Imperial College London

Extruded meshes for high aspect ratio simulations in Firedrake and PyOP2

Gheoghe-Teodor (Doru) Bercea[a], Andrew TT McRae[b], **Lawrence Mitchell**[c], David A Ham, Paul HJ Kelly

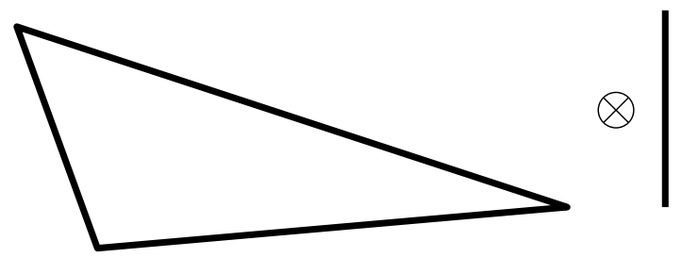
(a) gheorghe-teodor.bercea08@imperial.ac.uk
(b) a.mcrae12@imperial.ac.uk
(c) lawrence.mitchell@imperial.ac.uk

- (Very) high aspect ratio domains
- Want vertically column-aligned meshes
  - e.g. important that vertical gradients don't leak into horizontal (and vice-versa)
- Possible to achieve with fully unstructured meshes
- Often don't *need* unstructured benefits in short "vertical" direction

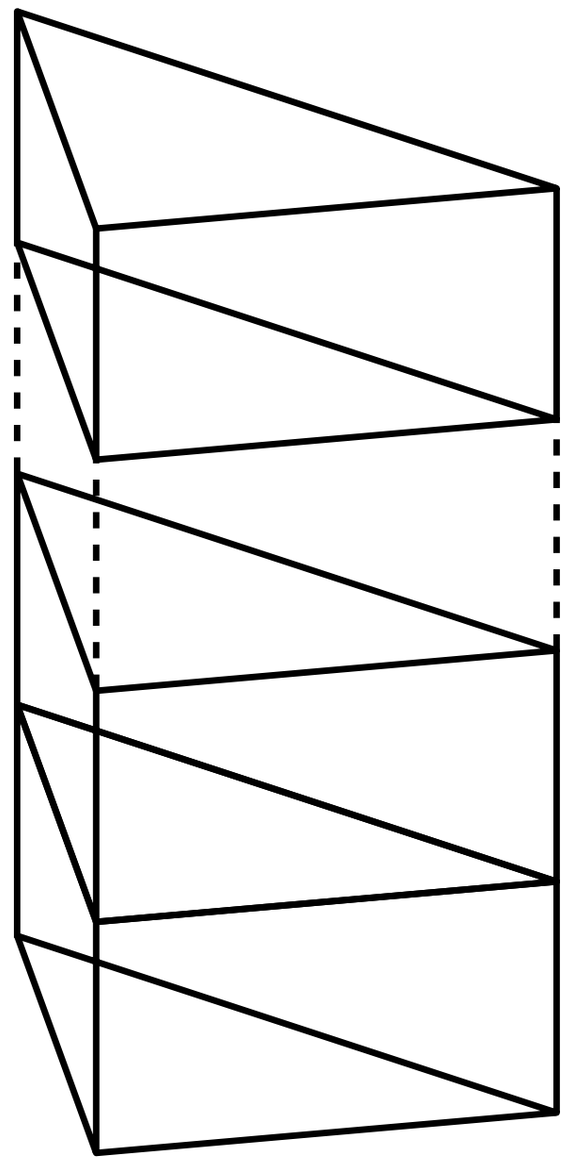
Exploit column structure

- *Extrude* an unstructured base mesh
  - New cells gain a dimension (interval→quad, triangle→prism)
- Arrange for extruded direction to be innermost iteration index: direct addressing (see, e.g., MacDonald et al. Int. J. HPC App. **25**(4):392 (2010))
  - Exploit this in execution

