



```
void push_42 (cow_vector<int>::inout v)
{
    std::thread t ([v]
                    {
                        // create a copy
                        auto vec2 = v;
                    });

    v.push_back (42); // no data-race as v has
                     // an internal copy
}
```



Modelled in out



```

class inout
{
public:
    ~inout()
    {
        *cow_vector_arg = cow_vector; // copy back modifications
    }

    // reflect to generate and forward all
    // functions to internal copy

private:
    friend class cow_vector;
    cow_vector* cow_vector_arg; // pointer to original
    cow_vector cow_vector;      // copy, safe to modify

    inout (cow_vector* v)
        : cow_vector_arg (v),
          cow_vector (*v)
    {}
};

inout make_inout()
{
    return inout (this);
}

```













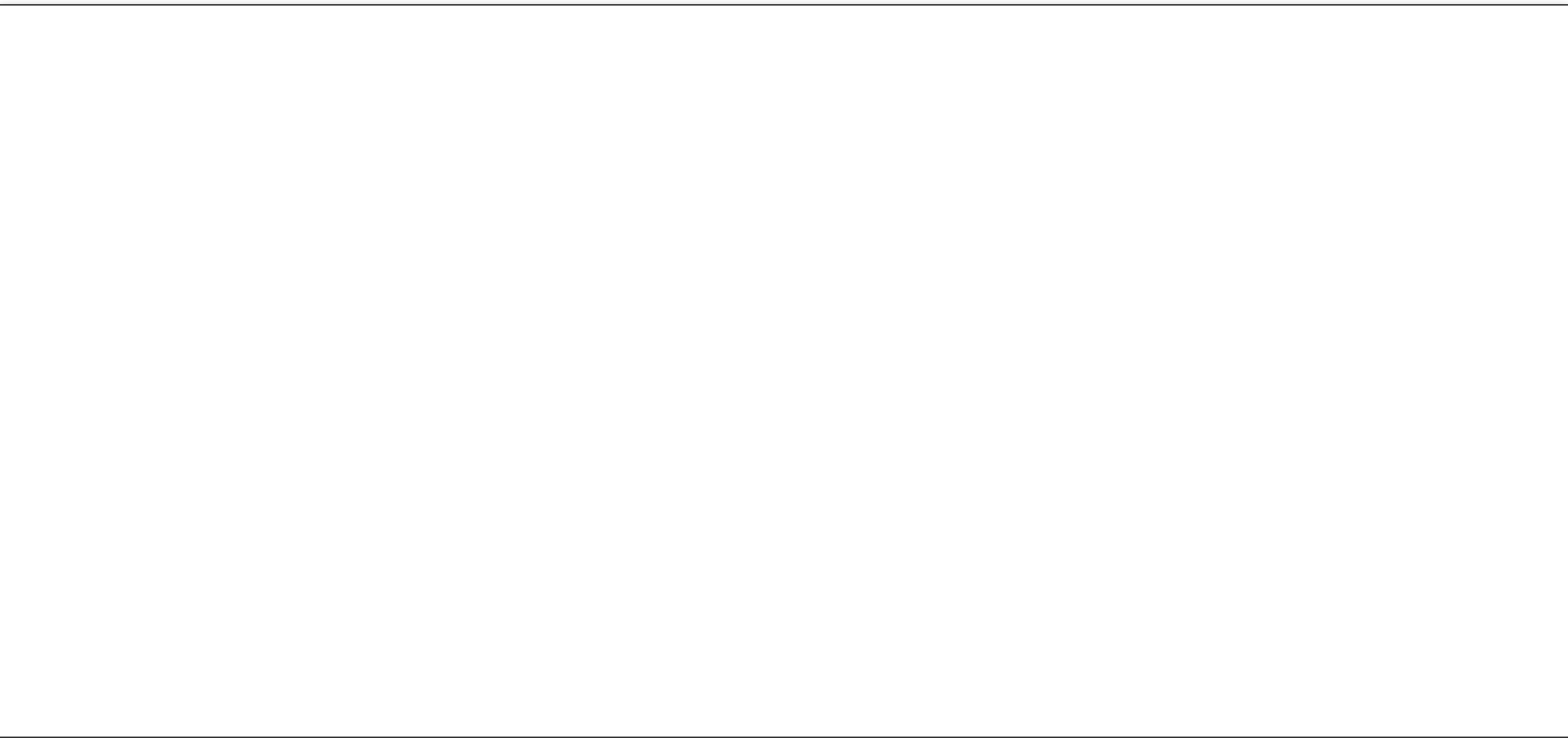


```
cow_vector<int> vec;  
vec.push_back (40);  
vec.push_back (41);
```

```
push_42 (vec.make_inout());
```

```
//... vec contains 42
```

