PLEASE INSTALL THE DOCKER FOR OUR FUN EXERCISES

Instructions **on our tutorial website:** http://accelergy.mit.edu/tutorial.html

0. If you do not have the docker app installed already, please install docker (community edition)
   (Windows users - please manually turn on virtualization via BIOS settings.)
1. Go to our tutorial repo and get a copy of the single file `docker-compose.yaml`
2. Make an empty directory and place the `docker-compose.yaml` inside.
3. To pull the newest docker image, run command `docker-compose pull`
4. Run command `docker-compose run --rm exercises`
5. Once invoked, you can `cd ./exercises/timeloop/01-conv1d-2level` and `timeloop-model *.yaml` to run a test example.

Change the UID or GID as instructed in the file (if needed)

Updated test: cd ./exercises/timeloop/00-conv1d-1level; timeloop-model */*.yaml

NVIDIA.

TIMELOOP / ACCELERGY TUTORIAL

Joel S. Emer	**NVIDIA, MIT**
Vivienne Sze	MIT
Angshuman Parashar	NVIDIA
Po-An Tsai	NVIDIA
Yakun Sophia Shao	UC Berkley
Yannan Nellie Wu	**MIT**

MICRO 2019

SCHEDULE

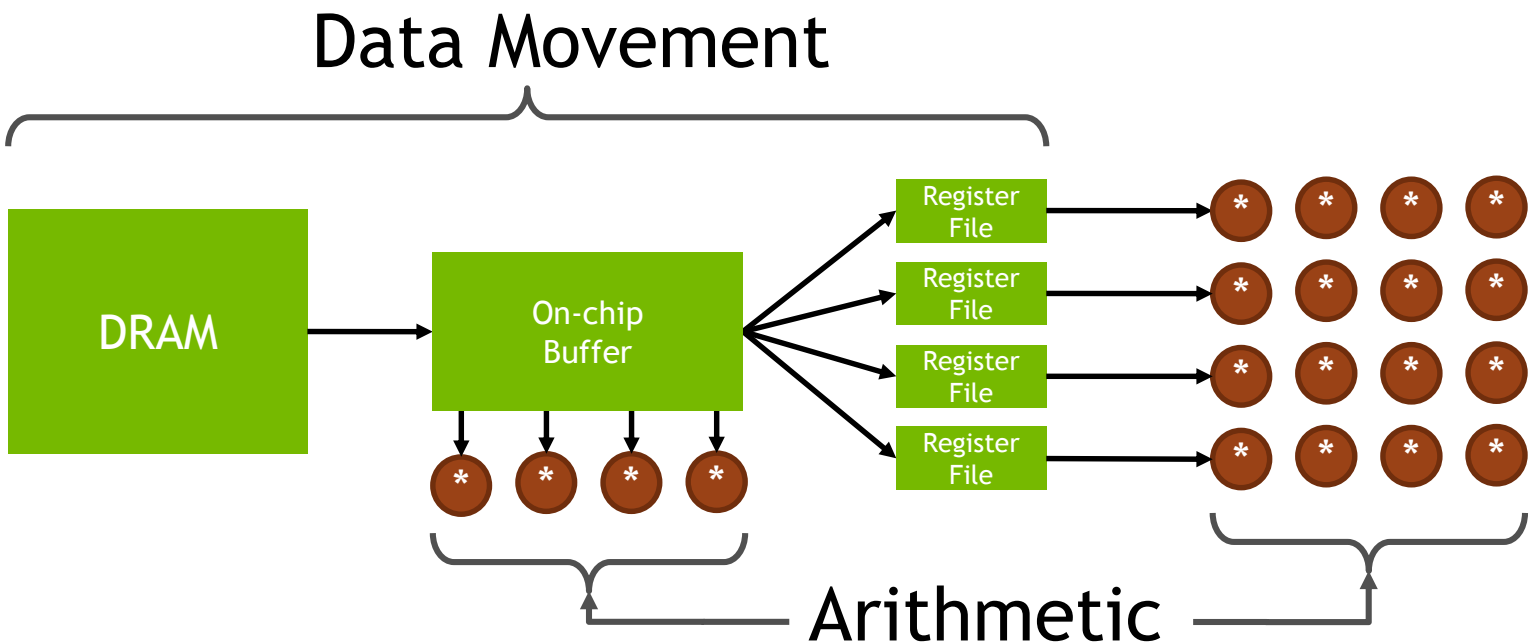
1:00 – 2:30PM	Timeloop lecture + exercises
2:30 – 3:00PM	Timeloop free lab time
3:00 - 3:30PM	*Coffee break*
3:30 – 4:30PM	Accelergy lecture + exercises
4:30 – 5:00PM	Accelergy + Timeloop free lab time

NVIDIA.

MOTIVATION

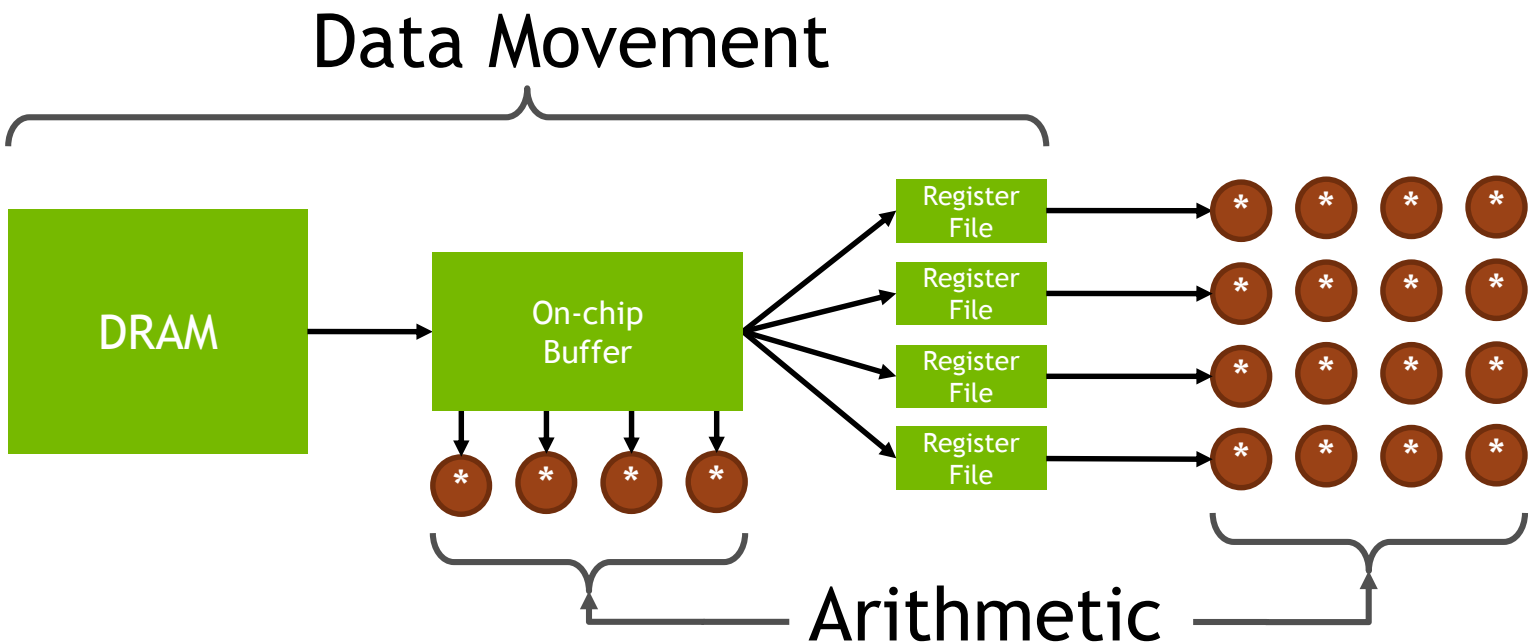
DNN ACCELERATORS

Design Considerations



DNN ACCELERATORS

Design Considerations



DATA MOVEMENT

Why it's important

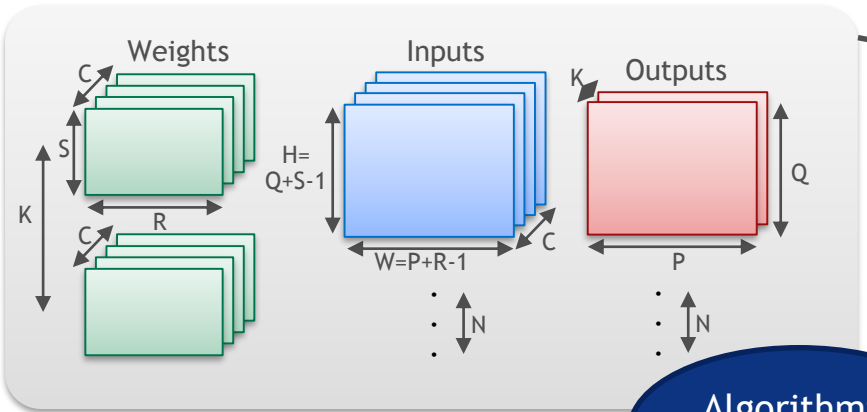
Energy costs	
8-bit Integer Multiply	0.2 pJ
Fetch two 8-bit operands from DRAM	128 pJ
Fetch two 8-bit operands from large SRAM	2 pJ

Fortunately…

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

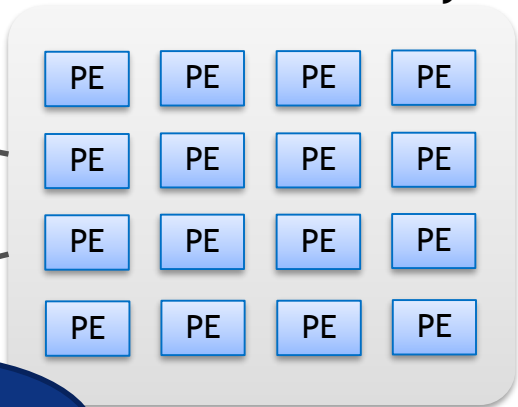
EXPLOITING REUSE

7-dimensional network layer



map

2D hardware array



Algorithmic Reuse

Hardware Reuse

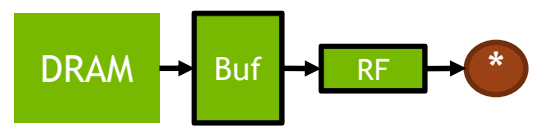
Convolutional Reuse
- Slide filter over input plane

Input Activation Reuse
- Multiple filter blocks over same inputs

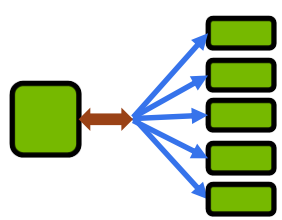
Output Activation Reuse
- Accumulation sum over channels

Batch Reuse
- Re-apply filters to new inputs

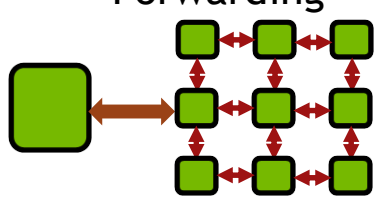
Temporal



Multicast



Forwarding

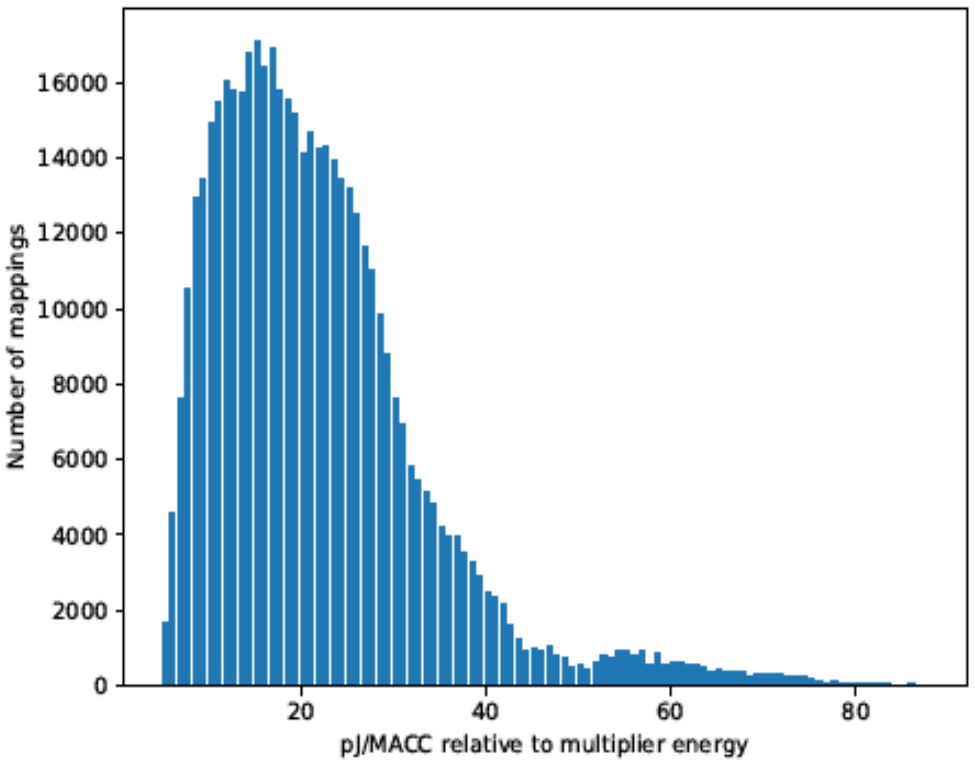


Flexible architectures may allow millions of alternative mappings of a single workload

NVIDIA.

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%

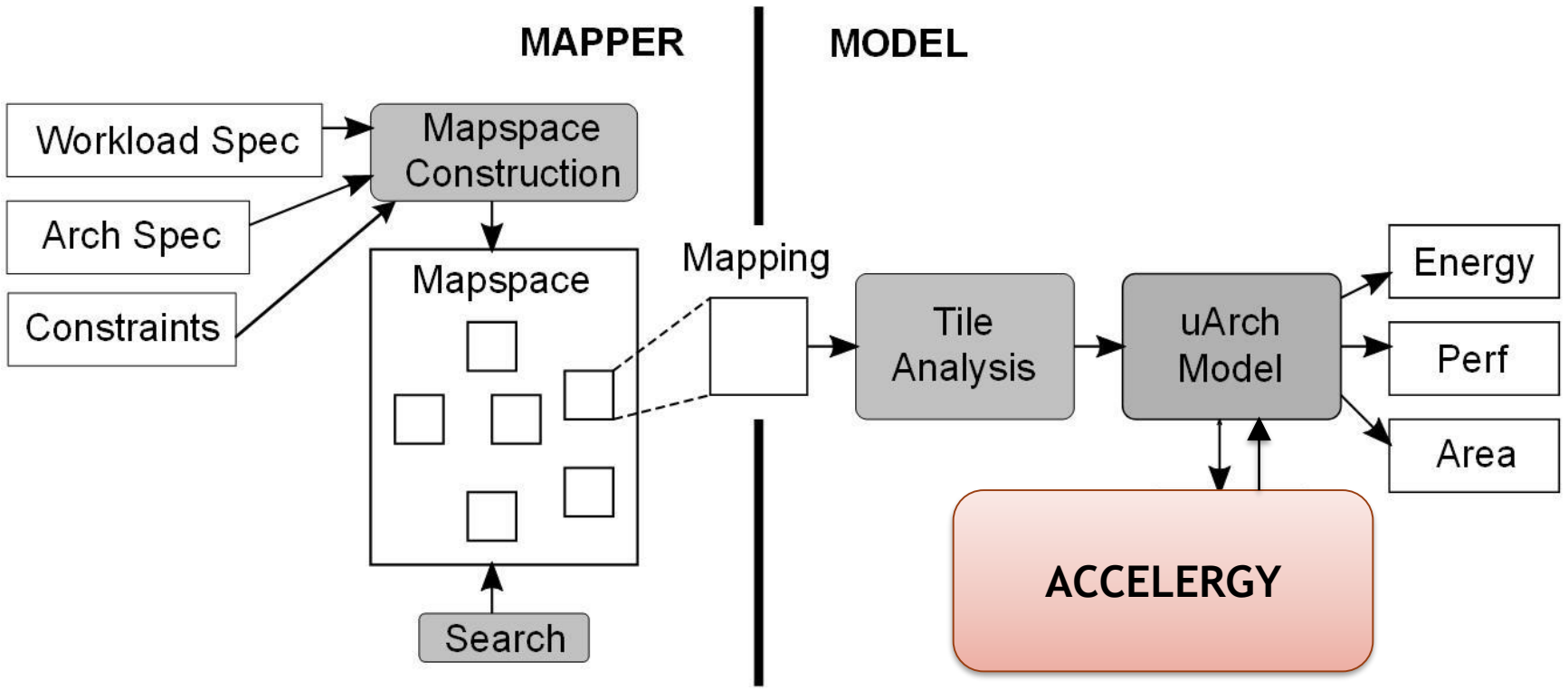
A model needs a mapper to evaluate a DNN workload on an architecture

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

NVIDIA.

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

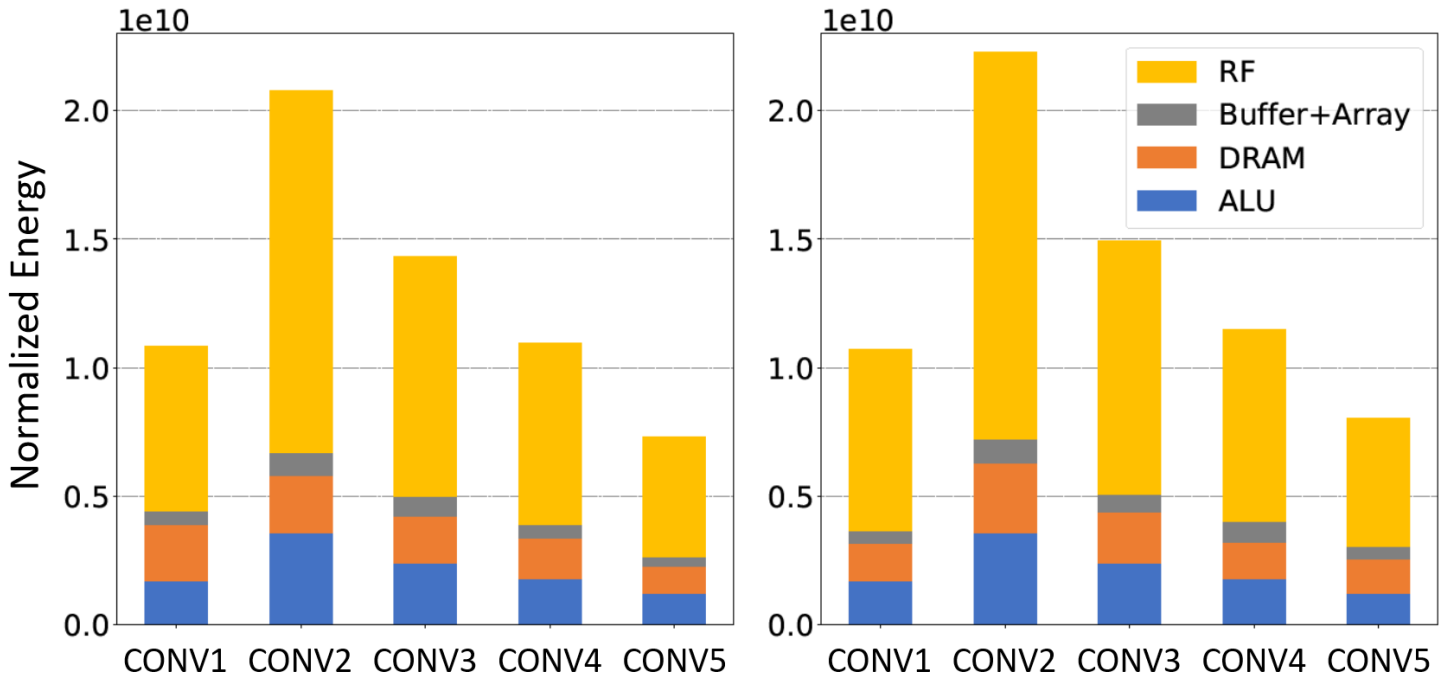
Built-in Mapper (Timeloop)

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

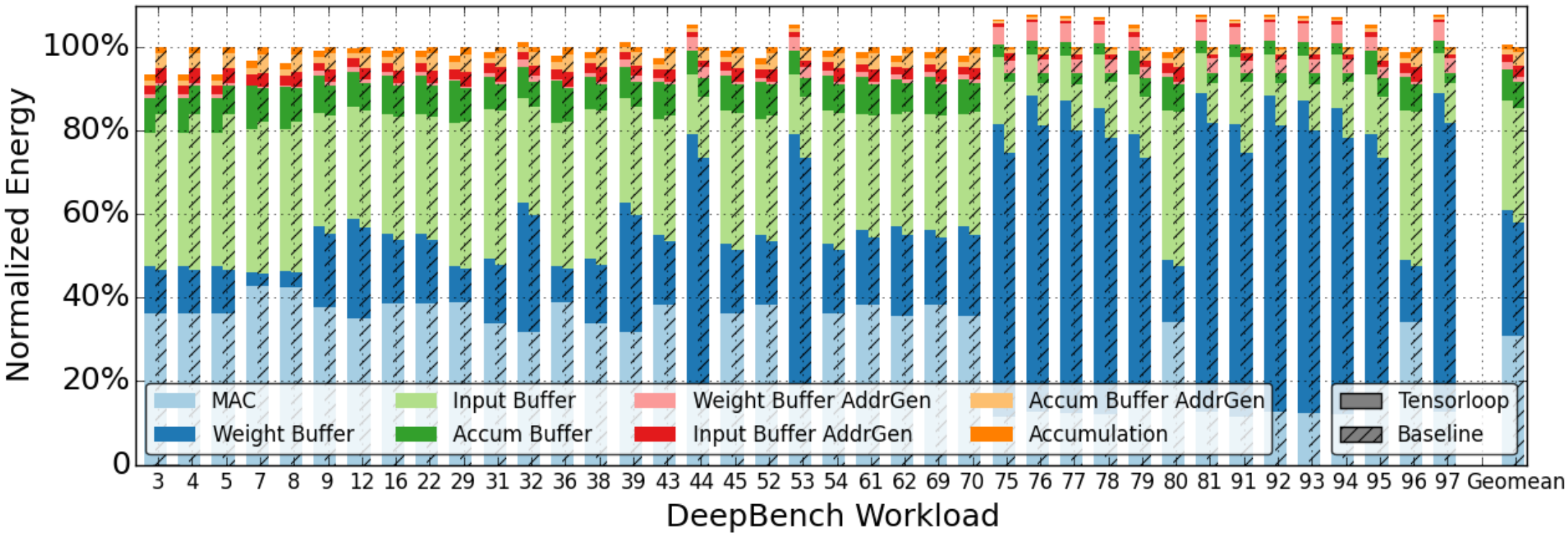

TIMELOOP VALIDATION

VALIDATION: EYERISS

Vs. ISCA 2016 Eyeriss Energy Model

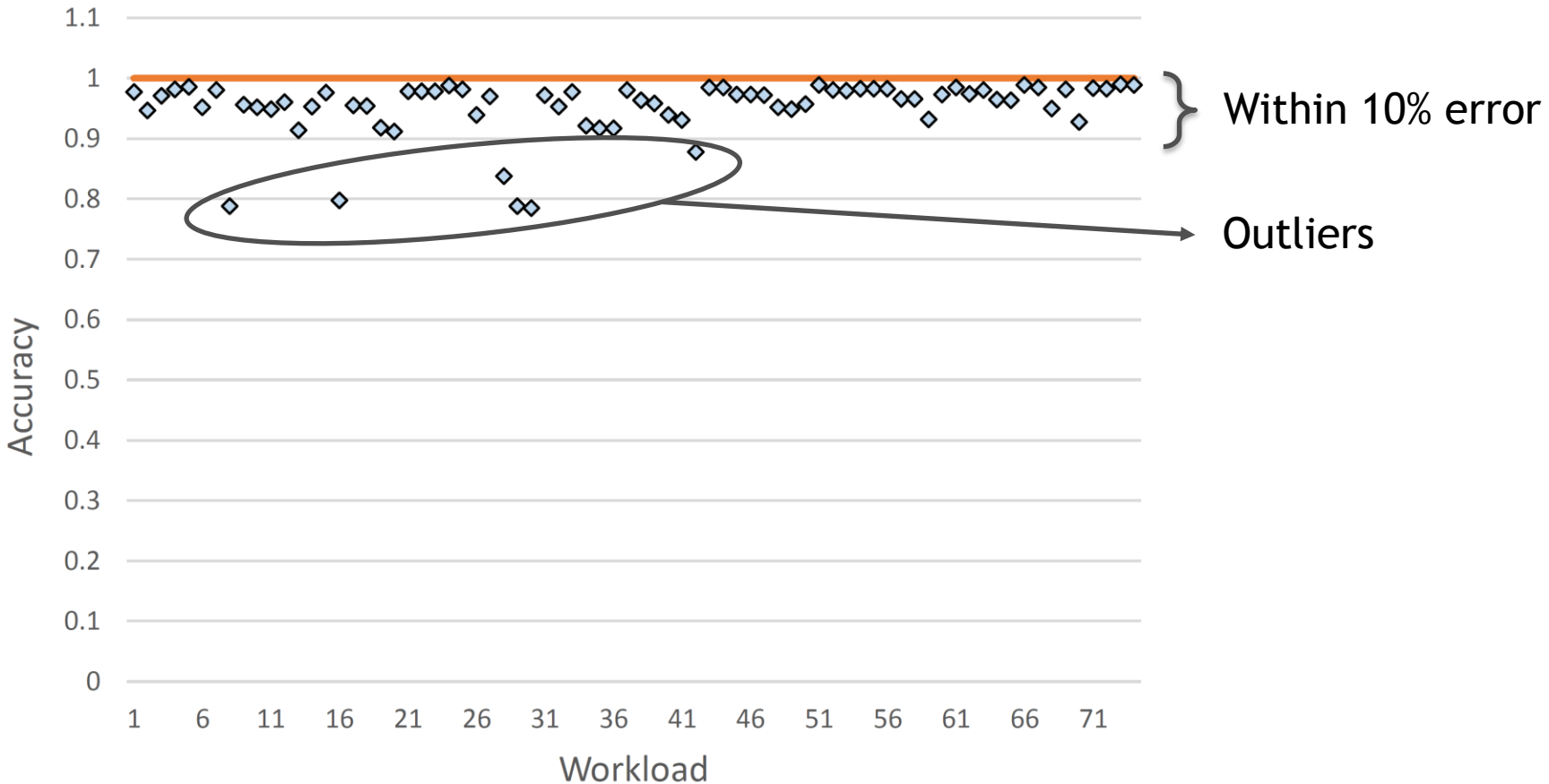


VALIDATION: NVDLA-DERIVED PE (ENERGY)



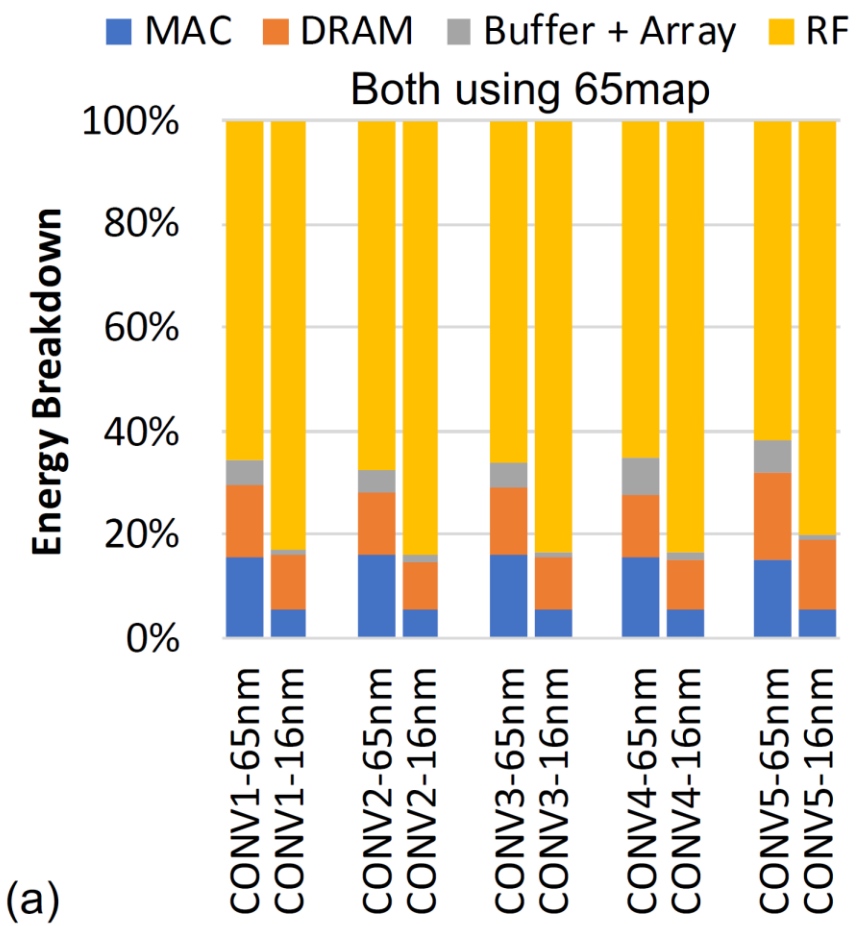
Within 8% error across all workloads

VALIDATION: NVDLA-DERIVED PE (PERFORMANCE)

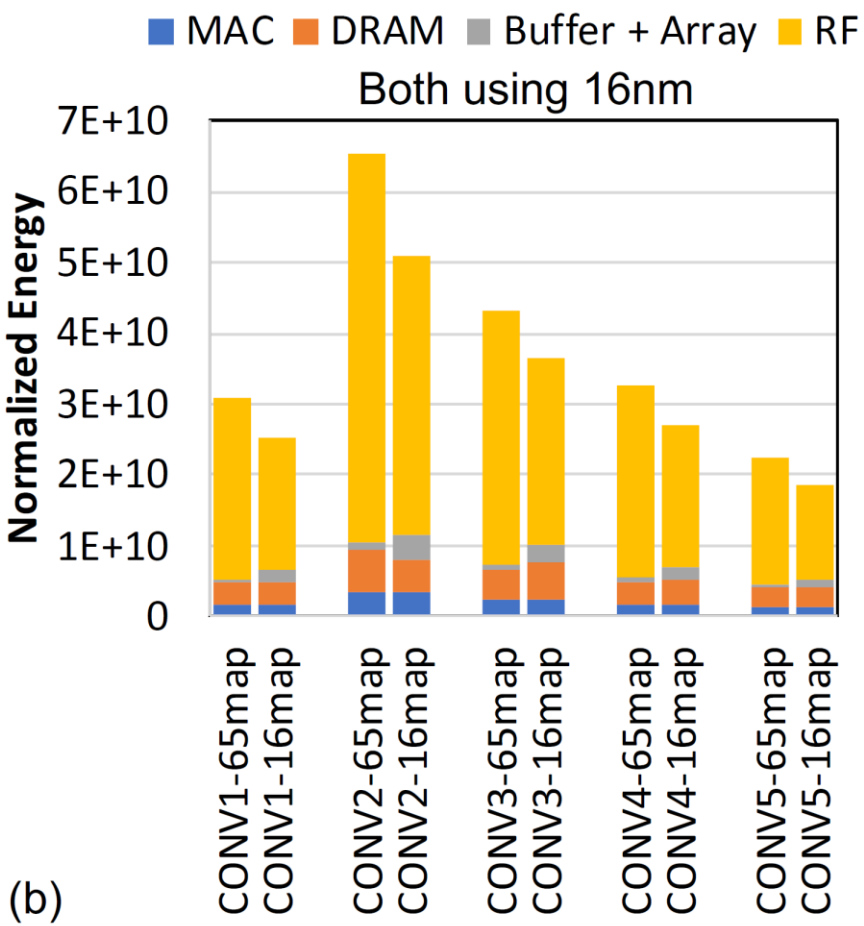


CASE STUDIES

CASE STUDY: TECHNOLOGY MODEL

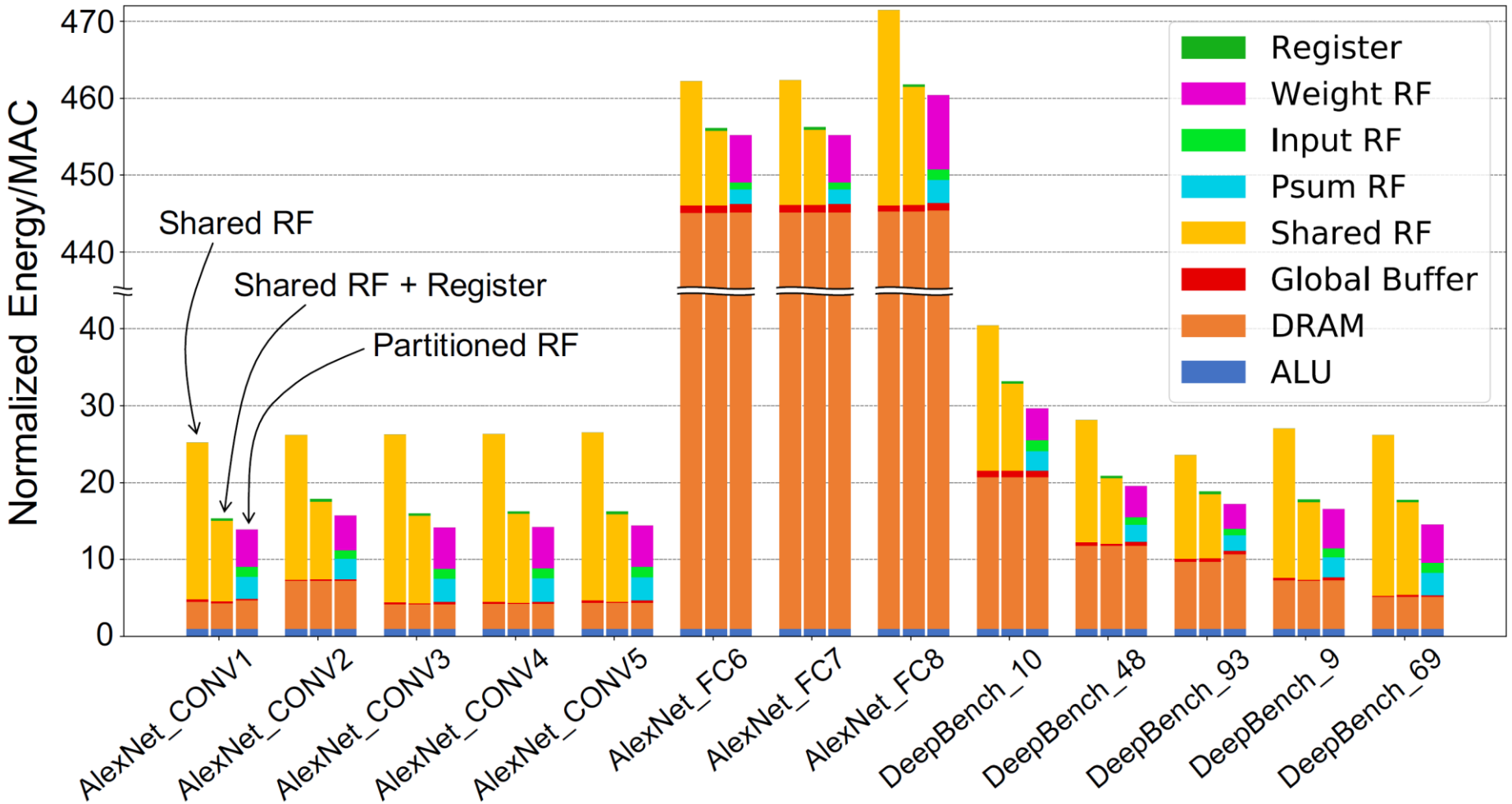


(a)

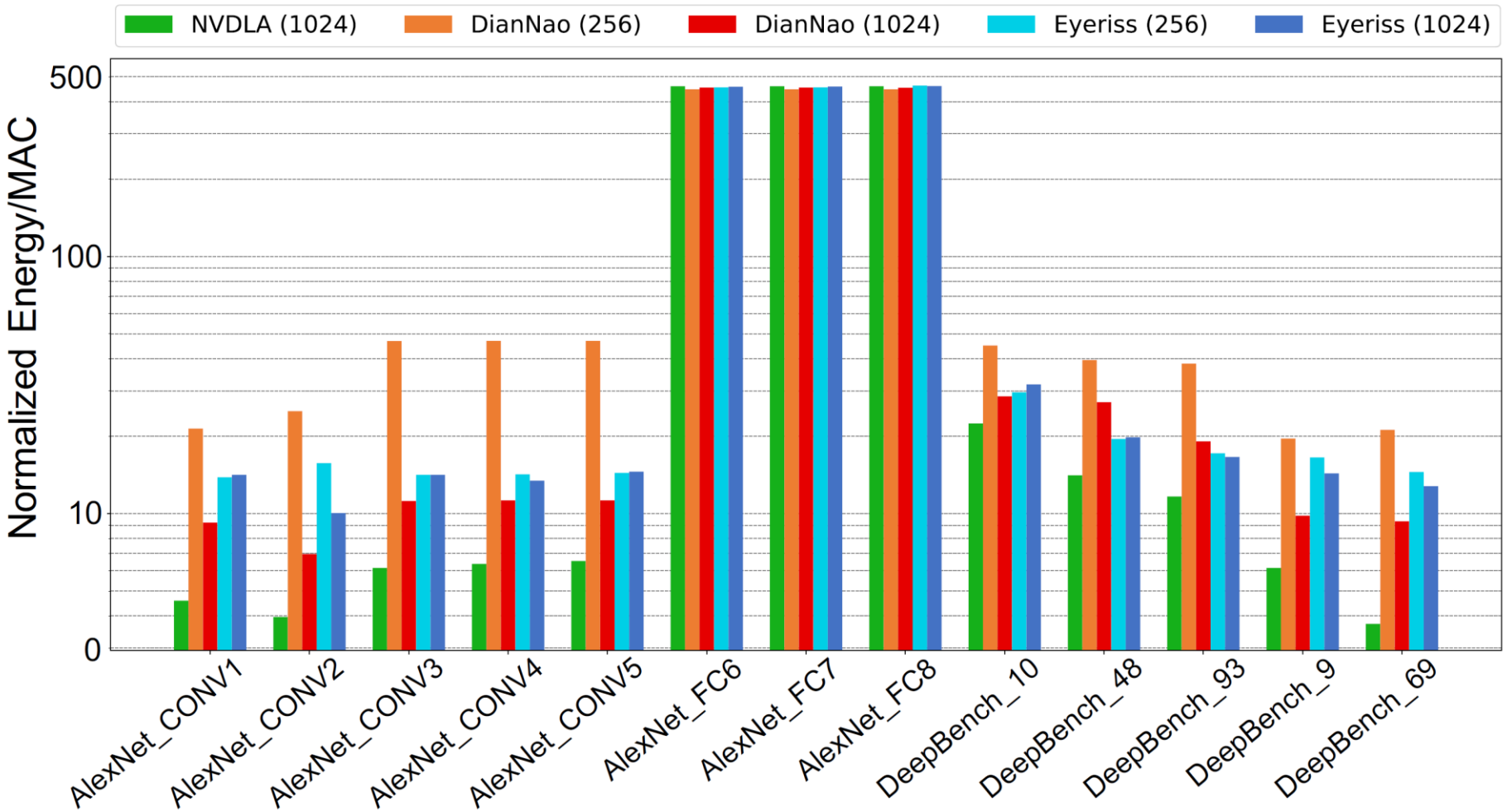


(b)

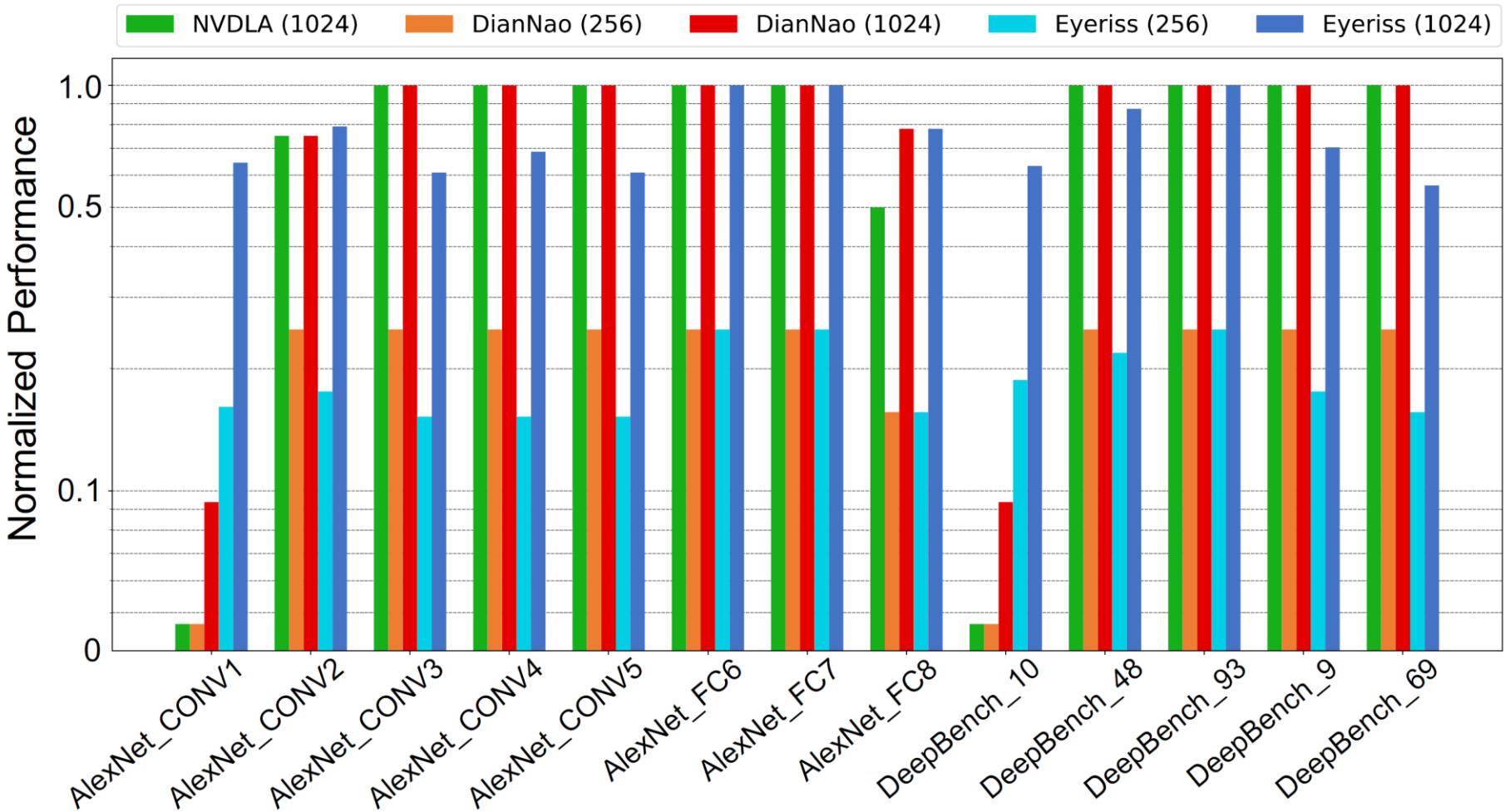
CASE STUDY: MEM HIERARCHY



CASE STUDY: EXISTING DESIGNS



CASE STUDY: EXISTING DESIGNS

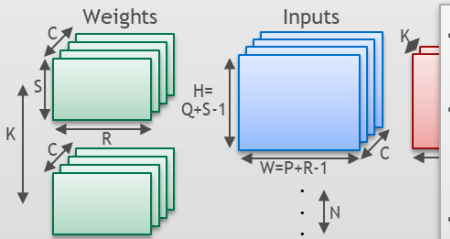



FUN WITH TIMELOOP
THE MODEL

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

```
for c in [0:16)
  for k2 in [0:2)
    for r in [0:3)
      spatial_for k1s in [0:16)
        for p in [0:16)
```

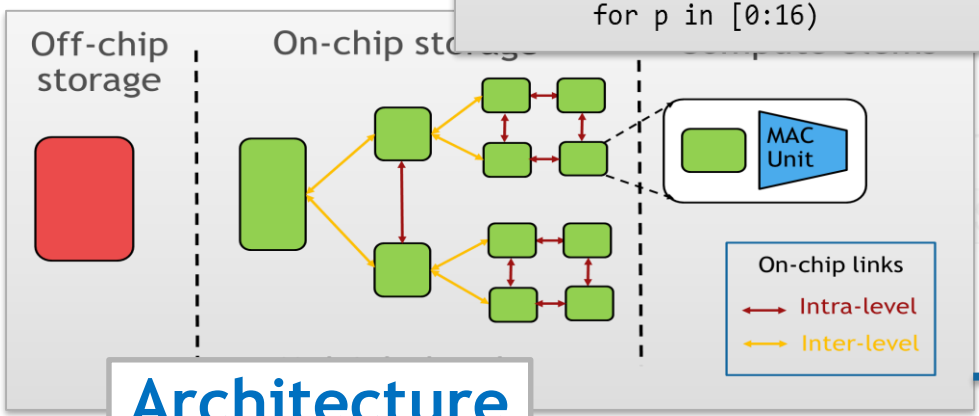
MainMemory

GlobalBuffer

Spatial: GlobalBuffer → RegiserFile

RegisterFile

Architecture



MODEL

Tile Analysis → uArch Model → Energy, Perf, Area

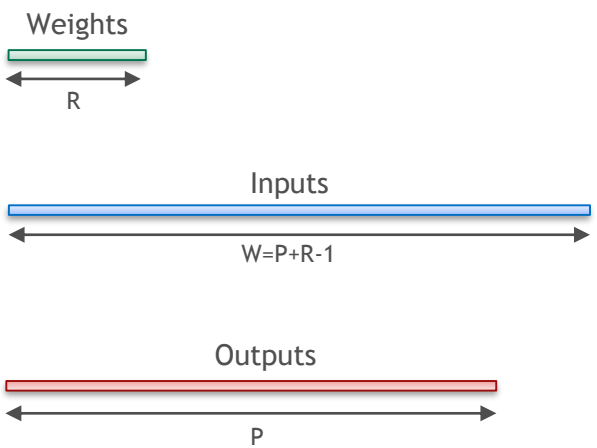
ACCELERGY

EXERCISE 0: PROBLEM

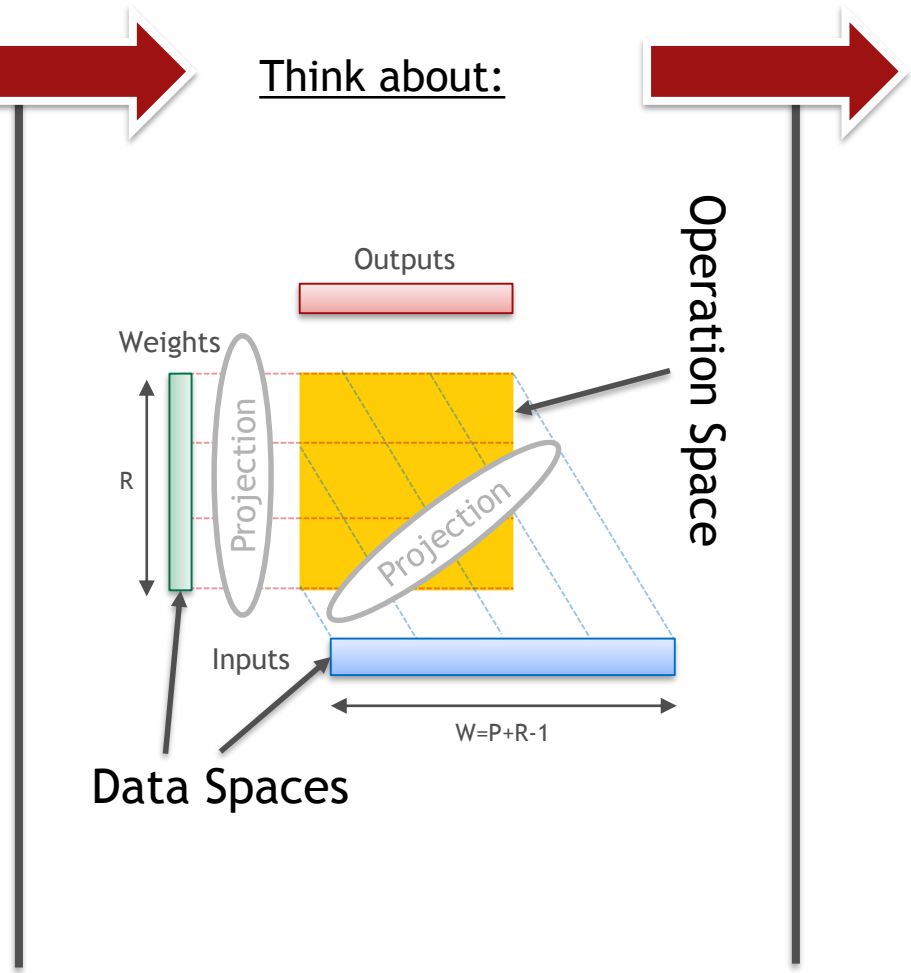
Conv1D

To represent this…

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



Think about:



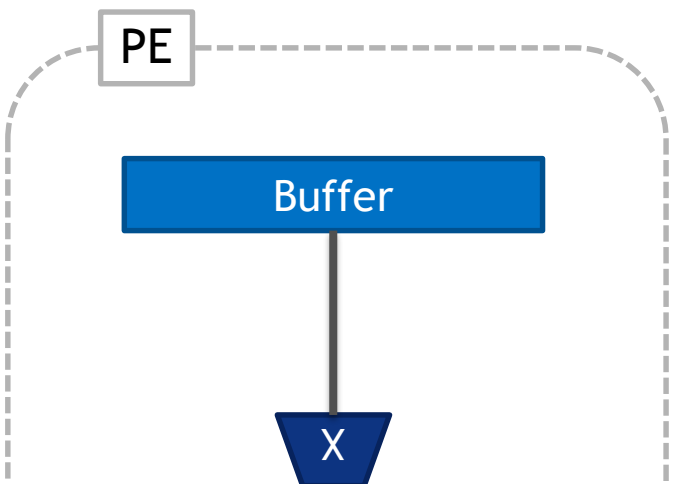
And write:

```
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
```

EXERCISE 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
```

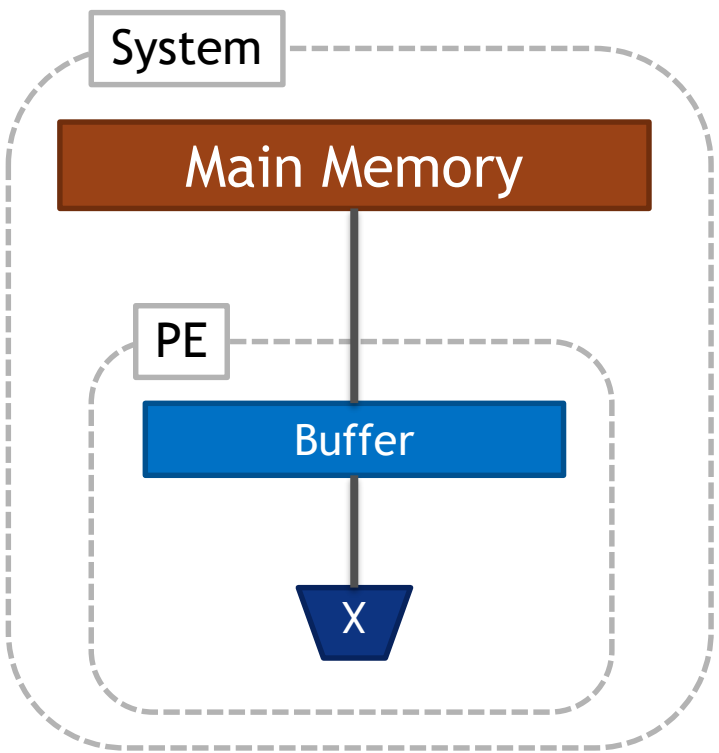
EXERCISE 0

Follow the instructions in the README.

EXERCISE 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
```

EXERCISE 1: MAPPING

Weight Stationary

To represent this...

Write:

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

    Buffer

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

Expected outputs

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

```
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
```

EXERCISE 1: MAPPING

Output Stationary

To represent this...

Write:

```
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];
```

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

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

Expected outputs

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

EXERCISE 1

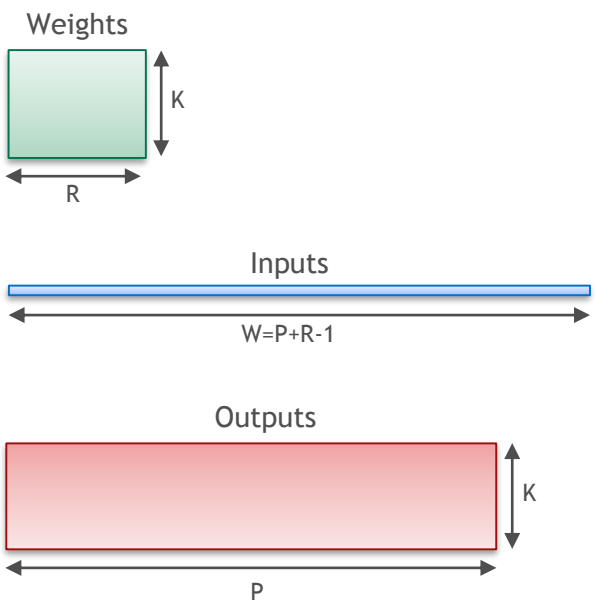
Follow the directions in the README.

EXERCISE 2: PROBLEM

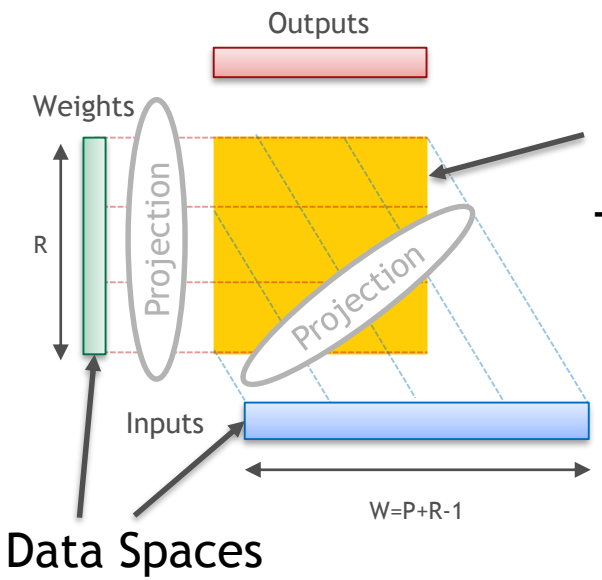
Conv1D + Output Channels

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
```

EXERCISE 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];
```

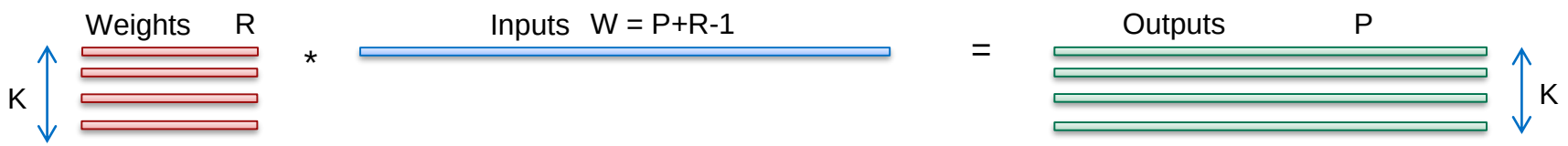
```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=16 K=16
    permutation: RPK

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


NVIDIA.

EXERCISE 2

Follow the directions in the README.

EXERCISE 2: O.S. DATAFLOW VARIANTS



Alg. Min. Backing Store Accesses

Weights	Inputs	Outputs
KR	W	KP

Buffer Utilization
(= minimum Buffer size)

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

Backing Store 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

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

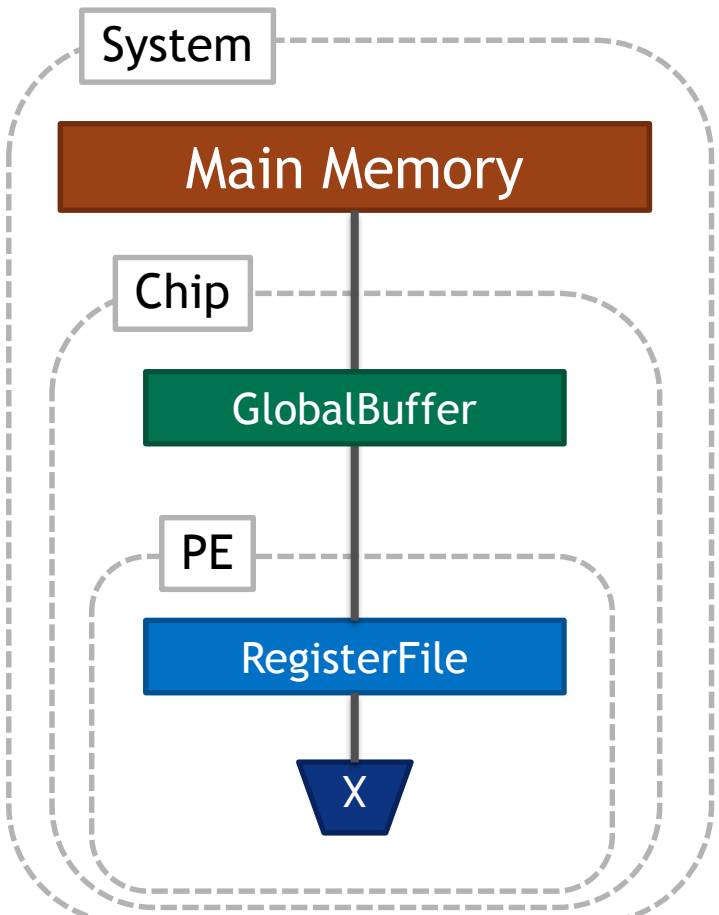
$$\bigvee_{p=1}^{P}\bigvee_{k=1}^{K}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$

$$\bigvee_{k_1=1}^{K_1}\bigvee_{p=1}^{P}\bigvee_{k_0=1}^{K_0}\bigvee_{r=1}^{R}(O_{kp} \mathrel{+}= W_{kr}I_{p+r-1})$$
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 $p = p_1P_0 + p_0$

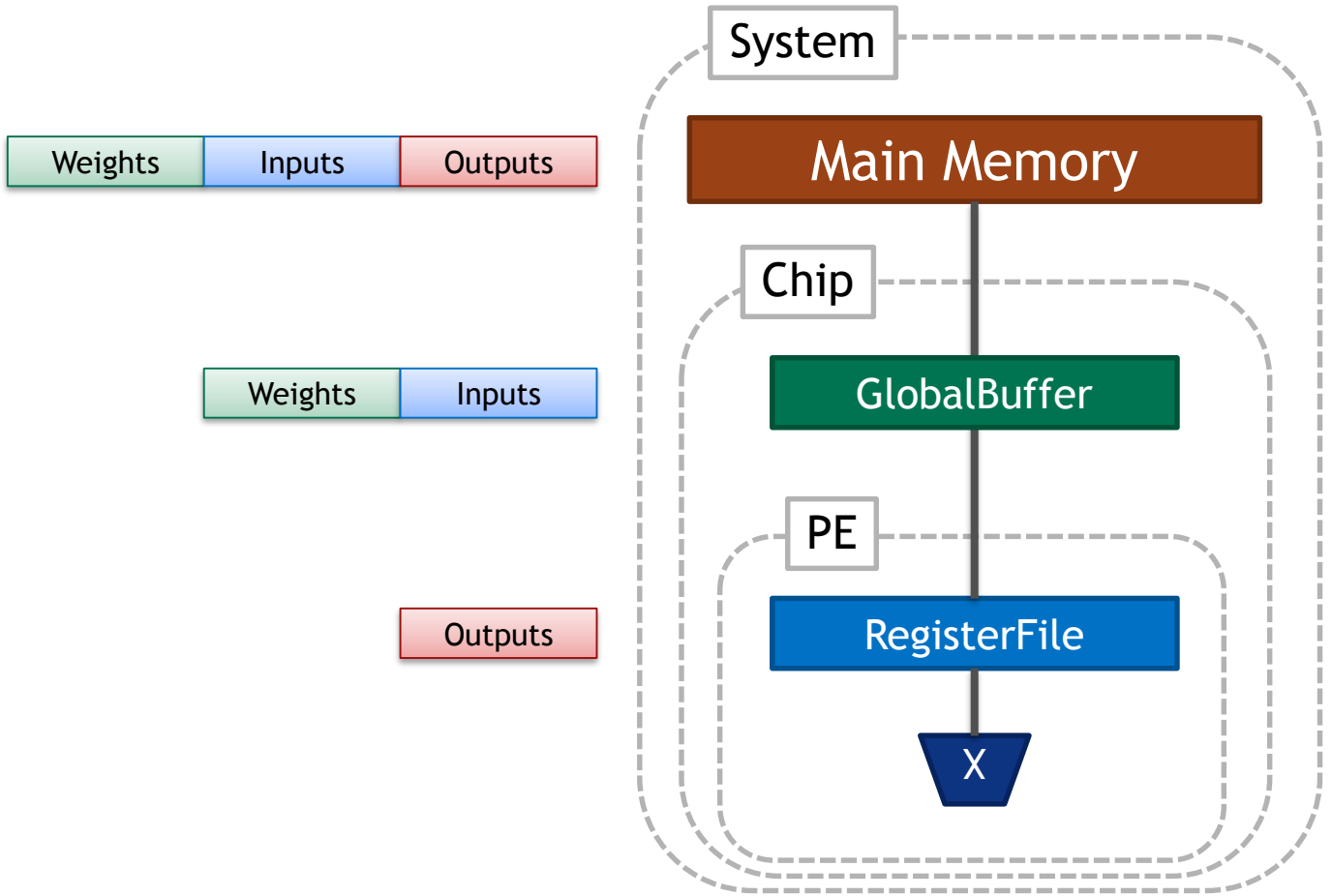
EXERCISE 3: ARCHITECTURE

3-Level Temporal



EXERCISE 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
```


NVIDIA.

EXERCISE 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 evaluator assumes each datatype is stored at each level.

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

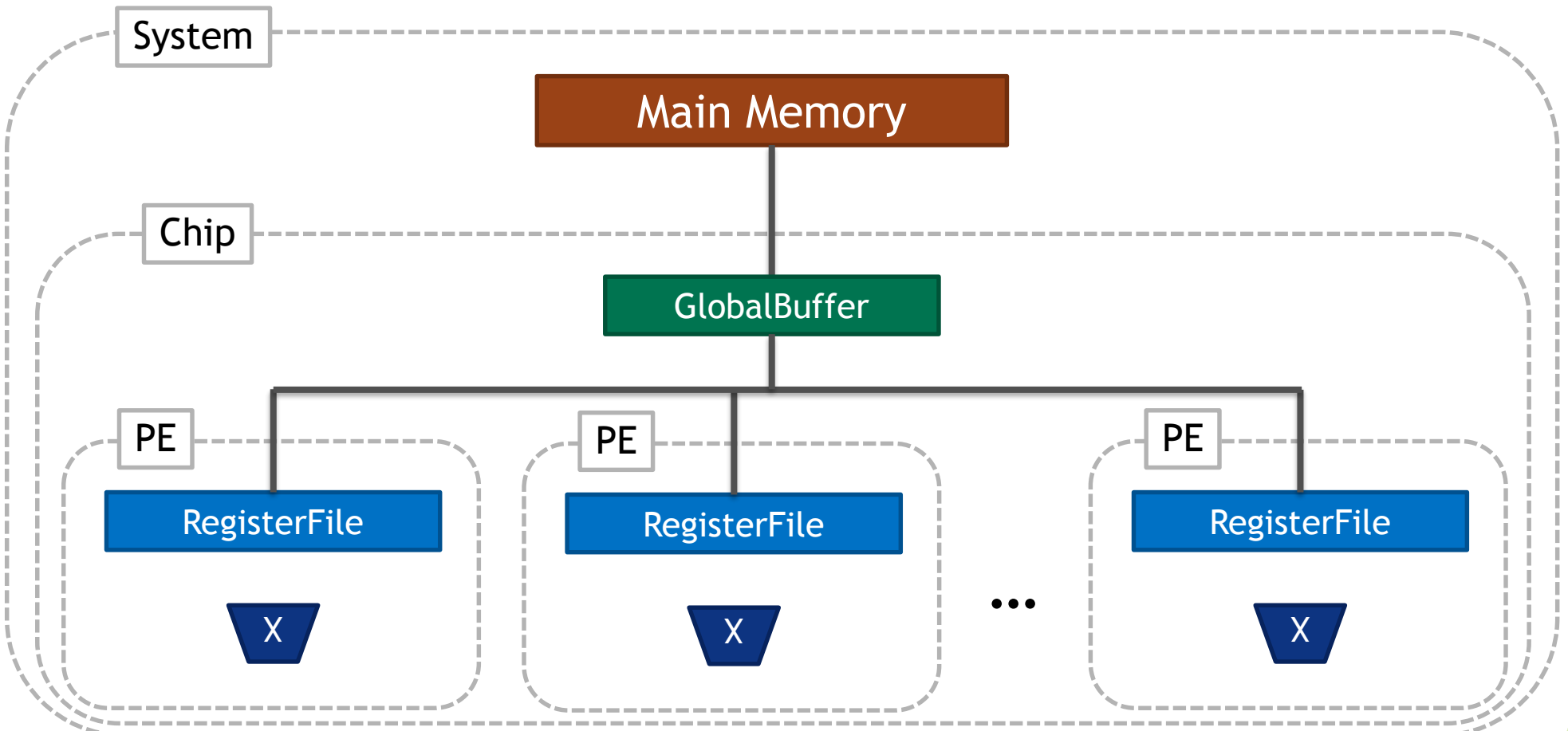
Follow the directions in the README.

Challenge

- Experiment with bypass strategies to find out if there's any benefit in bypassing for this problem.

EXERCISE 4: SPATIAL INSTANCES

3-Level with multiple PEs

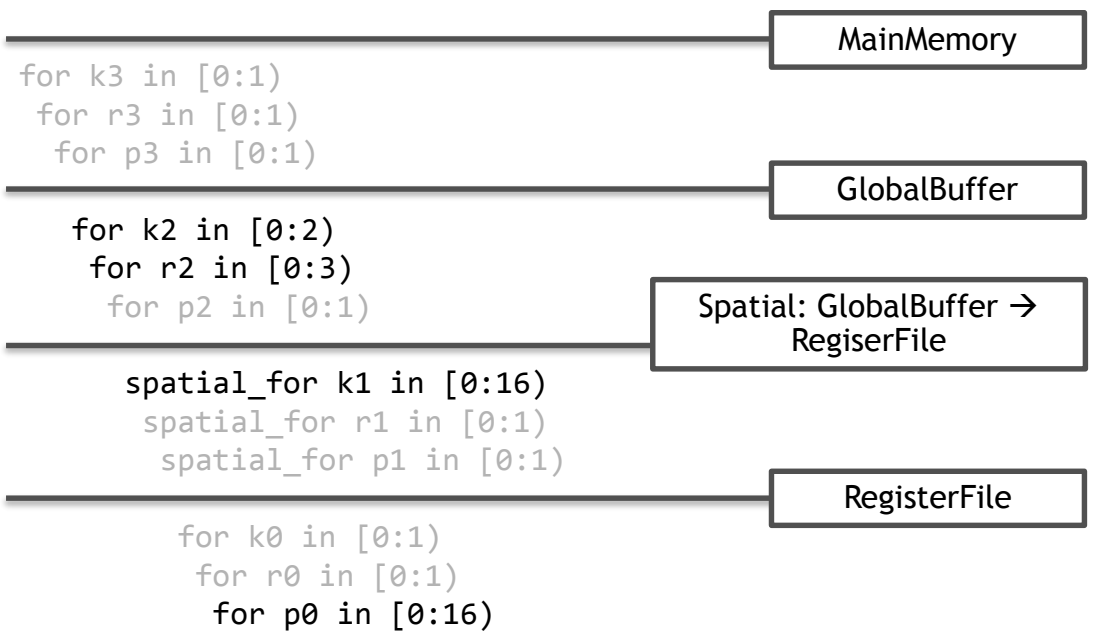


NVIDIA.

EXERCISE 4: MAPPING

Spatial levels need loops too

To represent this…



Write:

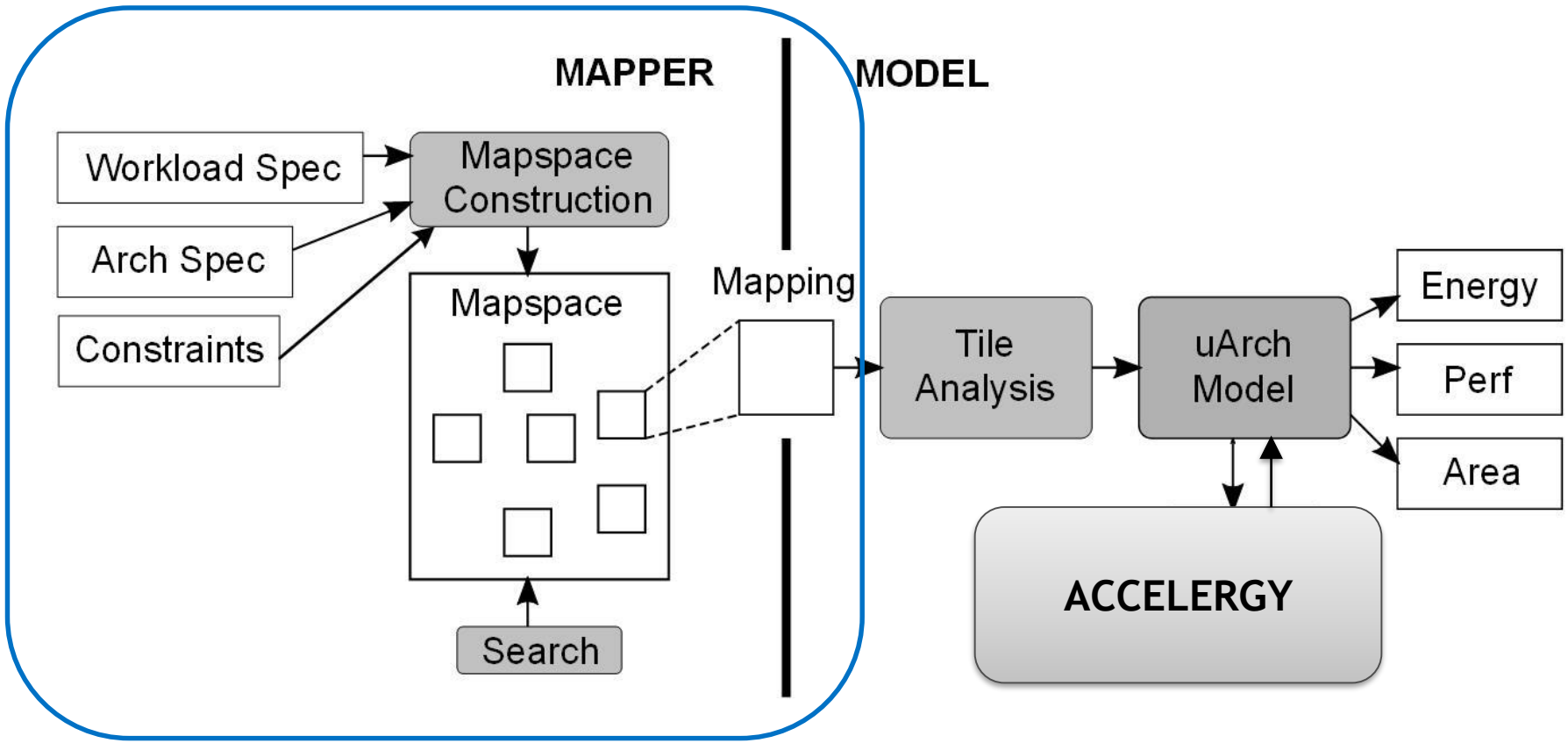
```
mapping:
  - target: MainMemory
    type: temporal
    factors: R=1 P=1 K=1
    permutation: PRK

  - 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
```

EXERCISE 4

Follow the directions in the README.

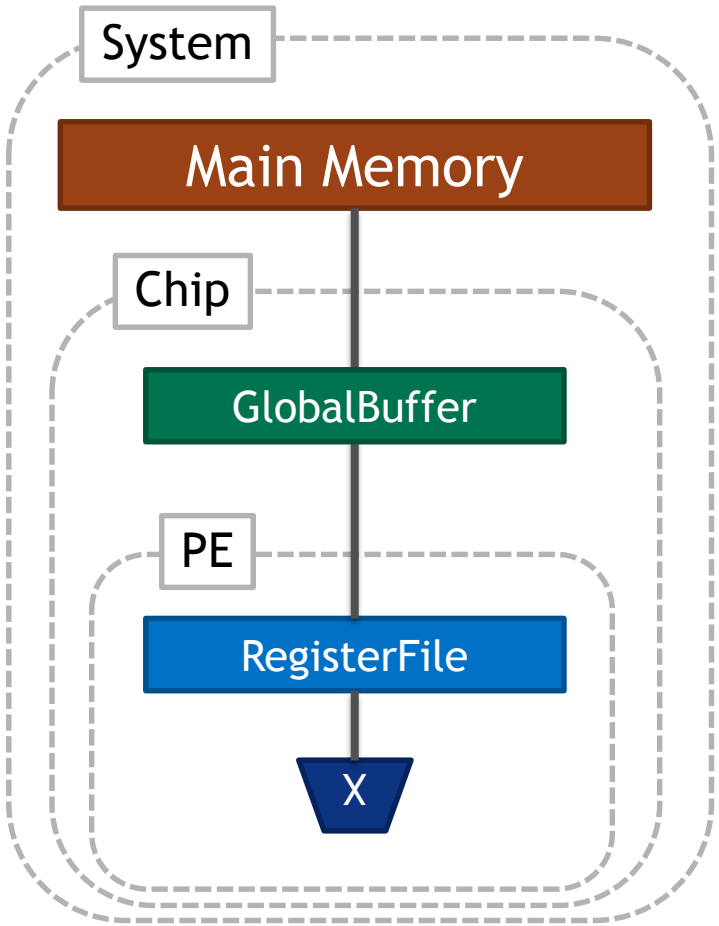

FUN WITH TIMELOOP
THE MAPPER

THE MAPPER


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

EXERCISE 5: ARCHITECTURE

3-Level Temporal



EXERCISE 5: MAPSPACE CONSTRAINTS

Retain old “mapping” directives, now treat them as *mapspace constraints*.

Does this set of constraints result in only 1 legal mapping?

- No! Bypass options haven’t been set. If you fire the model directly on the mapping, it assumes no bypass. But if you fire the *mapper* on these constraints, it recognizes that bypass has been left unspecified and explores all options.
- Each of 3 dataspaces may either be kept or bypassed at each of the 2 inner levels (RegisterFile and GlobalBuffer) => (2^2)^3 = 64 choices!

Run Timeloop. Does it find a better bypassing strategy?

Comment out all the factors and permutations, comment out the num-threads parameter and re-run Timeloop. Does it find a better mapping?

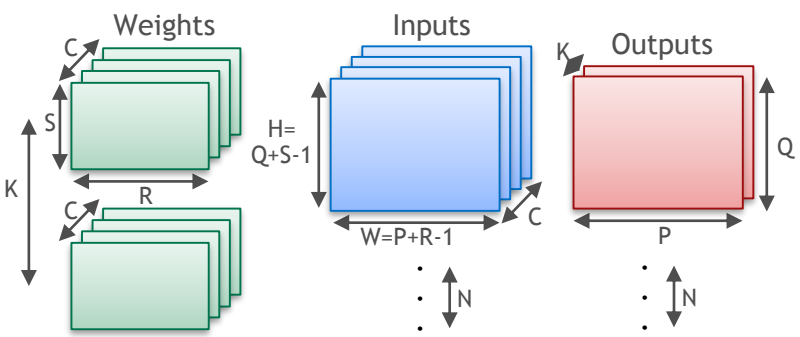
EXERCISE 5

Follow the directions in the README.

EXERCISE 6: PROBLEM

Convolutional Network Layer

```
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[n][k][q][p] +=
                  Weight[c][k][r][s] *
                  Input[n][c]
                       [q*Hstride+s*Hdilation]
                       [p*Wstride+r*Wdilation];
```

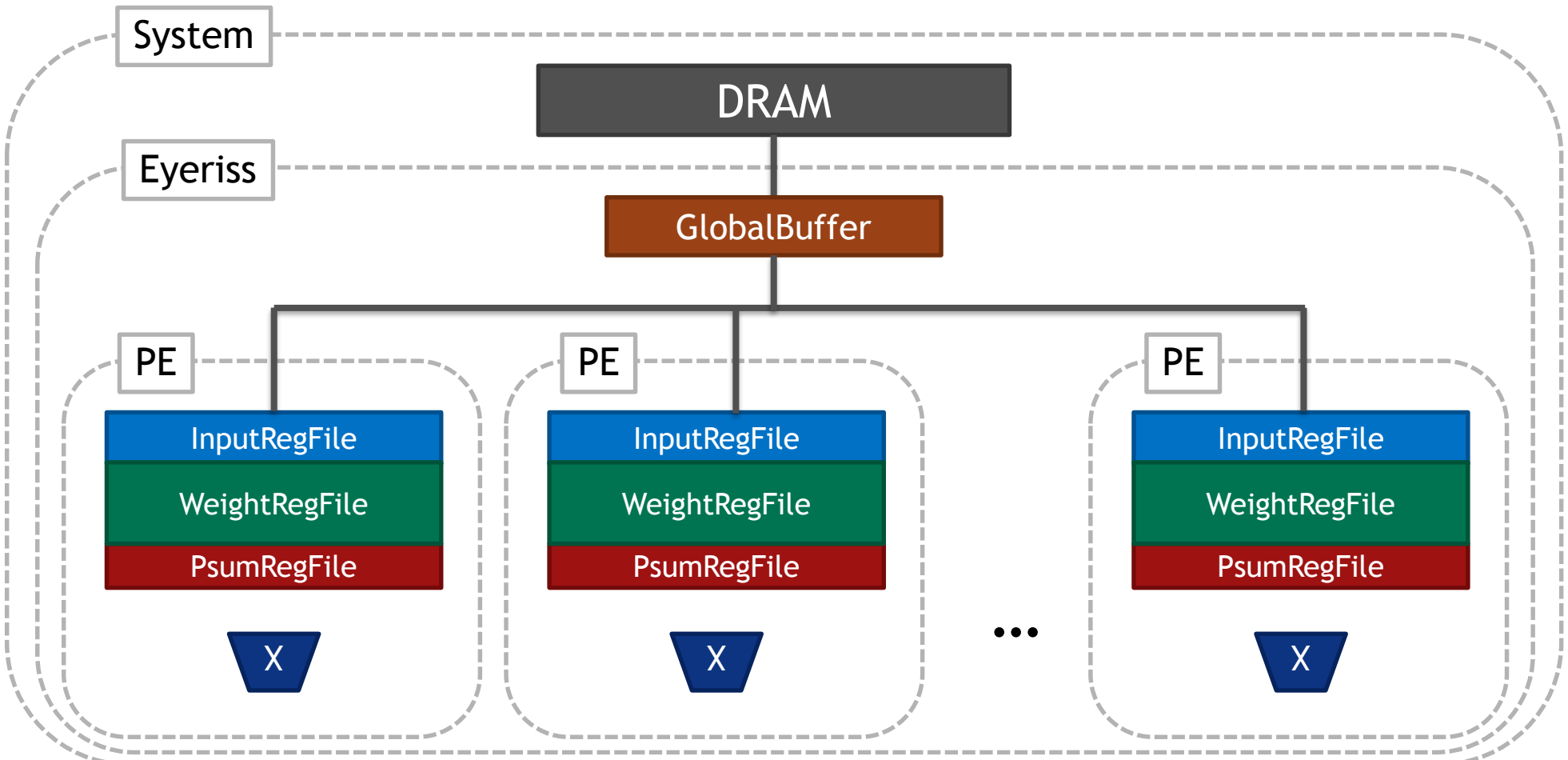



```
problem:
  shape:
    name: CNNLayer
    dimensions:
    - C
    - K
    - R
    - S
    - P
    - Q
    - N
  coefficients:
    - name: Wstride
      default: 1
    - name: Hstride
      default: 1
    - name: Wdilation
      default: 1
    - name: Hdilation
      default: 1
```

```
data-spaces:
   - name: Weights
     projection:
     - [ [C] ],
     - [ [K] ],
     - [ [R] ],
     - [ [S] ]
   - name: Inputs
     projection:
     - [ [N] ]
     - [ [C] ]
     - [ [S, Hdilation], [Q, Hstride] ]
     - [ [R, Wdilation], [P, Wstride] ]
   - name: Outputs
     projection:
     - [ [N] ]
     - [ [K] ]
     - [ [Q] ]
     - [ [P] ]
     read-write: True
```

EXERCISE 6: ARCHITECTURE

Eyeriss-256



NVIDIA.

EXERCISE 6: CNN LAYER ON EYERISS-256

Mapper is multi-threaded.

- Mapspace is split between each mapper thread.
- Default number of threads = number of logical CPUs on host machine.

For long mapper runs, you can use the interactive ncurses-based status tracker by setting mapper.live-status = True

- Tracks various statistics for each mapper thread:
  - Best energy-efficiency/performance seen so far
  - Number of legal/illegal/total mappings examined so far
  - Number of consecutive illegal mappings
  - Number of consecutive legal sub-optimal mappings

EXERCISE 6: TUNING THE MAPPER'S SEARCH

Termination conditions

1. Mapspace exhausted
2. #Valid mappings encountered >= "search-size"
3. #Consecutive invalid mappings encountered >= "timeout"
4. #Consecutive sub-optimal valid mappings encountered >= "victory-condition"
5. Ctrl+C

EXERCISE 6

Follow the directions in the README.

DEEP DIVE: UNDERSTANDING THE MAPPER

CANONICAL MAPPING REPRESENTATION

Mapping: way in which the operation space and the associated data spaces are split into tiles at each level of the architecture's hierarchy and among multiple instances at each level

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];
```

Split into tiling levels:

- R0*R1*R2*R3 = R
- P0*P1*P2*P3 = P

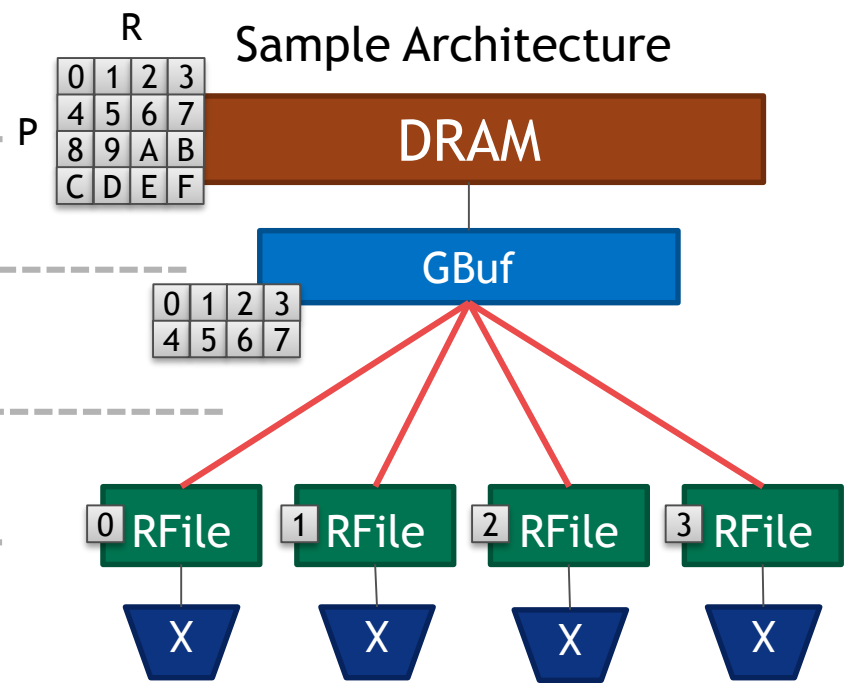
```
// === Mapping ===
// Main Memory
for r3=[0:R3):
  for p3=[0:P3):

    // Buffer
    for r2=[0:R2):
      for p2=[0:P2):

        // Spatial: GBuf->RFile
        parallel_for r1=[0:R1):
          parallel_for p1=[0:P1):

            // RFile
            for r0=[0:R0):
              for p0=[0:P0):
                p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
                r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
                Output[p] += Weight[r] * Input[r+p];
```

Tiling Level 3

Tiling Level 2

Tiling Level 1

Tiling Level 0

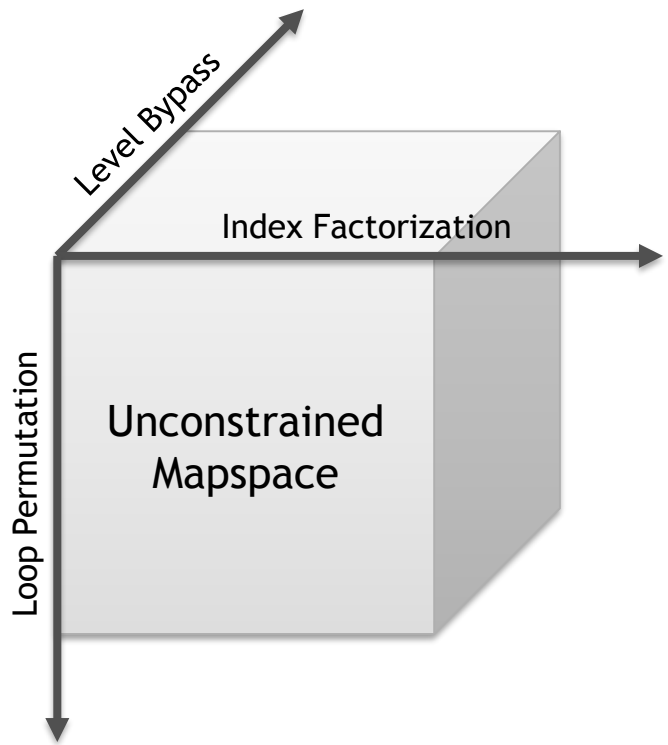


NVIDIA.

MAPSPACES

Set of legal mappings of a workload onto an architecture

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];

// === Mapping ===
// DRAM
for r3=[0:R3):
  for p3=[0:P3):

    // GBuf
    for r2=[0:R2):
      for p2=[0:P2):

      // Spatial: GBuf->RFile
      parallel_for r1=[0:R1):
        parallel_for p1=[0:P1):

        // RFile
        for r0=[0:R0):
          for p0=[0:P0):
            p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
            r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
            Output[p] += Weight[r] * Input[r+p];
```



MAPSPACE CONSTRAINTS

Index Factorization and Loop Permutation

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];

// === Mapping ===
// DRAM
for r3=[0:R3):
  for p3=[0:P3):

    // GBuf
    for r2=[0:R2):
      for p2=[0:P2):

      // Spatial: GBuf->RFile
      parallel_for r1=[0:R1):
        parallel_for p1=[0:P1):

        // RFile
        for r0=[0:3):
          for p0=[0:P0):
            p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
            r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
            Output[p] += Weight[r] * Input[r+p];
```

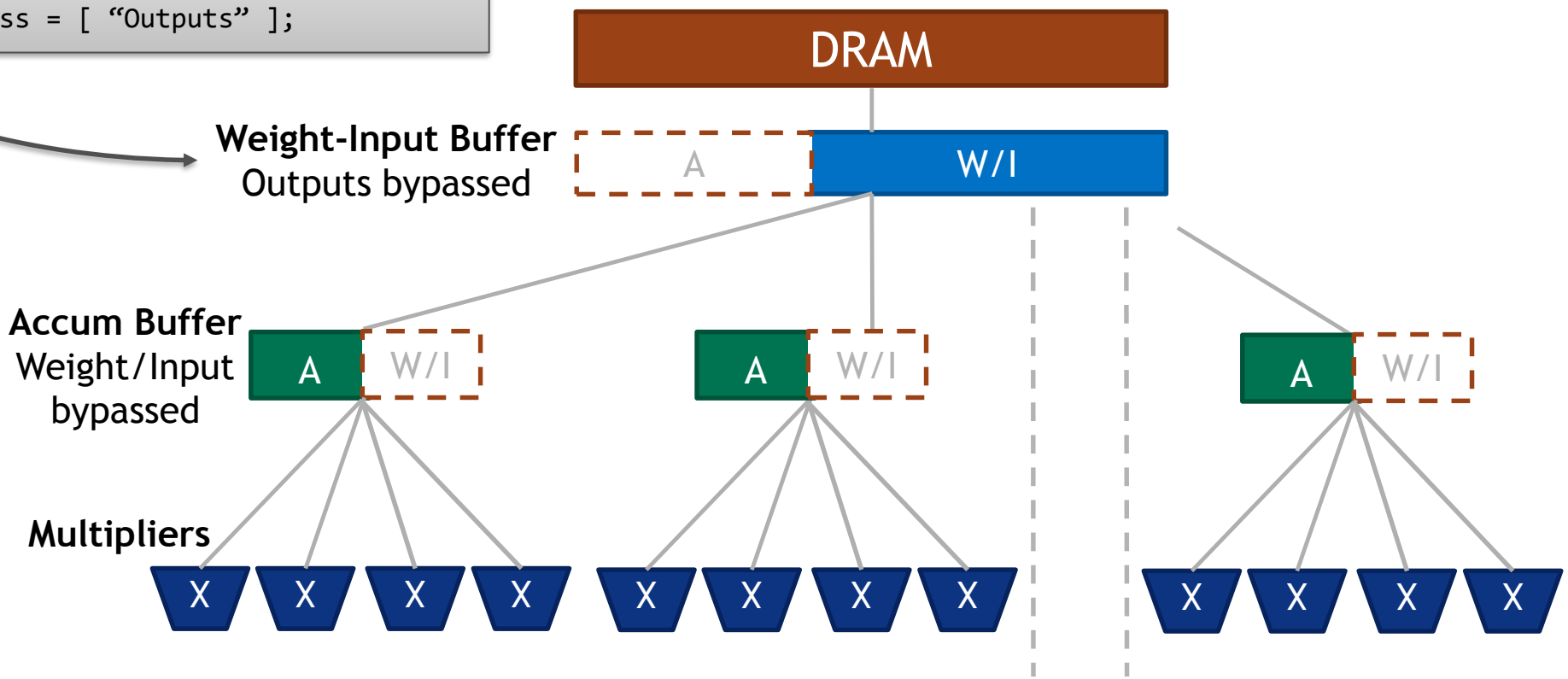
```
target = "RFile";
factors = "R=3";
permutation = “PR”;
```

- Note that P is a free variable. Timeloop will automatically choose an optimal P and will know not to exceed the capacity of the buffer at this level.
- Inequalities are allowed, e.g., P<=16.
- Permutations can be partially specified.

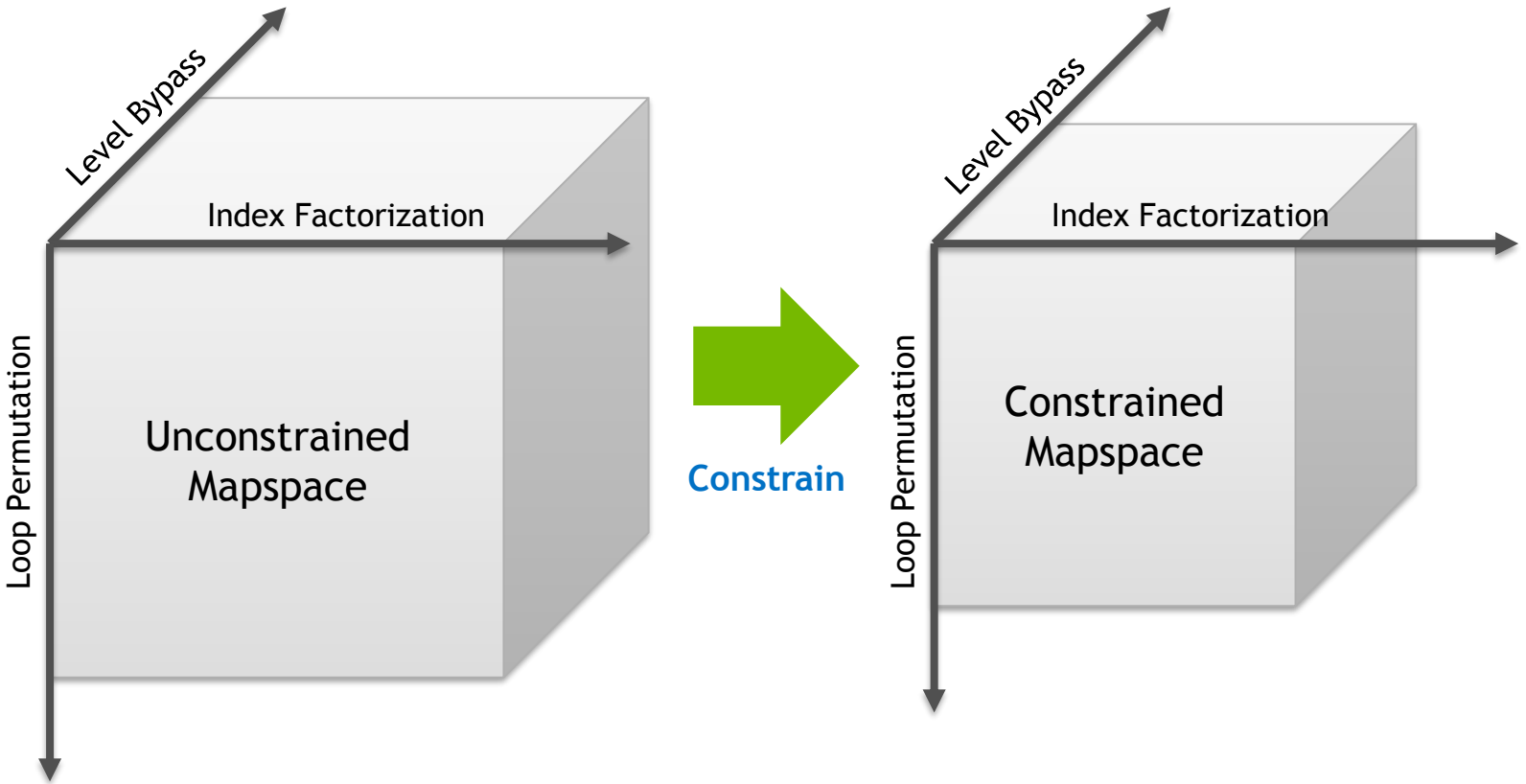
MAPSPACE CONSTRAINTS

Buffer Level Bypassing

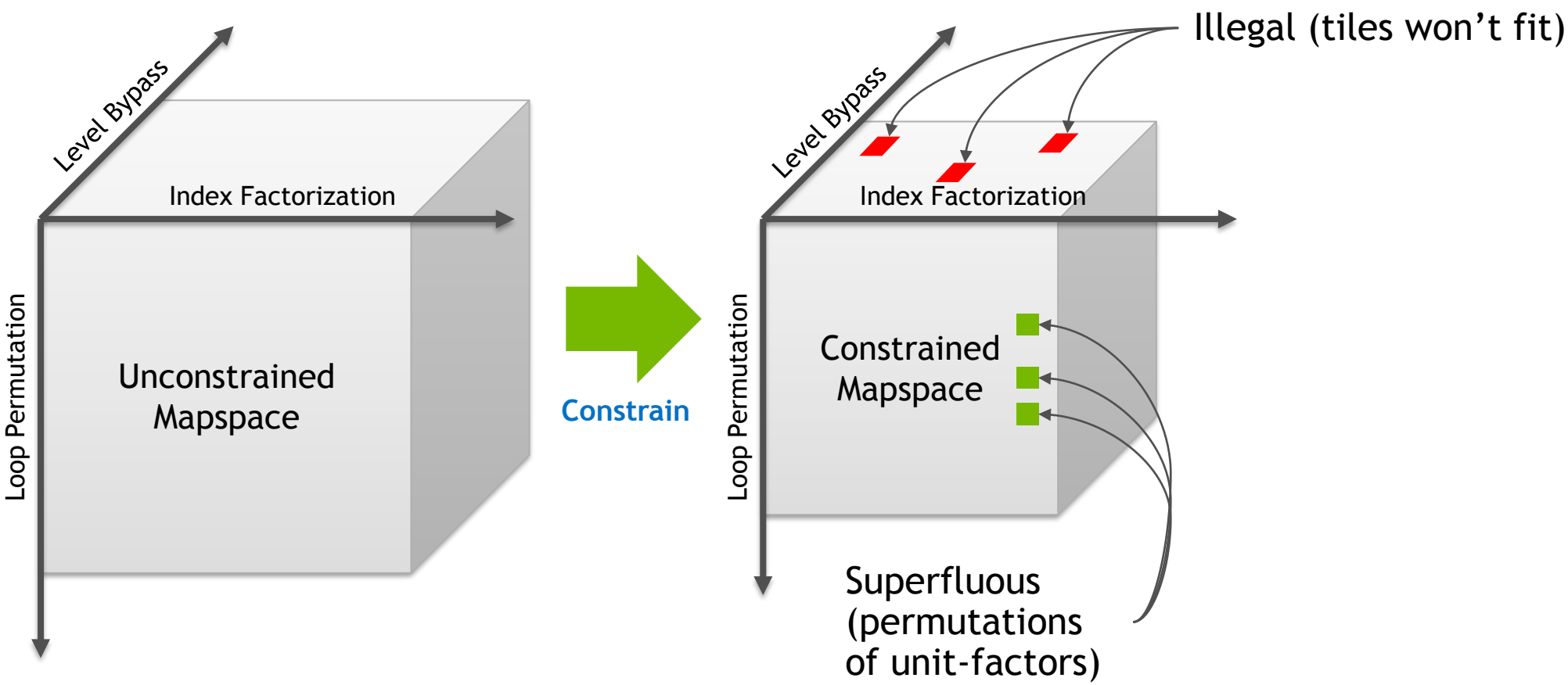
```
target = "Weight-Input Buffer";
keep = [ "Weights" ];
bypass = [ "Outputs" ];
```



CONSTRAINING A MAPSPACE

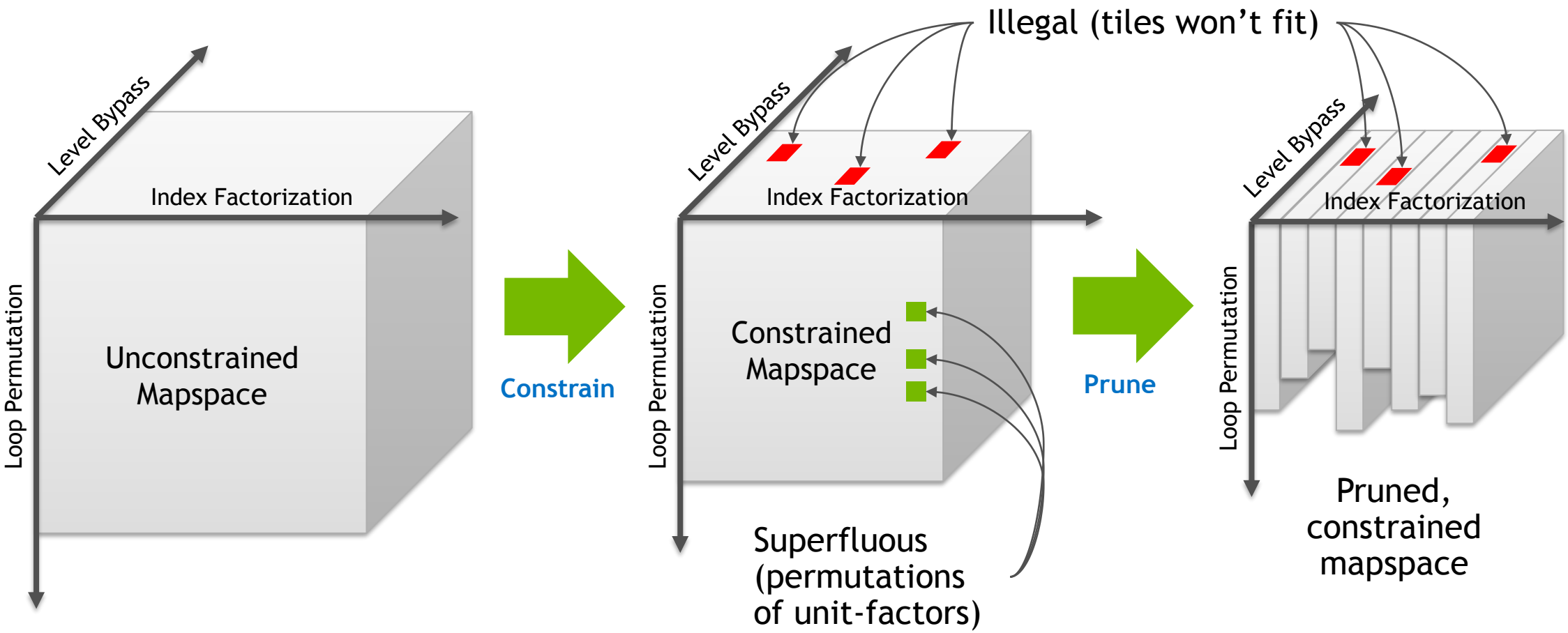


PRUNING A MAPSPACE



NVIDIA.

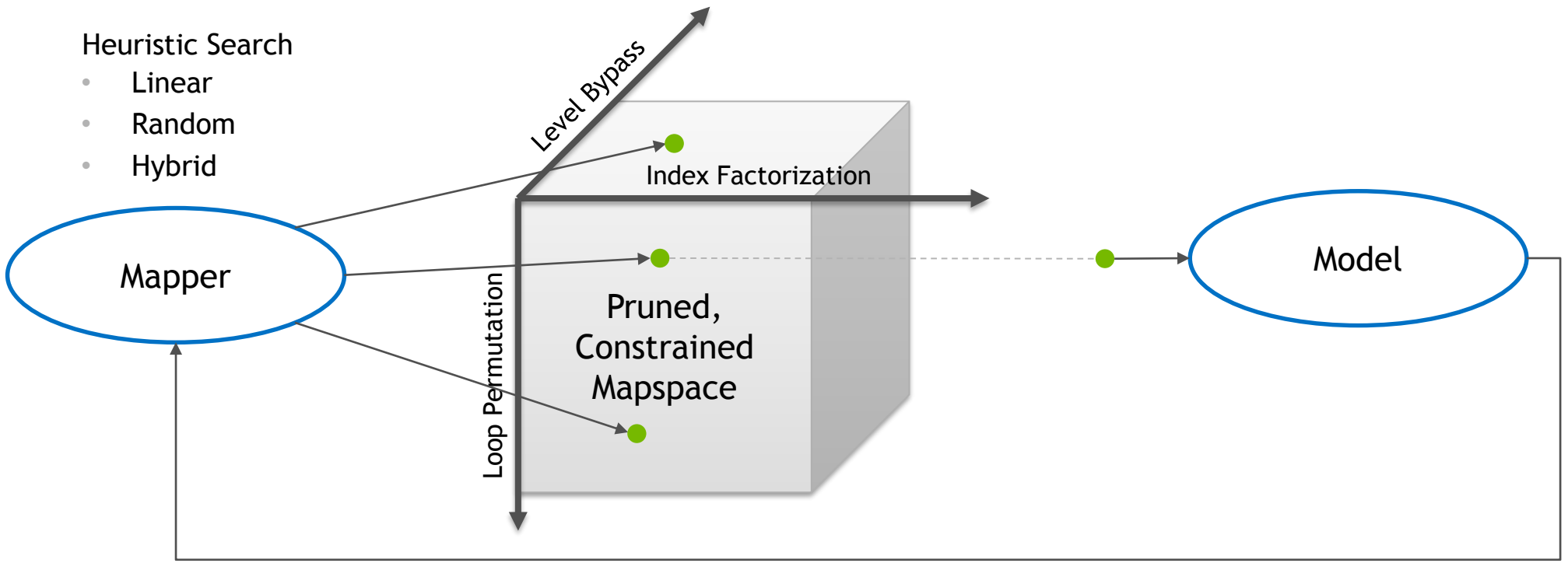
PRUNING A MAPSPACE



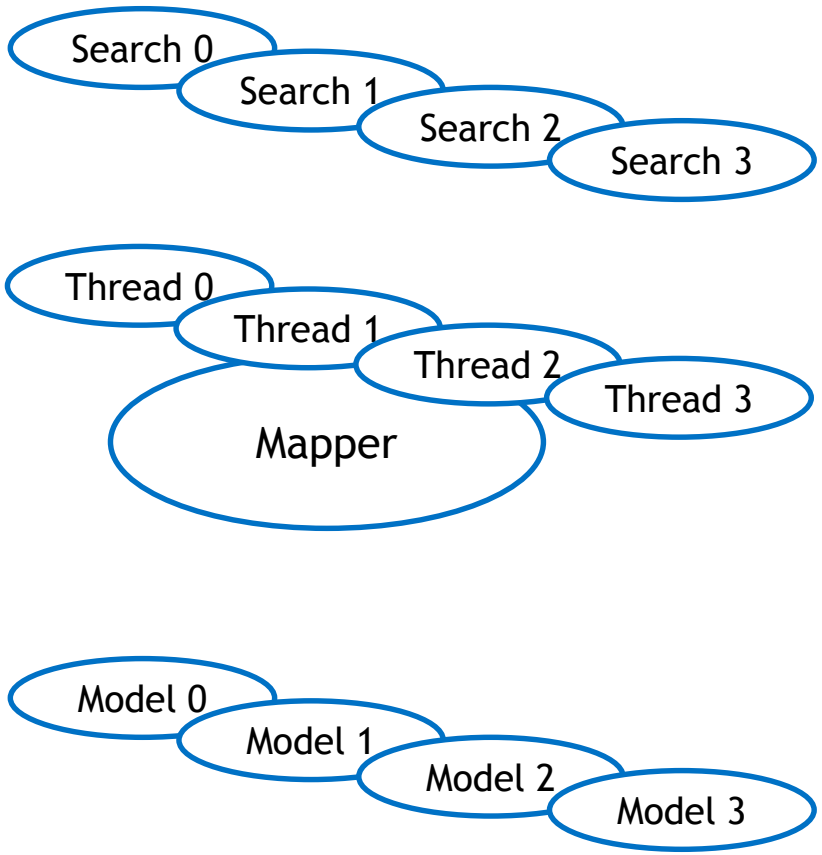
THE MAPPER

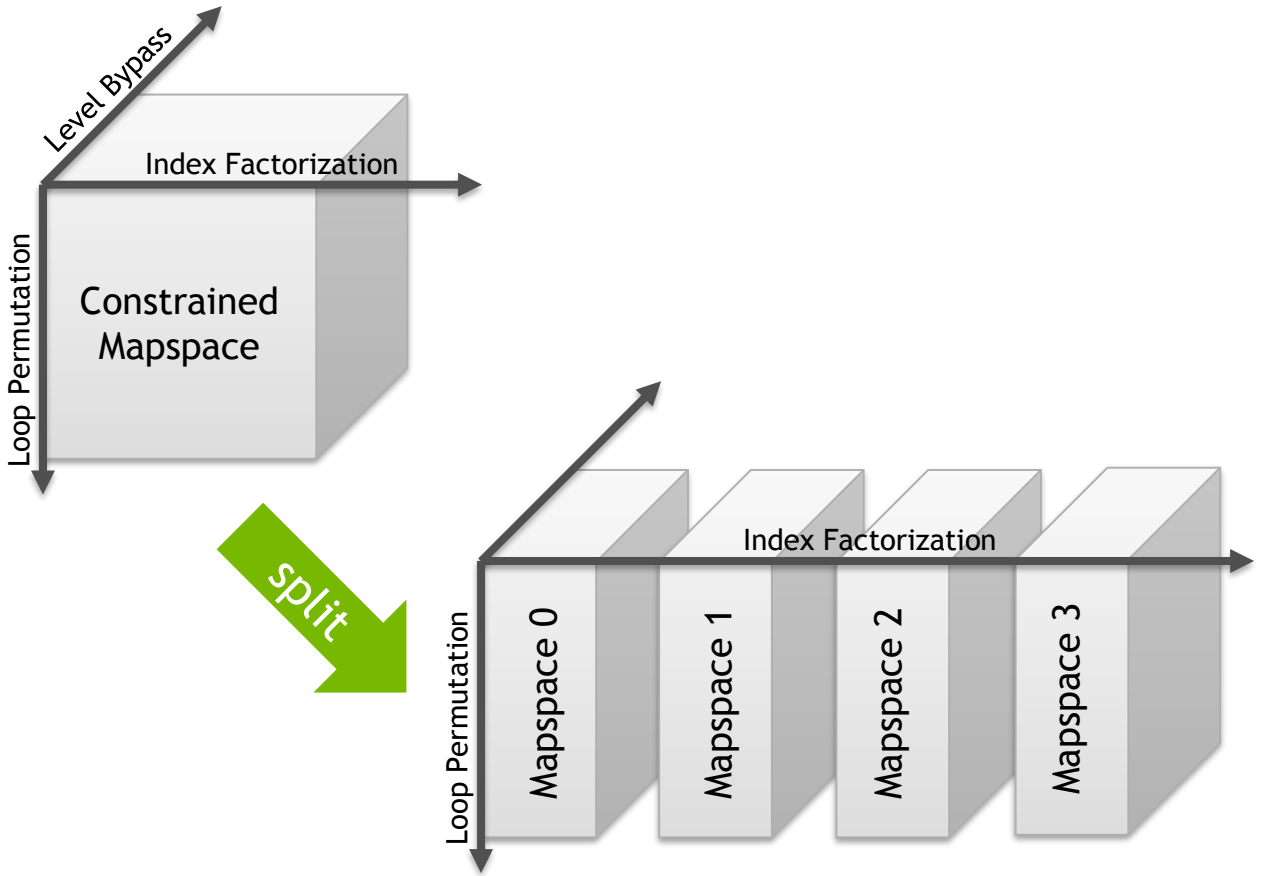
Heuristic Search

- Linear
- Random
- Hybrid



MULTI-THREADING



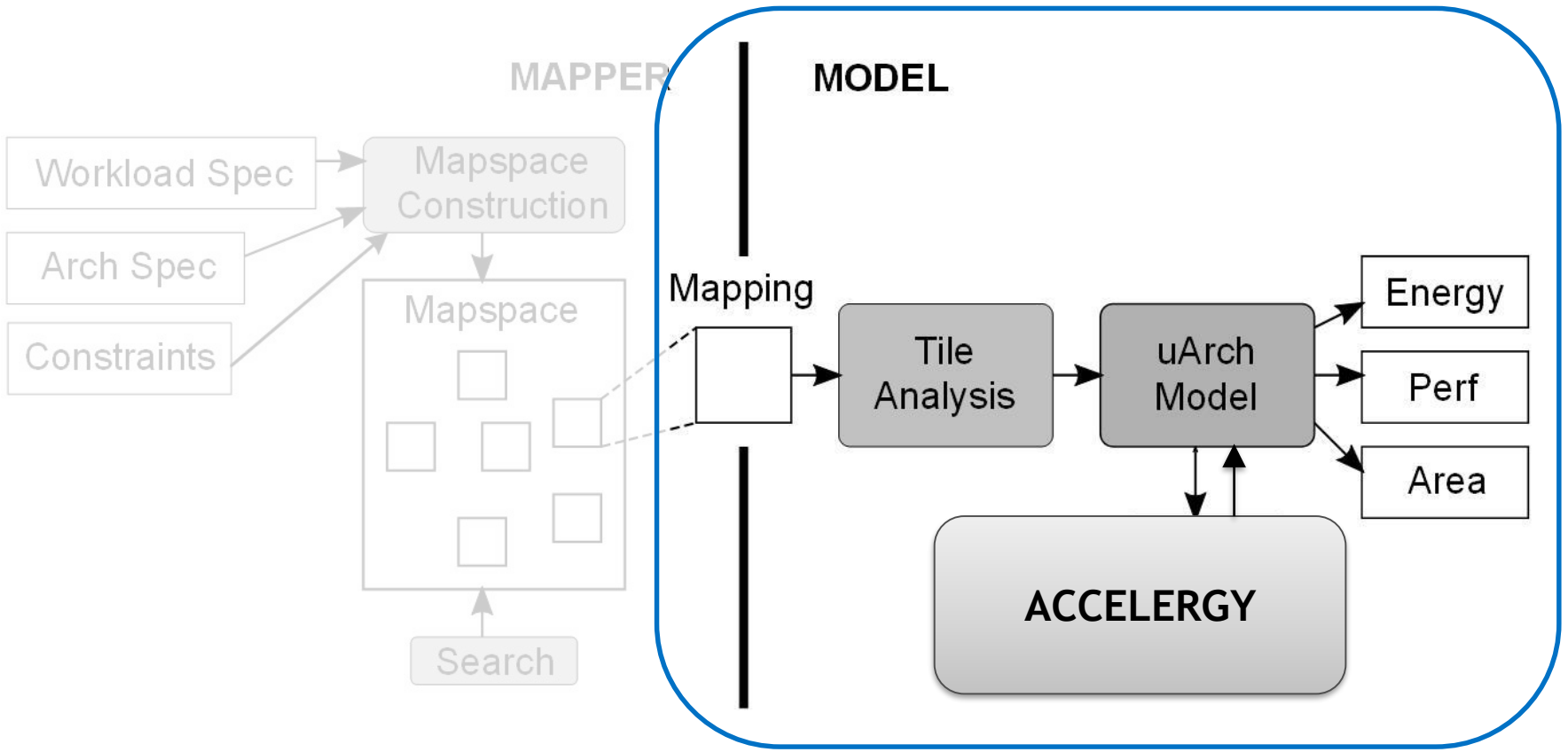


Can also split along other dimensions

NVIDIA.

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

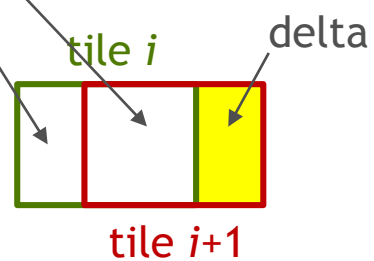
THE MODEL: TILE ANALYSIS

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

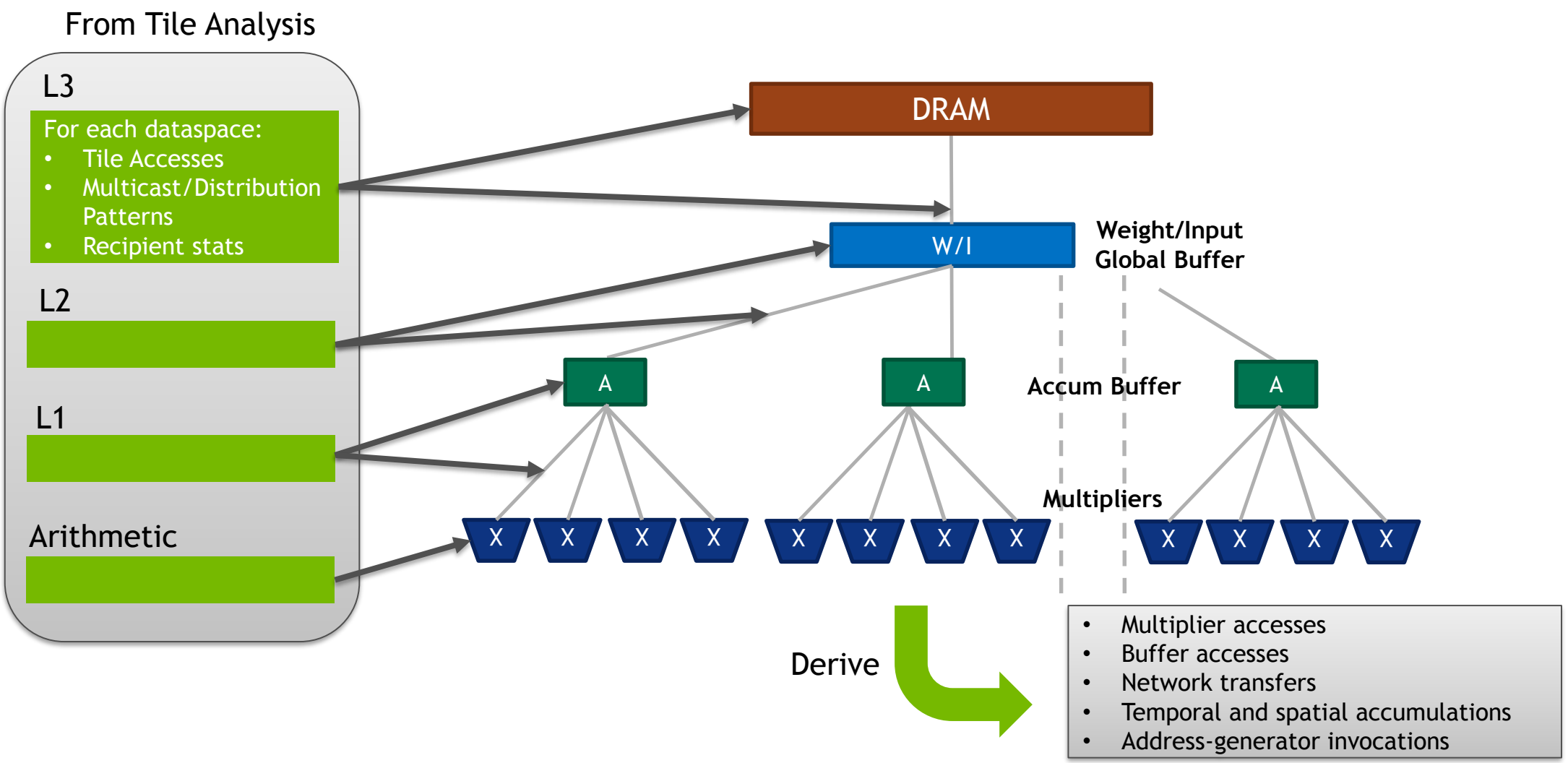
Tile Analysis: Measure and record deltas over all space and time.

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



ESTIMATING PERFORMANCE AND ENERGY

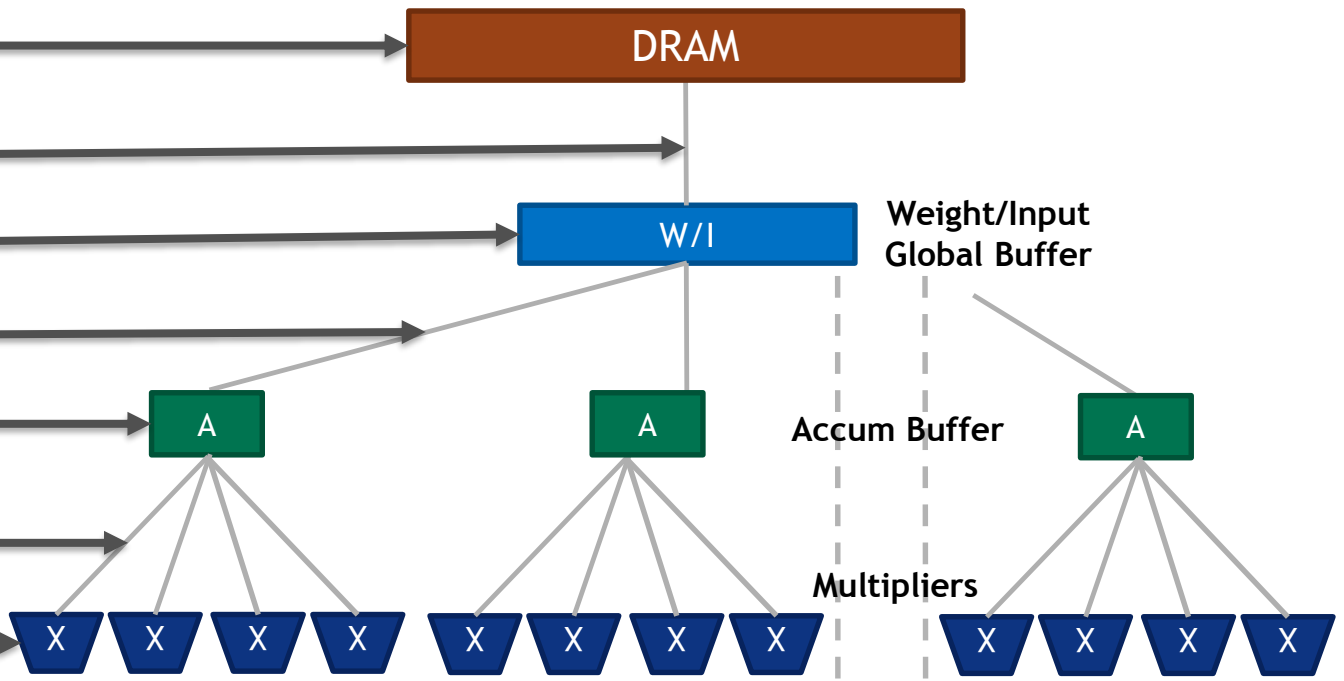
Performance: Throughput of rate-limiting step across:

- 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*

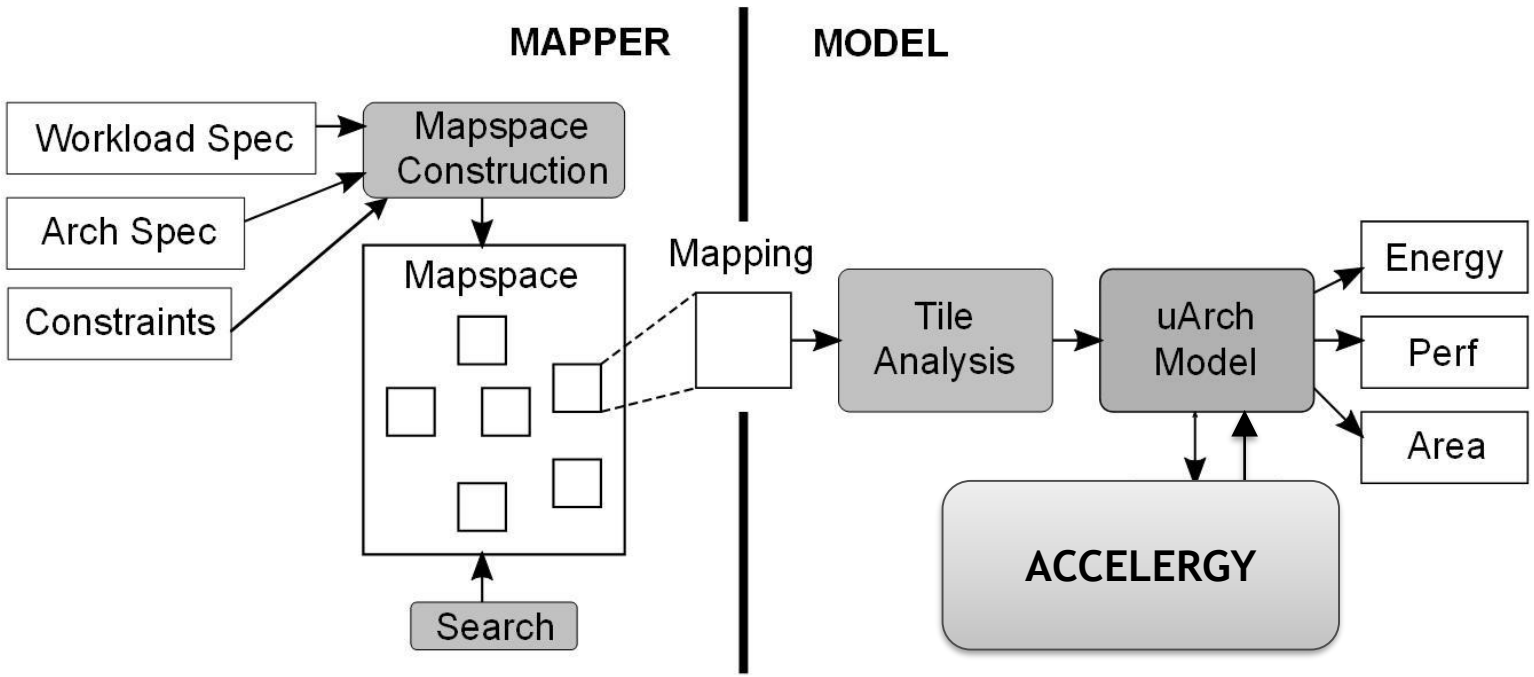
FUTURE WORK

Search Heuristics

Workloads: Complete networks, with inter-layer optimization

Modeling fragmentation and load imbalance in architectures that exploit unstructured sparsity for performance

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.

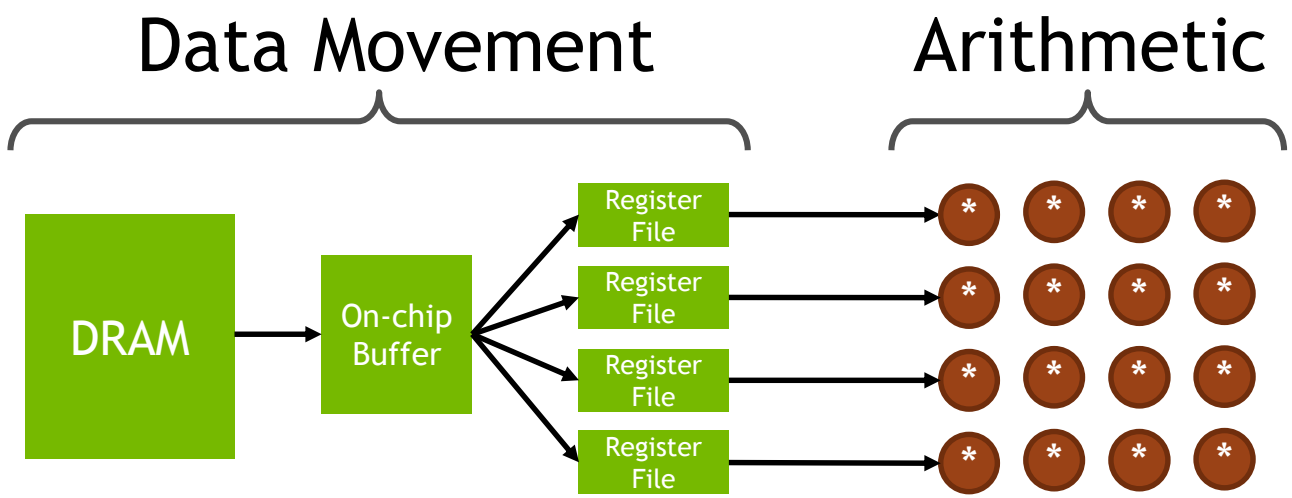
Please join us in making Timeloop better and more useful for research opportunities across the community.


NVIDIA


BACKUP

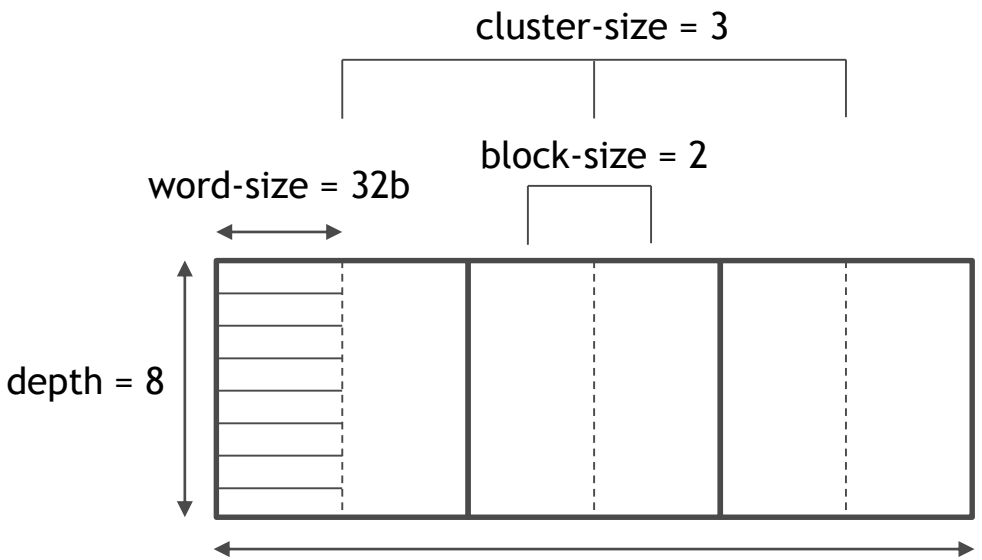
DNN ACCELERATORS

Design Considerations



NVIDIA.

BUFFER PARAMETERS



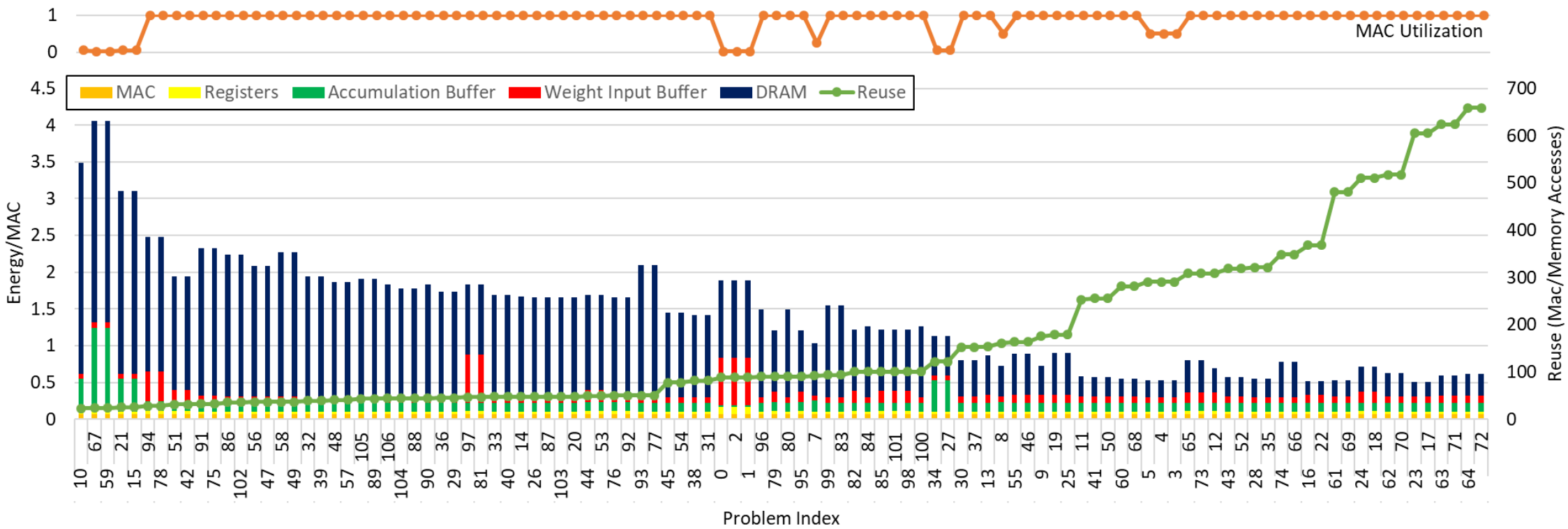
Temporal capacity = 16 * 32b words

- Per instance
- Counted in terms of scalar words
- Used to fit temporal tiles

Spatial instances = 6

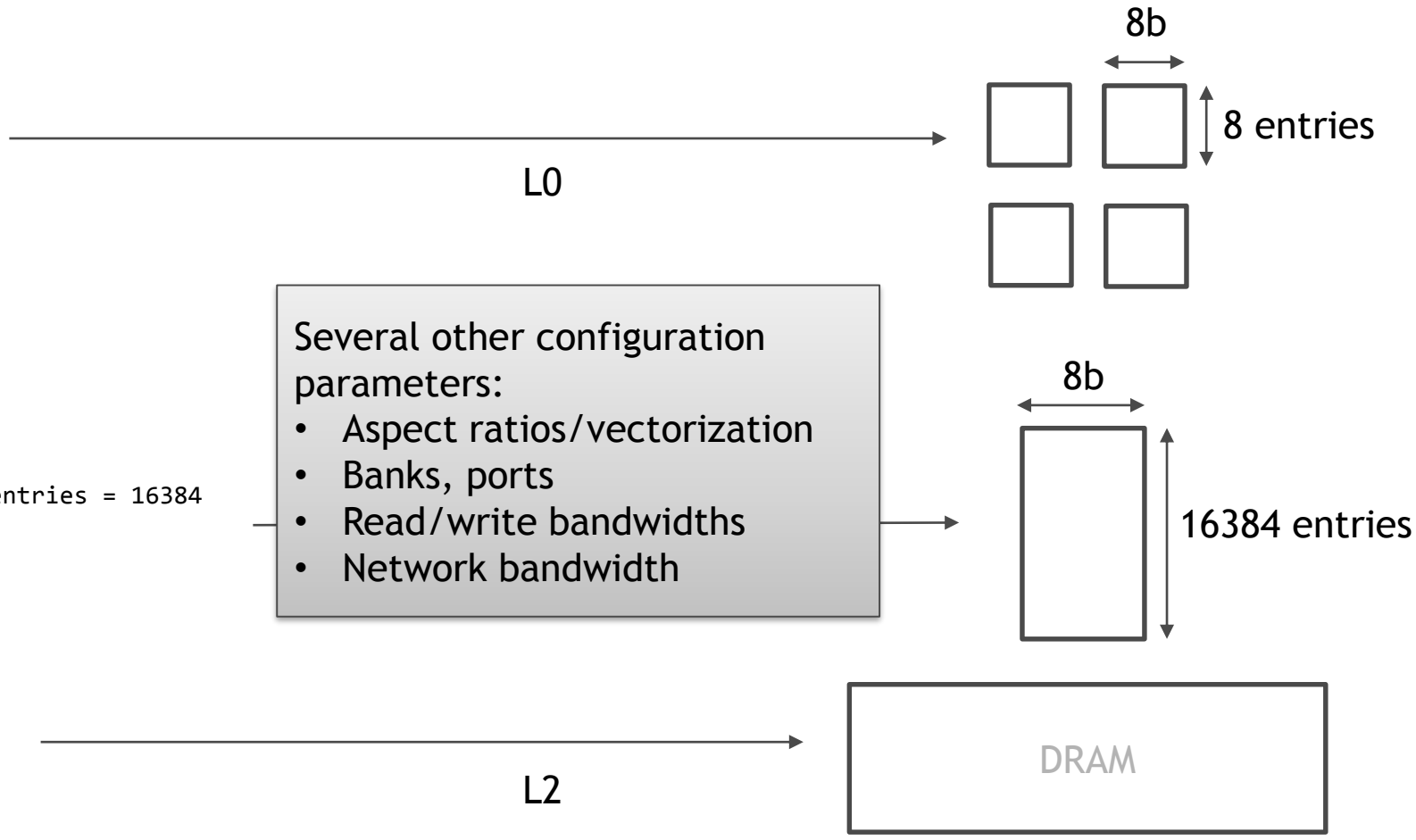
- Spatial fanout of 6 from the next-outer level.
- Can receive multicasts and produce partial sums for spatial reduction)

CASE STUDY: BENCHMARK SWEEP



ARCHITECTURE SPECIFICATION: EXAMPLE

```
storage =
 (
   {
     name = “Local";
     entries = 8;
     instances = 4;
     word-bits = 8;
     meshX = 2;
     meshY = 2;
   },

   {
     name = “Global";
     sizeKB = 16; // => entries = 16384
     instances = 1;
     word-bits = 8;
   },

   {
     name = “MainMemory”;
     technology = “DRAM”;
     instances = 1;
   }
 )
```



MAPPINGS

Mapping: way in which the operation space and the associated data spaces are split into tiles at each level of the architecture's hierarchy and among multiple instances at each level

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];
```

Split into tiling levels:

- R0*R1*R2*R3 = R
- P0*P1*P2*P3 = P

```
// === Mapping ===
// DRAM
for r3=[0:R3):
  for p3=[0:P3):

    // GBuf
    for r2=[0:R2):
      for p2=[0:P2):

      // Spatial: GBuf->RFile
      parallel_for r1=[0:R1):
        parallel_for p1=[0:P1):

        // RFile
        for r0=[0:R0):
          for p0=[0:P0):
            p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
            r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
            Output[p] += Weight[r] * Input[r+p];
```

Tiling Level 3

Tiling Level 2

Tiling Level 1

Tiling Level 0

Observations:

- Loop bounds => hierarchically slice up operation space (and projected dataspaces) into tiles (inclusive)
- Parallel_for => spatial partitioning across instances at a level
- Ordering of loops within a level => delivery sequence of sub-tiles (*stationarity*)

NVIDIA.

MAPPING CONSTRAINTS

Factors, Permutation and LevelBypass

```
target = "RFile";
type = "temporal";
factors = "Q1 R1 S1 C1 K1 N1";
permutation = “PQRSCKN”;

# Note: P is a free variable. Timeloop
# will automatically choose an optimal
# P and will know not to exceed the
# capacity of the buffer at this level.
```

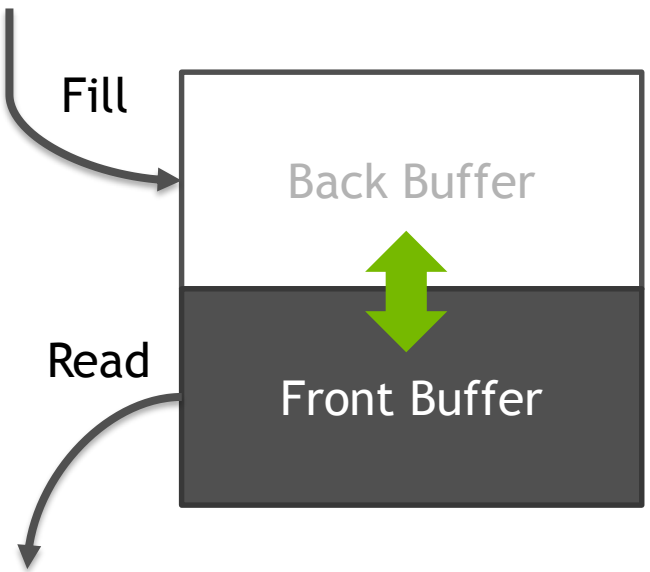
```
target = “GBuf_RFile";
type = "spatial";
factors = "C16 K1 R1 S1 P1 Q1 N1";
```

```
target = “RFile";
type = "datatype";
keep = [ "Weights" ];
bypass = [ "Inputs", "Outputs" ];
```

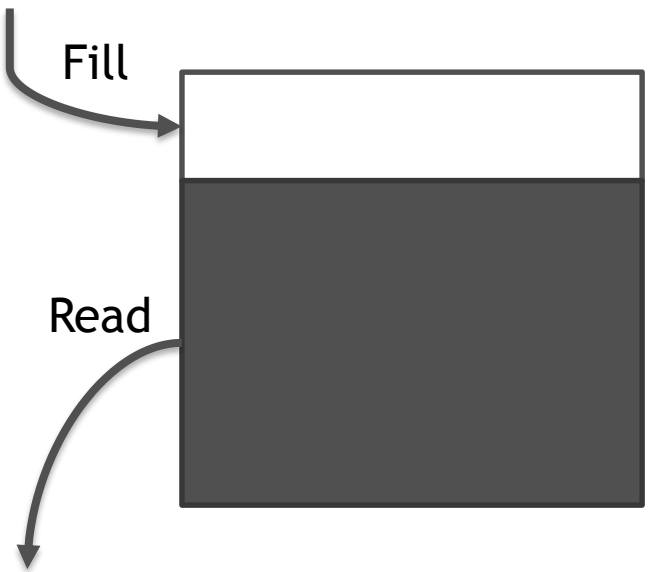

NVIDIA.

BUFFER MODEL

DOUBLE-BUFFERING



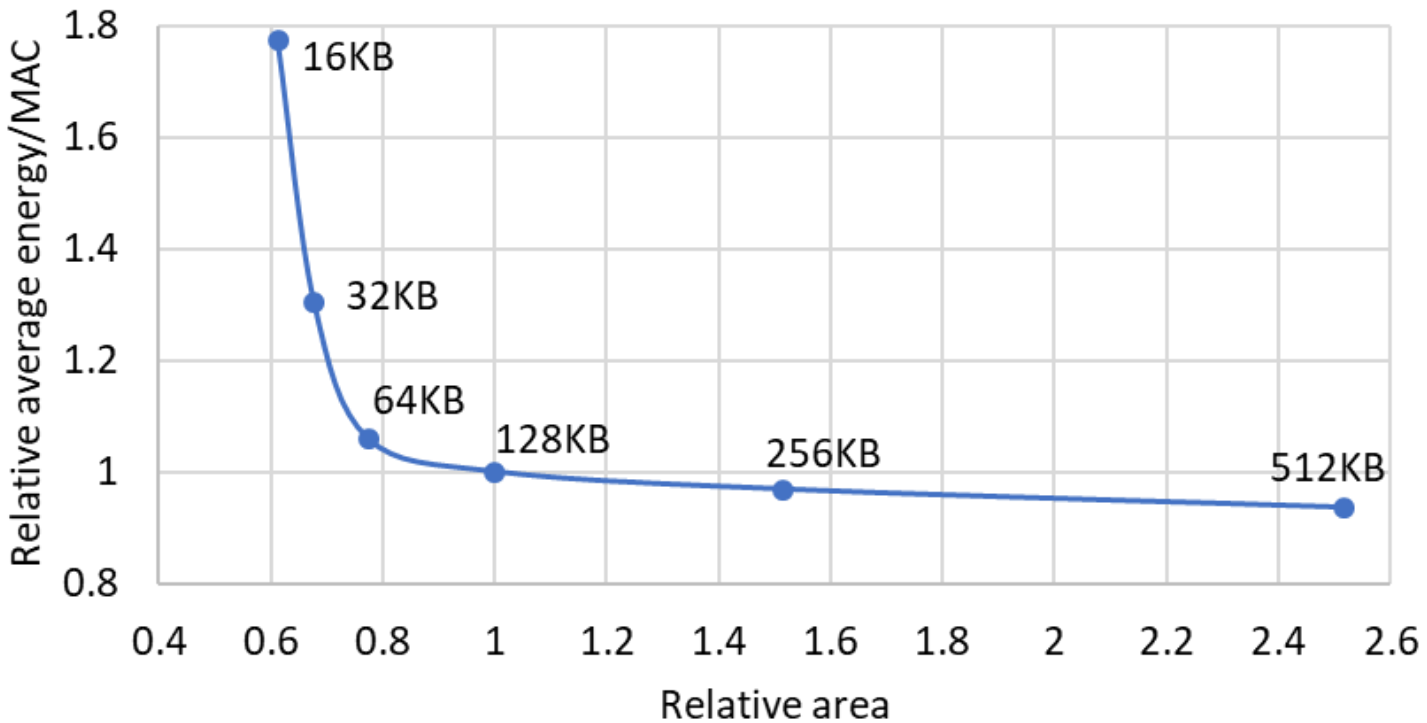
BUFFETS



- Fine-grained pre-buffering of Fills
- Implicit support for sliding windows
- Fine-grained synchronization between various operations, including Fill->Read

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*

CASE STUDY:AREA VS. ENERGY

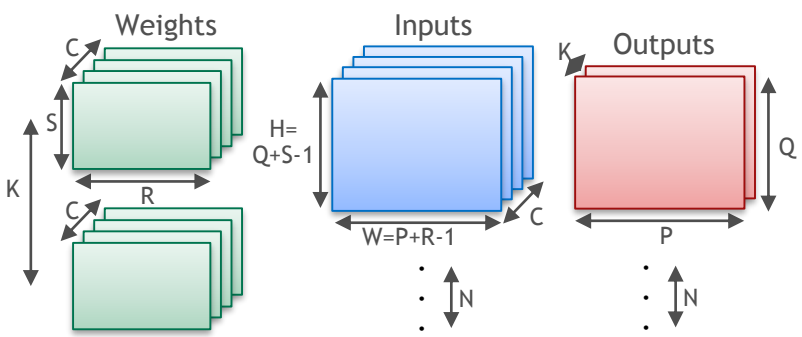


PROBLEM SPECIFICATION

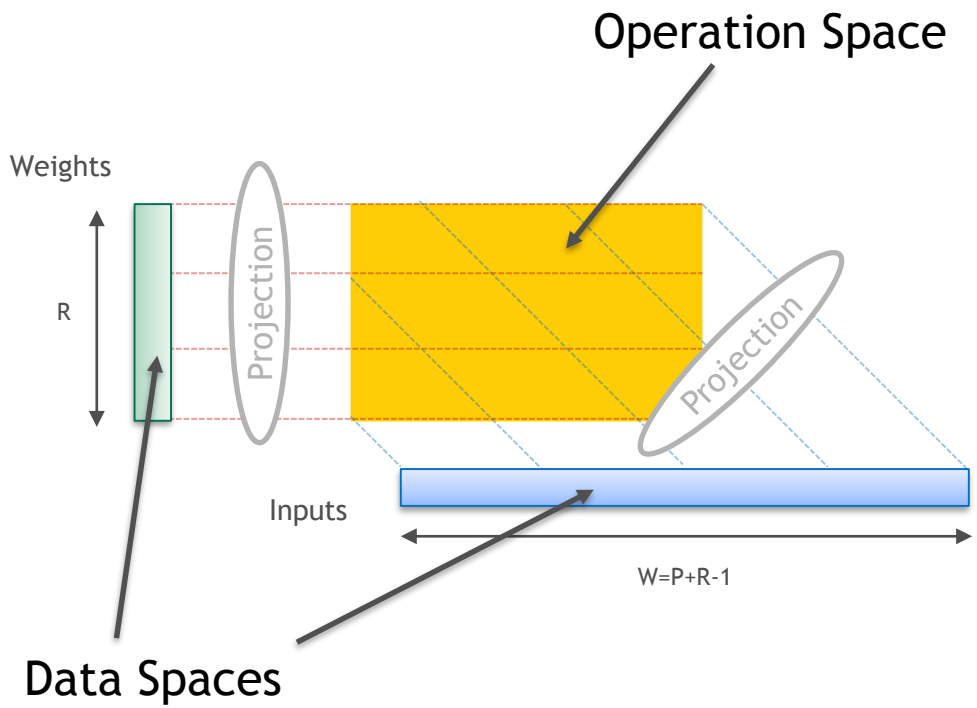
Deep Loop Nest

Provided to Timeloop

```
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];
```



Inferred by Timeloop

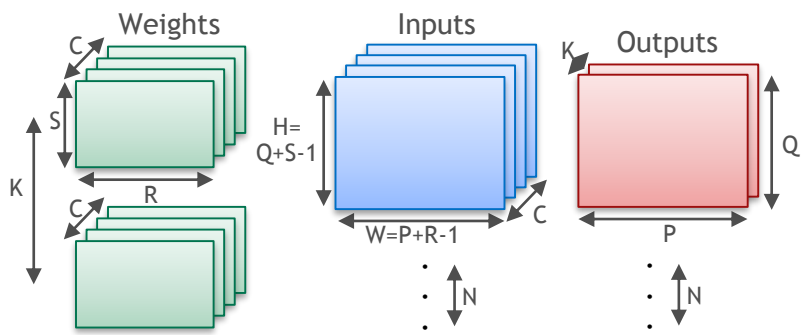


NVIDIA.

PROBLEM SPECIFICATION

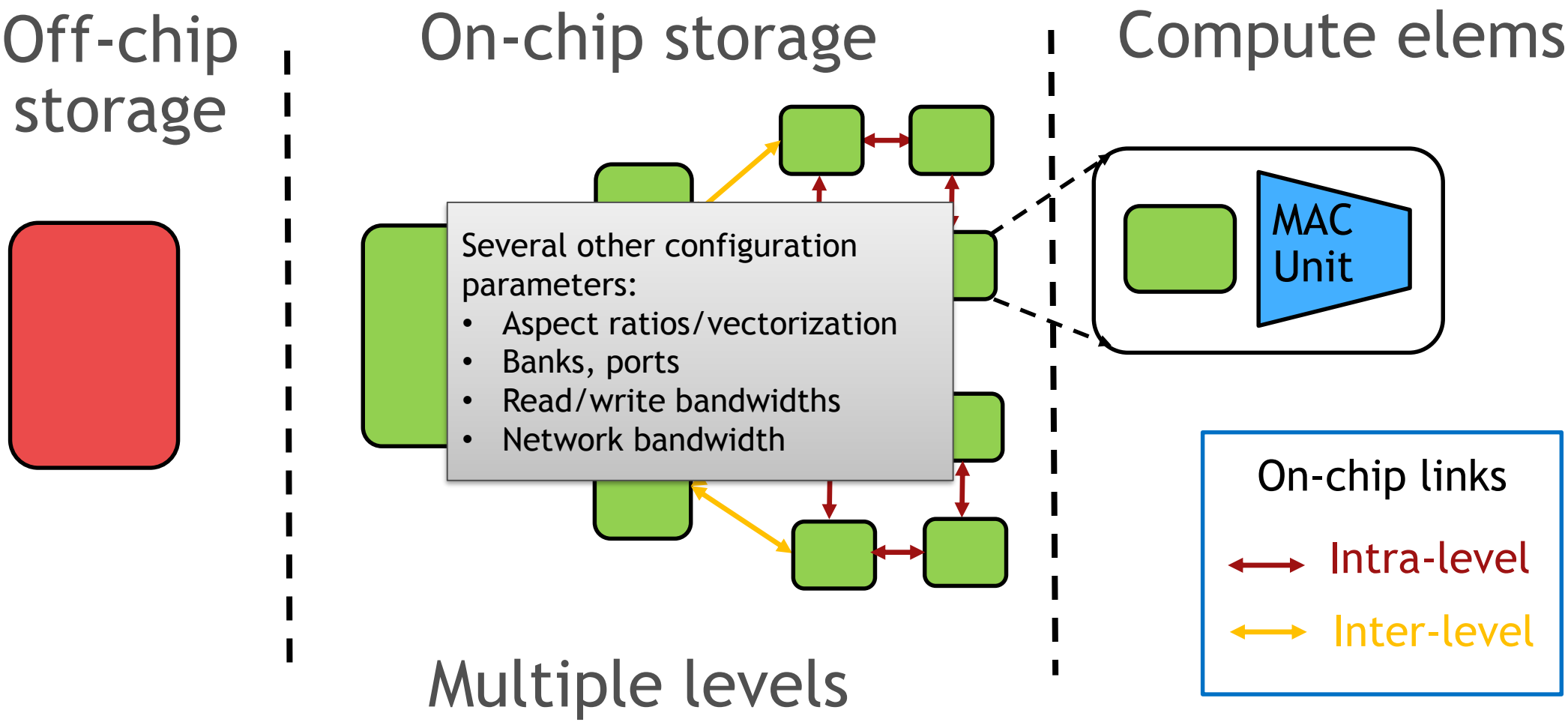
Deep Loop Nest

```
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];
```



NVIDIA.

ARCHITECTURE SPECIFICATION



MAPPINGS

Mapping: way in which the operation space and the associated data spaces are split into tiles at each level of the architecture's hierarchy and among multiple instances at each level

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];
```

Split into tiling levels:
- R0*R1*R2*R3 = R
- P0*P1*P2*P3 = P

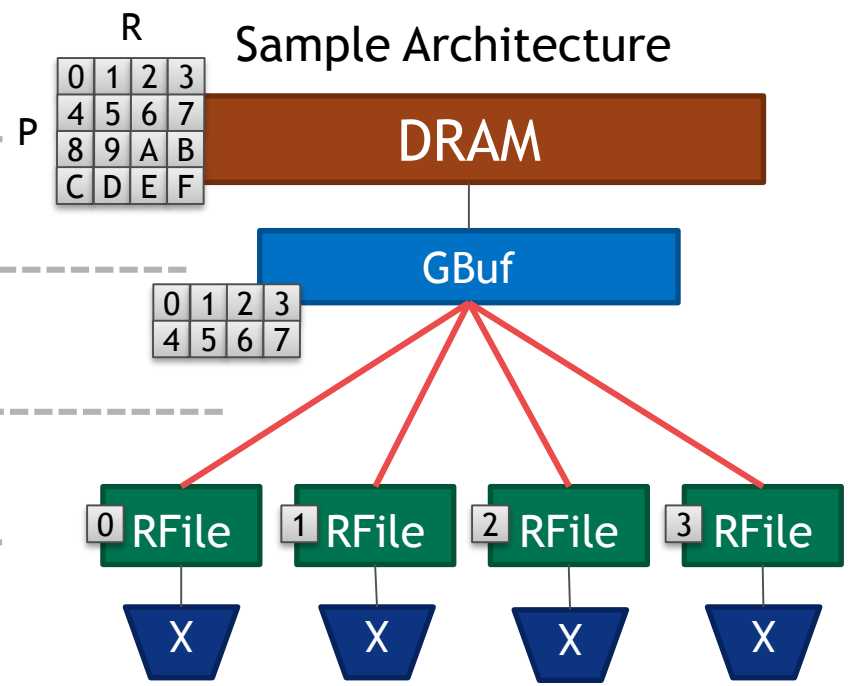
```
// === Mapping ===
// DRAM
for r3=[0:R3):
  for p3=[0:P3):

    // GBuf
    for r2=[0:R2):
      for p2=[0:P2):

        // Spatial: GBuf->RFile
        parallel_for r1=[0:R1):
          parallel_for p1=[0:P1):

            // RFile
            for r0=[0:R0):
              for p0=[0:P0):
                p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
                r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
                Output[p] += Weight[r] * Input[r+p];
```

Tiling Level 3

Tiling Level 2

Tiling Level 1

Tiling Level 0



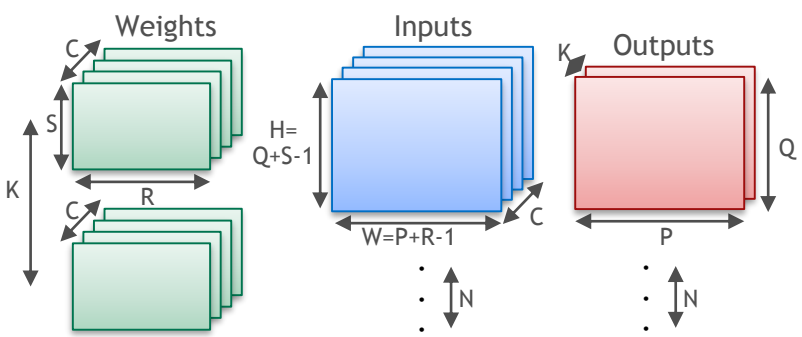
NVIDIA.

EXERCISE 6: PROBLEM

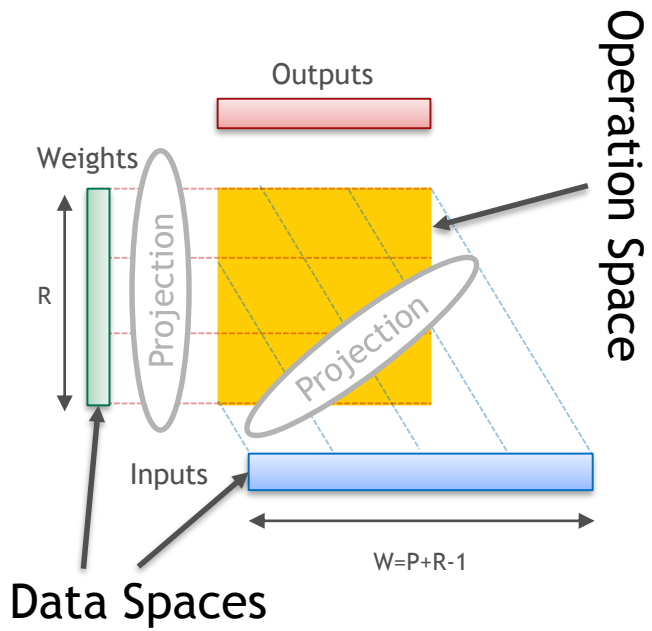
7D Convolutional Network Layer

To represent this…

```
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];
```



Think about:



And write:

```
problem:
  shape:
    name: Conv7D
    dimensions: [ C, K, R, S, P, Q, N ]
    data-spaces:
    - name: Weights
      projection: [ C, K, R, S ]
    - name: Inputs
      projection:
      - [ [N] ]
      - [ [C] ]
      - [ [Q], [S] ]
      - [ [P], [R] ]
    - name: Outputs
      projection: [ N, K, Q, P ]
      read-write: True
  R: 3
  P: 1920
```

MAPPINGS

Mapping: way in which the operation space and the associated data spaces are split into tiles at each level of the architecture's hierarchy and among multiple instances at each level

```
// === 1D Convolution Workload ===
for r=[0:R):
  for p=[0:P):
    Output[p] += Weight[r] * Input[r+p];
```

Split into tiling levels:
- R0*R1*R2*R3 = R
- P0*P1*P2*P3 = P

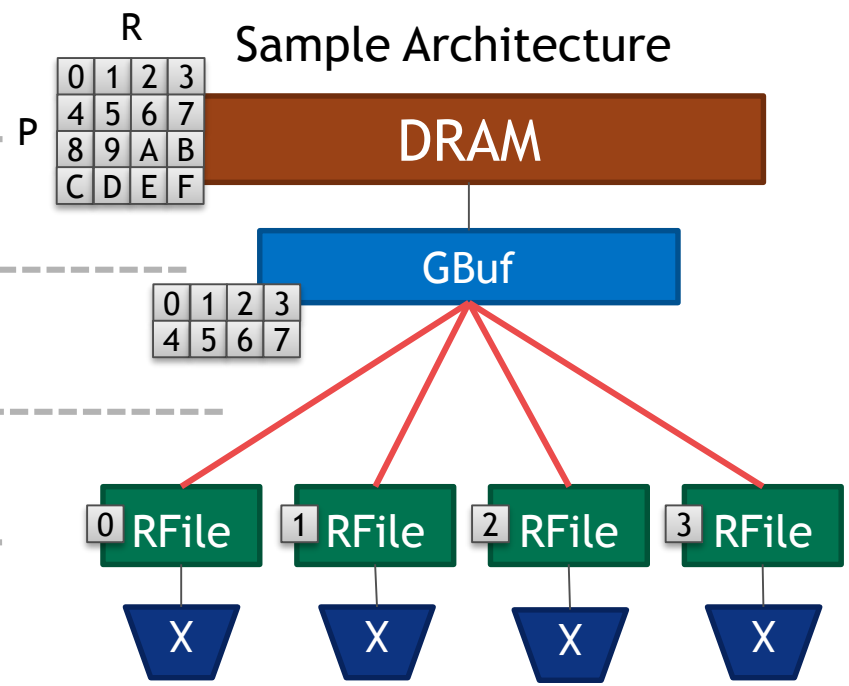
```
// === Mapping ===
// Main Memory
for r3=[0:R3):
  for p3=[0:P3):

    // Buffer
    for r2=[0:R2):
      for p2=[0:P2):

        // Spatial: GBuf->RFile
        parallel_for r1=[0:R1):
          parallel_for p1=[0:P1):

            // RFile
            for r0=[0:R0):
              for p0=[0:P0):
                p = p3*P2*P1*P0 + p2*P1*P0 + p1*P0 + p0;
                r = r3*R2*R1*R0 + r2*R1*R0 + r1*R0 + r0;
                Output[p] += Weight[r] * Input[r+p];
```

Tiling Level 3

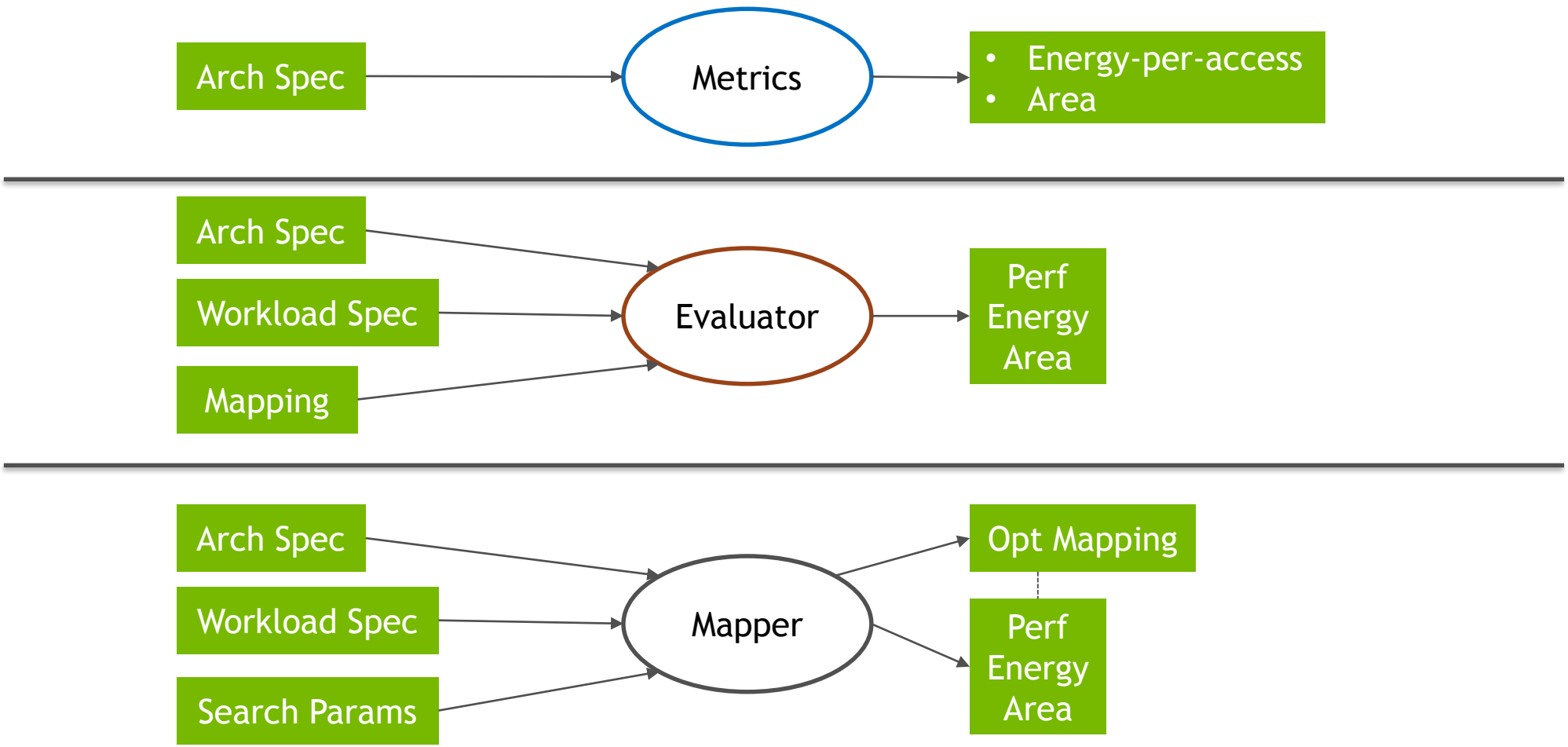
Tiling Level 2

Tiling Level 1

Tiling Level 0

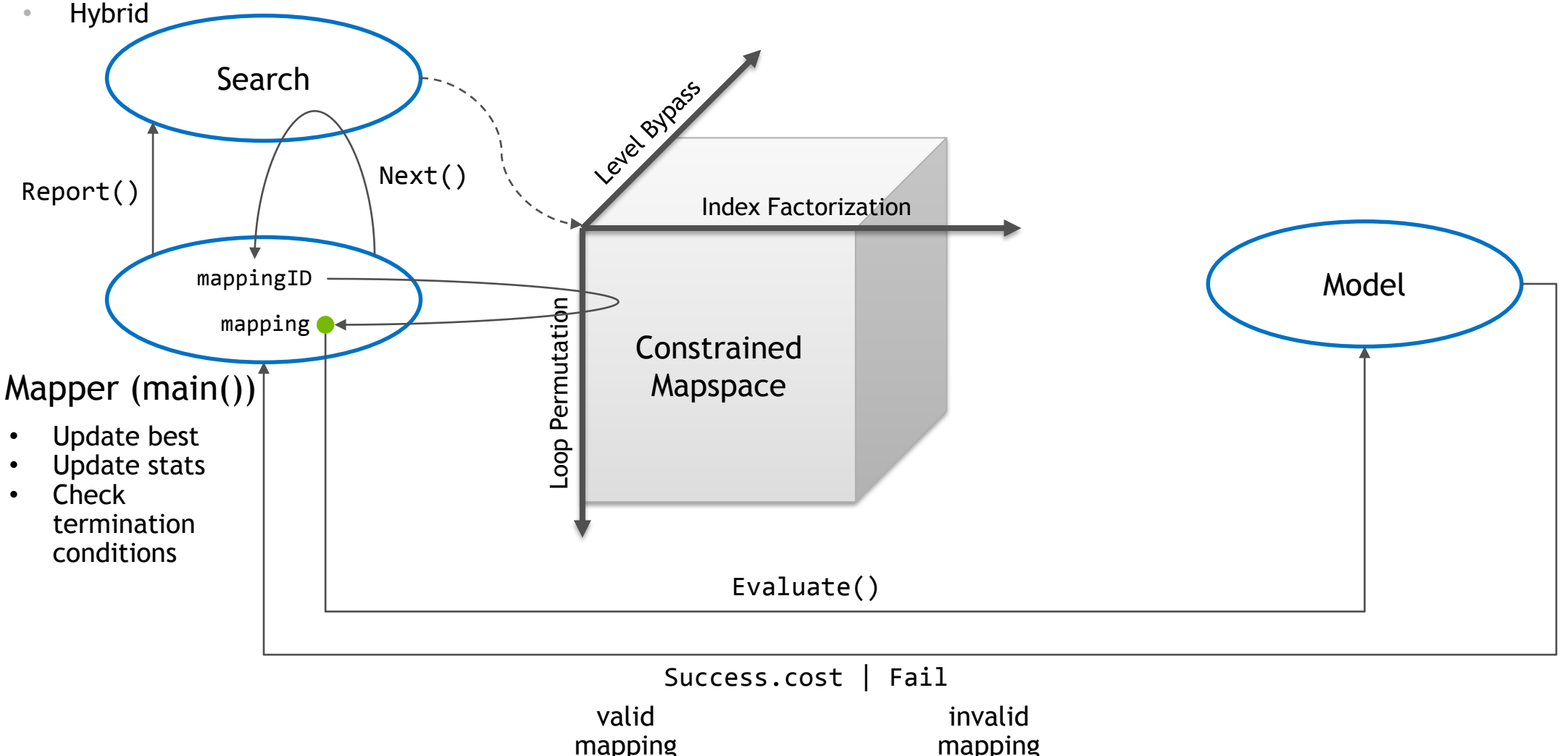


MULTIPLE TOOLS WITHIN TIMELOOP



THE MAPPER

- Linear
- Random
- Hybrid



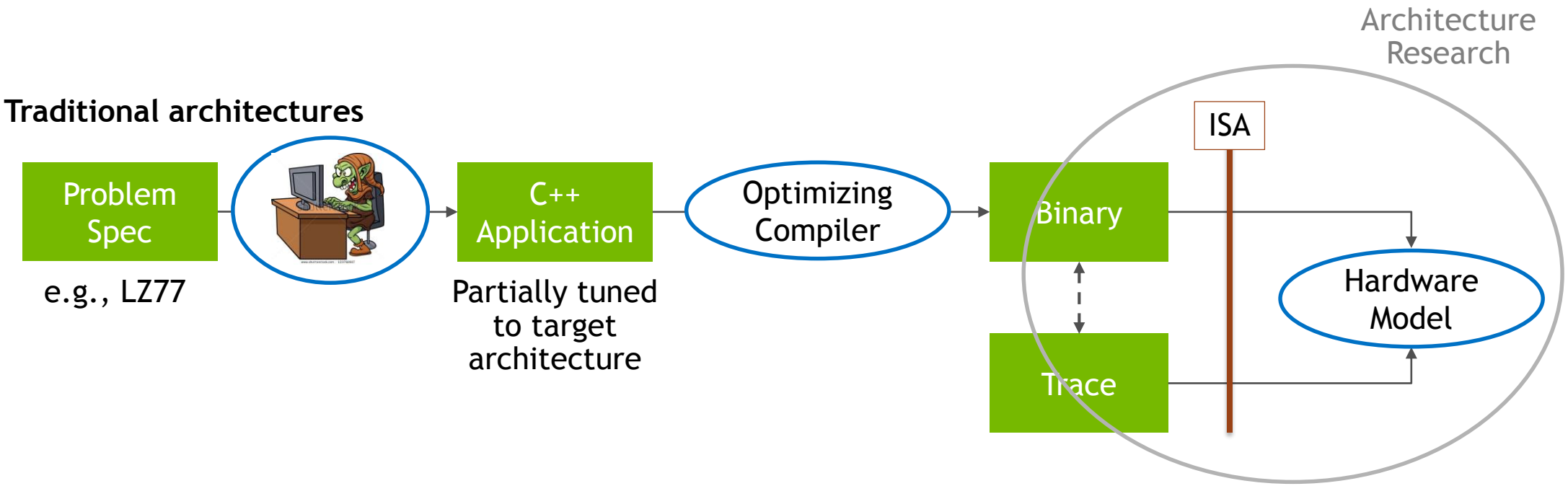
MAPPER TERMINATION CONDITIONS

Mapper termination conditions

1. Mapspace exhausted
2. #Valid mappings encountered >= “search-size”
3. #Consecutive invalid mappings encountered >= “timeout”
4. #Consecutive sub-optimal valid mappings encountered >= “victory-condition”
5. Ctrl+C


ALTERNATE INTRO

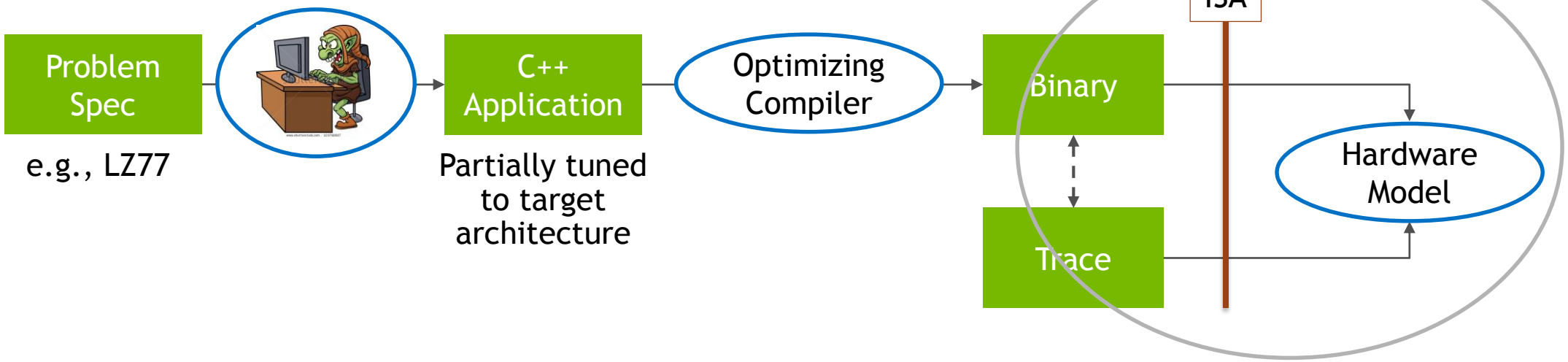
EVALUATING ARCHITECTURES



EVALUATING ARCHITECTURES

Architecture Research

Traditional architectures



Flexible accelerator architectures

e.g., DNN Layer
7D Loop Nest

Troll Programmer Photo by Unknown Author is licensed under CC BY-SA

EVALUATING ARCHITECTURES

Flexible accelerator architectures

e.g., DNN Layer
7D Loop Nest

A model needs a mapper

Troll Programmer Photo by Unknown Author is licensed under CC BY-SA