DEPT DEPendency Tracker

Deadlock Detection Tool

Byungchul Park

max.byungchul.park@gmail.com

Introduction

Lockdep

DEPT

Benefits

Introduction

Lockdep

DEPT

Benefits

CONTEXT X

```
// Holds A.
spin_lock A
...
// Holds B while still holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

CONTEXT Y

```
// Holds B.
spin_lock B
...
// Holds A while still holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

CONTEXT X

```
// Holds A.
spin_lock A
...
// Holds B while still holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

CONTEXT Y

```
// Holds B.
spin_lock B
...
// Holds A while still holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

Need a (runtime) checker tool to check lock acquisition order before deadlocks happen.

Introduction

Lockdep

DEPT

Benefits

Lockdep Lock Dependency

By Ingo Molar

Define Dependency

Lock A

Lock B (while still holding A)

= A depends on B. (A → B)

CONTEXT X

```
// Holds A.
spin_lock A
...
// Holds B while still holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

CONTEXT Y

```
// Holds B.
spin_lock B
...
// Holds A while still holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

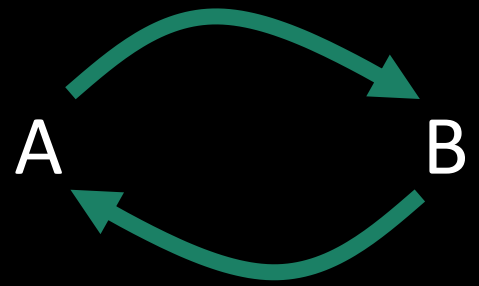
Build Dependency Graph

CONTEXT X

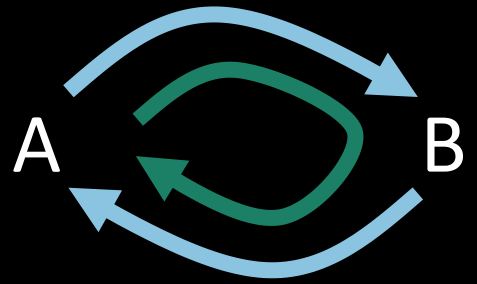
```
// Holds A.
spin_lock A
...
// Holds B while still holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

CONTEXT Y

```
// Holds B.
spin_lock B
...
// Holds A while still holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

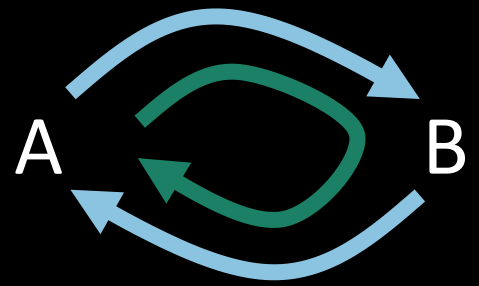


Build Dependency Graph

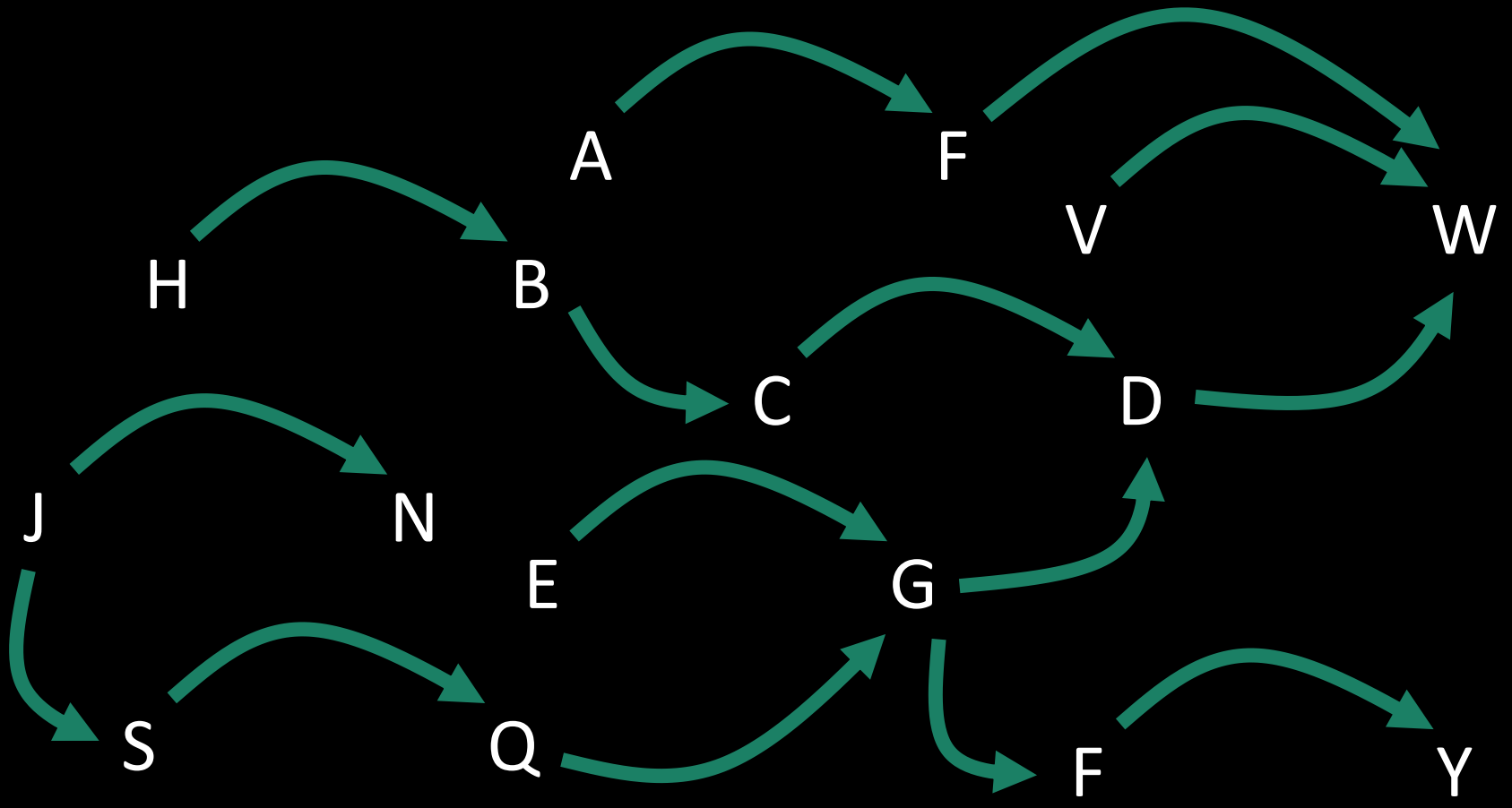


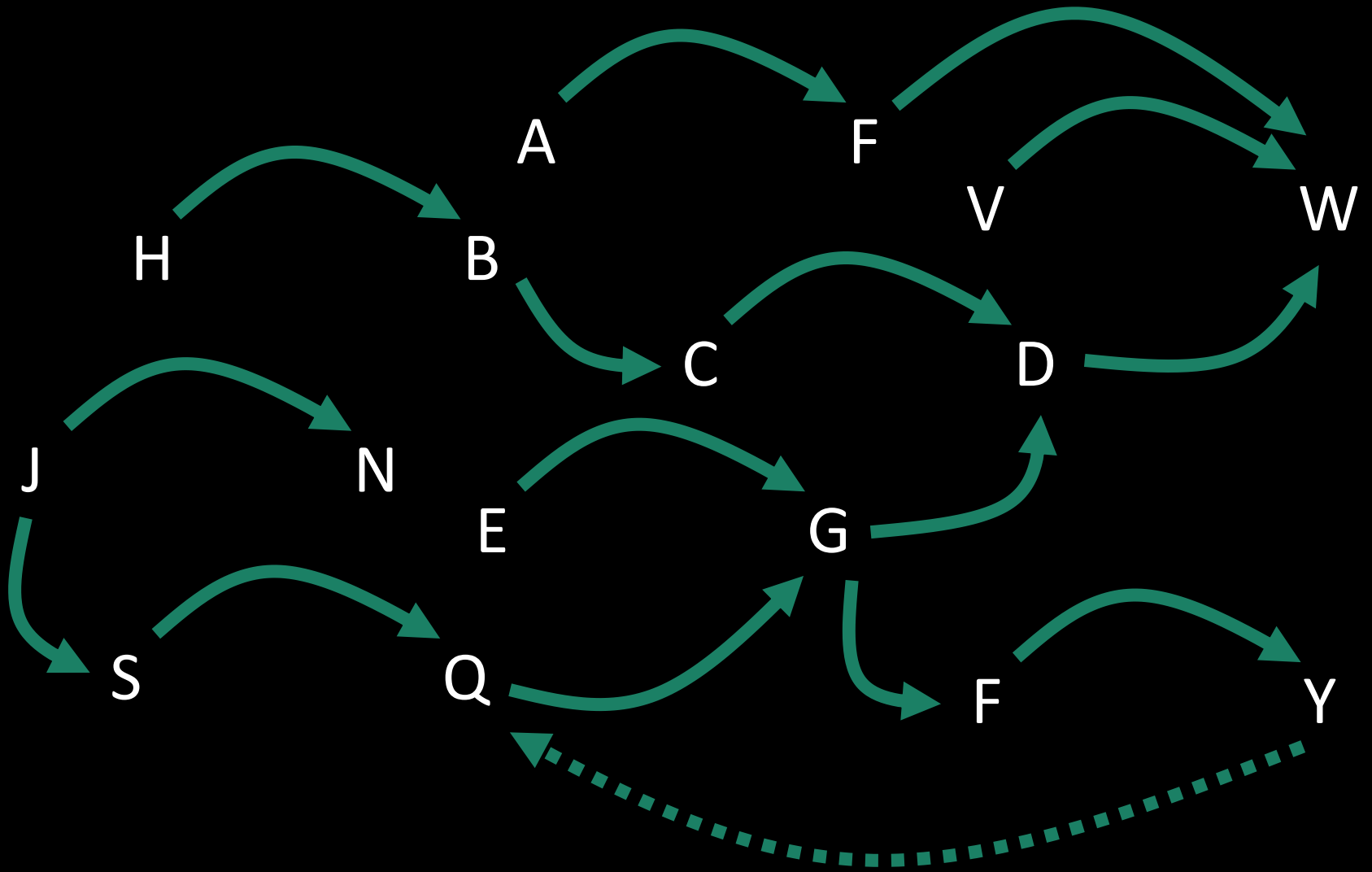
Circular Dependency

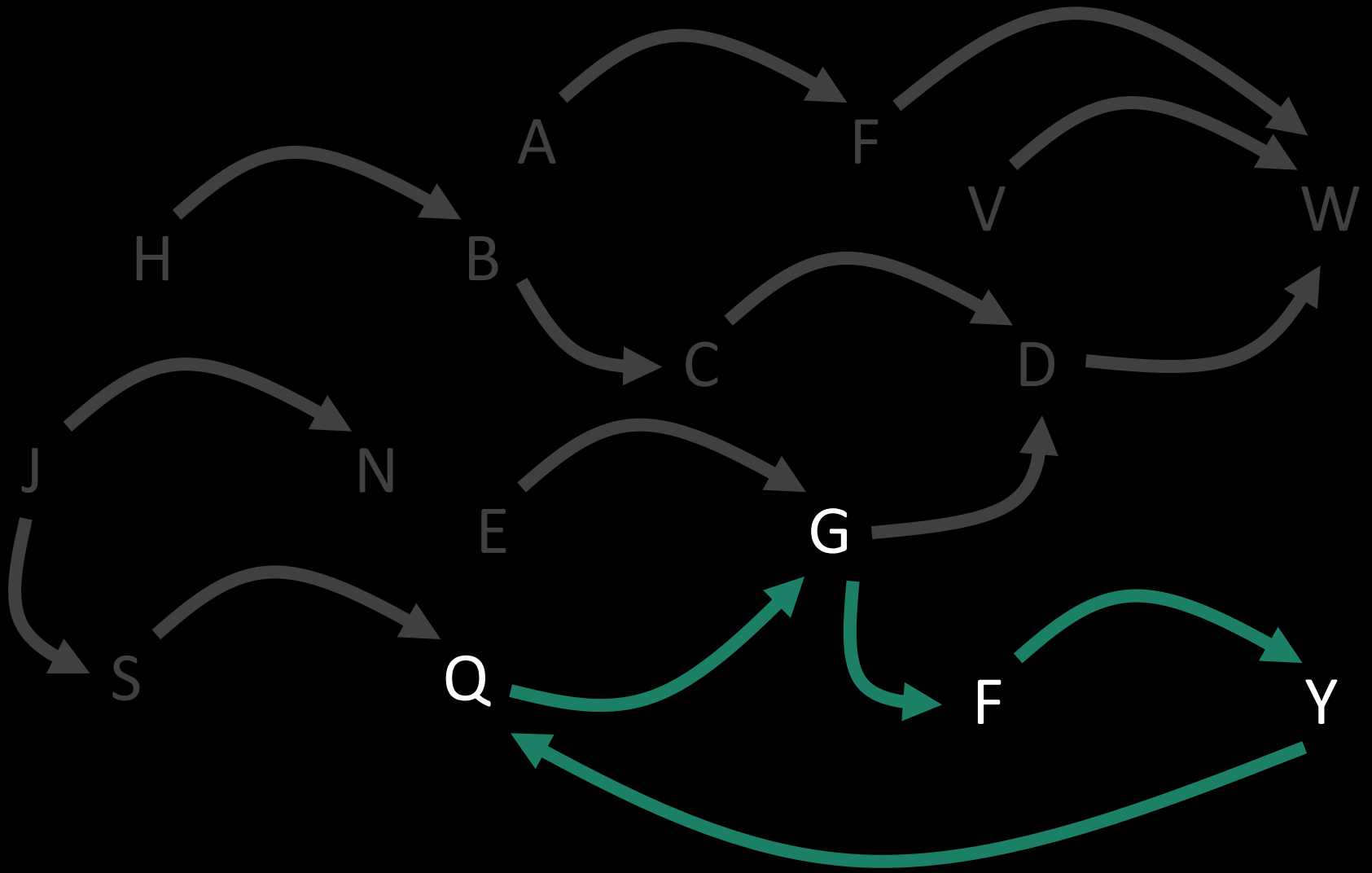
Report it !

