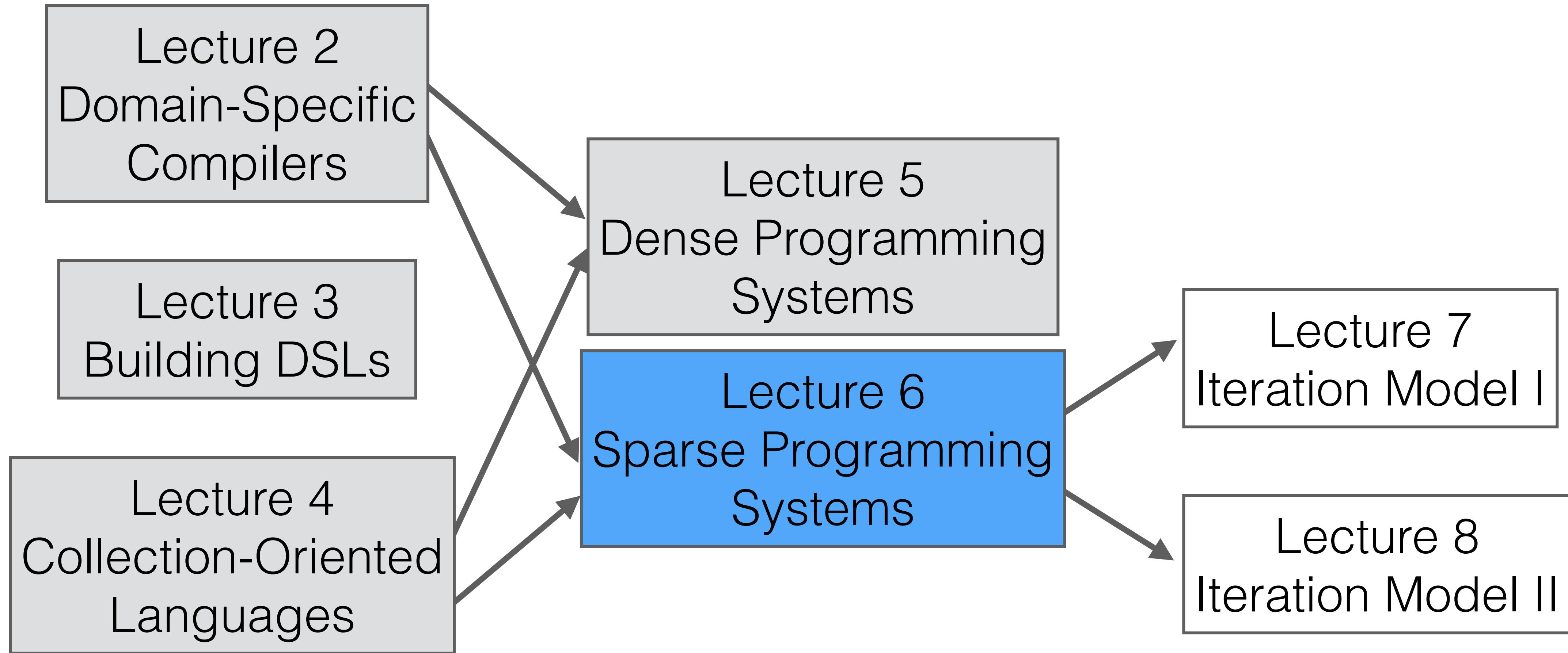


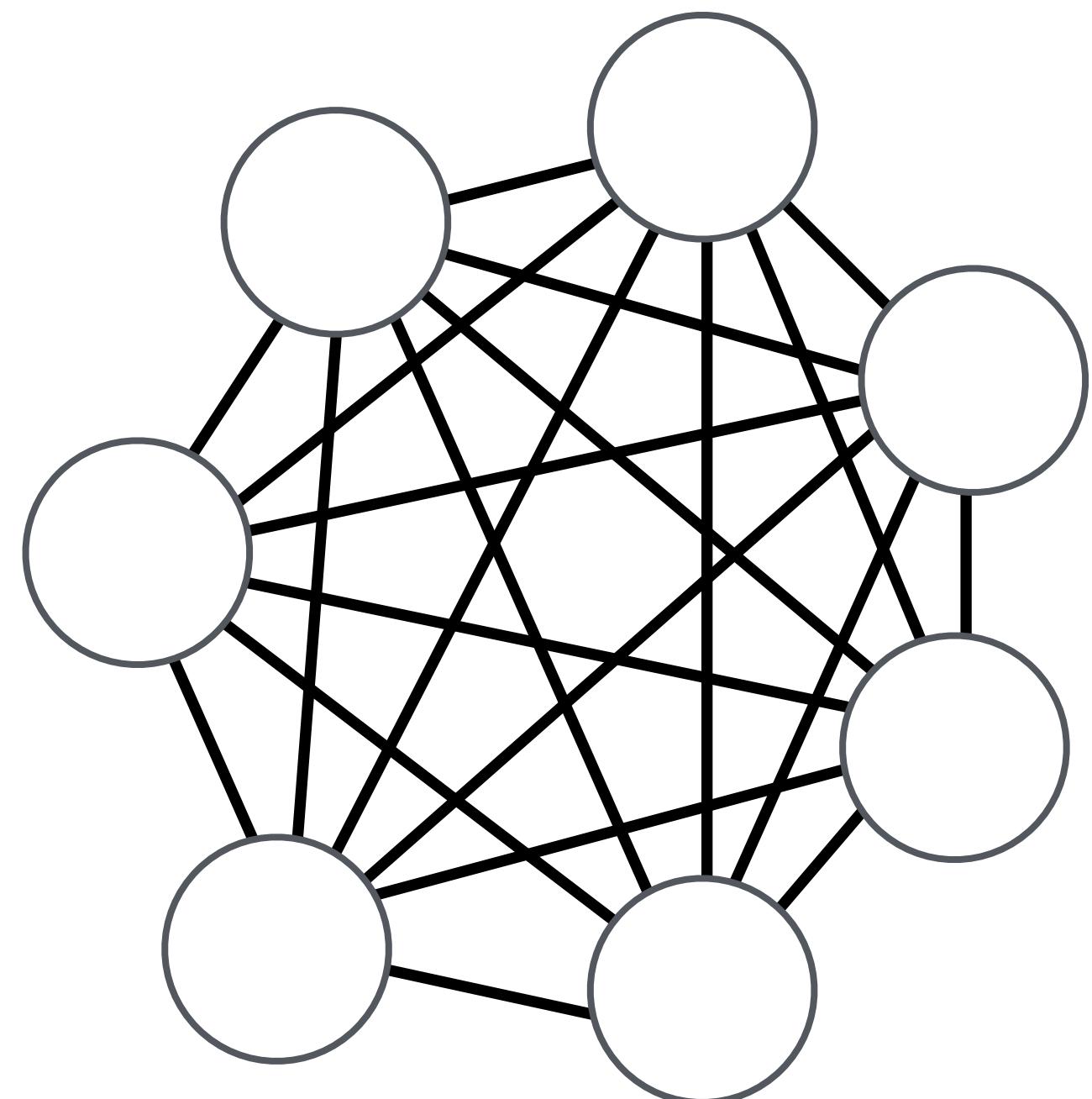
# Lecture 6 – Sparse Programming Systems

Stanford CS343D (Winter 2025)  
Fred Kjolstad

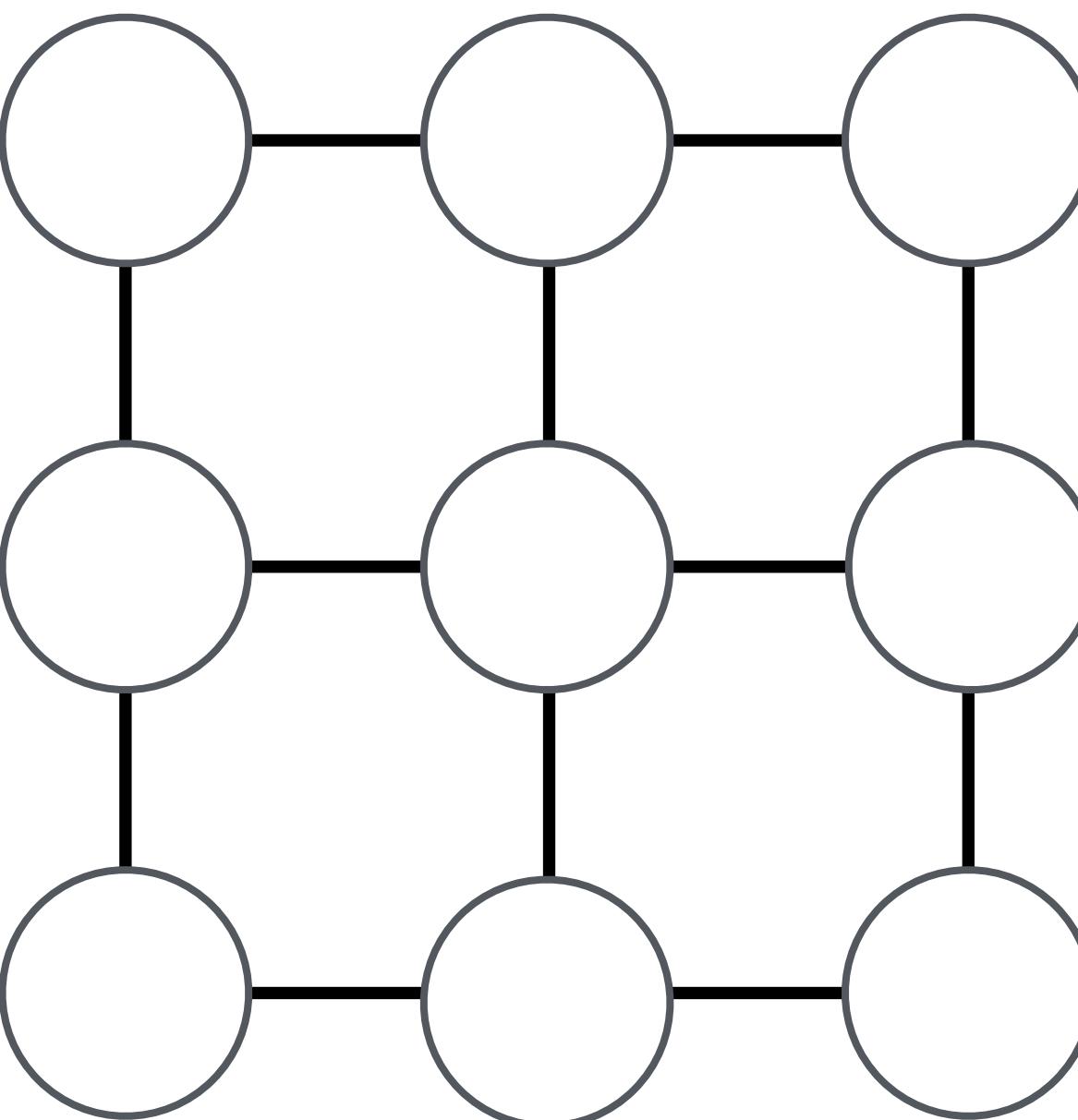


# Terminology: Regular and Irregular

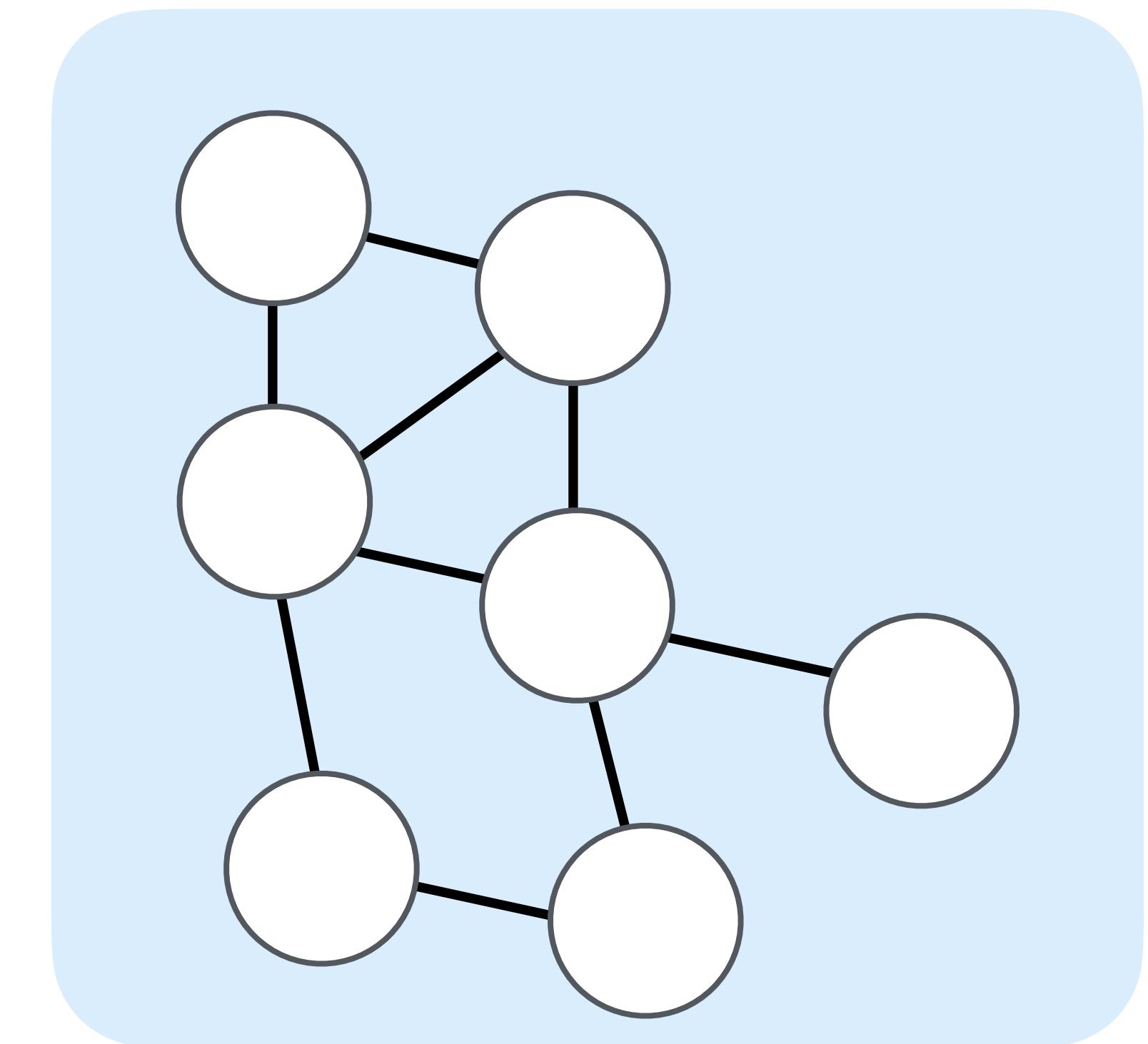
Fully Connected System



Regular System

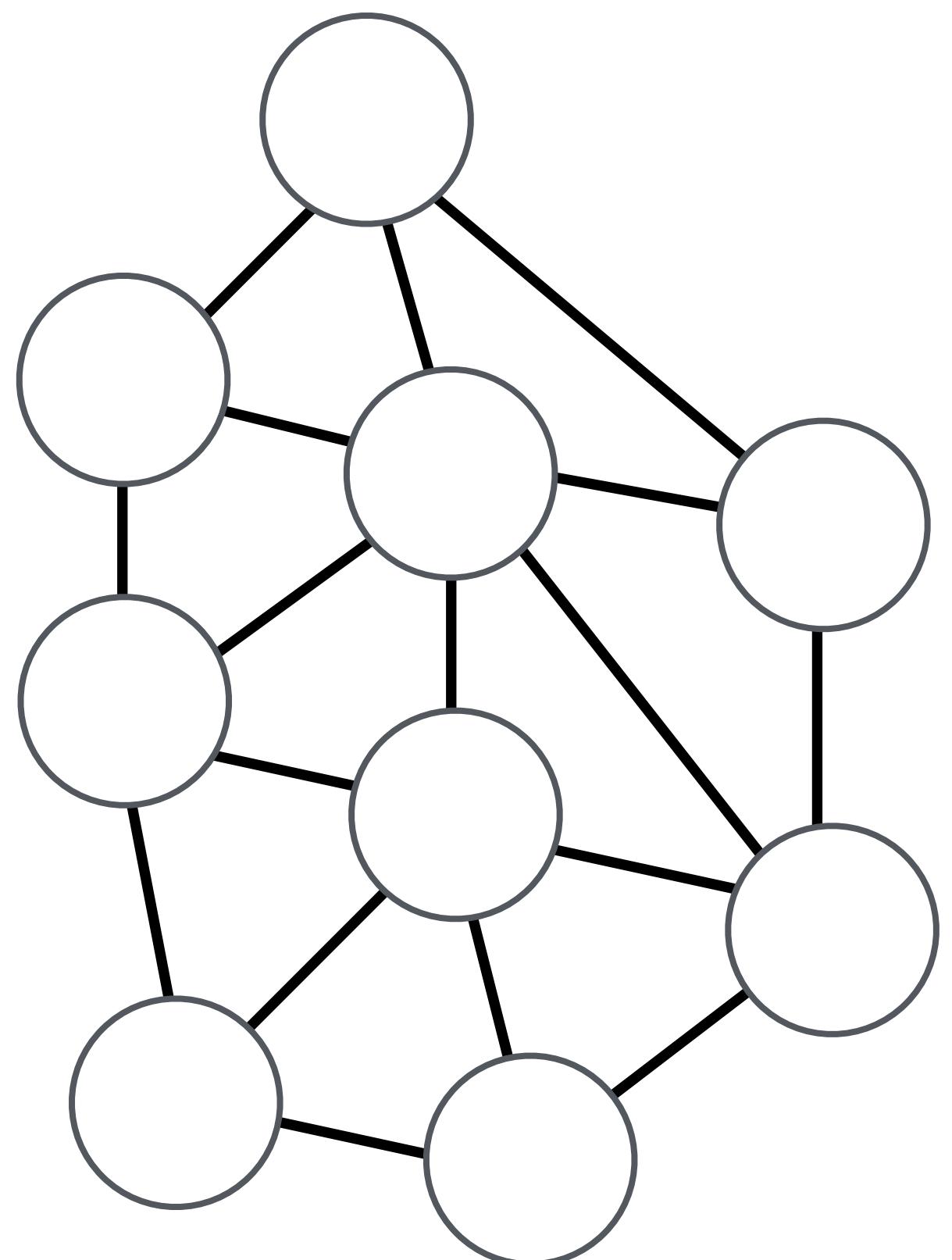


Irregular System

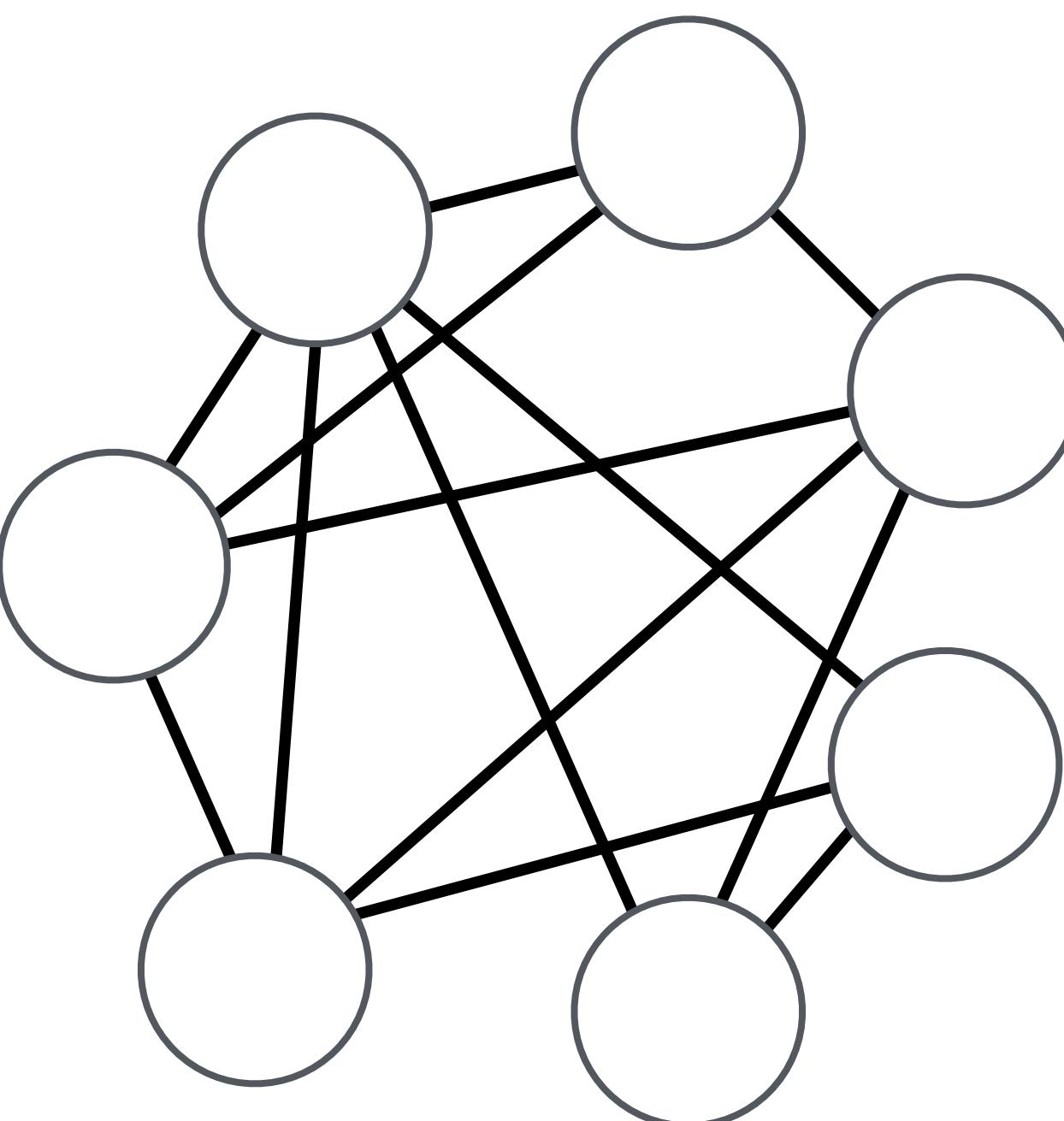


# Three classes of irregular systems

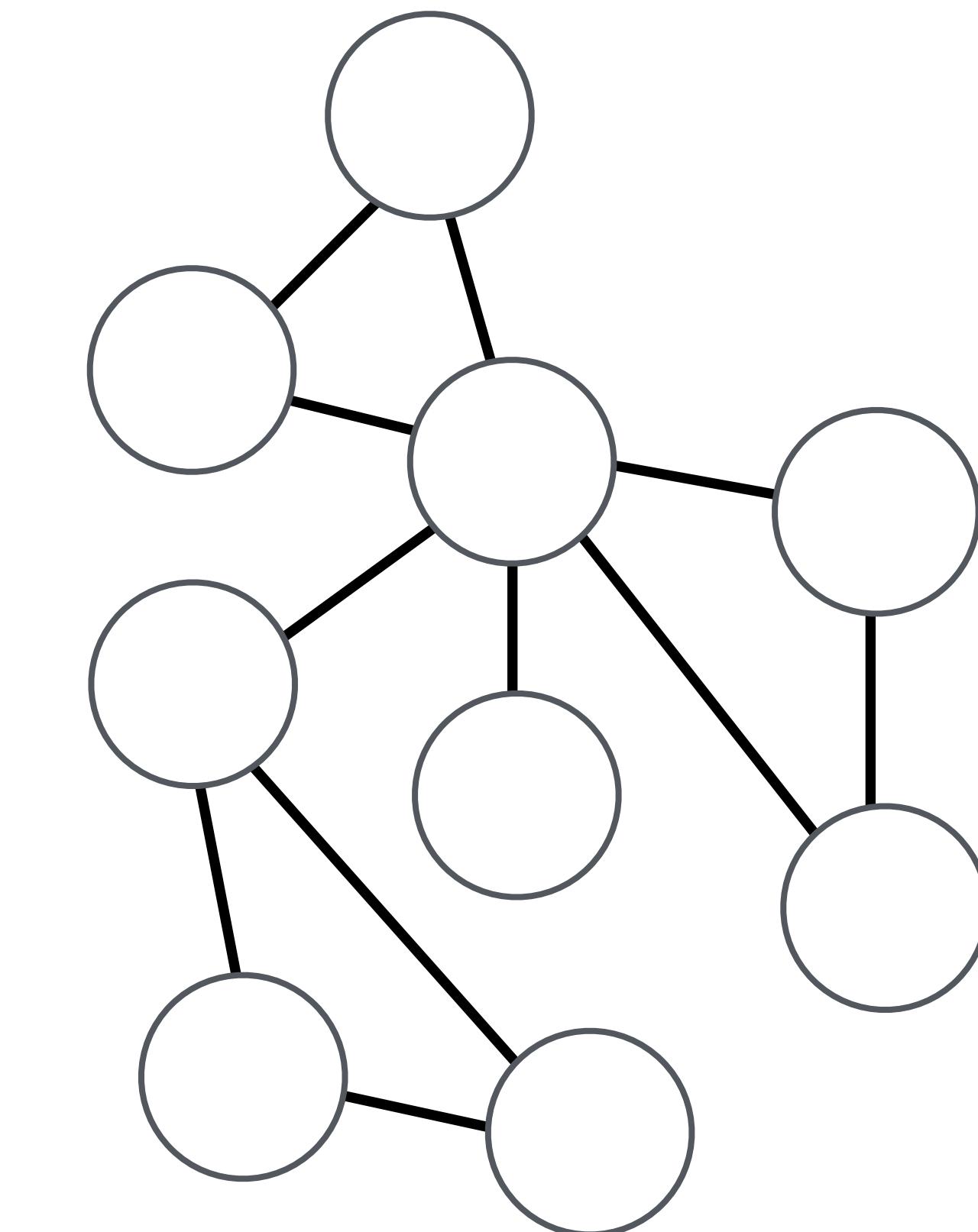
Road Networks



Fractional Sparsity



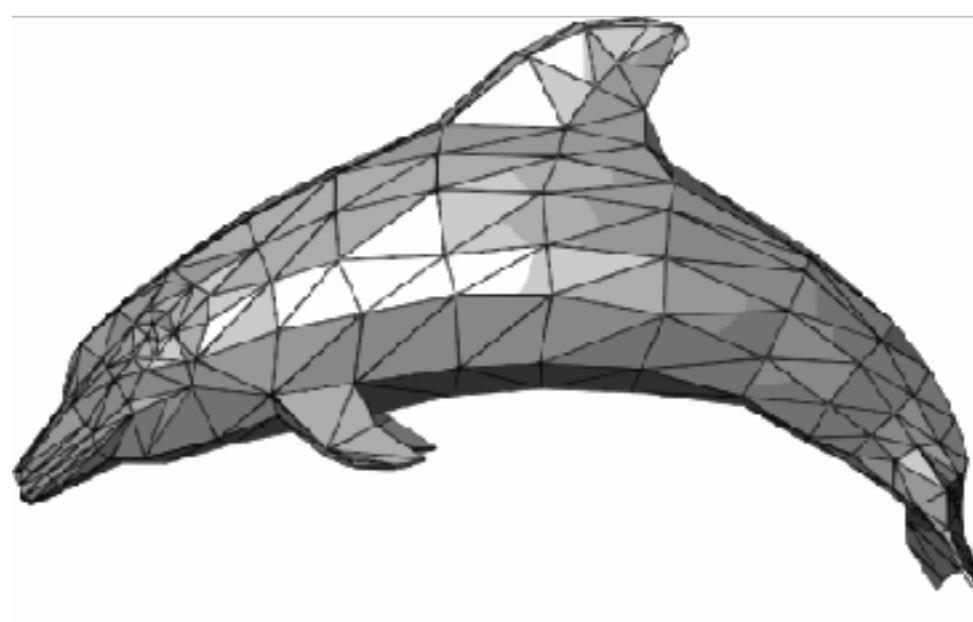
Power Law Graphs



How sparse is graph/relational data? Often asymptotically sparse.

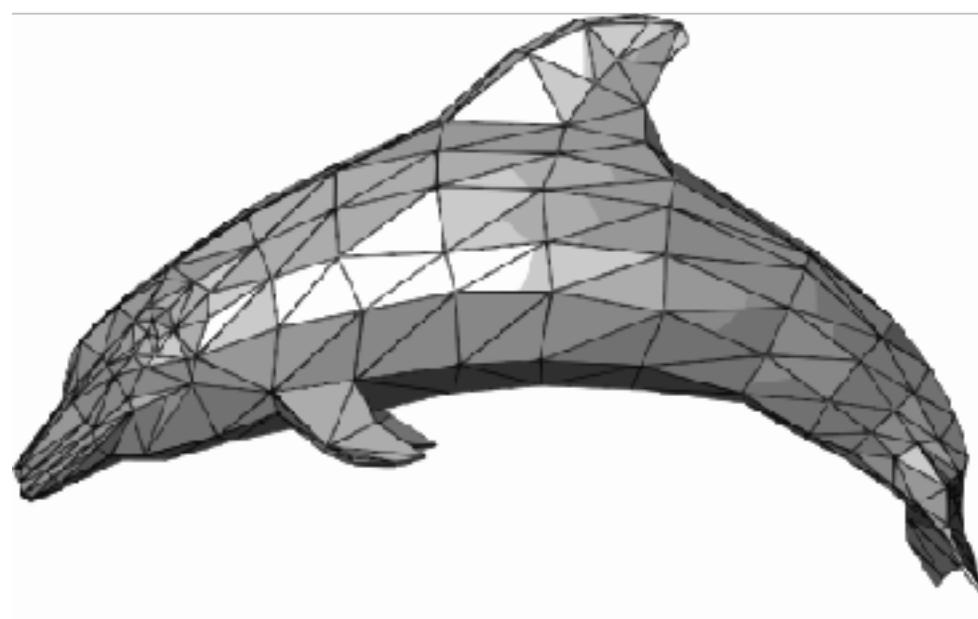
How sparse is graph/relational data? Often asymptotically sparse.

