Exercise 2
Diffusion and Multithreading

High Performance Computing for Science and Engineering I

October 7, 2016

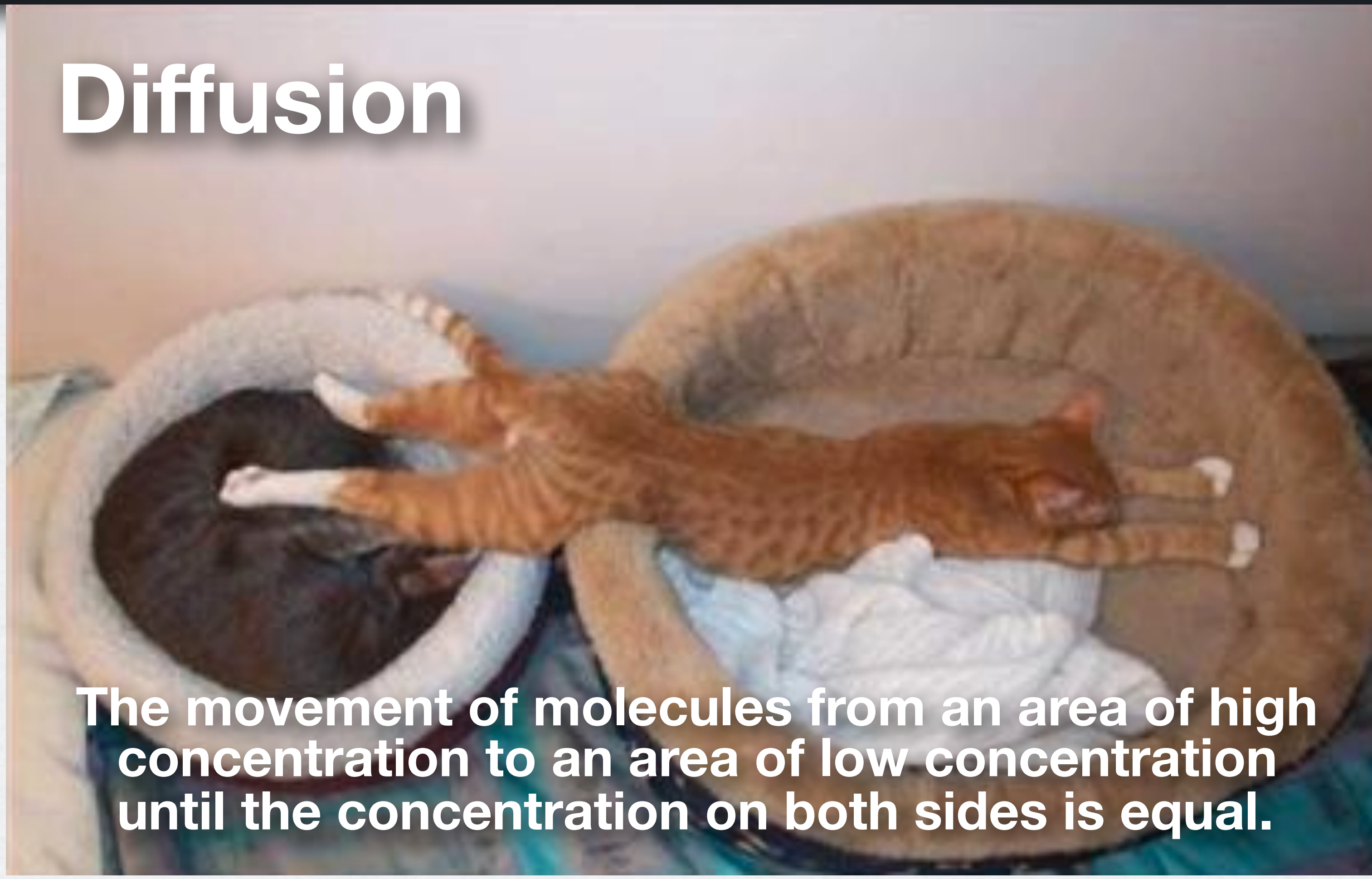
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

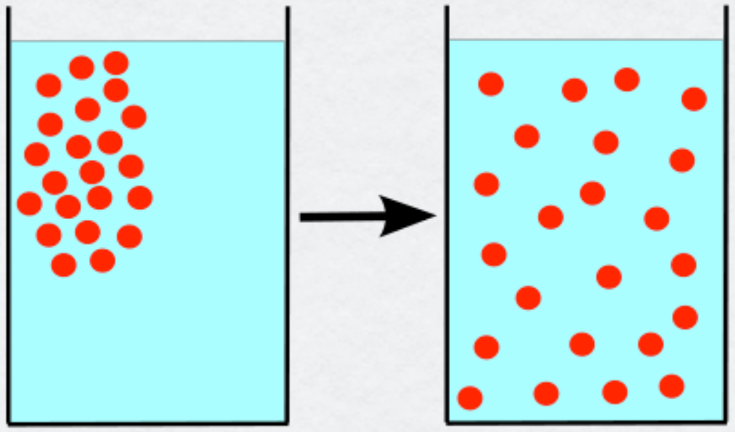
Diffusion

- describes spread of the quantity driven by its concentration gradient towards regions with lower density

Examples:

- distribution of heat in given region
- drop of ink in the glass of water
- teabag diffusion





Diffusion Equation

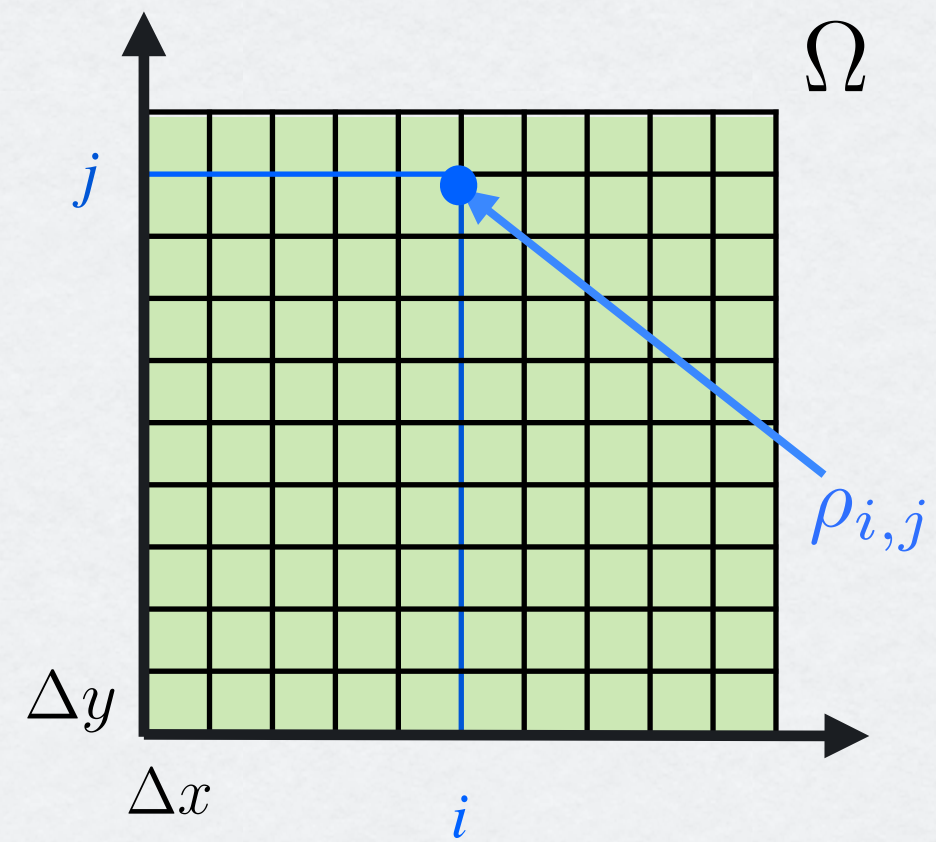
- Diffusion of quantity **ρ** (e.g. heat flow) can be described by diffusion equation of the form:

$$\frac{\partial \rho(\boldsymbol{r},t)}{\partial t} = D\nabla^2 \rho(\boldsymbol{r},t) \quad \text{in} \quad \Omega \qquad (1)$$

ρ(r,t) - measure for the amount of heat at position $r=(x,y)$ and time t

D - diffusion coefficient (constant here)

- Discretizing eq. (1) using forward Euler in time and central differences in space yields:



$$\frac{\rho_{i,j}^{(n+1)} - \rho_{i,j}^{(n)}}{\Delta t} = D\left(\frac{\rho_{i+1,j}^{(n)} - 2\rho_{i,j}^{(n)} + \rho_{i-1,j}^{(n)}}{\Delta x^2} + \frac{\rho_{i,j+1}^{(n)} - 2\rho_{i,j}^{(n)} + \rho_{i,j-1}^{(n)}}{\Delta y^2}\right)$$

- where *n* is the index of time step: $t_n = n \cdot \Delta t$
- i, j are indices of spatial discretization: $x_i = i \cdot \Delta x$, $y_j = j \cdot \Delta y$
- and $\rho_{i,j}^n = \rho(x_i, y_j, t_n)$

Parameter sets

	D	N	Δt
Set 1	1	128	0.00001
Set 2	1	256	0.000001
Set 3	1	1024	0.00000001

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?

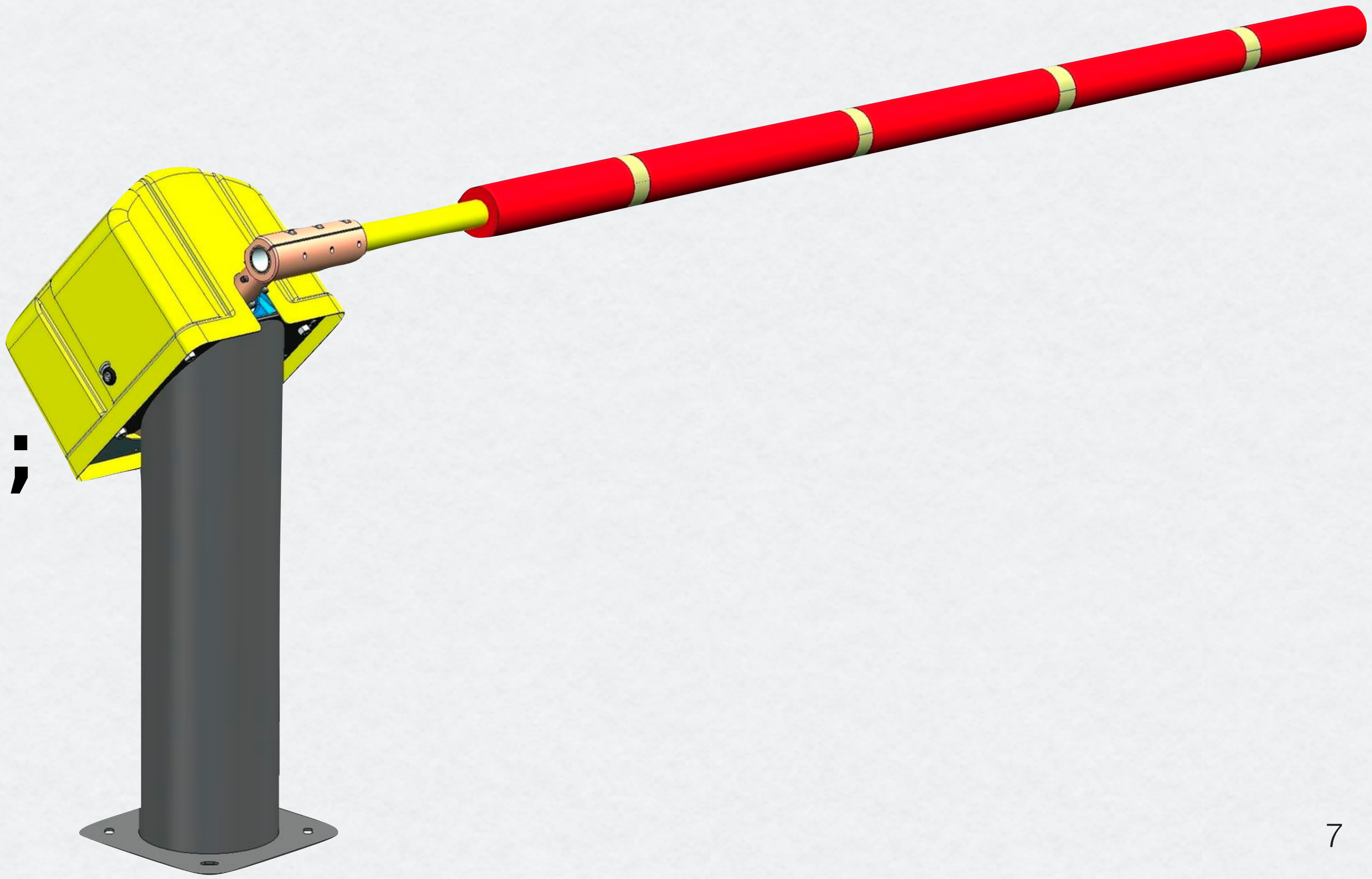
```
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

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




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

    });
```

thread 2

thread 1
access wrong data

What is a barrier?

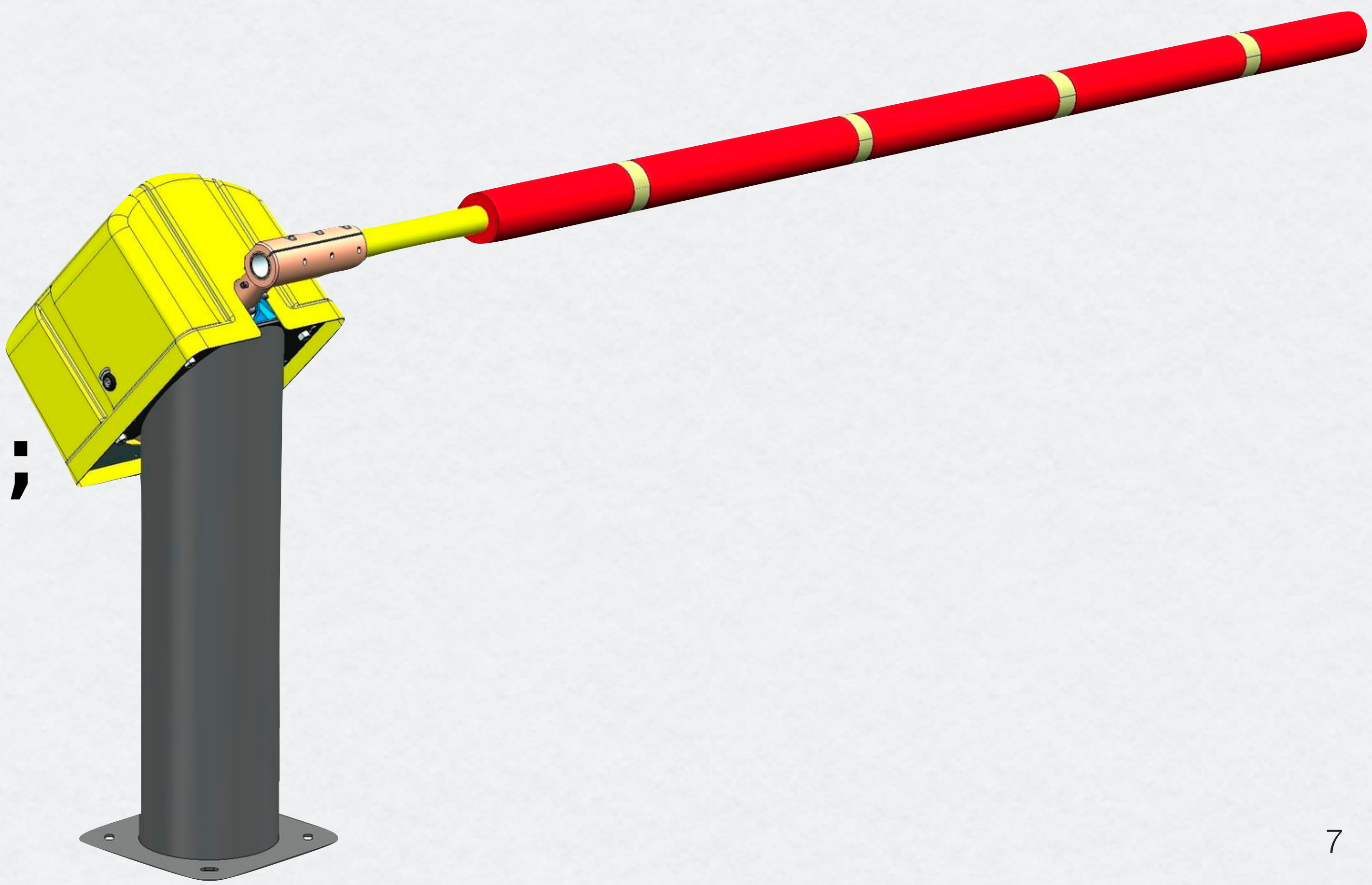
```
barrier b(nthreads);
for (int t=0; t<nthreads; ++t)
    threads[t] = std::thread([&,t]() {

        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

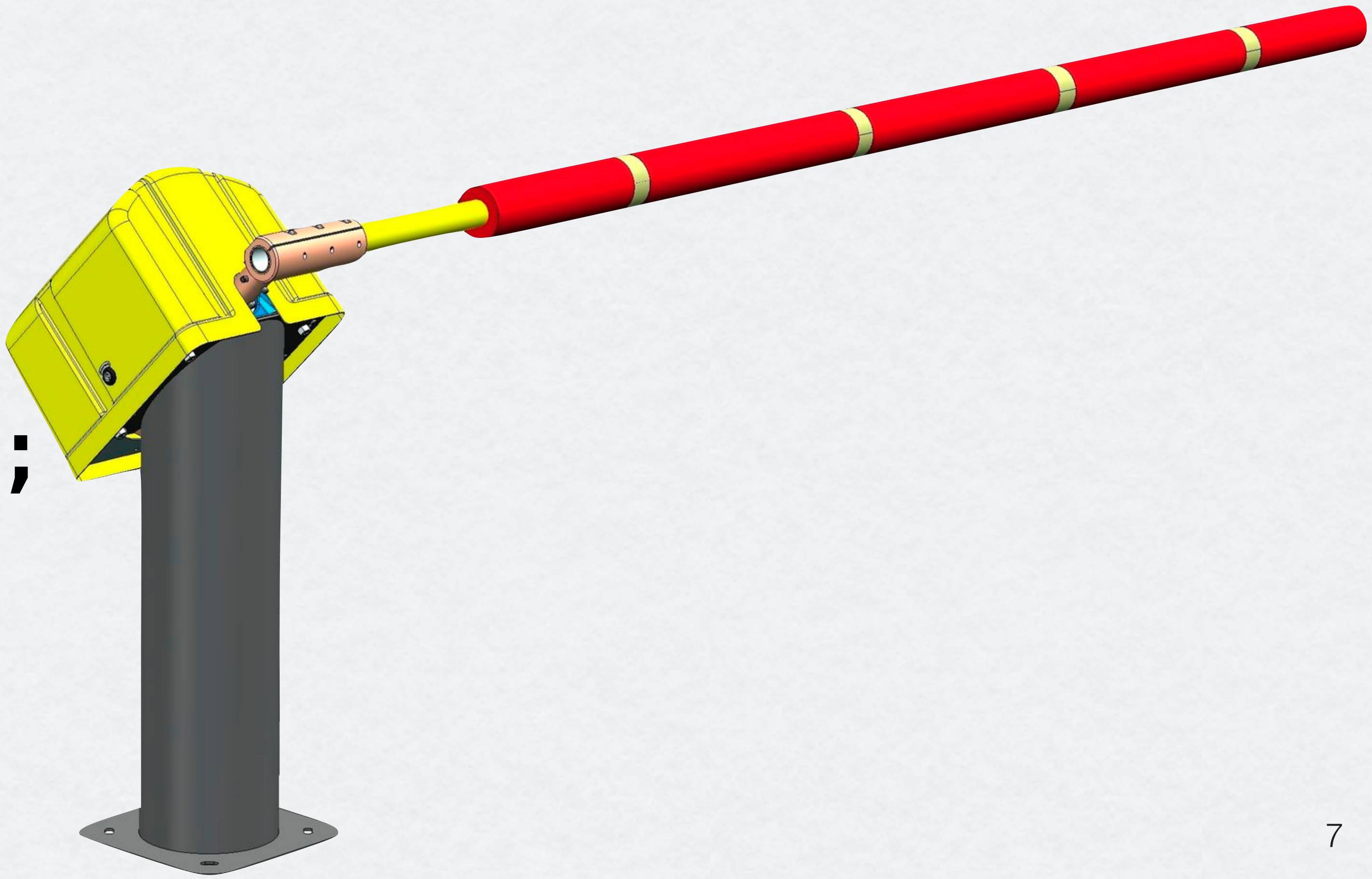


What is a barrier?

```
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]);

    });
```

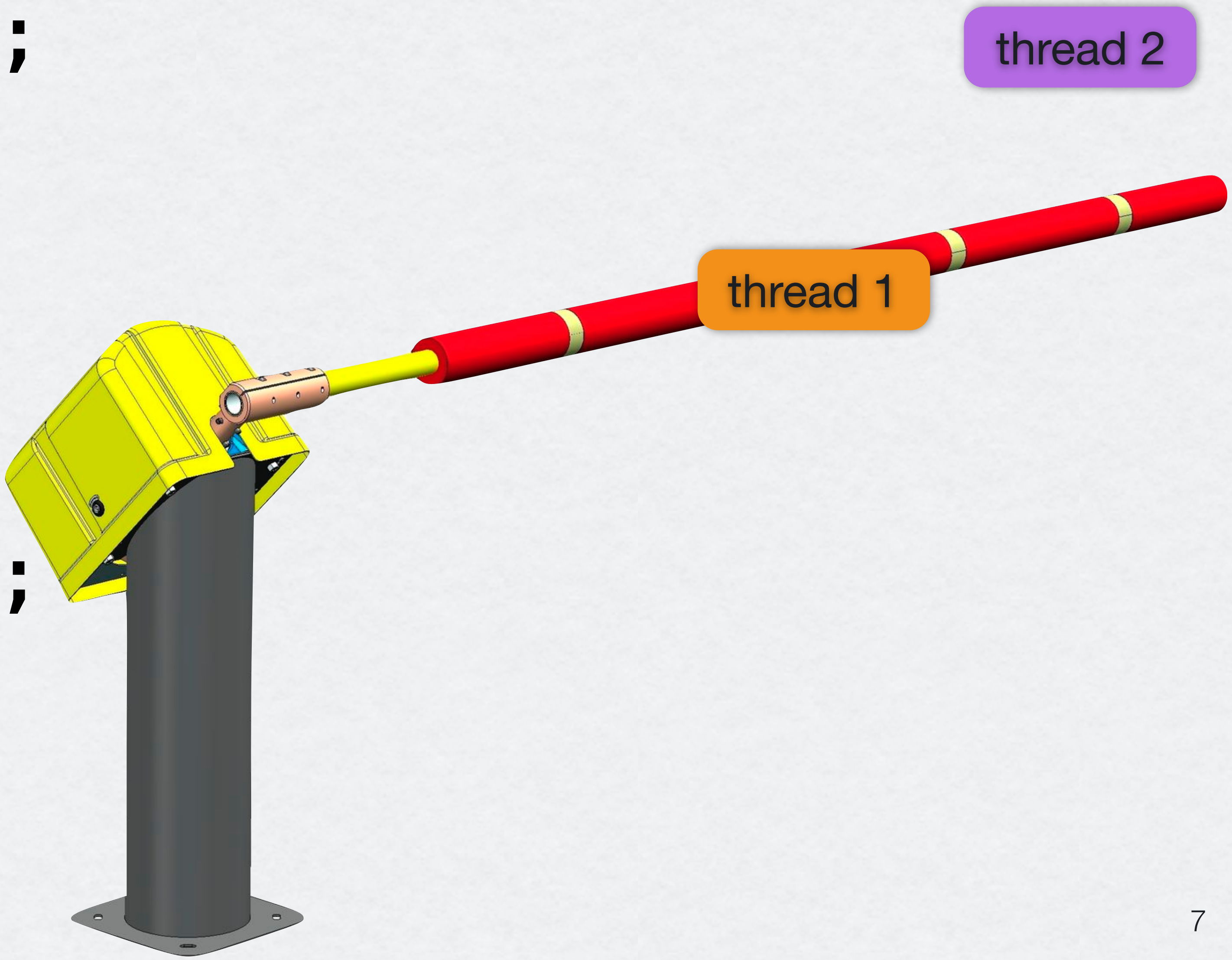
thread 2

thread 1



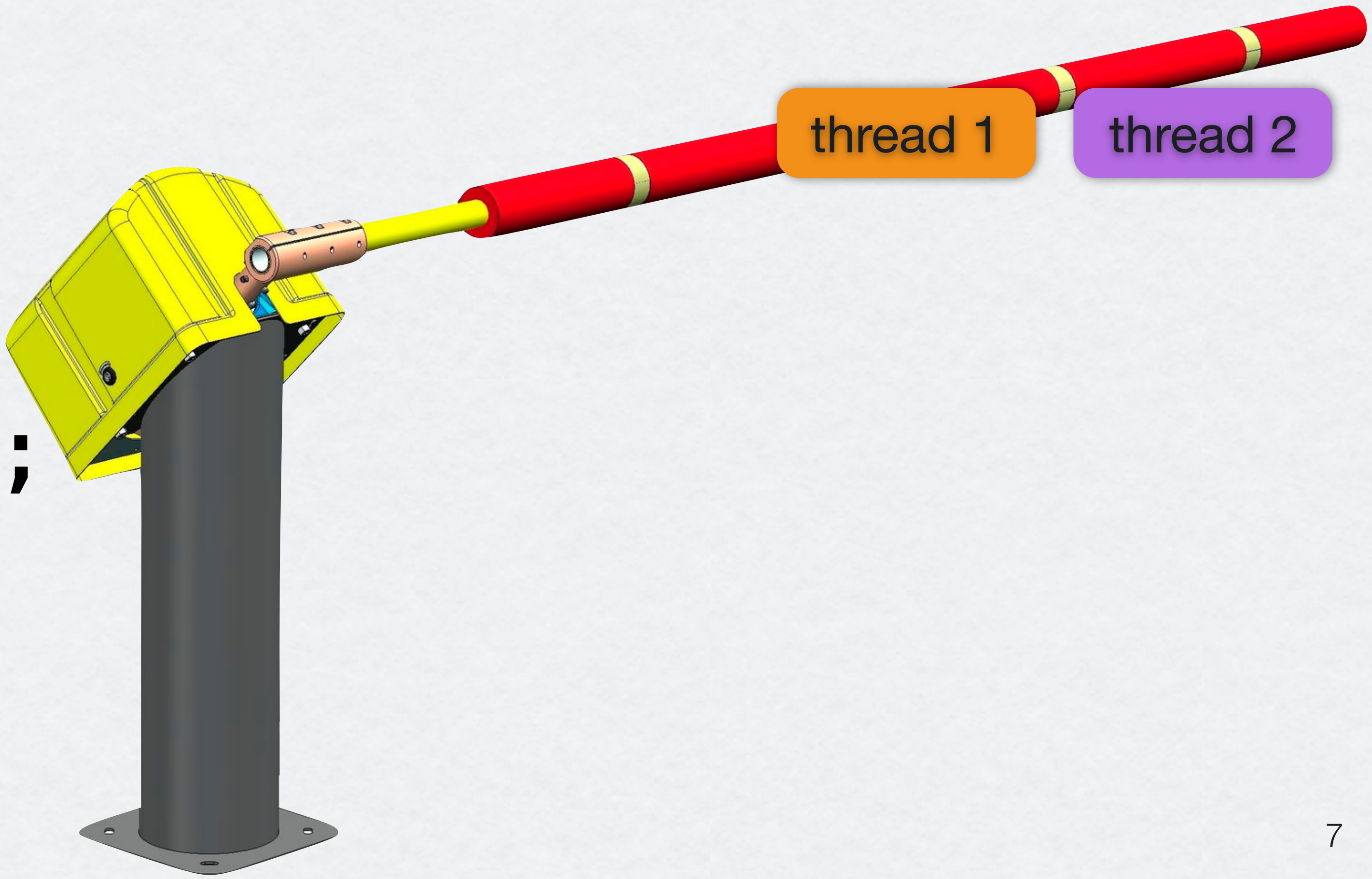
What is a barrier?

```
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