFI

IO



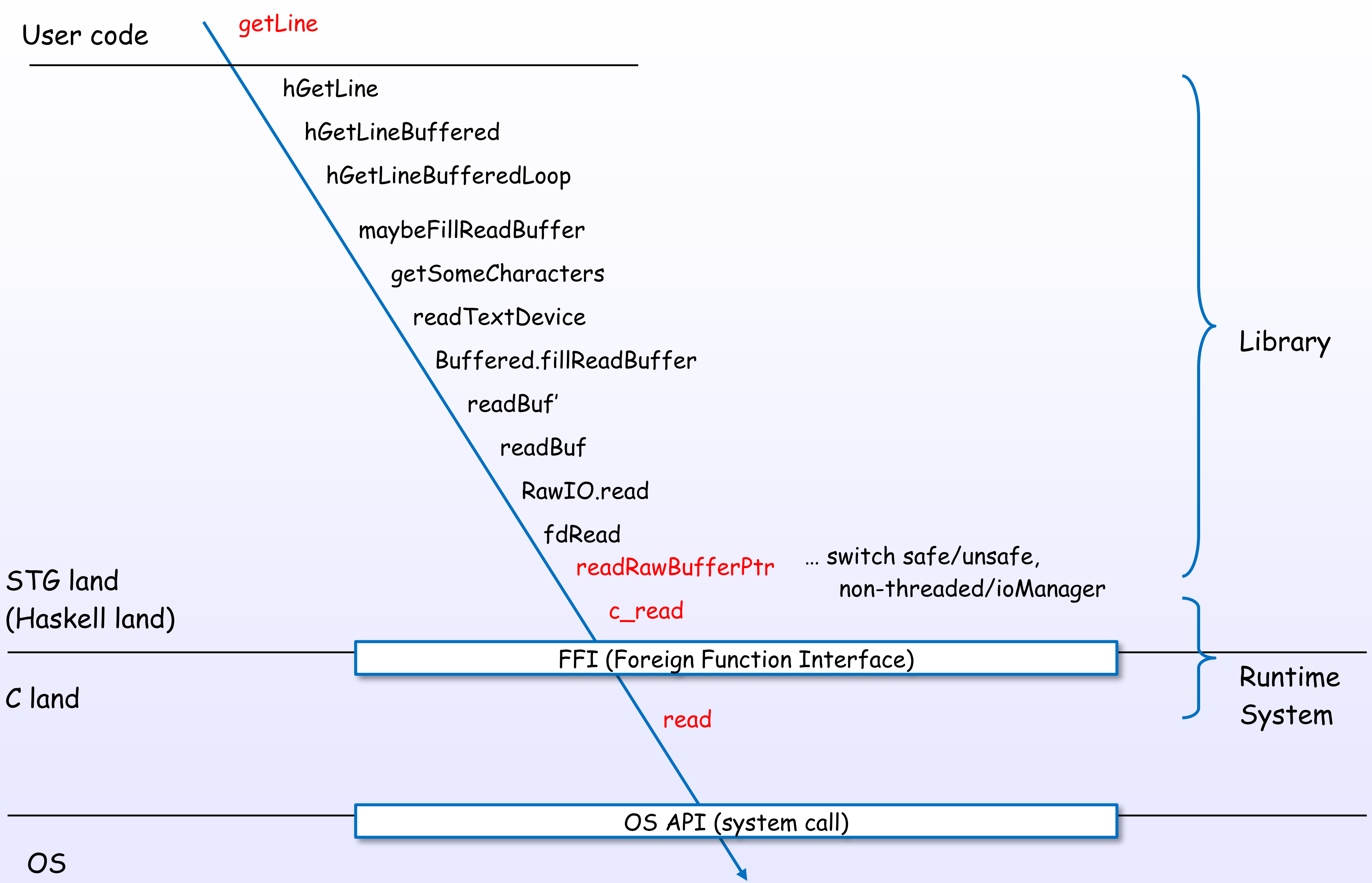
References : [6], [11]

