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.

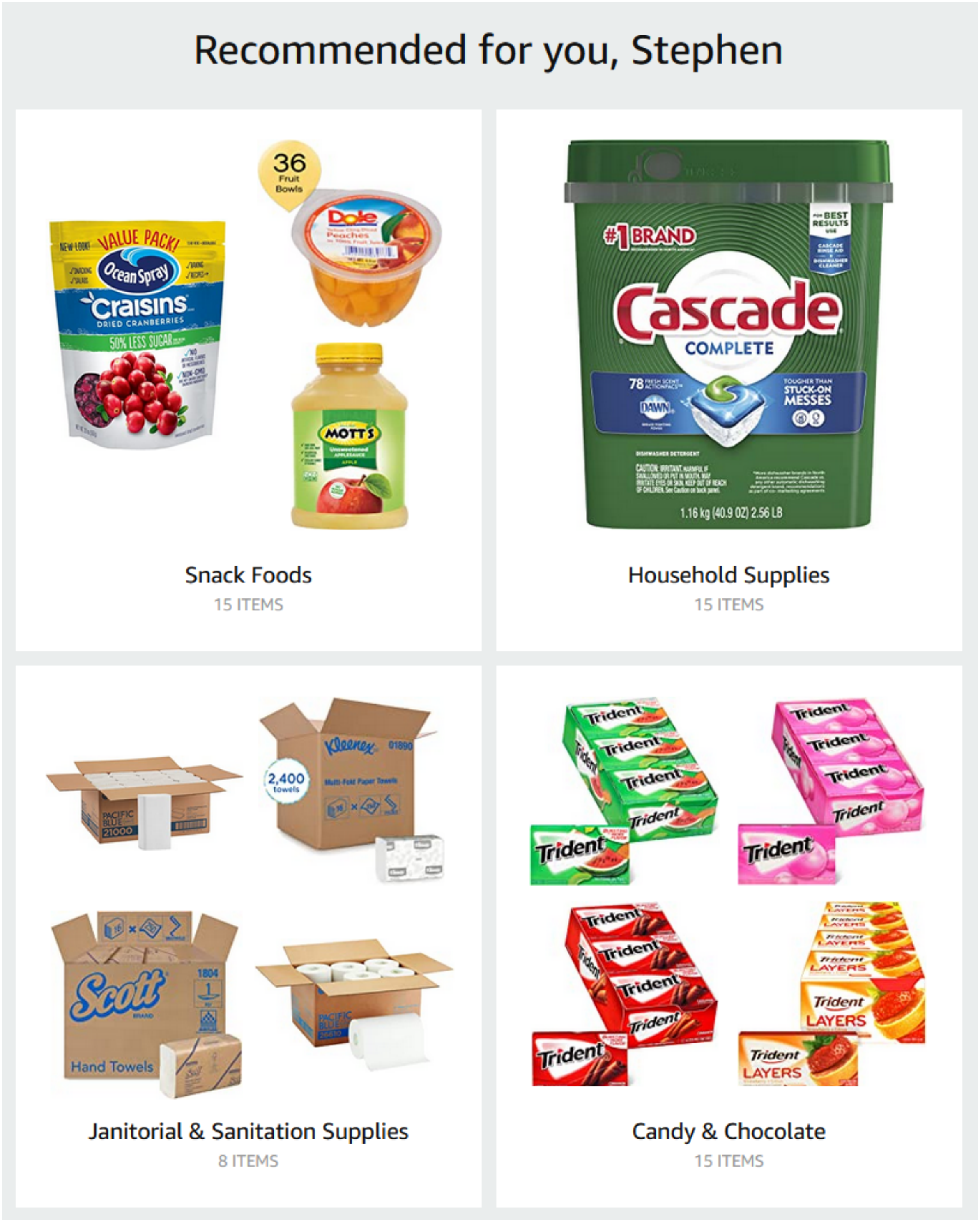
Sarah

Malfunctioned within a month. Waste of $.

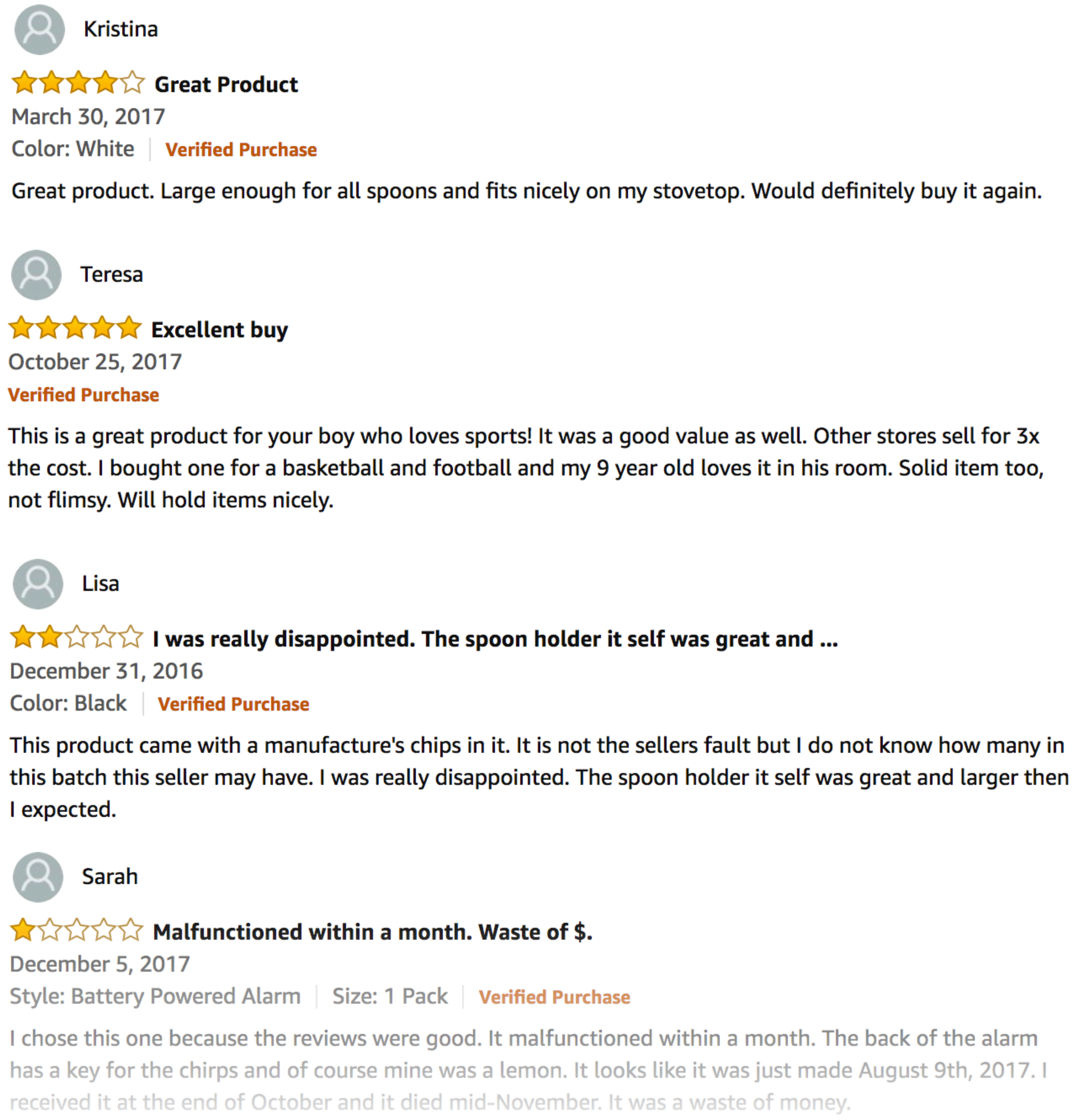
December 5, 2017

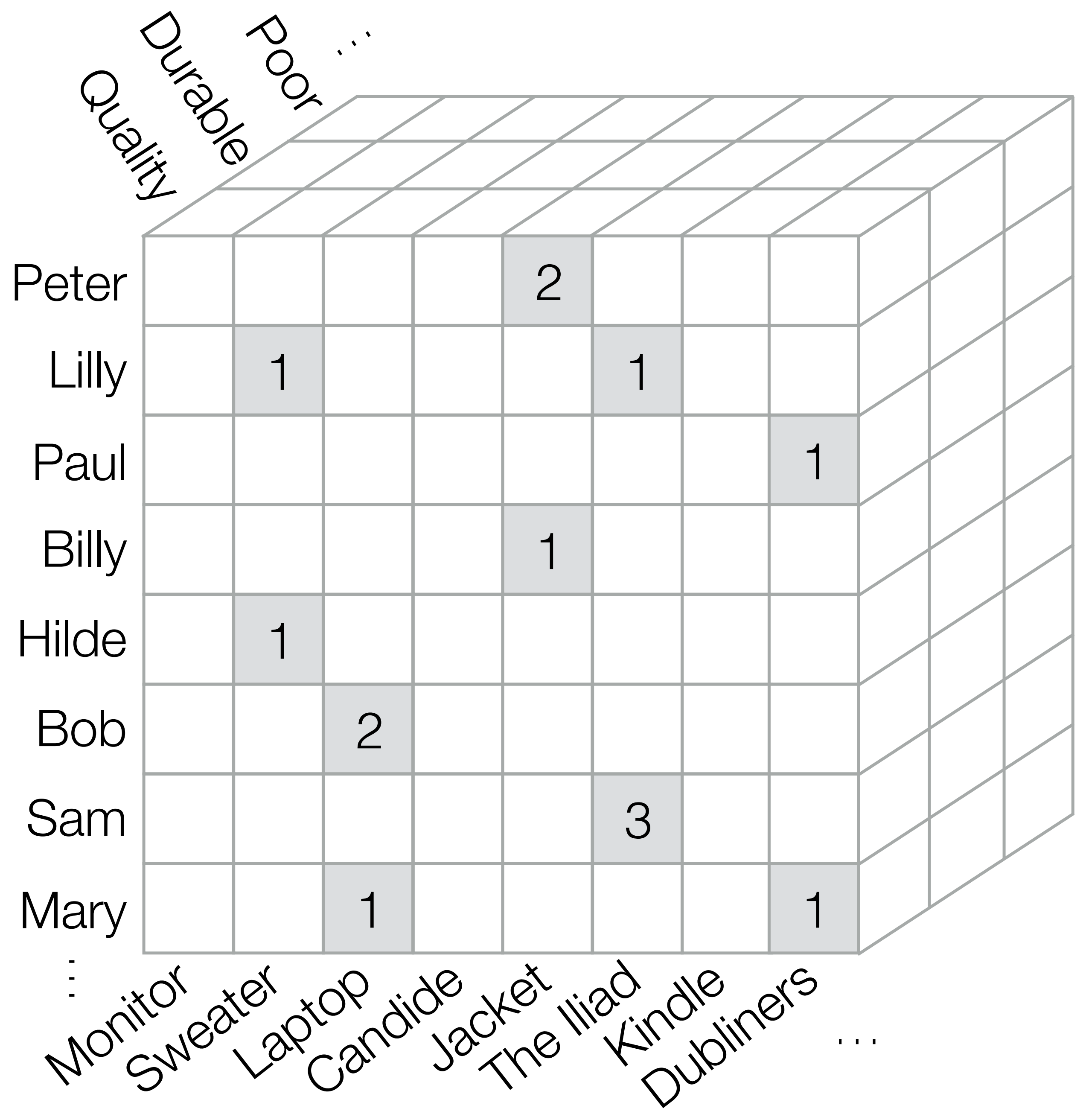
Style: Battery Powered Alarm | Size: 1 Pack | Verified Purchase

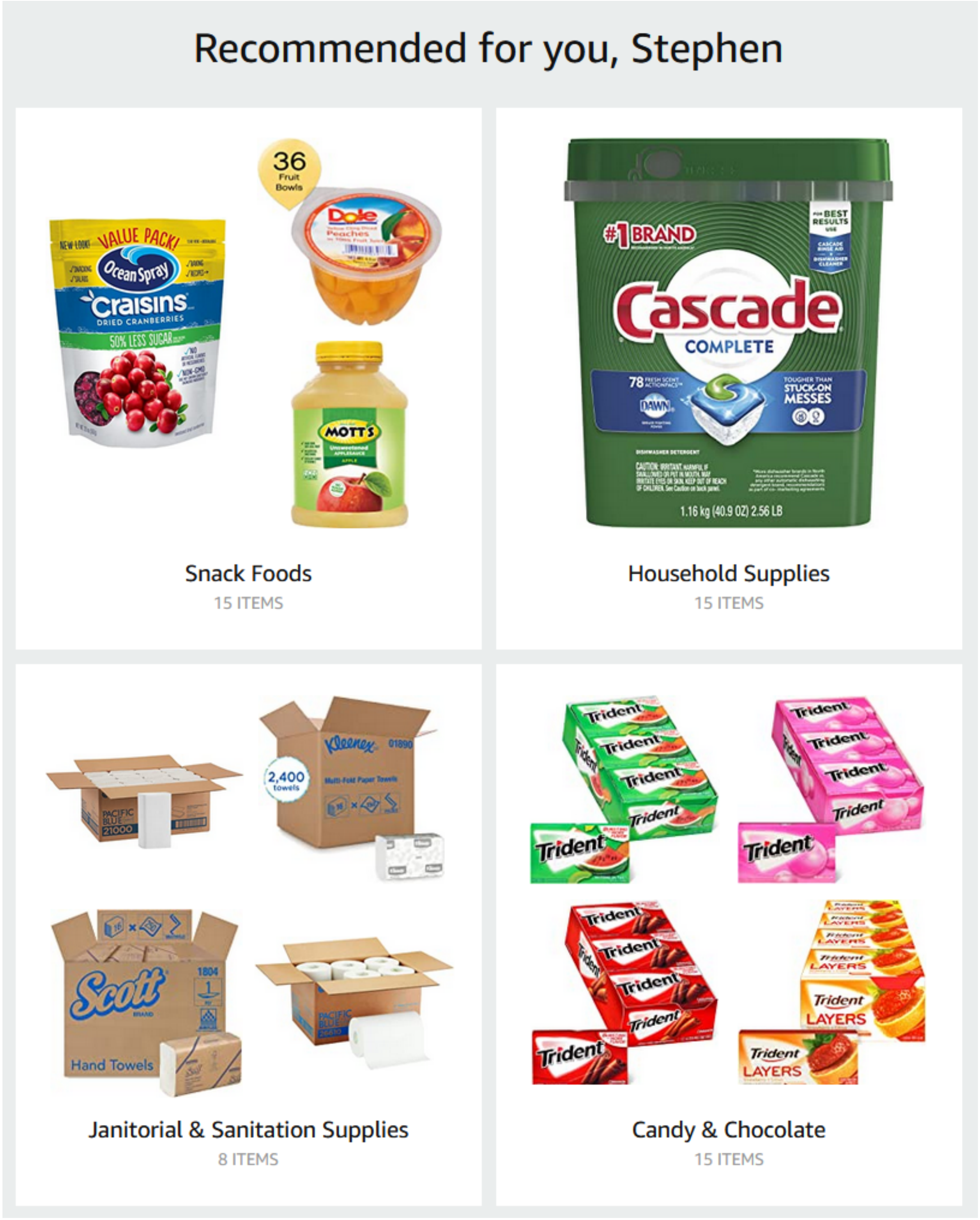
I chose this one because the reviews were good. It malfunctioned within a month. The back of the alarm has a key for the chirps and of course mine was a lemon. It looks like it was just made August 9th, 2017. I received it at the end of October and it died mid-November. It was a waste of money.



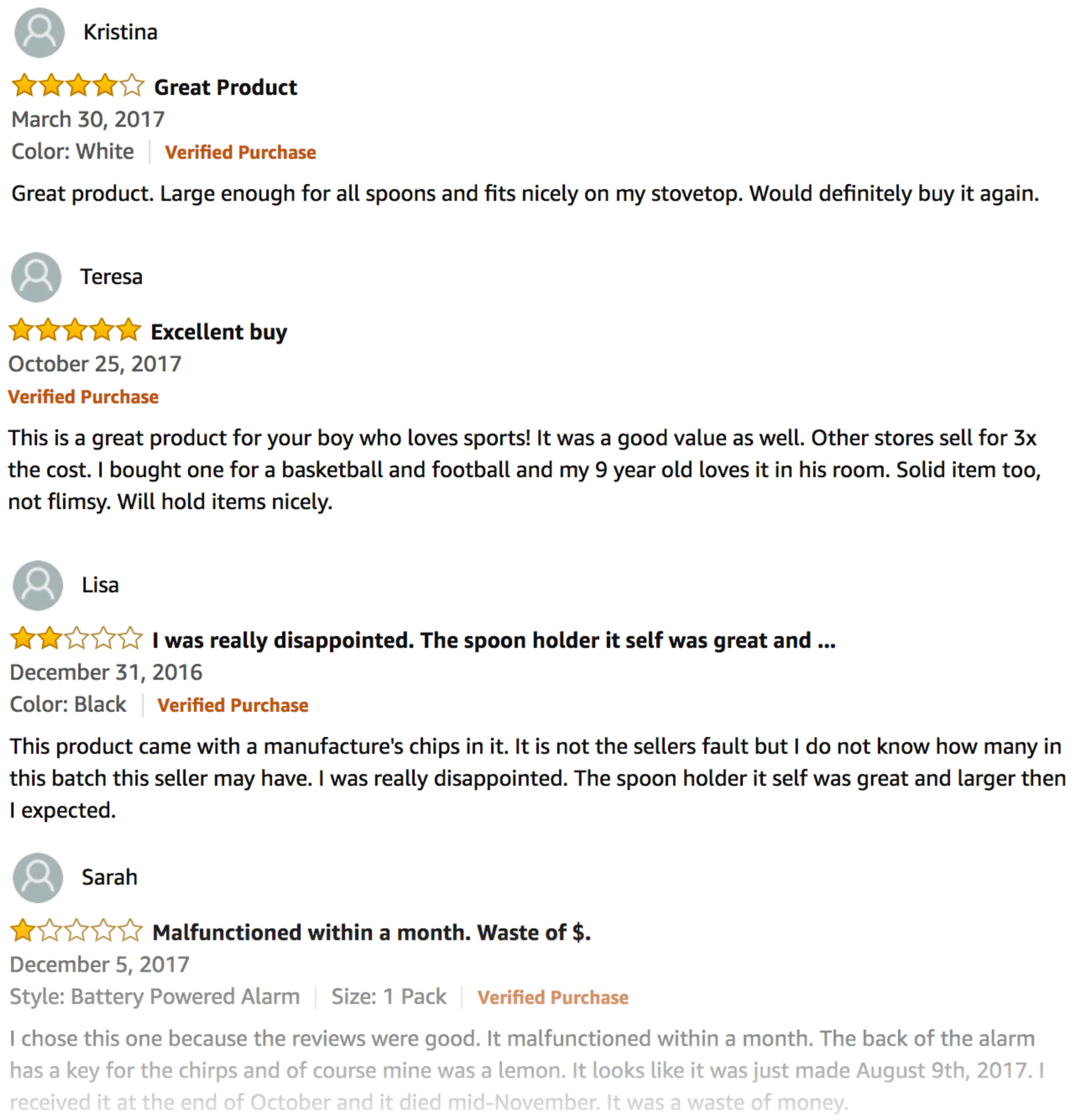
Sparse tensors are a natural way of representing real-world data

