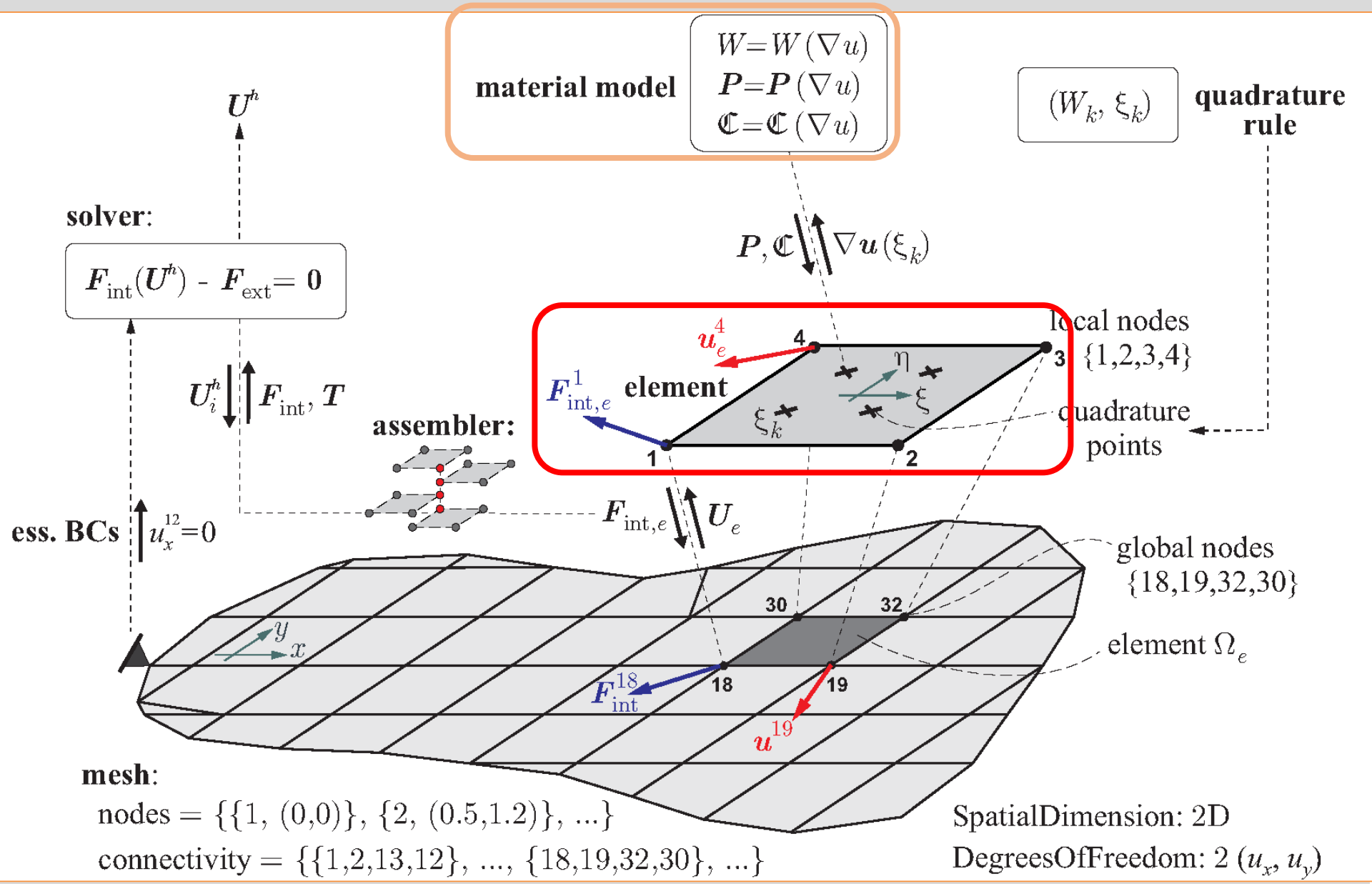
Finite Element Method: Code Overview

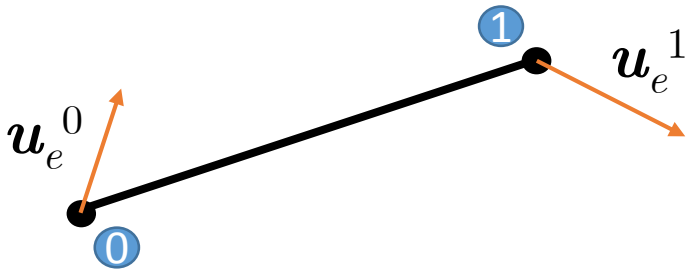
Ae214a: Computational Solid Mechanics

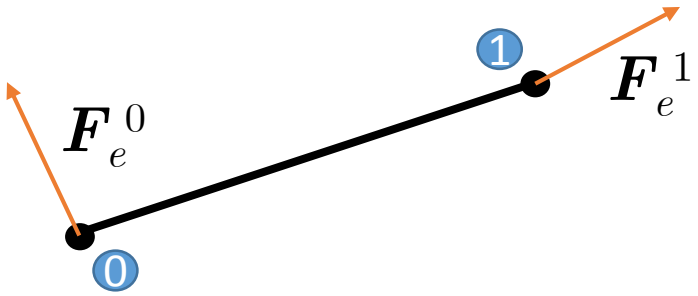


Finite Element Method: Bar Elements

Ae214a: Computational Solid Mechanics

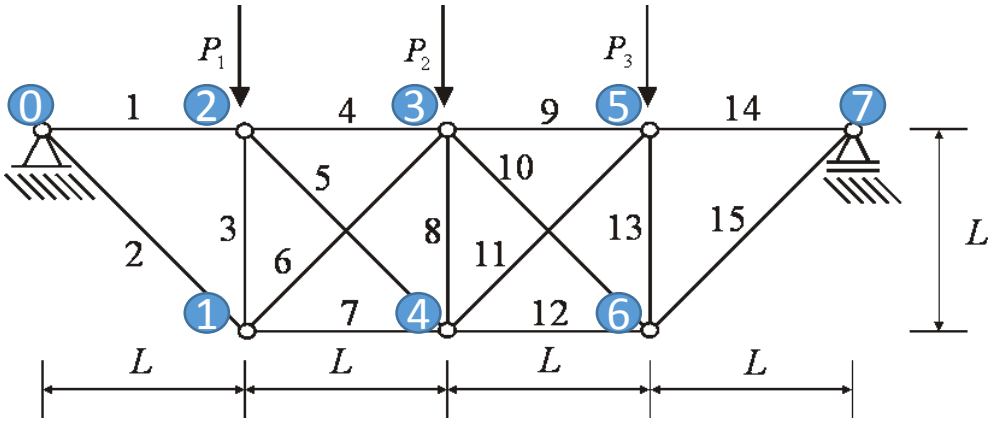
example: two-node (linear) bar elements

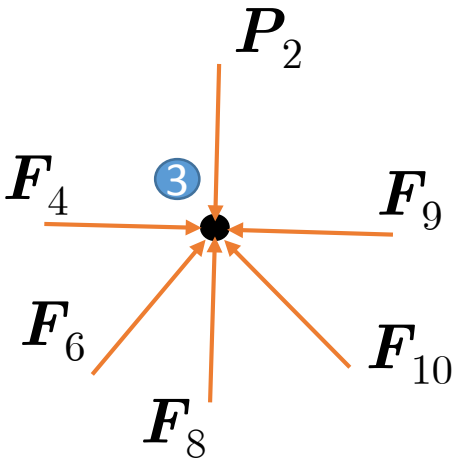




assembly operations: e.g., for the internal/external force vector

$$F^a_{\text{int},i} = \int_\Omega \sigma_{ij} N^a_{,j} = \sum_e \int_{\Omega_e} \sigma_{ij} N^a_{e,j} \qquad K^{ab}_{ik} = \sum_e \left(K^{ab}_{ik}\right)_e$$





Finite Element Method: Assembly Operations

Ae214a: Computational Solid Mechanics

- **assembly of the stiffness matrix**: e.g. element 5 **in 2D**

$$\boldsymbol{K}_5 = \begin{pmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{pmatrix}$$

nodal IDs of element 5: $(i, j) = (2, 4)$

element matrix is added to Rows and collums:

$$(2i, 2i+1, 2j, 2j+1)$$

$$\boldsymbol{K}_{\text{global}} =$$

	0	1	2	3	4		5	6	7	8		9	...
0	·	·	·	·	·		·	·	·	·		·	·
1	·	·	·	·	·		·	·	·	·		·	·
2	·	·	·	·	·		·	·	·	·		·	·
3	·	·	·	·	·		·	·	·	·		·	·
4	·	·	·	·	K_{11}		K_{12}	·	·	K_{13}		K_{14}	·
5	·	·	·	·	K_{21}		K_{22}	·	·	K_{23}		K_{24}	·
6	·	·	·	·	·		·	·	·	·		·	·
7	·	·	·	·	·		·	·	·	·		·	·
8	·	·	·	·	K_{31}		K_{32}	·	·	K_{33}		K_{34}	·
9	·	·	·	·	K_{41}		K_{42}	·	·	K_{43}		K_{44}	·
...	·	·	·	·	·		·	·	·	·		·	·

forces are assembled analogously into $\boldsymbol{F}_{\text{global}}$

Finite Element Method: Essential BCs

Ae214a: Computational Solid Mechanics

- **application of essential boundary conditions**: e.g. $u_x^1 = \delta$

$$
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ \dots
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \dots
\end{array} \\
\begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}
\end{array}
\begin{pmatrix}
u_x^0 \\ u_y^0 \\ u_x^1 \\ u_y^1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot
\end{pmatrix}
=
\begin{pmatrix}
F_x^0 \\ F_y^0 \\ \delta \\ F_y^1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot
\end{pmatrix}
$$

- for each boundary condition, we need: *nodeId* (e.g. 1), *coordinateId* (e.g. x), constraint (δ)

Finite Element Method: Mesh and Node Data Structures

Ae214a: Computational Solid Mechanics

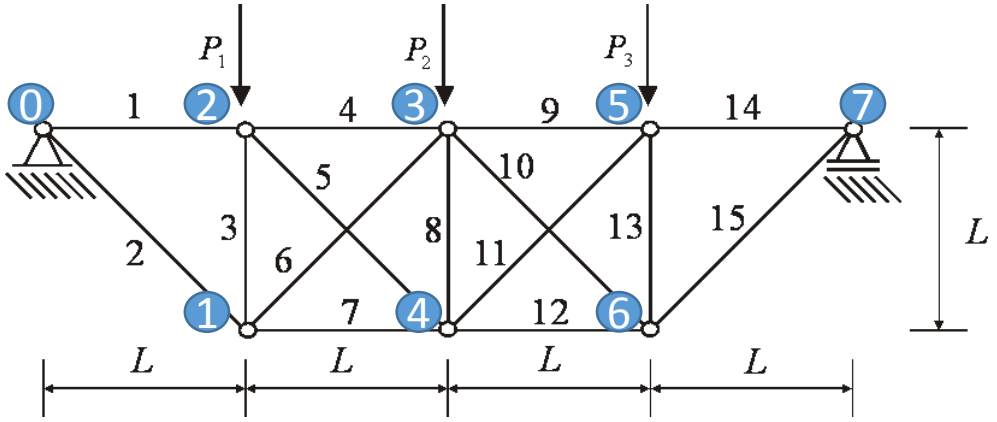
Let's translate this into C++ data structures and methods:

- **the geometric model**:
  - nodal coordinates:

$$\{(0,0),(L,-L),(L,0),\ldots\}$$

  - element connectivities

$$\{(0,2),(0,1),(1,2),\ldots\}$$



Note: In C++ code, everything should be numbered with **0-indexing**

- **c++ mesh datastructure**:

```
SingleElementMesh<NodalDimensions,NodesPerElement> mesh;
```

