Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

Linux Plumbers Conference
Dublin, Ireland September 12-14, 2022

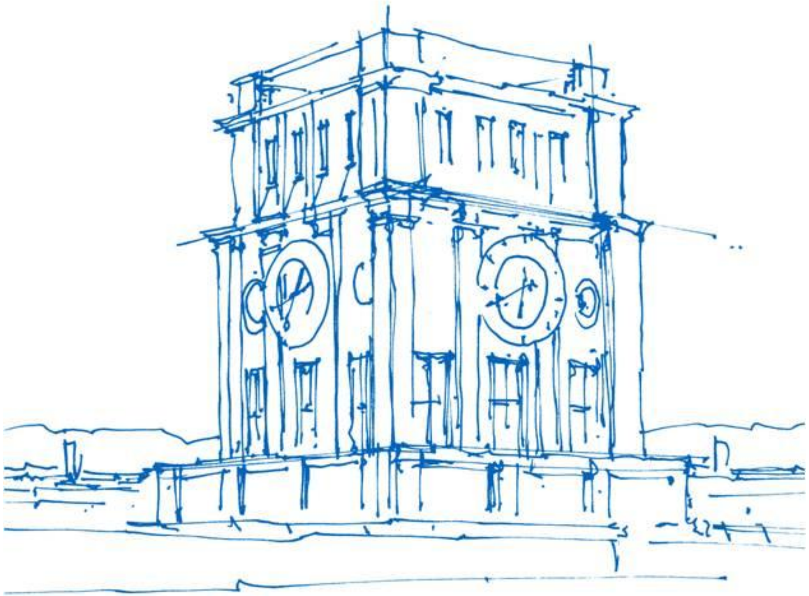
Status Report: Broken Dependency Orderings in the Linux Kernel

Paul Heidekrüger*, Marco Elver**

* *Technical University of Munich (TUM)*

** *Google*

Pramod Bhatotia (TUM)
Soham Chakraborty (TU Delft)
Martin Fink (TUM)
Charalampos Mainas (TUM)



Uhrenturm der TUM

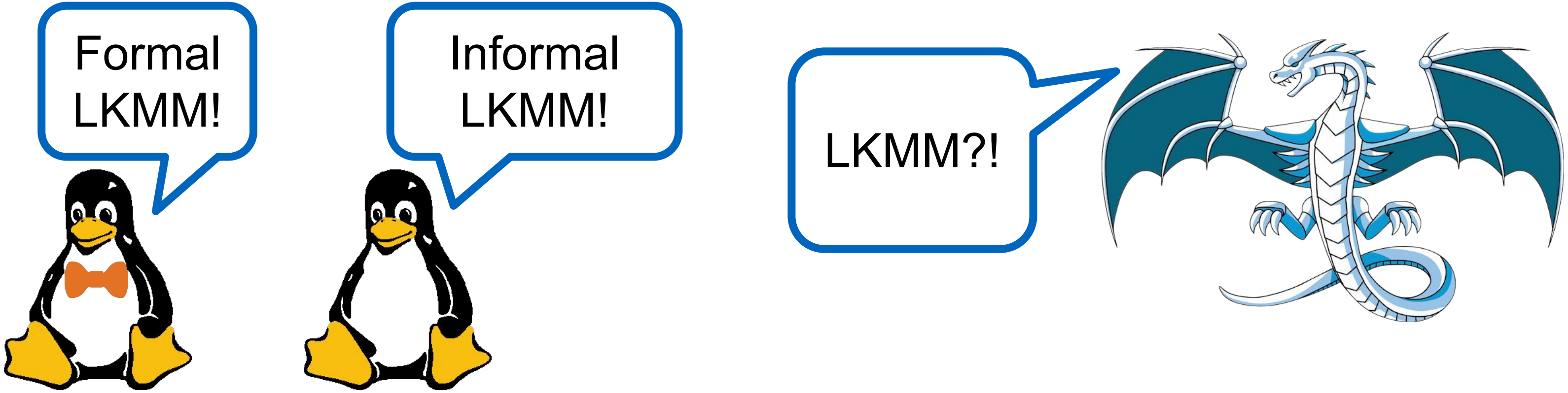
The “Fear” of Broken Dependencies

- **Linux kernel** uses **non-standard C**.
- **Linux-kernel Memory Consistency Model (LKMM)** differs from C11 memory model.
- **Compilers unaware of LKMM** ⇒ potential for miscompilations (?)

See "Who's afraid of a big bad optimizing compiler?" [https://lwn.net/Articles/793253/]

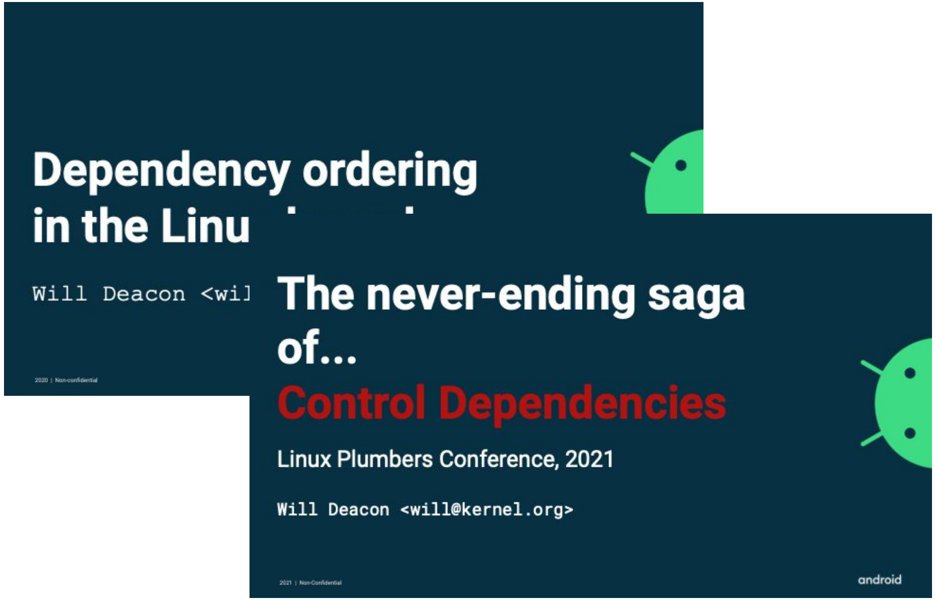
The “Fear” of Broken Dependencies

- **Linux kernel** uses **non-standard C**.
- **Linux-kernel Memory Consistency Model (LKMM)** differs from C11 memory model.
- **Compilers unaware of LKMM** ⇒ potential for miscompilations (?)



See "Who's afraid of a big bad optimizing compiler?" [https://lwn.net/Articles/793253/]

Previously on … Broken Dependency Orderings in the Linux Kernel



Dependency ordering in the Linu

Will Deacon <wi]

2020 | Non-confidential

The never-ending saga of...
Control Dependencies

Linux Plumbers Conference, 2021

Will Deacon <will@kernel.org>

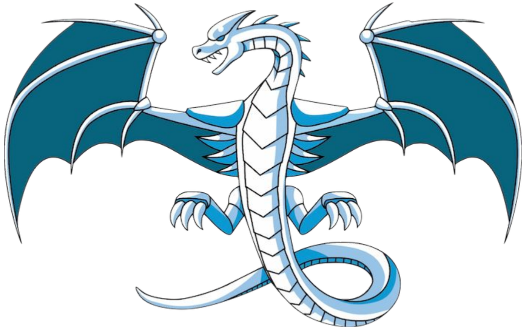
2021 | Non-Confidential

> “[...] but dammit, I want to see an actual real example arguing for why it would be relevant and why the compiler would need our help.”
>
> - Linus Torvalds -