IO example: getLine



References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

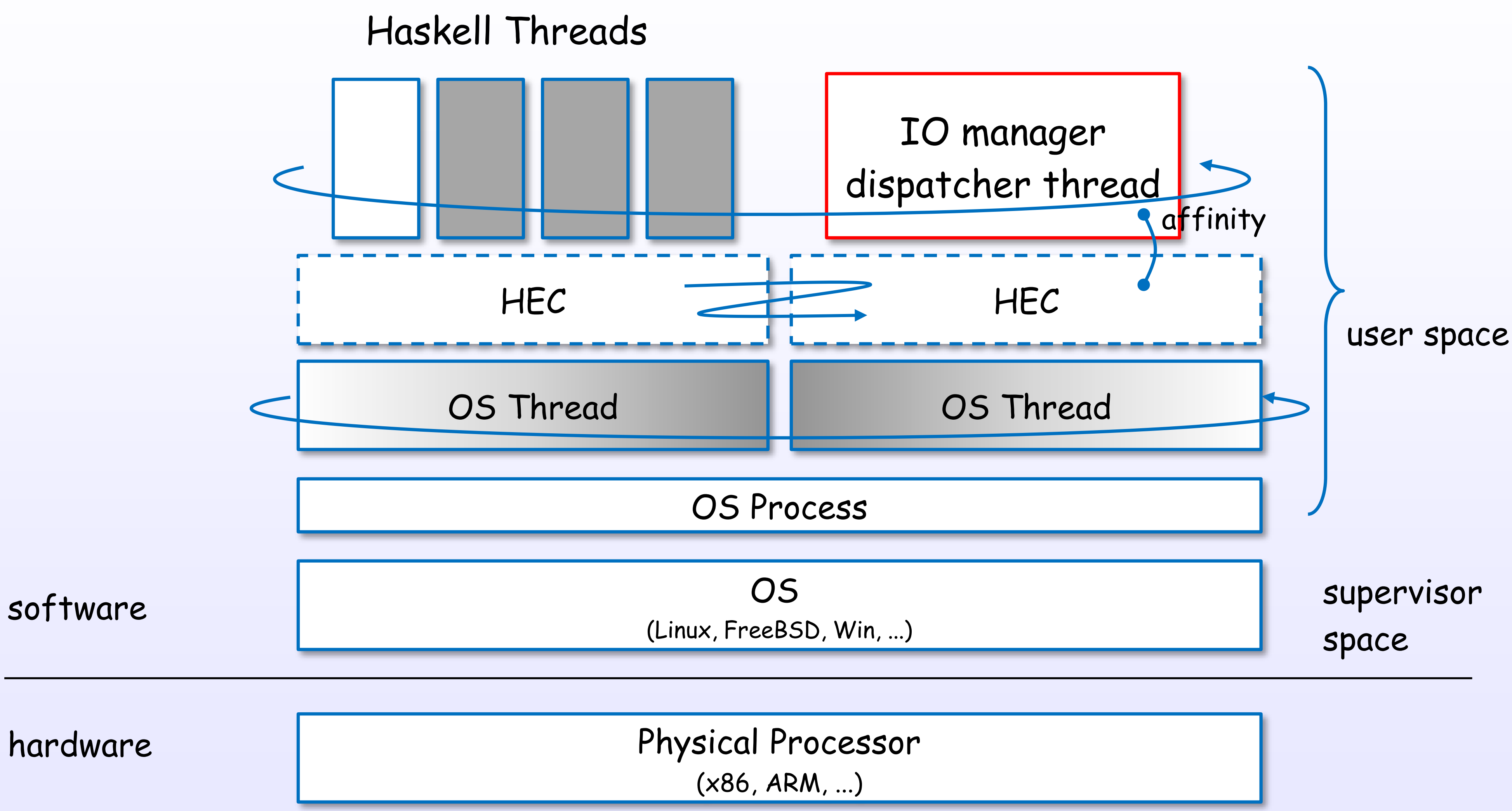
IO example: getLine (code)



References : [6], [11], [20], [S40], [S39], [S38], [S37], [S41]