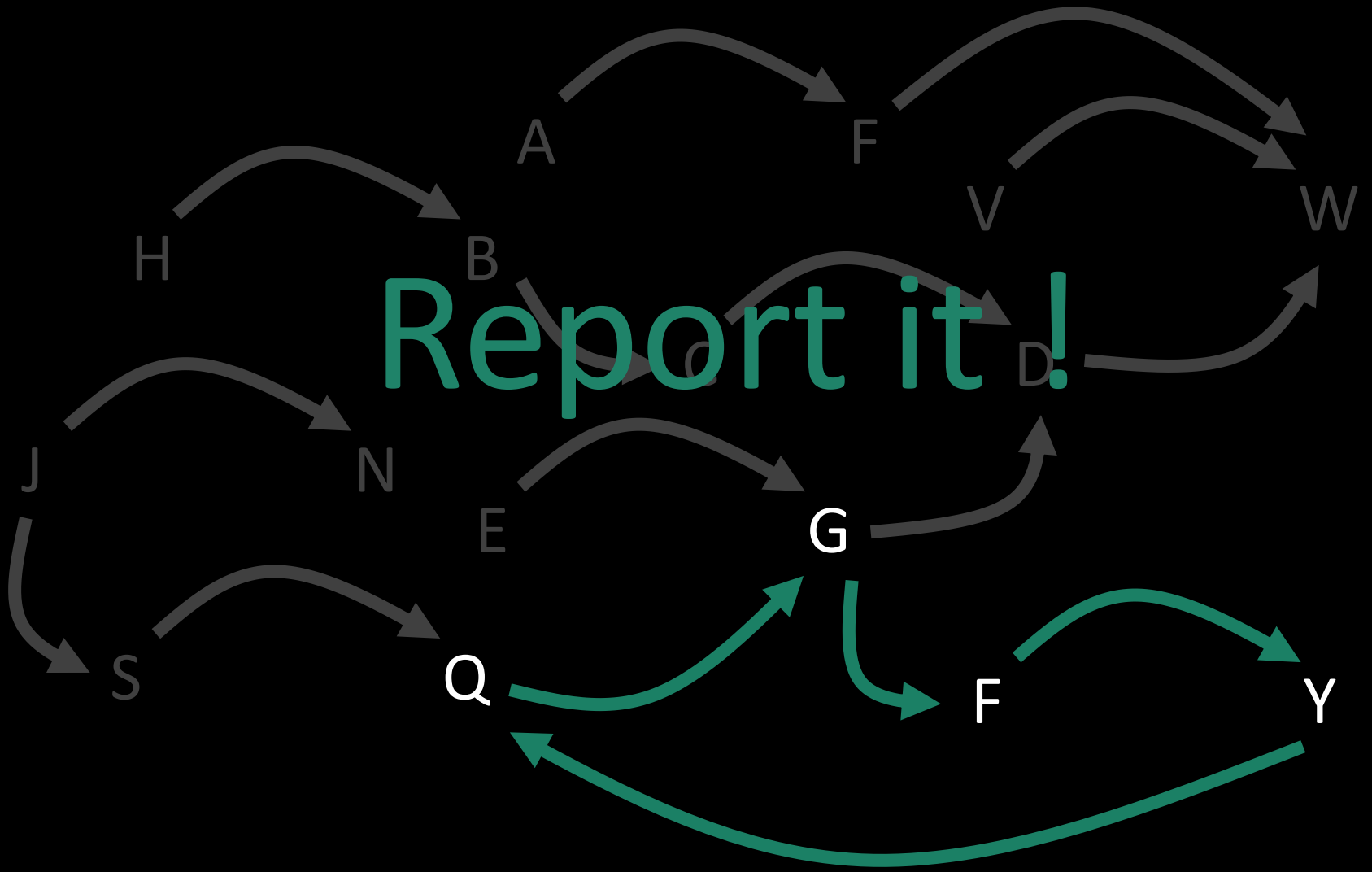


Circular Dependency


A
F
V
W
H
B
C
D
J
N
E
G
S
Q
F
Y


A
F
V
W
H
B
C
D
J
N
E
G
S
Q
F
Y


A
F
V
W
H
B
C
D
J
N
E
G
S
Q
F
Y


A
F
V
W
H
B
C
D
Report it !
J
N
E
G
S
Q
F
Y

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

Define Dependency

Lock A

Lock B (while still holding A)

= A depends on B. (A → B)

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

CONTEXT X

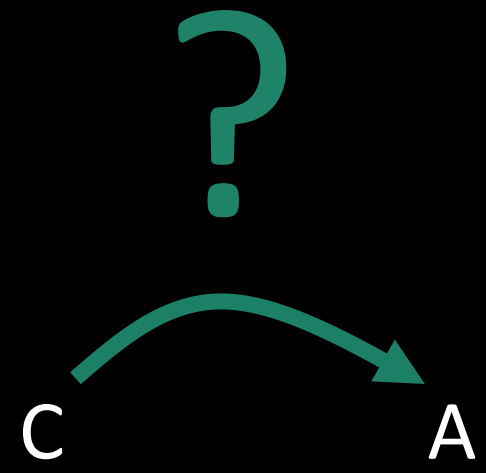
```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```



Build Dependency Graph

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

CONTEXT X

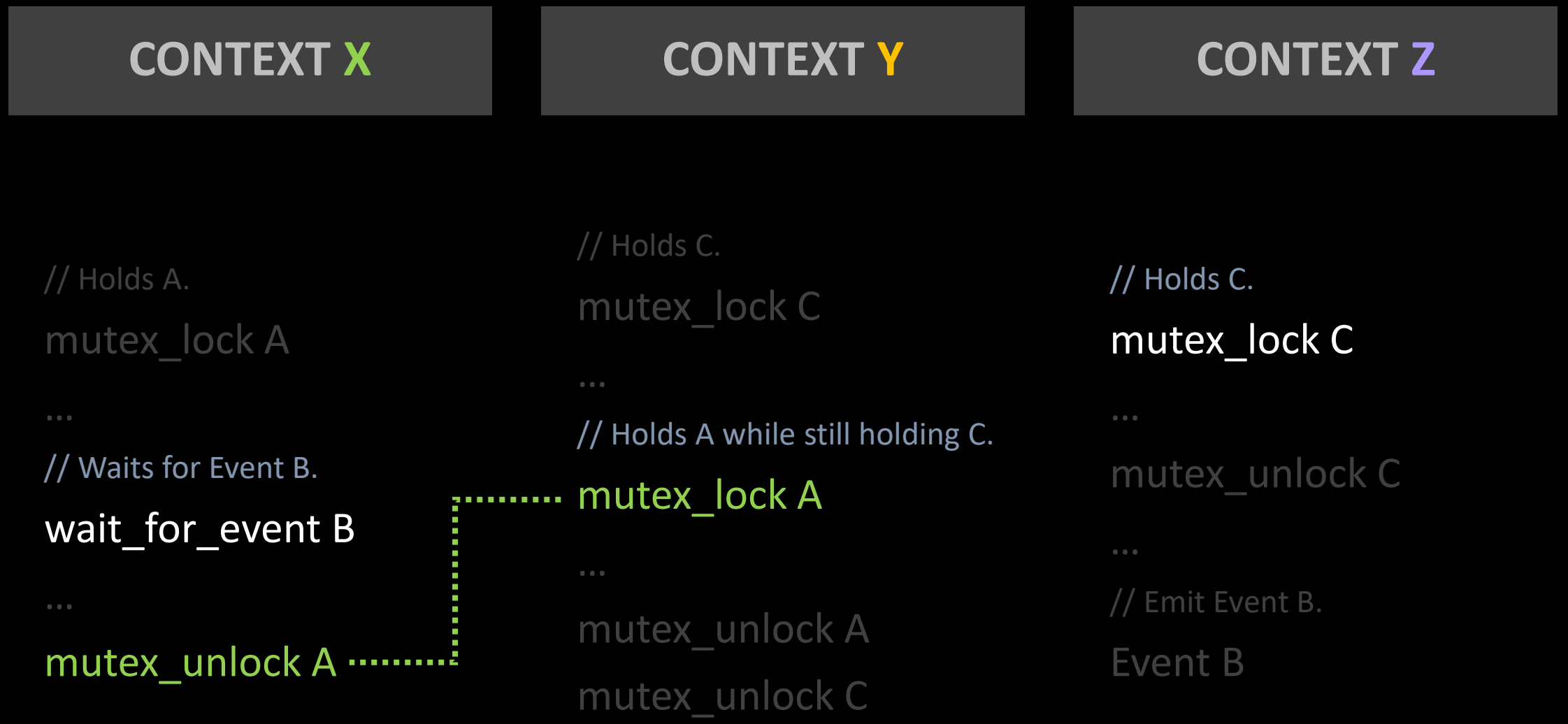
```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```


CONTEXT X
CONTEXT Y
CONTEXT Z
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

Not only typical locks but also waits and events can lead to deadlocks.

Introduction

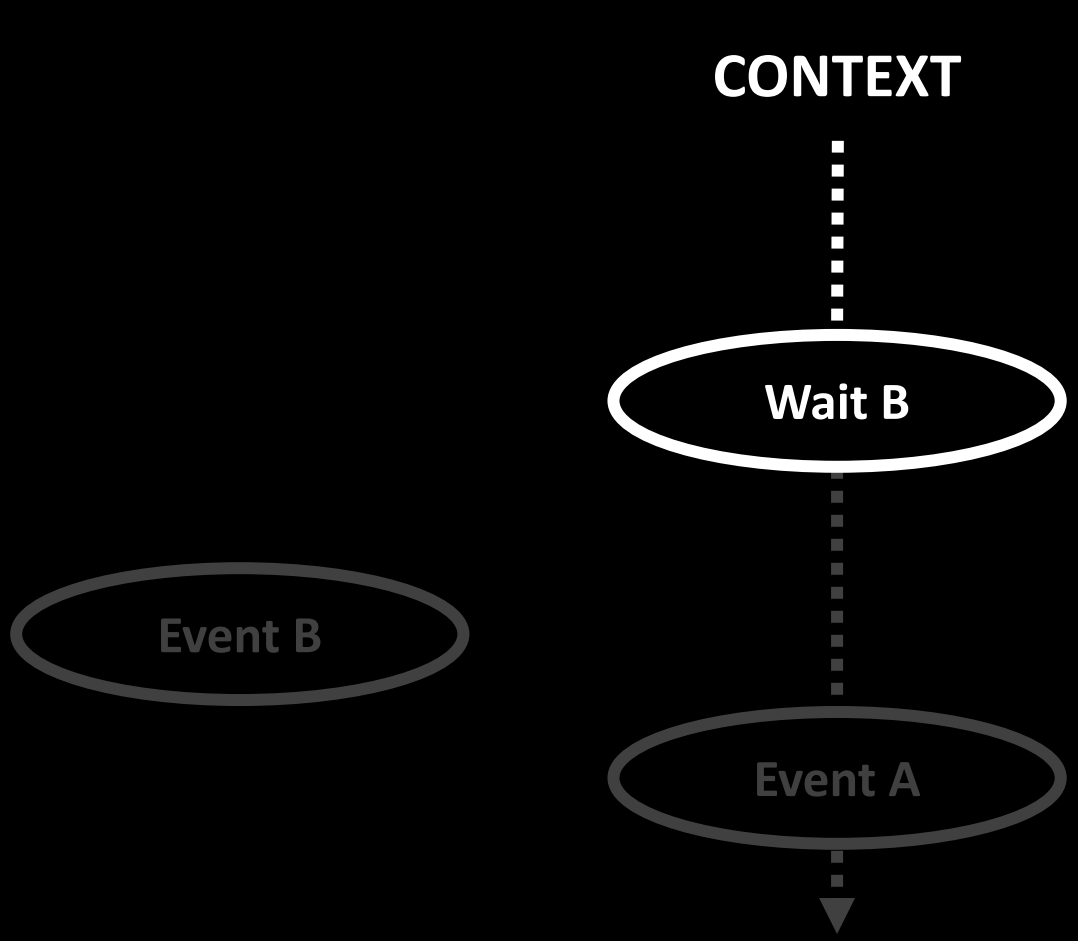
Lockdep

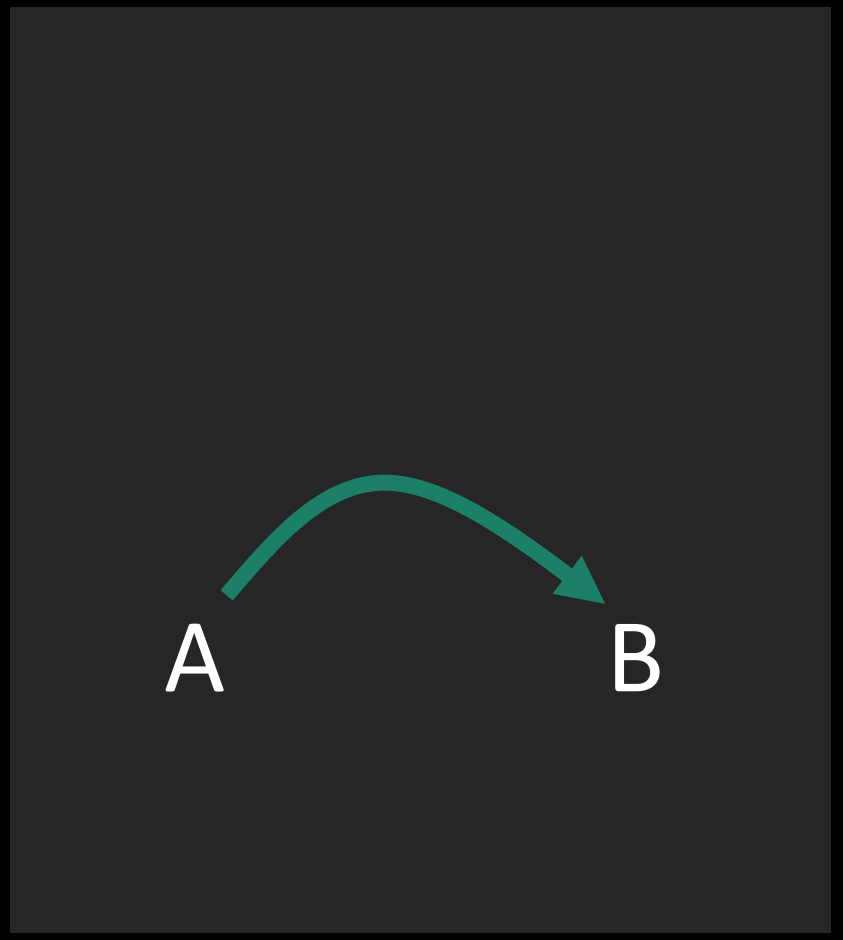
DEPT

Benefits

DEPT Dependency Tracker

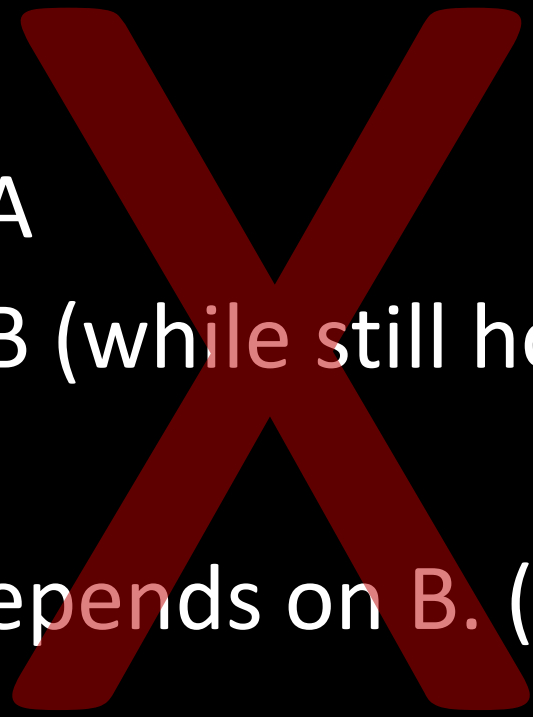
By Byungchul Park





Event A occurrence depends on Event B occurrence.

Define Dependency



Lock A

Lock B (while still holding A)

= A depends on B. (A → B)

Redefine Dependency

Event A occurrence depends on
Event B occurrence.

= A depends on B. (A → B)

CONTEXT X

```
// Holds A.
spin_lock A
...
// Holds B while still holding A.
spin_lock B
...
spin_unlock B
spin_unlock A
```

CONTEXT Y

```
// Holds B.
spin_lock B
...
// Holds A while still holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
```

CONTEXT X

```
spin_lock A
...
// Waits for Event B.
spin_lock B
...
spin_unlock B
// Emit Event A.
spin_unlock A
```

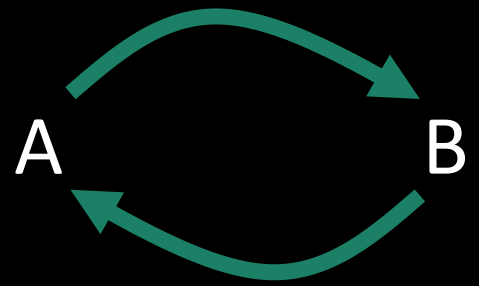
CONTEXT Y

```
spin_lock B
...
// Waits for Event A.
spin_lock A
...
spin_unlock A
// Emit Event B.
spin_unlock B
```

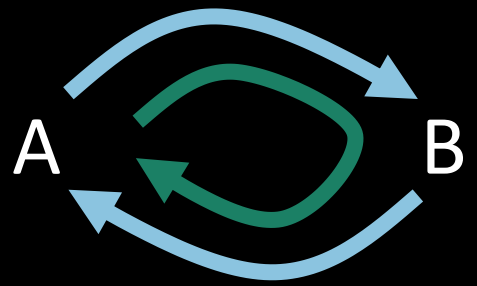
CONTEXT X	CONTEXT Y
spin_lock A	spin_lock B
...	...
// Waits for Event B.	// Waits for Event A.
spin_lock B	spin_lock A
...	...
spin_unlock B	spin_unlock A
// Emit Event A.	// Emit Event B.
spin_unlock A	spin_unlock B

Build Dependency Graph

CONTEXT X	CONTEXT Y
spin_lock A	spin_lock B
...	...
// Waits for Event B.	// Waits for Event A.
spin_lock B	spin_lock A
...	...
spin_unlock B	spin_unlock A
// Emit Event A.	// Emit Event B.
spin_unlock A	spin_unlock B

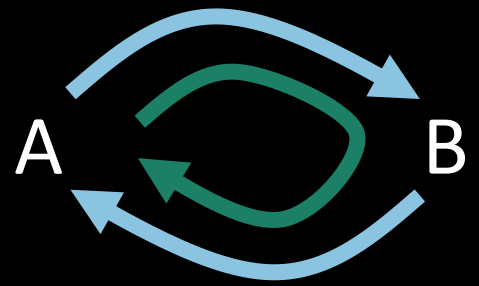


Build Dependency Graph



Circular Dependency

Report it !



