Format Abstraction for Sparse Tensor Algebra Compilers

Stephen Chou, Fredrik Kjolstad, and Saman Amarasinghe

Sparse tensors are a natural way of representing real-world data

Sparse tensors are a natural way of representing real-world data

Kristina

Great Product

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

Excellent buy

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

I was really disappointed. The spoon holder it self was great and ...

December 31, 2016

Color: Black | Verified Purchase

This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

Sarah

Malfunctioned within a month. Waste of $.

December 5, 2017

Style: Battery Powered Alarm | Size: 1 Pack | Verified Purchase

I chose this one because the reviews were good. It malfunctioned within a month. The back of the alarm has a key for the chirps and of course mine was a lemon. It looks like it was just made August 9th, 2017. I received it at the end of October and it died mid-November. It was a waste of money.

Sparse tensors are a natural way of representing real-world data

Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

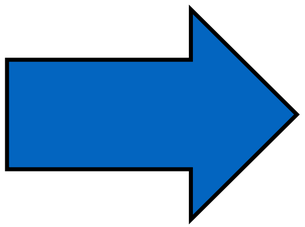
Sarah

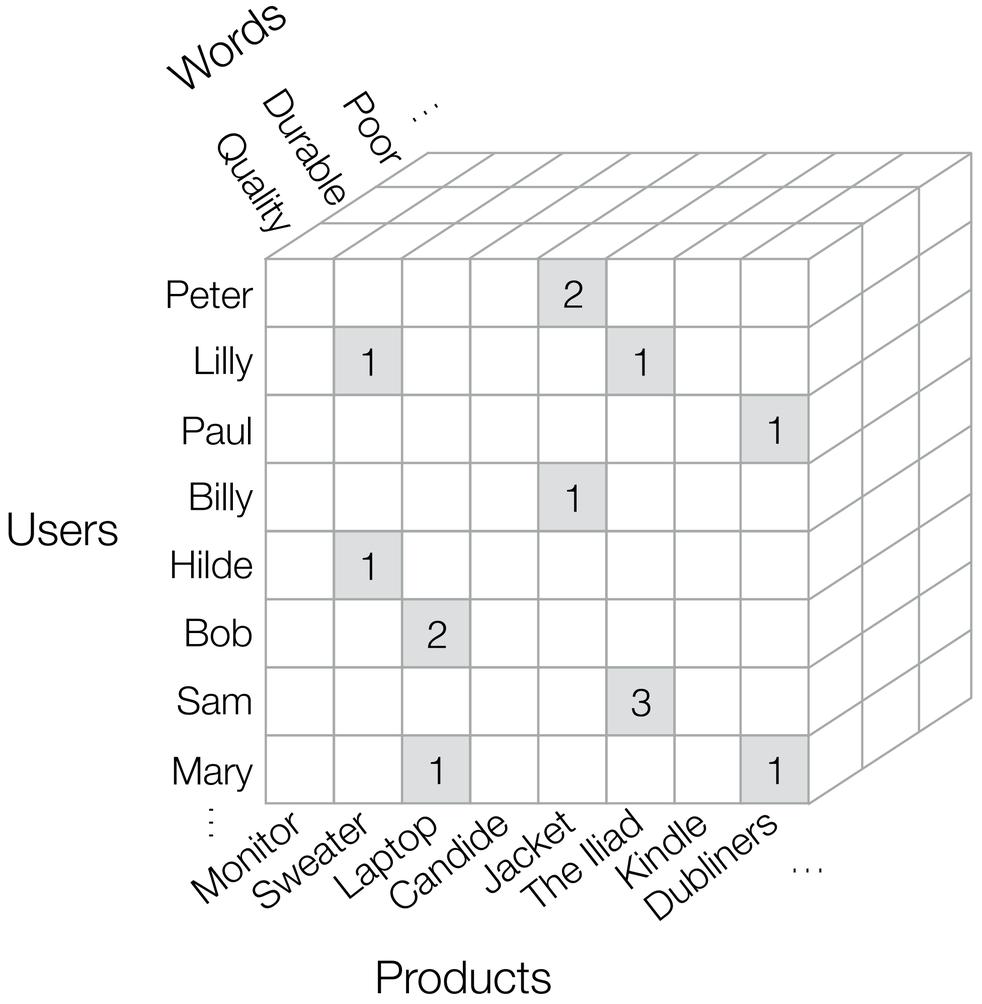
★☆☆☆☆ **Malfunctioned within a month. Waste of $.**

December 5, 2017

Style: Battery Powered Alarm | Size: 1 Pack | Verified Purchase

I chose this one because the reviews were good. It malfunctioned within a month. The back of the alarm has a key for the chirps and of course mine was a lemon. It looks like it was just made August 9th, 2017. I received it at the end of October and it died mid-November. It was a waste of money.





Sparse tensors are a natural way of representing real-world data

Kristina

★★★★☆ **Great Product**

March 30, 2017

Color: White | Verified Purchase

Great product. Large enough for all spoons and fits nicely on my stovetop. Would definitely buy it again.

Teresa

★★★★★ **Excellent buy**

October 25, 2017

Verified Purchase

This is a great product for your boy who loves sports! It was a good value as well. Other stores sell for 3x the cost. I bought one for a basketball and football and my 9 year old loves it in his room. Solid item too, not flimsy. Will hold items nicely.

Lisa

★★☆☆☆ **I was really disappointed. The spoon holder it self was great and ...**

December 31, 2016

Color: Black | Verified Purchase

This product came with a manufacture's chips in it. It is not the sellers fault but I do not know how many in this batch this seller may have. I was really disappointed. The spoon holder it self was great and larger then I expected.

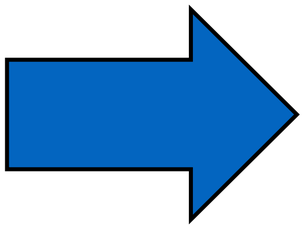
Sarah

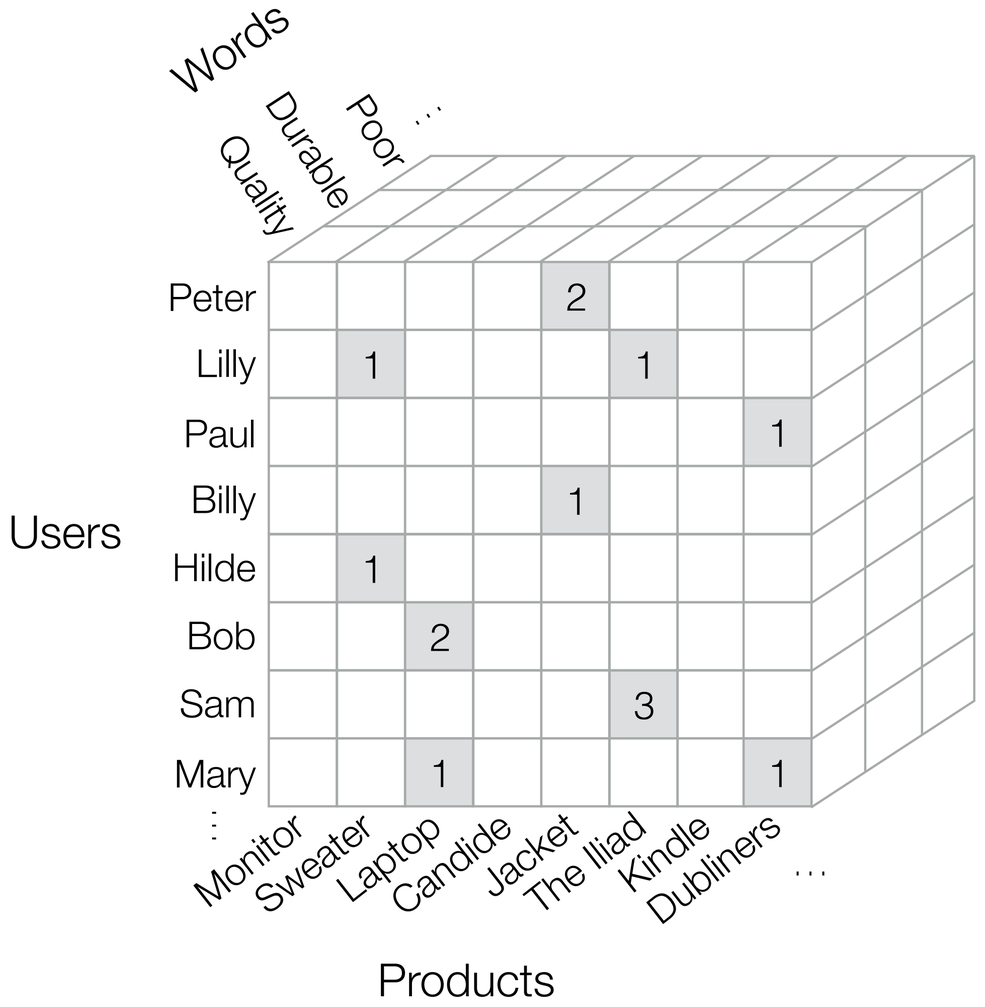
★☆☆☆☆ **Malfunctioned within a month. Waste of $.**

December 5, 2017

Style: Battery Powered Alarm | Size: 1 Pack | Verified Purchase

I chose this one because the reviews were good. It malfunctioned within a month. The back of the alarm has a key for the chirps and of course mine was a lemon. It looks like it was just made August 9th, 2017. I received it at the end of October and it died mid-November. It was a waste of money.





Dense storage: 107 exabytes

Sparse storage: 13 gigabytes

Many different formats for storing tensors exist

CSR

Dense array matrix

DCSR

BCSR

Coordinate matrix

ELLPACK

CSB

BCOO

CSC

DIA

Block DIA

DCSC

LIL

Skyline

SELL

BELL

Many different formats for storing tensors exist

CSR

Dense array matrix

DCSR

BCSR

Coordinate matrix

ELLPACK

CSB

BCOO

CSC

DIA

Block DIA

DCSC

LIL

Skyline

SELL

BELL

Sparse vector

Hash maps

Dense array vector

Many different formats for storing tensors exist

CSR

Dense array matrix

DCSR

BCSR

Coordinate matrix

ELLPACK

CSB

BCOO

CSC

DIA

Block DIA

DCSC

LIL

SELL

BELL

Skyline

Sparse vector

Dense array vector

Hash maps

HiCOO

Coordinate tensor

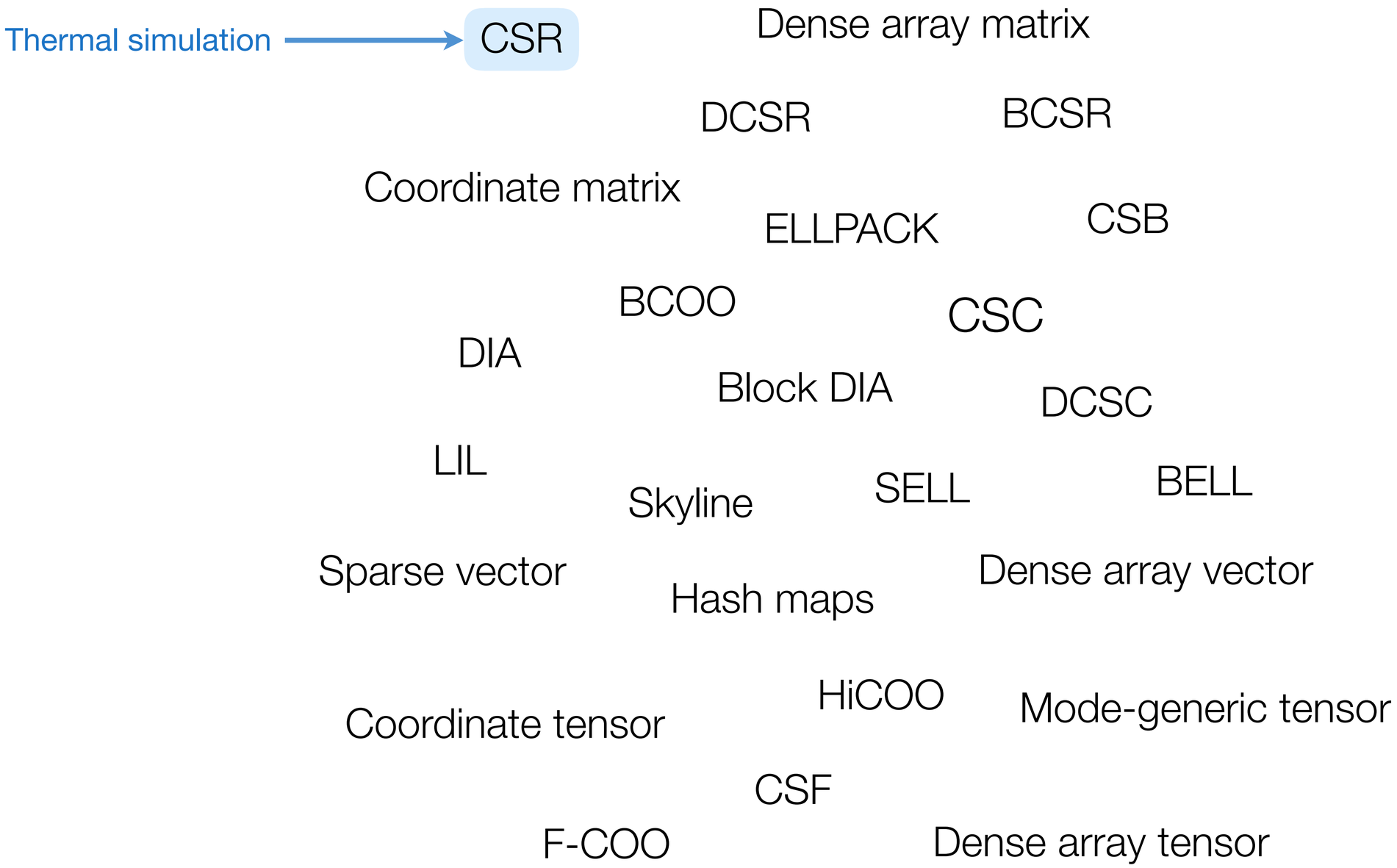
Mode-generic tensor

CSF

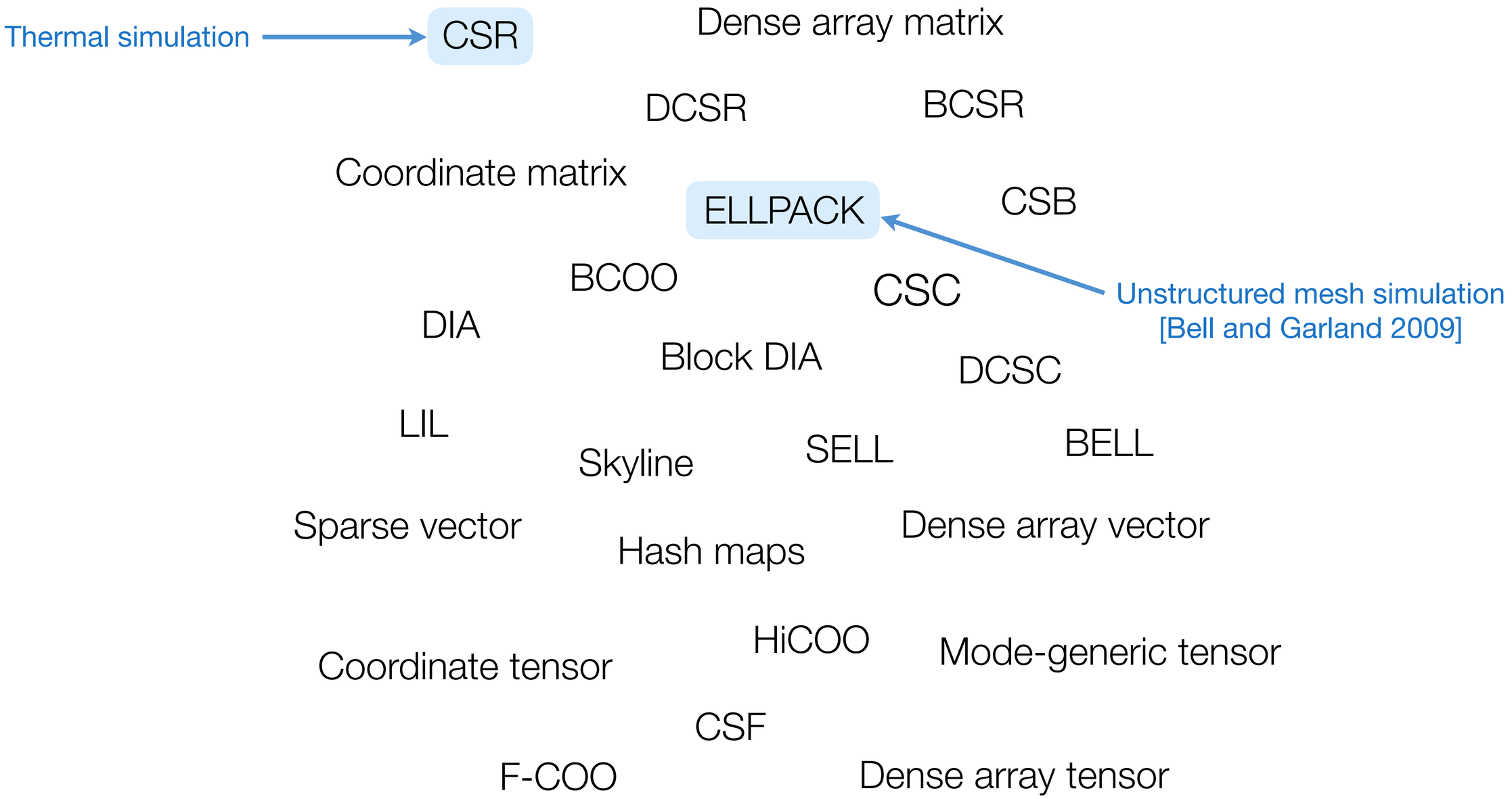
F-COO

Dense array tensor

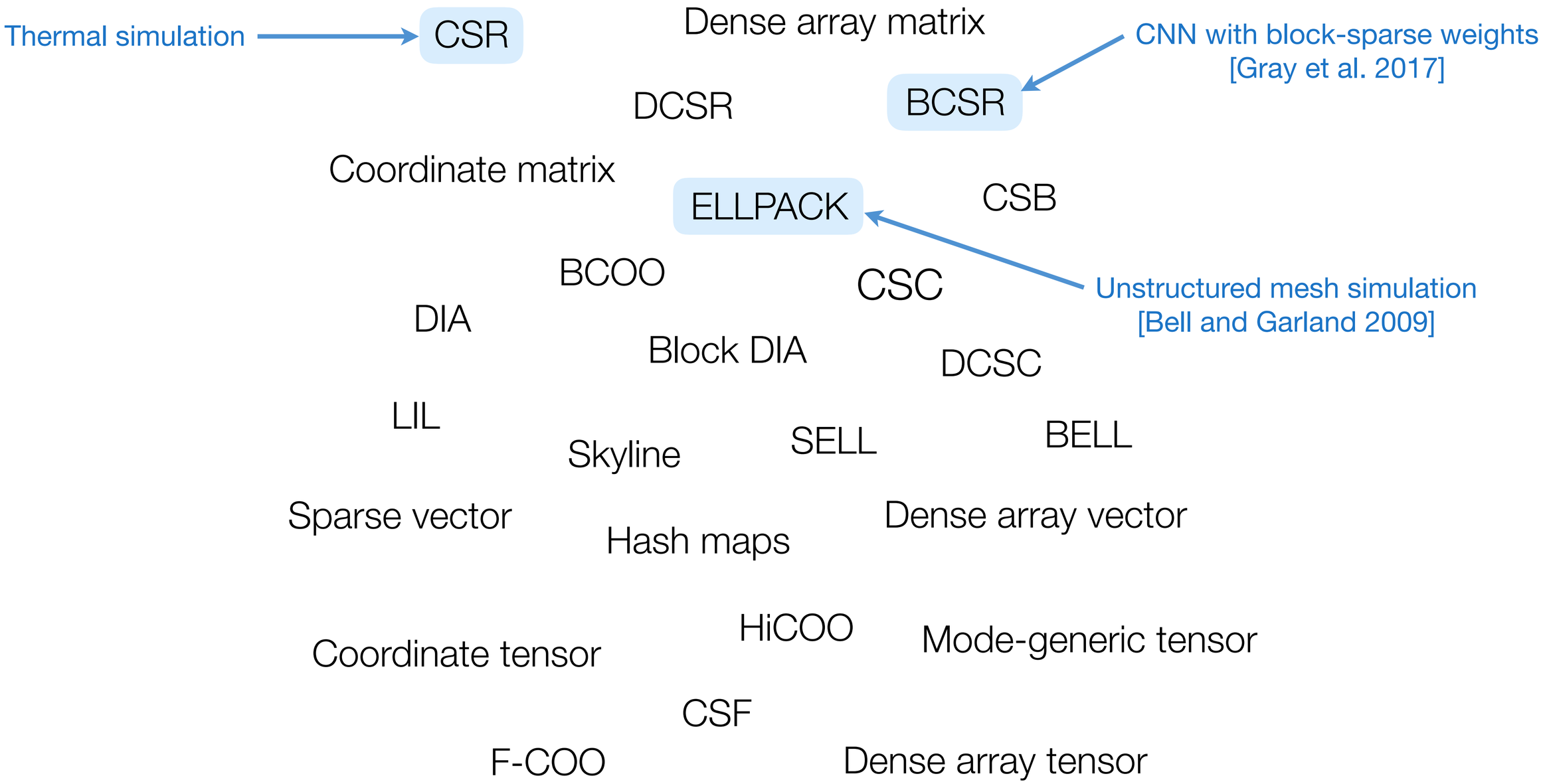
Many different formats for storing tensors exist



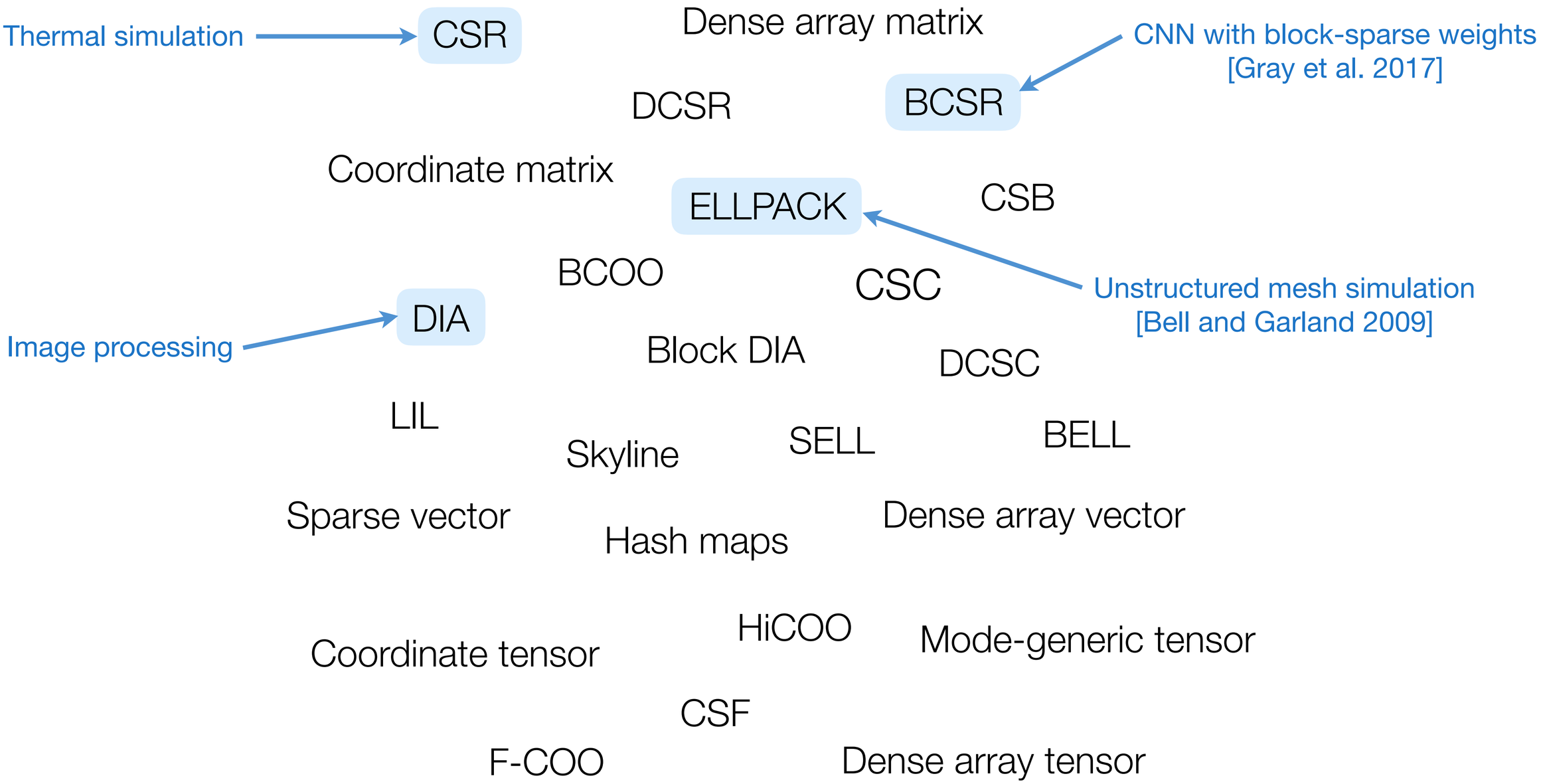
Many different formats for storing tensors exist



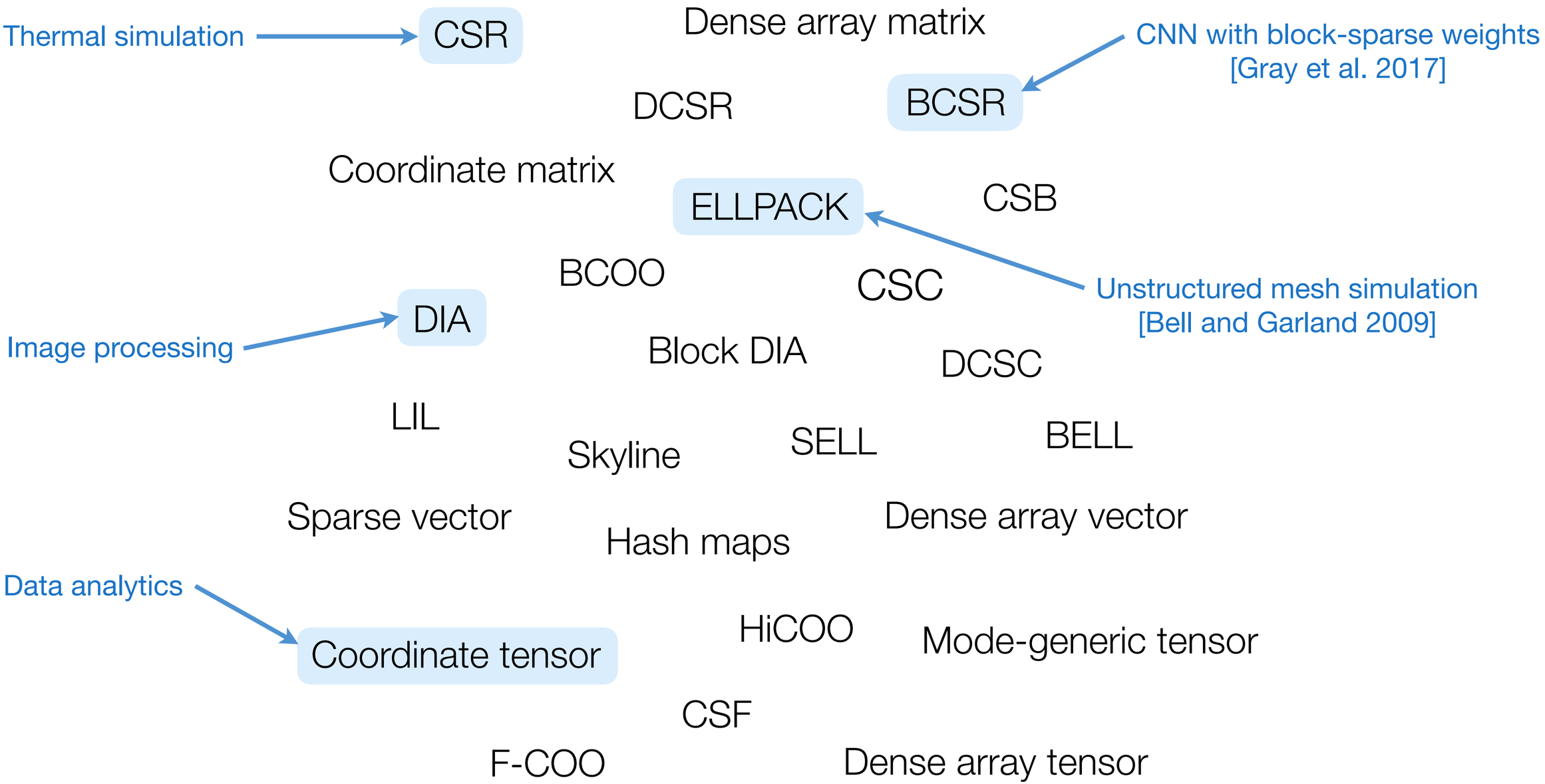
Many different formats for storing tensors exist



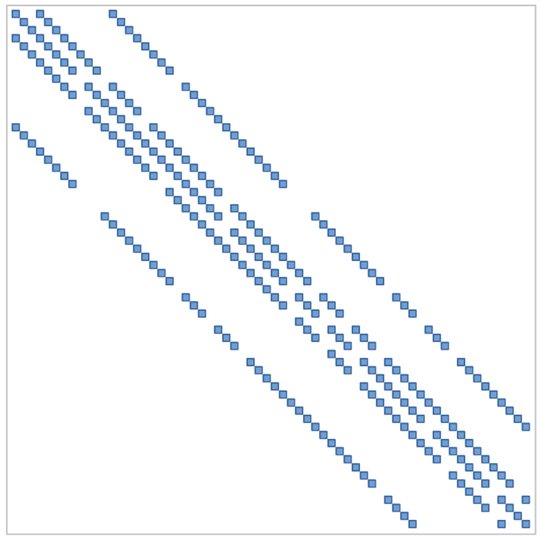
Many different formats for storing tensors exist

