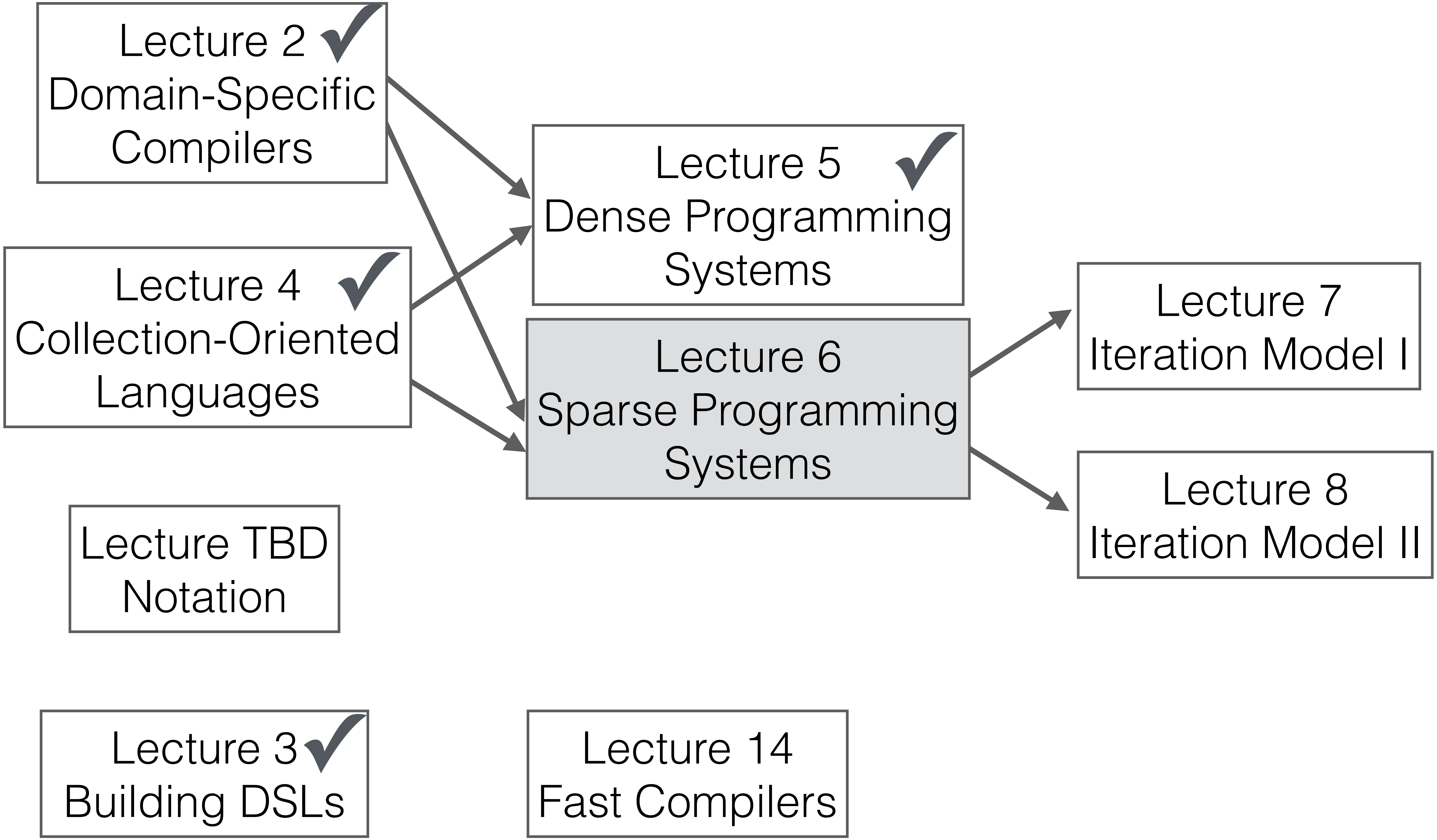
Lecture 6 — Sparse Programming Systems

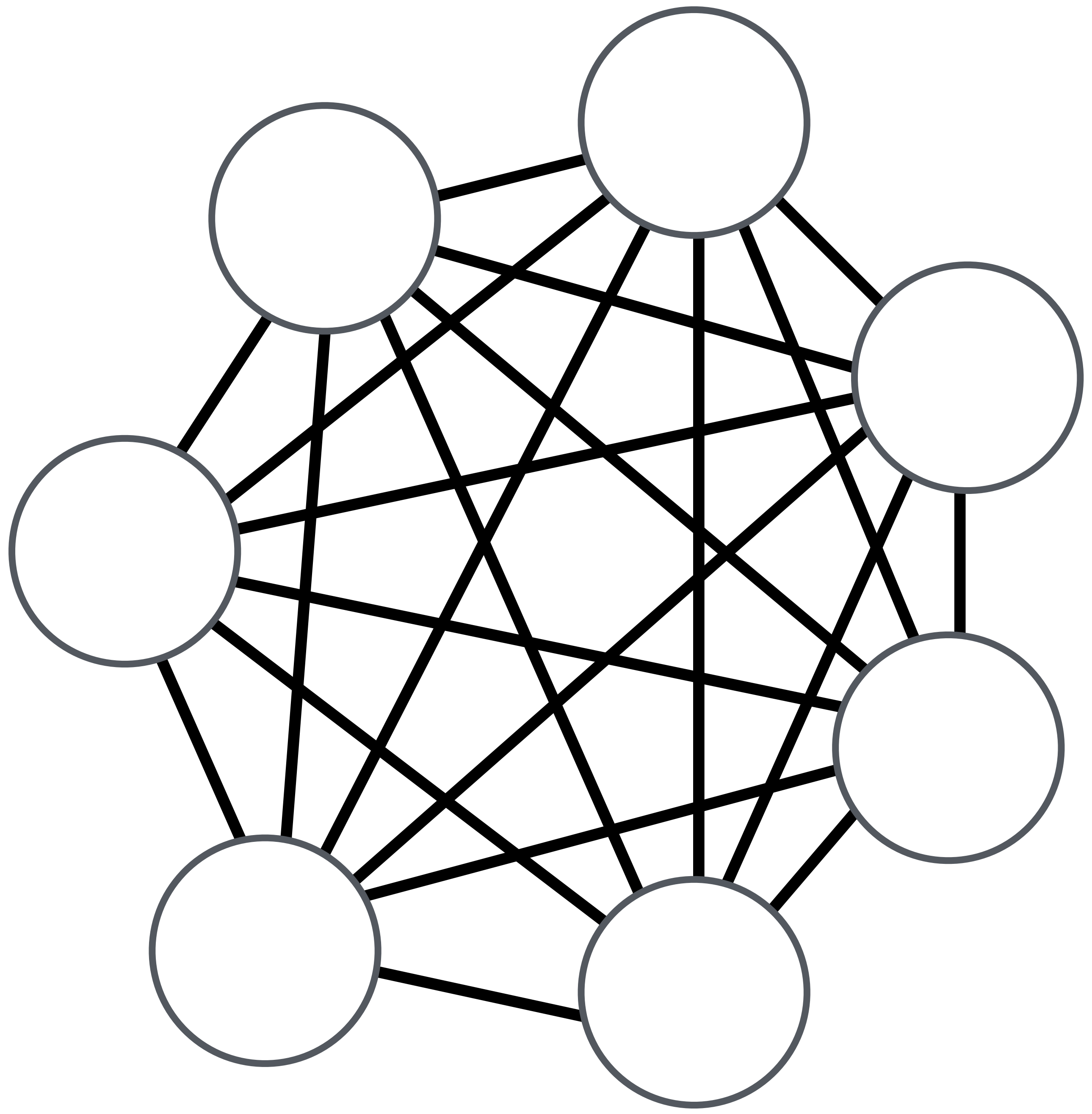
Stanford CS343D (Winter 2023)
Fred Kjolstad

Lecture Overview

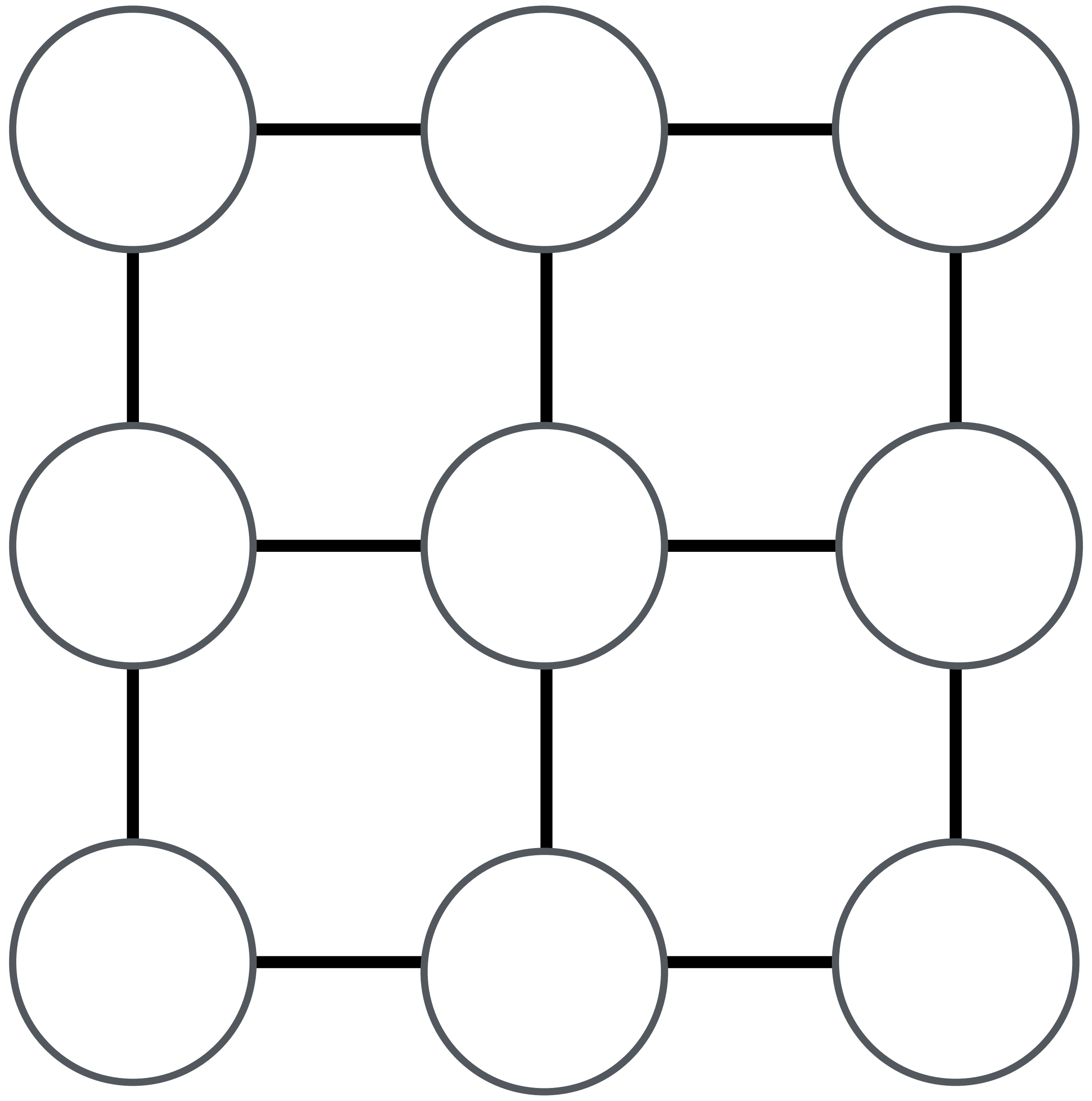


Terminology: Regular and Irregular

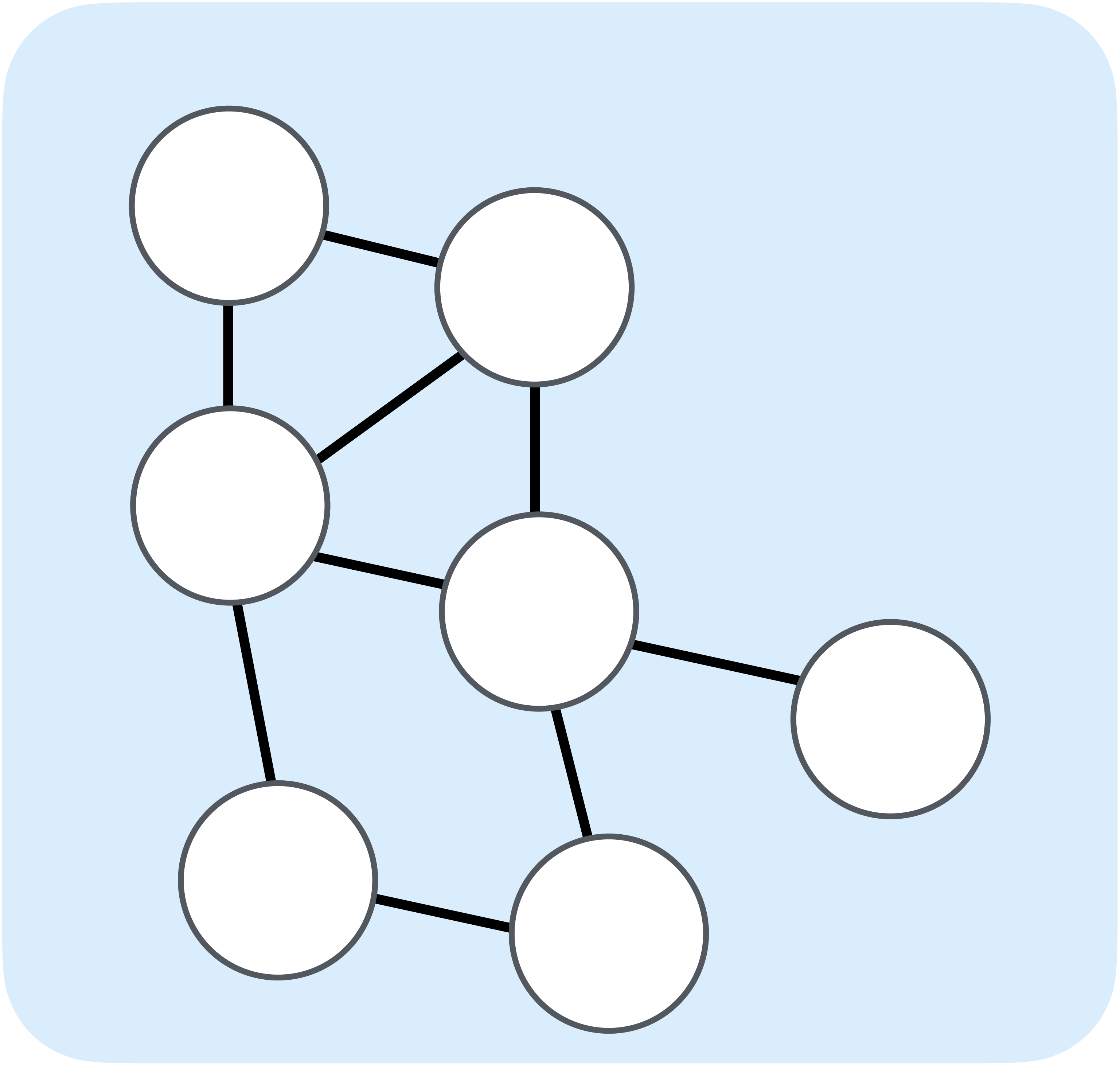
Fully Connected System



Regular System

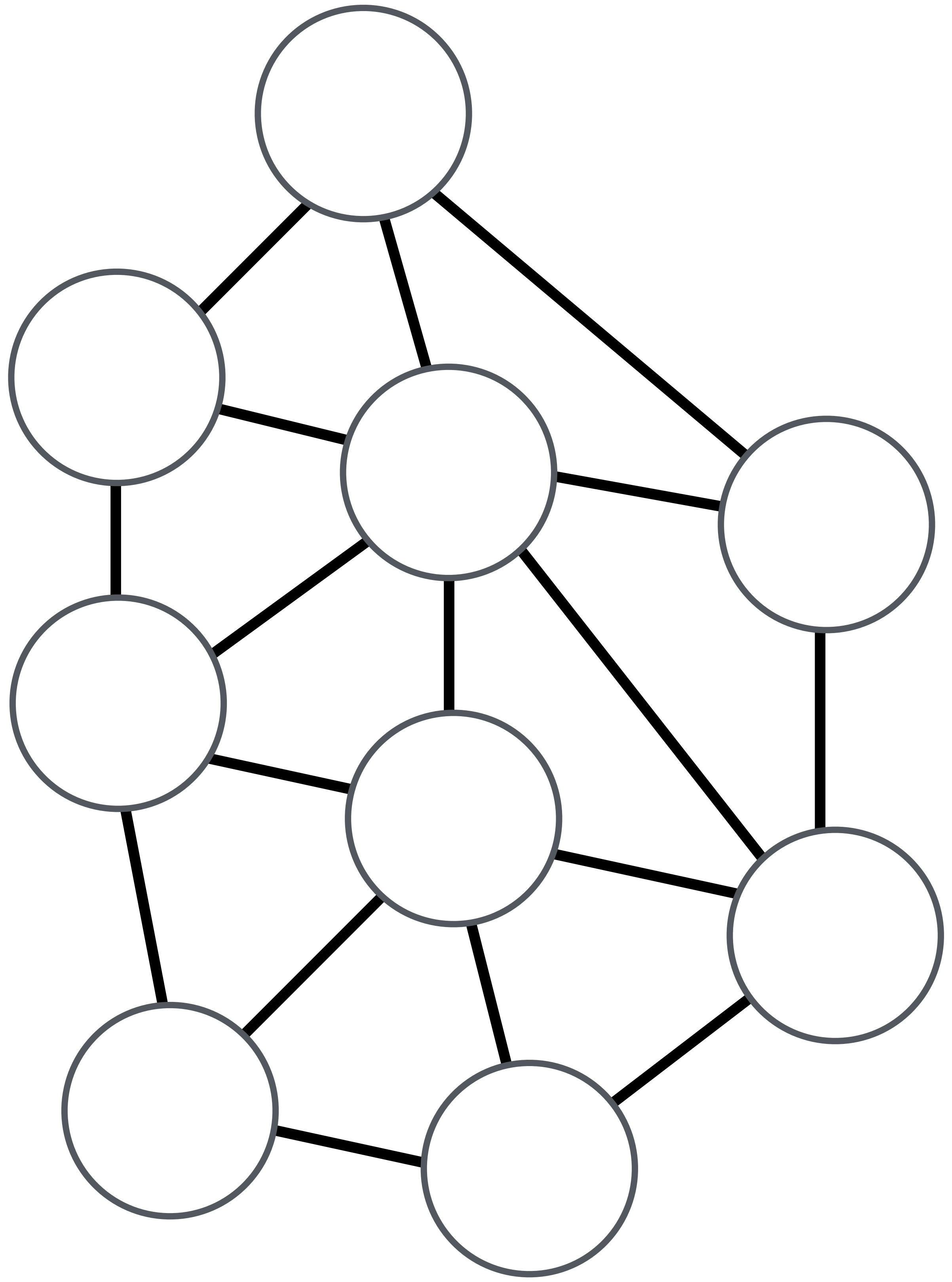


Irregular System

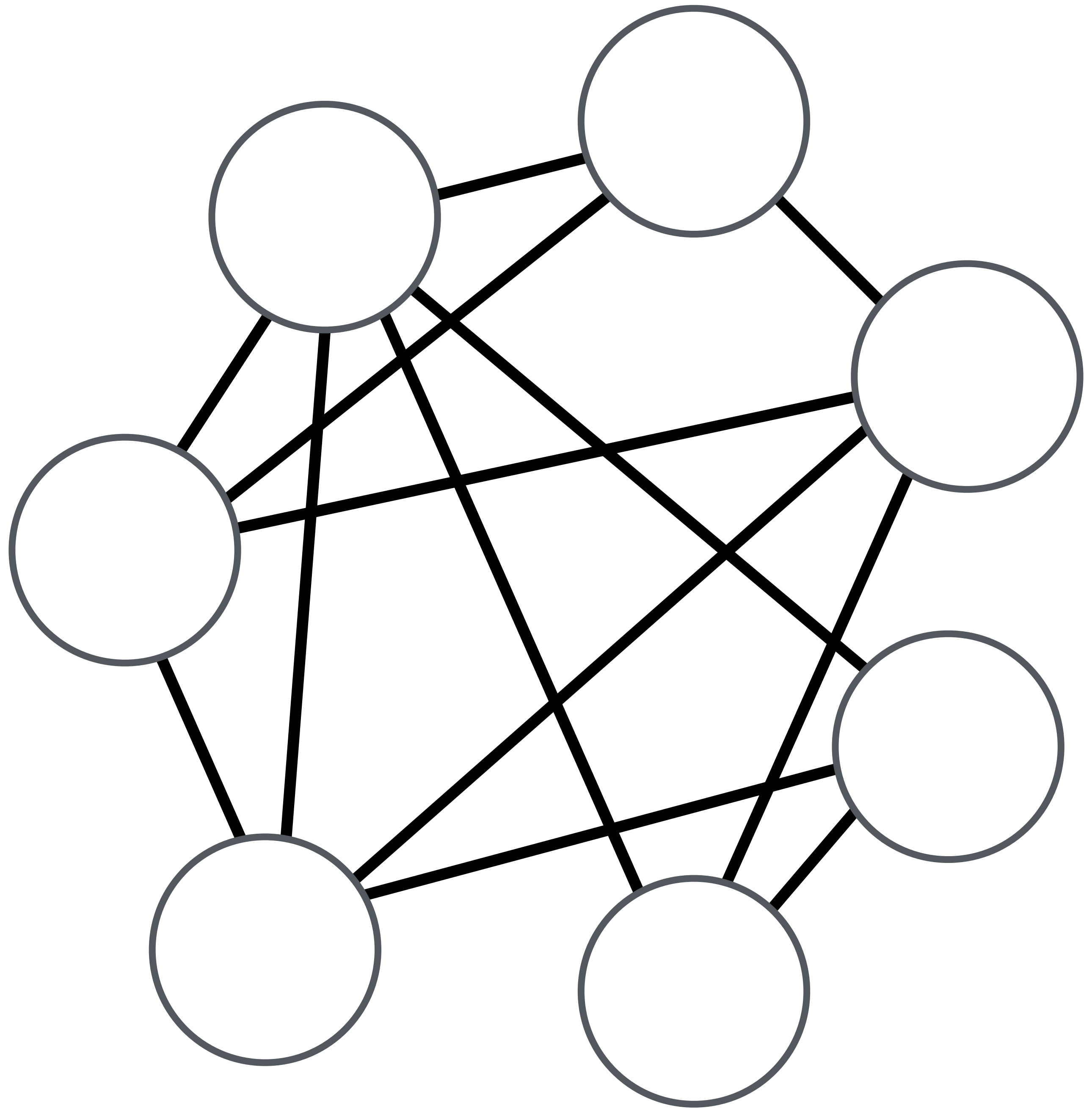


Three classes of irregular systems

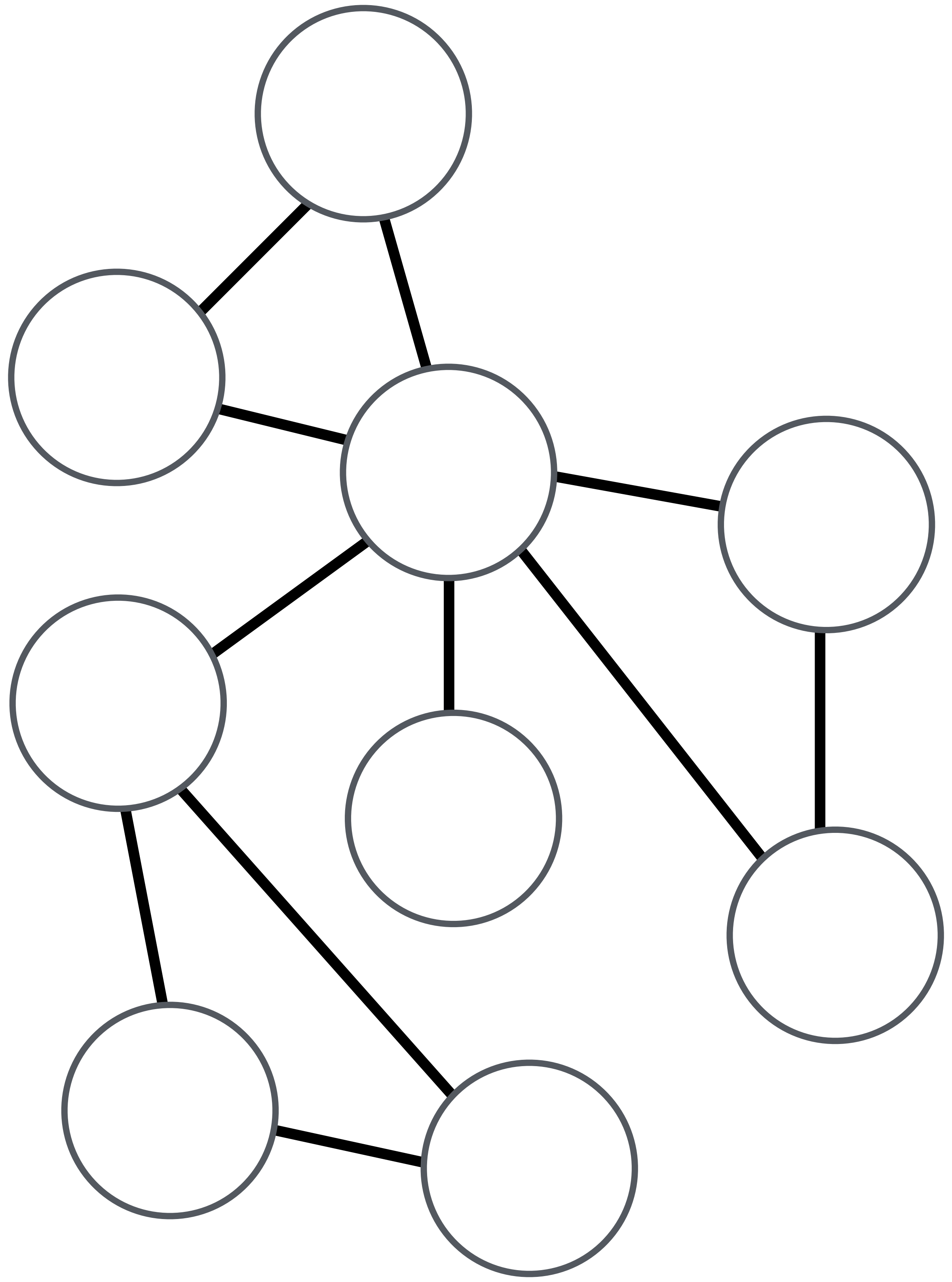
Road Networks



Fractional Sparsity

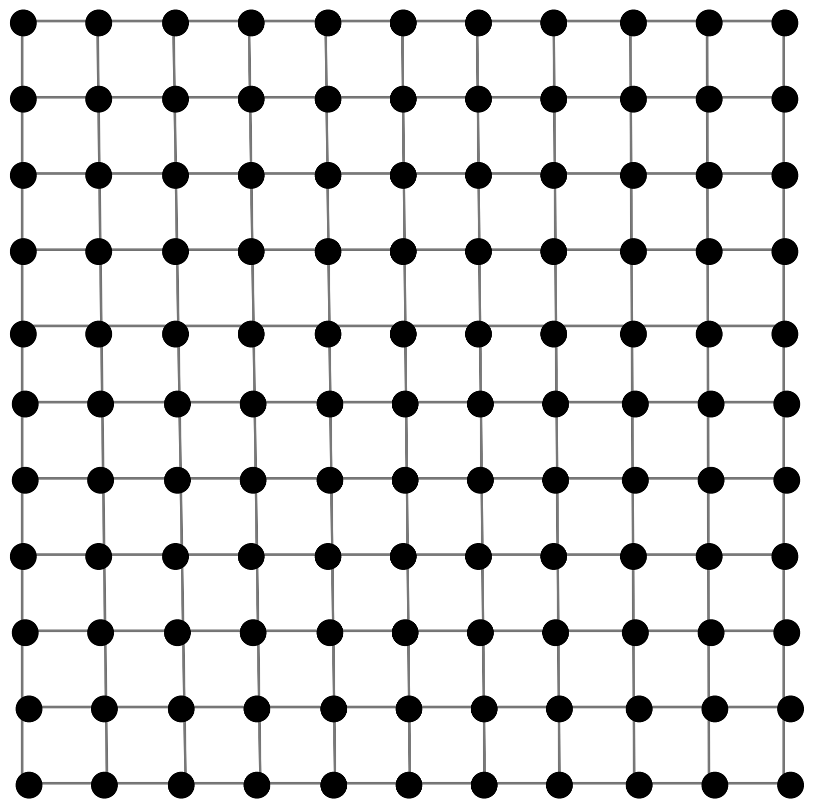


Power Law Graphs



Terminology: Dense and Sparse

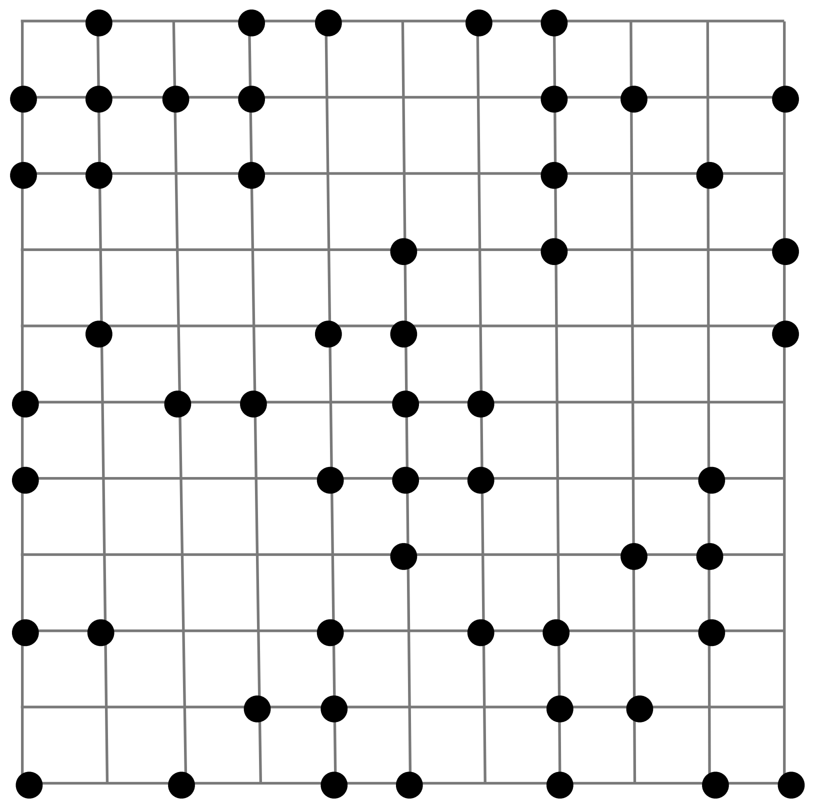
Dense loop iteration space



```
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; j++) {
    y[i] += A[i*n+j] * x[j];
  }
}
```

$$y = Ax$$

Sparse loop iteration space

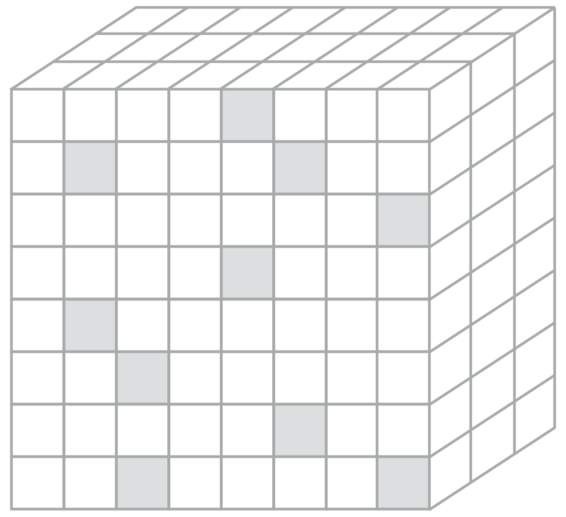


```
for (int i = 0; i < m; i++) {
 for (int pA = A2_pos[i]; pA < A_pos[i+1]; pA++) {
    int j = A_crd[pA];
    y[i] += A[pA] * x[j];
  }
}
```

$$y = Ax$$

Three sparse applications areas

Tensors



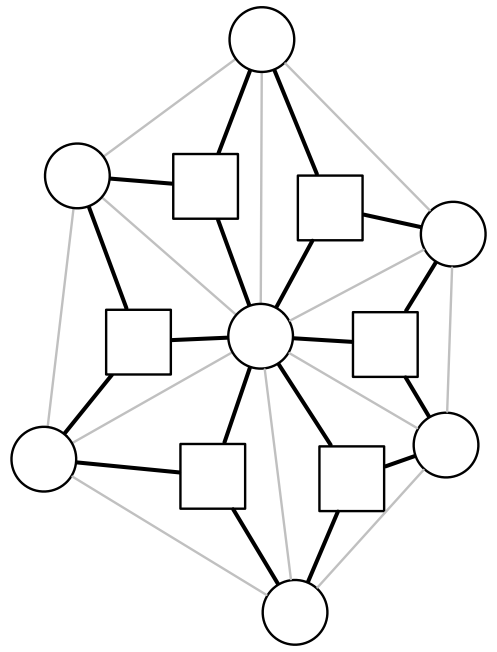
Nonzeros are a subset of the cartesian combination of sets