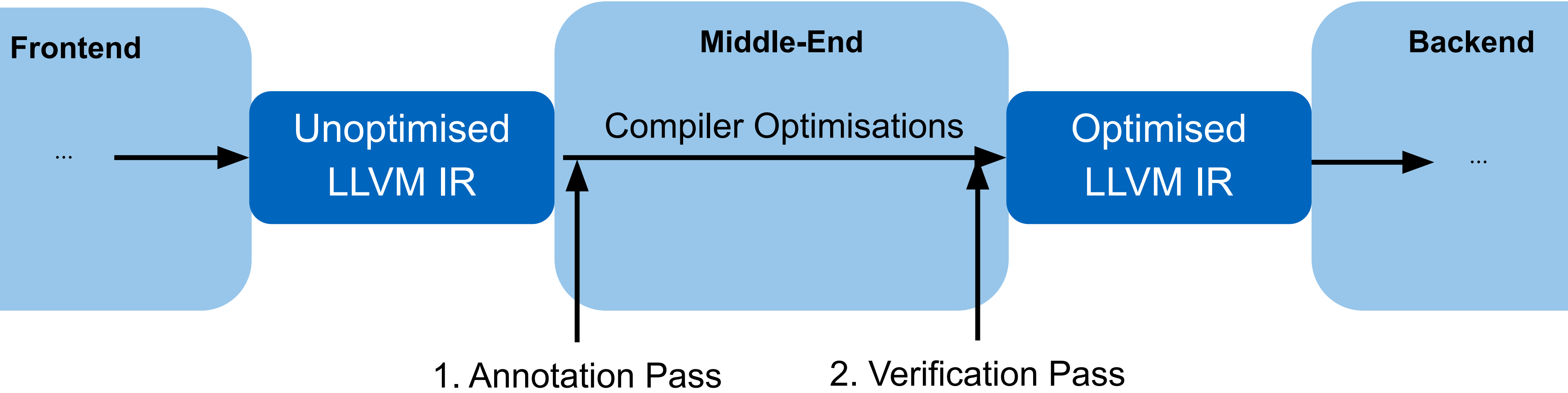
Dependency Ordering in the Linux Kernel: https://linuxplumbersconf.org/event/7/contributions/821/attachments/598/1075/LPC_2020_--_Dependency_ordering.pdf
The never-ending saga of … Control Dependencies: https://linuxplumbersconf.org/event/11/contributions/973/attachments/810/1587/Control%20deps%20-%20LPC%202021.pdf
Quote source: https://lore.kernel.org/all/CAHk-=wgmUbU6XPHz=4NFoLMxH7j_SR-ky4sKzOBrckmvk5AJow@mail.gmail.com/

LKMM Dependency Checker

- **Annotates** and **verifies syntactic addr** and **ctrl dependencies**
- **Interprocedural analysis** up to a given depth
- **ClangBuiltLinux**



Our Approach



Challenge: Unambiguous and Implementable Dependency Definitions for Analysis of Real Kernel Code

Part 1:

Address Dependencies
(and How to Break Them)

But What Is an Address Dependency?

> “A read event and another memory access event
> are linked by an address dependency
> if the value obtained by the read affects the
> location accessed by the other event.”
>
> - tools/memory-model/Documentation/explanation.txt -

Source: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/memory-model/Documentation/explanation.txt?h=v6.0-rc3

Here’s an Address Dependency

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

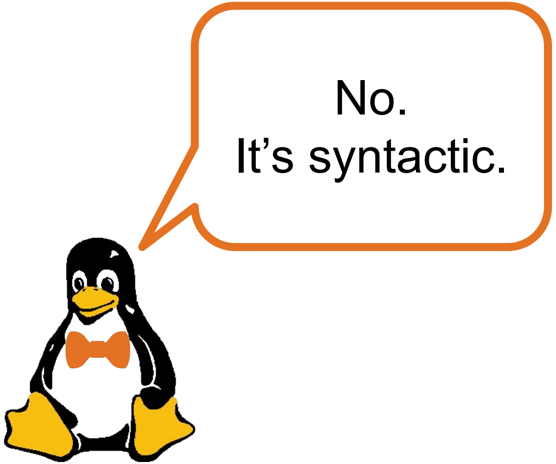
Source foo/bar/baz.c::42 - 44 (?)

Is This an Address Dependency?

```
int arr[1];


x = READ_ONCE(foo);
y = READ_ONCE(arr[x]);
```

Is This an Address Dependency?

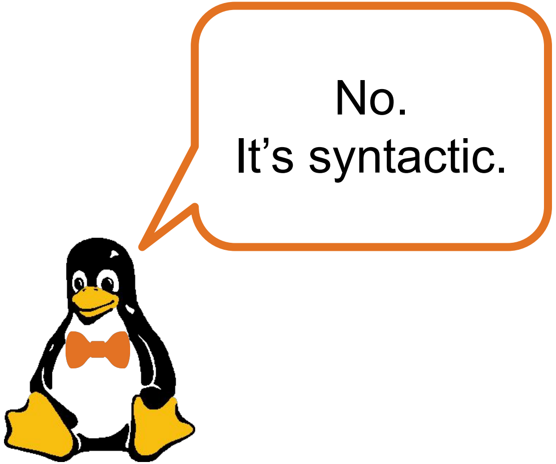
```
int arr[1];


x = READ_ONCE(foo);
y = READ_ONCE(arr[x]);
```



For a discussion on syntactic dependencies see:
https://lore.kernel.org/all/YXknxGFjvaB46d%2Fp@Pauls-MacBook-Pro/

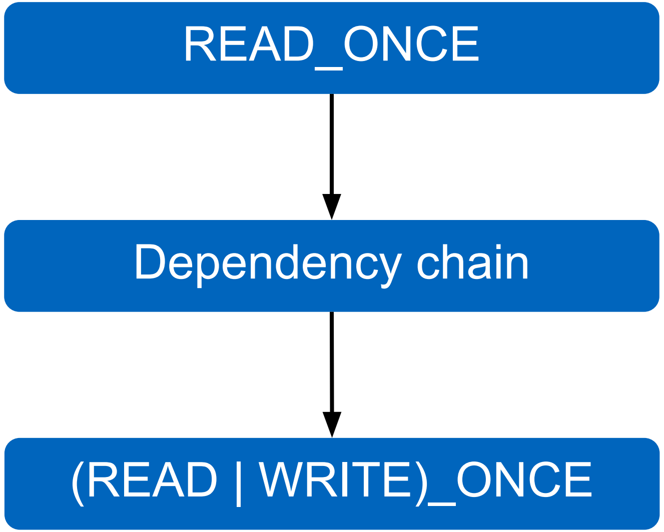
Is This an Address Dependency?

```
int arr[1];


x = READ_ONCE(foo);
y = READ_ONCE(arr[0]);
```



For a discussion on syntactic dependencies see:
https://lore.kernel.org/all/YXknxGFjvaB46d%2Fp@Pauls-MacBook-Pro/