Circular Dependency

CONTEXT X

```
// Holds A.
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
mutex_unlock A
```

CONTEXT Y

```
// Holds C.
mutex_lock C
...
// Holds A while still holding C.
mutex_lock A
...
mutex_unlock A
mutex_unlock C
```

CONTEXT Z

```
// Holds C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

CONTEXT X	CONTEXT Y	CONTEXT Z
mutex_lock A	mutex_lock C	// Waits for Event C.
...	...	mutex_lock C
// Waits for Event B.	// Waits for Event A.	...
wait_for_event B	mutex_lock A	mutex_unlock C
...	...	...
// Emit Event A.	mutex_unlock A	// Emit Event B.
mutex_unlock A	// Emit Event C.	Event B
	mutex_unlock C	

CONTEXT X

```
mutex_lock A
...
// Waits for Event B.
wait_for_event B
...
// Emit Event A.
mutex_unlock A
```

CONTEXT Y

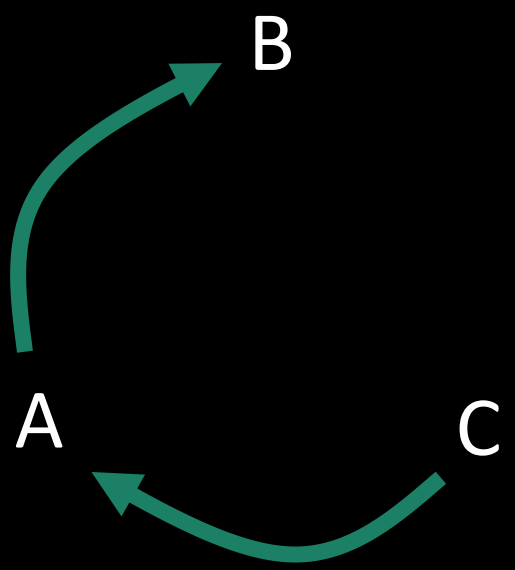
```
mutex_lock C
...
// Waits for Event A.
mutex_lock A
...
mutex_unlock A
// Emit Event C.
mutex_unlock C
```

CONTEXT Z

