Timeloop

Accelergy

Angshuman Parashar	NVIDIA
Yannan Nellie Wu	MIT
Po-An Tsai	NVIDIA
Vivienne Sze	MIT
Joel S. Emer	NVIDIA, MIT

ISPASS Tutorial

Part 1: Technical Background

August 2020

Massachusetts Institute of Technology

NVIDIA

Resources

- **Tutorial Website: https://accelergy.mit.edu/tutorial.html**

  Includes infrastructure download and installation instructions

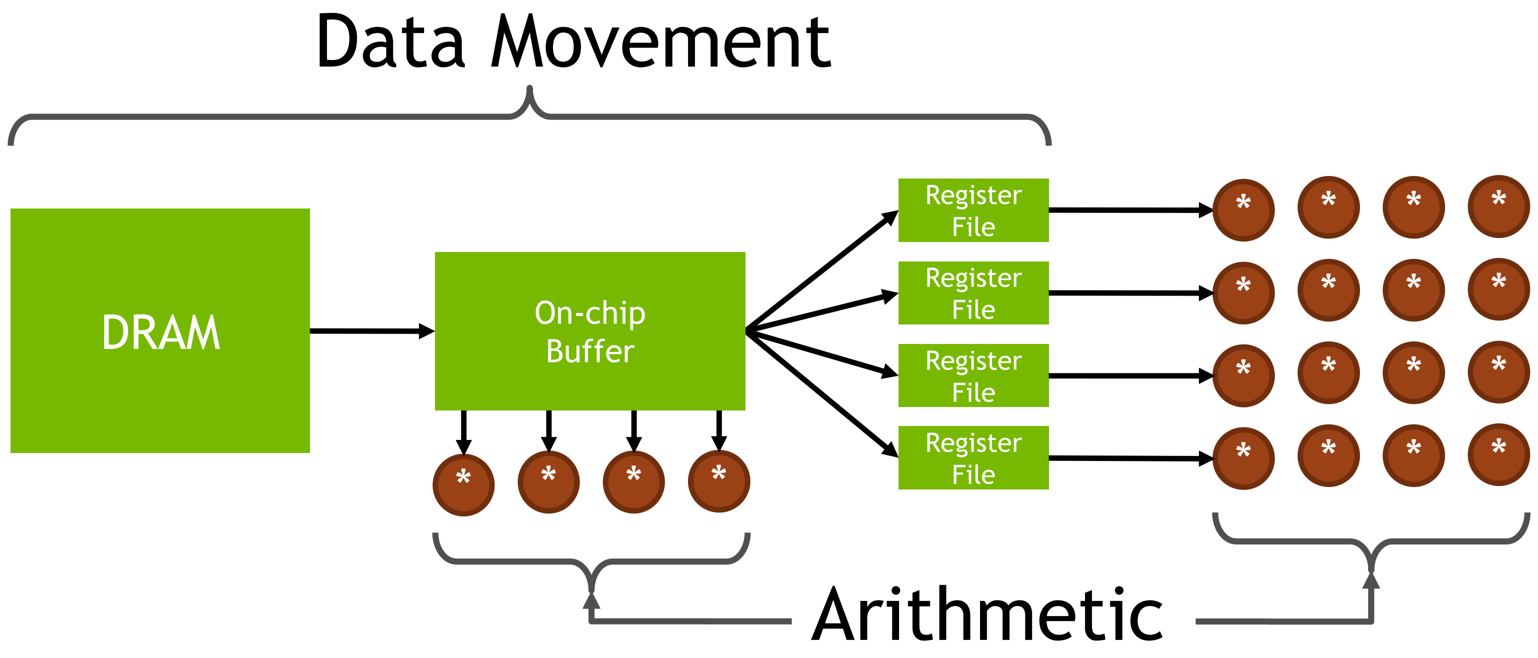
- **Open Source Tools**
  - Accelergy: http://accelergy.mit.edu/
  - Timeloop: https://github.com/NVlabs/timeloop

- **Papers:**
  - A. Parashar, et al. "Timeloop: A systematic approach to DNN accelerator evaluation," ISPASS, 2019.
  - Y. N. Wu, V. Sze, J. S. Emer, “An Architecture-Level Energy and Area Estimator for Processing-In-Memory Accelerator Designs,” *ISPASS,* 2020.
  - Y. N. Wu, J. S. Emer, V. Sze, “Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs,” *ICCAD*, 2019.

MOTIVATION

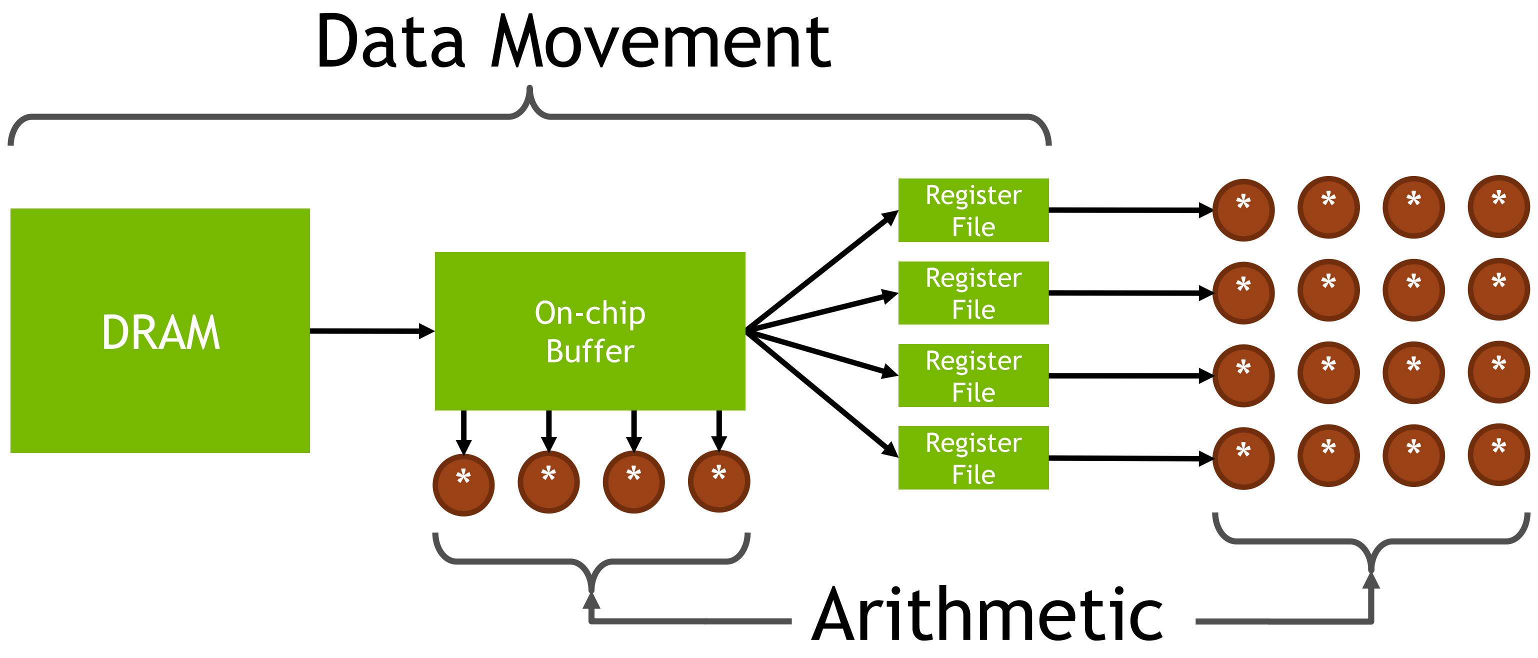
DNN ACCELERATORS

Design Considerations



DNN ACCELERATORS

Design Considerations



DATA MOVEMENT

Why it's important

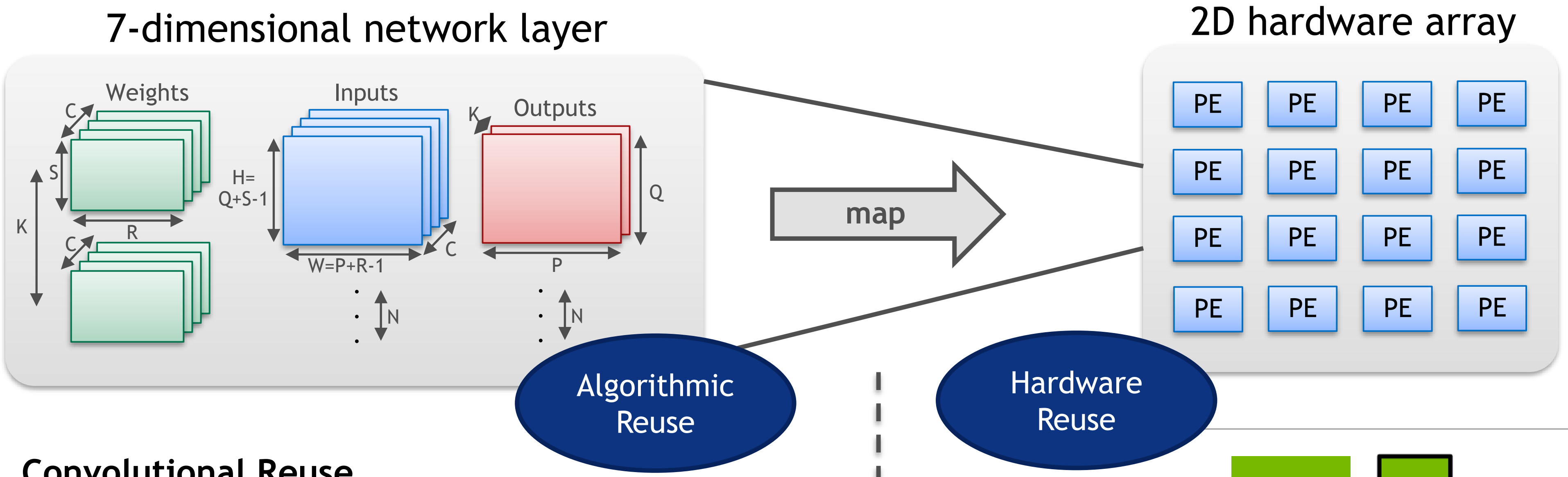
Energy costs	
8-bit Integer Multiply	0.2 pJ
Fetch two 8-bit operands from DRAM	128 pJ
Fetch two 8-bit operands from large SRAM	2 pJ

Fortunately…

VGG16 conv 3_2	
Multiply Add Ops	1.85 Billion
Weights	590 K
Inputs	803 K
Outputs	803 K

Re-use

EXPLOITING REUSE

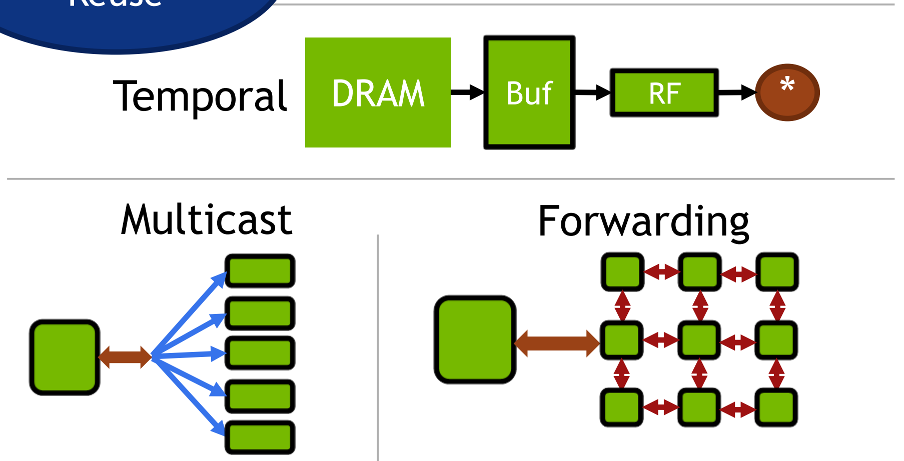


Convolutional Reuse
- Slide filter over input plane

Input Activation Reuse
- Multiple filter blocks over same inputs

Output Activation Reuse
- Accumulation sum over channels

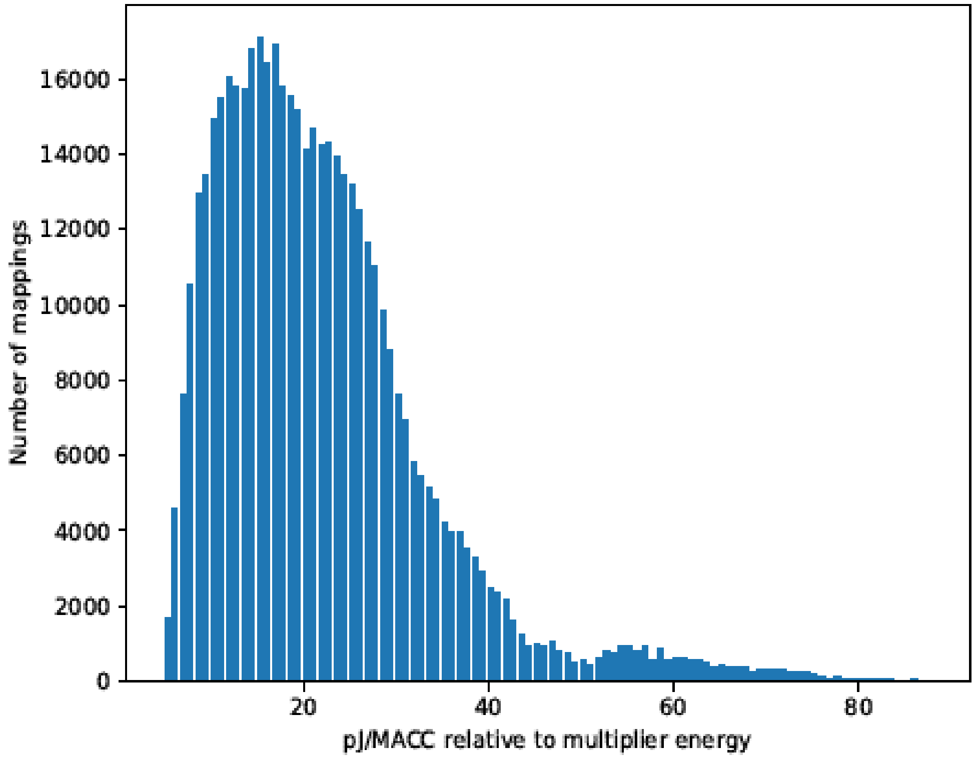
Batch Reuse
- Re-apply filters to new inputs



Flexible architectures may allow millions of alternative mappings of a single workload

MAPPING CHOICES

Energy-efficiency of peak-perf mappings of a single problem



480,000 mappings shown

Spread: 19x in energy efficiency

Only 1 is optimal, 9 others within 1%

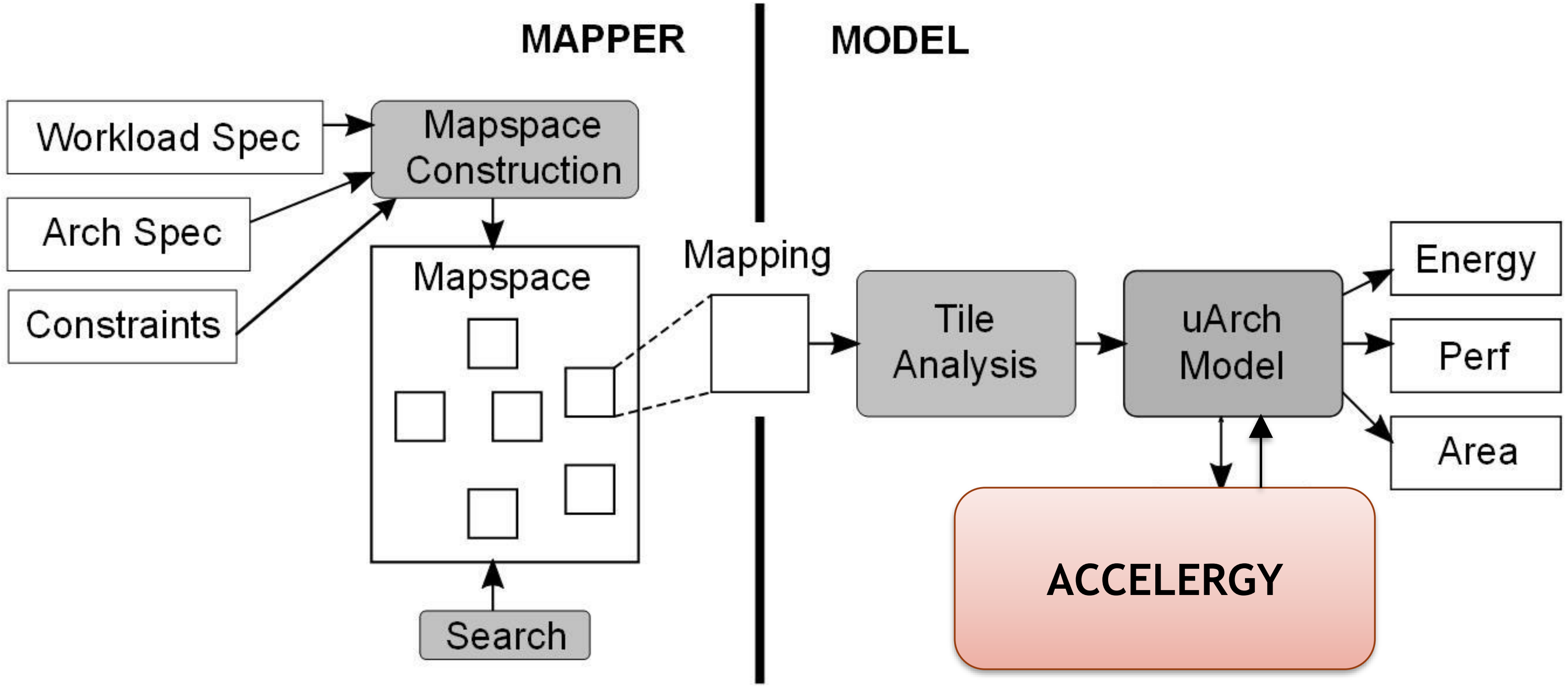
A model needs a mapper to evaluate a DNN workload on an architecture

6,582 mappings have min. DRAM accesses but vary 11x in energy efficiency

A mapper needs a good cost model to find an optimal mapping

TIMELOOP / ACCELERGY

Tools for Evaluation and Architectural Design-Space Exploration of DNN Accelerators



Target every architecture supported by Model

Model variety of DNN accelerators

WHY TIMELOOP/ACCELERGY?

Microarchitectural model (Timeloop/Accelergy)

- Expressive: generic, template based hardware model
- Fast: faster than native execution on host CPUs
- Accurate: validated vs. design-specific models

Technology model (Accelergy)

- Allows user-defined complex architectural components
- Plugins for various technology models, e.g., Cacti, Aladdin, proprietary databases

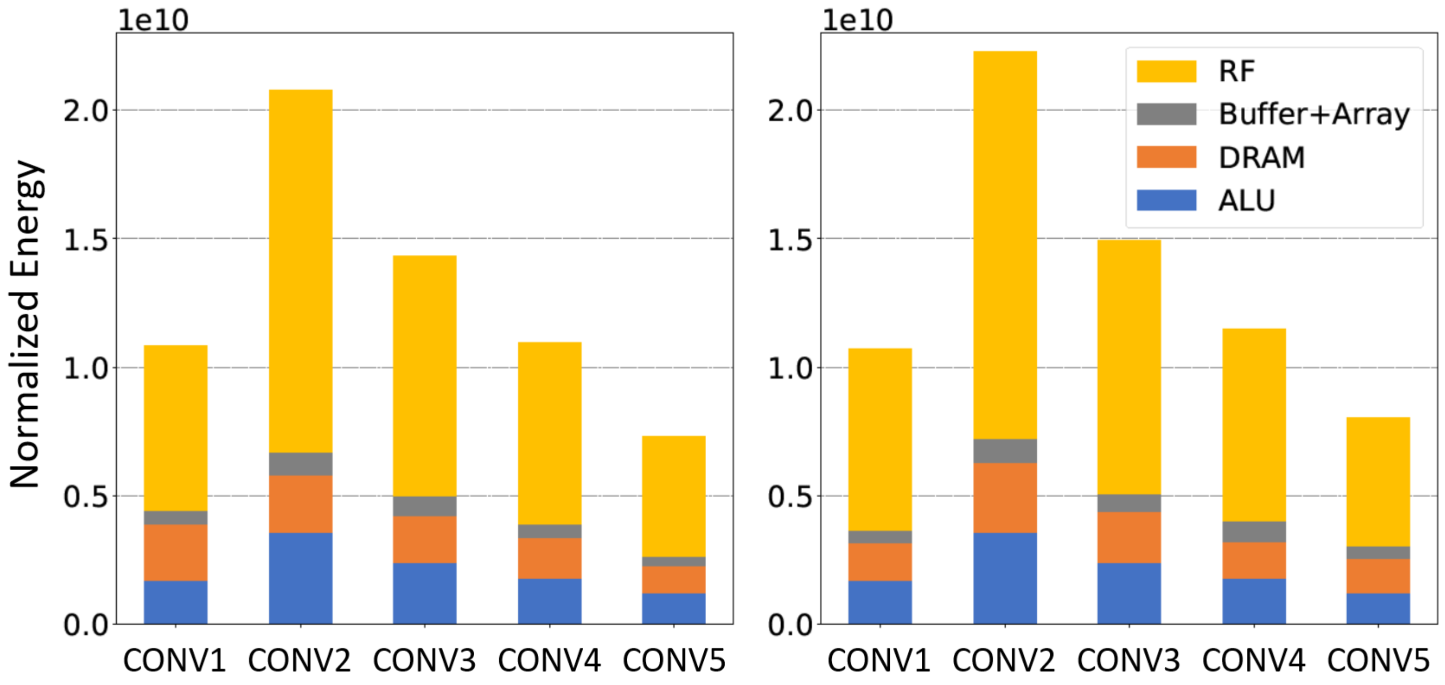
Built-in Mapper (Timeloop)

- Addresses the hard problem of optimizing data reuse, which is required for faithful evaluation of a workload on an architecture

TIMELOOP VALIDATION

VALIDATION: EYERISS

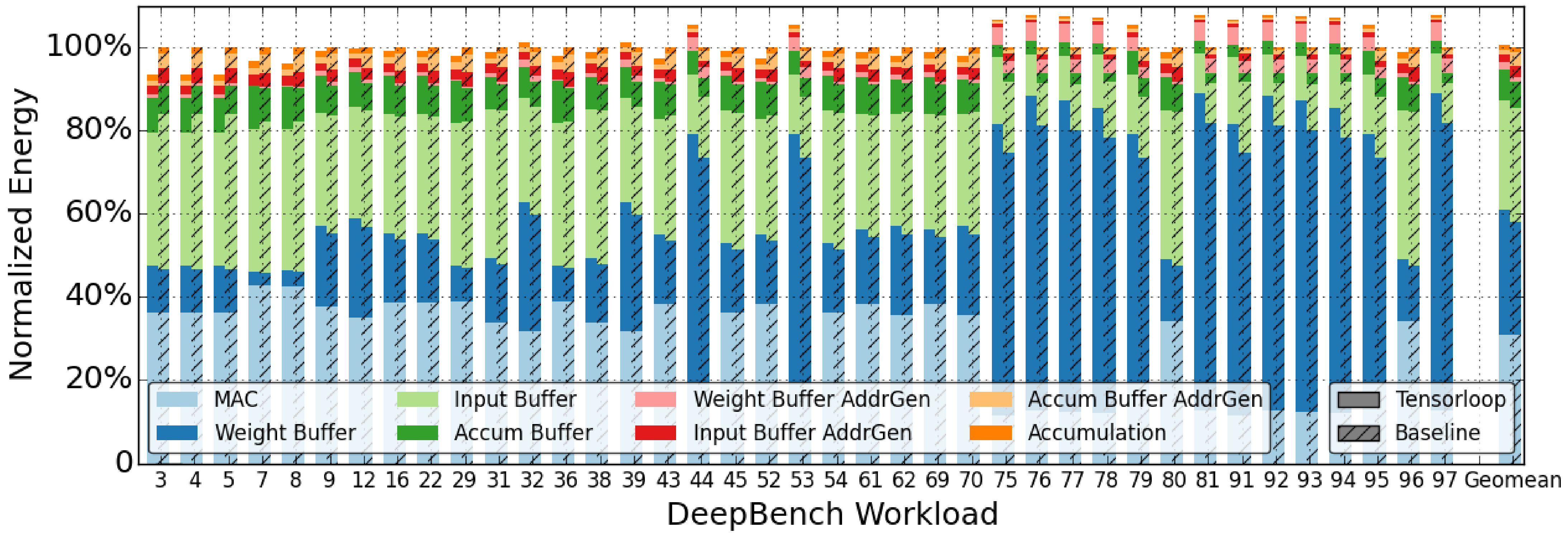
Vs. ISCA 2016 Eyeriss Energy Model



Reference

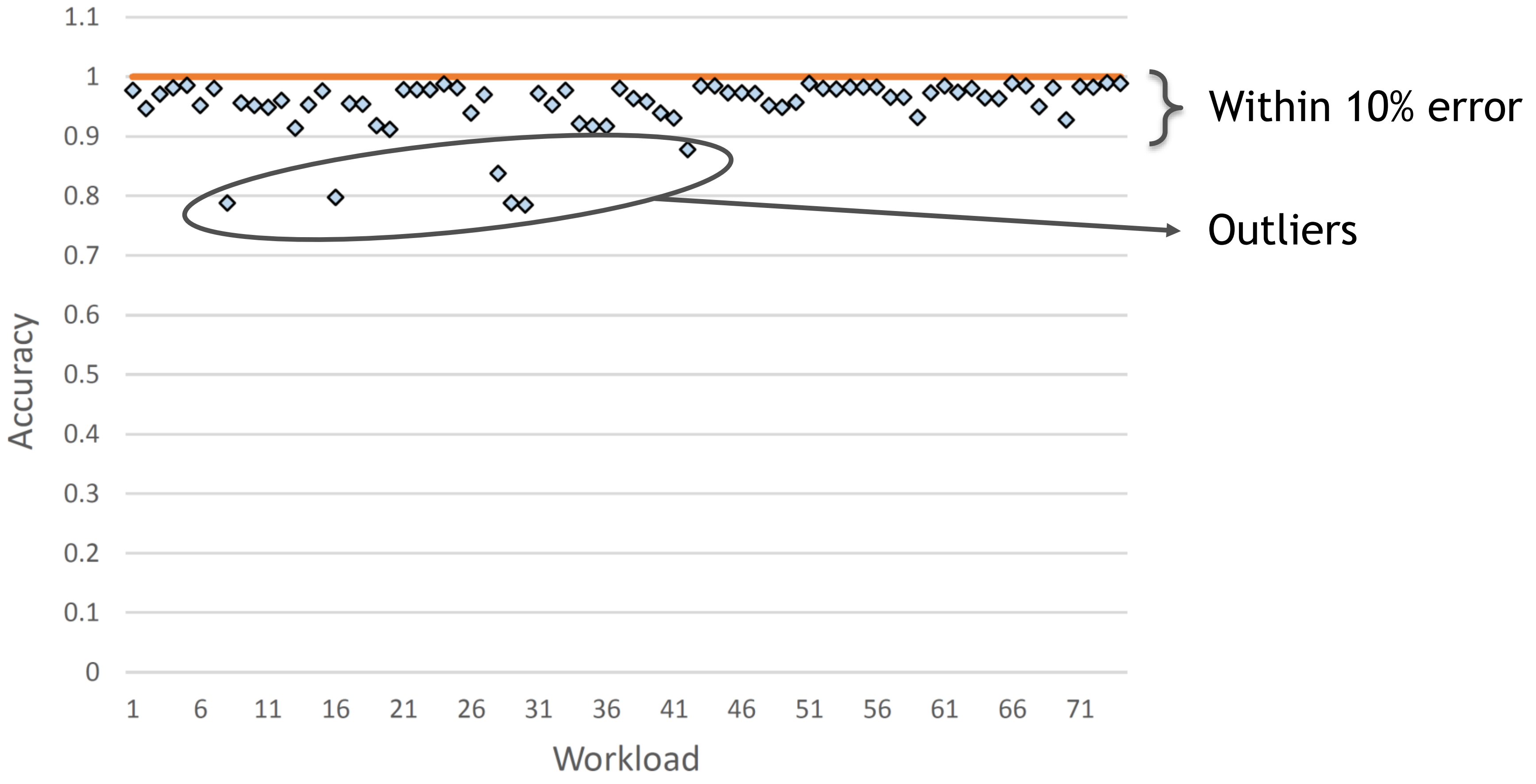
Timeloop

VALIDATION: SIMBA PE (ENERGY)



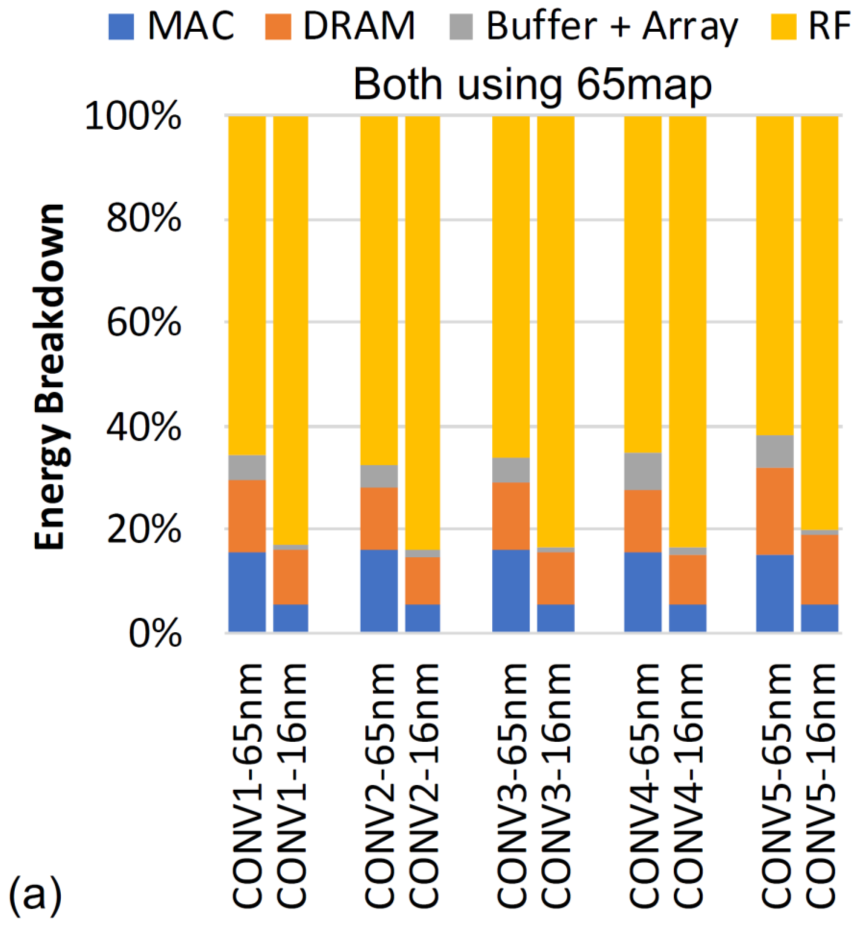
Within 8% error across all workloads

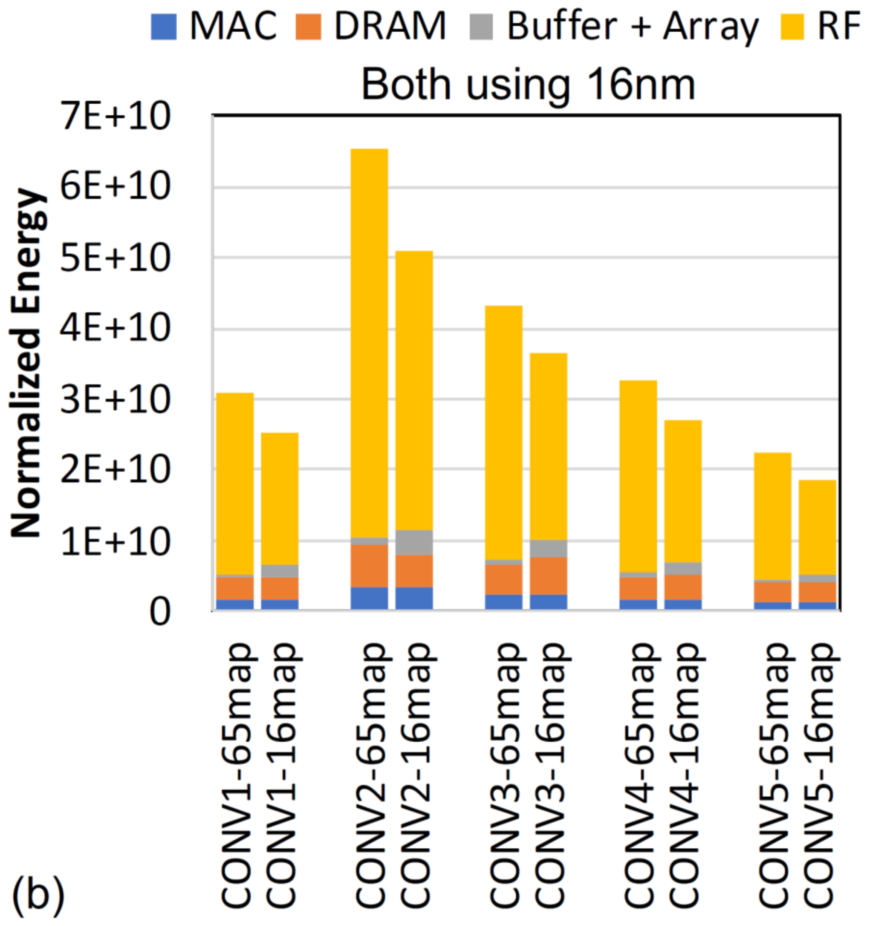
VALIDATION: SIMBA PE (PERFORMANCE)



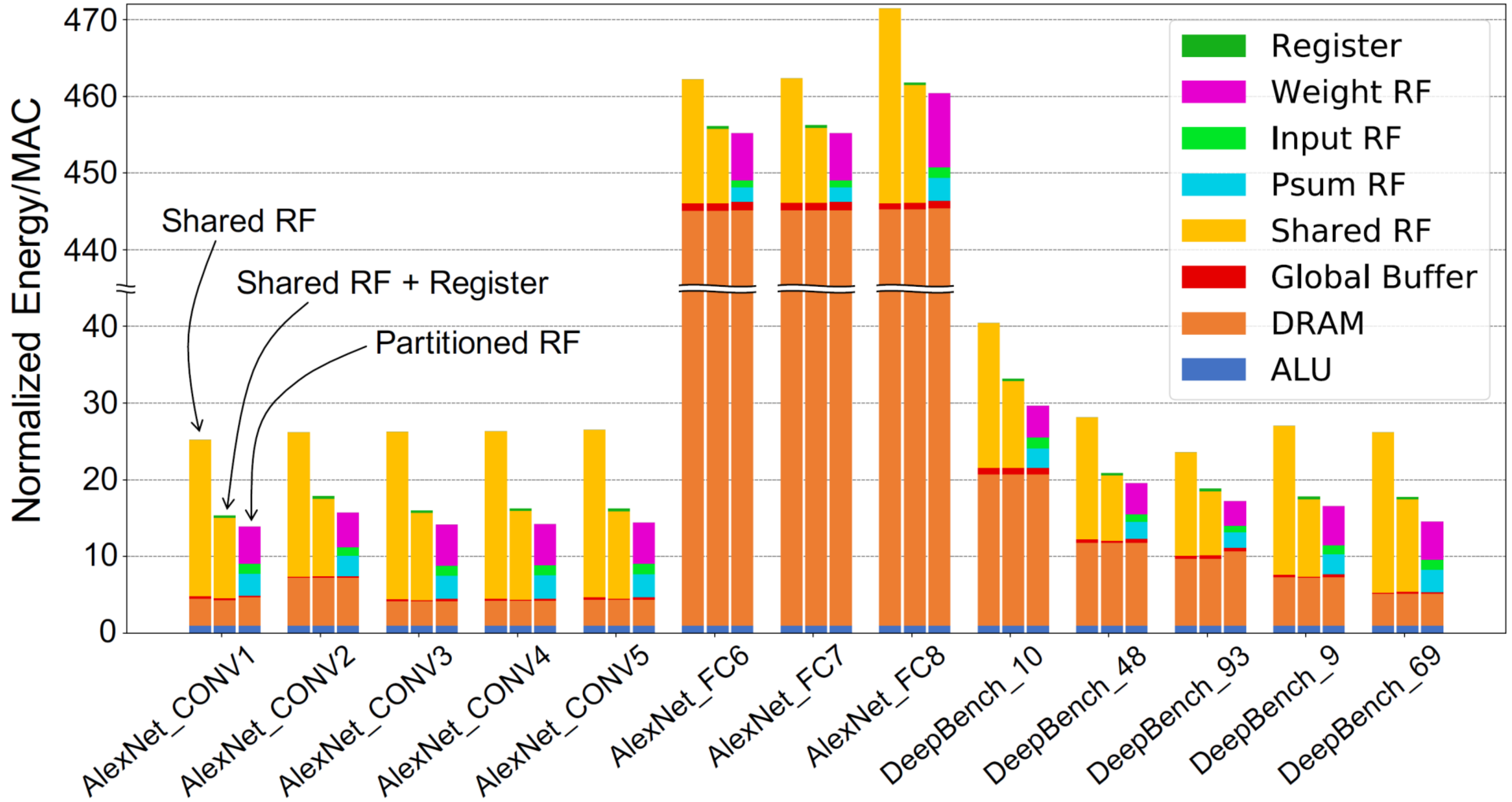
CASE STUDIES

CASE STUDY: TECHNOLOGY MODEL





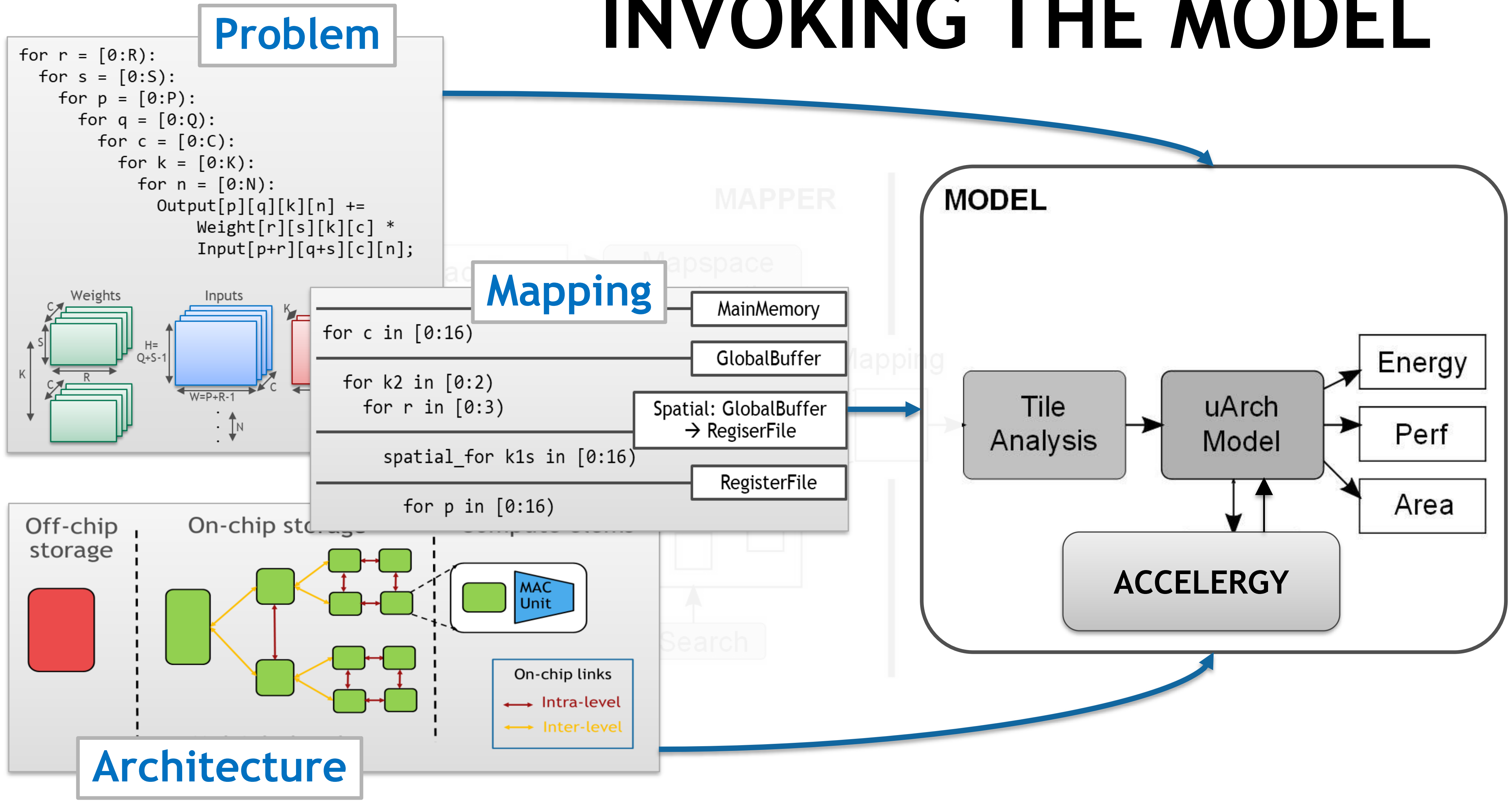
CASE STUDY: MEM HIERARCHY



USING TIMELOOP

THE MODEL

INVOKING THE MODEL



Problem

```
for r = [0:R):
  for s = [0:S):
    for p = [0:P):
      for q = [0:Q):
        for c = [0:C):
          for k = [0:K):
            for n = [0:N):
              Output[p][q][k][n] +=
                  Weight[r][s][k][c] *
                  Input[p+r][q+s][c][n];
```

Mapping

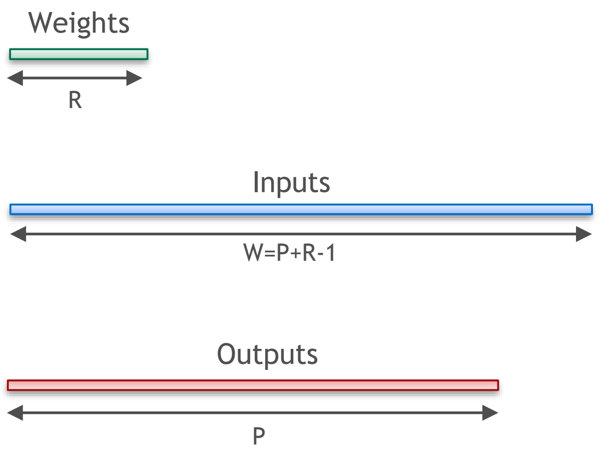
```
                                       MainMemory
for c in [0:16)
                                       GlobalBuffer
  for k2 in [0:2)
    for r in [0:3)
                                       Spatial: GlobalBuffer
                                       → RegiserFile
      spatial_for k1s in [0:16)
                                       RegisterFile
        for p in [0:16)
```

EXAMPLE 0: PROBLEM

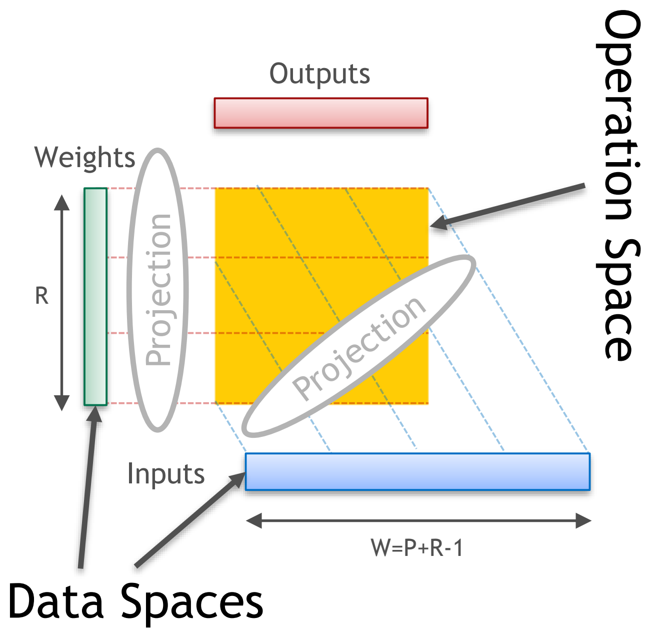
Conv1D

To represent this…

```
for r = [0:R):
  for p = [0:P):
    Output[p] += Weight[r] * Input[p+r];
```



Think about:



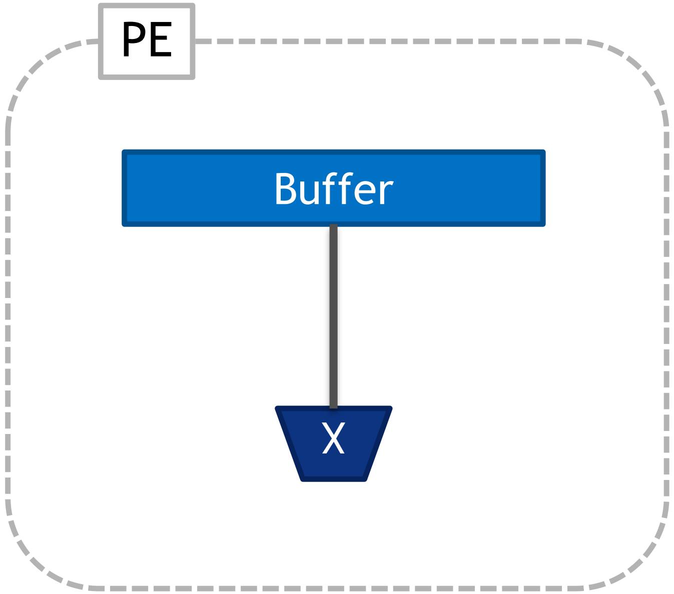
And write:

```
problem:
  shape:
    name: Conv1D
    dimensions: [ R, P ]
    data-spaces:
    - name: Weights
      projection:
      - [ [R] ]
    - name: Inputs
      projection:
      - [ [P], [R] ]
    - name: Outputs
      projection:
      - [ [P] ]
      read-write: True

  instance:
    R: 3
    P: 16
```

EXAMPLE 0: ARCHITECTURE

1-Level Temporal

To represent this...



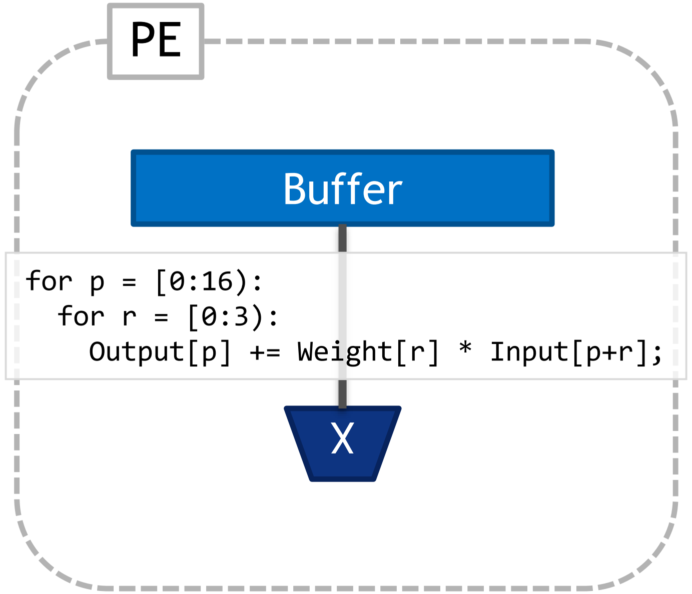
Write:

```
architecture:
subtree:
    - name: PE
      local:
      - name: Buffer
        class: SRAM
        attributes:
          entries: 64
          instances: 1
          word-bits: 8

      - name: MACC
        class: intmac
        attributes:
          word-bits: 8
```

EXAMPLE 0: MAPPING

1-Level Temporal

To represent this…



```
for p = [0:16):
  for r = [0:3):
    Output[p] += Weight[r] * Input[p+r];
```

Write:

```
mapping:
  - target: Buffer
    type: temporal
    factors: R=3 P=16
    permutation: RP
```

EXAMPLE 0

Run Timeloop model:

```
>> timeloop-model arch.yaml problem.yaml map.yaml
```

Output:

timeloop-model.map.txt

```
Buffer [ Weights:3 Inputs:18 Outputs:16 ]
-----------------------------------------
| for P in [0:16)
|   for R in [0:3)
```

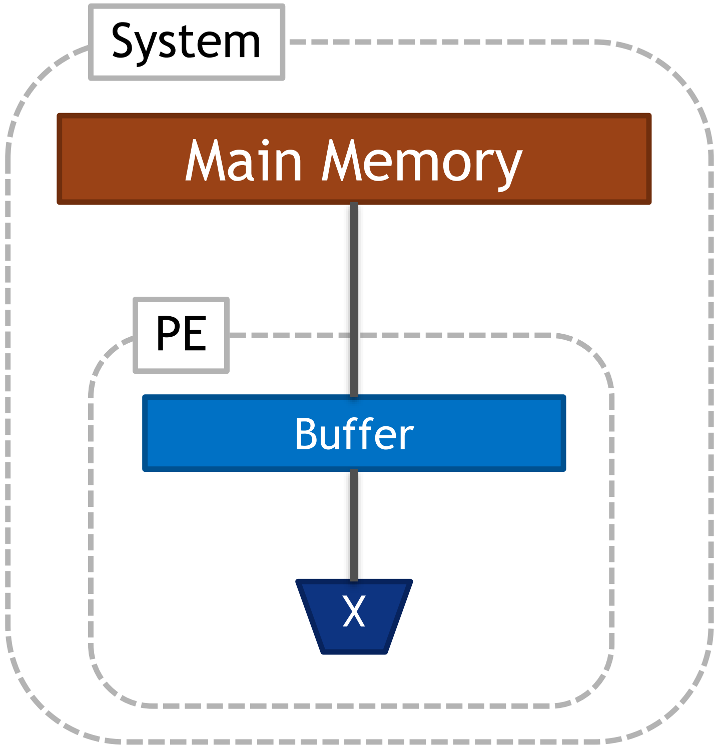
timeloop-model.stats.txt

```
......
......
Summary Stats
-------------
Utilization: 1.00
Cycles: 48
Energy: 0.00 uJ
Area: 0.00 mm^2

MACCs = 48
pJ/MACC
    MACC                        = 0.60
    Buffer                      = 1.54
    Total                       = 2.14
```

EXAMPLE 1: ARCHITECTURE

2-Level Temporal

To represent this…



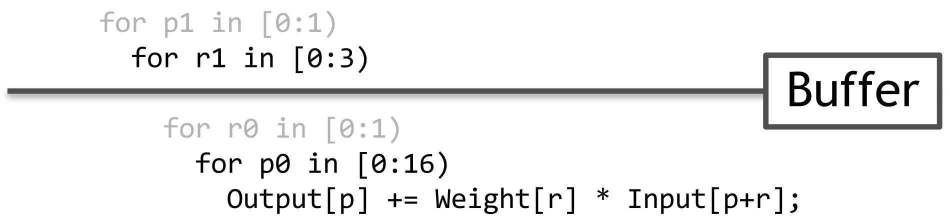
Write:

```
arch:
  subtree:
  - name: System
    local:
    - name: MainMemory
      class: DRAM
      attributes:
        word-bits: 8

    subtree:
    - name: PE
      local:
      - name: Buffer
        class: SRAM
        attributes:
          entries: 64
          instances: 1
          word-bits: 8

      - name: MACC
        class: intmac
        attributes:
          word-bits: 8
```

EXAMPLE 1: MAPPING

Weight Stationary

To represent this…



```
for p1 in [0:1)
  for r1 in [0:3)

    for r0 in [0:1)
      for p0 in [0:16)
        Output[p] += Weight[r] * Input[p+r];
```

Expected outputs

Metric	Weights	Inputs	Outputs
Buffer occupancy	1	P	P
MainMemory accesses	R	W	P
Buffer accesses	PR	PR	2PR

Write:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=3 P=1
    permutation: RP # inner to outer

  - target: Buffer
    type: temporal
    factors: R=1 P=16
    permutation: PR # inner to outer
```

EXAMPLE 1: MAPPING

Output Stationary

To represent this…

```
for r1 in [0:1)
  for p1 in [0:16)
------------------------------------ Buffer
    for p0 in [0:1)
      for r0 in [0:3)
        Output[p] += Weight[r] * Input[p+r];
```

Expected outputs

Metric	Weights	Inputs	Outputs
Buffer occupancy	R	R	1
MainMemory accesses	R	W	P
Buffer accesses	PR	PR	2PR

Write:

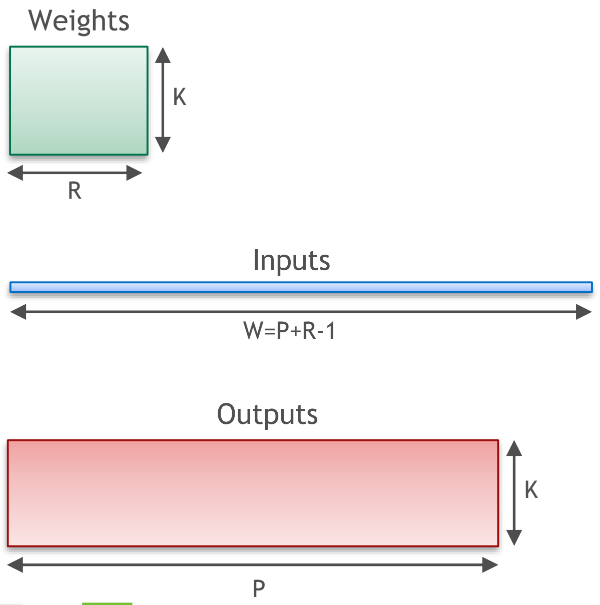
```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=16
    permutation: PR

  - target: Buffer
    type: temporal
    factors: R=3 P=1
    permutation: RP
```

EXAMPLE 2: PROBLEM

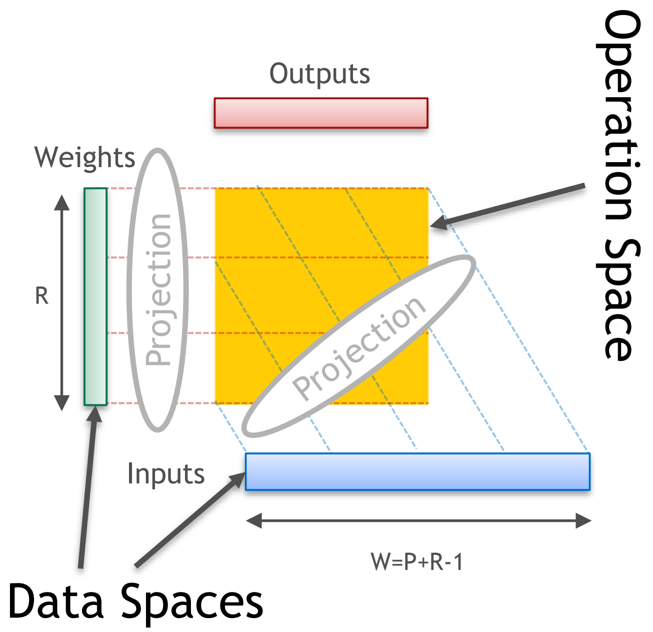
Conv1D + Output Channels

To represent this…

```
for k = [0:K)
  for r = [0:R):
    for p = [0:P):
      Output[k][p] += Weight[k][r] * Input[p+r];
```



Think about:



And write:

```
problem:
  shape:
    name: Conv1D
    dimensions: [ K, R, P ]
    data-spaces:
    - name: Weights
      projection:
      - [ [K] ]
      - [ [R] ]
    - name: Inputs
      projection:
      - [ [P], [R] ]
    - name: Outputs
      projection:
      - [ [K] ]
      - [ [P] ]
      read-write: True

  instance:
    K: 32
    R: 3
    P: 16
```

EXAMPLE 2: MAPPINGS

Untiled vs. K-tiled

Untiled

```
for k1 in [0:32)
  for p1 in [0:16)
    for r1 in [0:1)
                                        Buffer
      for k0 in [0:1)
        for p0 in [0:1)
          for r0 in [0:3)
            Output[p] += Weight[r] * Input[p+r];
```

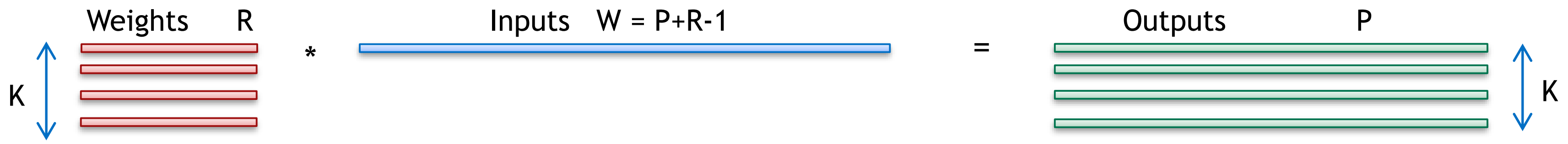
```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=16 K=32
    permutation: RPK

  - target: Buffer
    type: temporal
    factors: R=3 P=1 K=1
    permutation: RPK
```

K-tiled

```
for k1 in [0:16)
  for p1 in [0:16)
    for r1 in [0:1)
                                        Buffer
      for k0 in [0:2)
        for p0 in [0:1)
          for r0 in [0:3)
            Output[p] += Weight[r] * Input[p+r];
```

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=16 K=16
    permutation: RPK

  - target: Buffer
    type: temporal
    factors: R=3 P=1 K=2
    permutation: RPK
```

EXAMPLE 2: O.S. DATAFLOW VARIANTS



Alg. min. MainMemory accesses

Weights	Inputs	Outputs
KR	W	KP

Buffer occupancy

Weights	Inputs	Outputs
R	R	1
R	R	1
R	W	1
KR	R	1
K_bR	R	1
R	$R+P_b-1$	1

MainMemory accesses

Weights	Inputs	Outputs
KR	KW	KP
KPR	W	KP
KR	W	KP
KR	W	KP
KR	$(K/K_b)W$	KP
$K(P/P_b)R$	W	KP

$$\bigvee_{k=1}^{K}\bigvee_{p=1}^{P}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$

$$\bigvee_{p=1}^{P}\bigvee_{k=1}^{K}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$

$$\bigvee_{k_1=1}^{K_1}\bigvee_{p=1}^{P}\bigvee_{k_0=1}^{K_0}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$

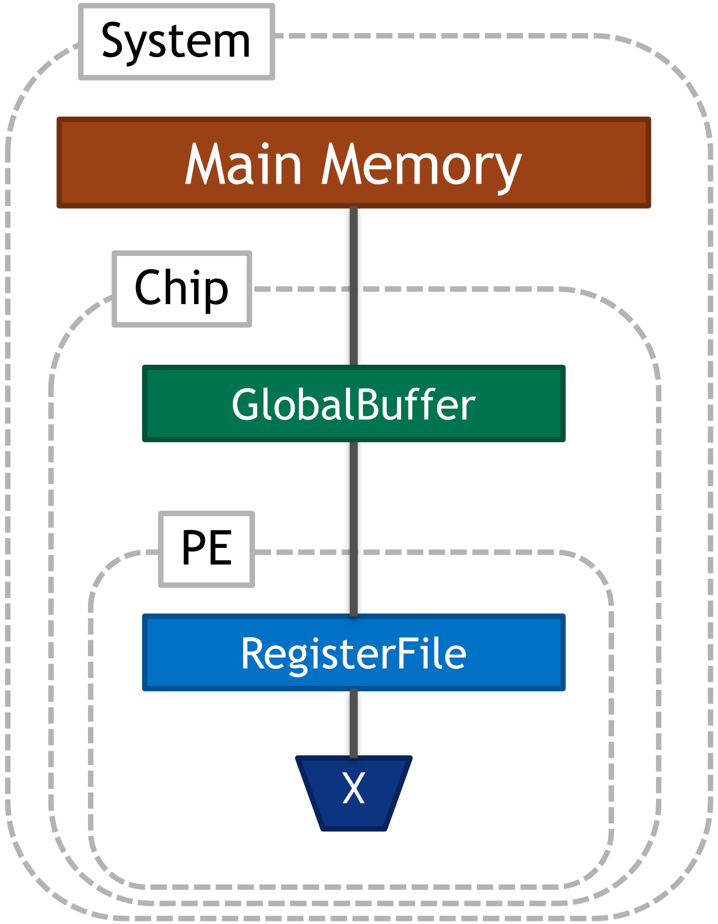
where $K = K_1 \times K_0$ and $k = k_1K_0 + k_0$

$$\bigvee_{p_1=1}^{P_1}\bigvee_{k=1}^{K}\bigvee_{p_0=1}^{P_0}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$

where $P = P_1 \times P_0$ and $p = p_1P_0 + p_0$

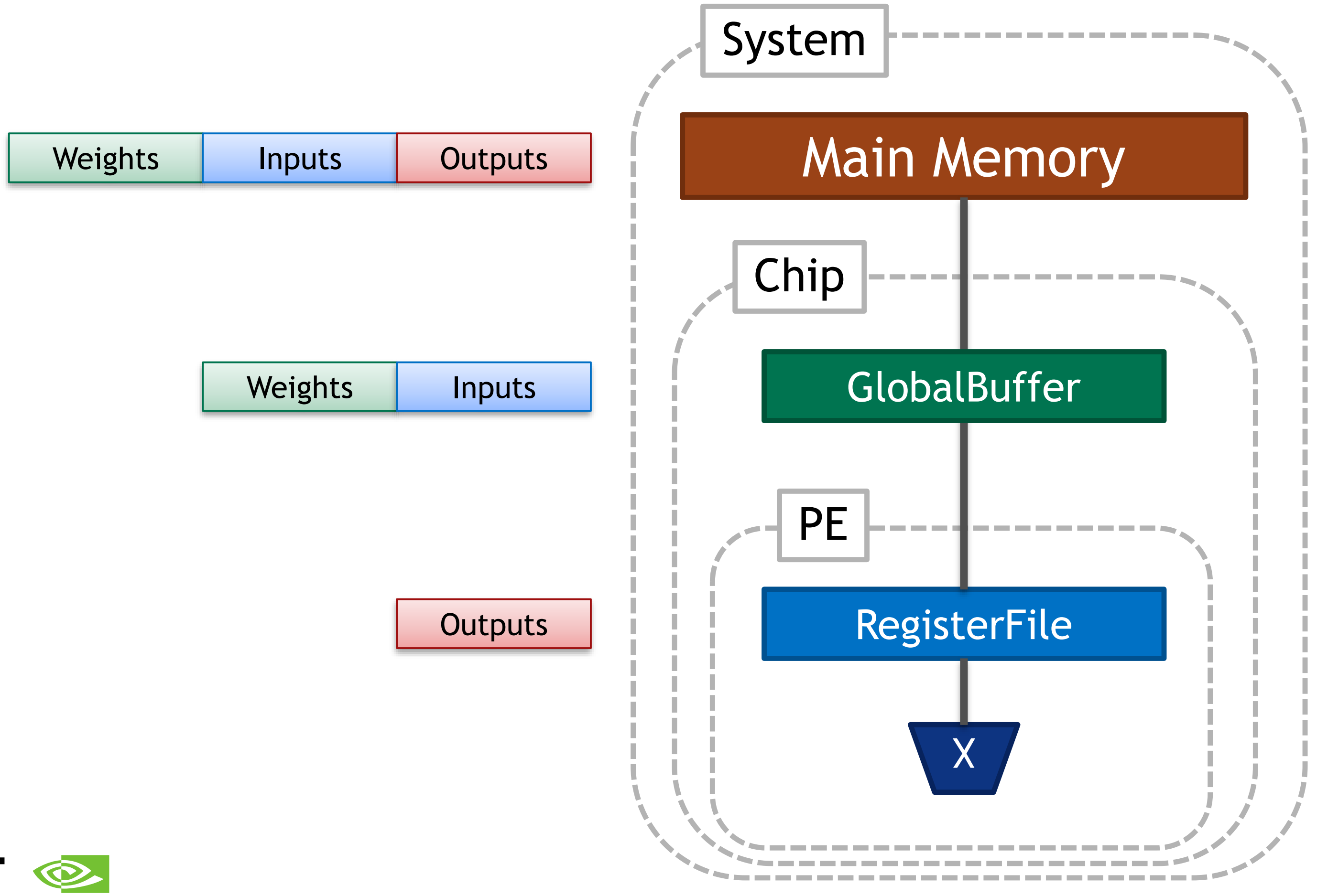
EXAMPLE 3: ARCHITECTURE

3-Level Temporal



EXAMPLE 3B: BYPASSING LEVELS

3-Level Temporal with Level Bypassing



```
mapping:

  ...

  - target: GlobalBuffer
    type: bypass
    keep:
    - Weights   # same as default
    - Inputs    # same as default
    bypass:
    - Outputs   # override

   - target: RegisterFile
    type: bypass
    keep:
    - Outputs   # same as default
    bypass:
    - Weights   # override
    - Inputs    # override
```

EXAMPLE 3B: BYPASSING

Bypassing

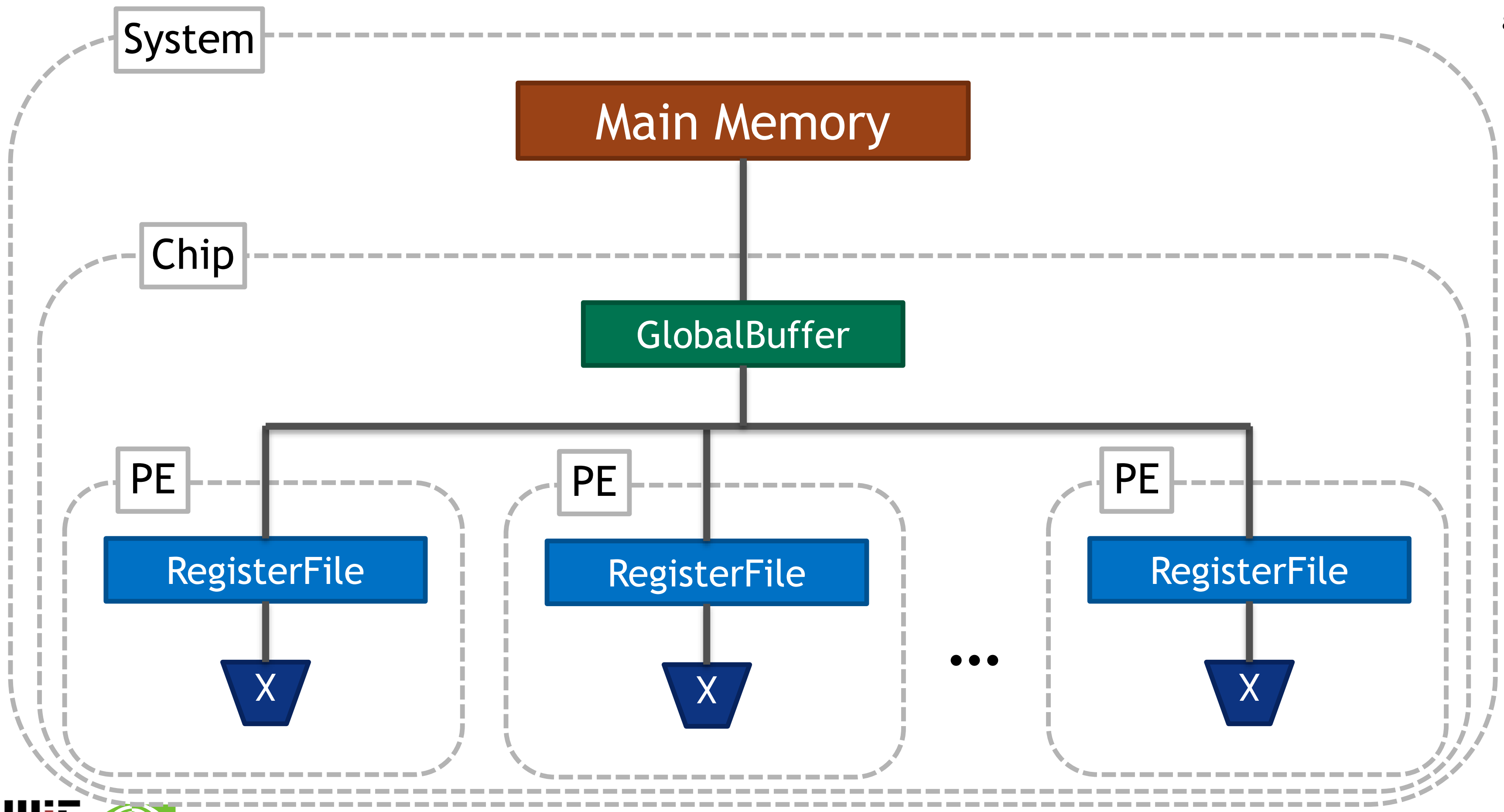
- Avoids energy cost of reading and writing buffers
- May result in additional accesses to outer buffers
- Does not change energy cost of moving data over network wires

For brevity in expressing mappings, Timeloop's model application assumes each data space is stored at each level.

- We will see later that Timeloop's *mapper* makes no such assumption

EXAMPLE 4: SPATIAL INSTANCES

3-Level with multiple PEs

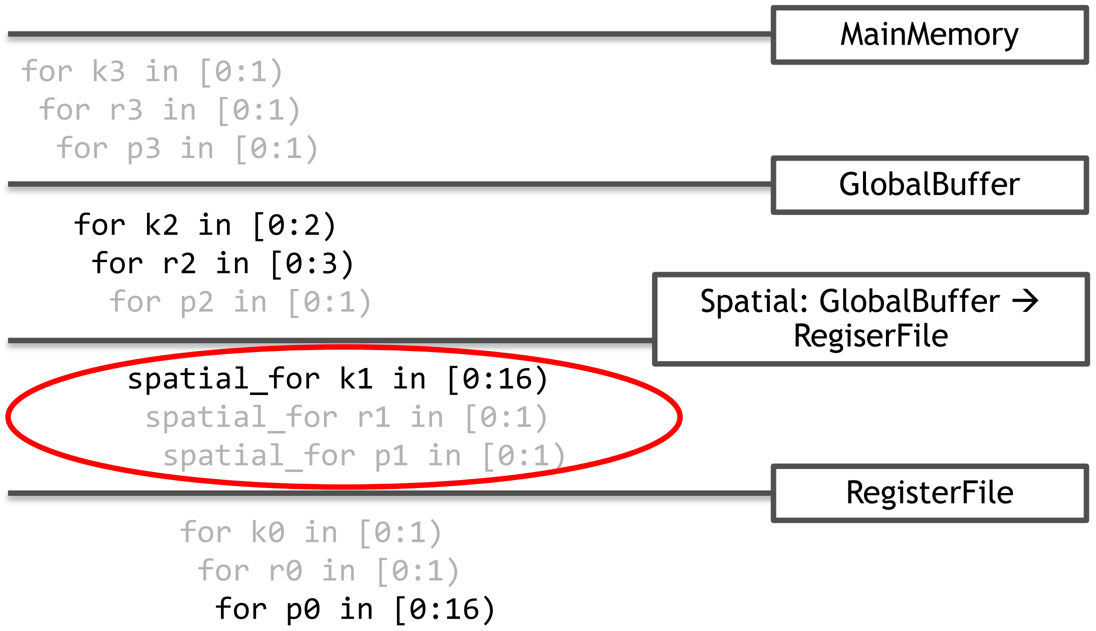


```
architecture:
  subtree:
  - name: System
    local:
    - name: MainMemory
      class: DRAM
      attributes:
        ......
    subtree:
    - name: Chip
      local:
      - name: GlobalBuffer
        class: SRAM
        attributes:
          ......
      subtree:
      - name: PE[0..15]
        local:
        - name: RegisterFile
          class: regfile
          attributes:
            ......
        - name: MACC
          class: intmac
          attributes:
            ......
```

EXAMPLE 4: MAPPING

Spatial levels need loops too

To represent this…



```
for k3 in [0:1)
 for r3 in [0:1)
  for p3 in [0:1)

   for k2 in [0:2)
    for r2 in [0:3)
     for p2 in [0:1)

      spatial_for k1 in [0:16)
       spatial_for r1 in [0:1)
        spatial_for p1 in [0:1)

         for k0 in [0:1)
          for r0 in [0:1)
           for p0 in [0:16)
```

Write:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=1 K=1
    permutation: PRK

  - target: GlobalBuffer
    type: temporal
    factors: R=3 P=1 K=2
    permutation: PRK

  - target: GlobalBuffer
    type: spatial
    factors: R=1 P=1 K=16
    permutation: PRK

  - target: RegisterFile
    type: temporal
    factors: R=1 P=16 K=1
    permutation: PRK
```

EXAMPLE 4: SPATIAL INSTANCES

Spatial levels need to be mapped.

By convention, a block of spatial_for loops representing a spatial fanout from storage level *Outer* to storage level *Inner* are described as a spatial mapping directive targeted at level *Outer*.

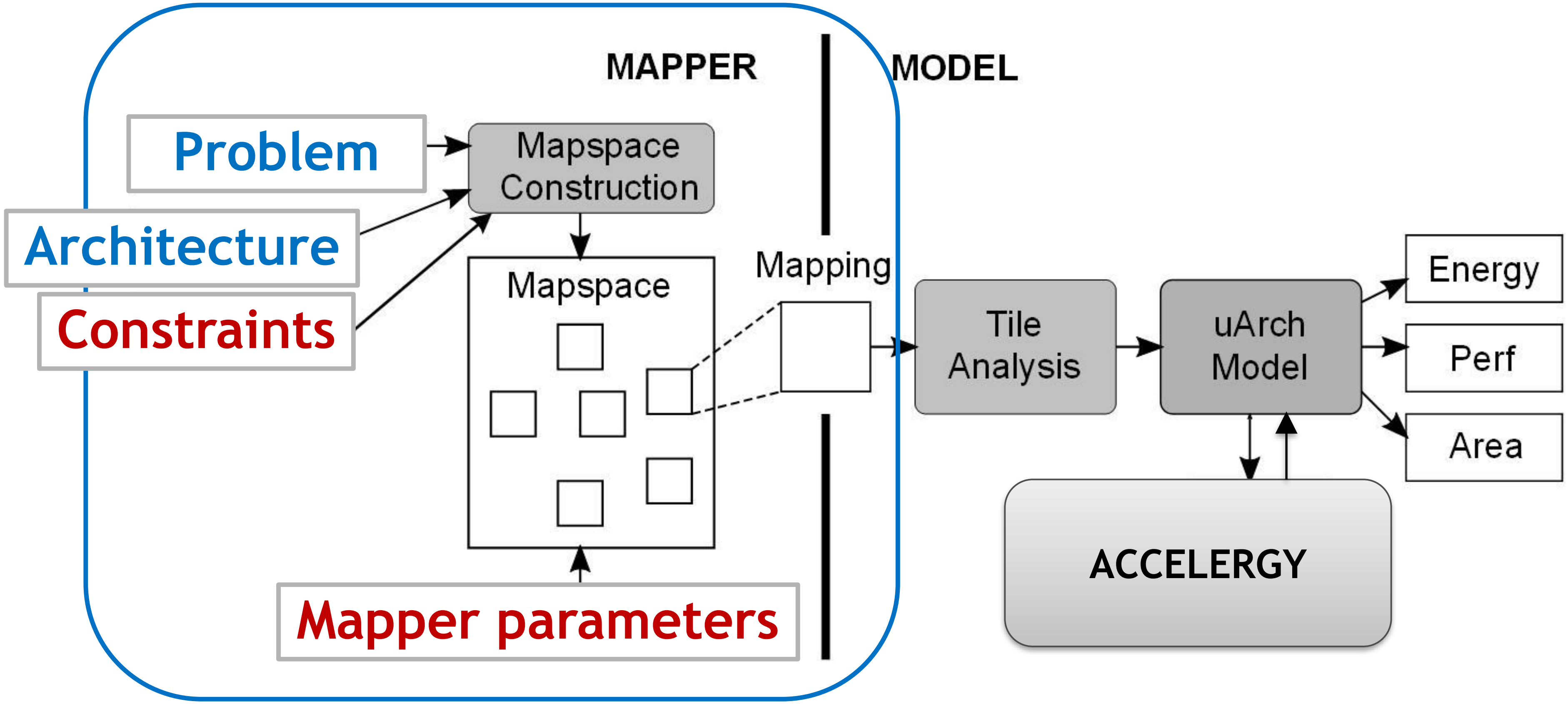
Specifying complete mappings manually is beginning to get tedious. Space of choices and consequences is getting larger. Moving to realistic problem shapes and hardware topologies, we get a combinatorial explosion.

Fortunately, Timeloop's mapper was built exactly for this.

USING TIMELOOP

THE MAPPER

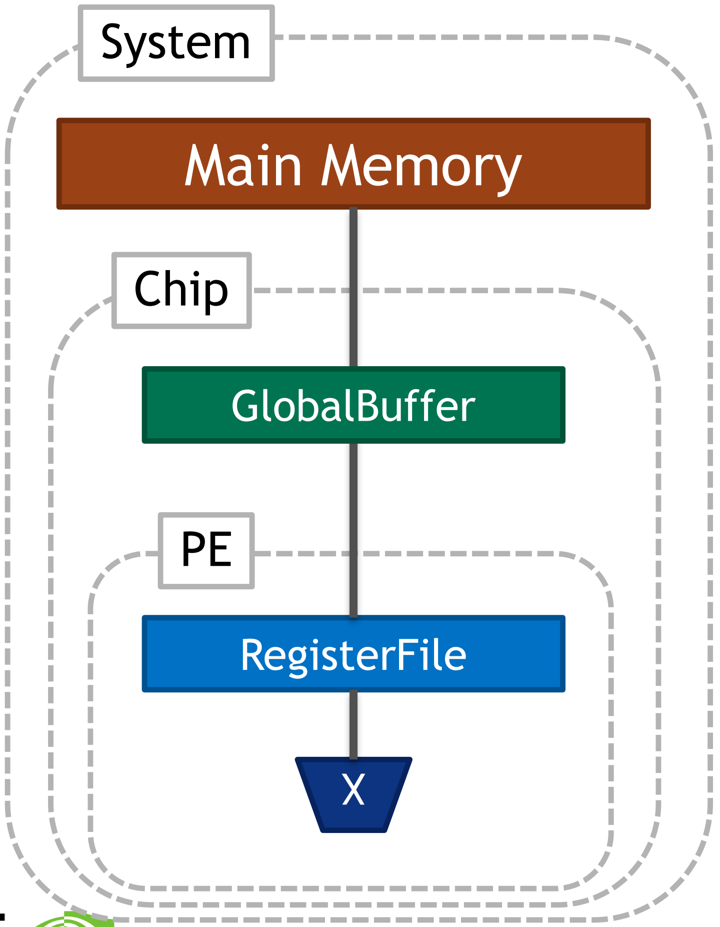
INVOKING THE MAPPER



To understand how the mapper works, let’s go back to a simpler hardware architecture.

EXAMPLE 5: MAPSPACE

Arch: 3-Level, Problem: 1D + Output Channels



Recall:

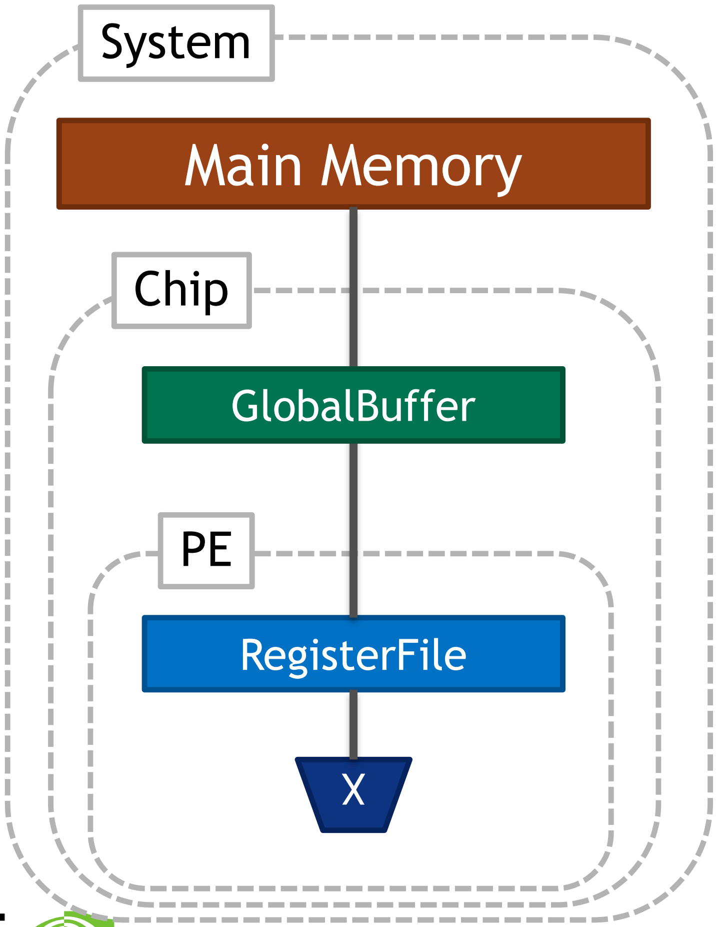
```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=16 K=4
    permutation: RPK

 - target: GlobalBuffer
    type: temporal
    factors: R=3 P=1 K=2
    permutation: RPK

  - target: RegisterFile
    type: temporal
    factors: R=1 P=1 K=4
    permutation: RPK
```

Mapper constructs a mapping template:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

 - target: GlobalBuffer
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

  - target: RegisterFile
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _
```

EXAMPLE 5: MAPSPACE

Arch: 3-Level, Problem: 1D + Output Channels



Mapspace: An enumeration of ways to fill in these _ red blanks:

- Factors
- Permutations
- Dataspace Bypass*

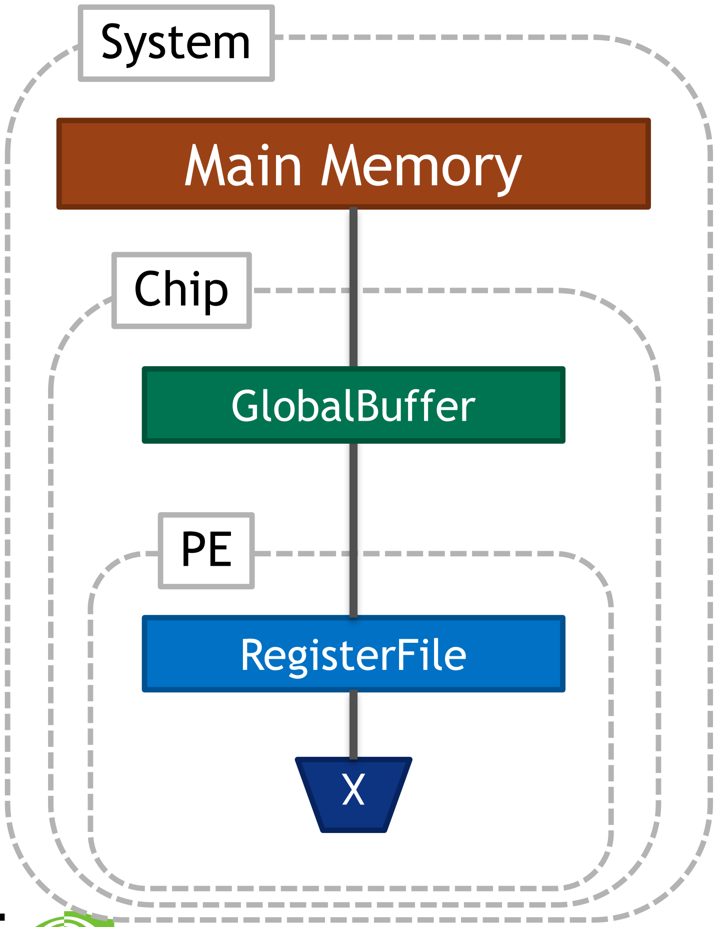
* = not shown in example

Mapper constructs a mapping template:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

  - target: GlobalBuffer
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

  - target: RegisterFile
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _
```

EXAMPLE 5: MAPSPACE

Arch: 3-Level, Problem: 1D + Output Channels



Mapspace: An enumeration of ways to fill in these _ red blanks:

- Factors
- Permutations
- Dataspace Bypass

Mapspaces can be **constrained** by the user.

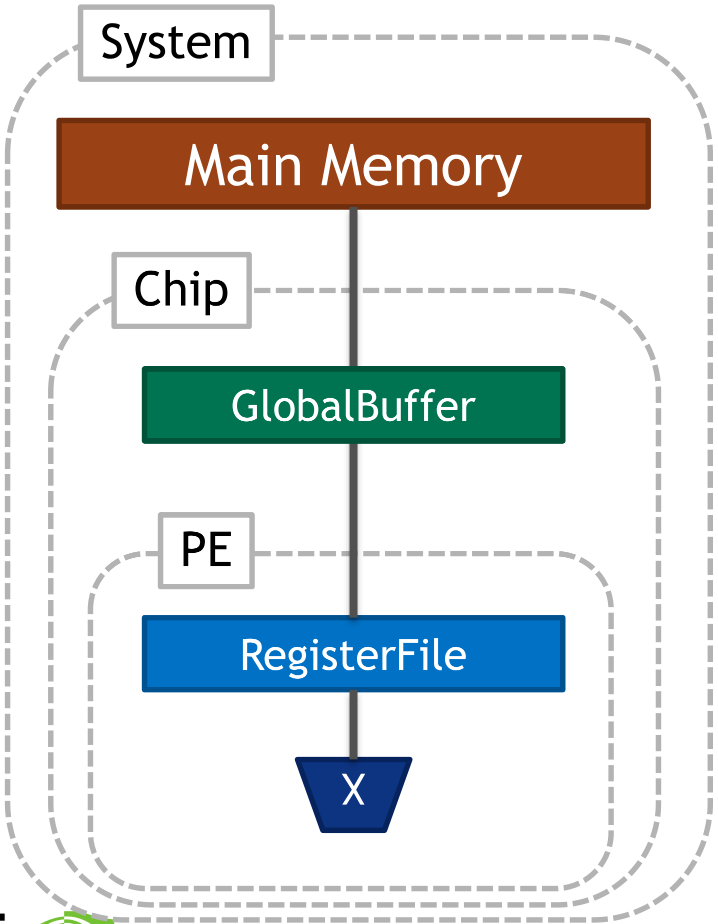
- Architecture constraints
- Mapspace constraints

Mapper constructs a mapping template:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

 - target: GlobalBuffer
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

  - target: RegisterFile
    type: temporal
    factors: R=_ P=1 K=1
    permutation: R _ _
```

EXAMPLE 5: MAPSPACE

Arch: 3-Level, Problem: 1D + Output Channels



Mapspace: An enumeration of ways to fill in these _ red blanks:
- Factors
- Permutations
- Dataspace Bypass

Mapspaces can be **constrained** by the user.
- Architecture constraints
- Mapspace constraints

Mapper runs a search heuristic over the constrained mapspace

Mapper constructs a mapping template:

```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

 - target: GlobalBuffer
    type: temporal
    factors: R=_ P=_ K=_
    permutation: _ _ _

  - target: RegisterFile
    type: temporal
    factors: R=_ P=1 K=1
    permutation: R _ _
```

TUNING THE MAPPER'S SEARCH

Search heuristics (as of this recording)

- Linear
- Random
- Hybrid

Optimization criteria: prioritized list of statistics emitted by the model, e.g.,

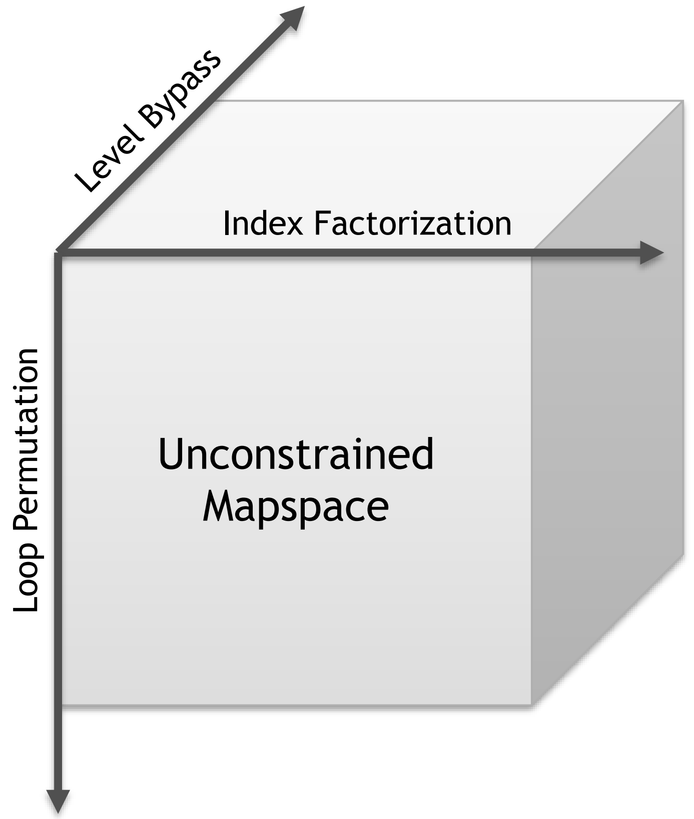
- [cycles, energy]
- [last-level-accesses]

Termination conditions

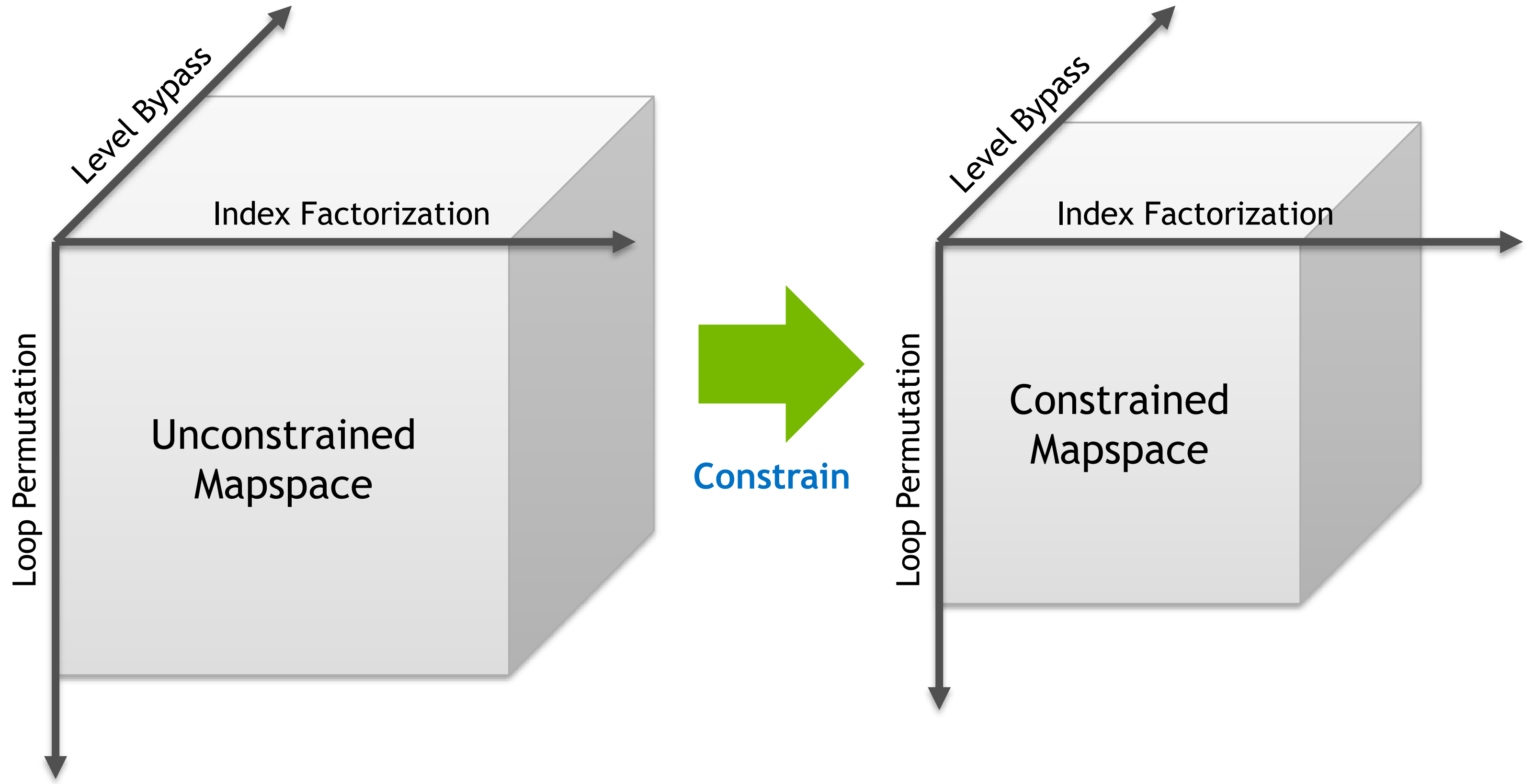
- Mapspace exhausted
- #Valid mappings encountered >= "search-size"
- #Consecutive invalid mappings encountered >= "timeout"
- #Consecutive sub-optimal valid mappings encountered >= "victory-condition"
- Ctrl+C

DEEP DIVE: UNDERSTANDING THE MAPPER

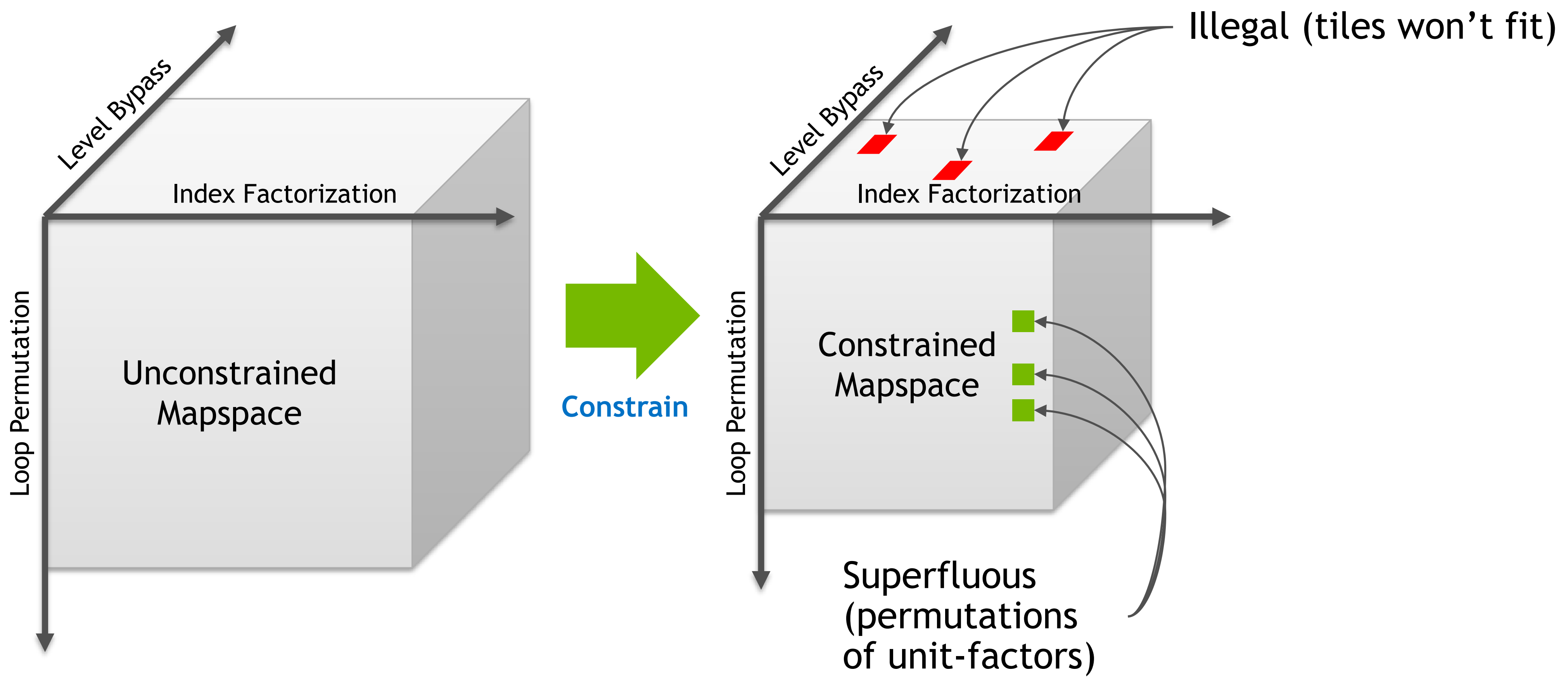
CONSTRAINING A MAPSPACE



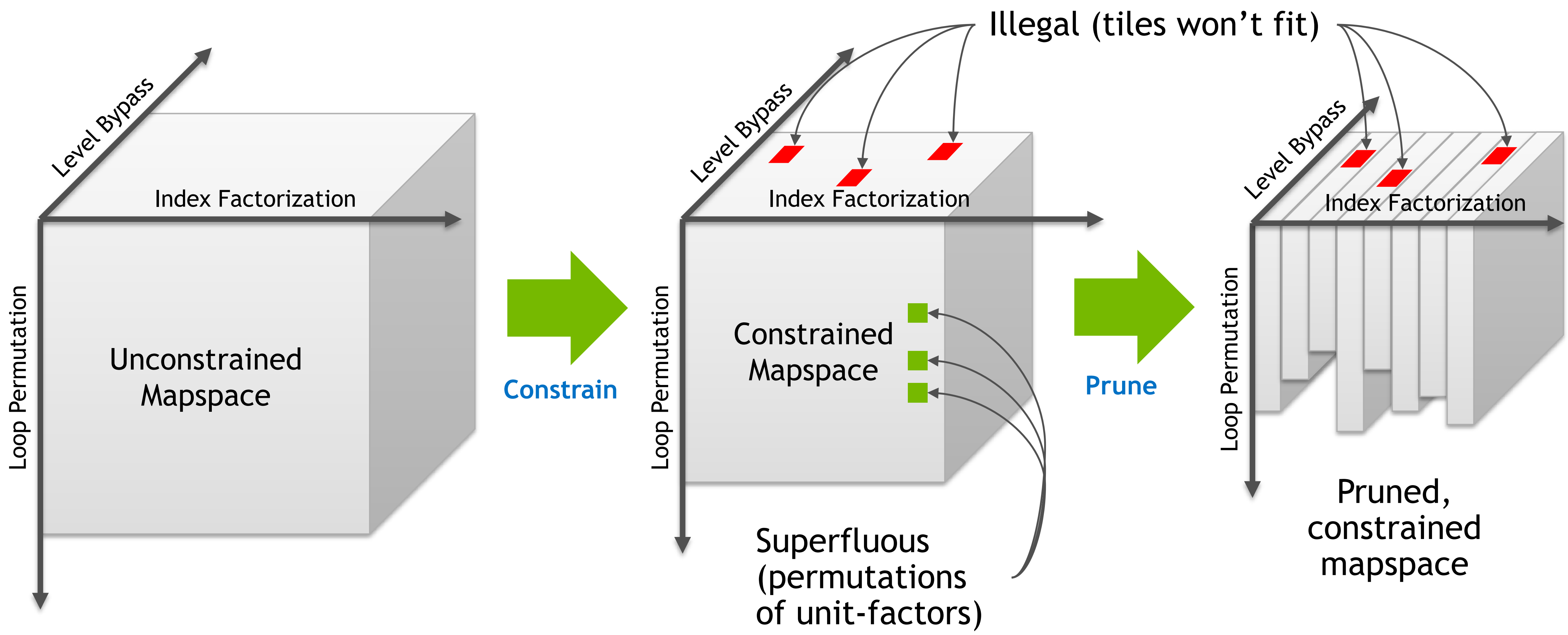
CONSTRAINING A MAPSPACE



PRUNING A MAPSPACE



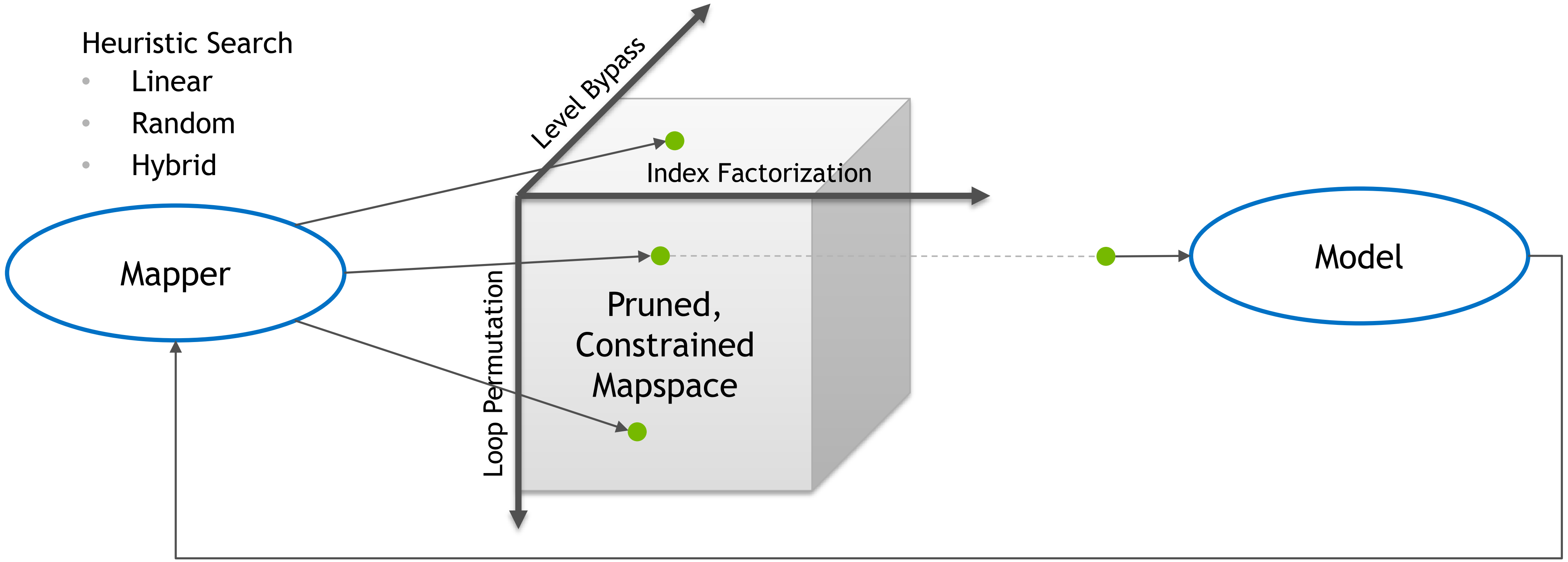
PRUNING A MAPSPACE



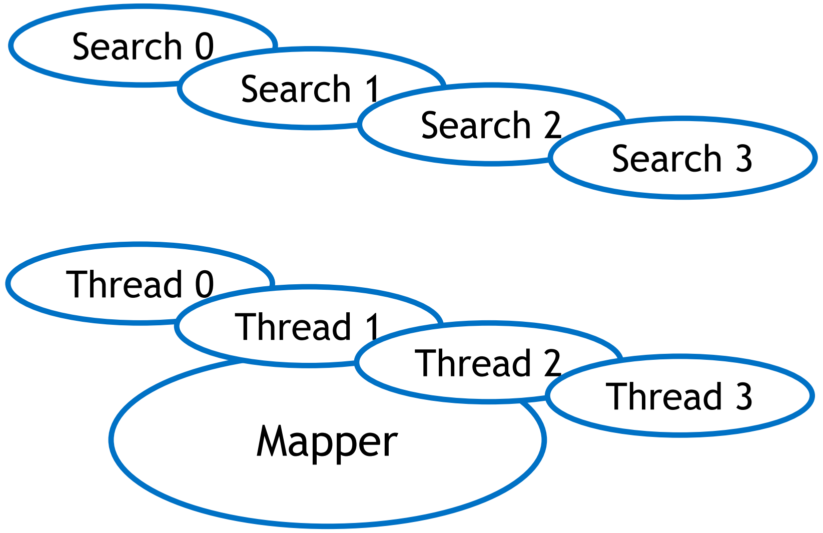
THE MAPPER

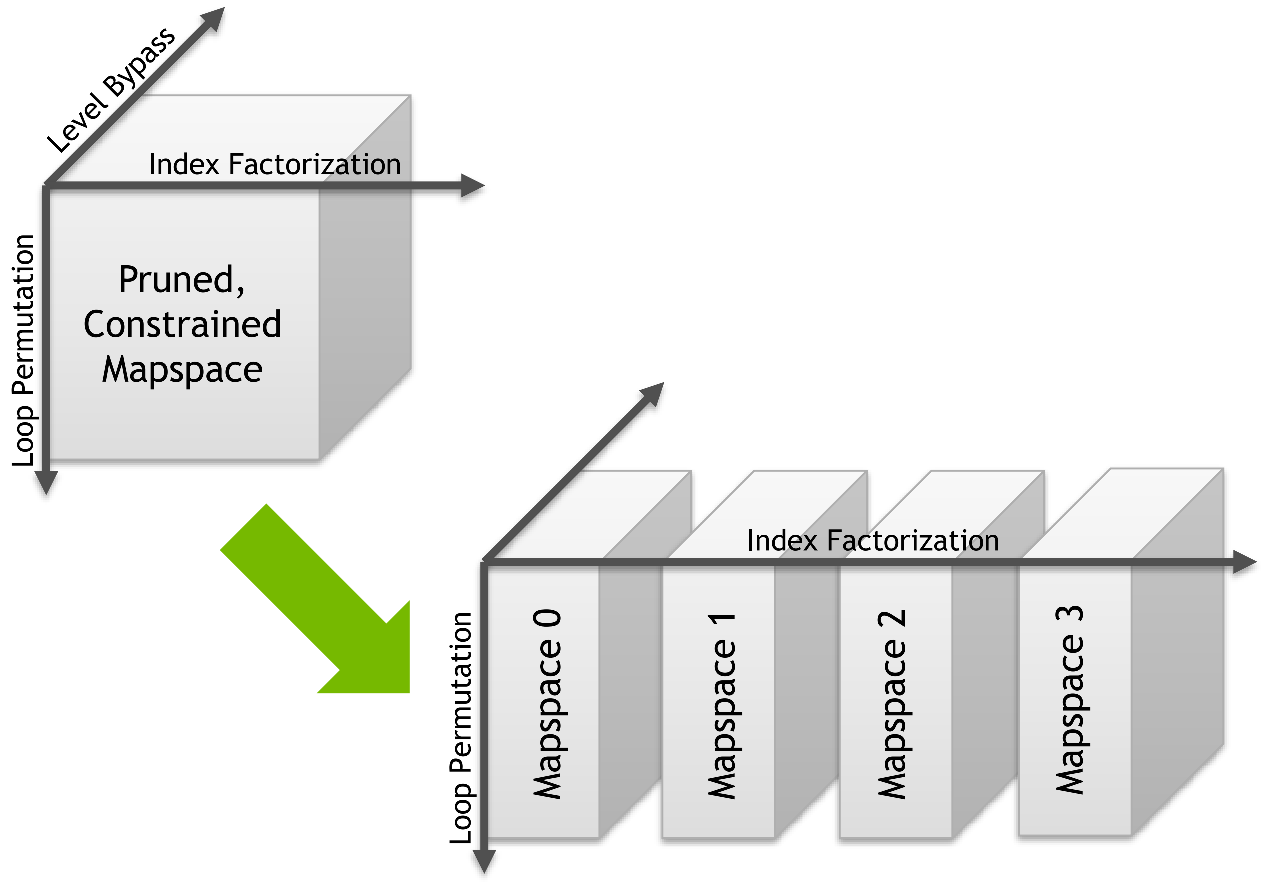
Heuristic Search

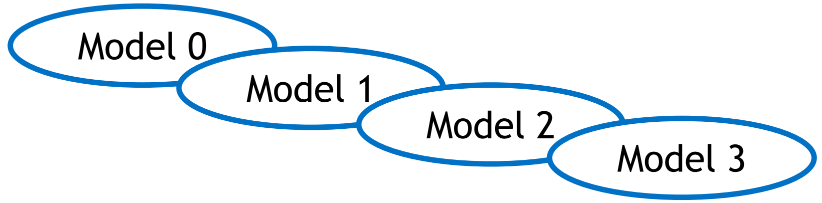
- Linear
- Random
- Hybrid



MULTI-THREADING



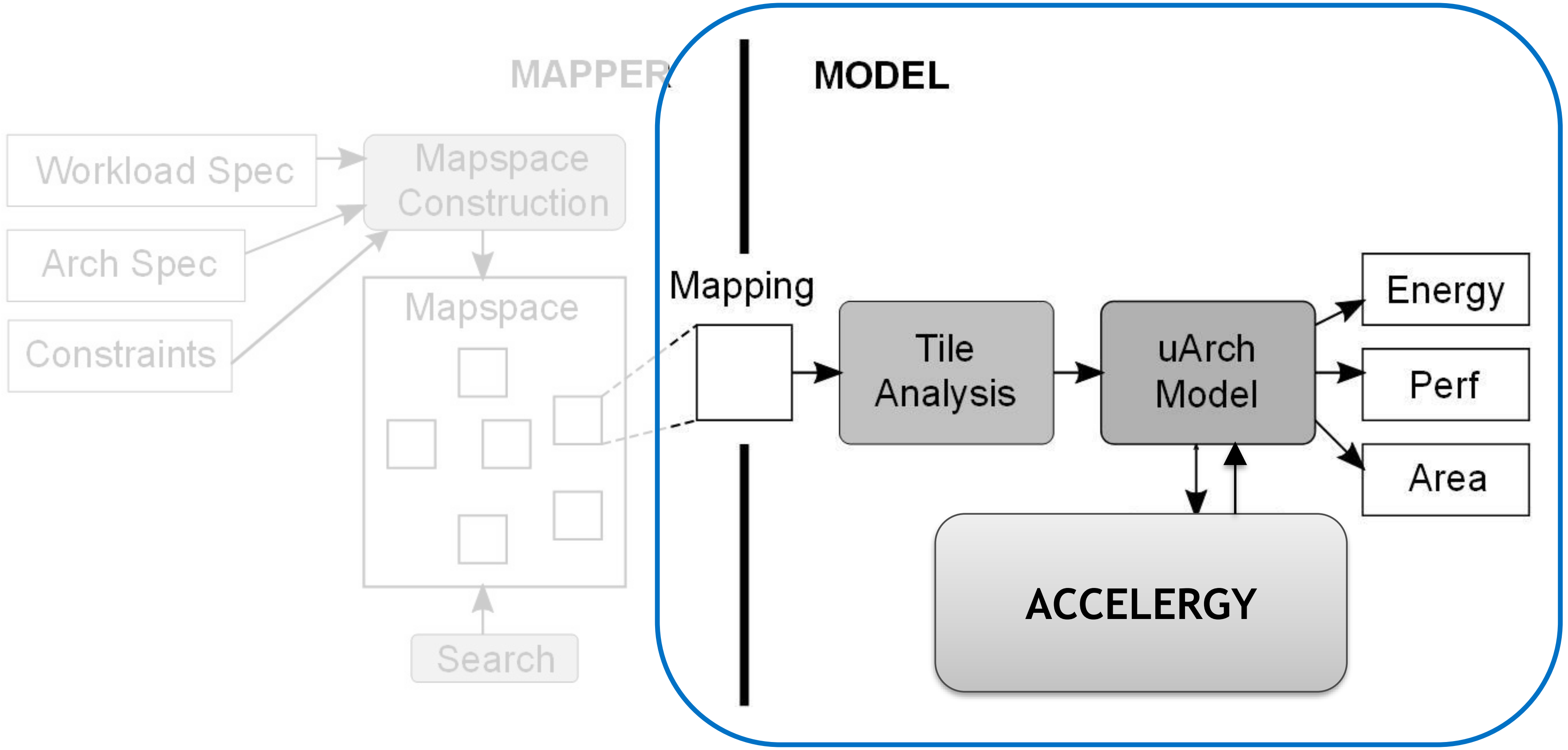




Can also split along other dimensions

DEEP DIVE: UNDERSTANDING THE MODEL

THE MODEL


MAPPER
MODEL
Workload Spec
Arch Spec
Constraints
Mapspace Construction
Mapspace
Search
Mapping
Tile Analysis
uArch Model
Energy
Perf
Area
ACCELERGY

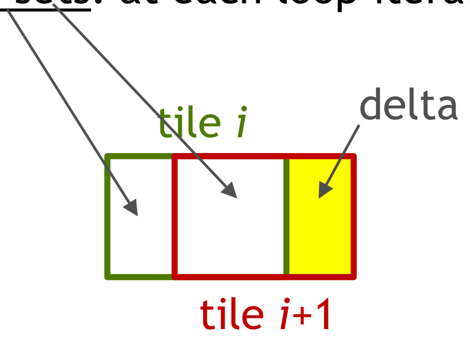
THE MODEL: TILE ANALYSIS

Mapping

```
for …
  for p=[0:8)
    for …
      for …
```

Determines tiles, spatial partitioning of tiles, and schedule

Point sets: at each loop iteration (time step OR instance of a hardware unit)



Deltas: set-difference between point sets

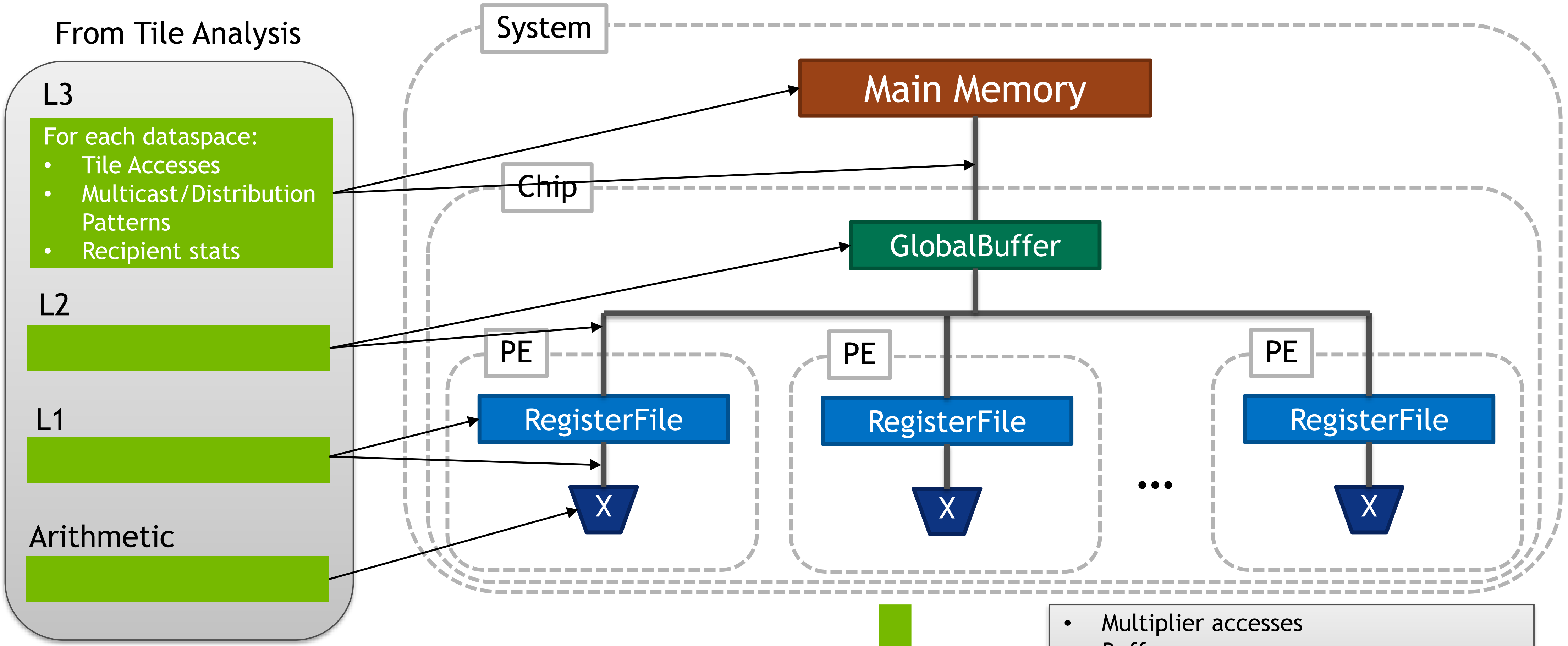
- Temporal: Indicates stationarity, sliding-window behavior, etc.
- Spatial: Indicates overlaps/halos between adjacent units, multicasting/forwarding opportunities

Tile Analysis: Measure and record deltas over all space and time.

Naïve but robust approach: *simulate* execution of entire loop nest.

Regular problems:
- Compute 1st, 2nd, and last iterations of each loop
- Point sets are Axis-Aligned Hyper Rectangles (AAHR)

THE MODEL: UARCH MODEL



Derive

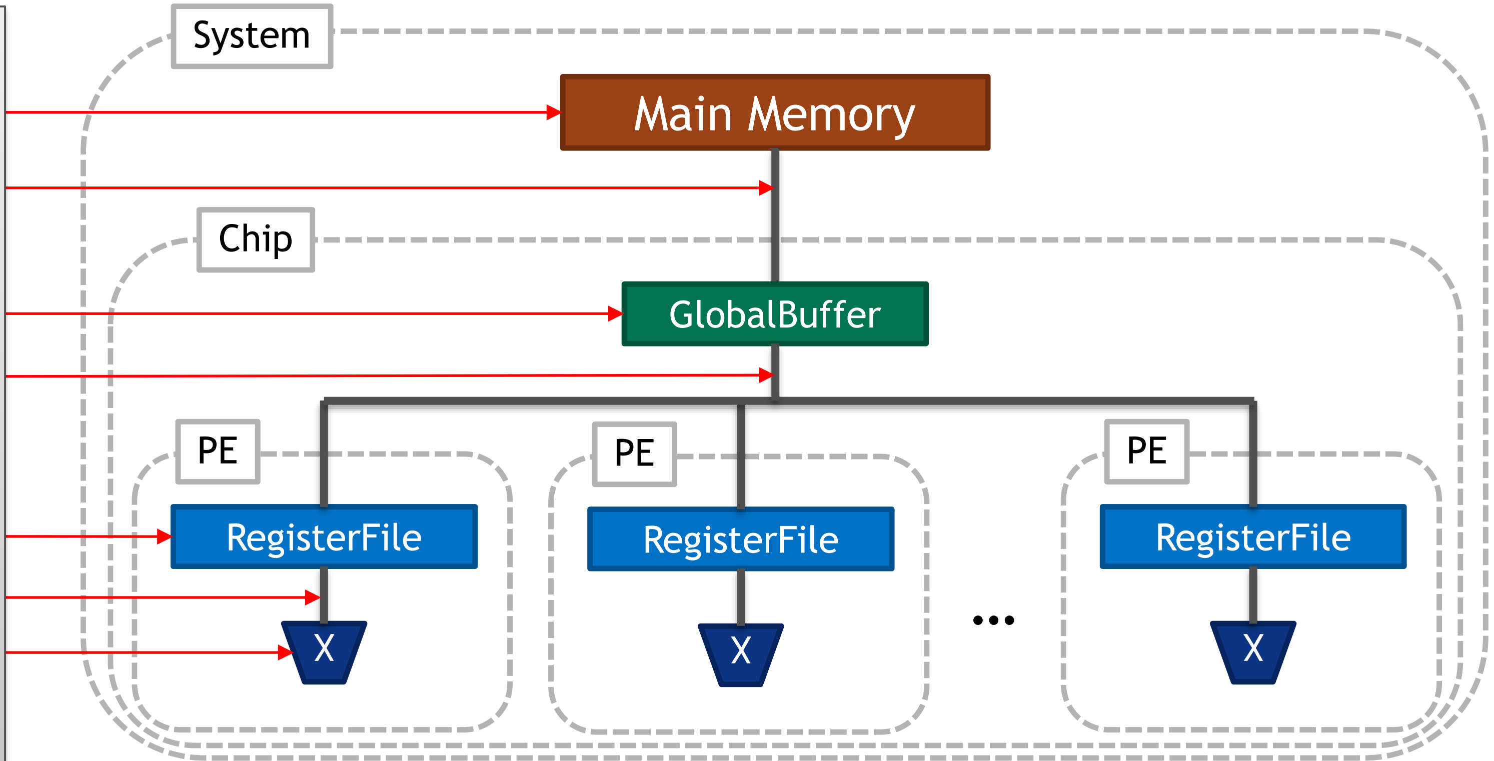
- Multiplier accesses
- Buffer accesses
- Network transfers
- Temporal and spatial accumulations
- Address-generator invocations

ESTIMATING PERFORMANCE AND ENERGY

Performance: Throughput of rate-limiting step across:
- Multipliers
- Buffer read/write ports
- Networks

Assumption: Buffers are either double-buffered or use buffets*

Energy: summation of costs for various activities:
- Multiplier accesses
- Buffer accesses
- Network transfers
- Temporal and spatial accumulations
- Address-generator in



What are the per-activity costs?

* **Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration;** Michael Pellauer, Yakun Sophia Shao, Jason Clemons, Neal Crago, Kartik Hegde, Rangarajan Venkatesan, Stephen W. Keckler, Christopher W Fletcher, Joel Emer; *ASPLOS 2019*

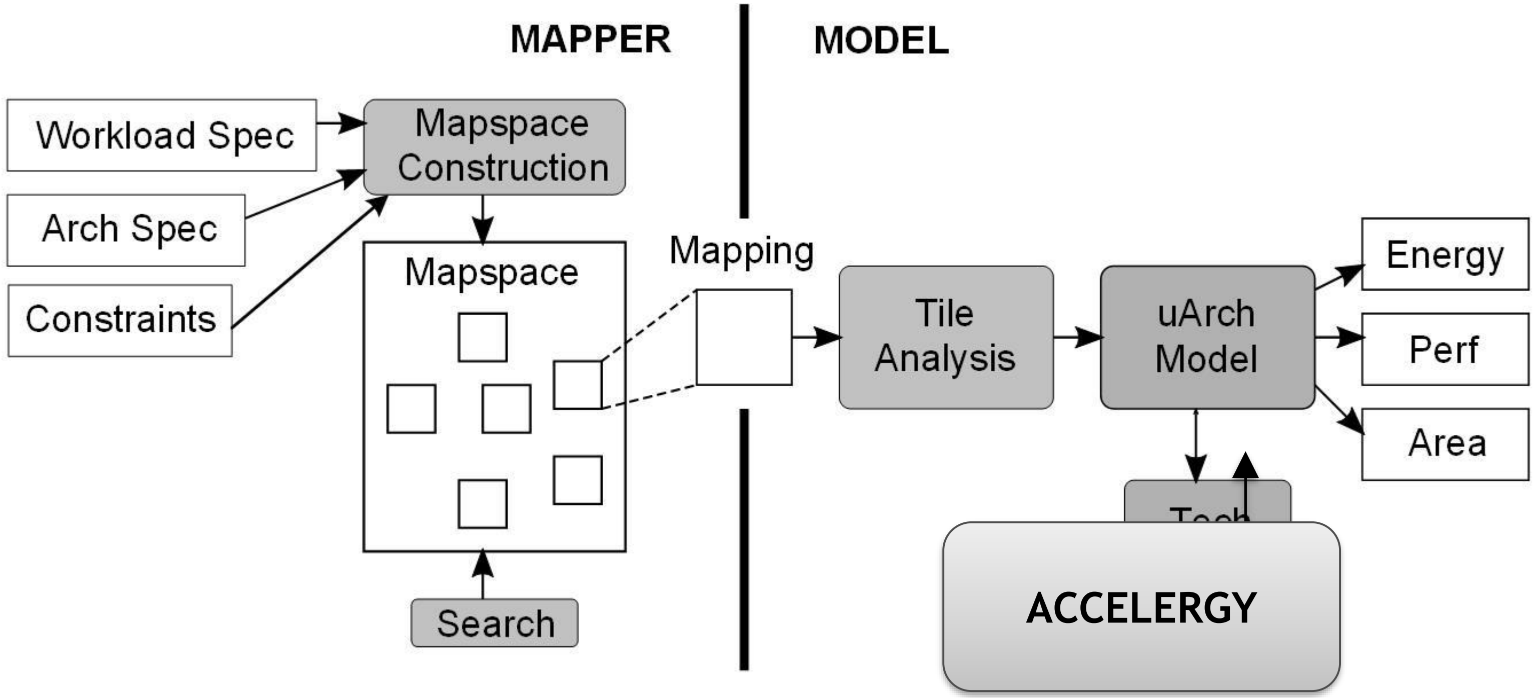
FUTURE WORK

Search Heuristics

Workloads: Complete networks, with inter-layer optimization

Compressed-sparse architectures: modeling fragmentation, load imbalance and metadata overheads

TIMELOOP



Timeloop aims to serve as a vehicle for quality research on flexible DNN accelerator architectures. The infrastructure is released at https://github.com/NVlabs/timeloop under a BSD license.

Please join us in making Timeloop better and more useful for research opportunities across the community.

Resources

- **Tutorial Related**
  - **Tutorial Website: http://accelergy.mit.edu/isca20_tutorial.html**
  - **Tutorial Docker: https://github.com/Accelergy-Project/timeloop-accelergy-tutorial**
    - **Various exercises and example designs and environment setup for the tools**
- **Other**
  - **Infrastructure Docker: https://github.com/Accelergy-Project/accelergy-timeloop-infrastructure**
    - **Pure environment setup for the tools without exercises and example designs**
  - **Open Source Tools**
    - **Accelergy: http://accelergy.mit.edu/**
    - **Timeloop: https://github.com/NVlabs/timeloop**
  - **Papers:**
    - A. Parashar, et al. "Timeloop: A systematic approach to DNN accelerator evaluation," ISPASS, 2019.
    - Y. N. Wu, V. Sze, J. S. Emer, "An Architecture-Level Energy and Area Estimator for Processing-In-Memory Accelerator Designs," *ISPASS,* 2020.
    - Y. N. Wu, J. S. Emer, V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," *ICCAD*, 2019.