Exercise 2
Diffusion and Multithreading

High Performance Computing for Science and Engineering I

October 3, 2014

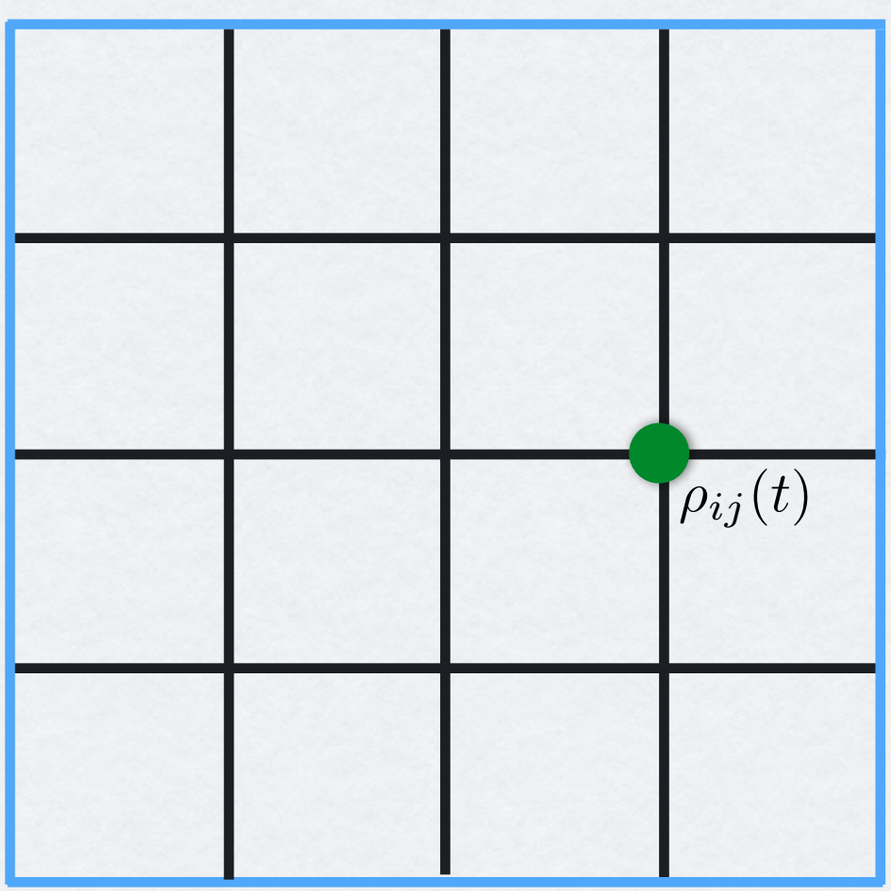
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Diffusion equation

- Solve the partial differential equation

$$\frac{\partial \rho(\mathbf{r},t)}{\partial t} = D\nabla^2\rho(\mathbf{r},t)$$

- on a NxN grid with Dirichlet boundary cond.



Finite difference in 1d:

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{f(x-\Delta x) + f(x+\Delta x) - 2f(x)}{(\Delta x)^2}$$

Parameter sets

	D	N	Δ
Set 1	1	128	0.00001
Set 2	1	256	0.000001
Set 3	1	1024	0.0000001

Parallel code with C++11 threads

- Use multiple threads to reduce execution time
  - Distribute one space dimension among threads
- Be careful about synchronization
  - Do not access data that another thread is still modifying
- Verify that your implementation is correct
  - Against the output of the serial program

What is a barrier?

```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);


        f2(vec[(t+1)%2]);


});
```

What is a barrier?

```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

        vec[t%2] = f1(t);

        f2(vec[(t+1)%2]);

});
```

thread 1

What is a barrier?

```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

        vec[t%2] = f1(t);

        f2(vec[(t+1)%2]);

});
```

thread 2

thread 1

What is a barrier?

```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);


    f2(vec[(t+1)%2]);


});
```

