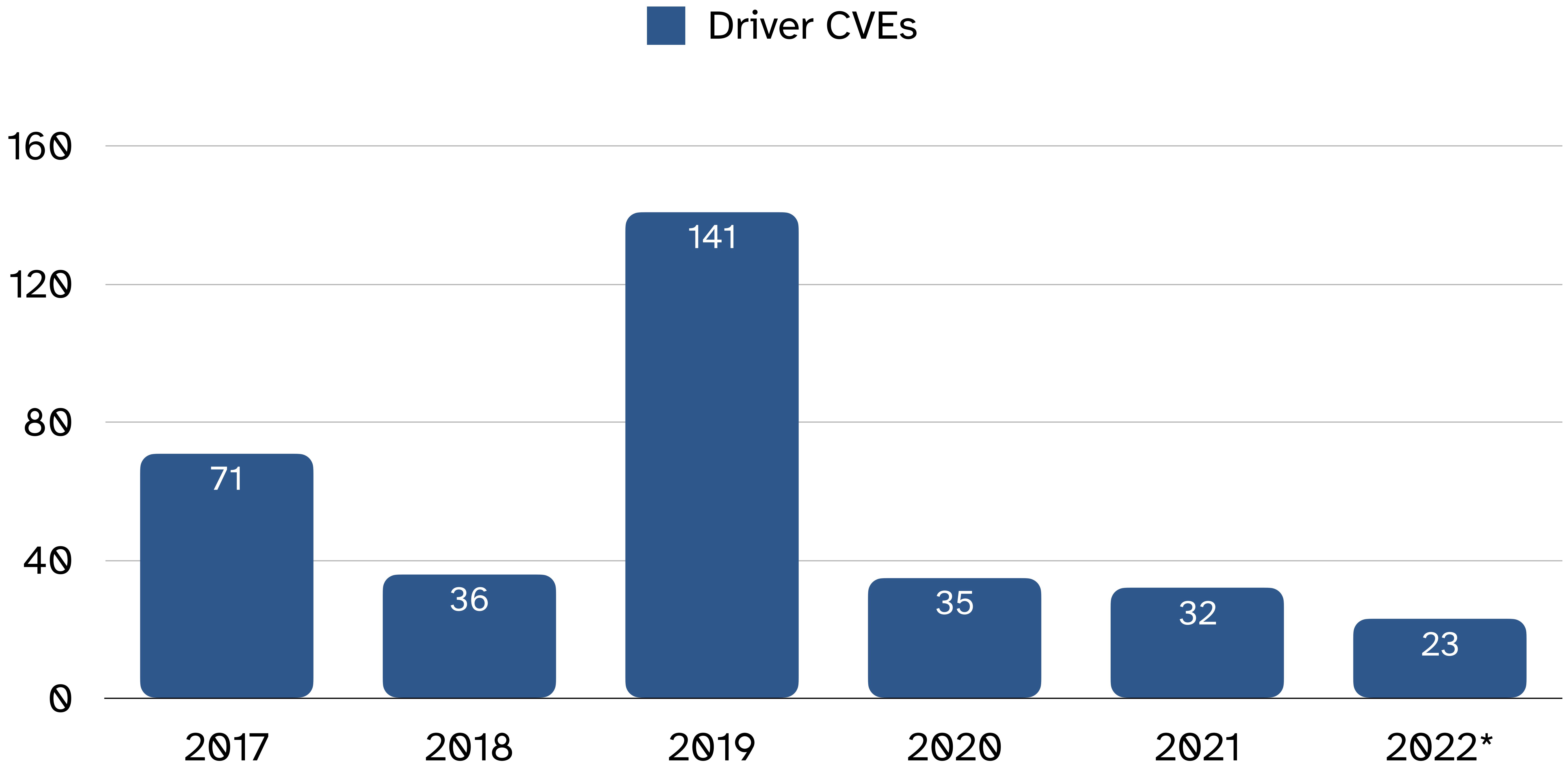
KSplit: Automating Device Driver Isolation

Yongzhe Huang[1], *Vikram Narayanan*[2], David Detweiler[2], Kaiming Huang[1], Gang Tan[1], Trent Jaeger[1], and Anton Burtsev[2,3]

[1]*Penn State University* [2]*University of California, Irvine* [3]*University of Utah*

Driver vulnerabilities

- 16-50 % of all Linux kernel CVEs



LXDs: Towards Isolation of Kernel Subsystems

Vikram N... ...vanan
University of ...nia, Irvine

Sarah Spall...
University ...

...Bauer
...of Utah

Abhiram Balasubramanian
University of Utah

Michael Quigley
University of ...

VirtuOS: an operating system with kernel virtualization

Ruslan Nikolaev, Godmar Back
rnikola@vt.edu, gback@cs.vt.edu
Virginia Tech
Blacksburg, VA

Lightweight Kernel Isolation with Virtualization and VM Functions

Tolerating Malicious Device Drivers in Linux

Silas Boyd-Wickizer and Nickolai Zeldovich
MIT CSAIL

MIT/LCS/TR-196

FINAL REPORT OF THE MULTICS KERNEL DESIGN PROJECT

by

M.D. Schro...
D.D. Clark
J.H. Saltz...
D.H. Wells

June 30, 19...

Decaf: Moving Device Drivers to a Modern Language

Matthew J. Renzelmann and Michael M. Swift
University of Wisconsin–Madison
{mjr, swift}@cs.wisc.edu

Microdrivers: A New Architecture for Device Drivers

Vinod Ganapathy, Arini Balakrishnan, Michael M. Swift
Computer Sciences Department, Universit...

Nooks: An Architecture for Reliable Device Drivers *

...Swift, Steven Martin, Henry M. Levy, and Susan J. Eggers
...Computer Science and Engineering
...Washington
...195, USA
...gers} @cs.washington.edu

The SawMill Multiserver Approach

Alain Gefflaut, Trent Jaeger, Yoonho Park
Jochen Liedtke *, Kevin J. Elphinstone †, Volkmar Uhlig †

Driver Isolation Architecture

kernel code

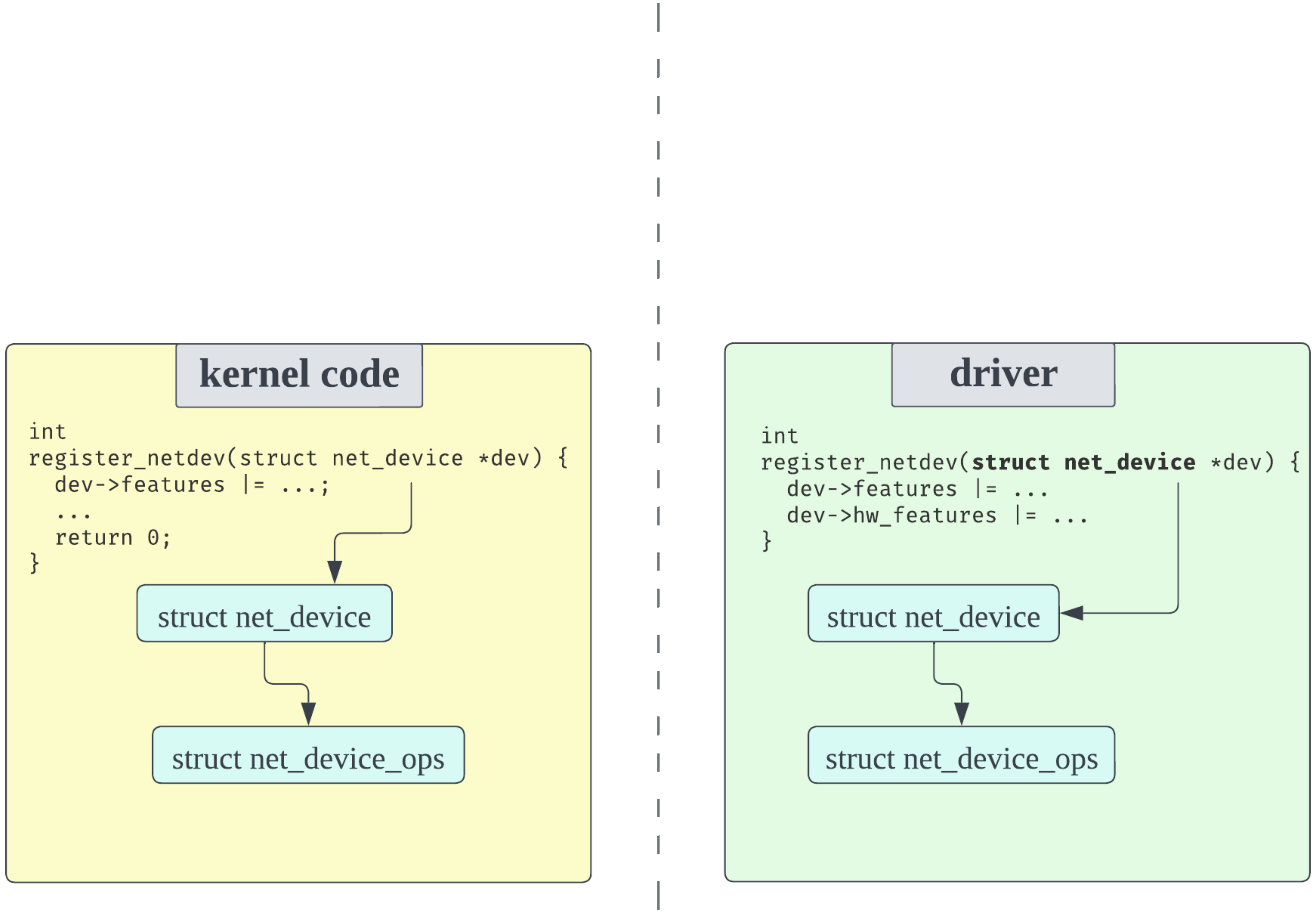
```
int
register_netdev(struct net_device *dev) {
  dev->features |= ...;
  ...
  return 0;
}
```

driver

```
int
register_netdev(struct net_device *dev) {
  dev->features |= ...
  dev->hw_features |= ...
}
```

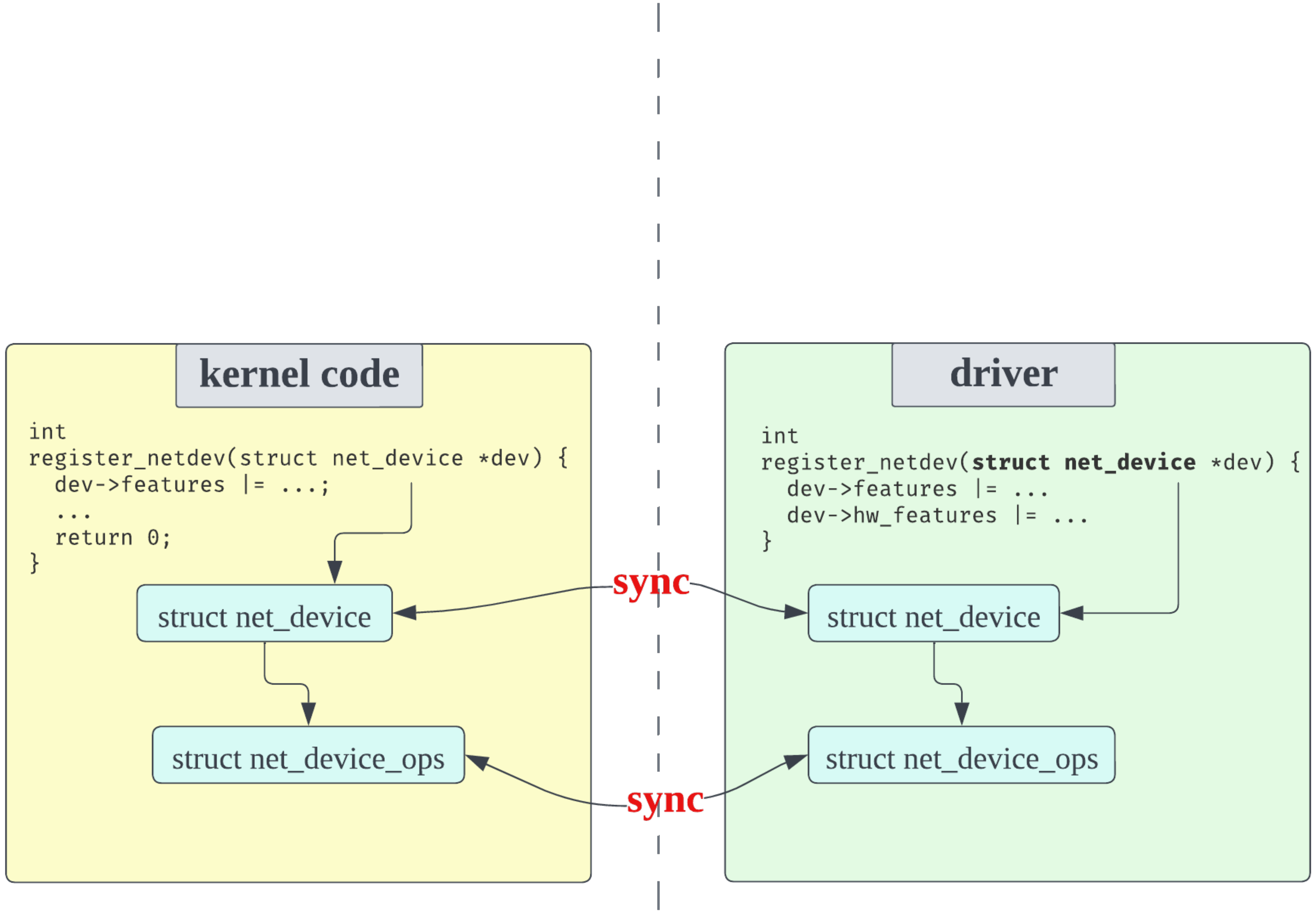
- Separate memory space

Driver isolation architecture



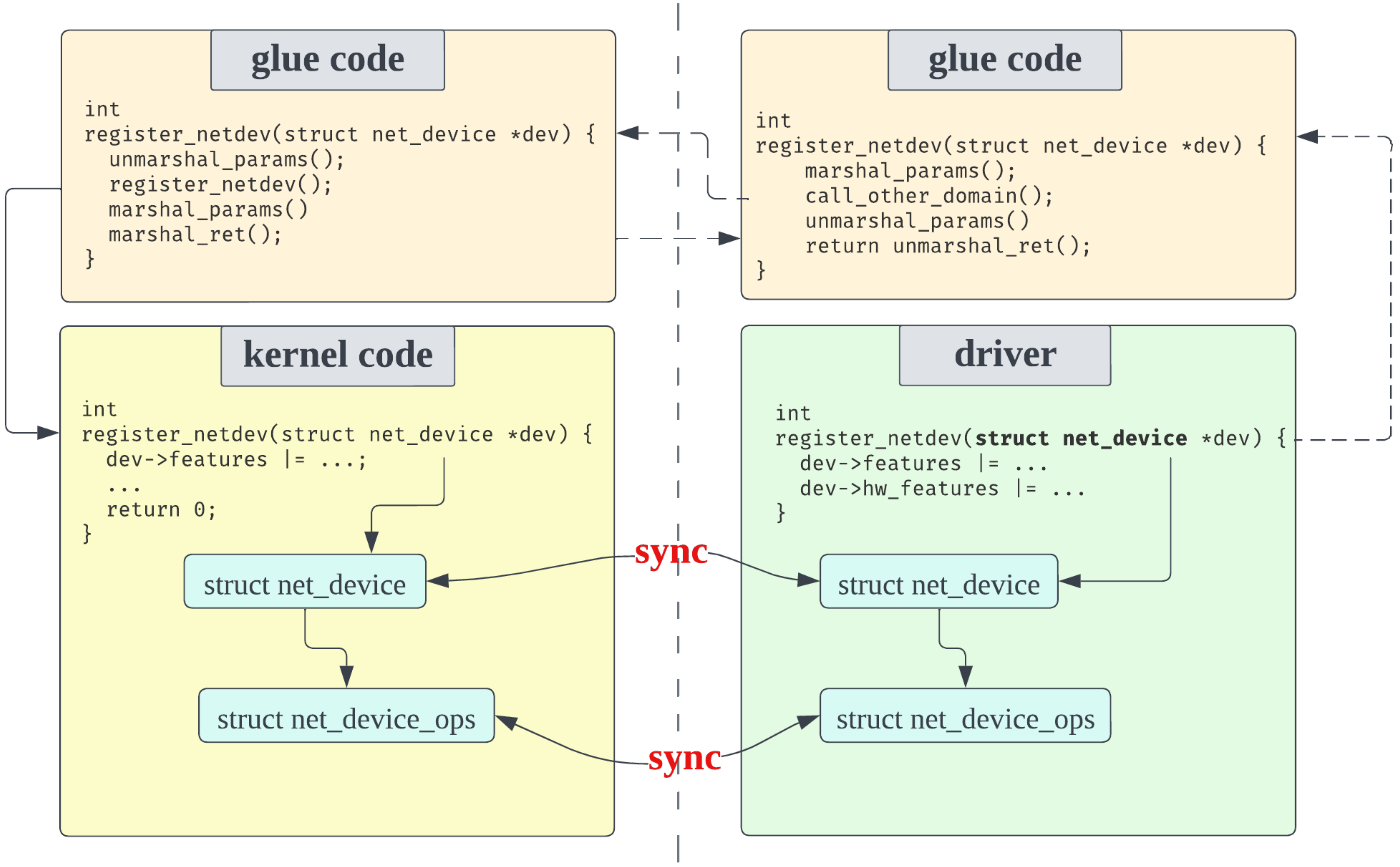
- Separate memory space
  - Two copies of object hierarchies

Driver isolation architecture



- Separate memory space
  - Two copies of object hierarchies
  - Keep them synchronized

Driver isolation architecture



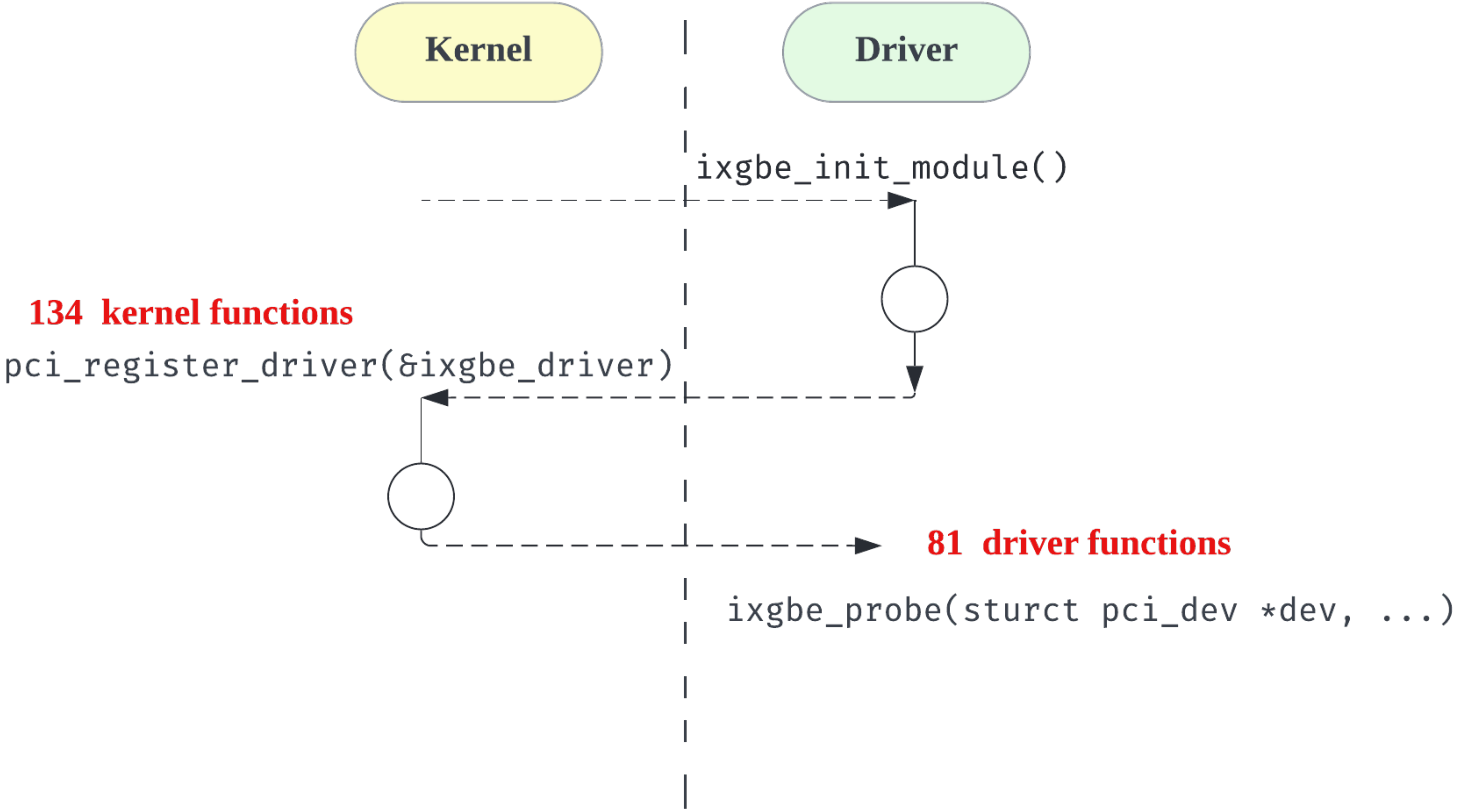
- Separate memory space
  - Two copies of object hierarchies
  - Keep them synchronized
- Glue code
  - Marshal/unmarshal params
  - Interface definition language (IDL) spec
  - Generated with IDL compiler

Isolation performance

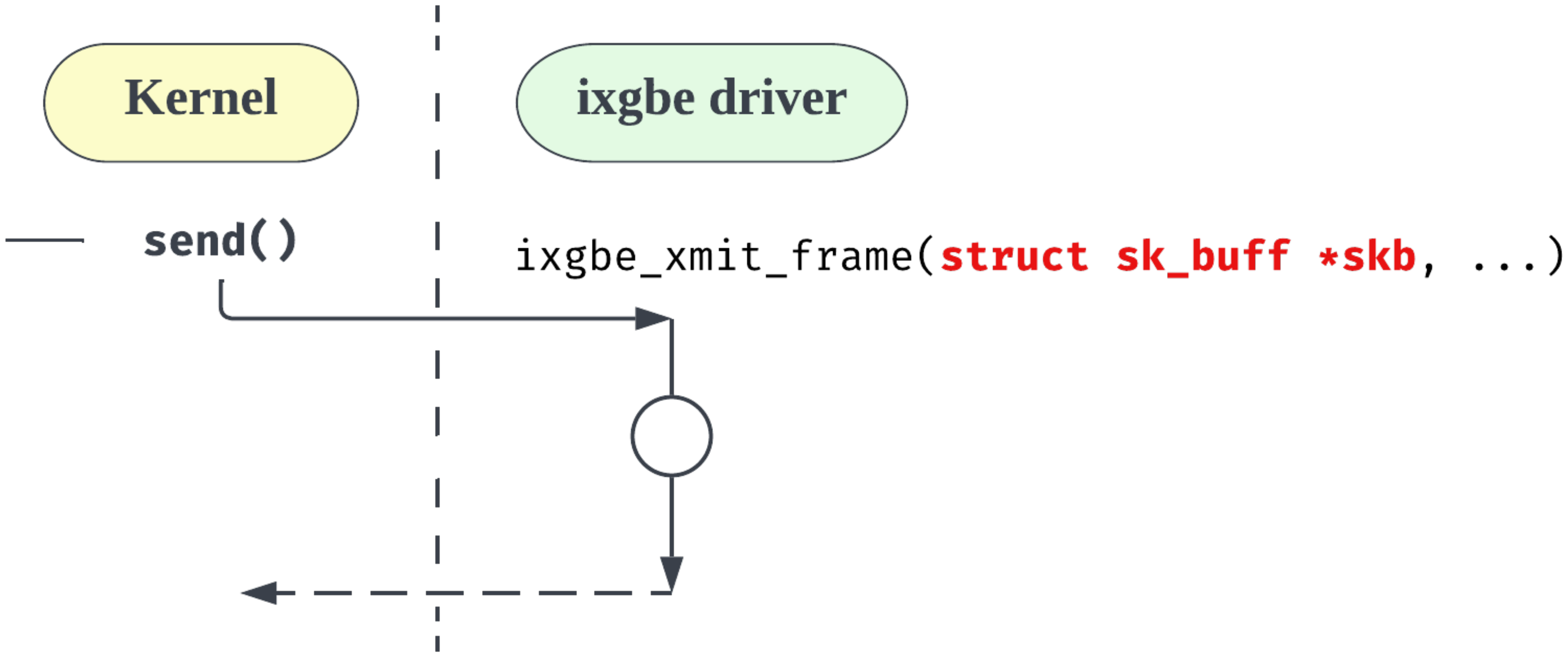
- Paging (834 cycles)
- Recent CPU mechanisms
  - VMFUNC - 396 cycles
  - MPK 11-260 cycles
  - Save/restore general/extended regs, pick a stack, etc.

Manually specifying the IDL for data synchronization between domains has become the major challenge

Challenge: Large interface boundary

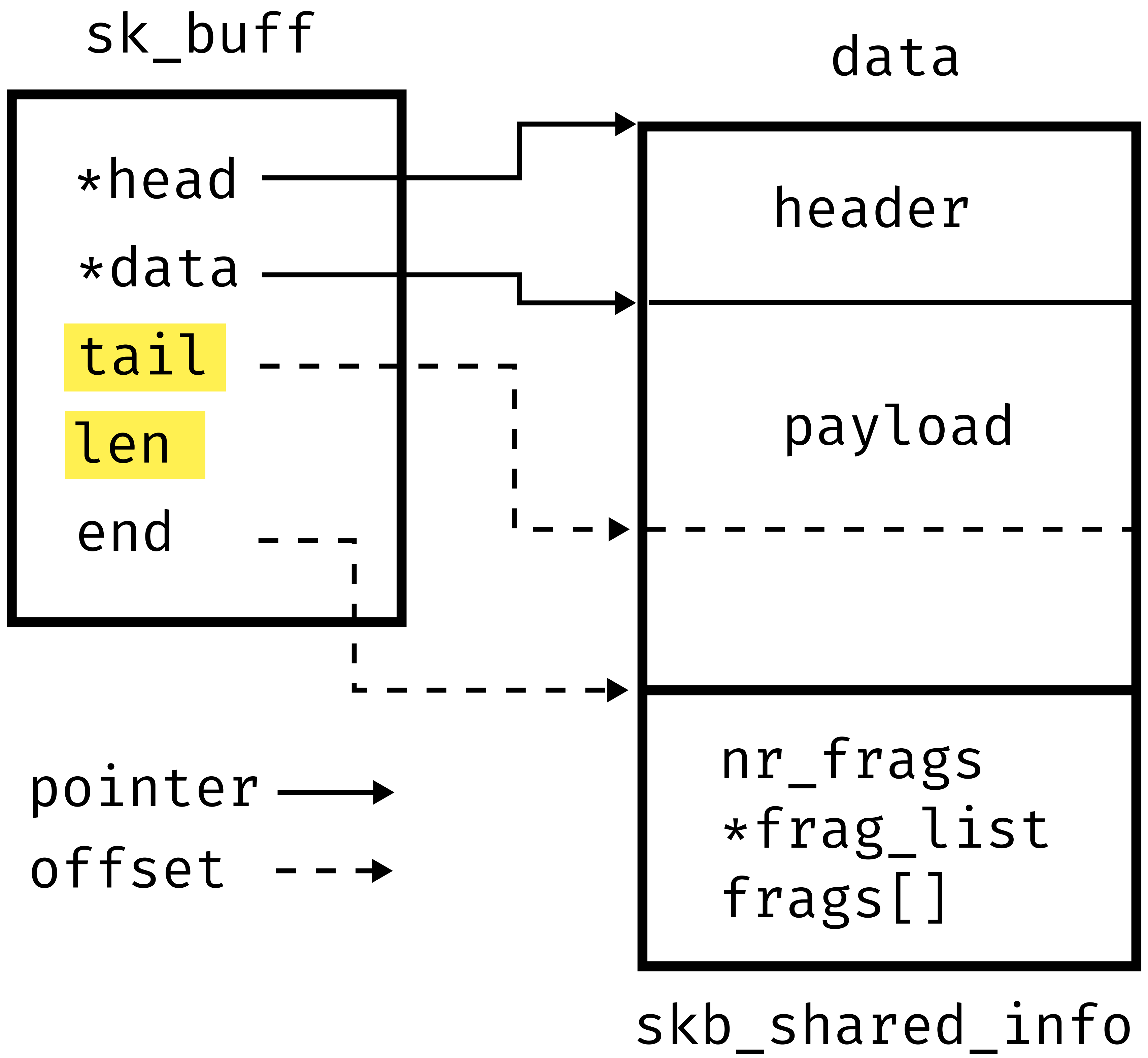


Challenge: Complex data exchange



Challenge: Complex data structures

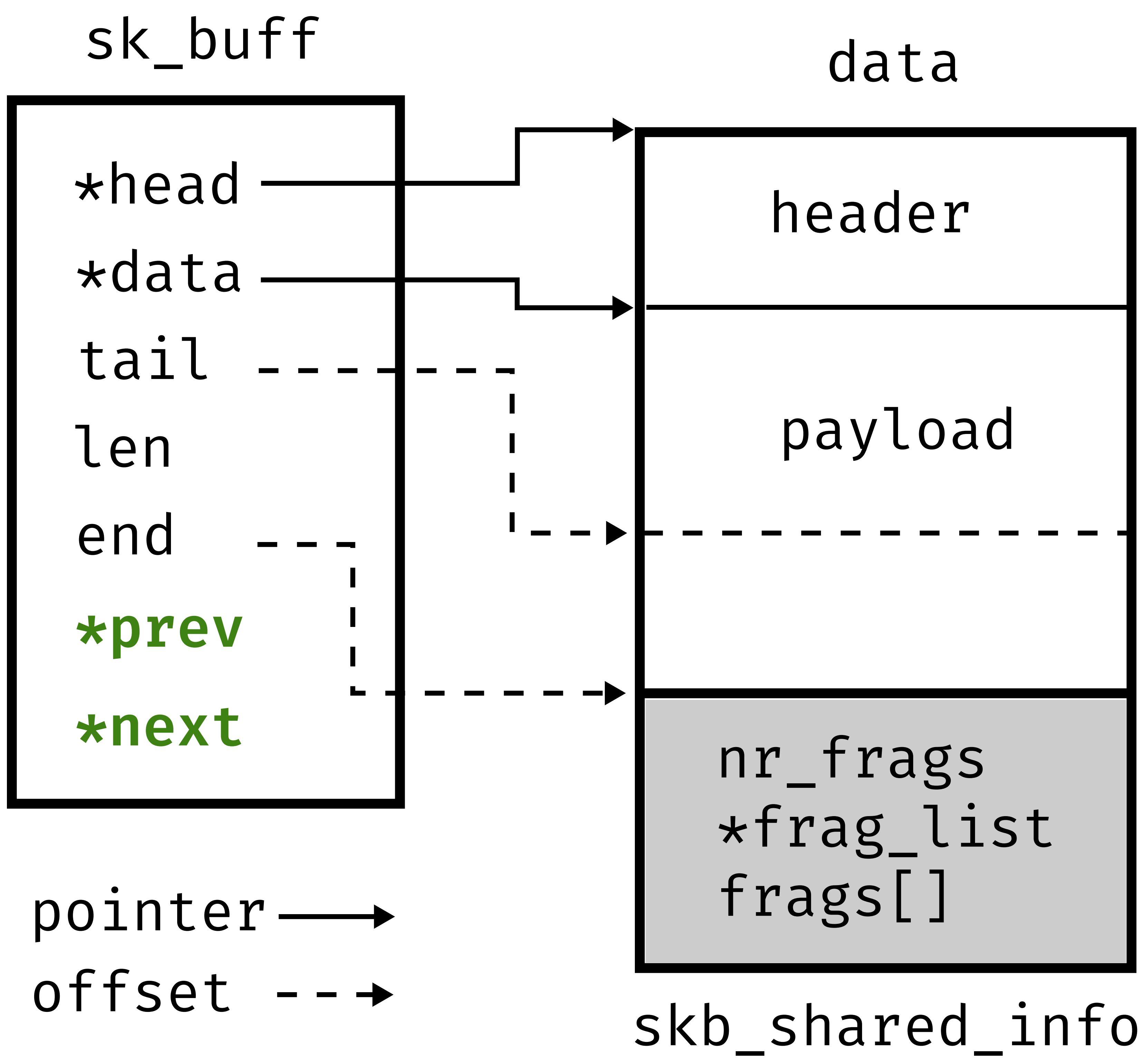
```
ixgbe_xmit_frame(struct sk_buff *skb, …)
```



- Represents a network packet
- Has 66 fields (5 pointers)
- 3,132 fields (1,214 pointers) are recursively reachable
- But only a small subset are accessed by both kernel and driver (shared)
  - 8 shared fields for this API

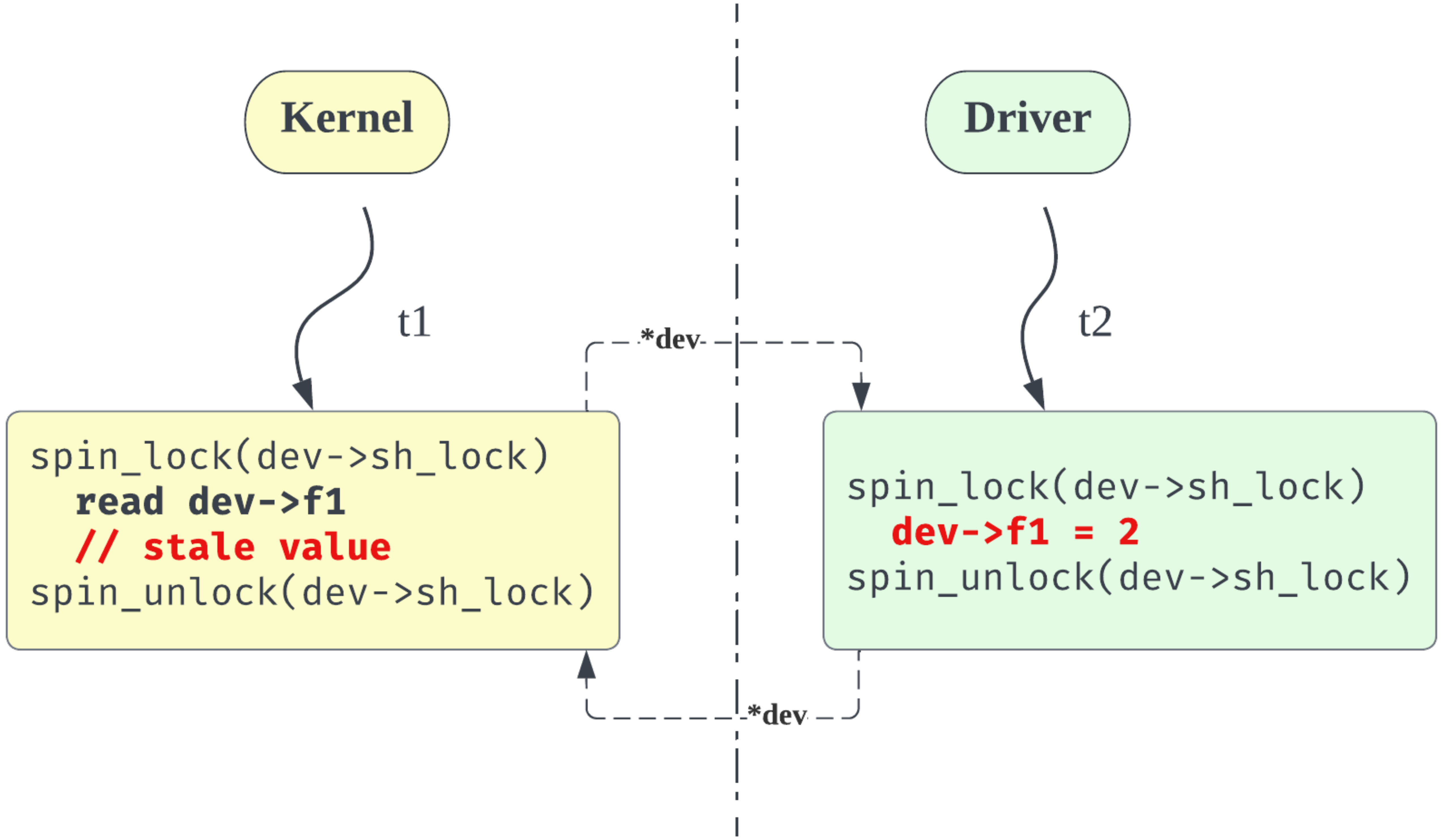
Challenge: Low-level kernel/C idioms

```
int ixgbe_xmit_frame(struct sk_buff* skb, …)
```



- Pointers
  - Singleton, array
  - Linked list
  - Collocated data structures
- Sized and sentinel arrays
- Special pointers (e.g., `__user`, `__iomem`)
- Tagged unions
- Return error as ptr (e.g., `ERR_PTR`)

Challenge: Concurrency primitives



- spin/mutex lock
- driver specific lock, e.g., rtnl_lock
- atomic operations, e.g., set_bit
- read-copy update (RCU)
- sequential lock

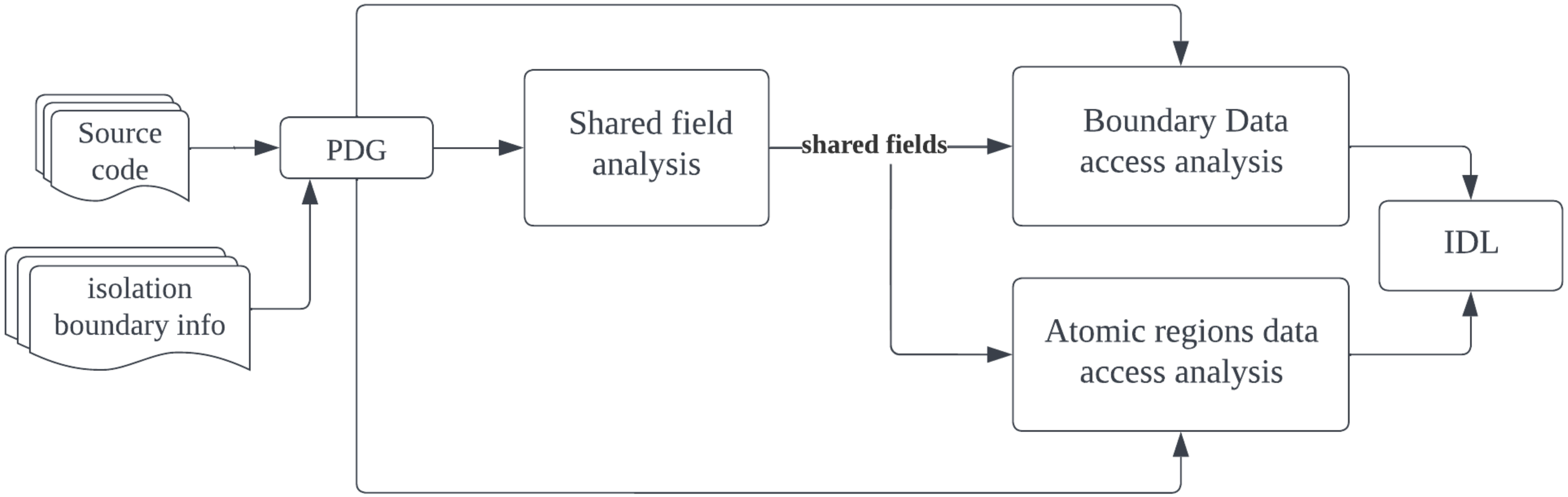
KSplit goals

- Build a set of static analyses to generate the IDL automatically (mostly) to
  - Isolate the complete driver
  - Identify shared/private data on the large interface boundary
  - Ensure each domain has the updated copy of the data structure
  - Identify marshaling requirements for the low-level kernel idioms
  - Identify atomic regions that access shared data
- Prior work
  - *Microdrivers* (isolated the control plane of the driver)

KSplit design choices

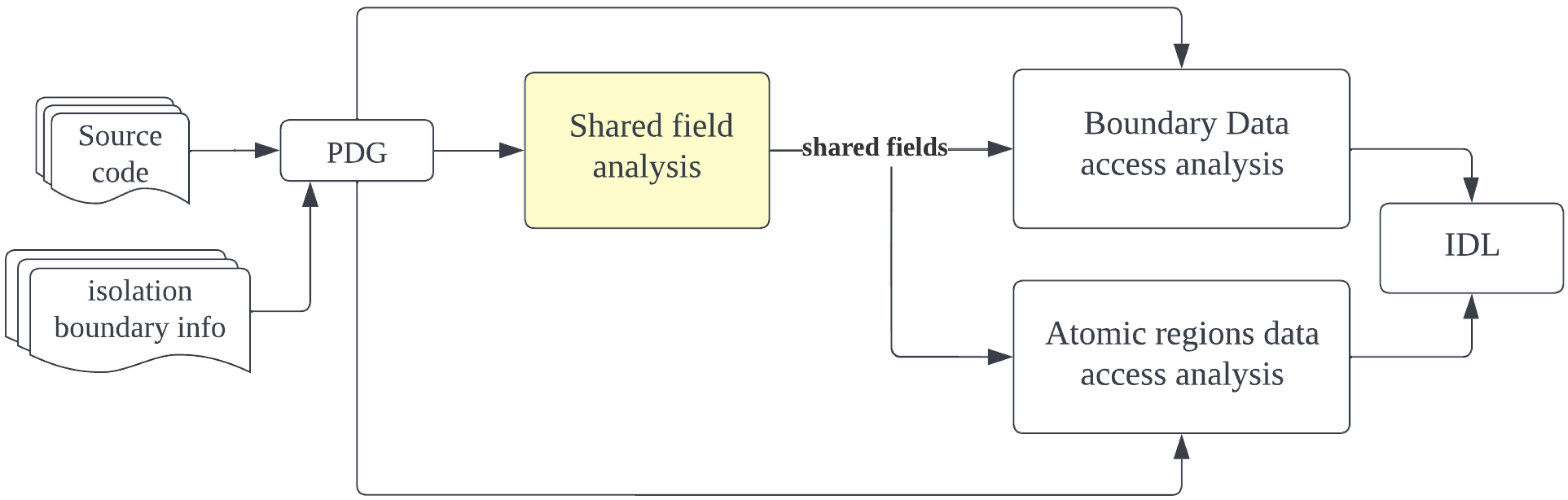
- Kernel is huge
  - Identify the relevant kernel code that the driver interacts with
- Aim to detect all shared data (sound)
  - We might classify some private data as shared
- Aim to infer marshaling requirements for low-level idioms
  - Provide warning for the cases that we cannot infer
- Aim to infer marshaling requirements for shared critical sections
  - Hypothesis: There are not many shared critical sections

KSplit workflow

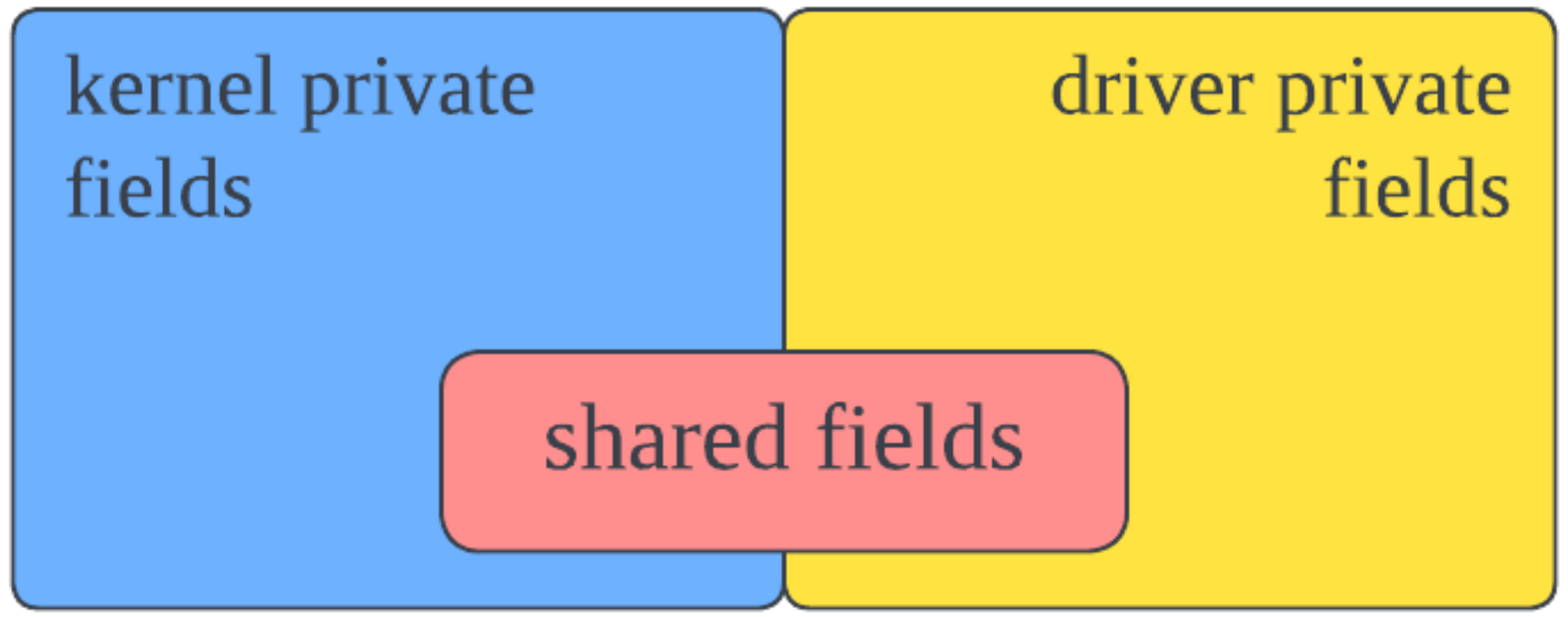


- **Input**: source code of kernel and target isolated driver
- **Output**: IDL file that specifies the communication interfaces and data synchronization requirements

Shared field analysis

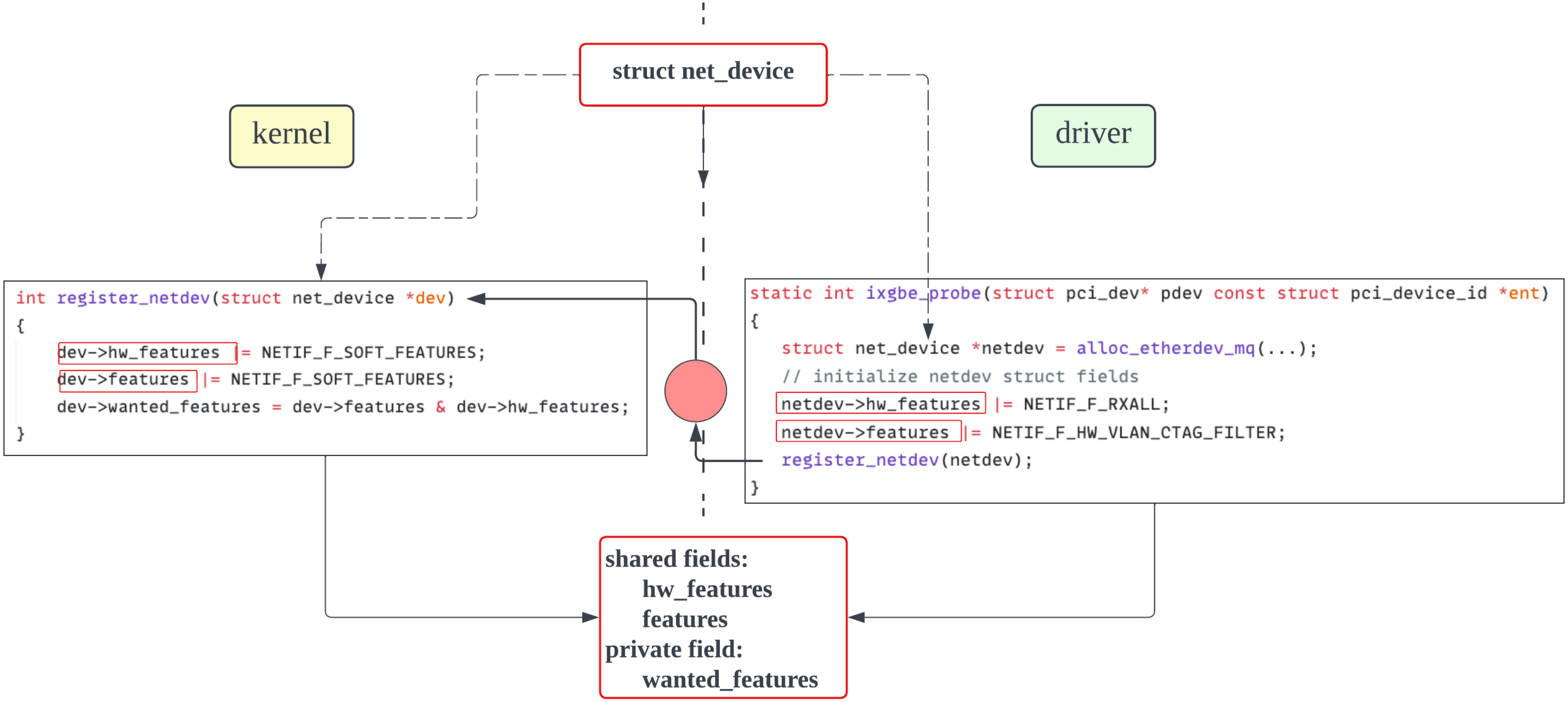


Shared field analysis

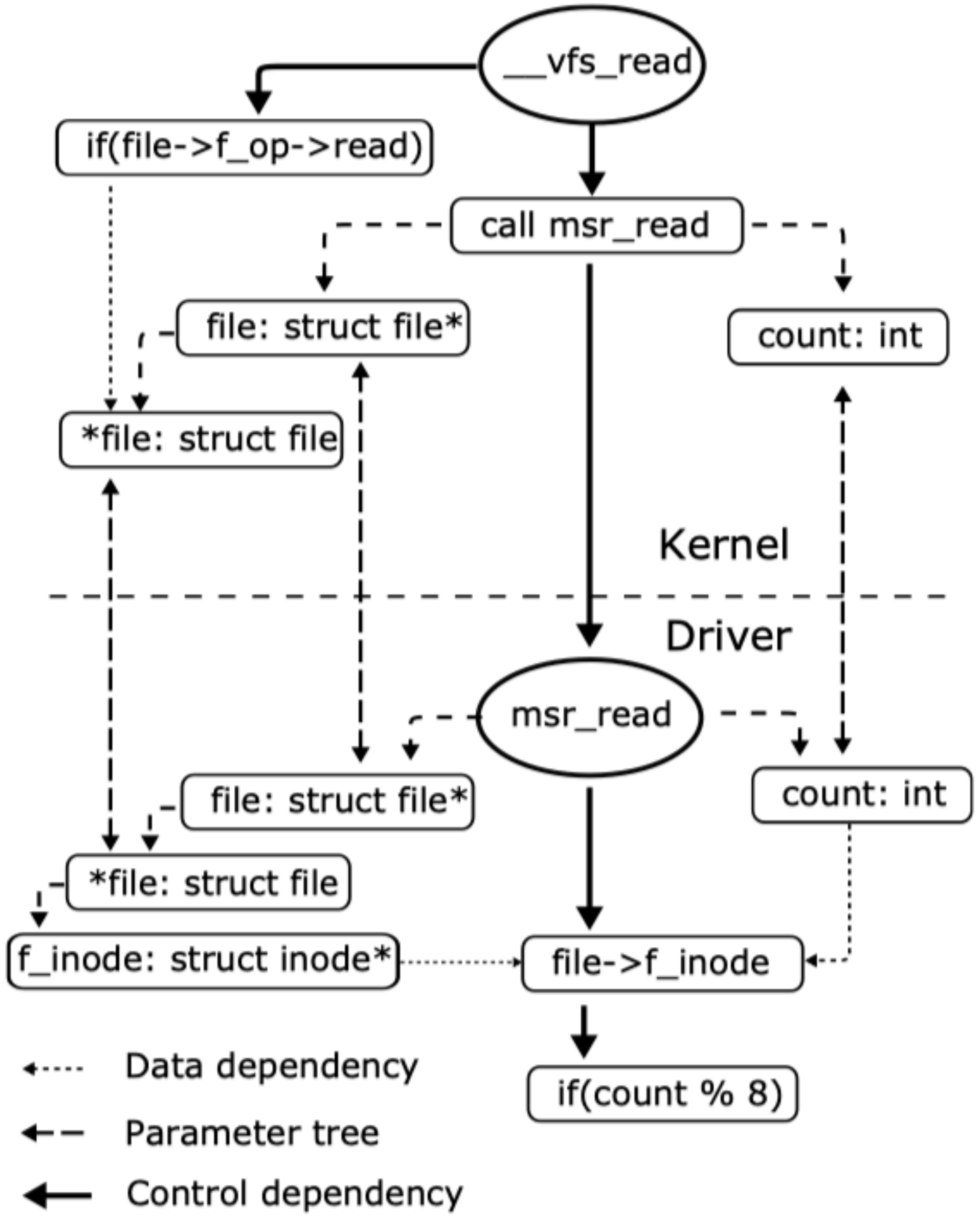


- **Input:**
  - data structure types on all the interface functions for the driver under analysis
- **Output:**
  - the set of struct fields accessed by both the kernel and this driver

Shared field analysis



Program Dependence Graph



- **PDG**: represents program dependencies
  - inter-procedural pointer alias relations
  - field-sensitive
  - data dependencies
  - control dependencies/flow