IO manager

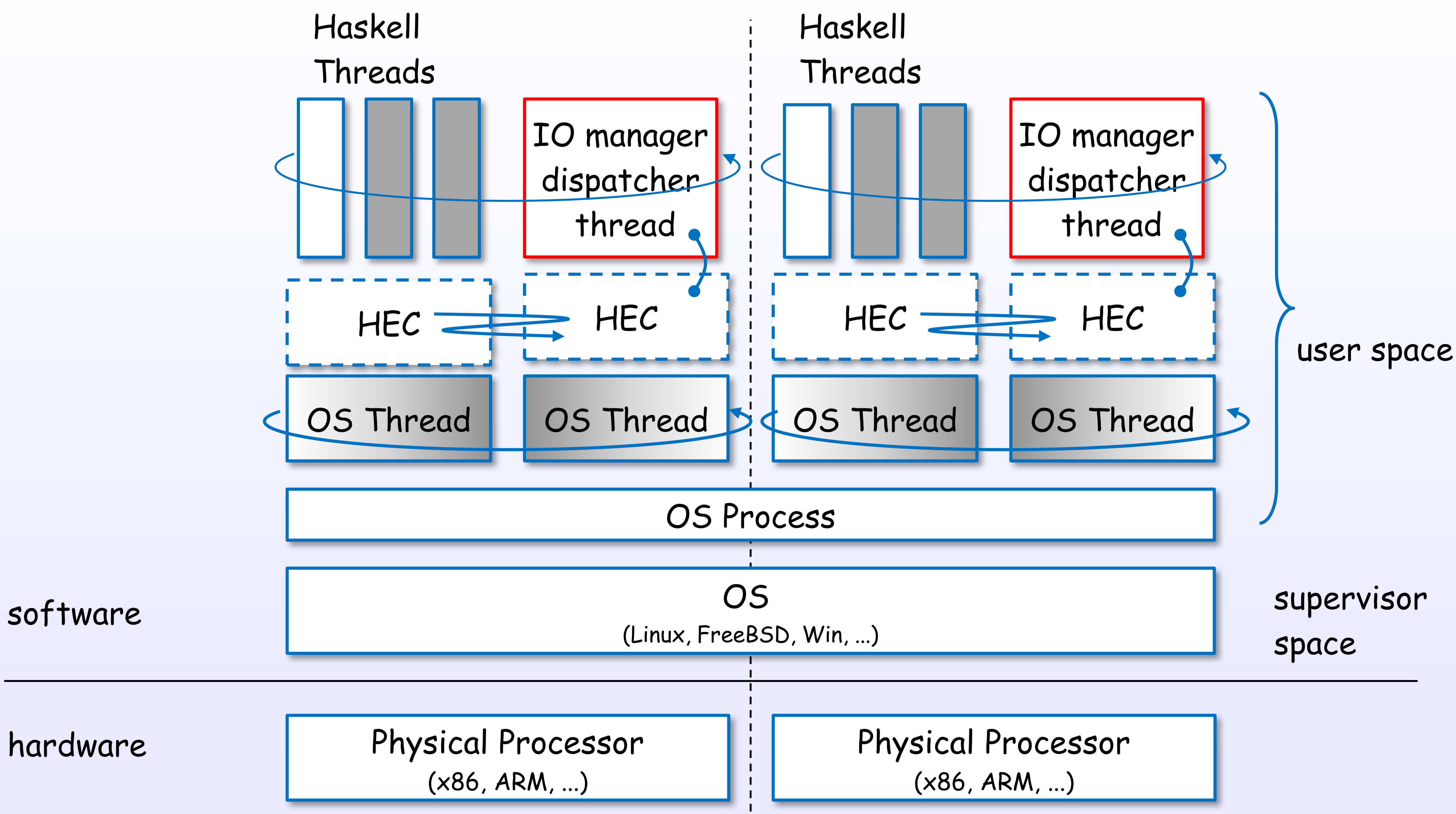
IO manager (single core)



*Threaded option case (ghc -threaded)

References : [7], [5], [8]

