A Data Layout Description Language for Cogent

Zilin Chen [1] [2]
Matthew Di Meglio [2]
Liam O'Connor [2]
Partha Susarla [1]
Christine Rizkallah [2]
Gabriele Keller [3]

[1] Data61, CSIRO
Sydney Australia

[2] UNSW
Sydney Australia

[3] Utrecht University
The Netherlands

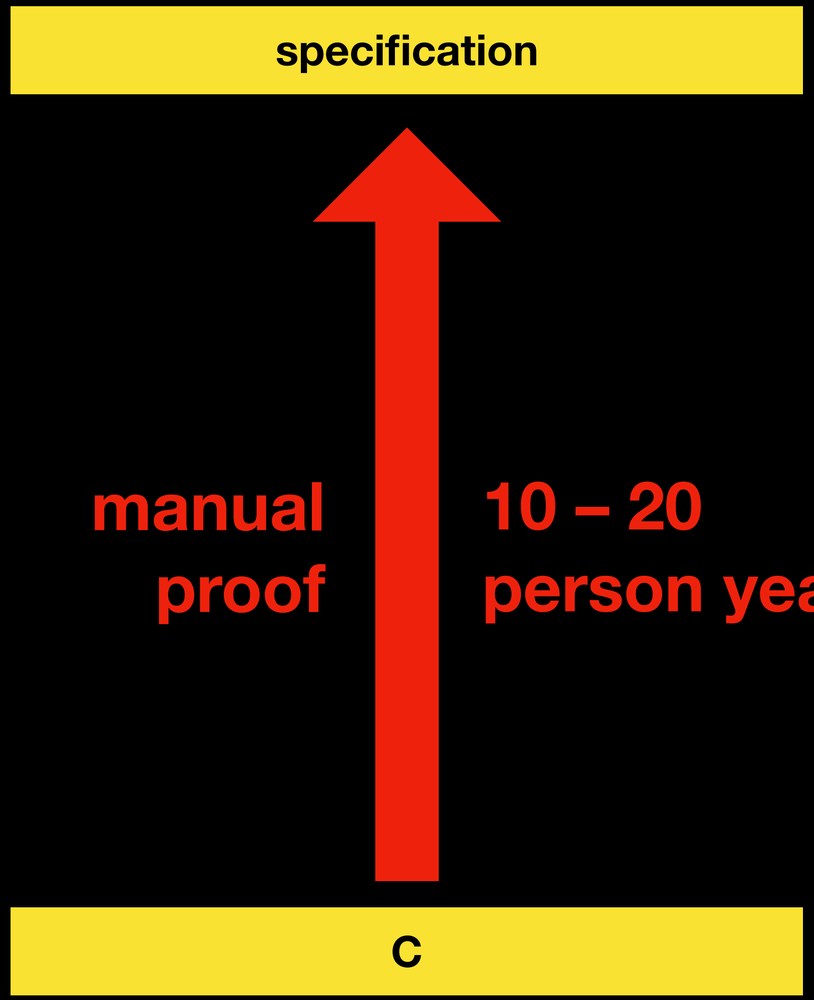
Utrecht University

Cogent is a **programming language** and **certifying compiler** for building verified low-level systems