## Conditioned Meshes

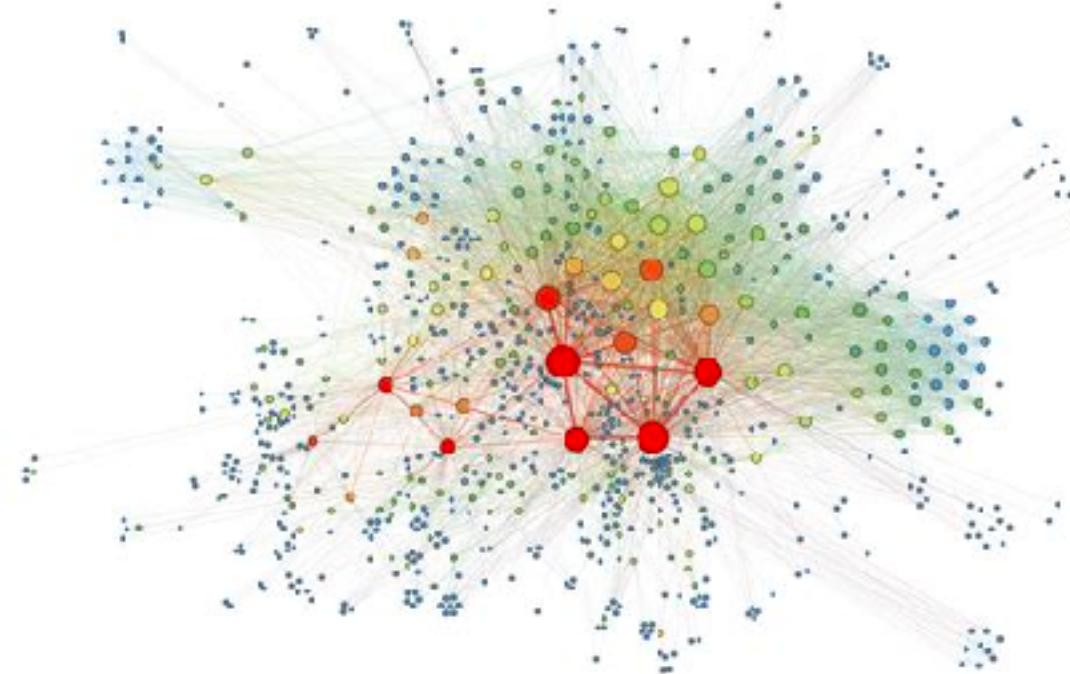


How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes

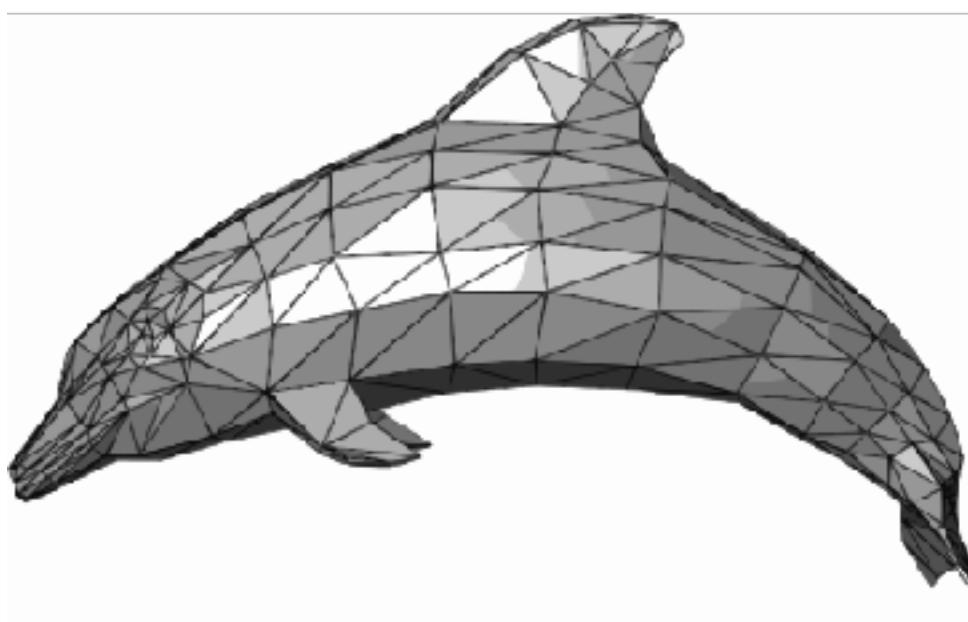


Power-law graphs



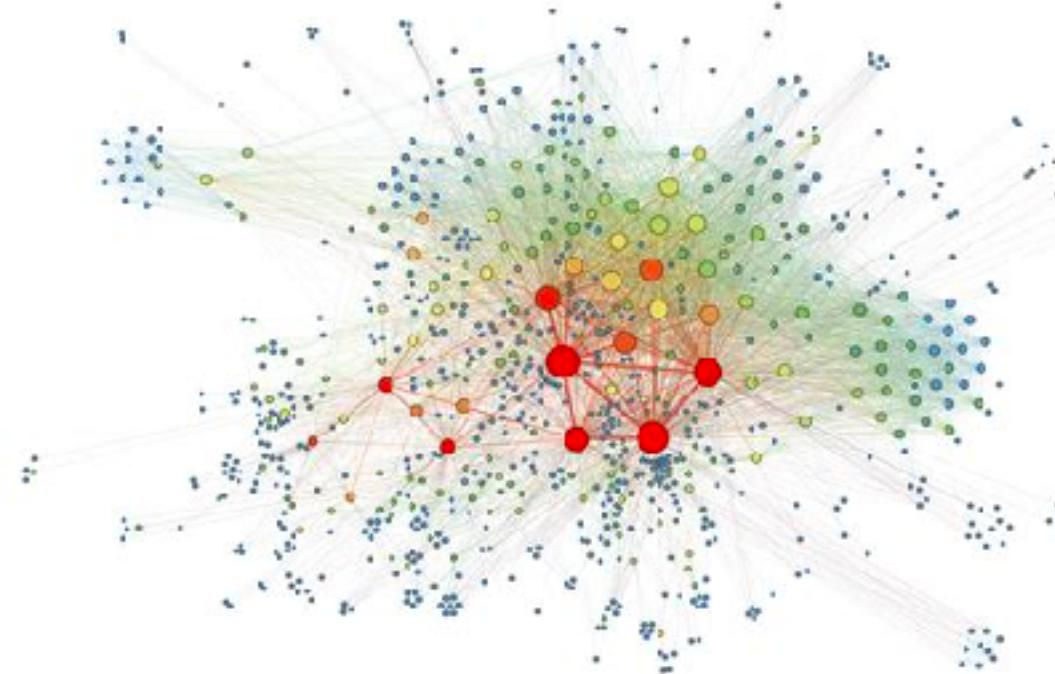
# How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes



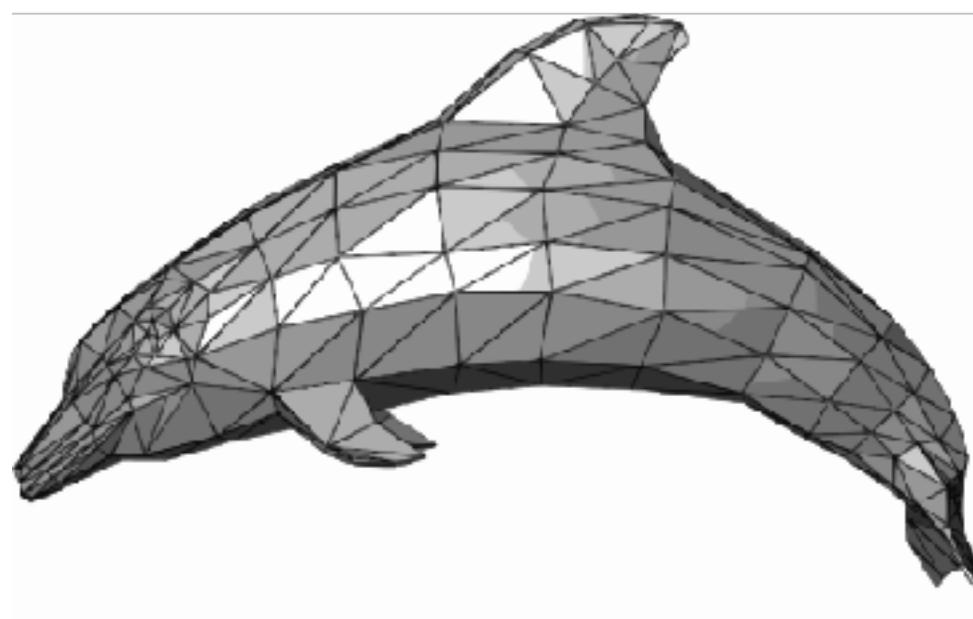
Assume an average degree of 150 (e.g., 150 friends)

Power-law graphs



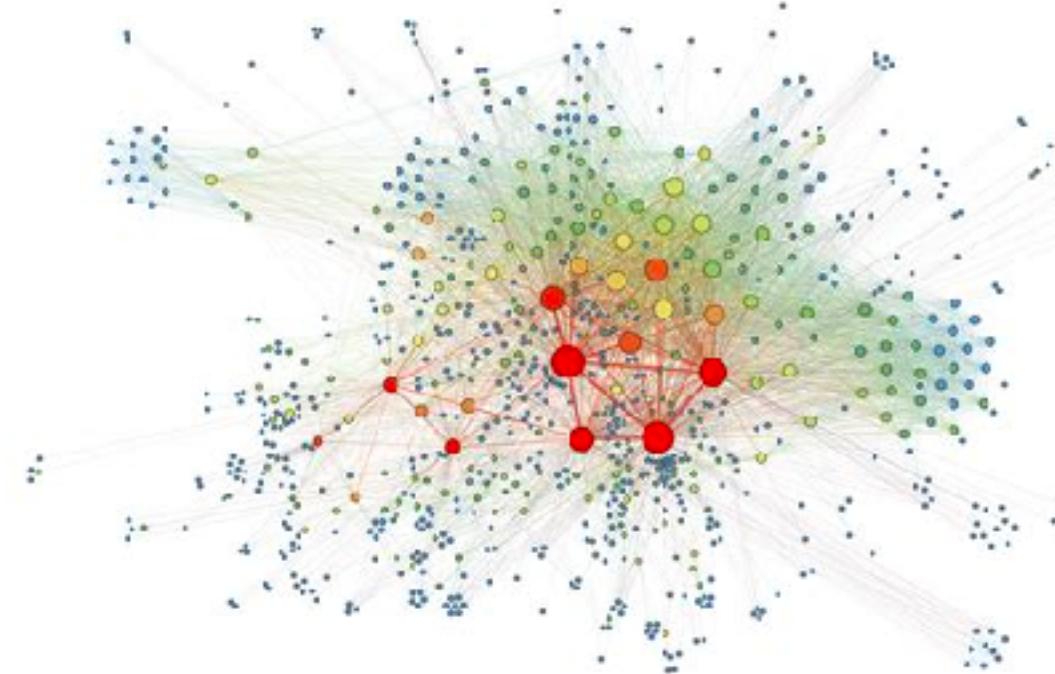
# How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes



Assume an average degree of 150 (e.g., 150 friends)

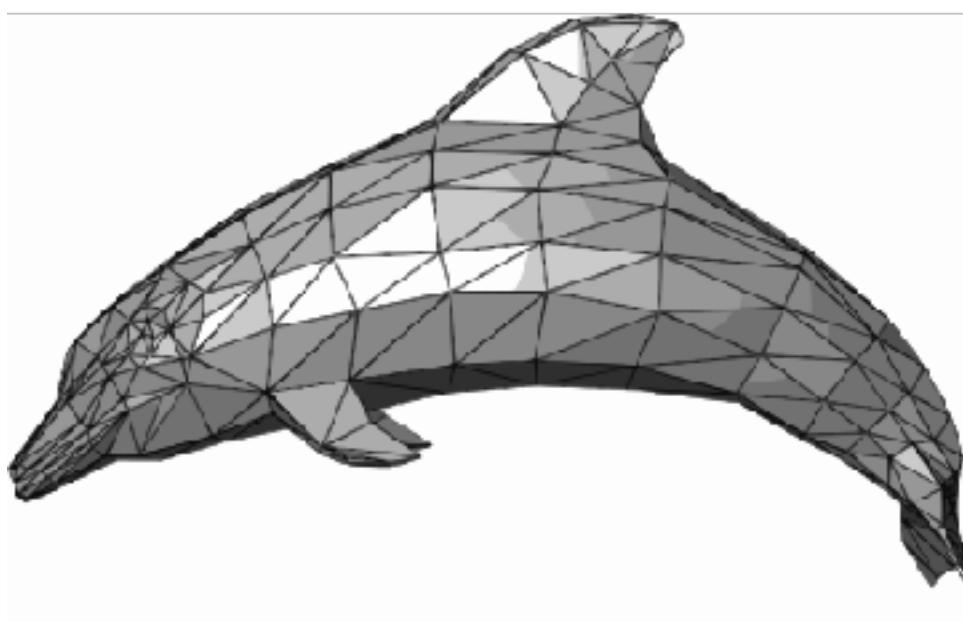
Power-law graphs



Each matrix row then has 150 nonzeros

# How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes

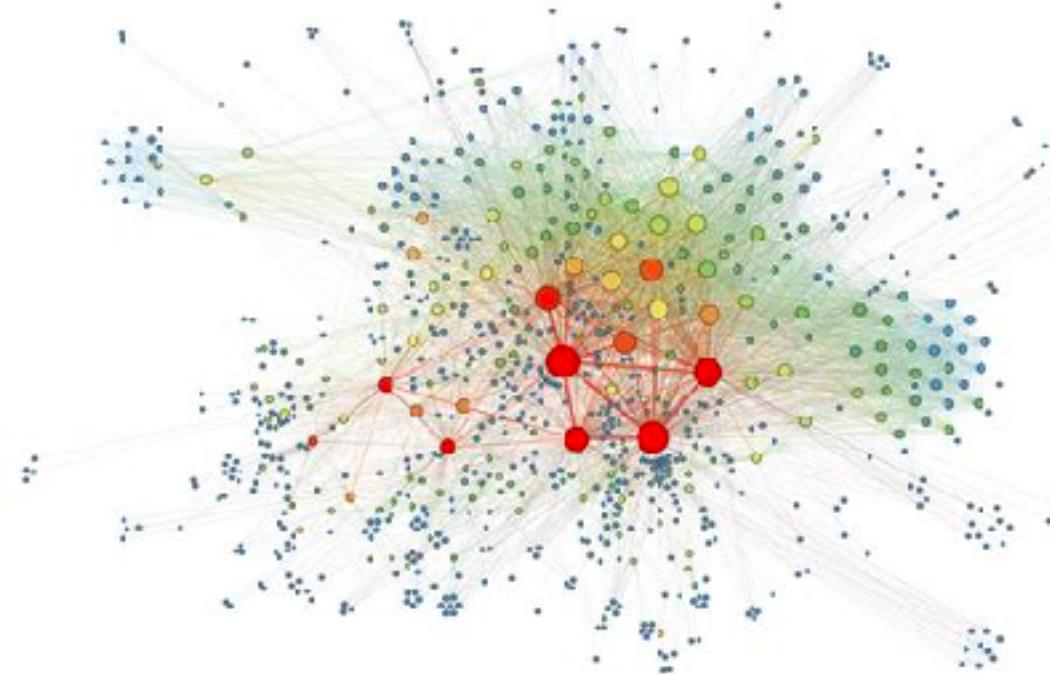


Assume an average degree of 150 (e.g., 150 friends)

Each matrix row then has 150 nonzeros

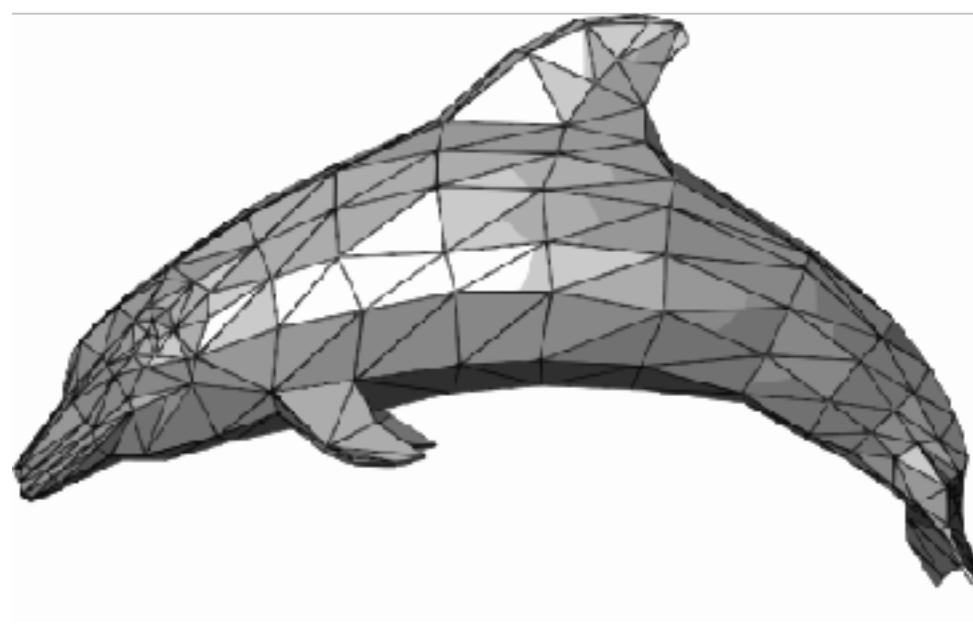
At 10,000 rows:  $\frac{150 \cdot 10,000}{10,000^2} = 1.5\% \text{ nonzeros}$

Power-law graphs



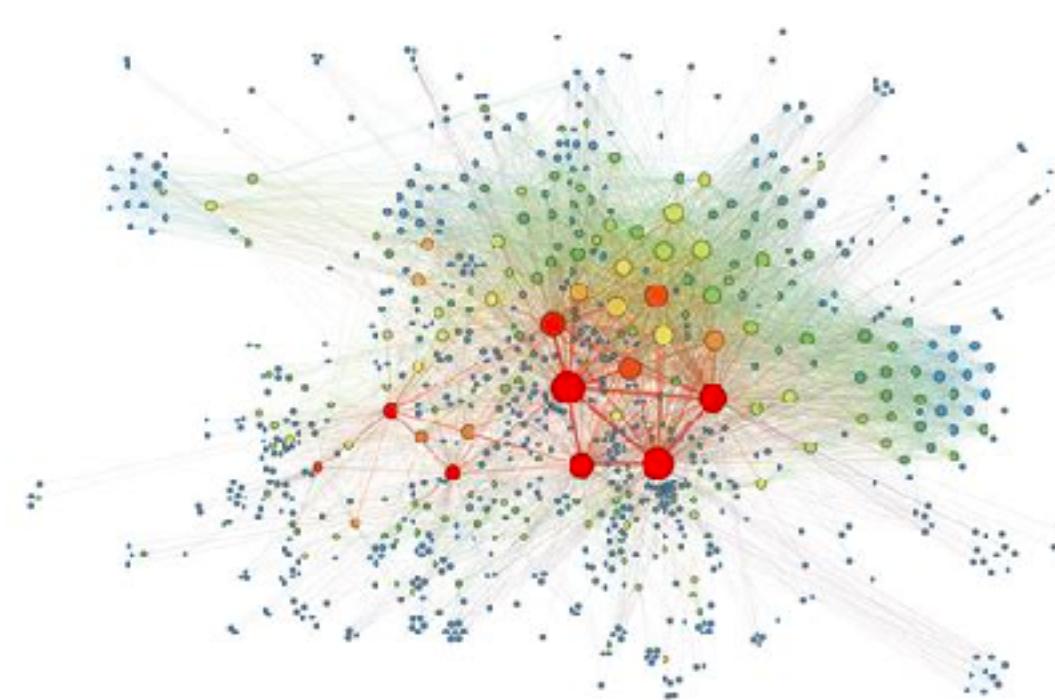
# How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes



Assume an average degree of 150 (e.g., 150 friends)

Power-law graphs



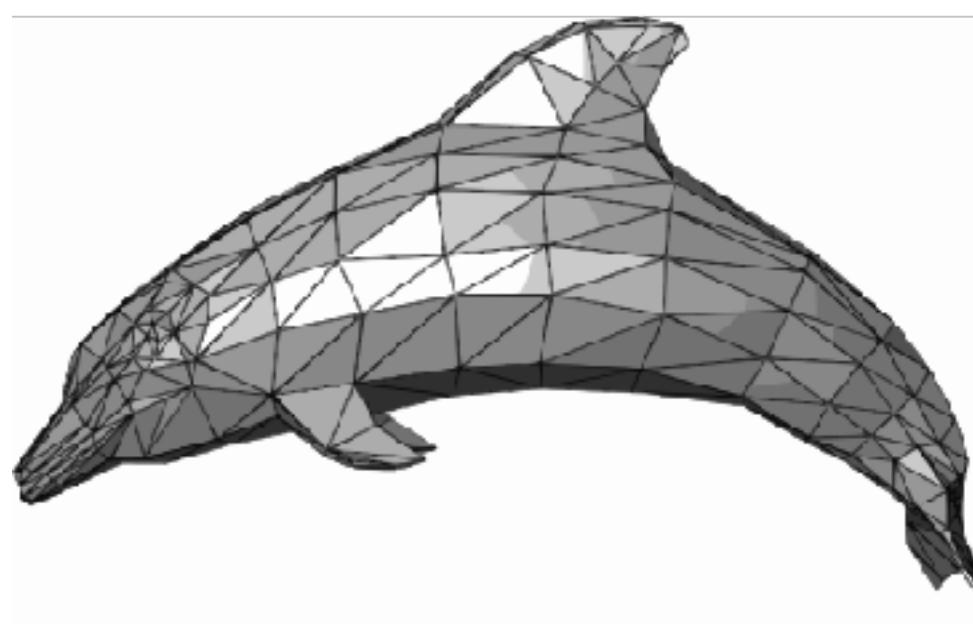
Each matrix row then has 150 nonzeros

At 10,000 rows:  $\frac{150 \cdot 10,000}{10,000^2} = 1.5\% \text{ nonzeros}$

At 100,000 rows:  $\frac{150 \cdot 100,000}{100,000^2} = 0.15\% \text{ nonzeros}$

# How sparse is graph/relational data? Often asymptotically sparse.

## Conditioned Meshes

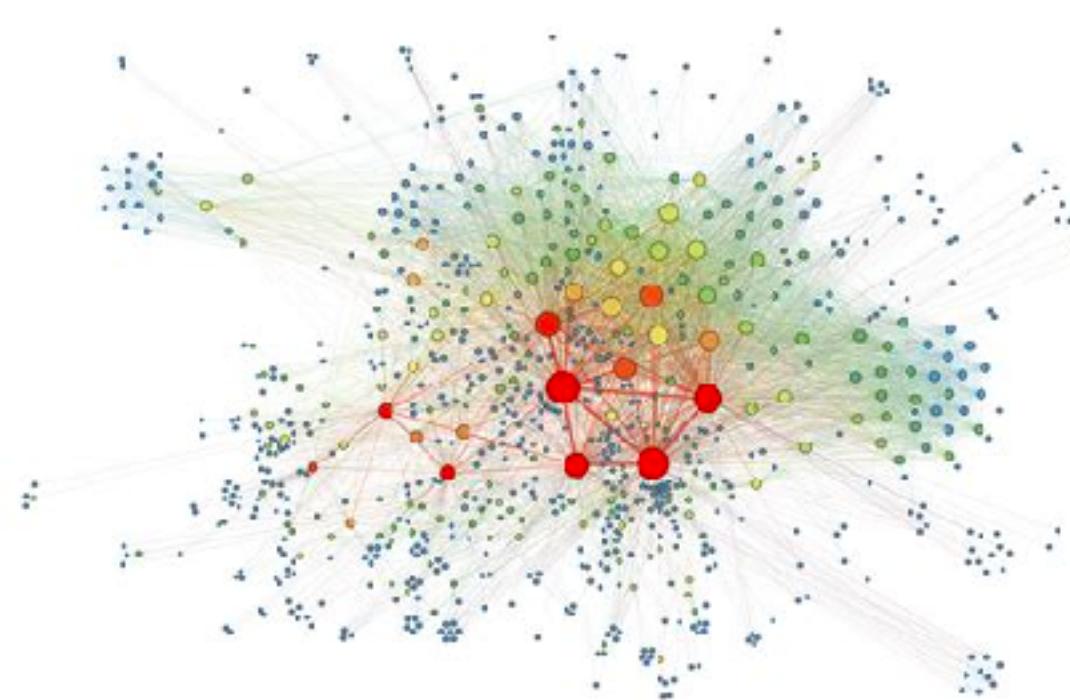


Assume an average degree of 150 (e.g., 150 friends)

Each matrix row then has 150 nonzeros

At 10,000 rows:  $\frac{150 \cdot 10,000}{10,000^2} = 1.5\% \text{ nonzeros}$

## Power-law graphs



At 100,000 rows:  $\frac{150 \cdot 100,000}{100,000^2} = 0.15\% \text{ nonzeros}$

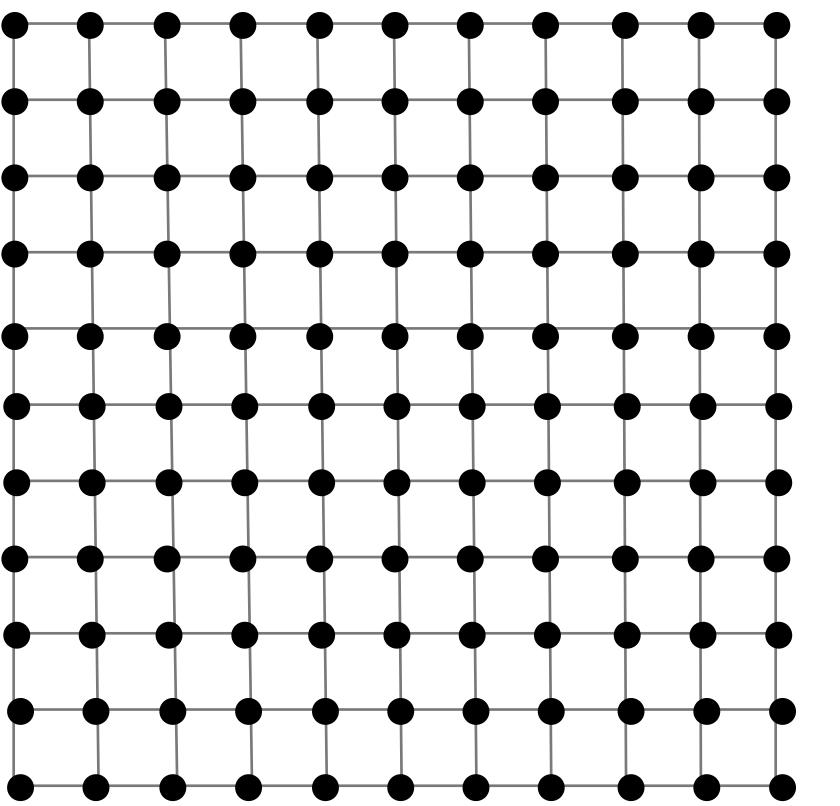
Matrix components:  $O(n^2)$

Nonzeros:  $O(n)$

Fraction of nonzeros:  $O(1/n)$

# Terminology: Dense and Sparse

Dense loop iteration space

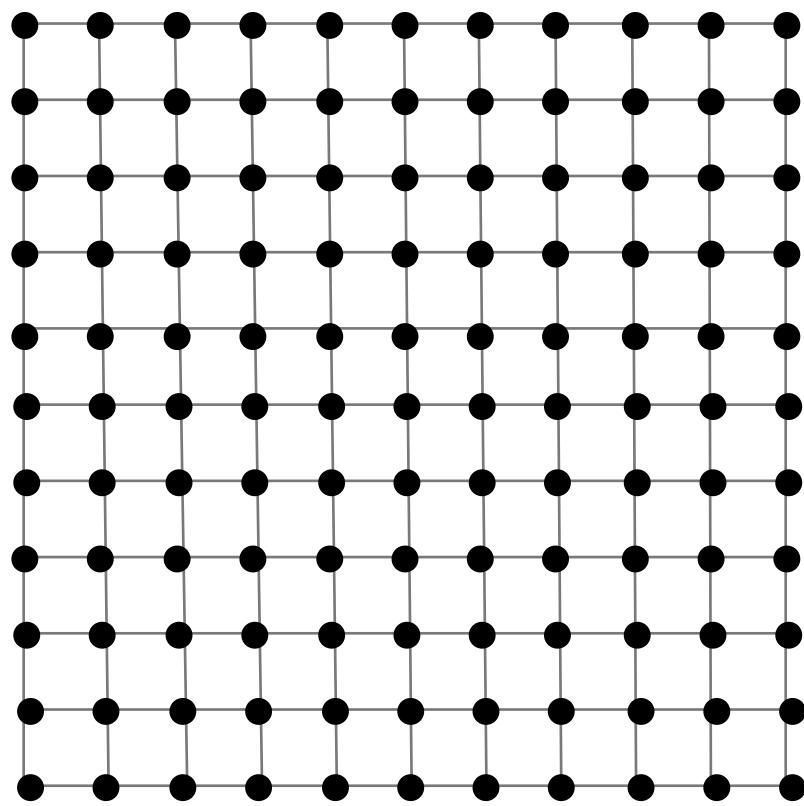


```
for (int i = 0; i < m; i++) {  
    for (int j = 0; j < n; j++) {  
        y[i] += A[i*n+j] * x[j];  
    }  
}
```

$$y = Ax$$

# Terminology: Dense and Sparse

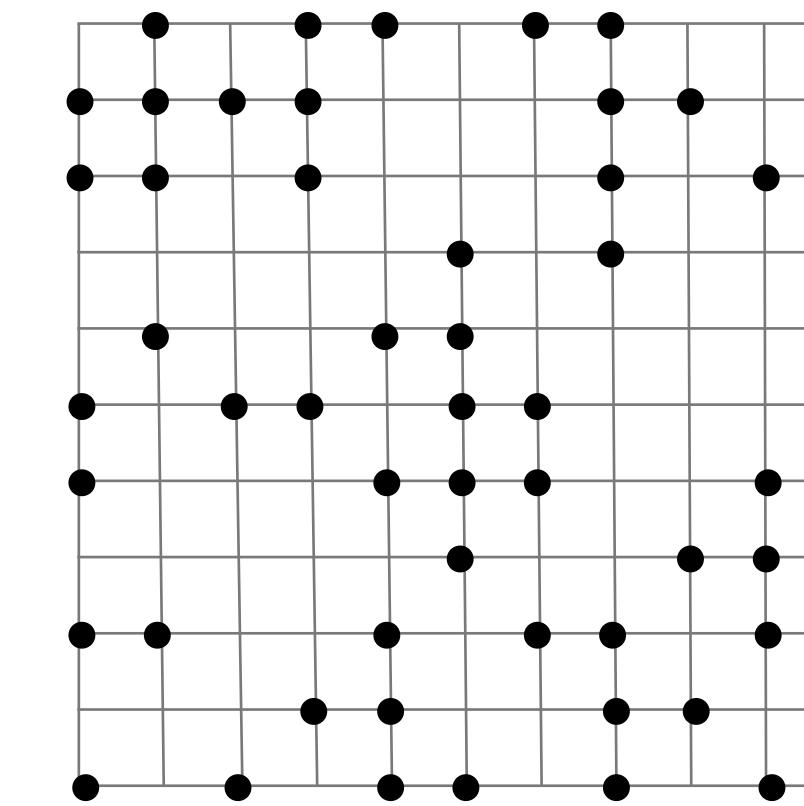
Dense loop iteration space



```
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

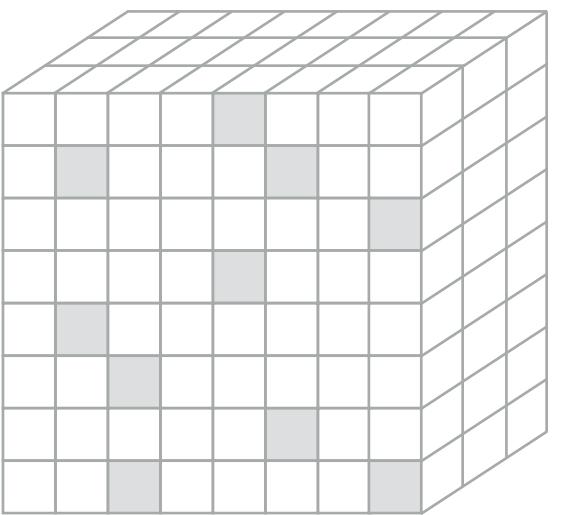


