

P&S Processing-in-Memory

Synchronization Support for Processing-in-Memory Architectures

Dr. Juan Gómez Luna

Prof. Onur Mutlu

ETH Zürich

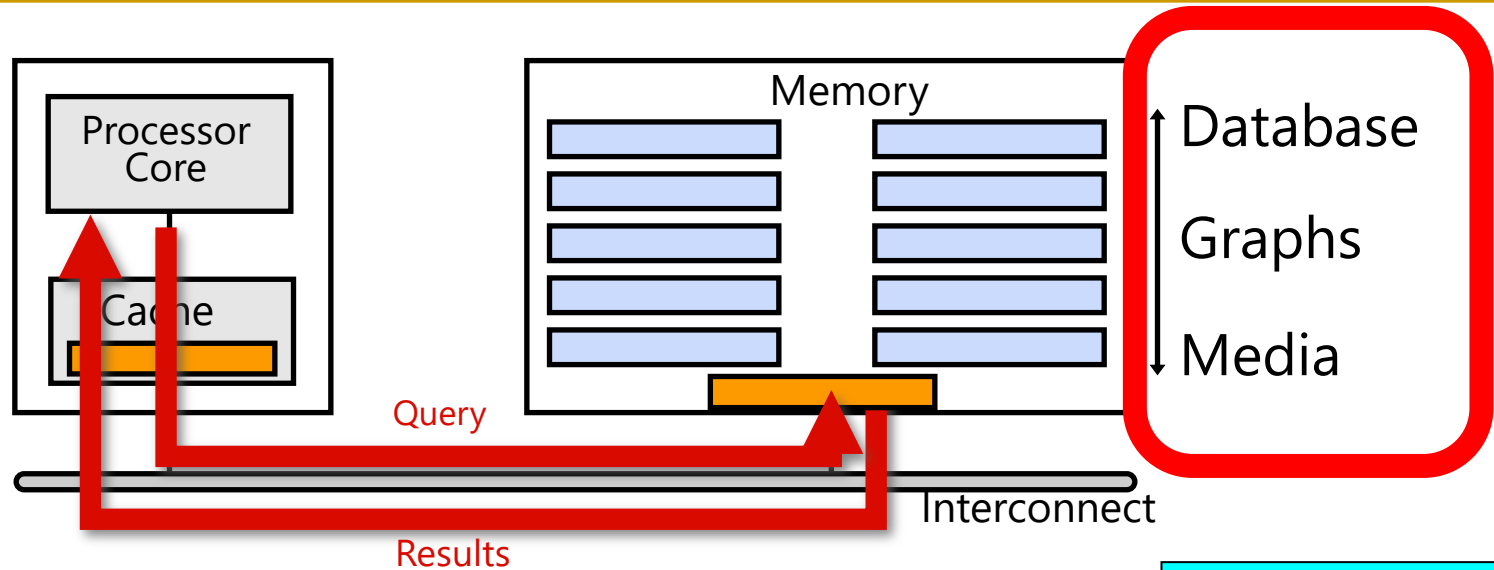
Fall 2021

14 December 2021

Recall: We Need A Paradigm Shift To ...

- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric

Goal: Processing Inside Memory



- Many questions... How do we design the:
 - ❑ compute-capable memory & controllers?
 - ❑ processor chip?
 - ❑ software and hardware interfaces?
 - ❑ system software and languages?
 - ❑ algorithms?

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons

Barriers to Adoption of PIM

1. Functionality of and applications & software for PIM
2. Ease of programming (interfaces and compiler/HW support)
3. System support: coherence & virtual memory
4. Runtime and compilation systems for adaptive scheduling, data mapping, access/sharing control
5. Infrastructures to assess benefits and feasibility

All can be solved with change of mindset

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures



P&S Processing-in-Memory
14.12.2021

Christina Giannoula

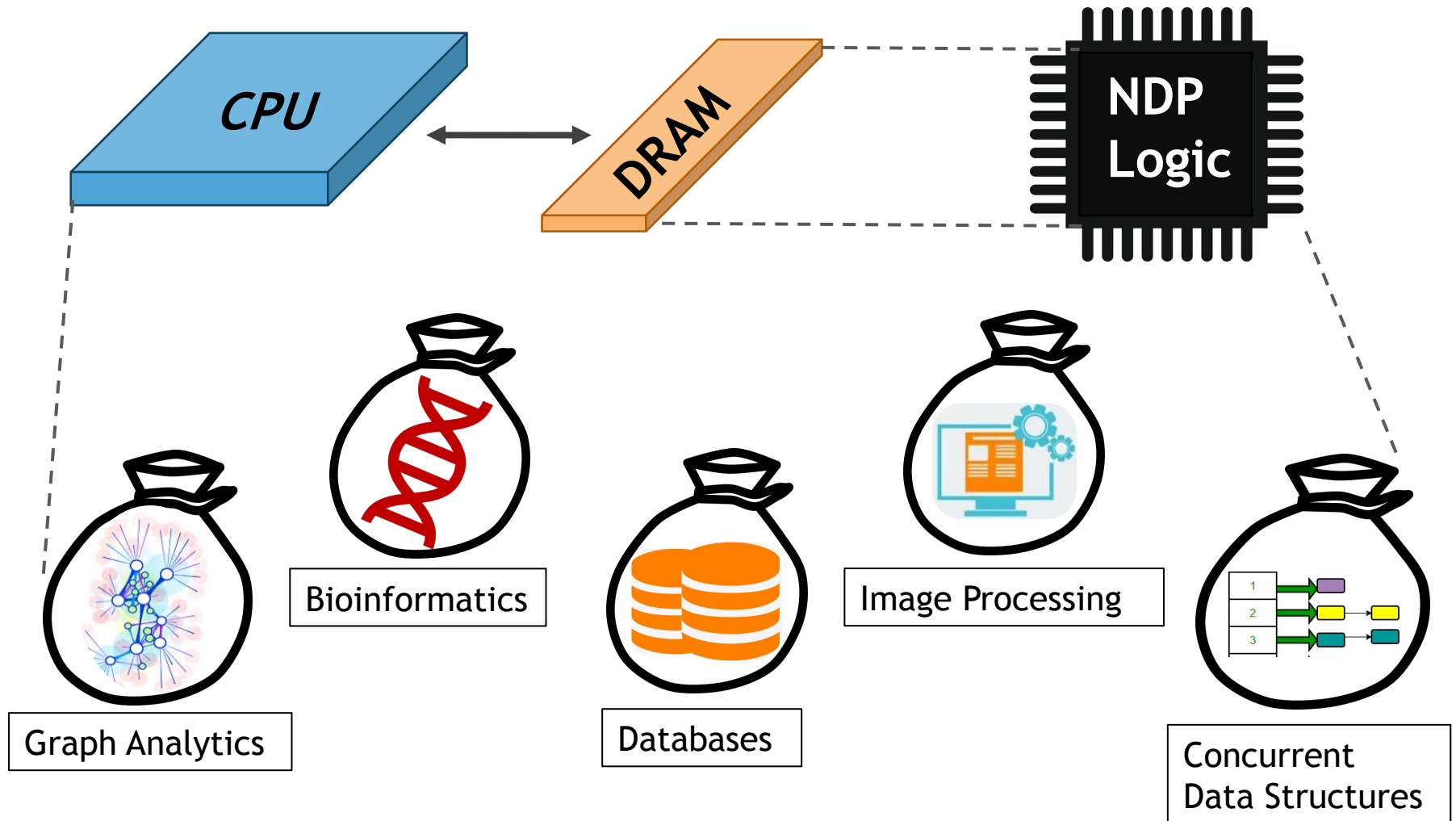
Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI **ETH** zürich

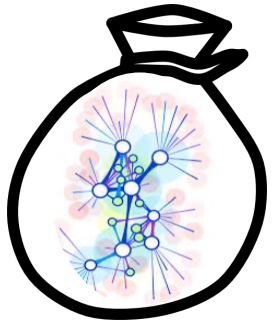
Agenda

- **How** is synchronization implemented in commodity architectures?
- Would **existing** hardware synchronization mechanisms **be efficient** in Near-Data-Processing Architectures?
- **How** to design an efficient synchronization mechanism for Near-Data-Processing Architectures?

