ASPLOS '23

WACO : Learning workload-aware co-optimization of the format and schedule for a sparse tensor program

Jaeyeon Won, Charith Mendis, Joel Emer, Saman Amarasinghe

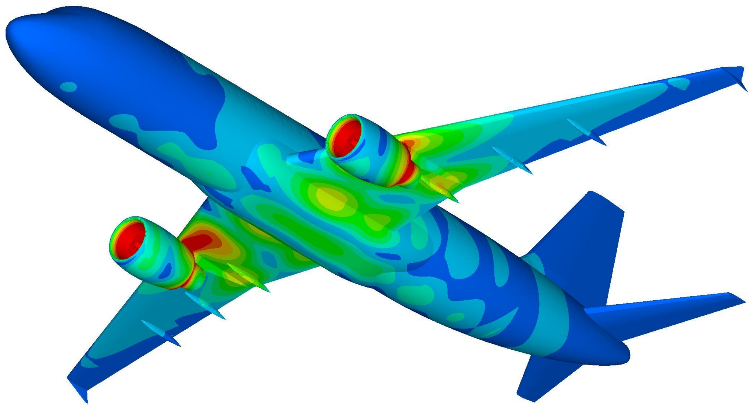
MIT CSAIL

University of Illinois Urbana-Champaign

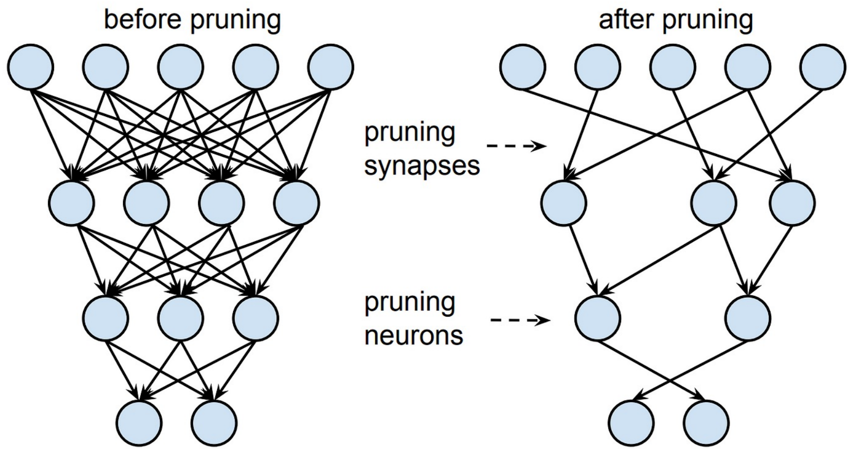
The COMMIT Compiler Group

COMPILERS @ MIT

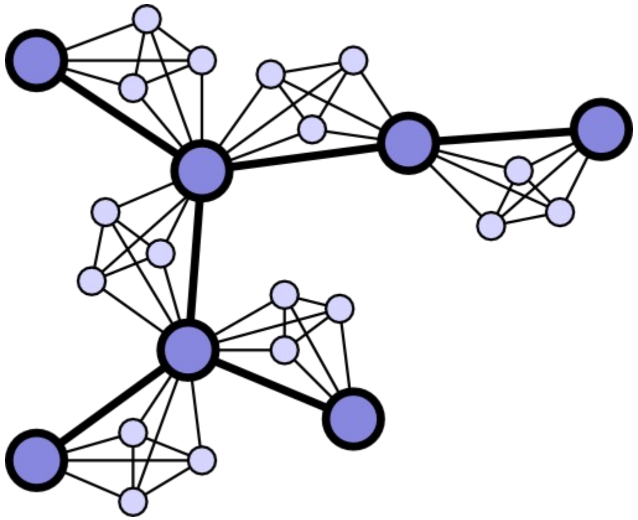
Sparse Tensors are Everywhere



Scientific Computing

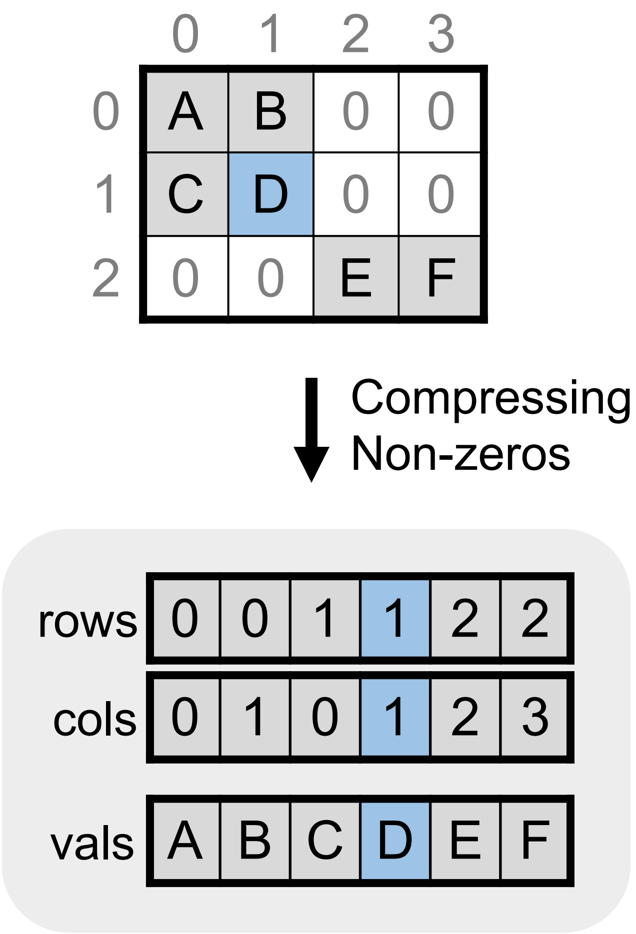


Deep Learning



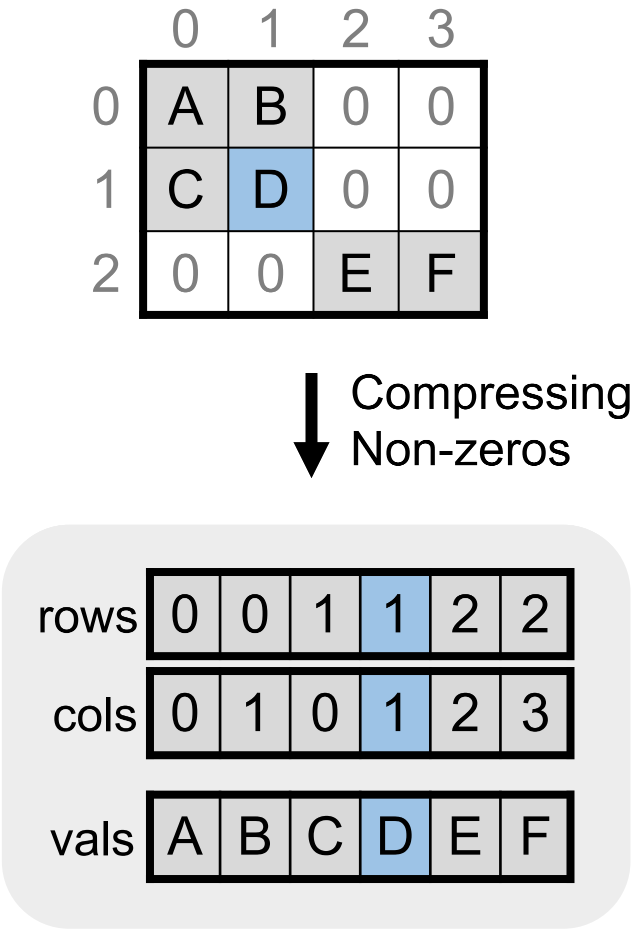
Graph Analytics

Writing Sparse Code is Hard

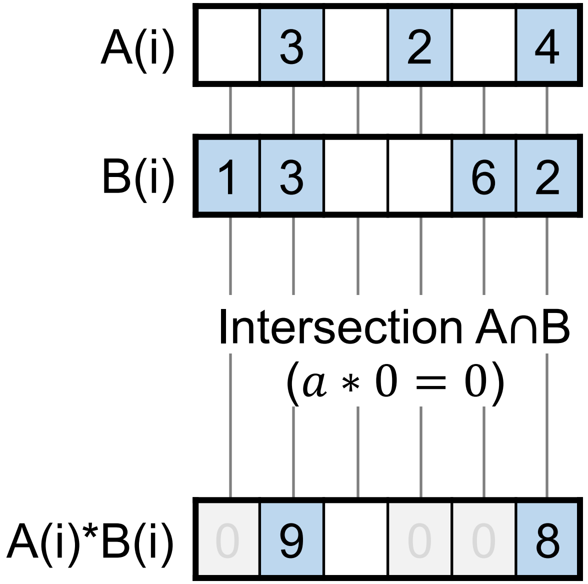


Sparse
Data Representation

Writing Sparse Code is Hard

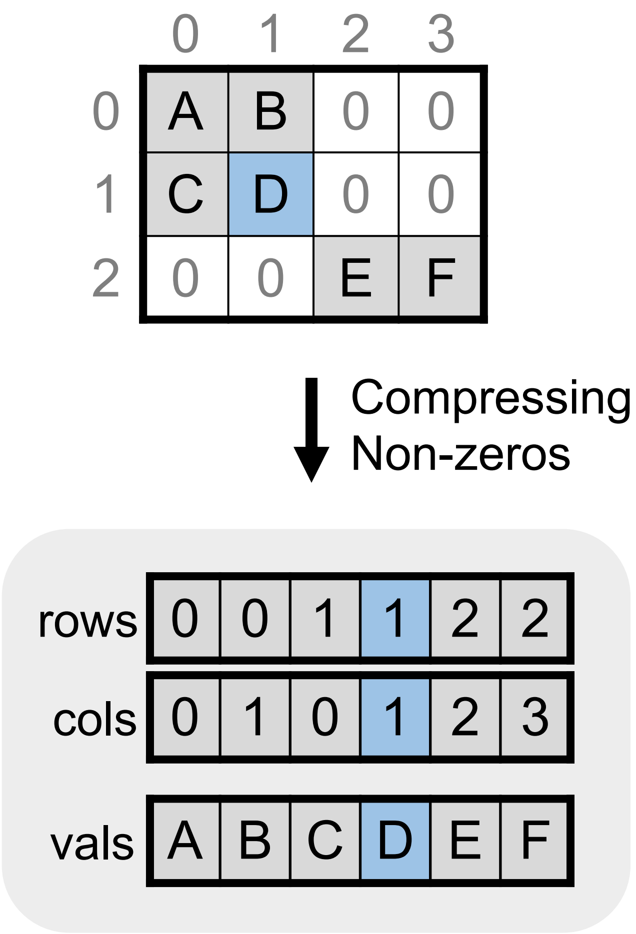


Sparse
Data Representation

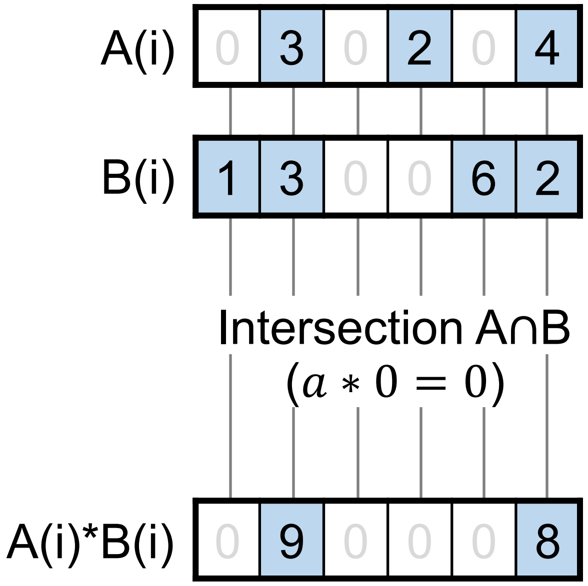


Skipping
Ineffectual Computation

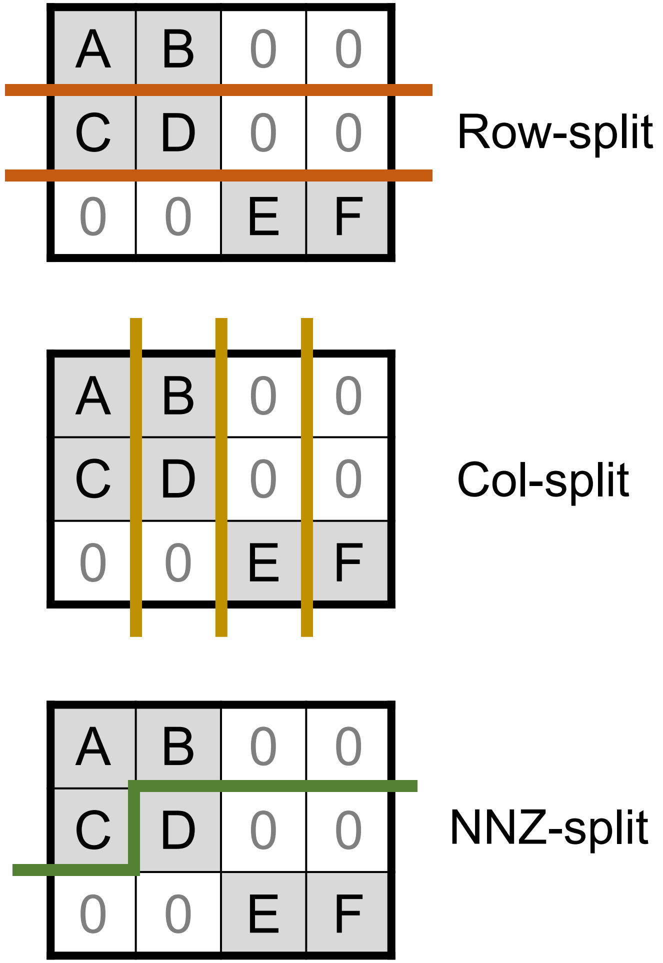
Writing Sparse Code is Hard



Sparse
Data Representation

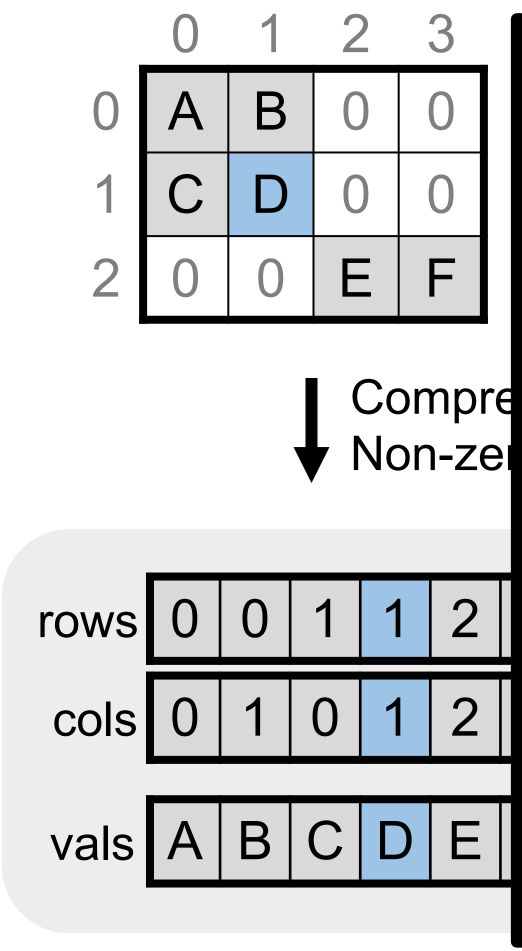


Skipping
Ineffectual Computation



Different Loop Traversal

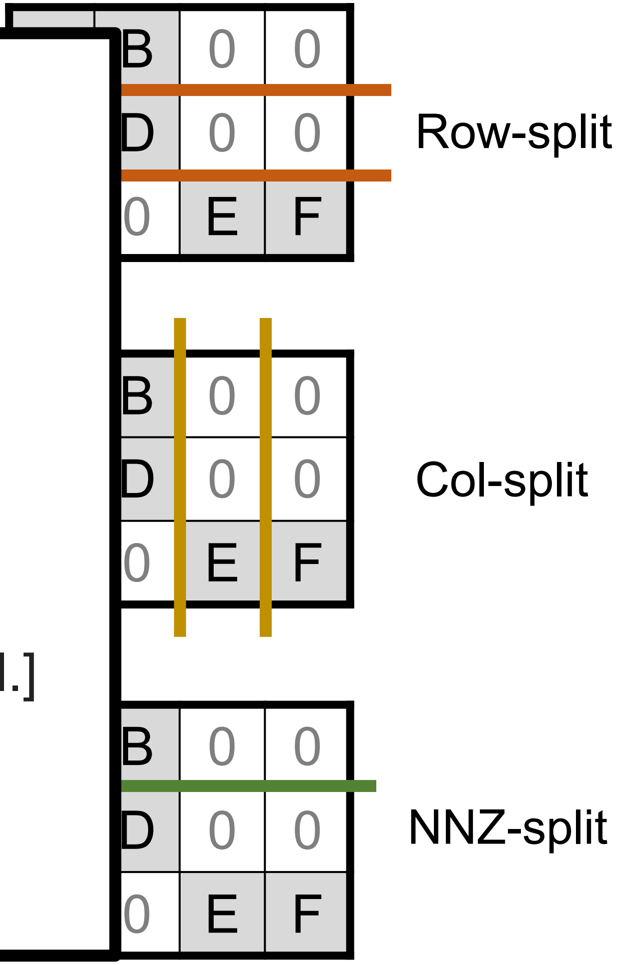
Writing Sparse Code is Hard



Compiler for Sparse Computation

- Tensor Algebra Compiler [Kjolstad et al.]
- Taichi [Hu et al.]
- SparseTIR [Ye et al.]
- Sparse CHiLL [Venkat et al.]
- Sparse Polyhedral Framework [Strout et al.]
- Sparse MLIR [Bik et al.]