```
for (int i = 0; i < m; i++) {  
    for (int pA = A2_pos[i]; pA < A_pos[i+1]; pA++) {  
        int j = A_crd[pA];  
        y[i] += A[pA] * x[j];  
    }  
}
```

$$y = Ax$$

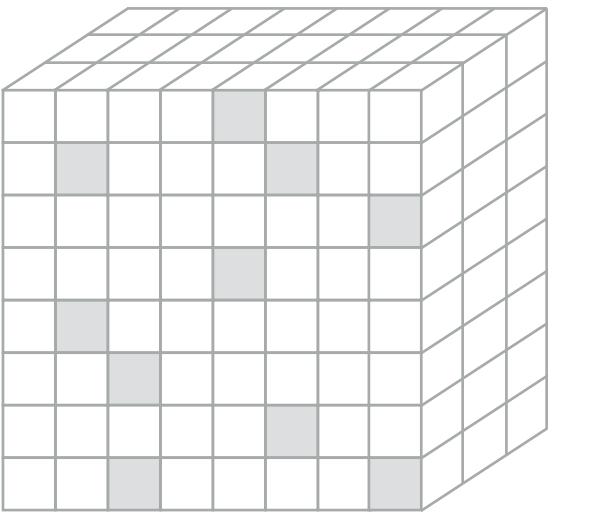
# Three sparse applications areas

Tensors



# Three sparse applications areas

Tensors

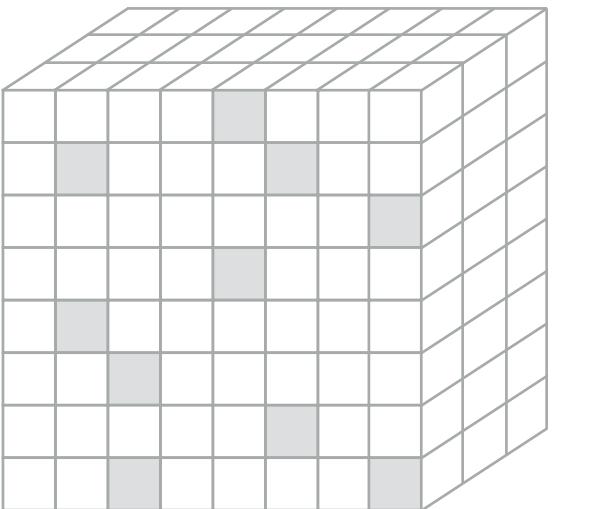


Relations

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

# Three sparse applications areas

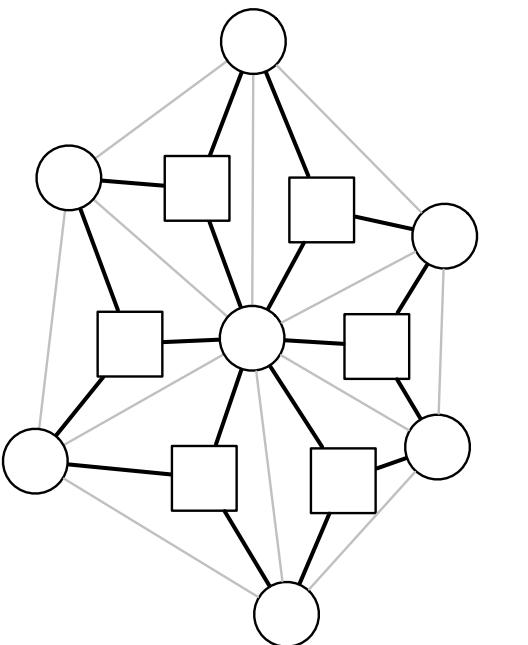
Tensors



Relations

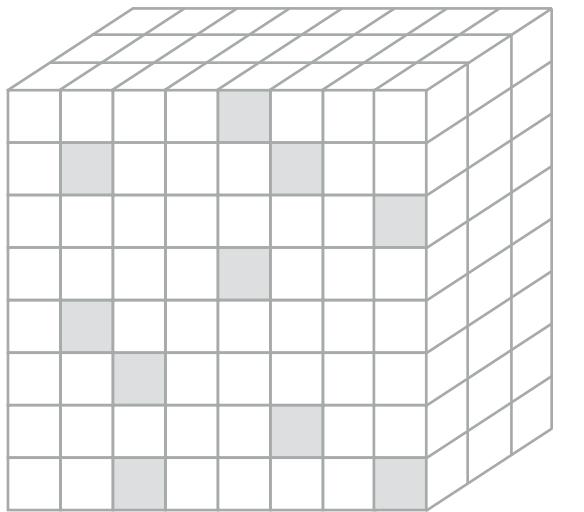
Names	City	Age
Peter	Boston	54
Mary	San Fransisco	35
Paul	New York	23
Adam	Seattle	84
Hilde	Boston	19
Bob	Chicago	76
Sam	Portland	32
Angela	Los Angeles	62

Graphs



# Three sparse applications areas

Tensors



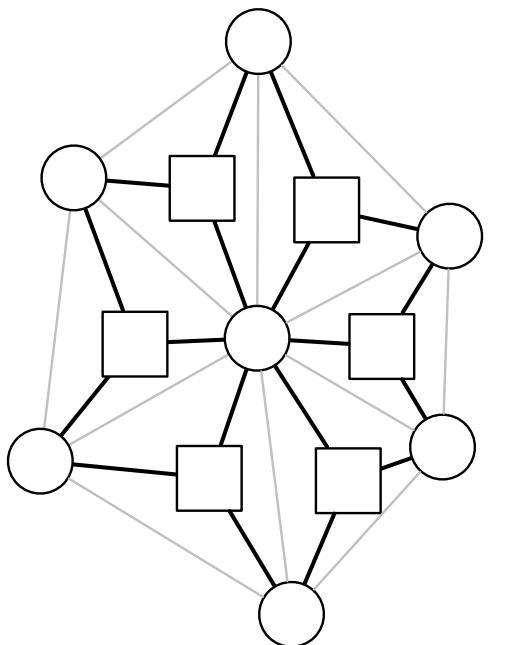
Nonzeros are a subset of the cartesian combination of sets



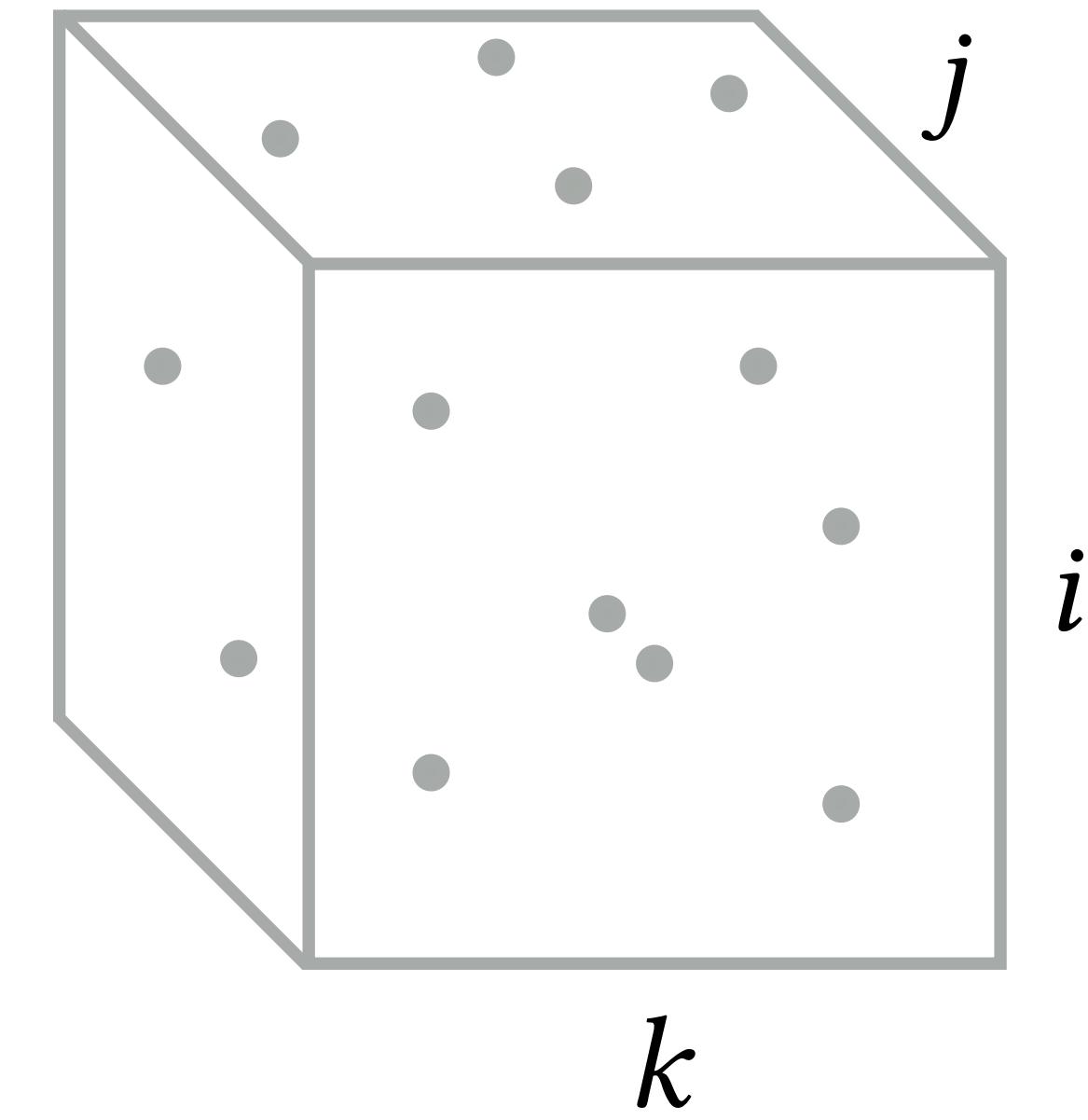
Relations

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

Graphs

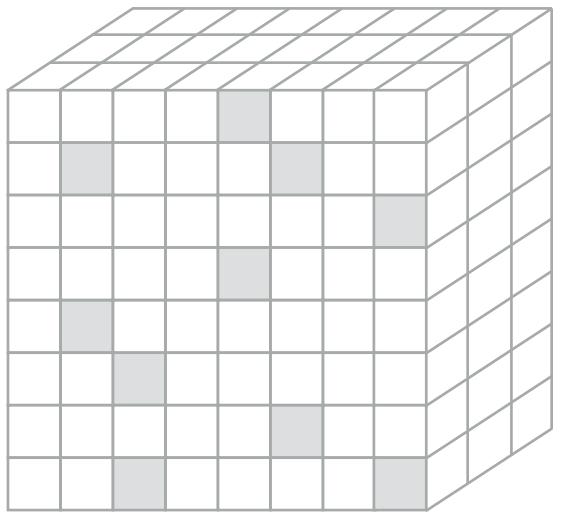


Sparse Iteration Spaces



# Three sparse applications areas

Tensors



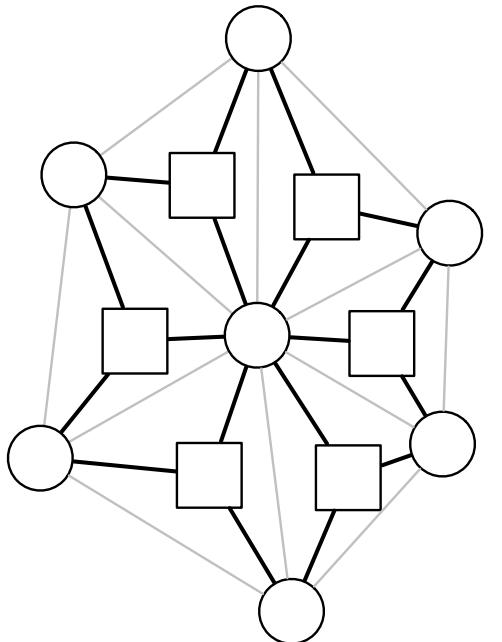
Nonzeros are a subset of the cartesian combination of sets

Relations

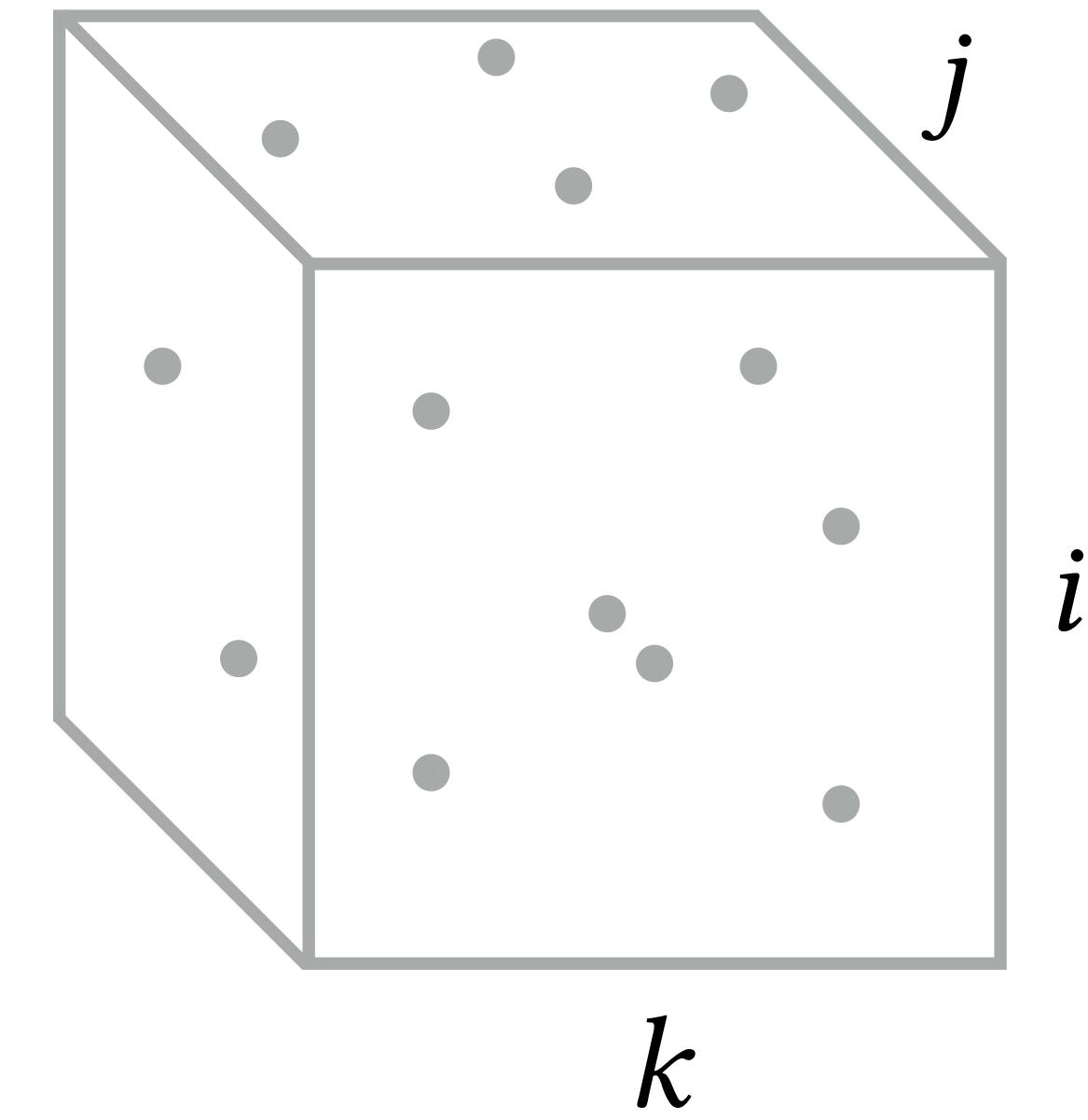
Names	City	Age
Peter	Boston	54
Mary	San Fransisco	35
Paul	New York	23
Adam	Seattle	84
Hilde	Boston	19
Bob	Chicago	76
Sam	Portland	32
Angela	Los Angeles	62

A relation is a subset of the cartesian combination of sets

Graphs

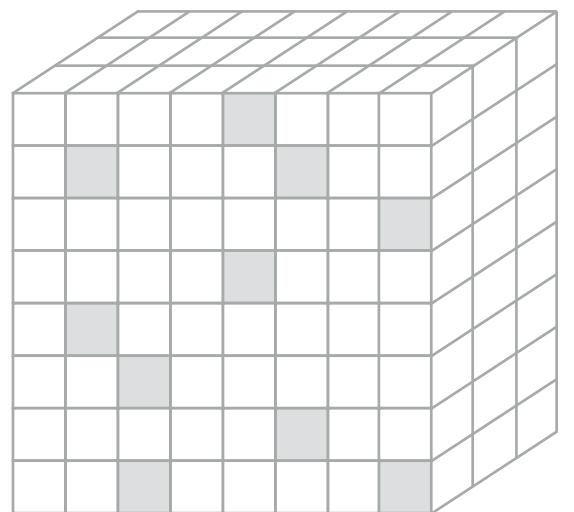


Sparse Iteration Spaces



# Three sparse applications areas

Tensors



Nonzeros are a subset of the cartesian combination of sets



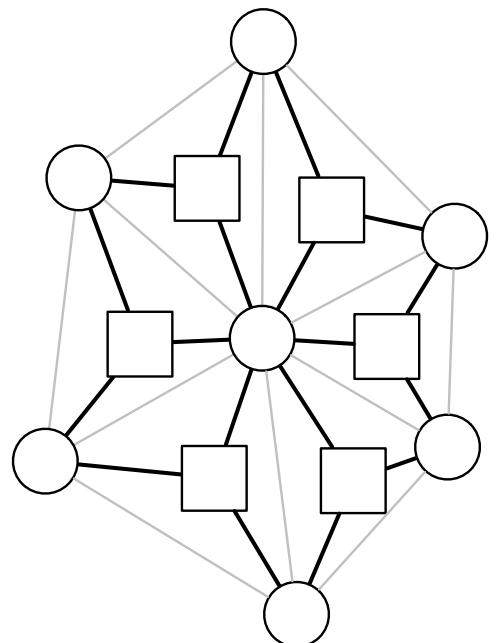
Relations

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



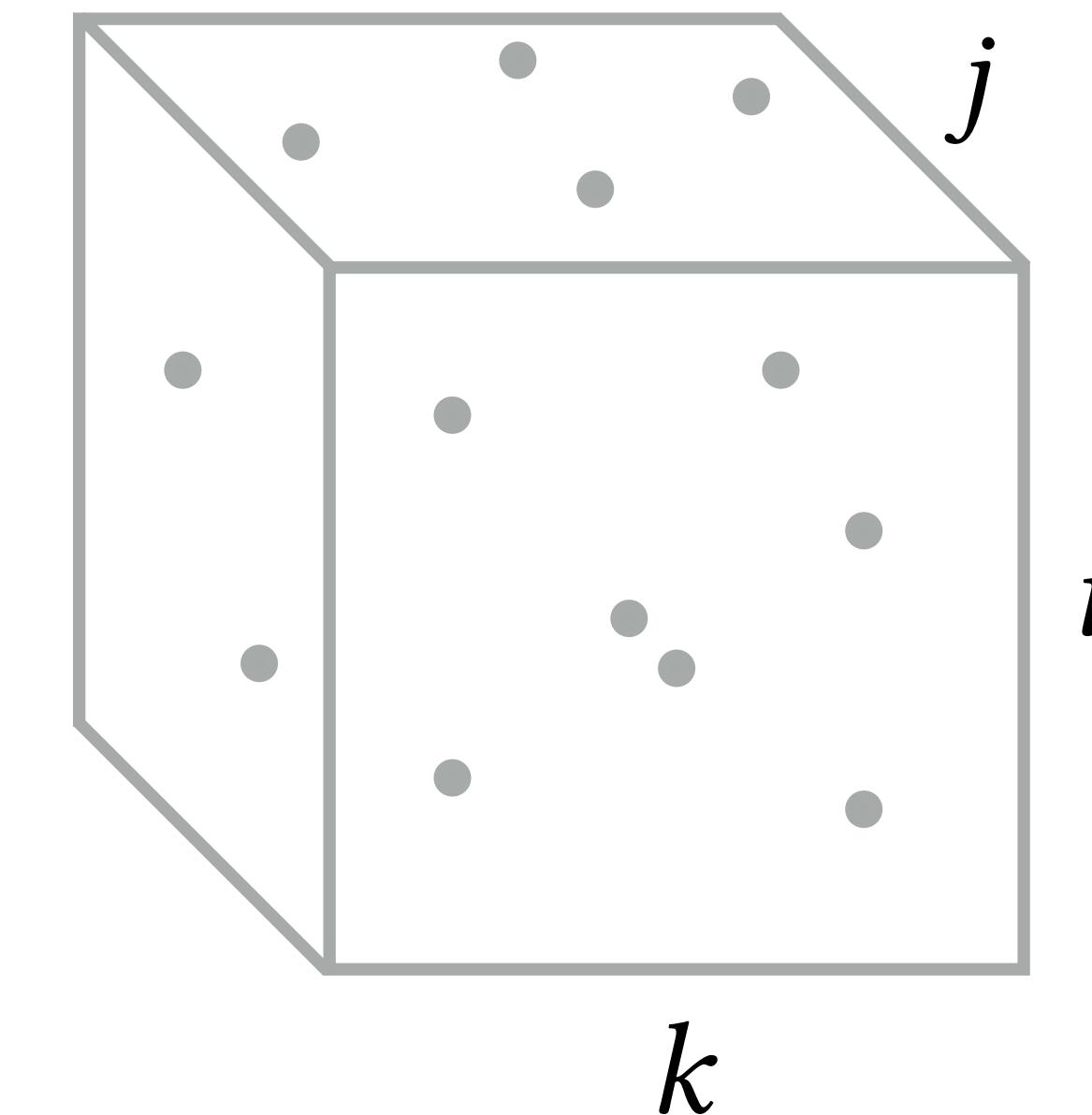
Graphs



Graph edges are a subset of the cartesian combination of sets



Sparse Iteration Spaces

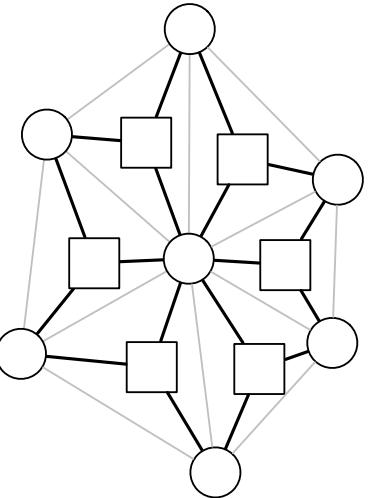


# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

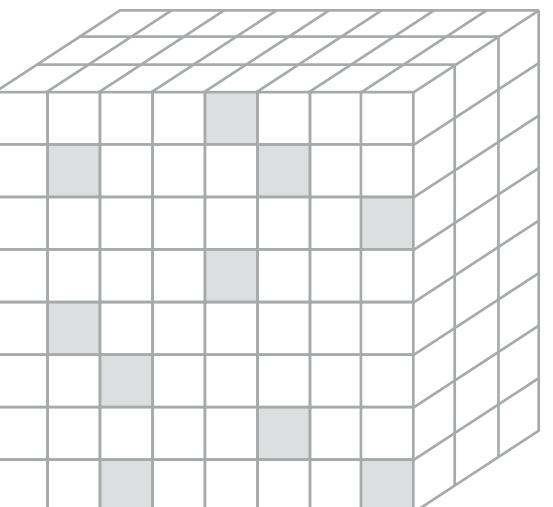
Relations

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

Graphs



Tensors



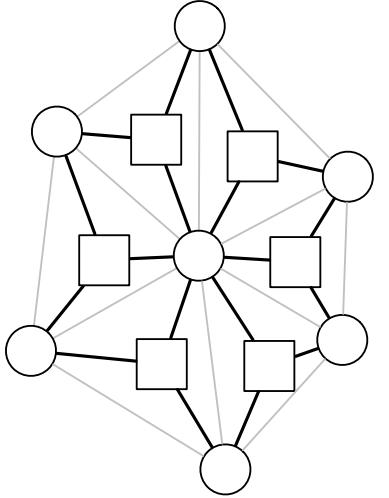
# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

Relations

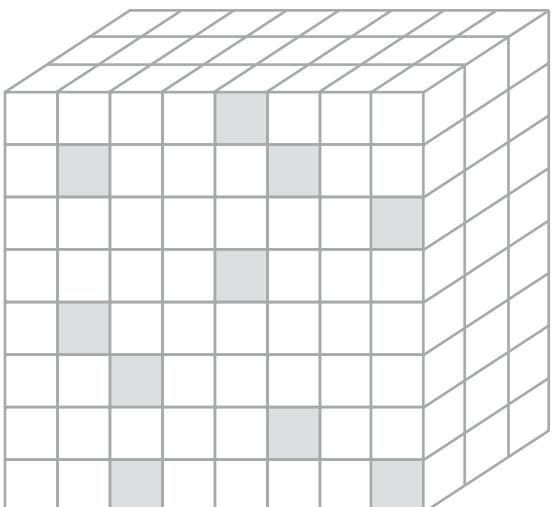
Names	City	Age
Peter	Boston	54
Mary	San Fransisco	35
Paul	New York	23
Adam	Seattle	84
Hilde	Boston	19
Bob	Chicago	76
Sam	Portland	32
Angela	Los Angeles	62

Combine data to  
form systems

Graphs



Tensors



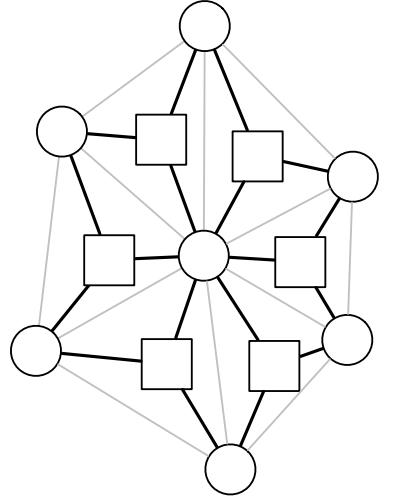
# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

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

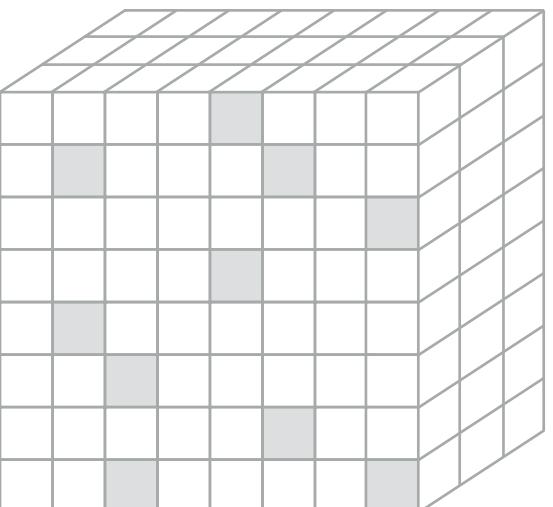
Combine data to  
form systems

Graphs