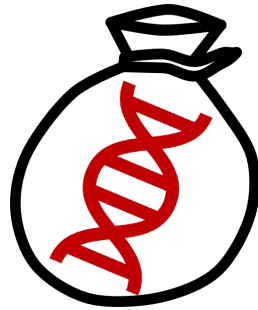
Near-Data-Processing (**NDP**) Systems



Synchronization is Necessary



Graph Analytics



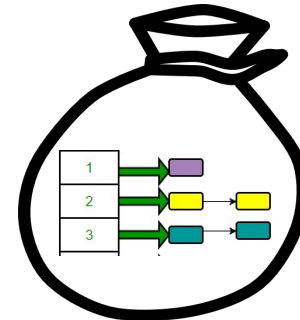
Bioinformatics



Databases



Image Processing



Concurrent Data Structures

Single Source Shortest Path (SSSP)

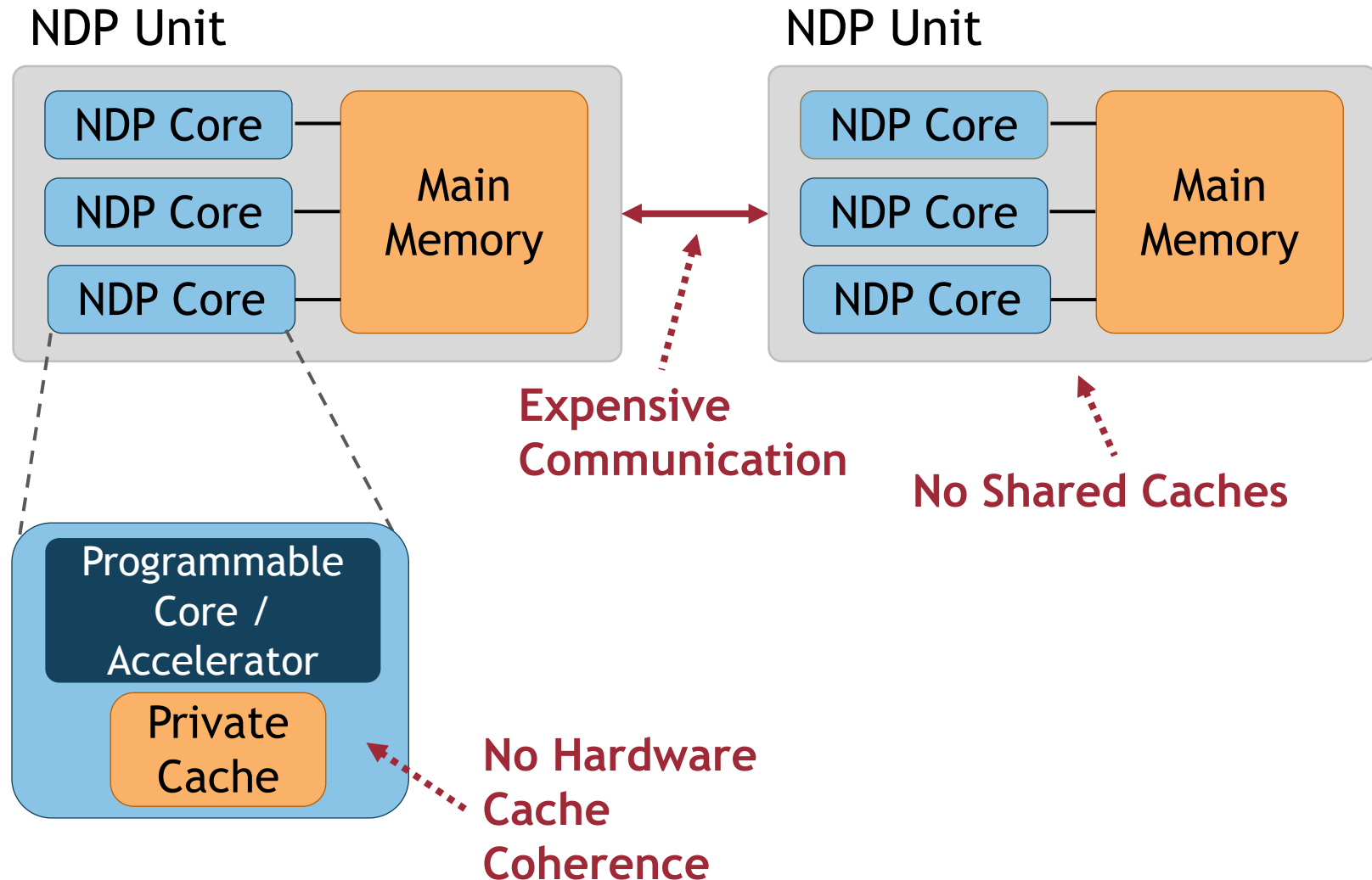
Locks

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. **Generality** to Cover a Wide Range of
Synchronization Primitives

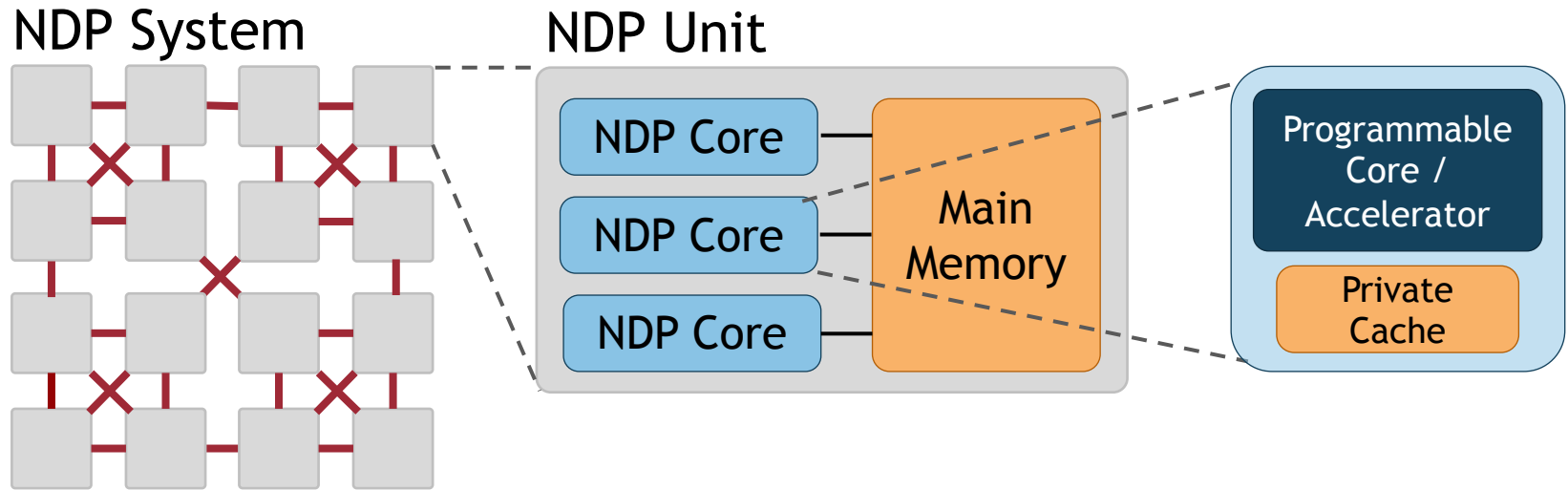
Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation

Baseline NDP Architecture



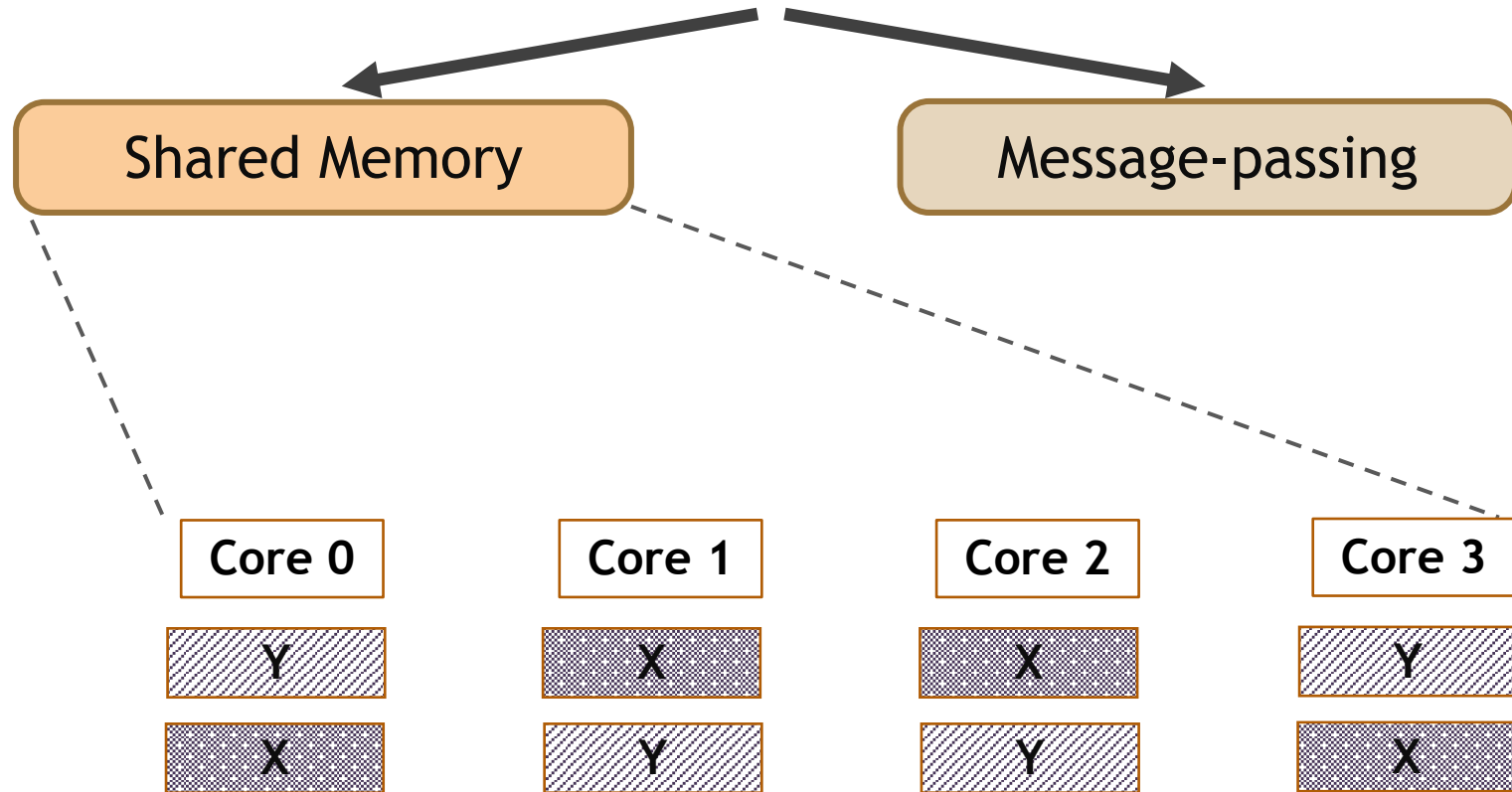
Synchronization **challenges in NDP** systems:

- (1) Lack of a shared level of cache memory
- (2) Lack of hardware cache coherence support
- (3) Expensive communication across NDP units

