GHC(STG,Cmm,asm) illustrated

for hardware person

exploring some mental models and implementations

Takenobu T.

Rev. 0.00.0
WIP

"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(GC) heap objects
- STG - C land interface

- Thread
- Heap and Threads
- Thread context switch
- Creating main thread and forkIO

- Spark

- MVar

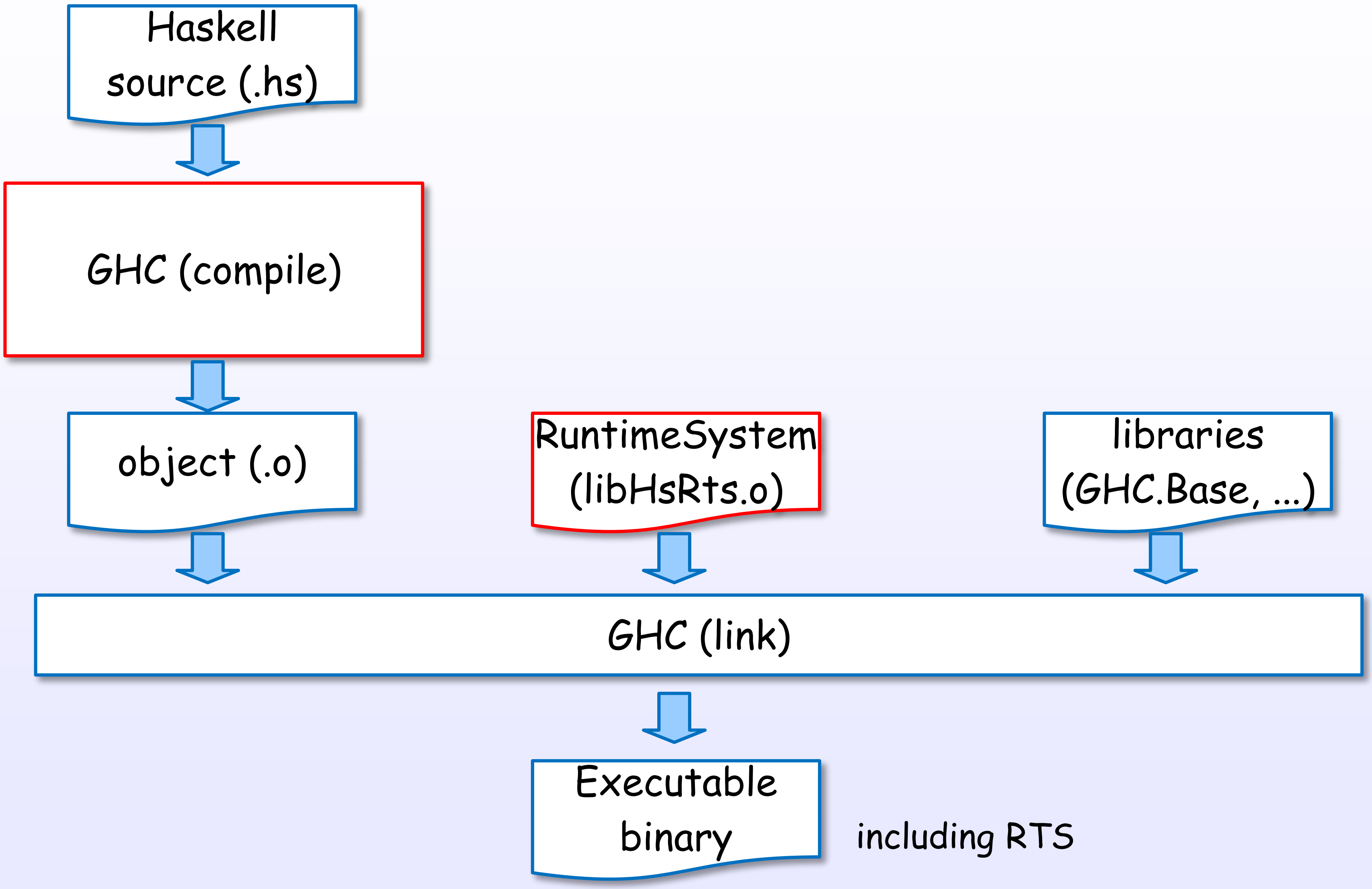
- IO and FFI
- IO manager

- Bootstrap

- References

Executable binary

GHC = Compiler + Runtime System (RTS)



Haskell source (.hs)

GHC (compile)

object (.o)

RuntimeSystem (libHsRts.o)

libraries (GHC.Base, ...)

GHC (link)

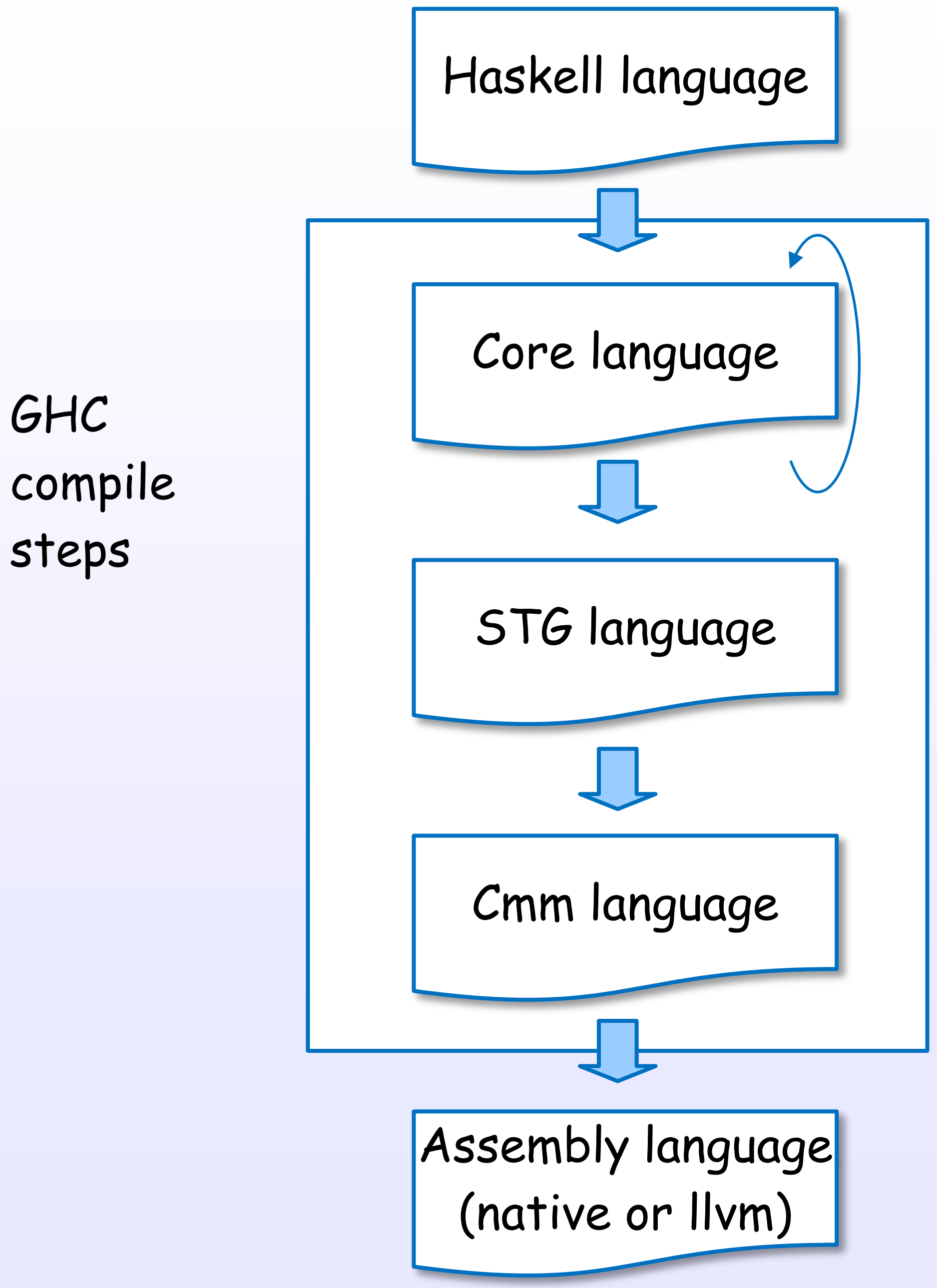
Executable binary

including RTS

References : [C1], [C3], [C10], [C18], [S7]

Compile steps

GHC transitions between five representations



Haskell language

GHC compile steps

Core language

STG language

Cmm language

Assembly language (native or llvm)

each code dumped by

% ghc -ddump-parsed
% ghc -ddump-rn

% ghc -ddump-ds
% ghc -ddump-simpl
% ghc -ddump-prep

% ghc -ddump-stg

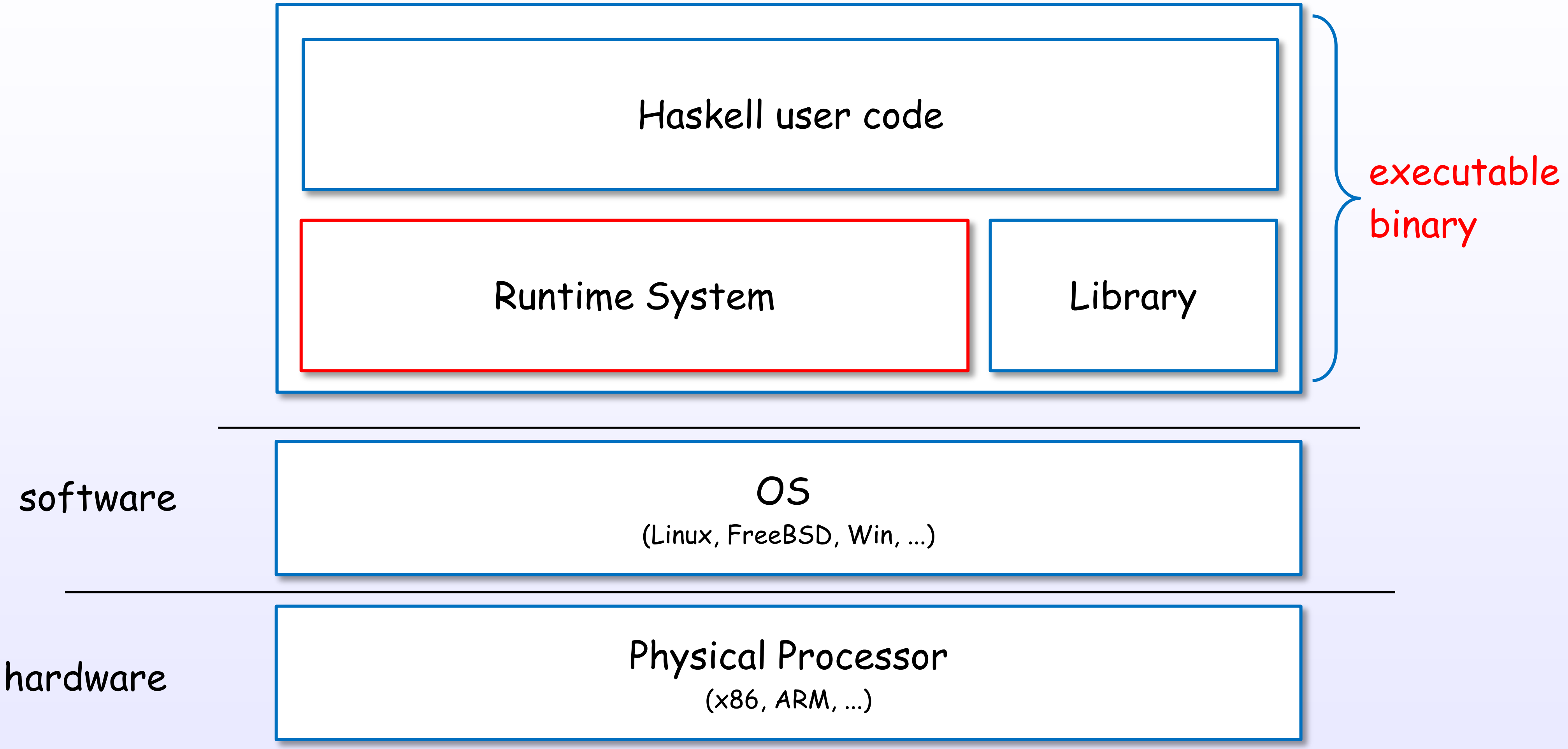
% ghc -ddump-cmm
% ghc -ddump-opt-cmm

% ghc -ddump-llvm
% ghc -ddump-asm

References : [C3], [C4], [8], [C5], [C6], [C7], [C8[], [S7], [S8]

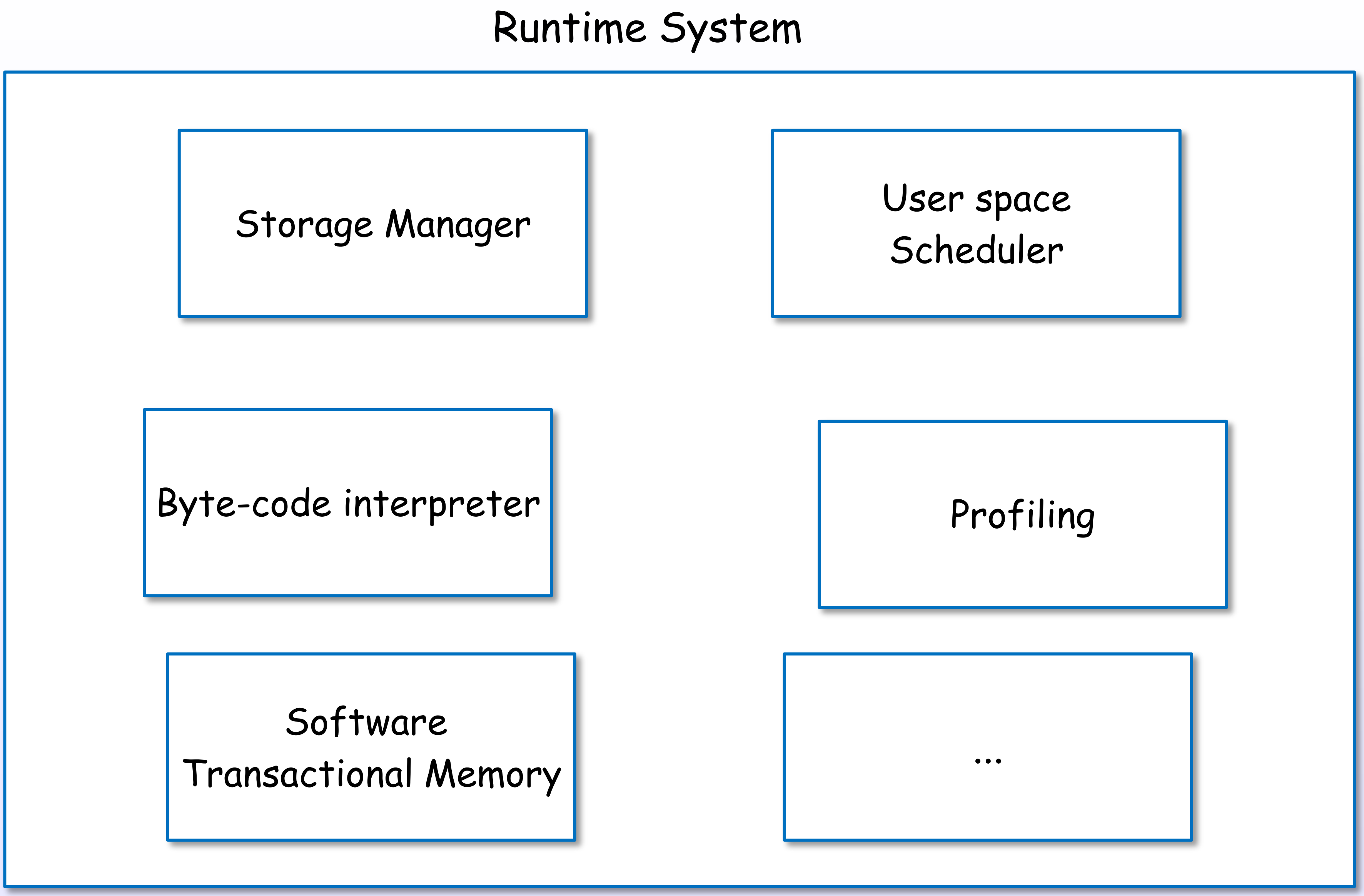
Runtime System

Generated binary includes RTS



References : [C10], [8]

Runtime System includes ...



Runtime System

Storage Manager

User space Scheduler

Byte-code interpreter

Profiling

Software Transactional Memory

...

References : [C10], [7], [8], [4], [16], [S13]

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

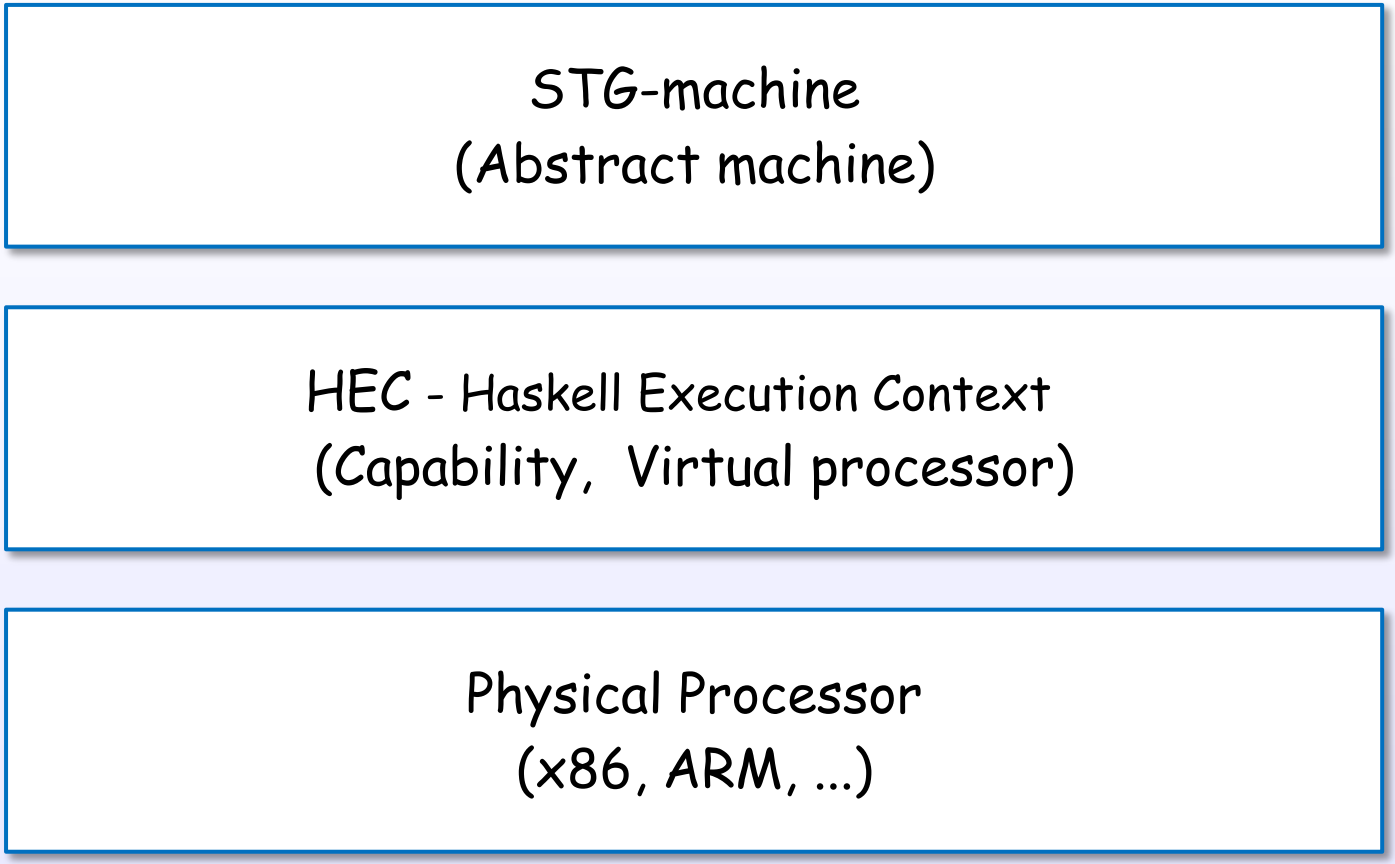
library
($(TOP)/libraries/*)

Haskell
+
C

References : [C2]

Machine layer/models

Machine layer



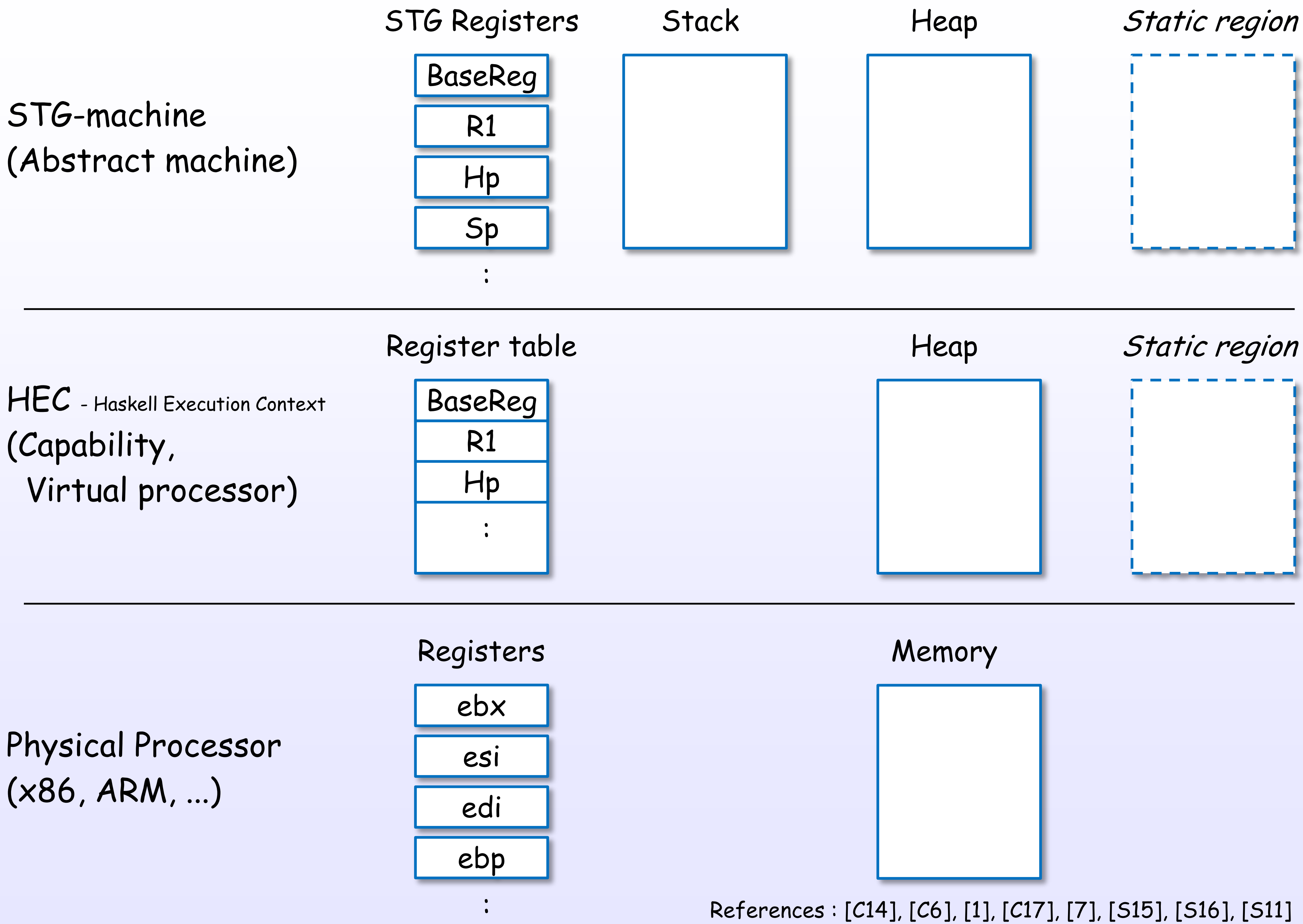
STG-machine
(Abstract machine)

HEC - Haskell Execution Context
(Capability, Virtual processor)

Physical Processor
(x86, ARM, ...)

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

Machine layer



STG-machine
(Abstract machine)

STG Registers

BaseReg

R1

Hp

Sp

:

Stack

Heap

Static region

HEC - Haskell Execution Context
(Capability,
Virtual processor)

Register table

BaseReg

R1

Hp

:

Heap

Static region

Physical Processor
(x86, ARM, ...)

