

GHC(STG,Cmm,asm) illustrated for hardware persons

exploring some mental models and implementations

Takenobu T.

"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 7.8 (and ghc 7.10).

Contents

- Executable binary
- Compile steps
- Runtime System
- Development languages

- Machine layer/models
- STG-machine
- Heap objects 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
- Bound thread

- Spark

- Mvar
- Software transactional memory

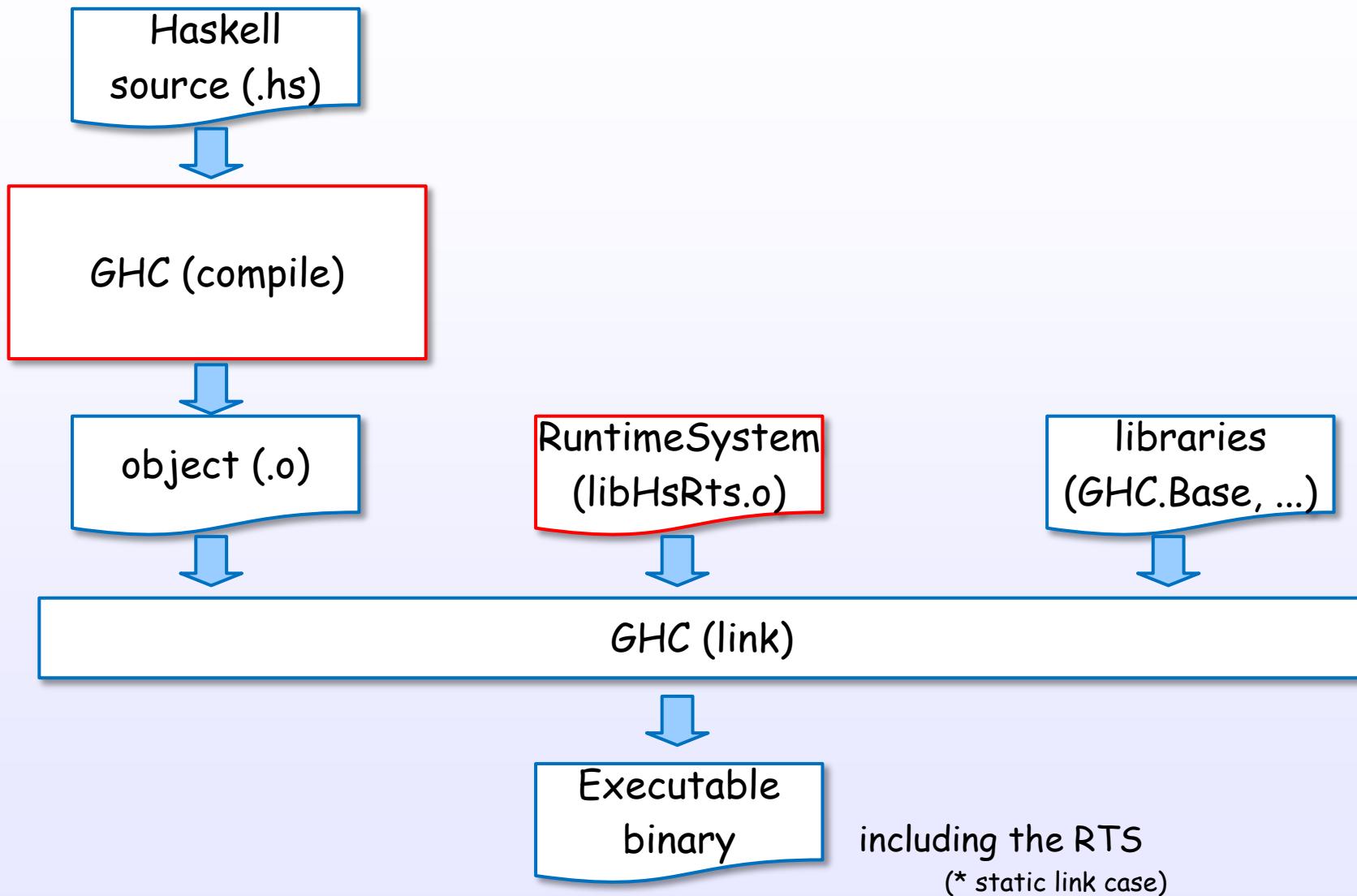
- FFI
- IO and FFI
- IO manager

- Bootstrap

- References

Executable binary

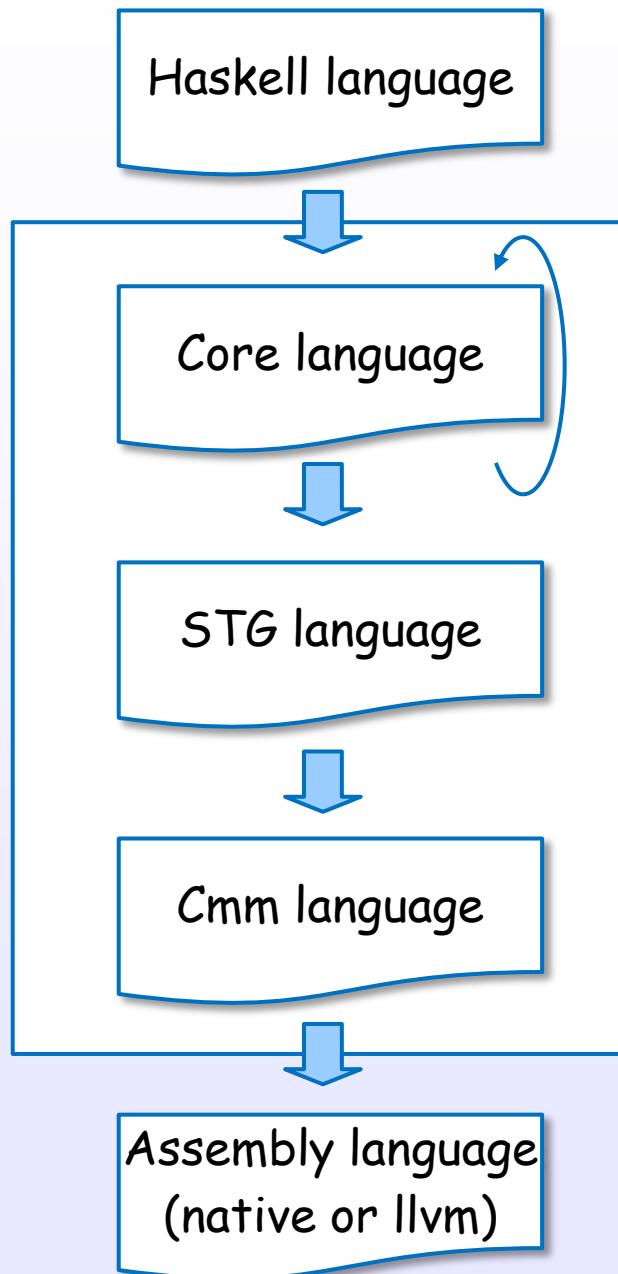
The GHC = Compiler + Runtime System (RTS)



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

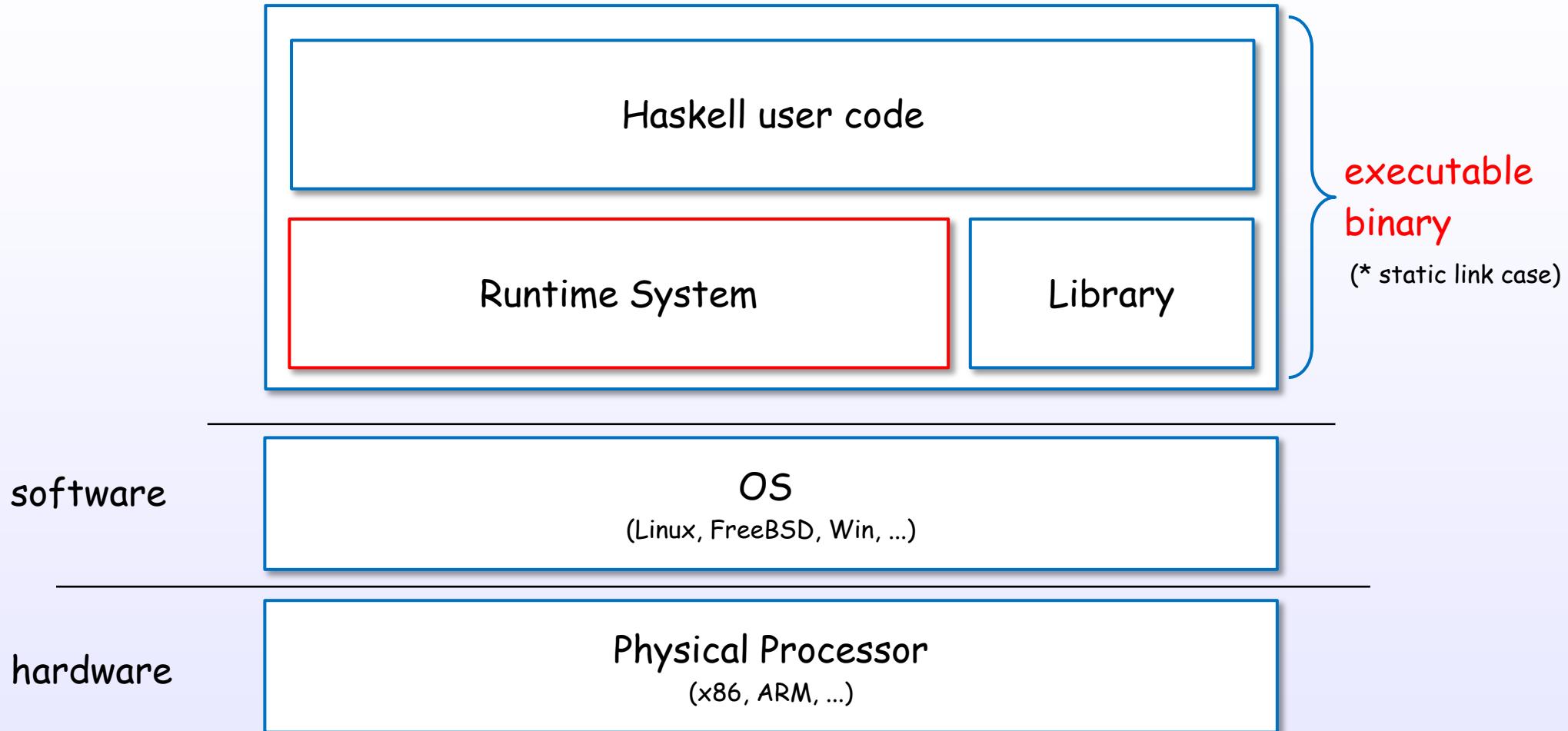
% ghc -ddump-stg

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

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

Runtime System

Generated binary includes the RTS



Runtime System includes ...

Runtime System

Storage Manager

User space
Scheduler

Byte-code interpreter

Profiling

Software
Transactional Memory

...

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

Machine layer/models

Machine layer

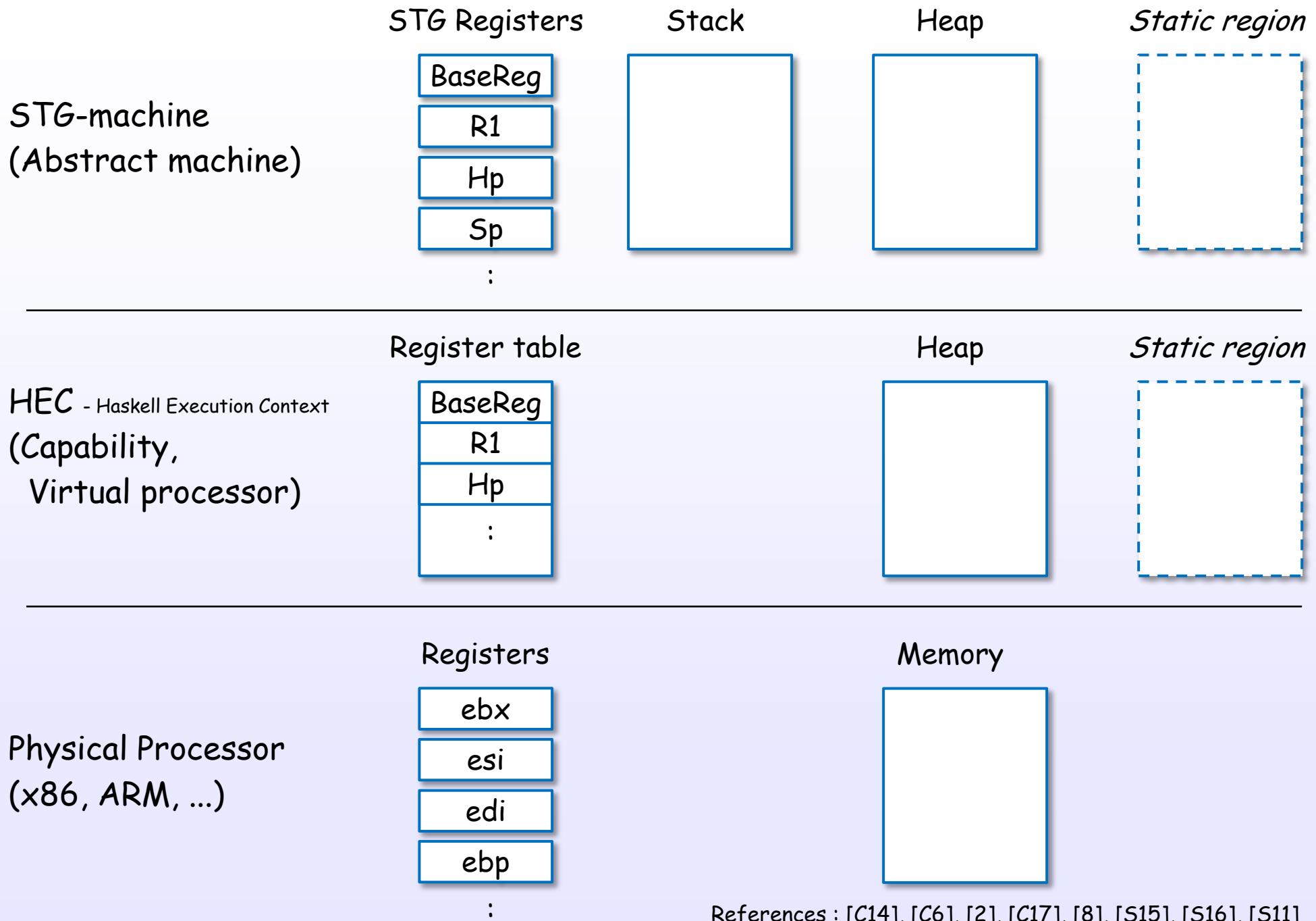
STG-machine
(Abstract machine)

HEC - Haskell Execution Context
(Capability, Virtual processor)

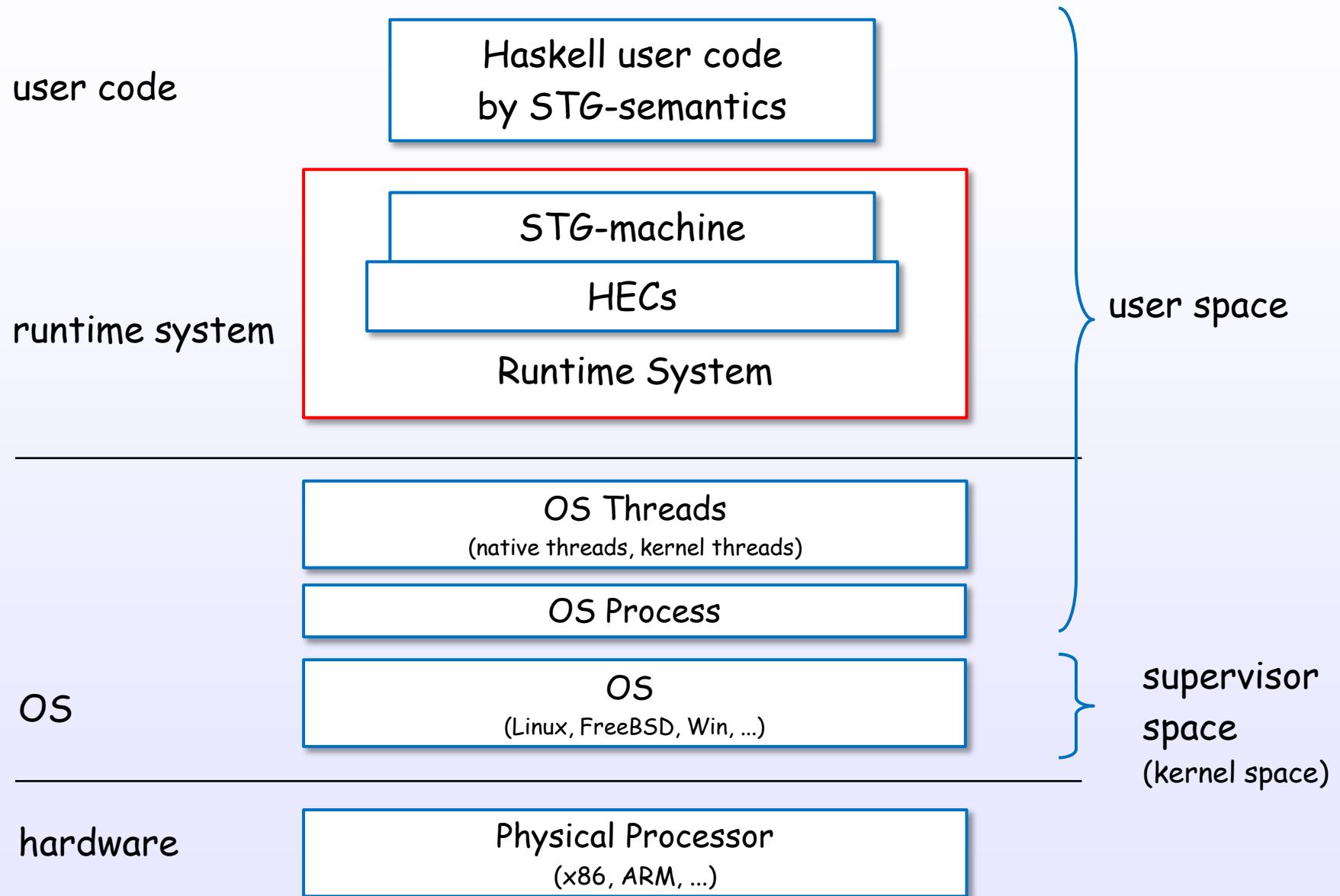
Physical Processor
(x86, ARM, ...)

Each Haskell code is executed in STG semantics.

Machine layer



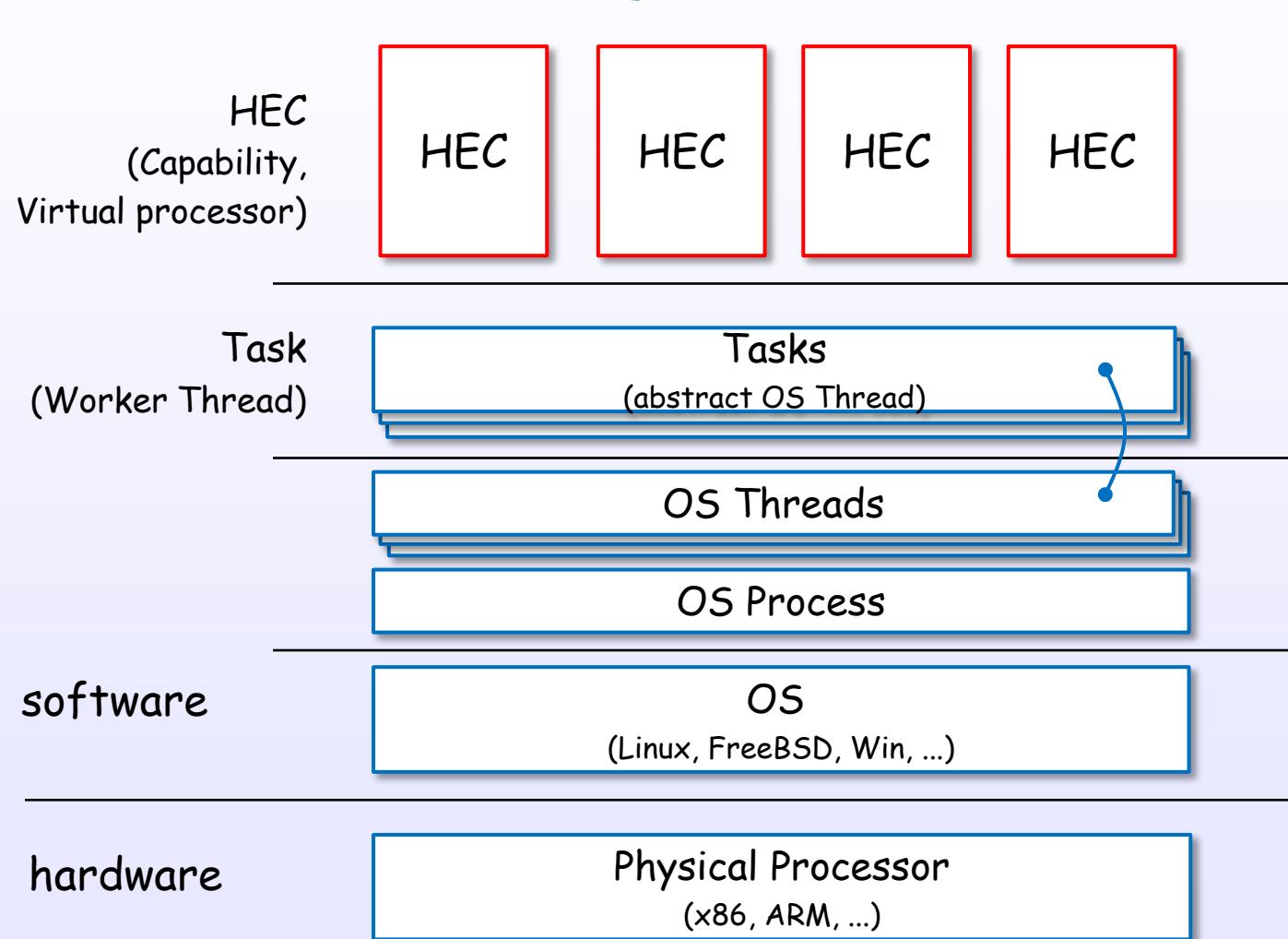
Runtime system and HEC



many HECs

Multi HECs can be generated by compile and runtime options :