Synchronization Solution Space

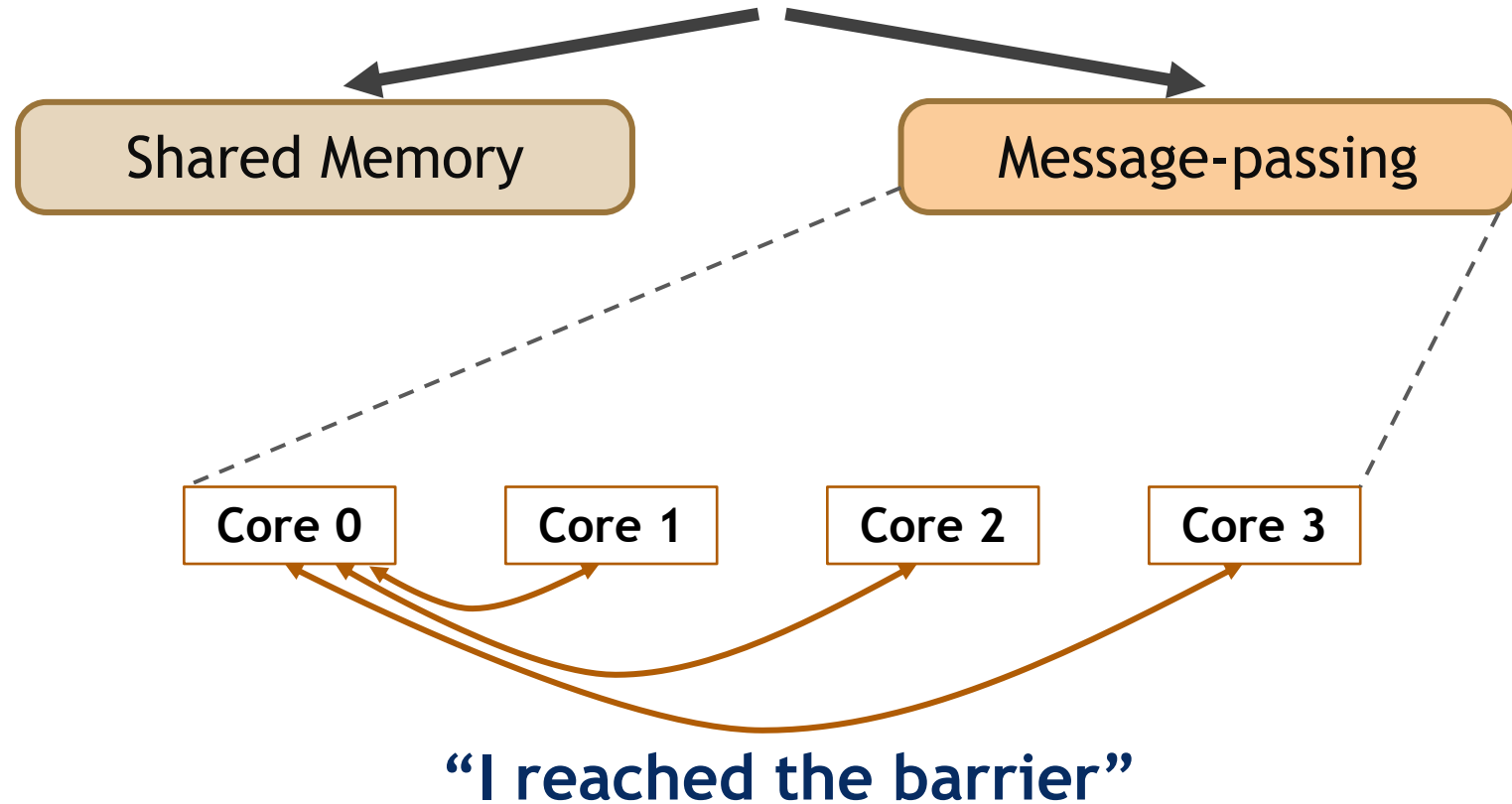


Synchronization Solution Space

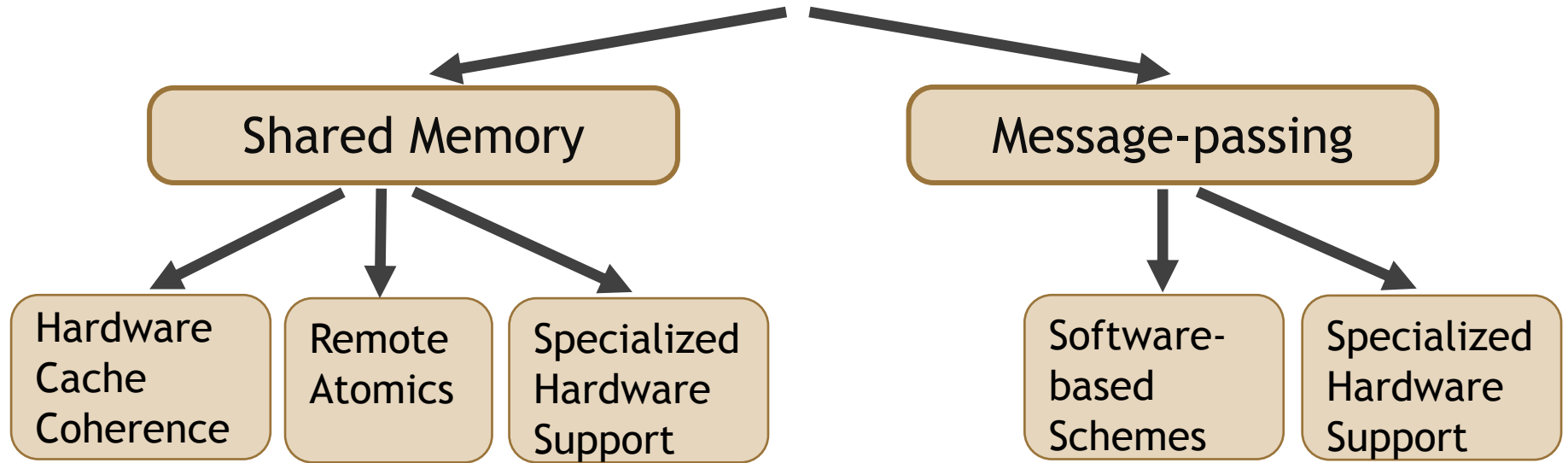


Shared memory locations

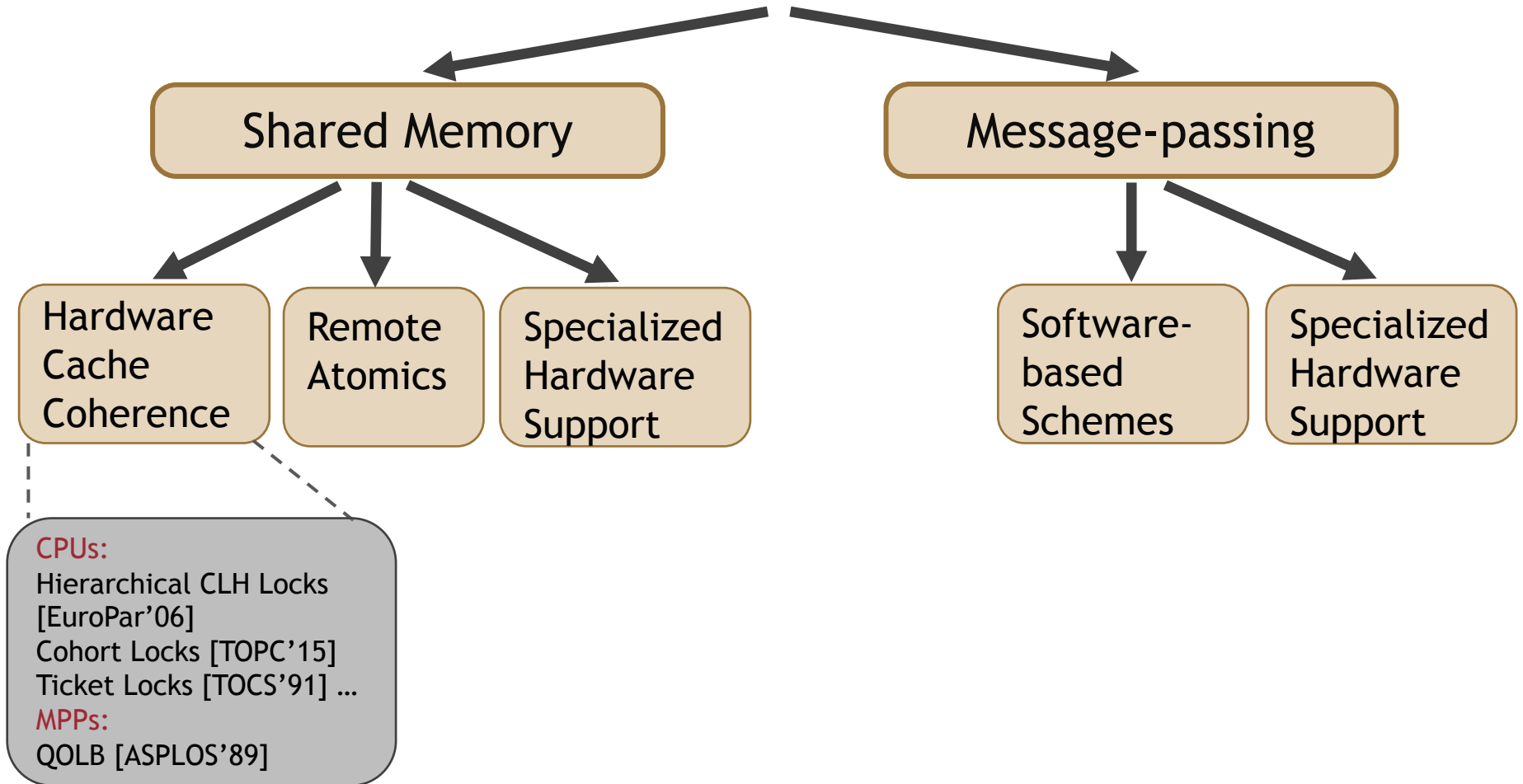
Synchronization Solution Space



NDP Synchronization Solution Space