Registers

ebx

esi

edi

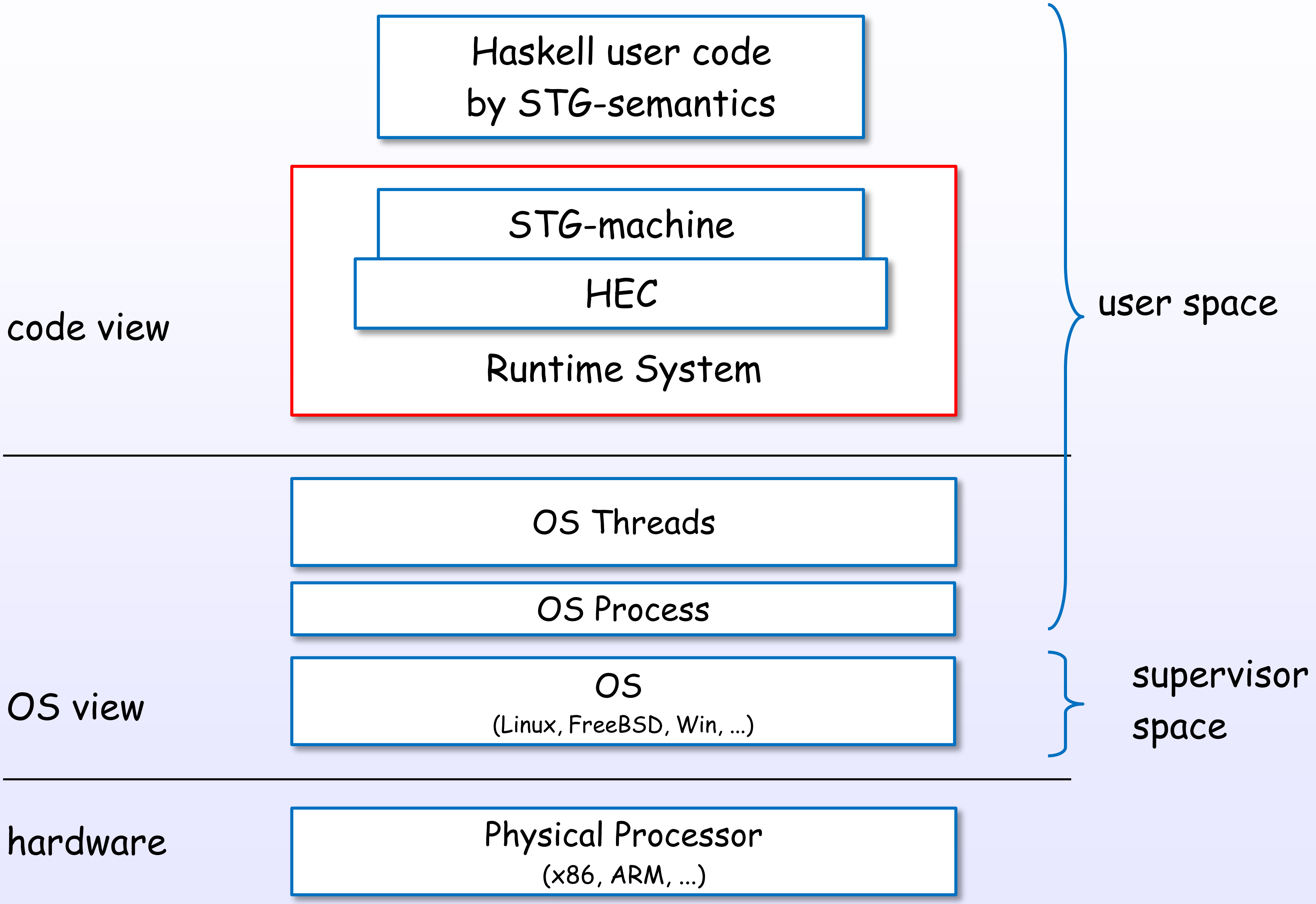
ebp

:

Memory

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

Runtime system and HEC

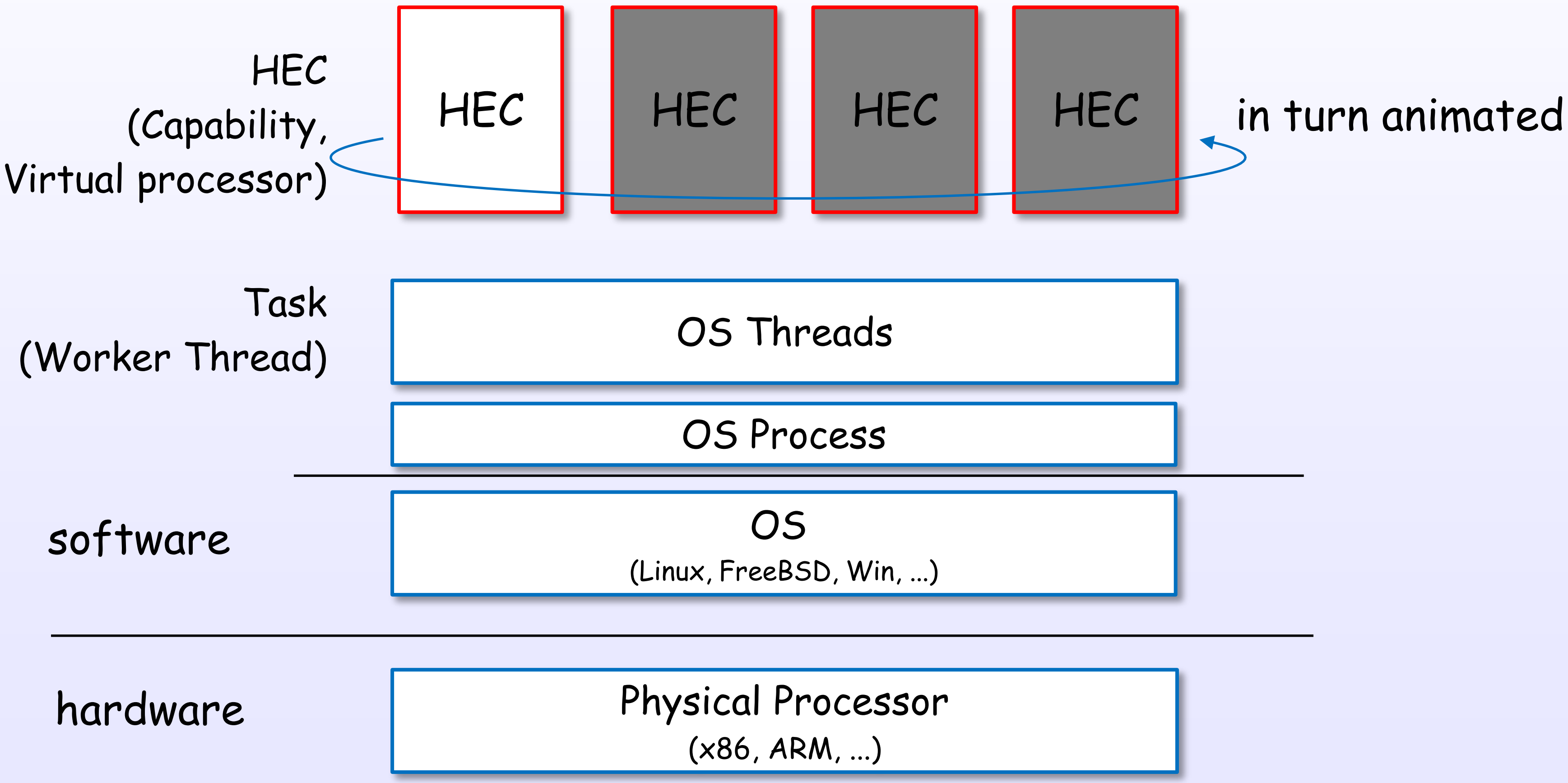


References : [C14], [C6], [1], [C17], [7], [S15], [S16], [S11]

many HECs

multi HEC generated by compile and runtime options :

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



HEC
(Capability,
Virtual processor)

HEC HEC HEC HEC

in turn animated

Task
(Worker Thread)

OS Threads

OS Process

software

OS
(Linux, FreeBSD, Win, ...)

hardware

Physical Processor
(x86, ARM, ...)

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

HEC (Capability) data structure

[rts/Capability.h]

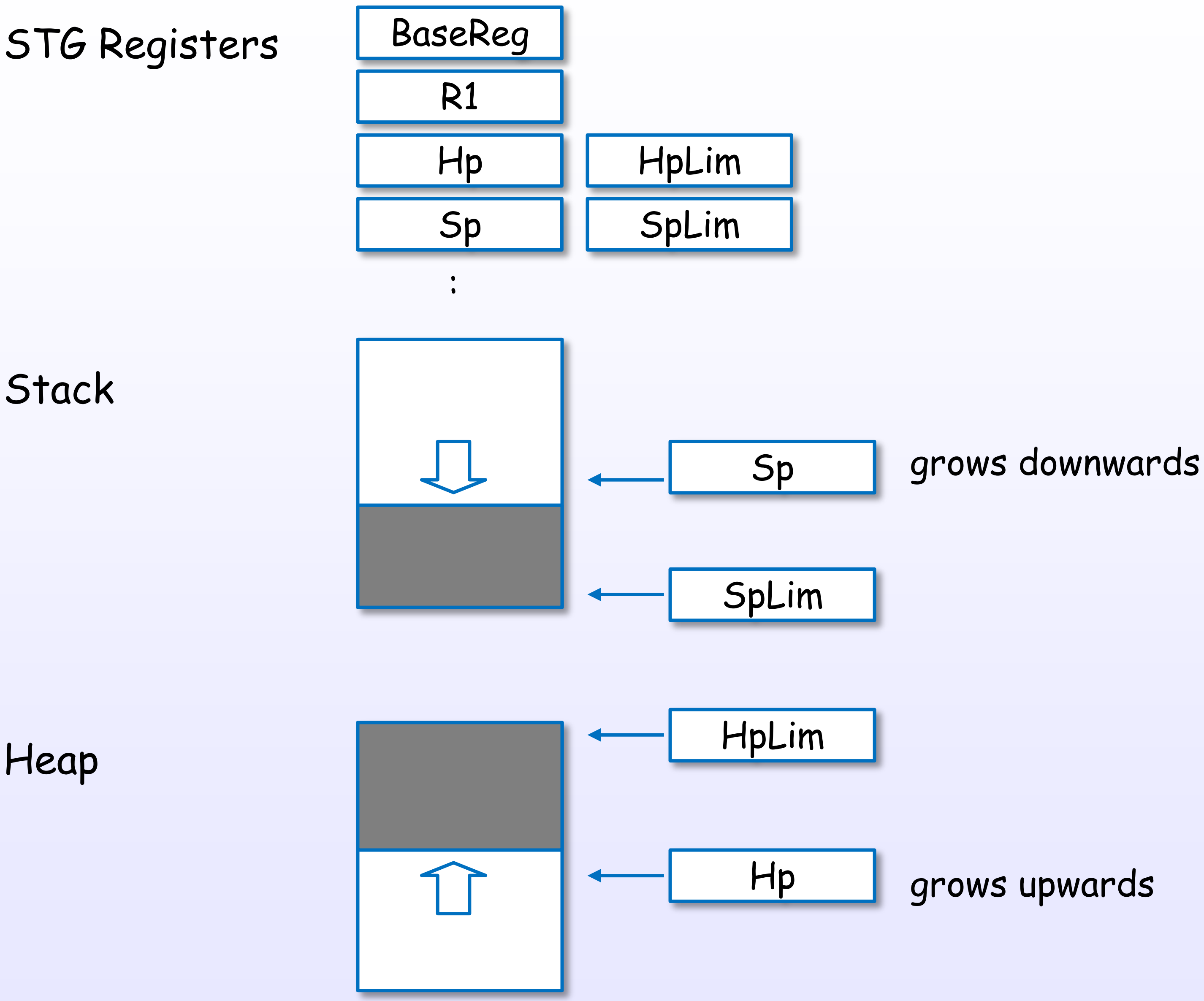
```
struct Capability_ {
    StgFunTable f;
    StgRegTable r;
    nat no;
    Task *running_task;
    rtsBool in_haskell;
    nat idle;
    rtsBool disabled;
    StgTSO *run_queue_hd;
    StgTSO *run_queue_tl;
    InCall *suspended_ccalls;
    bdescr **mut_lists;
    bdescr **saved_mut_lists;
    bdescr *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 (Capabiilty) has Register table and Run queue and ...
HEC (Capability) is initialized in initCapabilities () [rts/rts/Capability.c]

References : [S15], [S16], [C11], [C17]

STG-machine

STG-machine



STG Registers

BaseReg

R1

Hp HpLim

Sp SpLim

:

Stack

Sp grows downwards

SpLim

Heap

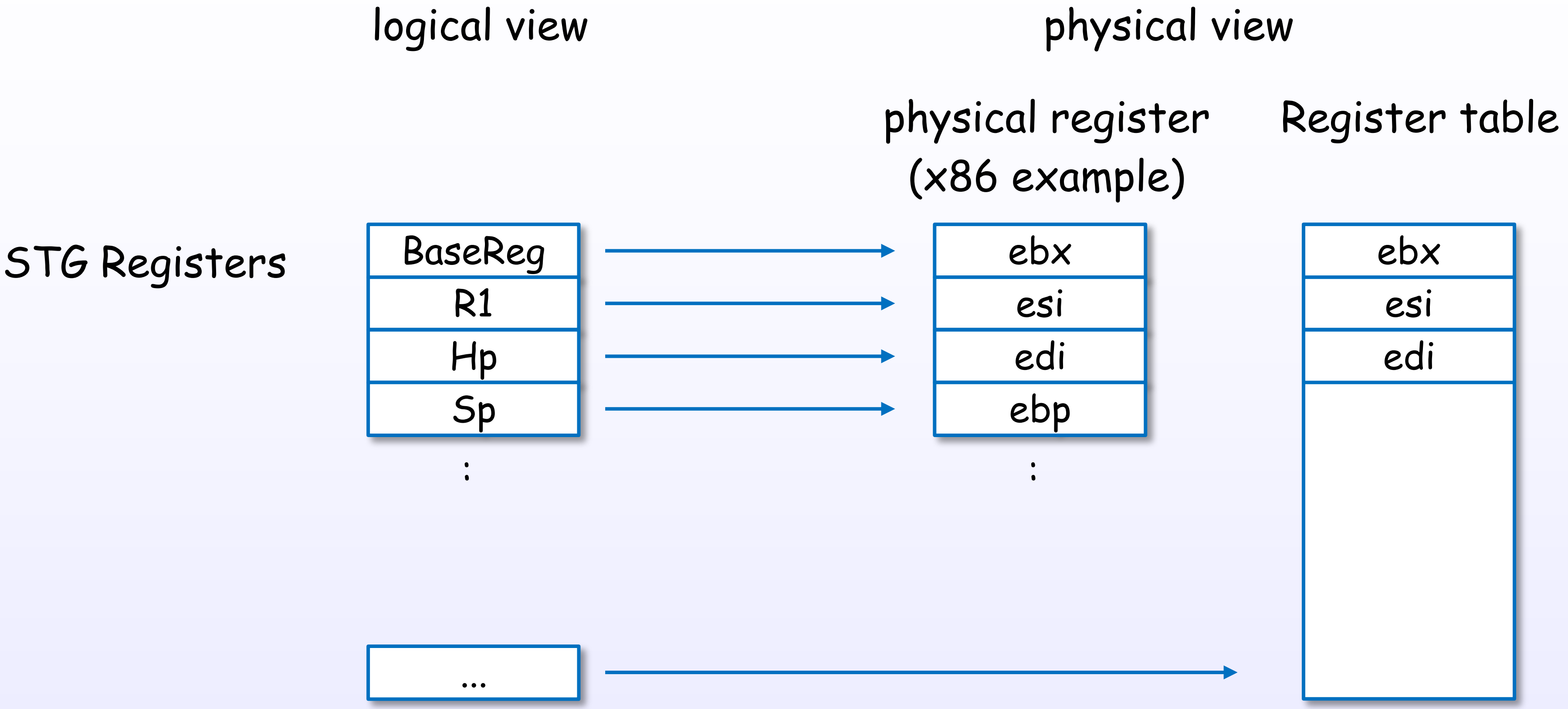
HpLim

Hp grows upwards

Real Haskell code executed in STG semantics.

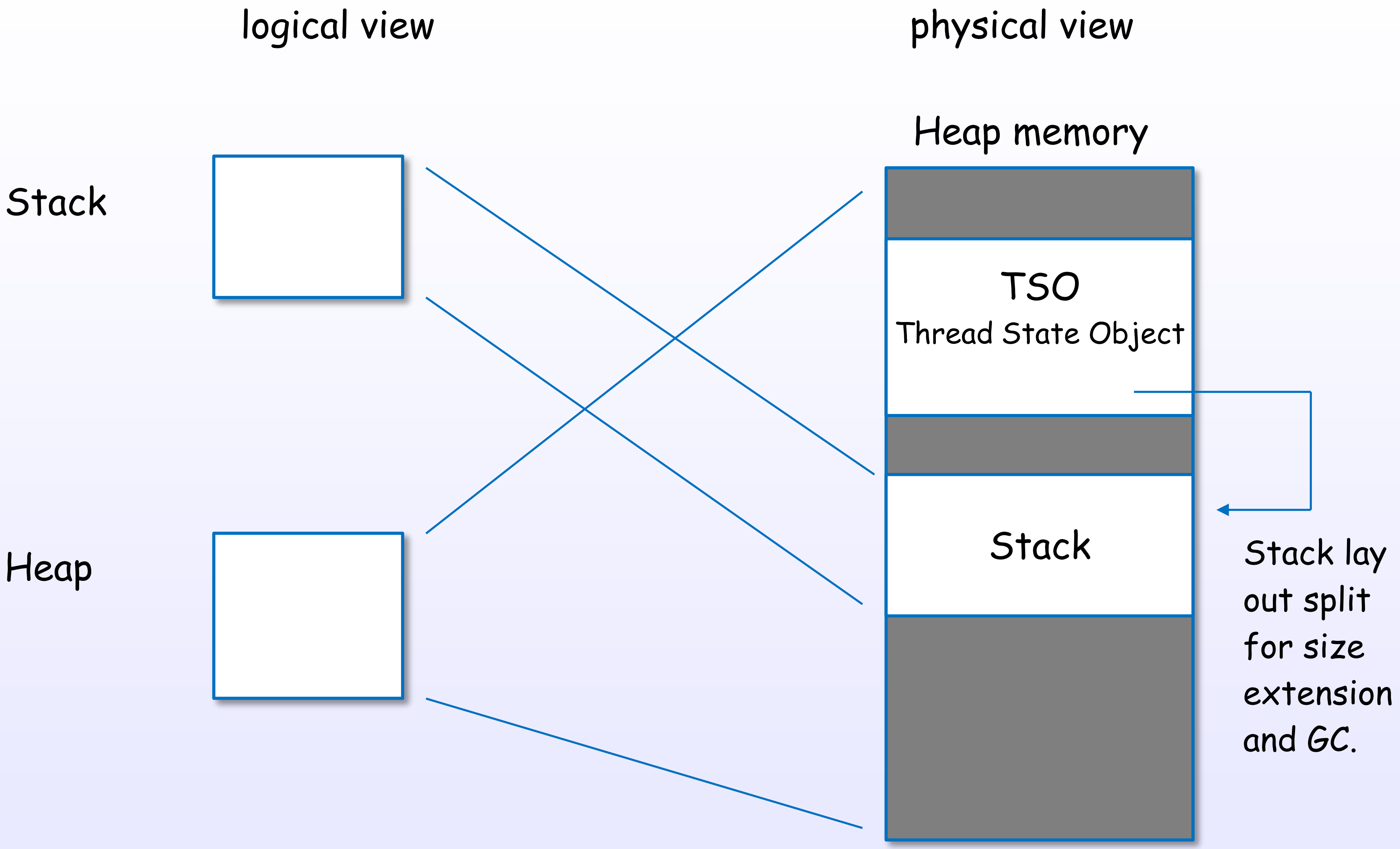
References : [1], [C15], [C11], [C12]

STG-machine mapped to physical processor



References : [C15], [S1], [S2]

STG-machine mapped to physical processor



Stack is in TSO object.
TSO object in heap.

References : [C11], [C12], [S16], [S5]

TSO object

[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;
```

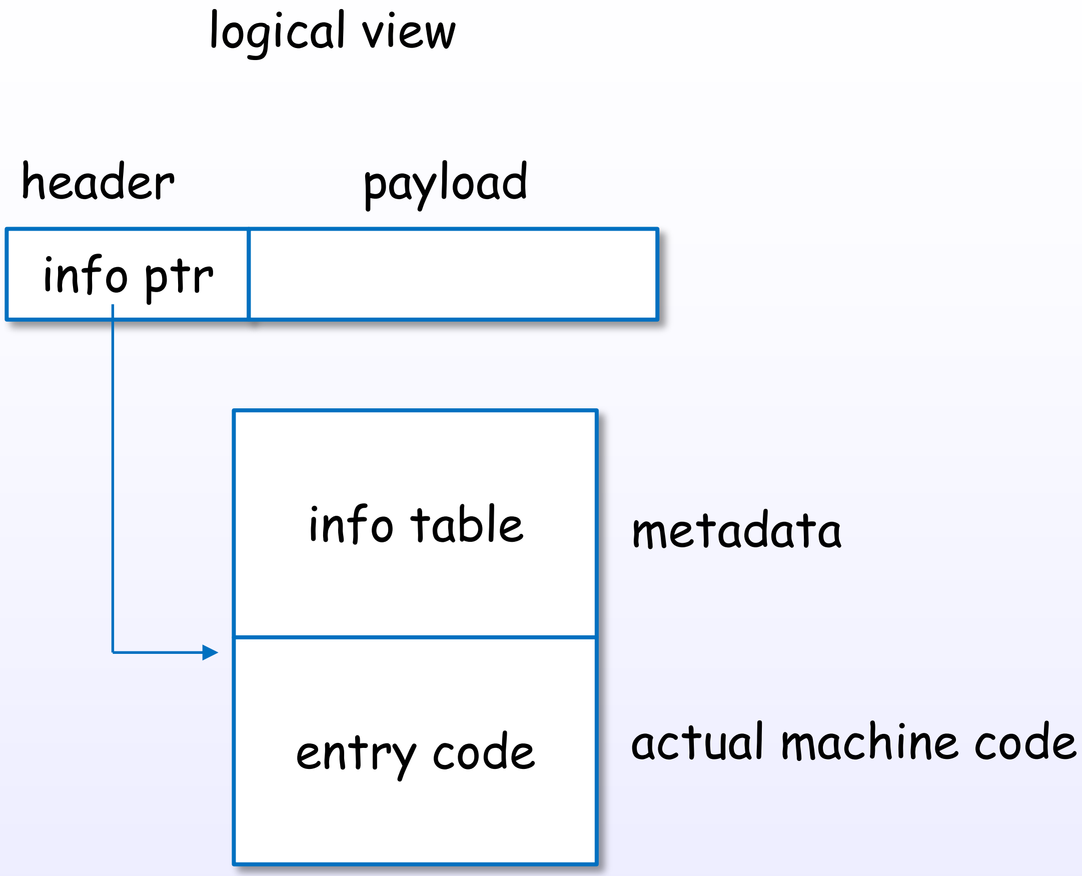
link to stack object

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

References : [S5]

Heap object in STG-machine

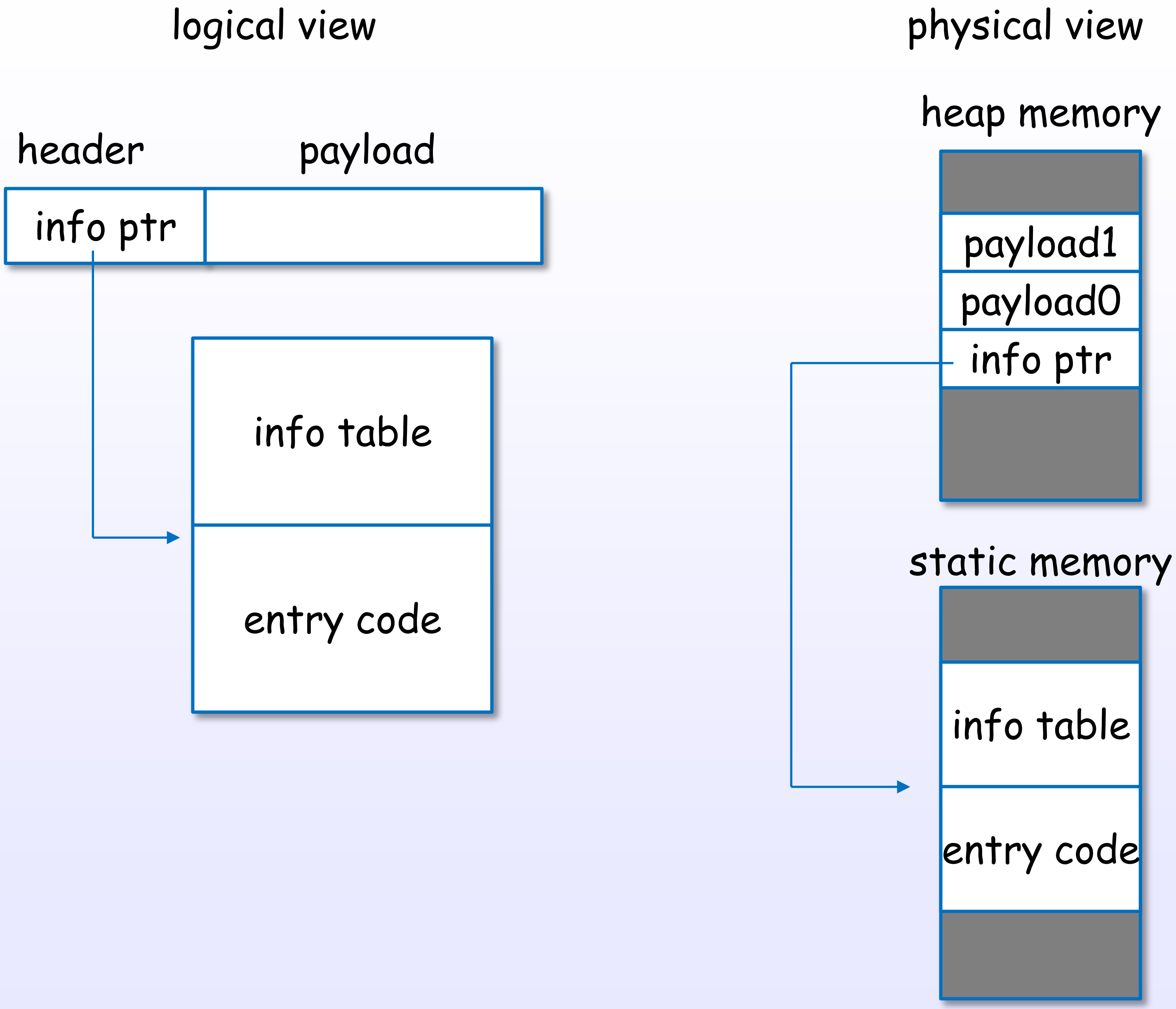
Heap object (closure)



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

References : [C11], [S3], [S4], [S6], [1]

Heap object (closure)



logical view

header

payload

info ptr

info table

entry code

physical view

heap memory

payload1

payload0

info ptr

static memory

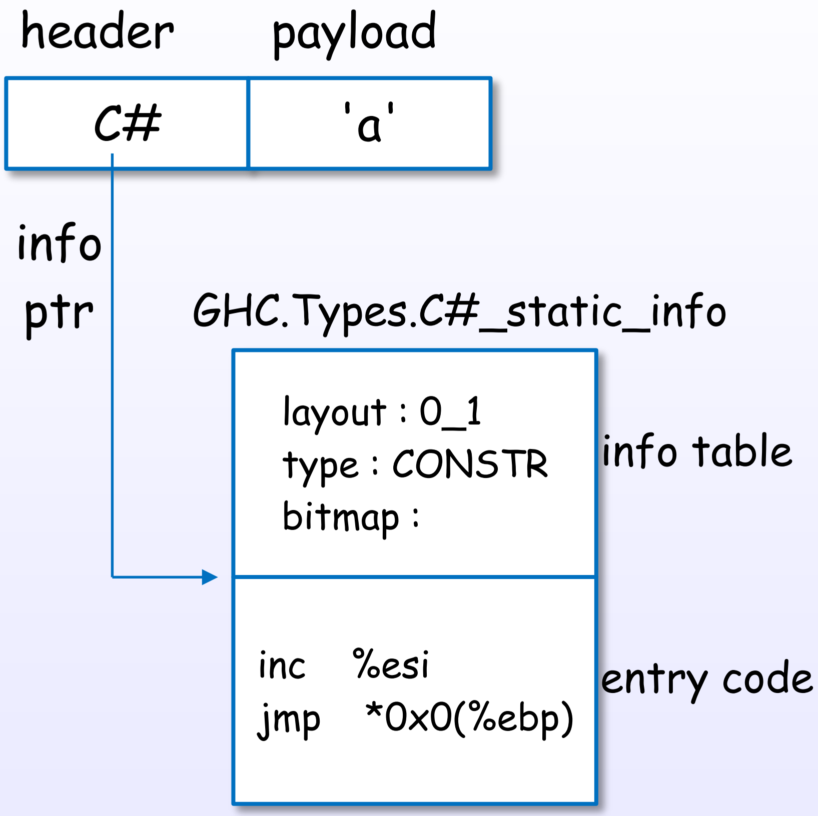
info table

entry code

References : [C11], [S3], [C9], [C8], [1]