Local operations  
on systems

Tensors



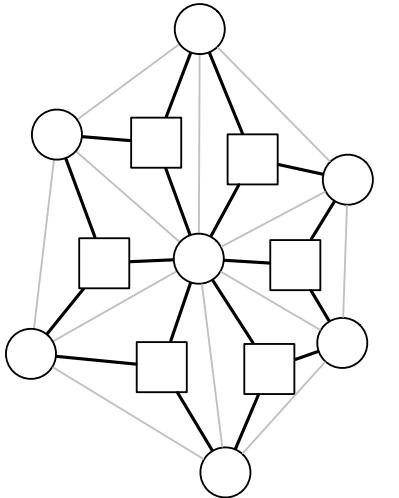
# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

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

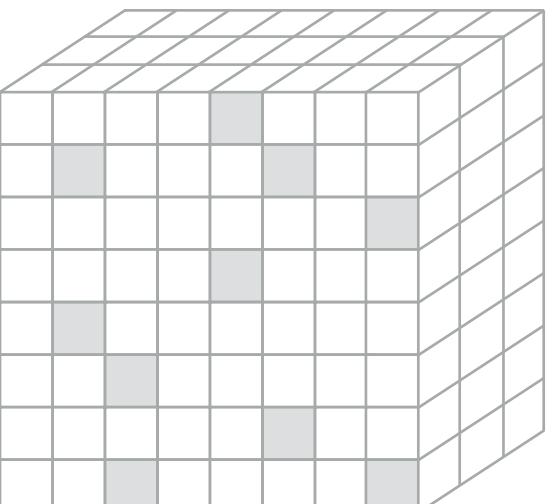
Combine data to  
form systems

Graphs



Local operations  
on systems

Tensors



Global operations  
on systems

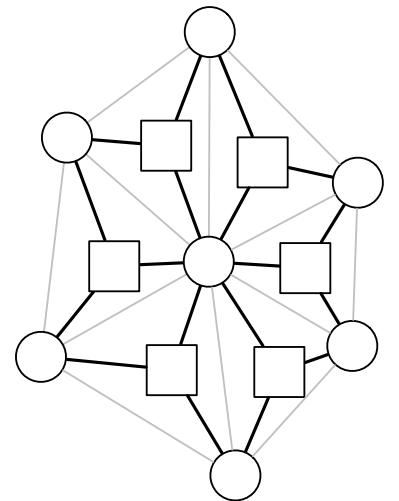
# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

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

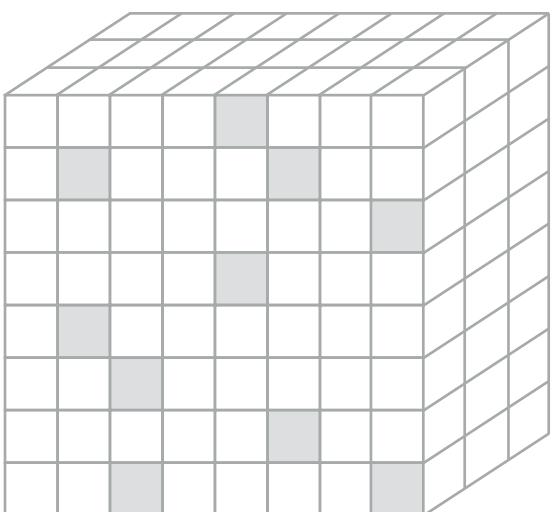
Combine data to form systems

Graphs



Local operations on systems

Tensors



Global operations on systems

Relations

Filters

Solves

Dijkstra's Algorithm

Tensor

Graphs

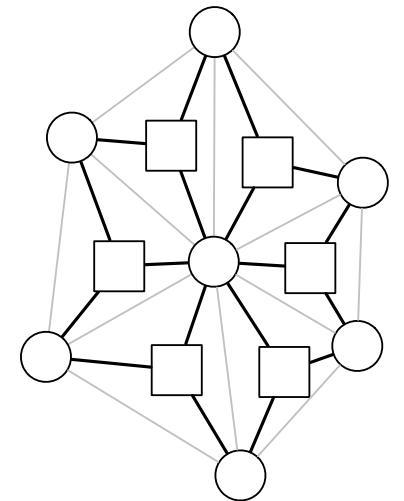
# Relations, graphs, and tensors share a lot of structure but are specialized for different purposes

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

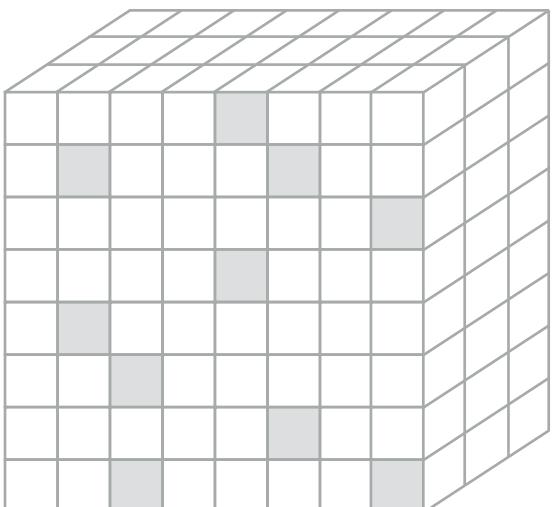
Combine data to form systems

Graphs



Local operations on systems

Tensors



Global operations on systems

Relations

Filters

Solves

Pagerank  
Triangle Counting

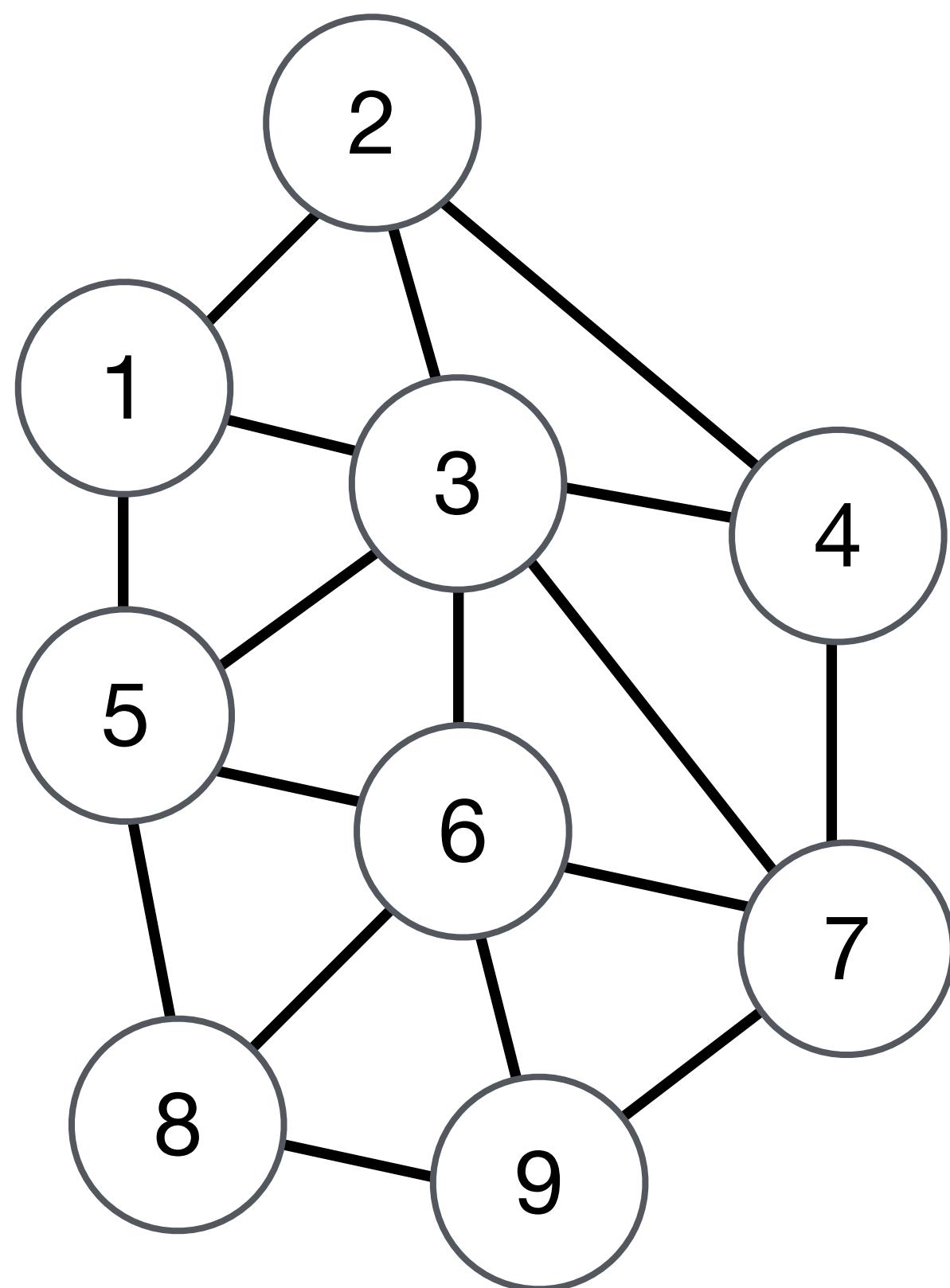
Tensor

Dijkstra's Algorithm

Graphs

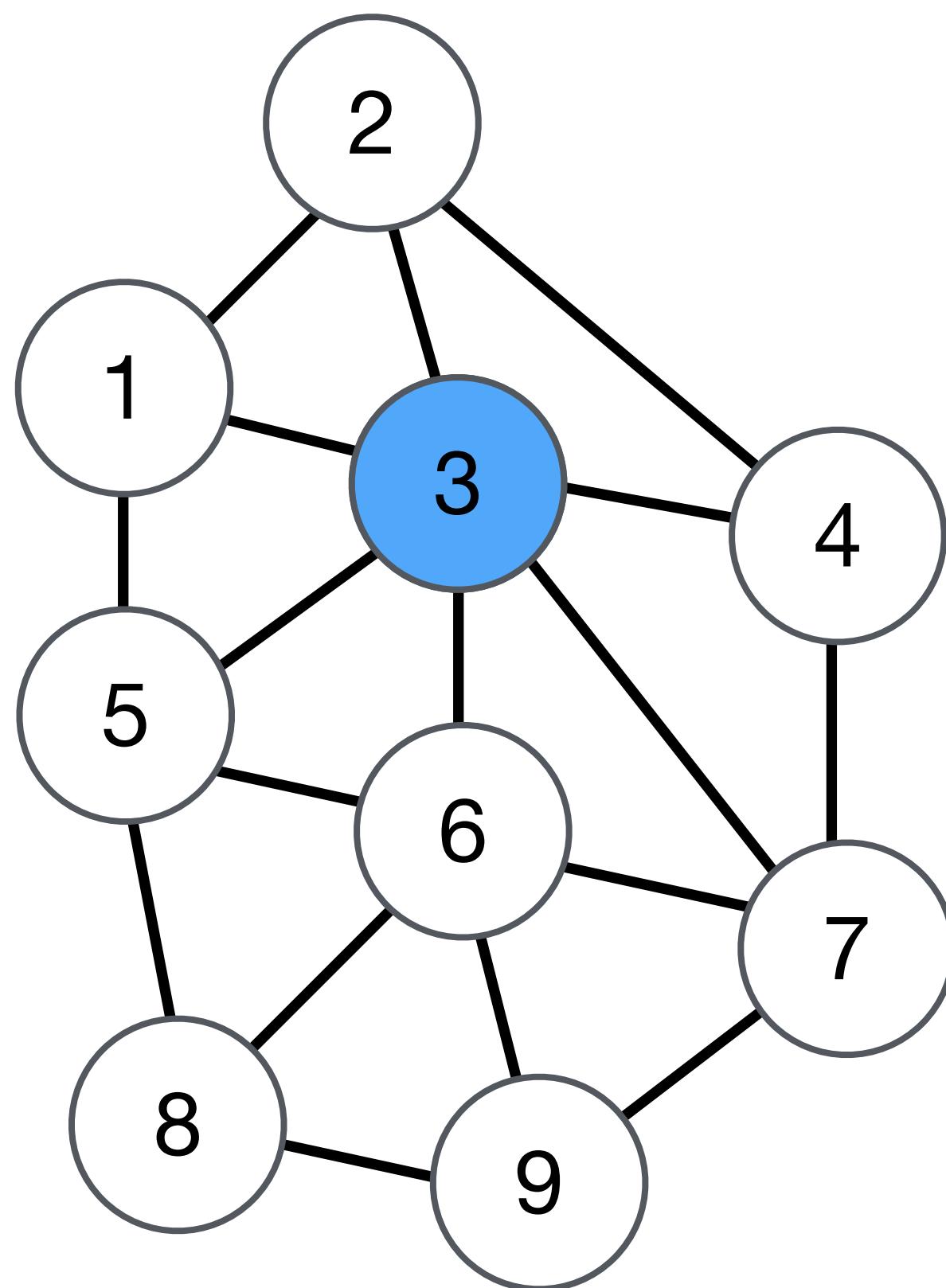
# Triangle counting on graphs, relations, and tensors

On graphs



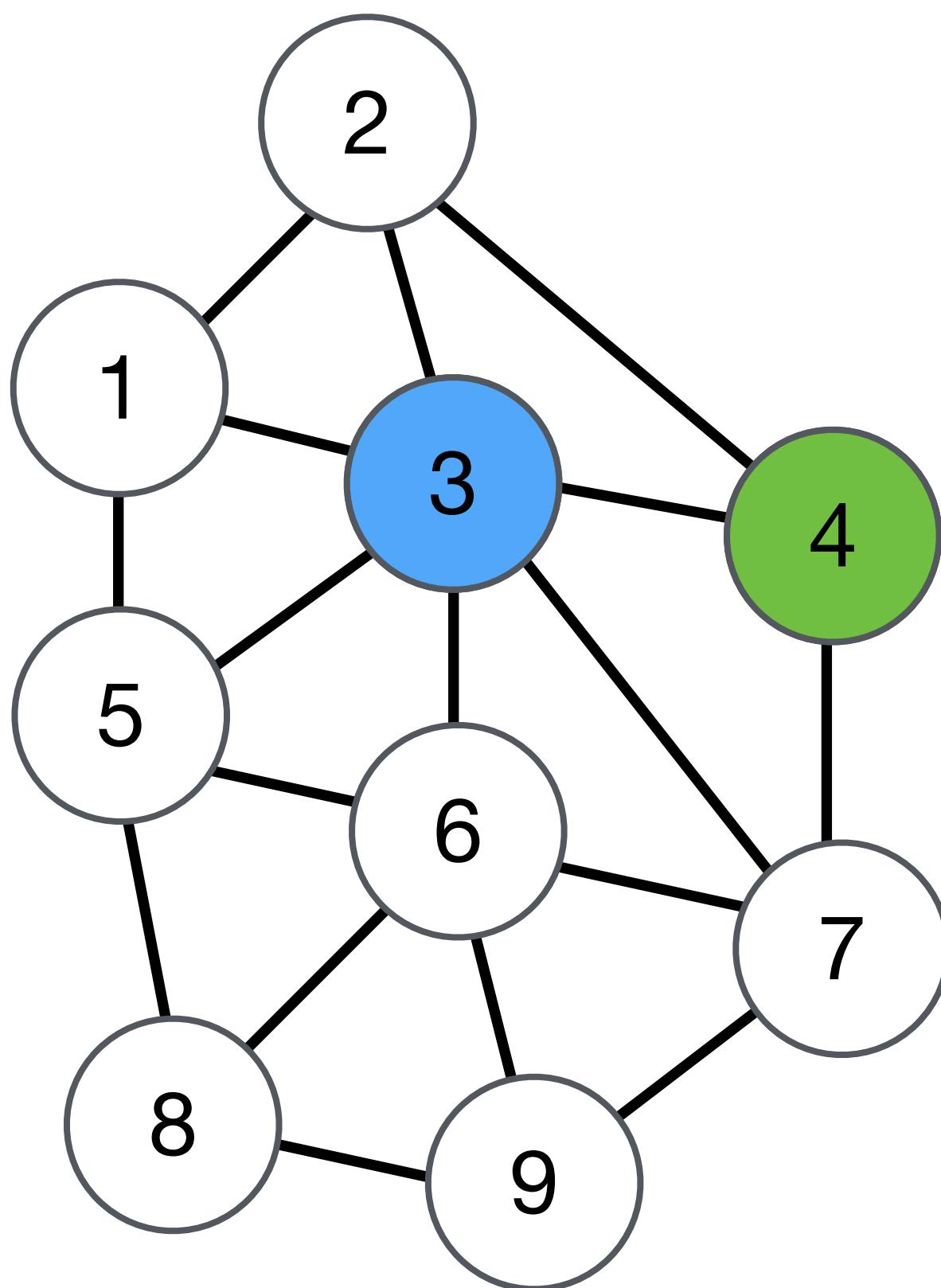
# Triangle counting on graphs, relations, and tensors

On graphs



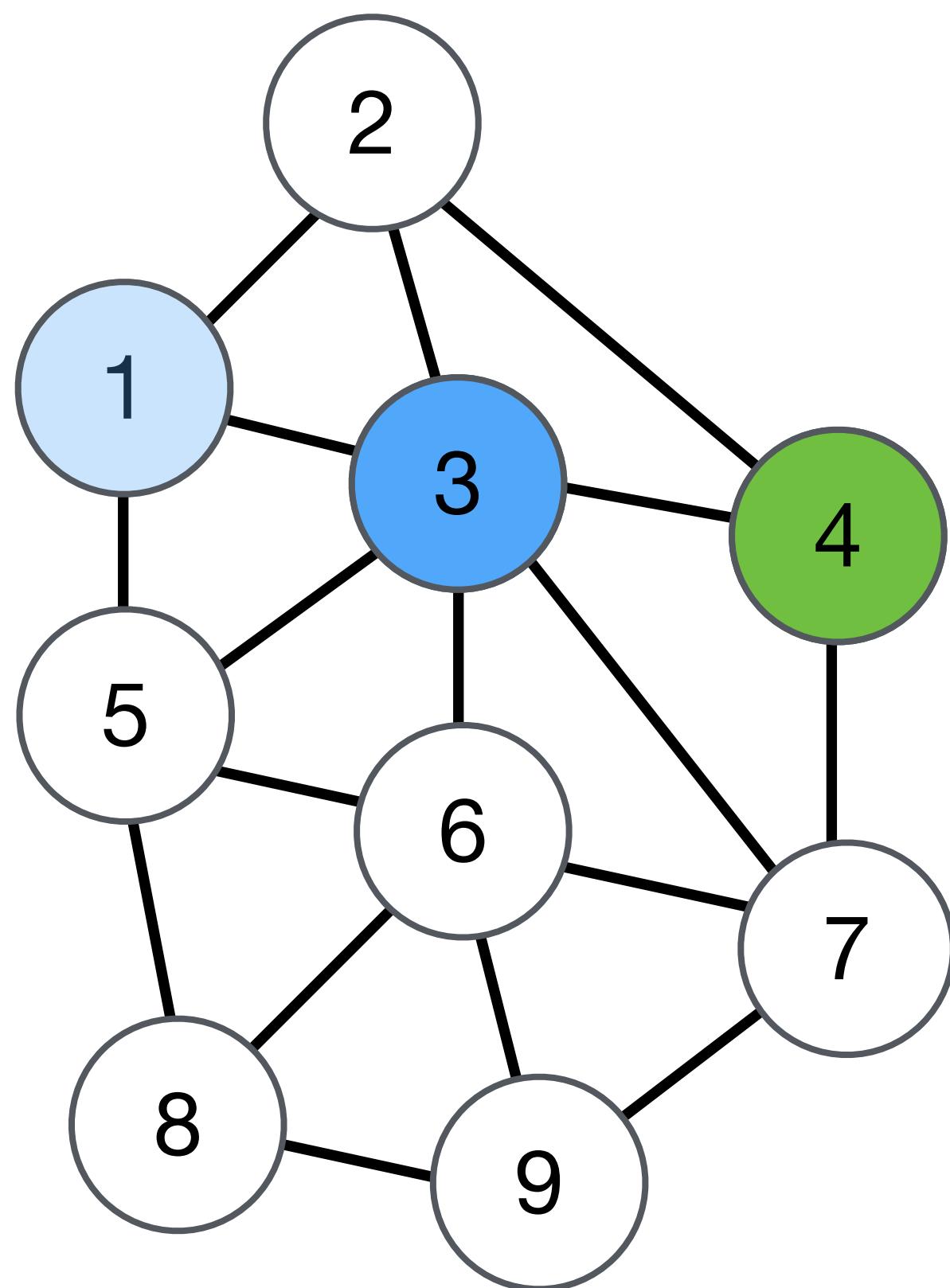
# Triangle counting on graphs, relations, and tensors

On graphs



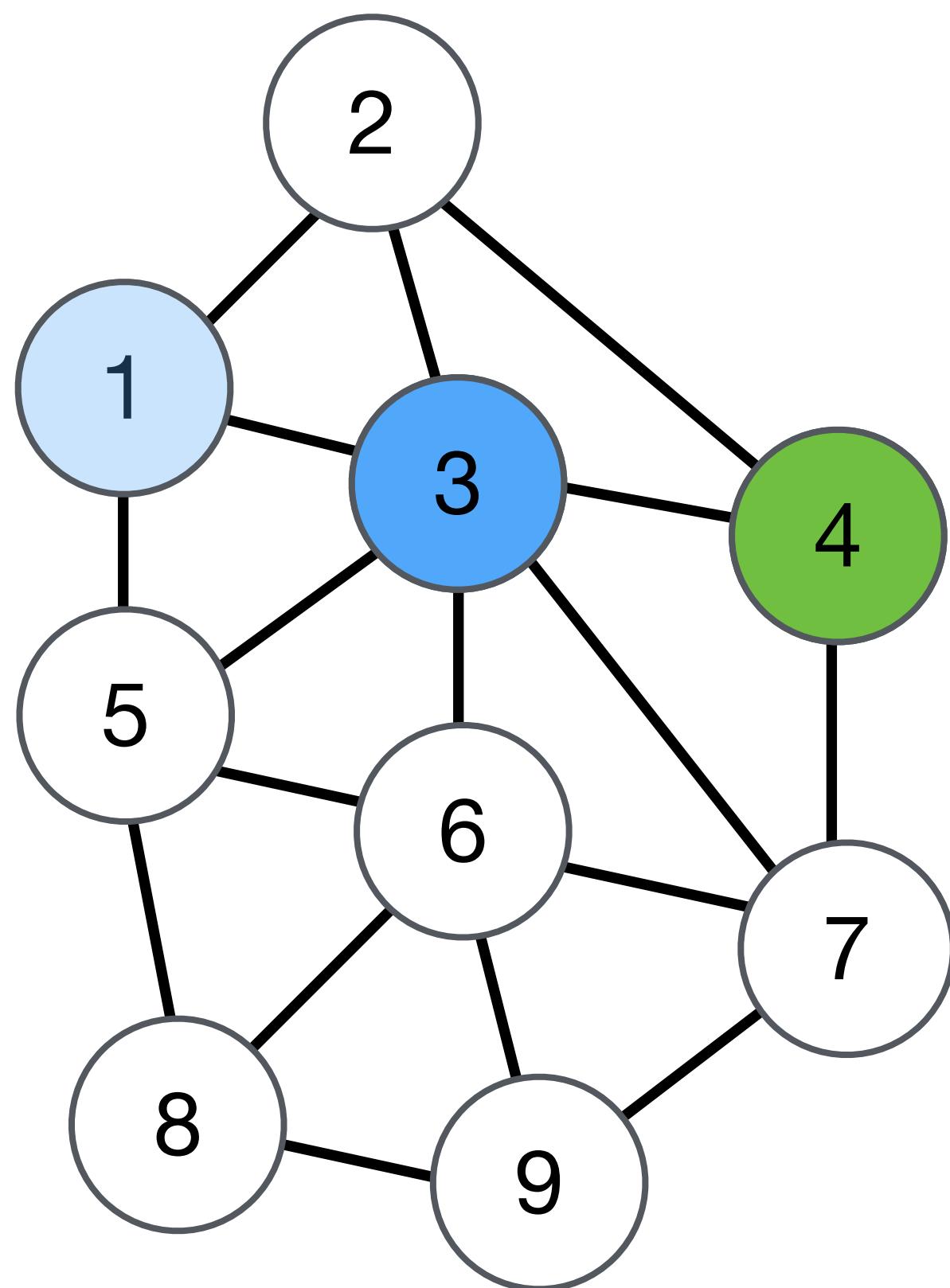
# Triangle counting on graphs, relations, and tensors

On graphs



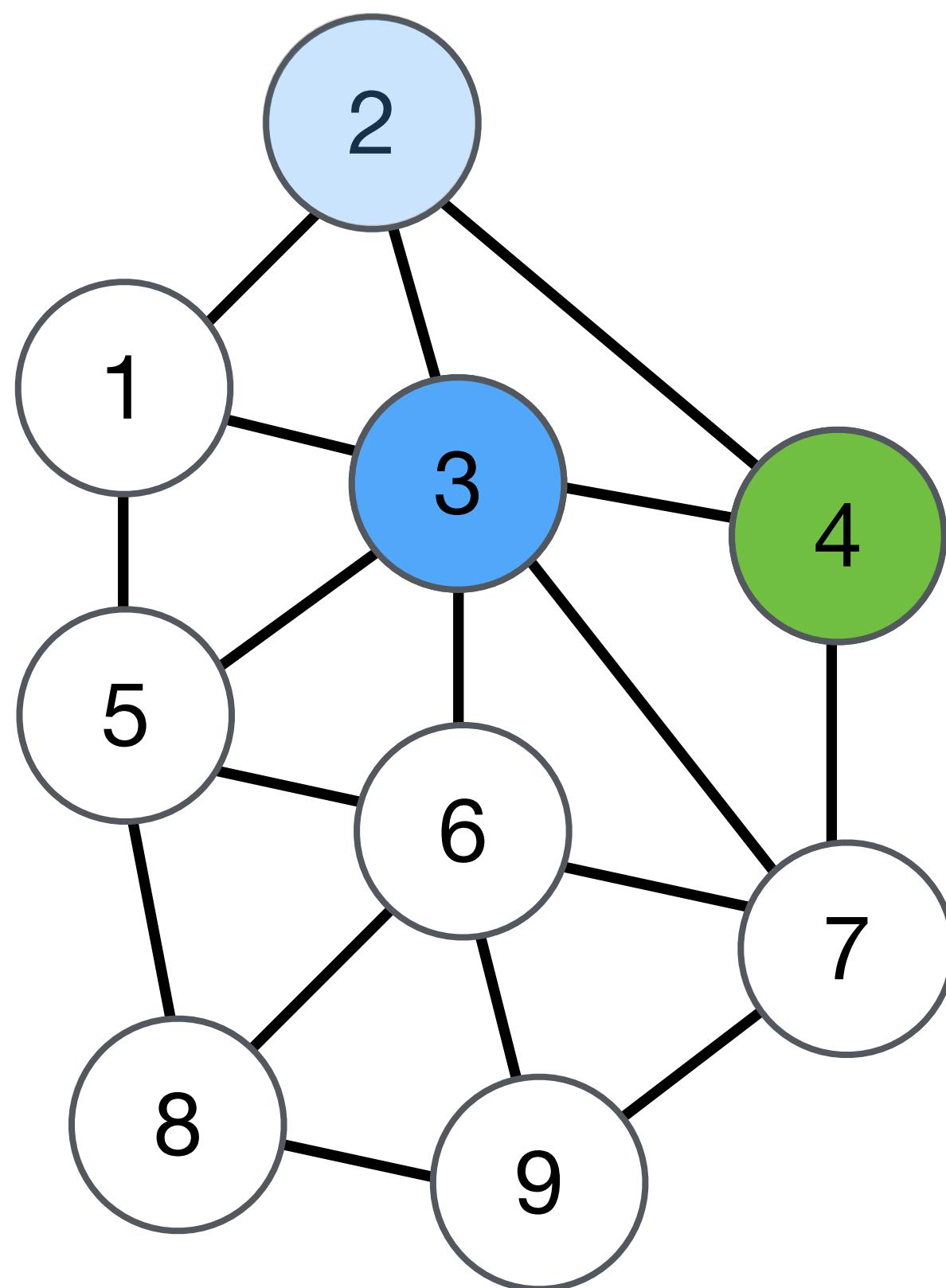
# Triangle counting on graphs, relations, and tensors

On graphs



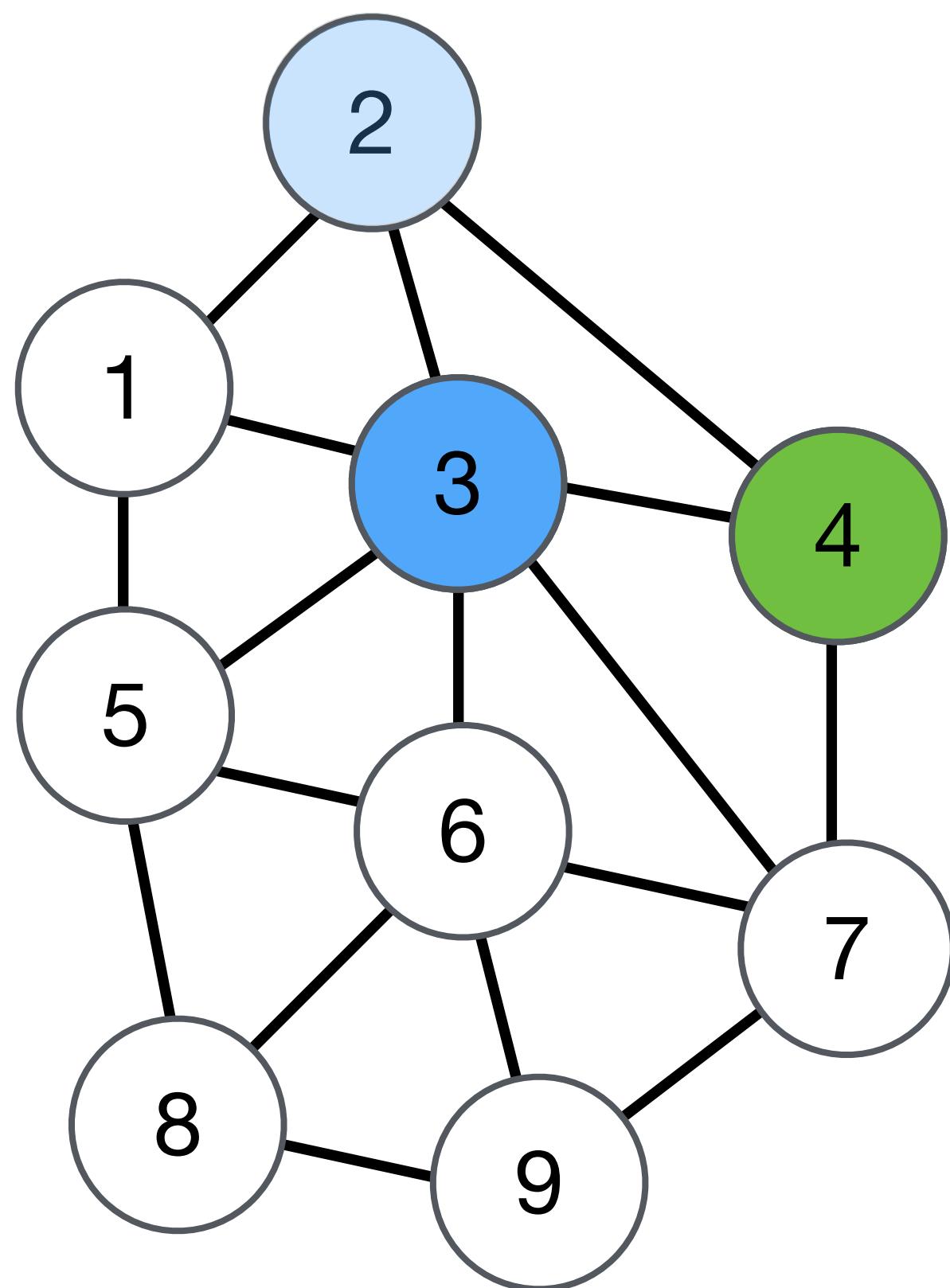
# Triangle counting on graphs, relations, and tensors

On graphs



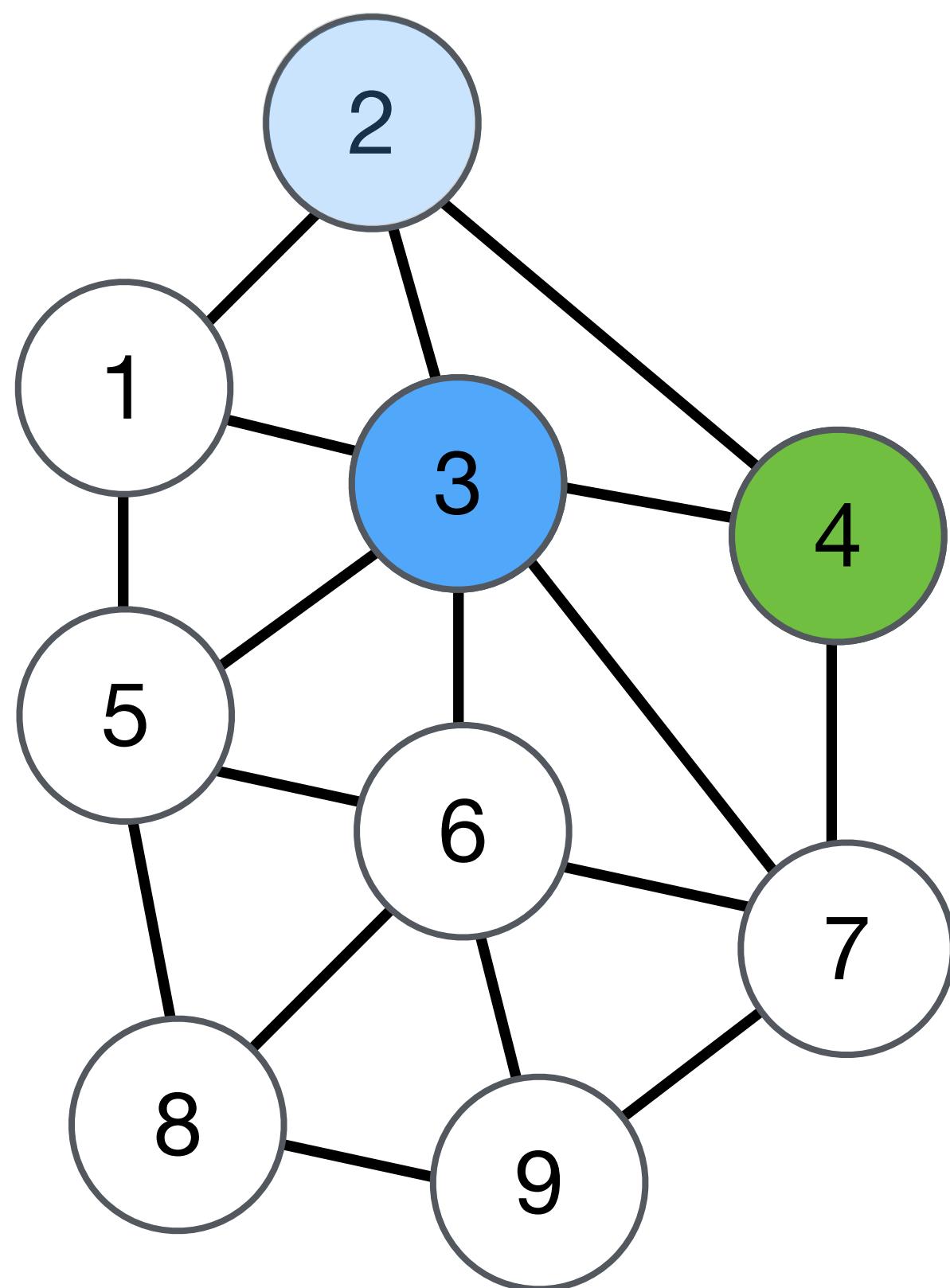
# Triangle counting on graphs, relations, and tensors

On graphs



# Triangle counting on graphs, relations, and tensors

On graphs

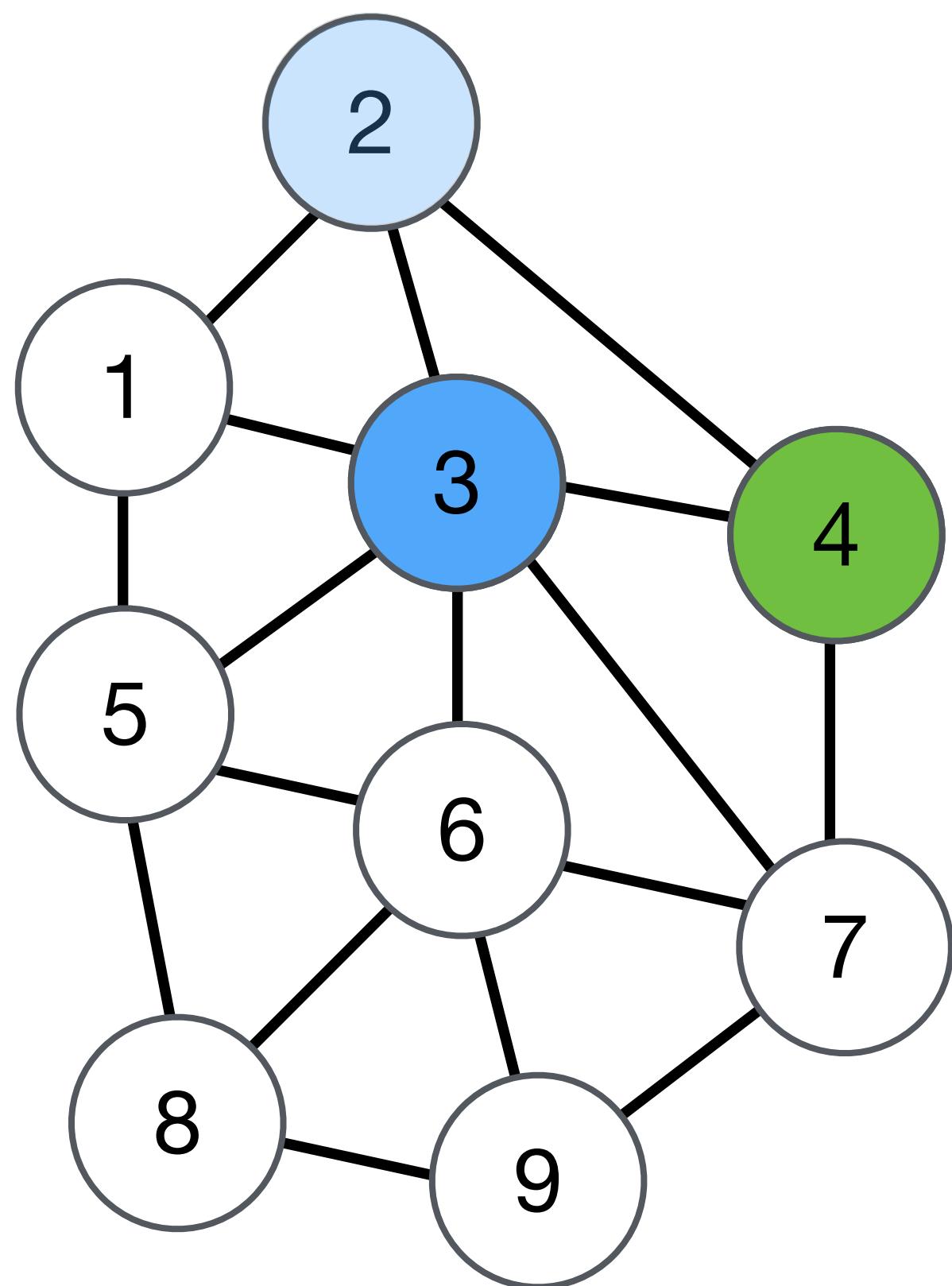


On relations

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

# Triangle counting on graphs, relations, and tensors

On graphs

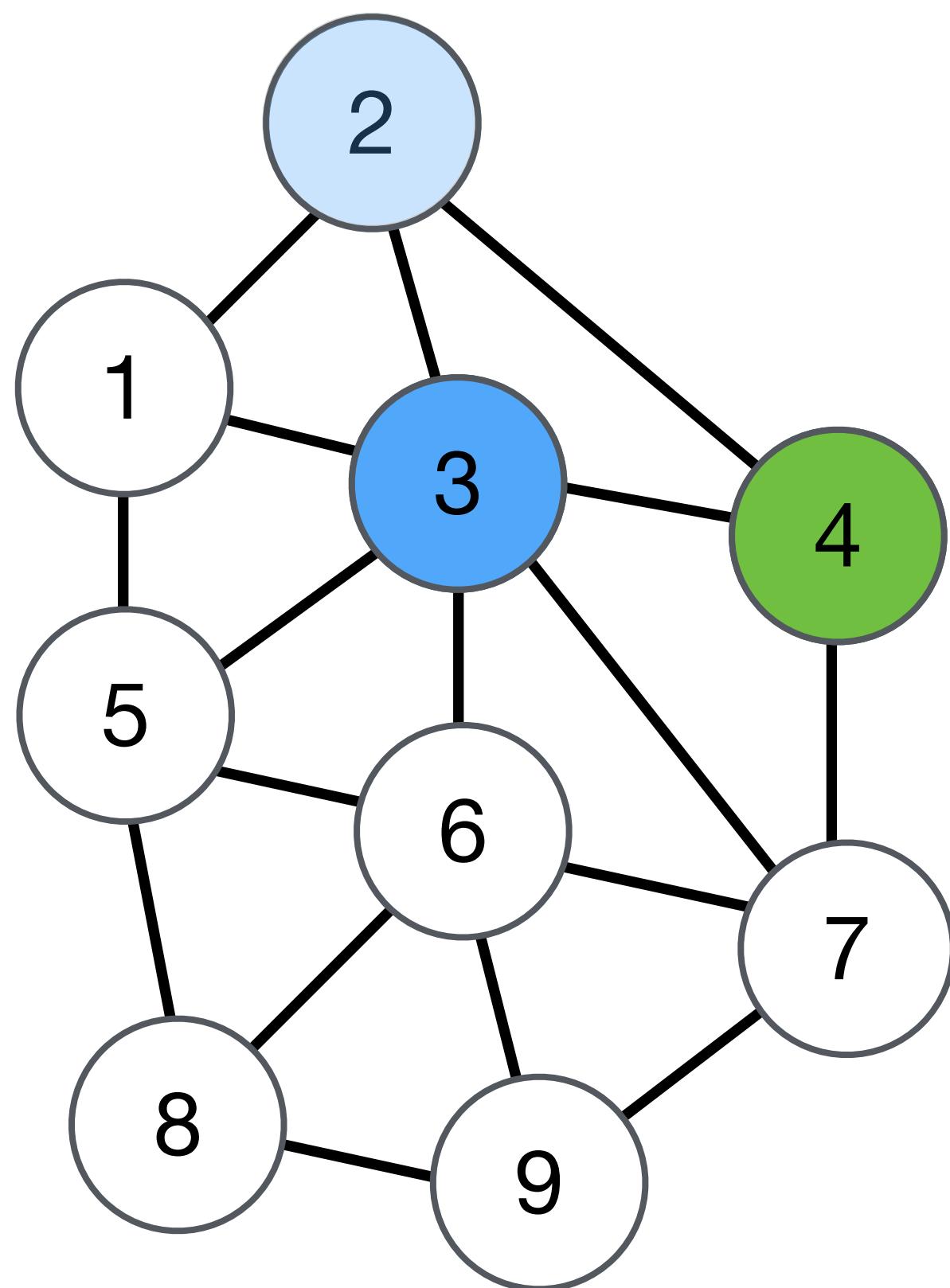


On relations

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

# Triangle counting on graphs, relations, and tensors

On graphs

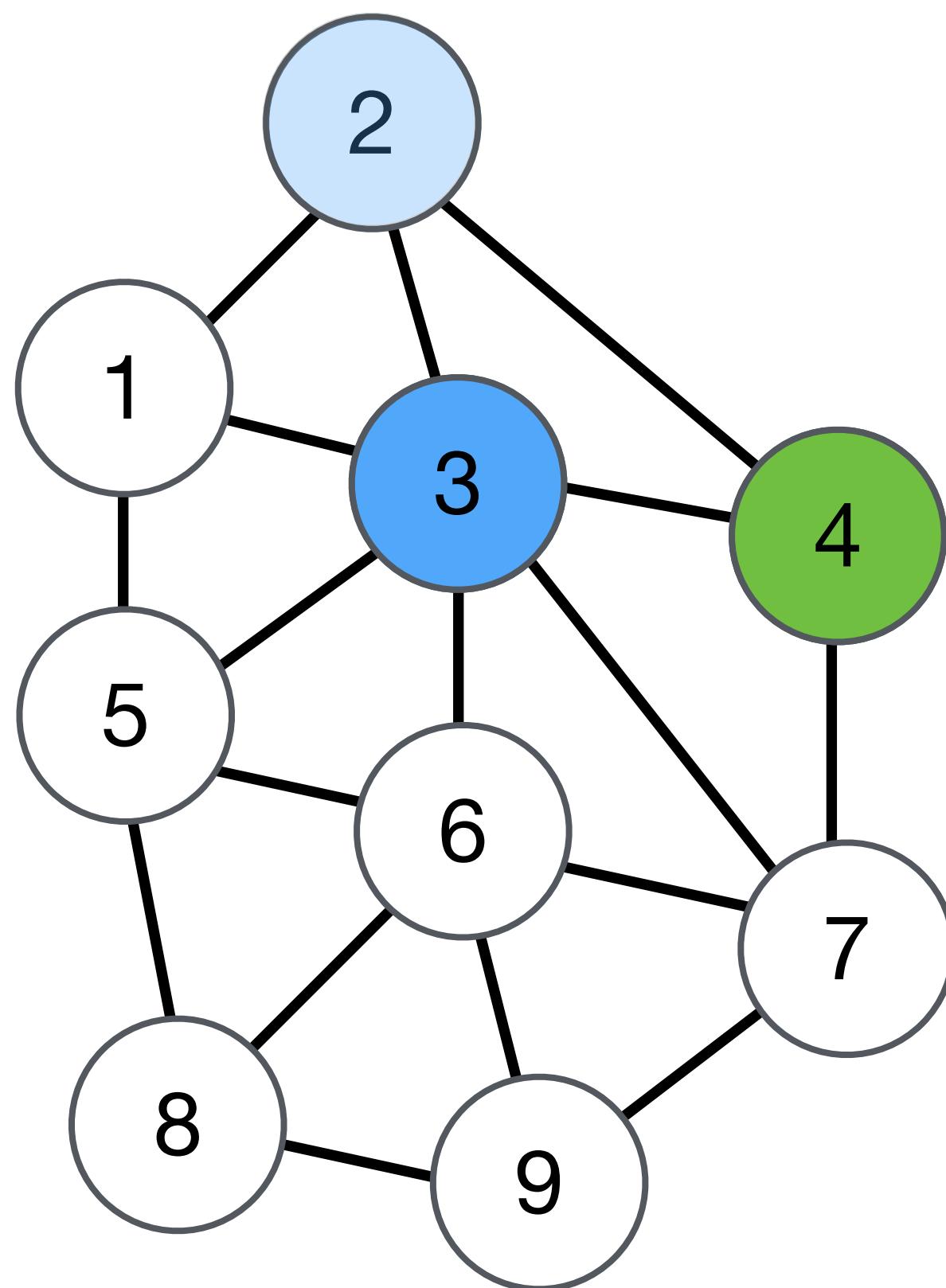


On relations

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

# Triangle counting on graphs, relations, and tensors

On graphs

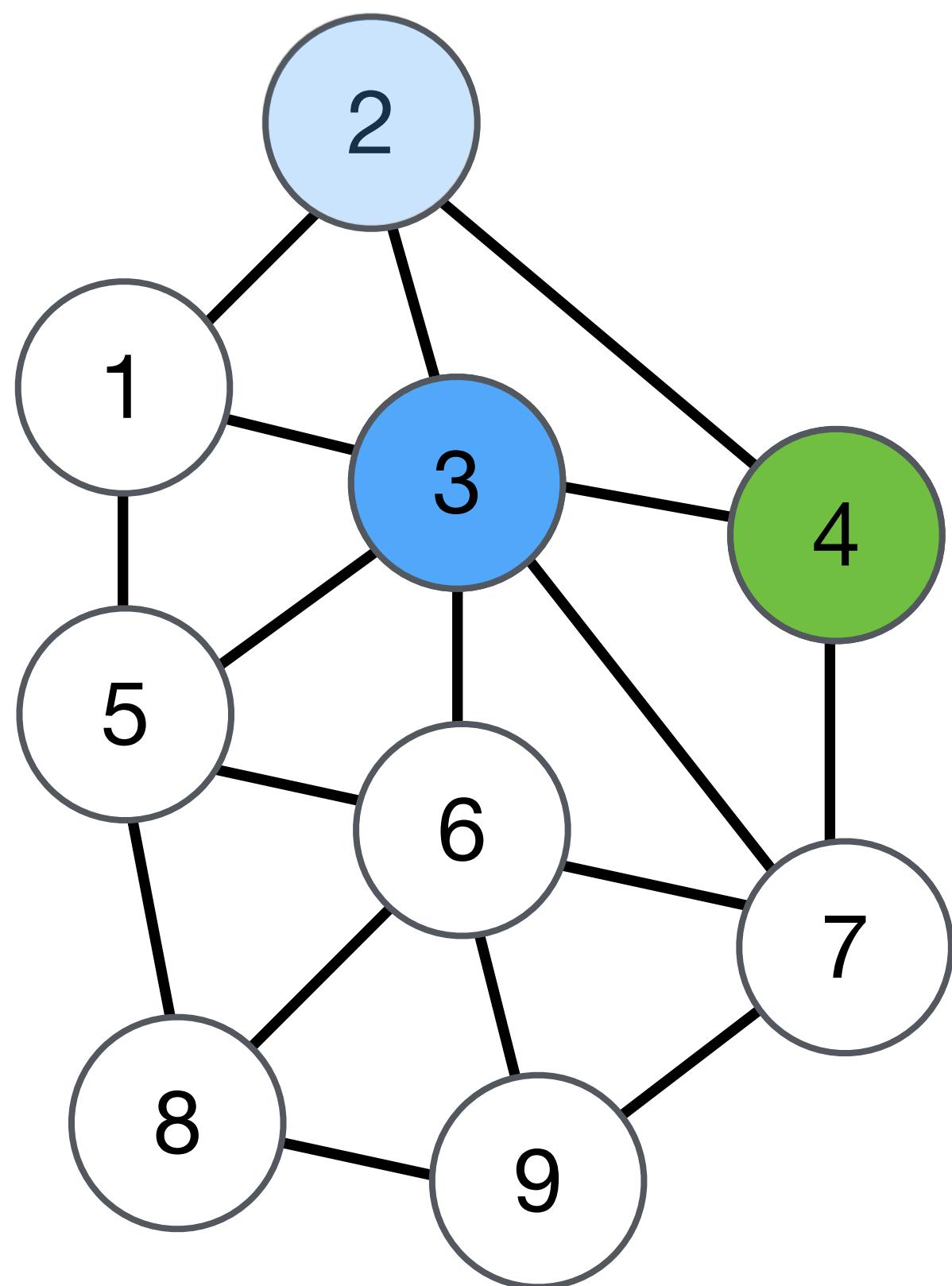


On relations

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

# Triangle counting on graphs, relations, and tensors

On graphs