Relations

Names	City	Age
Peter	Boston	54
Mary	San Fransisco	35
Paul	New York	23
Adam	Seattle	84
Hilde	Boston	19
Bob	Chicago	76
Sam	Portland	32
Angela	Los Angeles	62

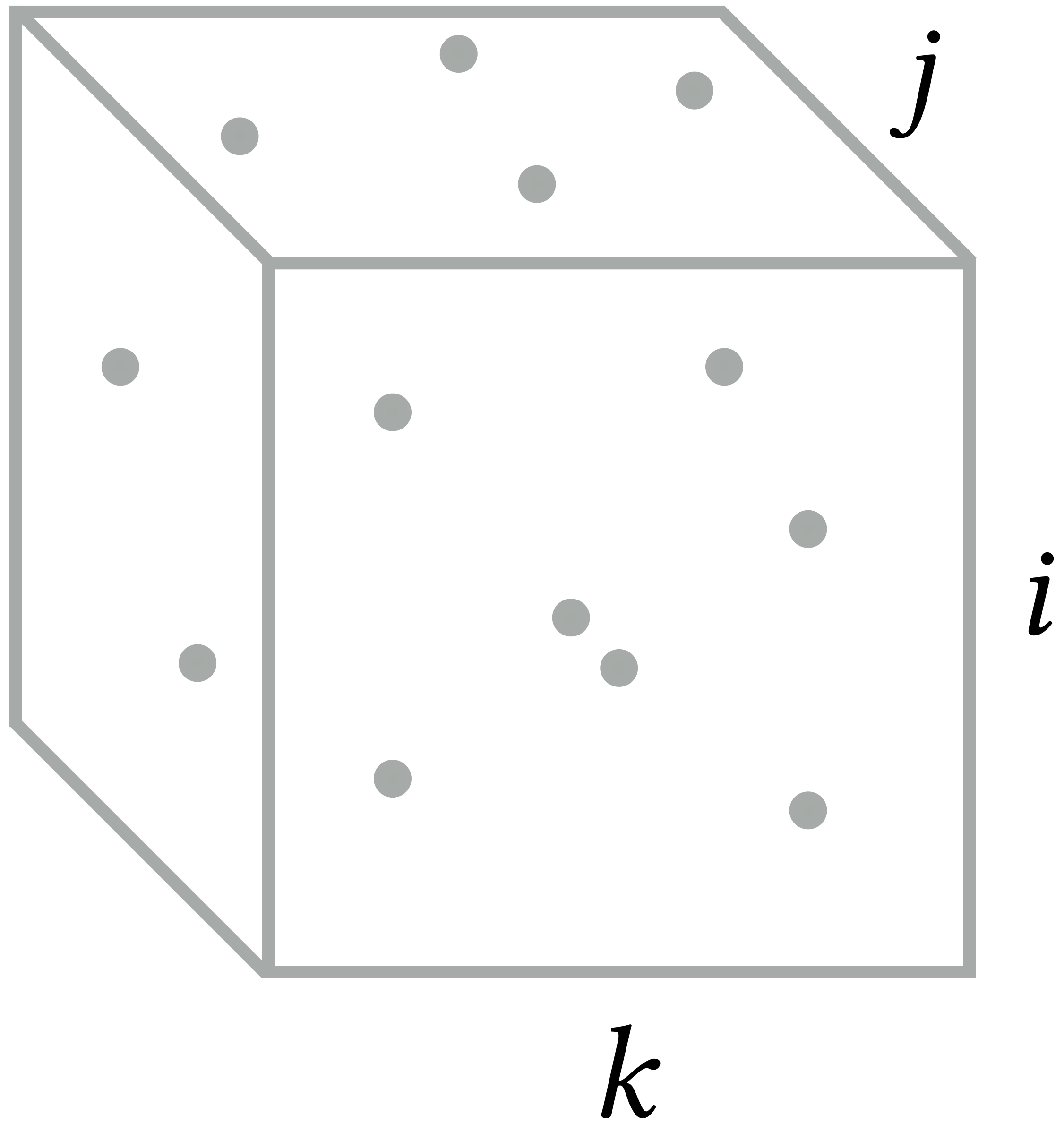
A relation is a subset of the cartesian combination of sets

Graphs



Graph edges are a subset of the cartesian combination of sets

Sparse Iteration Spaces



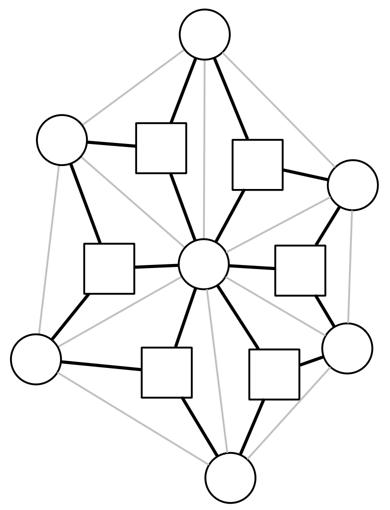
Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

Relations

Names	City	Age
Peter	Boston	54
Mary	San Fransisco	35
Paul	New York	23
Adam	Seattle	84
Hilde	Boston	19
Bob	Chicago	76
Sam	Portland	32
Angela	Los Angeles	62

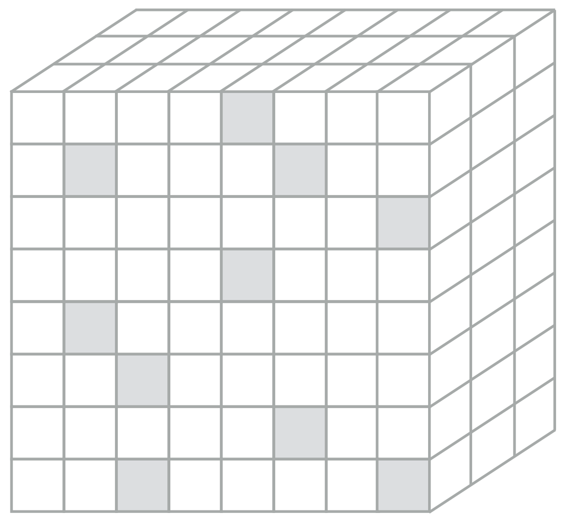
Combine data to form systems

Graphs

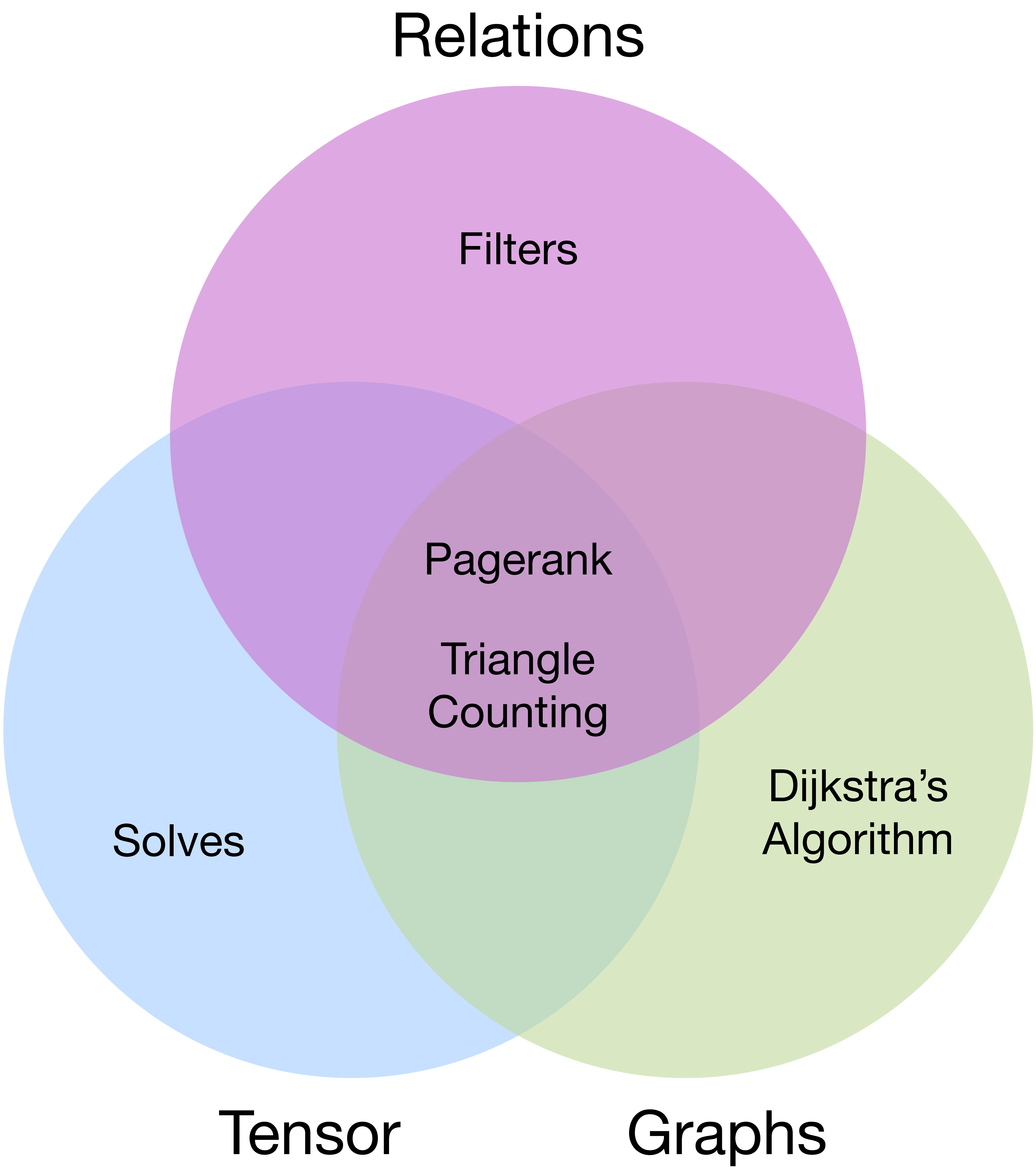


Local operations on systems

Tensors

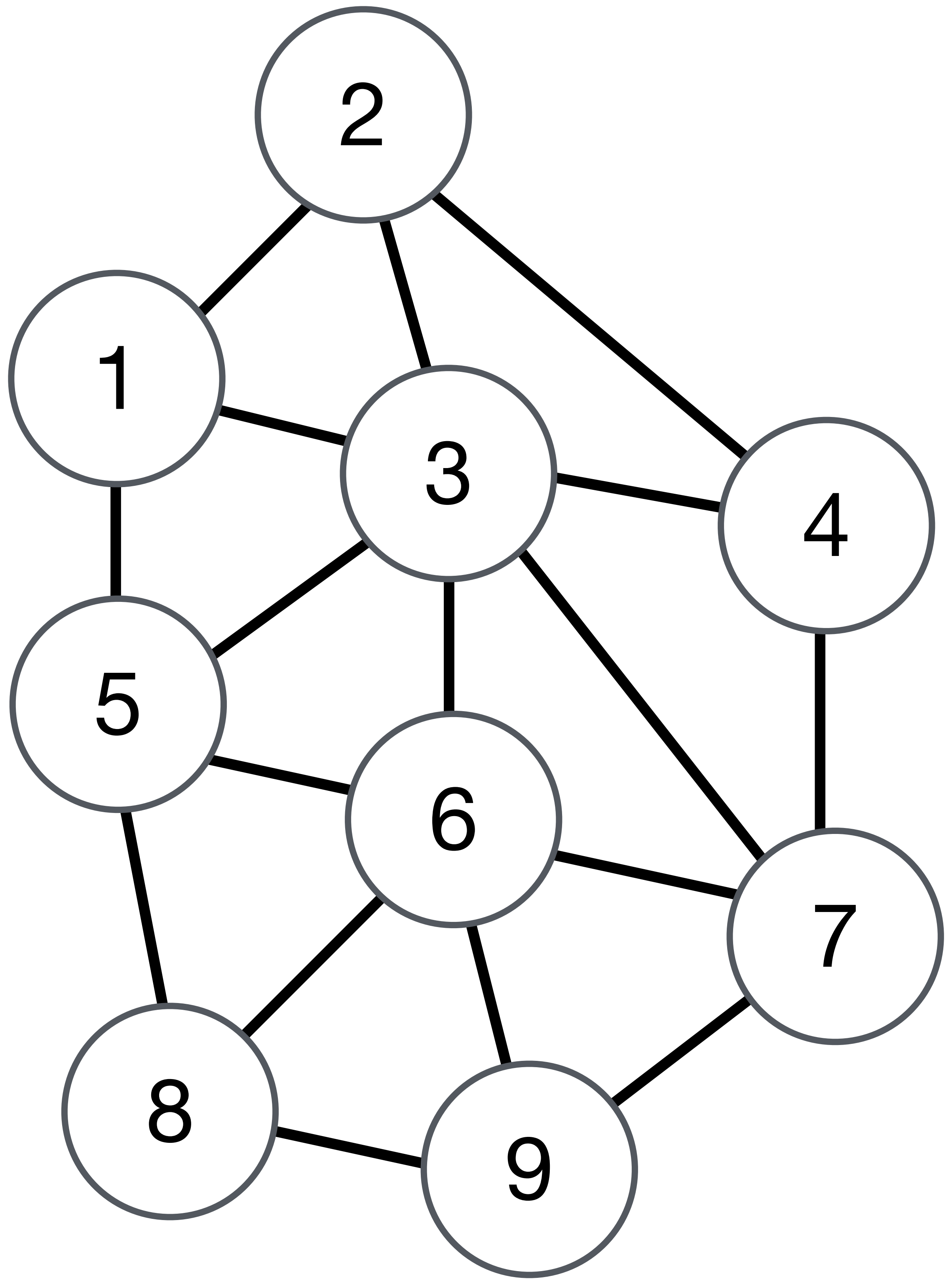


Global operations on systems



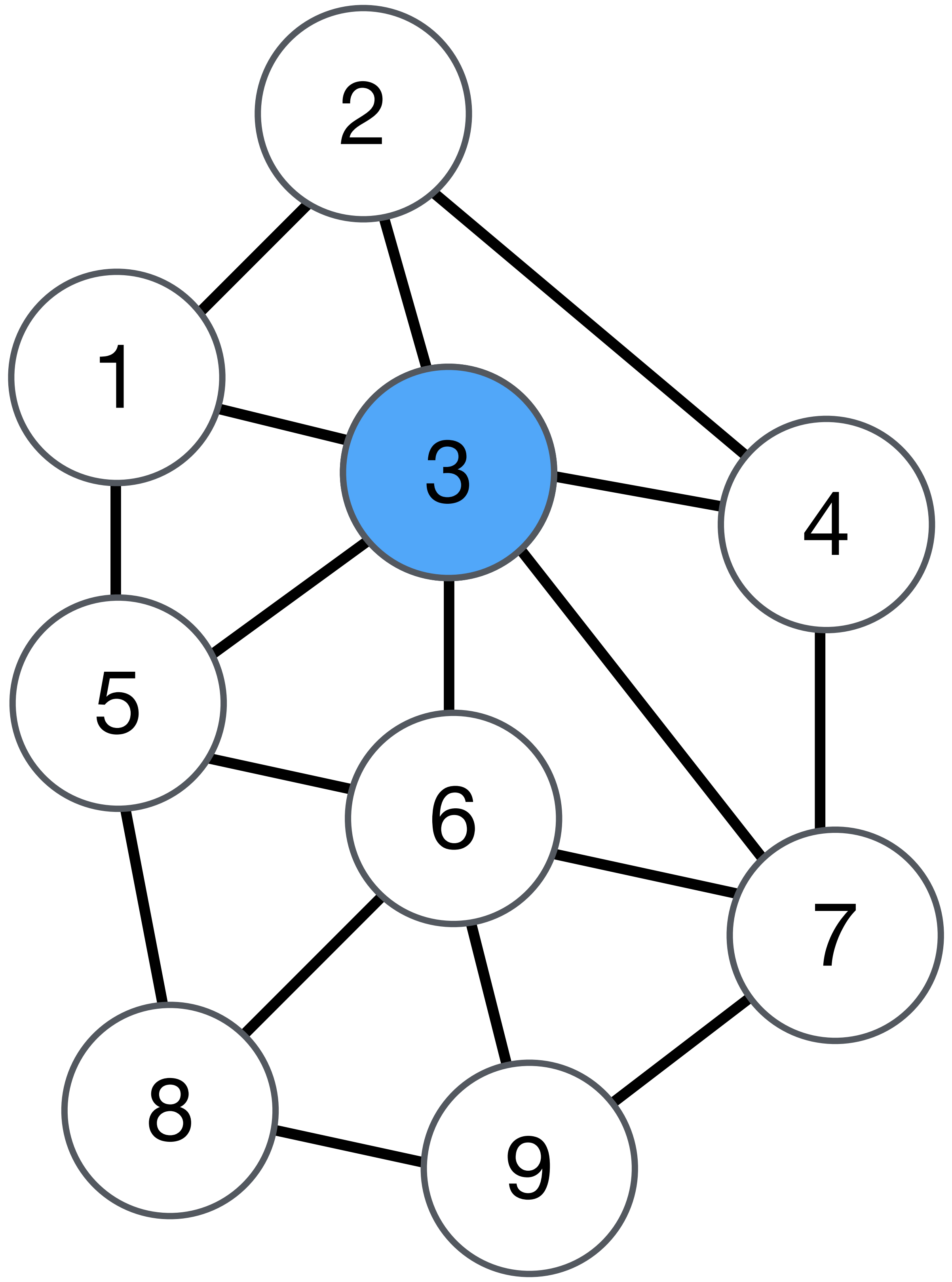
Triangle counting on graphs, relations, and tensors

On graphs



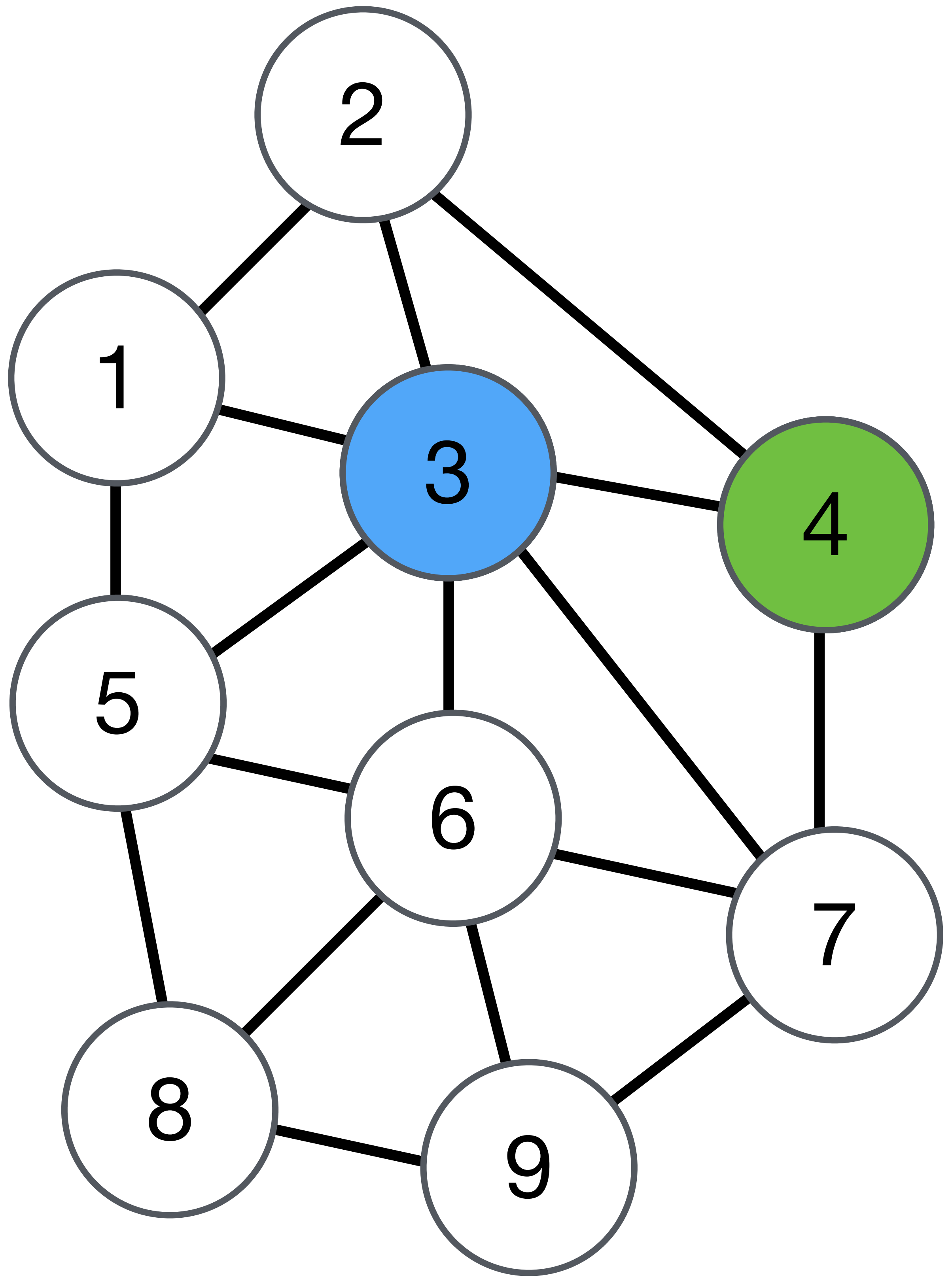
Triangle counting on graphs, relations, and tensors

On graphs



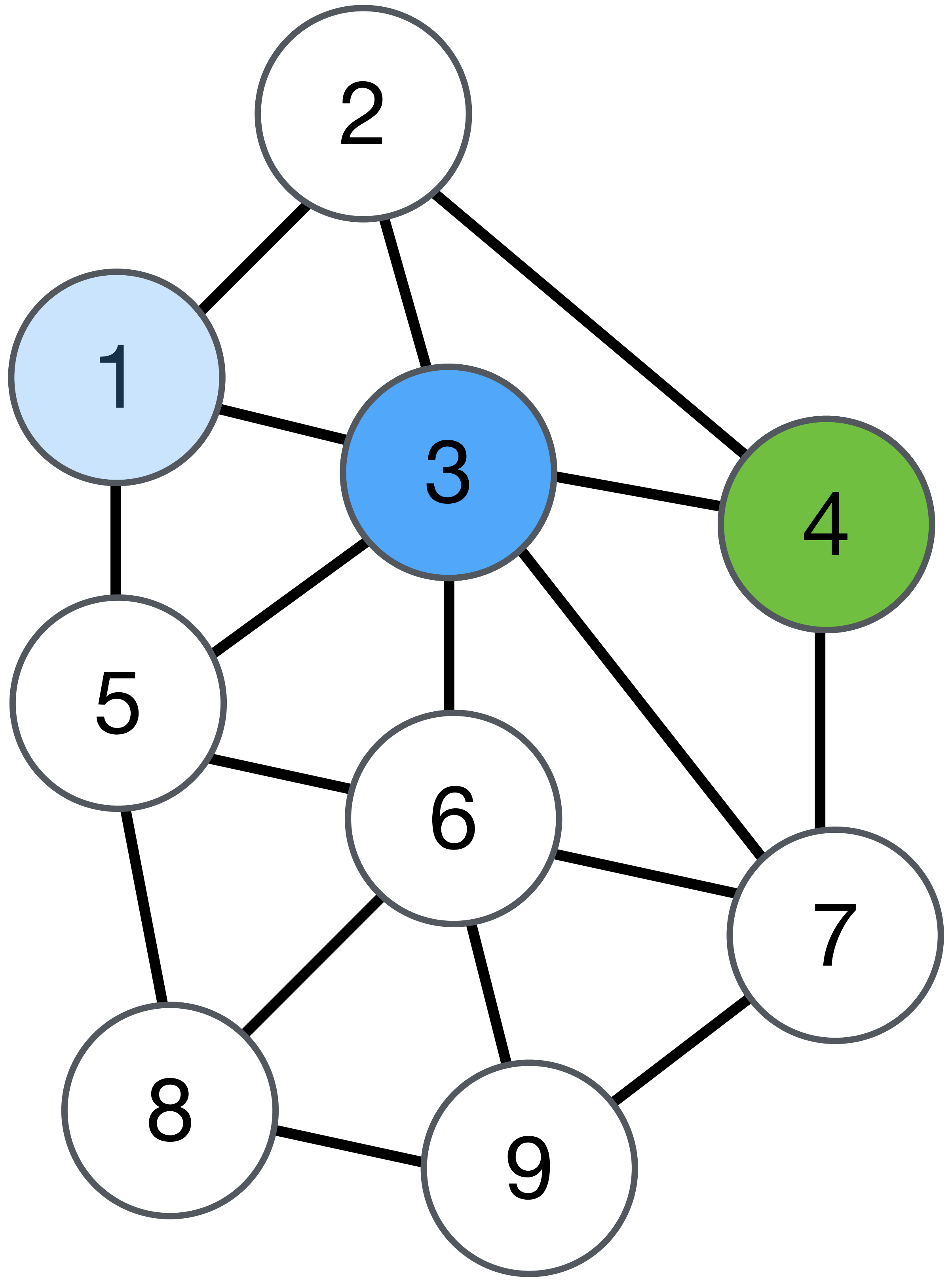
Triangle counting on graphs, relations, and tensors

On graphs



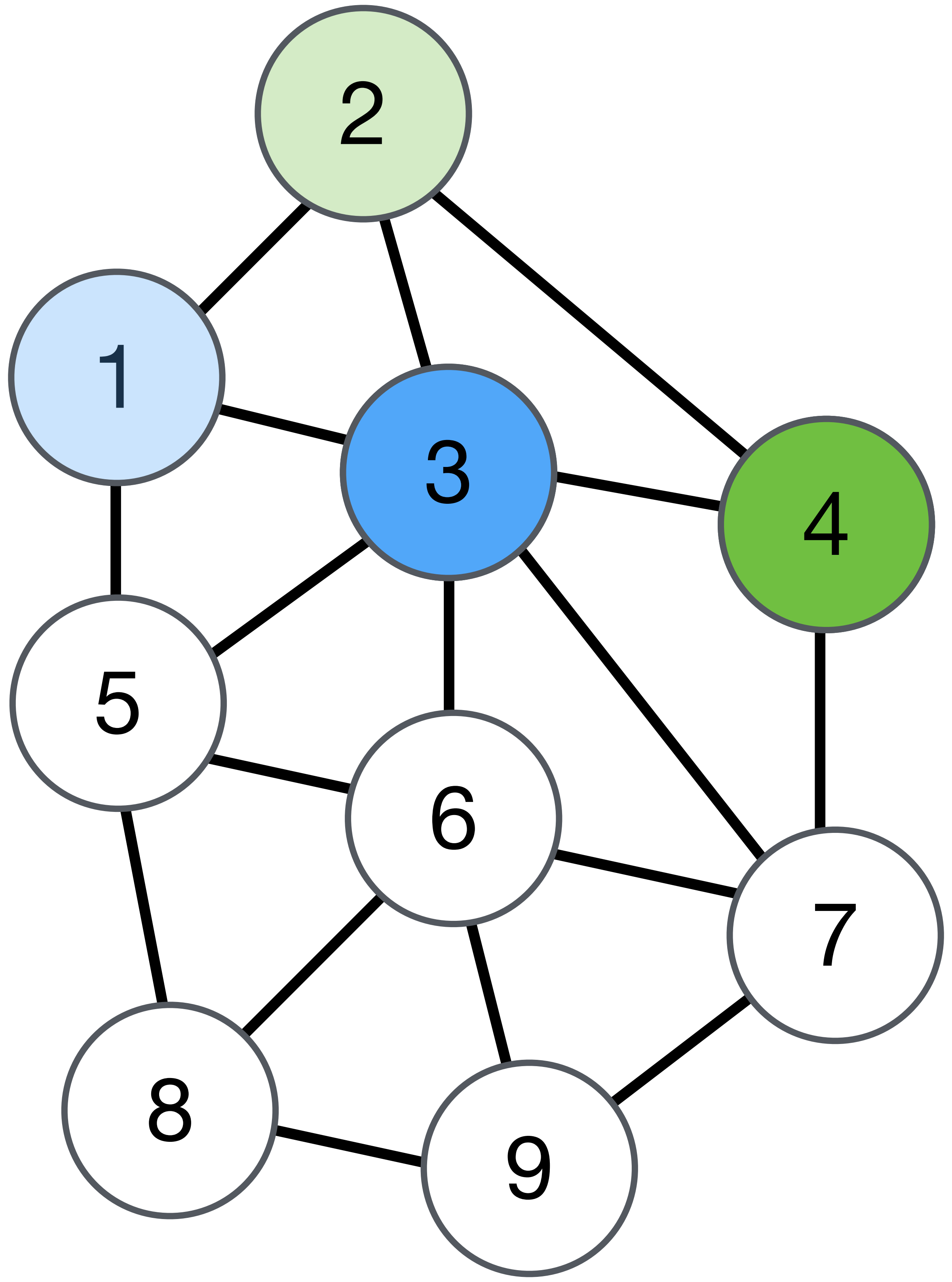
Triangle counting on graphs, relations, and tensors