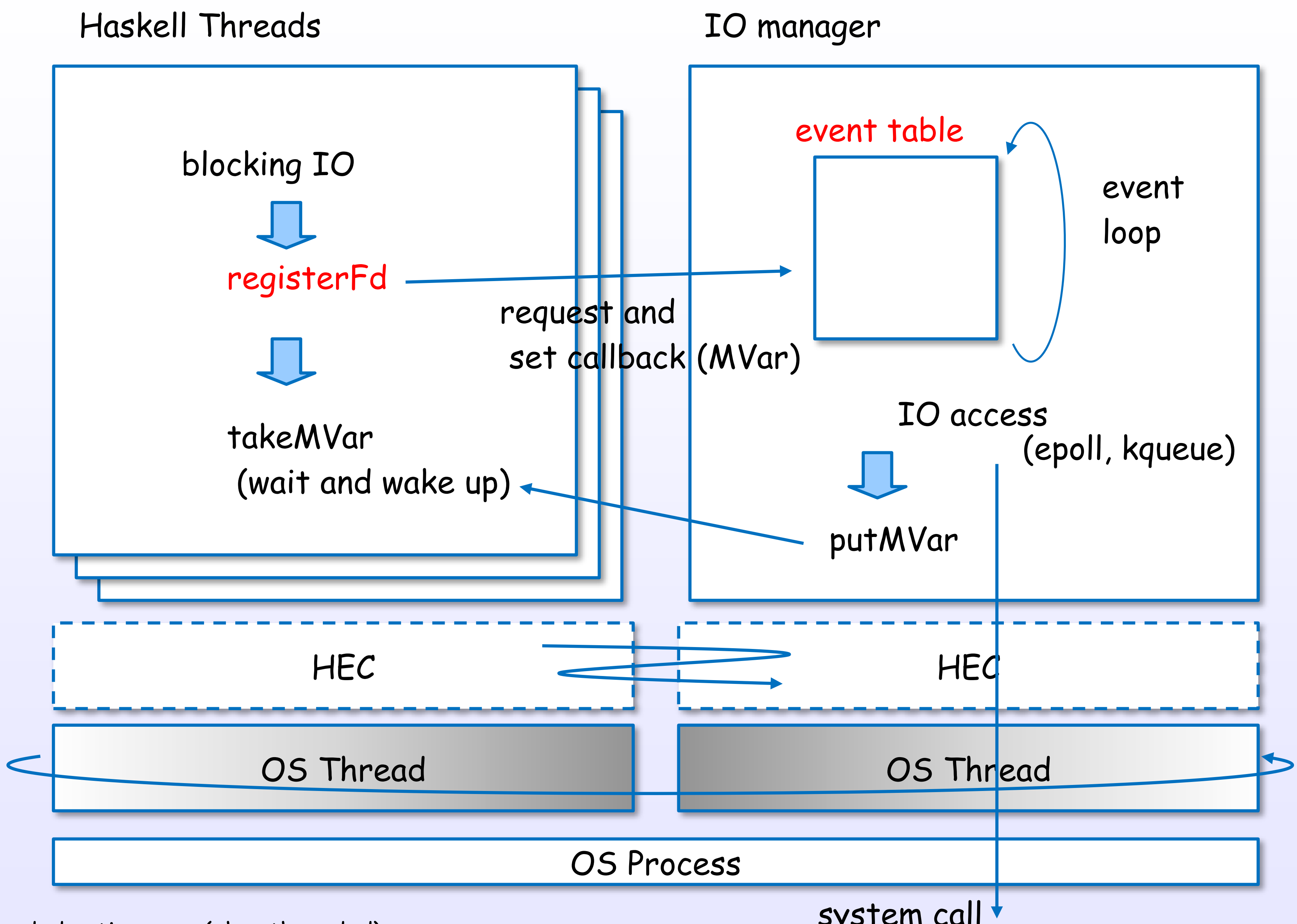
IO manager (multi core)



*Threaded option case (ghc -threaded)

References : [7], [5], [8]