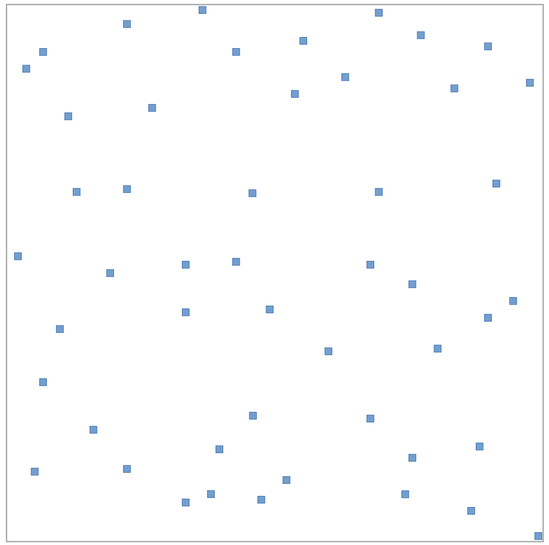


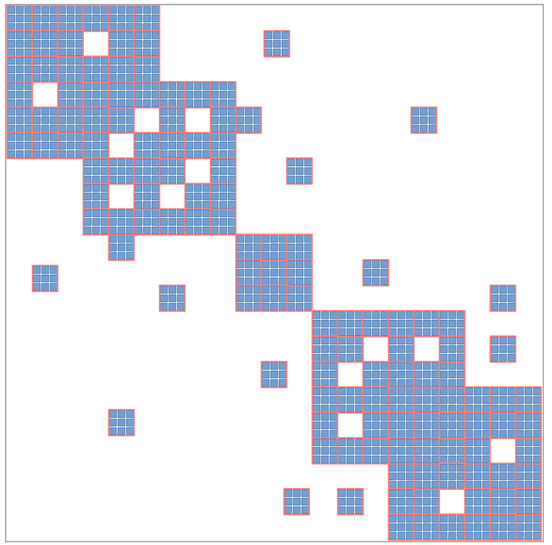
Many different formats for storing tensors exist



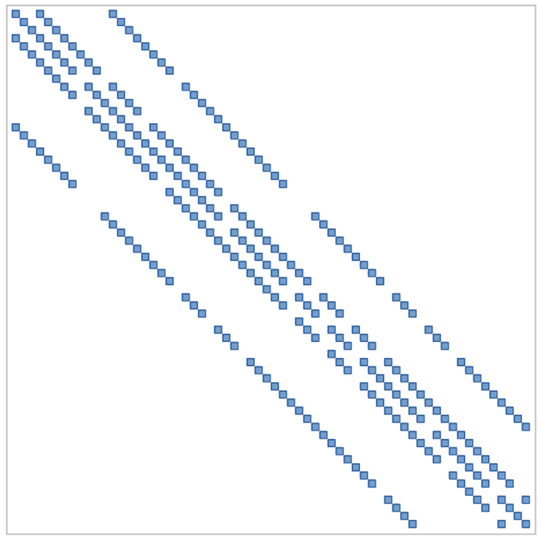
There is no universally superior tensor format



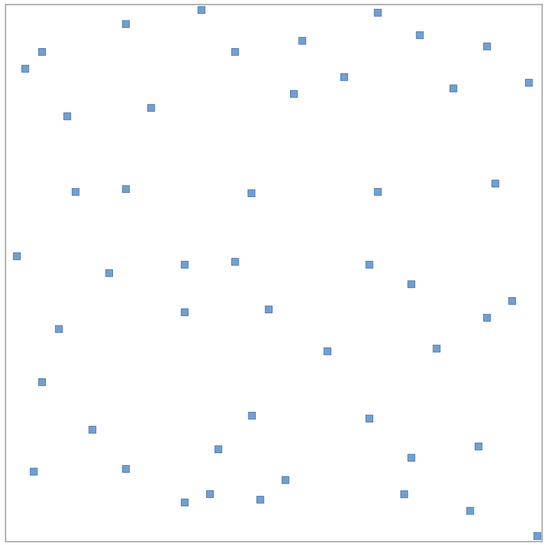




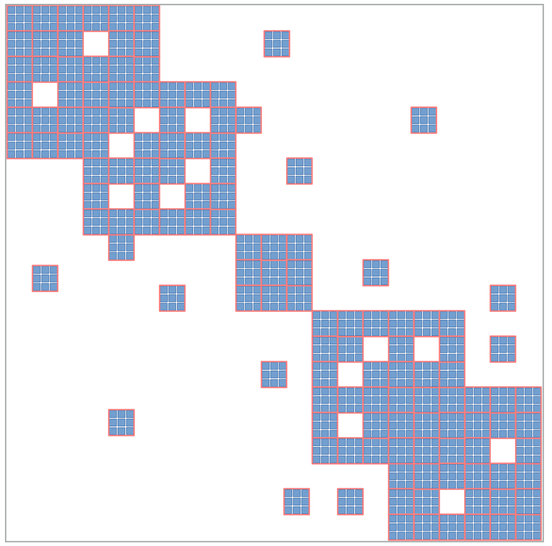
There is no universally superior tensor format



There is no universally superior tensor format

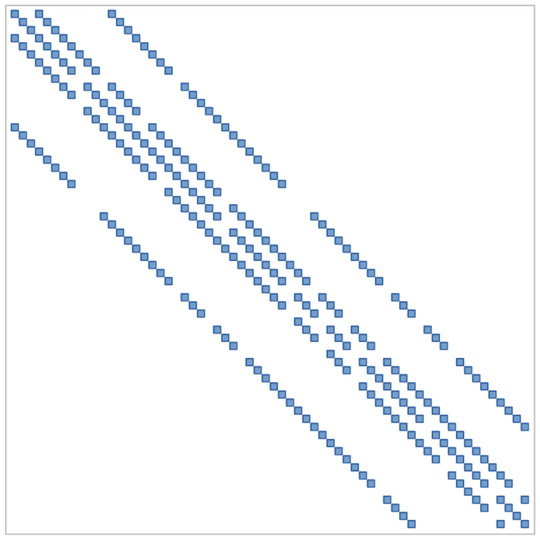


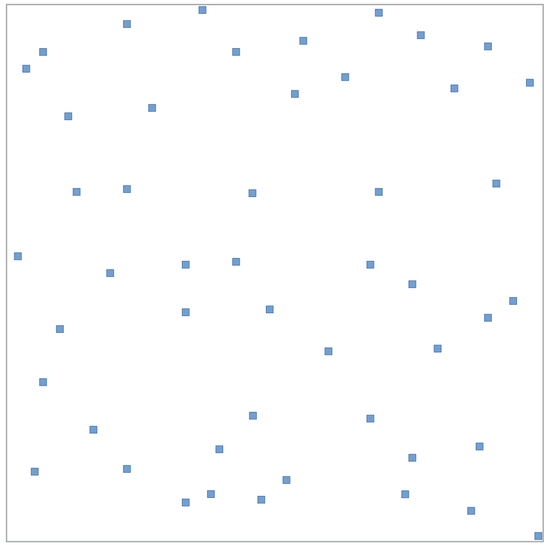
There is no universally superior tensor format

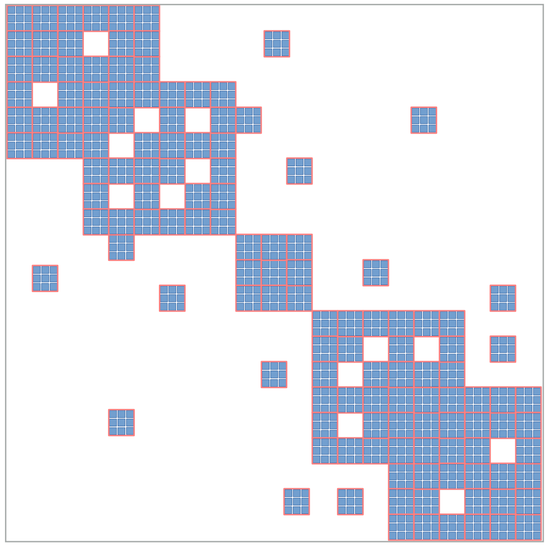


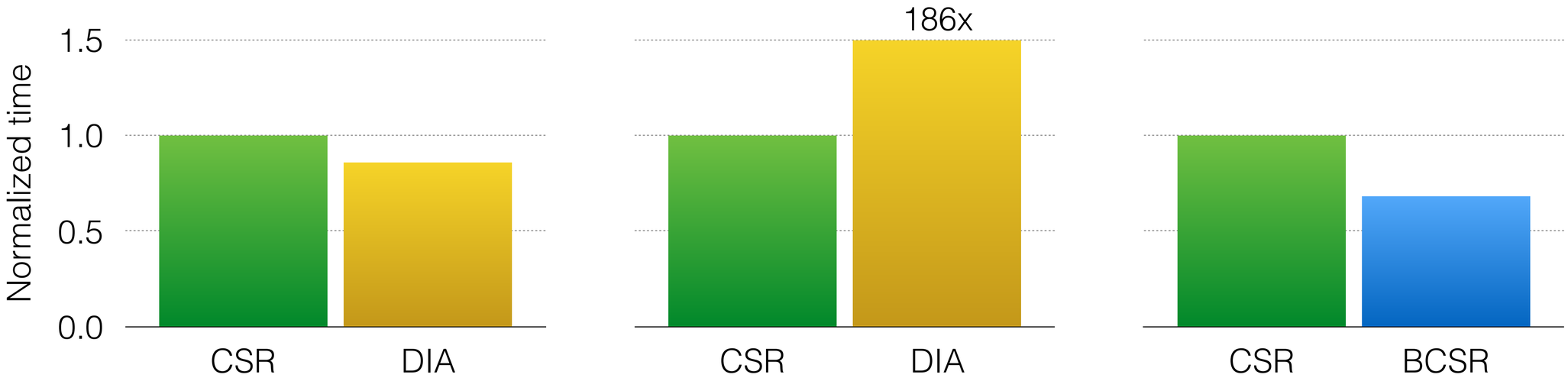
There is no universally superior tensor format

$$y = Ax$$



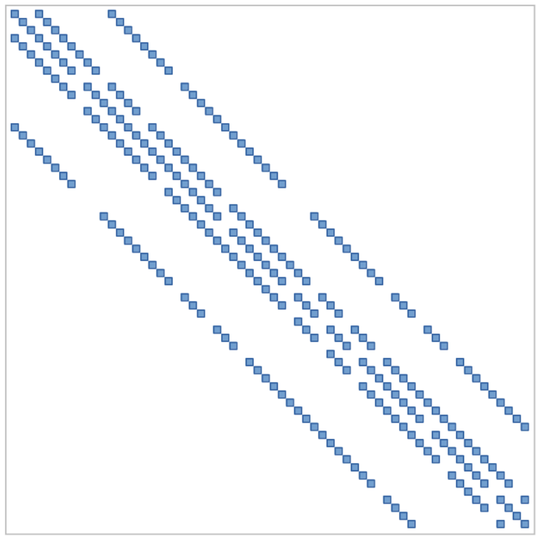


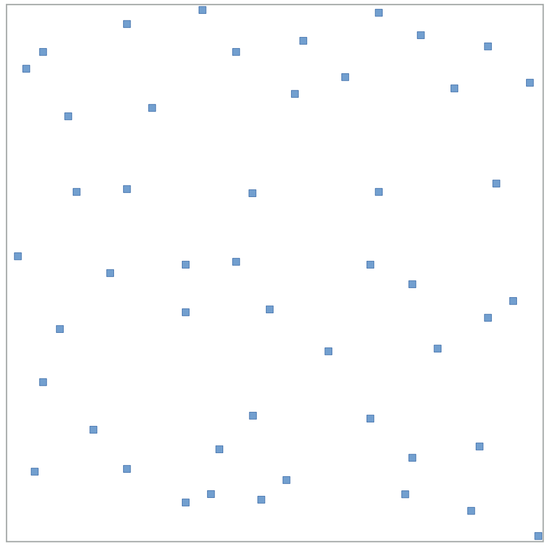


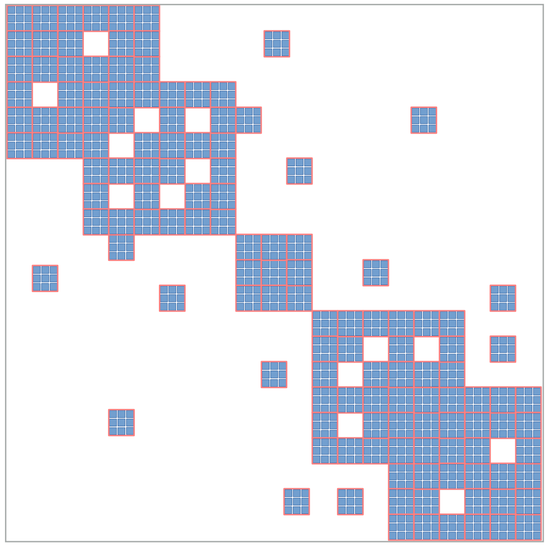


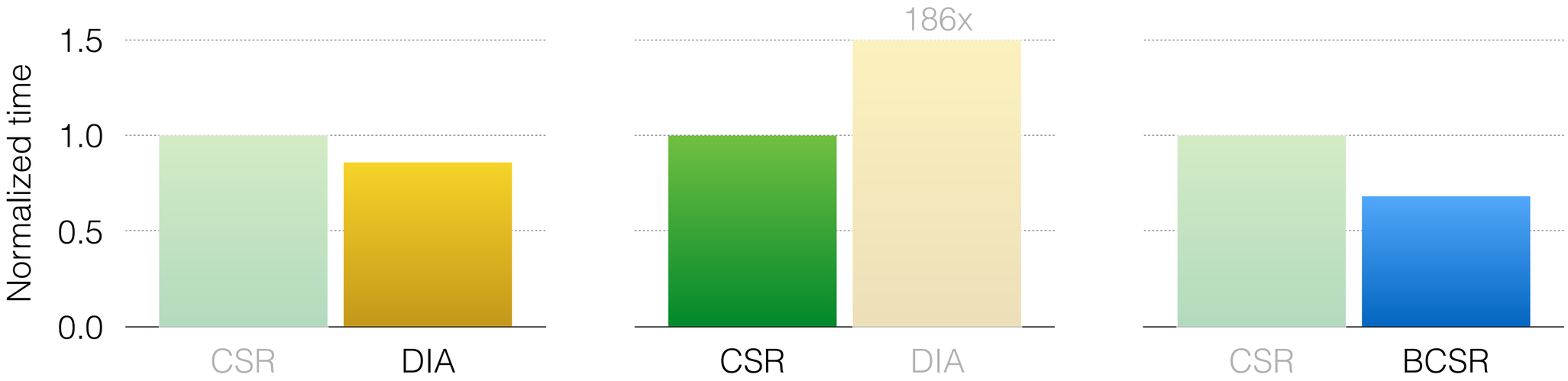
There is no universally superior tensor format

$$y = Ax$$





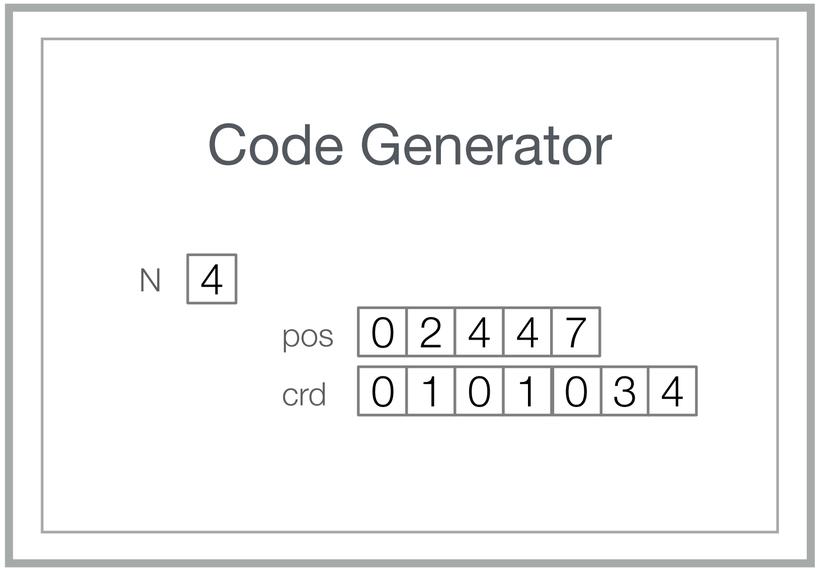




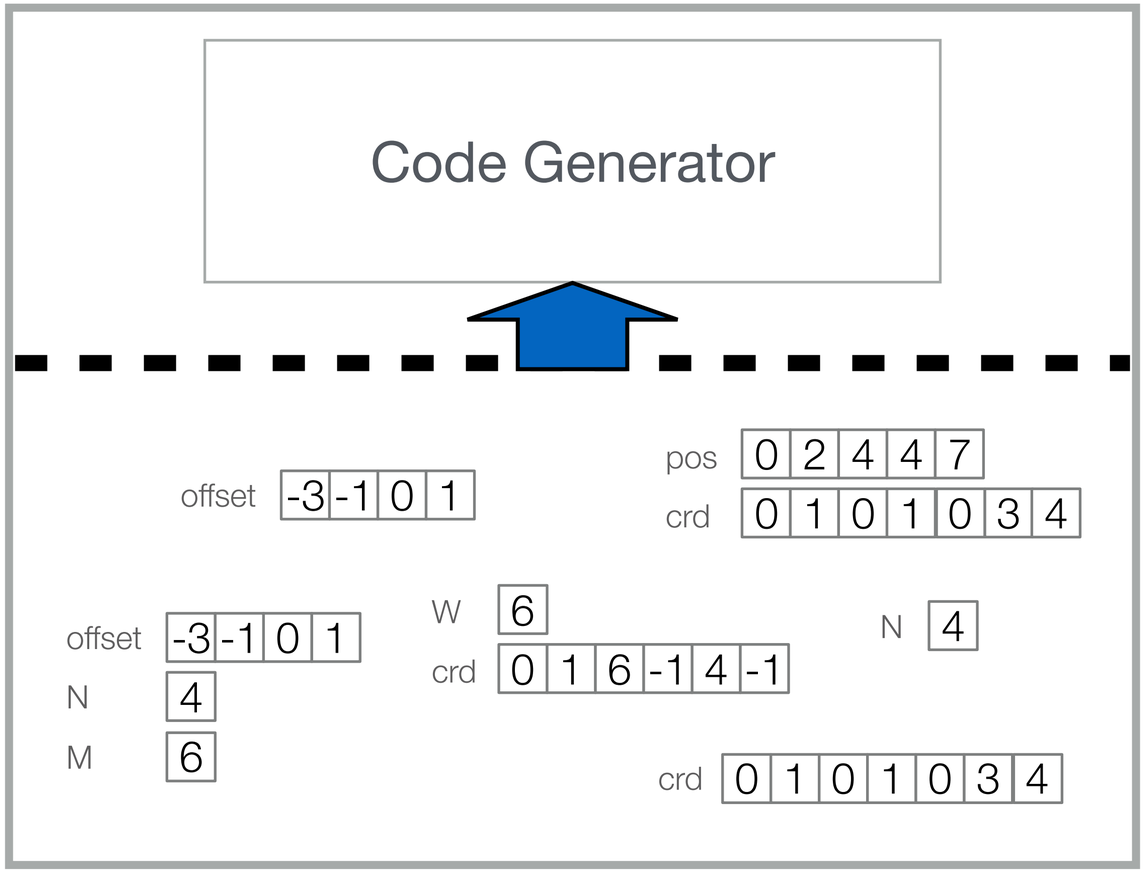
Format Abstraction for Sparse Tensor Algebra Compilers

Stephen Chou, Fredrik Kjolstad, and Saman Amarasinghe

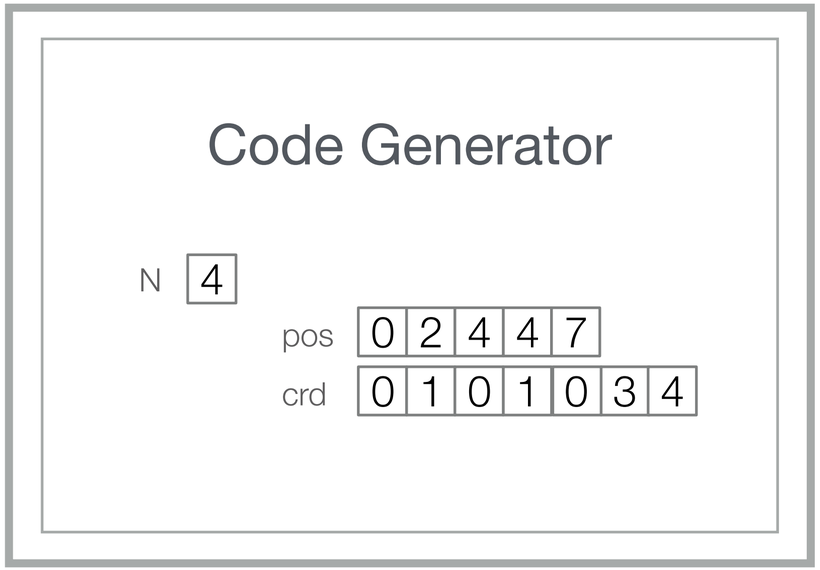
[Kjolstad et al. 2017]



This work

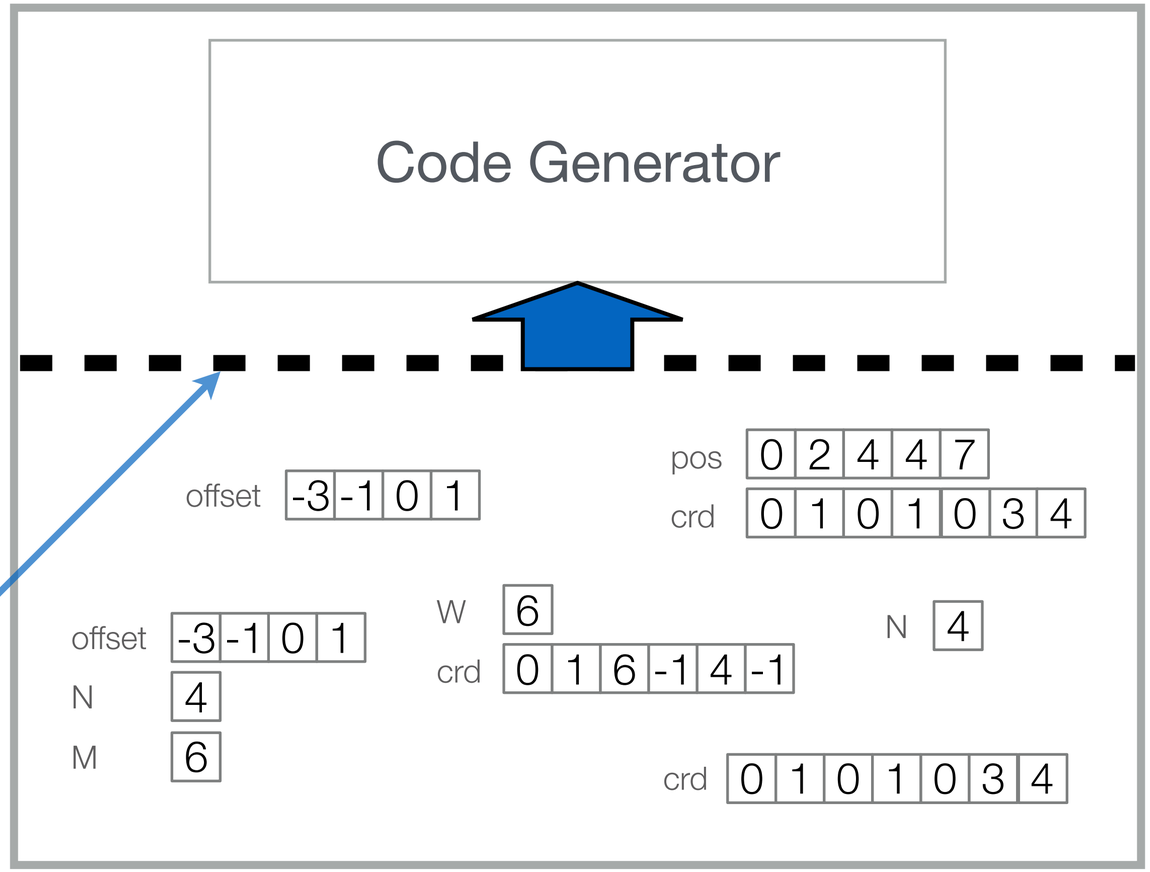


[Kjolstad et al. 2017]

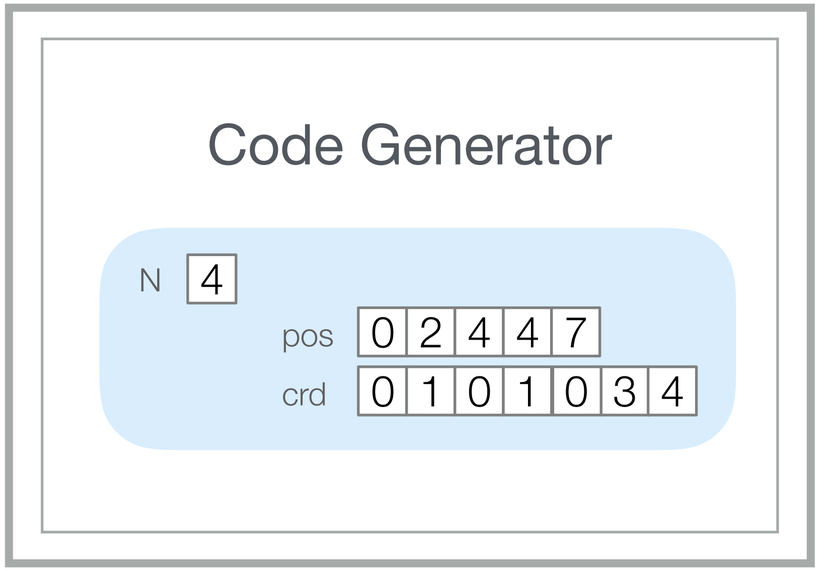


Format abstraction

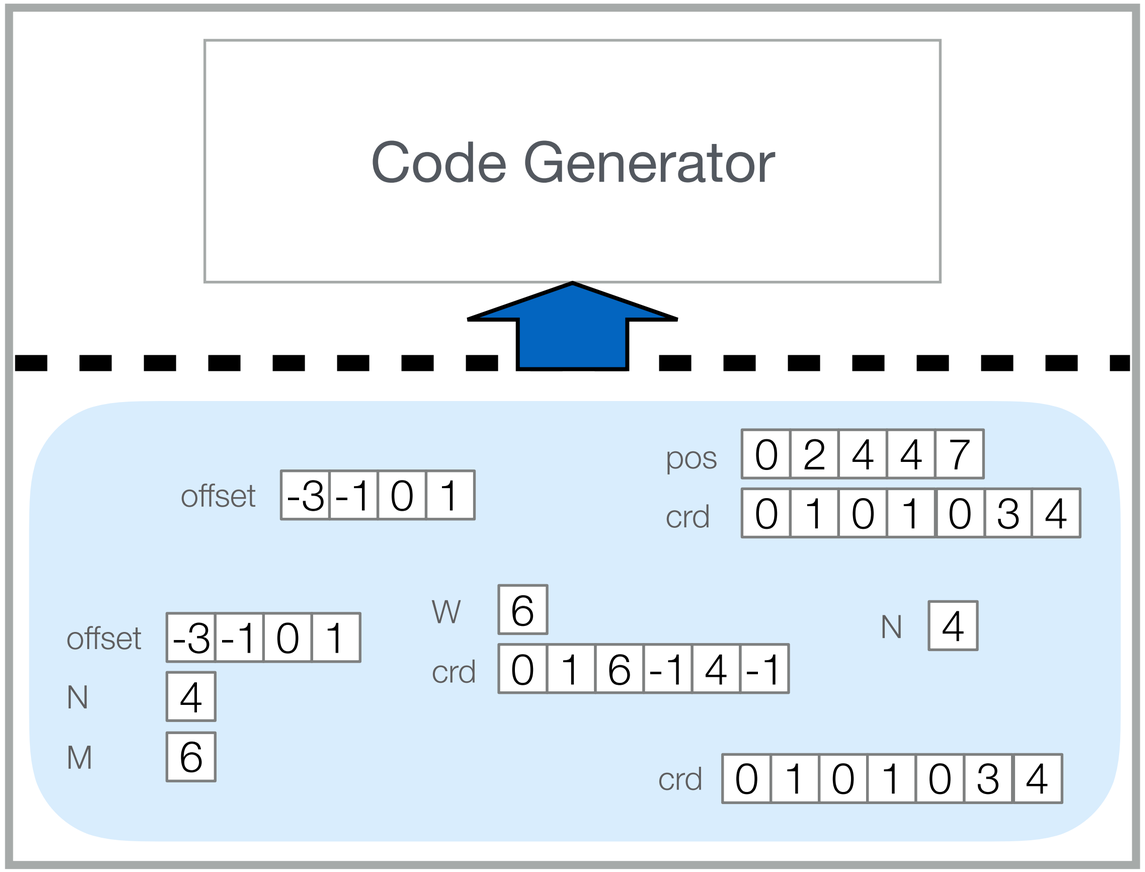
This work



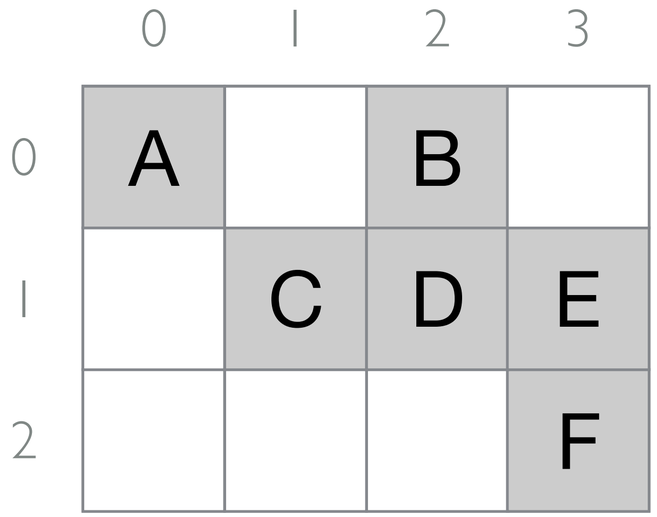
[Kjolstad et al. 2017]