Closure examples : Char, Int

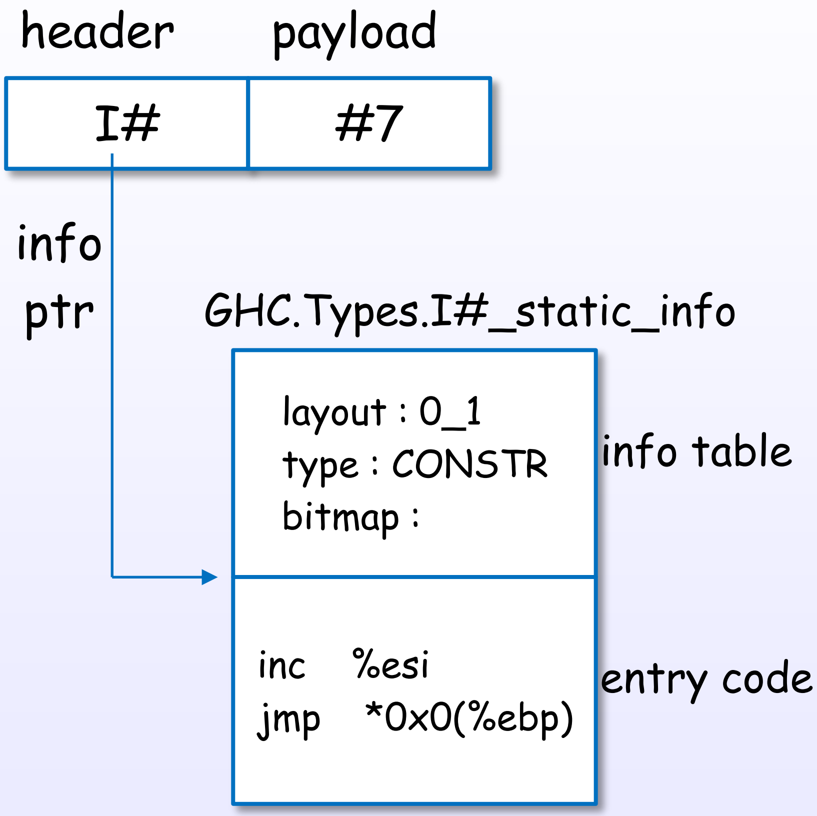
'a' :: Char



layout : 0_1
type : CONSTR
bitmap :

```
inc    %esi
jmp    *0x0(%ebp)
```

7 :: Int



layout : 0_1
type : CONSTR
bitmap :

```
inc    %esi
jmp    *0x0(%ebp)
```

References : [C11], [S3], [C9], [C8], [1], [S20]

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

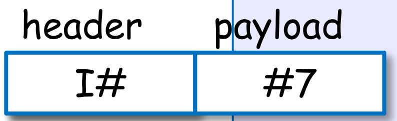
[ghc -O -ddump-opt-cmm Example.hs]

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

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

section "relreadonly" { SMc_srt:
}
```

asm

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

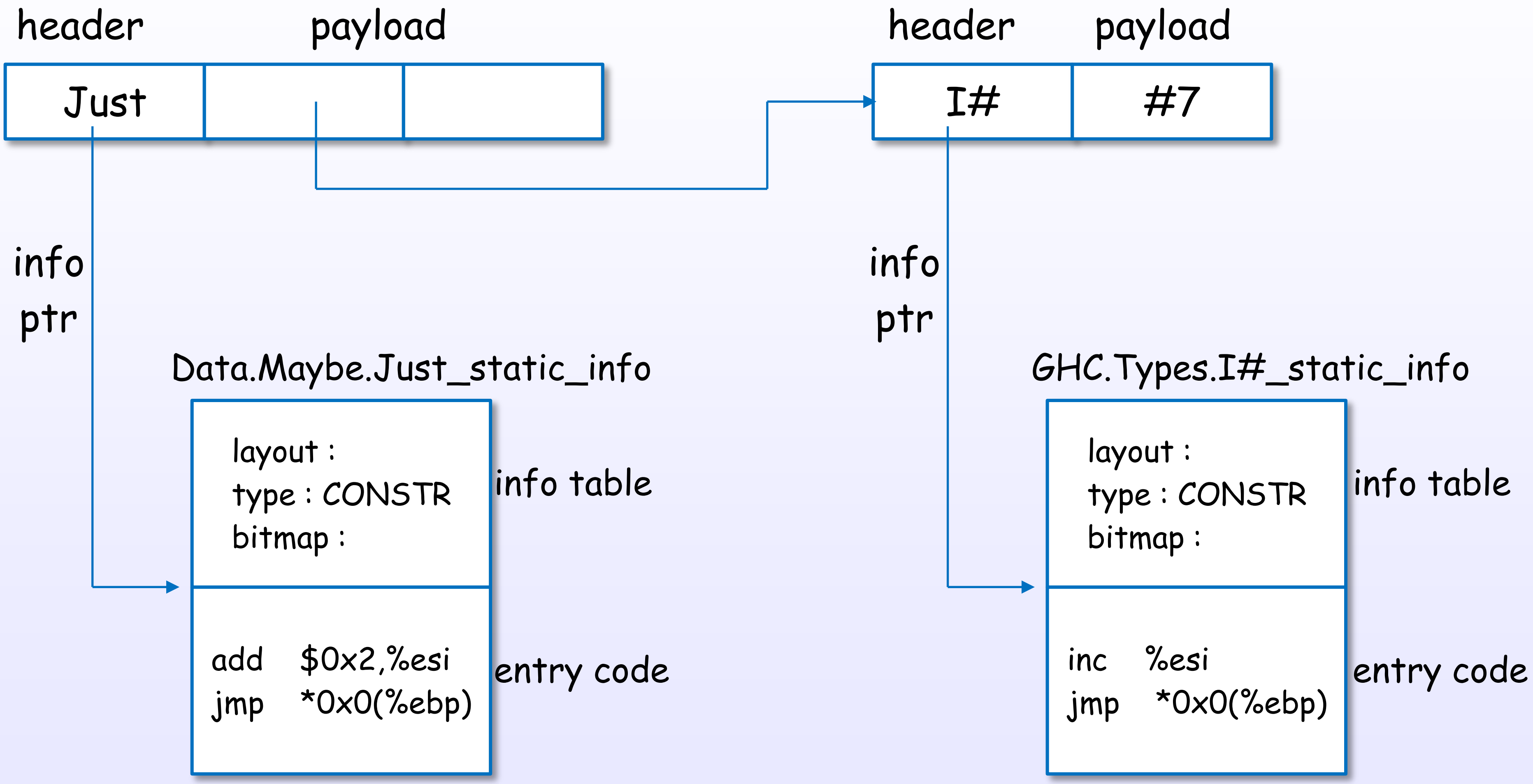
```
.data
        .align 4
.align 1
.globl __stginit_main:Example
__stginit_main:Example:
.data
        .align 4
.align 1
.globl Example.value1_closure
Example.value1_closure:
        .long   GHC.Types.I#_static_info
        .long   7
.section .data
        .align 4
.align 1
SMd_srt:
```



References : [C11], [S3], [C9], [C8], [1], [S20]

Closure examples : Maybe

Just 7 :: Maybe Int



layout :
type : CONSTR
bitmap :

```
add   $0x2,%esi
jmp   *0x0(%ebp)
```

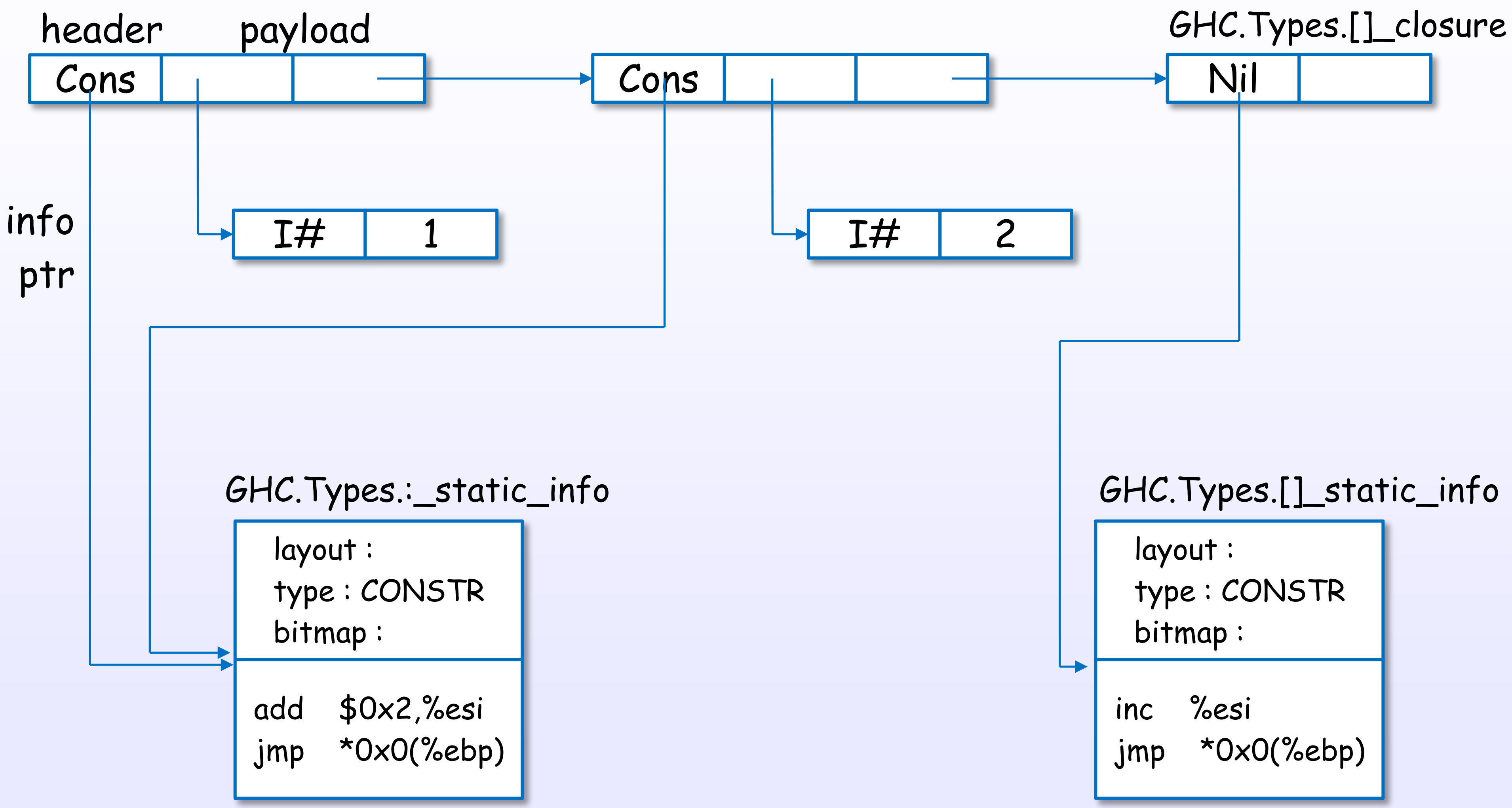
layout :
type : CONSTR
bitmap :

```
inc   %esi
jmp    *0x0(%ebp)
```

References : [C11], [S3], [C9], [C8], [1], [S20]

Closure examples : List

[1, 2] :: [Int]



GHC.Types.:_static_info

layout :
type : CONSTR
bitmap :

```
add   $0x2,%esi
jmp   *0x0(%ebp)
```

GHC.Types.[]_static_info

layout :
type : CONSTR
bitmap :

```
inc   %esi
jmp    *0x0(%ebp)
```

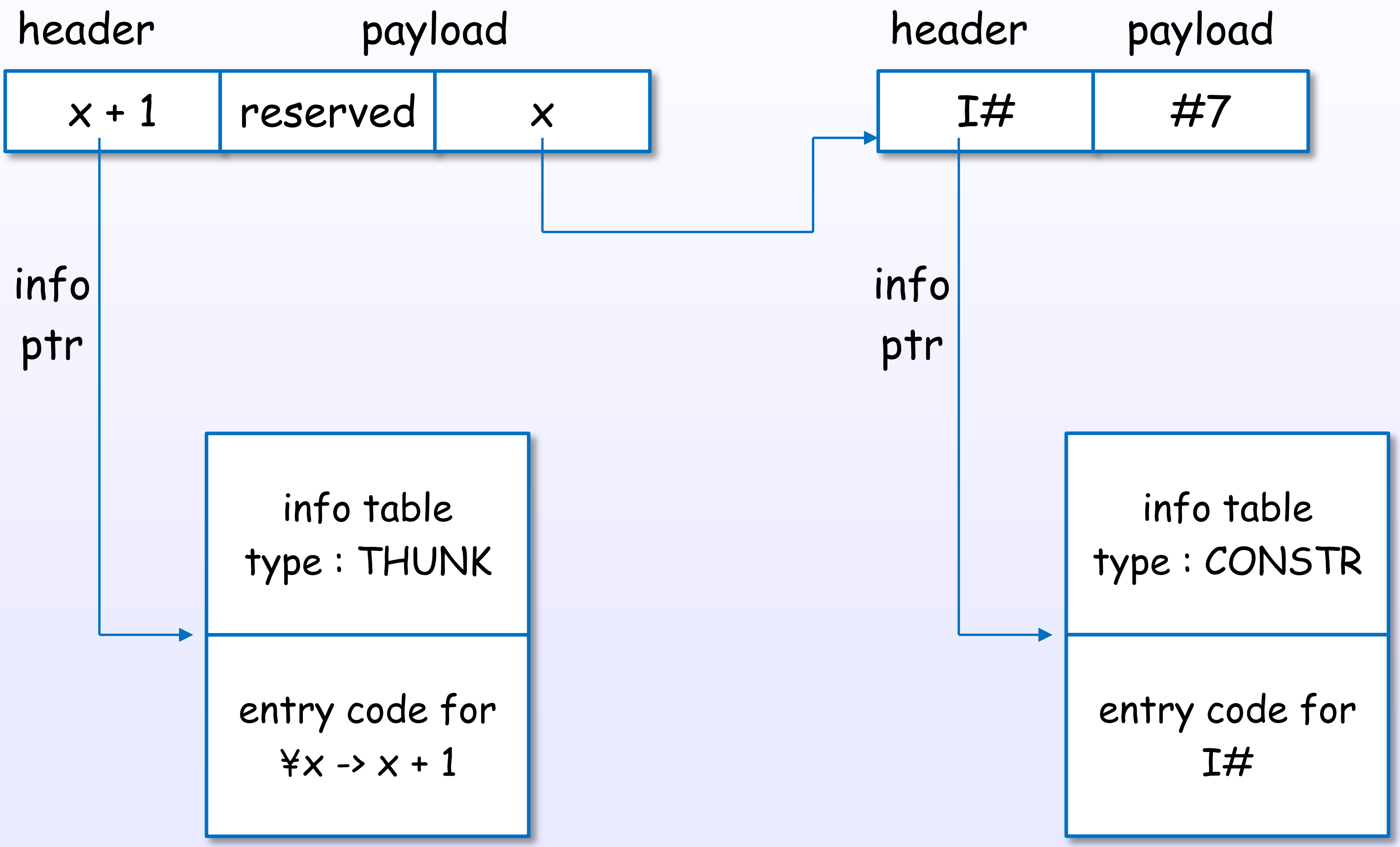
References : [C11], [S3], [C9], [C8], [1], [S20]

Closure examples : Thunk

"thunk"

x + 1 :: Int

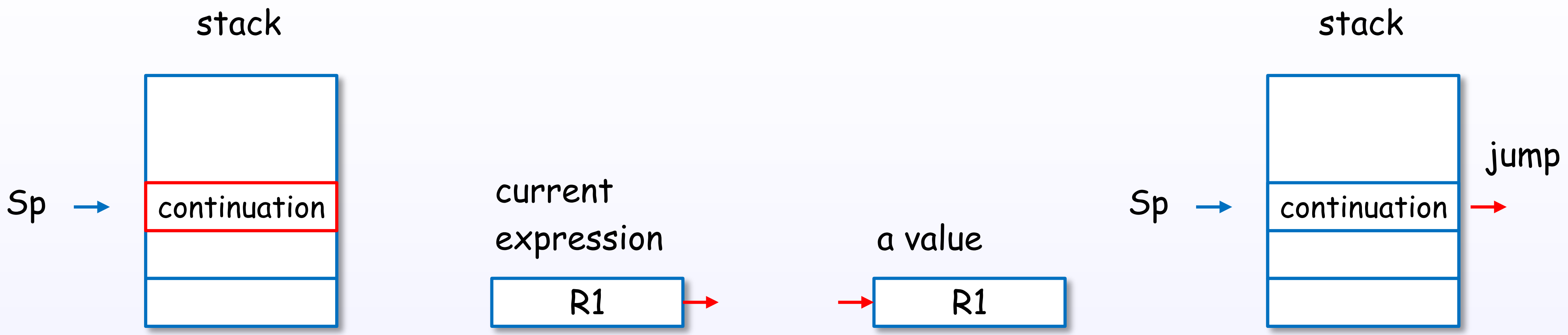
(free variable : x = 7)



References : [C11], [S3], [C9], [C8], [1], [S20]

STG-machine evaluation

STG evaluation flow



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

(2)
enter to R1 closure

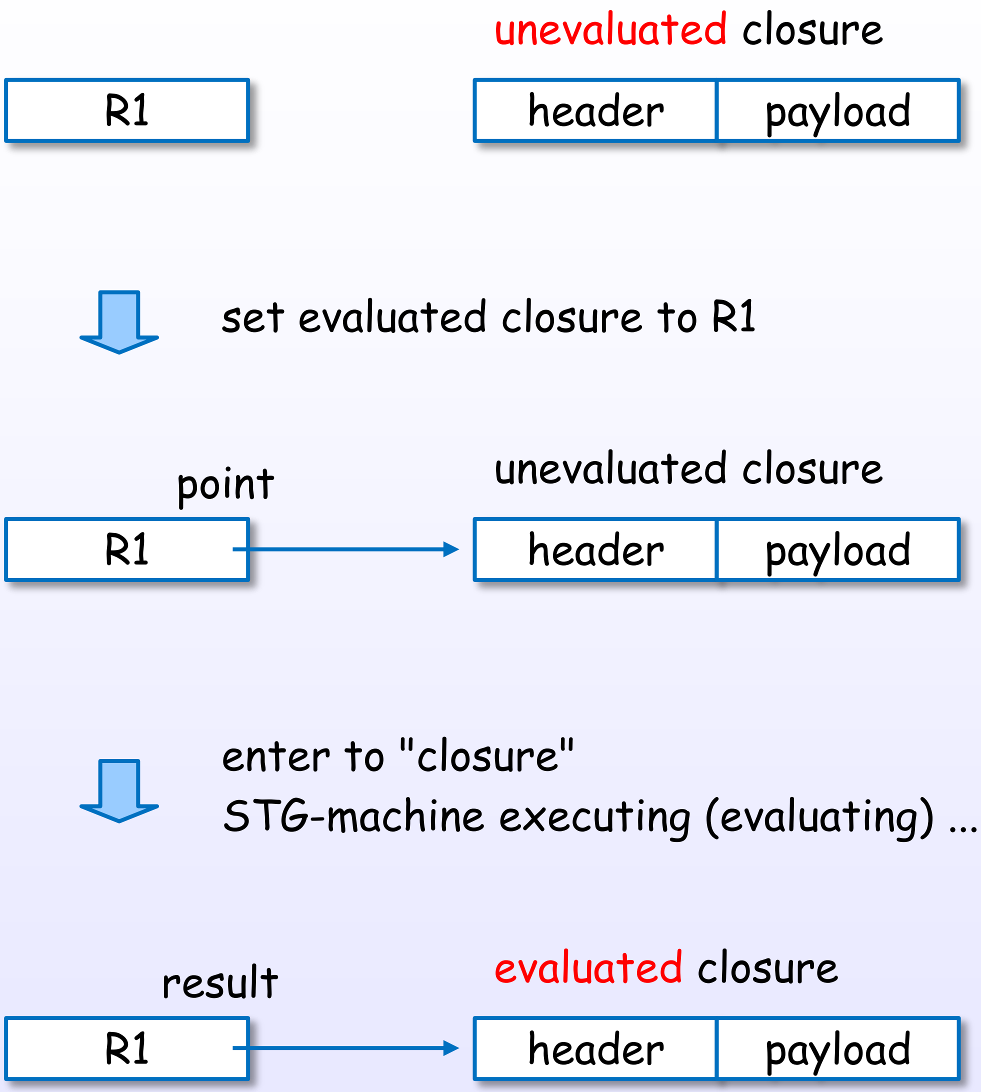
(3)
set result to R1

(4)
jump (return)
to stack top code

(5)
repeat from (1)

References : [C8], [2]

Evaluation is "enter to closure"



unevaluated closure

R1

header | payload

set evaluated closure to R1

point

unevaluated closure

R1

header | payload

enter to "closure"

STG-machine executing (evaluating) ...

stg_ap_v, ...

result

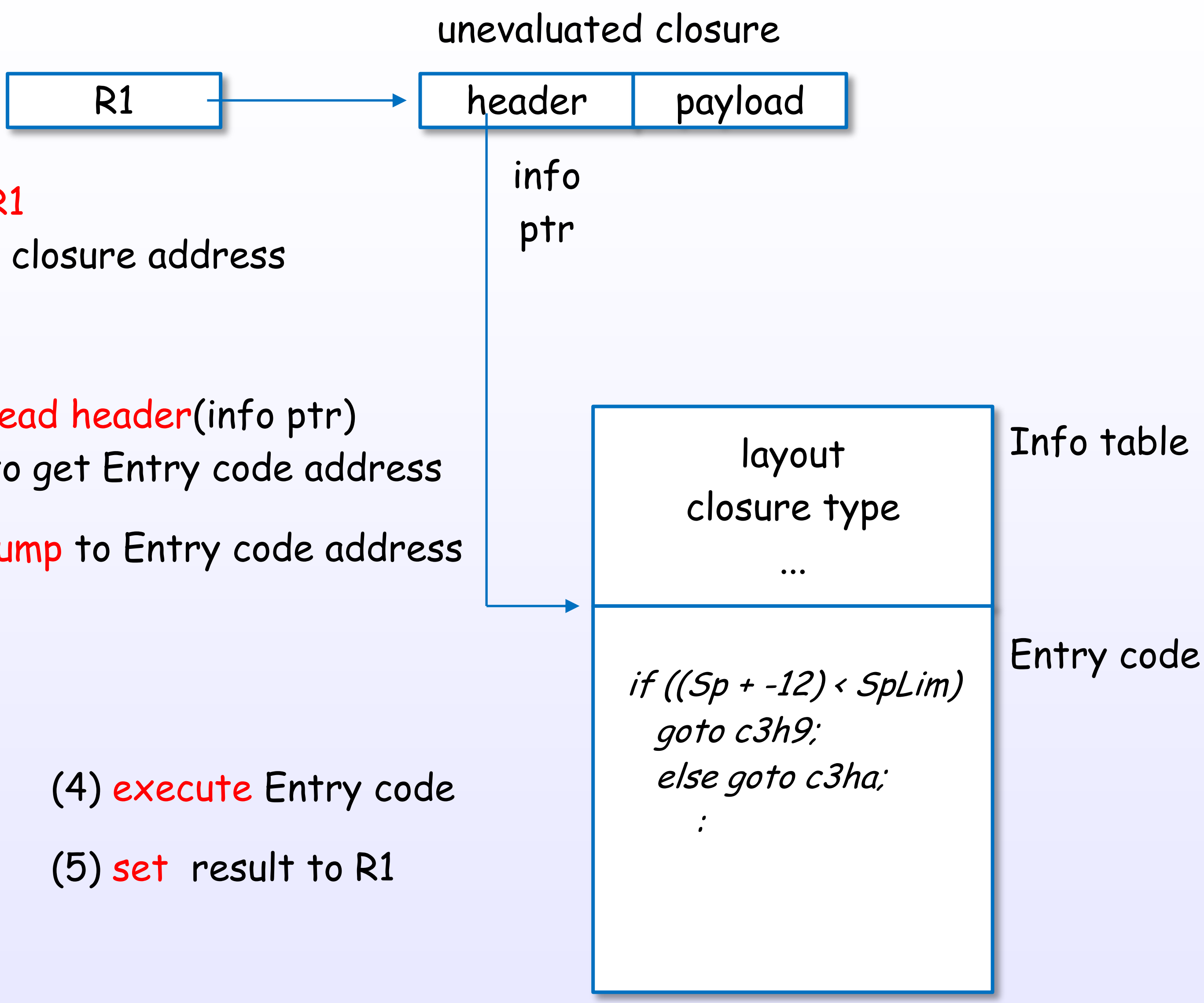
evaluated closure

R1

header | payload

References : [C11], [C9], [C8], [9], [2], [1], [11]

Enter to closure



