Lecture #16 - Adaptive Quadrature and Load Balancing

AMath 483/583

Announcements

- **Homework #4** — Due Next Friday @ 5:00pm (Last Day of Class)
  - short, but conceptually challenging
  - (i.e. not something you can do at the last minute)
  - clarification in this lecture
- Today:
  - MPI on multiple machines
  - Return to OpenMP — “nested parallelism”

Quick Aside

- MPI is for distributed memory environments

```
$ mpiexec -n 4 -H [server list] ./myprog
```

- Can also create a hostfile:

```
# my_hostfile
foo.example.com
bar.example.com slots=2
foobar.example.com slots=4 max-slots=4
```

Quick Aside

- MPI is for distributed memory environments

```
$ mpiexec -n 4 -H [server list] ./myprog
```

- Can also create a hostfile:

```
# my_hostfile
foo.example.com
bar.example.com slots=2
foobar.example.com slots=4 max-slots=4
```

Two-core machine

Quad-core machine - ensure no more than 4 processes run here

Disclaimer

- Hardest part is setting up computers / ssh-keys
- In following demo
  - americano.amath.washington.edu can send requests to mocha.amath.washington.edu
  - not vice versa (?)

Demo

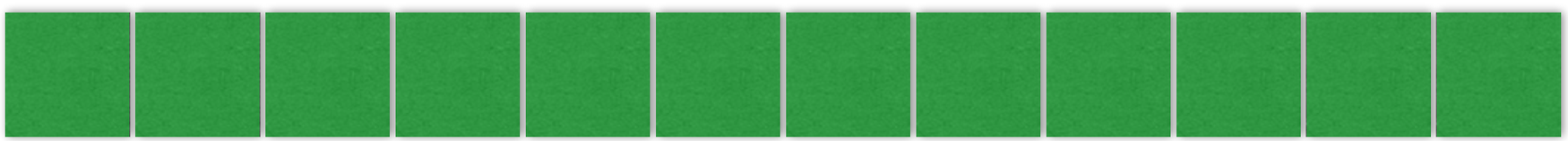
hello-distributed.c

On Homework #4

- **Clarify some misconceptions**: use the *strategy* from `shift.c`, not the same algorithm

```
double* u;
```

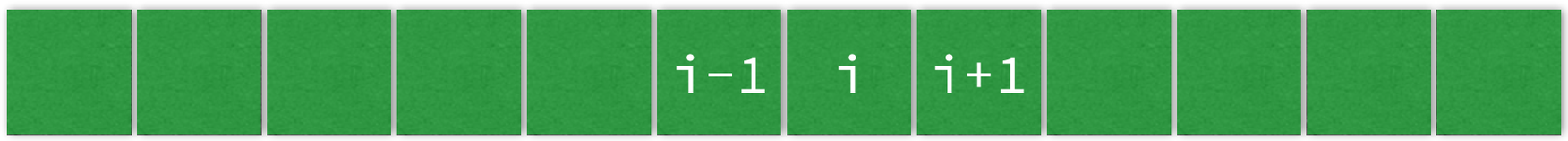
```
double* up1;
```

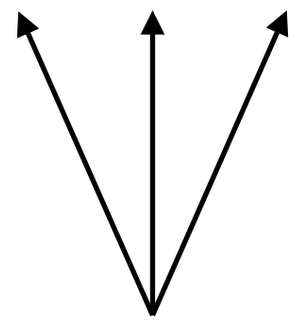


On Homework #4

```
up1[i] = up[i] +
         nu*(up[i-1] - 2*up[i] + up[i+1])
```

```
double* u;
```



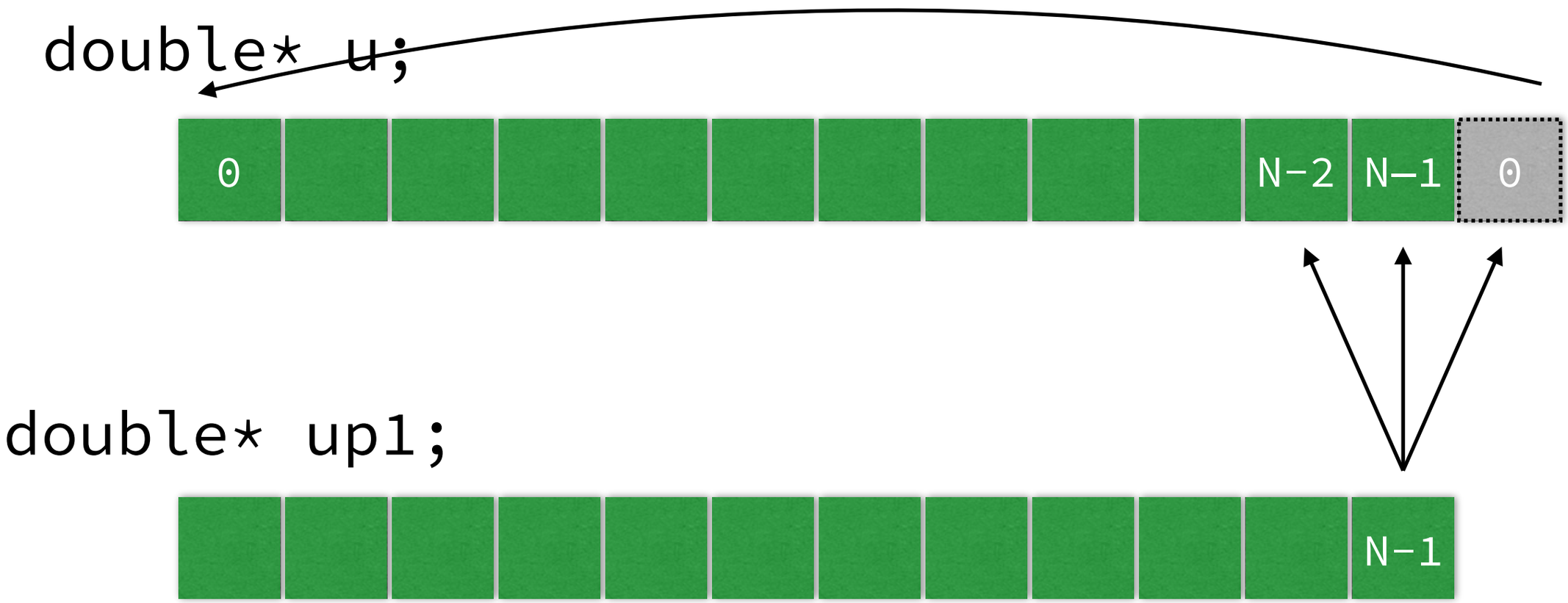


```
double* up1;
```

On Homework #4

Solving the periodic domain problem.

```
up1[N-1] = up[N-1] +
           nu*(up[N-2] - 2*up[N-1] + up[0])
```



On Homework #4

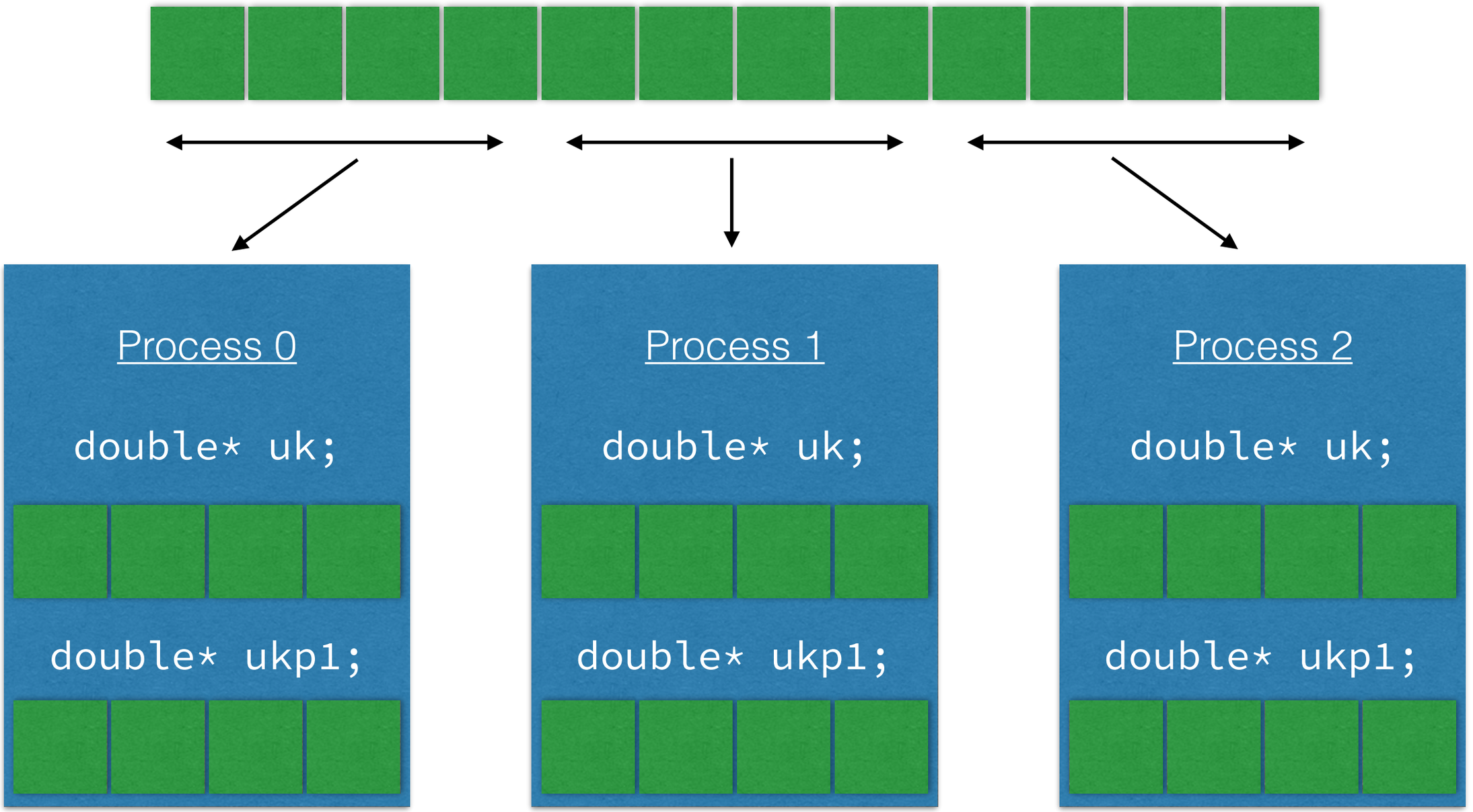
```
up1[N-1] =
                                    ] + up[0])
double* u
```

Idea

Divide the domain up into chunks `double* uk` and have each process perform “heat iteration” on each chunk.

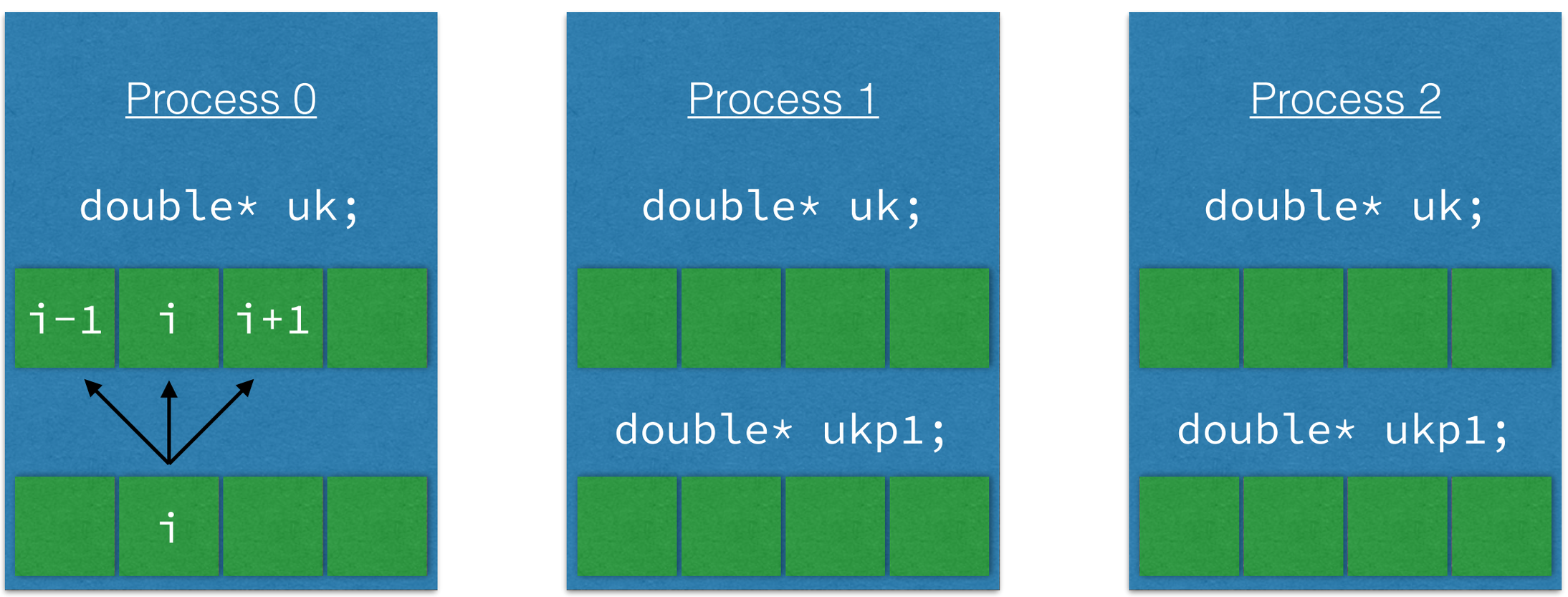
```
double* up1;
```

On Homework #4



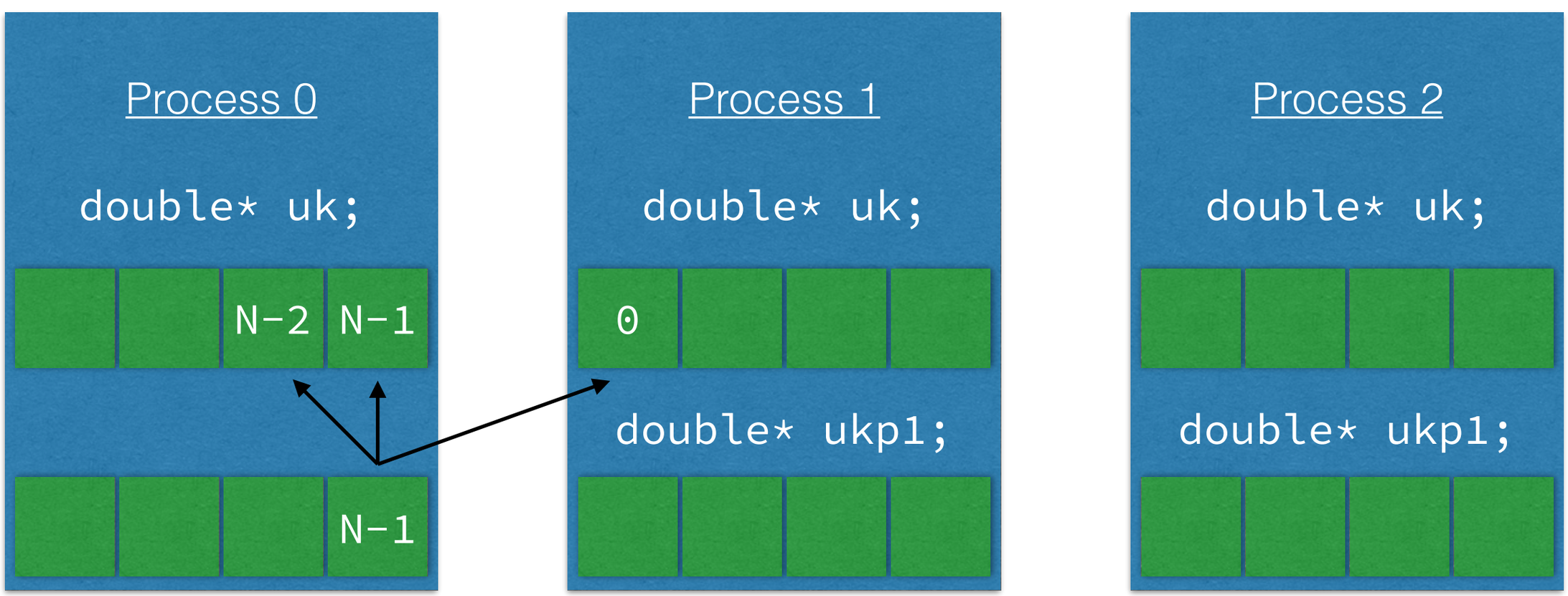
On Homework #4

```
ukp1[i] = ukp[i] +
          nu*(ukp[i-1] - 2*ukp[i] + ukp[i+1])
```



On Homework #4

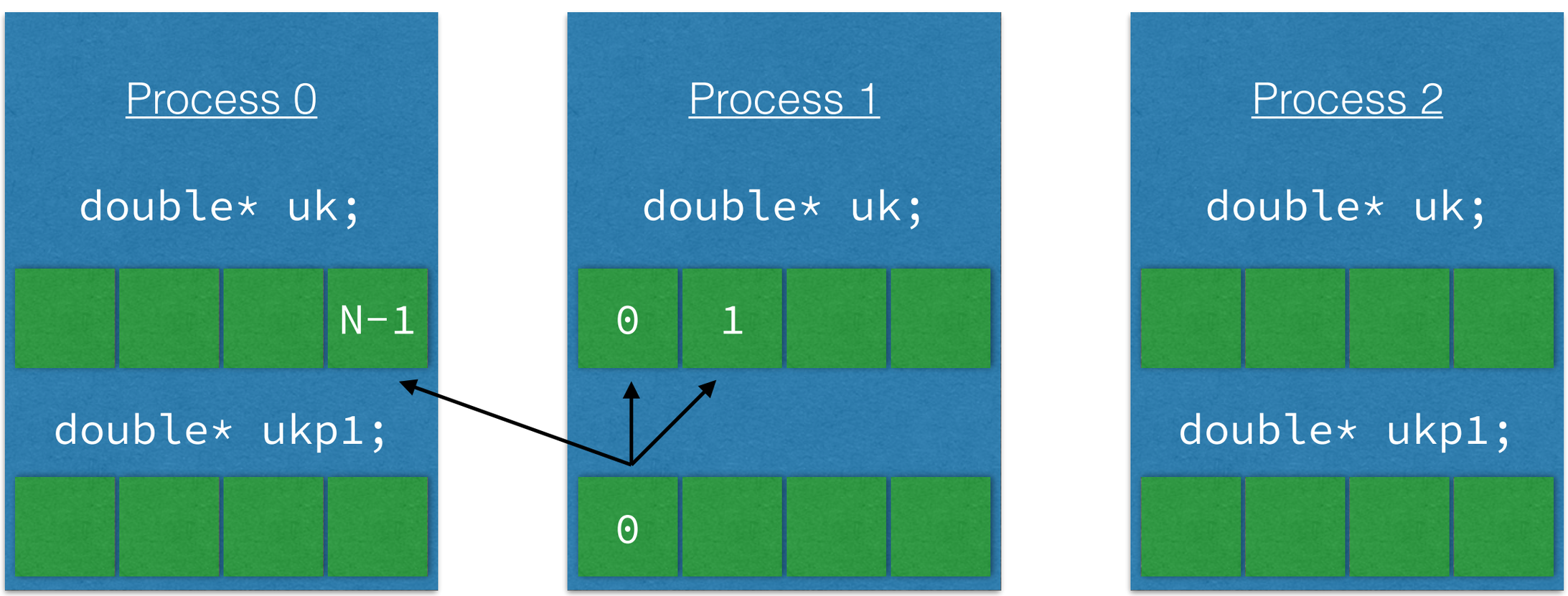
- **Uh oh**. Need to get **uk** information from neighboring processes. (Send / Recv)



On Homework #4

- **Same Issue**. Left side of process need info from left proc. Right side of process needs info from right proc.

  (***Hint***: *multiple ghost cells. More than one. More than two?)*



Adaptive Quadrature

Recursion and Nested Threading

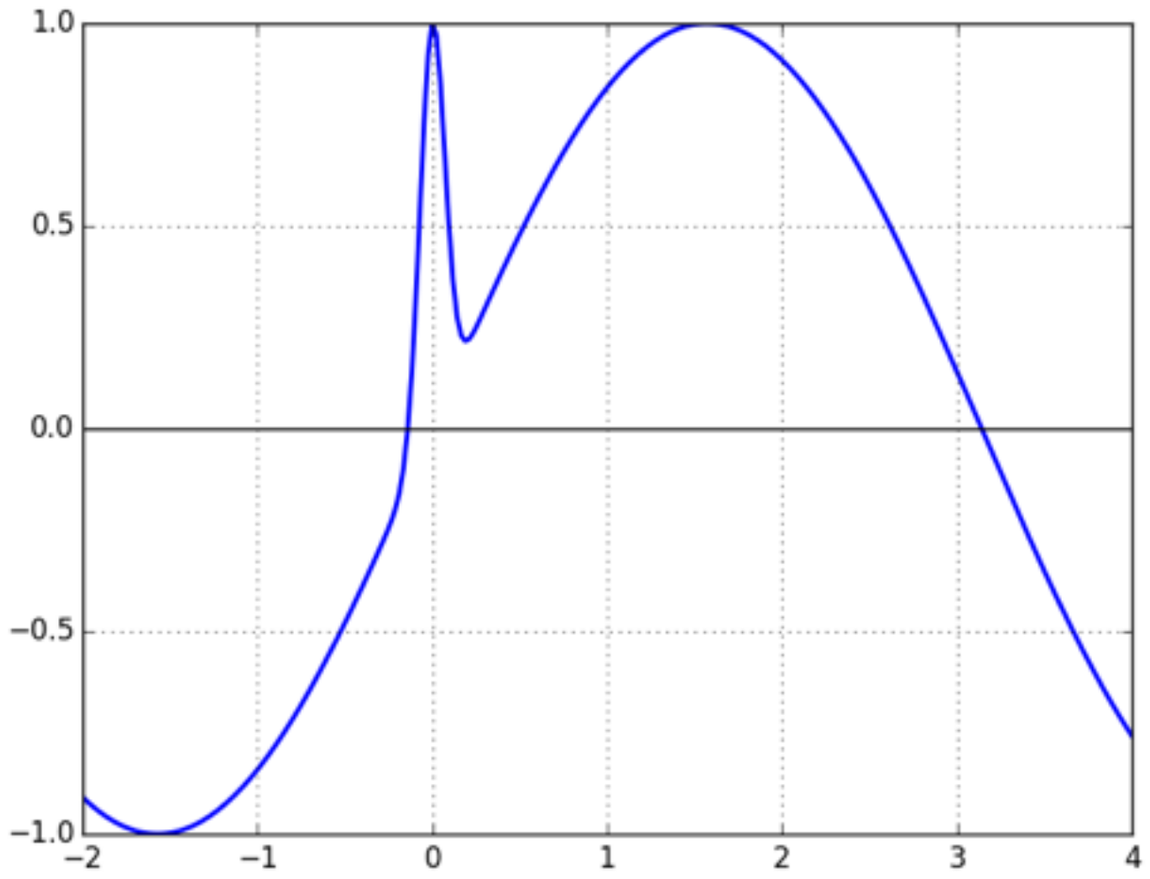
Adaptive Quadrature

- **Problem**: approximate

$$\int_{-2}^{4} e^{-\beta^2 x^2} + \sin(x) dx = \left[\frac{\pi}{2\beta} \mathrm{erf}(\beta x) - \cos(x) \right]_{-2}^{4}$$

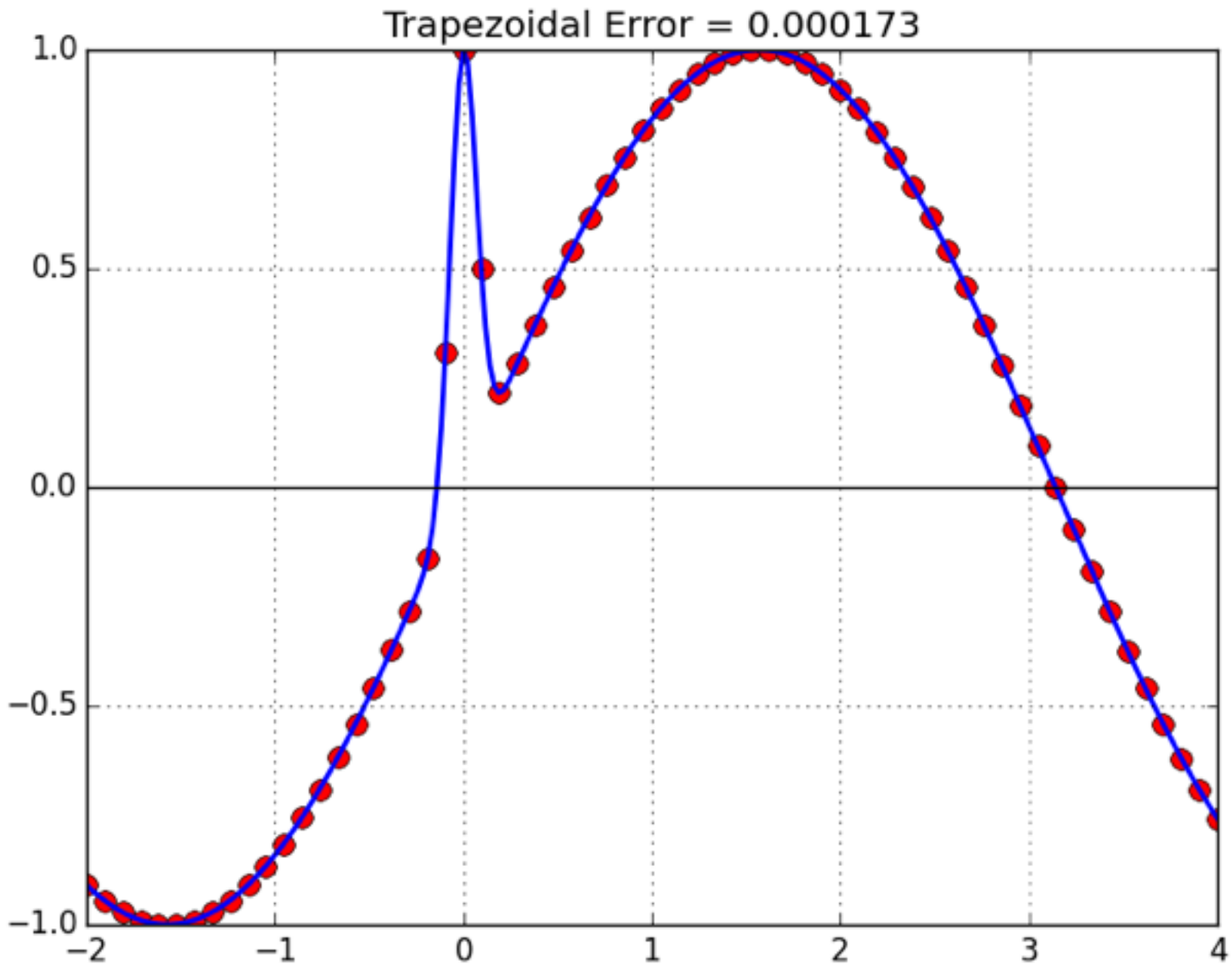
- where erf is the error function.

- beta = 10:



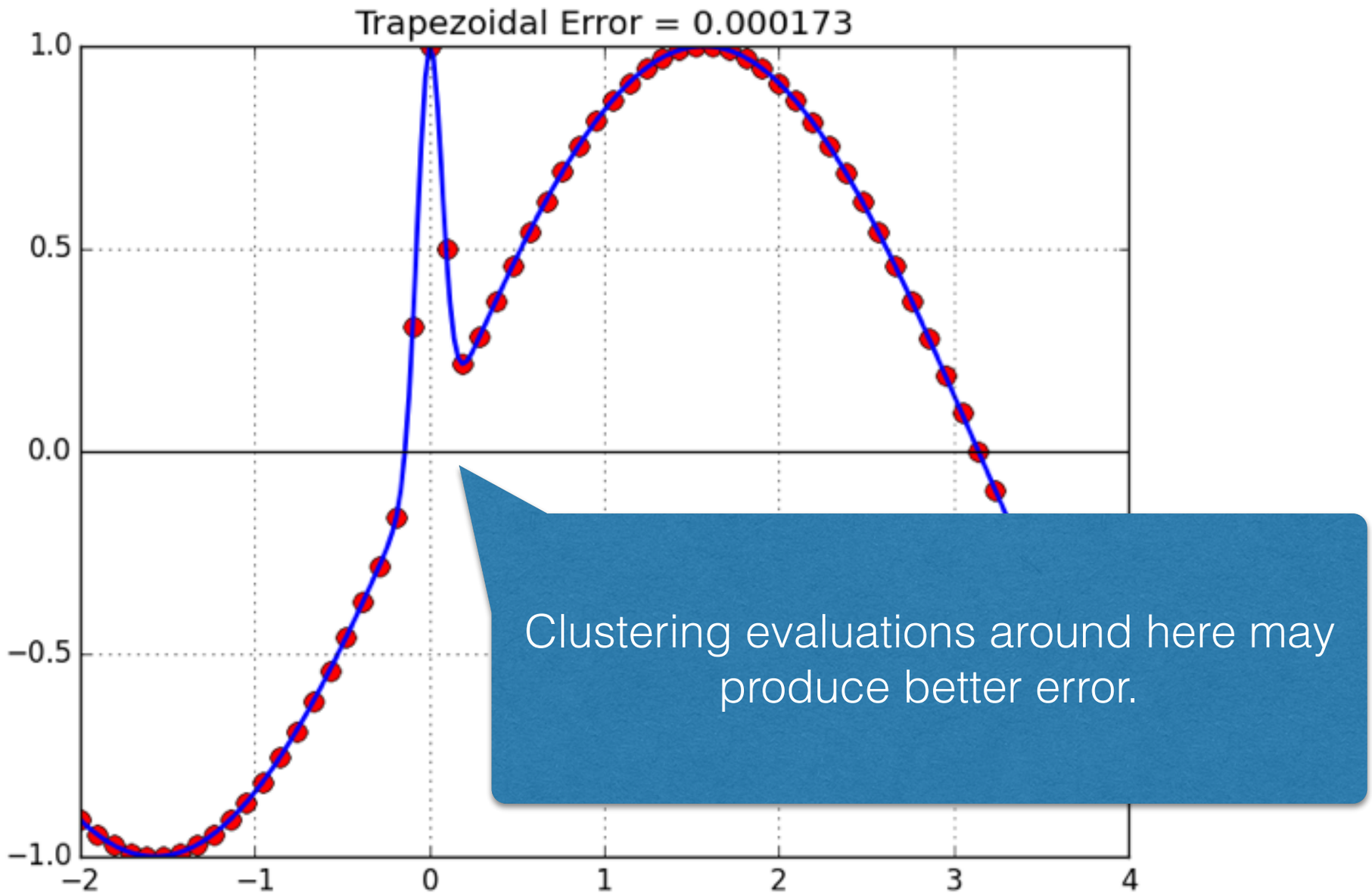
Estimating Integral

- Trapezoidal rule: n=64 points, equally spaced



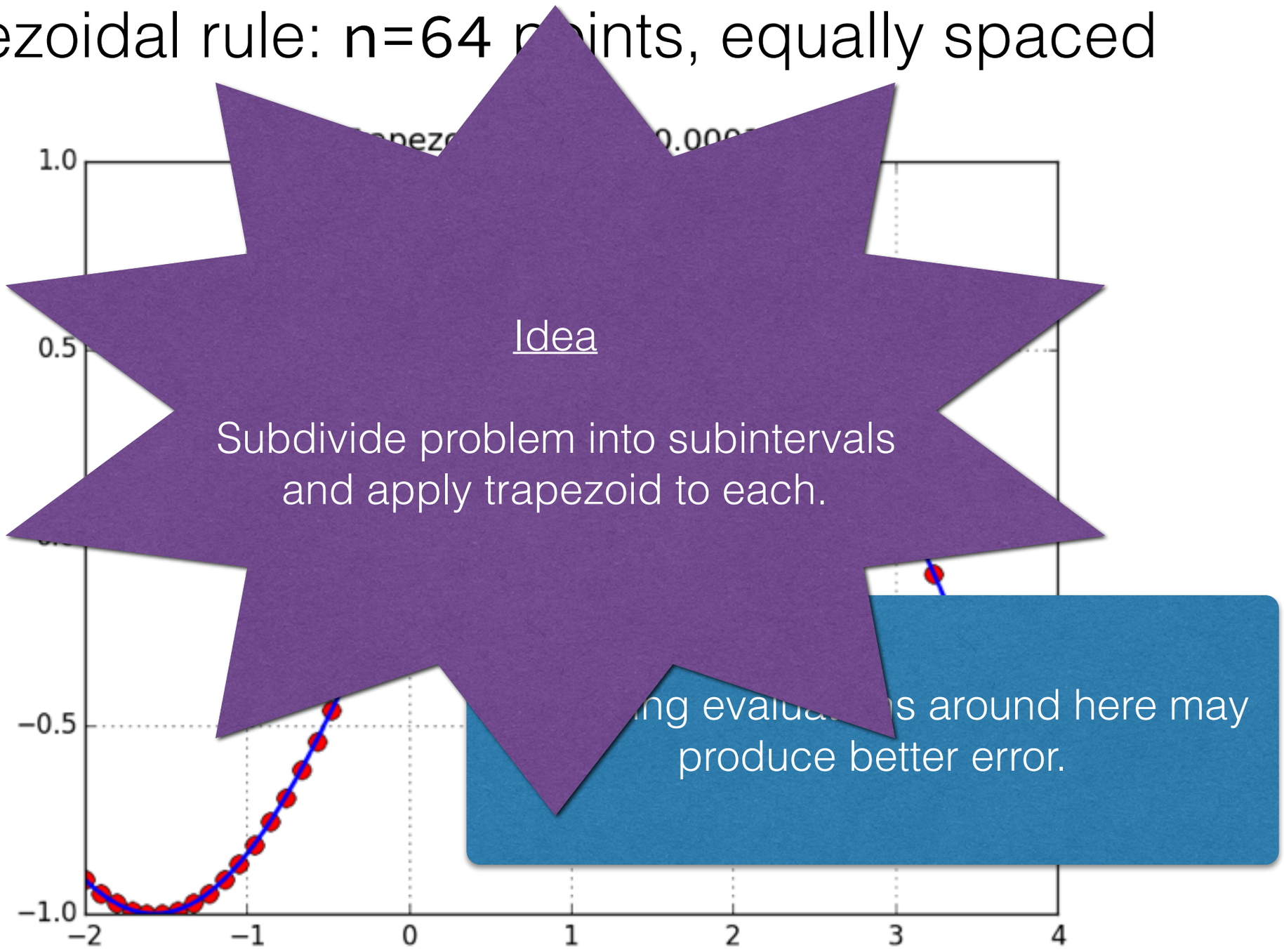
Estimating Integral

- Trapezoidal rule: n=64 points, equally spaced



Estimating Integral

- Trapezoidal rule: n=64 points, equally spaced



Idea

Subdivide problem into subintervals and apply trapezoid to each.

ng evalua s around here may produce better error.

Brief Aside

$$\int_a^b f(x)dx \approx \mathrm{trapz}(f, a, b)$$

- How to estimate approximation error?
- Compute both `trapz` and `simps` approx.
  - if abs. difference is large then **bad approx**.
  - if abs. difference is small then **good approx**.
  - Reason: $\mathrm{error} \sim O(|b-a|^n)$

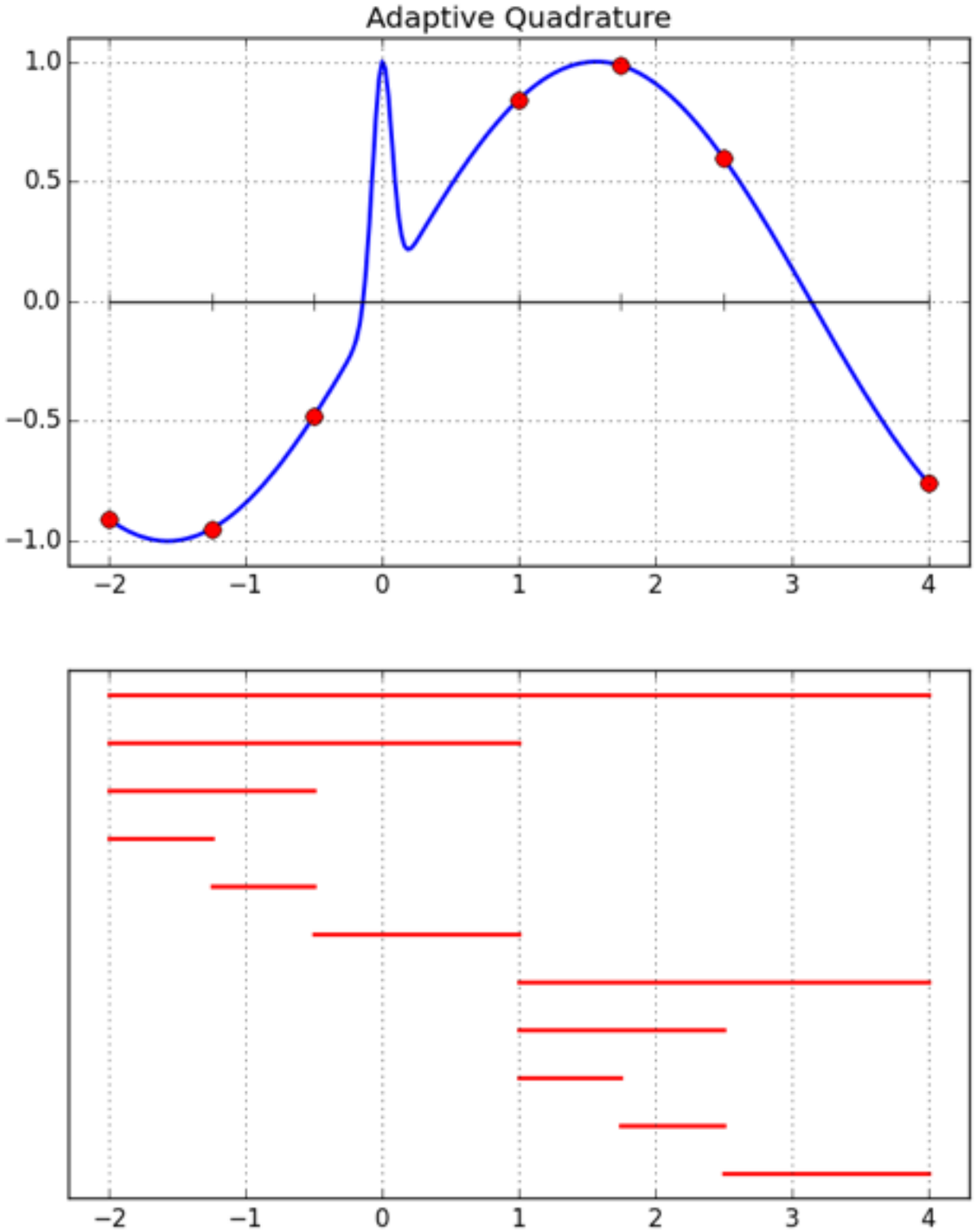
Central Idea

- What if approx. error > `tol` ?

$$\int_a^b f(x)dx = \int_a^{(a+b)/2} f(x)dx + \int_{(a+b)/2}^b f(x)dx$$

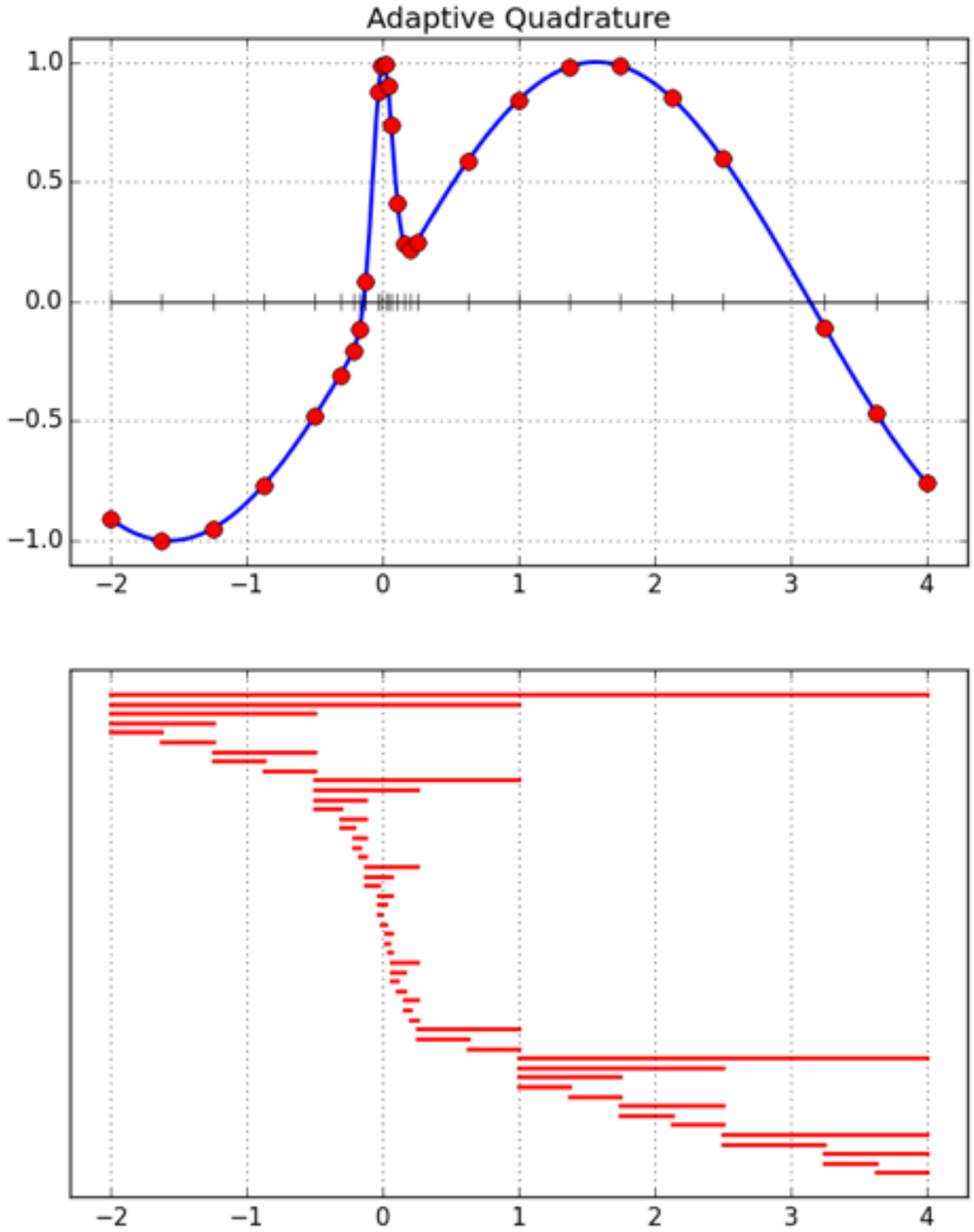
- Idea: split interval in half:
  - suppose error in each half is less than `tol/2`
  - then total error is less than `tol`
- If `err > tol/2` then divide that half of the interval and repeat

Quadrature `tol=0.5`




```
integral:   0.198245
(actual:    0.414742)
tolerance: 5.000000e-01
est error: 2.223337e-01
act error: 2.164973e-01
```

Quadrature `tol=0.1`




```
integral:   0.407417
(actual:    0.414742)
tolerance: 1.000000e-01
est error: 5.342968e-02
act error: 7.325371e-03
```

Demo

adaptive_serial.c and test_serial.c

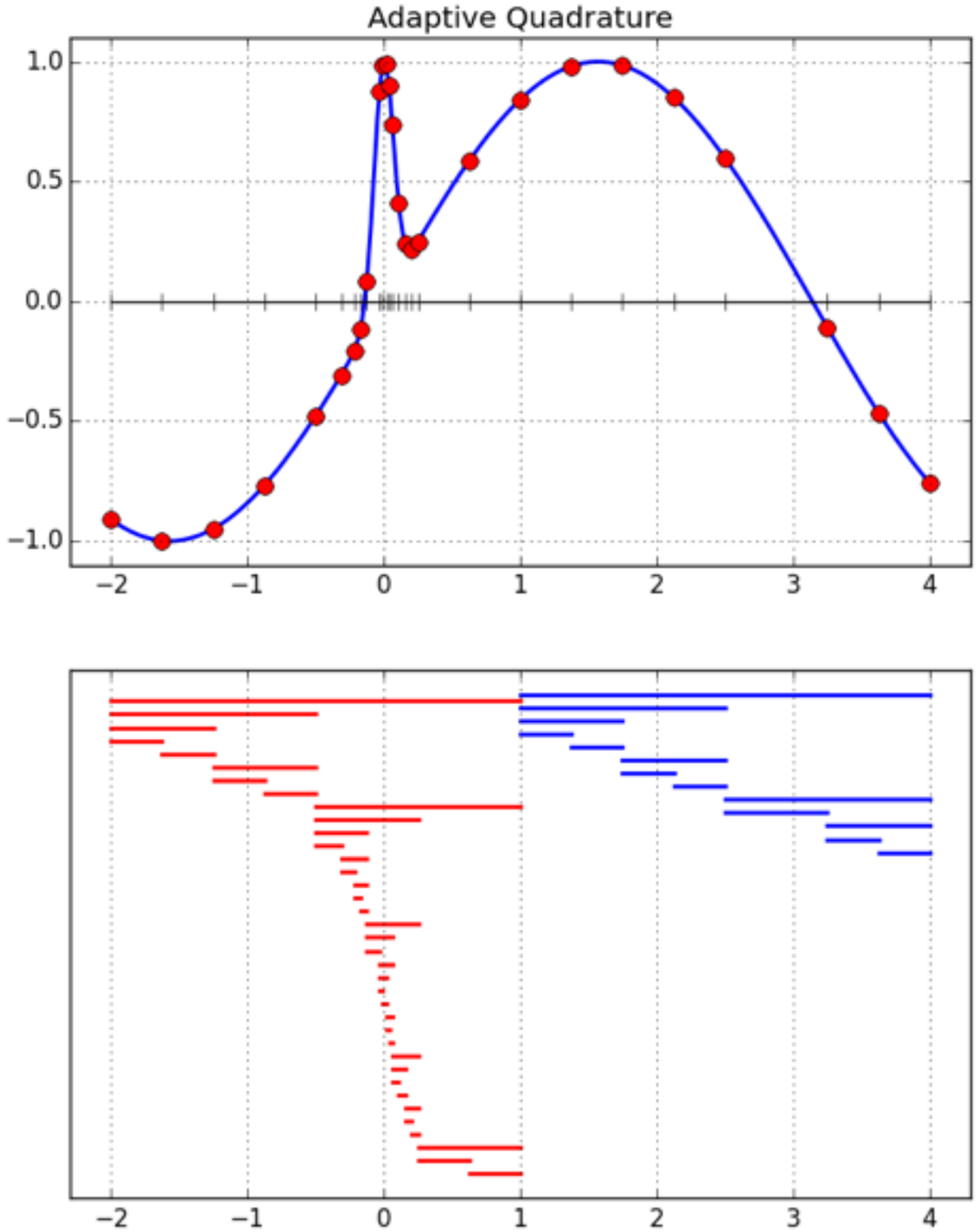
Parallelizing Using OpenMP

- **Naive Idea**: have each thread take a half of the domain
- “domain decomposition pattern”
- atomic / critical operations:
  - error calculation
  - integral summation

Demo

adaptive_serial.c and test_parallel.c

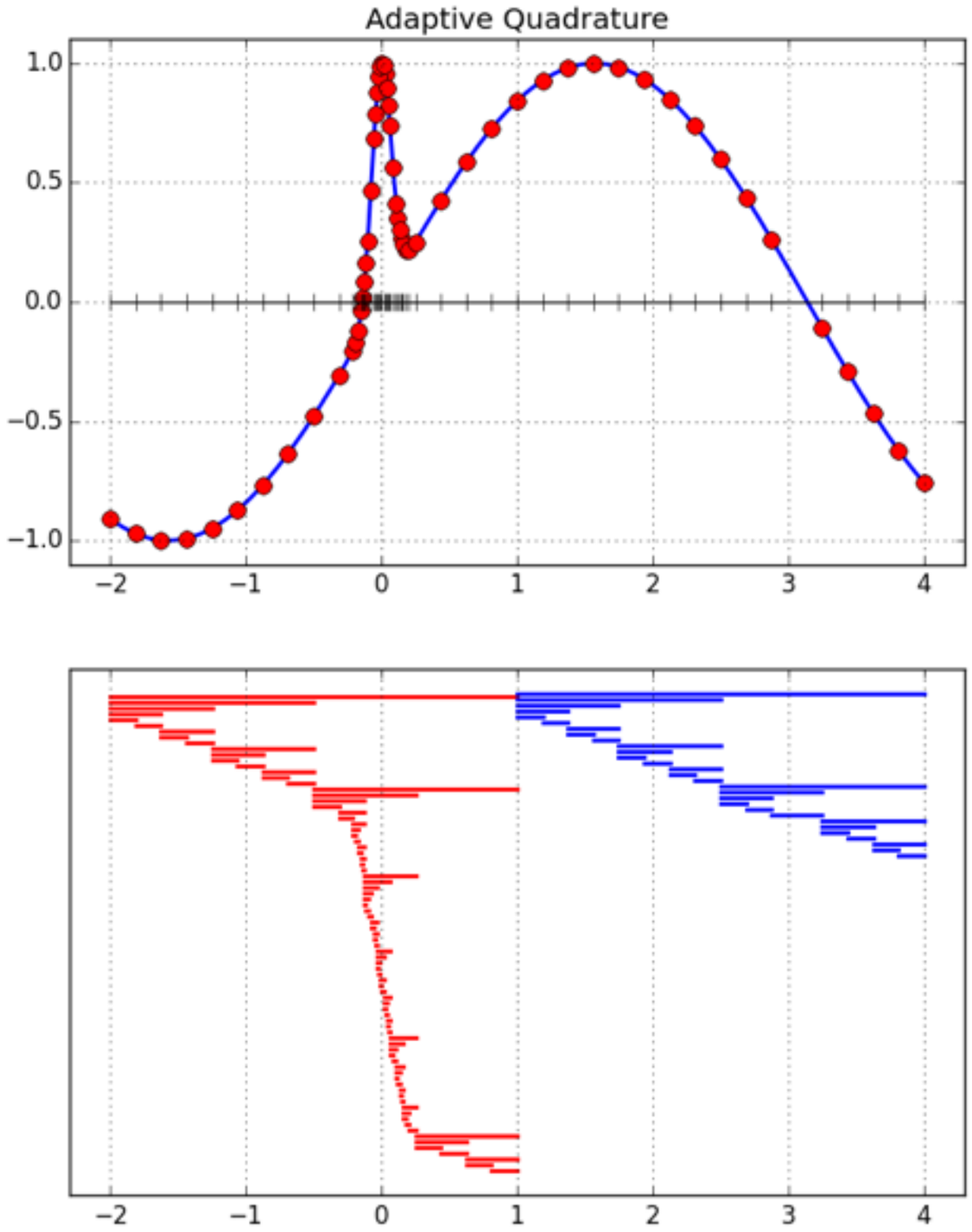
Par. Quad. `tol=0.1`




```
integral:  0.407417
(actual:   0.414742)
tolerance: 1.000000e-01
est error: 5.342968e-02
act error: 7.325371e-03
```

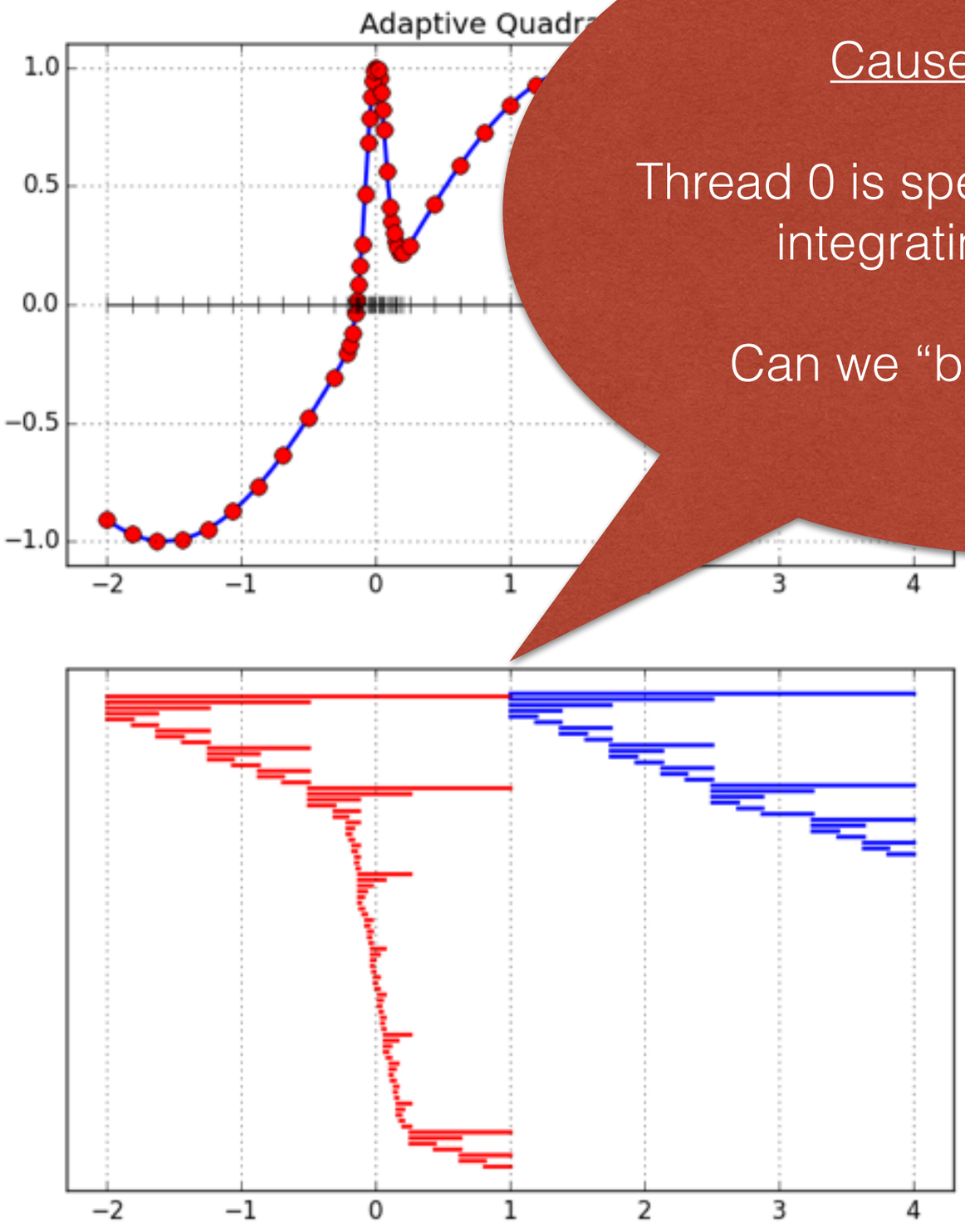
(Same as serial.)

Par. Quad. `tol=0.02`




```
integral:  0.414474
(actual:   0.414742)
tolerance: 2.000000e-02
est error: 1.189837e-02
act error: 2.678713e-04
```

Par. Quad[illegible]=0.02



Cause for Concern

Thread 0 is spending almost 3x time integrating as Thread 1.

Can we “balance” the load?

```
                0.414474
(actual:        0.414742)
tolerance: 2.000000e-02
est error: 1.189837e-02
act error: 2.678713e-04
```

OpenMP Sections

- Work-sharing Construct (like `omp for`) — each “section” is given to next available thread

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
      // block of code
    #pragma omp section
      // block of code
  }
}
```

OpenMP Sections

- Work-sharing Construct (like `omp for`) — each "section" is given to next available thread

```
#pragma omp parallel sections
{
  #pragma omp section
    // block of code
  #pragma omp section
    // block of code
}
```

Allowed to combine parallel and sections into one.

e.g.
`#pragma omp parallel for`

OpenMP Sections

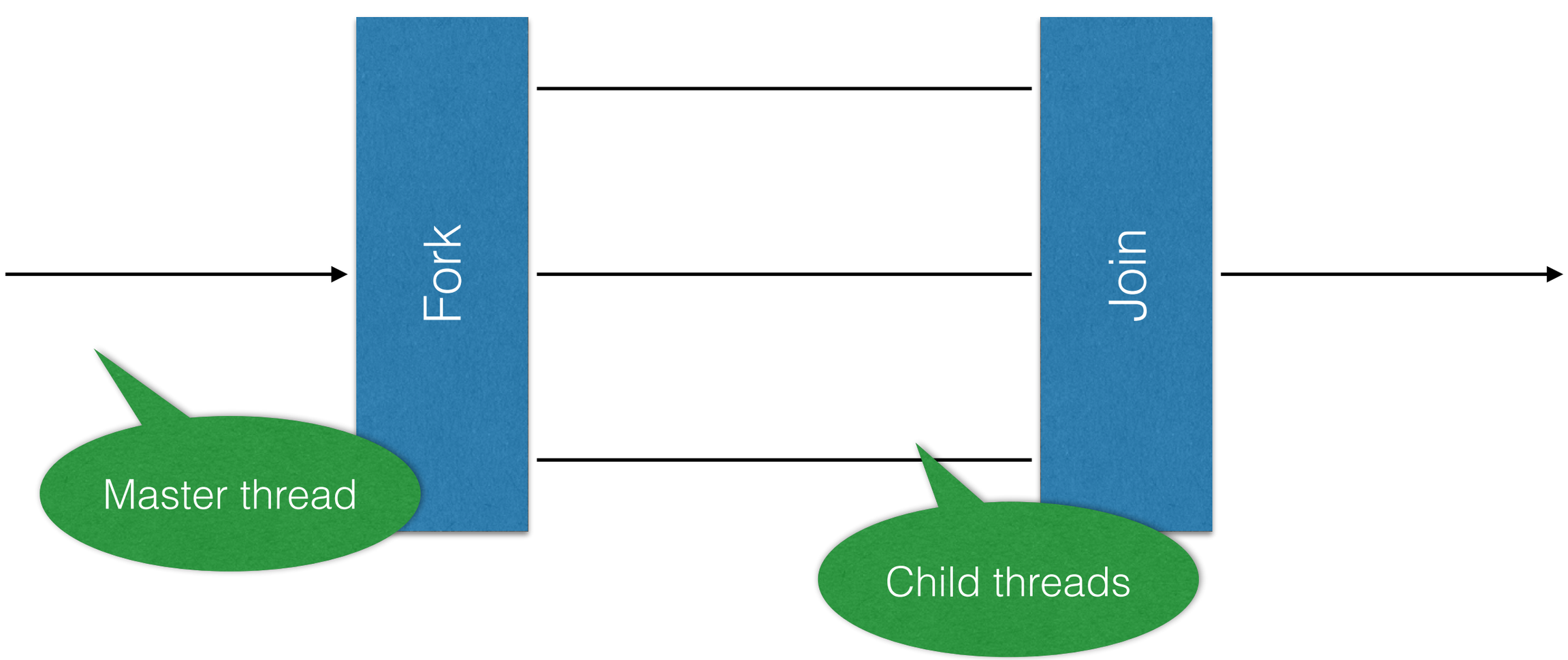
- What happens if `#threads =/= #sections`?
  - If more threads than sections?
  - If more sections than threads?

OpenMP Sections

- What happens if `#threads =/= #sections`?
  - If more threads than sections?
    - Some threads will execute sections and some won't.
- If more sections than threads?
  - OpenMP will allocate next available thread to next available section.

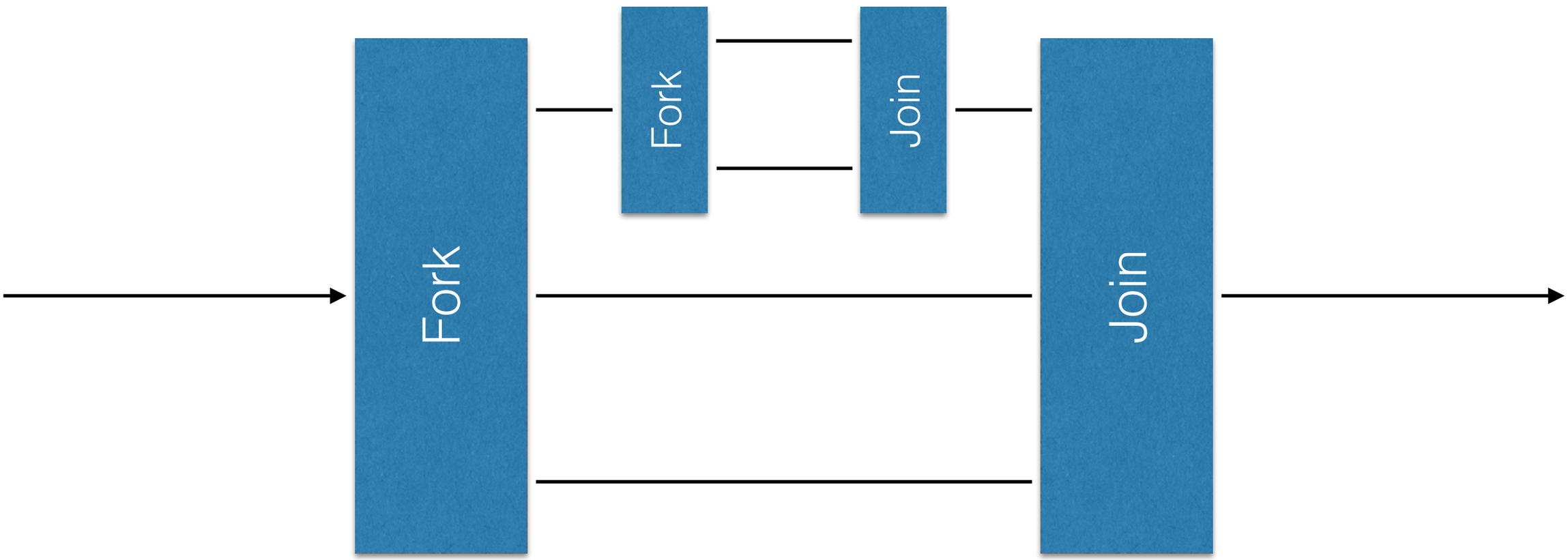
Nested Parallelism

- Fork-Join Model



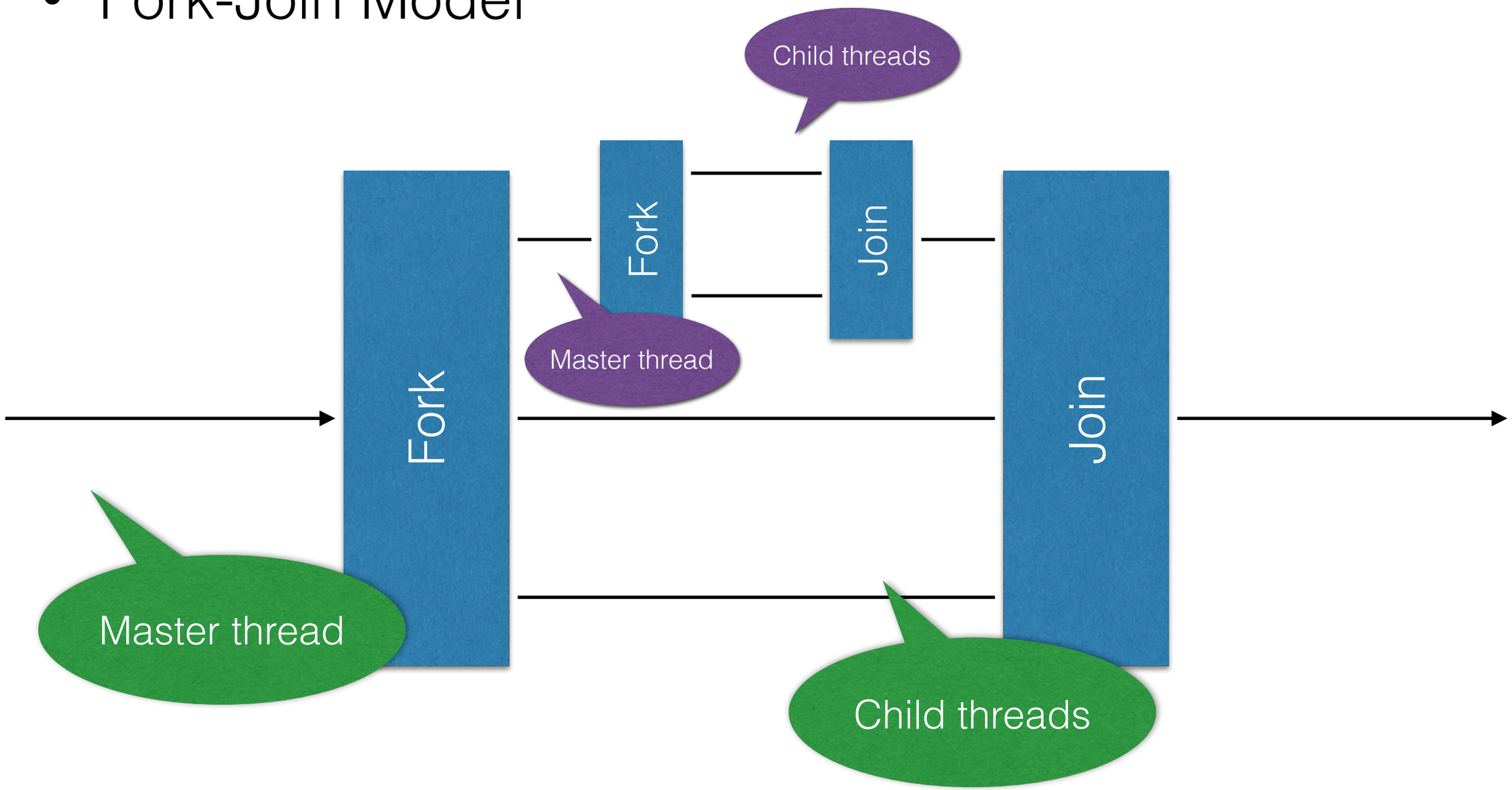
Nested Parallelism

- Fork-Join Model



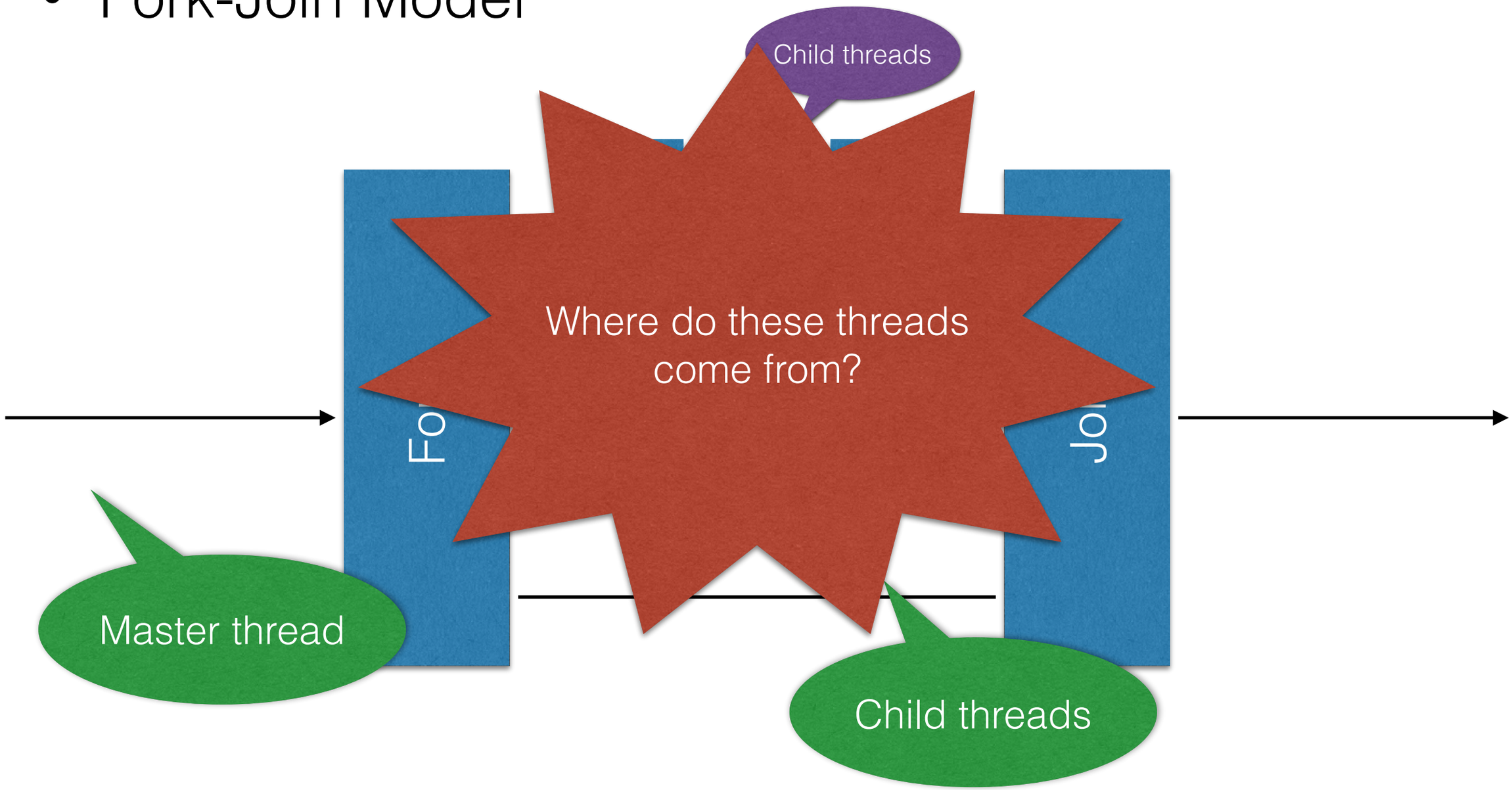
Nested Parallelism

- Fork-Join Model



Nested Parallelism

- Fork-Join Model



Nested Parallelism

- OpenMP maintains a *“pool”* of threads
  - encounter `omp parallel` —> take idle threads
  - encounter another `omp parallel` —> take idle threads
- After JOIN, threads are returned to pool.

Nested Parallelism

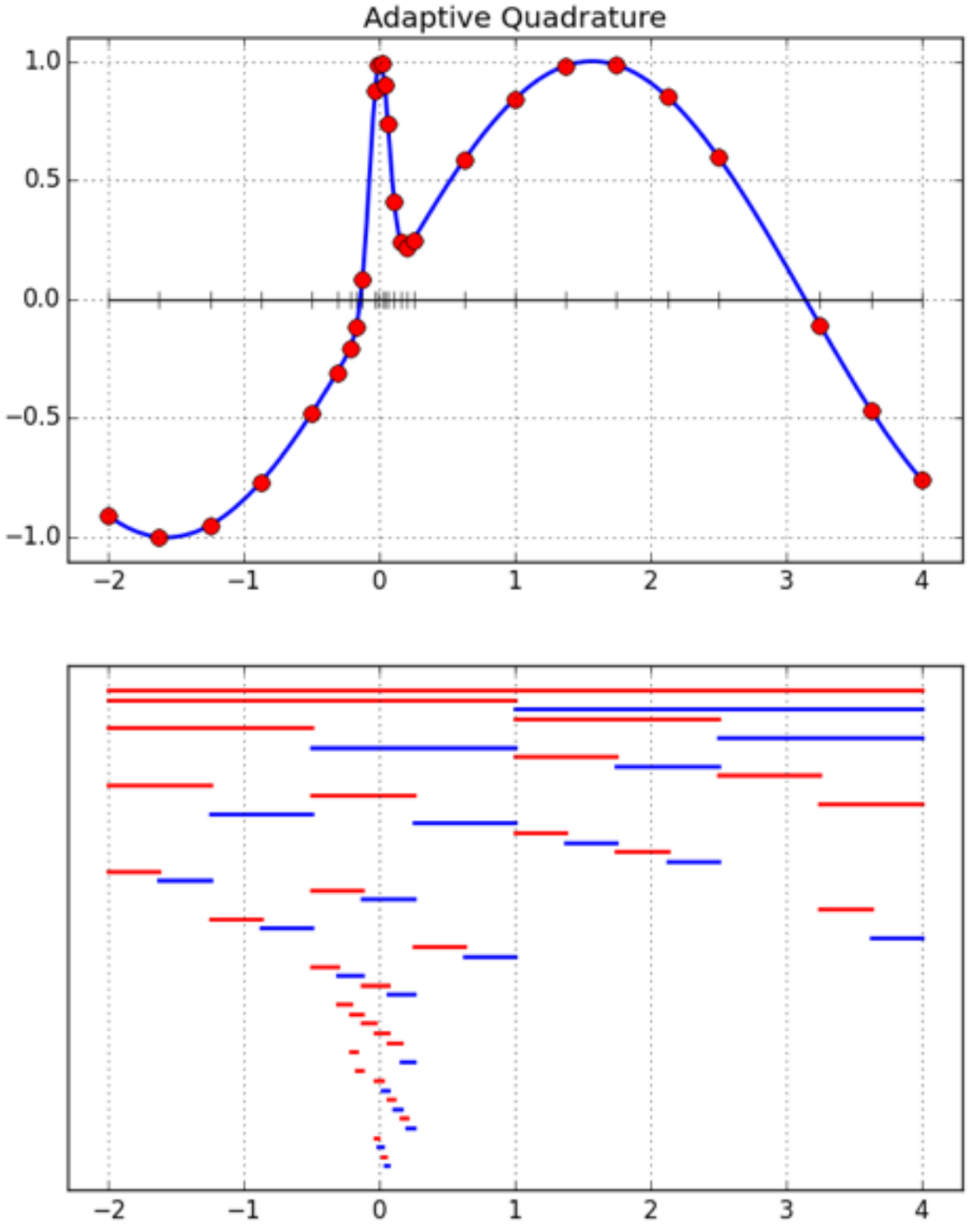
- Possible to create many total threads!

- Many ways to deal with this, see online resources.

Demo

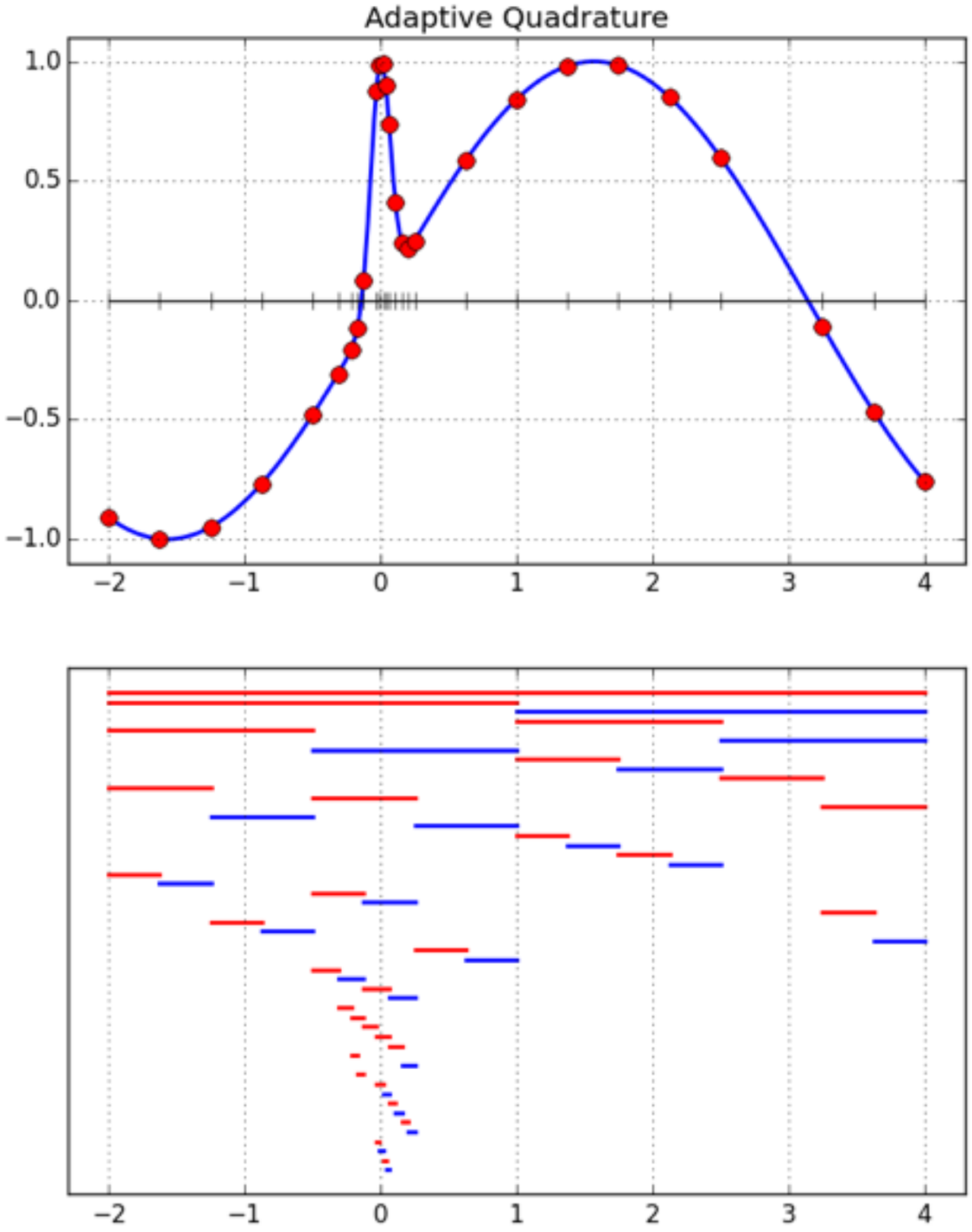
adaptive_parallel.c and test_balanced.c

Balanced Quad. `tol=0.1`




```
num_threads = 2
integral:  0.407417
(actual:   0.414742)
tolerance: 1.000000e-01
est error: 5.342968e-02
act error: 7.325371e-03
```

Balanced Quad. `tol=0.1`

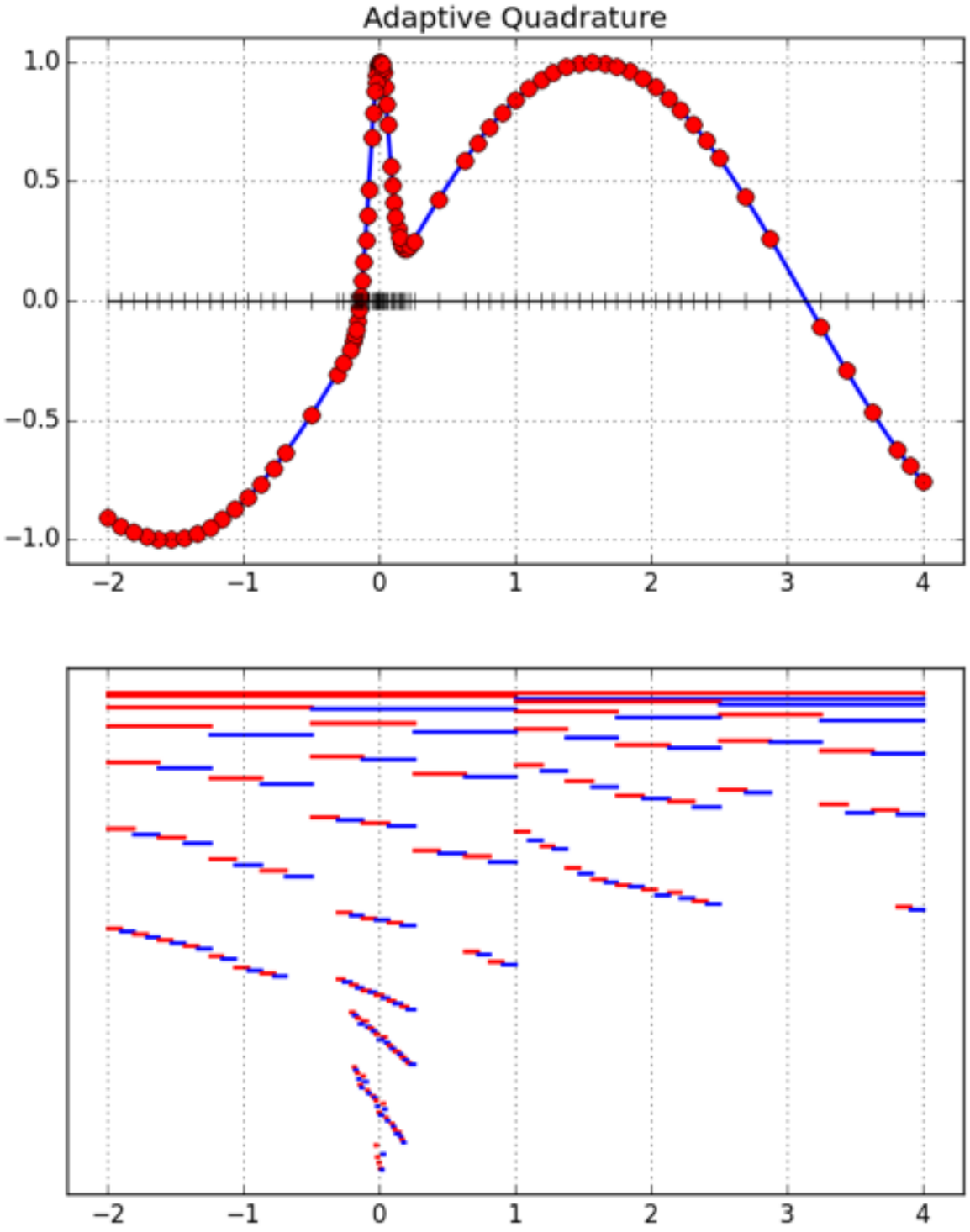



```
nu
in
(a
to
es
```

Work is shared evenly between Thread 0 and Thread 1.

(Each nested `omp section` given to next available thread.)

Balanced Quad. `tol=0.01`




```
num_threads = 2
integral:  0.414743
(actual:   0.414742)
tolerance: 1.000000e-02
est error: 4.902247e-03
act error: 9.255080e-07
```

Thoughts on MPI

- OpenMP has built in tools for thread management
- Possible with MPI via `MPI_Comm_spawn()` and master —> child design pattern
- Many recursive functions can be re-implemented w/o recursion using a manual stack.
    - parallel producer — consumer model

Thoughts on MPI