On relations

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

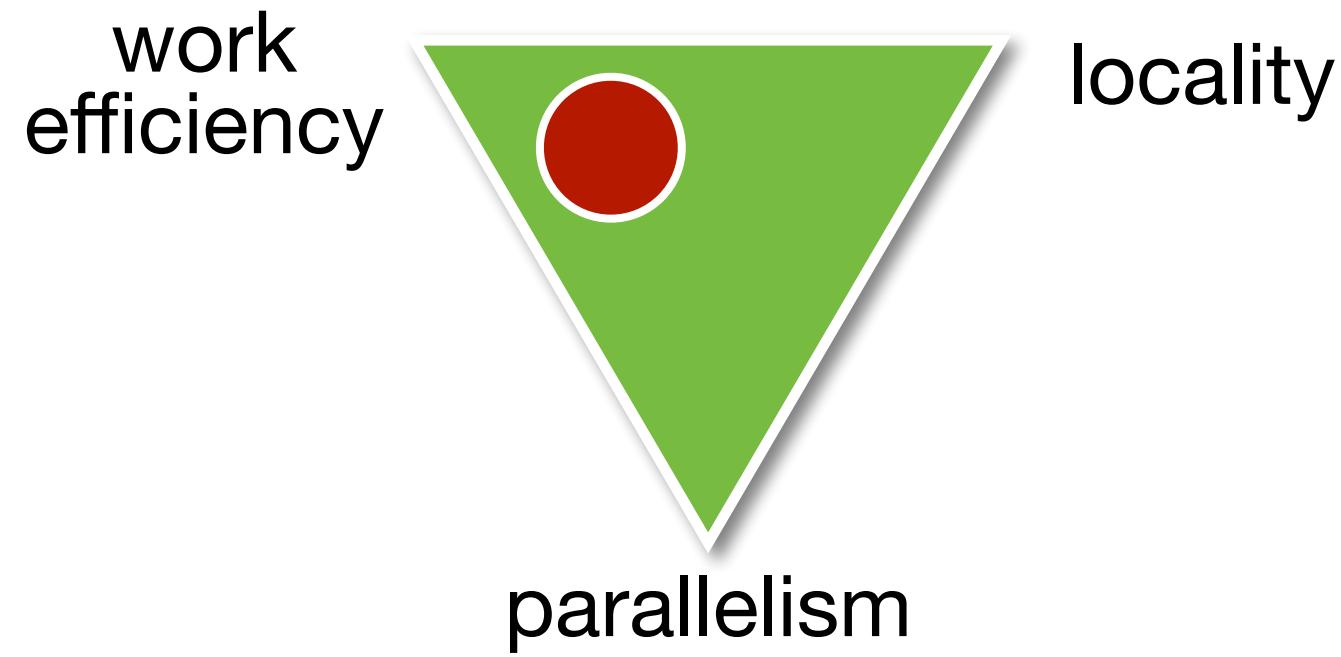
On tensors

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

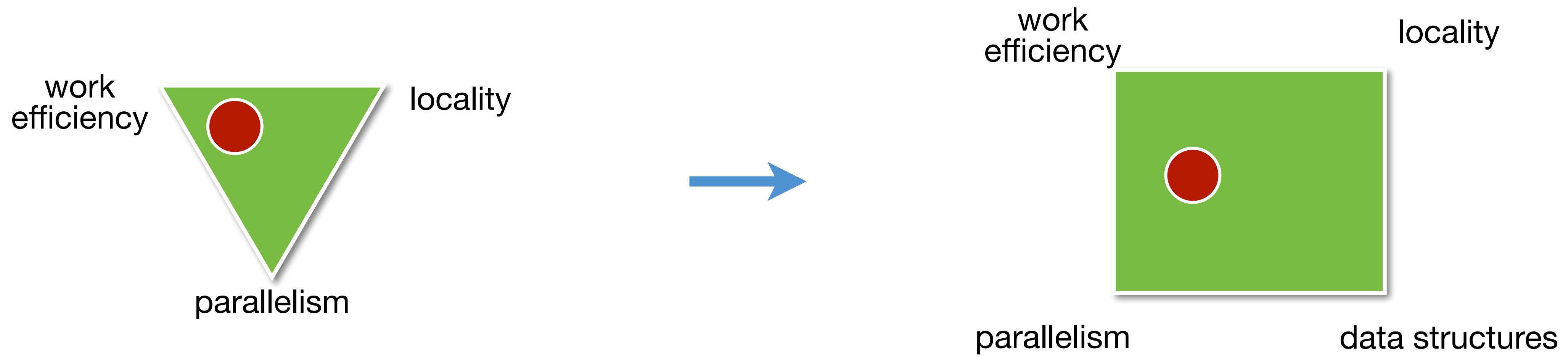
# Some important developments in libraries, languages, and compilers for sparse computations

- 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 sparse tensor algebra

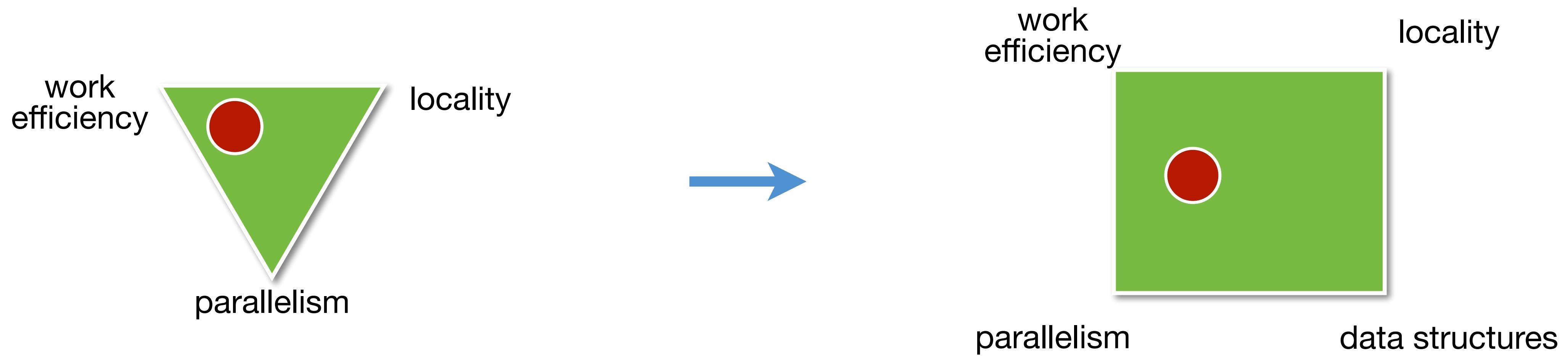
Parallelism, locality, work efficiency still matters,  
but the key is choosing efficient data structures



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

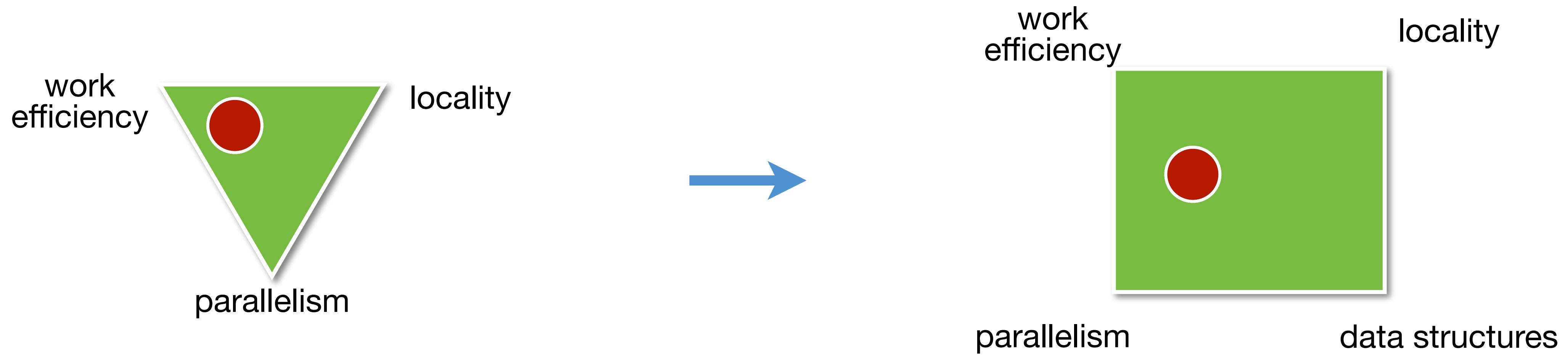


# Parallelism, locality, work efficiency still matters, but the key is choosing efficient data structures



Harry	CS
Sally	EE
George	CS
Mary	ME
Rita	CS

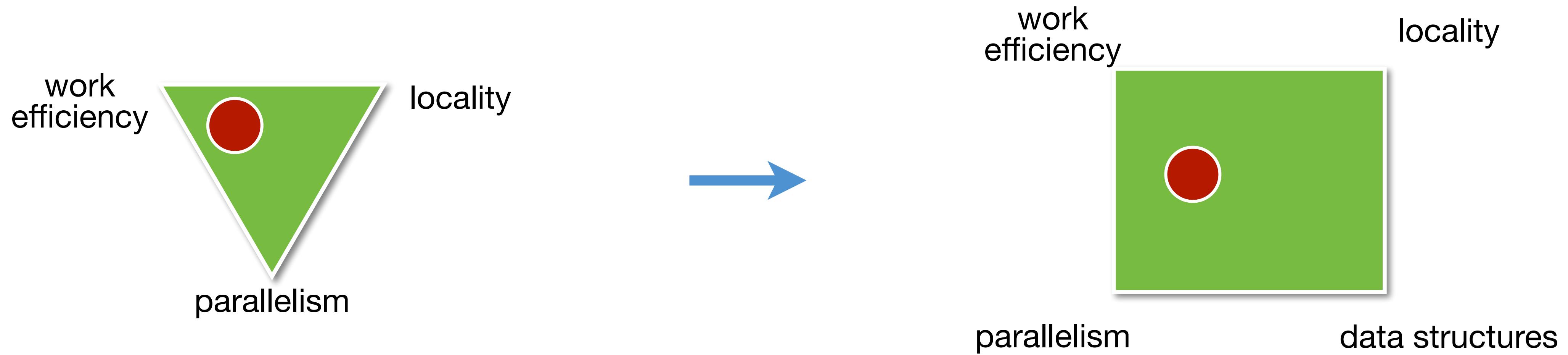
# Parallelism, locality, work efficiency still matters, but the key is choosing efficient data structures



Harry	CS
Sally	EE
George	CS
Mary	ME
Rita	CS

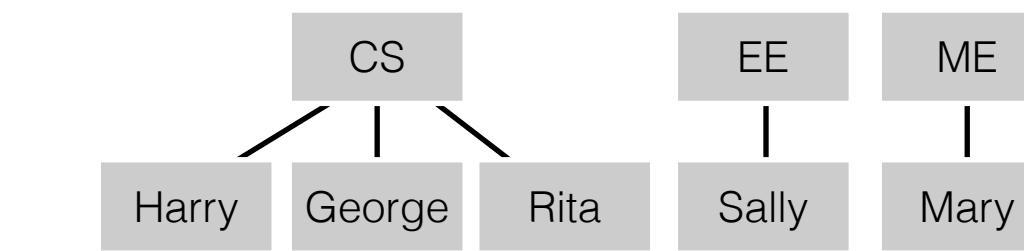
Harry	Sally	George	Mary	Rita
CS	EE	CS	ME	CS

# 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



# 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 <sub>1</sub> ,v <sub>5</sub> )
(2,3)	(0,5)	(Sally,EE)	(v <sub>4</sub> ,v <sub>3</sub> )
(5,5)	(7,5)	(George,CS)	(v <sub>5</sub> ,v <sub>3</sub> )
		(Rita,CS)	(v <sub>3</sub> ,v <sub>5</sub> )
		(Mary,ME)	(v <sub>3</sub> ,v <sub>1</sub> )

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

0		2	3
0	A	B	
1		C	D
2			E

A		B			C	D	E			F
0		2	3	4	5	6	7	8	9	10

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

A	B	C	D	E	F
0		2	3	4	5

0	A		B	
1		C	D	E
2				F

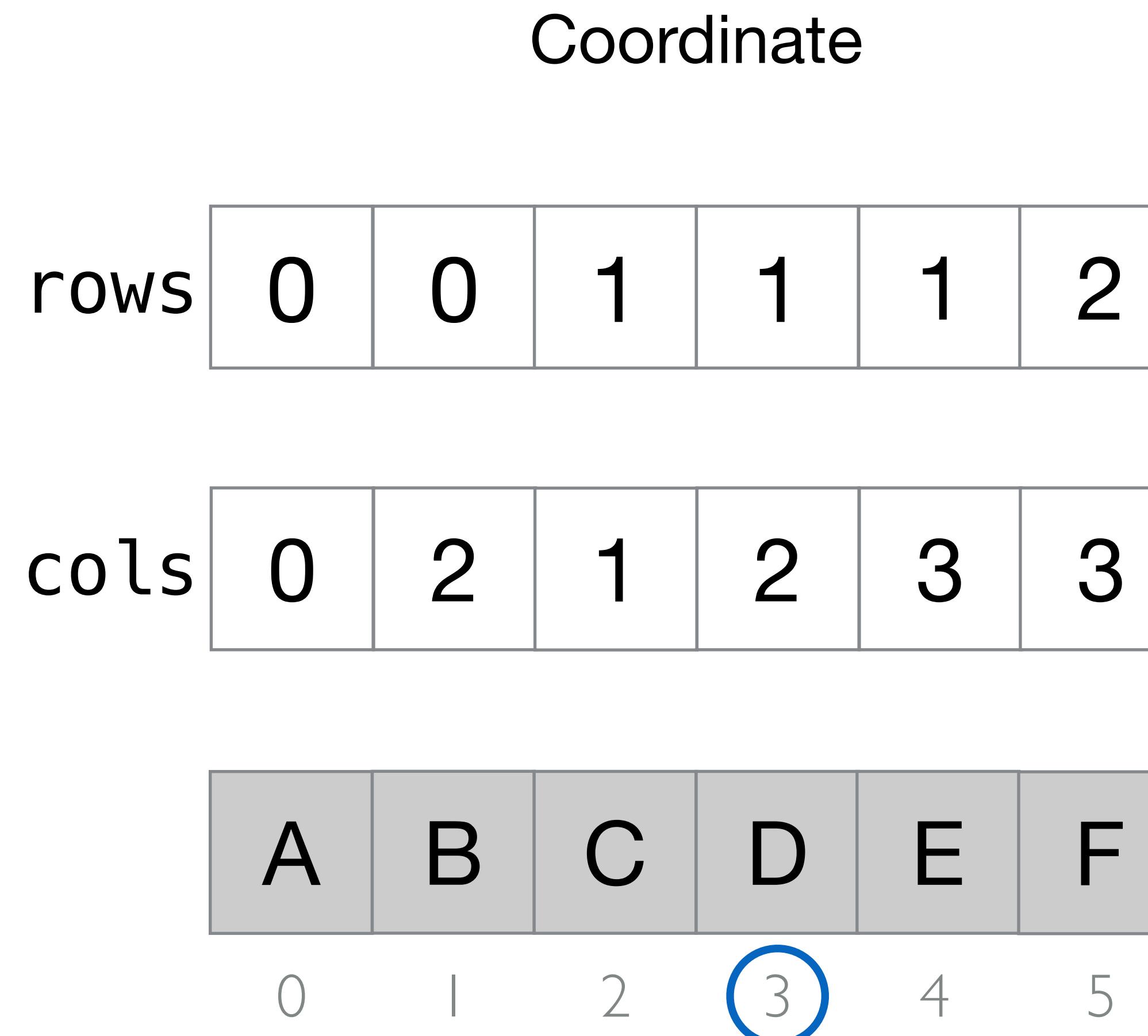
Hierarchically compressed data structures (tries)  
reduce the number of values that need to be stored

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

	0		2	3
0	A		B	
		C	D	E
2				F



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



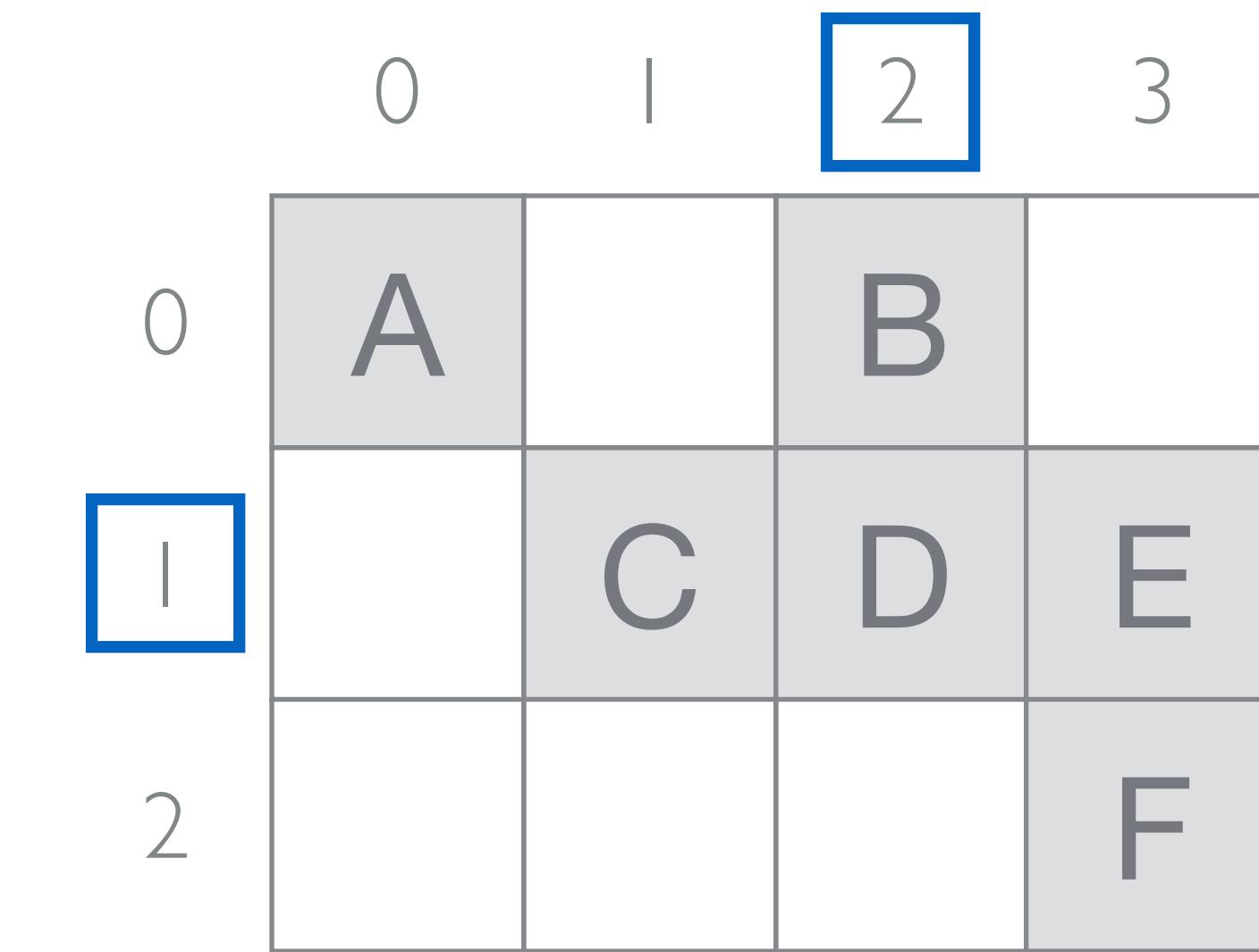
0	A		B	
1		C	D	E
2				F

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

Coordinate

rows	Coordinate					
0	0	1	1	1	2	

cols	Coordinate					
0	2	1	2	3	3	



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

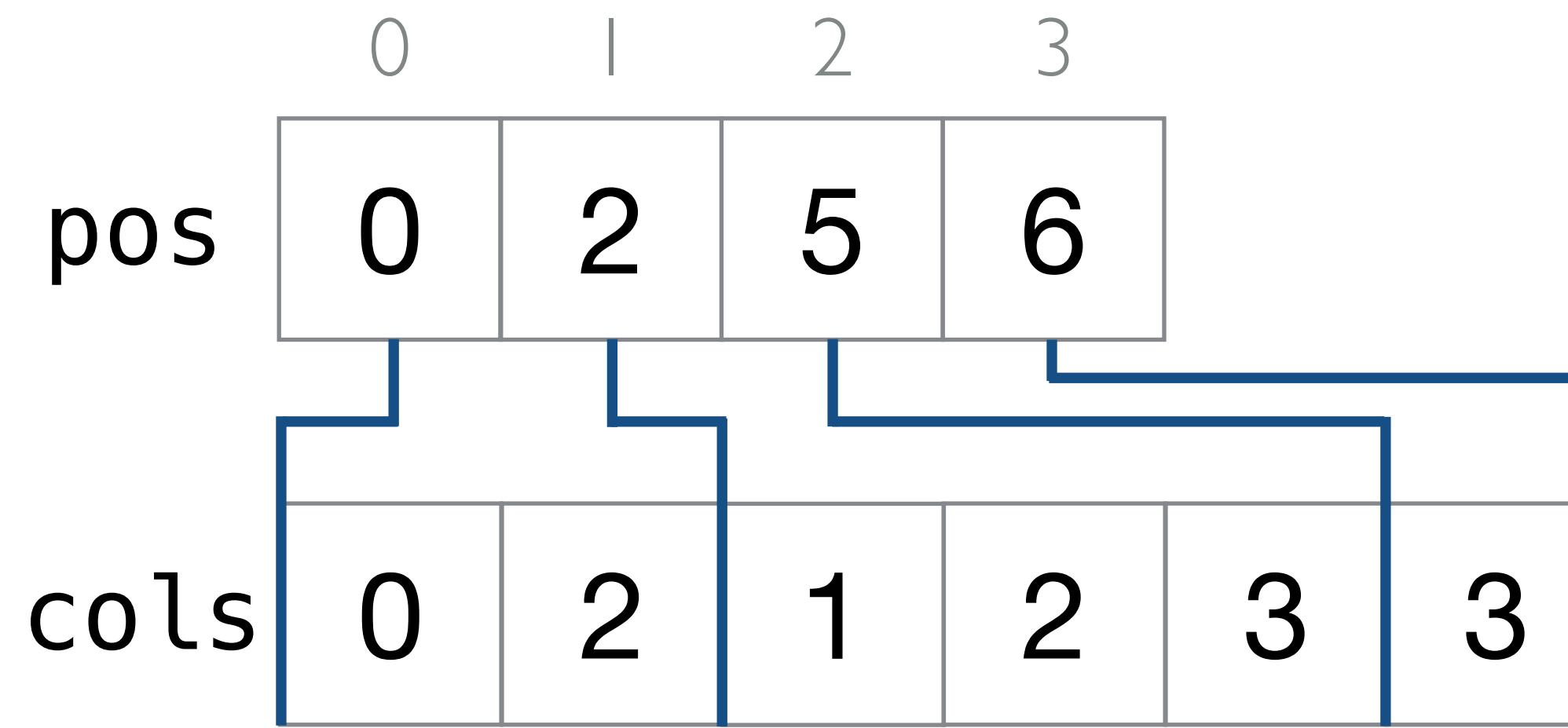
		Coordinate			Duplicates
		rows	0	1	2
cols	0	0	1	1	2

0	A	B		
1	C	D	E	
2				F



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

Compressed Sparse Rows (CSR)



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

A	B	C	D	E	F
0		2	3	4	5

Iteration over sparse iteration spaces imply coiteration  
over sparse data structures

Linear Algebra:  $A = B + C$

# Iteration over sparse iteration spaces imply coiteration over sparse data structures

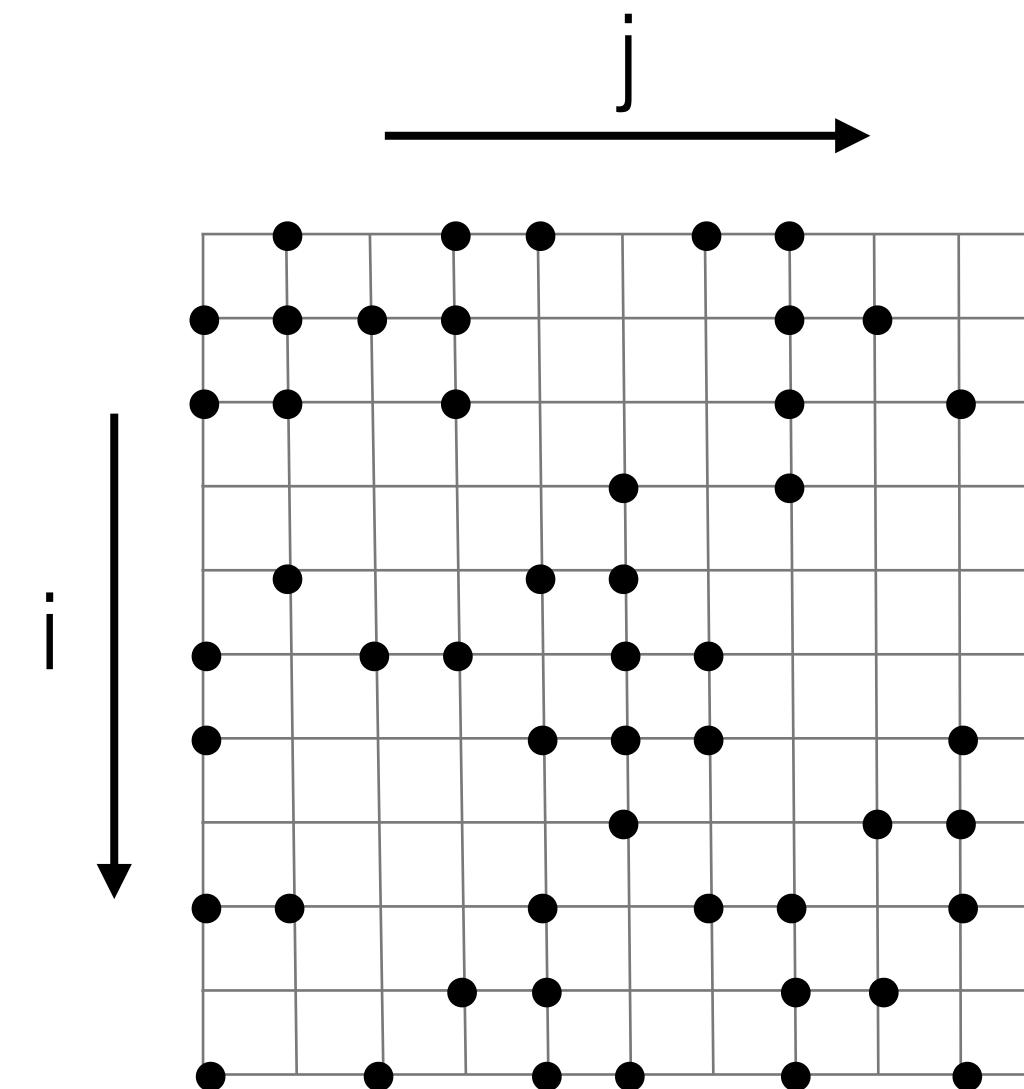
