The sorry state of dependency ordering

**Hardware**

CPU architectures guarantee that some dependencies enforce externally-visible ordering between memory accesses.

**Performance**

Dependency ordering is generally cheaper than using explicit fences, particularly where the dependency exists naturally as part of the algorithm.

**Linux**

The kernel relies on dependency ordering as a basis for RCU, but also to implement ring buffers and parts of the scheduler using volatile casts (READ_ONCE/WRITE_ONCE).

**C Compiler**

No high-performance implementations exist of memory_order_consume and the kernel does not follow the C11 memory model anyway.
Types of dependency

Please try to use this terminology!

- **Control dependency**
  - Read -> write generally ordered by all CPU architectures
  - Read -> read control dependencies can often be reordered by hardware!

- **Data dependency**
  - Read -> write only
  - Supported by all CPU architectures

- **Address dependency**
  - Read -> read/write
  - rcu_dereference()
  - Ordered by all CPU architectures other than Alpha (where we insert a fence)
Harmful compiler transformations

Converting a read -> read address dependency into a control dependency breaks hardware ordering!

Address dependency

```c
x = READ_ONCE(*foo);
bar = &x[42];
y = READ_ONCE(*bar);
```

Control dependency

```c
x = READ_ONCE(*foo);
if (x == baz)
   bar = &baz[42];
else
   bar = &x[42];
y = READ_ONCE(*bar);
```

https://lore.kernel.org/linux-arm-kernel/20200630173734.14057-19-will@kernel.org/
https://lore.kernel.org/lkml/20150520005510.GA23559@linux.vnet.ibm.com/
Harmful compiler transformations

Converting a read -> read address dependency into a control dependency breaks hardware ordering!

seq = READ_ONCE(tkf->seq.sequence);
tkr = tkf->base + (seq & 0x01);
now = tkr->base;

https://lore.kernel.org/kernel-hardening/20200625085745.GD117543@hirez.programming.kicks-ass.net/
We actually disable lots of "valid" (read: the standard allows them, but they are completely wrong for the kernel) optimizations because they are wrong.

[...]

So in general, we very much expect the compiler to do sane code generation, and not (for example) do store tearing on normal word-sized things or add writes that weren't there originally etc.

-- Linus Torvalds

https://lore.kernel.org/lkml/CAHk-=wi_KeD1M- - SU_H92vJ-vNkDnAGhAS=RR1vNNGWKW+aA@mail.gmail.com/
Some discussion points

Can we provide tooling to help the kernel use dependency ordering without disabling compiler optimisations on a case-by-case basis?

- How can we enforce dependencies at the source level?
- Can we detect broken dependencies and/or insert fences?
- Are annotations a non-starter?
- Does LTO make the situation worse?
- Where do we draw the line between “optimising compiler” and “portable assembler”?

Please don’t throw the standard at us! :)  
https://wg21.link/p0124