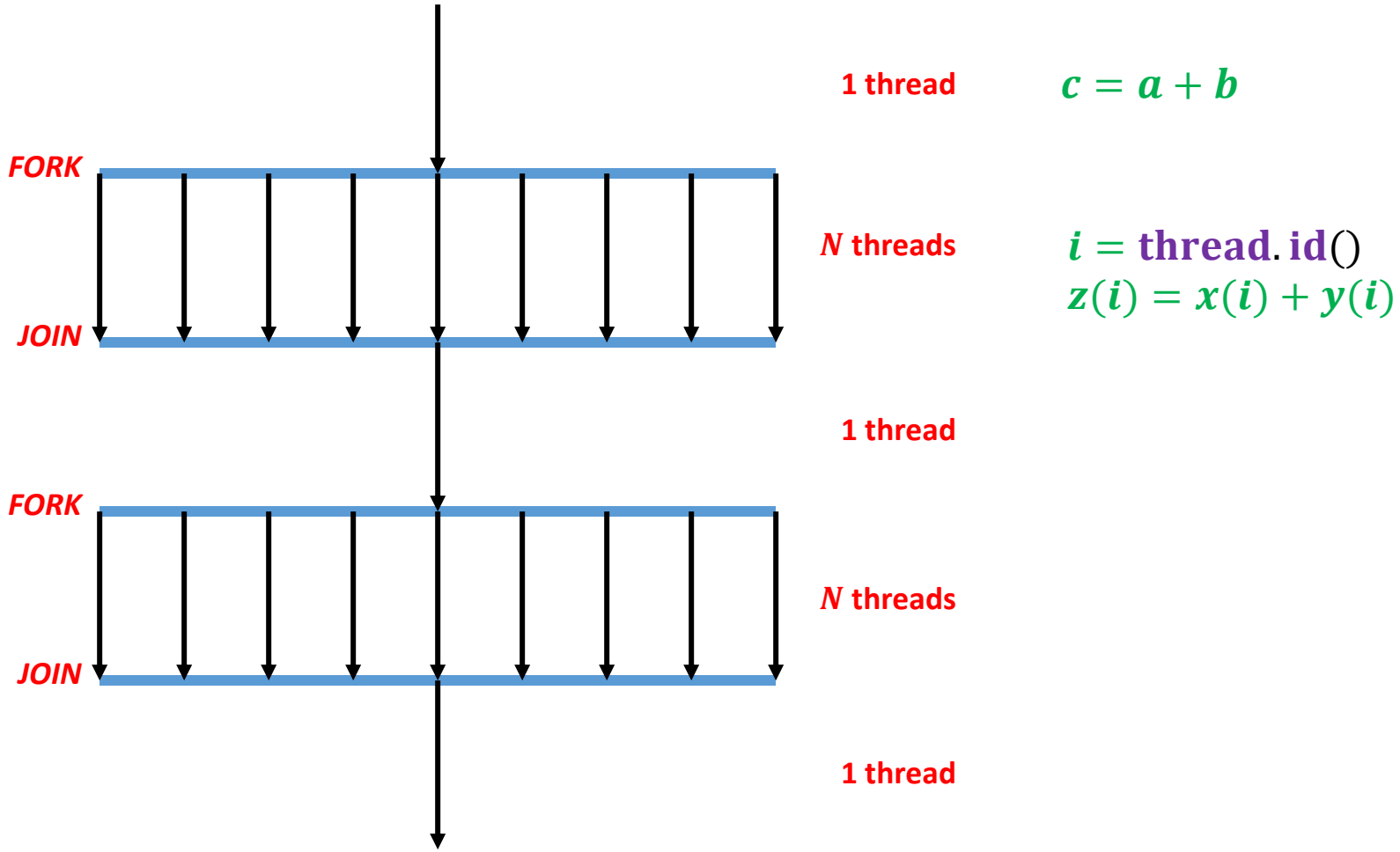
Register requirements for the SIMT model of computation

José Moreira
Chair, RISC-V Vector SIG

Execution model for SIMT

- SIMT execution alternates between sequential regions and parallel regions (similar to fork/join)



- In both kinds of regions, the source code manipulates scalar data (possibly in distinct address spaces)
- The N threads in a parallel region execute in lock step, always at the same instruction
- For any given instruction, a subset of the N threads may be *active*

Examples of SIMT parallel region code

```
i = thread.id()
for j = 0, ..., M − 1
    z(i, j) = x(i, j) + y(i, j)
end
```

```
i = thread.id()
float S = 0
for j = 0, ..., M − 1
    S = S + x(i, j) × y(i, j)
end
z(i) = S
```

