HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation

Qingcheng Xiao[1], Size Zheng[1], Bingzhe Wu[1], Pengcheng Xu[1], Xuehai Qian[2], and **Yun Liang**[1]

1. School of EECS, Peking University, Beijing, China
2. University of Southern California, USA

Group URL: http://ceca.pku.edu.cn/

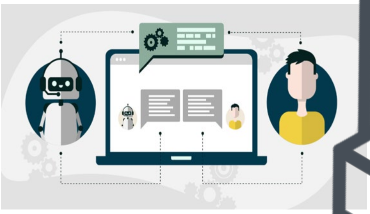
Email: {walkershaw, ericlyun}@pku.edu.cn

Tensor Computation

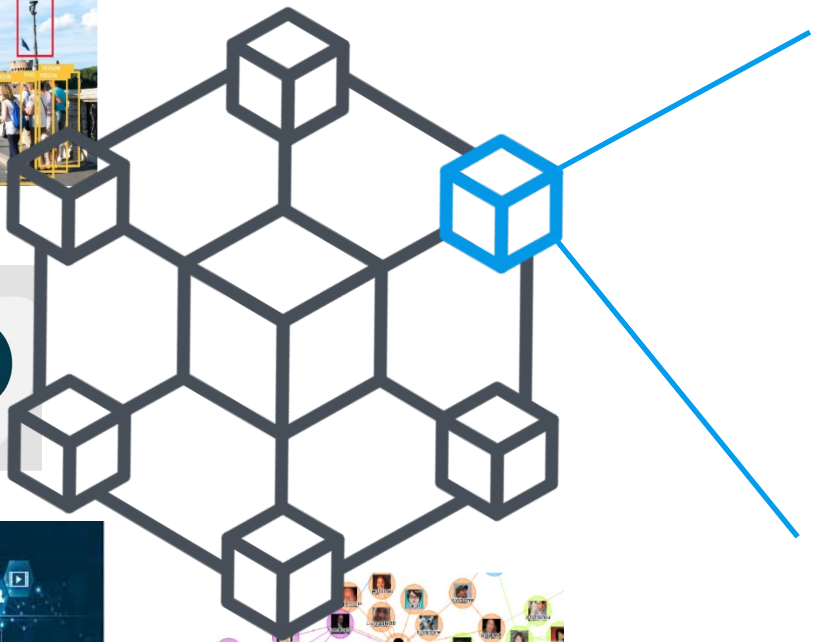
fundamental to compute-intensive applications

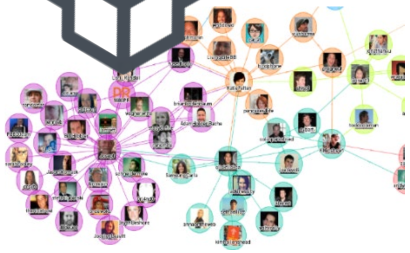
Computer Vision

NLP

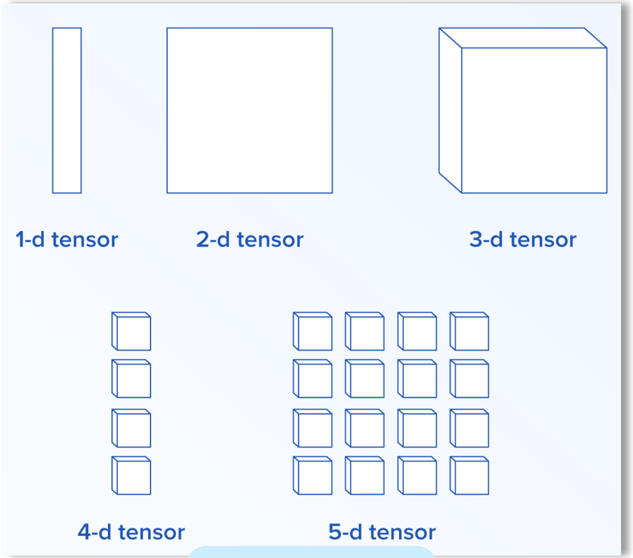


privacy & security





Social Network



Tensors

data organized in multidimensional arrays

Wide Coverage

General Matrix Multiply (GEMM): $L[i,j] = \sum M[i,k] * N[k,j]$

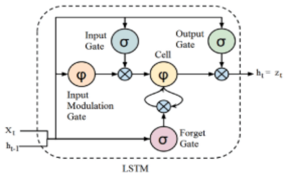
2D convolution: $C[k,x,y] = \sum A[c, x + r, y + s] * B[k,c,r,s]$

Tensor Times Matrix (TTM): $C[i,j,k] = \sum A[i,j,l] * B[l,k]$

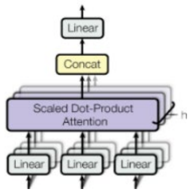
MTTKRP: $D[i,j] = \sum A[i,k,l] * B[l,j] * C[k,j]$

… …

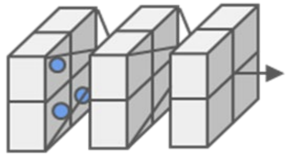
Recommender System



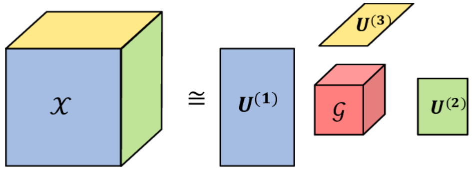
RNN



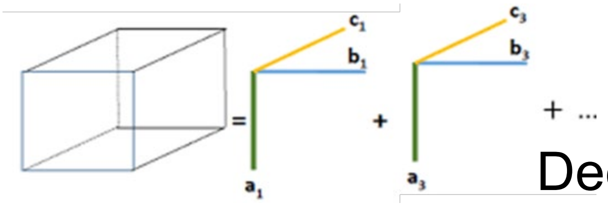
transformer



CNN



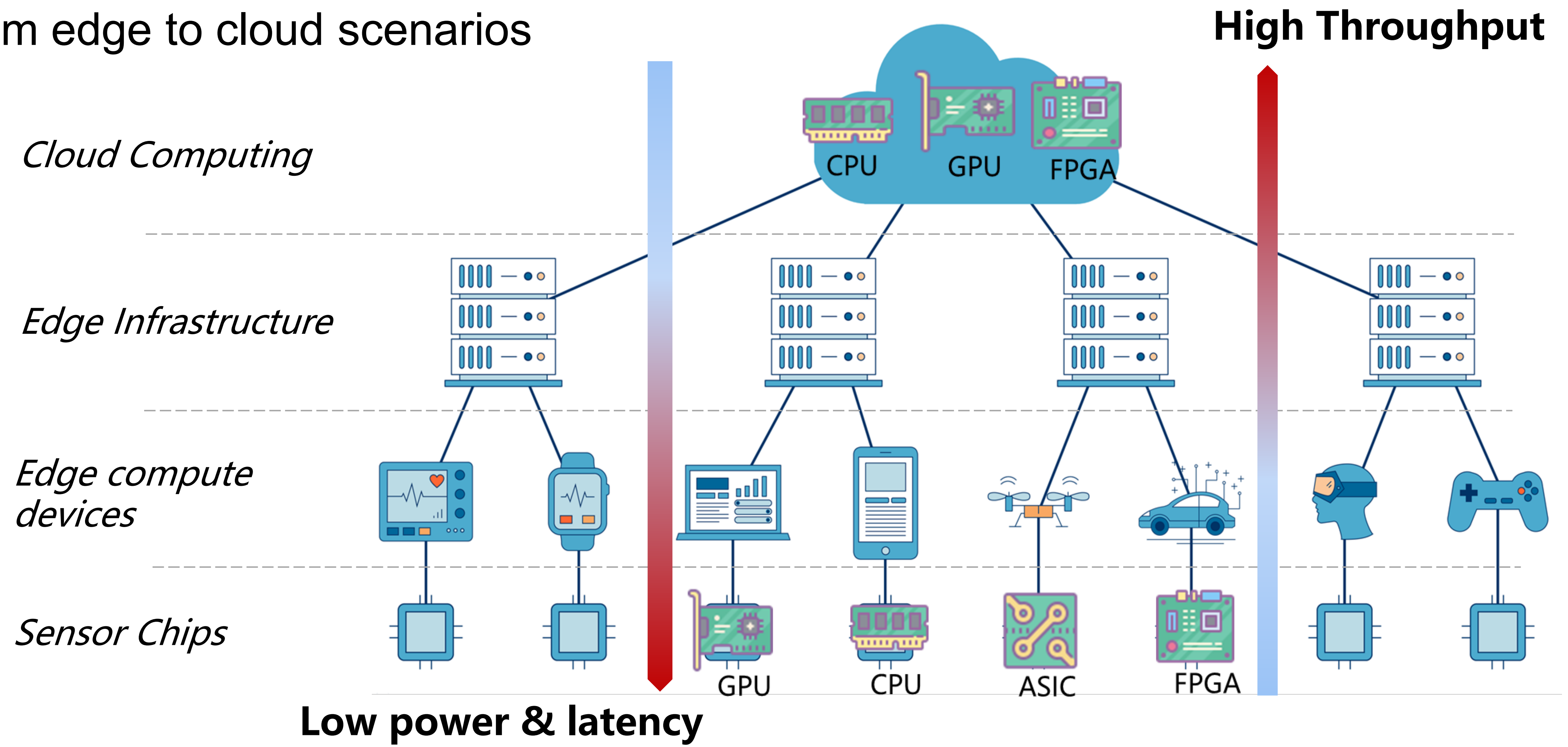
Tucker Decomposition



CP Decomposition

Various Scenarios

from edge to cloud scenarios



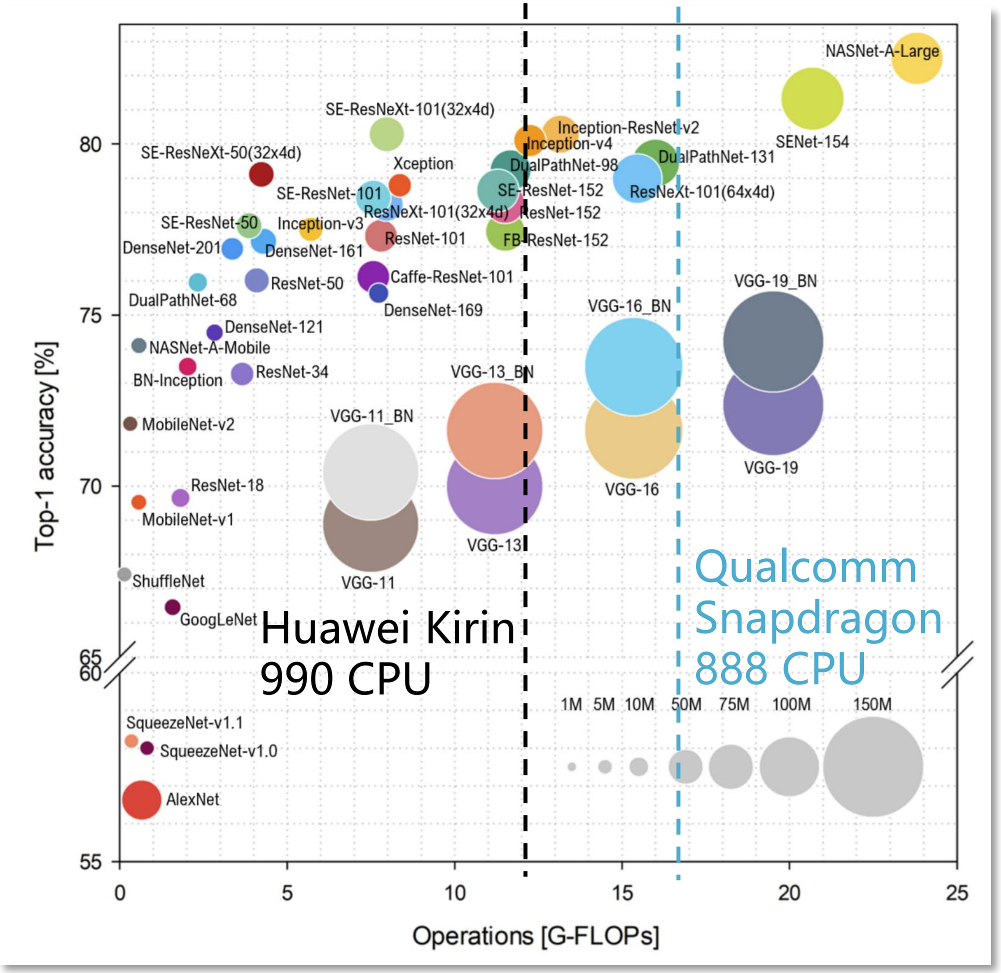
High Demand for Computing Power

tens to hundreds of 2D convolutions

⇩

the high accuracy of CNNs

No sufficient computing power



From Benchmark Analysis of Representative Deep Neural Network Architectures.

A Holistic Solution

tensor application

Workloads | Conv #1 | Conv #2 | . . . | GEMM #1 | . . .

Software Mapping | Prog. Conv #1 | Prog. Conv #2 | . . . | Prog. GEMM #1 | . . .

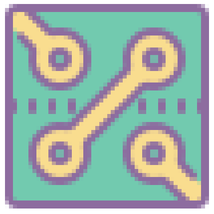
a large tensor => subtensors
& invoke the hardware execution

flexibility

A holistic solution

Hardware Acceleration

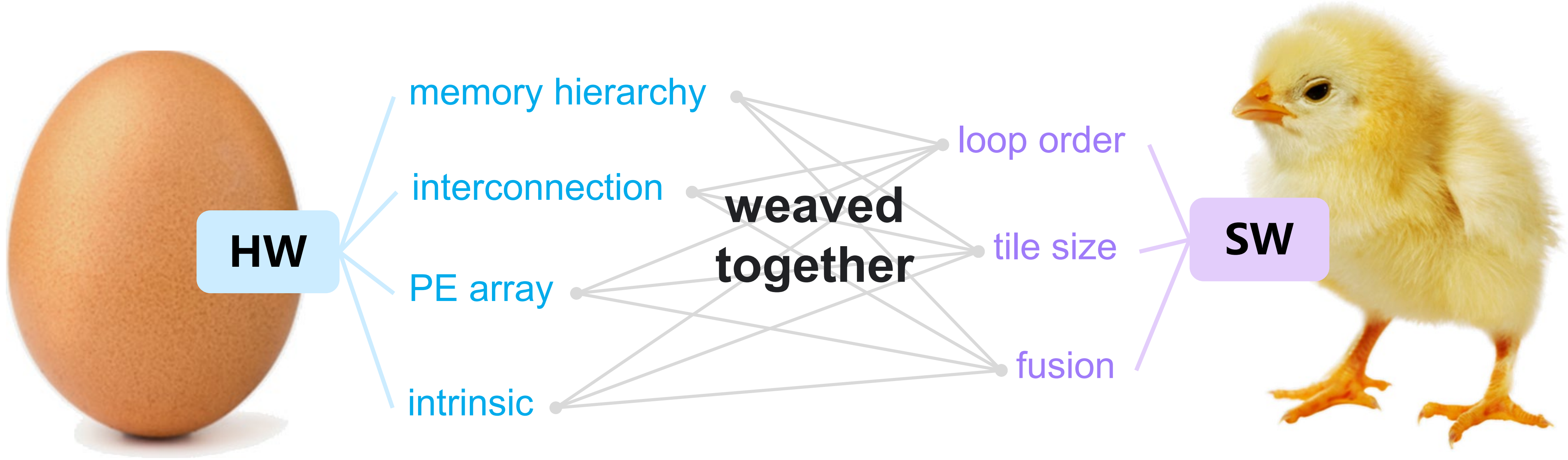
massive parallelism &
high energy efficiency



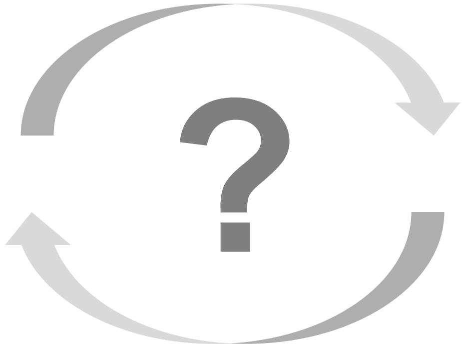
efficiency

A dedicated chip

The Chicken or the Egg?



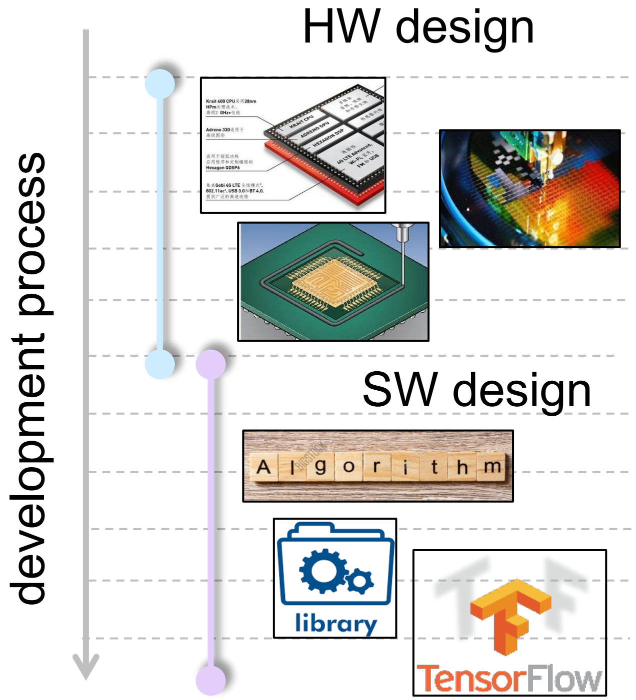
build chips without software evaluation



generate programs for a never-before-built hardware

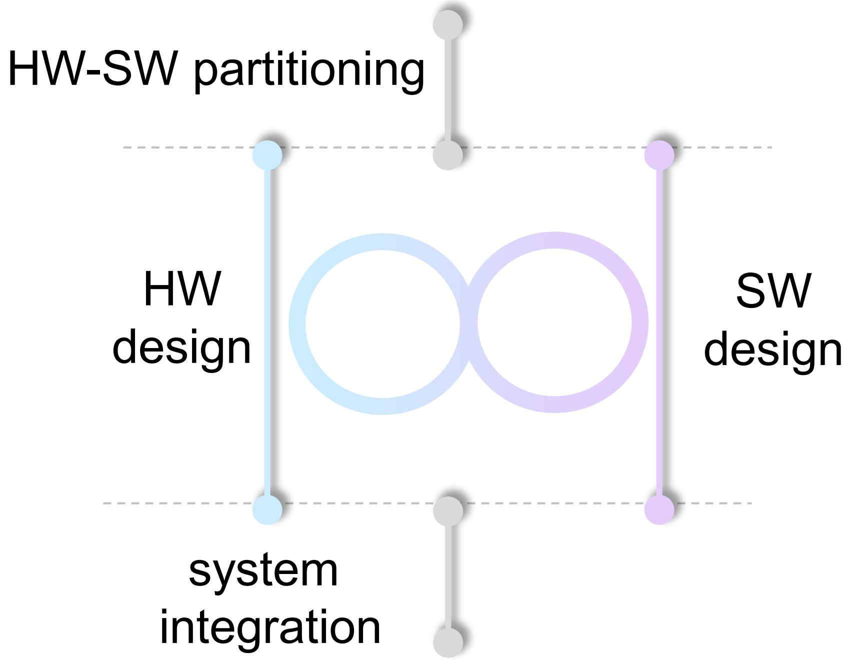
Hardware-software Co-design

Separated Design Flow



- a late system integration
- high modification costs
- poor solution qualities

Co-design Flow

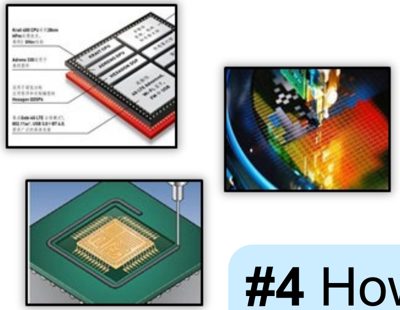


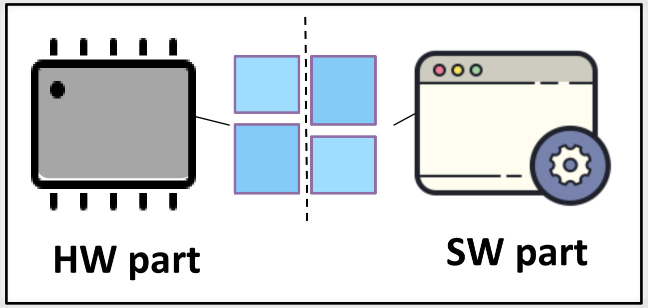
- develop in parallel
- deeply coupled optimization
- early feedbacks

Challenges in Co-design

HW-SW partitioning

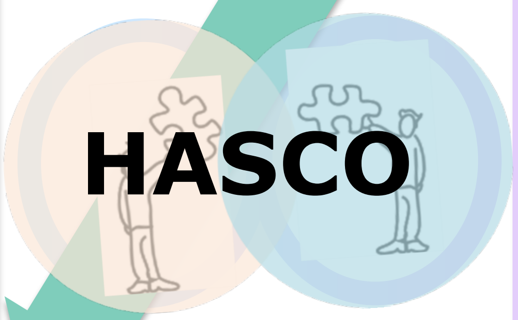
#1 How to partition hardware and software?





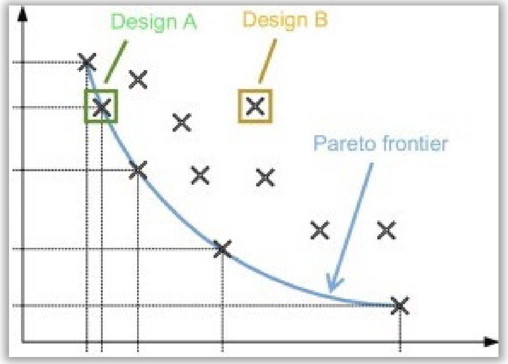
#4 How to generate chips rapidly?

HW design



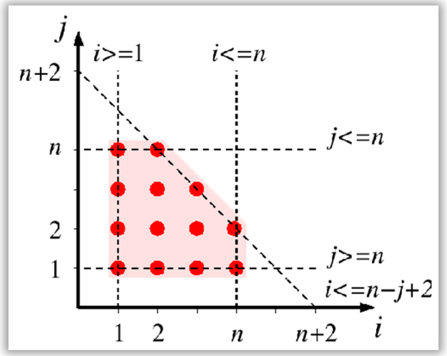
SW design

#3 How to generate efficient software mappings?

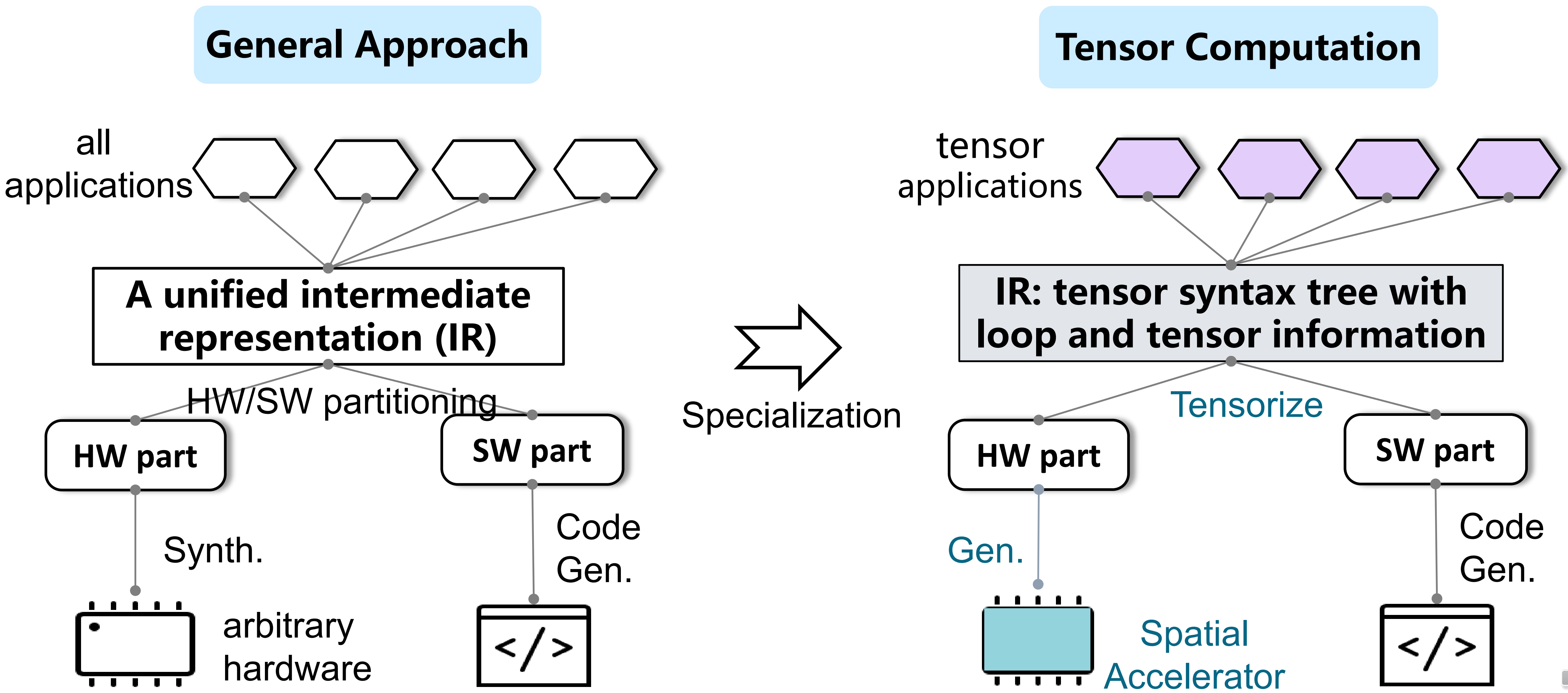


#2 How to explore design spaces efficiently?

system integration

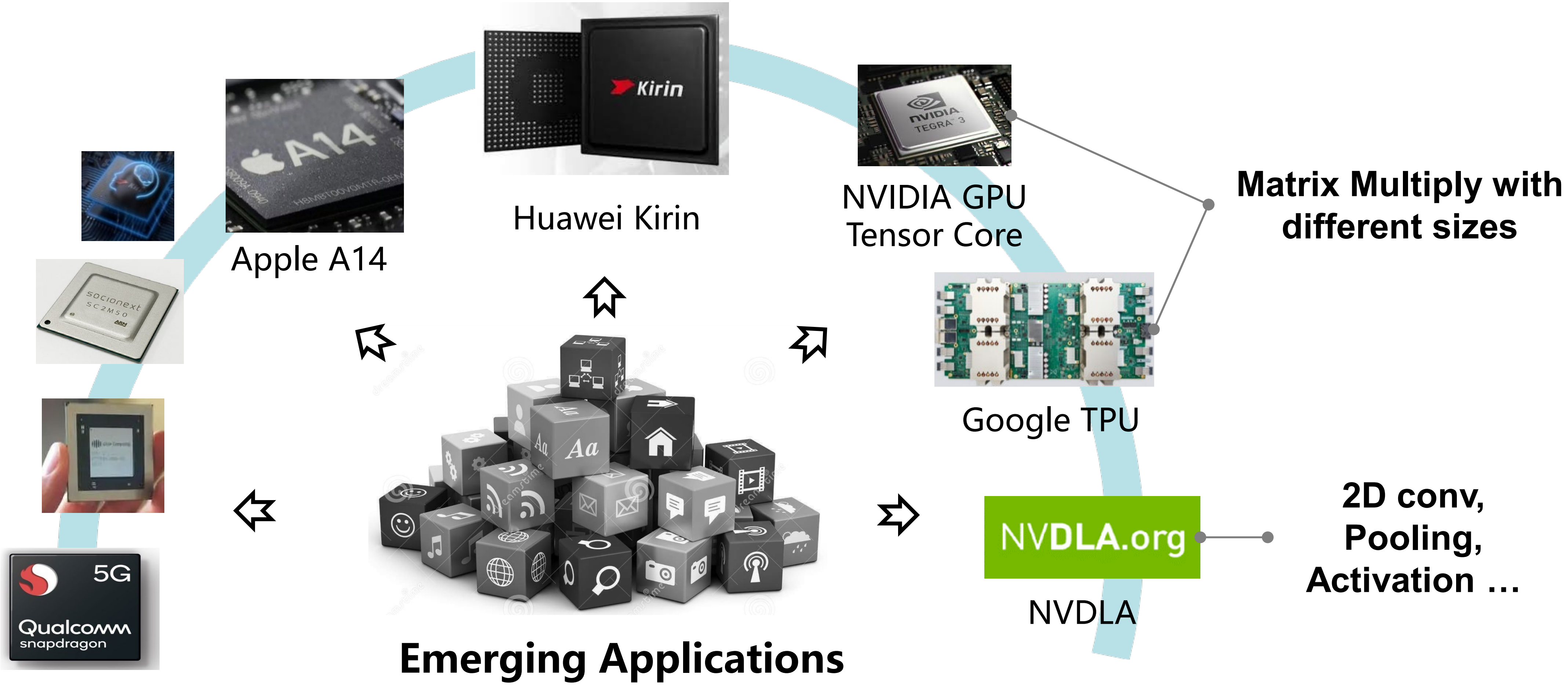


HW/SW Co-design for Tensor Computation



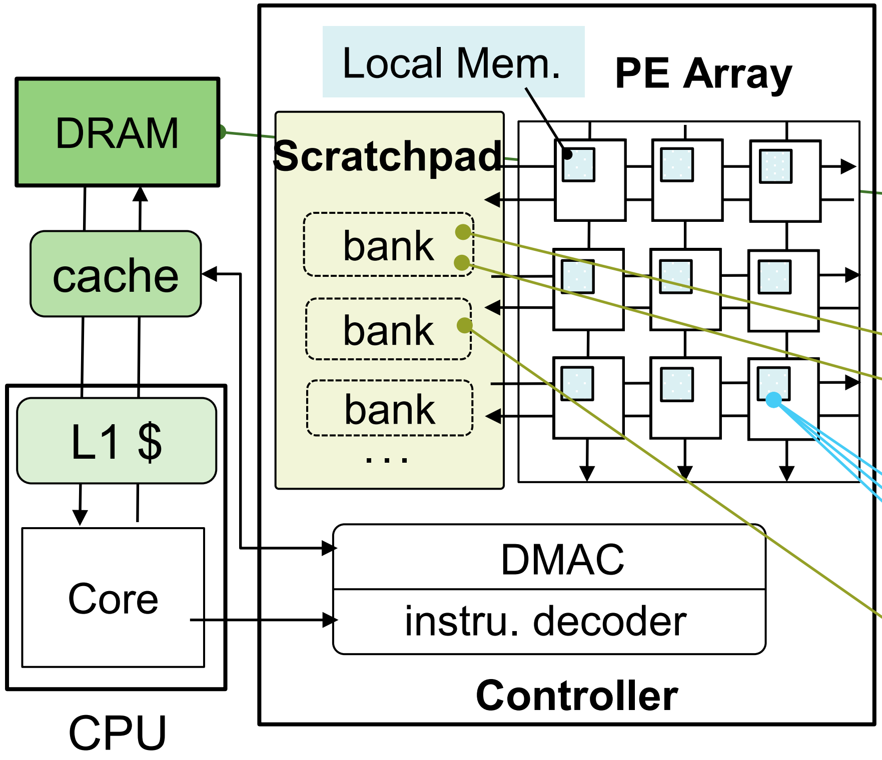
Dedicated Chips and Hardware Intrinsic

hardware intrinsic: one or a set of specific functions supported by the chips



HW-SW interface：Tensorize

Spatial Accelerator



Tensor Computation

```
def Conv_workload(A, B, C, ...):
 for y in range(0, 56):
  for r in range(0, 3):
   for s in range(0, 3):
    for k1 in range(0, 64, 32):
     for x1 in range(0, 56, 32):
      for c1 in range(0, 64, 8):
        Tensorized_GEMM(A, B, C, ...)

def Tensorized_GEMM(A, B, C, ...):
 Tensor sA, sB, sC
 sA = A[c1:c1+8, x1+r:x1+r+32, y+s]
 sB = B[k1:k1+32, c1:c1+8, r, s]
 for k2 in range(0, 32, 16):
  for x2 in range(0, 32, 16):
   for c2 in range(0, 8):
    M = sA[c2, x2:x2+16]
    N = sB[k2:k2+16, c2]
    L = GEMM_intrinsic(M, N, ...)
    sC[k2:k2+16, x2:x2+16] += L
 C[k1:k1+32, x1:x1+32, y] += sC
```

HW-SW interface：Tensorize

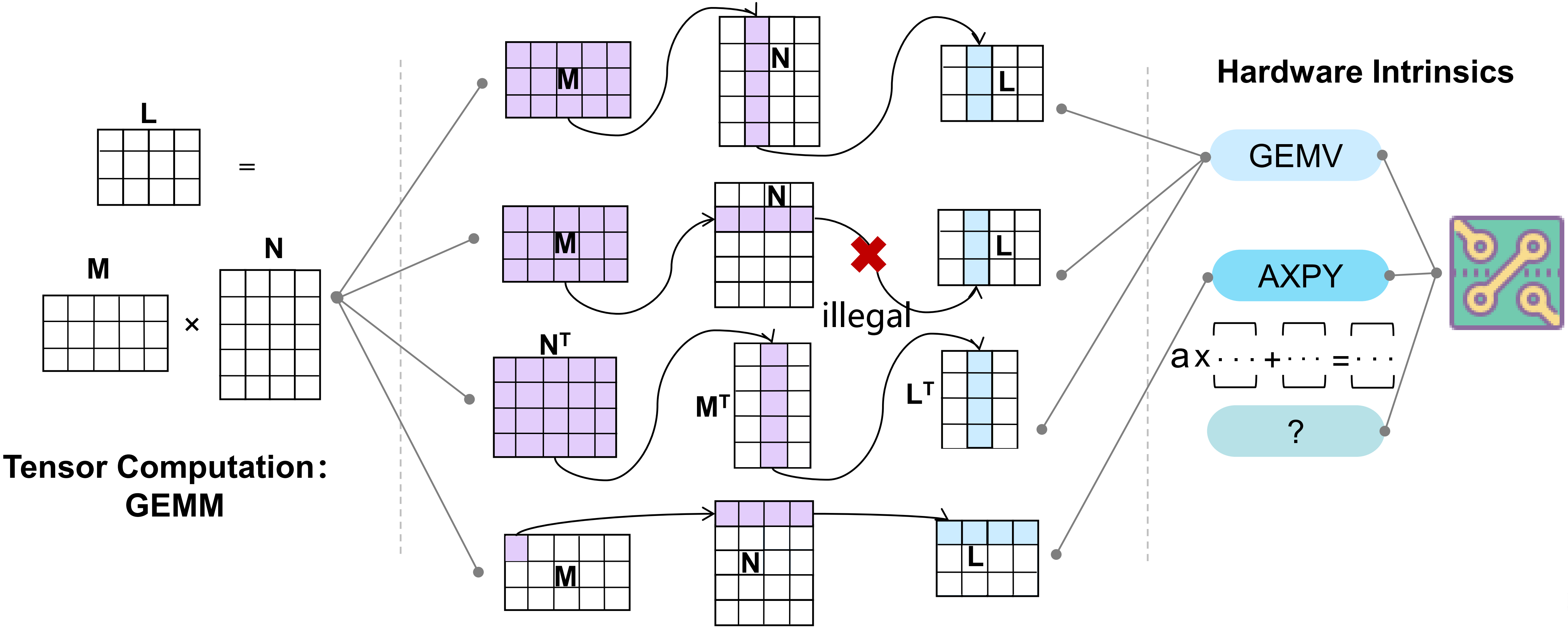
burst transfer

multi calls

Hardware Intrinsic: 16x16 GEMM

Tensorize Choices

divide & map a tensor computation onto an intrinsic



HW Design Space & SW Design Space

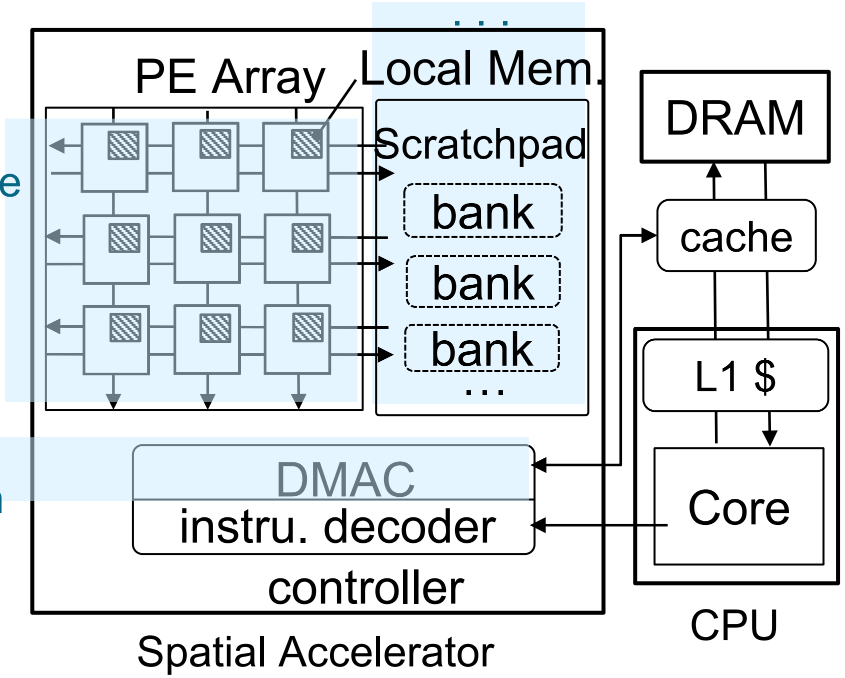
distinct metrics and costs

HW design space

- Scratchpad Size
- # Bank
- Local Mem. Size
- . . .

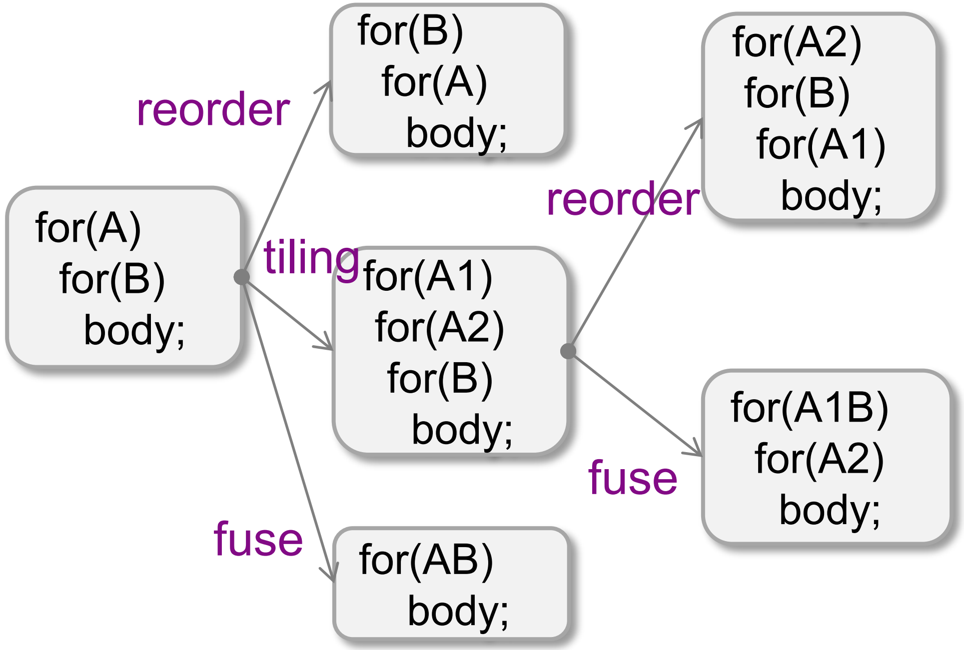
- PE Array Size
- Dataflow
- . . .

- Bandwidth
- Burst Length
- . . .



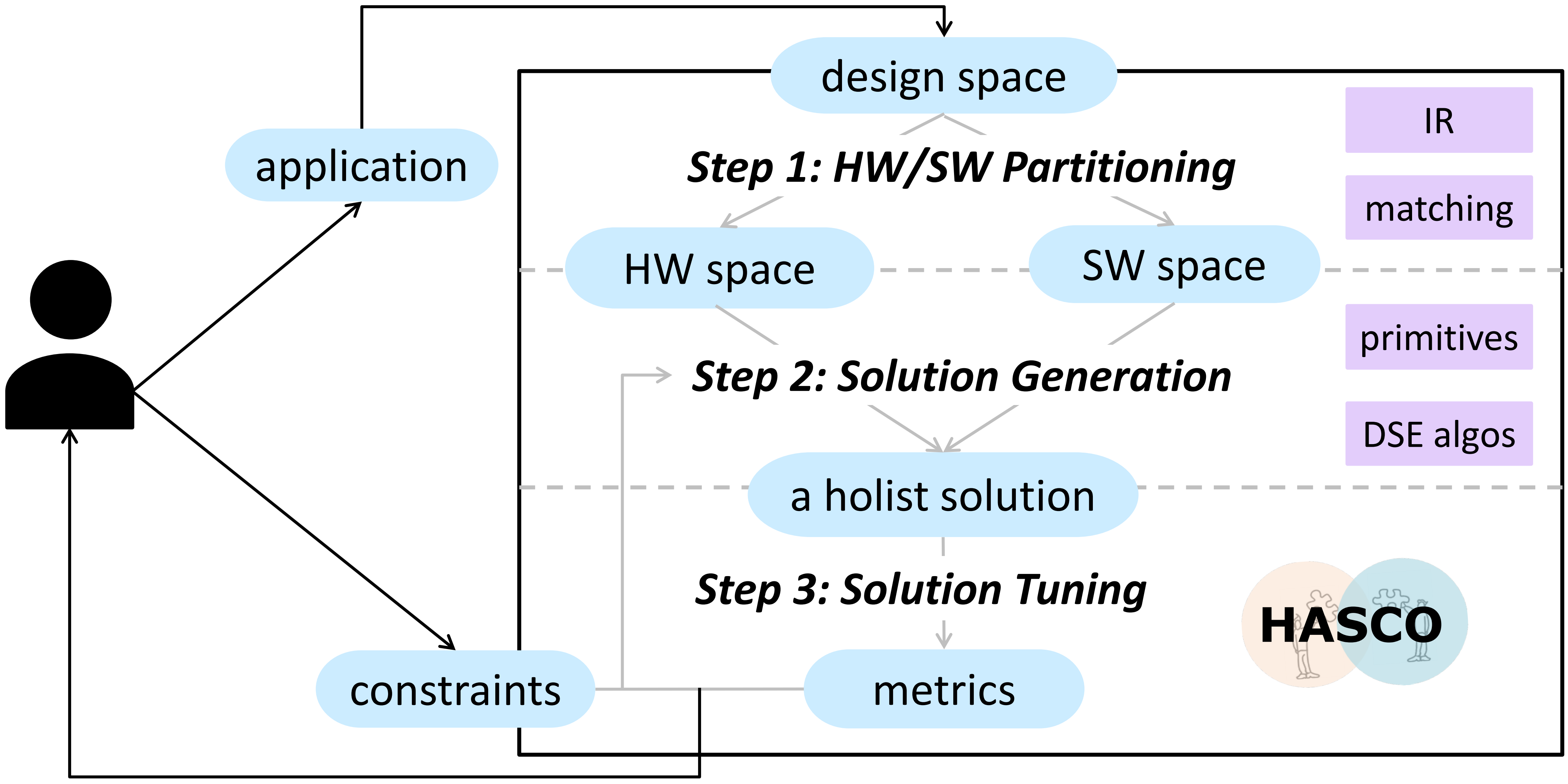
- **Multi-objective: performance,power, area**
- **High evaluation costs**

SW design space



- **Performance-driven**
- **Low evaluation costs**

HASCO: Towards Agile Co-design

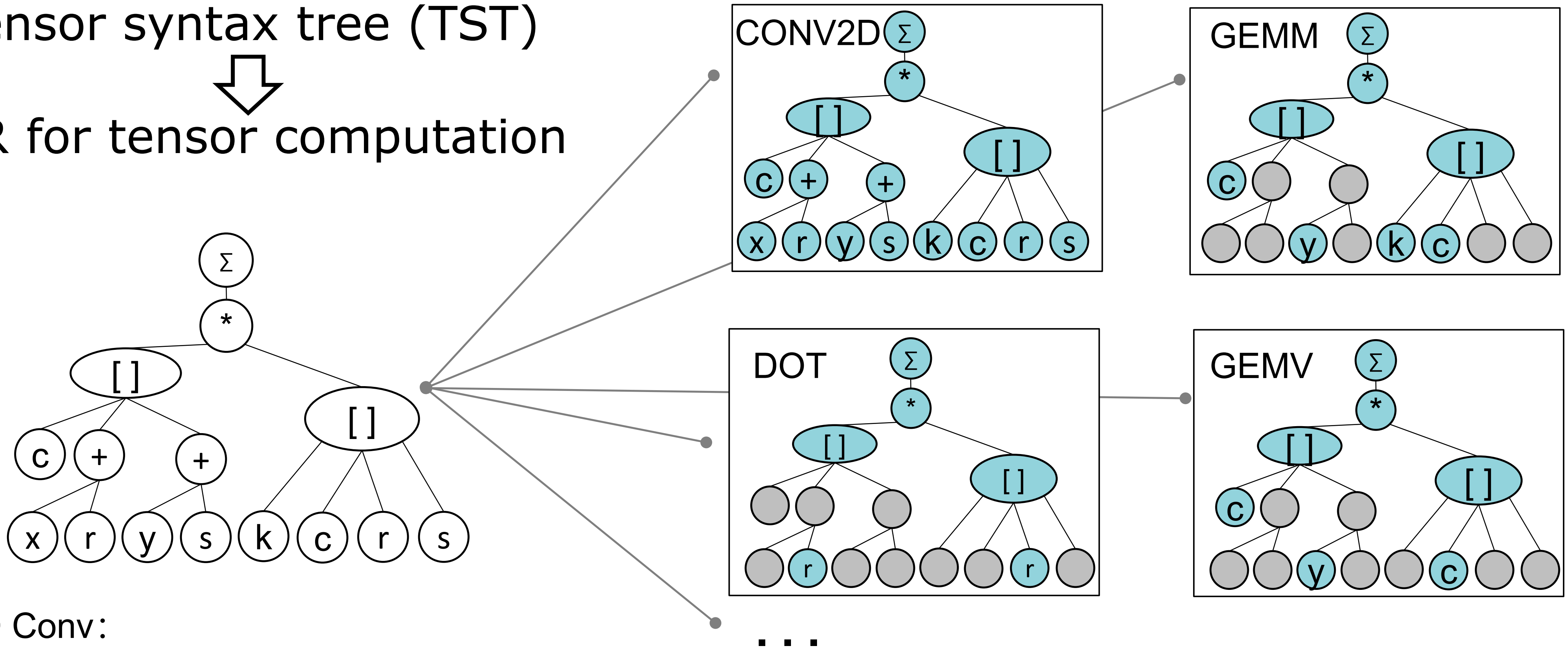


Partition Space

tensor syntax tree (TST)

⇩

IR for tensor computation

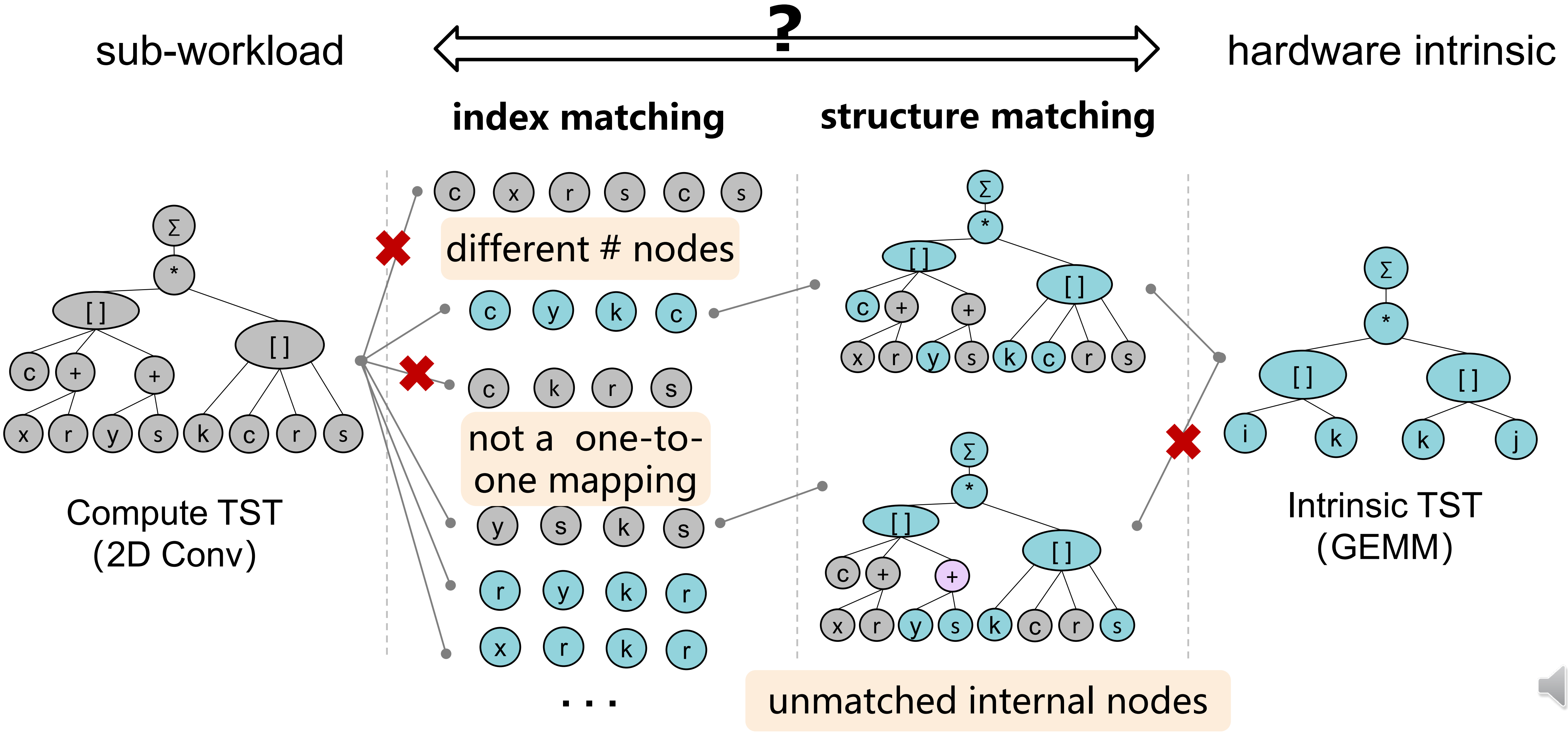


2D Conv：

$$C[k,x,y] = \sum A[c, x + r, y + s] * B[k,c,r,s]$$

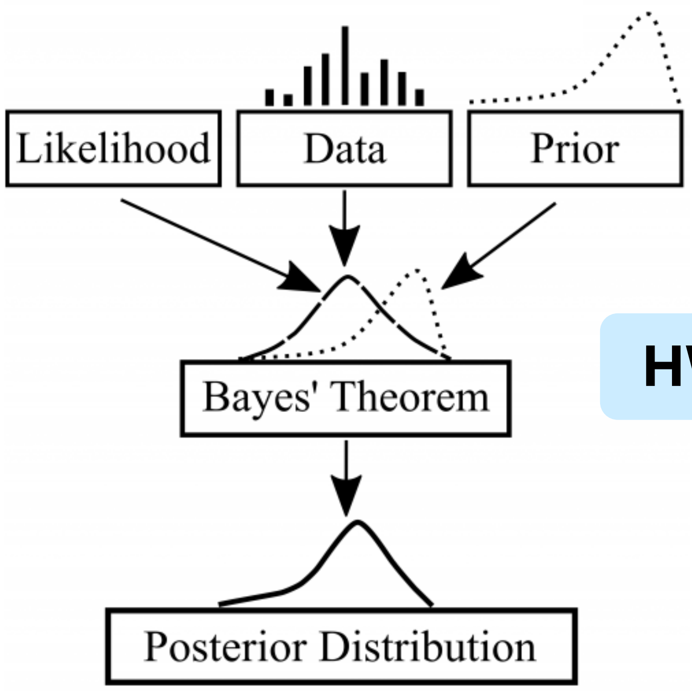
leaf nodes subsets
=> the entire partition space

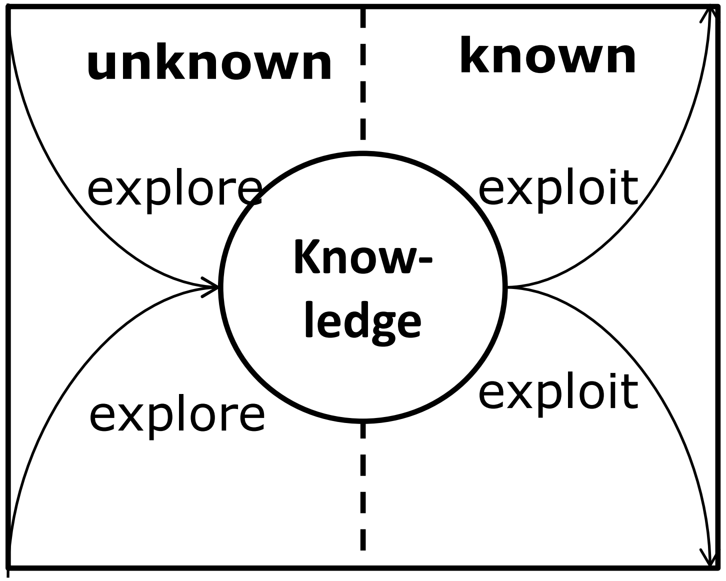
Two-step Matching



Design Space Exploration

jointly optimize software and hardware





Use software latency as a metric

Customize mappings for hardware architecture

HW DSE：Multi-objective Bayesian Optimization

A black-box optimization problem:

Throughput

Power

Area

Metrics

Workload

HW params

$$y = f(w; x)$$

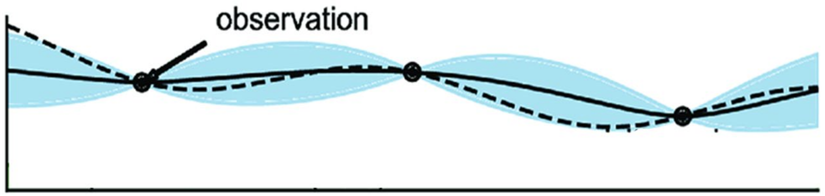
$$\chi = argmax_{x \in \mathcal{X}} f(w; x)$$

Pareto set of params

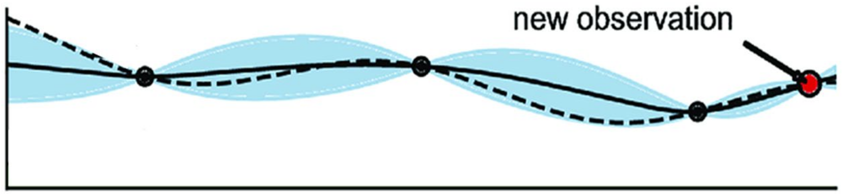
to find the parameters *x* that maximizes *y*

find good solutions in fewer iterations

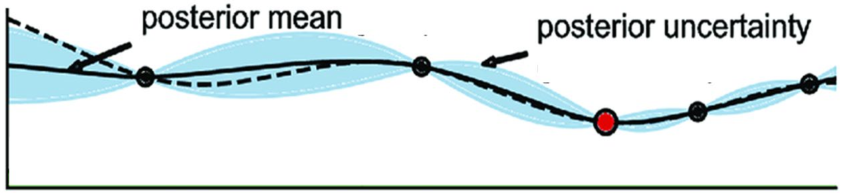
Init. the surrogate model



Observe new HW params and metrics, and update the surrogate model

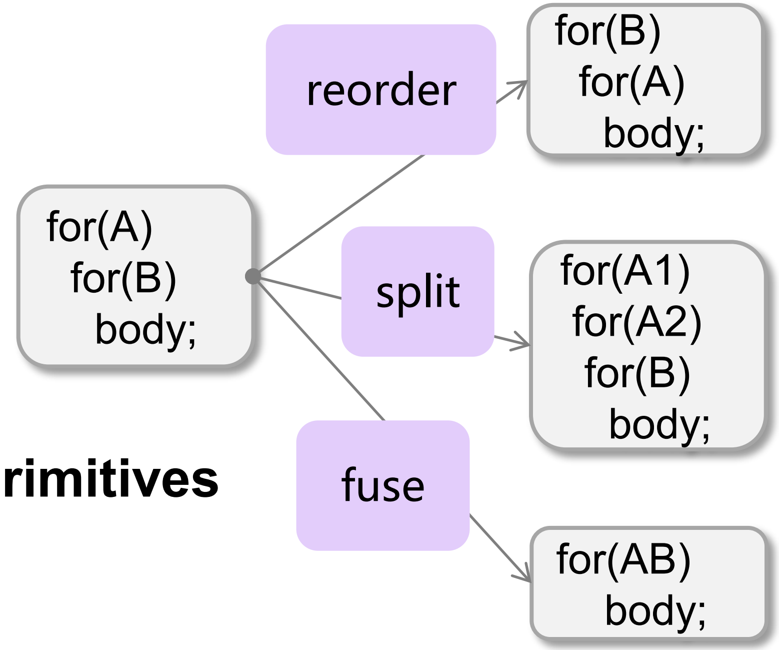


Observe new HW params and metrics, and update the surrogate model



SW Primitives and Optimization

SW DSE => determine primitive sequences and factors



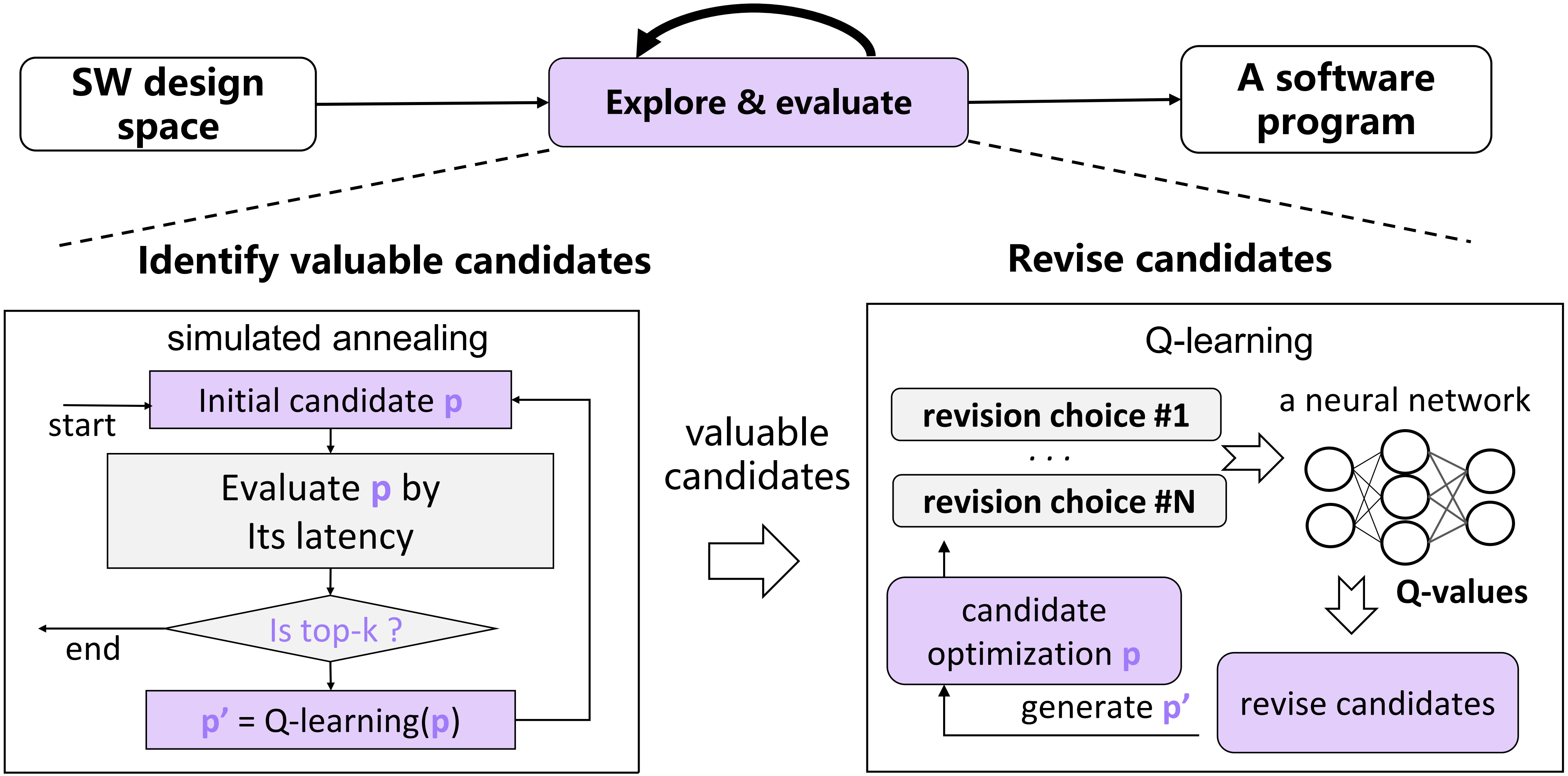
SW primitives

tensorize: uses loops to express a tensorized sub-workload

SW optimization:
a sequence of SW primitives

split:
y(56)→[y1(4), y2(14)],
x(56), r(3), s(3),
k(64)→[k1(2), k2(32)]
c(64)→[c1(2), c2(32)]
reorder:
x, y1, k1, c1, r, s, y2, k2, c2
fuse:
(x, y1, k1, c1) → outer
tensorize:
y2, k2, c2

SW DSE：Reinforcement Learning



HASCO In AHS: Install

```
Install with shell script
sh -c “$(wget https://pku-ahs.github.io/tutorial/en/master/_downloads/9064601015f9cd5e747a641dbdacf3aa/install_ahs.sh -O -)”
source ~/.bashrc

Alternative: use Docker
docker pull ericlyun/ahsmicro:latest
docker run -it ericlyun/ahsmicro:latest /bin/bash
```

Visit our website: https://pku-ahs.github.io/tutorial/en/master/steps.html for details.

HASCO In AHS: User Interface

Python Interface

```
# Defining Params
dtype       = “int8”
method      = “Model” # “Profile”/”Simulate”
constraints = {“latency”: 1000, “power”:20, “area”: 100}
stts        = …
hw_space    = …
# Defining Generator(Hardware Intrinsic)
generator = GEMMGenerator(stts, hw_space, dtype)
# Defining Benchmark
benchmark = BenchmarkCNN(“MobileNetV2”, dtype, generator.type)
# Start codesign
codesign(benchmark, generator, method, constraints,
         init_size=10, trail_num=20)
```

HASCO In AHS: User Interface

Details on HW-Space Definition

```
6   hw_space = {
      "x": [4, 8, 16, 32],
      "y": [4, 8, 16, 32],
      "dma_buswidth": [64, 128],
      "dataflow": ["WS", "OS"]
    }
```

HW DSE

HASCO In AHS: User Interface

Details on STT Definition

```
stts = {
  "OS": [[1, 0, 0],
         [0, 1, 0],
         [1, 1, 1]],
  "WS": [[1, 0, 0],
         [0, 0, 1],
         [1, 1, 1]]
}
```

$\underbrace{\text{"WS"}}_{\text{Name}}$ $\underbrace{\text{[[1, 0, 0], [0, 0, 1], [1, 1, 1]]}}_{\text{Matrix}}$

HW DSE → WS → Call

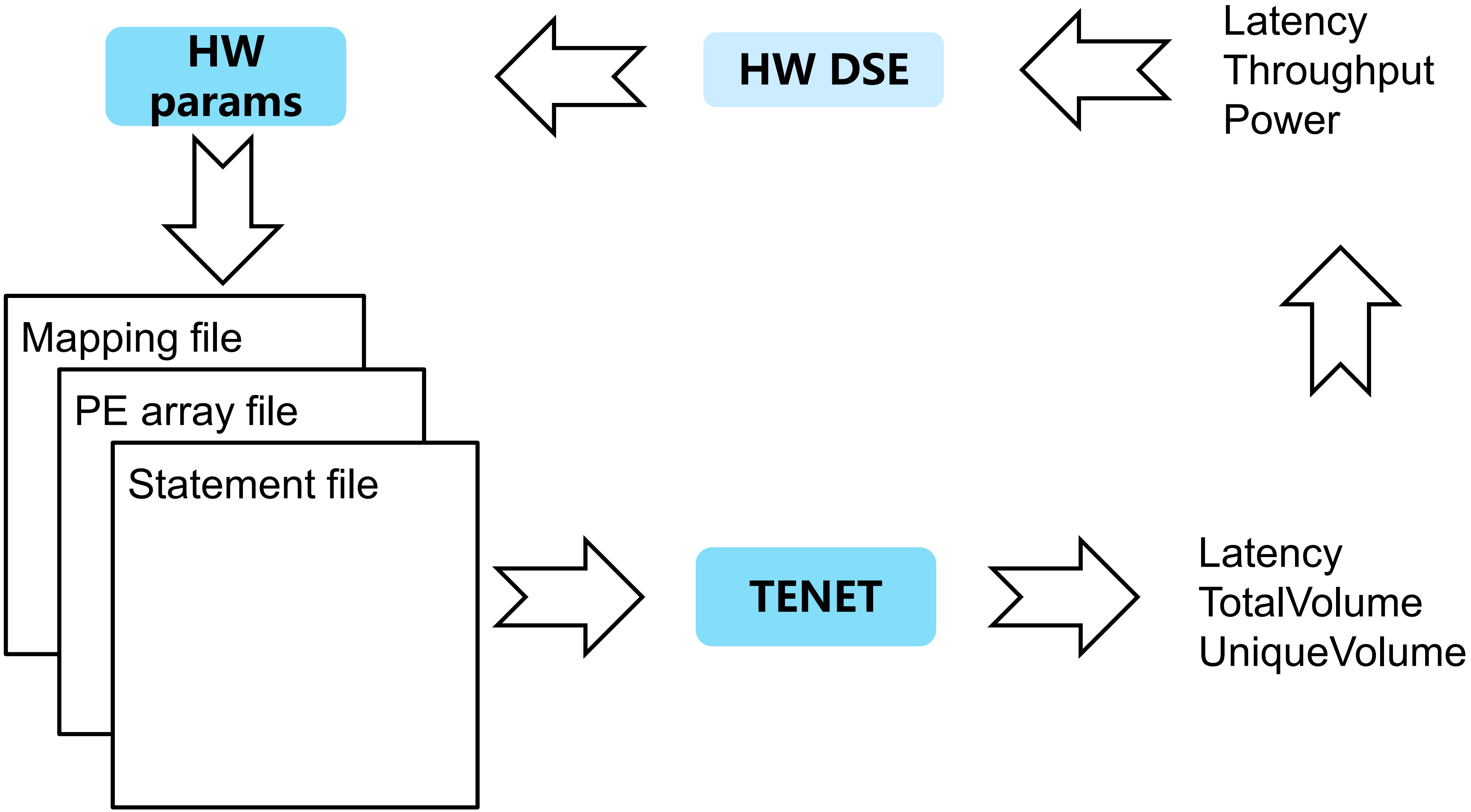
TENET: Relation Centric

```
{S[i,j,k]->PE[i,k]}
{S[i,j,k]->T[i+j+k]}
```

TensorLib: STT Matrix

$$\begin{bmatrix} x \\ y \\ t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

HASCO In AHS: Program Interface



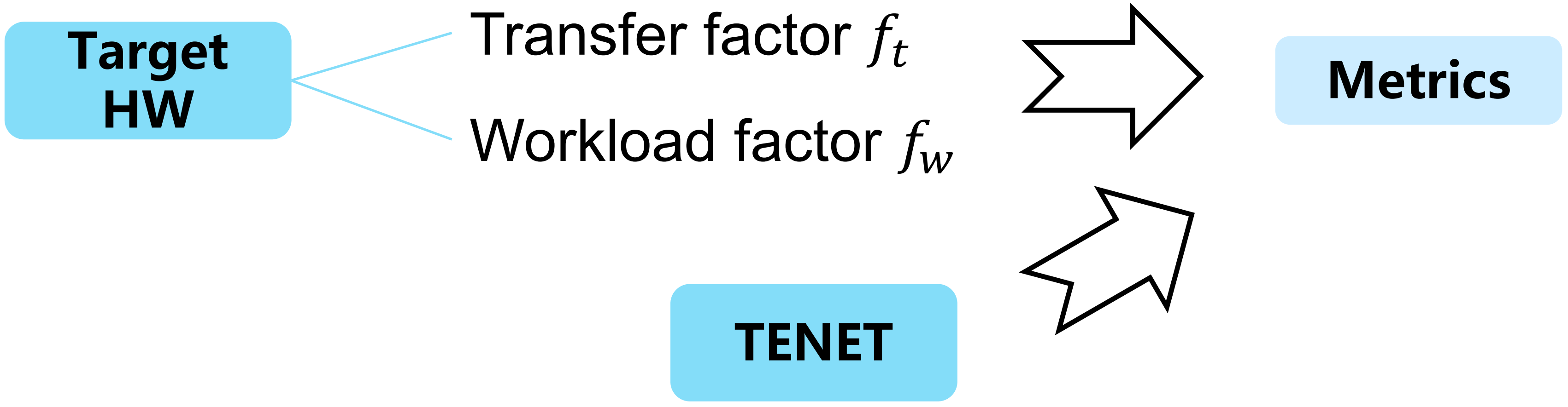
HASCO In AHS: Converting TENET Output

Latency = max(inputDelay, outputDelay, computationDelay)

Throughput = TotalVolume / Latency

Energy = UniqueVolume$\times f_t$ + TotalVolume$\times f_w$

Power = Energy / Latency



HASCO In AHS: Command Line Interface

hasco.py -i CONV -b MobileNetV2 –f space.json

- -i intrinsic, CONV/GEMM
- -b benchmark
- -f the JSON file that specified design space
- -l the restriction on latency
- -p the restriction on power
- -a the restriction on area
- …

```
# Space.json
{
  “stts”: {…}
  “hw_space”: {…}
}
```

HASCO In AHS: Output

- rst directory
  - Summary CSV
  - software
    - IR
  - hardware
    - Config JSON
    - Verilog

HASCO In AHS: DEMO