NICTA

Towards High-Assurance Multiprocessor Virtualisation

Michael von Tessin

PhD Student
NICTA and University of New South Wales
Sydney, Australia

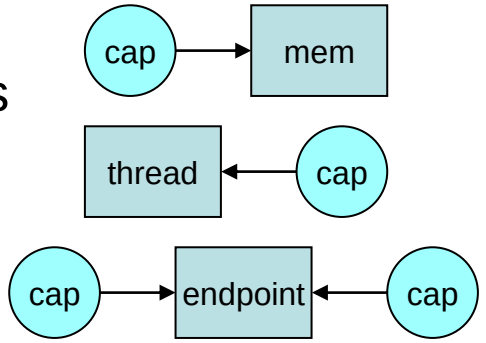
Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Introduction

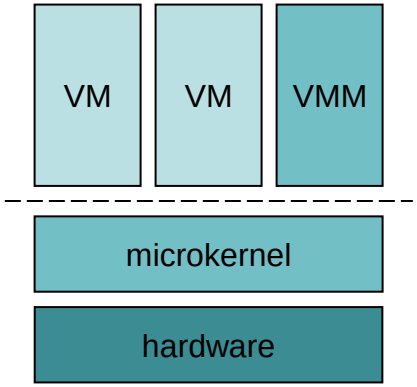
NICTA

- **seL4** (secure embedded L4) microkernel:
  - provides strong isolation between components
  - allows fine-grained controlled communication and resource management via capabilities
  - C implementation is formally verified
  - “verified” = refinement proof between an abstract specification of seL4 and the C implementation (L4.verified, ~25py)



L4.verified

- main area of application: **virtualisation**
  - microkernel is used as a hypervisor
    - resource isolation
  - virtual machine monitor (VMM)
    - resource management
    - runs deprivileged, on top of the microkernel
    - runs/manages virtual machines (VMs)



Introduction

- concurrency was not in scope of L4.verified:
  1. able to avoid preemption-induced concurrency:
     - no preemption in kernel
     - except from a few well-defined preemption points
     - state saved as continuation instead of doing a stack switch
  2. able to avoid hardware concurrency:
     - device drivers outside the kernel (standard microkernel approach)
     - **only support uniprocessor systems**

- whole world is going **multicore** (even in embedded systems)

→ want to have a multiprocessor version of seL4 with the same strong **isolation** guarantees

Problem?

Introduction

- concurrency verification complexity increases combinatorially

- model checking:
  - uses explicitly enumerated states

  → *state explosion*
- theorem proving:
  - *non-determinism* defines which instruction is executed next
  - proofs have to cover every possible non-deterministic choice

  → "*proof explosion*"

- mitigation techniques:
  - make proofs modular (rely-guarantee, VCC's ownership principle)
  - make the system modular → componentise it
  - components use each others' APIs (with component-local state)

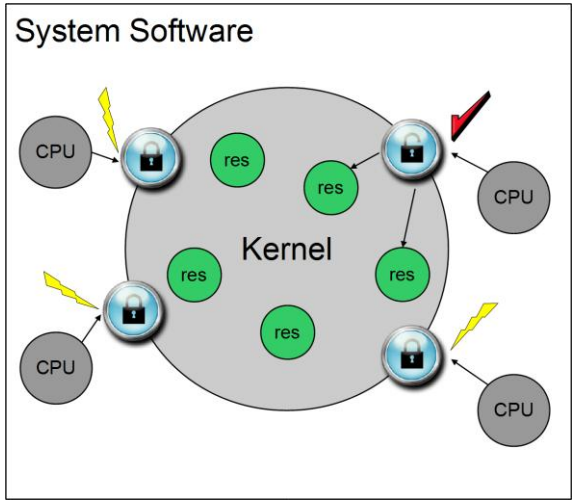
  → can we componentise seL4?