Linear Algebra:  $A = B + C$

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

# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

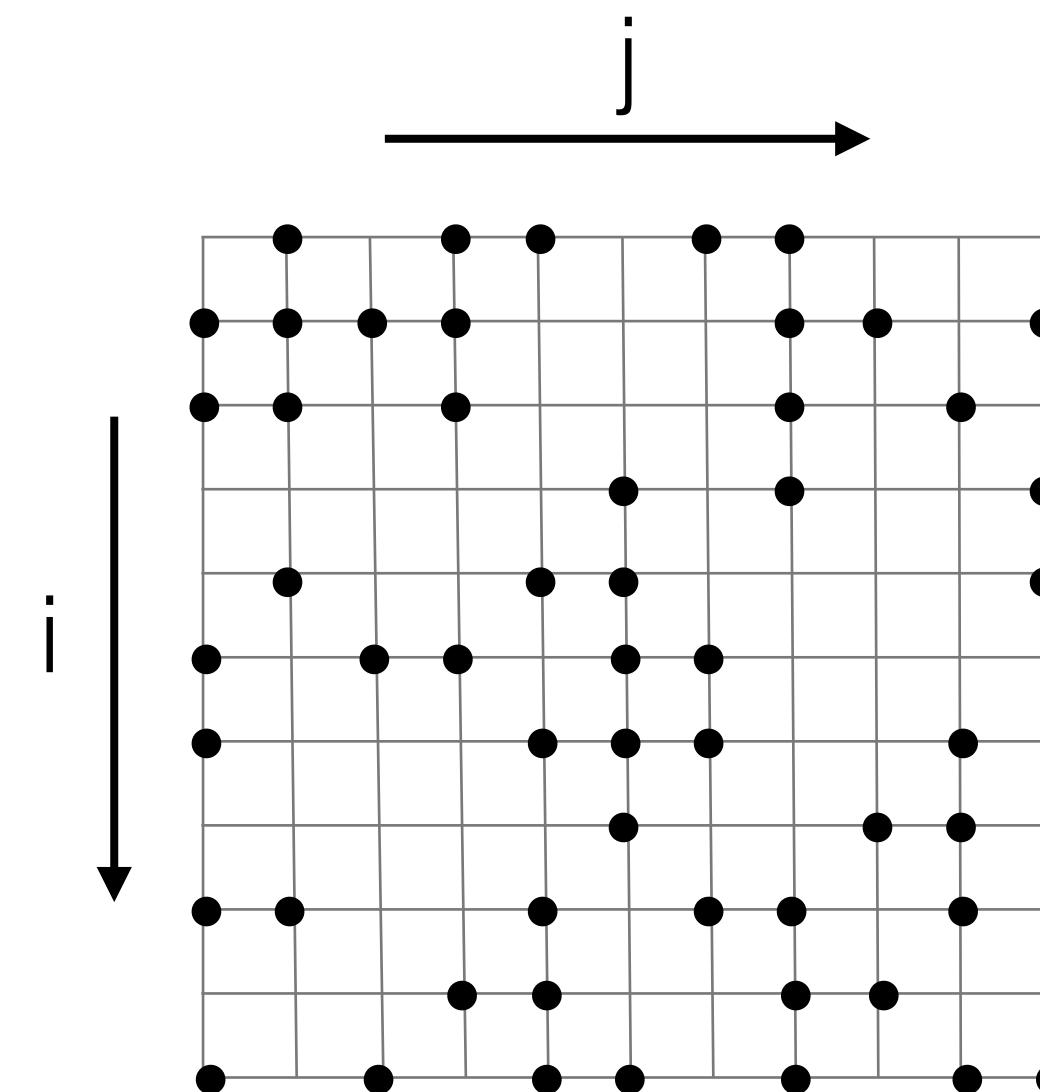
Tensor Index Notation:  $A_{ij} = B_{ij} + C_{ij}$



# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

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



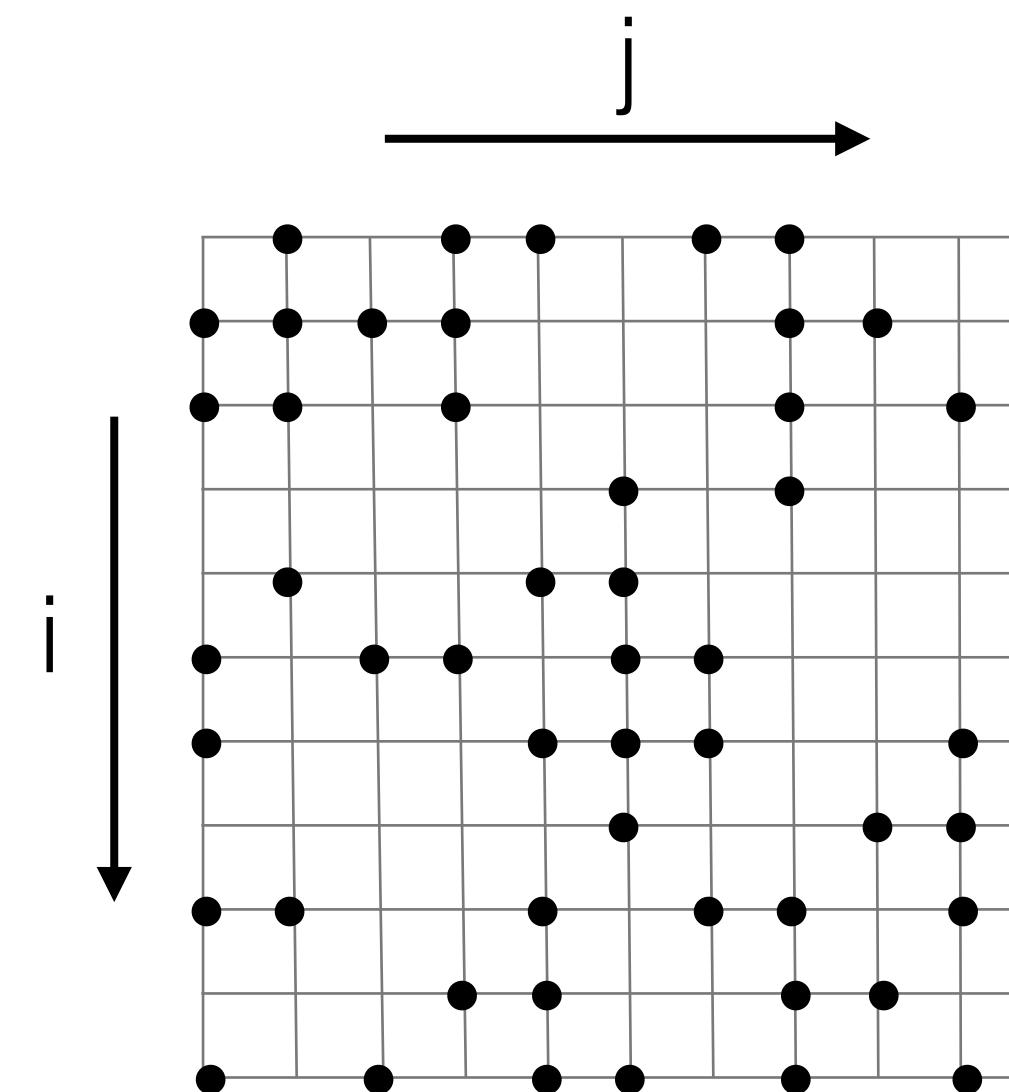
$B$

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

# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

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



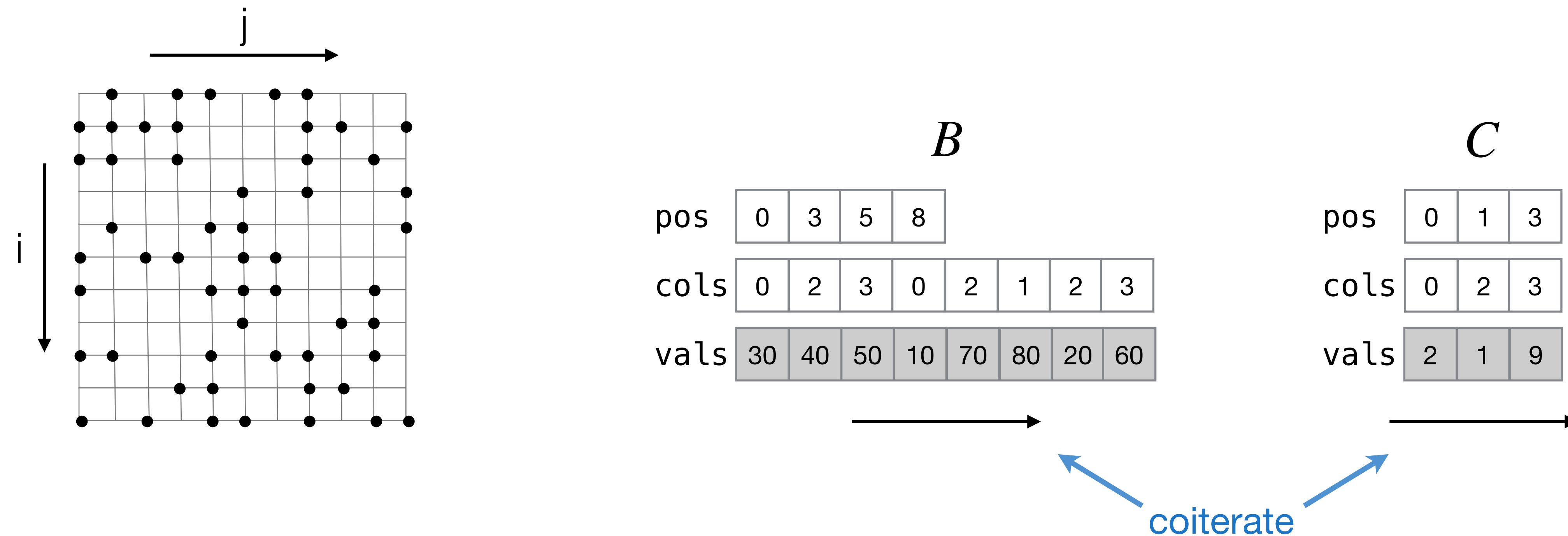
$B$								
pos	0	3	5	8				
cols	0	2	3	0	2	1	2	3
vals	30	40	50	10	70	80	20	60

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

# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

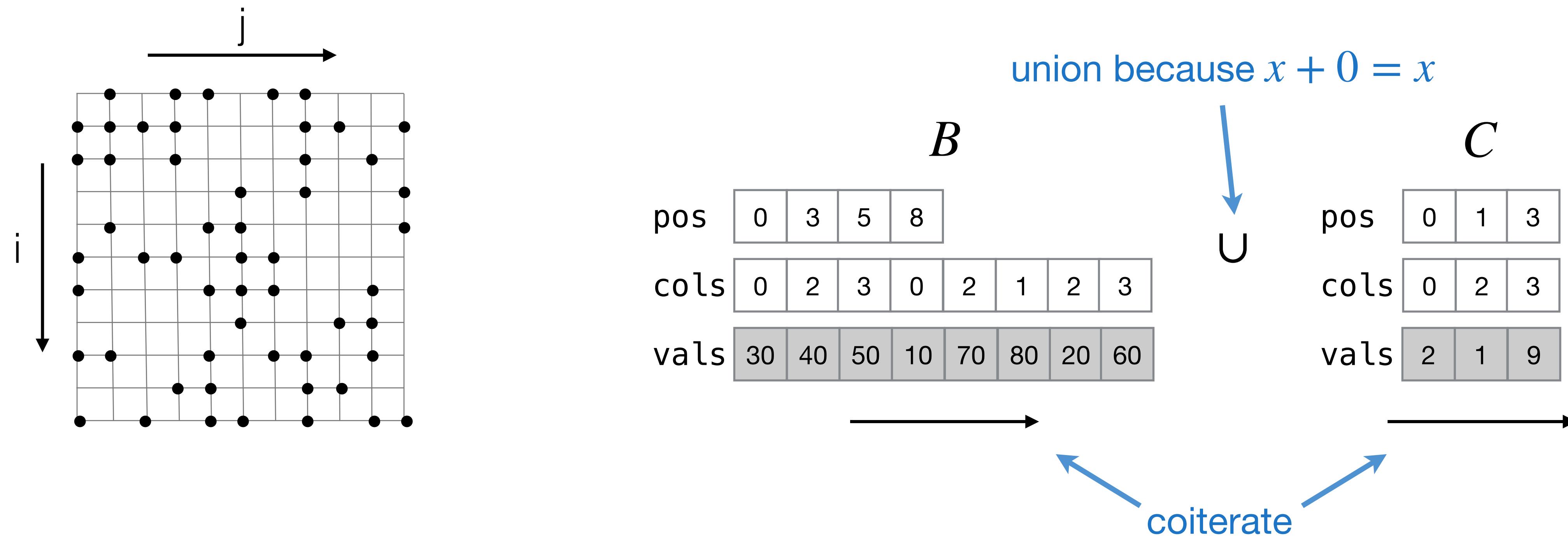
Tensor Index Notation:  $A_{ij} = B_{ij} + C_{ij}$



# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

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

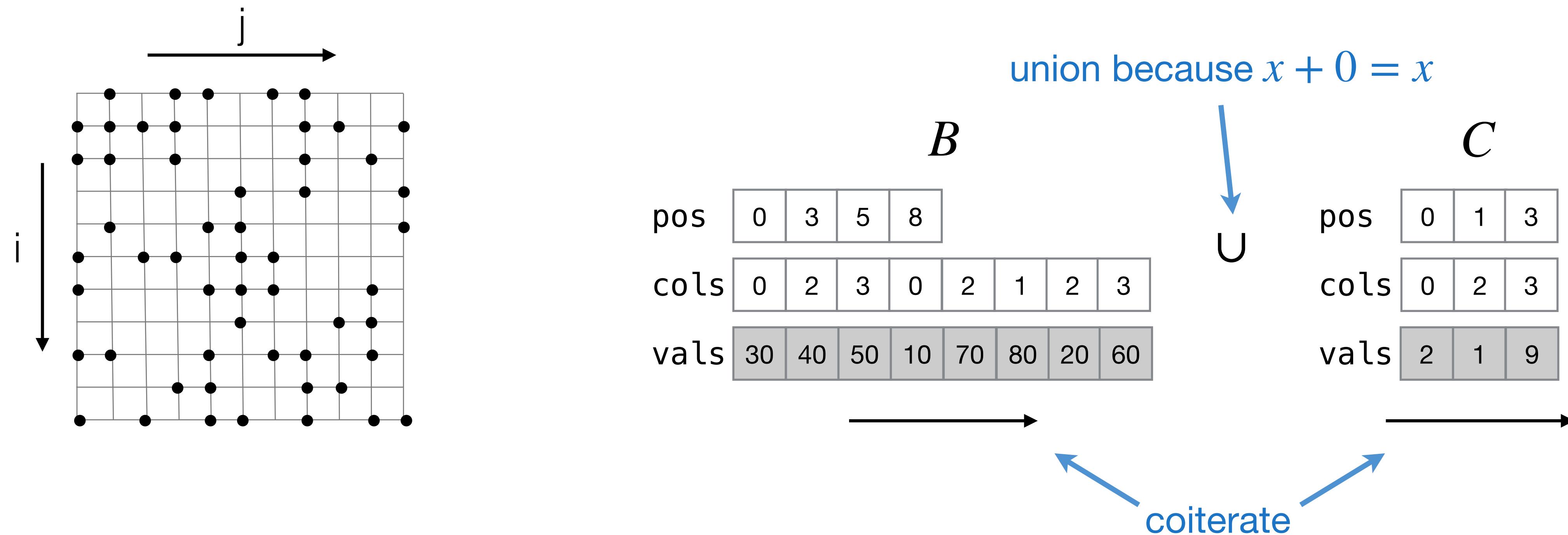


# Iteration over sparse iteration spaces imply coiteration over sparse data structures

Linear Algebra:  $A = B + C$

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

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

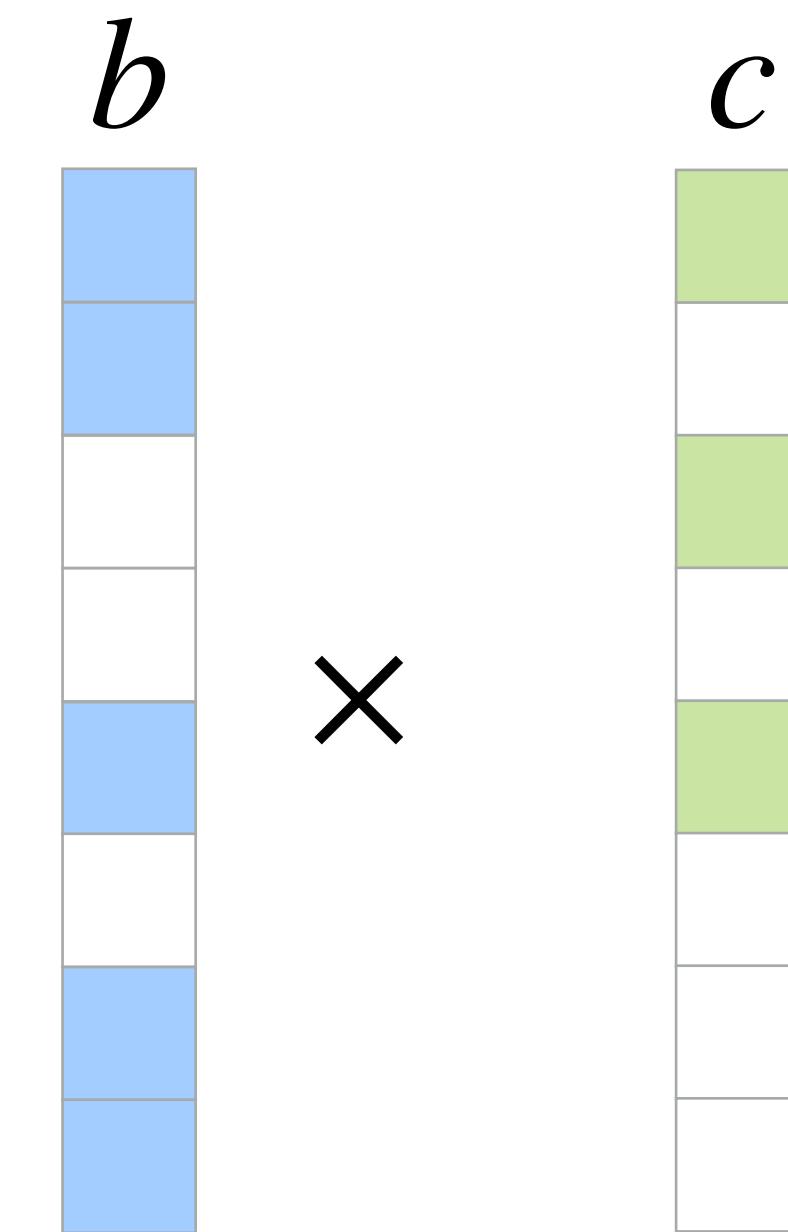


# Merged coiteration

Coordinate Space



$$a_i = b_i c_i$$

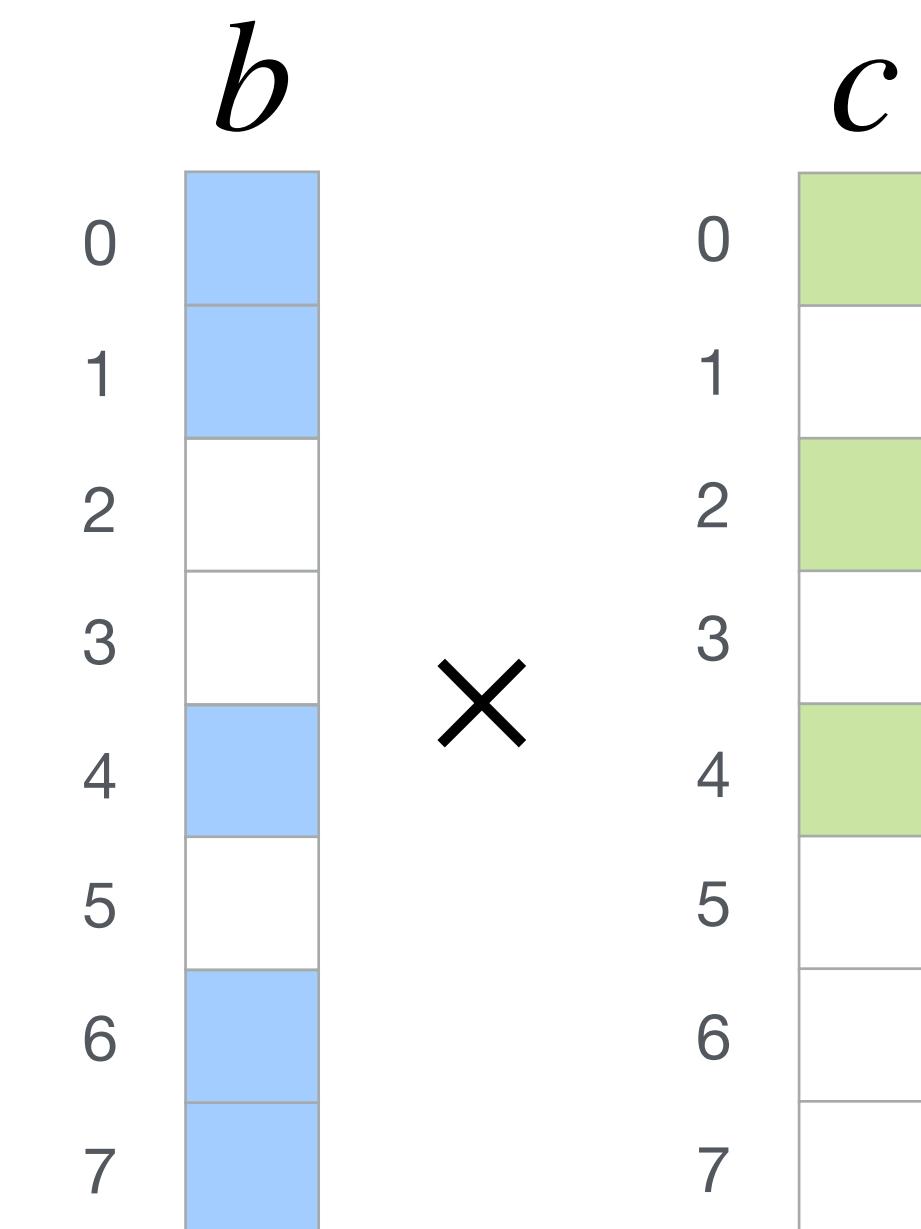


# Merged coiteration

Coordinate Space



$$a_i = b_i c_i$$

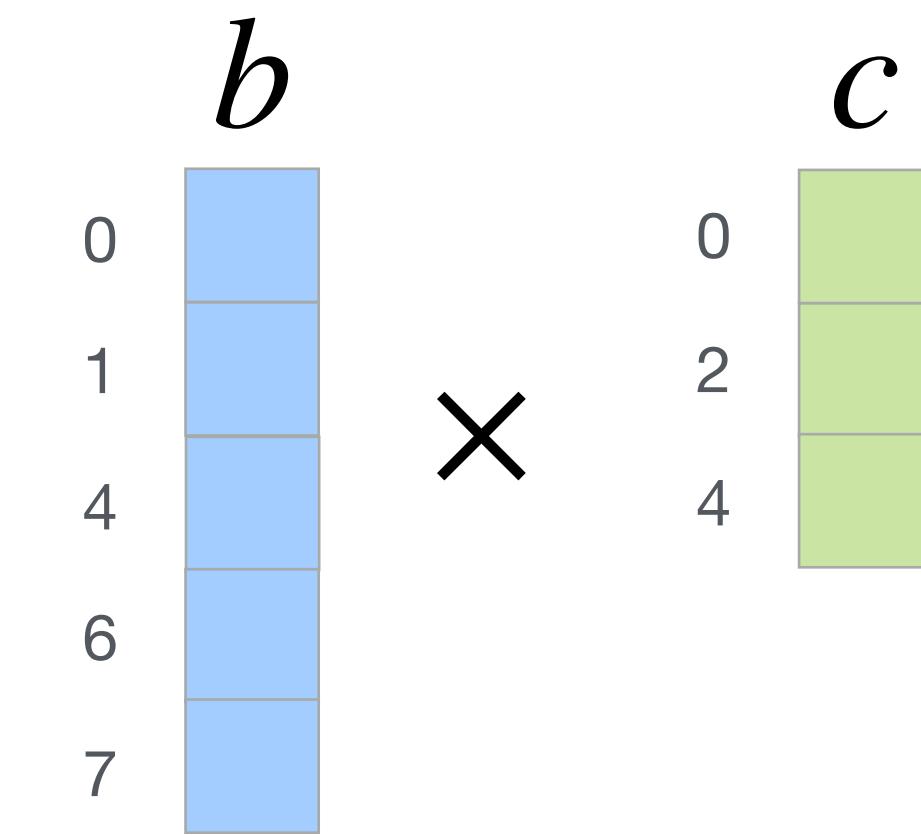


# Merged coiteration

Coordinate Space

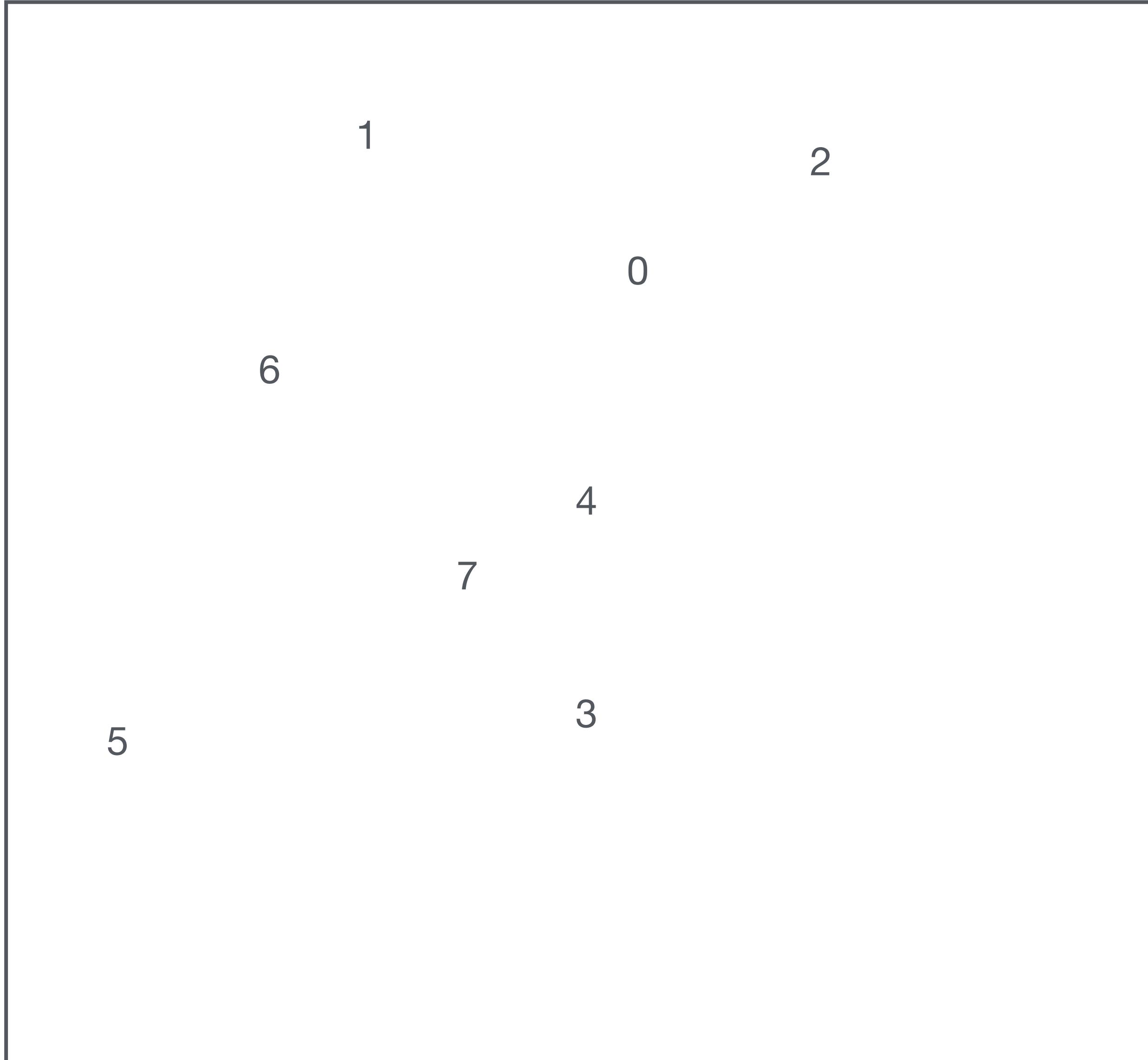


$$a_i = b_i c_i$$



# Merged coiteration

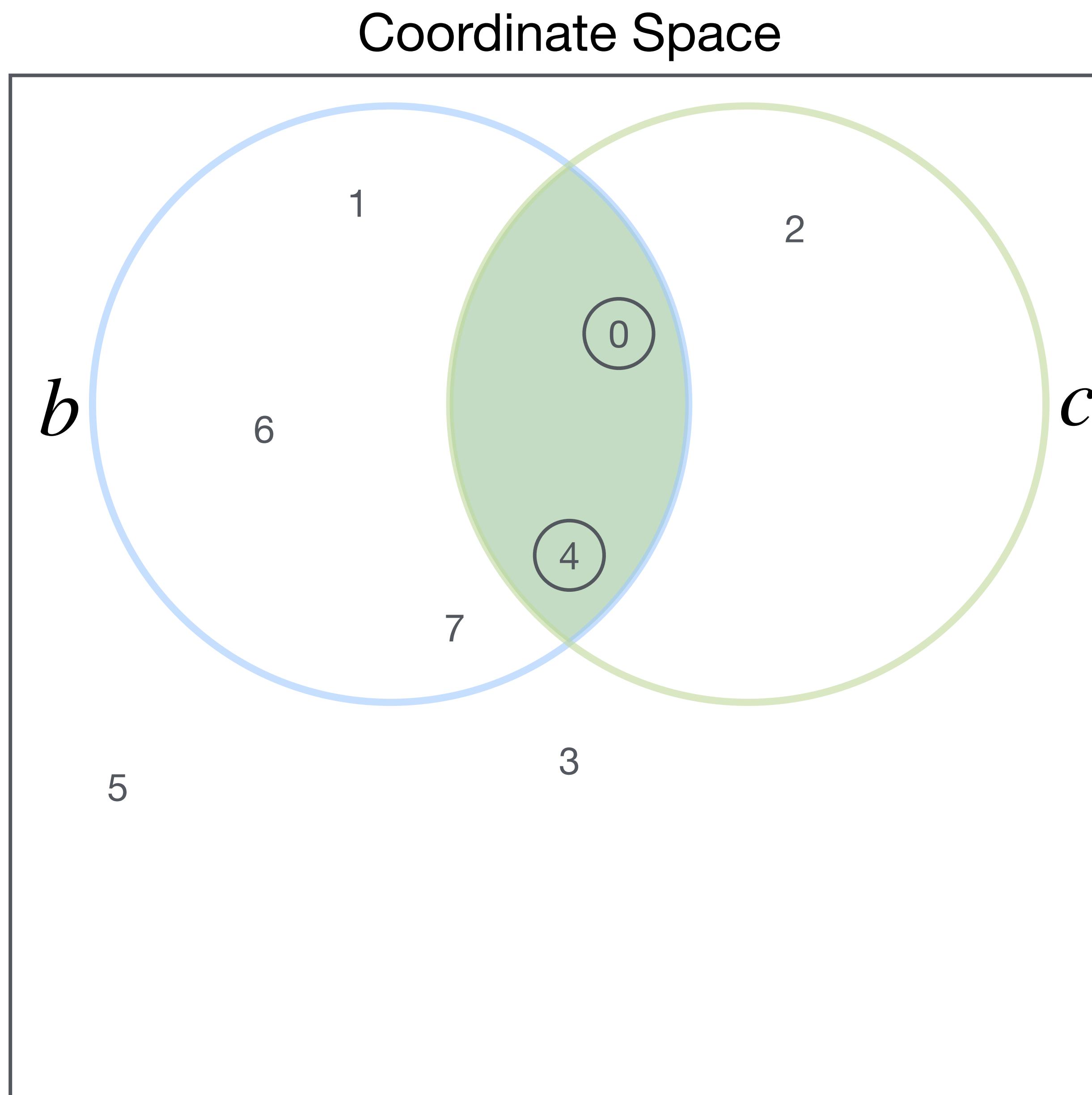
Coordinate Space



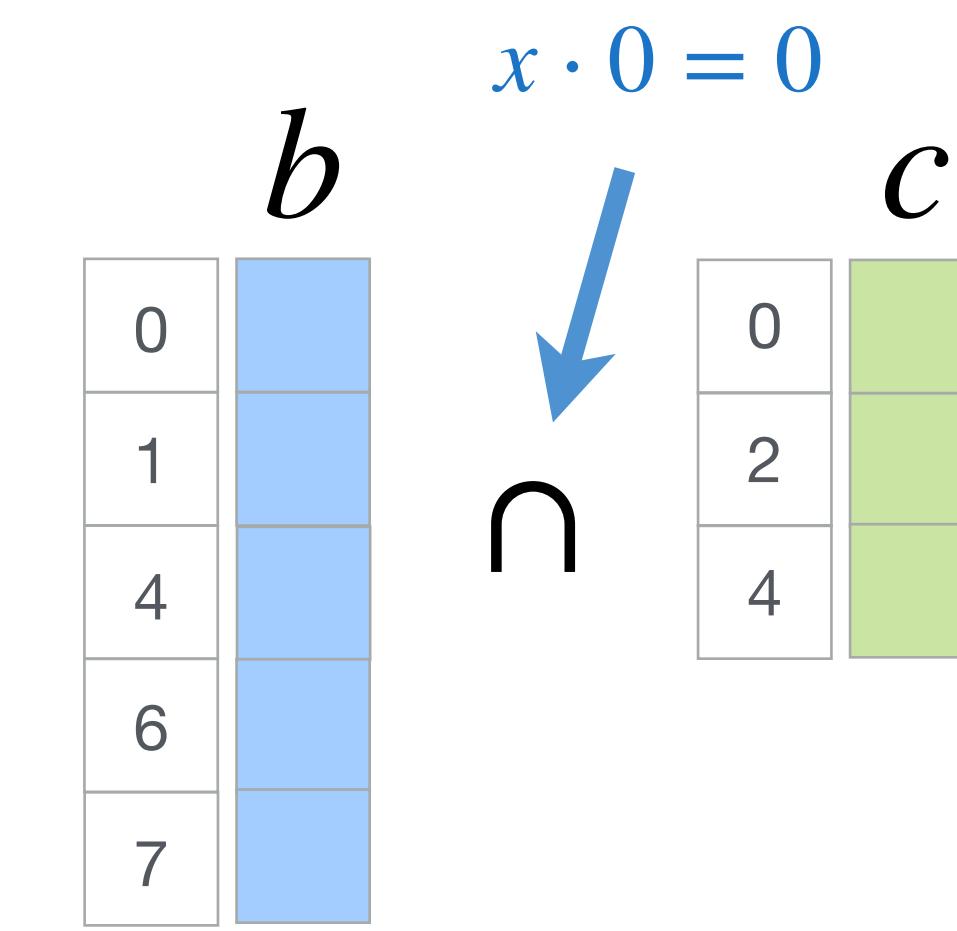
$$a_i = b_i c_i$$

$$\begin{matrix} b & \times & c \\ \begin{array}{|c|c|} \hline 0 & \text{blue} \\ \hline 1 & \text{blue} \\ \hline 4 & \text{blue} \\ \hline 6 & \text{blue} \\ \hline 7 & \text{blue} \\ \hline \end{array} & \times & \begin{array}{|c|c|} \hline 0 & \text{green} \\ \hline 2 & \text{green} \\ \hline 4 & \text{green} \\ \hline \end{array} \end{matrix}$$

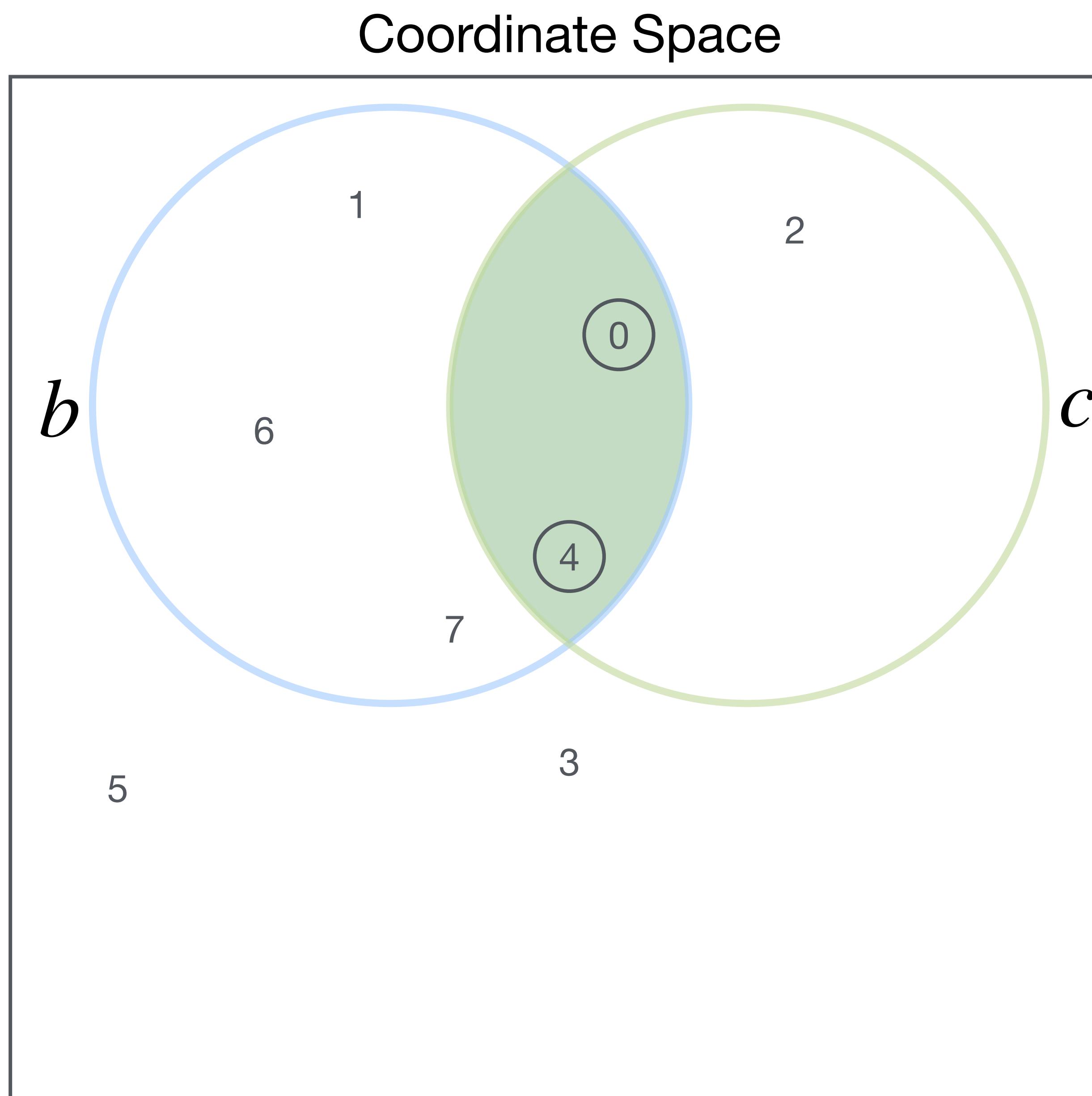
# Merged coiteration



$$a_i = b_i c_i$$



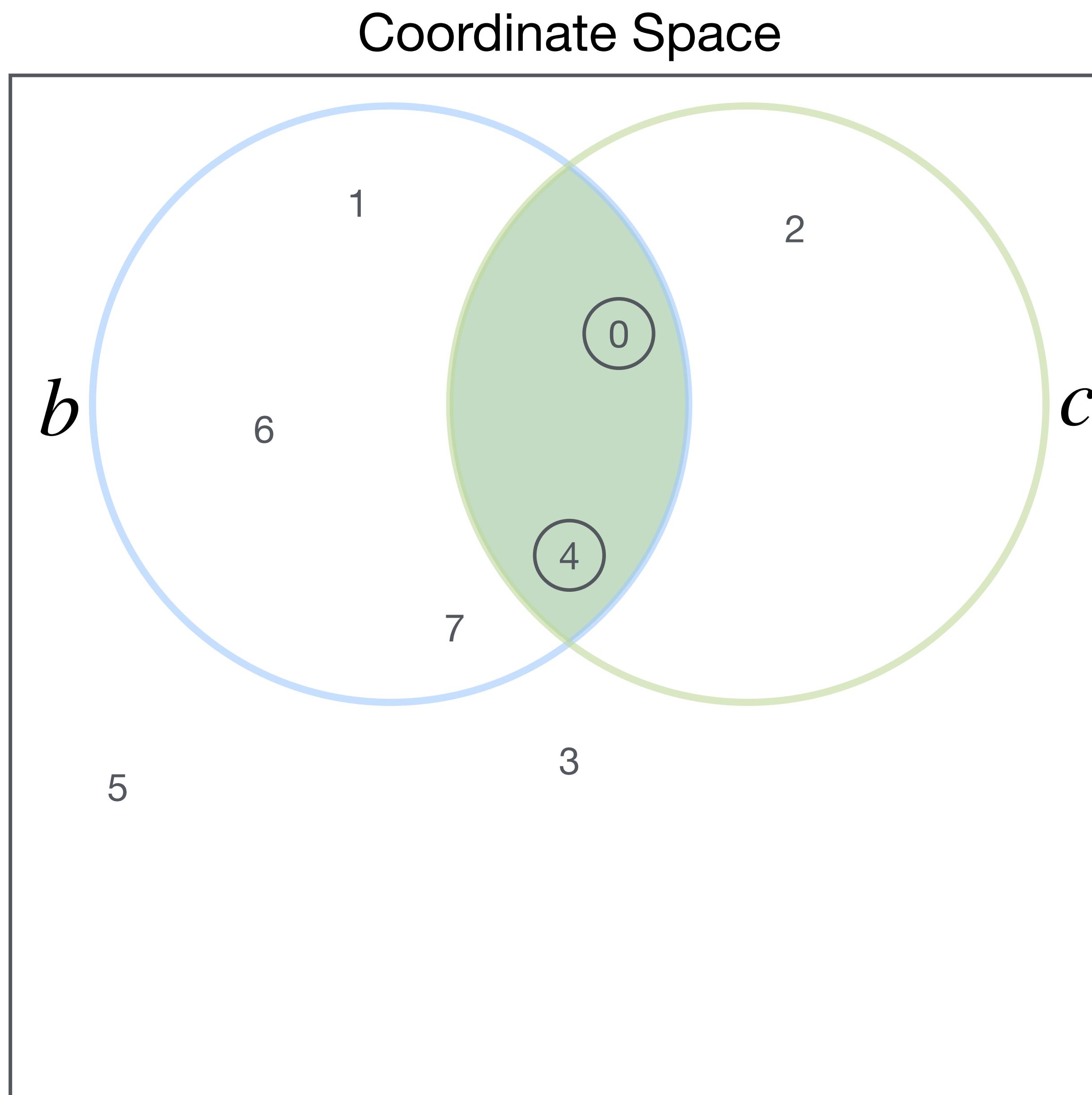
# Merged coiteration



$$a_i = b_i c_i$$

$$a = \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} = \begin{matrix} 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} 0 \\ 2 \\ 4 \end{matrix}$$

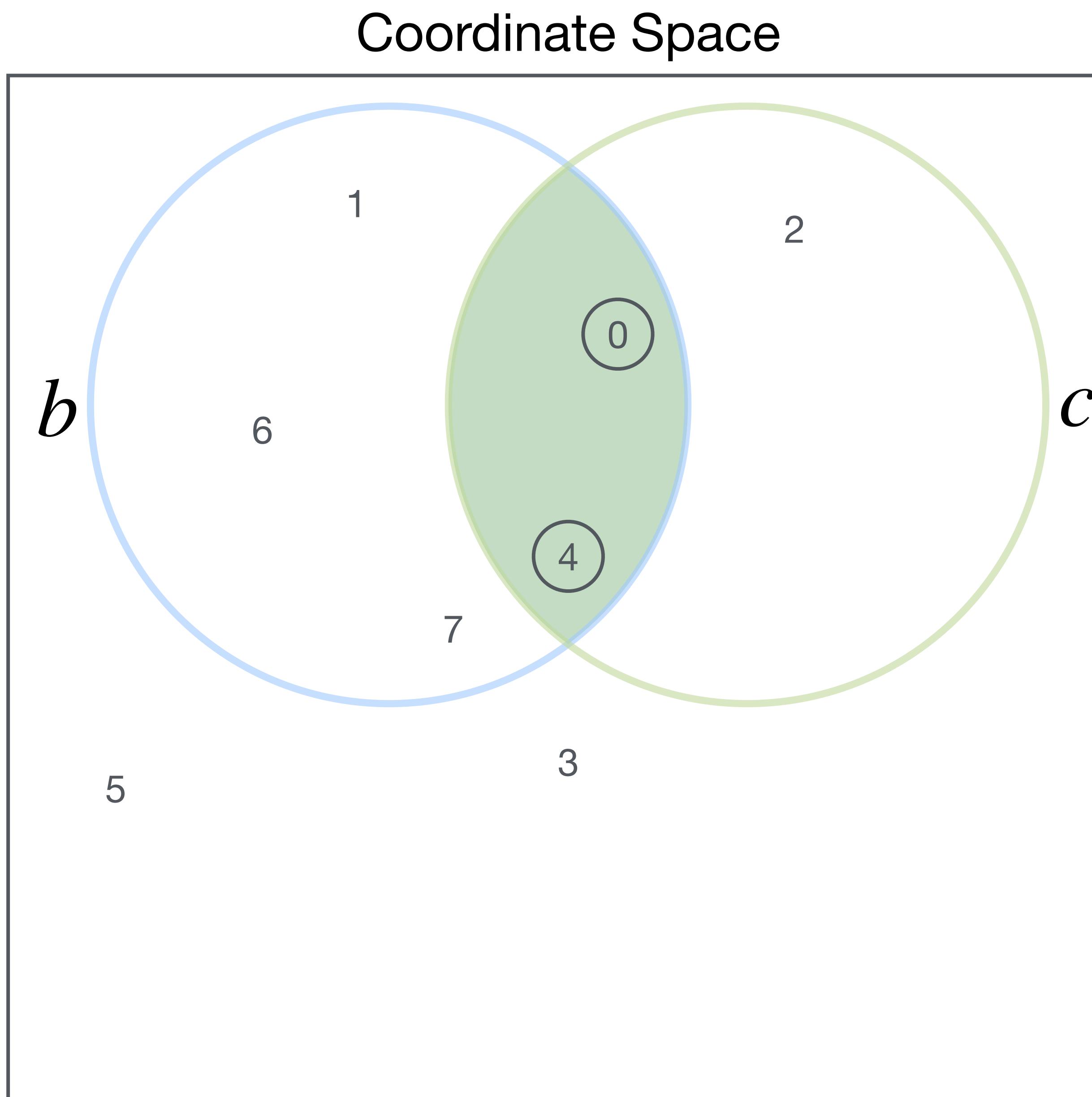
# Merged coiteration



$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} = \begin{matrix} b \\ \hline 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} c \\ \hline 0 \\ 2 \\ 4 \end{matrix}$$

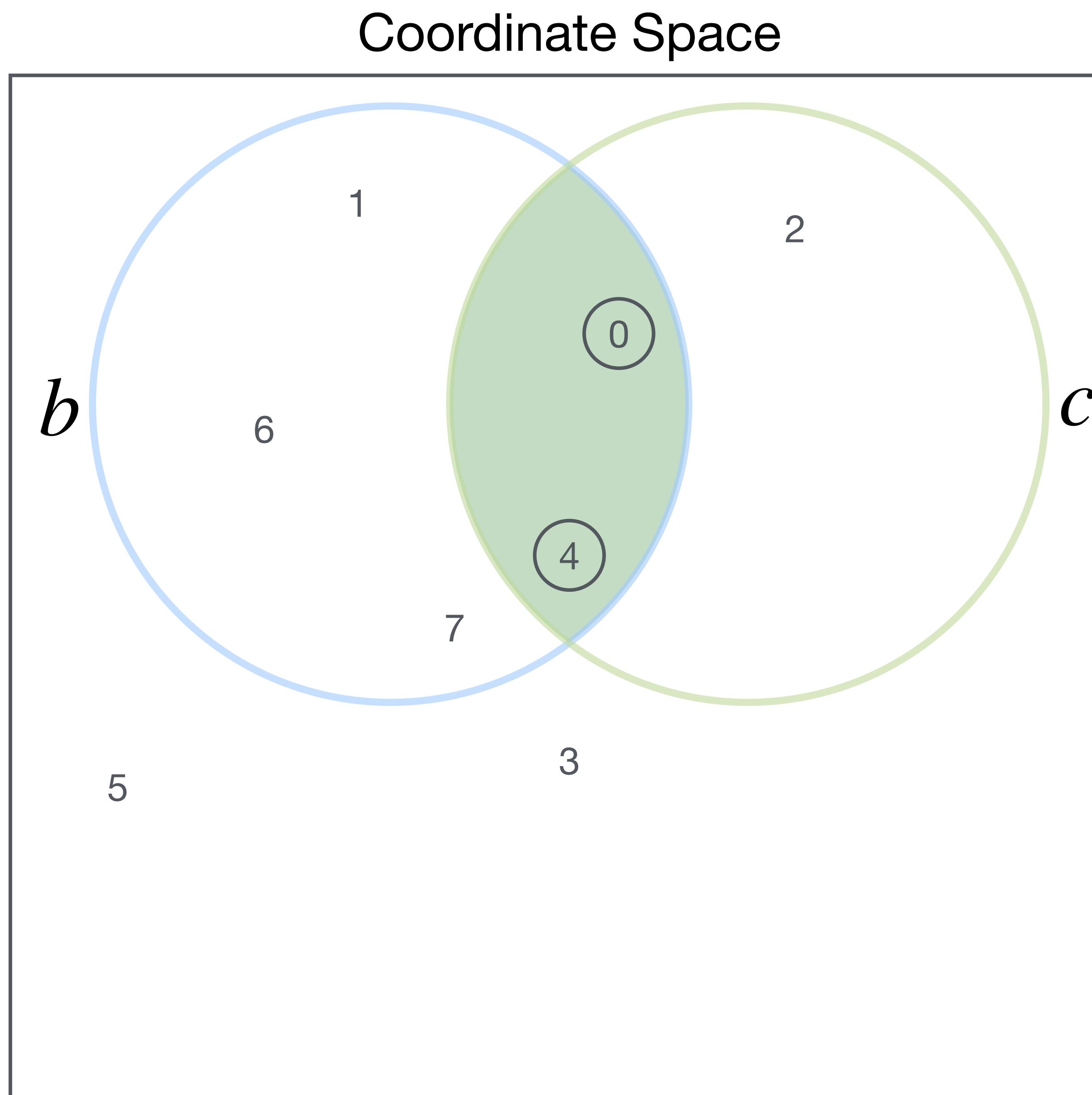