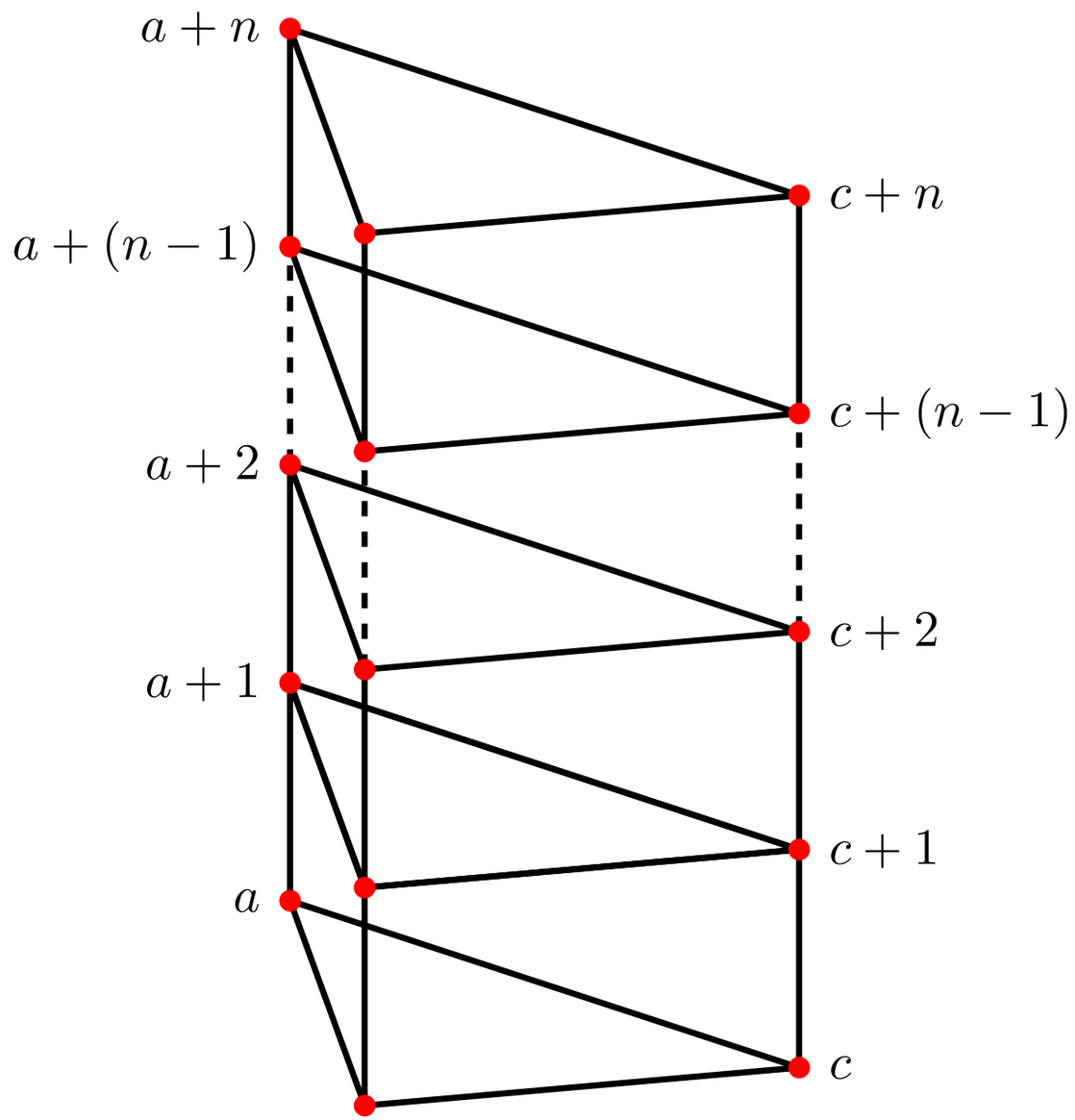
Direct addressing in vertical



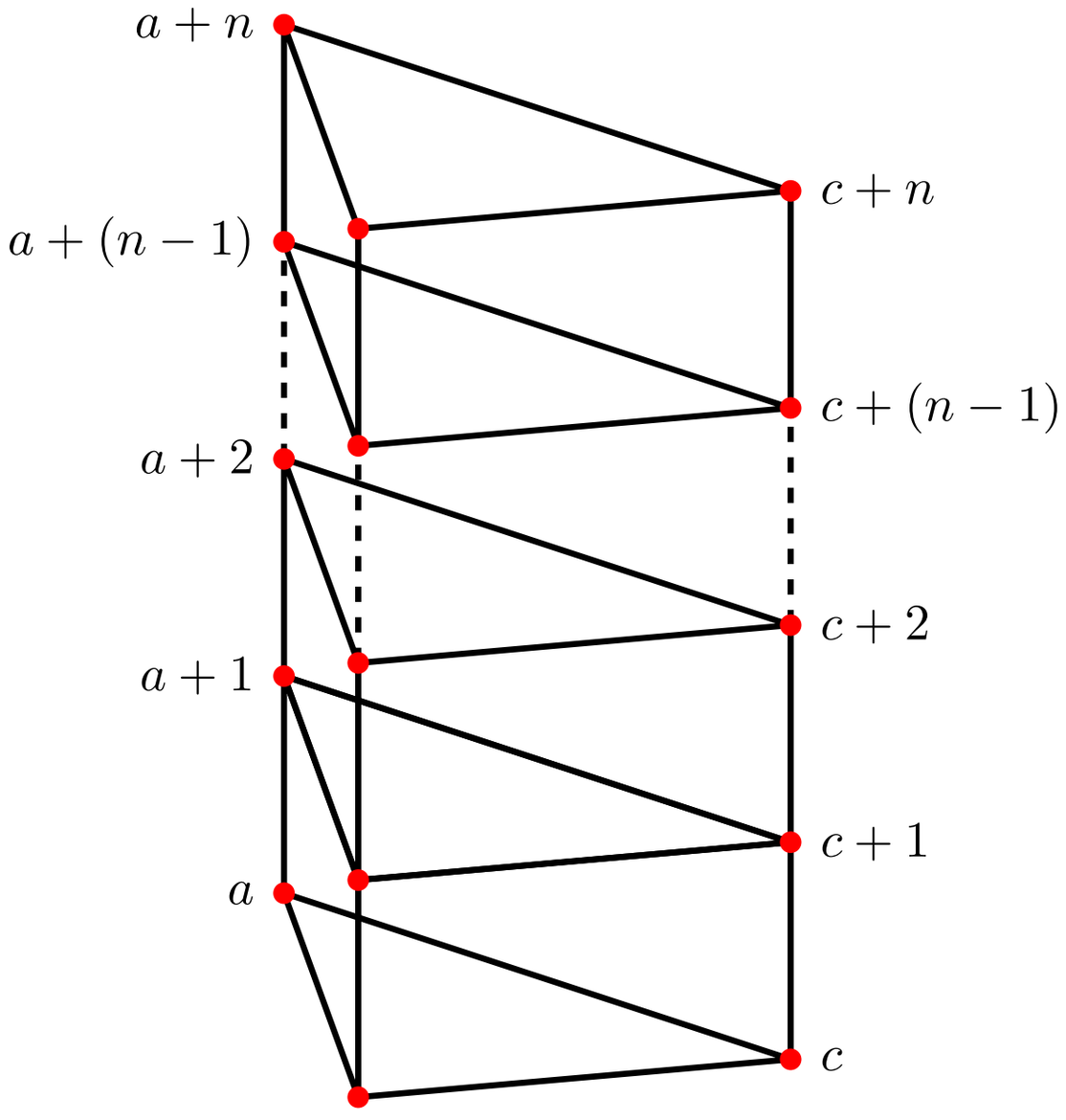
Direct addressing in vertical



