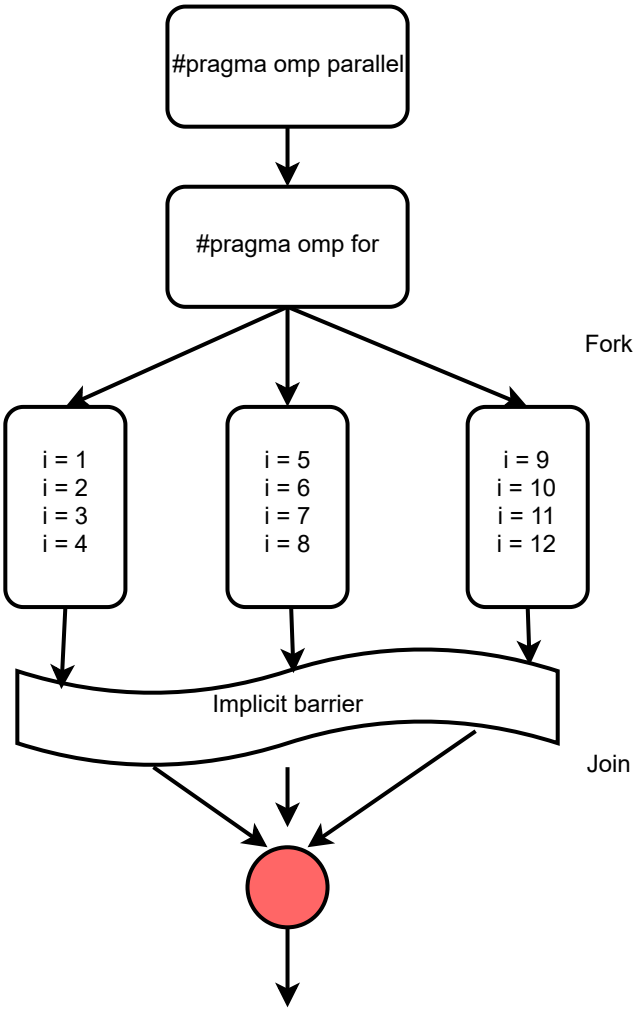
Open MP - reduction and parallel `for-loop`

Let's now try to understand how to parallelize a simple `for loop` using OpenMP. For example, we want to sum-up all the numbers from `1-12`, but using `3` threads in parallel. Let's observe the following figure



OpenMP parallel for-loop

```
#include <iostream>
#include <omp.h>
#include <cmath>

int main(){
    double sum_i = 0;
    #pragma omp parallel for reduction(+:sum_i)
    for (int i=0; i<12; i++){
        sum_i += i;
```

```
    }
    std::cout << "Sum = " << sum_i << std::endl;

    return 0;
}
```

How it works? First, threads are assigned an independent set of iterations and then threads must wait at the end of work-sharing construct. Note that you can do `reduction` by specifying more than one variable separated by a comma, i.e. a `list`:

```
#pragma omp parallel for default(shared) reduction(+:sum,result) ..
```

The `reduction` clause performs a reduction on the variables that appear in its list. A private copy for each list variable is created for each thread. At the end of the reduction, the reduction variable is applied to all private copies of the shared variable, and the final result is written to the global shared variable.