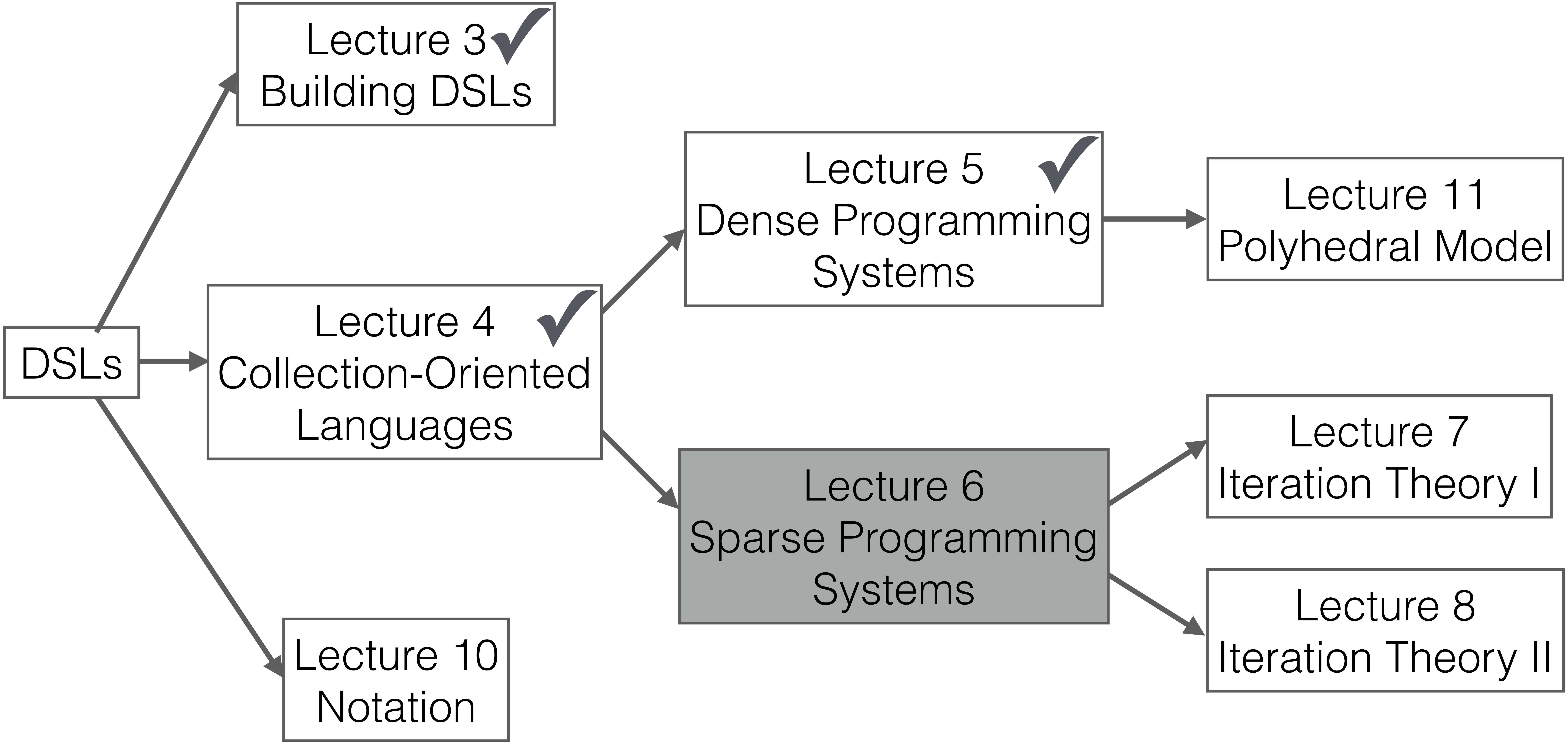
Lecture 6 - Sparse Programming Systems

Stanford CS343D (Fall 2020)