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



HEC (Capability) data structure

[rts/Capability.h] (ghc 7.8)

```
struct Capability_ {
    StgFunTable f;
    StgRegTable r; register table
    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;

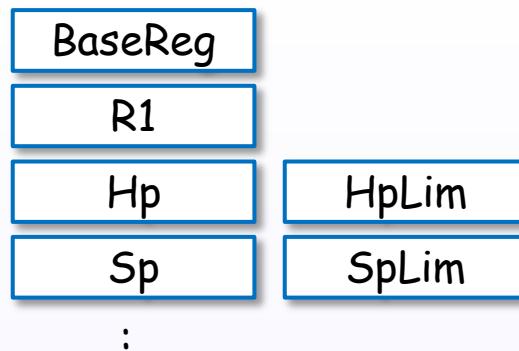
Each HEC (Capability) has a register table and a run queue and ...

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

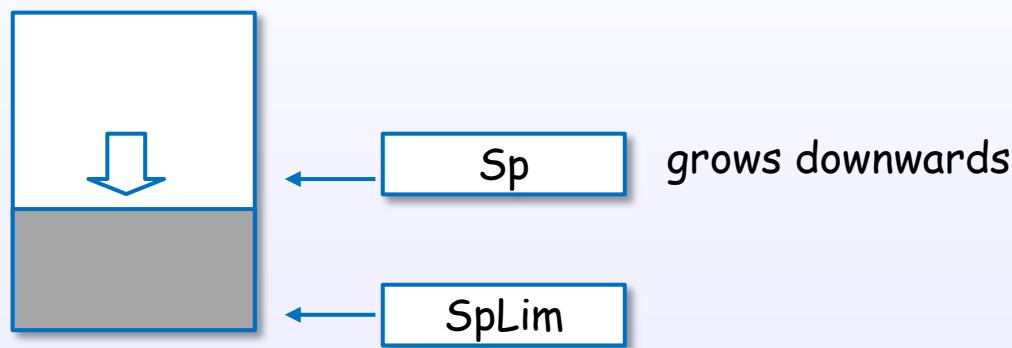
STG-machine

The STG-machine consists of three parts

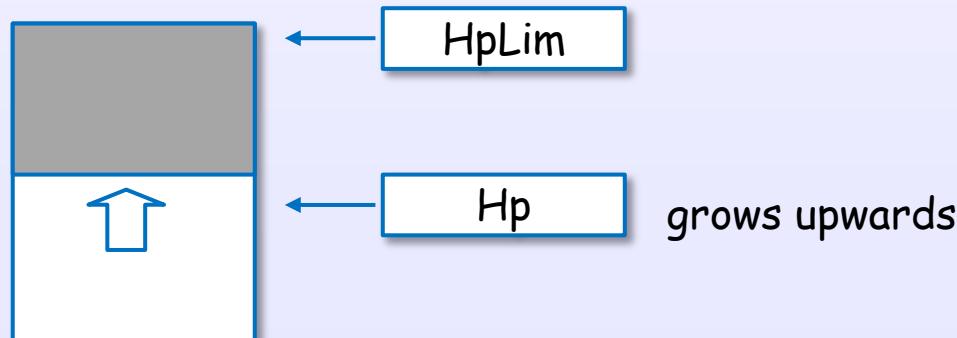
STG Registers



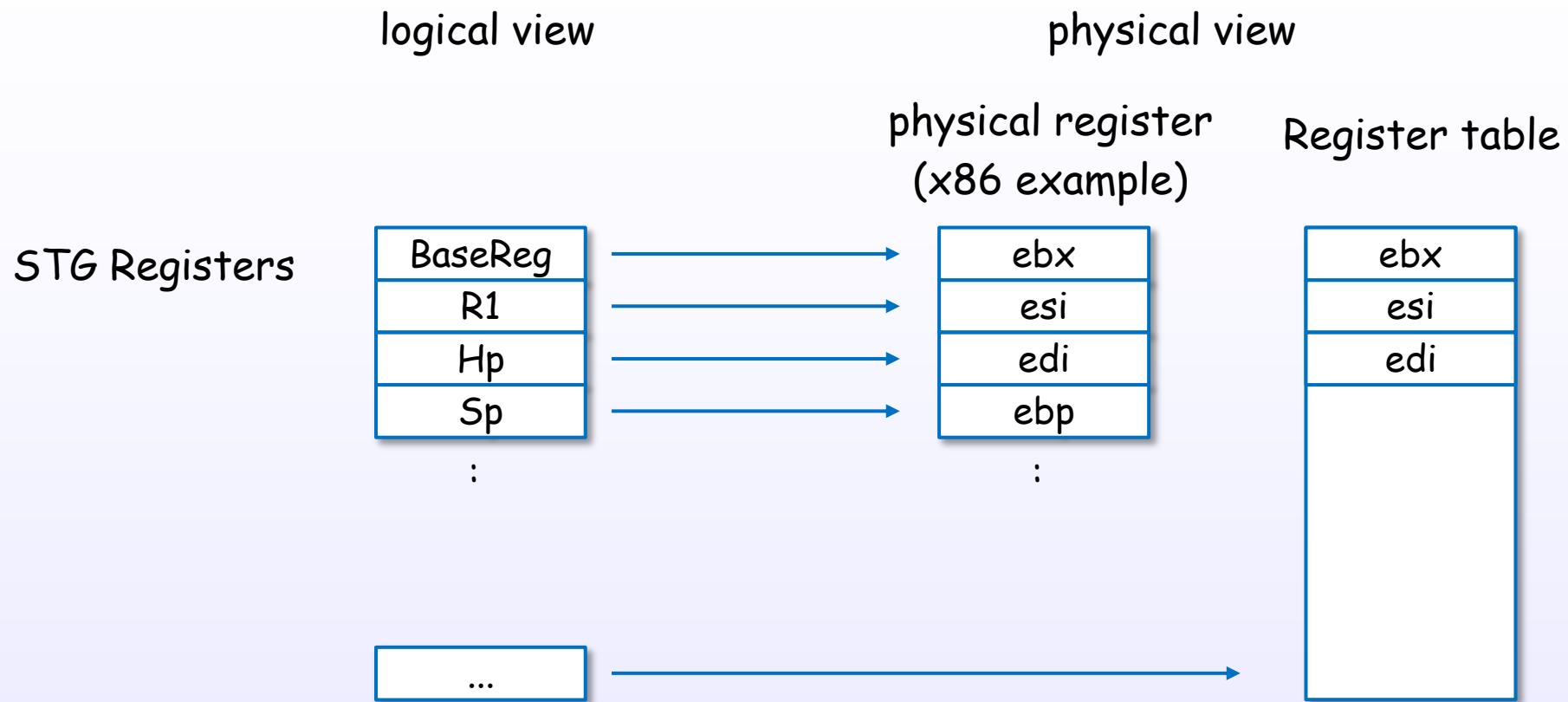
Stack



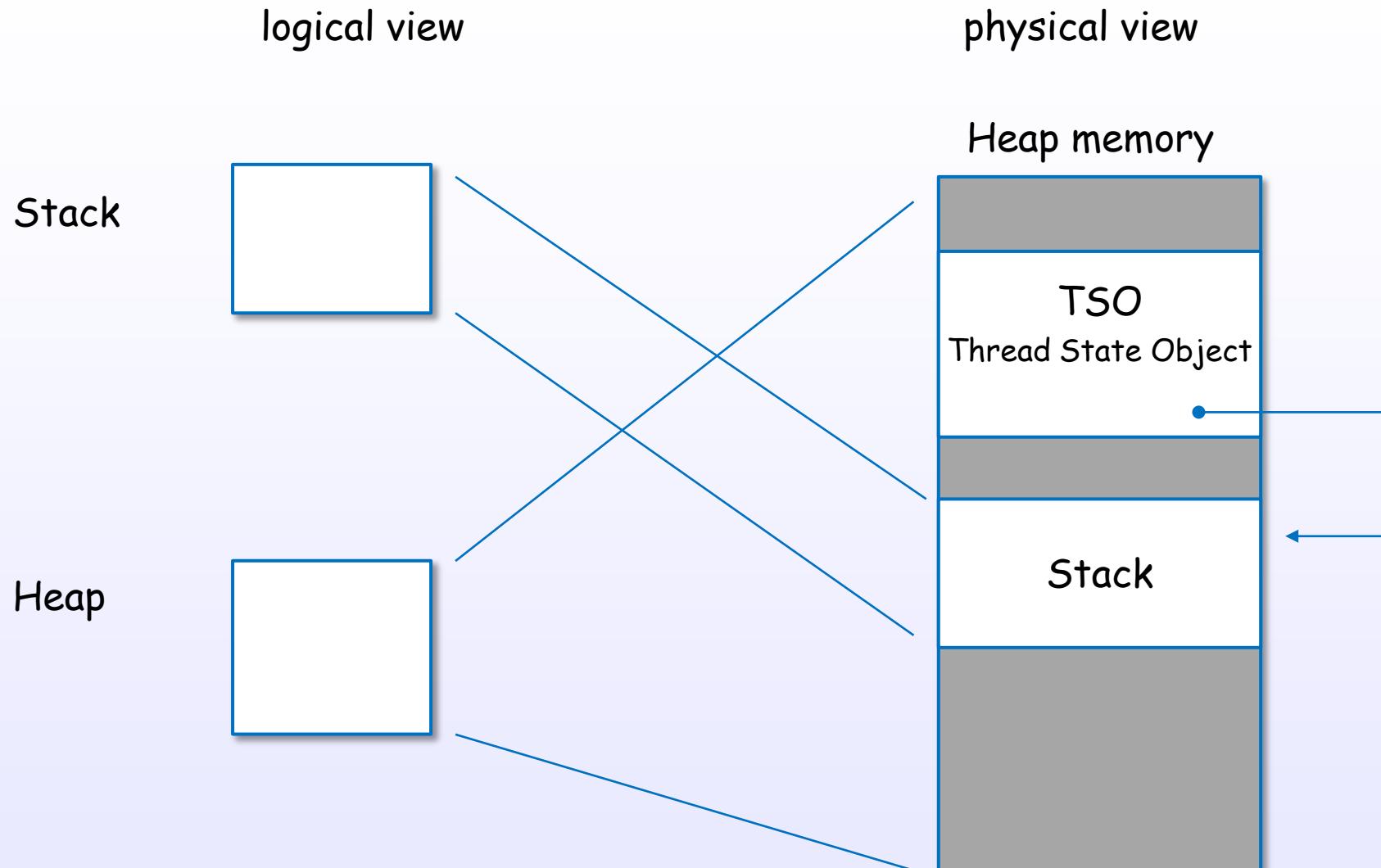
Heap



STG-machine is mapped to physical processor



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.

TSO data structure

[includes/rts/storage/TSO.h] (ghc 7.8)

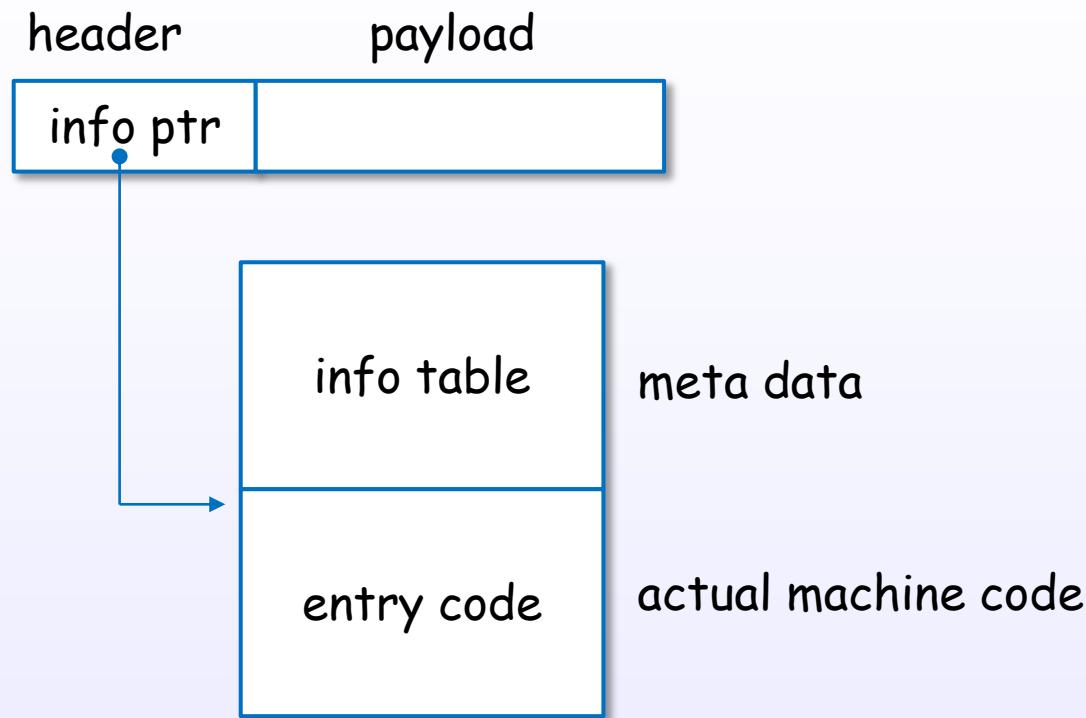
```
typedef struct StgTSO_ {
    StgHeader          header;
    struct StgTSO_*   _link;
    struct StgTSO_*   global_link;
    struct StgStack_* *stackobj; ← link to stack object
    StgWord16          what_next;
    StgWord16          why_blocked;
    StgWord32          flags;
    StgTSOBlockInfo   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;
```

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

References : [S5]

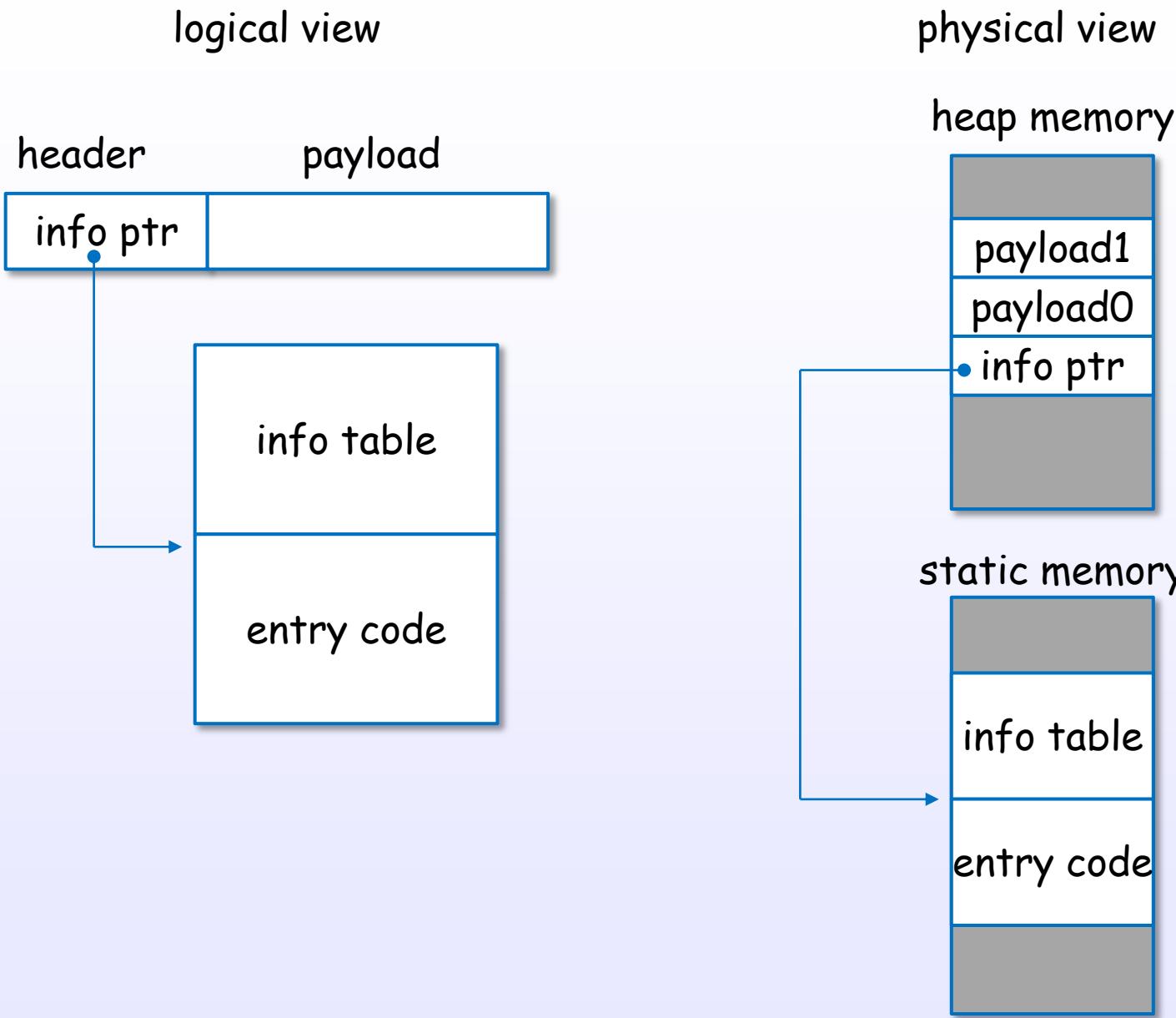
Heap objects in STG-machine

Every heap object is represented uniformly

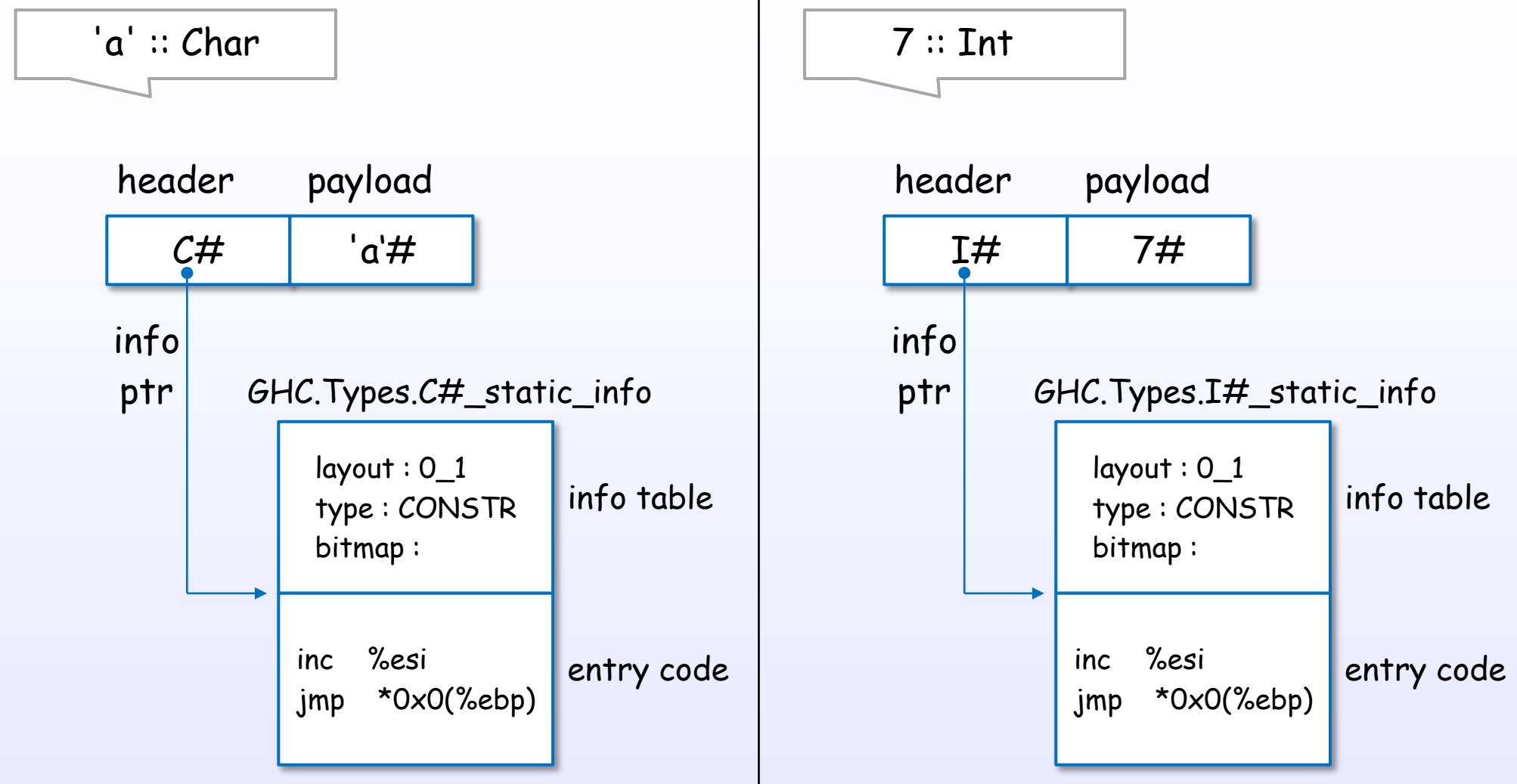


Closure (header + payload) + Info Table + Entry Code

Heap object (closure)



Closure examples : Char, Int



Closure example (code)

[Example.hs]

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

STG

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-stg Example.hs]

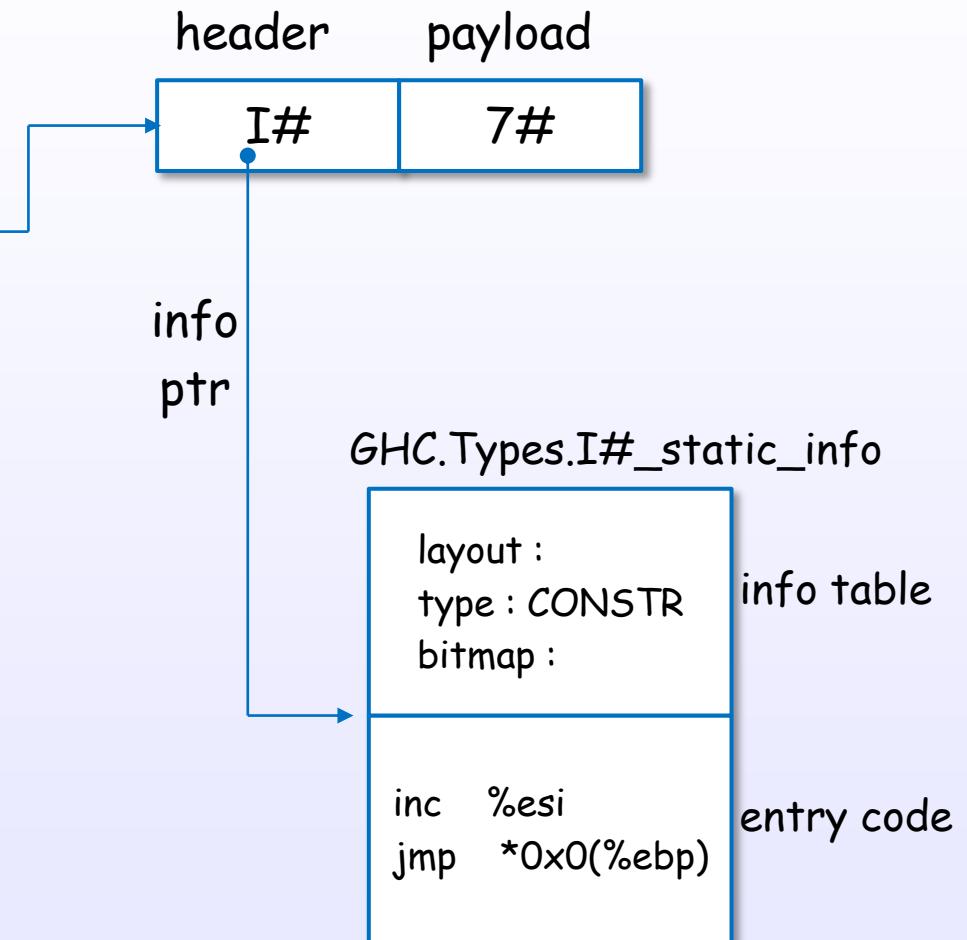
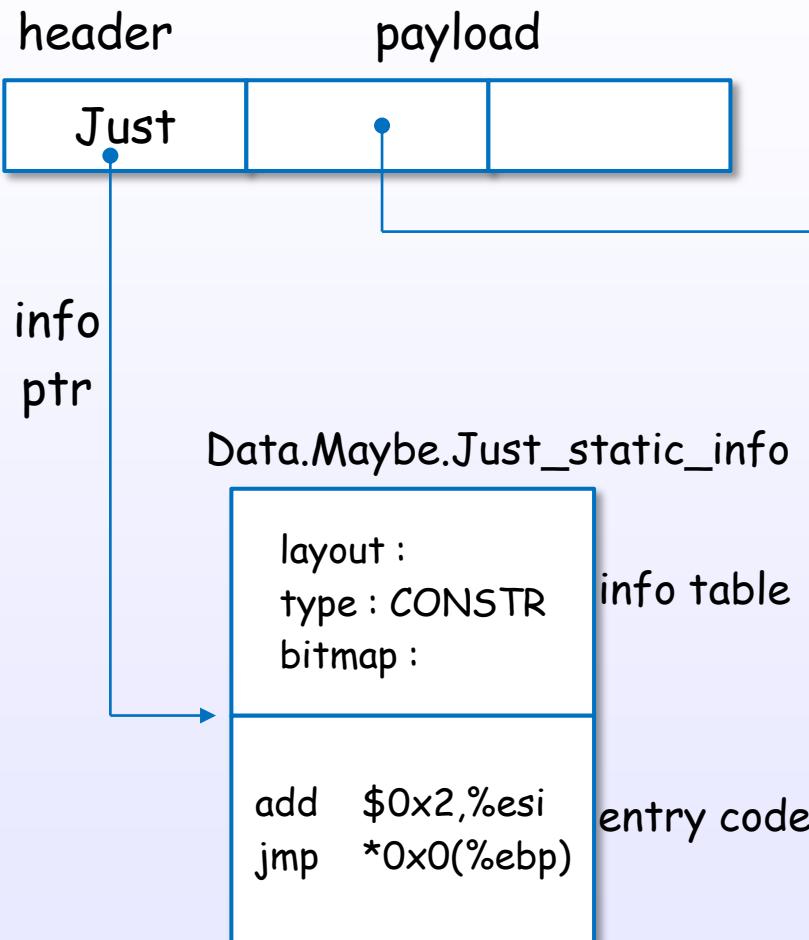
```
Example.value1 :: GHC.Types.Int  
[GblId, Caf=NoCafRefs, Str=DmdType m, Unf=OtherCon []] =  
    NO_CCS GHC.Types.I#! [8];
```

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

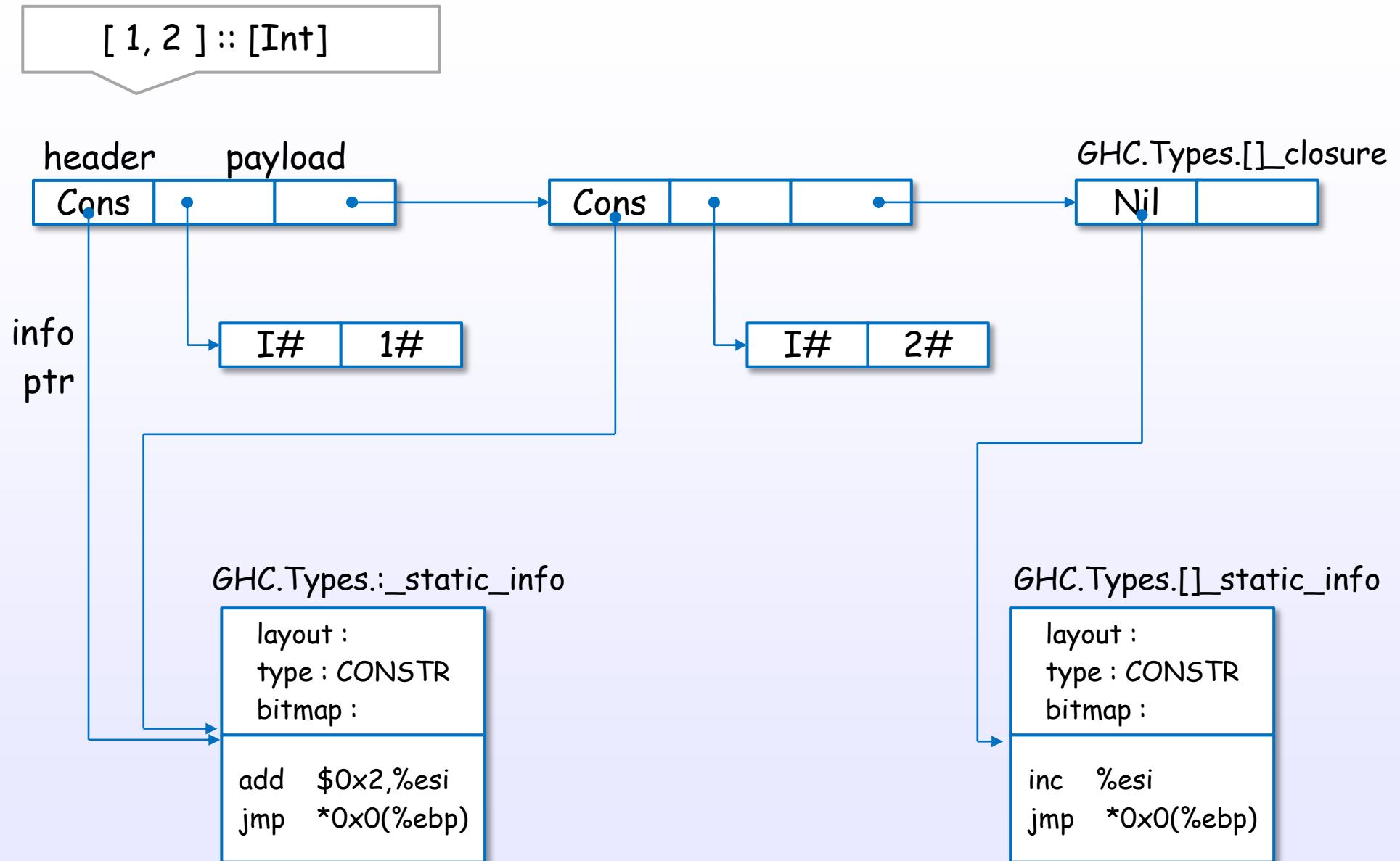
```
.data  
    .align 4  
.align 1  
.globl __stginit_main:Example  
__stginit_main:Example:  
.data  
    .align 4  
.align 1  
.globl Example.value1_closure  
Example.value1_closure:  
    .long GHC.Types.I#_static_info  
    long 7  
.section .data  
    .align 4  
.align 1  
SMd_srt:  
  
header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

Closure examples : Maybe

Just 7 :: Maybe Int



Closure examples : List



Closure examples : Thunk

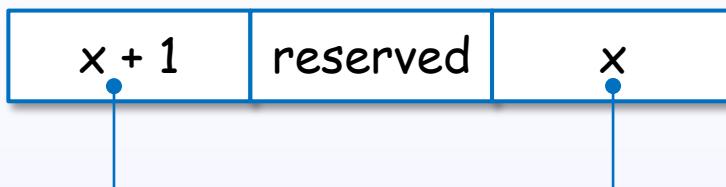
"thunk"

$x + 1 :: \text{Int}$

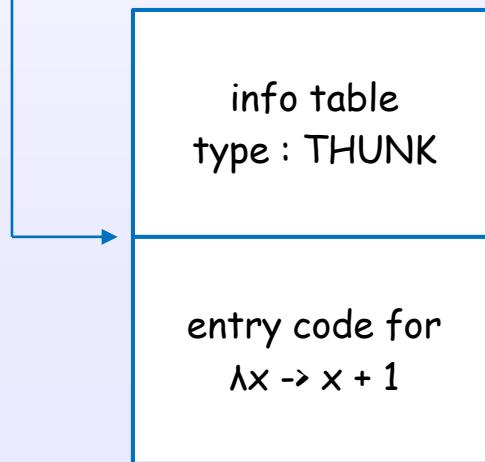
(free variable : $x = 7$)

header

payload

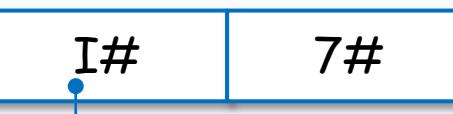


info
ptr

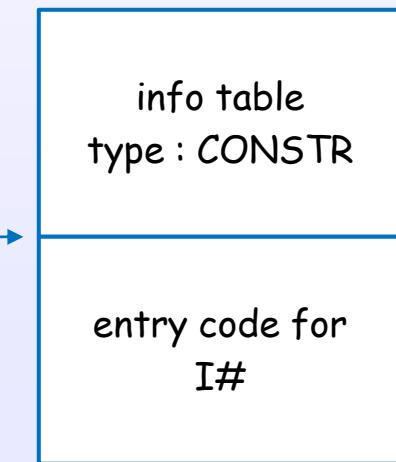


header

payload

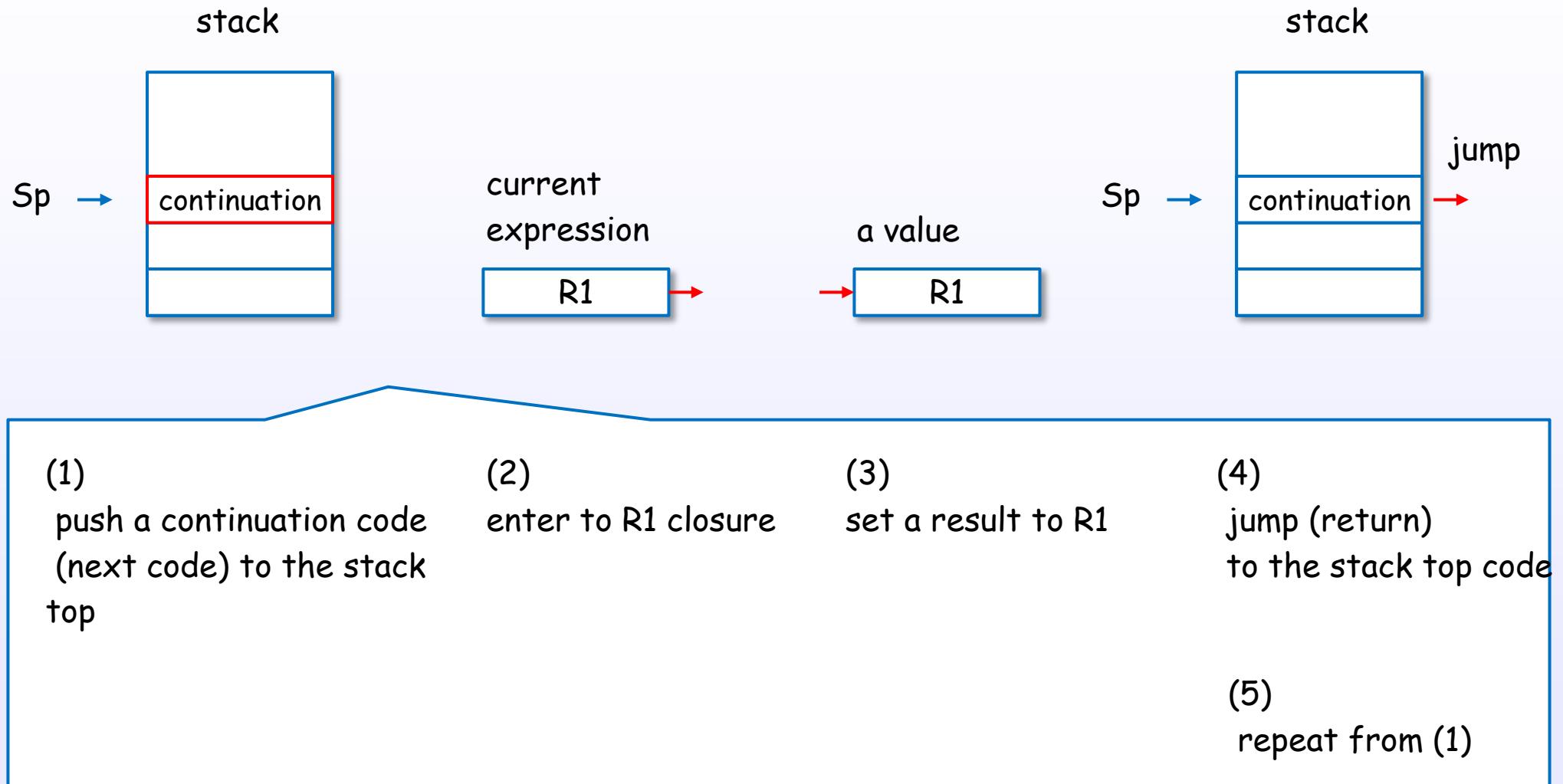


info
ptr

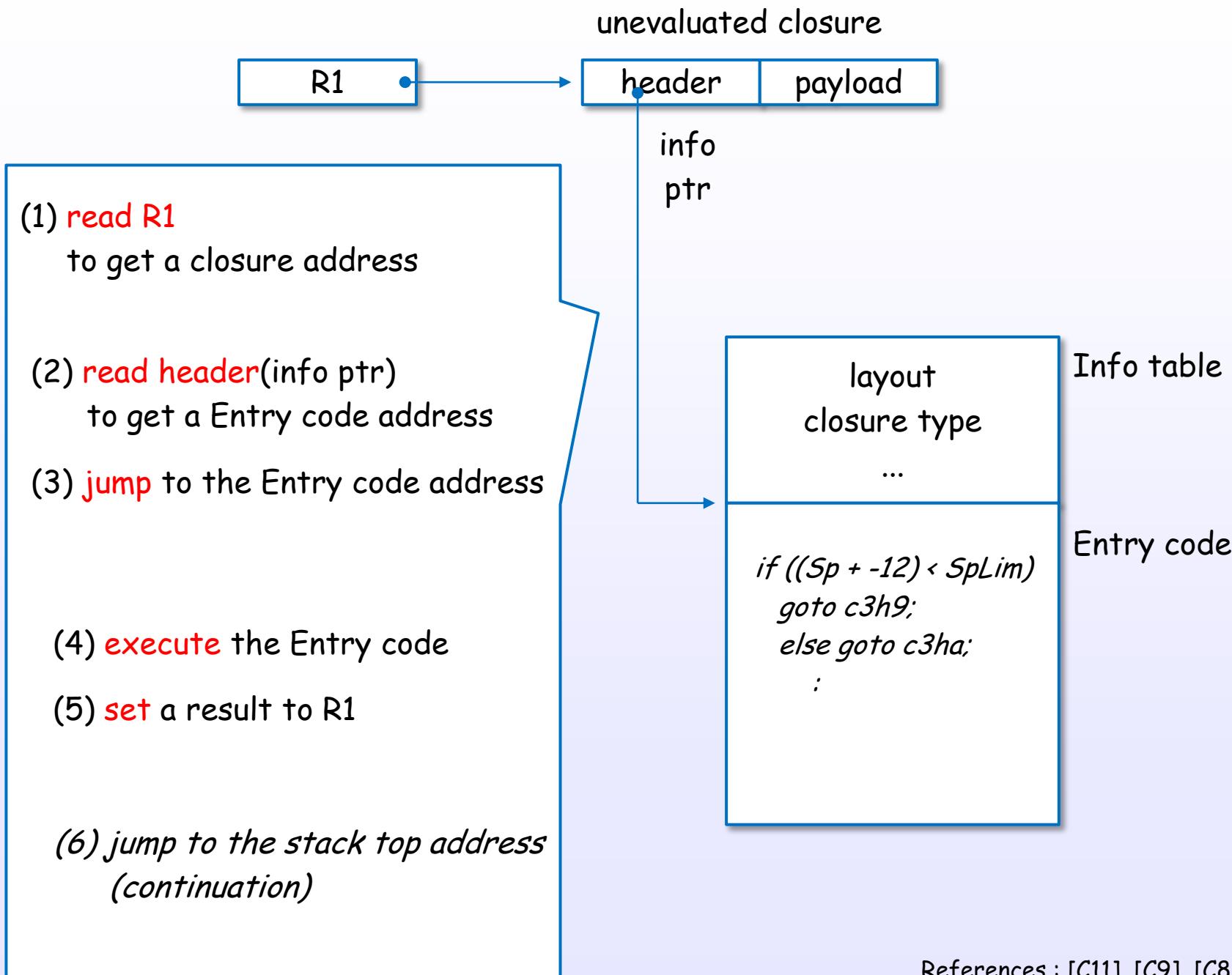


STG-machine evaluation

STG evaluation flow

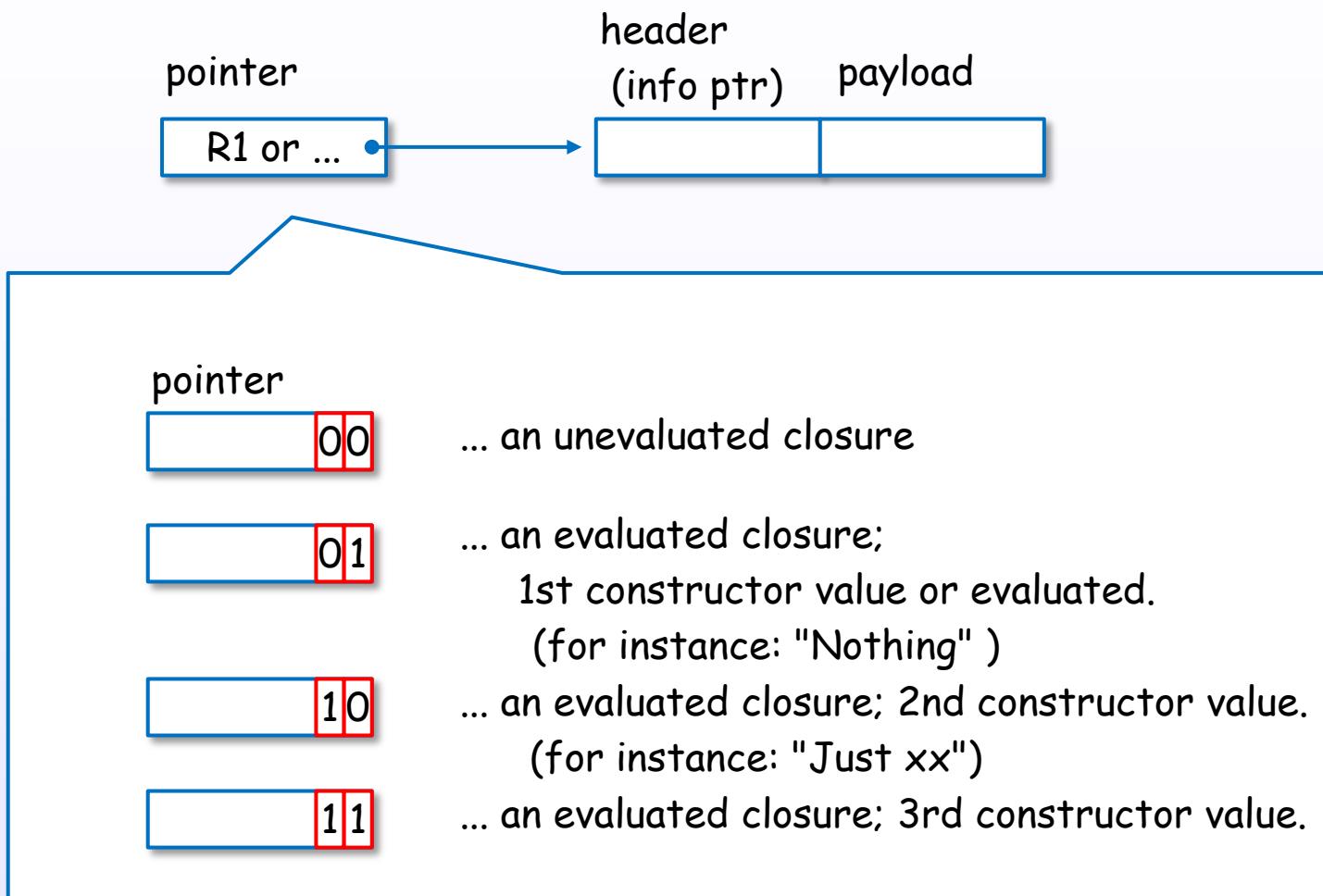


Enter to a closure



Pointer tagging

Pointer tagging



* 32bit machine case

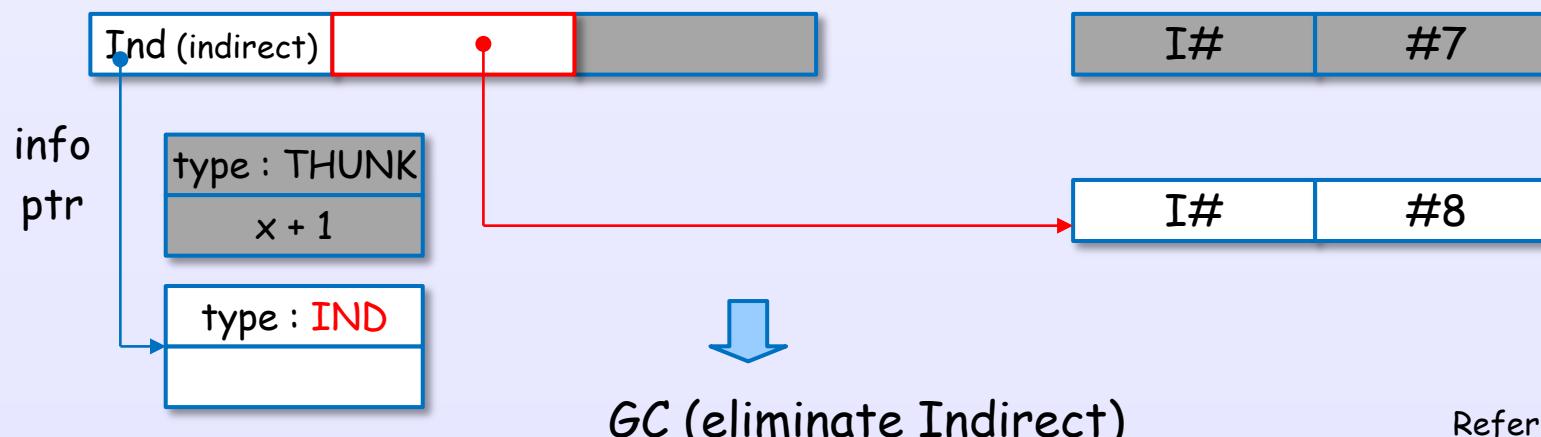
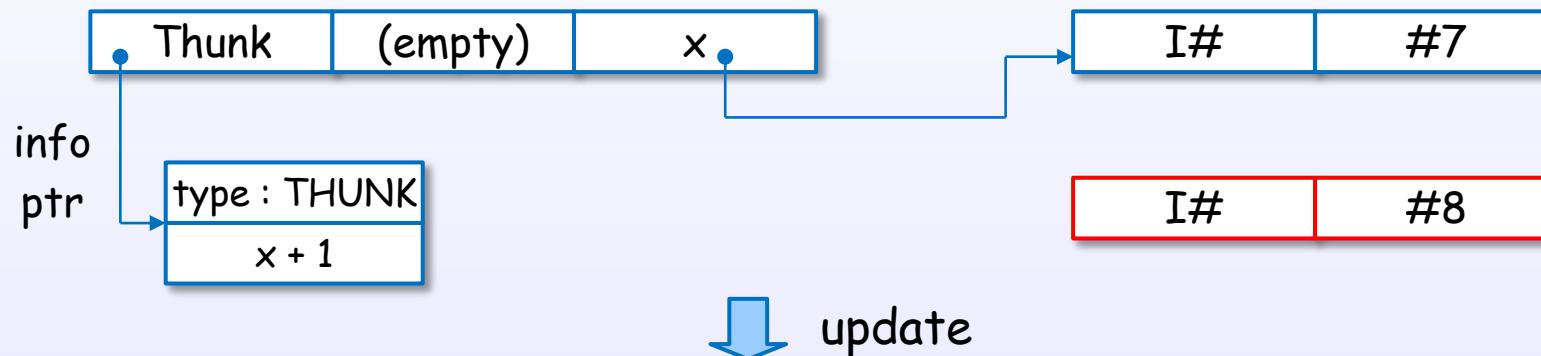
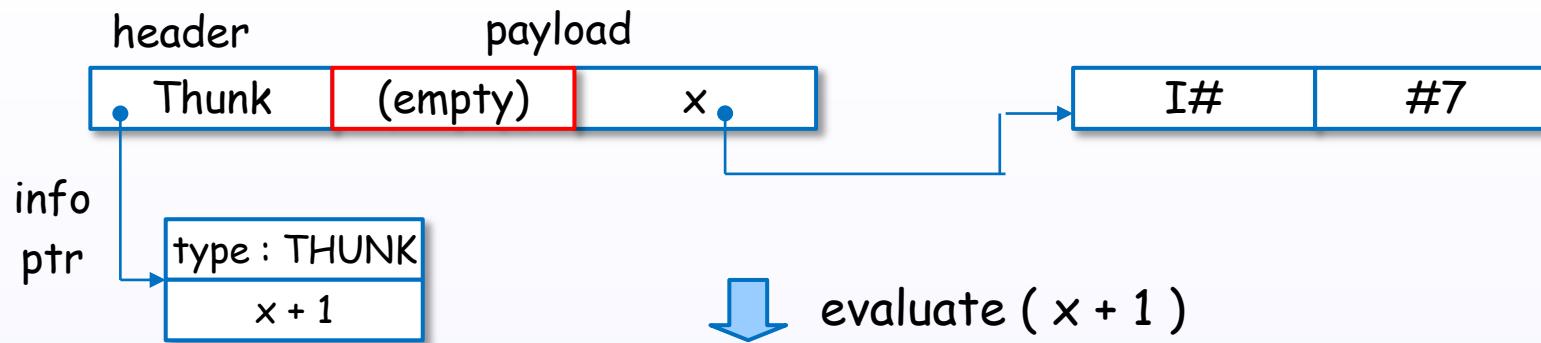
fast judgment!