Direct addressing in vertical



Direct addressing in vertical



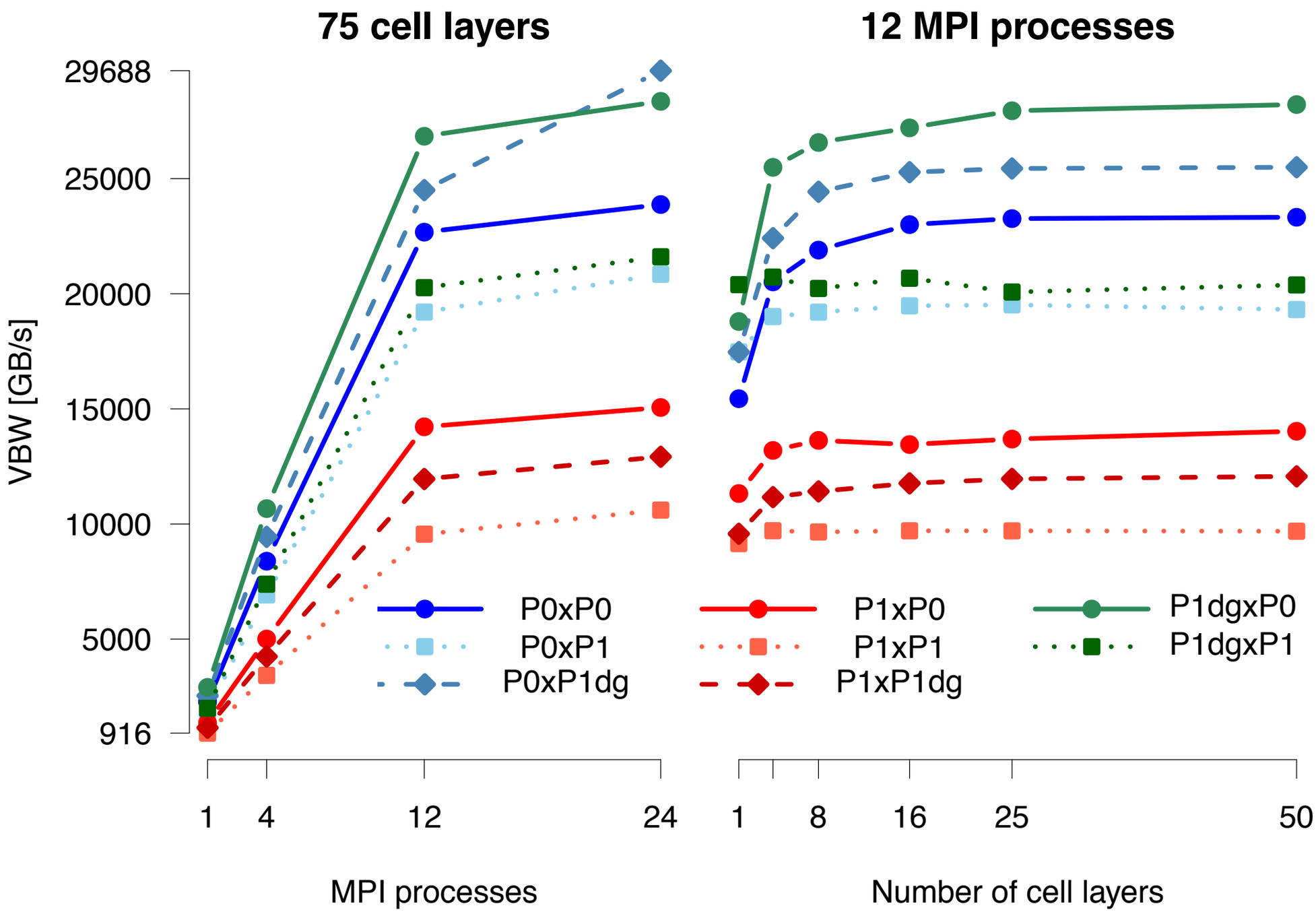
- Connectivity specified by *base cell* indirection and fixed *offset* for each dof in extruded cells
- Walking up column is prefetch-friendly stride-n access
- Also decreases memory footprint of indirections

Does it work?

