Imperial College London

EPSRC
Engineering and Physical Sciences Research Council

PyFR: An Open Source Python Framework for High-Order CFD on Many-Core Platforms

F. Witherden, A. Farrington, P. E. Vincent

Department of Aeronautics
Imperial College London

23rd May 2013

Introduction

- Various types of high-order accurate numerical methods for unstructured grids
- Continuous Galerkin (CG), Discontinuous Galerkin (DG), Spectral Difference (SD), Spectral Volume (SV), Flux Reconstruction (FR)

Introduction

- Various types of high-order accurate numerical methods for unstructured grids
- Continuous Galerkin (CG), Discontinuous Galerkin (DG), Spectral Difference (SD), Spectral Volume (SV), Flux Reconstruction (FR)

Introduction

- FR approach was first proposed by Huynh at NASA Glenn in 2007 [1]
- Based on differential form of governing system
- Unifying, can cast various known + new schemes within a single framework

[1] H.T. Huynh. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. AIAA Paper 2007-4079. 2007

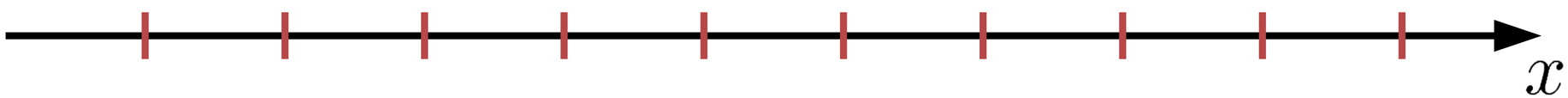
Theory

- Consider 1D scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0$$

- Divide 1D domain into elements

$$\Omega = \bigcup_{n=0}^{N-1} \Omega_n \qquad \bigcap_{n=0}^{N-1} \Omega_n = \emptyset \qquad \Omega_n = \{x | x_n < x < x_{n+1}\}$$



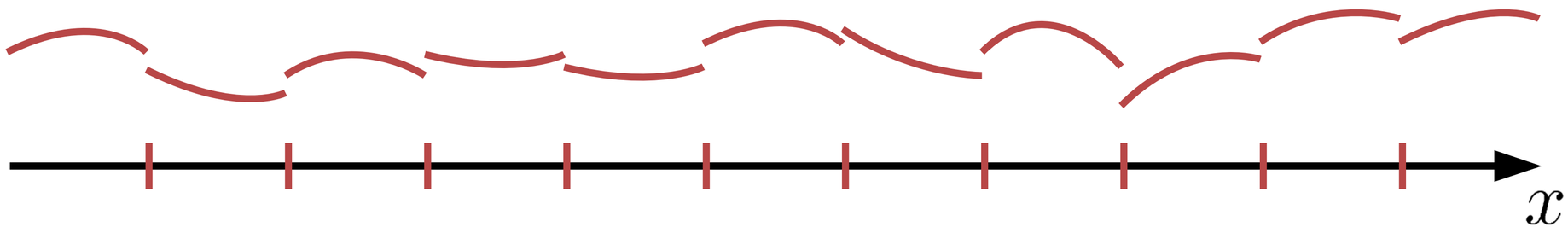
Theory

- Consider 1D scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0$$

- Represent solution by order k piecewise discontinuous polynomials in each element

$$u^\delta = \bigoplus_{n=0}^{N-1} u_n^\delta \approx u$$



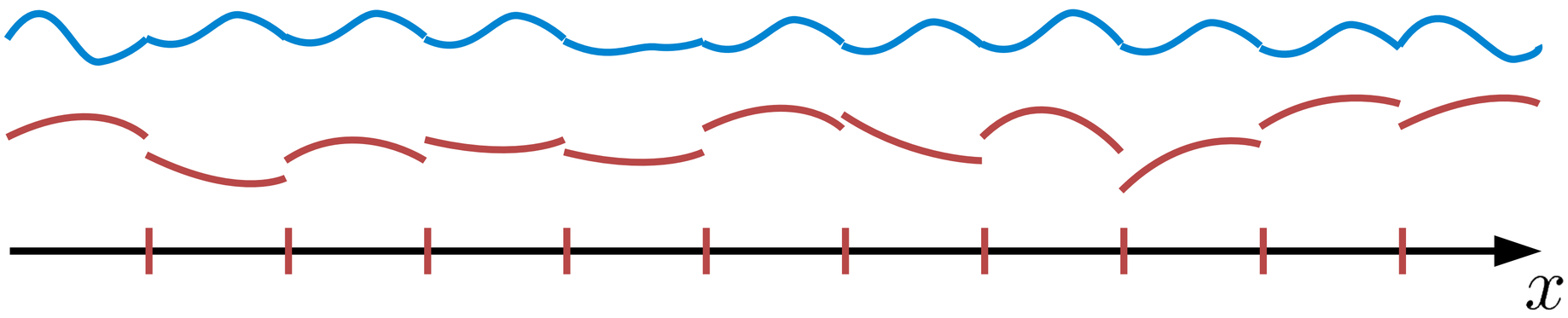
Theory

- Consider 1D scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0$$

- Represent flux by order k+1 piecewise continuous polynomials within each element

$$f^\delta = \bigoplus_{n=0}^{N-1} f_n^\delta \approx f$$



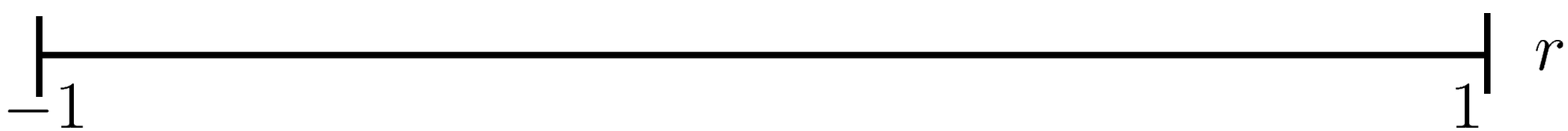
Theory

- Map each element to a standard element

$$\Omega_S = \{r | -1 \leq r \leq 1\} \qquad r = 2\left(\frac{x - x_n}{x_{n+1} - x_n}\right) - 1$$

$$\frac{\partial \hat{u}^\delta}{\partial t} + \frac{\partial \hat{f}^\delta}{\partial r} = 0 \qquad \hat{u}^\delta = u_n^\delta \qquad \hat{f}^\delta = \frac{f_n^\delta}{h_n}$$