thread 2

thread 1

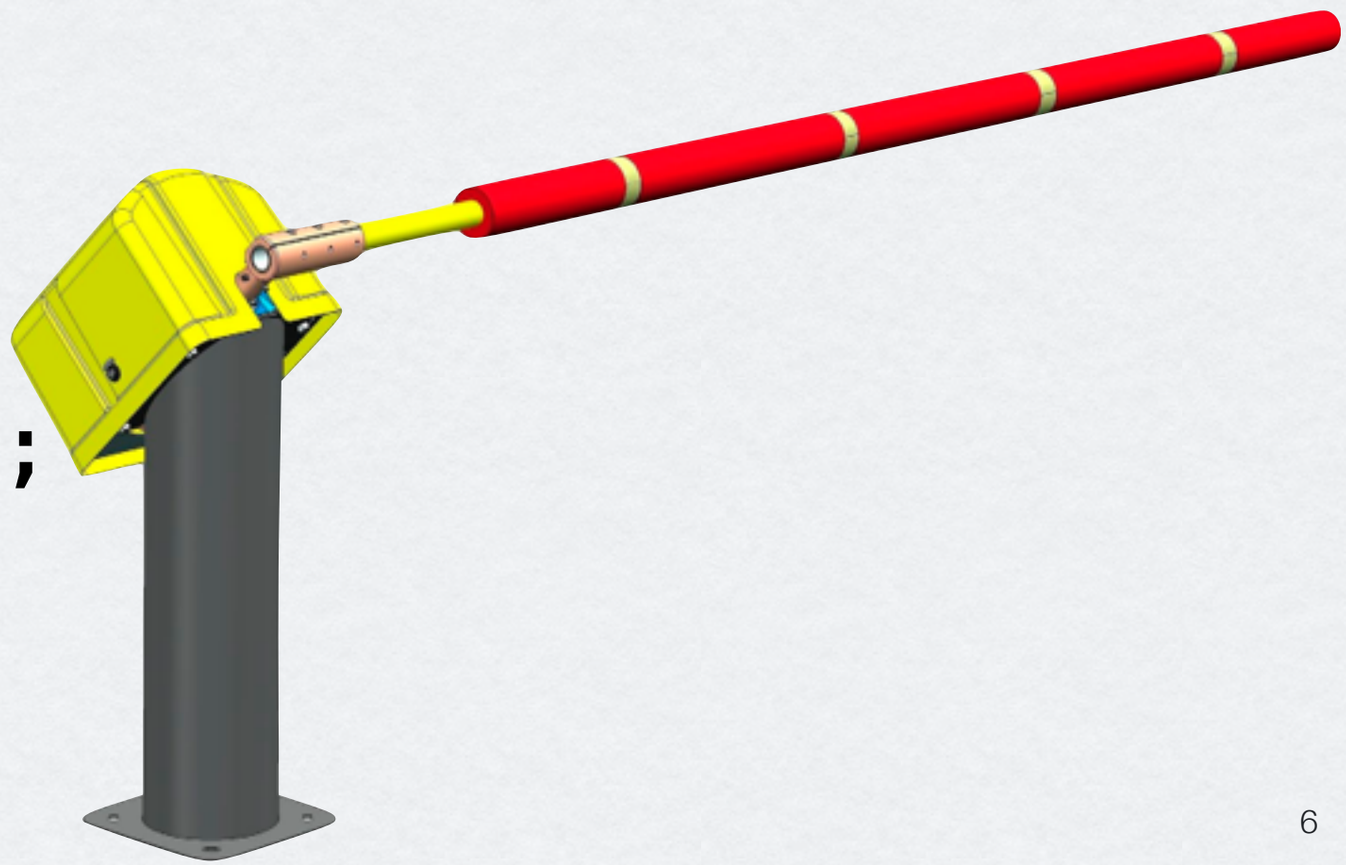
What is a barrier?

```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

        vec[t%2] = f1(t);

        f2(vec[(t+1)%2]);

});
```

thread 2

What is a barrier?