IO manager

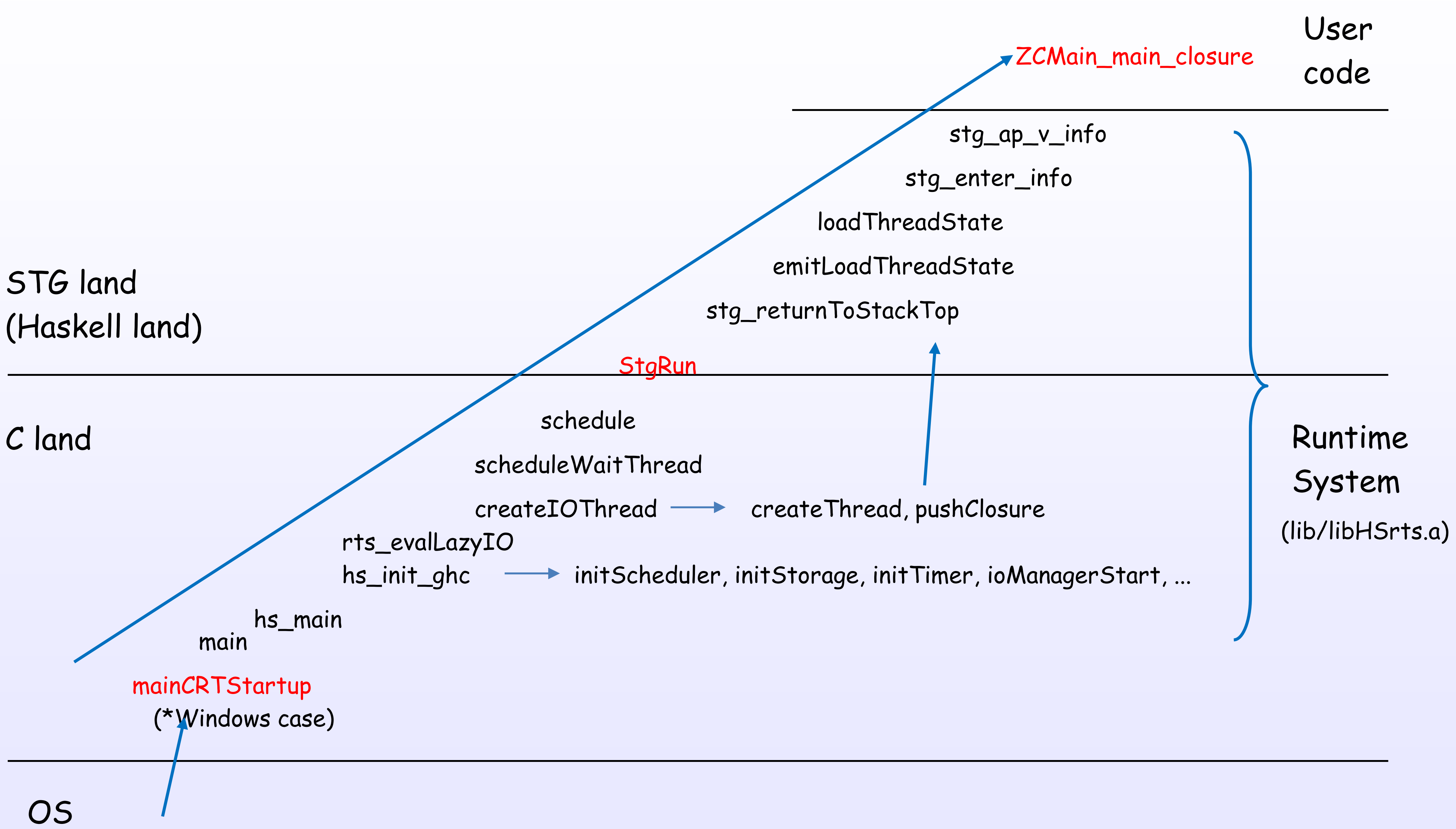


*Threaded option case (ghc -threaded)

References : [7], [5], [8], [S29], [S30], [S33], [S38], [S36], [S3]