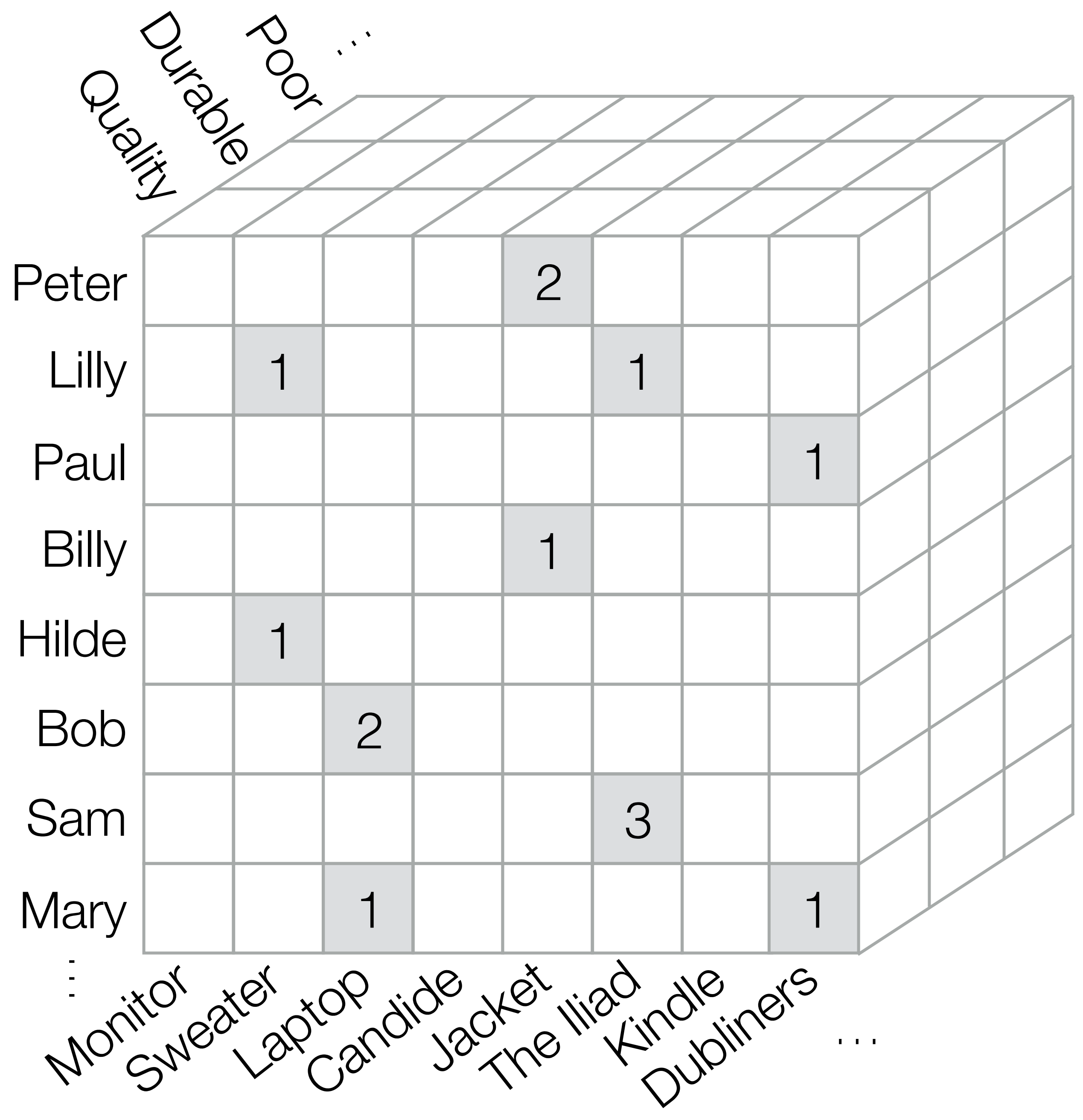


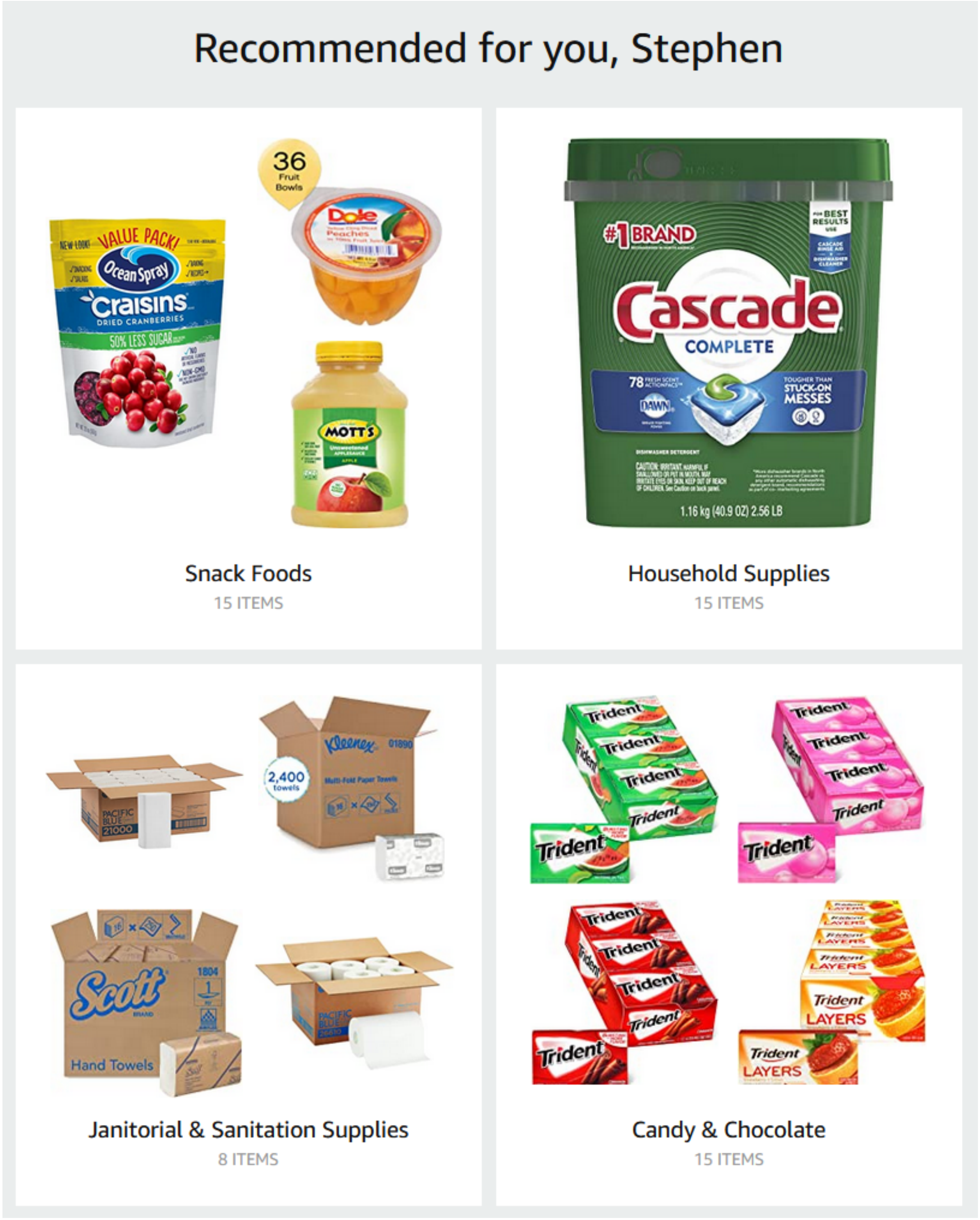




Sparse tensors are a natural way of representing real-world data







Dense storage: 107 exabytes
Sparse storage: 13 gigabytes

There exists many different formats for storing tensors

DNS CSB CSR

DCSR BCSR

COO USS ELL

BCOO CSC

DIA

BDIA DCSC

LIL

SELL SKY BELL

LNK MSR

DOK BND

VBR JAD

There exists many different formats for storing tensors

DNS

CSR

CSB

DCSR

BCSR

COO

USS

ELL

BCOO

CSC

DIA

BDIA

DCSC

LIL

SELL

SKY

BELL

Efficient insertions

LNK

MSR

DOK

BND

VBR

JAD

There exists many different formats for storing tensors

DNS

CSR

CSB

DCSR

BCSR

COO

ELL

USS

Structured stencils

BCOO

CSC

DIA

BDIA

DCSC

LIL

SKY

SELL

BELL

MSR

DOK

LNK

BND

JAD

VBR

There exists many different formats for storing tensors

DNS CSB CSR

DCSR BCSR

COO ELL USS

Unstructured mesh simulations

BCOO CSC

DIA

BDIA DCSC

LIL SELL

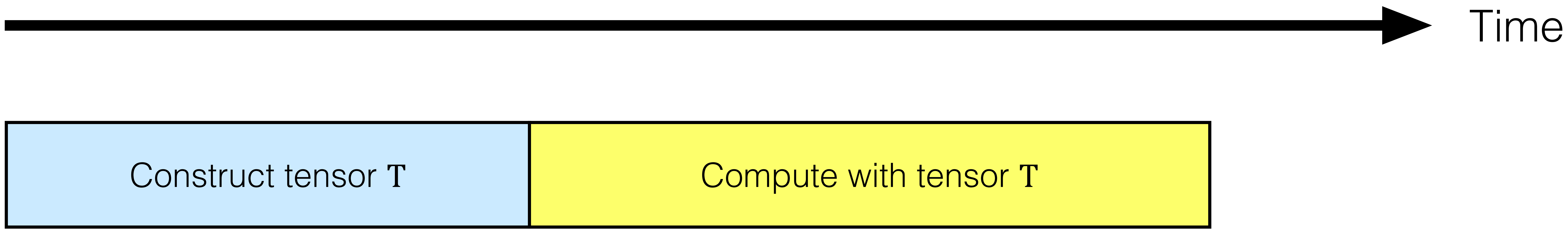
SKY BELL

MSR LNK

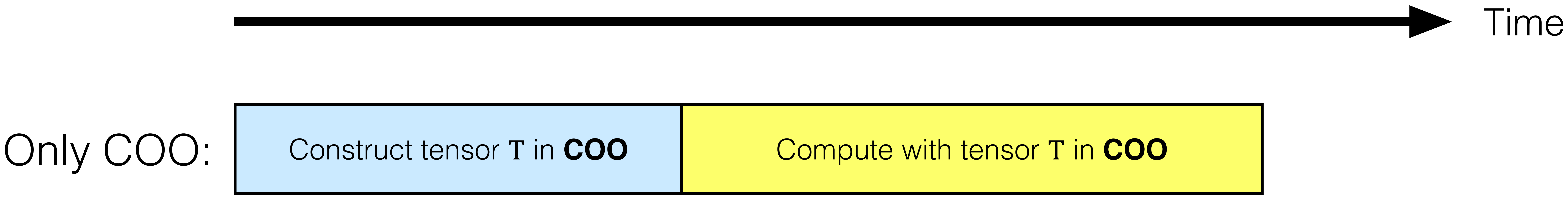
DOK BND

JAD VBR

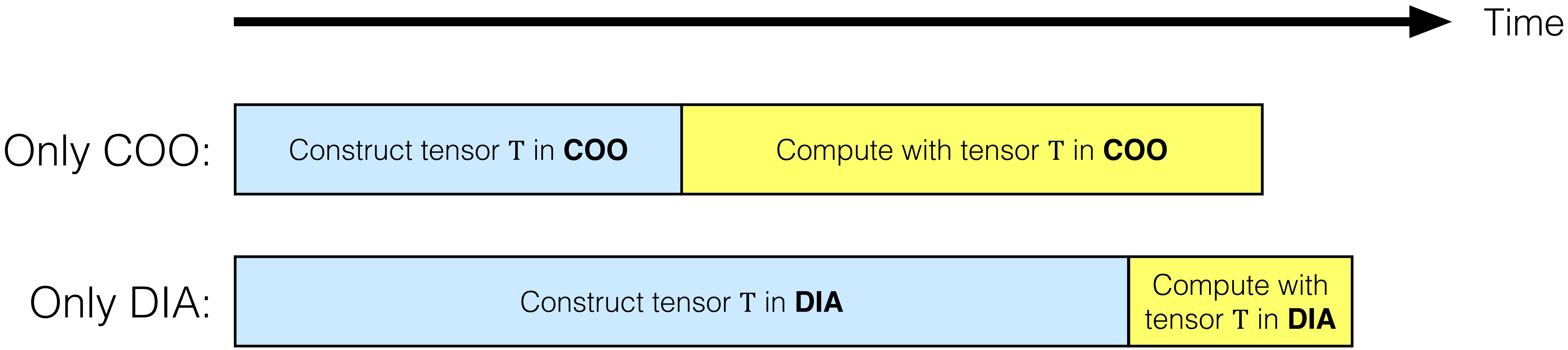
Applications must work with tensors in different formats for performance



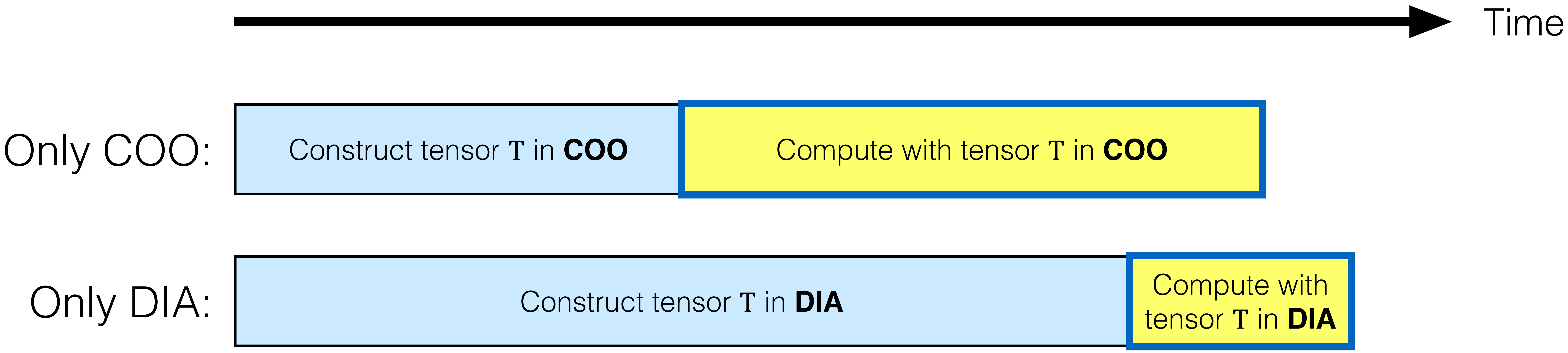
Applications must work with tensors in different formats for performance



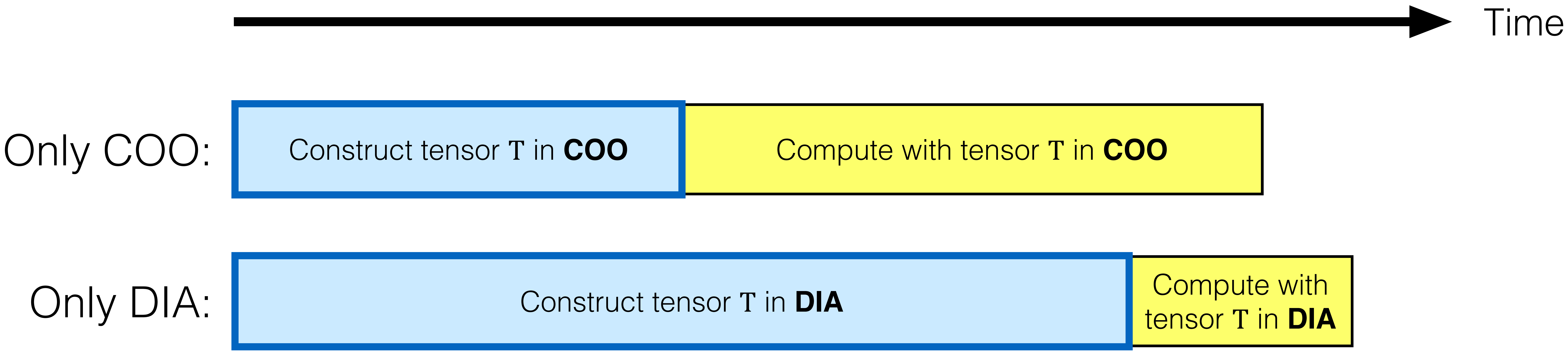
Applications must work with tensors in different formats for performance



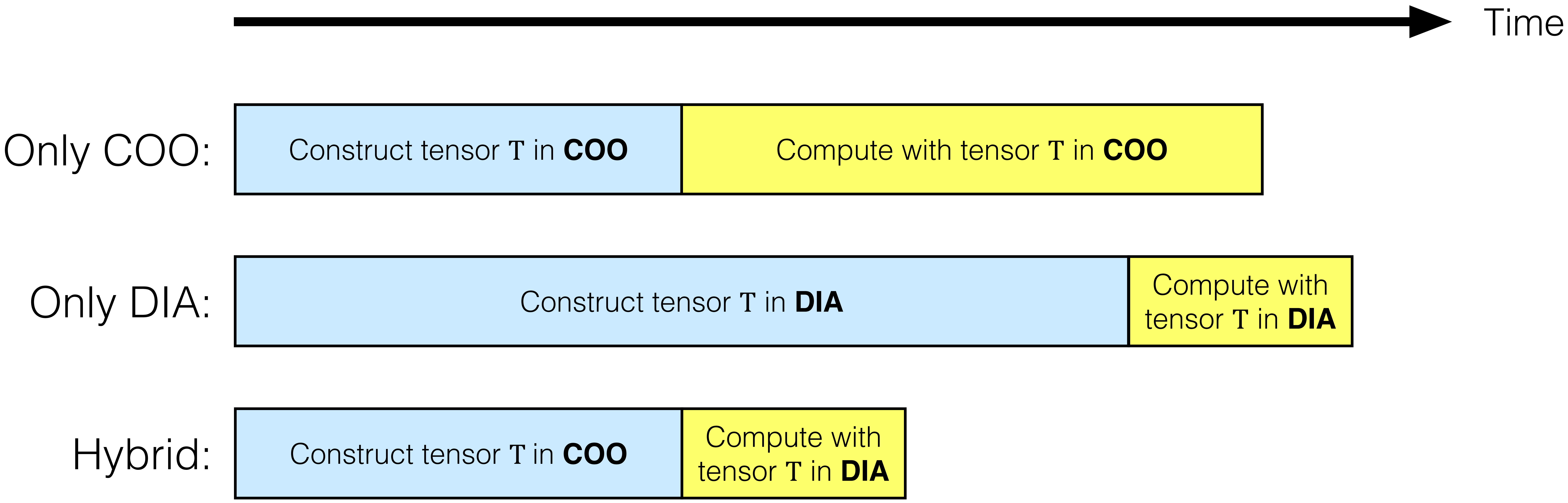
Applications must work with tensors in different formats for performance



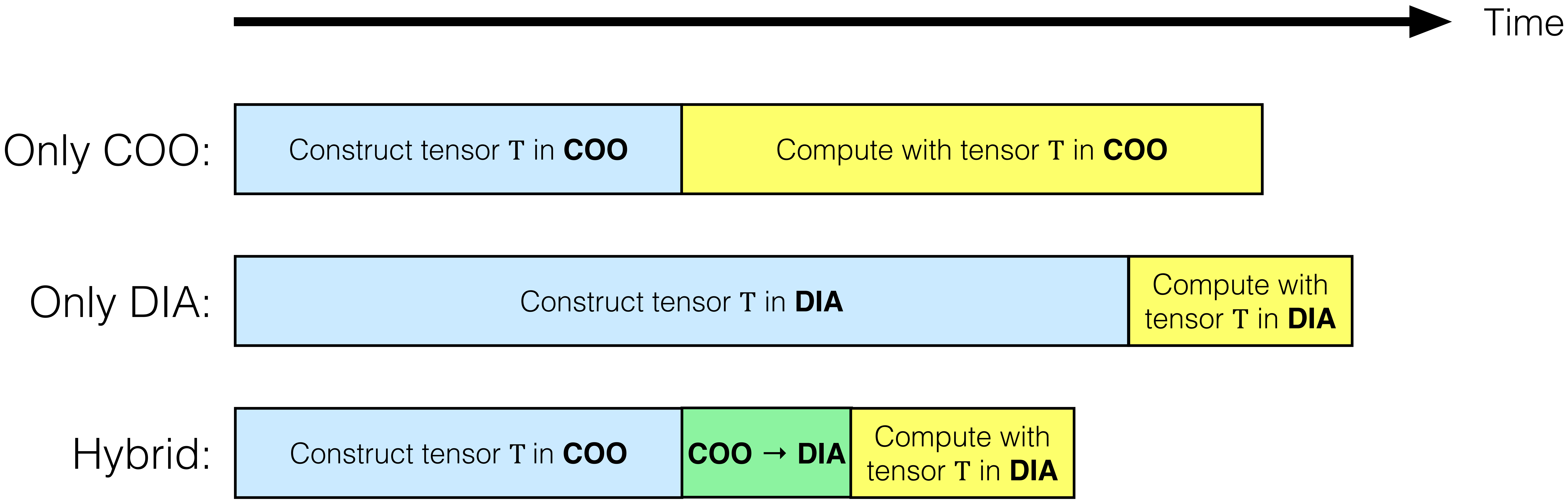
Applications must work with tensors in different formats for performance



Applications must work with tensors in different formats for performance



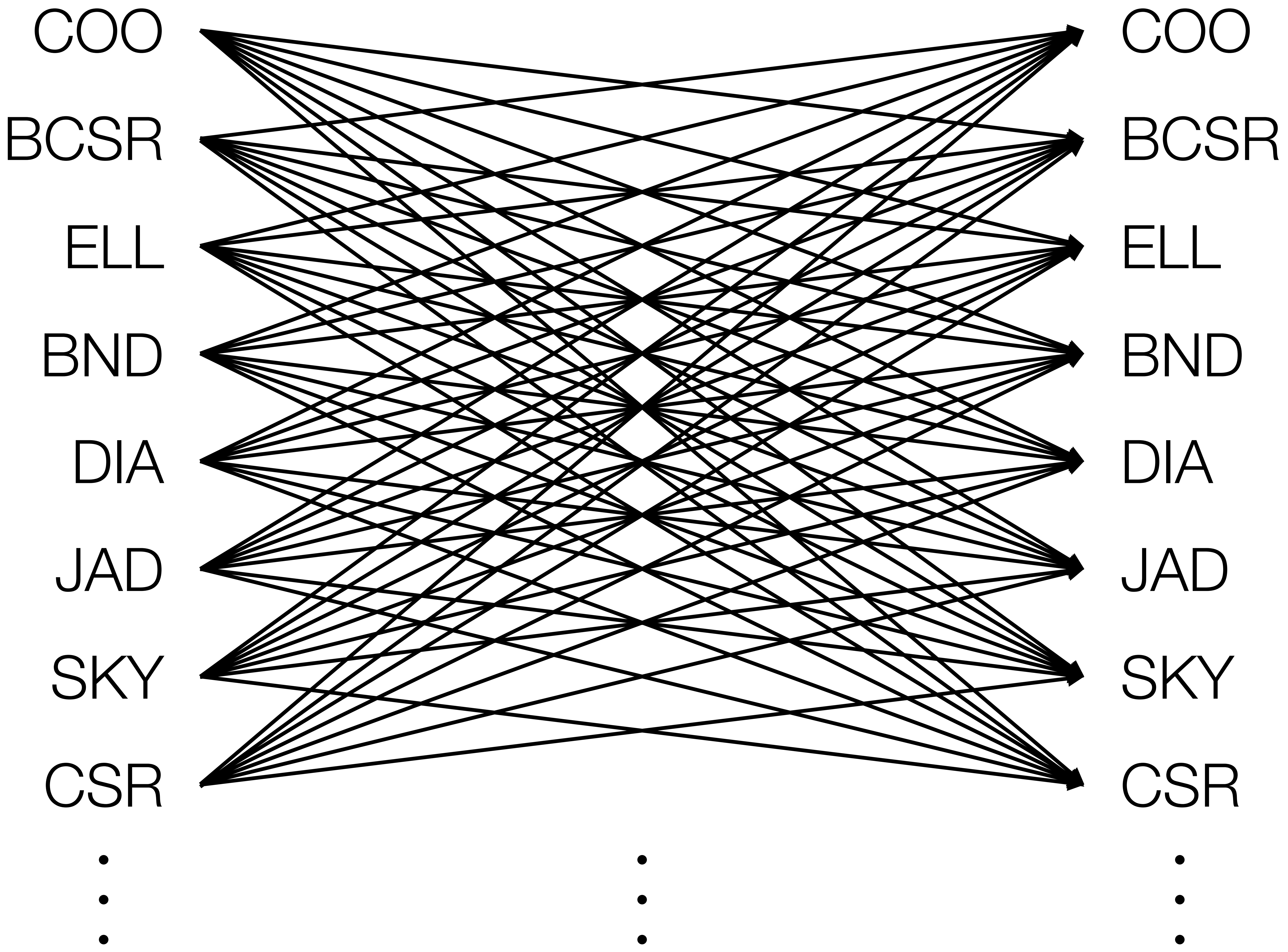
Applications must work with tensors in different formats for performance



Manually implementing support for efficient conversion between all combinations of formats is infeasible