```
if ((Sp + -12) < SpLim)
  goto c3h9;
  else goto c3ha;
        :
```

(1) read R1
to get closure address

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

(3) jump to Entry code address

(4) execute Entry code

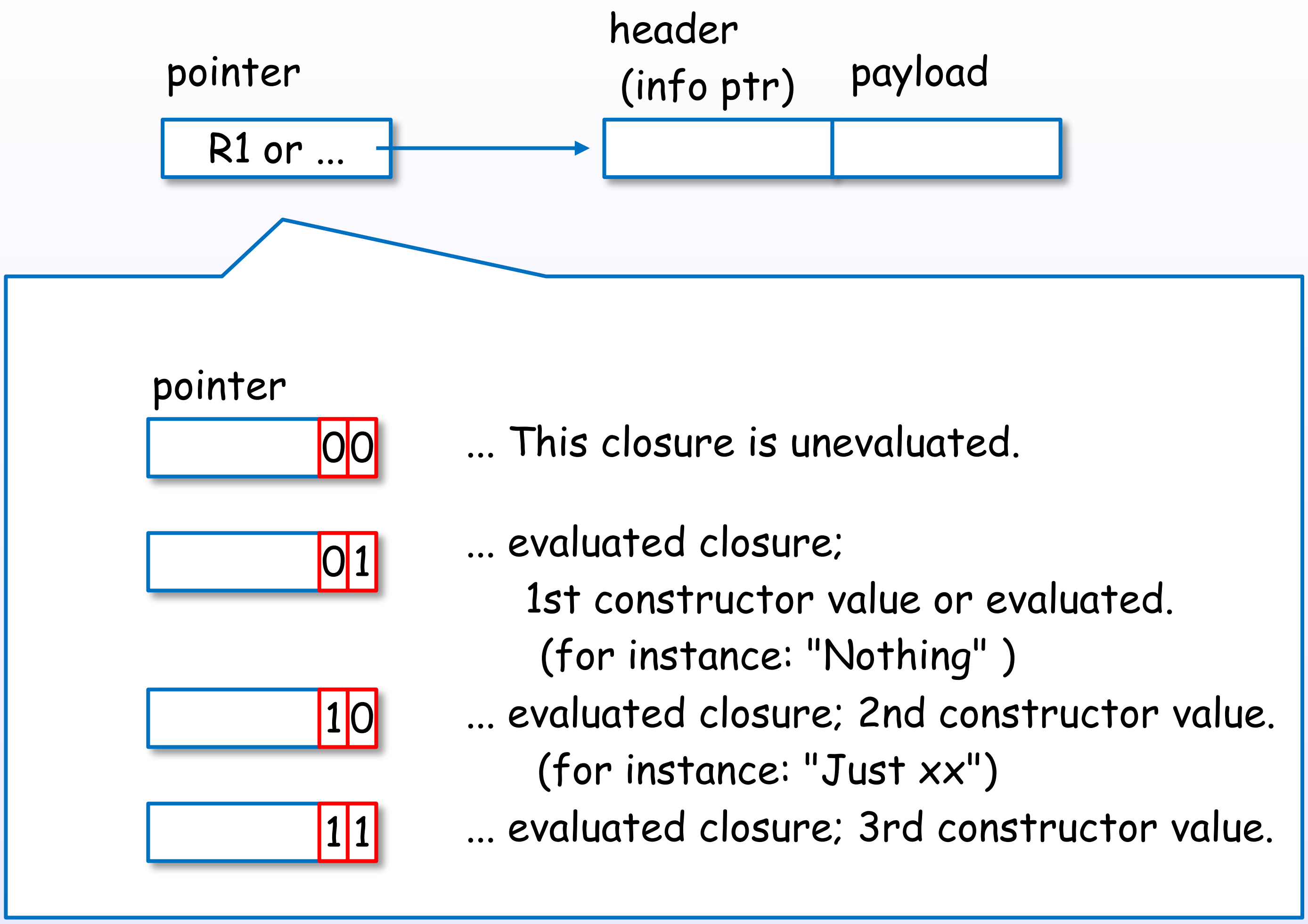
(5) set result to R1

(6) jump to stack top address
(continuation)

References : [C11], [C9], [C8], [9], [2], [1], [11]

Pointer tagging

Pointer tagging



pointer

R1 or ...

header
(info ptr)

payload

pointer

00 ... This closure is unevaluated.

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

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

11 ... evaluated closure; 3rd constructor value.

* 32bit machine case

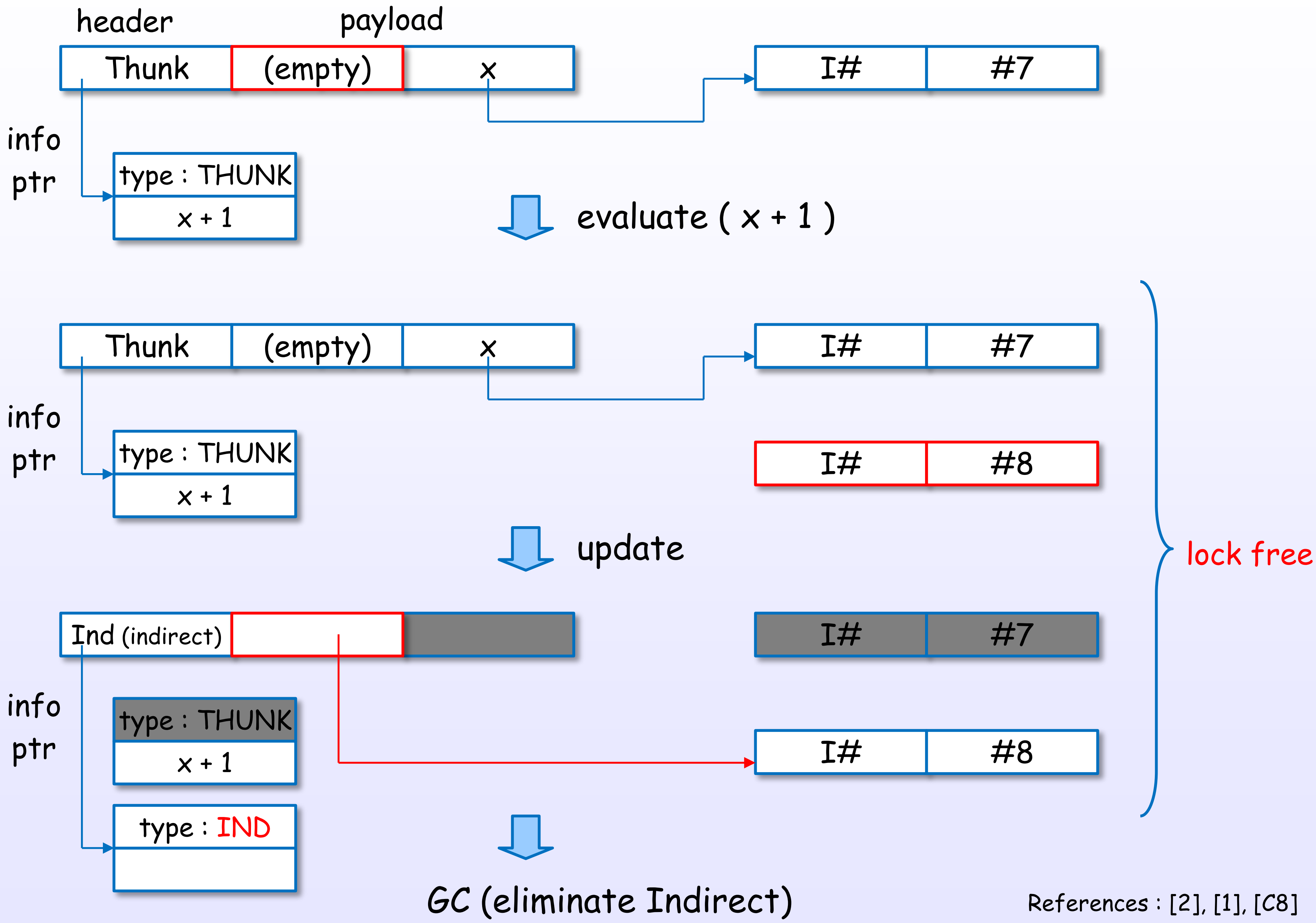
quick judgment!
check only pointer's lower bits without evaluating the closure.

References : [3], [1], [C16]

Thunk and update

Thunk and update

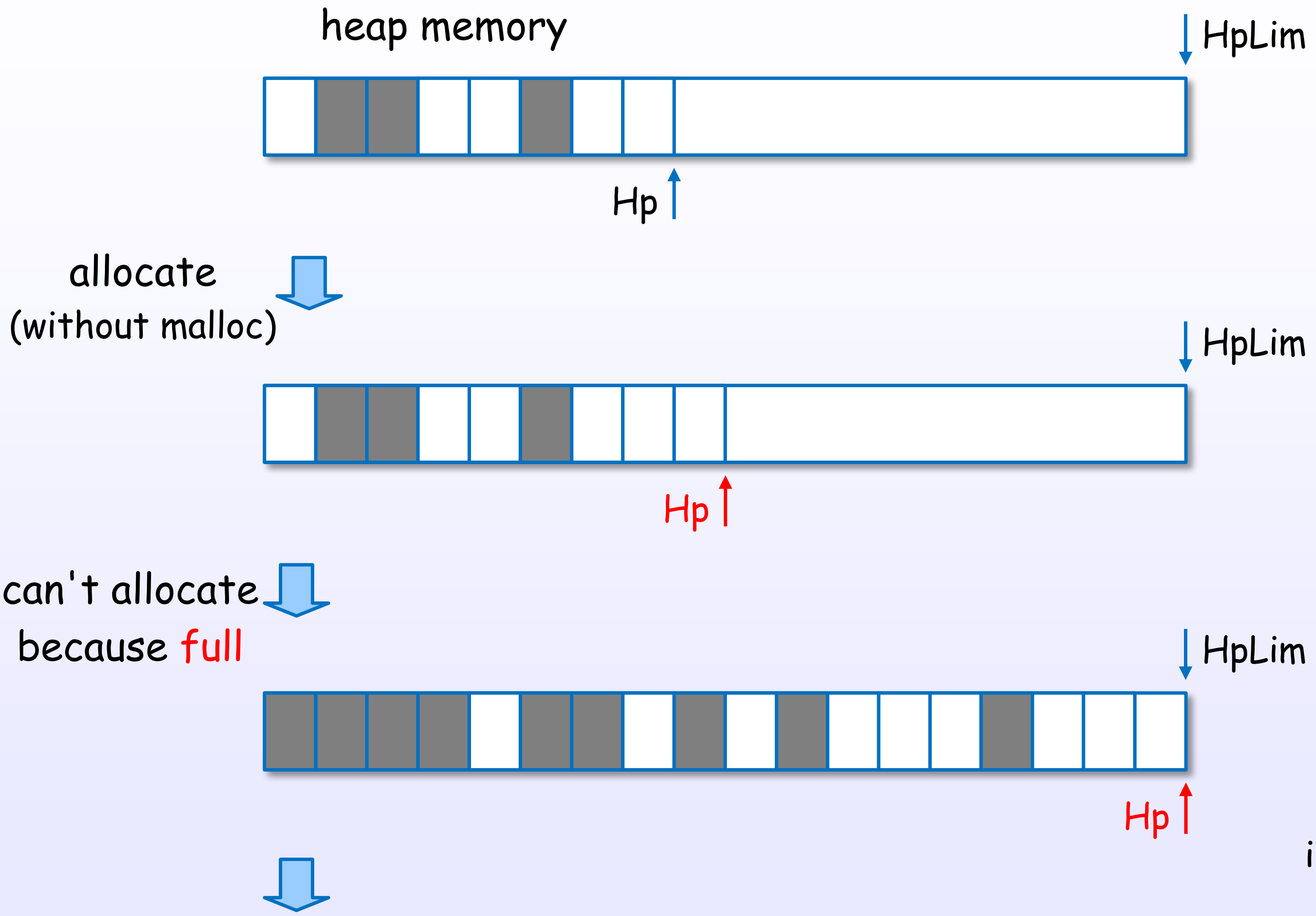
"thunk" x + 1 :: Int (free variable : x = 7)



References : [2], [1], [C8]

Allocate and free(GC) heap objects

Allocate heap objects



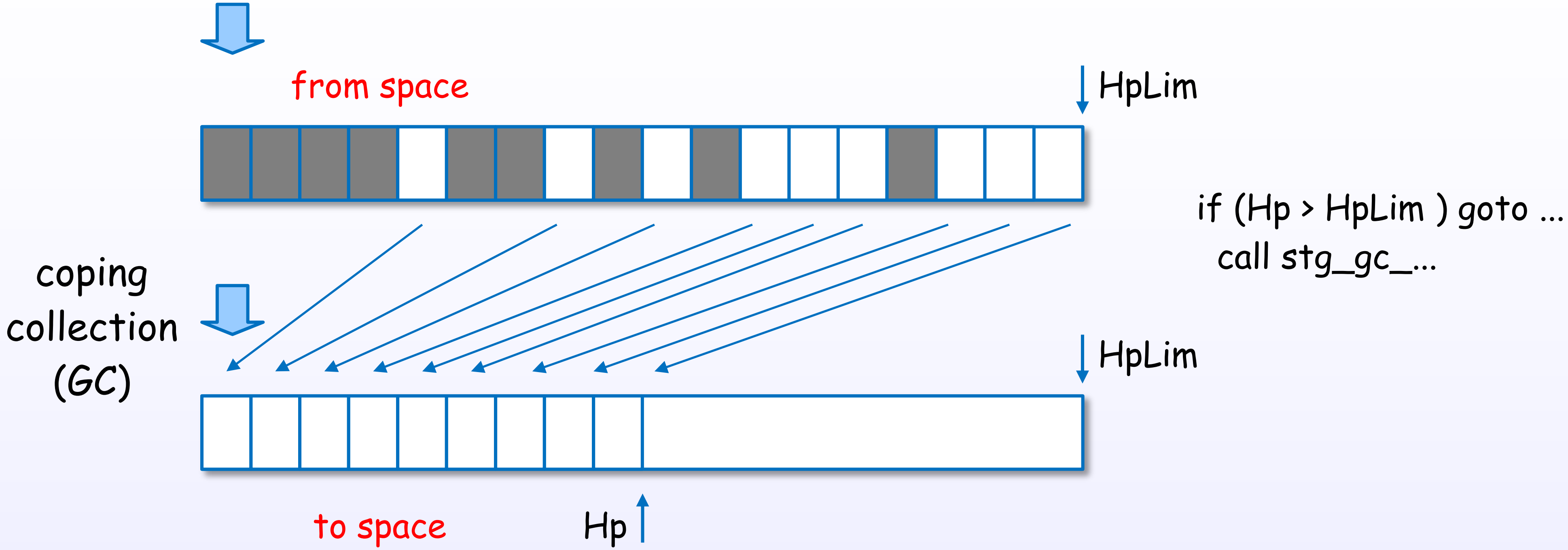
allocate
(without malloc)

can't allocate
because full

```
if (Hp > HpLim ) goto ...
  call stg_gc_...
```

References : [C11], [C13], [7], [8], [4], [14], [11], [12], [18], [S25]

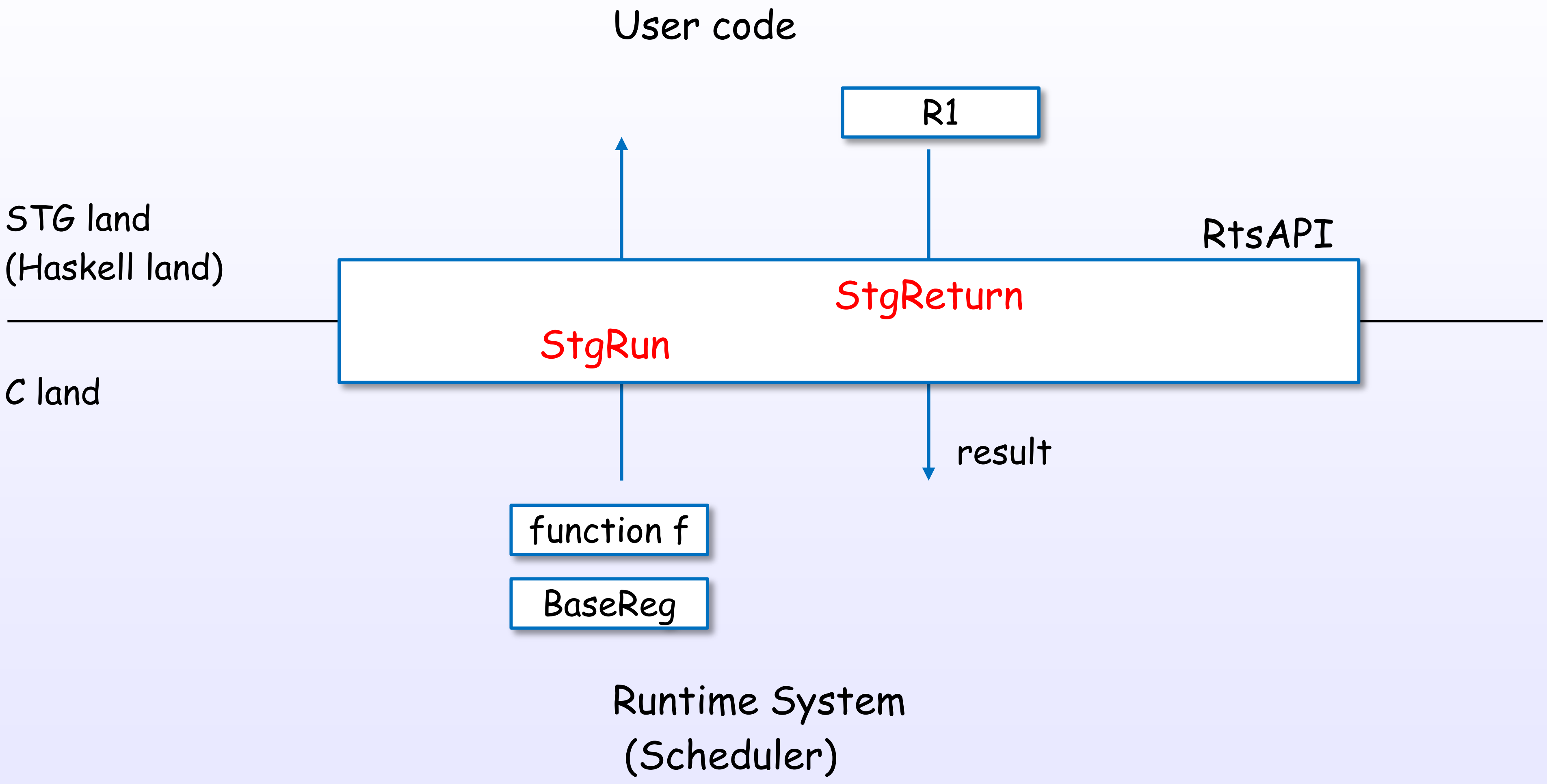
free and collection heap objects



References : [C11], [C13], [7], [8], [4], [14], [11], [12], [18], [S25]

STG - C land interface

STG (Haskell) land - C land interface



User code

R1

STG land
(Haskell land)

RtsAPI

StgReturn

StgRun

C land

result

function f

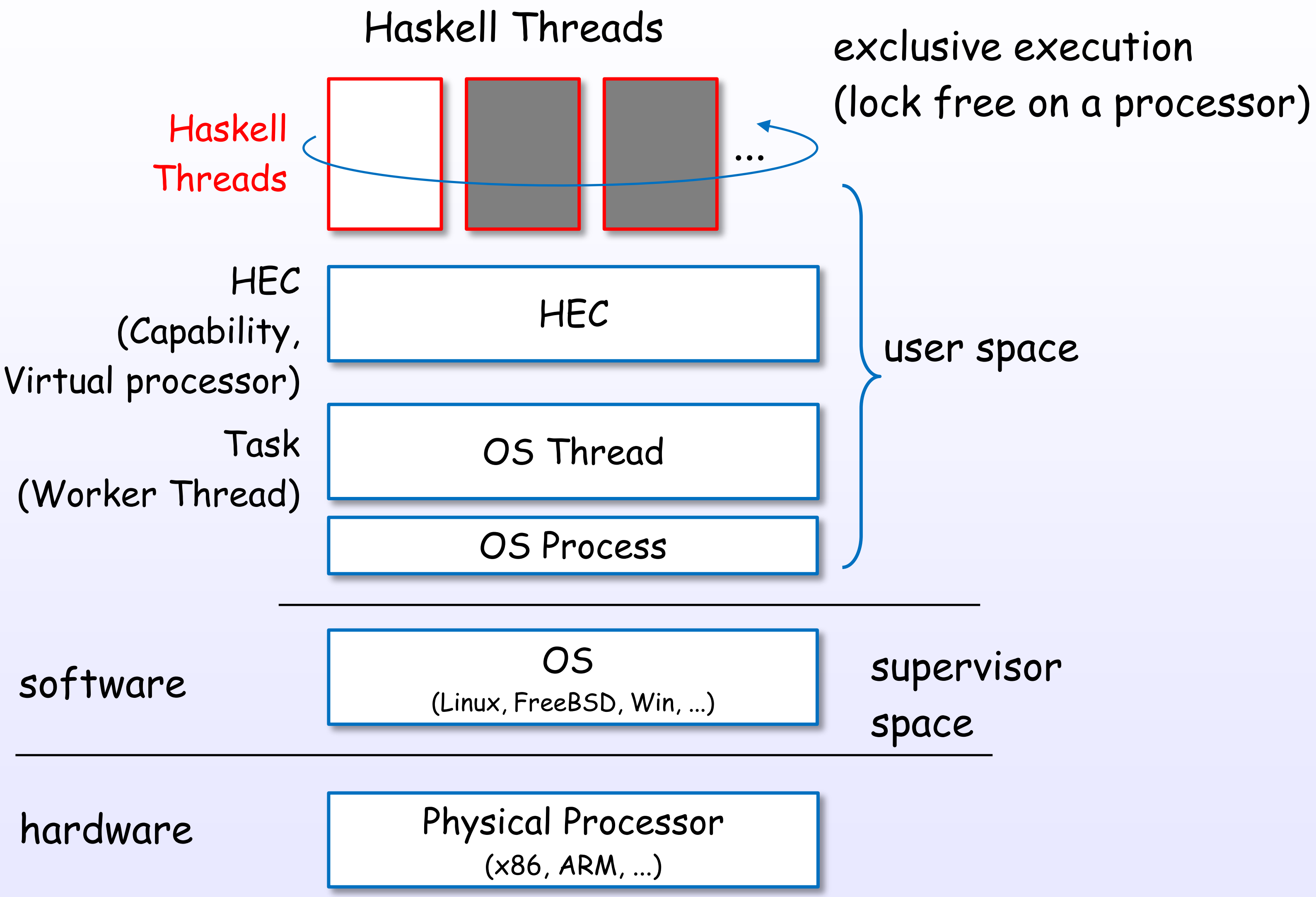
BaseReg

Runtime System
(Scheduler)

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

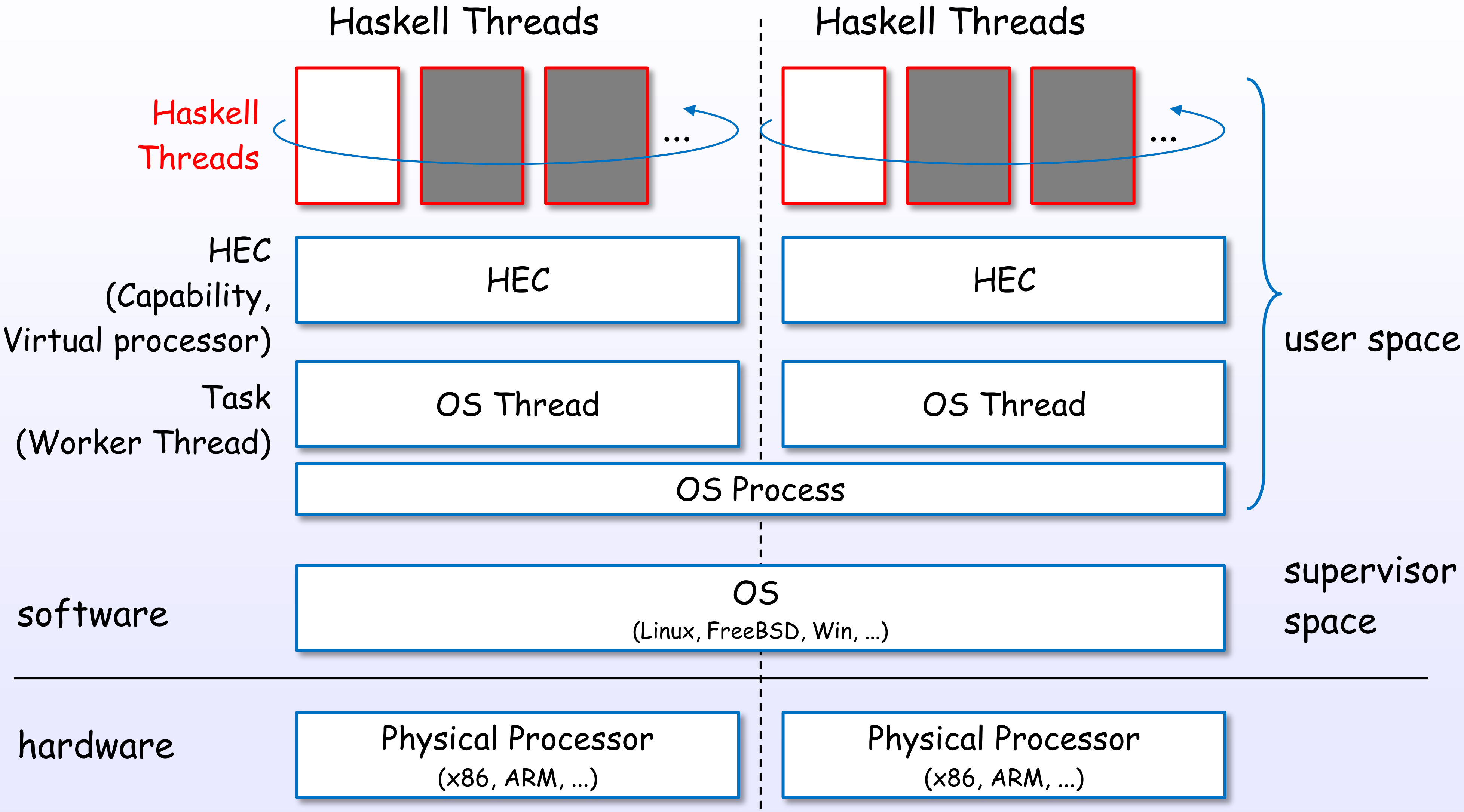
Thread

Thread layer (single core)



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

Thread layer (multi core)

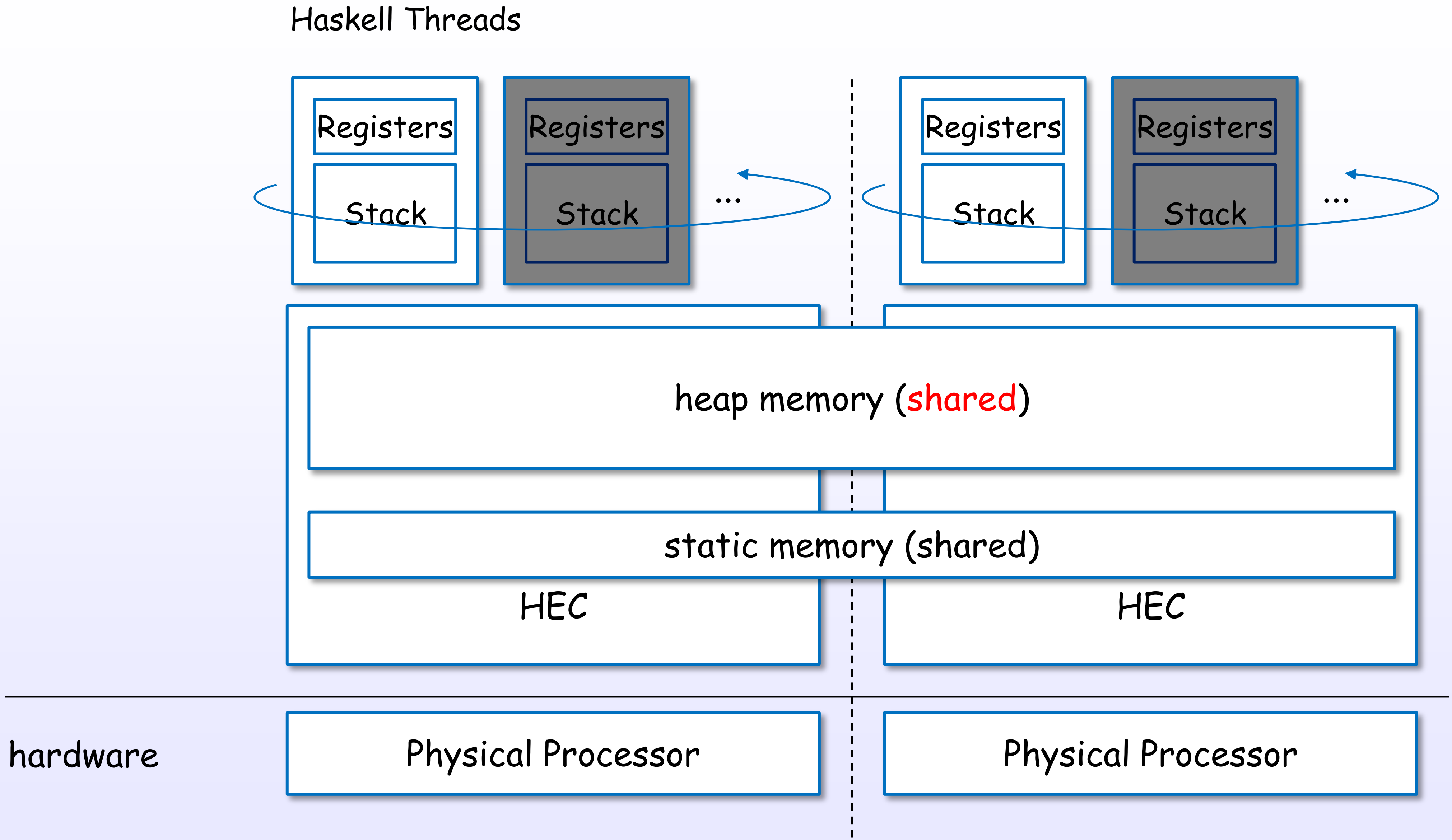


*Threaded option case (ghc -threaded)

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

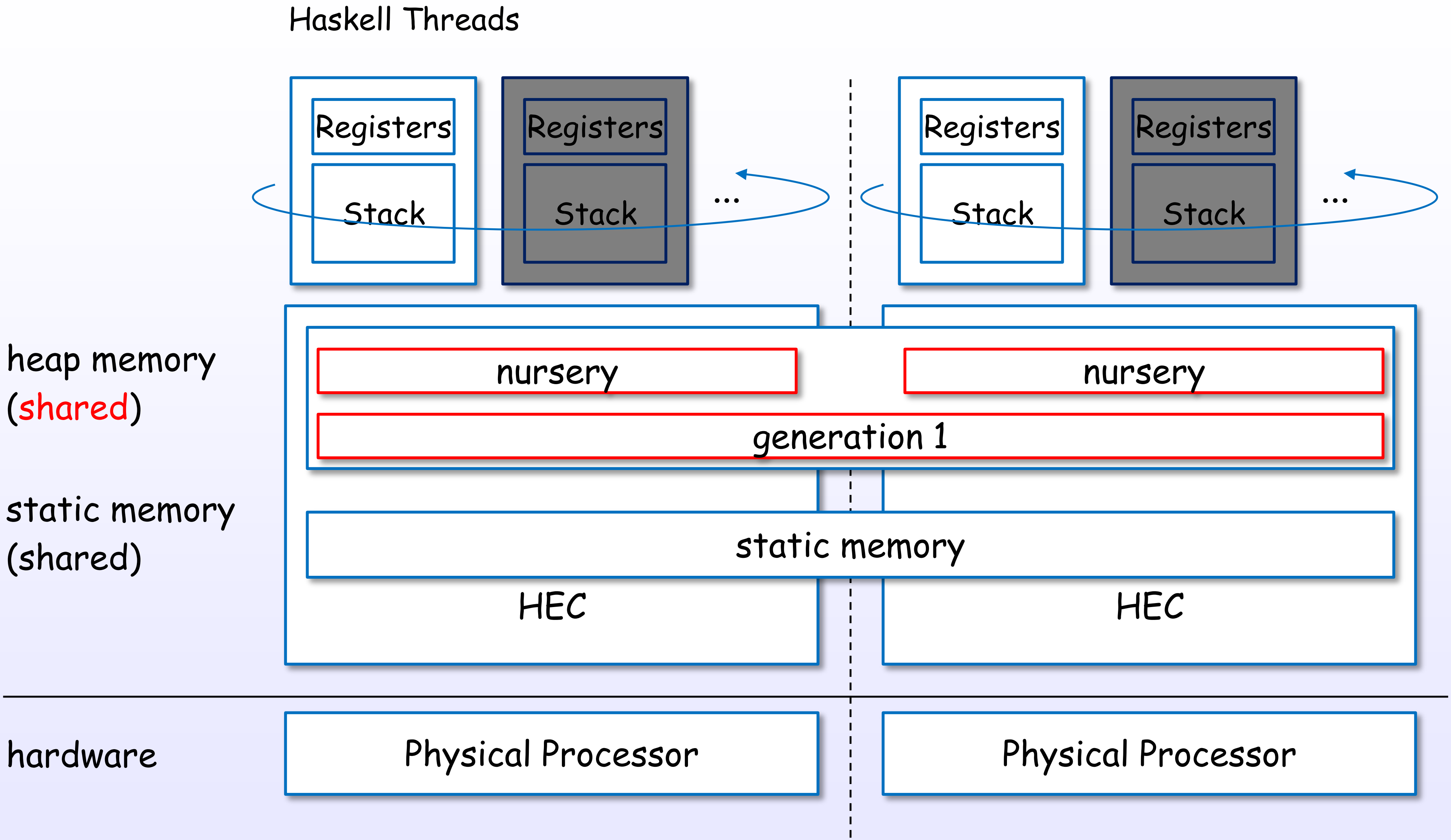
Heap and Threads

Threads and shared heap



References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14], [S17], [S16], [S25]

Local heap area



Haskell Threads

Registers

Stack

Registers

Stack

...

Registers

Stack

Registers

Stack

...

heap memory (shared)

nursery

nursery

generation 1

static memory (shared)

static memory

HEC

HEC

hardware

Physical Processor

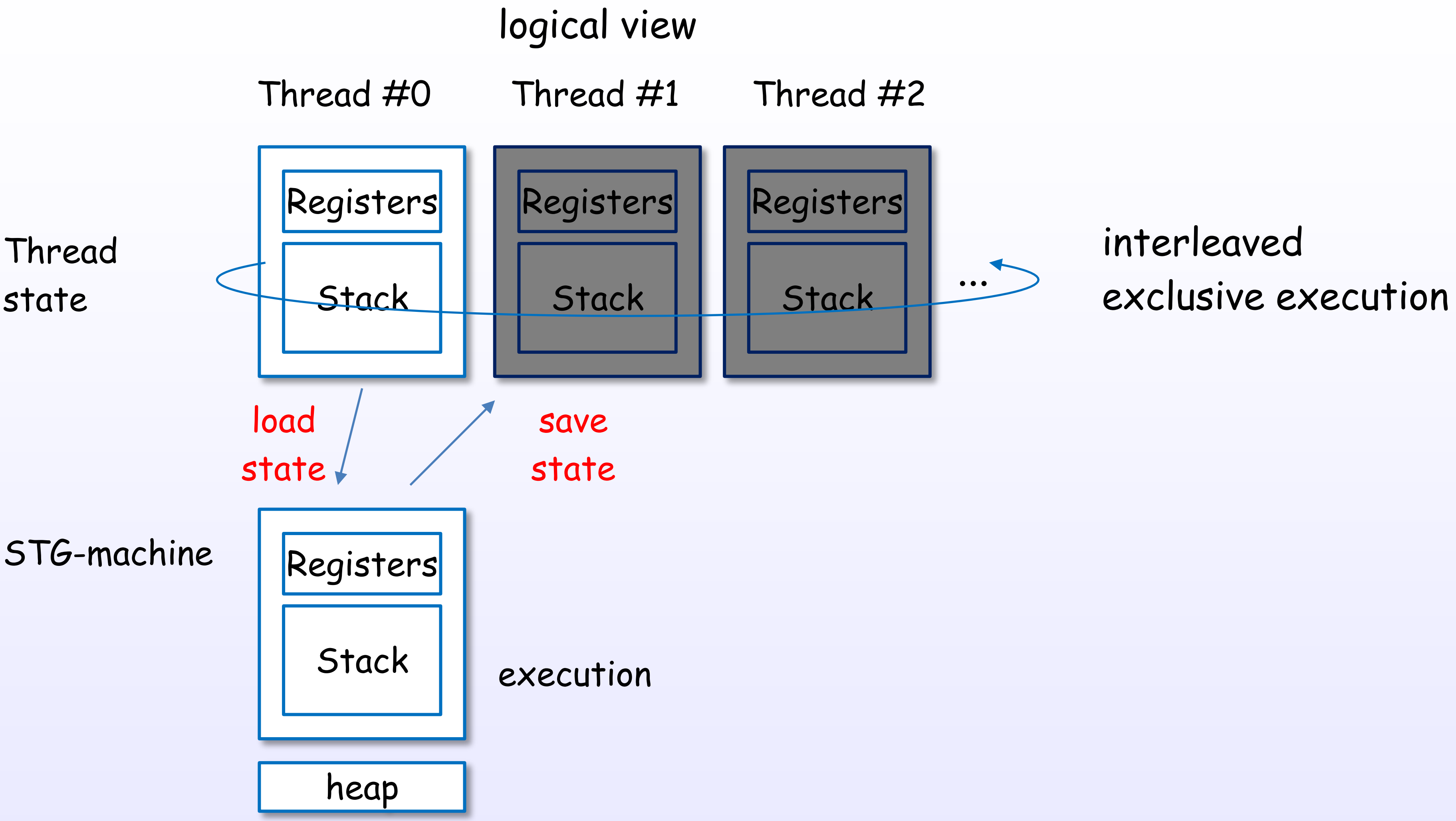
Physical Processor

Fast access on each processor

References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14], [S17], [S16], [S25]

Thread context switch

Threads and context switch



logical view

Thread #0 Thread #1 Thread #2

Thread state

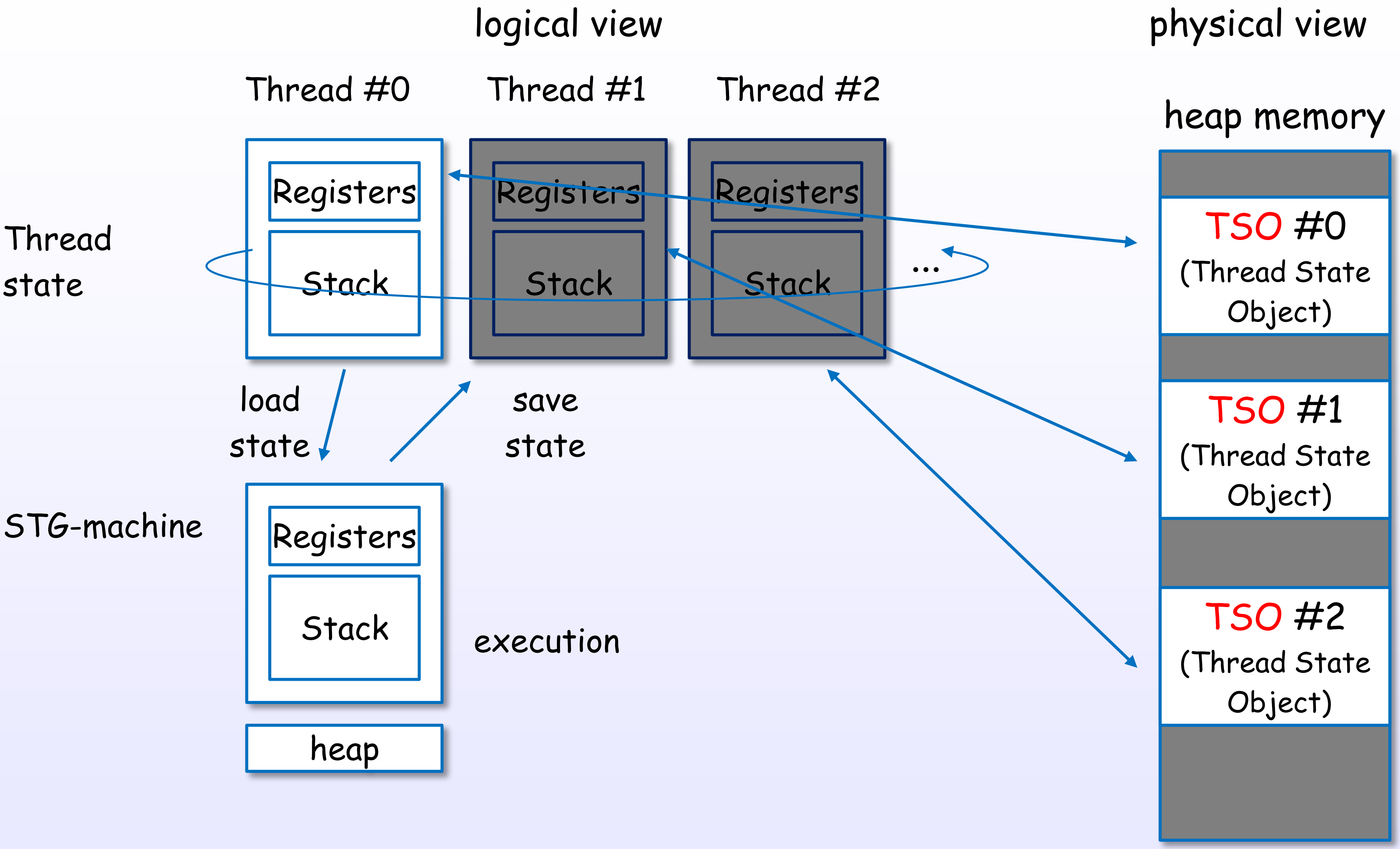
interleaved exclusive execution

STG-machine

execution

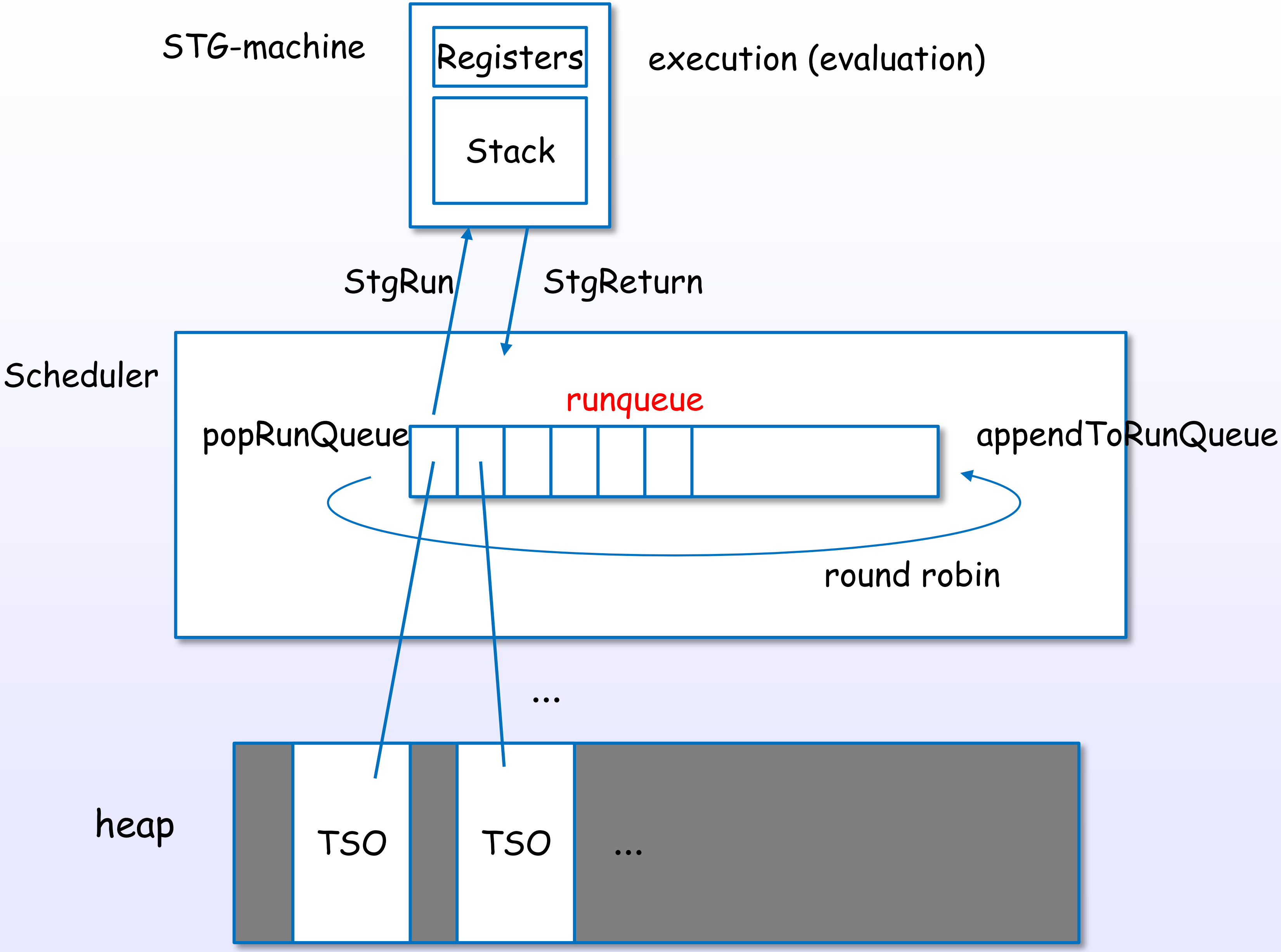
References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14]

Threads and TSOs



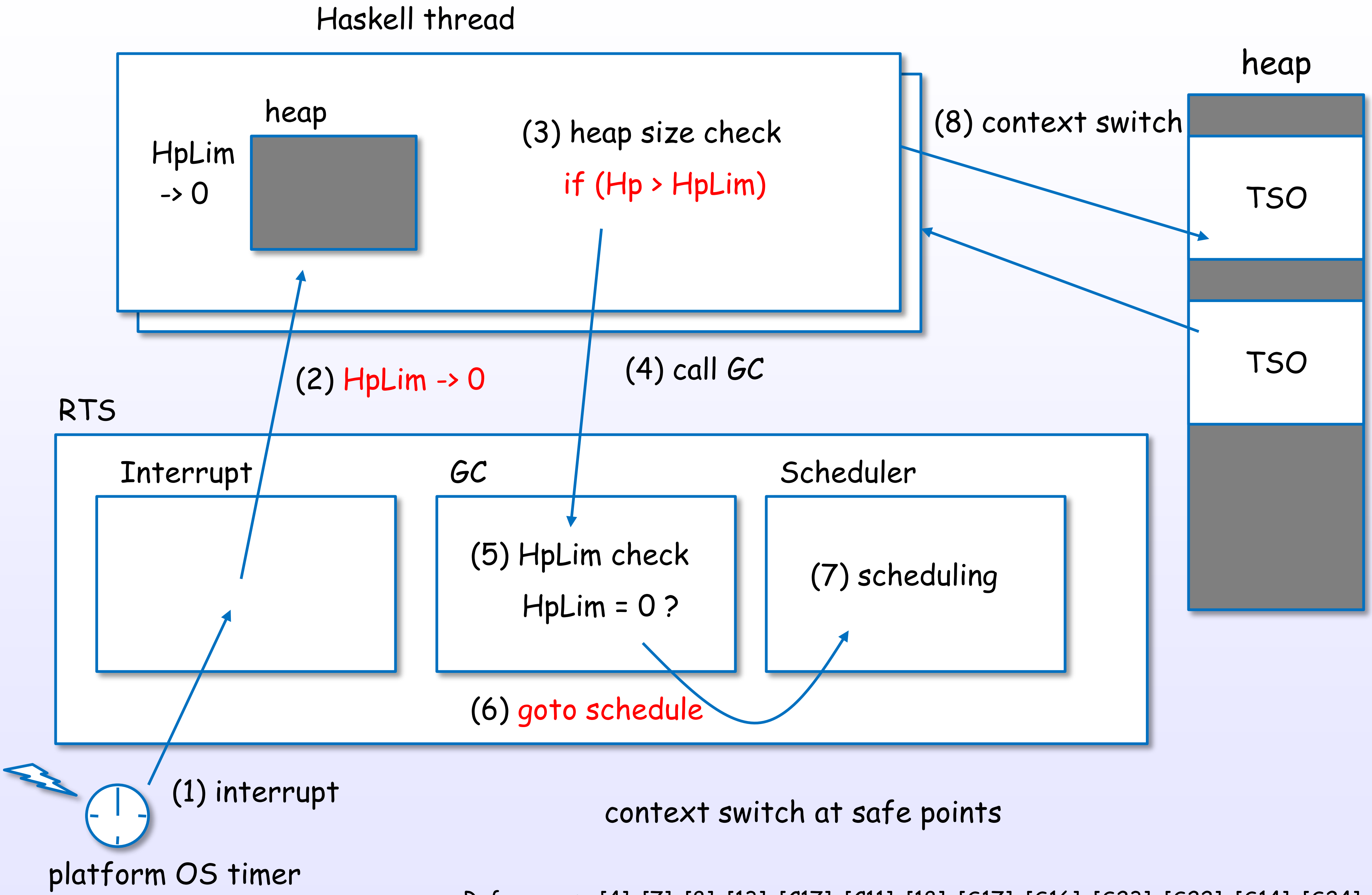
References : [4], [7], [8], [13], [C17], [C11], [18], [S17], [S16], [S23], [S22], [S14]

Scheduling threads



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

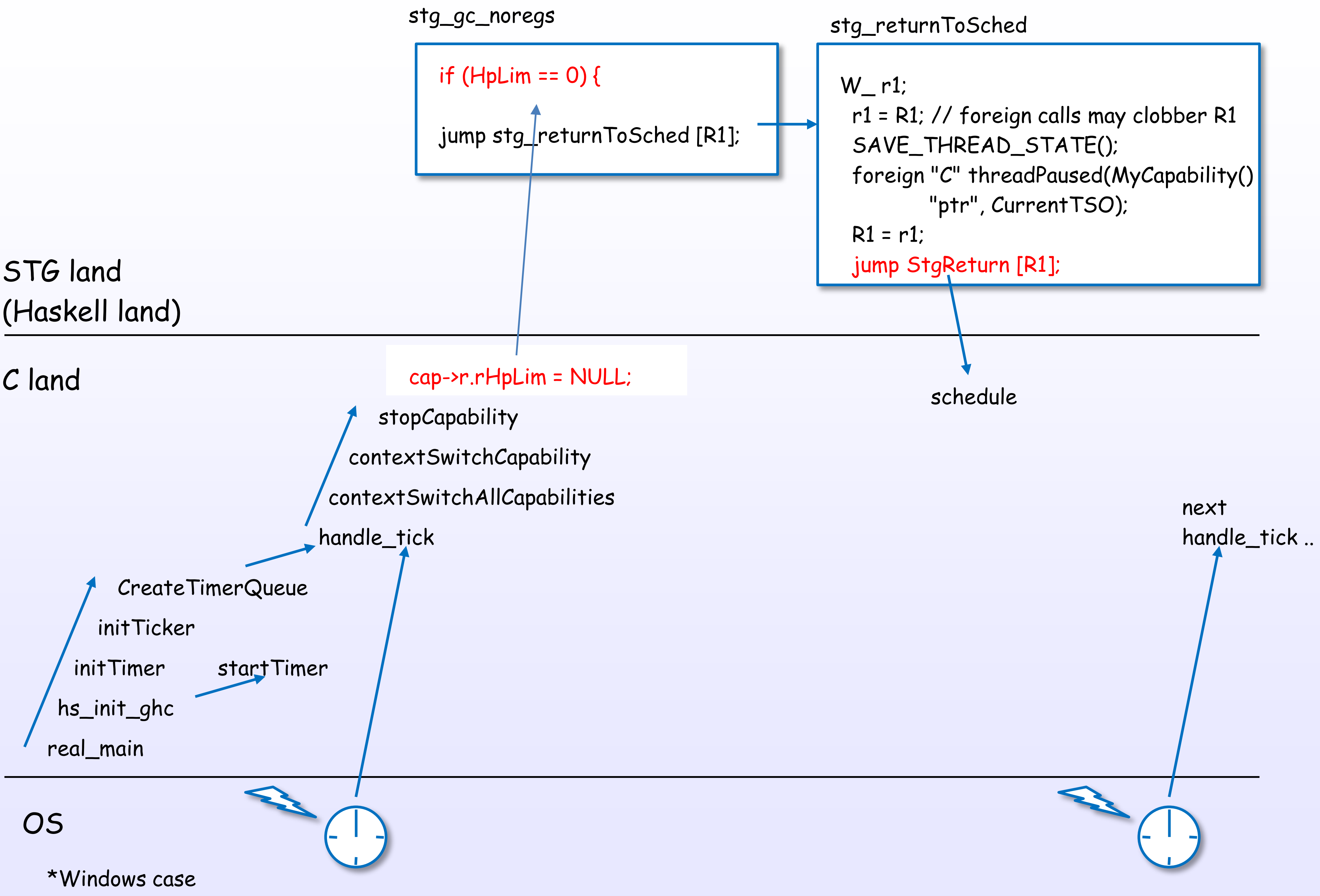
Context switch flow



context switch at safe points

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

Context switch flow (code)



stg_gc_noregs

```
if (HpLim == 0) {

jump stg_returnToSched [R1];
```

stg_returnToSched

```
W_ r1;
  r1 = R1; // foreign calls may clobber R1
  SAVE_THREAD_STATE();
  foreign "C" threadPaused(MyCapability()
              "ptr", CurrentTSO);
  R1 = r1;
  jump StgReturn [R1];
```

STG land
(Haskell land)

C land

cap->r.rHpLim = NULL;

stopCapability

contextSwitchCapability

contextSwitchAllCapabilities

handle_tick

schedule

next
handle_tick ..

CreateTimerQueue

initTicker

initTimer

startTimer

hs_init_ghc

real_main

OS

*Windows case

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

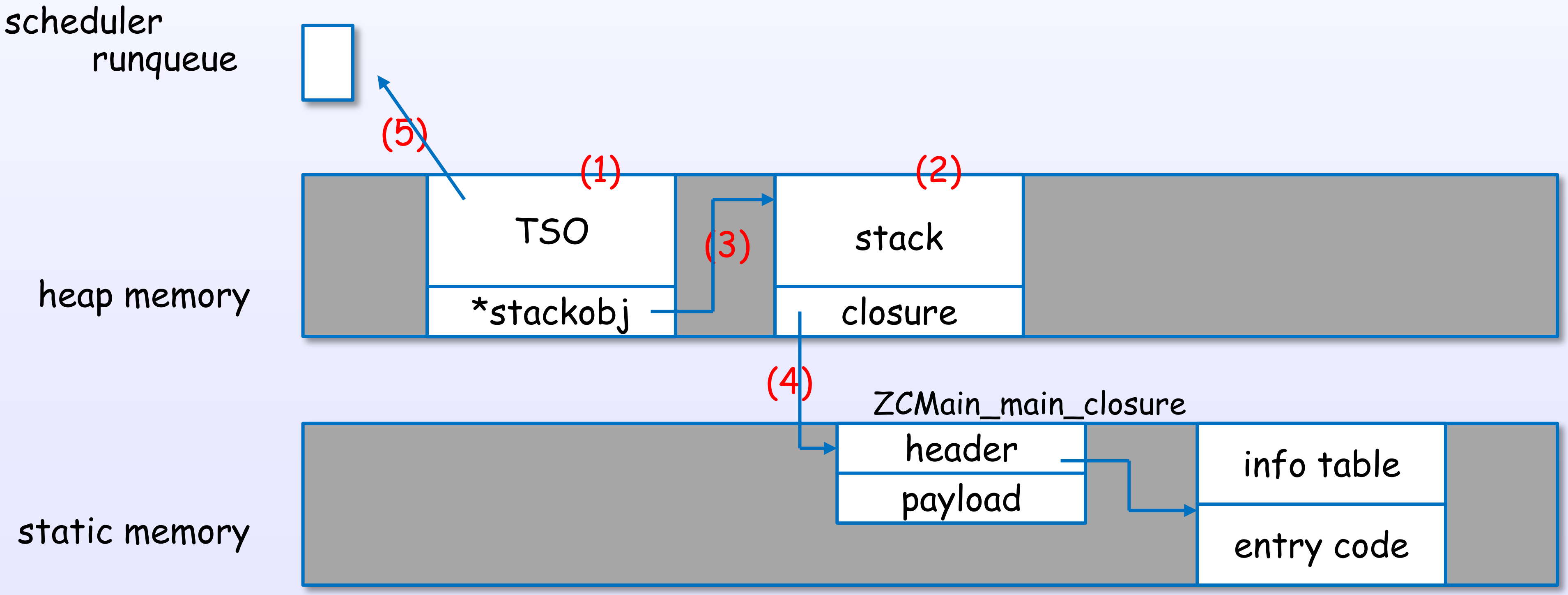
Creating main thread and forkIO

Create main thread

C land

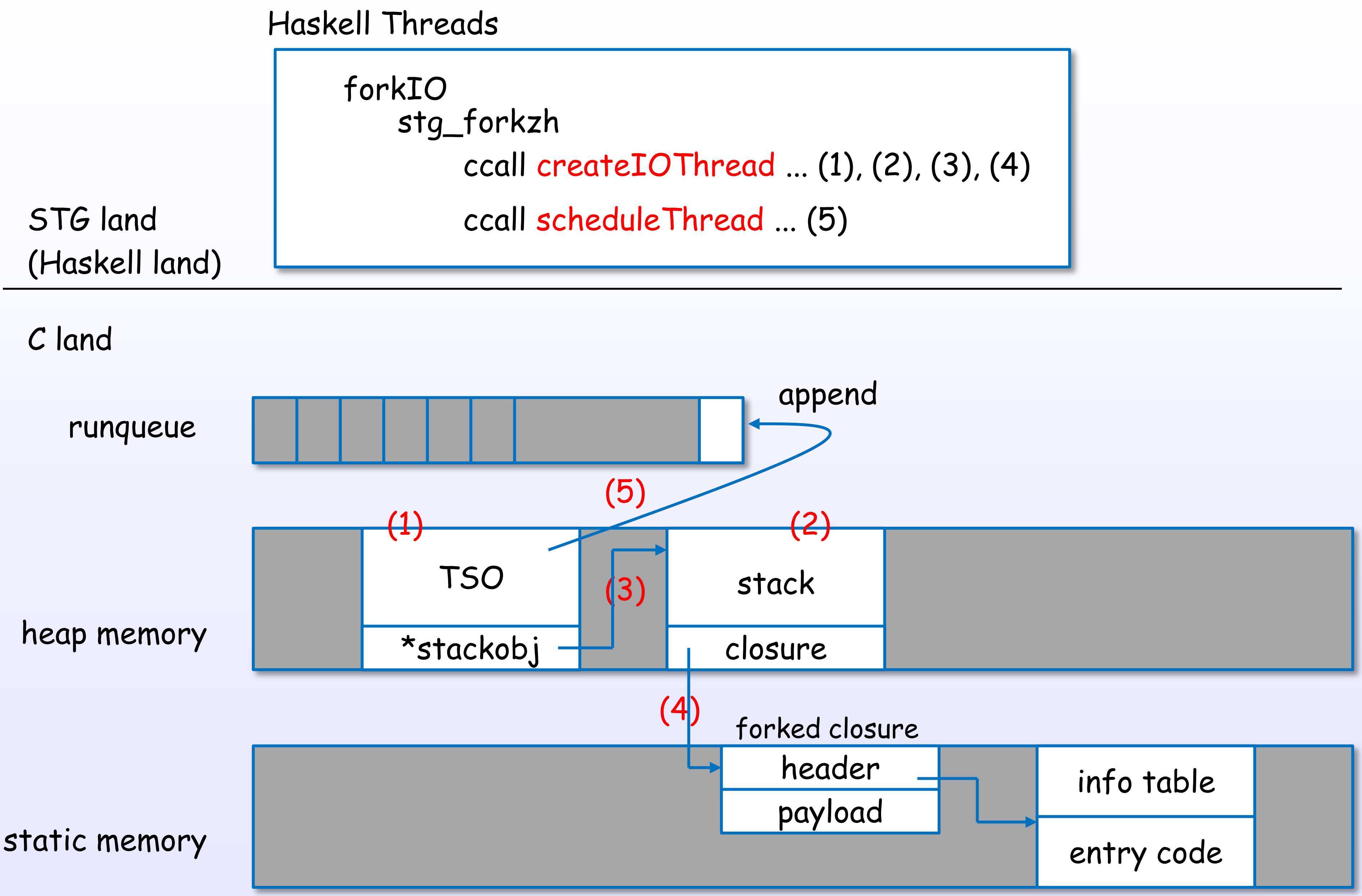
Runtime system bootstrap code [rts/RtsAPI.c]

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



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

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

runqueue

append

(5)

heap memory

(1)

TSO

*stackobj

(3)

(2)

stack

closure

(4)

forked closure

static memory

header

payload

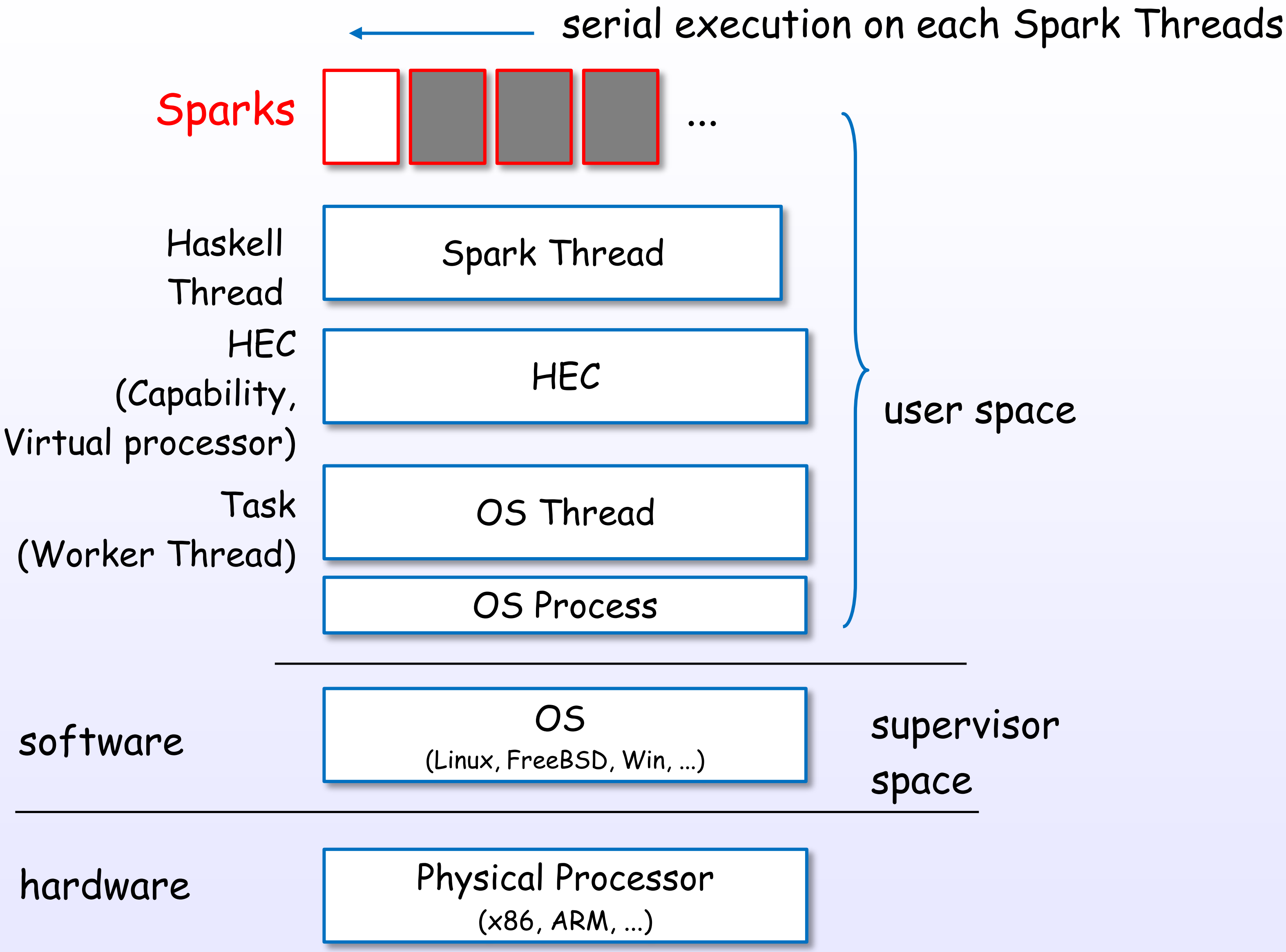
info table

entry code

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

Spark

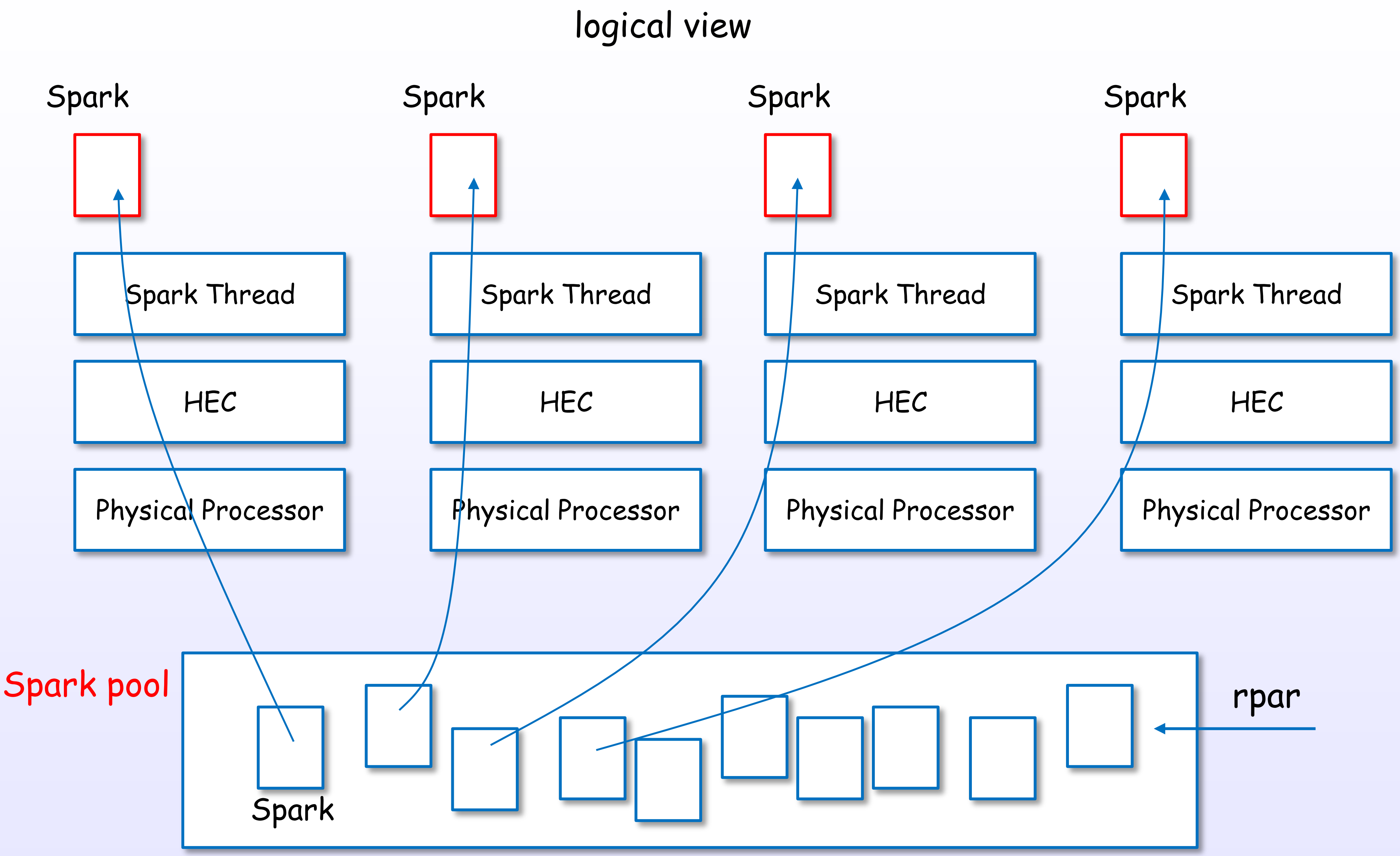
Spark layer



Spark Threads are generated on idle HECs.

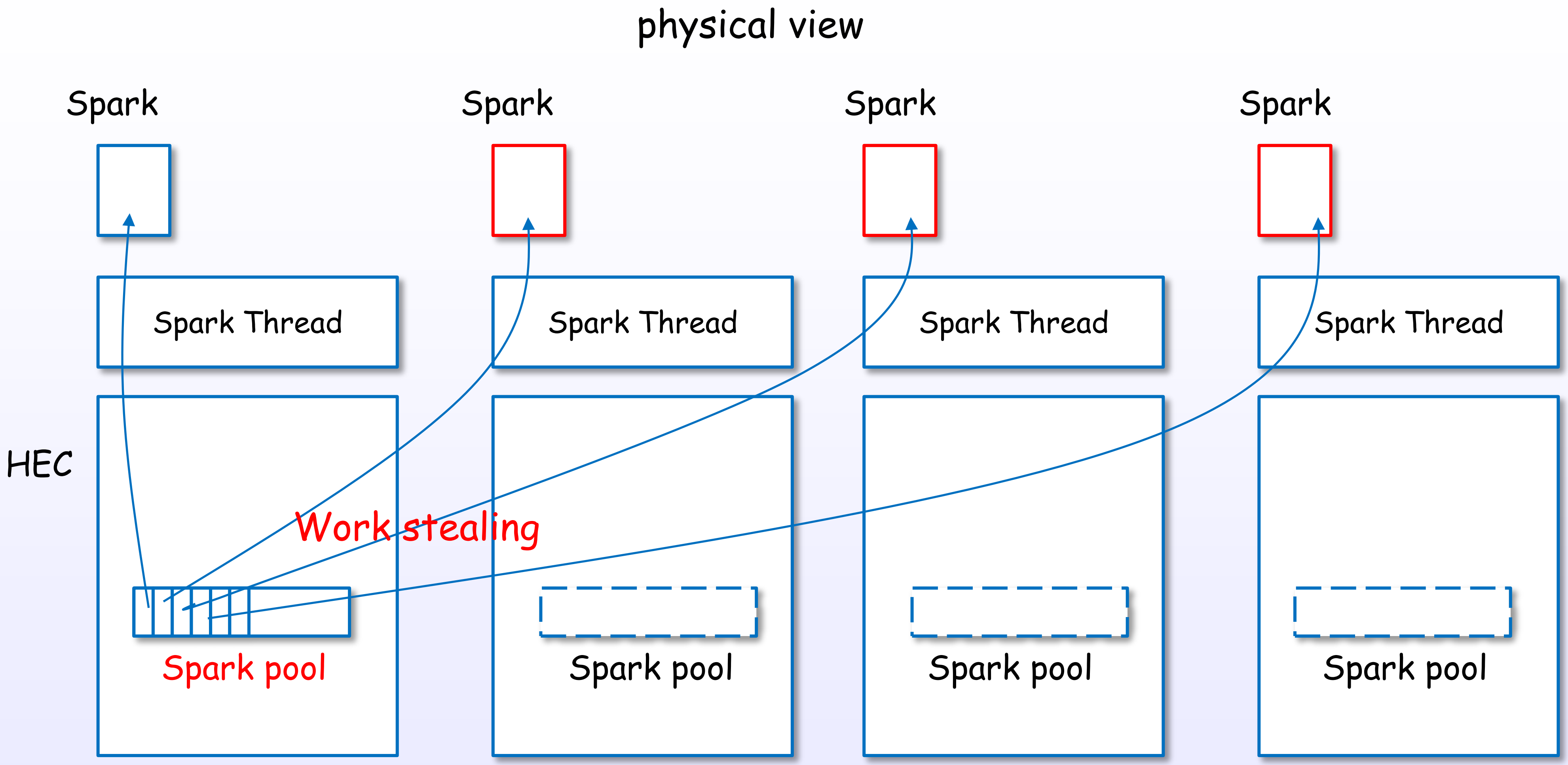
References : [C17], [18], [S17], [S26], [S27], [S32], [S12]

Sparks and Spark pool



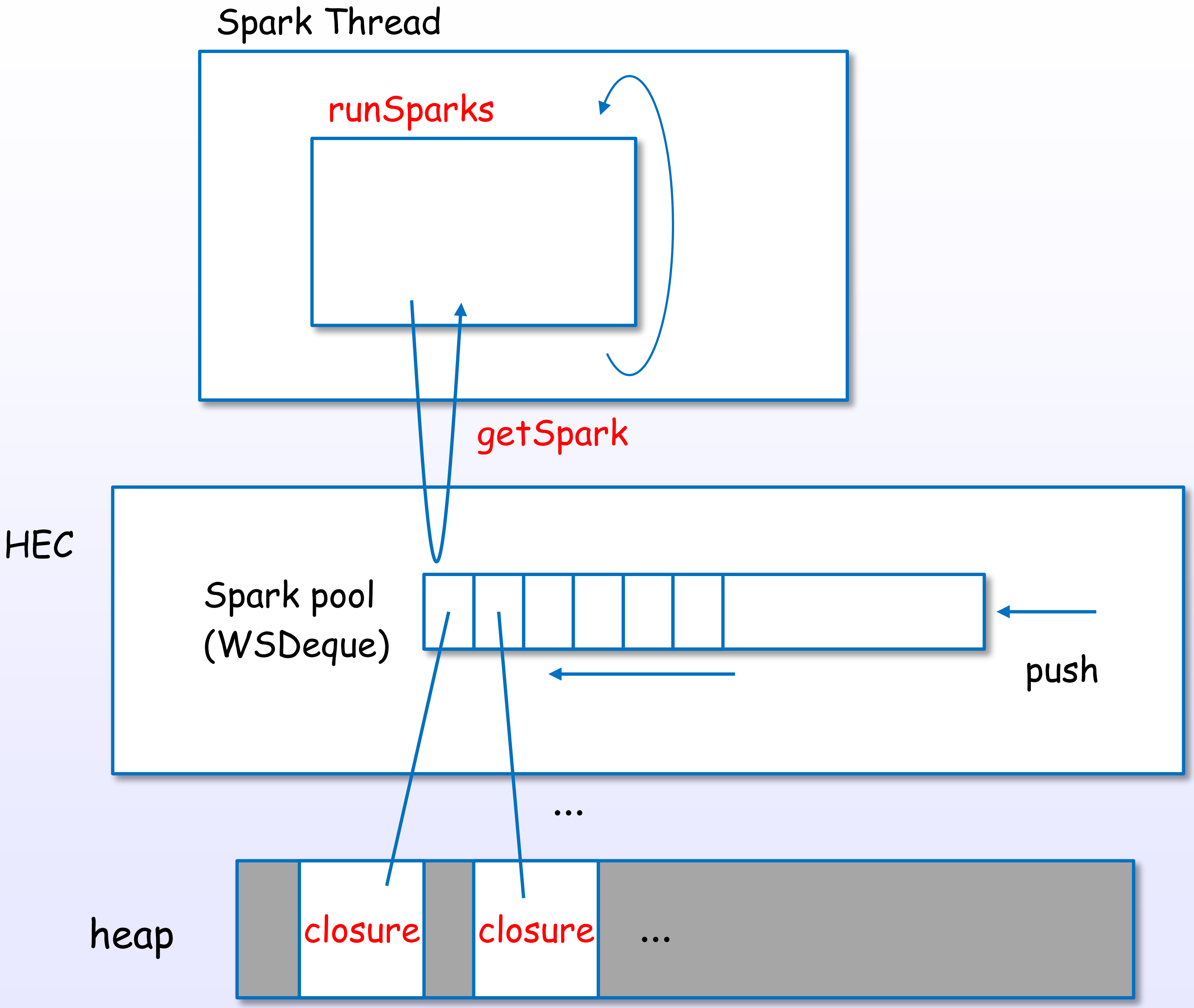
References : [C17], [18], [S17], [S26], [S27], [S32], [S12]

Spark pool and work stealing



References : [C17], [18], [S17], [S26], [S27], [S32], [S12]

Sparks and closures

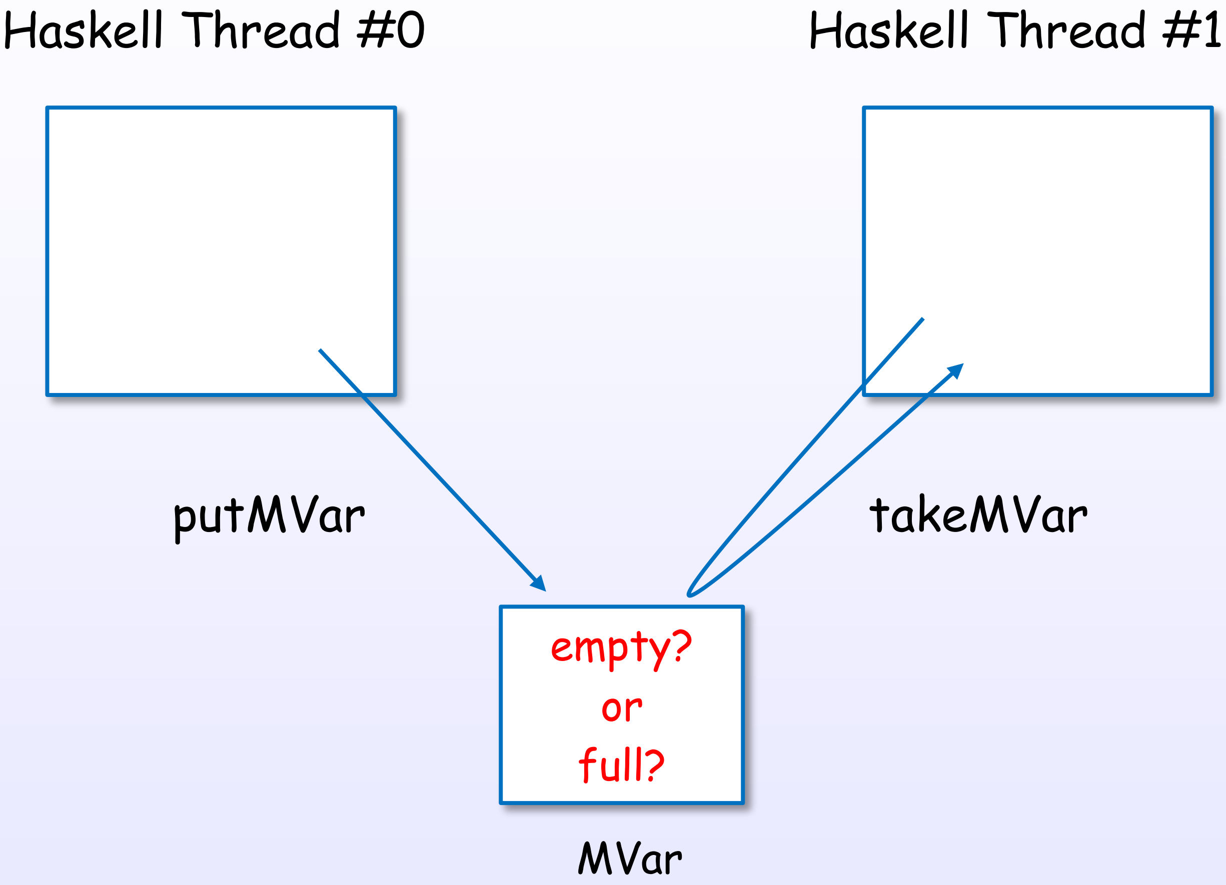


(not TSO objects, but closures. thus lightweight)

References : [C17], [18], [S17], [S26], [S27], [S32], [S12]

MVar

MVar


Haskell Thread #0

Haskell Thread #1

putMVar

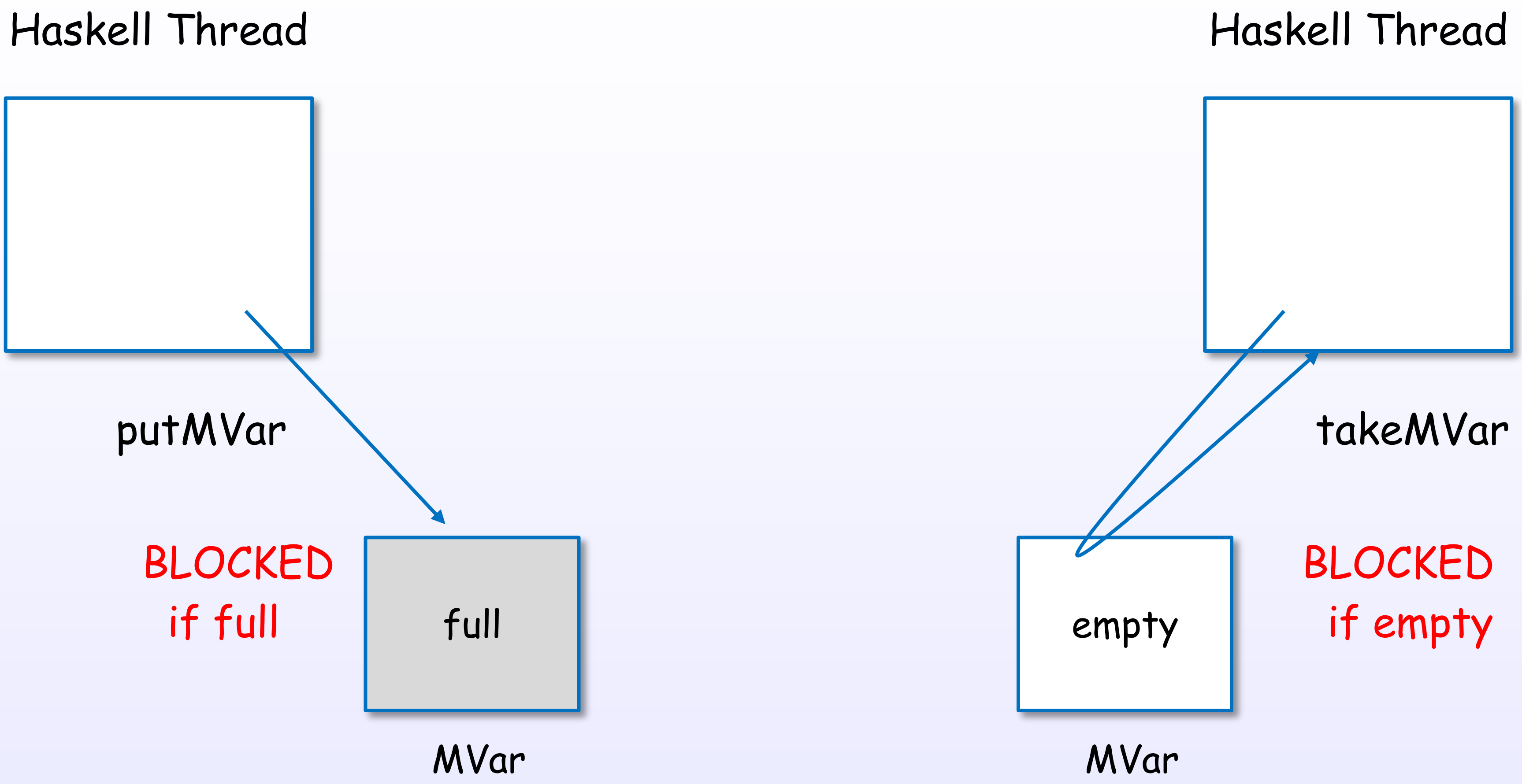
takeMVar

empty?
or
full?

MVar

References : [15], [17], [18], [S30], [S12]

MVar