Breaking Address Dependencies

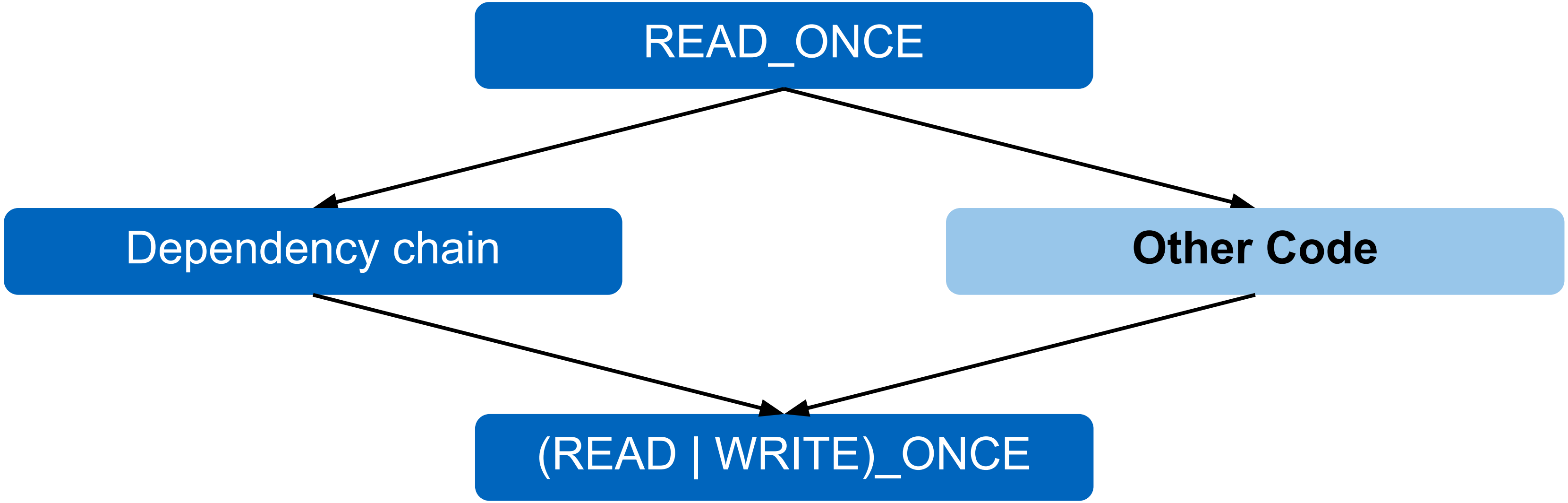


READ_ONCE

Dependency chain

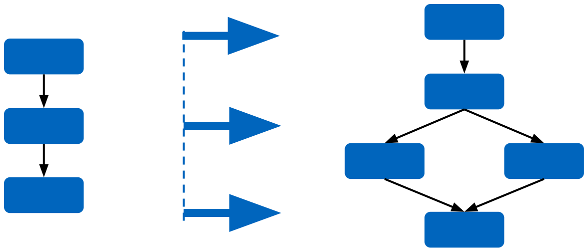
(READ | WRITE)_ONCE

Breaking Address Dependencies - Making the Dep Chain Conditional



Breaking Address Dependencies - Making the Dep Chain Conditional

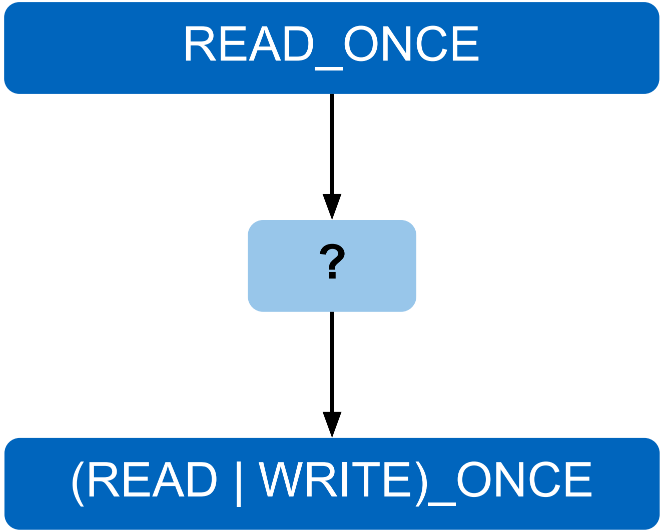
```
x = READ_ONCE(foo);

bar = &x[42];

y = READ_ONCE(*bar);
```



```
x = READ_ONCE(foo);

if (x == baz) /* Oh no! */

    bar = &baz[42];

else

    bar = &x[42];

y = READ_ONCE(*bar);
```

Source: foo/bar/baz.c::42 - 44 (?)

Breaking Address Dependencies - No Dep Chain



Broken Dependency Chain (mm/ksm.c::2032)

```
stable_node = page_stable_node(page); /* READ_ONCE(...) */
if (stable_node)
    /*More code*/
    stable_node->head = &migrate_nodes;
    list_add(&stable_node->list, stable_node->head); /* WRITE_ONCE() */
```

Source: mm/ksm.c::2032 - 2046

```
stable_node = page_stable_node(page); /* READ_ONCE(...) */
if (stable_node)
    /*More code*/
    stable_node->head = &migrate_nodes;
    list_add(&stable_node->list, stable_node->head); /* WRITE_ONCE() */
```

```
stable_node = page_stable_node(page); /* READ_ONCE(...) */
if (stable_node)
    /*More code*/
    /* Oh no! */
    list_add(&stable_node->list, &migrate_nodes); /* WRITE_ONCE() */
```

LKML Discussion: https://lore.kernel.org/all/YmKE%2FXgmRnGKrBbB@Pauls-MacBook-Pro.local/

Transformation to Conditional (fs/nfs/delegation.c::617 - 622)

```
if (place_holder)
    delegation = rcu_dereference([…]); /* READ_ONCE() */
if (!delegation || delegation != place_holder_deleg)
    delegation = list_entry_rcu([…]);
for([…], delegation = list_entry_rcu(delegation, […])) /* READ_ONCE() */
```

Source: fs/nfs/delegation.c 617 - 622

```
if(place_holder == NULL) {
    delegation = list_entry_rcu([…]);
else {
    cmp = rcu_dereference(place_holder->delegation); /* READ_ONCE(...) */
    if(cmp != NULL) {
        if(cmp == place_holder_deleg) /* Oh no! */
            delegation = place_holder_deleg; /* Oh no! (cont.) */
        else
            delegation = list_entry_rcu([…]);
    } else {
        delegation = list_entry_rcu([…]);
    }
}
for([…], delegation = list_entry_rcu(delegation, […])) /* READ_ONCE() */
```

LKML Discussion: https://lore.kernel.org/all/Yk7%2FT8BJITwz+Og1@Pauls-MacBook-Pro.local/

Challenge: Unambiguous and Implementable Dependency Definitions for Analysis of Real Kernel Code

Part 2:

Control Dependencies

(and How to Break Them)

But What Is a Control Dependency?

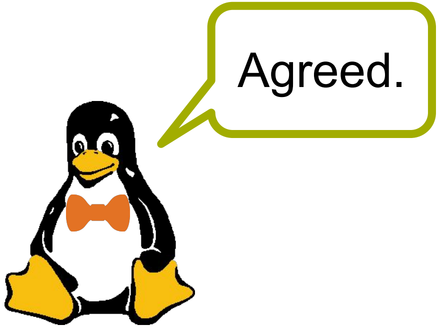
“Finally, a read event X and a write event Y are linked by a control dependency if Y syntactically lies within an arm of an if statement and X affects the evaluation of the if condition via a data or address dependency (or similarly for a switch statement)”

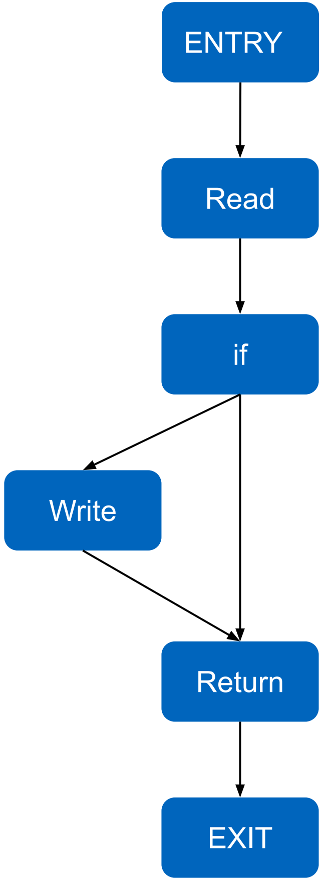
- tools/memory-model/Documentation/explanation.txt (in a few weeks) -

Source: https://lore.kernel.org/all/20220902211341.2585133-1-paul.heidekrueger@in.tum.de/

Here’s a Control Dependency

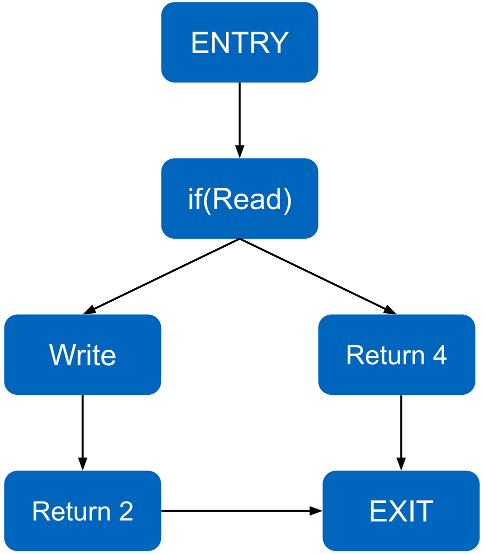
```
x = READ_ONCE(foo);
if(x == 42)
    WRITE_ONCE(bar, 42);
return true;
```





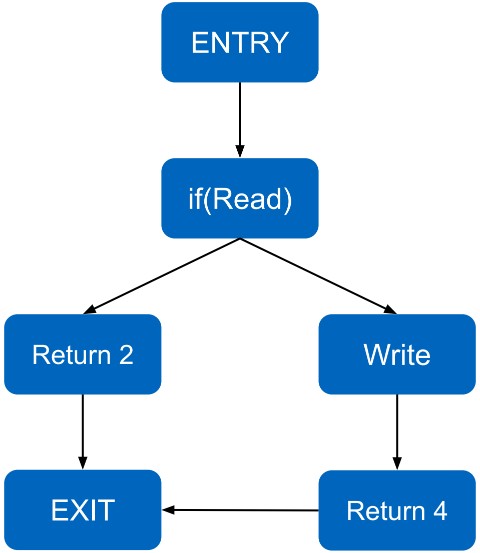
Is this a Control Dependency?

```
if(READ_ONCE(x))
   return 4;
WRITE_ONCE(y, 21);
return 2;
```



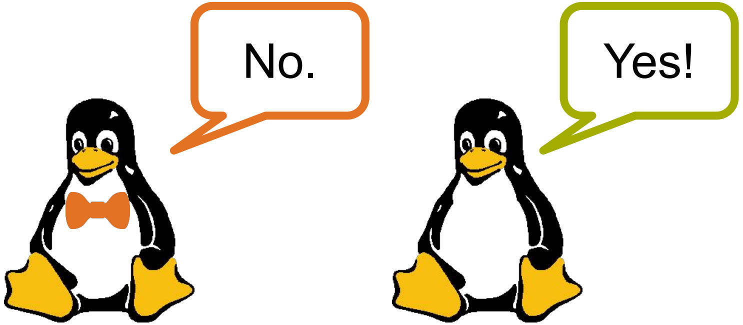
Is this a Control Dependency?