*vec = vec\_fun;*



```
void push_42(cw_vec_t <int>::inout v)
```





/// create copy



v.push\_back(42); // no data race as v has





autovec2 = v;

std::thread\_t(v)

~~/~~~~/~~~~/~~an internal copy



Modelled in not







class in our

$\sim$  input (

public







convert\*convert\_arg; // print original

friendclass\_cowvector;

\*cow\_vectr\_arg == cow\_vectr;~~///~~ ~~copy~~ ~~back~~ ~~modification~~

return invert(this);

inout make\_inout()

///functions to internal copy

//reflect to generate and forward all

input(cow\_vector)\*v)





private

:- convvectorrange(v),

convectors ( $*v$ )

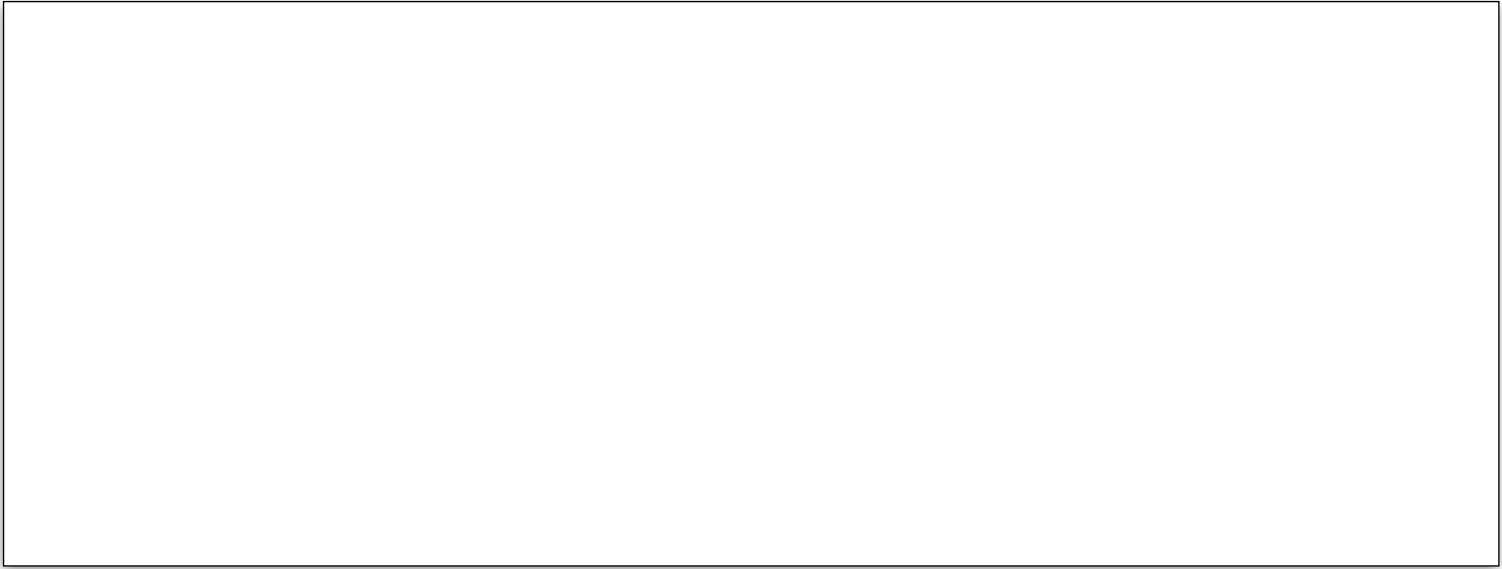




cow\_vec to cow\_vec; ~~copy, safe to modify~~







```
vec.push_back(40);
```

convertor <int> vec;

```
push_42(vec.make_inout());
```

```
vec.push_back(41);
```



$vec = vec_{func};$

input(cow\_vector\* p)



:- cow\_vector(rang(p),

cow\_vector(\*p)

return invert(this);

$$\text{auto vec2} = *v;$$

push\_42(&vec);



input operators()

returning in out (this);







