2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

BatchSolve: A Divide and Conquer Approach to Solving the Memory Ordering Problem

Debarshi Chatterjee, Ismet Bayraktaroglu, Nikhil Sathe, Kavya Shagrithaya, Siddhanth Dhodhi, Spandan Kachhadiya

NVIDIA

accellera SYSTEMS INITIATIVE

Memory Ordering Problem

- Given the Issue Order of MemOps, can we find a Global Order that satisfies all the Ordering Rules?
- Notion of Global Order: To an external observer, the MemOps appear to happen in this Order.
- Ordering Rules
  - Memory Consistency Models
  - Deadlock Avoidance Rules
  - Micro-Architectural Specifications

Execution 1: Legal

SRC1	SRC2
ST_1(A)X	LD_1(A)X
ST_2(A)Y	LD_2(A)Y

Issue Order

GO: ST_1, LD_1, ST_2, LD_2 ☑
GO: ST_2, LD_2, ST_1, LD_1 ☒

Execution 2: Illegal

SRC1	SRC2
ST_1(A)X	LD_1(A)Y
ST_2(A)Y	LD_2(A)X

Issue Order

GO: ST_1, LD_2, ST_2, LD_1 ☒
GO: ST_2, LD_1, ST_1, LD_2 ☒

Example Ordering Rule: MemOps from same source must appear in GO in the same order as the appear in issue order

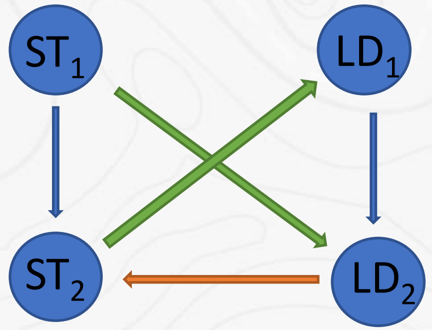
Prior Work

- Naïve Approaches
- Point of Serialization Snooping (POSS)
  - Basic Idea: Get hints from RTL to obtain GO
  - Pros: Good Coverage, Linear Time Complexity
  - Cons: Portability, Dev. & maintenance cost
- TSO Tool [Hangal et al., ISCA 2004]
  - Assumption: Unique store Data
  - Basic Idea: Construct Graph and check for cycles
  - Pros: Reasonably good coverage, Polynomial Time Complexity
  - Cons: Random atomics, not amenable to arbitrary ordering rules

Execution 2: Illegal

SRC1	SRC2
$ST_1(A)X$	$LD_1(A)Y$
$ST_2(A)Y$	$LD_2(A)X$

Issue Order



Edge Color Coding

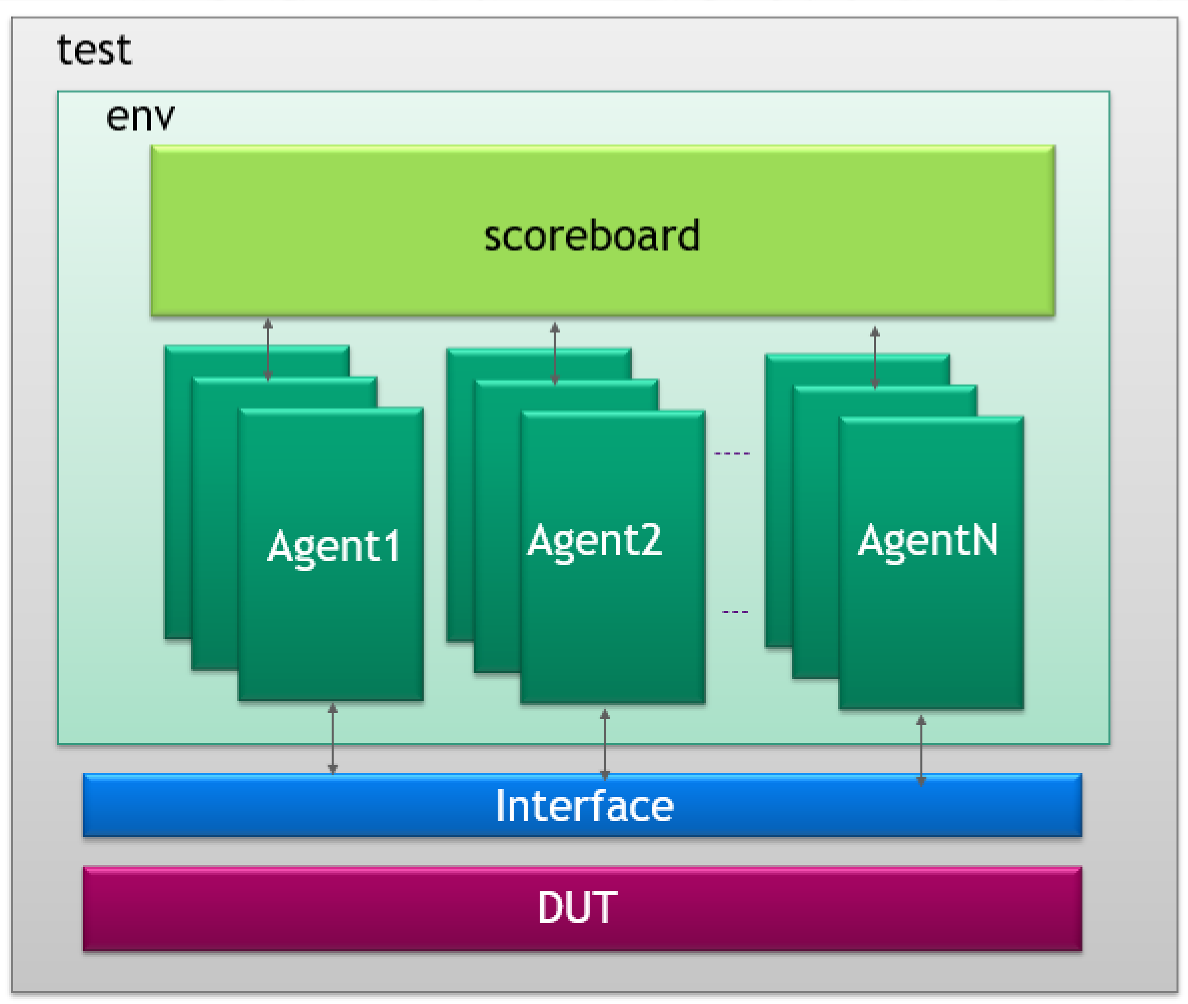
Blue : Issue Order Dep. Edges
Green : Observed Dep. Edges
Orange: Inferred Edges

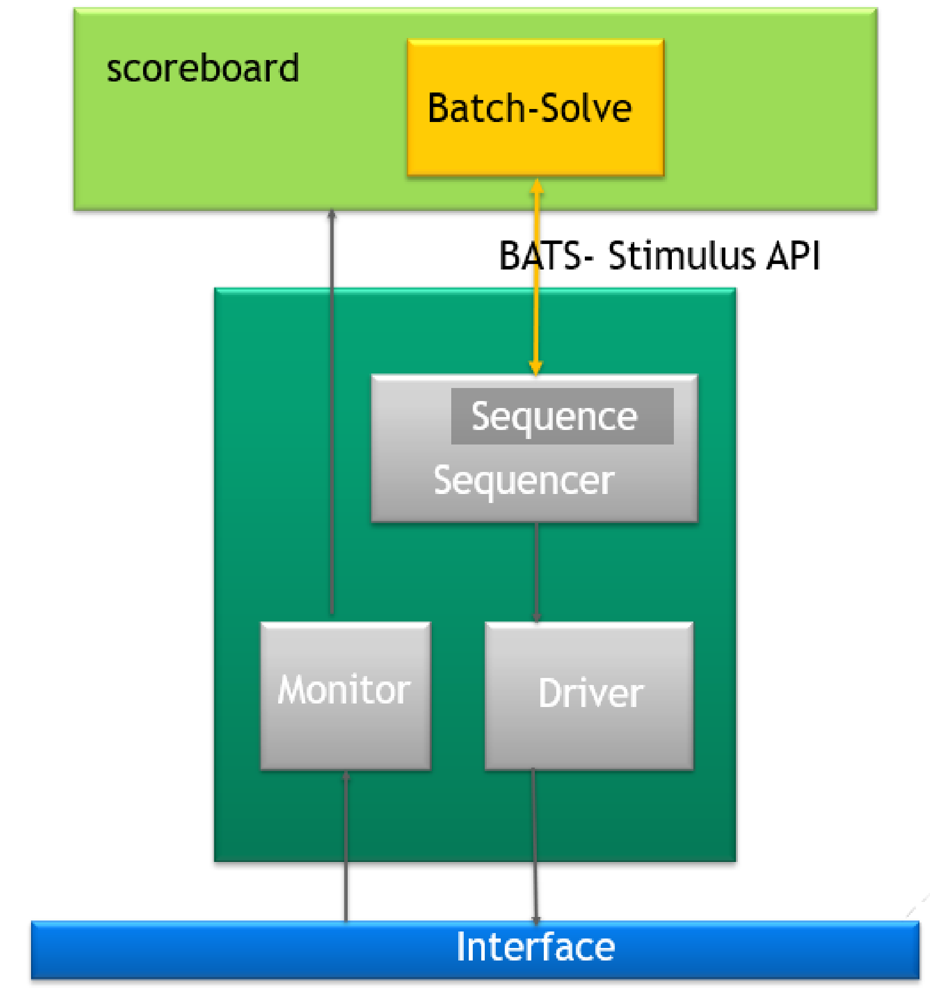
Motivation and High-Level Idea

- Need an ordering checking scheme with following:
  - Portable (Horizontal and Vertical Re-use)
  - Low Cost (Low Dev. And maintenance Effort; Immune to Arch. changes)
  - Flexible (Amenable to various ordering rules – PCIE, NVLink, other Link based rules)

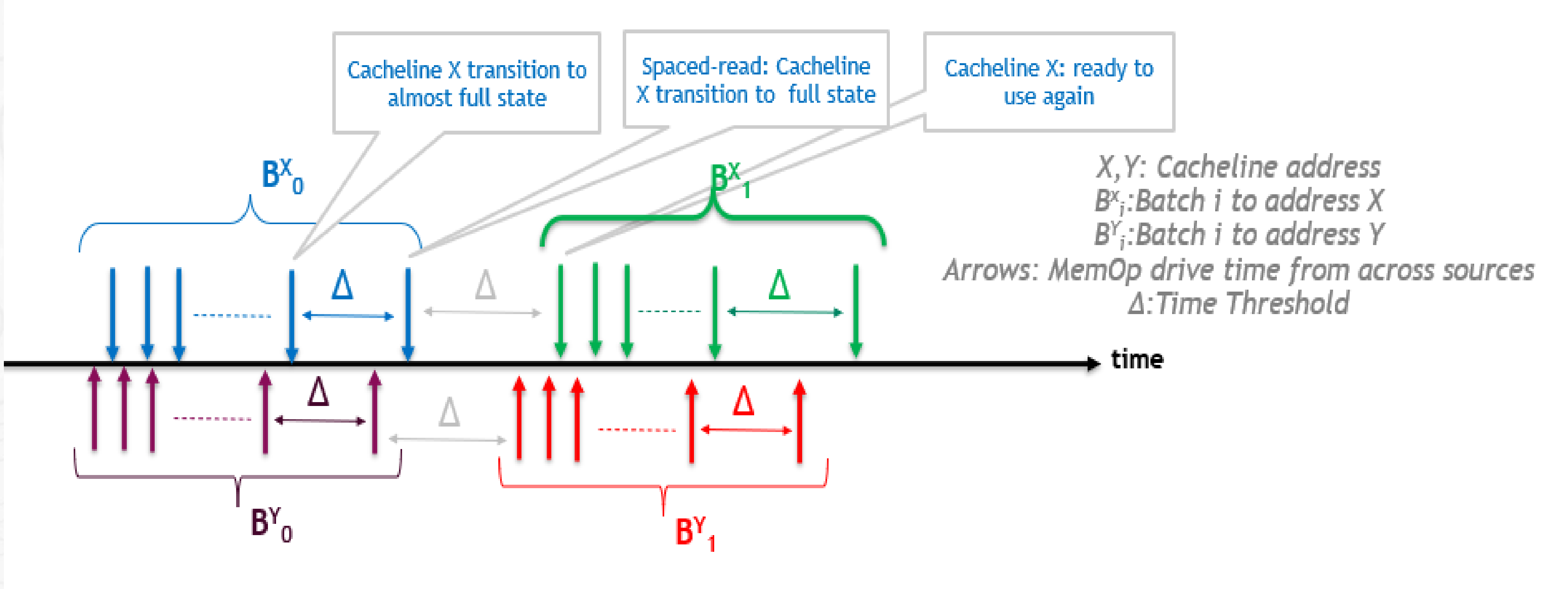
- BatchSolve - High Level Idea:
  - Specify Ordering Rules as high-level SV constraints
  - Formulate the problem so that SV solver can handle it
  - How do we address scalability issues? – Stimulus Batching

BATS – Integration into UVM Architecture





BATS – Stimulus Batching



Stimulus Batching - Continued

Issue-Time	SRC	MemOp	Sector-0	Sector-1	Sector-2	Sector-3	Count	STATE	Batch-num
T1	SRC1	Wr1	Wr1-C0	Wr1-C1	Wr1-C2		1	EMPTY	0
T2	SRC1	Rd1	Rd1-C0	Rd1-C1			2	EMPTY	0
T3	SRC2	Wr2	Wr2-C0	Wr2-C1	Wr2-C2	Wr2-C3	3	EMPTY	0
T4	SRC2	Wr3		Wr3-C0	Wr3-C1	Wr3-C2	4	EMPTY	0
T5	SRC2	Rd2	Rd2-C0	Rd2-C1	Rd2-C2	Rd2-C3	5	EMPTY	0
T6	SRC1	Rd3	Rd3-C0	Rd3-C1	Rd3-C2	Rd3-C3	6	EMPTY	0
T7	SRC1	Wr4	Wr4-C0	Wr4-C1	Wr4-C2		7	EMPTY	0
T8	SRC2	Wr5	Wr5-C0	Wr5-C1	Wr5-C2	Wr5-C3	8	EMPTY	0
T9	SRC2	Rd4	Rd4-C0	Rd4-C1	Rd4-C2		9	A-FULL	0
T10> T9 + Δ	SRC1	Rd5	Rd5-C0	Rd5-C1	Rd5-C2	Rd5-C3	10	FULL	0
T11 > T10 + Δ	SRC1	Wr6		Wr6-C0	Wr6-C1	Wr6-C2	1	EMPTY	1
T12	SRC2	Wr7	Wr7-C0	Wr7-C1			2	EMPTY	1

Spaced Read

SV Solver – Sample Input

MemOp at issued at T4 on SRC2 is omitted because it does not have a child to sector0 for which ordering is being tested

	SRC	UID	MemOp	Sector-0	Read Data Rcvd		Byte Enable		Write Data	
					Byte-0	Byte-1	Byte0	Byte1	Byte0	Byte1
0	-		Dummy Init Wr				1	1	I1	I2
T1	SRC1	1	Wr1	Wr1-C0			1	1	X1	X2
T2	SRC1	2	Rd1	Rd1-C0	X3	X4	1	1		
T3	SRC2	3	Wr2	Wr2-C0			1	1	X3	X4
T5	SRC2	4	Rd2	Rd2-C0	X3	X4	1	1		
T6	SRC1	5	Rd3	Rd3-C0	X7	X4	1	1		
T7	SRC1	6	Wr4	Wr4-C0			1	1	X5	X6
T8	SRC2	7	Wr5	Wr5-C0			1	0	X7	X8
T9	SRC2	8	Rd4	Rd4-C0	X5	X6	1	1		
T10	SRC1	9	Rd5	Rd5-C0	X5	X6	1	1		

SV Solver – Sample Output

Issue-Time	SRC	UID	GO	Sector-0	Read Data Rcvd		Byte Enable		Write Data	
					Byte-0	Byte-1	Byte0	Byte1	Byte0	Byte1
0	-		Dummy Init Wr				1	1	I1	I2
T1	SRC1	1	Wr1	Wr1-C0			1	1	X1	X2
T3	SRC2	3	Wr2	Wr2-C0			1	1	X3	X4
T2	SRC1	2	Rd1	Rd1-C0	X3	X4		1		
T5	SRC2	4	Rd2	Rd2-C0	X3	X4	1	1		
T8	SRC2	7	Wr5	Wr5-C0			1	0	X7	X8
T6	SRC1	5	Rd4	Rd4-C0	X7	X4	1	1		
T7	SRC1	6	Wr4	Wr4-C0				1	X5	X6
T9	SRC2	8	Rd4	Rd4-C0	X5	X6				
T10	SRC1	9	Rd5	Rd5-C0	X5	X6	1	1		

Notice in UID column MemOps from same SRC appears in issue order

Rd1 inferred to be ordered after Wr2

Rd4 inferred to be ordered after Wr2 and Wr5

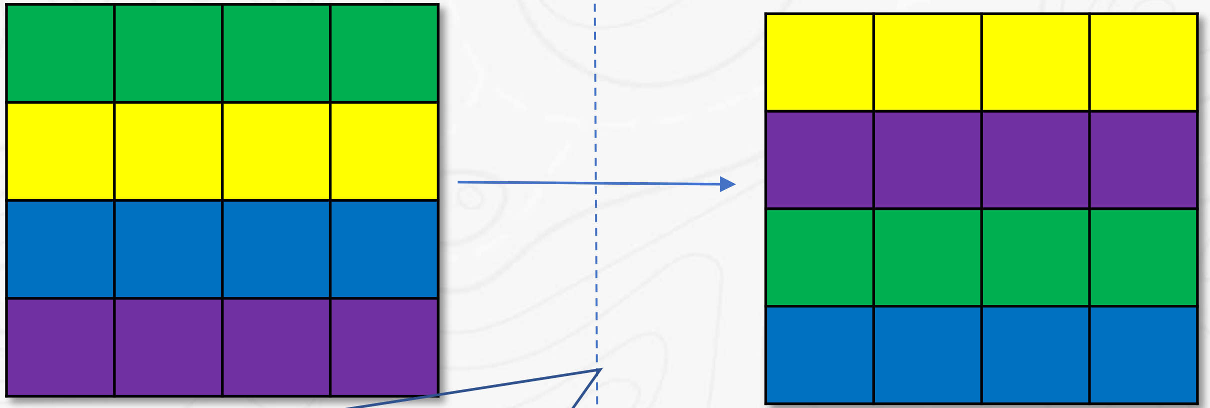
Inside the Solver – 1

Ordering Rule as Constraints

```
//This demo assumes a system where reodering is not possible from same source
constraint c_ordering_constraints {
  foreach (G_order[i]){
    foreach (G_order[j]){
      if(i!=j && i>j && SRC[G_order[i]] == SRC[G_order[j]]) {
        G_order[i] > G_order[j];
      }
    }
  }
};
```

Input Matrix I:
MemOps in issue order

Row Permuted
Random Matrix P:
MemOps in GO



```
//Generate Random Permutation of N integers
constraint c_random_permutation {
  G_order.size() == I_cmd_type.size;
  foreach(G_order[i]) {
    G_order[i] >= 0;
    G_order[i] < I_cmd_type.size;
  }
  unique {G_order};
  G_order[0] == 0;                          // Setting GO[0] to 0 for DUMMY INIT.
  G_order[G_order.size()-1] == G_order.size()-1;  // Setting GO[last] to last for spaced read.
};
```

Inside the Solver - 2

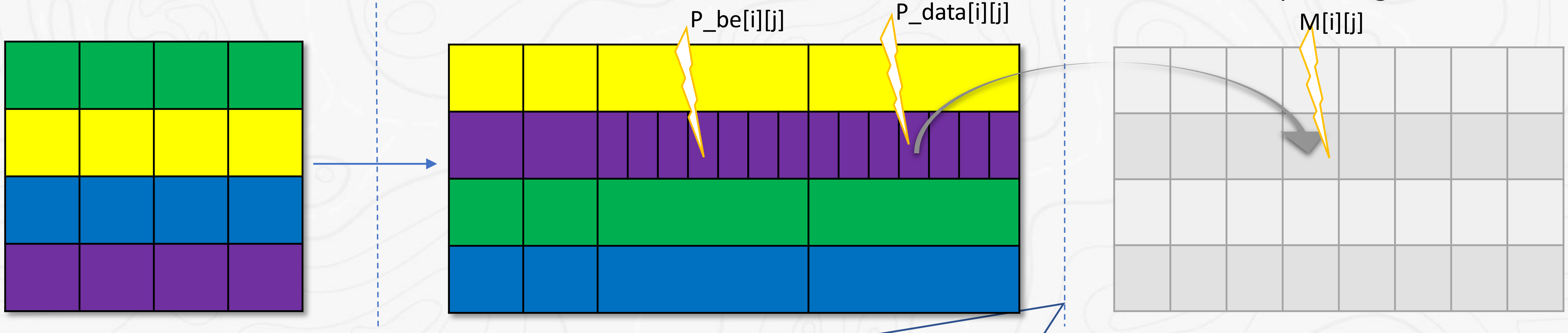
Ordering Rule as Constraints

GO Constraints

Input Matrix I:
MemOps in issue order

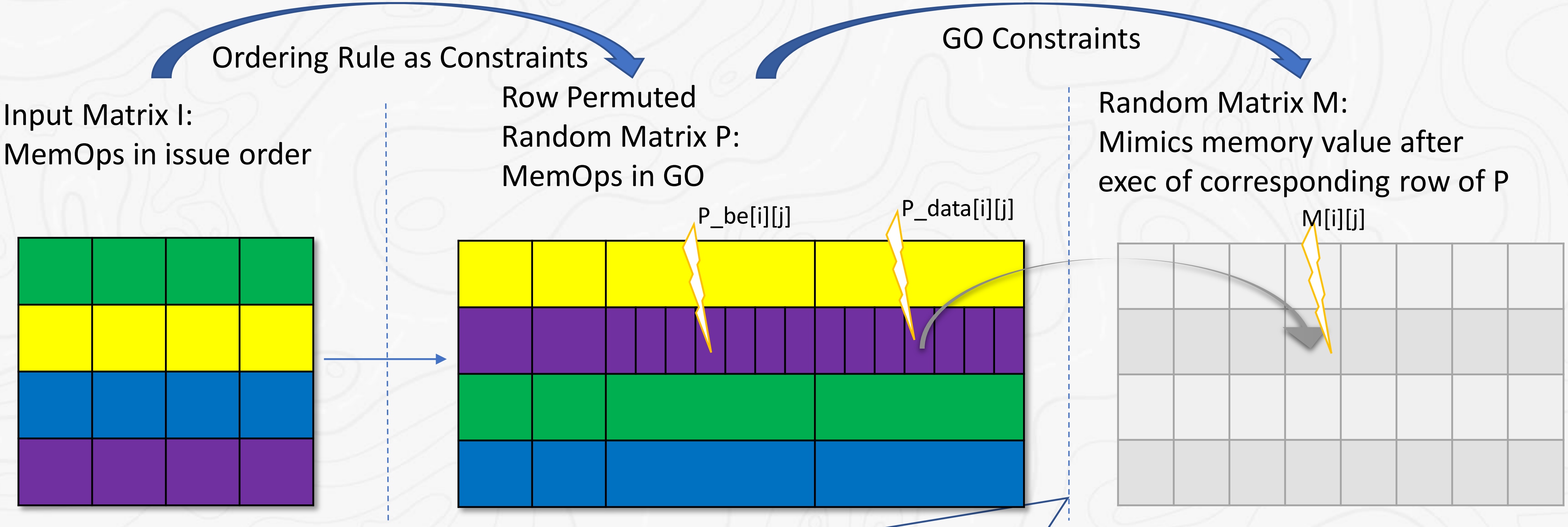
Row Permuted
Random Matrix P:
MemOps in GO

Random Matrix M:
Mimics memory value after
exec of corresponding row of P



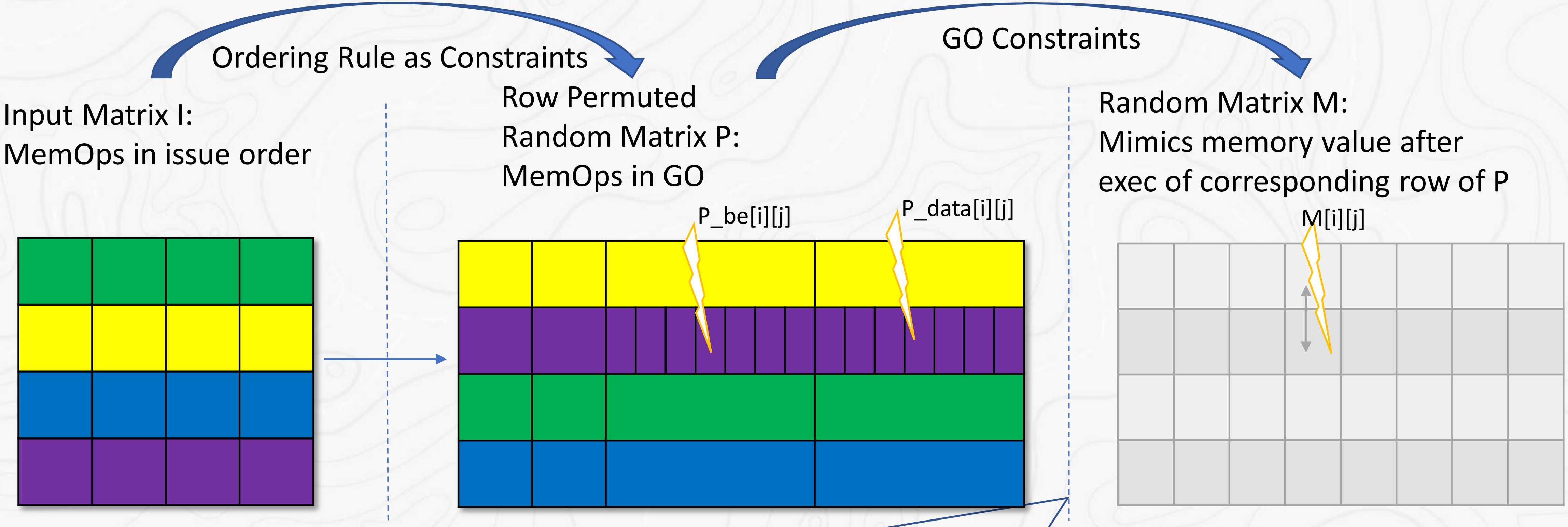

```
constraint c_read {
  foreach (M[i,j]){
    //Equate data from reads in GO into memory model, if corresponding byte enables are set.
    (P_be[i][j] == 1) && (P_cmd_type[i] == READ)  -> (M[i][j] == P_data[i][j]);
  }
};
```

Inside the Solver - 3




```
constraint c_write {
  foreach (M[i,j]){
    //Equate data from writes in GO into memory model, if corresponding byte enables are set.
    (P_be[i][j] == 1) && (P_cmd_type[i] == WRITE) -> (M[i][j] == P_data[i][j]);
  }
};
```

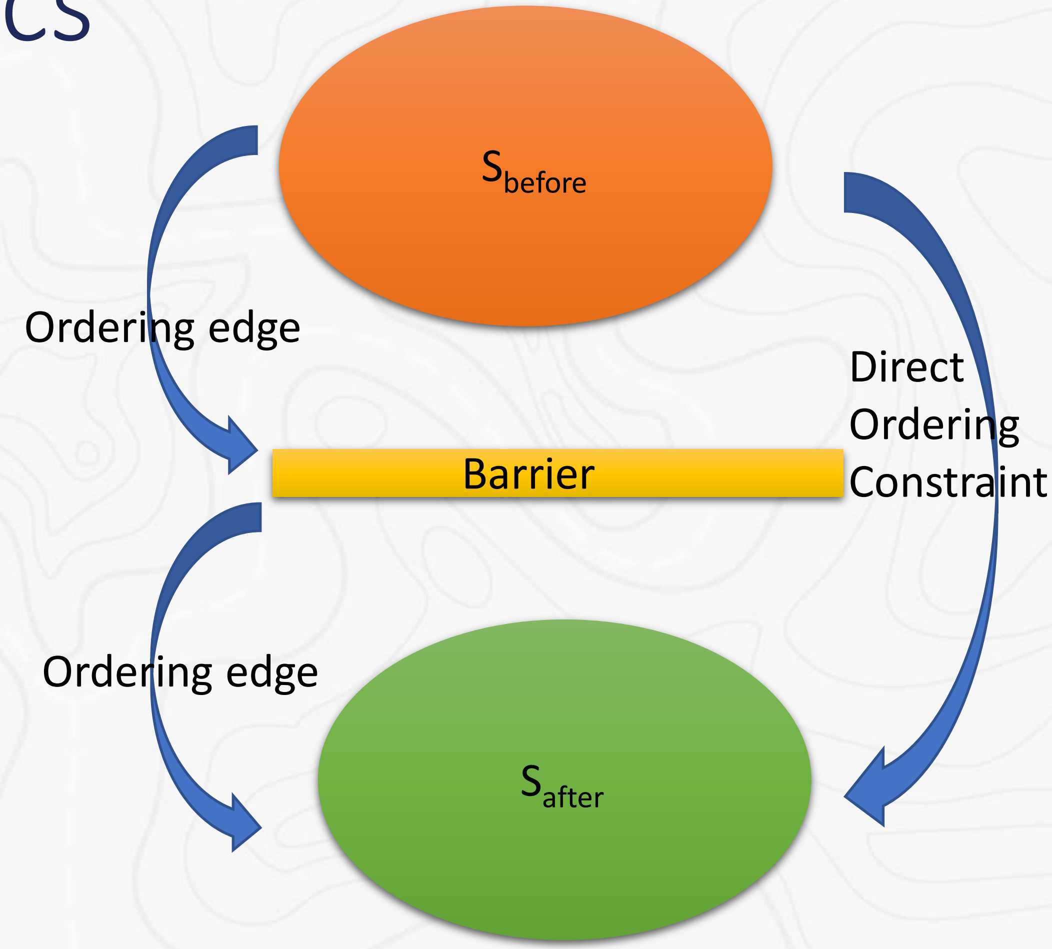
Inside the Solver - 4




```
constraint c_invariance{
  foreach (M[i,j]){
    if(i>0 && (P_cmd_type[i]==READ || (P_cmd_type[i] inside {WRITE, ATOMIC_AND} && P_be[i][j] == 0))){
      M[i][j] == M[i-1][j];
    }
  }
};
```

BatchSolve – Advanced Topics

- Atomic Handling
  - Replace Function calls with explicit SV constraints
- Barrier Handling
  - Using “Rules” determine which MemOps should be ordered before the Barrier and which should be ordered after it (some MemOps can be neither)
  - Remove the Barrier and draw edges from every MemOp in S_{before} to every MemOp in S_{after}
  - These edges are created in pre_randomize and introduced as constraints to the SV solver



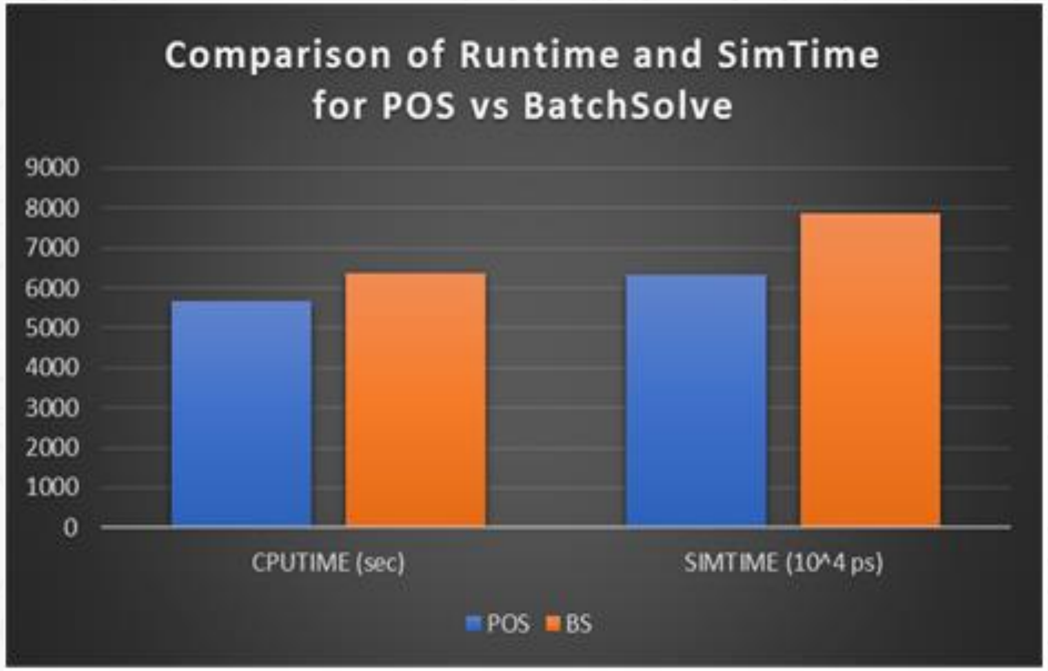
Sample Rule: *Reads issued after membar-ack from same or different source (as that of membar) must be ordered after the membar*

EDA Playground Demo Links

- *Link to simple DEMO (Reads/Writes/Atomics)*
  - *https://www.edaplayground.com/x/rXKN*
  - *Example of an execution for which GO exists and one for which it does not*
- Link to advanced DEMO (Barrier Handling)
  - https://www.edaplayground.com/x/DsUc
  - Examples of legal and illegal execution with barrier

Results

	POSS	BATS
Init Develop effort	80 weeks	8-weeks
Maintenance effort per project(estimate)	80 weeks	0-1 weeks (not including debug)
Porting effort to other UVM TB	Not portable easily	1 week (assuming TB has some score boarding)



Conclusion and Future Work

- BATS Pros
  - Low Development Cost
  - Low Maintenance Cost
  - Easily Portable
  - Easy to specify ordering rules as high level SV constraints
- BATS Cons
  - Slight Coverage loss due to Batching
  - Slight increase in runtime
- Future Work
  - Can a re-formulation such as convex relaxation allow to increase Batch-Size?

Thank you!