COO	COO
BCSR	BCSR
ELL	ELL
BND	BND
DIA	DIA
JAD	JAD
SKY	SKY
CSR	CSR
⋮	⋮

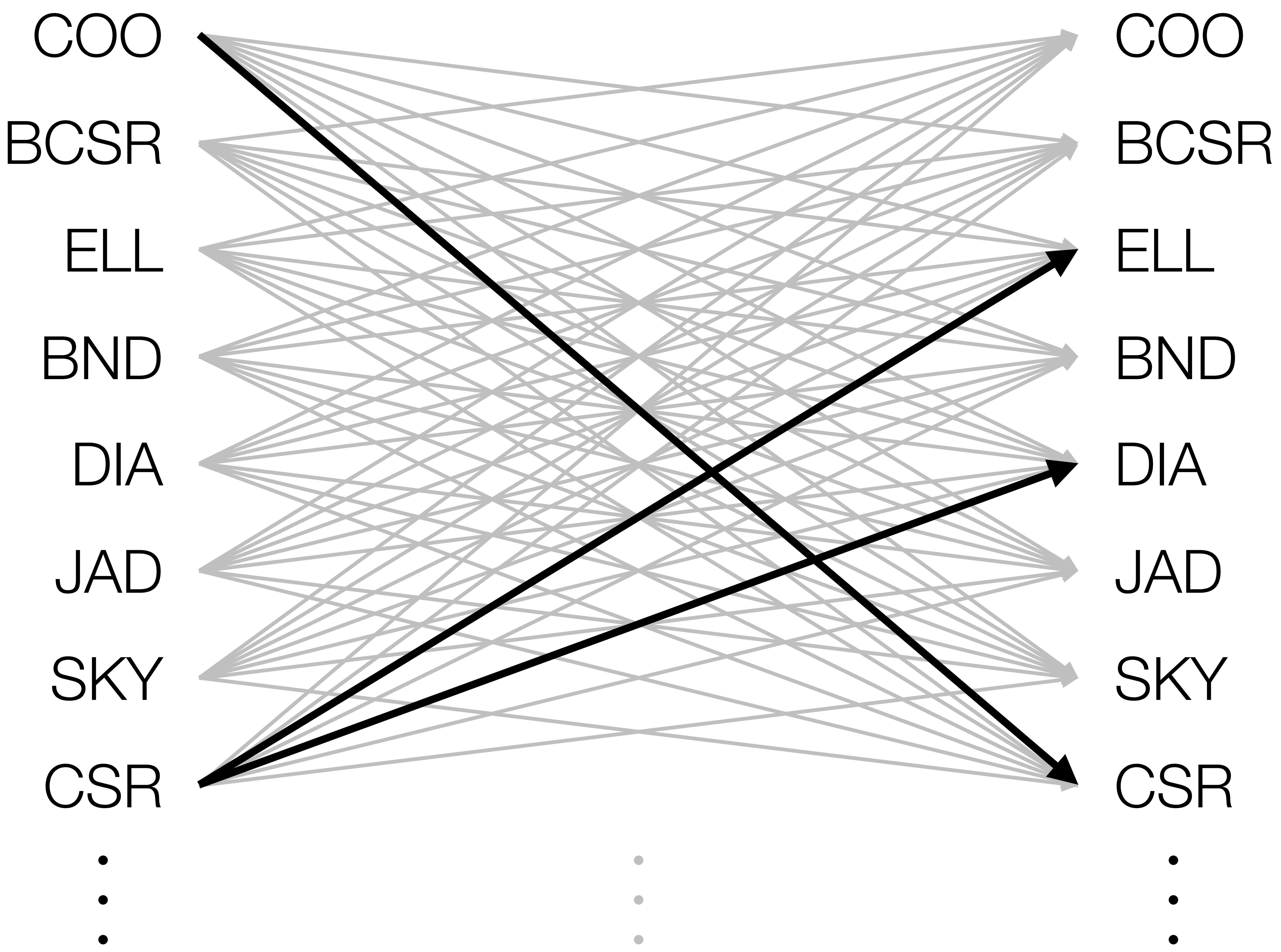
Manually implementing support for efficient conversion between all combinations of formats is infeasible



Manually implementing support for efficient conversion between all combinations of formats is infeasible

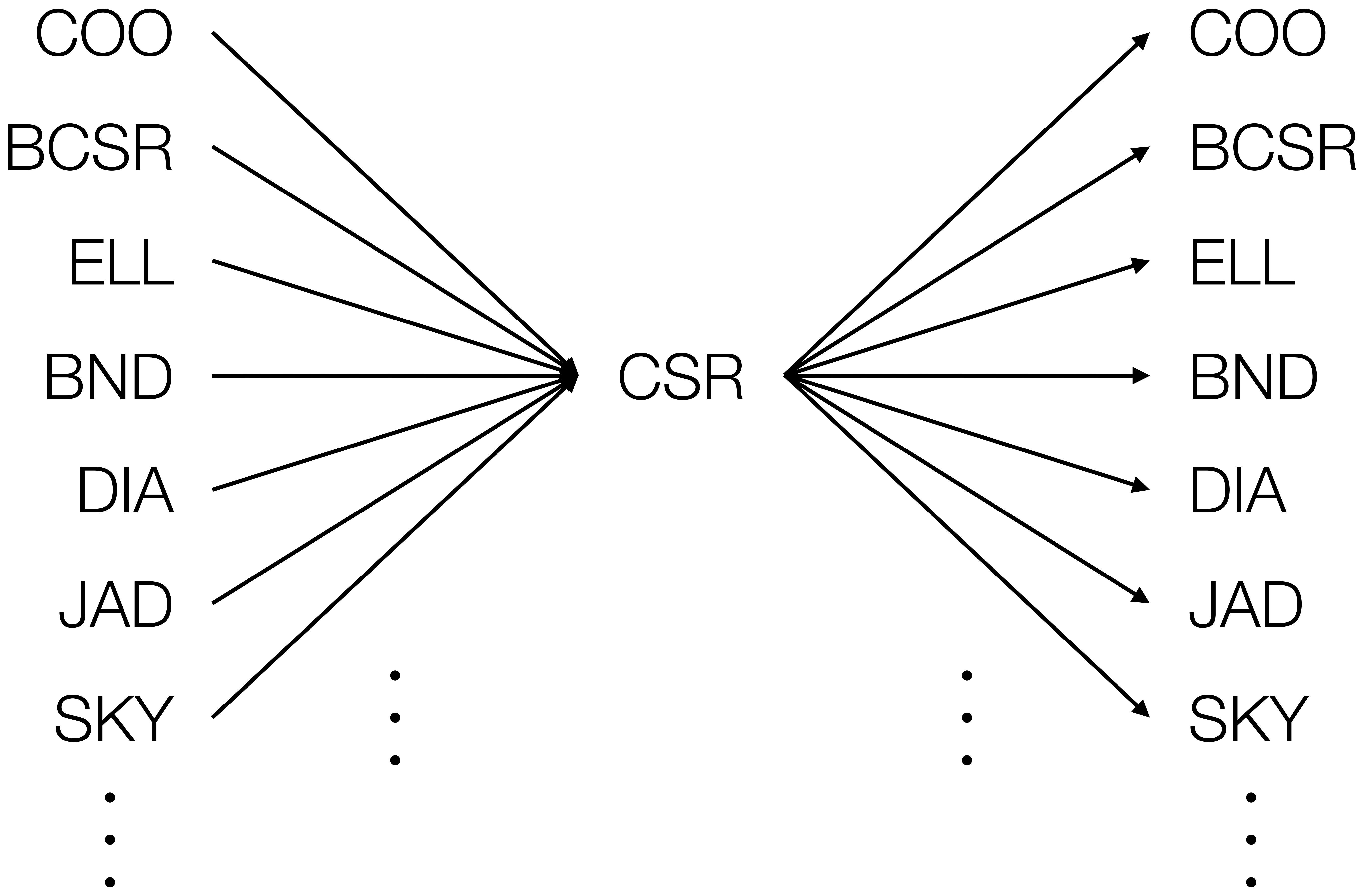
```
int K = 0;
for (int i = 0; i < N; i++) {
  int ncols = A_pos[i+1] - A_pos[i];
  K = max(K, ncols);
}
int* B_crd = new int[K * N]();
double* B_vals = new double[K * N]();
for (int i = 0; i < N; i++) {
  int count = 0;
  for (int pA2 = A_pos[i];
           pA2 < A_pos[i+1]; pA2++) {
    int j = A_crd[pA2];
    int k = count++;
    int pB2 = k * N + i;
    B_crd[pB2] = j;
    B_vals[pB2] = A_vals[pA2];
}}
```

```
int count[N] = {0};
for (int pA1 = A_pos[0];
         pA1 < A_pos[1]; pA1++) {
  int i = A1_crd[pA1];
  count[i]++;
}
int* B_pos = new int[N + 1];
B_pos[0] = 0;
for (int i = 0; i < N; i++) {
  B_pos[i + 1] = B_pos[i] + count[i];
}
int* B_crd = new int[pos[N]];
double* B_vals = new double[pos[N]];
for (int pA1 = A_pos[0];
         pA1 < A_pos[1]; pA1++) {
  int i = A1_crd[pA1];
  int j = A2_crd[pA1];
  int pB2 = pos[i]++;
  B_crd[pB2] = j;
  B_vals[pB2] = A_vals[pA2];
}
for (int i = 0; i < N; i++) {
  B_pos[N - i] = B_pos[N - i - 1];
}
B_pos[0] = 0;
```

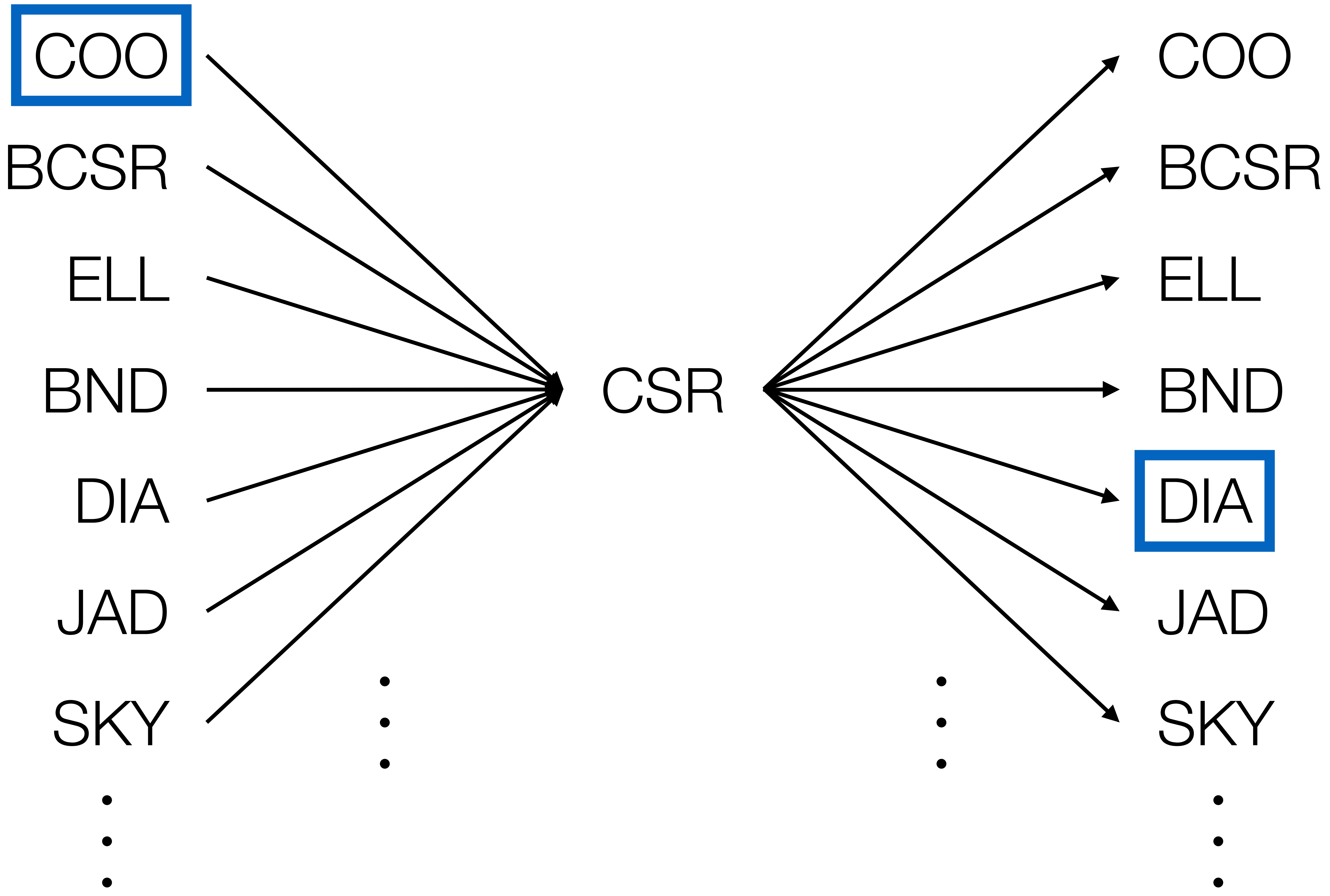



```
bool nz[2 * N - 1] = {0};
for (int i = 0; i < N; i++) {
  for (int pA2 = A_pos[i];
           pA2 < A_pos[i+1]; pA2++) {
    int j = A_crd[pA2];
    int k = j - i;
    nz[k + N - 1] = true;
}}
int* B_perm = new int[2 * N - 1];
int K = 0;
for (int i = -N + 1; i < N; i++) {
  if (nz[i + N - 1])
    B_perm[K++] = i;
}
double* B_vals = new double[K * N]();
int* B_rperm = new int[2 * N - 1];
for (int i = 0; i < K; i++) {
  B_rperm[B_perm[i] + N - 1] = i;
}
for (int i = 0; i < N; i++) {
  for (int pA2 = A_pos[i];
           pA2 < A_pos[i+1]; pA2++) {
    int j = A_crd[pA2];
    int k = j - i;
    int pB1 = B_rperm[k + N - 1];
    int pB2 = pB1 * N + i;
    B_vals[pB2] = A_vals[pA2];
}}
```

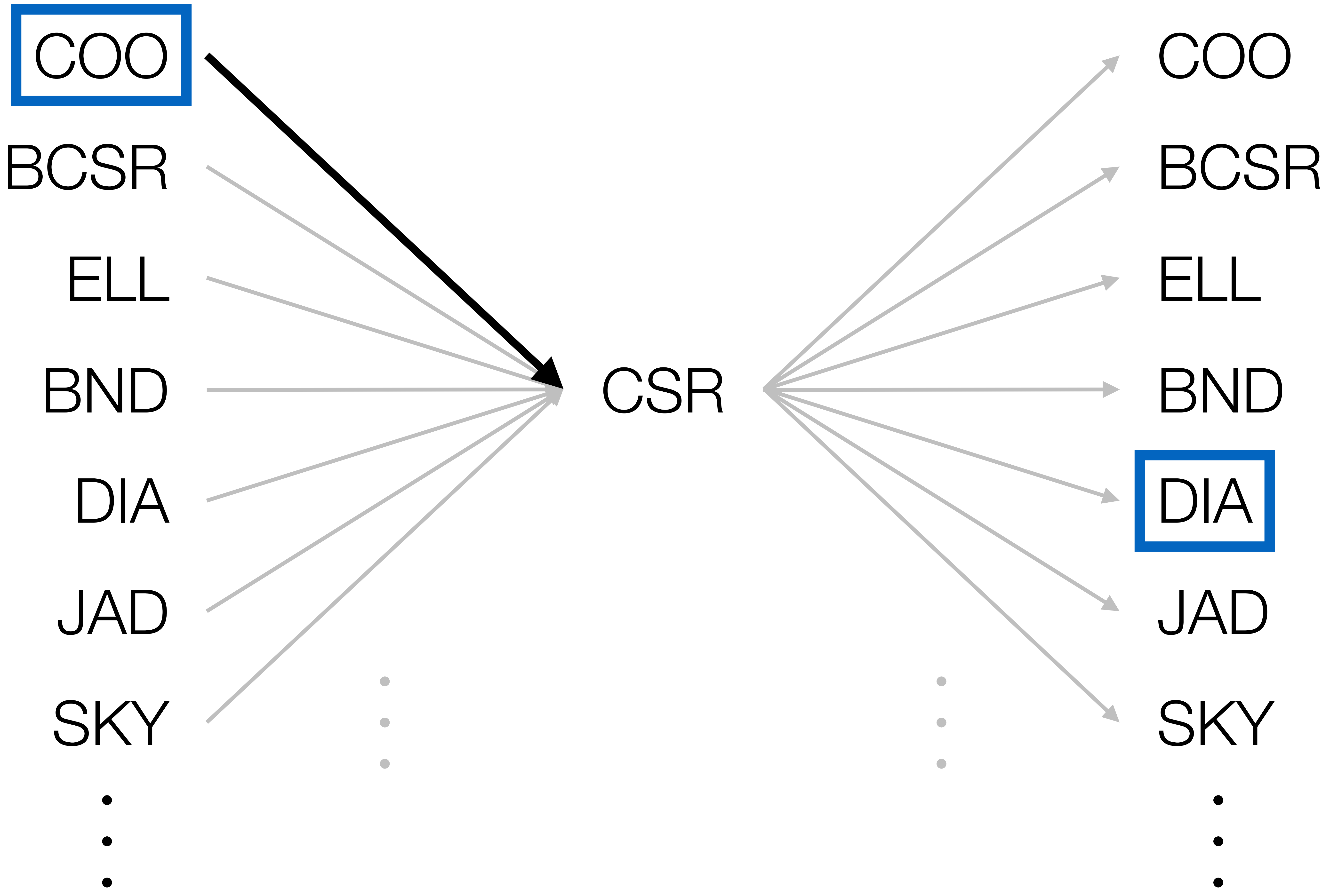
Hand-optimized libraries limit support for efficient conversion to few combinations of formats



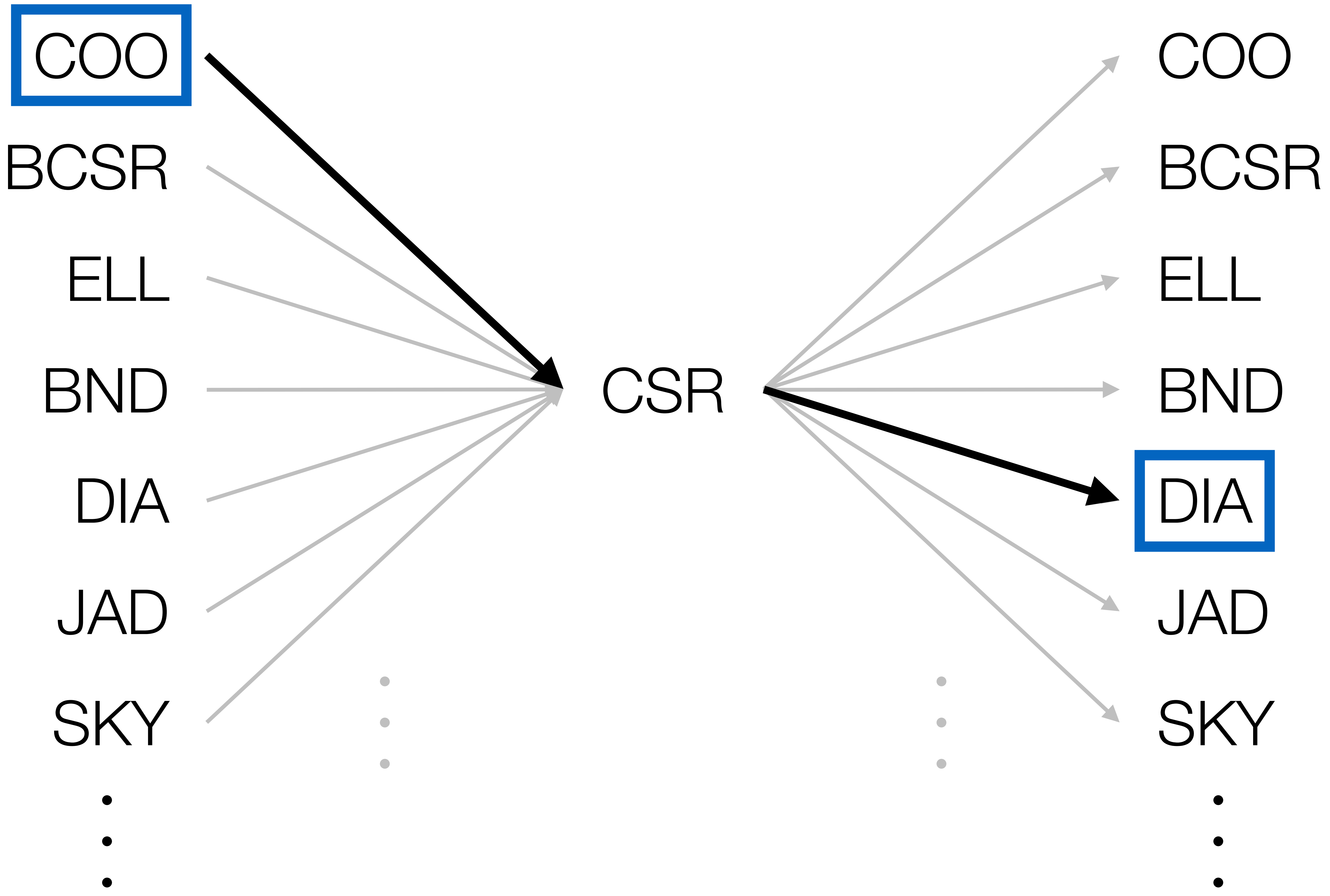
Hand-optimized libraries limit support for efficient conversion to few combinations of formats



