SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI

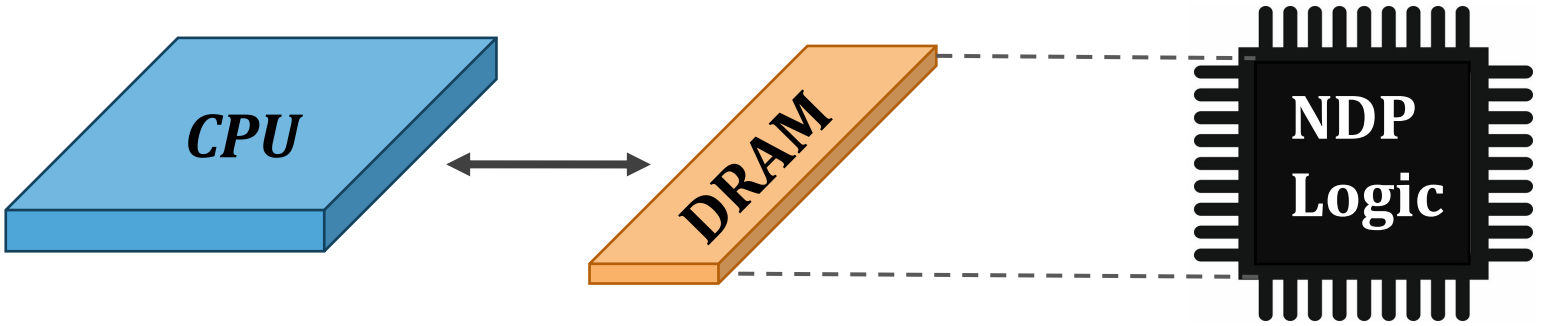
ETH zürich

National Technical University of Athens
CSLab

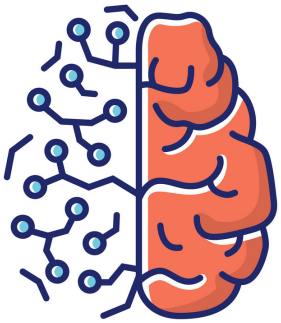
UNIVERSITY OF TORONTO

UNIVERSIDAD DE MÁLAGA

Near-Data-Processing (**NDP**) Systems

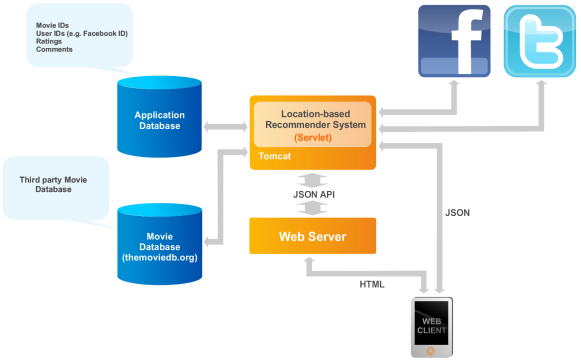


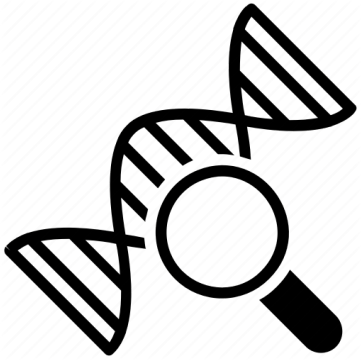
Graph Analytics



Neural Networks

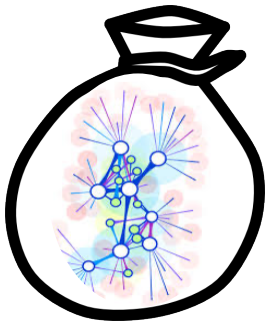
Recommendation Systems





Bioinformatics

Synchronization is Necessary

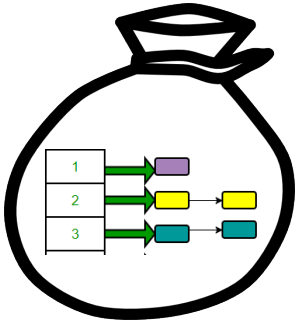


Graph Analytics

Bioinformatics

Databases

Image Processing



Concurrent Data Structures

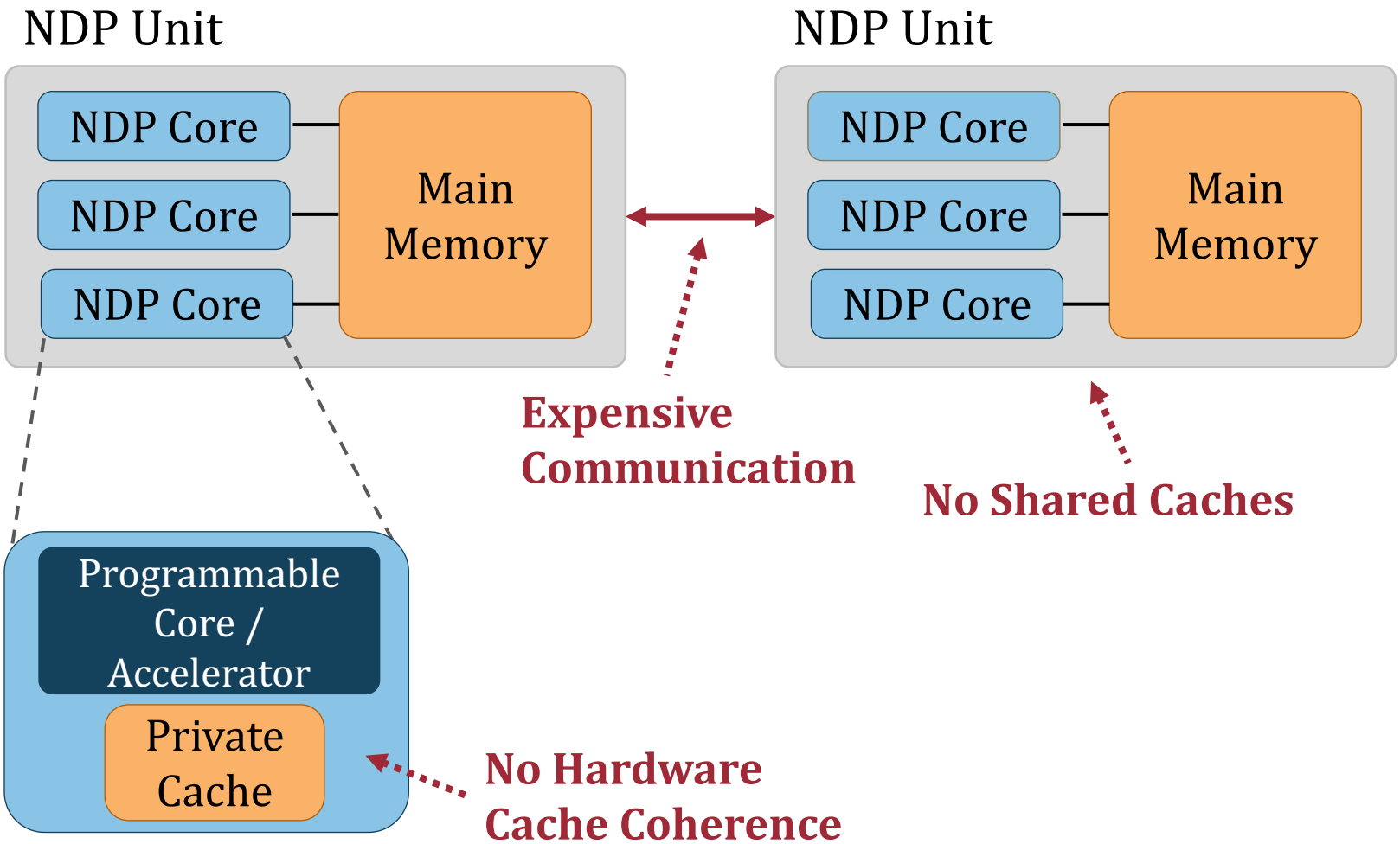
Single Source Shortest Path (SSSP)

Locks

Barriers

```
for v in Graph:
   for u in neighbors[v]:
      if distance[v] + edge_weight[v, u] < distance[u]
         lock_acquire(u)
         if distance[v] + edge_weight[v, u] < distance[u]
            distance[u] = distance[v] + edge_weight[v, u]
         lock_release(u)
```

Challenge: Efficient Synchronization



SynCron

The first end-to-end synchronization solution for NDP architectures

SynCron's Benefits:

1. High System Performance
2. Low Hardware Cost
3. Programming Ease
4. General Synchronization Support

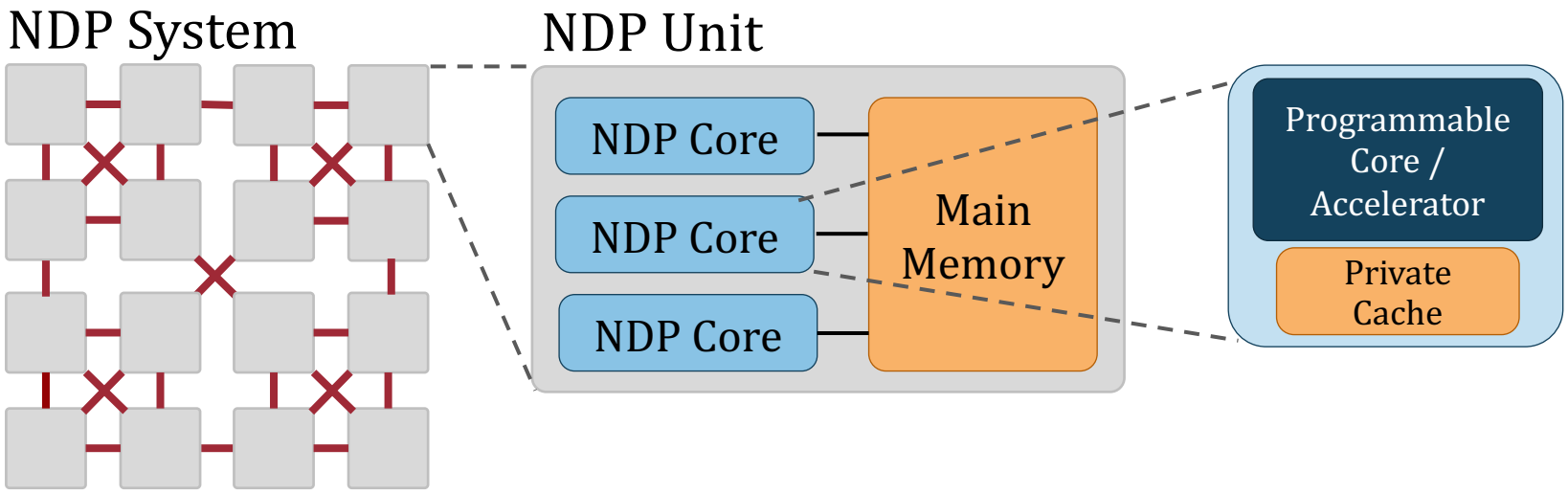
Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation

Baseline NDP Architecture



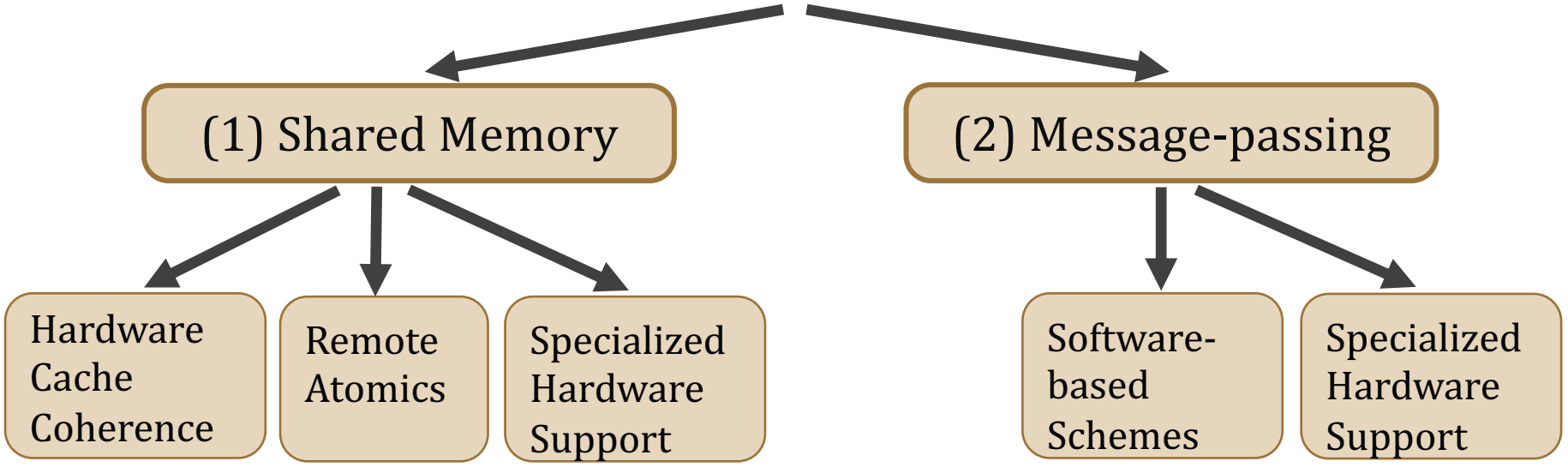
Synchronization challenges in NDP systems:

(1) Lack of hardware cache coherence support

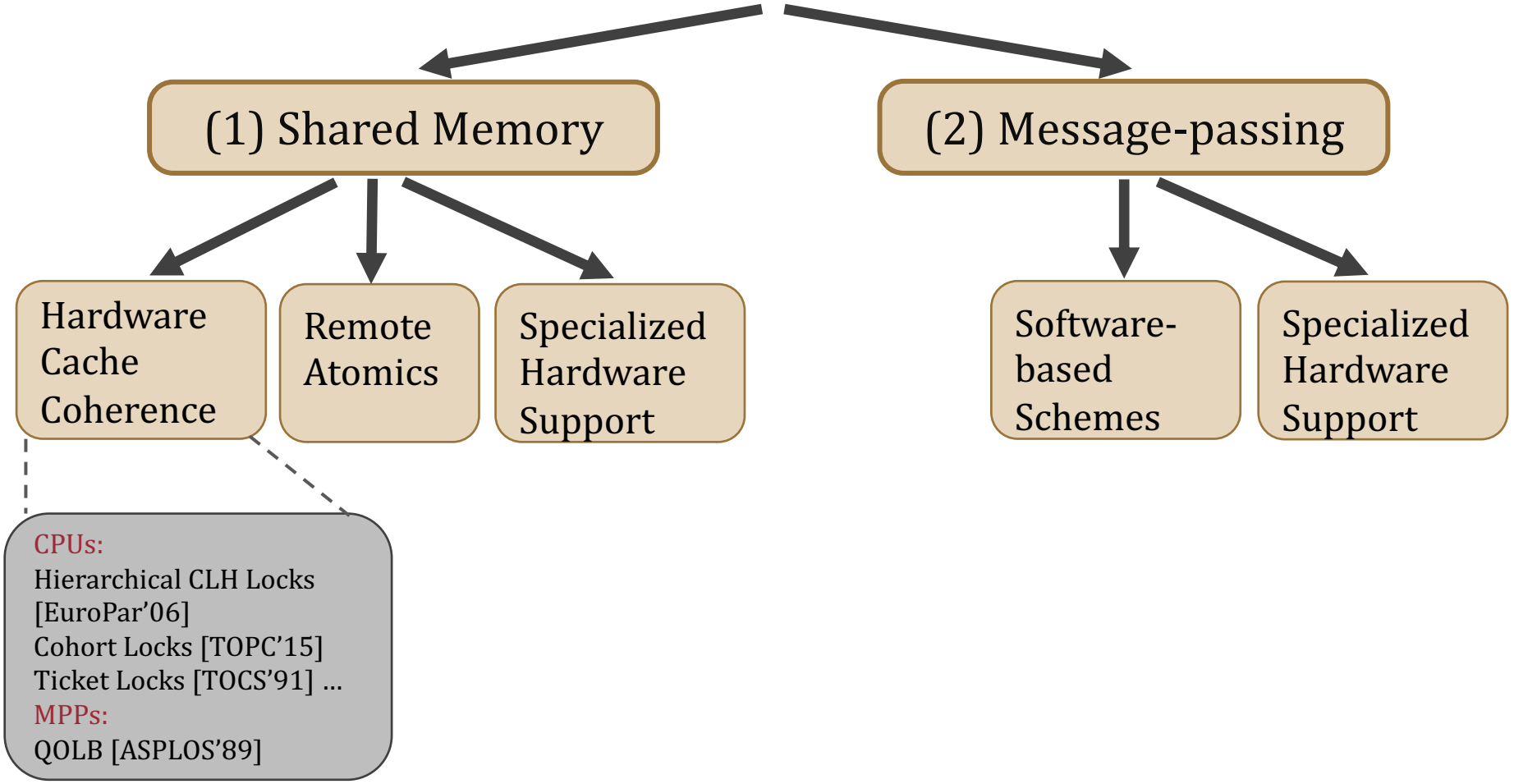
(2) Expensive communication across NDP units

(3) Lack of a shared level of cache memory

NDP Synchronization Solution Space

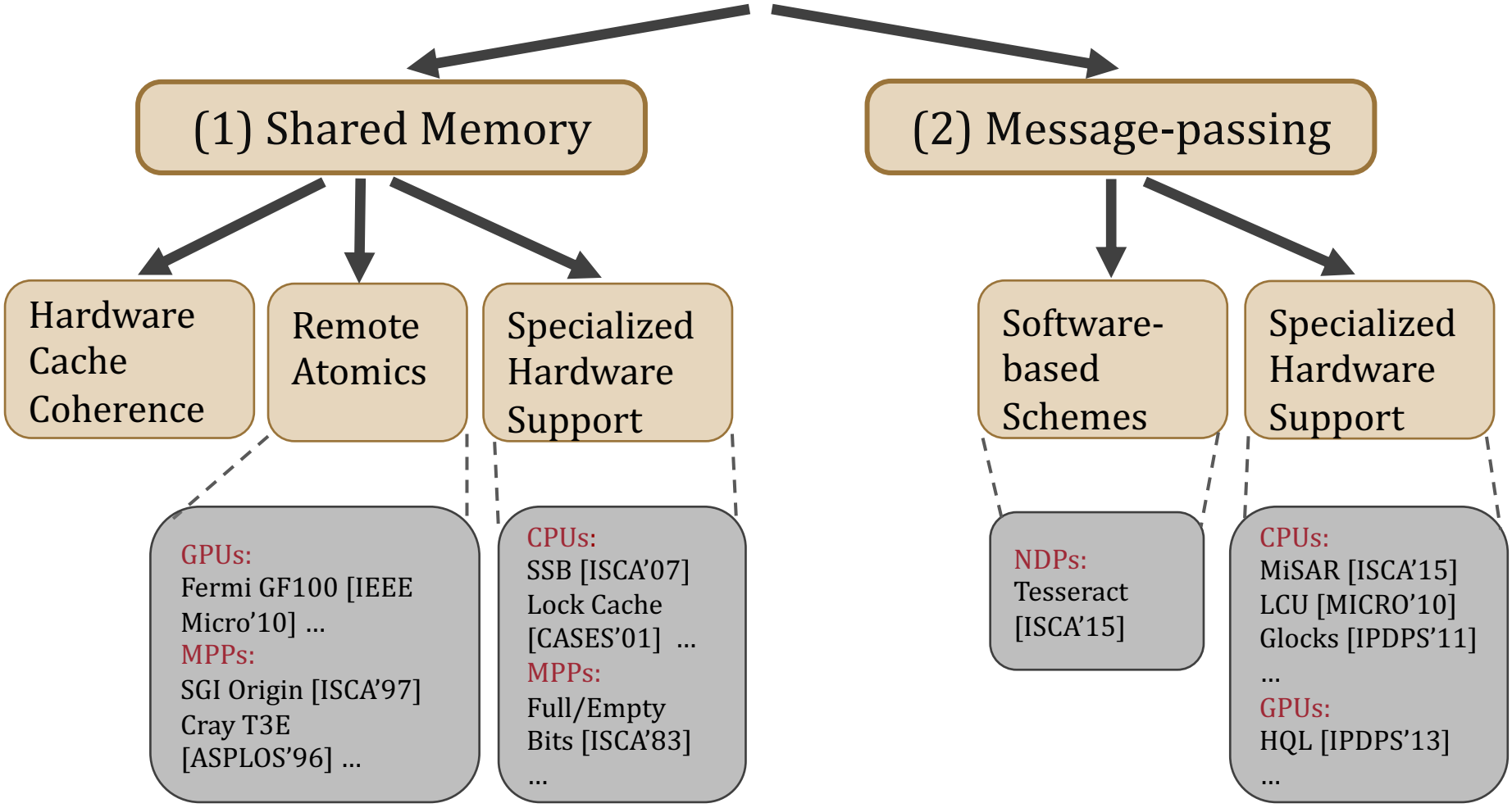


NDP Synchronization Solution Space



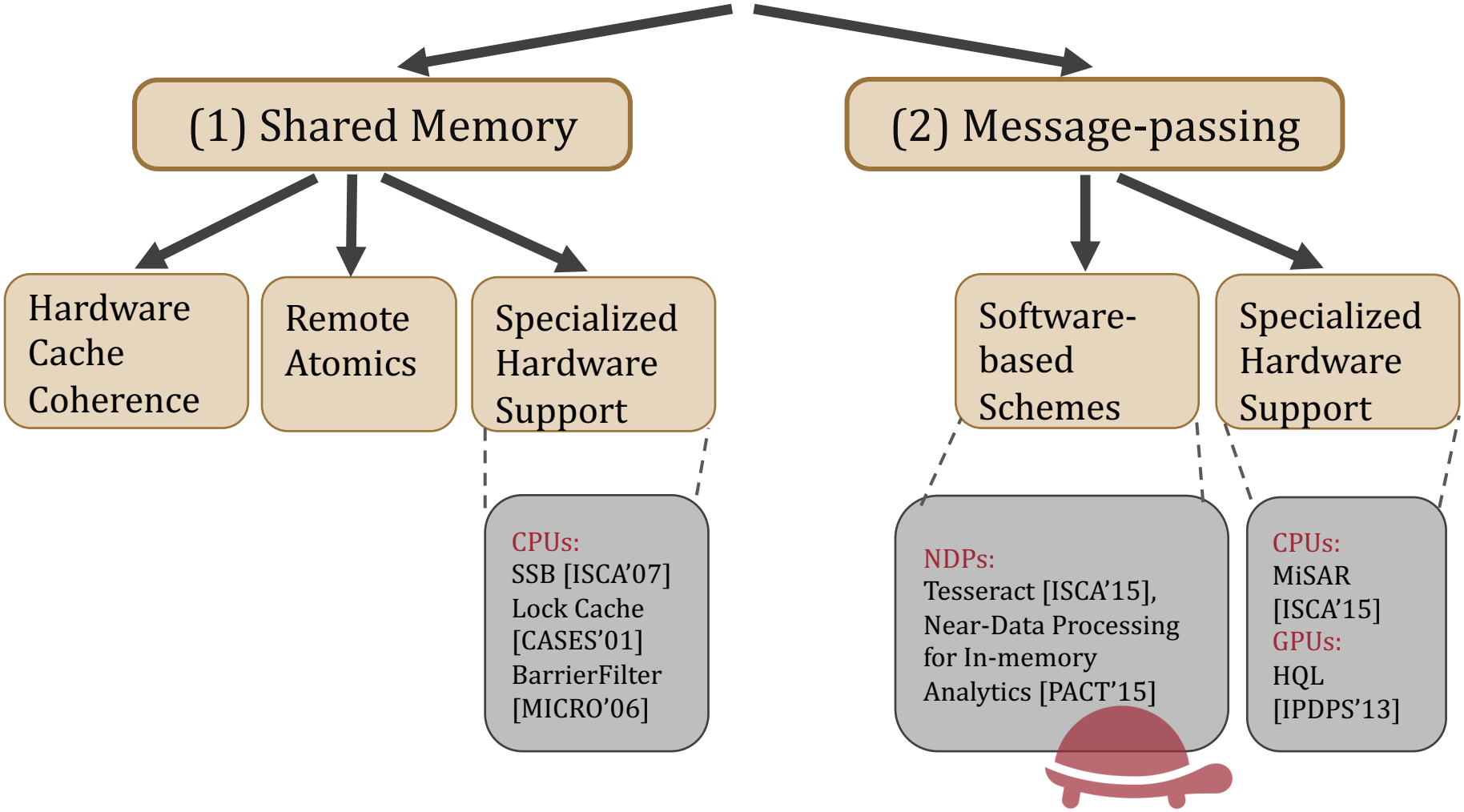
Lack of hardware cache coherence support