```
// Waits for Event C.
mutex_lock C
...
mutex_unlock C
...
// Emit Event B.
Event B
```

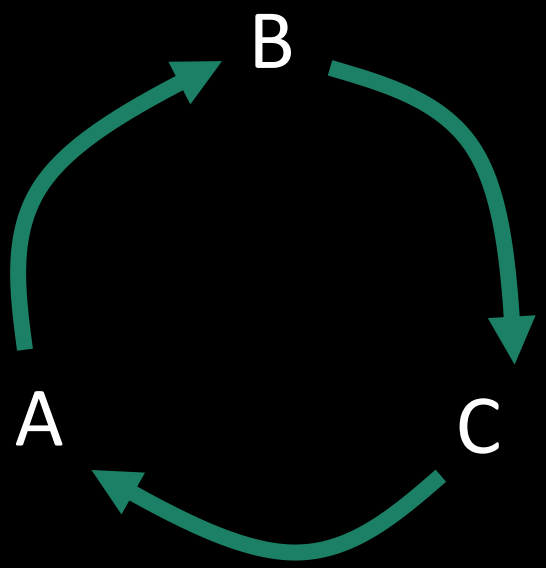
Build Dependency Graph

CONTEXT X	CONTEXT Y	CONTEXT Z
mutex_lock A	mutex_lock C	// Waits for Event C.
...	...	mutex_lock C
// Waits for Event B.	// Waits for Event A.	...
wait_for_event B	mutex_lock A	mutex_unlock C
...	...	...
// Emit Event A.	mutex_unlock A	// Emit Event B.
mutex_unlock A	// Emit Event C.	Event B
	mutex_unlock C	

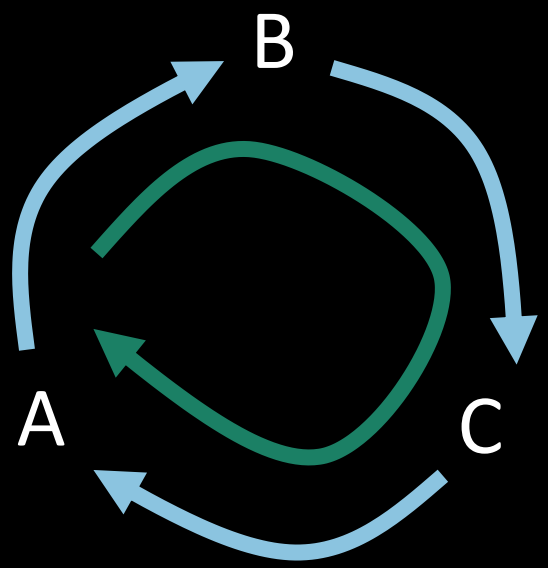


Build Dependency Graph

CONTEXT X	CONTEXT Y	CONTEXT Z
mutex_lock A	mutex_lock C	// Waits for Event C.
...	...	mutex_lock C
// Waits for Event B.	// Waits for Event A.	...
wait_for_event B	mutex_lock A	mutex_unlock C
...	...	...
// Emit Event A.	mutex_unlock A	// Emit Event B.
mutex_unlock A	// Emit Event C.	Event B
	mutex_unlock C	

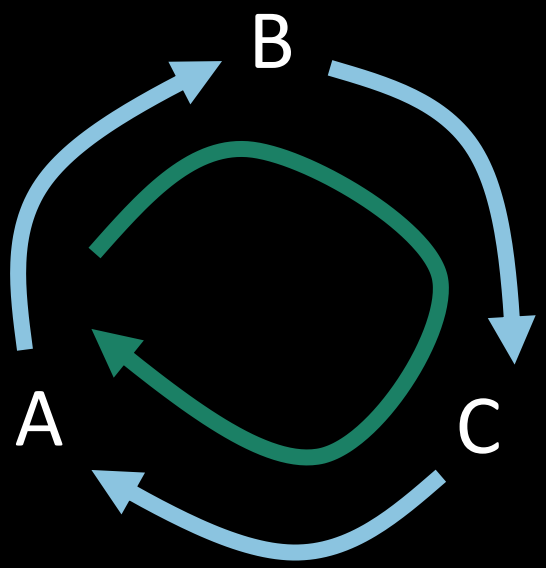


Build Dependency Graph

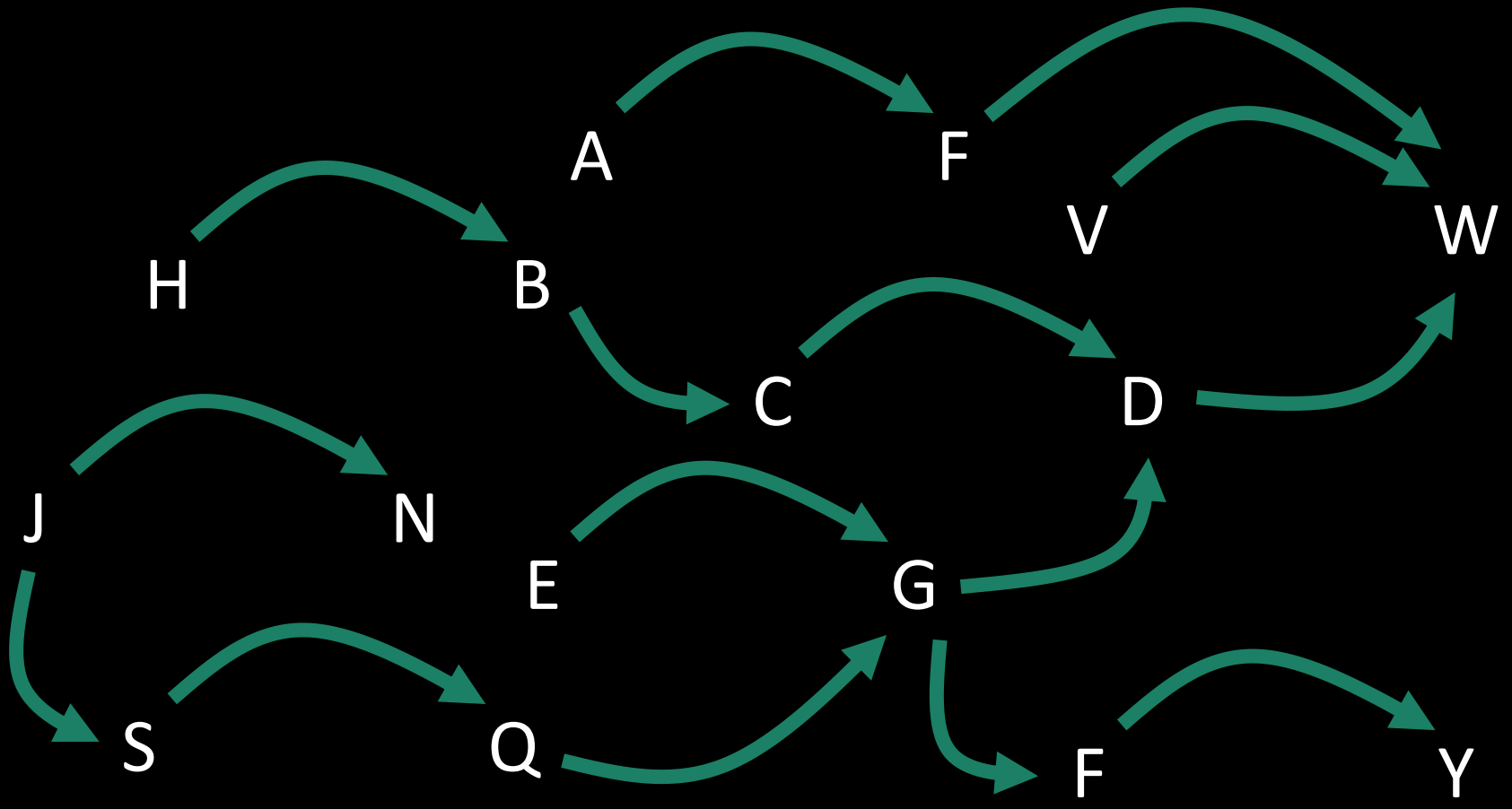


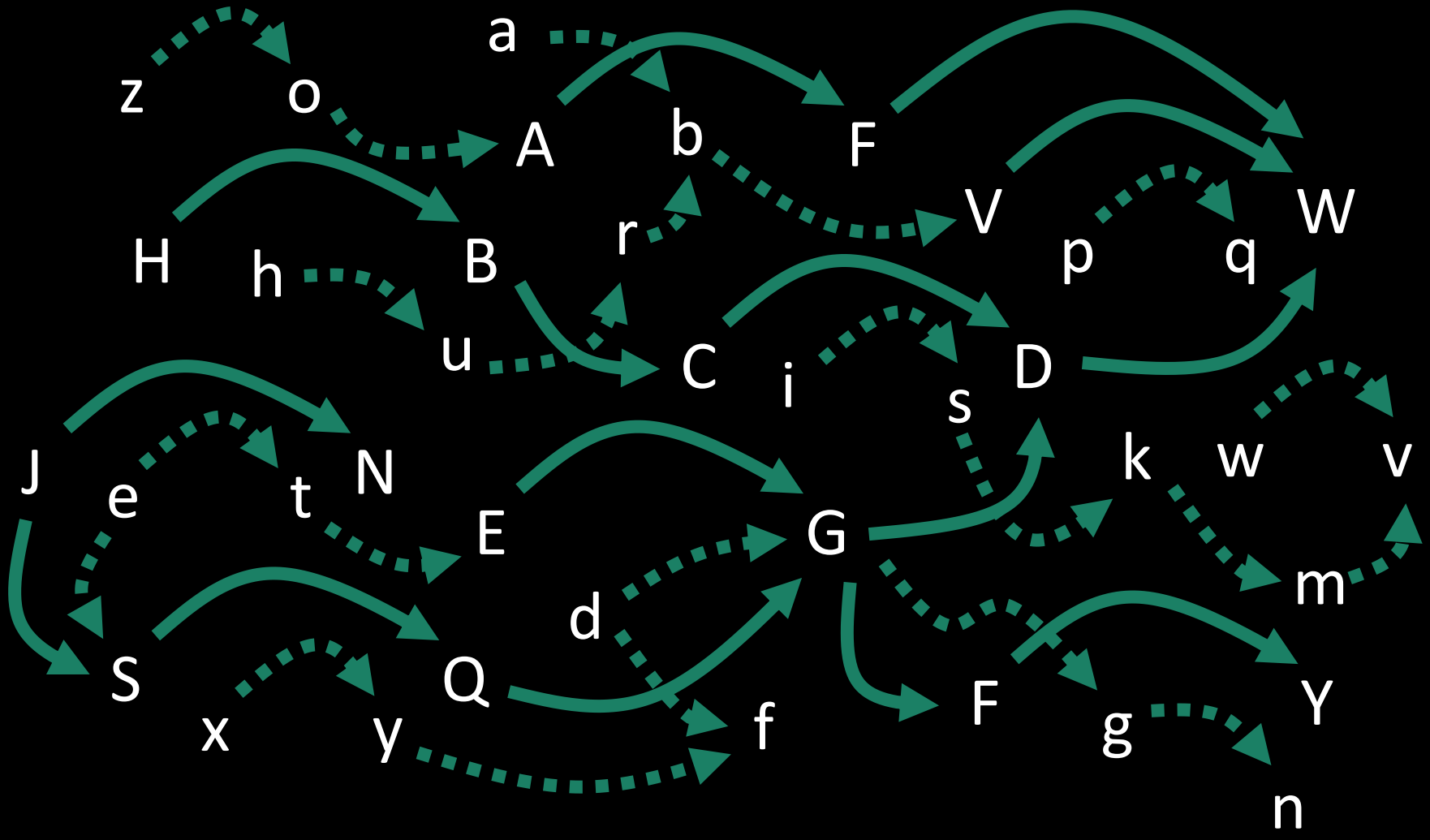
Circular Dependency

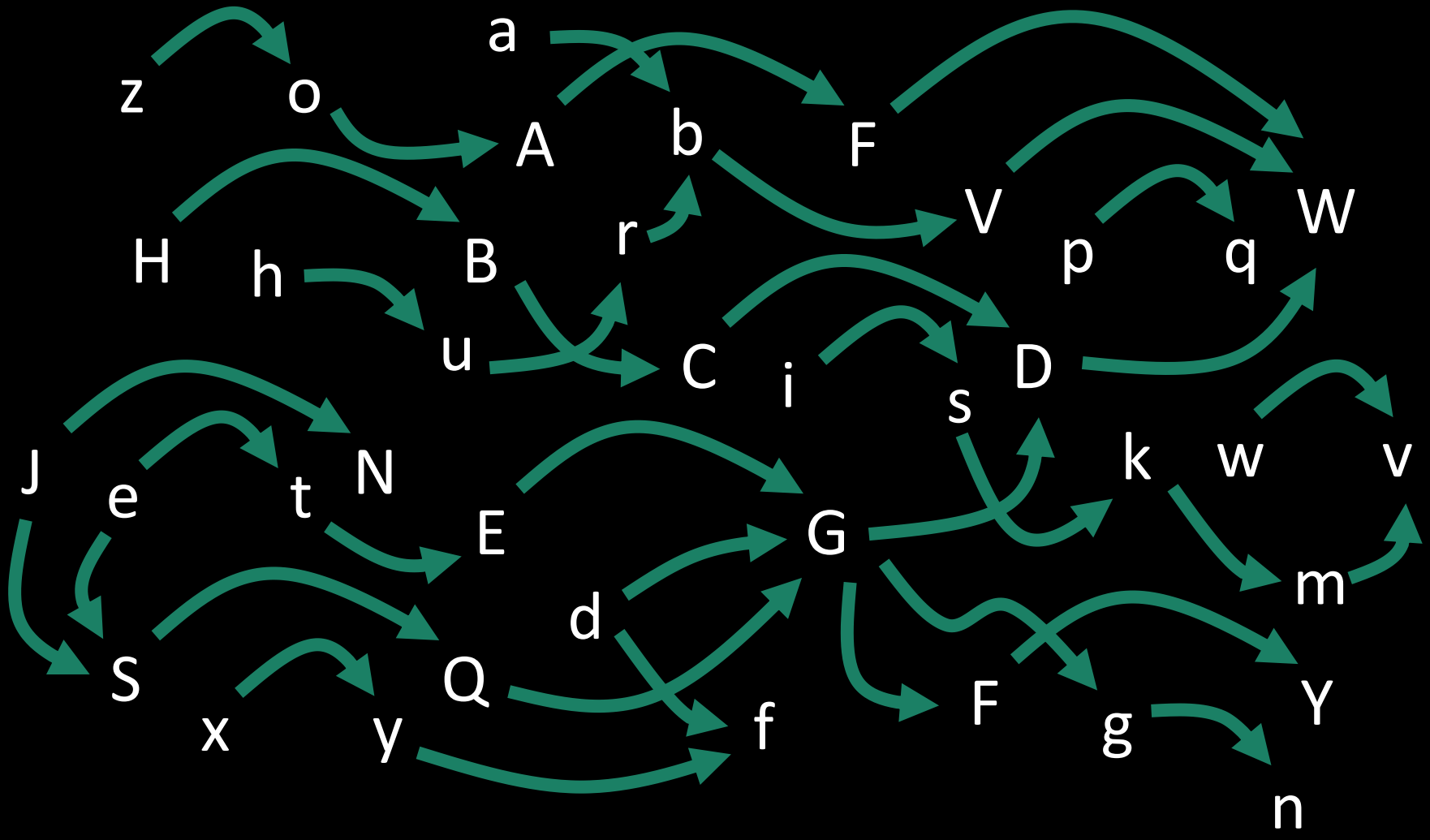
Report it !

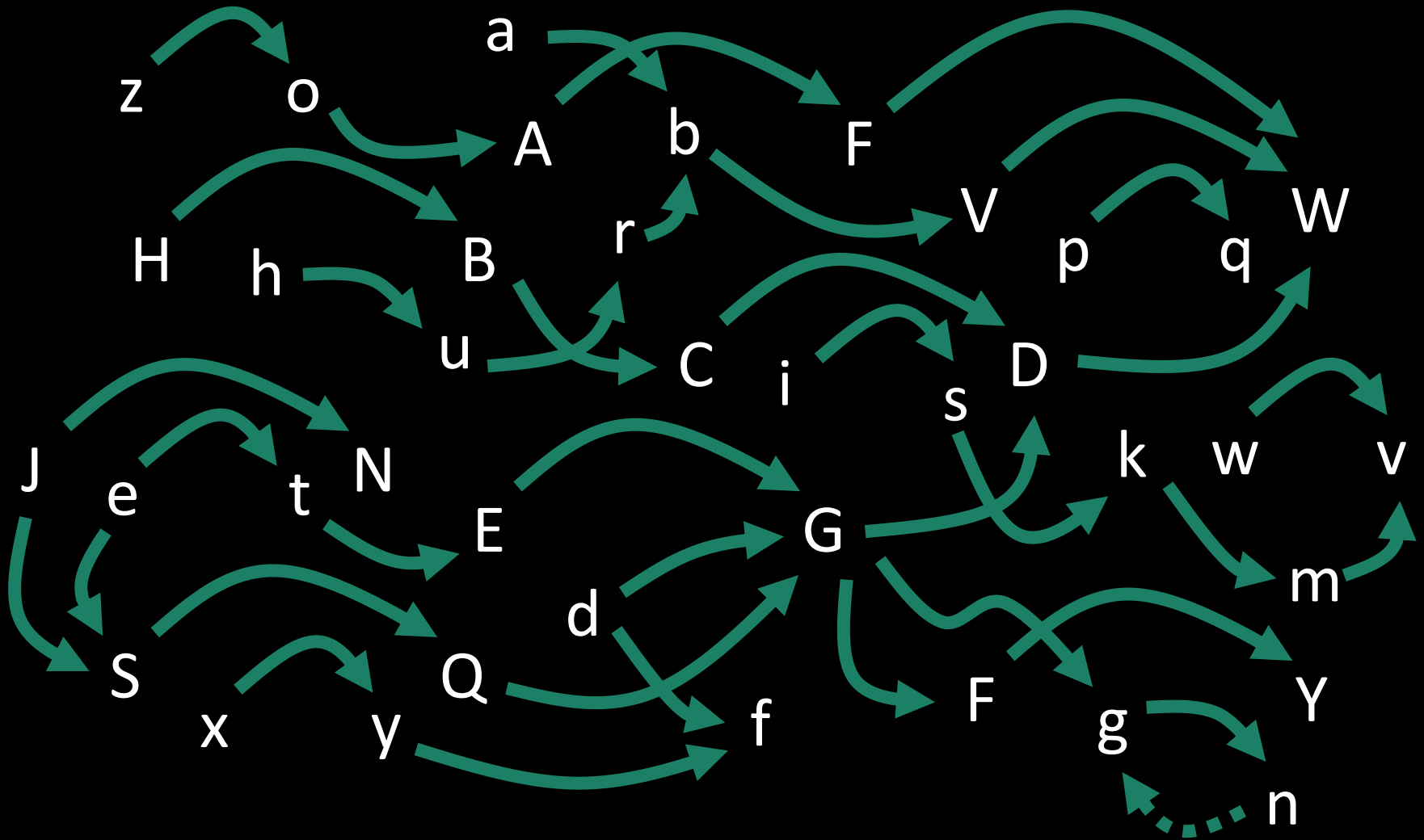


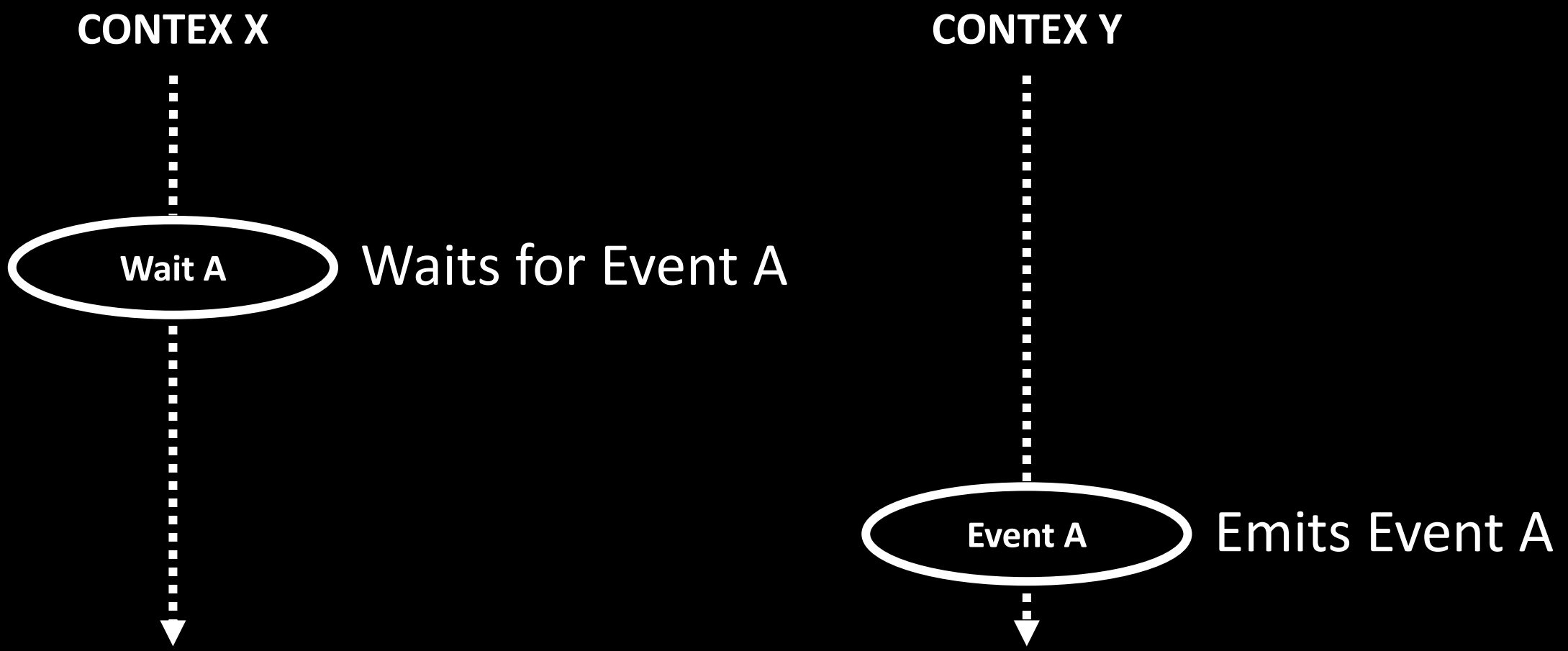
Circular Dependency

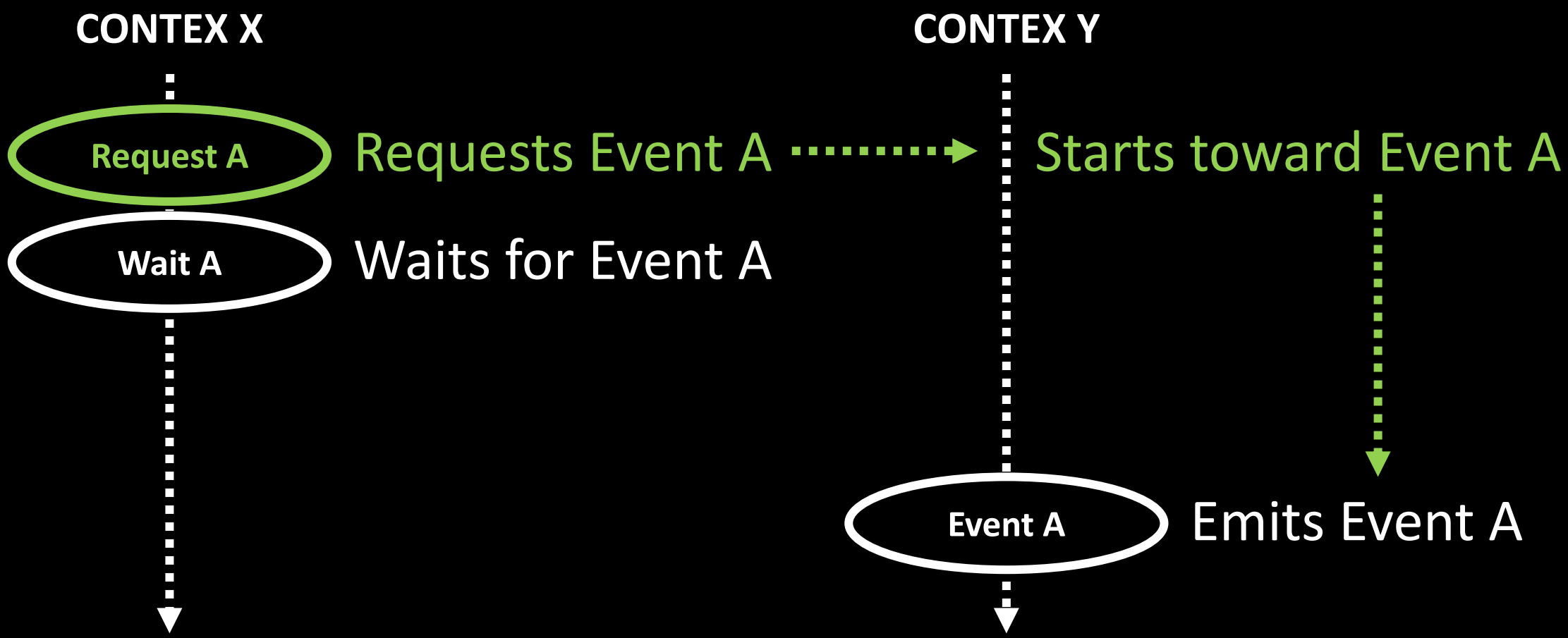

A
F
V
W
H
B
C
D
J
N
E
G
S
Q
F
Y

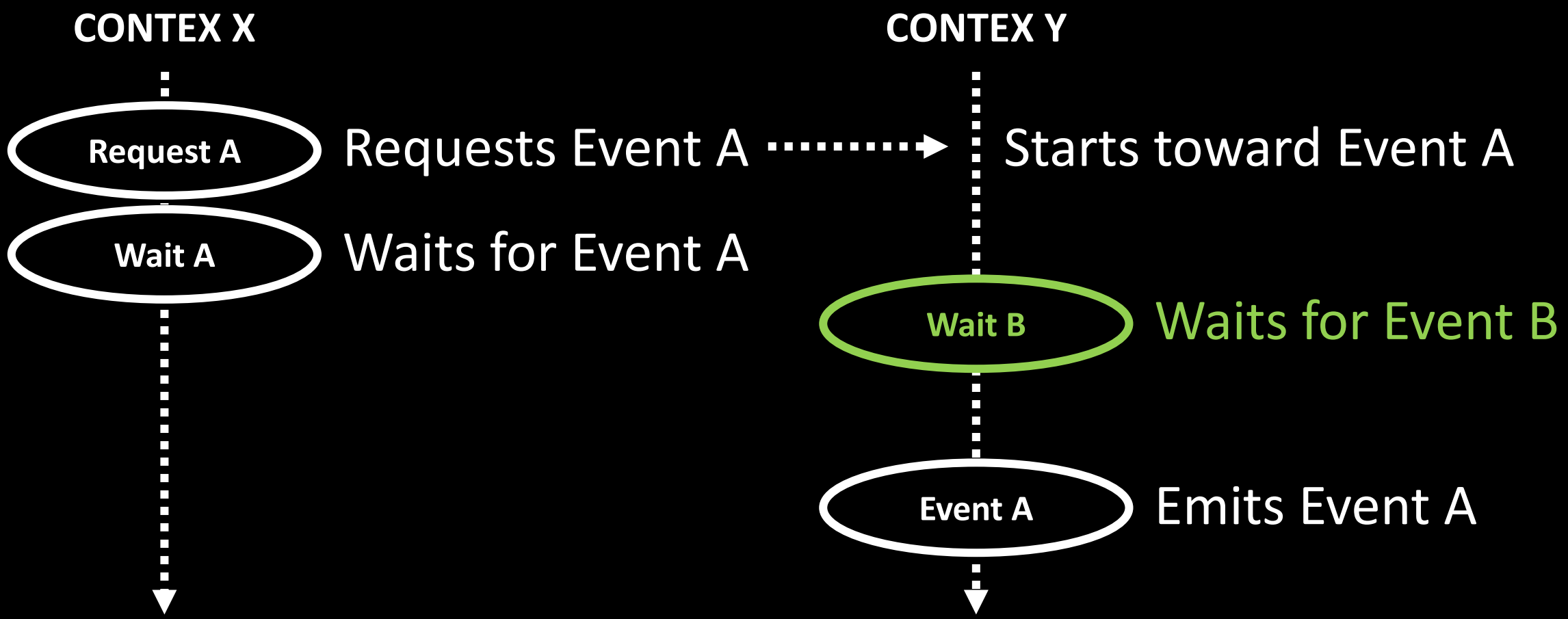

z
o
a
A
b
F
V
p
q
W
H
h
B
r
u
C
i
D
s
J
e
t
N
E
G
k
w
v
m
d
S
Q
x
y
f
F
g
Y
n

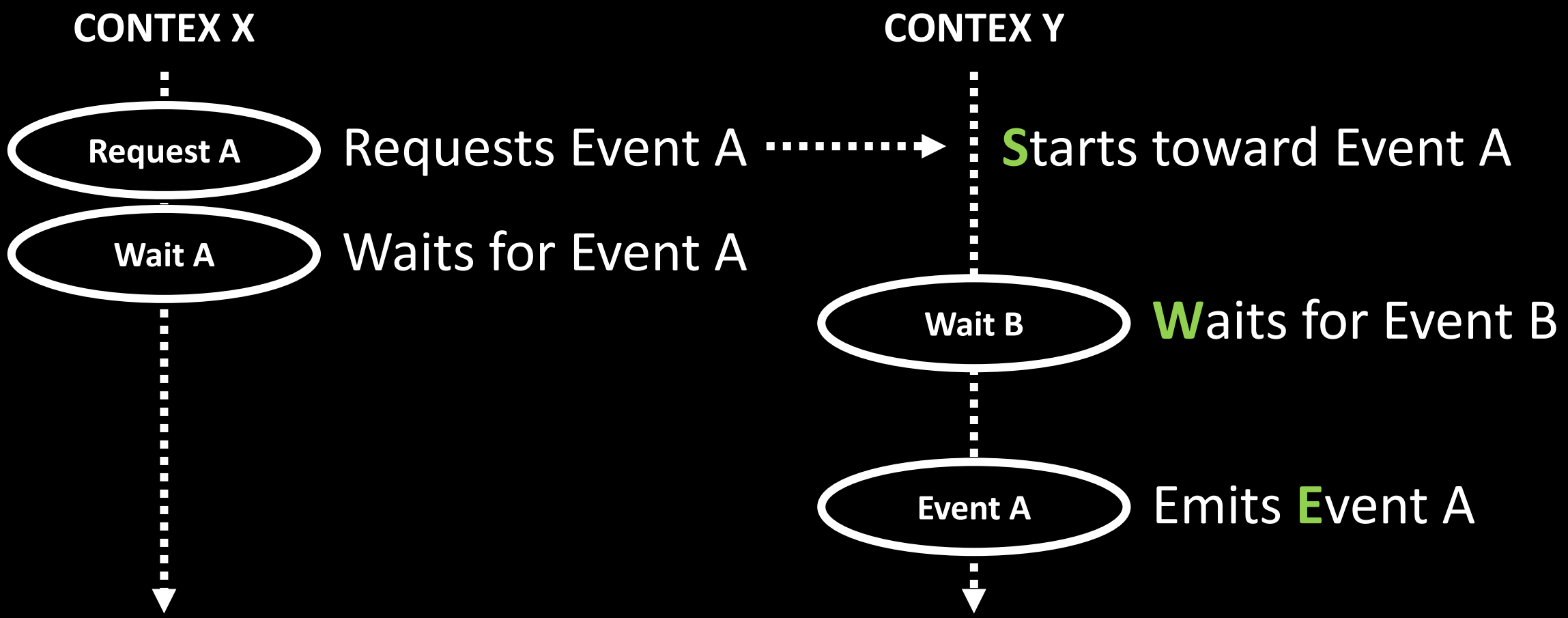

z
o
a
A
b
F
V
W
H
B
r
p
q
h
u
C
i
D
s
J
e
t
N
E
G
k
w
v
m
d
S
Q
F
Y
x
y
f
g
n

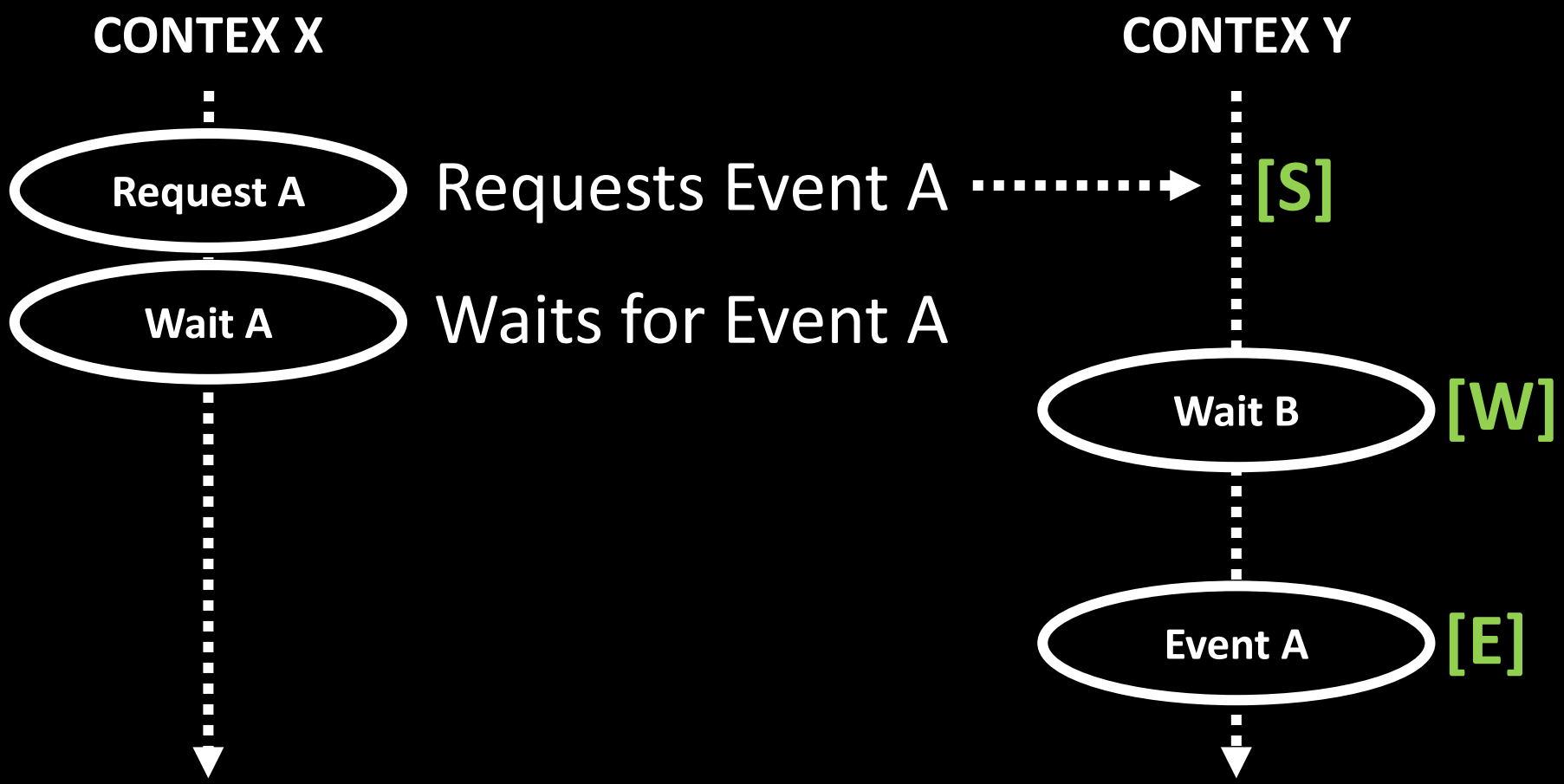

z
o
a
A
b
F
V
W
H
h
B
r
p
q
u
C
i
D
s
J
e
t
N
E
G
k
w
v
m
d
S
Q
F
Y
x
y
f
g
n

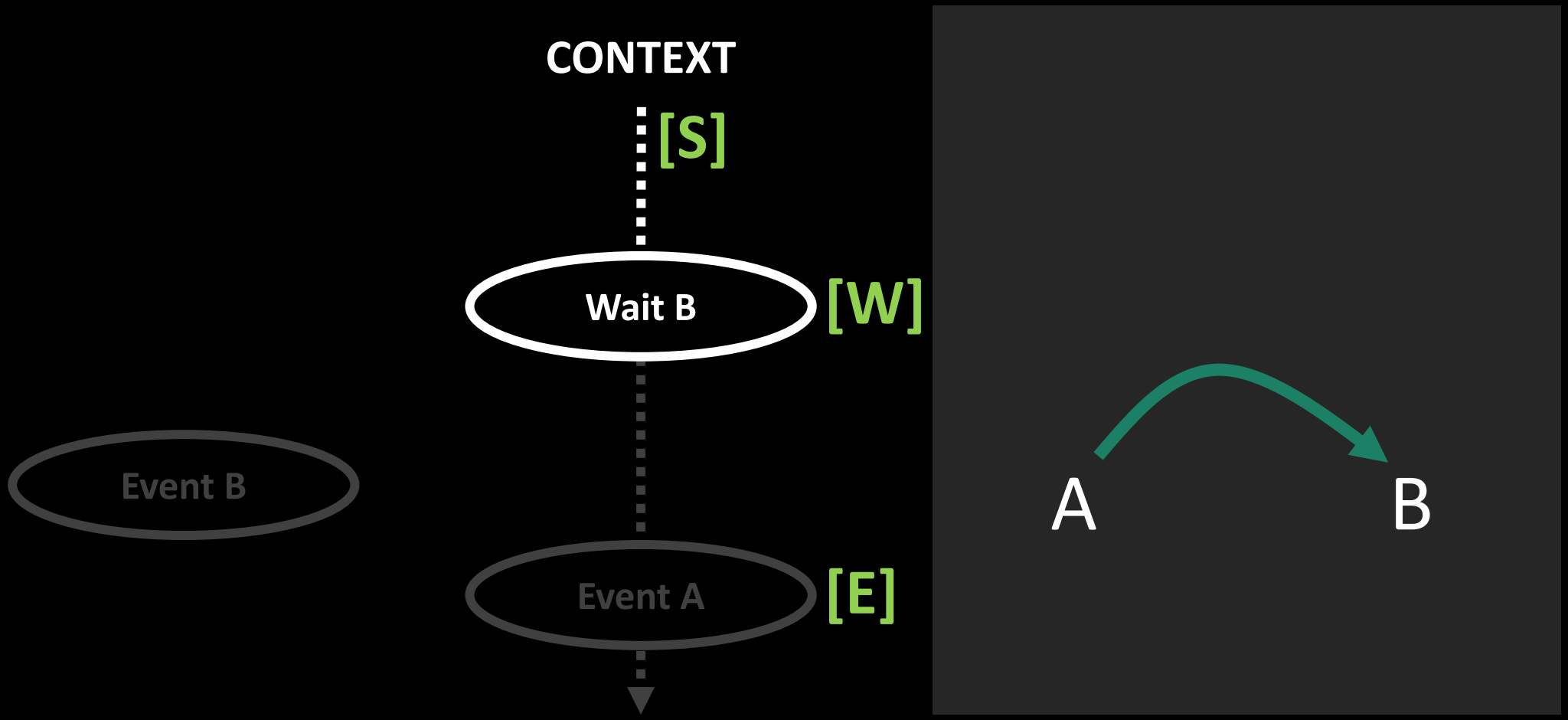

CONTEX X
Wait A
Waits for Event A
CONTEX Y
Event A
Emits Event A


CONTEX X
CONTEX Y
Request A
Requests Event A
Starts toward Event A
Wait A
Waits for Event A
Event A
Emits Event A


CONTEX X
CONTEX Y
Request A
Requests Event A
Starts toward Event A
Wait A
Waits for Event A
Wait B
Waits for Event B
Event A
Emits Event A


CONTEX X
CONTEX Y
Request A
Requests Event A
Starts toward Event A
Wait A
Waits for Event A
Wait B
Waits for Event B
Event A
Emits Event A


CONTEX X
CONTEX Y
Request A
Requests Event A
[S]
Wait A
Waits for Event A
Wait B
[W]
Event A
[E]



Event A occurrence depends on Event B occurrence.

https://lore.kernel.org/lkml/6383cde5-cf4b-facf-6e07-1378a485657d@I-love.SAKURA.ne.jp/

…this is a deadlock between PG_locked bit and ni_lock lock.

filemap_update_page() calls filemap_read_folio() after calling folio_trylock(). Since folio_trylock() sets PG_locked bit,