CCS'17

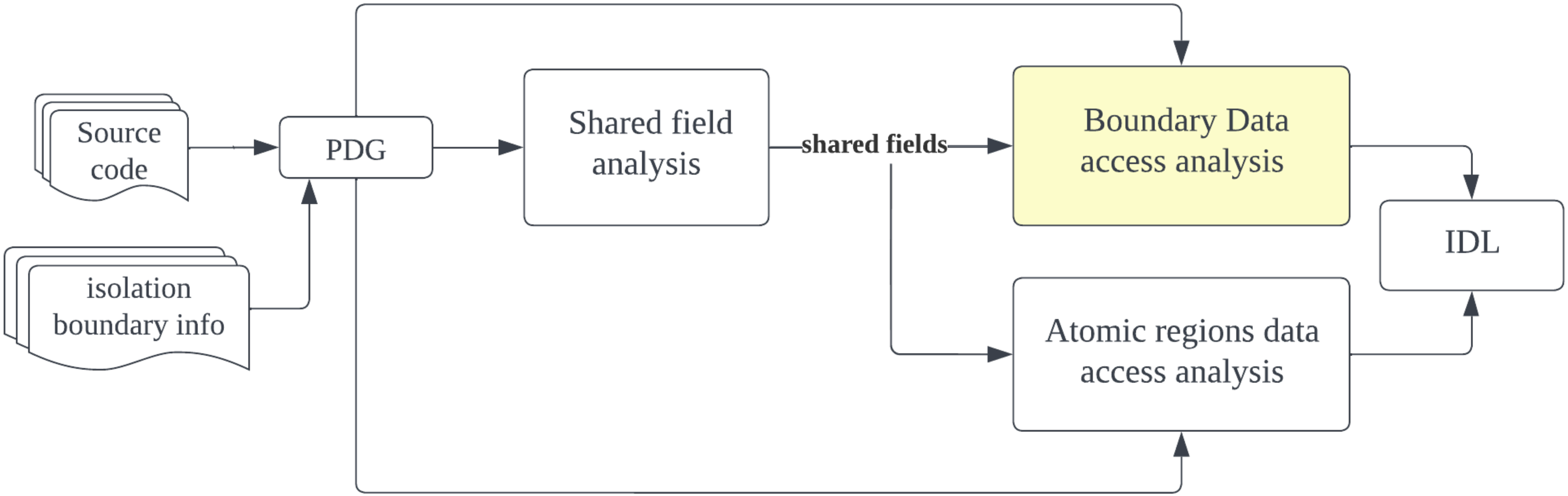
PtrSplit: Supporting General Pointers in Automatic Program Partitioning

Shen Liu
The Pennsylvania State University
University Park, PA
sxl463@cse.psu.edu

Gang Tan
The Pennsylvania State University
University Park, PA
gtan@cse.psu.edu

Trent Jaeger
The Pennsylvania State University
University Park, PA
tjaeger@cse.psu.edu

Boundary data access analysis



Boundary data access analysis

- **Purpose**:
  - Infer synchronization requirements for every interface call and return
- **How**:
  - figure out the subset of shared fields that are read/written in an interface function
  - synchronize data **read by callee** at the function call
  - synchronize data **updated by callee** at the function return

Boundary Data Access Analysis: example

kernel | **driver**

call edge
return edge

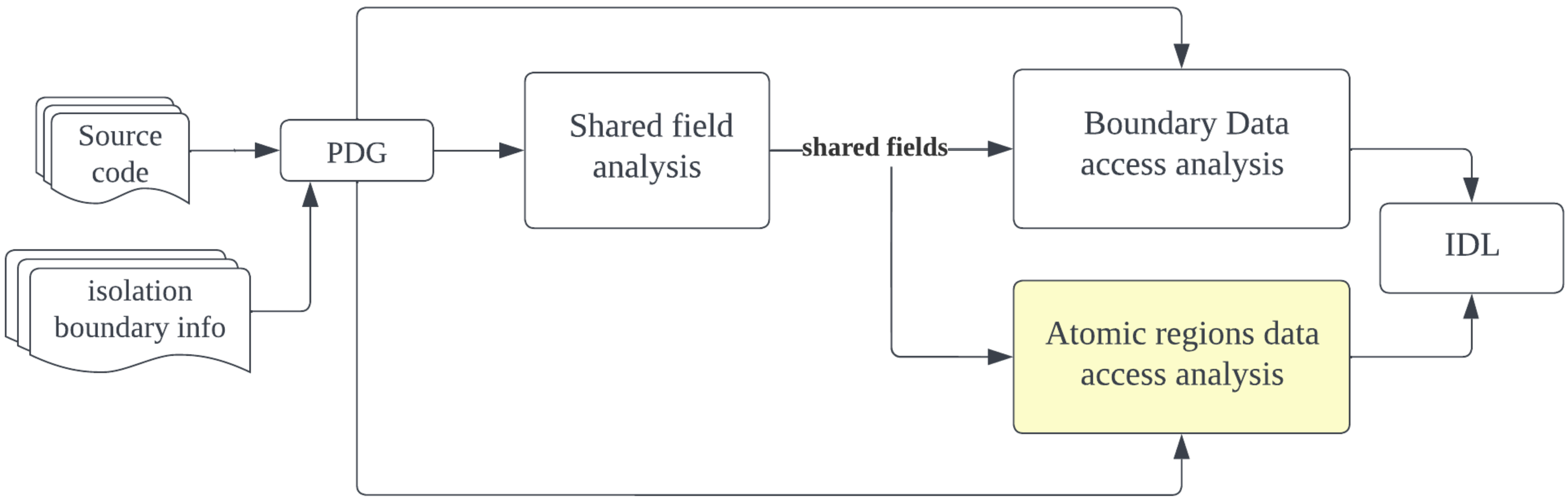
```
/* ixgbe driver */
int ixgbe_probe(struct pci_dev *pdev, ...) {
  struct net_device *dev = alloc_etherdev(...);
  /* initialize struct fields */
  dev->features |= NETIF_F_GSO_PARTIAL;
  dev->hw_features |= NETIF_F_HW_TC;
  register_netdev(dev);
  ...
}
```

sync fields:
hw_features
features

```
/* kernel code */
int register_netdev(struct net_device *dev) {
  dev->features |= NETIF_F_SOFT_FEATURES;
  dev->wanted_features = dev->features & dev->hw_features;
  if (dev->hw_features & NETIF_F_TSO)
    dev->hw_features |= NETIF_F_TSO_MANGLED;
  ...
  return ret;
}
```

sync fields:
hw_features
features

Atomic Region Analysis



Atomic Region Analysis

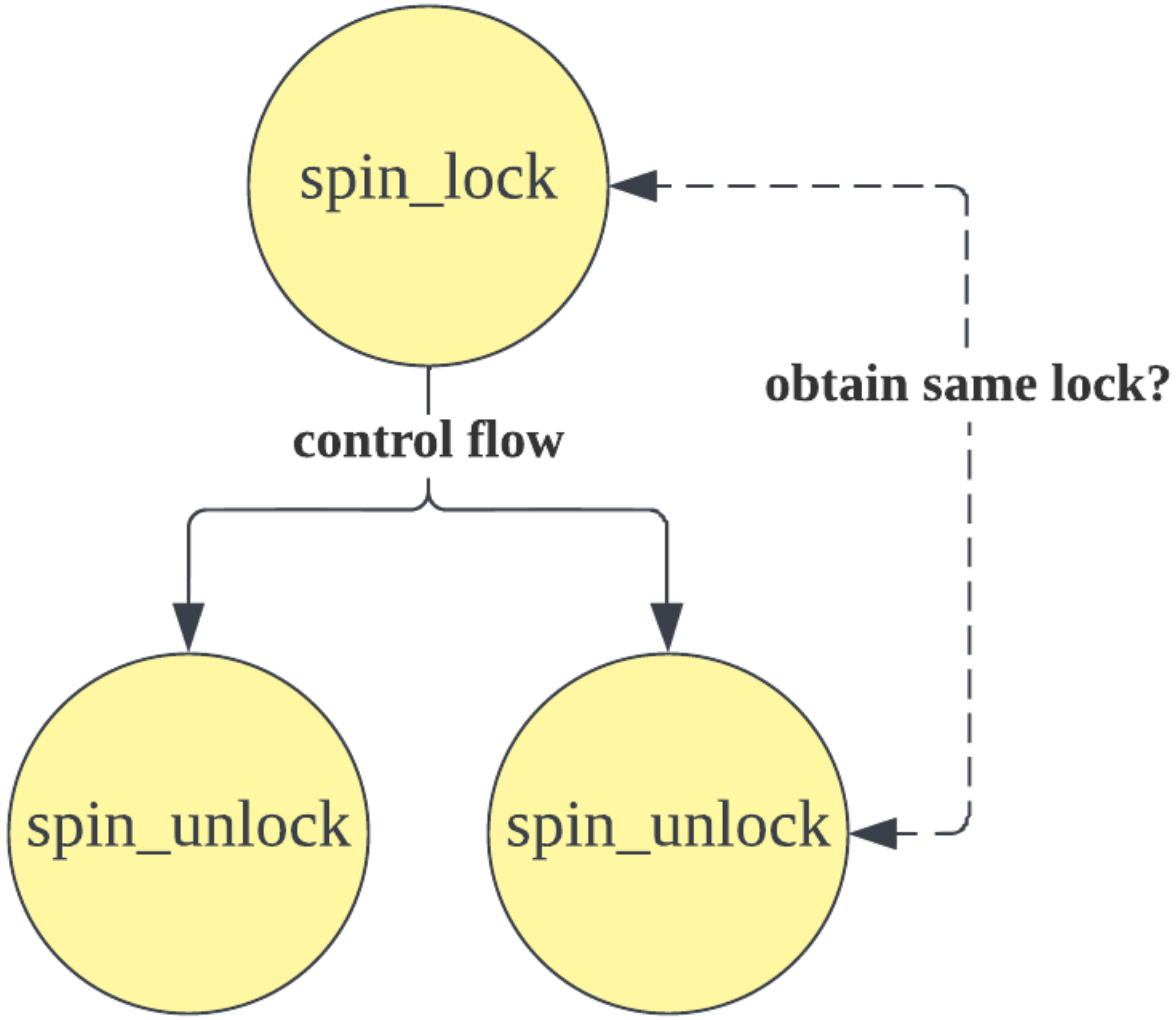
- **Purpose**:
  - Find shared data accessed within the atomic regions
  - Infer synchronization requirements for shared atomic regions
- **When to synchronize**:
  - after/before the **entry/exit** of each atomic region.

```
spin_lock(&lock)
   sync_call(read data)
   ...
   sync_call(updated data)
spin_unlock(&lock)
```

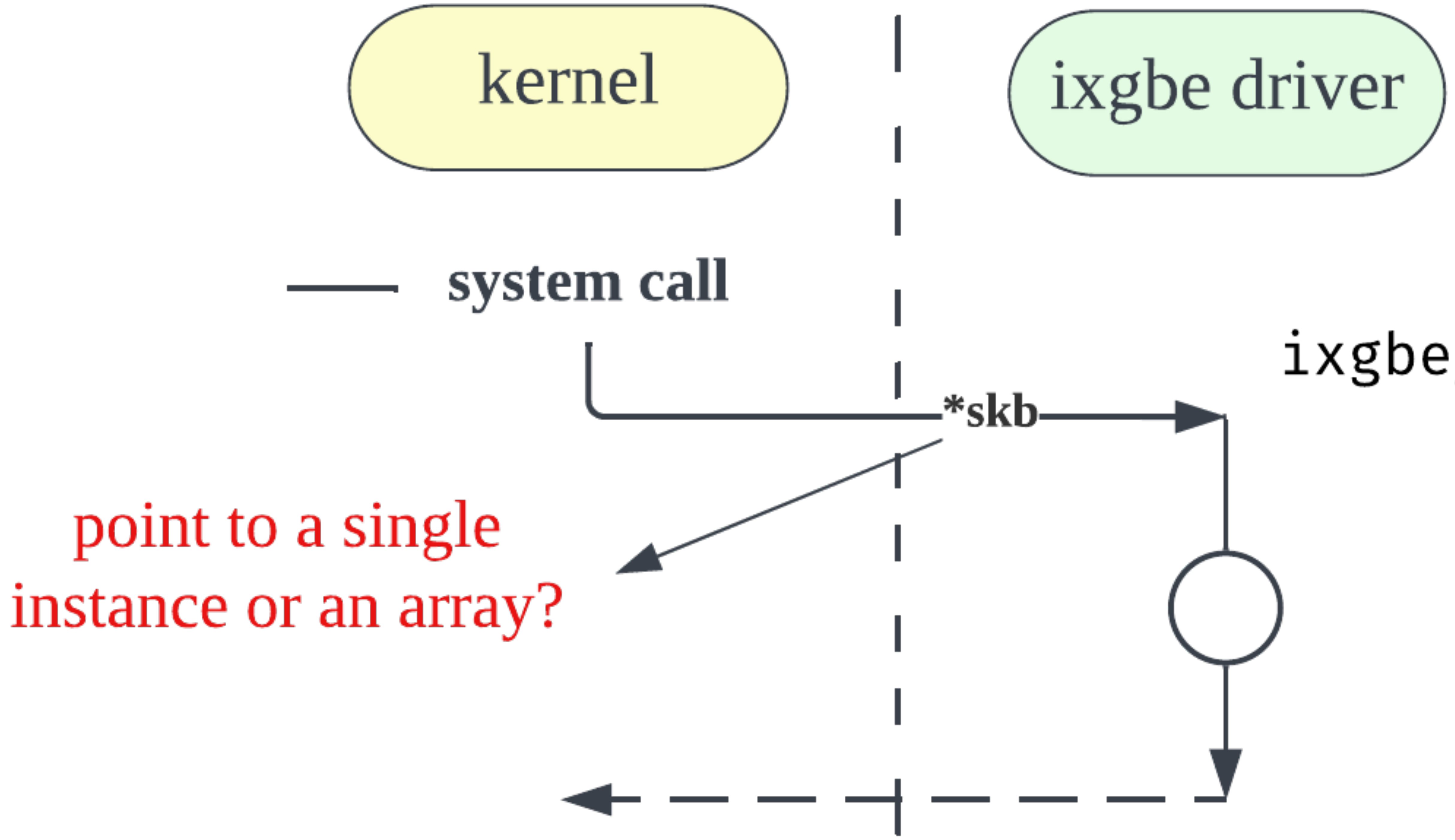
Atomic Region Analysis

- Compute atomic regions using control flow graph

```
spin_lock(&lock)
  ...
spin_unlock(&lock)
```

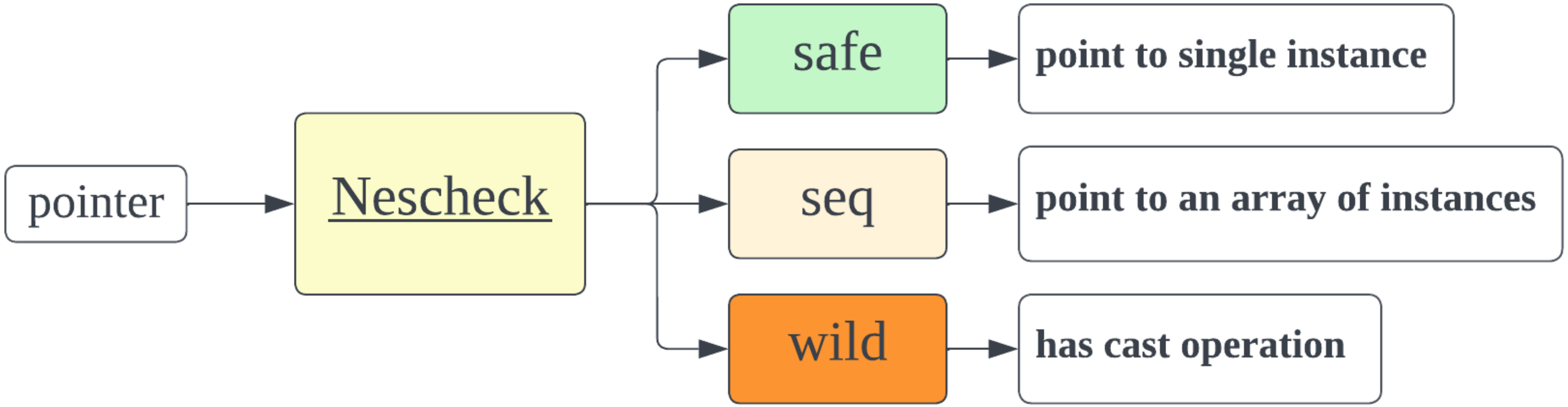


Infer marshaling requirements for pointers




```
ixgbe_xmit_frame(struct sk_buff *skb, ...)
```

Classify Pointers with Nescheck



POPL'2002

CCured: Type-Safe Retrofitting of Legacy Code

George C. Necula Scott McPeak Westley Weimer
University of California, Berkeley
{necula,smcpeak,weimer}@cs.berkeley.edu

AsiaCCS'17

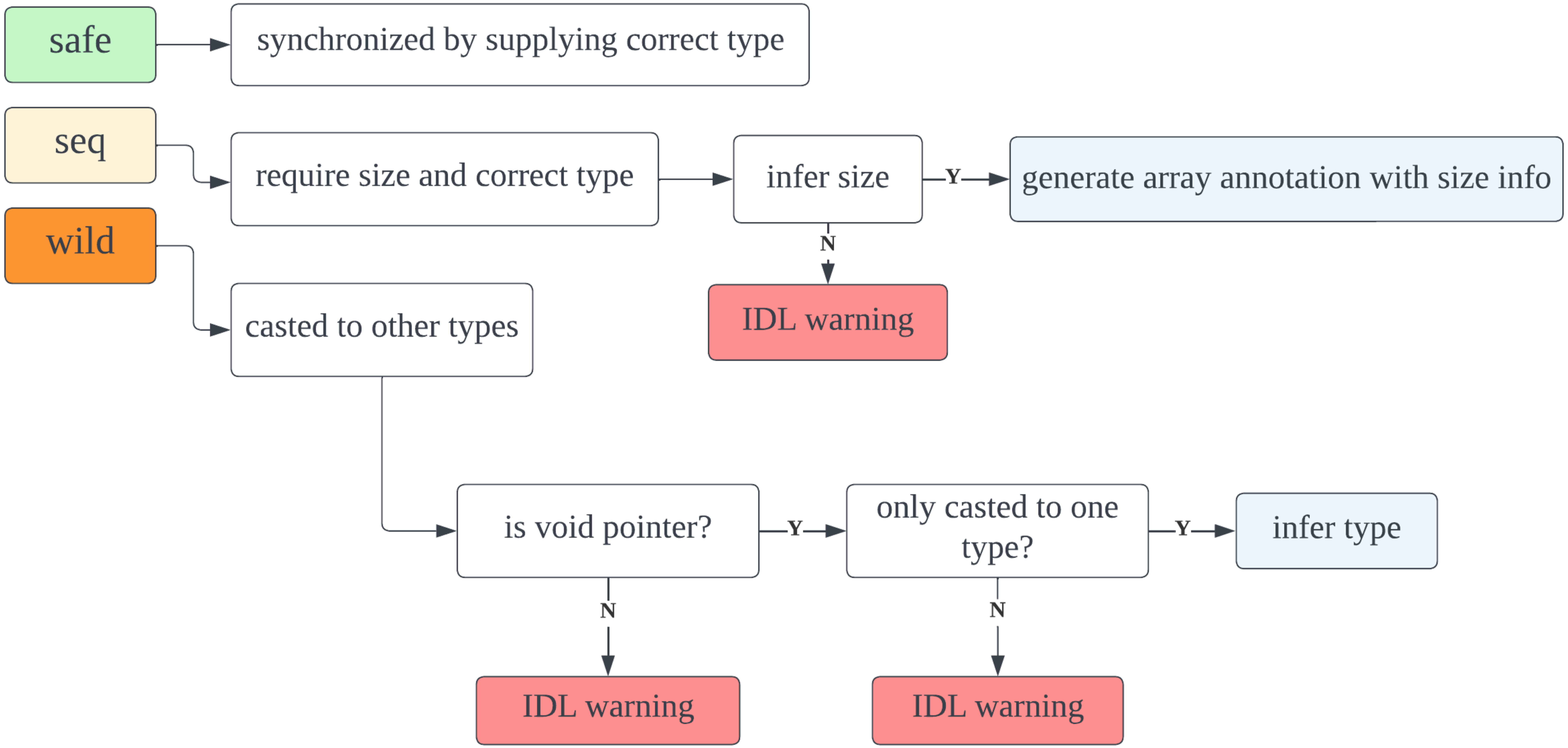
Memory Safety for Embedded Devices with nesCheck

Daniele Midi
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
dmidi@purdue.edu

Mathias Payer
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
mpayer@purdue.edu

Elisa Bertino
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
bertino@purdue.edu

Classify Pointers with Nescheck

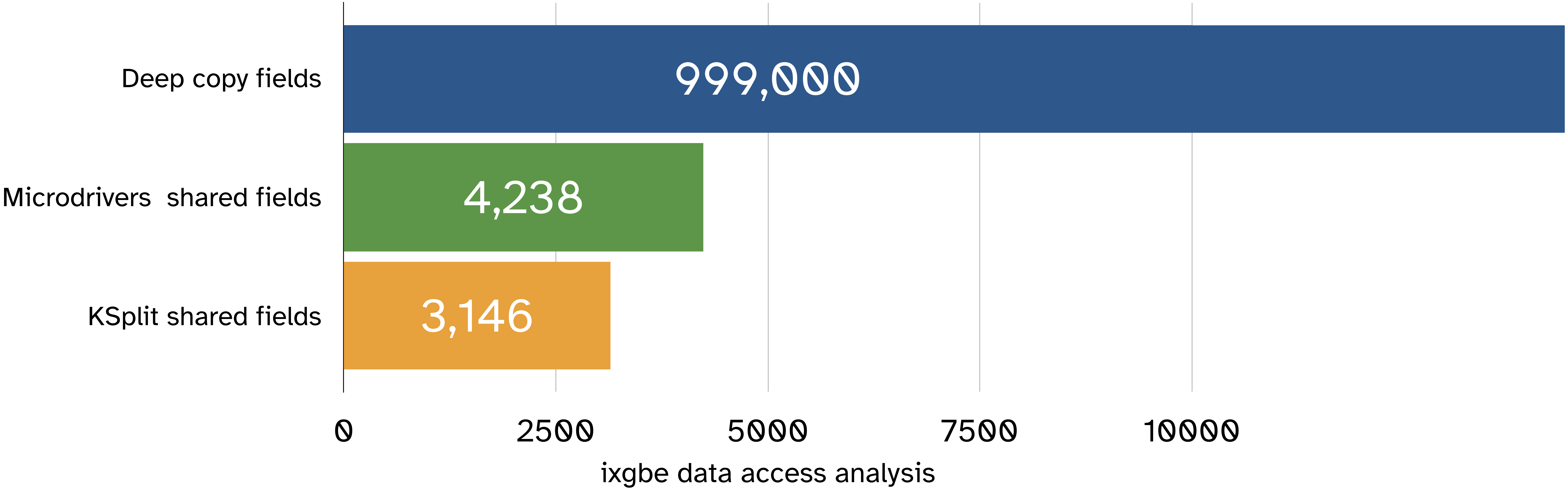


Evaluation

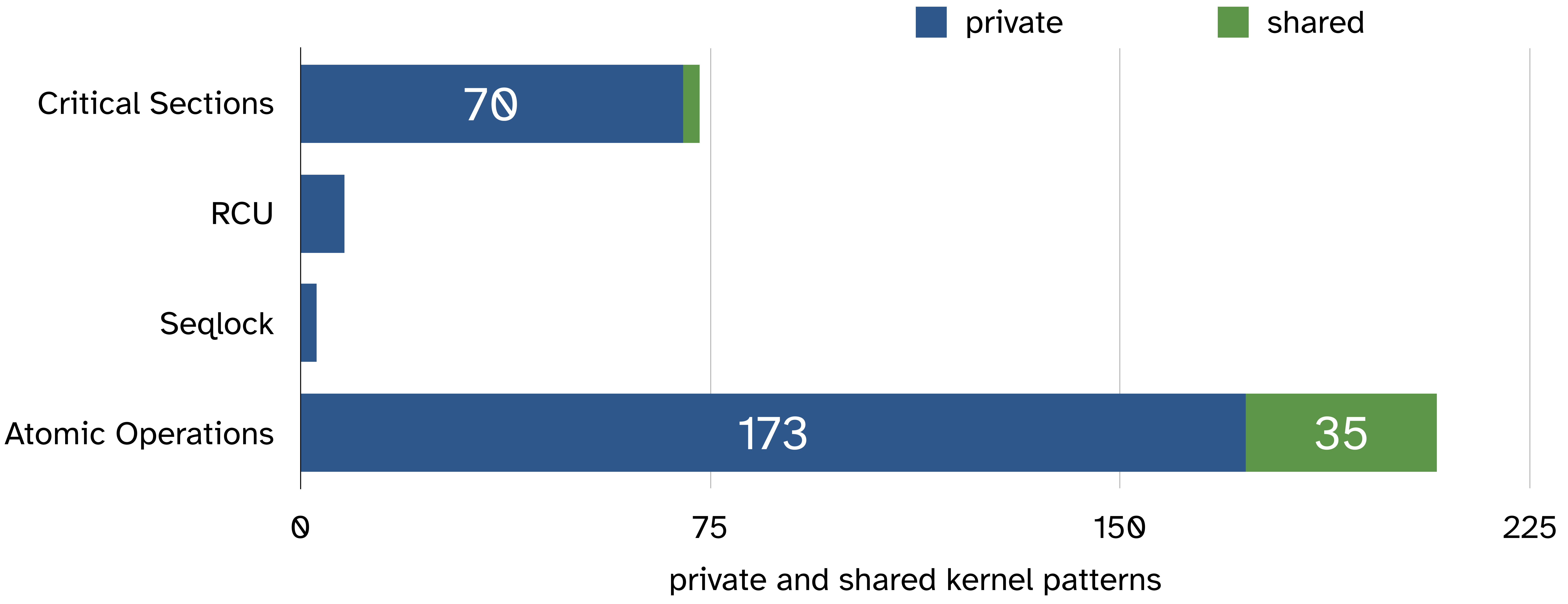
- **Research questions:**
  - How much data synchronization can we reduce?
  - How much manual work required?
  - How to test correctness of the isolated drivers?
- Compare to *Microdrivers*
- Run KSplit on **354** drivers from **9** subsystems
- Fully isolate and 10 drivers and validate the correctness
- **Performance Overhead**:
  - Memcached benchmark

Case study: Ixgbe driver

Ixgbe: data synchronization optimization



Ixgbe: synchronization primitives



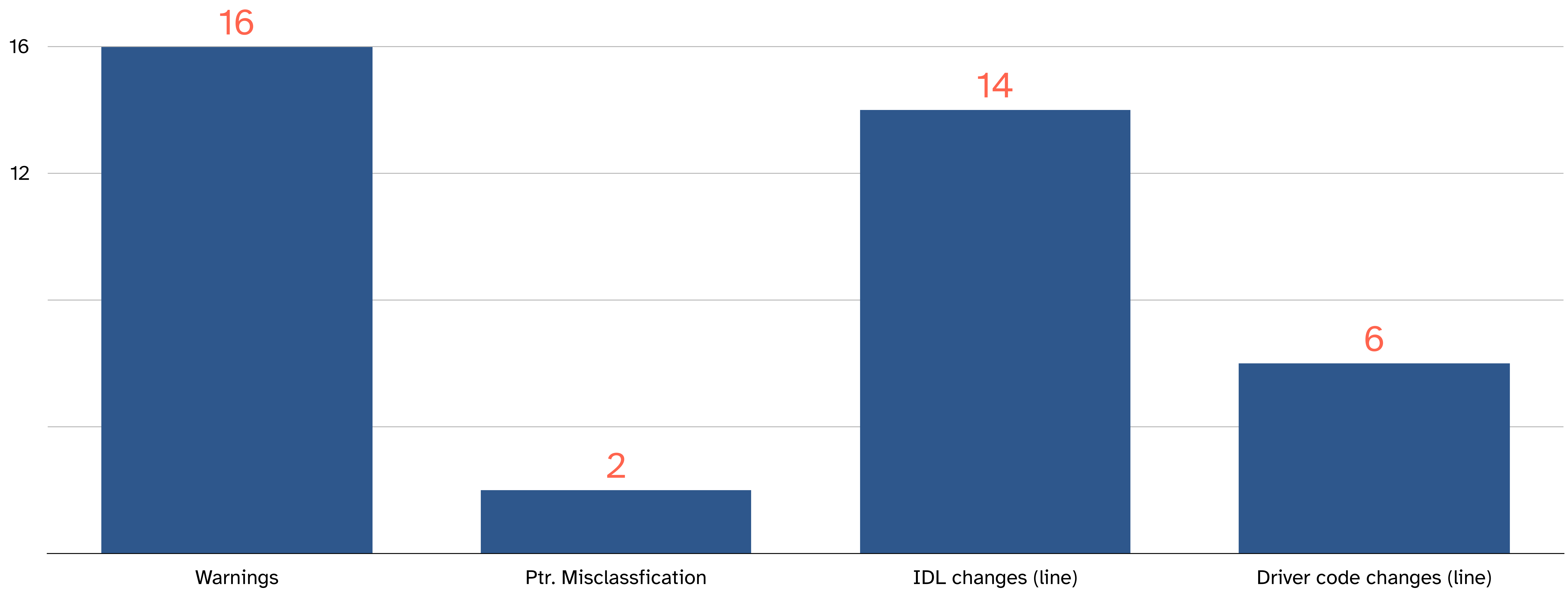
Ixgbe: pointer classification

	singleton	array	string	wild pointer (void)	wild pointer (other)
manual	0	27	0	1	3
handled	1261	92	2	142	1

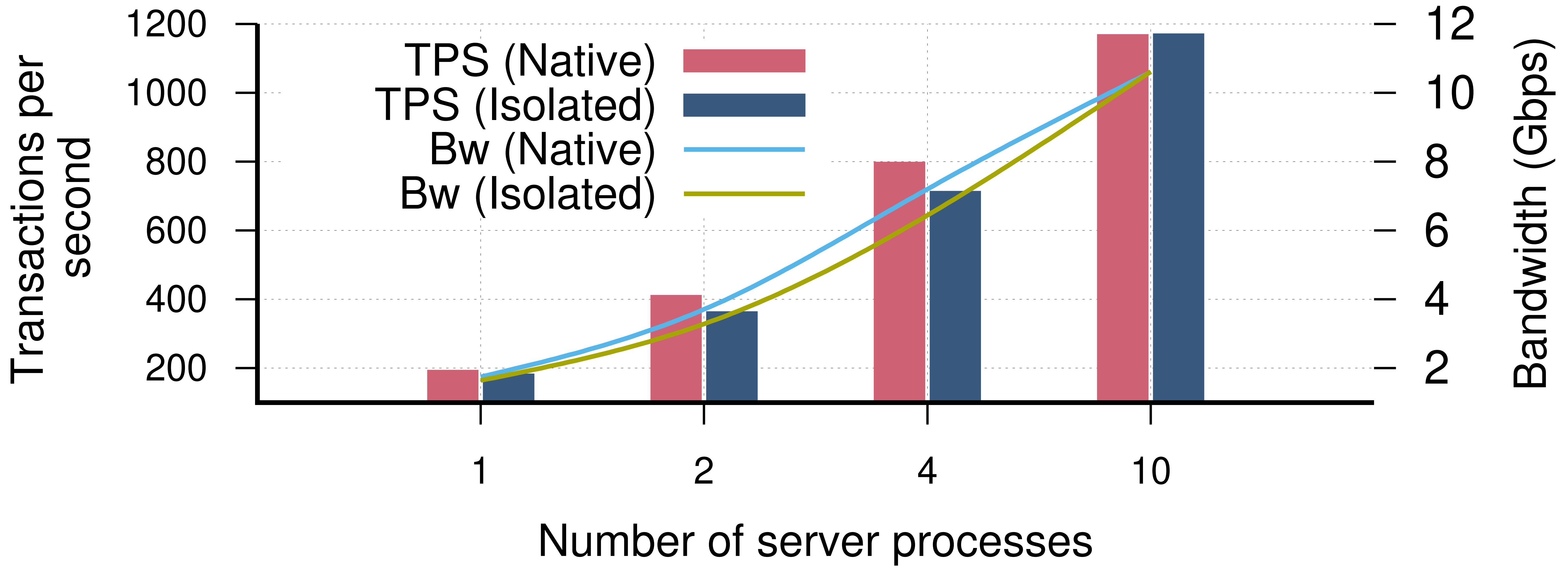
Ixgbe: Manual work

- Source code - 27,000 lines
- Generated IDL spec - 2000 lines
- Pointer misclassifications - 7
- Warnings - 65 (33 anonymous unions, 16 arrays, wild pointers)
  - IDL (changes) - 53 lines
  - Driver (changes) - 19 lines

Manual Work (average across isolated drivers)



Performance overhead: memcached



VEE'20

Lightweight Kernel Isolation with Virtualization and VM Functions

Vikram Narayanan
University of California, Irvine

Yongzhe Huang
Pennsylvania State University

Gang Tan
Pennsylvania State University

Trent Jaeger
Pennsylvania State University

Anton Burtsev
University of California, Irvine

- Memcached/memaslap
- 64B keys, 1024B values (90% set, 10% get)
- We report the bandwidth and transactions per second
- For 1-4 threads, KSplit overhead (5-18%)
- With 10 threads, we saturate the network bandwidth

Conclusions

- We are moving closer to low-overhead isolation mechanisms
- Complexity of isolation becomes a major challenge
- Static analysis framework with small manual effort

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

The source code is available at: https://github.com/ksplit/ksplit-artifacts

Thank you