with

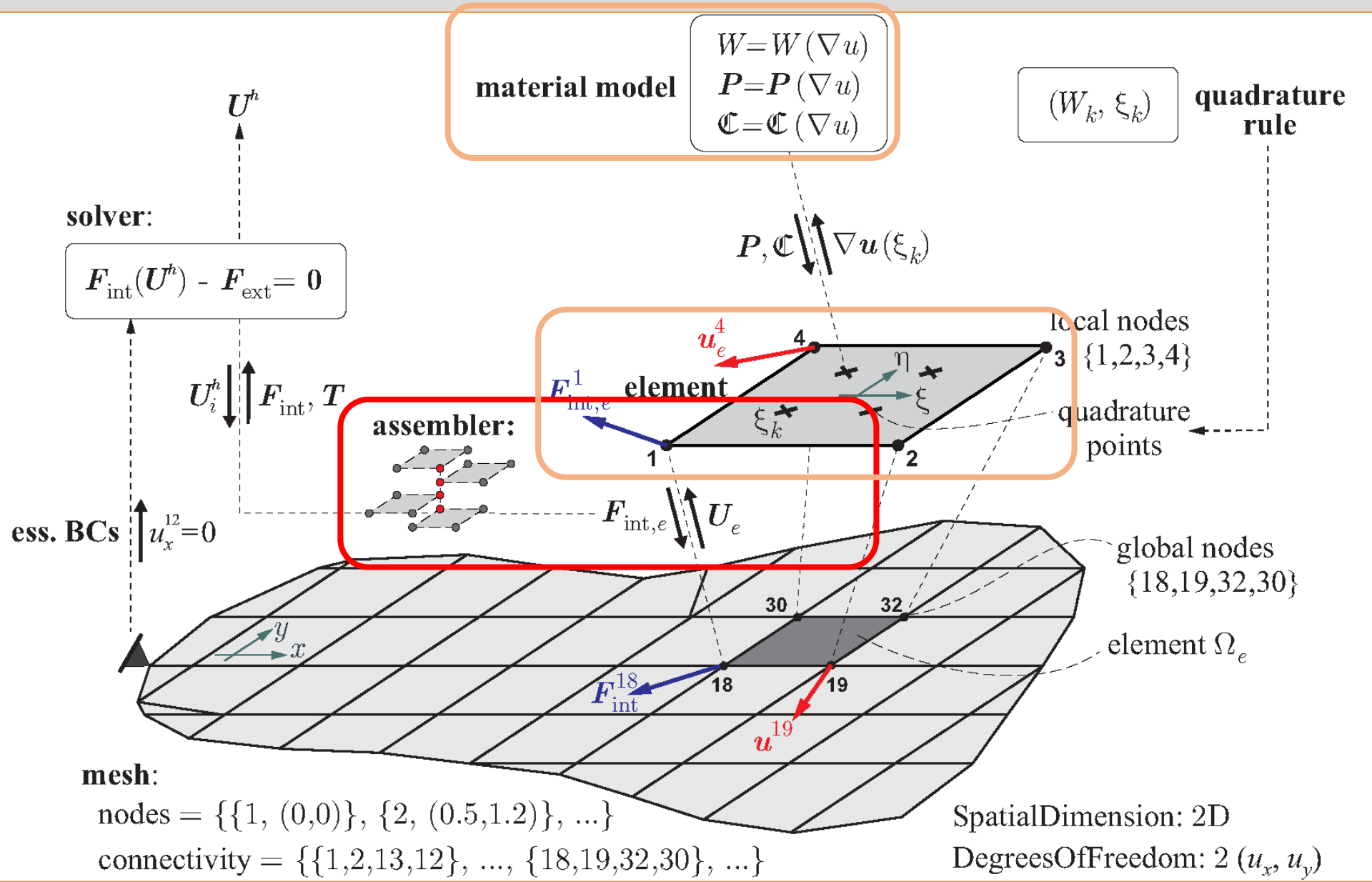
```
vector<NodeWithID>  mesh._nodes
vector<array<int, NodesPerElement> >  mesh._connectivity
```

- **storing nodal IDs and position in one object**: `NodeWithID`

```
NodeWithID._id
NodeWithID._position
```

Finite Element Method: Overall Code Structure

Ae214a: Computational Solid Mechanics



Finite Element Method: Assembler

Ae214a: Computational Solid Mechanics

- **Assembler class:** assembly of element quantities, e.g.,

```
double Assembler::assembleEnergy(const vector<Vector> & displacements)
```

*This vector contains the displacements for **every** node!*

- steps to do:
  - visit each element in `vector<Element>` and let it compute its energy:

```
double Element::computeEnergy(const array<Vector, 2> & displacements)
```

