```
# Define our velocity fields and initialize with hat
    function
u = TimeFunction(name='u', grid=grid, space_order=2)
v = TimeFunction(name='v', grid=grid, space_order=2)
init_hat(field=u.data[0], dx=dx, dy=dy, value=2.)
init_hat(field=v.data[0], dx=dx, dy=dy, value=2.)

# Write down the equations with explicit backward
    differences
a = Constant(name='a')
u_dx = first_derivative(u, dim=x, side=left, order=1)
u_dy = first_derivative(u, dim=y, side=left, order=1)
v_dx = first_derivative(v, dim=x, side=left, order=1)
v_dy = first_derivative(v, dim=y, side=left, order=1)
eq_u = Eq(u.dt + u*u_dx + v*u_dy, a*u.laplace,
    subdomain=grid.interior)
eq_v = Eq(v.dt + u*v_dx + v*v_dy, a*v.laplace,
    subdomain=grid.interior)

# Let SymPy rearrange our stencils to form the update
    expressions
stencil_u = solve(eq_u, u.forward)
stencil_v = solve(eq_v, v.forward)
update_u = Eq(u.forward, stencil_u)
update_v = Eq(v.forward, stencil_v)

# Create Dirichlet BC expressions using the low-level
    API
bc_u = [Eq(u[t+1, 0, y], 1.)] # left
bc_u += [Eq(u[t+1, nx-1, y], 1.)] # right
bc_u += [Eq(u[t+1, x, ny-1], 1.)] # top
bc_u += [Eq(u[t+1, x, 0], 1.)] # bottom
bc_v = [Eq(v[t+1, 0, y], 1.)] # left
bc_v += [Eq(v[t+1, nx-1, y], 1.)] # right
bc_v += [Eq(v[t+1, x, ny-1], 1.)] # top
bc_v += [Eq(v[t+1, x, 0], 1.)] # bottom

# Create the operator
op = Operator([update_u, update_v] + bc_u + bc_v)
```