On graphs



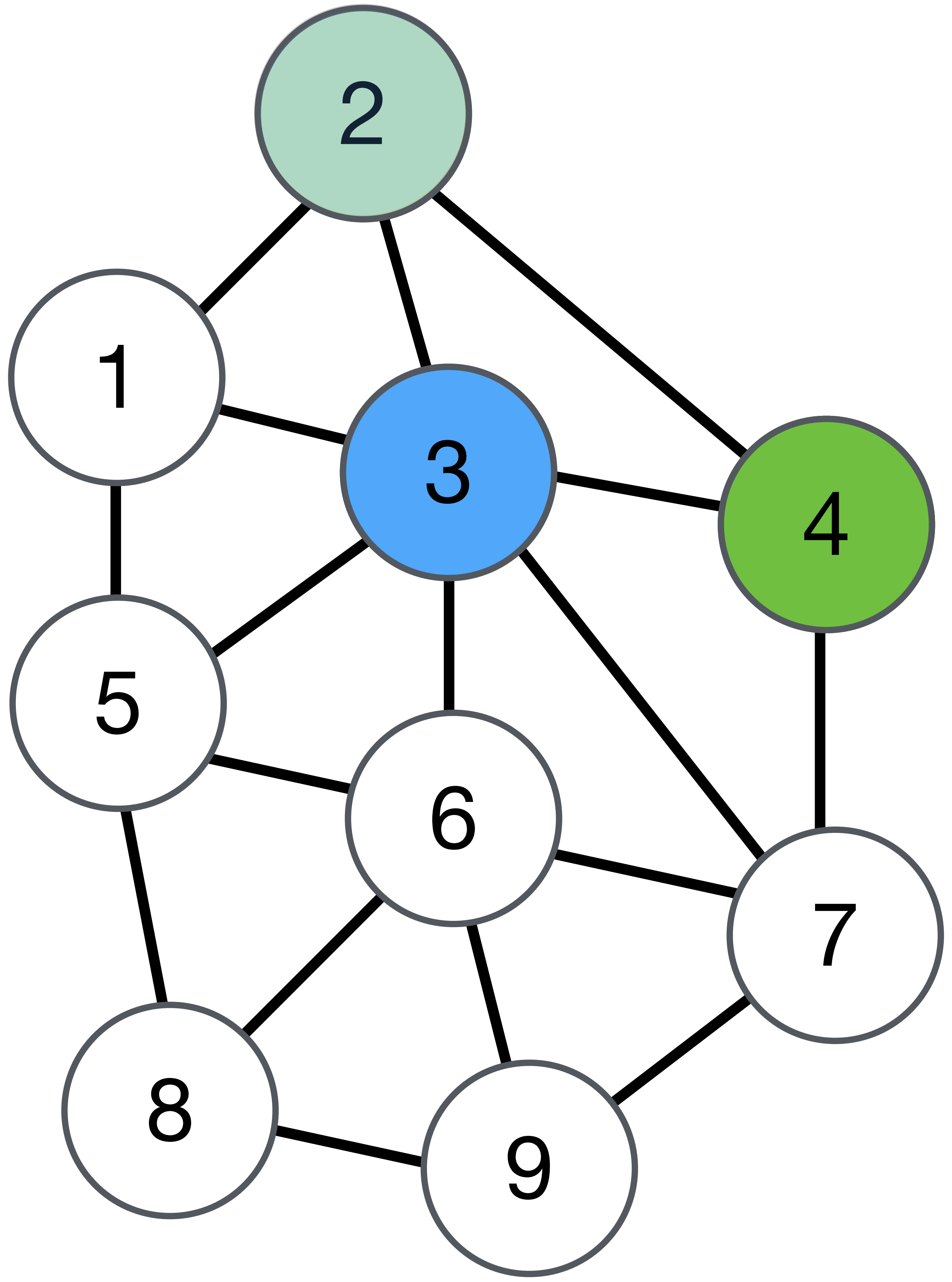
Triangle counting on graphs, relations, and tensors

On graphs



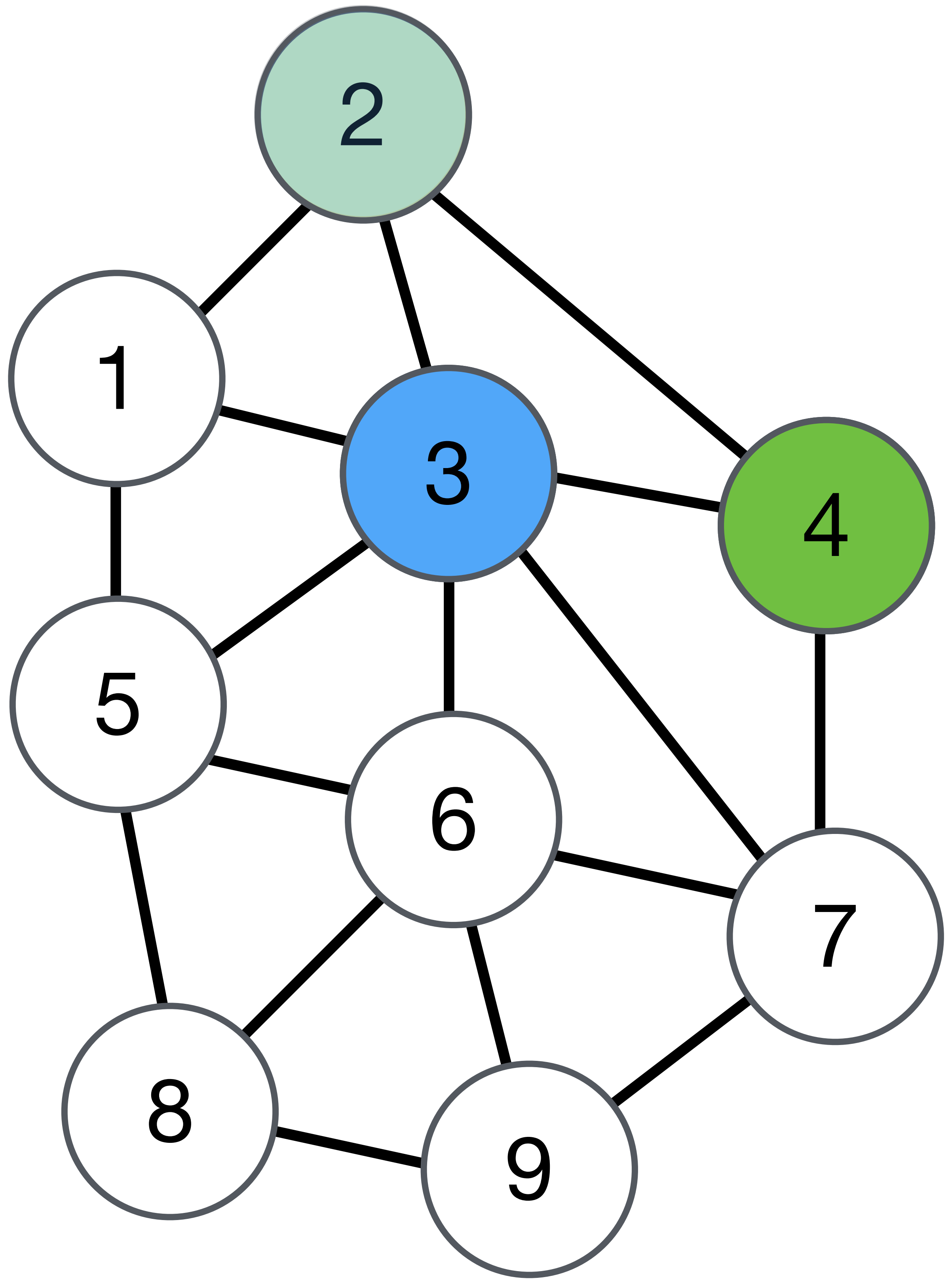
Triangle counting on graphs, relations, and tensors

On graphs



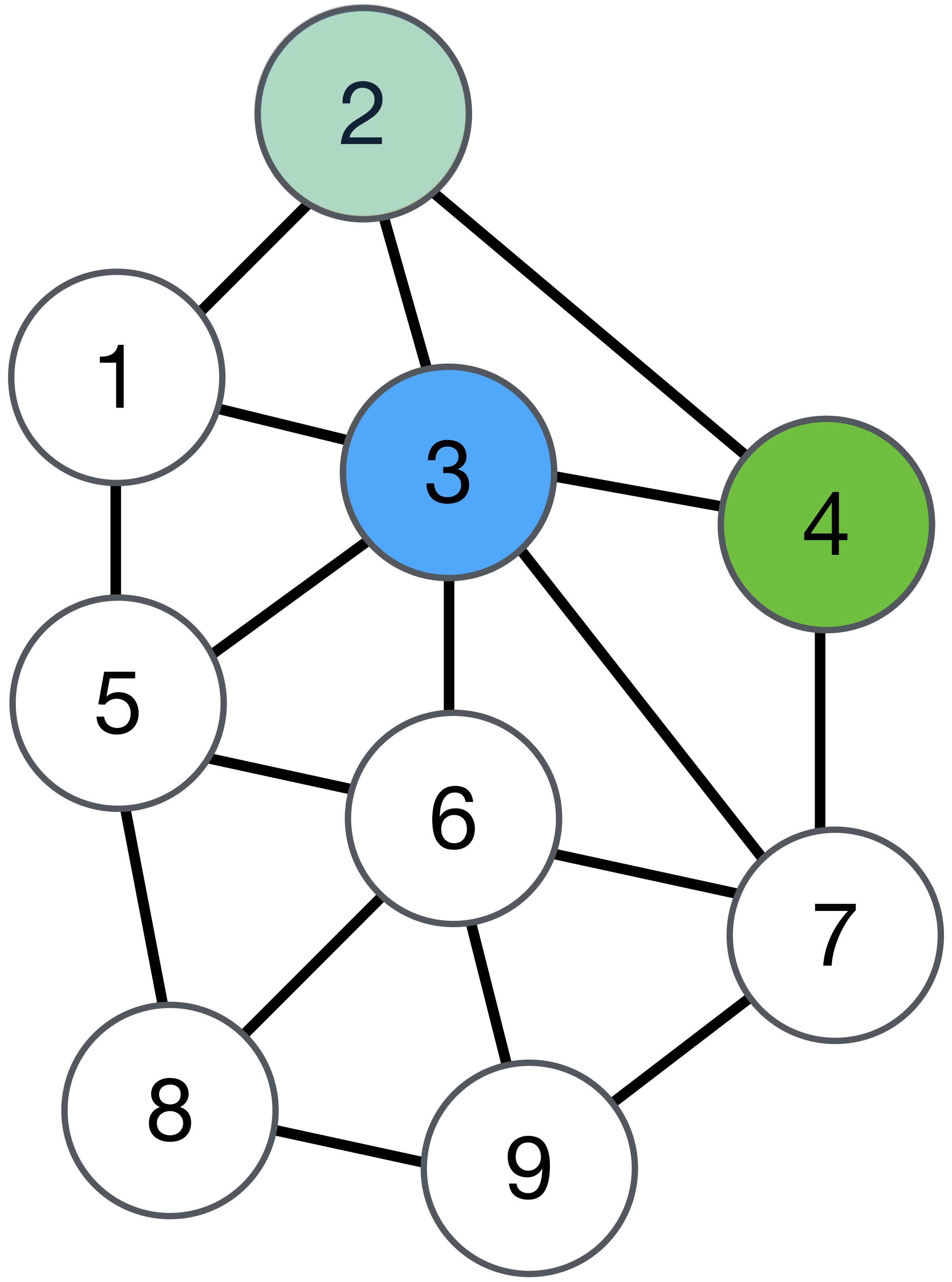
Triangle counting on graphs, relations, and tensors

On graphs



Triangle counting on graphs, relations, and tensors

On graphs

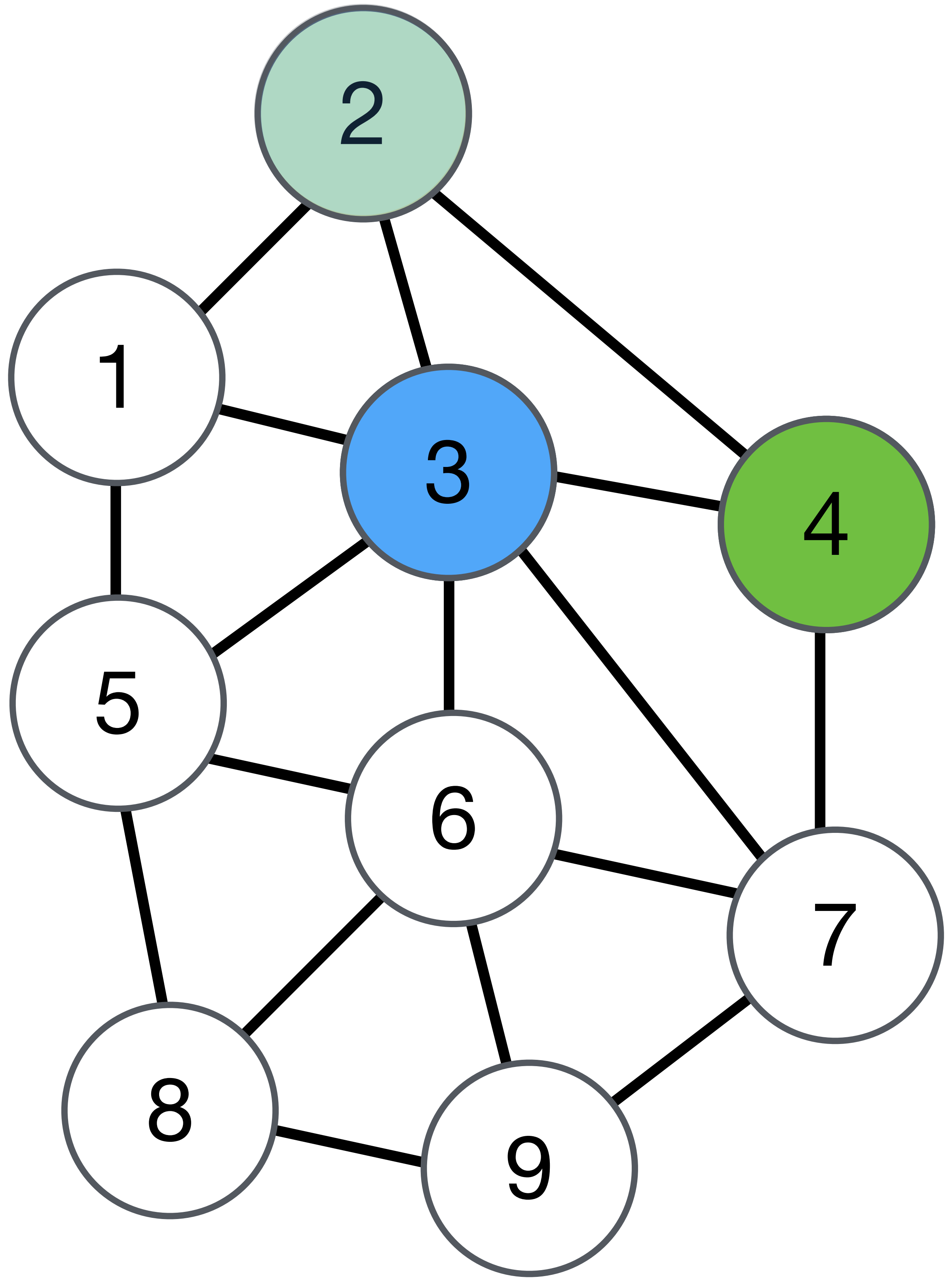


On relations

$$Q_{\triangle} = E(A,B) \bowtie E(B,C) \bowtie E(C,A)$$

Triangle counting on graphs, relations, and tensors

On graphs

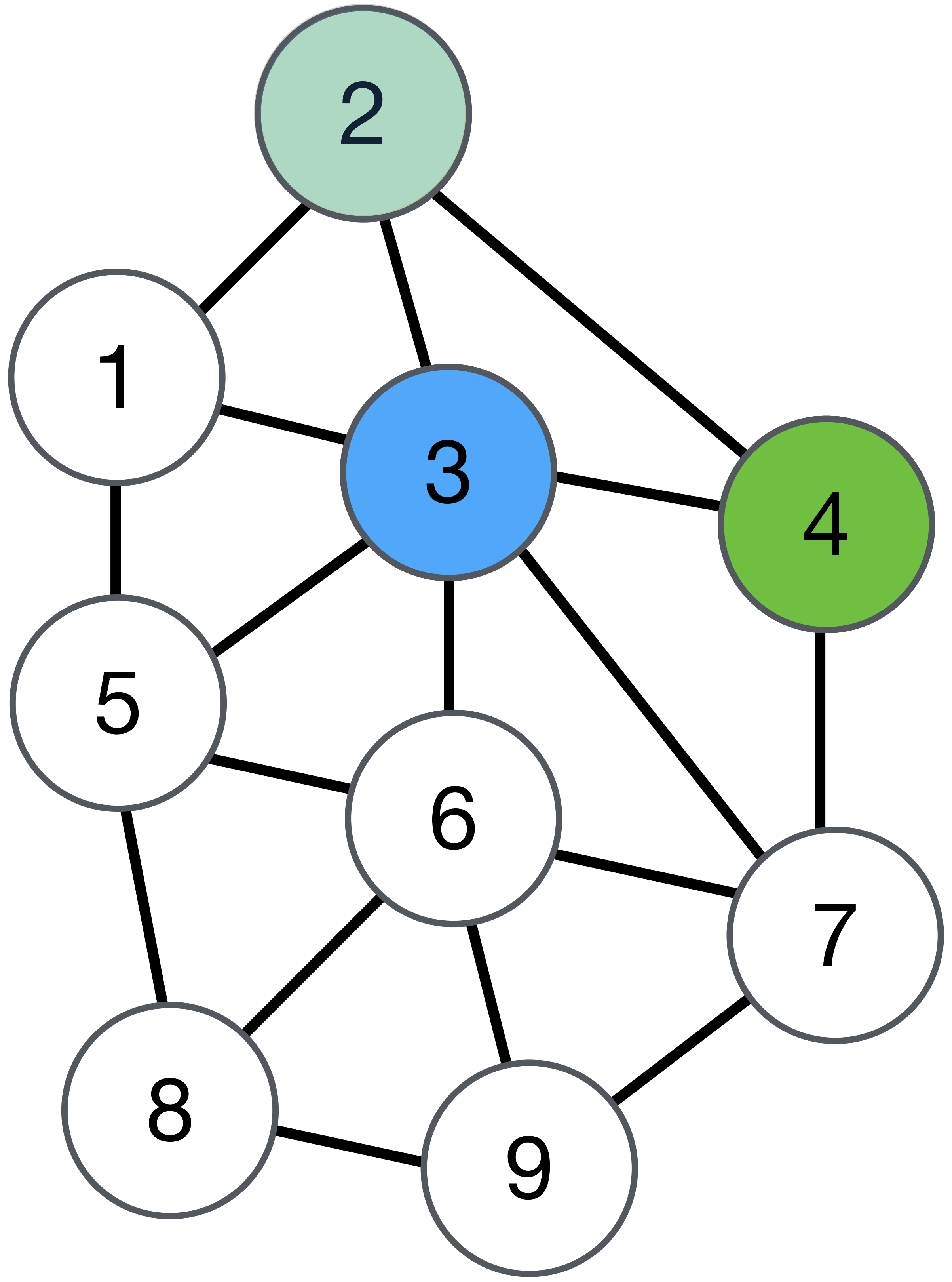


On relations

$$Q_{\triangle} = E(A,B) \bowtie E(B,C) \bowtie E(C,A)$$

Triangle counting on graphs, relations, and tensors

On graphs

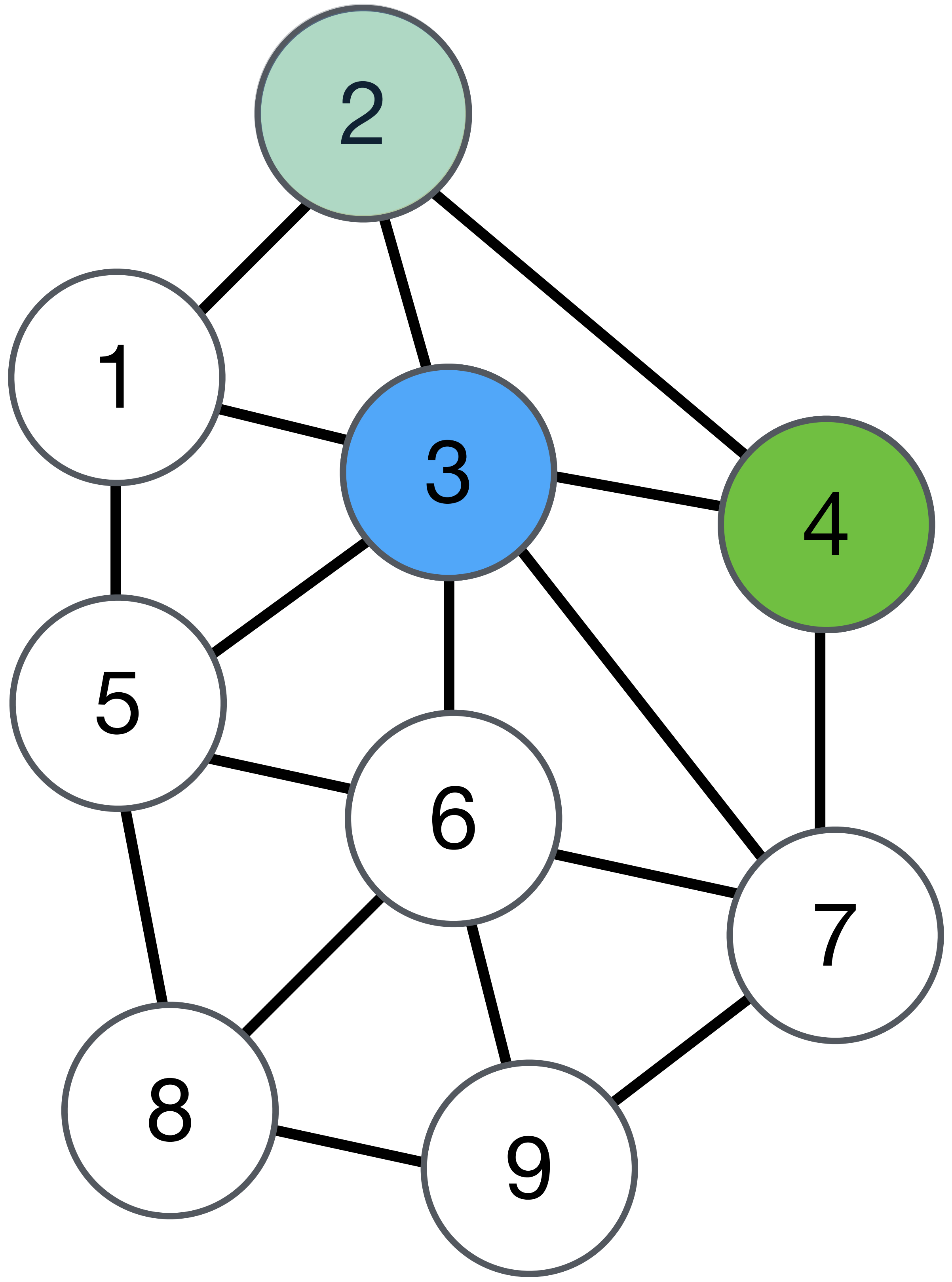


On relations

$$Q_{\triangle} = E(A, B) \bowtie E(B, C) \bowtie E(C, A)$$

Triangle counting on graphs, relations, and tensors

On graphs

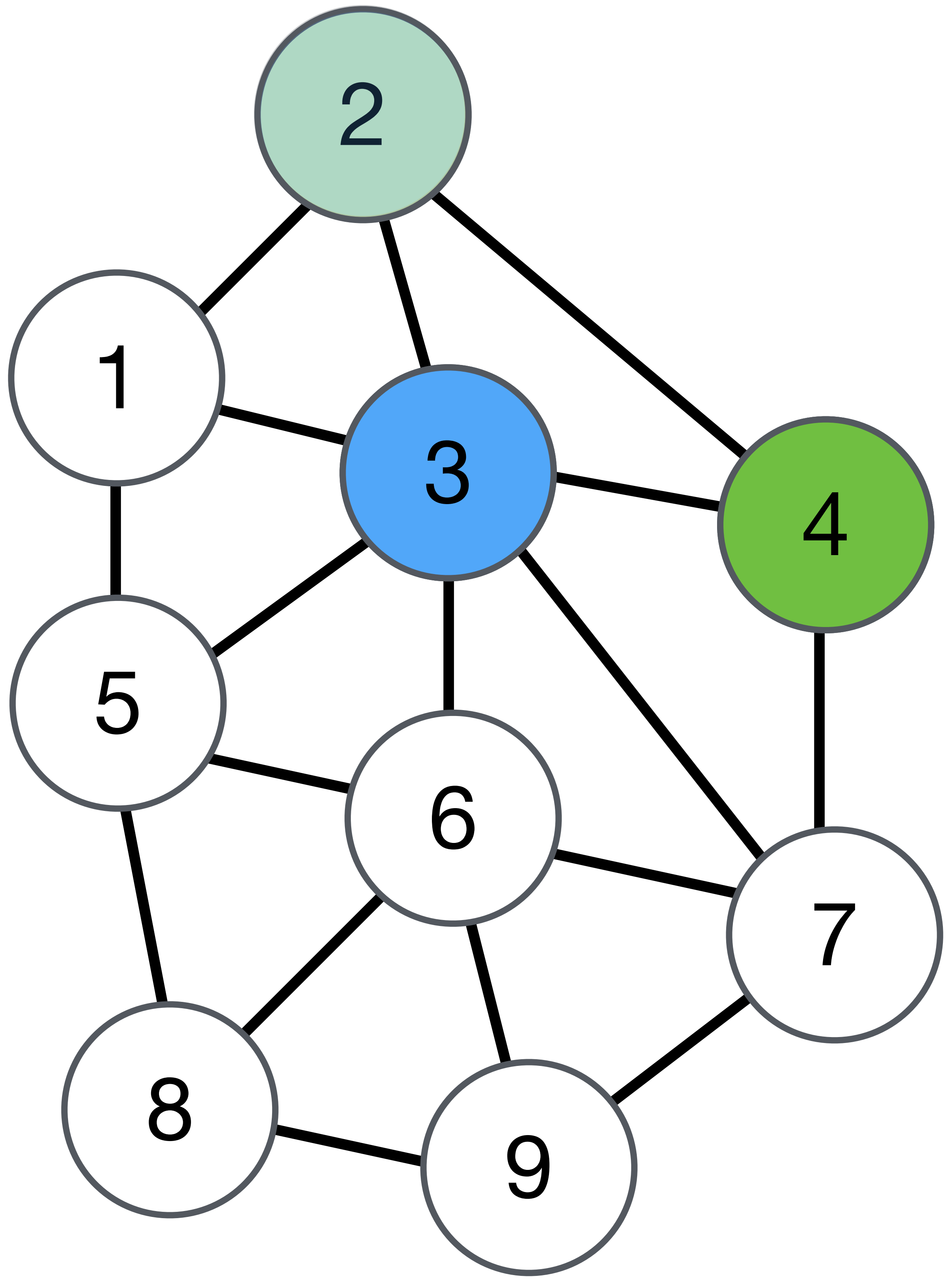


On relations

$$Q_{\triangle} = E(A, B) \bowtie E(B, C) \bowtie E(C, A)$$

Triangle counting on graphs, relations, and tensors

On graphs



On relations

$$Q_{\triangle} = E(A, B) \bowtie E(B, C) \bowtie E(C, A)$$

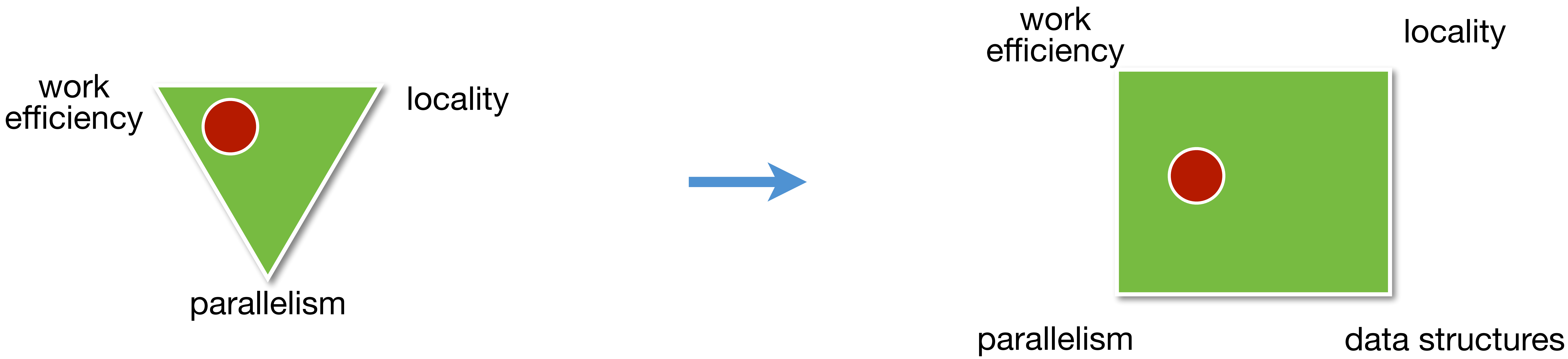
On tensors

$$\frac{1}{6}\text{trace}(A^3).$$

Some important developments in compilers and programming languages for sparse compilers

- 1960s: Development of libraries for sparse linear algebra
- 1970s: Relational algebra and the first relational database management systems: System R and INGRES
- 1980s: SQL is developed and has commercial success
- 1990s: Matlab gets sparse matrices and some dense to sparse linear algebra compilers are developed
- 2000s: Sparse linear algebra libraries for supercomputers and GPUs
- 2010s: Graph processing libraries become popular, compilers for databases, and compilers for tensor algebra

Parallelism, locality, work efficiency still matters, but the key is choosing efficient data structures



Harry	CS
Sally	EE
George	CS
Mary	ME
Rita	CS

Harry	Sally	George	Mary	Rita
CS	EE	CS	ME	CS

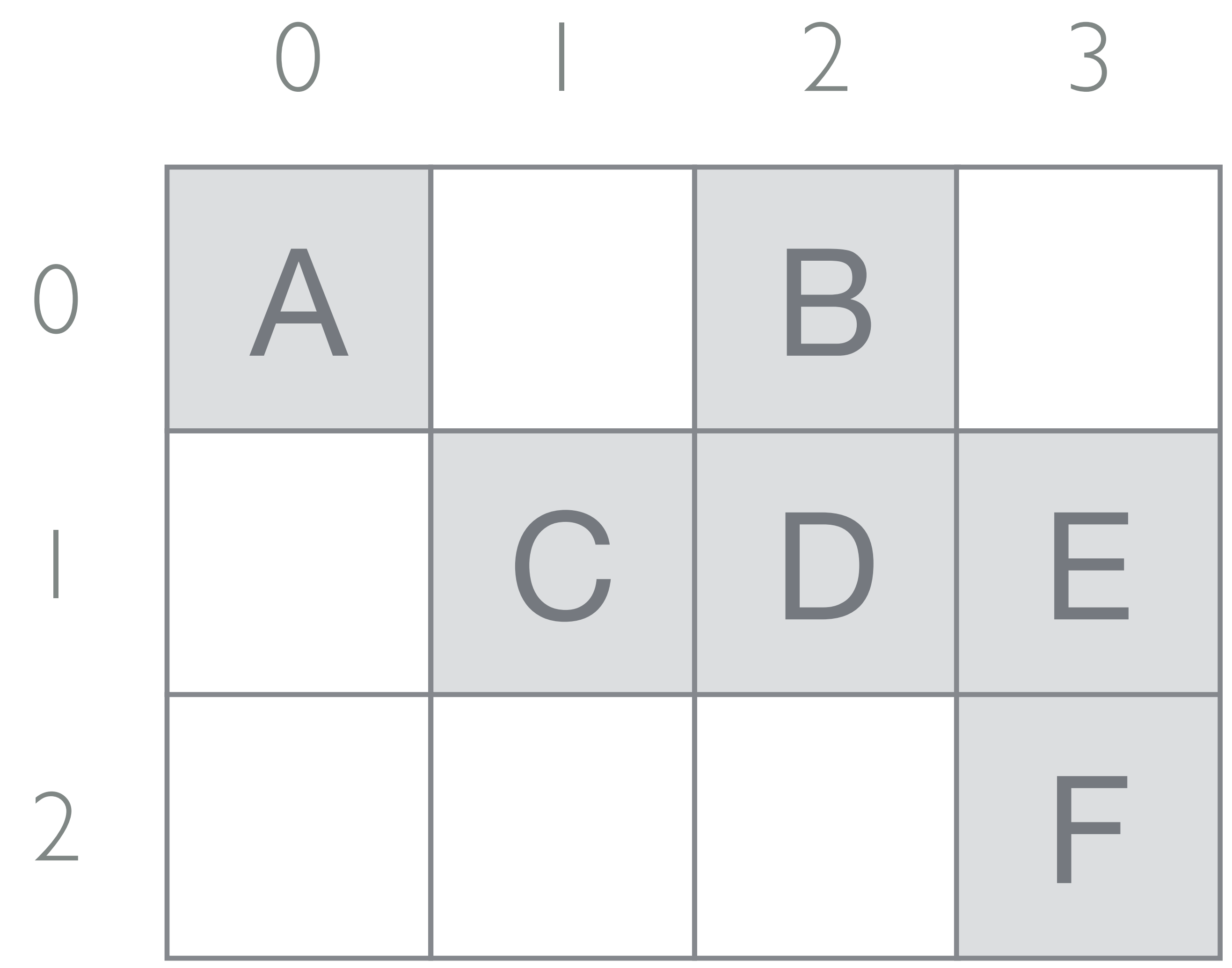
0	2						
0	2						
0	1	3					
0	0	1					
0	3	5	8				
0	2	3	0	2	1	2	3
30	40	50	10	70	80	20	60