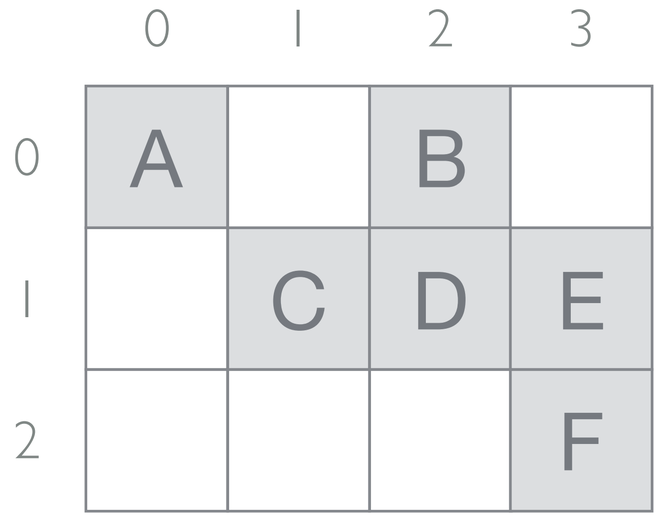
This work



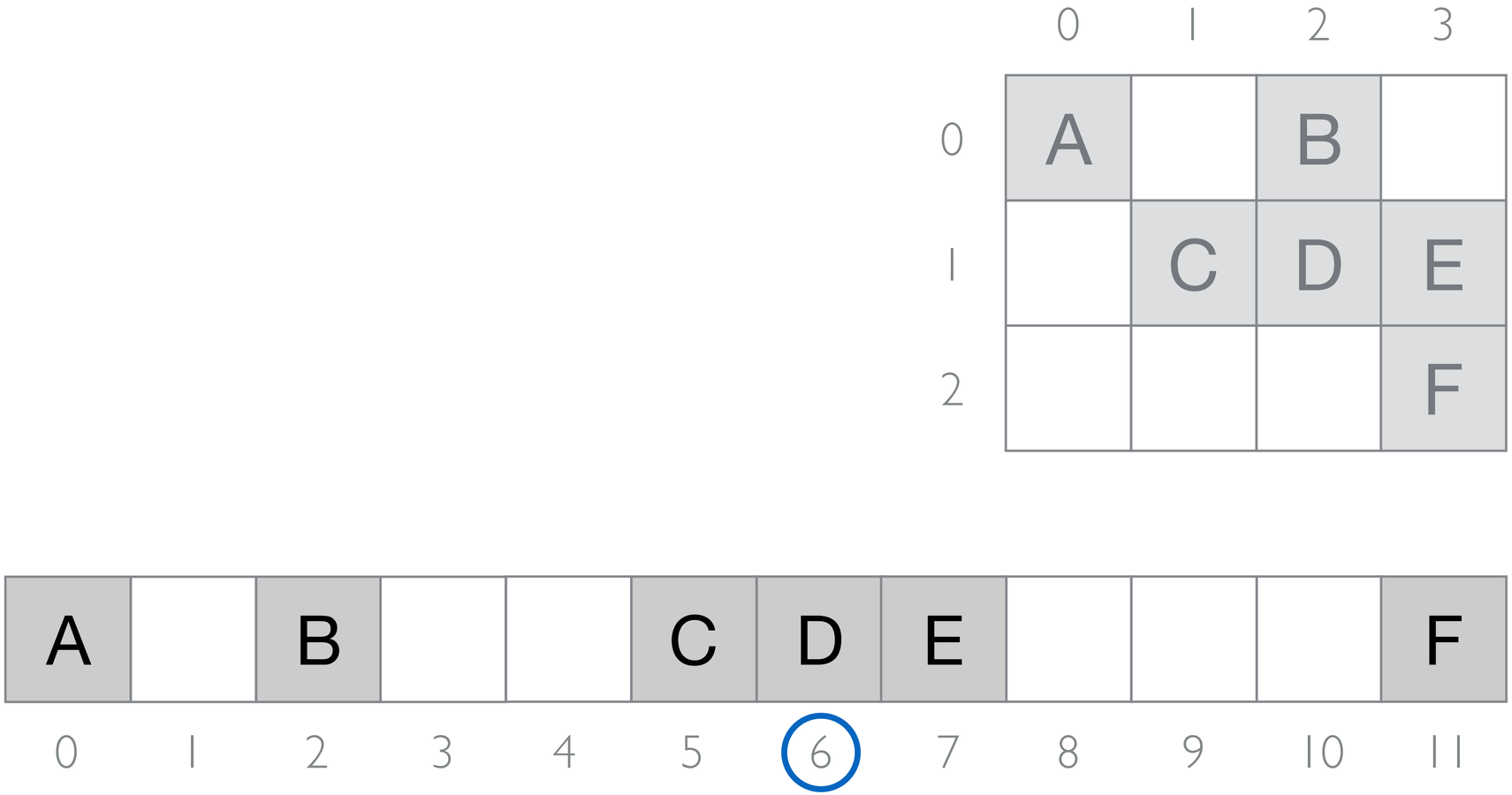
Storing sparse tensors efficiently requires additional metadata



Storing sparse tensors efficiently requires additional metadata

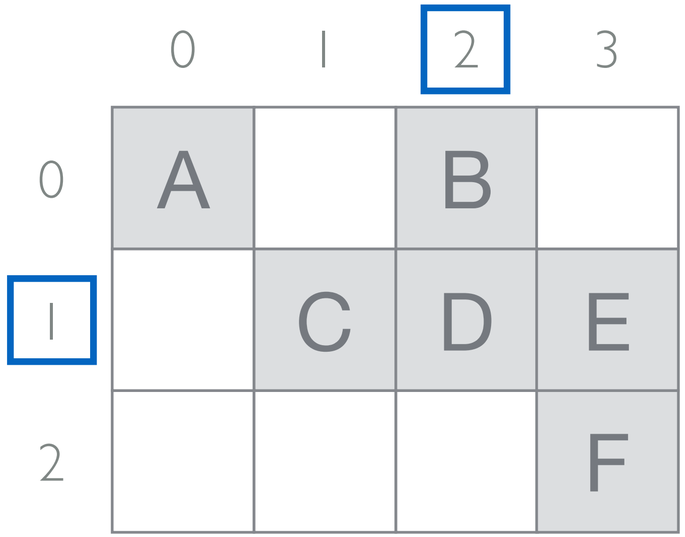


Storing sparse tensors efficiently requires additional metadata

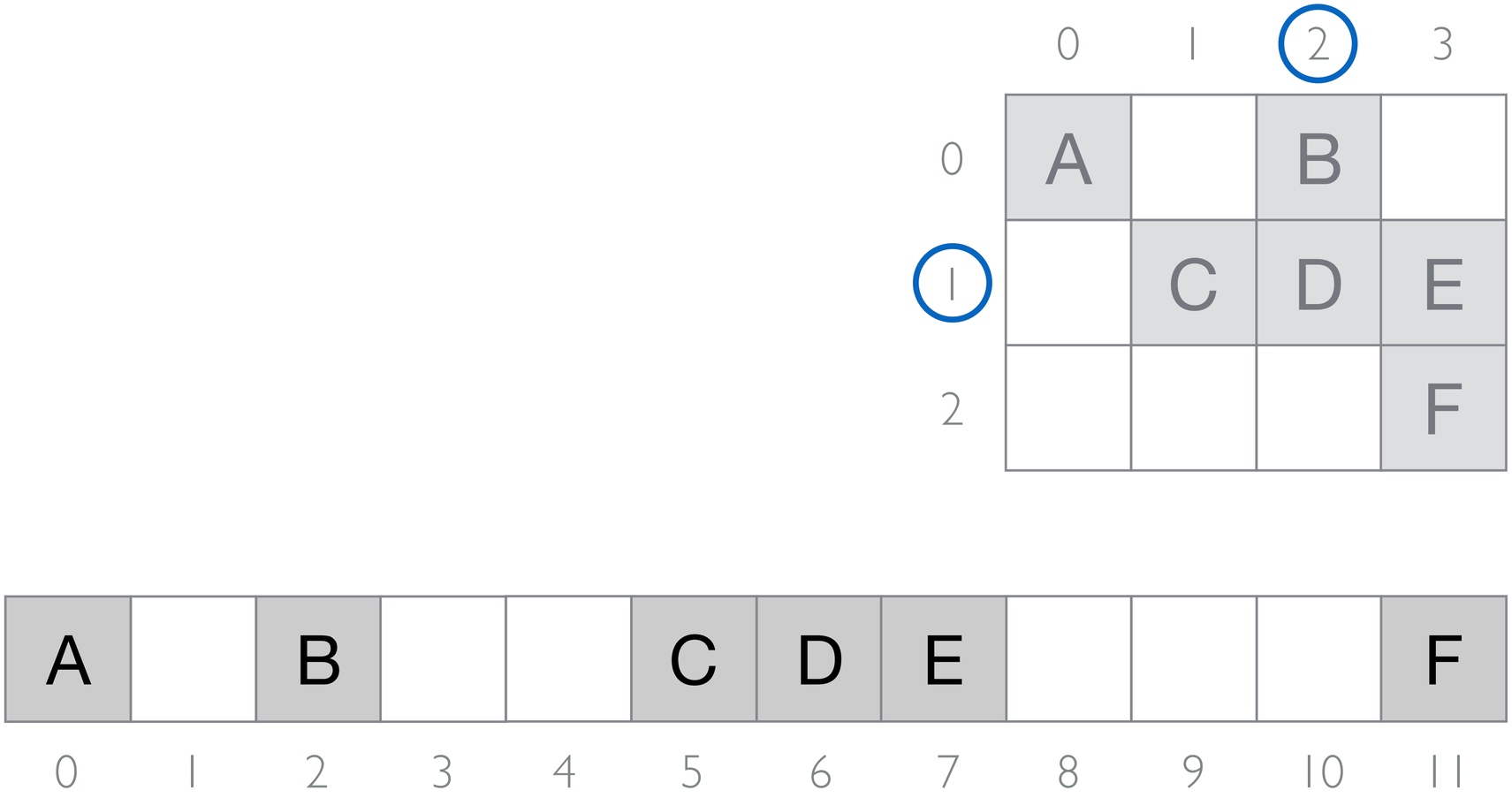


Storing sparse tensors efficiently requires additional metadata

```
row(6) = 6 / 4 = 1
col(6) = 6 % 4 = 2
```

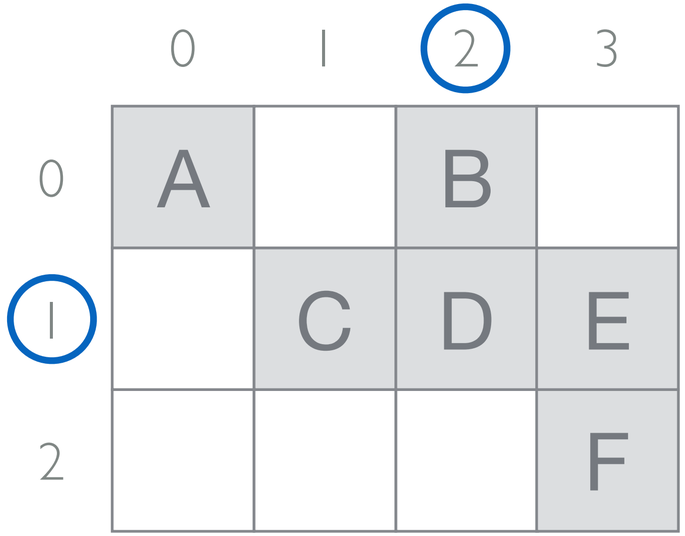


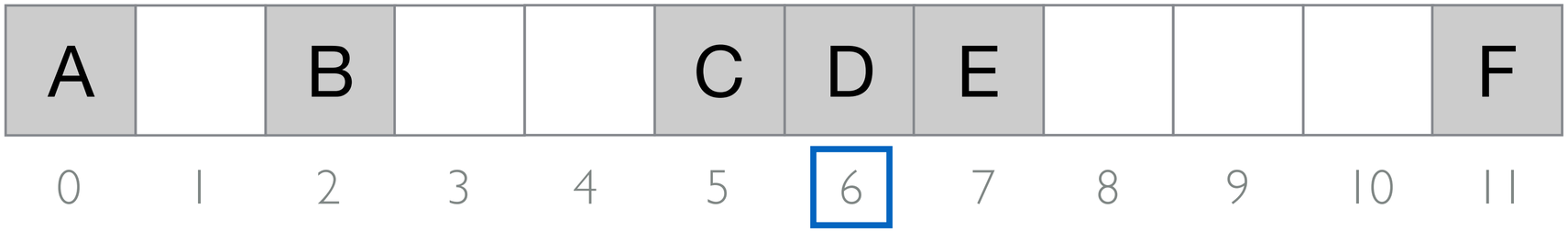
Storing sparse tensors efficiently requires additional metadata



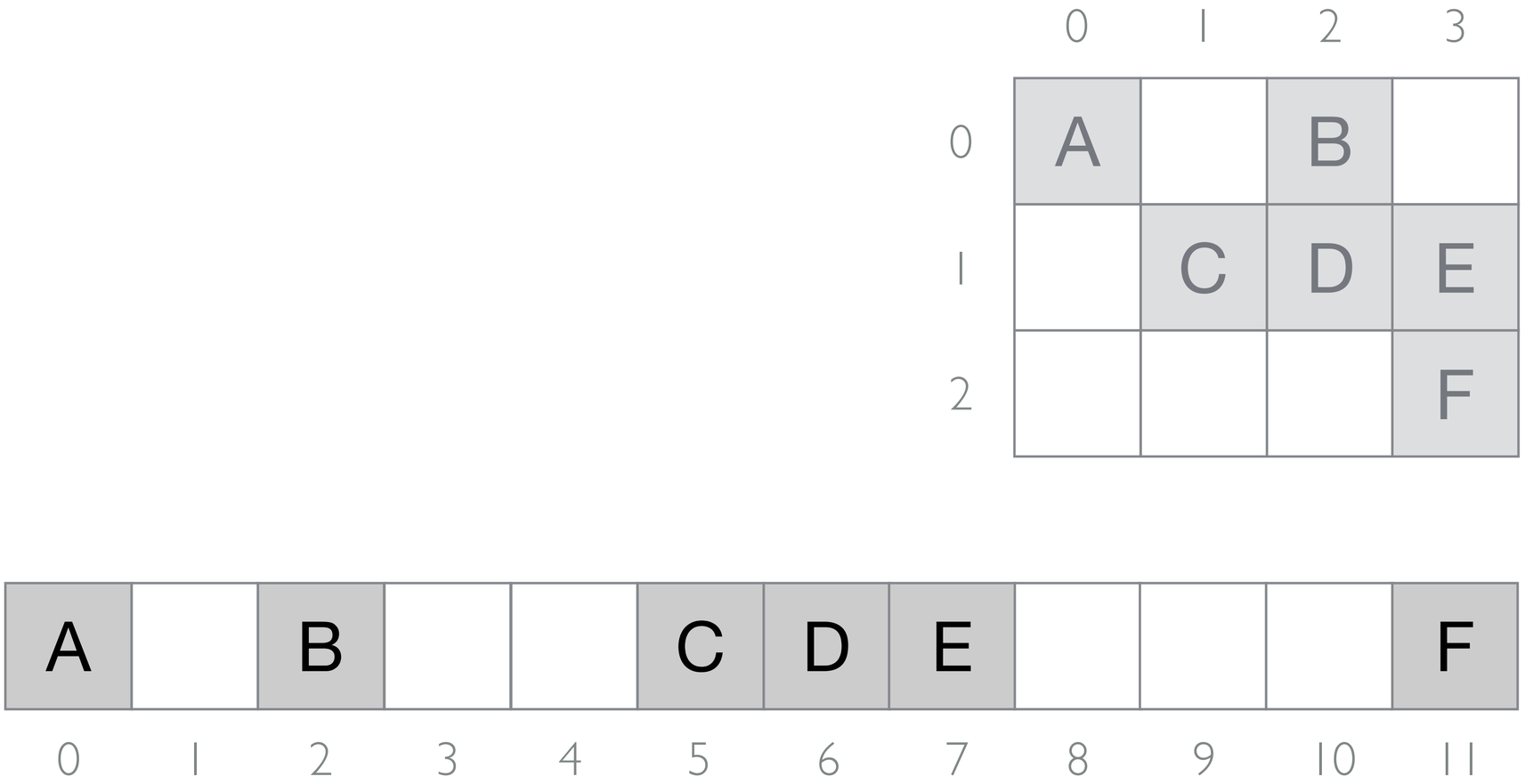
Storing sparse tensors efficiently requires additional metadata

```
locate(1,2) = 1 * 4 + 2
            = 6
```

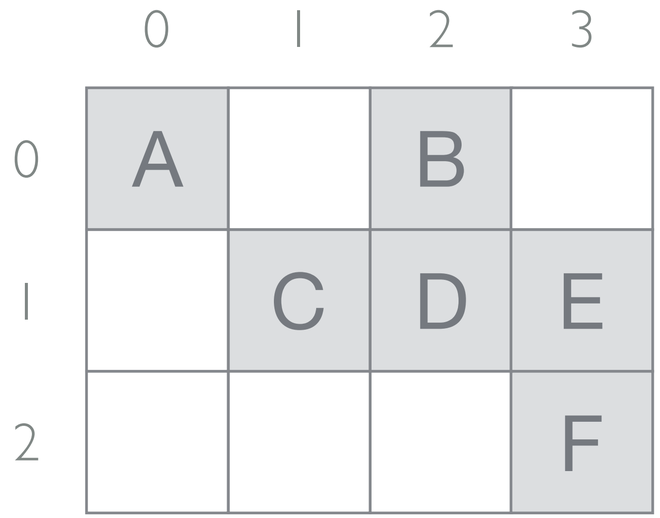


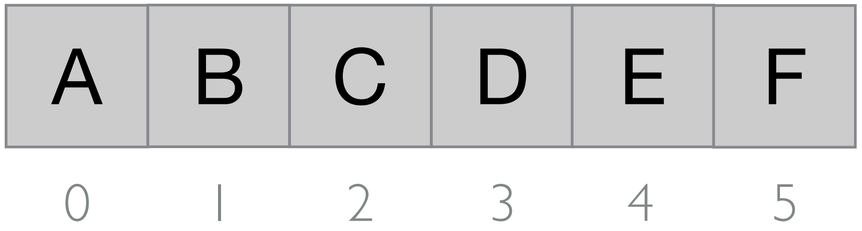


Storing sparse tensors efficiently requires additional metadata



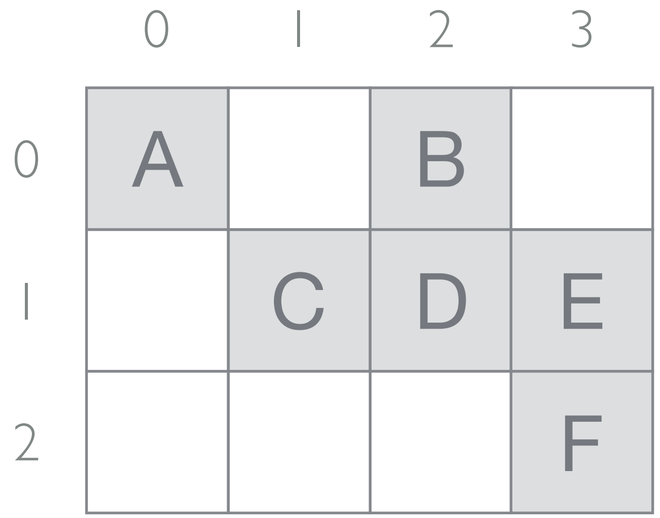
Storing sparse tensors efficiently requires additional metadata



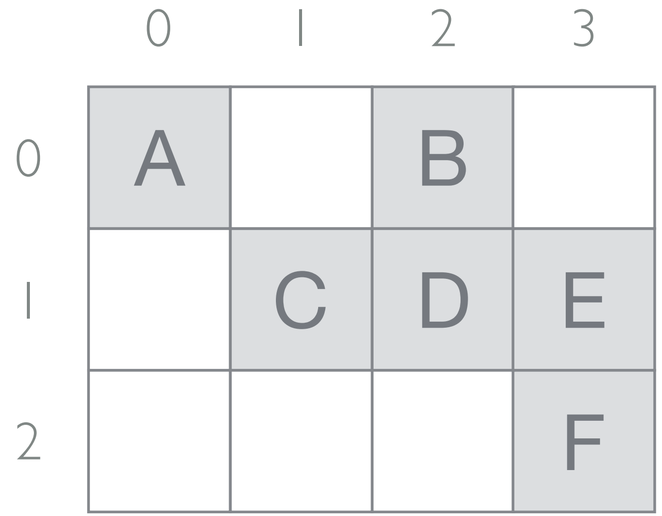


Storing sparse tensors efficiently requires additional metadata

```
row(3) = ???
col(3) = ???
```