```
mutex_lock_nested+0x17/0x20 kernel/locking/mutex.c:799
ni_lock fs/ntfs3/ntfs_fs.h:1122 [inline]
attr_data_get_block+0x4a6/0x2e40 fs/ntfs3/attrib.c:919
ntfs_get_block_vbo+0x374/0xd20 fs/ntfs3/inode.c:573
do_mpage_readpage+0x98b/0x1bb0 fs/mpage.c:208
mpage_read_folio+0x103/0x1d0 fs/mpage.c:379
filemap_read_folio+0x1ba/0x7f0 mm/filemap.c:2426
filemap_update_page+0x3ca/0x550 mm/filemap.c:2511
filemap_get_pages+0x8d8/0x1110 mm/filemap.c:2624
filemap_read+0x3e7/0xee0 mm/filemap.c:2694
```

is trying to take ni_lock after setting PG_locked bit.

On the other hand, folio_lock() waits until PG_locked bit is cleared, but unfortunately ntfs3_setattr() already took ni_lock before calling folio_lock().

```
io_schedule+0x83/0x100 kernel/sched/core.c:8811
folio_wait_bit_common+0x8ca/0x1390 mm/filemap.c:1297
folio_lock include/linux/pagemap.h:938 [inline]
truncate_inode_pages_range+0xc8d/0x1650 mm/truncate.c:421
truncate_inode_pages mm/truncate.c:448 [inline]
truncate_pagecache mm/truncate.c:743 [inline]
truncate_setsize+0xcb/0xf0 mm/truncate.c:768
ntfs_truncate fs/ntfs3/file.c:395 [inline]
ntfs3_setattr+0x5a5/0xca0 fs/ntfs3/file.c:696
```

Since no Lockdep annotation is used for e.g. PG_locked bit, this deadlock cannot be detected by Lockdep...

https://lore.kernel.org/lkml/6383cde5-cf4b-facf-6e07-1378a485657d@I-love.SAKURA.ne.jp/

…this is a deadlock between PG_locked bit and ni_lock lock.

filemap_update_page() calls filemap_read_folio() after calling
folio_trylock(). Since folio_trylock() sets PG_locked bit,

```
 mutex_lock_nested+0x17/0x20 kernel/locking/mutex.c:799
 ni_lock fs/ntfs3/ntfs_fs.h:1122 [inline]
 attr_data_get_block+0x4a6/0x2e40 fs/ntfs3/attrib.c:919
 ntfs_get_block_vbo+0x374/0xd20 fs/ntfs3/inode.c:573
 do_mpage_readpage+0x98b/0x1bb0 fs/mpage.c:208
 mpage_read_folio+0x103/0x1d0 fs/mpage.c:379
 filemap_read_folio+0x1ba/0x7f0 mm/filemap.c:2426
 filemap_update_page+0x3ca/0x550 mm/filemap.c:2511
 filemap_get_pages+0x8d8/0x1110 mm/filemap.c:2624
 filemap_read+0x3e7/0xee0 mm/filemap.c:2694
```

is trying to take ni_lock after setting PG_locked bit.

On the other hand, folio_lock() waits until PG_locked bit is cleared,
but unfortunately ntfs3_setattr() already took ni_lock before calling
folio_lock().

```
 io_schedule+0x83/0x100 kernel/sched/core.c:8811
 folio_wait_bit_common+0x8ca/0x1390 mm/filemap.c:1297
 folio_lock include/linux/pagemap.h:938 [inline]
 truncate_inode_pages_range+0xc8d/0x1650 mm/truncate.c:421
 truncate_inode_pages mm/truncate.c:448 [inline]
 truncate_pagecache mm/truncate.c:743 [inline]
 truncate_setsize+0xcb/0xf0 mm/truncate.c:768
 ntfs_truncate fs/ntfs3/file.c:395 [inline]
 ntfs3_setattr+0x5a5/0xca0 fs/ntfs3/file.c:696
```

Since no Lockdep annotation is used for e.g. PG_locked bit, this deadlock
cannot be detected by Lockdep...

https://lore.kernel.org/lkml/1674268856-31807-1-git-send-email-byungchul.park@lge.com/

Hi Torvalds and folks,

I reproduced the issue with DEPT on (after making DEPT work a lil more aggressively for PG_locked), and obtain a DEPT report. I wish this is the true positive, explaining the issue coSRectly!

Let me remind you guys again, "DEPT is designed exactly for that kind of deadlock issue by e.g. PG_locked, PG_writeback and any wait APIs".

I attach the report and add how to interpret it at the end.

```
[  227.854322] ==================================================
[  227.854880] DEPT: Circular dependency has been detected.
[  227.855341] 6.2.0-rc1-00025-gb0c20ebf51ac-dirty #28 Not tainted
[  227.855864] ---------------------------------------------------
[  227.856367] Benefits
[  227.856601] ---------------------------------------------------
[  227.857107] *** DEADLOCK ***

[  227.857551] context A
[  227.857803]    [S] lock(&ni->ni_lock:0)
[  227.858175]    [W] folio_wait_bit_common(PG_locked_map:0)
[  227.858658]    [E] unlock(&ni->ni_lock:0)

[  227.859233] context B
[  227.859484]    [S] (unknown)(PG_locked_map:0)
[  227.859906]    [W] lock(&ni->ni_lock:0)
[  227.860277]    [E] folio_unlock(PG_locked_map:0)

[  227.860883] [S]: start of the event context
[  227.861263] [W]: the wait blocked
[  227.861581] [E]: the event not reachable
[  227.861941] ---------------------------------------------------
[  227.862436] context A's detail
[  227.862738] ---------------------------------------------------
[  227.863242] context A
[  227.863490]    [S] lock(&ni->ni_lock:0)
[  227.863865]    [W] folio_wait_bit_common(PG_locked_map:0)
[  227.864356]    [E] unlock(&ni->ni_lock:0)

[  227.864929] [S] lock(&ni->ni_lock:0):
[  227.865279] [<ffffffff82b396fb>] ntfs3_setattr+0x54b/0xd40
[  227.865803] stacktrace:
[  227.866064]    ntfs3_setattr+0x54b/0xd40
[  227.866469]    notify_change+0xcb3/0x1430
[  227.866875]    do_truncate+0x149/0x210
[  227.867277]    path_openat+0x21a3/0x2a90
[  227.867692]    do_filp_open+0x1ba/0x410
[  227.868110]    do_sys_openat2+0x16d/0x4e0
[  227.868520]    __x64_sys_creat+0xcd/0x120
[  227.868925]    do_syscall_64+0x41/0xc0
[  227.869322]    entry_SYSCALL_64_after_hwframe+0x63/0xcd

[  227.870019] [W] folio_wait_bit_common(PG_locked_map:0):
[  227.870491] [<ffffffff81b228b0>] truncate_inode_pages_range+0x9b0/0xf20
[  227.871074] stacktrace:
[  227.871335]    folio_wait_bit_common+0x5e0/0xaf0
[  227.871796]    truncate_inode_pages_range+0x9b0/0xf20
[  227.872287]    truncate_pagecache+0x67/0x90
[  227.872730]    ntfs3_setattr+0x55a/0xd40
[  227.873152]    notify_change+0xcb3/0x1430
[  227.873578]    do_truncate+0x149/0x210
[  227.873981]    path_openat+0x21a3/0x2a90
[  227.874395]    do_filp_open+0x1ba/0x410
[  227.874803]    do_sys_openat2+0x16d/0x4e0
[  227.875215]    __x64_sys_creat+0xcd/0x120
[  227.875623]    do_syscall_64+0x41/0xc0
[  227.876035]    entry_SYSCALL_64_after_hwframe+0x63/0xcd