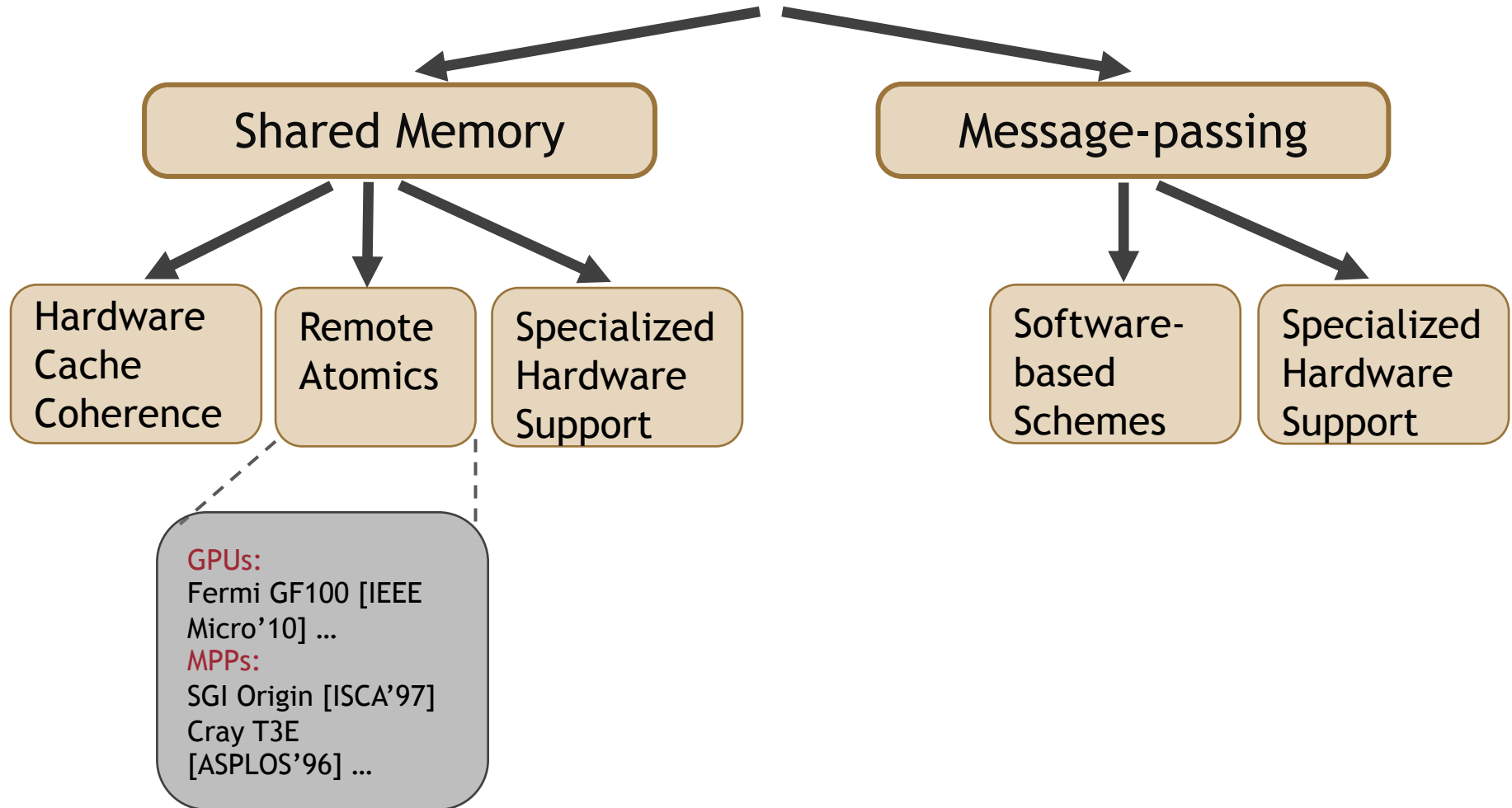


NDP Synchronization Solution Space



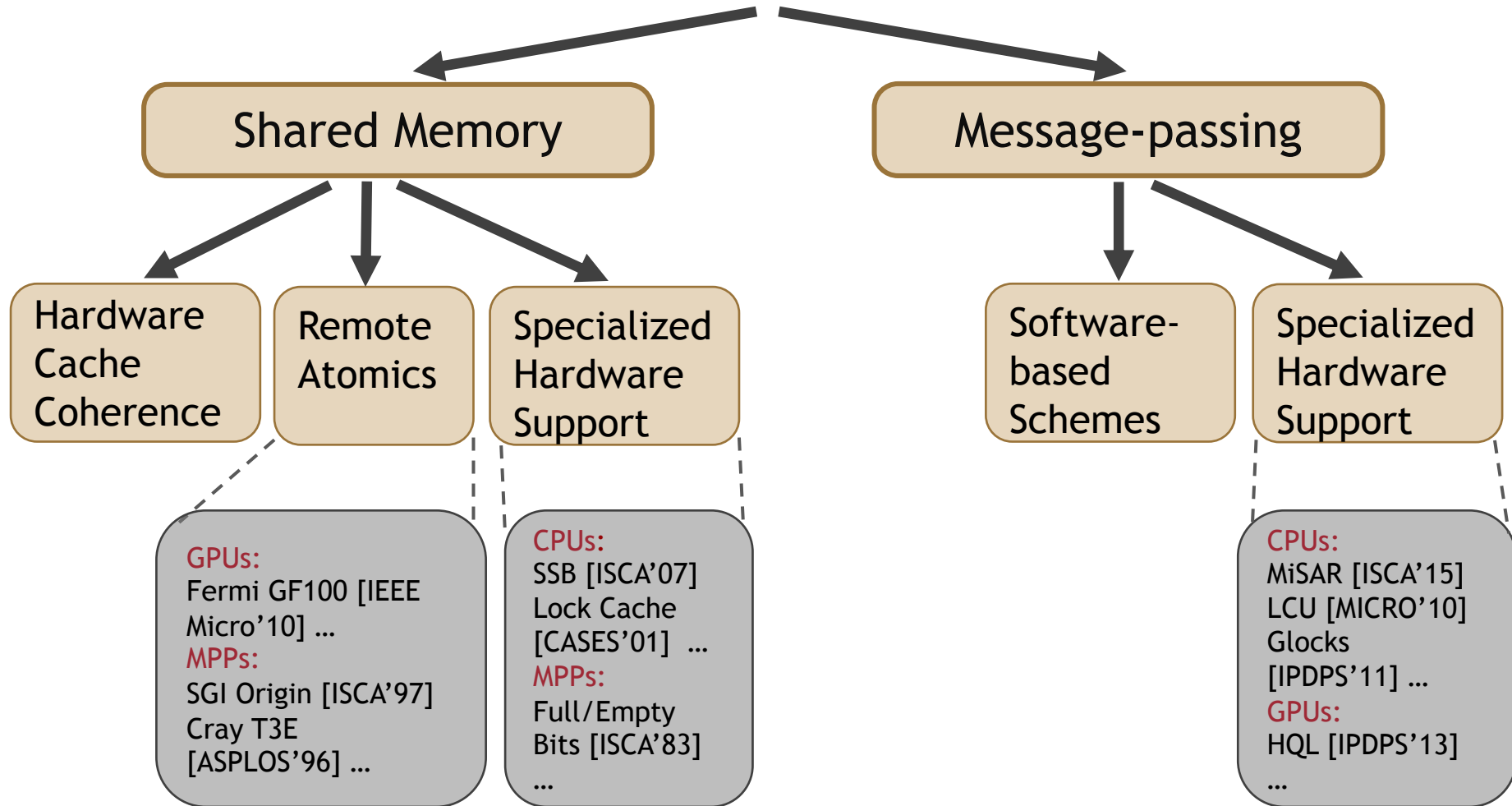
Lack of hardware cache coherence support

NDP Synchronization Solution Space



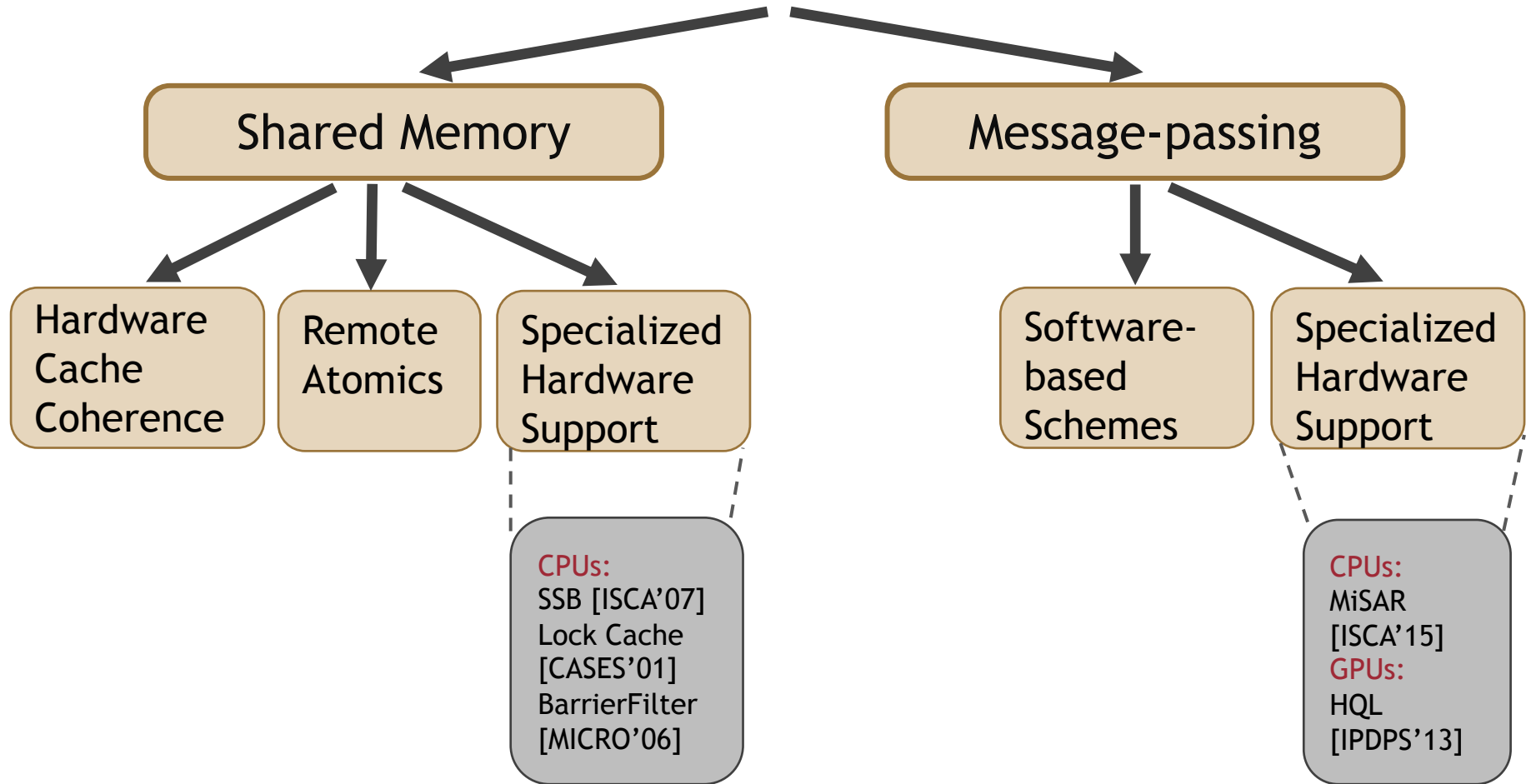
Expensive communication across NDP units

