

# FloatZone

Accelerating Memory Error Detection  
using the Floating Point Unit

Floris Gorter, Enrico Barberis,  
**Raphael Isemann**, Erik van der Kouwe,  
Cristiano Giuffrida, Herbert Bos

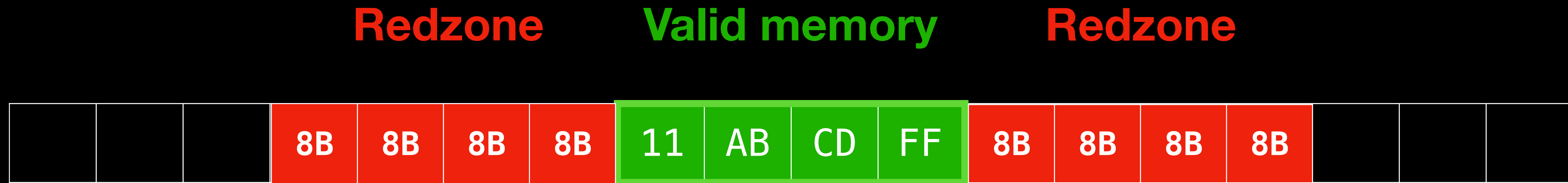


# Memory Errors

- Problem: Is this memory access safe?
- The pointer could be...
  - **out of bounds**
  - **point to free'd memory**
- (Hidden) **security issue!** 💣
- How can we detect this?

```
int *ptr = ...  
*ptr = 123; // safe?
```

# Error Detection with Redzones



Check:

```
int m = *ptr;  
if (m == 0x8b8b8b8b) {  
    error_and_exit();  
}
```



Can we go faster than this?

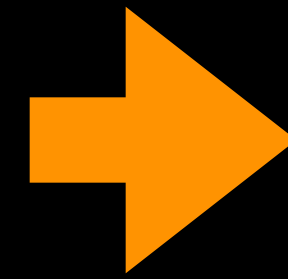
Original code:

```
*ptr = 123;
```

# Encoding Checks

## Check Logic

```
int m = *ptr;  
if (m == 0x8b8b8b8b) {  
    error_and_exit();  
}
```



## Instructions

```
// load  
mov eax, [rdi]  
// compare  
cmp eax, ERRVAL  
// branch  
je .error
```

# What are the Ideal Instructions?

## 1. Use an **underutilized part of the CPU**

- For instruction-level parallelism

## 2. Use **high-throughput instructions**

## 3. **Avoid the branch predictor**

- Error branch effectively never taken
- No miss-speculation
- No resources spent on speculation

```
// load
mov eax, [rdi]
// compare
cmp eax, ERRVAL
// branch
je .error
```

# Floating Point Operations

1. Are **underutilized part of the CPU** ✓

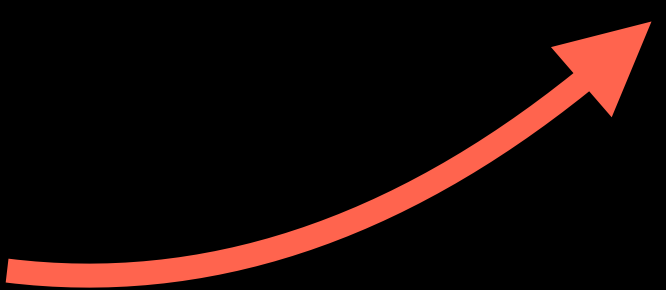
- Light FPU workloads are common

2. Many **high-throughput instructions** ✓

3. **Avoids the branch predictor** ✓

- Can't branch with the FPU!
- But how do encode our jump?

```
// load
mov eax, [rdi]
// compare
cmp eax, ERRVAL
// branch
je .error
```



# Floating Point Exceptions

- FP operations can cause '**exceptions**'
  - Translated into signal to process (SIGFPE)
  - **Redirect control flow without branching!**
- Exception creation depends on operands
  - Effectively it's a conditional jump
  - Can we do **the whole check in one instruction?**
  - We just have to find the right operation + operands...