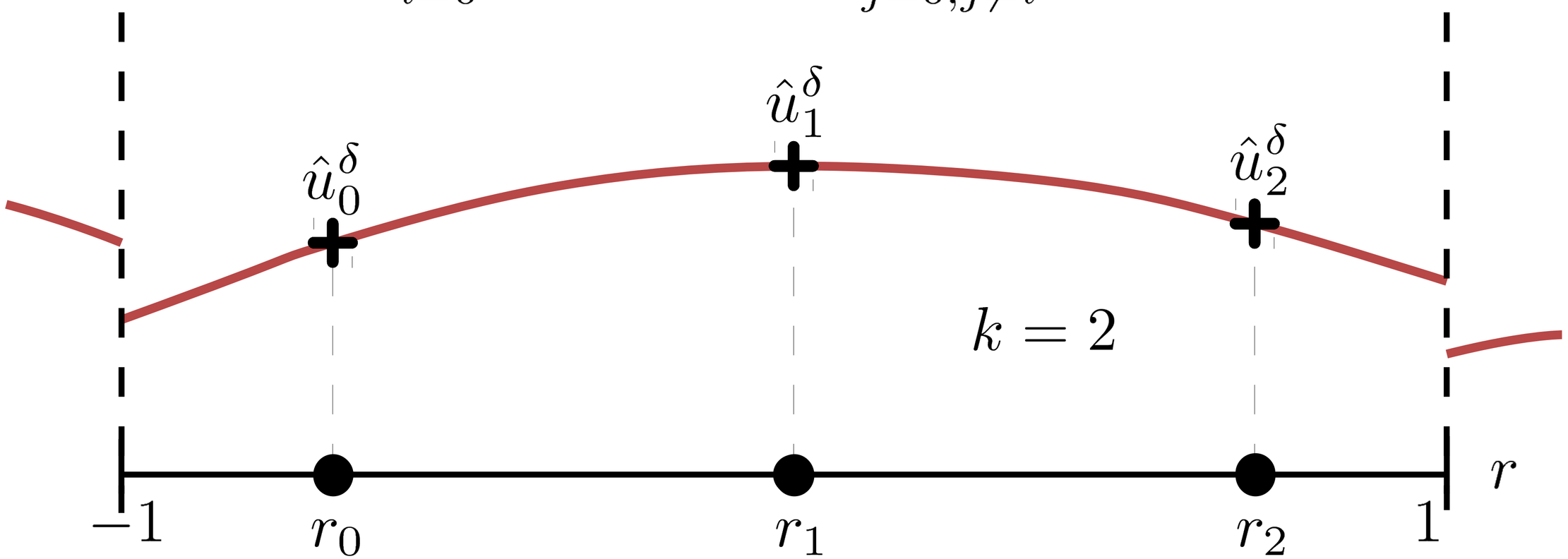
$$h_n = (x_{n+1} - x_n)/2$$



Theory

- Represent order k solution within standard element using a nodal basis

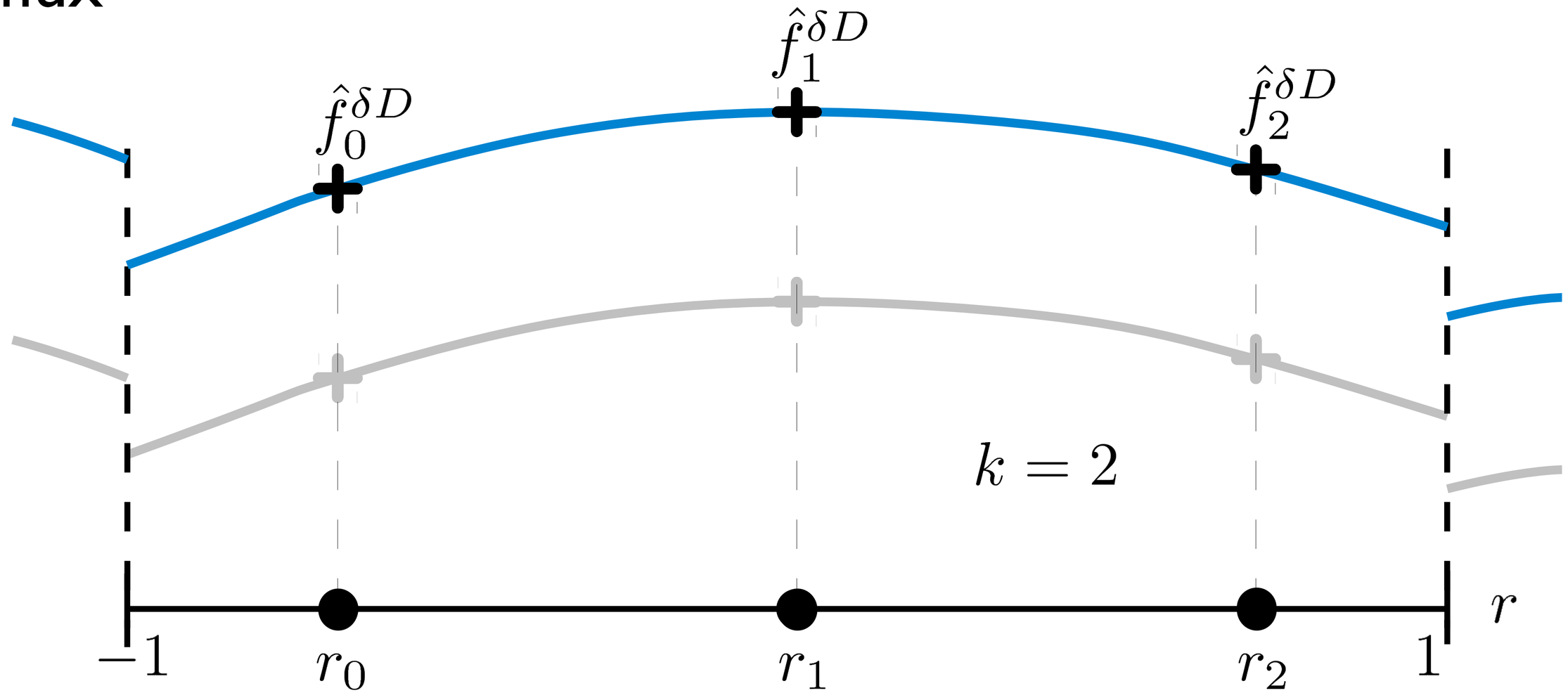
$$\hat{u}^{\delta} = \sum_{i=0}^{k} \hat{u}_i^{\delta} \, l_i \qquad l_i = \prod_{j=0, j \neq i}^{k} \left(\frac{r - r_j}{r_i - r_j} \right)$$



Theory

- Reconstruct discontinuous flux

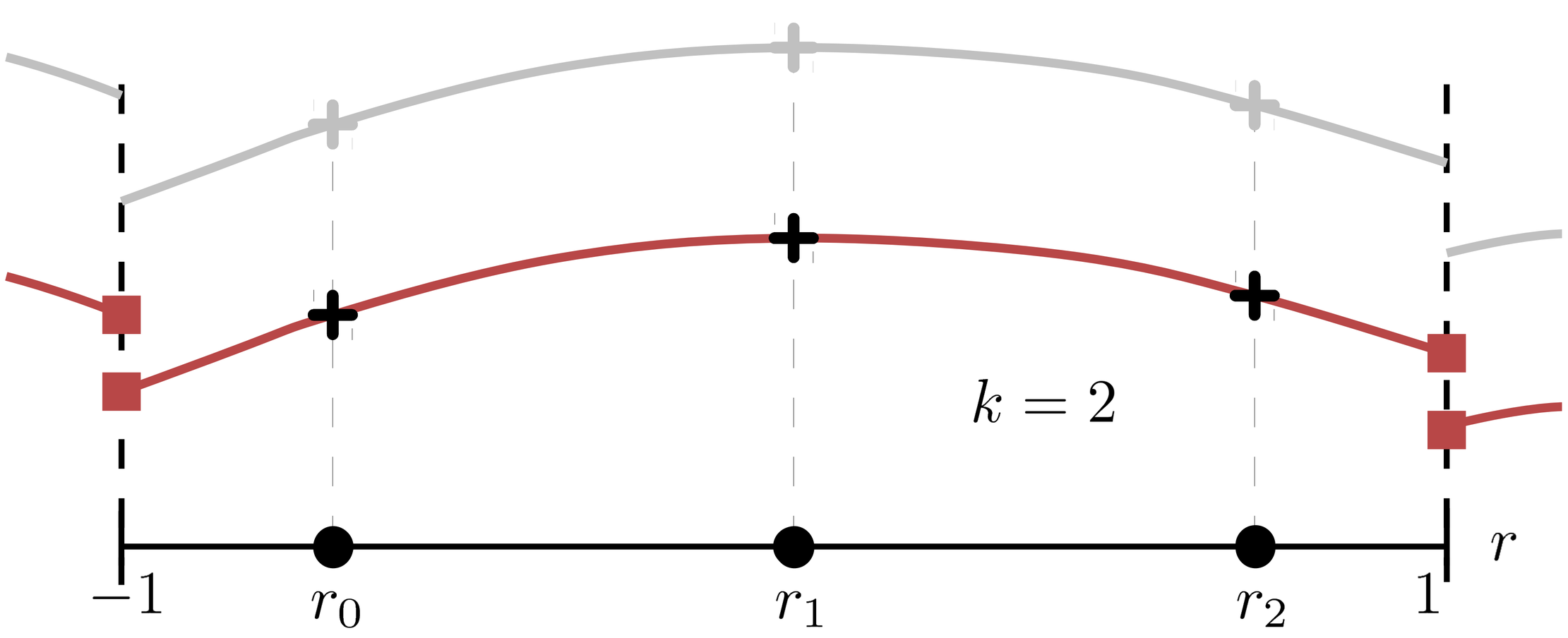
$$\hat{f}^{\delta D} = \sum_{i=0}^{k} \hat{f}_i^{\delta D} \, l_i$$



Theory

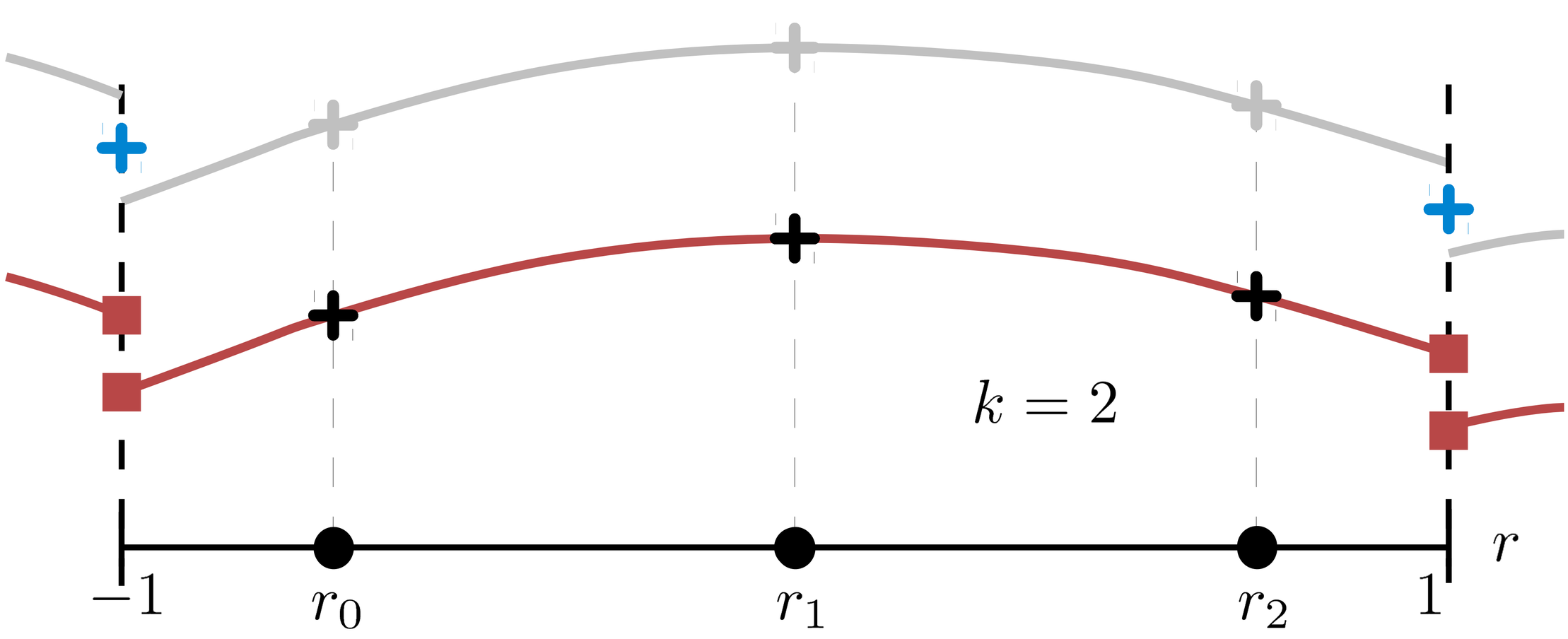
- Evaluate solution at element boundaries

$$\hat{u}^{\delta}(-1) = \sum_{i=0}^{k} \hat{u}_i^{\delta}\ l_i(-1) \qquad \hat{u}^{\delta}(1) = \sum_{i=0}^{k} \hat{u}_i^{\delta}\ l_i(1)$$



Theory

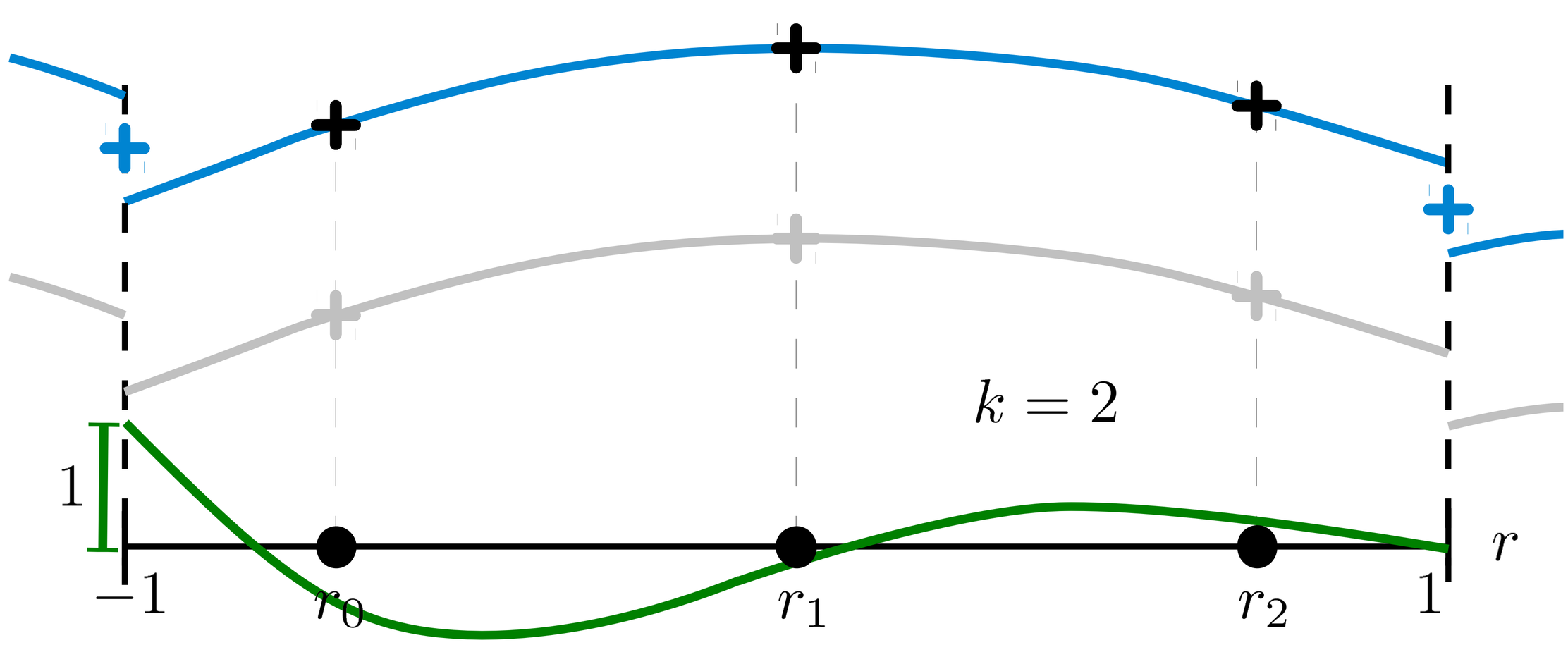
- Calculate common interface fluxes



Theory

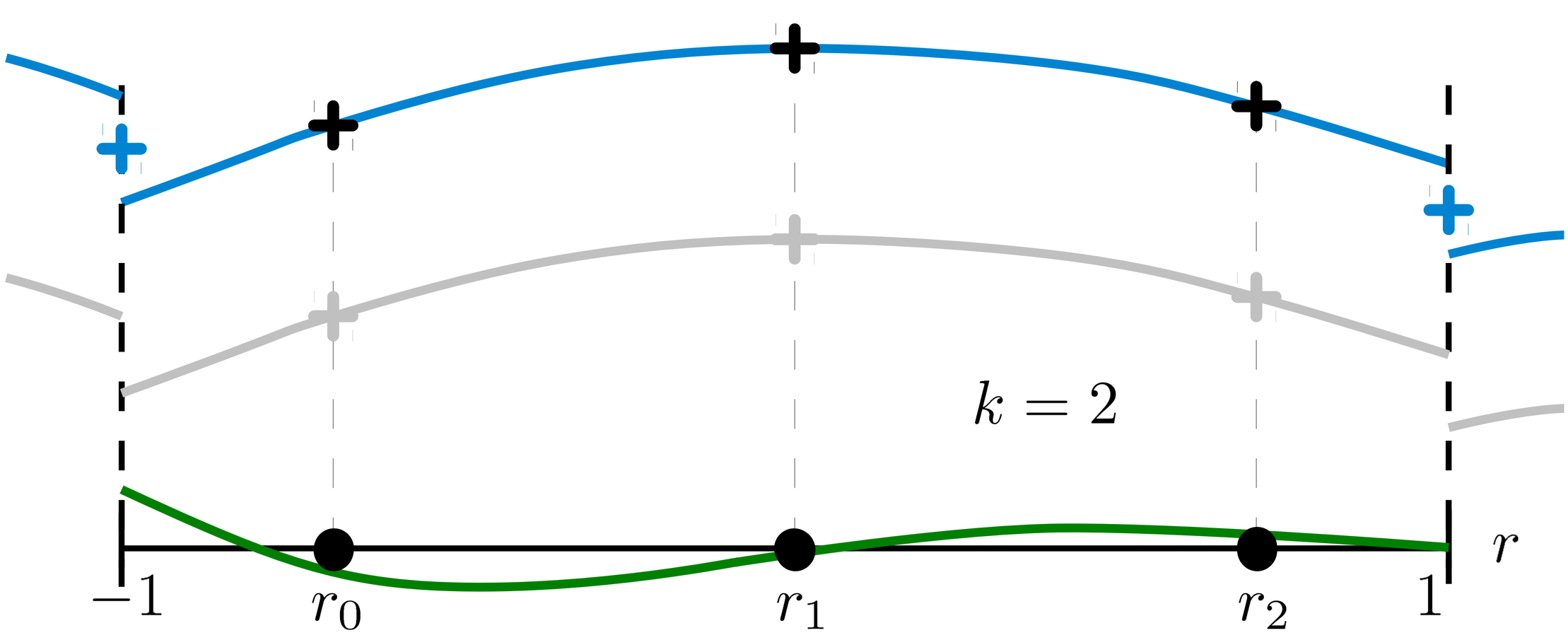
- Define order k+1 'correction function'

$$g_L(-1) = 1, \qquad g_L(1) = 0$$



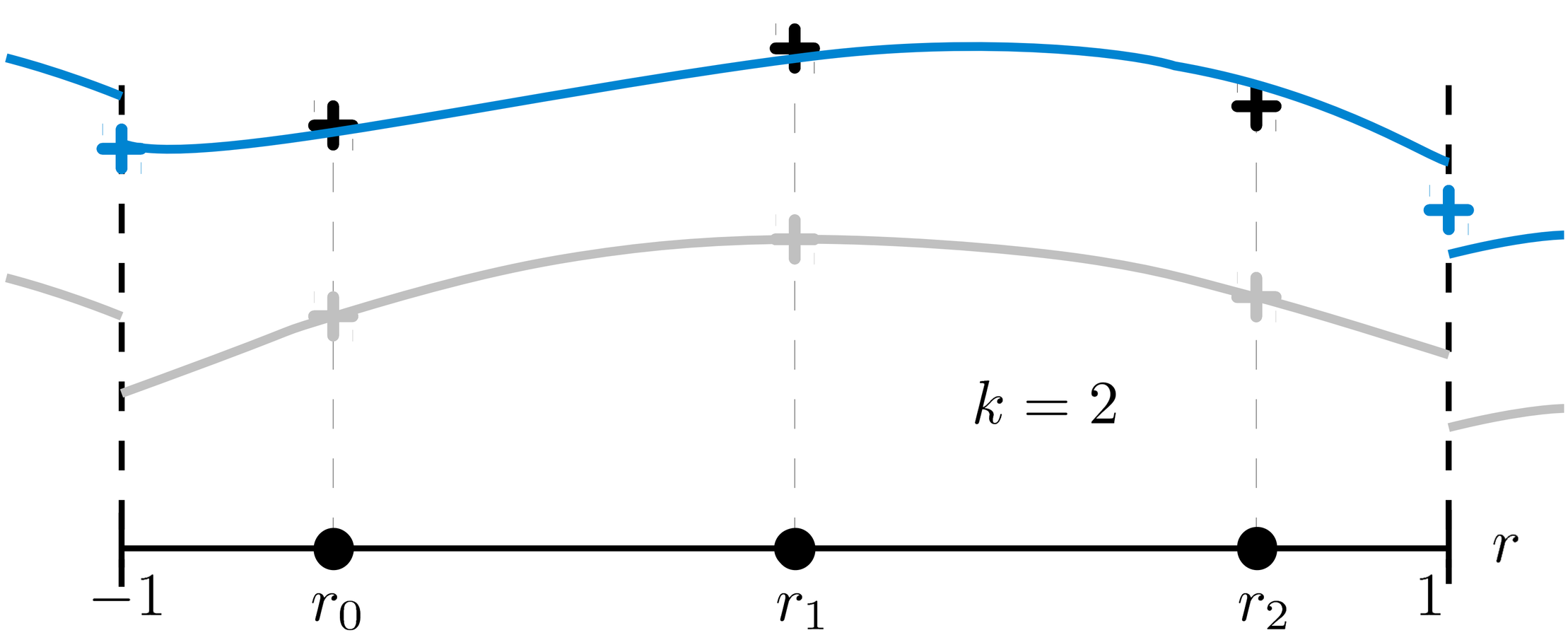
Theory

- Scale this correction function...



Theory

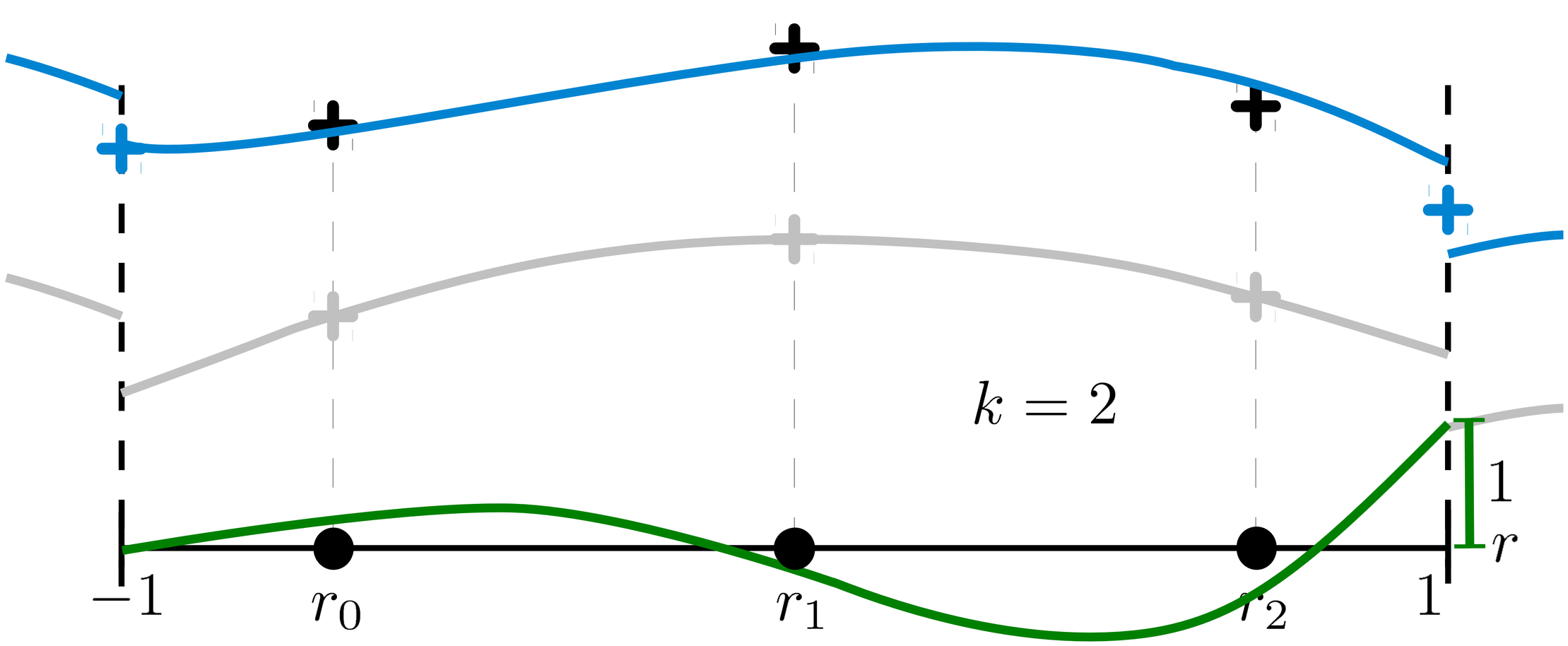
- ... and add it to the discontinuous flux