Coordinates of tensor elements can be encoded in many ways

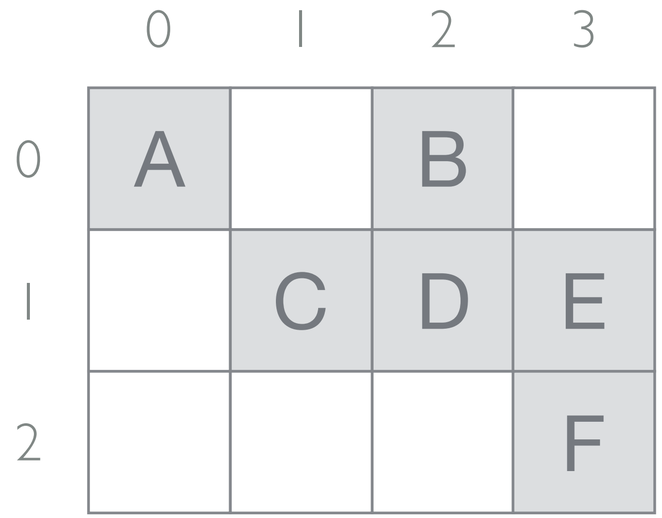


Coordinates of tensor elements can be encoded in many ways

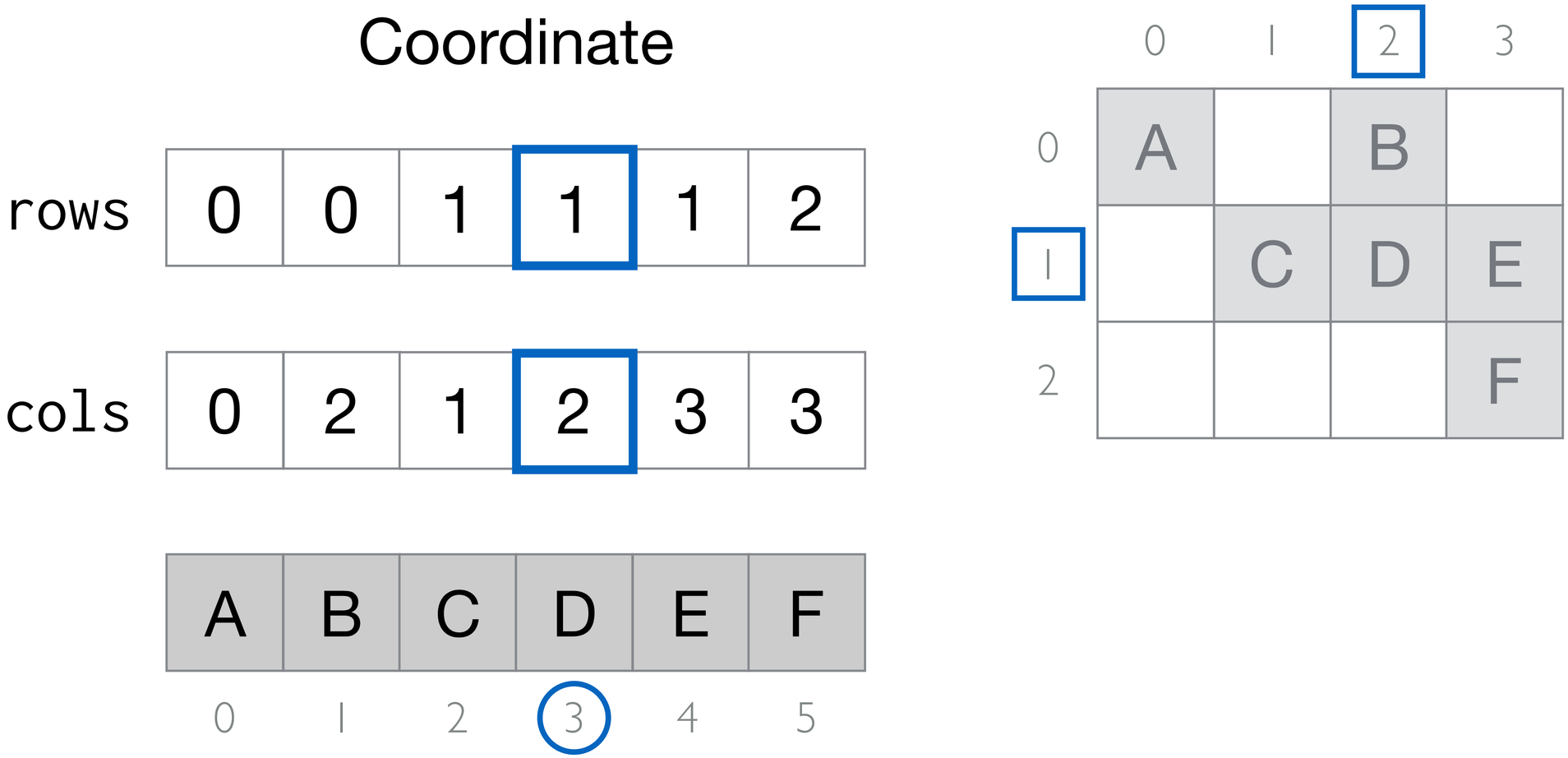
Coordinate

`rows`	0	0	1	1	1	2
`cols`	0	2	1	2	3	3

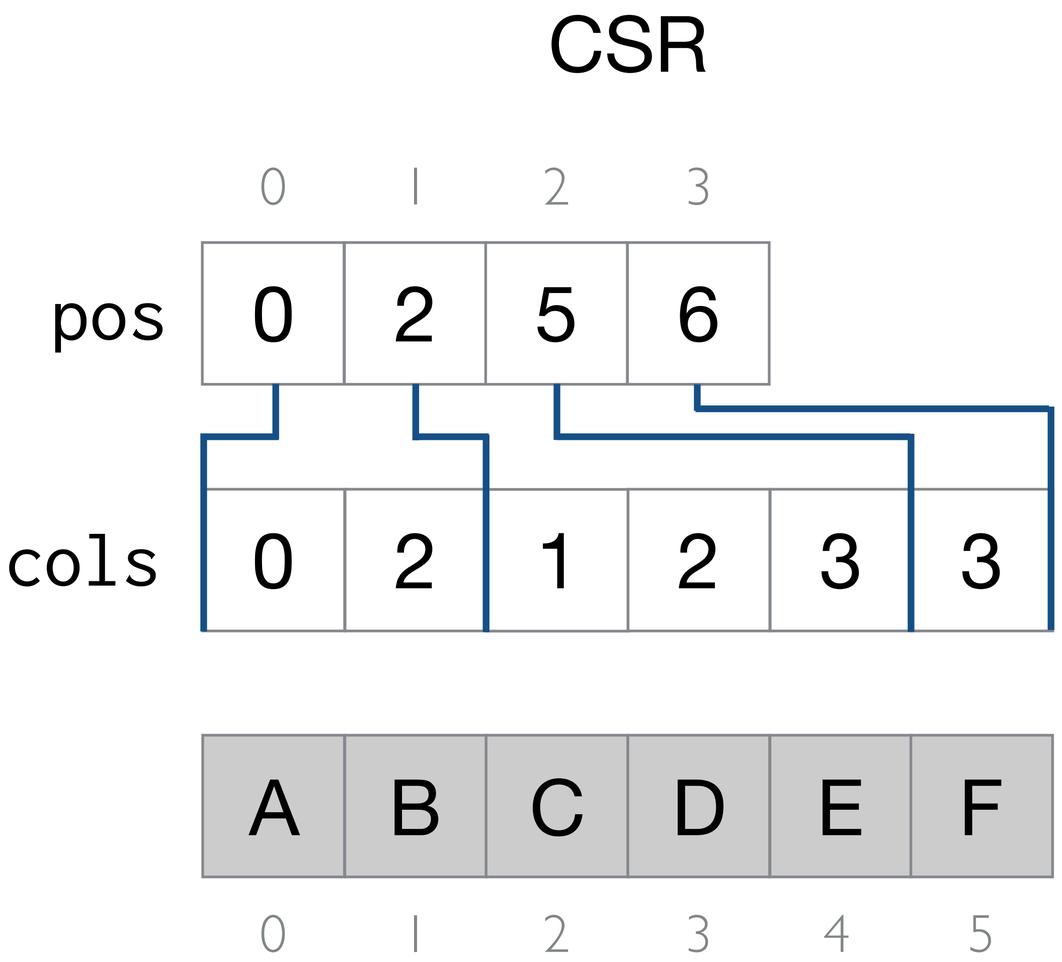
A	B	C	D	E	F
0	1	2	3	4	5

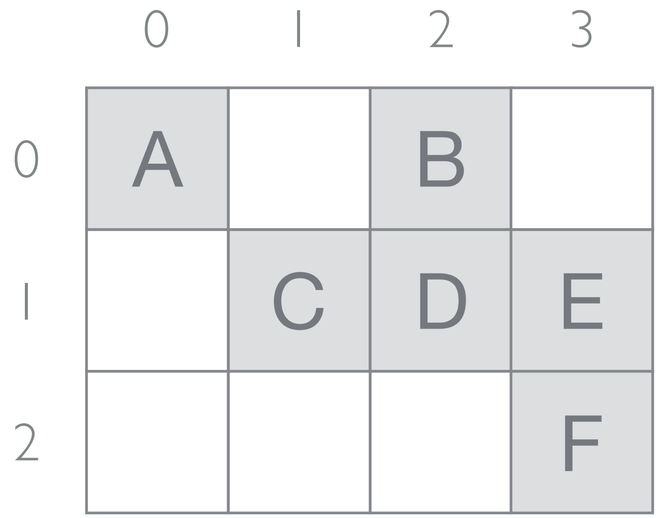


Coordinates of tensor elements can be encoded in many ways

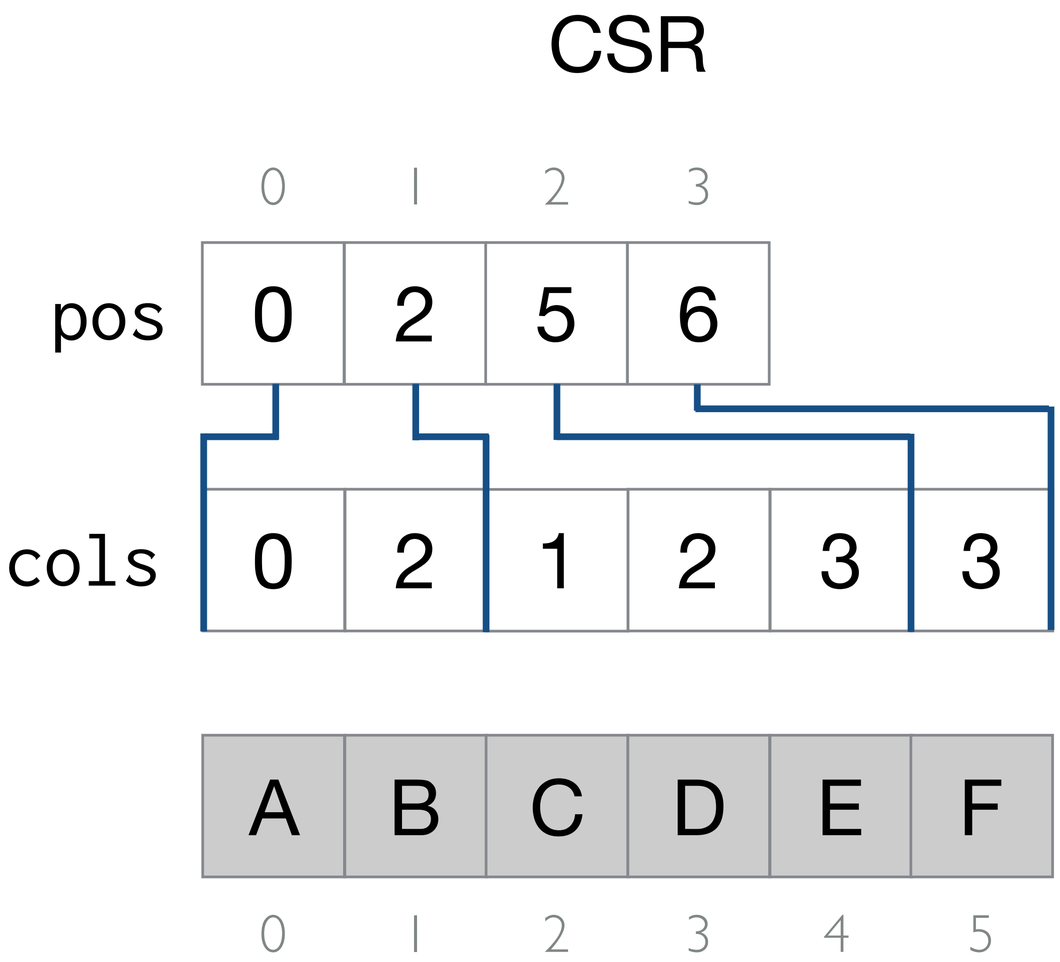


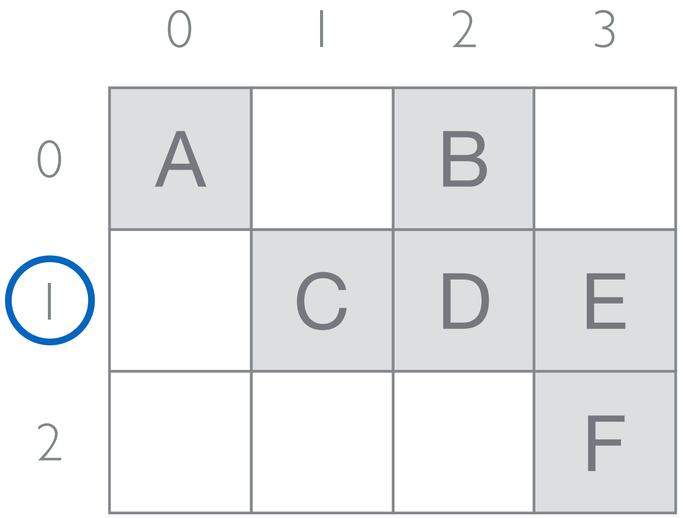
Coordinates of tensor elements can be encoded in many ways



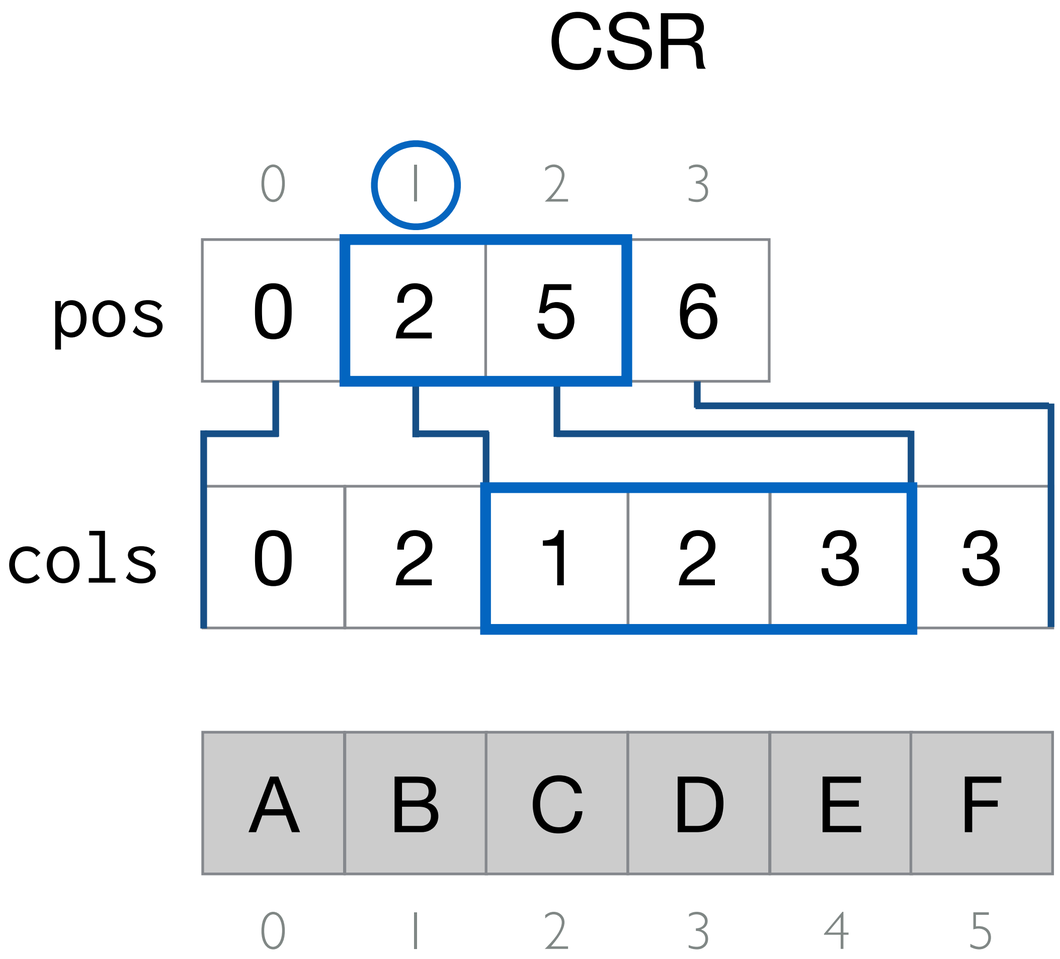


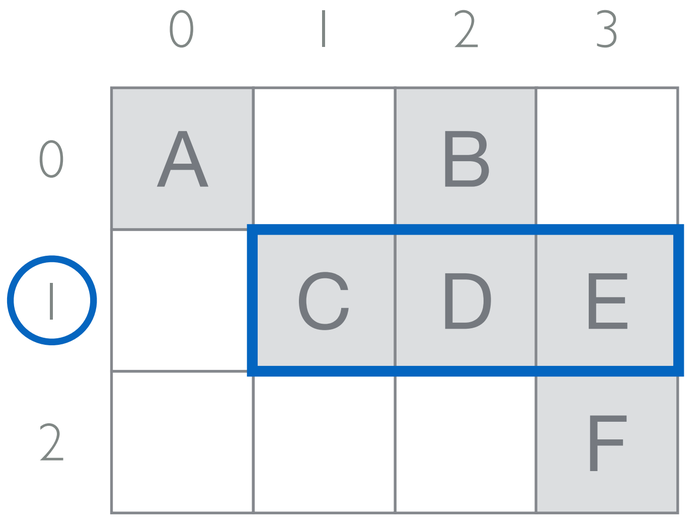
Coordinates of tensor elements can be encoded in many ways



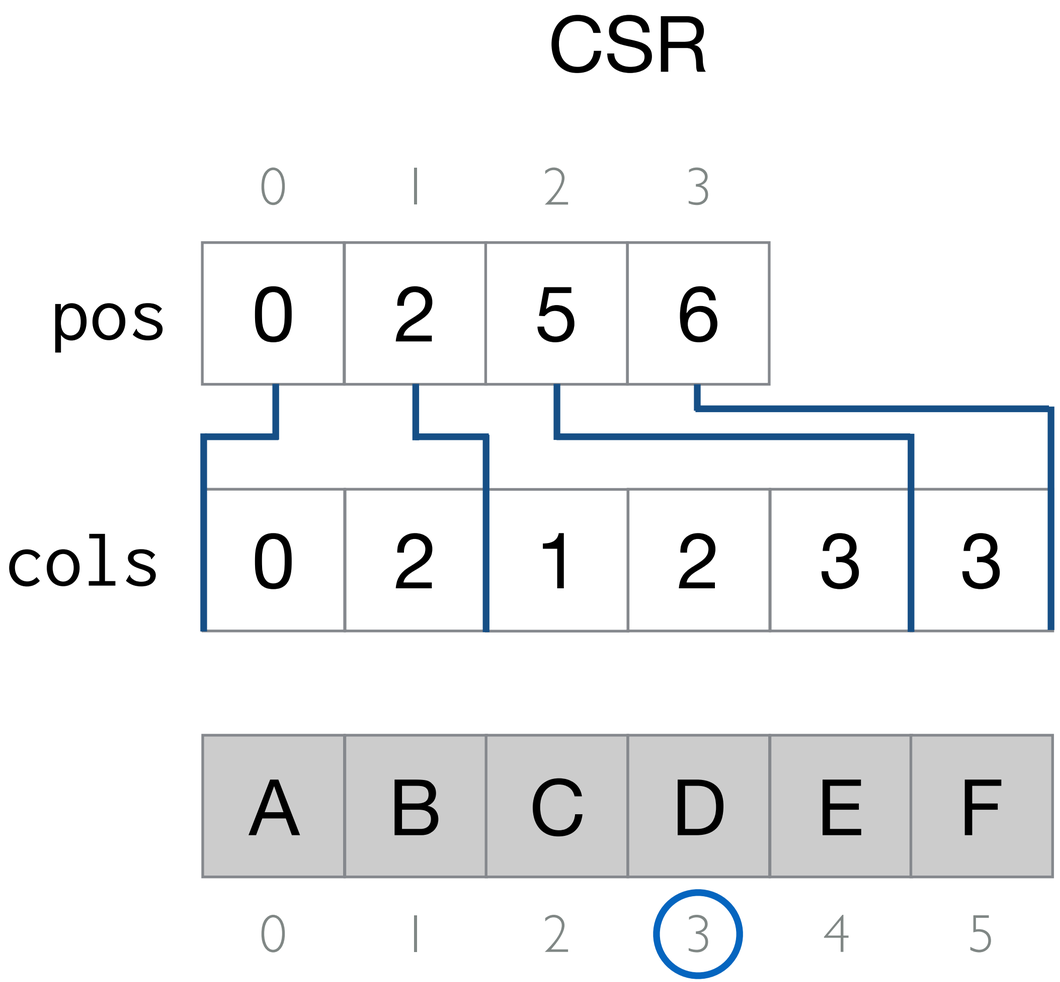


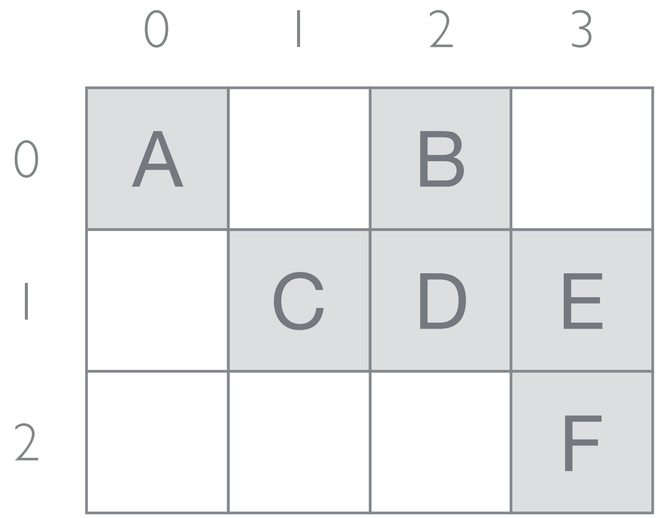
Coordinates of tensor elements can be encoded in many ways



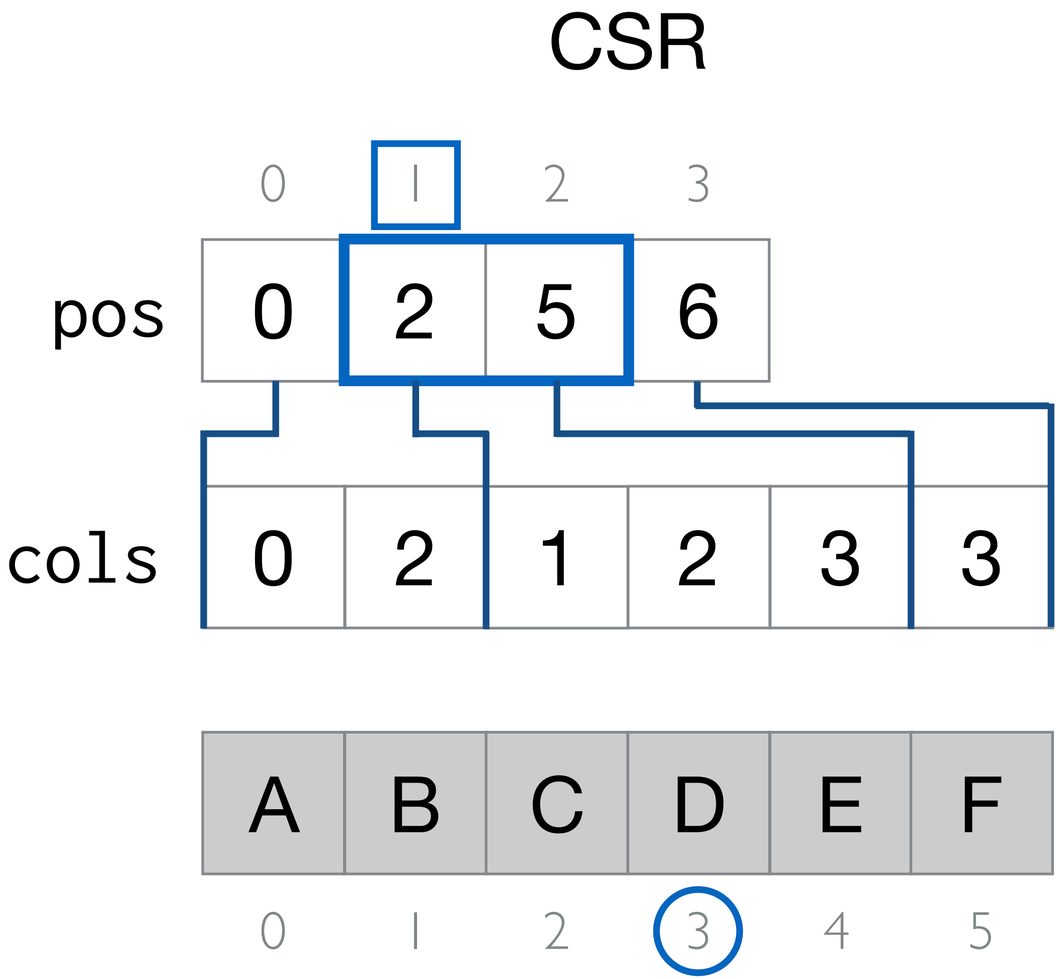


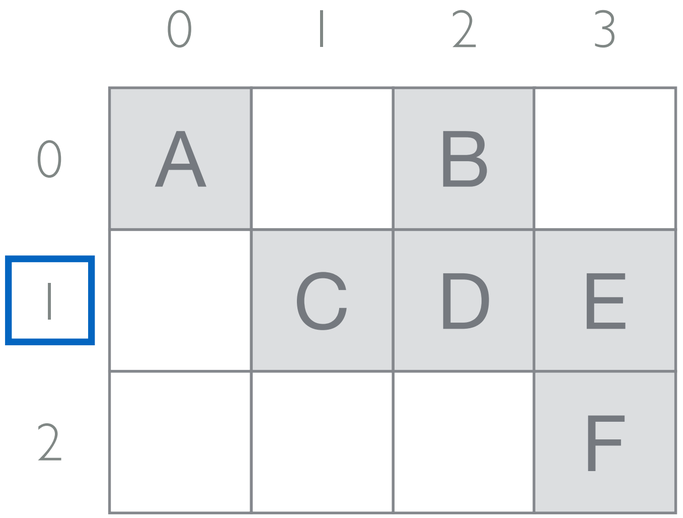
Coordinates of tensor elements can be encoded in many ways





Coordinates of tensor elements can be encoded in many ways





Computing with different formats can require very different code

$$A = B \circ C$$

Computing with different formats can require very different code

$$A = B \circ C$$

Coordinate × Dense array

```
for (int pB = B1_pos[0];
         pB < B1_pos[1];
         pB++) {
  int i = B1_crd[pB];
  int j = B2_crd[pB];
  int pC = i * N + j;
  int pA = i * N + j;
  A[pA] = B[pB] * C[pC];
}
```

Computing with different formats can require very different code

$$A = B \circ C$$

Coordinate × Dense array

```
for (int pB = B1_pos[0];
         pB < B1_pos[1];
         pB++) {
  int i = B1_crd[pB];
  int j = B2_crd[pB];
  int pC = i * N + j;
  int pA = i * N + j;
  A[pA] = B[pB] * C[pC];
}
```

CSR × Dense array

```
for (int i = 0;
         i < M;
         i++) {
  for (int pB = B2_pos[i];
           pB < B2_pos[i + 1];
           pB++) {
    int j = B2_crd[pB];
    int pC = i * N + j;
    int pA = i * N + j;
    A[pA] = B[pB] * C[pC];
  }
}
```

Computing with different formats can require very different code

$$A = B \circ C$$

Coordinate × Dense array

```
for (int pB = B1_pos[0];
         pB < B1_pos[1];
         pB++) {
  int i = B1_crd[pB];
  int j = B2_crd[pB];
  int pC = i * N + j;
  int pA = i * N + j;
  A[pA] = B[pB] * C[pC];
}
```

CSR × Dense array

```
for (int i = 0;
         i < M;
         i++) {
  for (int pB = B2_pos[i];
           pB < B2_pos[i + 1];
           pB++) {
    int j = B2_crd[pB];
    int pC = i * N + j;
    int pA = i * N + j;
    A[pA] = B[pB] * C[pC];
  }
}
```

CSR × Coordinate

```
int pC1 = C1_pos[0];
while (pC1 < C1_pos[1]) {
  int i = C1_crd[pC1];
  int C1_segend = pC1 + 1;
  while (C1_segend < C1_pos[1] &&
         C1_crd[C1_segend] == i)
    C1_segend++;
  int pB2 = B2_pos[i];
  int pC2 = pC1;
  while (pB2 < B2_pos[i + 1] &&
         pC2 < C1_segend) {
    int jB2 = B2_crd[pB2];
    int jC2 = C2_crd[pC2];
    int j = min(jB2, jC2);
    int pA = i * N + j;
    if (jB2 == j && jC2 == j)
      A[pA] = B[pB2] * C[pC2];
    if (jB2 == j) pB2++;
    if (jC2 == j) pC2++;
  }
  pC1 = C1_segend;
}
```

Hand-coding support for a wide range of formats is infeasible

$$A = B \circ C$$

Coordinate × Dense array

CSR × Dense array

CSR × Coordinate

Hand-coding support for a wide range of formats is infeasible

$$A = B \circ C$$

Coordinate × Dense array

CSR × Dense array

CSR × Coordinate

Hand-coding support for a wide range of formats is infeasible

$$A = B \circ C$$

Coordinate × Dense array

CSR × Dense array

CSR × Coordinate

Dense array × Dense array

Coordinate × Coordinate

CSR × CSR

DIA × DIA

DIA × Dense array

DIA × Coordinate

DIA × CSR

ELLPACK × ELLPACK

ELLPACK × Dense array

ELLPACK × Coordinate

ELLPACK × CSR

ELLPACK × DIA

BCSR × BCSR

BCSR × Dense array

BCSR × Coordinate

Hand-coding support for a wide range of formats is infeasible

$$A = B \circ C \qquad A = B \circ C \circ D$$

Coordinate × Dense array
CSR × Dense array
CSR × Coordinate
Dense array × Dense array
Coordinate × Coordinate
CSR × CSR
DIA × DIA
DIA × Dense array
DIA × Coordinate
DIA × CSR
ELLPACK × ELLPACK
ELLPACK × Dense array
ELLPACK × Coordinate
ELLPACK × CSR
ELLPACK × DIA
BCSR × BCSR
BCSR × Dense array
BCSR × Coordinate

Dense array × CSR × CSR
Coordinate × CSR × CSR
CSR × CSR × CSR
Dense array × Coordinate × CSR
Dense array × Dense array × CSR
Coordinate × Coordinate × CSR
DIA × Coordinate × Dense array
DIA × Coordinate × CSR
DIA × Dense array × CSR
DIA × CSR × CSR
DIA × Coordinate × Coordinate
DIA × Dense array × Dense array
DIA × DIA × CSR
DIA × DIA × Coordinate
DIA × DIA × Dense array
DIA × DIA × DIA
ELLPACK × ELLPACK × DIA
ELLPACK × CSR × DIA
ELLPACK × BCSR × DIA

Hand-coding support for a wide range of formats is infeasible

$A = B \circ C$

Coordinate × Dense array
CSR × Dense array
CSR × Coordinate
Dense array × Dense array
Coordinate × Coordinate
CSR × CSR
DIA × DIA
DIA × Dense array
DIA × Coordinate
DIA × CSR
ELLPACK × ELLPACK
ELLPACK × Dense array
ELLPACK × Coordinate
ELLPACK × CSR
ELLPACK × DIA
BCSR × BCSR
BCSR × Dense array
BCSR × Coordinate

$A = B \circ C \circ D$

Dense array × CSR × CSR
Coordinate × CSR × CSR
CSR × CSR × CSR
Dense array × Coordinate × CSR
Dense array × Dense array × CSR
Coordinate × Coordinate × CSR
DIA × Coordinate × Dense array
DIA × Coordinate × CSR
DIA × Dense array × CSR
DIA × CSR × CSR
DIA × Coordinate × Coordinate
DIA × Dense array × Dense array
DIA × DIA × CSR
DIA × DIA × Coordinate
DIA × DIA × Dense array
DIA × DIA × DIA
ELLPACK × ELLPACK × DIA
ELLPACK × CSR × DIA
ELLPACK × BCSR × DIA

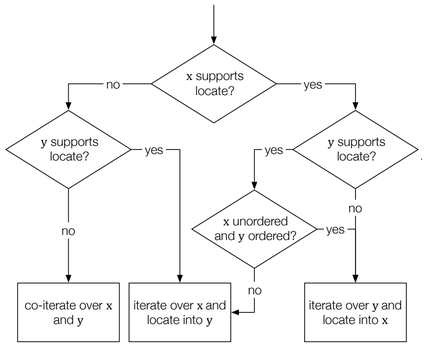
$y = Ax + z$

Dense array × Dense array × Dense array
Dense array × Dense array × Sparse vector
Dense array × Dense array × Hash map
Dense array × Sparse vector × Sparse vector
Dense array × Sparse vector × Hash map
Dense array × Hash map × Sparse vector
Dense array × Sparse vector × Dense array
Coordinate × Dense array × Dense array
Coordinate × Sparse vector × Dense array
Coordinate × Dense array × Hash map
Coordinate × Sparse vector × Hash map
Coordinate × Hash map × Sparse vector
CSR × Dense array × Dense array
CSR × Dense array × Sparse vector
CSR × Hash map × Sparse vector
CSR × Hash map × Dense array
CSR × Sparse vector × Dense array
DIA × Dense array × Dense array
DIA × Hash map × Dense array
ELLPACK × Dense array × Sparse vector

Format Abstraction & Code Generation

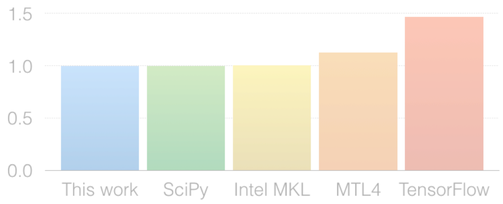
DIA
Dense
Range
Offset

Mode-generic tensor
Compressed
Singleton
Dense
Dense



Evaluation

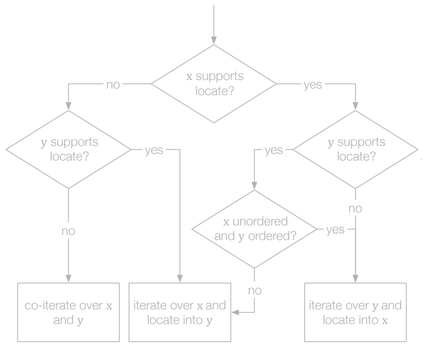
	This work	Kjolstad et al. 2017	Intel MKL	SciPy	MTL4	MATLAB Tensor Toolbox	TensorFlow
Sparse vector	✓	✓	✓	✓	✓	✓	✓
Hash map vector	✓			✓			
Coordinate matrix	✓		✓	✓	✓	✓	✓
CSR	✓	✓	✓	✓	✓	✓	
DCSR	✓	✓					
ELL	✓				✓		
DIA	✓		✓	✓			
BCSR	✓	✓	✓	✓	✓		
CSB	✓						
DOK				✓			
LIL				✓			
Skyline	✓		✓				
Banded	✓				✓		
Coordinate tensor	✓					✓	✓
CSF	✓	✓					
Mode-generic	✓						



Format Abstraction & Code Generation

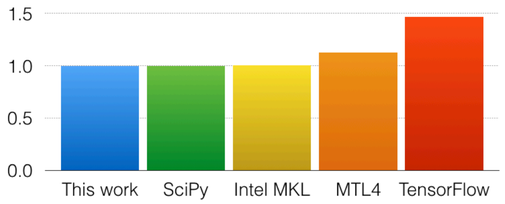
DIA
Dense
Range
Offset

Mode-generic tensor
Compressed
Singleton
Dense
Dense



Evaluation

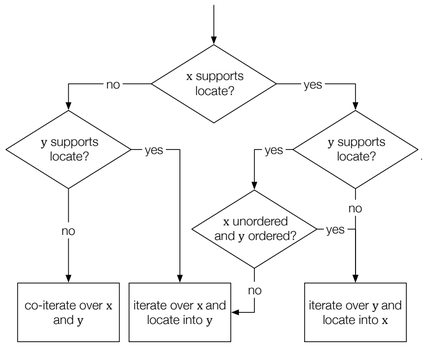
	This work	Kjolstad et al. 2017	Intel MKL	SciPy	MTL4	MATLAB Tensor Toolbox	TensorFlow
Sparse vector	✔	✔	✔	✔	✔	✔	✔
Hash map vector	✔			✔			
Coordinate matrix	✔		✔	✔	✔	✔	✔
CSR	✔	✔	✔	✔	✔	✔	
DCSR	✔	✔					
ELL	✔				✔		
DIA	✔		✔	✔			
BCSR	✔	✔	✔	✔	✔		
CSB	✔						
DOK				✔			
LIL				✔			
Skyline	✔		✔				
Banded	✔				✔		
Coordinate tensor	✔					✔	✔
CSF	✔	✔					
Mode-generic	✔						



