```
# Element tensors defining the local 3-by-3 block system
_A = Tensor(a)
_F = Tensor(L)

# Extracting blocks for Slate expression of the reduced system
A = _A.blocks
F = _F.blocks
Sexp = A[2, 2] - A[2, :2] * A[:2, :2].inv * A[:2, 2]  # Slate expression for S^K
Eexp = F[2] - A[2, :2] * A[:2, :2].inv * F[:2]        # Slate expression for E^K
S = assemble(Sexp, bcs=[...])                         # Assemble S
E = assemble(Eexp)                                    # Assemble E
lambda_h = Function(M)                                # Function to store the result: Λ

# Solve for the Lagrange multipliers: Λ
solve(S, lambda_h, E, solver_parameters={"ksp_type": "preonly", "pc_type": "lu"})
p_h = Function(V)                                     # Function to store the result: P
u_h = Function(U)                                     # Function to store the result: U^d
Lambda = AssembledVector(lambda_h)                    # Local coefficient vector: Λ^K
P = AssembledVector(p_h)                              # Local coefficient vector: P^K

# Intermediate expressions
Sd = A[1, 1] - A[1, 0] * A[0, 0].inv * A[0, 1]
Sl = A[1, 2] - A[1, 0] * A[0, 0].inv * A[0, 2]

# Slate expressions for local recovery
p_sys = Sd.solve(F[1] - A[1, 0] * A[0, 0].inv * F[0] - Sl * Lambda,
                 decomposition="PartialPivLu")
u_sys = A[0, 0].solve(F[0] - A[0, 1] * P - A[0, 2] * Lambda,
                      decomposition="PartialPivLu")
assemble(p_sys, p_h)                                  # Solve for P
assemble(u_sys, u_h)                                  # Solve for U^d
```