Theory

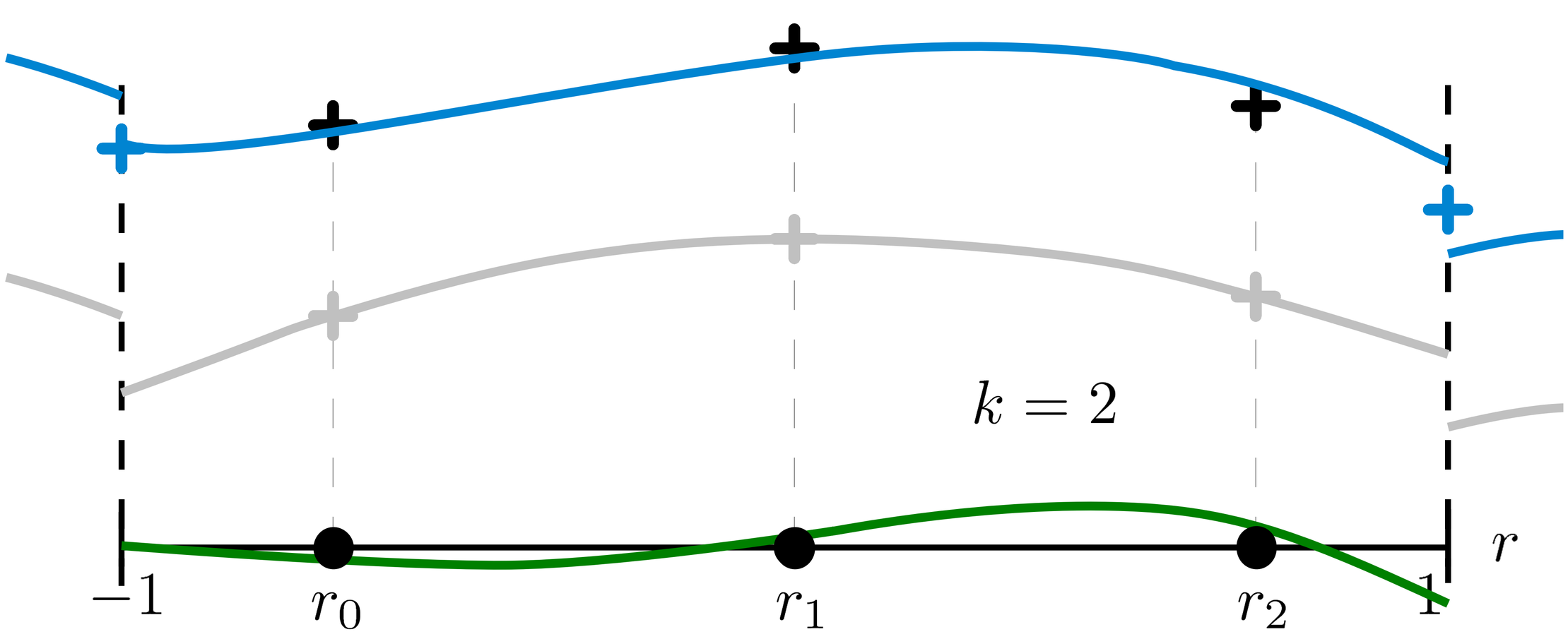
- Repeat from the right...

$$g_R(r) = g_L(-r)$$



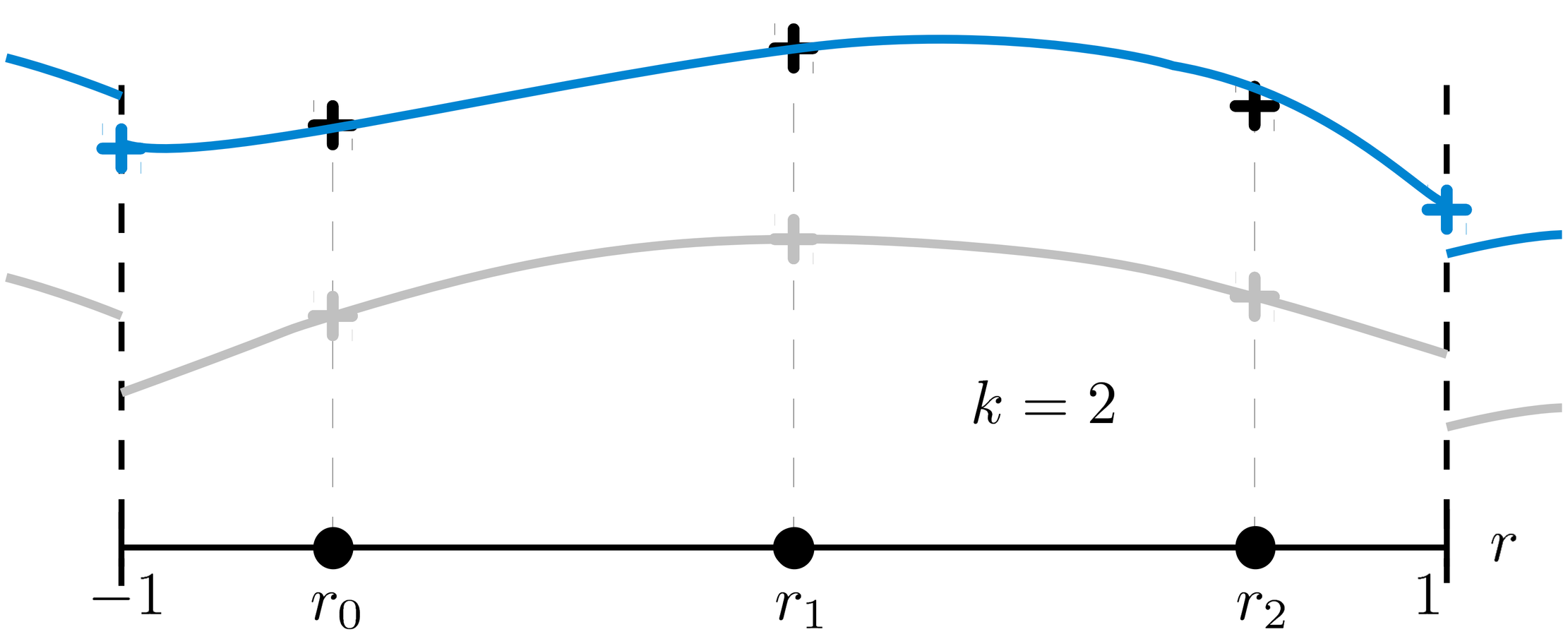
Theory

- Repeat from the right...



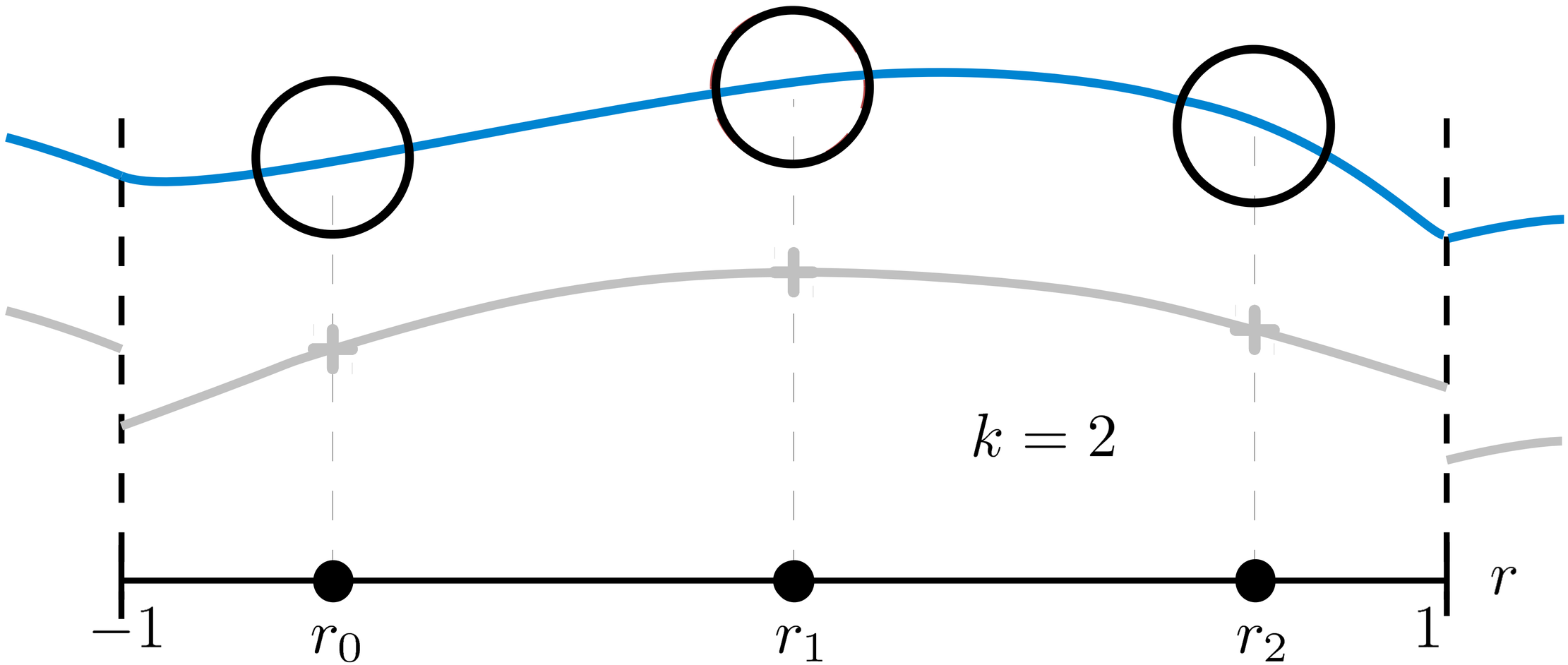
Theory

- Repeat from the right...