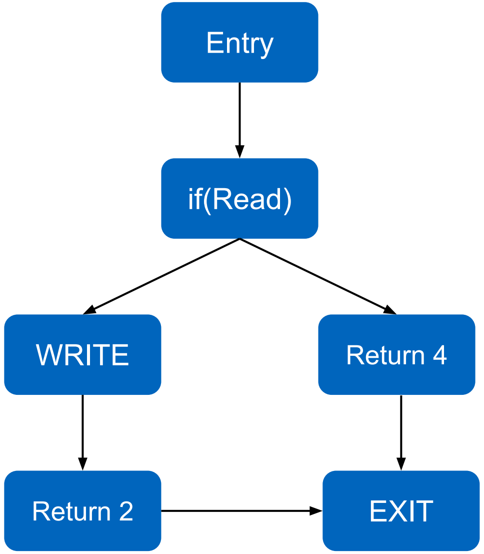
```
if(!READ_ONCE(x)) {
    WRITE_ONCE(y, 42);
    return 2;
}
return 4;
```



Is this a Control Dependency?

```
if(READ_ONCE(x))
    return 42;
WRITE_ONCE(y, 42); /* Uhm? */
return 0;
```




Entry

if(Read)

WRITE

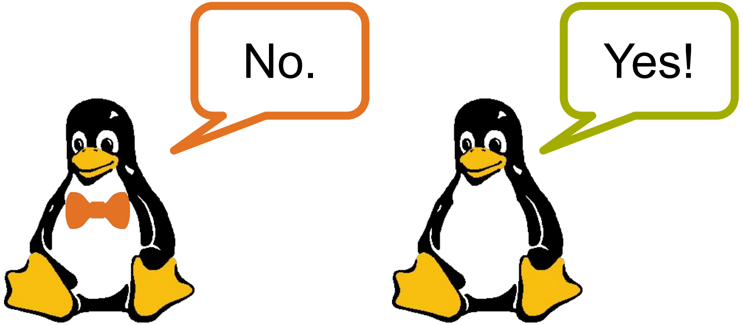
Return 4

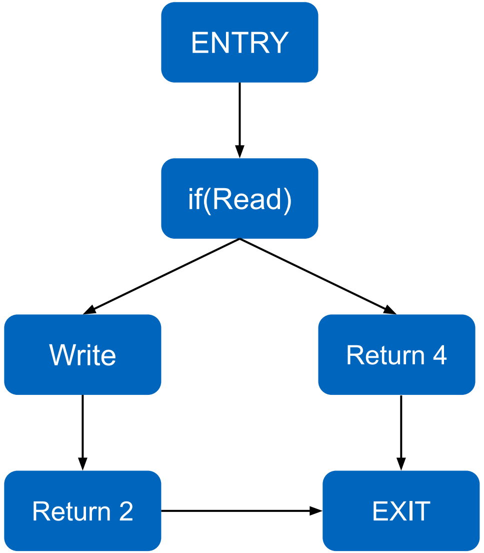
Return 2

EXIT

Is this a Control Dependency?

```
if(READ_ONCE(x))
   return 42;
WRITE_ONCE(y, 42);
/* The answer has to be Yes! */
return 0;
```





But What Is a Control Dependency?

> “Let G be a control flow graph. Let X and Y be nodes in G.
> Y is control dependent on X iff
> (1) there exists a directed path P from X to Y with any Z in P
> (excluding X and Y) post-dominated by Y and
> (2) X is not post-dominated by Y.”

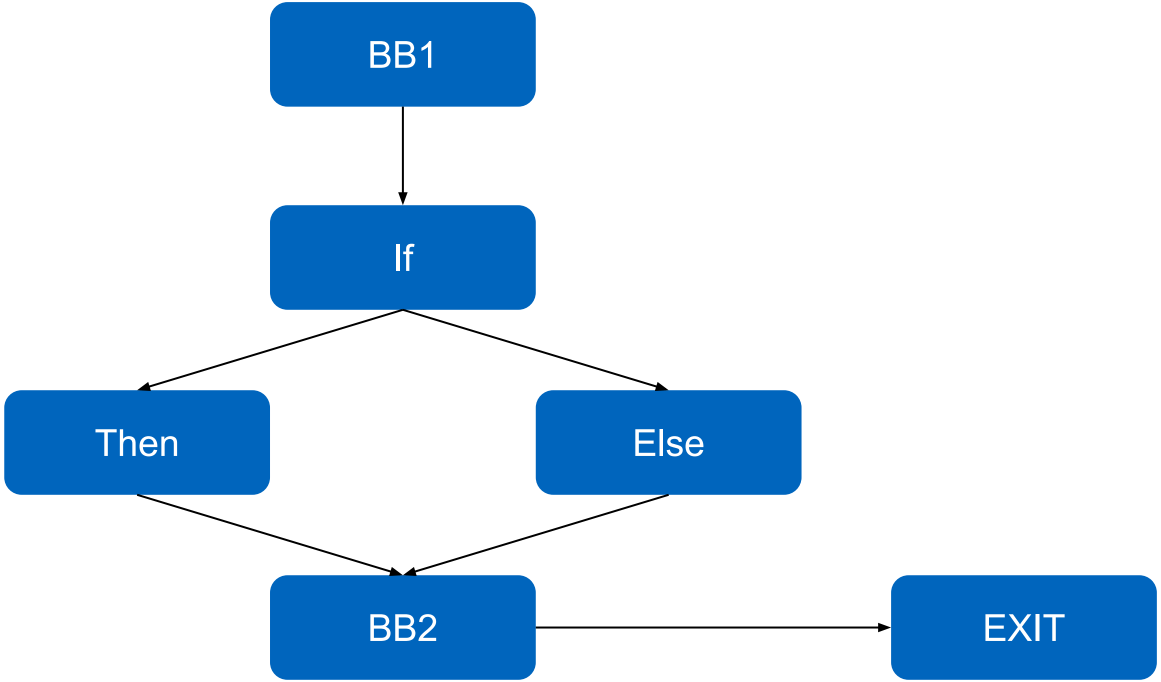
Source: Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. 1987. The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9, 3 (July 1987), 319–349. https://doi.org/10.1145/24039.24041

But What Is Post Dominance?

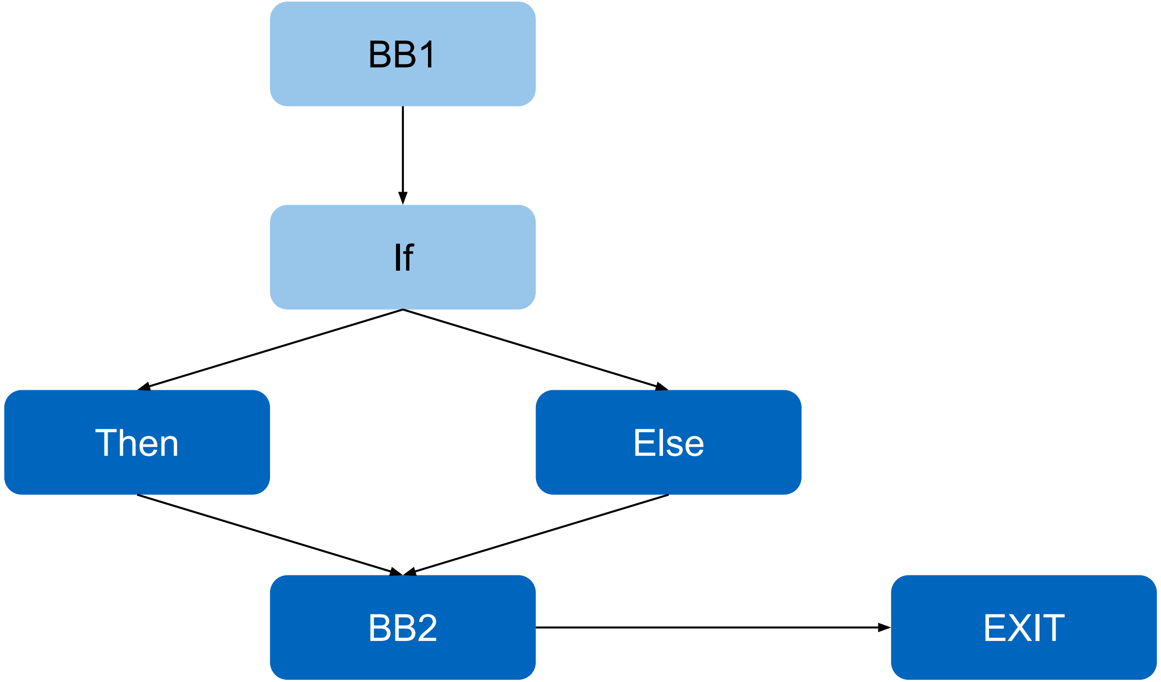
> “A node V is post-dominated by a node W in G
> if every directed path
> from V to STOP (not including V)
> contains W.”

