Anmerkungen zu Dragonfly

Berechnung eines passiven Skalars mit Hilfe der Finiten Volumen Methode (FVM)

(Instationär, Konvektion, Diffusion, Strömungsfeld bekannt)

Bilanzgleichungen

- Nicht konservativ

$$\rho\frac{\partial\phi}{\partial t}+\rho\vec{v}\cdot\vec{\nabla}\phi=\vec{\nabla}\cdot(a\vec{\nabla}\phi)+s$$

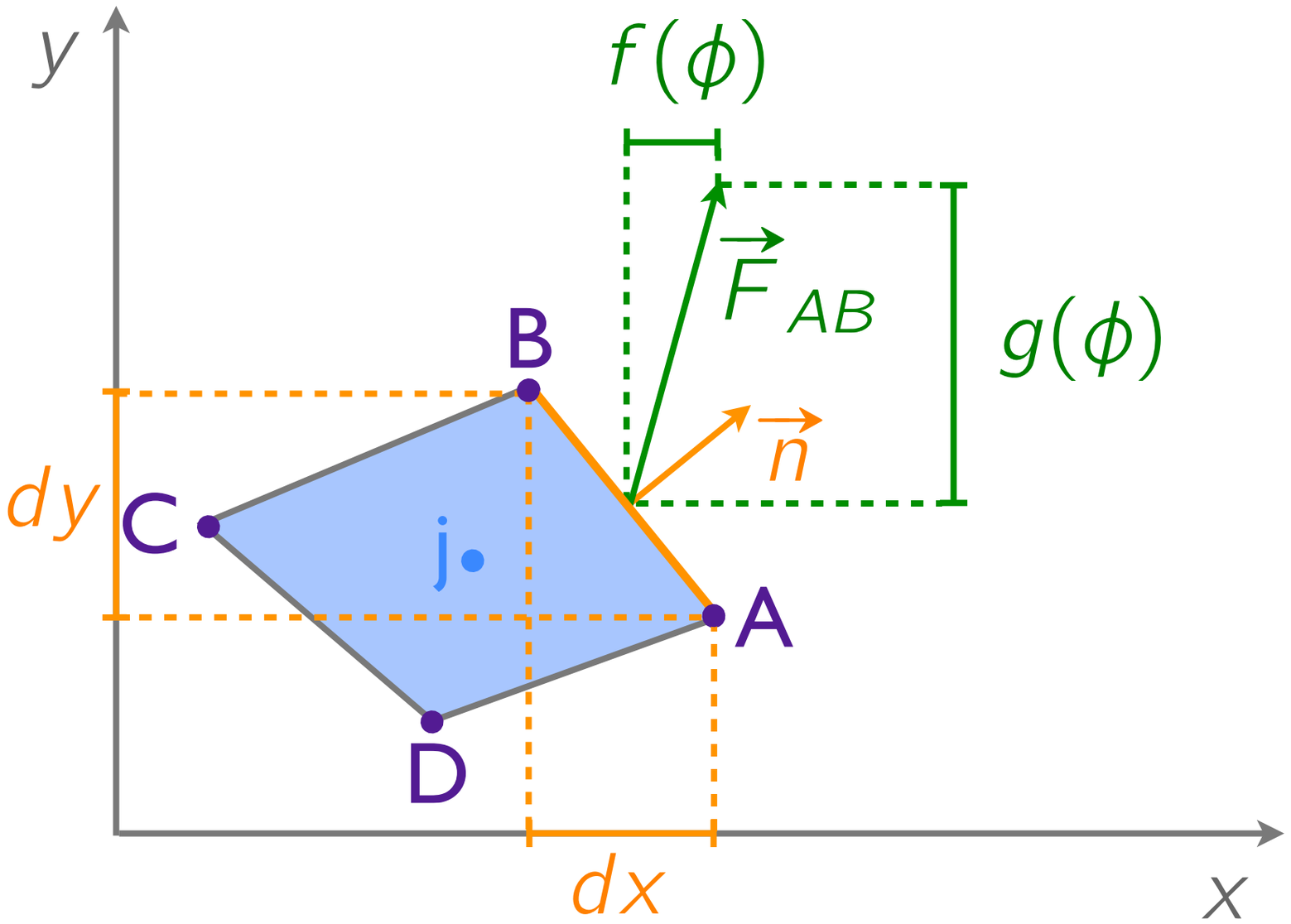
- Konservativ

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\phi dV+\oint_{\partial\Omega}\vec{F}(\phi)\cdot\overrightarrow{dA}=\int_{\Omega}s dV$$

- Für finites Volumen *j*:

$$\frac{\partial}{\partial t}(\rho_j\phi_j\Delta V_j)+\sum_{f=1}^{n_{\text{faces}}}\vec{F}_f(\phi)\cdot\overrightarrow{dA_f}=s_j\Delta V_j$$

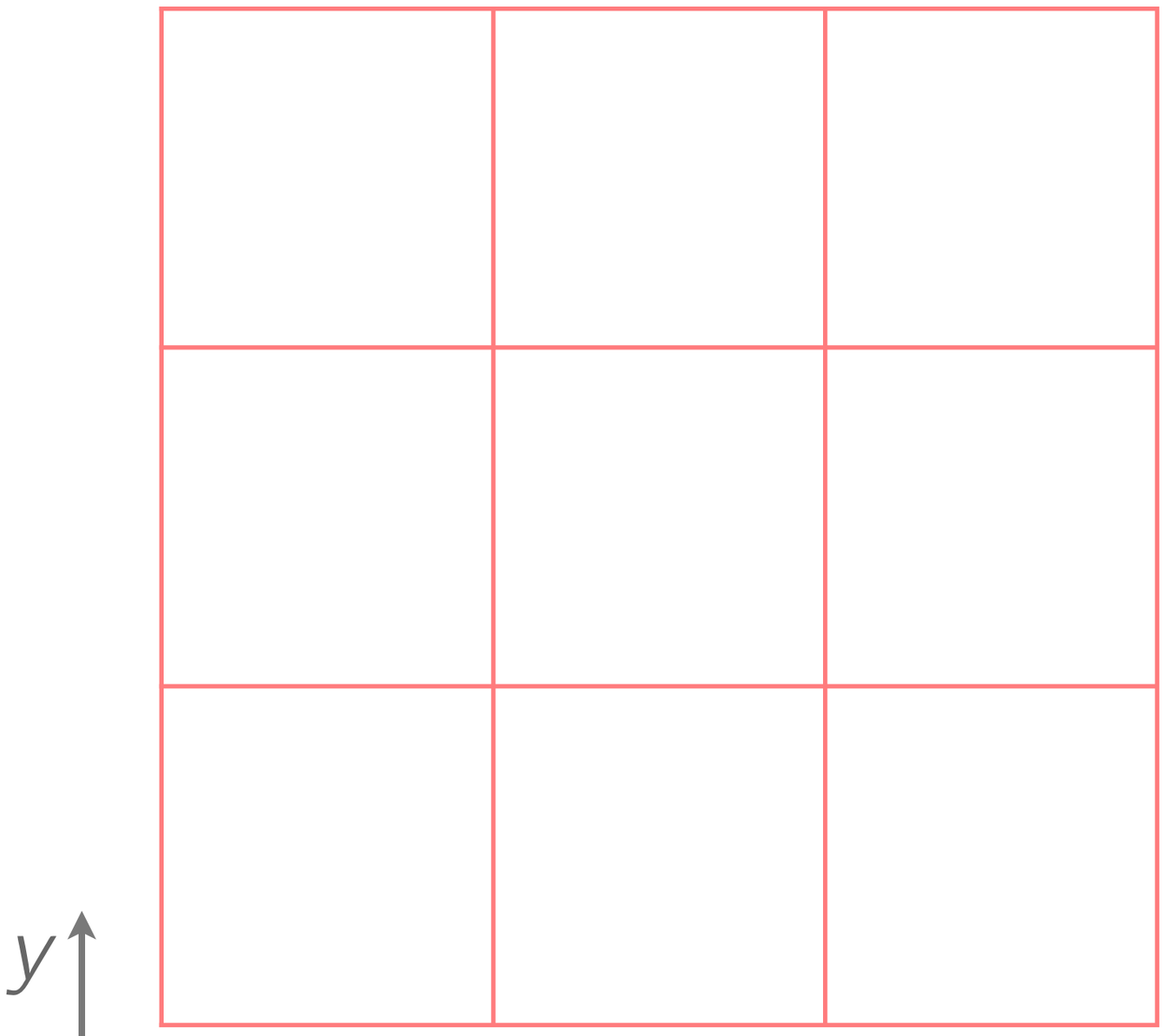
Der Flussvektor $\vec{F}(\phi)$



$$\vec{F}_{AB}(\phi) \cdot \overrightarrow{d\vec{A}} = f(\phi)\Delta y - g(\phi)\Delta x$$

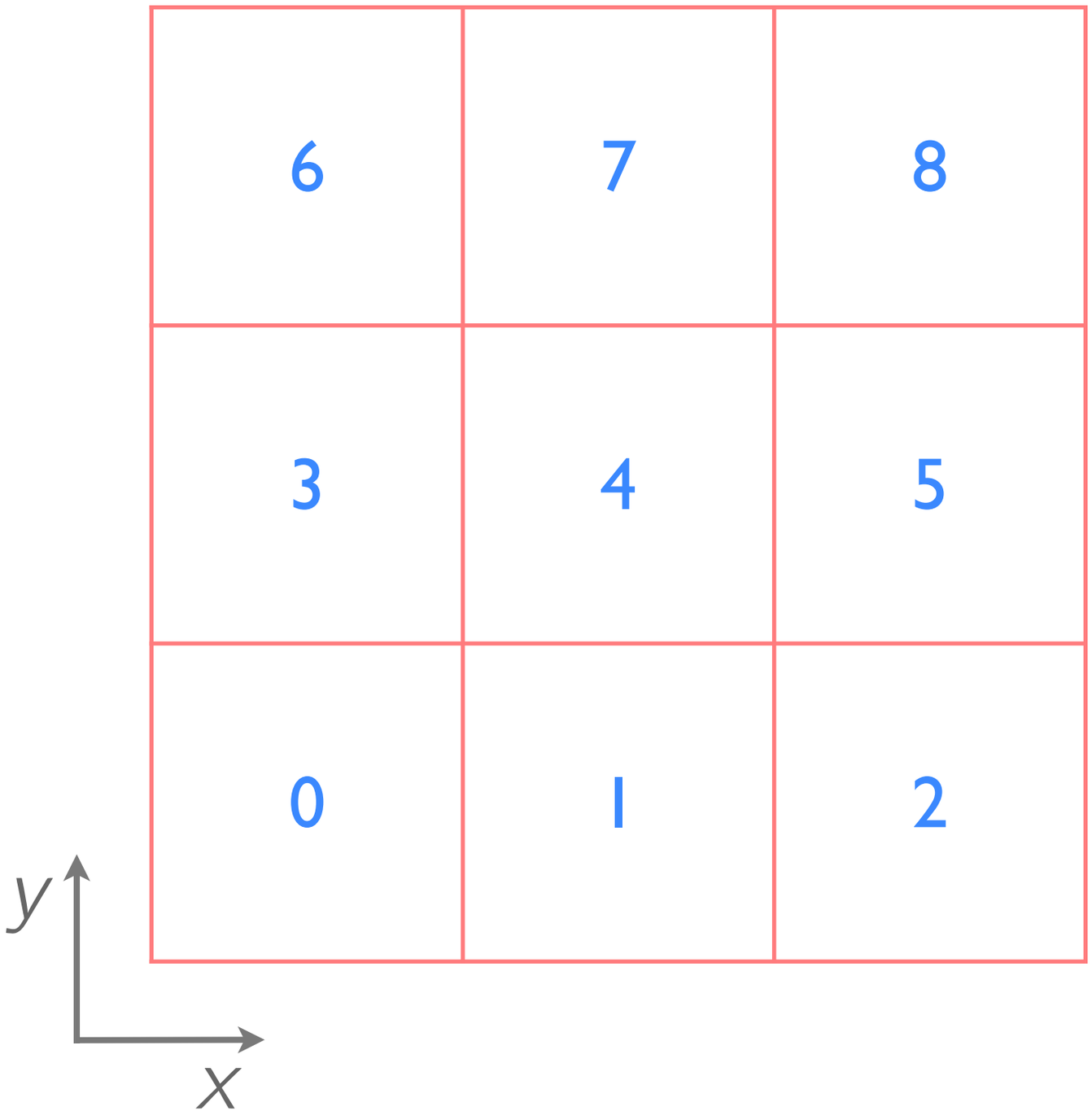
Datenstruktur

```
struct sData{
```



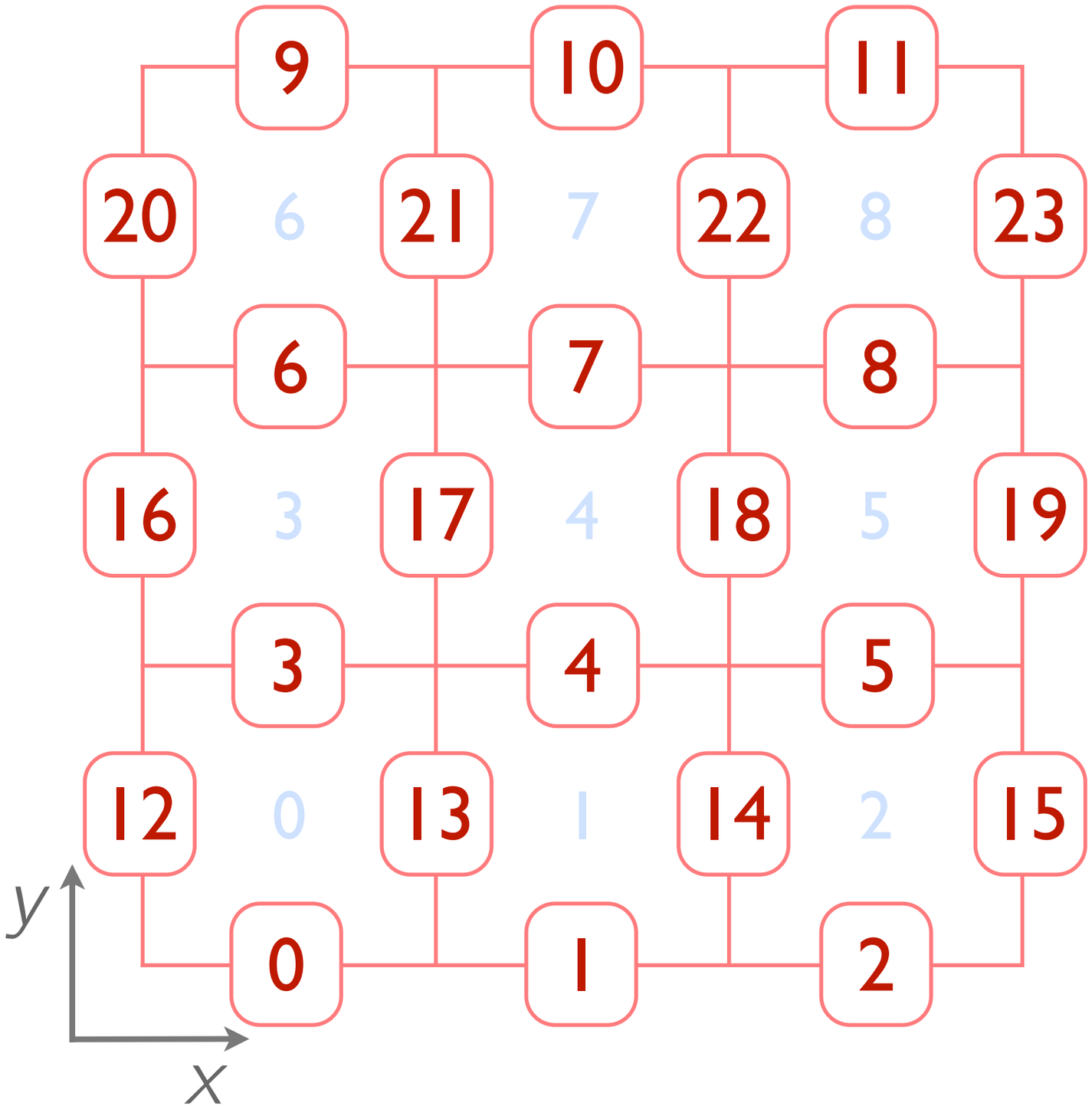
Datenstruktur

```
struct sData{
        sCell* cells;
```




```
struct sCell{
        int id;
```

Datenstruktur

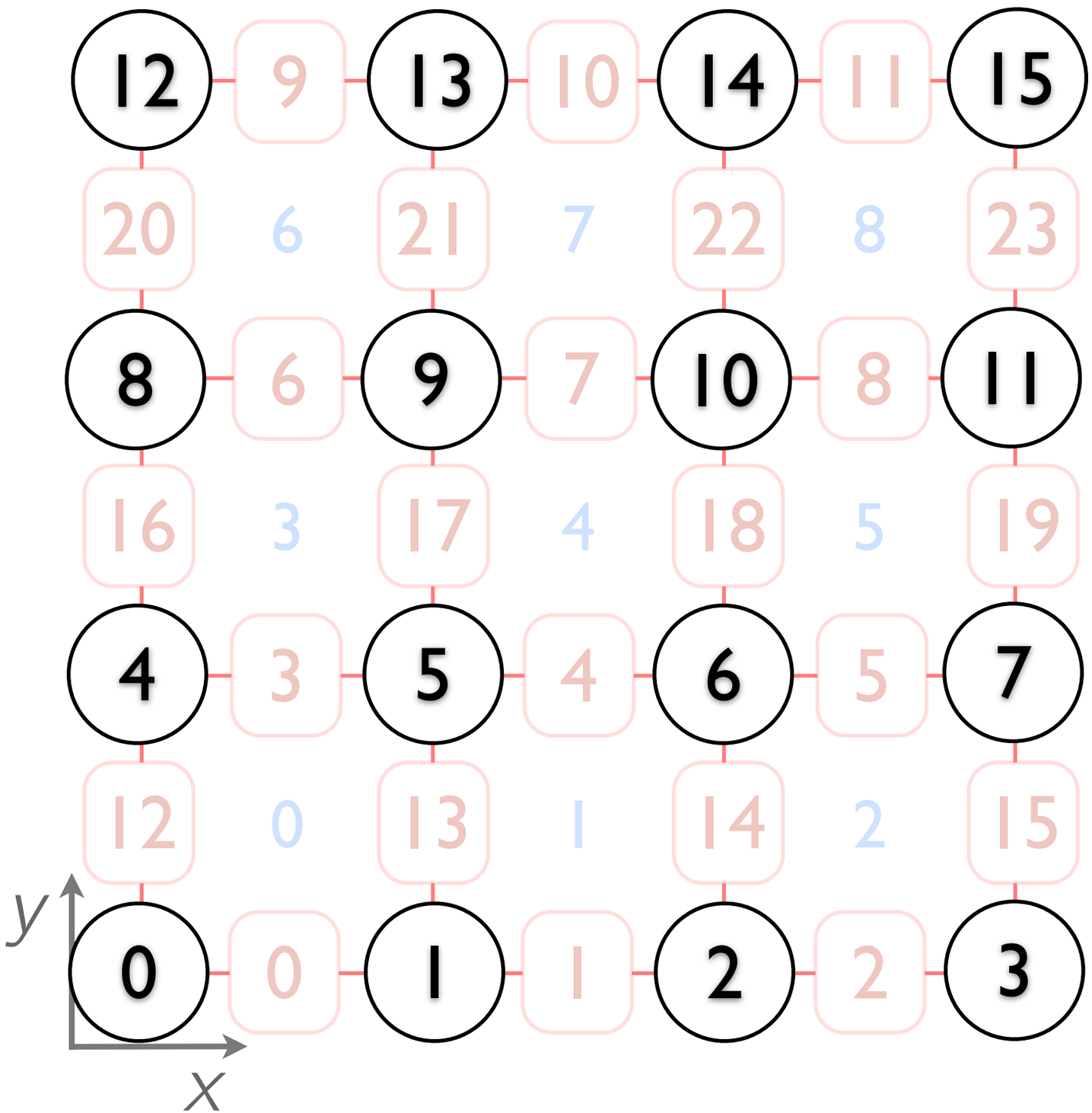



```
struct sData{
        sCell* cells;
        sFace* faces;
```

```
struct sFace{
        int id;
```

```
struct sCell{
        int id;
```

Datenstruktur

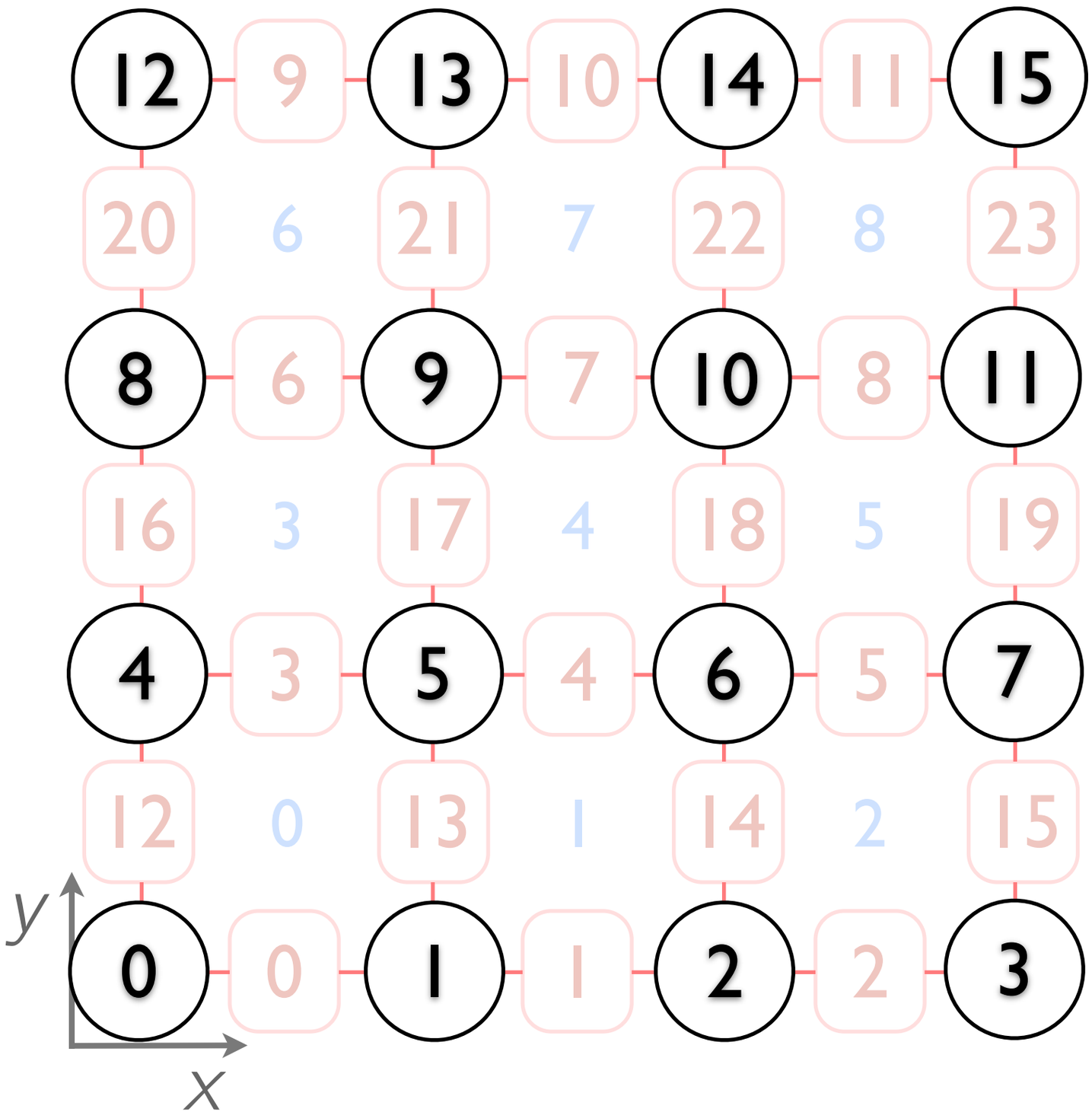



```
struct sData{
                sCell* cells;
                sFace* faces;
                sPoint* points };
struct sPoint{
                int id;

struct sFace{
                int id;

struct sCell{
                int id;
```

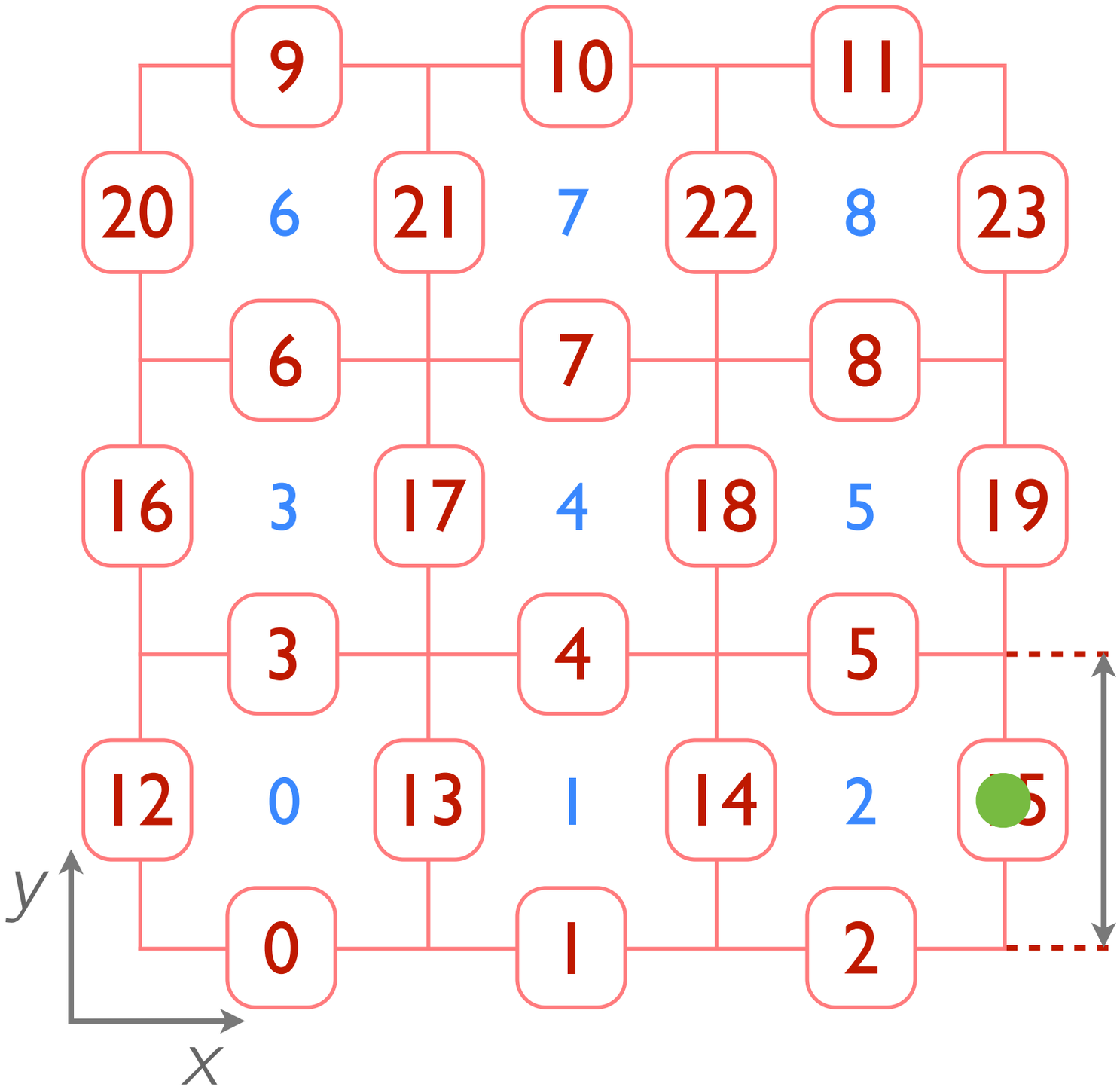
Datenstruktur




```
struct sData{
                sCell* cells;
                sFace* faces;
                sPoint* points };
struct sPoint{
                int id;
                double x,y; };
struct sFace{
                int id;
```

```
struct sCell{
                int id;
```

Datenstruktur

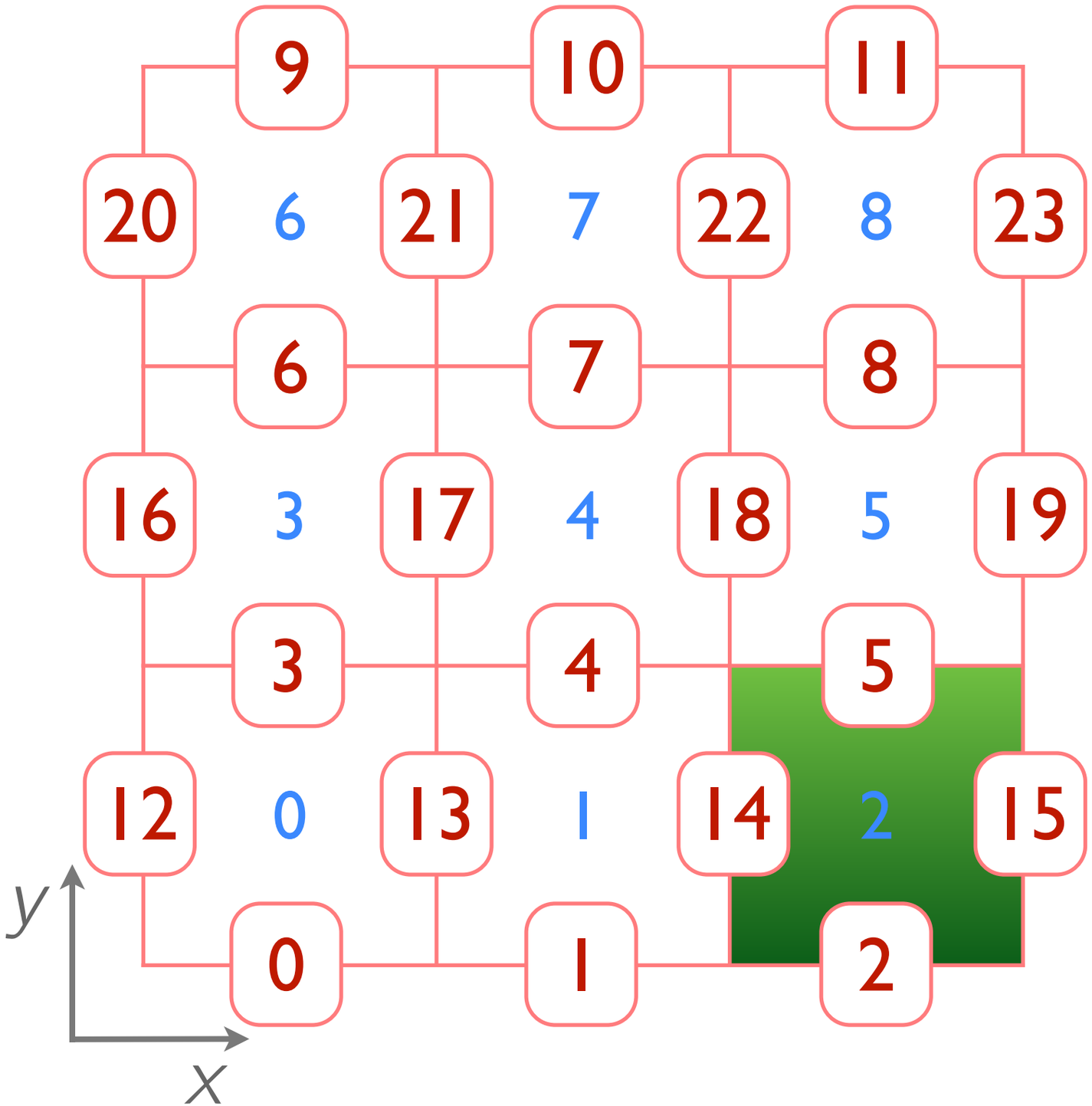



```
struct sData{
                sCell* cells;
                sFace* faces;
                sPoint* points };
struct sPoint{
                int id;
                double x,y; };
struct sFace{
                int id;
                double x,y;
                double dx,dy;


struct sCell{

                int id;
```

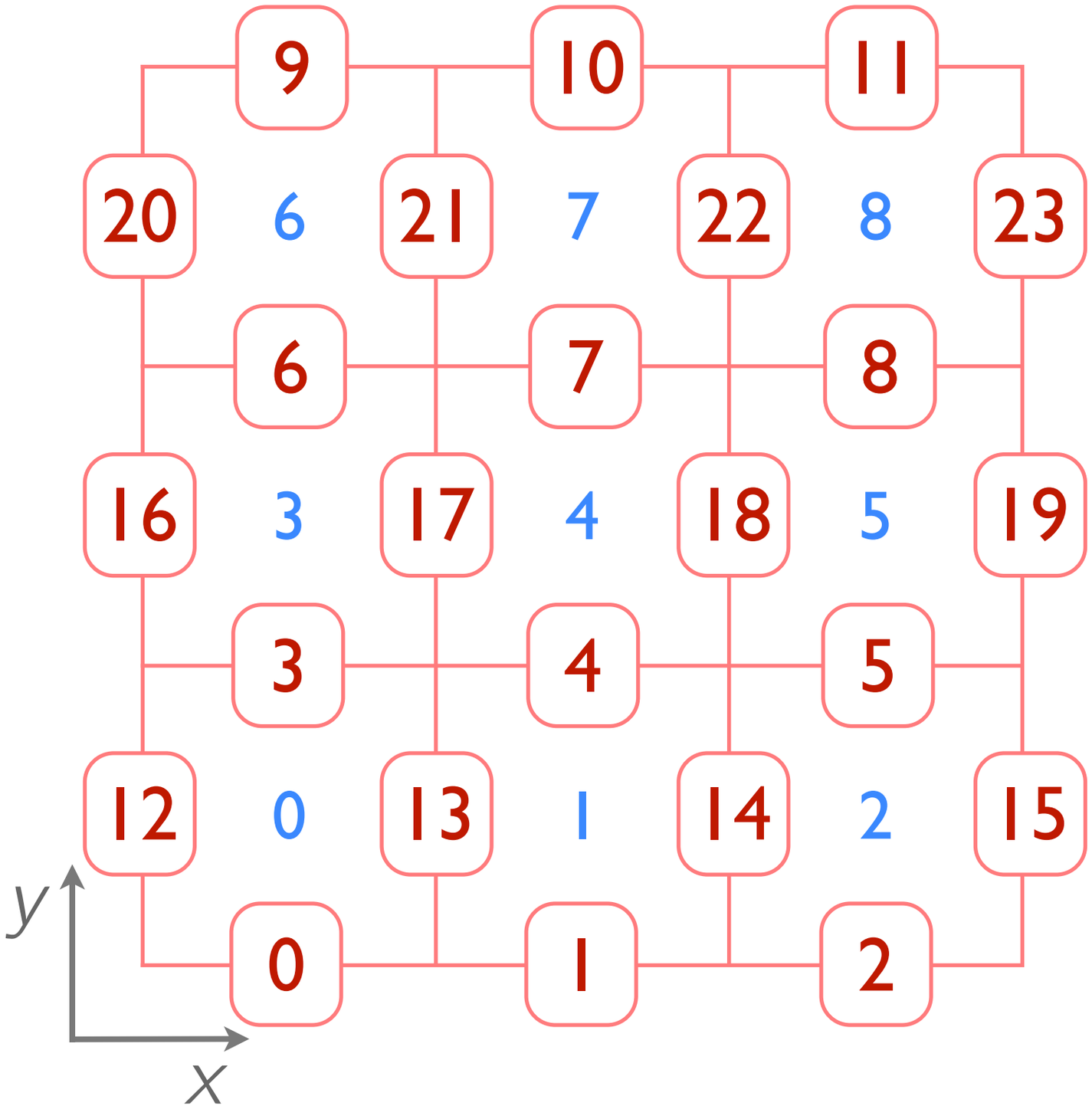
Datenstruktur




```
struct sData{
                sCell* cells;
                sFace* faces;
                sPoint* points };
struct sPoint{
                int id;
                double x,y; };
struct sFace{
                int id;
                double x,y;
                double dx,dy;

struct sCell{
                int id;
                double x,y;
                double volume;
```

Datenstruktur




```
struct sData{
                sCell* cells;
                sFace* faces;
                sPoint* points };
struct sPoint{
                int id;
                double x,y; };
struct sFace{
                int id;
                double x,y;
                double dx,dy;
                sPoint* points[2];
                sCell* neighCells[2]; };
struct sCell{
                int id;
                double x,y;
                double volume;
                sPoint* points[4];
                sFace* faces[4];
                sCell* neighCells[4]; };
```

Mesh file

dragonfly.mesh

```
pointDimensions
4 4

points
# id x y
0 0.0 0.0
1 1.0 0.0
2 2.0 0.0
3 3.0 0.0
…
```

```
boundaryConditions
# type: 1=DIRICHLET
# type: 2=NEUMANN
# cellId type value
0 1 42
1 1 42
2 1 42
3 1 42
5 1 42
…
```

```
initialConditions
# cellId value
0 4711
1 4711
2 4711
3 4711
…
```

Tips

Verwende die Definitionen in data.h, z.B.

```
#define YM  0
#define XP  1
#define YP  2
#define XM  3
```

```
Cell->volume = Cell->faces[XM]->dy *
               Cell->faces[YM]->dx;
```

Tips

Die Implementierung der Finite Volumen Methode ist wesentlich kürzer als die der Differenzen Methode!