NDP Synchronization Solution Space



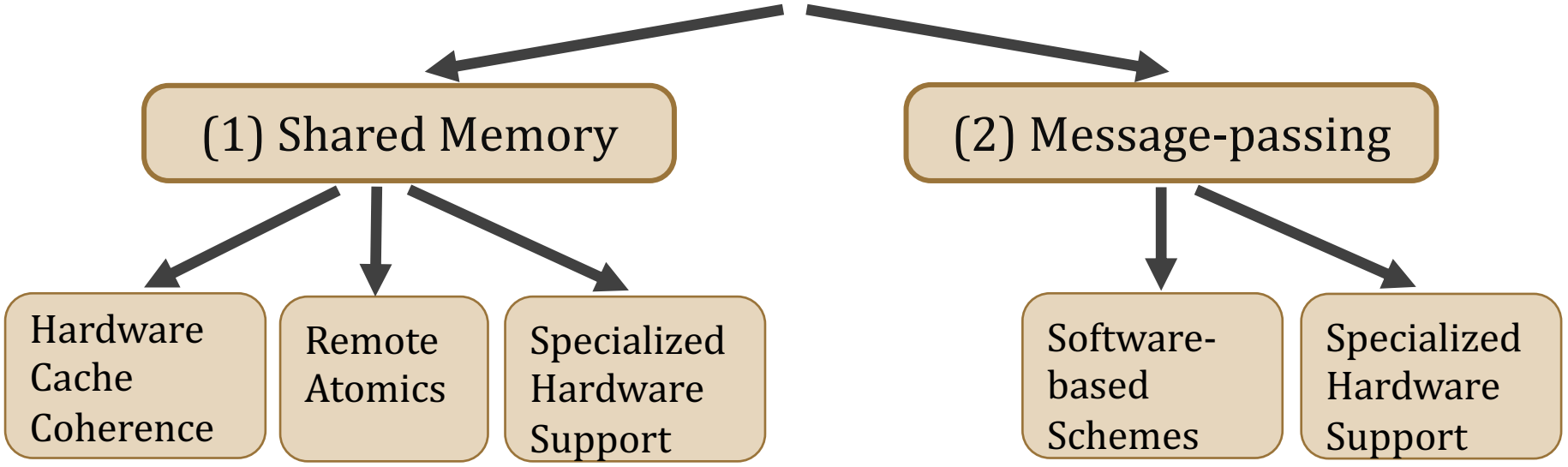
Expensive communication across NDP units

NDP Synchronization Solution Space



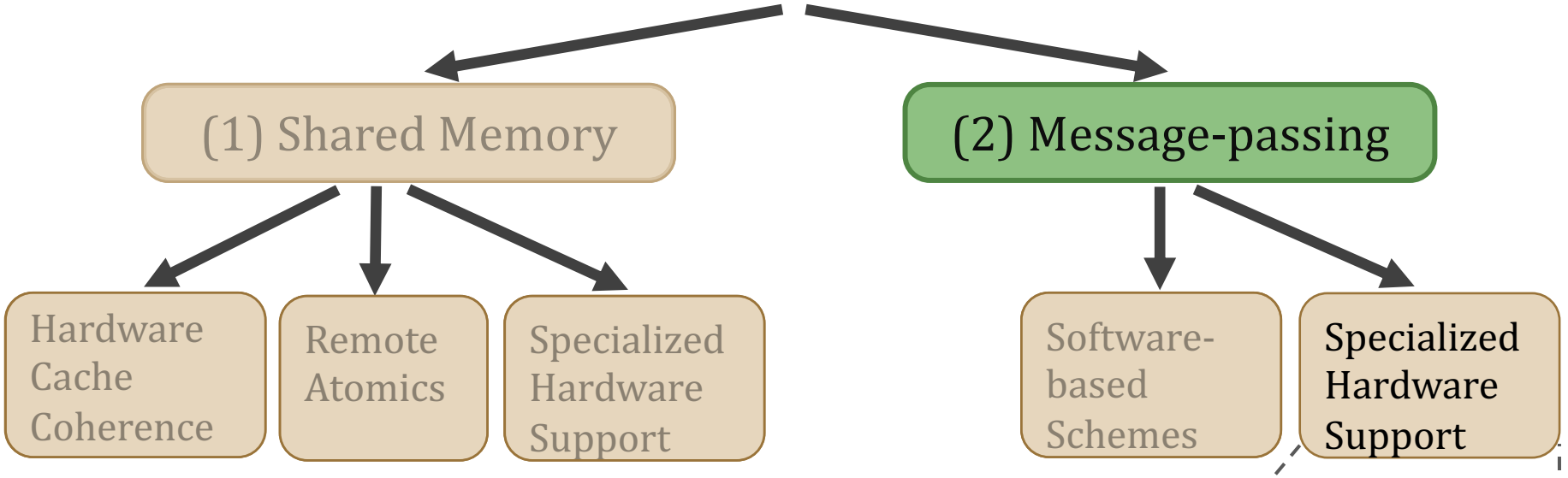
Lack of a shared level of cache memory

NDP Synchronization Solution Space



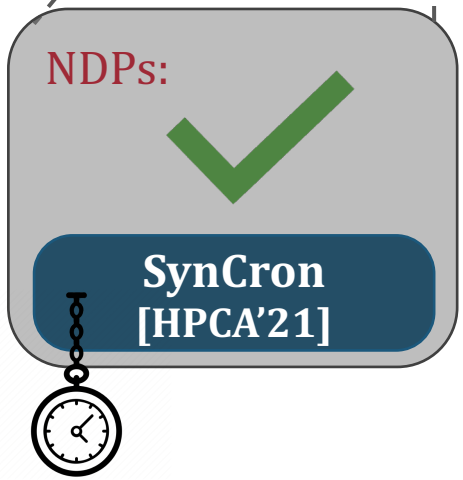
Prior schemes are not suitable or efficient for NDP systems

NDP Synchronization Solution Space



SynCron's Design Choices

- Hardware Message-passing to Avoid Synchronization via Shared Memory
- Hierarchical Communication to Eliminate Expensive Network Traffic
- Specialized Cache Structure to Minimize Latency Costs



Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

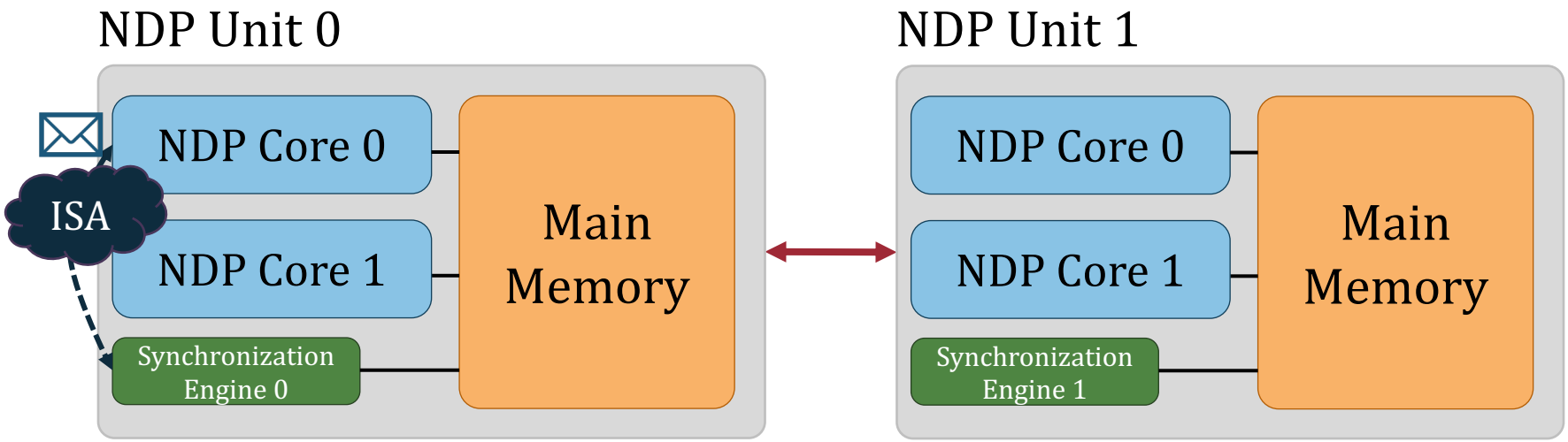
Evaluation

SynCron: Overview

SynCron consists of **four key techniques**:

1. **Hardware support** for synchronization acceleration
2. **Direct buffering** of synchronization variables
3. **Hierarchical** message-passing **communication**
4. Integrated hardware-only **overflow management**

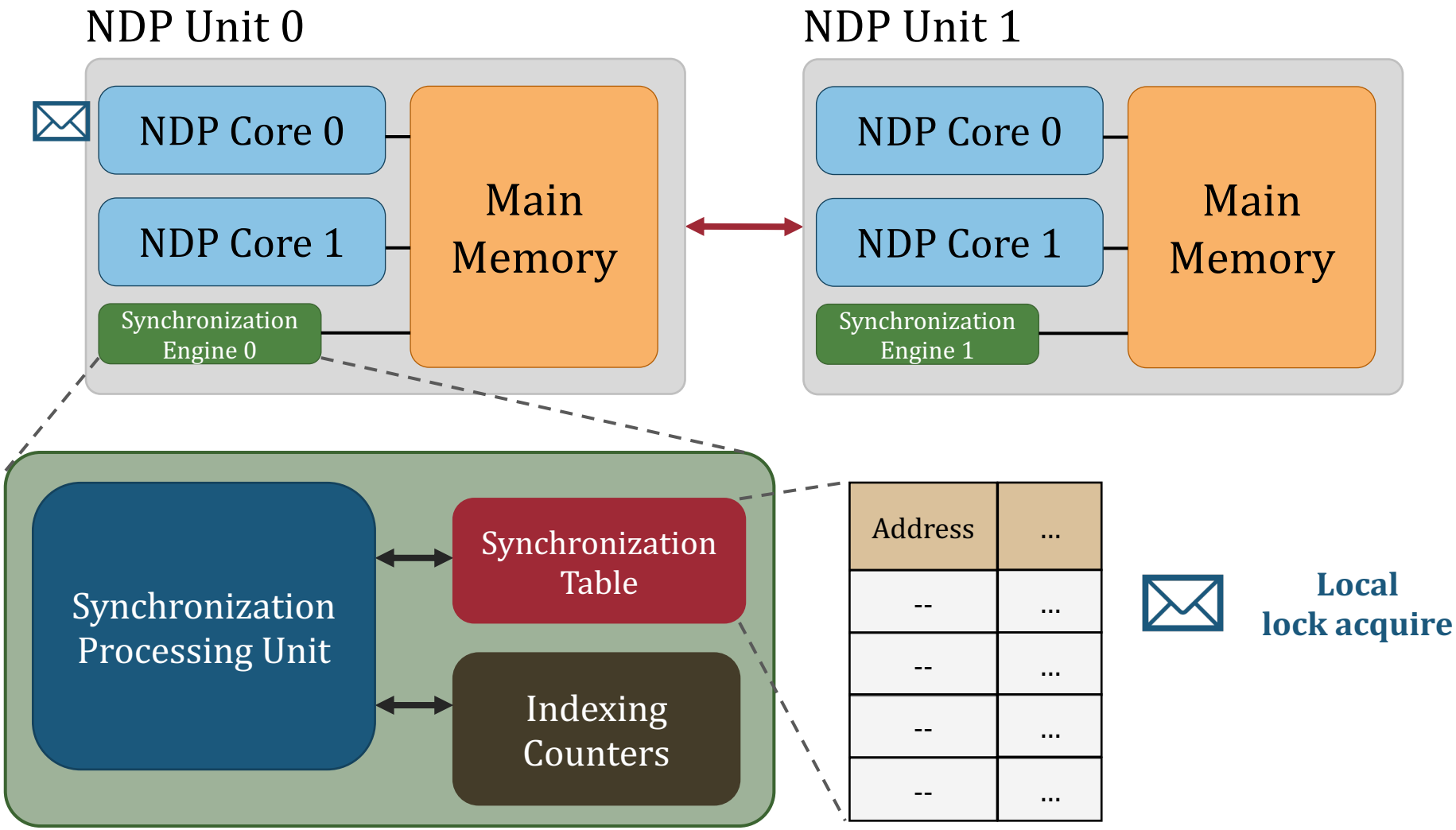
1. Hardware Synchronization Support



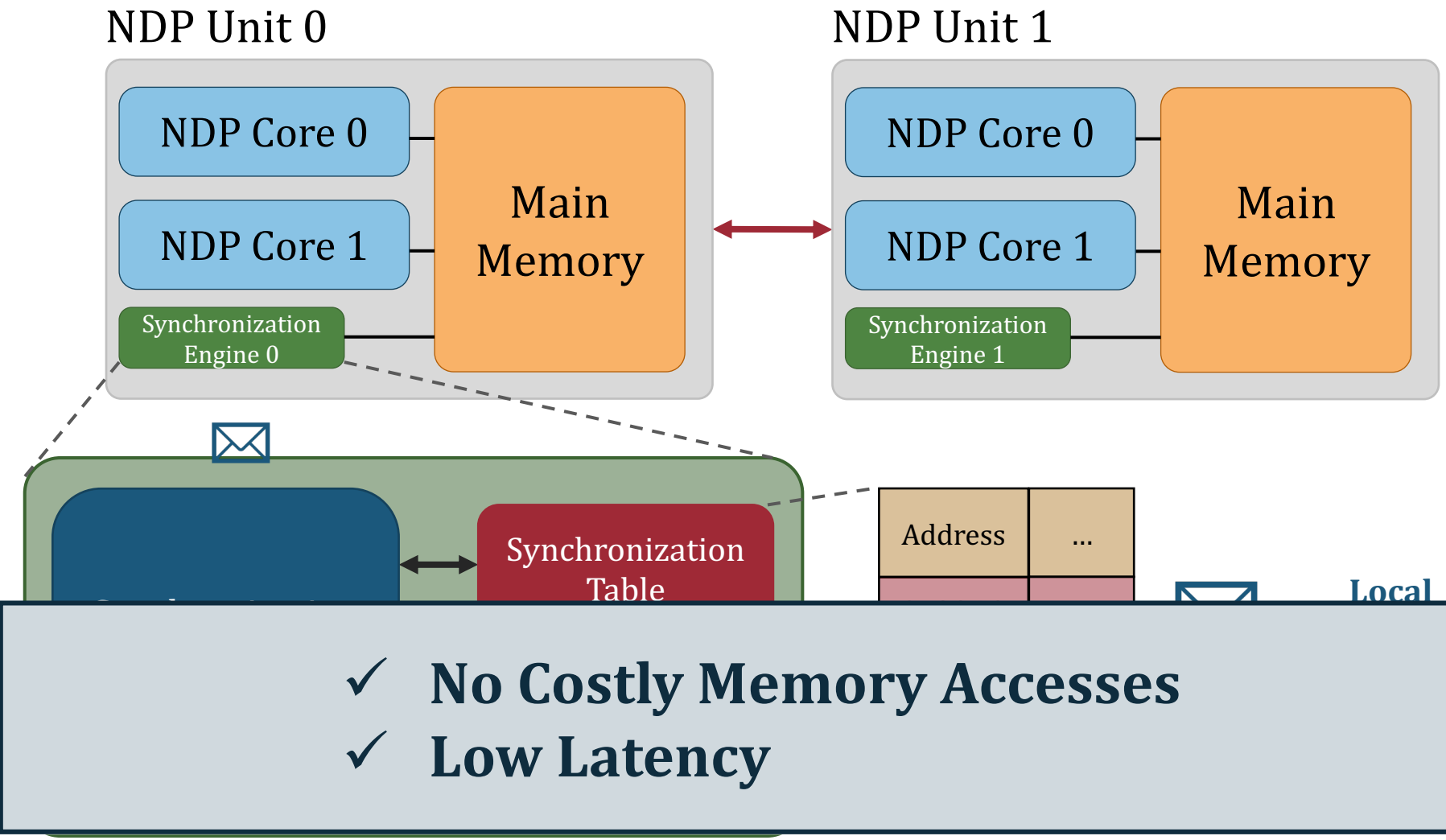
Local lock acquire

- ✓ **No Complex Cache Coherence Protocols**
- ✓ **No Expensive Atomic Operations**
- ✓ **Low Hardware Cost**

