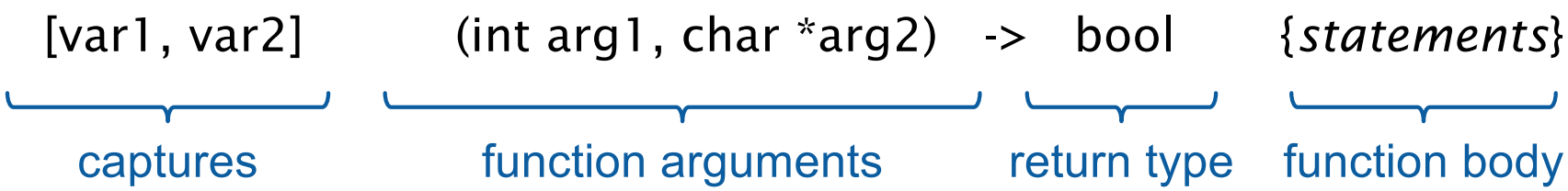
CS 111 Assignment 1:

Lambdas, Threads, and Processes

Exercise 1: C++ Lambdas

- **Lambdas: a convenient way to encapsulate little bits of code:**
  - Typically passed as an argument to a method
  - The method invokes the lambda later
  - Often used to customize the behavior of a function
- **What is a lambda?**
  - A function with no name
  - Can capture values from the environment where it was declared



nearest.cc: function argument

```
bool compare(int val1, int val2) {
    int dist1 = abs(val1 - 50);
    int dist2 = abs(val2 - 50);
    return dist1 < dist2;
}

int main(int argc, char **argv)
{
    int values[] = {47, 102, 13, 18, 95, 66, 63, 5, 173, 6, 6, 44, 52};
    size_t num_values = sizeof(values) / sizeof(values[0]);

    std::sort(&values[0], &values[num_values], compare);

    std::string prefix = "";
    std::cout << "Sorted values: ";
    for (size_t i = 0; i < num_values; i++) {
        std::cout << prefix << values[i];
        prefix = ", ";
    }
    std::cout << std::endl;
}
```

nearest2.cc: use lambda

```
int main(int argc, char **argv)
{
    int values[] = {47, 102, 13, 18, 95, 66, 63, 5, 173, 6, 6, 44, 52};
    size_t num_values = sizeof(values) / sizeof(values[0]);

    std::sort(&values[0], &values[num_values], [] (int val1, int val2) -> bool {
        int dist1 = abs(val1 - 50);
        int dist2 = abs(val2 - 50);
        return dist1 < dist2;
    });

    std::string prefix = "";
    std::cout << "Sorted values: ";
    for (size_t i = 0; i < num_values; i++) {
        std::cout << prefix << values[i];
        prefix = ", ";
    }
    std::cout << std::endl;
}
```

nearest3.cc: captures

```
int main(int argc, char **argv)
{
    if (argc != 2) {
        std::cerr << "Usage: nearest3 target" << std::endl;
        return 1;
    }
    int target = std::stoi(argv[1]);

    int values[] = {47, 102, 13, 18, 95, 66, 63, 5, 173, 6, 6, 44, 52};
    size_t num_values = sizeof(values) / sizeof(values[0]);

    std::sort(&values[0], &values[num_values], [target] (int val1, int val2) -> bool {
        int dist1 = abs(val1 - target);
        int dist2 = abs(val2 - target);
        return dist1 < dist2;
    });

    ... print array just like nearest2.cc ...
}
```

Parse command-line argument

Capture: makes copy of variable for use inside lambda

Exercise 3: print

```
#define NUM_THREADS 5

void thread_main(int id)
{
    int i;
    for (i = 0; i < 50; i++) {
        printf("Thread %d used printf for line %d\n", id, i);
        //std::cout << "Thread " << id << " used <iostream> for line " << i << std::endl;
    }
}

int main(int argc, char **argv)
{
    std::thread *threads[NUM_THREADS];
    for (int i = 0; i < NUM_THREADS; i++) {
        threads[i] = new std::thread([i](){thread_main(i);});
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        threads[i]->join();
        delete threads[i];
    }
}
```

Exercise 6: fork_print

```
static int x = 44;

int main(int argc, char **argv)
{
    pid_t child = fork();
    if (child == 0) {
        // A zero return value means we are now running in the child.
        std::cout << "Child is running, variable x is " << x << std::endl;
        x += 1;
        std::cout << "Child incremented x, new value is " << x << std::endl;
        exit(0);
    }

    // This code runs in the parent process; wait for the child to exit.
    waitpid(child, NULL, 0);
    std::cout << "Child (pid " << child << ") exited; variable x is "
            << x << std::endl;
    return 0;
}
```