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

Thunk and update

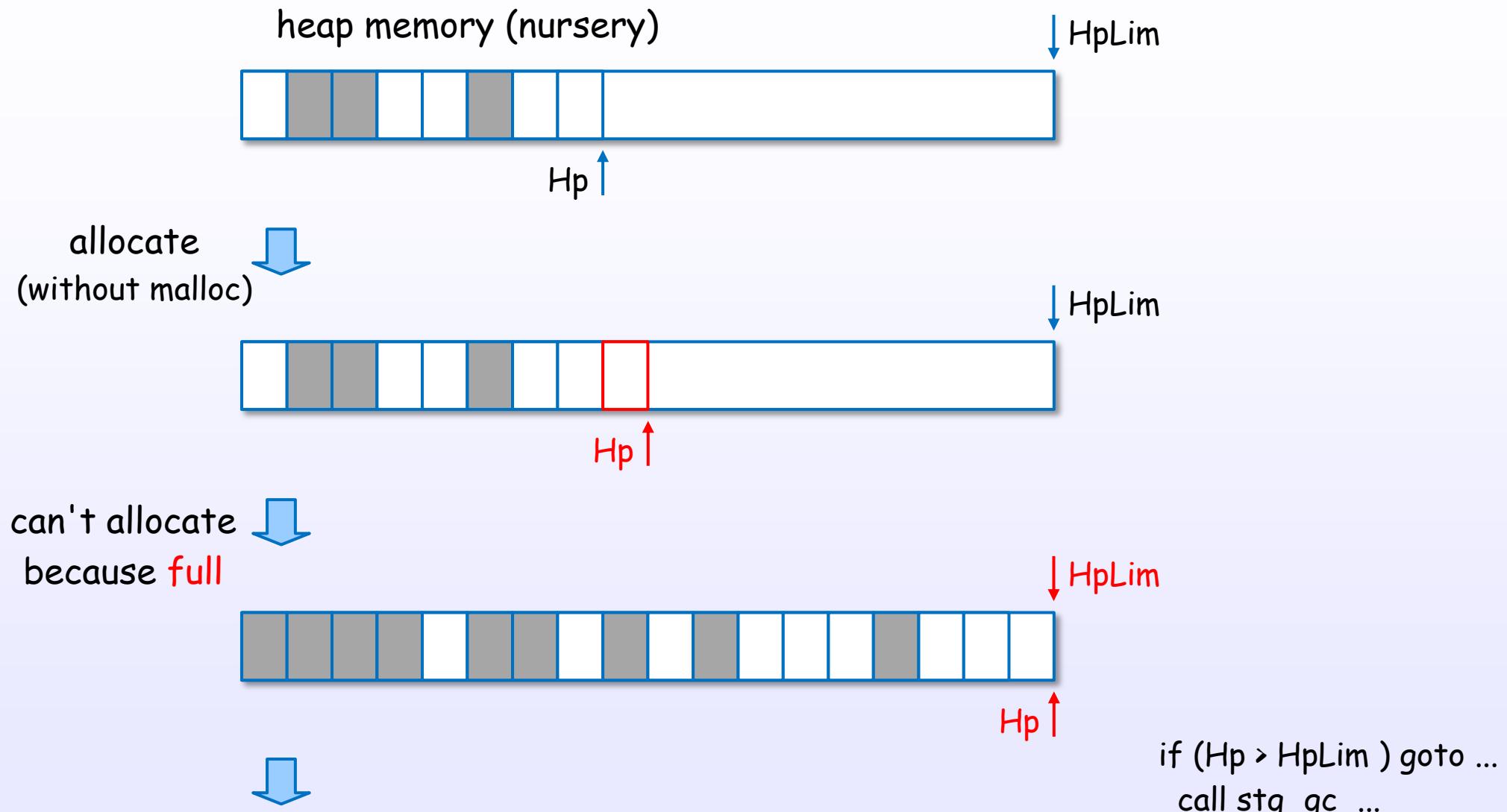
Thunk and update

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

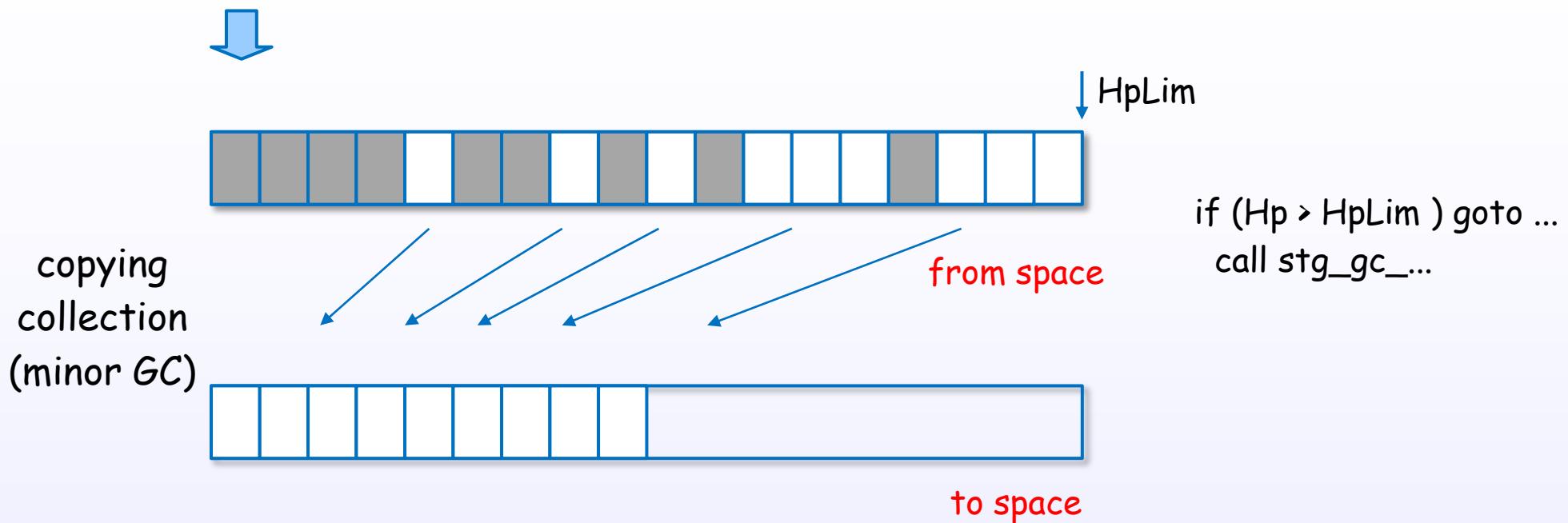


Allocate and free heap objects

Allocate heap objects

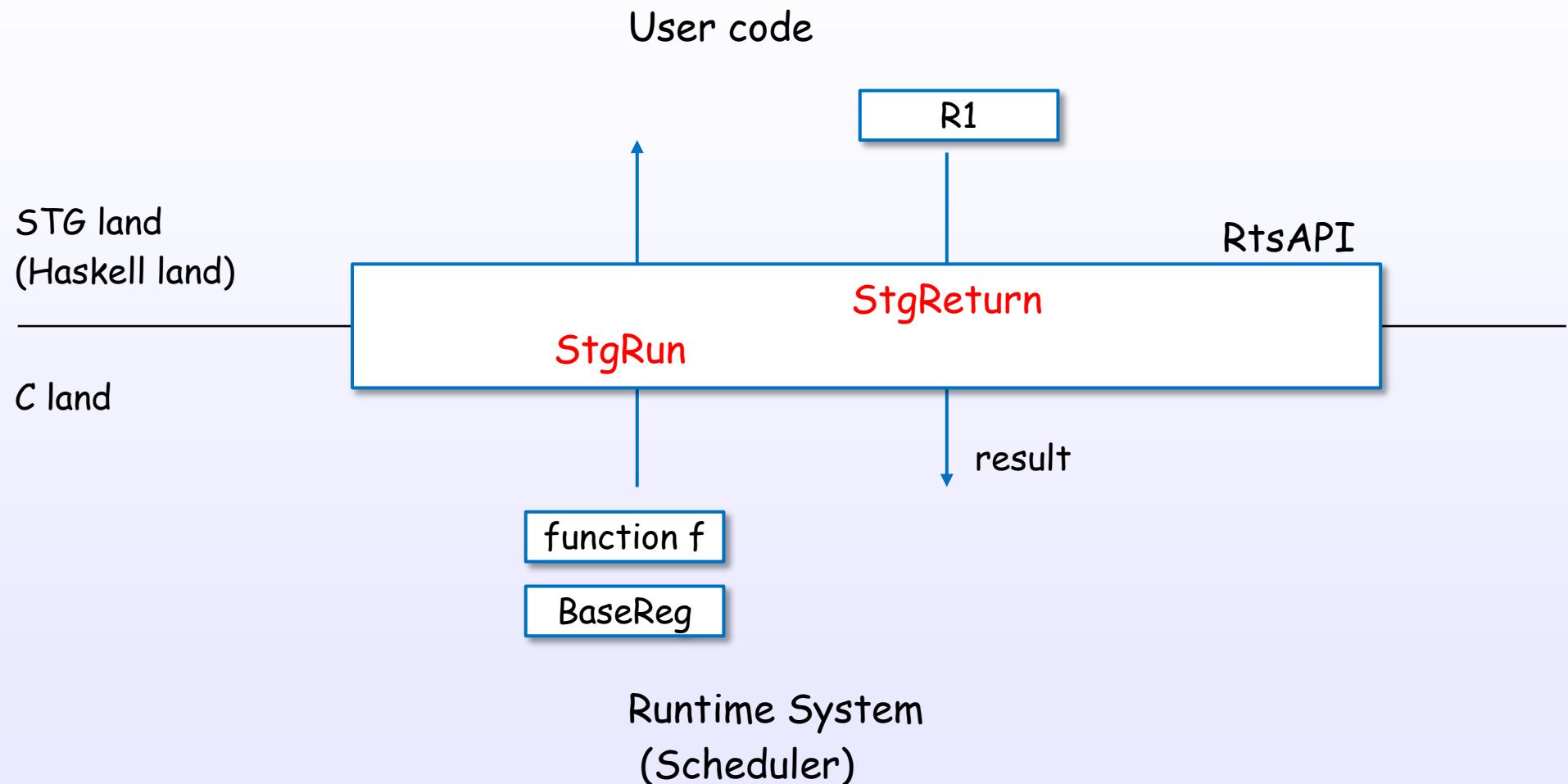


free and collect heap objects



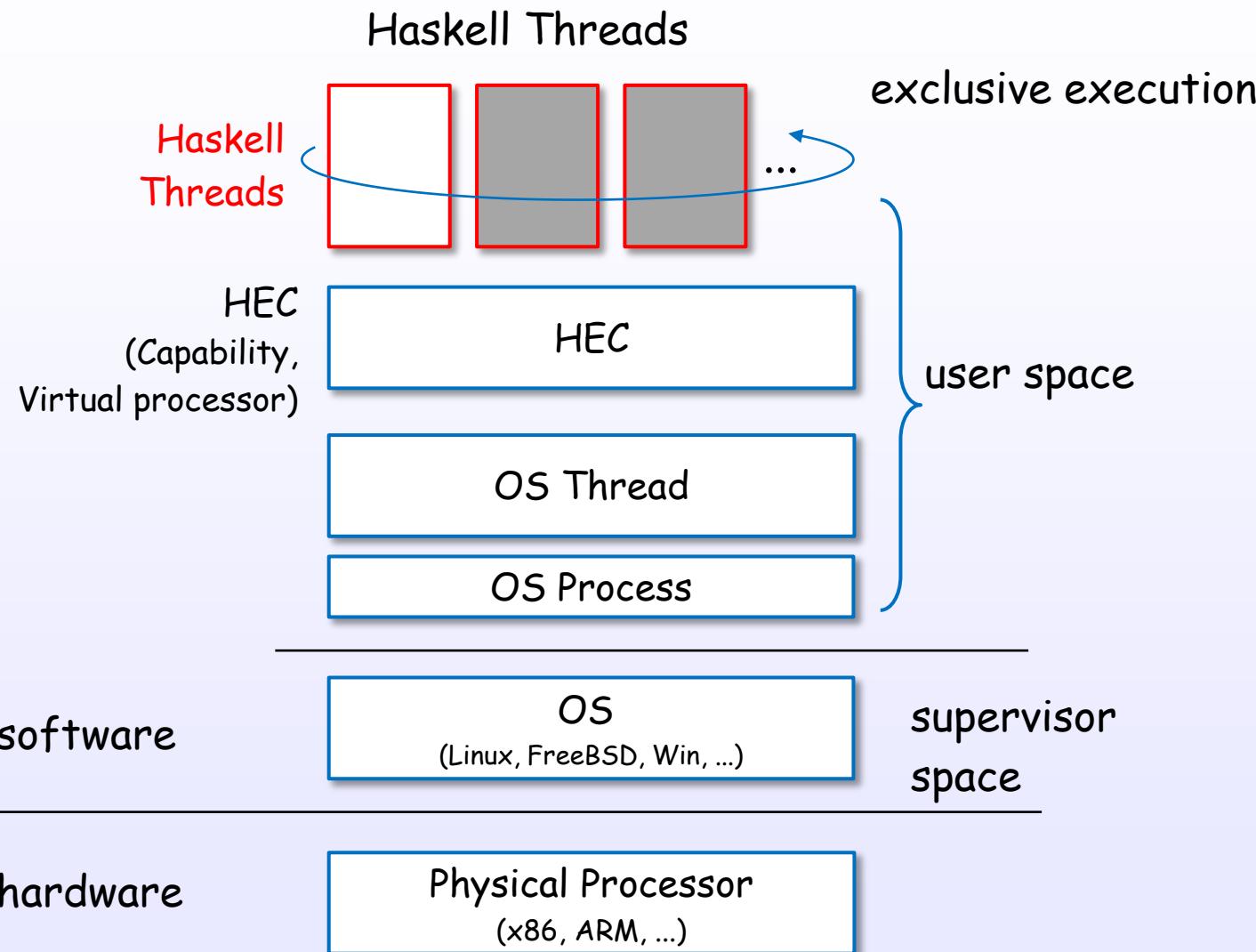
STG - C land interface

STG (Haskell) land - C land interface

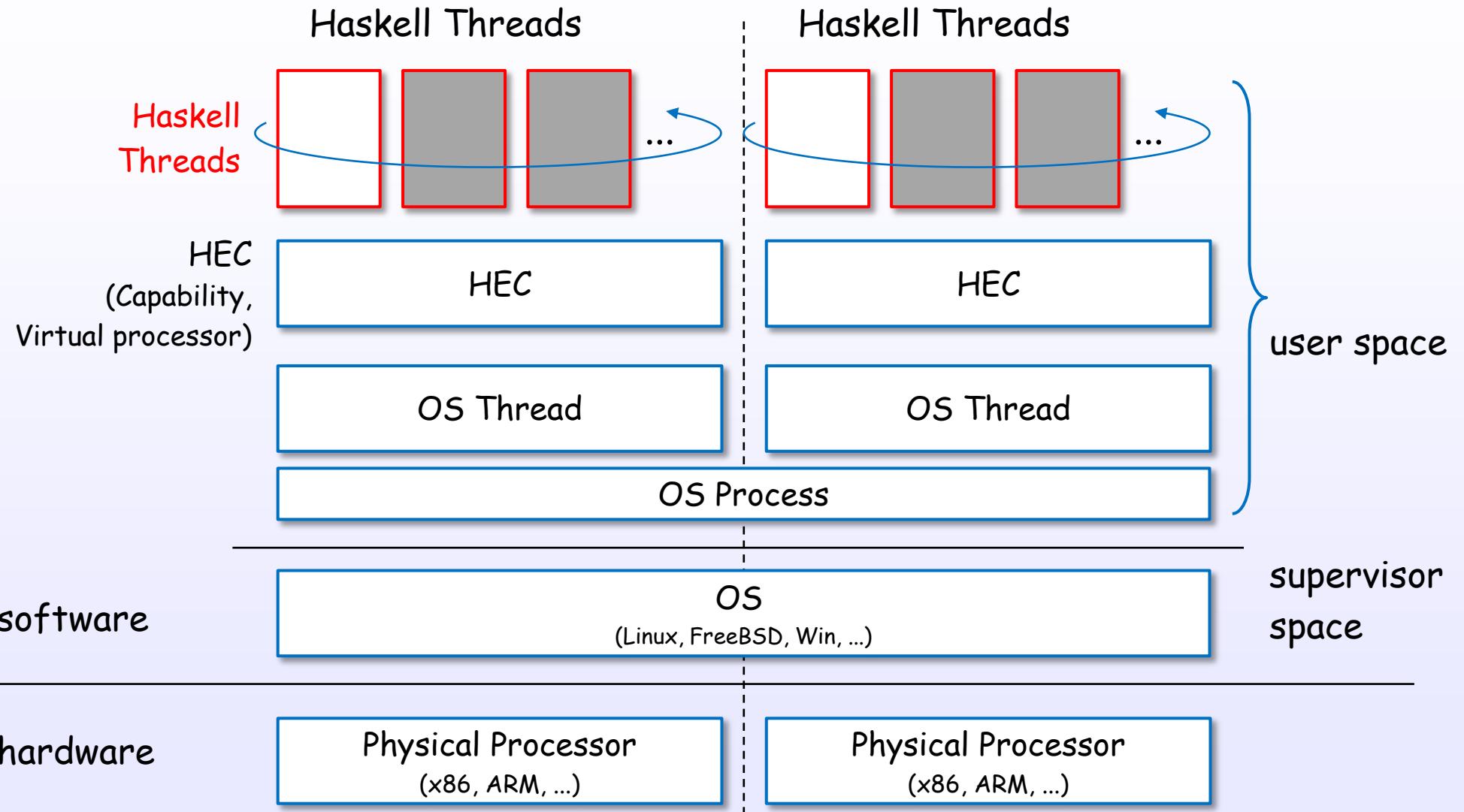


Thread

Thread layer (single core)



Thread layer (multi core)

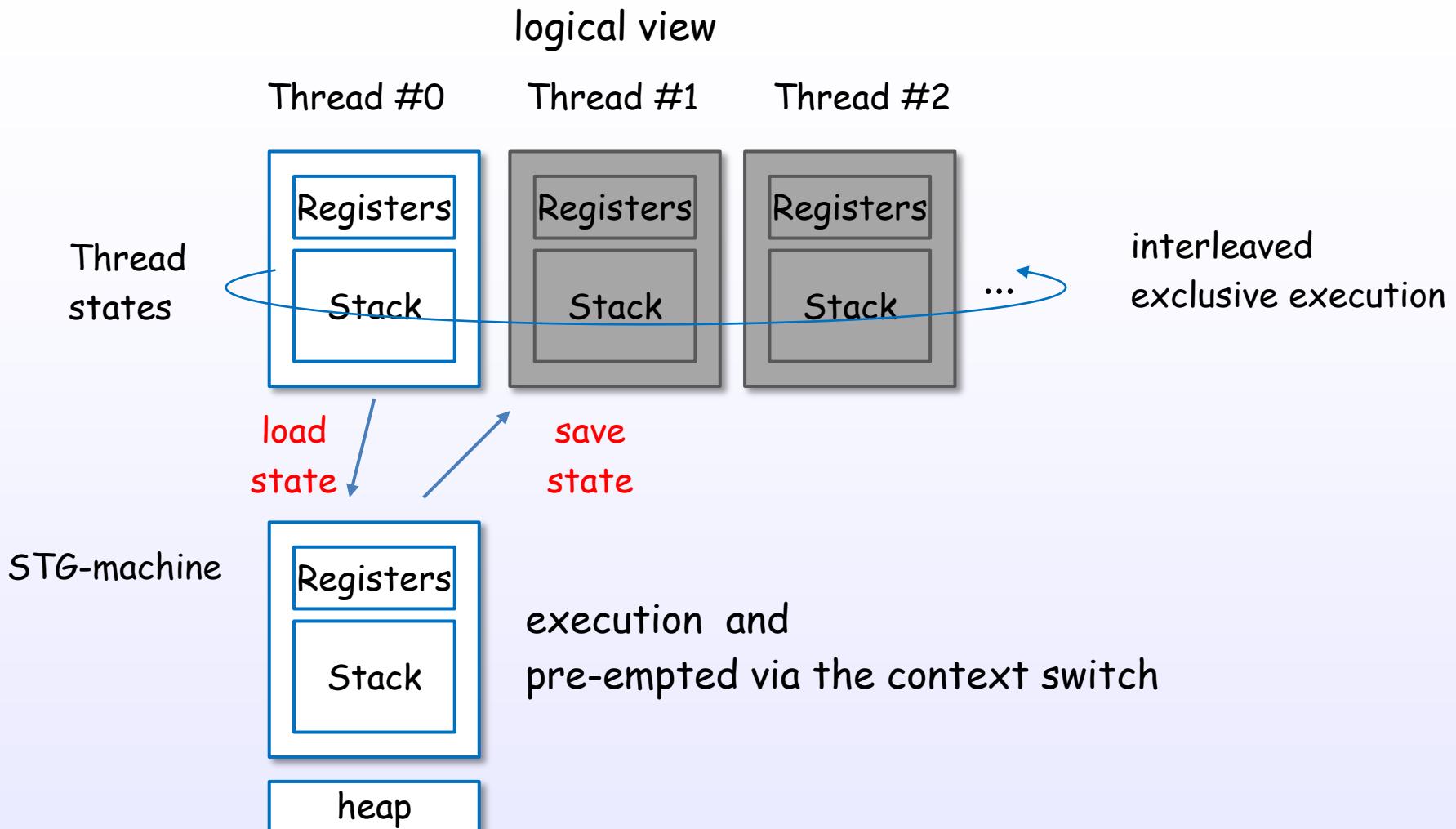


*Threaded option case (ghc -threaded)

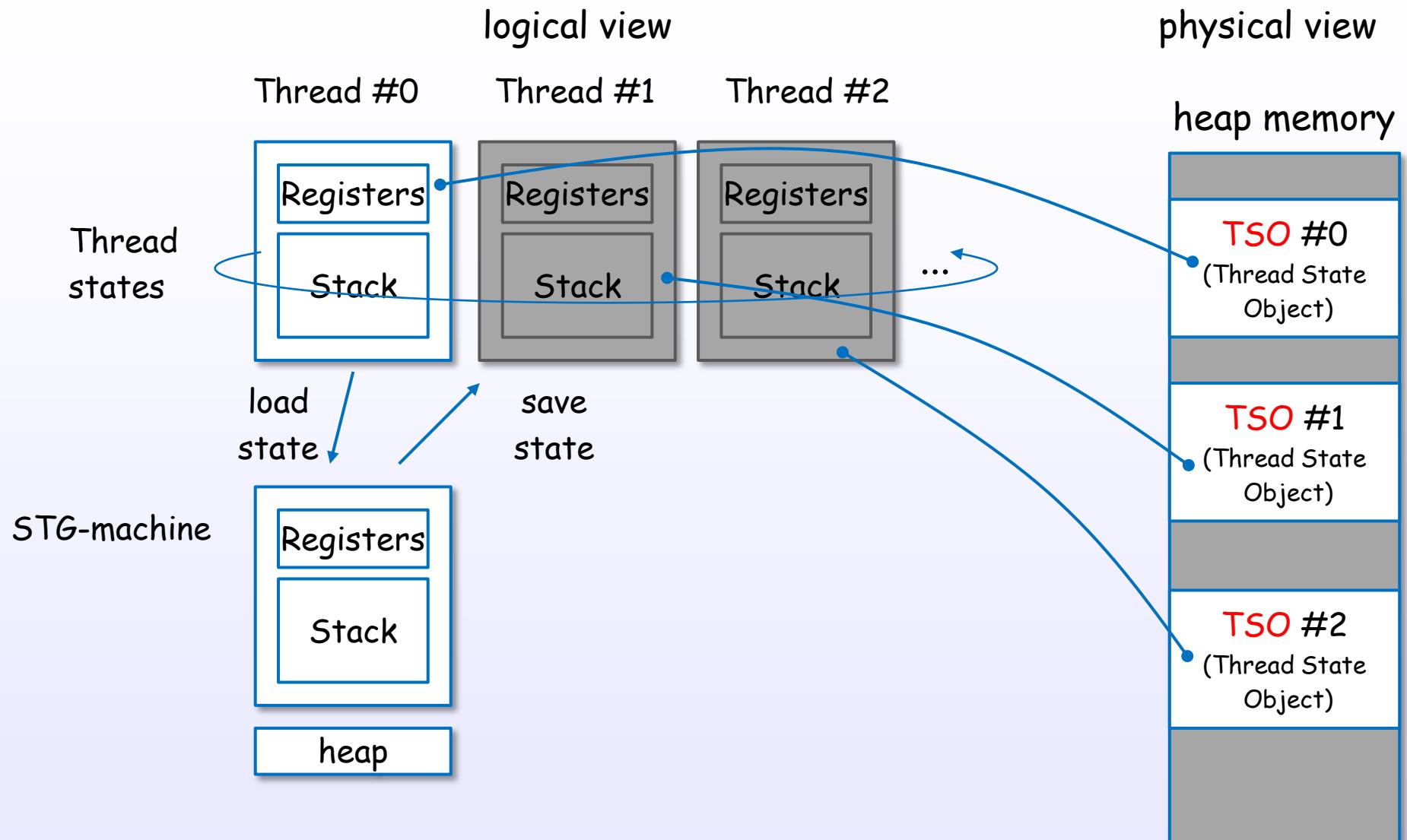
References : [5], [8], [9], [14], [C17], [C11], [19], [S17], [S16], [S23], [S22], [S14]