[  227.876738] [E] unlock(&ni->ni_lock:0):
[  227.877105] (N/A)
[  227.877331] ---------------------------------------------------
[  227.877850] context B's detail
[  227.878169] ---------------------------------------------------
[  227.878699] context B
[  227.878956]    [S] (unknown)(PG_locked_map:0)
[  227.879381]    [W] lock(&ni->ni_lock:0)
[  227.879774]    [E] folio_unlock(PG_locked_map:0)

[  227.880429] [S] (unknown)(PG_locked_map:0):
[  227.880825] (N/A)

[  227.881249] [W] lock(&ni->ni_lock:0):
[  227.881607] [<ffffffff82b009ec>] attr_data_get_block+0x32c/0x19f0
[  227.882151] stacktrace:
[  227.882421]    attr_data_get_block+0x32c/0x19f0
[  227.882877]    ntfs_get_block_vbo+0x264/0x1330
[  227.883316]    __block_write_begin_int+0x3bd/0x14b0
[  227.883809]    block_write_begin+0xb9/0x4d0
[  227.884231]    ntfs_write_begin+0x27e/0x480
[  227.884650]    generic_perform_write+0x256/0x570
[  227.885155]    __generic_file_write_iter+0x2ae/0x500
[  227.885658]    ntfs_file_write_iter+0x66d/0x1d70
[  227.886136]    do_iter_readv_writev+0x20b/0x3c0
[  227.886596]    do_iter_write+0x188/0x710
[  227.887015]    vfs_iter_write+0x74/0xa0
[  227.887425]    iter_file_splice_write+0x745/0xc90
[  227.887913]    direct_splice_actor+0x114/0x180
[  227.888364]    splice_direct_to_actor+0x33b/0x8b0
[  227.888831]    do_splice_direct+0x1b7/0x280
[  227.889256]    do_sendfile+0xb49/0x1310

[  227.889854] [E] folio_unlock(PG_locked_map:0):
[  227.890265] [<ffffffff81f10222>] generic_write_end+0xf2/0x440
[  227.890788] stacktrace:
[  227.891056]    generic_write_end+0xf2/0x440
[  227.891484]    ntfs_write_end+0x42e/0x980
[  227.891920]    generic_perform_write+0x316/0x570
[  227.892393]    __generic_file_write_iter+0x2ae/0x500
[  227.892899]    ntfs_file_write_iter+0x66d/0x1d70
[  227.893378]    do_iter_readv_writev+0x20b/0x3c0
[  227.893838]    do_iter_write+0x188/0x710
[  227.894253]    vfs_iter_write+0x74/0xa0
[  227.894660]    iter_file_splice_write+0x745/0xc90
[  227.895133]    direct_splice_actor+0x114/0x180
[  227.895585]    splice_direct_to_actor+0x33b/0x8b0
[  227.896082]    do_splice_direct+0x1b7/0x280
[  227.896521]    do_sendfile+0xb49/0x1310
[  227.896926]    __x64_sys_sendfile64+0x1d0/0x210
[  227.897389]    do_syscall_64+0x41/0xc0
[  227.897804]    entry_SYSCALL_64_after_hwframe+0x63/0xcd
```

https://lore.kernel.org/all/20250513175633.85f4e19f4232a68ab04c8e41@linux-foundation.org/

> The patch fixes a deadlock which can be triggered by an internal
> syzkaller [1] reproducer and captured by bpftrace script [2] and its log
> [3] in this scenario:
>
```
> Process 1                   Process 2
> ---                         ---
> hugetlb_fault
>   mutex_lock(B) // take B
>   filemap_lock_hugetlb_folio
>     filemap_lock_folio
>       __filemap_get_folio
>         folio_lock(A) // take A
>   hugetlb_wp
>     mutex_unlock(B) // release B
>     ...                       hugetlb_fault
>     ...                         mutex_lock(B) // take B
>                                 filemap_lock_hugetlb_folio
>                                   filemap_lock_folio
>                                     __filemap_get_folio
>                                       folio_lock(A) // blocked
>     unmap_ref_private
>     ...
>     mutex_lock(B) // retake and blocked
>
> This is a ABBA deadlock involving two locks:
> - Lock A: pagecache_folio lock
> - Lock B: hugetlb_fault_mutex_table lock
```

Nostalgia. A decade or three ago many of us spent much of our lives staring at ABBA deadlocks. Then came Lockdep and after a few more years, it all stopped. I've long hoped that Lockdep would gain a solution to custom locks such as folio_wait_bit_common(), but not yet.

https://lore.kernel.org/all/20250513175633.85f4e19f4232a68ab04c8e41@linux-foundation.org/

> The patch fixes a deadlock which can be triggered by an internal
> syzkaller [1] reproducer and captured by bpftrace script [2] and its log
> [3] in this scenario:
>
> Process 1 Process 2
> --- ---
> hugetlb_fault
> mutex_lock(B) // take B
> filemap_lock_hugetlb_folio
> filemap_lock_folio
> __filemap_get_folio
> folio_lock(A) // take A
> hugetlb_wp
> mutex_unlock(B) // release B
> ... hugetlb_fault
> ... mutex_lock(B) // take B
> filemap_lock_hugetlb_folio
> filemap_lock_folio
> __filemap_get_folio
> folio_lock(A) // blocked
> unmap_ref_private
> ...
> mutex_lock(B) // retake and blocked
>
> This is a ABBA deadlock involving two locks:
> - Lock A: pagecache_folio lock
> - Lock B: hugetlb_fault_mutex_table lock

Nostalgia. A decade or three ago many of us spent much of our lives staring at ABBA deadlocks. Then came Lockdep and after a few more years, it all stopped. I've long hoped that Lockdep would gain a solution to custom locks such as folio_wait_bit_common(), but not yet.

https://lore.kernel.org/all/b6e00e77-4a8c-4e05-ab79-266bf05fcc2d@igalia.com/

It seems that after the first round, the deadlock is captured:
ubuntu@localhost:~$ ./repro_20250402_0225_154f8fb0580000
executing program

```
[  80.425842][ T3416] ====================================================
[  80.426707][ T3416] DEPT: Circular dependency has been detected.
[  80.427497][ T3416] 6.15.0-rc6+ #31 Not tainted
[  80.428084][ T3416] ----------------------------------------------------
[  80.428964][ T3416] Benefits
[  80.429330][ T3416] ----------------------------------------------------
[  80.430078][ T3416] *** DEADLOCK ***
[  80.430078][ T3416]
[  80.430736][ T3416] context A
[  80.431076][ T3416]    [S] (unknown)(pg_locked_map:0)
[  80.431637][ T3416]    [W] lock(&hugetlb_fault_mutex_table[i]:0)
[  80.432312][ T3416]    [E] DEPT_page_clear_bit(pg_locked_map:0)
[  80.432977][ T3416]
[  80.433246][ T3416] context B
[  80.433595][ T3416]    [S] lock(&hugetlb_fault_mutex_table[i]:0)
[  80.434245][ T3416]    [W] DEPT_page_wait_on_bit(pg_locked_map:0)
[  80.434880][ T3416]    [E] unlock(&hugetlb_fault_mutex_table[i]:0)
[  80.435592][ T3416]
[  80.435852][ T3416] [S]: start of the event context
[  80.436369][ T3416] [W]: the wait blocked
[  80.436789][ T3416] [E]: the event not reachable
[  80.437275][ T3416] ----------------------------------------------------
[  80.437950][ T3416] context A's detail
[  80.438367][ T3416] ----------------------------------------------------
[  80.439006][ T3416] context A
[  80.439337][ T3416]    [S] (unknown)(pg_locked_map:0)
[  80.439883][ T3416]    [W] lock(&hugetlb_fault_mutex_table[i]:0)
[  80.440489][ T3416]    [E] DEPT_page_clear_bit(pg_locked_map:0)
[  80.441075][ T3416]
[  80.441318][ T3416] [S] (unknown)(pg_locked_map:0):
[  80.441816][ T3416] (N/A)
[  80.442077][ T3416]
[  80.442309][ T3416] [W] lock(&hugetlb_fault_mutex_table[i]:0):
[  80.442872][ T3416] [<ffffffff82144644>] hugetlb_wp+0xfa4/0x3490
[  80.443502][ T3416] stacktrace:
[  80.443810][ T3416]      hugetlb_wp+0xfa4/0x3490
[  80.444267][ T3416]      hugetlb_fault+0x1505/0x2c70
[  80.444776][ T3416]      handle_mm_fault+0x1845/0x1ab0
[  80.445275][ T3416]      do_user_addr_fault+0x637/0x1450
[  80.445779][ T3416]      exc_page_fault+0x67/0x110
[  80.446239][ T3416]      asm_exc_page_fault+0x26/0x30
[  80.446722][ T3416]      __put_user_4+0xd/0x20
[  80.447157][ T3416]      copy_process+0x1f64/0x3d80
[  80.447621][ T3416]      kernel_clone+0x216/0x940
[  80.448068][ T3416]      __x64_sys_clone+0x18d/0x1f0
[  80.448548][ T3416]      do_syscall_64+0x6f/0x120
[  80.448999][ T3416]      entry_SYSCALL_64_after_hwframe+0x76/0x7e
[  80.449556][ T3416]
[  80.449765][ T3416] [E] DEPT_page_clear_bit(pg_locked_map:0):
[  80.450272][ T3416] [<ffffffff8214263b>] hugetlb_fault+0x1ccb/0x2c70
[  80.450861][ T3416] stacktrace:
[  80.451148][ T3416]      hugetlb_fault+0x1ccb/0x2c70
[  80.451611][ T3416]      handle_mm_fault+0x1845/0x1ab0
[  80.452080][ T3416]      do_user_addr_fault+0x637/0x1450
[  80.452566][ T3416]      exc_page_fault+0x67/0x110
[  80.453014][ T3416]      asm_exc_page_fault+0x26/0x30
[  80.453497][ T3416]      __put_user_4+0xd/0x20
[  80.453923][ T3416]      copy_process+0x1f64/0x3d80
[  80.454379][ T3416]      kernel_clone+0x216/0x940
[  80.454817][ T3416]      __x64_sys_clone+0x18d/0x1f0
[  80.455277][ T3416]      do_syscall_64+0x6f/0x120
[  80.455722][ T3416]      entry_SYSCALL_64_after_hwframe+0x76/0x7e
[  80.456253][ T3416] ----------------------------------------------------
[  80.456842][ T3416] context B's detail
[  80.457198][ T3416] ----------------------------------------------------
[  80.457842][ T3416] context B
[  80.458122][ T3416]    [S] lock(&hugetlb_fault_mutex_table[i]:0)
[  80.458661][ T3416]    [W] DEPT_page_wait_on_bit(pg_locked_map:0)
[  80.459187][ T3416]    [E] unlock(&hugetlb_fault_mutex_table[i]:0)
[  80.459763][ T3416]
[  80.459988][ T3416] [S] lock(&hugetlb_fault_mutex_table[i]:0):
[  80.460509][ T3416] [<ffffffff82140d36>] hugetlb_fault+0x3c6/0x2c70
[  80.461074][ T3416] stacktrace:
[  80.461374][ T3416]      hugetlb_fault+0x3c6/0x2c70
[  80.461812][ T3416]      handle_mm_fault+0x1845/0x1ab0
[  80.462281][ T3416]      do_user_addr_fault+0x637/0x1450
[  80.462775][ T3416]      exc_page_fault+0x67/0x110
[  80.463220][ T3416]      asm_exc_page_fault+0x26/0x30
[  80.463694][ T3416]      __put_user_4+0xd/0x20
[  80.464129][ T3416]      copy_process+0x1f64/0x3d80
[  80.464577][ T3416]      kernel_clone+0x216/0x940
[  80.464994][ T3416]      __x64_sys_clone+0x18d/0x1f0
[  80.465466][ T3416]      do_syscall_64+0x6f/0x120
[  80.465909][ T3416]      entry_SYSCALL_64_after_hwframe+0x76/0x7e
[  80.466457][ T3416]
[  80.466660][ T3416] [W] DEPT_page_wait_on_bit(pg_locked_map:0):
[  80.467177][ T3416] [<ffffffff82141187>] hugetlb_fault+0x817/0x2c70
[  80.467740][ T3416] stacktrace:
[  80.468032][ T3416]      hugetlb_fault+0x817/0x2c70
[  80.468479][ T3416]      handle_mm_fault+0x1845/0x1ab0
[  80.468947][ T3416]      do_user_addr_fault+0x637/0x1450
[  80.469428][ T3416]      exc_page_fault+0x67/0x110
[  80.469865][ T3416]      asm_exc_page_fault+0x26/0x30
[  80.470332][ T3416]      __put_user_4+0xd/0x20
[  80.470742][ T3416]      copy_process+0x1f64/0x3d80
[  80.471186][ T3416]      kernel_clone+0x216/0x940
[  80.471616][ T3416]      __x64_sys_clone+0x18d/0x1f0
[  80.472060][ T3416]      do_syscall_64+0x6f/0x120
[  80.472492][ T3416]      entry_SYSCALL_64_after_hwframe+0x76/0x7e
[  80.473040][ T3416]
[  80.473271][ T3416] [E] unlock(&hugetlb_fault_mutex_table[i]:0):
[  80.473863][ T3416] (N/A)
```

Introduction

Lockdep

DEPT

Benefits

BENEFIT 1.

Focuses on waits and events which also covers typical locks

folio_lock() / folio_unlock()

wait_for_completion() / complete()

Any types of waits and events

BENEFIT 2.

Provides APIs to directly annotate waits and events

```
DEFINE_DEPT_SDT(my_wait_event);

sdt_wait(&my_wait_event);

