School of Computer Science & Engineering

Trustworthy Systems Group

Adding Memory Barriers to Pancake

Rihui (Kurt) Wu

rihui.wu@unsw.edu.au

Question:

how to add memory barriers to Pancake

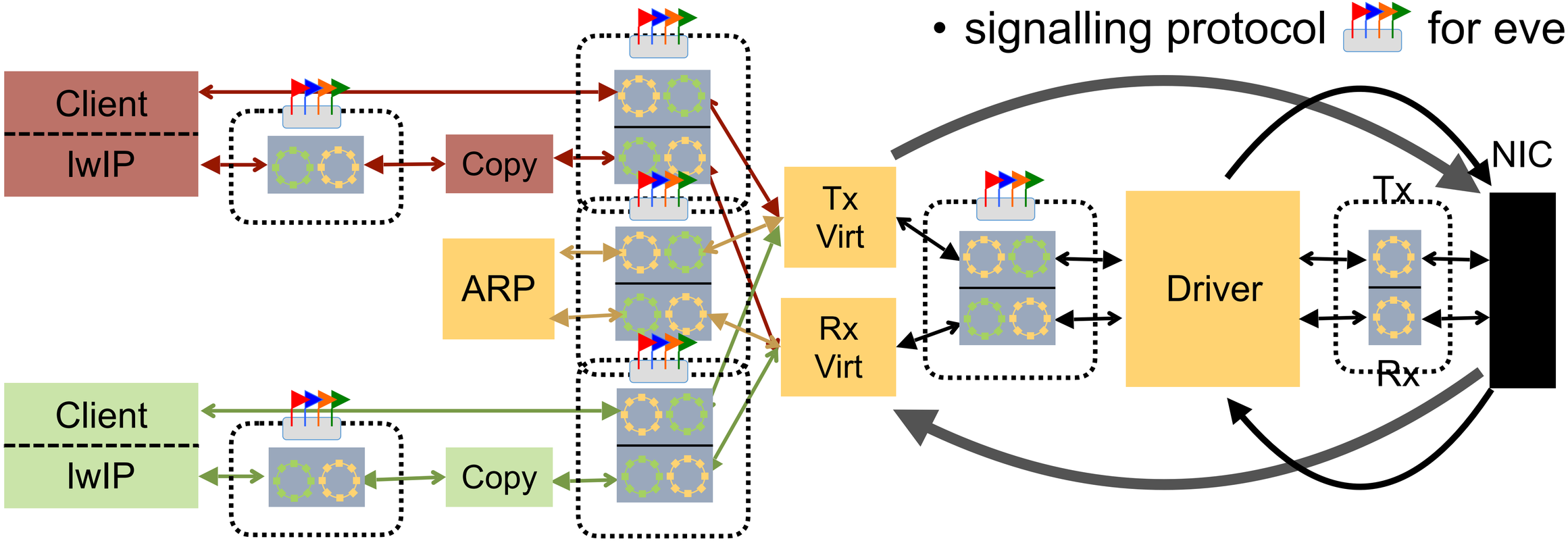
Question:

how to add memory barriers to Pancake

how to add *concurrency* primitives to a *sequential* language

Networking: Where is concurrency?

- SPSC queues
- signalling protocol for events



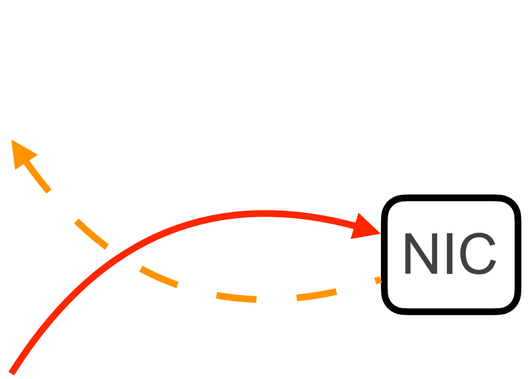
Components may be on the same/different cores.