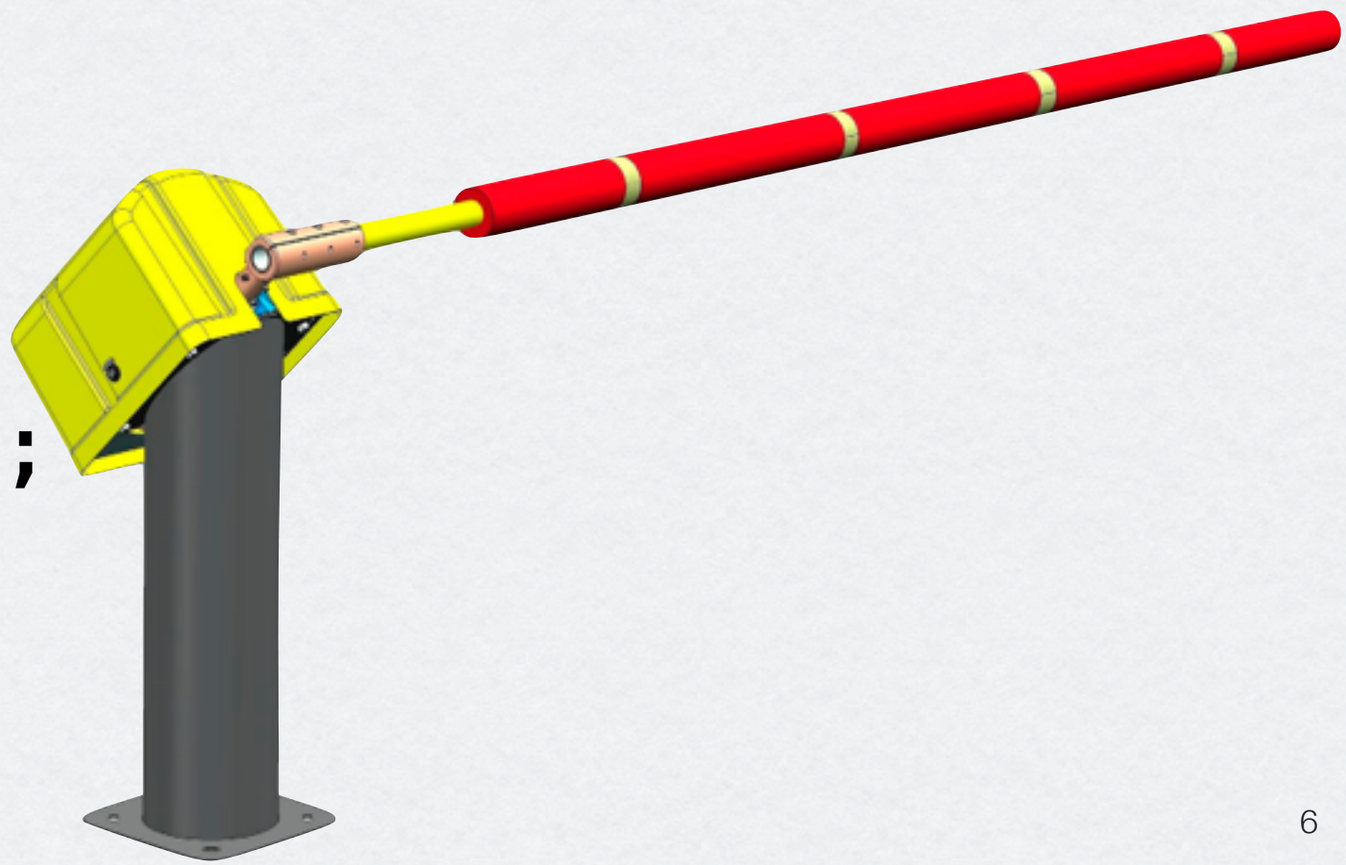
```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```



What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```

thread 1

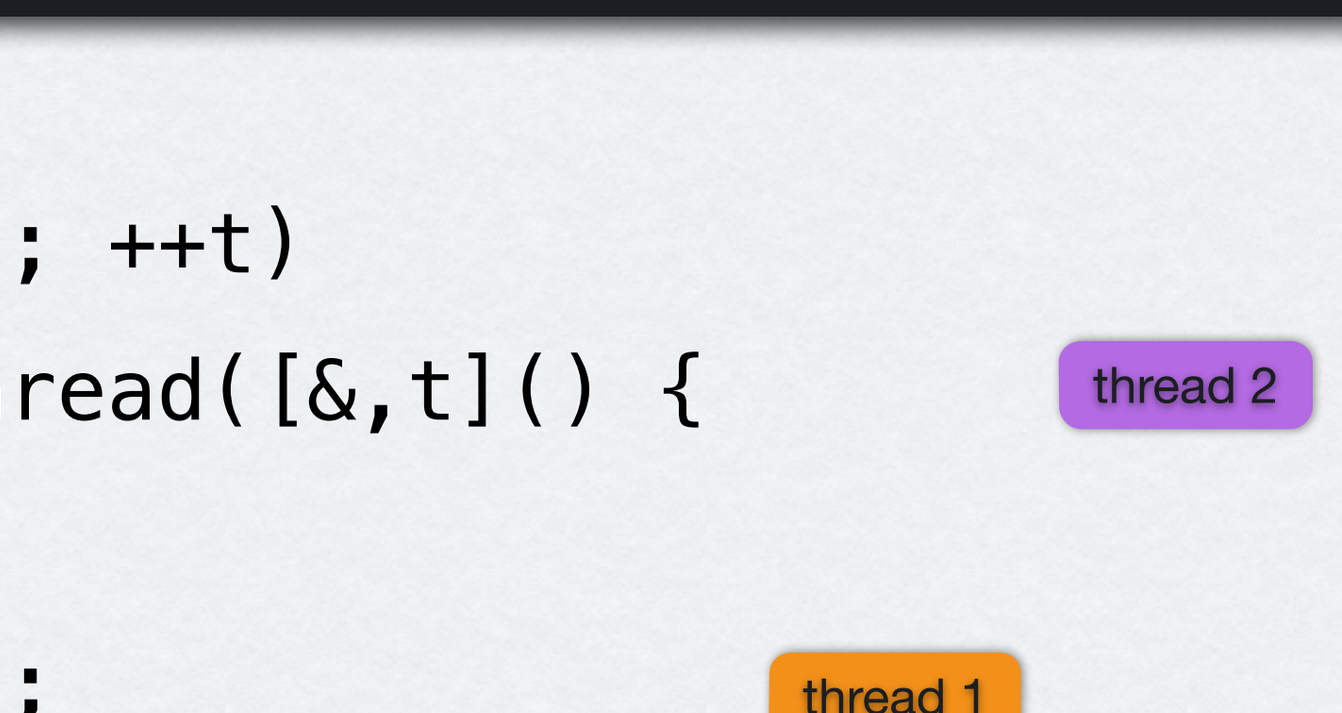


What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```