Sparse data structures in graphs, tensors, and relations encode coordinates in a sparse iteration space

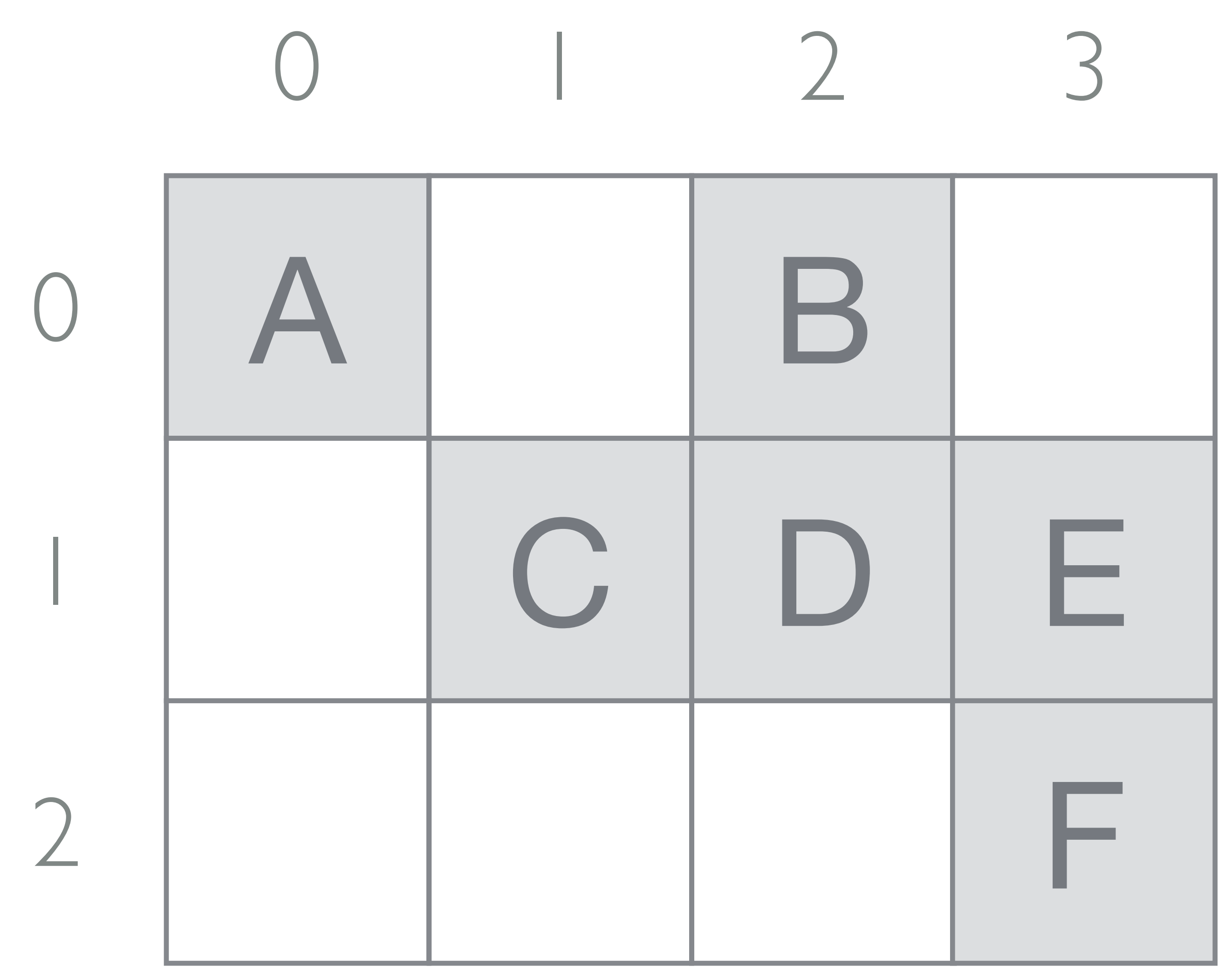
Tensor (nonzeros)	Relation (rows)	Graph (edges)
(0,1)	(Harry,CS)	(v_1,v_5)
(2,3)	(Sally,EE)	(v_4,v_3)
(0,5)	(George,CS)	(v_5,v_3)
(5,5)	(Mary,ME)	(v_3,v_1)
(7,5)	(Rita,CS)	(v_3,v_5)

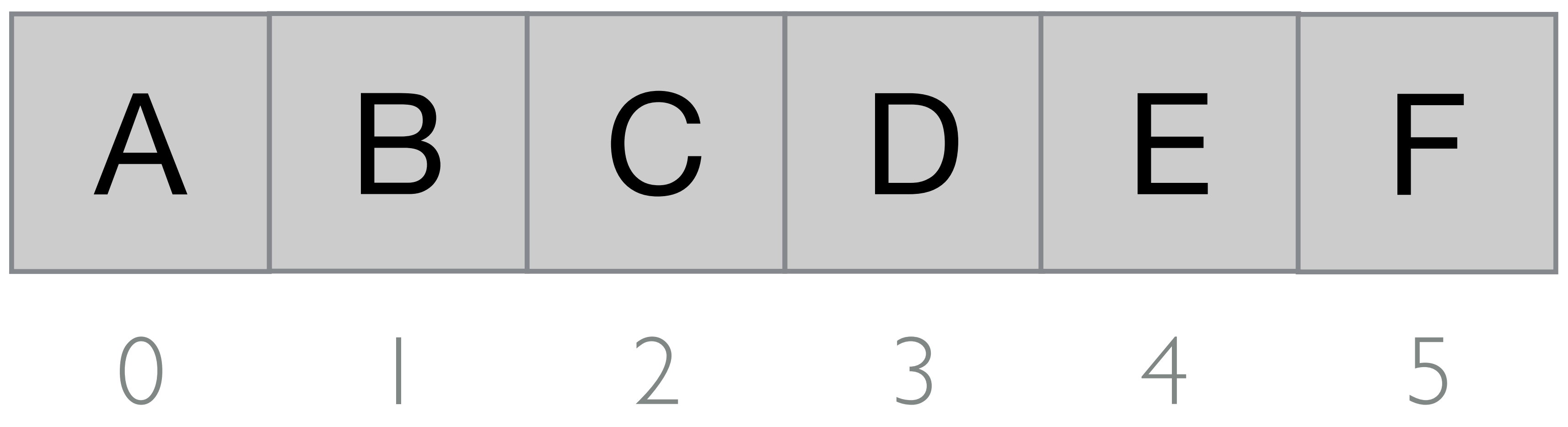
Values may be attached to these coordinates: e.g., nonzero values, edge attributes

Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



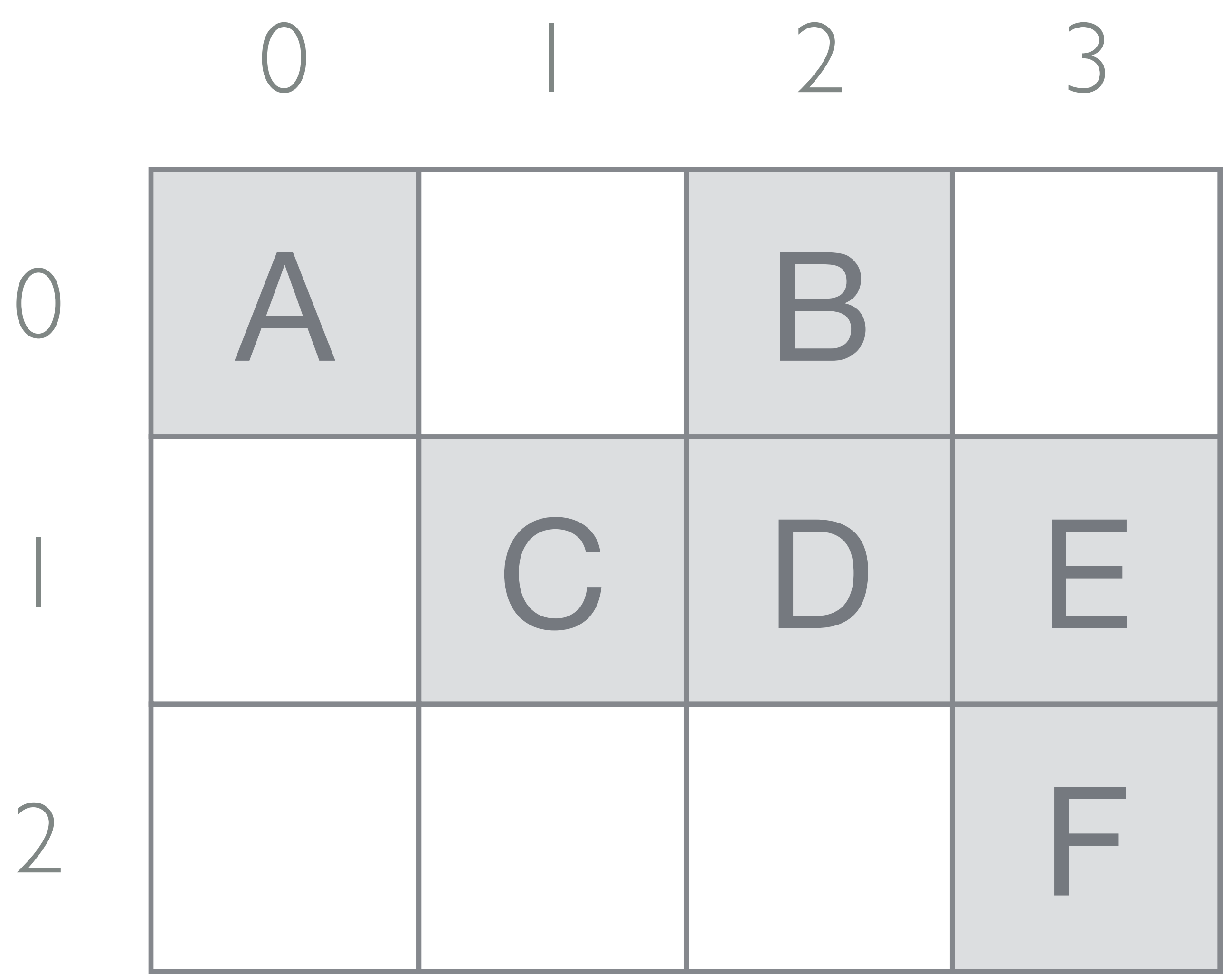
Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



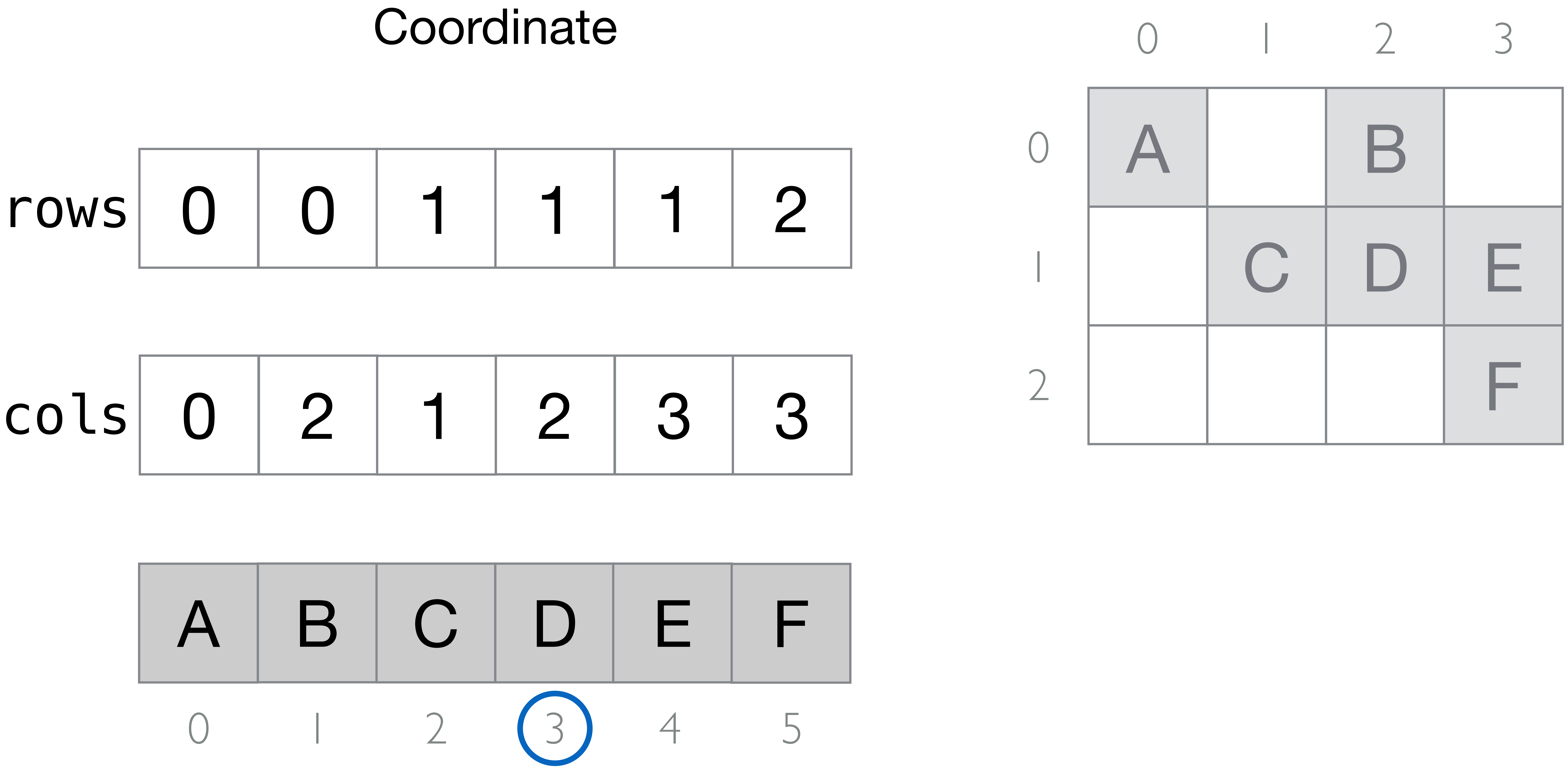


Hierarchically compressed data structures (tries) reduce the number of values that need to be stored

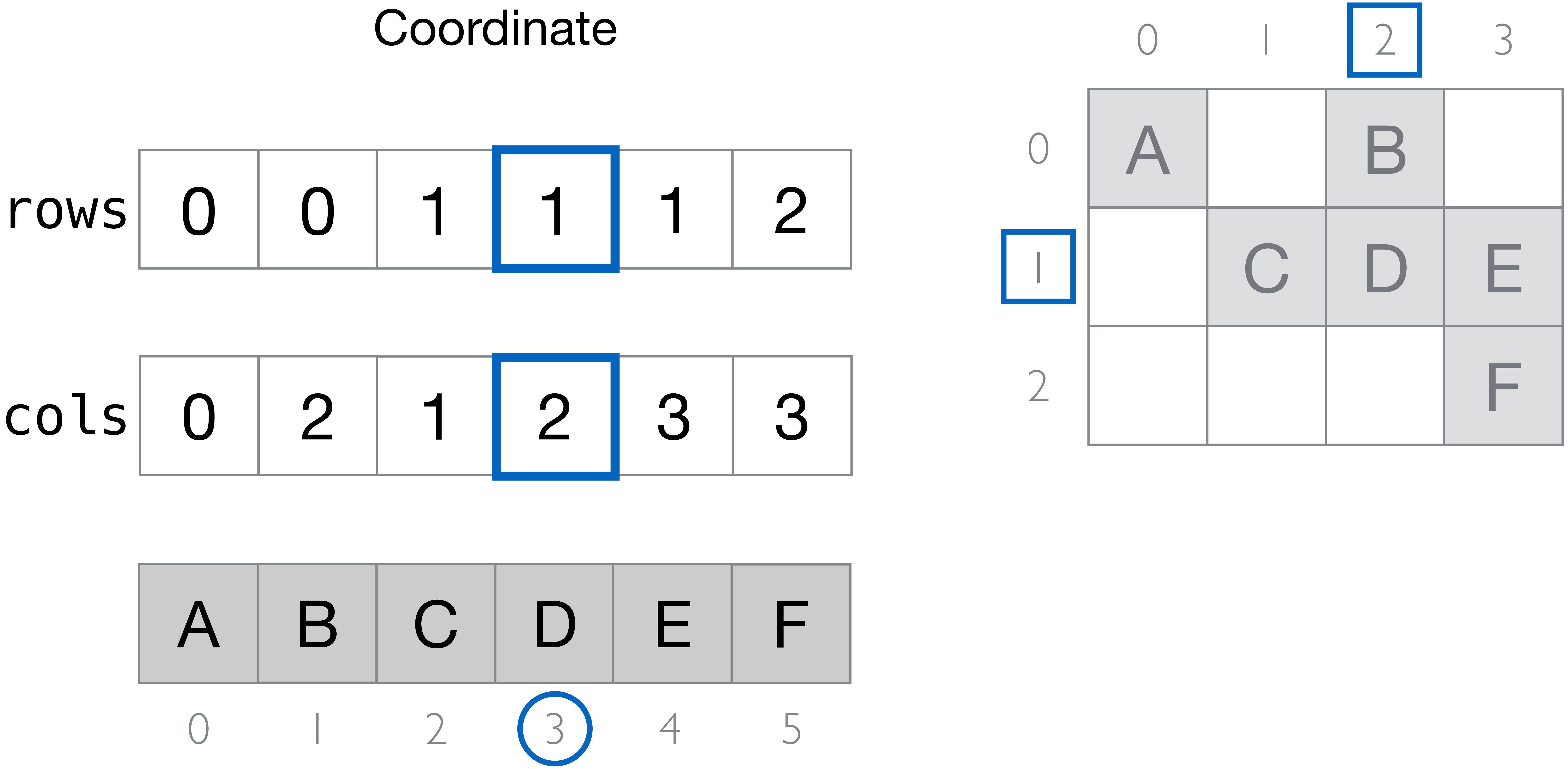
```
row(3) = ???
col(3) = ???
```



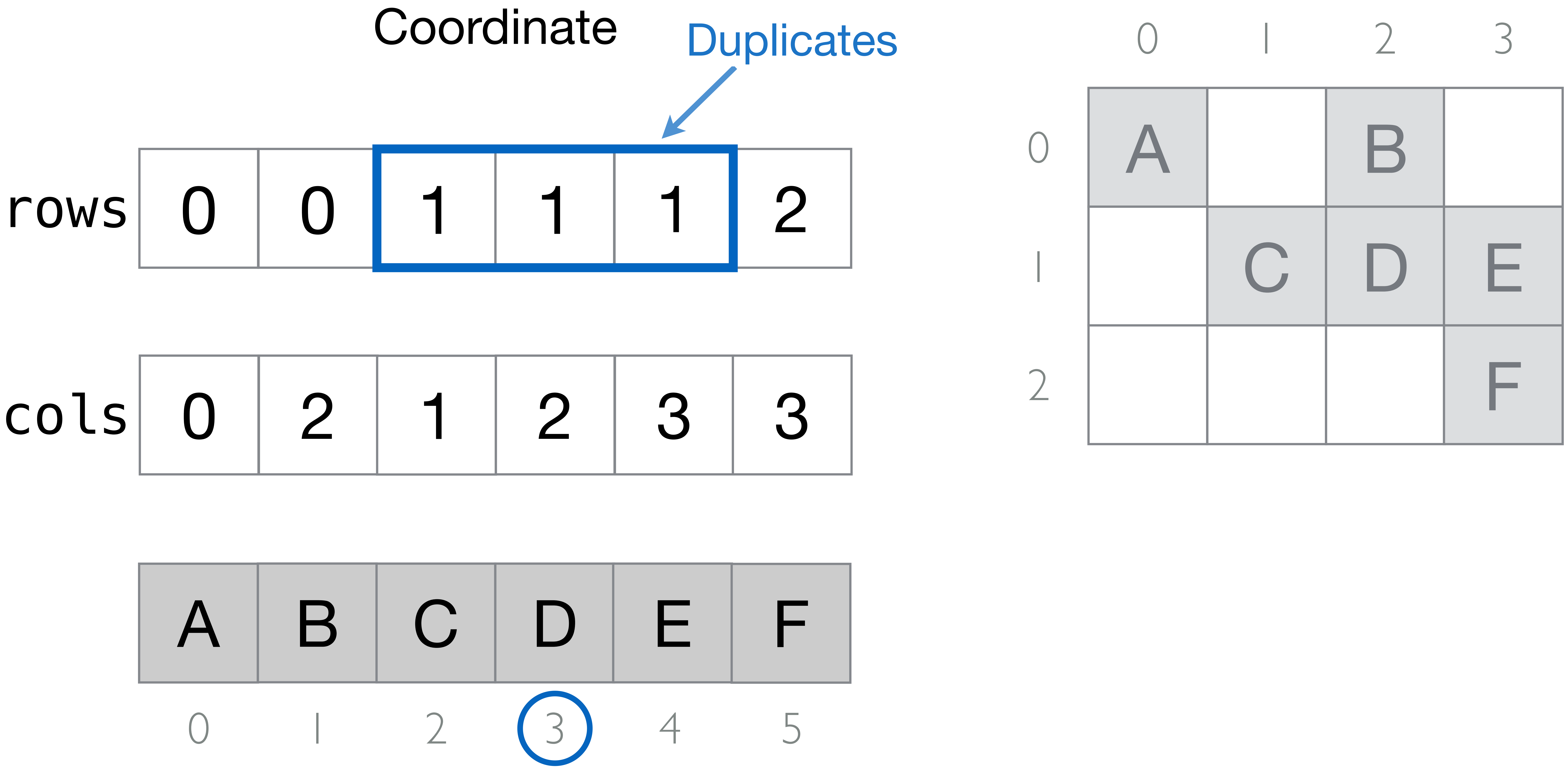
Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



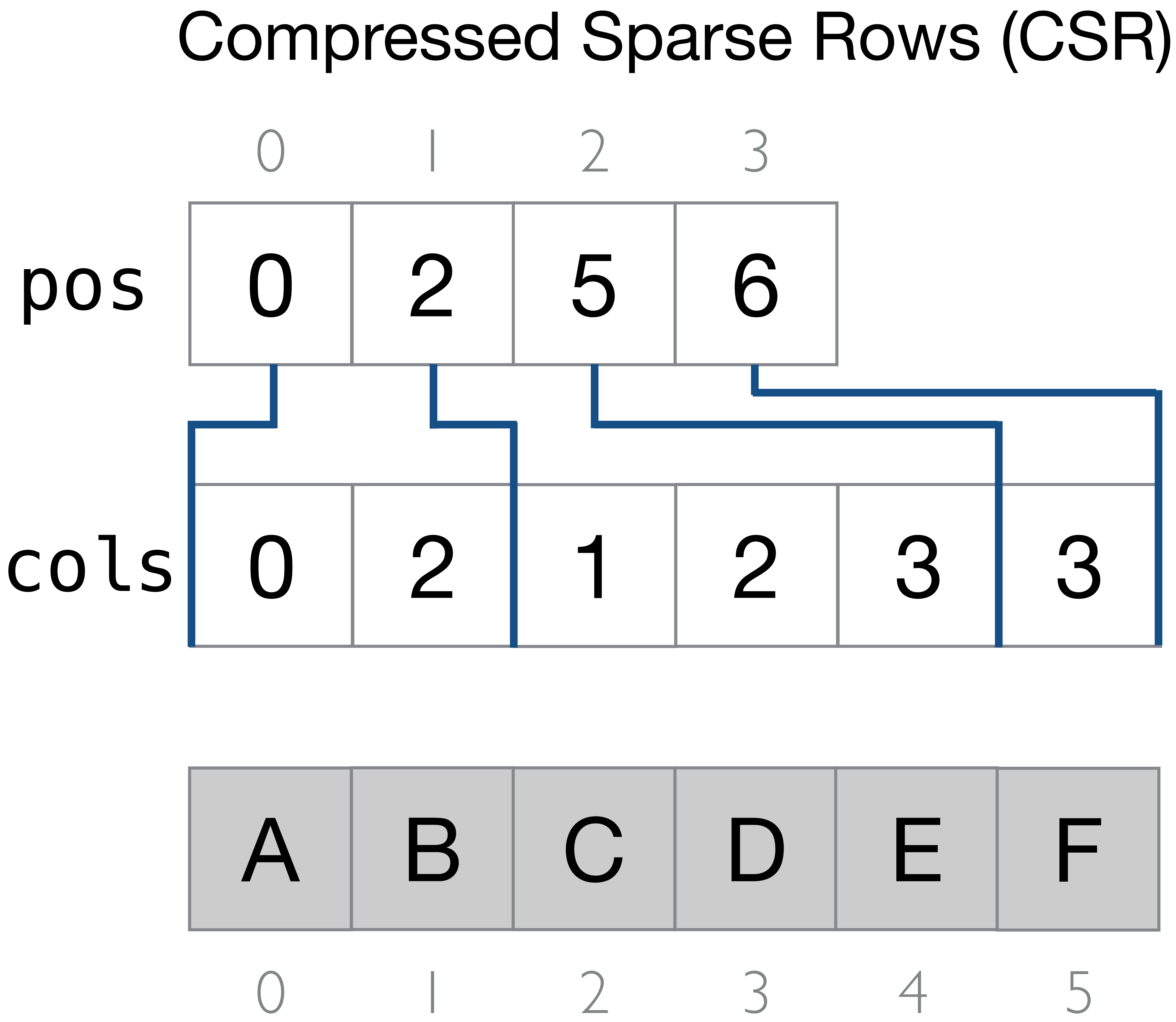
Hierarchically compressed data structures (tries) reduce the number of values that need to be stored

