```
# Now with Devito we will turn ‘p‘ into ‘TimeFunction‘
# object to make all the buffer switching implicit
p = TimeFunction(name=’p’, grid=grid, space_order=2)

# Initialize the source term ‘b‘
b = Function(name=’b’, grid=grid)
b.data[:] = 0.
b.data[int(nx / 4), int(ny / 4)] = 100
b.data[int(3 * nx / 4), int(3 * ny / 4)] = -100

# Create Laplace equation base on ‘p‘
eq = Eq(p.laplace, b)
# Let SymPy solve for the central stencil point
stencil = solve(eq, p)
# Let our stencil populate the buffer ‘p.forward‘
eq_stencil = Eq(p.forward, stencil)

# Create boundary condition expressions
# Note that we now add an explicit "t + 1"
# for the time dimension.
bc = [Eq(p[t + 1, x, 0], 0.)]
bc += [Eq(p[t + 1, x, ny-1], 0.)]
bc += [Eq(p[t + 1, 0, y], 0.)]
bc += [Eq(p[t + 1, nx-1, y], 0.)]

# We can even switch performance logging back on,
# since we only require a single kernel invocation.
configuration[’log-level’] = ’INFO’

# Create and execute the operator for nt iterations
op = Operator([eq_stencil] + bc)
op(time=nt)
```