Format Abstraction & Code Generation

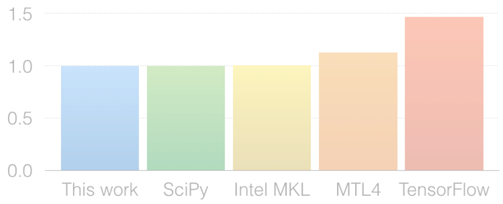
DIA
Dense
Range
Offset

Mode-generic tensor
Compressed
Singleton
Dense
Dense

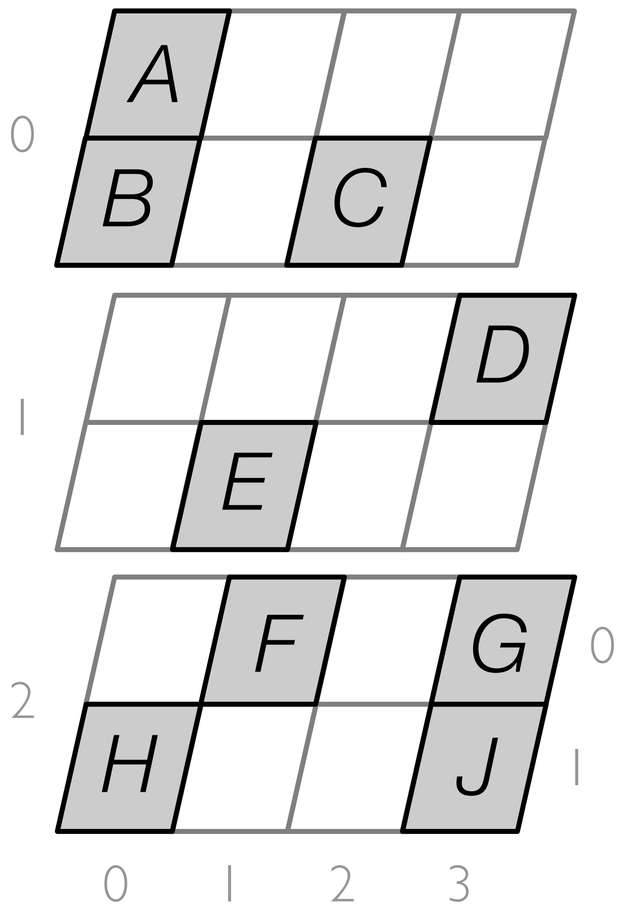


Evaluation

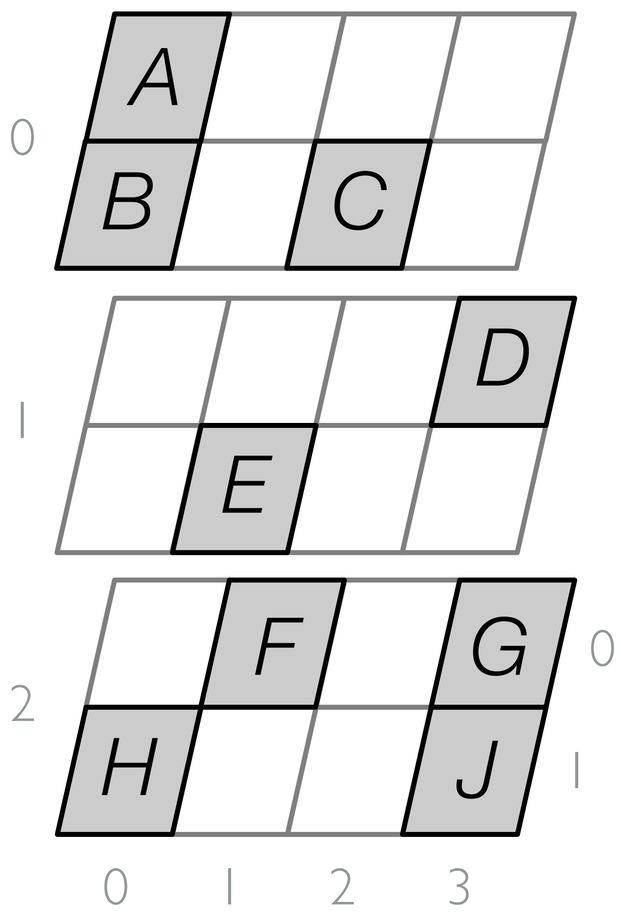
	This work	Kjolstad et al. 2017	Intel MKL	SciPy	MTL4	MATLAB Tensor Toolbox	TensorFlow
Sparse vector	✓	✓	✓	✓	✓	✓	✓
Hash map vector	✓			✓			
Coordinate matrix	✓		✓	✓	✓	✓	✓
CSR	✓	✓	✓	✓	✓	✓	
DCSR	✓	✓					
ELL	✓				✓		
DIA	✓		✓	✓			
BCSR	✓	✓	✓	✓	✓		
CSB	✓						
DOK				✓			
LIL				✓			
Skyline	✓		✓				
Banded	✓				✓		
Coordinate tensor	✓					✓	✓
CSF	✓	✓					
Mode-generic	✓						



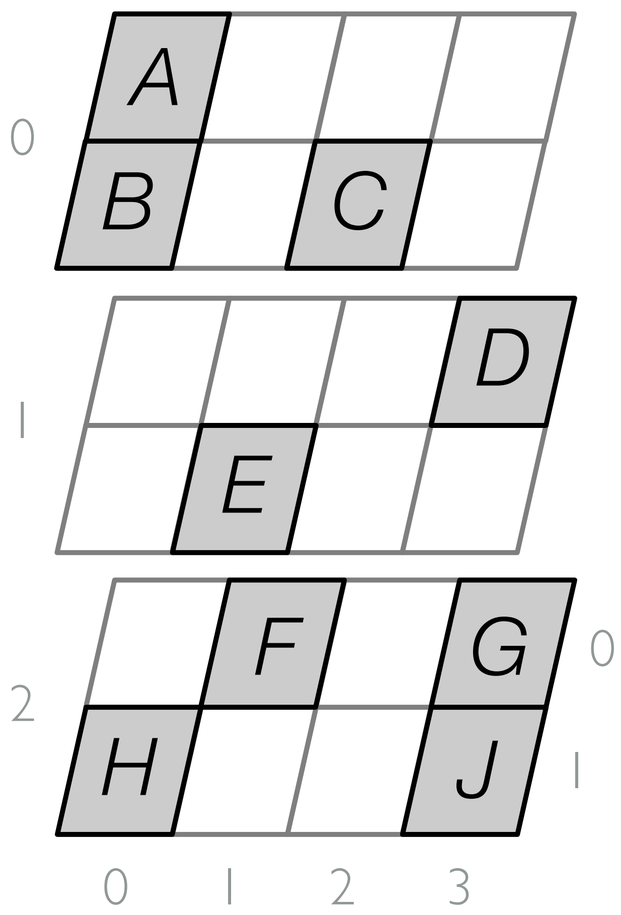
Tensor formats can be viewed as compositions of level formats



Tensor formats can be viewed as compositions of level formats



Tensor formats can be viewed as compositions of level formats



3

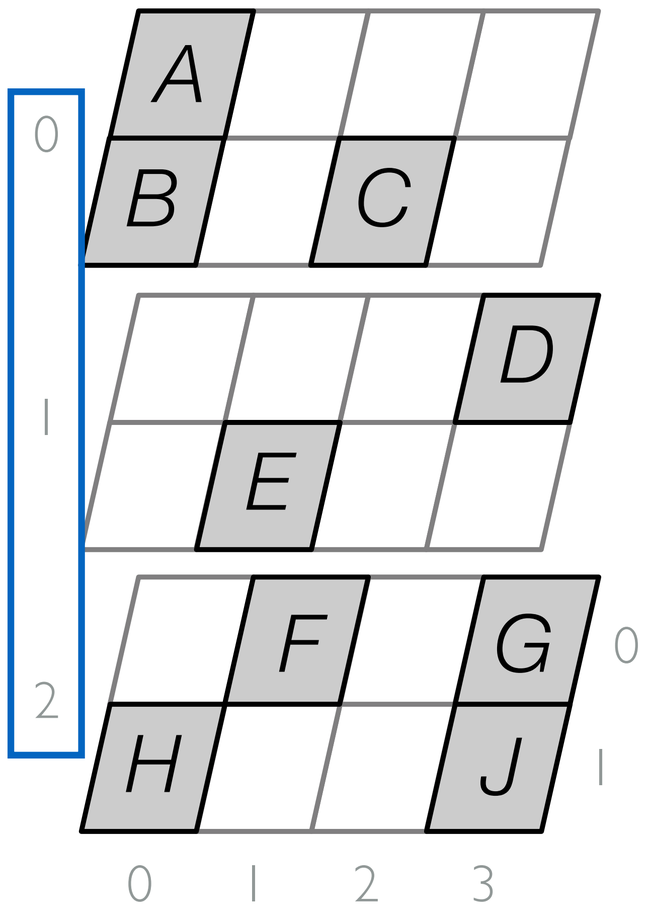
0	3	5	9

0	1	1	0	1	0	0	1	1

0	0	2	3	1	1	3	0	3

A	B	C	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8

Tensor formats can be viewed as compositions of level formats



Slices

3

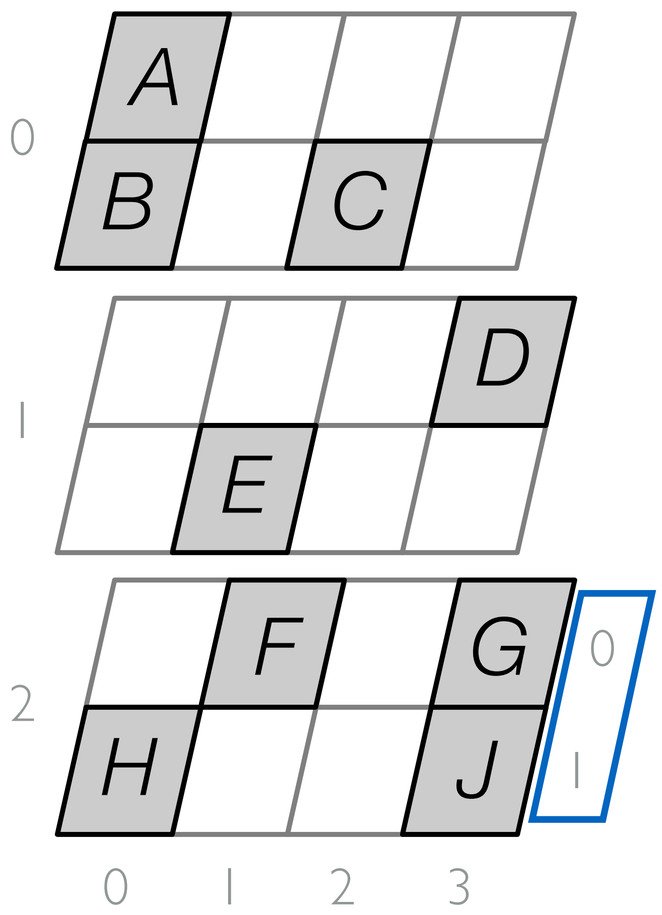
0	3	5	9

0	1	1	0	1	0	0	1	1

0	0	2	3	1	1	3	0	3

A	B	C	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8

Tensor formats can be viewed as compositions of level formats



Rows

3

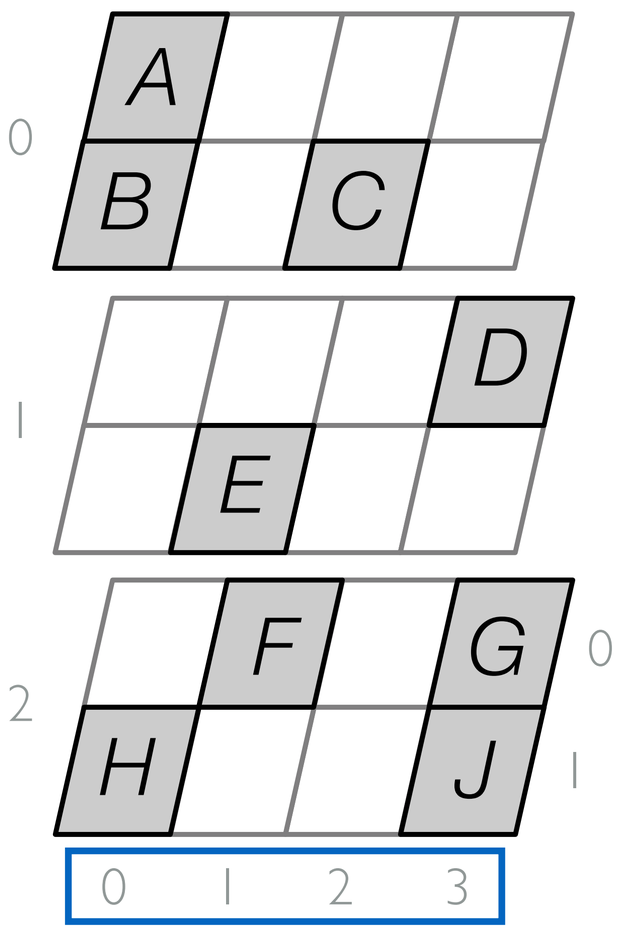
0	3	5	9

0	1	1	0	1	0	0	1	1

0	0	2	3	1	1	3	0	3

A	B	C	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8

Tensor formats can be viewed as compositions of level formats



Columns

3

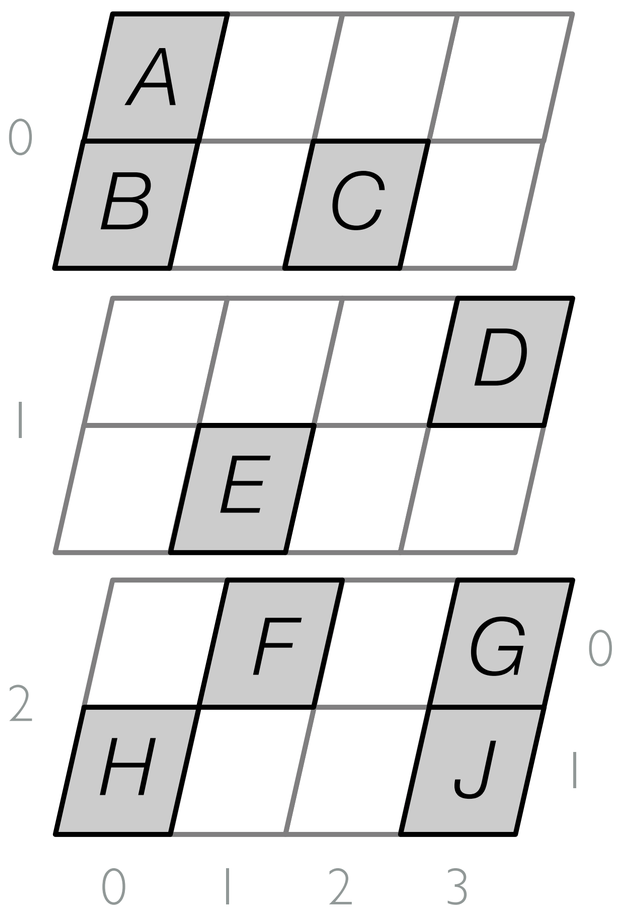
0	3	5	9

0	1	1	0	1	0	0	1	1

0	0	2	3	1	1	3	0	3

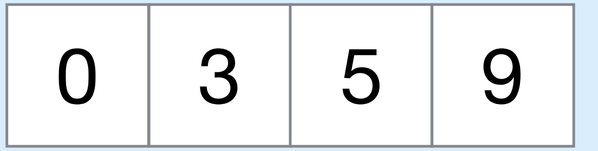
A	B	C	D	E	F	G	H	J
0	1	2	3	4	5	6	7	8

Tensor formats can be viewed as compositions of level formats



Dense

Compressed



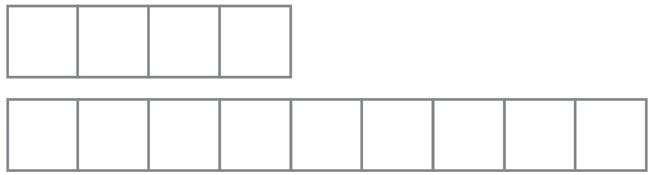
Singleton

0 1 2 3 4 5 6 7 8

The same level formats can be composed in many ways

Dense

Compressed

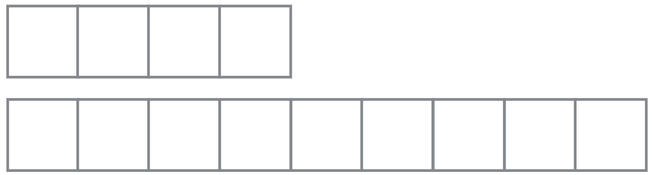


Singleton

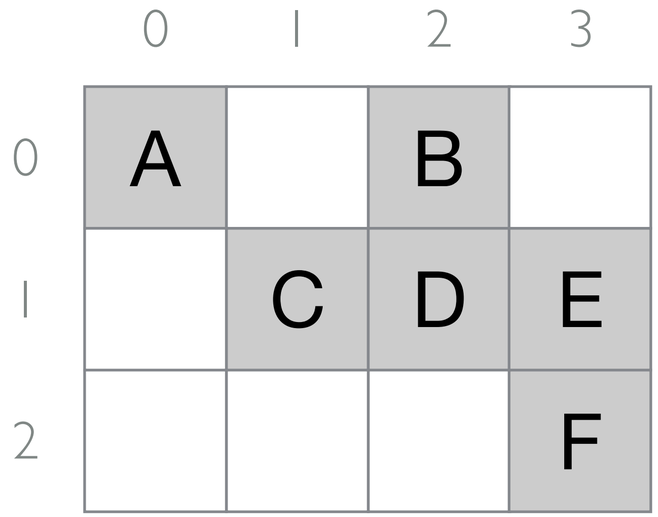
The same level formats can be composed in many ways

Dense

Compressed



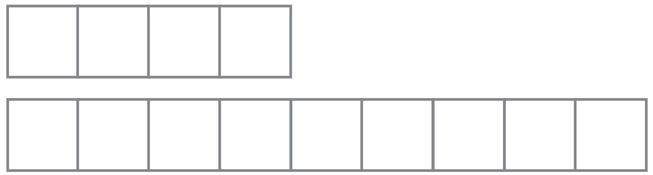
Singleton



The same level formats can be composed in many ways

Dense

Compressed



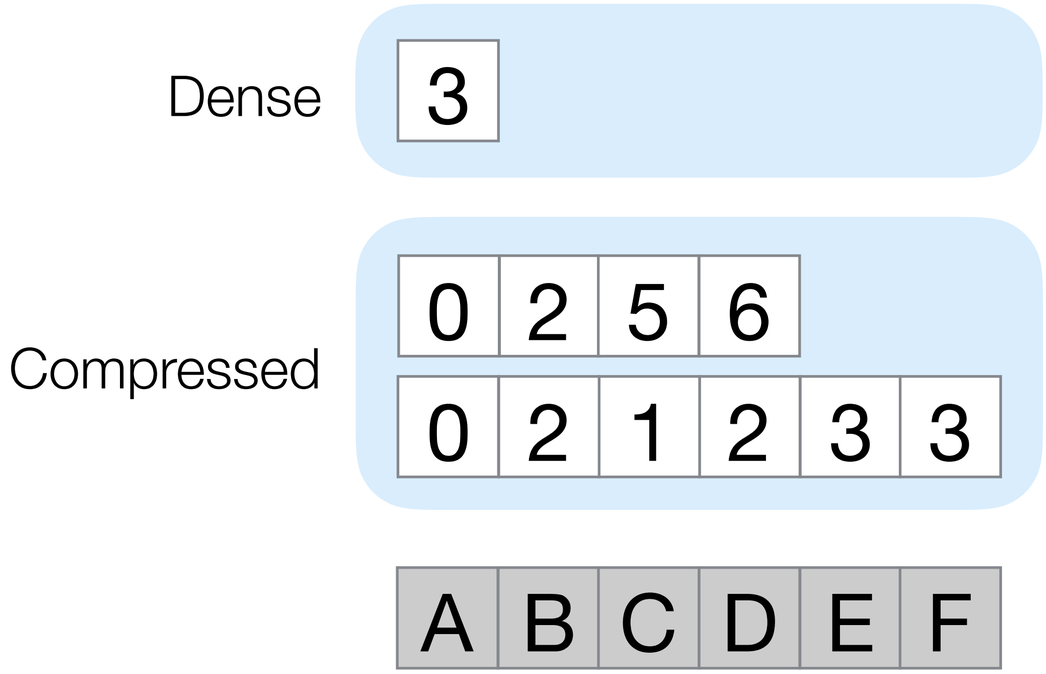
Singleton

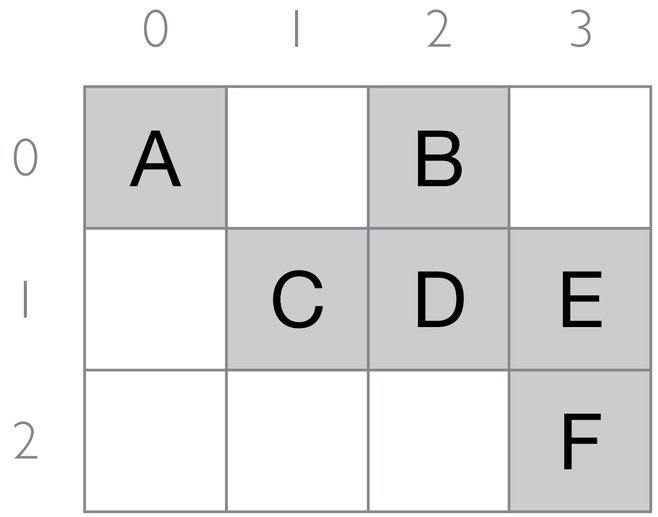
Dense

3

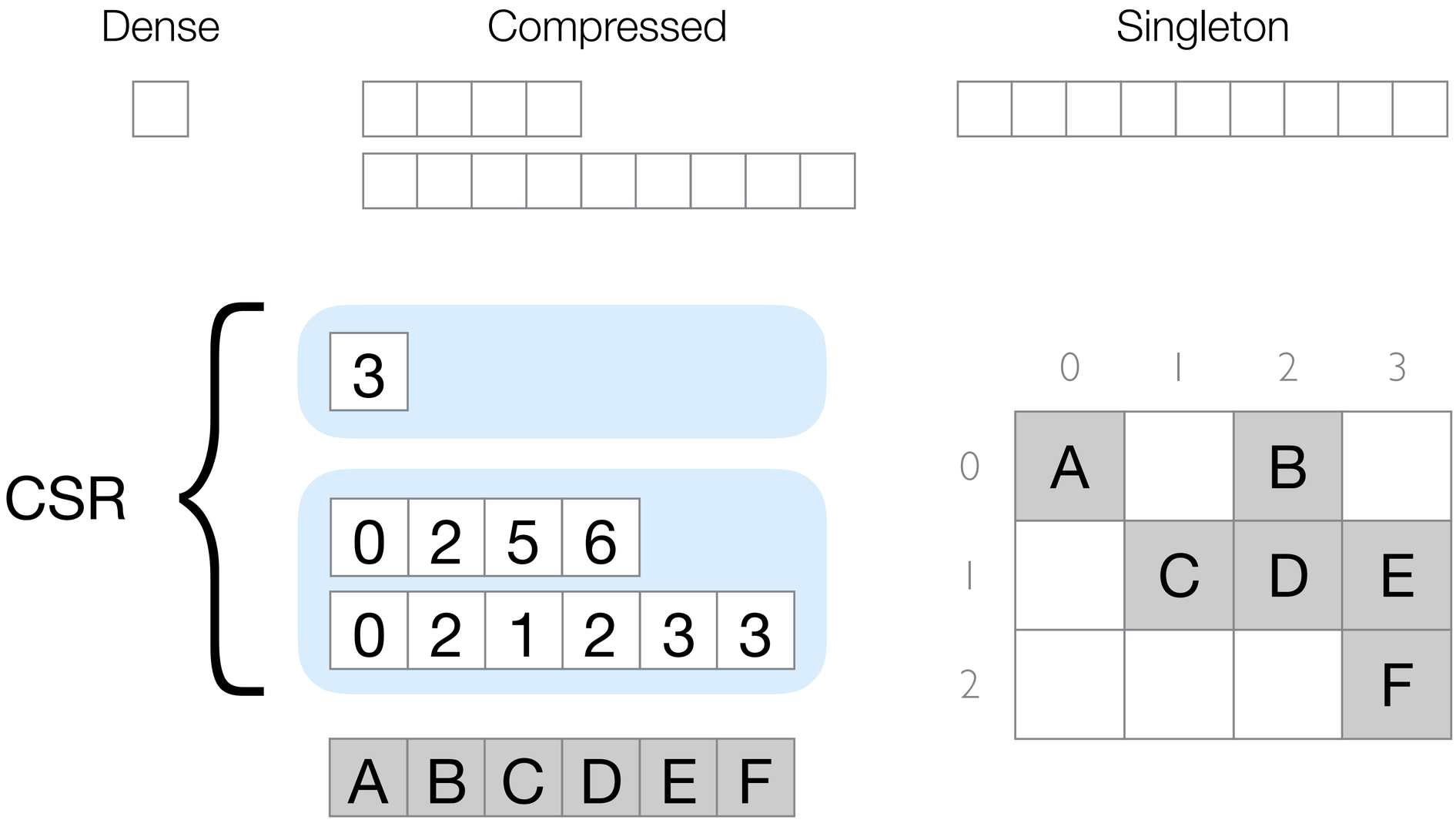
	0	1	2	3
0	A		B	
1		C	D	E
2				F

The same level formats can be composed in many ways





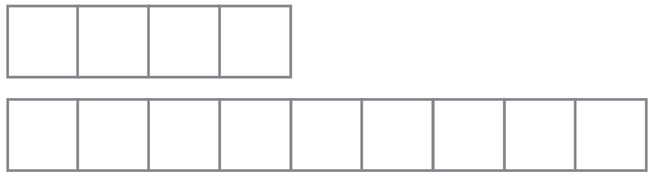
The same level formats can be composed in many ways



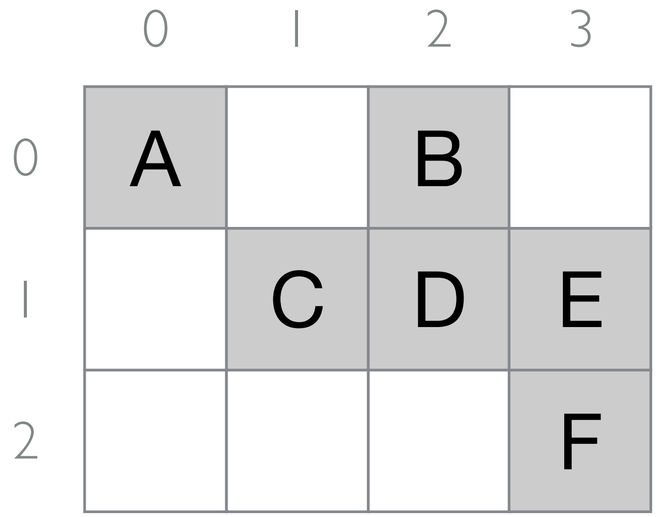
The same level formats can be composed in many ways

Dense

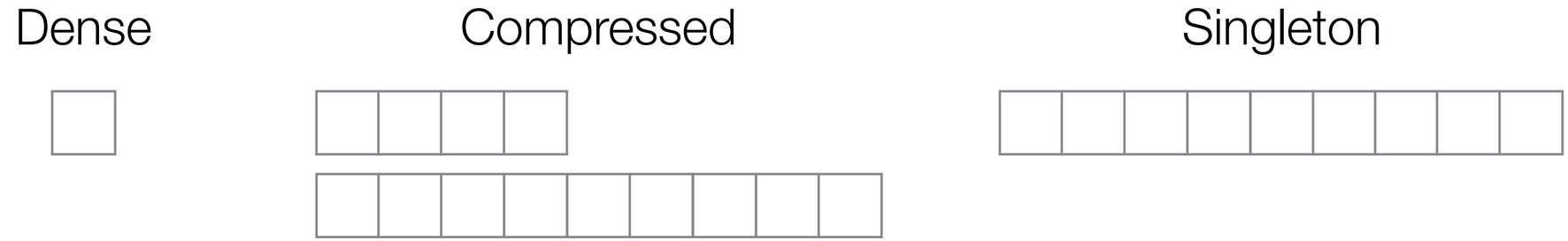
Compressed



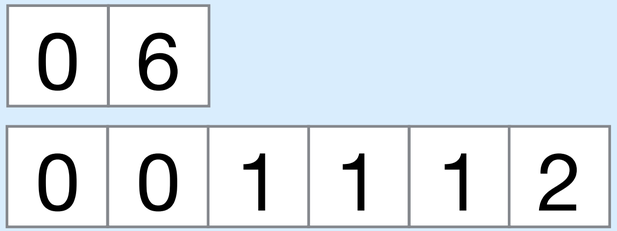
Singleton

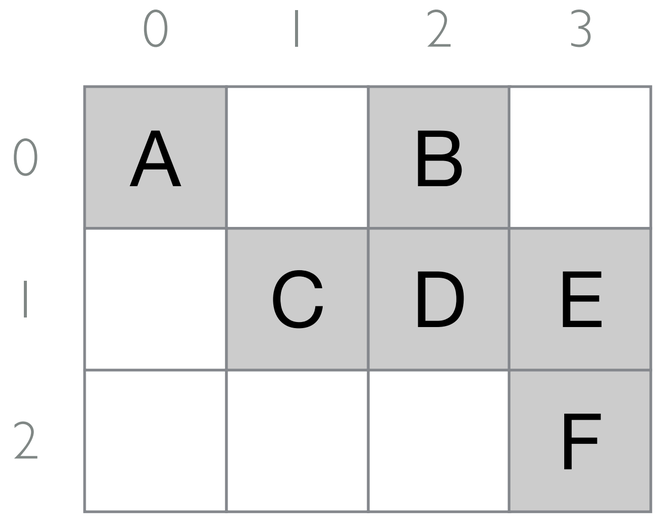


The same level formats can be composed in many ways

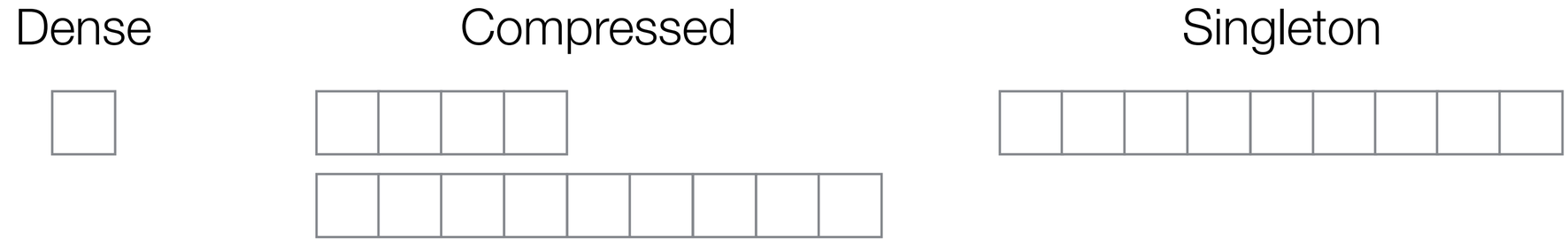


Compressed





The same level formats can be composed in many ways



Compressed

0	6

0	0	1	1	1	2

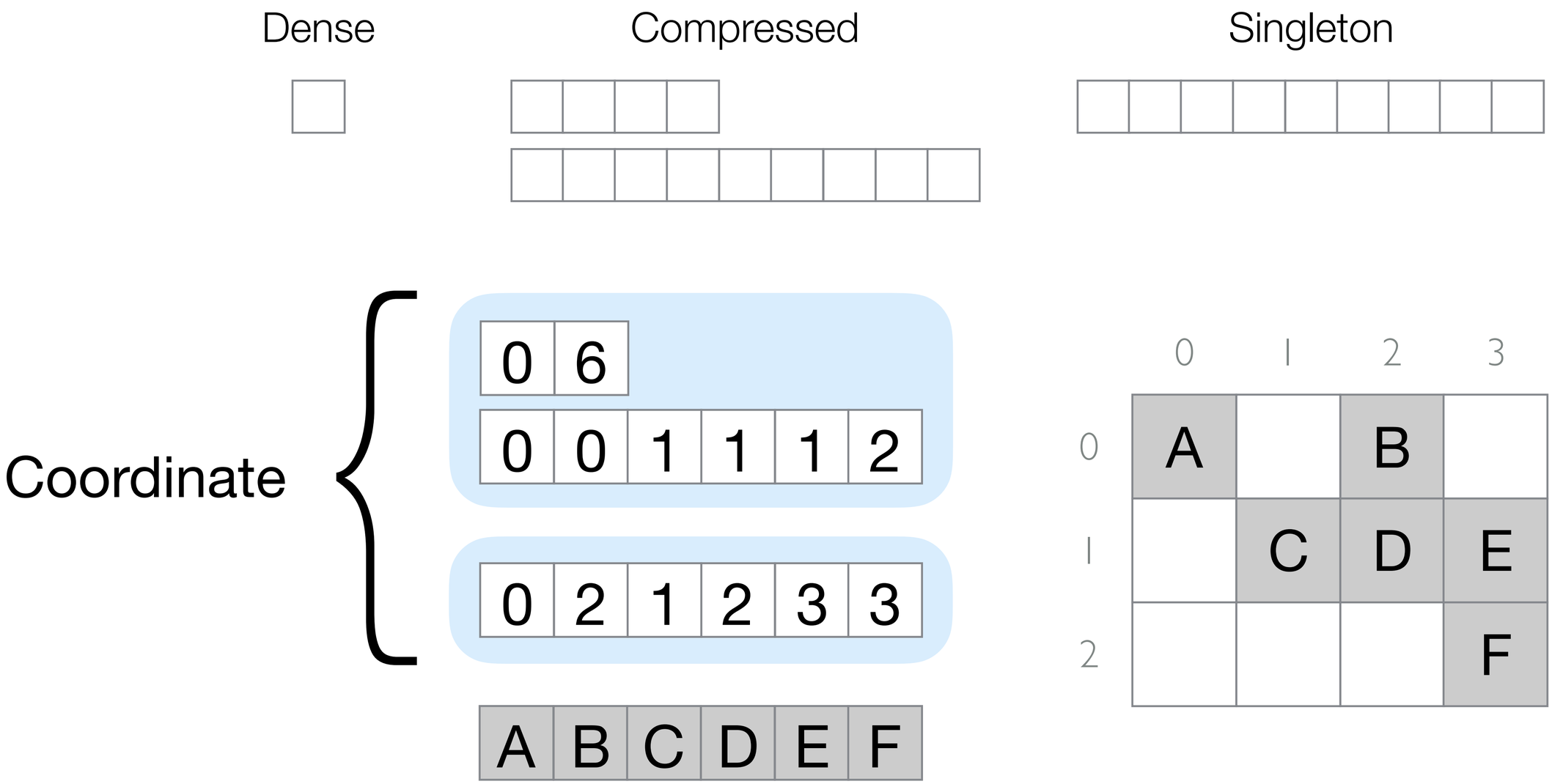
Singleton

0	2	1	2	3	3

A	B	C	D	E	F

	0	1	2	3
0	A		B	
1		C	D	E
2				F

The same level formats can be composed in many ways



The same level formats can be composed in many ways

Dense

Compressed

Singleton

The same level formats can be composed in many ways

Level formats

Dense Compressed Singleton

Tensor formats

Coordinate matrix
Compressed
Singleton

CSR
Dense
Compressed

[Tinney and Walker, 1967]