# **MATH!**

**+ some Brute Force**



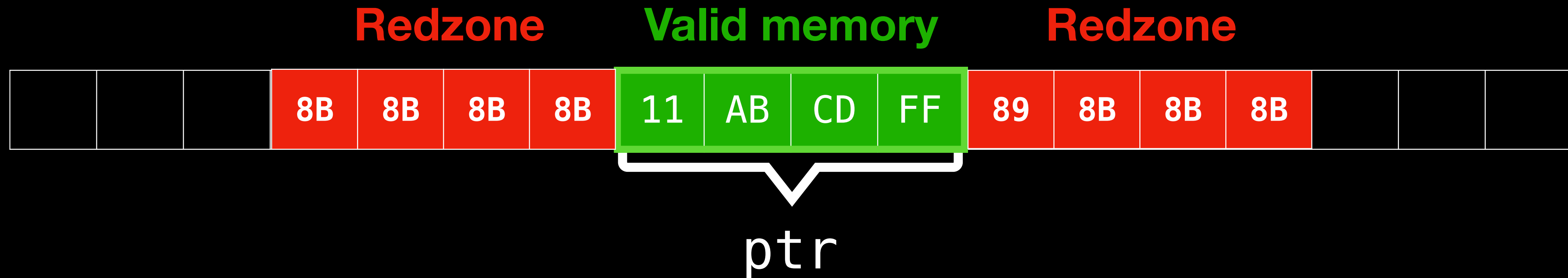
# FloatZone

Initial setup: `// Enable exceptions on subnormal results.  
feenableexcept(FE_UNDERFLOW);`

---



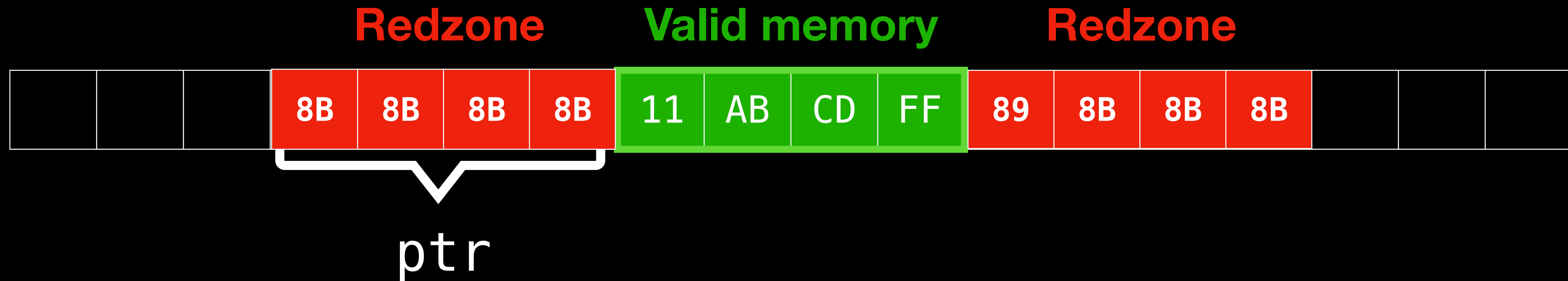
# FloatZone - Checks



```
vaddss xmm15, 0x0b8b8b8a, [ptr]
```

1. (float)0x0b8b8b8a + (float)0xffcdab11 is computed.
2. Outcome: Nothing happens ✓