# Merged coiteration



$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} = \begin{matrix} b \\ \hline 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} c \\ \hline 0 \\ 2 \\ 4 \end{matrix}$$

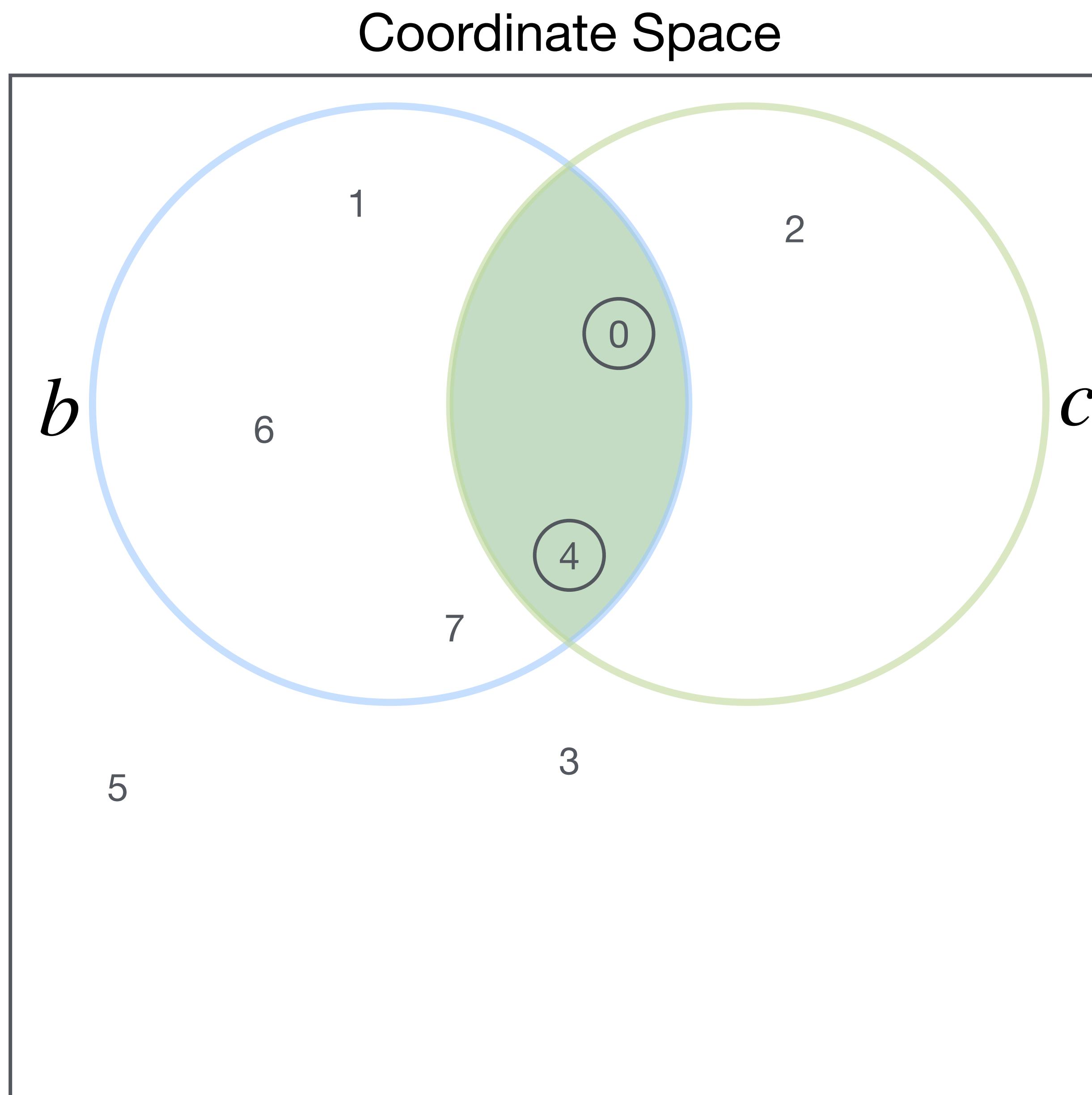
# Merged coiteration



$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} = \begin{matrix} b \\ \hline 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} c \\ \hline 0 \\ 2 \\ 4 \end{matrix}$$

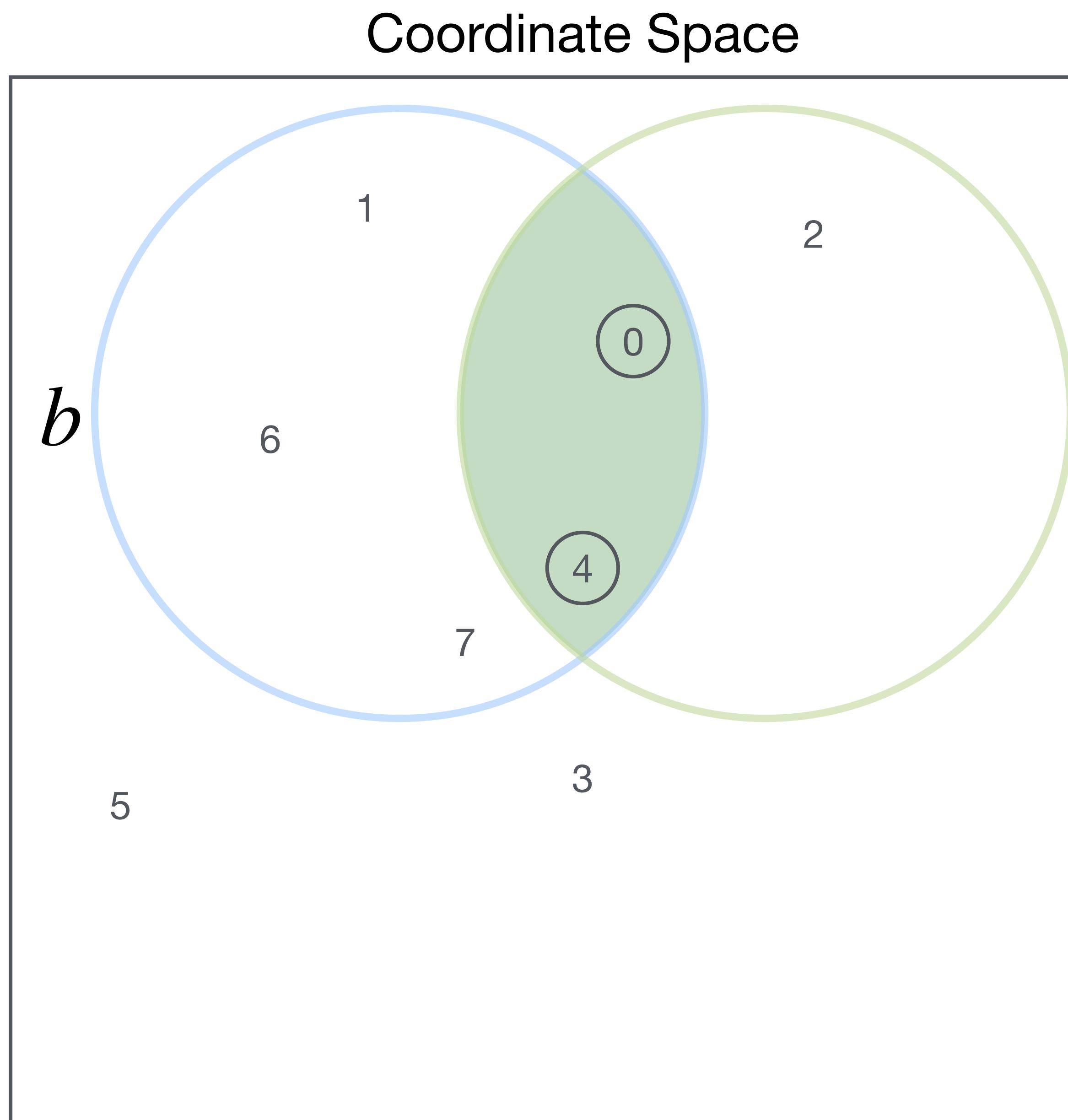
# Merged coiteration



$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} = \begin{matrix} b \\ \hline 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} c \\ \hline 0 \\ 2 \\ 4 \end{matrix}$$

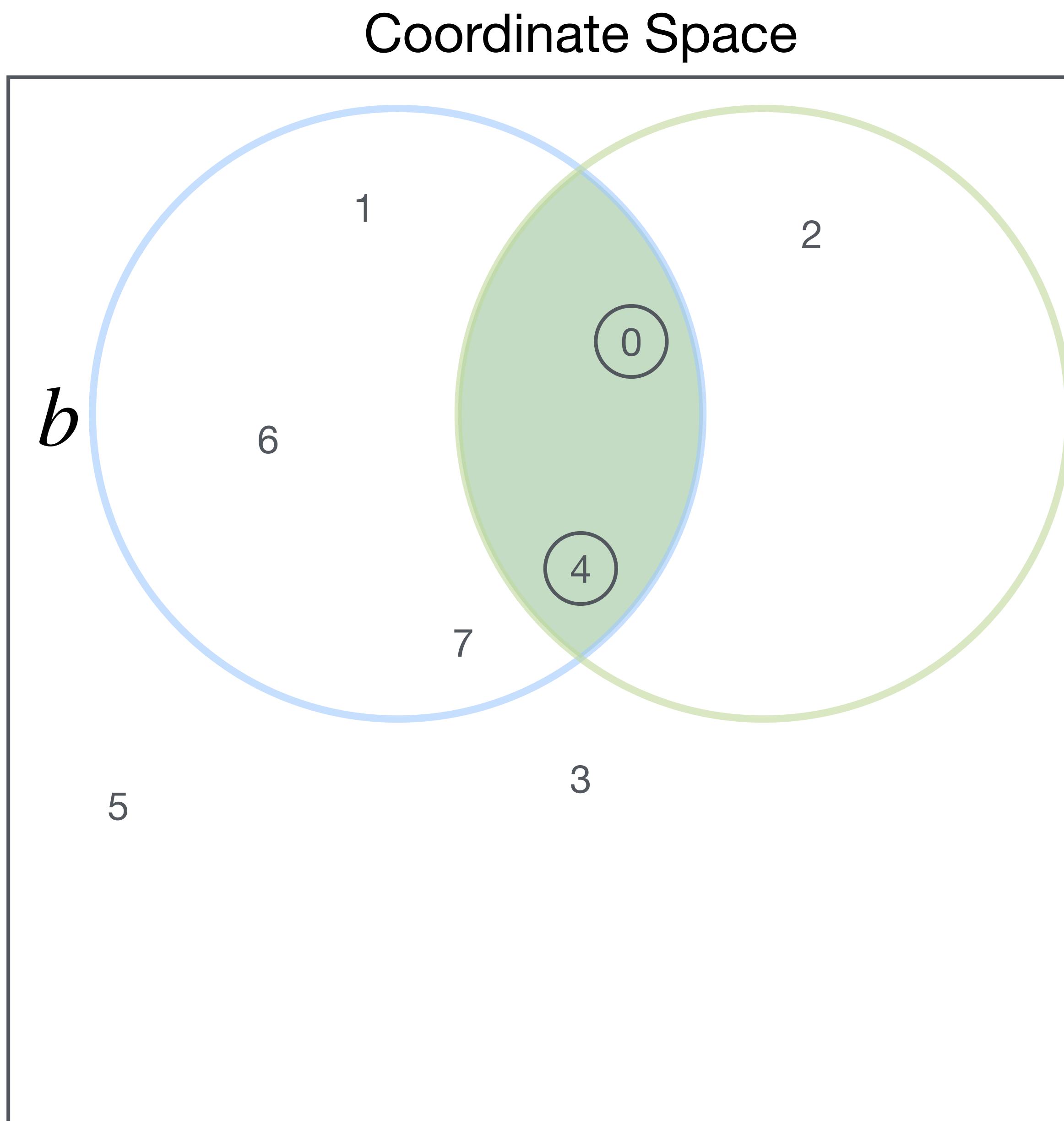
# Merged coiteration



$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} = \begin{matrix} b \\ \hline 0 \\ 1 \\ 4 \\ 6 \\ 7 \end{matrix} \cap \begin{matrix} c \\ \hline 0 \\ 2 \\ 4 \end{matrix}$$

# Merged coiteration

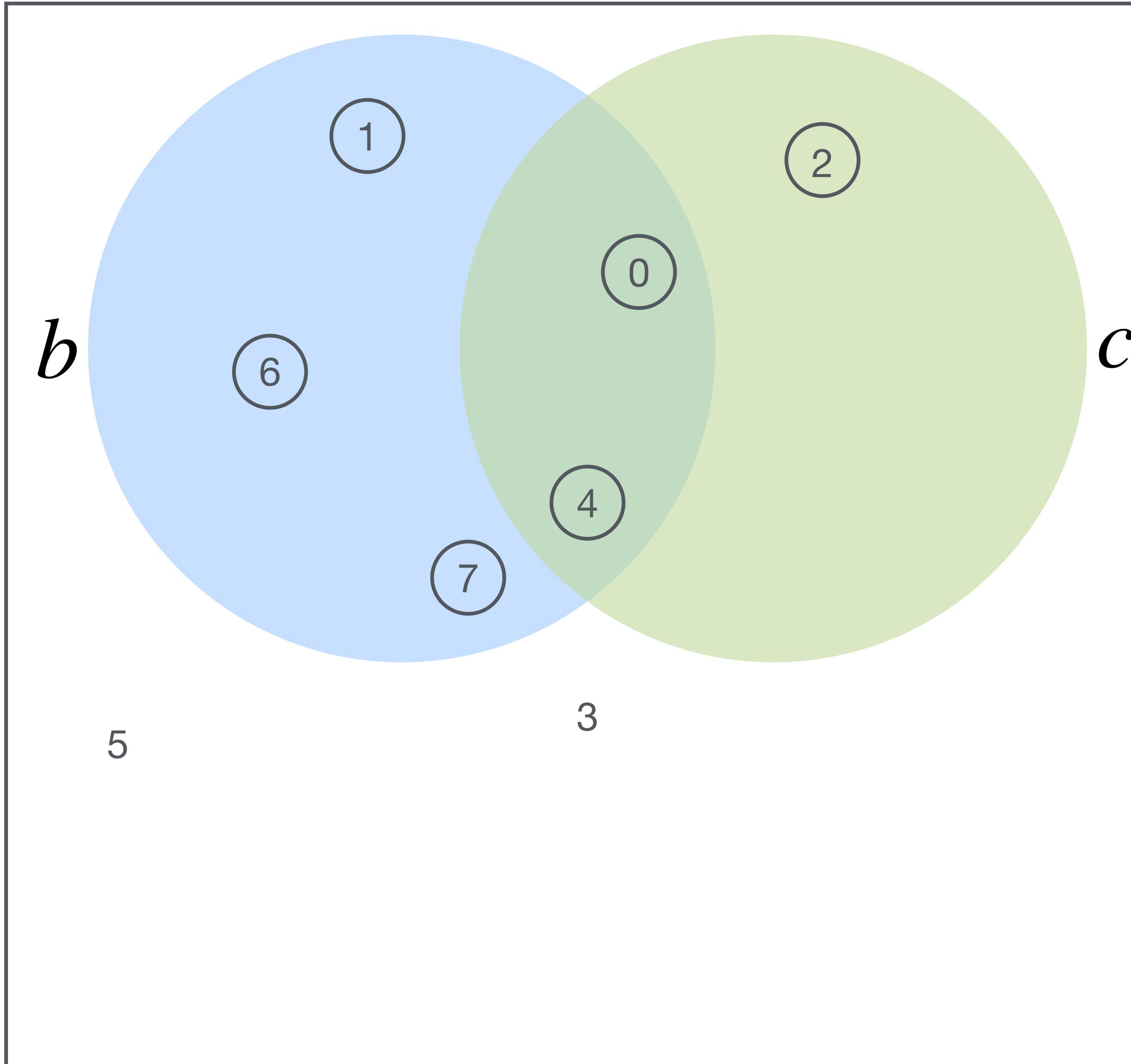


$$a_i = b_i c_i$$

$$\begin{matrix} a \\ \hline 0 & \text{blue} \\ 1 & \text{white} \\ 2 & \text{white} \\ 3 & \text{white} \\ 4 & \text{blue} \\ 5 & \text{white} \\ 6 & \text{white} \\ 7 & \text{white} \end{matrix} = \begin{matrix} b \\ \hline 0 & \text{white} \\ 1 & \text{white} \\ 4 & \text{white} \\ 6 & \text{gray} \\ 7 & \text{blue} \end{matrix} \cap \begin{matrix} c \\ \hline 0 & \text{white} \\ 2 & \text{white} \\ 4 & \text{white} \\ 6 & \text{white} \\ 7 & \text{green} \end{matrix}$$

# Merged coiteration

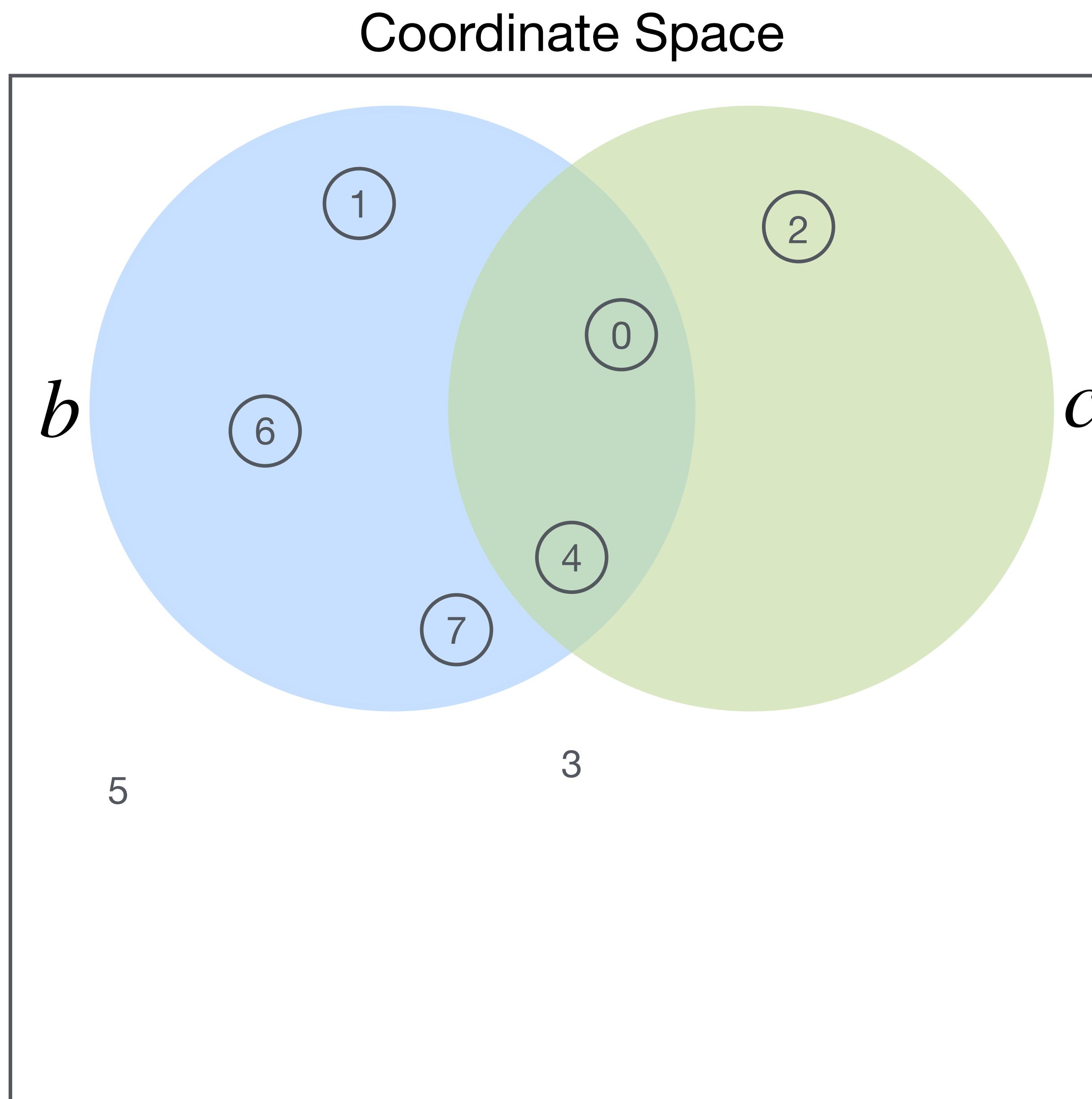
Coordinate Space



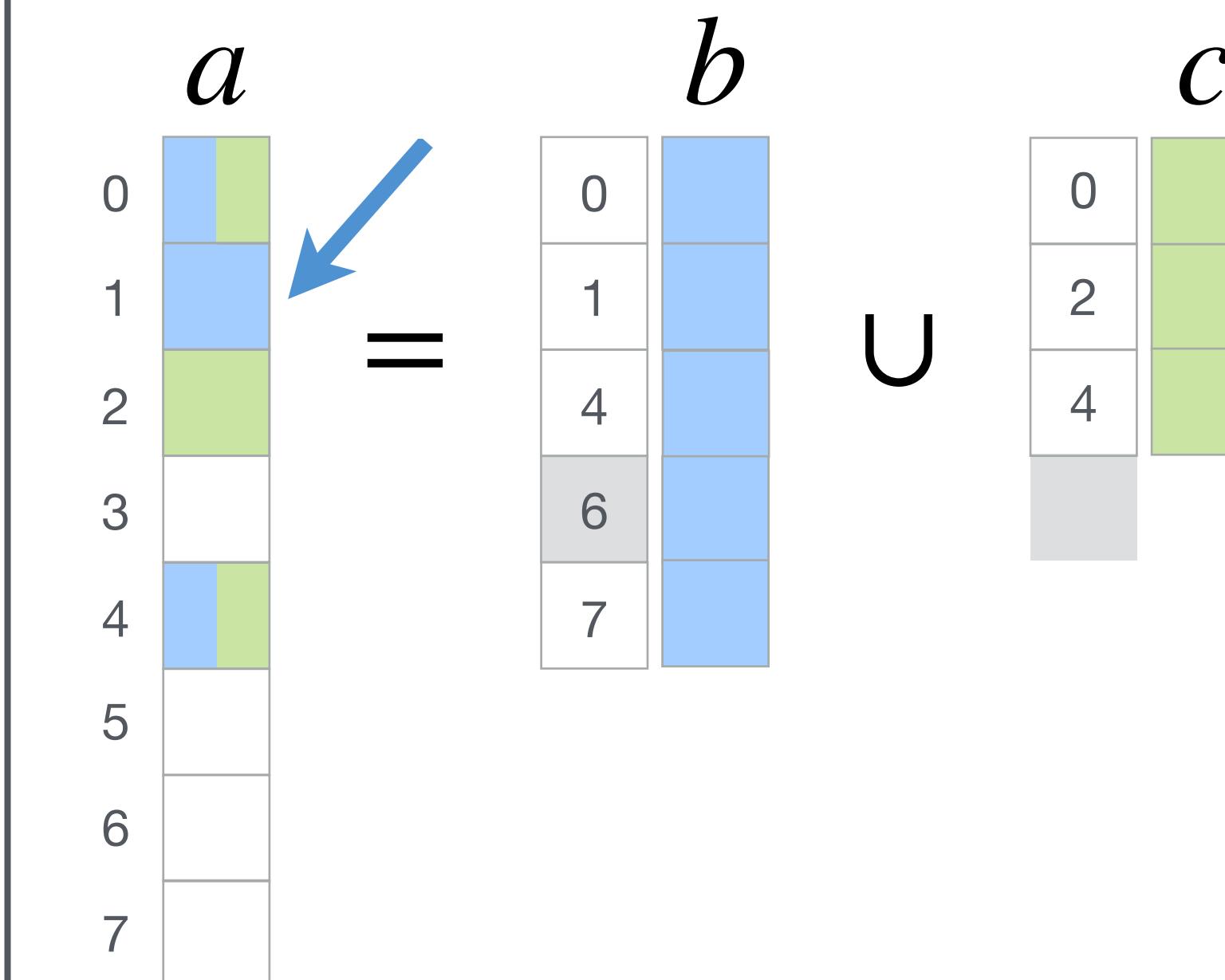
$$a_i = b_i + c_i$$

$$\begin{matrix} a \\ \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} = \begin{matrix} b \\ \hline 0 & 1 & 4 & 6 & 7 \end{matrix} \cup \begin{matrix} c \\ \hline 0 & 2 & 4 \end{matrix}$$

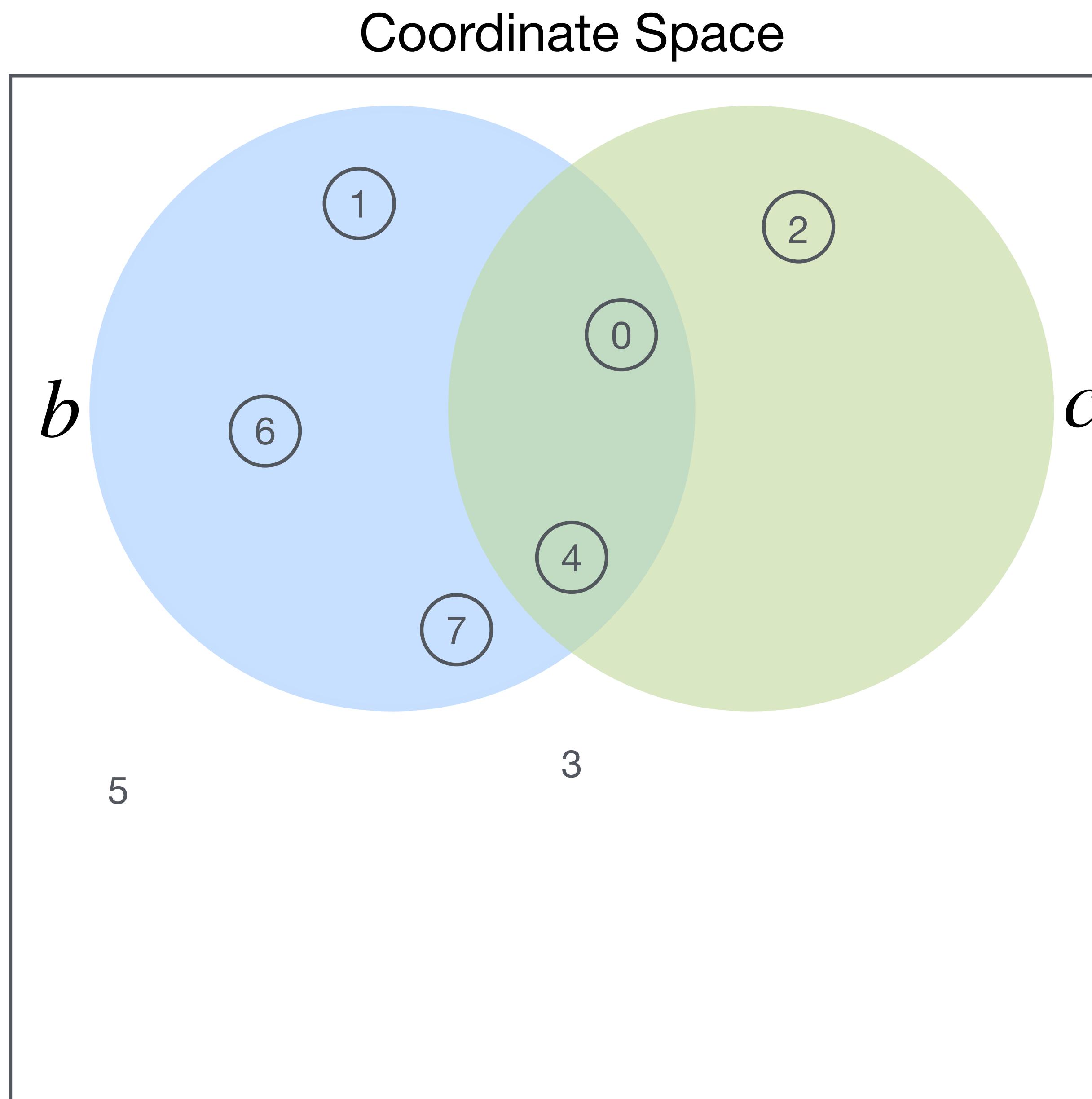
# Merged coiteration



$$a_i = b_i + c_i$$



# Merged coiteration



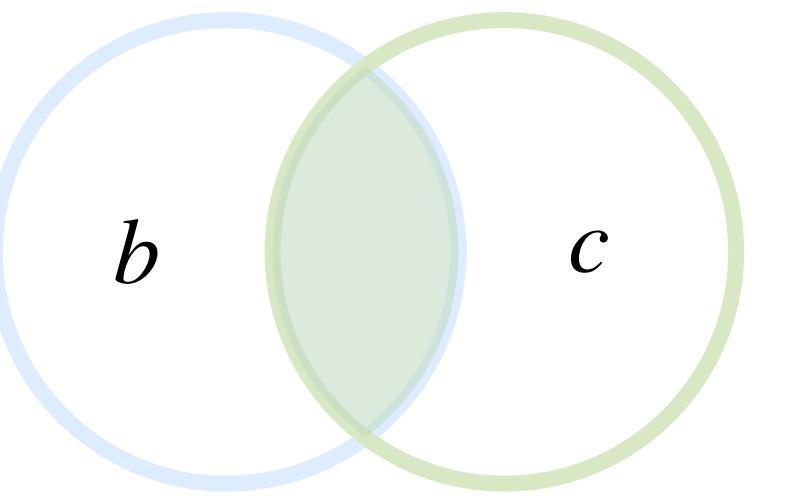
$$a_i = b_i + c_i$$

$$a = \begin{matrix} & b \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{matrix} 0 \\ 1 \\ 2 \\ 4 \\ 6 \\ 7 \end{matrix} \end{matrix} \cup \begin{matrix} & c \\ \begin{matrix} 0 \\ 2 \\ 4 \end{matrix} & \begin{matrix} 0 \\ 2 \\ 4 \end{matrix} \end{matrix}$$

# Merged coiteration code

Intersection  $b \cap c$

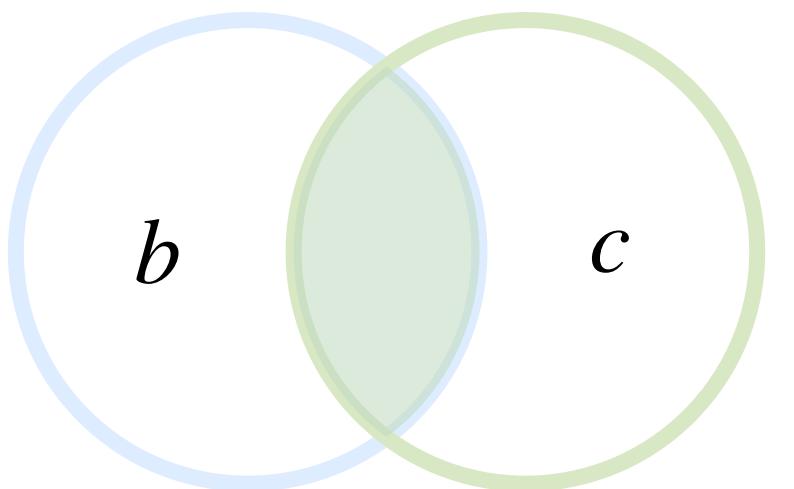
```
int pb = b_pos[0];
int pc = c_pos[0];
while (pb < b_pos[1] && pc < c_pos[1]) {
    int ib = b_crd[pb];
    int ic = c_crd[pc];
    int i = min(ib, ic);
    if (ib == i && ic == i) {
        a[i] = b[pb] * c[pc];
    }
    if (ib == i) pb++;
    if (ic == i) pc++;
}
```



# 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++;
}
```