Theory

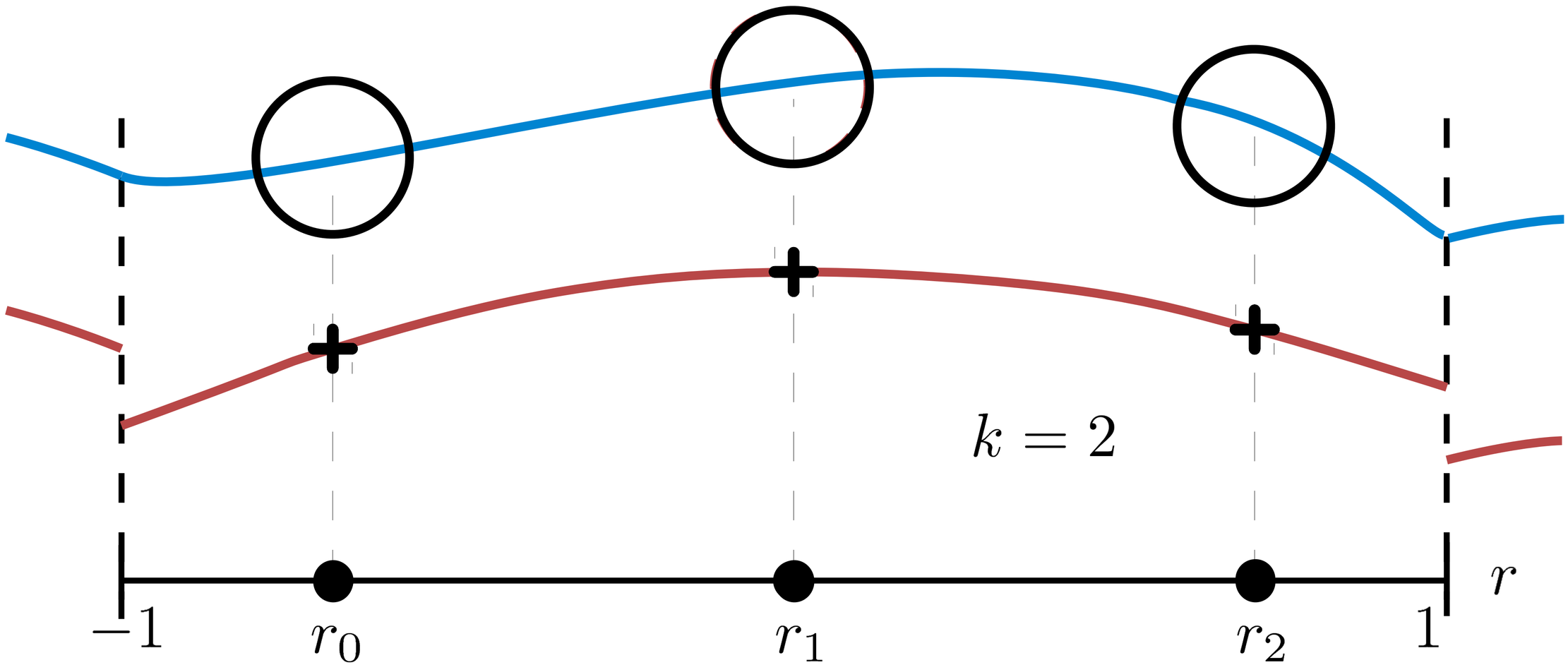
- Evaluate divergence of the continuous flux at the solution points



Theory

- And advance the solution in time...

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0 \quad \longrightarrow \quad \frac{\partial \hat{u}_i^{\delta}}{\partial t} = -\frac{\partial \hat{f}^{\delta}}{\partial r}(r_i)$$



Theory

- Nature of FR scheme depends on location of solution points, interface flux, correction function
- Can recover nodal Discontinuous Galerkin (DG) method, and any Spectral Difference (SD) method via judicious choice of correction function [1]

[1] H.T. Huynh. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. AIAA Paper 2007-4079. 2007

Theory

- In 2011 Vincent, Castonguay and Jameson identified a range of stable correction functions for all orders of accuracy using an energy method [2][3]

[2] P. E. Vincent, P. Castonguay, A. Jameson. A New Class of High-Order Energy Stable Flux Reconstruction Schemes. Journal of Scientific Computing. 2011
[3] A. Jameson, P. E. Vincent, P. Castonguay. On the Non-Linear Stability of High-Order Flux Reconstruction Schemes. Journal of Scientific Computing. 2011

Theory

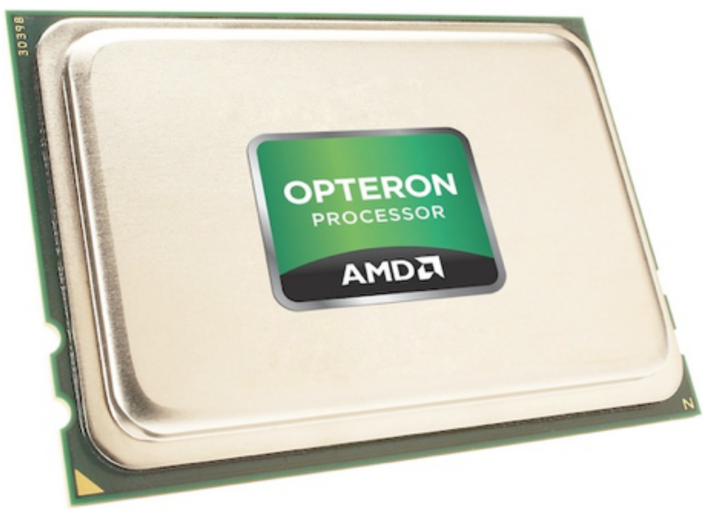
- Most operations point/element local
- No explicit quadrature (c.f. DG)
- No global assembly (c.f. FEM)
- No matrix inversions

PyFR

- Recent rapid evolution of hardware available for scientific computing – exciting times

PyFR

- Multi-core CPUs now ubiquitous (with 10's of cores per device)
- Rely extensively on vectorization for throughput



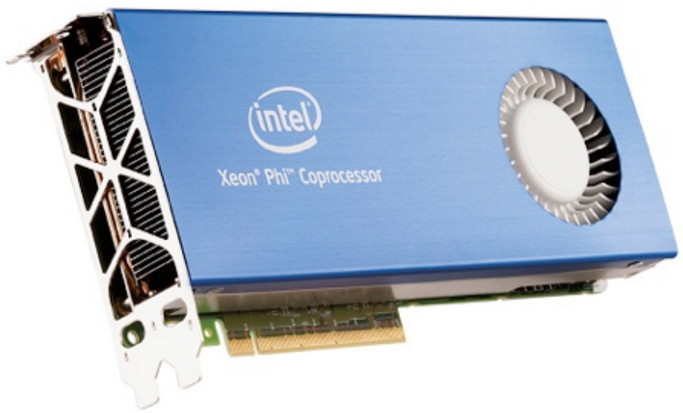
PyFR

- Many-core accelerators a hot topic (100s - 1000s cores per device)

PyFR

- GPUs may or may not be the future... but many-core + low memory-per-core quite likely is

Nvidia K20



Intel Phi

AMD S10000

PyFR

- What approach have we adopted?
- Write platform-independent code in Python and interface with accelerator via bespoke kernels
- Leverage the near universal availability of optimized BLAS libraries for CPUs/accelerators

PyFR

- Current status

Capabilities

- Compressible 2D/3D Navier-Stokes
- Unstructured grids of quads or hexahedra
- High-order FR in space
- Explicit Runge-Kutta in time

Implementation

- Setup (SymPy + NumPy)
- Kernel generation (Mako)
- Kernel invocation (PyCUDA &c.)
- Communication (mpi4py)

CUDA

C/OpenMP

PyFR

- Current status
- 4000 lines of Python
- 800 lines of Mako/CUDA
- 800 lines of Mako/C
- Slated for release under a BSD license in late summer

PyFR

- Theory of VCJH correction functions

$$\eta_k \in \left\{0, \frac{k}{k+1}, \frac{k+1}{k}, \dots\right\} \qquad g_R' = \frac{1}{2}\frac{d}{dr}\left[L_k + \frac{\eta_k L_{k-1} + L_{k+1}}{1+\eta_k}\right]$$

- Implementation using SymPy

```
def diff_vcjh_correctionfn(k, eta, sym):
    # Expand shorthand forms of eta_k for common schemes
    etacommon = dict(dg='0', sd='k/(k+1)', hu='(k+1)/k')

    eta_k = sy.S(etacommon.get(eta, eta), locals=dict(k=k))

    lkm1, lk, lkp1 = [sy.legendre_poly(m, sym) for m in [k-1, k, k+1]]

    # Correction function derivatives, Eq. 3.46 and 3.47
    diffgr = (sy.S(1)/2 * (lk + (eta_k*lkm1 + lkp1)/(1 + eta_k))).diff()
    diffgl = -diffgr.subs(sym, -sym)

    return diffgl, diffgr
```

PyFR

- Kernel generation using Mako templates

```
<%include file='idx_of.cu.mak' />

__global__ void
negdivconf(int nupts, int neles,
           ${dtype}* __restrict__ tdivtconf,
           const ${dtype}* __restrict__ rcpdjac,
           int ldt, int ldr)
{
    int eidx = blockIdx.x * blockDim.x + threadIdx.x;

    if (eidx < neles)
    {
        for (int uidx = 0; uidx < nupts; ++uidx)
        {
            ${dtype} s = -rcpdjac[IDX_OF(uidx, eidx, ldr)];

        % for i in range(nvars):
            tdivtconf[U_IDX_OF(uidx, eidx, ${i}, neles, ldt)] *= s;
        % endfor
        }
    }
}
```

PyFR

- Kernel generation using Mako templates

```
#ifndef _PYFR_IDX_OF
#define _PYFR_IDX_OF

#define IDX_OF(i, j, ldim) ((i)*(ldim) + (j))

#define U_IDX_OF(upt, ele, var, nele, ldim) \
    IDX_OF(upt, nele*var + ele, ldim)

#endif // _PYFR_IDX_OF
```

```
__global__ void
negdivconf(int nupts, int neles,
           double* __restrict__ tdivtconf,
           const double* __restrict__ rcpdjac,
           int ldt, int ldr)
{
    int eidx = blockIdx.x * blockDim.x + threadIdx.x;

    if (eidx < neles)
    {
        for (int uidx = 0; uidx < nupts; ++uidx)
        {
            double s = -rcpdjac[IDX_OF(uidx, eidx, ldr)];

            tdivtconf[U_IDX_OF(uidx, eidx, 0, neles, ldt)] *= s;
            tdivtconf[U_IDX_OF(uidx, eidx, 1, neles, ldt)] *= s;
            tdivtconf[U_IDX_OF(uidx, eidx, 2, neles, ldt)] *= s;
            tdivtconf[U_IDX_OF(uidx, eidx, 3, neles, ldt)] *= s;
            tdivtconf[U_IDX_OF(uidx, eidx, 4, neles, ldt)] *= s;
        }
    }
}
```