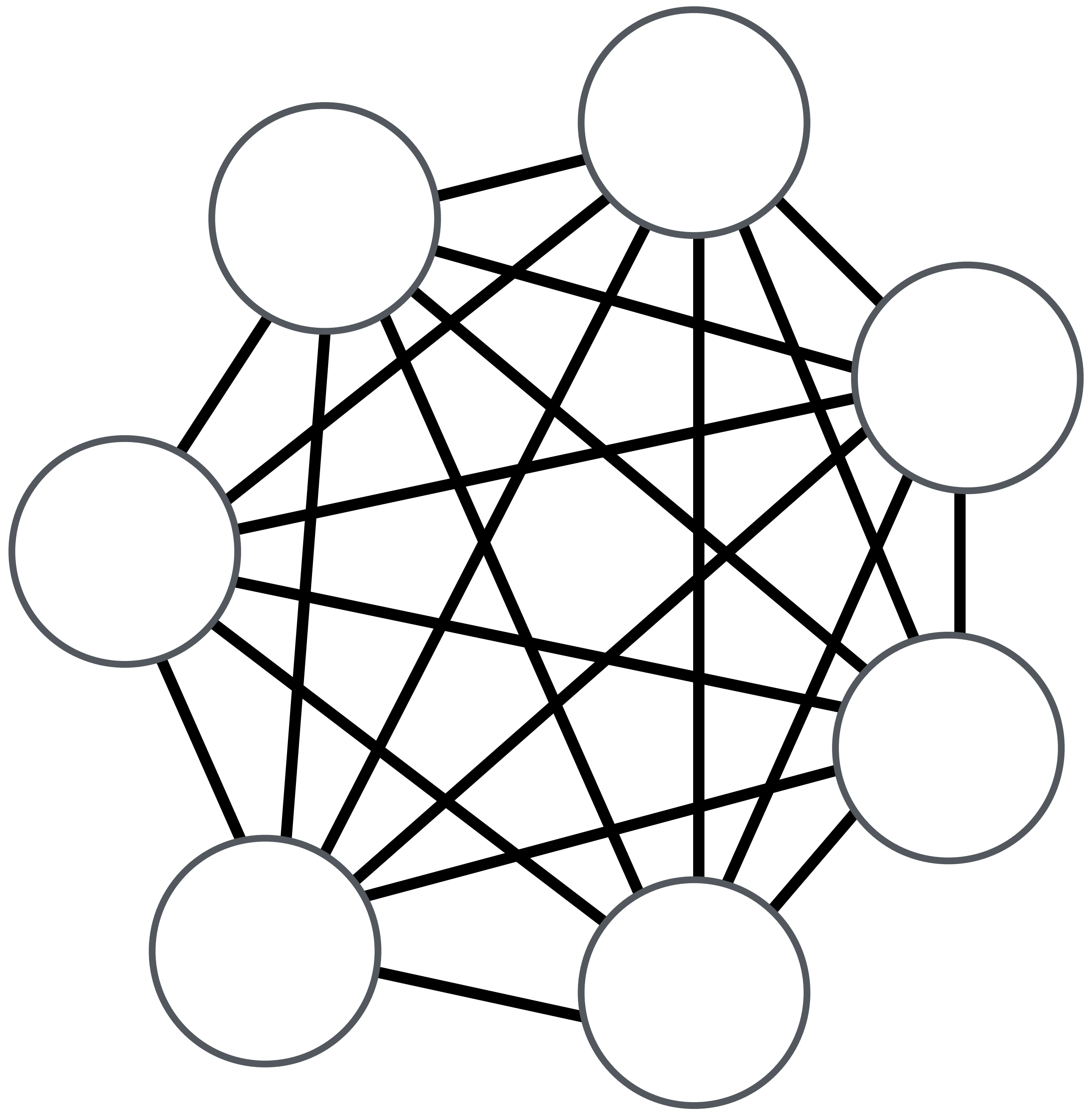
Fred Kjolstad and Pat Hanrahan

Overview of lectures in the coming weeks

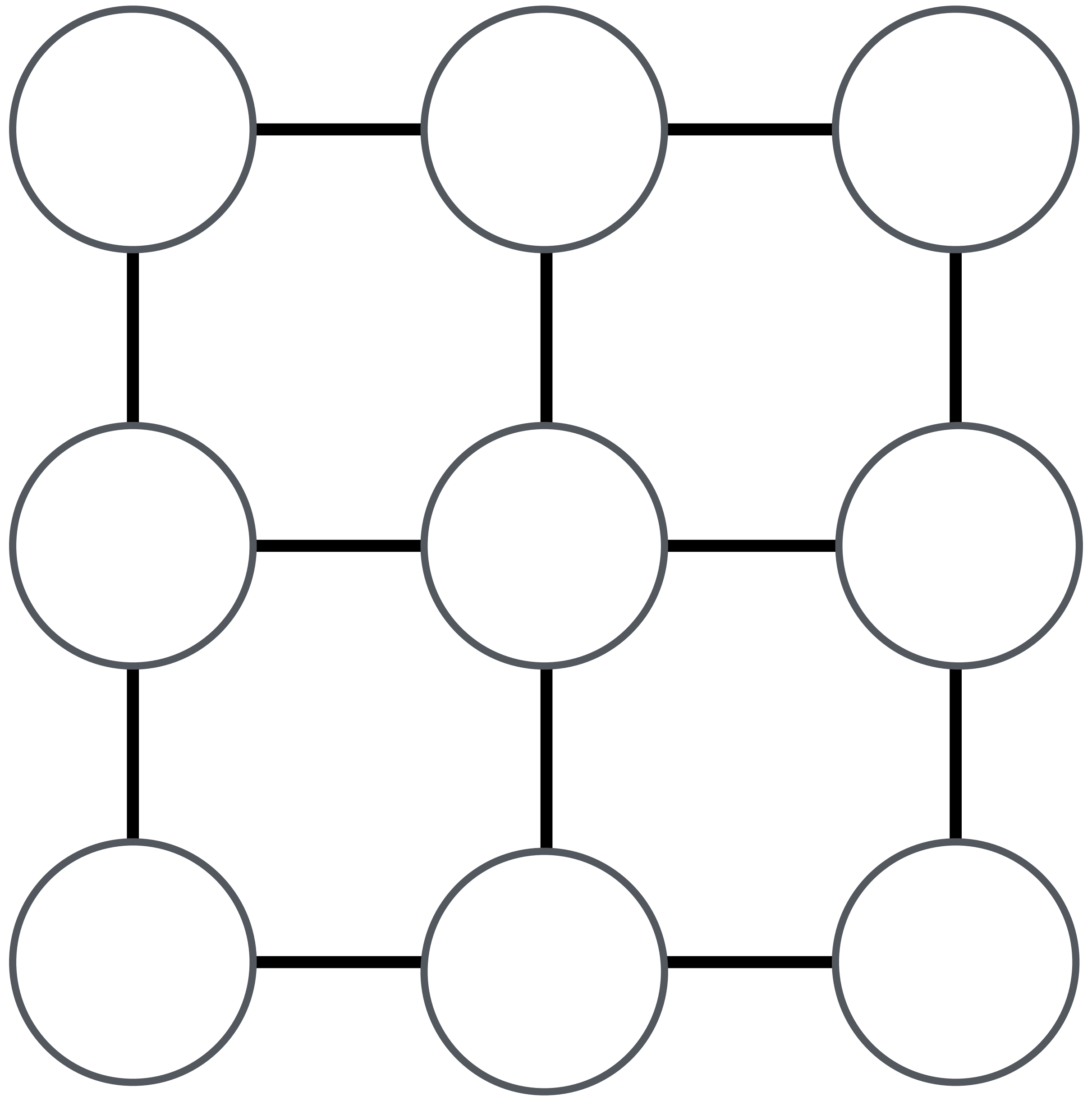


Terminology: Regular and Irregular

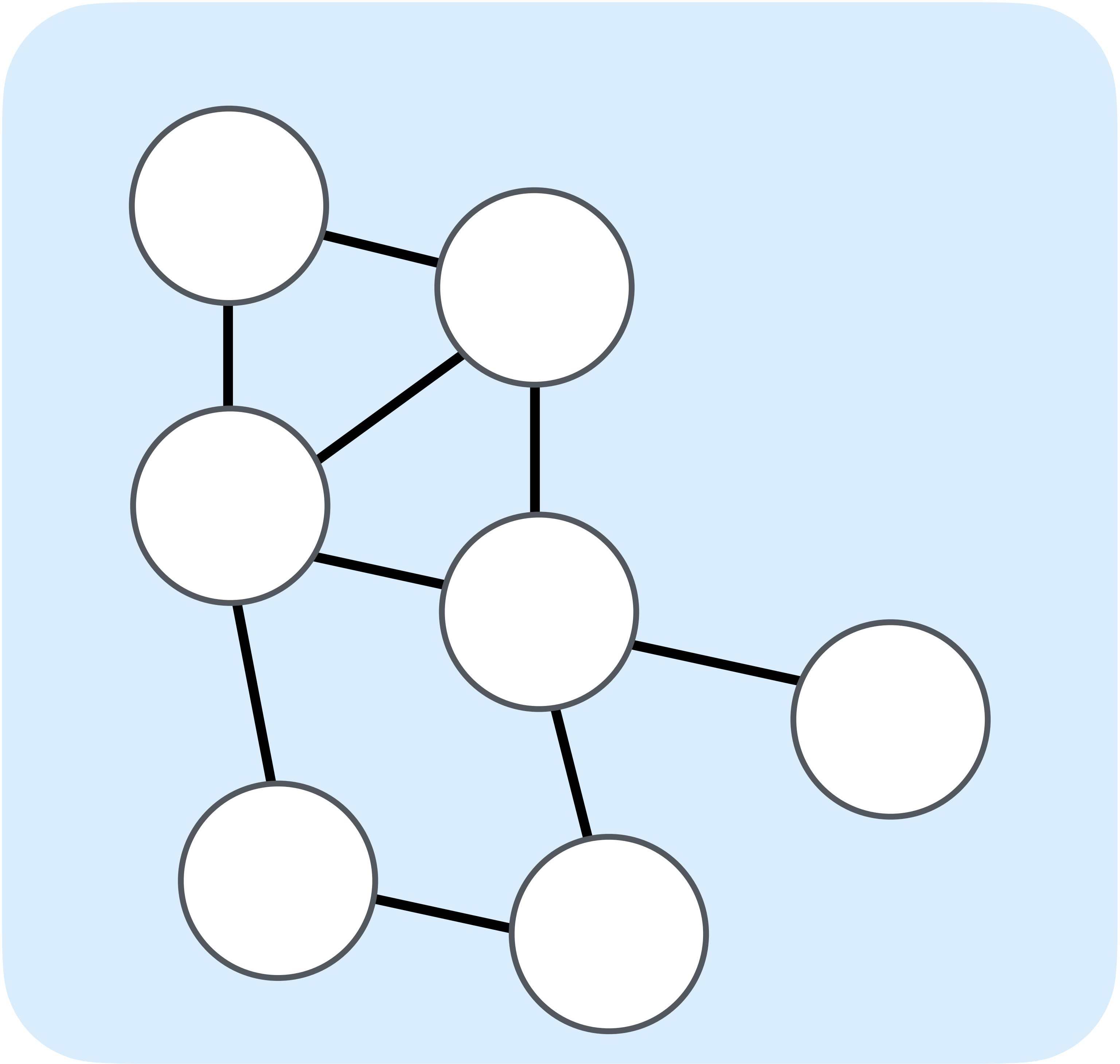
Fully Connected System



Regular System

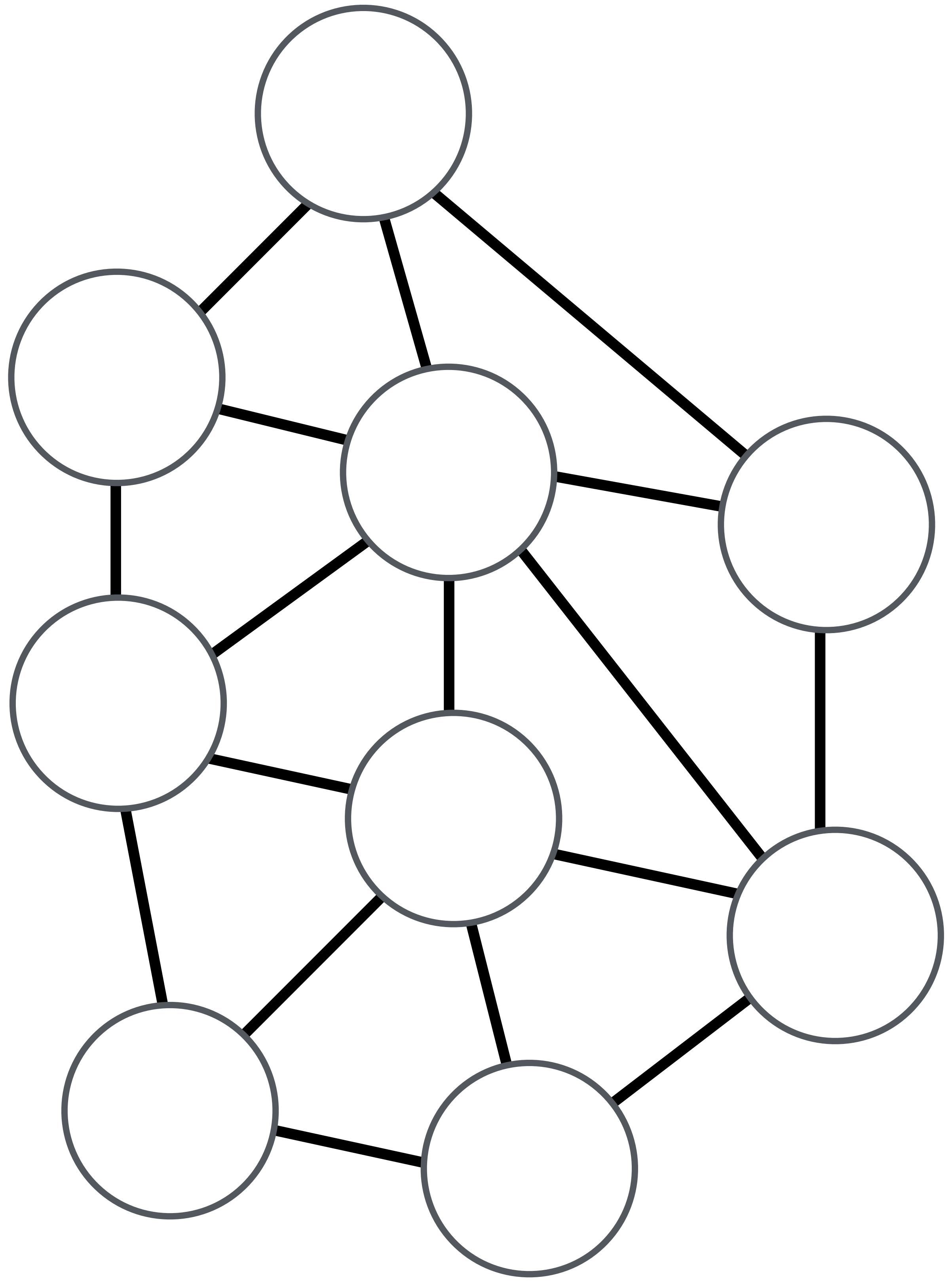


Irregular System

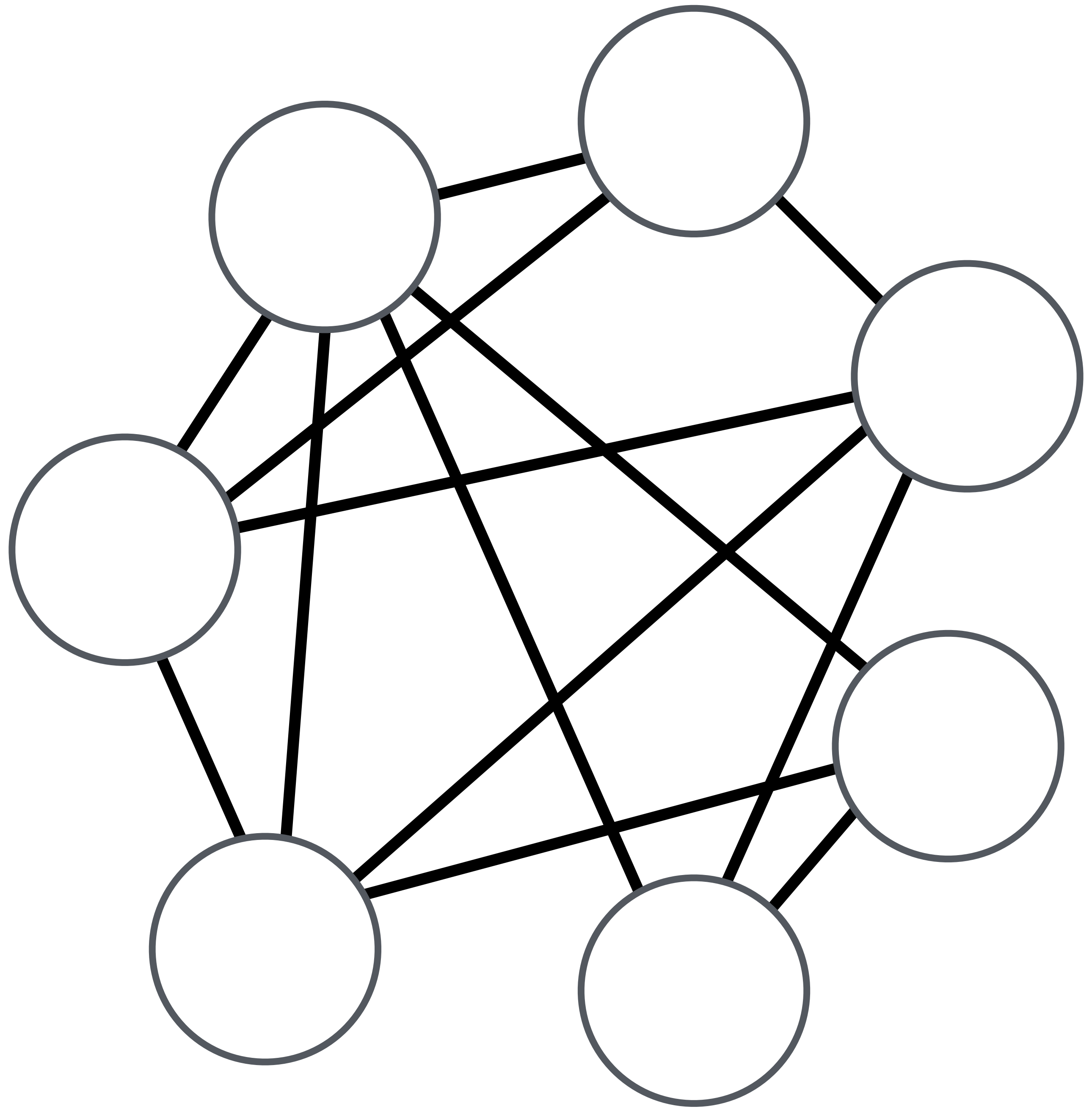


Three main classes of irregular systems

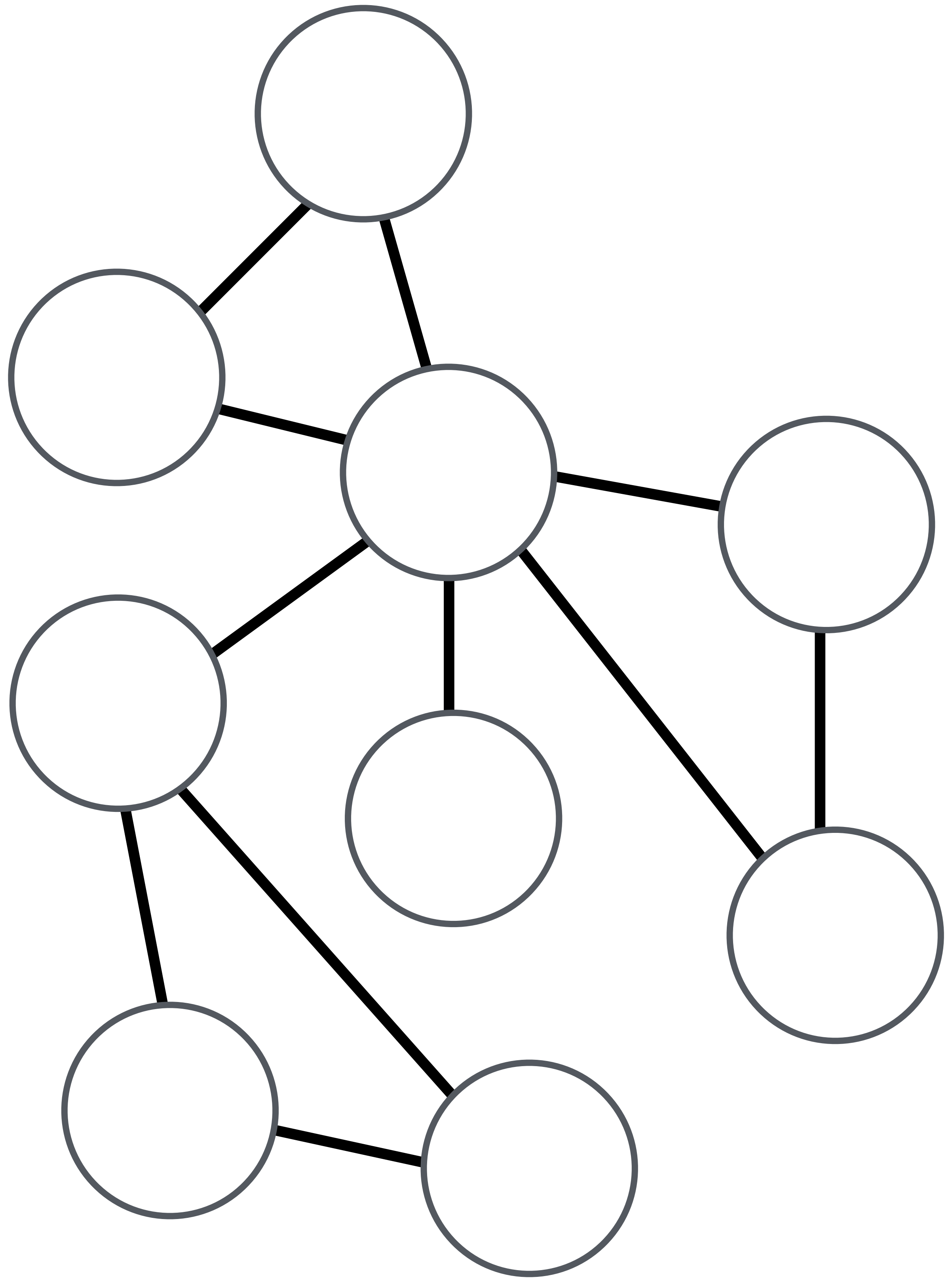
Road Networks



Random Sparsity

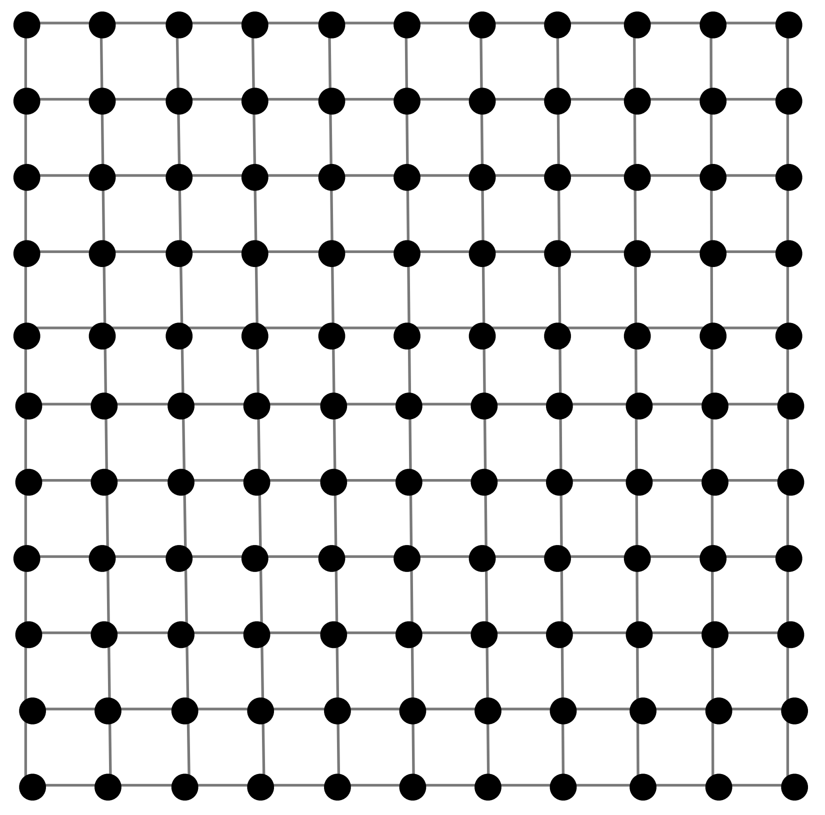


Power Law Graphs



Terminology: Dense and Sparse

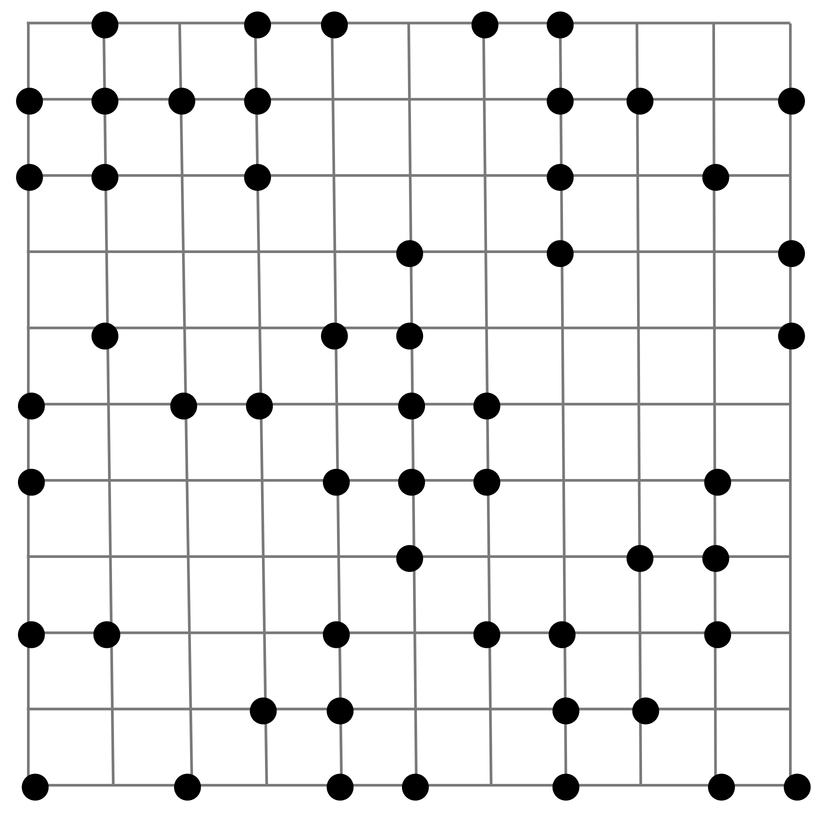
Dense loop iteration space



```
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

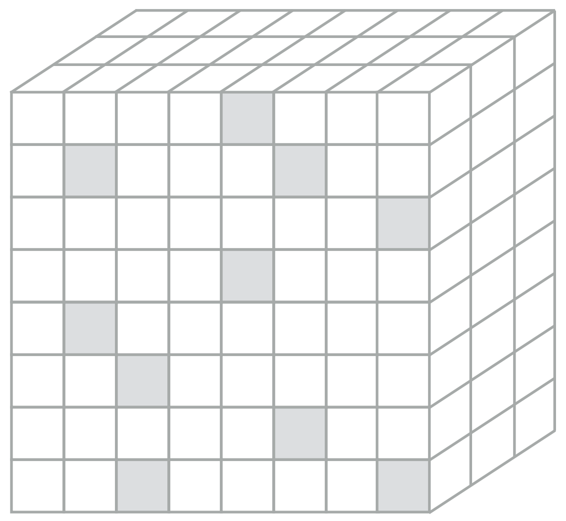