2. Direct Buffering of Variables

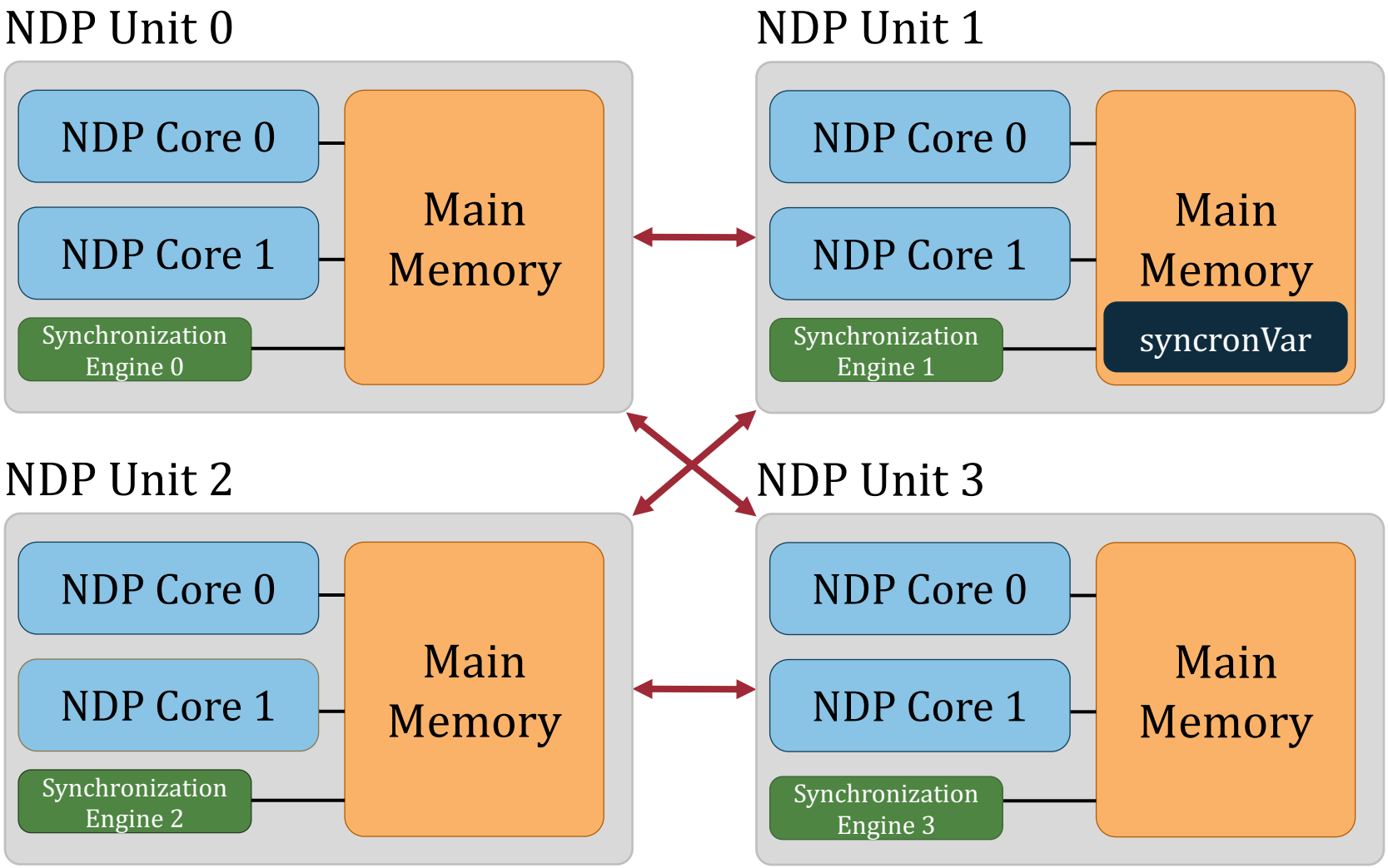


2. Direct Buffering of Variables

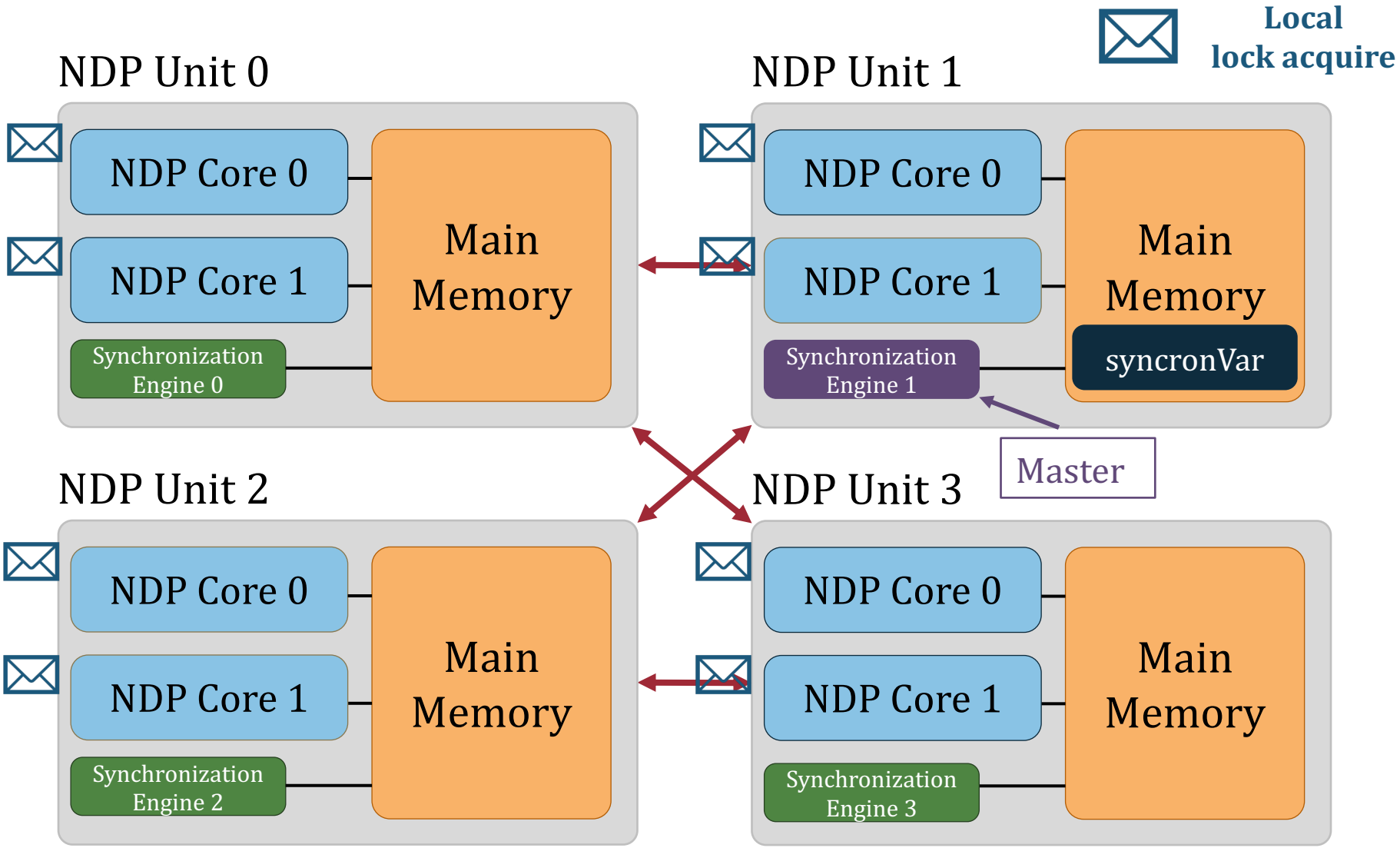


- ✓ **No Costly Memory Accesses**
- ✓ **Low Latency**

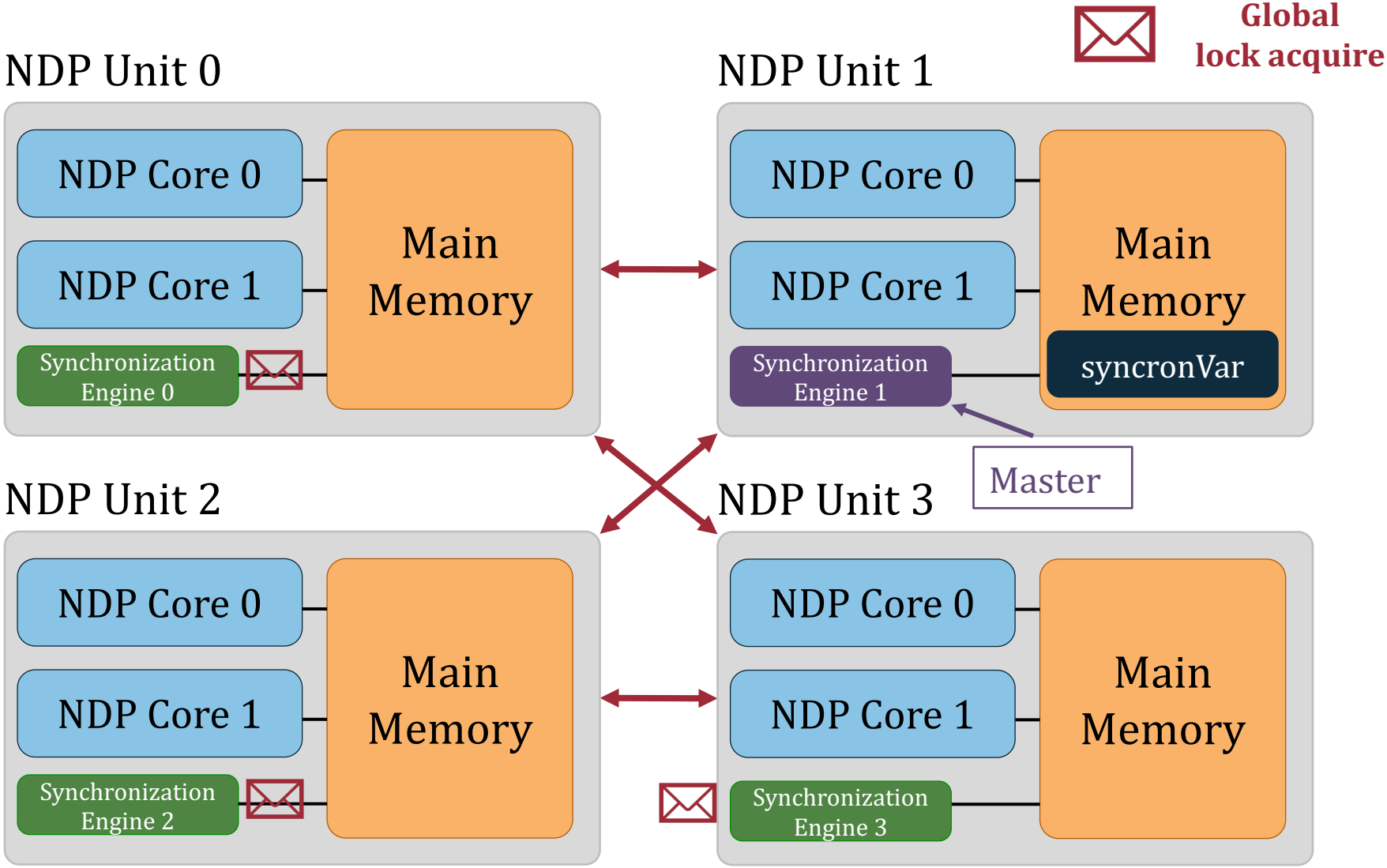
3. Hierarchical Communication



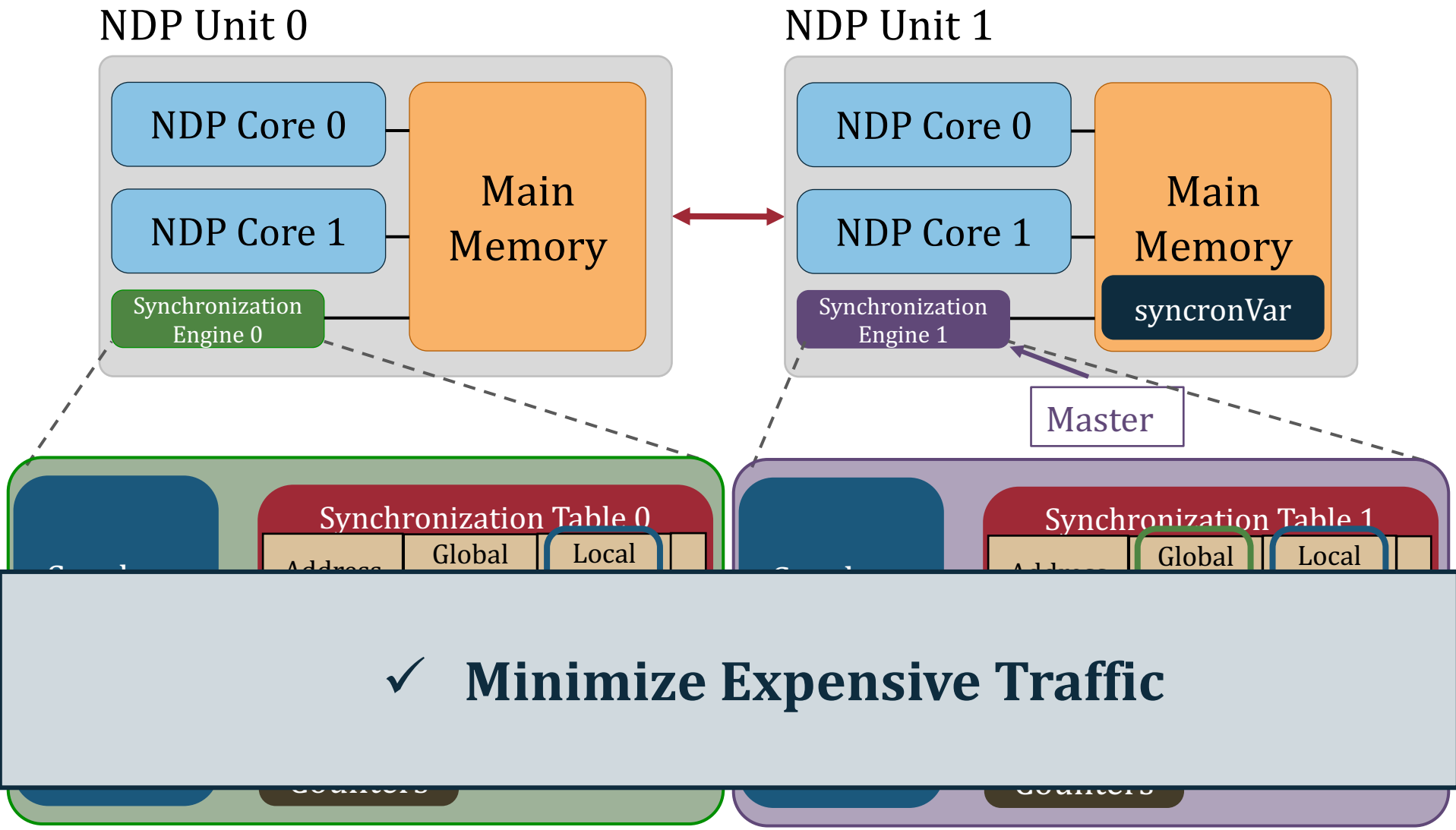
3. Hierarchical Communication



3. Hierarchical Communication

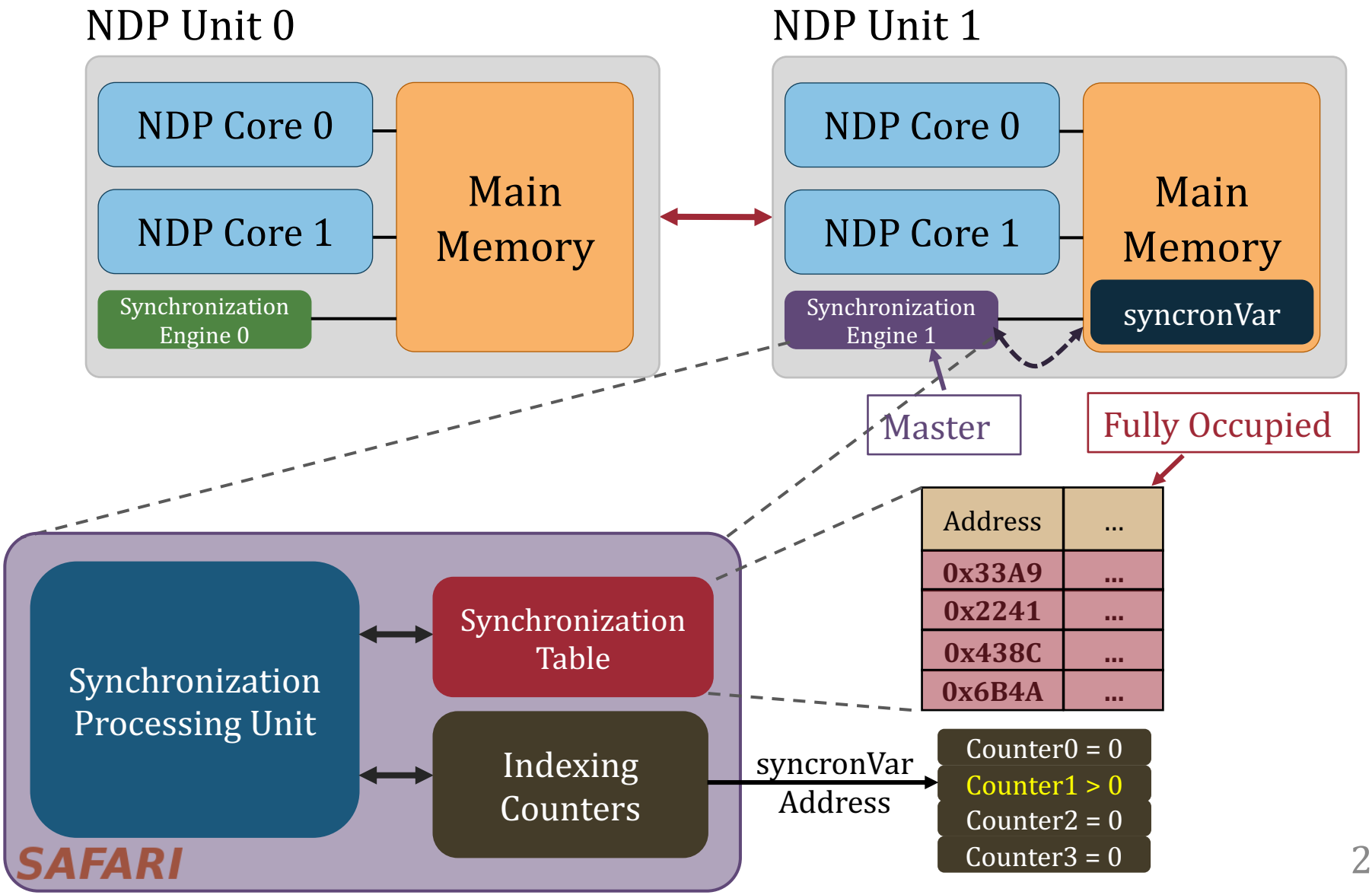


3. Hierarchical Communication

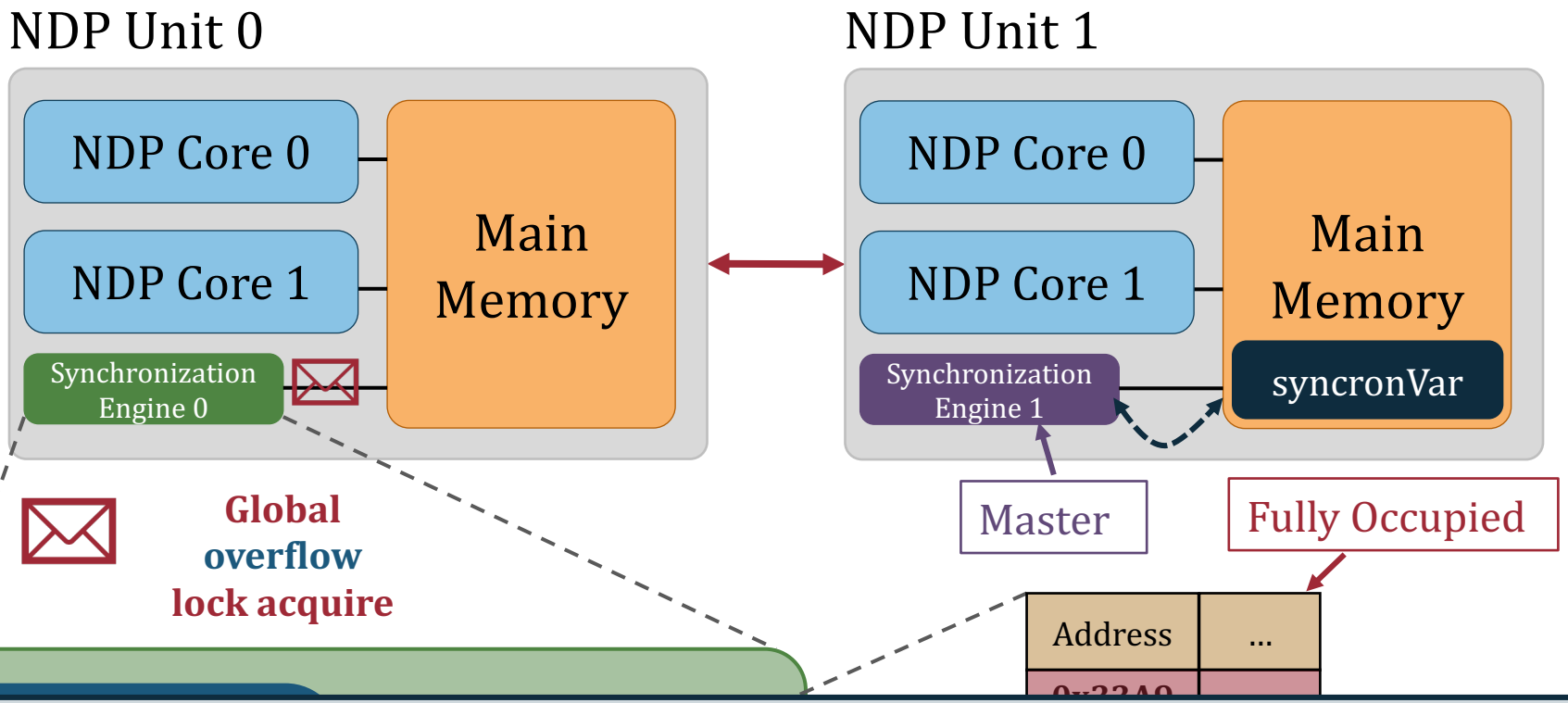


✓ **Minimize Expensive Traffic**

4. Integrated Overflow Management



4. Integrated Overflow Management



- ✓ **Low Performance Degradation**
- ✓ **High Programming Ease**

SynCron's Supported Primitives

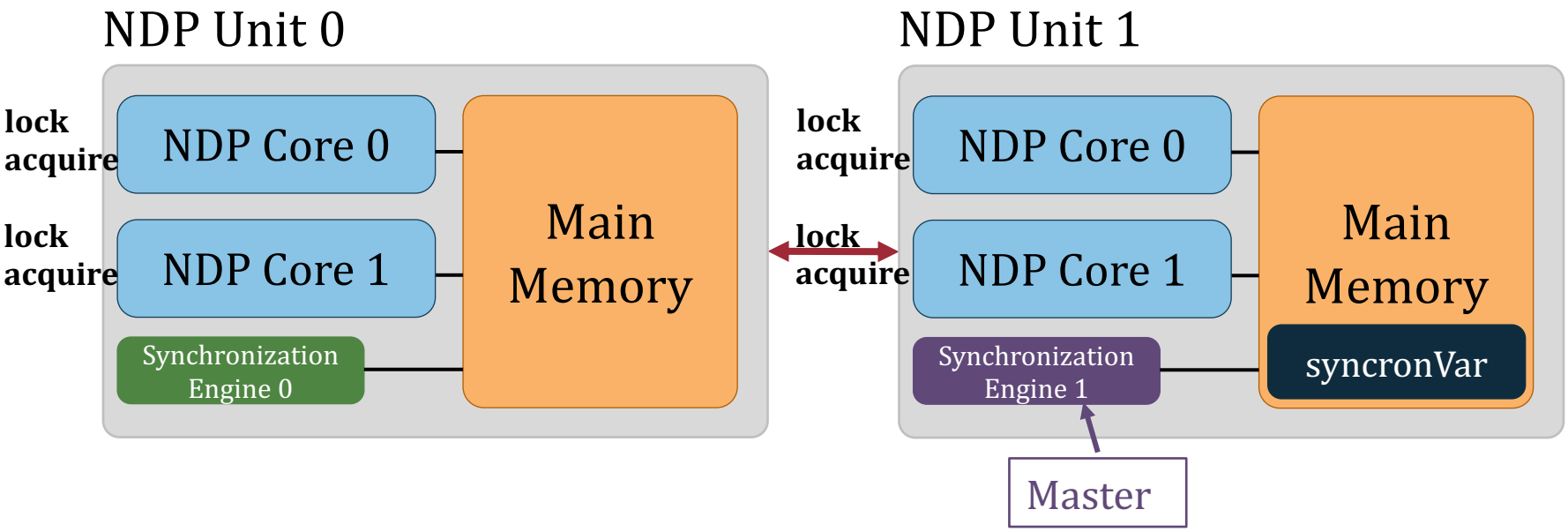
Lock primitive
- lock_acquire()
- lock_release ()

Barrier primitive
- barrier_wait_within_NDP_unit()
- barrier_wait_across_NDP_units()

Semaphore primitive
- sem_wait()
- sem_post()