NDP Synchronization Solution Space



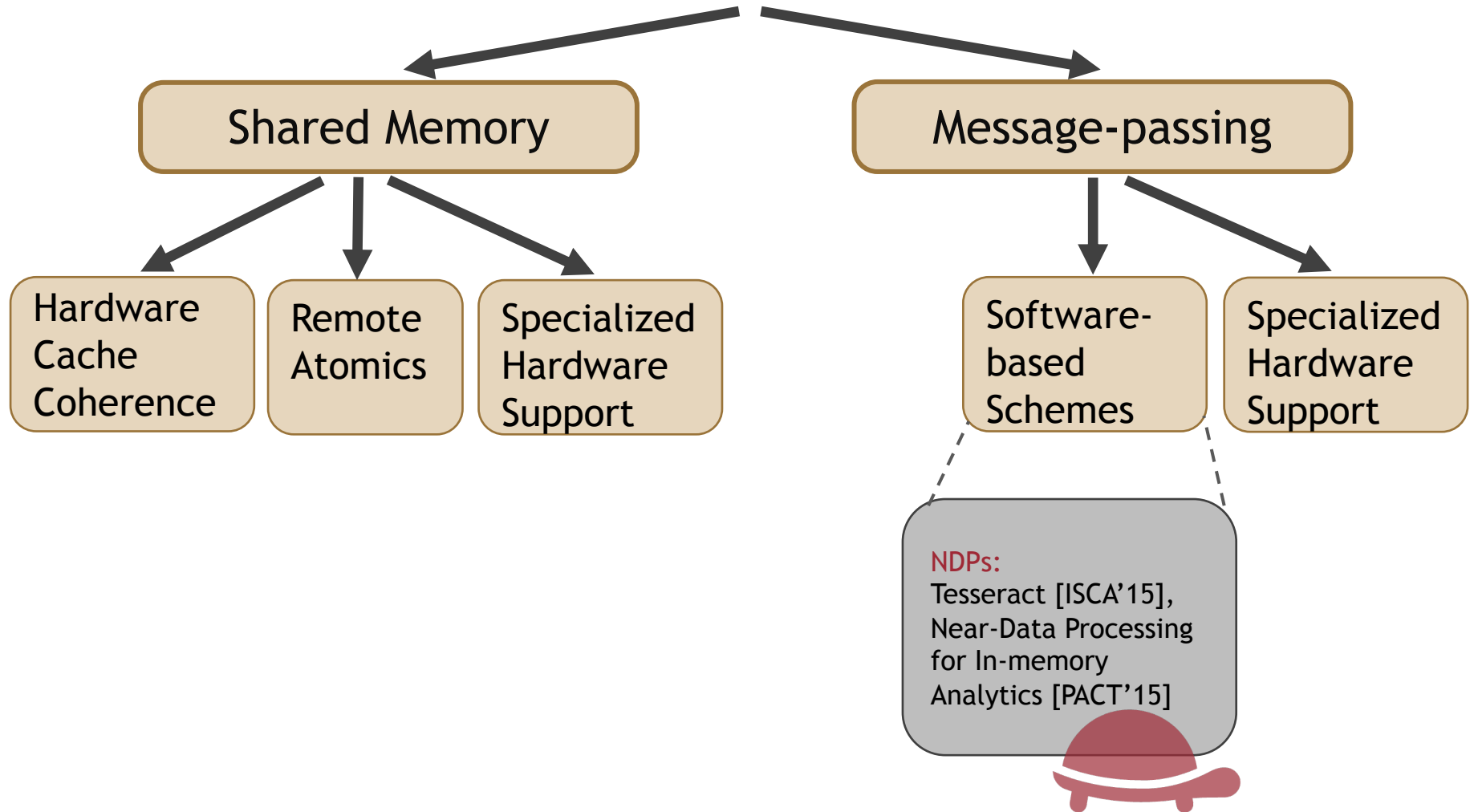
Expensive communication across NDP units