[1]: Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. 1987. The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9, 3 (July 1987), 319–349. https://doi.org/10.1145/24039.24041

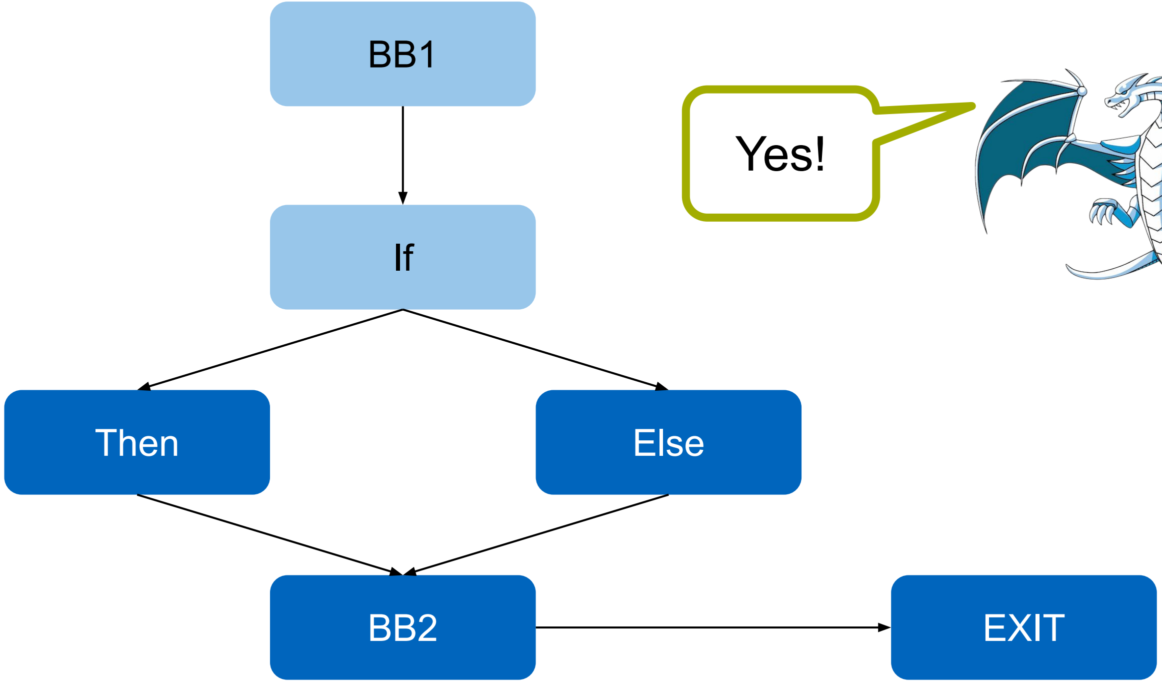
Illustrating Post-Dominance



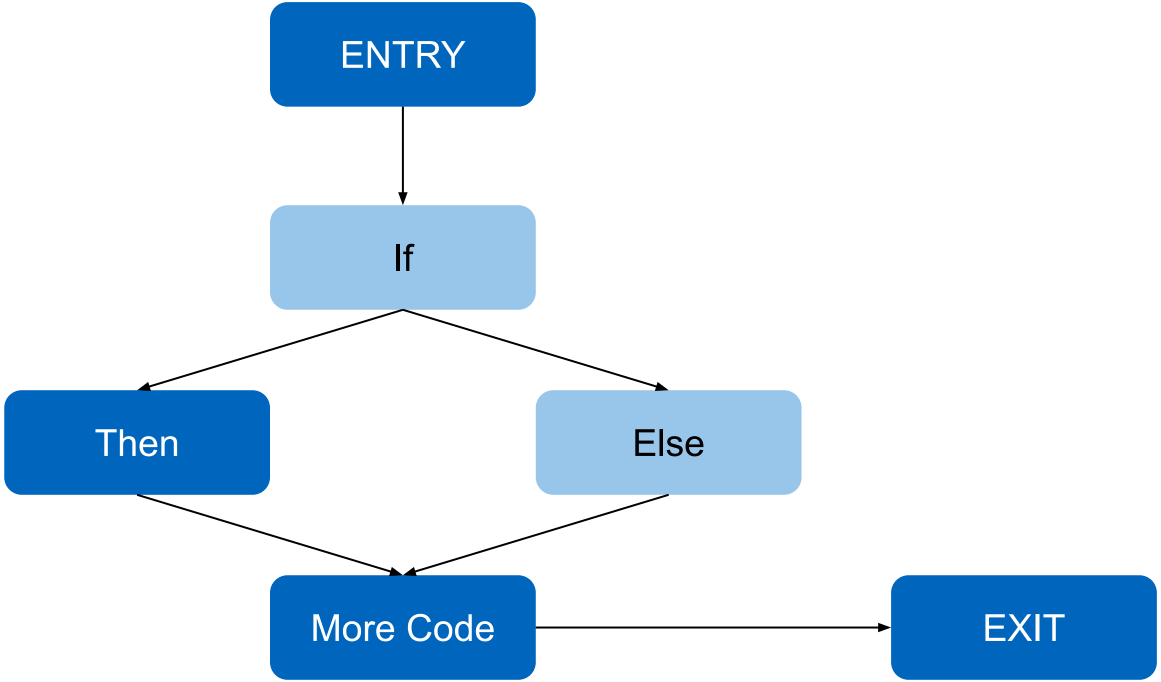
Does “If” Post-Dominate “BB1”?



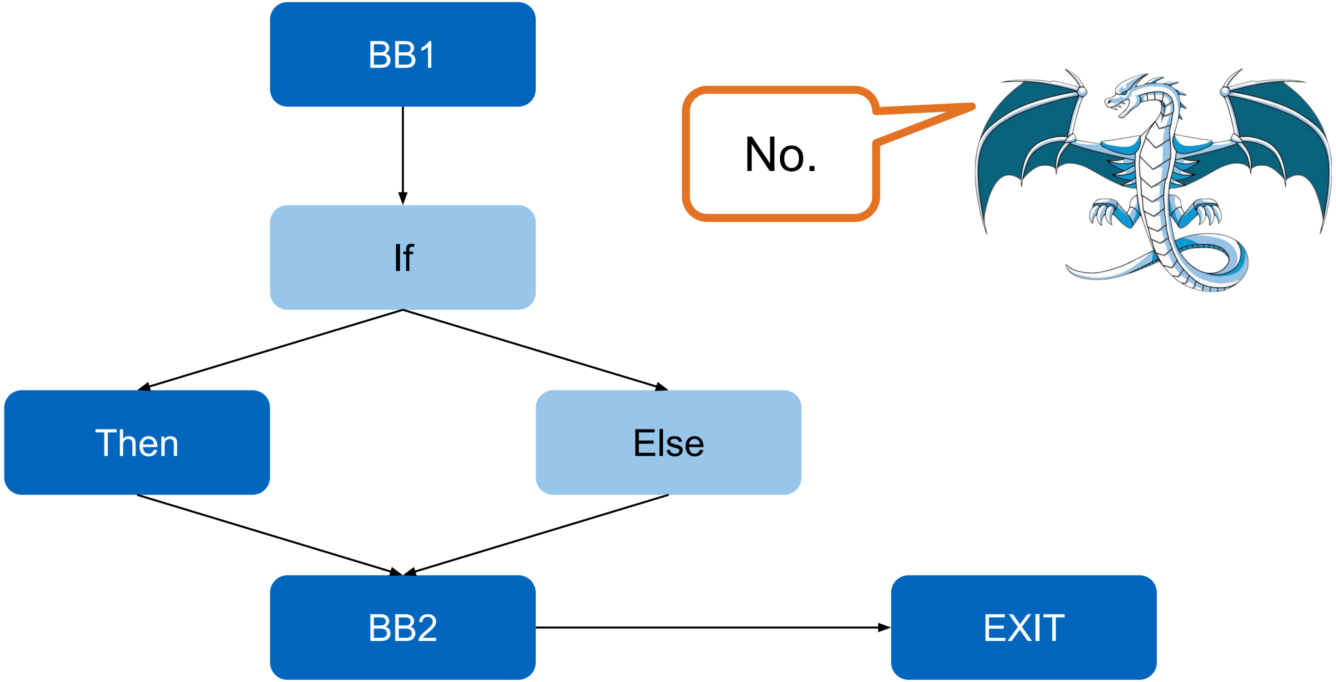
Does “If” Post-Dominate “BB1”?