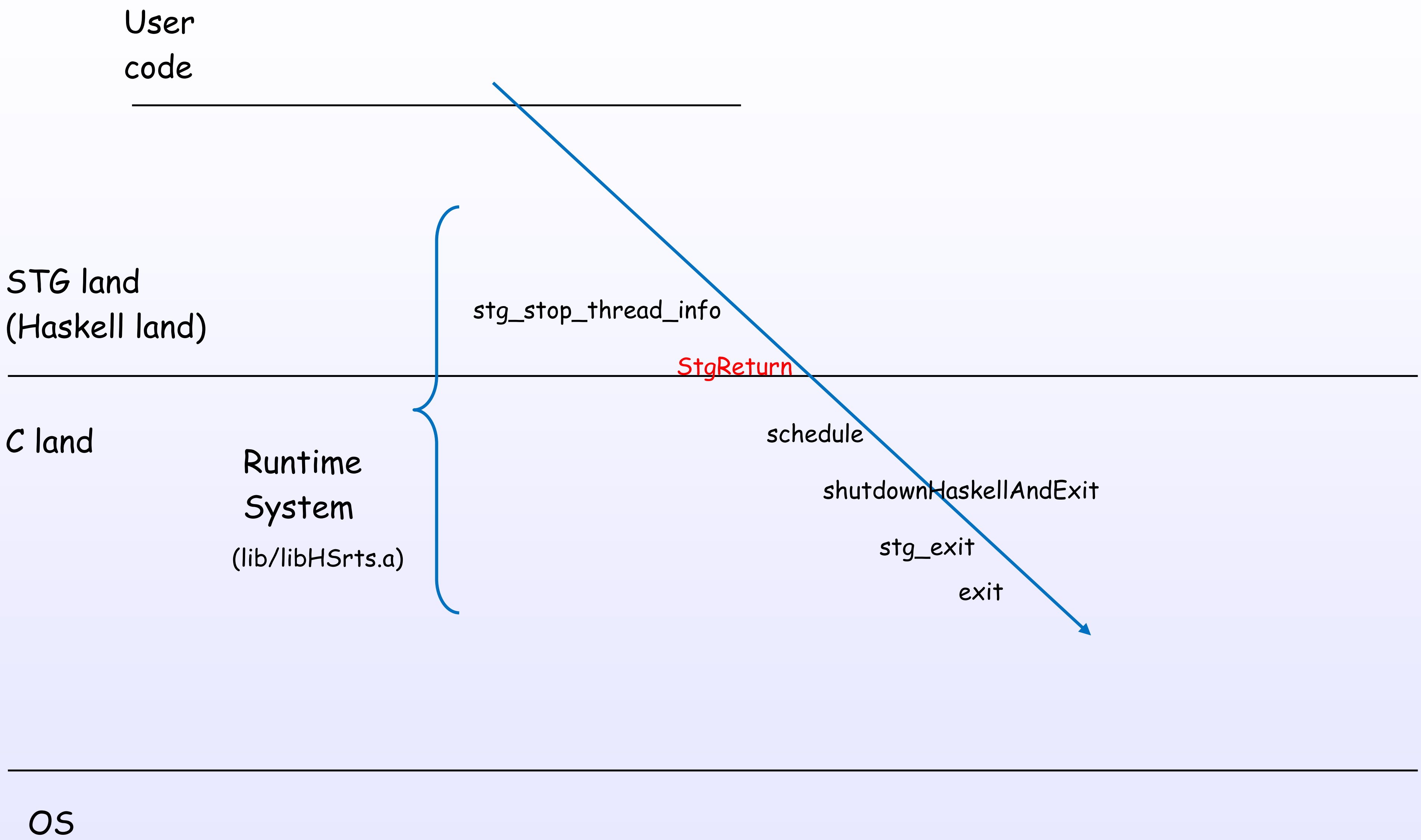
Bootstrap

Bootstrap sequence



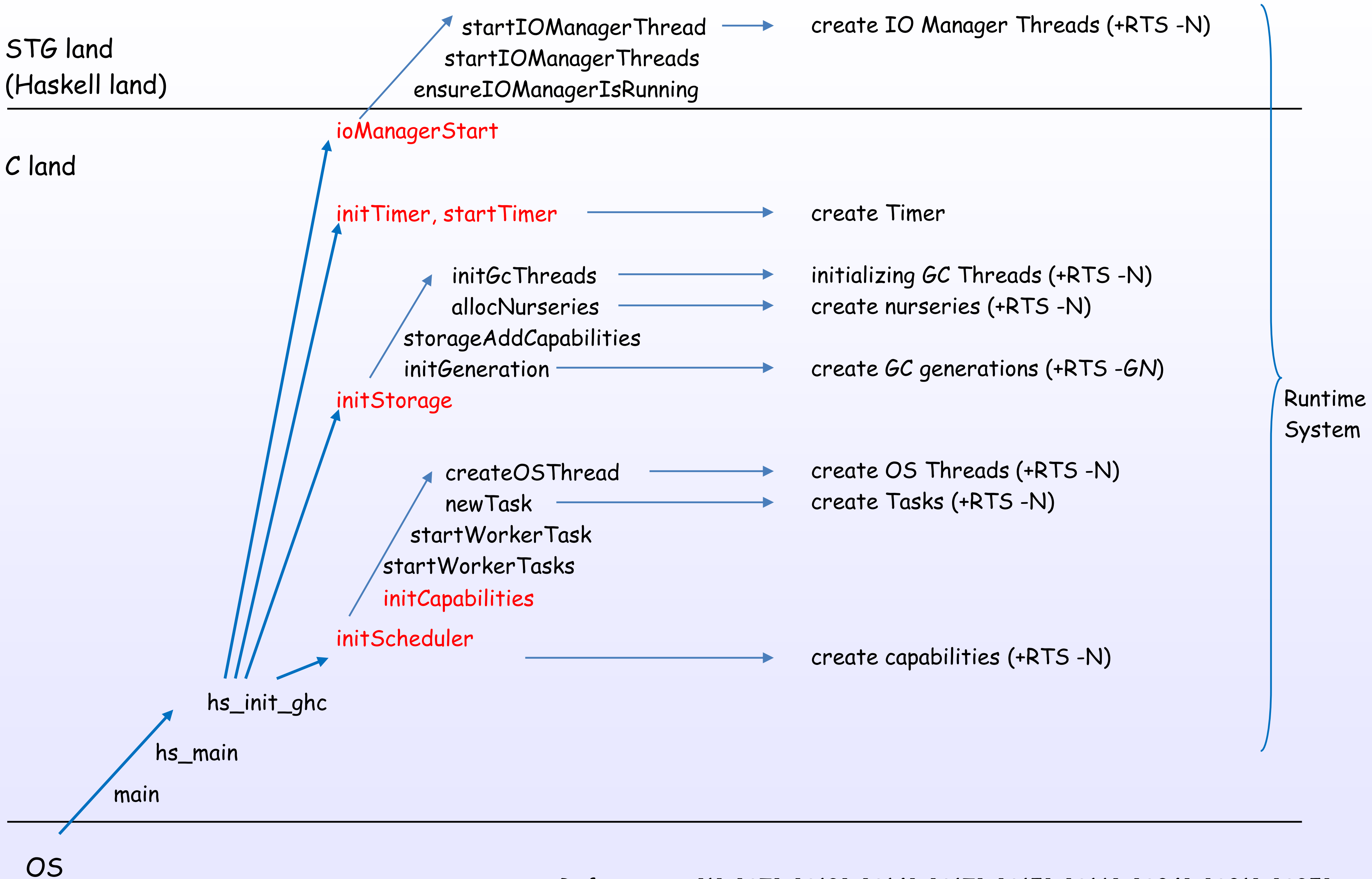
References : [S7], [S13], [S14], [S17], [S18], [S19], [S9], [S10], [S21], [S42]

Exit sequence



References : [S19], [S18], [S17]

Initializing



References : [1], [S7], [S13], [S14], [S17], [S15], [S16], [S24], [S21], [S35]

Create each layers

	Layer	User code	Runtime system	OS API
User code	Sparks	rpar	newSpark	
	Haskell threads	forkIO forkOn forkOS	stg_forkzh, createThread stg_forkOnzh, createThread forkOS_createThread	
Runtime system	HECs (Capability, Virtual processor)		initCapability	
	Tasks (Worker thread) abstract OS Thread	forkOS	newTask	
OS	OS Threads	forkOS	createOSThread	pthread_create
	OS Process			fork
software	OS (Linux, FreeBSD, Win, ...)			
hardware	Physical Processor (x86, ARM, ...)			