Exercise 6: exec

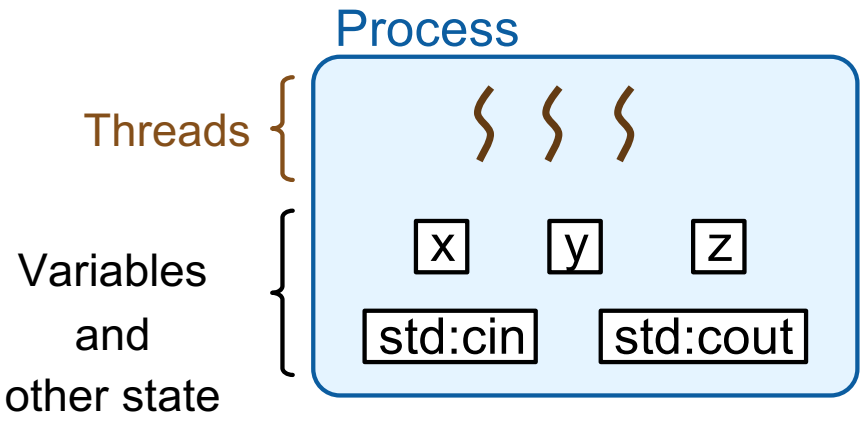
```
int main(int argc, char **argv)
{
    while (1) {
        std::string line;

        std::cout << "% " << std::flush;
        if (!getline(std::cin, line))
            break;
        Splitter split_line(line);
        char **words = split_line.get_words();

        pid_t child = fork();
        if (child == 0) {
            // This code runs in the child process.
            execvp(words[0], words);
            std::cout << "Couldn't exec " << words[0] << ": "
                    << strerror(errno) << std::endl;
            exit(1);
        }

        // This code runs in the parent process; wait for the child to exit.
        waitpid(child, NULL, 0);
    }
    exit(0);
}
```