Does “Then” Post-Dominate “If”?

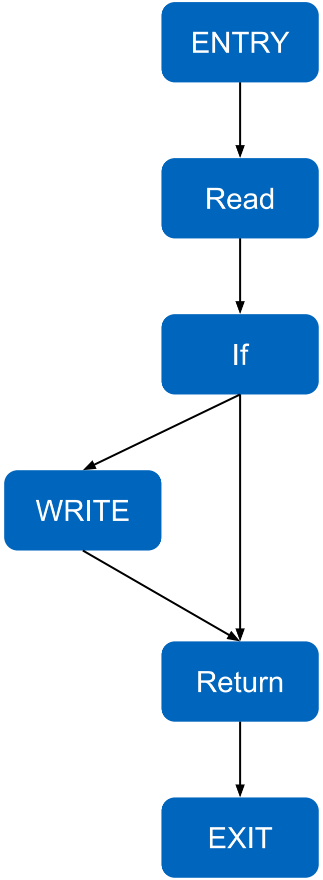


Does “Then” Post-Dominate “If”?



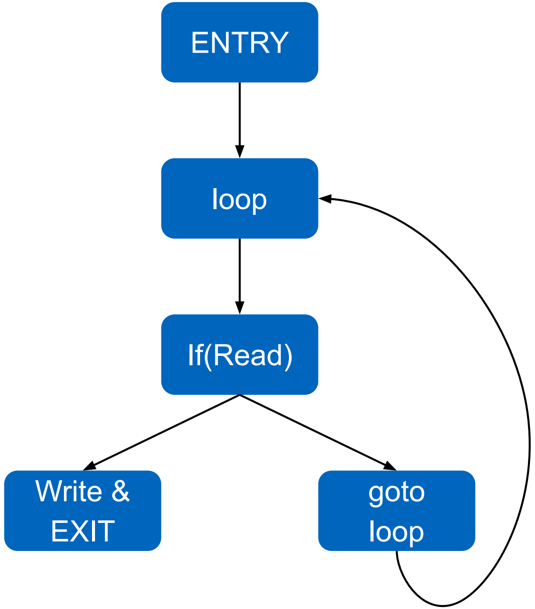
Here’s a Control Dependency

```
x = READ_ONCE(foo);
if(x == 42)
    WRITE_ONCE(bar, 24);
return true;
```



Is This a Control Dependency?

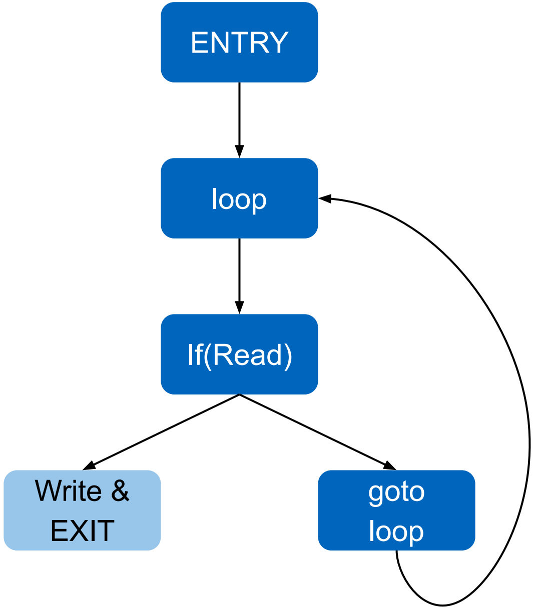
```
loop:
if(READ_ONCE(x)) {
    WRITE_ONCE(y, 42);
    return 0;
}
goto loop;
```



Loop in drivers/gpu/drm/qxl/qxl_cmd.c::227. Dependency in drivers/gpu/drm/qxl/qxl_cmd.c::162 - 169. Derived from optimised LLVM IR

Is This a Control Dependency?

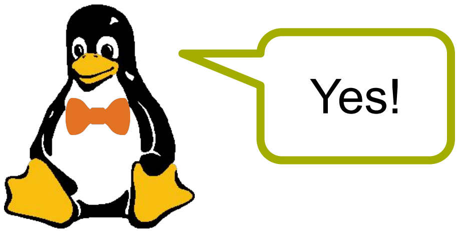
```
loop:
if(READ_ONCE(x)) {
   WRITE_ONCE(y, 42);
   return 0;
}
goto loop;
```

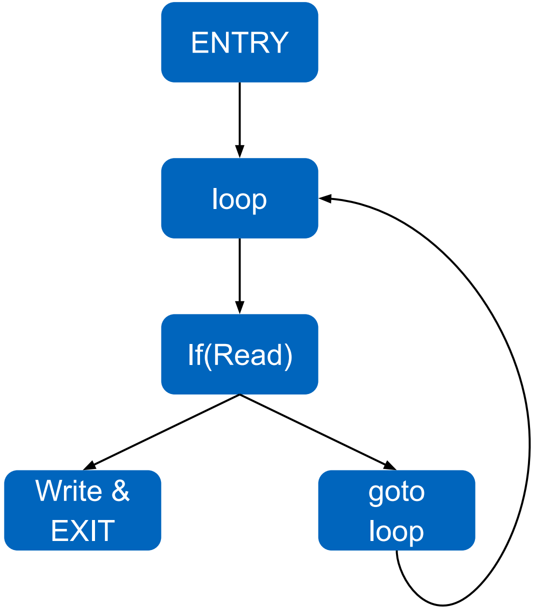


Loop in drivers/gpu/drm/qxl/qxl_cmd.c::227. Dependency in drivers/gpu/drm/qxl/qxl_cmd.c::162 - 169. Derived from optimised LLVM IR.

Is this a Control Dependency?

```
loop:
if(READ_ONCE(x)) {
   WRITE_ONCE(y, 42);
   return 0;
}
goto loop;
```




ENTRY

loop

If(Read)

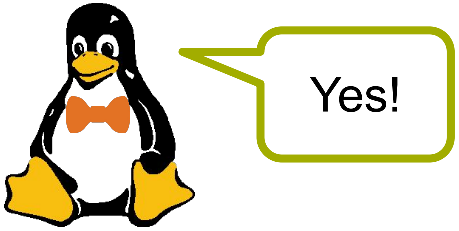
Write & EXIT

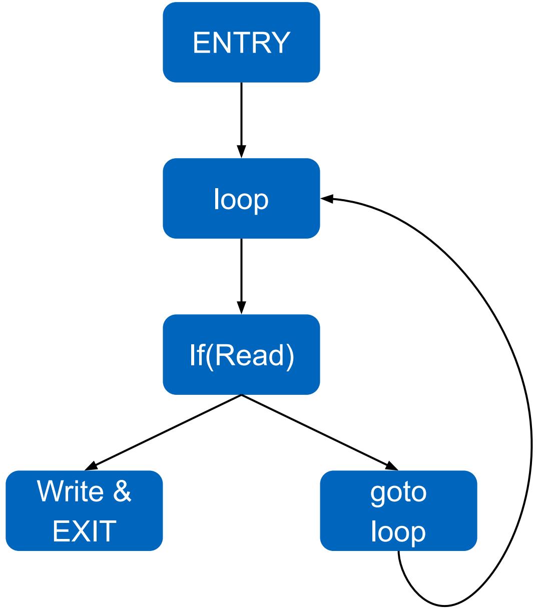
goto loop

Loop in drivers/gpu/drm/qxl/qxl_cmd.c::227. Dependency in drivers/gpu/drm/qxl/qxl_cmd.c::162 - 169. Derived from optimised LLVM IR.