References : [1], [5], [8], [9], [C13], [C19], [S12], [S26], [S22], [S15], [S23]

References

References

[1] The Glorious Glasgow Haskell Compilation System User's Guide
https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/index.html

[2] Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine Version 2.5
https://www.microsoft.com/en-us/research/wp-content/uploads/1992/04/spineless-tagless-gmachine.pdf

[3] Making a Fast Curry Push/Enter vs Eval/Apply for Higher-order Languages
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/eval-apply.pdf

[4] Faster Laziness Using Dynamic Pointer Tagging
http://research.microsoft.com/en-us/um/people/simonpj/papers/ptr-tag/ptr-tagging.pdf

[5] Runtime Support for Multicore Haskell
http://research.microsoft.com/en-us/um/people/simonpj/papers/parallel/multicore-ghc.pdf

[6] Extending the Haskell Foreign Function Interface with Concurrency
http://community.haskell.org/~simonmar/papers/conc-ffi.pdf

[7] Mio: A High-Performance Multicore IO Manager for GHC
http://haskell.cs.yale.edu/wp-content/uploads/2013/08/hask035-voellmy.pdf

[8] The GHC Runtime System
http://web.mit.edu/~ezyang/Public/jfp-ghc-rts.pdf

[9] The GHC Runtime System
http://www.scs.stanford.edu/14sp-cs240h/slides/ghc-rts.pdf

[10] Evaluation on the Haskell Heap
http://blog.ezyang.com/2011/04/evaluation-on-the-haskell-heap/

References

[11] IO evaluates the Haskell Heap
http://blog.ezyang.com/2011/04/io-evaluates-the-haskell-heap/

[12] Understanding the Stack
http://www.well-typed.com/blog/94/

[13] Understanding the RealWorld
http://www.well-typed.com/blog/95/

[14] The GHC scheduler
http://blog.ezyang.com/2013/01/the-ghc-scheduler/

[15] GHC's Garbage Collector
http://www.mm-net.org.uk/workshop190404/GHC's_Garbage_Collector.ppt

[16] Concurrent Haskell
https://www.microsoft.com/en-us/research/wp-content/uploads/1996/01/concurrent-haskell.pdf

[17] Beautiful Concurrency
https://www.fpcomplete.com/school/advanced-haskell/beautiful-concurrency

[18] Anatomy of an MVar operation
http://blog.ezyang.com/2013/05/anatomy-of-an-mvar-operation/

[19] Parallel and Concurrent Programming in Haskell
https://simonmar.github.io/pages/pcph.html

[20] Real World Haskell
http://book.realworldhaskell.org/

References

[21] A Haskell Compiler
http://www.scs.stanford.edu/16wi-cs240h/slides/ghc-compiler-slides.html

[22] Dive into GHC
http://www.stephendiehl.com/posts/ghc_01.html

[23] Unboxed Values as First-Class Citizens
https://www.microsoft.com/en-us/research/wp-content/uploads/1991/01/unboxed-values.pdf

[24] Levity Polymorphism (extended version)
https://cs.brynmawr.edu/~rae/papers/2017/levity/levity.pdf

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/Compiler/TypeType
[C11] https://ghc.haskell.org/trac/ghc/wiki/LevityPolymorphism
[C12] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts
[C13] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/HeapObjects
[C14] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/Stack
[C15] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/GC
[C16] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution
[C17] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution/Registers
[C18] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/HaskellExecution/PointerTagging
[C19] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Scheduler
[C20] https://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/STM
[C21] 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
[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/ghc-prim/GHC/Types.hs
[S32] libraries/base/GHC/MVar.hs
[S33] libraries/base/GHC/Conc/IO.hs
[S34] libraries/base/GHC/Conc/Sync.hs
[S35] libraries/base/GHC/Event/Manager.hs
[S36] libraries/base/GHC/Event/Thread.hs
[S37] libraries/base/GHC/IO/BufferedIO.hs
[S38] libraries/base/GHC/IO/FD.hs
[S39] libraries/base/GHC/IO/Handle/Text.hs
[S40] libraries/base/System/IO.hs
[S41] libraries/base/System/Posix/Internals.hs
[S42] AutoApply.o (utils/genapply/Main.hs)

Connect the algorithm and transistor