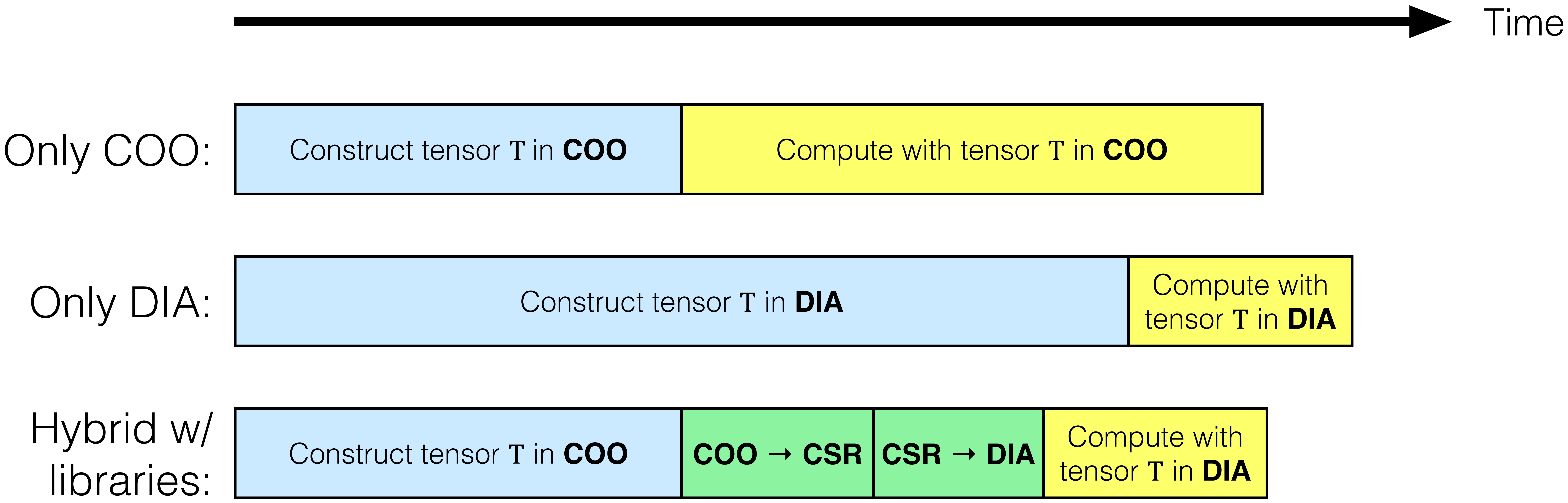
Hand-optimized libraries limit support for efficient conversion to few combinations of formats



Hand-optimized libraries limit support for efficient conversion to few combinations of formats



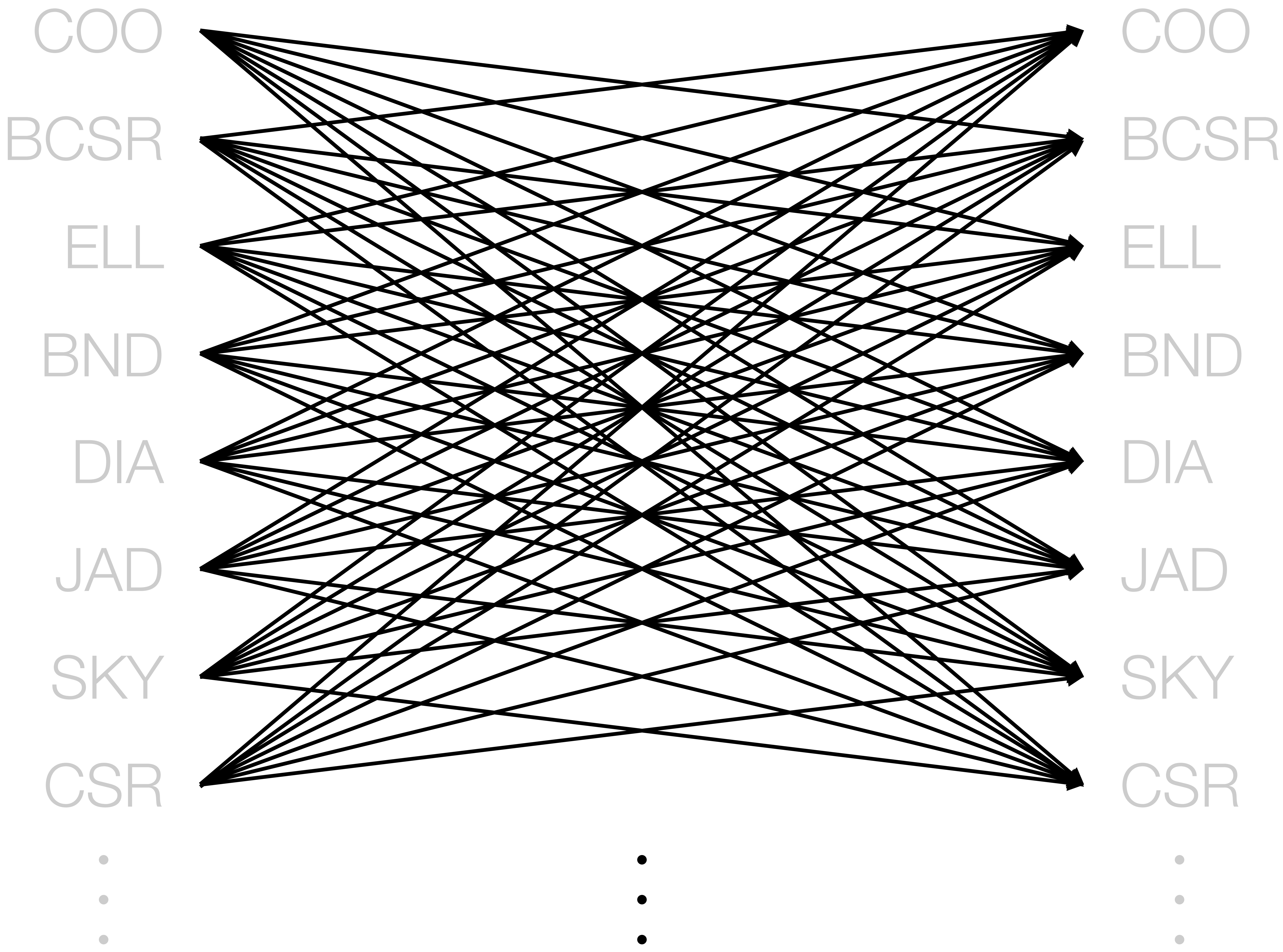
Inefficient conversion eliminates benefit of using different formats



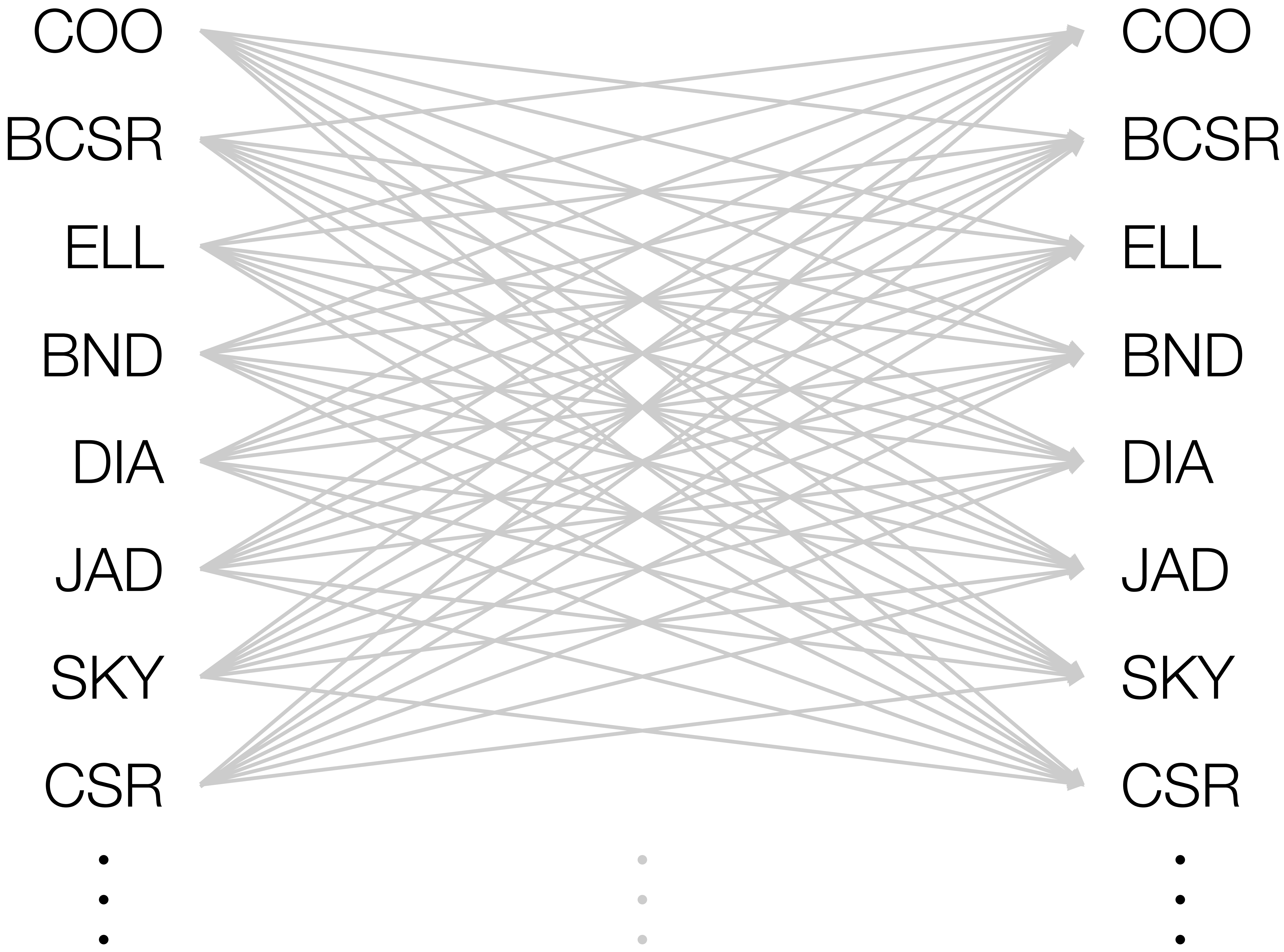
Automatic Generation of Efficient Sparse Tensor Format Conversion Routines

Stephen Chou, Fredrik Kjolstad, and Saman Amarasinghe

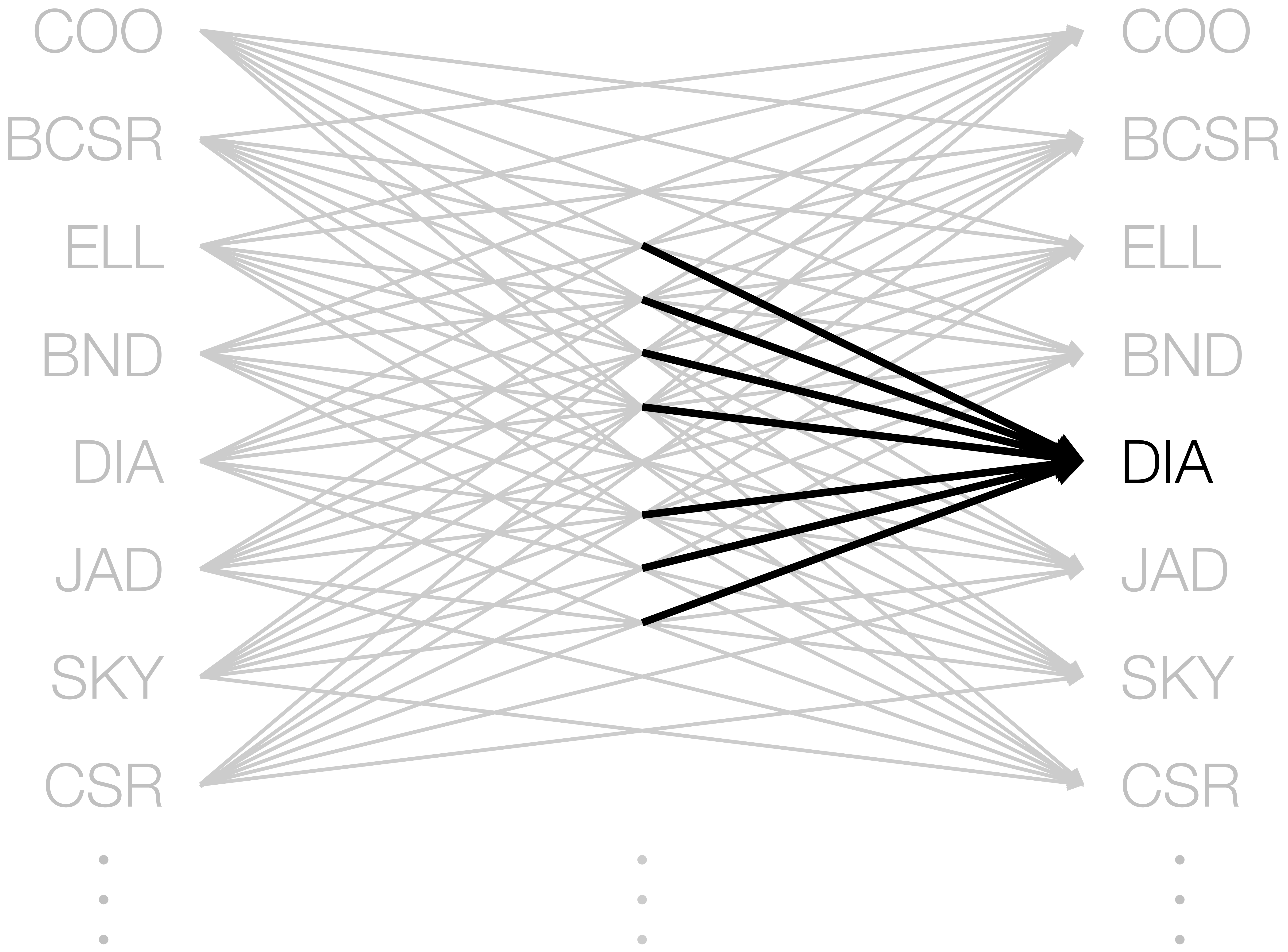
A compiler can generate efficient conversion routines from standalone specifications for each tensor format



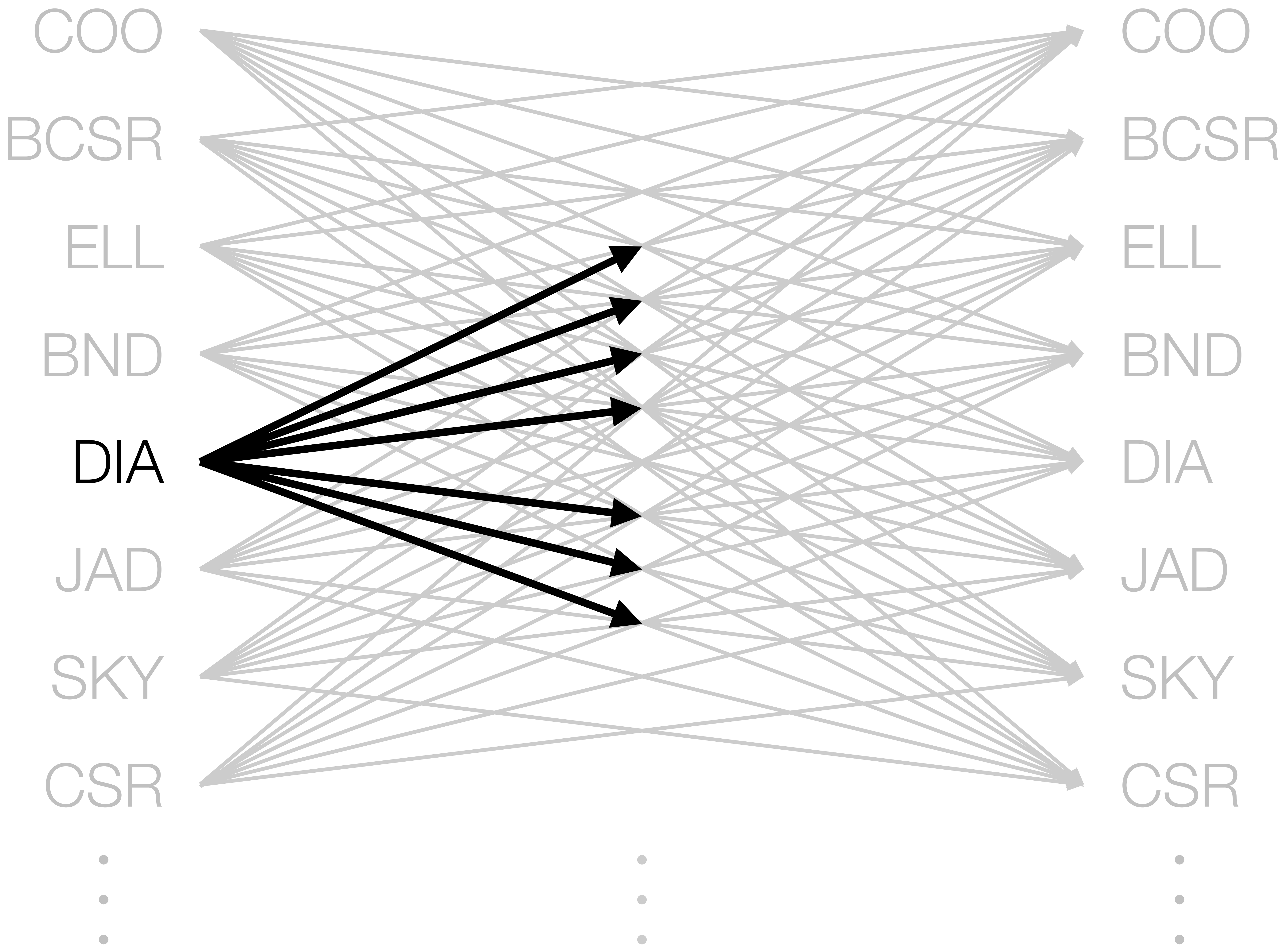
A compiler can generate efficient conversion routines from standalone specifications for each tensor format



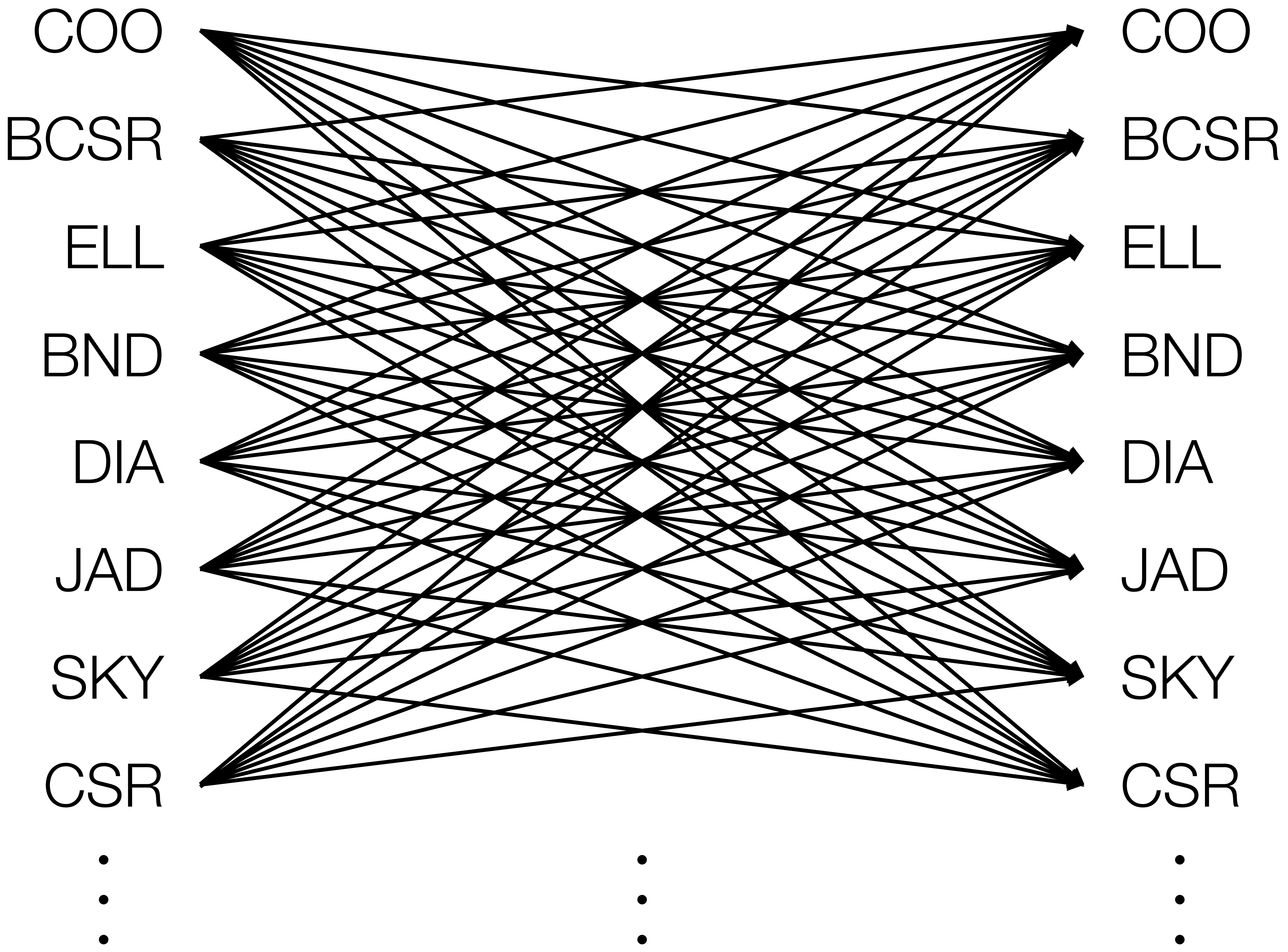
A compiler can generate efficient conversion routines from standalone specifications for each tensor format