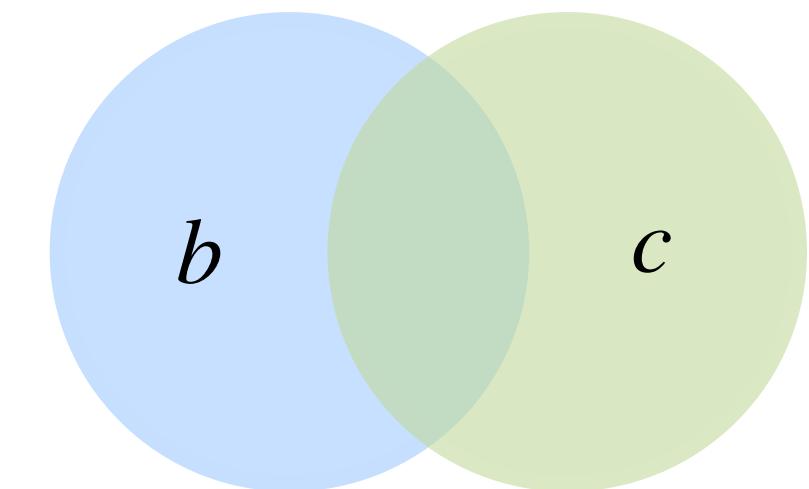


Union  $b \cup c$

```
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)

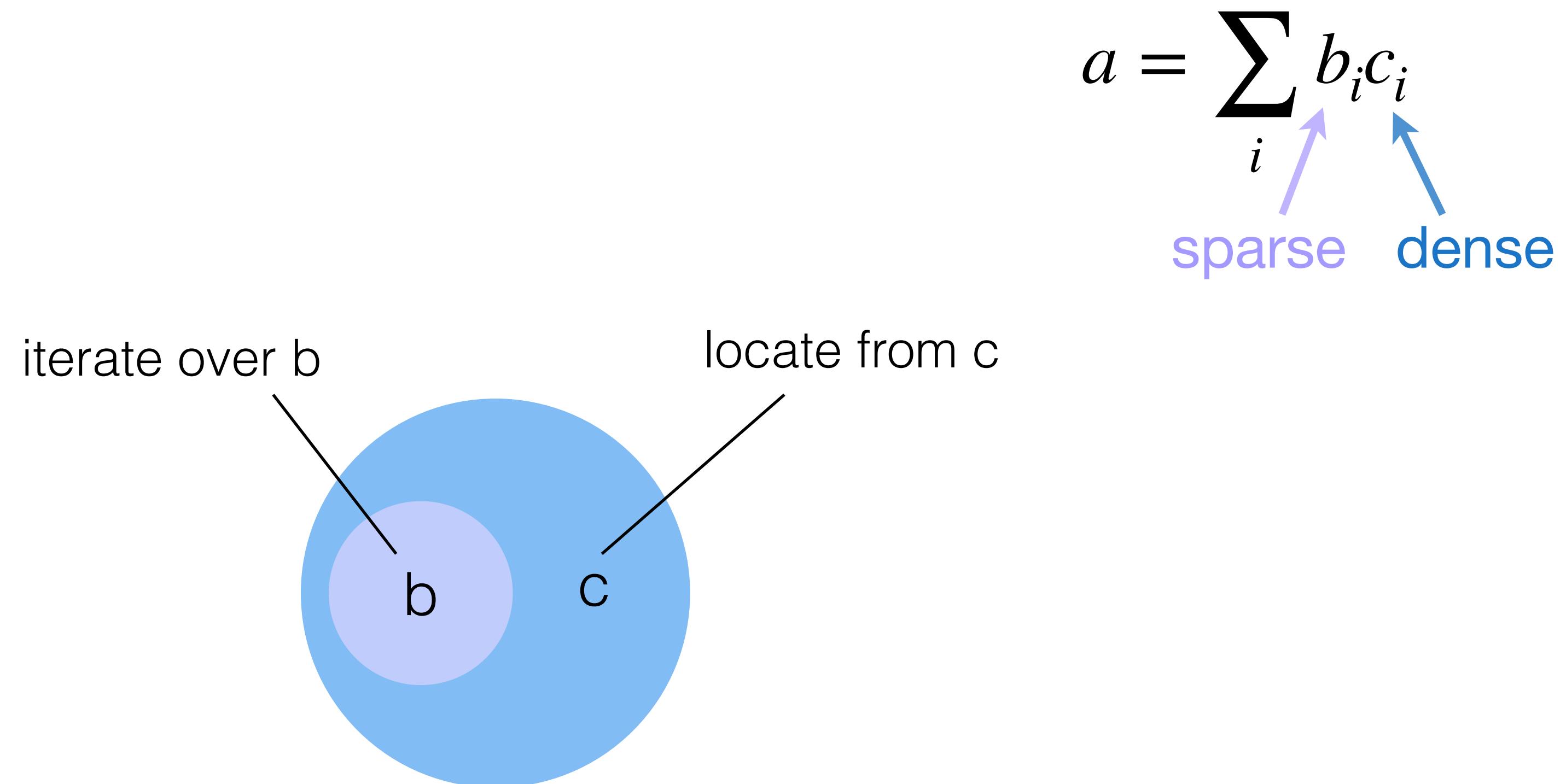
$$a = \sum_i b_i c_i$$

# Iterate-and-locate examples (intersection)

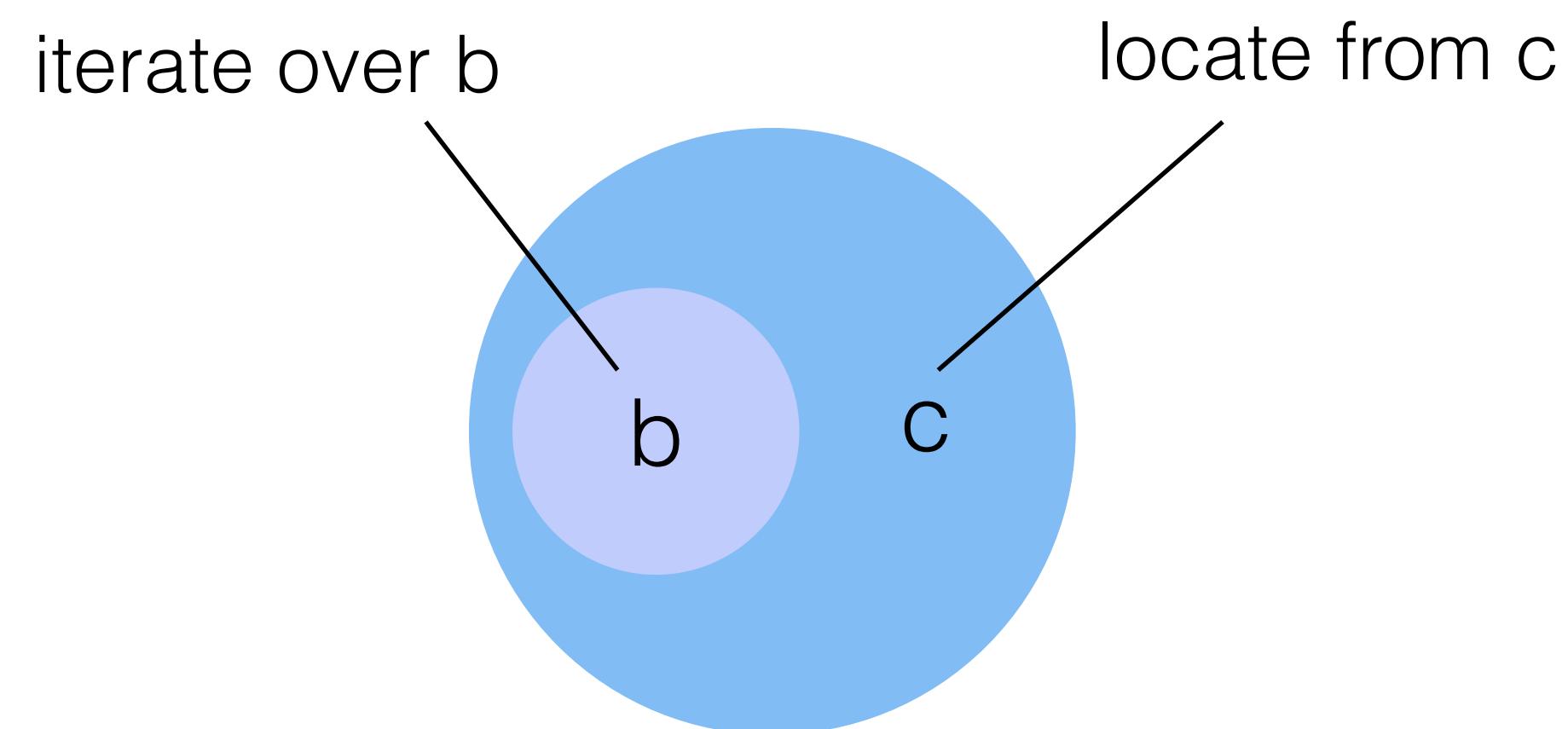
$$a = \sum_i b_i c_i$$

The diagram shows the expression  $a = \sum_i b_i c_i$ . A purple arrow points from the label "sparse" to the scalar  $b_i$ , indicating that  $b_i$  is a sparse vector. A blue arrow points from the label "dense" to the vector  $c_i$ , indicating that  $c_i$  is a dense vector.

# Iterate-and-locate examples (intersection)



# Iterate-and-locate examples (intersection)

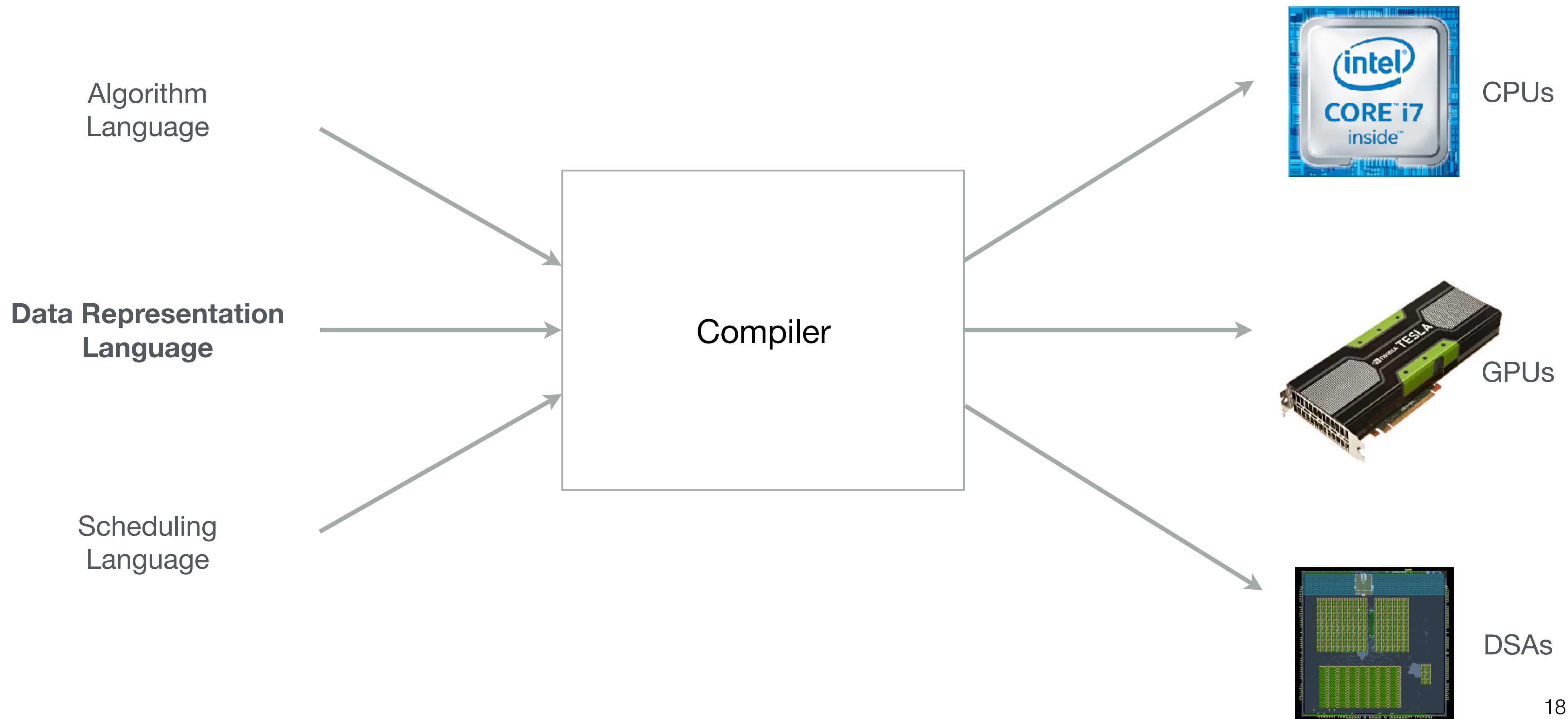


$$a = \sum_i b_i c_i$$

sparse      dense

```
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