verified

proofs in
Isabelle HOL

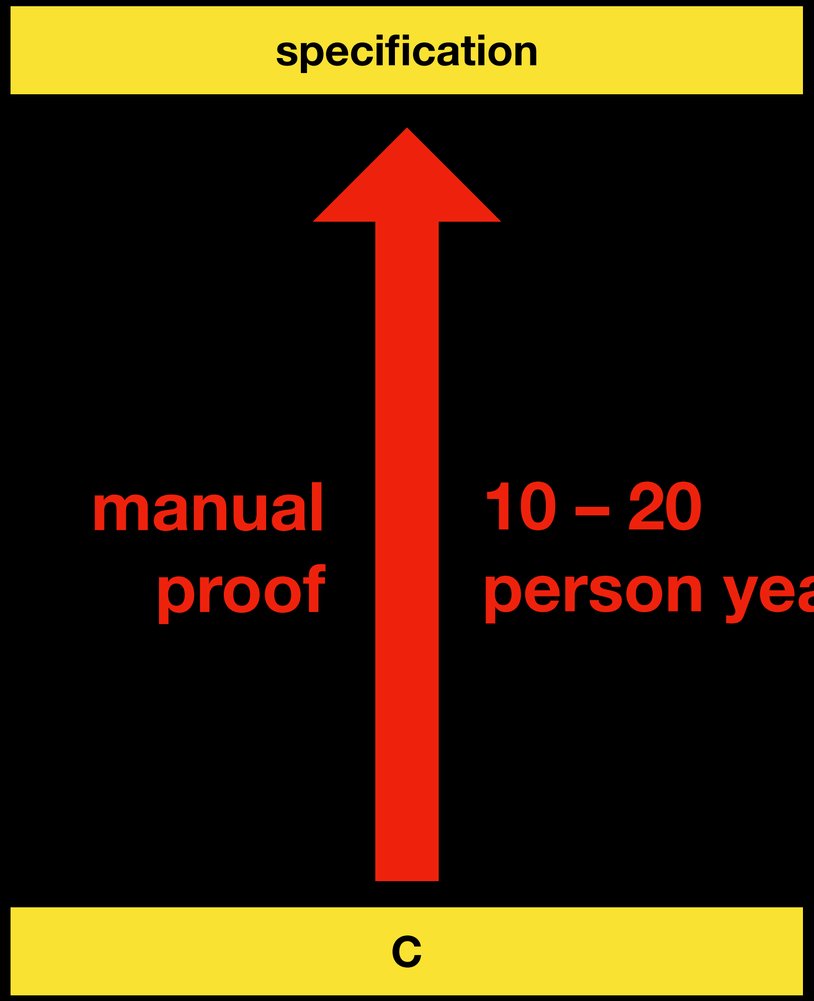
seL4



verified

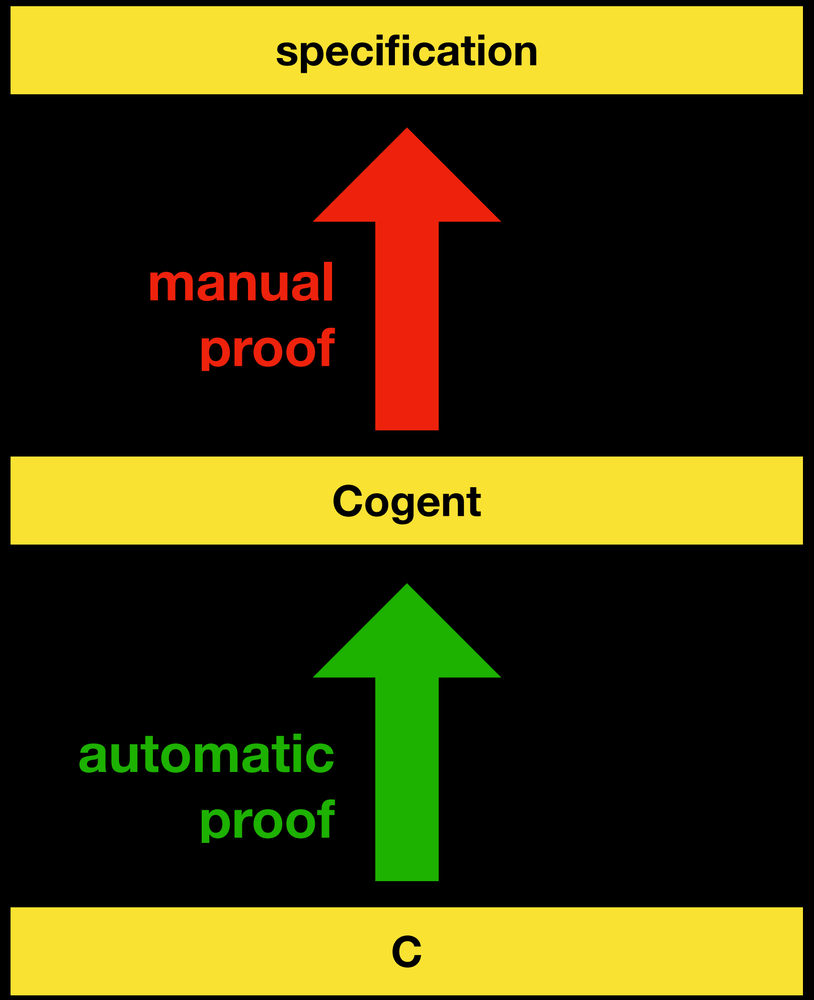
proofs in
Isabelle HOL

seL4



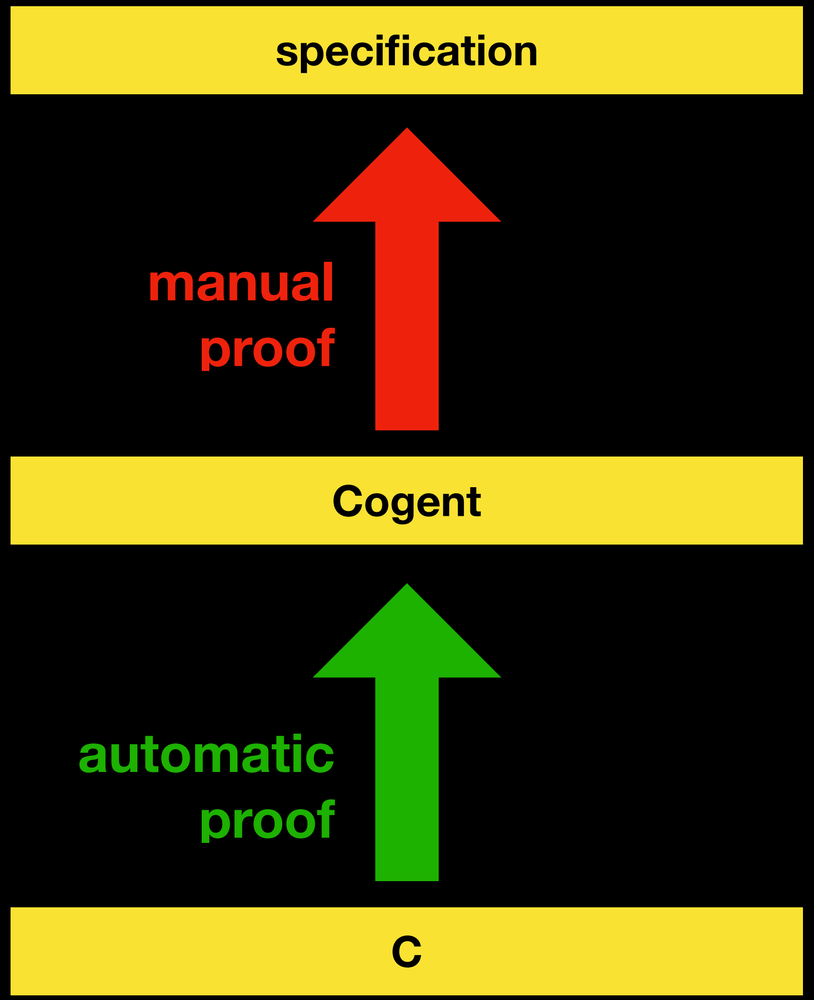
verified

proofs in
Isabelle HOL



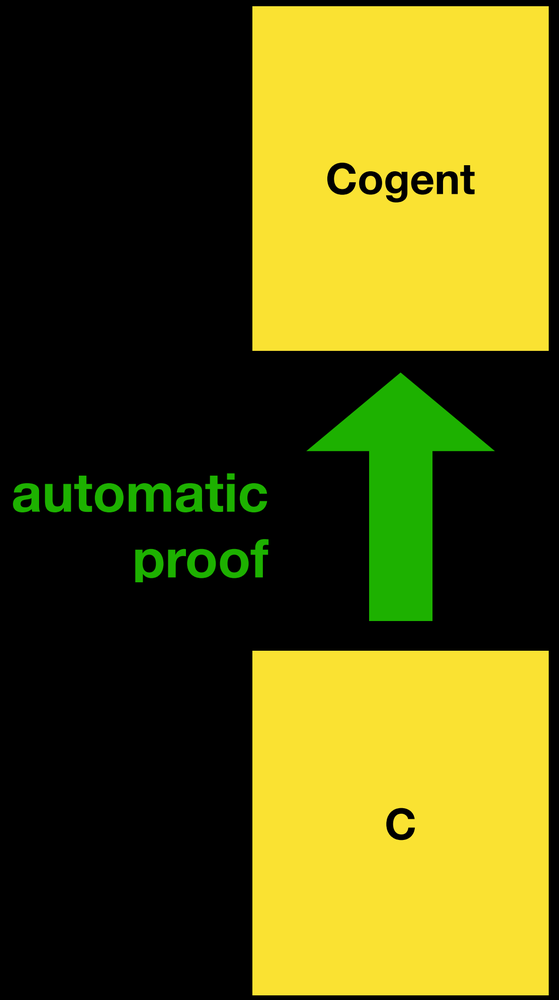
verified

proofs in
Isabelle HOL



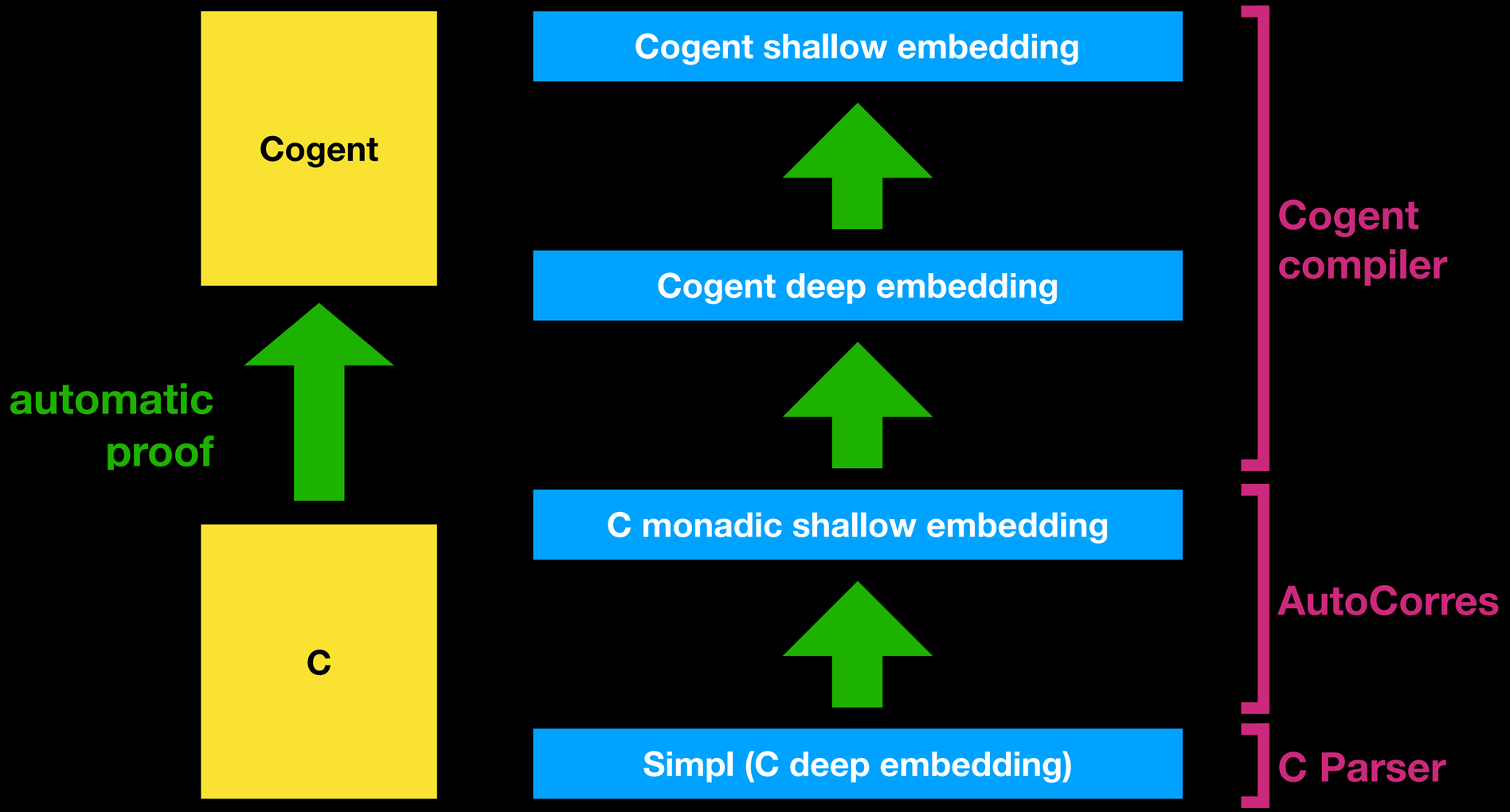
verified

proofs in
Isabelle HOL



verified

proofs in Isabelle HOL



verified

- purely functional
- uniqueness types
- type safe (implies memory safe)

Cogent is a programming language and certifying compiler for building verified low-level systems

low-level systems

- C Foreign Function Interface
- no garbage collection
- destructive updates

Cogent is a programming language and certifying compiler
for building verified low-level systems

primitive types

```
 U8 typedef unsigned char        u8;
U16 typedef unsigned short      u16;
U32 typedef unsigned int        u32;
U64 typedef unsigned long long u64;
 () typedef struct unit_t { int   dummy; } unit_t;
Bool typedef struct bool_t { u8  boolean; } bool_t;
```

record types

```
#{f1: U32, f2: Bool} typedef struct t1 {
                        u32 f1;
                        bool_t f2;
                     } t1;

{f1: U32, f2: Bool} typedef struct t1 * t2;
```

variant types

```
<Success U32|Failure ()> enum tag_t {
                             TAG_ENUM_Success,
                             TAG_ENUM_Failure
                         };

                         typedef struct t3 {
                             tag_t tag;
                             u32 Success;
                             unit_t Failure;
                         } t3;
```

C foreign function interface

	efficient	minimises C boilerplate	integrates with existing C code
cogent types only	✔	✔	✗
abstract types			

abstract types

abstract types

list.cogent

```
type ListNode a


type List' a
```

list.ah

```
typedef struct $id:(ListNode a) {
    struct $id:(ListNode a) * next;
    $ty:a val;
} $id:(ListNode a);

typedef struct $id:(List' a) {
    $ty:a head;
} $id:(List' a);
```

abstract types

list.cogent

```
listnode_get: all(a). (ListNode a)! -> a
```

list.ac

```
$ty:a $id:listnode_get($ty:((ListNode a)!) node) {
  return node->val;
}
```

abstract types

C foreign function interface

	efficient	minimises C boilerplate	integrates with existing C code
cogent types only	✔	✔	✗
abstract types	✔	✗	●
conversion by copy	✗	✗	✔
conversion by cast	✔	✔	✔

binary data

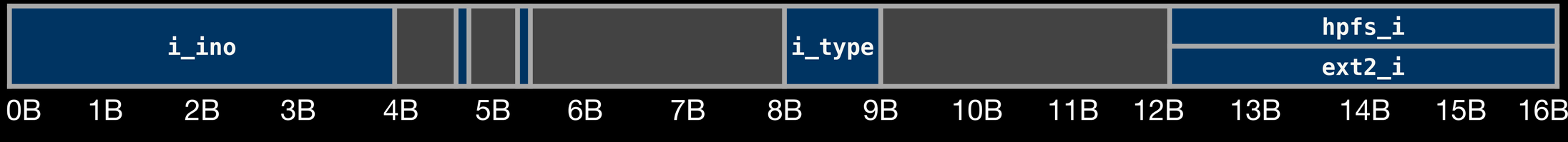
- bit masks and bitwise operations
- **memory layout aware types**

Dargent

attach memory layouts to Cogent types

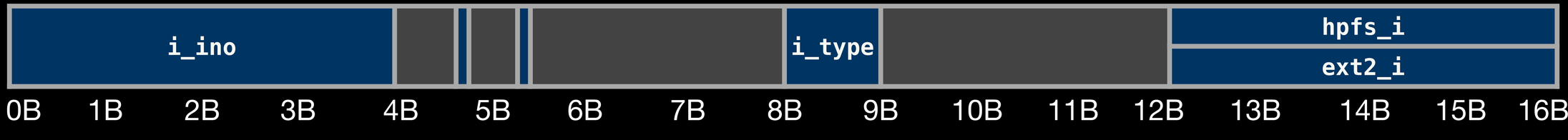
```c
struct inode {
  unsigned int i_ino;
  unsigned int i_state;
  unsigned char i_type;
  union {
    struct hpfs_inode_info hpfs_i;
    struct ext2_inode_info ext2_i;
  } u;
};

// i_state bitmap masks
#define I_FREEING  (1<<5)
#define I_LINKABLE (1<<10)

// i_type enum values
#define I_HPFS_TYPE 0
#define I_EXT2_TYPE 1
```

```
type Inode =
  { i_ino:      U32
  , i_freeing:  Bool
  , i_linkable: Bool
  , u:
    < Ext2I Ext2InodeInfo
    | HpfsI HpfsInodeInfo
    >
  } with layout
  { i_ino:      4B
  , i_freeing:  1b at 4B+5b
  , i_linkable: 1b at 4B+10b
  , u:          1B at 8B
    | 0 -> Ext2I Ext2InodeL at 12B
    | 1 -> HpfsI HpfsInodeL at 12B
  }
```



```
type Inode =
  { i_ino:       U32
  , i_freeing:   Bool
  , i_linkable: Bool
  , u:
    < Ext2I Ext2InodeInfo
    | HpfsI HpfsInodeInfo
    >
  } with layout
  { i_ino:       4B
  , i_freeing:   1b at 4B+5b
  , i_linkable: 1b at 4B+10b
  , u:           1B at 8B
    | 0 -> Ext2I Ext2InodeL at 12B
    | 1 -> HpfsI HpfsInodeL at 12B
  }
```

```
typedef struct t1 {
  unsigned int data[4U];
} *t1;
```



```c
typedef struct t1 {
  unsigned int data[4U];
} *t1;
```

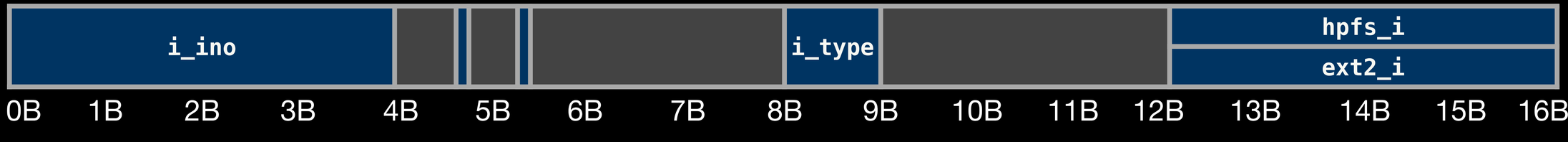
```c
static inline unsigned int
d4_get_i_linkable_part0(t1 b) {
  return b->data[1U] >> 10U & 1U;
}

static inline bool_t
d3_get_i_linkable(t1 b) {
  return (bool_t) {
    .boolean = (unsigned char)
d4_get_i_linkable_part0(b) << 0U
  };
}
```

```c
typedef struct t1 {
  unsigned int data[4U];
} *t1;
```

```c
static inline void
d6_set_i_linkable_part0(
    t1 b, unsigned int v) {
  b->data[1U]
    = (b->data[1U] & ~(1U << 10U))
    | (1U & v) << 10U;
}

static inline void
d5_set_i_linkable(t1 b, bool_t v){
  d6_set_i_linkable_part0(b,
    (unsigned int)
    (v.boolean >> 0U));
}
```

```
link: (Inode, Bool) -> Inode
link (inode {i_linkable=old}, new)
  = inode {i_linkable=new}
```

```
typedef struct t2 {
  t1      p1;
  bool_t p2;
} t2;

static inline t1 link(t2 a1) {
  t1 r2 = a1.p1;
  bool_t r3 = a1.p2;
  bool_t r4 =
      d3_get_i_linkable(r2);
  t1 r5 = r2;
  d5_set_i_linkable(r5, r3);
  t1 r6 = r5;
  return r6;
}
```



Dargent

- layouts only affect types
- conversion by cast
- native bitmaps

plan

- ☑ initial language design
- ☑ compiler data structures
- ☑ code generation
- ☑ parser, type checker and desugarer
- ☐ verification

verification

- update refinement proofs
- relation between getters/setters and layouts

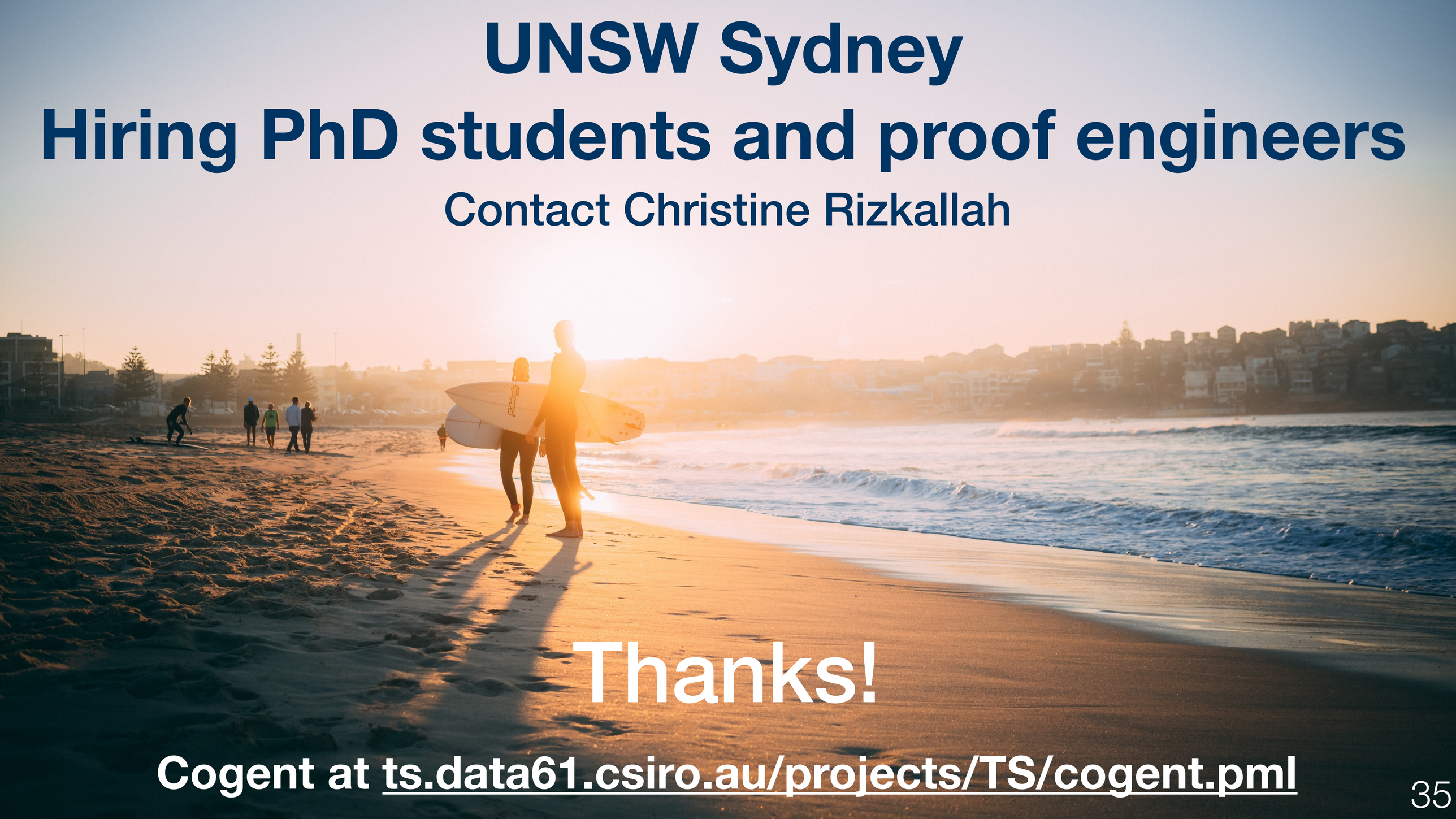
future work

- layout polymorphism and inference
- endianness
- nested pattern matching

Sydney

UNSW Sydney
Hiring PhD students and proof engineers

Contact Christine Rizkallah



Thanks!

Cogent at ts.data61.csiro.au/projects/TS/cogent.pml