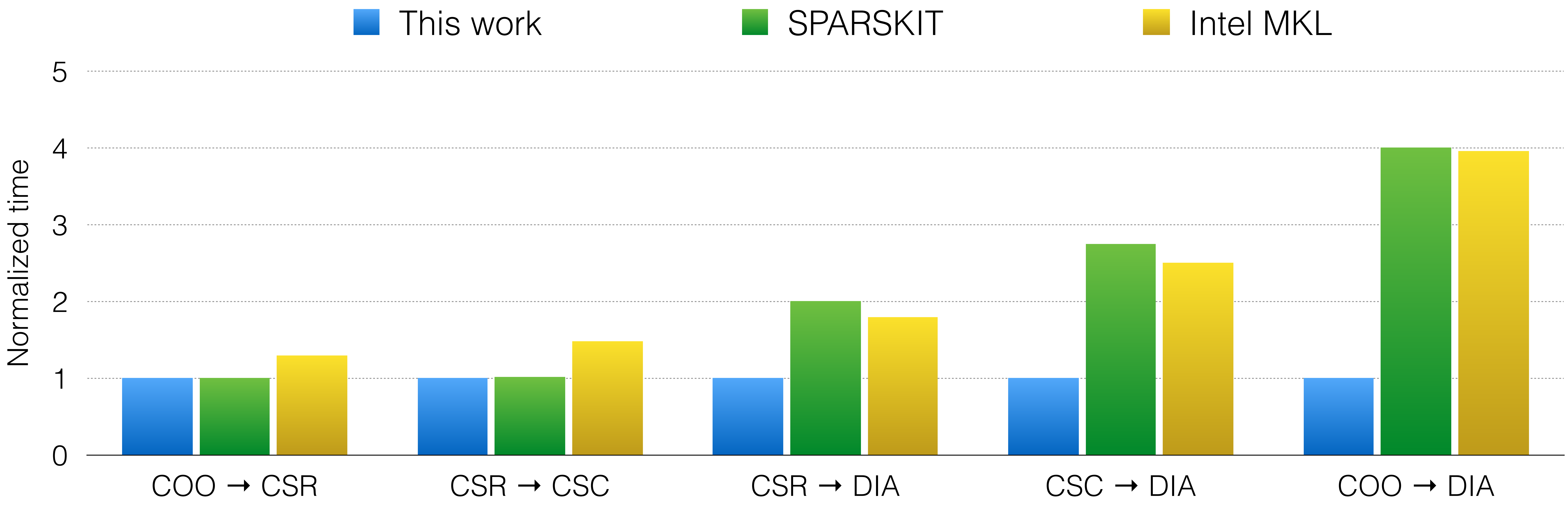
A compiler can generate efficient conversion routines from standalone specifications for each tensor format



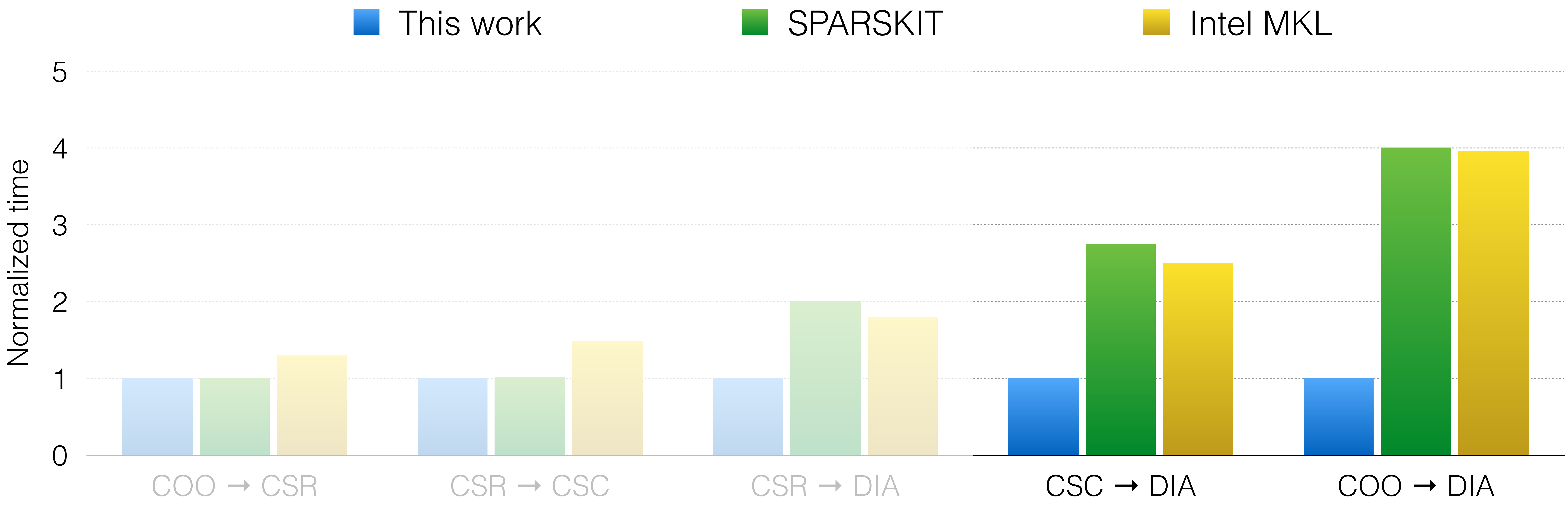
A compiler can generate efficient conversion routines from standalone specifications for each tensor format



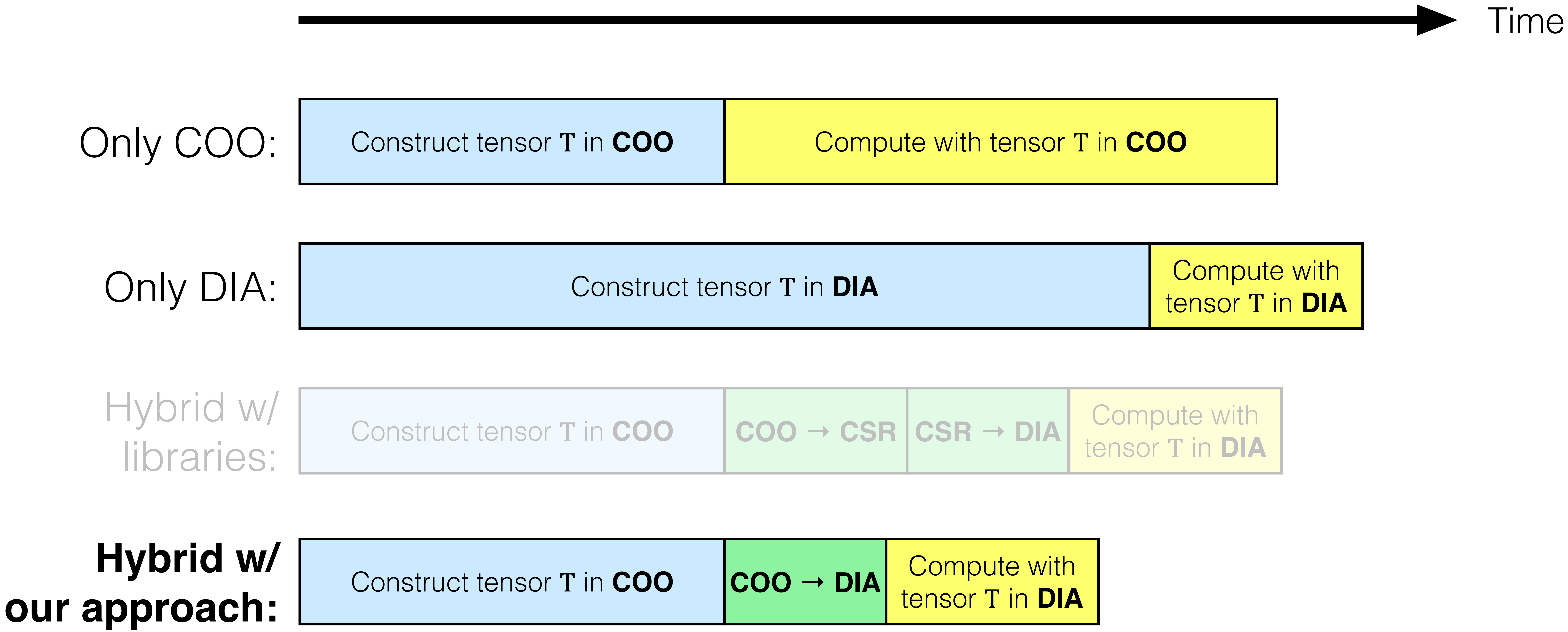
Our technique generates efficient code



Our technique generates efficient code



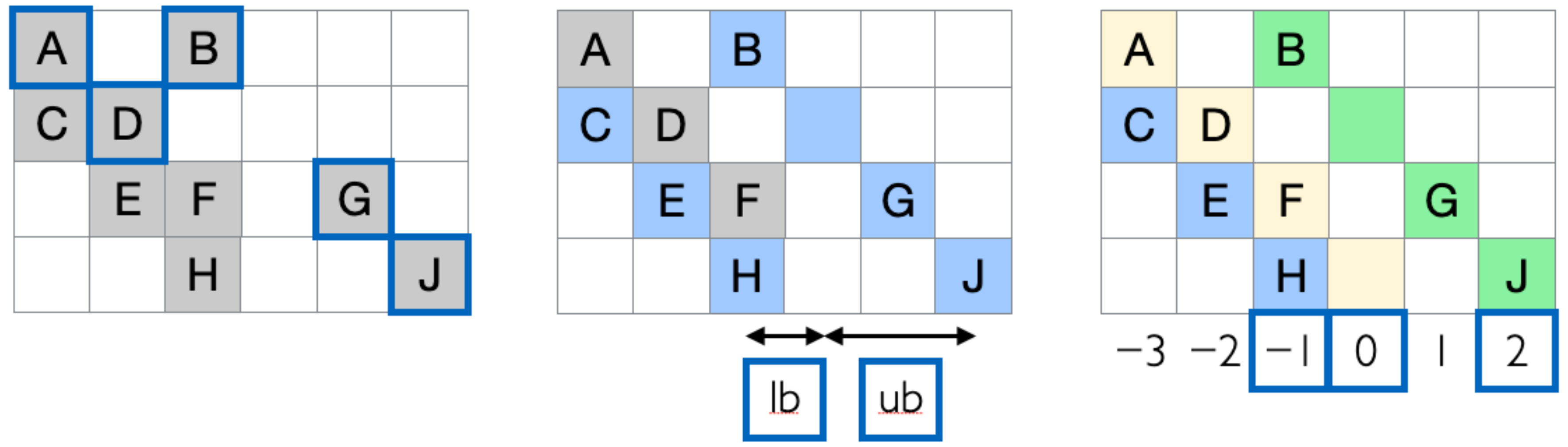
Being able to generate efficient conversion routines lets users exploit different formats for performance



Coordinate Remappings

j-i	-1	-1	-1	0	0	0	2	2	2
i	1	2	3	0	1	2	0	2	3
j	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

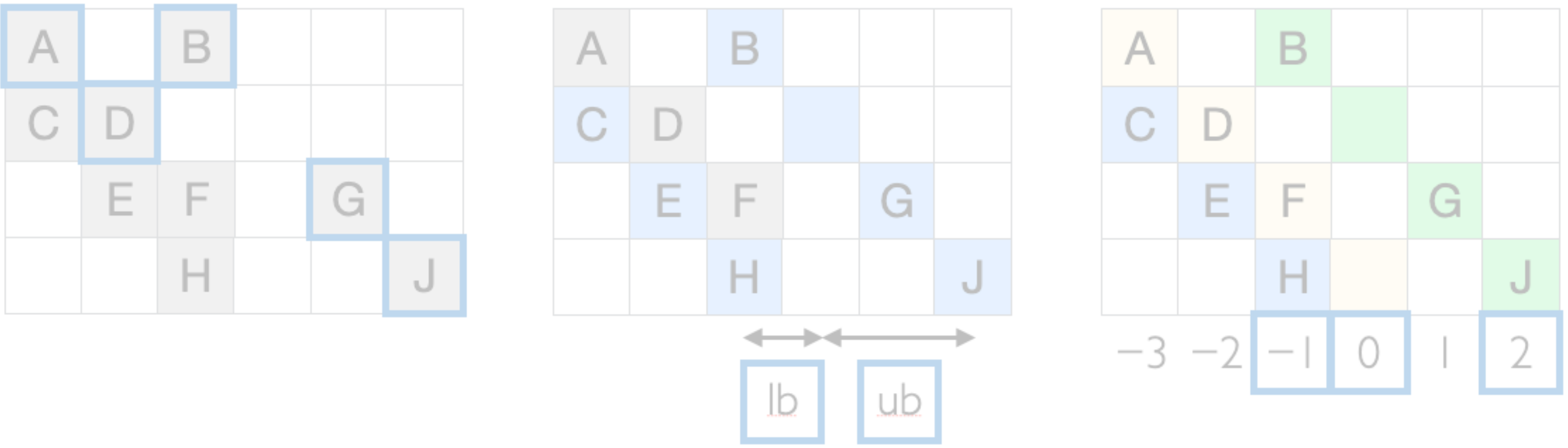
Attribute Queries



Coordinate Remappings

j-i	-1	-1	-1	0	0	0	2	2	2
i	1	2	3	0	1	2	0	2	3
j	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

Attribute Queries



Different tensor formats arrange nonzeros in memory in different ways

A		B			
C	D				
	E	F		G	
		H			J

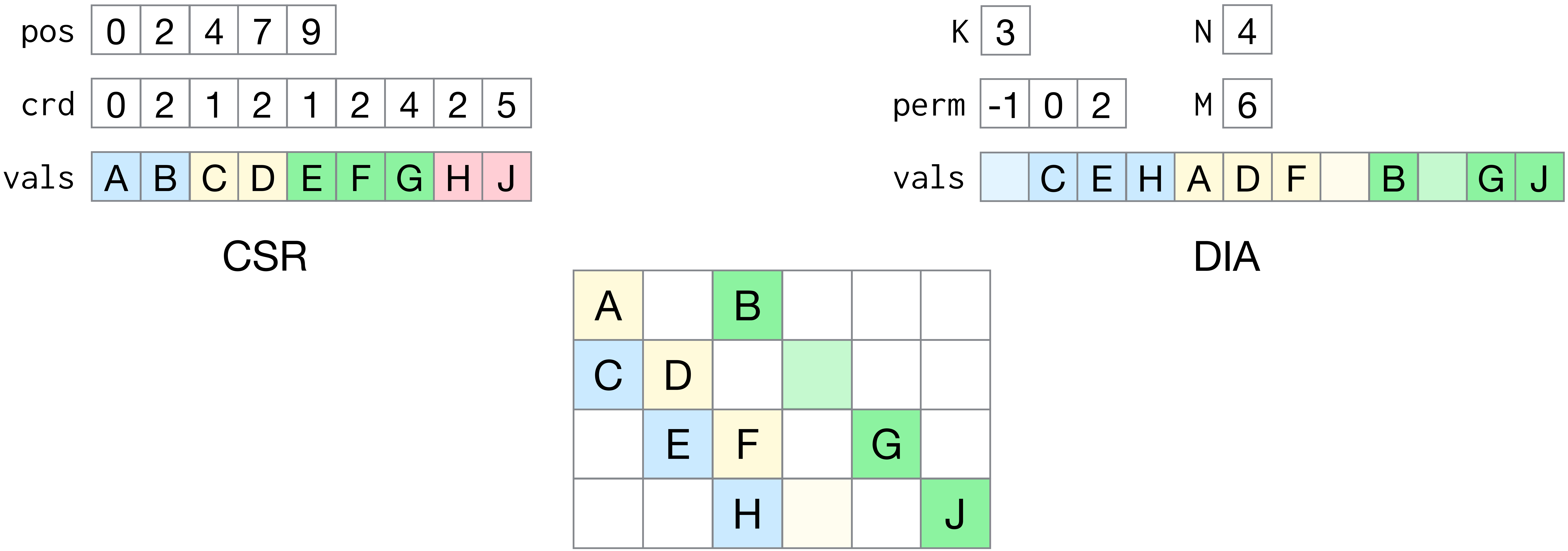
Different tensor formats arrange nonzeros in memory in different ways

`pos`	0	2	4	7	9				
`crd`	0	2	1	2	1	2	4	2	5
`vals`	A	B	C	D	E	F	G	H	J

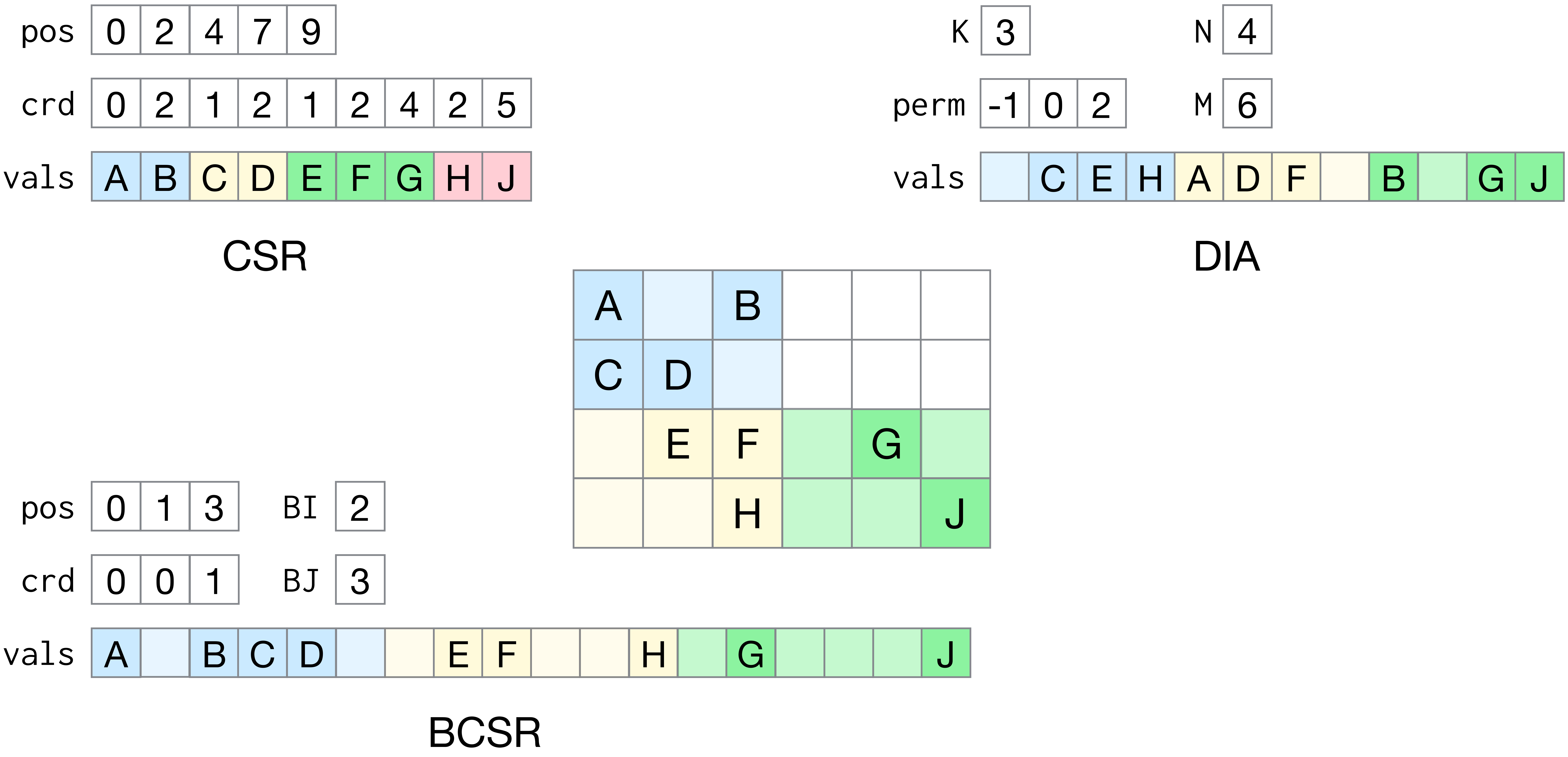
CSR

A		B			
C	D				
	E	F		G	
		H			J

Different tensor formats arrange nonzeros in memory in different ways



Different tensor formats arrange nonzeros in memory in different ways



Coordinate remapping captures how nonzeros are arranged in memory

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

Coordinate remapping captures how nonzeros are arranged in memory

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

`i`	0	0	1	1	2	2	2	3	3
`j`	0	2	0	1	1	2	4	2	5
	A	B	C	D	E	F	G	H	J

Coordinate remapping captures how nonzeros are arranged in memory

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

`j-i`	0	2	-1	0	-1	0	2	-1	2
`i`	0	0	1	1	2	2	2	3	3
`j`	0	2	0	1	1	2	4	2	5
	A	B	C	D	E	F	G	H	J

Coordinate remapping captures how nonzeros are arranged in memory

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

`j-i`	-1	-1	-1	0	0	0	2	2	2
`i`	1	2	3	0	1	2	0	2	3
`j`	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

Coordinate remapping captures how nonzeros are arranged in memory

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

j-i	-1	-1	-1	0	0	0	2	2	2
i	1	2	3	0	1	2	0	2	3
j	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

Coordinate remapping captures how nonzeros are arranged in memory

`j-i`	-1	-1	-1	0	0	0	2	2	2
`i`	1	2	3	0	1	2	0	2	3
`j`	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

Coordinate remapping captures how nonzeros are arranged in memory

K	3
N	4
M	6

perm	-1	0	2

vals		C	E	H	A	D	F		B		G	$\boxed{J}$

j-i	-1	-1	-1	0	0	0	2	2	2
i	1	2	3	0	1	2	0	2	3
j	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	$\boxed{J}$

Coordinate remapping captures how nonzeros are arranged in memory

K	3											
N	4											
perm	-1	0	2									
M	6											
vals		C	E	H	A	D	F		B		G	J

```
(i,j) -> (j-i,i,j)
```

