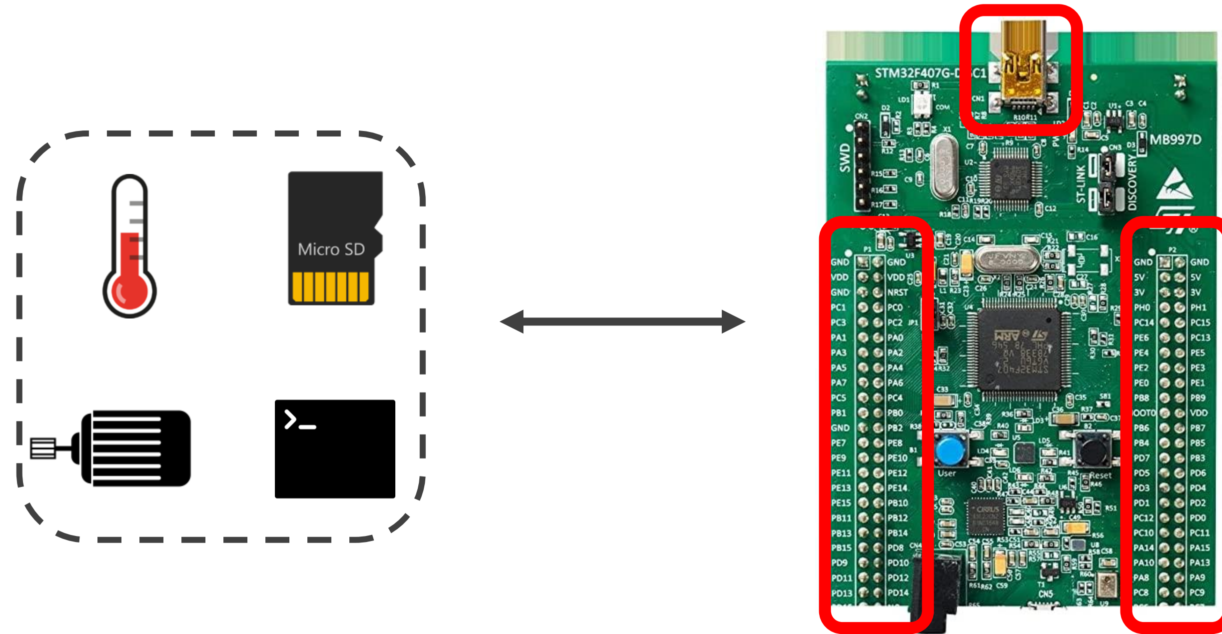
PASAN: Detecting Peripheral Access Concurrency Bugs within Bare-metal Embedded Applications

Taegyu Kim[1], Vireshwar Kumar[2], Junghwan Rhee[3], Jizhou Chen[1], Kyungtae Kim[1], Chunghwan Kim[4], Dongyan Xu[1], Dave (Jing) Tian[1]

[1]Purdue University, [2]Indian Institute of Technology Delhi, [3]University of Central Oklahoma, [4]University of Texas at Dallas

Background

- Peripheral Devices in Embedded Systems



- Attached to MMIO ports (e.g., UART, USB, SPI, etc)

Concurrency Bugs for "Transactions"

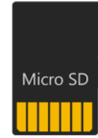
```
lock(&lock_var);
count++; // global variable
unlock(&lock_var);
```

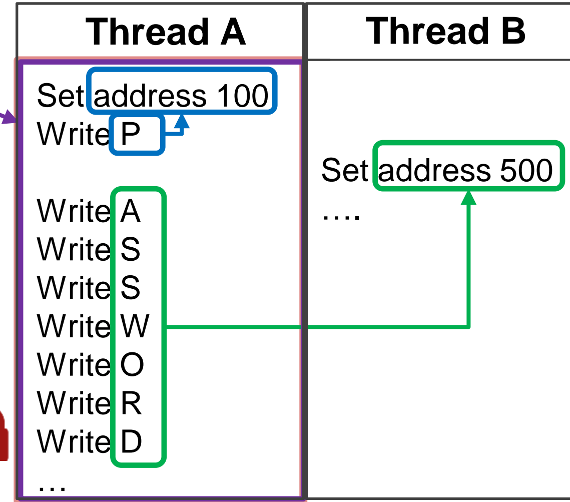
Thread A	Thread B
Read count	
	Read count
Write count + 1	
	Write count + 1