*This vector contains the displacements for each **element node** only!*

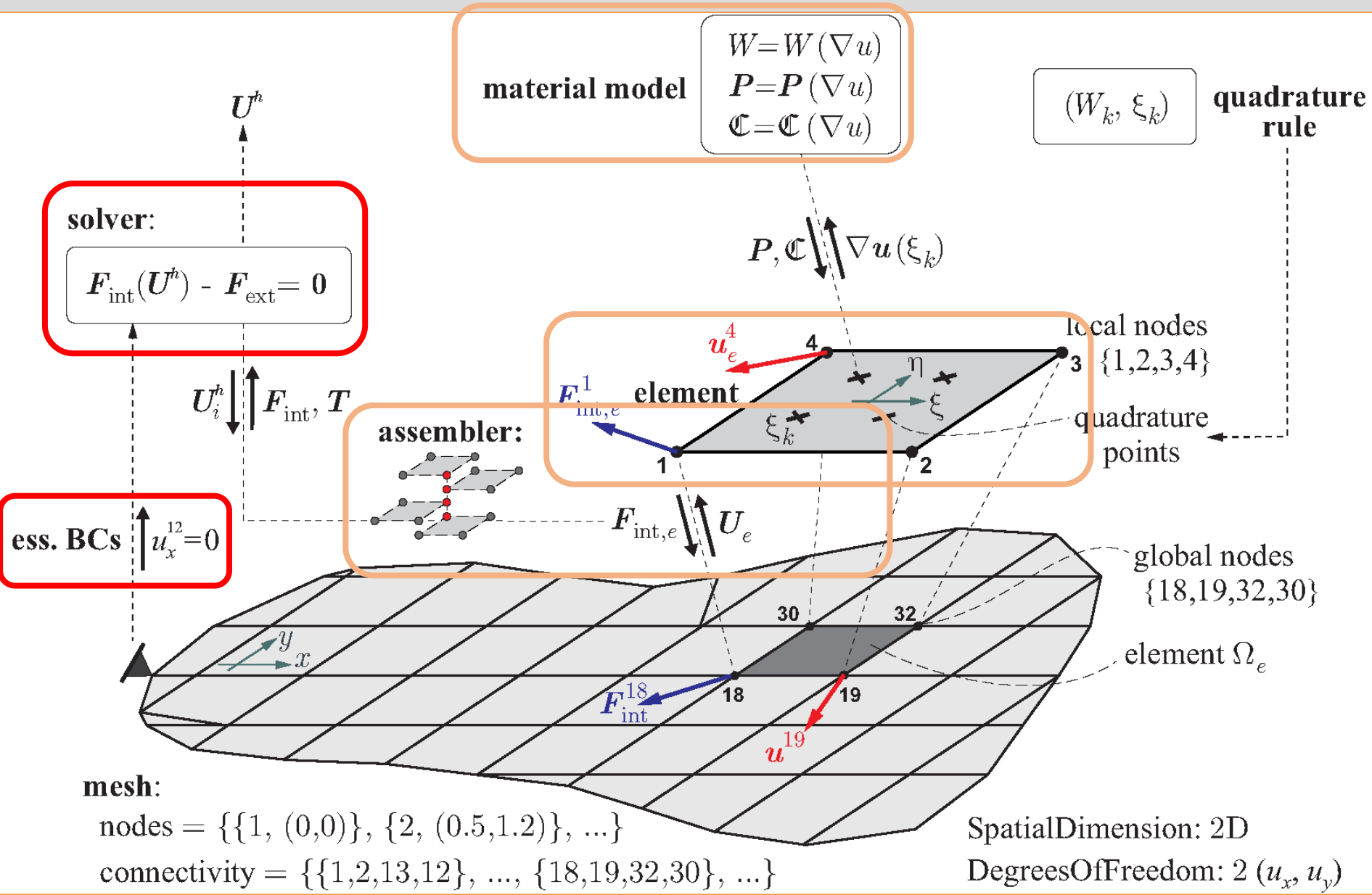
therefore: *ask the element* for its nodal IDs and only pass the corresponding displacements.

- the total energy is the sum of all element energies.

- similarly, assembly of the **global stiffness matrix** and **global internal force vector**

Finite Element Method: Solver and BCs (provided)

Ae214a: Computational Solid Mechanics



Finite Element Method: Mesh Read-In Functionality

Ae214a: Computational Solid Mechanics

contents of a **mesh file** (can be read by the code):

0.0, 0.0, 0.0
1.0, 0.0, 0.0
0.0, 0.0, 1.0
0.0, 1.0, 0.0
connectivity
0, 1
0, 2
0, 3
1, 2
1, 3
2, 3

node locations

element connectivities

Finite Element Method: Have Fun!

Ae214a: Computational Solid Mechanics