...

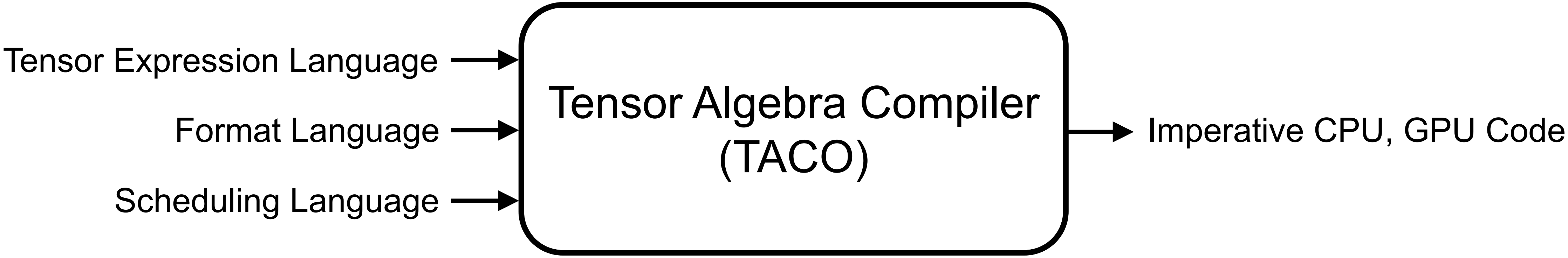


Sparse Data Representation

Skipping Ineffectual Computation

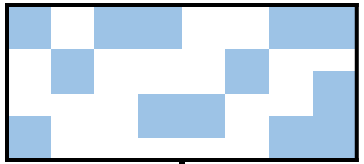
Different Loop Traversal

Writing Sparse Code is Hard



Writing Sparse Code is Hard

Input Sparse Matrix $A_{i,j}$



(Matrix-Vector Multiply)

Tensor Expression Language

$y_i = A_{i,j} * x_j$

Format Language

Format of $A_{i,j}$ = COO

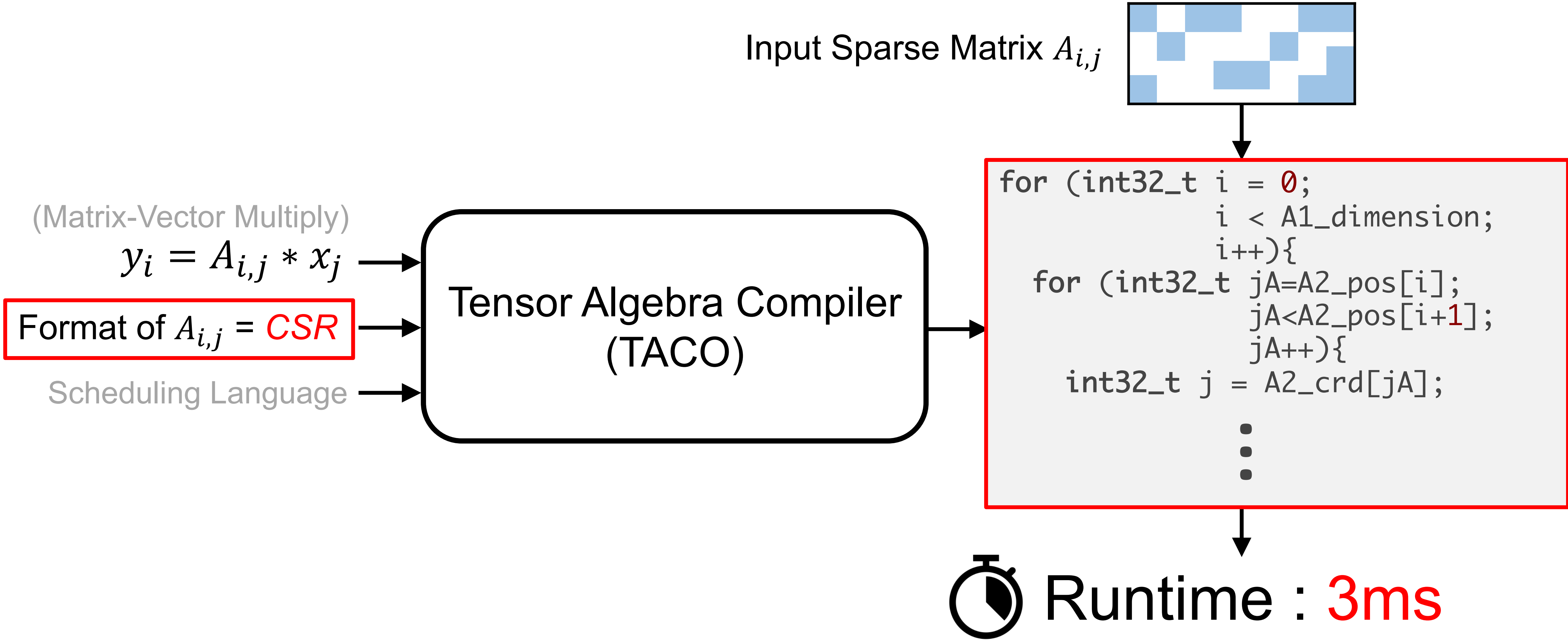
Scheduling Language

Tensor Algebra Compiler (TACO)

```
int32_t iA = A1_pos[0];
int32_t pA1_end = A1_pos[1];
while (iA < pA1_end) {
  int32_t i = A1_crd[iA];
  int32_t A1_segend = iA + 1;
  while (A1_segend < pA1_end &&
         A1_crd[A1_segend] == i){
          ⋮
```

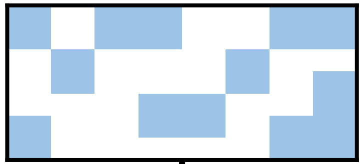
Runtime : 10ms

Writing Sparse Code is Hard



Writing Sparse Code is Hard

Input Sparse Matrix $A_{i,j}$



(Matrix-Vector Multiply)

$y_i = A_{i,j} * x_j$

Format of $A_{i,j}$ = *CSR*

```
.split(i,i0,i1,32)
.reorder(i0,i1,j)
.parallelize(i0)
```

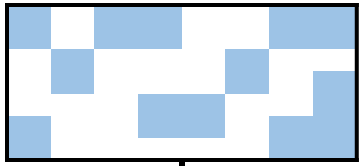
Tensor Algebra Compiler (TACO)

```
#pragma omp parallel for
for (int32_t i0=0;
      i0<(A1_dimension/32); i0++)
{
  for (int32_t i1=0; i1<32; i1++)
  {
    int32_t i = i0*32 + i1;
          ⋮
```

Runtime : 1ms

Writing Fast Sparse Code is Hard!

Input Sparse Matrix $A_{i,j}$



(Matrix-Vector Multiply)

$y_i = A_{i,j} * x_j$

Format : ???

Schedule : ???

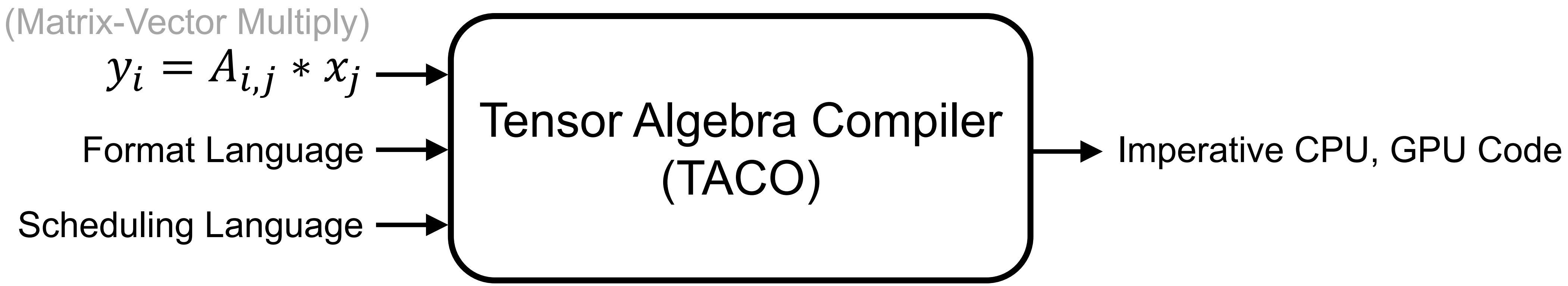
Tensor Algebra Compiler
(TACO)

Optimal
Generated Code

Fastest Runtime

What would be the optimal format and schedule?

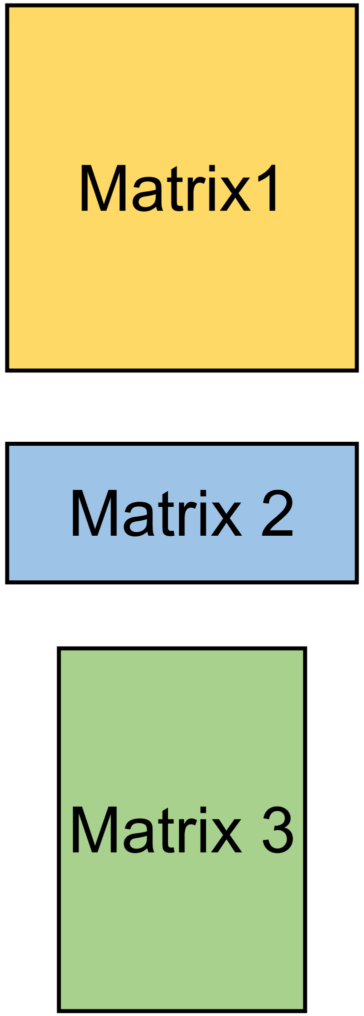
Writing Sparse Code is Hard



Writing Fast Tensor Program is Hard!

Optimization depends on **tensor's shape**

Dense Matrix



Optimal Loop Transformation
(Optimal Scheduling Language)

```
.split(i,i1,i0,256)
.split(k,k1,k0,256)
.split(j,j1,j0,16)
.reorder(i1,k1,j1,i0,k0,j0)
.unroll(k0,4)
.vectorize(i0)
.parallelize(i1)
```

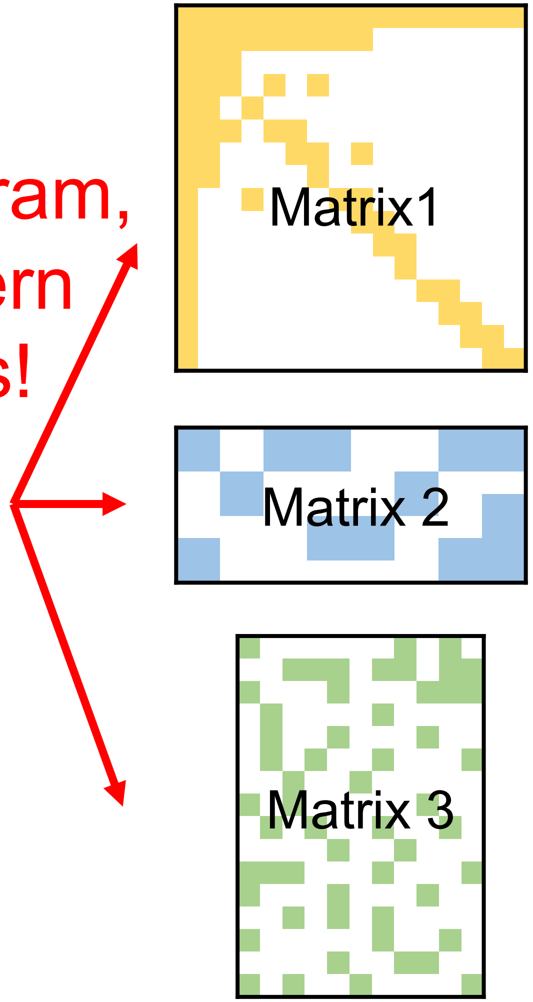
```
.split(i,i1,i0,64)
.reorder(i1,k,i0)
.parallelize(i1)
```

```
.split(i,i1,i0,64)
.split(k,k1,k0,16)
.reorder(k1,i1,i0,k0)
.parallelize(i1)
```

Writing Fast Sparse Tensor Program is Even Harder!

In sparse program, sparsity pattern now matters!