Is this a Control Dependency?

```
loop:
if(READ_ONCE(x)) {
   WRITE_ONCE(y, 42); /* Yes! */
   return 0;
}
goto loop;
```



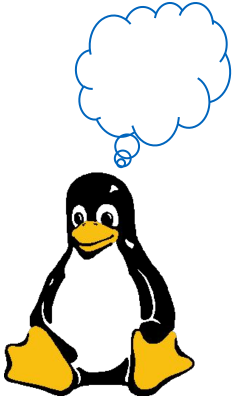


Loop in drivers/gpu/drm/qxl/qxl_cmd.c::227. Dependency in drivers/gpu/drm/qxl/qxl_cmd.c::162 - 169. Derived from optimised LLVM IR.

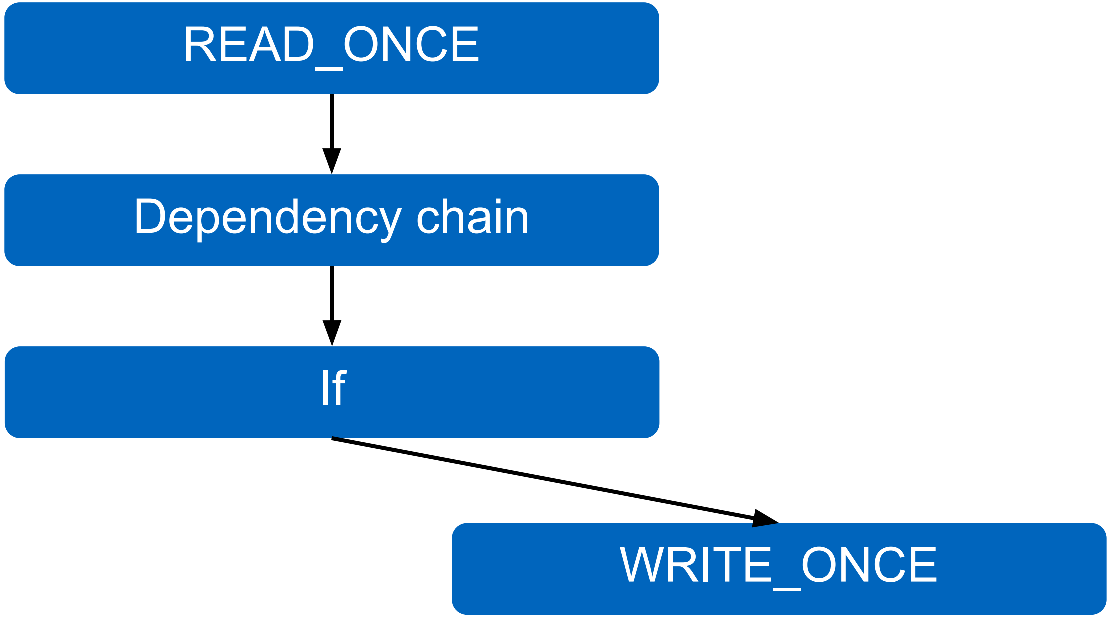
But What Is a Control Dependency?

There is a control dependency from a marked read A to a marked write B if there is a condition C s.t. B is within the scope of C and C depends on A.

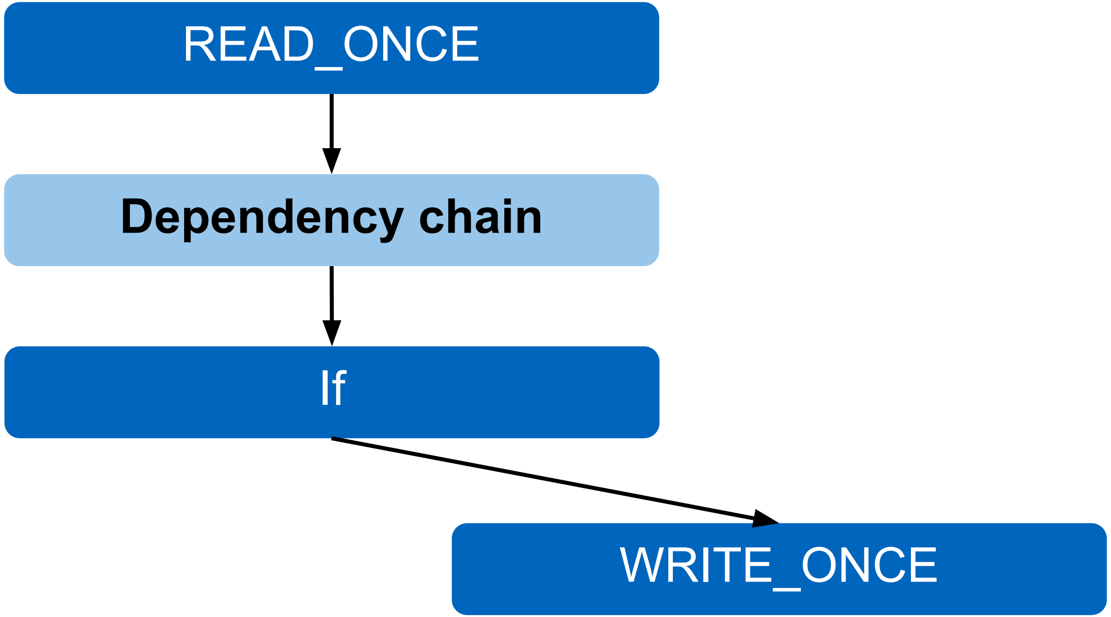
- WIP Control Dependency Definition -



Breaking Control Dependencies (?)



Breaking Control Dependencies (?)

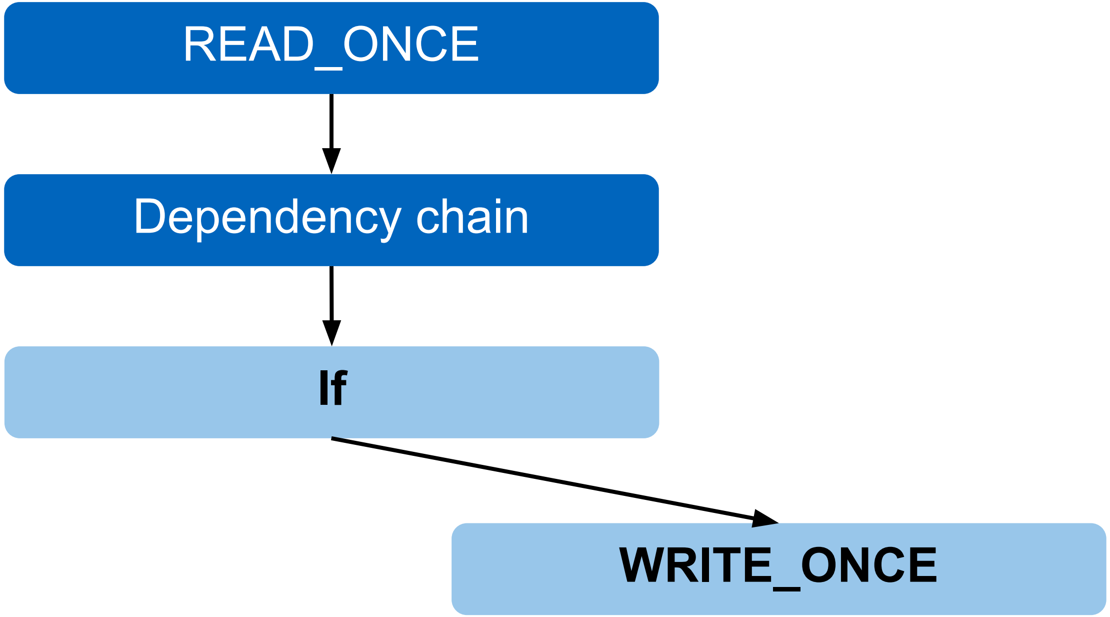

READ_ONCE

Dependency chain

If

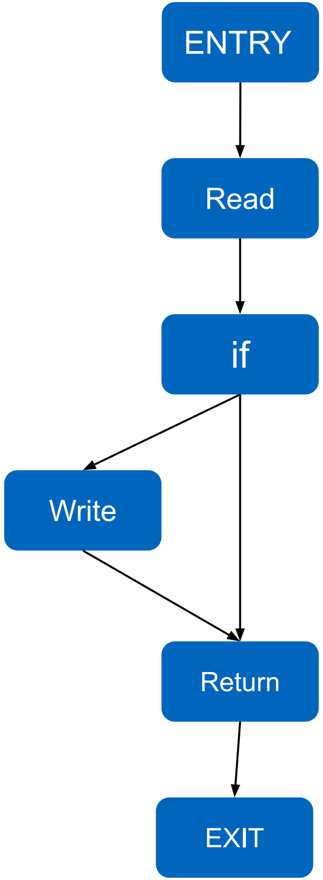
WRITE_ONCE

Breaking Control Dependencies (?)