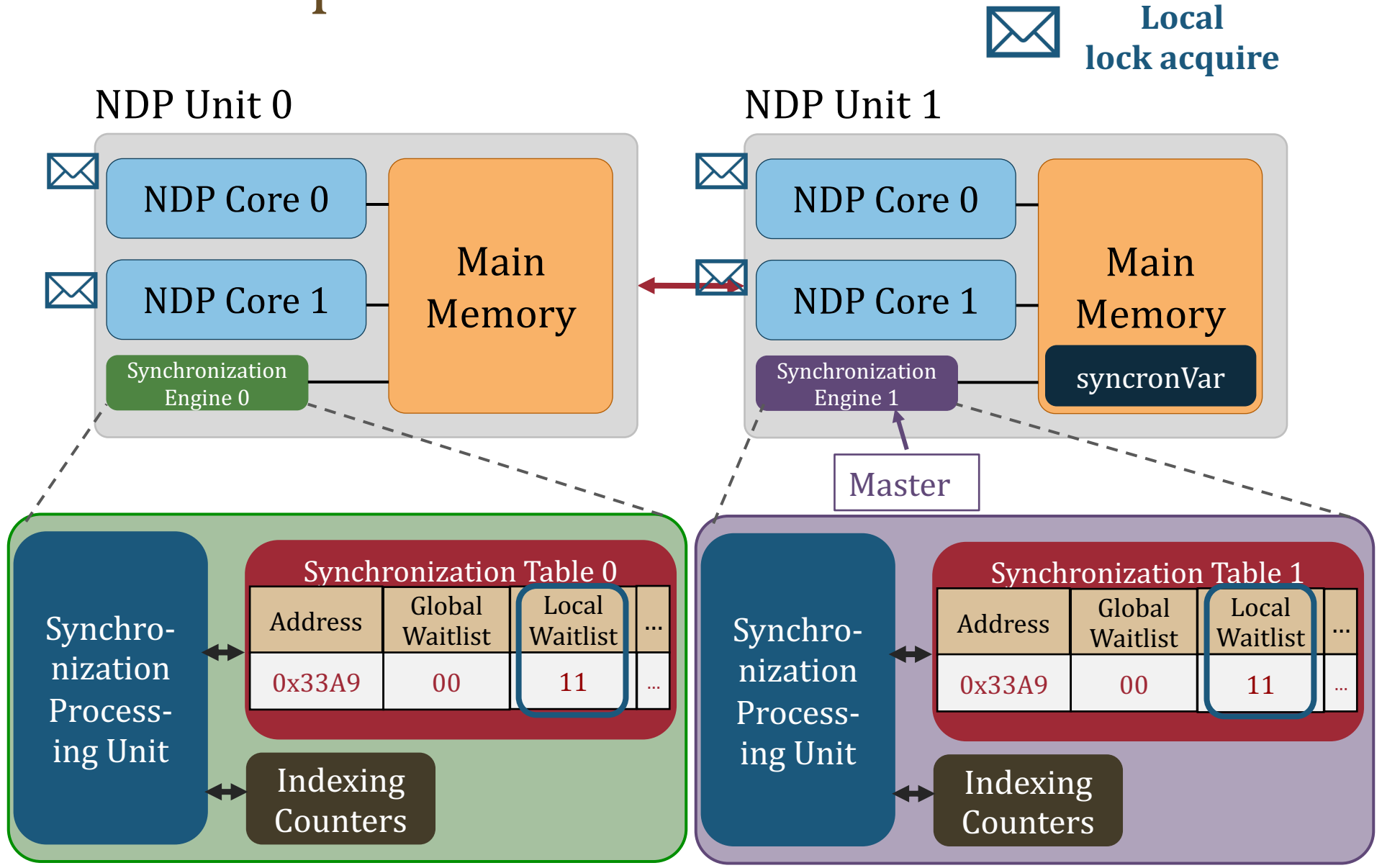
Condition variable primitive
- cond_wait()
- cond_signal()
- cond_broadcast()