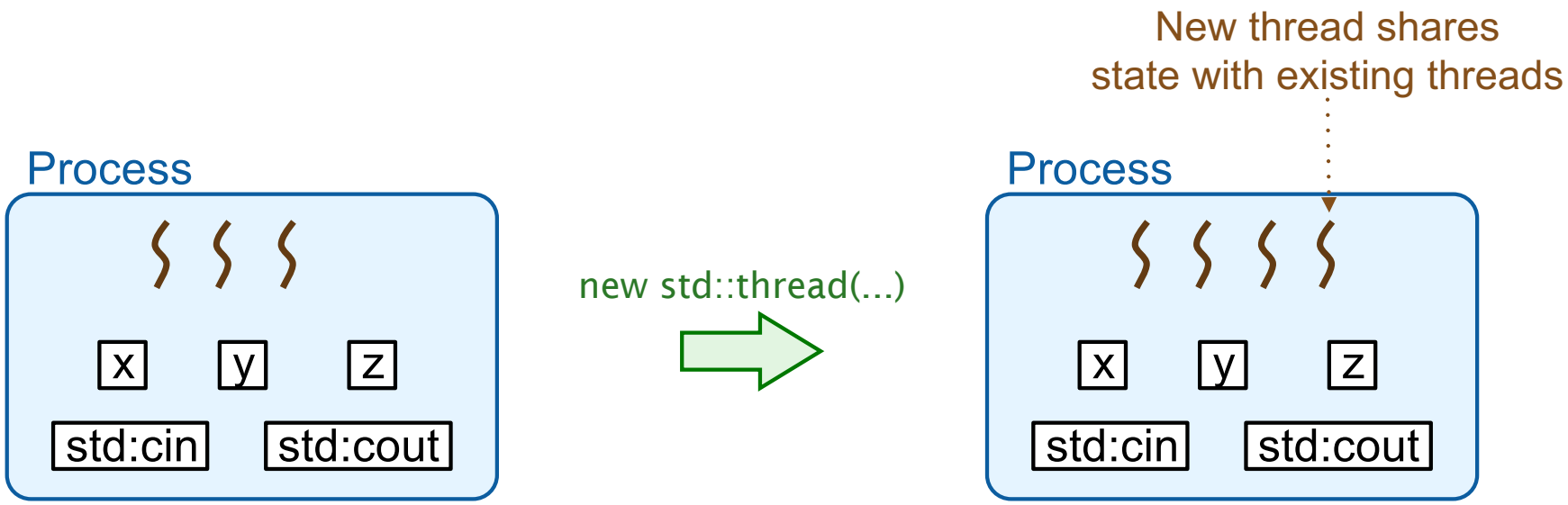
Processes vs. Threads



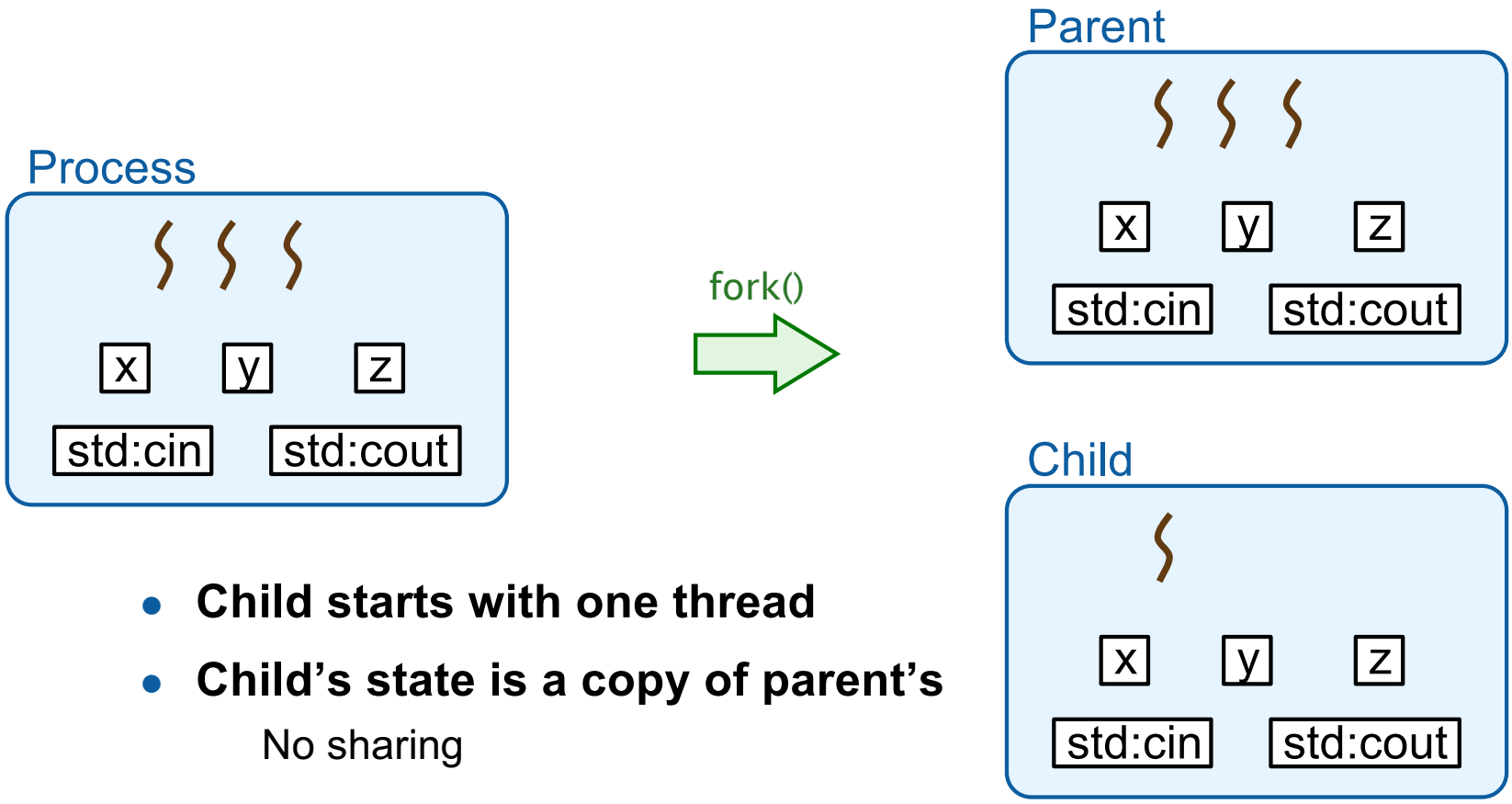
A thread is part of a process:

- A process can contain many threads
- A process also contains state

Create New Thread



Create New Process



- **Child starts with one thread**
- **Child's state is a copy of parent's**

  No sharing