The same level formats can be composed in many ways

Level formats

Dense Compressed Singleton

Tensor formats

Coordinate matrix
Compressed
Singleton

CSR
Dense
Compressed

[Tinney and Walker, 1967]

Dense array tensor
Dense
Dense
Dense

Coordinate tensor
Compressed
Singleton
Singleton

Mode-generic tensor
Compressed
Singleton
Dense
Dense

[Baskaran et al. 2012]

The same level formats can be composed in many ways

Level formats

Dense Compressed Singleton

Tensor formats

Coordinate matrix
Compressed
Singleton

CSR
Dense
Compressed

[Tinney and Walker, 1967]

Dense array tensor
Dense
Dense
Dense

Coordinate tensor
Compressed
Singleton
Singleton

Mode-generic tensor
Compressed
Singleton
Dense
Dense

[Baskaran et al. 2012]

BCSR
Dense
Compressed
Dense
Dense

[Im and Yelick 1998]

CSB
Dense
Dense
Compressed
Singleton

[Buluç et al. 2009]

ELLPACK
Dense
Dense
Singleton

[Kincaid et al. 1989]

The same level formats can be composed in many ways

Level formats

Dense	Compressed	Singleton
Hashed	Range	Offset

Tensor formats

Coordinate matrix
Compressed
Singleton

CSR
Dense
Compressed

[Tinney and Walker, 1967]

Dense array tensor
Dense
Dense
Dense

Coordinate tensor
Compressed
Singleton
Singleton

Mode-generic tensor
Compressed
Singleton
Dense
Dense

[Baskaran et al. 2012]

BCSR
Dense
Compressed
Dense
Dense

[Im and Yelick 1998]

CSB
Dense
Dense
Compressed
Singleton

[Buluç et al. 2009]

ELLPACK
Dense
Dense
Singleton

[Kincaid et al. 1989]

The same level formats can be composed in many ways

Level formats

Dense	Compressed	Singleton
Hashed	Range	Offset

Tensor formats

Coordinate matrix
Compressed
Singleton

CSR
Dense
Compressed

[Tinney and Walker, 1967]

Dense array tensor
Dense
Dense
Dense

Coordinate tensor
Compressed
Singleton
Singleton

Mode-generic tensor
Compressed
Singleton
Dense
Dense

[Baskaran et al. 2012]

BCSR
Dense
Compressed
Dense
Dense

[Im and Yelick 1998]

CSB
Dense
Dense
Compressed
Singleton

[Buluç et al. 2009]

ELLPACK
Dense
Dense
Singleton

[Kincaid et al. 1989]

Hash map vector
Hashed

[Patwary et al. 2015]

Hash map matrix
Hashed
Hashed

DIA
Dense
Range
Offset

[Saad 2003]

Block DIA
Dense
Range
Offset
Dense
Dense

$$A_{ij} = \sum_k B_{ijk} c_k$$

Tensor Algebra Compiler (taco)

[Kjolstad et al. 2017]

$A : (\text{dense}, \text{dense})$

$B : (\text{dense}, \text{compressed}, \text{compressed})$

$c : (\text{compressed})$

```
for (int i = 0; i < m; i++) {

  for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1 + 1]; pB2++) {
    int j = B2_idx[pB2];
    int pA2 = (i * n) + j;

    int pB3 = B3_pos[pB2];
    int pc1 = c1_pos[0];
    while (pB3 < B3_pos[pB2 + 1] && pc1 < c1_pos[1]) {
      int kB = B3_idx[pB3];
      int kc = c1_idx[pc1];
      int k = min(kB, kc);
      if (kB == k && kc == k) {
        a[pA2] += b[pB3] * c[pc1];
      }
      if (kB == k) pB3++;
      if (kc == k) pc1++;
    }
  }
}
```

taco generates code dimension by dimension

$$A_{ij} = \sum_k B_{ijk} \cdot c_k$$

taco generates code dimension by dimension

$$A_{ij} = \sum_k B_{ijk} \cdot c_k$$

```
for (int i = 0; i < m; i++) {

}
```

taco generates code dimension by dimension

$$A_{ij} = \sum_k B_{ijk} \cdot c_k$$

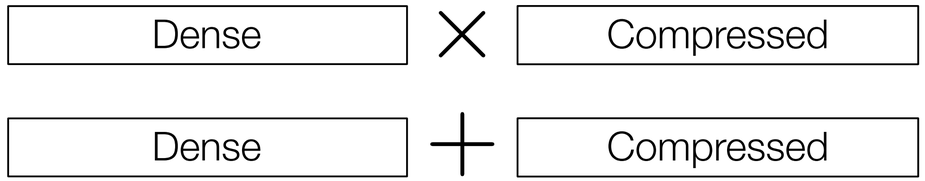
```
for (int i = 0; i < m; i++) {
  for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1 + 1]; pB2++) {
    int j = B2_idx[pB2];
    int pA2 = (i * n) + j;









  }
}
```

taco generates code dimension by dimension

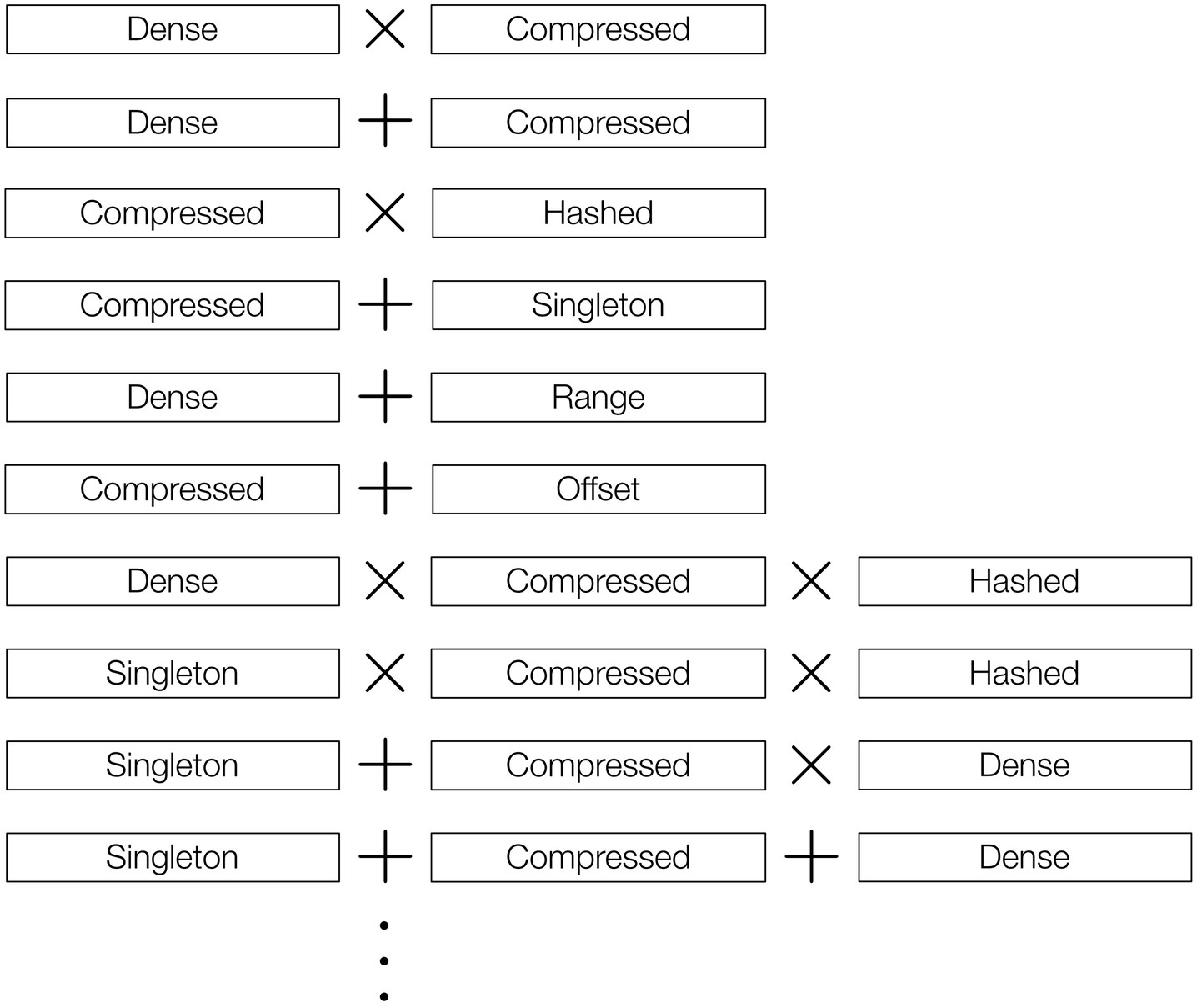
$$A_{ij} = \sum_k B_{ijk} \cdot c_k$$

```
for (int i = 0; i < m; i++) {
  for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1 + 1]; pB2++) {
    int j = B2_idx[pB2];
    int pA2 = (i * n) + j;
    int pB3 = B3_pos[pB2];
    int pc1 = c1_pos[0];
    while (pB3 < B3_pos[pB2 + 1] && pc1 < c1_pos[1]) {
      int kB = B3_crd[pB3];
      int kc = c1_crd[pc1];
      int k = min(kB, kc);
      if (kB == k && kc == k) {
        A[pA2] += B[pB3] * c[pc1];
      }
      if (kB == k) pB3++;
      if (kc == k) pc1++;
    }
  }
}
```

Hand-coding support for a wide range of level formats is also infeasible



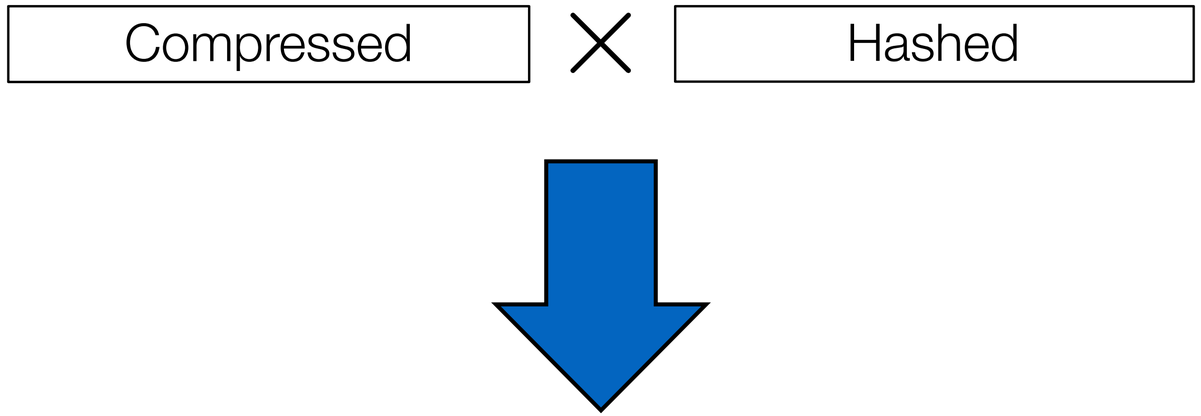
Hand-coding support for a wide range of level formats is also infeasible



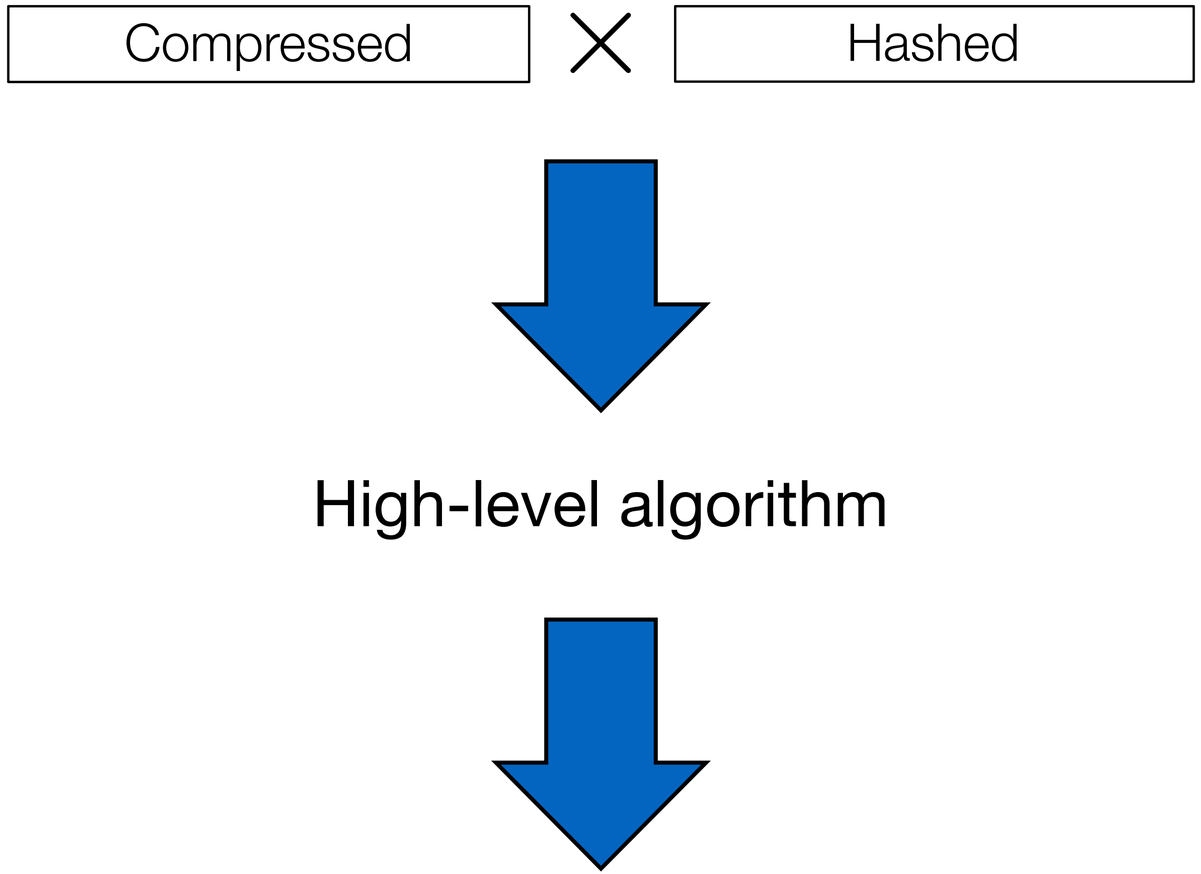
Code generation is performed in two stages

Code generation is performed in two stages

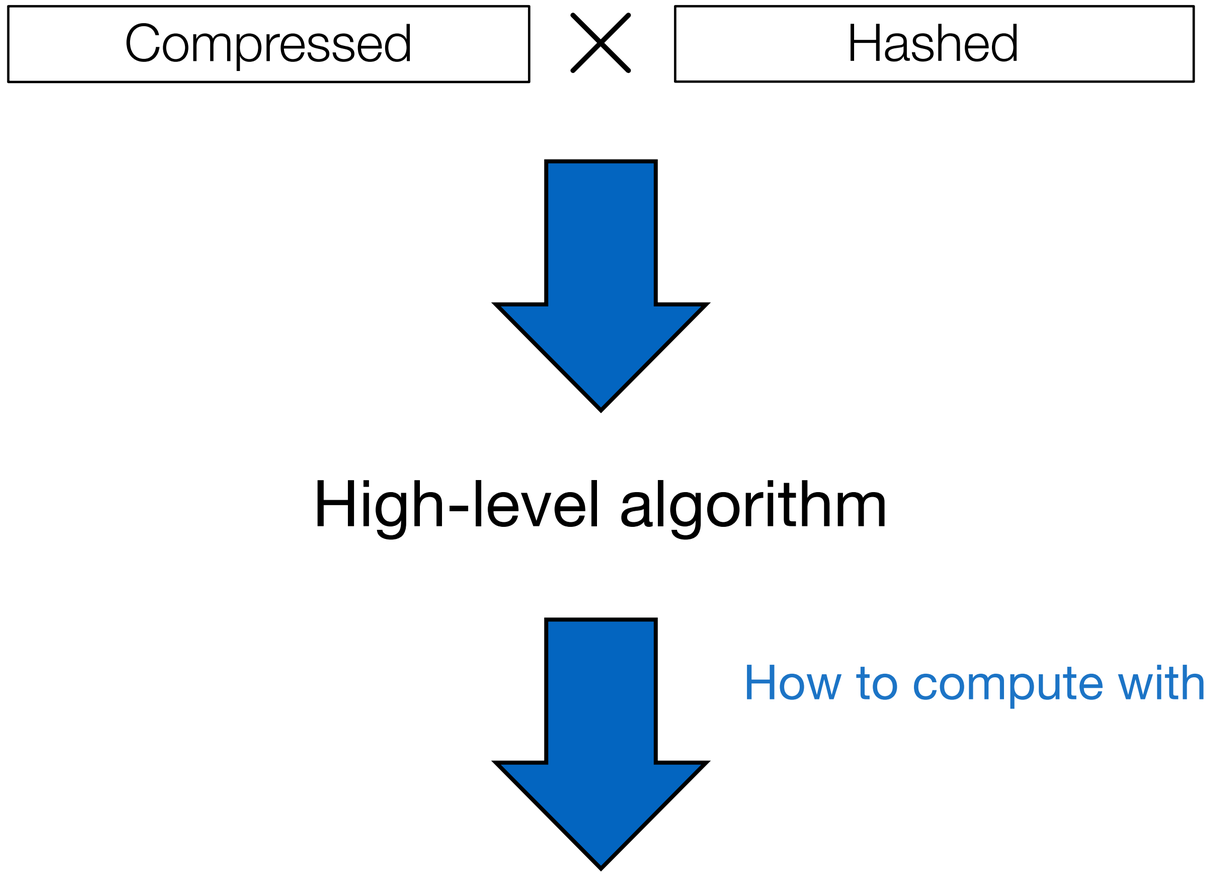
Code generation is performed in two stages



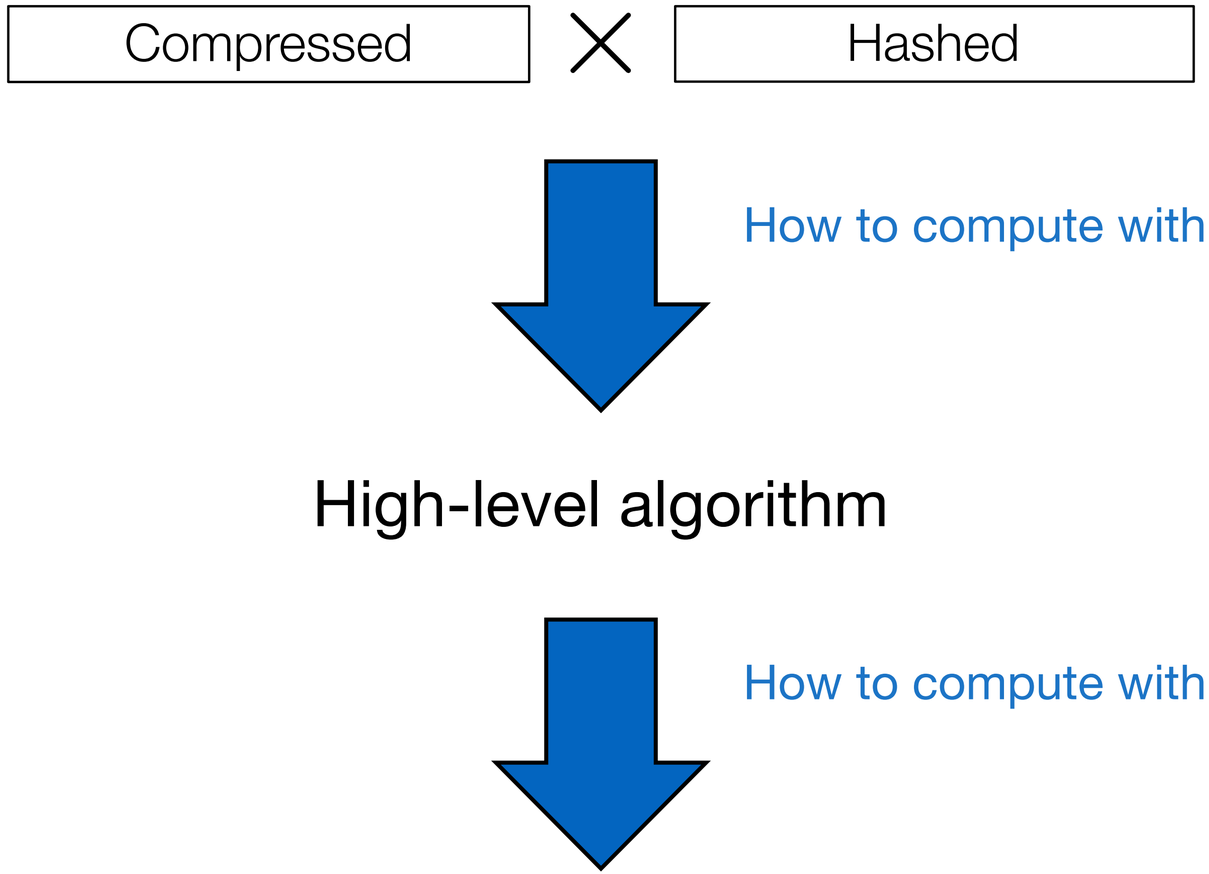
Code generation is performed in two stages



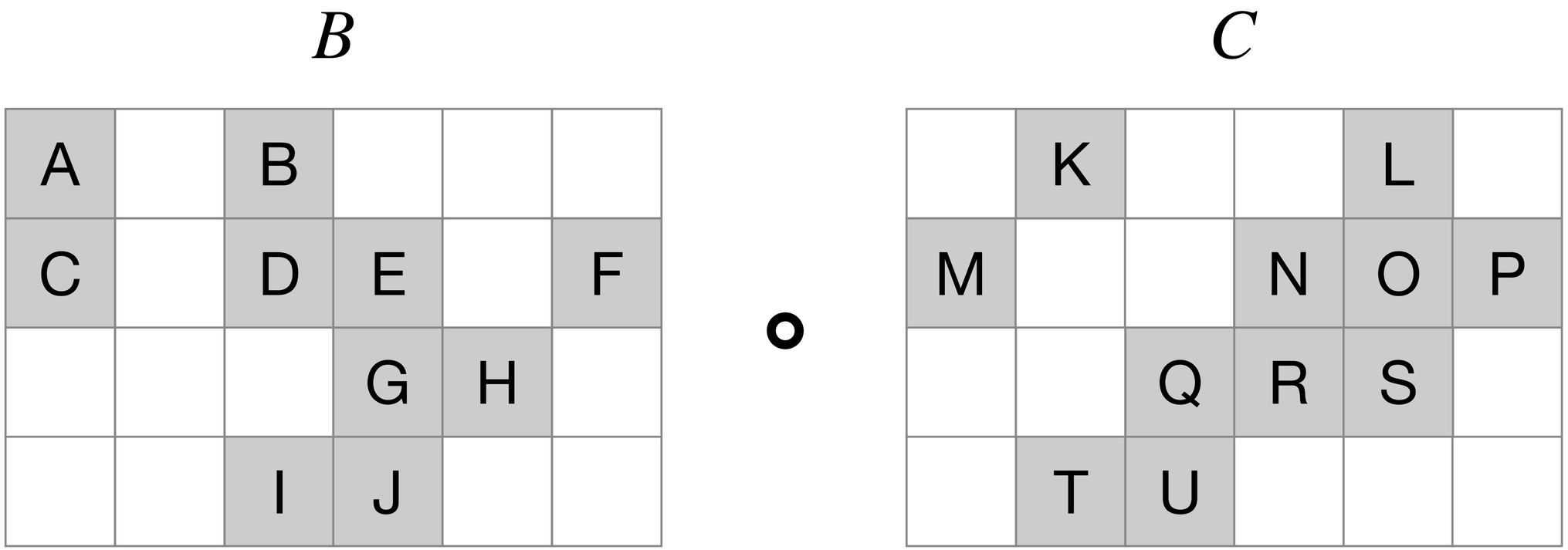
Code generation is performed in two stages



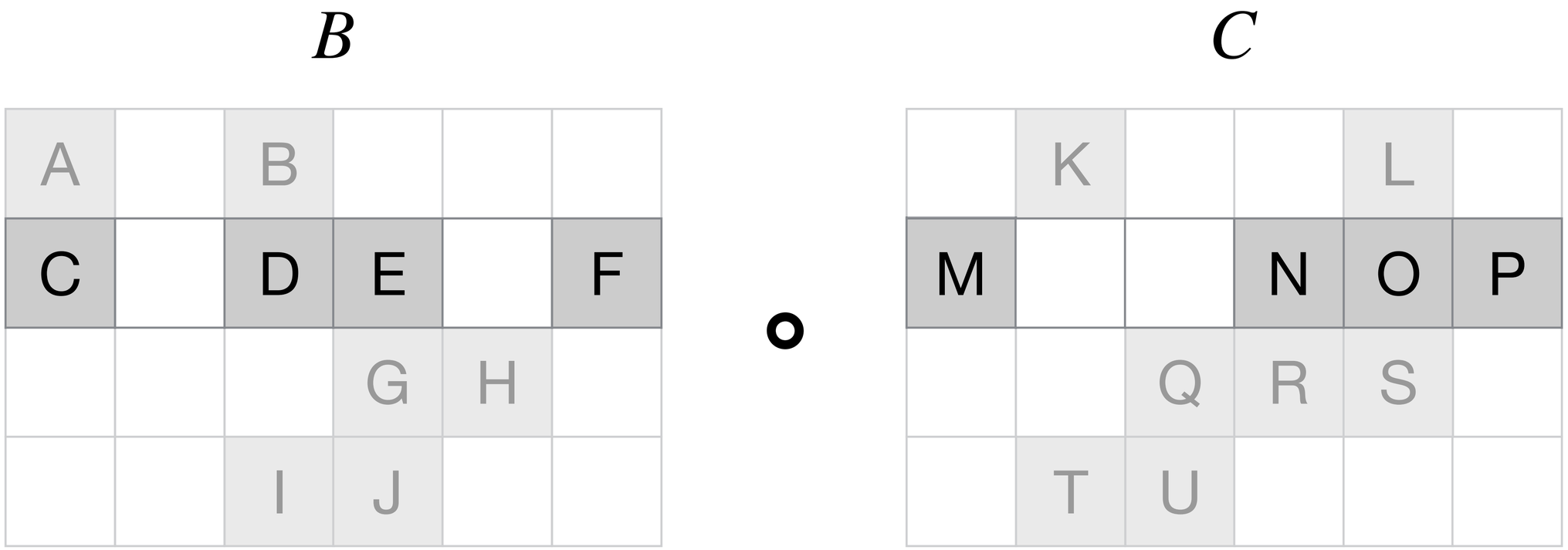
Code generation is performed in two stages