Thread context switch

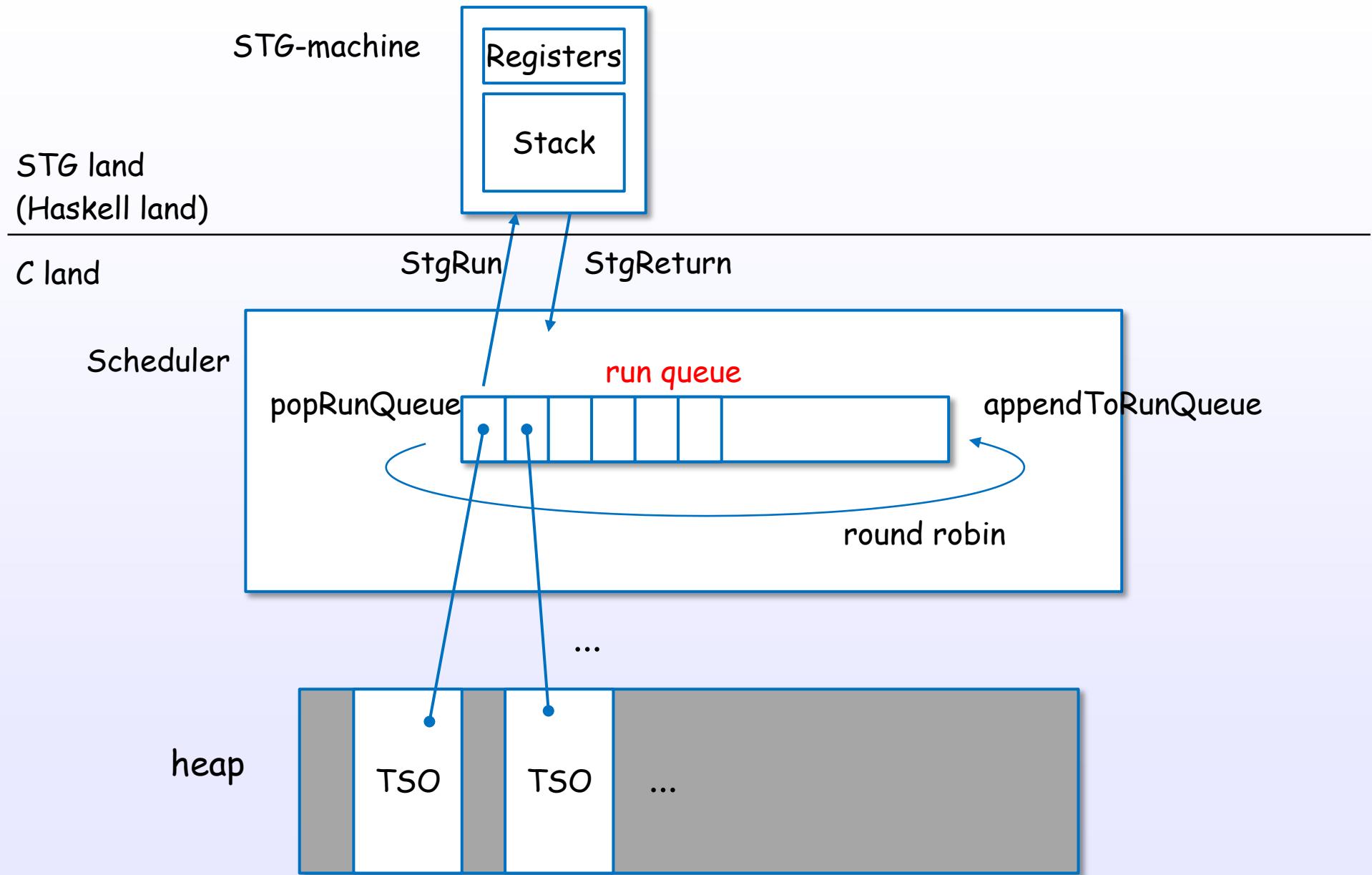
Threads and context switch



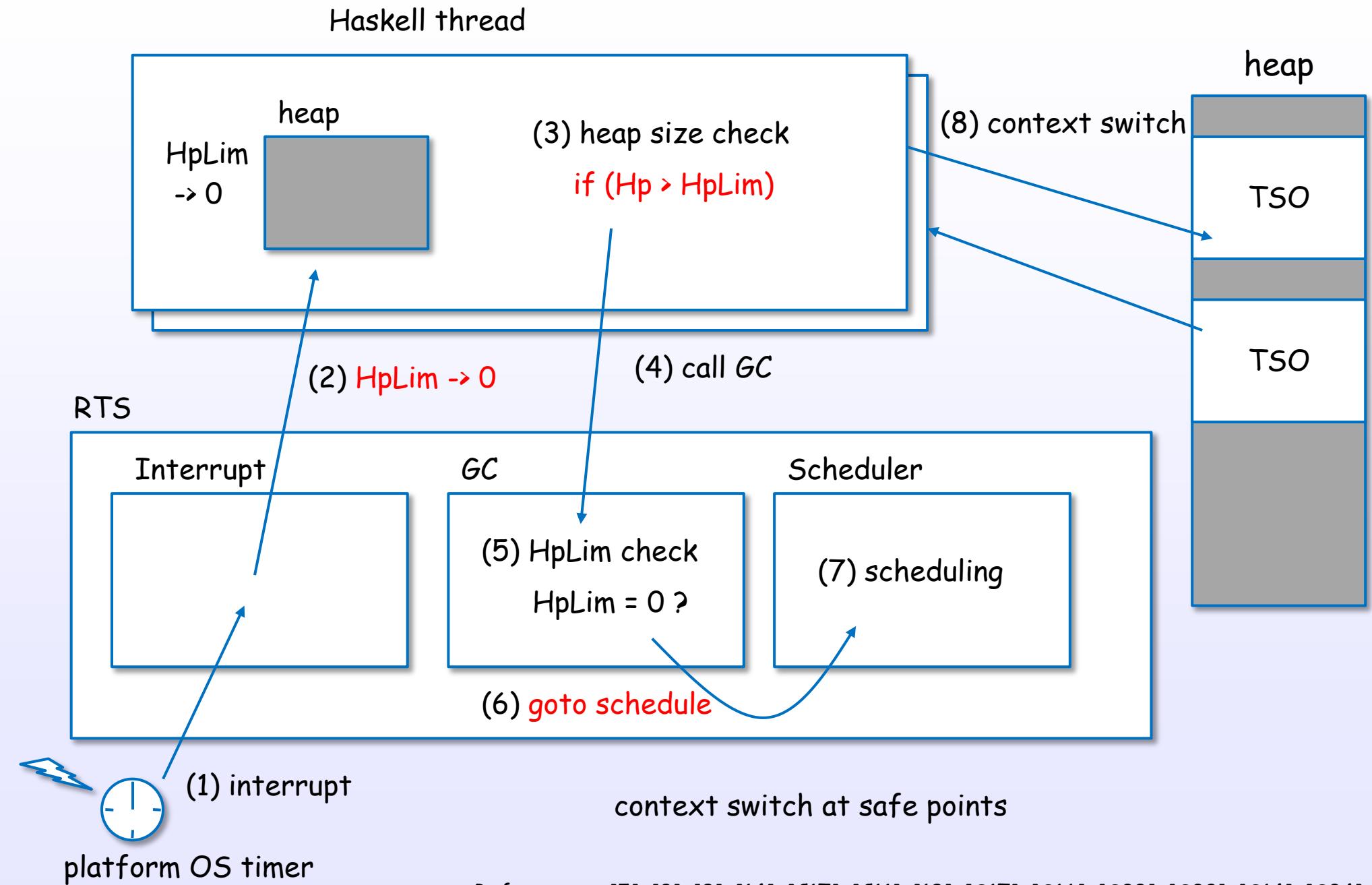
Threads and TSOs



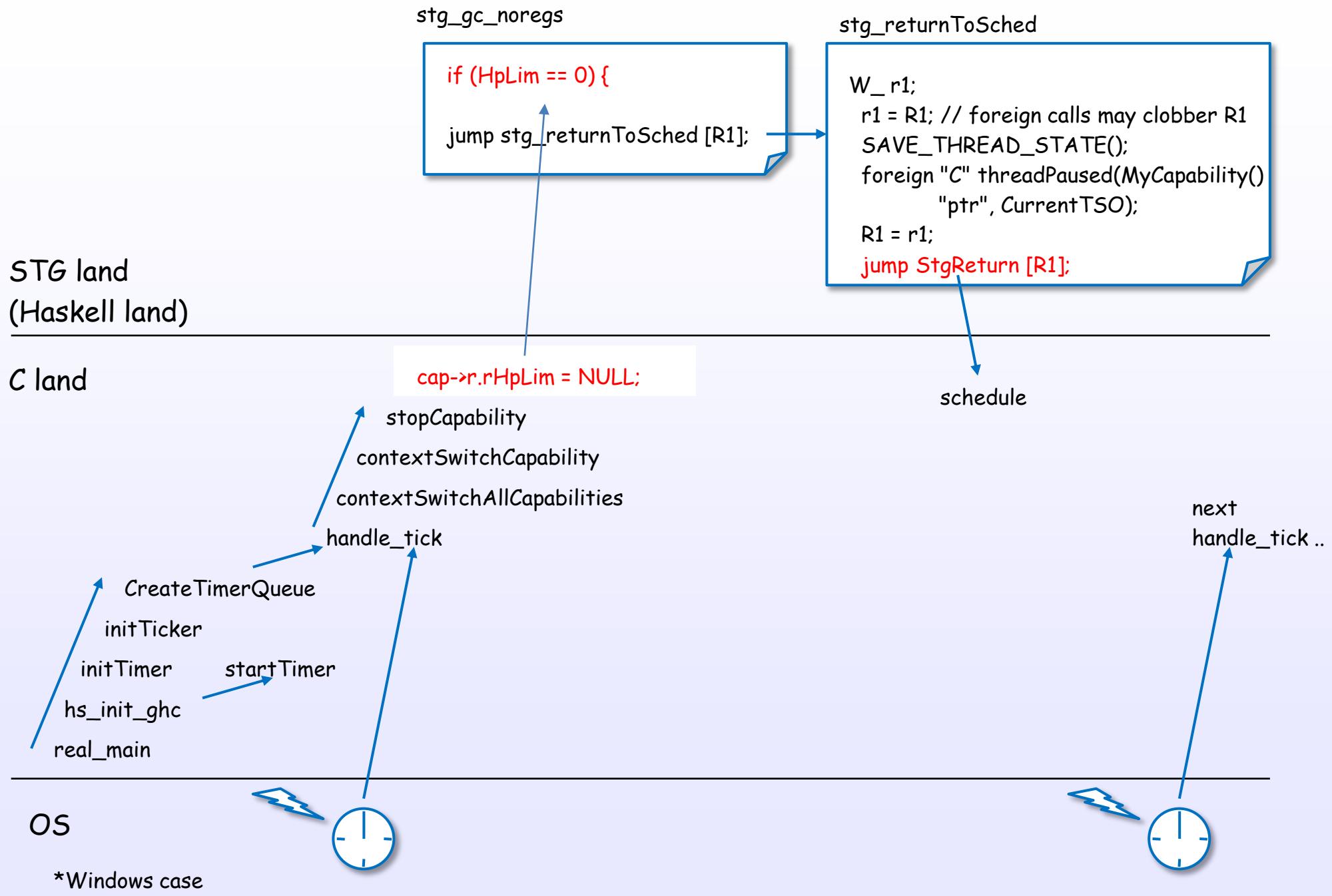
Scheduling by run queue



Context switch flow



Context switch flow (code)