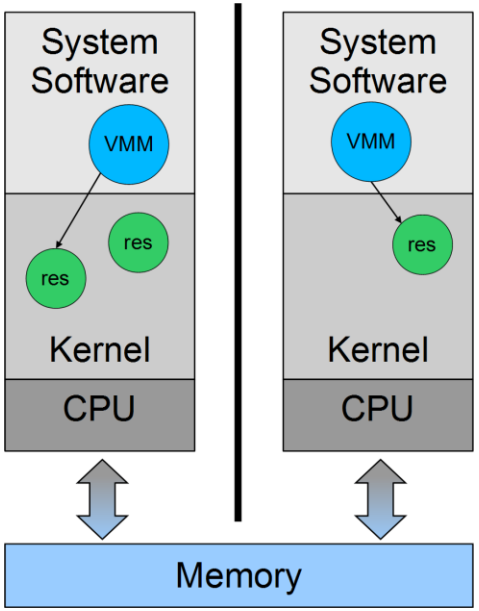
Multiprocessor Kernel Designs

NICTA

There are two fundamental ways to avoid concurrency:

- avoid parallelism (run things sequentially):
  - solution: ***big lock*** around the whole kernel
  - existing system software can be run unmodified and benefit from multiple CPUs
  - low scalability
  - still have to deal with TLB invalidation etc.
- avoid sharing (partition the global state):
  - solution: ***multikernel*** approach (like in Barrelfish)
  - run one *instance* of uniprocessor seL4 per CPU
  - resources (memory, devices) are statically partitioned between seL4 instances
  - perfect scalability
  - no interaction possible between instances



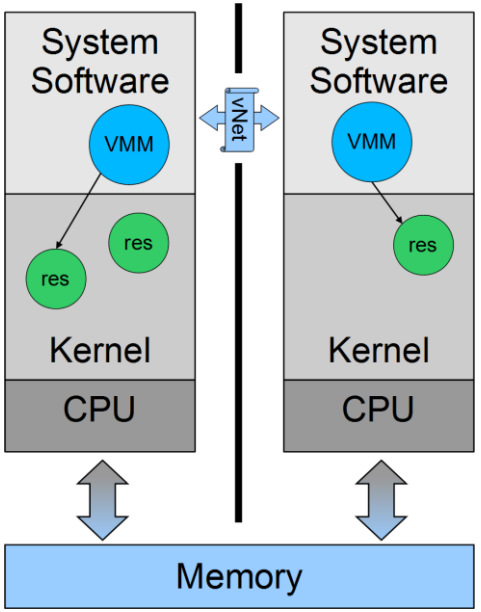


Multikernel Design

NICTA

- how to enable communication?
  - designate region of *shared memory* (shared between seL4 instances)
  - kernel provides it to the system software running on top (e.g. VMM)
  - VMMs are able to implement communication mechanisms between instances (e.g. a virtual network)
  - kernel never accesses this shared memory

  → does not introduce concurrency into kernel



Verification

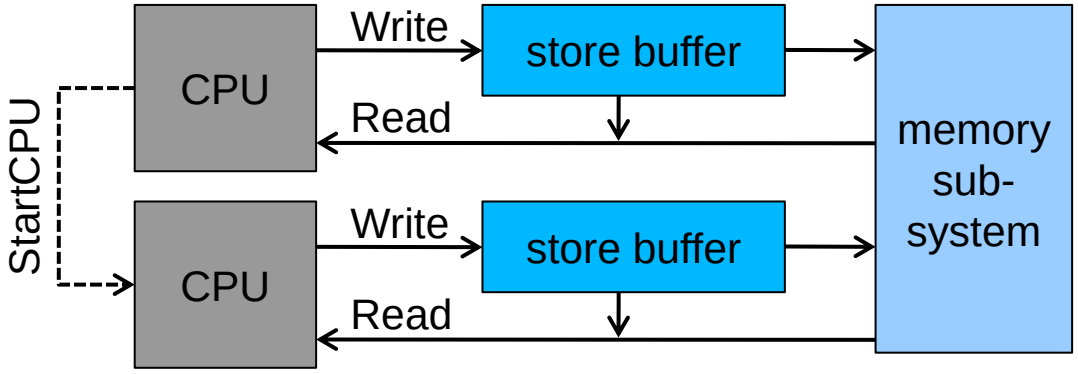
NICTA

- goals:
  - lift L4.verified proof into a multiprocessor context
  - proof some important isolation/correctness properties

- milestones:
  - multiprocessor execution model
  - chose and implement a design: multikernel
  - specification of the new multikernel-specific code
  - formal connection to the L4.verified models/proofs

Multiprocessor Execution Model

- requirements:
  - model weak memory ordering and fences
    (version in paper only models sequentially consistent memory)
  - model CPUs starting up other CPUs
  - integratable into L4.verified verification framework
- result:
  - operational model inspired by the Cambridge x86-TSO model
  - written in Isabelle/HOL
  - 4 *high-level instructions*: Read, Write, MFENCE, StartCPU

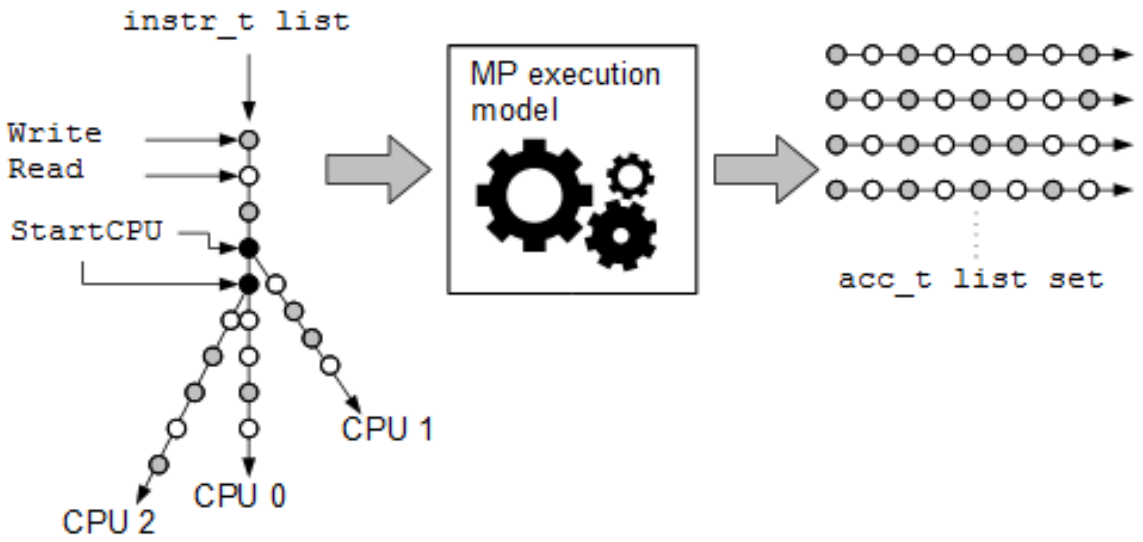


R↔R	×
R↔W	×
W↔R	✓
W↔W	×

Multiprocessor Execution Model

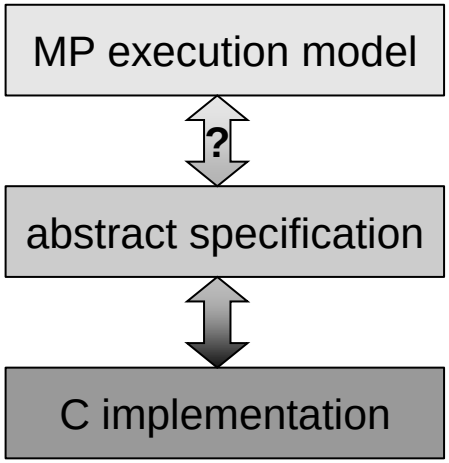
```
datatype instr_t
    Read      "paddr_t set"
  | Write     "paddr_t set"
  | MFENCE
  | StartCPU cpu_t "instr_t list"
```

```
datatype acc_t
    Read  "paddr_t set" cpu_t
  | Write "paddr_t set" cpu_t
```