Sparse Matrix | **Optimal Loop Transformation**

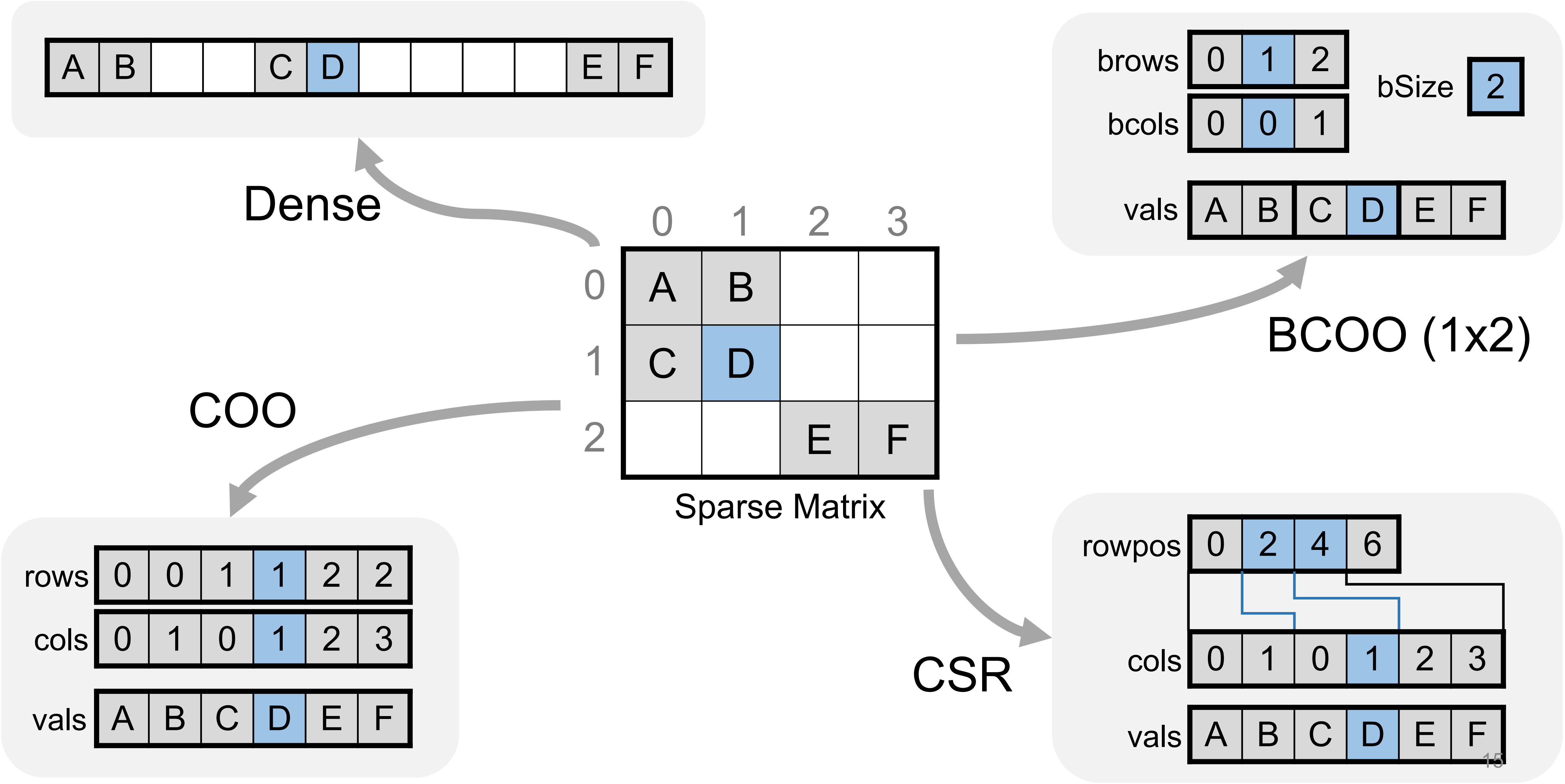



```
.split(i,i1,i0,256)
.split(k,k1,k0,256)
.split(j,j1,j0,16)
.reorder(i1,k1,j1,i0,k0,j0)
.unroll(k0,4)
.vectorize(i0)
.parallelize(i1)
```

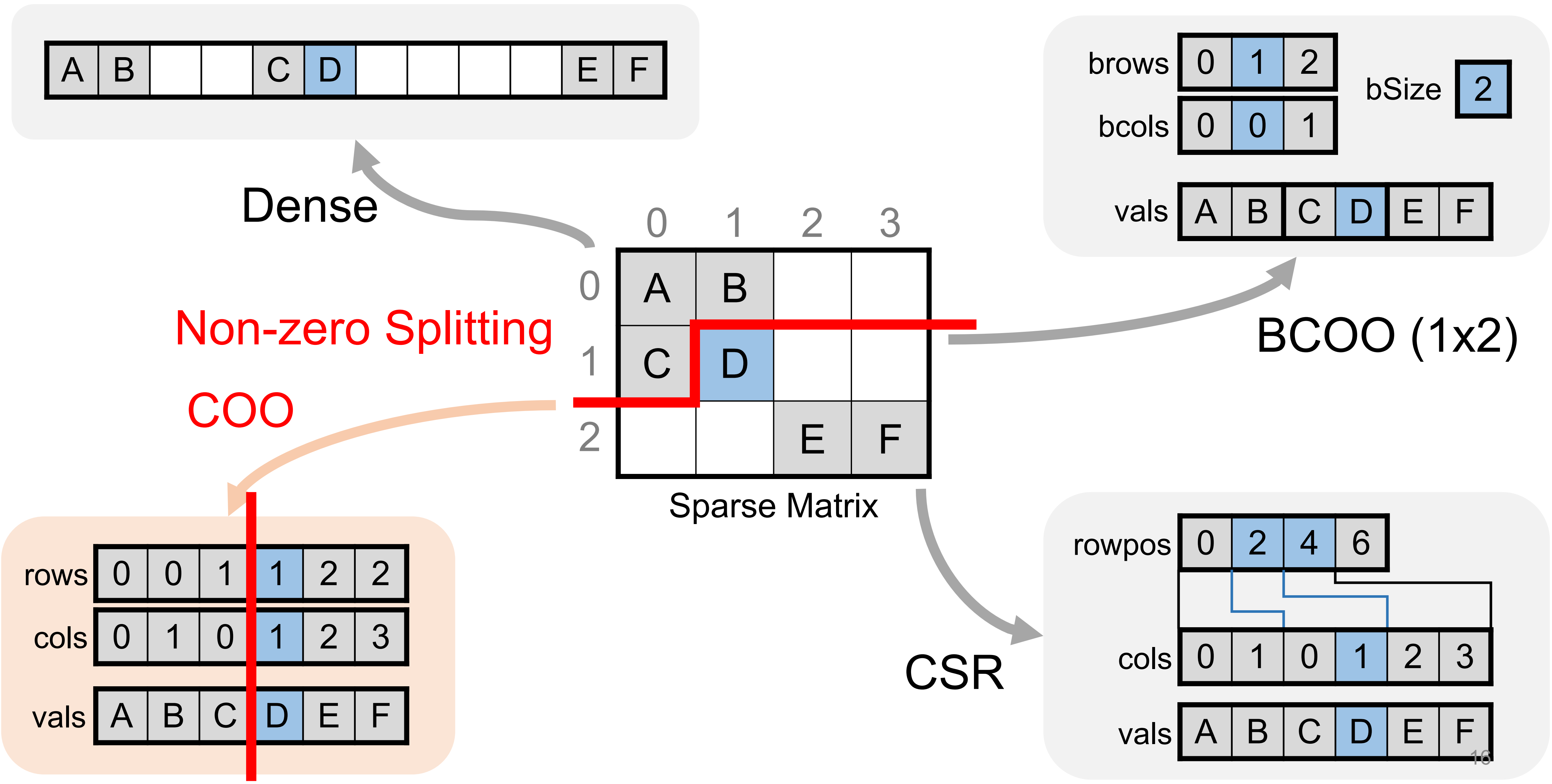
```
.split(i,i1,i0,64)
.reorder(i1,k,i0)
.parallelize(i1)
```

```
.split(i,i1,i0,64)
.split(k,k1,k0,16)
.reorder(k1,i1,i0,k0)
.parallelize(i1)
```

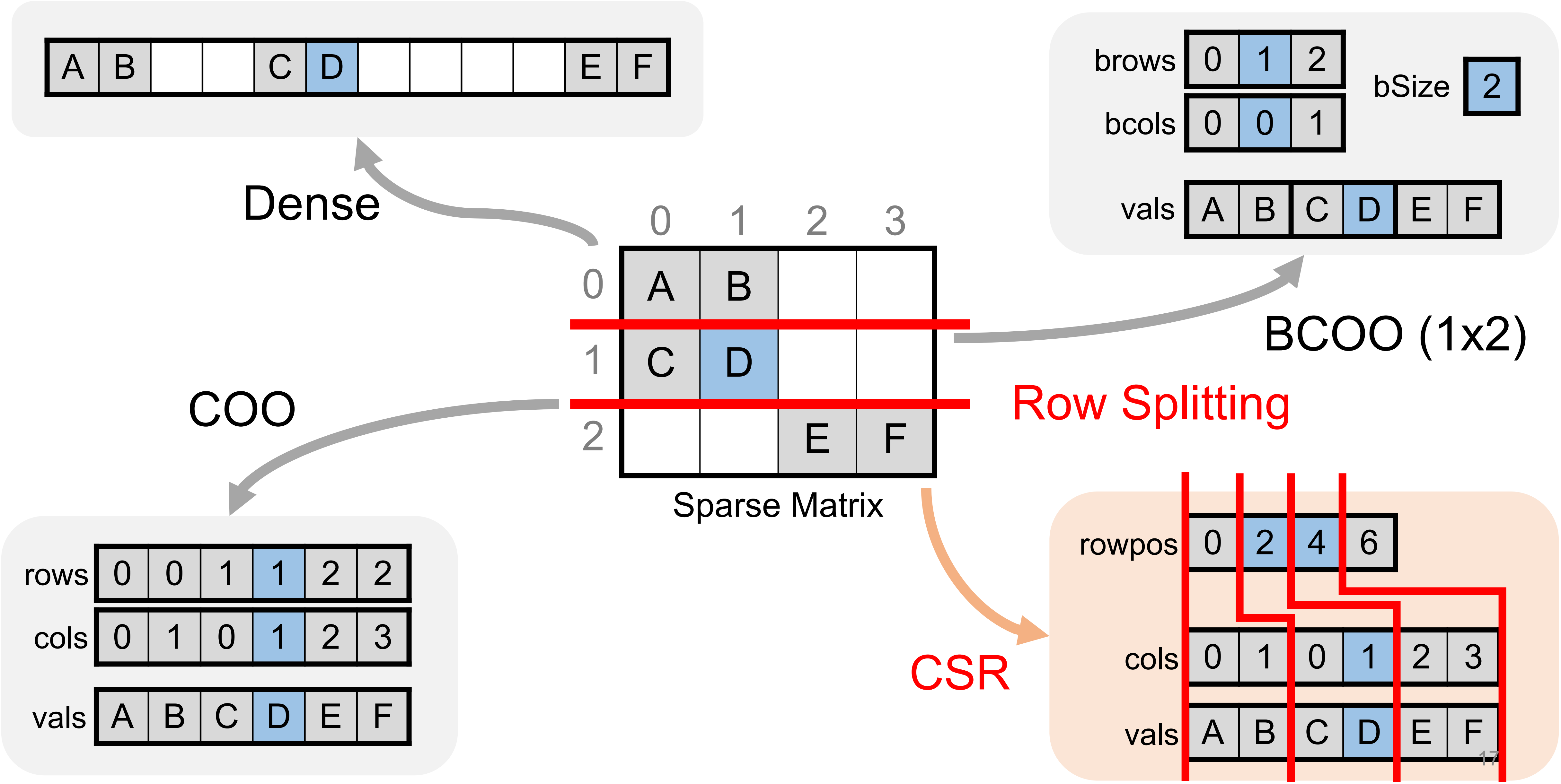
Writing Fast Sparse Tensor Program is Even Harder!



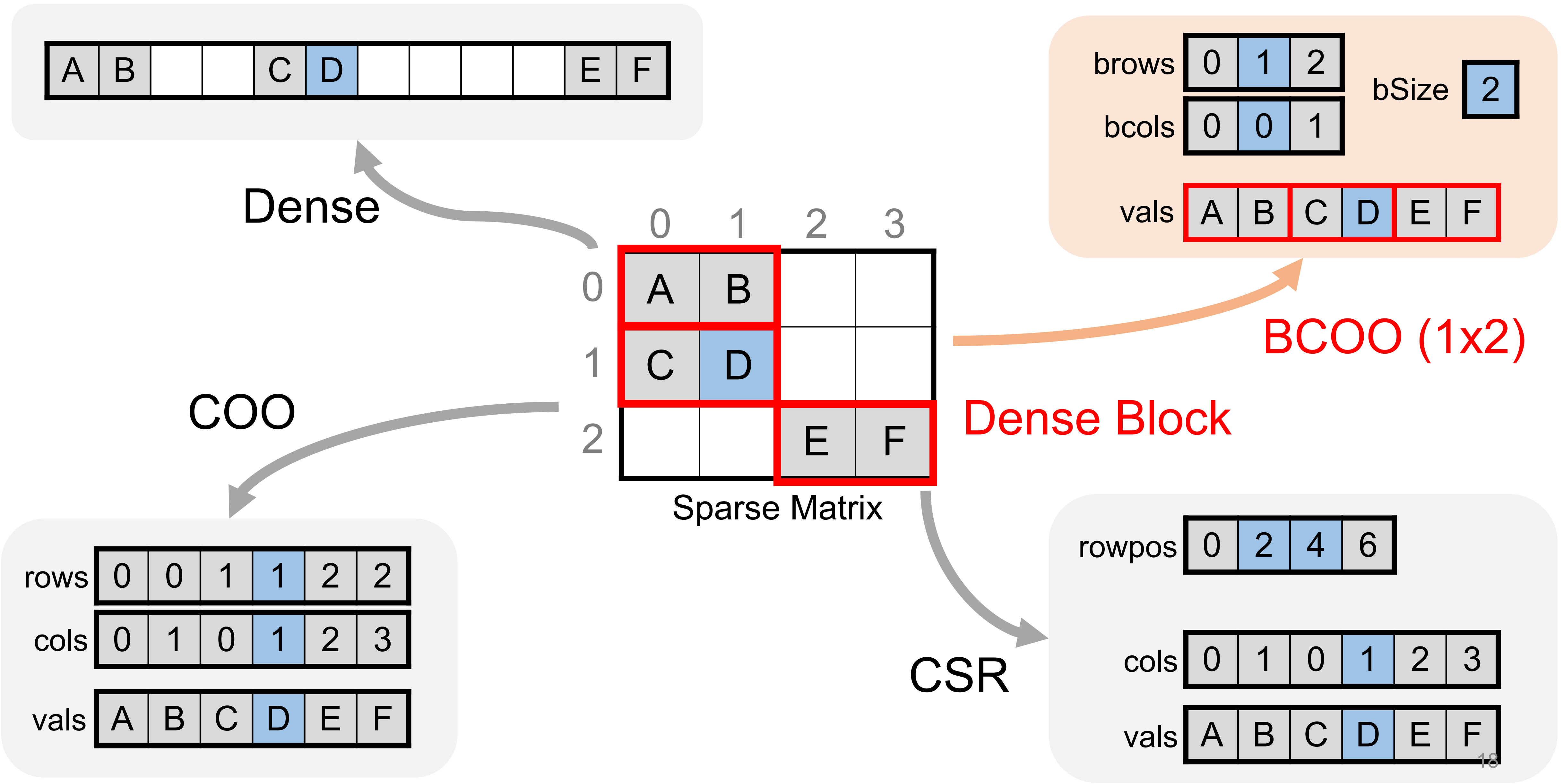
Writing Fast Sparse Tensor Program is Even Harder!



Writing Fast Sparse Tensor Program is Even Harder!



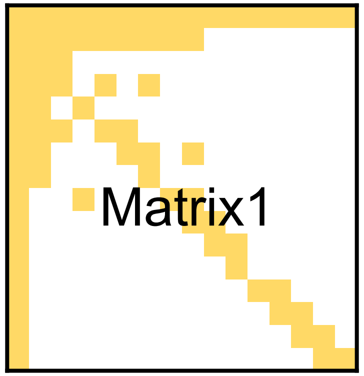
Writing Fast Sparse Tensor Program is Even Harder!



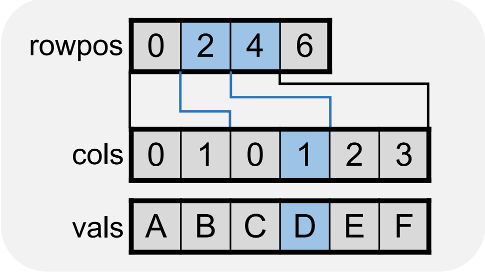
Writing Fast Sparse Tensor Program is Even Harder!