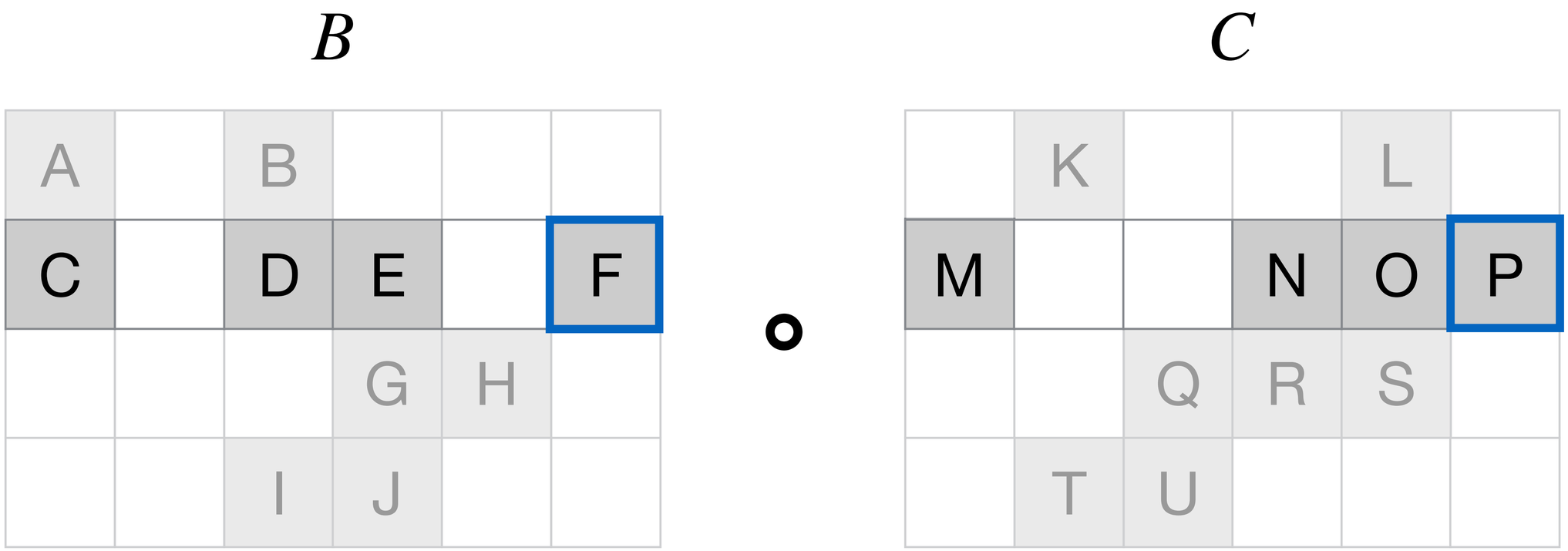
Tensor algebra computations can be expressed in terms of high-level operations on tensor operands



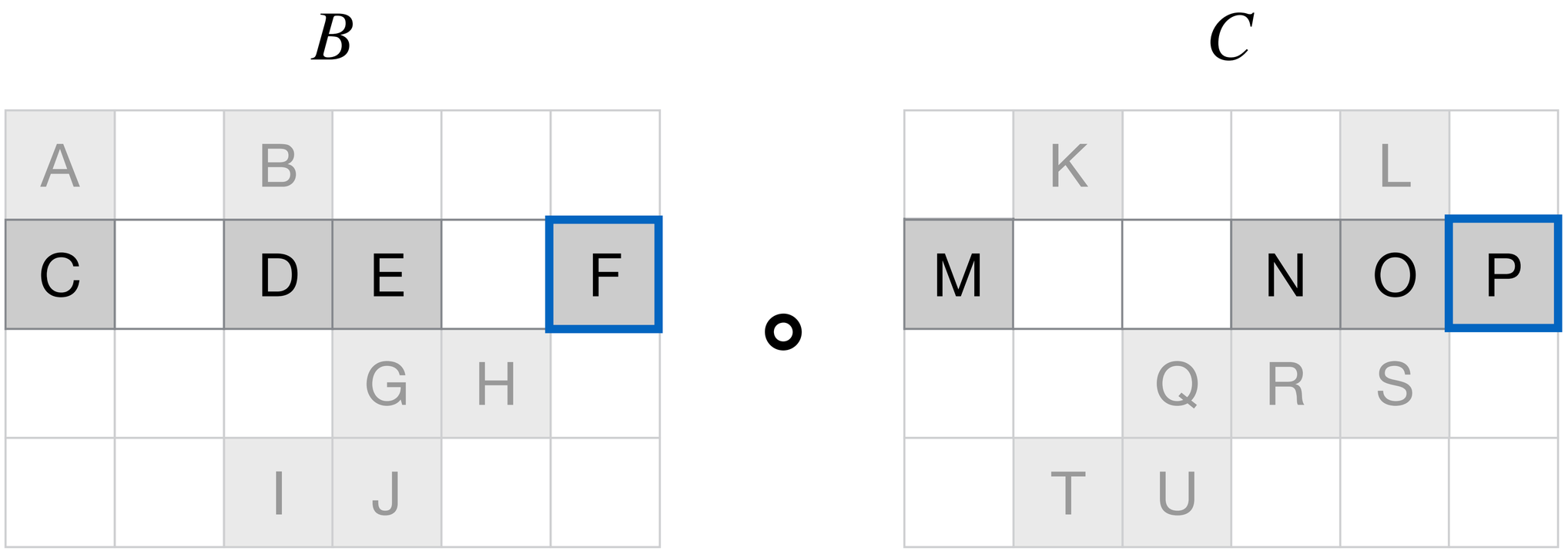
Tensor algebra computations can be expressed in terms of high-level operations on tensor operands



Tensor algebra computations can be expressed in terms of high-level operations on tensor operands



Tensor algebra computations can be expressed in terms of high-level operations on tensor operands



Level formats declare whether they support various high-level operations

Dense

Range

Compressed

Singleton

Offset

Hashed

Level formats declare whether they support various high-level operations

Level formats declare whether they support various high-level operations

	Random access	**Iteration**
Dense	✓	
Range		
Compressed		
Singleton		
Offset		
Hashed	✓	

Level formats declare whether they support various high-level operations

Level formats declare whether they support various high-level operations

Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations

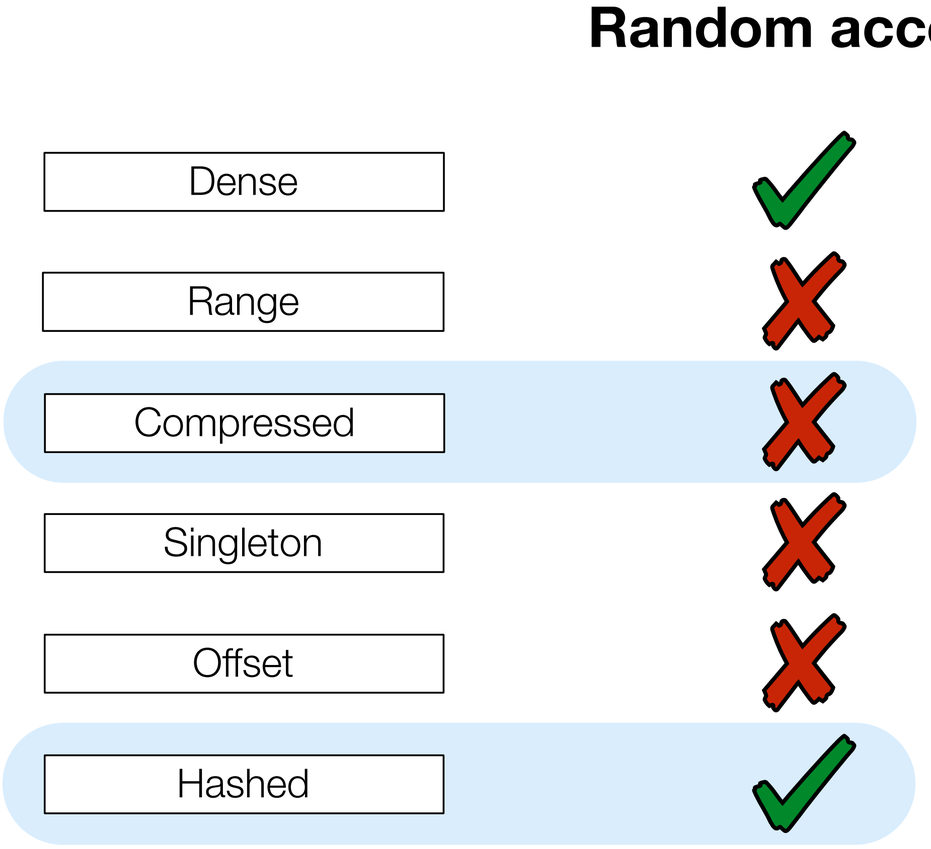
Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations

Random access

Dense	✓
Range	✗
Compressed	✗
Singleton	✗
Offset	✗
Hashed	✓

$$B \quad \circ \quad C$$

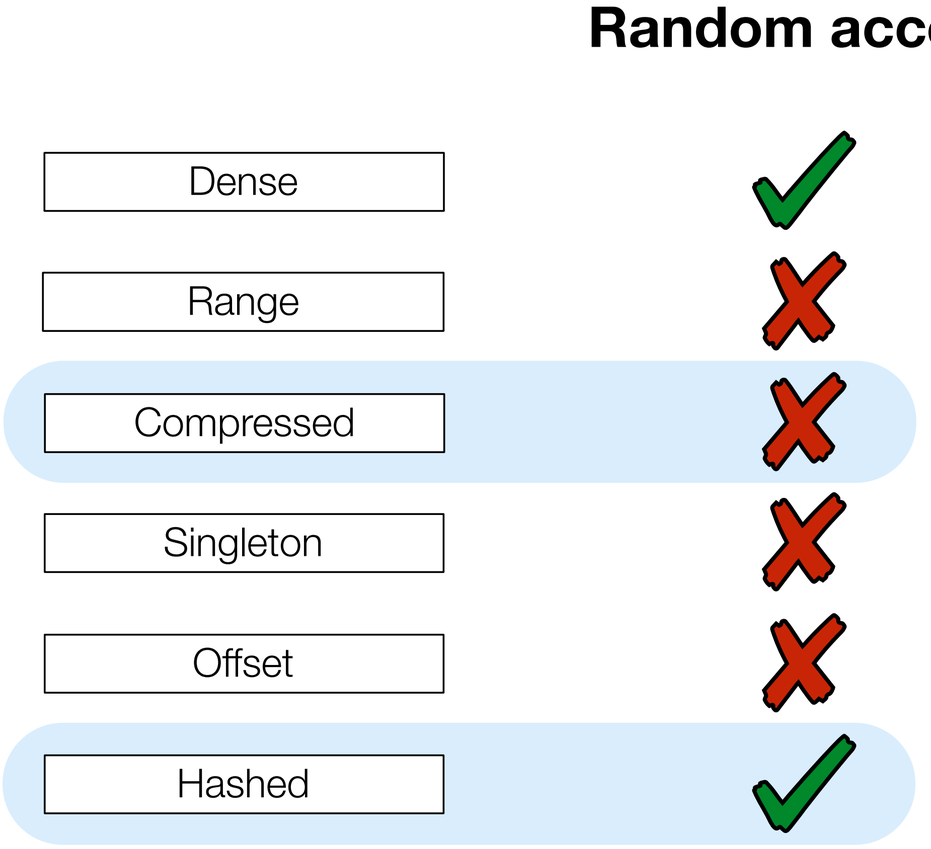
Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations

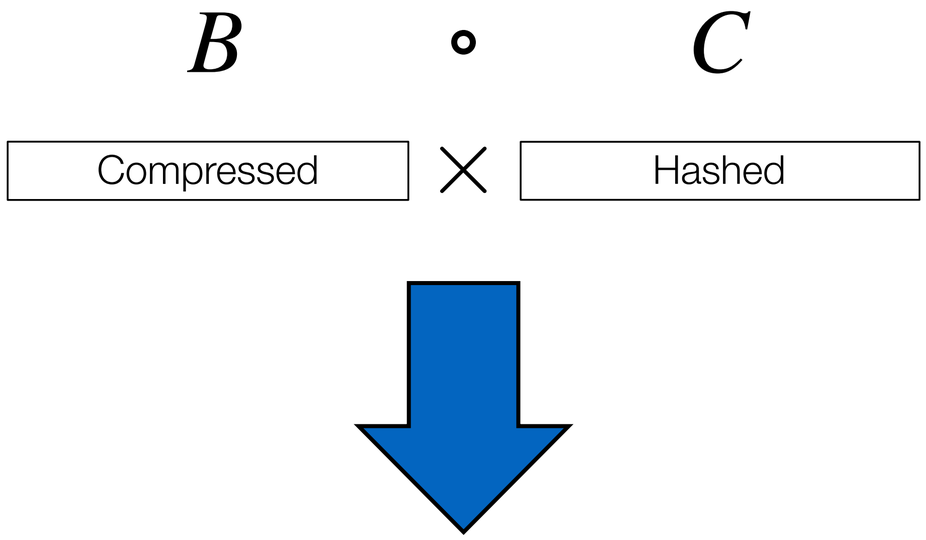


$$B \circ C$$

Compressed × Hashed

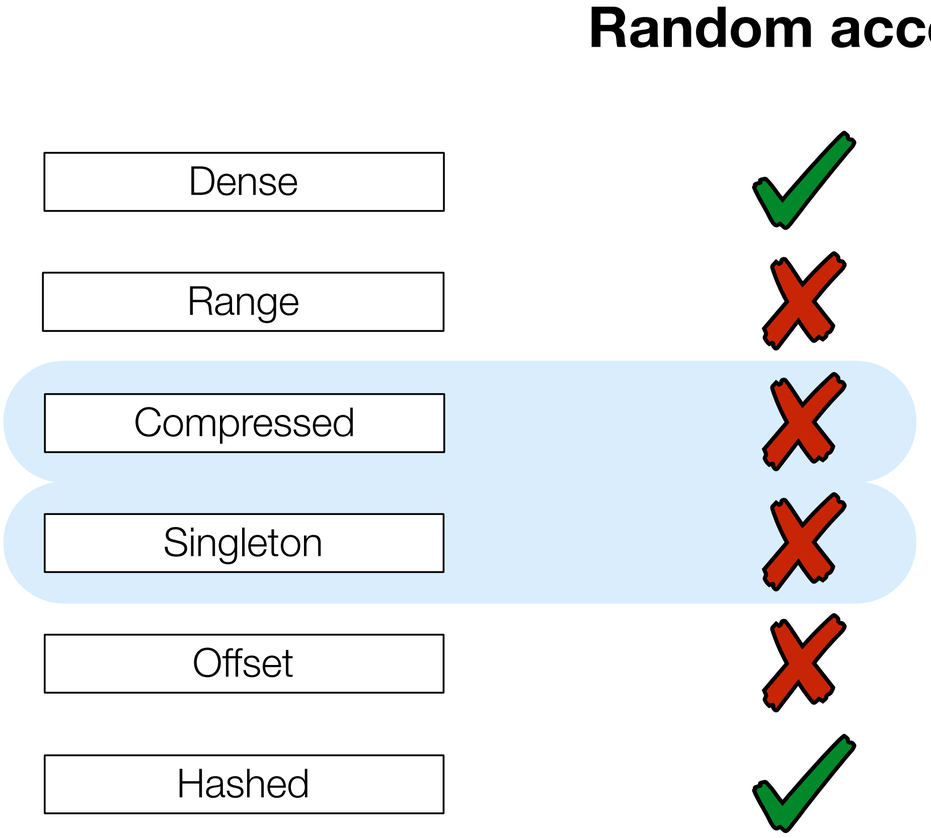
Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations





Iterate over *B* and random access *C*

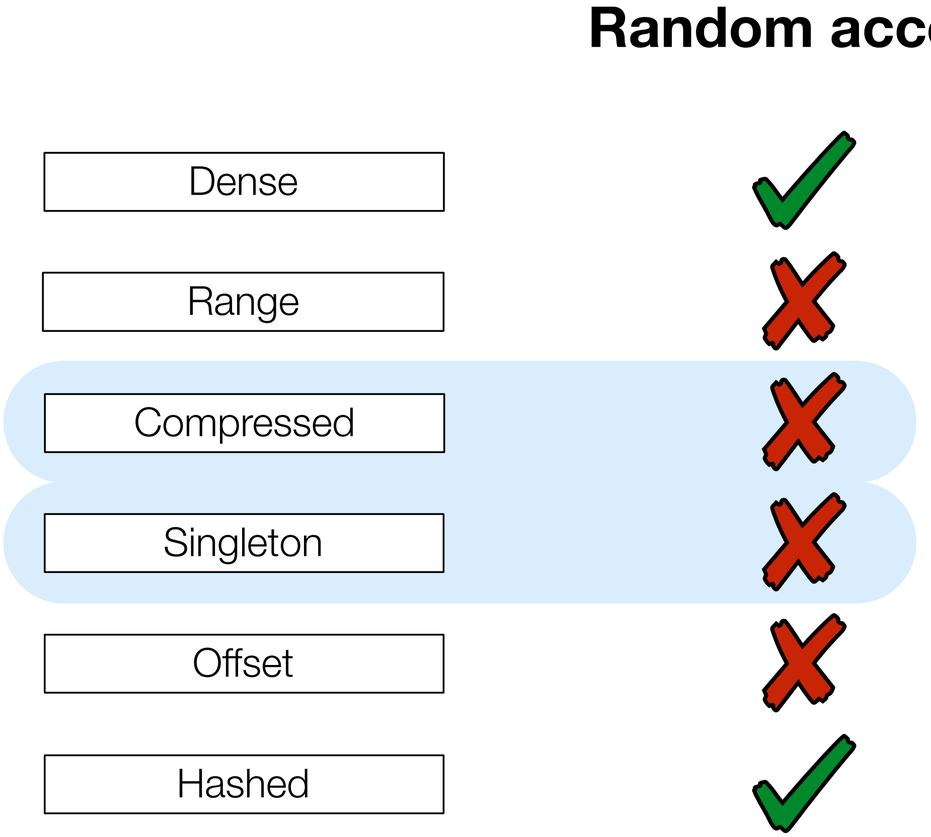
Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations

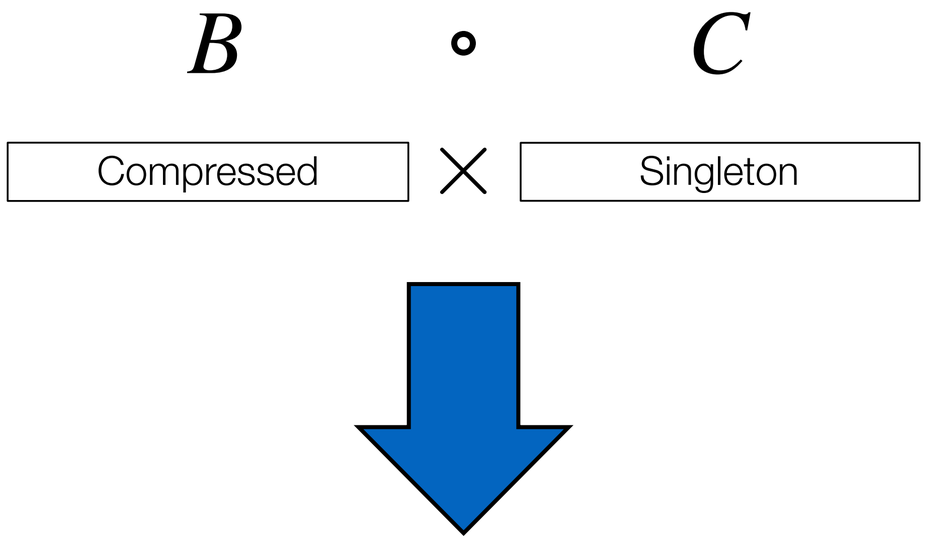


$$B \circ C$$

Compressed × Singleton

Compiler constructs efficient algorithm by reasoning about whether operands support required high-level operations





Simultaneously iterate over *B* and *C*

Level formats also specify how they support high-level operations

	Random access	**Iteration**
Dense	✓	✓
Compressed	✗	✓
Hashed	✓	✓

Level formats also specify how they support high-level operations

	Random access	Iteration
Dense	`int pB2 = pB1 * N + j;`	✓
Compressed	✗	✓
Hashed	✓	✓

Level formats also specify how they support high-level operations

	Random access	Iteration
Dense	`int pB2 = pB1 * N + j;`	✓
Compressed	✗	✓

Hashed	`int pB2 = j % W + pB1 * W;`
	`if (crd[pB2] != j &&`
	`  crd[pB2] != -1) {`
	`  int end = pB2;`
	`  do {`
	`  pB2 = (pB2 + 1) % W;`
	`  } while (crd[pB2] != j &&`
	`  crd[pB2] != -1 &&`
	`  pB2 != end);`
	`}`
	`if (crd[pB2] == j) {`
		✓

⋮		

Level formats also specify how they support high-level operations

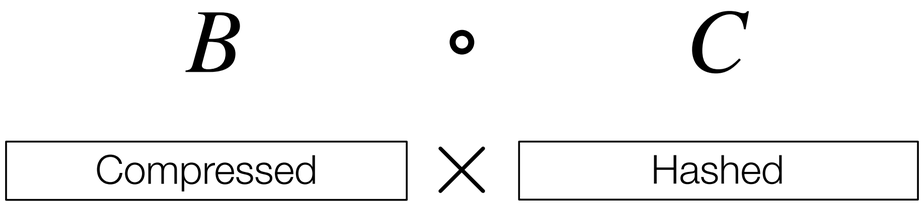
	Random access	Iteration
Dense	`int pB2 = pB1 * N + j;`	`for (int j = 0; j < N; j++) {` `  int pB2 = pB1 * N + j;`

Compressed		`for (int pB2 = pos[pB1];`
		`  pB2 < pos[pB1+1];`
		`  pB2++) {`
		`  int j = crd[pB2];`

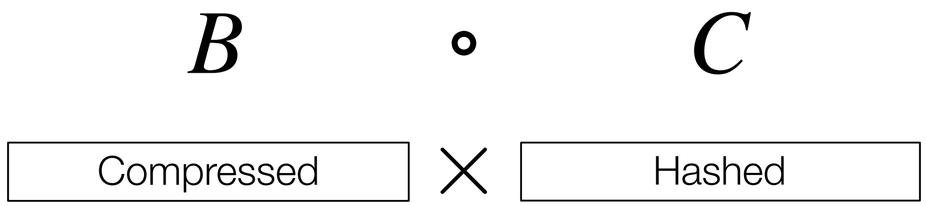
Hashed	`int pB2 = j % W + pB1 * W;`
	`if (crd[pB2] != j &&`
	`  crd[pB2] != -1) {`
	`  int end = pB2;`
	`  do {`
	`  pB2 = (pB2 + 1) % W;`
	`  } while (crd[pB2] != j &&`
	`  crd[pB2] != -1 &&`
	`  pB2 != end);`
	`}`
	`if (crd[pB2] == j) {`
		`for (int pB2 = pB1 * W;`
		`  pB2 < (pB1 + 1) * W;`
		`  pB2++) {`
		`  int j = crd[pB2];`
		`  if (j != -1) {`

•
•
•

Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations

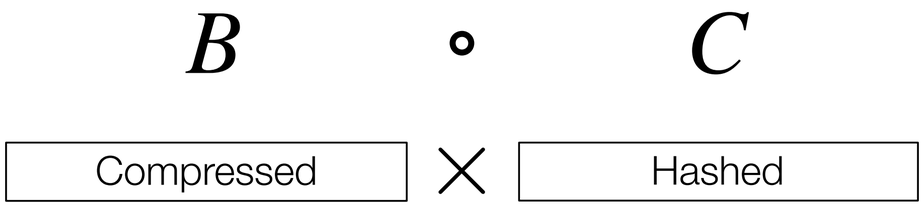


Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations



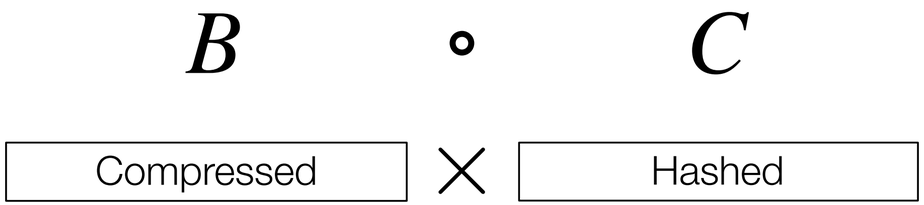

```
for every element b in B:
  find corresponding element c in C
  A[i][j] = b * c;
```

Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations




```
for every element b in B:
  find corresponding element c in C
  A[i][j] = b * c;
```

Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations




```
for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1+1]; pB2++) {
  int j = B2_crd[pB2];
  find corresponding element c in C
  A[i][j] = B[pB2] * c;
}
```

Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations

$$B \quad \circ \quad C$$


```
for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1+1]; pB2++) {
  int j = B2_crd[pB2];
  find corresponding element c in C
  A[i][j] = B[pB2] * c;
}
```

Compiler specializes constructed algorithm to operand formats by inlining code that implements required high-level operations

$$B \quad \circ \quad C$$

Compressed × Hashed

```
for (int pB2 = B2_pos[pB1]; pB2 < B2_pos[pB1+1]; pB2++) {
  int j = B2_crd[pB2];
  int pC2 = j % W + pC1 * W;
  if (C2_crd[pC2] != j && C2_crd[pC2] != -1) {
    int end = pC2;
    do {
      pC2 = (pC2 + 1) % W;
    } while (C2_crd[pB2] != j && C2_crd[pB2] != -1 && pC2 != end);
  }
  if (C2_crd[pC2] == j) {
    A[i][j] = B[pB2] * C[pC2];
  }
}
```

The same process can be repeated dimension by dimension

$$A_{ijk} = B_{ijk} + C_{ijk}$$

The same process can be repeated dimension by dimension

$$A_{ijk} = B_{ijk} + C_{ijk}$$

```
int iB = 0;
int C0_pos = C0_pos_arr[0];
while (C0_pos < C0_pos_arr[1]) {
  int iC = C0_idx_arr[C0_pos];
  int C0_end = C0_pos + 1;
  if (iC == iB)
    while ((C0_end < C0_pos_arr[1]) && (C0_idx_arr[C0_end] == iB)) {
      C0_end++;
    }
  if (iC == iB) {
    int B1_pos = B1_pos_arr[iB];
    int C1_pos = C0_pos;
    while ((B1_pos < B1_pos_arr[iB + 1]) && (C1_pos < C0_end)) {
      int jB = B1_idx_arr[B1_pos];
      int jC = C1_idx_arr[C1_pos];
      int j = min(jB, jC);
      int A1_pos = (iB * A1_size) + j;
      int C1_end = C1_pos + 1;
      if (jC == j)
        while ((C1_end < C0_end) && (C1_idx_arr[C1_end] == j)) {
          C1_end++;
        }
      if ((jB == j) && (jC == j)) {
        int B2_pos = B2_pos_arr[B1_pos];
        int C2_pos = C1_pos;
        while ((B2_pos < B2_pos_arr[B1_pos + 1]) && (C2_pos < C1_end)) {
          int kB = B2_idx_arr[B2_pos];
          int kC = C2_idx_arr[C2_pos];
          int k = min(kB, kC);
          int A2_pos = (A1_pos * A2_size) + k;
          if ((kB == k) && (kC == k)) {
            A_val_arr[A2_pos] = B_val_arr[B2_pos] + C_val_arr[C2_pos];
          } else if (kB == k) {
            A_val_arr[A2_pos] = B_val_arr[B2_pos];
          } else {
            A_val_arr[A2_pos] = C_val_arr[C2_pos];
          }
          if (kB == k) B2_pos++;
          if (kC == k) C2_pos++;
        }
        while (B2_pos < B2_pos_arr[B1_pos + 1]) {
          int kB0 = B2_idx_arr[B2_pos];
          int A2_pos0 = (A1_pos * A2_size) + kB0;
          A_val_arr[A2_pos0] = B_val_arr[B2_pos];
          B2_pos++;
        }
        while (C2_pos < C1_end) {
          int kC0 = C2_idx_arr[C2_pos];
          int A2_pos1 = (A1_pos * A2_size) + kC0;
          A_val_arr[A2_pos1] = C_val_arr[C2_pos];
          C2_pos++;
        }
      } else if (jB == j) {
        for (int B2_pos0 = B2_pos_arr[B1_pos];
                 B2_pos0 < B2_pos_arr[B1_pos + 1]; B2_pos0++) {
          int kB1 = B2_idx_arr[B2_pos0];
          int A2_pos2 = (A1_pos * A2_size) + kB1;
          A_val_arr[A2_pos2] = B_val_arr[B2_pos0];
        }
      } else {
        for (int C2_pos0 = C1_pos; C2_pos0 < C1_end; C2_pos0++) {
          int kC1 = C2_idx_arr[C2_pos0];
          int A2_pos3 = (A1_pos * A2_size) + kC1;
          A_val_arr[A2_pos3] = C_val_arr[C2_pos0];
        }
      }
      if (jB == j) B1_pos++;
      if (jC == j) C1_pos = C1_end;
    }
    while (B1_pos < B1_pos_arr[iB + 1]) {
      int jB0 = B1_idx_arr[B1_pos];
      int A1_pos0 = (iB * A1_size) + jB0;
      for (int B2_pos1 = B2_pos_arr[B1_pos];
               B2_pos1 < B2_pos_arr[B1_pos + 1]; B2_pos1++) {
        int kB2 = B2_idx_arr[B2_pos1];
        int A2_pos4 = (A1_pos0 * A2_size) + kB2;
        A_val_arr[A2_pos4] = B_val_arr[B2_pos1];
      }
      B1_pos++;
    }
    while (C1_pos < C0_end) {
      int jC0 = C1_idx_arr[C1_pos];
      int A1_pos1 = (iB * A1_size) + jC0;
      int C1_end0 = C1_pos + 1;
      while ((C1_end0 < C0_end) && (C1_idx_arr[C1_end0] == jC0)) {
        C1_end0++;
      }
      for (int C2_pos1 = C1_pos; C2_pos1 < C1_end0; C2_pos1++) {
        int kC2 = C2_idx_arr[C2_pos1];
        int A2_pos5 = (A1_pos1 * A2_size) + kC2;
        A_val_arr[A2_pos5] = C_val_arr[C2_pos1];
      }
      C1_pos = C1_end0;
    }
  } else {
    for (int B1_pos0 = B1_pos_arr[iB];
             B1_pos0 < B1_pos_arr[iB + 1]; B1_pos0++) {
      int jB1 = B1_idx_arr[B1_pos0];
      int A1_pos2 = (iB * A1_size) + jB1;
      for (int B2_pos2 = B2_pos_arr[B1_pos0];
               B2_pos2 < B2_pos_arr[B1_pos0 + 1]; B2_pos2++) {
        int kB3 = B2_idx_arr[B2_pos2];
        int A2_pos6 = (A1_pos2 * A2_size) + kB3;
        A_val_arr[A2_pos6] = B_val_arr[B2_pos2];
      }
    }
  }
  if (iC == iB) C0_pos = C0_end;
  iB++;
}
while (iB < B0_size) {
  for (int B1_pos1 = B1_pos_arr[iB];
           B1_pos1 < B1_pos_arr[iB + 1]; B1_pos1++) {
    int jB2 = B1_idx_arr[B1_pos1];
    int A1_pos3 = (iB * A1_size) + jB2;
    for (int B2_pos3 = B2_pos_arr[B1_pos1];
             B2_pos3 < B2_pos_arr[B1_pos1 + 1]; B2_pos3++) {
      int kB4 = B2_idx_arr[B2_pos3];
      int A2_pos7 = (A1_pos3 * A2_size) + kB4;
      A_val_arr[A2_pos7] = B_val_arr[B2_pos3];
    }
  }
  iB++;
}
```

The same process can be repeated dimension by dimension

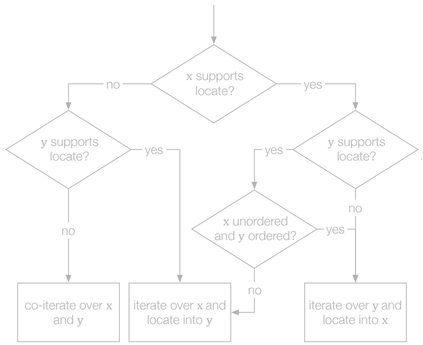
$$A_{ijk} = B_{ijk} + C_{ijk}$$

```
int iB = 0;
int C0_pos = C0_pos_arr[0];
while (C0_pos < C0_pos_arr[1]) {
  int iC = C0_idx_arr[C0_pos];
  int C0_end = C0_pos + 1;
  if (iC == iB)
    while ((C0_end < C0_pos_arr[1]) && (C0_idx_arr[C0_end] == iB)) {
      C0_end++;
    }
  if (iC == iB) {
    int B1_pos = B1_pos_arr[iB];
    int C1_pos = C0_pos;
    while ((B1_pos < B1_pos_arr[iB + 1]) && (C1_pos < C0_end)) {
      int jB = B1_idx_arr[B1_pos];
      int jC = C1_idx_arr[C1_pos];
      int j = min(jB, jC);
      int A1_pos = (iB * A1_size) + j;
      int C1_end = C1_pos + 1;
      if (jC == j)
        while ((C1_end < C0_end) && (C1_idx_arr[C1_end] == j)) {
          C1_end++;
        }
      if ((jB == j) && (jC == j)) {
        int B2_pos = B2_pos_arr[B1_pos];
        int C2_pos = C1_pos;
        while ((B2_pos < B2_pos_arr[B1_pos + 1]) && (C2_pos < C1_end)) {
          int kB = B2_idx_arr[B2_pos];
          int kC = C2_idx_arr[C2_pos];
          int k = min(kB, kC);
          int A2_pos = (A1_pos * A2_size) + k;
          if ((kB == k) && (kC == k)) {
            A_val_arr[A2_pos] = B_val_arr[B2_pos] + C_val_arr[C2_pos];
          } else if (kB == k) {
            A_val_arr[A2_pos] = B_val_arr[B2_pos];
          } else {
            A_val_arr[A2_pos] = C_val_arr[C2_pos];
          }
          if (kB == k) B2_pos++;
          if (kC == k) C2_pos++;
        }
        while (B2_pos < B2_pos_arr[B1_pos + 1]) {
          int kB0 = B2_idx_arr[B2_pos];
          int A2_pos0 = (A1_pos * A2_size) + kB0;
          A_val_arr[A2_pos0] = B_val_arr[B2_pos];
          B2_pos++;
        }
        while (C2_pos < C1_end) {
          int kC0 = C2_idx_arr[C2_pos];
          int A2_pos1 = (A1_pos * A2_size) + kC0;
          A_val_arr[A2_pos1] = C_val_arr[C2_pos];
          C2_pos++;
        }
      } else if (jB == j) {
        for (int B2_pos0 = B2_pos_arr[B1_pos];
                 B2_pos0 < B2_pos_arr[B1_pos + 1]; B2_pos0++) {
          int kB1 = B2_idx_arr[B2_pos0];
          int A2_pos2 = (A1_pos * A2_size) + kB1;
          A_val_arr[A2_pos2] = B_val_arr[B2_pos0];
        }
      } else {
        for (int C2_pos0 = C1_pos; C2_pos0 < C1_end; C2_pos0++) {
          int kC1 = C2_idx_arr[C2_pos0];
          int A2_pos3 = (A1_pos * A2_size) + kC1;
          A_val_arr[A2_pos3] = C_val_arr[C2_pos0];
        }
      }
      if (jB == j) B1_pos++;
      if (jC == j) C1_pos = C1_end;
    }
    while (B1_pos < B1_pos_arr[iB + 1]) {
      int jB0 = B1_idx_arr[B1_pos];
      int A1_pos0 = (iB * A1_size) + jB0;
      for (int B2_pos1 = B2_pos_arr[B1_pos];
               B2_pos1 < B2_pos_arr[B1_pos + 1]; B2_pos1++) {
        int kB2 = B2_idx_arr[B2_pos1];
        int A2_pos4 = (A1_pos0 * A2_size) + kB2;
        A_val_arr[A2_pos4] = B_val_arr[B2_pos1];
      }
      B1_pos++;
    }
    while (C1_pos < C0_end) {
      int jC0 = C1_idx_arr[C1_pos];
      int A1_pos1 = (iB * A1_size) + jC0;
      int C1_end0 = C1_pos + 1;
      while ((C1_end0 < C0_end) && (C1_idx_arr[C1_end0] == jC0)) {
        C1_end0++;
      }
      for (int C2_pos1 = C1_pos; C2_pos1 < C1_end0; C2_pos1++) {
        int kC2 = C2_idx_arr[C2_pos1];
        int A2_pos5 = (A1_pos1 * A2_size) + kC2;
        A_val_arr[A2_pos5] = C_val_arr[C2_pos1];
      }
      C1_pos = C1_end0;
    }
  } else {
    for (int B1_pos0 = B1_pos_arr[iB];
             B1_pos0 < B1_pos_arr[iB + 1]; B1_pos0++) {
      int jB1 = B1_idx_arr[B1_pos0];
      int A1_pos2 = (iB * A1_size) + jB1;
      for (int B2_pos2 = B2_pos_arr[B1_pos0];
               B2_pos2 < B2_pos_arr[B1_pos0 + 1]; B2_pos2++) {
        int kB3 = B2_idx_arr[B2_pos2];
        int A2_pos6 = (A1_pos2 * A2_size) + kB3;
        A_val_arr[A2_pos6] = B_val_arr[B2_pos2];
      }
    }
  }
  if (iC == iB) C0_pos = C0_end;
  iB++;
}
while (iB < B0_size) {
  for (int B1_pos1 = B1_pos_arr[iB];
           B1_pos1 < B1_pos_arr[iB + 1]; B1_pos1++) {
    int jB2 = B1_idx_arr[B1_pos1];
    int A1_pos3 = (iB * A1_size) + jB2;
    for (int B2_pos3 = B2_pos_arr[B1_pos1];
             B2_pos3 < B2_pos_arr[B1_pos1 + 1]; B2_pos3++) {
      int kB4 = B2_idx_arr[B2_pos3];
      int A2_pos7 = (A1_pos3 * A2_size) + kB4;
      A_val_arr[A2_pos7] = B_val_arr[B2_pos3];
    }
  }
  iB++;
}
```

Format Abstraction & Code Generation

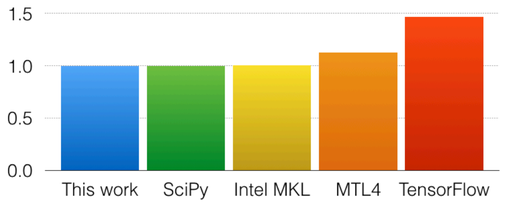
DIA
Dense
Range
Offset

Mode-generic tensor
Compressed
Singleton
Dense
Dense



Evaluation

	This work	Kjolstad et al. 2017	Intel MKL	SciPy	MTL4	MATLAB Tensor Toolbox	TensorFlow
Sparse vector	✔	✔	✔	✔	✔	✔	✔
Hash map vector	✔			✔			
Coordinate matrix	✔		✔	✔	✔	✔	✔
CSR	✔	✔	✔	✔	✔	✔	
DCSR	✔	✔					
ELL	✔				✔		
DIA	✔		✔	✔			
BCSR	✔	✔	✔	✔	✔		
CSB	✔						
DOK				✔			
LIL				✔			
Skyline	✔		✔				
Banded	✔				✔		
Coordinate tensor	✔					✔	✔
CSF	✔	✔					
Mode-generic	✔						



Our technique supports a wide range of disparate tensor formats

	taco
	This work

Our technique supports a wide range of disparate tensor formats

	taco	
	This work	**[Kjolstad et al. 2017]**

Our technique supports a wide range of disparate tensor formats

	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					

Our technique supports a wide range of disparate tensor formats

	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					
Sparse vector							
Hash map vector							
Coordinate matrix							
CSR							
DCSR							
ELL							
DIA							
BCSR							
CSB							
DOK							
LIL							
Skyline							
Banded							
Coordinate tensor							
CSF							
Mode-generic							

Our technique supports a wide range of disparate tensor formats

	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					
Sparse vector			✓	✓	✓	✓	✓
Hash map vector				✓			
Coordinate matrix			✓	✓	✓	✓	✓
CSR			✓	✓	✓	✓	
DCSR							
ELL					✓		
DIA			✓	✓			
BCSR			✓	✓	✓		
CSB							
DOK				✓			
LIL				✓			
Skyline			✓				
Banded					✓		
Coordinate tensor						✓	✓
CSF							
Mode-generic							

Our technique supports a wide range of disparate tensor formats

	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					
Sparse vector		✓	✓	✓	✓	✓	✓
Hash map vector				✓			
Coordinate matrix			✓	✓	✓	✓	✓
CSR		✓	✓	✓	✓	✓	
DCSR		✓					
ELL					✓		
DIA			✓	✓			
BCSR		✓	✓	✓	✓		
CSB							
DOK				✓			
LIL				✓			
Skyline			✓				
Banded					✓		
Coordinate tensor						✓	✓
CSF		✓					
Mode-generic							

Our technique supports a wide range of disparate tensor formats

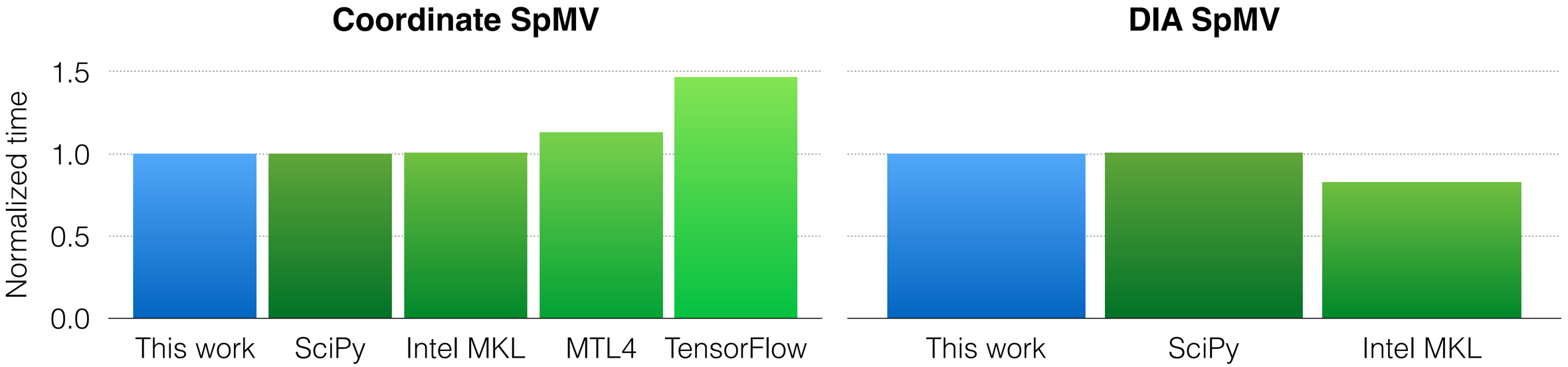
	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					
Sparse vector	✓	✓	✓	✓	✓	✓	✓
Hash map vector	✓			✓			
Coordinate matrix	✓		✓	✓	✓	✓	✓
CSR	✓	✓	✓	✓	✓	✓	
DCSR	✓	✓					
ELL	✓				✓		
DIA	✓		✓	✓			
BCSR	✓	✓	✓	✓	✓		
CSB	✓						
DOK				✓			
LIL				✓			
Skyline			✓				
Banded					✓		
Coordinate tensor	✓					✓	✓
CSF	✓	✓					
Mode-generic	✓						

Our technique supports a wide range of disparate tensor formats

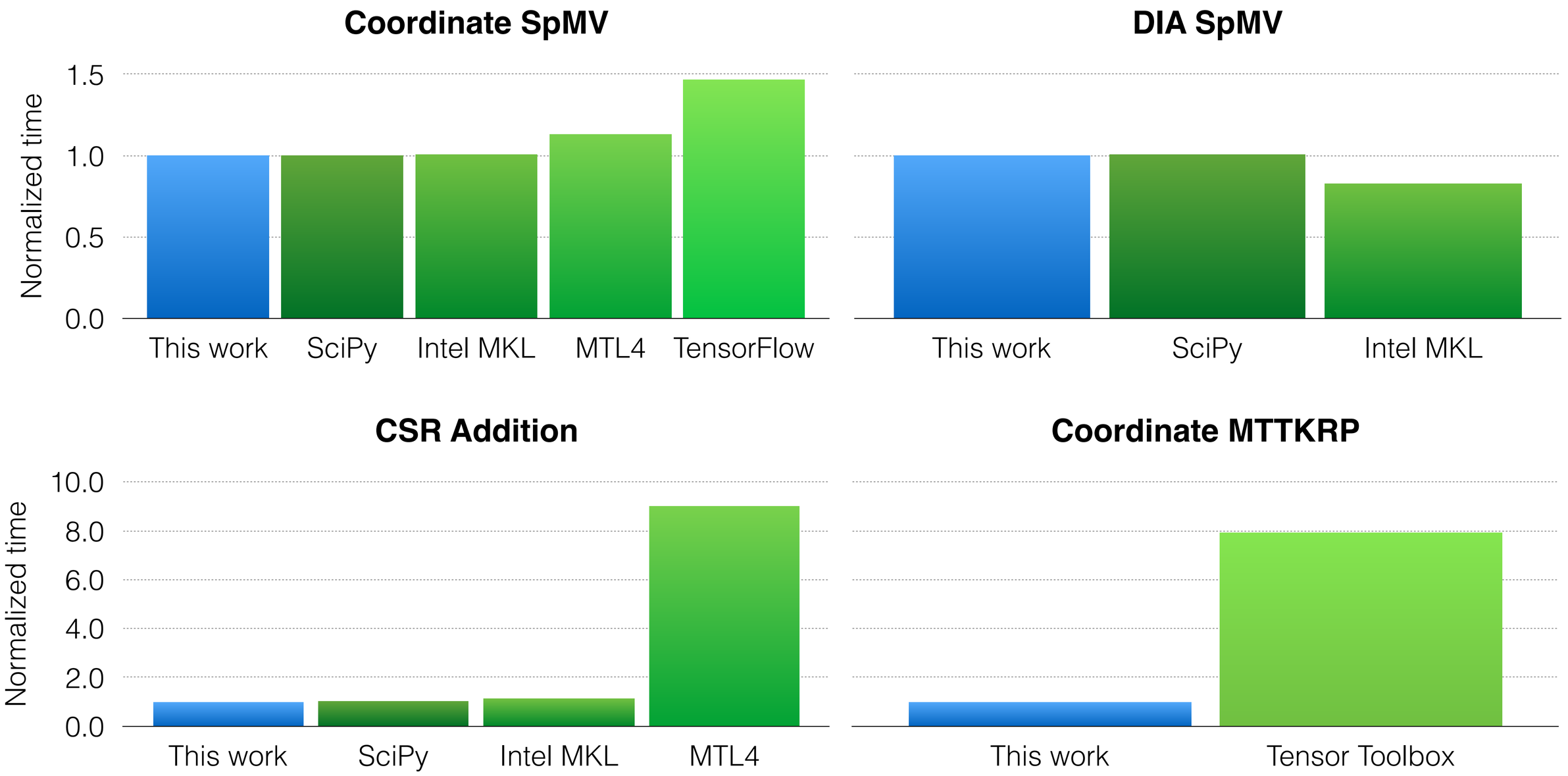
	taco		Intel MKL	SciPy	MTL4	Tensor Toolbox	TensorFlow
	This work	[Kjolstad et al. 2017]					
Sparse vector	✓	✓	✓	✓	✓	✓	✓
Hash map vector	✓			✓			
Coordinate matrix	✓		✓	✓	✓	✓	✓
CSR	✓	✓	✓	✓	✓	✓	
DCSR	✓	✓					
ELL	✓				✓		
DIA	✓		✓	✓			
BCSR	✓	✓	✓	✓	✓		
CSB	✓						
DOK				✓			
LIL				✓			
Skyline	✓		✓				
Banded	✓				✓		
Coordinate tensor	✓					✓	✓
CSF	✓	✓					
Mode-generic	✓						

Our technique generates efficient code

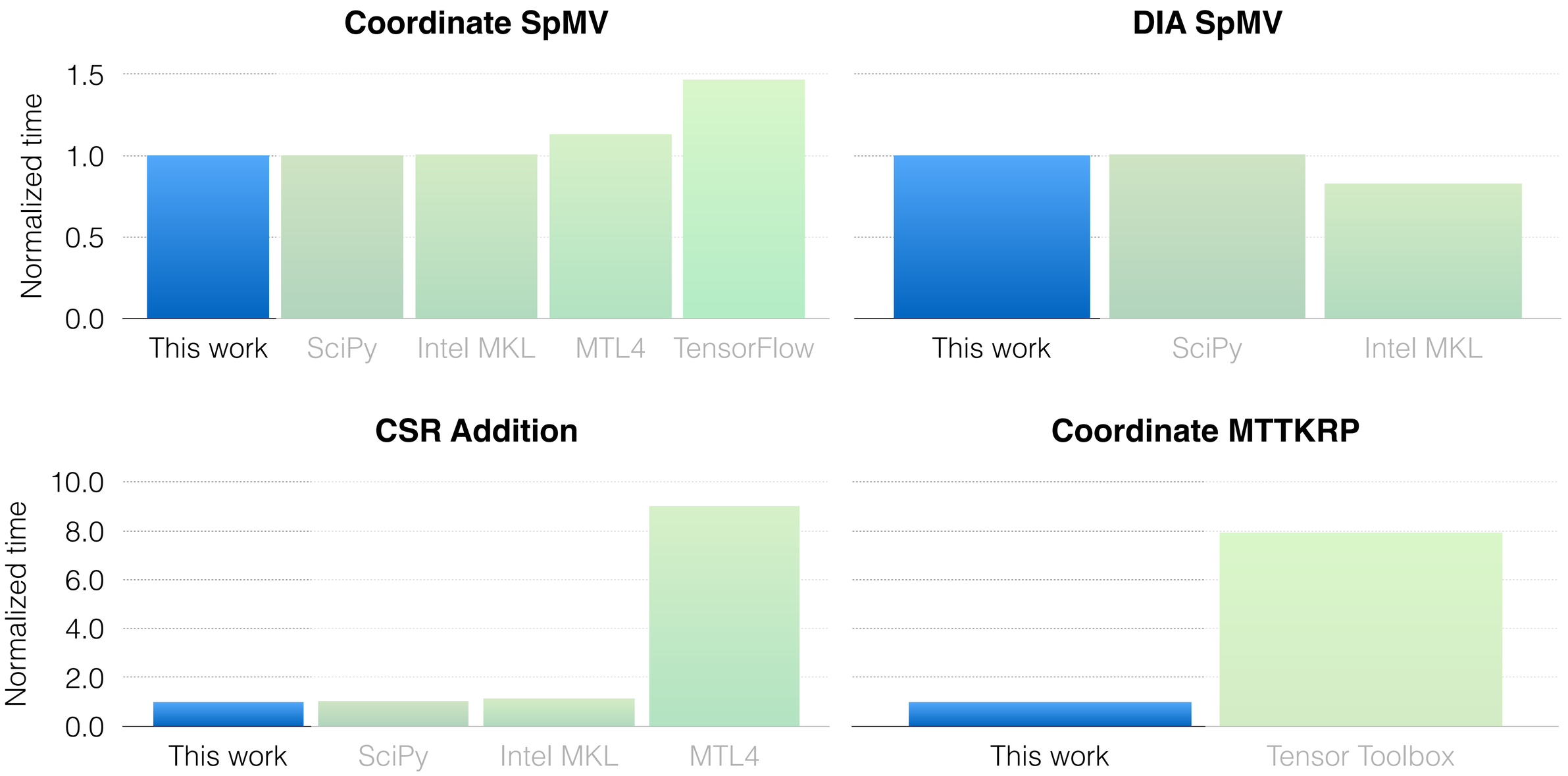
Our technique generates efficient code



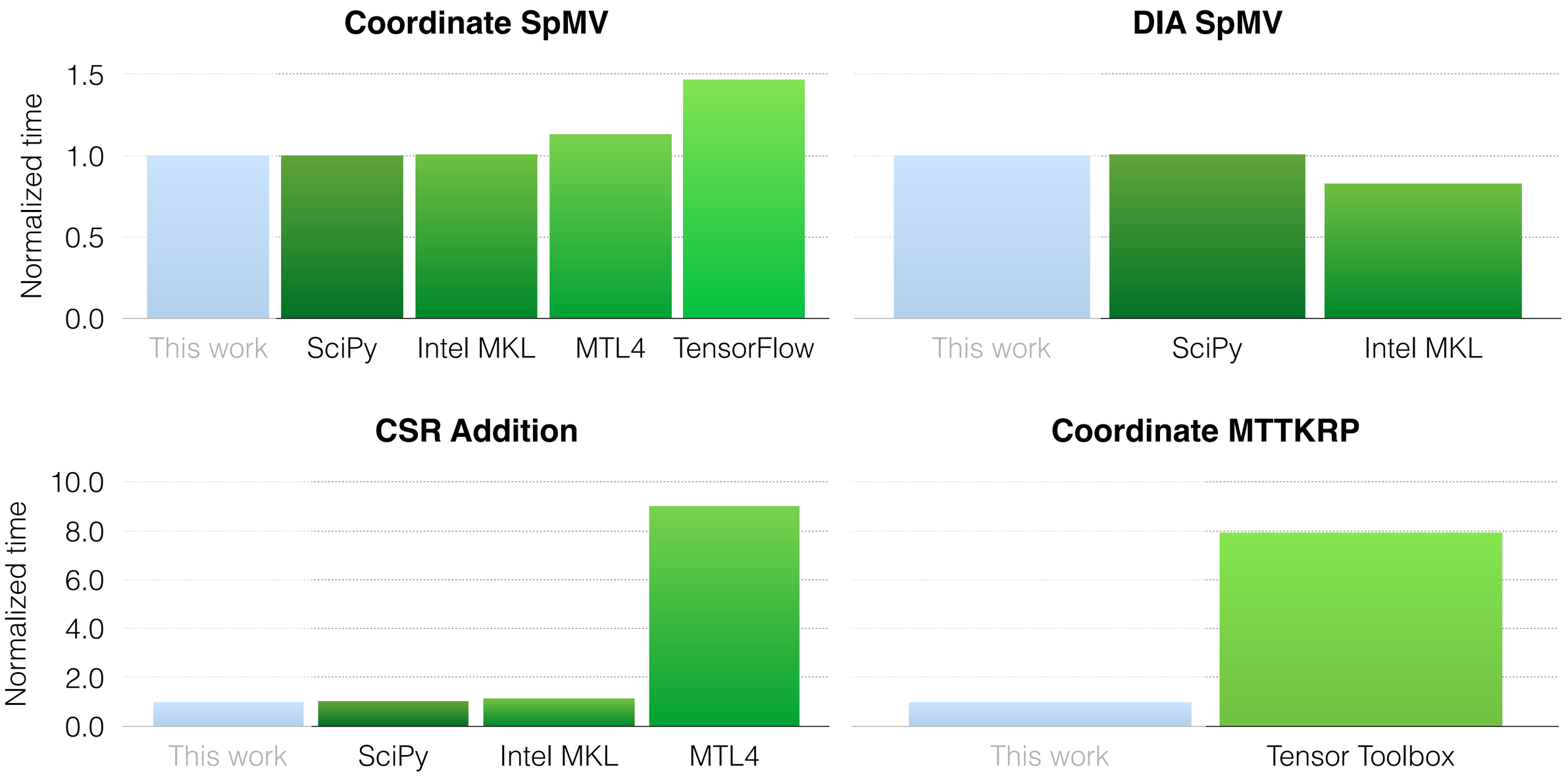
Our technique generates efficient code



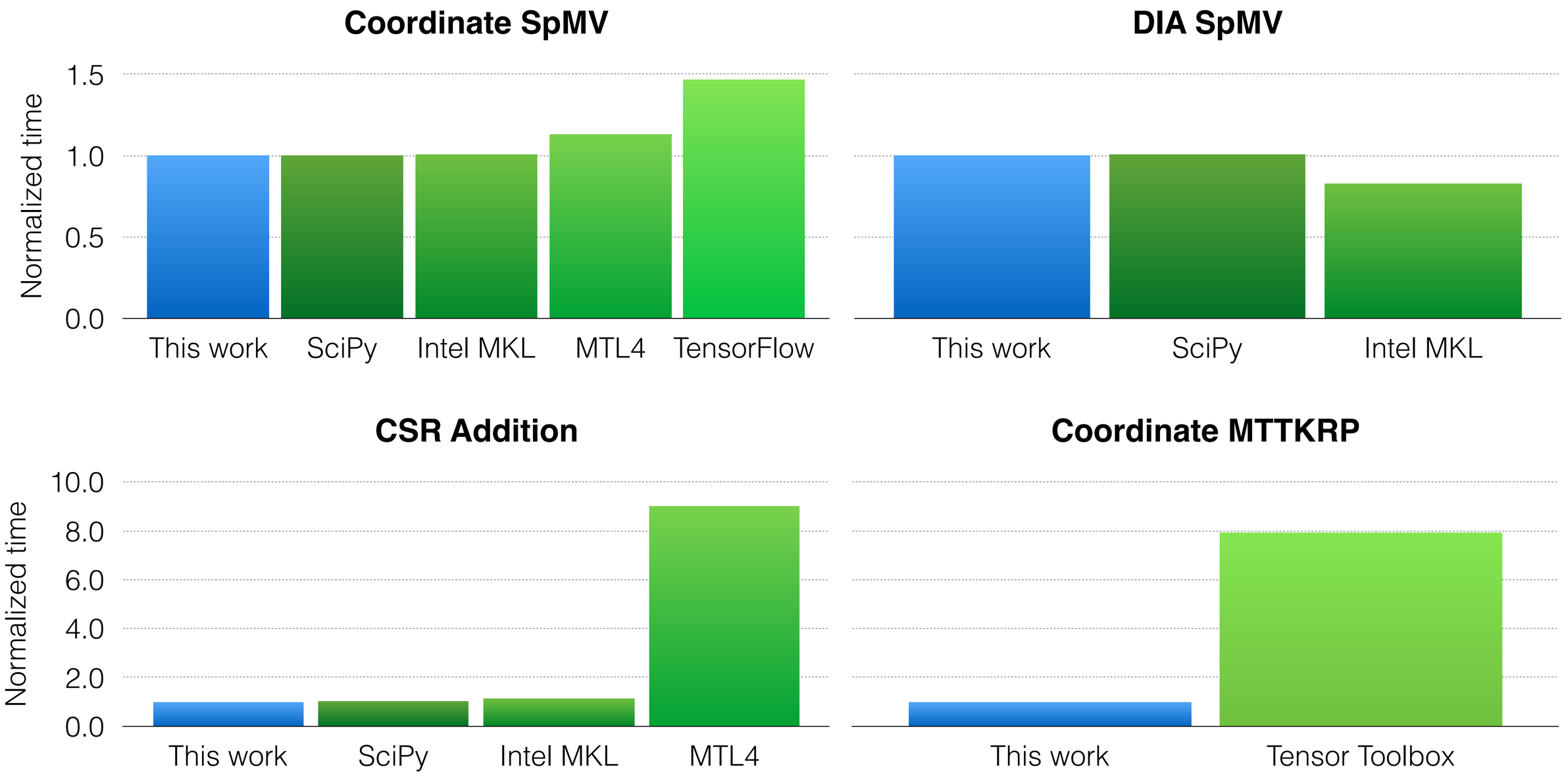
Our technique generates efficient code



Our technique generates efficient code



Our technique generates efficient code



In conclusion…

We can automatically generate kernels that compute with disparate tensor formats

In conclusion…

We can automatically generate kernels that compute with disparate tensor formats

Adding support for even more tensor formats is straightforward

In conclusion…

We can automatically generate kernels that compute with disparate tensor formats

Adding support for even more tensor formats is straightforward

Supporting many disparate tensor formats is essential for performance

In conclusion…

We can automatically generate kernels that compute with disparate tensor formats

Adding support for even more tensor formats is straightforward

Supporting many disparate tensor formats is essential for performance

tensor-compiler.org

This work supported by: