Code Optimizations Using Formally Verified Properties

Yao Shi, Bernard Blackham, Gernot Heiser
NICTA and University of New South Wales

October 2013

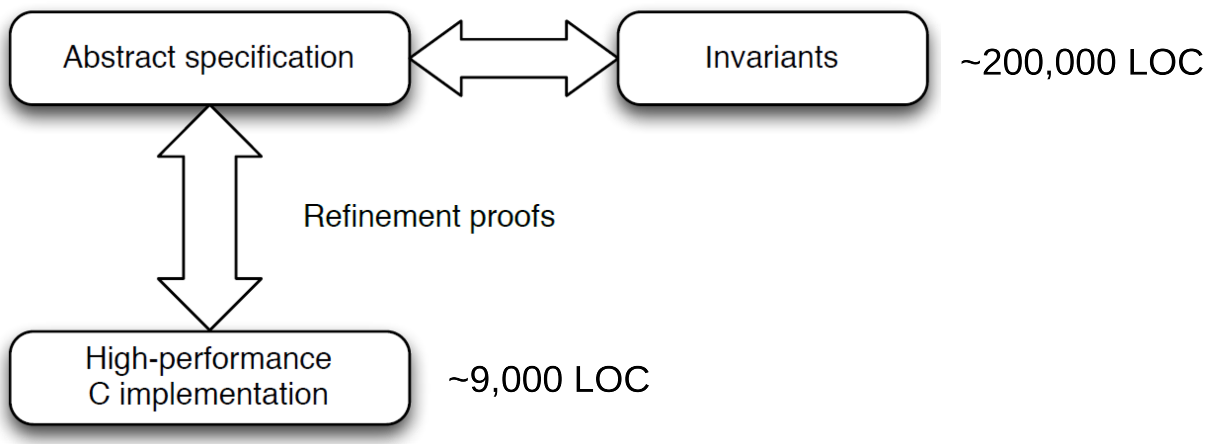
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Motivation

- seL4
  - Formally verified micro-kernel
    - Formal specification of functional behavior
  - Bug-free
  - Micro-kernel in C language ~9,000 LOC
  - Formal specification ~200,000 LOC

    [Klein et al, seL4: Formal verification of an OS kernel. SOSP'09]



Motivation

- Formal specification
  - Designed for proof
  - More human effort
  - More information
  - Code optimizations
  - …

Case Study I – Use of Invariants

NICTA

```
void deleteCallerCap (tcb_t *receiver) {
    callerSlot = TCB_PTR_CTE_PTR(receiver, 3);
    deleteCap(callerSlot);
}
```

```
void deleteCap(slot_t *slot) {
    …
    ret = finalizeCap(slot->cap);
    …
}
```

```
int finalizeCap (cap_t cap …) {
    switch (getCapType(cap)) {
        case cap_reply_cap:
        case cap_null_cap:
            fc_ret.remainder = cap_null_cap_new();
            fc_ret.irq = irqInvalid;
            return fc_ret;
        case cap_endpoint_cap:
            …
        case cap_async_endpoint_cap:
            …
    }
    …
}
```

tcb_cnode_index 3 ↦
(λ _ thread_state .
case thread_state of
BlockedOnReceive e d → (op ≡ **NullCap**)
| _ → (op ∈ {**ReplyCap**, **NullCap**}))