In sparse program, sparsity pattern now matters!

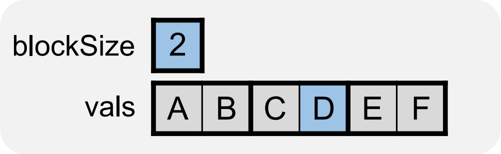
Sparse Matrix	Optimal Loop Transformation	Optimal Sparse Format

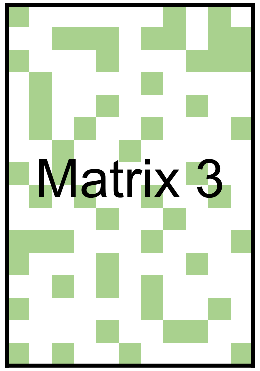



```
.split(i,i1,i0,256)
.split(k,k1,k0,256)
.split(j,j1,j0,16)
.reorder(i1,k1,j1,i0,k0,j0)
.unroll(k0,4)
.vectorize(i0)
.parallelize(i1)
```

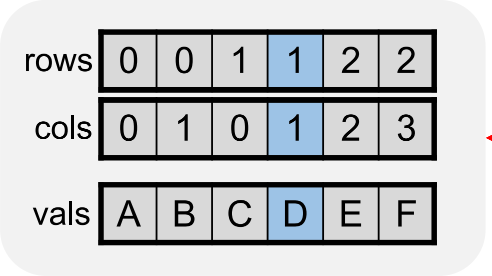



```
.split(i,i1,i0,64)
.reorder(i1,k,i0)
.parallelize(i1)
```





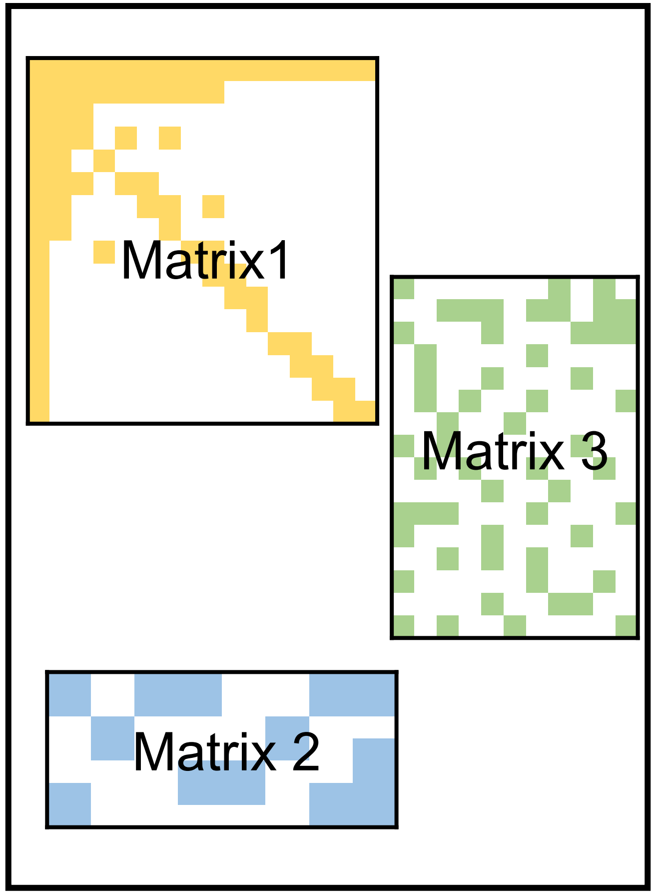

```
.split(i,i1,i0,64)
.split(k,k1,k0,16)
.reorder(k1,i1,i0,k0)
.parallelize(i1)
```



Sparse format also matters!

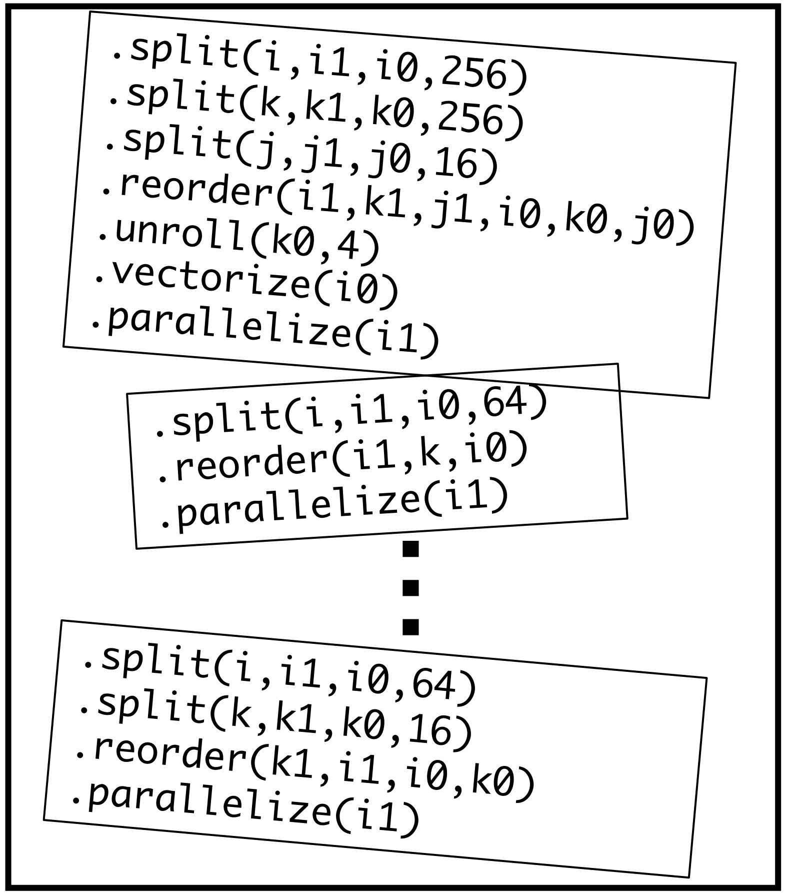
Writing Fast Sparse Tensor Program is Even Harder!

Given Sparsity Pattern



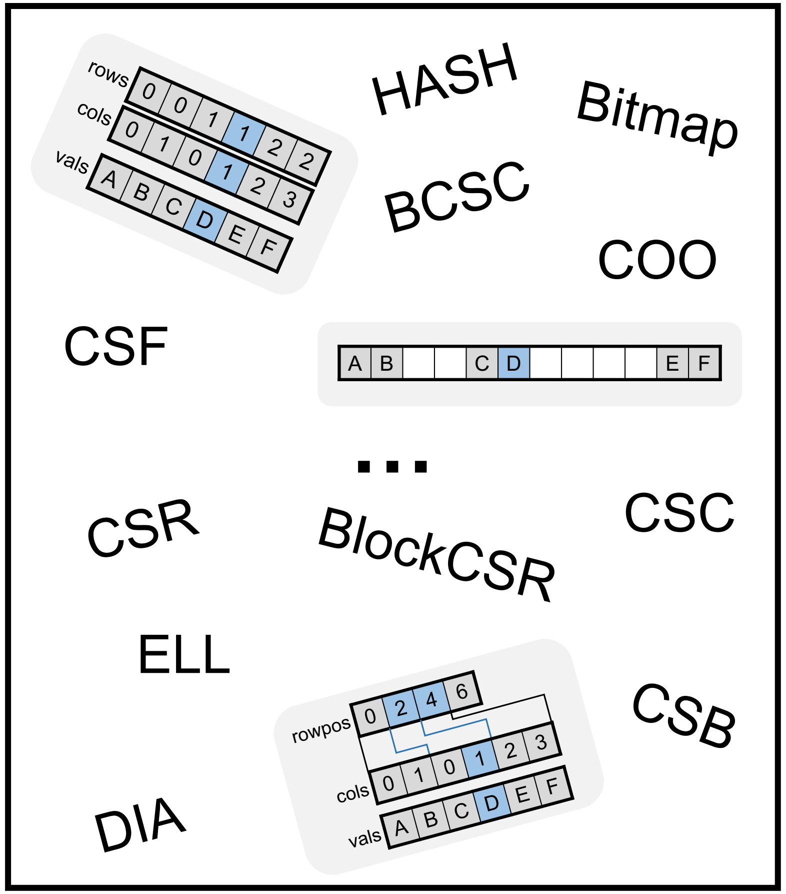
×

Choice of Schedules
(Choice of Loop Transformations)



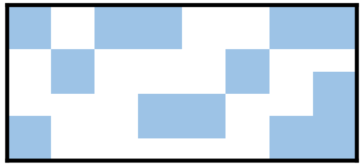
×

Choice of Formats
(Choice of Data Representations)



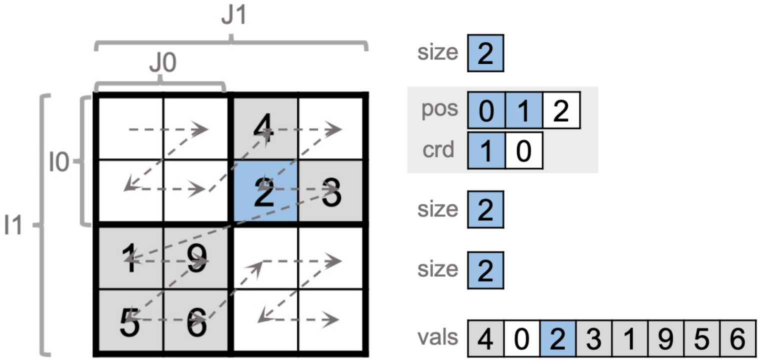
Given an input sparsity pattern, what is the best schedule and format?

Proposed Approach: WACO



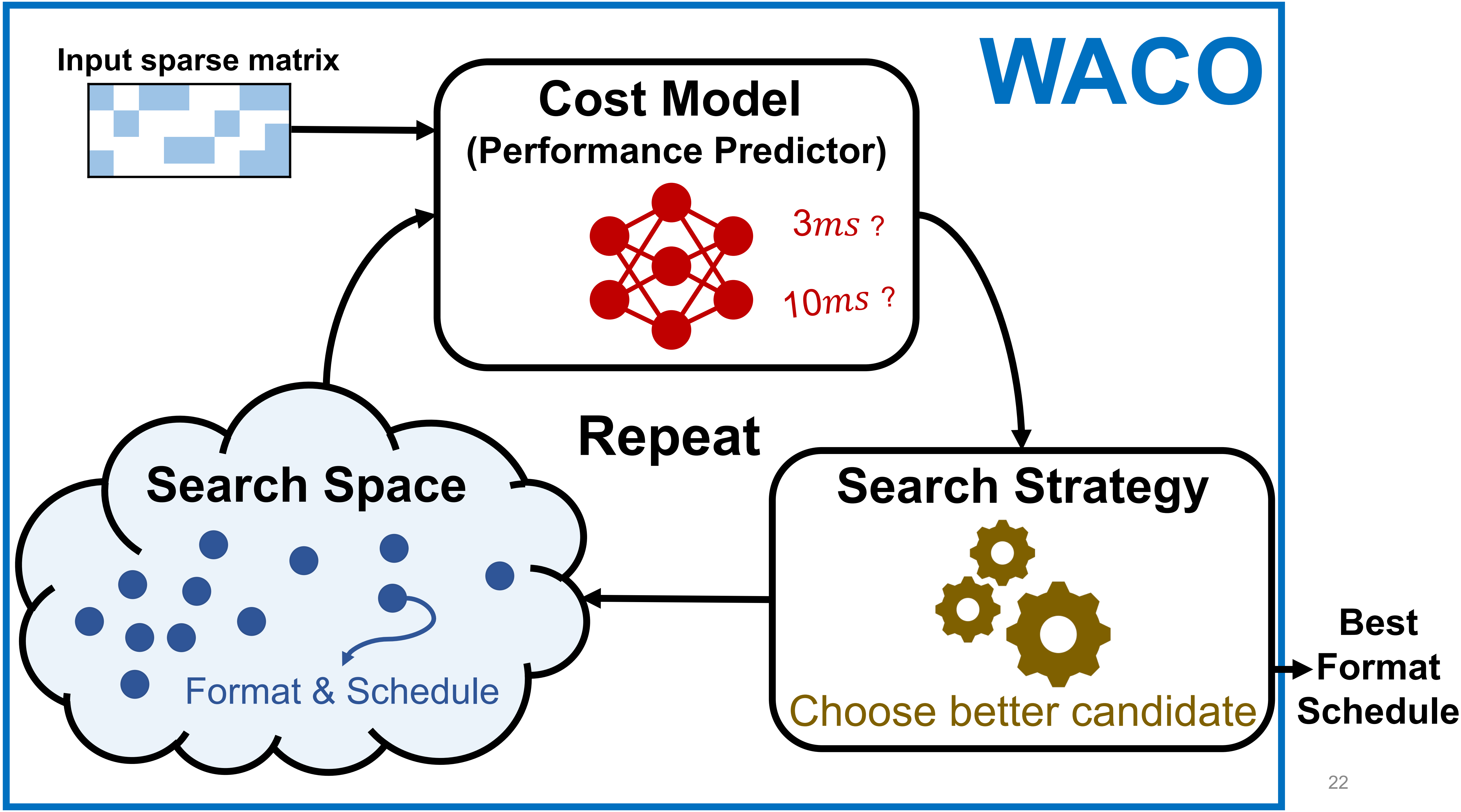
Input sparse matrix

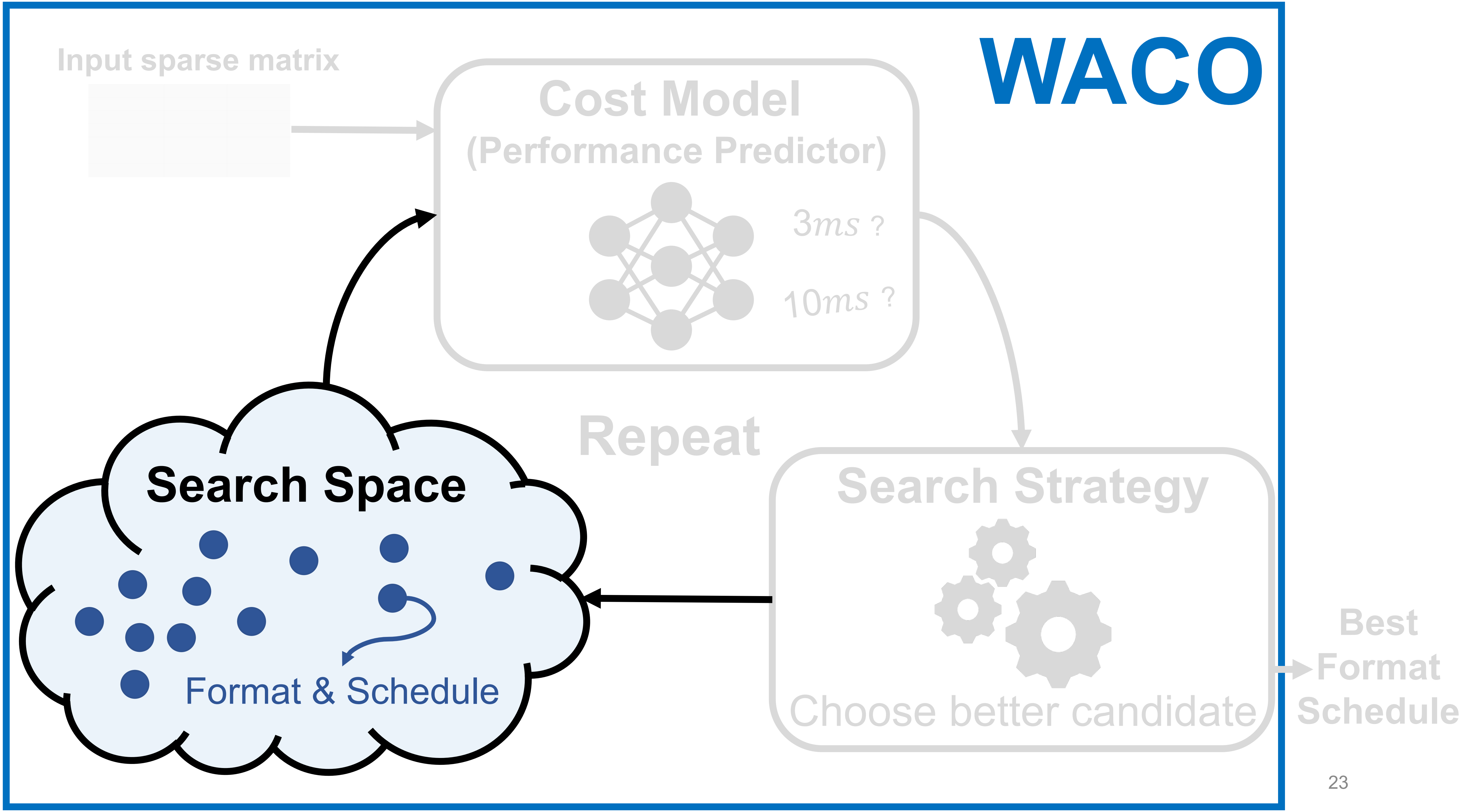
WACO
(Workload-Aware Co-Optimization)




```
.split(i,i1,i0,4)
.split(k,k1,k0,2)
.split(j,j1,j0,32)
.reorder(i1,k1,j1,i0,k0,j0)
.parallelize(i1,48,4)
```