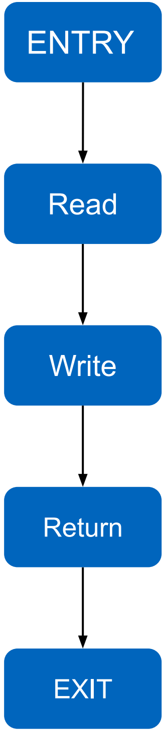
Is this a Control Dependency?

```
#define MAX 1


x = READ_ONCE(*foo);
if (x % MAX == 0)
    WRITE_ONCE(*bar, 1);
Return true;
```



Is this a Control Dependency?

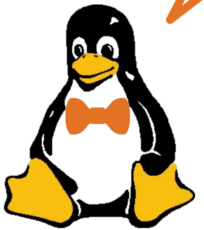
```
#define MAX 1


x = READ_ONCE(foo);
/* No branch?! */
WRITE_ONCE(*bar, 1);
Return true;
```



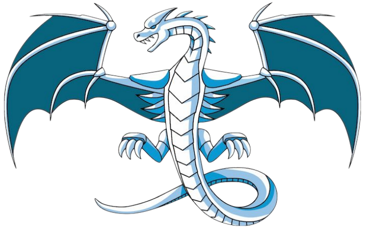
Is this a Control Dependency?

```
#define MAX 1


x = READ_ONCE(*foo);
if (x % MAX == 0)
    WRITE_ONCE(*bar, 1);
```

No.
It’s syntactic.



It depends.



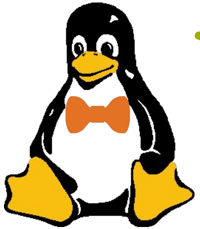
Is this a Control Dependency?

```
x = READ_ONCE(*bar);
if (x == 42)
    WRITE_ONCE(*baz, 1);
else
    WRITE_ONCE(*baz, 2);
WRITE_ONCE(*y, 42);
```

Example by Will Deacon

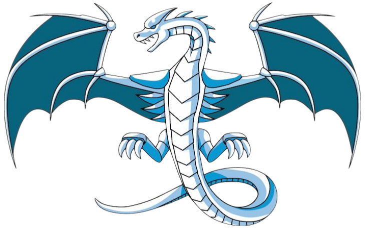
Is this a Control Dependency?

```
x = READ_ONCE(*bar);
if (x == 42)
    /* arm64 compiler can make this a conditional select */
    WRITE_ONCE(*baz, 1);
else
    WRITE_ONCE(*baz, 2);
WRITE_ONCE(*y, 42);
```



Plural - Control Dependencies! There is more than one.

It depends.

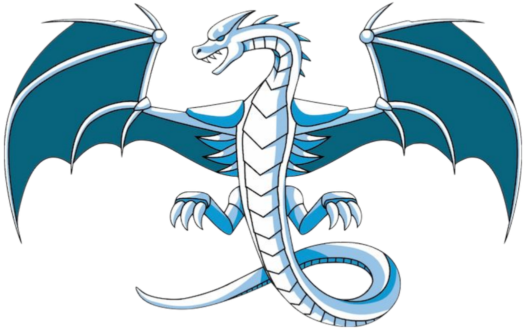


Example by Will Deacon

Where Does this Leave Us?

“To What Extent Are Compilers Undermining the Linux Kernel Memory Model?”

- LKMM **addr dependency checker has found broken dependencies**
- LKMM addr dependency checker **LLVM RFC** hopefully very soon
- WIP LKMM **ctrl dependency checker**

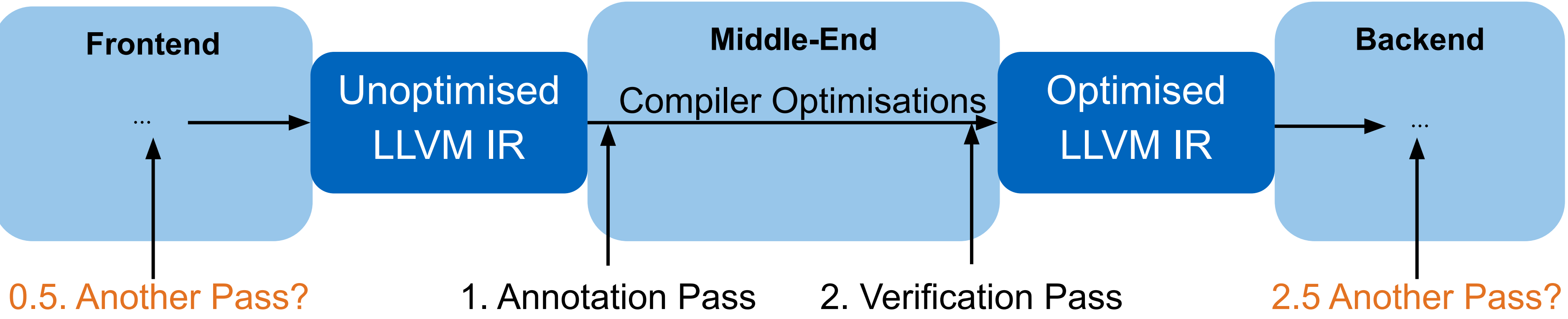


Broken dependencies are real! 😱

The LKMM Dependency Checker found broken dependencies!

Limitations

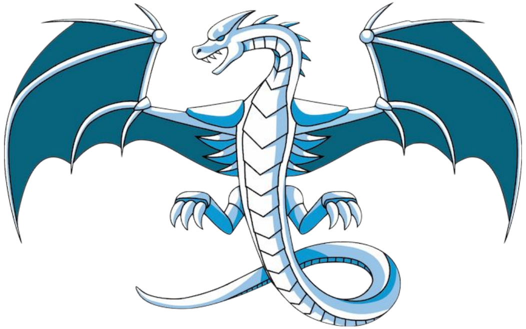
- **Syntactic dependency** !=> **semantic dependency**
- **Computational feasibility** (interprocedural analysis, variadic functions …)
- Our analysis avoids **backedges** (right now)
- **Frontend** and **backend** optimisations not accounted for



The Hunt for Broken Dependencies - Future Plans

"To What Extent Are Compilers Undermining the Linux Kernel Memory Model?"

- **AArch64 defconfigs** for various Linux kernel flavours (**incl. LTO**)
- **Random testing** with random, but relevant, Linux kernel configs
- **LLVM RFCs**
- Find **ctrl to addr** dependency **transformations**
- Extend our analysis to the **backend**
- Get the ball rolling on **short-term** to **long-term fixes** with sufficient evidence



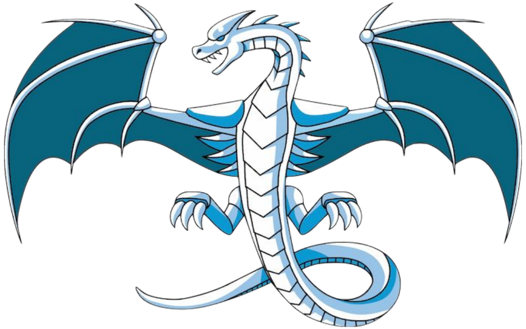
Get in touch!

twitter.com/pbhdk Paul.Heidekrueger@in.tum.de linkedin.com/in/pbhdk

The Hunt for Broken Dependencies - Discussion

"To What Extent Are Compilers Undermining the Linux Kernel Memory Model?"

- **Annotating dependency chains**?
- **`-memory-model=lkmm`** option for compilers?
- **Combating false positives**:
  - **"safe regions"** via **pragmas** or **function attributes**?
  - **Exclusion lists**?
- **Suggesting fixes** to the user? Automatically **insert barriers**?



Get in touch!

twitter.com/pbhdk

Paul.Heidekrueger@in.tum.de

linkedin.com/in/pbhdk