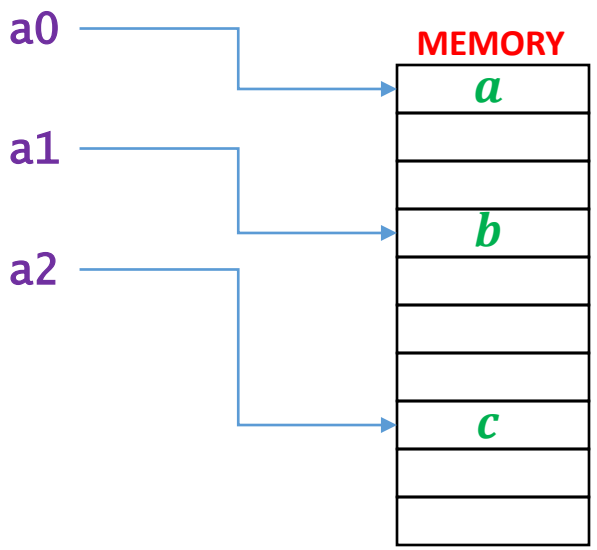
```
i = thread.id()
for j = 0, ..., M − 1
    if (x(i, j) < y(i, j)) then
        z(i, j) = x(i, j)
    else
        z(i, j) = y(i, j)
    end
end
```

- In all cases, each thread is performing scalar operations
- The parallelization already happened, by creating multiple threads

Implementing the SIMT model with vector processing (1)

- If we want to compute $c = a + b$ in a sequential region, with 32-bit floating-point numbers

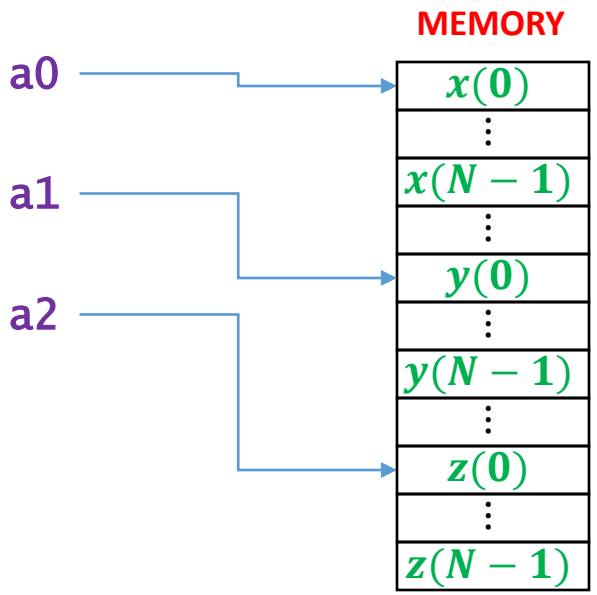
```
flw f0, 0(a0)          # Load single-precision float from memory address in a0 into f0
flw f1, 0(a1)          # Load single-precision float from memory address in a1 into f1
fadd.s f2, f0, f1      # Add f0 and f1, store result in f2
fsw f2, 0(a2)          # Store result from f2 into memory address in a2
```



Implementing the SIMT model with vector processing (2)

- If we want to compute $z(i) = x(i) + y(i)$ in threads $i \in [0, N)$ of a parallel region

```
vle32.v     v0, (a0)        # Load vector from address in a0 into v0
vle32.v     v1, (a1)        # Load vector from address in a1 into v1
vfadd.vv    v2, v0, v1      # Add vectors v0 and v1, store result in v2
vse32.v     v2, (a2)        # Store vector v2 to address in a2
```



- In this example, $\mathrm{LMUL} = 1$ and $N = \frac{\mathrm{vlen}}{32}$, but we don't want N to change with the type

Choosing N

- The more natural choice is to set $N = \text{vlenb}$, the vector register length in bytes
- Different data types require different number of vector registers to store data for all N threads
- In the examples below, let $N = 16$ and $\mathbf{T}(i)$ is the corresponding variable in thread i

Data type	Elements in vector registers
uint8_t	**v0** = T(0) T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8) T(9) T(10) T(11) T(12) T(13) T(14) T(15)
fp16	**v0** = T(0) T(1) T(2) T(3) T(4) T(5) T(6) T(7) **v1** = T(8) T(9) T(10) T(11) T(12) T(13) T(14) T(15)
int32_t	**v0** = T(0) T(1) T(2) T(3) **v1** = T(4) T(5) T(6) T(7) **v2** = T(8) T(9) T(10) T(11) **v3** = T(12) T(13) T(14) T(15)

How many registers do we need?

- It takes 8 vector registers to hold a 64-bit variable from each of the N threads
- A scalar processor has 32×64-bit registers to hold program variables for a single thread
- Keeping the same amount of state for N threads requires $32 \times 8 = 256$ vector registers
- Maybe we can save some registers since not all data types are 64-bit
- On the other hand, scalar processors often have separate integer/address and floating-point registers
- When planning for more vector registers, we should target 256 registers!