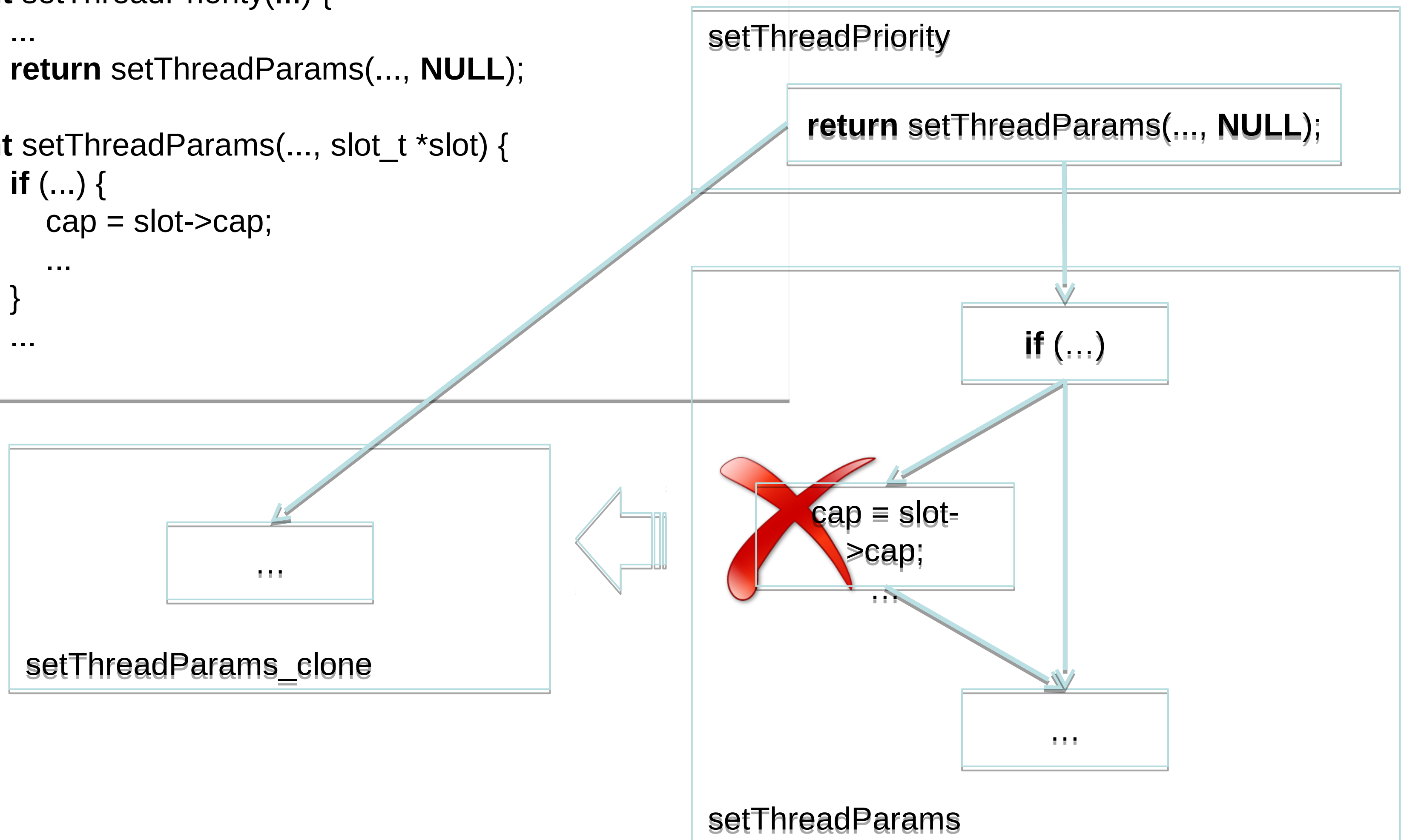
TCB_PTR_CTE_PTR(receiver,3)->cap
is cap_reply_cap (ReplyCap)
or cap_null_cap (NullCap)

```
int finalizeCap_clone (cap_t cap …) {
    fc_ret.remainder = cap_null_cap_new();
    fc_ret.irq = irqInvalid;
    return fc_ret;
}
```

Case Study II – Use of Program Safety

NICTA

```
int setThreadPriority(...) {
    ...
    return setThreadParams(..., NULL);
}
int setThreadParams(..., slot_t *slot) {
    if (...) {
        cap = slot->cap;
        ...
    }
    ...
}
```