- What to measure?
  - Bandwidth, flops? Depends what regime we're in.
- For bandwidth bound problems, we measure *valuable bandwidth*
  - assume there is a *perfect* ordering that allows us to stream data from main memory, use it and then evict, always using full cache lines
  - Doesn't count memory footprint of indirections
  - *Easy* to measure: data volume / execution time
  - Lower bound on actual data movement, can compare to STREAM

(Very) simple (bandwidth bound) rhs assembly: $\int v \, dx$

Base mesh: ~800000 triangles (Hilbert curve numbering)



12 core Intel Xeon E5-2620 @ 2.0 GHz
Intel 14.0.1 (-O3 -xAVX), Intel MPI 3.1.038
STREAM Triad 42 GB/s

What about FLOP-limited cases?

- We use (modified) versions of the FEniCS toolchain to generate assembly kernels

- These are further optimised by an FE-aware loop-nest compiler, Luporini et al. `arxiv:1407.0904 [cs.MS]`

    - Performs loop-invariant code motion, vector-alignment and padding, loop unrolling/permutation and manual vectorisation

- For P2 Helmholtz operator, we sustain ~20GFlop/s on a single core

    - Guaranteed not to exceed 30.4 GFlop/s, with simultaneous issue of 1 AVX mul and 1 AVX add per cycle (not Haswell, so no FMA)

Building elements

- Performance no good if we can't express variational problems

- Drive Firedrake using (mostly) DOLFIN-compatible Python interface

- Express variational problems on extruded meshes using *extensions* to UFL

```
m = UnitIntervalMesh(5)

mesh = ExtrudedMesh(m, layers=5)

U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W0_elt = OuterProductElement(U0, V0)
W1_a = HDiv(OuterProductElement(U1, V0))
W1_b = HDiv(OuterProductElement(U0, V1))

W1_elt = W1_a + W1_b

W0 = FunctionSpace(mesh, W0_elt)
W1 = FunctionSpace(mesh, W1_elt)

W = W0*W1
sigma, u = TrialFunctions(W)
tau, v = TestFunctions(W)

L = assemble((sigma*tau - inner(rot(tau), u) + inner(rot(sigma), v) +
              div(u)*div(v))*dx)
```

```
m = UnitIntervalMesh(5)

mesh = ExtrudedMesh(m, layers=5)

U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W0_elt = OuterProductElement(U0, V0)
W1_a = HDiv(OuterProductElement(U1, V0))
W1_b = HDiv(OuterProductElement(U0, V1))

W1_elt = W1_a + W1_b

W0 = FunctionSpace(mesh, W0_elt)
W1 = FunctionSpace(mesh, W1_elt)

W = W0*W1
sigma, u = TrialFunctions(W)
tau, v = TestFunctions(W)

L = assemble((sigma*tau - inner(rot(tau), u) + inner(rot(sigma), v) +
              div(u)*div(v))*dx)
```

Product complexes

- We support elements that are a tensor product of base mesh basis functions, and interval basis functions
- Motivating application: mimetic FE for numerical weather prediction, requires a discrete de Rham complex
- Construct from product of base mesh and interval complexes

Product complexes

- We support elements that are a tensor product of base mesh basis functions, and interval basis functions
- Motivating application: mimetic FE for numerical weather prediction, requires a discrete de Rham complex
- Construct from product of base mesh and interval complexes

$$\mathbb{U}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_1 \xrightarrow{\mathrm{d}} \mathbb{U}_2$$

Product complexes

- We support elements that are a tensor product of base mesh basis functions, and interval basis functions
- Motivating application: mimetic FE for numerical weather prediction, requires a discrete de Rham complex
- Construct from product of base mesh and interval complexes

$$\mathbb{U}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_1 \xrightarrow{\mathrm{d}} \mathbb{U}_2$$

$$\mathbb{V}_0 \xrightarrow{\mathrm{d}} \mathbb{V}_1$$

Product complexes

- We support elements that are a tensor product of base mesh basis functions, and interval basis functions
- Motivating application: mimetic FE for numerical weather prediction, requires a discrete de Rham complex
- Construct from product of base mesh and interval complexes

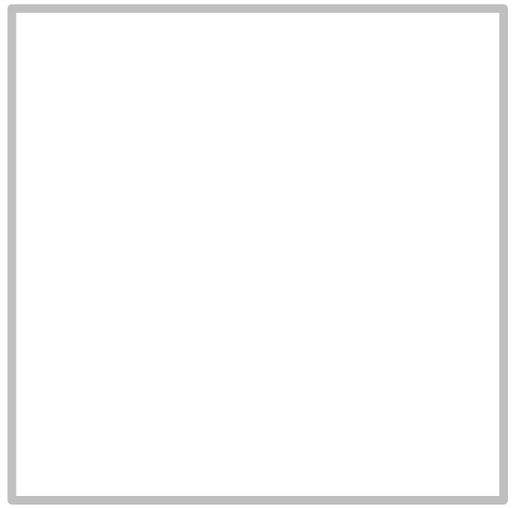
$$\mathbb{U}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_1 \xrightarrow{\mathrm{d}} \mathbb{U}_2$$

$$\mathbb{V}_0 \xrightarrow{\mathrm{d}} \mathbb{V}_1$$

$$\mathbb{U}_0 \otimes \mathbb{V}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_0 \otimes \mathbb{V}_1 \oplus \mathbb{U}_1 \otimes \mathbb{V}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_1 \otimes \mathbb{V}_1 \oplus \mathbb{U}_2 \otimes \mathbb{V}_0 \xrightarrow{\mathrm{d}} \mathbb{U}_2 \otimes \mathbb{V}_1$$

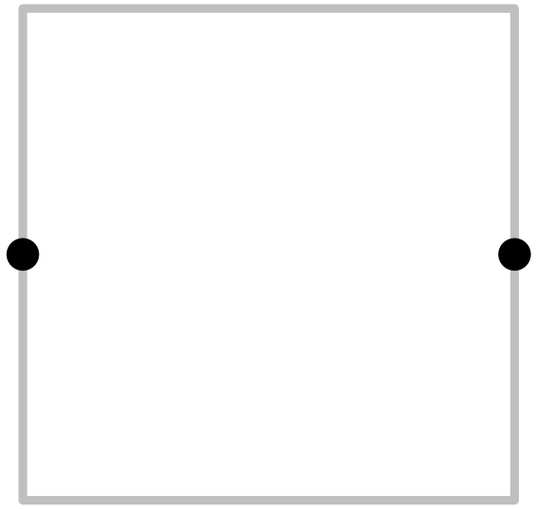
Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```



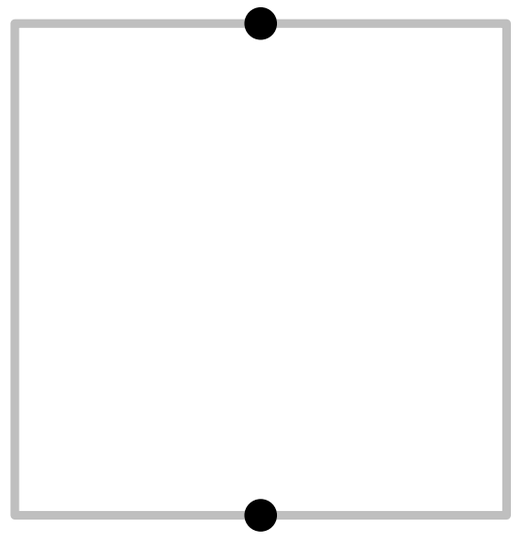
Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```



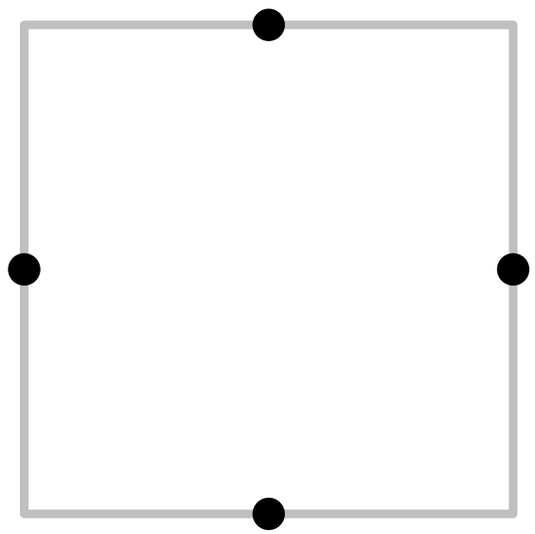
Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```



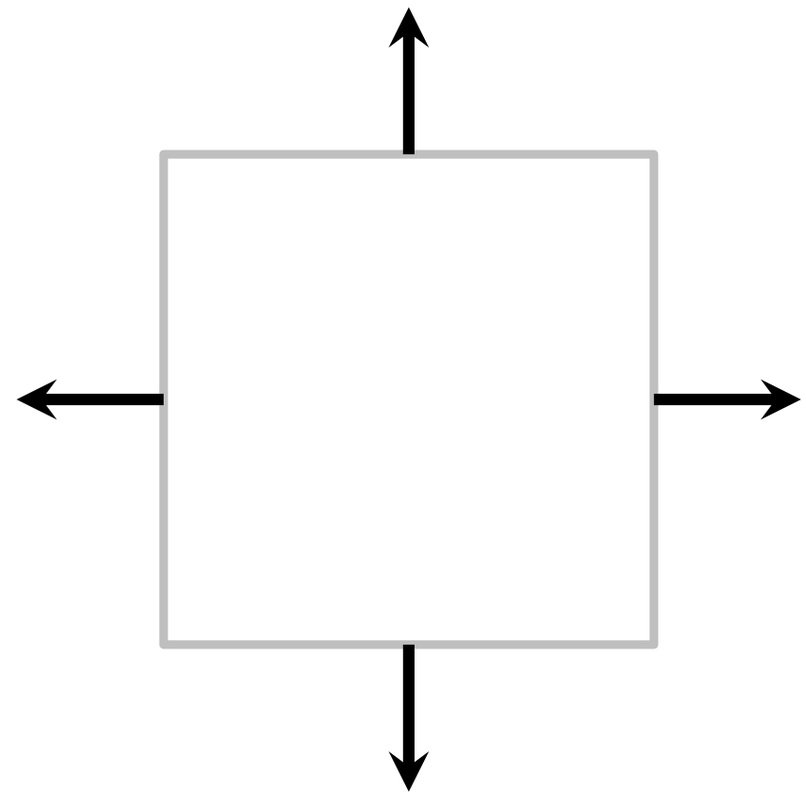
Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```



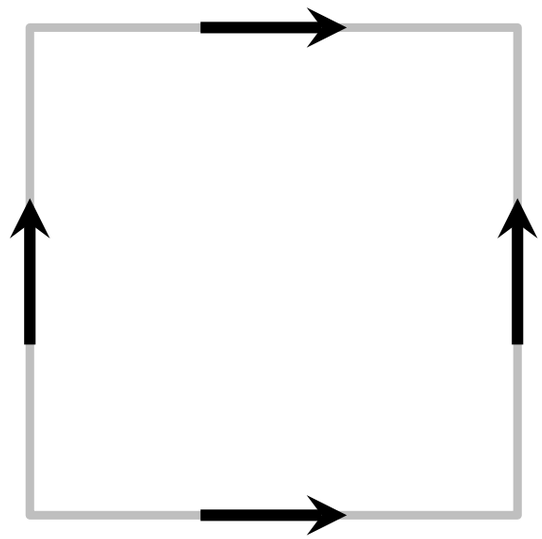
Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```



Example: lowest order RT and Nedelec 1st kind

```
U0 = FiniteElement("CG", interval, 1)
U1 = FiniteElement("DG", interval, 0)
V0 = FiniteElement("CG", interval, 1)
V1 = FiniteElement("DG", interval, 0)

W1_a = OuterProductElement(U0, V1)

W1_b = OuterProductElement(U1, V0)

W1 = W1_a + W1_b

RT1 = HDiv(W1)

N1 = HCurl(W1)
```

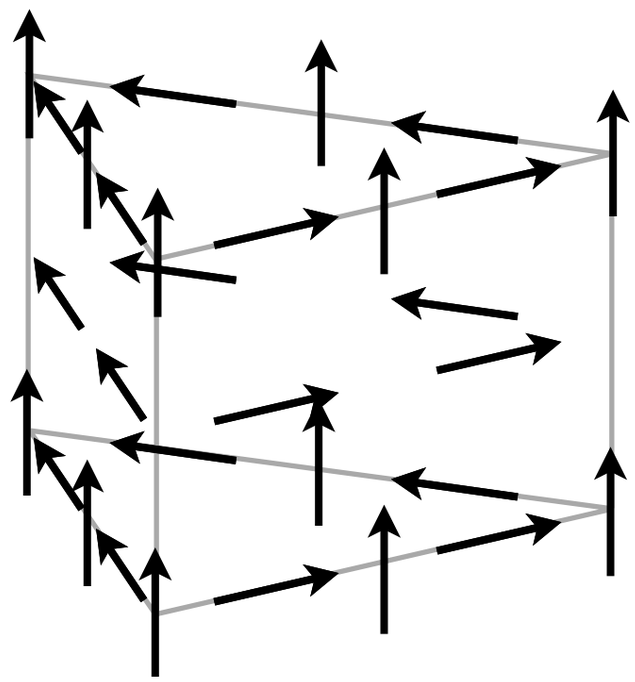


Lowest order Nedelec 2nd kind on prism

```
N2_1 = FiniteElement("N2curl", triangle, 1)
CG_2 = FiniteElement("CG", interval, 2)

Ned_horiz = HCurl(OuterProductElement(N2_1, CG_2))

P2tri = FiniteElement("CG", triangle, 2)
P1dg = FiniteElement("DG", interval, 1)
Ned_vert = HCurl(OuterProductElement(P2tri, P1dg))

Ned_wedge = Ned_horiz + Ned_vert
```

Lowest order Nedelec 2nd kind on prism

```
N2_1 = FiniteElement("N2curl", triangle, 1)
CG_2 = FiniteElement("CG", interval, 2)

Ned_horiz = HCurl(OuterProductElement(N2_1, CG_2))

P2tri = FiniteElement("CG", triangle, 2)
P1dg = FiniteElement("DG", interval, 1)
Ned_vert = HCurl(OuterProductElement(P2tri, P1dg))

Ned_wedge = Ned_horiz + Ned_vert
```



Future work

- MF Multigrid for mixed Helmholtz problems on extruded meshes, with Eike Müller (Bath)

- Exploit structure inside tensor product kernels (currently we just splat everything out)

- 3d numerics and performance characterisation for atmosphere, with Colin Cotter (Imperial)

Questions?

www.firedrakeproject.org
firedrake@imperial.ac.uk

Funding:
Grantham Institute for climate change
NERC grants NE/K008951/1, NE/K006789/1, NE/G523512/1
EPSRC grants EP/L000407/1, EP/K008730/1, EP/I00677X/1