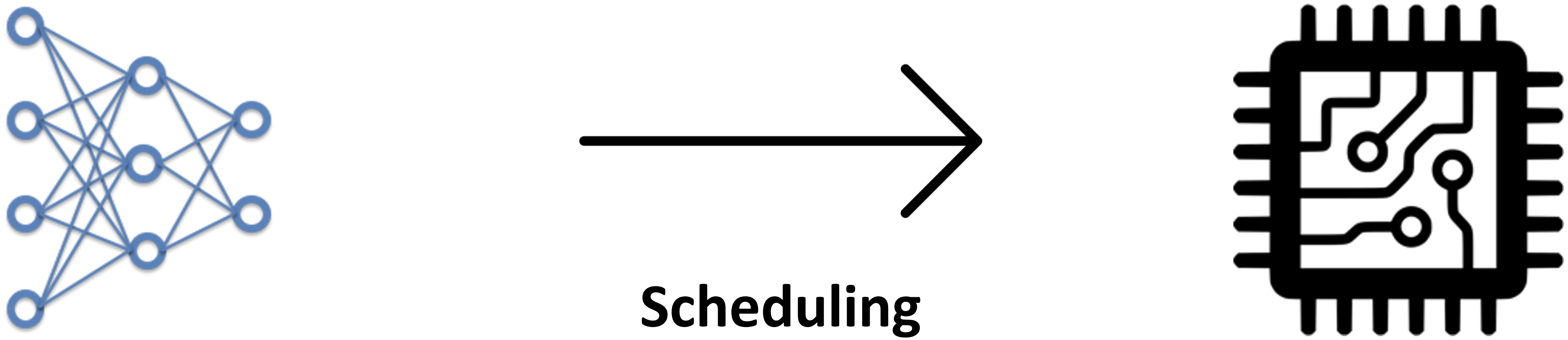
CoSA: Scheduling by Constrained Optimization for Spatial Accelerators

Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell,
Aravind Kalaiah†, James Demmel, John Wawrzynek, Yakun Sophia Shao

UC Berkeley, †Facebook

Berkeley
UNIVERSITY OF CALIFORNIA

Scheduling is required everywhere



Scheduling

- Algorithm

algorithmic states
to be run

- Hardware

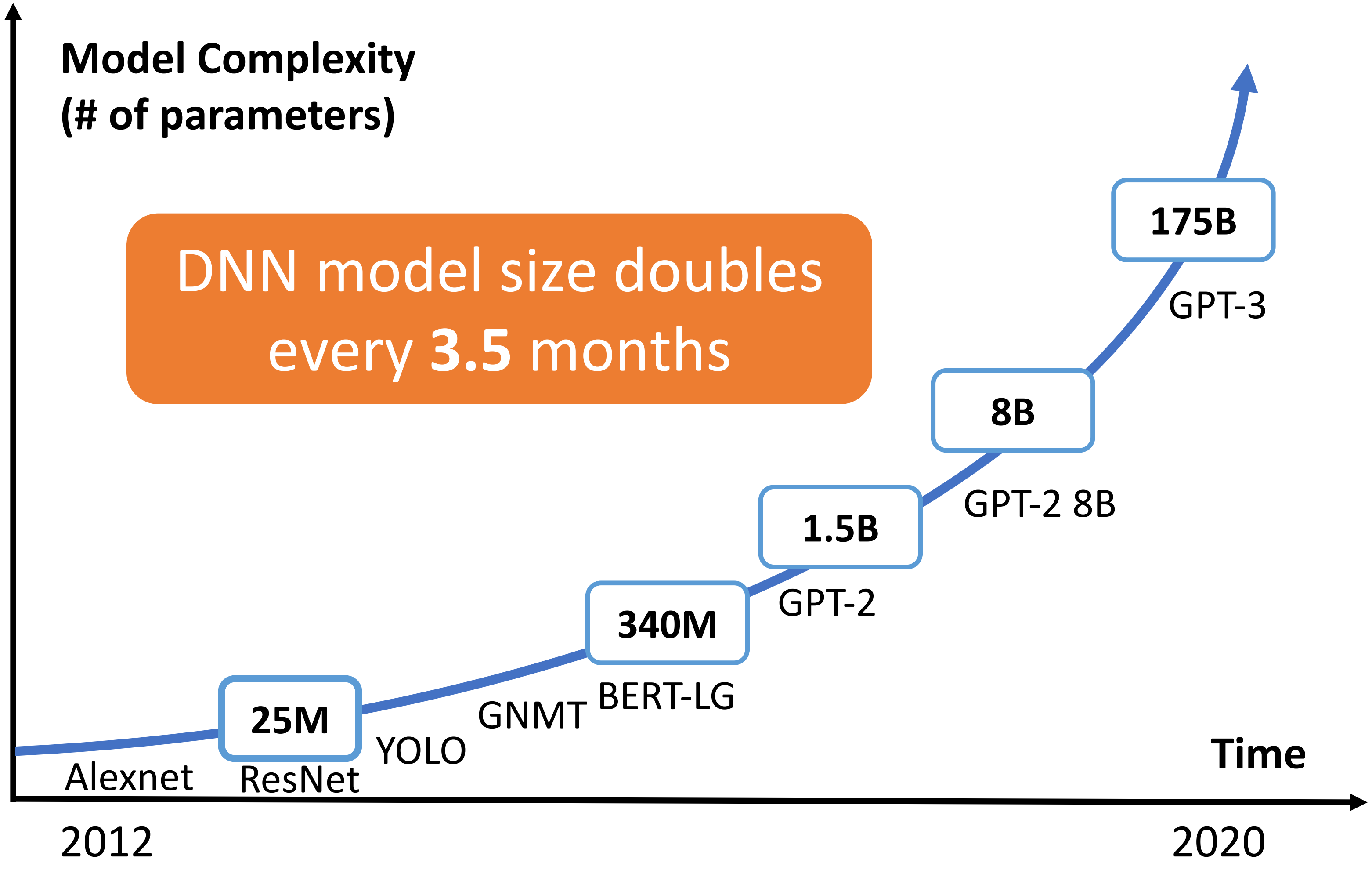
hardware resources
to be allocated

Scheduling is a big challenge

- Algorithm

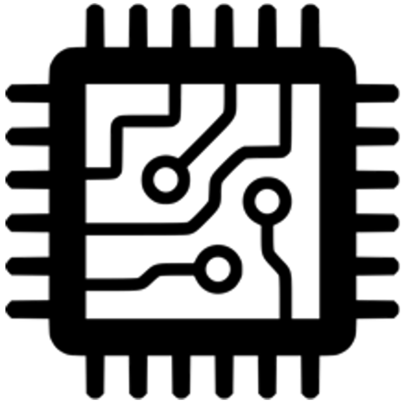
1. Exponentially growing algorithm complexity

Exponentially growing algorithm complexity



* source from Intel AI

Scheduling is a big challenge



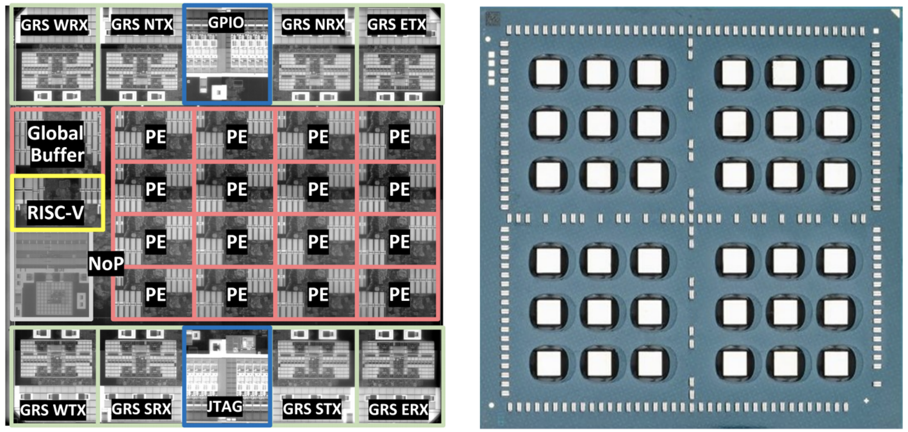
- Algorithm
- Hardware

1. Exponentially growing algorithm complexity

2. Rapidly increasing hardware capacity

Rapidly increasing hardware capacity

NoC/NoP Chip

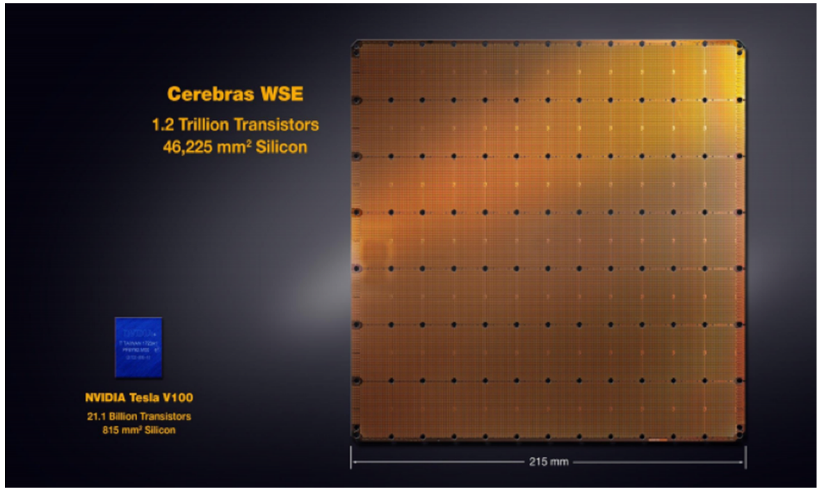


(a) Simba chiplet

(b) Simba package

Simba[1]
16PEs x 36 Chiplets

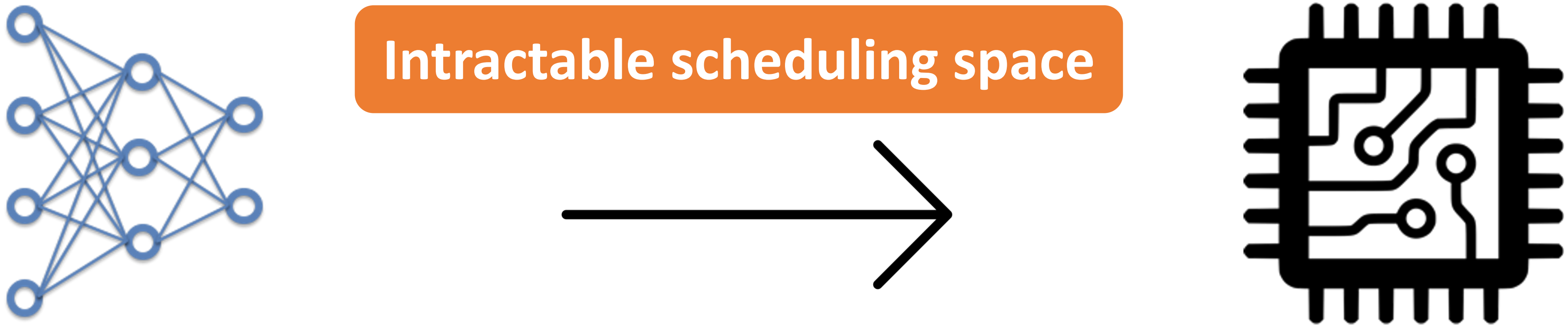
Wafer-scale Chip



Cerebras[2]
84 Interconnected Chips

[1] Shao, Yakun Sophia, and et al. "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture." 2019 MICRO.
[2] "Wafer-Scale Deep Learning", https://cerebras.net/blog/wafer-scale-deep-learning-hot-chips-2019-presentation/

Scheduling is a big challenge



Intractable scheduling space

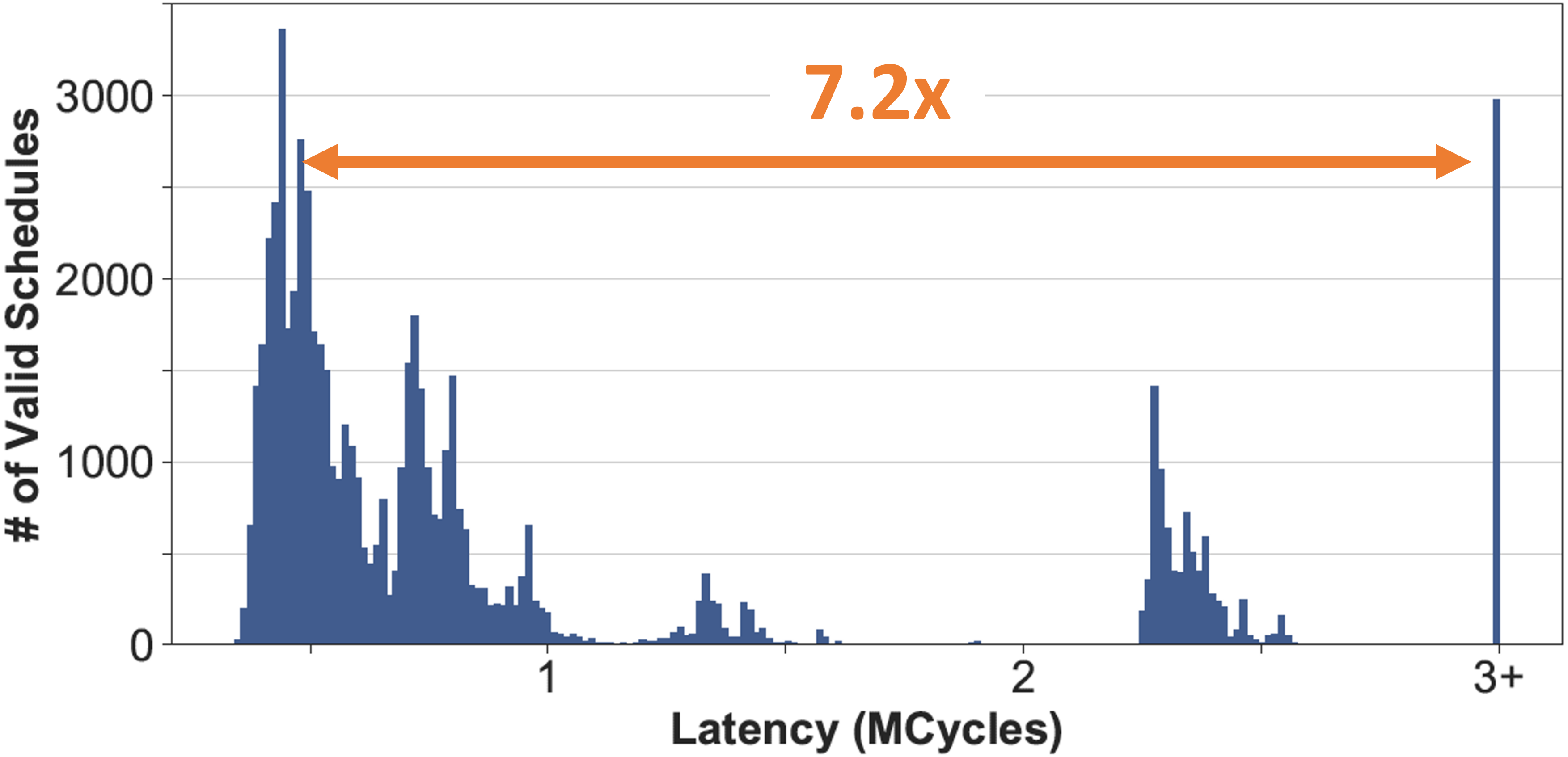
Scheduling

- Algorithm

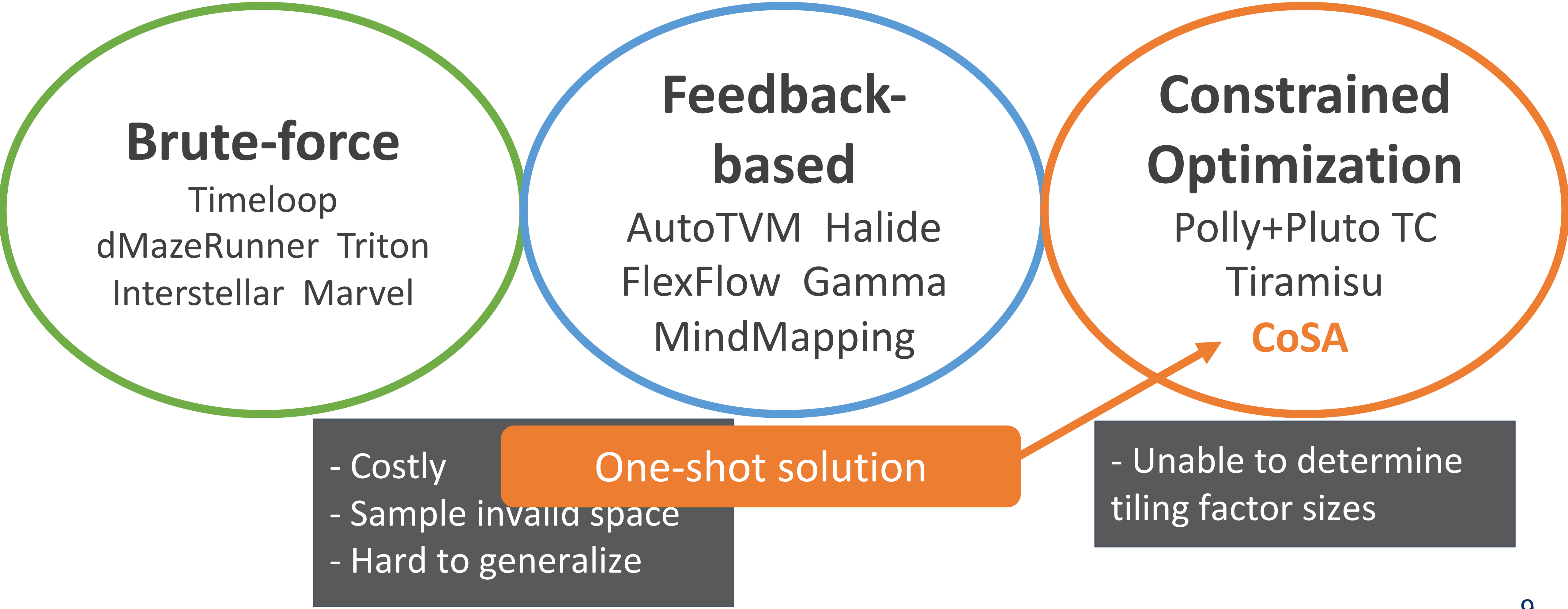
- Hardware

1. **Exponentially growing algorithm complexity**
2. **Rapidly increasing hardware capacity**

Scheduling significantly affects performance



State-of-the-art DNN accelerator schedulers



Brute-force

Timeloop

dMazeRunner Triton

Interstellar Marvel

Feedback-based

AutoTVM Halide

FlexFlow Gamma

MindMapping

Constrained Optimization

Polly+Pluto TC

Tiramisu

CoSA

- Costly
- Sample invalid space
- Hard to generalize

One-shot solution

- Unable to determine tiling factor sizes

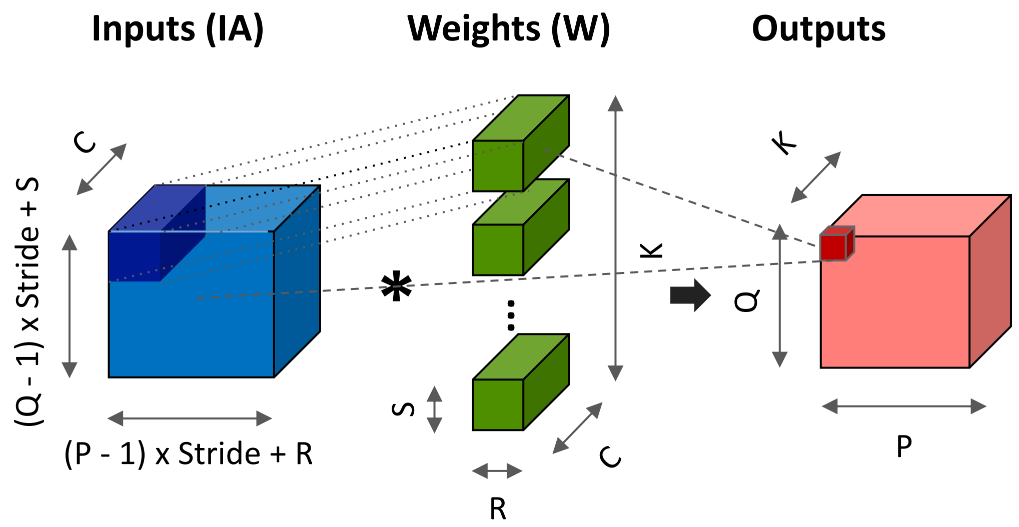
Opportunities

Workload Regularity

Hardware Regularity

Explicit Data Movement

Target Workload



R, S: weight width and height
P, Q: output width and height
C: input channel size
K: output channel size
N: batch size

DNN Layer :

```
for n in [0:N)
    for k in [0:K)
        for c in [0:C)
            for p in [0:P)
                for q in [0:Q)
                    for r in [0:R)
                        for s in [0:S)
OA[n,p,q,k] +=
    IA[n,p+r-(R-1)/2,q+s-(S-1)/2,c]
          × W[r,s,c,k]
```

Target Architecture

- Spatial PEs
- Multi-level Memory Hierarchy

64 MAC

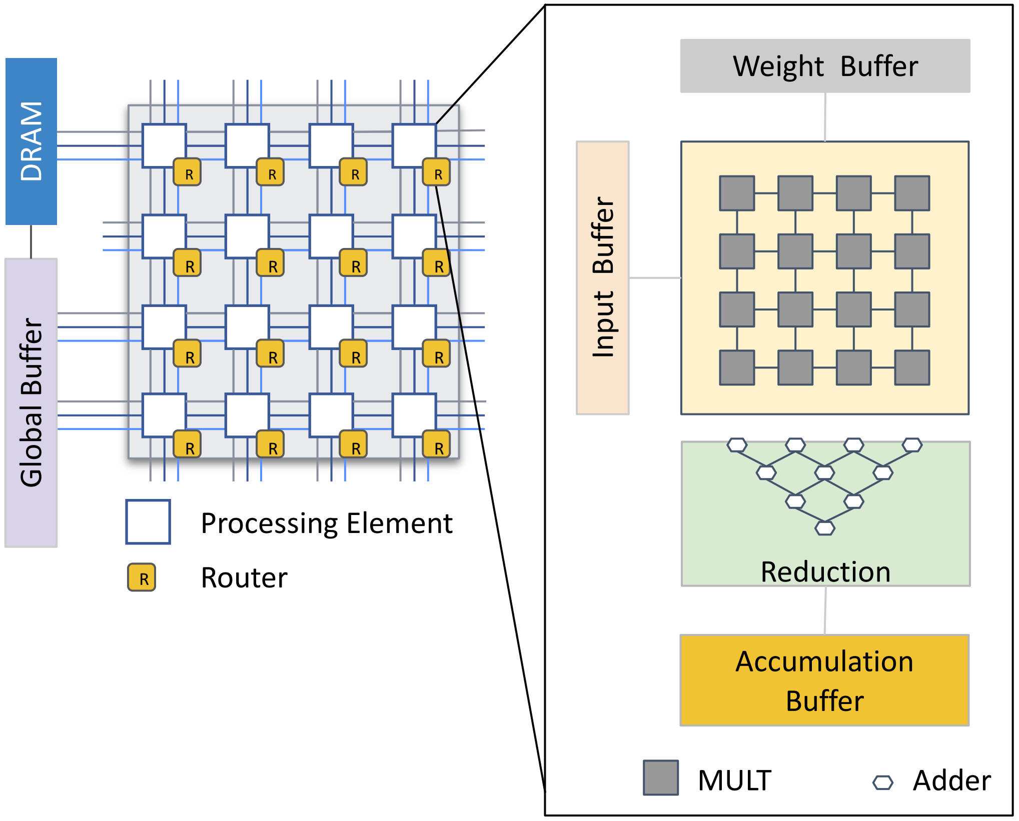
Registers
8x8
entries: 1
size: 64B

AccumBuffer
1x8
entries: 128
size: 384B

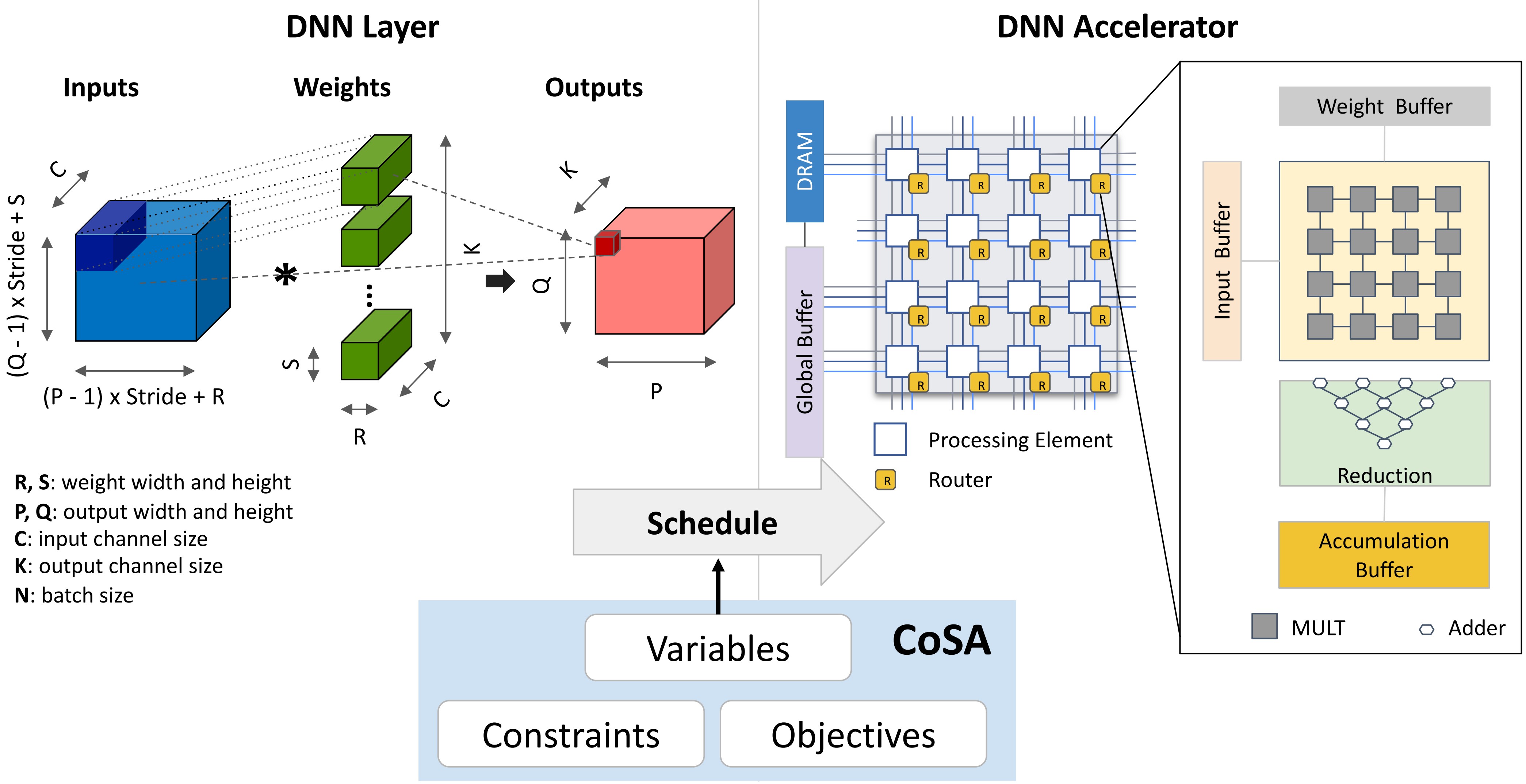
WeightBuffer
1x8
entries: 4096
size: 4KB

InputBuffer
1x1
entries: 8192
size: 8KB

DNN Accelerator



DNN scheduling problem formulation with CoSA



DNN Layer

R, S: weight width and height
P, Q: output width and height
C: input channel size
K: output channel size
N: batch size

DNN Accelerator

Schedule

CoSA

Variables

Constraints

Objectives

Three scheduling decisions

```
DRAM level
for q2 = [0 : 2) :
Global Buffer level
 for q1 = [0 : 7) :
  for n0 = [0 : 3) :
   spatial_for r0 = [0 : 3) :
    spatial_for k1 = [0 : 2) :
Input Buffer level
     for c1 = [0 : 2) :
      for p1 = [0 : 2) :
Weight Buffer level
       for p0 = [0 : 2) :
        spatial_for k0 = [0 : 2) :

      …
```

1. Tiling Factors

2. Spatial / Temporal

3. Loop Permutation

Key idea: prime factor allocation problem

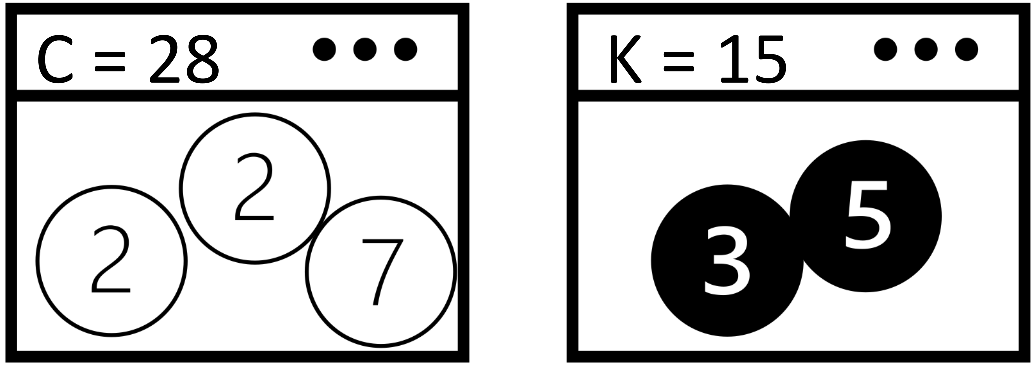
Matrix-vector mult:

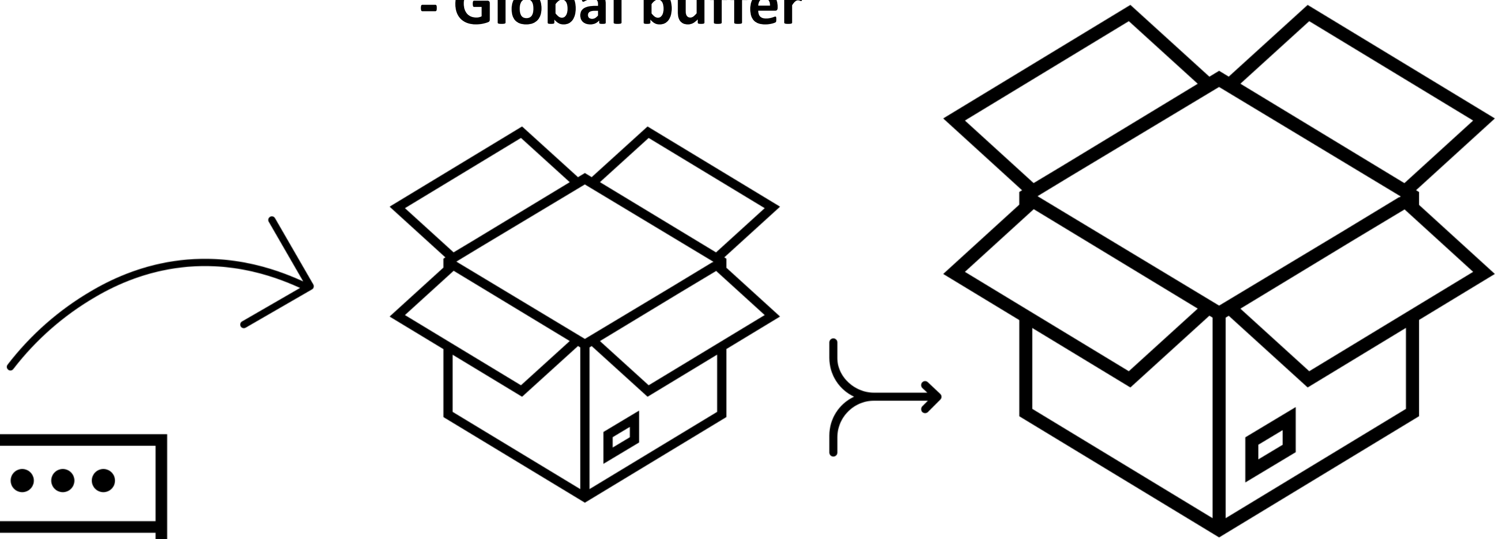
```
for c in [0:C)  // C = 28
    for k in [0:K)   // K = 15
        OA[k] += IA[c] × W[c,k]
```

Local buffers:

- **Weight buffer**
- **Global buffer**

Prime factor items:



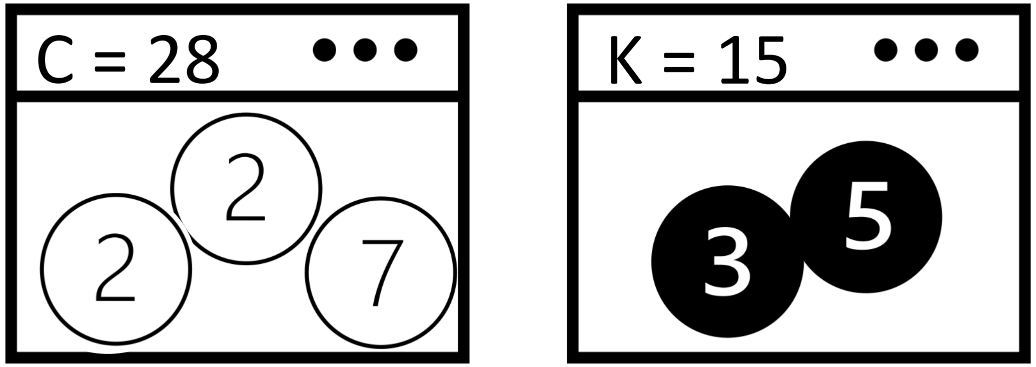


Weight Buffer
(Size = 4)

Global Buffer
(Size = 20)

CoSA Variable X – Tiling Factors

Prime factor items :



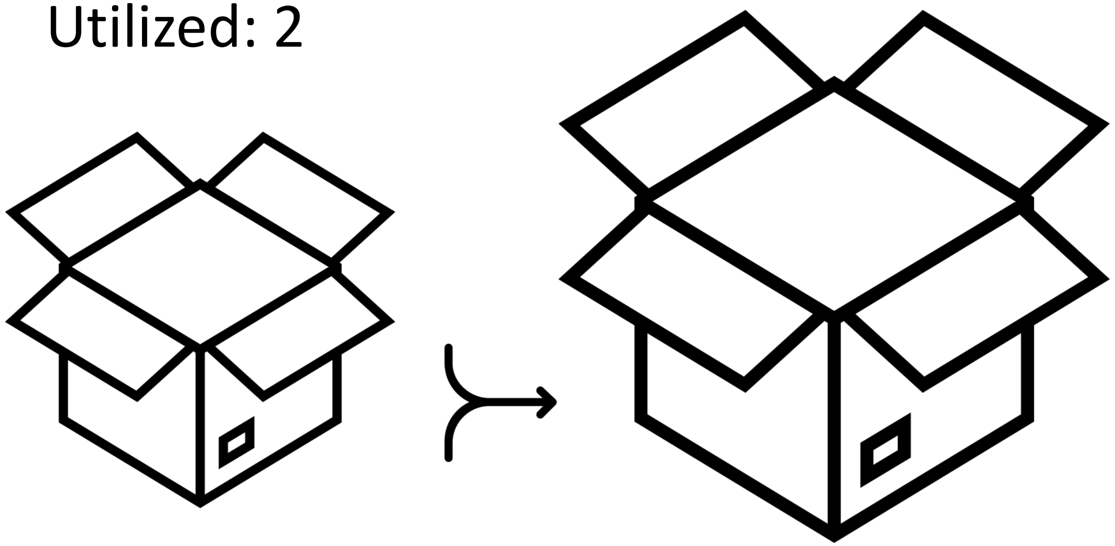
Binary allocation var X:

	C=28			K=15	
Prime Factors	2	2	7	3	5
WeightBuf	✓				
GlobalBuf		✓		✓	✓
DRAM			✓		

Local buffers:

Utilized: (2x3x5)x(2)=60

Utilized: 2

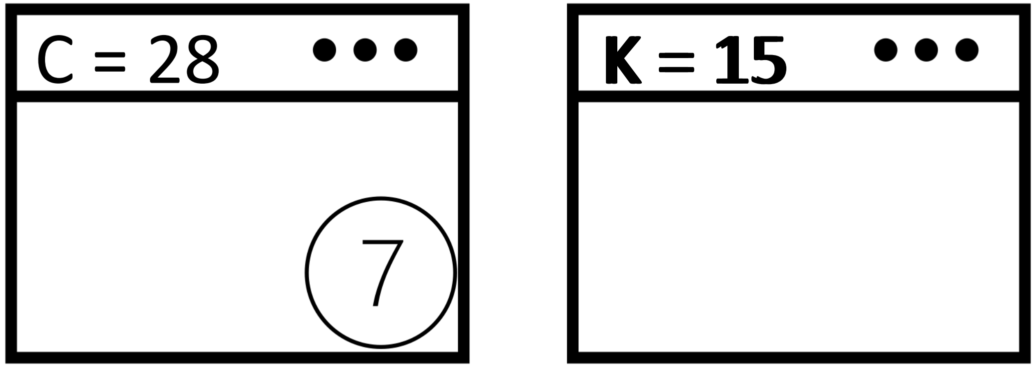


Weight Buffer (Size = 4)

Global Buffer (Size = 80)

CoSA Variable X – Spatial/Temporal Mapping

Prime factor items :



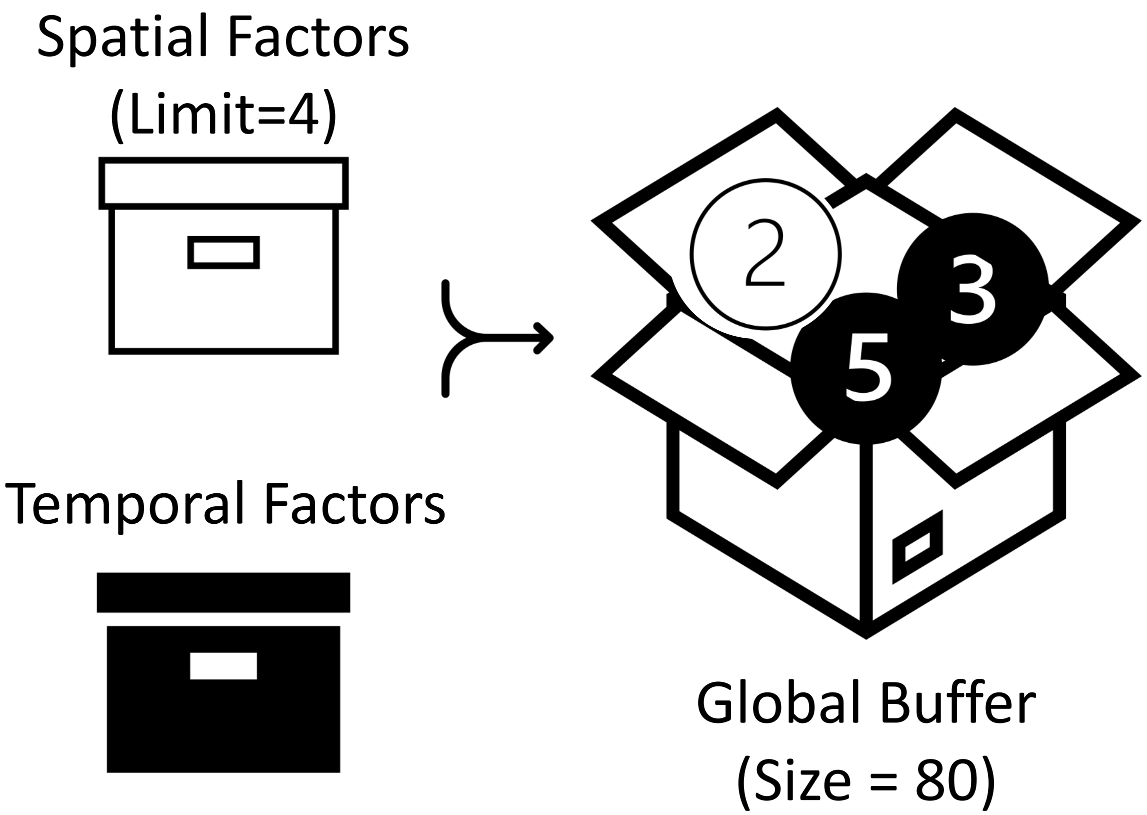
C = 28 •••

K = 15 •••

Binary allocation var X:

		C=28			K=15	
GlobalBuf	**Prime Factors**	2	2	7	3	5
	Spatial				✓	
	Temporal	✓				✓

4 PEs in the accelerator:



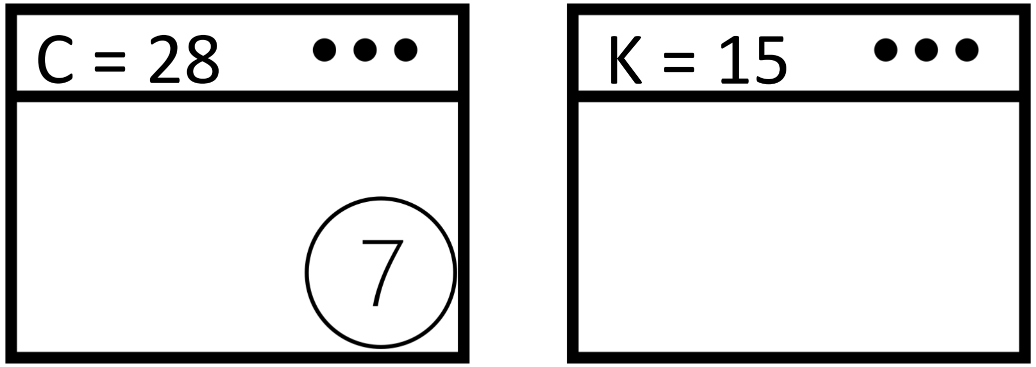
Spatial Factors (Limit=4)

Temporal Factors

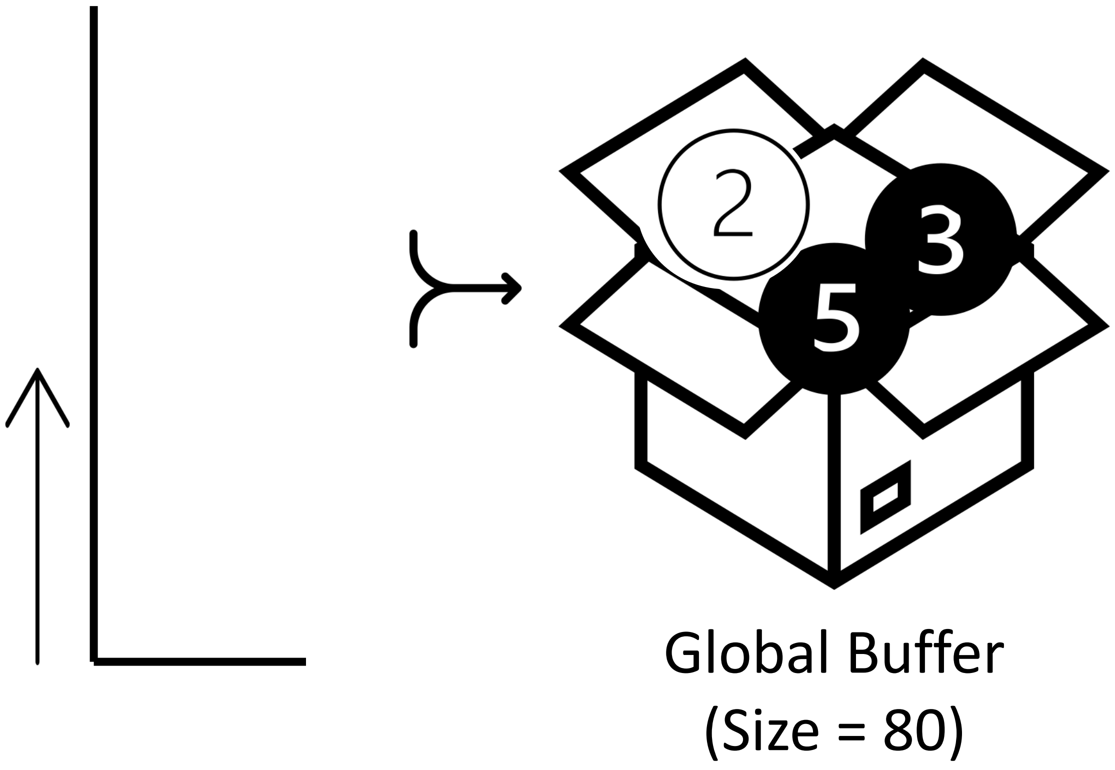
Global Buffer (Size = 80)

CoSA Variable X – Loop Permutation

Prime factor items :



Rank in global buf:



Binary allocation var X:

		C=28			K=15	
	Prime Factors	2	2	7	3	5
GlobalBuf	**rank0**	✓				
	rank1					✓
	rank2					
	rank3					
	rank4					

CoSA Variable X – Putting it altogether

	Memory	Perm	C=28			K=15	
Prime Factors			2	2	7	3	5
	WeightBuf	...	t				
	GlobalBuf	rank0		t		s	
		rank1					t
		rank2					
		rank3					
		rank4					
	DRAM	...			t		

s - Spatial, t - Temporal

DRAM level

```
for c2 = [0 : 7) :
```

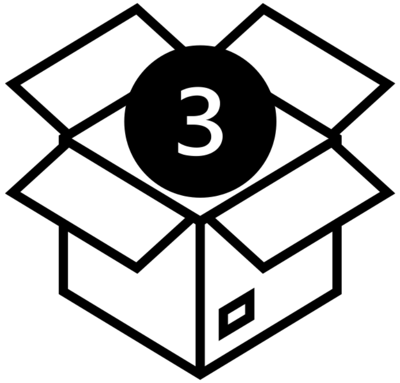
Global Buffer level

```
 for k1 = [0 : 5) :
   for c1 = [0 : 2) :
     spatial_for k0 = [0 : 3) :
```

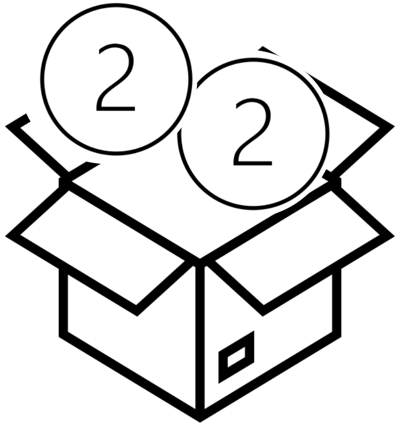
Weight Buffer level

```
       for c0 = [0 : 2) :
```

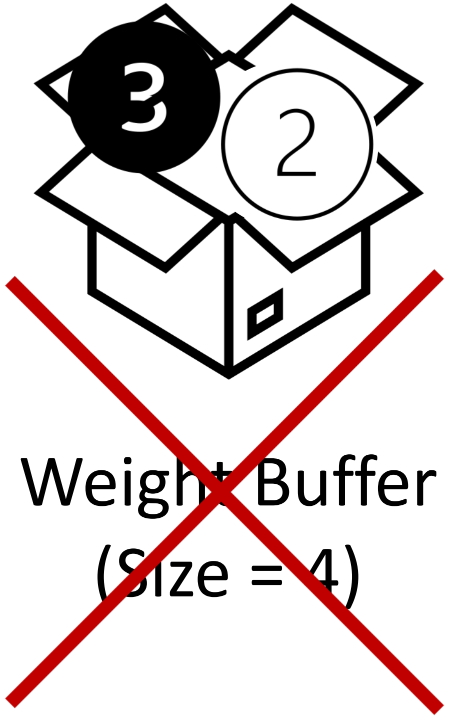
CoSA Constraints: Buffer Utilization



Weight Buffer
(Size = 4)

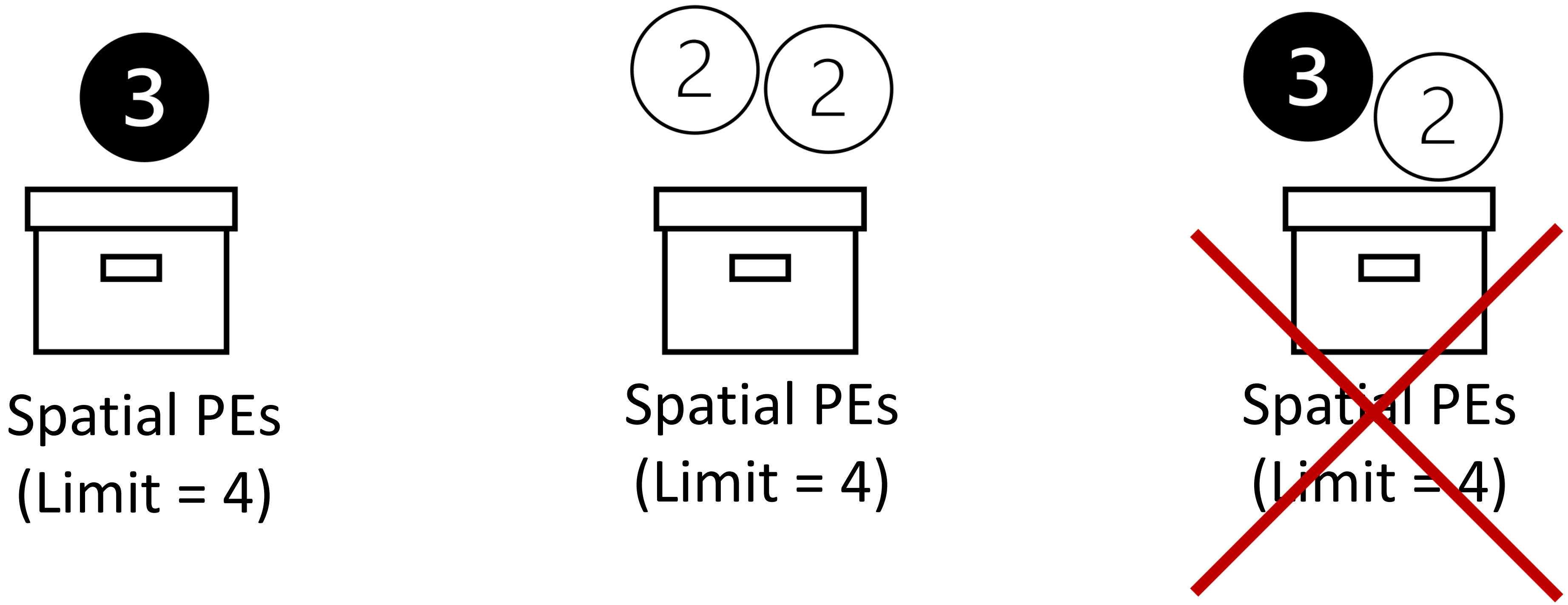


Weight Buffer
(Size = 4)



Weight Buffer
(Size = 4)

CoSA Constraints: Spatial Resources



Spatial PEs
(Limit = 4)

Spatial PEs
(Limit = 4)

Spatial PEs
(Limit = 4)

CoSA Objectives

- Utilization-driven
- Compute-driven
- Traffic-driven

CoSA Objectives

- Utilization-driven
- Compute-driven
- Traffic-driven

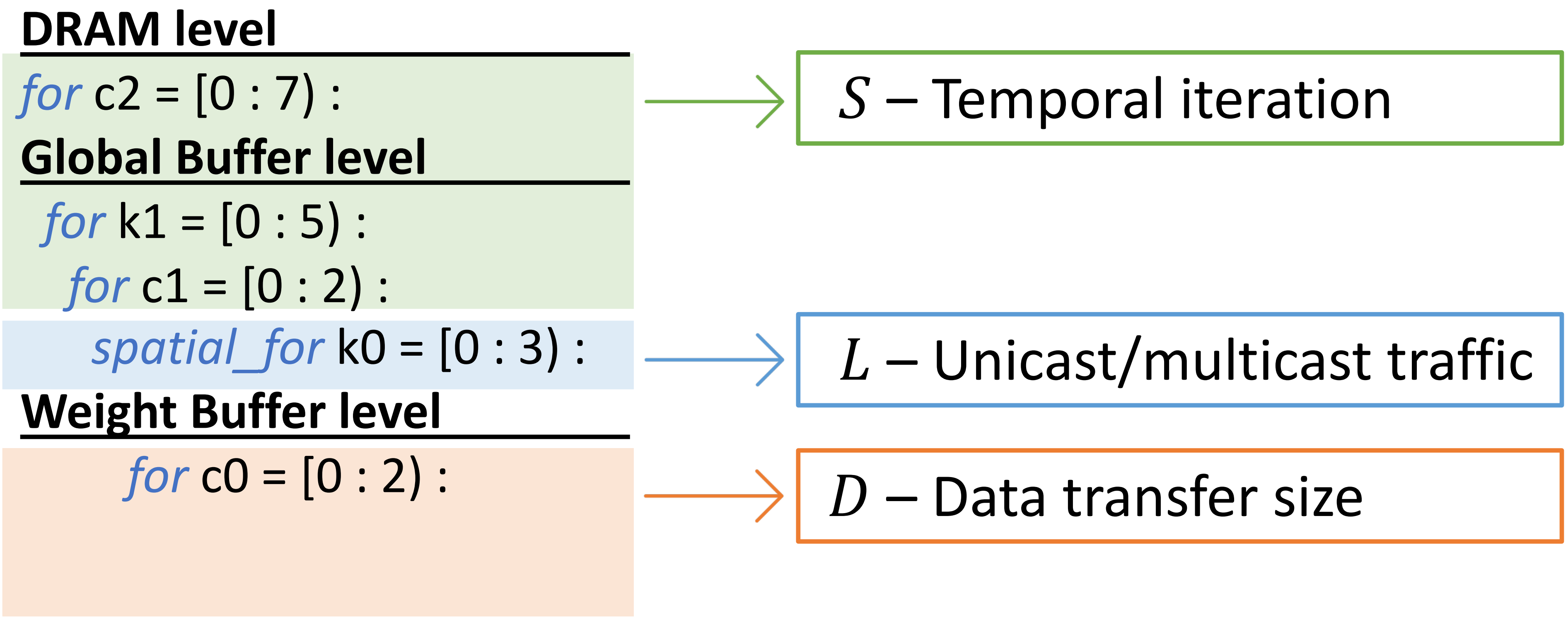
CoSA Objectives

- Utilization-driven
- Compute-driven
- Traffic-driven

CoSA Objectives

- Utilization-driven
- Compute-driven
- Traffic-driven

CoSA Traffic-driven Objective



```
DRAM level
for c2 = [0 : 7) :
Global Buffer level
  for k1 = [0 : 5) :
    for c1 = [0 : 2) :
      spatial_for k0 = [0 : 3) :
Weight Buffer level
        for c0 = [0 : 2) :
```

S – Temporal iteration

L – Unicast/multicast traffic

D – Data transfer size

Overall Traffic = $S \times L \times D$

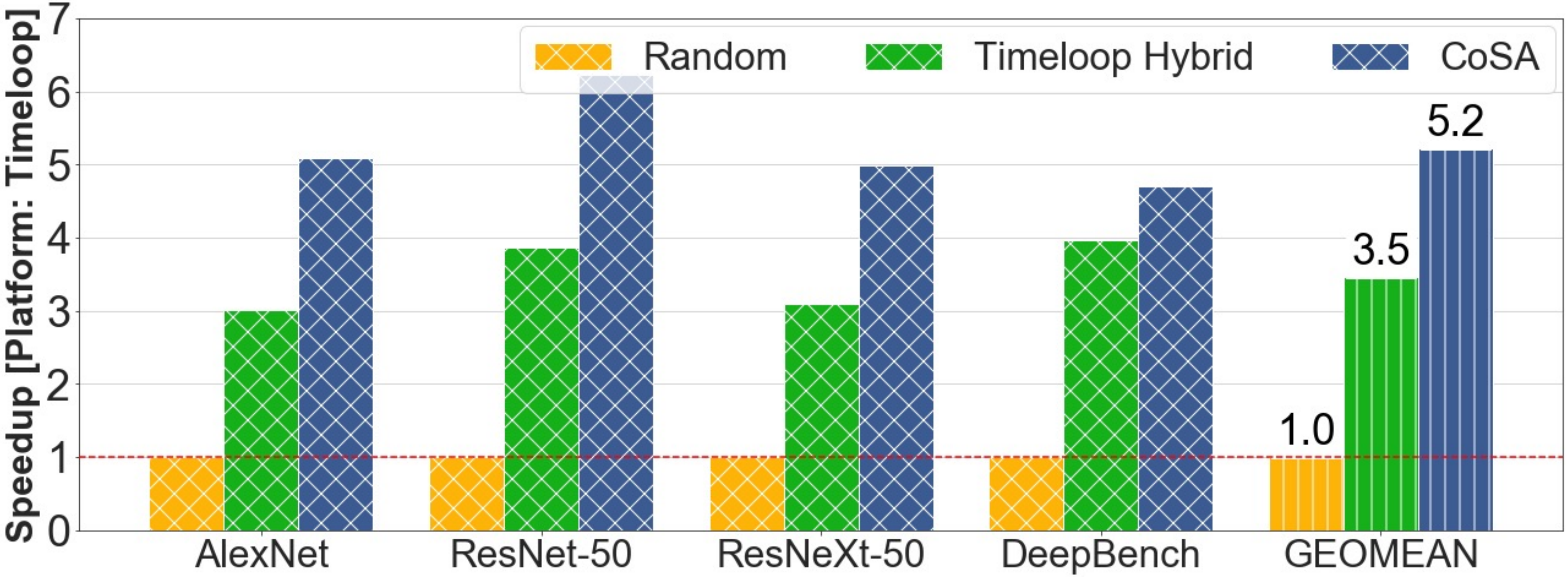
CoSA Evaluation

- Baselines:
  - Random (best out of 5 valid schedules)
  - Timeloop Hybrid (best out of 16K valid schedules)

- DNN workloads:
  - AlexNet, ResNet-50, ResNext-50, DeepBench

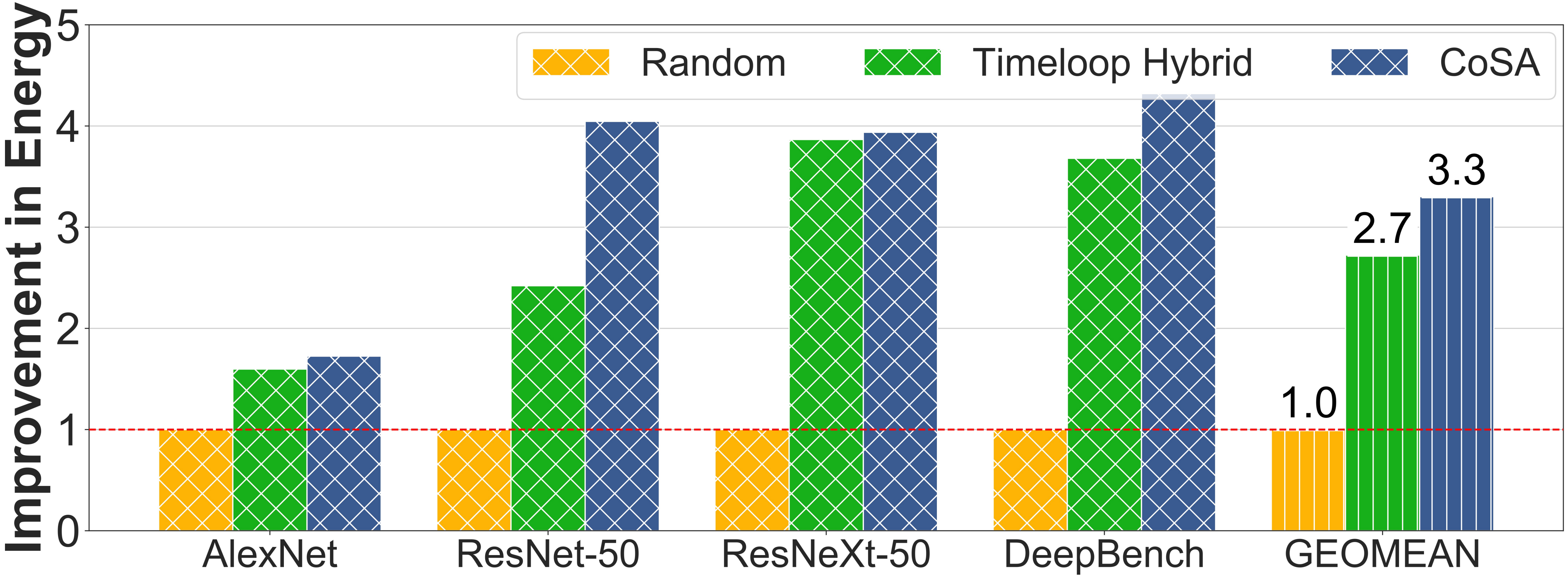
- Platforms:
  - Timeloop Simulator
  - SystemC NoC Simulator
  - GPU

1.5x latency speedup



- 5.2x better than Random
- 1.5x better than Timeloop Hybrid

1.2x better energy efficiency



- 3.3x better than Random
- 1.2x better than Timeloop Hybrid

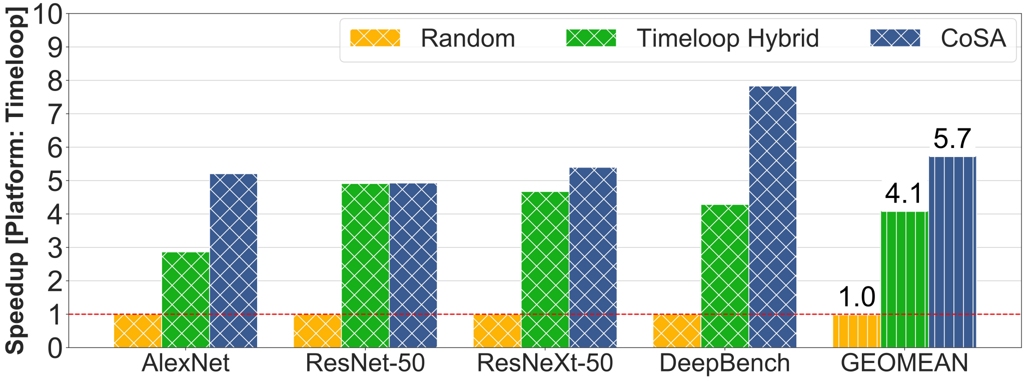
90x faster time-to-solution with CoSA

	CoSA	Random	Timeloop Hybrid
Runtime / Layer	4.2s	4.6s (1.1x)	379.9s (90.5x)
Samples / Layer	1	20K	67M
Evaluations/ Layer	1	5	16K

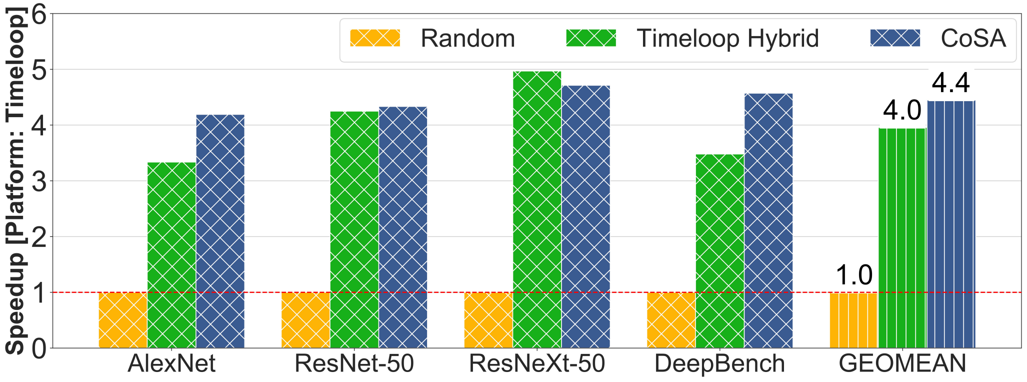
- Generates schedules within seconds
- Significantly reduces the number of samples and evaluations

CoSA generalizes to different architectures

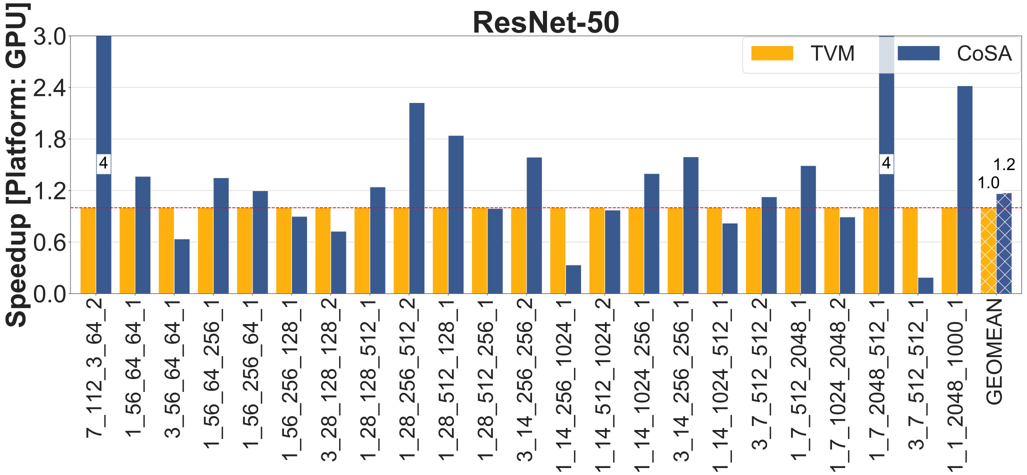
- Larger Buffers – 1.4x speedup



- 8x8 PEs – 1.1x speedup



- GPU – 1.2x speedup, 2500x faster time-to-solution over TVM (50 samples)



Conclusion

- We formulate DNN accelerator scheduling as a constrained optimization that can be solved in ***one shot***.
- We take *a* ***communication-oriented*** approach in the formulation and exposes the cost through clearly-defined objective functions.
- We demonstrate that CoSA can ***quickly*** generate ***high-performance*** schedules outperforming state-of-the-art approaches.

Github: https://github.com/ucb-bar/cosa

Questions?

qijing.huang@berkeley.edu