```
for (int i = 0; i < m; i++) {
 for (int pA = A2_pos[i]; pA < A_pos[i+1]; pA++) {
    int j = A_crd[pA];
    y[i] += A[pA] * x[j];
  }
}
```

$$y = Ax$$

Three sparse applications areas

Tensors



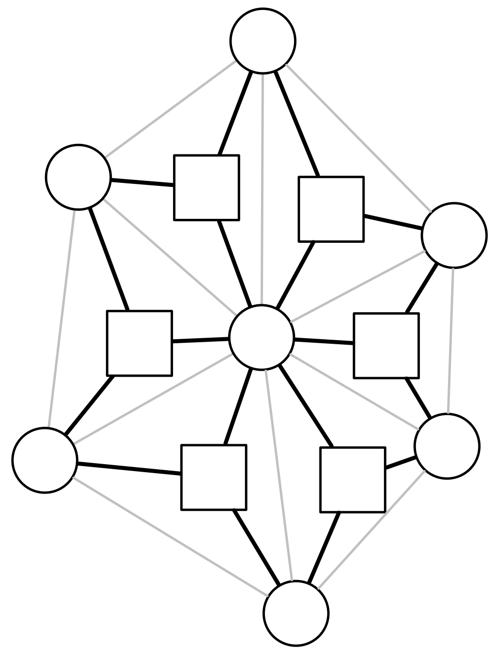
Nonzeros are a subset of the cartesian combination of sets

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

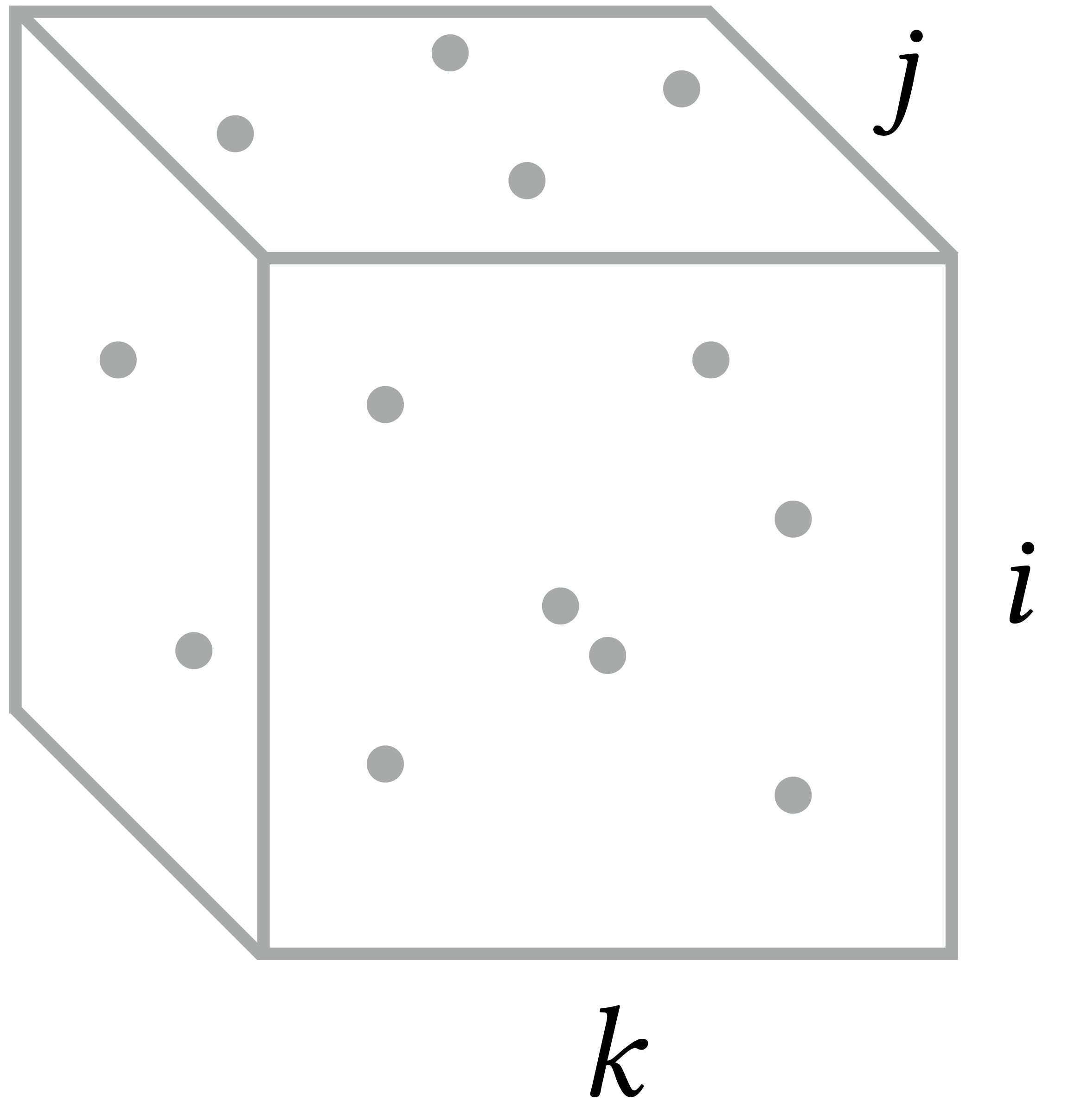
A relation is a subset of the cartesian combination of sets

Graphs



Graph edges are a subset of the cartesian combination of sets

Sparse Iteration Spaces



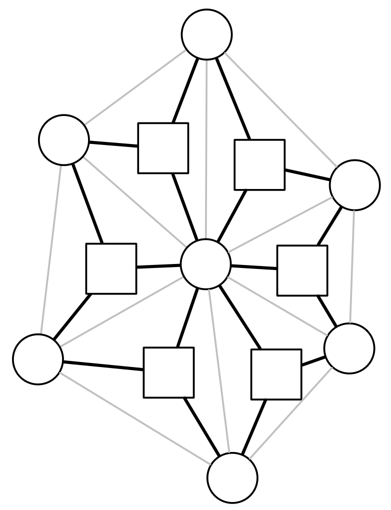
Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

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

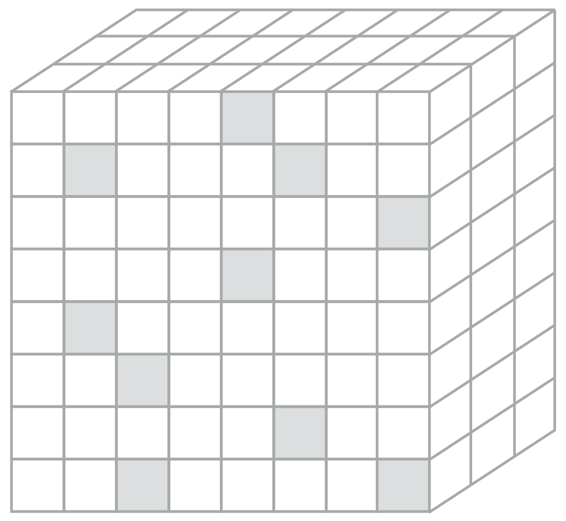
Combine data to form systems

Graphs

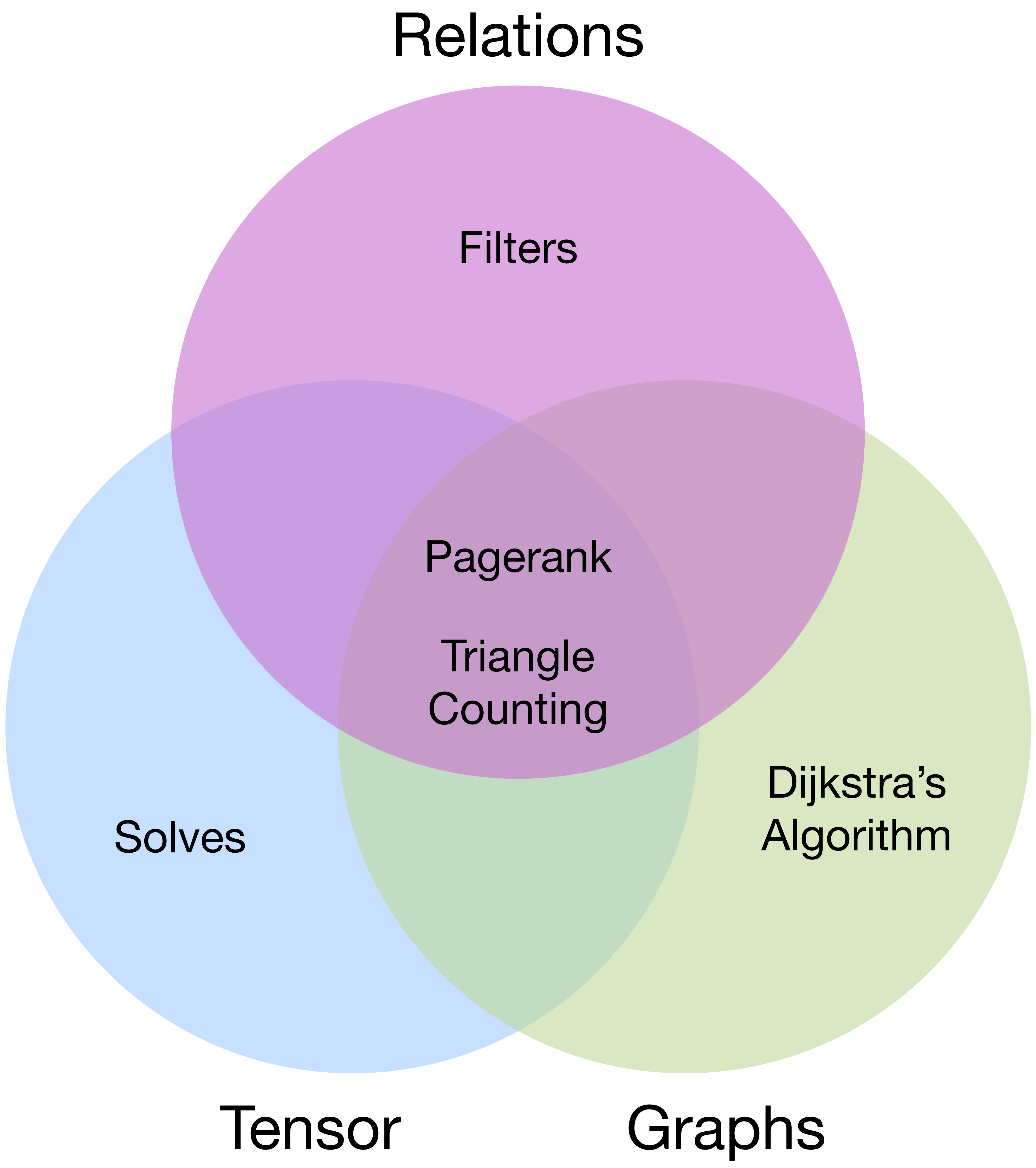


Local operations on systems

Tensors

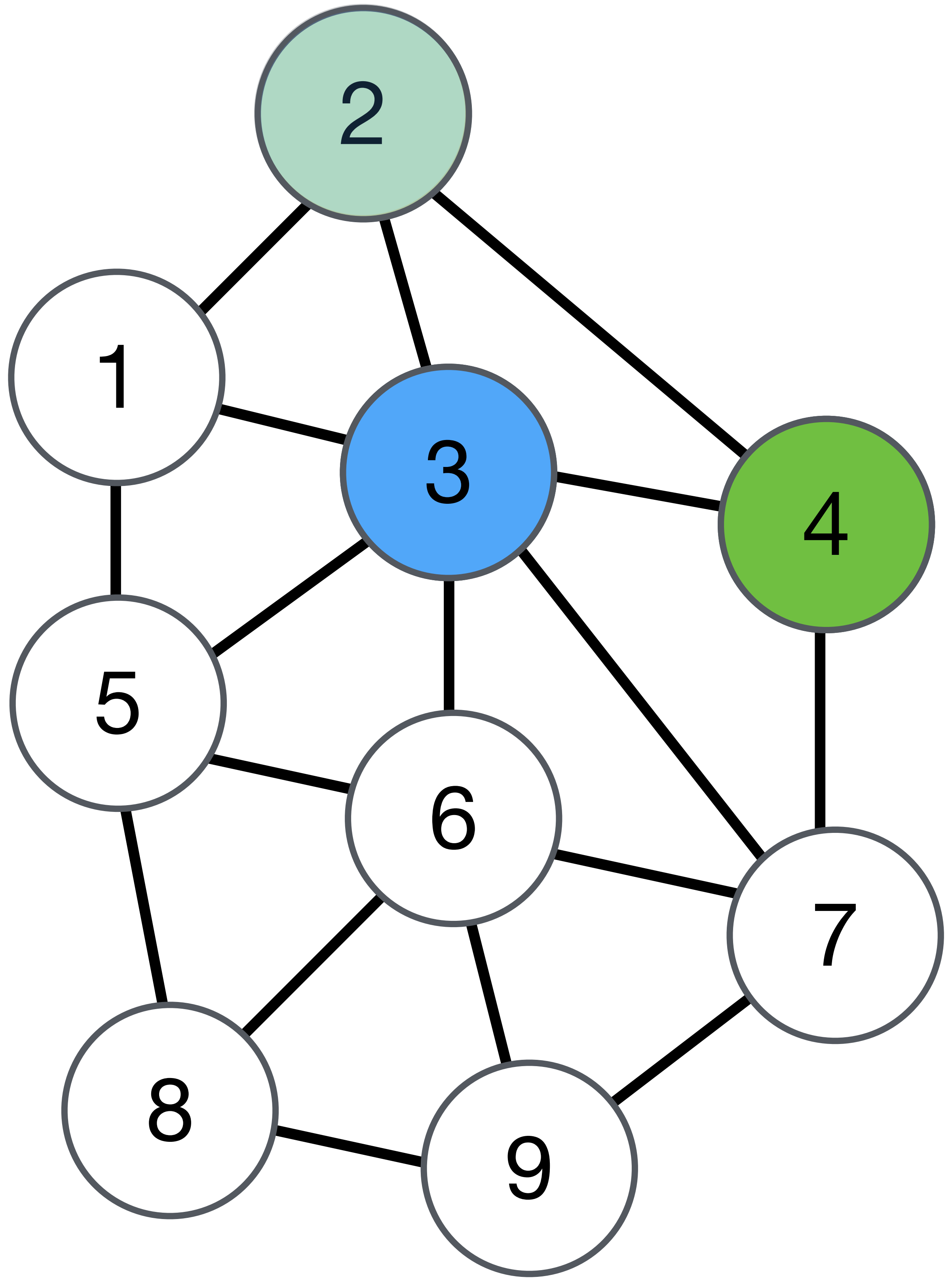


Global operations on systems



Triangle counting on graphs, relations, and tensors

On graphs



On relations

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$$

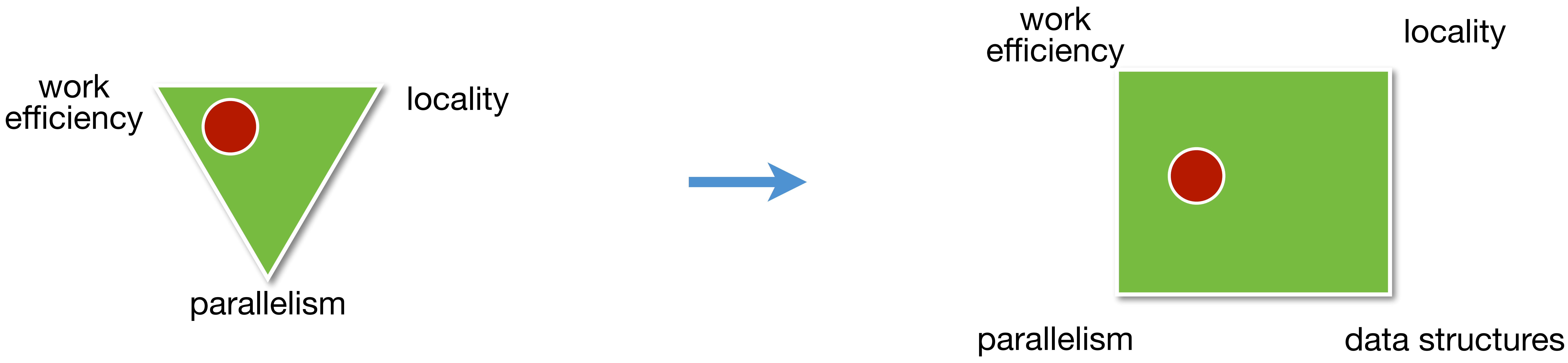
On tensors

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

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)

(0,1)
(2,3)
(0,5)
(5,5)
(7,5)

Relation (rows)

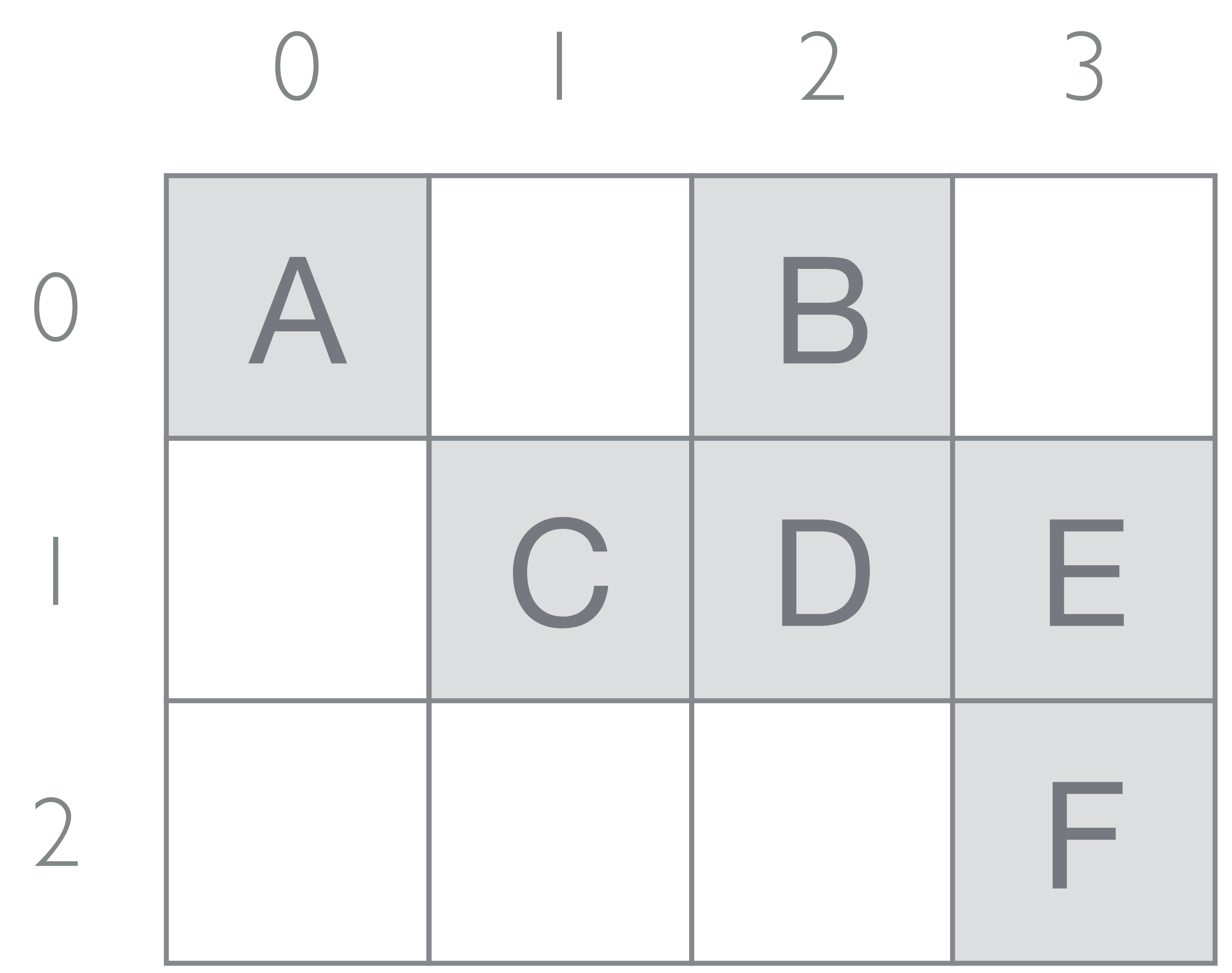
(Harry,CS)
(Sally,EE)
(George,CS)
(Mary,ME)
(Rita,CS)

Graph (edges)

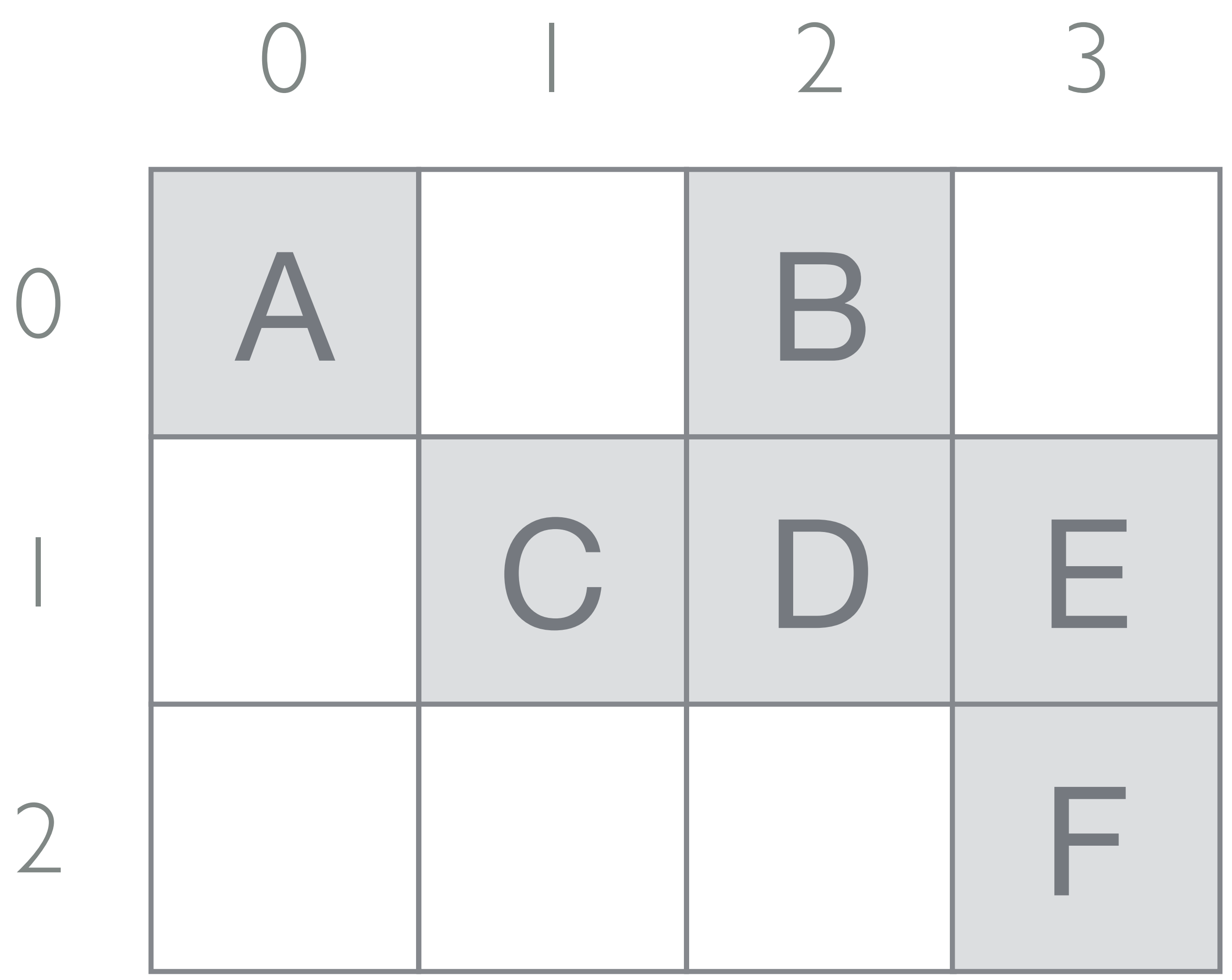
(v_1,v_5)
(v_4,v_3)
(v_5,v_3)
(v_3,v_1)
(v_3,v_5)

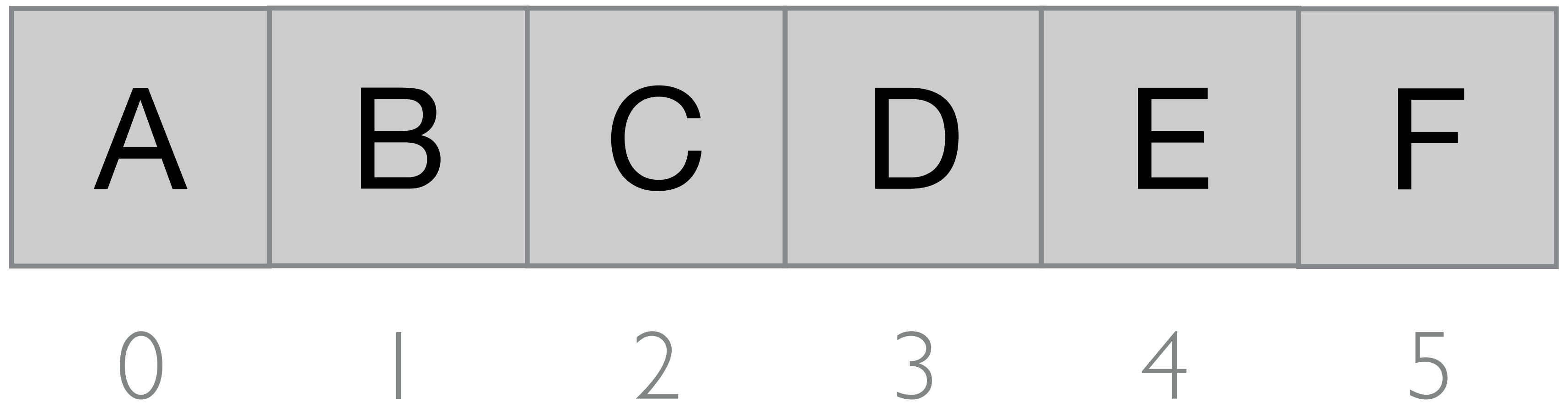
Values may be attached to these coordinates: e.g., nonzero values, edge attributes

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

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

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

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

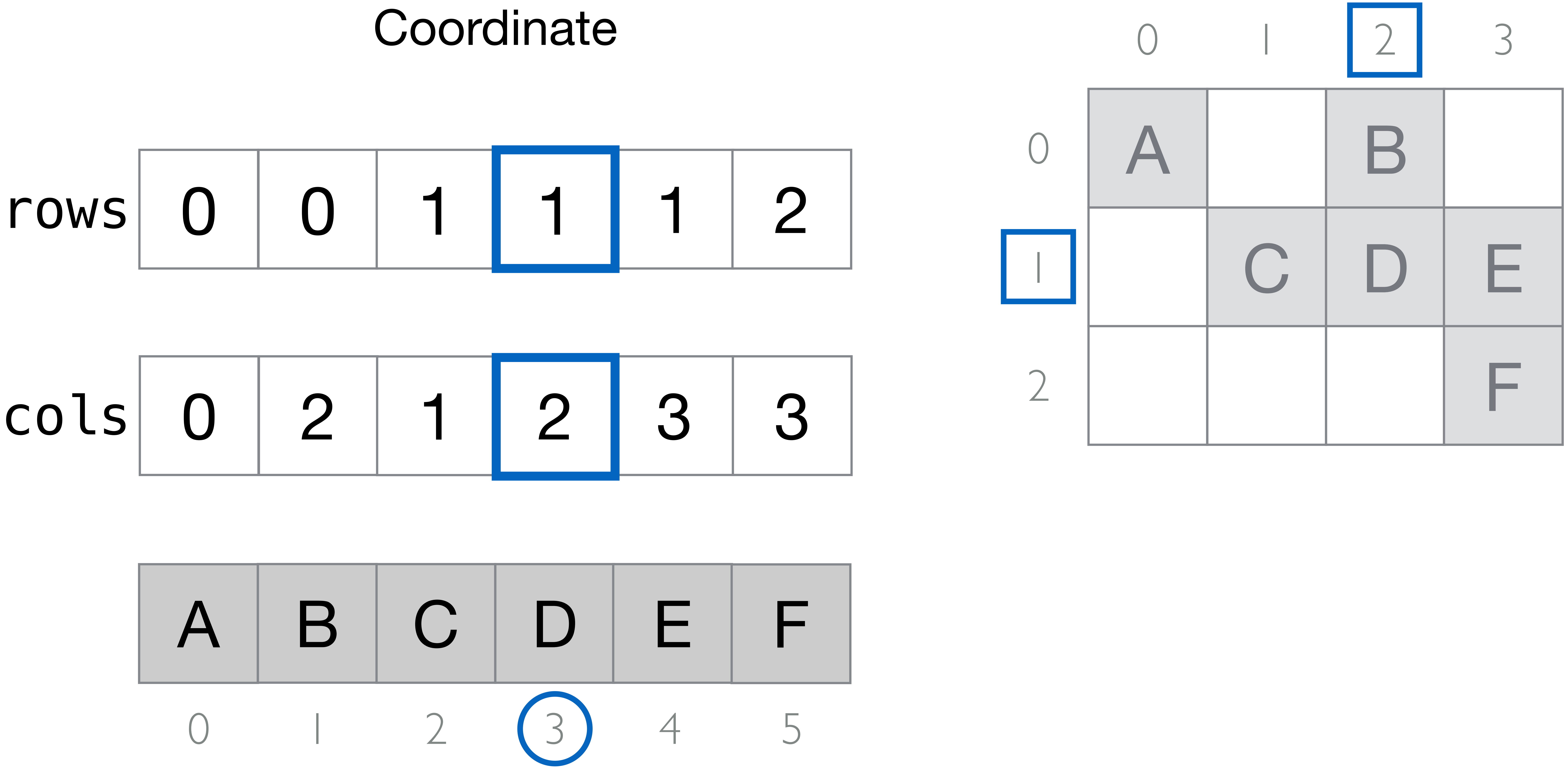
Hierarchically compressed data structures (tries) reduce the number of values that need to be stored

Coordinate

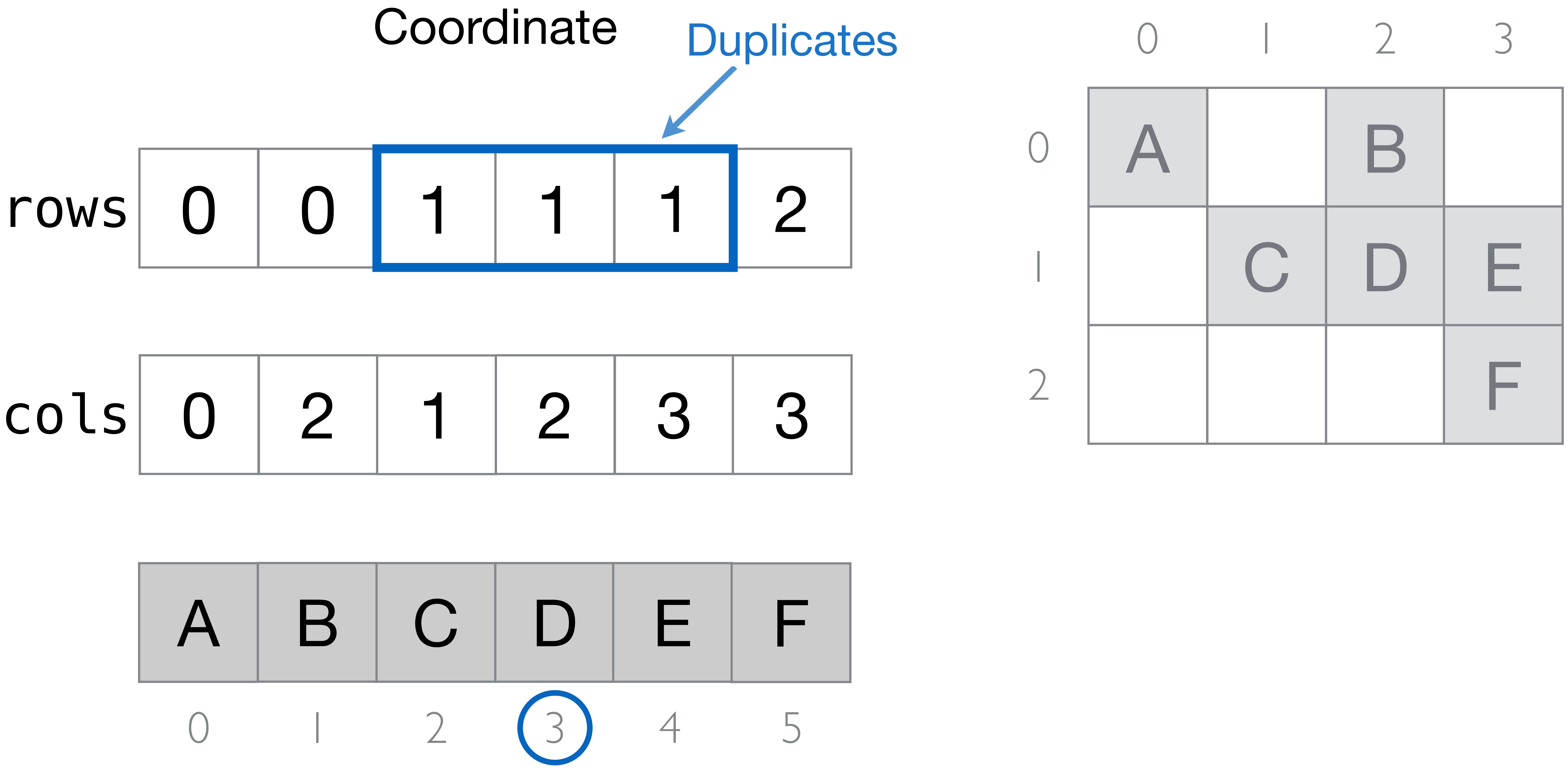
rows	0	0	1	1	1	2
cols	0	2	1	2	3	3
	A	B	C	D	E	F
	0	1	2	3	4	5

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

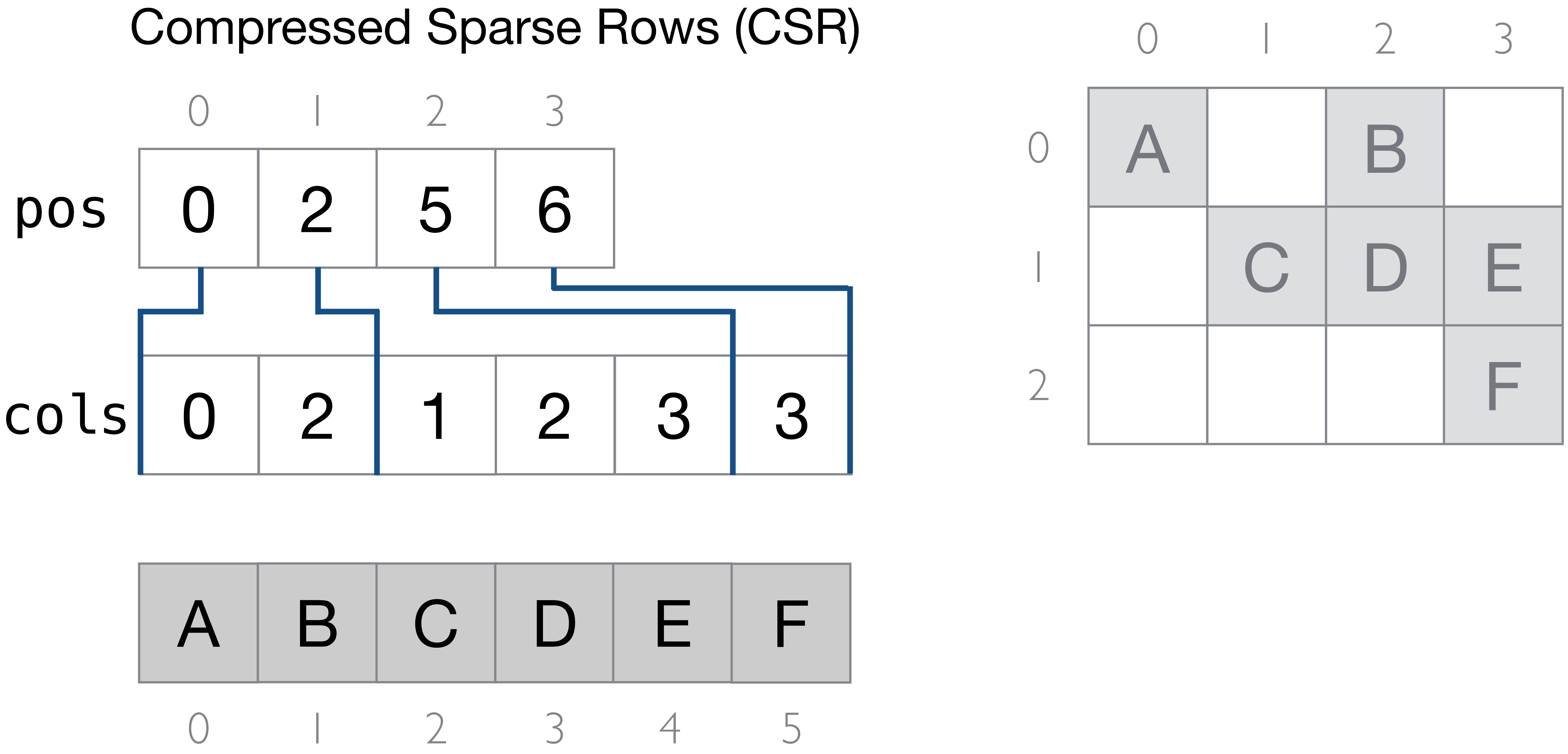
Hierarchically compressed data structures (tries) reduce the number of values that need to be stored



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

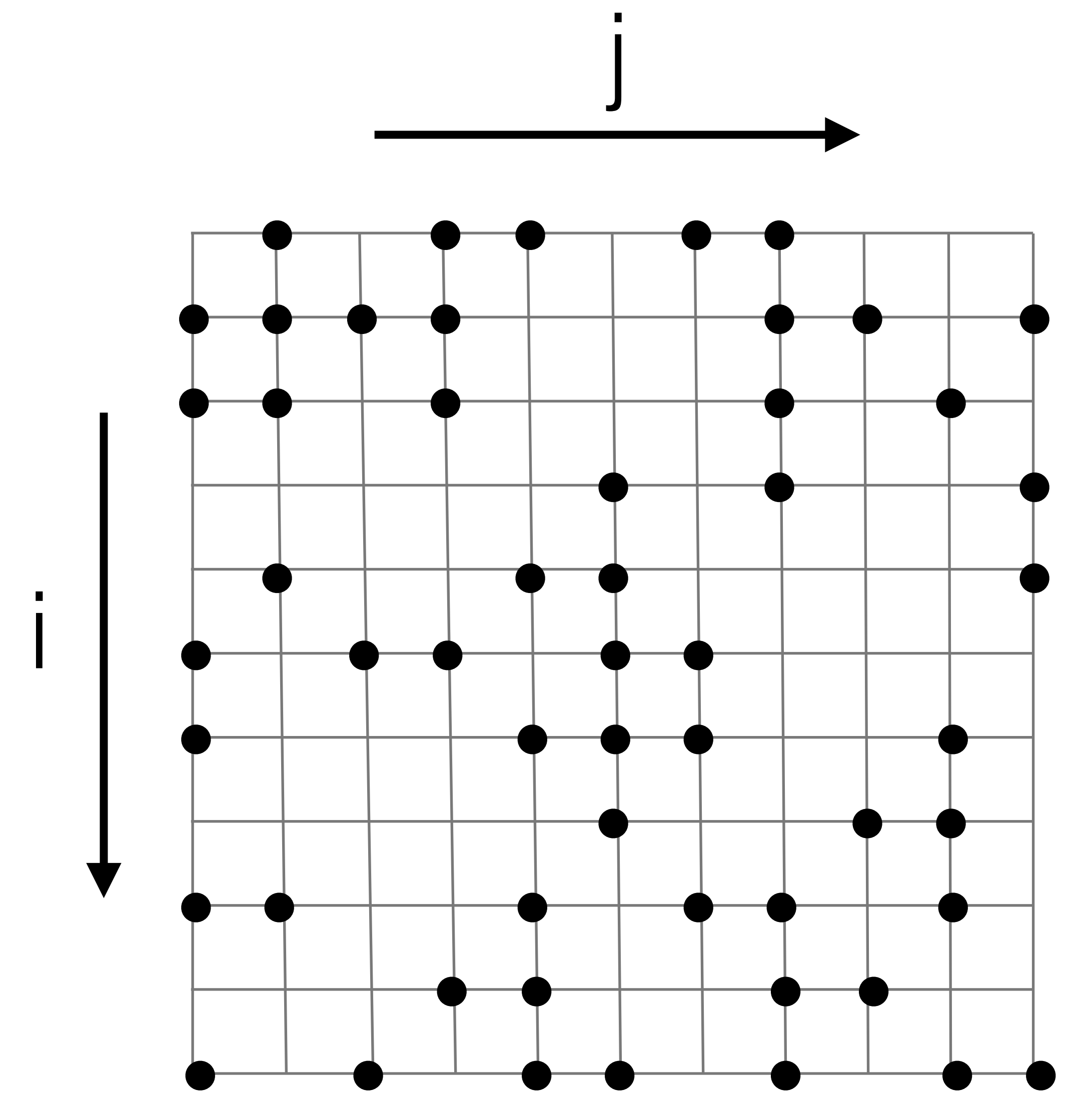


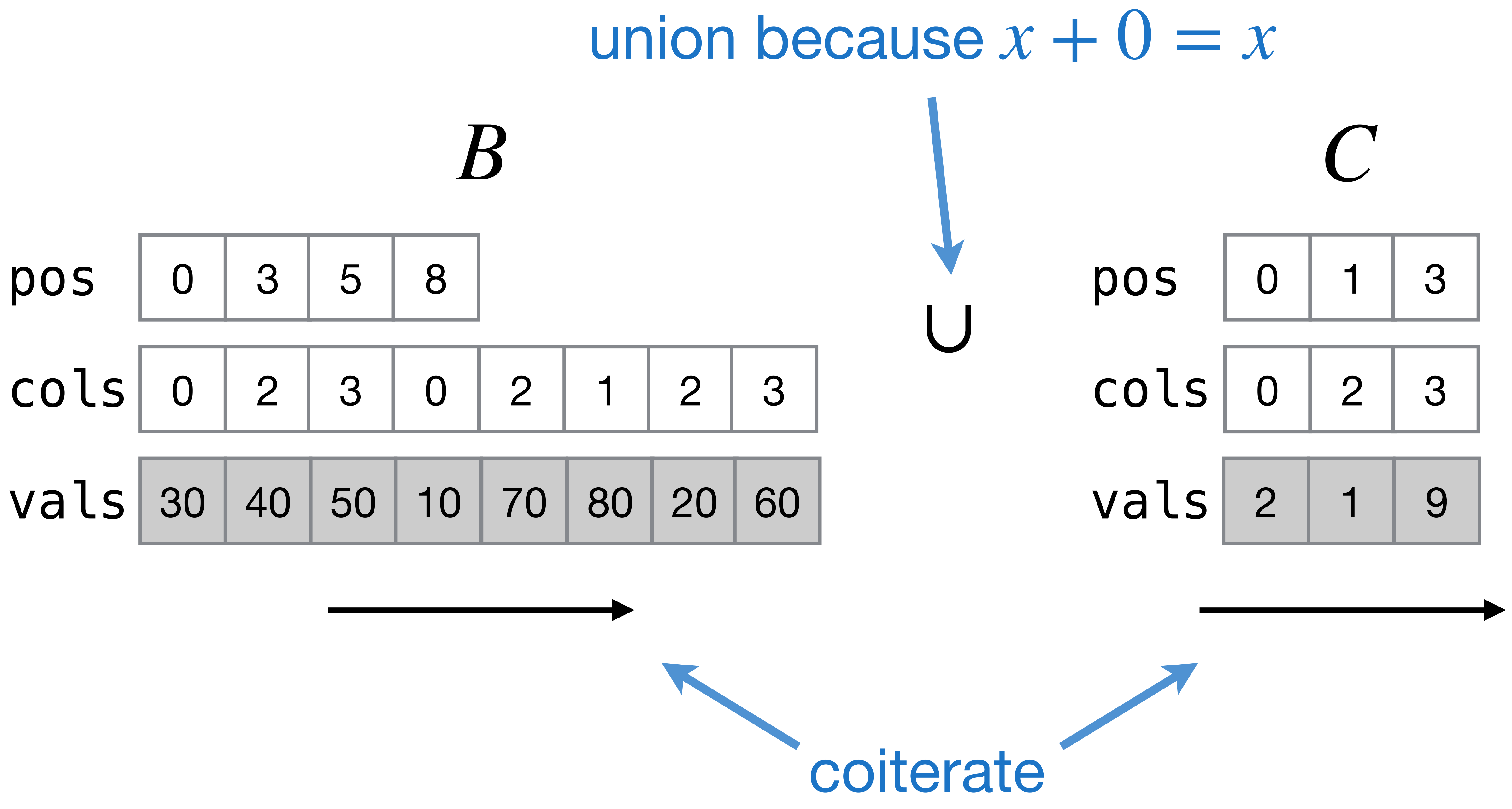
Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra: $A = B + C$

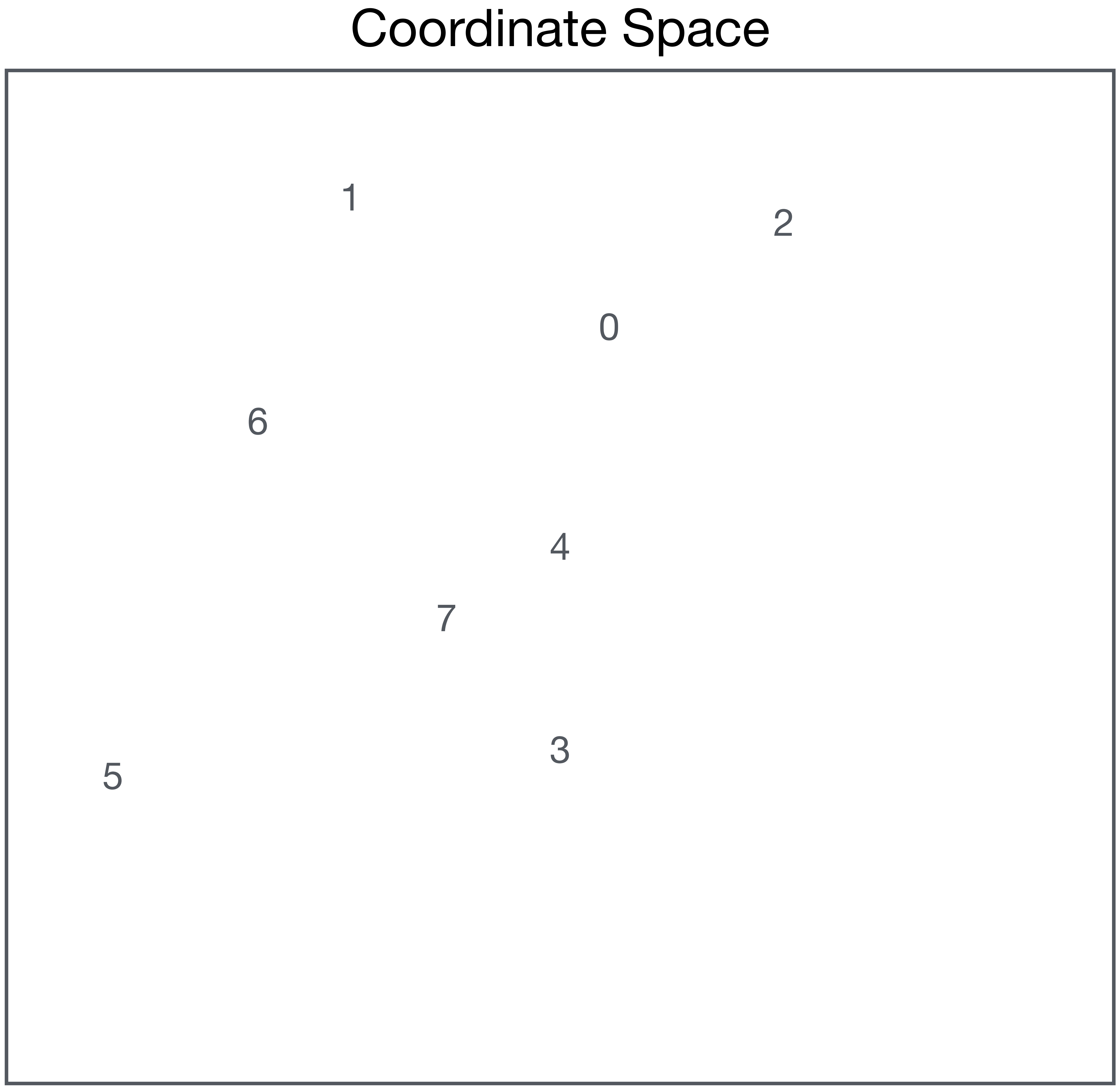
Tensor Index Notation: $A_{ij} = B_{ij} + C_{ij}$

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

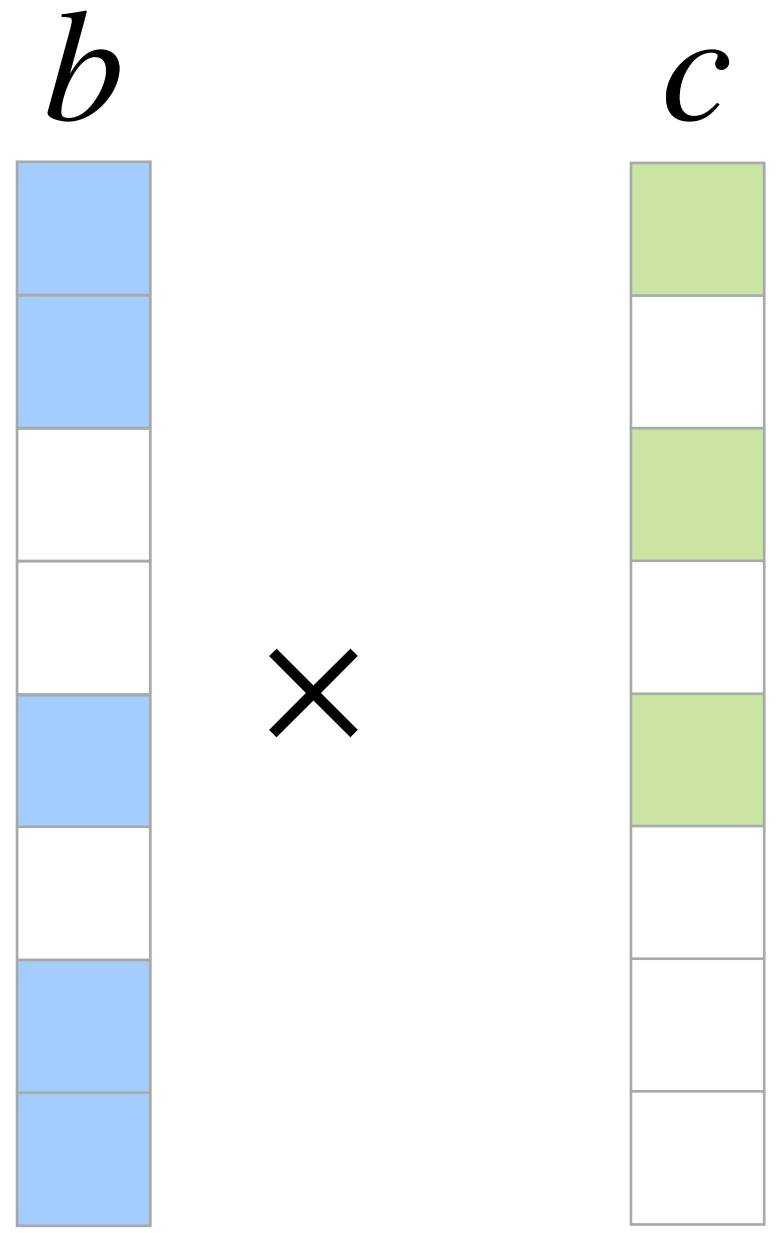




Merged coiteration

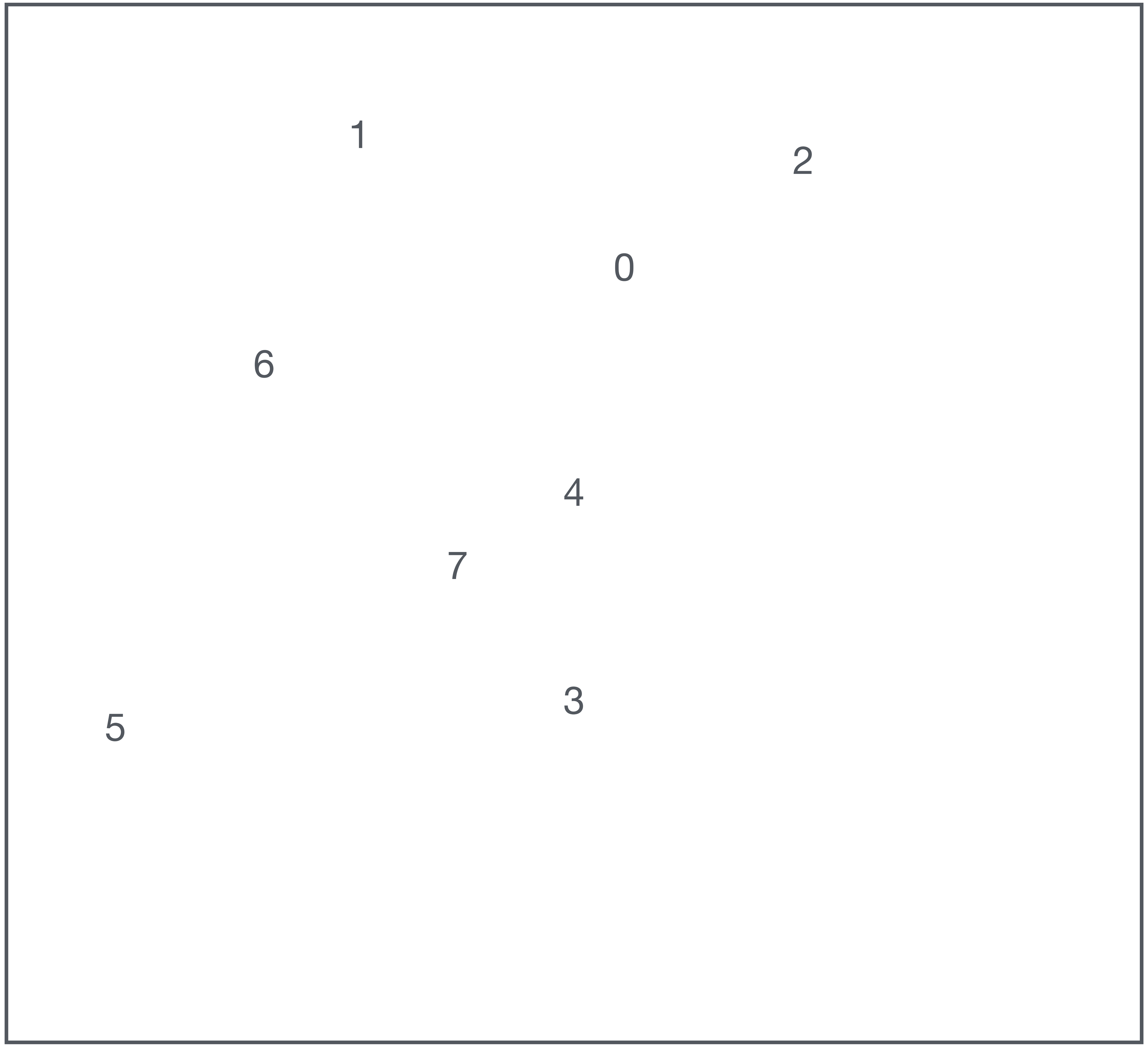


$$a_i = b_i c_i$$

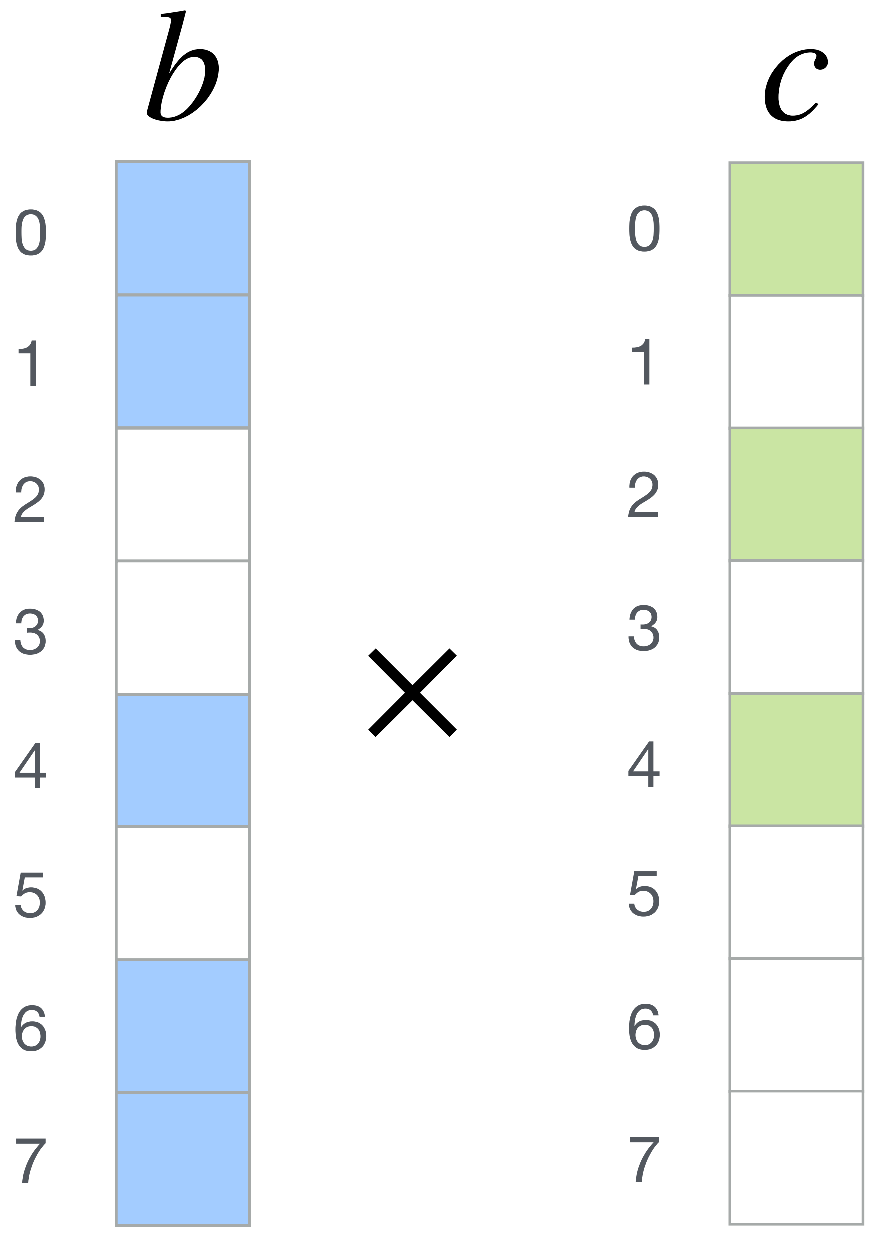


Merged coiteration

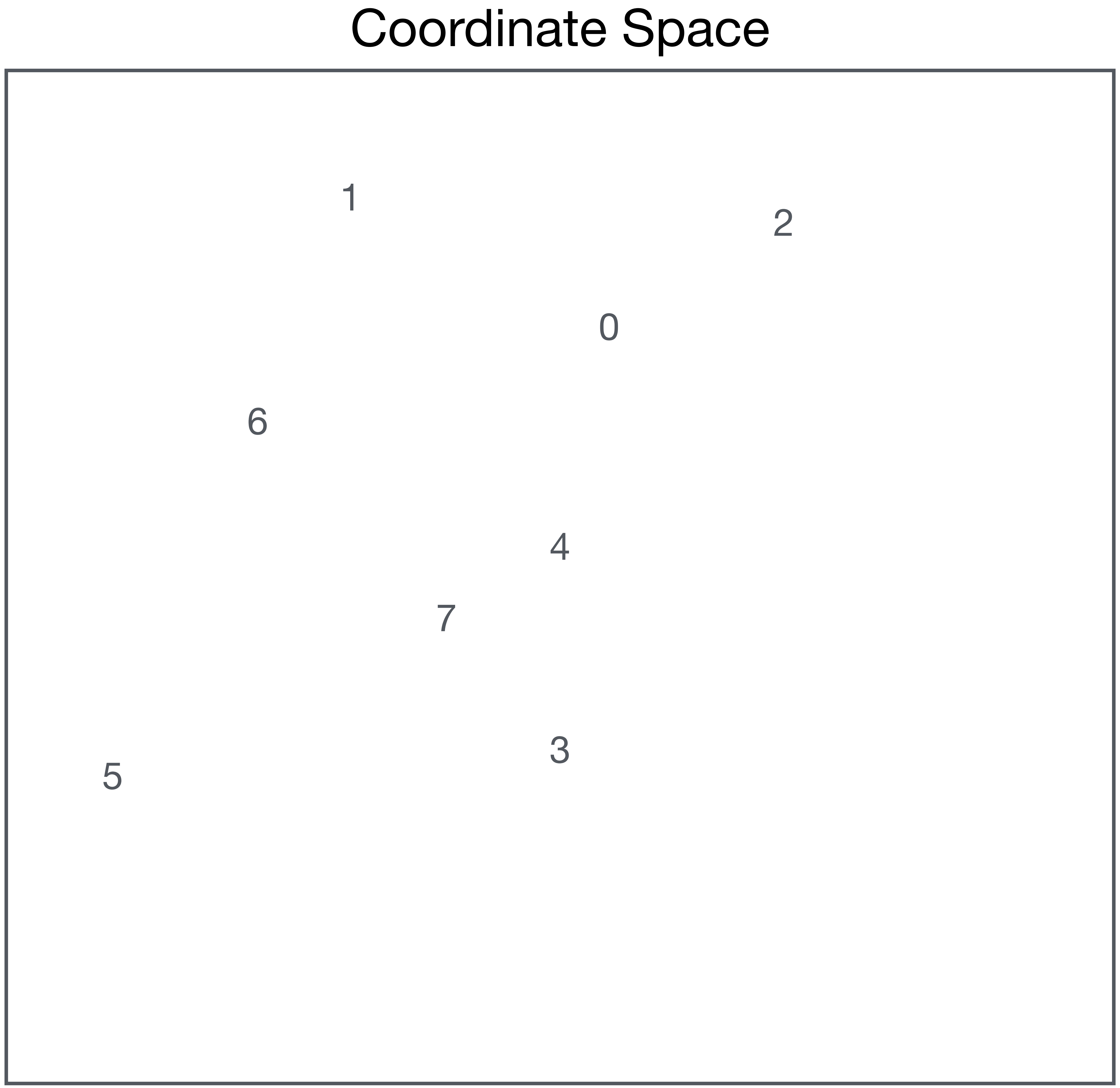
Coordinate Space



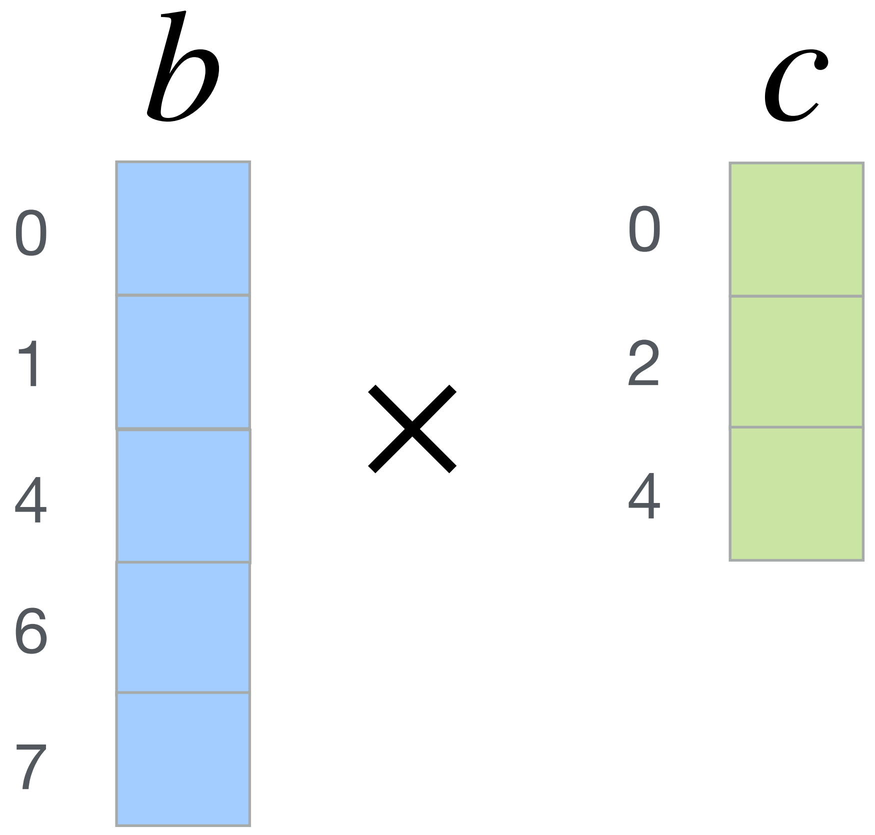
$$a_i = b_i c_i$$



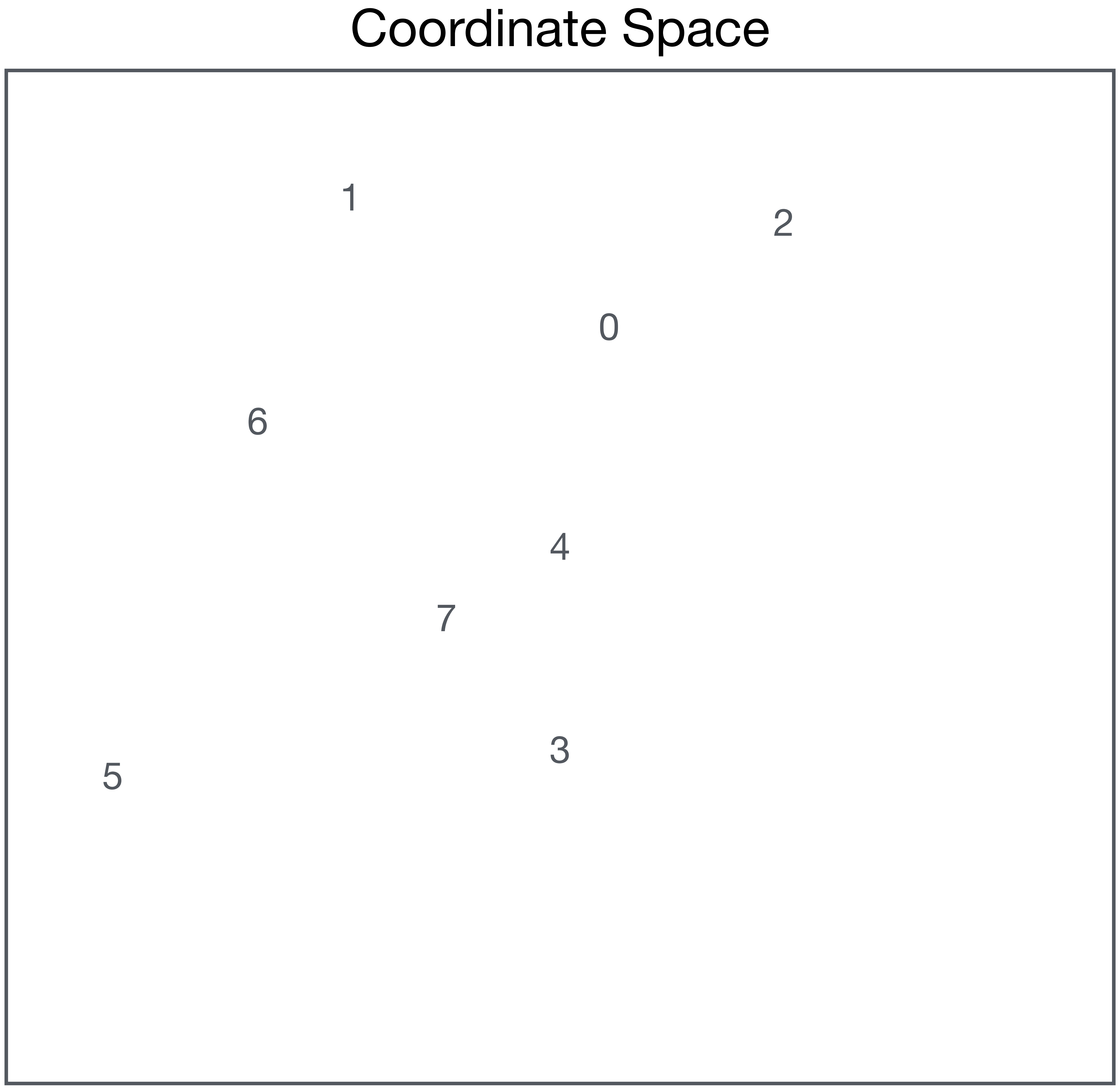
Merged coiteration



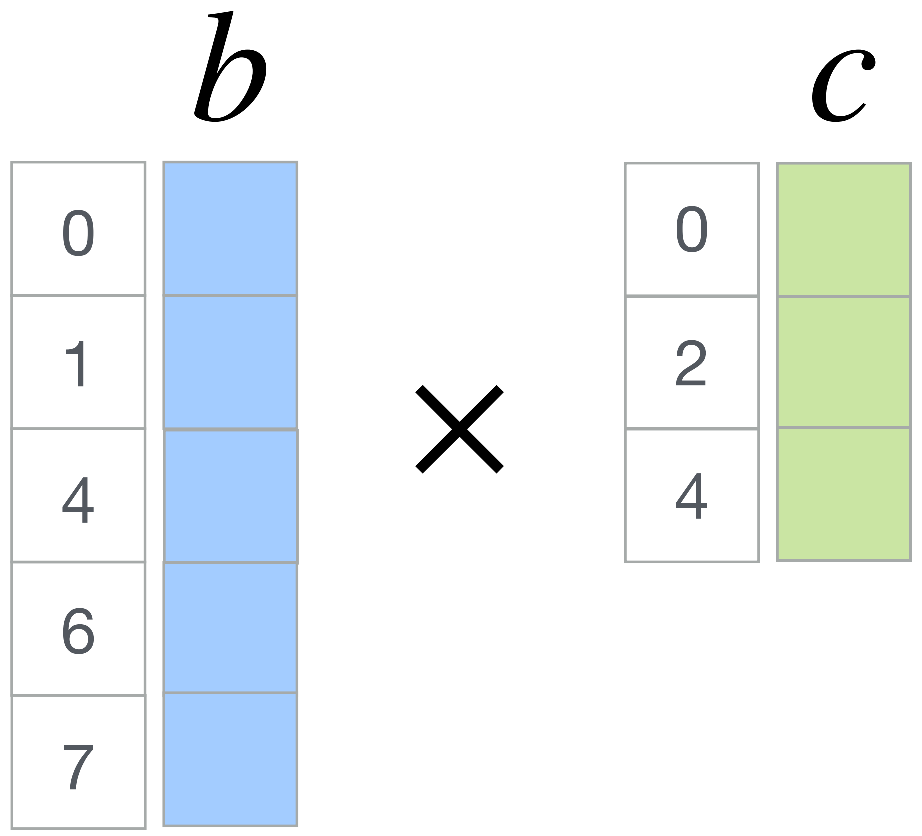
$$a_i = b_i c_i$$



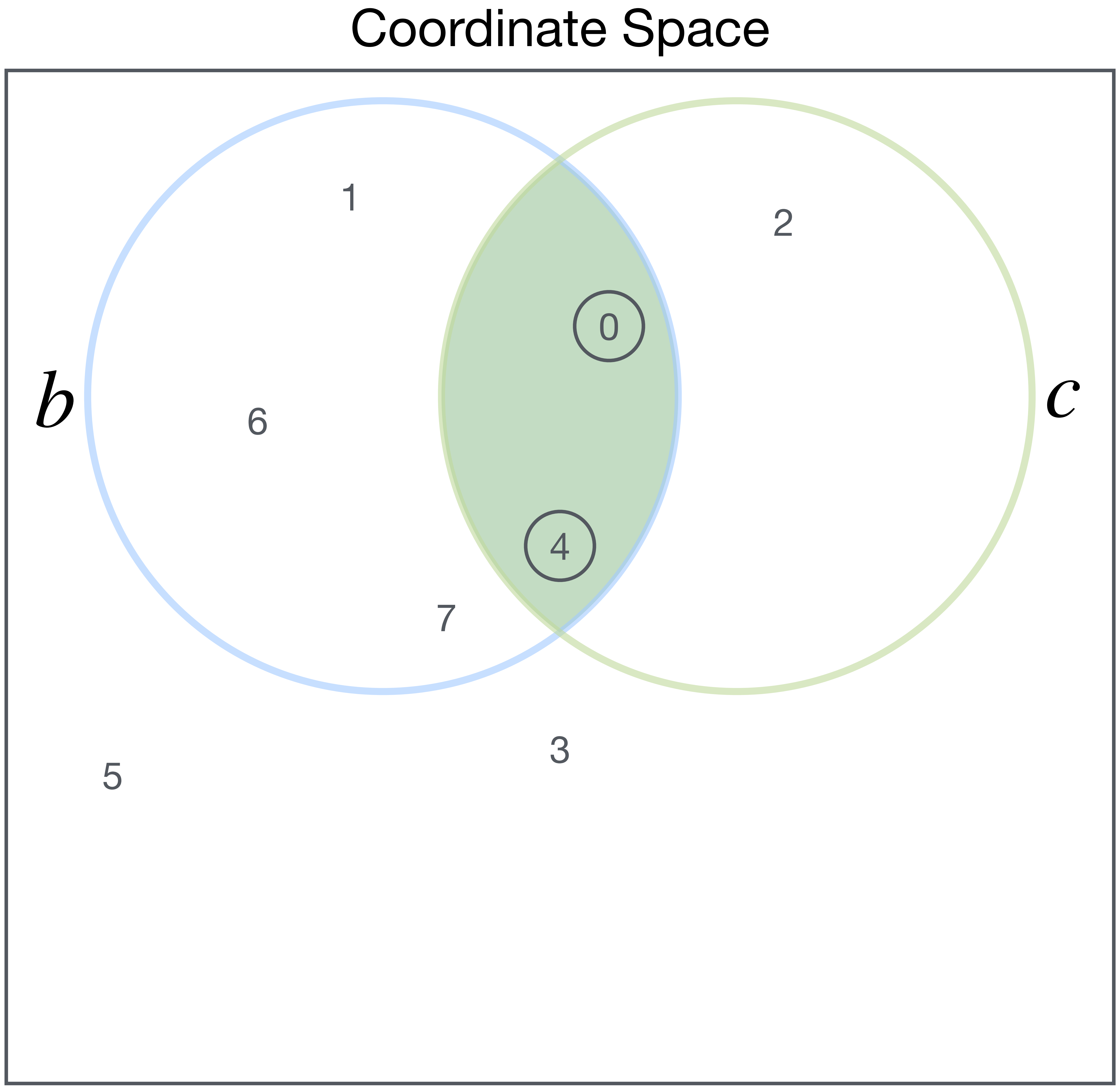
Merged coiteration



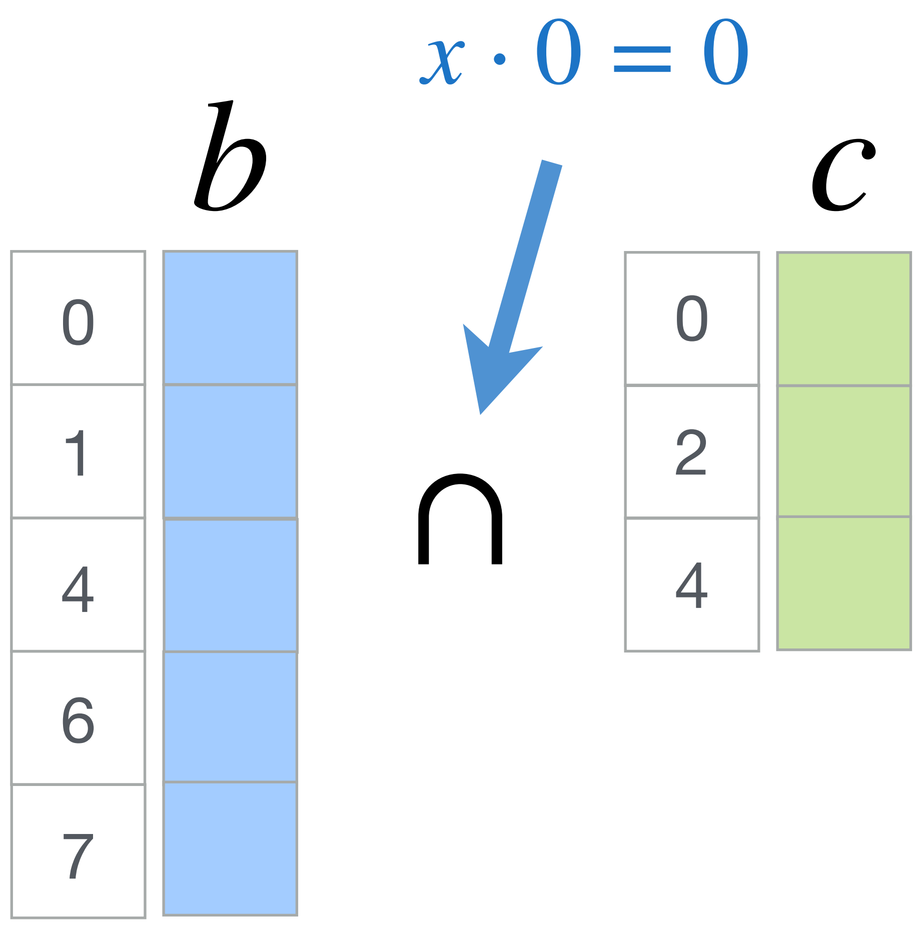
$$a_i = b_i c_i$$



Merged coiteration

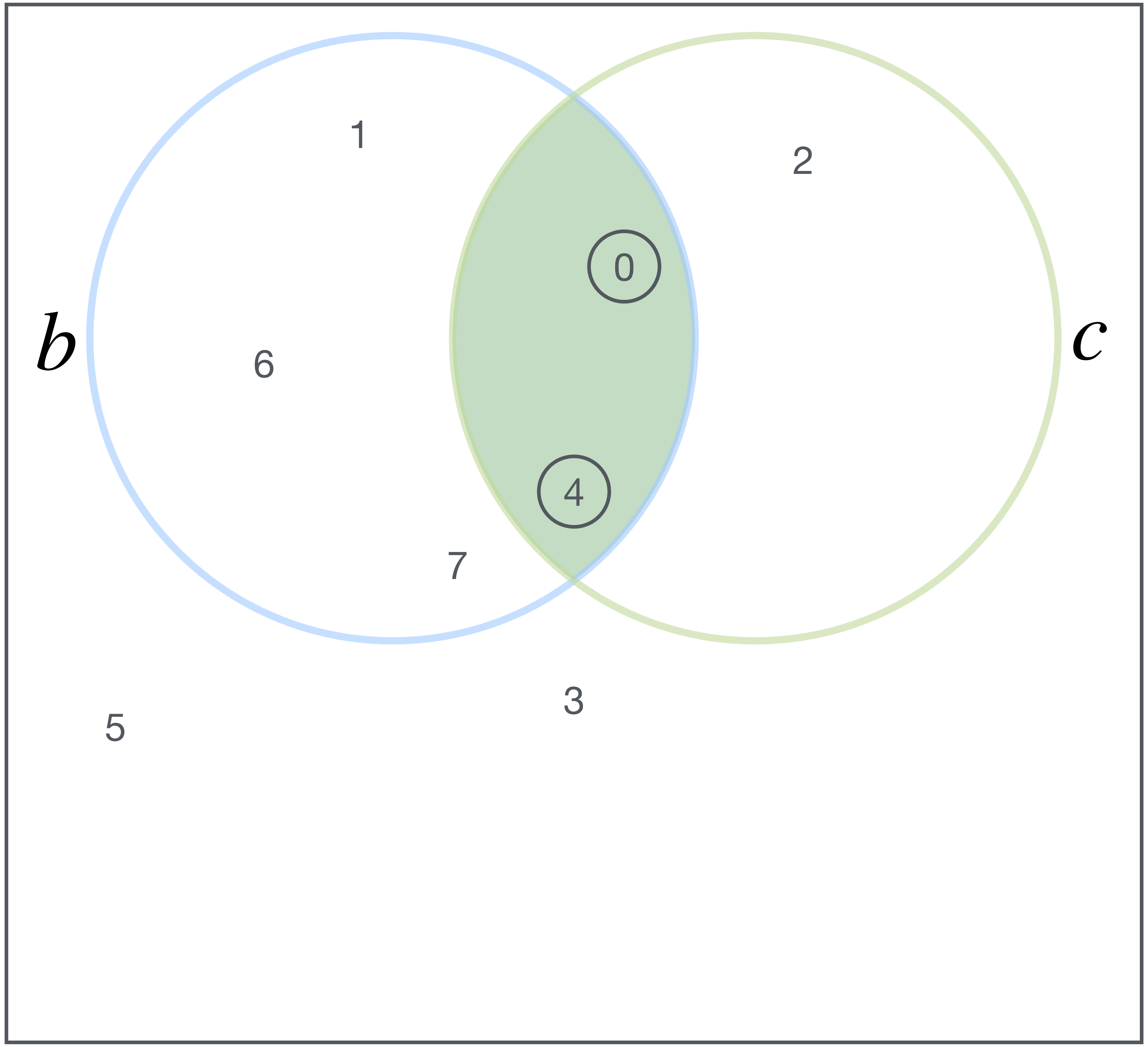


$$a_i = b_i c_i$$

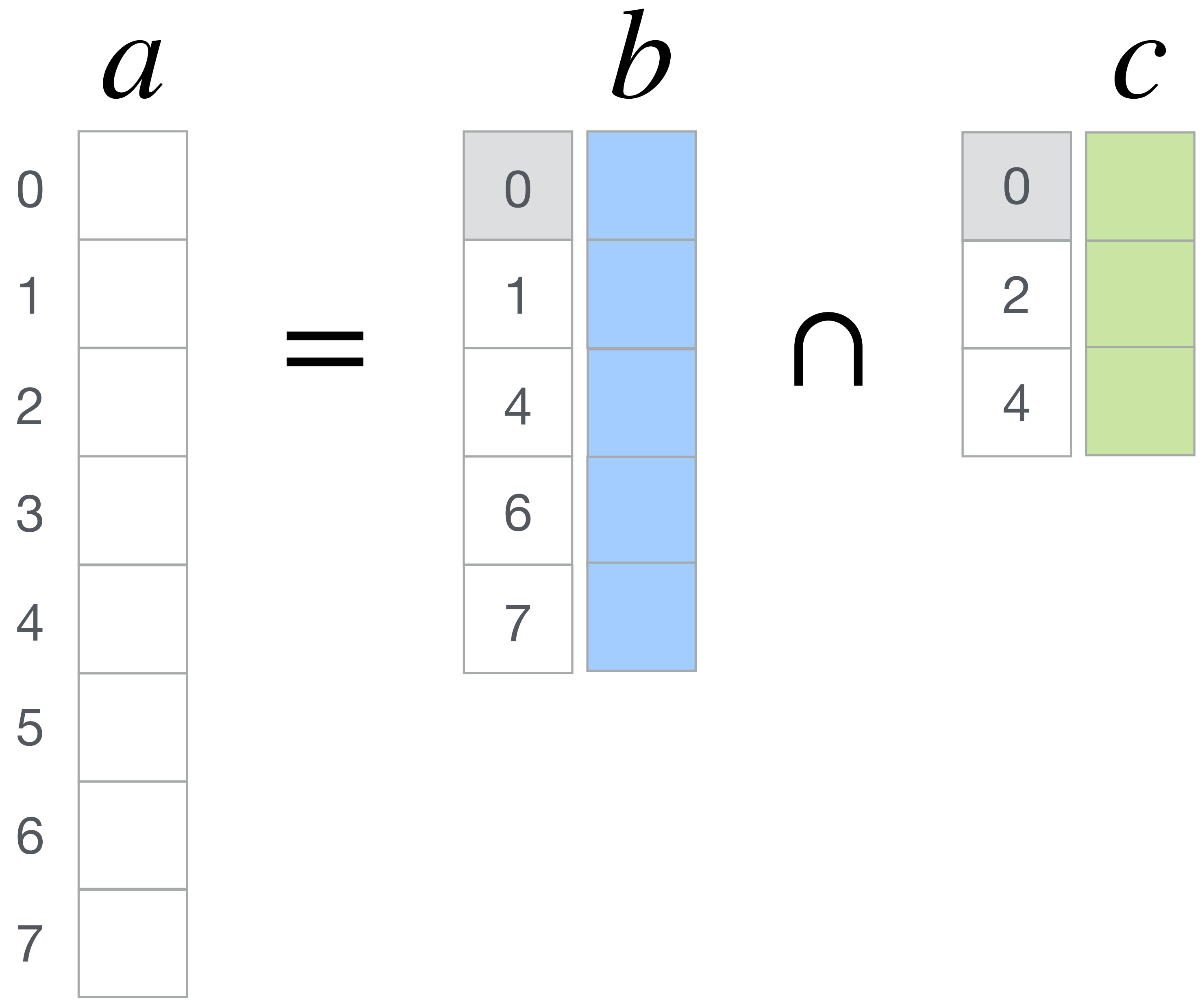


Merged coiteration

Coordinate Space

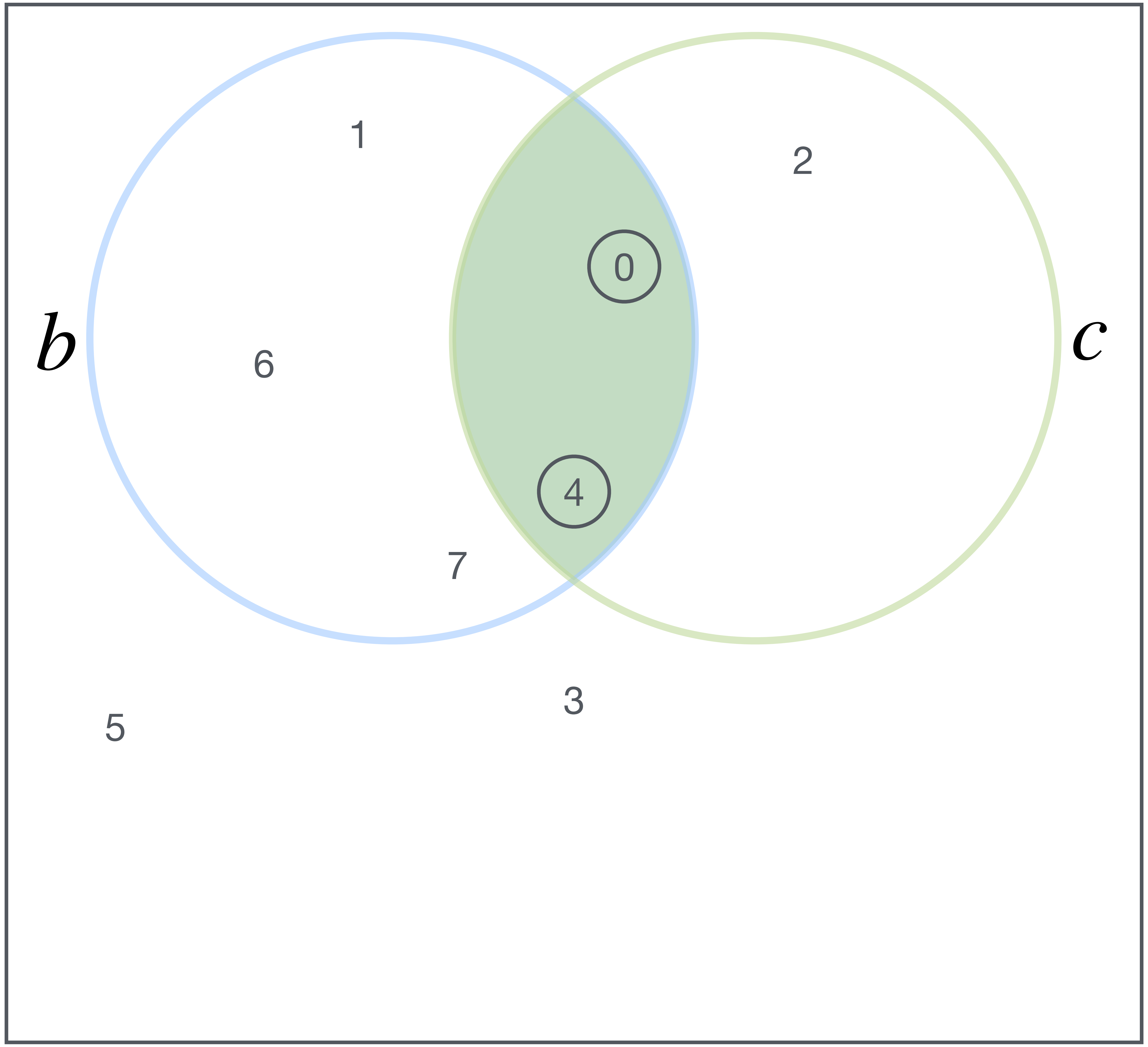


$$a_i = b_i c_i$$

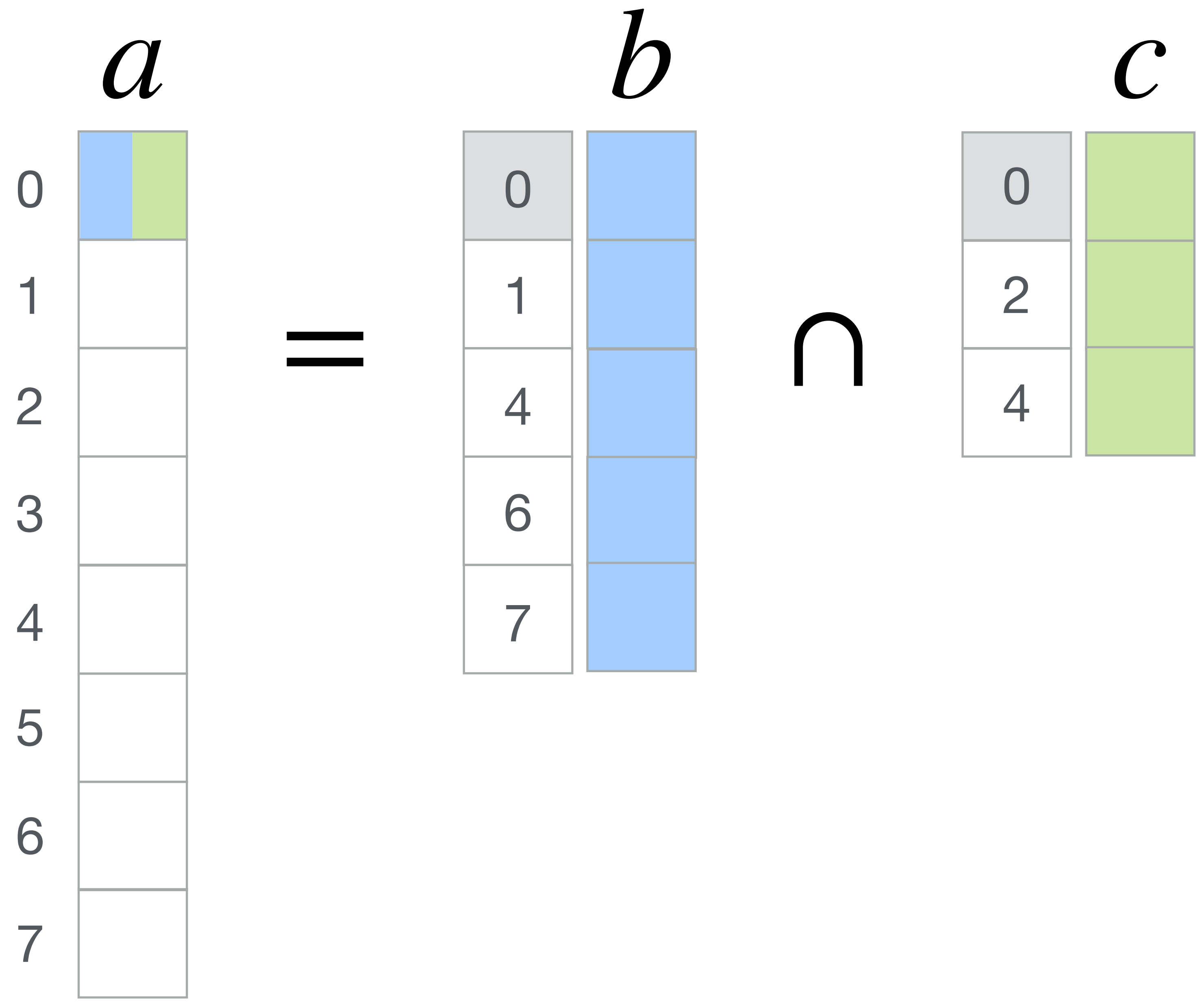


Merged coiteration

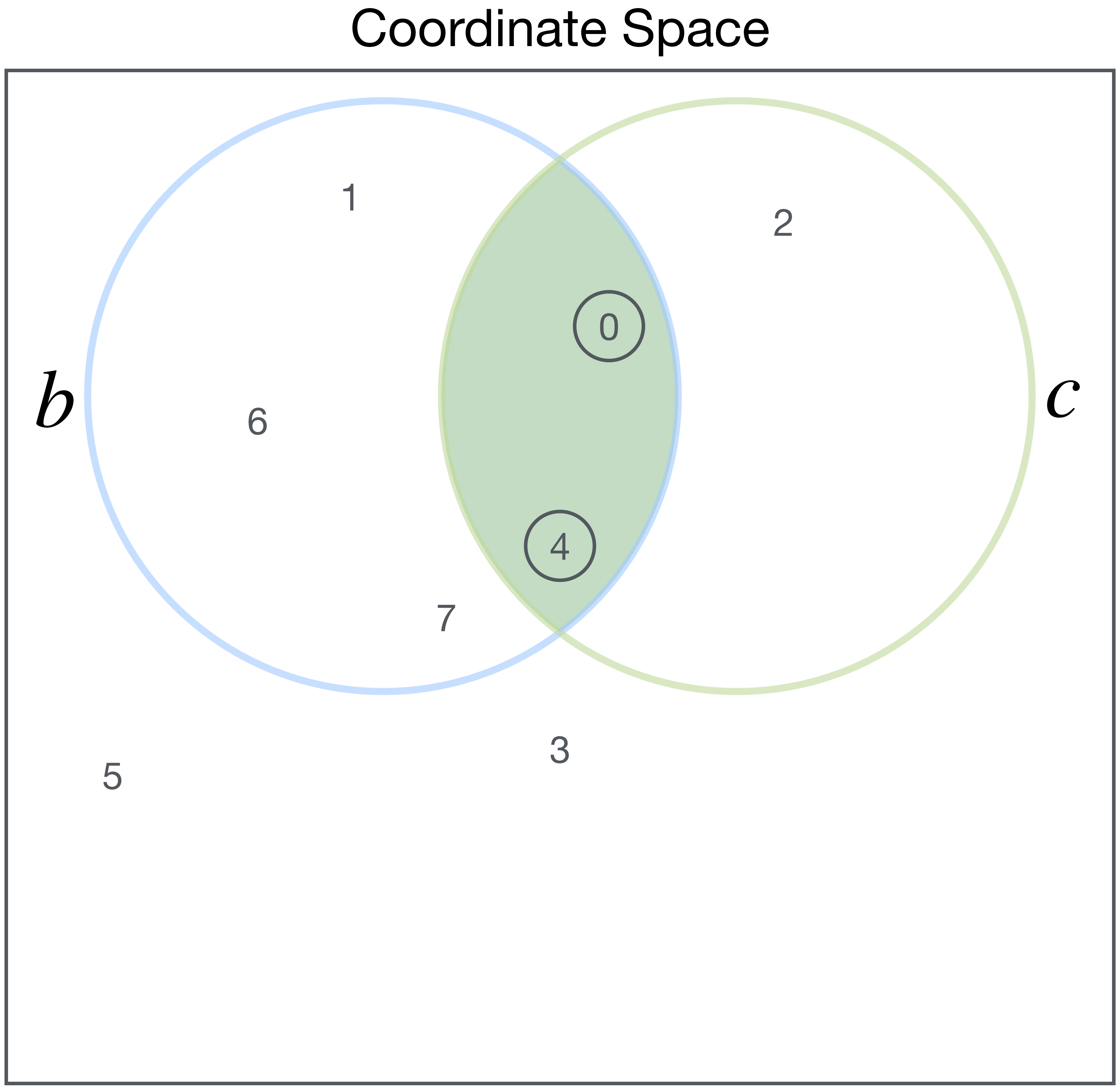
Coordinate Space



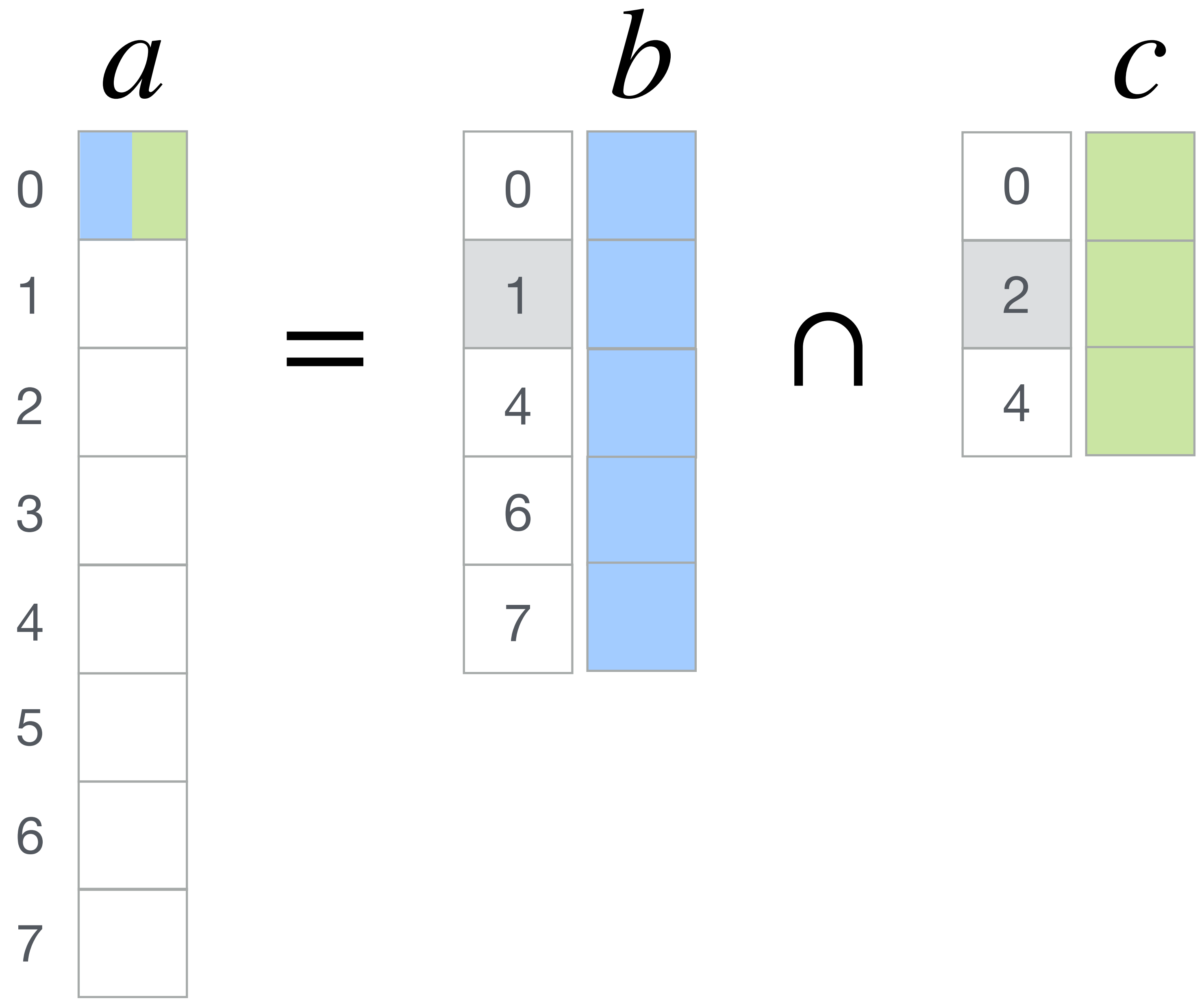
$$a_i = b_i c_i$$



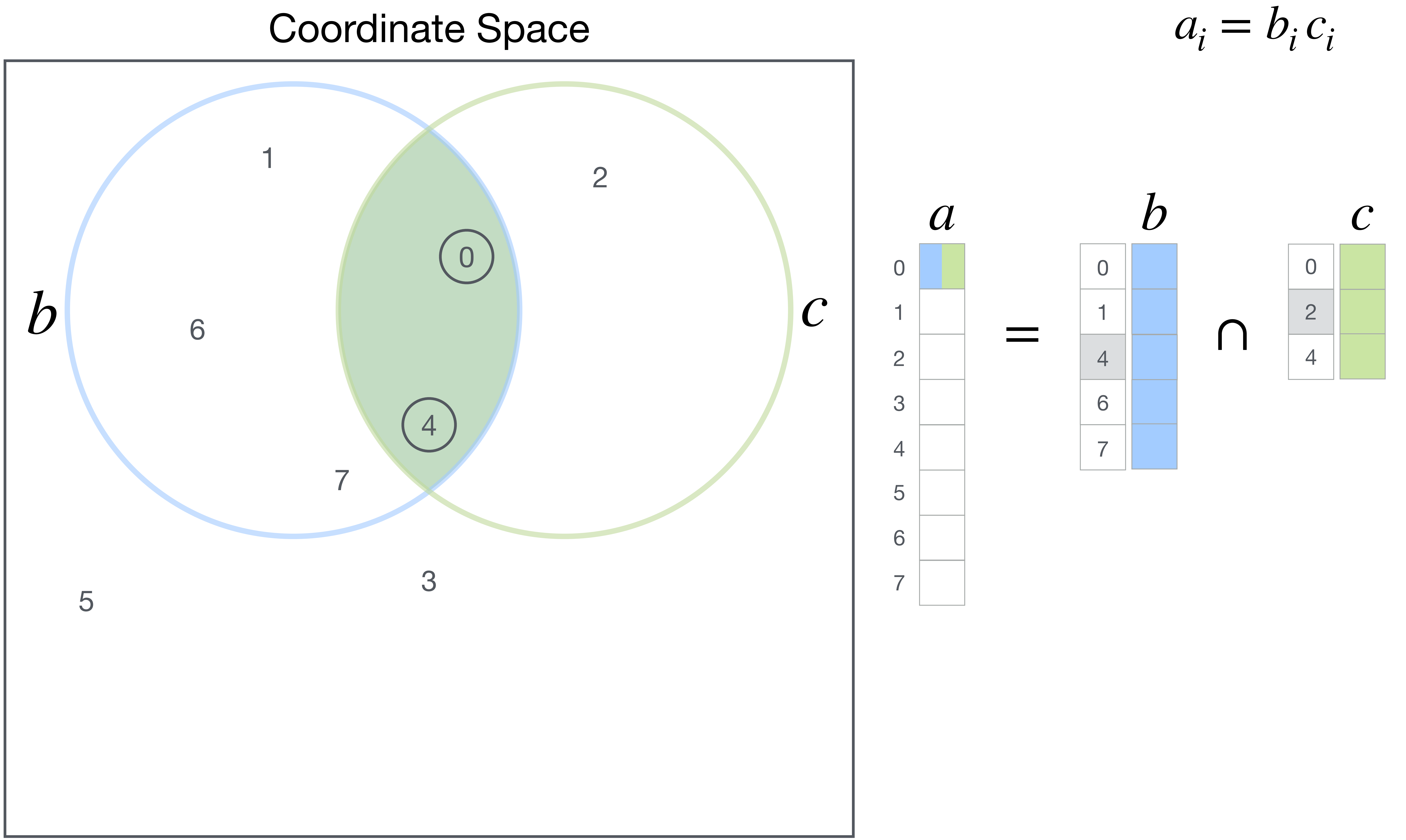
Merged coiteration



$$a_i = b_i c_i$$

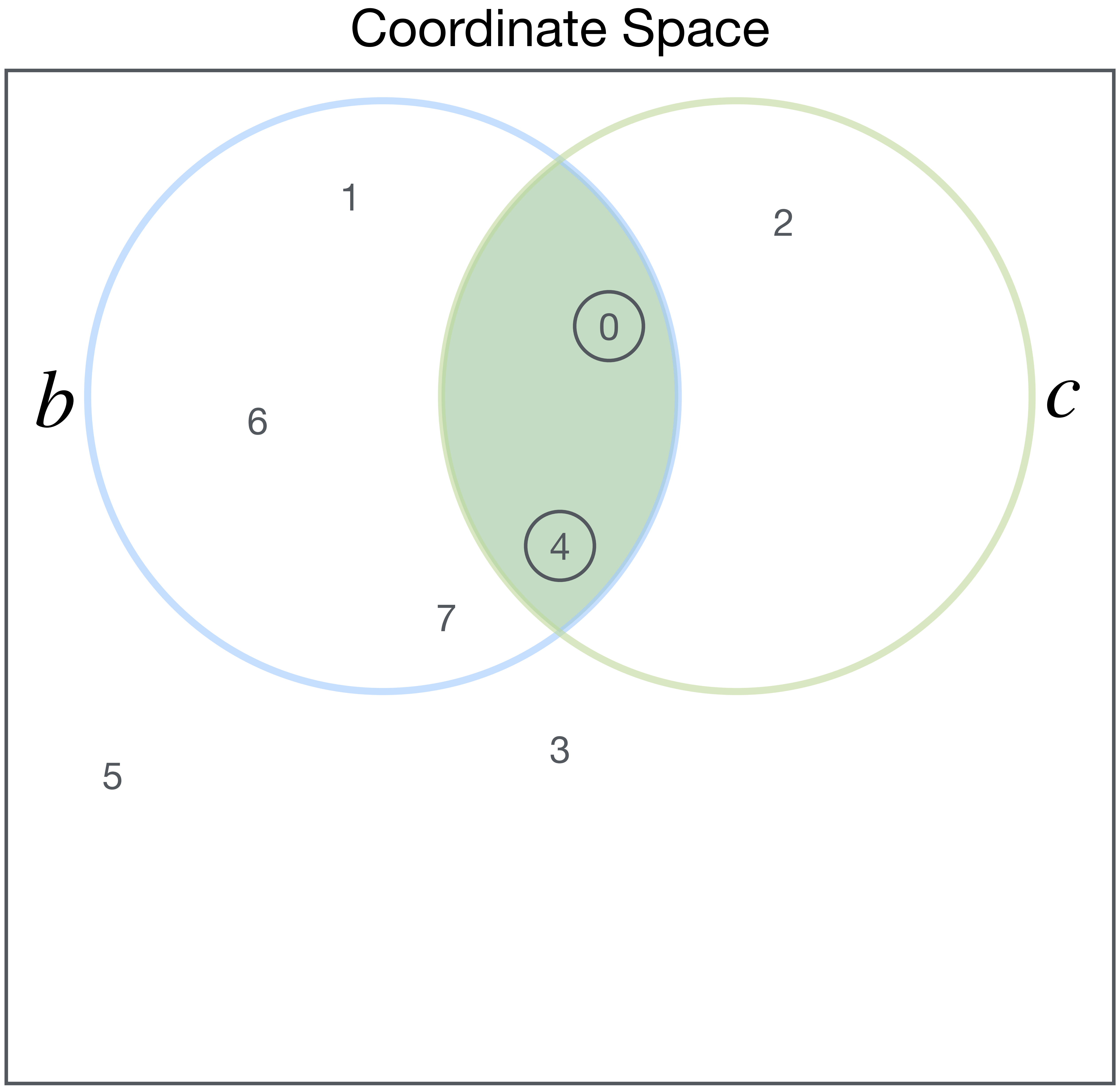


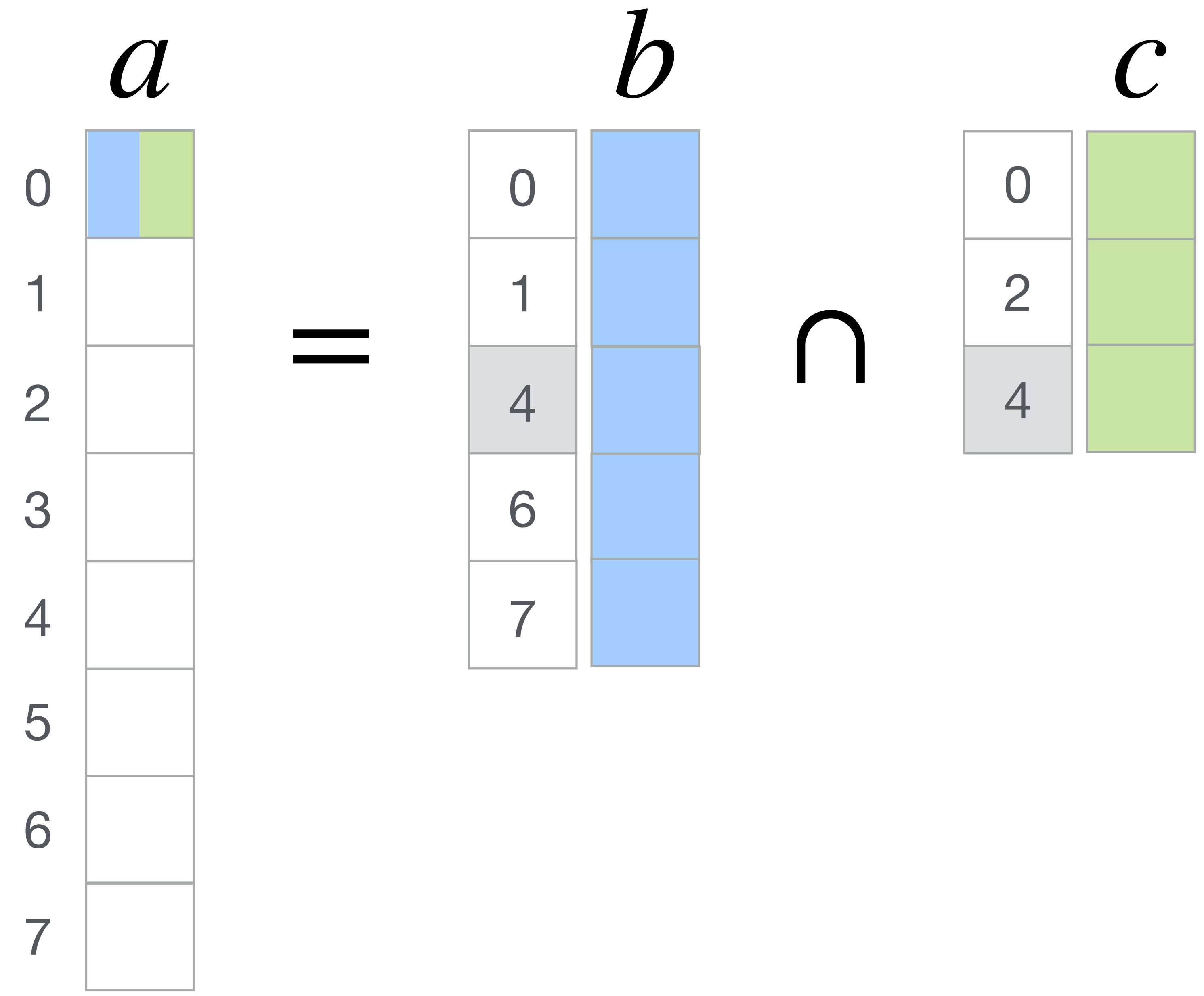
Merged coiteration



Merged coiteration

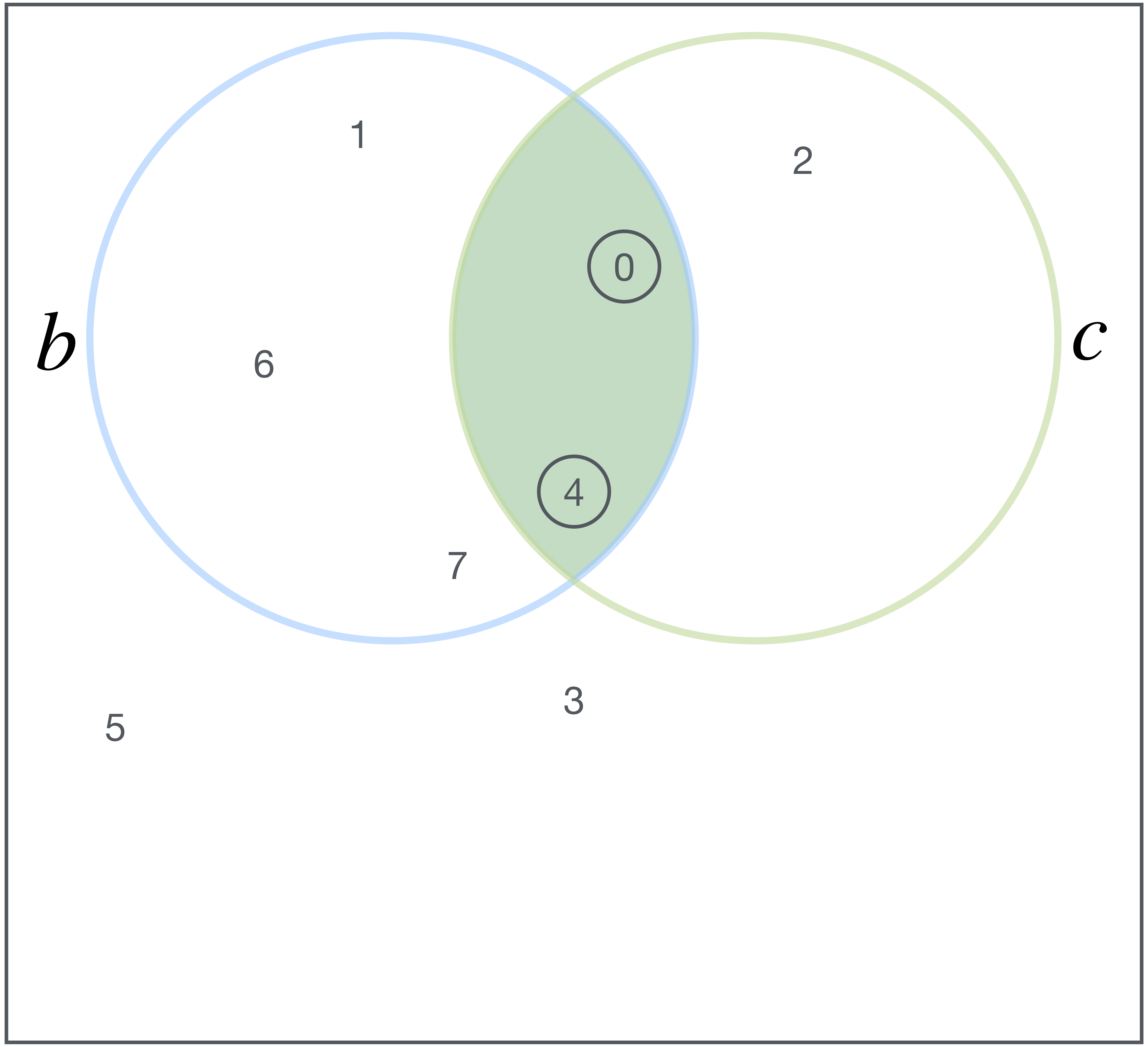
$$a_i = b_i c_i$$



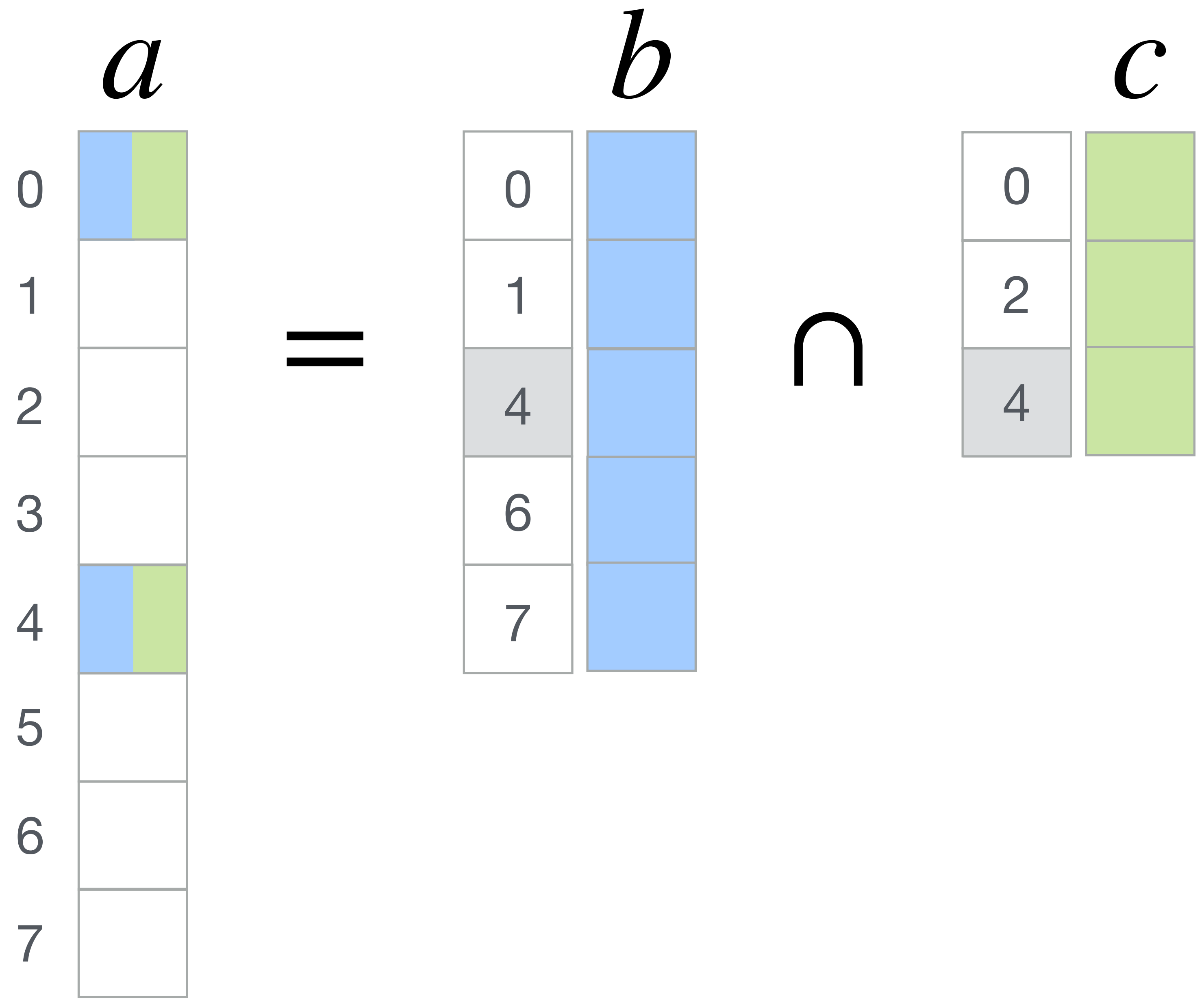


Merged coiteration

Coordinate Space

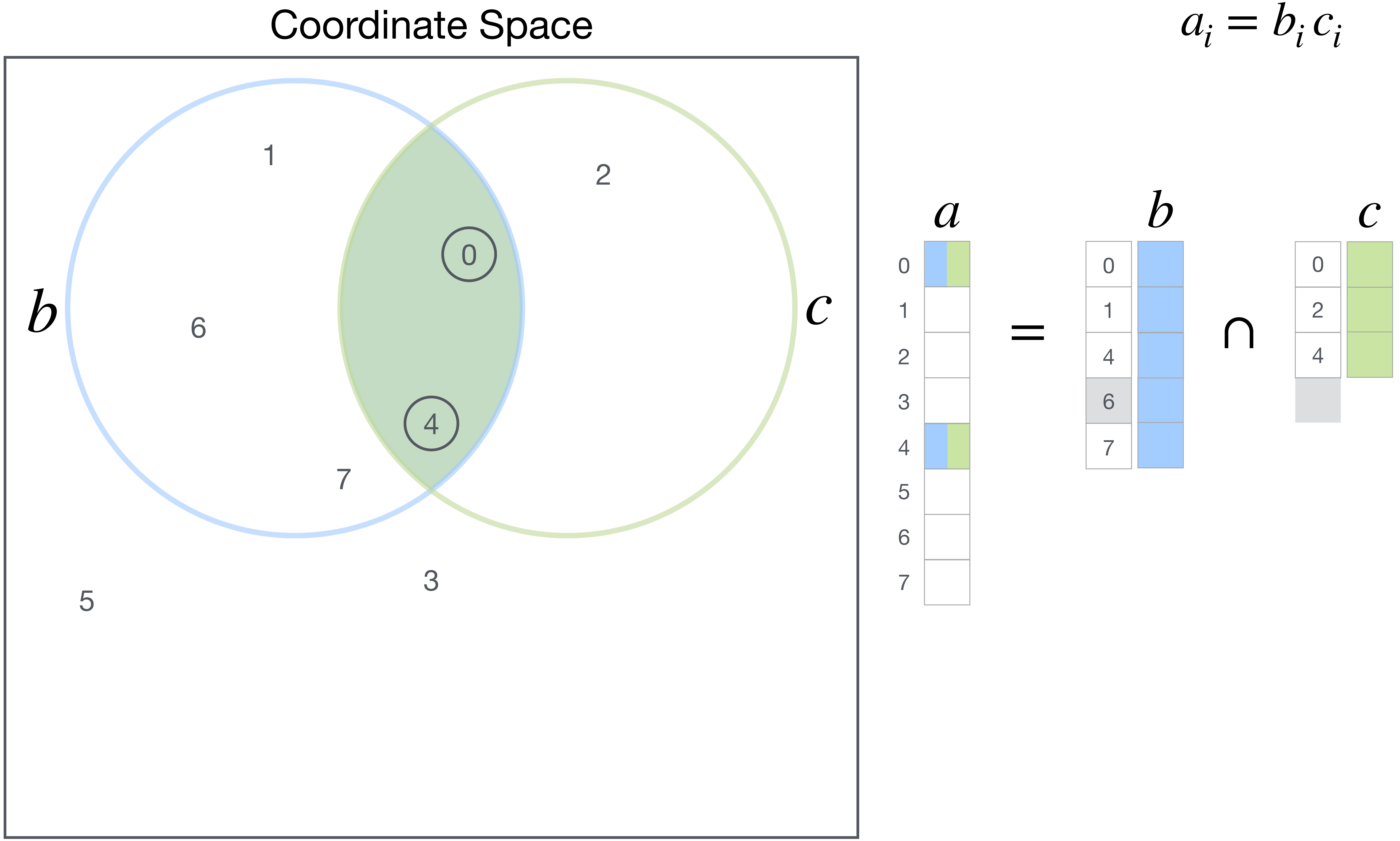


$$a_i = b_i c_i$$



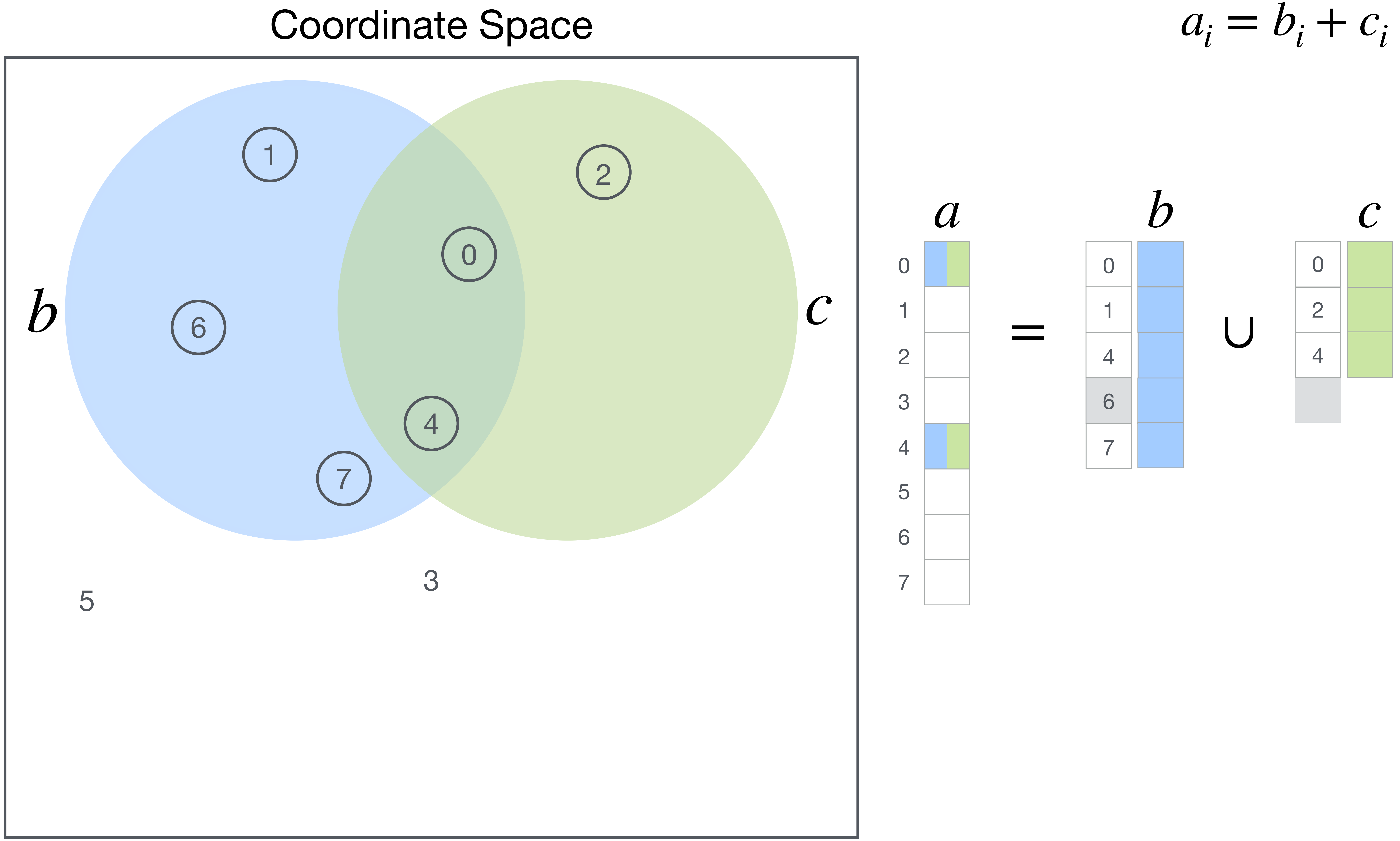
Merged coiteration

$$a_i = b_i c_i$$



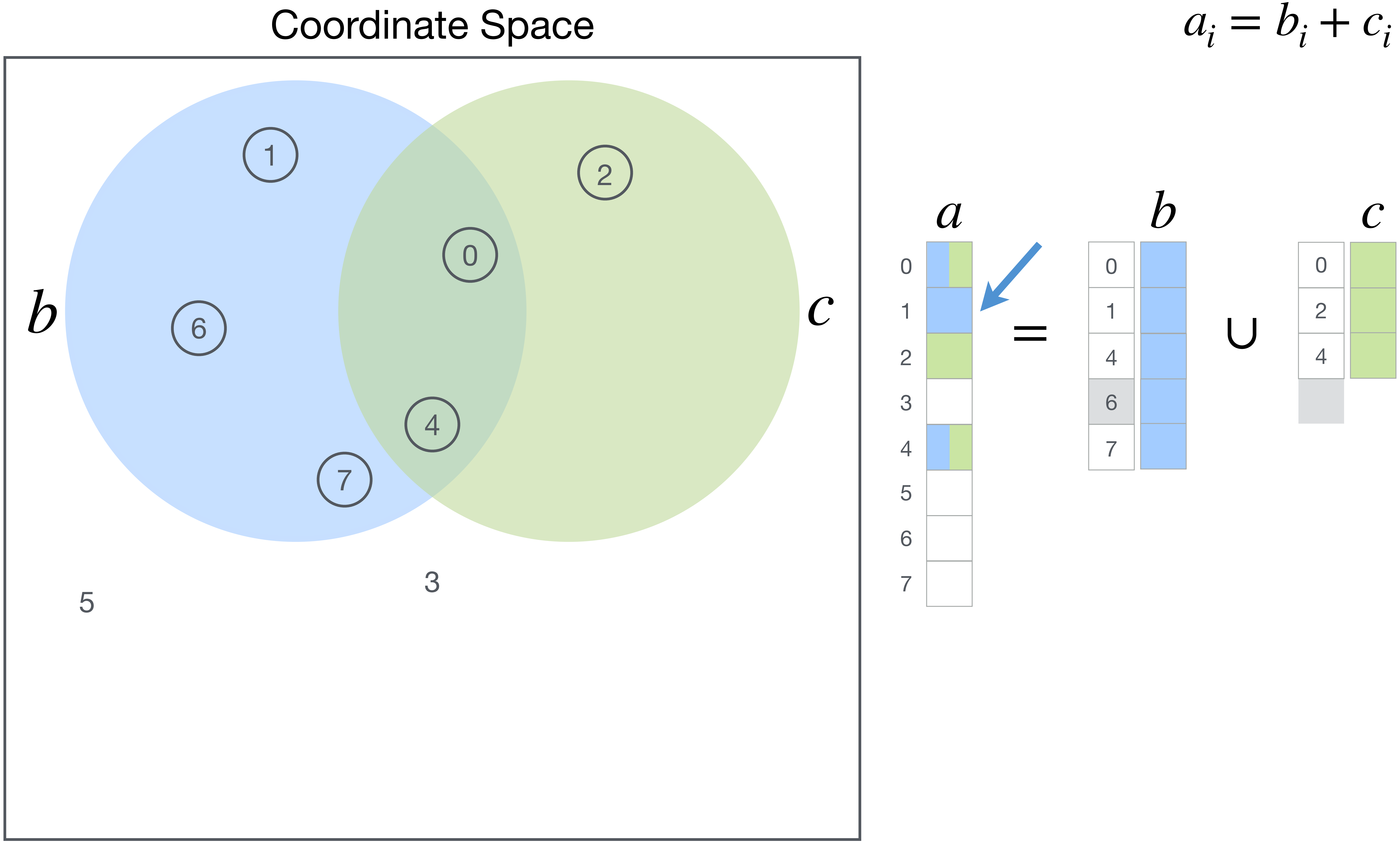
Merged coiteration

$$a_i = b_i + c_i$$



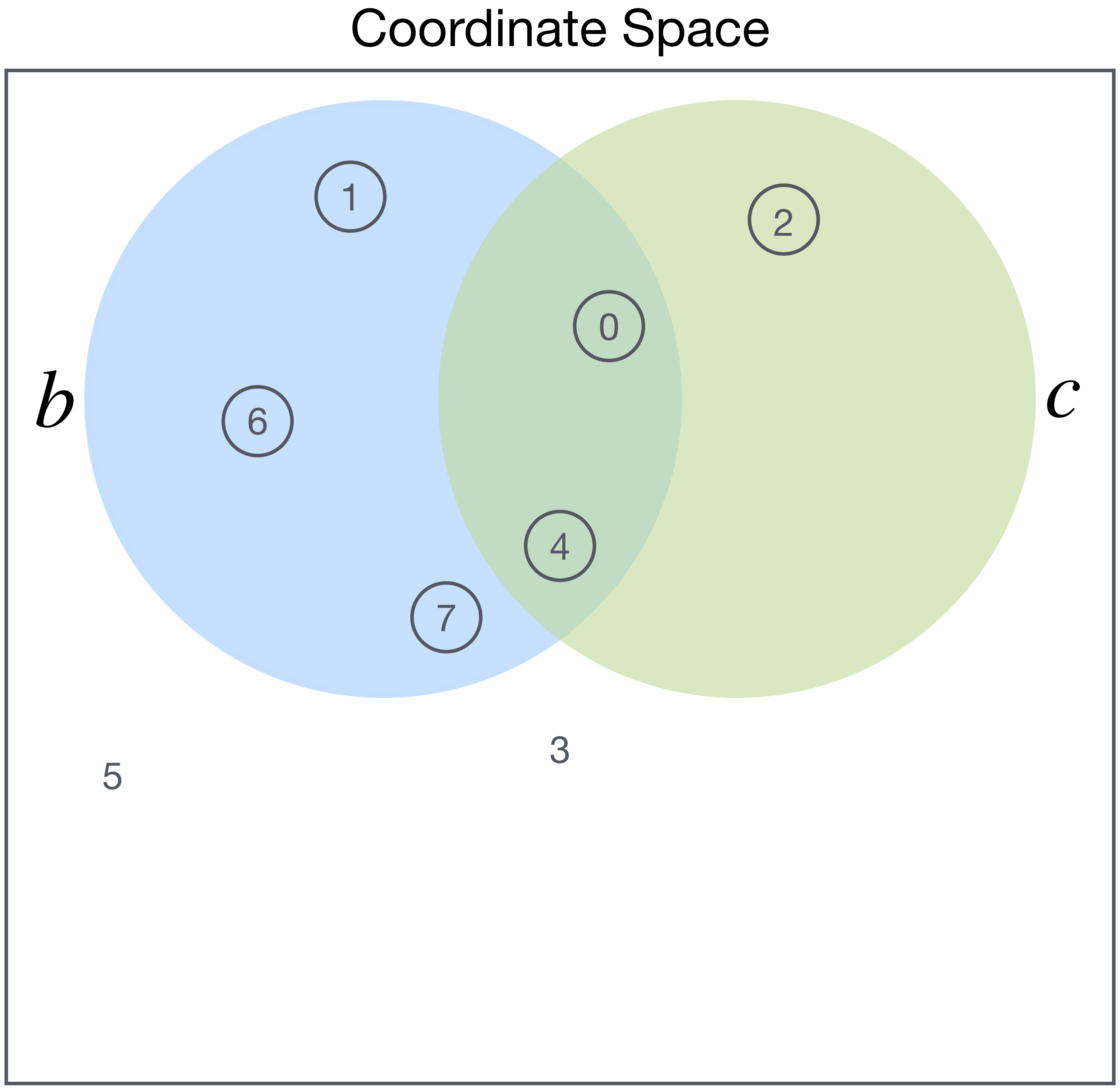
Merged coiteration

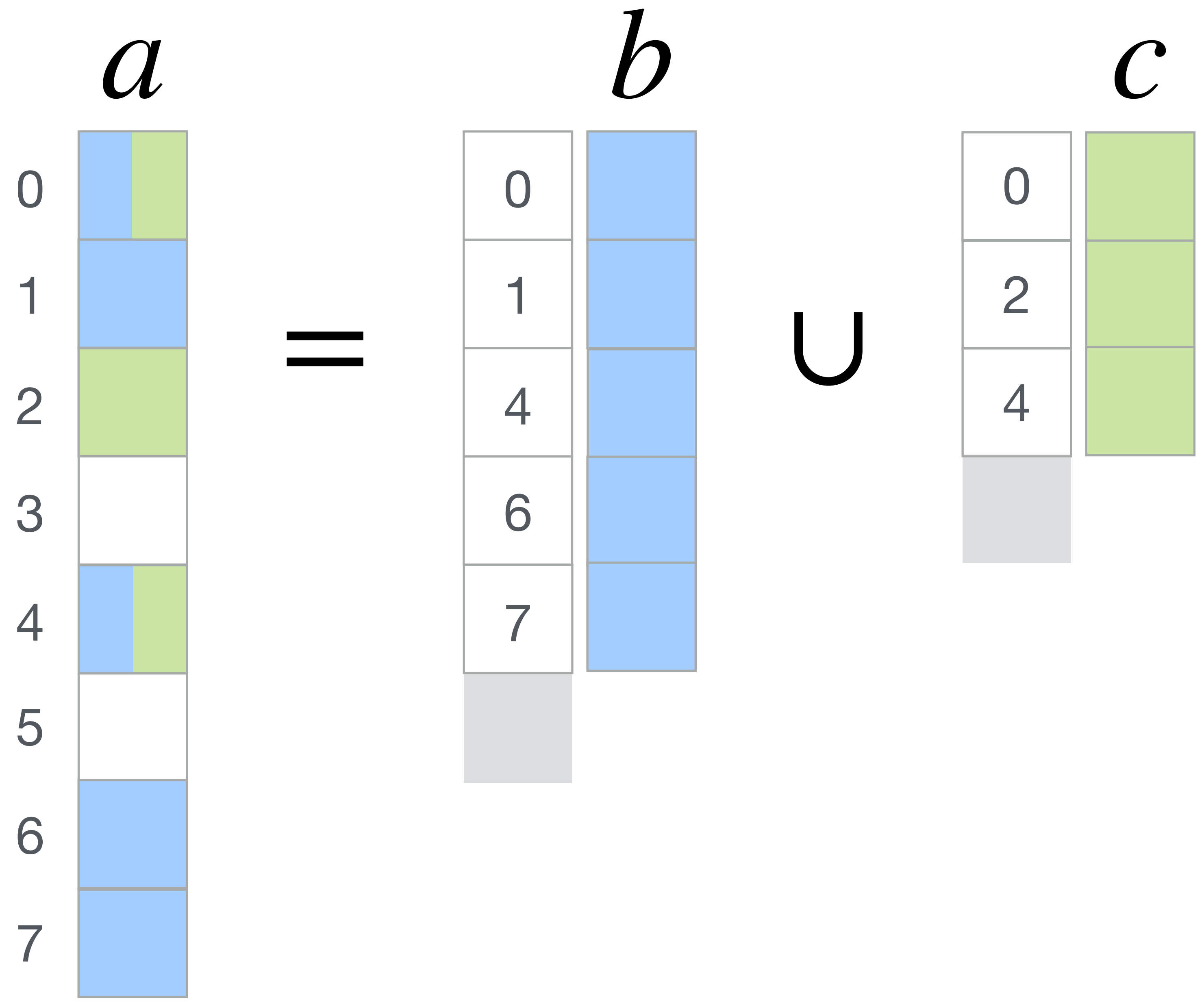
$$a_i = b_i + c_i$$



Merged coiteration

$$a_i = b_i + c_i$$

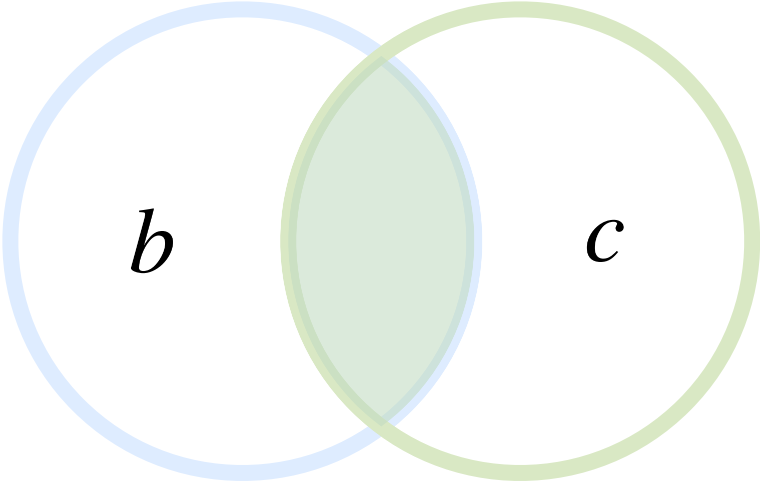




Merged coiteration code

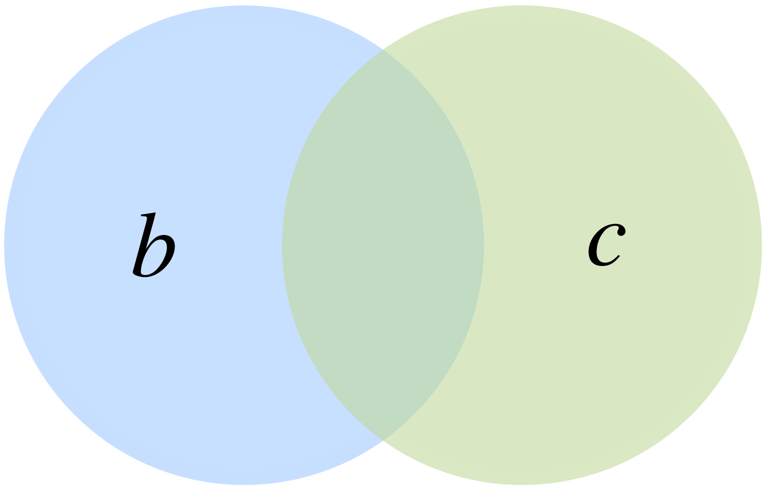
Intersection $b \cap c$

```
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++;
}
```



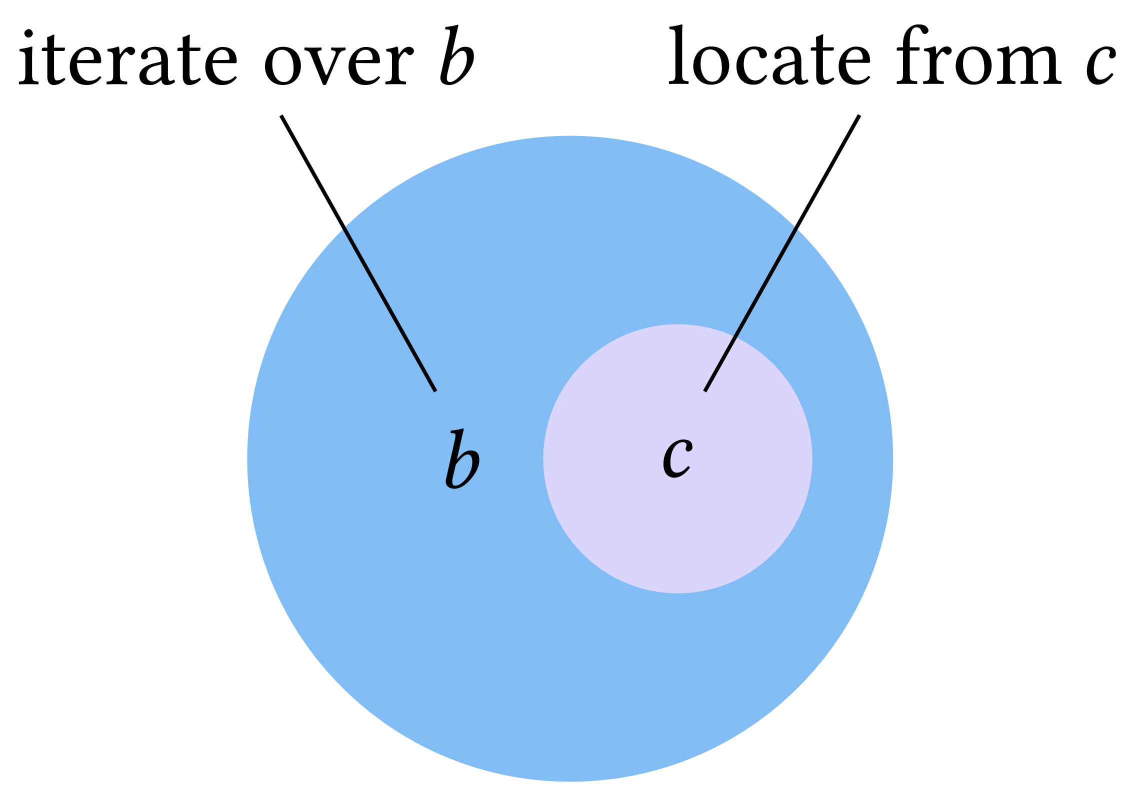
Union $b \cup c$

```
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)

$$a = \sum_i b_i c_i$$




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

Separation of Algorithm, Data Representation, and Schedule

Algorithm Language

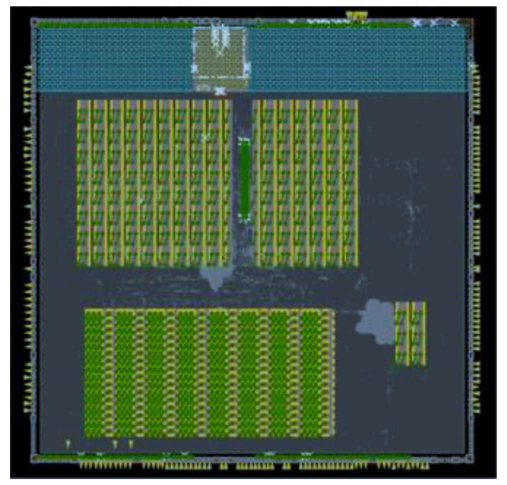
Data Representation Language

Scheduling Language

Compiler

CPUs

GPUs



DSAs

Most of HW retargeting is about changing schedules and data representations

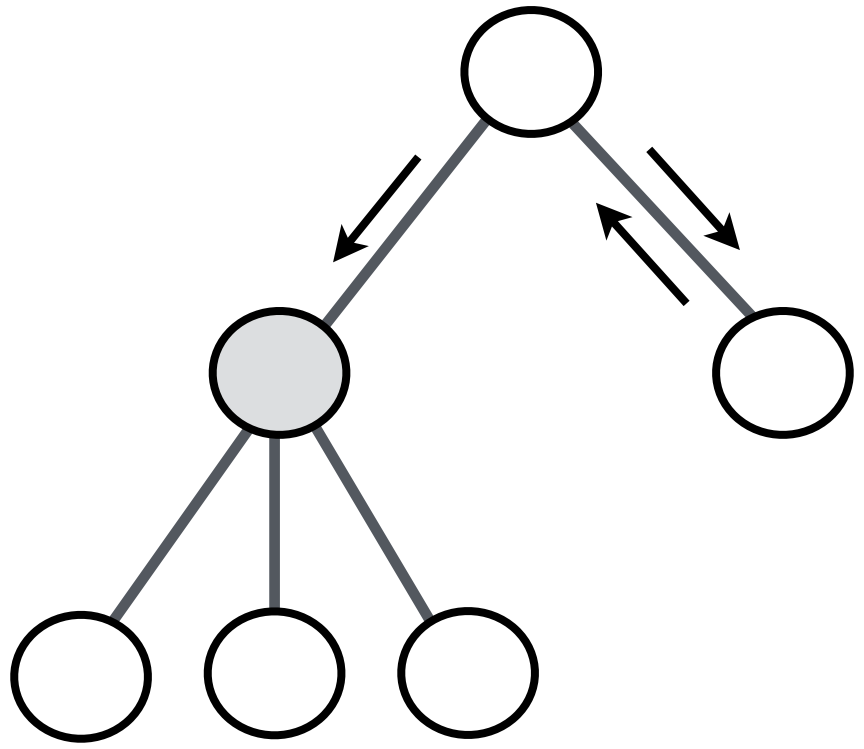
Data independence separates algorithm from data layout and order of computation

- You want a logical (abstract) representation that exposes only characteristics of the data — not how the data is stored
- Tree models and network models require programmers to write programs to traverse indices
- If the indices change, the programs must change
- A flat abstract view—as coordinates—allows the system to change data representation without users changing their programs
- Underneath the hood, the system can impose hierarchies and networks

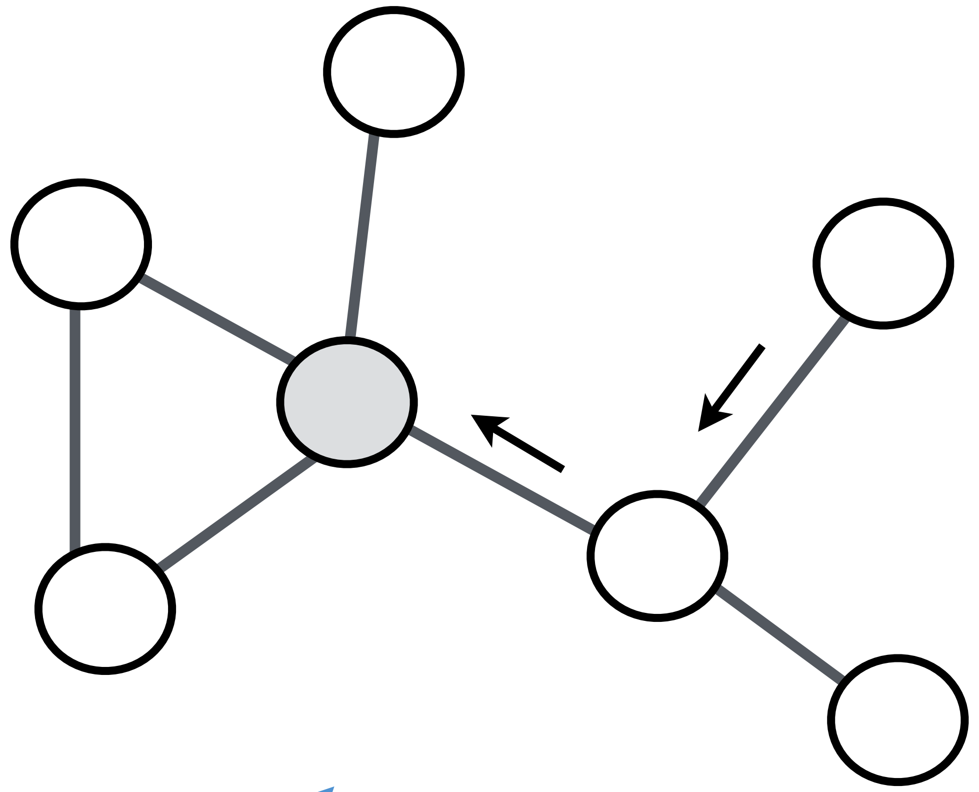
A Relational Model of Data for Large Shared Data Banks. *E. F. Codd* (1970)

Tree models, to network models, to the relational model

Tree model (1960s)
(e.g., IMS)

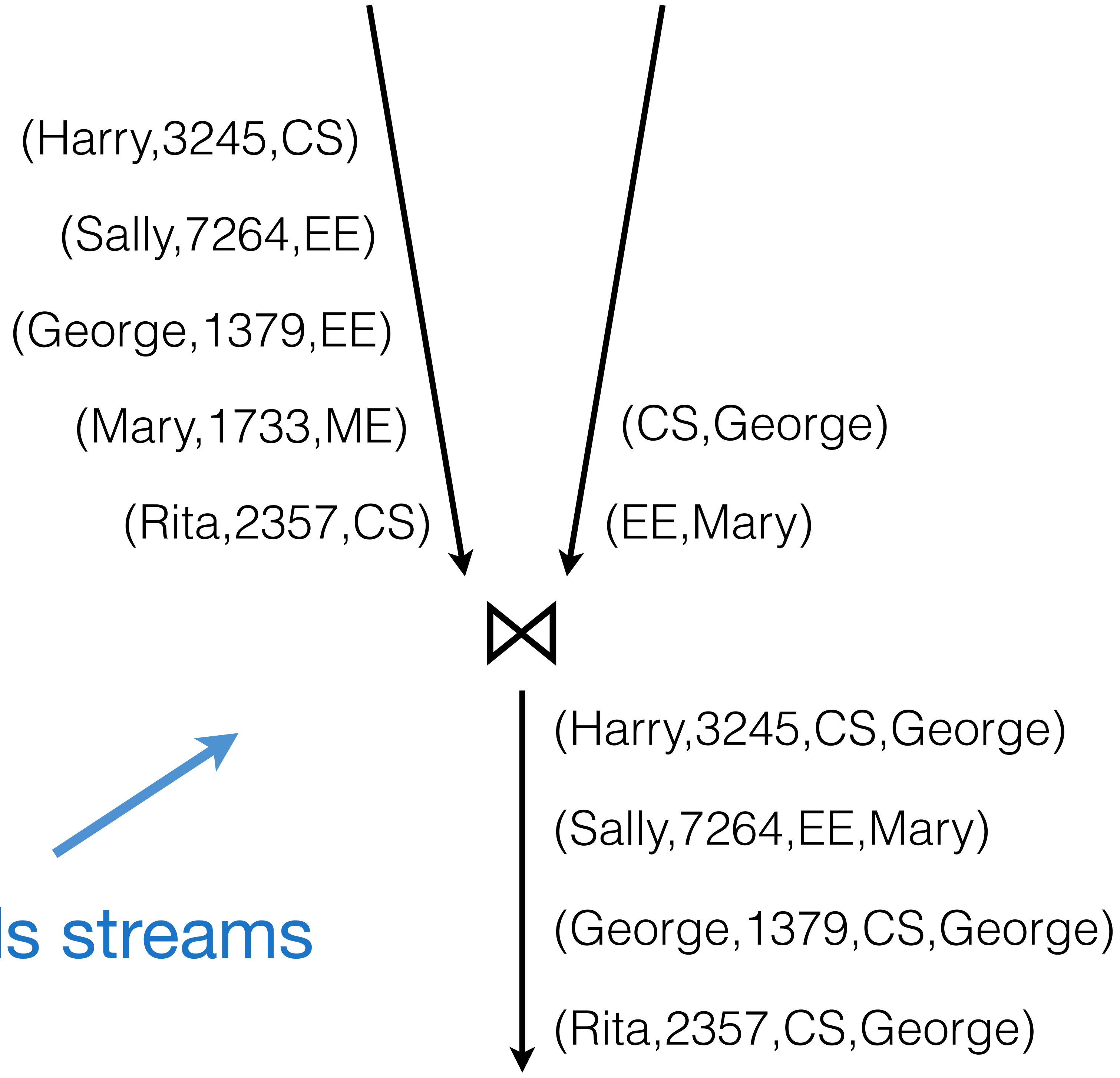


Network model (1970s)
(e.g., CODASYL)



Programmer as navigator

Relational model



Programmer controls streams

A Relational Model of Data for Large Shared Data Banks. *Codd* (1970)

The Programmer as Navigator. *Bachman* (1974)

What Goes Around Comes Around. *Stonebraker and Hellerstein* (2005)

Data structures in relational database management systems

- **Row stores** for efficient insertion
- **Column stores** for spatial locality during scans
- **B-trees** to efficiently support sorted data
- **Hash maps** for random access
- **Tries** for compression
- **Spatial data structures** for geo-spatial queries

But database management systems are still rewritten for each new major data structure

Separation of algorithm, schedule, and formats in tensor algebra

Tensor Expression

$A = Bc + a$ $\quad a = Bc$

$A = B \odot C$ $\quad A = B + C$ $\quad a = \alpha Bc + \beta a$

$A = BCd$ $\quad A = \alpha B$ $\quad A = 0$ $\quad A = BC$

$a = b \odot c$ $\quad A = B \odot (CD)$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $\quad A = B^T$ $\quad a = B^T Bc$

$A_{ik} = \sum_{j} B_{ijk} c_j$ $\quad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $\quad A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\quad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Formats

Dense Matrix CSR BCSR DCSR COO ELLPACK CSB DIA Blocked COO CSC Blocked DIA DCSC Sparse vector Hash Maps CSF Dense Tensors Blocked Tensors

Schedule

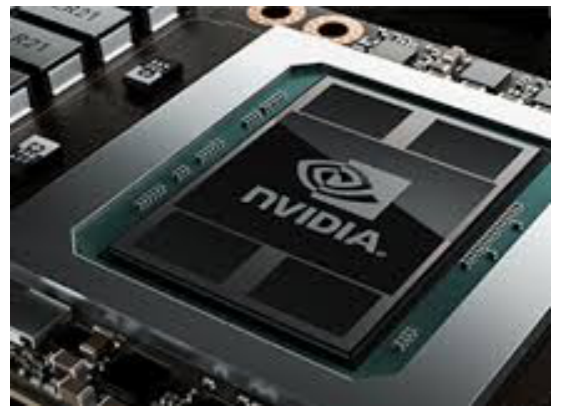
reorder split collapse precmpute gpu unroll parallelize

Sparse Tensor Algebra Compiler (taco)

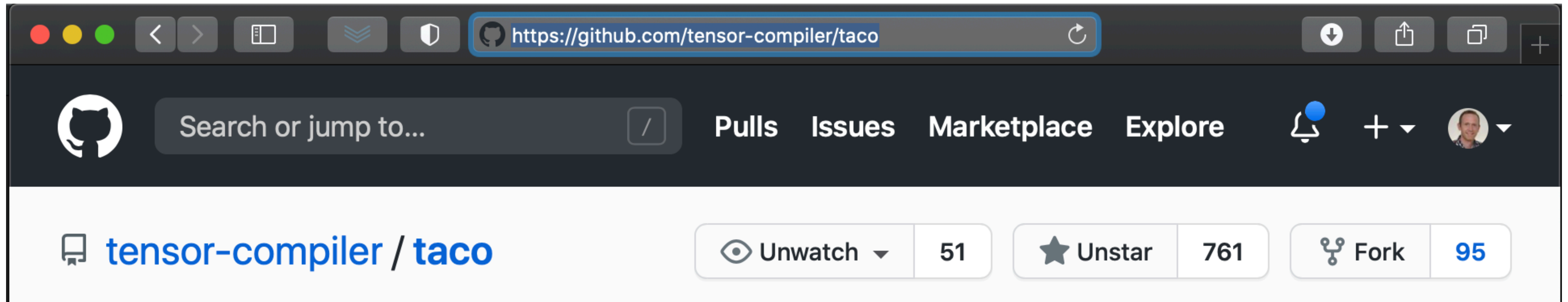
THE C PROGRAMMING LANGUAGE

NVIDIA CUDA

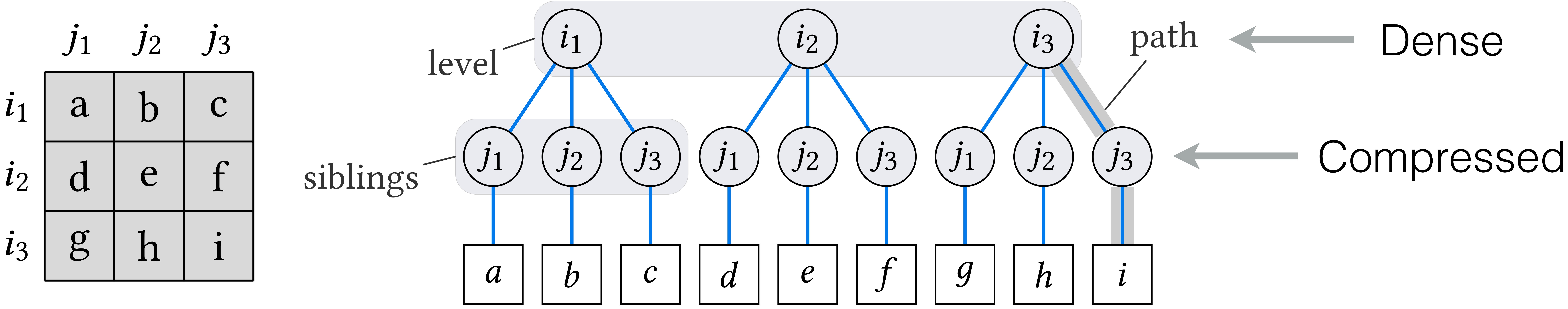
DSAs



(work in progress)



Tensor algebra format data representation language



CSR
Dense
Compressed

CSF
Dense
Compressed
Compressed

Coordinate matrix
Compressed
Singleton

BCSR
Dense
Compressed
Dense
Dense

Hash map vector
Hashed

Tensor algebra scheduling language

- **reorder(i, j)** interchanges loops i and j
- **split(i, i_1, i_2, d, s, t)** strip-mines i into two loops i_1 and i_2, where i_1 or i_1 is of size s depending on the direction d. The tensor t is optional and, if given, means the loop is strip-mined w.r.t. its nonzeros.
- **collapse(i, j, f)** collapses loops i and j into a new loop f, which iterates over their Cartesian combination.
- **precompute(S, e, t, I)** precomputes expression e in index statement S before the loops I and stores the results in tensor t.
- **unroll**, **parallelize**, **vectorize**, …

Overview of lectures in the coming weeks