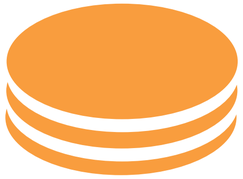
- MMIO registers (control, driver ↔ NIC)
- DMA (data, virtualisers ↔ NIC)

Memory barriers in LionsOS

- Memory barriers for concurrency control.

```
fn tx() {
    st MEM, desc;
    wmb;
    st DOORBELL, 1;
}
```



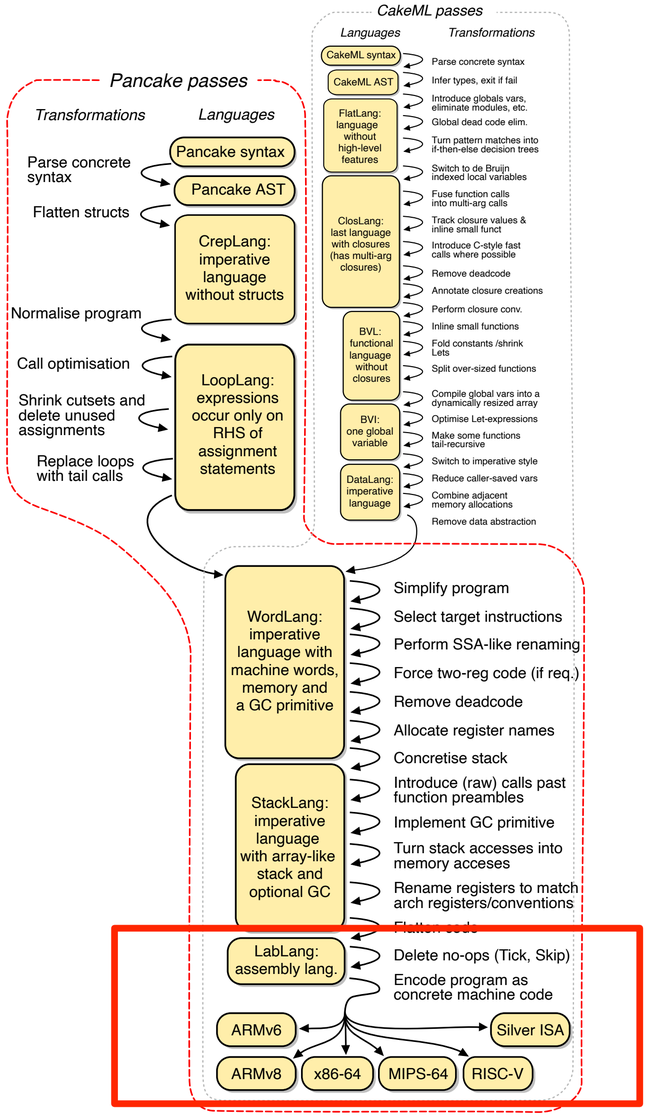


PANCAKE

A Language for Verified Systems Programming

- A verified language implemented in the HOL4 theorem prover.

- The compiler correctness theorem says that language semantics are preserved down to machine code.