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)
```

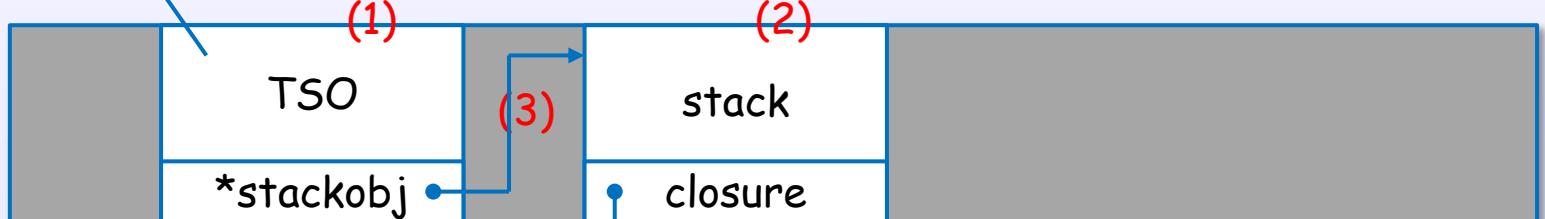
scheduler

run queue



(5)

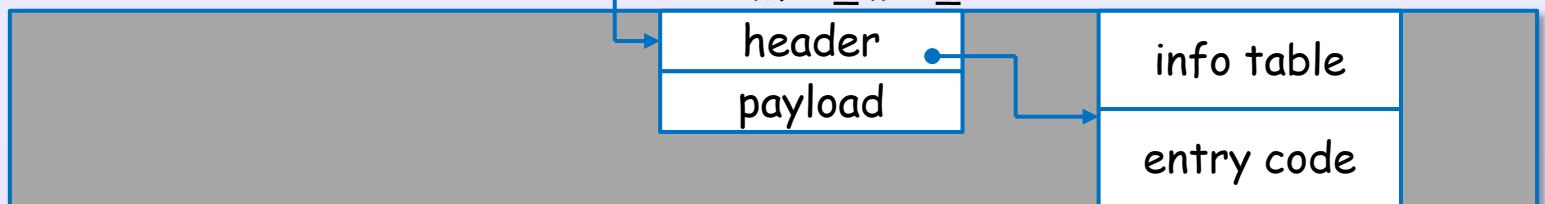
heap memory



(2)

(3)

static memory



ZCMain_main_closure

header

payload

info table

entry code

Create a sub thread using forkIO

Haskell Threads

forkIO

stg_forkzh

ccall `createIOThread ... (1), (2), (3), (4)`

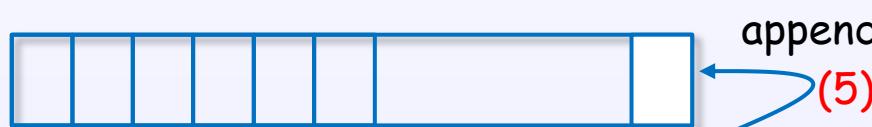
ccall `scheduleThread ... (5)`

User code

Runtime System

scheduler

run queue



(1)

TSO

(2)

stack

heap memory

*stackobj

(3)

closure

forked closure

header
payload

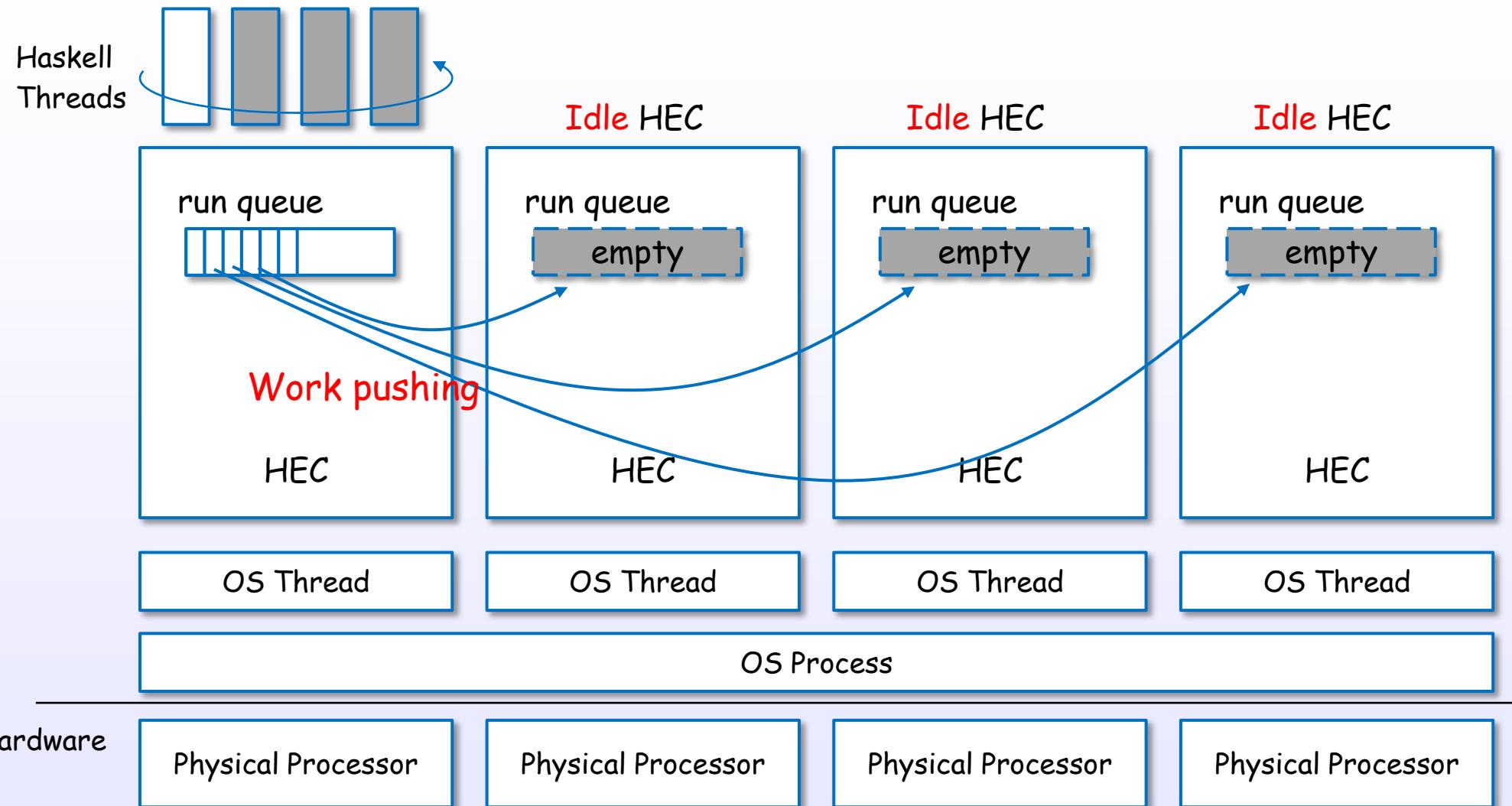
(4)

static memory

info table
entry code

Thread migration

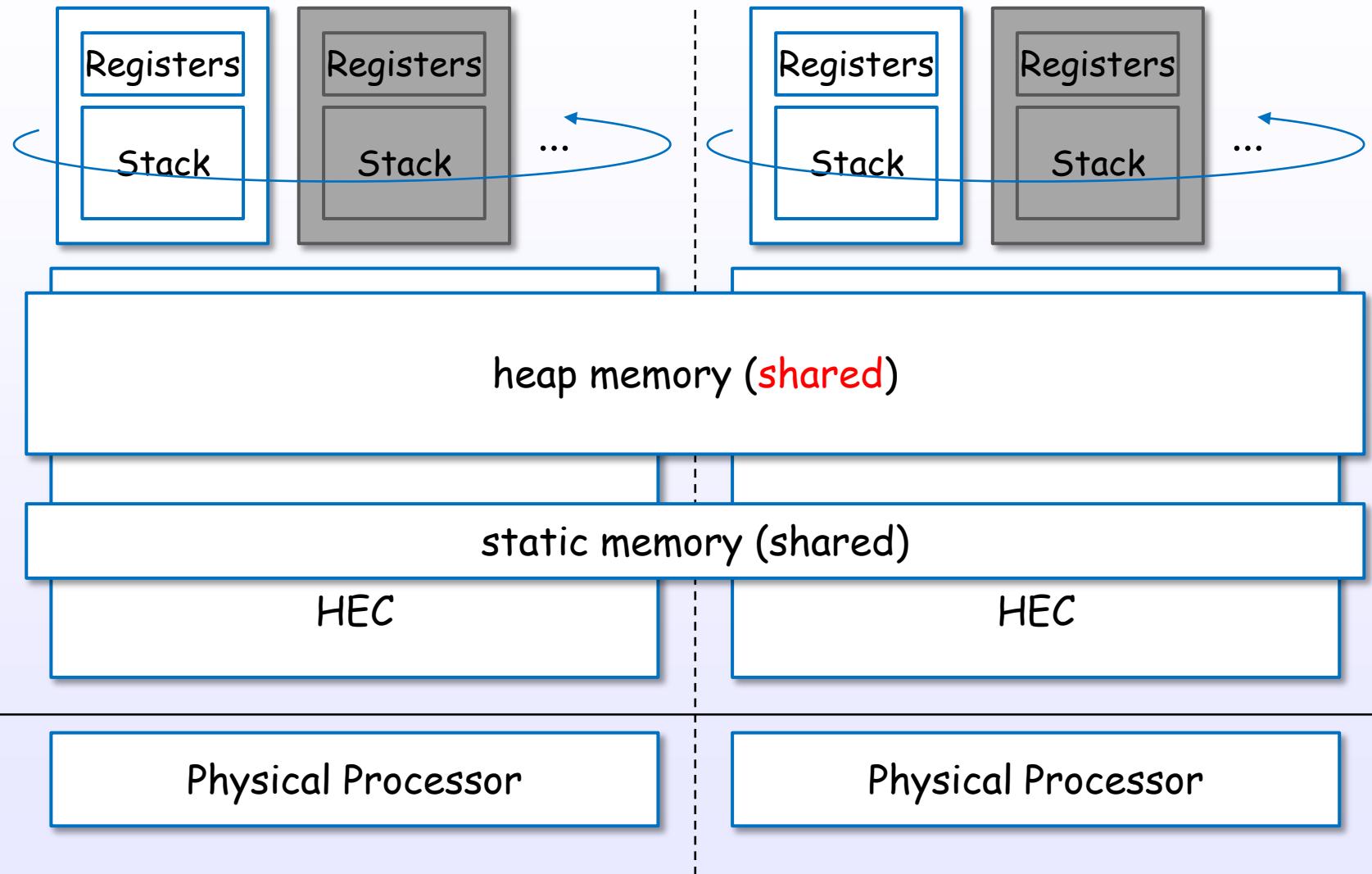
Threads are migrated to idle HECs



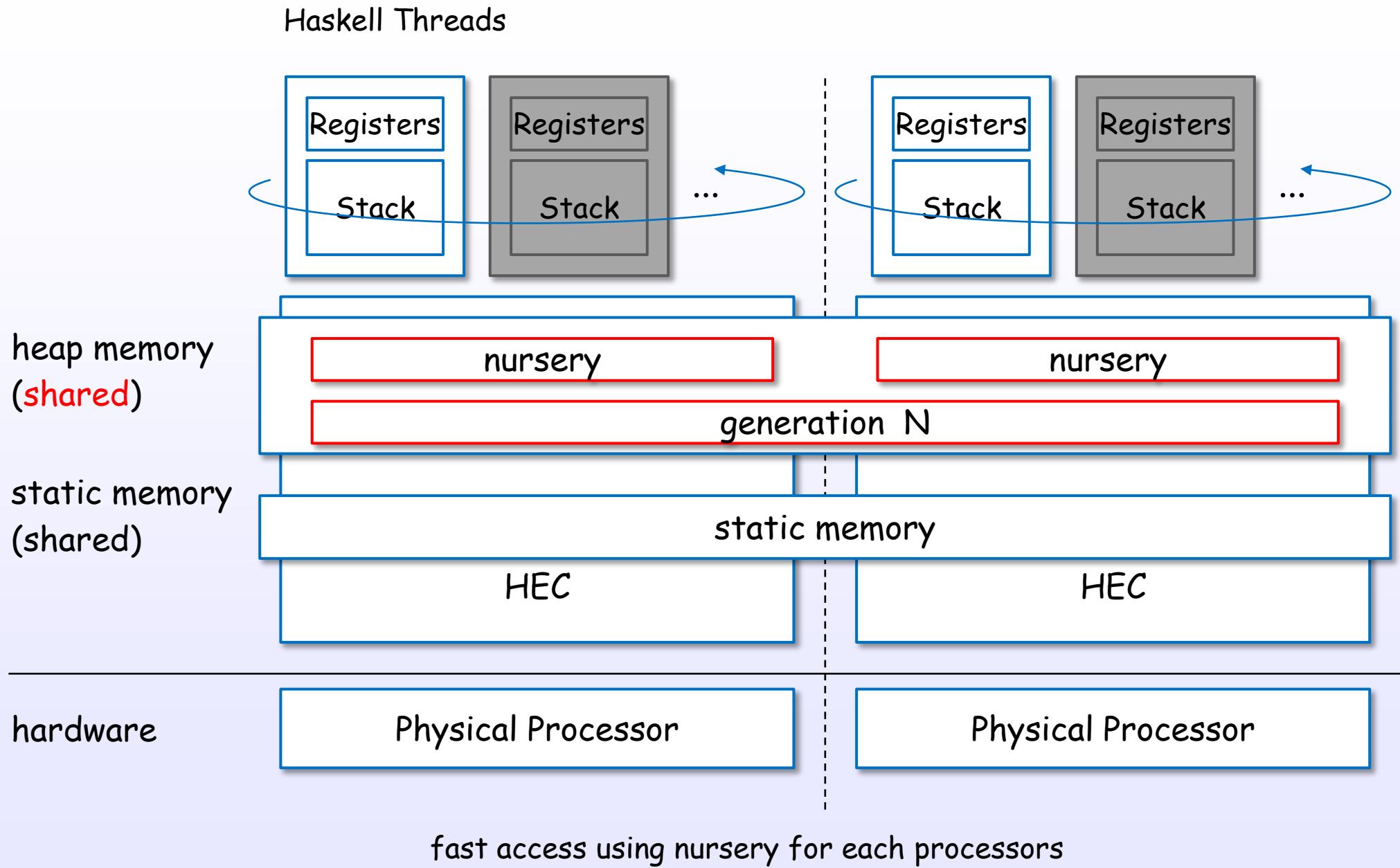
Heap and Threads

Threads shared a heap

Haskell Threads

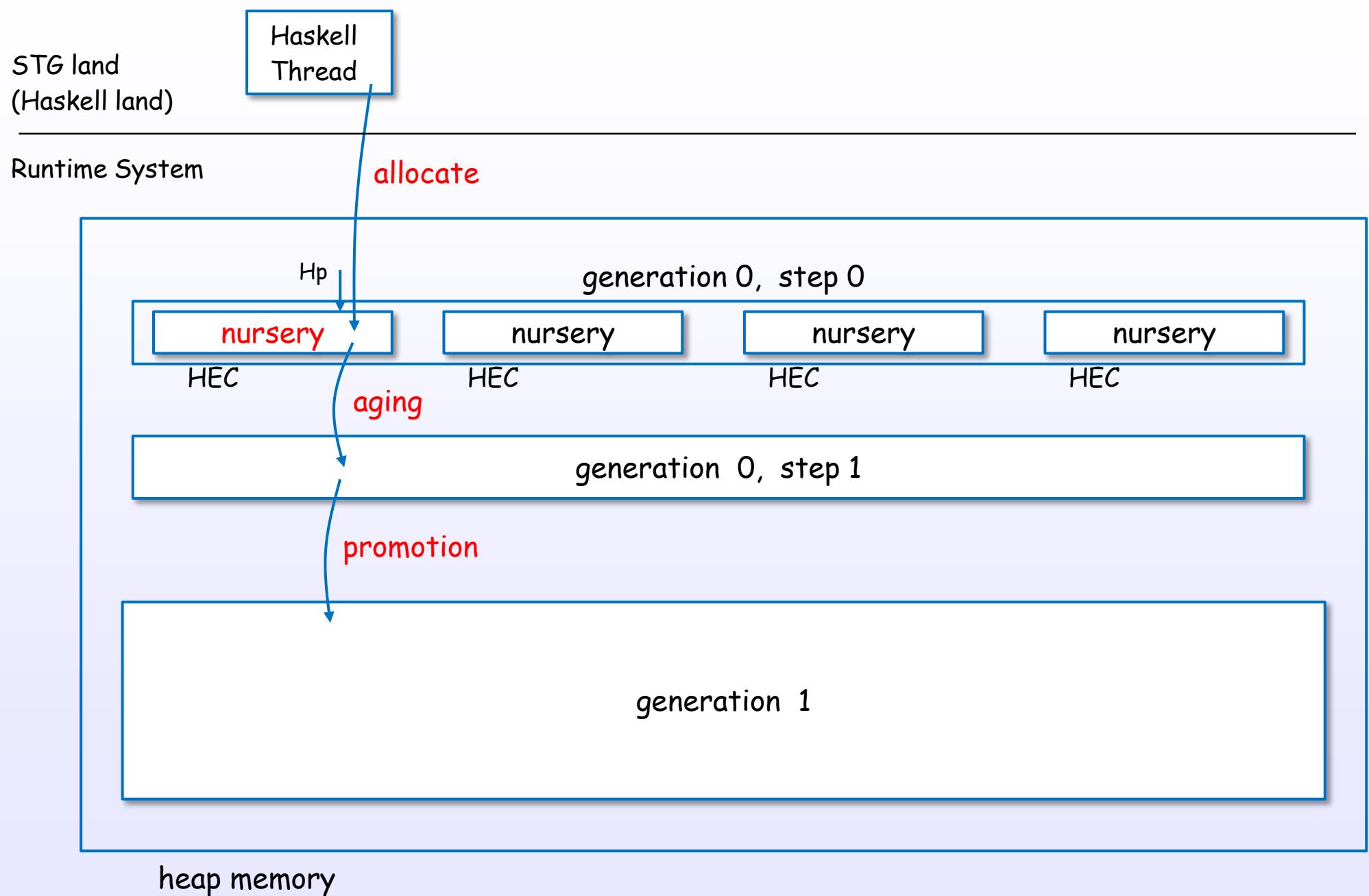


Local allocation area (nursery)



Threads and GC

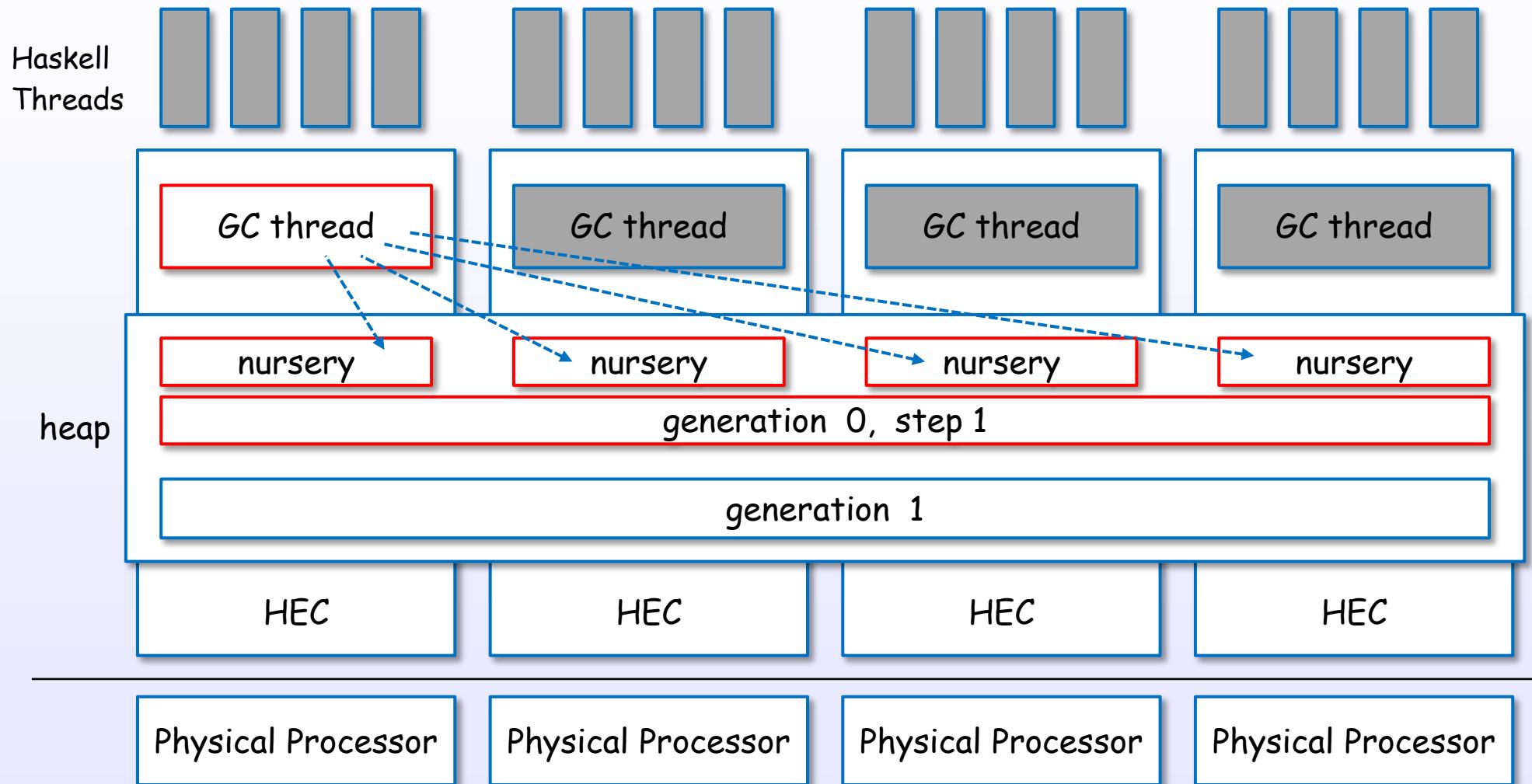
GC, nursery, generation, aging, promotion



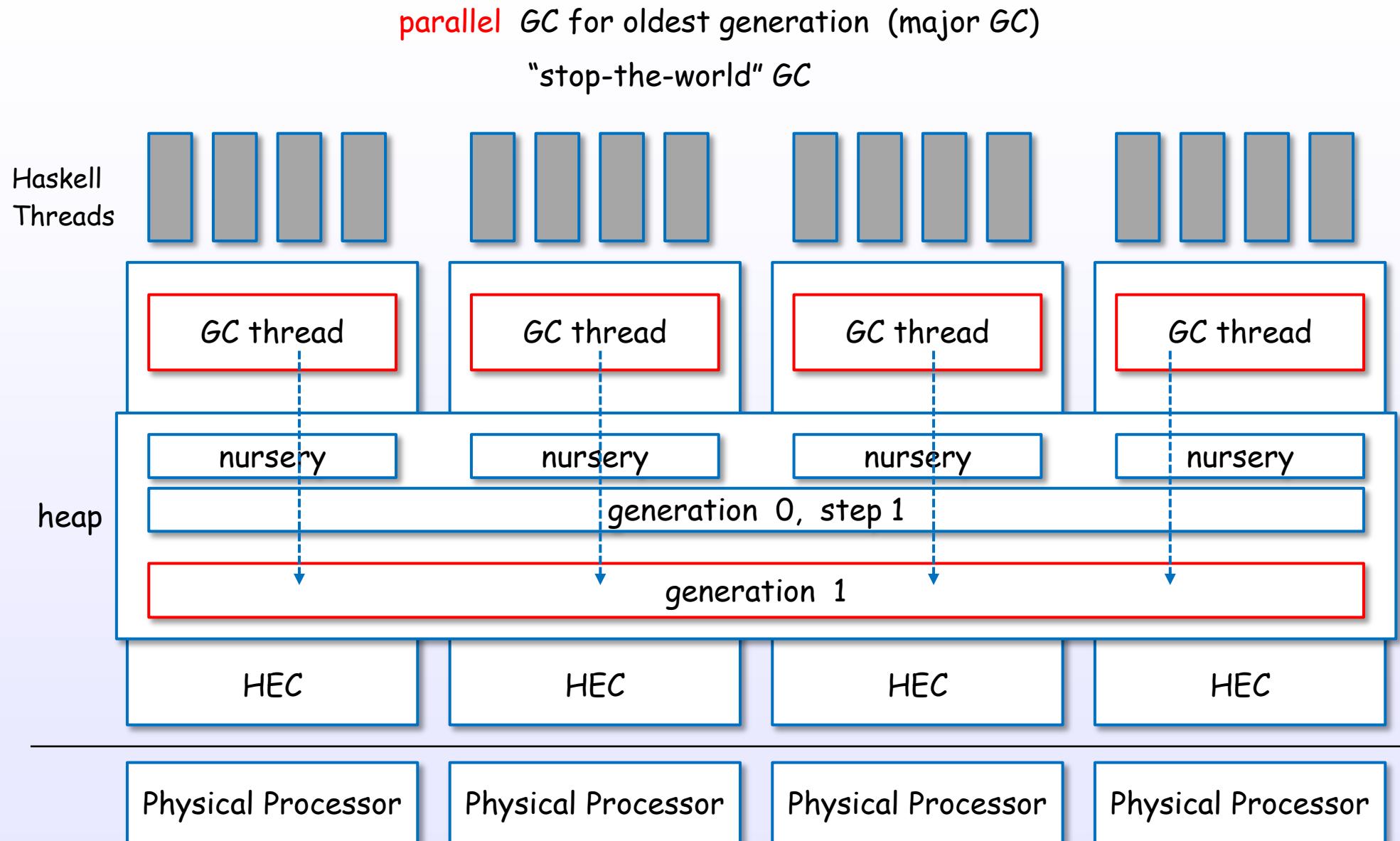
Threads and minor GC

sequential GC for young generation (minor GC)

"stop-the-world" GC

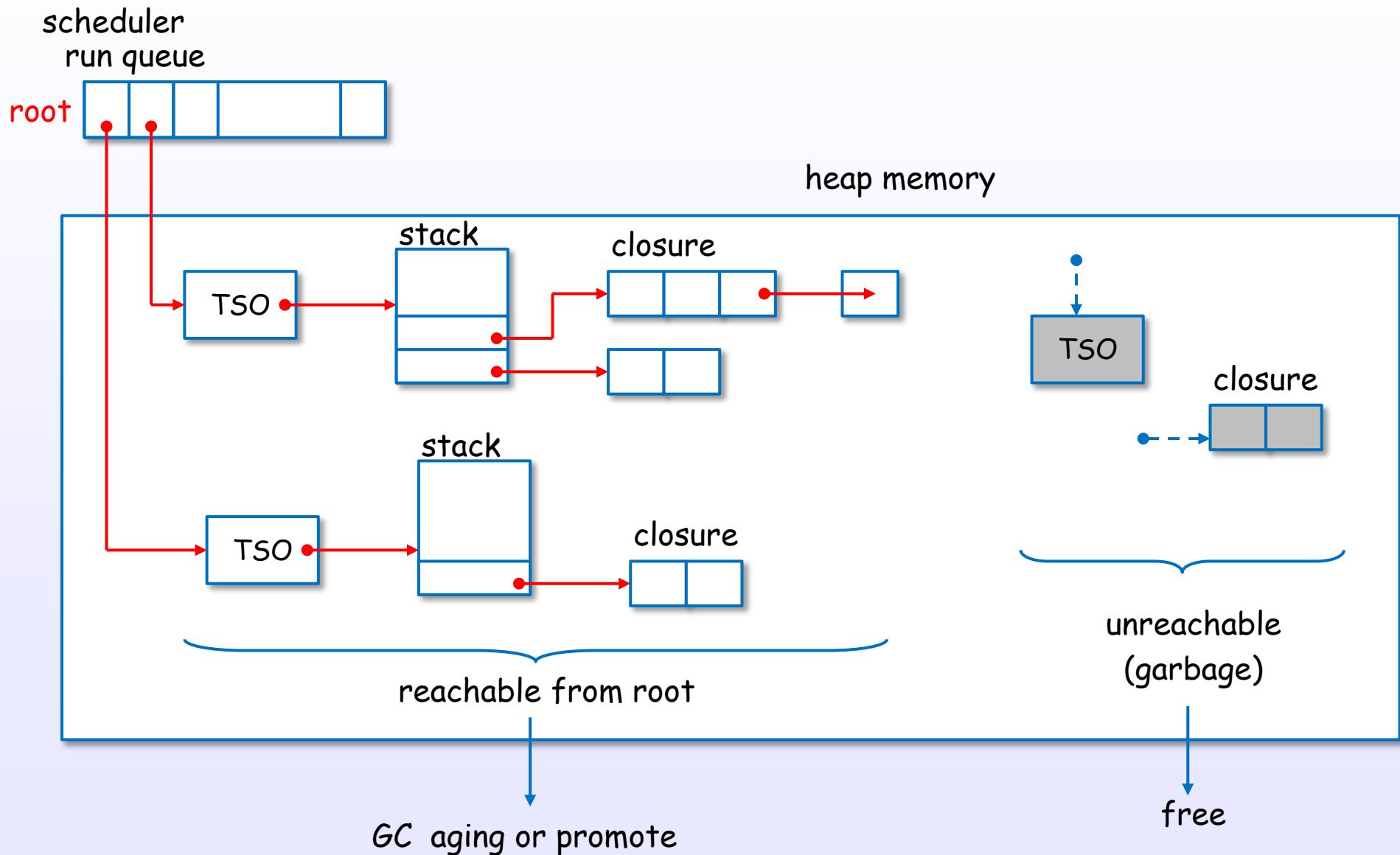


Threads and major GC



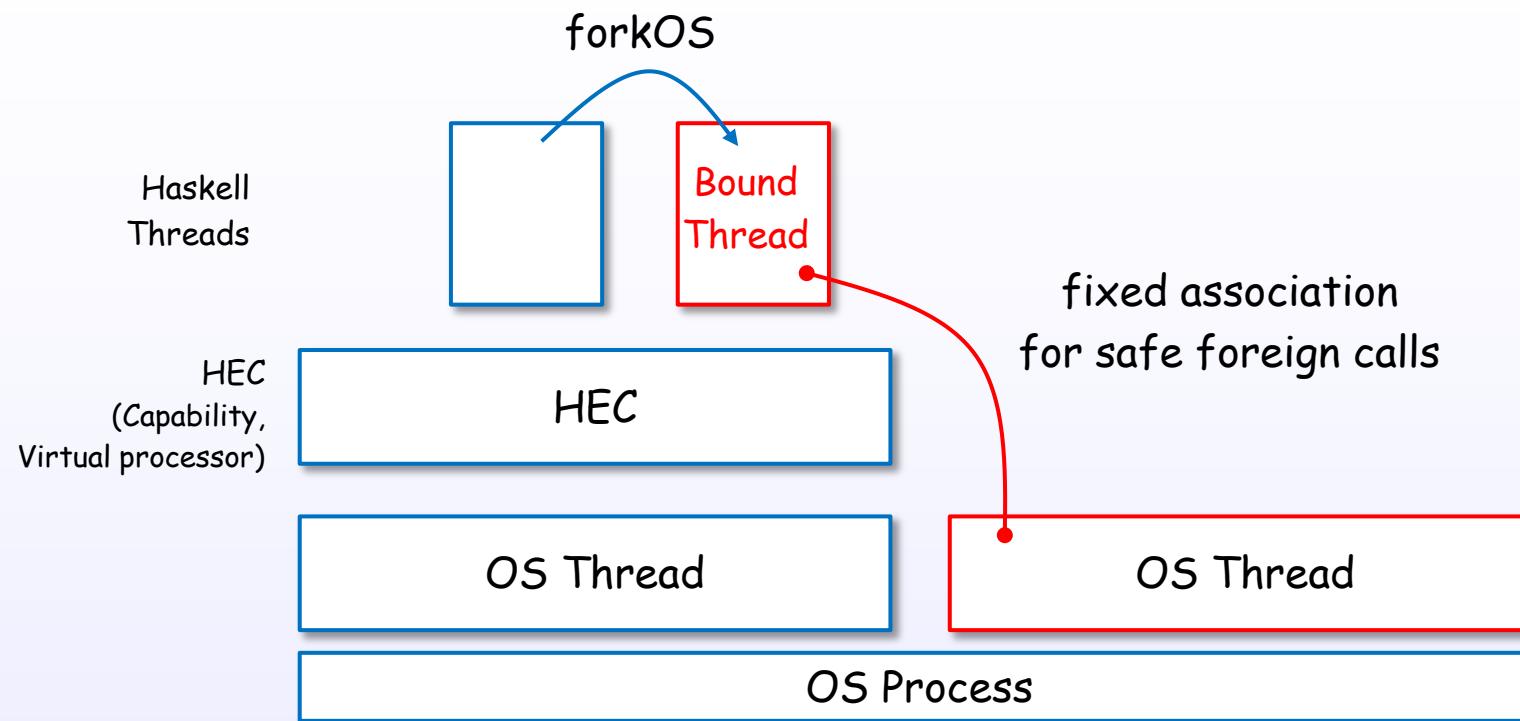
GC discover live objects from the root

Runtime System



Bound thread

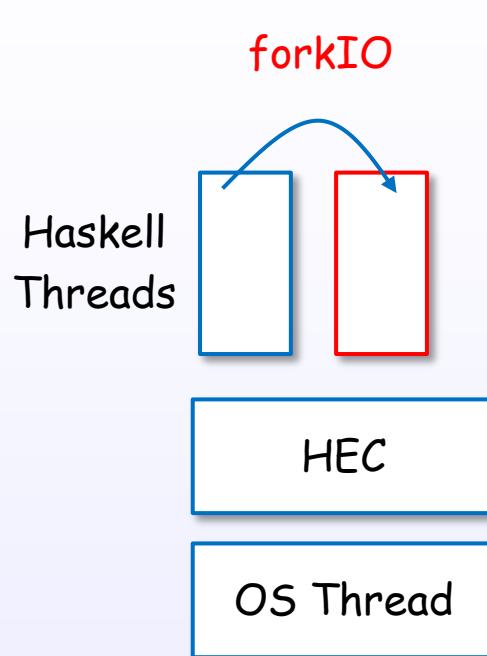
A bound thread has a fixed associated OS Thread



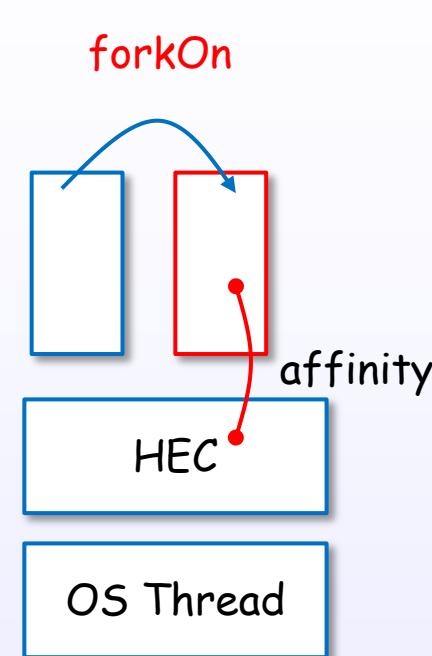
Foreign calls from a bound thread are all made by the same OS thread.
A bound thread is created using forkOS.

The main thread is bound thread.

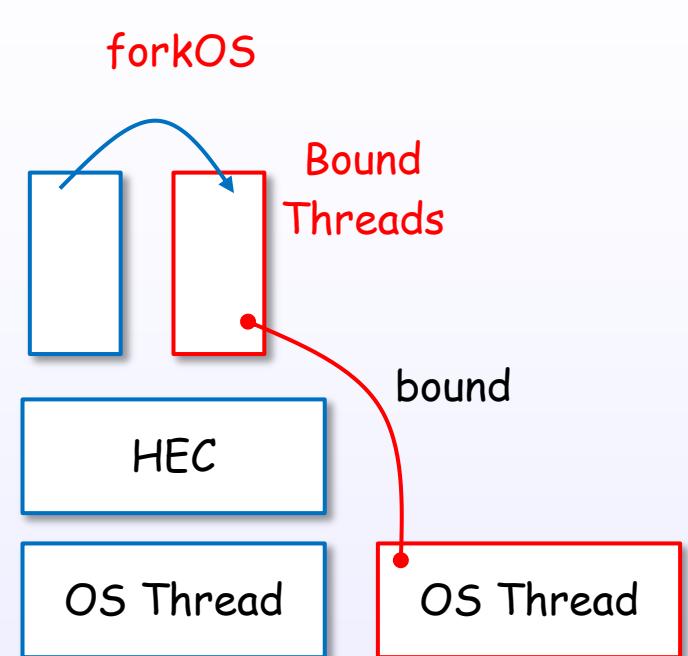
forkIO, forkOn, forkOS



create a haskell unbound
thread



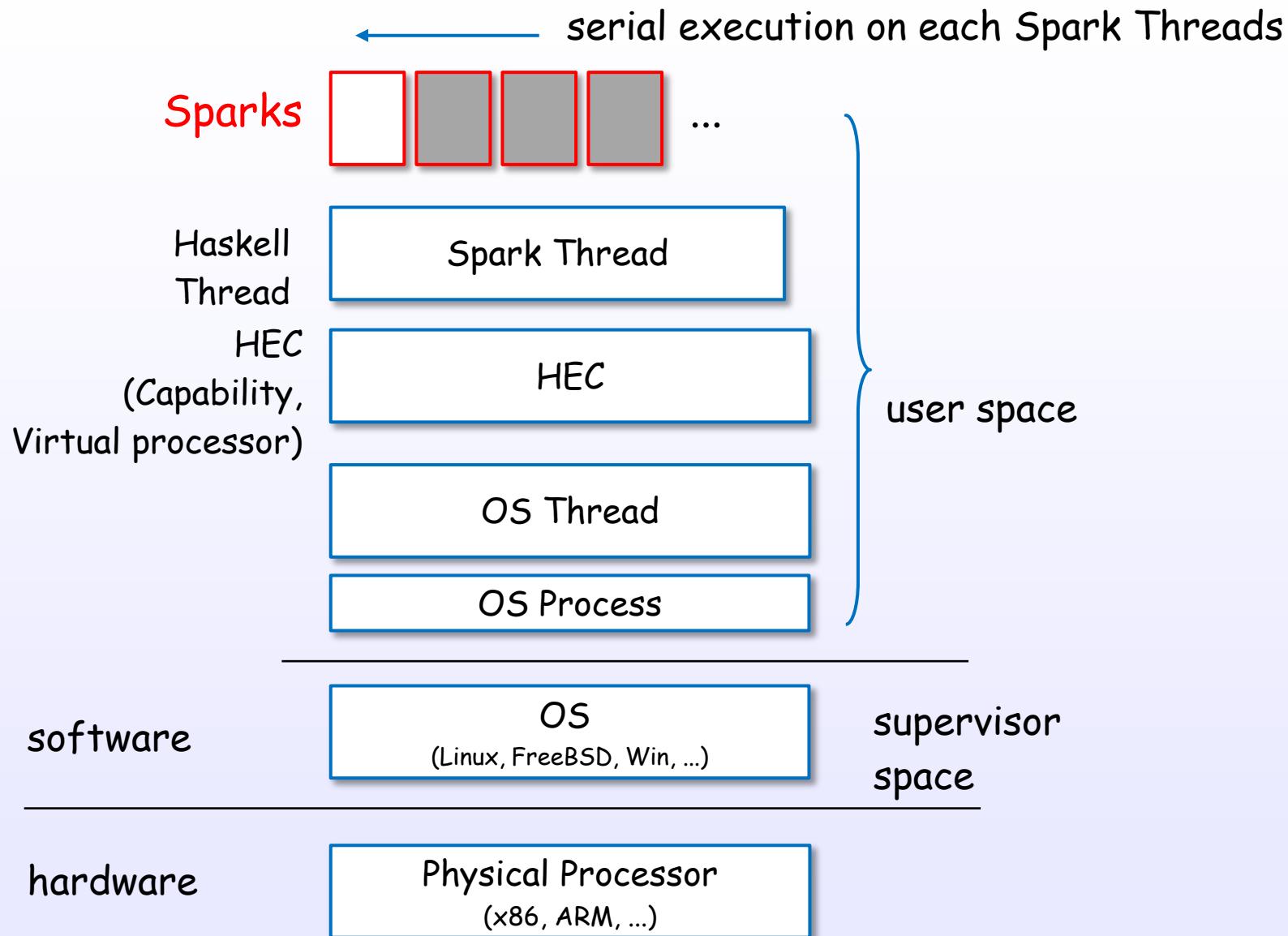
create a haskell unbound
thread
on the specified HEC



create a haskell **bound** thread
and an OS thread

Spark

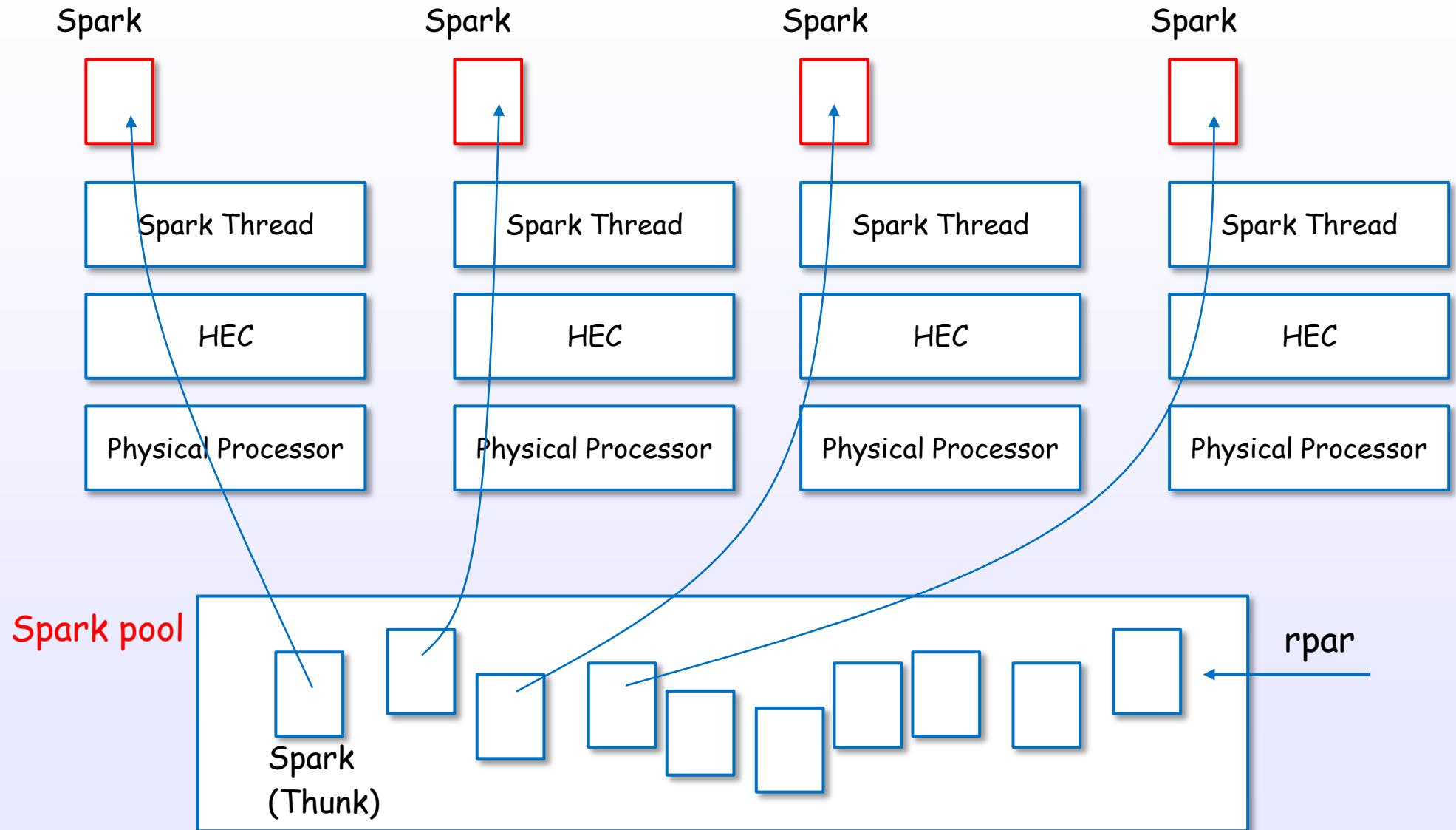
Spark layer



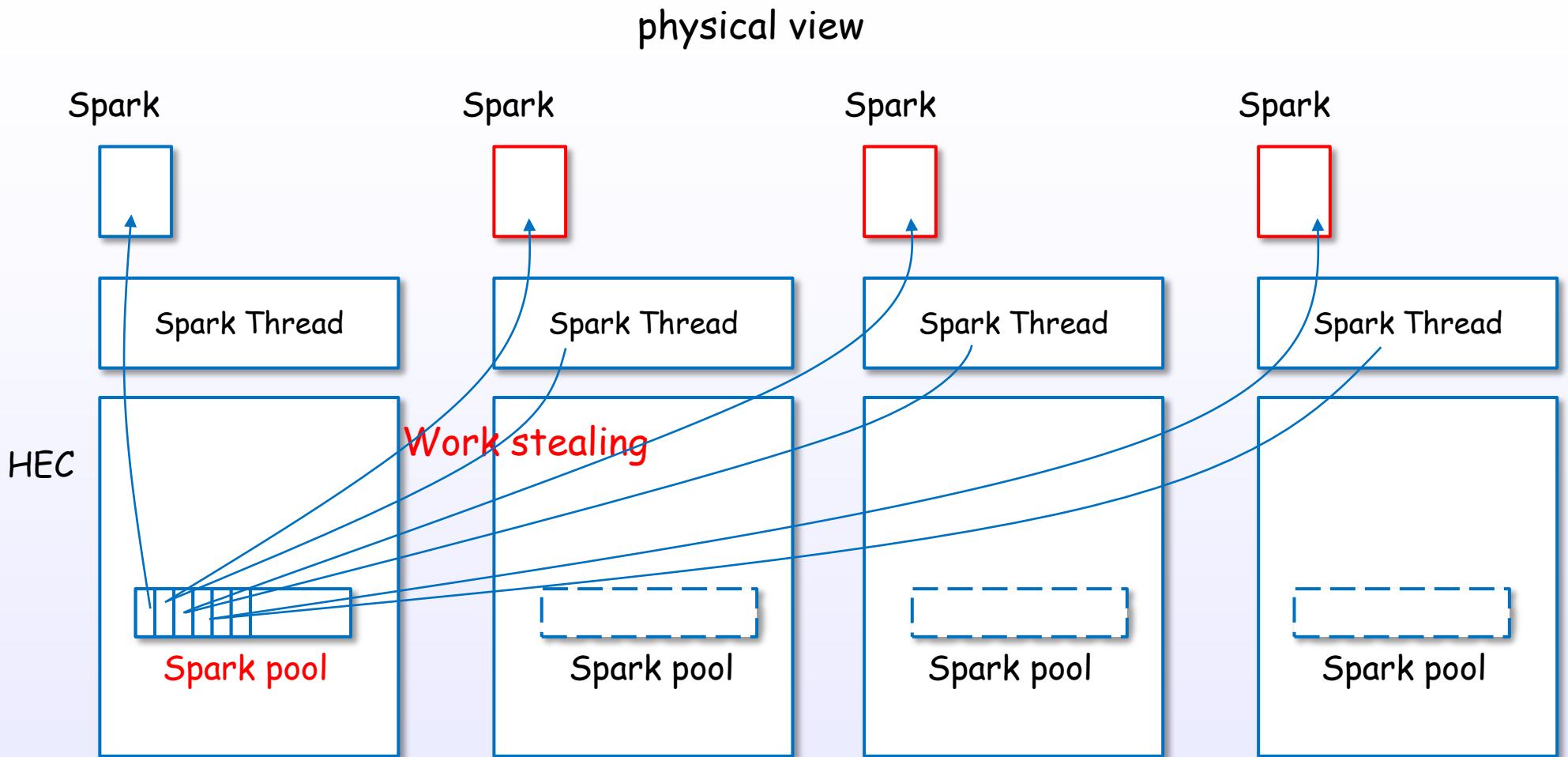
Spark Threads are generated on idle HECs.

Sparks and Spark pool

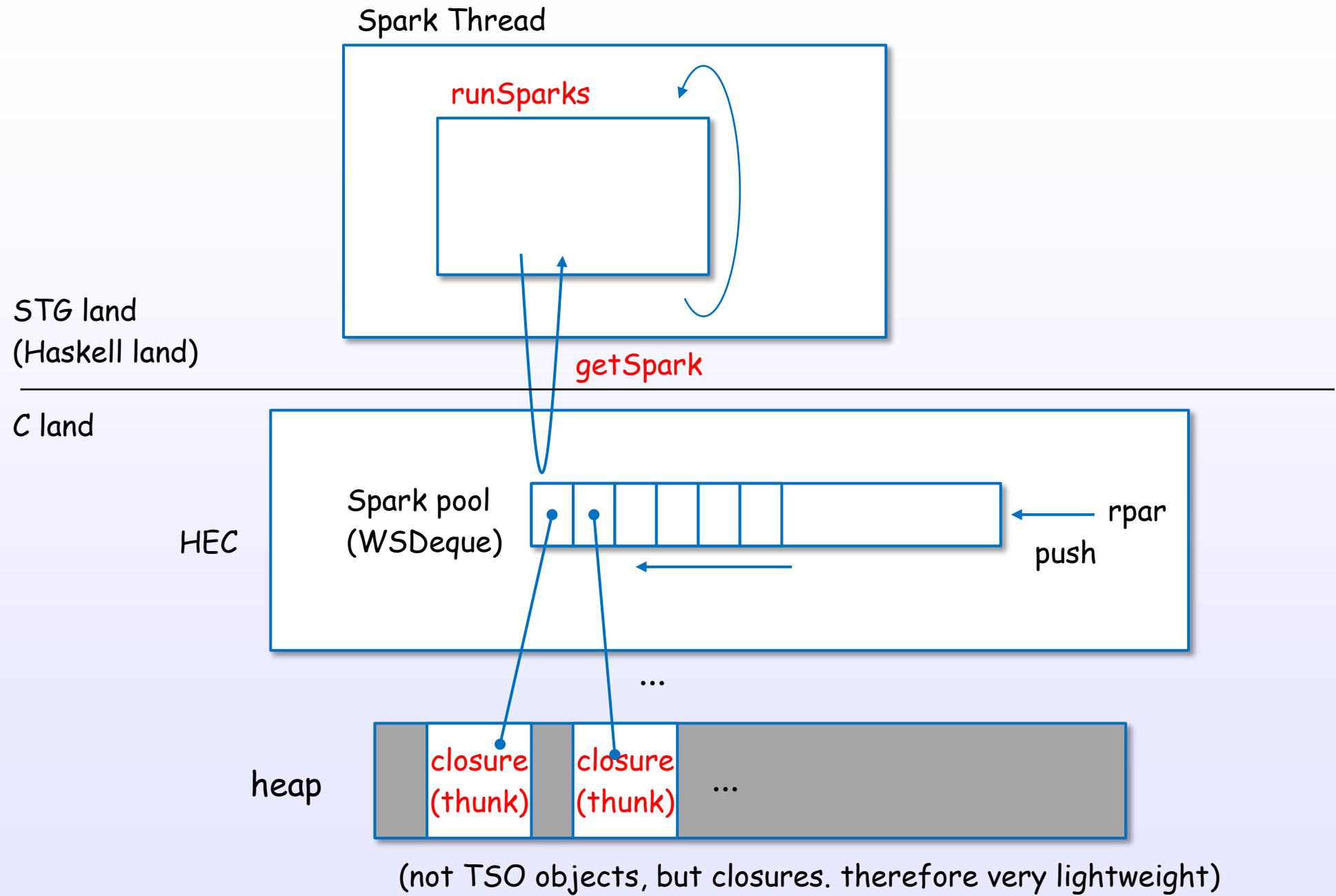
logical view



Spark pool and work stealing



Sparks and closures



MVar

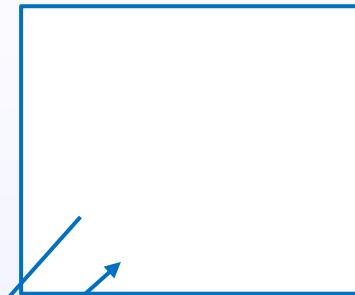
MVar

Haskell Thread #0

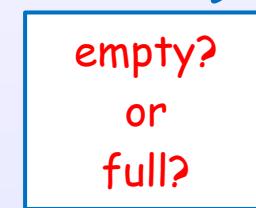


putMVar

Haskell Thread #1



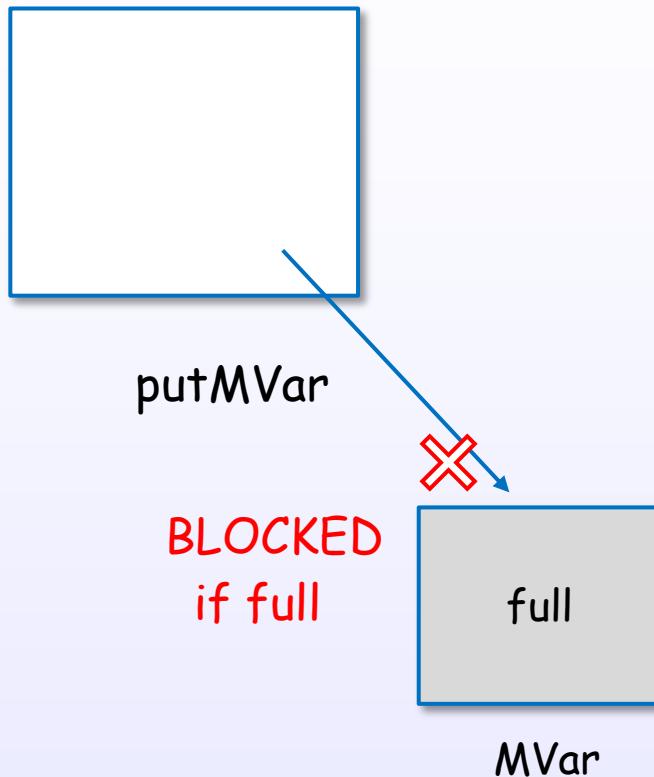
takeMVar



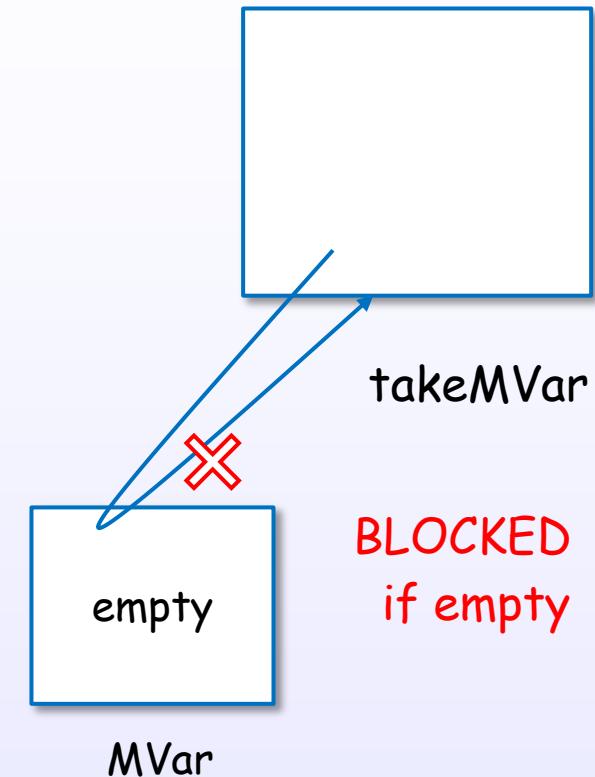
MVar

MVar and blocking

Haskell Thread

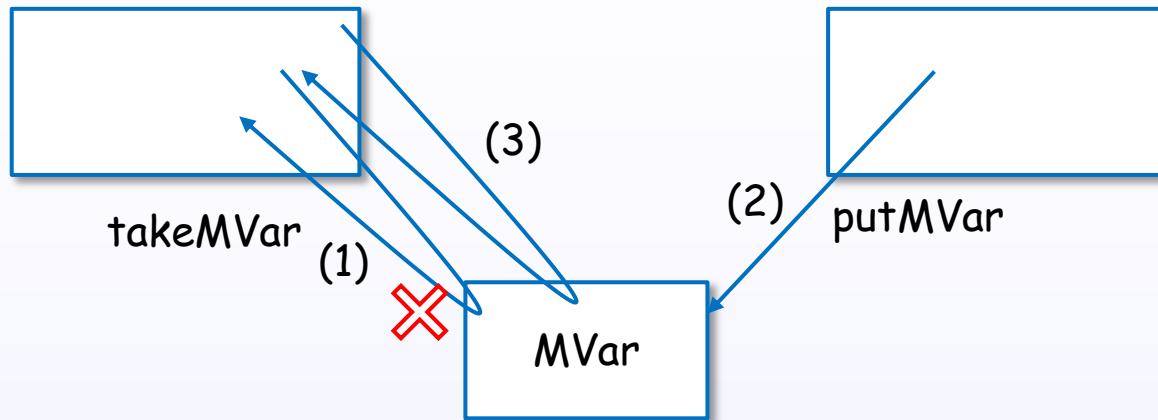


Haskell Thread

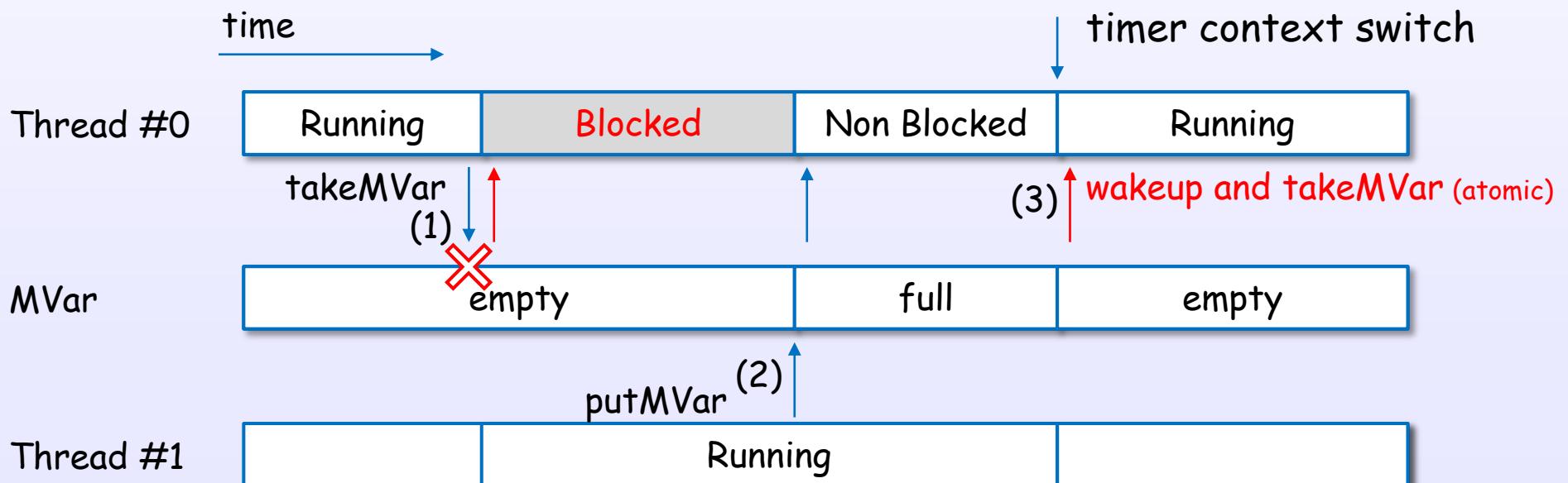
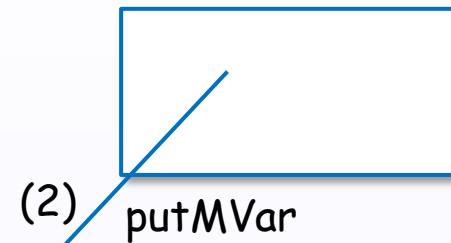


MVar example

Haskell Thread #0



Haskell Thread #1



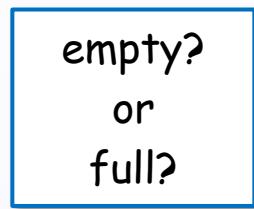
* single core case

References : [16], [18], [19], [S31], [S12]

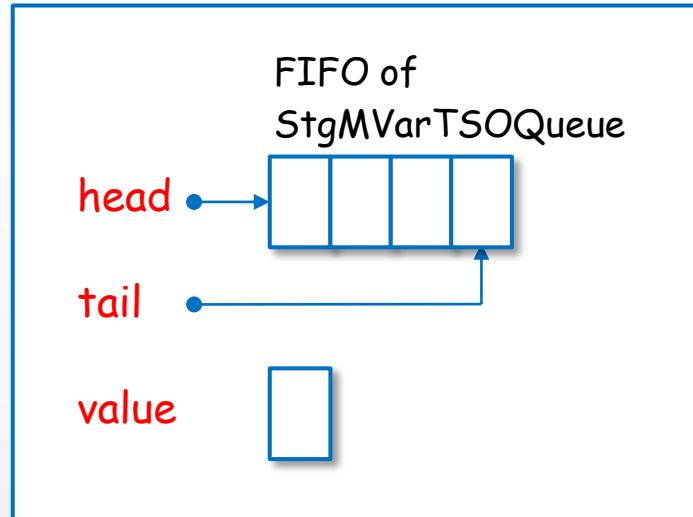
MVar object view

User view

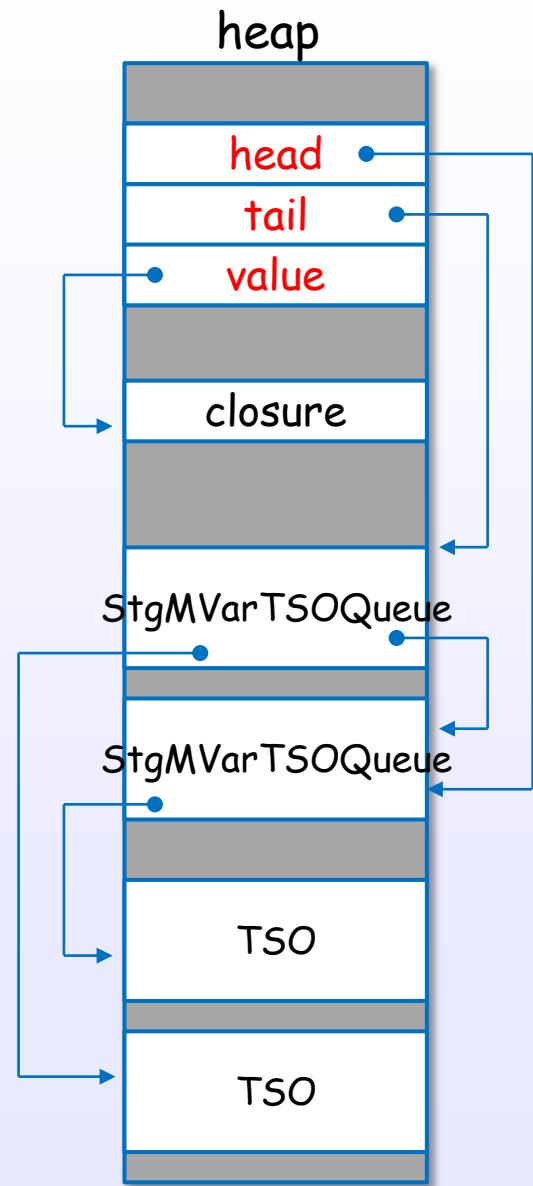
MVar



logical MVar object



physical MVar object



newEmptyMVar

Haskell Threads

newEmptyMVar
newMVar#

(1) call the Runtime primitive

Runtime System

stg_newMVarzh
ALLOC_PRIM_
SET_HDR
StgMVar_head
StgMVar_tail
StgMVar_value

(2) create a MVar object in the heap

MVar object

stg_END_TSO_QUEUE_closure

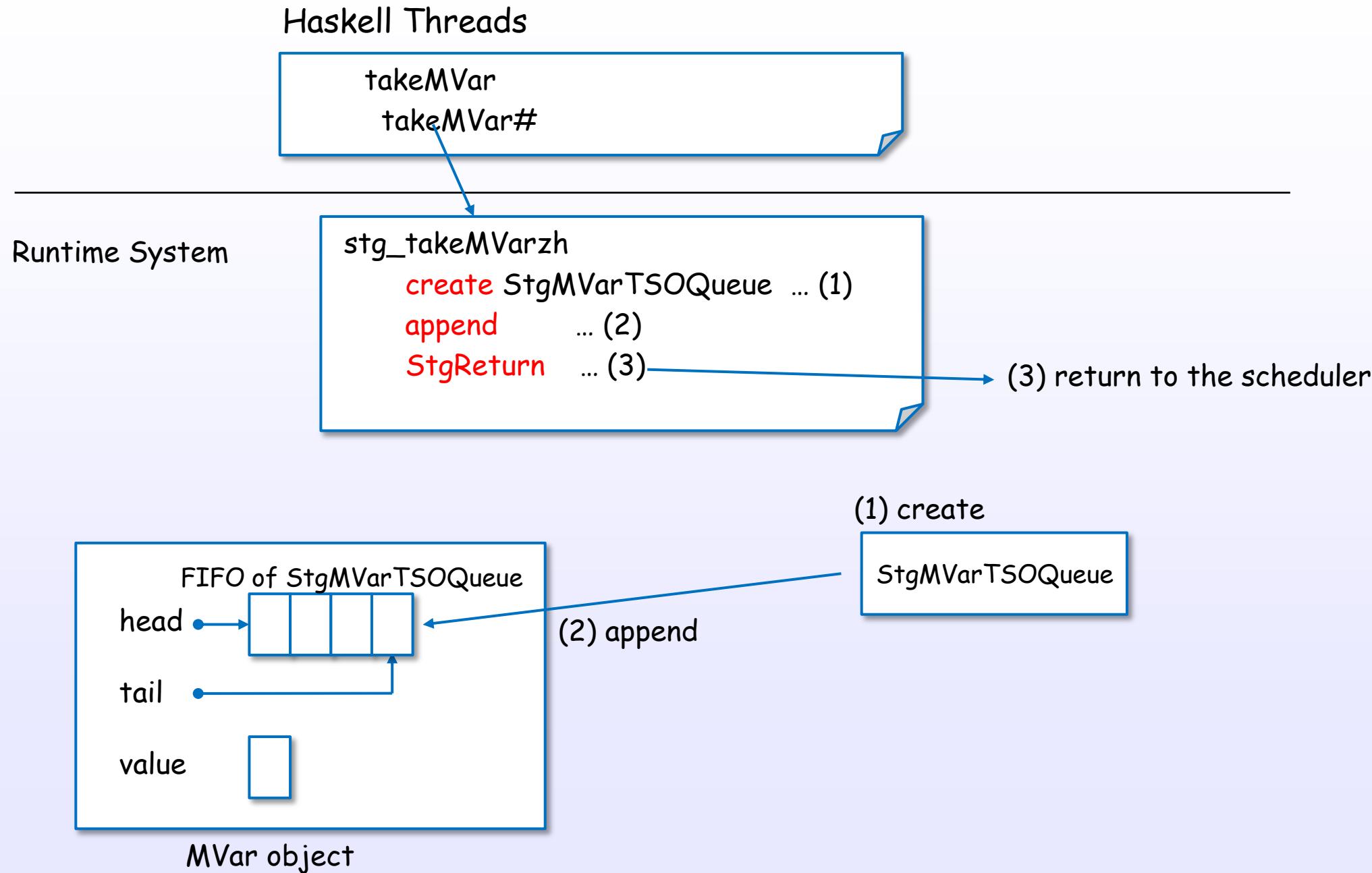
heap

head
tail
value

(3) link each fields

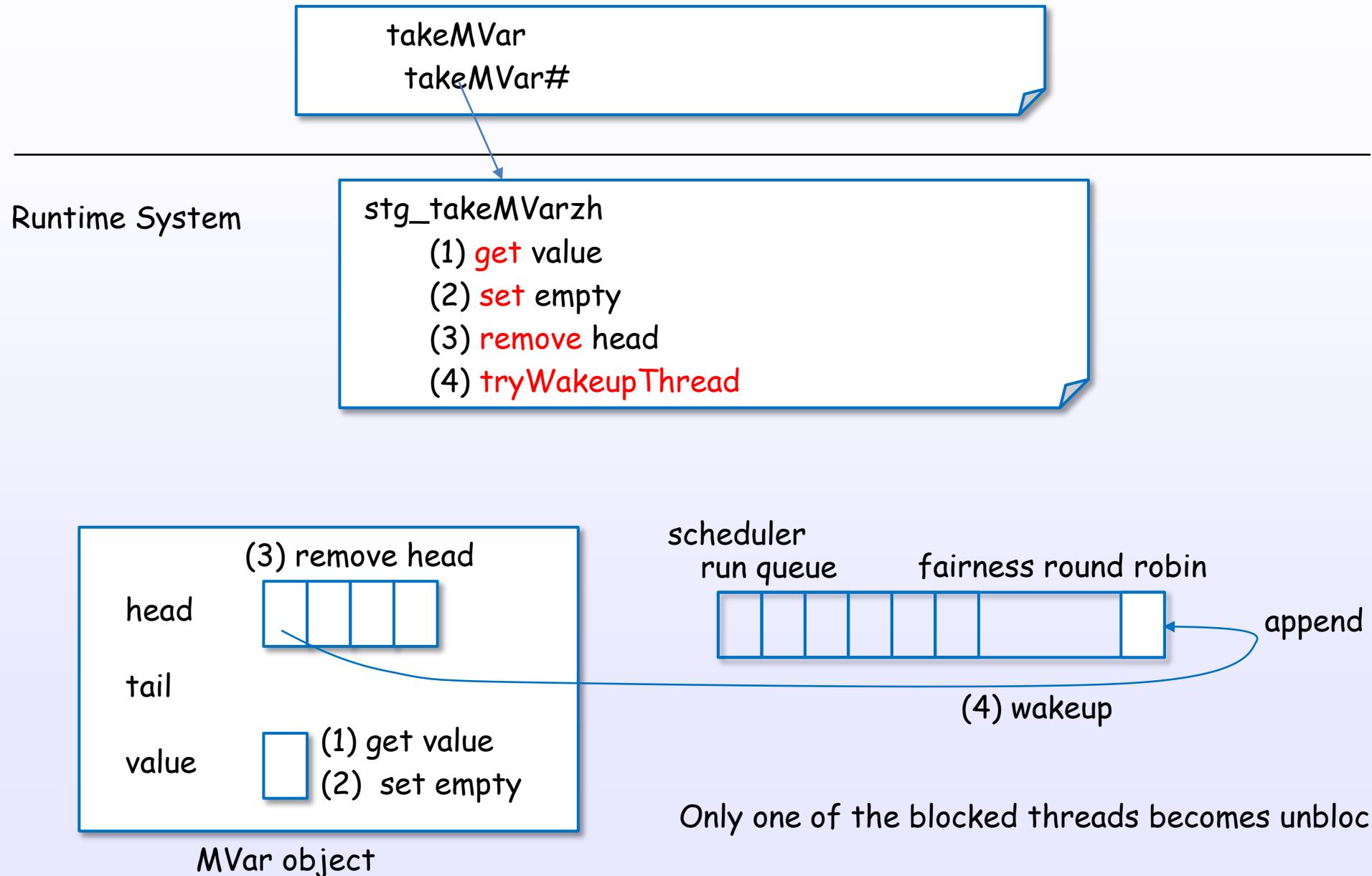
References : [16], [18], [19], [S31], [S12]

takeMVar (empty case)



takeMVar (full case)

Haskell Threads



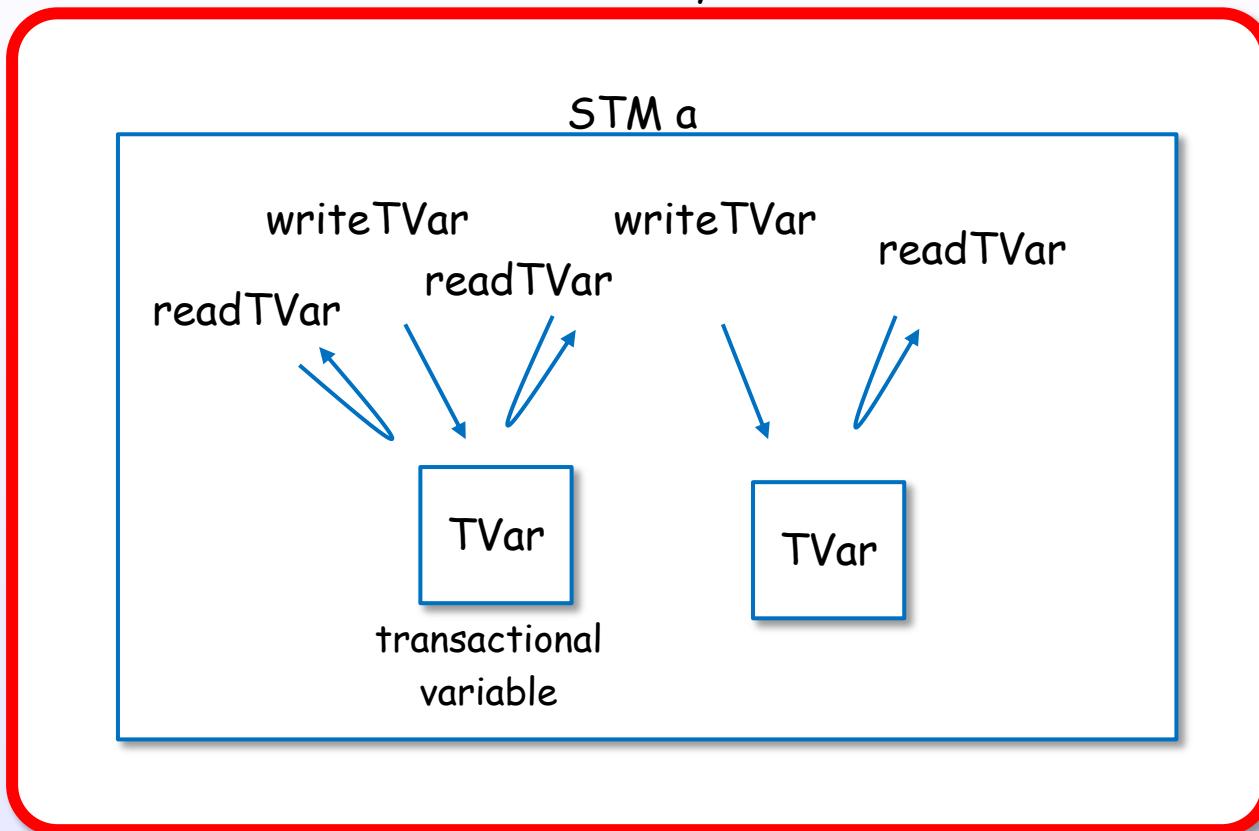
Only one of the blocked threads becomes unblocked.

Software transactional memory

Create a atomic block using atomically

atomically :: STM a -> IO a

atomically

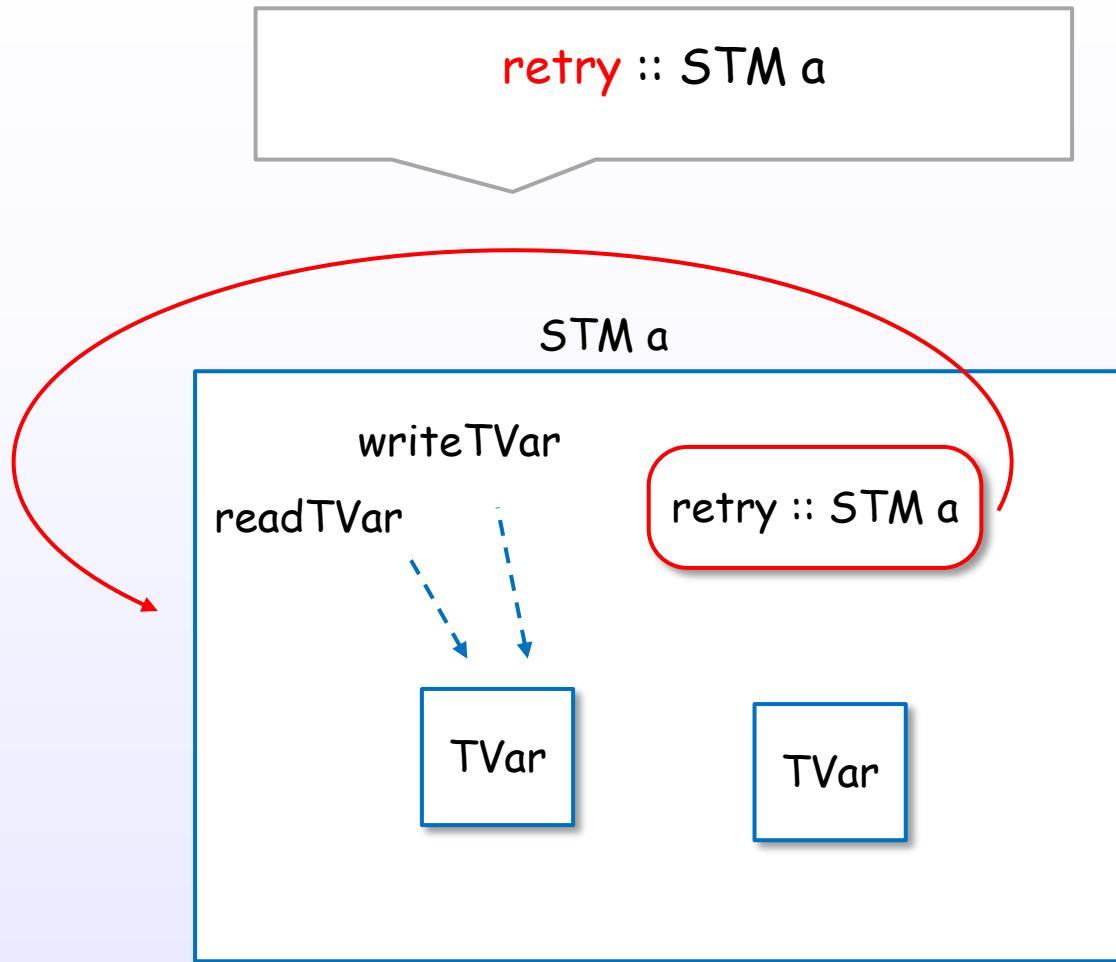


Create and evaluate a **composable “atomic block”**

Atomic block = All or Nothing

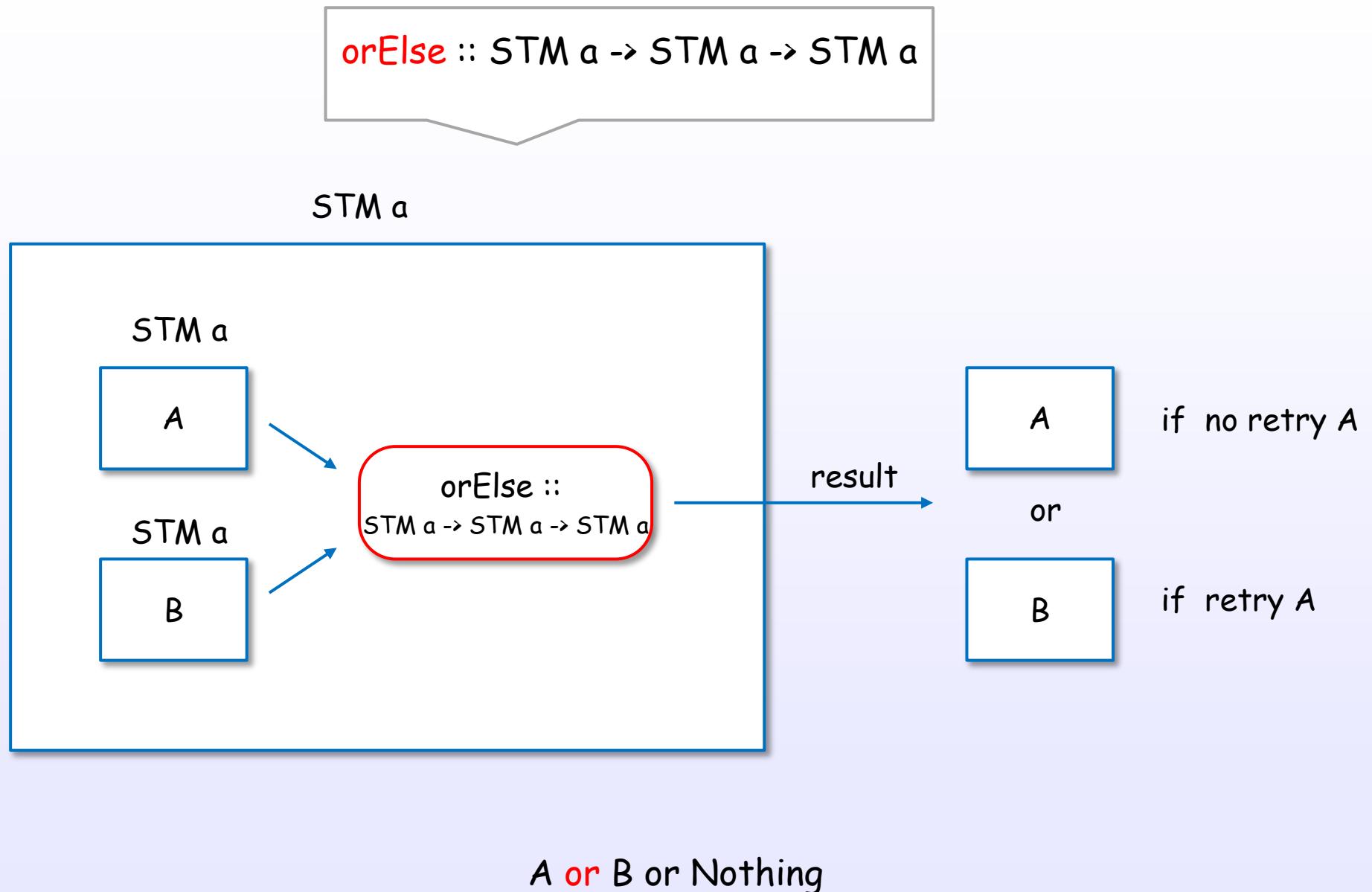
References : [17], [19], [20], [C18], [S12], [S28]

Rollback and blocking control using retry

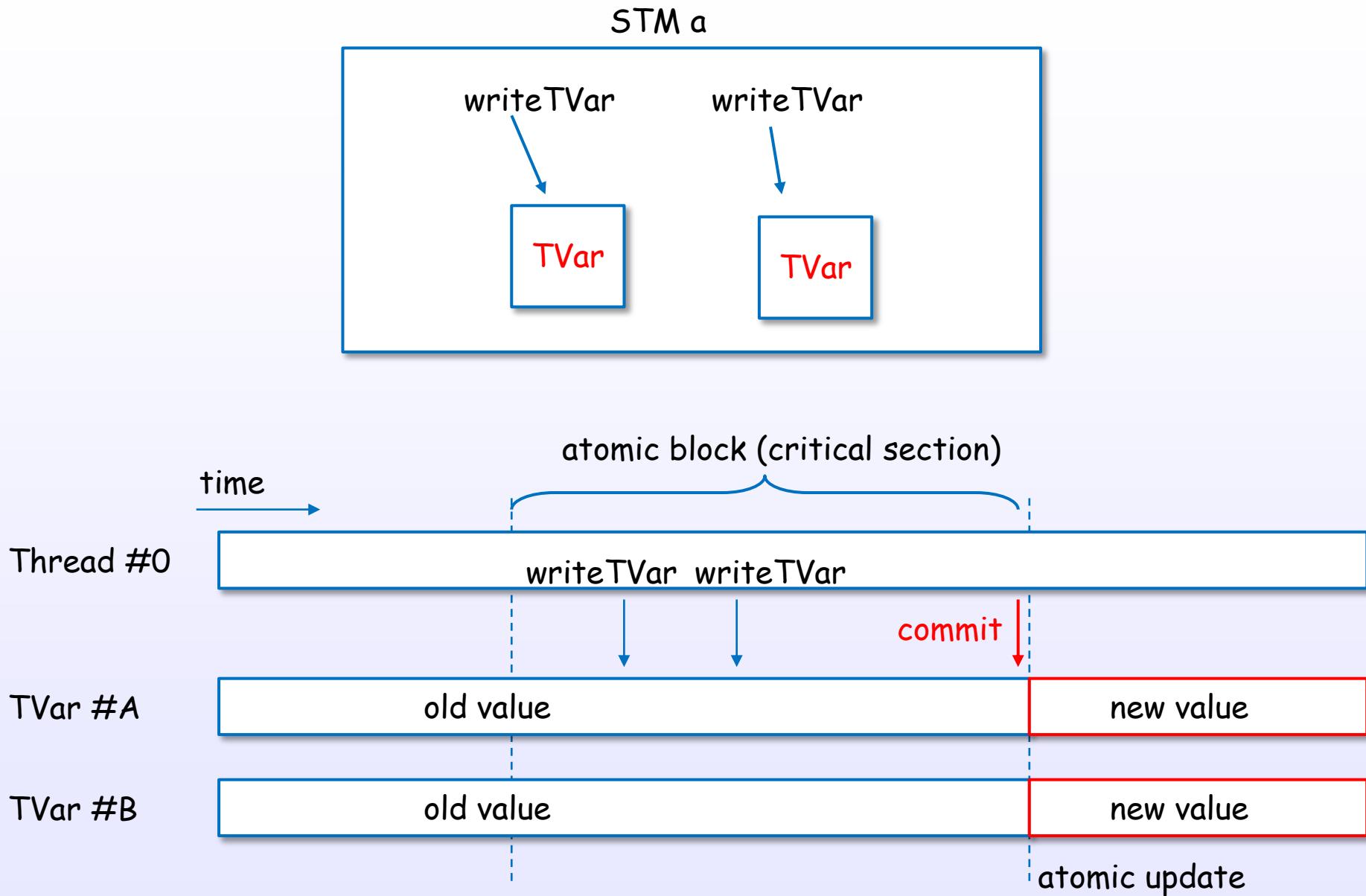


Discard, blocking and try again

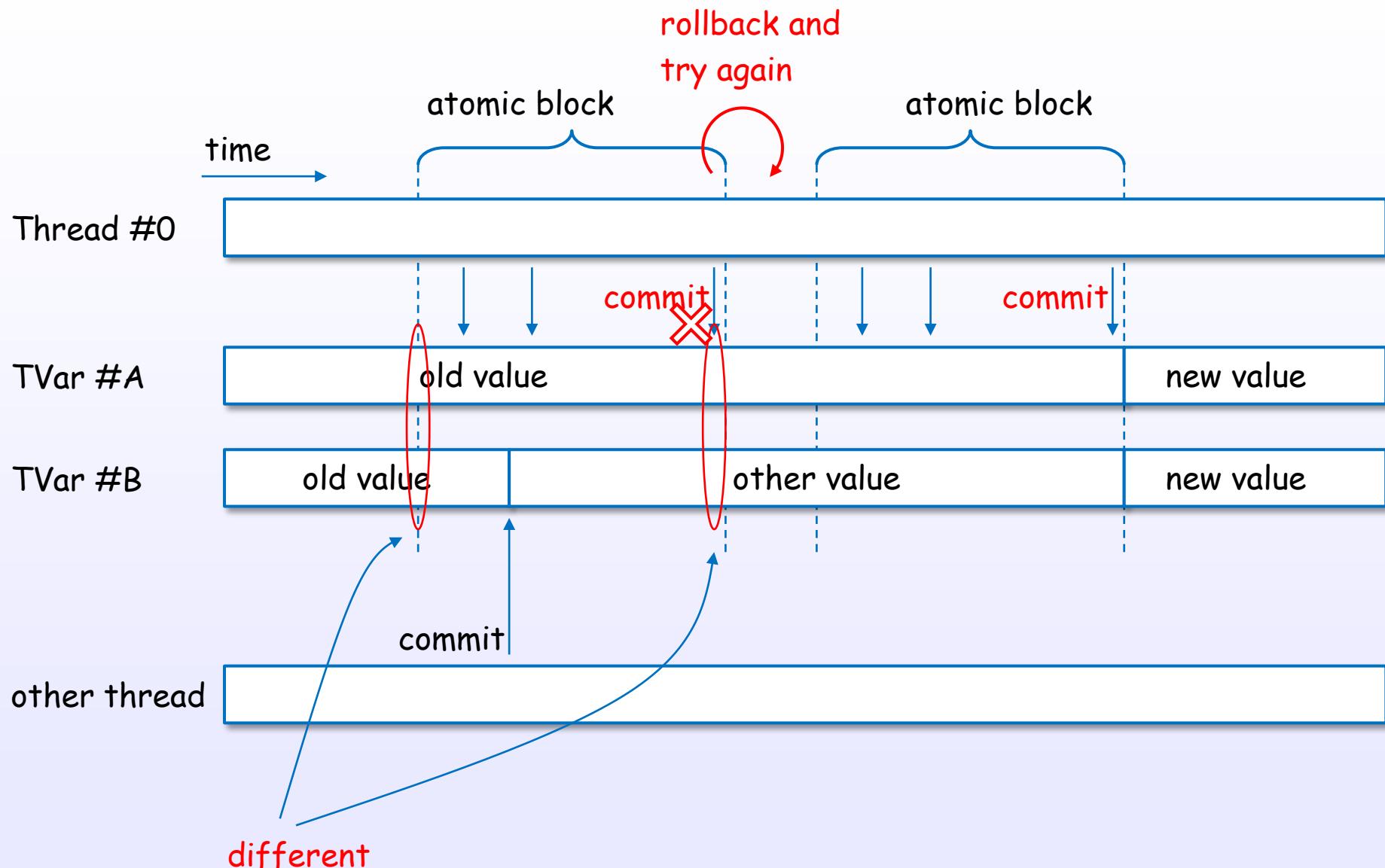
Compose OR case using orElse



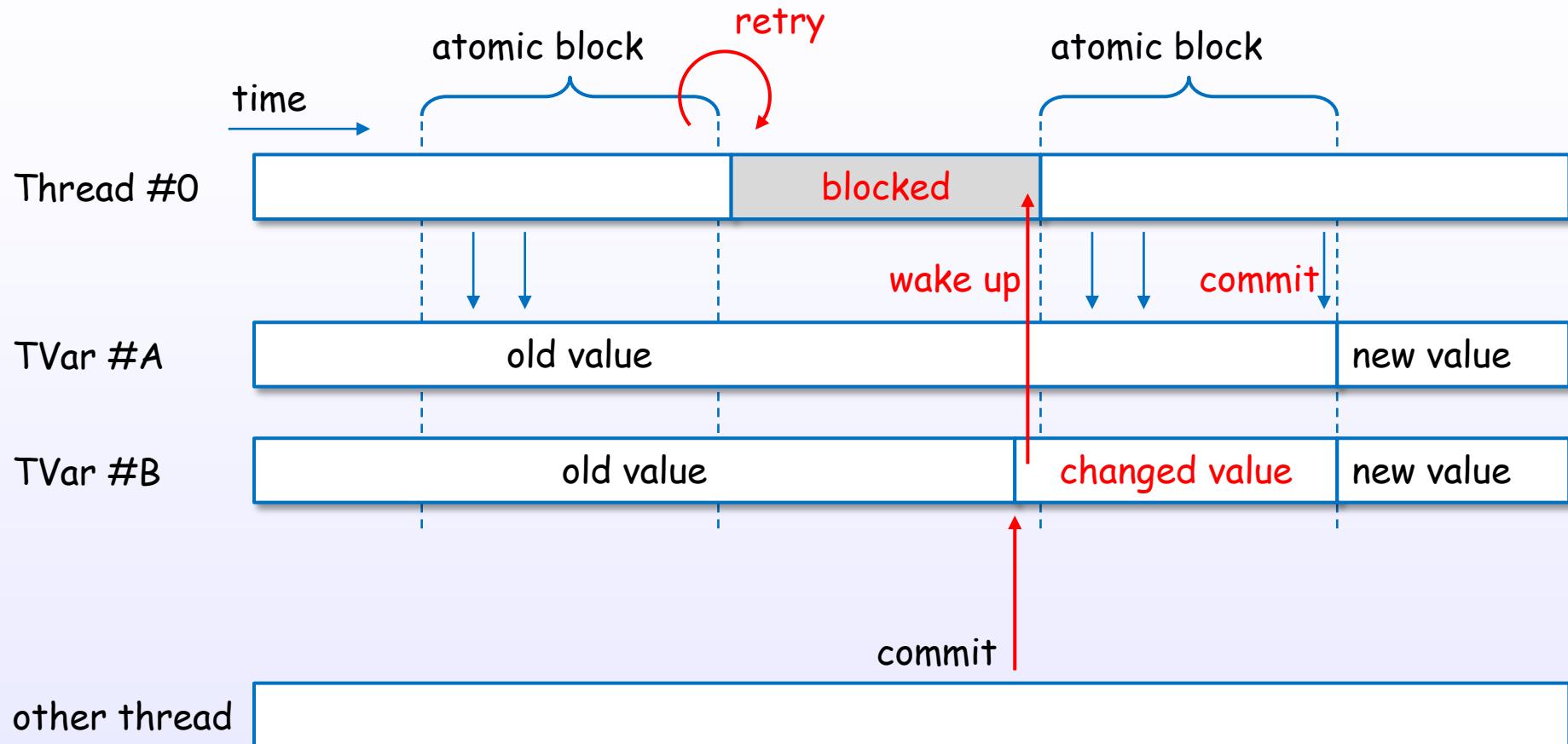
STM, TVar example (normal case)



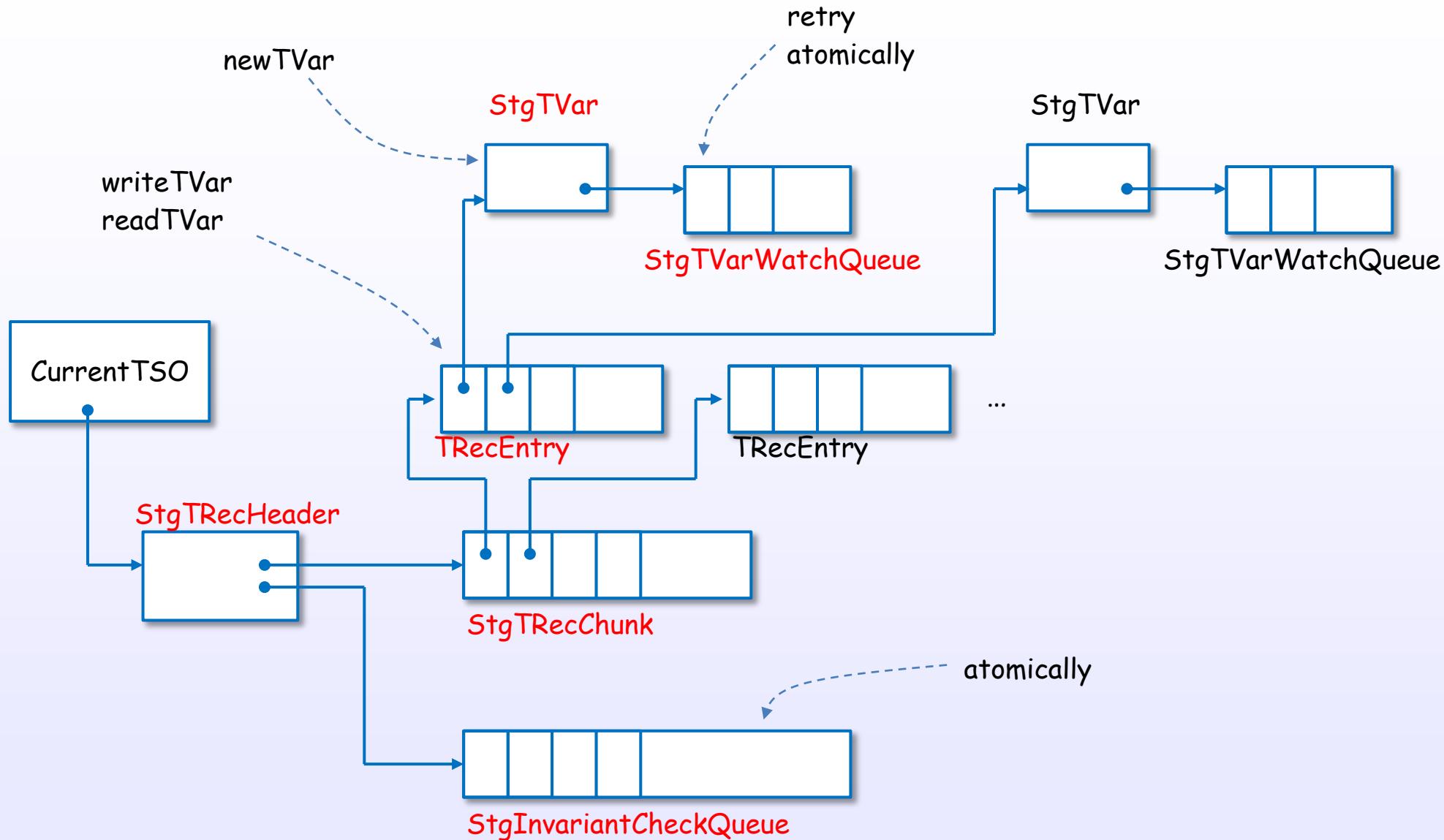
STM, TVar example (conflict case)



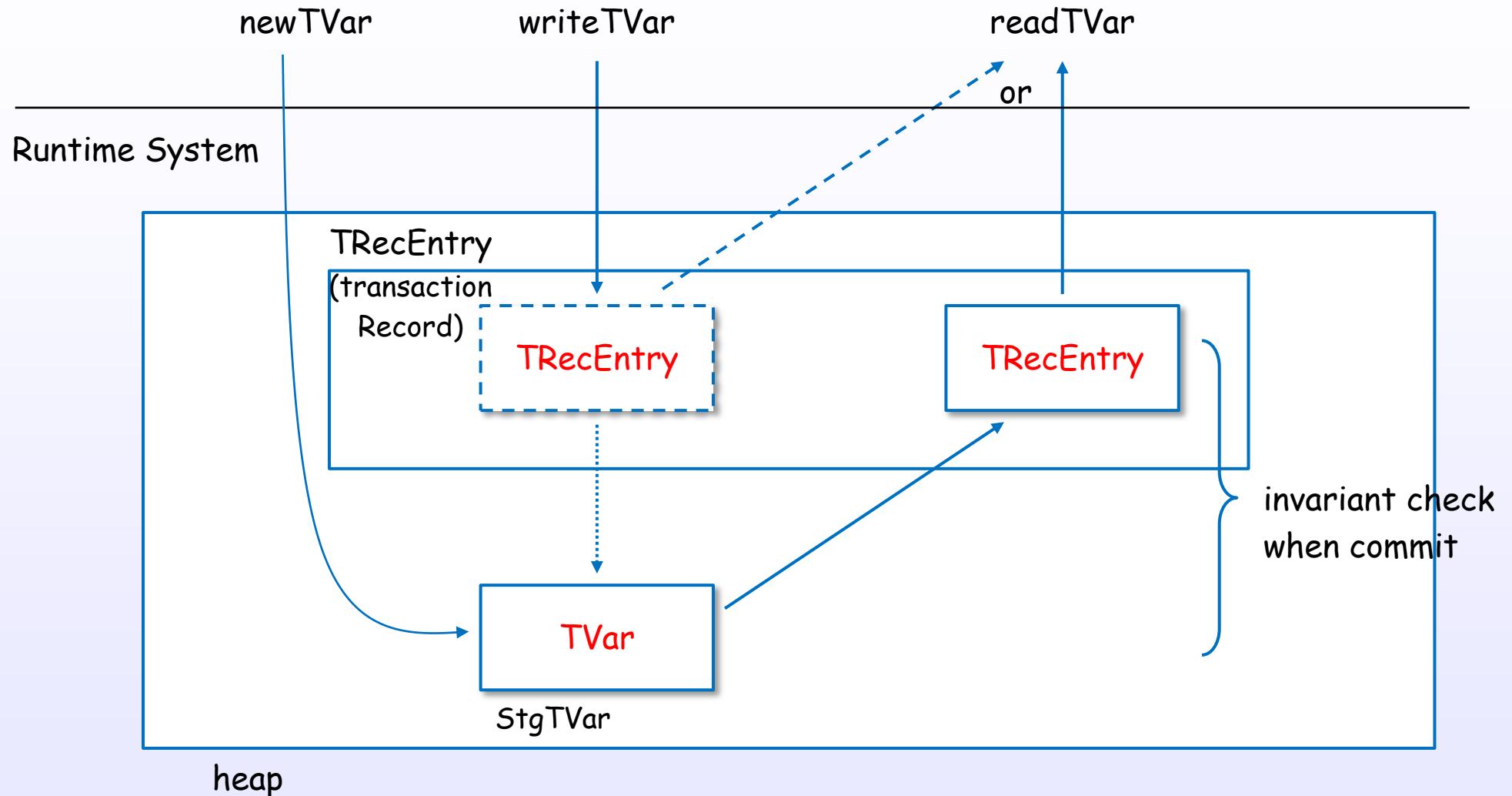
retry example



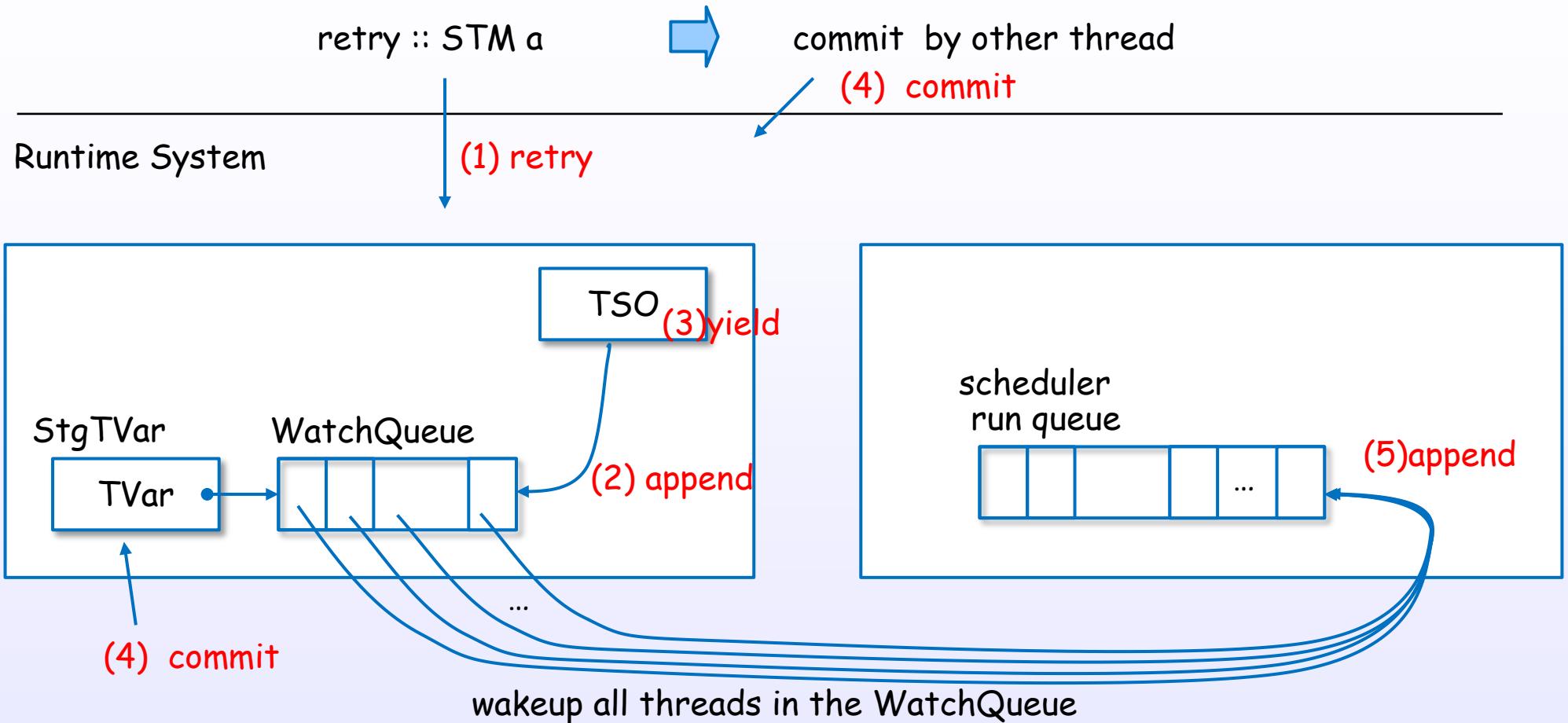
STM, TVar data structure



newTVar, writeTVar, readTVar



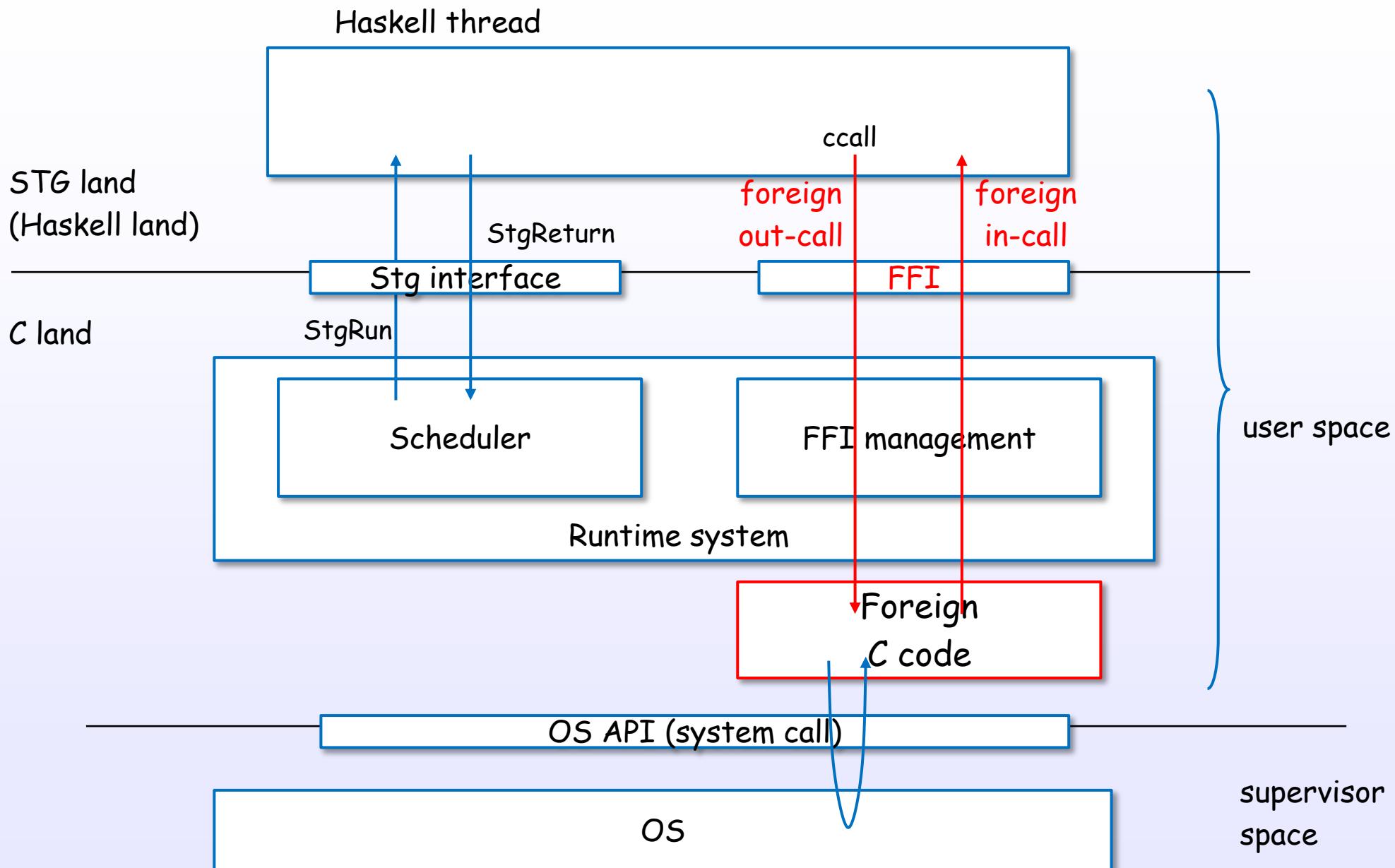
block by retry, wake up by commit



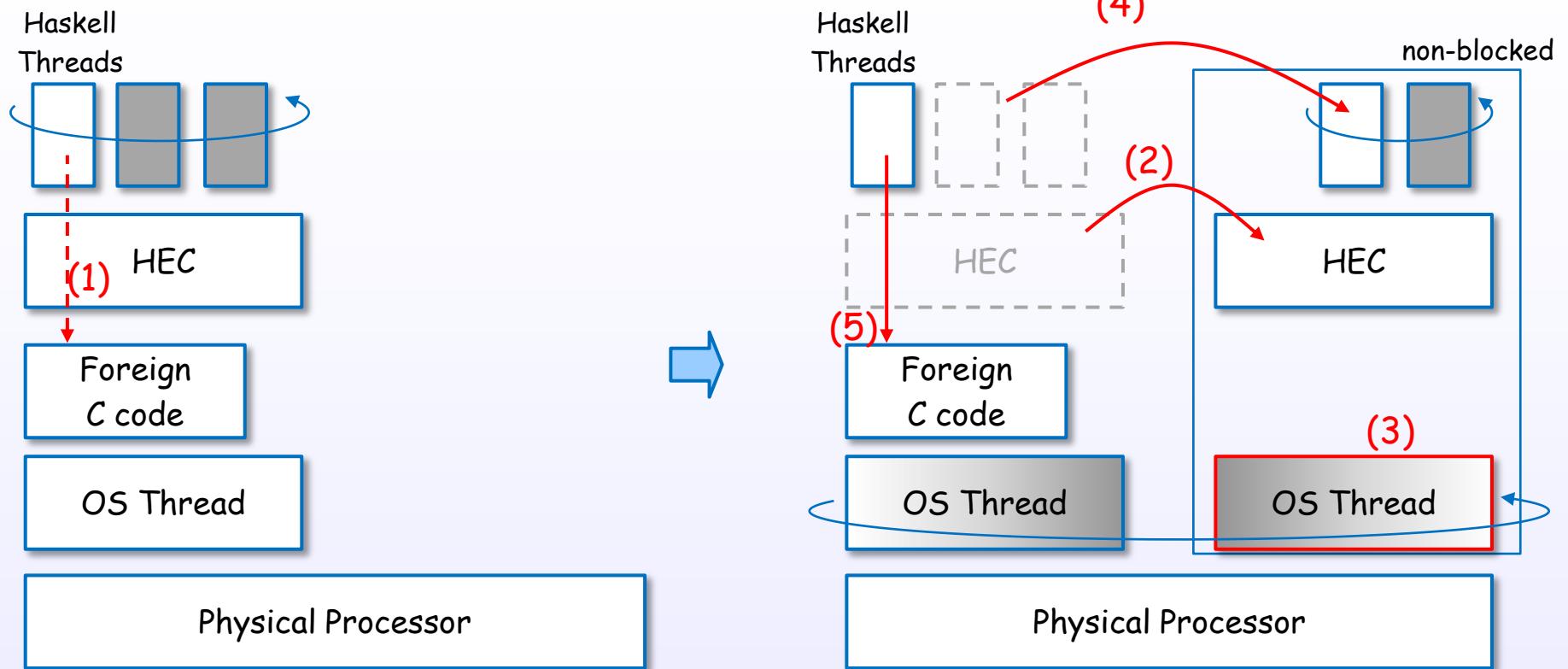
no guarantee of fairness,
because the RTS has to run all the blocked transaction.

FFI

FFI (Foreign Function Interface)



FFI and OS Threads



(1) a safe foreign call (FFI)

(2) move the HEC to other OS thread

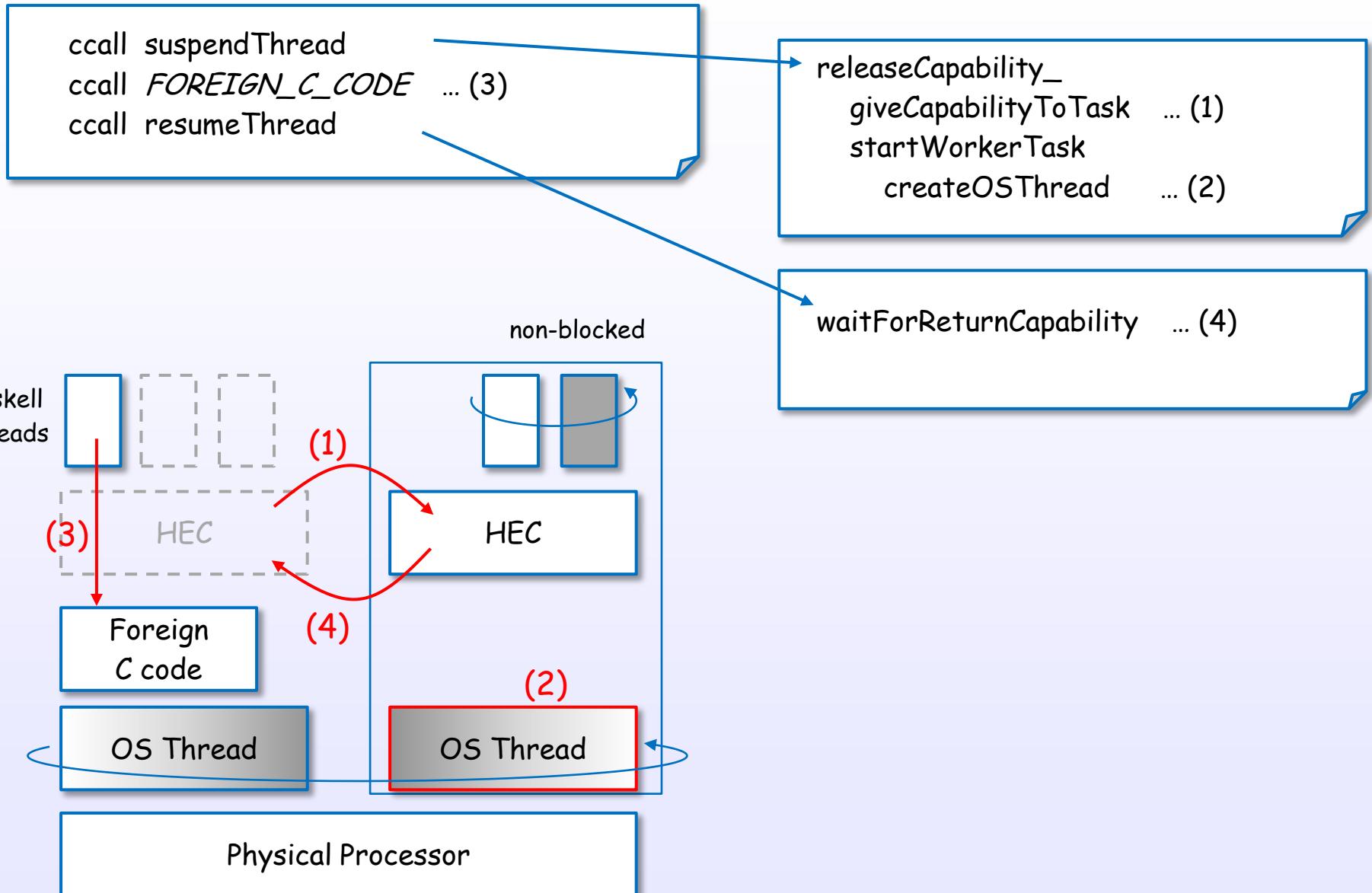
(3) spawn or draw an OS thread

(4) move Haskell threads

(5) call the foreign C code

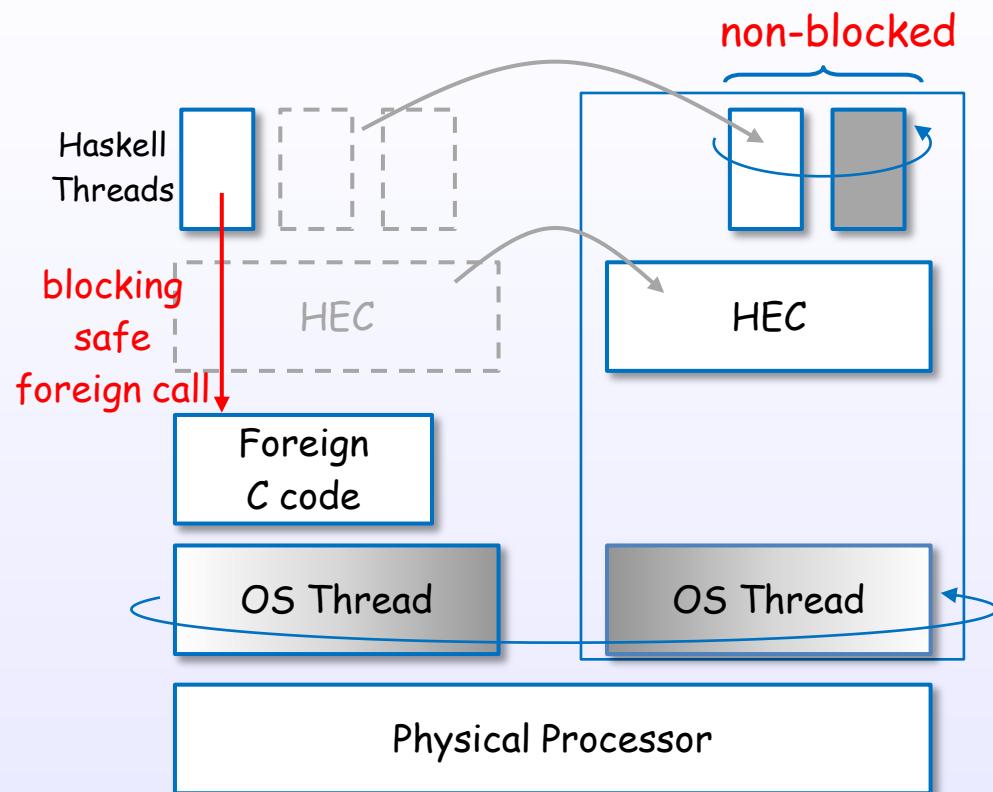
A safe foreign call (code)

Haskell Threads

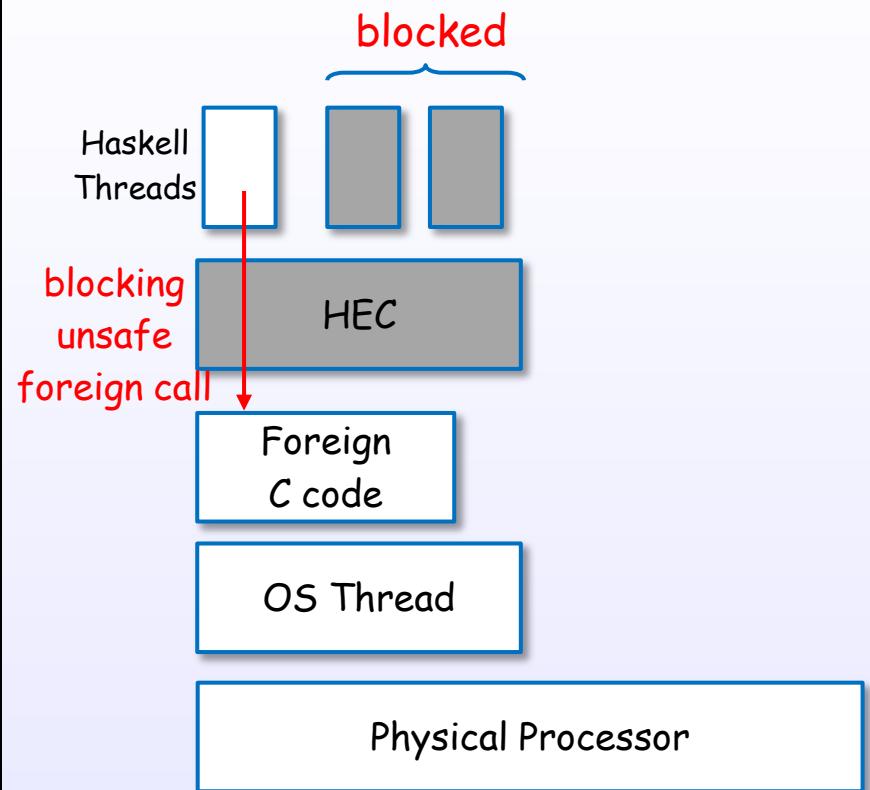


a safe and an unsafe foreign call

a **safe** foreign call



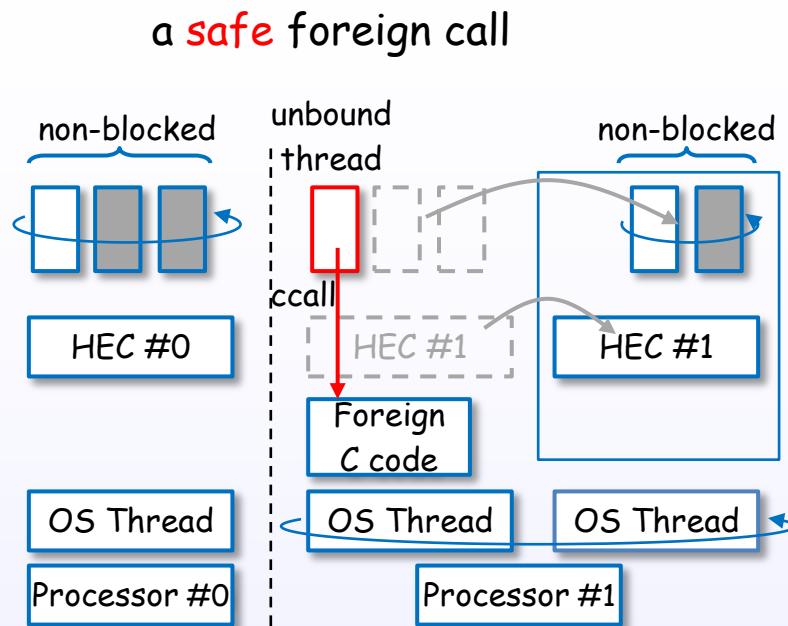
an **unsafe** foreign call



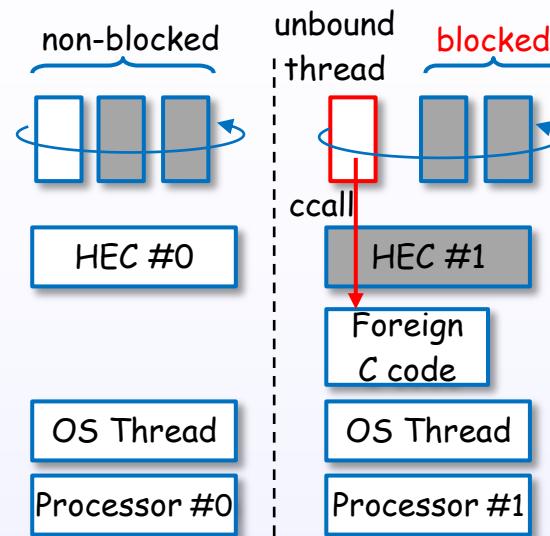
faster,
but blocking to the other Haskell threads

