Automatic High-Performance Kernel Generation for EDDO-Style Accelerators*

Prasanth Chatarasi

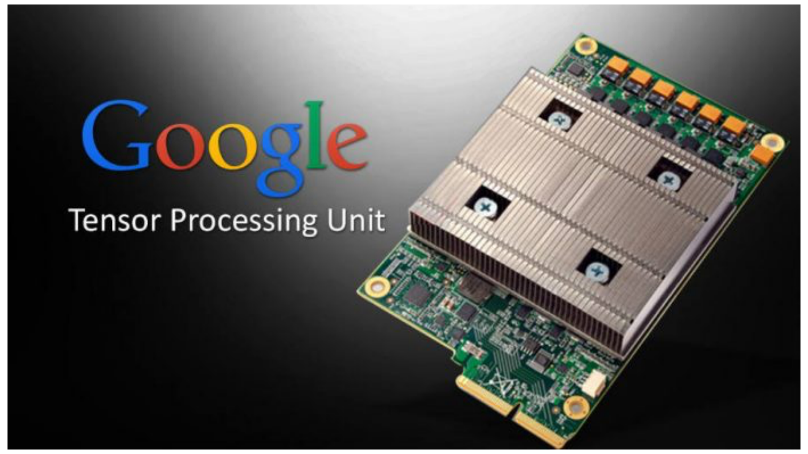
Research Staff Member @ AI Hardware Group,
IBM T.J. Watson Research Center,
YorkTown Heights, NY, USA
https://www.research.ibm.com/artificial-intelligence/hardware/
prasanth@ibm.com

SIAM Conference on Parallel Processing for Scientific Computing (PP22) , Feb 26th, 2022

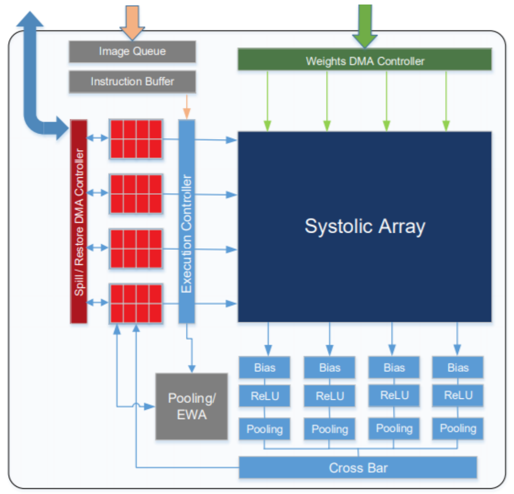
*Work done during PhD studies in Habanero Research Group
at Georgia Institute of Technology

Deep Learning (DL) Accelerators

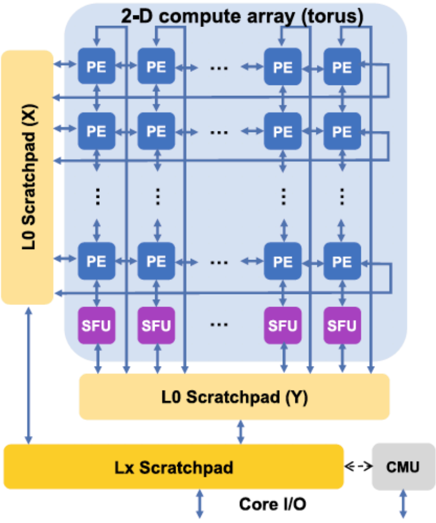
- **Emerged to address the demands of DL models training and inferences**
  - A large array of processing elements to provide high performance
  - Direct communication between PEs for energy efficiency
    - PE & PE requires ~3x less energy compared to PE & L2



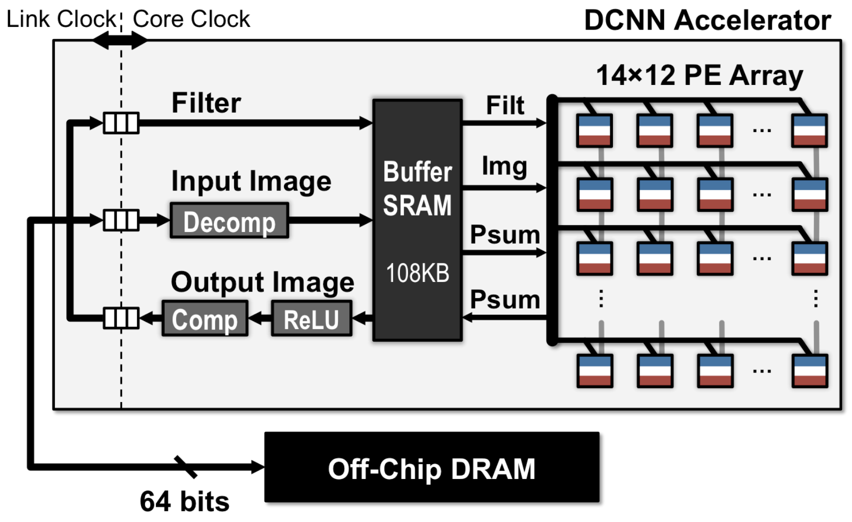
TPU, Google



xDNN, Xilinx



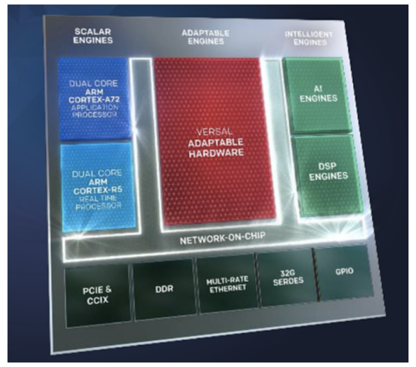
RAPID, IBM



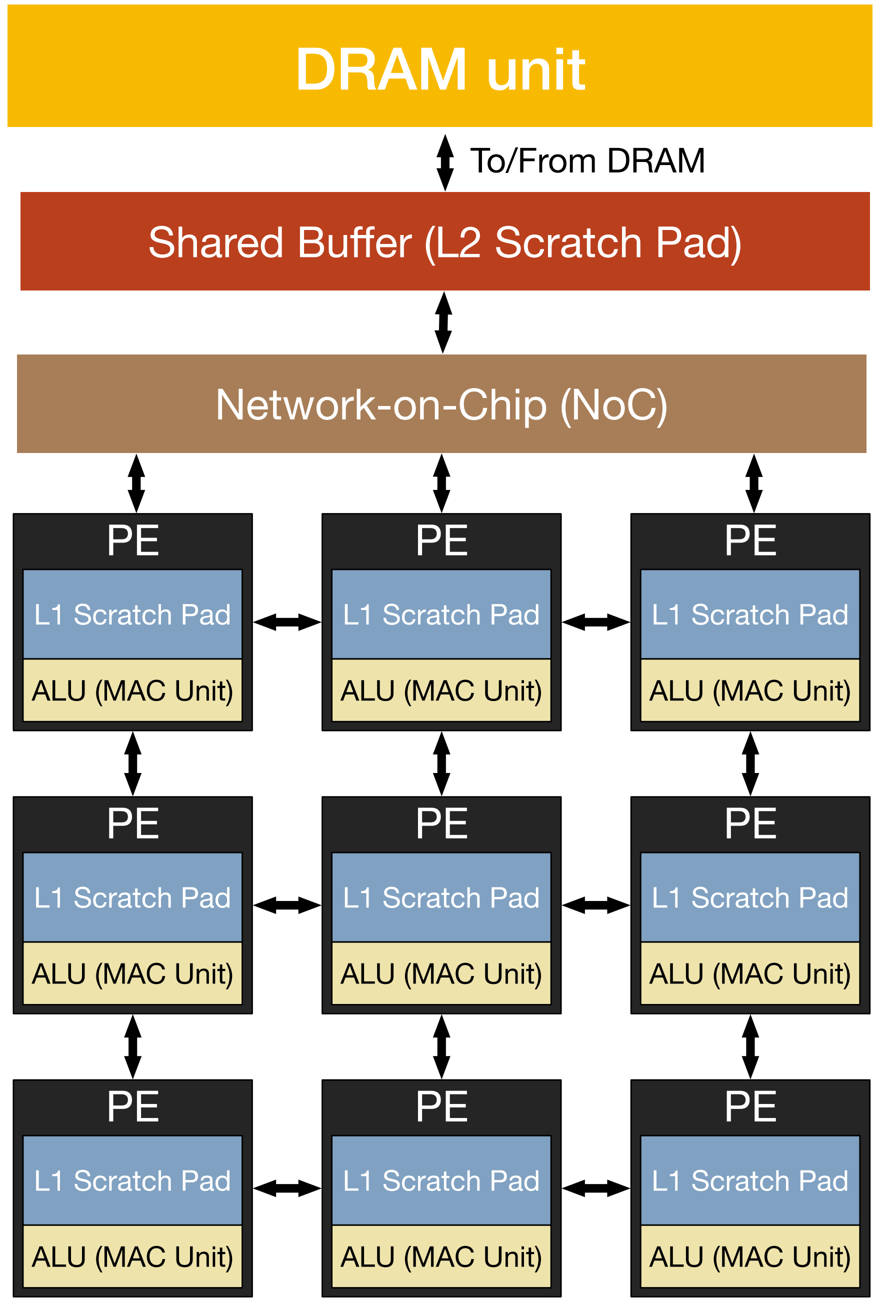
Eyeriss, MIT

NVDLA.org

DLA, NVIDIA



AI Engine, Xilinx (Versal)



Abstract template

Trend in Accelerators

- **Monolithic (coarse-grained) accelerators have served the purpose of proving higher performance and energy efficiency for specific computations (e.g., GEMM, CONV2D)**

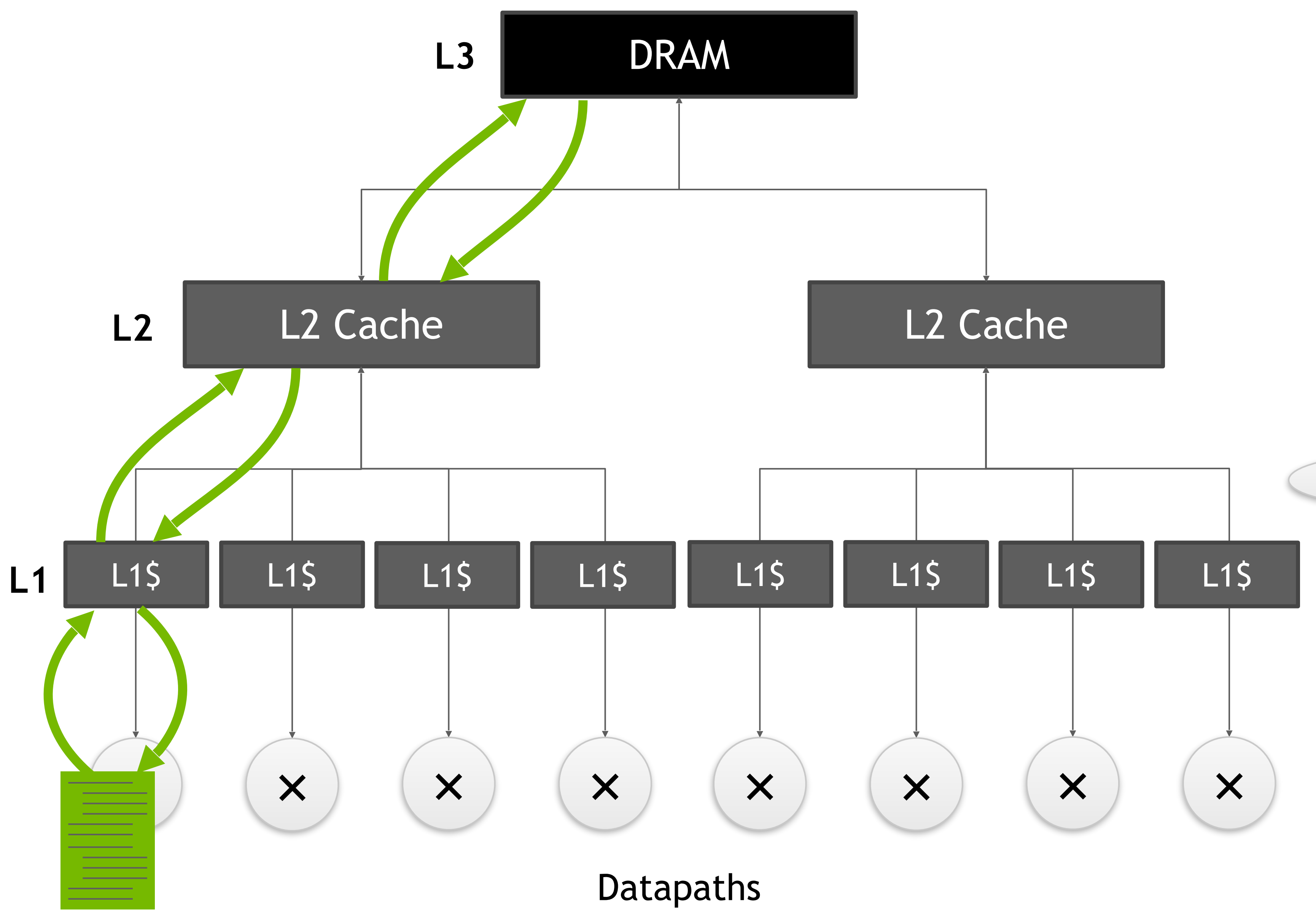
- **Challenges:**
  - **Often times, the "dataflow" is hard-wired in the accelerator**
    - Inhibits exploring different mappings for emerging workload shapes
  - **Suffer from flexibility in supporting "little more" broader computations**
    - E.g., supporting interleaving of computations (e.g., point-wise + Depth-wise)
  - **Current approach involves adding enhancements to the monolithic accelerators**
    - E.g., adding another compute unit meant for the new computations.
    - E.g., adding more SIMD units or Systolic arrays (e.g., TPU V1 vs TPU V2)
    - Hard to manage going forward with evolving operators in DL space

- **Another approach: "Programmable explicit-decoupled data orchestration accelerators"**
  - Compute and memory operations are decoupled
  - Data movement is explicit

ICDO vs EDDO Architectures

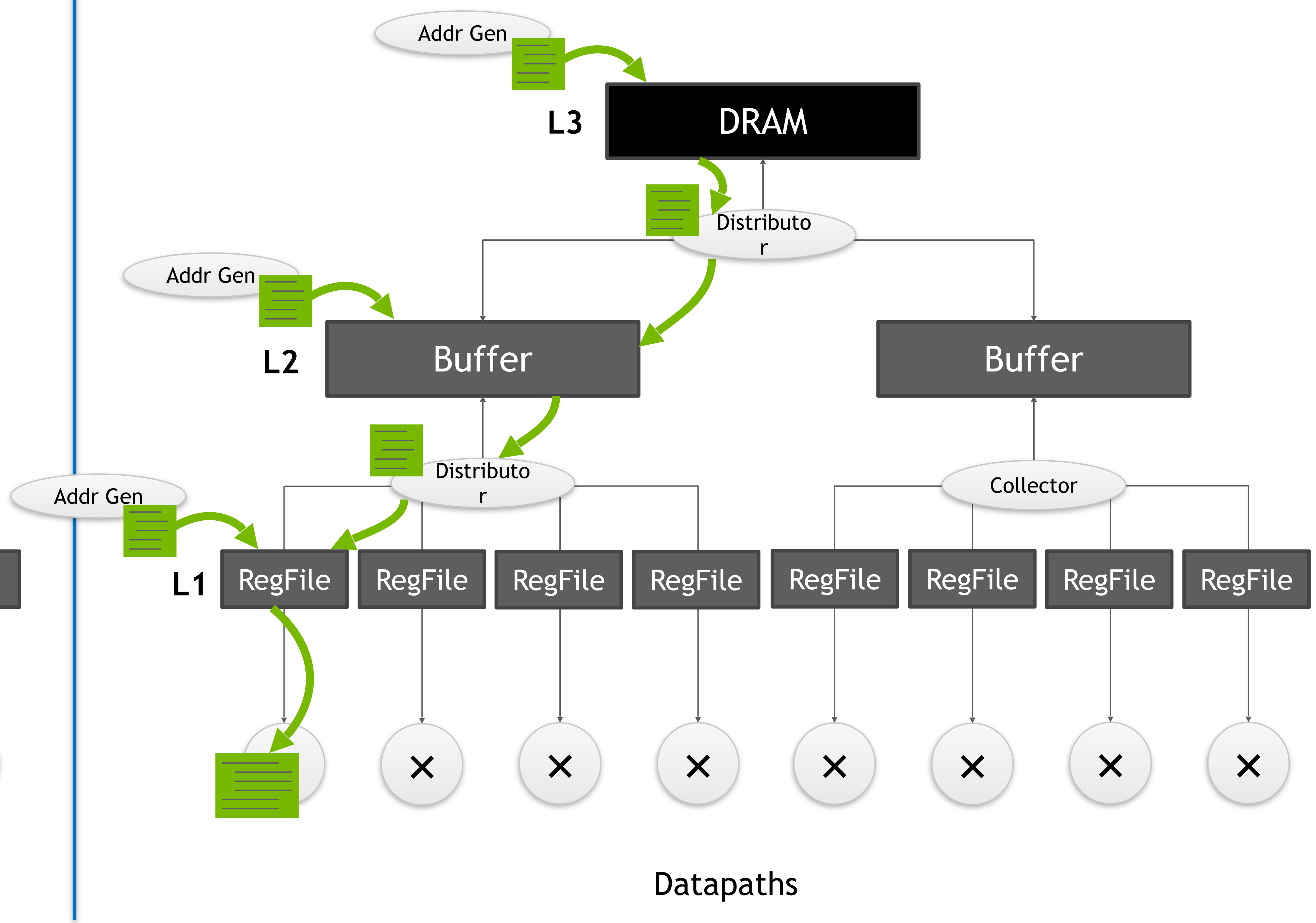
*Implicit Coupled Data Orchestration (**ICDO**)*

e.g., CPUs, GPUs



*Explicit Decoupled Data Orchestration (**EDDO**)*

e.g., IBM Rapid AI, NVIDIA Simba,
Xilinx AI Engine array etc.

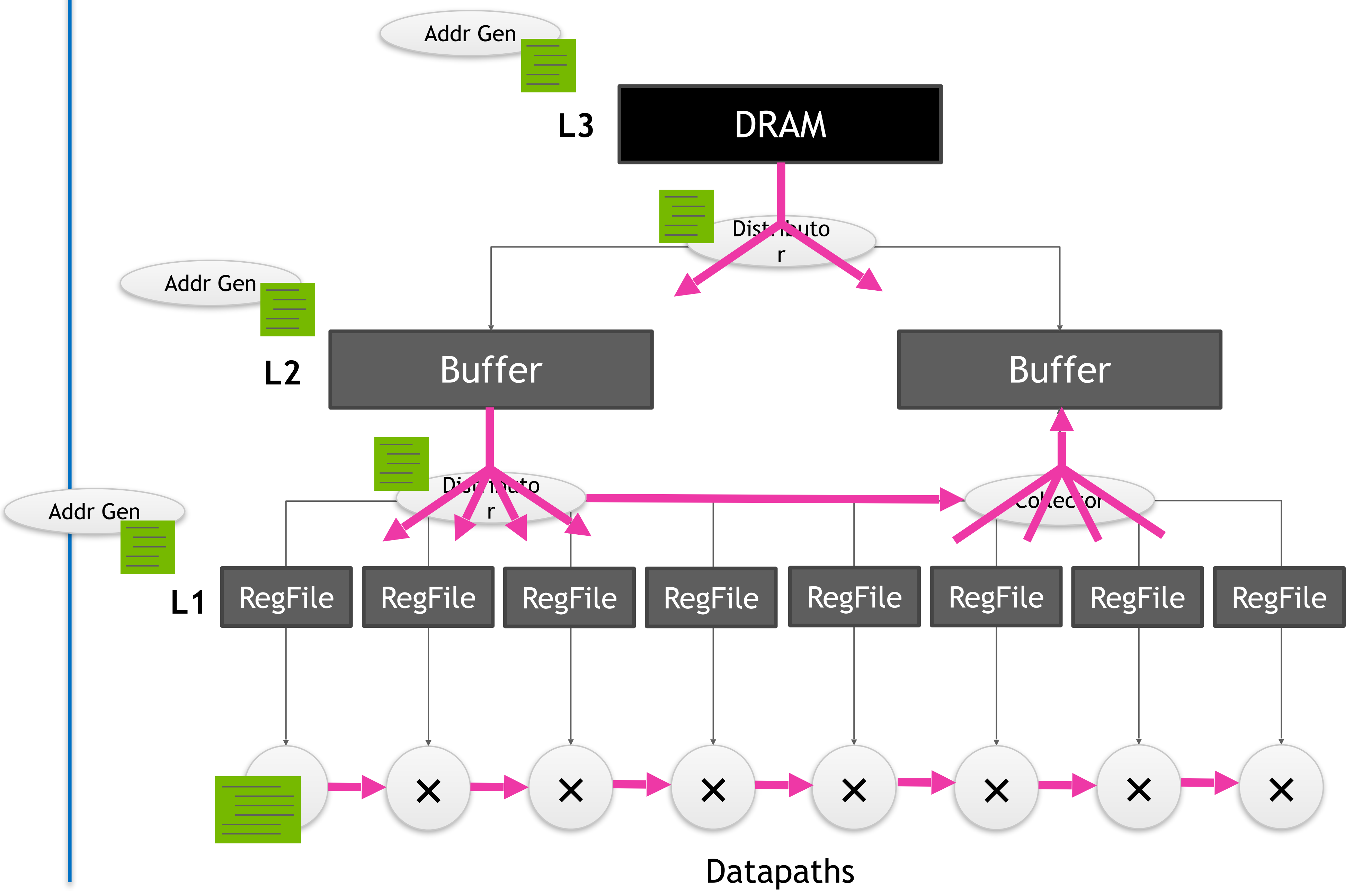


EDDO Architectures

Benefits

- Dedicated (and often statically programmed) state machines more efficient than general cores
- Perfect “prefetching”
- **Buffet** storage idiom provides fine-grain synchronization and efficient storage, or scratchpads + Send/Recv synchronization
- Hardware mechanisms for reuse

*Explicit Decoupled Data Orchestration (**EDDO**)*
e.g., IBM Rapid AI, NVIDIA Simba,
Xilinx AI Engine array etc.

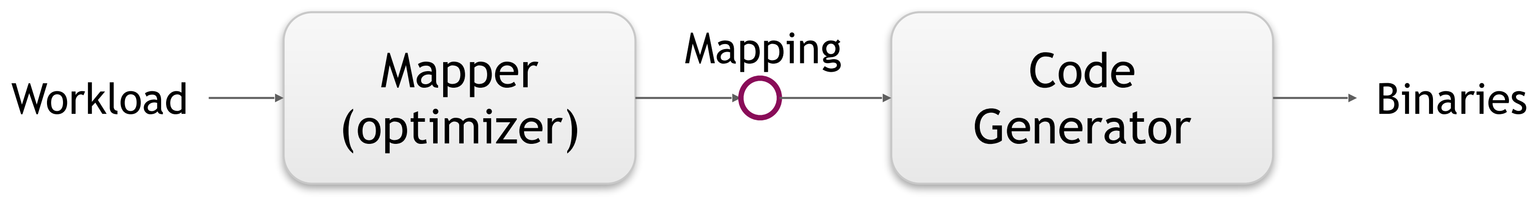


Pellauer et. al., *“Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration”*, ASPLOS 2019

Challenges: EDDO Architectures

Challenges

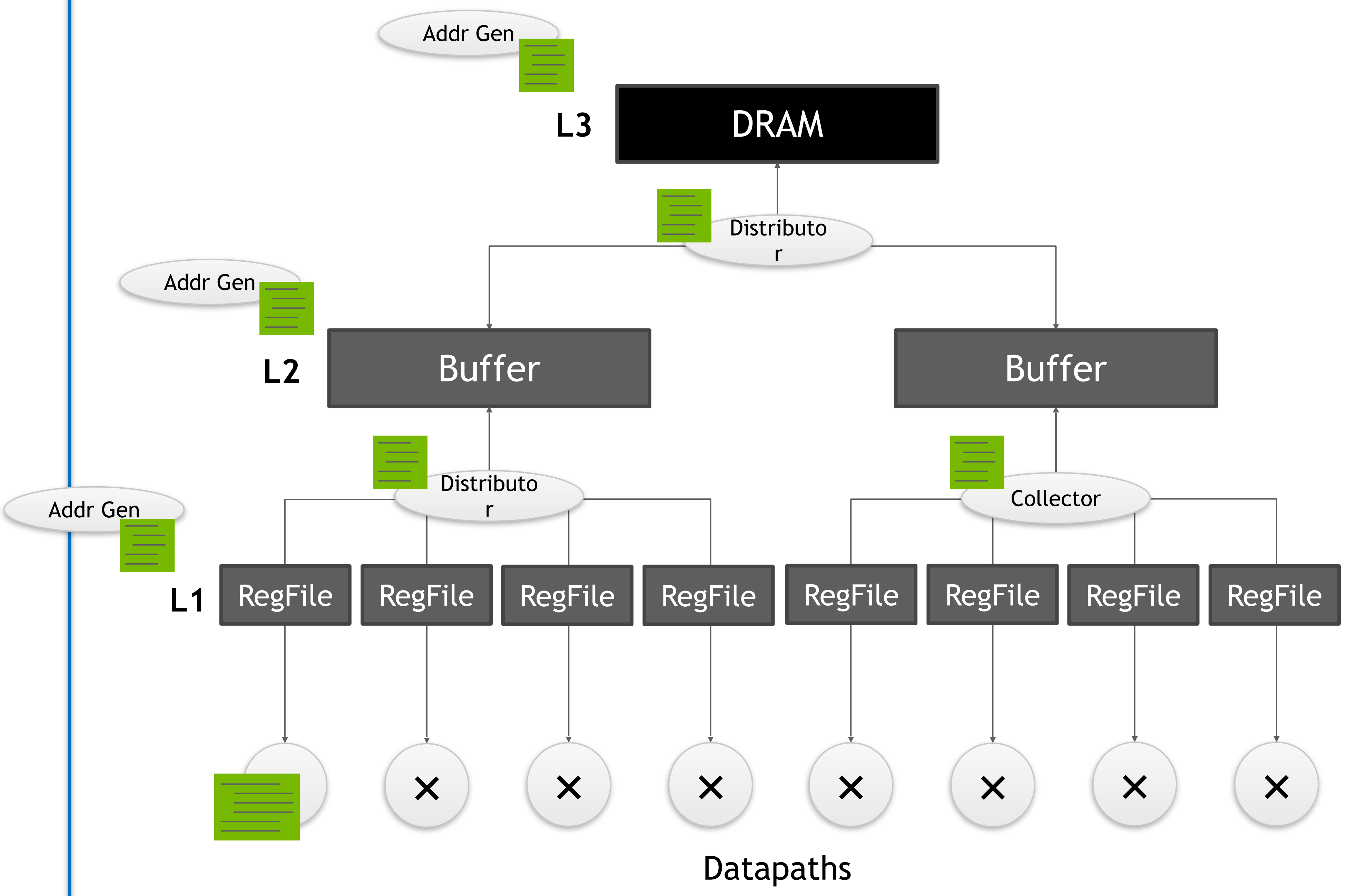
1. **No single binary:** Collection of distinct binaries that program distributed state machines working together to execute algorithm
   - E.g., CNN layer on EDDO arch → ~250 distinct state machines.
2. **Reuse optimization** is critical for efficiency
   - E.g., CNN layer on EDDO arch → 480,000 mappings, 11x spread in energy efficiency, 1 optimal mapping
   - Need an optimizer or *mapper*



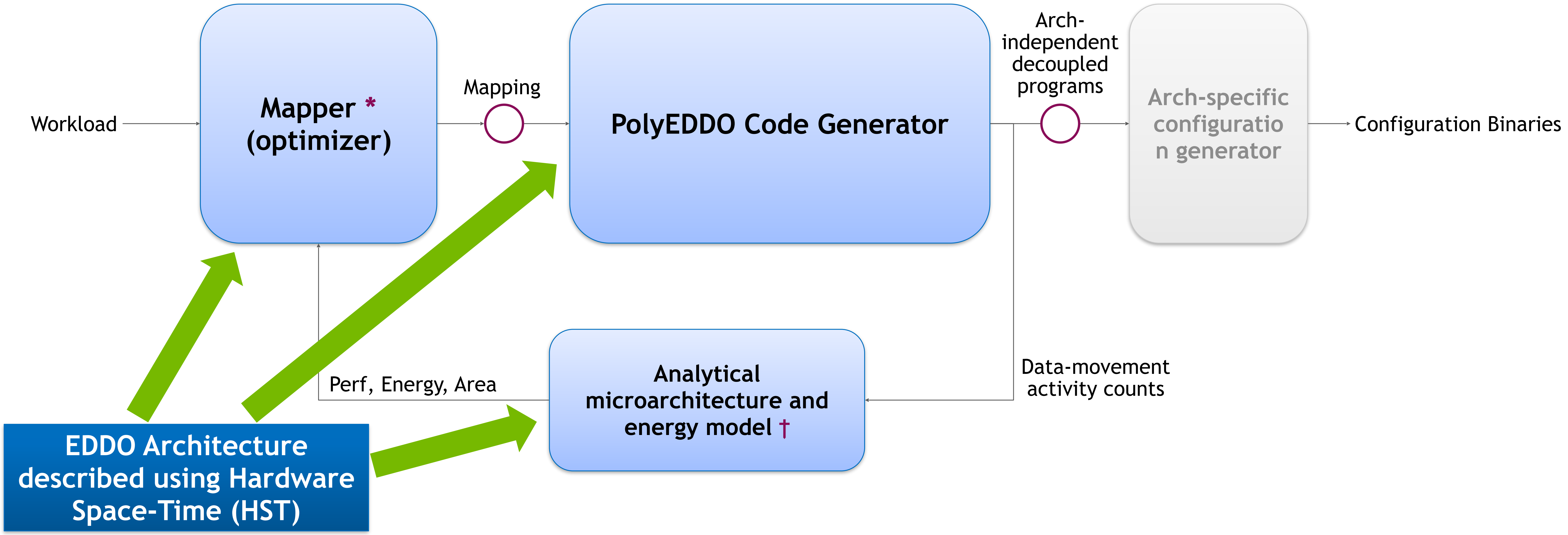
3. Variety of EDDO architectures, constantly evolving
   - Need an abstraction that Mapper and Code Generator will target

*Explicit Decoupled Data Orchestration (**EDDO**)*

e.g., IBM Rapid AI, NVIDIA Simba, Xilinx AI Engine array etc.



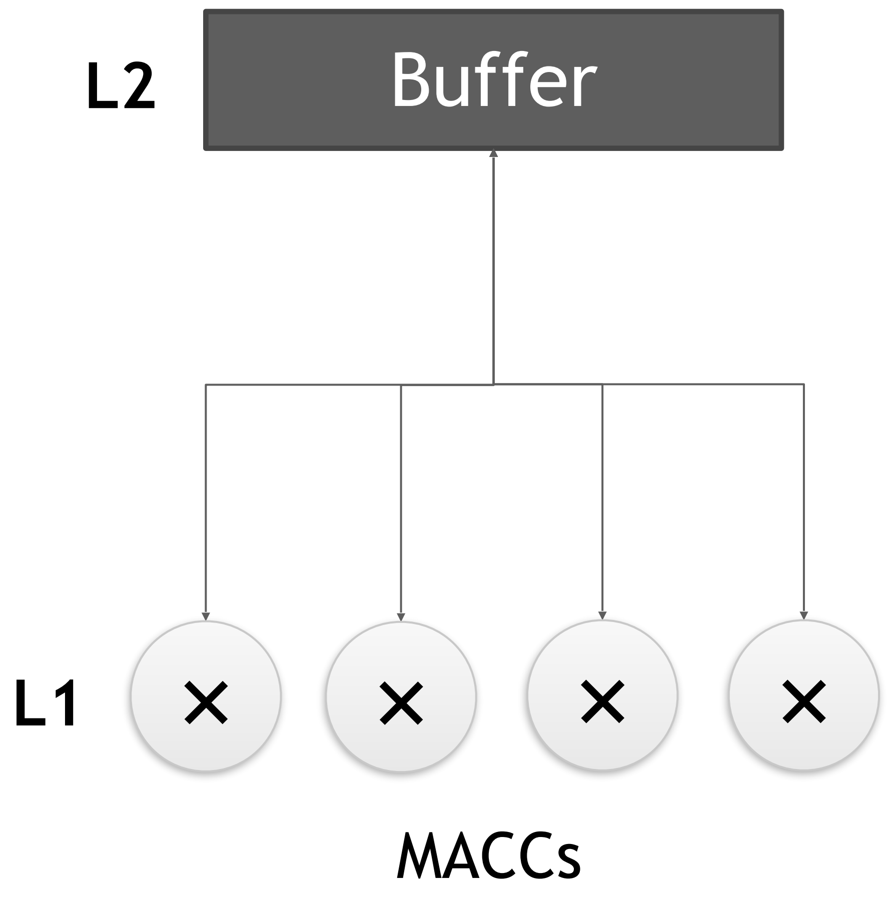
Today's talk: PolyEDDO Overall Compilation Flow

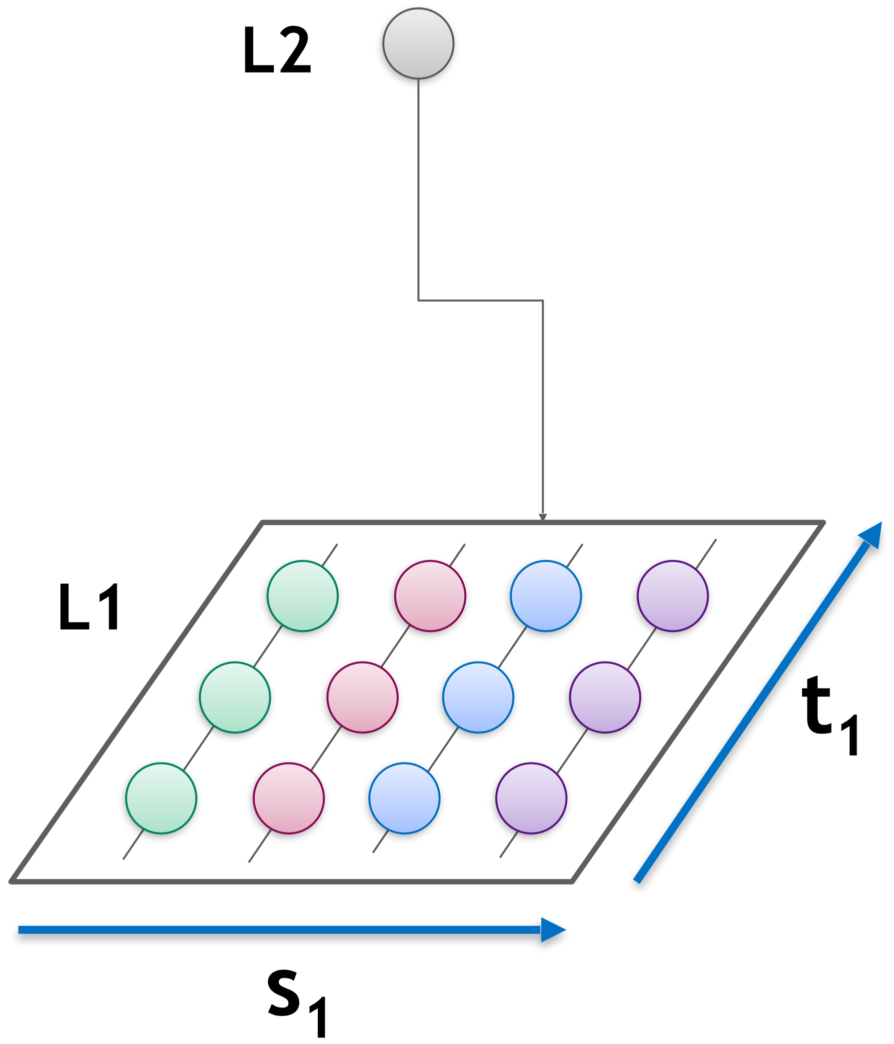


"Hardware Abstractions for targeting EDDO Architectures with the Polyhedral Model"
Angshuman Parashar, **Prasanth Chatarasi**, and Po-An Tsai,
11th International Workshop on Polyhedral Compilation Techniques (IMPACT'21)

*† Parashar et. al., *"Timeloop: Timeloop: A Systematic Approach to DNN Accelerator Evaluation"*, ISPASS 2019
† Wu et. al., *"Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs"*, ICCAD 2019

Example1 — Symbolic Hardware Space-Time (SHST)





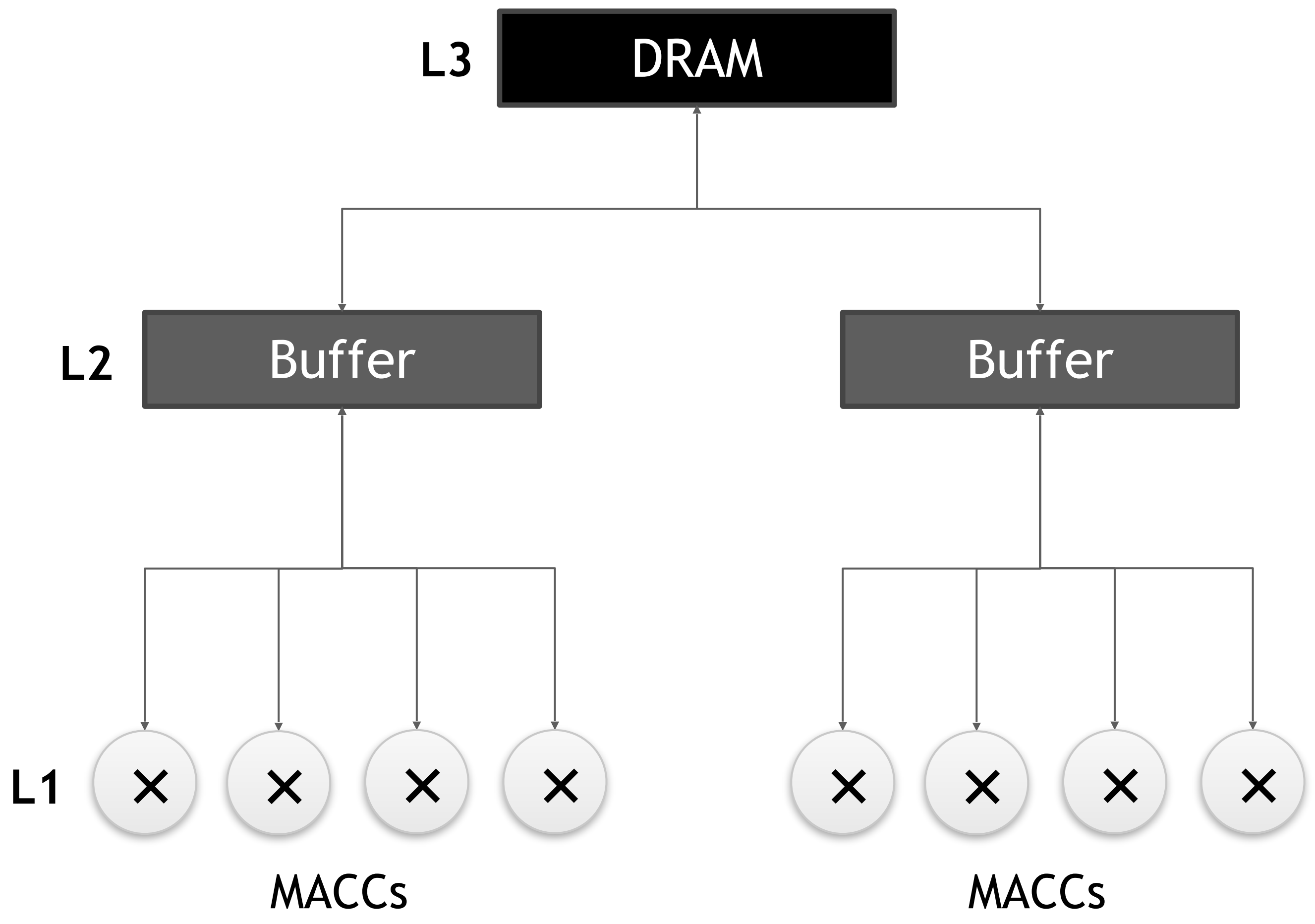
$$SpaceTime_2\ [s_2,\ t_2] \rightarrow SpaceTime_1\ [s_1,\ t_1] :$$

$$s_2 = 0\ \&\ t_2 = 0\ \&$$
$$0 \leq s_1 < 4\ \&\ 0 \leq t_1 < 3$$

Single L2, 4 L1s, 3 time-steps

- In each step, the L2 delivers a tile of data to each L1
- Across all these L1 time steps, the resident tile in L2 does not change. In effect, **time is stagnant for L2**

Example2 — Symbolic Hardware Space-Time (SHST)

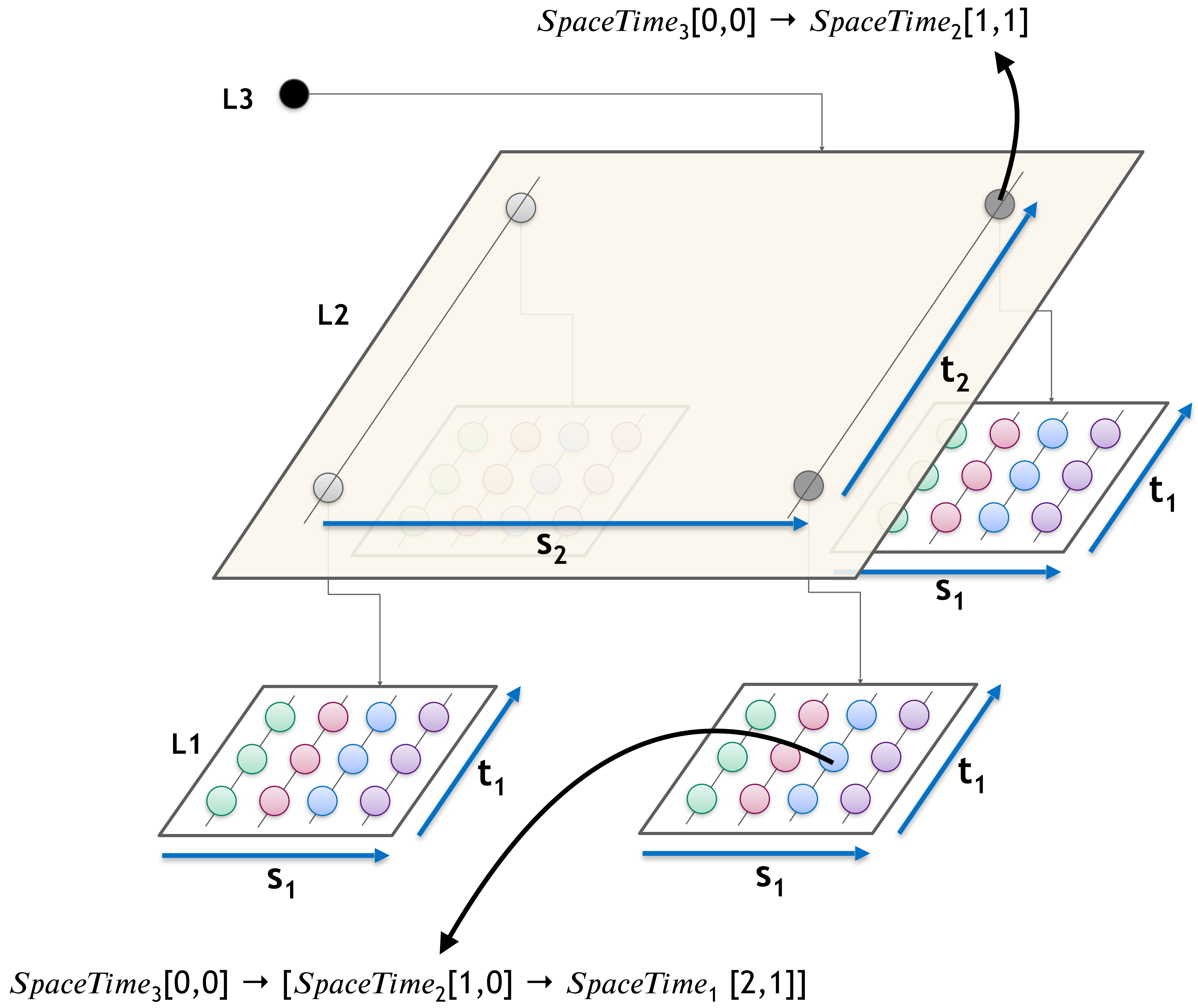


$$SpaceTime_3\,[s_3,\,t_3] \rightarrow [SpaceTime_2\,[s_2,\,t_2] \rightarrow SpaceTime_1\,[s_1,\,t_1]]:$$

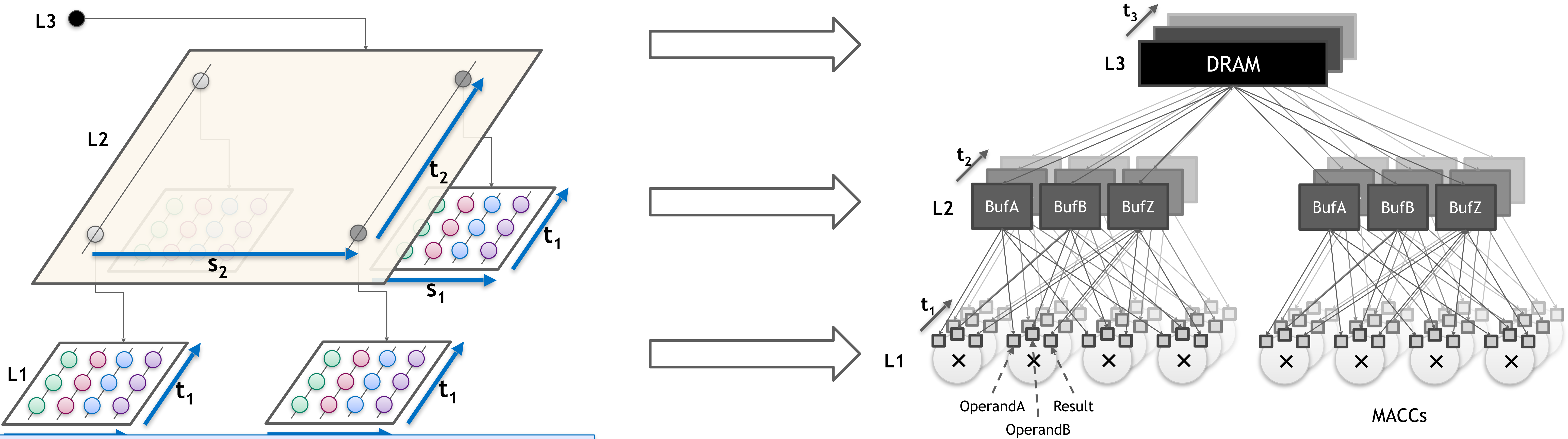
$$s_3 = 0 \qquad t_3 = 0$$

$$0 \leq s_2 < 2 \qquad 0 \leq t_2 < 2$$

$$0 \leq s_1 < 4 \qquad 0 \leq t_1 < 3$$



Example3 — Partitioned Buffers



Workload mappings target SHST

SHST $SpaceTime_3\,[s_3, t_3] \rightarrow [SpaceTime_2\,[s_2, t_2] \rightarrow SpaceTime_1\,[s_1, t_1]]$

PHST

HST

Θ^{HST}(DRAM)	=	$SpaceTime_3\,[0, 0]$	$\rightarrow$	DRAM $[s_3, t_3]$
Θ^{HST}(BufA)	=	$SpaceTime_3\,[0, 0] \rightarrow SpaceTime_2\,[s_2, t_2]$	$\rightarrow$	BufA $[s_2, t_2]$
Θ^{HST}(BufB)	=	$SpaceTime_3\,[0, 0] \rightarrow SpaceTime_2\,[s_2, t_2]$	$\rightarrow$	BufB $[s_2, t_2]$
Θ^{HST}(BufZ)	=	$SpaceTime_3\,[0, 0] \rightarrow SpaceTime_2\,[s_2, t_2]$	$\rightarrow$	BufZ $[s_2, t_2]$
Θ^{HST}(OperandA)	=	$SpaceTime_3\,[0, 0] \rightarrow [SpaceTime_2\,[s_2, t_2] \rightarrow SpaceTime_1\,[s_1, t_1]]$	$\rightarrow$	OperandA $[2s_2 + s_1, t_2, t_1]$
Θ^{HST}(OperandB)	=	$SpaceTime_3\,[0, 0] \rightarrow [SpaceTime_2\,[s_2, t_2] \rightarrow SpaceTime_1\,[s_1, t_1]]$	$\rightarrow$	OperandB $[2s_2 + s_1, t_2, t_1]$
Θ^{HST}(Result)	=	$SpaceTime_3\,[0, 0] \rightarrow [SpaceTime_2\,[s_2, t_2] \rightarrow SpaceTime_1\,[s_1, t_1]]$	$\rightarrow$	Result $[2s_2 + s_1, t_2, t_1]$

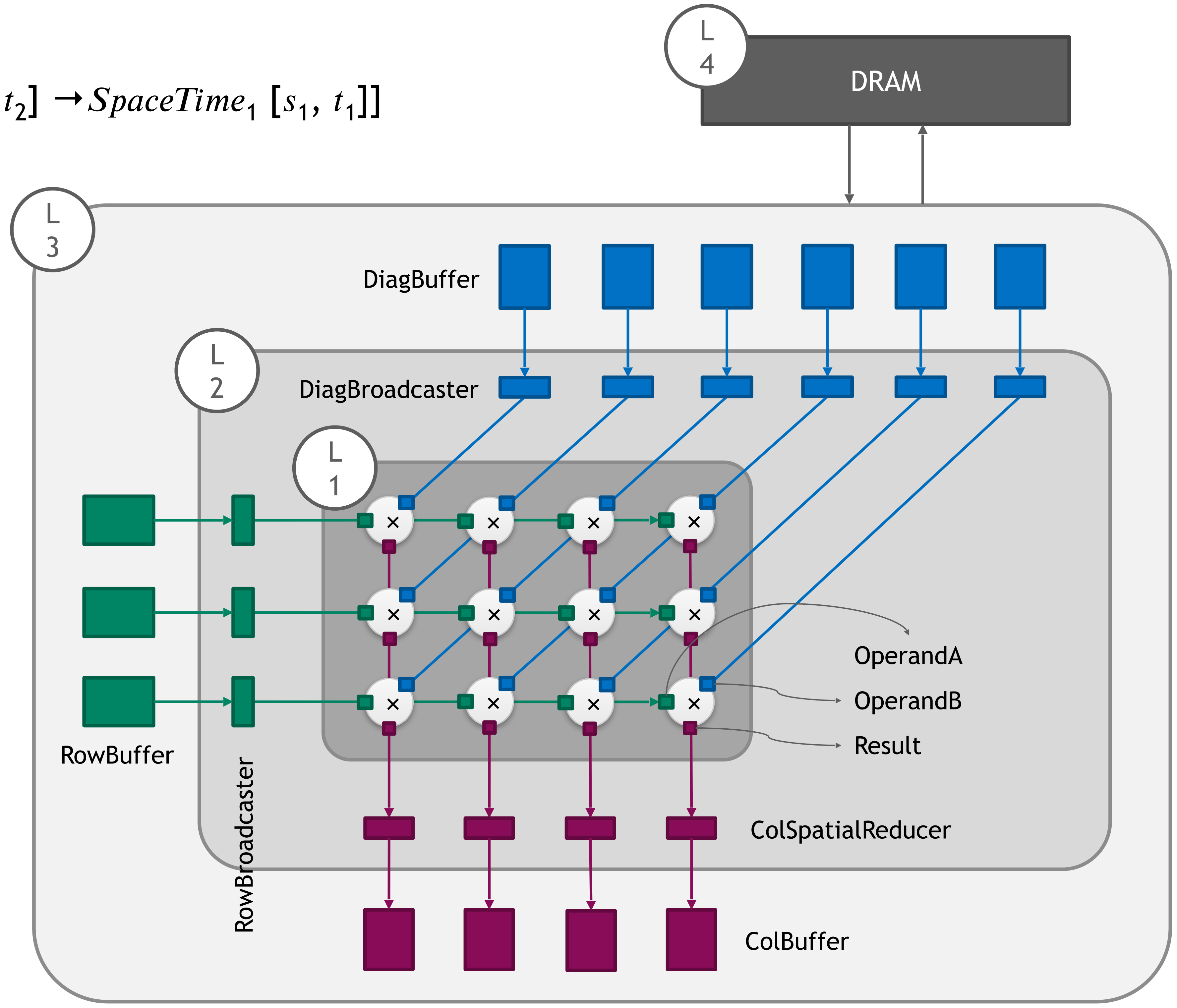
Example4 — Eyeriss-like accelerator

SHST: $SpaceTime_4\ [s_4, t_4] \rightarrow SpaceTime_3\ [s_3, t_3] \rightarrow [SpaceTime_2\ [s_2, t_2] \rightarrow SpaceTime_1\ [s_1, t_1]]$

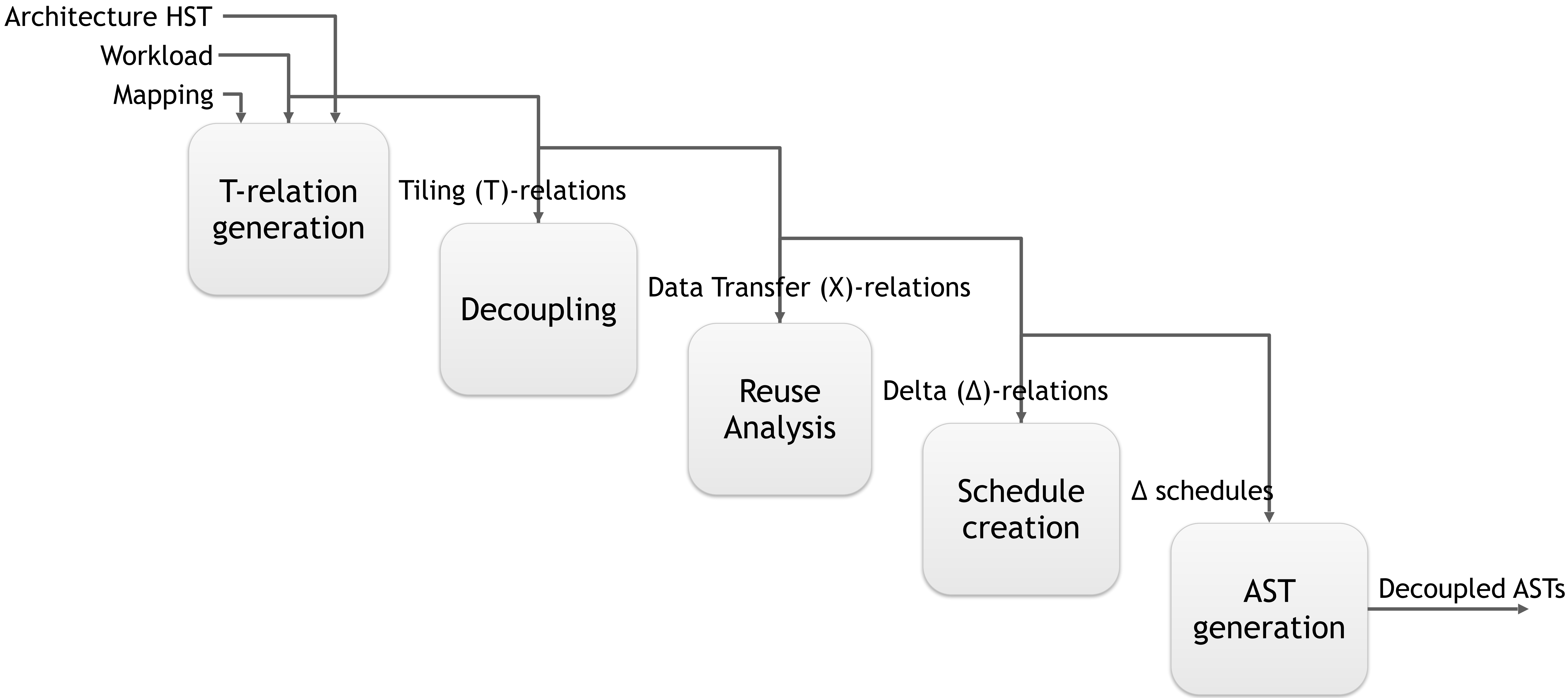
See paper for full HST

Observe how different the architecture is from CPUs and GPUs

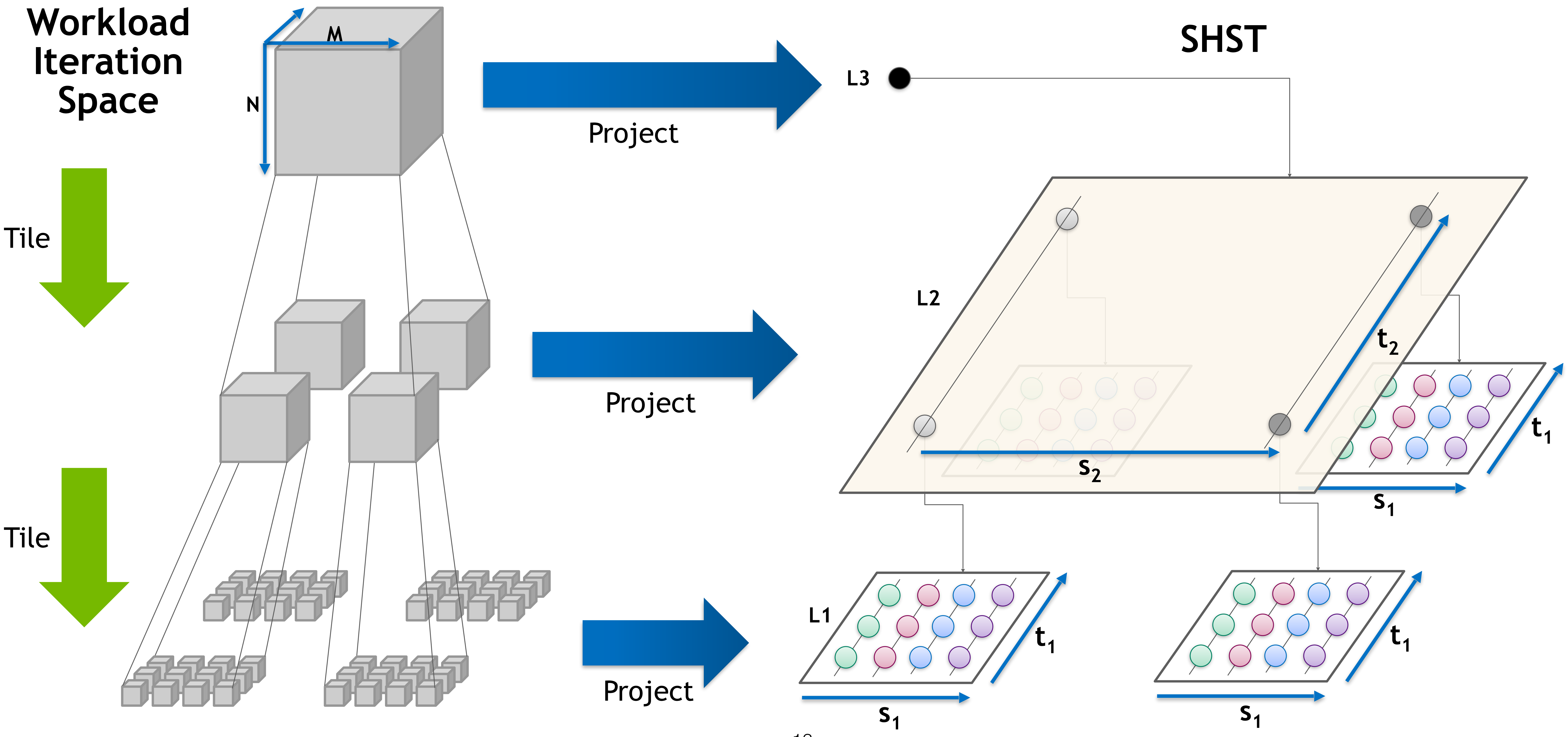
Workload mappings target SHST



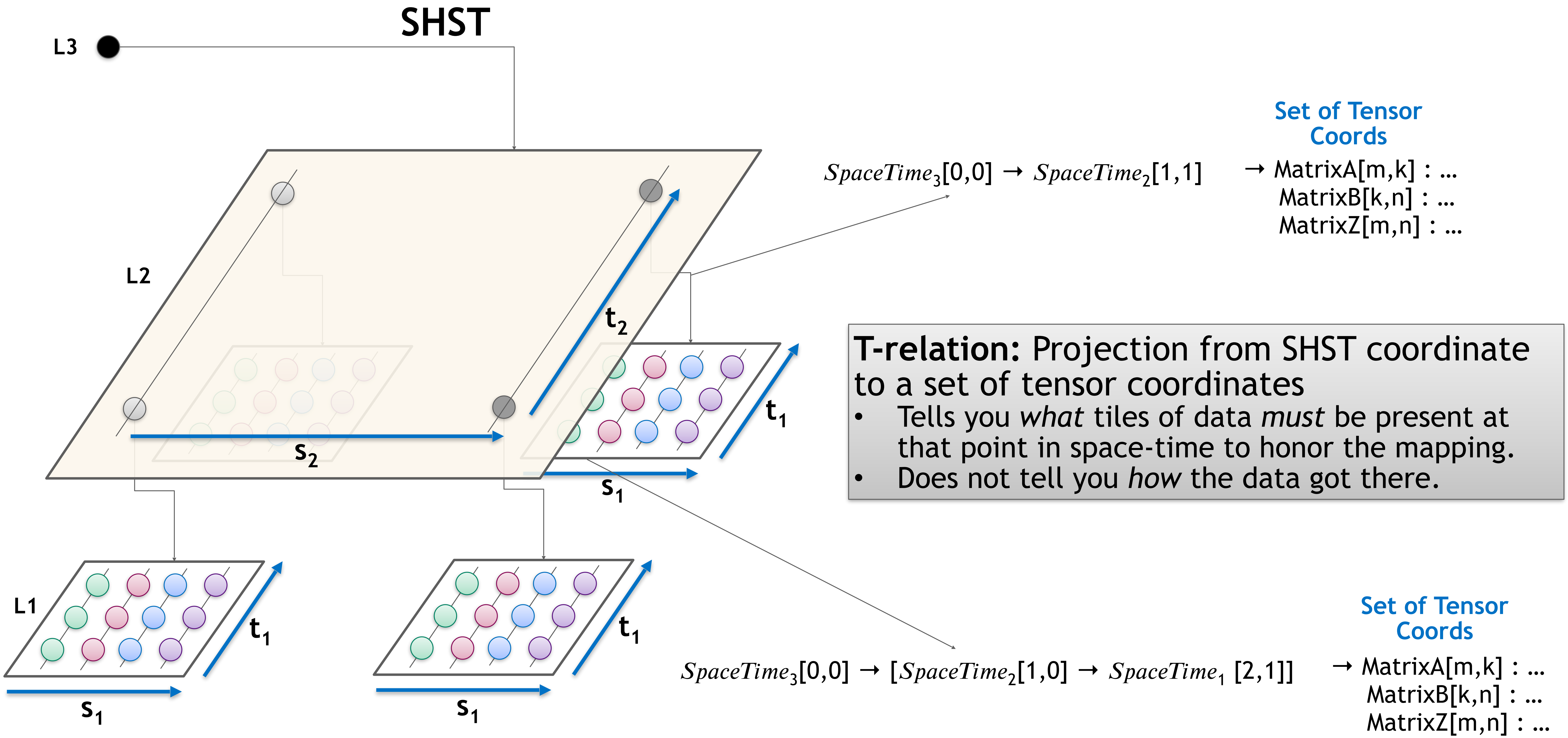
PolyEDDO Code Generator



Mapping workloads (Tensor operations)



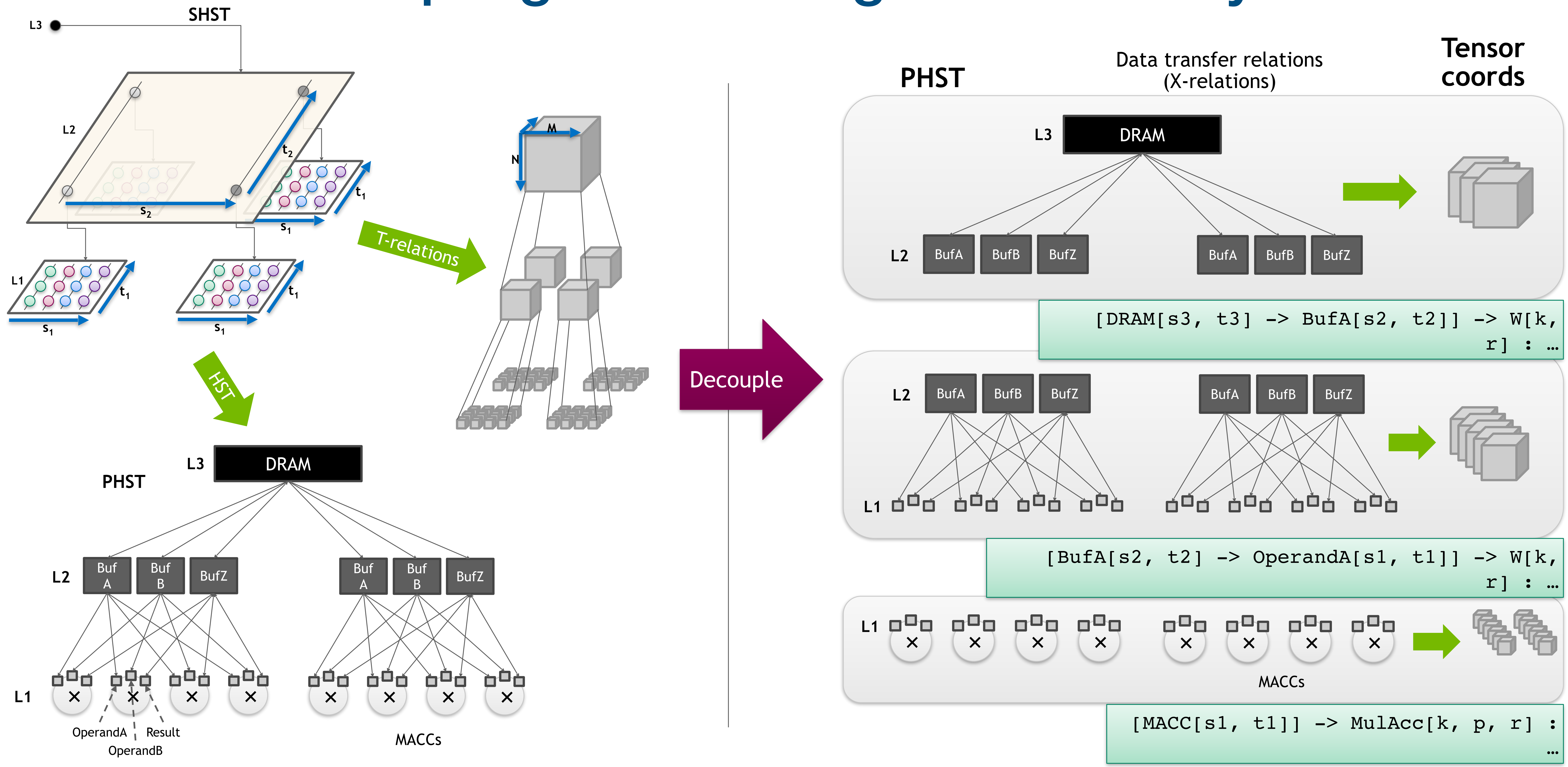
Mapping workloads (The Tiling-relation, T-relation)



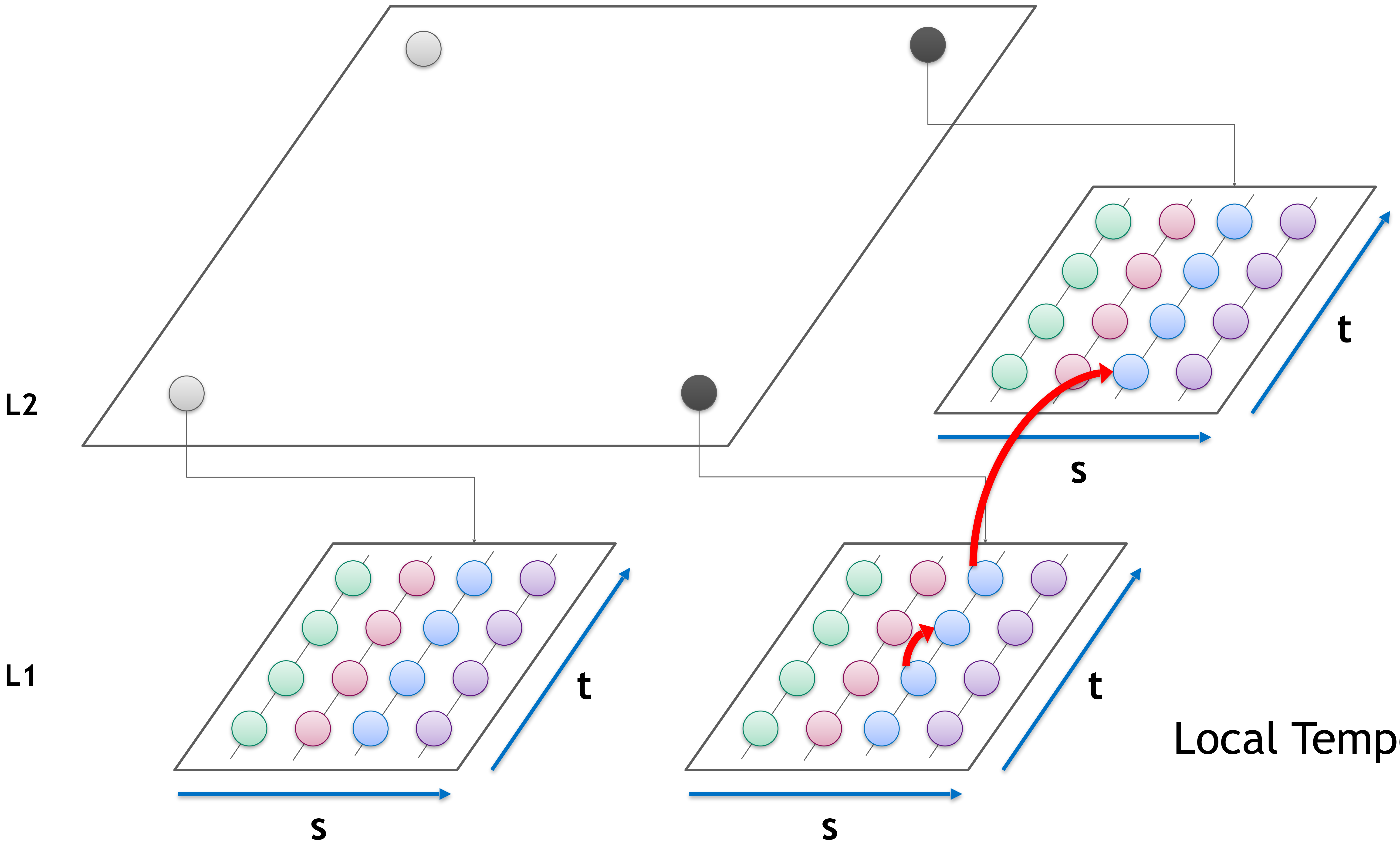
T-relation: Projection from SHST coordinate to a set of tensor coordinates

- Tells you *what* tiles of data *must* be present at that point in space-time to honor the mapping.
- Does not tell you *how* the data got there.

Decoupling — Breaking the hierarchy

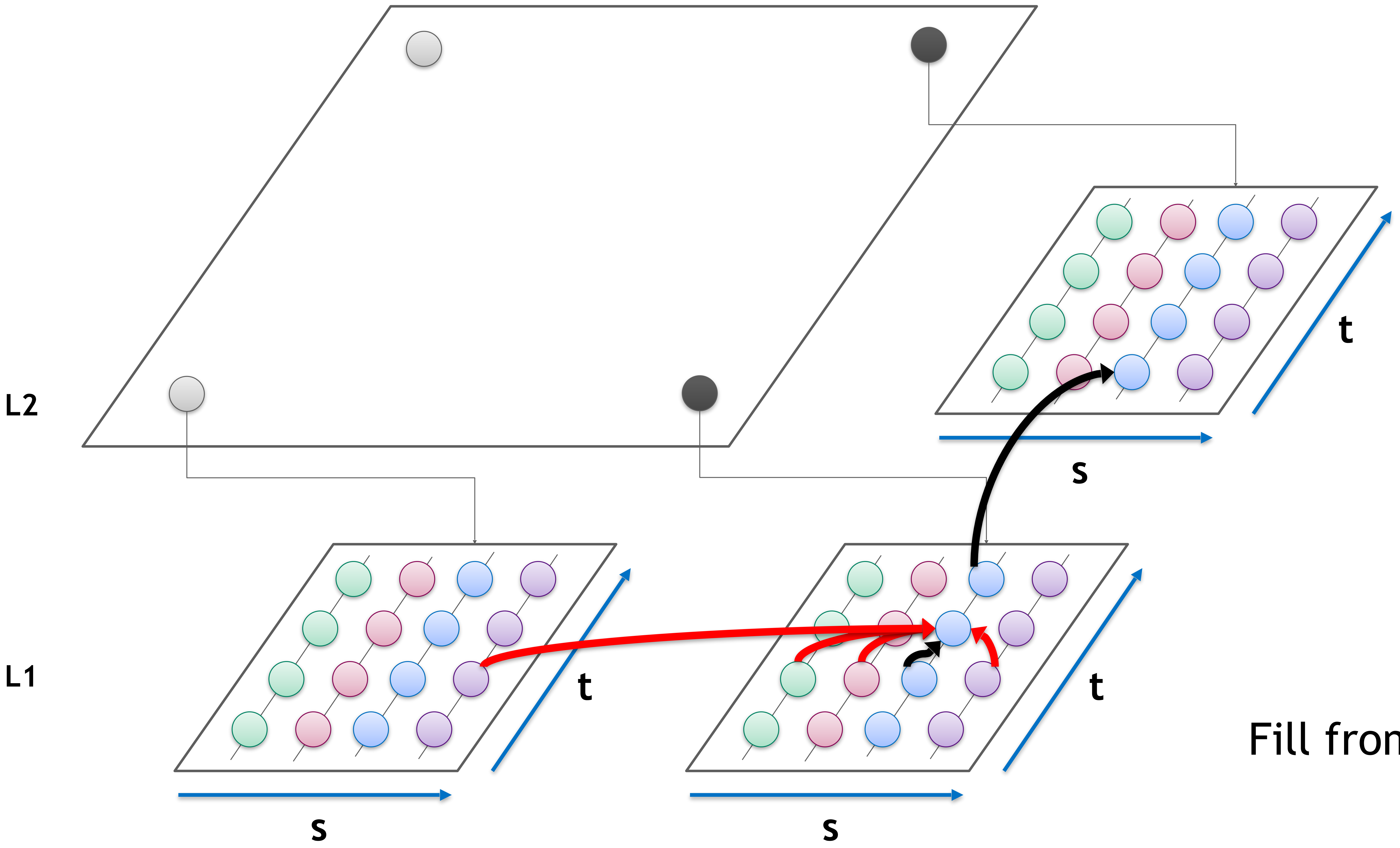


REUSE ANALYSIS



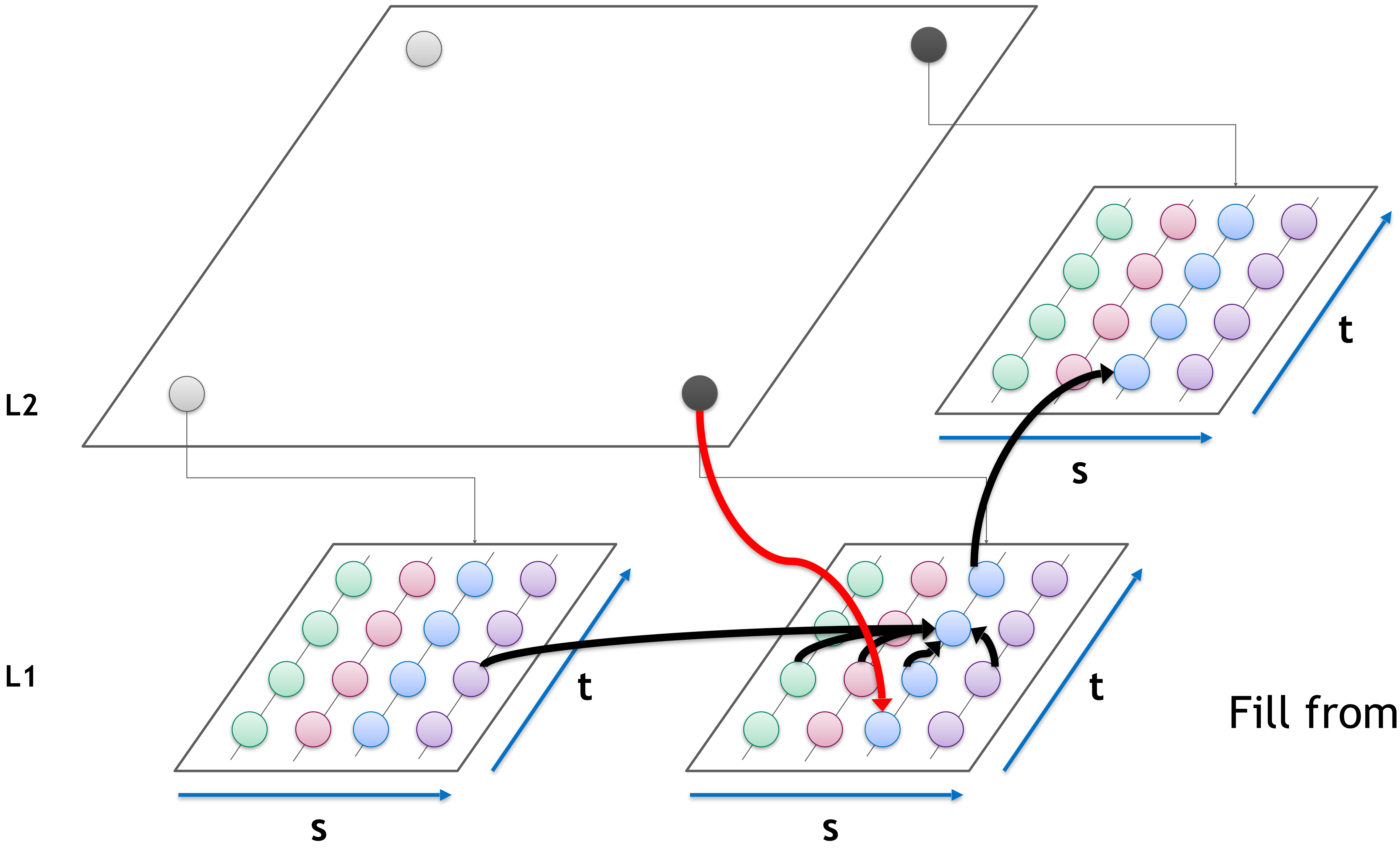
Local Temporal Reuse

REUSE ANALYSIS



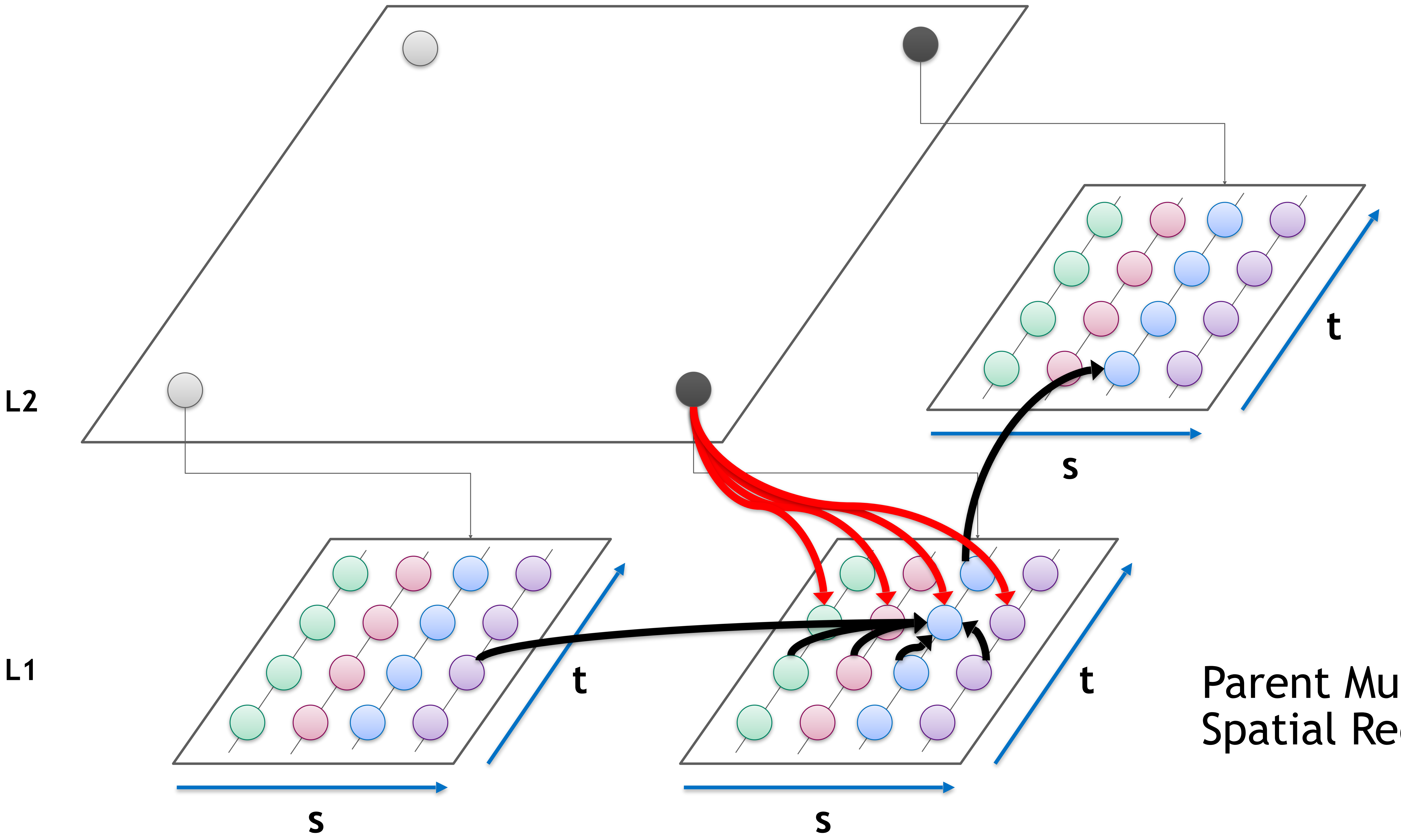
Fill from Peer

REUSE ANALYSIS



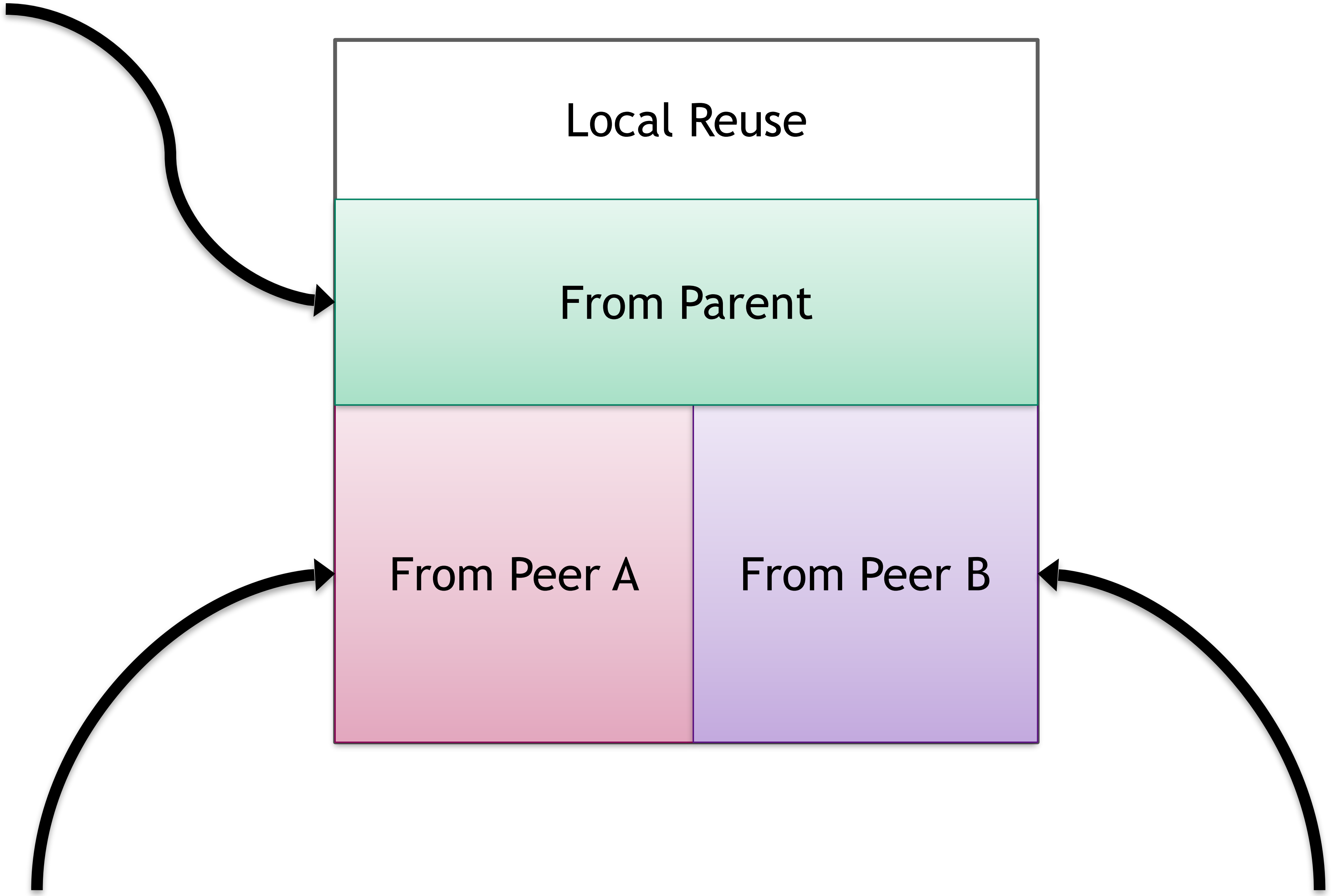
Fill from parent

REUSE ANALYSIS



Parent Multicast/
Spatial Reduction

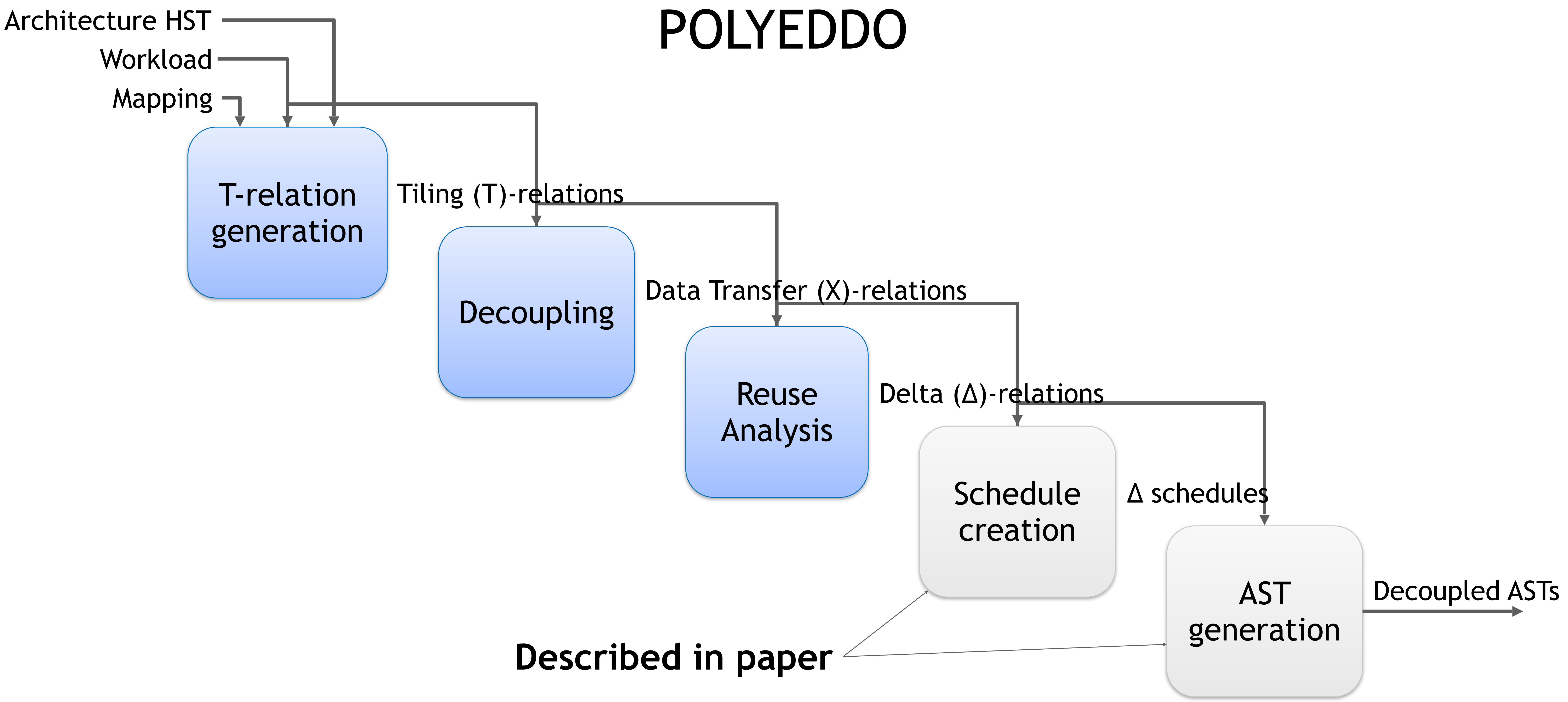
OPTIMIZATION PROBLEM (FOR A SINGLE MAPPING!)



Options:

1. Enumerate all possibilities and find optimum solution
2. Use a heuristic
3. Expose choices to mapping (and thereby the mapspace)

POLYEDDO


Architecture HST
Workload
Mapping
T-relation generation
Tiling (T)-relations
Decoupling
Data Transfer (X)-relations
Reuse Analysis
Delta (Δ)-relations
Schedule creation
Δ schedules
AST generation
Decoupled ASTs
Described in paper

EXAMPLE OUTPUT

```
// Program to read Weights from DRAM into RowBuffer.
if (P >= 1)
  for (int c3 = 0; c3 <= min(15, K - 1); c3 += 1)
    for (int c4 = 0; c4 <= min(2, R - 1); c4 += 1)
      ACTION_READ("DRAM", "DRAM", "RowBuffer", "Weights", 2)(0, 0, c4, 0, c3, c4);

// Program to read Inputs from DRAM into DiagBuffer.
if (K >= 1 && P >= 1 && R >= 1)
  for (int c3 = 0; c3 <= min(min(min(15, P + 1), P + R - 2), R + 12); c3 += 1)
    ACTION_READ("DRAM", "DRAM", "DiagBuffer", "Inputs", 1)(0, 0, c3, 0, c3);

// Program to read Outputs from DRAM into ColBuffer.
if (R >= 1)
  for (int c3 = 0; c3 <= min(15, K - 1); c3 += 1)
    for (int c4 = 0; c4 <= min(13, P - 1); c4 += 1)
      ACTION_READ_IU("DRAM", "DRAM", "ColBuffer", "Outputs", 2)(0, 0, c4, 0, c3, c4);

// Program to read Weights from RowBuffer into RowBroadcaster.
if (P >= 1) {
  for (int c2 = 0; c2 <= min(15, K - 1); c2 += 1)
    for (int c4 = 0; c4 <= min(2, R - 1); c4 += 1)
      ACTION_READ("RowBuffer", "RowBuffer", "RowBroadcaster", "Weights", 2)(c4, 0, c4, c2, c2, c4);
  for (int c3 = 0; c3 <= min(15, K - 1); c3 += 1)
    for (int c4 = 0; c4 <= min(2, R - 1); c4 += 1)
      ACTION_SHRINK("RowBuffer", "RowBuffer", "Weights", 2)(0, 0, c4, 0, c3, c4);
}

// Program to read Inputs from DiagBuffer into DiagBroadcaster.
if (K >= 1 && P >= 1 && R >= 1) {
  for (int c3 = 0; c3 <= min(min(min(15, P + 1), P + R - 2), R + 12); c3 += 1)
    ACTION_READ("DiagBuffer", "DiagBuffer", "DiagBroadcaster", "Inputs", 1)(c3, 0, c3, 0, c3);
  for (int c3 = 0; c3 <= min(min(min(15, P + 1), P + R - 2), R + 12); c3 += 1)
    ACTION_SHRINK("DiagBuffer", "DiagBuffer", "Inputs", 1)(0, 0, c3, 0, c3);
}

// Program to read Outputs from ColBuffer into ColSpatialReducer.
if (R >= 1) {
  for (int c2 = 0; c2 <= min(15, K - 1); c2 += 1)
    for (int c4 = 0; c4 <= min(13, P - 1); c4 += 1)
      ACTION_READ_IU("ColBuffer", "ColBuffer", "ColSpatialReducer", "Outputs", 2)(c4, 0, c4, c2, c2, c4);
  for (int c3 = 0; c3 <= min(15, K - 1); c3 += 1)
    for (int c4 = 0; c4 <= min(13, P - 1); c4 += 1)
      ACTION_UPDATE("ColBuffer", "DRAM", "ColBuffer", "Outputs", 2)(0, 0, c4, 0, c3, c4);
}

// Program to read Weights from RowBroadcaster into OperandA.
```

```
// Program to read Inputs from DiagBroadcaster into OperandB.
if (K >= 1) {
  for (int c3 = 0; c3 <= min(min(min(6, P + 1), P + R - 2), R + 3); c3 += 1)
    for (int c8 = max(max(5 * c3 - 16, c3), -4 * P + 5 * c3 + 4); c8 <= min(min(4 * R + c3 - 4, 5 * c3), c3 + 8);
      ACTION_READ("DiagBroadcaster", "DiagBroadcaster", "OperandB", "Inputs", 1)(c3, 0, c8, 0, c3);
  if (K >= 16 && P >= 1 && R >= 1) {
    for (int c3 = 0; c3 <= min(min(min(15, P + 1), P + R - 2), R + 12); c3 += 1)
      ACTION_SHRINK("DiagBroadcaster", "DiagBroadcaster", "Inputs", 1)(c3, 0, c3, 15, c3);
  } else if (K <= 15 && P >= 1 && R >= 1) {
    for (int c3 = 0; c3 <= min(min(min(15, P + 1), P + R - 2), R + 12); c3 += 1)
      ACTION_SHRINK("DiagBroadcaster", "DiagBroadcaster", "Inputs", 1)(c3, 0, c3, K - 1, c3);
  }
}

// Program to read Outputs from ColSpatialReducer into Result.
if (R >= 1)
  for (int c0 = 0; c0 <= min(15, K - 1); c0 += 1) {
    for (int c4 = 0; c4 <= min(4, P - 1); c4 += 1)
      for (int c8 = c4; c8 <= min(5 * R + c4 - 5, c4 + 10); c8 += 5)
        ACTION_READ_IU("ColSpatialReducer", "ColSpatialReducer", "Result", "Outputs", 2)(c4, c0, c8, c0, c0, c4);
    for (int c4 = 0; c4 <= min(13, P - 1); c4 += 1)
      ACTION_UPDATE("ColSpatialReducer", "ColBuffer", "ColSpatialReducer", "Outputs", 2)(c4, 0, c4, c0, c0, c4);
  }

// Program to compute Multiply at Multiplier.
for (int c0 = 0; c0 <= 15; c0 += 1) {
  for (int c4 = 0; c4 <= 4; c4 += 1)
    for (int c5 = 0; c5 <= 2; c5 += 1)
      COMPUTE_Multiplier_Multiply(c4 + 5 * c5, c0, c0, c4, c5);
  if (K >= c0 + 1) {
    for (int c4 = 0; c4 <= min(4, P - 1); c4 += 1)
      for (int c6 = c4; c6 <= min(5 * R + c4 - 5, c4 + 10); c6 += 5)
        ACTION_UPDATE("Multiplier", "ColSpatialReducer", "Result", "Outputs", 2)(c4, c0, c6, c0, c0, c4);
    if (K <= 15 && c0 + 1 == K) {
      for (int c3 = 0; c3 <= min(min(min(6, K - 2), P + 1), P + R - 2), R + 3); c3 += 1)
        for (int c6 = max(max(5 * c3 - 16, c3), -4 * P + 5 * c3 + 4); c6 <= min(min(4 * R + c3 - 4, 5 * c3), c3 + 4)
          ACTION_SHRINK("Multiplier", "OperandB", "Inputs", 1)(c3, K - 1, c6, K - 1, c3);
    } else if (c0 == 15) {
      for (int c3 = 0; c3 <= min(min(min(6, P + 1), P + R - 2), R + 3); c3 += 1)
        for (int c6 = max(max(5 * c3 - 16, c3), -4 * P + 5 * c3 + 4); c6 <= min(min(4 * R + c3 - 4, 5 * c3), c3 + 4)
          ACTION_SHRINK("Multiplier", "OperandB", "Inputs", 1)(c3, 15, c6, 15, c3);
    }
    for (int c4 = 0; c4 <= min(2,
      for (int c6 = 5 * c4; c6 <=
        ACTION_SHRINK("Multiplier"
```

- Present capability: build generated code against an EDDO emulator (automatically configured from the PHST)

Summary and Questions?

- ***Summary***
  - HST (Hardware Space-Time) – an abstraction for EDDO architectures represented using the Polyhedral Model
  - PolyEDDO (WIP) – an analysis and code-generation flow based on HST

- ***Research questions***
  - How do we think about mapping imperfectly-nested loops to generic EDDO architectures?
  - How do we capture sparsity extensions of the accelerators?

Acknowledgments

- **Angshuman Parashar, NVIDIA**
- **Po-An Tsai, NVIDIA**

Backup

Why do we need accelerators?

1) DNN models have tight constraints on latency, throughput, and energy consumption, esp. on edge devices

2) DNN models have trillions of computations

Need high throughput — Makes CPUs inefficient

3) DNN models involve heavy data movement

Need to reduce energy — Makes GPUs inefficient

Landscape of DNN Accelerators