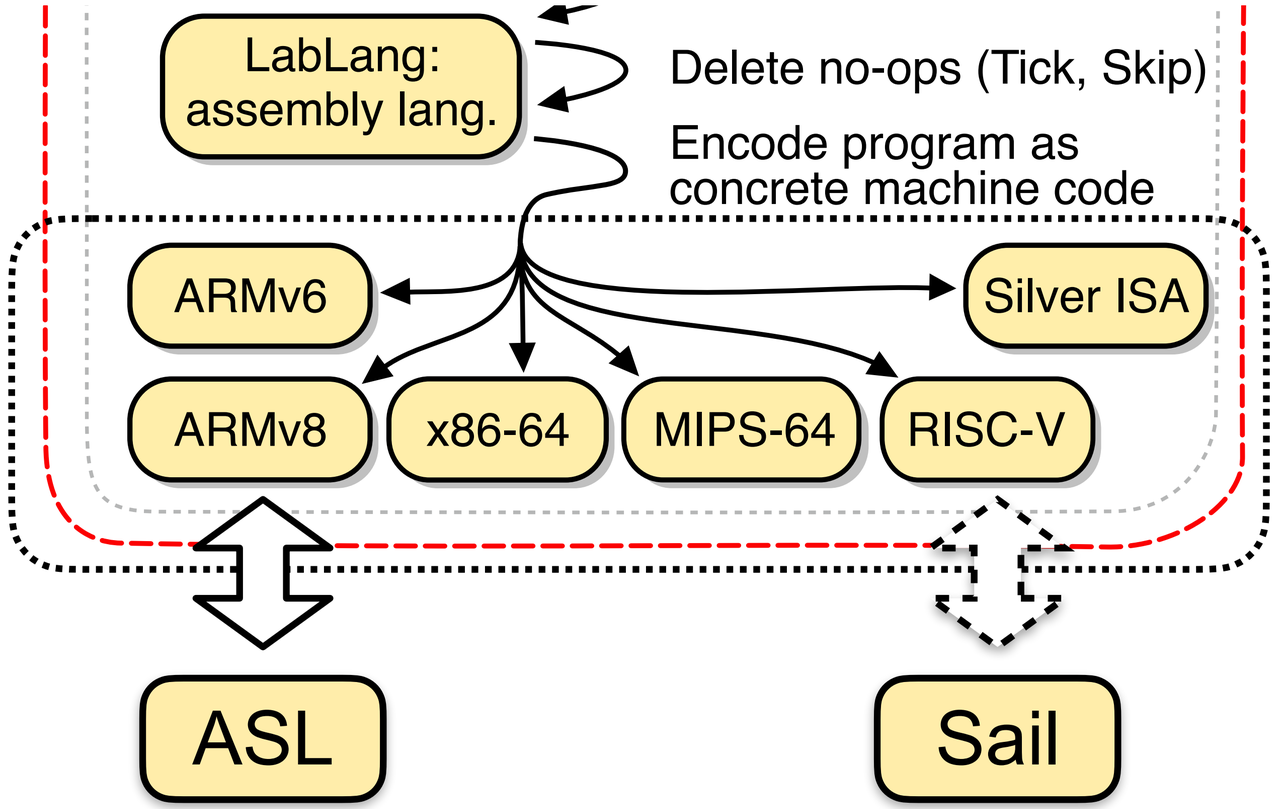
UNSW SYDNEY

Pancake ISA semantics

- Based on L3, a sequential machine code model.

- Describes the semantics of individual instructions as both
  - low-level state transformations
  - high-level observational semantics

- Equivalence proof w.r.t.
  - Arm's ASL
  - Sail for RISC-V (ongoing)



ASL's memory barrier semantics

- Surprisingly, memory barrier is modelled as a no-op.

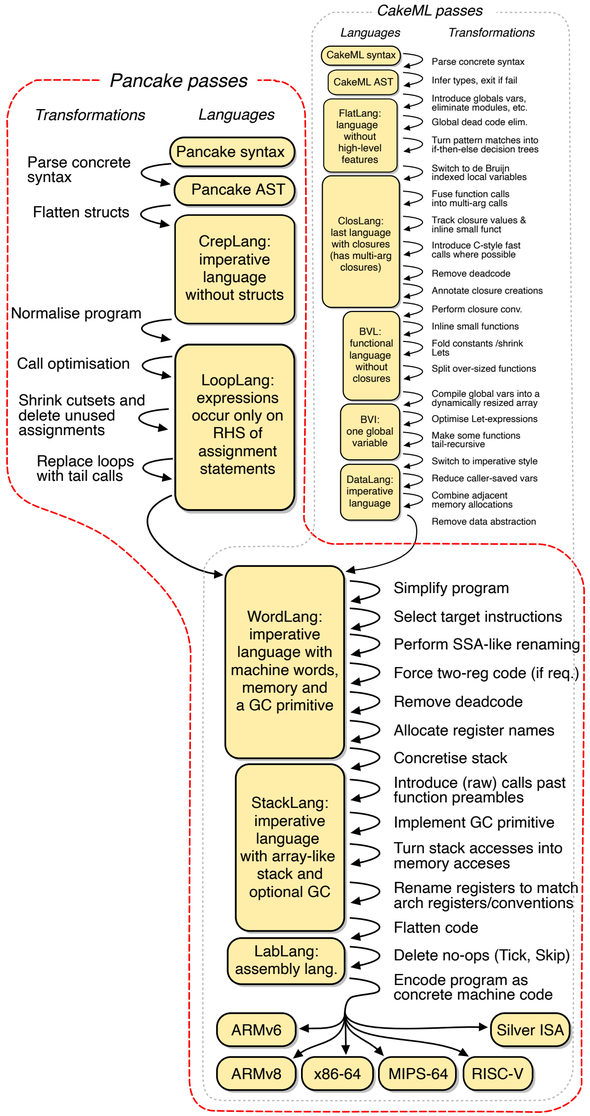
```
impdef func DataMemoryBarrier(types : MBReqTypes)
begin
    return;
end;
```