Co-Optimized Format and Schedule


WACO
Input sparse matrix
Cost Model
(Performance Predictor)
$3ms$?
$10ms$?
Repeat
Search Space
Format & Schedule
Search Strategy
Choose better candidate
Best
Format
Schedule



WACO
Input sparse matrix
Cost Model
(Performance Predictor)
3ms ?
10ms ?
Repeat
Search Space
Format & Schedule
Search Strategy
Choose better candidate
Best
Format
Schedule

WACO : Search Space

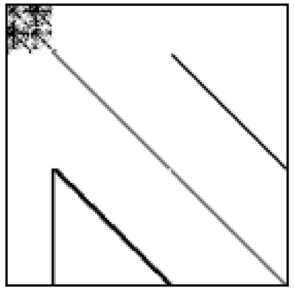
1. Existing approach considers either format or schedule

2. Existing approach considers small search space

WACO : Search Space

1. Existing approach considers either format or schedule

TSOPF_RS_b2052_c1



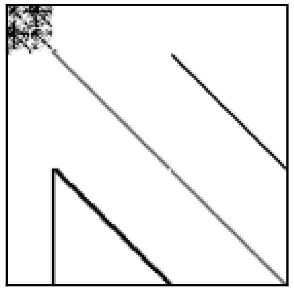
SpMM	Format-only	Schedule-only	Co-optimization
Speedup	1.11×	1.12×	**2.02×**

2. Existing approach considers small search space

WACO : Search Space

1. Existing approach considers either format or schedule

TSOPF_RS_b2052_c1



	Format-only	Schedule-only	Co-optimization
Speedup	1.11×	1.12×	**2.02×**

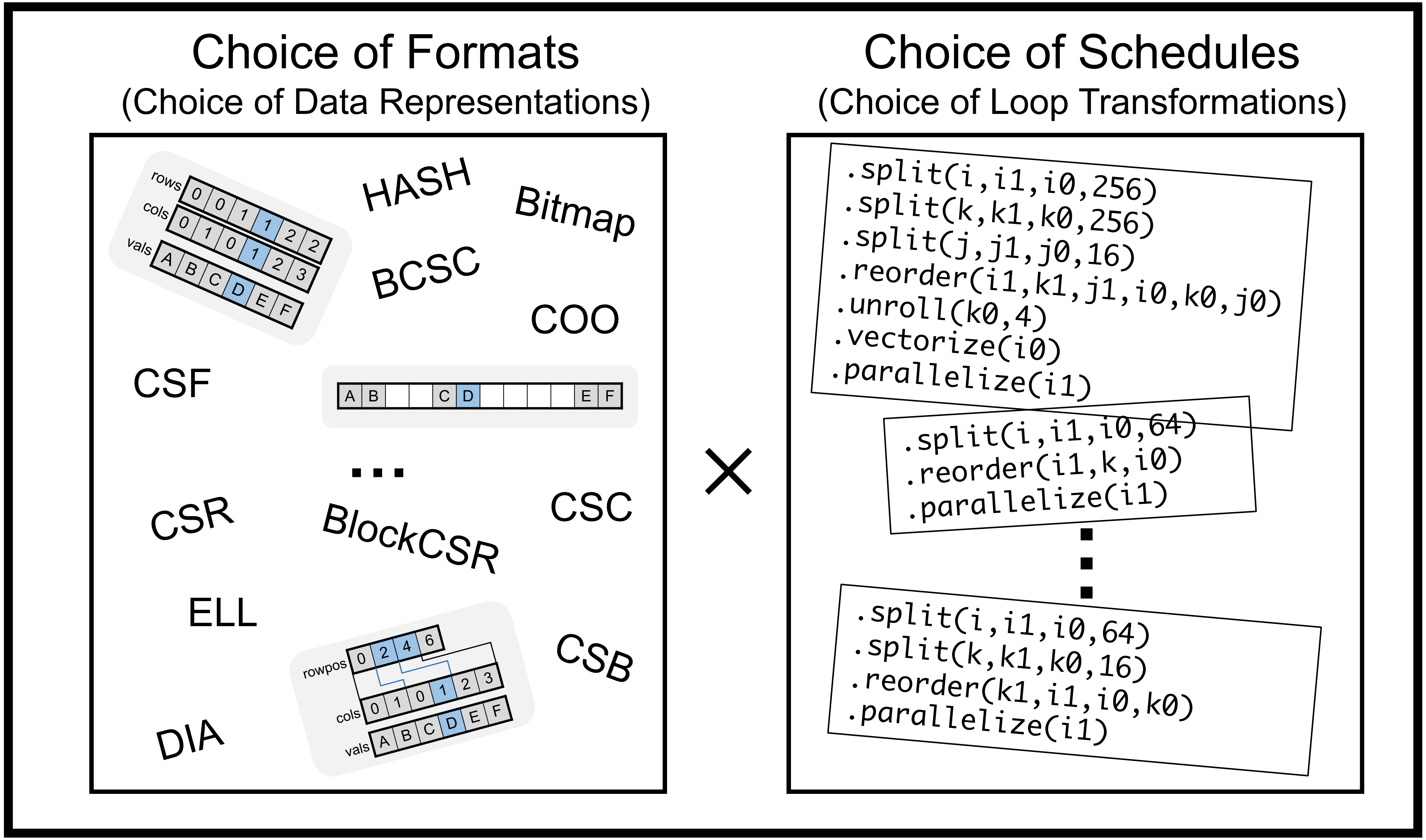
2. Existing approach considers small search space

PLDI’13 [Li et al.]
4 formats

PPoPP’18 [Zhao et al.]
4 formats

SC’20 [Sun et al.]
5 formats

WACO : Search Space



WACO : Search Space

(Matrix-Vector Multiply)

SuperSchedule Template of $C_i = A_{i,k} * B_k$

```
.split(i,i1,i0,?)
.split(k,k1,k0,?)
.reorder(?, ?, ?, ?)
.parallelize(?, ?)
```

Compute Schedule

```
A.reorder(?,?,?,?)
A.lvlFormat(i1,?)
A.lvlFormat(i0,?)
A.lvlFormat(k1,?)
A.lvlFormat(k0,?)
```

Format Schedule

WACO : Search Space

```
.split(i,i1,i0,?)
.split(k,k1,k0,?)
.reorder(?, ?, ?, ?)
.parallelize(?, ?)
```

Compute Schedule

```
for i in range(32):
  for k in range(32):
```

Initial loop

WACO : Search Space

```
.split(i,i1,i0,2)
.split(k,k1,k0,2)
.reorder(i1,k1,i0,k0)
.parallelize(i1, 4)
```

Compute Schedule

```
for i in range(32):
  for k in range(32):
```

Initial loop

```
#pragma omp schedule(dynamic,4)
parallel-for i1 in range(16):
  for k1 in range(16):
    for i0 in range(2):
      for k0 in range(2):
```

Transformed loop

Determines what loop transformations to apply.

WACO : Search Space

SuperSchedule Template of $C_i = A_{i,k} * B_k$

```
.split(i,i1,i0,[?])
.split(k,k1,k0,[?])
.reorder([?], [?], [?], [?])
.parallelize([?], [?])
```

Compute Schedule

```
A.reorder([?],[?],[?],[?])
A.lvlFormat(i1,[?])
A.lvlFormat(i0,[?])
A.lvlFormat(k1,[?])
A.lvlFormat(k0,[?])
```

Format Schedule

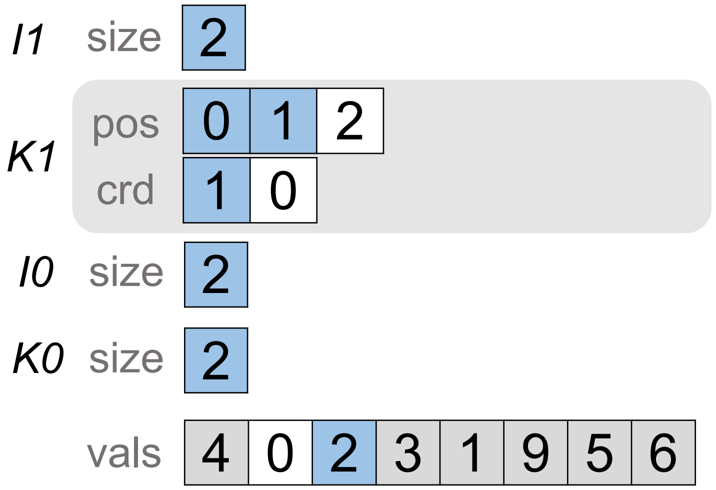
WACO : Search Space

```
A.reorder(?,?,?,?)
A.lvlFormat(i1,?)
A.lvlFormat(i0,?)
A.lvlFormat(k1,?)
A.lvlFormat(k0,?)
```

Format Schedule

Different Format Schedules made different formats.

WACO : Search Space



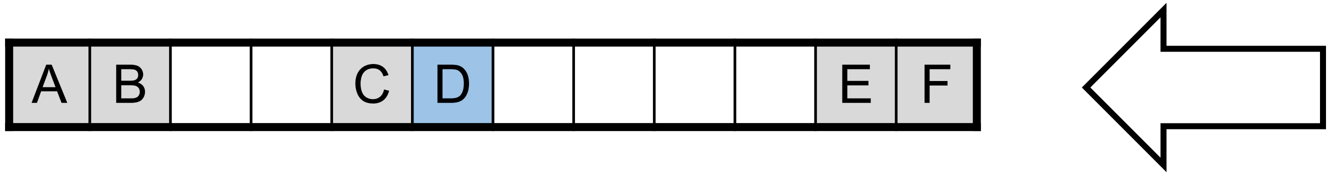
BCSR with 2x2 block

```
A.reorder(i1,k1,i0,k0)
A.lvlFormat(i1,Uncompressed)
A.lvlFormat(i0,Compressed)
A.lvlFormat(k1,Uncompressed)
A.lvlFormat(k0,Uncompressed)
```

Format Schedule

Different Format Schedules made different formats.

WACO : Search Space



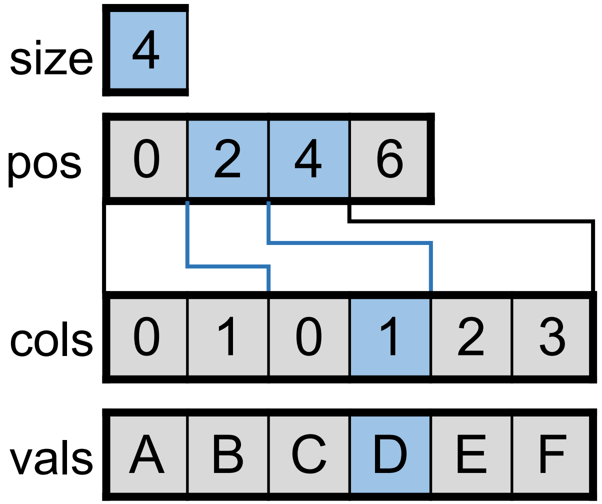
Dense Format

```
A.reorder(i1,i0,k1,k0)
A.lvlFormat(i1,Uncompressed)
A.lvlFormat(i0,Uncompressed)
A.lvlFormat(k1,Uncompressed)
A.lvlFormat(k0,Uncompressed)
```

Format Schedule

Different Format Schedules made different formats.

WACO : Search Space



Compressed Sparse Row Format

```
A.reorder(i1,i0,k1,k0)
A.lvlFormat(i1,Uncompressed)
A.lvlFormat(i0,Uncompressed)
A.lvlFormat(k1,Compressed)
A.lvlFormat(k0,Compressed)
```

Format Schedule

Different Format Schedules made different formats.

WACO : Search Space

SuperSchedule Template of $C_i = A_{i,k} * B_k$

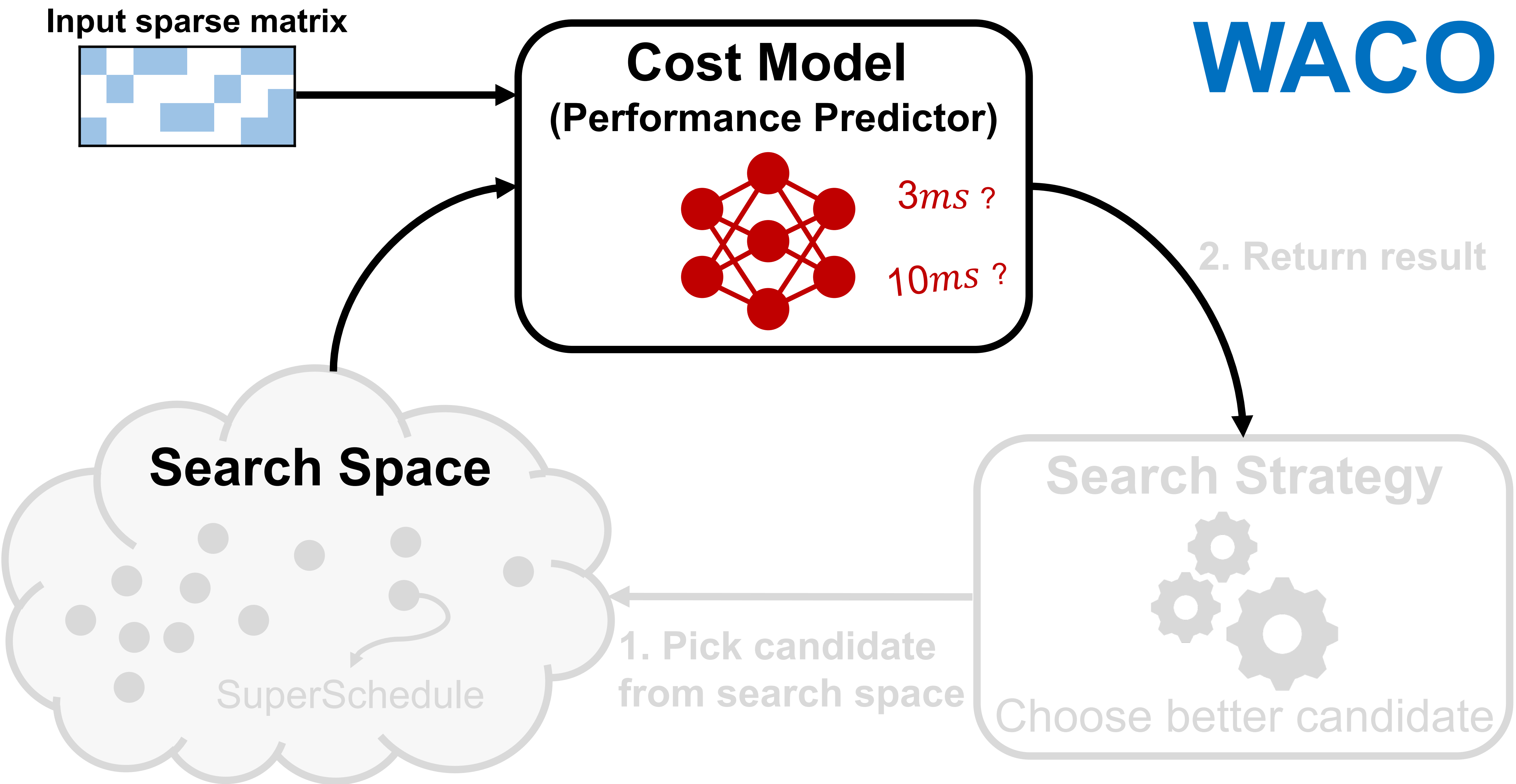
```
.split(i,i1,i0,?)
.split(k,k1,k0,?)
.reorder(?, ?, ?, ?)
.parallelize(?, ?)
```