PyFR

- Queued compute and MPI kernel execution

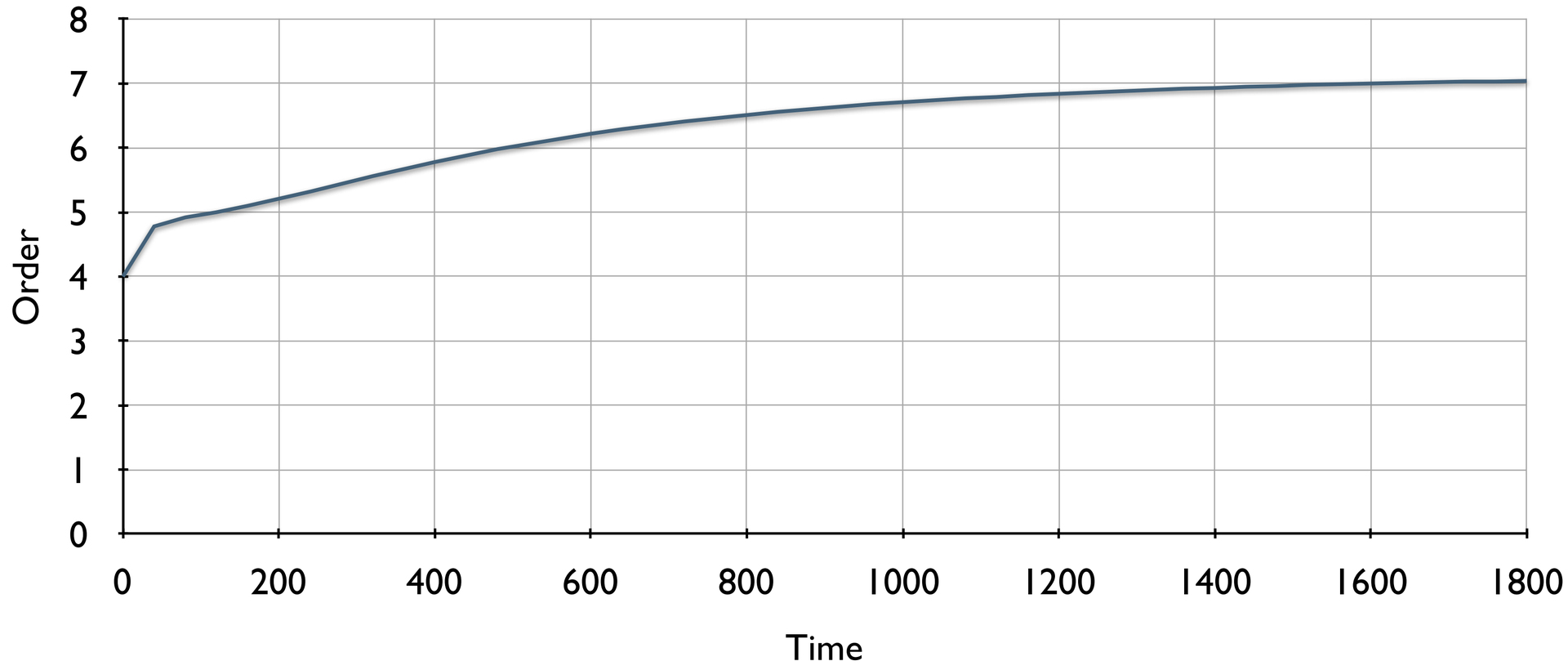
```
    def _get_negdivf(self):
        runall = self._backend.runall
        q1, q2 = self._queues

        q1 << self._disu_fpts_kerns()
        q1 << self._mpi_inters_scal_fpts0_pack_kerns()
        runall([q1])

        q1 << self._tdisf_upts_kerns()
        q1 << self._tdivtpcorf_upts_kerns()
        q1 << self._int_inters_rsolve_kerns()
        q1 << self._bc_inters_rsolve_kerns()

        q2 << self._mpi_inters_scal_fpts0_send_kerns()
        q2 << self._mpi_inters_scal_fpts0_recv_kerns()
        q2 << self._mpi_inters_scal_fpts0_unpack_kerns()

        runall([q1, q2])

        q1 << self._mpi_inters_rsolve_kerns()
        q1 << self._tdivtconf_upts_kerns()
        q1 << self._negdivconf_upts_kerns()
        runall([q1])
```

Results

- Some preliminary results on our 4 × K20 workstation

Results

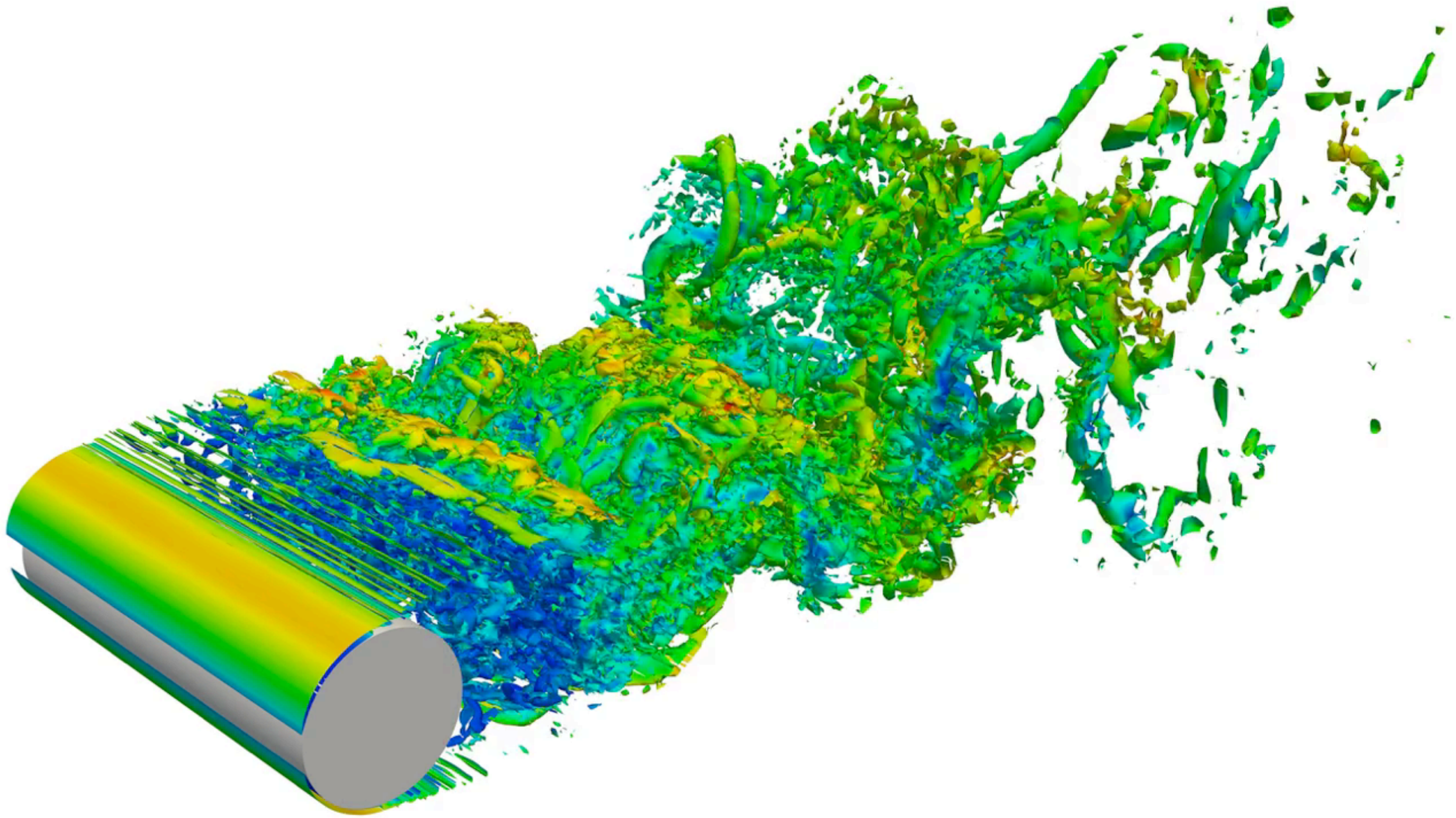
- Super $2k + 1$ accuracy for a 2D Euler vortex with $k = 3$ [4]



[4] P. E. Vincent, P. Castonguay, A. Jameson. Insights from von Neumann analysis of high-order flux reconstruction schemes. Journal of Computational Physics. 2011