Modelling seL4's MP-specific Code

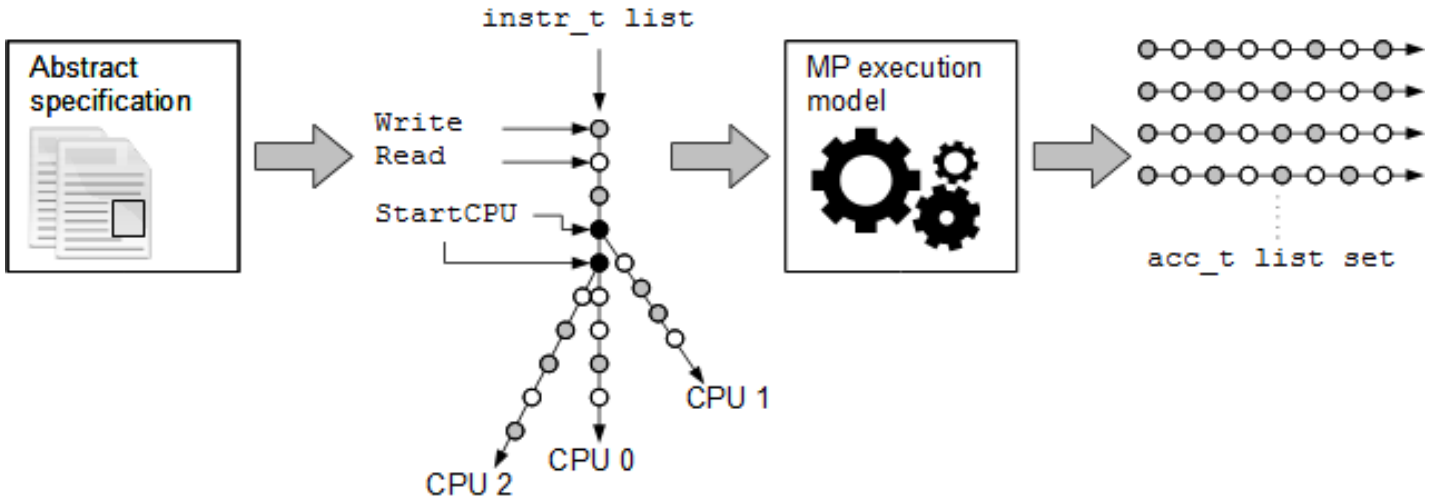
- abstract specification:
  - Haskell-like monadic style
  - shallowly embedded into Isabelle/HOL
  - one corresponding function for each C function
  - use same framework (VCG, libs) as for L4.verified
    - only allows us to reason about sequential programs
    - i.e. we read the same value from memory that we have read/written before
    - how can we connect this framework to the multiprocessor execution model?



Modelling seL4's MP-specific Code

NICTA

1. want to assume CPU-local sequential semantics (proofs over the abstract specification rely on it)
2. map abstract operations to MP execution model (monadic "execution" creates high-level-instruction list)
3. use MP execution model to prove that sequential semantics are observed



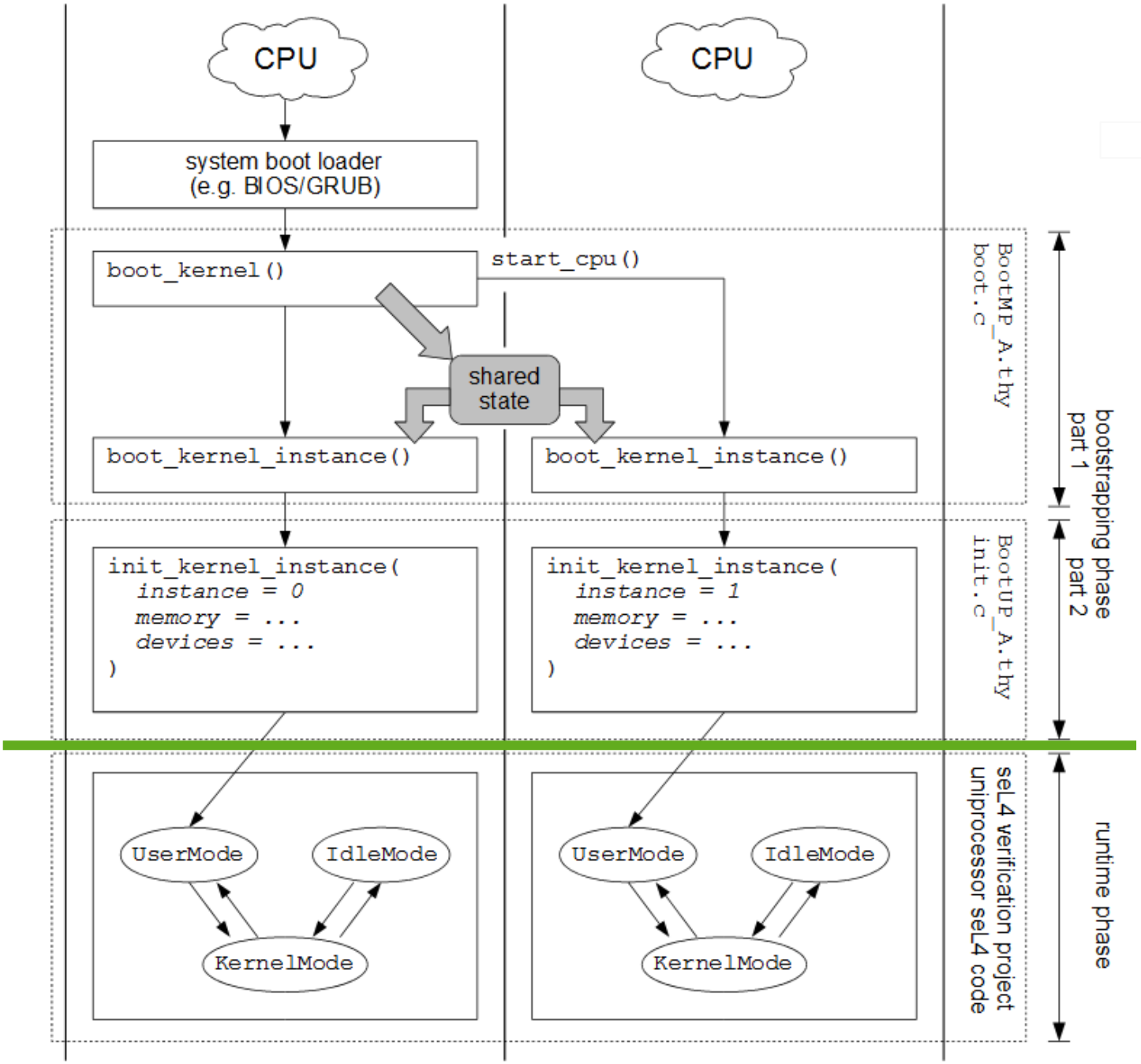
Modelling seL4's MP-specific Code

NICTA

- advantages:
  - decouples reasoning about concurrency artefacts from reasoning about program internals (functionality)
  - makes modelling and proofs more modular

- limitations:
  - restricts the kind of parallel programs and properties that can be proved
  - e.g. no lock-free data structures and algorithms (where functionality and concurrency are inherently coupled)

→ not a problem because I took special care while doing the **multikernel** design/implementation of seL4


CPU
CPU
system boot loader
(e.g. BIOS/GRUB)
boot_kernel()
start_cpu()
shared
state
boot_kernel_instance()
boot_kernel_instance()
init_kernel_instance(
instance = 0
memory = ...
devices = ...
)
init_kernel_instance(
instance = 1
memory = ...
devices = ...
)
UserMode
IdleMode
KernelMode
UserMode
IdleMode
KernelMode
BootMP_A.thy
boot.c
BootUP_A.thy
init.c
seL4 verification project
uniprocessor seL4 code
bootstrapping phase
part 1
part 2
runtime phase

Modelling seL4's MP-specific Code

NICTA

- isolation → memory:
  - proof wouldn't be that hard if the kernel only had static data
  - seL4 allows allocating/deallocating kernel objects at runtime
  - heavily uses dynamic allocation during bootstrapping
  - memory contents *(kernel heap)* modeled as:
    - *partial function from memory addresses to kernel objects*
  - complexity of model:

Isabelle code:	semantics of model	data structures	functions
bootstrapping phase	200 LOC	400 LOC	600 LOC

Properties Proved

NICTA

1. Kernel-Memory-Access Theorem
   - The kernel behaves correctly wrt. concurrency, i.e. it always observes CPU-local sequential semantics.
2. Virtualisation Theorem
   - correctness of system software (e.g. VMM)
   - system software can rely on
     - the provided shared memory to actually be shared between instances.
     - all other memory (code, data, devices) not to be shared.
3. The refinement proof of uniprocessor seL4 remains valid.

Proof size:

- 5500 LOC of machine-checked Isabelle proof script
- mostly hoare triples

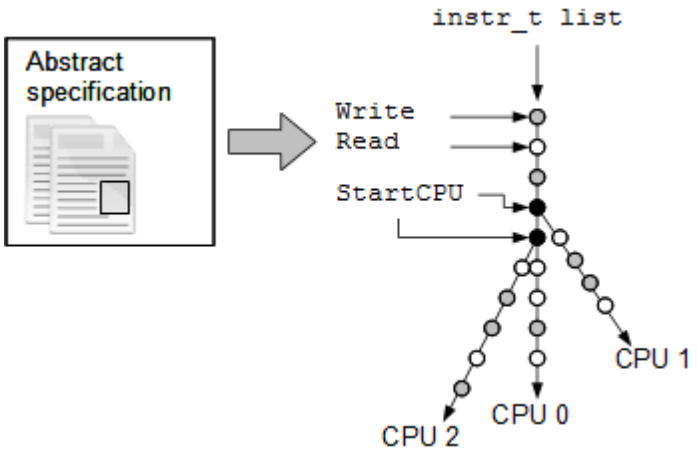
Kernel-Memory-Access Theorem

NICTA

Three proof steps:

1. prove that all kernel objects and capabilities created during bootstrapping of an seL4 instance lie in the memory region assigned to that instance
2. prove the following property about the parallel high-level-instruction program of the bootstrapping phase:

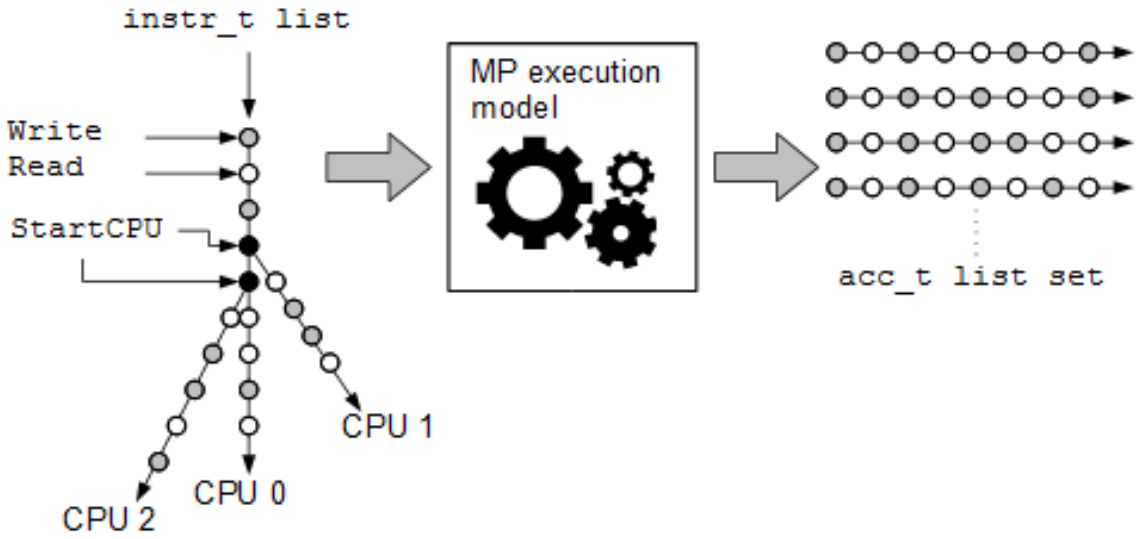
"For all **StartCPU** high-level instructions,
there no overlap between the physical memory region
read by the subsequent instructions and **written** by the instructions of the started CPU,
written by the subsequent instructions and **read** by the instructions of the started CPU."



Kernel-Memory-Access Theorem

3. [almost completed]
   prove, using the MP execution model, that all observable memory access histories exhibit sequential semantics:

"For all physical addresses and for all CPUs,
between every pair of a read/write followed by a later read,
no write by another CPU must occur."



Lifting the L4.verified proof

NICTA

"The refinement proof of uniprocessor seL4 remains valid."

- informal argument:
  - L4.verified proof only covers the runtime phase, not the bootstrapping phase
  - for the multikernel design, only bootstrapping code had to be changed
  - the code of the runtime phase stayed the same
  - each instance observes sequential semantics

  → refinement proof still holds

- formal connection to L4.verified model/proof:
  - state relation between my kernel state and L4.verified's (~150 LOC)
  - L4.verified refinement statement: $\mathbf{C} \sqsubseteq \mathbf{A}$
  - multikernel refinement statement: **C** inst_id $\sqsubseteq$ **A** inst_id
  - only had to modify a few dozen LOC in the top-level refinement proof

→ details are in paper

Conclusion

NICTA

Contributions:

- two kernel designs that avoid concurrency:
  - **big-lock** design
  - **multikernel** design, extended with shared memory provided to system software (e.g. VMM)
- multiprocessor execution model
  - **input**: a parallel high-level-instruction program
  - **output**: all observable memory access histories
- abstract specification of seL4's multikernel-specific code
- kernel-memory-access and virtualisation theorems
- state relation connecting my kernel state with L4.verified's
- new multikernel top-level refinement statement

interested in the Isabelle code?

→ michael.vontessin@nicta.com.au

Thank you!