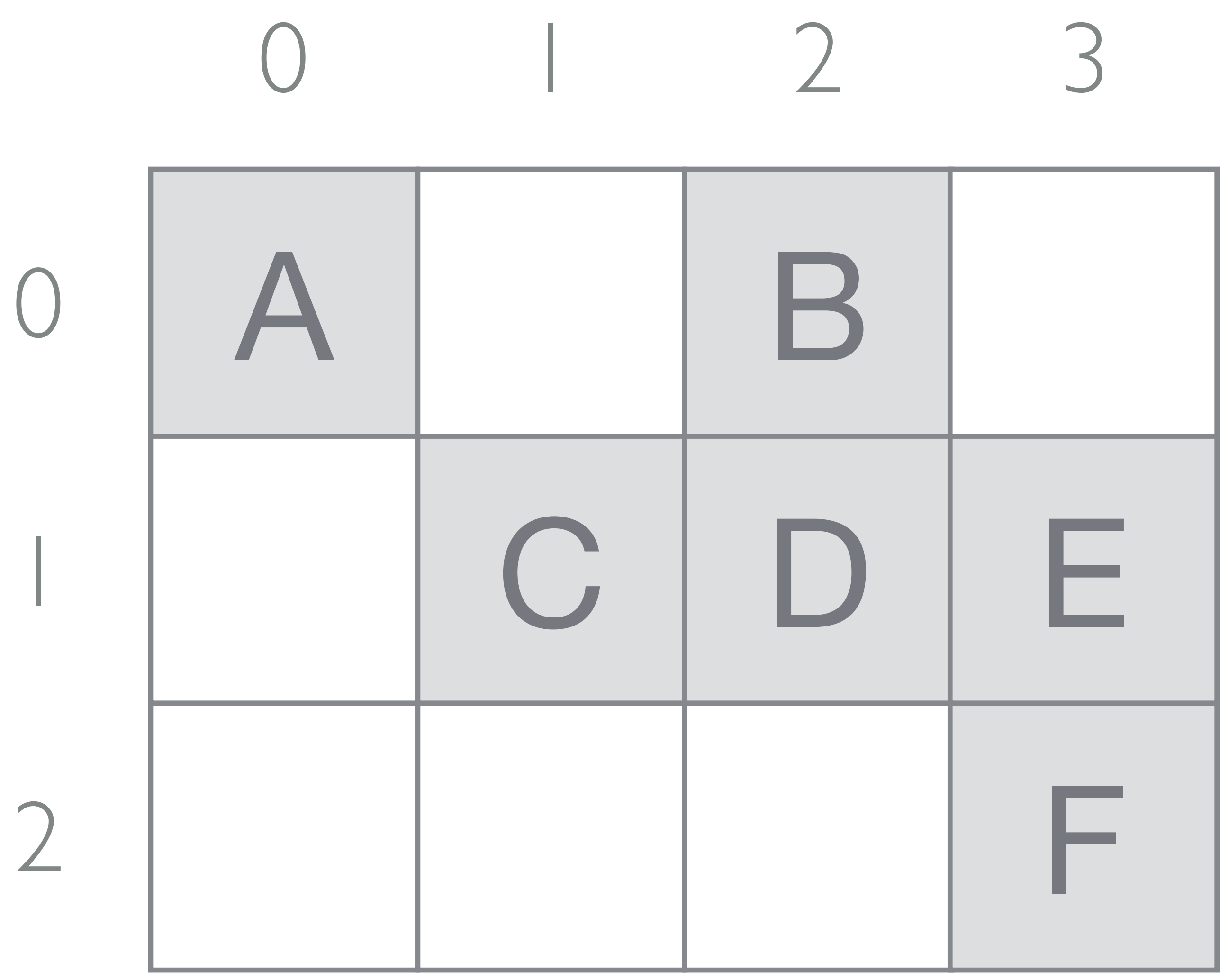


Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



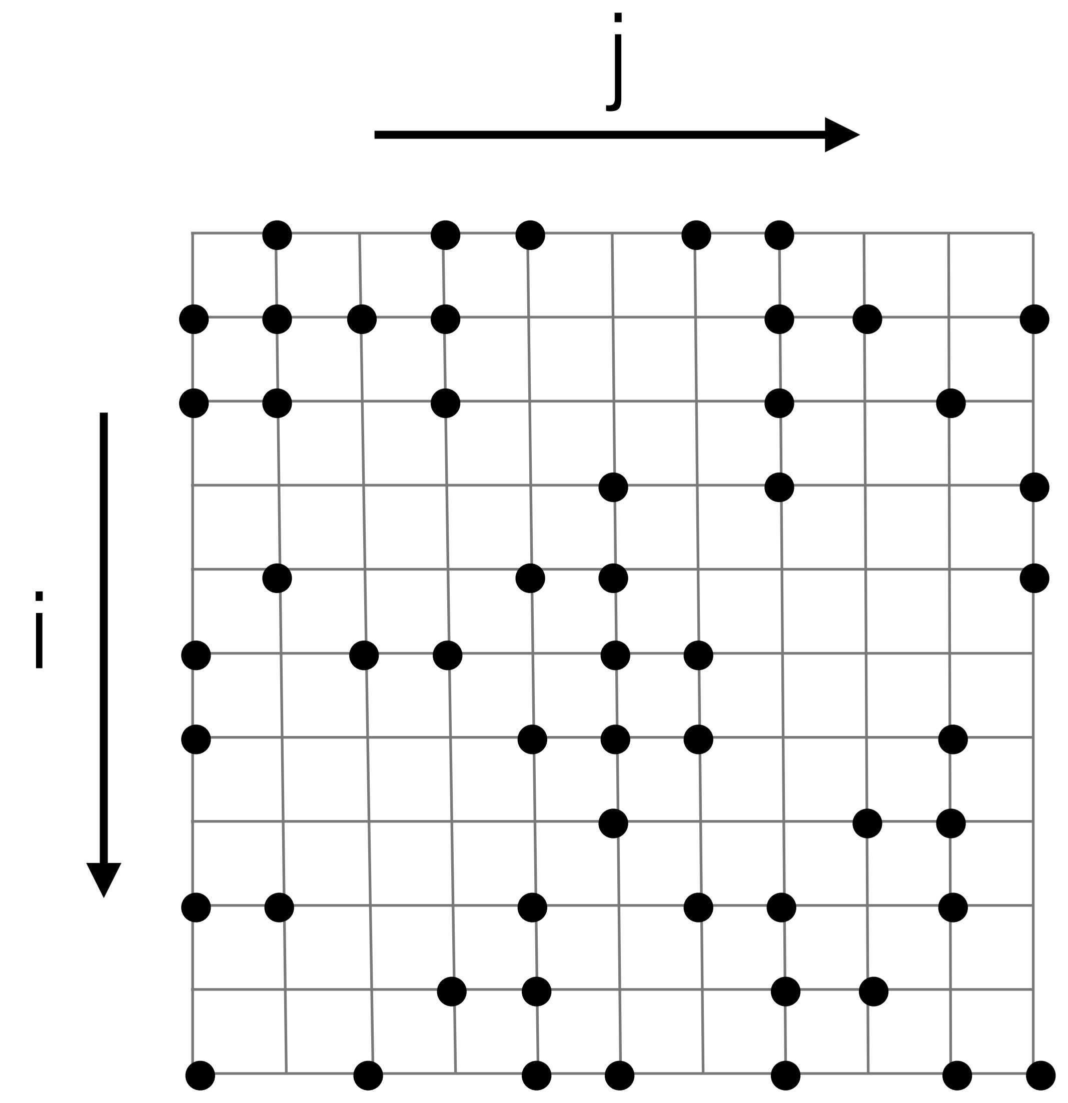


Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra: $A = B + C$

Tensor Index Notation: $A_{ij} = B_{ij} + C_{ij}$

Iteration Space: $B_{ij} \cup C_{ij}$



union because $x + 0 = x$

B

pos	0	3	5	8				
cols	0	2	3	0	2	1	2	3
vals	30	40	50	10	70	80	20	60

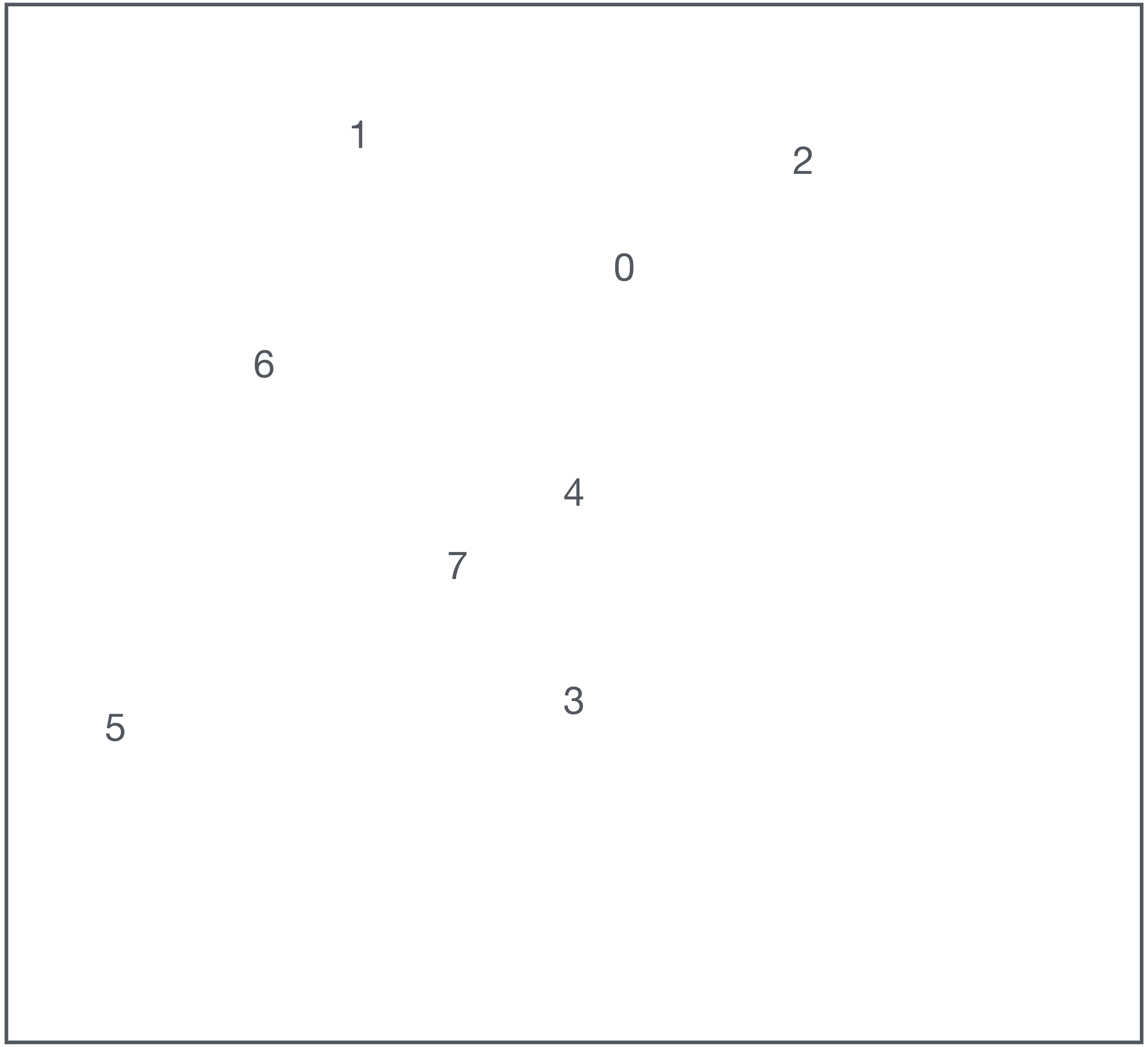
$\cup$

C

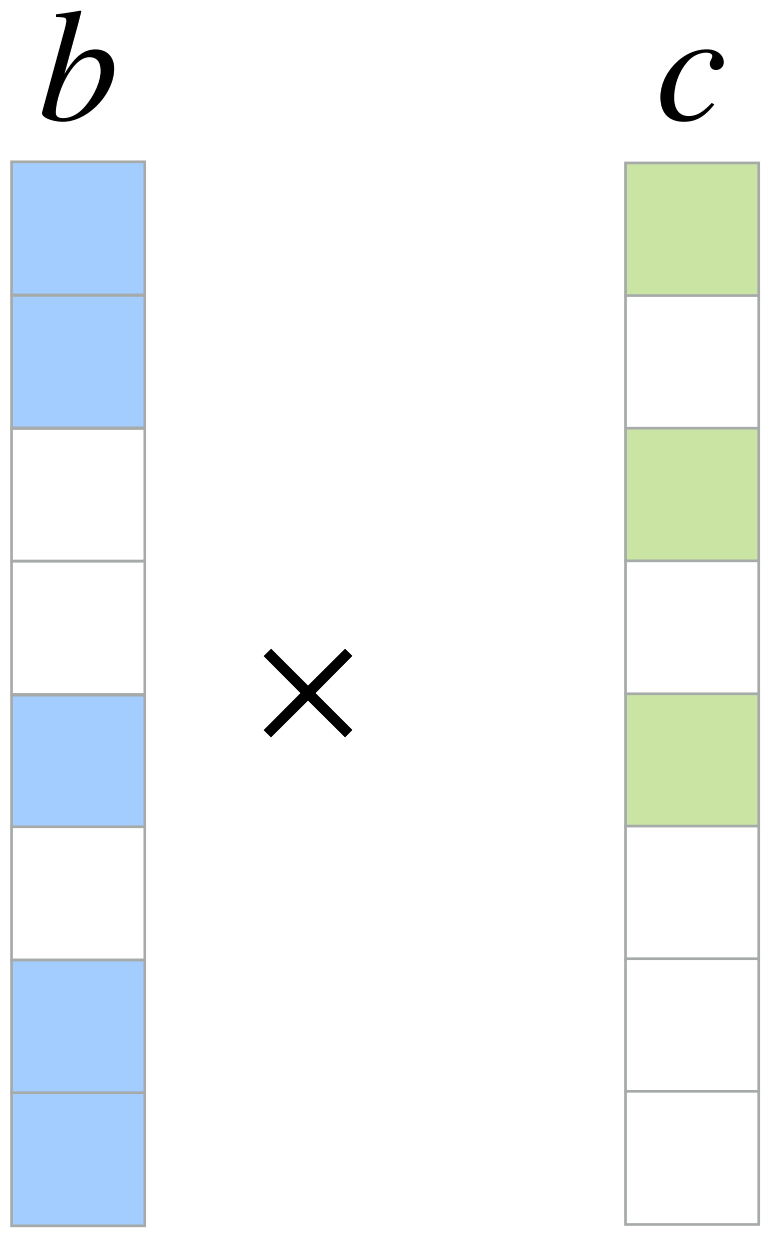
pos	0	1	3
cols	0	2	3
vals	2	1	9

coiterate

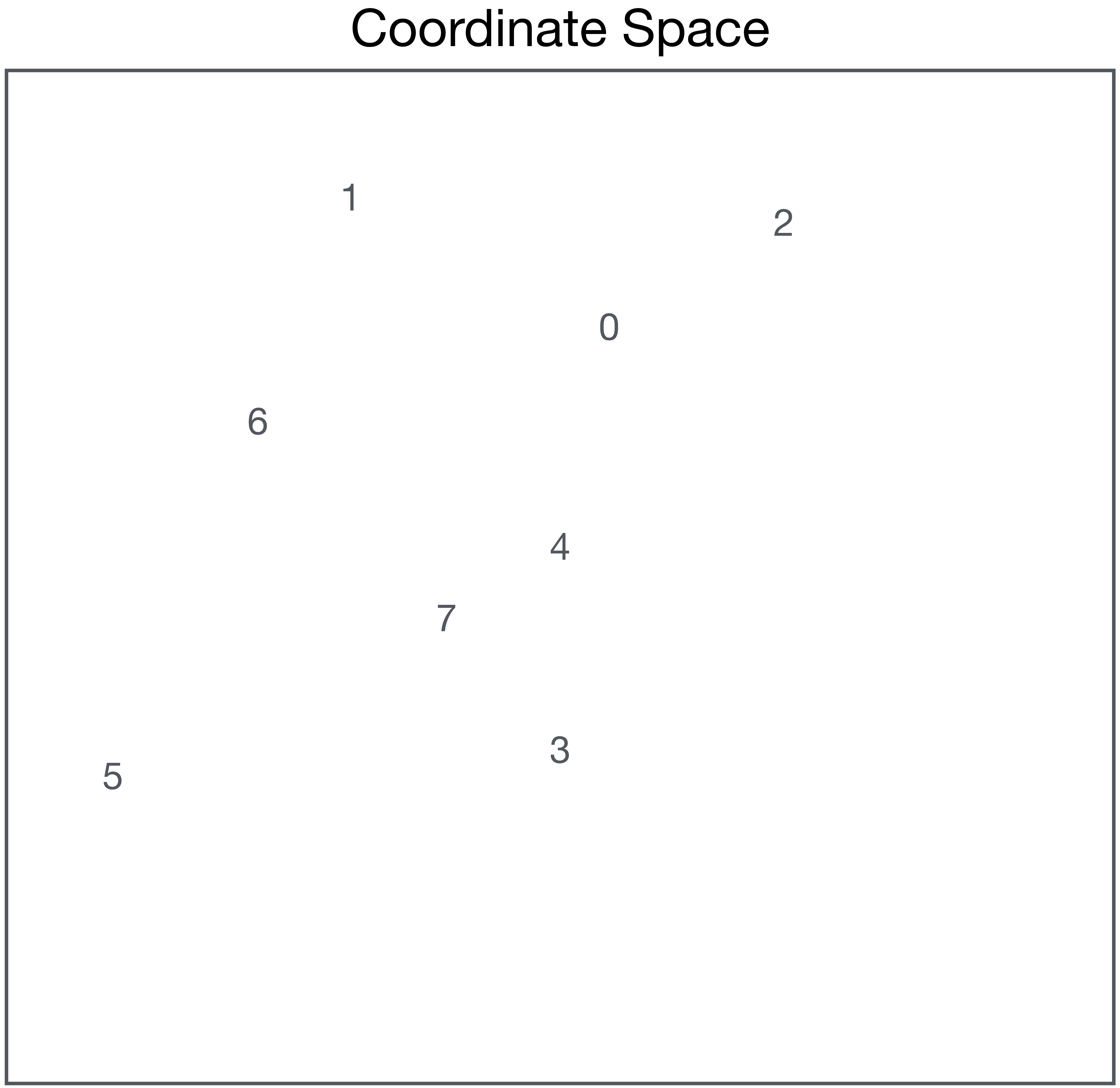
Merged coiteration



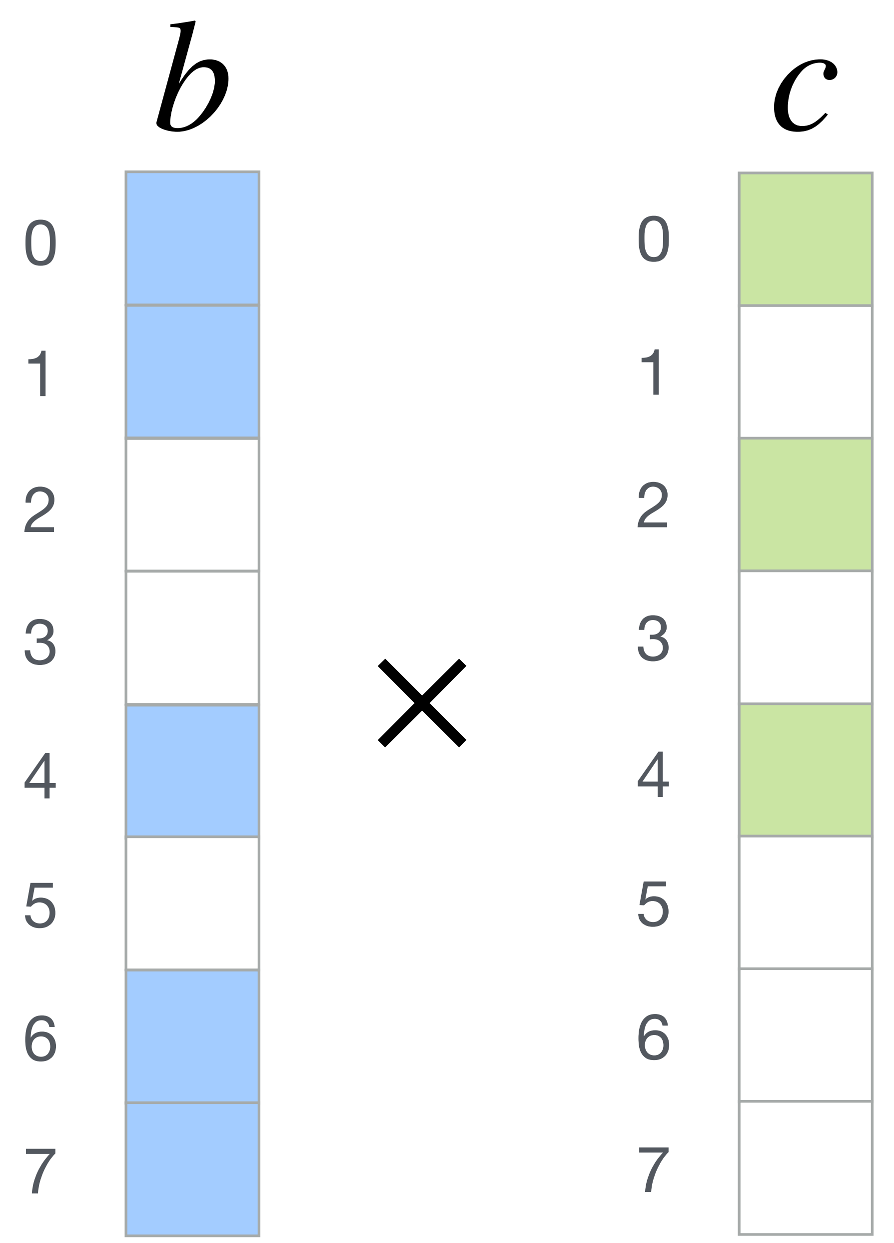
$$a_i = b_i c_i$$



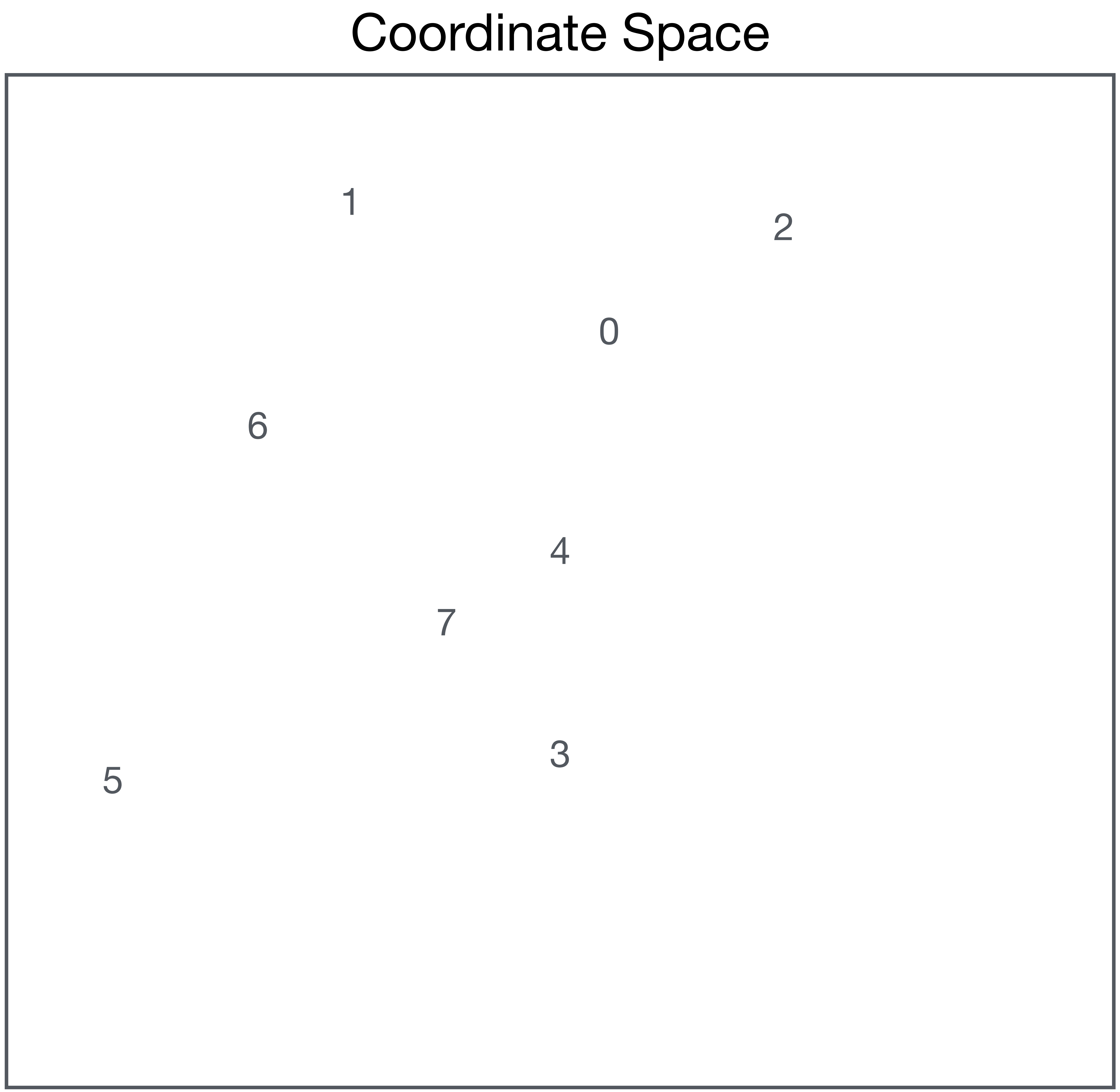
Merged coiteration



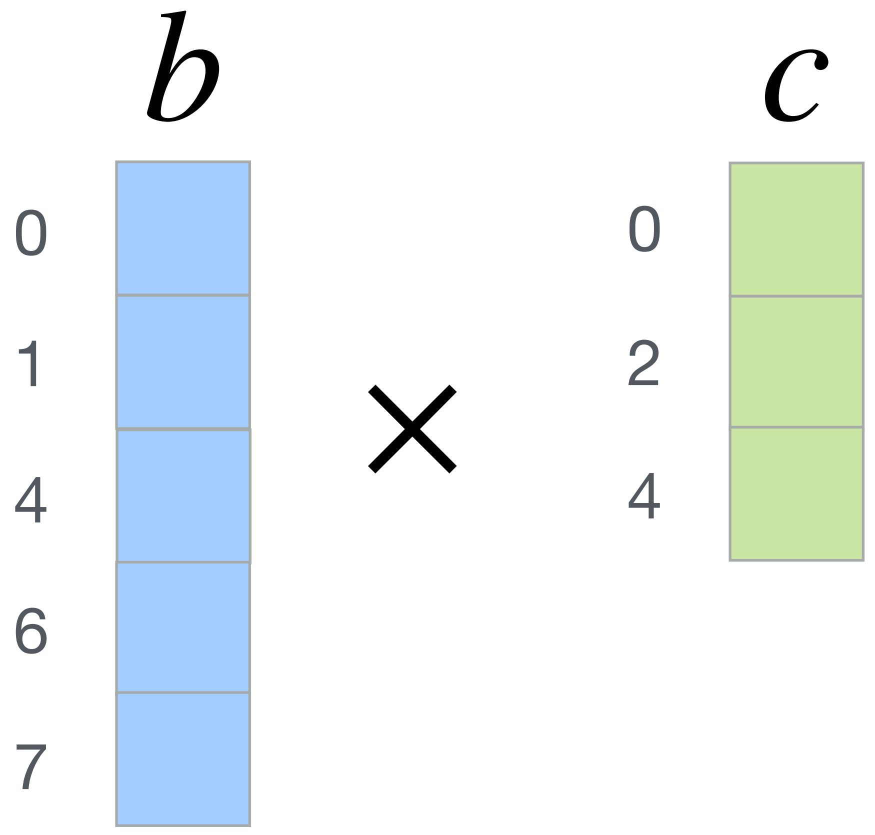
$$a_i = b_i c_i$$



Merged coiteration

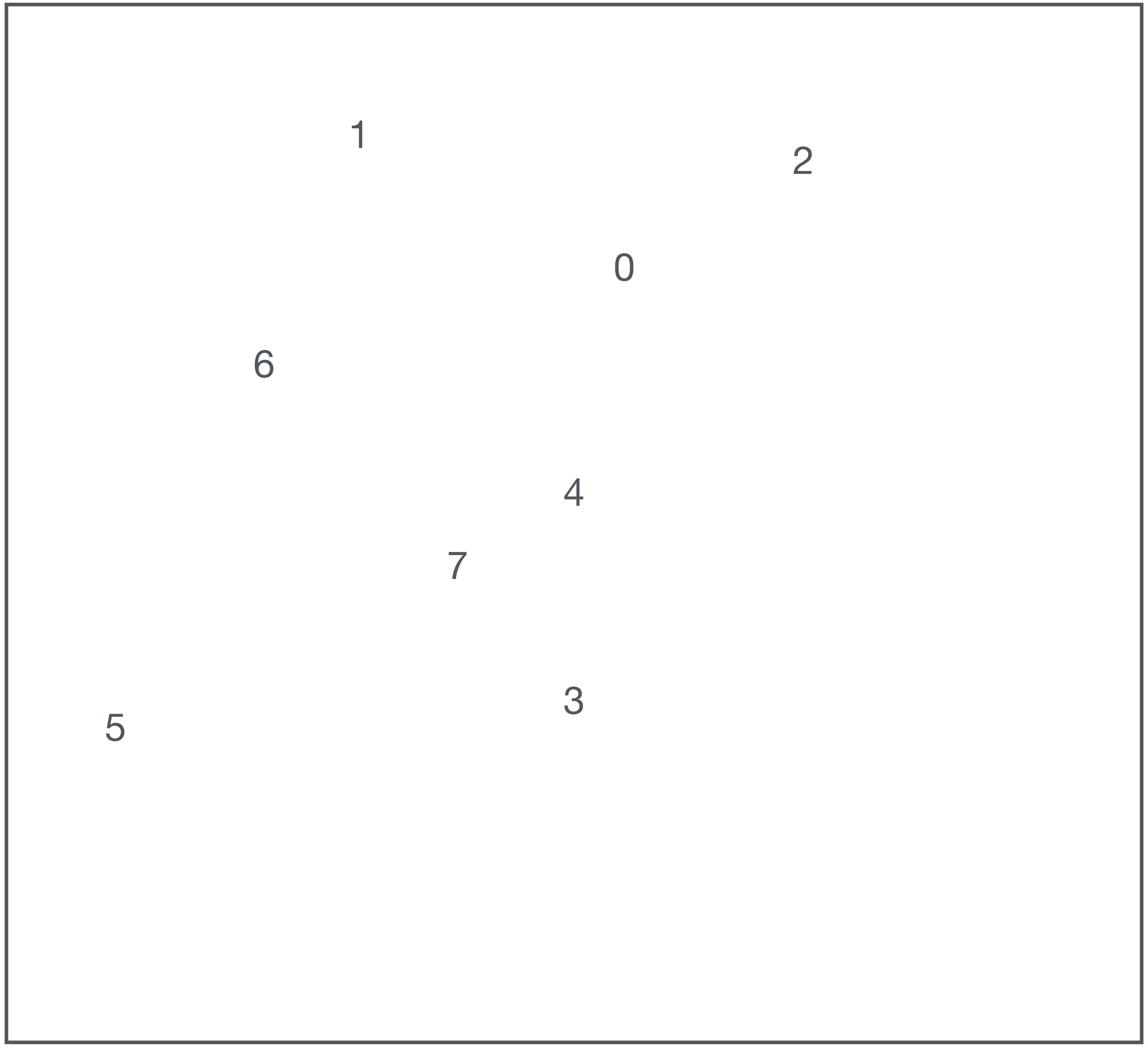


$$a_i = b_i c_i$$

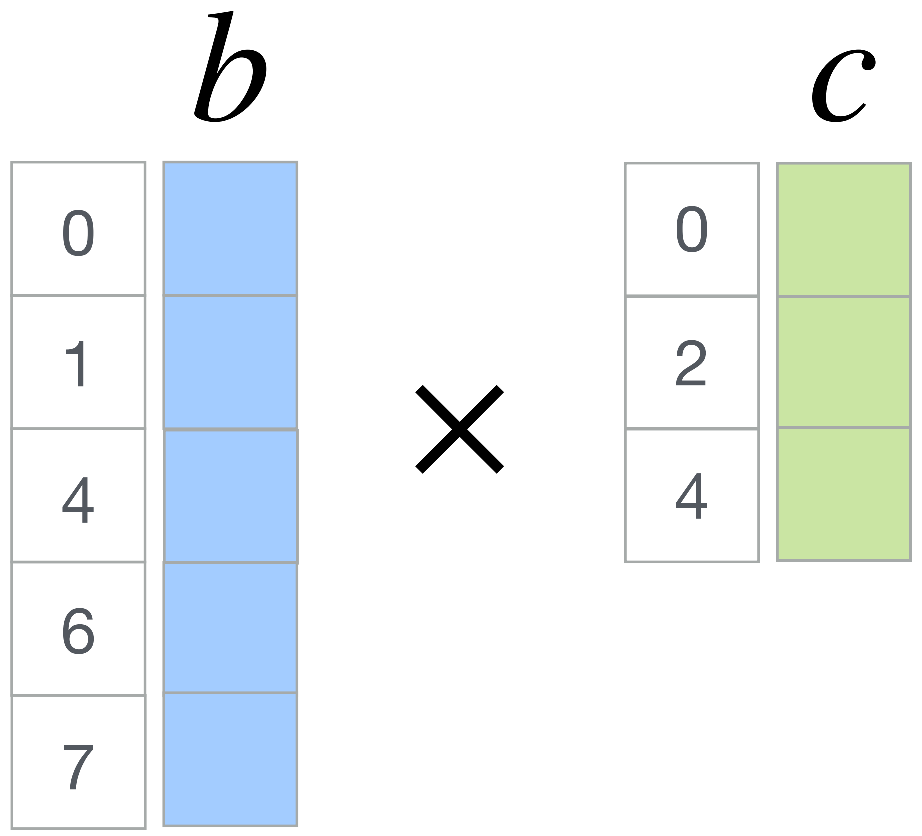


Merged coiteration

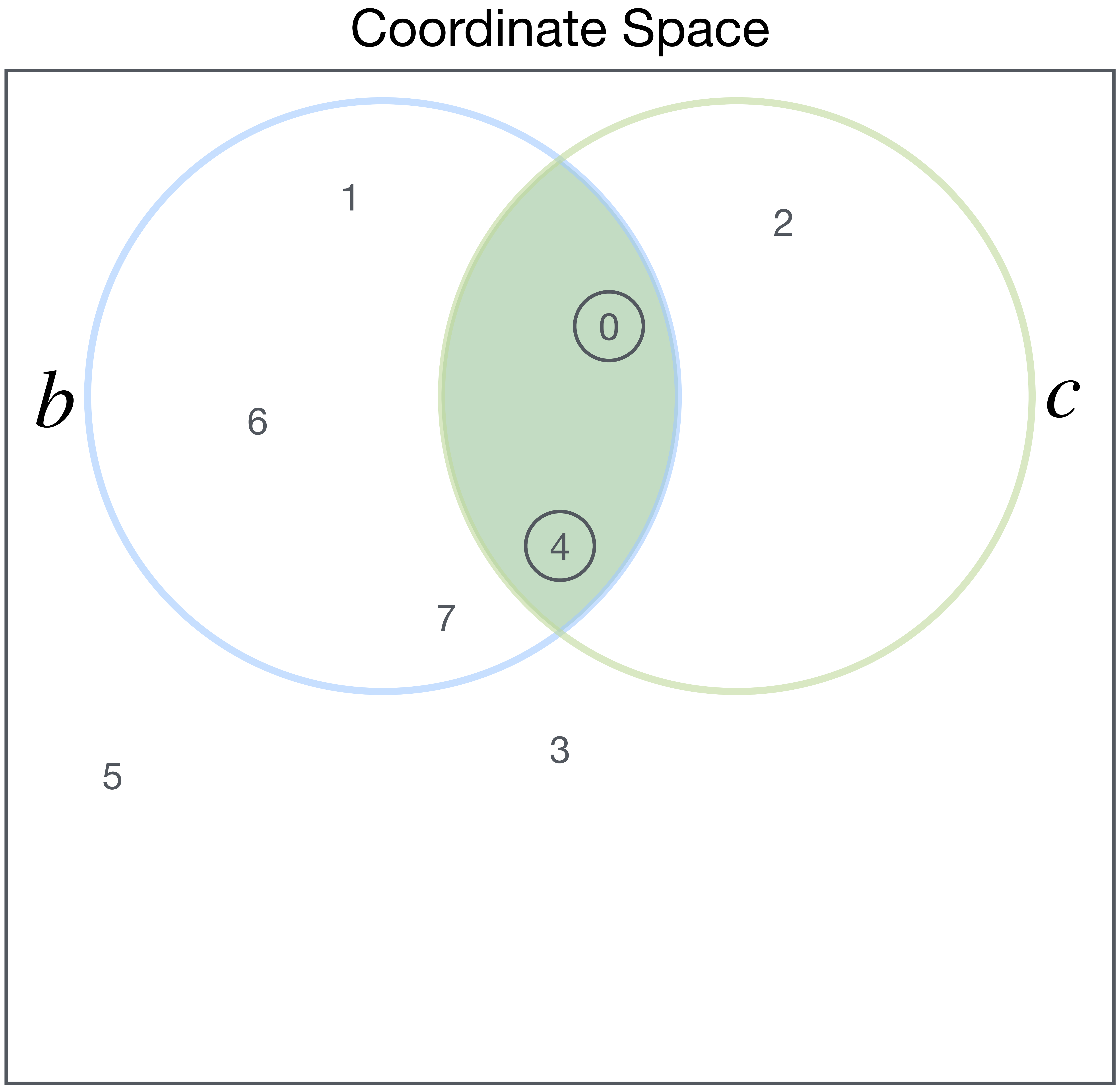
Coordinate Space



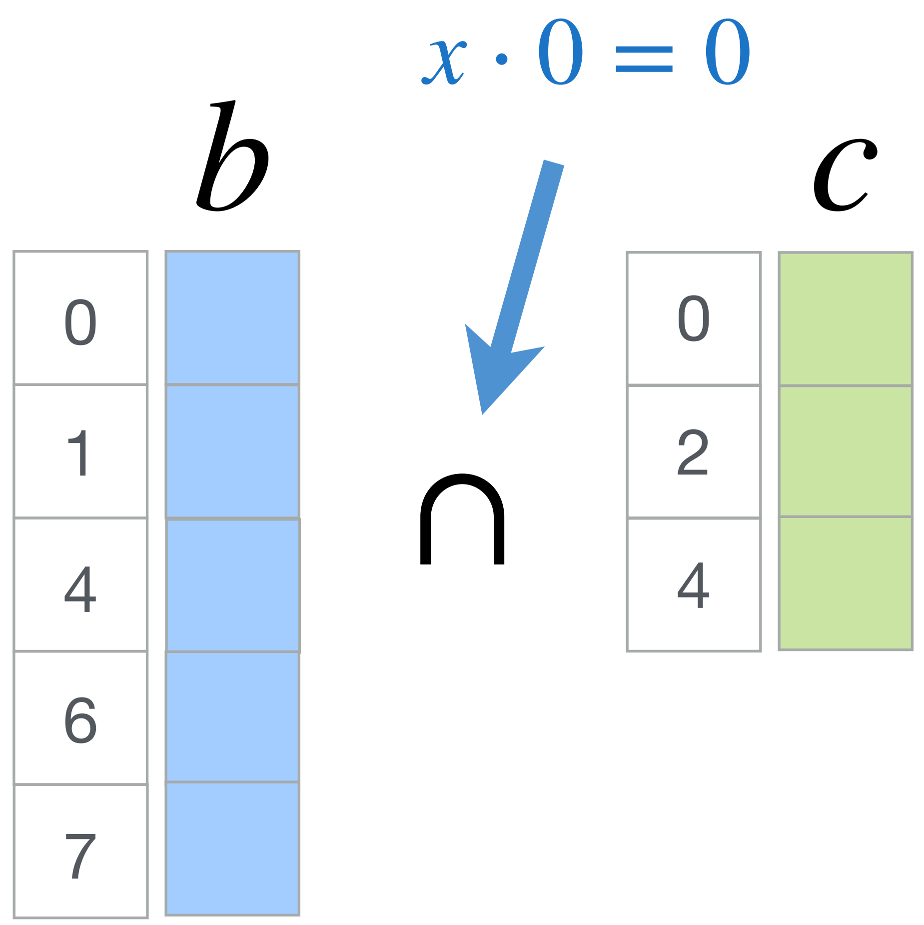
$$a_i = b_i c_i$$