sdt_event(&my_wait_event);
```

BENEFIT 3.

Tracks read-write locks simply with no clever tricks

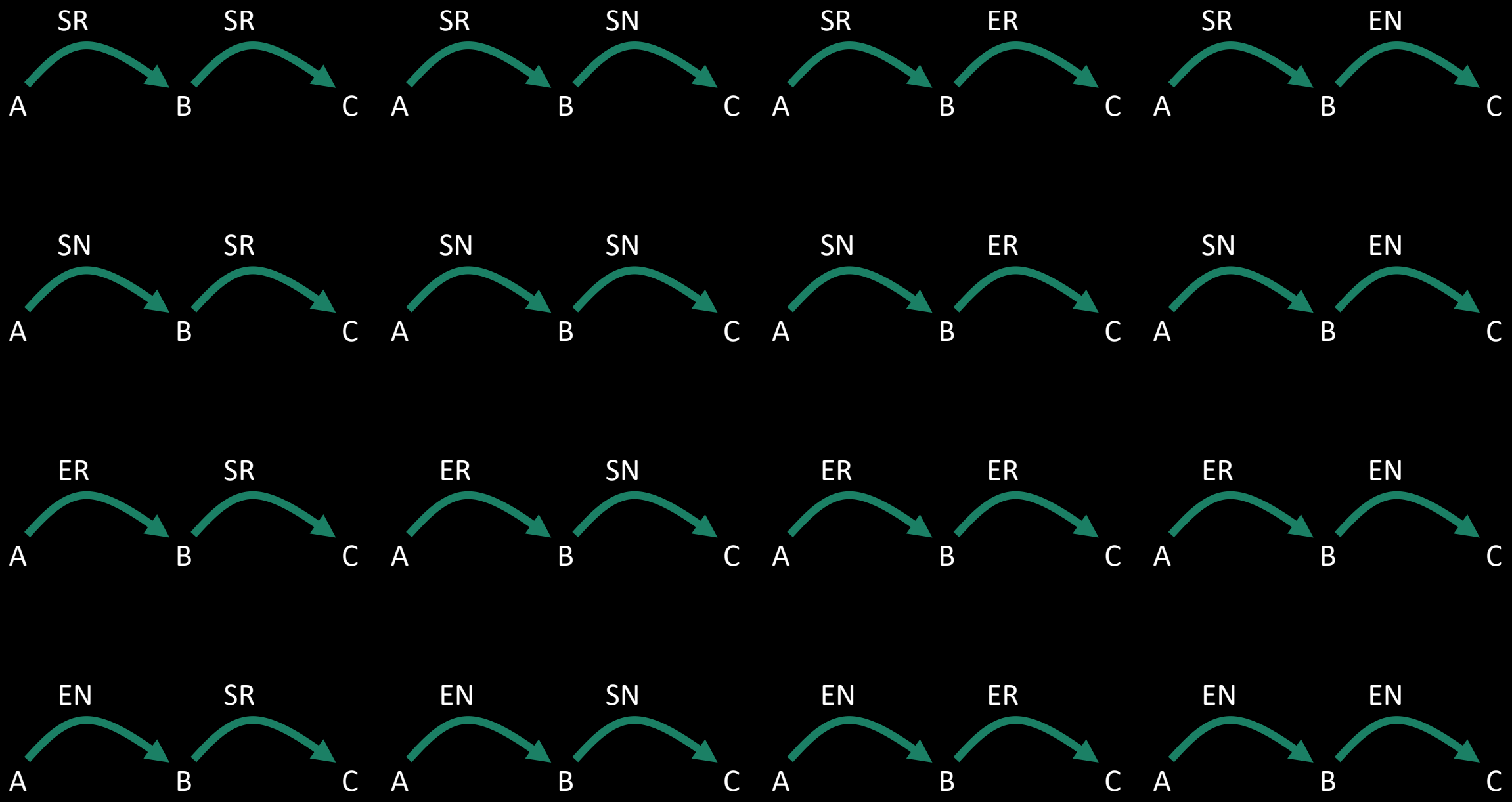
Lockdep considers all possible cases of two consecutive lock chains to track read-write locks.

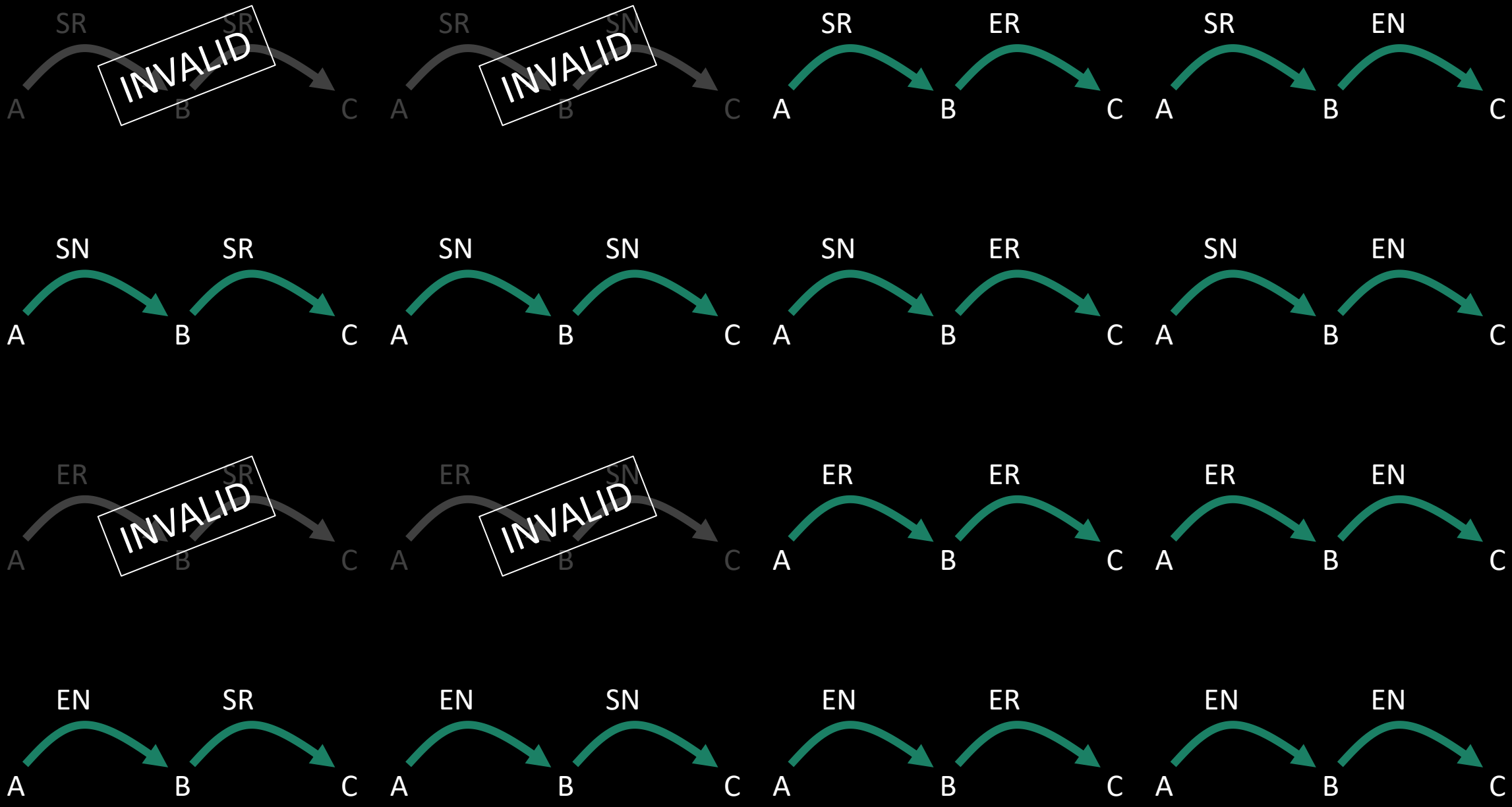
SR Shared reader - Recursive reader

ER Exclusive locker - Recursive reader

SN Shared reader - Non-recursive locker

EN Exclusive locker - Non-recursive locker


SR SR
A B C
SR SN
A B C
SR ER
A B C
SR EN
A B C
SN SR
A B C
SN SN
A B C
SN ER
A B C
SN EN
A B C
ER SR
A B C
ER SN
A B C
ER ER
A B C
ER EN
A B C
EN SR
A B C
EN SN
A B C
EN ER
A B C
EN EN
A B C


SR SR INVALID A B C
SR SN INVALID A B C
SR ER A B C
SR EN A B C
SN SR A B C
SN SN A B C
SN ER A B C
SN EN A B C
ER SR INVALID A B C
ER SN INVALID A B C
ER ER A B C
ER EN A B C
EN SR A B C
EN SN A B C
EN ER A B C
EN EN A B C

DEPT just focuses on waits and events even for read-write locks.

read lock waits for read unlocks.

read lock waits for write unlocks.

write lock waits for read unlocks.

write lock waits for write unlocks.

read lock waits for read unlocks.

read lock waits for write unlocks.

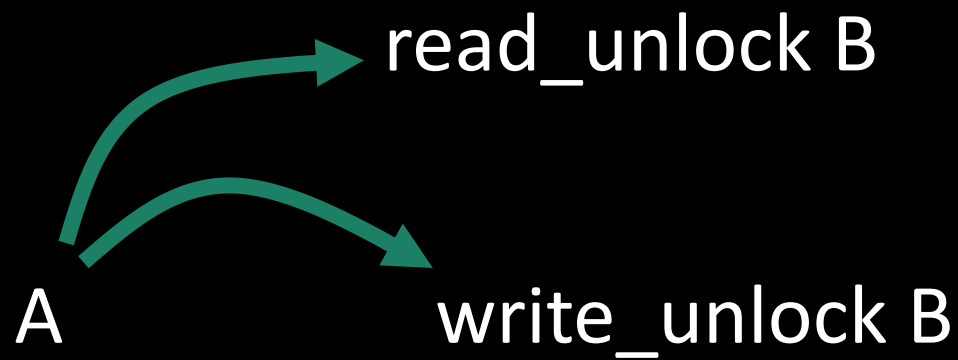
write lock waits for read unlocks.

write lock waits for write unlocks.

```
mutex_lock A
...
// Waits for Event B, by write_unlock B.
read_lock B
...
// Emit Event A.
mutex_unlock A
```

Build Dependency Graph

```
mutex_lock A
...
// Waits for Event B, by read_unlock B or write_unlock B.
write_lock B
...
// Emit Event A.
mutex_unlock A
```



Build Dependency Graph

Read-write locks are also waits and events after all.

BENEFIT 4.

Support multi reports

Lockdep halts after a report. We must resolve each issue sequentially to proceed to the next.

DEPT doesn’t stop and generates all possible reports in a single run unless memory is exhausted.

Q1: Do you intend to totally replace Lockdep?

A: No. Lockdep doesn’t perform only dependency check but also lock usage correctness check. The dependency check should be replaced but the lock usage correctness check should stay.

AS IS

Lockdep

Lock usage correctness check

Lock dependency check

DEPT

General dependency check

TO BE

Lockdep

Lock usage correctness check

Utilize DEPT for dependency check

DEPT

General dependency check

Q2: Lockdep halts after the first report, so any later valid issues get buried - false positives make this even worse. That was the biggest obstacle to adopting cross-release. Does DEPT handle this better?

A: As coverage increases, it's inevitable that the number of false positives also goes up.

False positives still need to be fixed, but since DEPT supports multiple reports, they're no longer fatal to adoption unlike cross-release.

Q3: Why don't you resolve all the false positives first?

A: Like Lockdep, which has relied on annotations to avoid false positives over the last two decades, DEPT too will require the annotation efforts.

Performing the annotation work within the mainline will help us add annotations more appropriately and will also make DEPT a useful tool for a wider range of users more quickly.

Q4: Why don’t you develop DEPT as a Lockdep extension?

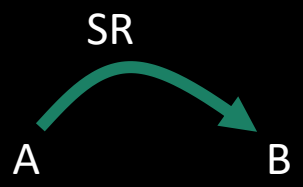
A: Reusing BFS(Breadth First Search) and hashing of Lockdep is an advantage, but all the other parts need to be rewritten to reflect the perspective of DEPT since Lockdep was originally designed to track lock acquisition orders and doesn’t fit on wait-event model.

Appendix

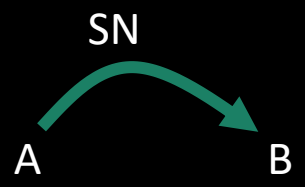
How Lockdep tracks read-write locks

Considers all possible cases of two consecutive lock chains.

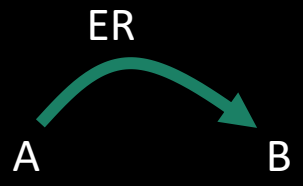
```
// Holds A.
read_lock A
...
// Holds B while still holding A.
read_lock B
...
read_unlock B
read_unlock A
```



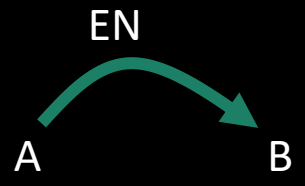

```
// Holds A.
read_lock A
...
// Holds B while still holding A.
write_lock B
...
write_unlock B
read_unlock A
```

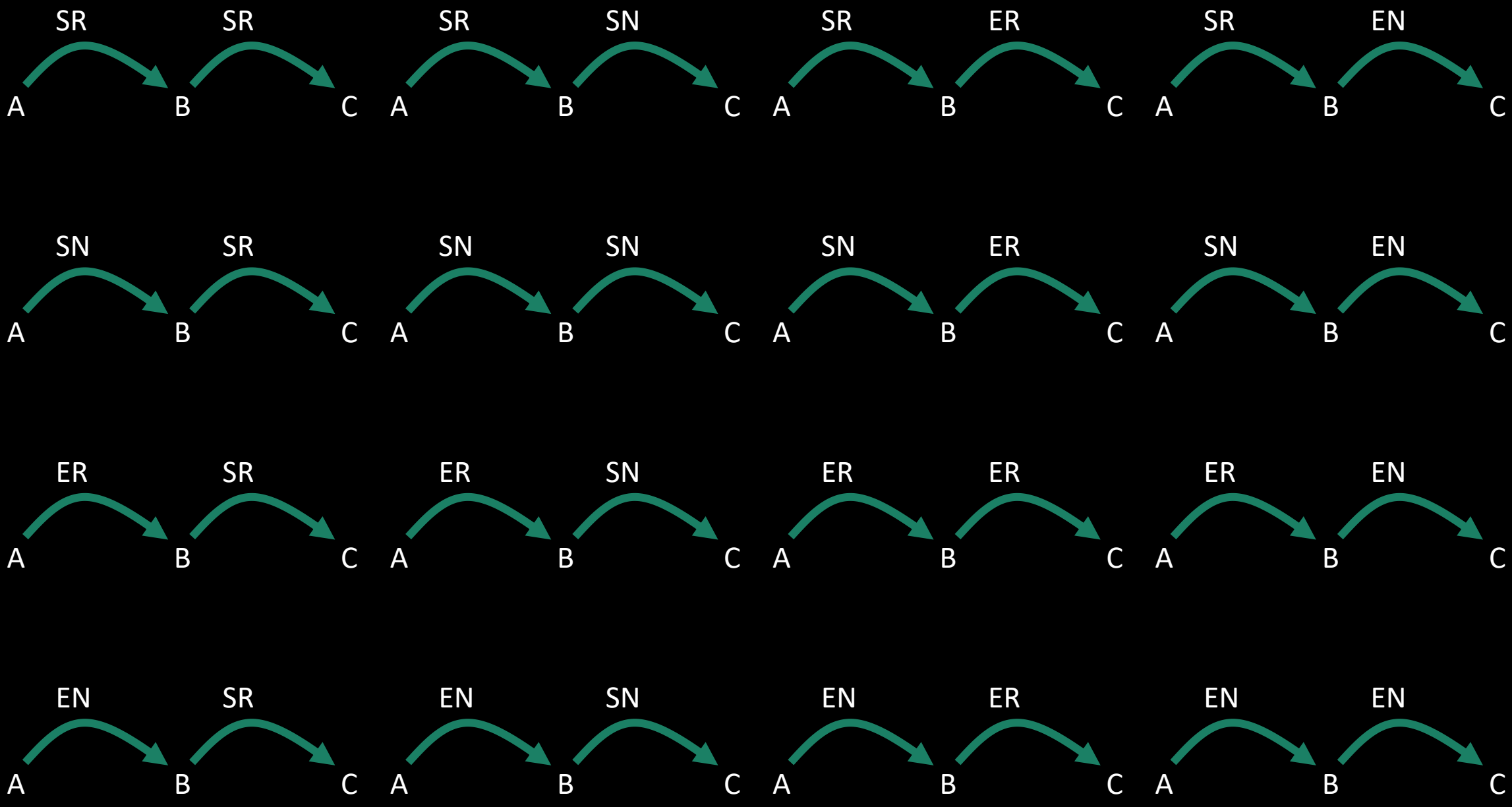



```
// Holds A.
write_lock A
...
// Holds B while still holding A.
read_lock B
...
read_unlock B
write_unlock A
```

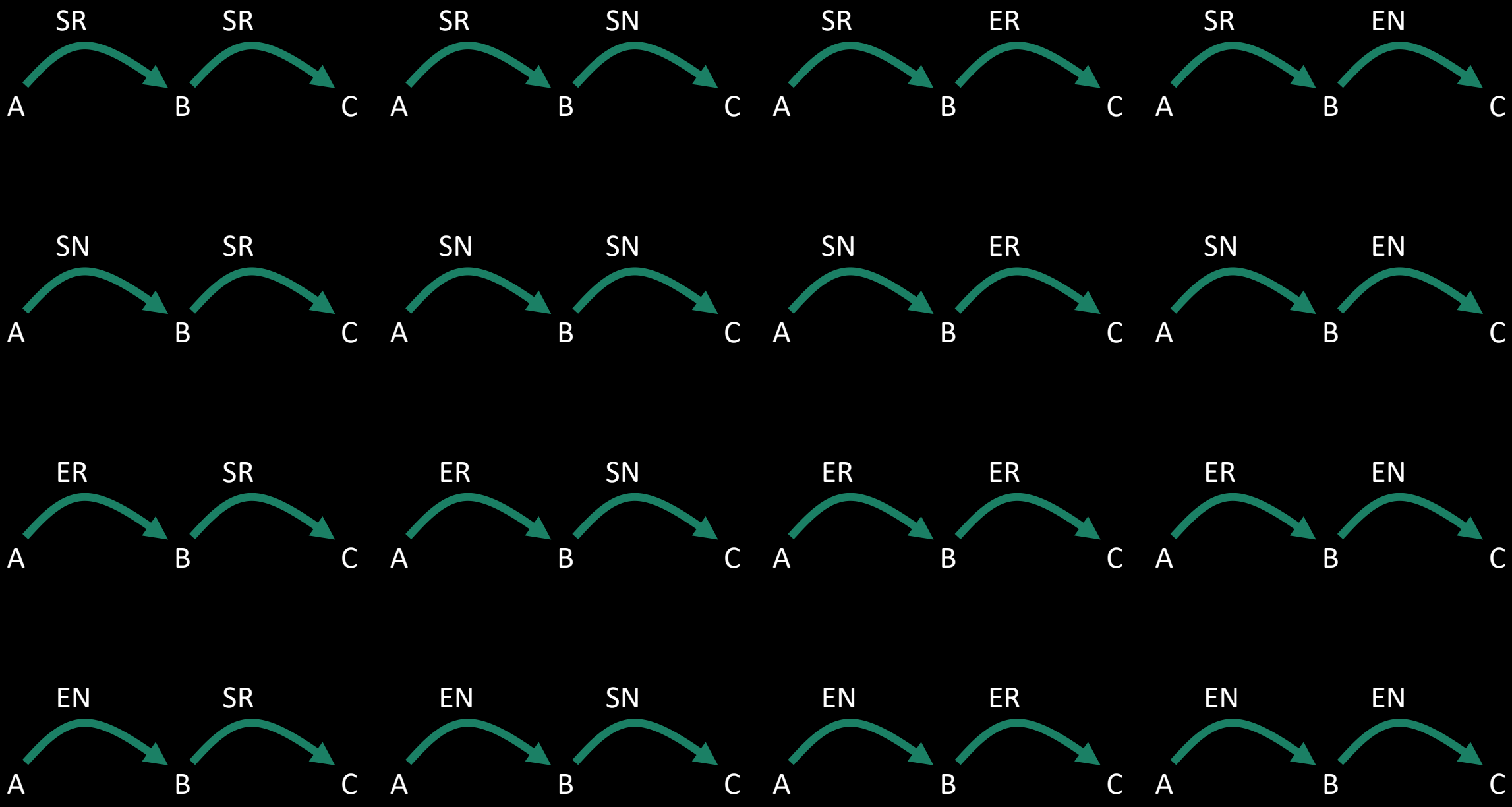


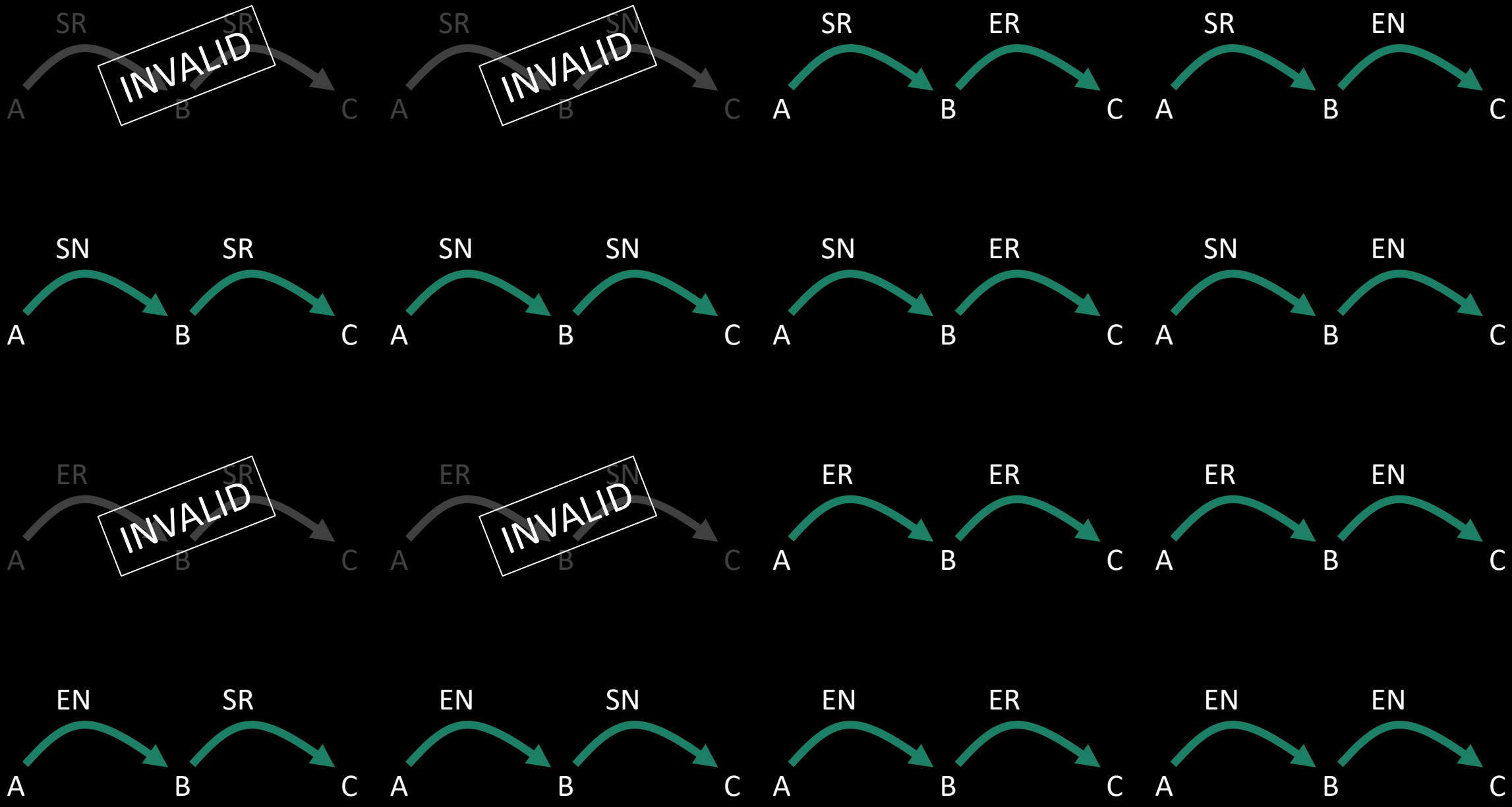

```
// Holds A.
write_lock A
...
// Holds B while still holding A.
write_lock B
...
write_unlock B
write_unlock A
```



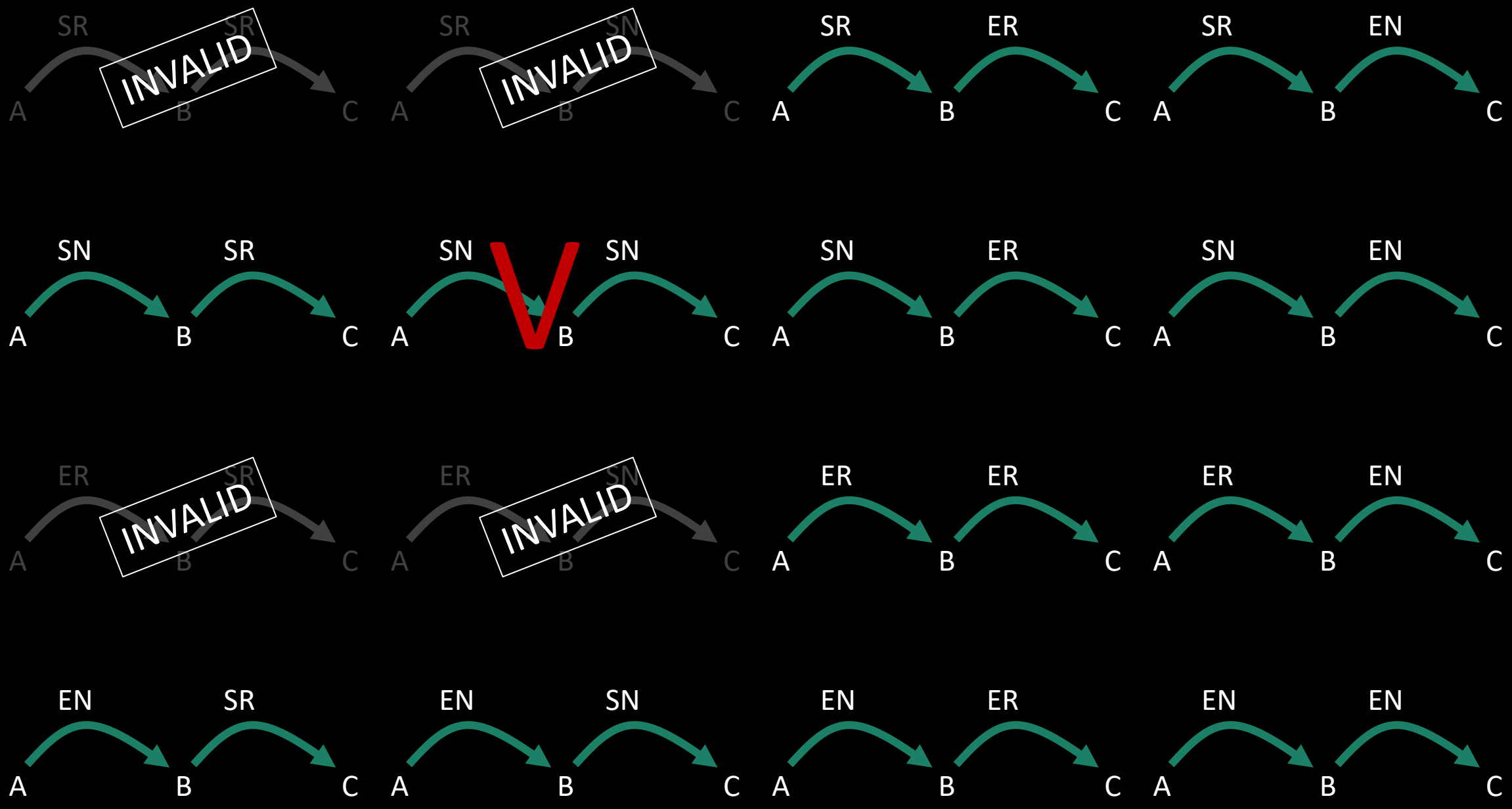

SR SR
A B C
SR SN
A B C
SR ER
A B C
SR EN
A B C
SN SR
A B C
SN SN
A B C
SN ER
A B C
SN EN
A B C
ER SR
A B C
ER SN
A B C
ER ER
A B C
ER EN
A B C
EN SR
A B C
EN SN
A B C
EN ER
A B C
EN EN
A B C

Verifies only the valid chains and ignores the others.


SR SR
A B C
SR SN
A B C
SR ER
A B C
SR EN
A B C
SN SR
A B C
SN SN
A B C
SN ER
A B C
SN EN
A B C
ER SR
A B C
ER SN
A B C
ER ER
A B C
ER EN
A B C
EN SR
A B C
EN SN
A B C
EN ER
A B C
EN EN
A B C


SR
SR
INVALID
A
B
C
SR
SN
INVALID
A
B
C
SR
ER
A
B
C
SR
EN
A
B
C
SN
SR
A
B
C
SN
SN
A
B
C
SN
ER
A
B
C
SN
EN
A
B
C
ER
SR
INVALID
A
B
C
ER
SN
INVALID
A
B
C
ER
ER
A
B
C
ER
EN
A
B
C
EN
SR
A
B
C
EN
SN
A
B
C
EN
ER
A
B
C
EN
EN
A
B
C

Let’s see a valid chains and an invalid chains as an example.


SR
SR
INVALID
A
B
C
SR
SN
INVALID
A
B
C
SR
ER
A
B
C
SR
EN
A
B
C
SN
SR
A
B
C
SN
SN
A
B
C
SN
ER
A
B
C
SN
EN
A
B
C
ER
SR
INVALID
A
B
C
ER
SN
INVALID
A
B
C
ER
ER
A
B
C
ER
EN
A
B
C
EN
SR
A
B
C
EN
SN
A
B
C
EN
ER
A
B
C
EN
EN
A
B
C

Example of Valid Chain

```
// Holds A.
read_lock A
...
// Holds B while still holding A.
write_lock B
...
write_unlock B
read_unlock A
```

```
// Holds B.
read_lock B
...
// Holds C while still holding B.
write_lock C
...
write_unlock C
read_unlock B
```

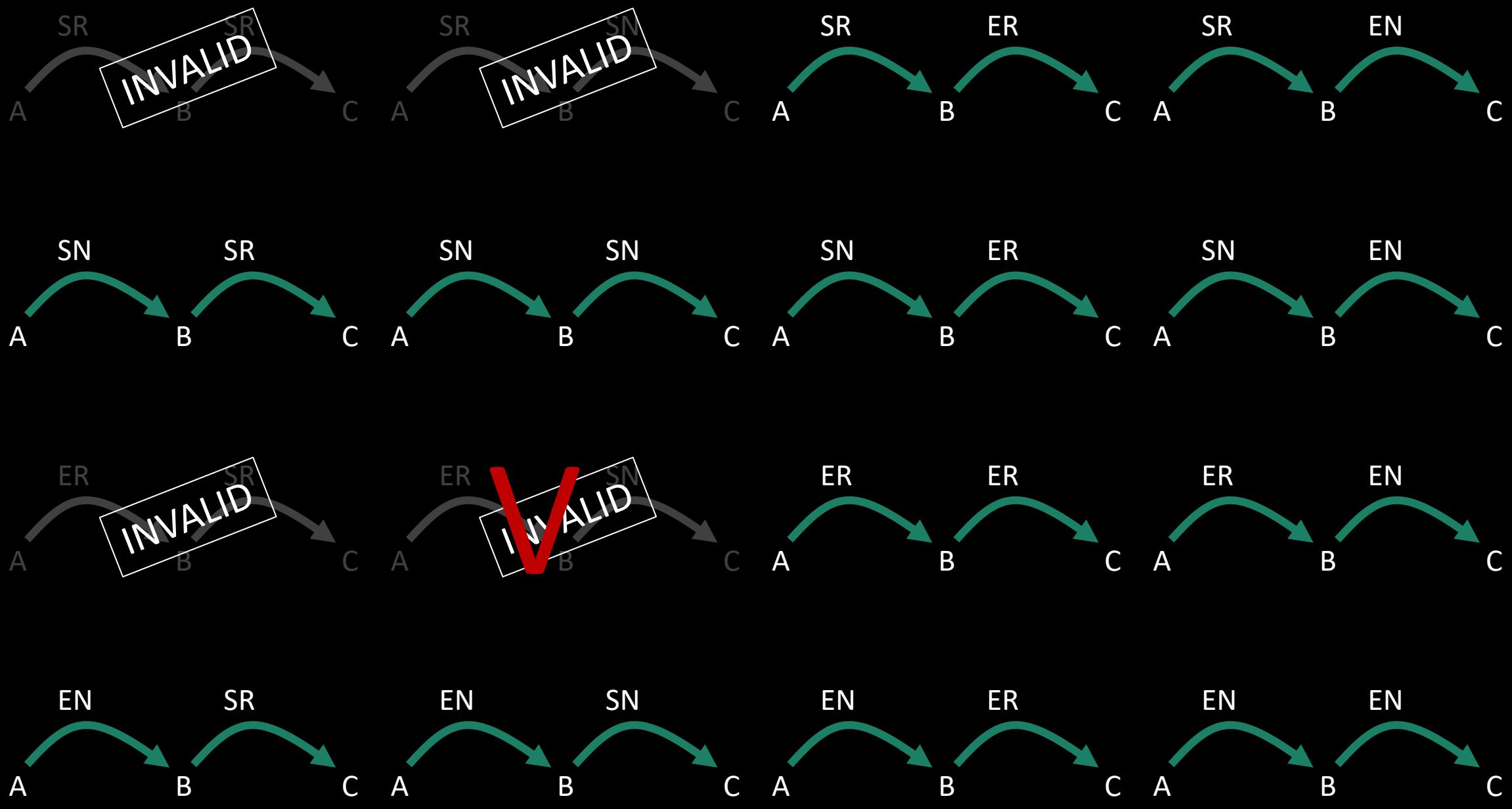
Example of Valid Chain

```
// Holds A.
read_lock A

...

// Holds B while still holding A.
write_lock B

...
write_unlock B
read_unlock A
```

```
// Holds B.
read_lock B

...

// Holds C while still holding B.
write_lock C

...
write_unlock C
read_unlock B
```

write_lock B is blocked by the read_lock B holder.

It would contribute to a deadlock.


SR SR
INVALID
A B C
SR SN
INVALID
A B C
SR ER
A B C
SR EN
A B C
SN SR
A B C
SN SN
A B C
SN ER
A B C
SN EN
A B C
ER SR
INVALID
A B C
ER SN
INVALID
A B C
ER ER
A B C
ER EN
A B C
EN SR
A B C
EN SN
A B C
EN ER
A B C
EN EN
A B C

Example of Invalid Chain

```
// Holds A.
write_lock A
...
// Holds B while still holding A.
read_lock B
...
read_unlock B
write_unlock A
```

```
// Holds B.
read_lock B
...
// Holds C while still holding B.
write_lock C
...
write_unlock C
read_unlock B
```

Example of Invalid Chain

```
// Holds A.
write_lock A
...
// Holds B while still holding A.
read_lock B
...
read_unlock B
write_unlock A
```

```
// Holds B.
read_lock B
...
// Holds C while still holding B.
write_lock C
...
write_unlock C
read_unlock B
```

read_lock B is not blocked by the read_lock B holder.

It wouldn't contribute to a deadlock so can be ignored.