# FloatZone - Checks



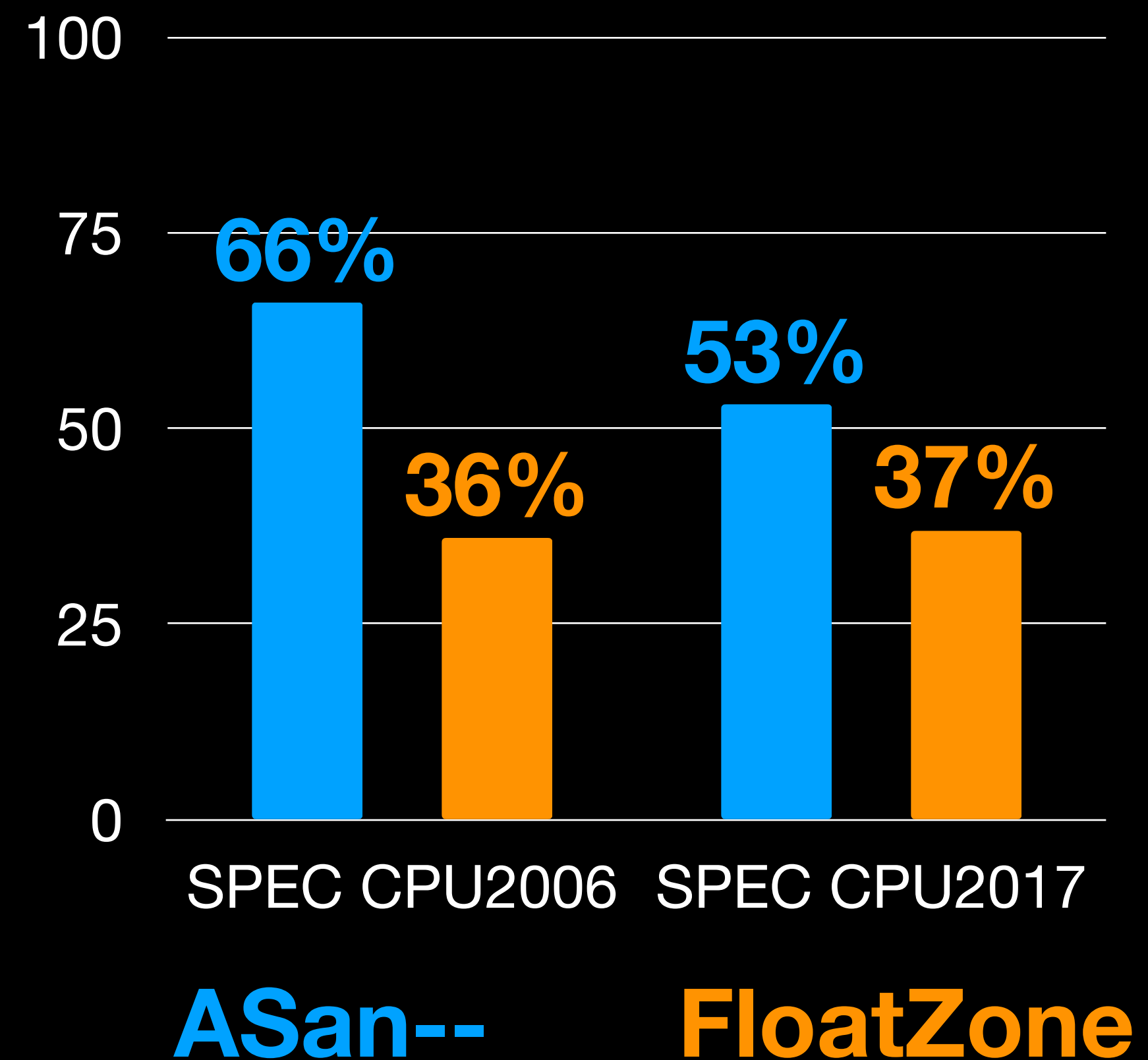
```
vaddss xmm15, 0x0b8b8b8a, [ptr]
```

1. **(float)0x0b8b8b8a + (float)0x8b8b8b8b underflows!**
2. **CPU creates FP exception** → SIGFPE.
3. **SIGFPE handler reports error** (and filters false-positives).

# FloatZone - Overhead

- We built a **full FloatZone sanitizer.**
- On SPEC CPU benchmarks:
  - **about 30% and 16% faster** (compared to ASan--)
- On Fuzzing benchmarks:
  - **72% times higher throughput** (compared to ReZZan)

## Runtime overhead:



# Summary

- We can **express common redzone checks via FP operations.**
- Using FP operations can have **microarchitectural benefits.**
  - **Less interference with branch prediction.**
  - **Higher instruction-level parallelism.**
- **Memory error detection using this technique is notably faster.**

# Questions?

More info at: [vusec.net/projects/floatzone](https://vusec.net/projects/floatzone)