Merged coiteration

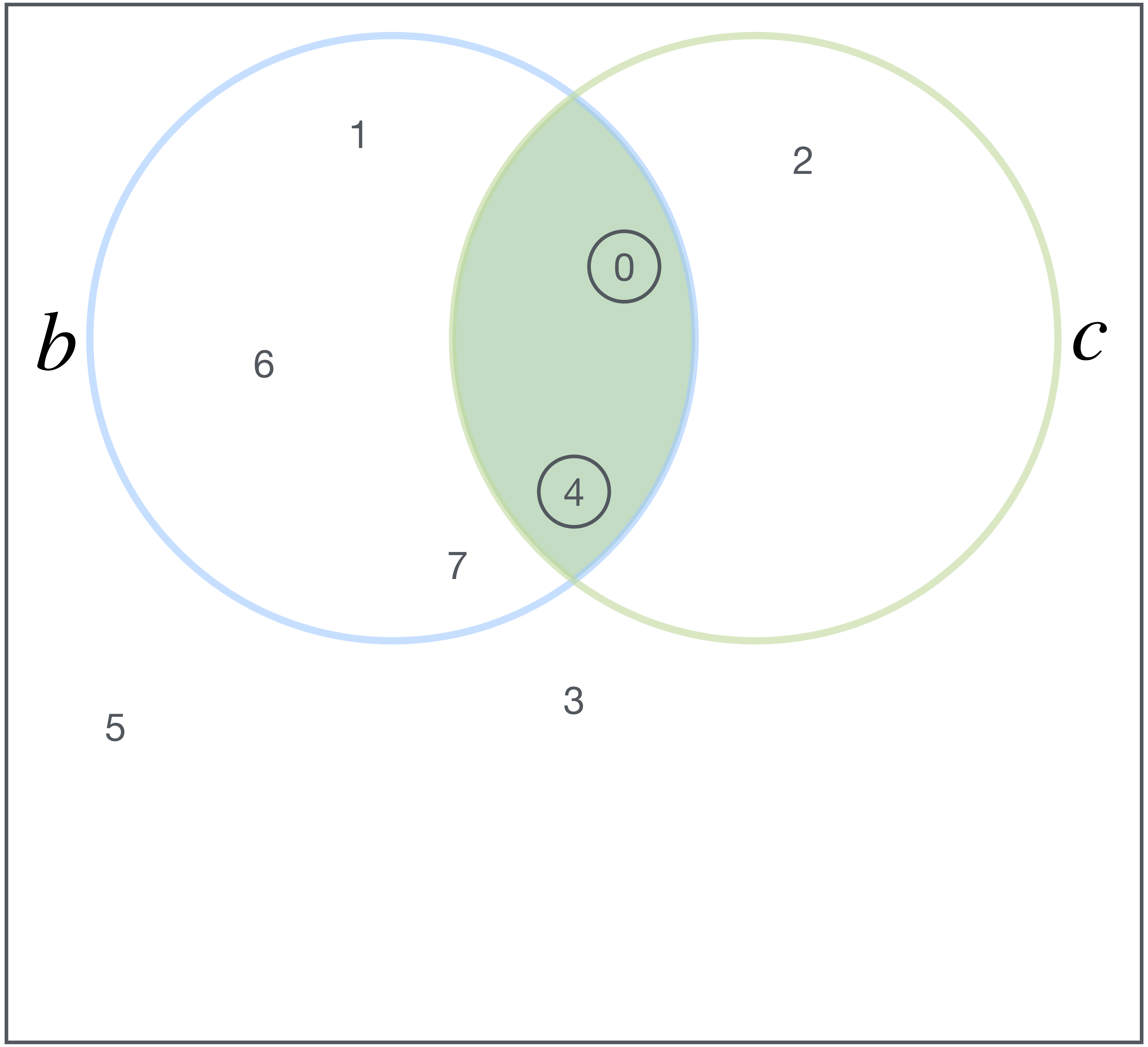


$$a_i = b_i c_i$$

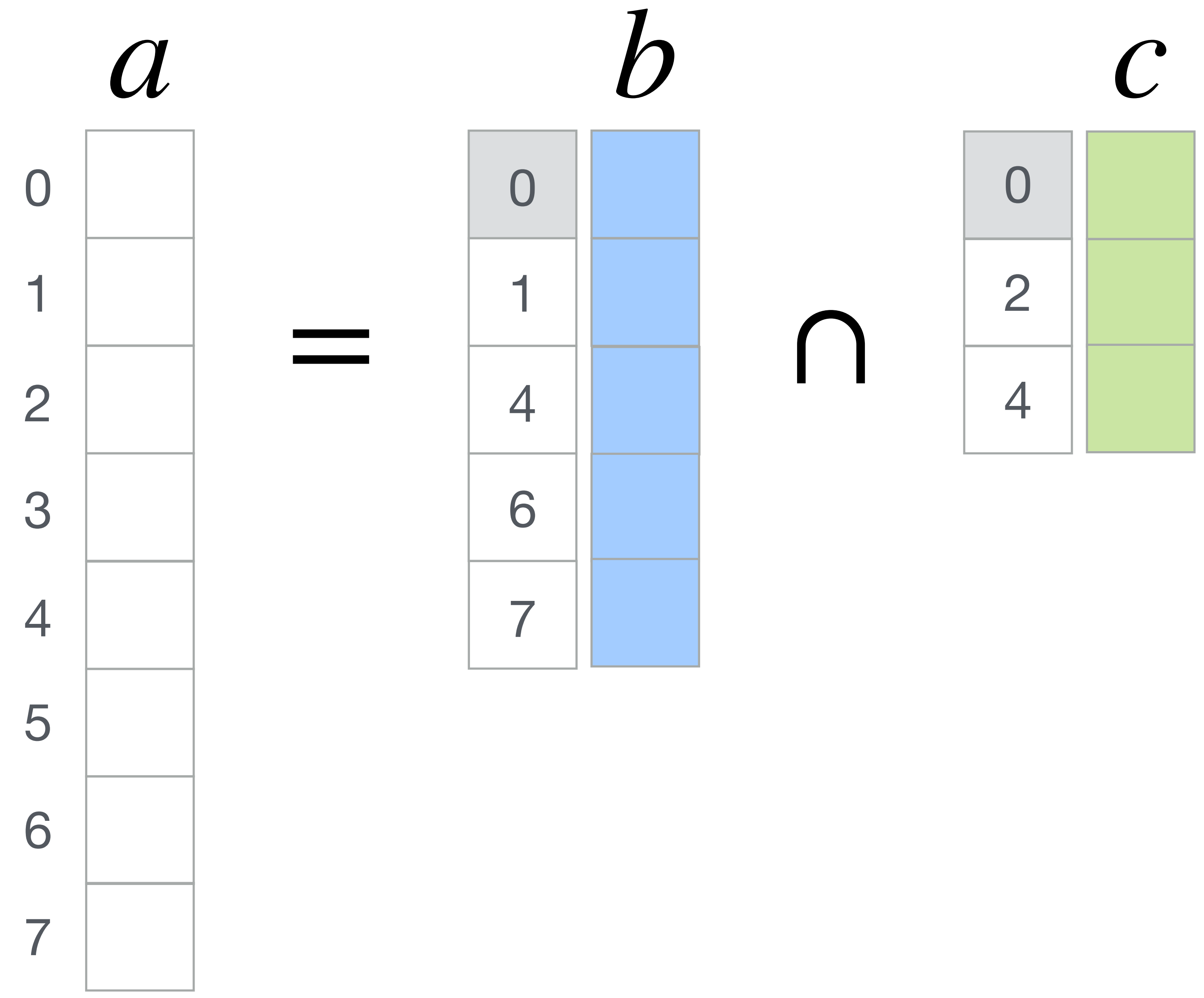


Merged coiteration

Coordinate Space

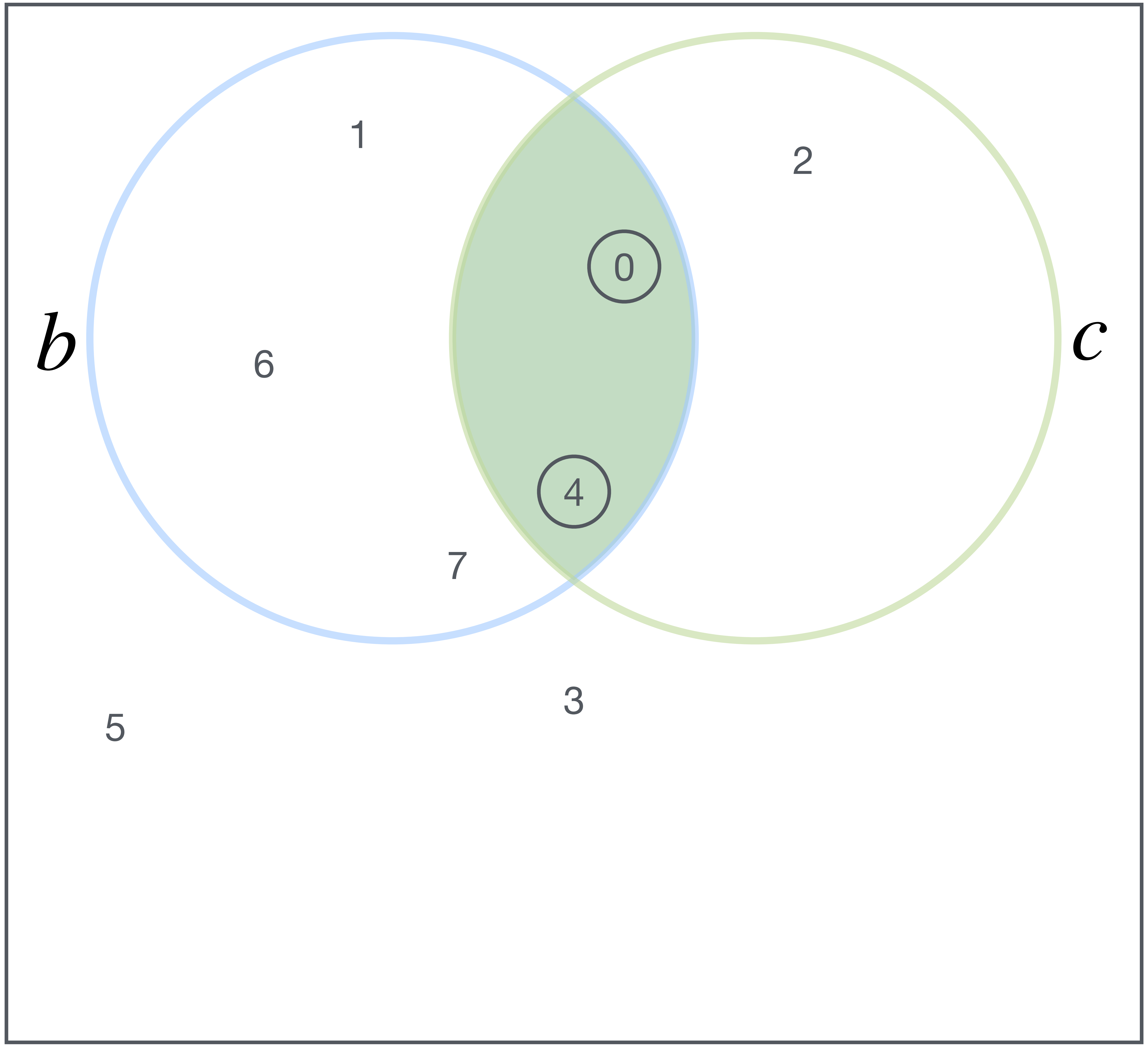


$$a_i = b_i c_i$$

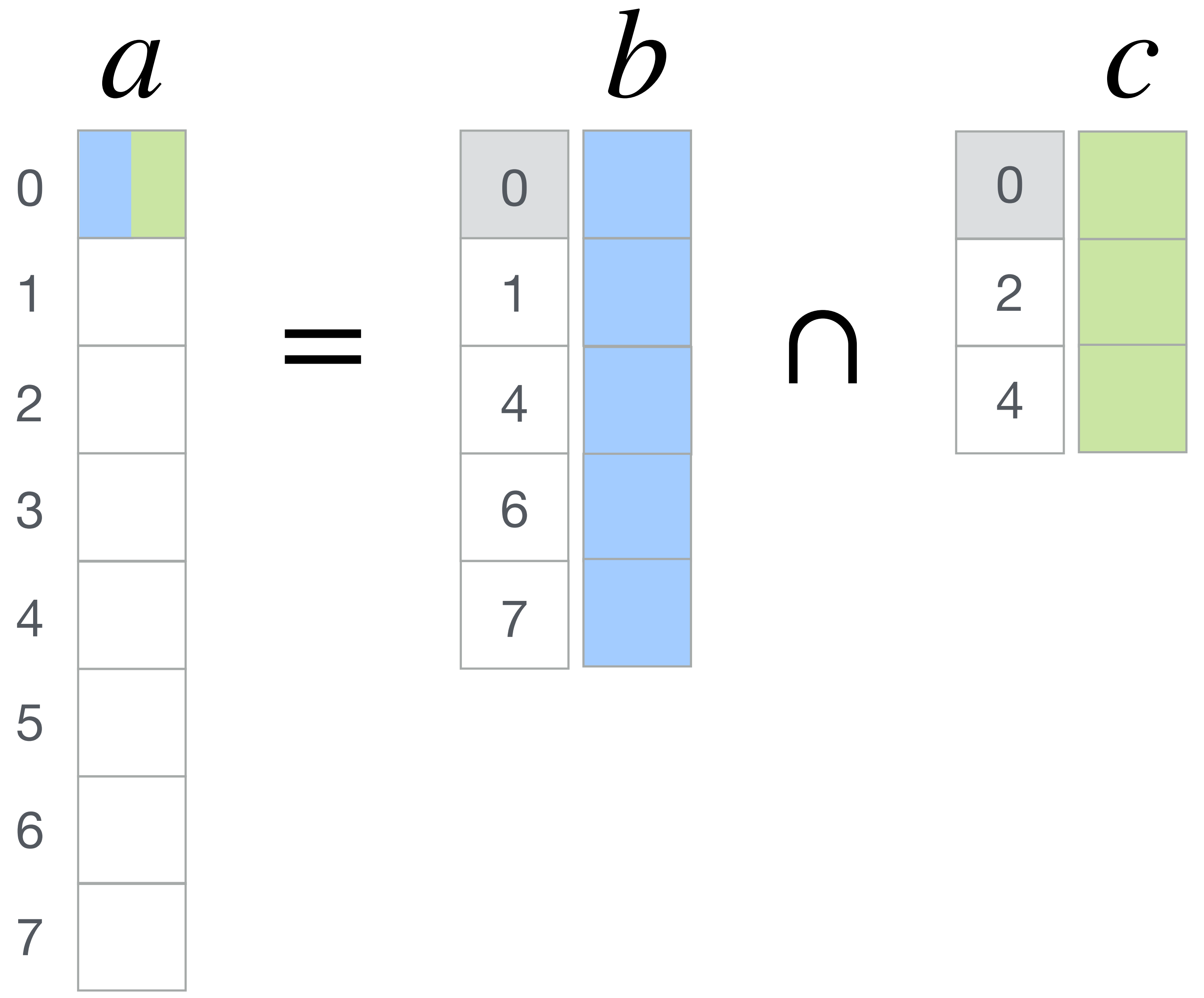


Merged coiteration

Coordinate Space



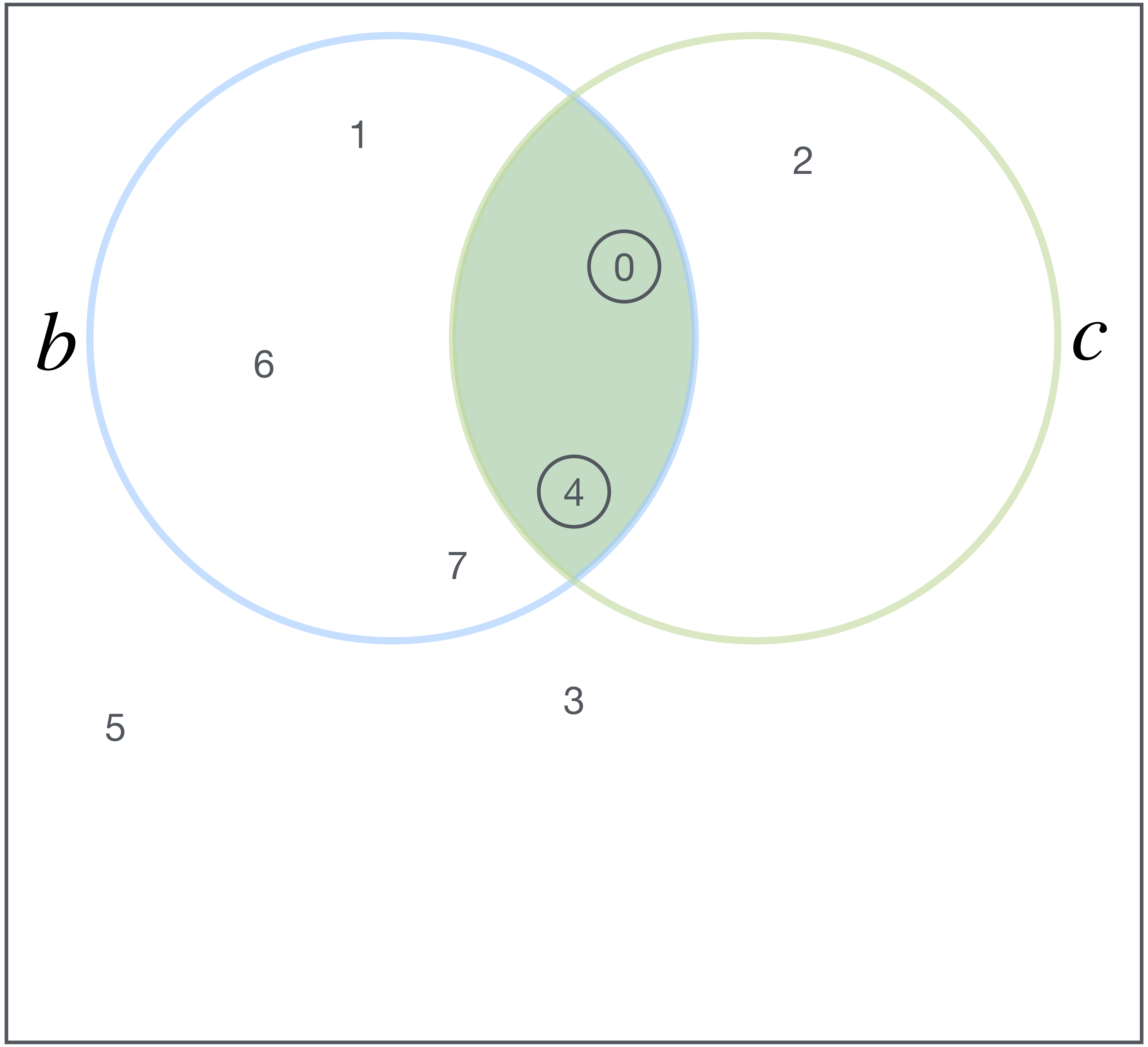
$$a_i = b_i c_i$$

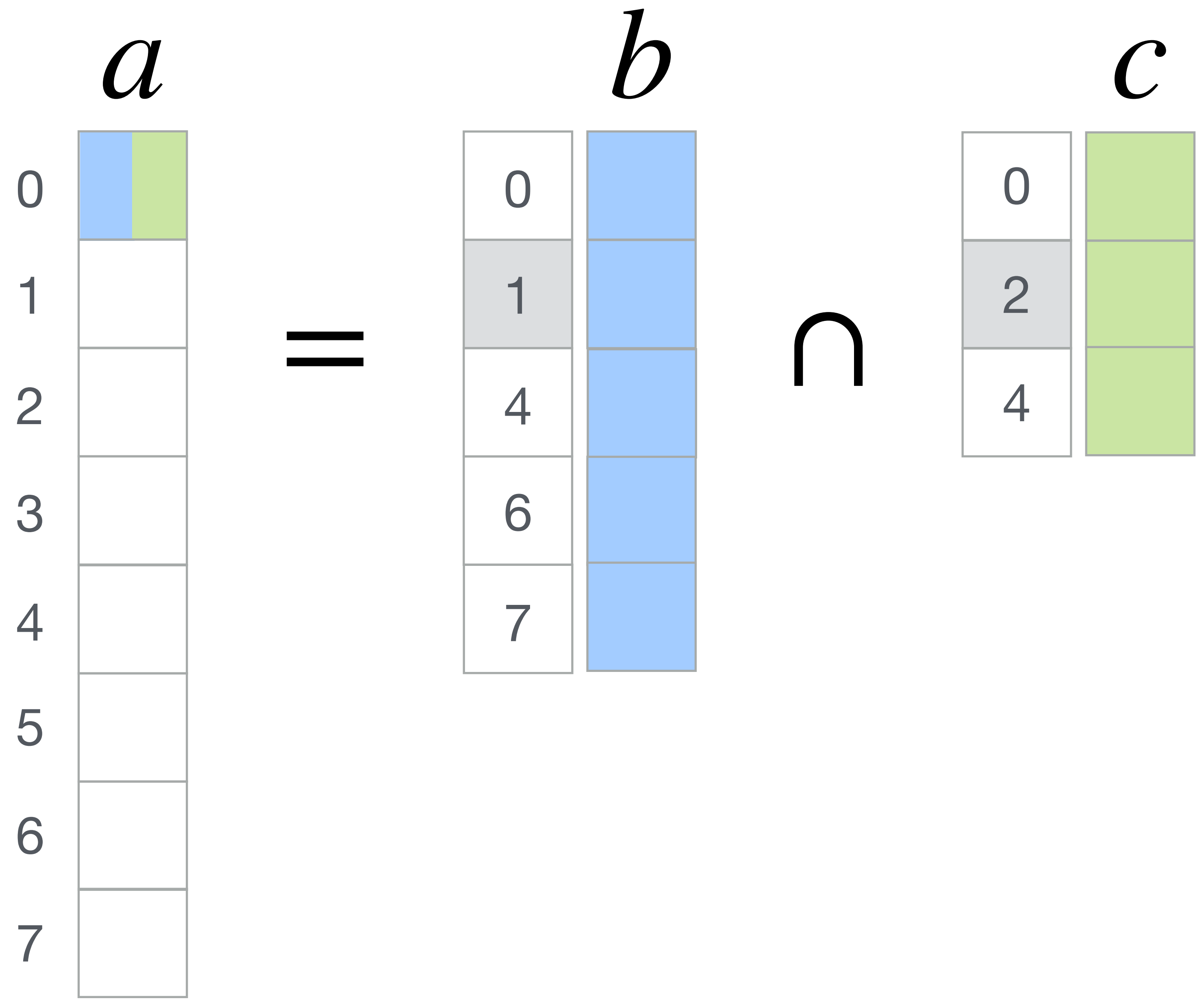


Merged coiteration

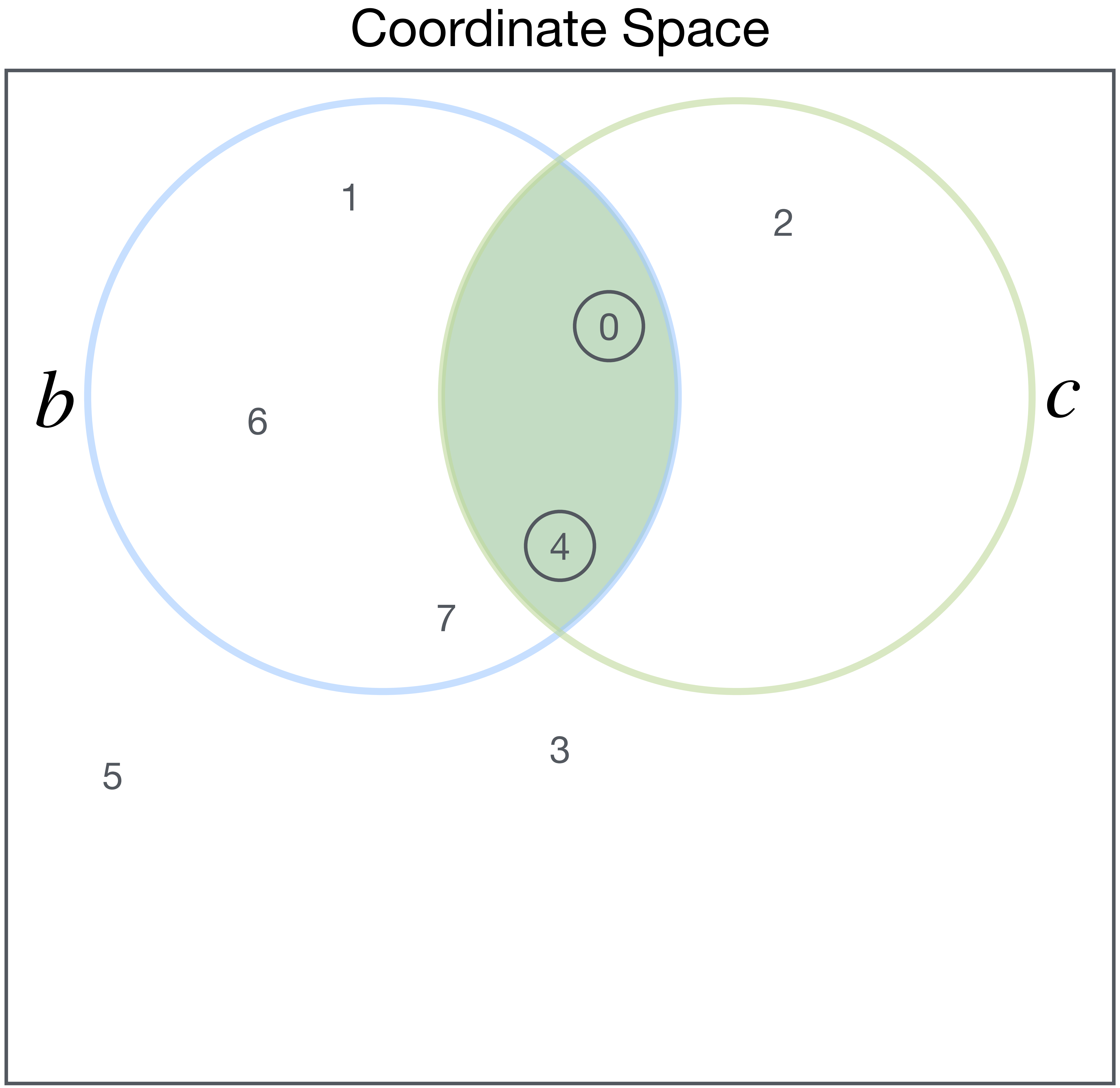
$$a_i = b_i c_i$$

Coordinate Space

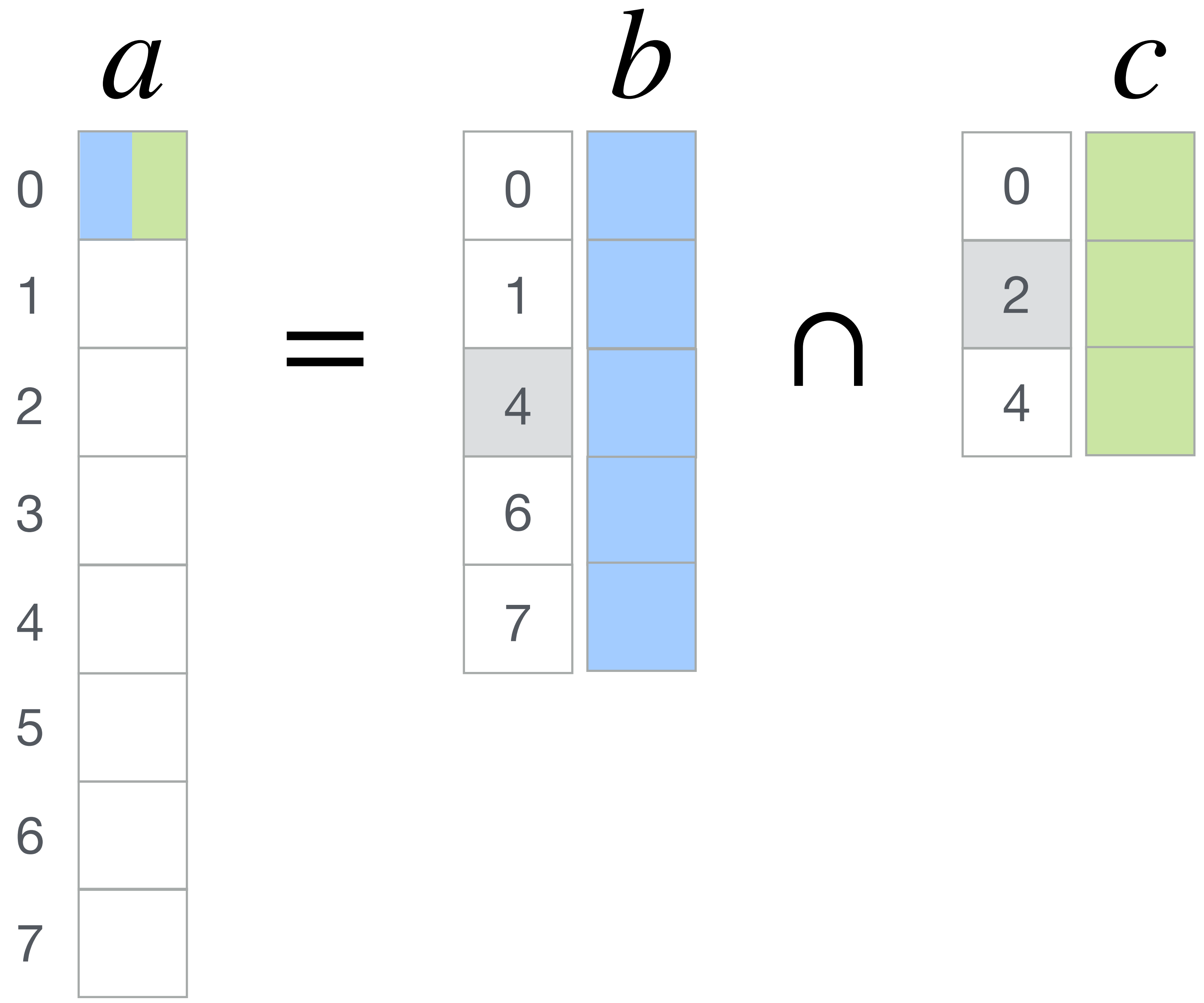




Merged coiteration

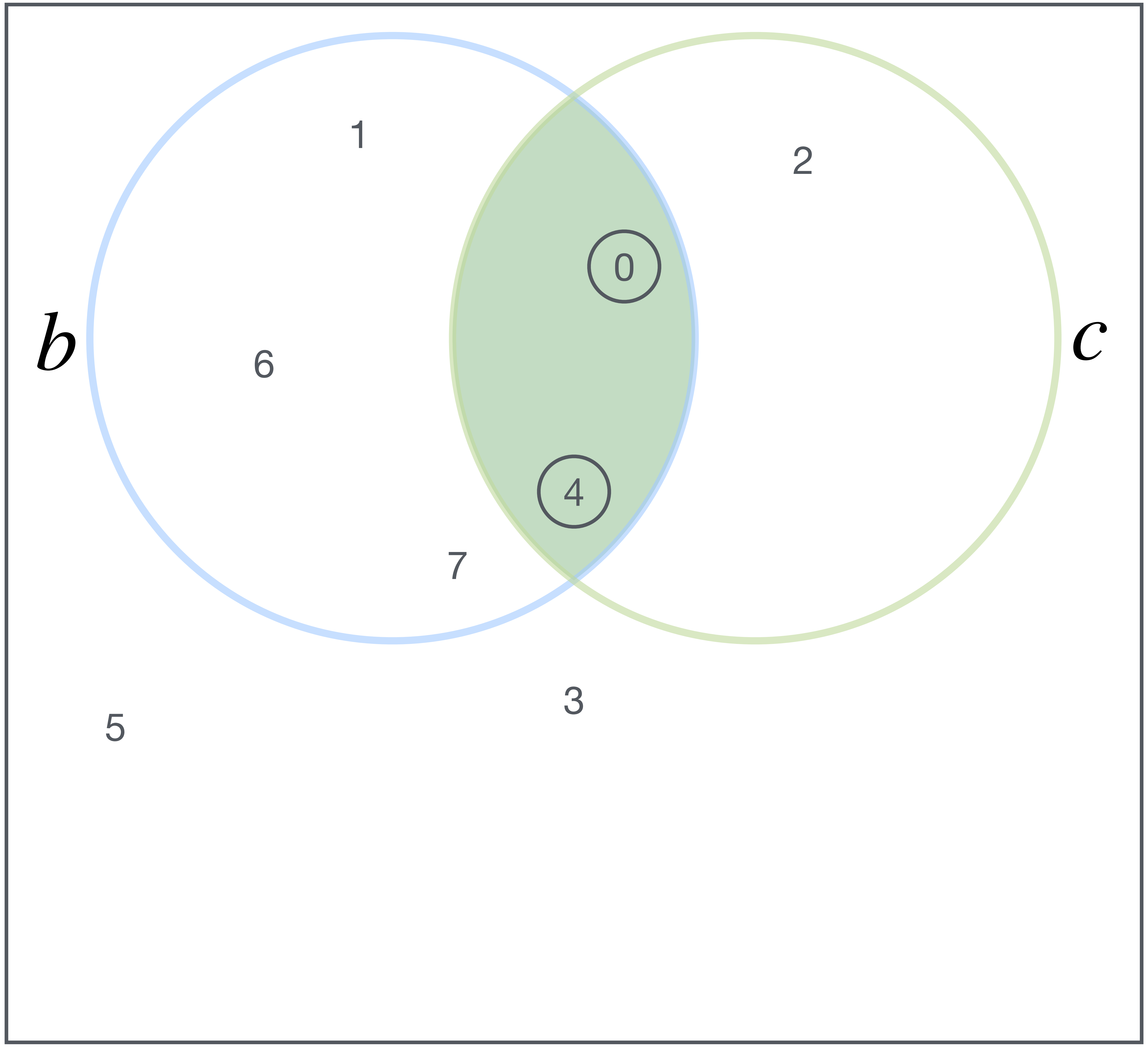


$$a_i = b_i c_i$$

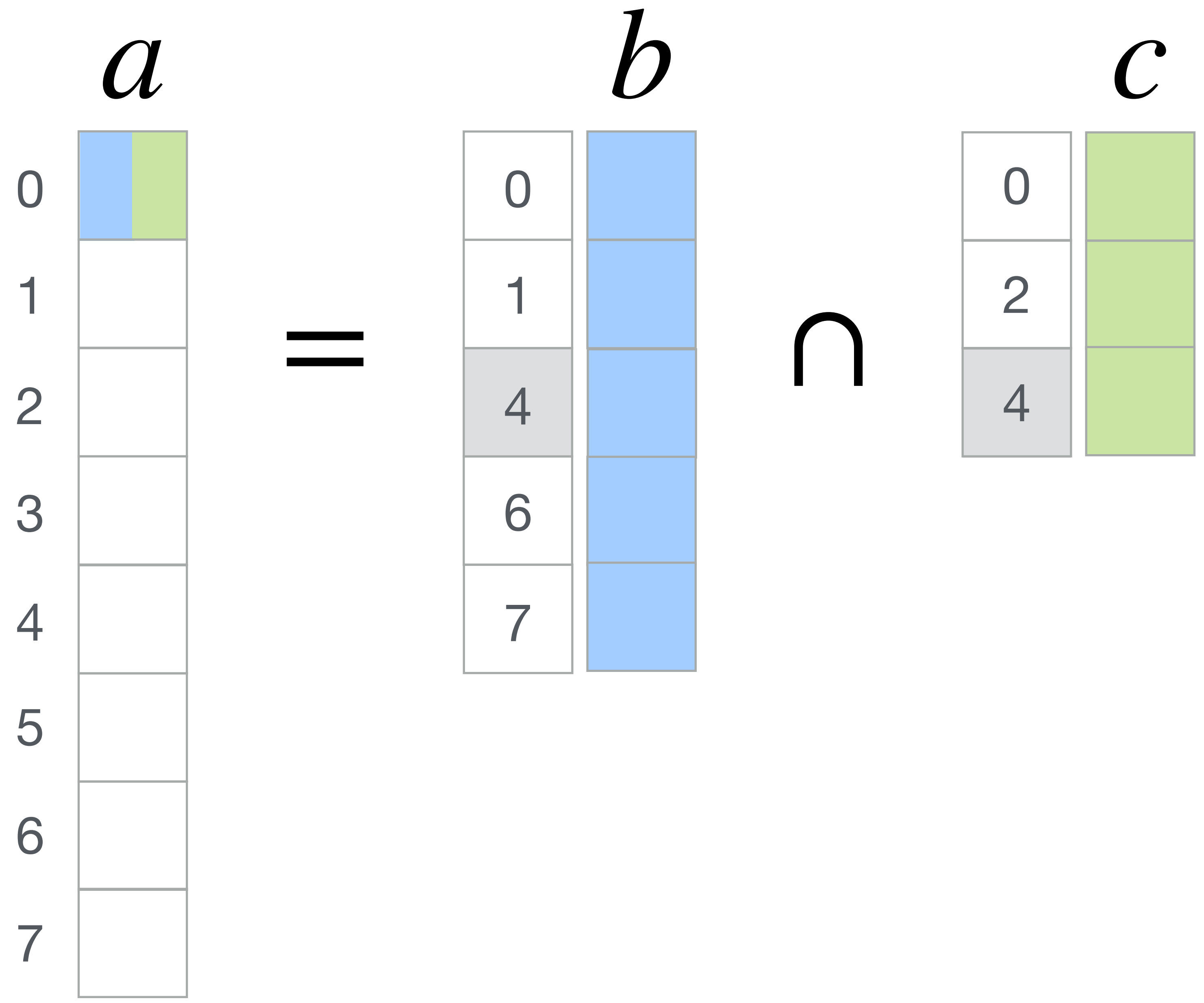


Merged coiteration

Coordinate Space

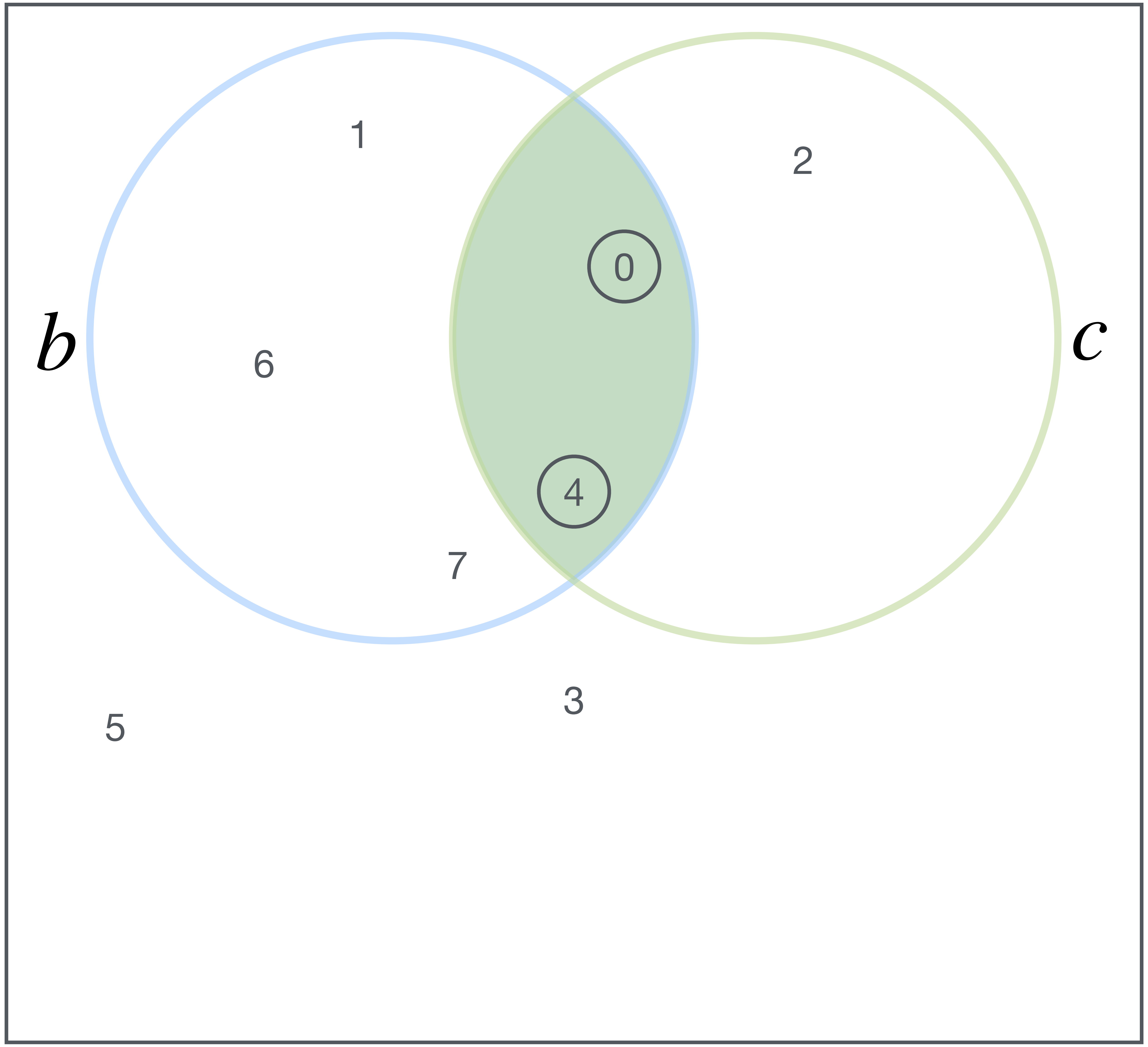


$$a_i = b_i c_i$$

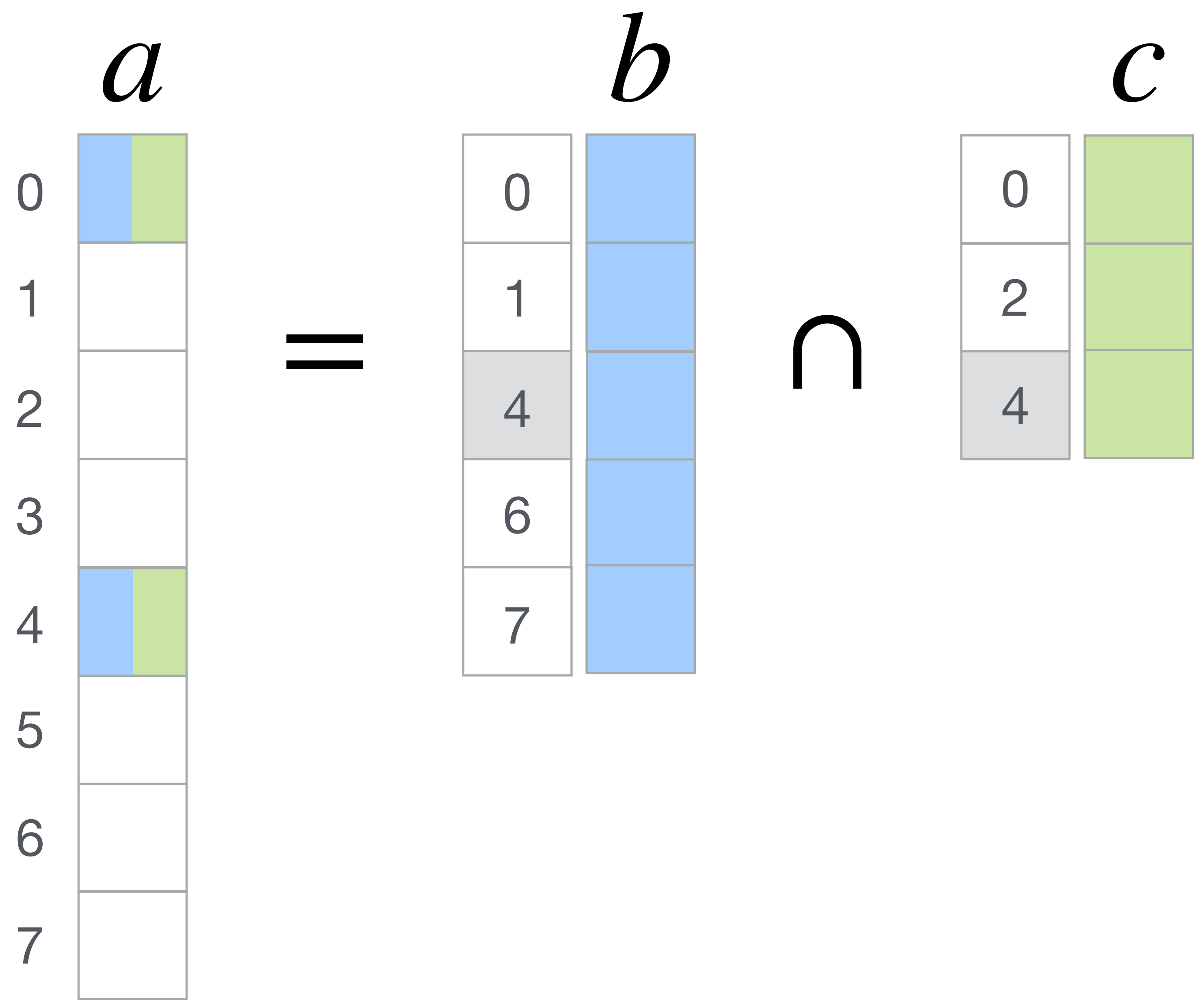


Merged coiteration

Coordinate Space

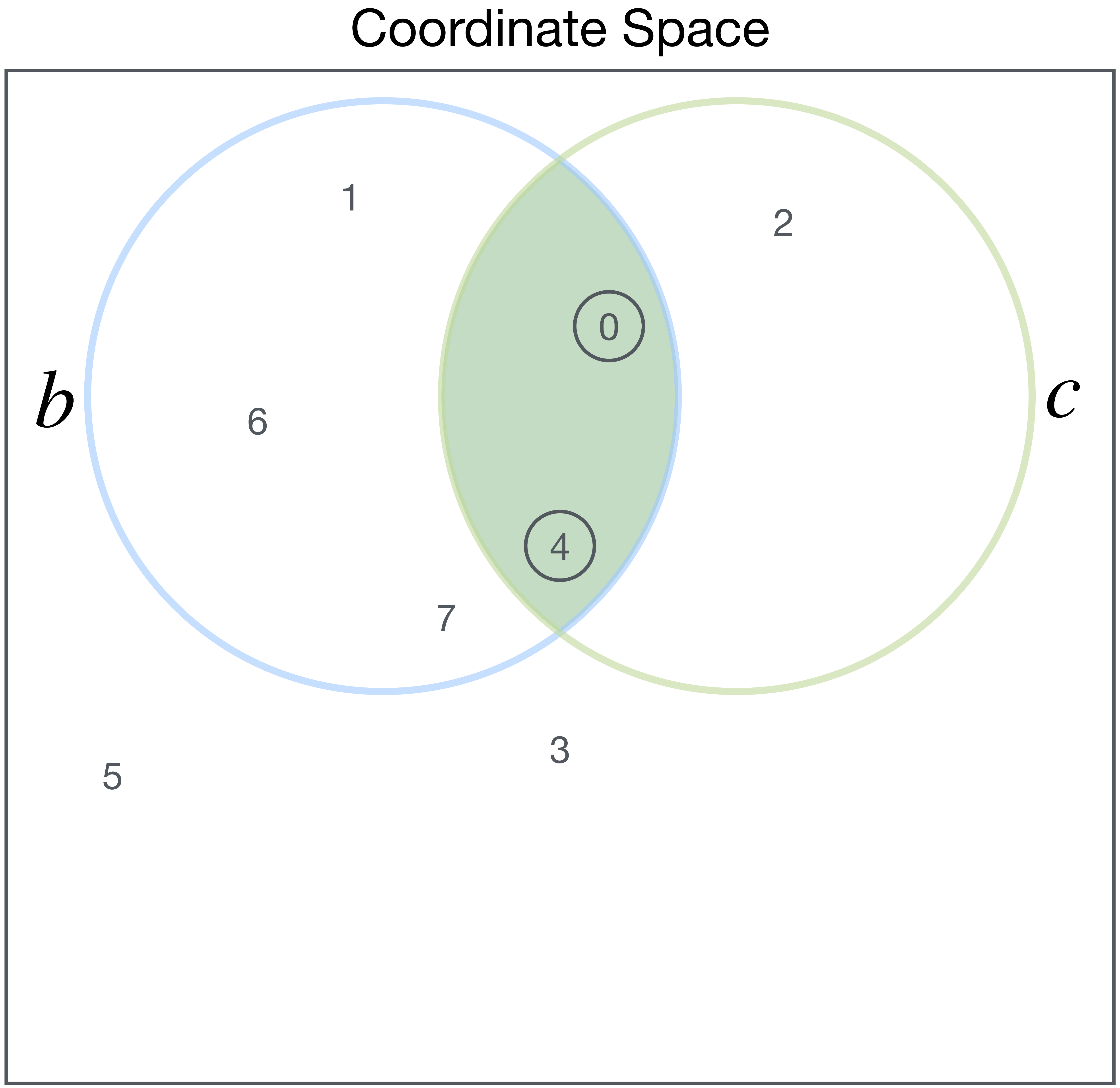


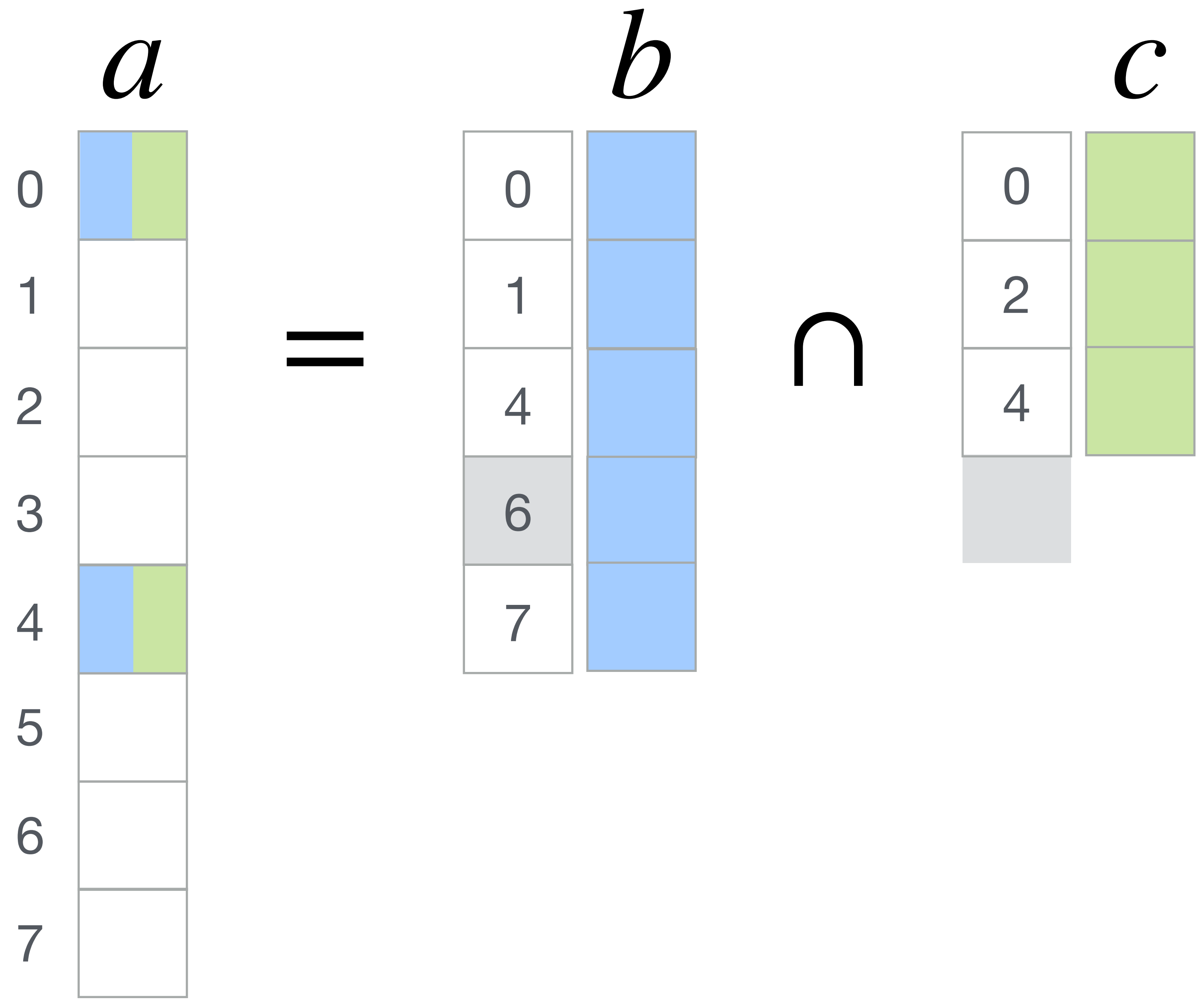
$$a_i = b_i c_i$$



Merged coiteration

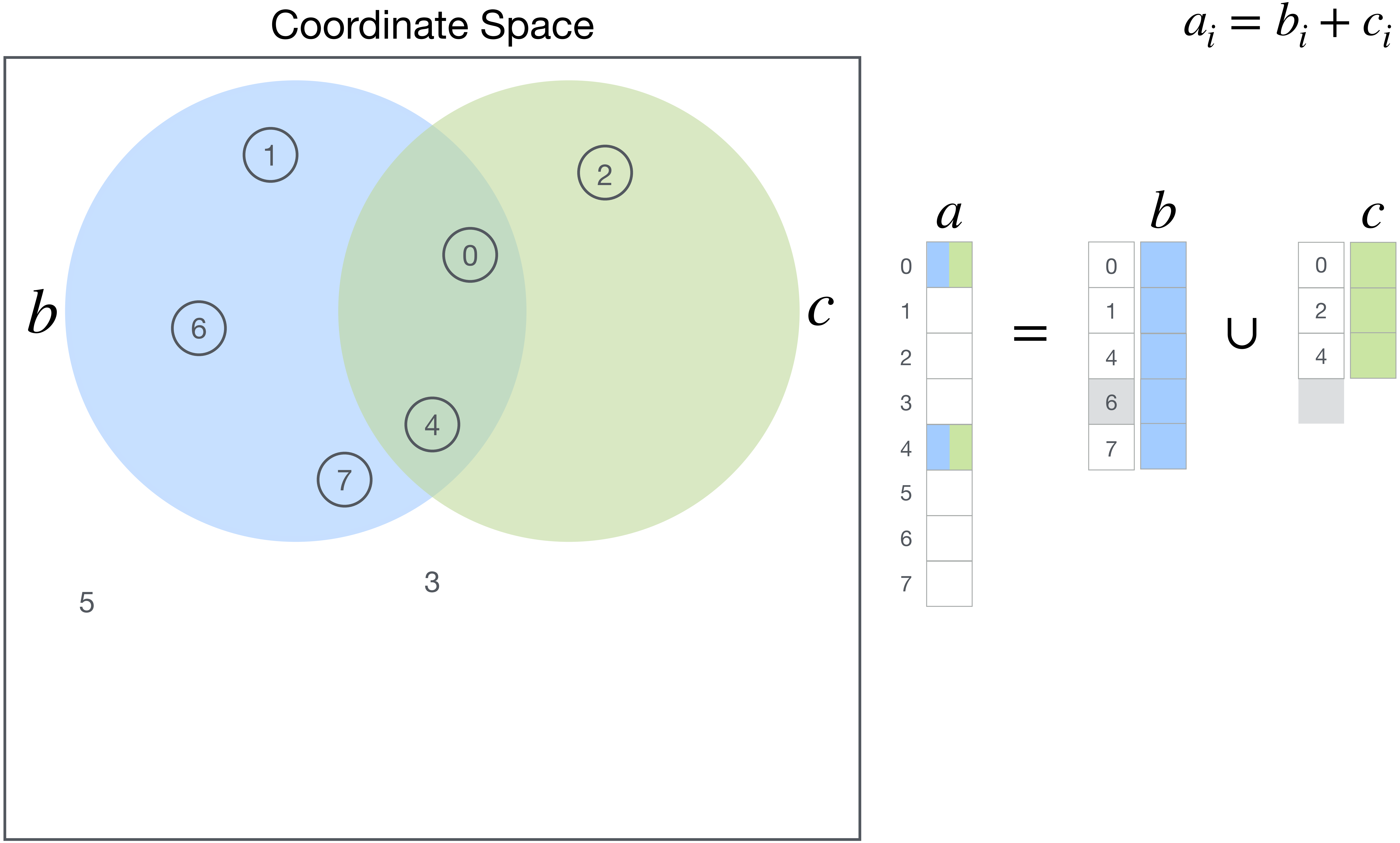
$$a_i = b_i c_i$$