- Because ASL is a sequential language, concurrency is out of the scope.

Solution 1: no-ops semantics

- ✅ minimal compiler changes required
  - extend each compiler intermediate language with a new constructor
  - pass the constructor throughout different IRs untouched
  - bypass compiler optimisations

type	lines of code
arch-independent code	53
arch-independent proof	107
armv8 target code	3
armv8 target proof	6



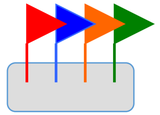
Solution 1: no-ops semantics

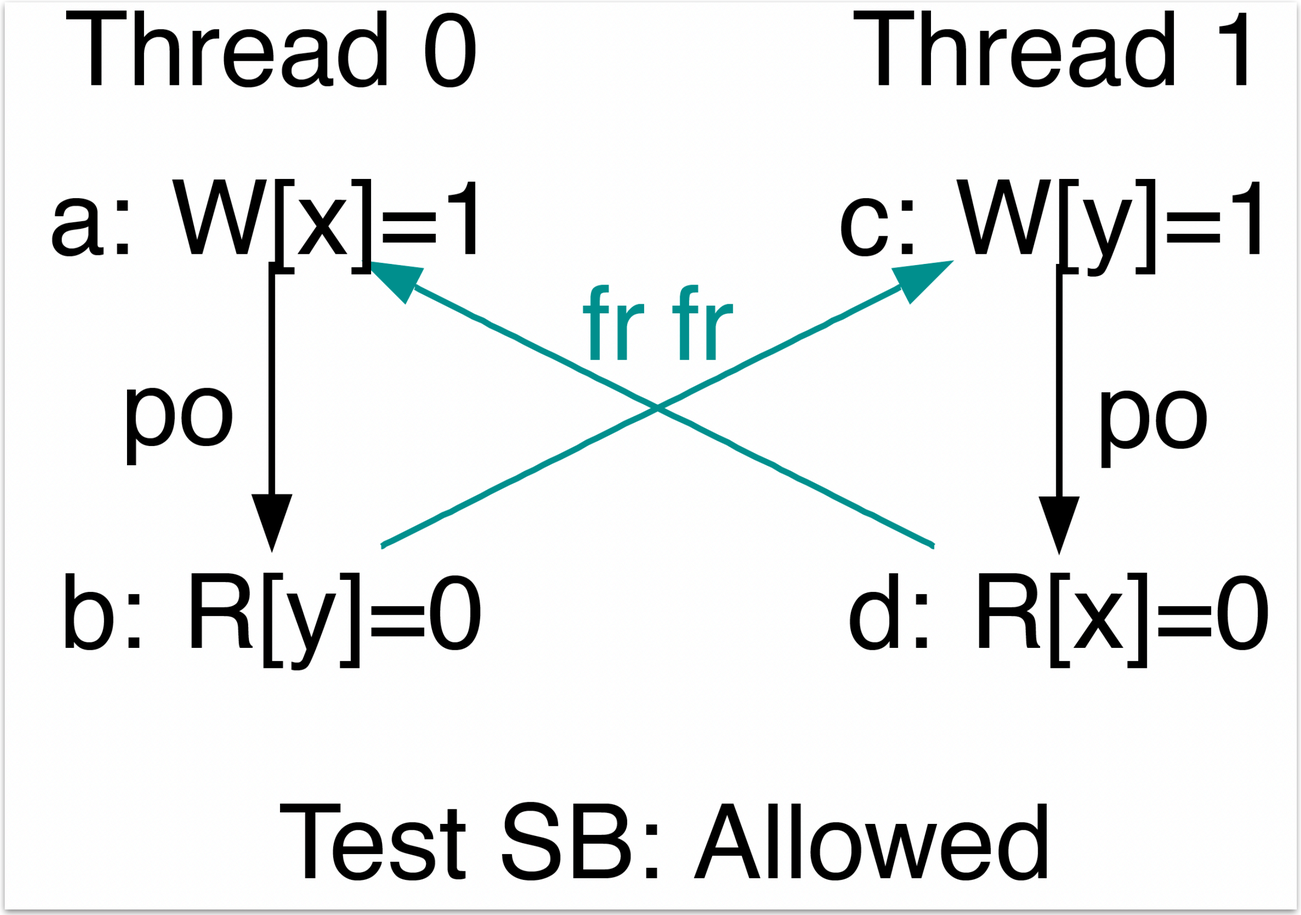
- Is this a good solution?

- ❌ information loss
  - what if a compiler pass removes a memory barrier?
  - what if a compiler pass introduces new memory barriers?

- what do memory barriers really mean?

concurrency semantics

- axiomatic modelling
- for the signalling protocol  representing a producer and a consumer
- if both components do not observe the data, deadlocks occur



concurrency semantics

- Other semantics exist.
  - Micro-architectural operational model.
  - Operational semantics with explicit reordering.
  - Operational promising semantics and view-based semantics.
  - ...
- However, none of them is sufficient for OS code.

Peripheral Arrival Order in Arm

- For normal memory, there is a total order over writes to each location (coherence).

- For peripherals, there is a total order over memory accesses to that peripheral.
  - *Multiple* locations may be associated with that peripheral.

  - VM-dependent barrier requirements:
    - Device no-Reordering memory does not require explicit barriers.
    - Other configurations require explicit barriers, currently not used.

Missing device semantics

- A read from an MMIO register may not return the value from the latest write?
  - Need a detailed device model, will be addressed by device interface formalisation.

- Difference between shareability domains, e.g. inner-shareable?
  - ✅ Deprecated in the latest manual (published two months ago) for data memory barrier.

- Reordered accesses to normal memory and *non-Reordering* MMIO registers?
  - Yes, after consulting with Arm.

- Is device access multi-copy atomic?
  - Maybe? Based on Linux commit 22ec716.

The semantics are *unknown*.

- Ordering requirement for cache maintenance operations (for DMA)?

FFI semantics in pancake

- Used to model external unknown functions, e.g. I/O, and shared-memory loads and stores.
- Parameterised by an oracle function, representing external effects on the shared program state.
- The oracle is run after every program step.
- A good fit for the memory barrier semantics (and other similar operations).

Solution 2: FFI semantics

- When a barrier is executed, it is as if an unknown external function is called.
  - Equivalent to the current FFI-based implementation without function calls.
- The function may update the shared memory.
- As with I/O, the memory barriers executed in the program form a trace, and compiler correctness requires that the trace be preserved under compiler transformations.
- Under implementation.

Evaluation

- insignificant performance improvement for the Ethernet driver
  - runtime cost dominated by uncached memory loads (80% of stall cycles)

- native memory barrier lowers #instructions by 10%

Takeaway

- Can model memory barriers as no-ops, but that removes useful information.
- Instead, model memory barriers as FFI calls, to represent unknown semantics explicitly.
- Trivial to implement the no-ops approach.

Future work

- Add cache maintenance instructions following the same semantics, for DMAs
- Instantiate the FFI interface with concurrency models.
- Verify against the (future) concurrent semantics of ASL and Sail.

Thanks!