- "count" → Shared resource
- Atomic update necessary

Example of Transaction

Data update on SD card





- Peripheral devices → Shared resources
- Two aspects of atomicity guarantee
  - Spatial (address-range) atomicity
  - Temporal (concurrent execution) atomicity

Otherwise, data corruption happens (e.g., file content, sensor value, etc)

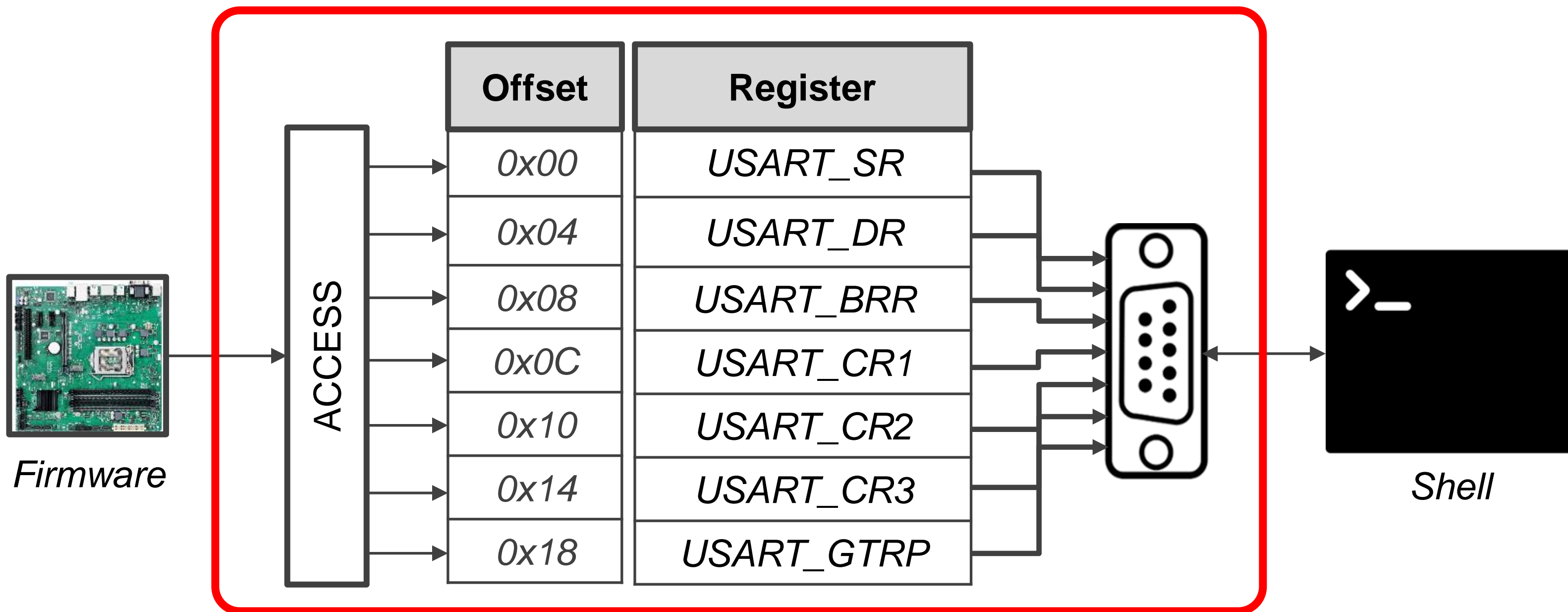
Background: Spatial Atomicity

UART Start (Base) Address: *0x40004400*

UART End Address: *0x4000441C*

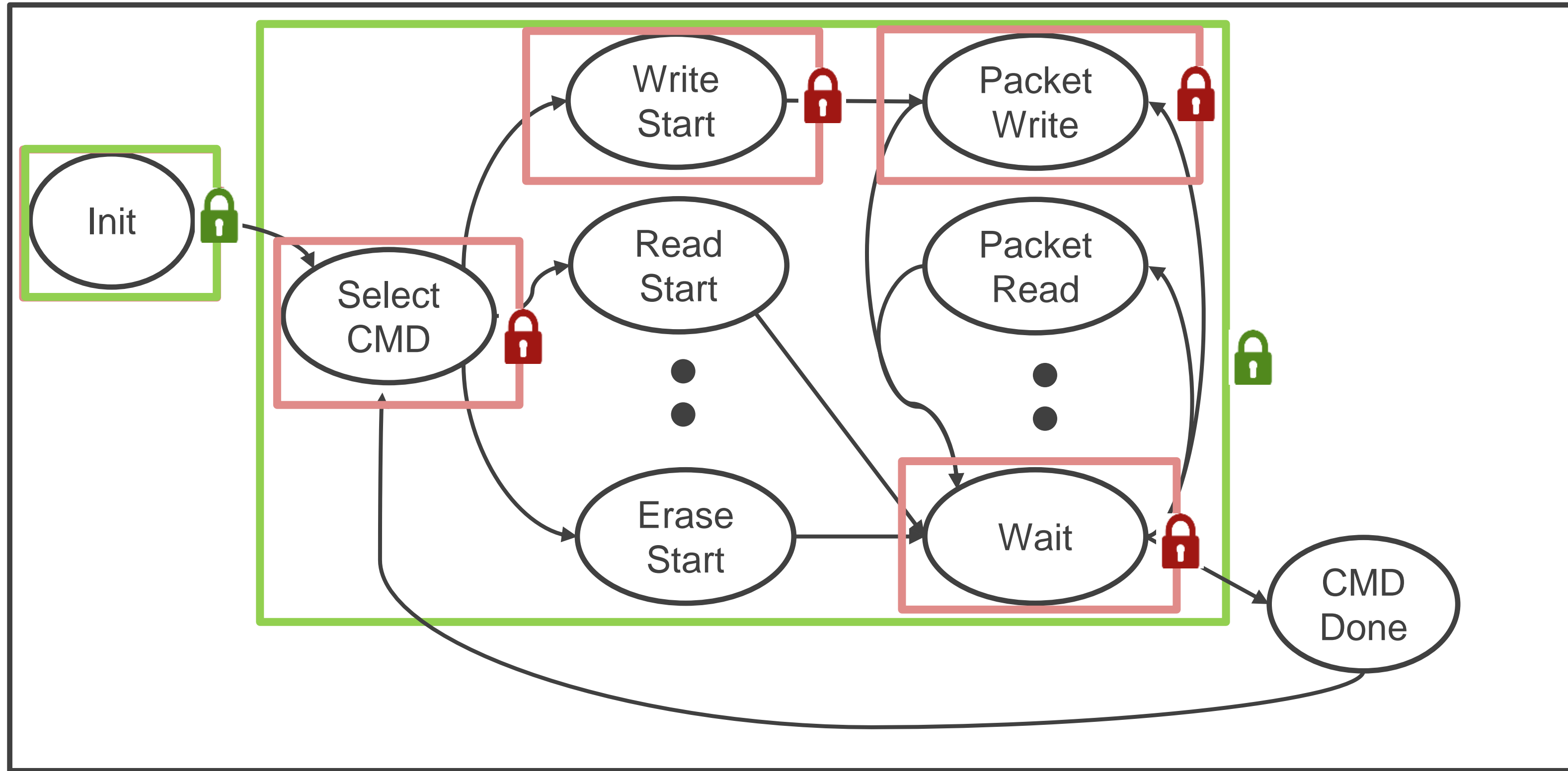
= One MMIO Region

i.e., Spatial Atomicity



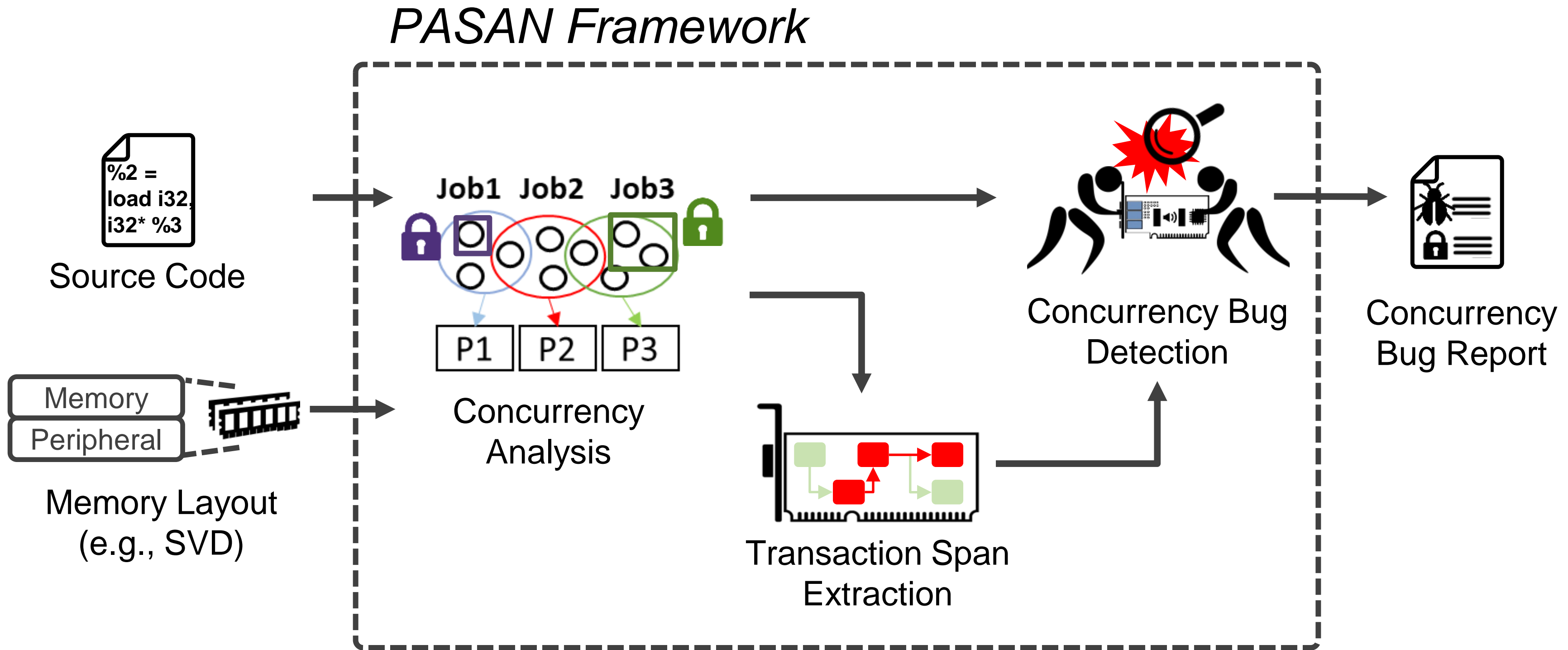
Background: Temporal Atomicity

Simplified SD Card Controller



"Temporal atomicity" should be guaranteed

PASAN Workflow



MMIO and Lock Alias Identification

Start Address:	*0x40047400*
End Address:	*0x4004741C*

```
void spi_write(){
    …
    lock(&lock1);
    *0x40047400 = 0x10;
    unlock(&lock1);
    ….
    lock(&lock2);
    *0x40047404 = 0x20;
    unlock(&lock2);
    ….
}
```

Within Identical MMIO Region

```
uint32_t *mmio_ptr = 0x40047400;
mutex_t lock;

void spi_write1(){
    …
    lock(&lock);
    *(mmio_ptr) = 0x10;
    unlock(&lock);
    ….
}
void spi_write2(){
    lock(&lock);
    *(mmio_ptr + 4) = 0x20;
    unlock(&lock);
    ….
}
```

Lock Span Identification

```
void spi_write(){
    …
    lock(&lock1);
    *0x40047400 = 0x20;
    unlock(&lock1);
    ….
    lock(&lock2);
    *0x40047404 = 0x10;
    unlock(&lock2);
    ….
}
```

Lock Span 1

Lock Span 2

```
void spi_write(){
    lock(&lock);
    spi_write1();
    ….
    spi_write2();
    unlock(&lock);
    ….
}
void spi_write1(){
    …
    *0x40047400 = 0x10;
    ….
}
void spi_write2(){
    ….
    *0x40047404 = 0x20;
    ….
}
```

Lock Span

Lock Span

Call Context -Aware

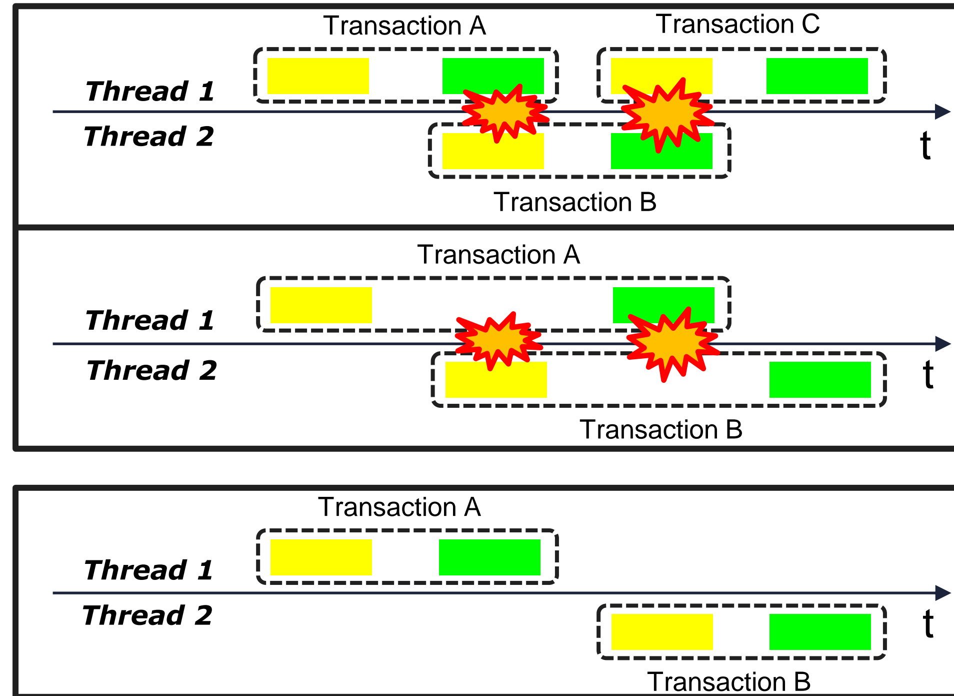
Transaction Span Analysis

```
void spi_cmd(){
    …
    lock(&lock1);
    *0x40047400 = 0x10;
    ….
    9 instructions
    ….
    *0x40047404 = 0x20;
    unlock(&lock2);
    ….
}
```

- Distance-based Span Detection
  - Count # of instructions between MMIO access instructions
  - e.g., 10 instructions < threshold
- One exception example we should consider
  - "Wait state" after a job request
    - e.g., sleep or waiting for job completion (i.e., interrupt)
  - What if "wait state" is interrupted by another MMIO access?
    - e.g., Ongoing job can fail

Concurrency Bug Detection

```
void spi_write(){
    lock(&lock1);
    *0x40047400 = 0x20;
    unlock(&lock1);
    ....
    lock(&lock2);
    *0x40047404 = 0x10;
    unlock(&lock2);
}
```



lock1 != lock2

Single lock

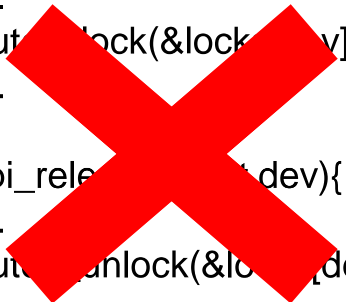
Single lock covering each transaction

Result: Bug Discovery

Platform	# of Bugs	# of True Positive Bugs	# of False Positive Bugs
ArduPilot	20	8	12
RaceFlight	0	0	0
RIOT	9	8	1
Contiki	0	0	0
TS100	1	1	0
grbl	0	0	0
rusEFI	6	0	6
Total	36	17	19

Case Study

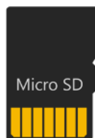
- Security-Critical Data Corruption

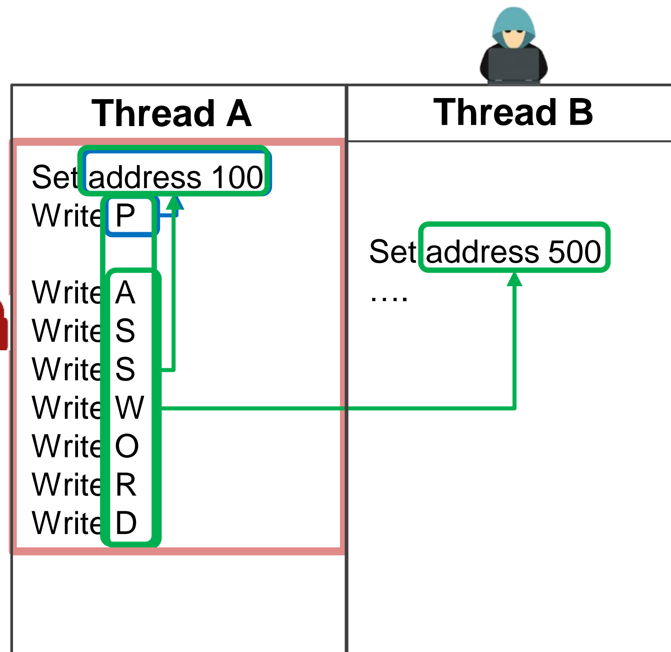


```
int spi_acquire(spi_t dev){
    ....
    mut    lock(&lock    v]);
    ....
}
int spi_rele       dev){
    ....
    mut    unlock(&lo    [dev]);
    ....
}
```

```
int sdcard_spi_write();
int sdcard_spi_read();
....
```

Data update on SD card





Discussion

- Limitations Inherited from Static Analysis
  - False positives and negatives caused by points-to analysis
  - Cannot handle individual interrupts
- Using Incorrectly Inferred Transaction Spans
  - Cause false positives and negatives
  - Useful to detect “transaction-level” concurrency bugs

Conclusion

- Concurrency bugs in embedded platforms
  - Threat to system safety and security
- PASAN: a static analysis-based concurrency bug detector
  - Targeting peripheral devices
  - Transaction-aware
    - Considering spatial and temporal atomicity
- 17 bugs on 7 real-world embedded applications

Thank you!

This work was supported in part by ONR Grant #N00014-17-1-2045

tgkim@purdue.edu