New matrix class in DUNE

Santiago Ospina de los Ríos, OPM-OP

List of Contents

- Standard vs DUNE matrix layout
- Consequences on memory
- Issues with current matrix
- New matrix layout
- Closing remarks

Compressed Row Storage Layout

```
uint64_t rows              = 4;
uint64_t offset[rows+1] = { 0, 2, 4, 7, 8 };
uint64_t nnz               = offset[rows];
uint32_t cols[nnz]         = {  0,  1,  1,  3,  2,  3,  4,  5 };
double   val[nnz]          = { 10, 20, 30, 40, 50, 60, 70, 80 };
```

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

Compressed Row Storage Layout

```
uint64_t rows              = 4;
uint64_t offset[rows+1] = { 0, 2, 4, 7, 8 };
uint64_t nnz               = offset[rows];
uint32_t cols[nnz]         = {  0,  1,  1,  3,  2,  3,  4,  5 };
double   val[nnz]          = { 10, 20, 30, 40, 50, 60, 70, 80 };
```

$$\begin{pmatrix} 10 & 20 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 40 & 0 & 0 \\ 0 & 0 & 50 & 60 & 70 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 \end{pmatrix}$$

Memory

```
(rows+1) * sizeof(uint64_t) +
nnz      * (sizeof(double)+sizeof(uint32_t));
```

Dune::BCRSMatrix Layout

```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  enum BuildMode {...};
  enum BuildStage {...};
  using size_type = std::allocator_traits<A>::size_type;

  BuildMode                                          build_mode;                // 4
  BuildStage                                         ready;                     // 4
  size_type                                          n;                         // 8
  size_type                                          m;                         // 8
  mutable size_type                                  nnz_;                      // 8
  size_type                                          allocationSize_;           // 8
  CompressedRow<B,size_type>*                        r;                         // 8
  B*                                                 a;                         // 8
  std::shared_ptr<size_type>                         j_;                        // 2*8
  size_type                                          avg;                       // 8
  double                                             compressionBufferSize_;    // 8
  std::map<std::pair<size_type,size_type>, B>        overflow;                  // 3*8
  [[no_unique_address]] A                            allocator_;                // 0
                                                                                // ---
};                                                                              // 112
```

Dune::BCRSMatrix Layout

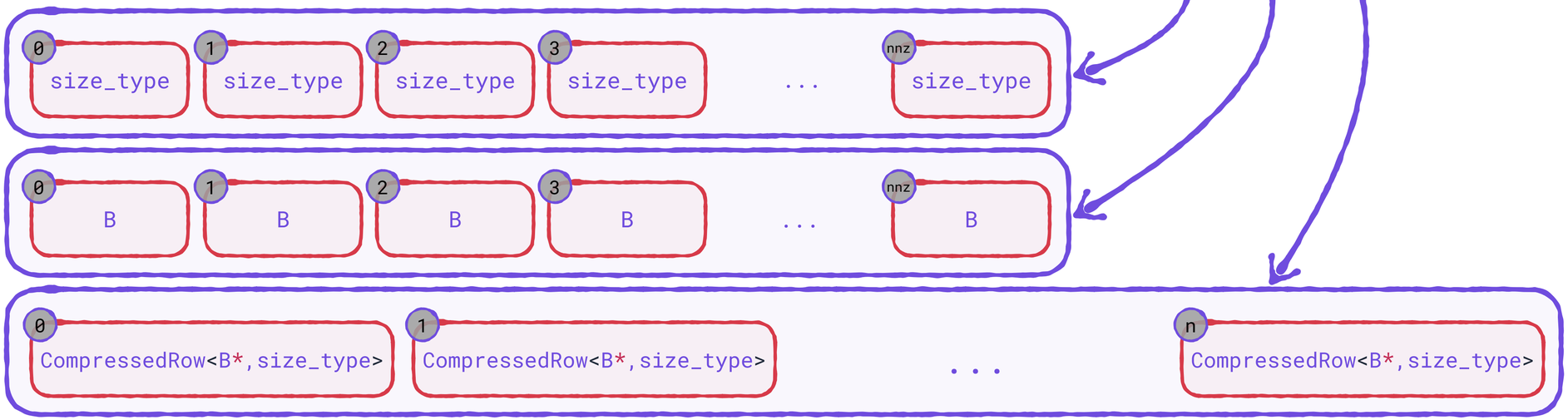
```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  enum BuildMode {...};
  enum BuildStage {...};
  using size_type = std::allocator_traits<A>::size_type;

  BuildMode                                          build_mode;                // 4
  BuildStage                                         ready;                     // 4
  size_type                                          n;                         // 8
  size_type                                          m;                         // 8
  mutable size_type                                  nnz_;                      // 8
  size_type                                          allocationSize_;           // 8
  CompressedRow<B,size_type>*                        r;                         // 8
  B*                                                 a;                         // 8
  std::shared_ptr<size_type>                         j_;                        // 2*8
  size_type                                          avg;                       // 8
  double                                             compressionBufferSize_;    // 8
  std::map<std::pair<size_type,size_type>, B>        overflow;                  // 3*8
  [[no_unique_address]] A                            allocator_;                // 0
                                                                                // ---
};                                                                              // 112
```

Dune::BCRSMatrix Layout

```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  using size_type = std::allocator_traits<A>::size_type;

  size_type                       n;      //   8
  CompressedRow<B,size_type>*     r;      //   8
  B*                              a;      //   8
  std::shared_ptr<size_type>      j_;     // 2*8
  ...                                     // ---
};                                        // 112
```

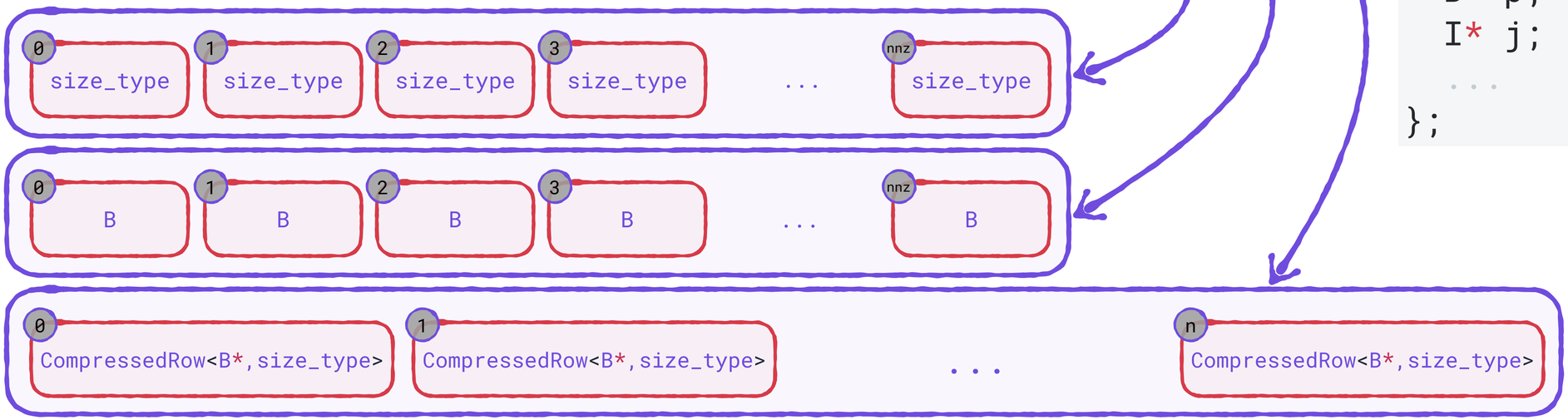
Dune::BCRSMatrix Layout

```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  using size_type = std::allocator_traits<A>::size_type;

  size_type                        n;      //   8
  CompressedRow<B,size_type>*      r;      //   8
  B*                               a;      //   8
  std::shared_ptr<size_type>       j_;     // 2*8
  ...                                      // ---
};                                         // 112
```



Dune::BCRSMatrix Layout

```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  using size_type = std::allocator_traits<A>::size_type;

  size_type                       n;      //   8
  CompressedRow<B,size_type>*     r;      //   8
  B*                              a;      //   8
  std::shared_ptr<size_type>      j_;     // 2*8
  ...                                     // ---
};                                        // 112
```




```
template<class B, class I>
class CompressedRow {
  I n;  //   8
  B *p; //   8
  I* j; //   8
  ...   // ---
};      //  24
```

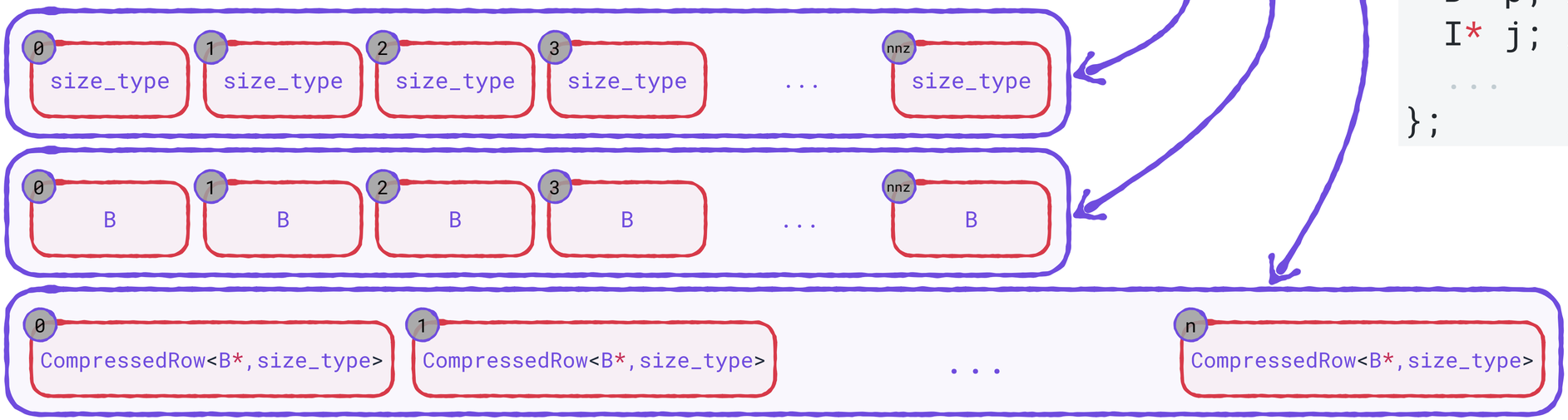
Dune::BCRSMatrix Layout

```
template<class B, class A = std::allocator<B>>
class BCRSMatrix {
  using size_type = std::allocator_traits<A>::size_type;

  size_type                          n;      //    8
  CompressedRow<B,size_type>*        r;      //    8
  B*                                 a;      //    8
  std::shared_ptr<size_type>         j_;     // 2*8
  ...                                        // ---
};                                           // 112
```

Memory

```
rows * sizeof(row_type) +
nnz  * (sizeof(B)+sizeof(size_type));
```

```
template<class B, class I>
class CompressedRow {
  I n;  //    8
  B *p; //    8
  I* j; //    8
  ...   // ---
};      //   24
```

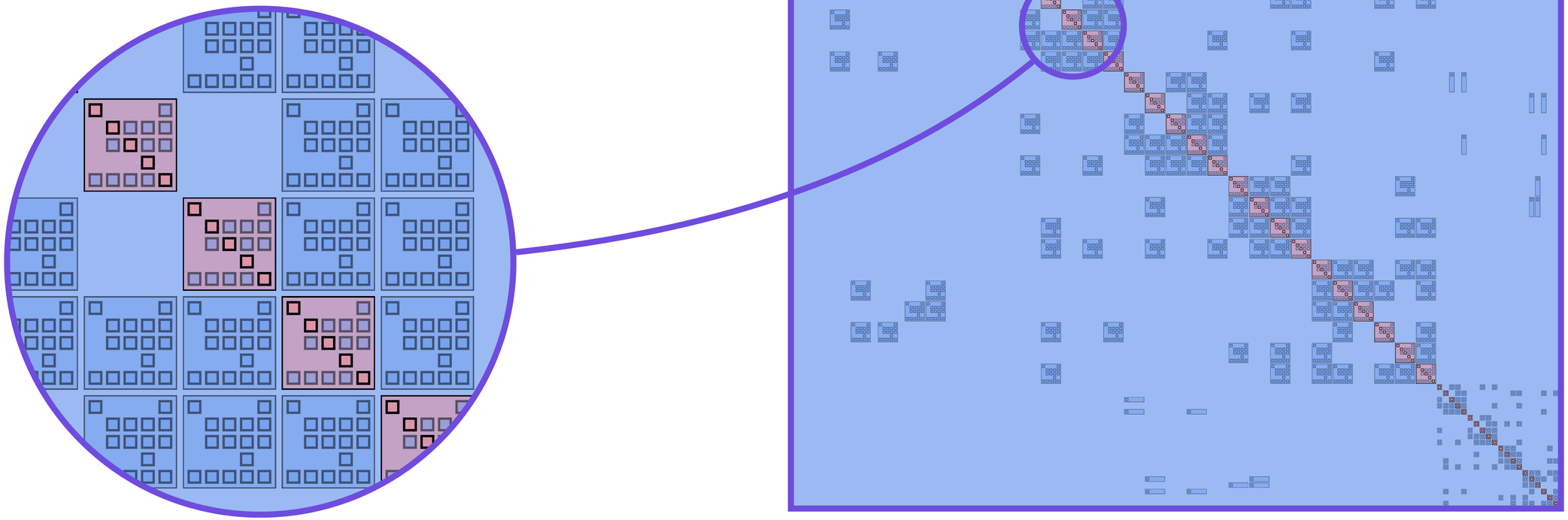


Dune::BCRSMatrix Recursive Layout

```
template<class B, class A>
class BCRSMatrix<Dune::BCRSMatrix<B>,A> {
  ...                          // ---
  Dune::BCRSMatrix<B>* a;      //   8
  ...                          // ---
};                             // 112
```

Dune::BCRSMatrix Recursive Layout

```
template<class B, class A>
class BCRSMatrix<Dune::BCRSMatrix<B>,A> {
  ...                          // ---
  Dune::BCRSMatrix<B>* a;      //   8
  ...                          // ---
};                             // 112
```



Dune::BCRSMatrix Recursive Layout

```
template<class B, class A>
class BCRSMatrix<Dune::BCRSMatrix<B>,A> {
  ...                          // ---
  Dune::BCRSMatrix<B>* a;      //   8
  ...                          // ---
};                             // 112
```

- Each block costs 112 bytes *without* even counting its data!

Example

300K x 300K with 2M non-zeroes

Dune::BCRSMatrix Layout

Memory

```
rows * sizeof(row_type) +
nnz  * (sizeof(B)+sizeof(size_type));
```

Vanilla CRSMatrix Layout

Memory

```
(rows+1) * sizeof(uint64_t) +
nnz      * (sizeof(B)+sizeof(uint32_t));
```

Example

300K x 300K with 2M non-zeroes

Dune::BCRSMatrix Layout

Memory

```
rows * sizeof(row_type) +
nnz  * (sizeof(B)+sizeof(size_type));
```

Vanilla CRSMatrix Layout

Memory

```
(rows+1) * sizeof(uint64_t) +
nnz      * (sizeof(B)+sizeof(uint32_t));
```

37,4 MB	~32% less memory (B=`double`)	25,2 MB
159,5 MB	~8% less memory (B=`FieldMatrix<double,3,3>`)	147,2 MB
295,3 MB	~28% less memory (B=`double` with 3x3 flat blocks)	212,9 MB
235,7 MB	~70% less memory (B=`BCRSMatrix` or B=`CRSMatrix`)	70,9 MB

Example

300K x 300K with 2M non-zeroes

Dune::BCRSMatrix Layout

Memory

```
rows * sizeof(row_type) +
nnz  * (sizeof(B)+sizeof(size_type));
```

Vanilla CRSMatrix Layout

Memory

```
(rows+1) * sizeof(uint64_t) +
nnz      * (sizeof(B)+sizeof(uint32_t));
```

37,4 MB	~32% less memory (B=`double`)	25,2 MB
159,5 MB	~8% less memory (B=`FieldMatrix<double,3,3>`)	147,2 MB
295,3 MB	~28% less memory (B=`double` with 3x3 flat blocks)	212,9 MB
235,7 MB	~70% less memory (B=`BCRSMatrix` or B=`CRSMatrix`)	70,9 MB

Reminder

Performance on memory bound programs (e.g., OPMFlow):
More memory usage => More memory bandwidth => Slower program

Issues with current implementation

- Each BCRSMatrix is memory wasteful for blocking
- Stored column index is memory wasteful
- Row offsets are doubly stored
- Rows may be not be allocated in one chunk
- Pattern does not grow efficiently
- Pattern is only shared partially
- Only stateless allocator supported
- No move semantics

Expected outcomes

- ~10% linear solver speed up when memory bandwidth is saturated.
- Improve sparse block support:
  - Sparse blocks allow to have many more components per DoFs (e.g. compositional solver).
- Dynamic pattern building (i.e. more flexible choice for preallocation size).

Solution

Implement a new object `Dune::IBCRSMatrix`

- Separate pattern creation from matrix data-structure
- Template choice for index storage (uint64_t —> uint32_t)
- True compressed row storage

Layout

Solution

Implement a new object `Dune::IBCRSMatrix`

- Separate pattern creation from matrix data-structure
- Template choice for index storage (uint64_t —> uint32_t)
- True compressed row storage

Layout

```
template<class I, class A>
class SparseIndexSets {
  using size_type = typename A::size_type;
  size_type                 n_, m_, nnz_;   // 3*8
  I*                        col_indices_;   //   8
  size_type*                row_offset_;    //   8
  [[no_unique_address]] A   allocator_;     //   0
                                            // ---
};                                          //  40
```

Solution

Implement a new object `Dune::IBCRSMatrix`

- Separate pattern creation from matrix data-structure
- Template choice for index storage (uint64_t —> uint32_t)
- True compressed row storage

Layout

```
template<class I, class A>
class SparseIndexSets {
  using size_type = typename A::size_type;
  size_type                 n_, m_, nnz_;   // 3*8
  I*                        col_indices_;   //   8
  size_type*                row_offset_;    //   8
  [[no_unique_address]] A   allocator_;     //   0
                                            // ---
};                                          //  40
```

```
template<class B, class IS, class A>
class IBCRSMatrix {
  B*                                    data_;       //   8
  std::shared_ptr<IS>                   pattern_;    // 2*8
  std::unique_ptr<typename IS::Builder> builder_;    //   8
  [[no_unique_address]] A               allocator_;  //   0
                                                     // ---
};                                                   //  32
```

Compromises

- Rows become proxy-objects as they are not stored. This means that non-mutable references are not possible.
- Implicit pattern builder does not store data while creating the pattern.

Compromises

- Rows become proxy-objects as they are not stored. This means that non-mutable references are not possible.
- Implicit pattern builder does not store data while creating the pattern.

```
using Matrix = Dune::IBCRSMatrix<double>;
Matrix mat(10, 10, Matrix::BuildMode::implicit);

mat.entry(1, 1) = double{1.0}; // throw exception: values are not stored
mat.entry(1, 1) = double{0.0}; // OK: matrix is initialized with default values

      auto& row = mat[0];      // error: operator[](size_type) returns an pr-value
const auto& row = mat[0];      // OK: pr-values lifetime is extended
```

Thanks for your attention

Questions?