Compute Schedule

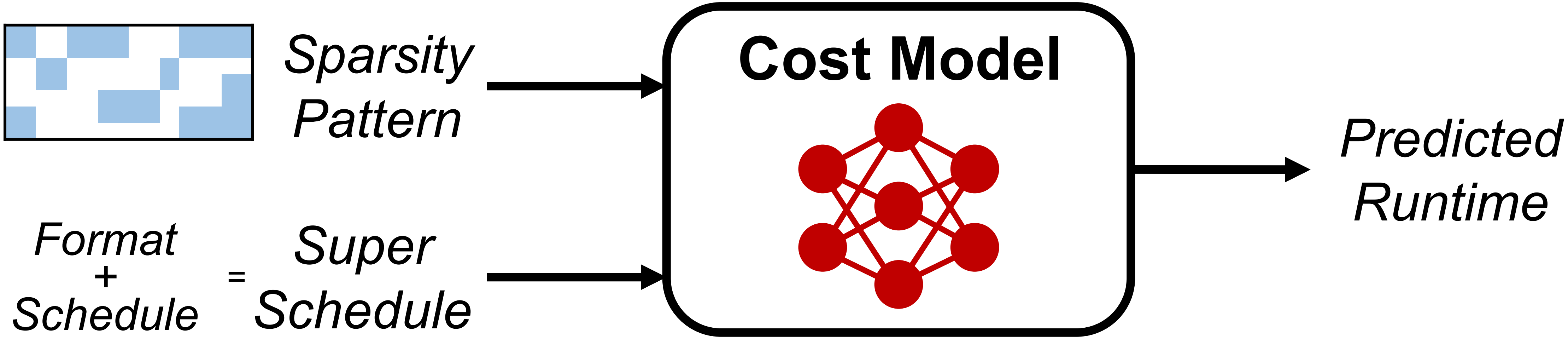
```
A.reorder(?,?,?,?)
A.lvlFormat(i1,?)
A.lvlFormat(i0,?)
A.lvlFormat(k1,?)
A.lvlFormat(k0,?)
```

Format Schedule

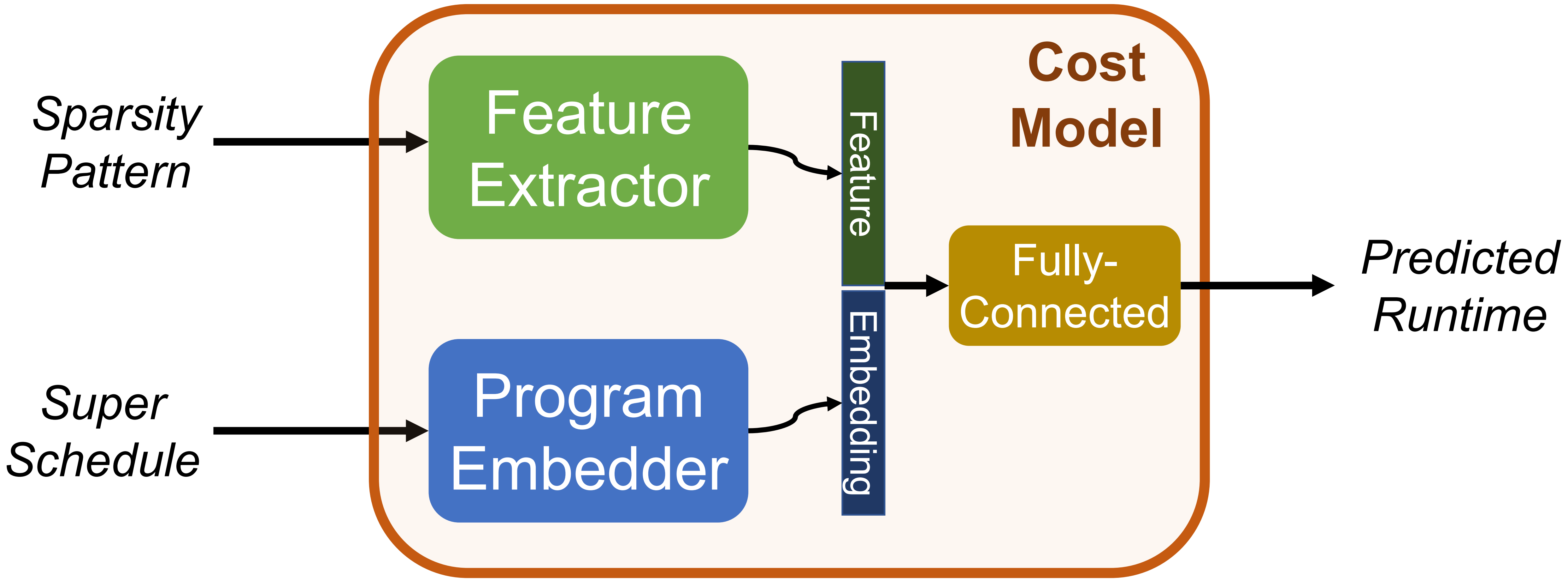
1. Our space considers both format and schedule.
2. Our space contains $\sim 10^6$ SuperSchedules.


Input sparse matrix
Cost Model
(Performance Predictor)
3ms ?
10ms ?
WACO
2. Return result
Search Space
SuperSchedule
1. Pick candidate
from search space
Search Strategy
Choose better candidate

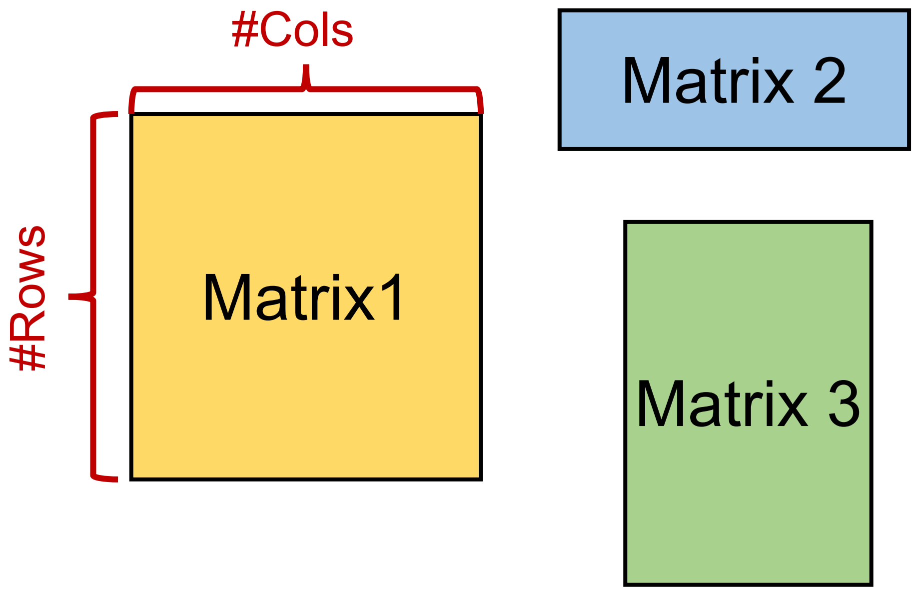
WACO : Cost Model



WACO : Cost Model



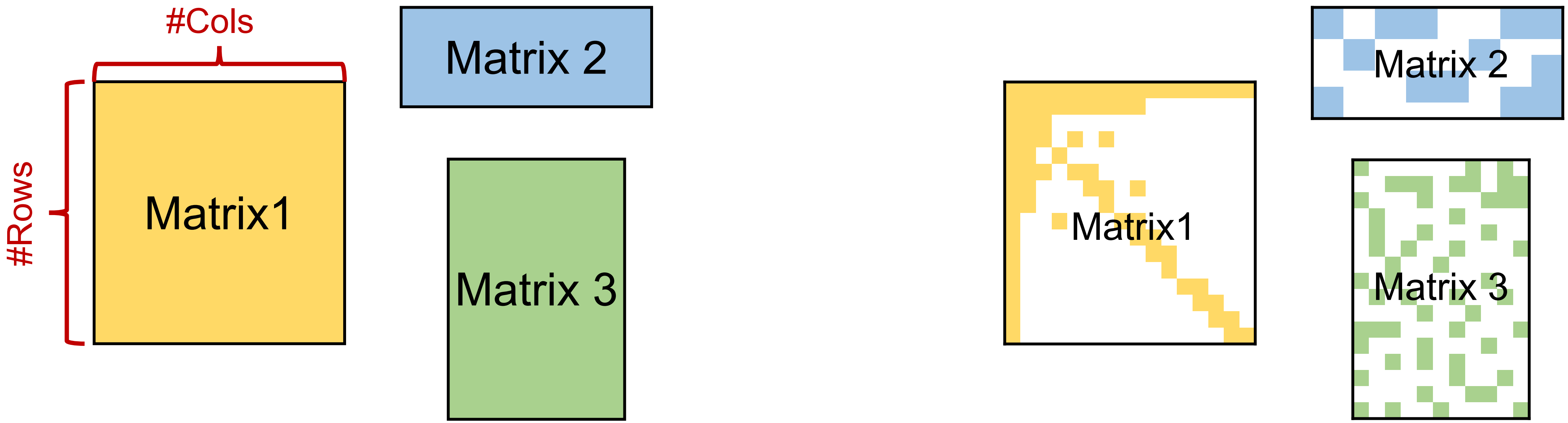
WACO : Cost Model (Pattern Feature Extractor)



Dense World

[#Rows, #Cols]

WACO : Cost Model (Pattern Feature Extractor)



Dense World

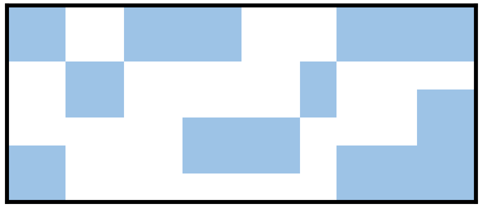
[#Rows, #Cols]

Sparse World

Is this enough?

WACO : Cost Model (Pattern Feature Extractor)

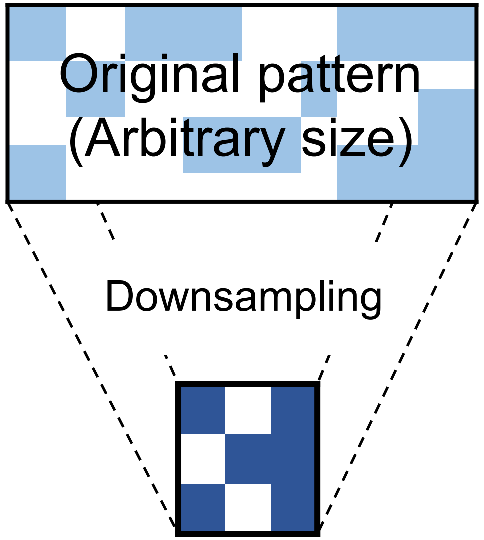
Human-crafted features



Expert-Knowledge

Feature List
Number of Rows
Number of Cols
Number of Non-Zeros
Average NNZ per row
Min/Max NNZ per row
...

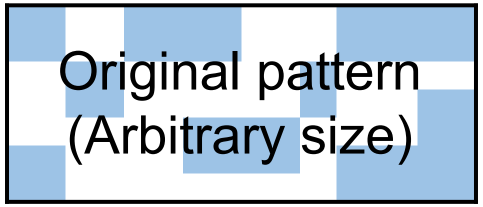
CNN after downsampling



Resized Image (e.g,128x128)

Convolutional Neural Network

Our Approach
(Submanifold Sparse CNN)

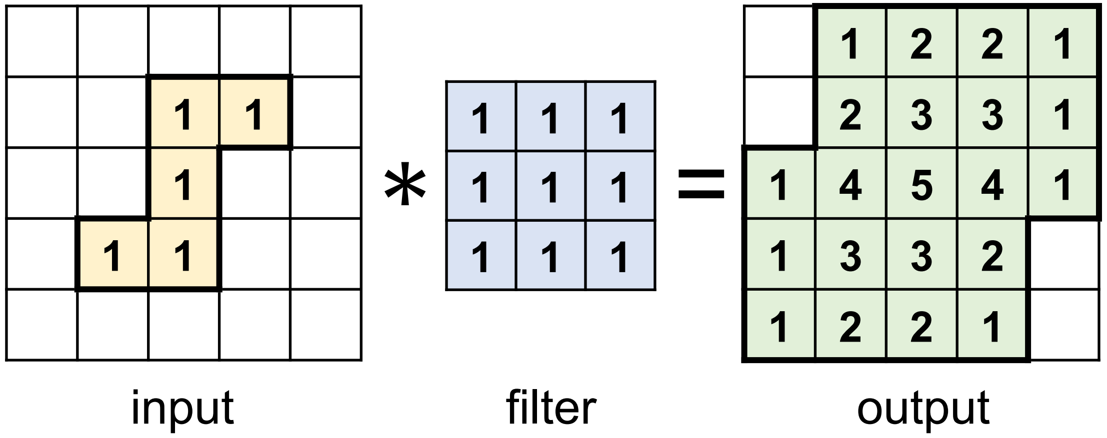


Submanifold Sparse Convolutional Neural Network*

Graham, Benjamin, and Laurens Van der Maaten. "Submanifold sparse convolutional networks." *arXiv preprint arXiv:1706.01307* (2017)

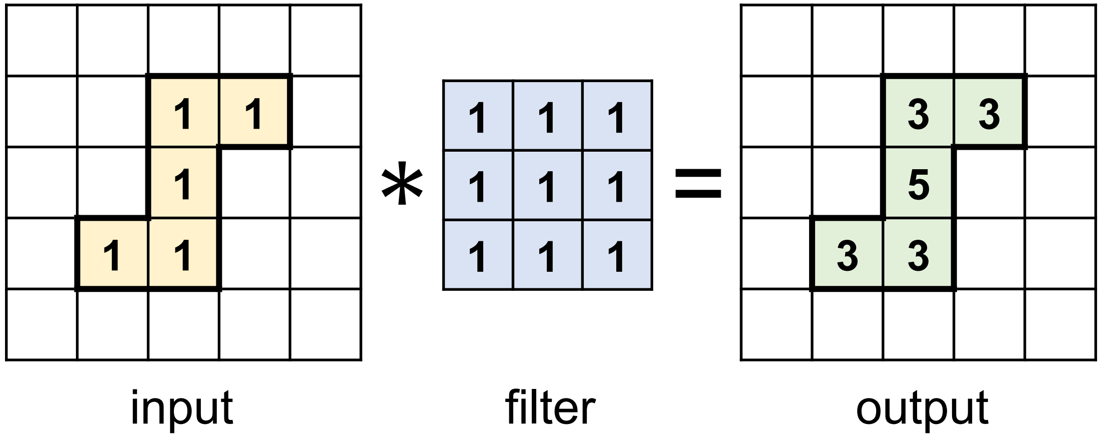
WACO : Cost Model (Pattern Feature Extractor)

Conventional Convolution



Nonzero area grows quickly

Submanifold Sparse Convolution[+]

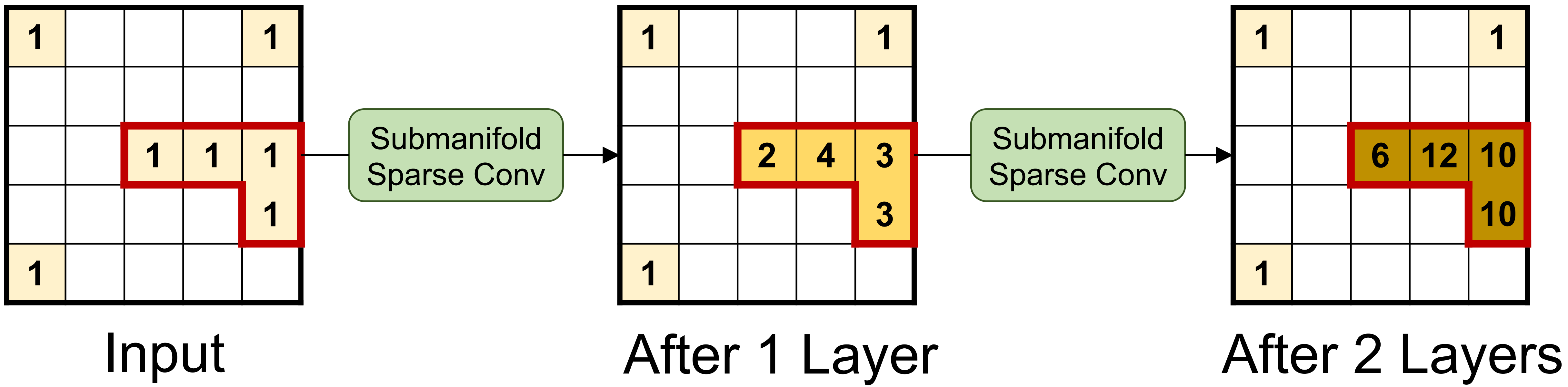


Sparsity pattern is unchanged

+ Graham, Benjamin, and Laurens Van der Maaten. "Submanifold sparse convolutional networks." *arXiv preprint arXiv:1706.01307* (2017)

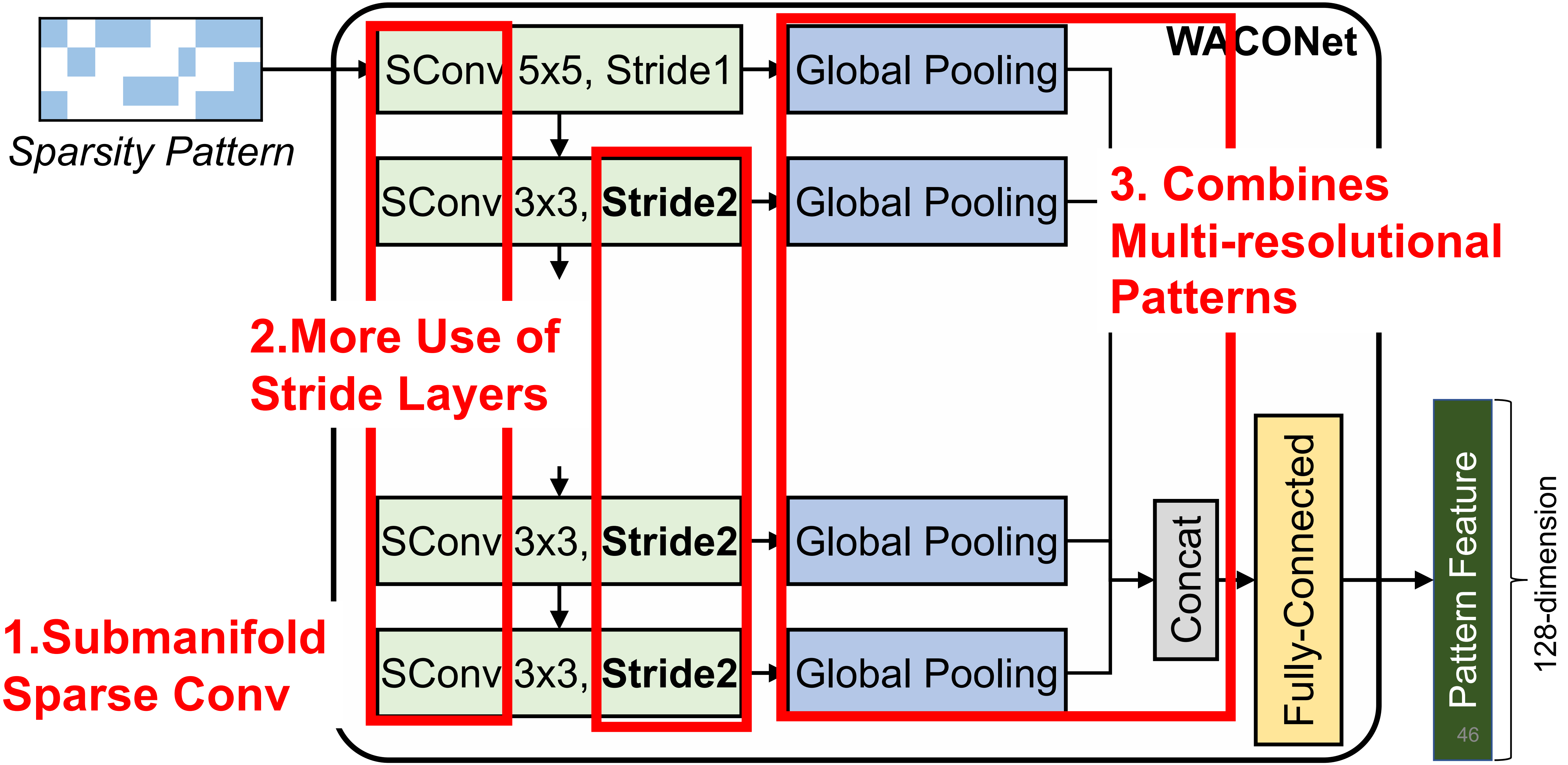
WACO : Cost Model (Pattern Feature Extractor)

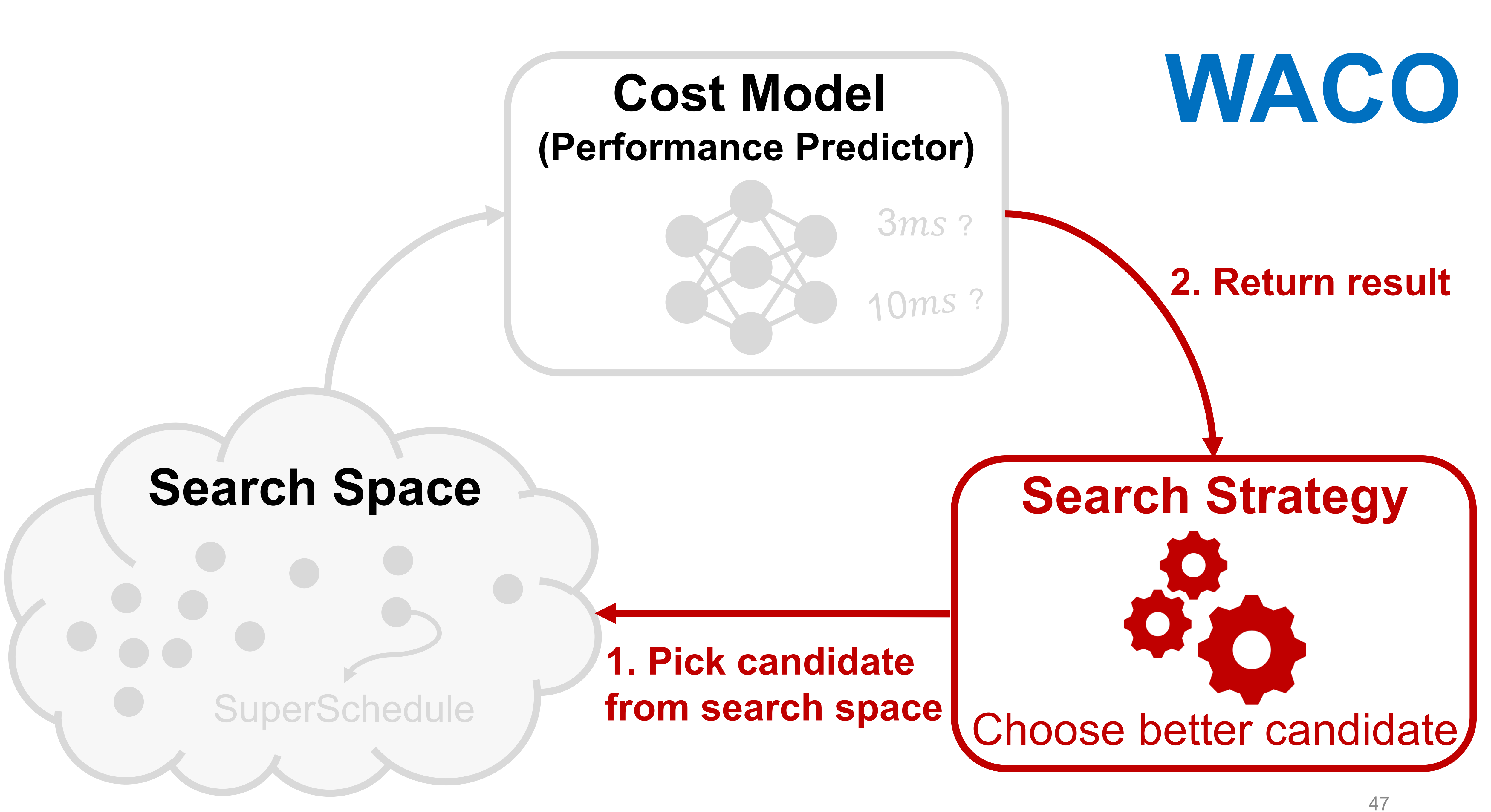
When we simply use a popular submanifold vision model,



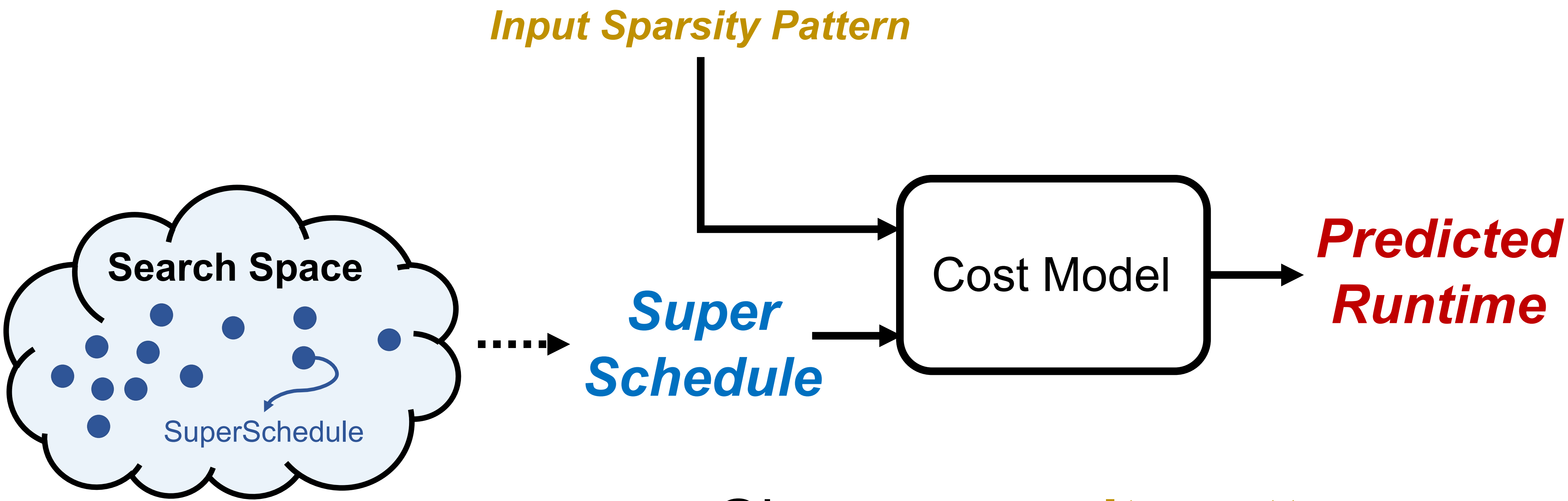
Information does not propagate across distant non-zeros!

WACO : Cost Model (Pattern Feature Extractor)




WACO
Cost Model
(Performance Predictor)
3ms ?
10ms ?
2. Return result
Search Space
SuperSchedule
Search Strategy
Choose better candidate
1. Pick candidate
from search space

WACO : Search Strategy

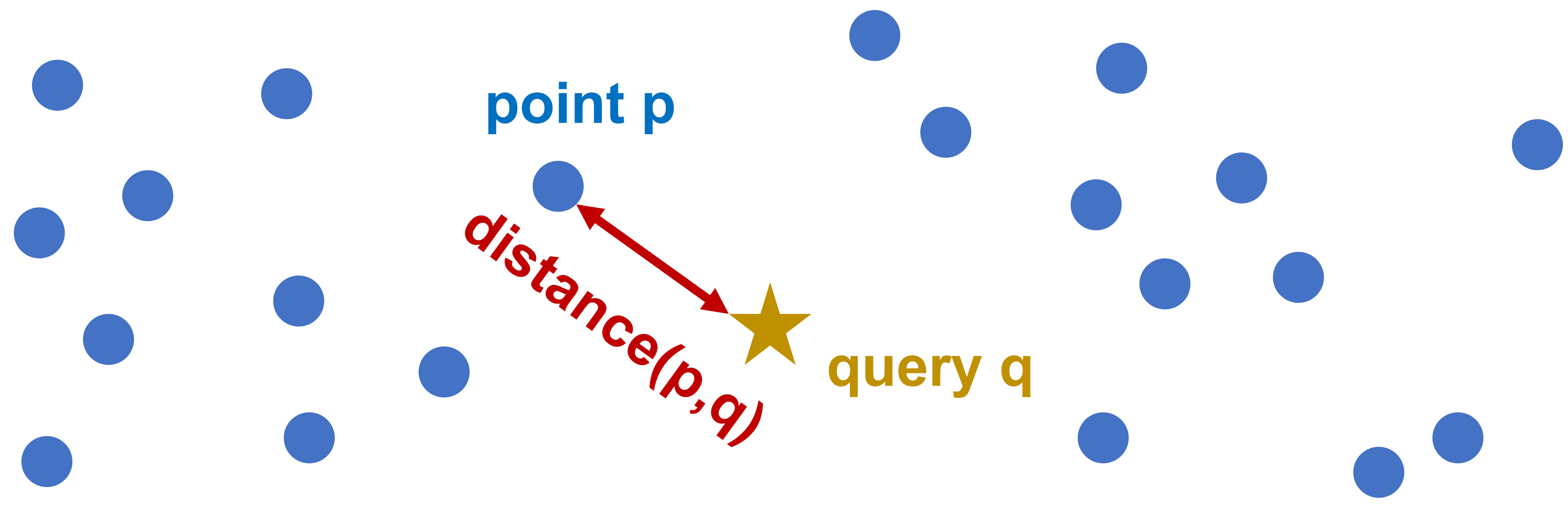


Given a **sparsity pattern**,
Search the **SuperSchedule**
that minimizes the **cost**.