Haskell Thread

Haskell Thread

putMVar

takeMVar

BLOCKED
if full

full

empty

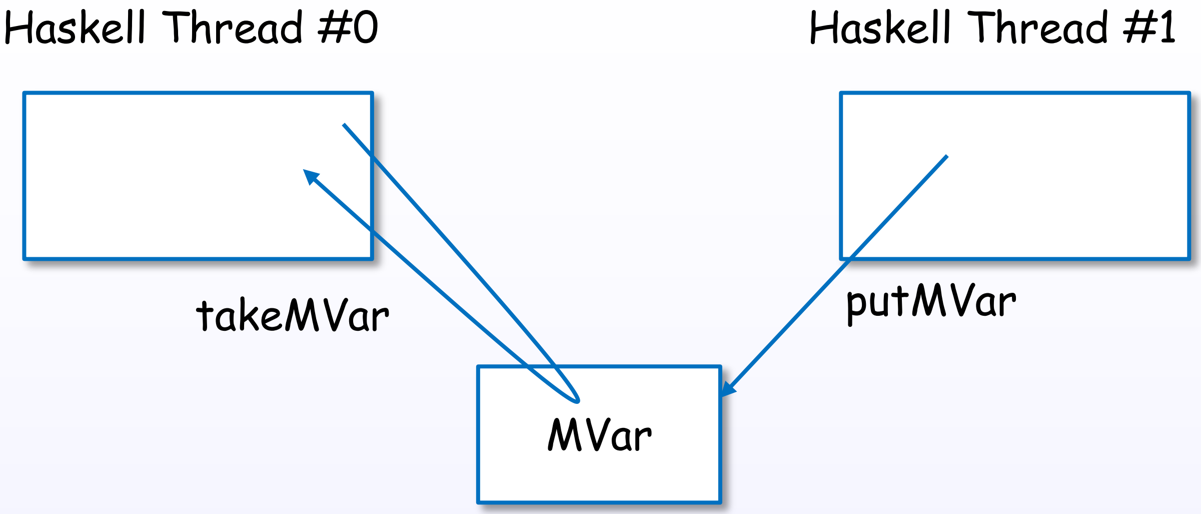
BLOCKED
if empty

MVar

MVar

References : [15], [17], [18], [S30], [S12]

MVar example



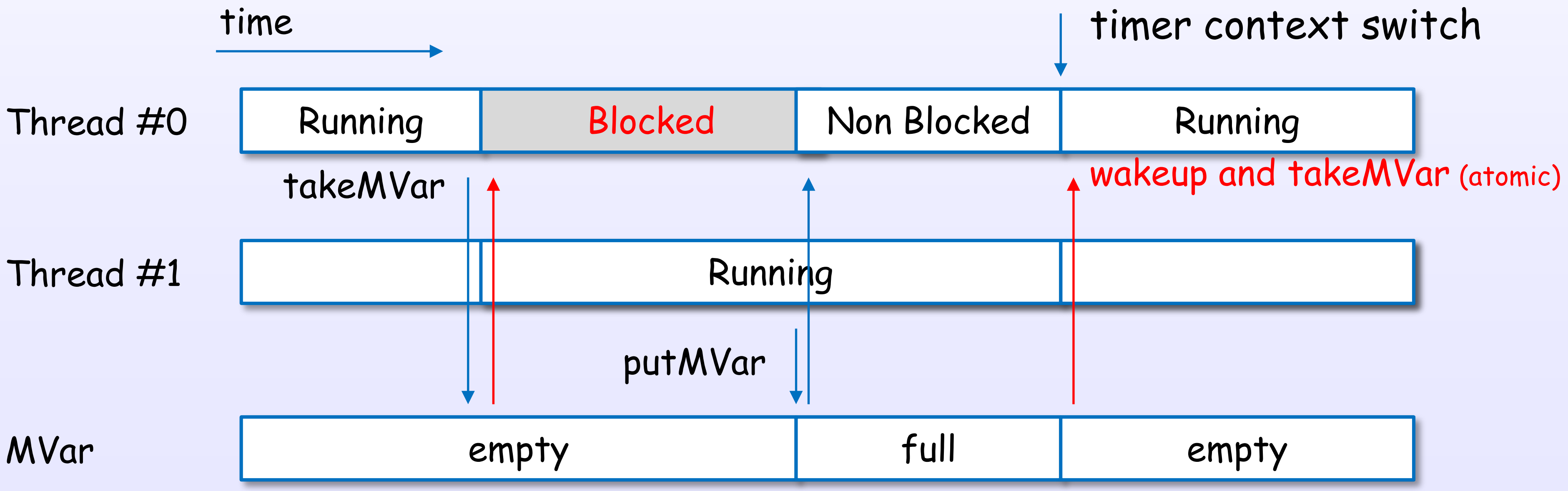
Haskell Thread #0

Haskell Thread #1

takeMVar

putMVar

MVar



time

timer context switch

Thread #0

Running

Blocked

Non Blocked

Running

takeMVar

wakeup and takeMVar (atomic)

Thread #1

Running

putMVar

MVar

empty

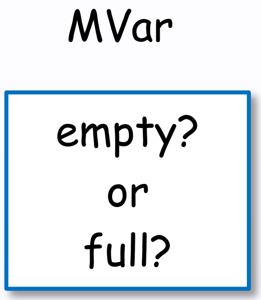
full

empty

References : [15], [17], [18], [S30], [S12]

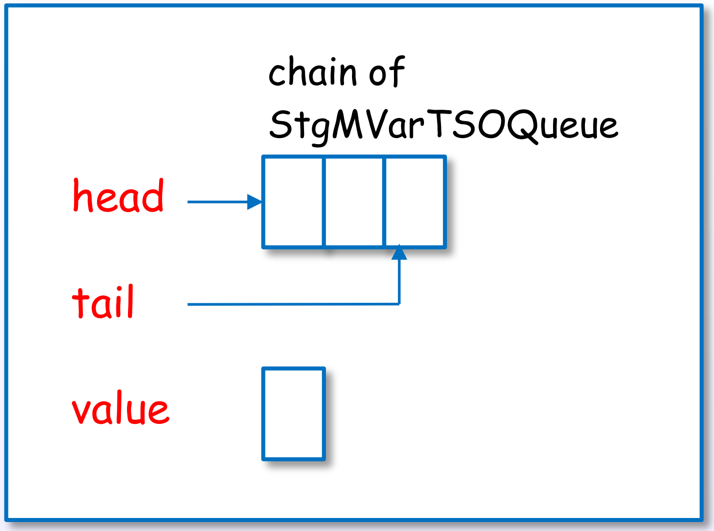
MVar view

User view


MVar

empty?
or
full?

logical MVar object

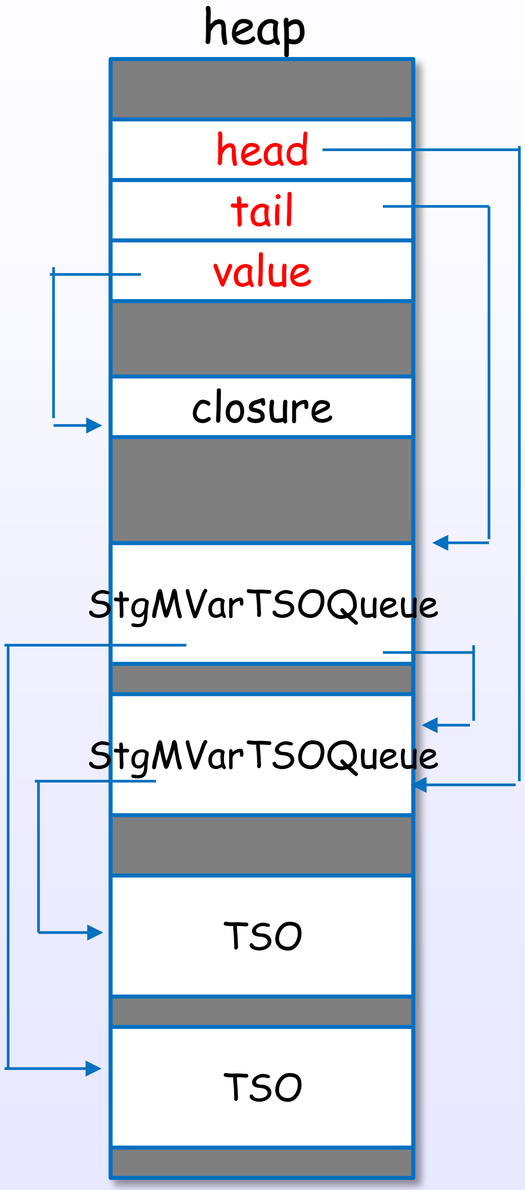

chain of StgMVarTSOQueue

head

tail

value

physical MVar object


heap

head

tail

value

closure

StgMVarTSOQueue

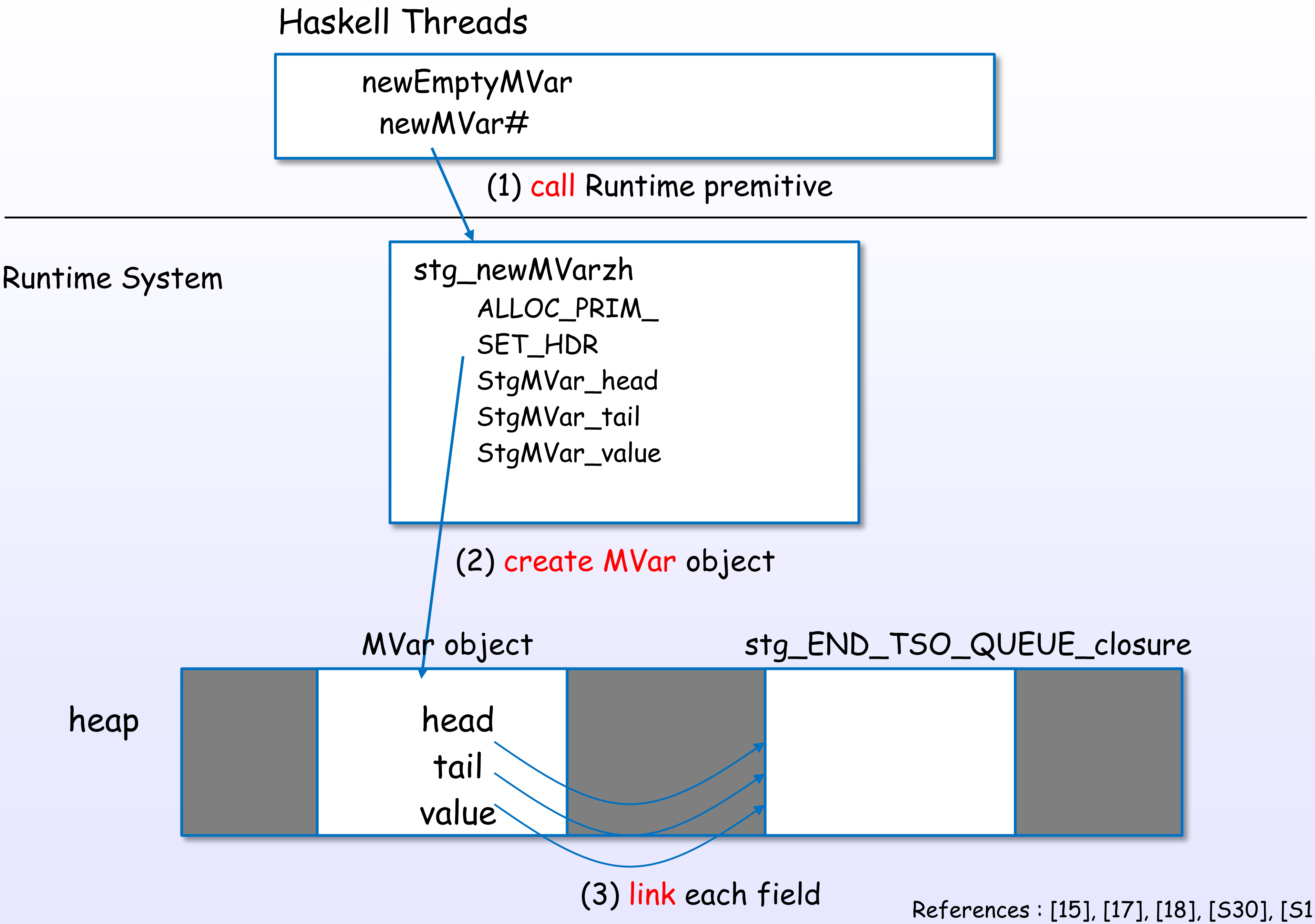
StgMVarTSOQueue

TSO

TSO

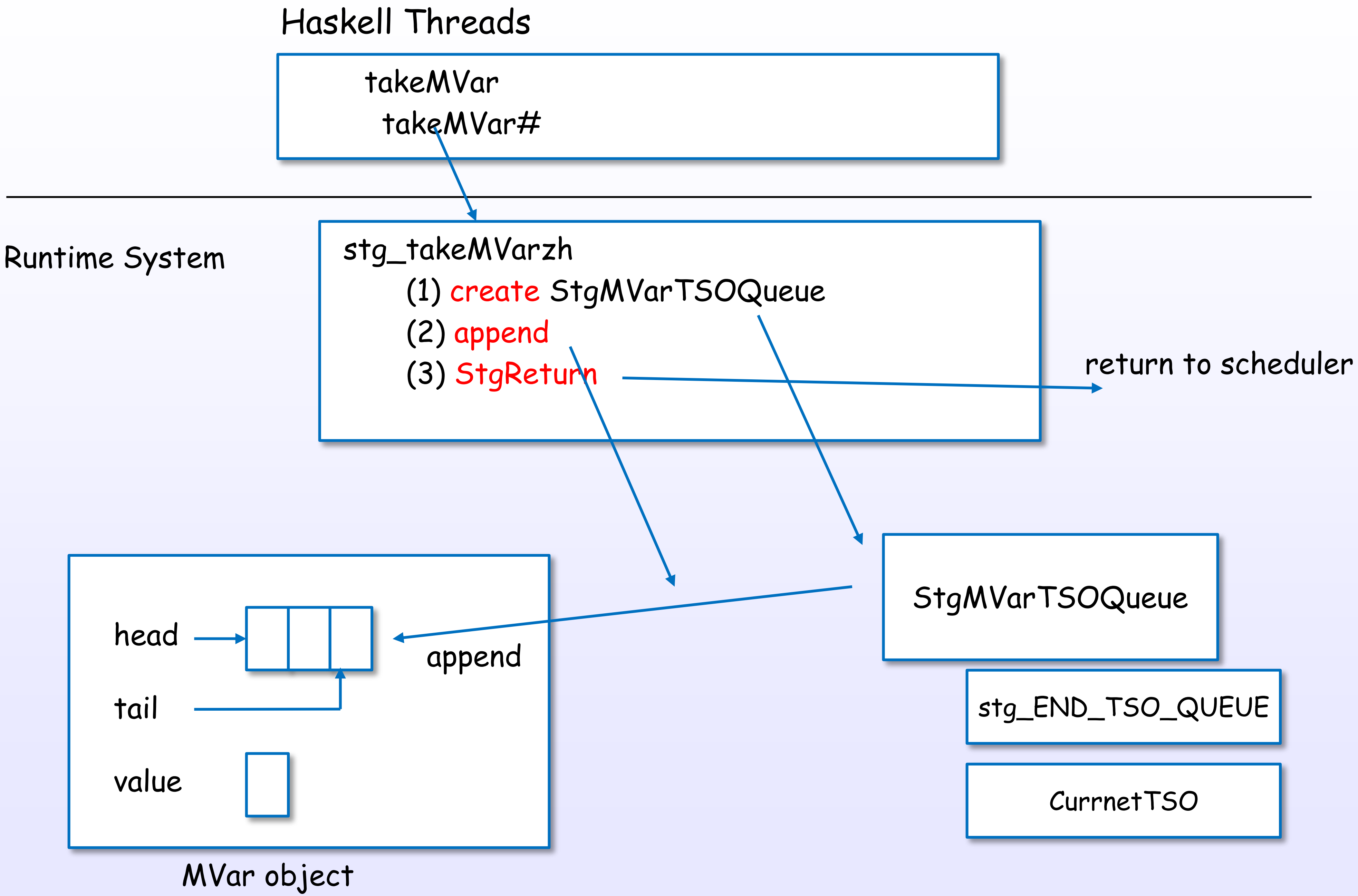
References : [15], [17], [18], [S30], [S12]

newEmptyMVar



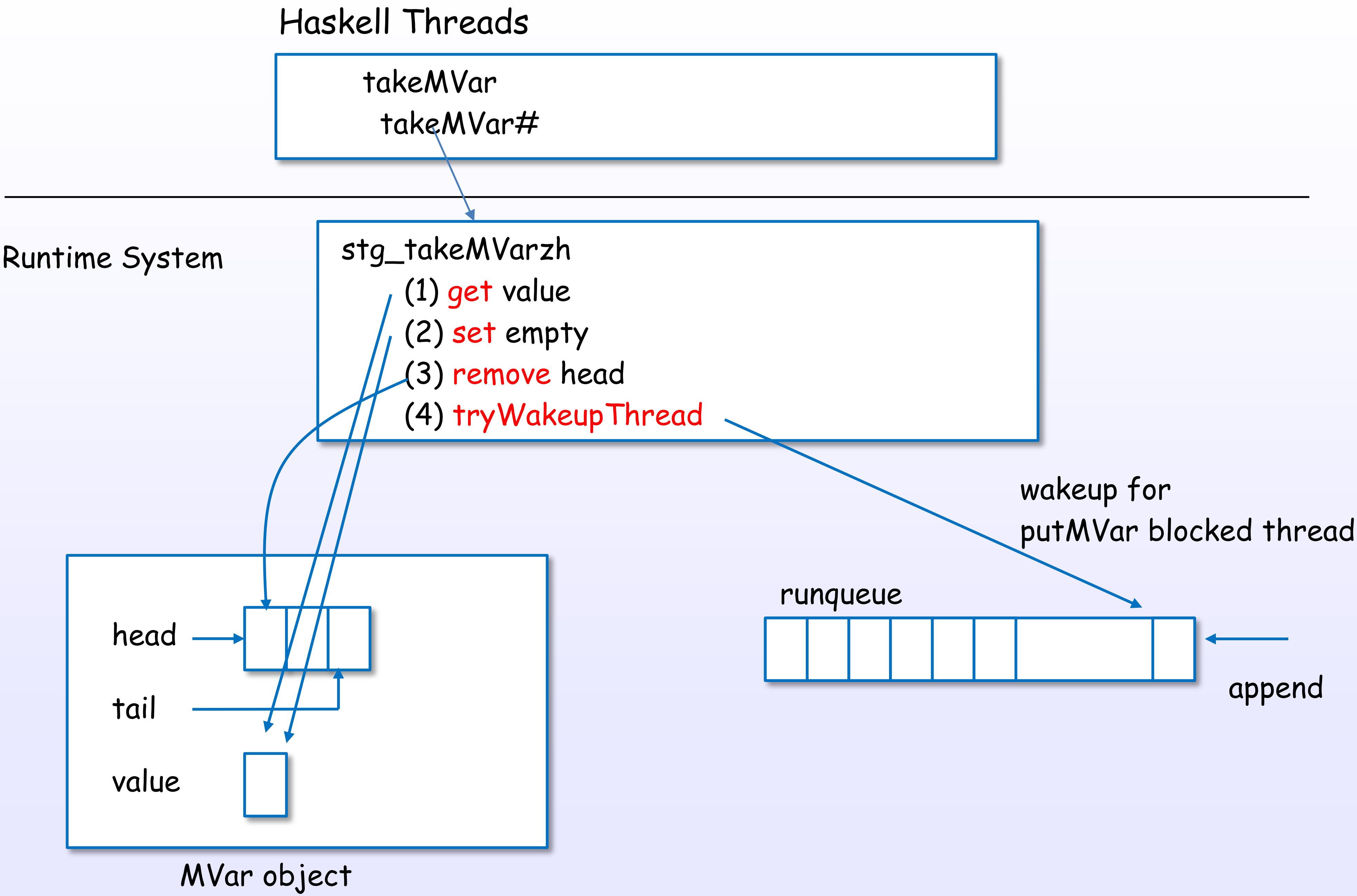
References : [15], [17], [18], [S30], [S12]

takeMVar (empty case)



References : [15], [17], [18], [S30], [S12]

takeMVar (full case)



Haskell Threads

takeMVar

takeMVar#

Runtime System

stg_takeMVarzh

(1) get value

(2) set empty

(3) remove head

(4) tryWakeupThread

wakeup for
putMVar blocked thread

head

tail

value

MVar object

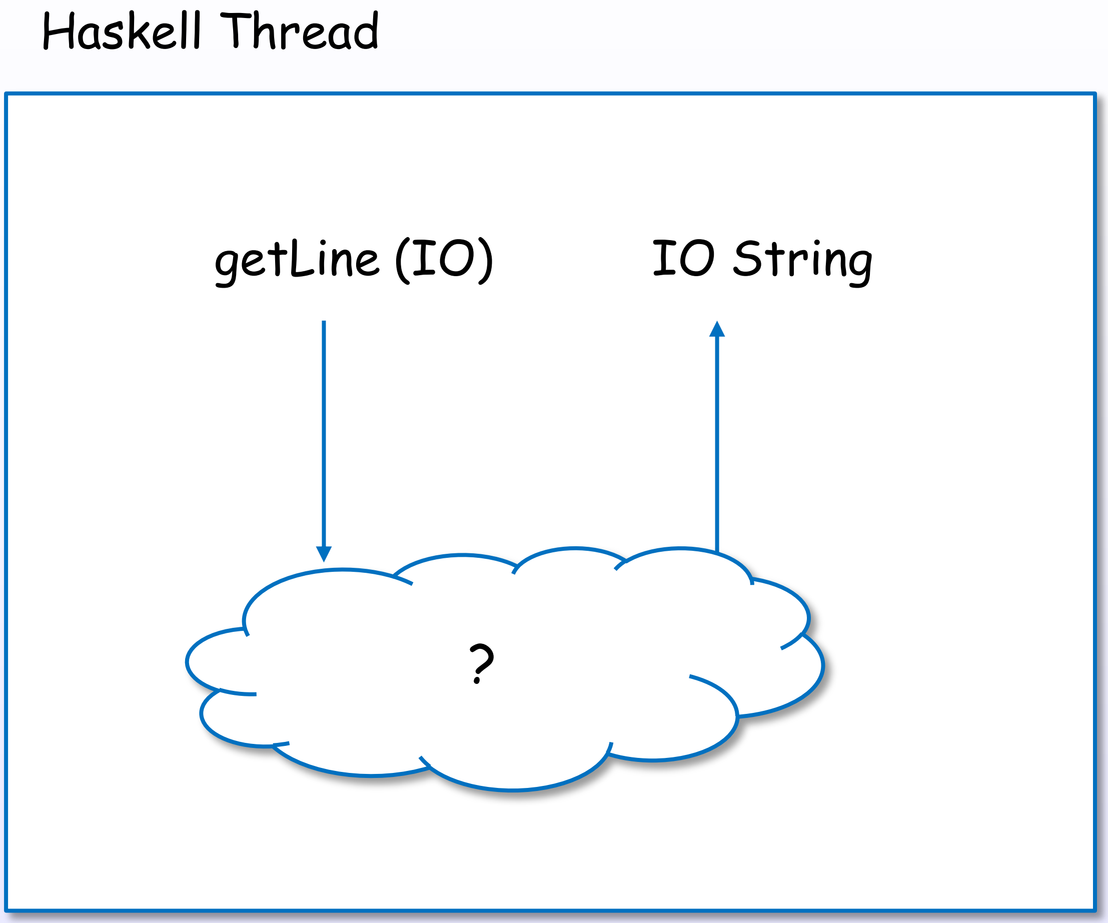
runqueue

append

References : [15], [17], [18], [S30], [S12]

IO and FFI

IO


Haskell Thread

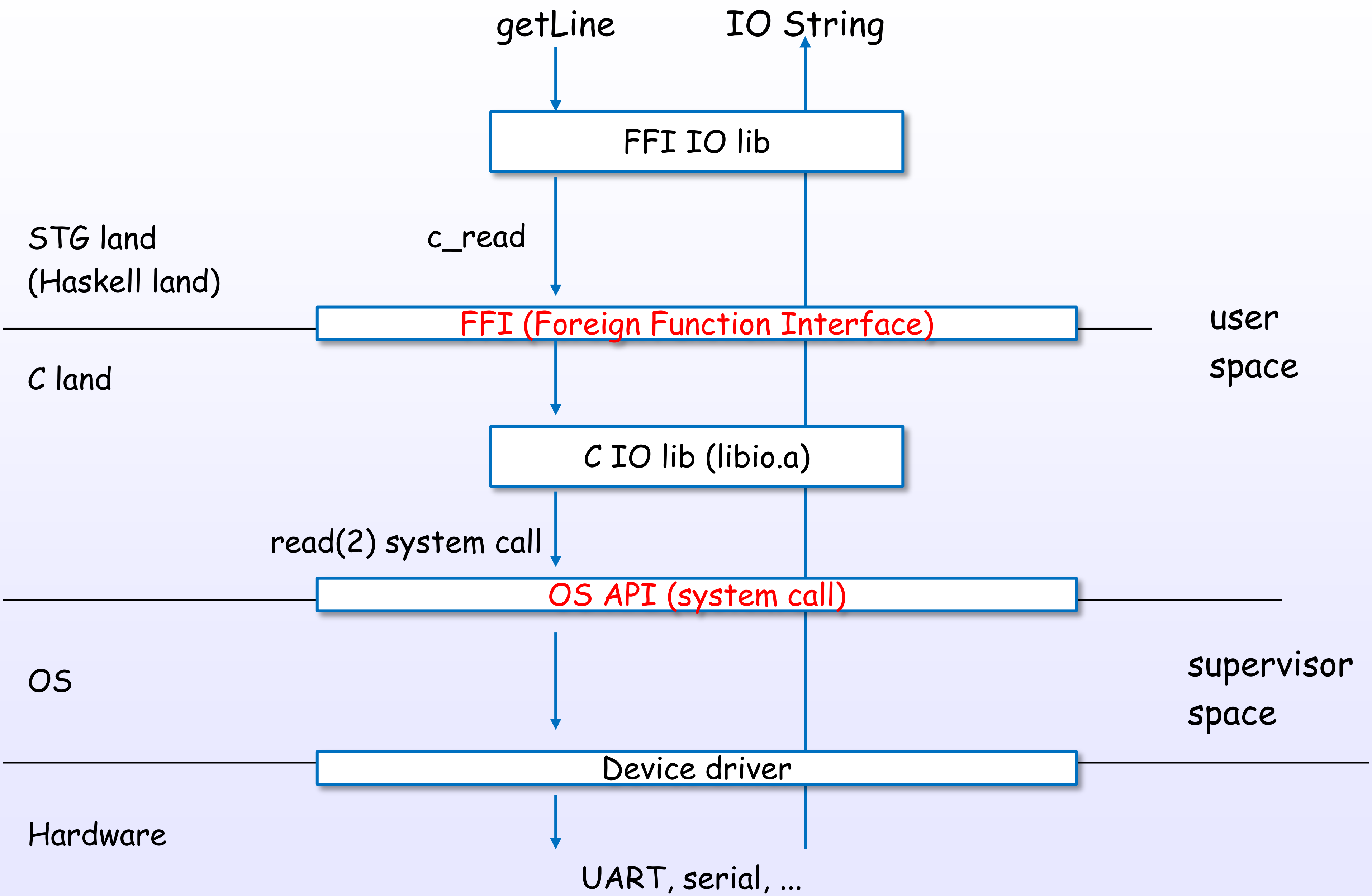
getLine (IO)

IO String

?

References : [5], [10]

IO example: getLine



getLine

IO String

FFI IO lib

STG land
(Haskell land)

c_read

FFI (Foreign Function Interface)

user space

C land

C IO lib (libio.a)

read(2) system call

OS API (system call)

OS

supervisor space

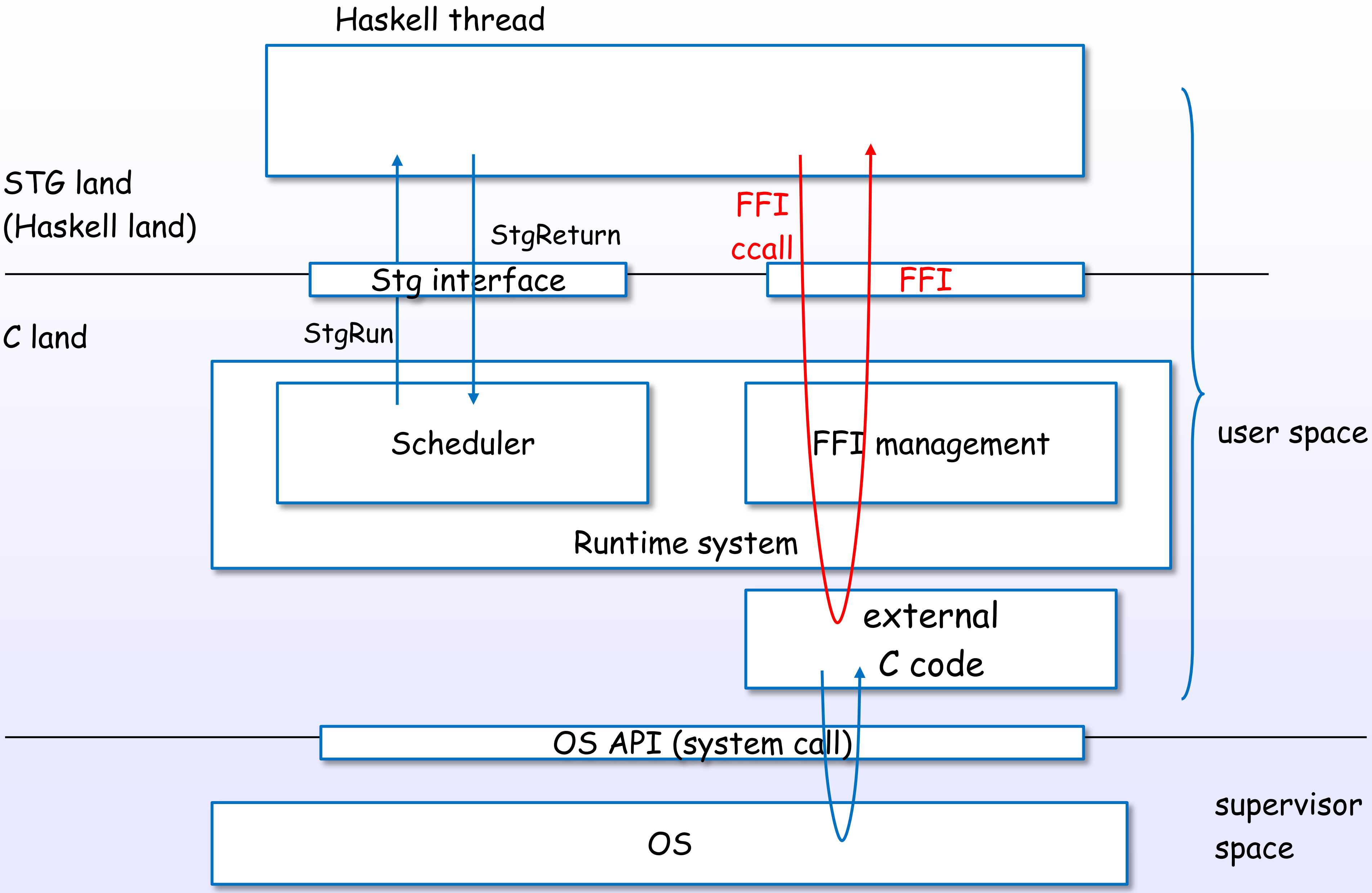
Device driver

Hardware

UART, serial, ...

References : [5], [10], [19], [S38], [S37], [S36], [S35], [S39]