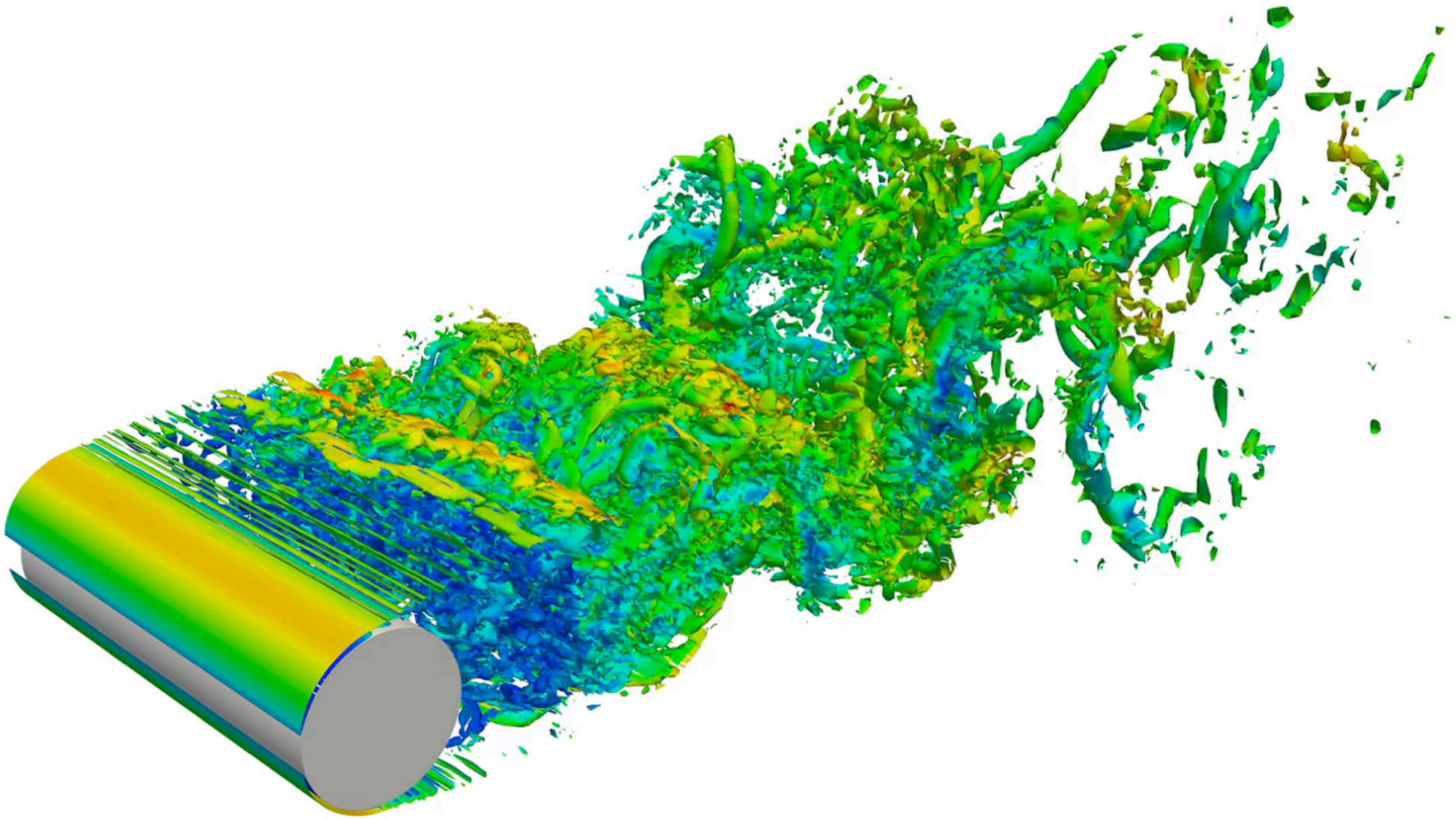
Results

- Flow over cylinder at Ma = 0.2 Re = 3900



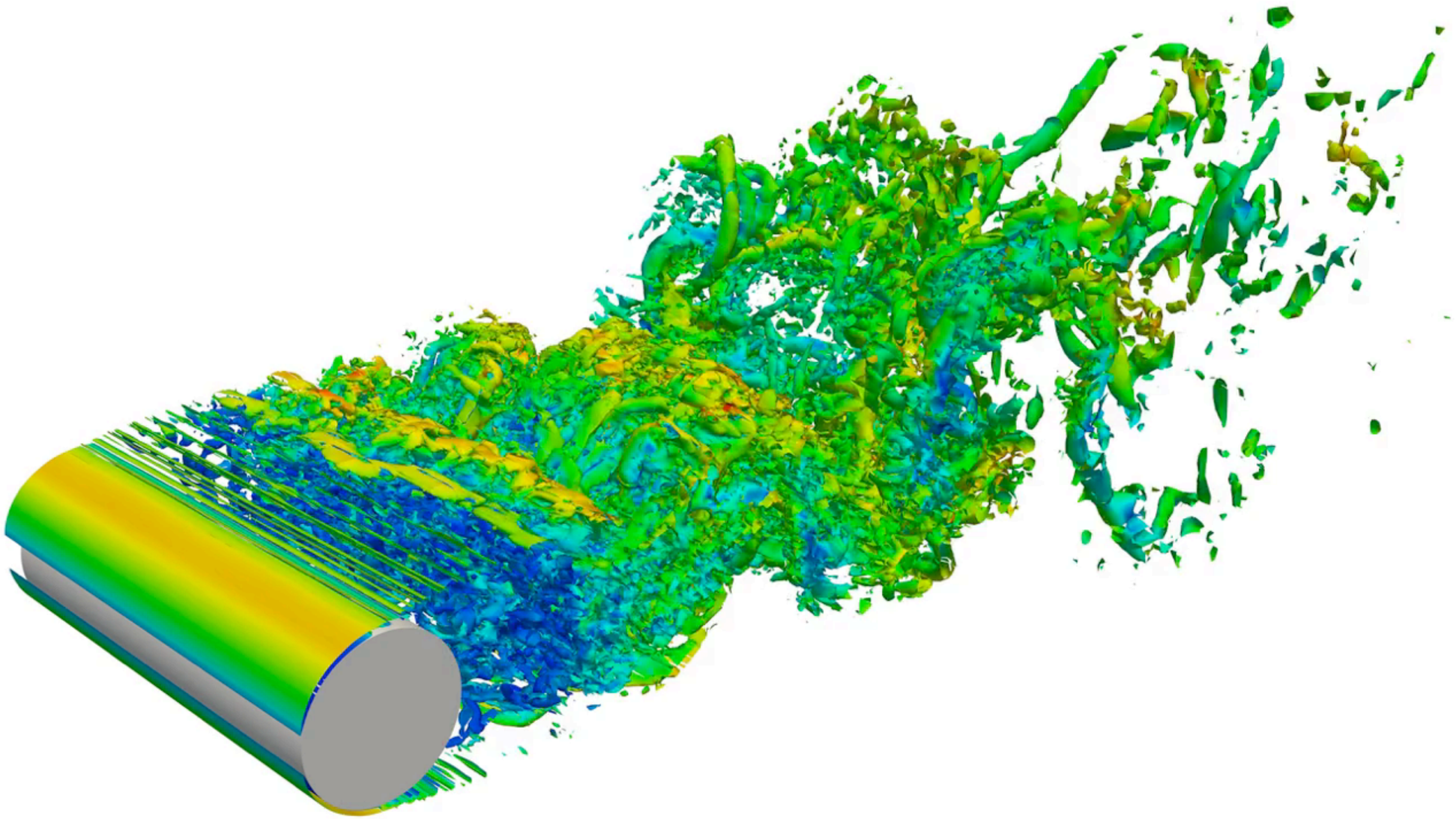
Results

- 5th order accurate with 2.9 x 10^7 DOFs



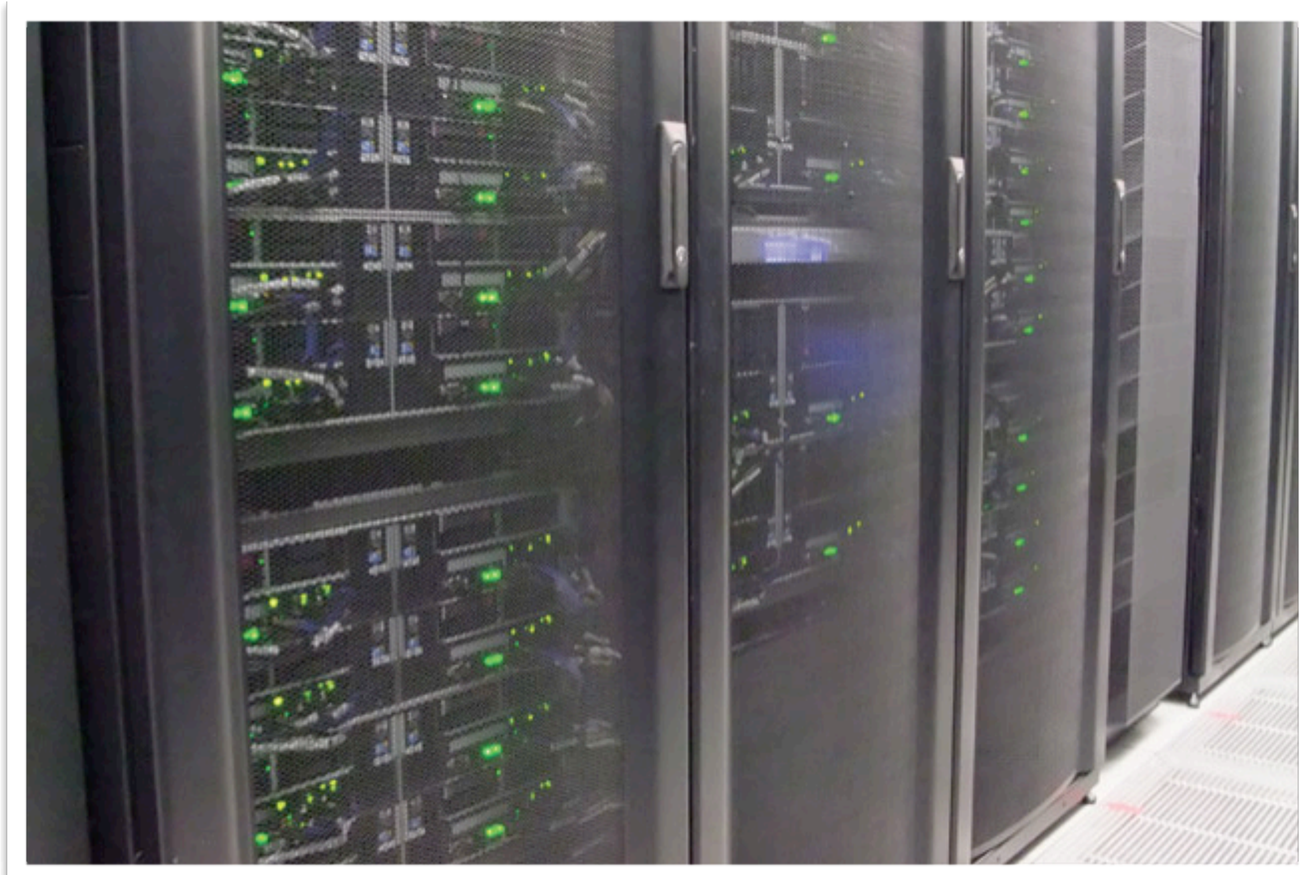
Results

- Double precision



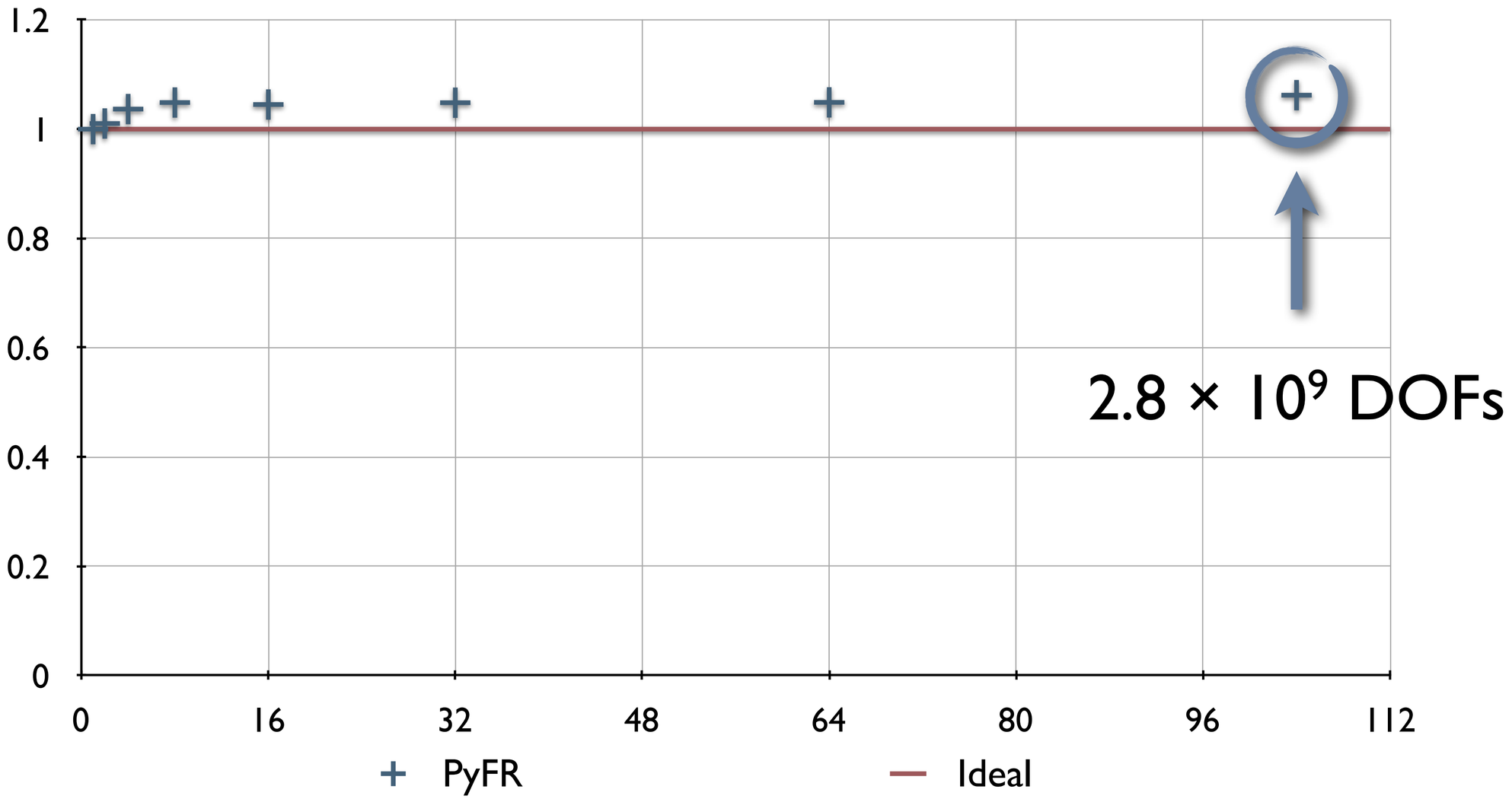
Results

- Scaling on Emerald, UK's largest multi-GPU cluster
- 372 × Nvidia M2090s each with 512 CUDA cores



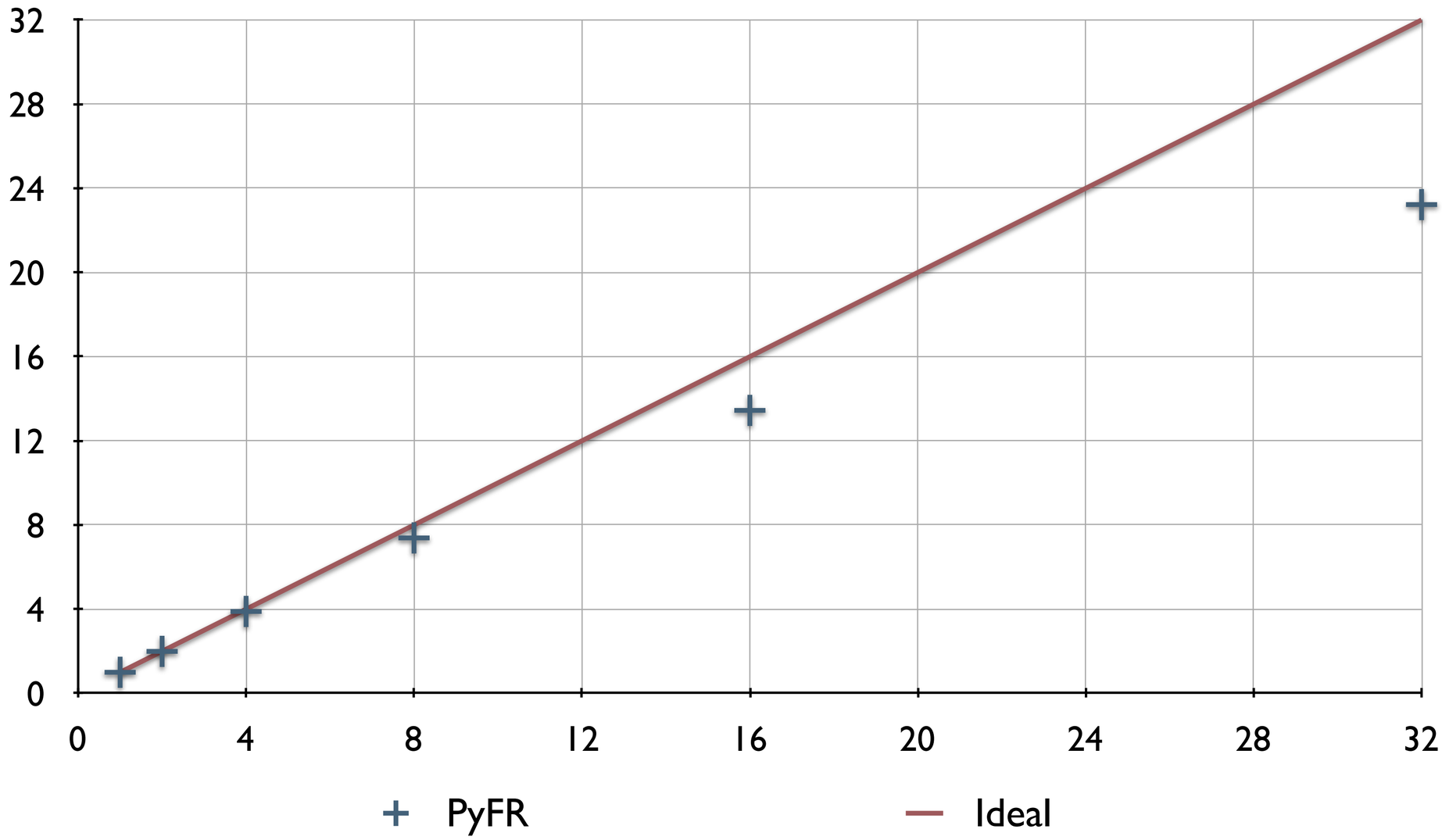
Results

- 3D Navier-Stokes weak scaling on Emerald



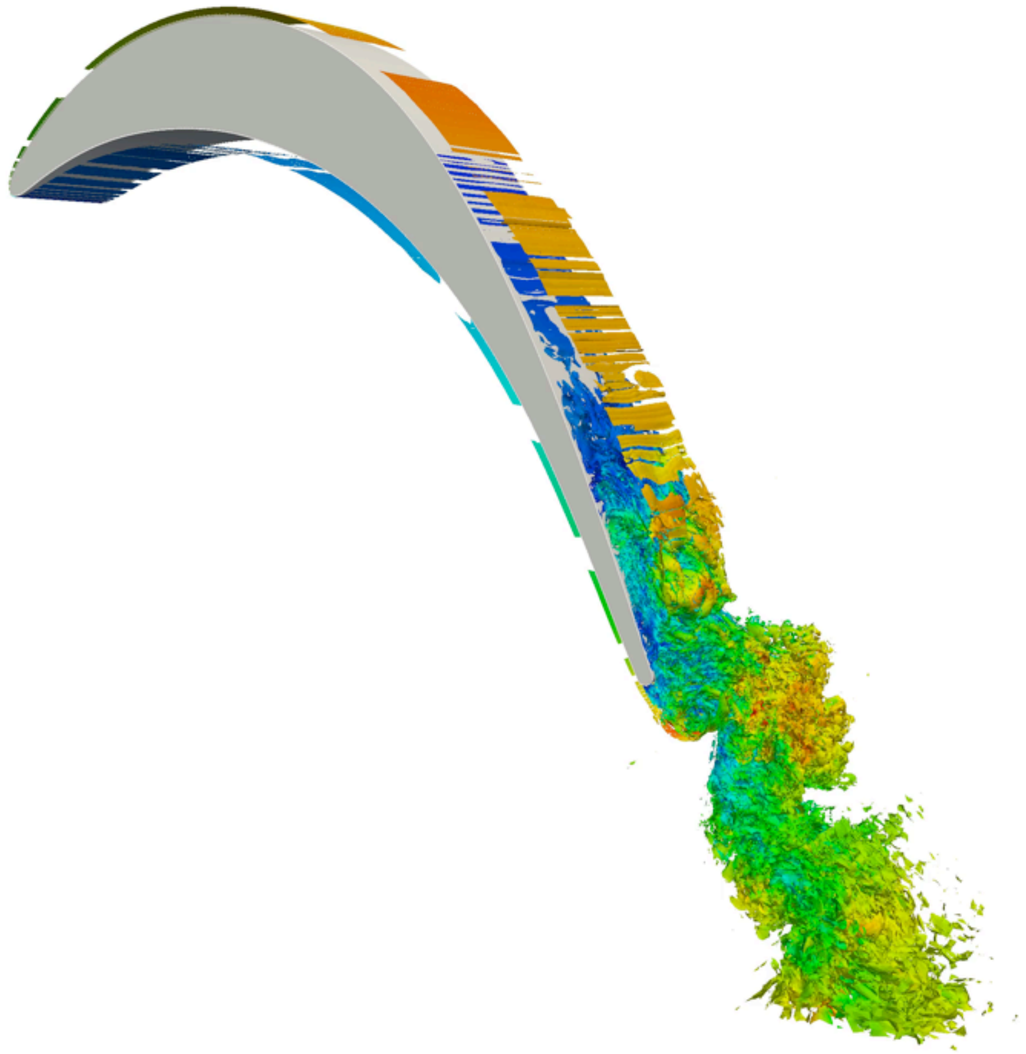
Results

- 3D Navier-Stokes strong scaling on Emerald



Results

- T106c Low Pressure Turbine Blade at Re = 80,000



Summary

- Interested in FR methods
- Interested in simple and efficient Python based implementations that target multiple hardware platforms
- Interested in application to solve real world industrial flow problems

Summary

- Funded and supported by

EPSRC
Engineering and Physical Sciences Research Council

ROYAL COMMISSION EXHIBITION OF 1851
DIEU ET MON DROIT

- Any questions?
- E-mail: freddie.witherden08@imperial.ac.uk