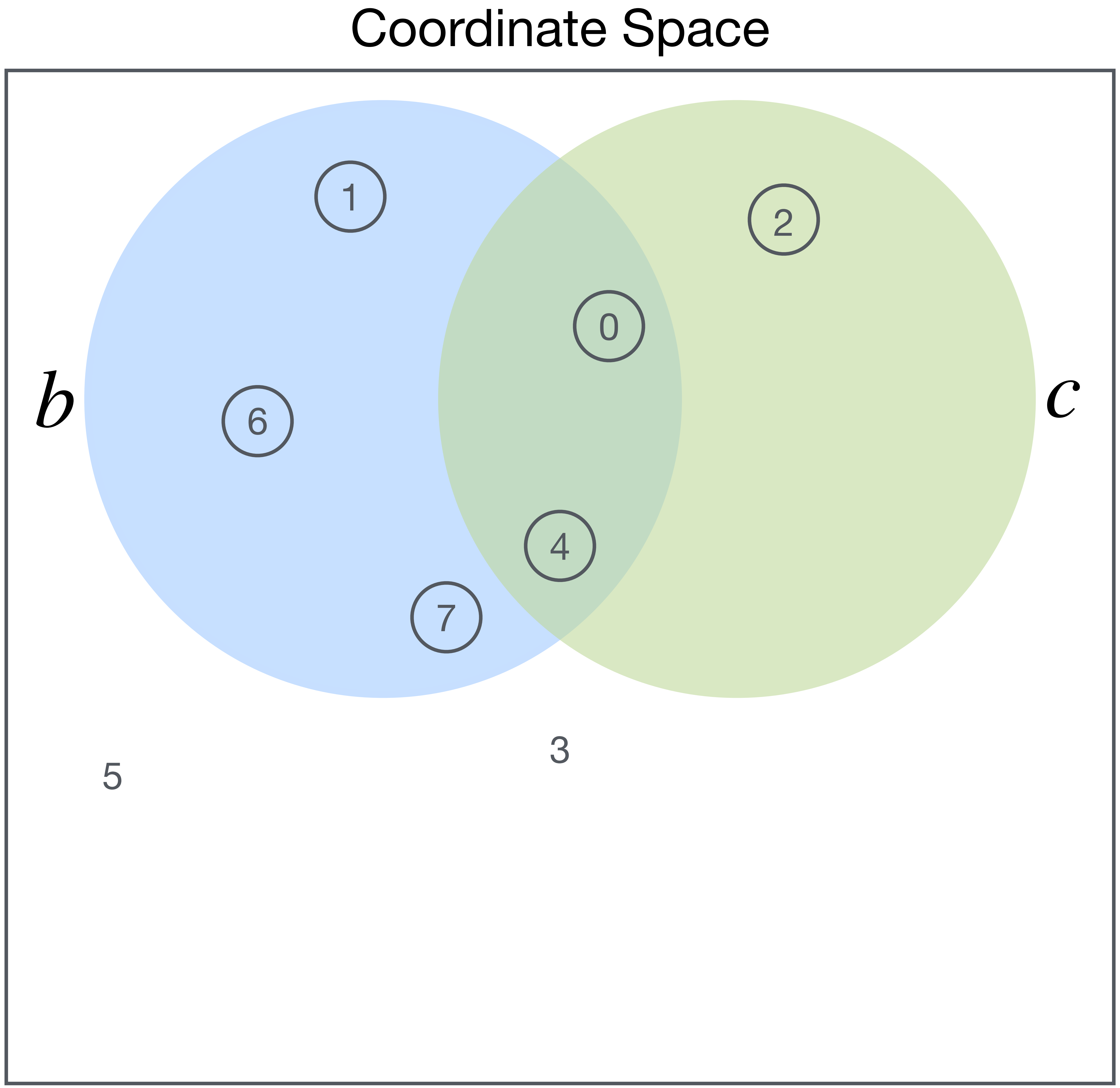


Merged coiteration

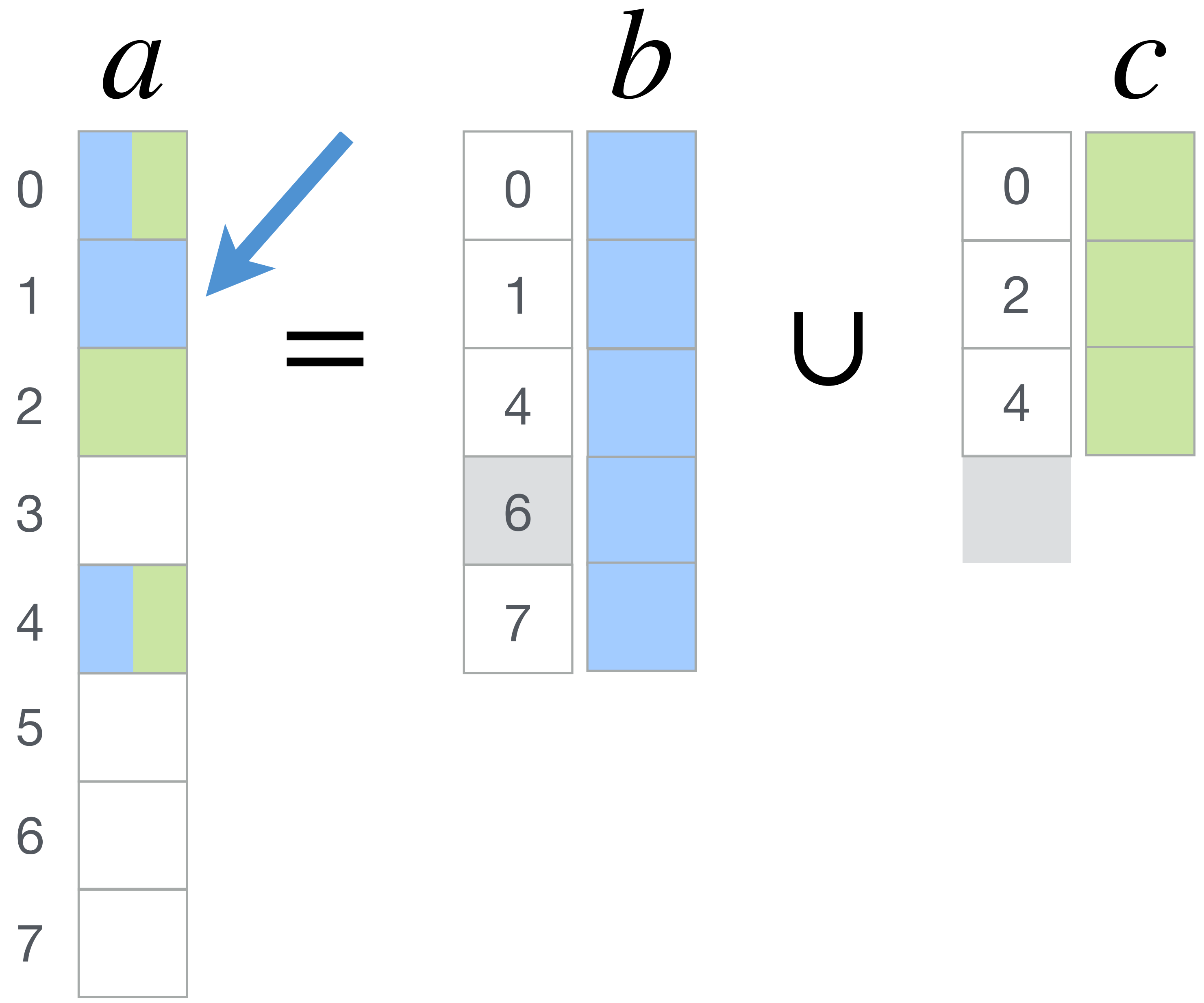
$$a_i = b_i + c_i$$



Merged coiteration

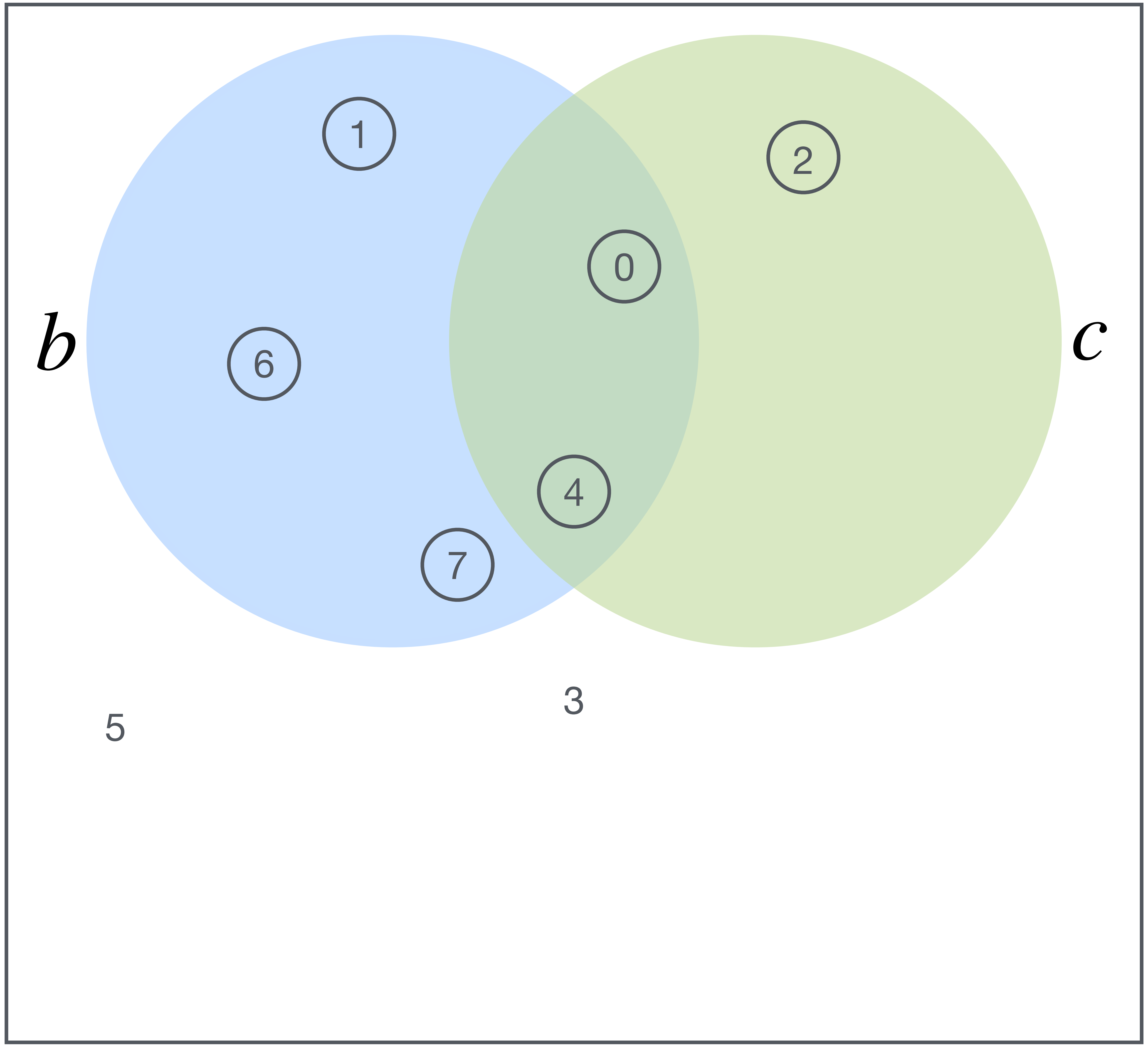


$$a_i = b_i + c_i$$

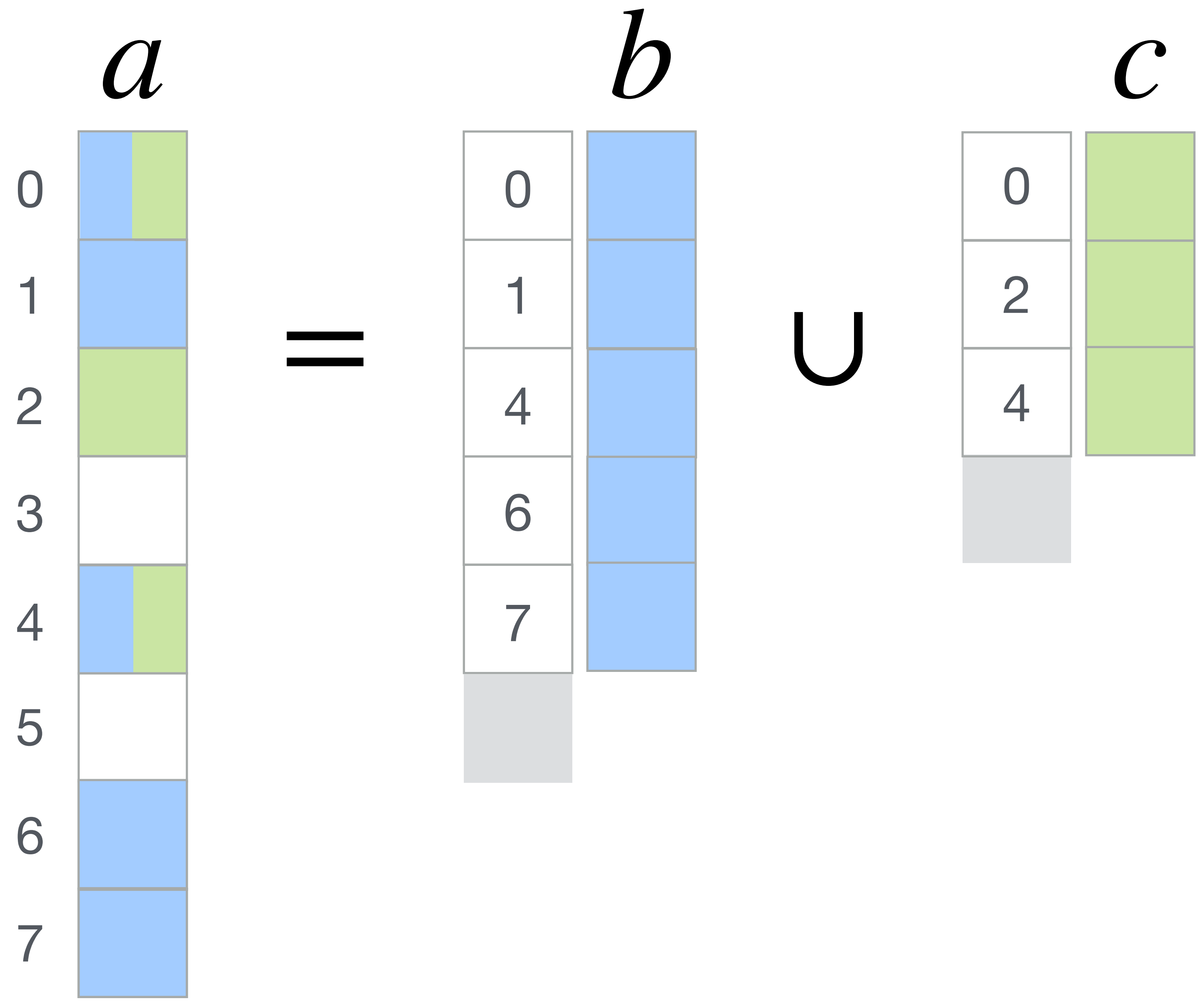


Merged coiteration

Coordinate Space



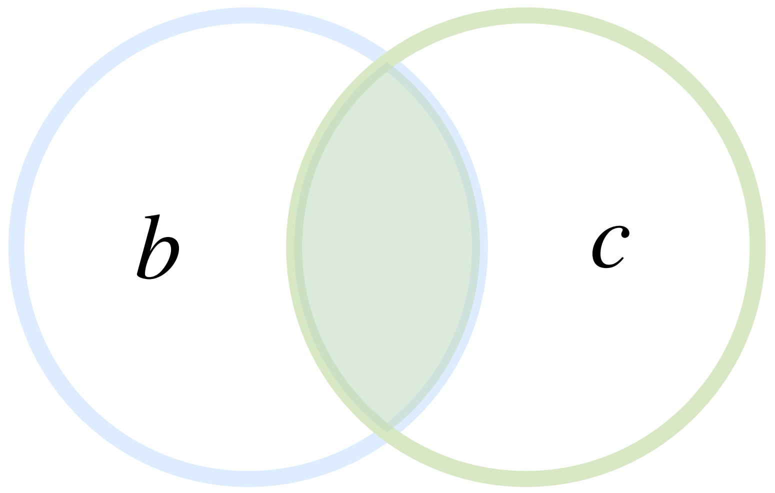
$$a_i = b_i + c_i$$



Merged coiteration code

Intersection $b \cap c$

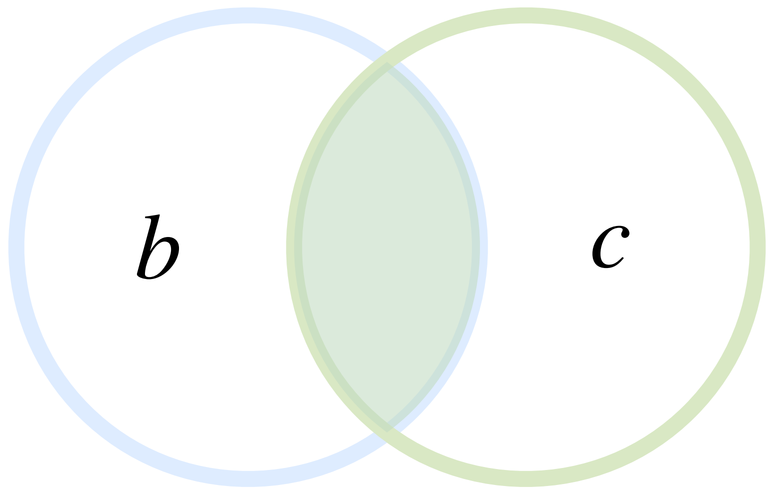
```
int pb = b_pos[0];
int pc = c_pos[0];
while (pb < b_pos[1] && pc < c_pos[1]) {
  int ib = b_crd[pb];
  int ic = c_crd[pc];
  int i = min(ib, ic);
  if (ib == i && ic == i) {
    a[i] = b[pb] * c[pc];
  }
  if (ib == i) pb++;
  if (ic == i) pc++;
}
```



Merged coiteration code

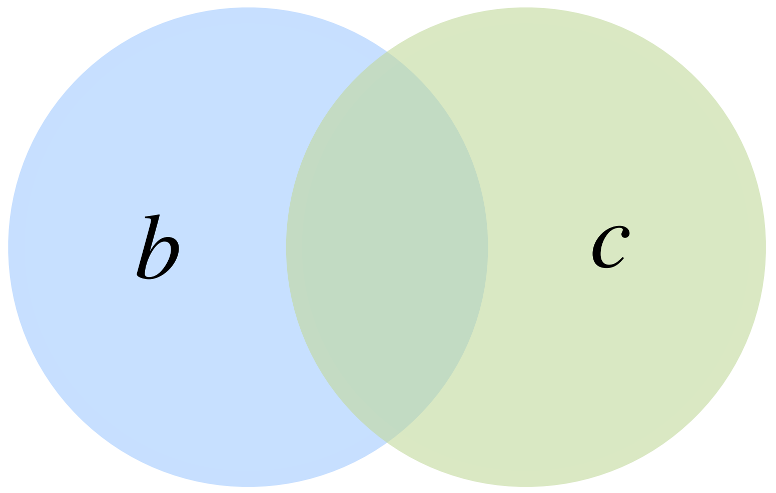
Intersection $b \cap c$

```
int pb = b_pos[0];
int pc = c_pos[0];
while (pb < b_pos[1] && pc < c_pos[1]) {
  int ib = b_crd[pb];
  int ic = c_crd[pc];
  int i = min(ib, ic);
  if (ib == i && ic == i) {
    a[i] = b[pb] * c[pc];
  }
  if (ib == i) pb++;
  if (ic == i) pc++;
}
```



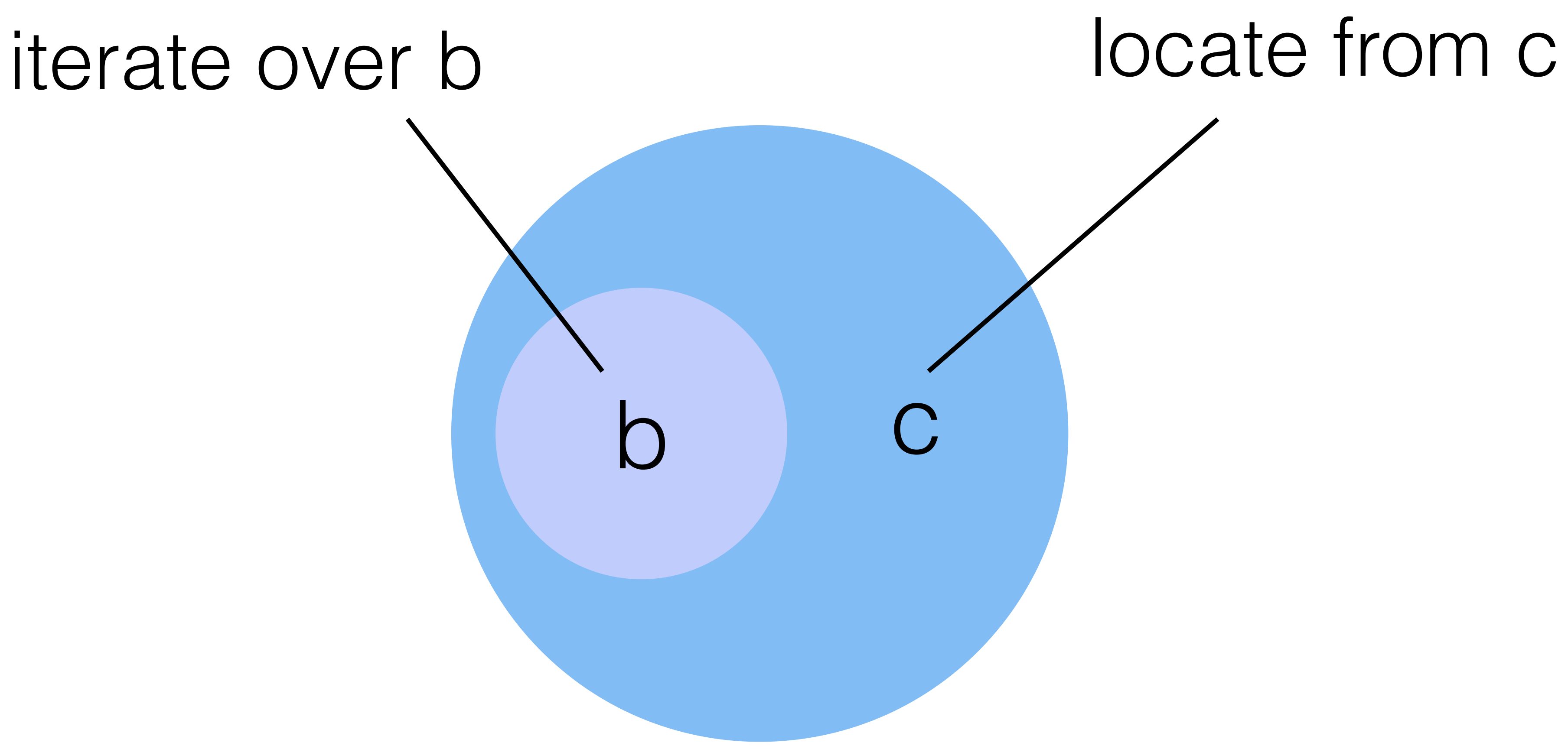
Union $b \cup c$

```
int pb = b_pos[0];
int pc = c_pos[0];
while (pb < b_pos[1] && pc < c_pos[1]) {
  int ib = b_crd[pb];
  int ic = c_crd[pc];
  int i = min(ib, ic);
  if (ib == i && ic == i) {
    a[i] = b[pb] + c[pc];
  }
  else if (ib == i) {
    a[i] = b[pb];
  }
  else {
    a[i] = c[pc];
  }
  if (ib == i) pb++;
  if (ic == i) pc++;
}

while (pb < b_pos[1]) {
  int i = b_crd[pb];
  a[i] = b[pb++];
}

while (pc < c_pos[1]) {
  int i = c_crd[pc];
  a[i] = c[pc++];
}
```



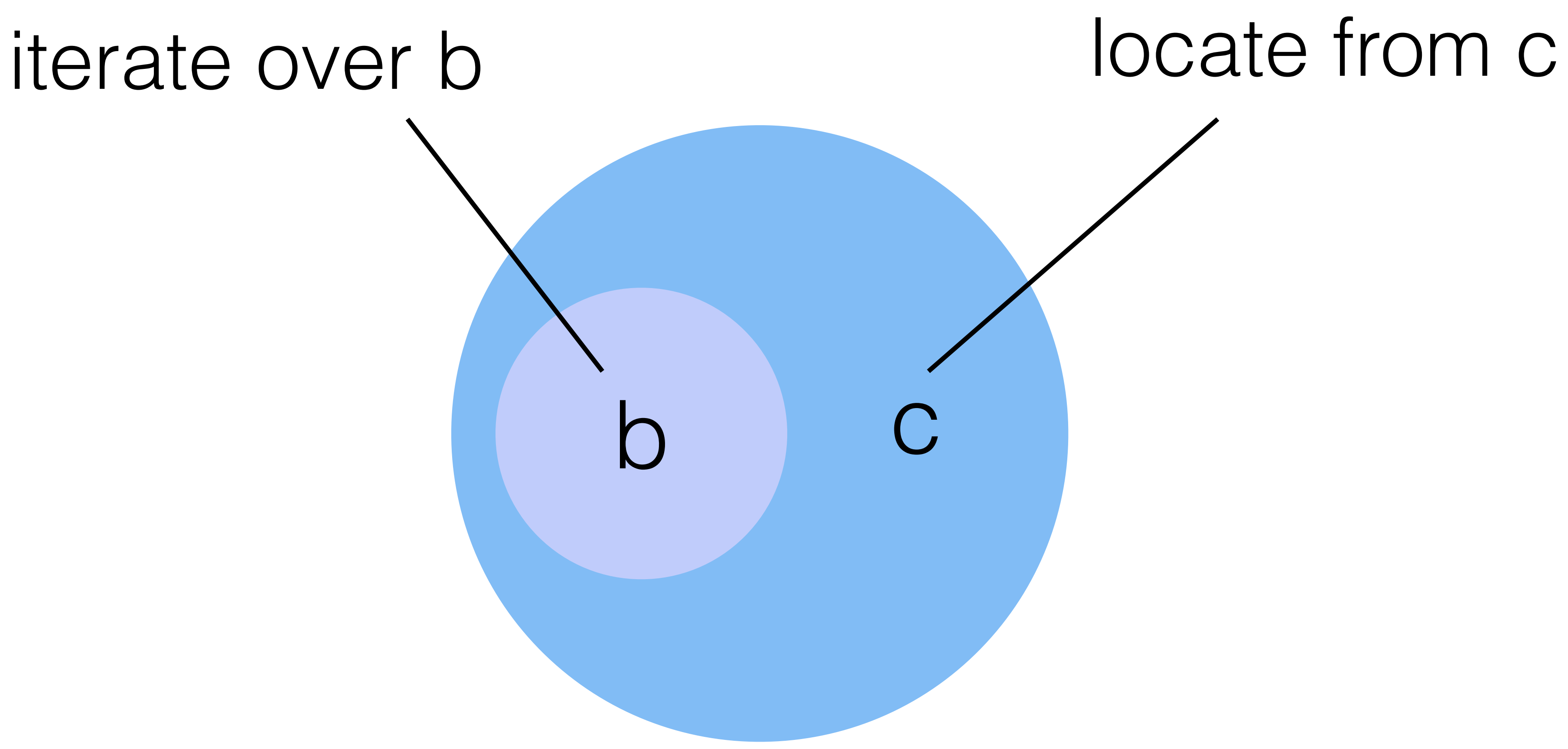
Iterate-and-locate examples (intersection)

$$a = \sum_i b_i c_i$$



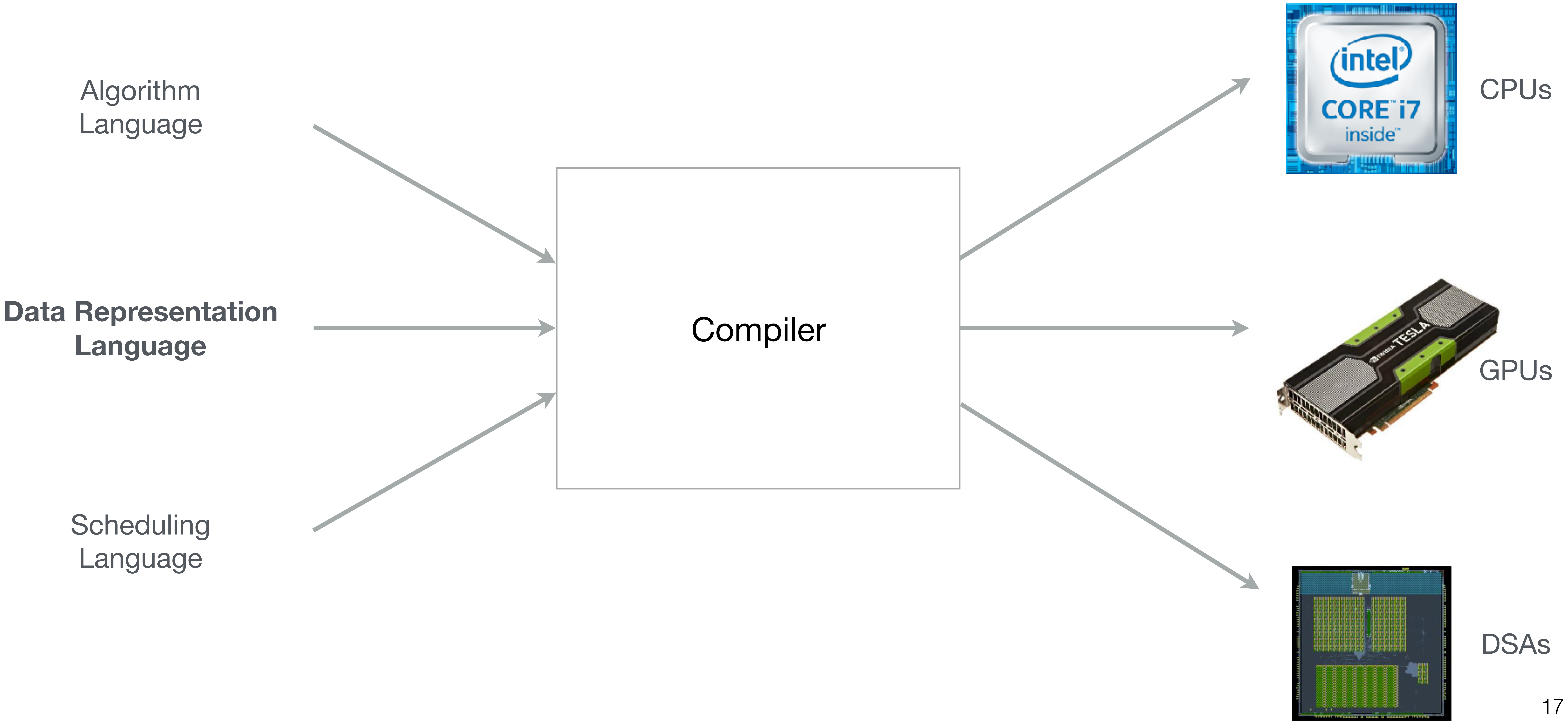
Iterate-and-locate examples (intersection)

$$a = \sum_i b_i c_i$$

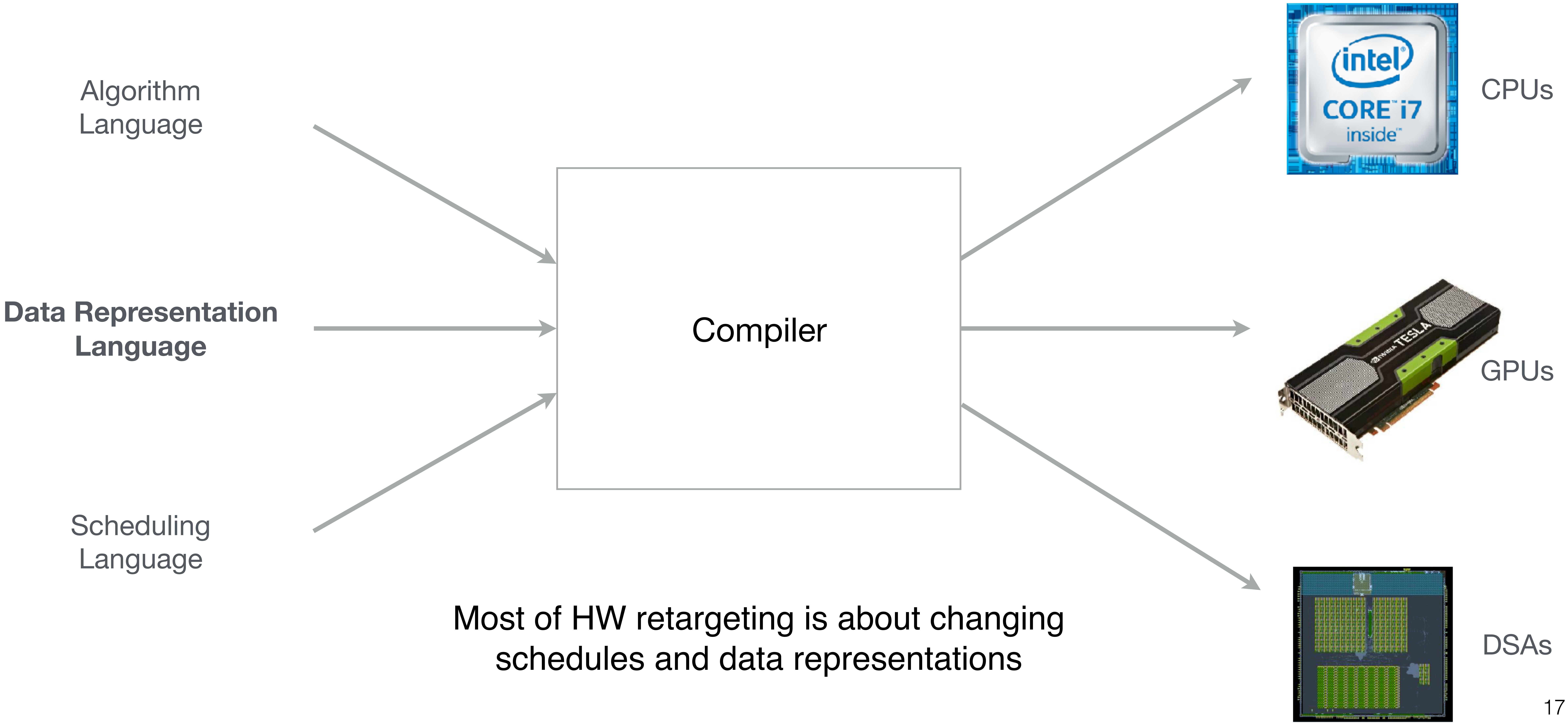



```
for (int pb = b_pos[0]; pb < b_pos[1]; pb ++) {
  int i = b_crd[pb];
  a += b[pb] * c[i];
}
```

Separation of Algorithm, Data Representation, and Schedule



Separation of Algorithm, Data Representation, and Schedule



Most of HW retargeting is about changing
schedules and data representations