Lock Operation

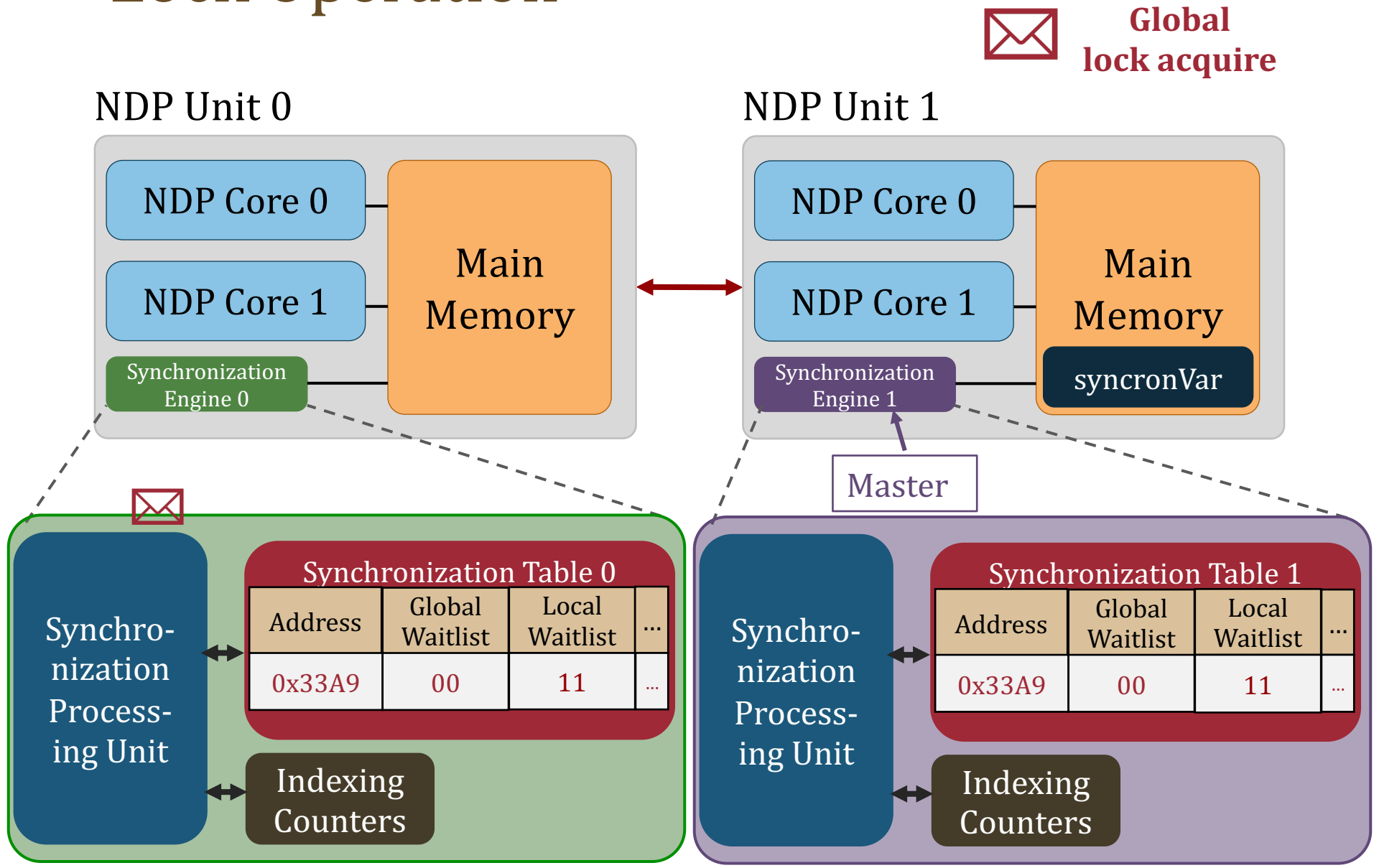


All NDP cores compete for the same lock variable

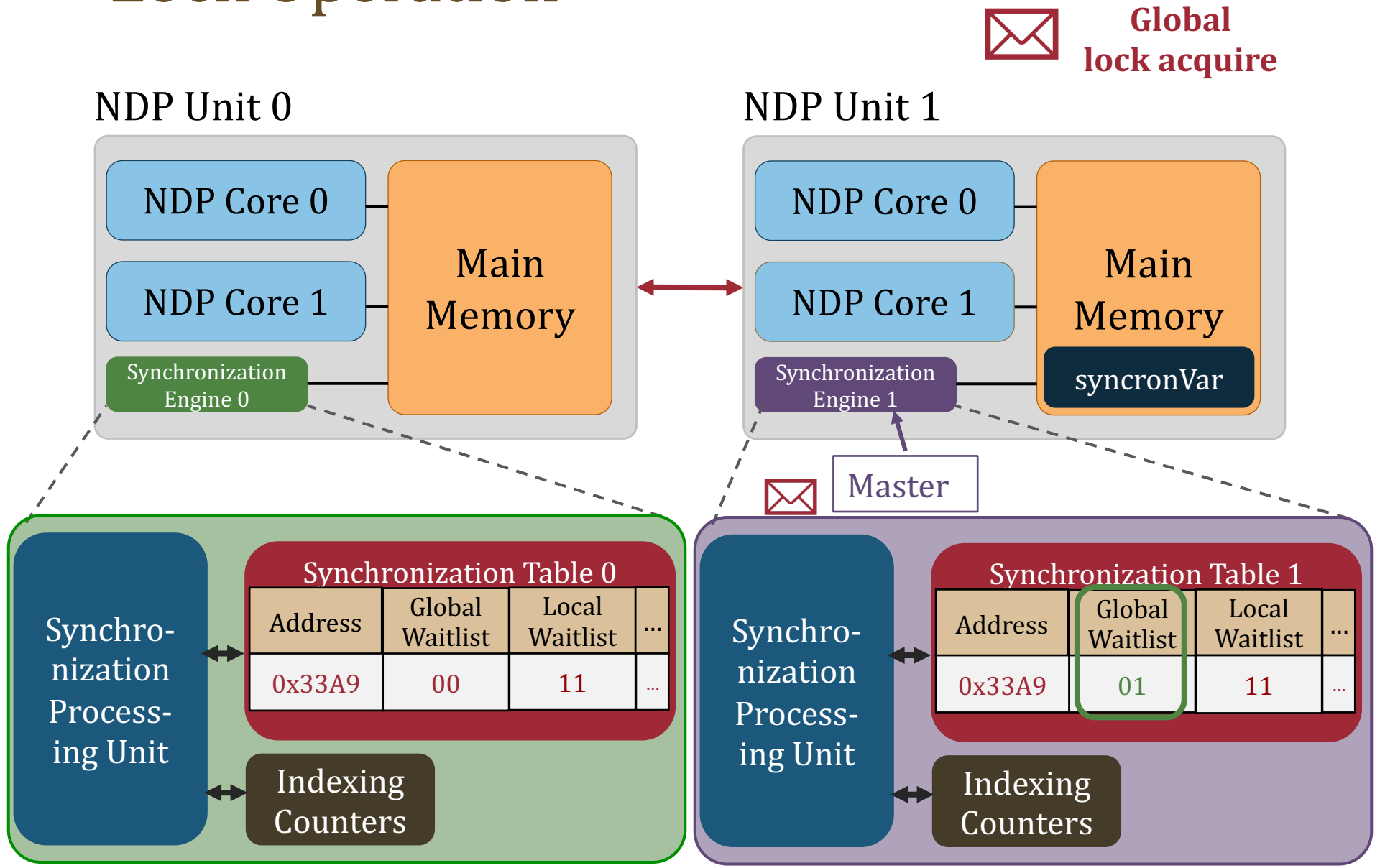
Lock Operation



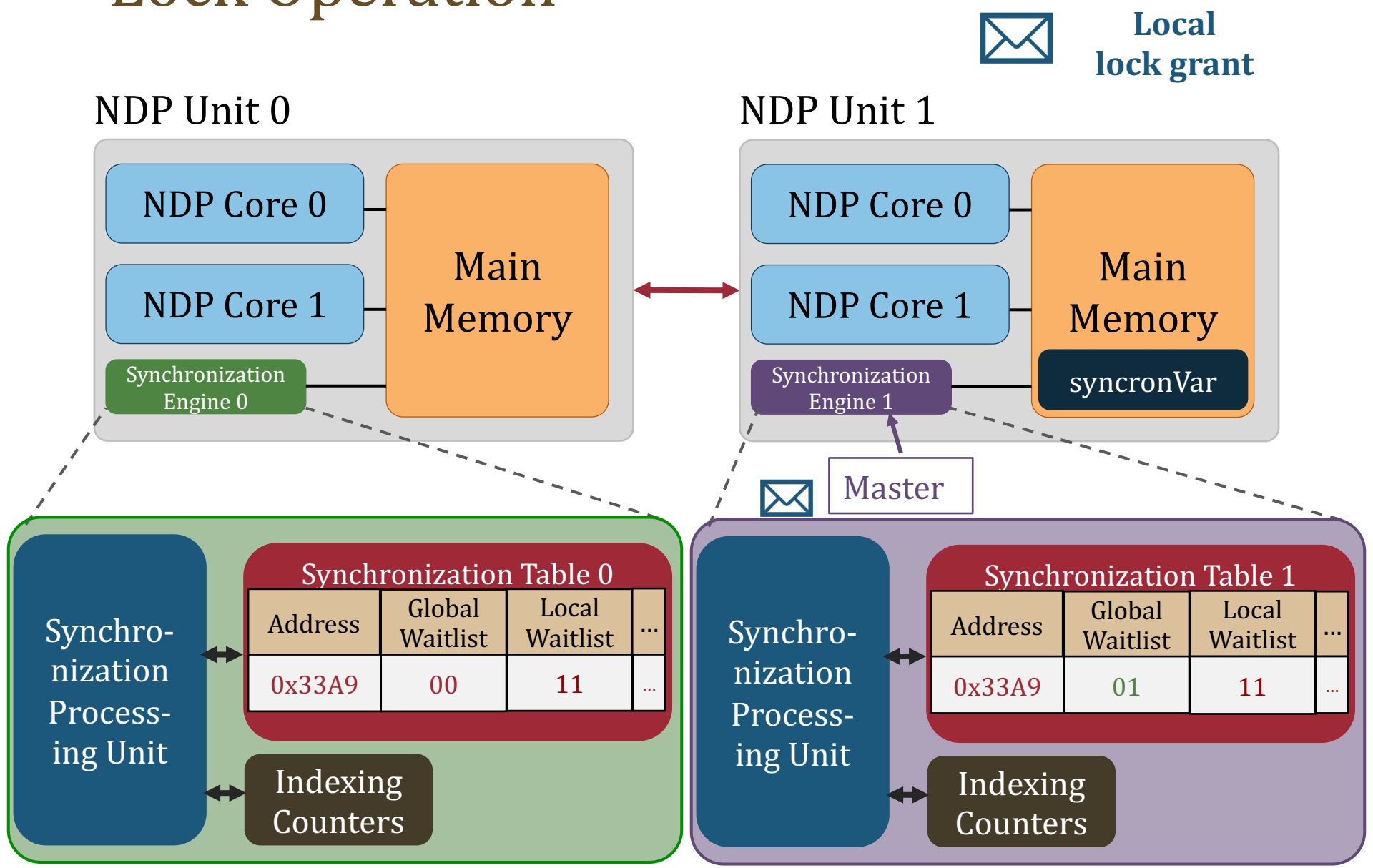
Lock Operation



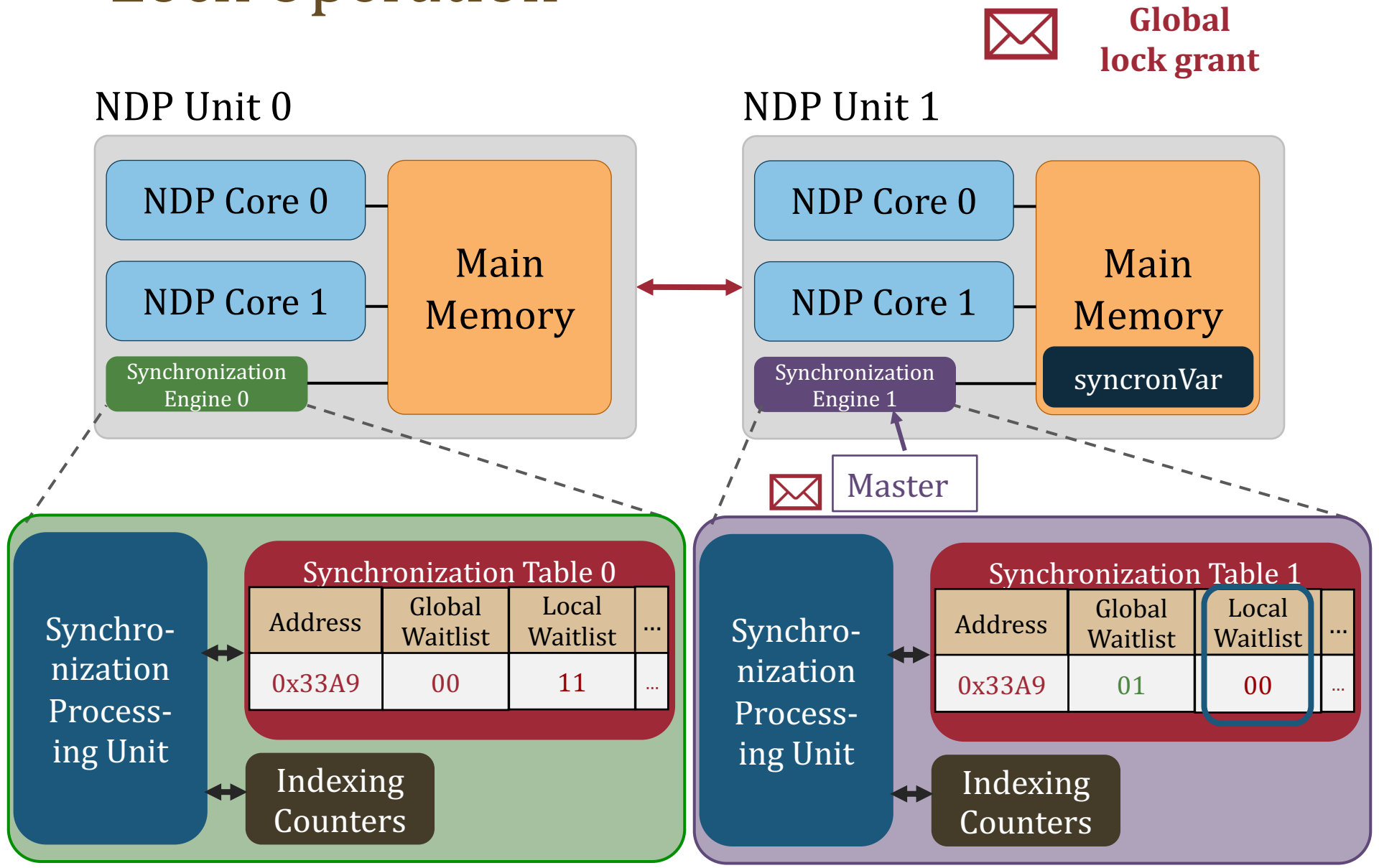
Lock Operation



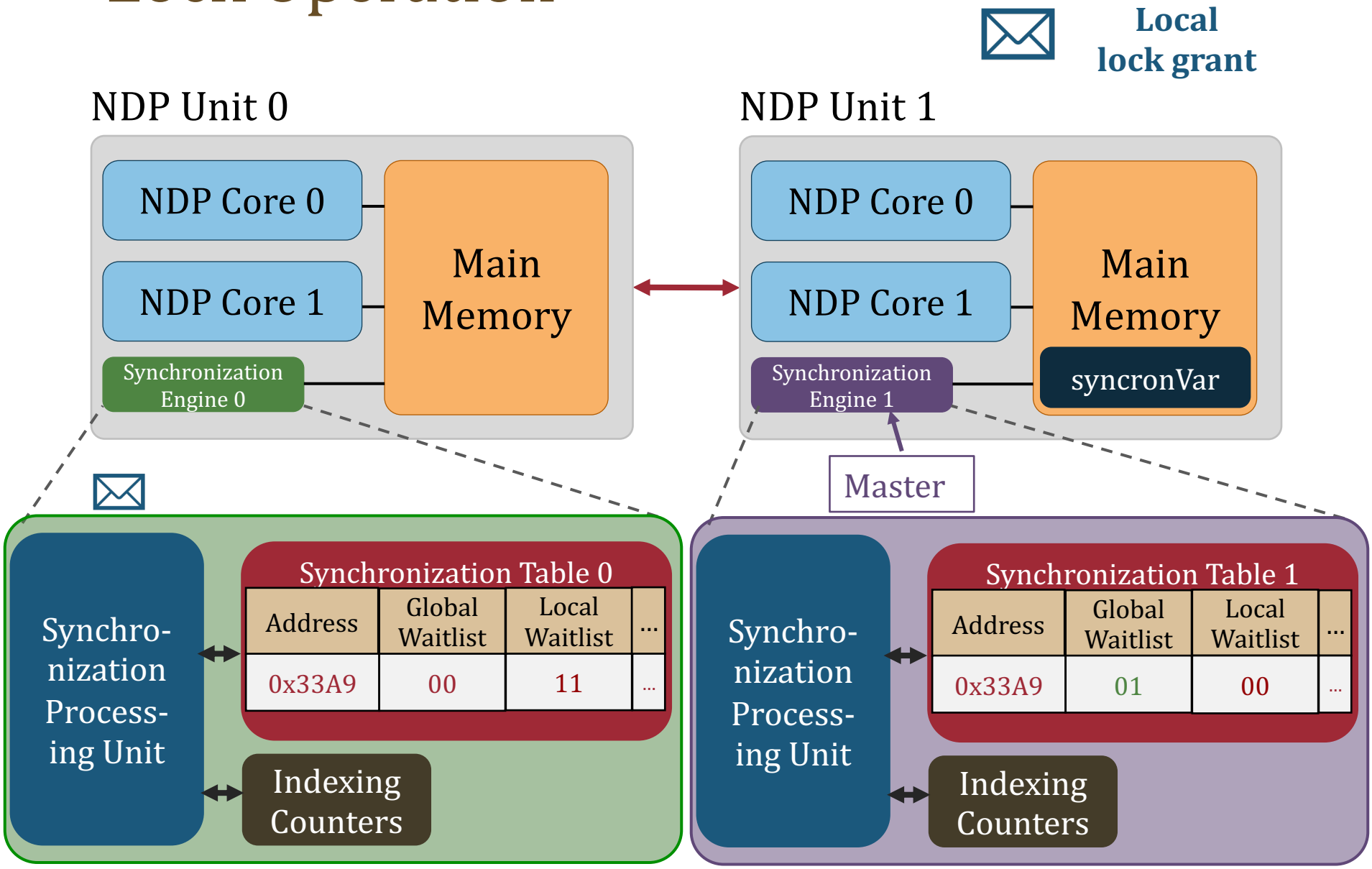
Lock Operation



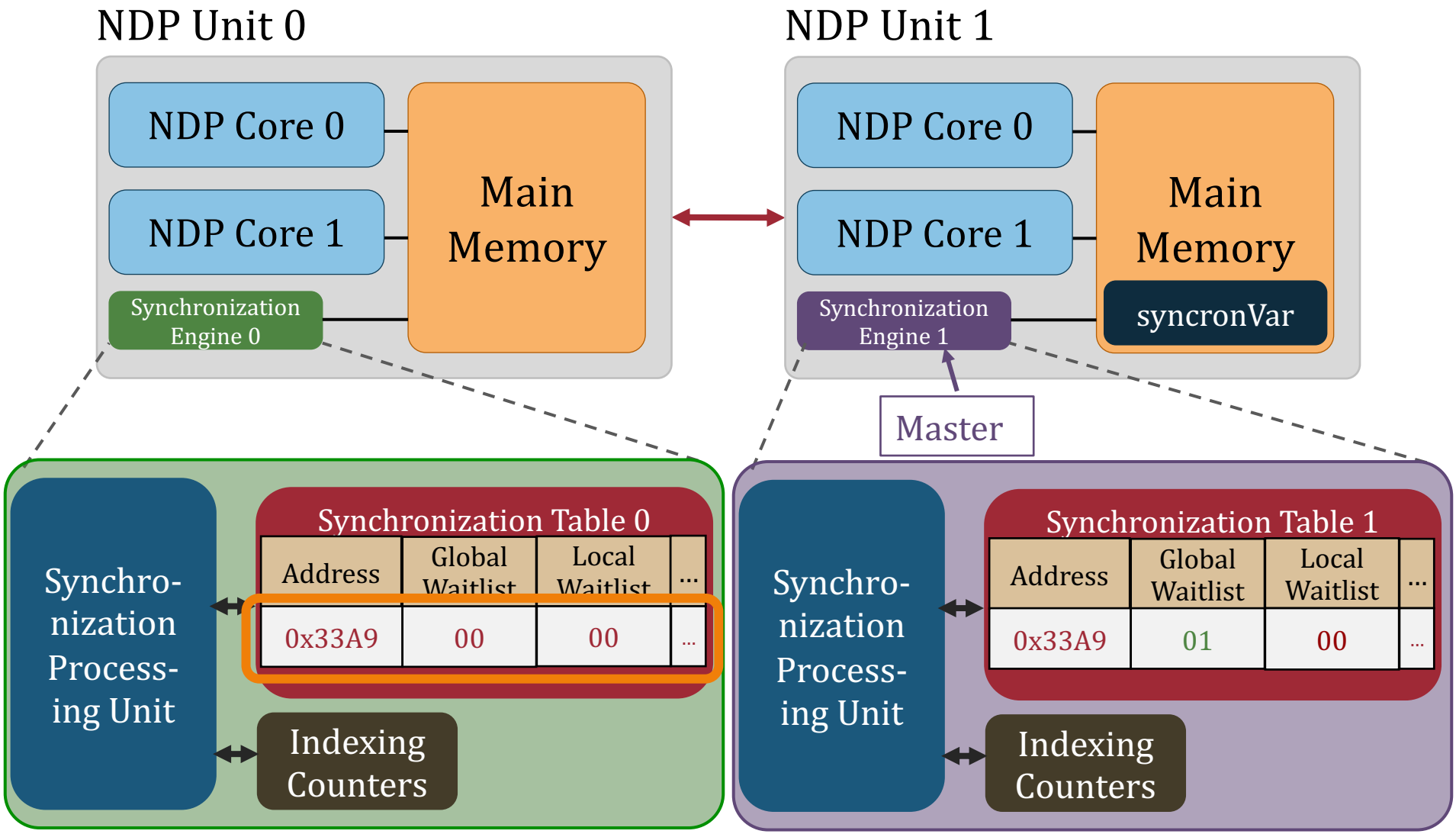
Lock Operation



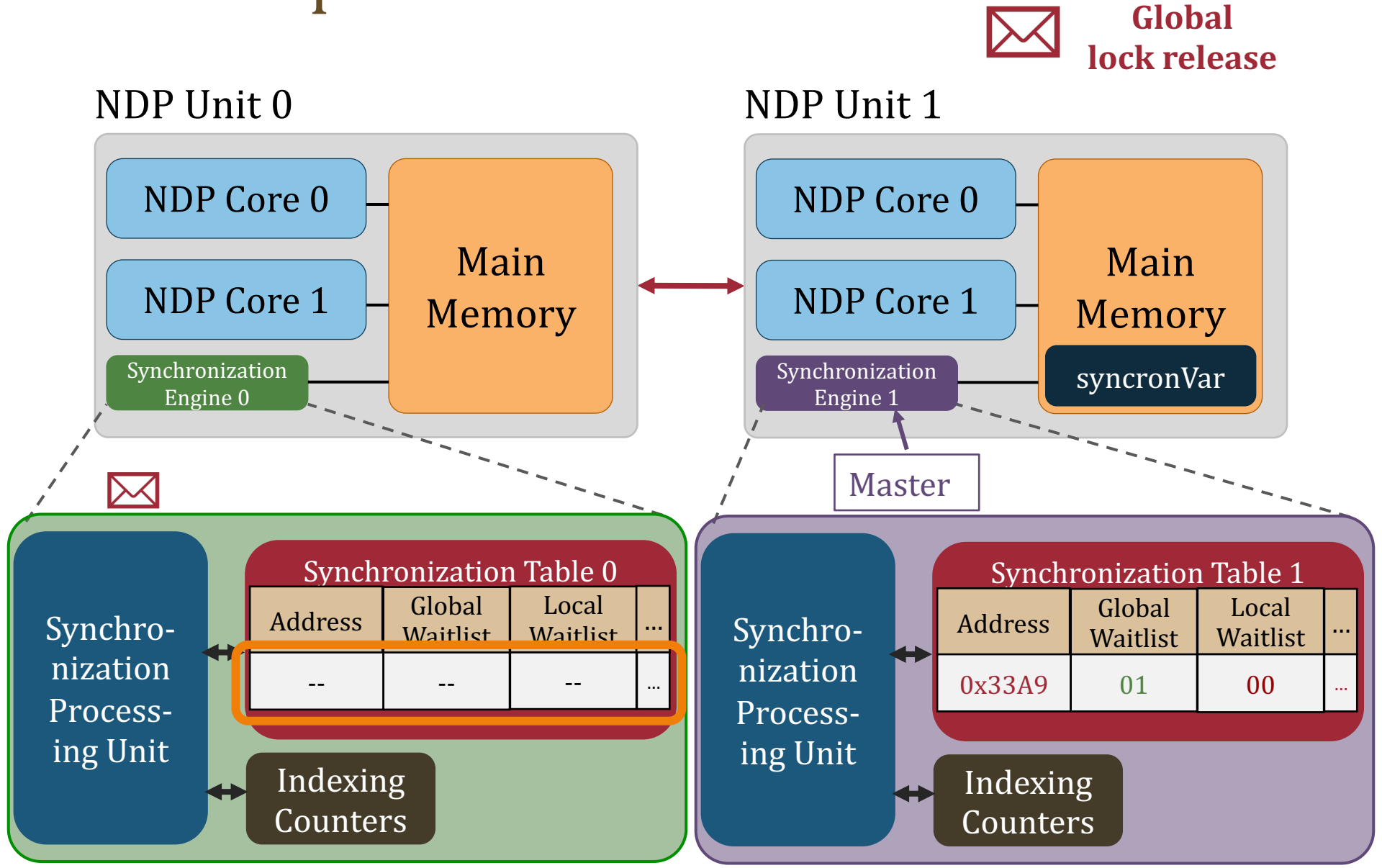
Lock Operation



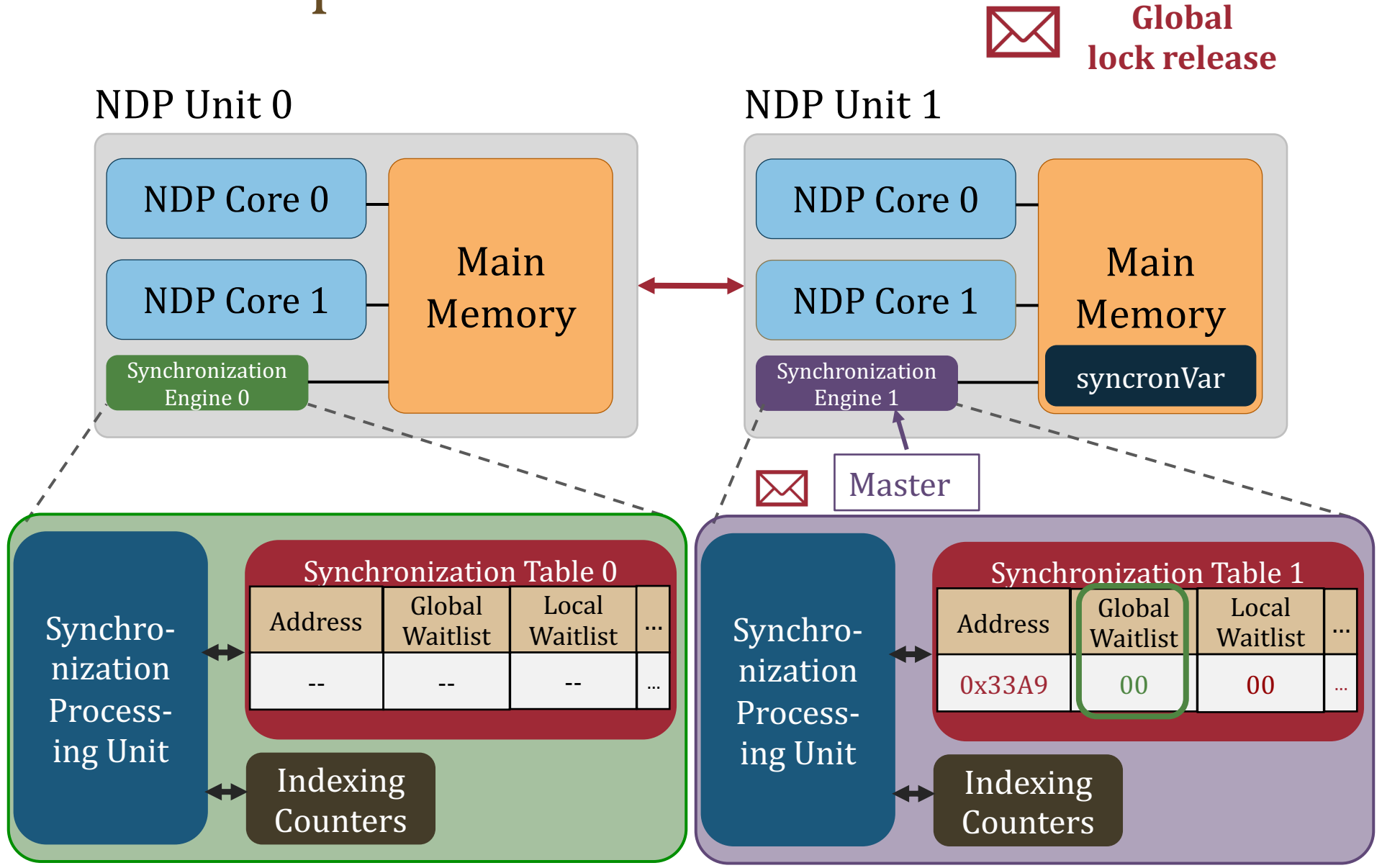
Lock Operation



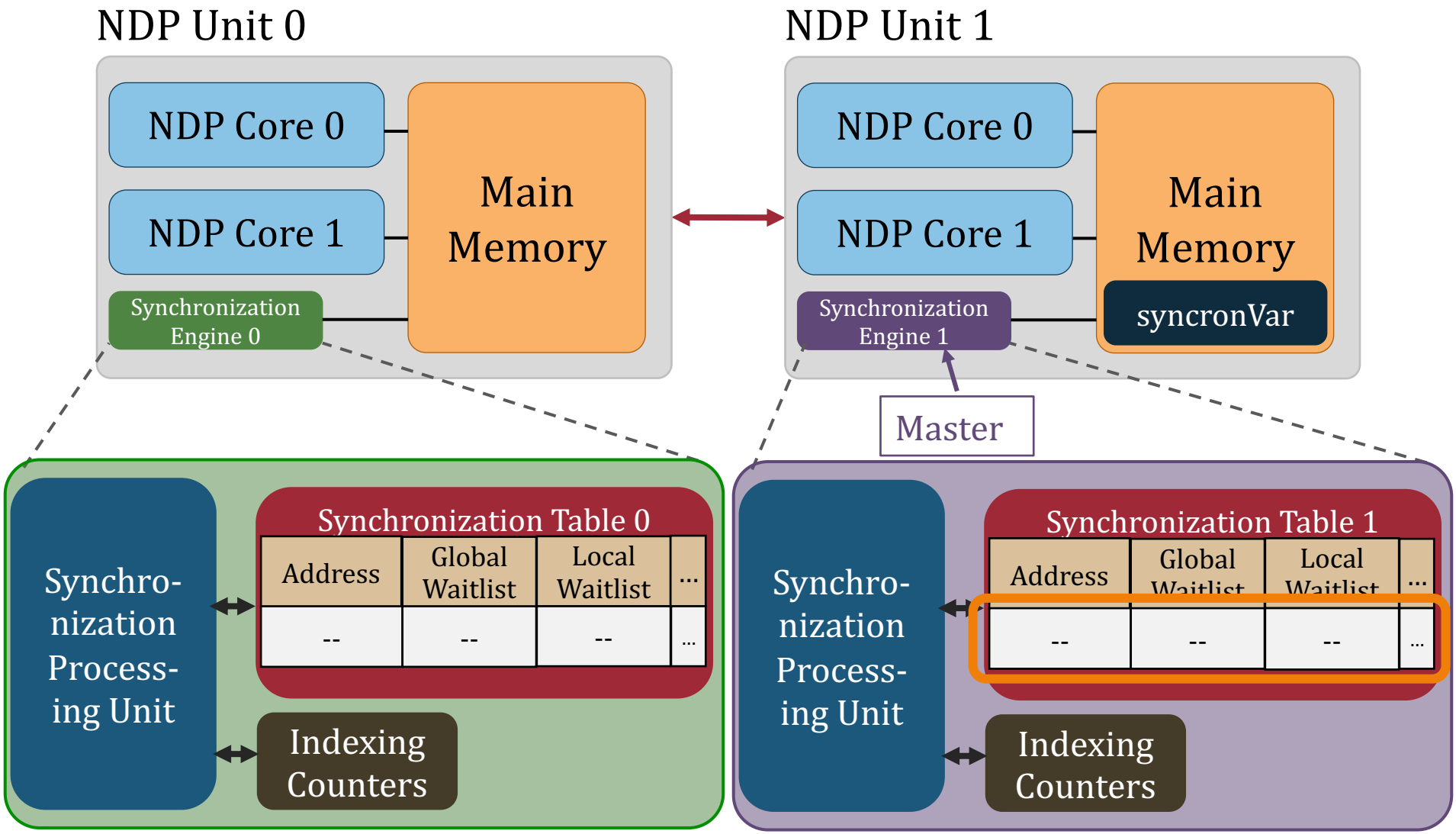
Lock Operation



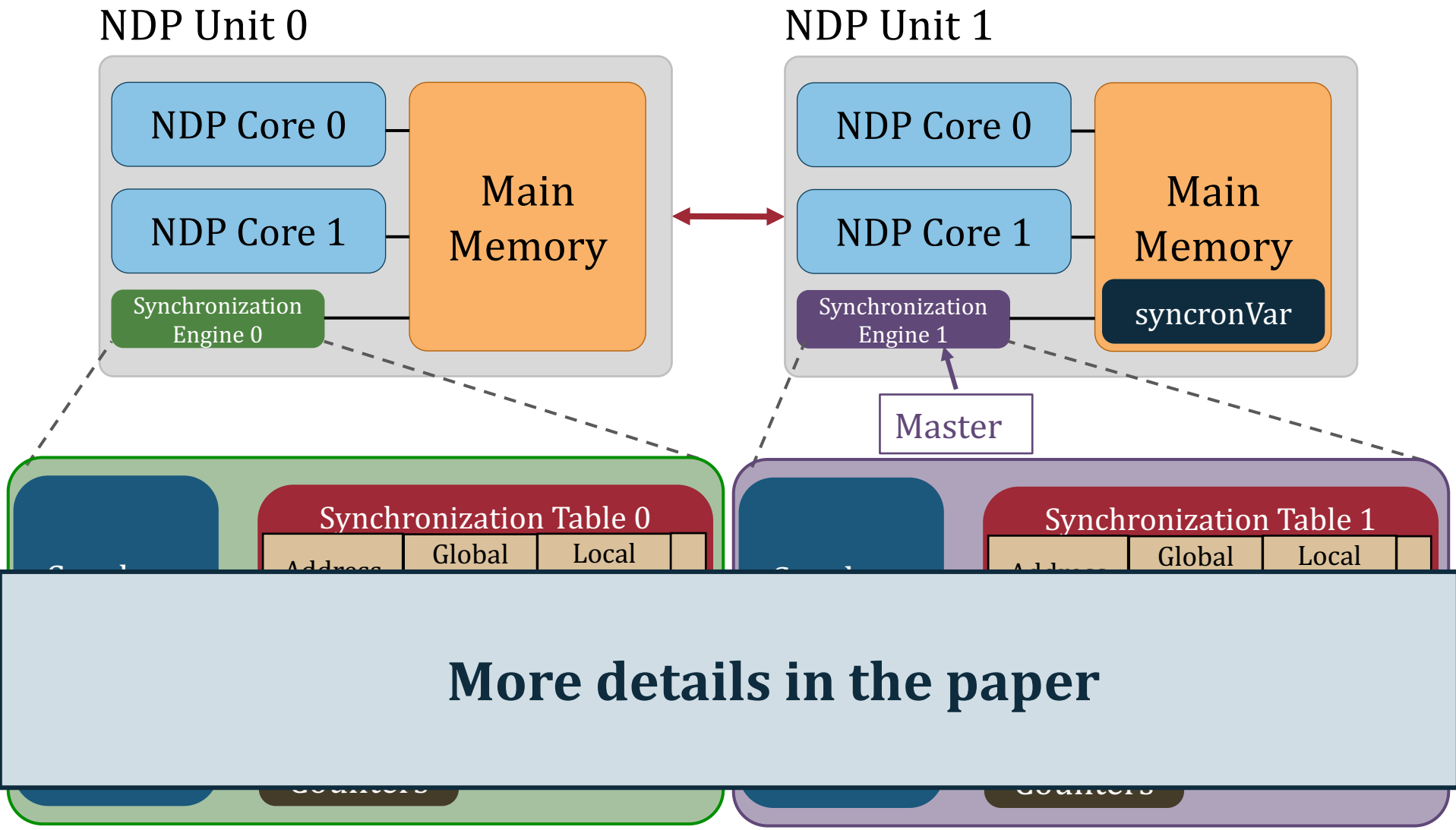
Lock Operation



Lock Operation



Lock Operation



Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation

Evaluation Methodology

- Simulators:
  - **Zsim** [Sanchez+, ISCA'13]
  - **Ramulator** [Kim+, CAL'15]
- System Configuration:
  - **4x NDP units of 16 in-order cores**
  - 16KB L1 Data + Instr. Cache
  - **4GB HBM memory**
- SynCron's Default Parameters:
  - Synchronization Processing Unit @1GHz
  - 12-cycle worst-case latency for a message to be served [Aladdin]
  - 64 entries in Synchronization Table, 1-cycle latency [CACTI]
  - 256 entries in indexing counters 2-cycle latency [CACTI]
- Workloads:
  - 9x **Pointer-chasing** Data Structures from ASCYLIB [David+, ASPLOS'15]
  - 6x **Graph Applications** from Crono [Ahmad+, IISWC'15]
  - **Time Series Analysis** from Matrix Profile [Yeh+, ICDM'16]

Comparison Points for SynCron

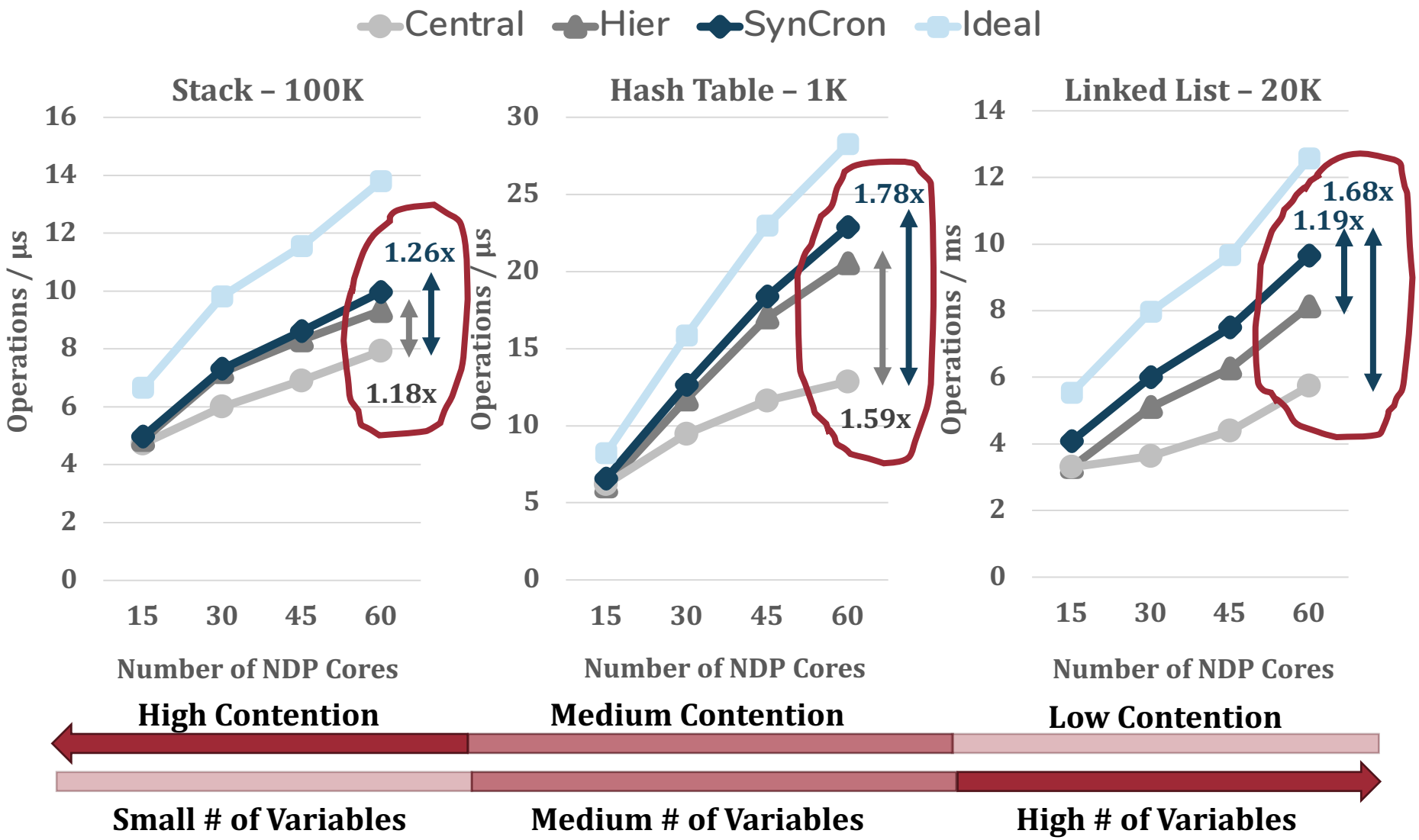
1. **SynCron**

2. **Central** [Ahn+, ISCA'15]:
   - Synchronization Server: One NDP core of the NDP system
   - Centralized hardware message-passing communication

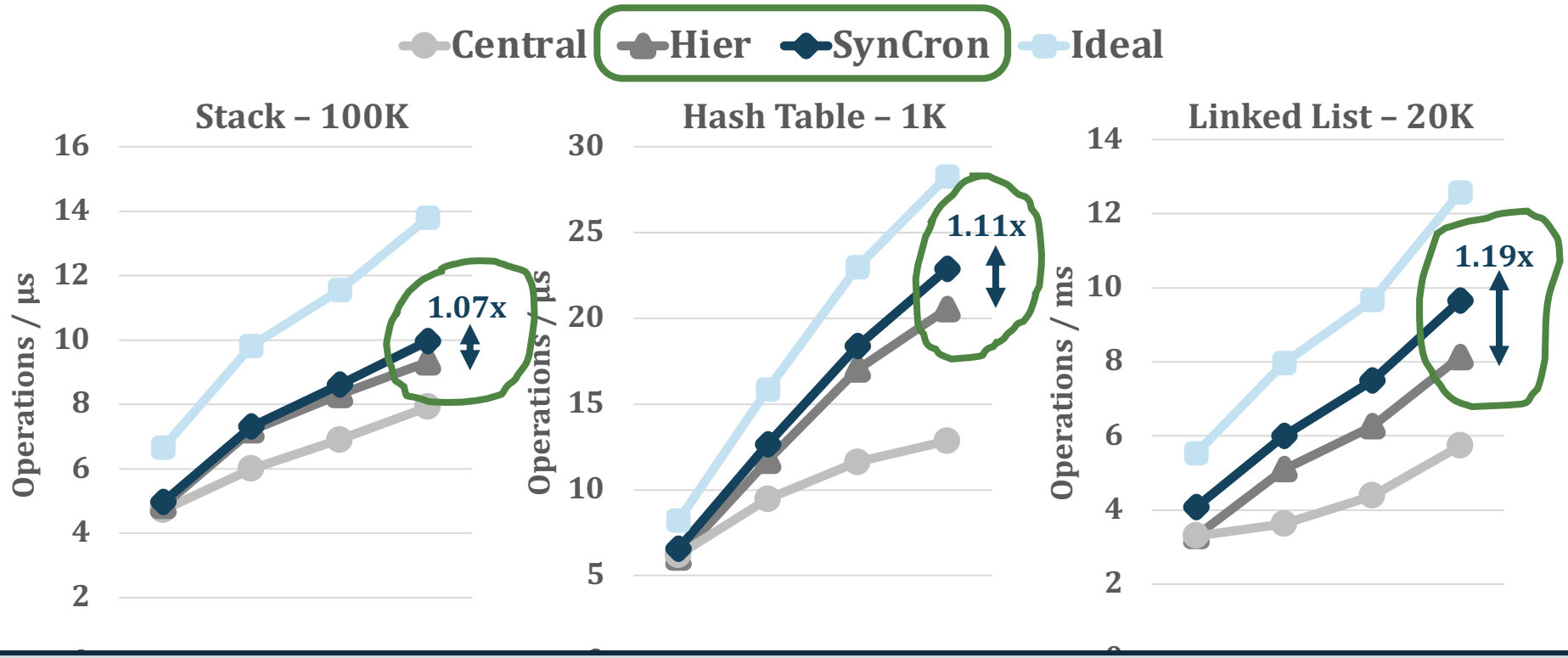
3. **Hier** [Gao+, PACT'15 / Tang+, ASPLOS'19]:
   - Synchronization Servers: One NDP core per NDP unit
   - Hierarchical hardware message-passing communication

4. **Ideal**
   - Zero overhead for synchronization

Throughput of Pointer Chasing

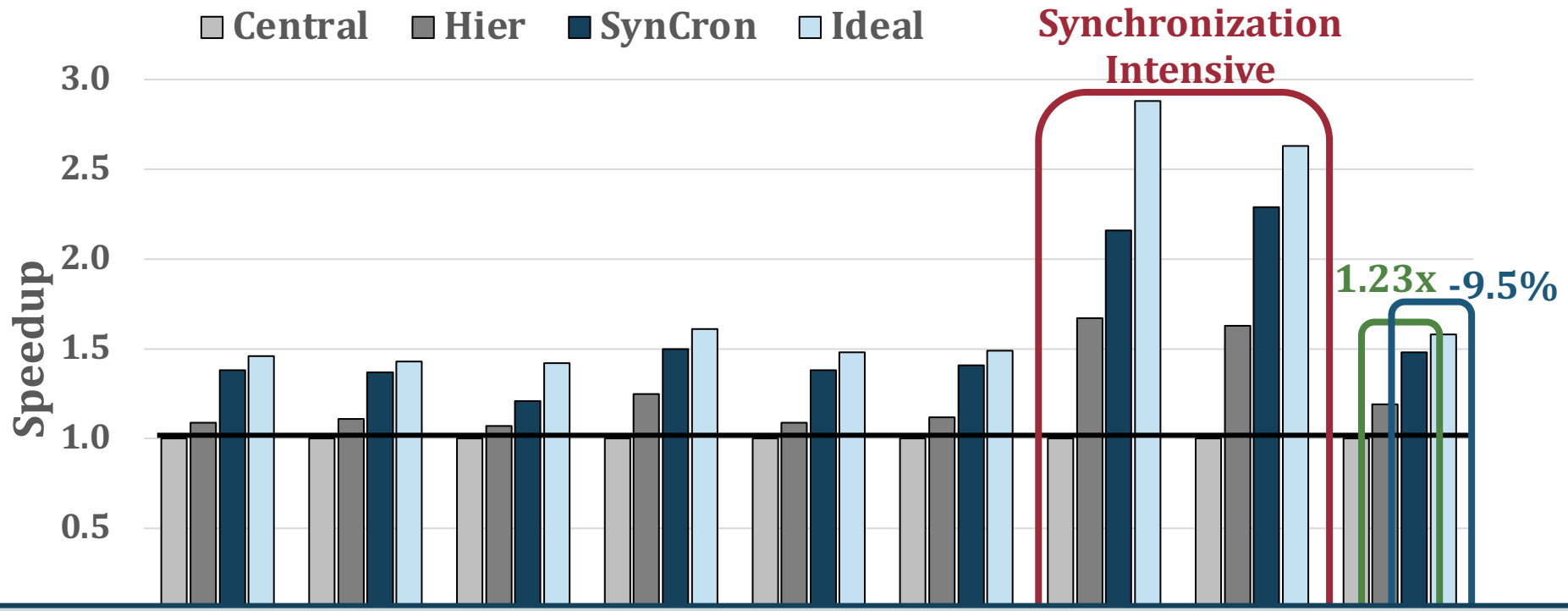


Throughput of Pointer Chasing



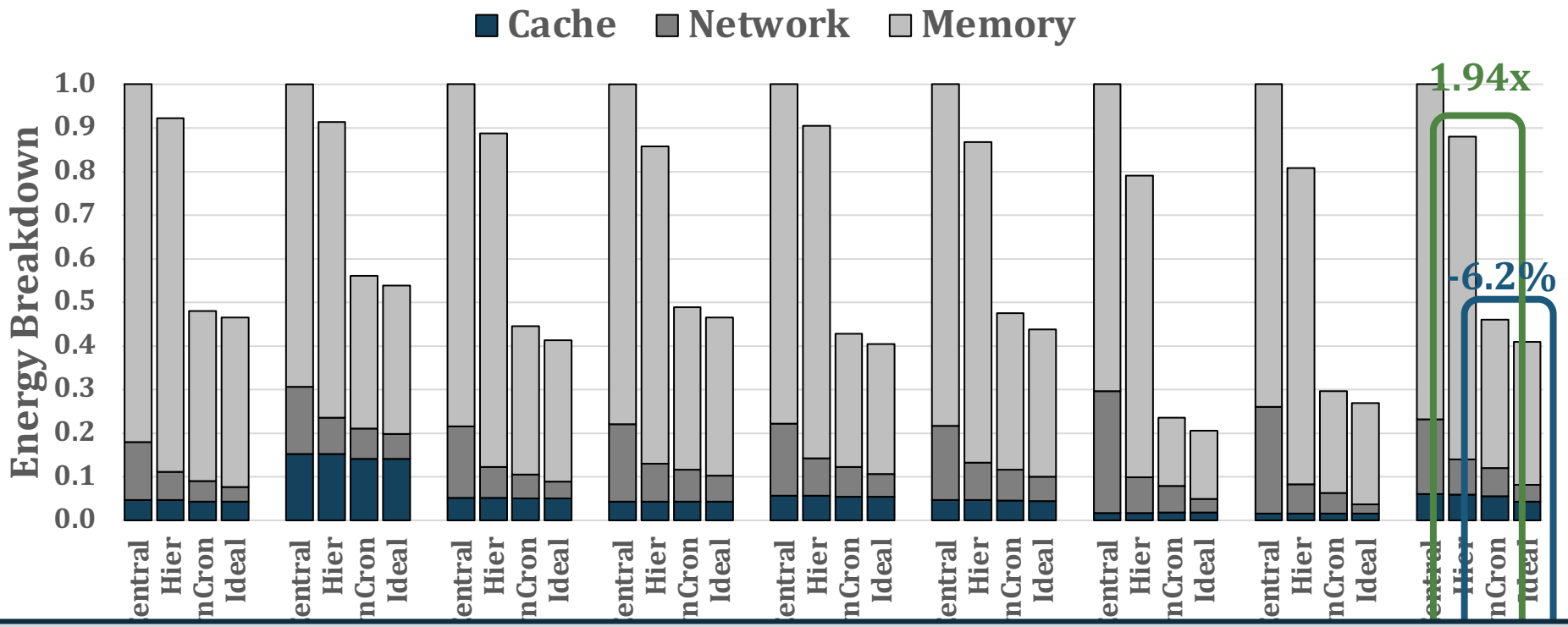
SynCron achieves the highest throughput under all scenarios

Speedup in Real Applications



SynCron performs best across all real applications

System Energy in Real Applications



SynCron reduces system energy significantly

Area and Power Overheads

		Synchronization Engine	ARM Cortex A7
Technology		40nm	28nm
Area	9.78%	Total: 0.0461mm2	Total: 0.45mm2
Power	2.70%	2.7mW	100mW

SynCron has low area and power overheads

Sensitivity Studies

- Different memory technologies (HBM, HMC, DDR4)
- Various data placement techniques
- Various transfer latencies on links across NDP units
- Overflow management cost
- Various sizes for the Synchronization Table

SynCron is effective for a wide variety of configurations

Summary & Conclusion

- Synchronization is a **major system challenge** for NDP systems
- **Prior** schemes are **not suitable** or **efficient** for NDP systems
- **SynCron** is the **first end-to-end** synchronization solution for NDP architectures
- Syncron consists of **four** key techniques:
  i. **Hardware support** for synchronization acceleration
  ii. **Direct buffering** of synchronization variables
  iii. **Hierarchical** message-passing **communication**
  iv. Integrated hardware-only **overflow management**
- SynCron's benefits: **90.5%** and **93.8%** of performance and energy of an **Ideal** zero-overhead scheme
- SynCron is **highly-efficient**, **low-cost**, **easy-to-use**, and **general** to support many synchronization primitives

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula
christina.giann@gmail.com

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI

ETH zürich

National Technical University of Athens
CSLab

UNIVERSITY OF TORONTO

UNIVERSIDAD DE MÁLAGA

Backup Slides

Baseline NDP Architecture

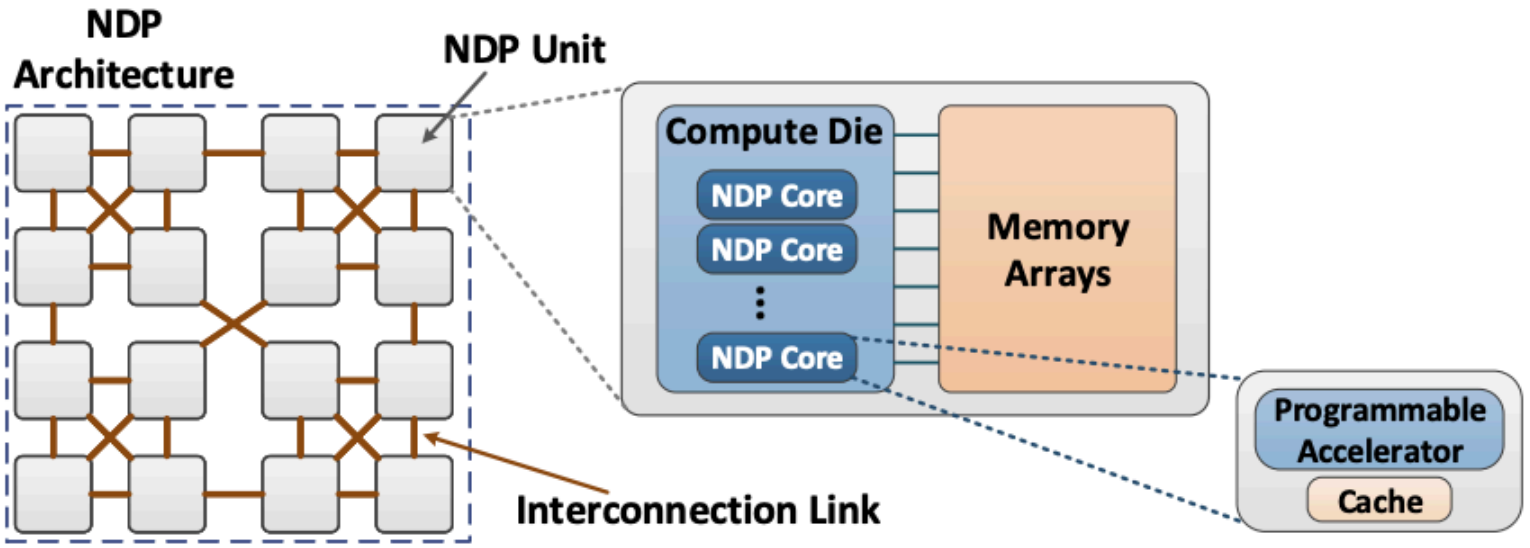


Figure 1: High-level organization of an NDP architecture.

Low Scalability of Coherence-based Synchronization in Real System

Million Operations per Second	1 thread single-socket	14 threads single-socket	2 threads same-socket	2 threads different-socket
TTAS lock [122]	8.92	2.28	9.91	4.32
Hierarchical Ticket lock [103]	8.06	2.91	9.01	6.79

Table 1: Throughput of two coherence-based lock algorithms on an Intel Xeon Gold server using the libslock library [30].

Low Scalability of Coherence-based Synchronization in NDP Simulated System

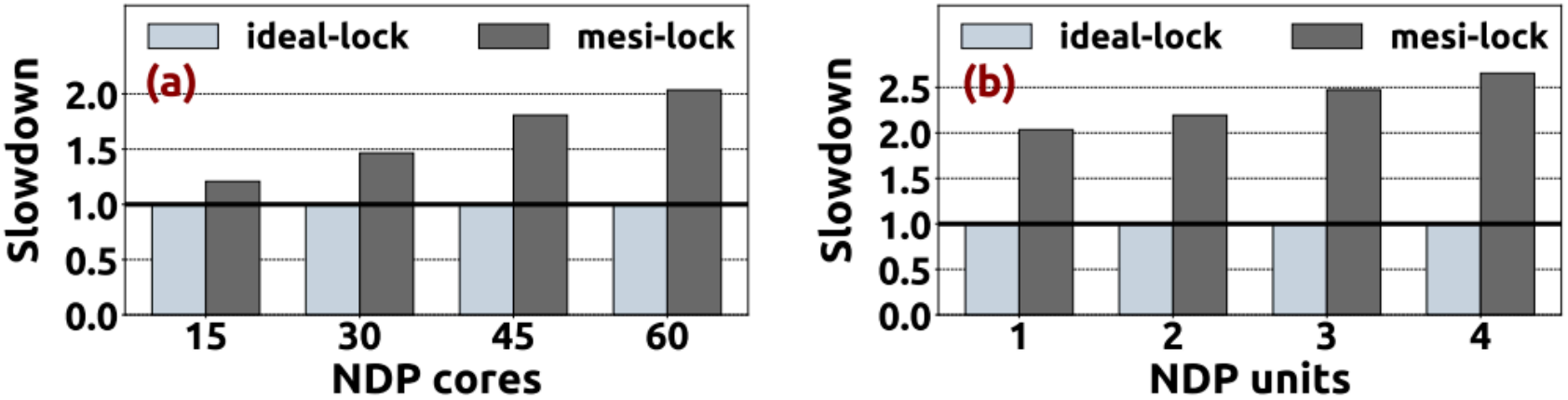


Figure 2: Slowdown of a stack data structure using a coherence-based lock over using an *ideal* zero-cost lock, when varying (a) the NDP cores within a single NDP unit and (b) the number of NDP units while keeping core count constant at 60.

SynCron's Overview

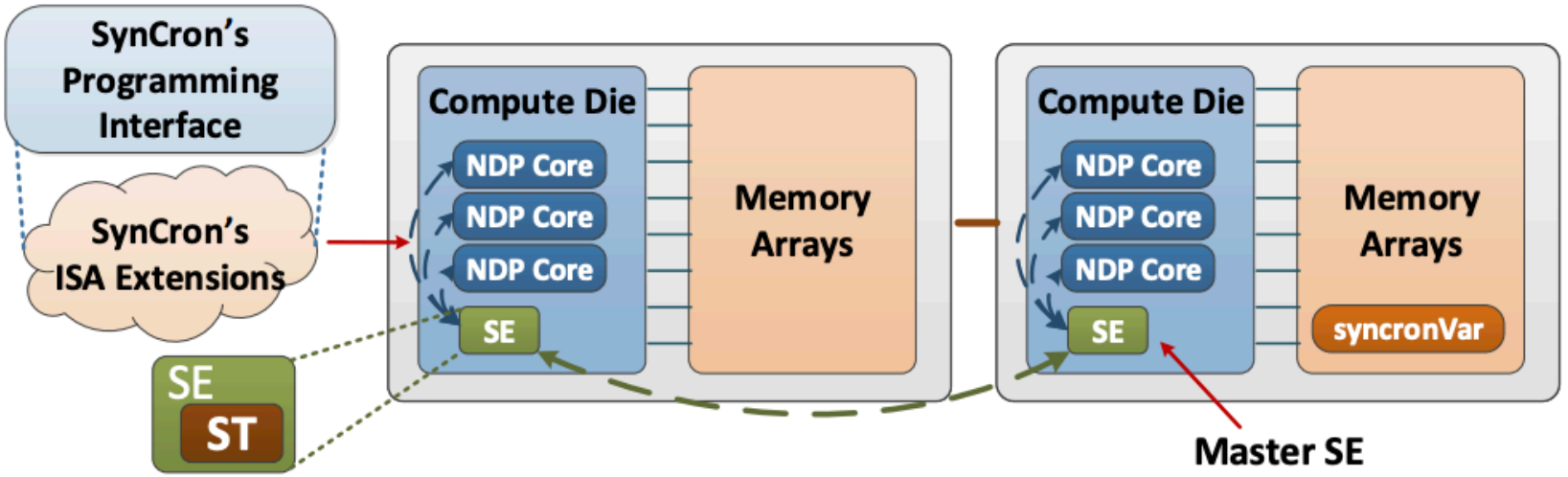


Figure 3: High-level overview of *SynCron*.

Lock Operation using SynCron

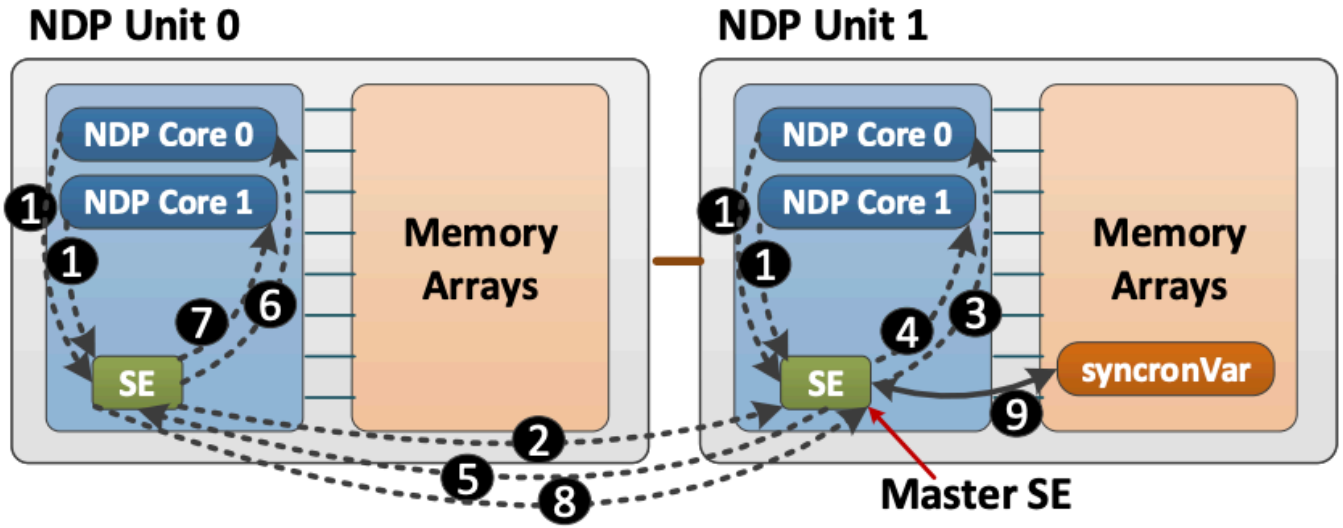


Figure 4: An example execution scenario for a lock requested by *all* NDP cores.

SynCron's API

SynCron Programming Interface
syncronVar *create_syncvar ();
void destroy_syncvar (syncronVar *svar);
void lock_acquire (syncronVar *lock);
void lock_release (syncronVar *lock);
void barrier_wait_within_unit (syncronVar *bar, int initialCores);
void barrier_wait_across_units (syncronVar *bar, int initialCores);
void sem_wait (syncronVar *sem, int initialResources);
void sem_post (syncronVar *sem);
void cond_wait (syncronVar *cond, syncronVar *lock);
void cond_signal (syncronVar *cond);
void cond_broadcast (syncronVar *cond);

Table 2: *SynCron*'s Programming Interface (i.e., API).

Message Encoding of SynCron

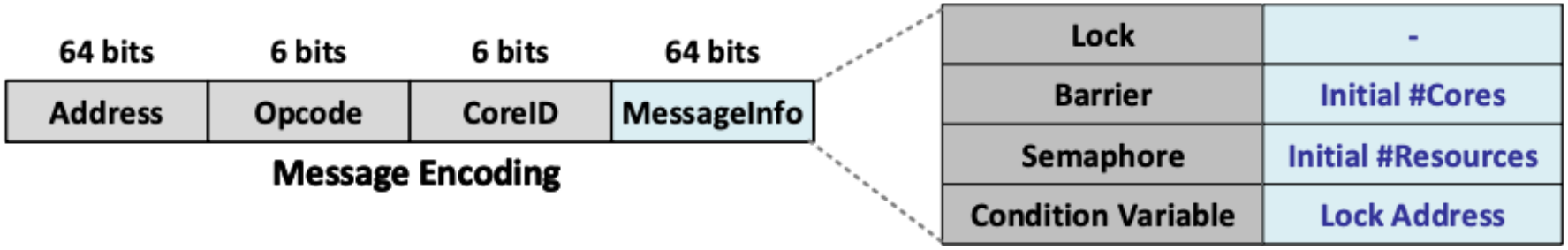


Figure 5: Message encoding of *SynCron*.

Message Opcodes for All Primitives

Primitives	*SynCron* Message Opcodes
Locks	lock_acquire_global, lock_acquire_local, lock_release_global lock_release_local, lock_grant_global, lock_grant_local lock_acquire_overflow, lock_release_overflow, lock_grant_overflow
Barriers	barrier_wait_global, barrier_wait_local_within_unit barrier_wait_local_across_units, barrier_depart_global, barrier_depart_local barrier_wait_overflow, barrier_departure_overflow
Semaphores	sem_wait_global, sem_wait_local, sem_grant_global sem_grant_local, sem_post_global, sem_post_local sem_wait_overflow, sem_grant_overflow, sem_post_overflow
Condition Variables	cond_wait_global, cond_wait_local, cond_signal_global cond_signal_local, cond_broad_global, cond_broad_local cond_grant_global, cond_grant_local, cond_wait_overflow cond_signal_overflow, cond_broad_overflow, cond_grant_overflow
Other	decrease_indexing_counter

Table 3: Message opcodes of *SynCron*.

The Synchronization Engine

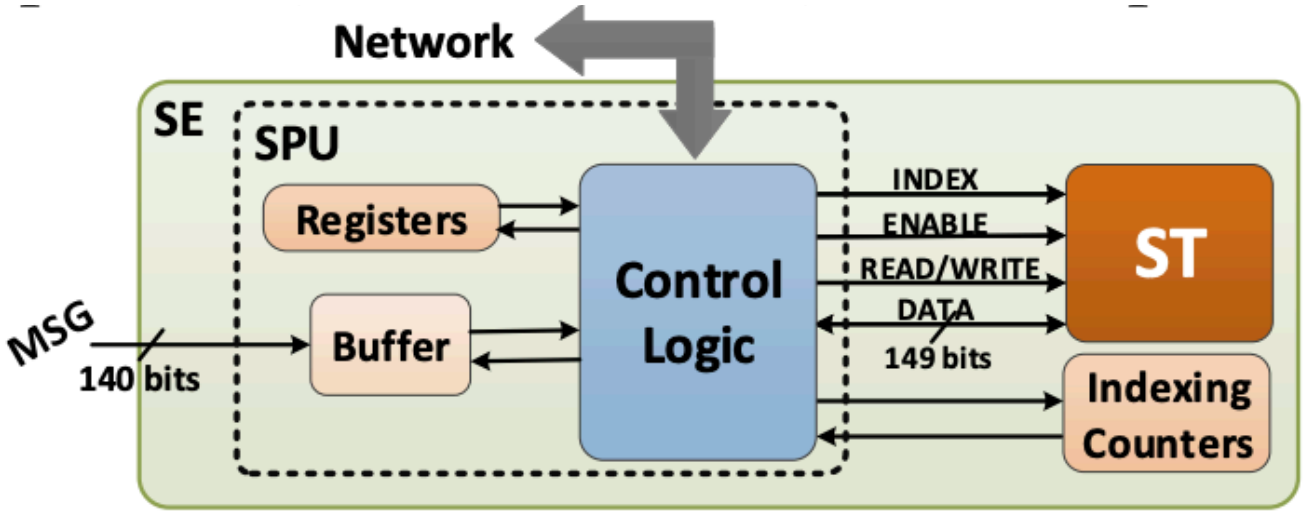


Figure 6: The Synchronization Engine (SE).

Synchronization Table Entry

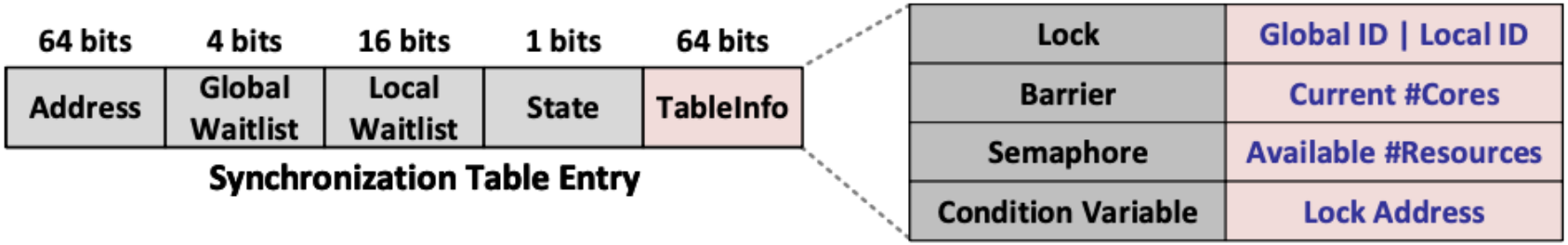


Figure 7: Synchronization Table (ST) entry.

Control Flow of Message in Synchronization Engine

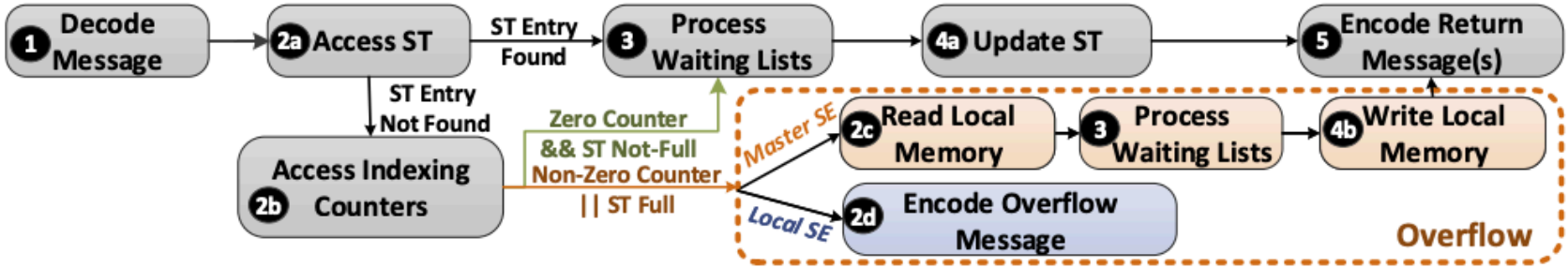


Figure 8: Control flow in SE.

Synchronization Variable of SynCron

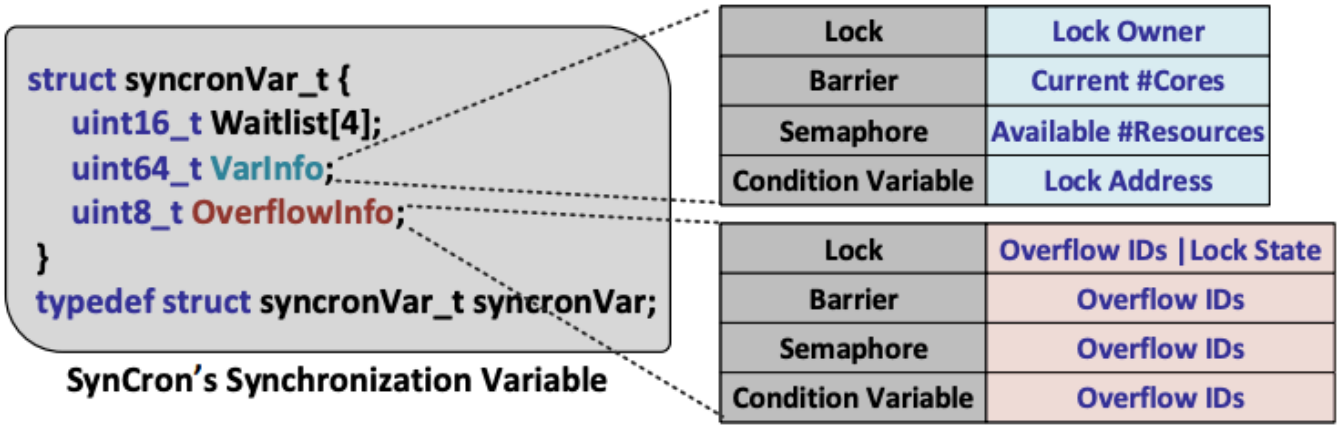


Figure 9: Synchronization variable of *SynCron* (*syncronVar*).

Qualitative Comparison with Closely Related Works

	SSB [157]	LCU [146]	MiSAR [97]	**SynCron**
Supported Primitives	1	1	3	**4**
ISA Extensions	2	2	7	**2**
Spin-Wait Approach	yes	yes	no	**no**
Direct Notification	no	yes	yes	**yes**
Target System	uniform	uniform	uniform	**non-uniform**
Overflow Management	partially integrated	partially integrated	handled by programmer	**fully integrated**

Table 4: Comparison of *SynCron* with prior mechanisms.

Simulated System

NDP Cores	16 in-order cores @2.5 GHz per NDP unit
L1 Data + Inst. Cache	private, 16KB, 2-way, 4-cycle; 64 B line; 23/47 pJ per hit/miss [109]
NDP Unit Local Network	buffered crossbar network with packet flow control; 1-cycle arbiter; 1-cycle per hop [6]; 0.4 pJ/bit per hop [149]; M/D/1 model [18] for queueing latency;
DRAM HBM	4 stacks; 4GB HBM 1.0 [92,93]; 500MHz with 8 channels; nRCDR/nRCDW/nRAS/nWR 7/6/17/8 ns [47,85]; 7 pJ/bit [151]
DRAM HMC	4 stacks; 4GB HMC 2.1; 1250MHz; 32 vaults per stack; nRCD/nRAS/nWR 17/34/19 ns [47,85]
DRAM DDR4	4 DIMMs; 4GB each DIMM DDR4 2400MHz; nRCD/nRAS/nWR 16/39/18 ns [47,85]
Interconnection Links Across NDP Units	12.8GB/s per direction; 40 ns per cache line; 20-cycle; 4 pJ/bit
Synchronization Engine	SPU @1GHz clock frequency [129]; 8× 64-bit registers; buffer: 280B; ST: 1192B, 64 entries, 1-cycle [109]; indexing counters: 2304B, 256 entries (8 LSB of the address), 2-cycle [109]

Table 5: Configuration of our simulated system.

Evaluated Workloads

Data Structure	Configuration
Stack [31]	100K - 100% push
Queue [31, 104]	100K - 100% pop
Array Map [31, 56]	10 - 100% lookup
Priority Queue [11, 31, 118]	20K - 100% deleteMin
Skip List [31, 118]	5K - 100% deletion
Hash Table [31, 63]	1K - 100% lookup
Linked List [31, 63]	20K - 100% lookup
Binary Search Tree Fine-Grained (BST_FG) [130]	20K - 100% lookup
Binary Search Tree Drachsler (BST_Drachsler) [31, 37]	10K - 100% deletion

Real Application	Locks	Barriers
Breadth First Search (**bfs**) [7]	✓	✓
Connected Components (**cc**) [7]	✓	✓
Single Source Shortest Paths (**sssp**) [7]	✓	✓
Pagerank (**pr**) [7]	✓	✓
Teenage Followers (**tf**) [65]	✓	-
Triangle Counting (**tc**) [7]	✓	✓
Time Series Analysis (**ts**) [152]	✓	✓

Real Application	Input Data Set
bfs, cc, sssp, pr, tf, tc	wikipedia -20051105 (**wk**)
	soc-LiveJournal1 (**sl**)
	sx-stackoverflow (**sx**)
	com-Orkut (**co**)
ts	air quality (**air**)
	energy consumption (**pow**)

Table 6: Summary of all workloads used in our evaluation.

Speedup in Simple Microbenchmarks

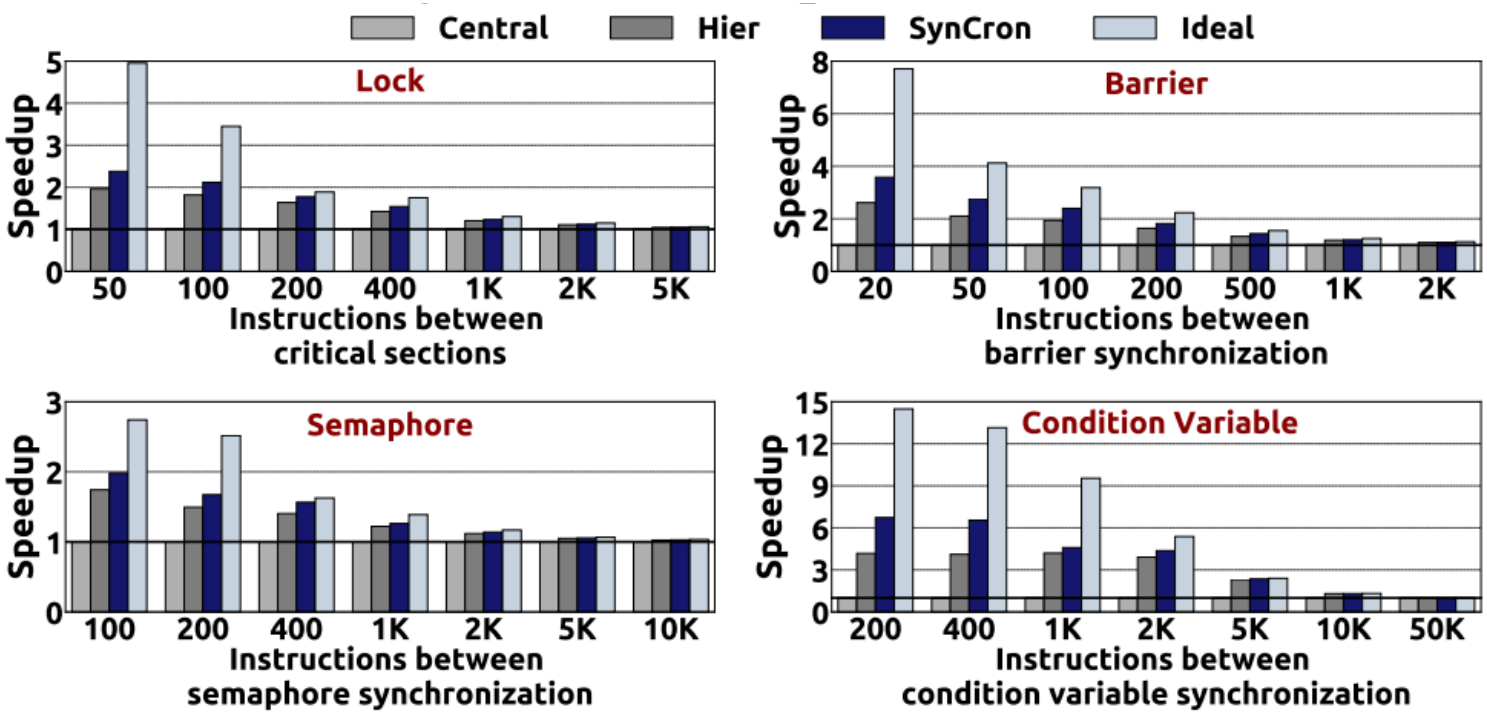


Figure 10: Speedup of different synchronization primitives.

Throughput of Pointer Chasing

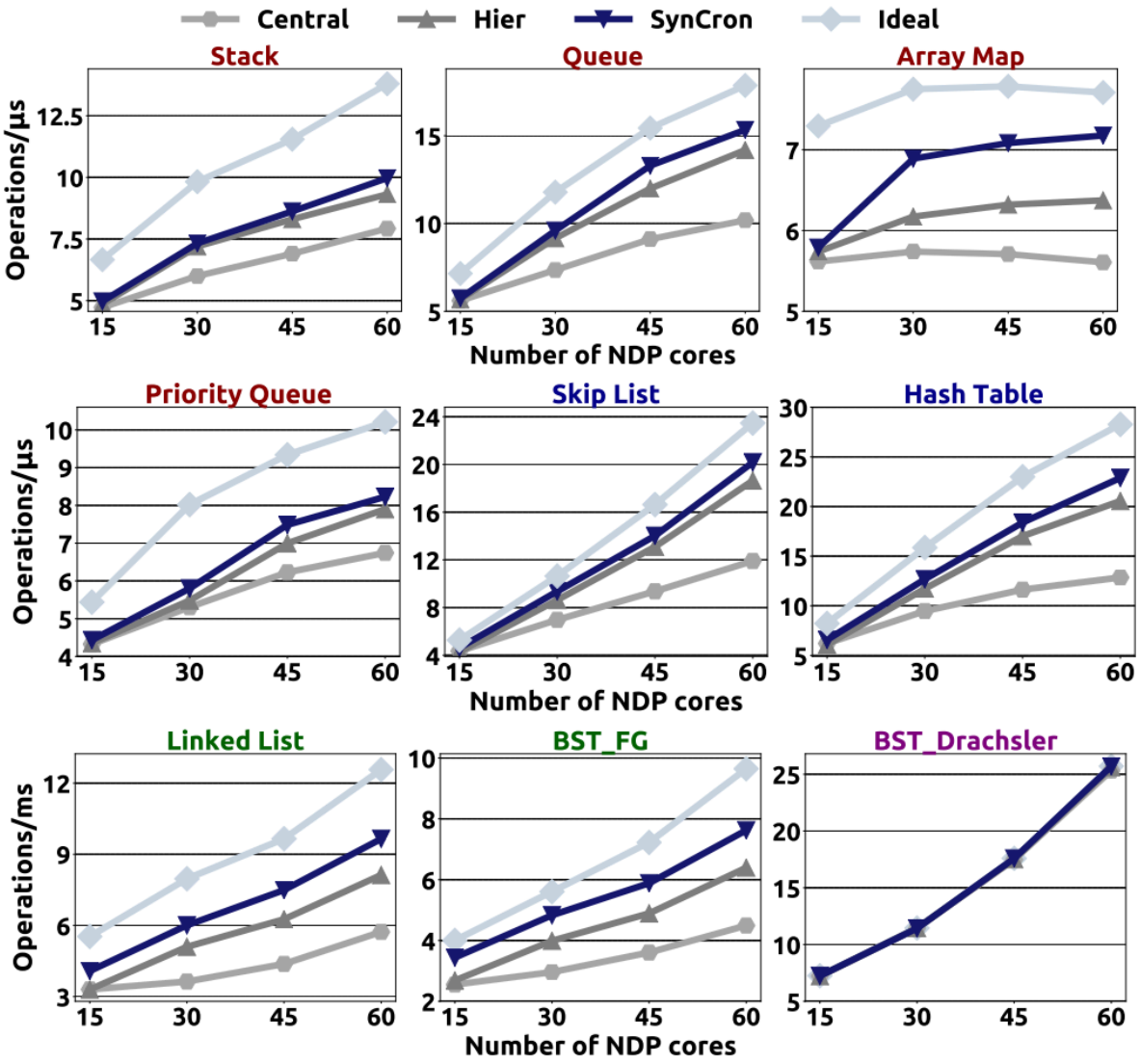


Figure 11: Throughput of pointer chasing using data structures.

Speedup in Real Applications

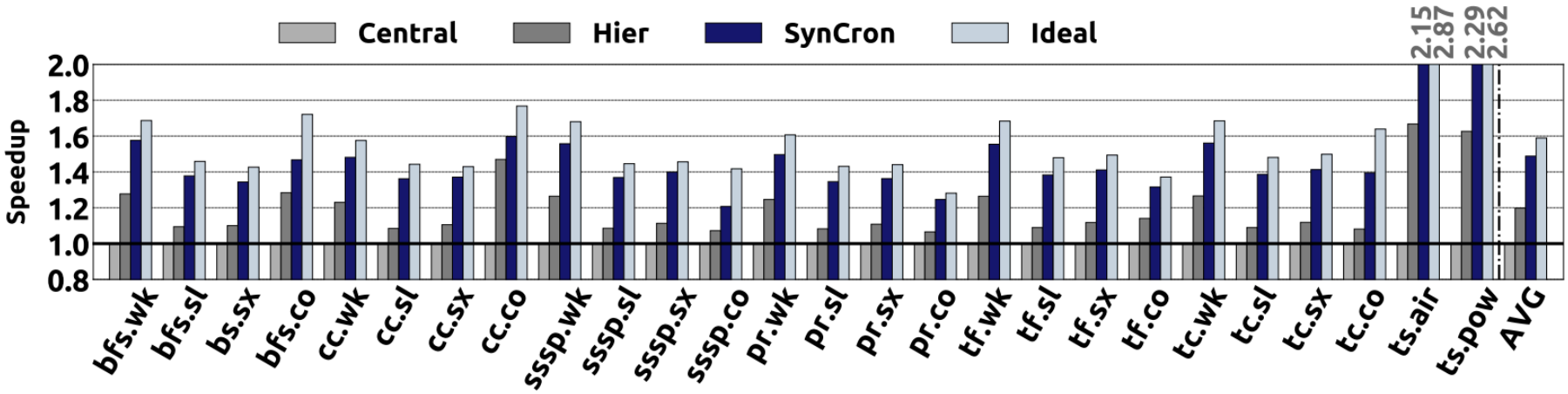


Figure 12: Speedup in real applications normalized to *Central*.

Scalability in Real Applications using SynCron

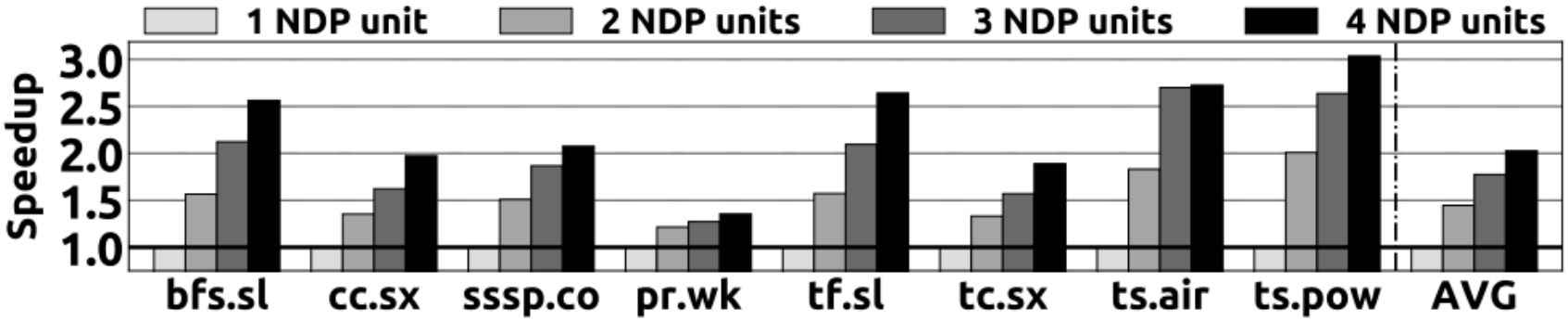


Figure 13: Scalability of real applications using *SynCron*.

System Energy in Real Applications

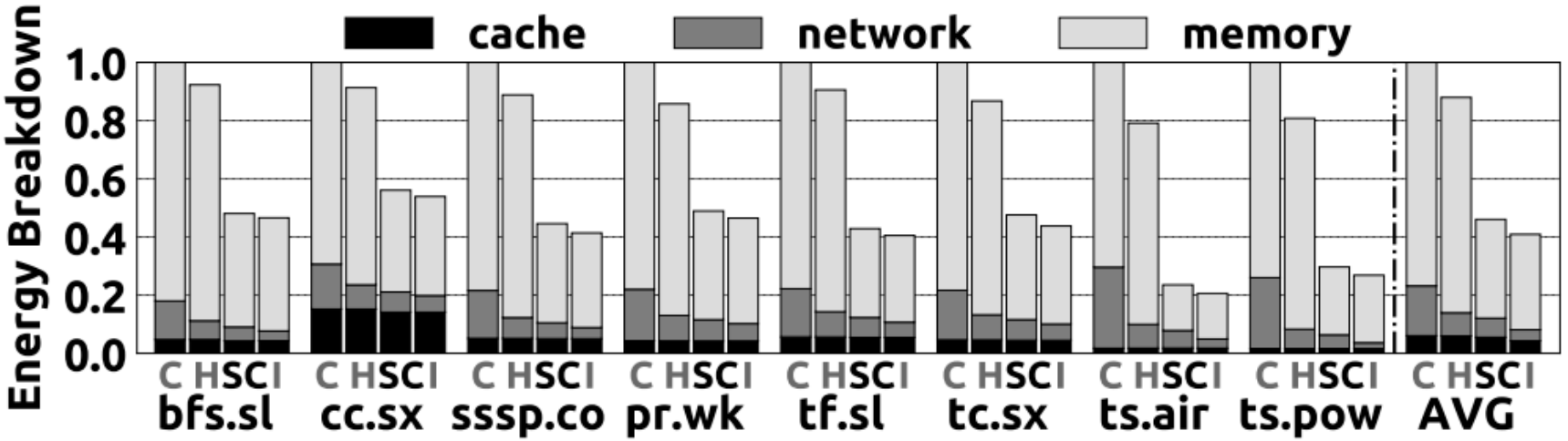


Figure 14: Energy breakdown in real applications for C: *Central*, H: *Hier*, SC: *SynCron* and I: *Ideal*.

Data Movement in Real Applications

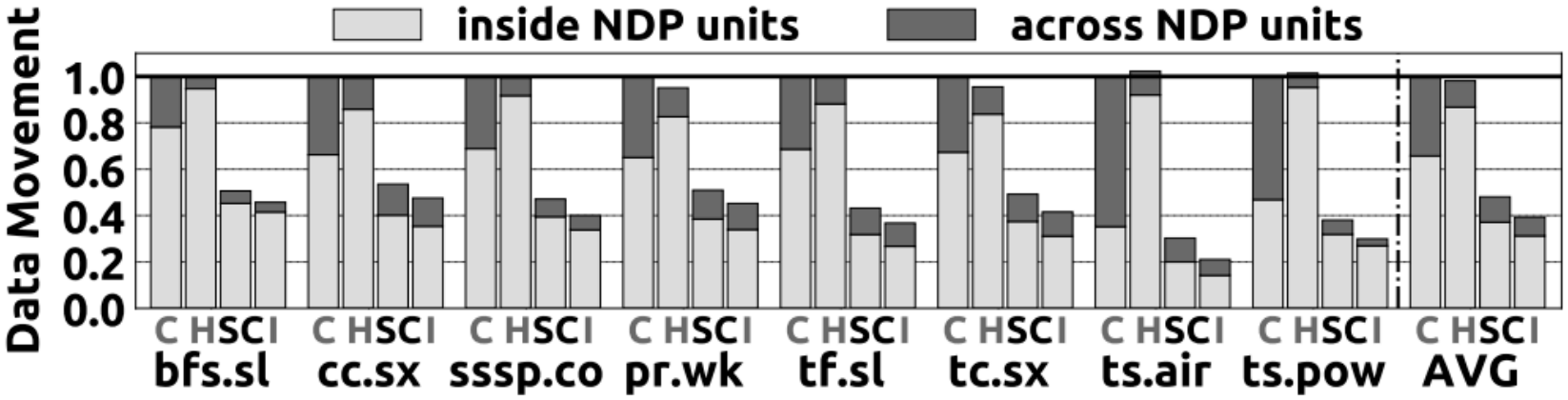


Figure 15: Data movement in real applications for C: *Central*, H: *Hier*, SC: *SynCron* and I: *Ideal*.

Various Transfer Latencies on Links Across NDP Units – High Contention

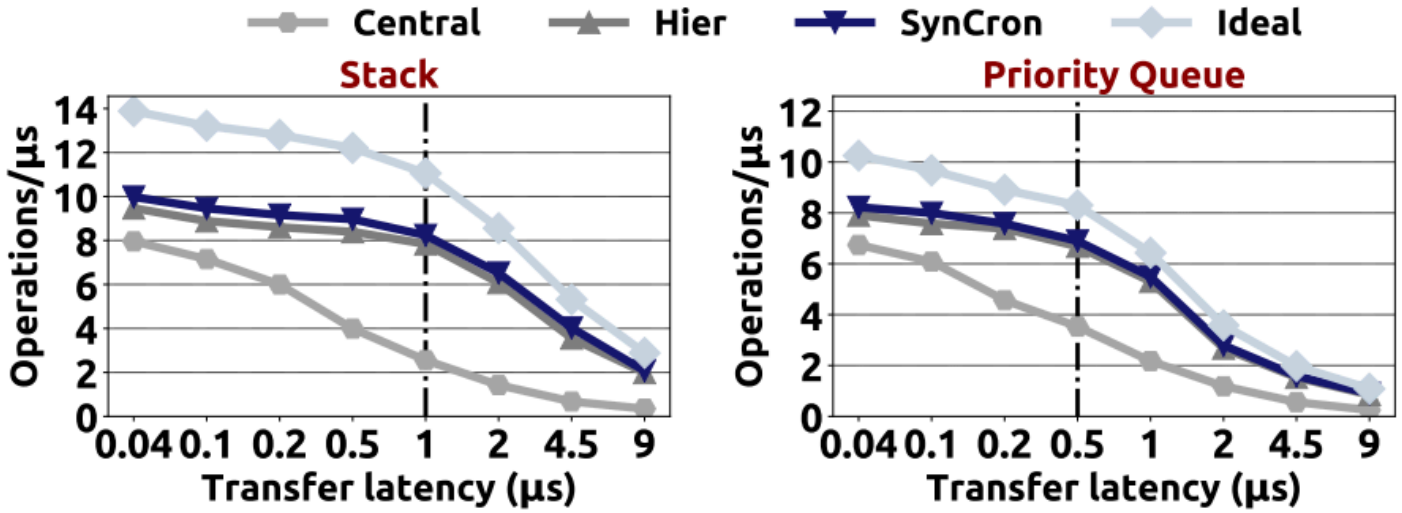


Figure 16: Performance sensitivity to the transfer latency of the interconnection links used to connect the NDP units.

Various Transfer Latencies on Links Across NDP Units – Low Contention

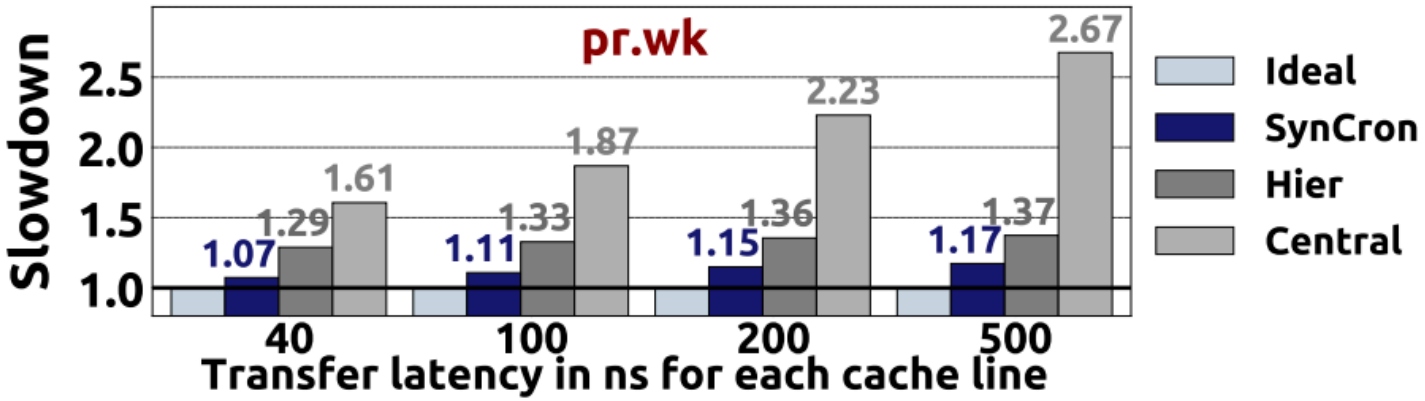


Figure 17: Performance sensitivity to the transfer latency of the interconnection links used to connect the NDP units. All data is normalized to *Ideal* (*lower is better*).

Speedup using Different Memory Technologies

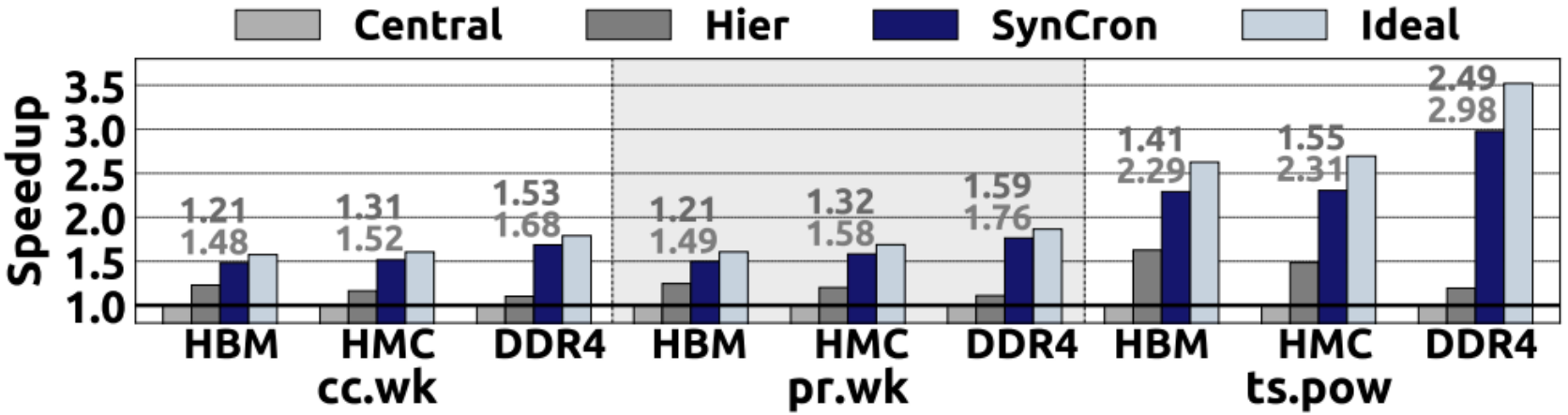
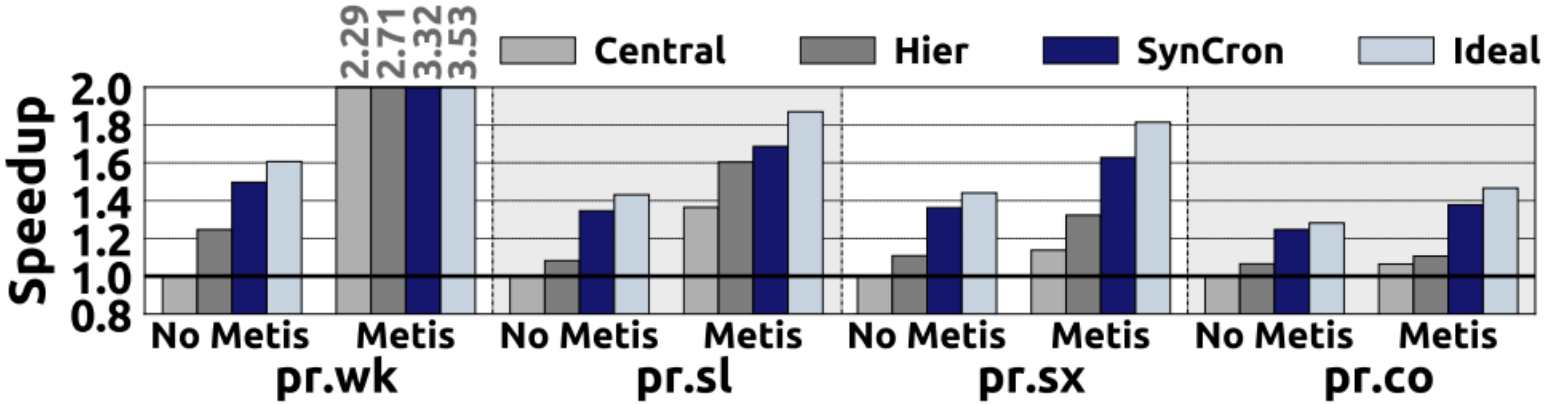


Figure 18: Speedup with different memory technologies.

Speedup in Real Applications using two Different Data Placement Techniques



Max ST Occupancy (%)	pr.wk	pr.sl	pr.sx	pr.co
No Metis	62	51	53	48
Metis	39	29	38	34

Figure 19: Performance sensitivity to a better graph partitioning and maximum ST occupancy of *SynCron*.

Speedup of Syncron over its Flat Variant in a Low-contention and Synchronization Non-intensive Scenario

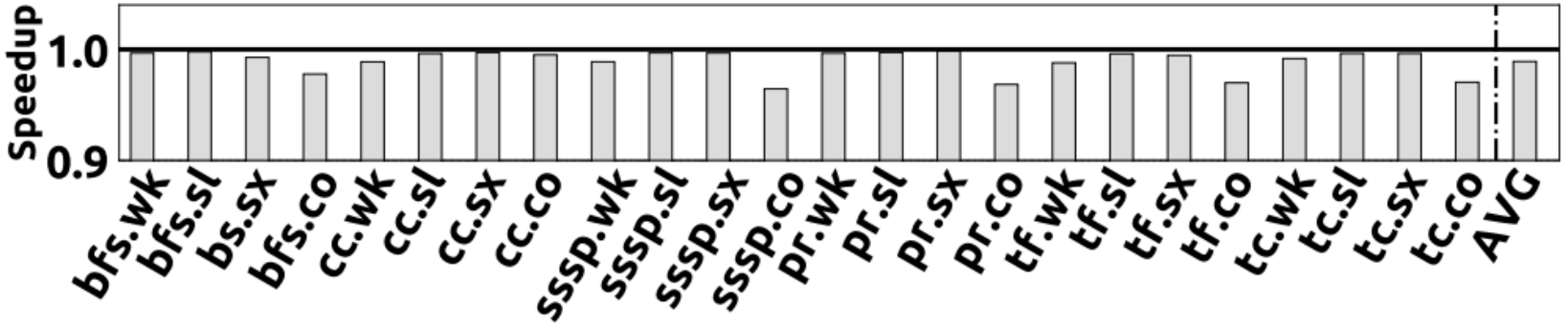


Figure 20: Speedup of *SynCron* normalized to *flat* with 40 ns link latency between NDP units, under a low-contention and synchronization non-intensive scenario.

Speedup of Syncron over its Flat Variant in a Low-contention and Synchronization Intensive Scenario and in a High-contention Scenario

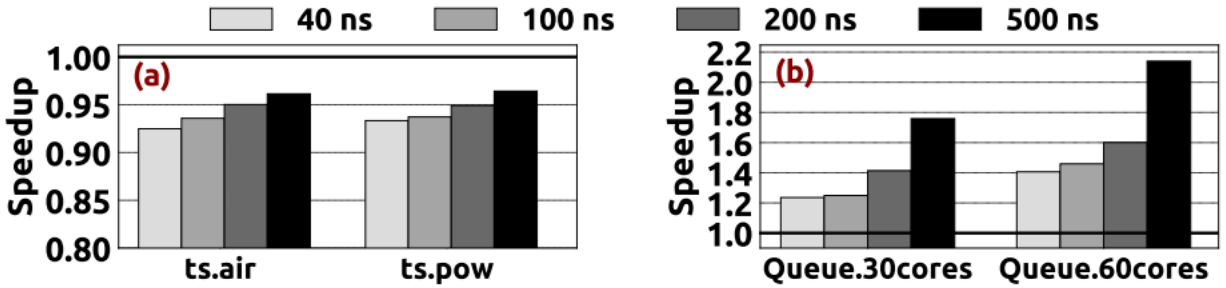


Figure 21: Speedup of *SynCron* normalized to *flat*, as we vary the transfer latency of the interconnection links used to connect NDP units, under (a) a low-contention and synchronization-intensive scenario using 4 NDP units, and (b) a high-contention scenario using 2 and 4 NDP units.

Synchronization Table Occupancy in Real Applications

ST Occupancy	Max (%)	Avg (%)
bfs.wk	51	1.33
bfs.sl	59	1.49
bfs.sx	51	3.24
bfs.co	55	6.09
cc.wk	63	1.27
cc.sl	61	2.16
cc.sx	48	2.43
cc.co	46	4.53
sssp.wk	62	1.18
sssp.sl	54	2.08
sssp.sx	50	2.20
sssp.co	48	5.23
pr.wk	62	4.27

ST Occupancy	Max (%)	Avg (%)
pr.sl	51	2.27
pr.sx	53	2.46
pr.co	48	4.72
tf.wk	62	1.44
tf.sl	53	2.21
tf.sx	50	2.99
tf.co	48	4.61
tc.wk	62	1.26
tc.sl	48	2.08
tc.sx	50	2.77
tc.co	51	4.52
ts.air	84	44.20
ts.pow	89	43.51

Table 7: ST occupancy in real applications.

Slowdown in Real Applications when Varying the Synchronization Table Size

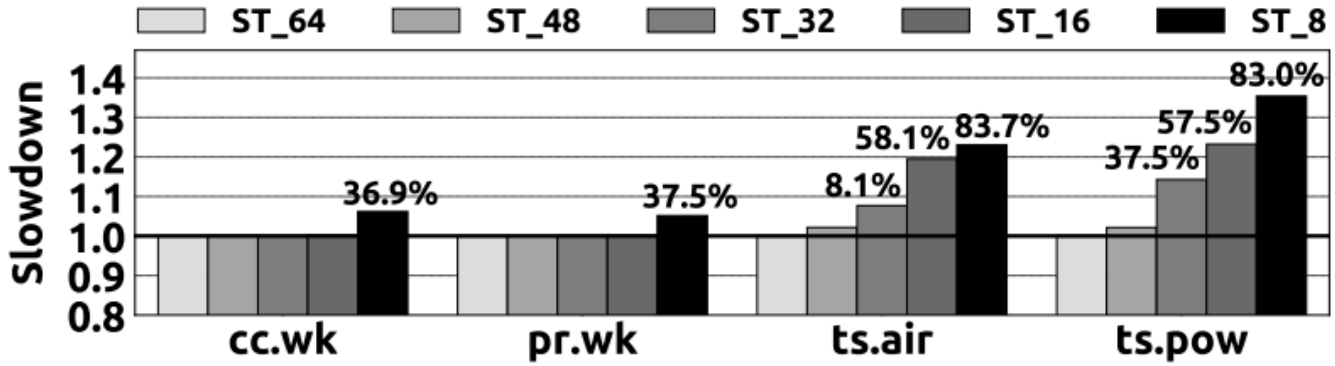


Figure 22: Slowdown with varying ST size (normalized to 64-entry ST). Numbers on top of bars show the percentage of overflowed requests.

Overflow Management Cost of Different Overflow Schemes

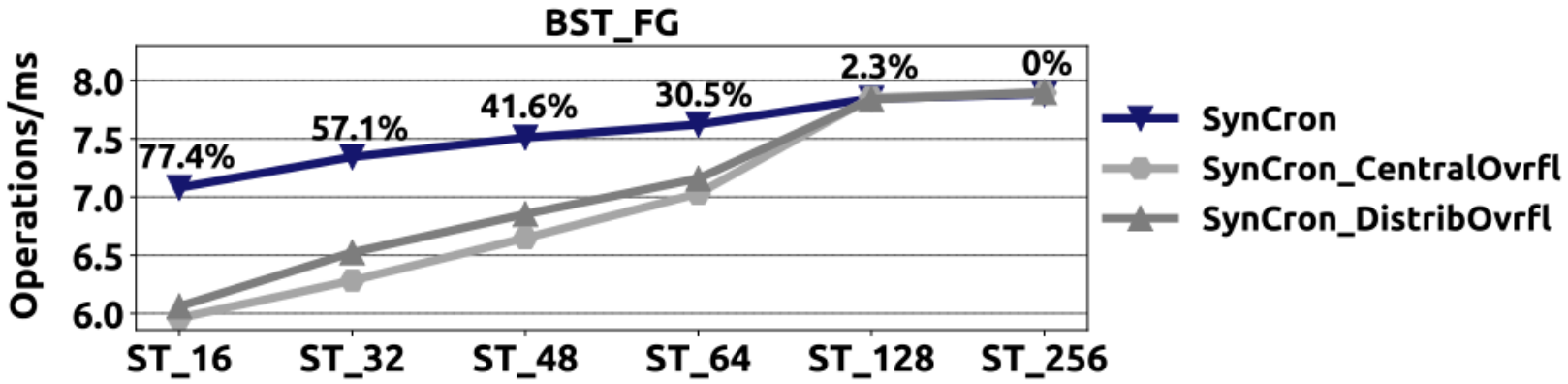


Figure 23: Throughput achieved by BST_FG using different overflow schemes and varying the ST size. The reported numbers show to the percentage of overflowed requests.

Area and Power Overheads of the Synchronization Engine

	SE (Synchronization Engine)	ARM Cortex A7 [14]
Technology	40nm	28nm
Area	SPU: 0.0141mm^2, ST: 0.0112mm^2 Indexing Counters: 0.0208mm^2 **Total:** 0.0461mm^2	32KB L1 Cache **Total:** 0.45mm^2
Power	2.7 mW	100mW

Table 8: Comparison of SE with a simple general-purpose in-order core, ARM Cortex A7.