WACO : Search Strategy

Nearest-Neighbor Search

Given a **query**,
Search the **point** that minimizes a **distance function**.



We viewed our problem as a nearest neighbor search.

WACO : Search Strategy

Nearest-Neighbor Search

Given a **query**,
Search the **point** that minimizes a **distance function**.

=

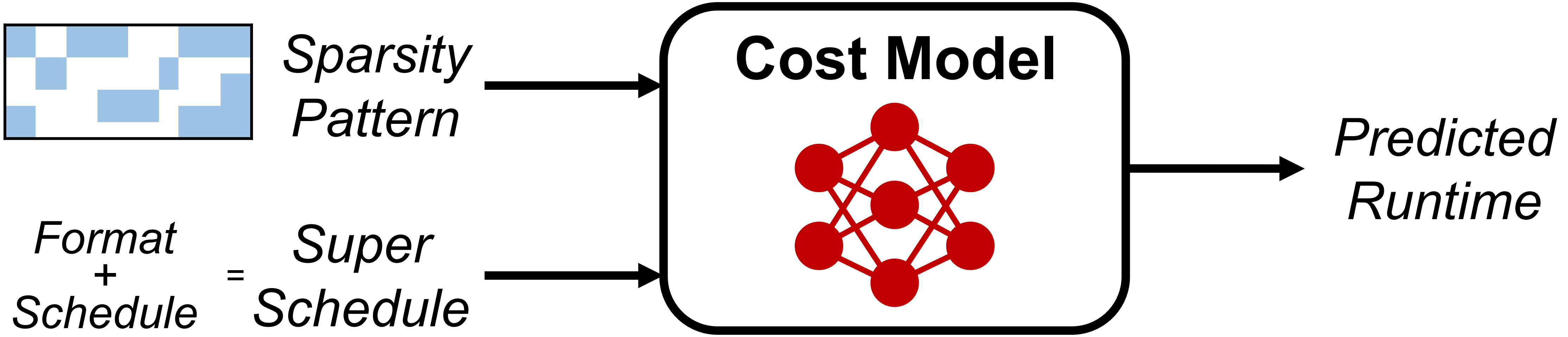
WACO Search

Given a **sparsity pattern**,
Search the **SuperSchedule** that minimizes **predicted runtime**.

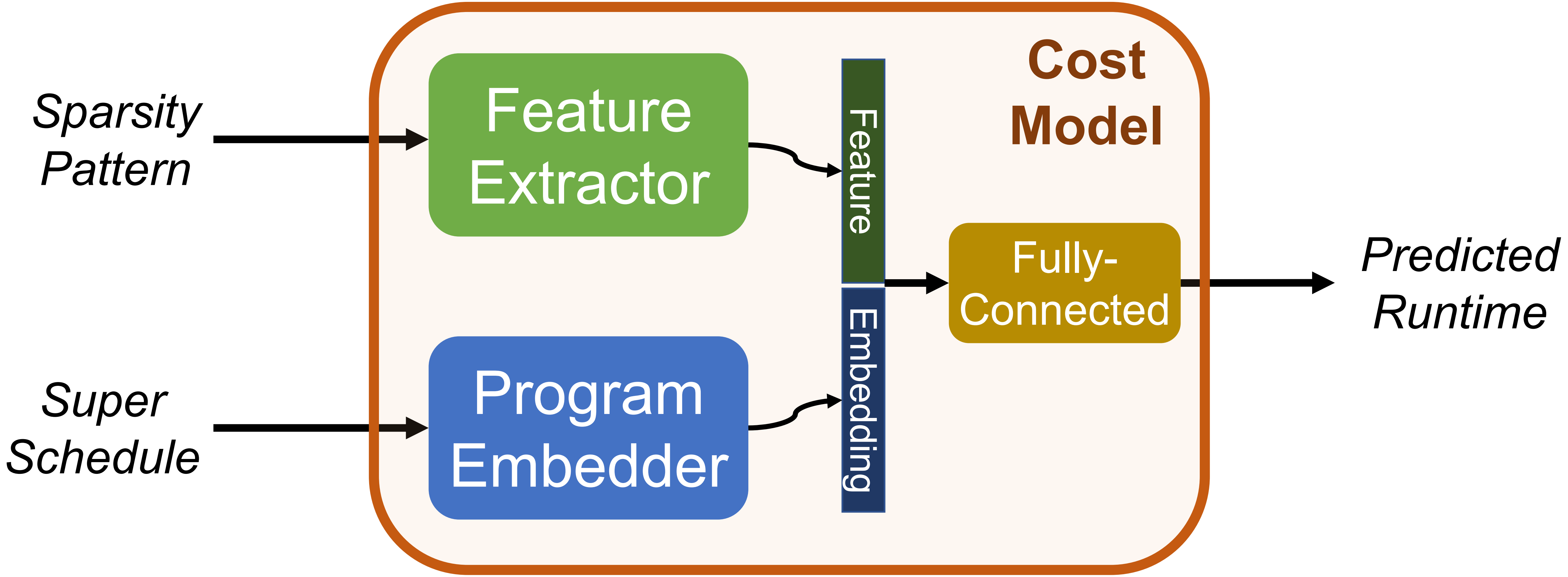
WACO is implemented with an existing NNS Library+.

Malkov, Yu A., and Dmitry A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." (2018)

Evaluation – Cost Model



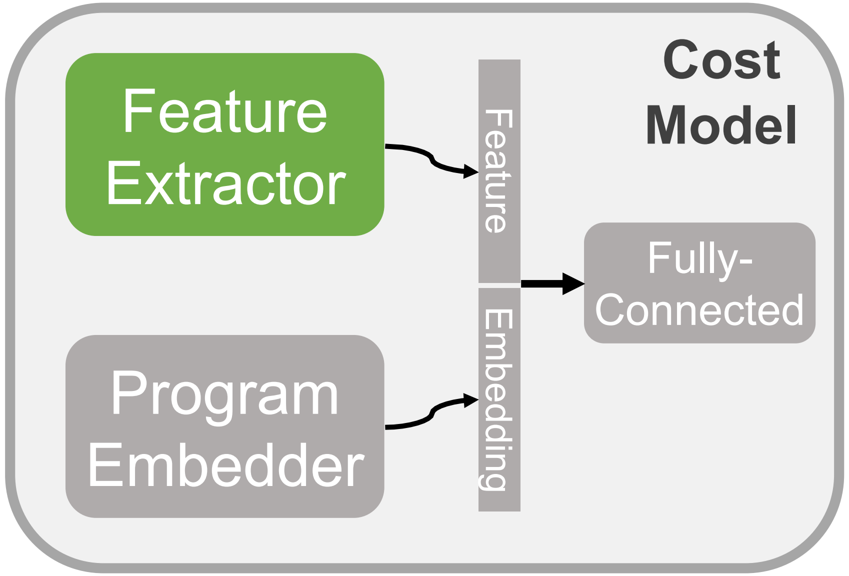
Evaluation – Cost Model



Evaluation – Cost Model

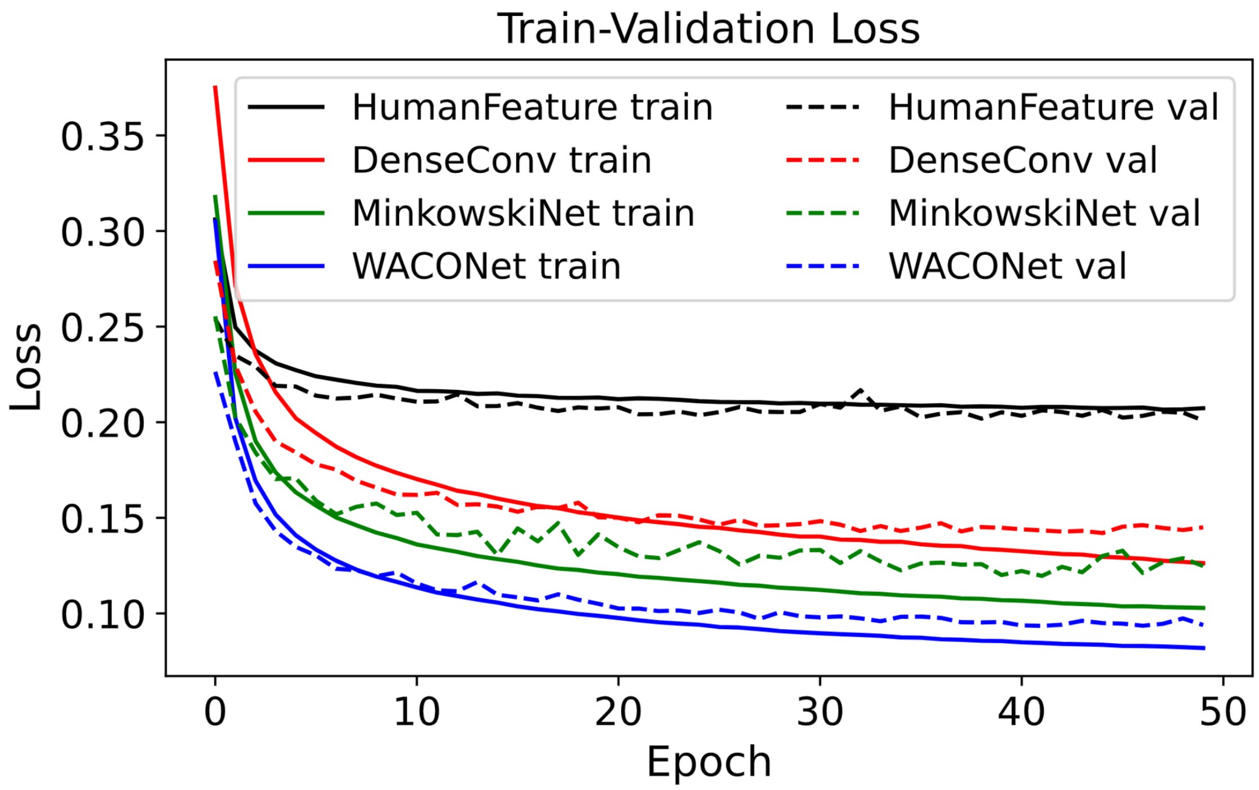
Four Feature Extractors

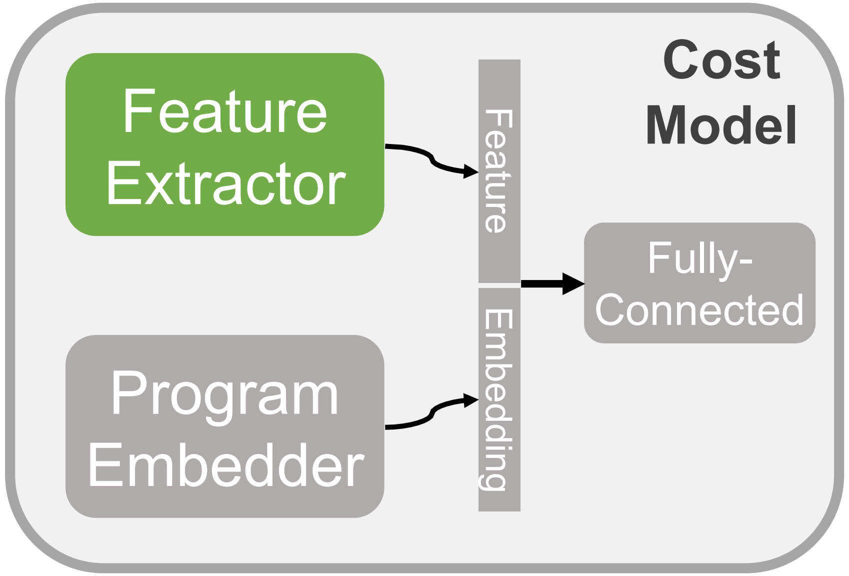
1. Hand-crafted features
2. Dense CNN after downsample
3. Sparse CNN from a computer vision
   - MinkowskiNet
4. WACONet
   - More Stride Layers



Evaluation – Cost Model

(Lower the Better)





Evaluation

- CPU: Intel Xeon E5-2680 v3
- Data: 975 Real-World Sparse Matrices

Evaluation

- CPU: Intel Xeon E5-2680 v3
- Data: 975 Real-World Sparse Matrices

	Auto-tuner		Hand-Written	
Kernels	Format-only	Schedule-only	TACO w/ Expert	ASpT
SpMV				
SpMM				
SDDMM				
MTTKRP				

Evaluation

- CPU: Intel Xeon E5-2680 v3
- Data: 975 Real-World Sparse Matrices

	Auto-tuner		Hand-Written	
Kernels	Format-only	Schedule-only	TACO w/ Expert	ASpT
SpMV	1.43x	2.32x	1.54x	-
SpMM	1.18x	1.68x	1.26x	1.36x
SDDMM	-	-	1.29x	1.14x
MTTKRP	1.27x	-	1.35x	-

1. Outperforms **all baselines** on **all kernels** on average
2. Shows good result on **3D sparsity pattern** (MTTKRP)

WACO : Summary

1. Search space considering both format and schedule.

- Explore space with Nearest Neighbor Search.

2. WACONet with submanifold sparse convolution.

- Avoid downsampling.
- More stride layers identifies distant non-zeros.

Key takeaways

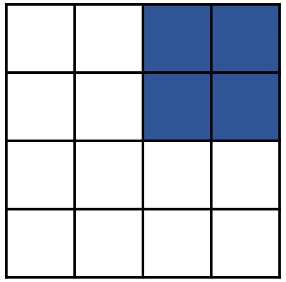
1. Auto-tuning pays the cost

- 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial

- Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!



Key takeaways

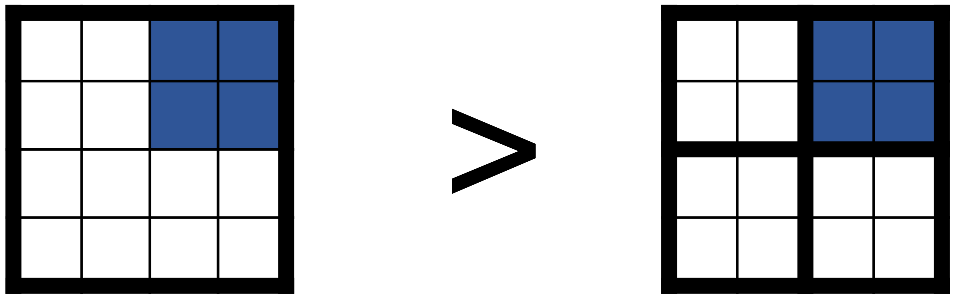
1. Auto-tuning pays the cost

- 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial

- Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!



4x4 Block (25% Fill) 2x2 Block (100% Fill)

Future Direction

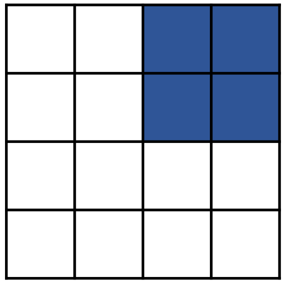
1. Auto-tuning pays the cost

- 1000(100) runs needed in SpMV(SpMM) to amortize.

2. Load-balancing is crucial

- Over 50% of matrices had improved performance with better load-balancing.

3. Increasing sparsity in dense block format can be helpful!



Thank you!