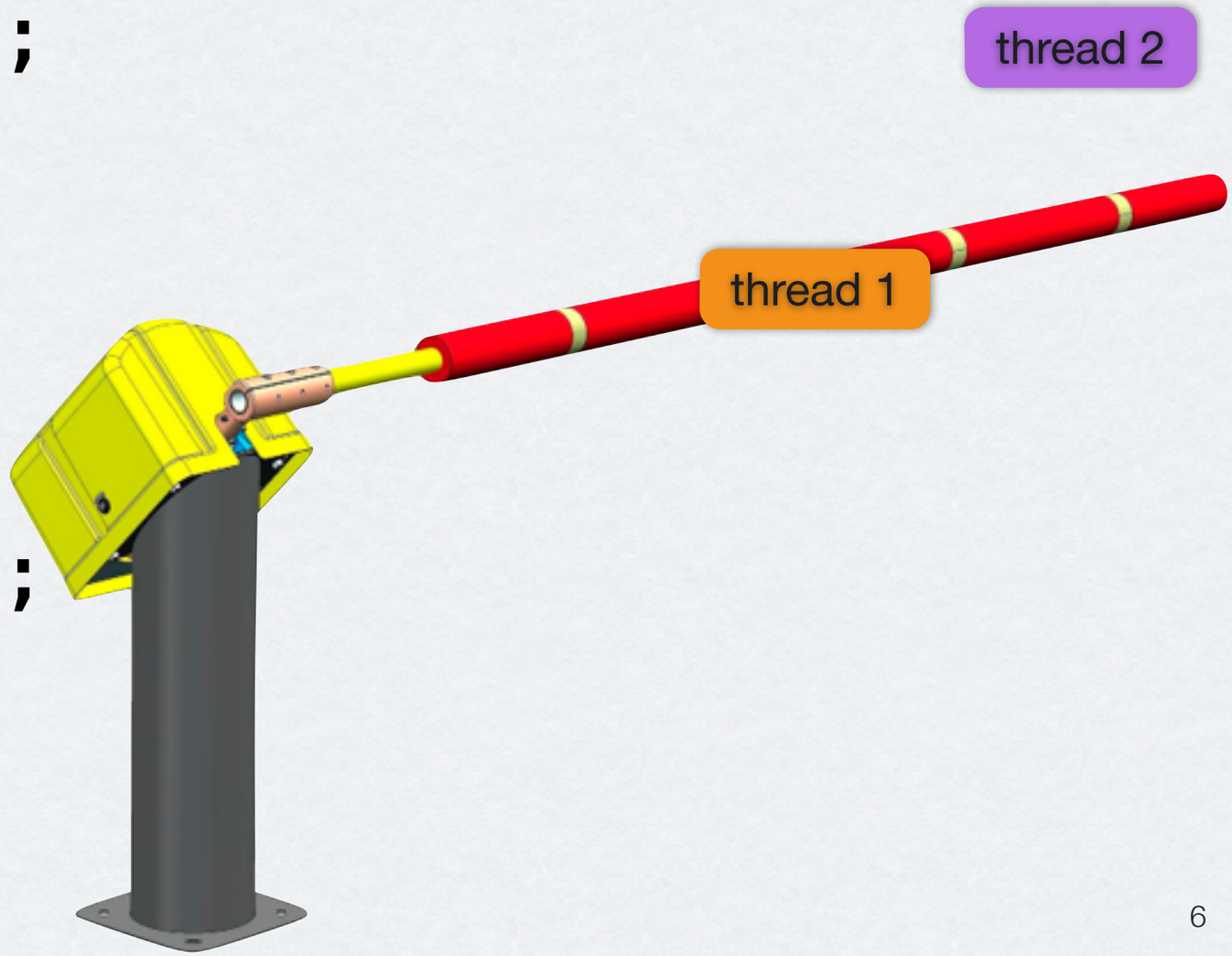
thread 2

thread 1



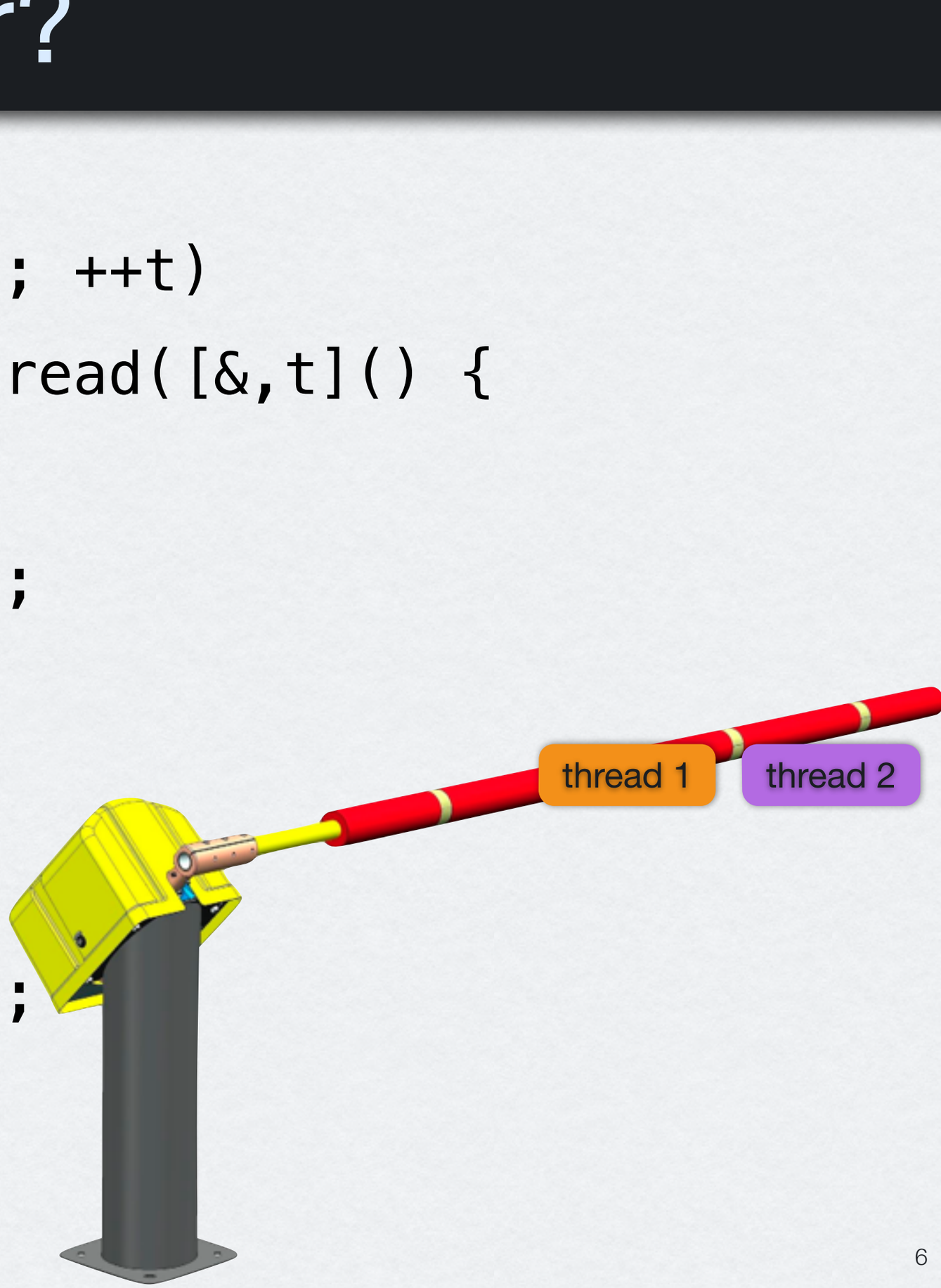
What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```



What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```



What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```

thread 1 thread 2

What is a barrier?

```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {


        vec[t%2] = f1(t);

        b.wait();

        f2(vec[(t+1)%2]);

});
```

thread 1 thread 2