Coordinate remapping captures how nonzeros are arranged in memory

K	3

N	4

perm			
	-1	0	2

M	6

vals												
		C	E	H	A	D	F		B		G	J

$$(i,j) \rightarrow (\boxed{j-i},i,j)$$

Coordinate remapping captures how nonzeros are arranged in memory

K	3	N	4

perm	-1	0	2		M	6

vals		C	E	H	A	D	F		B		G	J

$$\texttt{(i,j) -> (j-i,}\boxed{\texttt{i,j}}\texttt{)}$$

Compiler uses coordinate remapping to generate code to reorder nonzeros

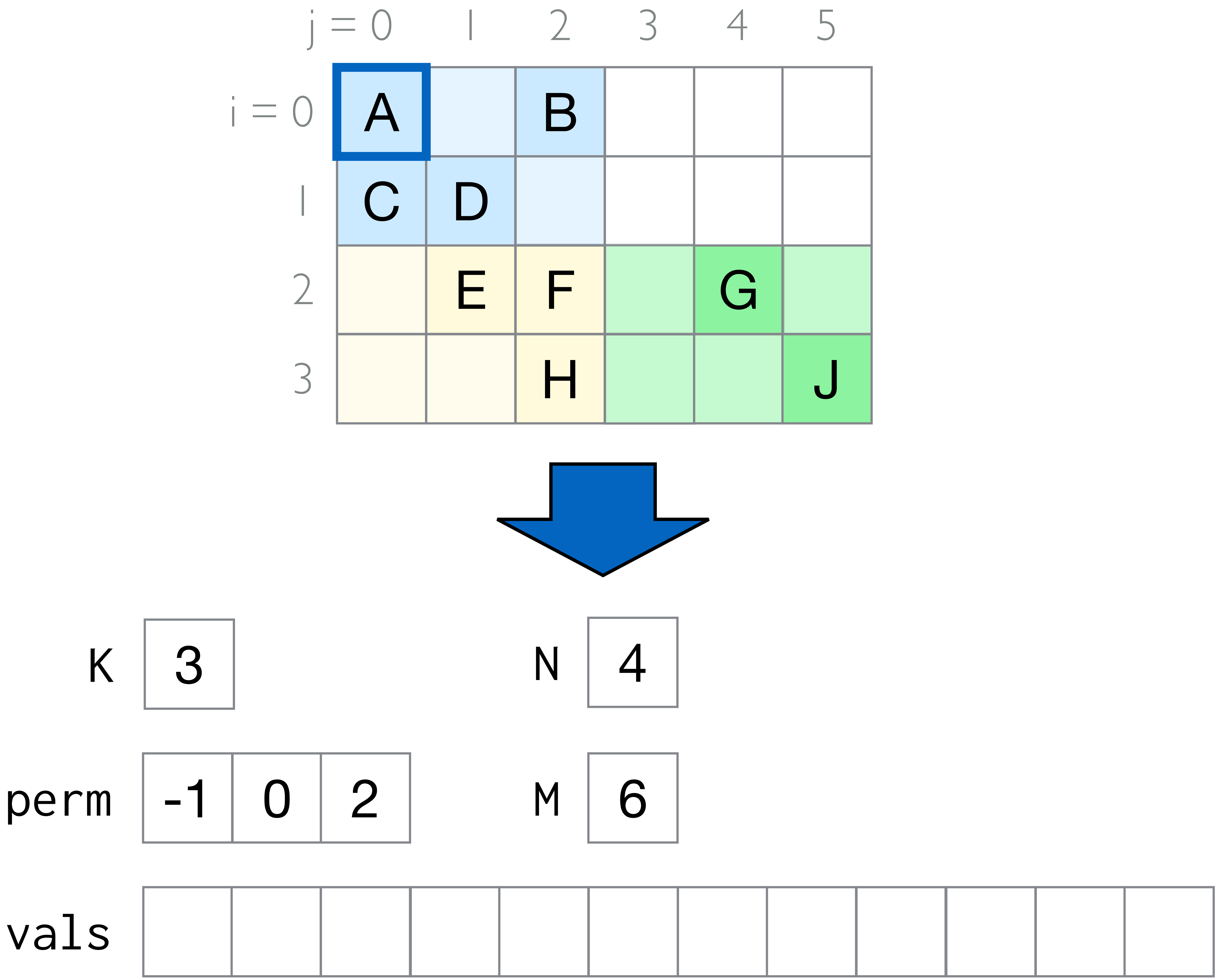
```
(i,j) -> (j-i,i,j)
```

Compiler uses coordinate remapping to generate code to reorder nonzeros

```
(i,j) -> (j-i,i,j)
```

```
Identify segment d in vals
  that corresponds to j - i
Identify position p in d
  that corresponds to i and j
vals[p] = B[i,j]
```

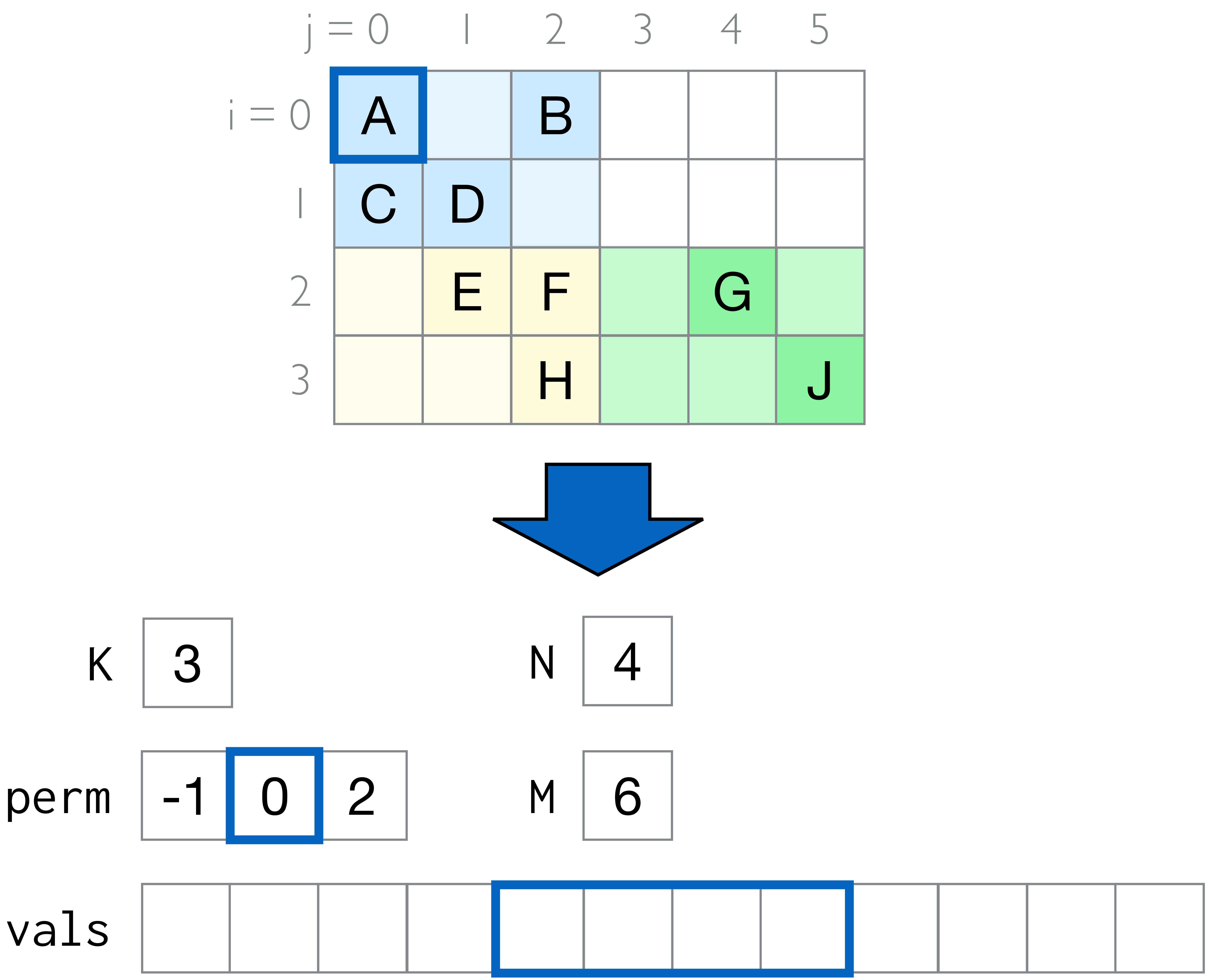
Compiler uses coordinate remapping to generate code to reorder nonzeros




```
(i,j) -> (j-i,i,j)
```

Identify segment d in vals
  that corresponds to **j - i**
Identify position p in d
  that corresponds to **i** and **j**
vals[p] = B[i,j]

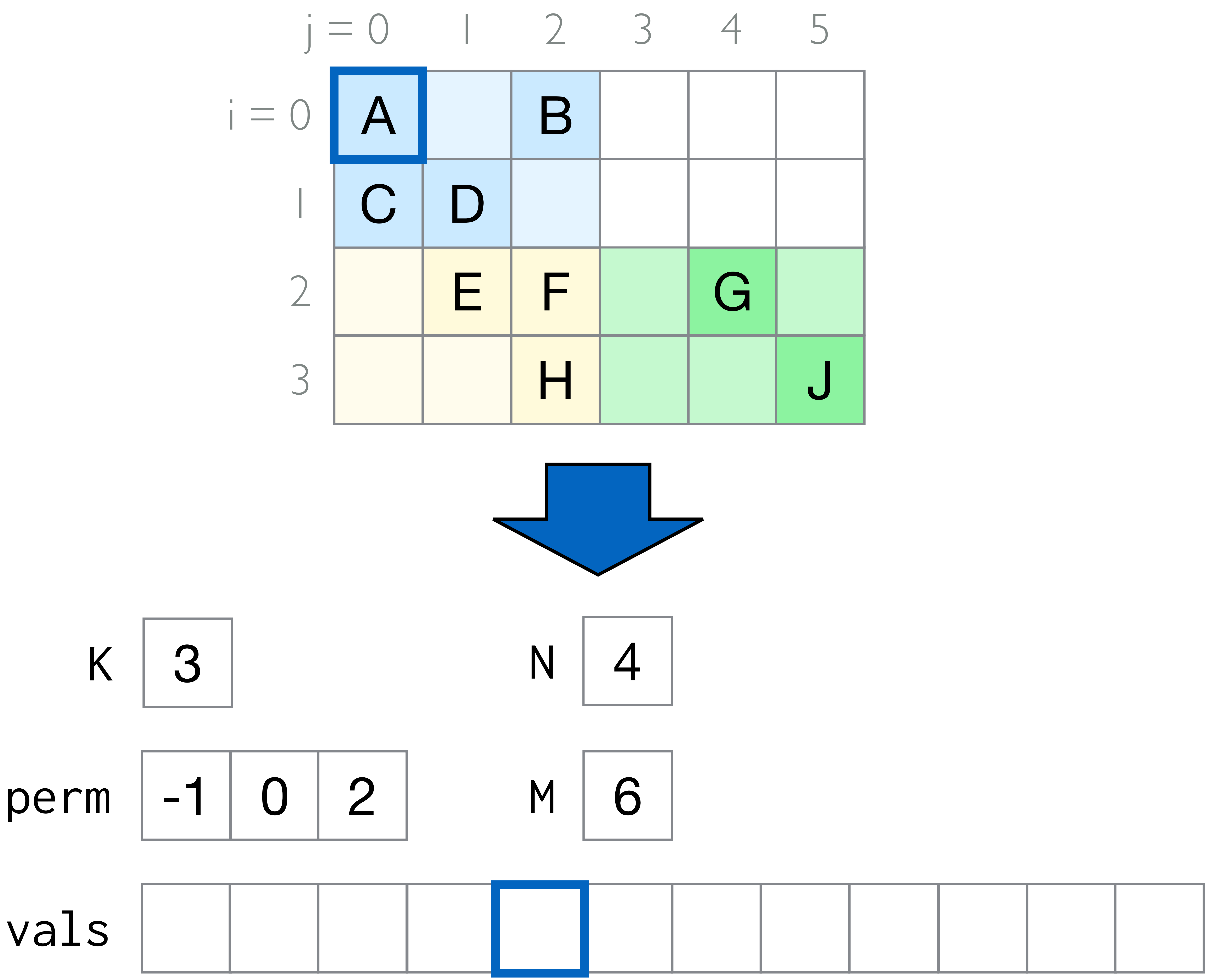
Compiler uses coordinate remapping to generate code to reorder nonzeros




```
(i,j) -> (j-i,i,j)
```

```
Identify segment d in vals
  that corresponds to j - i
Identify position p in d
  that corresponds to i and j
vals[p] = B[i,j]
```

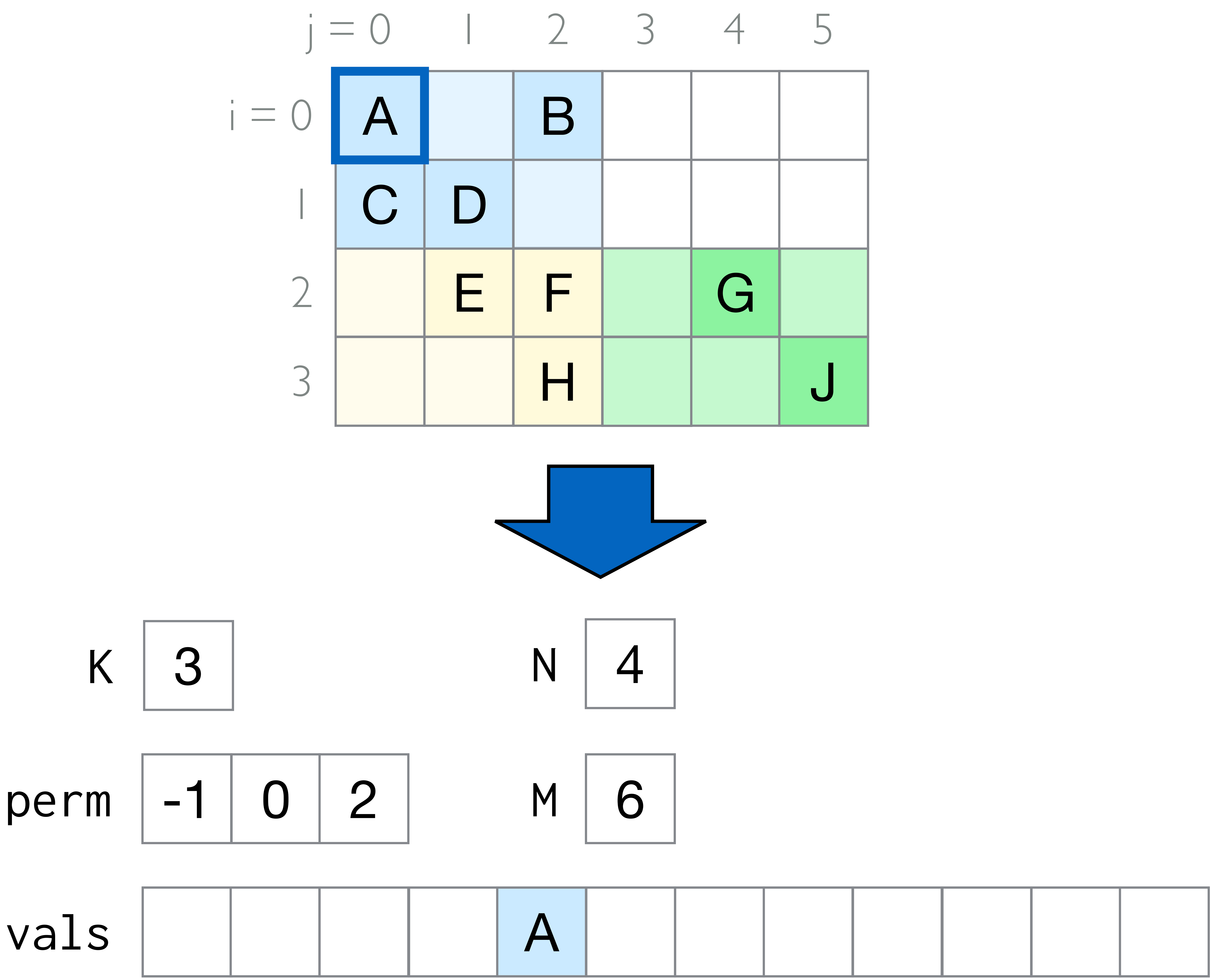
Compiler uses coordinate remapping to generate code to reorder nonzeros




```
(i,j) -> (j-i,i,j)
```

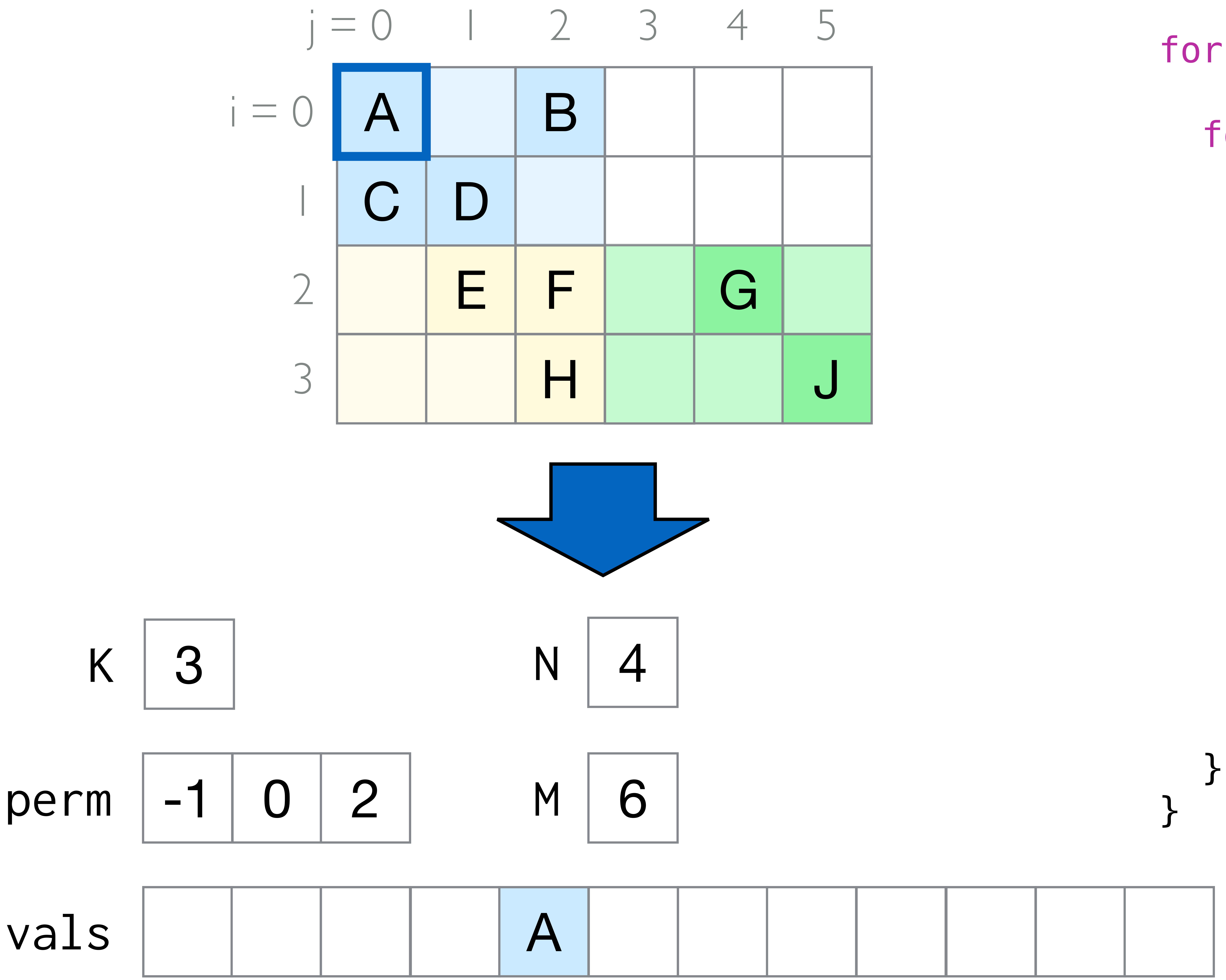
```
Identify segment d in vals
  that corresponds to j - i
Identify position p in d
  that corresponds to i and j
vals[p] = B[i,j]
```

Compiler uses coordinate remapping to generate code to reorder nonzeros



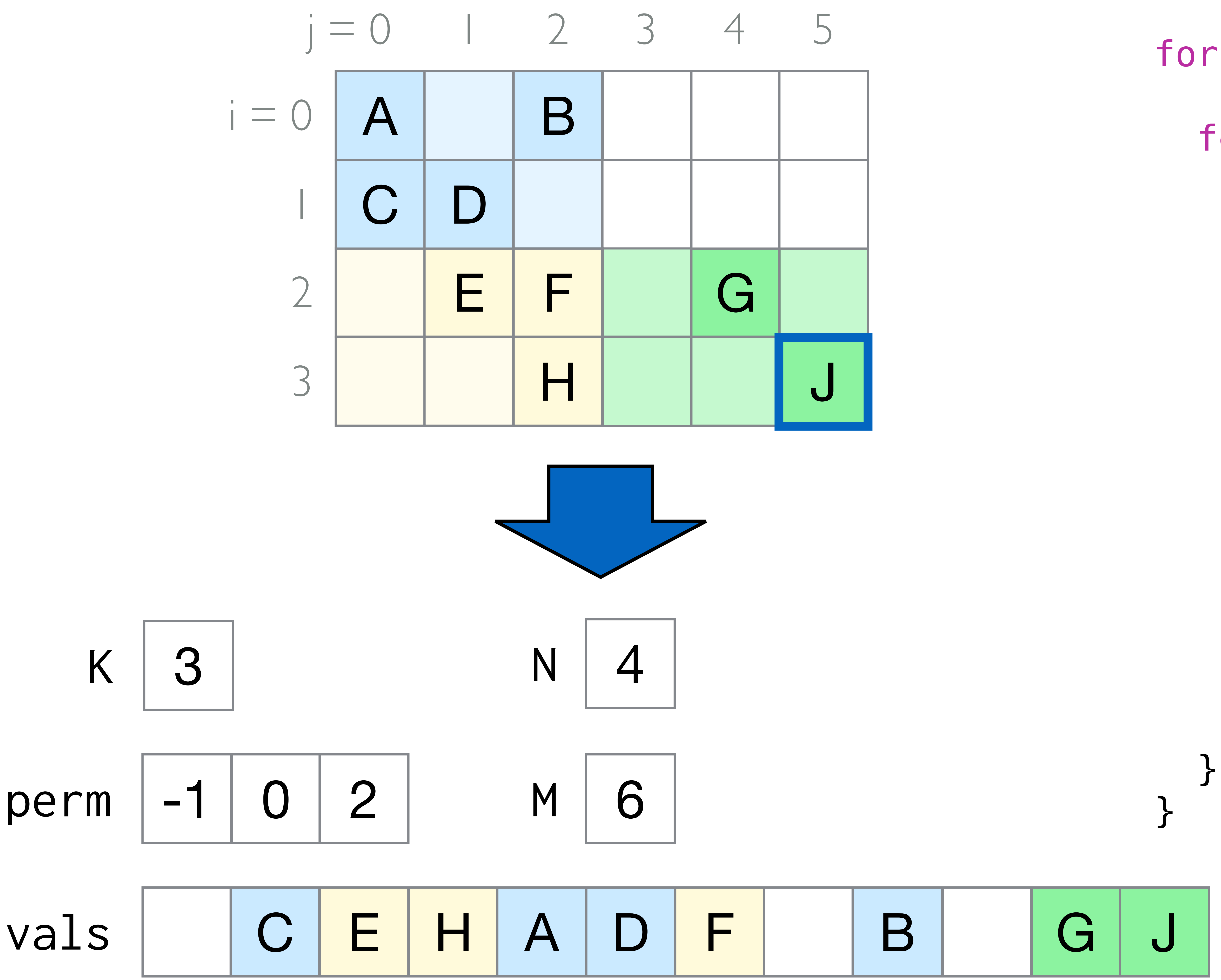

```
(i,j) -> (j-i,i,j)

Identify segment d in vals
  that corresponds to j - i
Identify position p in d
  that corresponds to i and j
vals[p] = B[i,j]
```