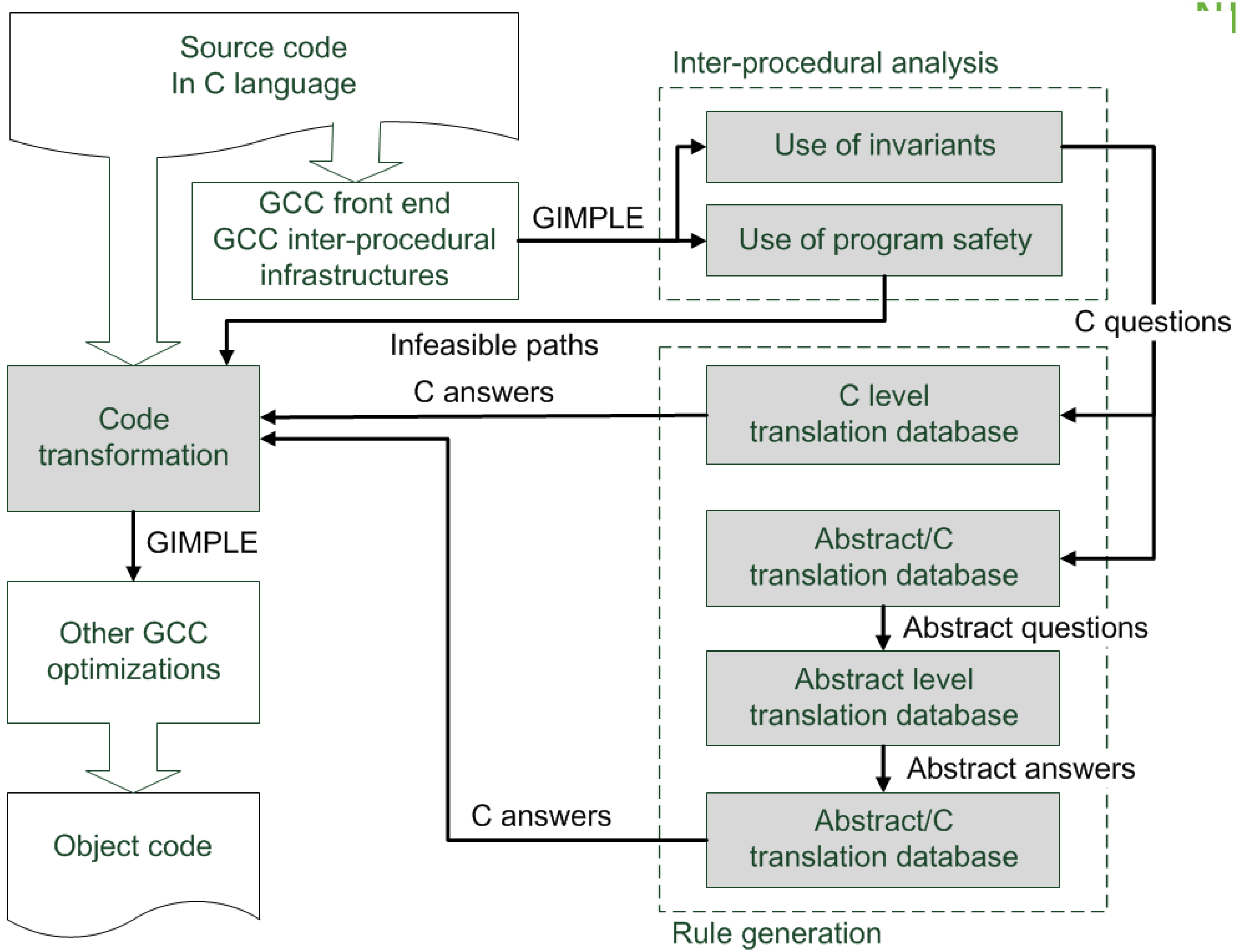


Overview

- Use of invariants
  - Hard to implement invariants at C level
  - Demand-driven approach
- Use of program safety
  - Some compilers have implemented the similar function but cause more serious bugs or security vulnerabilities
    - [Wang et al. Undefined behavior: what happened to my code? Apsys'12]
- Optimizations
  - Context separation
  - Code transformation according to the results of use of invariants and use of program safety

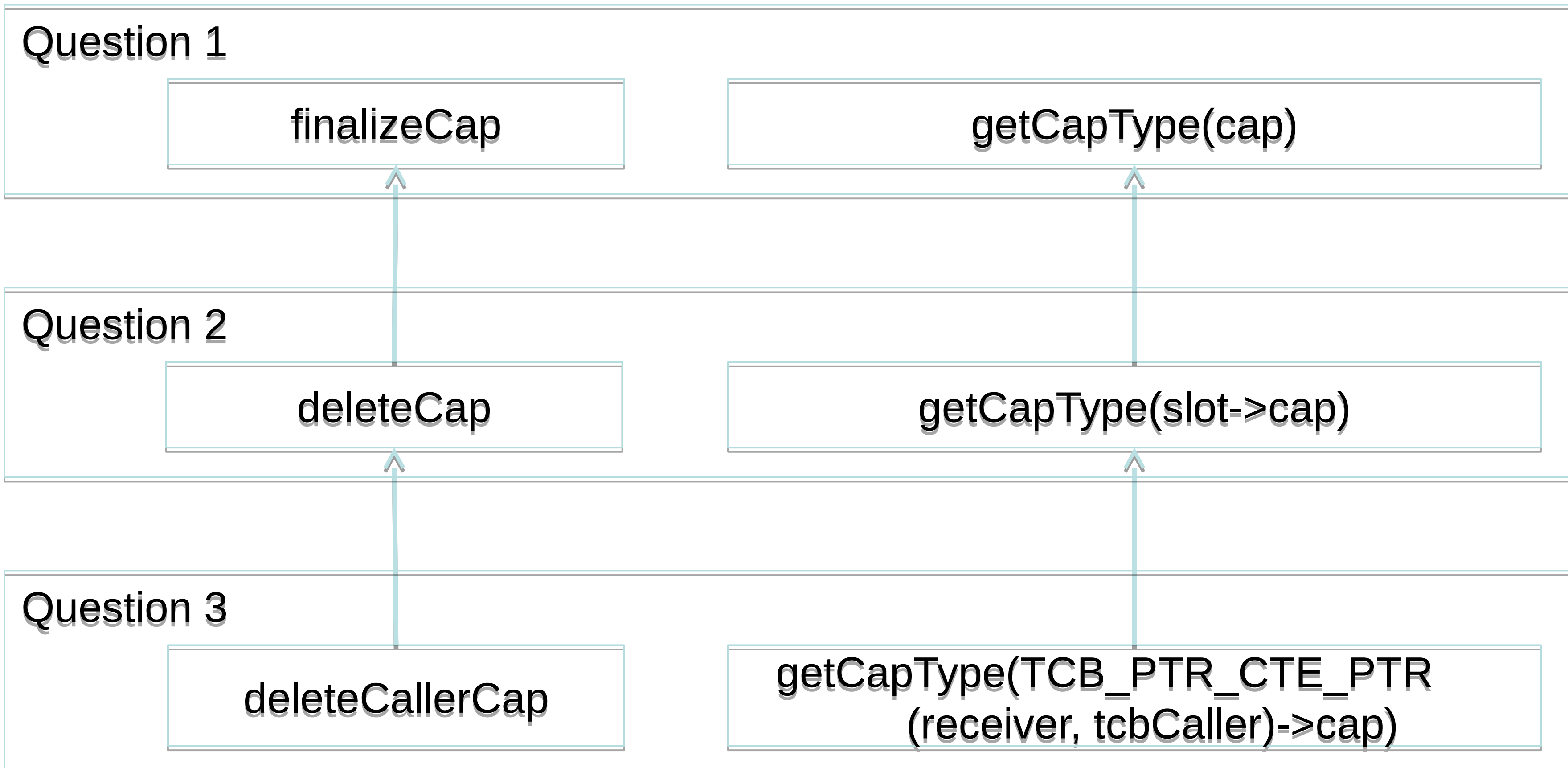
NICTA Copyright 2012

Overview



NICTA Copyright 2012 From imagination to impact

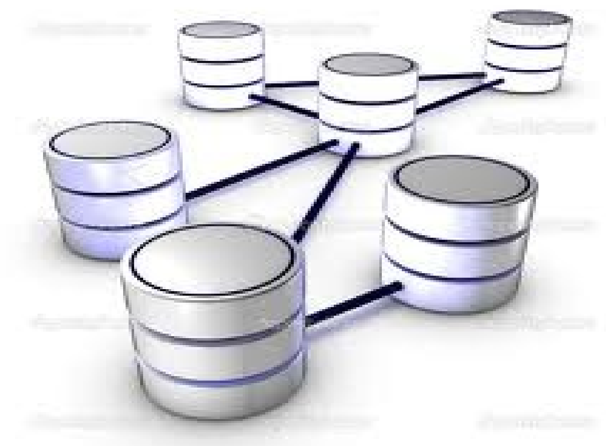
Approach I: Use of Invariants



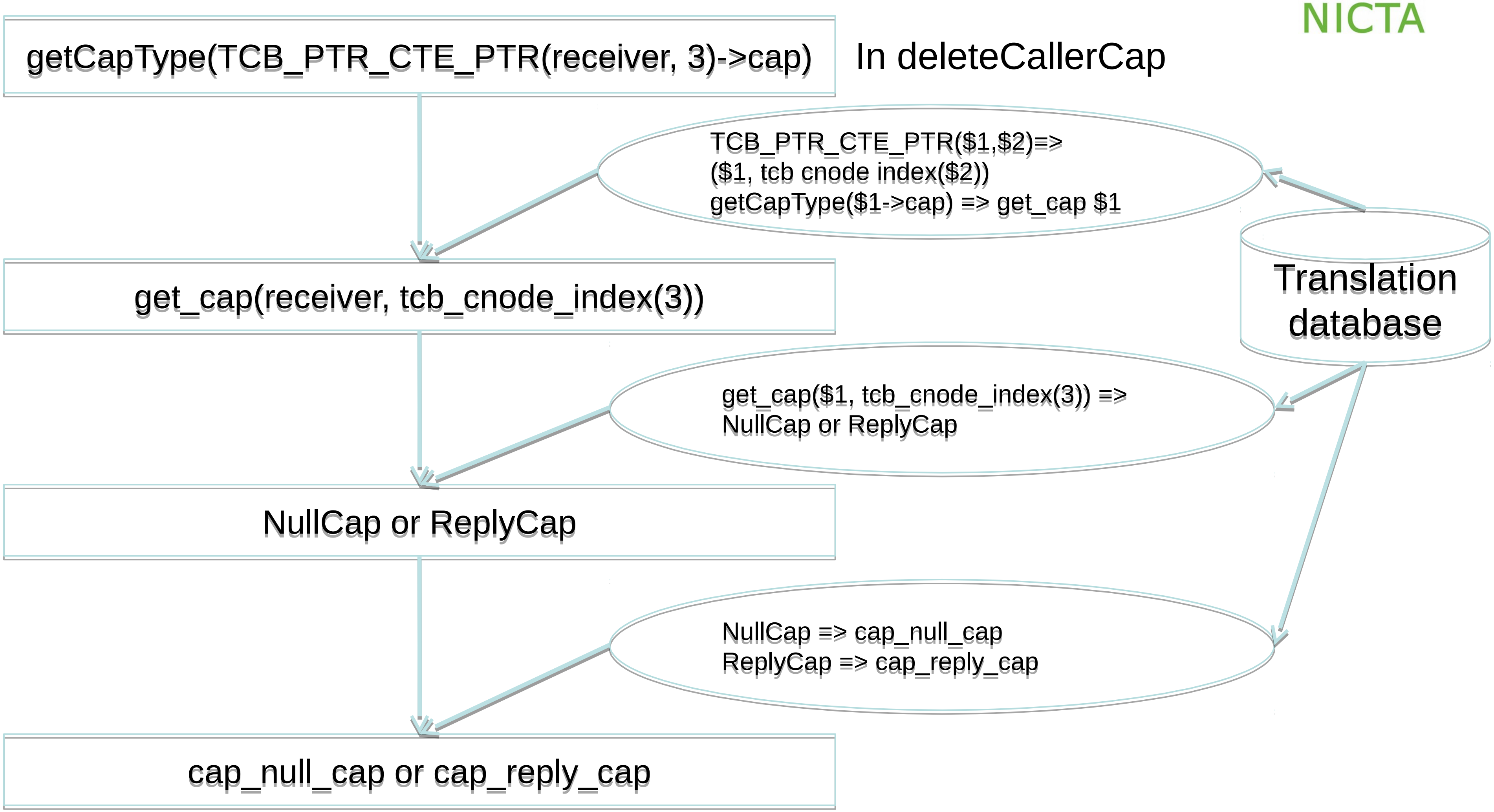
Approach I: Use of Invariants

NICTA

- Accumulated translation database
  - C=>C
  - C=>abstract, abstract=>abstract, abstract=>C
- Input by manual
  - 125 rules / 1540 answers
  - Many questions are similar with different contexts



Approach I: Use of Invariants

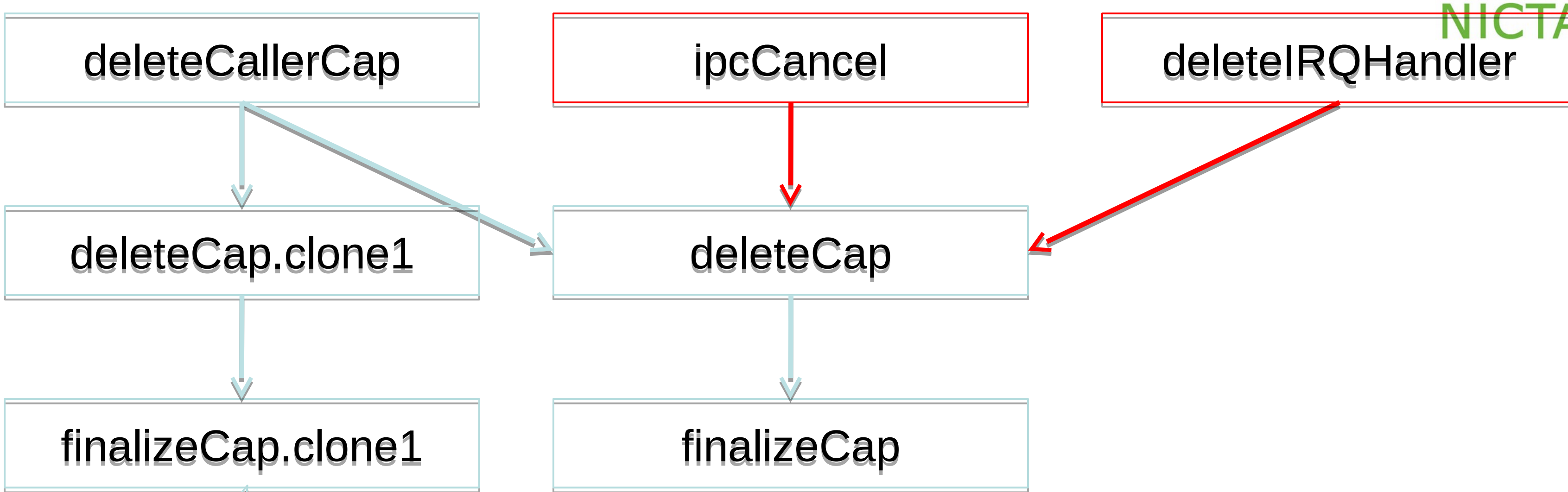


getCapType(cap) in finalizeCap is indirectly answered!

Approach II: Use of Program Safety

- Find a static “buggy” path
- This “buggy” path must be infeasible due to the bug-free program
- Pass the the infeasible path to the next stage for further optimizations
- Null-pointer dereference, uninitialized memory read, memory leak…

Optimizations




```
int finalizeCap_clone1 (cap_t cap …) {
    fc_ret.remainder = cap_null_cap_new();
    fc_ret.irq = irqInvalid;
    return fc_ret;
}
```

```
int finalizeCap (cap_t cap …) {
    switch (getCapType(cap)) {
        case cap_reply_cap:
        case cap_null_cap:
            fc_ret.remainder =
cap_null_cap_new();
            fc_ret.irq = irqInvalid;
            return fc_ret;
        case cap_endpoint_cap:
            …
        case cap_async_endpoint_cap:
            …
    }
    …
}
```

Evaluation

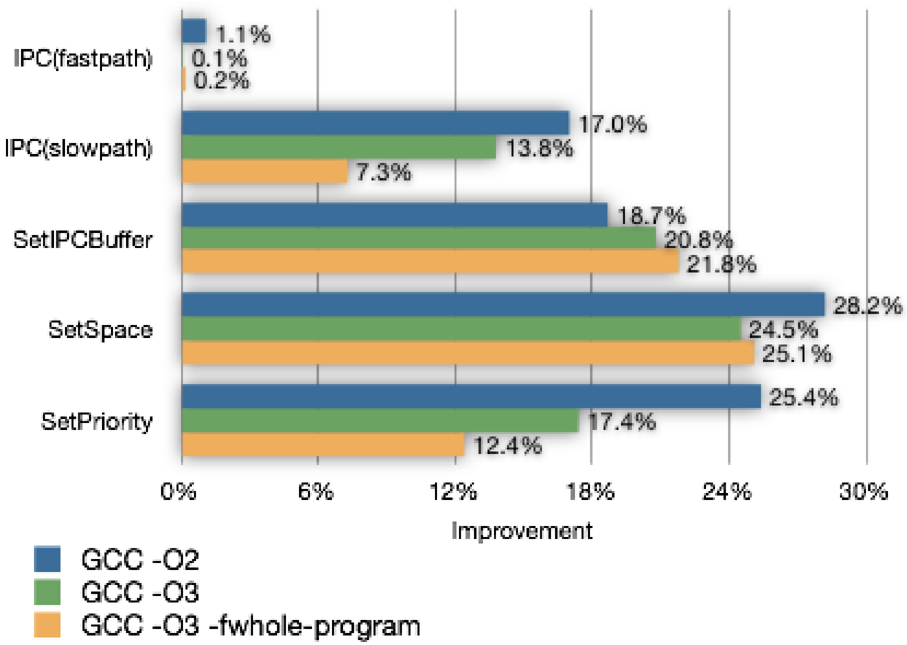
NICTA

- GCC 4.5.2
  - Cross compiler for ARM on X86
- Target
  - BeagleBoard-xM with TI DM3730
  - 1GHz ARM Cortex-A8
  - 32KB 4-way set-assoc. L1 instruction cache
  - 32KB 4-way set-assoc. L1 data cache
  - 128KB unified L2 cache
- Compilation
  - 2.66GHz Intel Xeon with 10GB memory

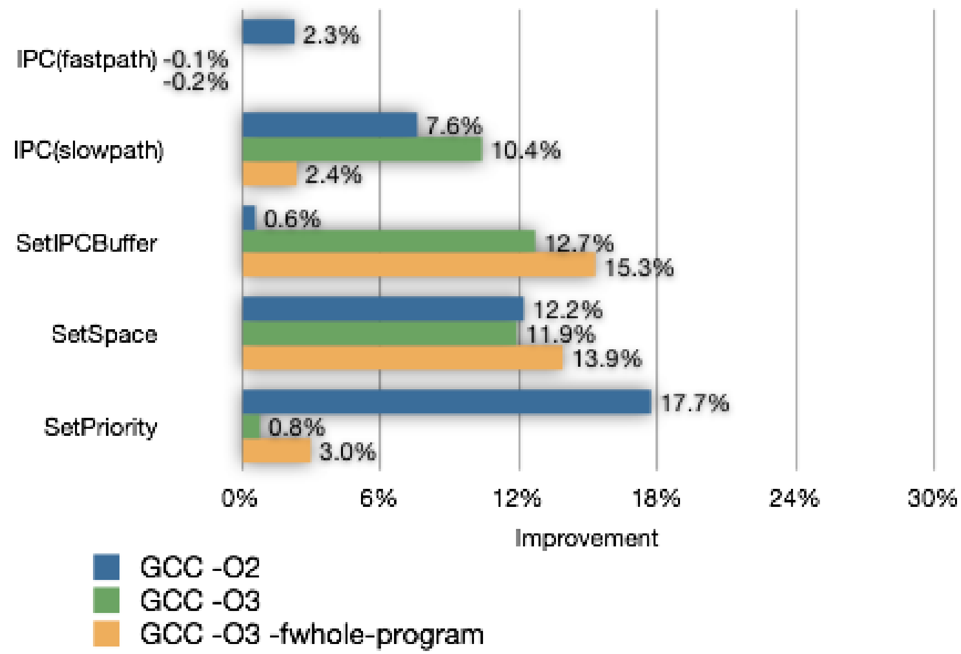
Evaluation

- Summary
  - Line of code: 9,525
  - Functions: 276
  - Total contexts: 217,439
  - C questions: 10,223
  - C answers: 1,540
  - Rules: 125
  - Separated contexts: 4,671

Evaluation



Relative performance of seL4 micro-benchmarks

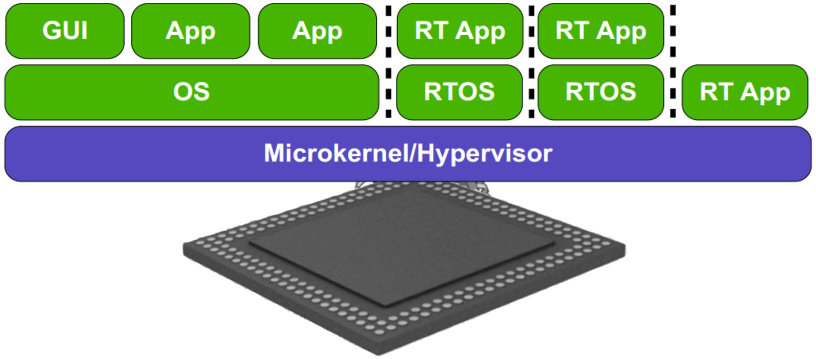


Relative performance of seL4 micro-benchmarks without invariant-based optimizations

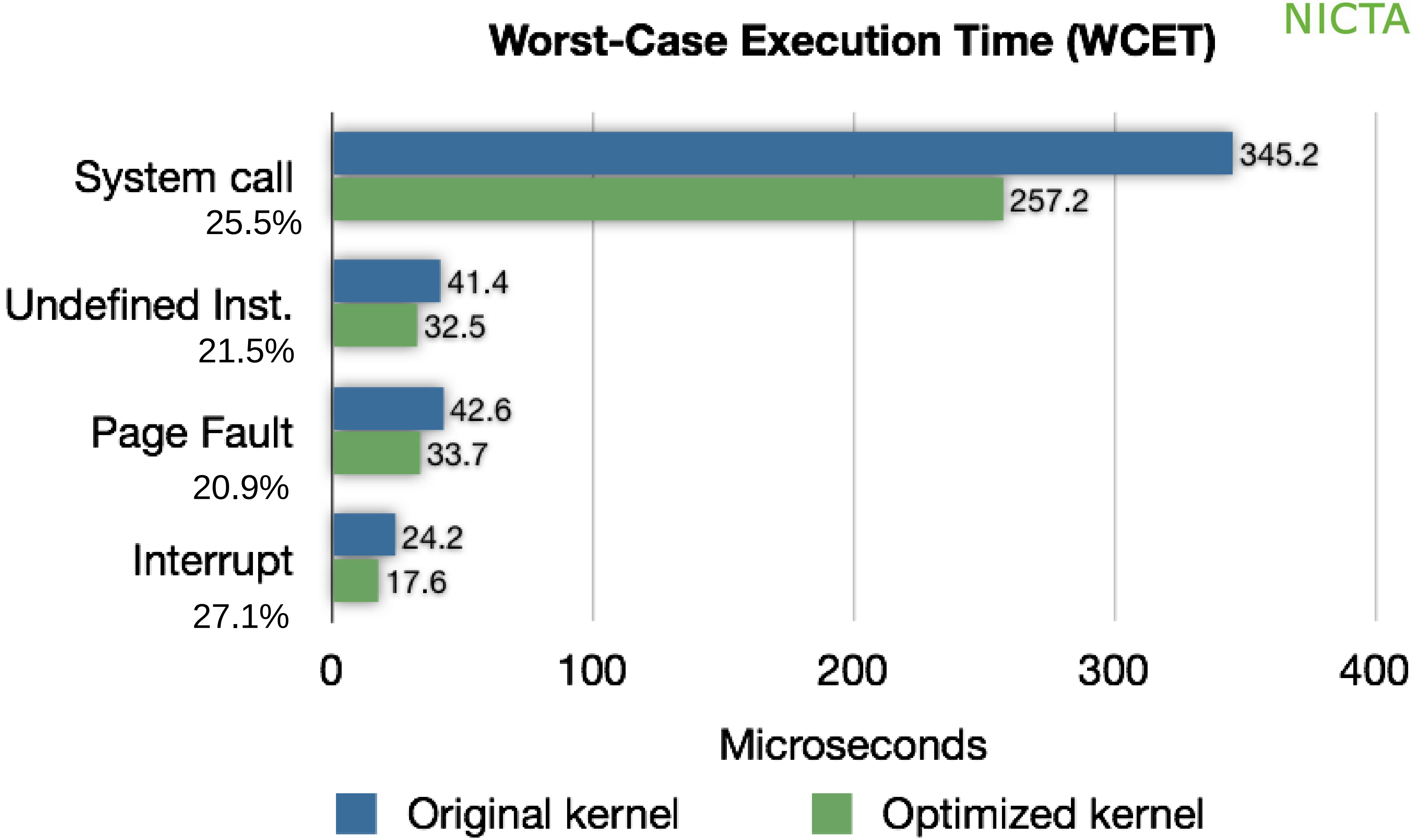
NICTA Copyright 2012

Evaluation

- Virtualized Linux based on seL4
- Performance improvement
  - LMBench 3.0
    - Linux operations: up to 20.8%
    - Context switch: up to 21.2%
    - Communication latency: up to 11.0%
    - Filesystem operations: up to 21.1%
    - Insignificant for bandwidth throughput items
  - tar
    - Single file: 6.4%, batch of files and dirs: 16.0%
  - cp
    - Single file: 6.4%, batch of files and dirs: 12.9%

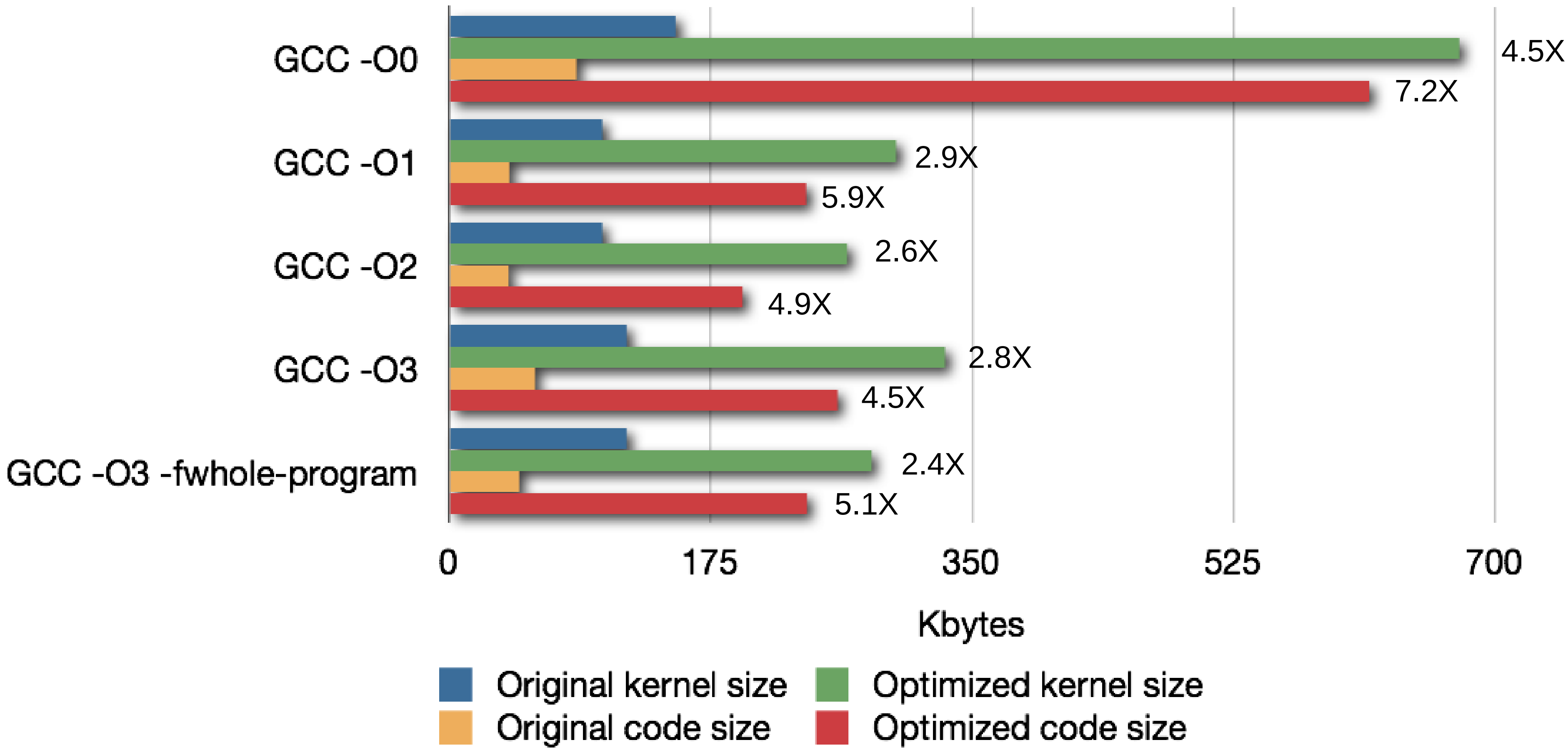


Evaluation



[Blackham et al. Timing analysis of a protected operating system kernel. RTSS'11]

Evaluation



GCC -O0
GCC -O1
GCC -O2
GCC -O3
GCC -O3 -fwhole-program
4.5X
7.2X
2.9X
5.9X
2.6X
4.9X
2.8X
4.5X
2.4X
5.1X
0
175
350
525
700
Kbytes
Original kernel size
Optimized kernel size
Original code size
Optimized code size

Conclusion/Contribution

- First implementation for code optimizations using formal verification
- Optimized a practical micro-kernel, seL4
  - Runtime performance improved by up to 28%
  - Real-world applications on virtualized Linux: 6-16%
  - Reduced worst-case execution time by 25% (IPC)

Thank You!

Q & A