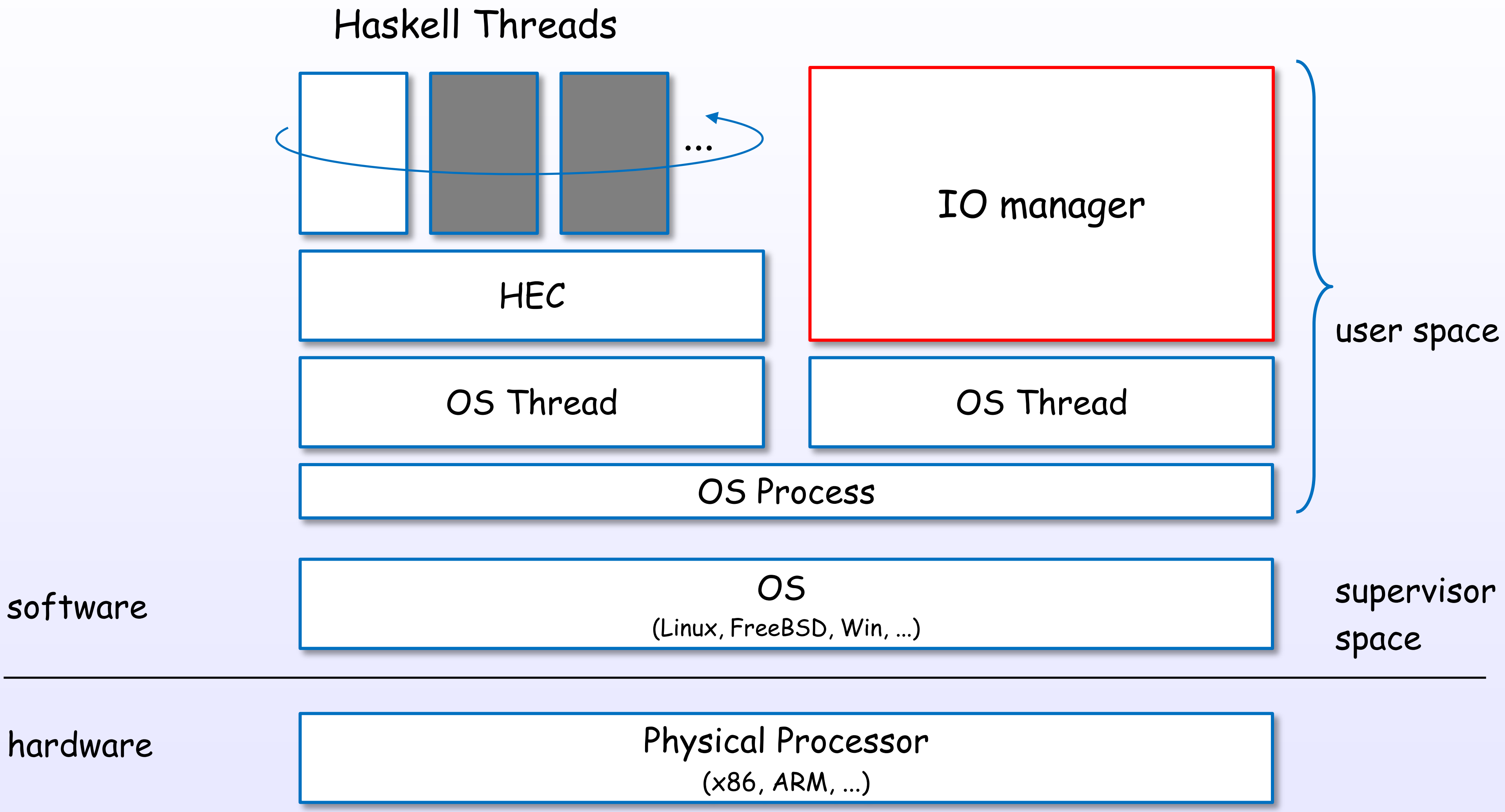
FFI (Foreign Function Interface)



References : [5], [10], [19], [S38], [S37], [S36], [S35], [S39]

IO manager

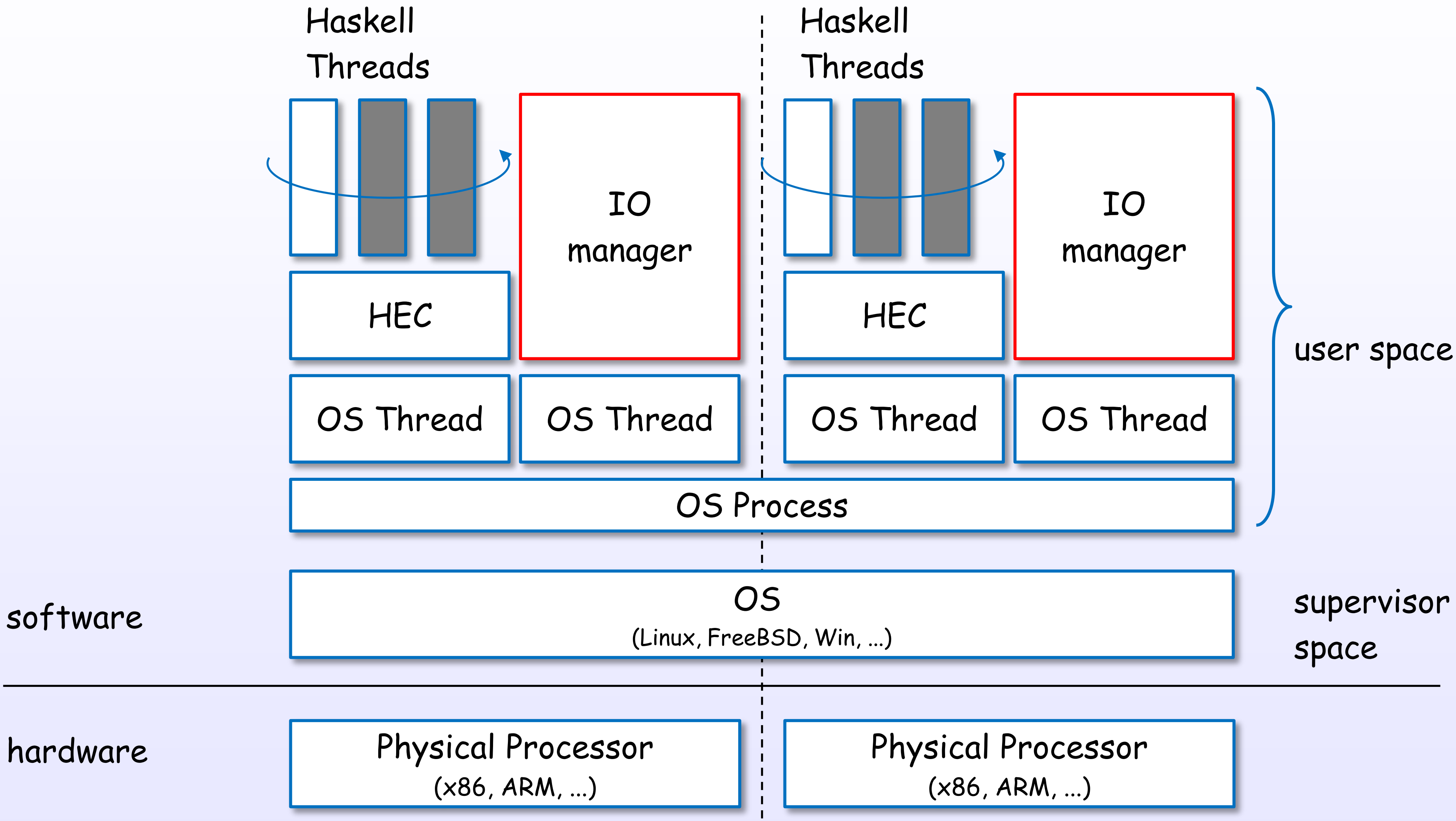
IO manager (single core)



*Threaded option case (ghc -threaded)

References : [6], [4], [7]

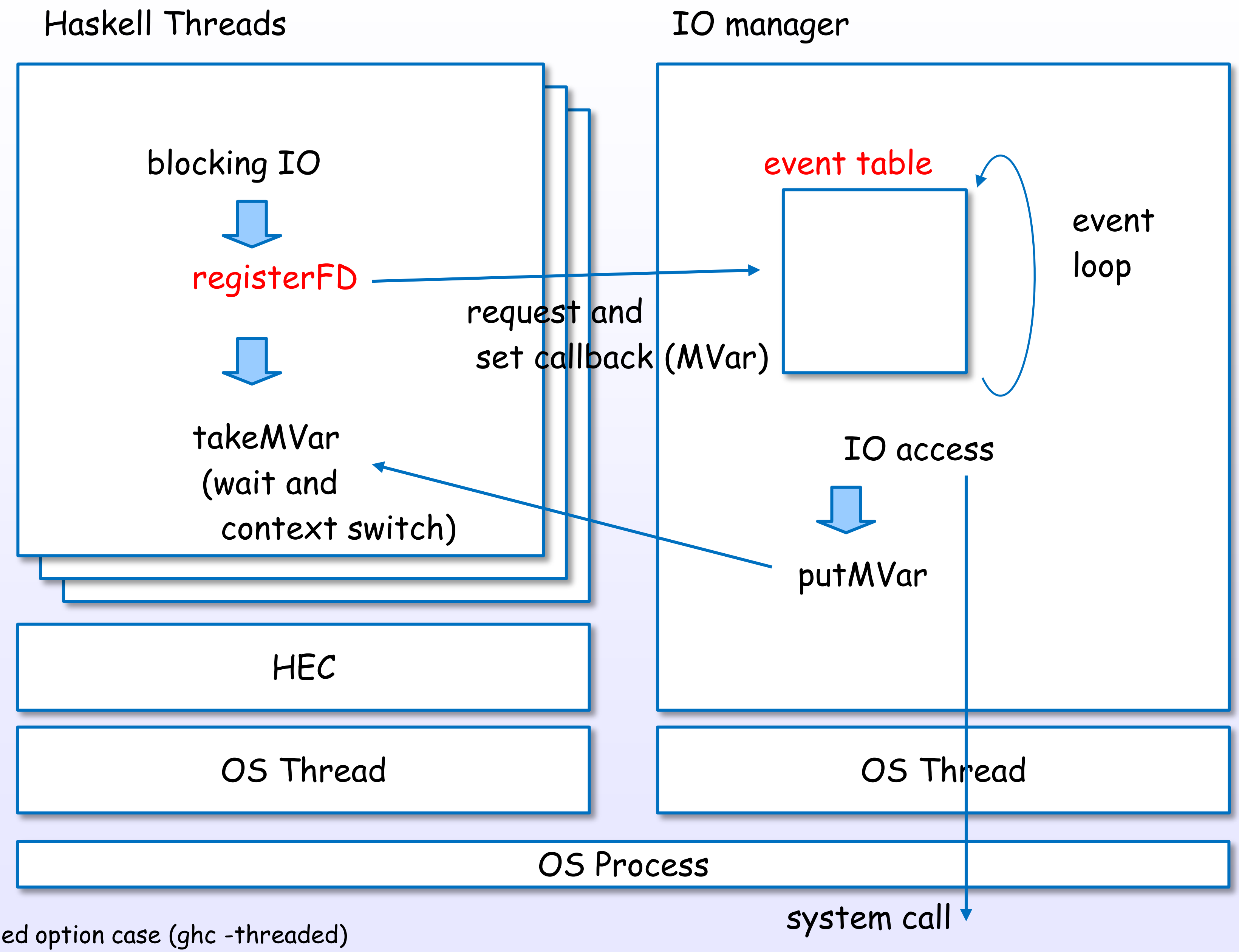
IO manager (multi core)



*Threaded option case (ghc -threaded)

References : [6], [4], [7]

IO manager

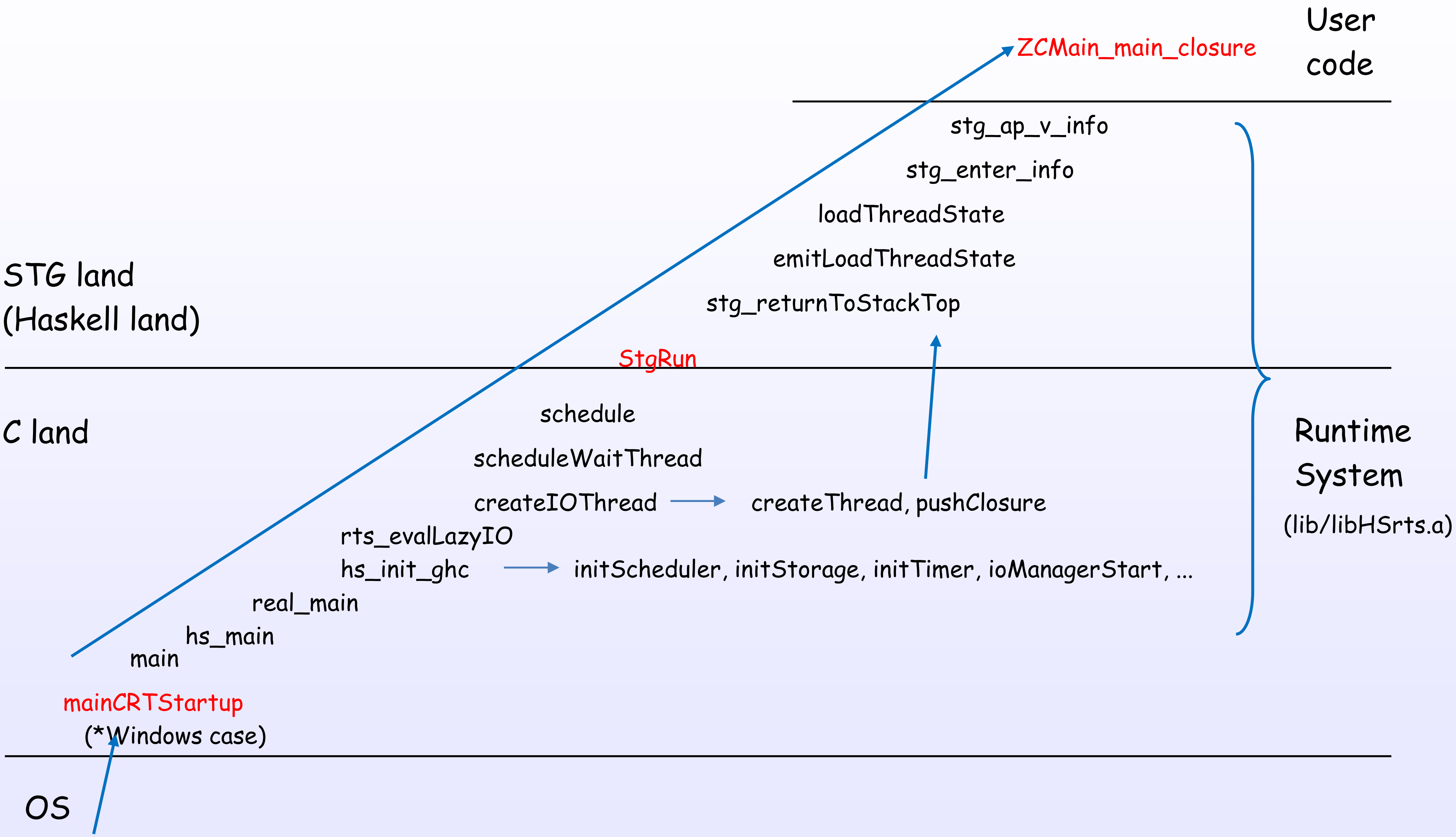


*Threaded option case (ghc -threaded)

References : [6], [4], [7], [S28], [S29], [S31], [S36], [S34], [S3]

Bootstrap

Bootstrap sequence



User code

ZCMain_main_closure

stg_ap_v_info

stg_enter_info

loadThreadState

emitLoadThreadState

stg_returnToStackTop

STG land
(Haskell land)

StgRun

C land

schedule

scheduleWaitThread

createIOThread → createThread, pushClosure

rts_evalLazyIO

hs_init_ghc → initScheduler, initStorage, initTimer, ioManagerStart, ...

real_main

hs_main

main

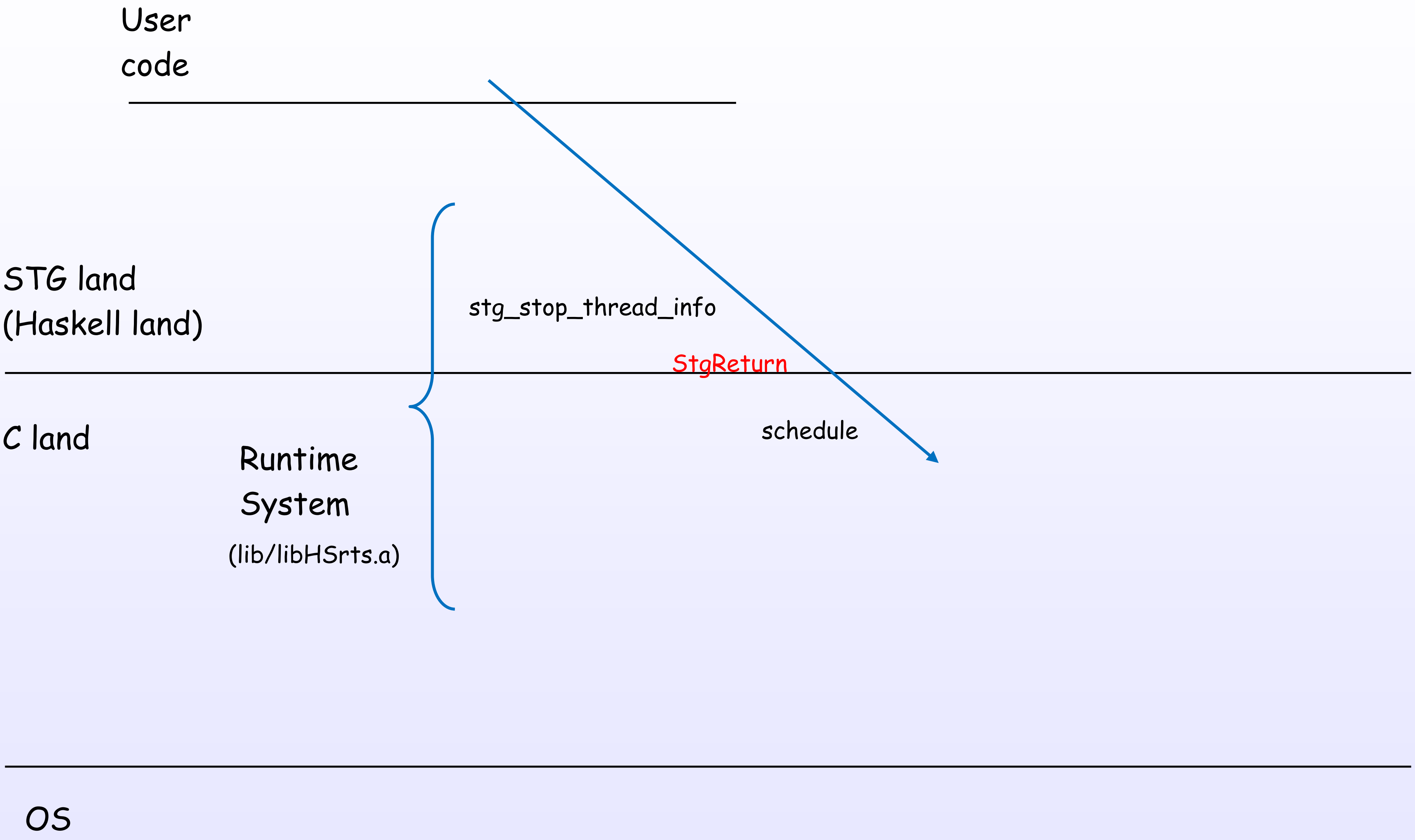
mainCRTStartup
(*Windows case)

Runtime System
(lib/libHSrts.a)

OS

References : [S7], [S13], [S14], [S17], [S18], [S19], [S9], [S10], [S21], [S40]

Exit sequence



User code

STG land (Haskell land)

C land

OS

Runtime System

(lib/libHSrts.a)

stg_stop_thread_info

StgReturn

schedule

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

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/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/posix/Signals.c
[S29] rts/win32/ThrIOManager.c
[S30] libraries/base/GHC/MVar.hs
[S31] libraries/base/GHC/Conc/IO.hs
[S32] libraries/base/GHC/Conc/Sync.lhs
[S33] libraries/base/GHC/Event/Manager.hs
[S34] libraries/base/GHC/Event/Thread.hs
[S35] libraries/base/GHC/IO/BufferedIO.hs
[S36] libraries/base/GHC/IO/FD.hs
[S37] libraries/base/GHC/IO/Handle/Text.hs
[S38] libraries/base/System/IO.hs
[S39] libraries/base/System/Posix/Internals.hs
[S40] AutoApply.o (utils/genapply/GenApply.hs)

Connect the algorithm and transistar