SAND2019-14284C

INCA: In-Network Compute Assistance

Whit Schonbein — Sandia National Laboratories, UNM
Ryan E. Grant — Sandia National Laboratories, UNM
Matthew G. F. Dosanjh — Sandia National Laboratories
Dorian Arnold — Emory University

EMORY UNIVERSITY

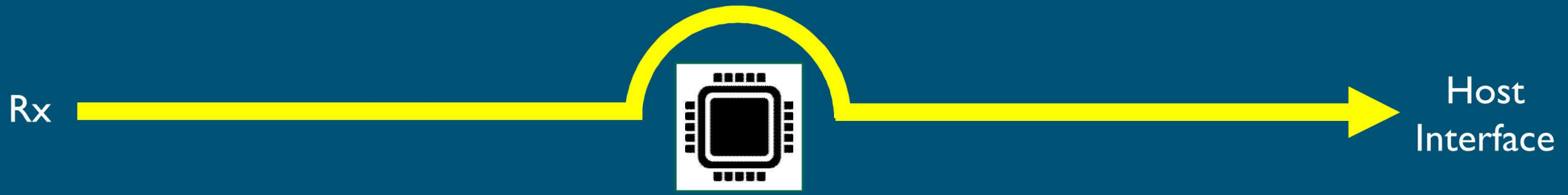
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

A Case for In-Network Computing

- SmartNICs everywhere
- Idle network = Idle computational resources
- An opportunity to harvest computational resources

How It's Done Today

- Two approaches to SmartNICs:
  - Bump-in-the-wire



2.5GHz CPU
250 Million Messages / Second
10 Instructions / Message

DEADLINE

How It's Done Today

- Two approaches to SmartNICs:
  - Bump-in-the-wire
  - Compute outside message processing pipeline

How INCA is Different

	Bump-in-the-wire	Outside the Pipeline
Deadline		
Deadline-Free		

How INCA is Different

	Bump-in-the-wire	Outside the Pipeline
Deadline	Myrinet Quadrics SPiN (SC'17) Atos BXI Broadcom Stingray Azure	
Deadline-Free		

How INCA is Different

	Bump-in-the-wire	Outside the Pipeline
Deadline	Myrinet Quadrics SPiN (SC'17) Atos BXI Broadcom Stingray Azure	
Deadline-Free		Mellanox Bluefield

How INCA is Different

	Bump-in-the-wire	Outside the Pipeline
Deadline	Myrinet Quadrics SPiN (SC'17) Atos BXI Broadcom Stingray Azure	
Deadline-Free	**INCA**	Mellanox Bluefield

- Leverage bump-in-the-wire capabilities
- Enable execution of kernels of arbitrary length

Why Use INCA?

- With INCA:
  - Accelerate scientific applications
  - Enable novel types of distributed applications

Technical Challenges and Desired Features

Technical Challenges and Desired Features

- Support kernels of arbitrary length

Technical Challenges and Desired Features

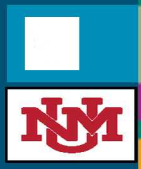
- Support kernels of arbitrary length
- Forward compatibility

2.5GHz CPU 250 Million Messages / Second 10 Instructions / Message	→	2.5GHz CPU 350 Million Messages / Second 7.14 Instructions / Message

Technical Challenges and Desired Features

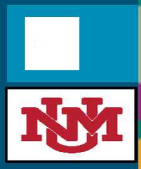
- Support kernels of arbitrary length
- Forward compatibility
- Host-independent progress

Technical Challenges and Desired Features



- Support kernels of arbitrary length
- Forward compatibility
- Host-independent progress
- Leverage existing hardware

Technical Challenges and Desired Features



- Support kernels of arbitrary length
- Forward compatibility
- Host-independent progress
- Leverage existing hardware
- Ease of use

INCA: In-Network Compute Assistance

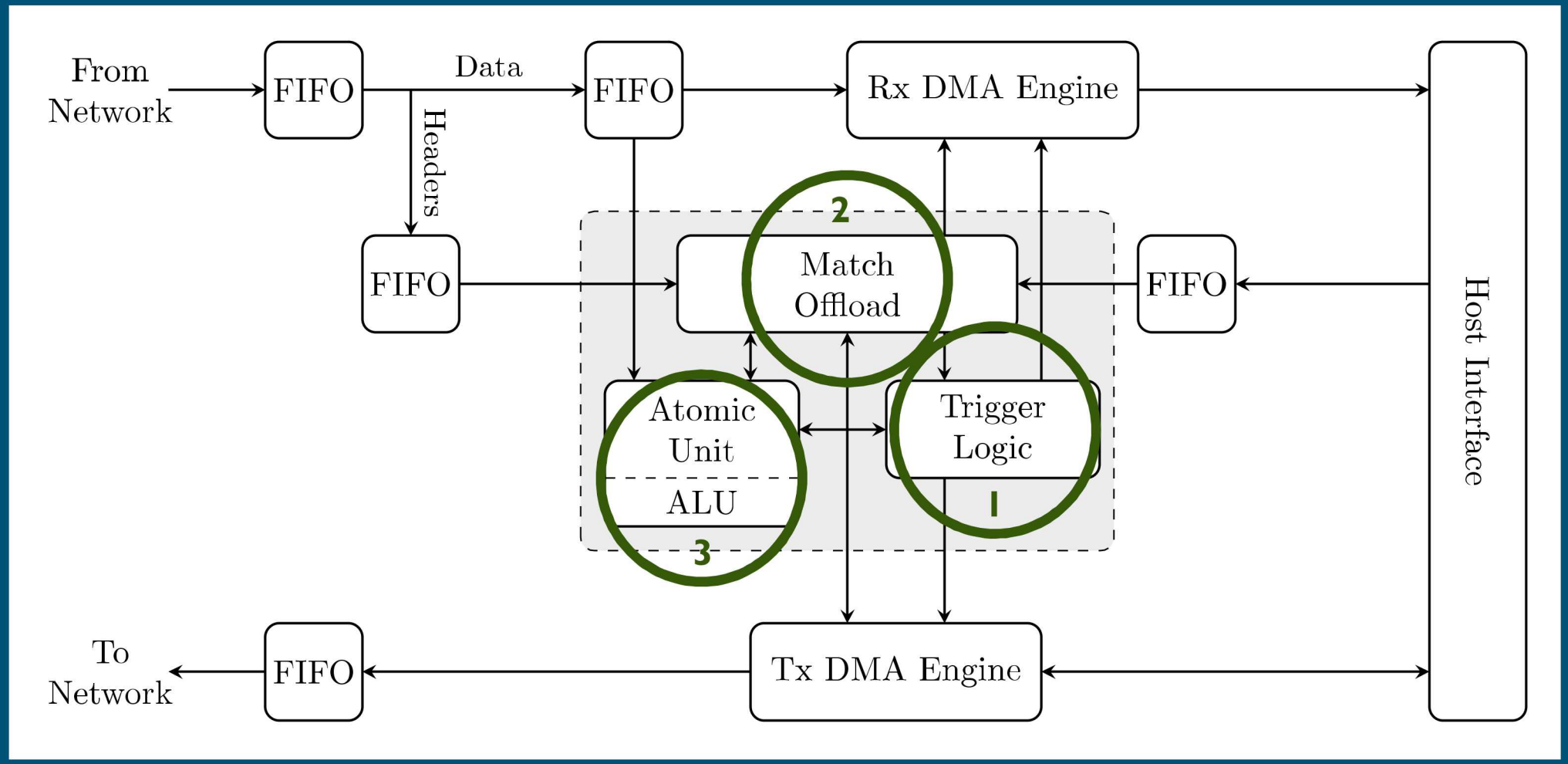
What Is INCA?

- INCA is:
  - A model of computation ★
  - A programming ecosystem:
    - High and low level languages for expressing kernels ★
    - Compiler
    - Interpreter
  - A network programming API reference implementation

The INCA Model of Computation

- Motivated by the challenge of leveraging existing hardware

The INCA Model of Computation



1. Triggered Operations 2. Message Matching 3. Atomic operations

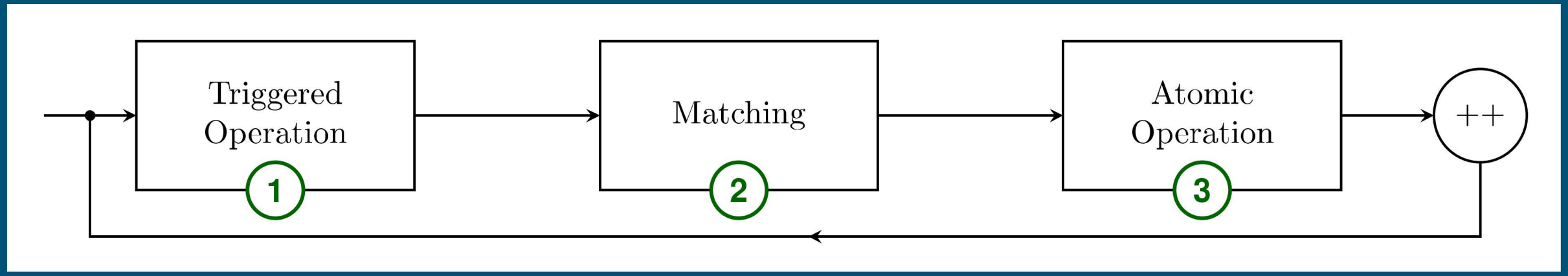
Adapted from: Underwood *et al.*, "Enabling Flexible Collective Communication Offload with Triggered Operations,", 2011

The INCA Model of Computation

1. Triggered operation generates message containing 1st argument.

2. Unique matching element specifies buffer containing 2nd argument and atomic.

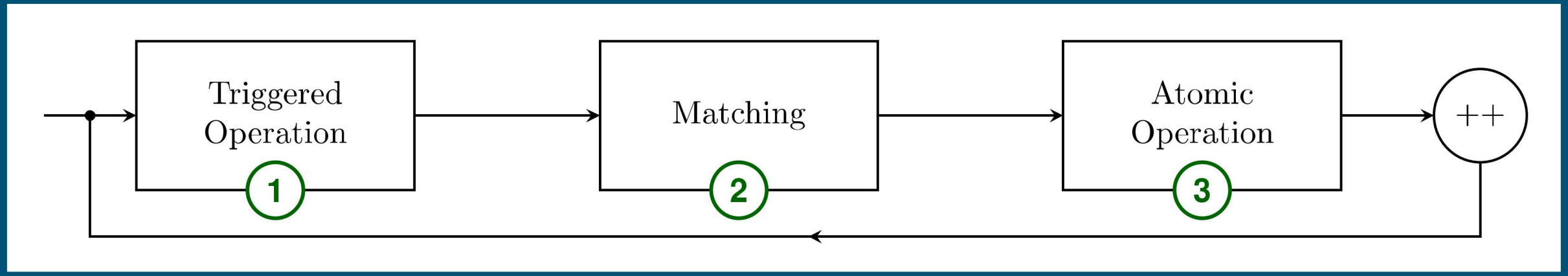
3. Atomic unit performs specified operation and stores result.



- Program: A list of tuples of triggered operations, matching entries, and atomic operations, ordered by thresholds, sharing the same counter.
- Turing complete

Complexity is Easy, Simplicity is Hard

- Minimal modifications required for INCA kernel execution

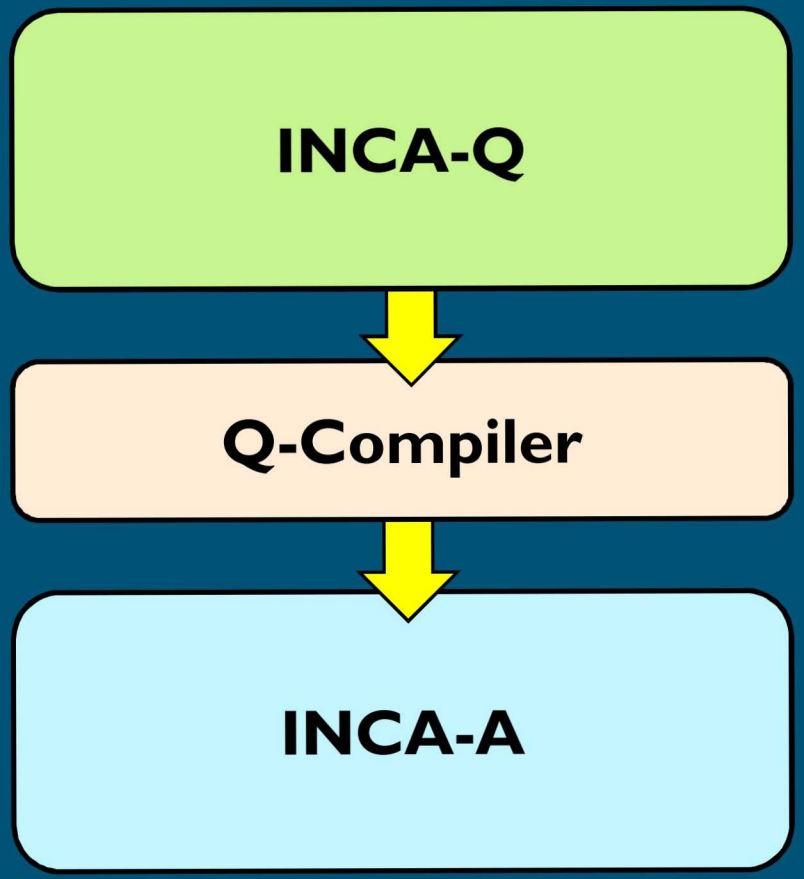


Triggering conditions

Matching entries

New atomic

INCA Ecosystem



INCA Kernels

Algorithm 1 INCA-Q Dot Product

```
1: i = 0
2: while i < 50 do {
3:     c = c + (A[i] * B[i])
4:     i = i + 1
5: }
```

Algorithm 2 INCA-A Dot Product

```
1 PUTL i, 0
2 PUTL r0, i
3 LT r0, r0, 50
4 BLEZ r0, 10
5 PUTL r1, A[i]
6 MUL r1, r1, B[i]
7 ADD c, c, r1
8 ADD i, i, 1
9 JMP 2
10 END
```

Technical Challenges and Desired Features

- Support kernels of arbitrary length
  - Deadline-free

- Forward-compatible
  - Increasing network speeds = faster execution

- Host-independent progress
  - Instruction processing pipeline fully offloaded

- Leverages existing hardware

- Ease of use

Potential Pitfalls

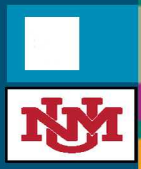
- Idle host processor
  - Fast handoff
- INCA-induced network endpoint congestion
  - It's free work
- Slow message rates = slow INCA execution speed
  - It's free work

Evaluation

Evaluation

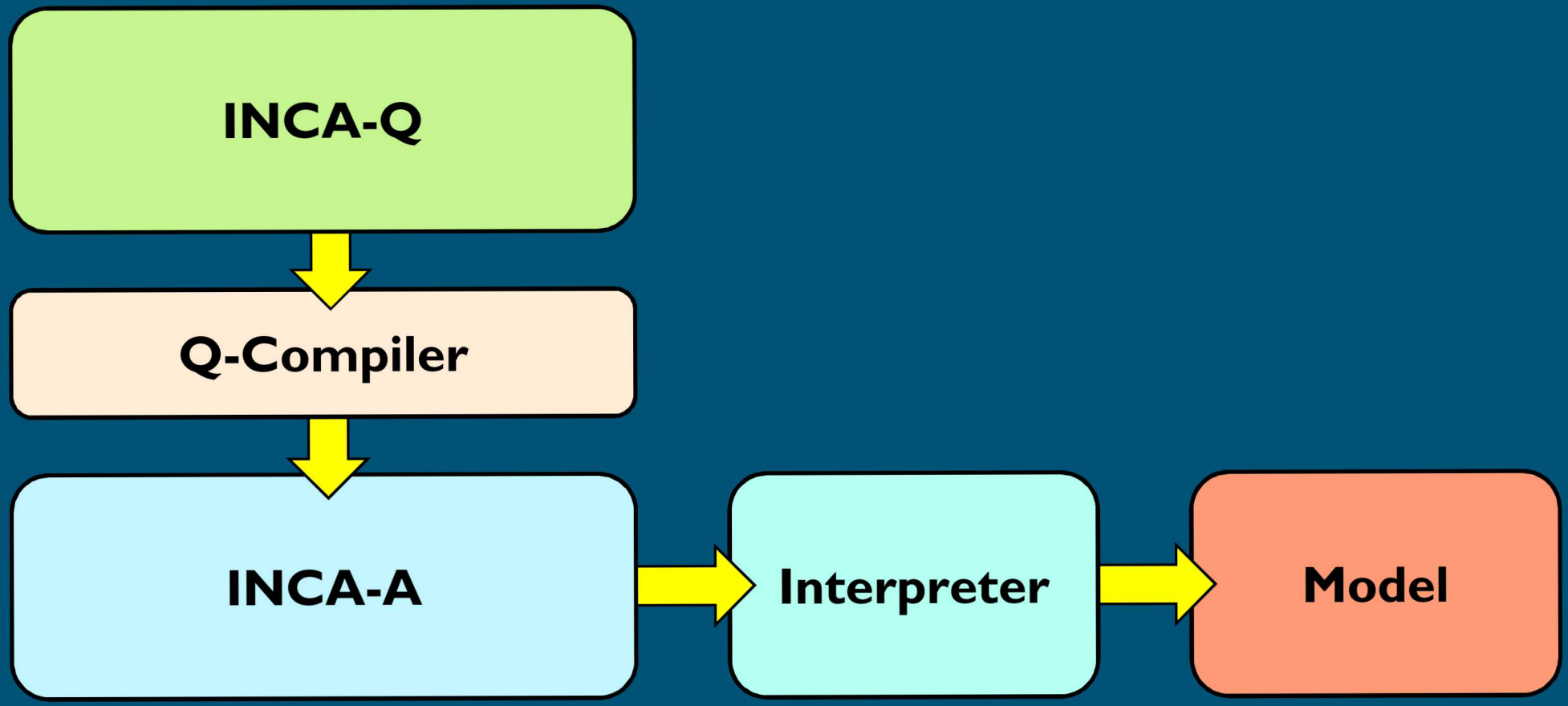
- How fast can INCA kernels execute?
- How can INCA help accelerate applications?

INCA Kernel Execution



- How fast can INCA kernels execute?
- How can INCA help accelerate applications?

INCA Kernel Execution



INCA Kernel Execution

- Kernels
  - Matrix transposition
  - Filter
  - Matrix unpack
  - Convolution
  - Linear interpolation
  - Hadamard product
  - Dot product
  - Matrix multiplication

INCA Kernel Execution

- Kernels
  - Matrix transposition
  - Filter
  - Matrix unpack
  - Convolution
  - Linear interpolation
  - Hadamard product
  - Dot product
  - Matrix multiplication

INCA Kernel Execution

- Model Parameters
  - 200 million messages / second
  - Scratchpad scenario:
    - 1MiB NIC-local memory
    - 1ns access time
  - Negligible loopback latency
  - Vary payloads from 128B to 8192B
- Instruction counts: 132 to 271652
- ~77% are greater than 500 instructions

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	

All times microseconds

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	
scratchpad 400Gb/s							1067.50	2.89x

All times microseconds

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	
scratchpad 400Gb/s							1067.50	2.89x
scratchpad 1000Gb/s							650.24	4.75x

8KiB on 2.3GH[illegible]aswell CPU
139.49 – 10.56 microsecs

All times microseconds

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	

All times microseconds

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	
parallel	1.13	3.61	7.07	26.59	52.92	208.86	417.68	7.68x

All times microseconds

INCA Kernel Execution – Matrix Multiplication

	Payload							
Scenario	128B	256B	512B	1024B	2048B	4096B	8192B	Average Speedup wrt scratchpad
scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	
parallel	1.13	3.61	7.07	26.59	52.92	208.86	417.68	7.68x
advanced-parallel	0.29	1.10	1.50	5.89	7.82	31.29	47.42	42.09x

8KiB on 2.3GH[illegible]aswell CPU
139.49 – 10.56 microsecs

All times microseconds

INCA Kernel Execution

- INCA runtimes can be significantly slower than CPU runtimes.
- However:
  - We get this work for free
  - Kernel runtimes improve as network speeds increase
  - There are ample opportunities for additional optimizations

Application Acceleration

- How fast can INCA kernels execute?
- How can INCA help accelerate applications?

Application Acceleration

- (Mini)Apps
  - MiniAMR
  - MiniMD
  - MiniFE
  - LAMMPS

Application Acceleration

- Identify regions of code as candidates for INCA offloading.
- Time those candidates as well as the regions they appear in.
- Calculate ideal speedup assuming 100% overlap.

Application Acceleration

	MiniAMR	MiniMD	MiniFE	LAMMPS
Potential speedup without code refactor		11%	2.98%	11.50%

Application Acceleration

	MiniAMR	MiniMD	MiniFE	LAMMPS
Potential speedup without code refactor		11%	2.98%	11.50%
Potential speedup with code refactor	26%	37.20%	25.70%	28.90%

Conclusion

Conclusion

- INCA harvests idle network resources
- Deadline-free, host-independent, kernel execution
- Requires modest modifications to existing hardware
- INCA ecosystem → ease-of-use
- Accelerate application performance

Thank You

This work was funded through the Computational Systems and Software Environment sub-program of the Advanced Simulation and Computing Program funded by the National Nuclear Security Administration

Extras

Application run details

- MiniFE 2.1.0
- MiniMD 2.0
- MiniAMR 1.0
- LAMMPS Stable Release
- Compiled: Intel 19.0.3.199 and OpenMPI 3.0.
- System: dual socket 2.1 GHz Intel Broadwell E5-2695 v4, 18 cores per processor, 128 GB of RAM per node
- Network: Intel Omni-Path Network.

Haswell 2.3GHz kernel runtimes (microsecs), -O0

	128B	256B	512B	1024B	2048B	4096B	8192B
unpack	0.17	0.28	0.48	0.90	1.74	3.51	7.23
convolution	0.72	1.51	1.89	4.28	5.50	12.43	17.51
lerp	0.35	0.59	1.15	2.22	5.99	13.68	28.15
hadamard	0.16	0.23	0.33	0.55	0.96	1.76	4.51
dp	0.13	0.20	0.33	0.61	1.13	2.23	4.72
mm	0.45	0.77	2.38	4.69	17.66	36.79	139.49

Haswell 2.3GHz kernel runtimes (microsecs), -O2

	128B	256B	512B	1024B	2048B	4096B	8192B
unpack	0.12	0.17	0.27	0.47	0.86	1.68	3.27
convolution	0.29	0.38	0.49	0.76	1.14	2.01	3.41
lerp	0.19	0.27	0.45	0.75	3.09	7.55	16.20
hadamard	0.09	0.11	0.13	0.16	0.20	0.29	0.73
dp	0.05	0.05	0.05	0.05	0.05	0.05	0.05
mm	0.20	0.27	0.44	0.75	1.95	3.81	10.56

All kernel runtimes (from SC paper)

Kernel	Optimization	Payload size							Average speedup wrt base	Average speedup wrt scratchpad
		128B	256B	512B	1024B	2048B	4096B	8192B		
matrix-transpose	base	42.16	83.08	142.6	283.96	522.04	1042.84	1995.16		
	scratchpad	1.52	3.0	5.14	10.24	18.81	37.58	71.89	27.74×	
filter	base	65.98	130.36	260.23	525.57	1046.01	2091.8	4190.72		
	scratchpad	2.34	4.58	9.12	18.56	36.88	73.88	147.81	28.36×	
matrix-unpack	base	96.51	190.91	379.71	757.31	1512.51	3022.91	6043.71		
	scratchpad	2.8	5.54	11.01	21.96	43.84	87.62	175.17	34.48×	
convolution	base	328.48	657.96	1274.28	2549.56	5014.84	10030.68	19891.8		
	scratchpad	9.26	18.52	35.45	70.91	138.62	277.24	548.09	35.94×	
linear-interpolation	base	301.31	617.15	1248.83	2512.19	5038.91	10092.35	20199.23		
	scratchpad	9.38	19.18	38.8	78.03	156.5	313.42	627.28	32.18×	
hadamard-product	base	56.08	110.92	198.28	395.32	744.76	1488.28	2886.04		
	scratchpad	1.89	3.73	6.61	13.18	24.7	49.36	95.44	30.0×	
	clobber	1.54	3.03	5.21	10.37	19.07	38.09	72.91	38.13×	1.27×
	parallel	0.02	0.03	0.05	0.09	0.18	0.35	0.69	3918.83×	116.99×
	parallel-clobber	0.01	0.02	0.03	0.05	0.09	0.18	0.35	7208.41×	240.11×
dot-product	base	48.92	96.6	191.96	382.68	764.12	1527.0	3052.76		
	scratchpad	1.50	2.96	5.87	11.69	23.34	46.64	93.23	32.70×	
	clobber	1.32	2.60	5.17	10.29	20.52	41.01	81.97	37.18×	1.14×
	parallel	1.16	2.25	4.45	8.84	17.63	35.21	70.37	43.09×	1.32×
	parallel-clobber	1.15	2.24	4.42	8.8	17.54	35.04	70.03	43.33×	1.33×
	advanced-parallel	0.04	0.05	0.08	0.12	0.21	0.42	0.84	2807.45×	85.81×
matrix-multiplication	base	247.76	965.0	1727.88	6863.16	12966.2	51771.8	100596.12		
	scratchpad	7.89	30.61	53.91	213.88	400.25	1597.64	3088.59	32.06×	
	parallel	1.13	3.61	7.07	26.59	52.92	208.86	417.68	246.12×	7.68×
	parallel-clobber	1.08	3.48	6.86	25.84	51.52	203.24	406.43	254.01×	7.92×
	advanced-parallel	0.29	1.1	1.5	5.89	7.82	31.29	47.42	1354.69×	42.09×

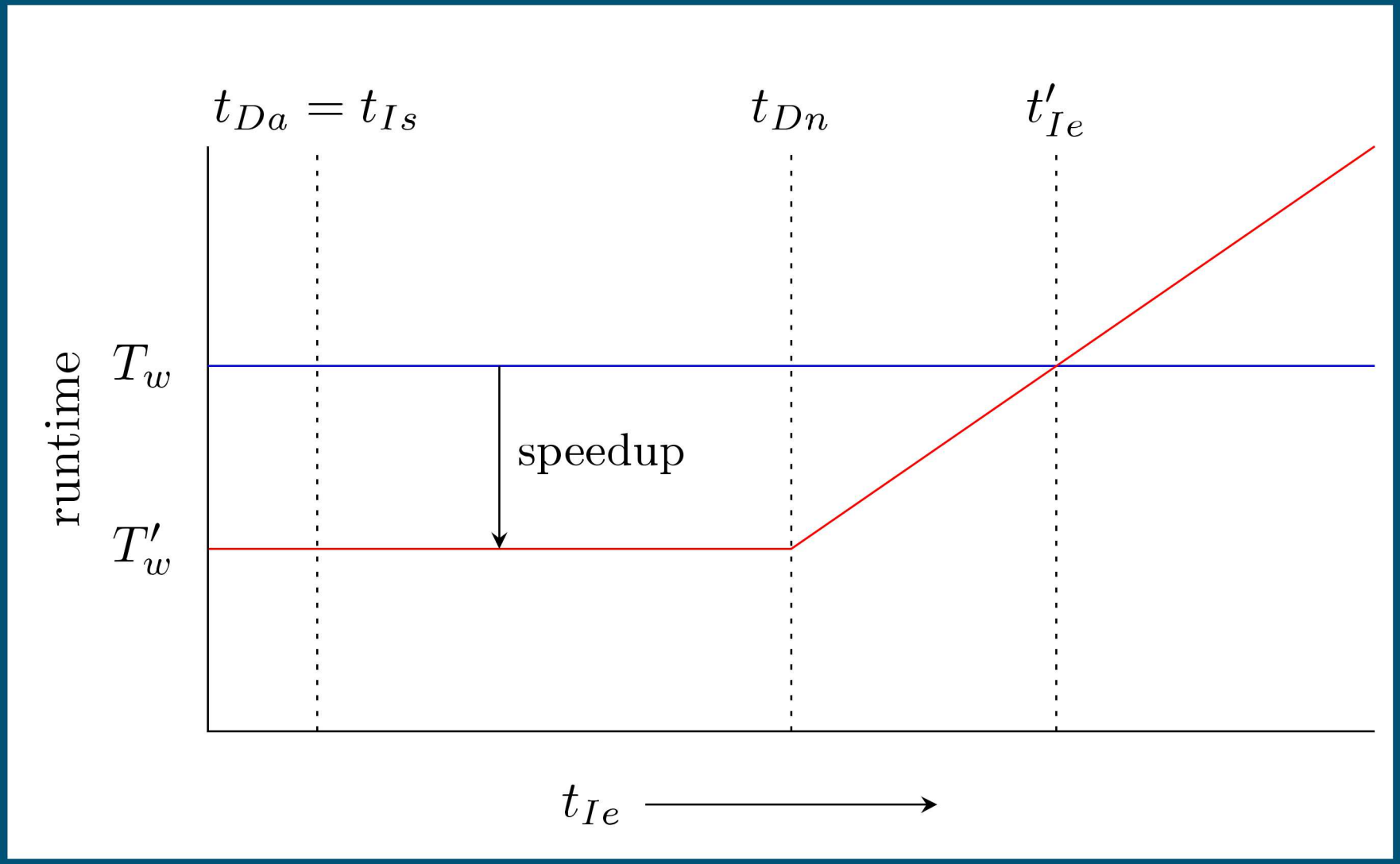
Kernel instruction counts (in SC paper)

	Payload size						
Kernel	128B	256B	512B	1024B	2048B	4096B	8192B
vector dot product	132	260	516	1028	2052	4100	8196
matrix transpose	136	268	460	916	1684	3364	6436
hadamard product	168	332	588	1172	2196	4388	8484
filter	208	406	808	1647	3271	6555	13112
matrix unpack	246	486	966	1926	3846	7686	15366
matrix multiplication	696	2700	4748	18836	35220	140580	271652
convolution	808	1616	3088	6176	12064	24128	47680
linear interpolation	826	1690	3418	6874	13786	27610	55258

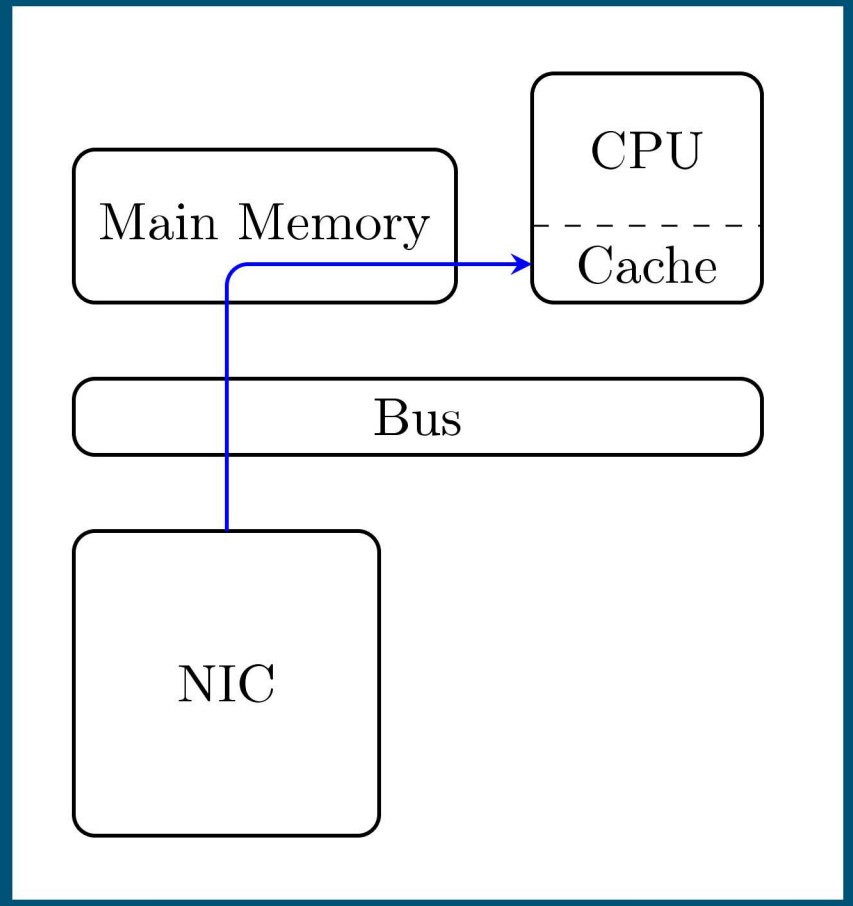
Full potential speedup table

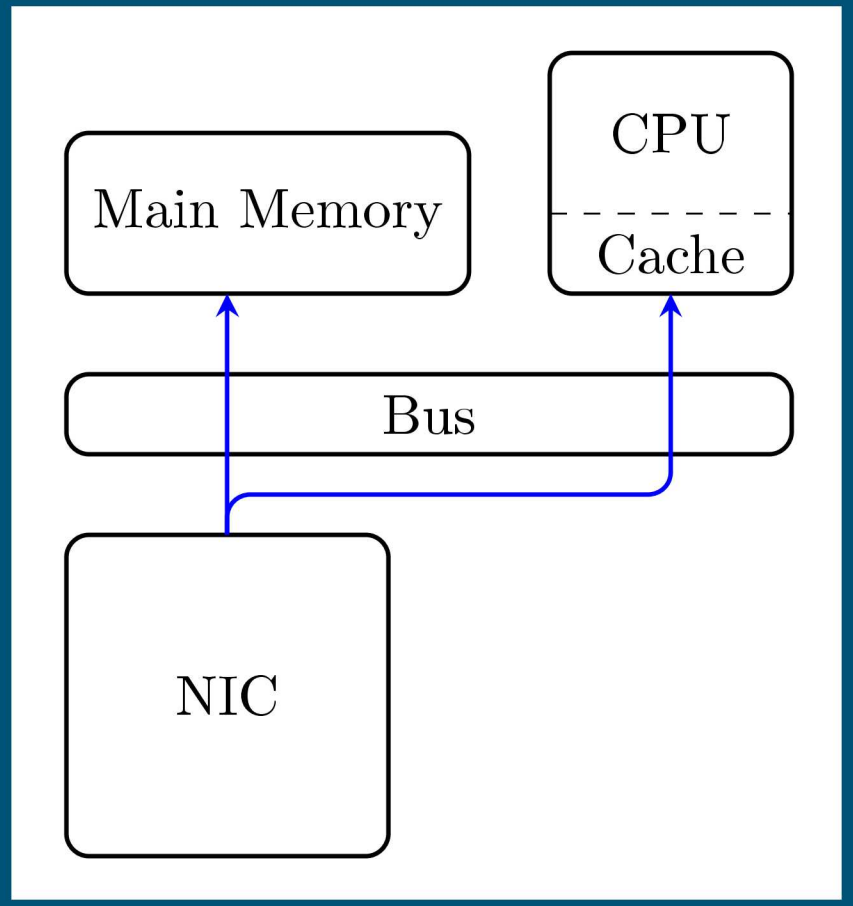
	MiniAMR	MiniMD	MiniFE	LAMMPS
Runtime	174	43.1	37	39.2
Communication	52.8	6.98	4.36	6.23
INCA Target	45.4	4.73	1.1	3.8
Overlap Target	76.3	34.3	22.3	21.5
Potential speedup without refactor		11%	2.98%	11.50%
Potential speedup with refactor	26%	37.20%	25.70%	28.90%

Interrupting INCA (1)

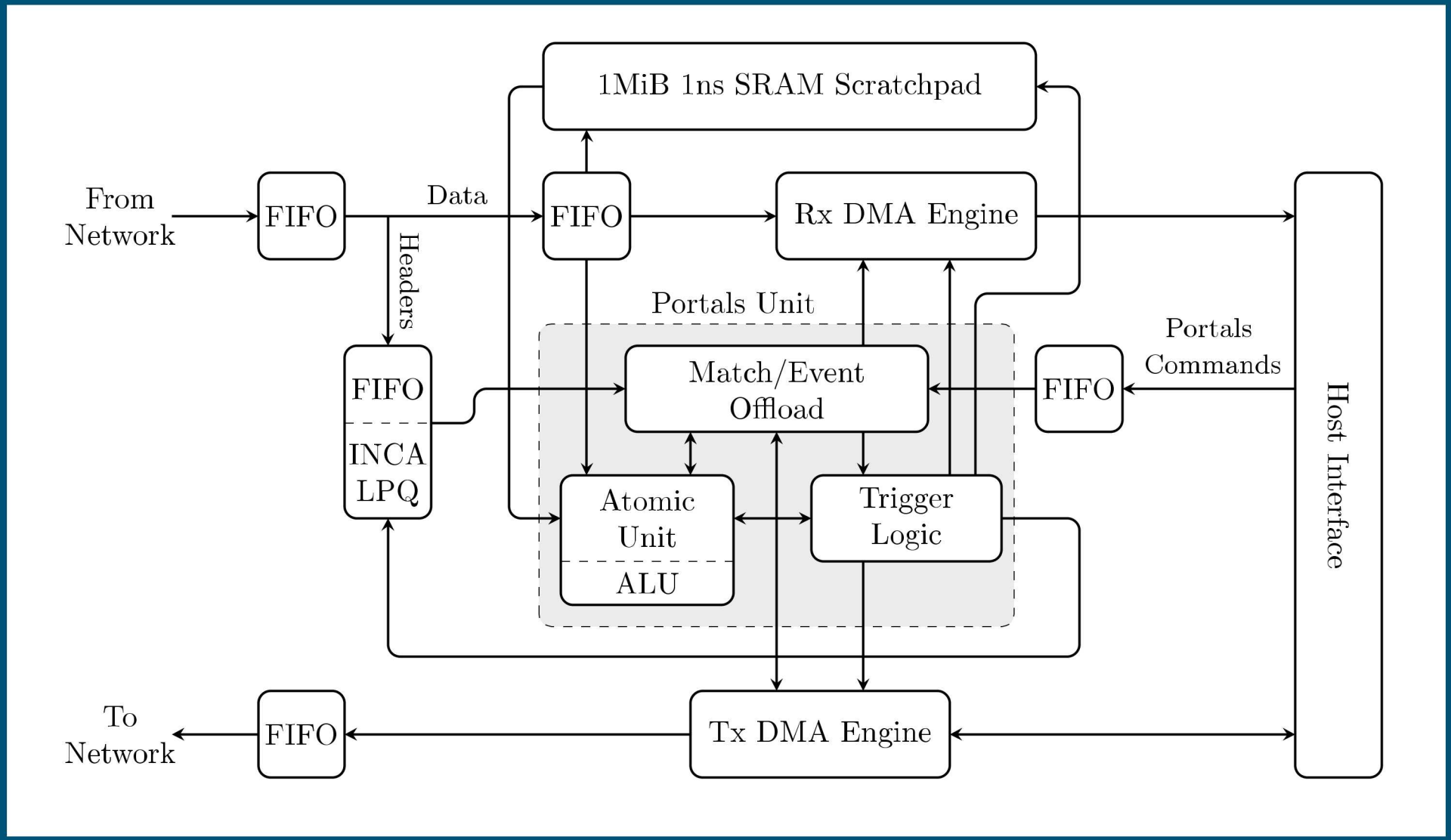


Interrupting INCA (II)

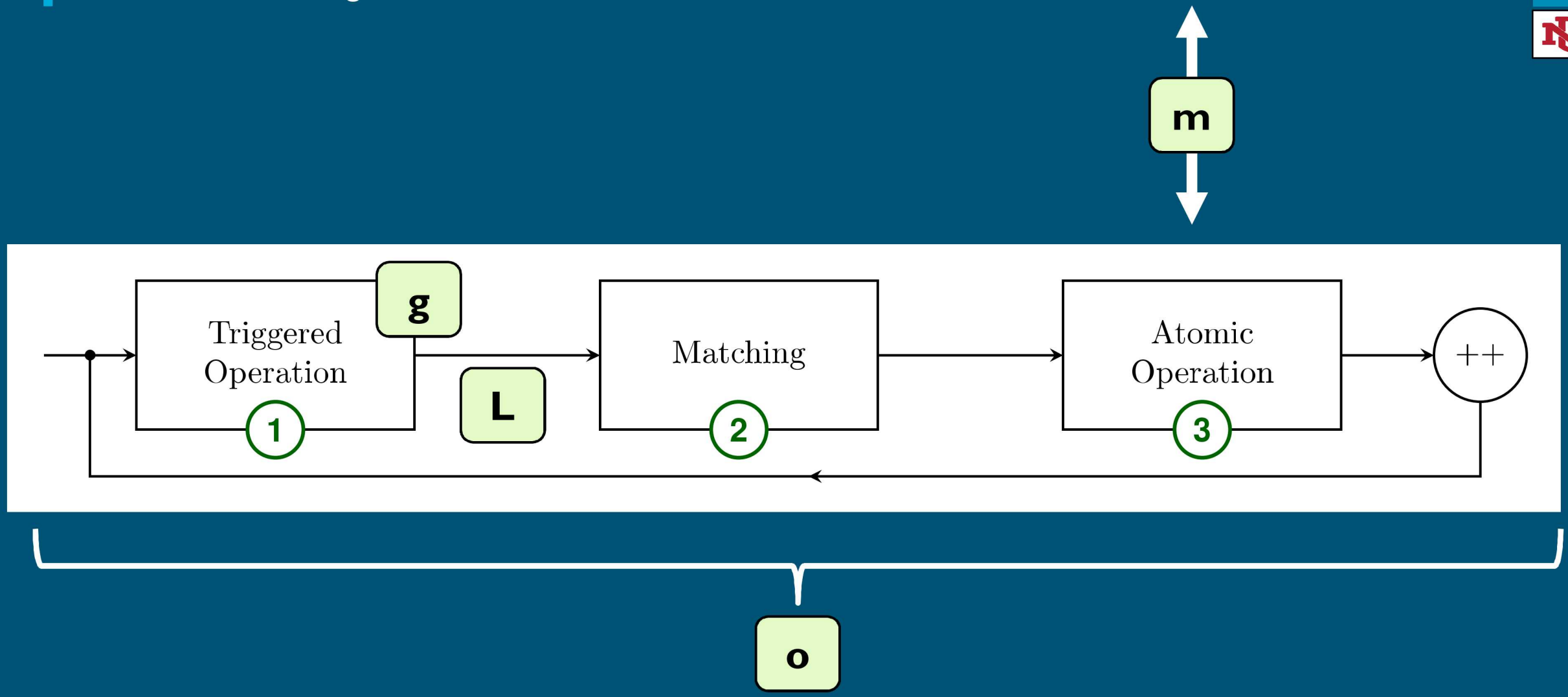




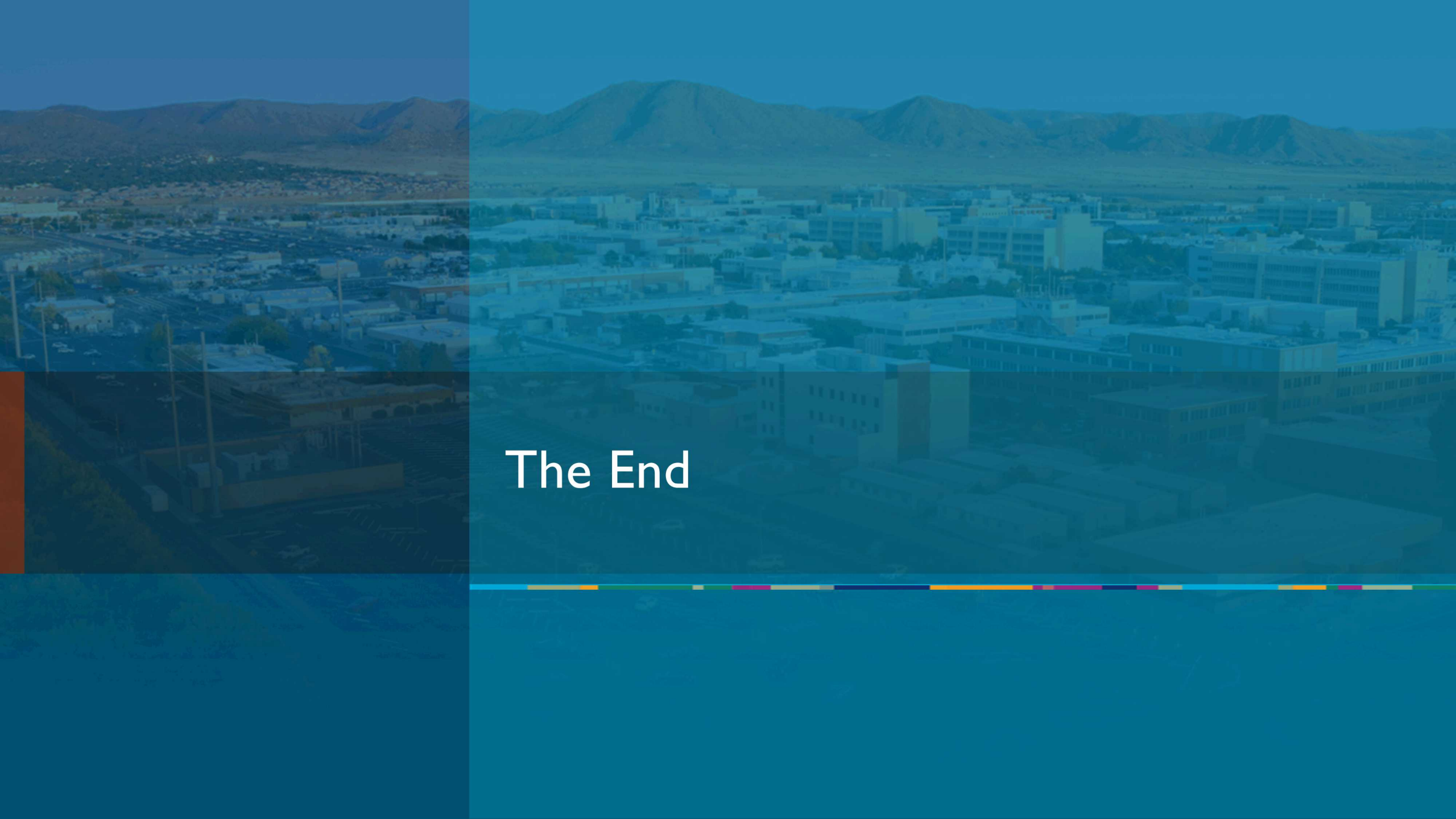
INCA Scratchpad architecture



The extended LogGP model



Time to execute instruction = L + g + o + m



The End