Compiler uses coordinate remapping to generate code to reorder nonzeros




```
for (int bi = 0;
         bi < M / BI; bi++) {
  for (int bj = 0;
           bj < N / BJ; bj++) {
    for (int i = bi * BI;
             i < (bi + 1) * BI; i++) {
      for (int j = bj * BJ;
               j < (bj + 1) * BJ; j++) {
        if (B[i,j] != 0.0) {
          Identify segment d in vals
            that corresponds to j - i
          Identify position p in d
            that corresponds to i and j
          vals[p] = B[i,j]
        }
      }
    }
  }
}
```

Compiler uses coordinate remapping to generate code to reorder nonzeros

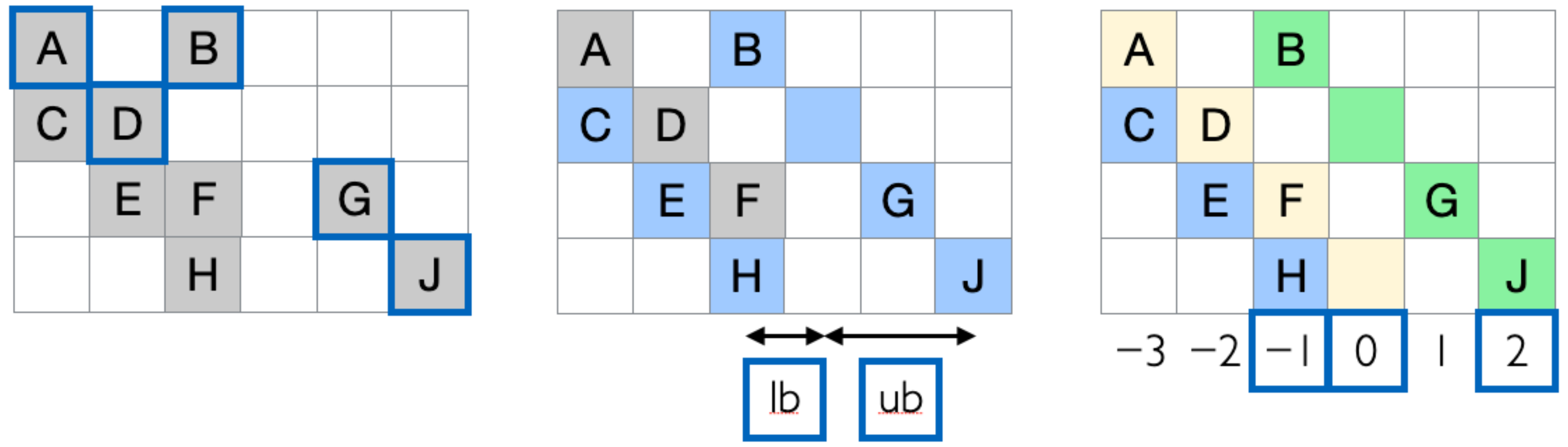



```
for (int bi = 0;
         bi < M / BI; bi++) {
  for (int bj = 0;
           bj < N / BJ; bj++) {
    for (int i = bi * BI;
             i < (bi + 1) * BI; i++) {
      for (int j = bj * BJ;
               j < (bj + 1) * BJ; j++) {
        if (B[i,j] != 0.0) {
          Identify segment d in vals
            that corresponds to j - i
          Identify position p in d
            that corresponds to i and j
          vals[p] = B[i,j]
        }
      }
    }
  }
}
```

Coordinate Remappings

j-i	-1	-1	-1	0	0	0	2	2	2
i	1	2	3	0	1	2	0	2	3
j	0	1	2	0	1	2	2	4	5
	C	E	H	A	D	F	B	G	J

Attribute Queries



Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

COO

`rows`	0	1	1	2	0	3	3	2	2
`cols`	0	0	1	1	2	2	5	2	4
`vals`	A	C	D	E	B	H	J	F	G

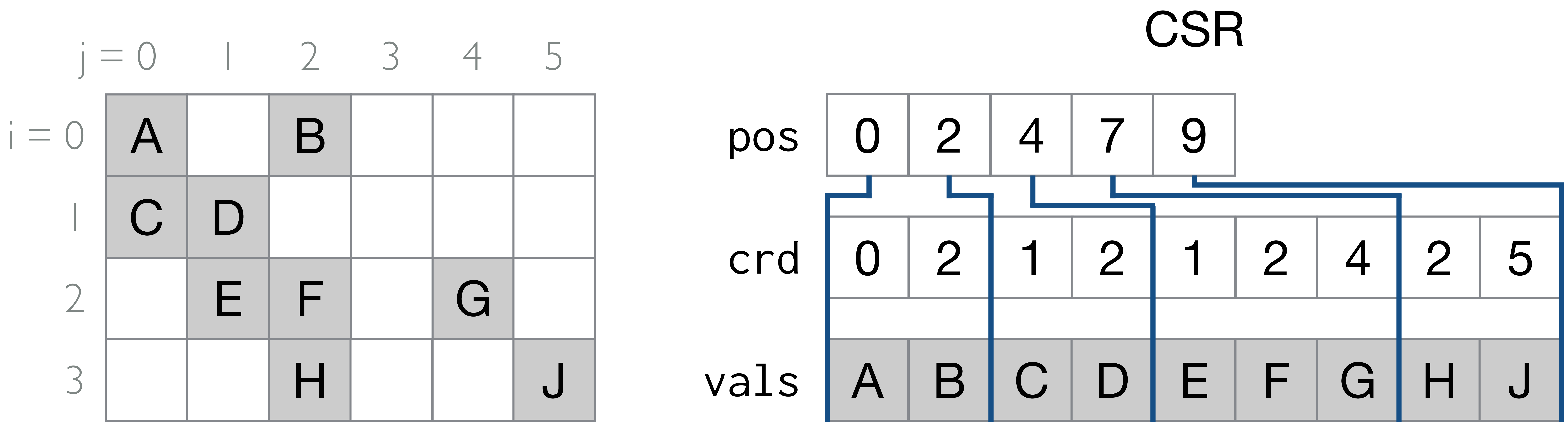
Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

CSR

`pos`	0	2	4	7	9				
`crd`	0	2	1	2	1	2	4	2	5
`vals`	A	B	C	D	E	F	G	H	J

Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor



Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

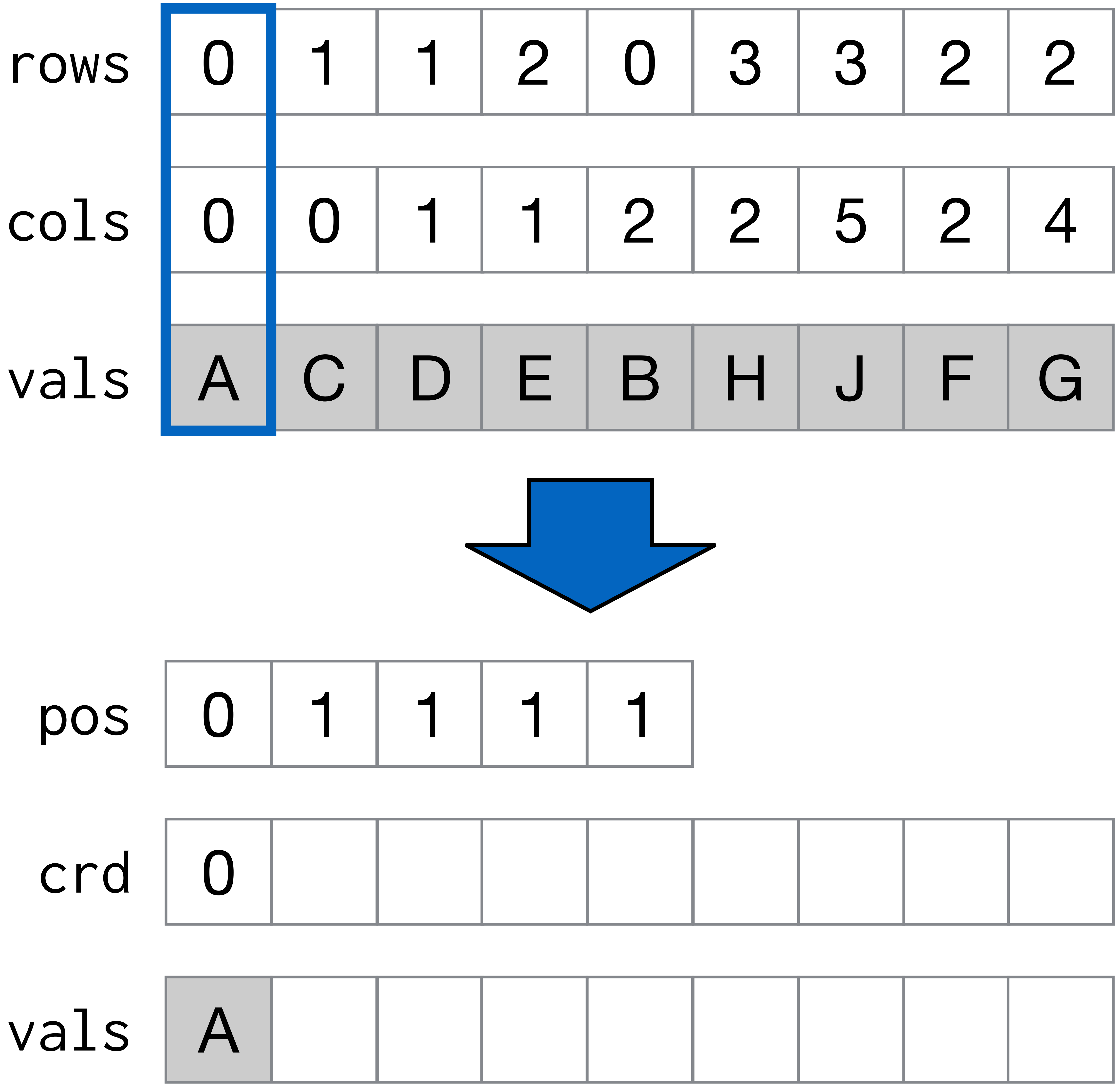
pos	0	0	0	0	0				
crd									
vals									

Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

pos	0	0	0	0	0				
crd	0								
vals	A								

Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor



Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

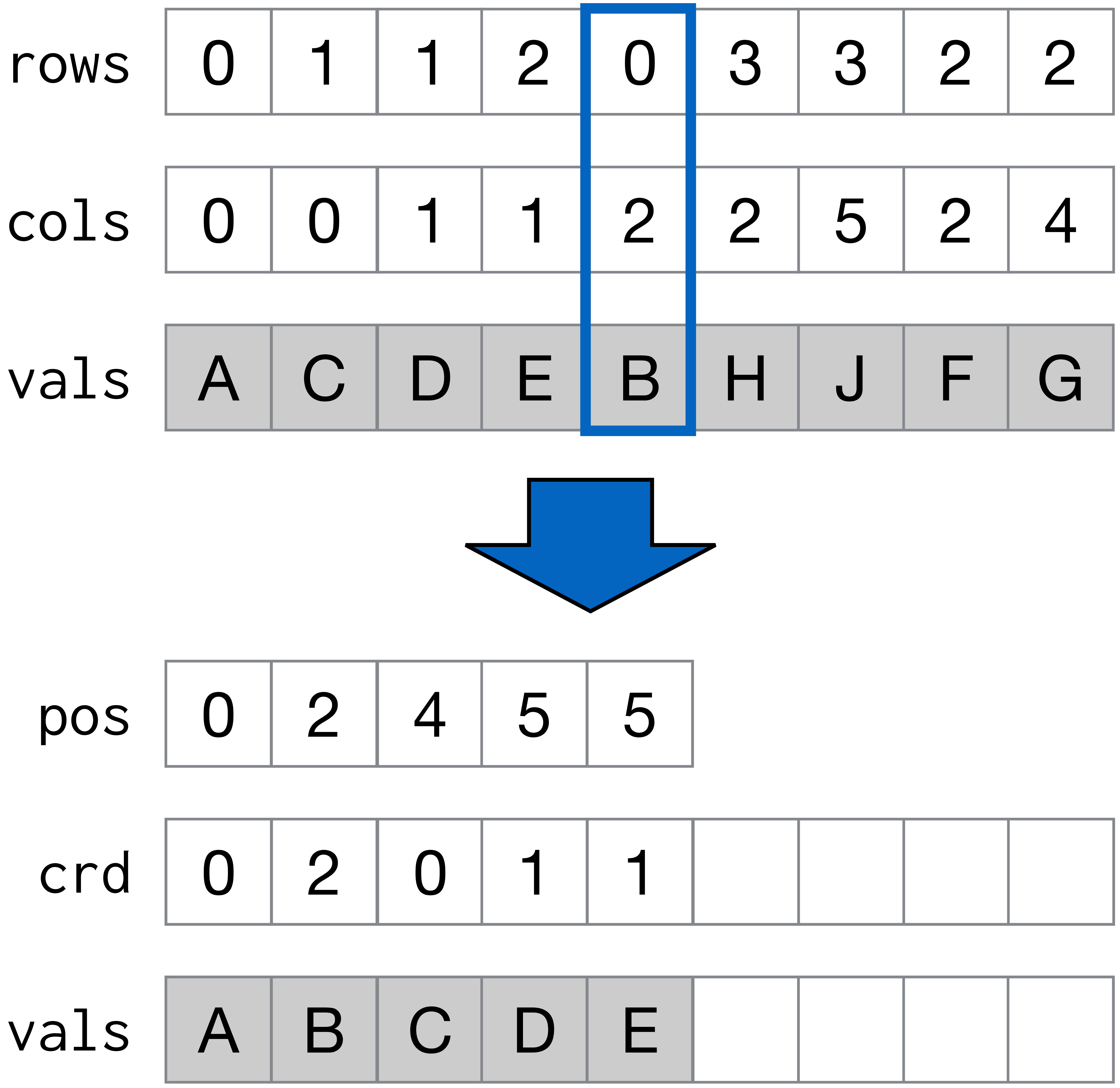
pos	0	1	3	4	4				
crd	0	0	1	1					
vals	A	C	D	E					

Reordering a tensor’s nonzeros without explicitly sorting them requires knowing statistics about the tensor

`rows`	0	1	1	2	0	3	3	2	2
`cols`	0	0	1	1	2	2	5	2	4
`vals`	A	C	D	E	B	H	J	F	G

`pos`	0	1	3	4	4				
`crd`	0	0	1	1					
`vals`	A	C	D	E					

Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor



Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

pos	0	2	4	7	9

crd	0	2	0	1	1	2	4	2	5
vals	A	B	C	D	E	F	G	H	J

Reordering a tensor’s nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

pos	0				

crd									
vals									

i	nnz
0	2
1	2
2	3
3	2

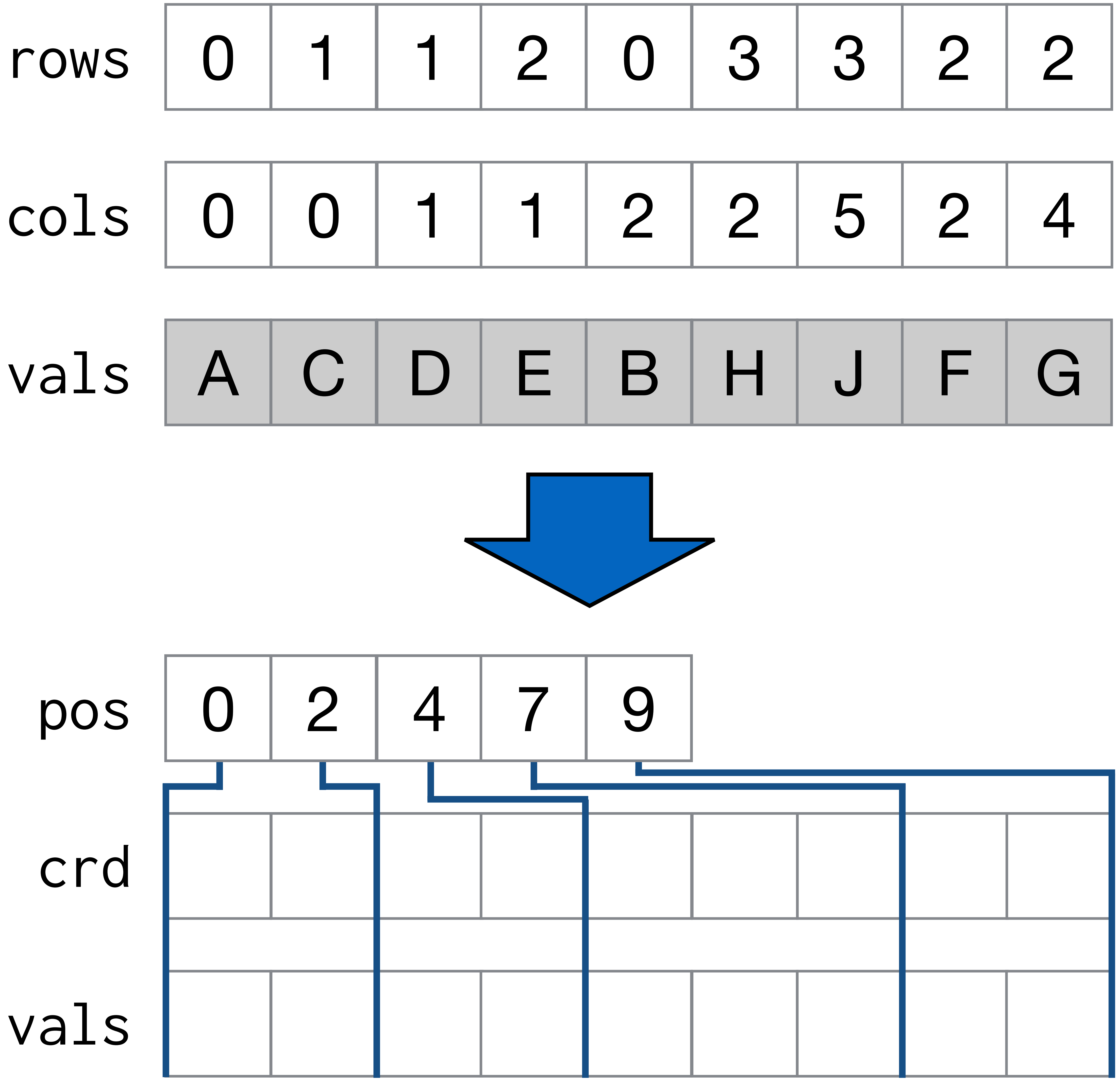
Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor

rows	0	1	1	2	0	3	3	2	2
cols	0	0	1	1	2	2	5	2	4
vals	A	C	D	E	B	H	J	F	G

i	nnz
0	2
1	2
2	3
3	2

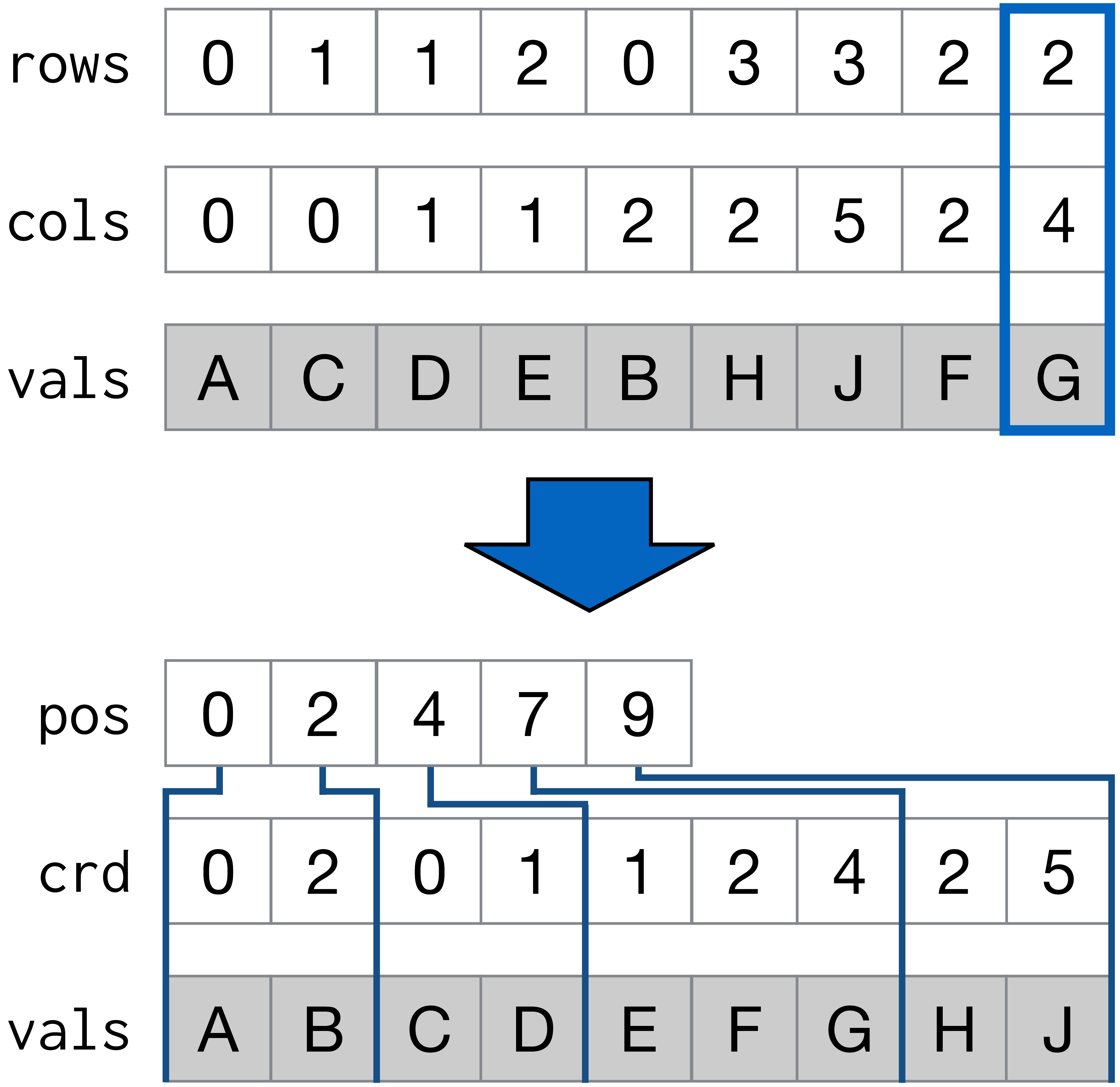
pos	0	2	4	7	9				
crd									
vals									

Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor



i	nnz
0	2
1	2
2	3
3	2

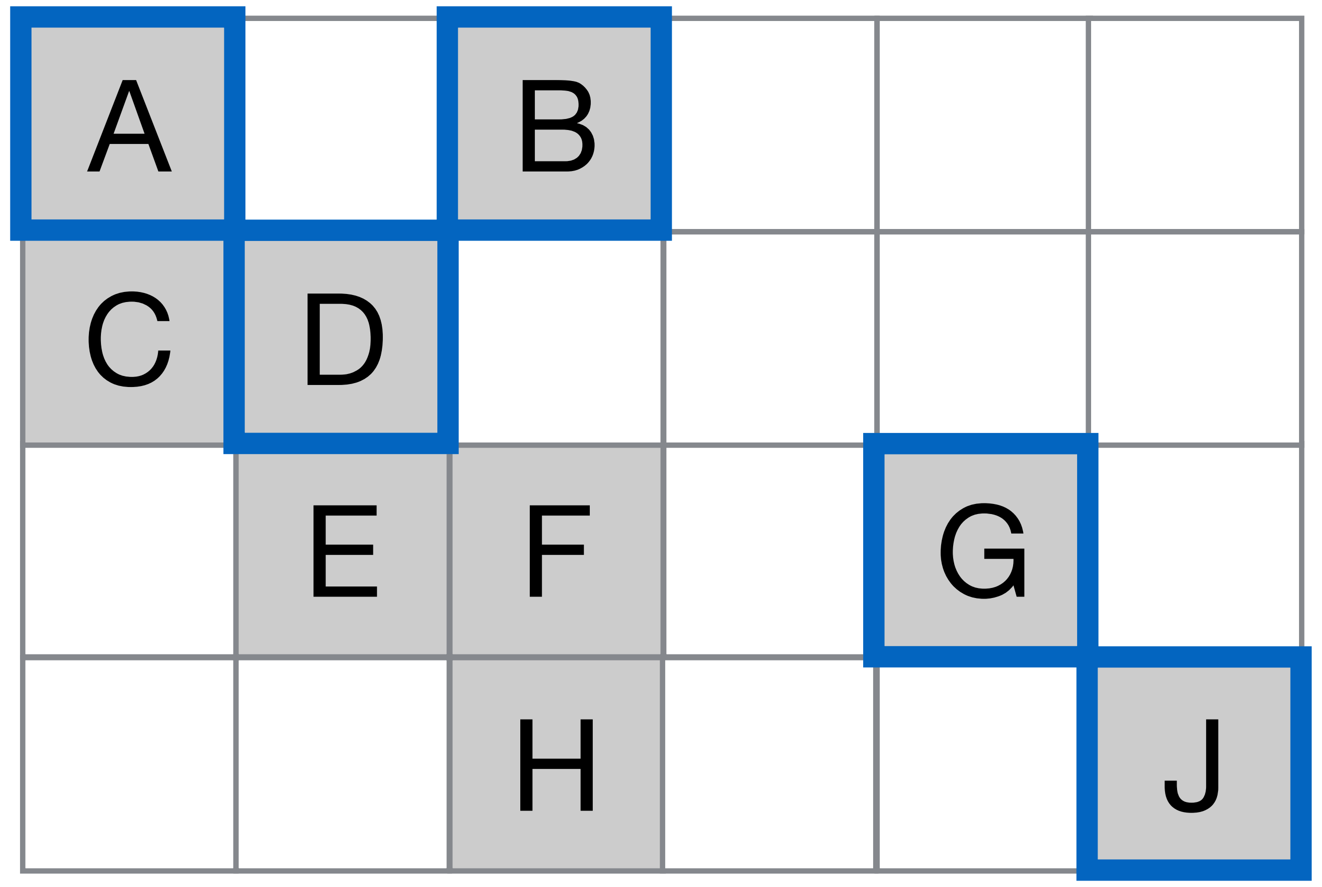
Reordering a tensor's nonzeros without explicitly sorting them requires knowing statistics about the tensor



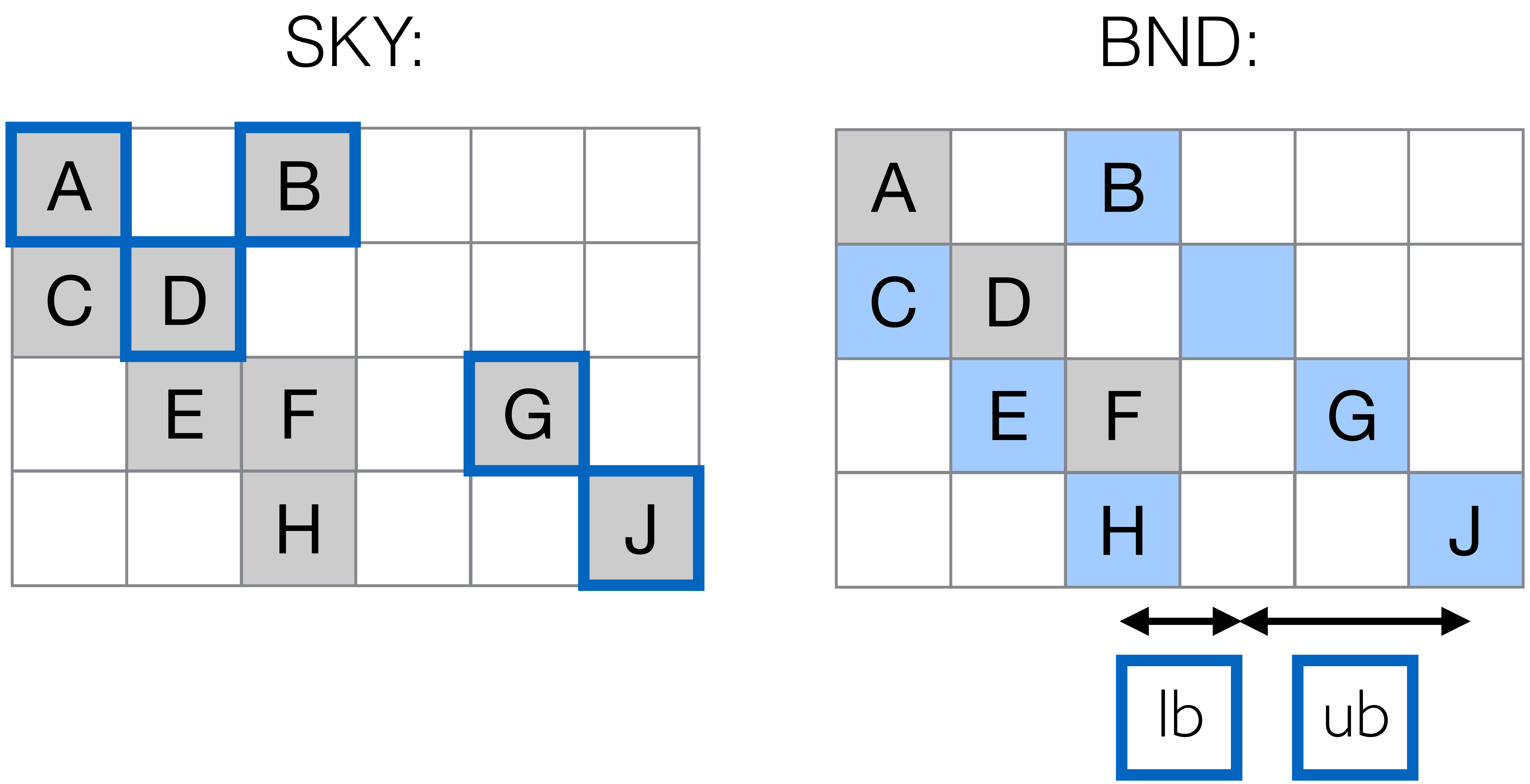
i	nnz
0	2
1	2
2	3
3	2

Converting tensors to different formats requires knowing different statistics about the tensors

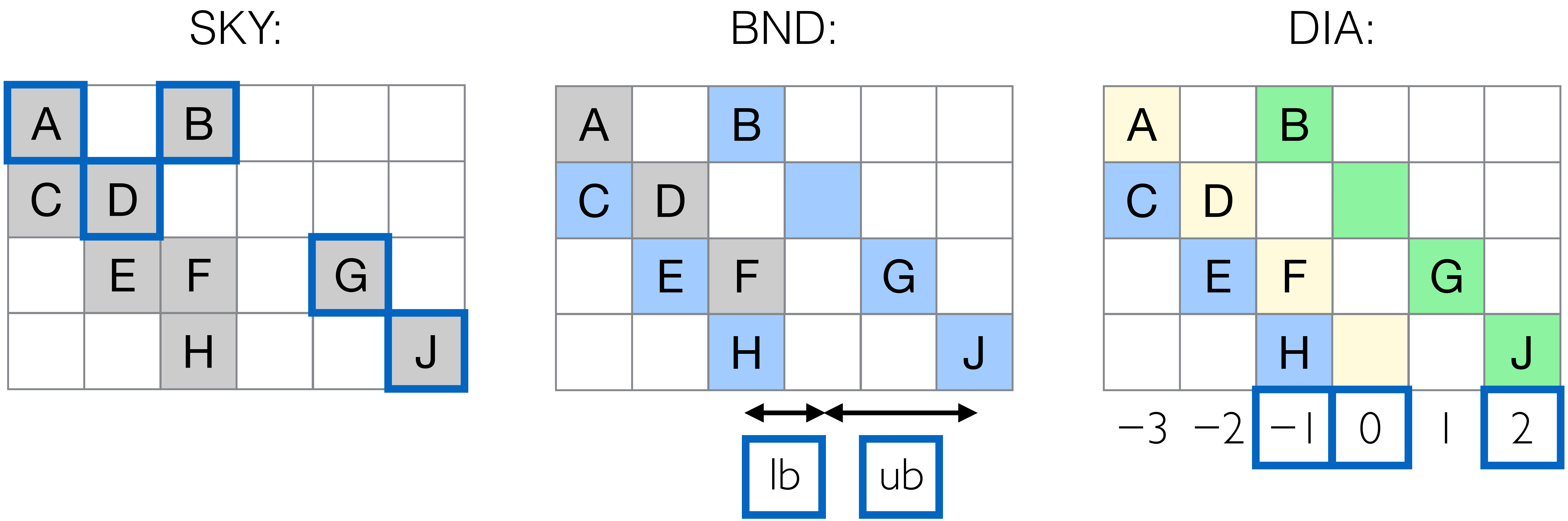
SKY:



Converting tensors to different formats requires knowing different statistics about the tensors



Converting tensors to different formats requires knowing different statistics about the tensors

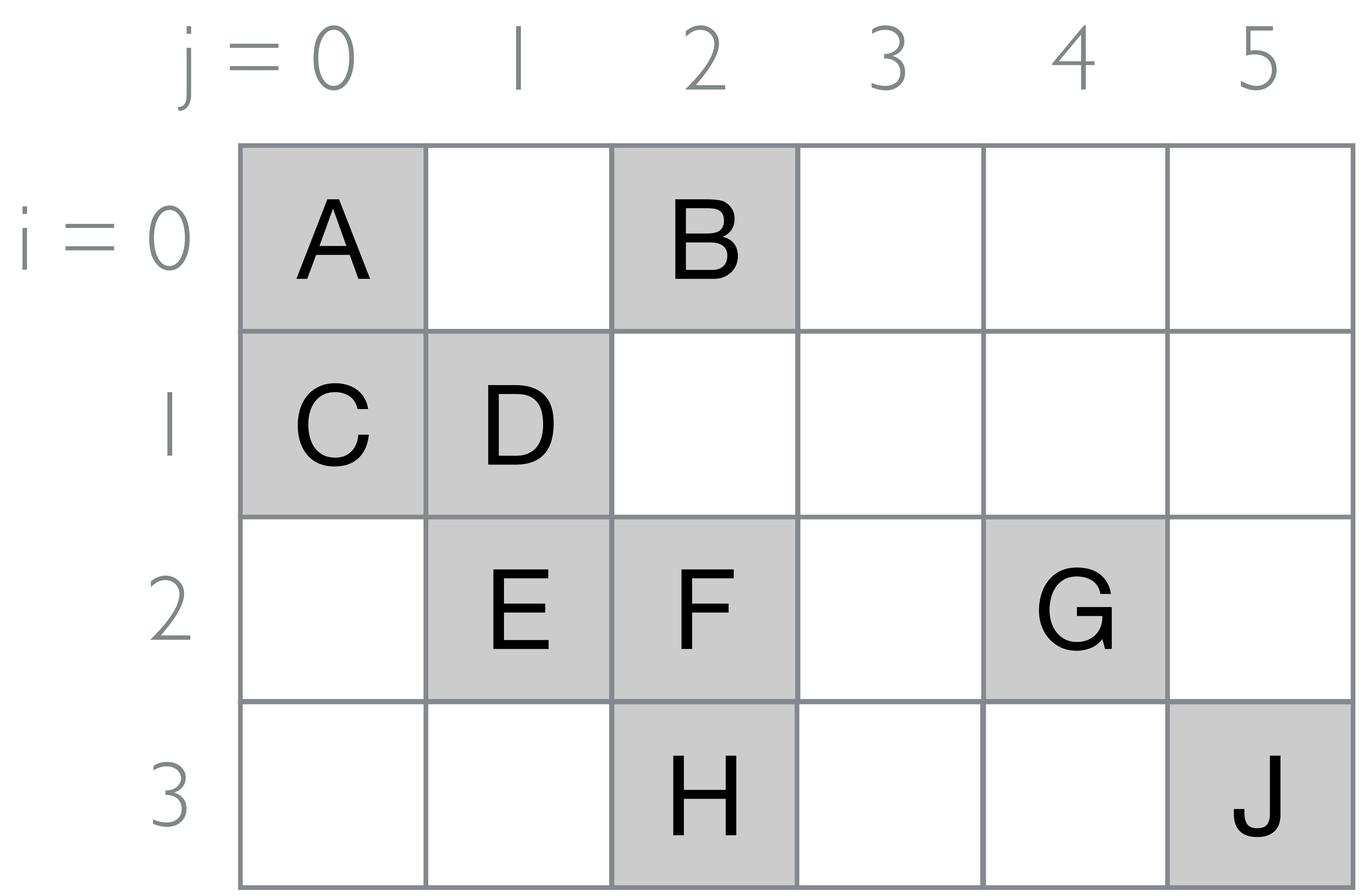


Attribute queries express tensor statistics as aggregations over the coordinates of nonzeros

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

```
select [i] -> count(j) as nnz
```

Attribute queries express tensor statistics as aggregations over the coordinates of nonzeros




```
select [i] -> count(j) as nnz
```

Attribute queries express tensor statistics as aggregations over the coordinates of nonzeros

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

```
select [i] -> count(j) as nnz
```

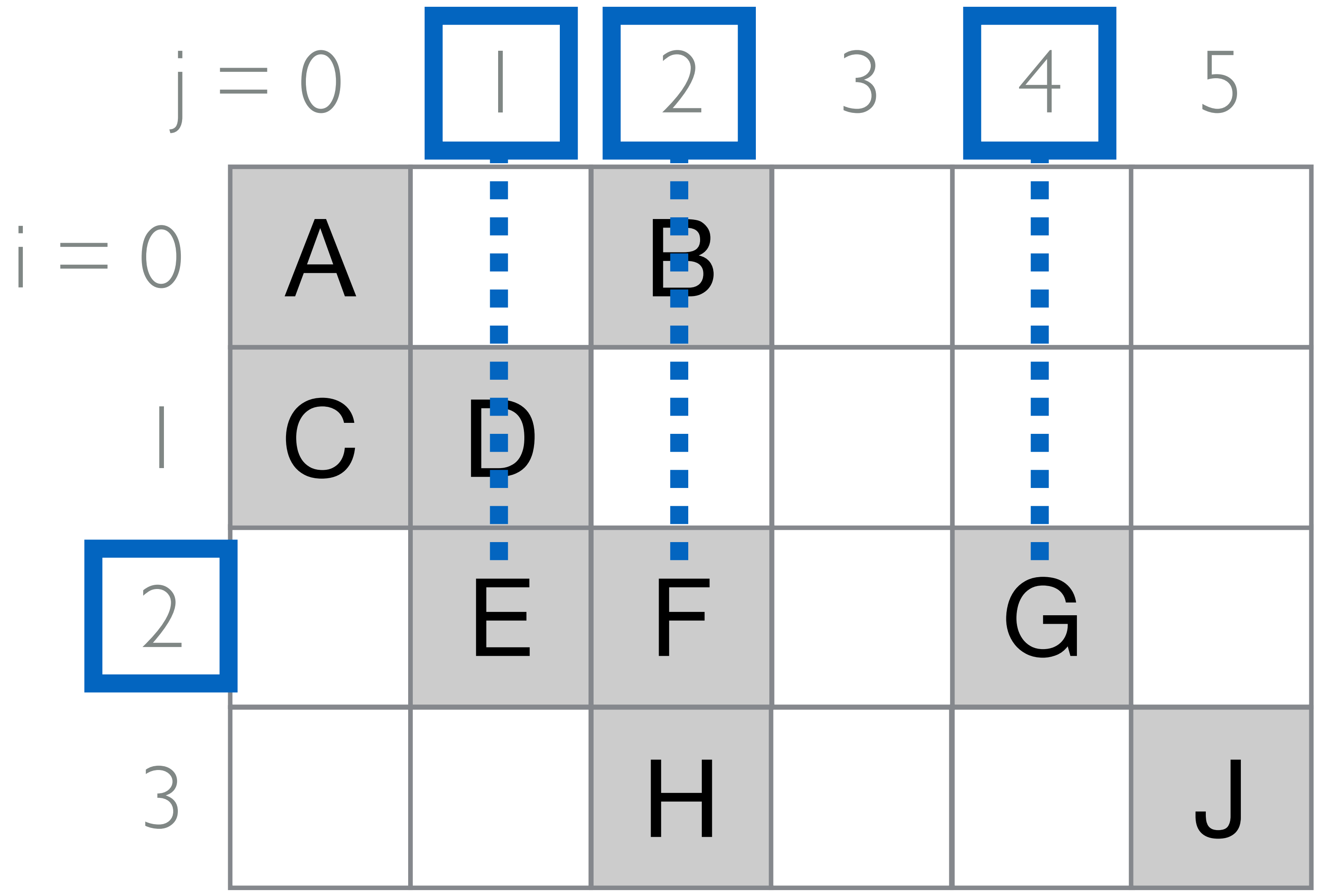
Attribute queries express tensor statistics as aggregations over the coordinates of nonzeros

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

```
select [i] -> count(j) as nnz
```

i	nnz
0	2
1	2
2	3
3	2

Attribute queries express tensor statistics as aggregations over the coordinates of nonzeros




```
select [i] -> count(j) as nnz
```

i	nnz
0	2
1	2
2	3
3	2

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \boxed{\mathrm{map}(B_{ij}, 1)}$$

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

	j = 0	1	2	3	4	5
i = 0	1		1			
1	1	1				
2		1	1		1	
3			1			1

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

	j = 0	1	2	3	4	5
i = 0	A		B			
1	C	D				
2		E	F		G	
3			H			J

	j = 0	1	2	3	4	5
i = 0	1		1			
1	1	1				
2		1	1		1	
3			1			1

	0	1	2	3
Q	2	2	3	2

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\boxed{\forall_i \forall_j} Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

B is CSC

```
for (int j = 0; j < N; j++) {
  for (int pB = pos[j];
           pB < pos[j+1]; pB++) {
    int i = crd[pB2];

  }
}
```

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \boxed{Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)}$$

B is CSC

```
for (int j = 0; j < N; j++) {
  for (int pB = pos[j];
           pB < pos[j+1]; pB++) {
    int i = crd[pB2];
    Q[i] += 1;
  }
}
```

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

B is CSC

```
for (int j = 0; j < N; j++) {
  for (int pB = pos[j];
           pB < pos[j+1]; pB++) {
    int i = crd[pB2];
    Q[i] += 1;
  }
}
```

B is COO

```
for (int pB = 0;
         pB < NNZ; pB++) {
  int i = rows[pB];
  Q[i] += 1;
}
```

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

B is CSR

$$B'_i \equiv (\texttt{pos[i+1]} - \texttt{pos[i]})$$

$$\forall_i \; Q_i = B'_i$$

B is CSC

```
for (int j = 0; j < N; j++) {
  for (int pB = pos[j];
           pB < pos[j+1]; pB++) {
    int i = crd[pB2];
    Q[i] += 1;
  }
}
```

B is COO

```
for (int pB = 0;
         pB < NNZ; pB++) {
  int i = rows[pB];
  Q[i] += 1;
}
```

Compiler generates code to compute attribute queries by reducing them to sparse tensor computations

```
select [i] -> count(j) as Q
```

$$\forall_i \forall_j \; Q_i \mathrel{+}= \mathrm{map}(B_{ij}, 1)$$

B is CSR

$$B'_i \equiv (\texttt{pos[i+1]} - \texttt{pos[i]})$$

$$\forall_i \; Q_i = B'_i$$

```
for (int i = 0; i < N; i++) {
  Q[i] = pos[i+1] - pos[i];
}
```

B is CSC

```
for (int j = 0; j < N; j++) {
  for (int pB = pos[j];
           pB < pos[j+1]; pB++) {
    int i = crd[pB2];
    Q[i] += 1;
  }
}
```

B is COO

```
for (int pB = 0;
         pB < NNZ; pB++) {
  int i = rows[pB];
  Q[i] += 1;
}
```

In conclusion…

Efficient sparse tensor conversion routines can be automatically generated from per-format specifications

Artifacts Evaluated acm Functional

tensor-compiler.org

Artifacts Evaluated acm Reusable

This work was supported by:

DARPA

TOYOTA RESEARCH INSTITUTE

NSF

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

SRC® Semiconductor Research Corporation

In conclusion…

Efficient sparse tensor conversion routines can be automatically generated from per-format specifications

Adding support for new sparse tensor formats is straightforward

tensor-compiler.org

This work was supported by:

In conclusion…

Efficient sparse tensor conversion routines can be automatically generated from per-format specifications

Adding support for new sparse tensor formats is straightforward

Our technique makes it simple to fully exploit disparate tensor formats for performance

tensor-compiler.org

This work was supported by: