NICTA

L4.verified: An Overview

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

ANU THE AUSTRALIAN NATIONAL UNIVERSITY
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT Industry & Investment
Victoria The Place To Be
THE UNIVERSITY OF MELBOURNE
THE UNIVERSITY OF SYDNEY
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

The L4.verified project aims to formally verify the functional correctness of the seL4 microkernel …

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Michael Norrish, Rafal Kolanski, Thomas Sewell, Harvey Tuch and Simon Winwood. **seL4: Formal verification of an OS Kernel.** In 22nd SOSP 2009.

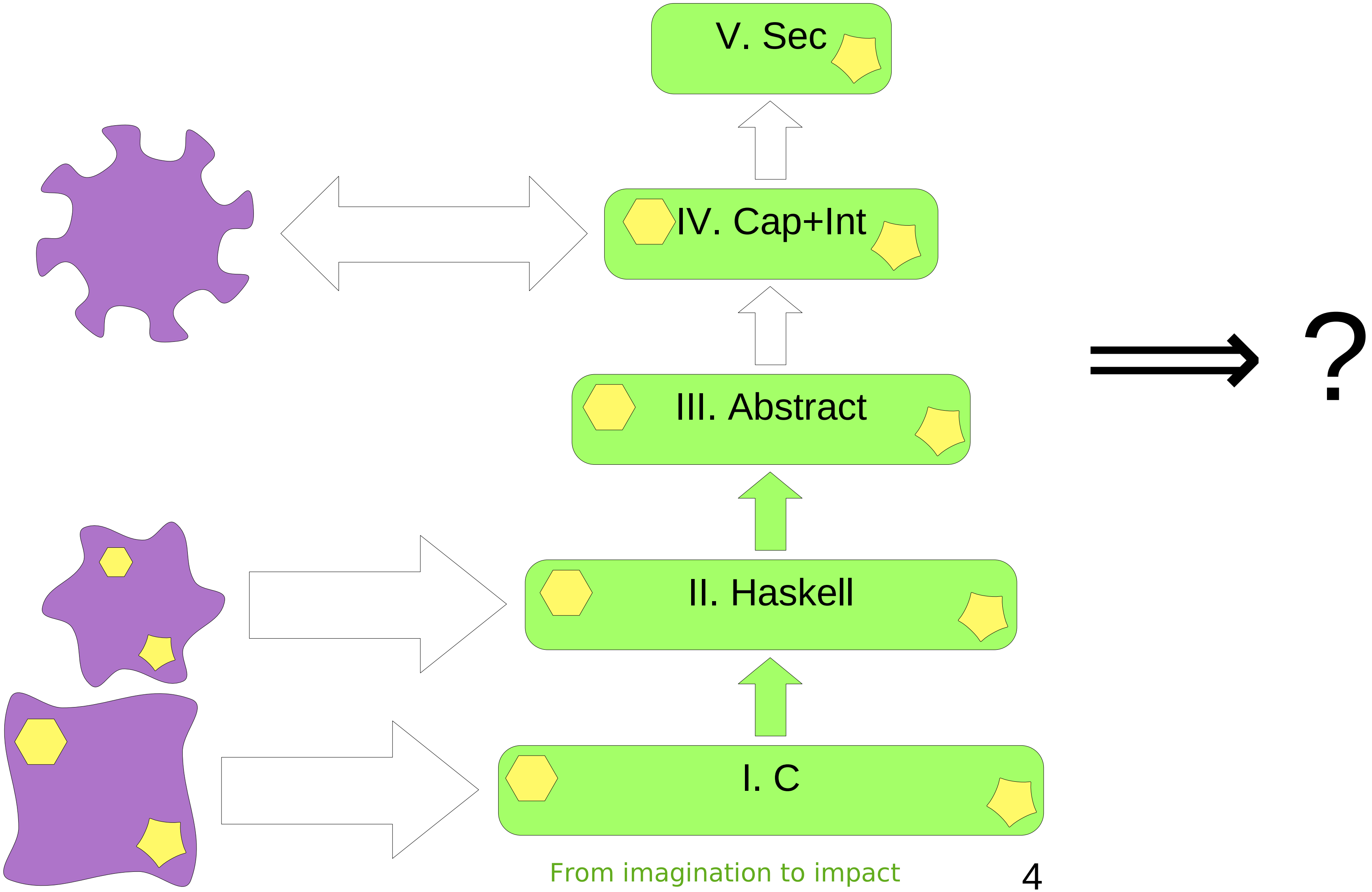
Klein et al. **seL4: Formal verification of an Operating-System Kernel.** In CACM 2010 No. 6.

seL4

NICTA

- Microkernel developed at UNSW/NICTA
- L4-style IPC
- Capability-based access control
- Capability-based control of kernel memory layout

Refinement



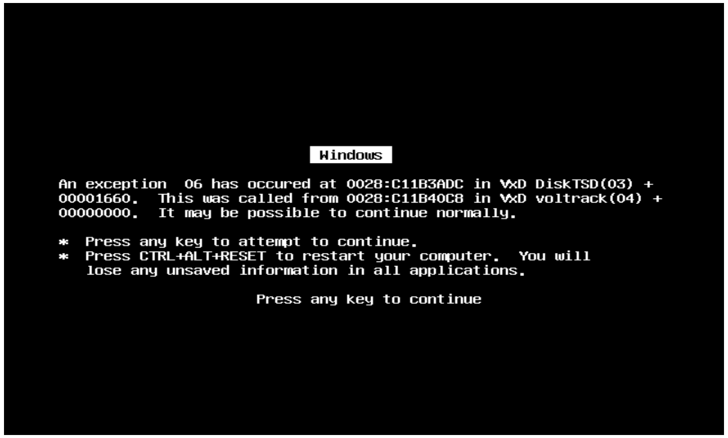
What do we want to show?

NICTA

$$\Longrightarrow ?$$

- We want to develop an OS that can't crash.

- We want to address old frustrations about security & reliability.



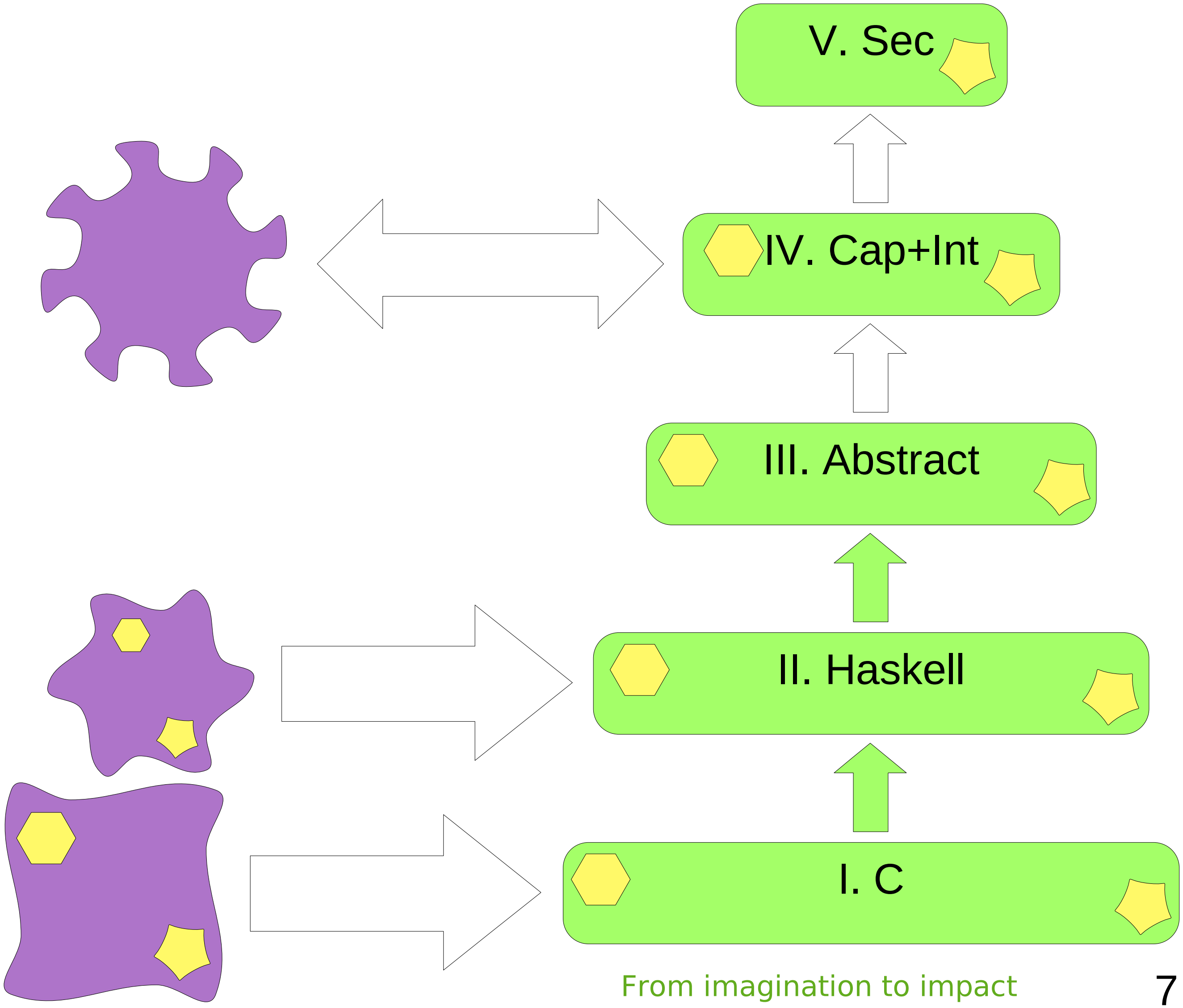
What do we want to show?

$$\Longrightarrow ?$$

- seL4 has explicit memory management
  - Guaranteed to provide a level of service
  - Few shared global objects
  - Service is provided to capability holders

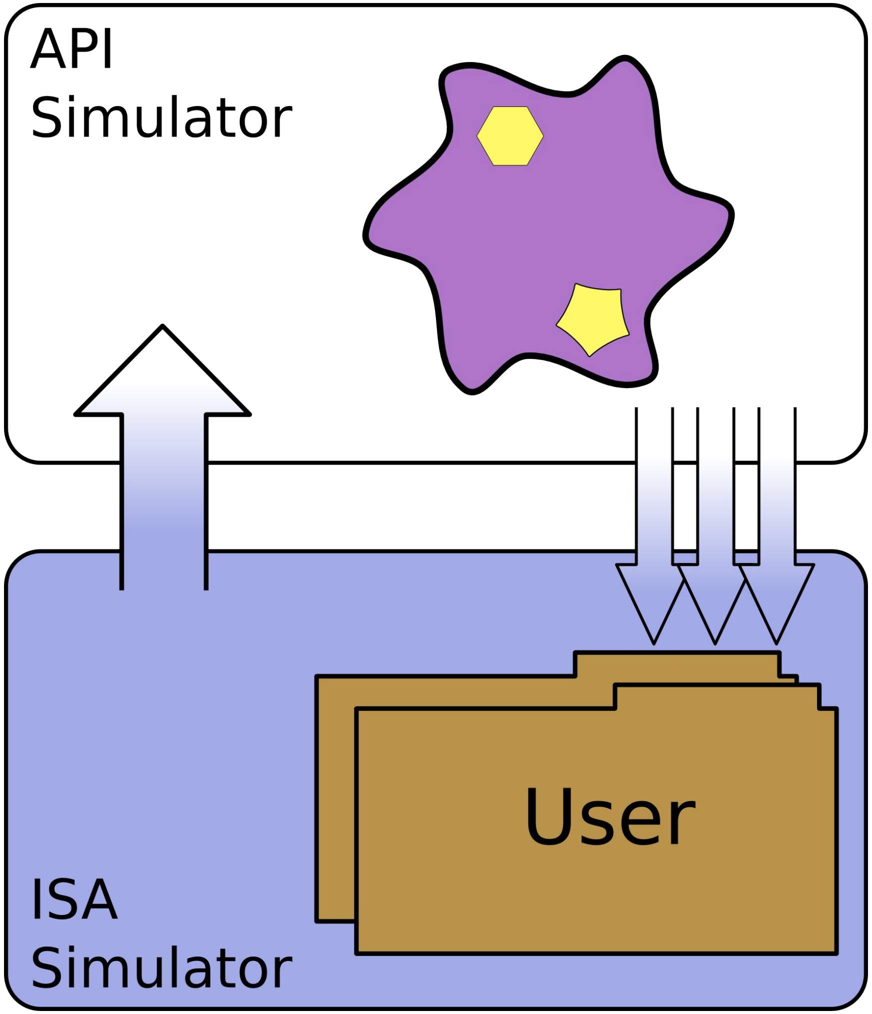
Refinement

NICTA

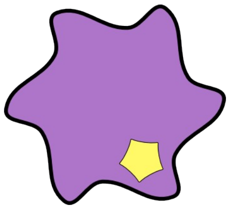


Haskell Prototype

- Predates L4.verified
- Written as a prototype
- Allowed the API to be exercised while still incomplete



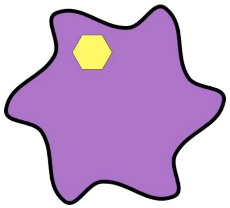
Haskell Prototype



- Written in Haskell
- Written as an implementation guide
- Simulates C with:
  - pointers
  - state
  - early return
- Has capabilities (caps)
- Full of details
  - including doubly linked lists

```
emptySlot :: PPtr CTE -> Maybe IRQ -> Kernel ()
emptySlot slot irq = do
    newCTE <- getCTE slot
    let mdbNode = cteMDBNode newCTE
    let prev = mdbPrev mdbNode
    let next = mdbNext mdbNode

    case (cteCap newCTE) of
        NullCap   -> return ()
        _         -> do
            updateMDB prev (\mdb -> mdb { mdbNext = next })
            updateMDB next (\mdb -> mdb {
                    mdbPrev = prev,
                    mdbFirstBadged = mdbFirstBadged mdb
                      || mdbFirstBadged mdbNode })
            updateCap slot NullCap
            updateMDB slot (const nullMDBNode)

            case irq of
                Just irq  -> deletedIRQHandler irq
                Nothing   -> return ()
```

Haskell Prototype



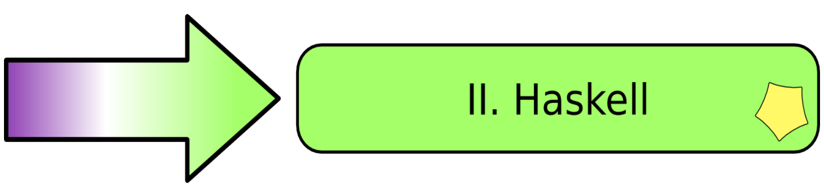
- Haskell lists and list operations are used for thread queues
  - Size at any address is not bounded
- Explicit recursion is usually avoided

```
chooseThread :: Kernel ()
chooseThread = do
        r <- findM chooseThread' (reverse [minBound ..
maxBound])
        when (r == Nothing) $ switchToIdleThread
    where
        chooseThread'' :: PPtr TCB -> Kernel Bool
        chooseThread'' thread = do
            runnable <- isRunnable thread
            if not runnable
                then do
                    tcbSchedDequeue thread
                    return False
                else do
                    switchToThread thread
                    return True
        chooseThread' :: Priority -> Kernel Bool
        chooseThread' prio = do
            q <- getQueue prio
            liftM isJust $ findM chooseThread'' q
```

Haskell Specification

- Isabelle translation of Haskell specification
  - Similar to the Haskabelle approach
  - Not a faithful representation of the semantics of Haskell.

```
defs emptySlot_def:
"emptySlot slot irq≡ (do
   newCTE ← getCTE slot;
   mdbNode ← return ( cteMDBNode newCTE);
   prev ← return ( mdbPrev mdbNode);
   next ← return ( mdbNext mdbNode);
   (case (cteCap newCTE) of
        NullCap ⇒   return ()
      | _ ⇒   (do
          updateMDB prev (λ mdb. mdb (| mdbNext := next |) );
          updateMDB next (λ mdb. mdb (|
                mdbPrev := prev,
                mdbFirstBadged :=
                   mdbFirstBadged mdb ∨ mdbFirstBadged mdbNode |) );
          updateCap slot NullCap;
          updateMDB slot (const nullMDBNode);
          (case irq of
               Some irq ⇒   deletedIRQHandler irq
             | None ⇒   return ()
             )
      od)
   )
od)"
```

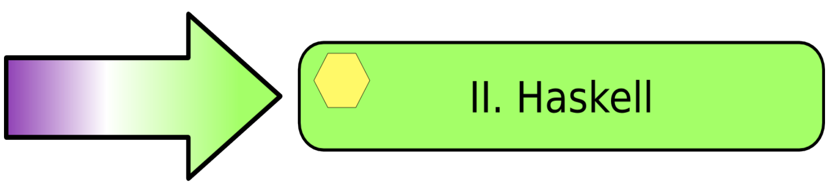


Haskell Specification

NICTA

- Isabelle translation of Haskell specification
  - Similar to the Haskabelle approach
  - Not a faithful representation of the semantics of Haskell.

```
defs chooseThread_def:
"chooseThread ≡
  let
    chooseThread'' = (λ  thread. (do
      runnable ← isRunnable thread;
      if Not runnable
        then (do
          tcbSchedDequeue thread;
          return False
        od)
        else (do
          switchToThread thread;
          return True
        od)
    od));
    chooseThread' = (λ  prio. (do
      q ← getQueue prio;
      liftM isJust $ findM chooseThread'' q
    od))
  in
            (do
    r ← findM chooseThread' (reverse [minBound  .e.  maxBound]);
    when (r = Nothing) $ switchToIdleThread
            od)"
```

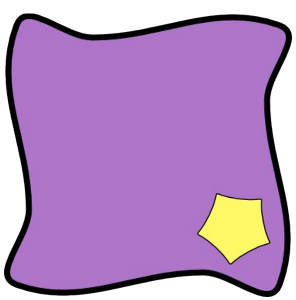


C Implementation

NICTA

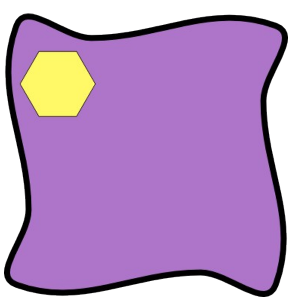
- Implementation of seL4 in C
- Written with the Haskell prototype as an implementation guide.
- Uses custom bitfield implementation rather than unions

```
static void
emptySlot(cte_t *slot, irq_t irq) {
    if(cap_get_capType(slot->cap) != cap_null_cap) {
        mdb_node_t mdbNode;
        cte_t *prev, *next;

        mdbNode = slot->cteMDBNode;
        prev = CTE_PTR(mdb_node_get_mdbPrev(mdbNode));
        next = CTE_PTR(mdb_node_get_mdbNext(mdbNode));

        if(prev)
            mdb_node_ptr_set_mdbNext(&prev->cteMDBNode,
                CTE_REF(next));
        if(next)
            mdb_node_ptr_set_mdbPrev(&next->cteMDBNode,
                CTE_REF(prev));
        if(next)
            mdb_node_ptr_set_mdbFirstBadged(&next->cteMDBNode,
                    mdb_node_get_mdbFirstBadged(next->cteMDBNode) ||
                    mdb_node_get_mdbFirstBadged(mdbNode));
        slot->cap = cap_null_cap_new();
        slot->cteMDBNode = nullMDBNode;

        if(irq != irqInvalid) deletedIRQHandler(irq);
    }
}
```



C Implementation

- Implementation of seL4 in C
- Written with the Haskell prototype as an implementation guide.
- Frequently more compact than the Haskell code.



```
void
chooseThread(void) {
    prio_t prio;
    tcb_t *thread, *next;

    for(prio = maxPrio; prio >= 0; prio--) {
        for(thread = ksReadyQueues[prio].head;
            thread; thread = next) {

            if(!isRunnable(thread)) {
                next = thread->tcbSchedNext;
                tcbSchedDequeue(thread);
            }
            else {
                switchToThread(thread);
                return;
            }
        }
    }

    switchToIdleThread();
}
```

Haskell vs C

NICTA

- Haskell was written as a prototype and implementation guide by OS team
- Aimed to capture C behaviour
- Abstracted some details:
  - datatypes not tagged unions
  - lambdas not field updates
  - list operations not loops
- Ideal formal abstraction of C code

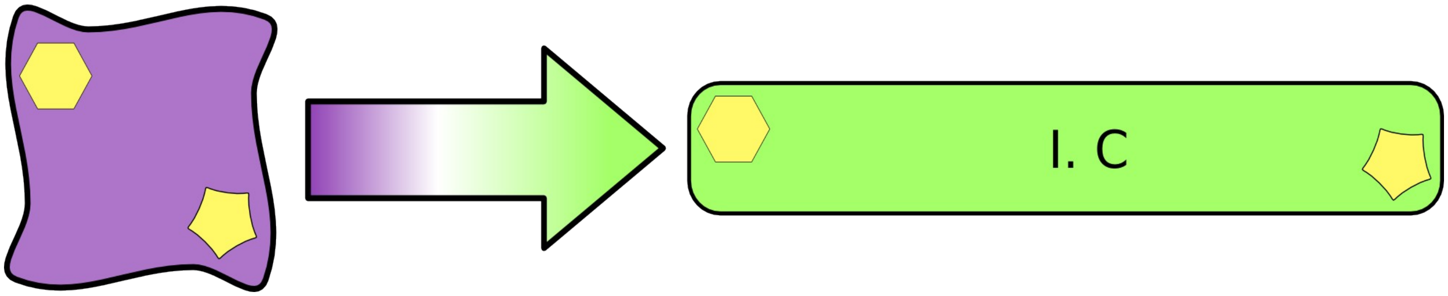
```
newCTE <- getCTE slot
let mdbNode = cteMDBNode newCTE
let prev = mdbPrev mdbNode
let next = mdbNext mdbNode

updateMDB prev (\mdb -> mdb { mdbNext = next })
```

```
mdbNode = slot->cteMDBNode;
prev = CTE_PTR(mdb_node_get_mdbPrev(mdbNode));
next = CTE_PTR(mdb_node_get_mdbNext(mdbNode));

if(prev)
    mdb_node_ptr_set_mdbNext(&prev->cteMDBNode,
        CTE_REF(next));
```

C Specification

- Import of C code into Isabelle/HOL
- Parser is written in ML by Michael Norrish.
- Uses Norbert Schirmer's SimpL language framework.
- Uses Harvey Tuch's C memory semantics

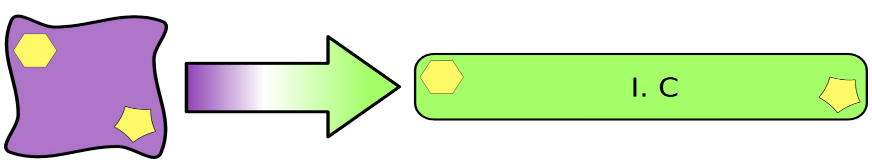
```
theory Kernel_C imports ARMMachineTypes CTranslation

begin

install_C_file packed "c/kernel_all.c_pp"
  [machinety=machine_state]

end
```



C Specification

- Import of C code into Isabelle/HOL
- Parser is written in ML by Michael Norrish.
- Uses Norbert Schirmer's SimpL language framework.
- Uses Harvey Tuch's C memory semantics

- Conservative
- Detailed

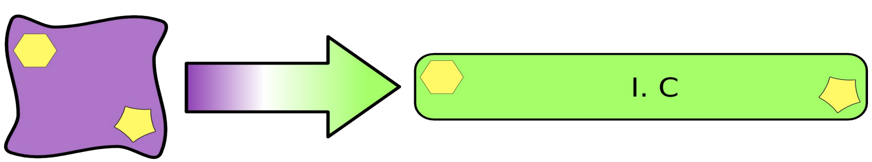



```
emptySlot_body ==
TRY
  Guard C_Guard {|c_guard (Ptr &(´slot→["cap_C"]))|}
   (´ret__unsigned_long :== CALL cap_get_capType_'proc(h_val
                          (hrs_mem ´t_hrs)
                          (Ptr &(´slot→["cap_C"]))));;
  IF ´ret__unsigned_long ~= cap_null_cap THEN
    ´mdbNode :== arbitrary;;
    ´prev :== arbitrary;;
    ´next__ptr_to_struct_cte_C :== arbitrary;;
    Guard C_Guard
     {|c_guard (Ptr &(´slot→["cteMDBNode_C"]))|}
     (´mdbNode :==
        h_val (hrs_mem ´t_hrs)
         (Ptr &(´slot→["cteMDBNode_C"])));;
    ´ret__unsigned_long :== CALL mdb_node_get_mdbPrev_'proc(´mdbNode);;
    ´prev :== Ptr (ucast ´ret__unsigned_long);;
    ´ret__unsigned_long :== CALL mdb_node_get_mdbNext_'proc(´mdbNode);;
    ´next__ptr_to_struct_cte_C :==
      Ptr (ucast ´ret__unsigned_long);;
    IF ´prev ~= NULL THEN
      Guard C_Guard
       {|c_guard (Ptr &(´prev→["cteMDBNode_C"]))|}
       (CALL mdb_node_ptr_set_mdbNext_'proc(Ptr
       &(´prev→["cteMDBNode_C"]),
         ucast (ptr_val ´next__ptr_to_struct_cte_C)))
    FI;;

   ...
```

C Specification

- Import of C code into Isabelle/HOL
- Parser is written in ML by Michael Norrish.
- Uses Norbert Schirmer's SimpL language framework.
- Uses Harvey Tuch's C memory semantics

- Conservative
- Detailed

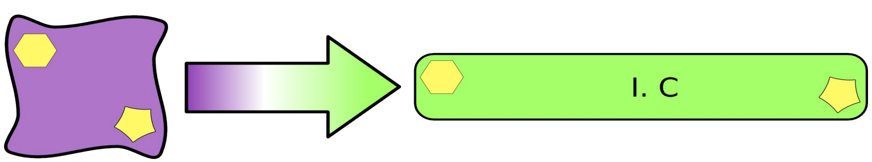



```
IF ´next__ptr_to_struct_cte_C ~= NULL THEN
  Guard C_Guard
   {|c_guard
      (Ptr &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]))|}
   (CALL mdb_node_ptr_set_mdbPrev_'proc(Ptr
   &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]),
    ucast (ptr_val ´prev)))
FI;;
IF ´next__ptr_to_struct_cte_C ~= NULL THEN
  Guard C_Guard
   {|c_guard
      (Ptr &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]))|}
   (´ret__unsigned_long :== CALL mdb_node_get_mdbFirstBadged_'proc(h_val
(hrs_mem ´t_hrs)
(Ptr &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]))));;
 ´ret__int :==
    (if ´ret__unsigned_long ~= 0 then 1 else 0);;
  IF ´ret__int ~= 0 THEN
    SKIP
  ELSE
    ´ret__unsigned_long :== CALL mdb_node_get_mdbFirstBadged_'proc(´mdbNode);;
    ´ret__int :==
      (if ´ret__unsigned_long ~= 0 then 1 else 0)
  FI;;
  Guard C_Guard
   {|c_guard
      (Ptr &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]))|}
   (CALL mdb_node_ptr_set_mdbFirstBadged_'proc(Ptr
        &(´next__ptr_to_struct_cte_C→["cteMDBNode_C"]),
    ucast ´ret__int))
FI;;
´ret__struct_cap_C :== CALL cap_null_cap_new_'proc();;
Guard C_Guard
 {|c_guard (Ptr &(´slot→["cap_C"]))|}
 (´globals :==
    t_hrs_'_update
     (hrs_mem_update
       (heap_update (Ptr &(´slot→["cap_C"]))
         ´ret__struct_cap_C)));;
´ret__struct_mdb_node_C :== CALL mdb_node_new_'proc(ucast 0,
ucast false,ucast false,ucast 0);;
Guard C_Guard
 {|c_guard (Ptr &(´slot→["cteMDBNode_C"]))|}
 (´globals :==
    t_hrs_'_update
     (hrs_mem_update
       (heap_update
         (Ptr &(´slot→["cteMDBNode_C"]))
         ´ret__struct_mdb_node_C)));;
IF ucast ´irq ~= irqInvalid THEN
  CALL deletedIRQHandler_'proc(´irq)
FI
FI
CATCH SKIP
END
```

C Specification

- Import of C code into Isabelle/HOL
- Parser is written in ML by Michael Norrish.
- Uses Norbert Schirmer's SimpL language framework.
- Uses Harvey Tuch's C memory semantics

- Conservative
- Detailed




```
chooseThread_body ==
TRY
 ´prio :== arbitrary;;
 ´thread :== arbitrary;;
 ´next :== arbitrary;;
 ´prio :== scast maxPrio;;
 WHILE 0 <=s ´prio INV {|arbitrary|} VAR arbitrary DO
  ´thread :==
   head_C (index ´ksReadyQueues (unat ´prio));;
  WHILE ´thread ~= NULL INV {|arbitrary|} VAR arbitrary DO
   ´ret__enum__bool :== CALL isRunnable_'proc(´thread);;
   IF ~ ´ret__enum__bool ~= 0 THEN
    Guard C_Guard
     {|c_guard
        (Ptr &(´thread→["tcbSchedNext_C"]))|}
     (´next :==
        h_val (hrs_mem ´t_hrs)
         (Ptr &(´thread→["tcbSchedNext_C"])));;
    CALL tcbSchedDequeue_'proc(´thread)
   ELSE
    CALL switchToThread_'proc(´thread);;
    creturn_void global_exn_var_'_update
   FI;;
   ´thread :== ´next
  OD;;
  ´prio :== ´prio - 1
 OD;;
 CALL switchToIdleThread_'proc()
CATCH SKIP
END
```

HOL Specification

NICTA

- Written directly in Isabelle/HOL
- Not executable
- Captures the intent of computations declaratively
- Abstracts data structures
  - doubly linked MDB replaced with CDT map
- Also monadic
- Also has details
  - semantic details

```
definition
  empty_slot :: "cslot_ptr ⇒ irq option ⇒ unit s_monad"
where
 "empty_slot slot free_irq ≡ do
     cap ← get_cap slot;
     if cap = NullCap then
       return ()
     else do
       cdt ← gets cdt;
       parent ← return $ cdt slot;
       set_cdt ((λp. if cdt p = Some slot
                  then parent
                  else cdt p) (slot := None));
       set_revokable slot False;
       set_cap NullCap slot;

       case free_irq of Some irq ⇒ deleted_irq_handler irq
          | None ⇒ return ()
     od
  od"
```

III. Abstract

HOL Specification

NICTA

- Written directly in Isabelle/HOL
- Not executable
- Captures the intent of computations declaratively

```
"schedule = do
     cur ← gets cur_thread;
     threads ← allActiveTCBs;
     thread ← select threads;
     if thread = cur then
       return () OR switch_to_thread thread
     else switch_to_thread thread
 od OR switch_to_idle_thread"
```

- Uses nondeterminism

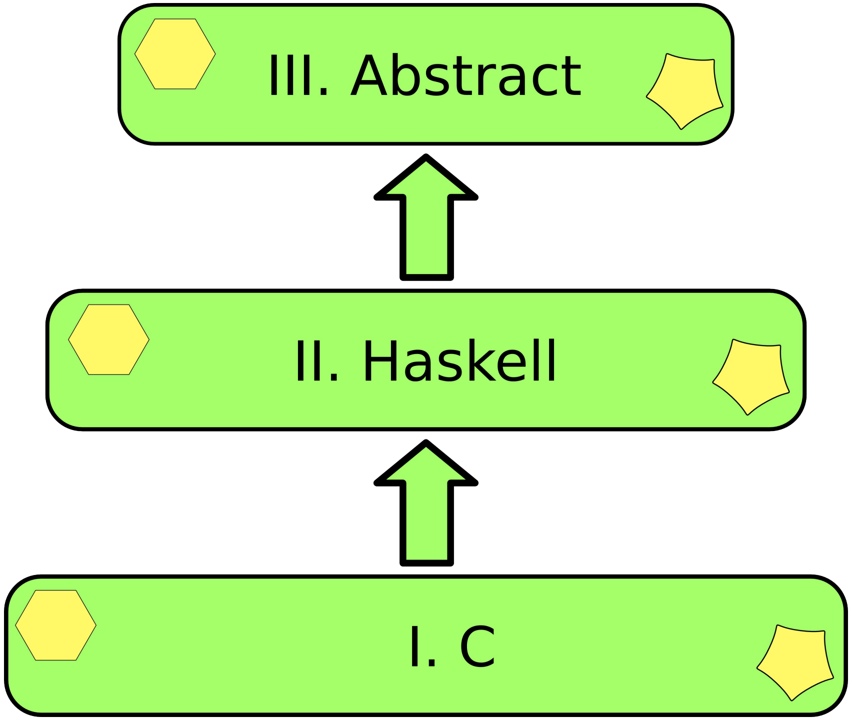
III. Abstract

Refinement

NICTA

- L4.verified aims to show refinement
- Refinement equates to the subset property on behaviours
- Refinement is transitive

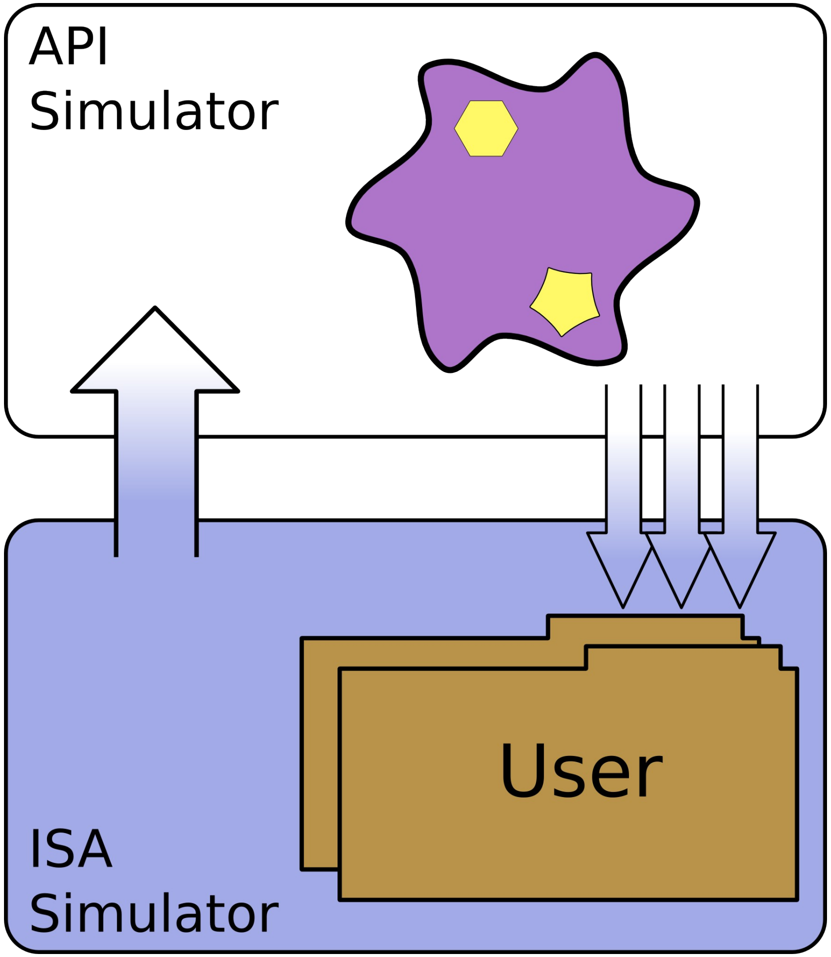
- We establish assertions as side conditions



Refinement: Behaviours

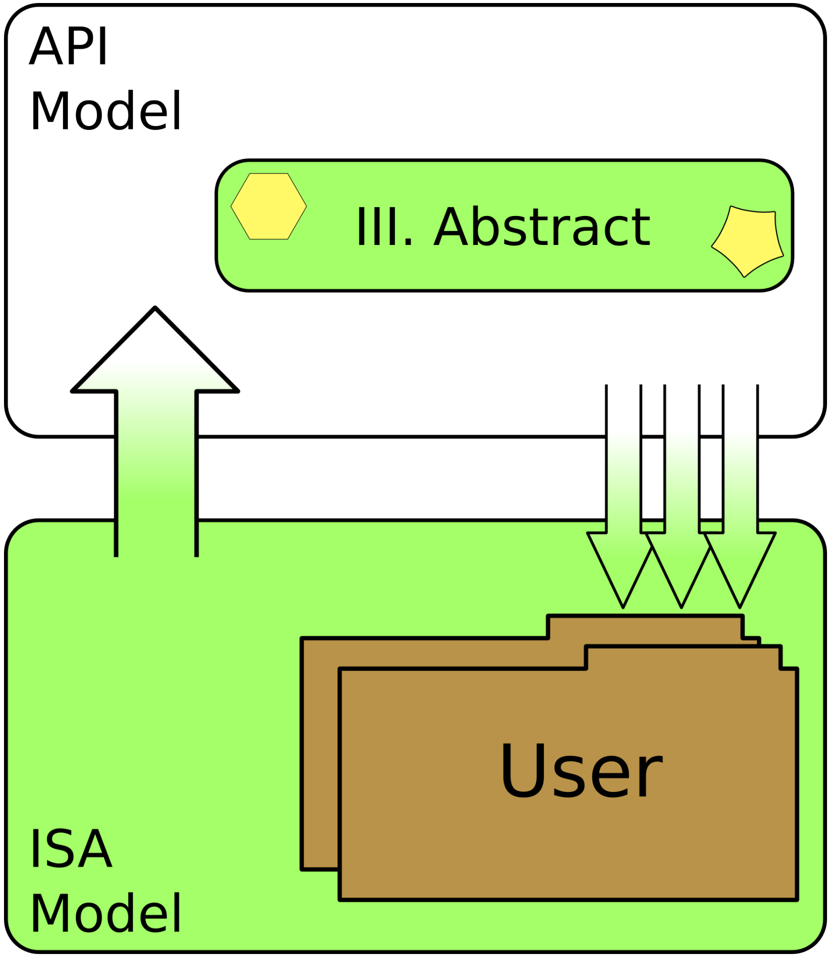
NICTA

- Recall that the Haskell specification was originally designed to fit in a simulator



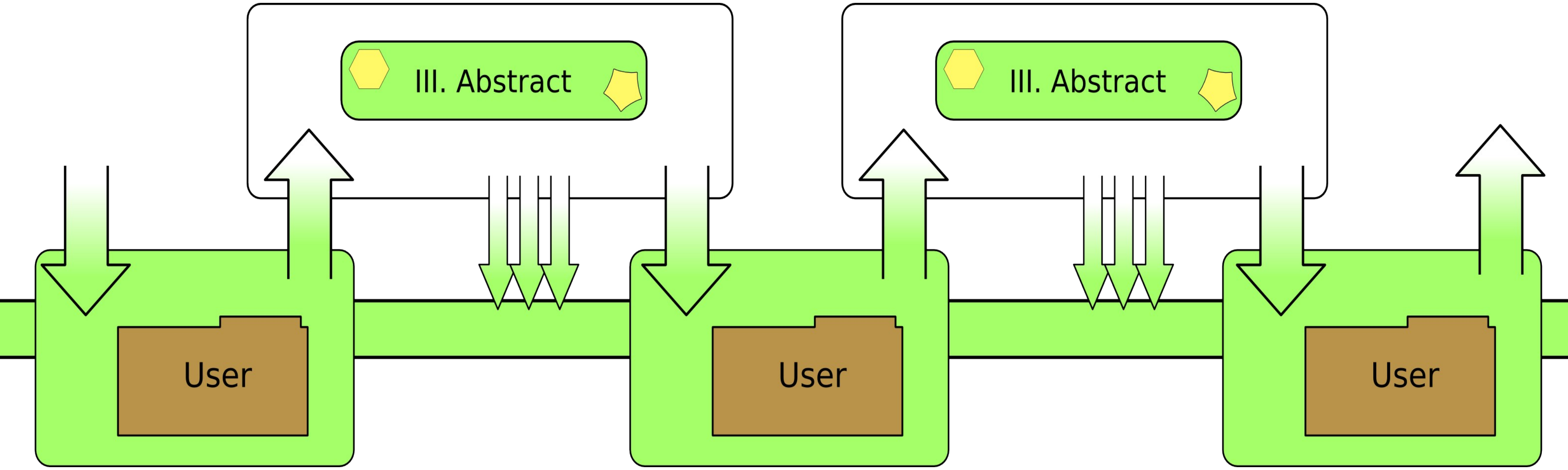
Refinement: Behaviours

- Recall that the Haskell specification was originally designed to fit in a simulator
- The formal models are conceptually used the same way.
- The simulator for user behaviour is replaced with nondeterminism



Refinement: Behaviours

- Recall that the Haskell specification was originally designed to fit in a simulator
- The formal models are conceptually used the same way.
- Behaviours of the models are traces of user and kernel interaction



Decomposition

- The specifications share common structure
  - Partly by design
  - Partly by failure of abstraction
  - Largely because of correct programming

- Verification methods should exploit this

Decomposition

NICTA

- All models have the same overall structure:
  - Figure out what the user wants/needs
  - Check if that is permitted
  - Perform an action or send an error message
  - Decide which user will run next
- Sequential decomposition of steps
- Case decomposition by action types

Decomposition Example

NICTA

```
definition
  empty_slot :: "cslot_ptr ⇒ irq option ⇒ unit s_monad"
where
 "empty_slot slot free_irq ≡ do
     cap ← get_cap slot;
     if cap = NullCap then
       return ()
     else do
       cdt ← gets cdt;
       parent ← return $ cdt slot;
       set_cdt ((λp. if cdt p = Some slot
                     then parent
                     else cdt p) (slot := None));
       set_revokable slot False;
       set_cap NullCap slot;

       case free_irq of Some irq ⇒ deleted_irq_handler irq
          | None ⇒ return ()
     od
  od"
```

```
defs emptySlot_def:
"emptySlot slot irq≡ (do
    newCTE ← getCTE slot;
    mdbNode ← return ( cteMDBNode newCTE);
    prev ← return ( mdbPrev mdbNode);
    next ← return ( mdbNext mdbNode);
    (case (cteCap newCTE) of
          NullCap ⇒   return ()
        | _ ⇒   (do
            updateMDB prev (λ mdb. mdb (| mdbNext := next |) );
            updateMDB next (λ mdb. mdb (|
                    mdbPrev := prev,
                    mdbFirstBadged :=
                        mdbFirstBadged mdb
                         ∨ mdbFirstBadged mdbNode |) );
            updateCap slot NullCap;
            updateMDB slot (const nullMDBNode);
            (case irq of
                  Some irq ⇒   deletedIRQHandler irq
                | None ⇒   return ()
                )
        od)
        )
od)"
```

Decomposition Example II

NICTA

```
"schedule = do
    cur ← gets cur_thread;
    threads ← allActiveTCBs;
    thread ← select threads;
    if thread = cur then
      return () OR switch_to_thread thread
    else switch_to_thread thread
 od OR switch_to_idle_thread"
```

```
defs chooseThread_def:
"chooseThread ≡
   let
      chooseThread'' = (λ  thread. (do
         runnable ← isRunnable thread;
         if Not runnable
            then (do
               tcbSchedDequeue thread;
               return False
            od)
            else (do
               switchToThread thread;
               return True
            od)
      od));
      chooseThread' = (λ  prio. (do
         q ← getQueue prio;
         liftM isJust $ findM chooseThread'' q
      od))
   in
               (do
      r ← findM chooseThread' (reverse [minBound  .e.  maxBound]);
      when (r = Nothing) $ switchToIdleThread
               od)"
```

Decomposition Example II

NICTA

```
corres

(do
    cur ← gets cur_thread;
    threads ← allActiveTCBs;
    thread ← select threads;
    if thread = cur then
      return () OR switch_to_thread thread
    else switch_to_thread thread
 od OR switch_to_idle_thread)
```

```
let
   chooseThread'' = (λ  thread. (do
      runnable ← isRunnable thread;
      if Not runnable
         then (do
            tcbSchedDequeue thread;
            return False
         od)
         else (do
            switchToThread thread;
            return True
         od)
   od));
   chooseThread' = (λ  prio. (do
      q ← getQueue prio;
      liftM isJust $ findM chooseThread'' q
   od))
in
              (do
   r ← findM chooseThread' (reverse [minBound  .e.  maxBound]);
   when (r = Nothing) $ switchToIdleThread
              od)
```

Decomposition Example II

NICTA

$\forall$x. corres (op =)

(return False

OR (do assert (x $\in$ S); return True))

(f x)

$\Longrightarrow$ corres (op =)

(select S) (findM f xs)

Decomposition Example II

NICTA

$\forall$x. corres (op =) (P S) UNIV
  (return False
    OR (do assert (x $\in$ S); return True))
  (f x)
$\Longrightarrow$ corres (op =) (P S) UNIV
   (select S) (findM f xs)

Decomposition Example II

NICTA

corres rvr {s. S = allActiveTCBs s} UNIV

f f'

$\Longrightarrow$ corres rvr UNIV

{s. st_tcb_at active' t s $\longrightarrow$ t $\in$ S}

f f'

Refinement: Observations

- Approach uses predicates and decomposition by introduction rules
- Both powerful and fiddly
- Challenge to automate

Haskell/C Refinement

- Refinement between Haskell and C proceeds similarly
- Very similar sequential structure
- C introduces new decomposition challenges
  - Local variables
  - Early return

Evaluation

- Refinement process works.
- 200,000 lines of proof.
- Estimates of effort:
  - 15% on apparatus (C parser, memory semantics, Haskell translator)
  - 10-15% on Abstract/Haskell refinement
  - 20% on Haskell/C refinement

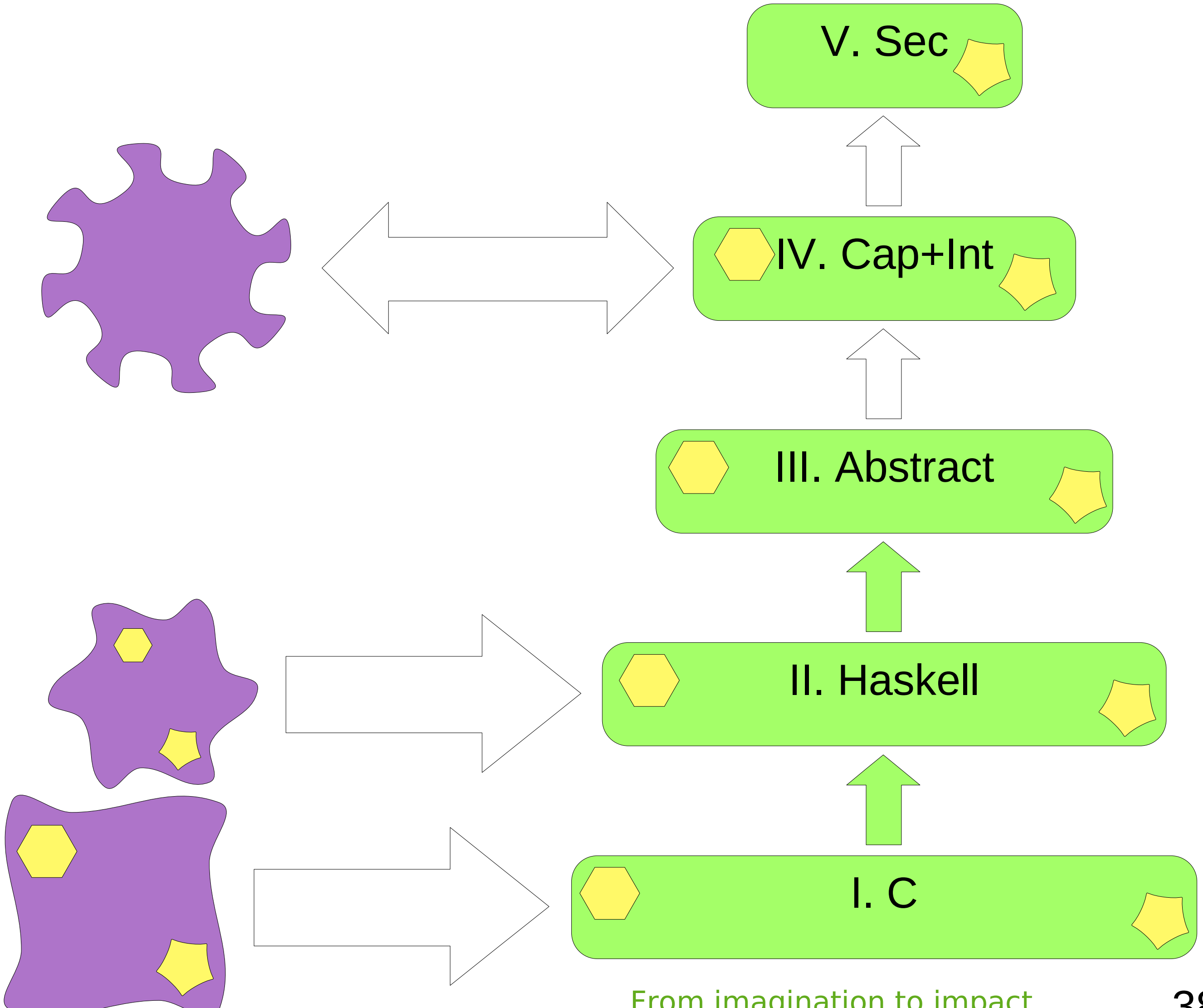
What have we shown?

$$\Longrightarrow ?$$

- The system never behaves exceptionally
- The system always behaves as you would expect
  – If you understood the specification very well

- We've met our simplistic goals.
- How about security & reliability?

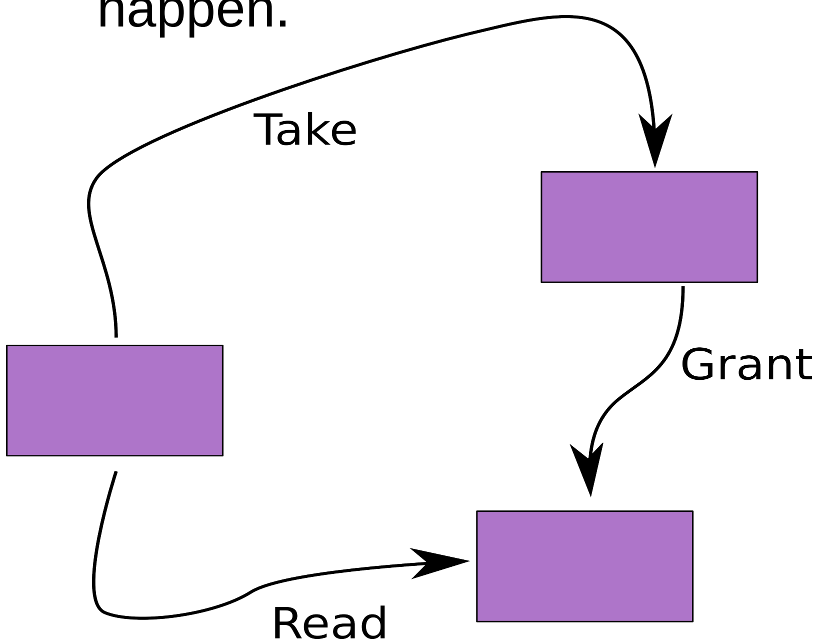
Refinement



SEC Specification

NICTA

- Abstraction of API to a graph of entities connected by capabilities
- Captures what operations are possible, not why operations happen.



V. Sec

```
definition
  removeOperation ::
  "entity_id ⇒ scap ⇒ scap ⇒
modify_sstate"
where
  "removeOperation e c c' s ≡
  if is_entity s (entity c) then
   s ((entity c) ↦ (direct_scaps s (entity c)) -
        {c'} )
  else
   s"
```

(* No notion of current thread *)

SEC Specification

- Abstraction of API to a graph of entities connected by capabilities
- Captures what operations are possible, not why operations happen.

- Connects to Take/Grant model from security literature.
- Weak model of security: no way to trust an entity.

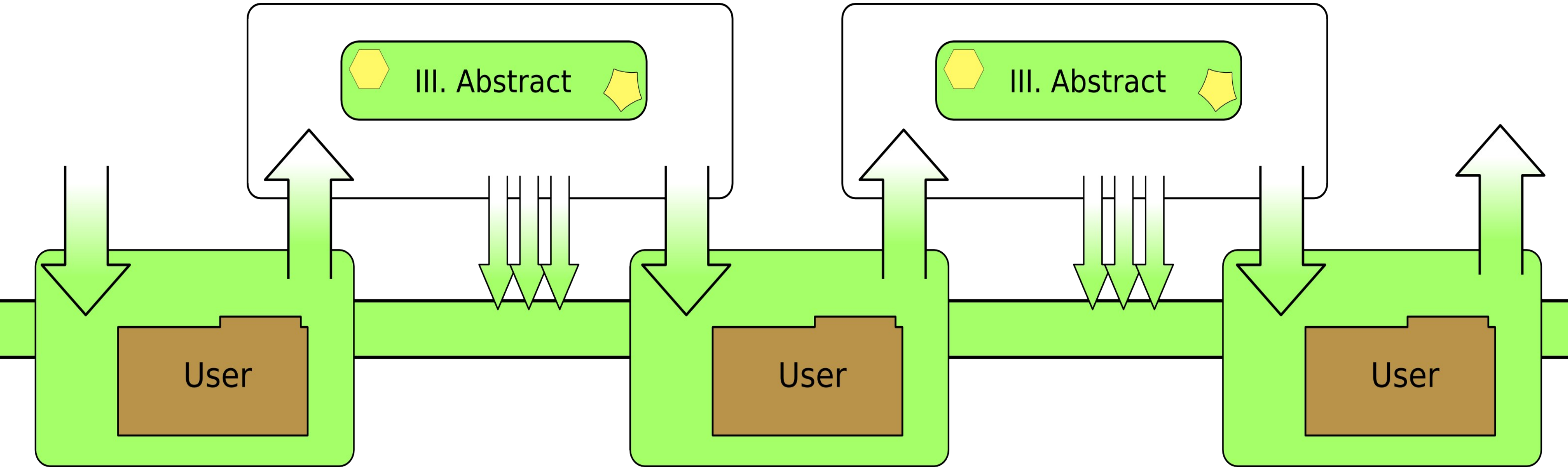
V. Sec

```
definition
  removeOperation ::
  "entity_id ⇒ scap ⇒ scap ⇒
modify_sstate"
where
  "removeOperation e c c' s ≡
  if is_entity s (entity c) then
   s ((entity c) ↦ (direct_scaps s (entity c)) -
        {c'} )
  else
   s"


(* No notion of current thread *)
```

User Behaviours

NICTA

- The system is composed sequentially with user execution and in parallel with hardware operation. The current model confuses these.
- The current model confuses the actions of different user tasks.

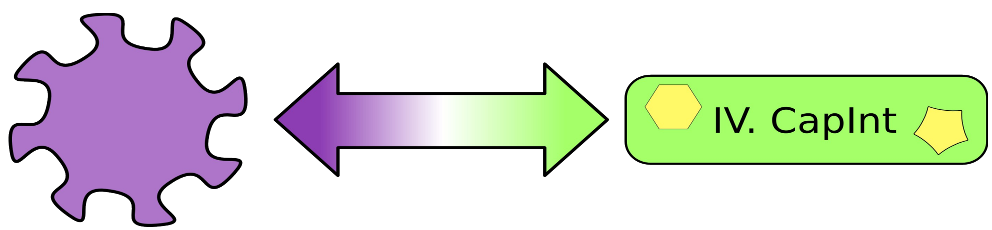


Cap+Intent Specification

- Began as a bootstrappping project
- Abstraction of API to capabilities, threads with known intentions, and other entities
- Work in progress

```
definition
  empty_slot :: "cdl_cap_ref ⇒ cdl_irq option
        ⇒ unit k_monad"
where
 "empty_slot slot free_irq ≡ TODO"

(* from definition of schedule *)
"do
     threads ← gets all_active_tcbs;
     next_thread ← option_select threads;
     return thread
 od"
```

- Together with a model change will provide a stronger execution model



Conclusion

- Verification of a microkernel is possible using straightforward approaches
- Decomposition is crucial, but there are still hard problems
- There is always more work to be done