Safe/unsafe foreign call and bound/unbound thread

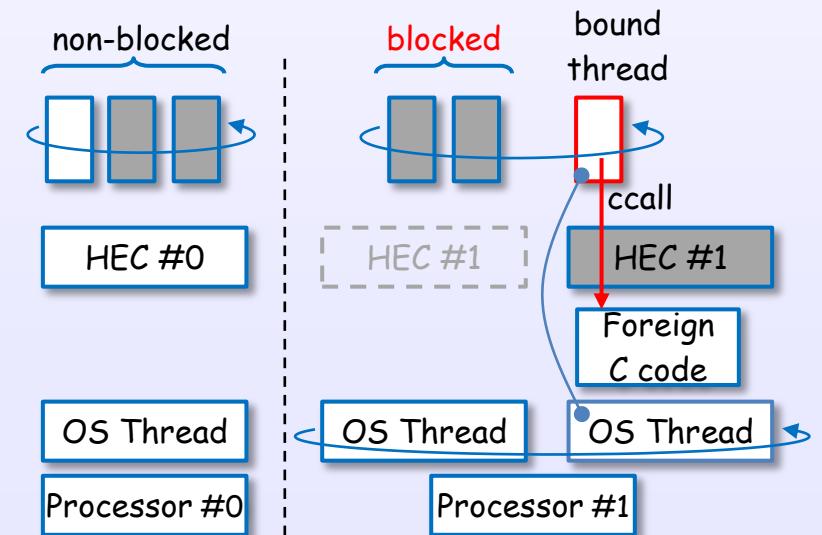
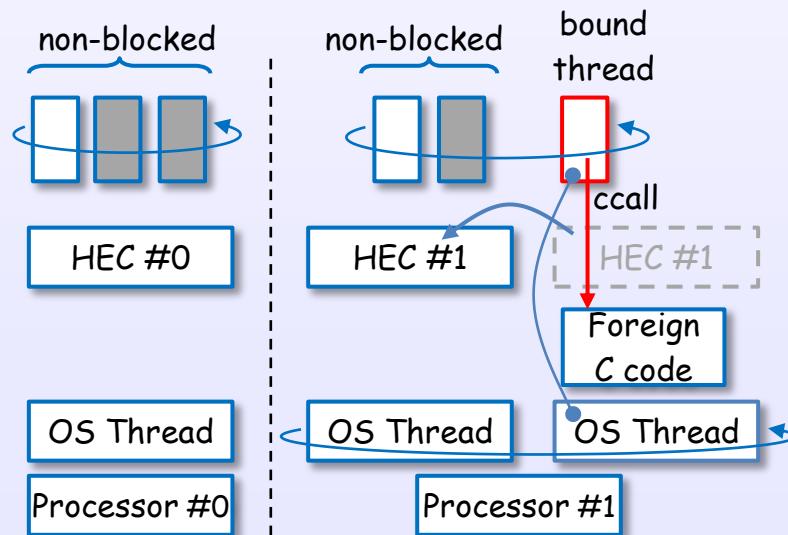
an **unbound**
thread



an **unsafe** foreign call



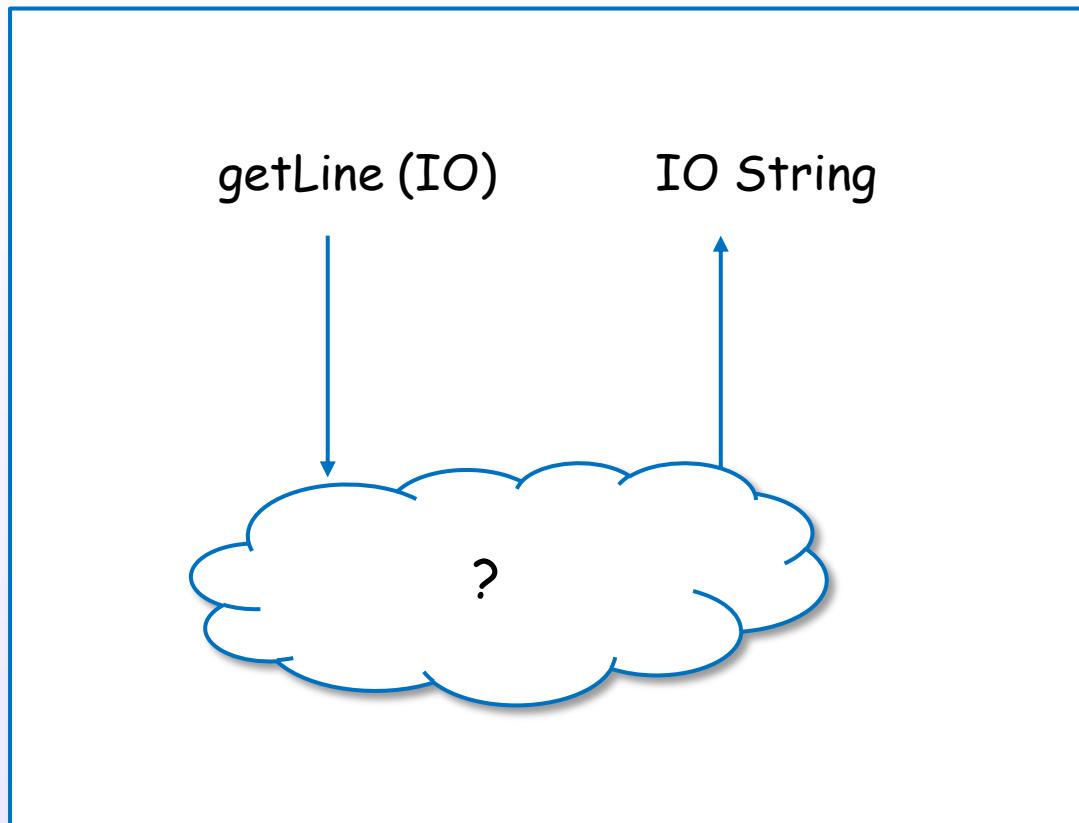
a **bound**
thread



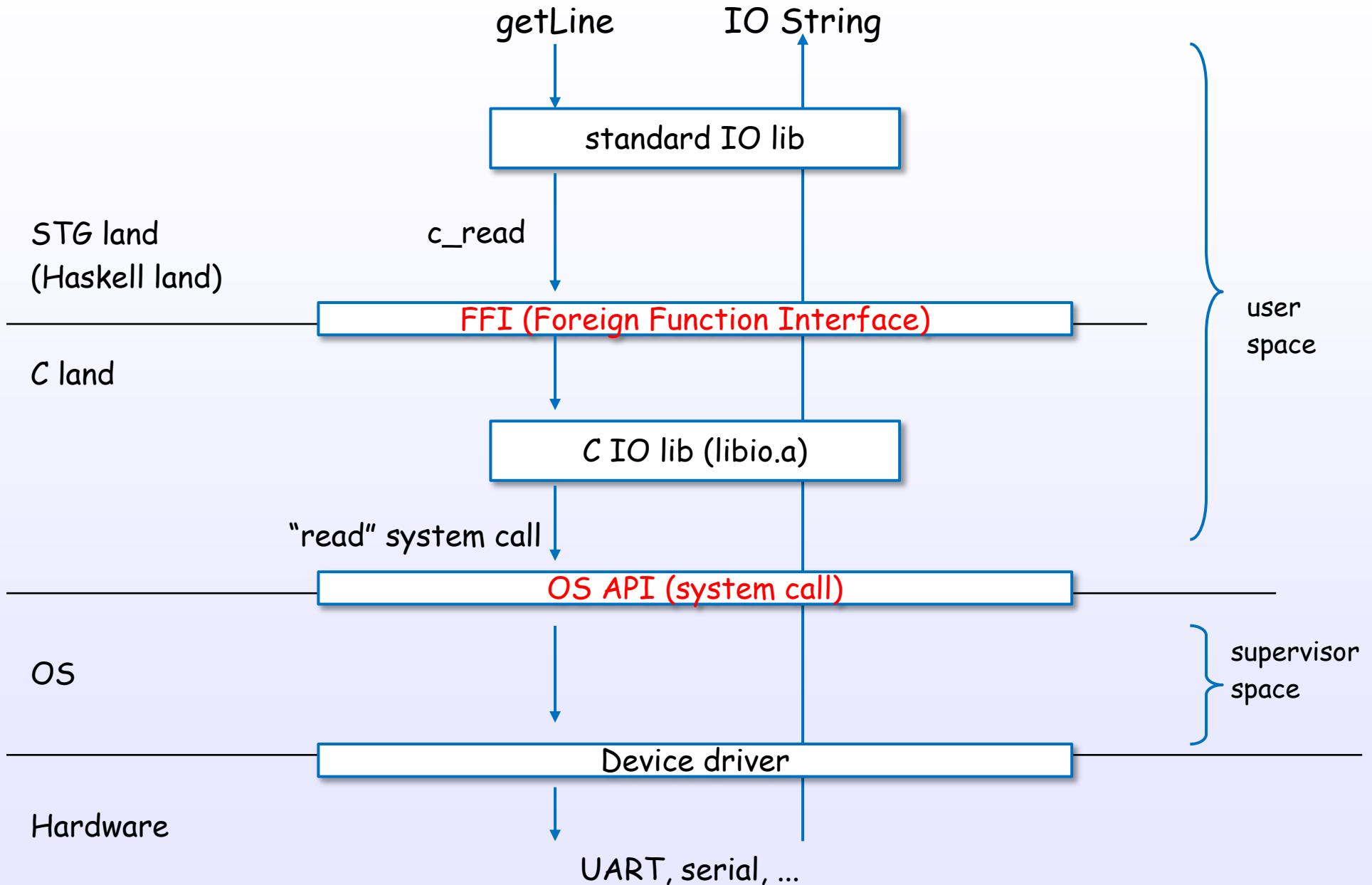
IO and FFI

IO

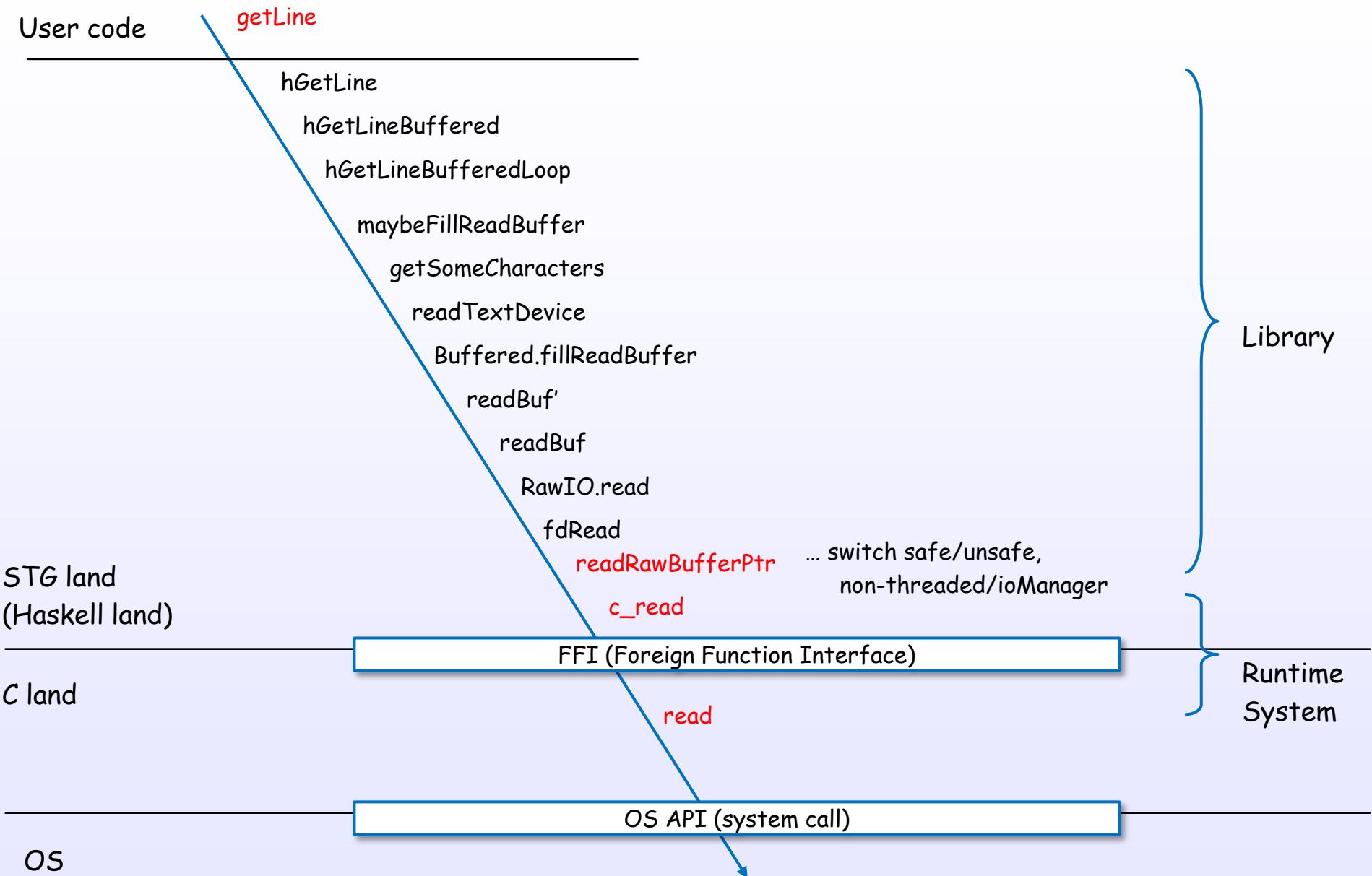
Haskell Thread



IO example: getLine

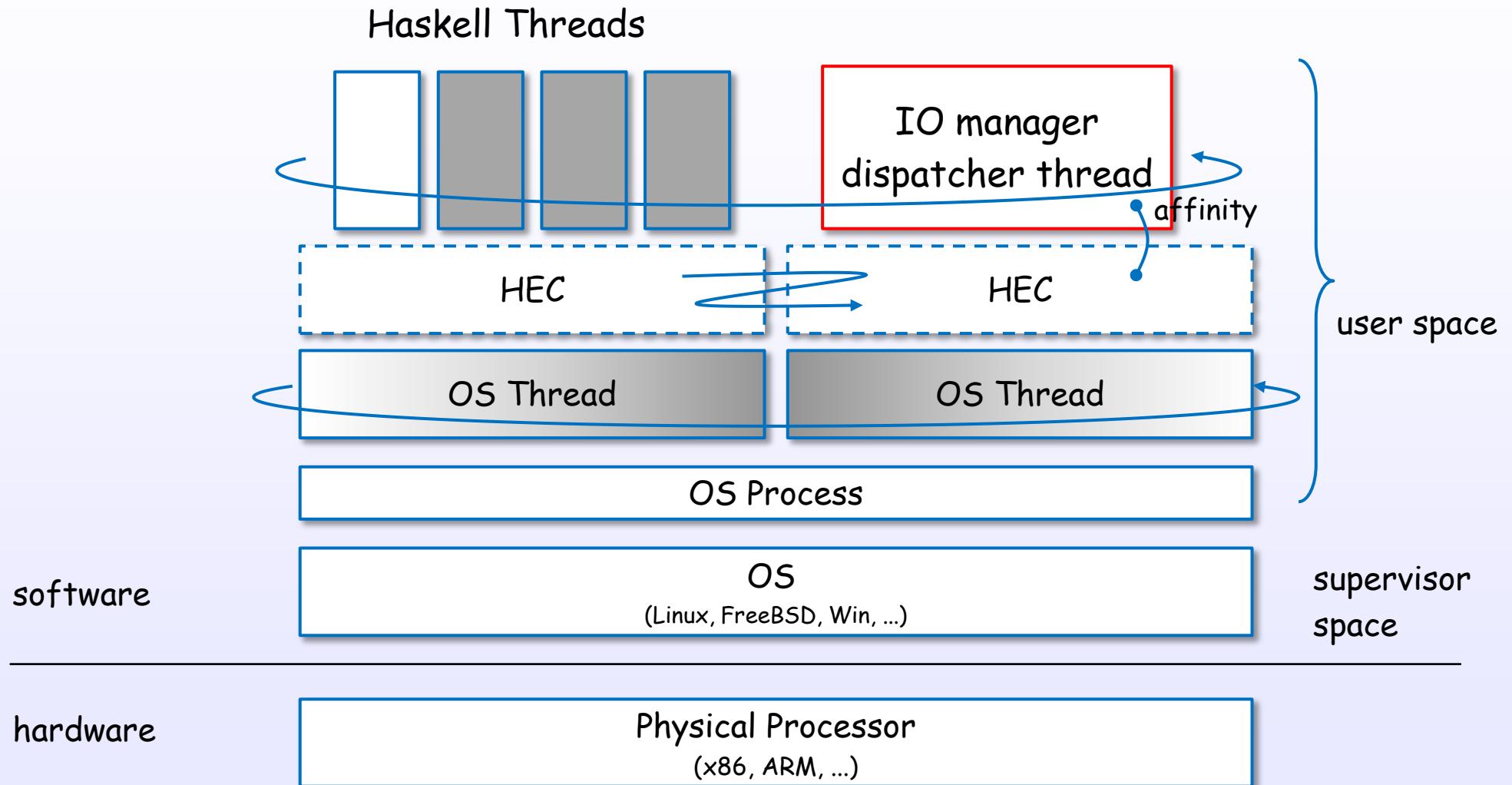


IO example: getLine (code)



IO manager

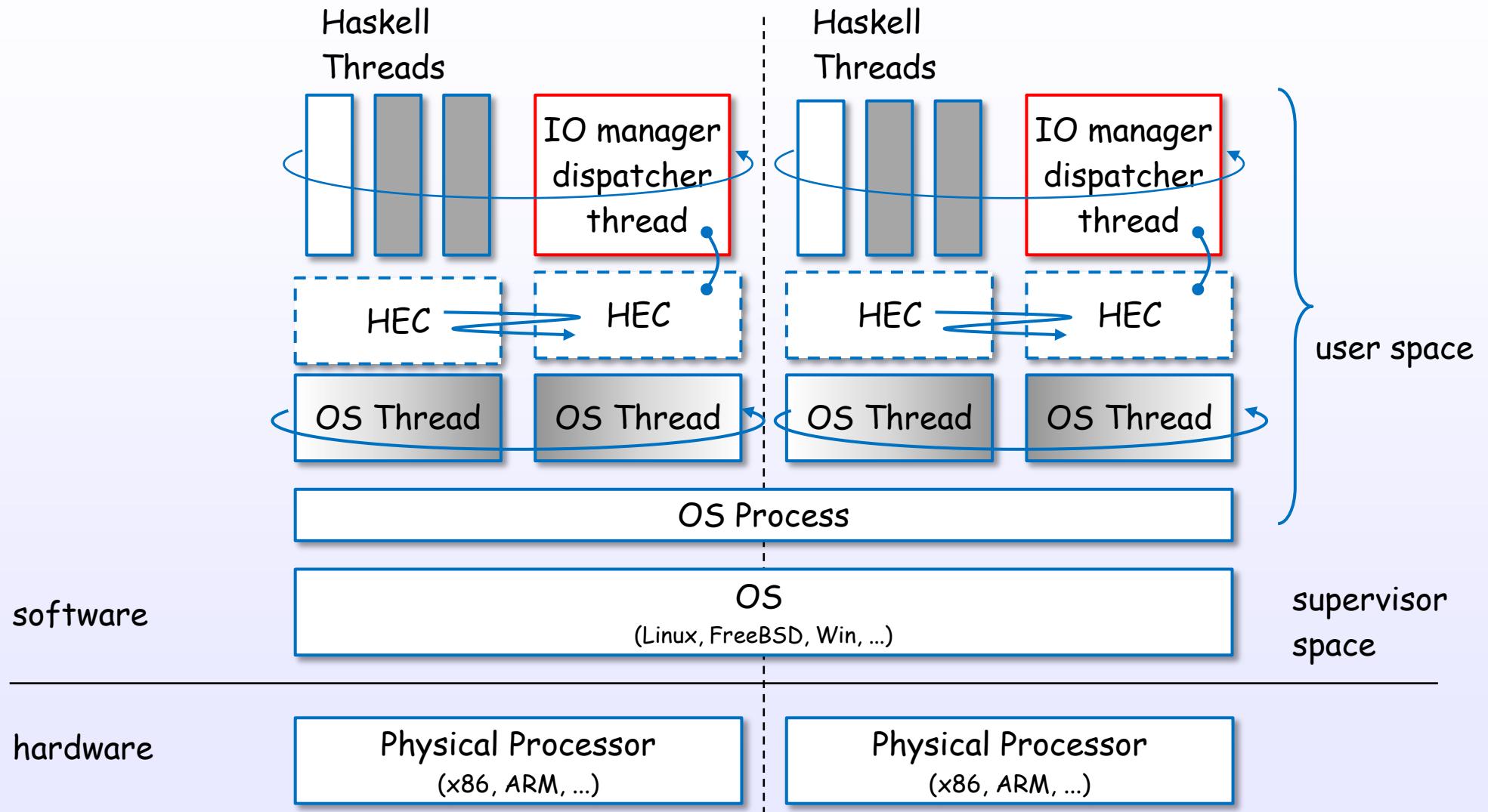
IO manager (single core)



*Threaded option case (ghc -threaded)

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

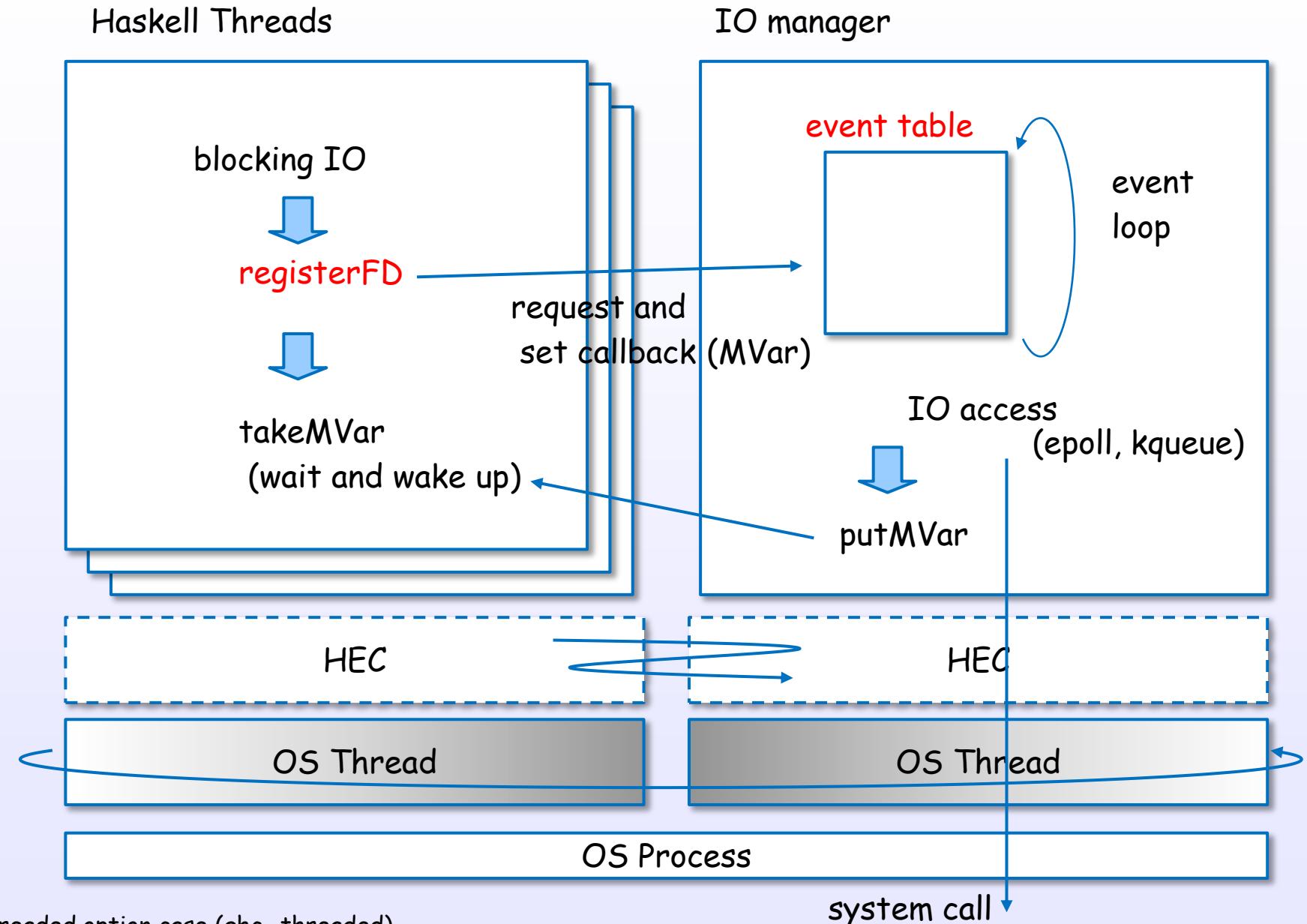
IO manager (multi core)



*Threaded option case (ghc -threaded)

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

IO manager

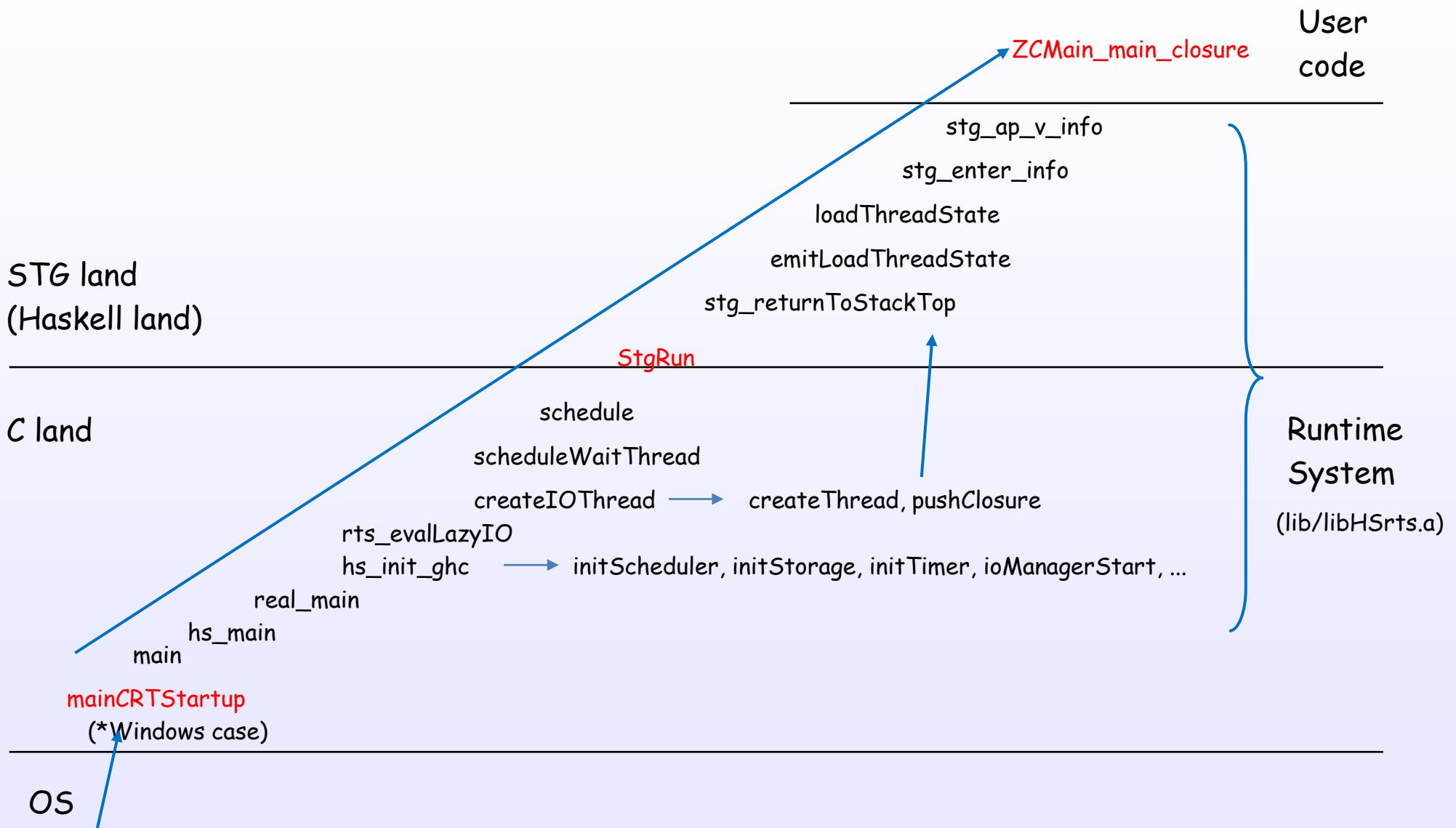


*Threaded option case (ghc -threaded)

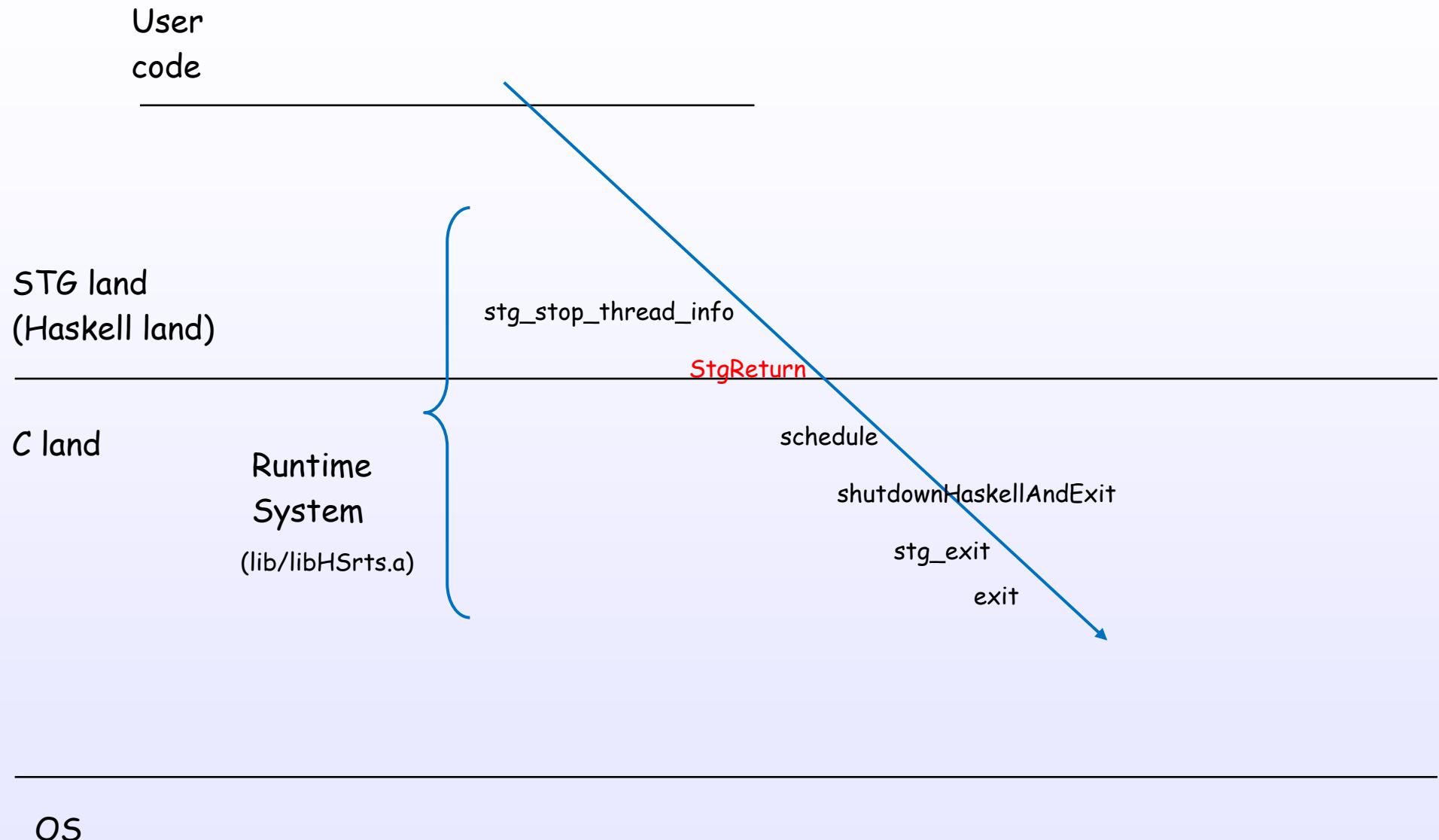
References : [7], [5], [8], [S29], [S30], [S32], [S37], [S35], [S3]

Bootstrap

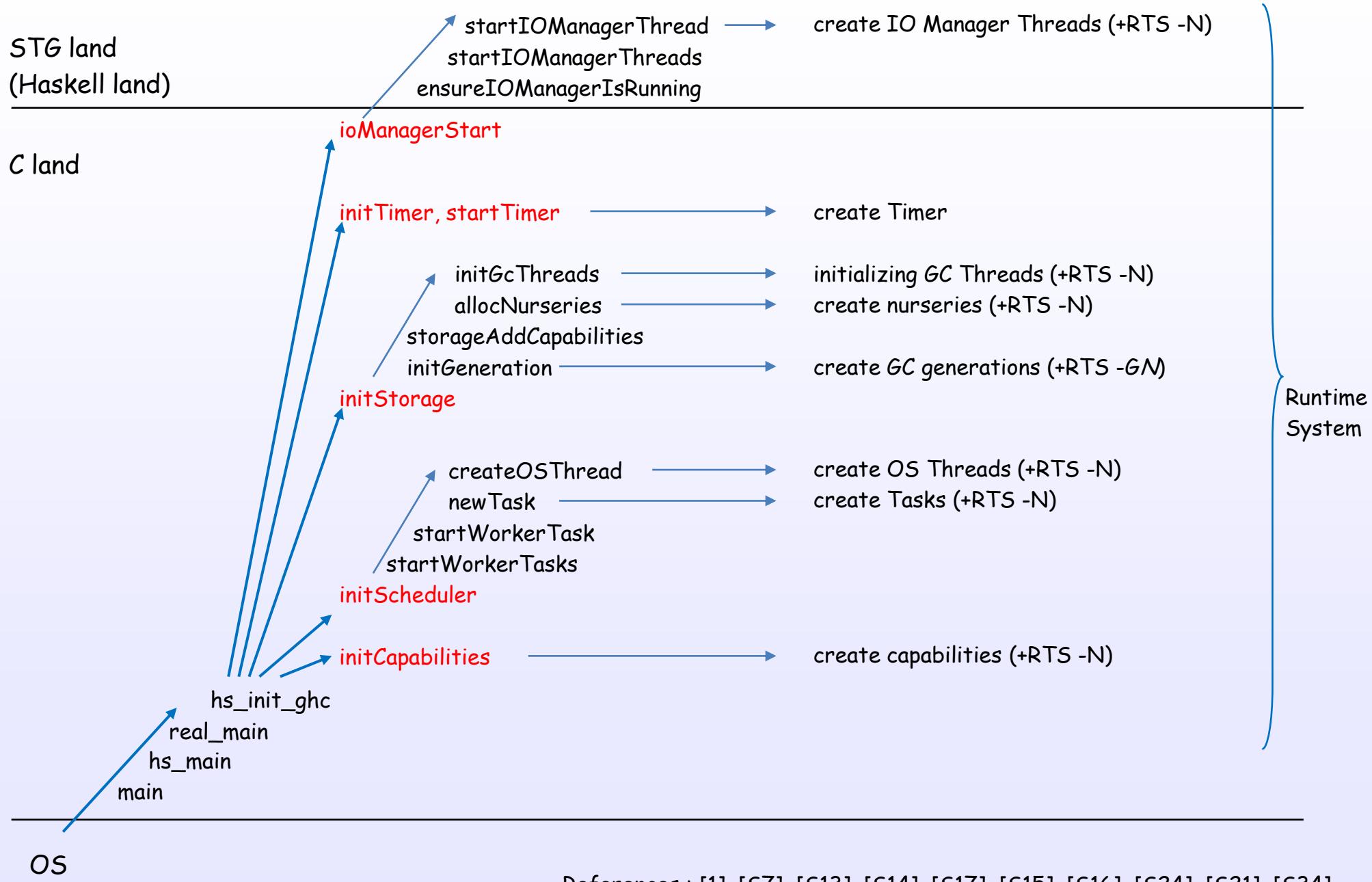
Bootstrap sequence



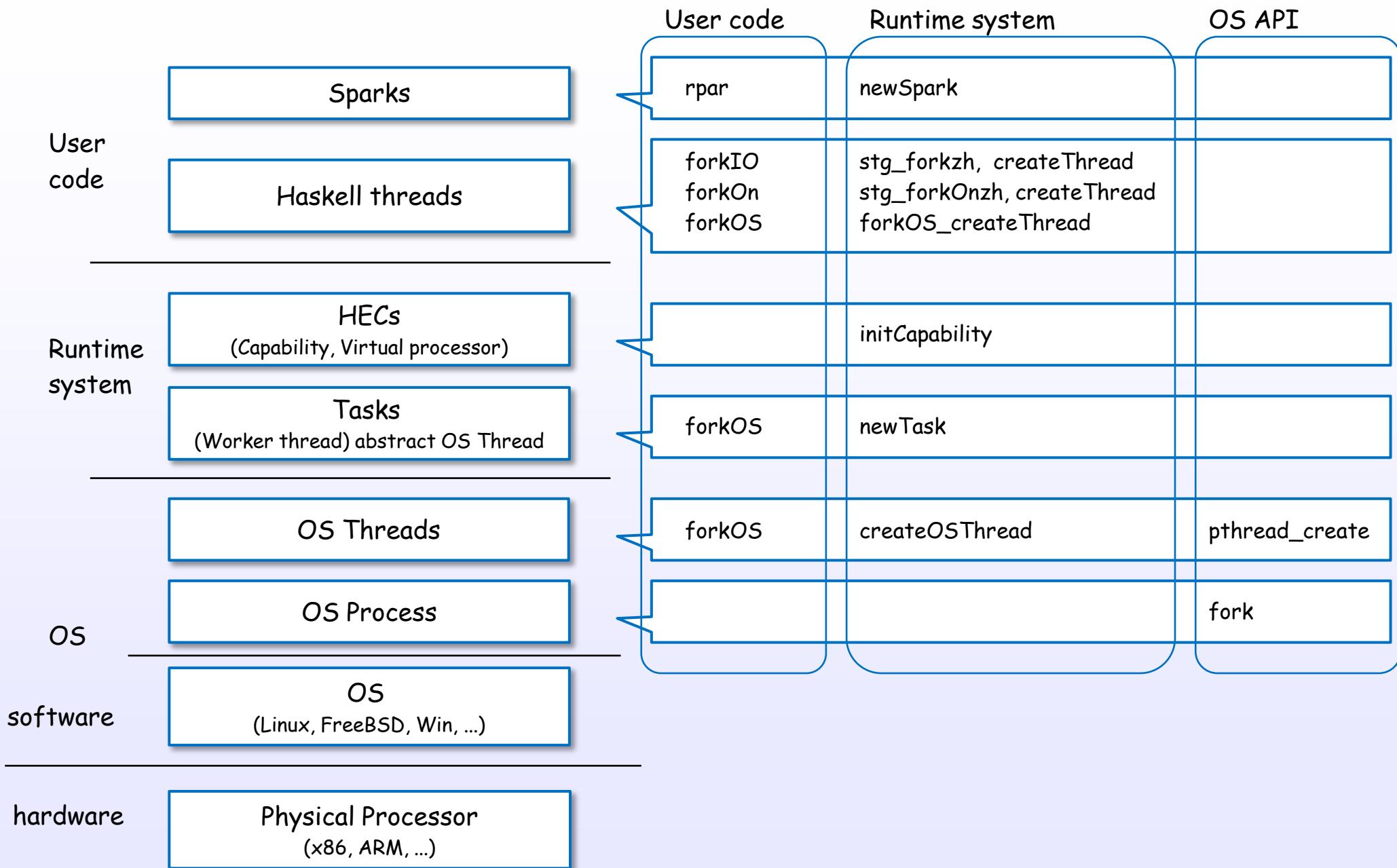
Exit sequence



Initializing



Create each layers



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
<http://research.microsoft.com/en-us/um/people/simonpj/Papers/spineless-tagless-gmachine.ps.gz>
- [3] 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/>
- [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
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
<http://www.haskell.org/ghc/docs/papers/concurrent-haskell.ps.gz>
- [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
<http://community.haskell.org/~simonmar/pcph/>
- [20] 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