NDP Synchronization Solution Space

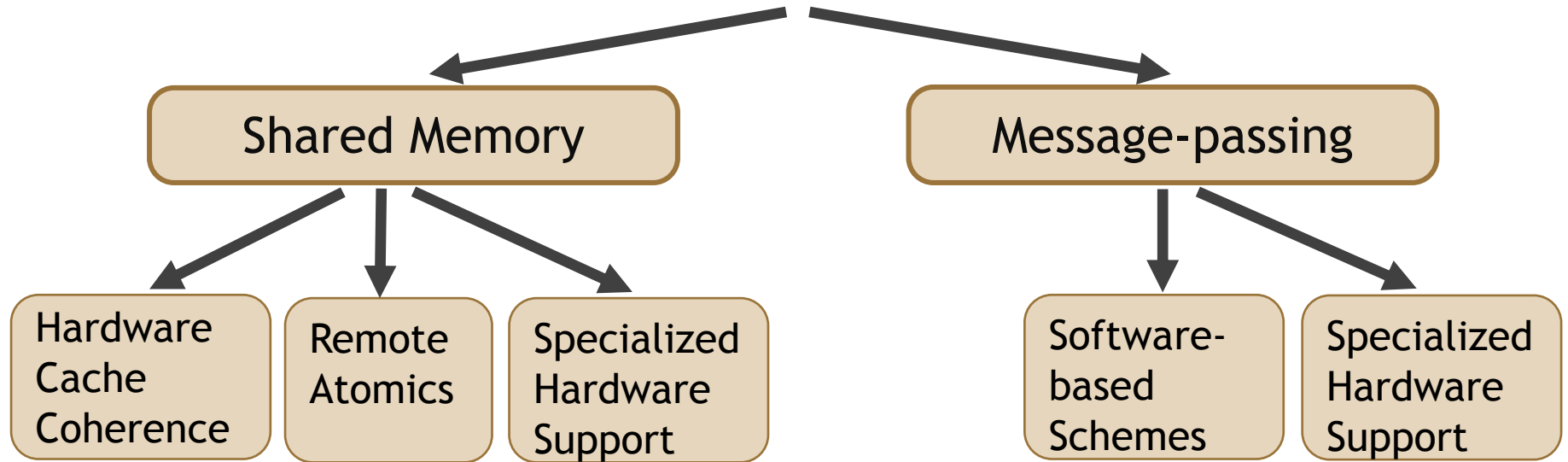


Lack of a shared level of cache memory

NDP Synchronization Solution Space

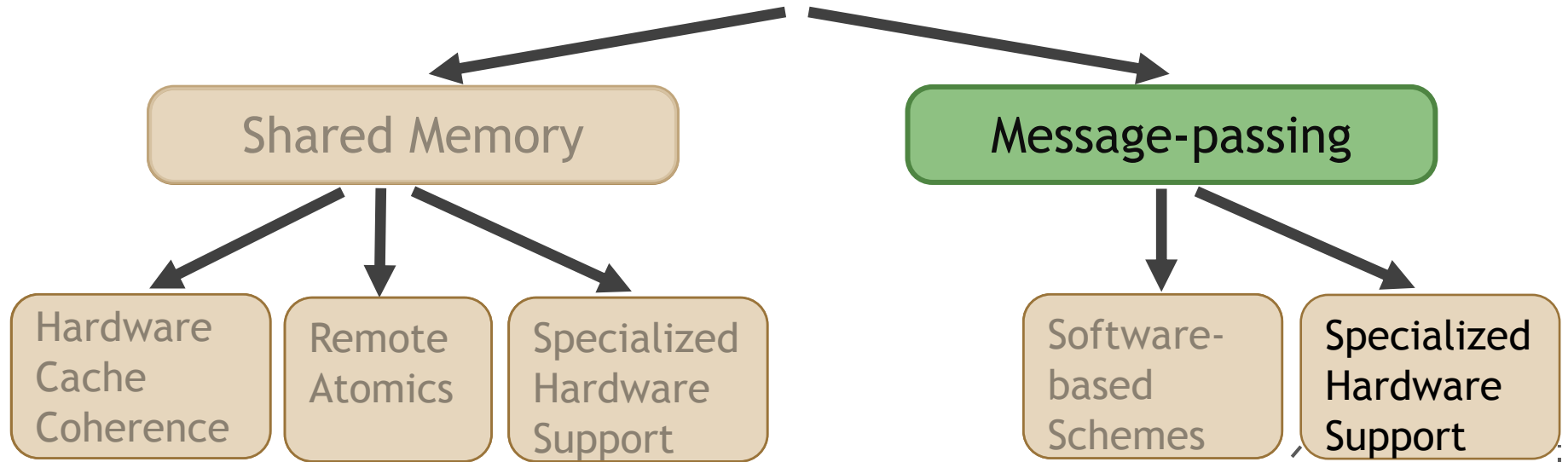


NDP Synchronization Solution Space



Prior schemes are **not suitable** or **efficient**
for NDP systems

NDP Synchronization Solution Space

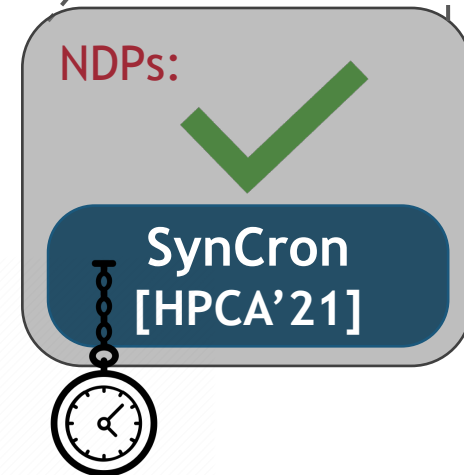


SynCron's Design Choices

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

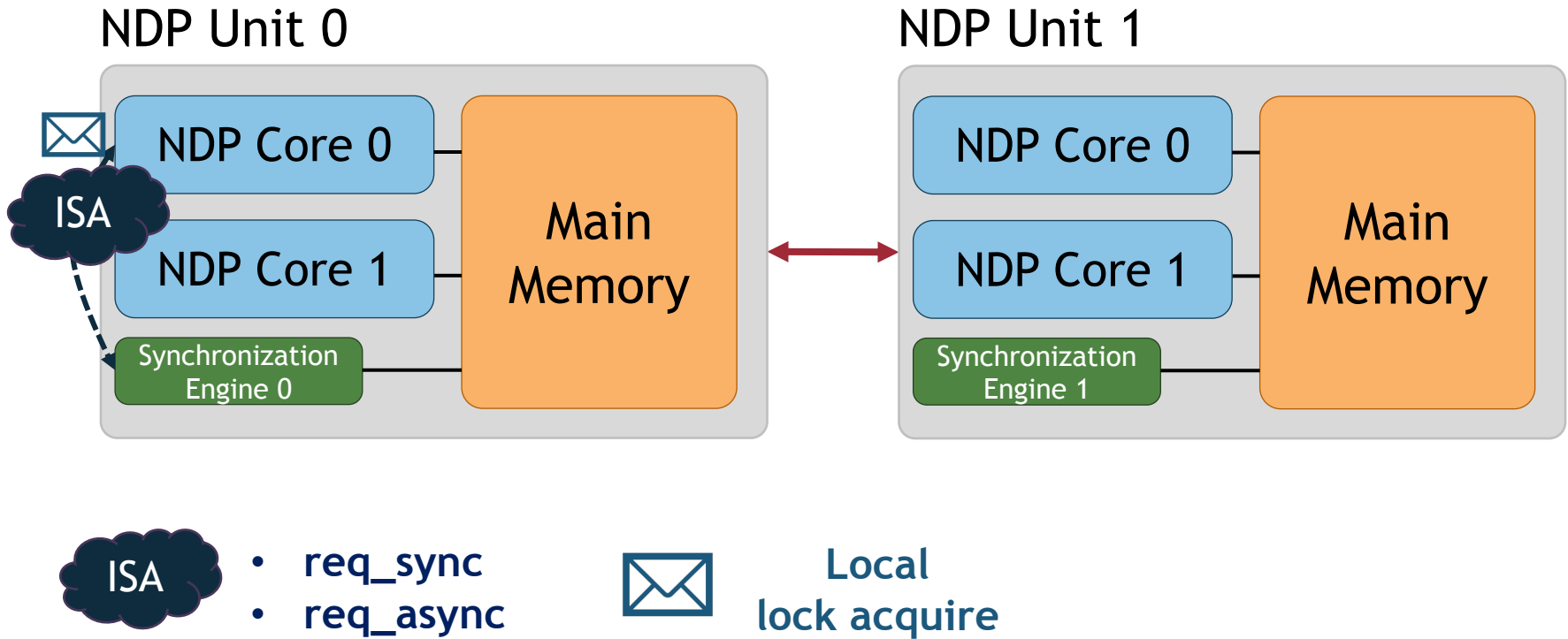
Evaluation

SynCron: Overview

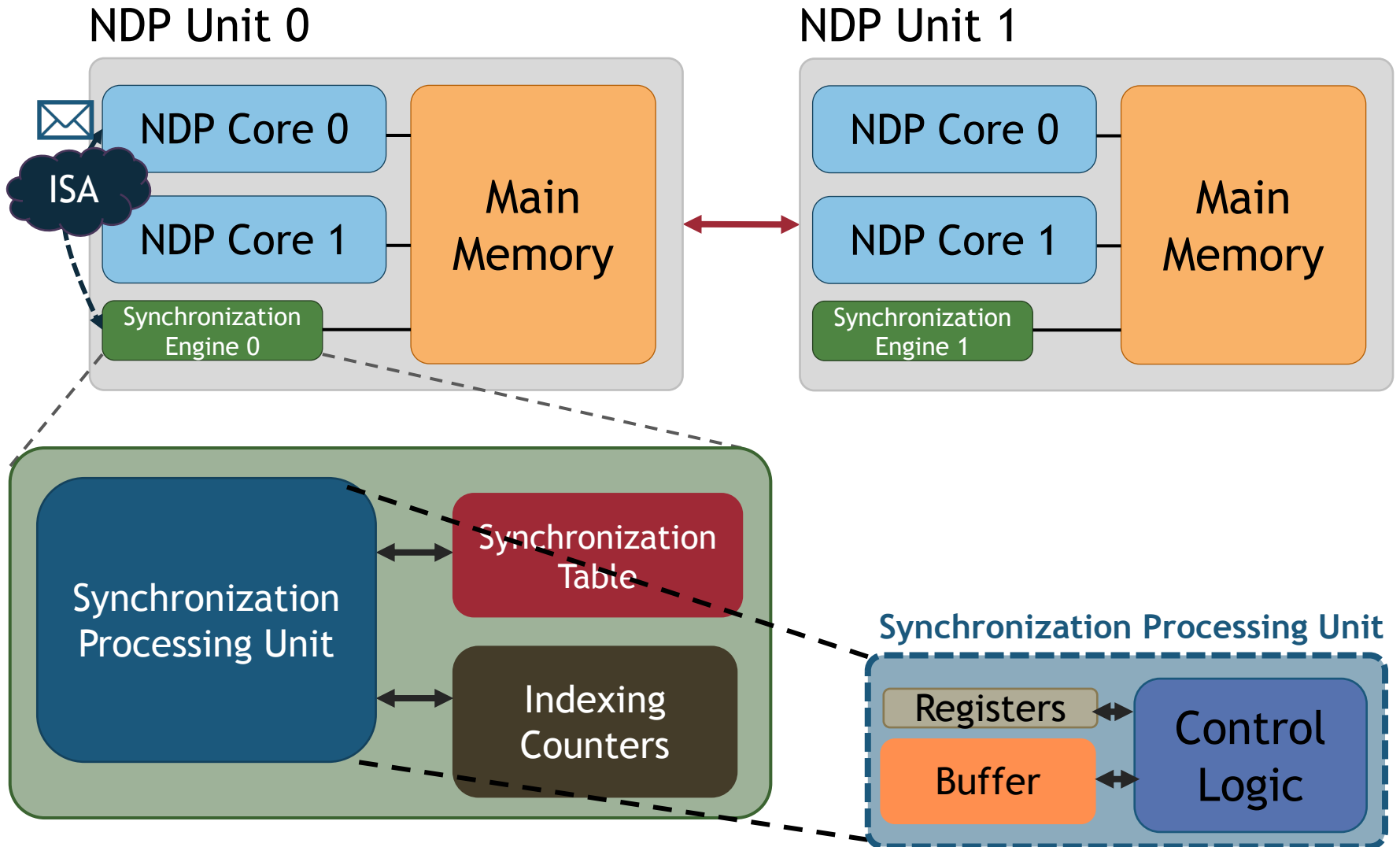
SynCron consists of **four key techniques**:

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

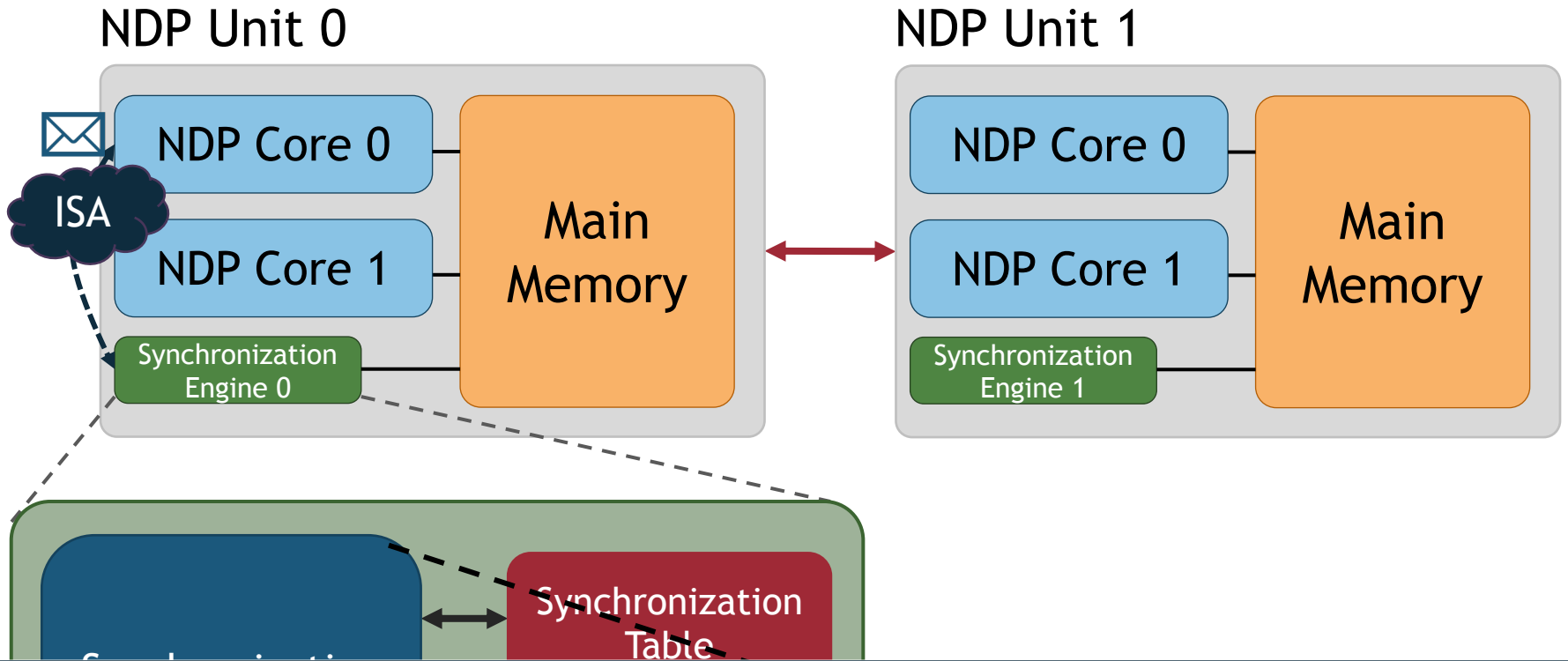
1. Hardware Synchronization Support



1. Hardware Synchronization Support

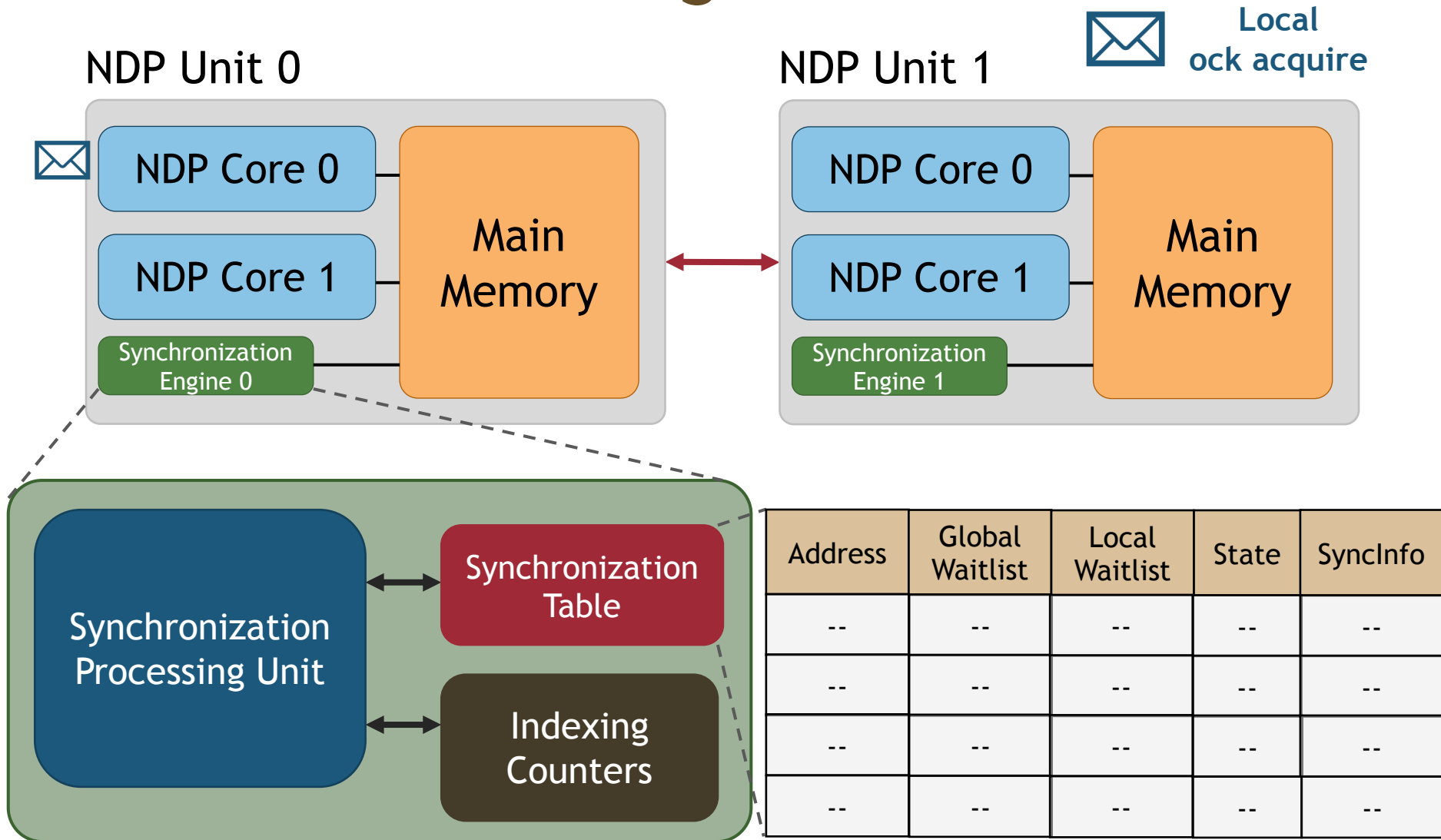


1. Hardware Synchronization Support

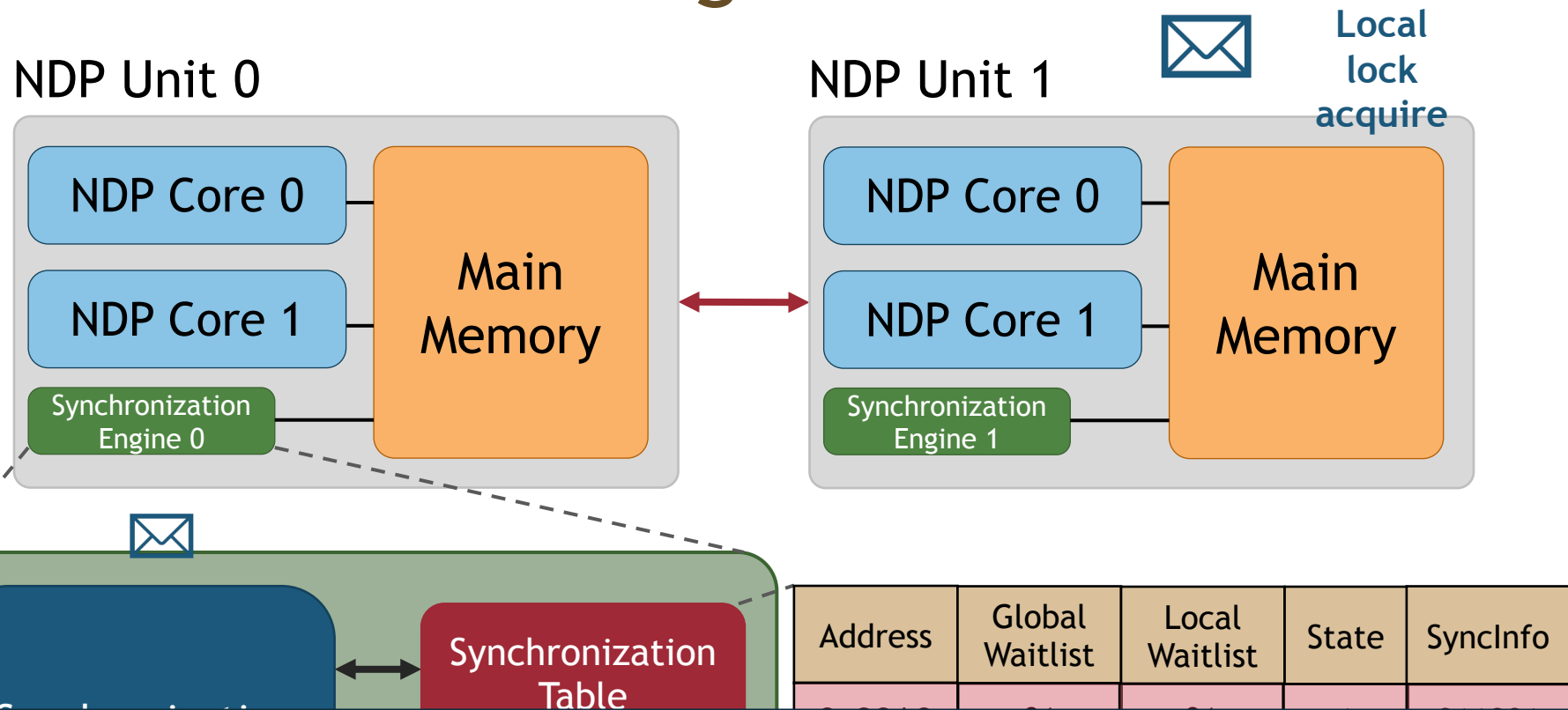


- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

2. Direct Buffering of Variables



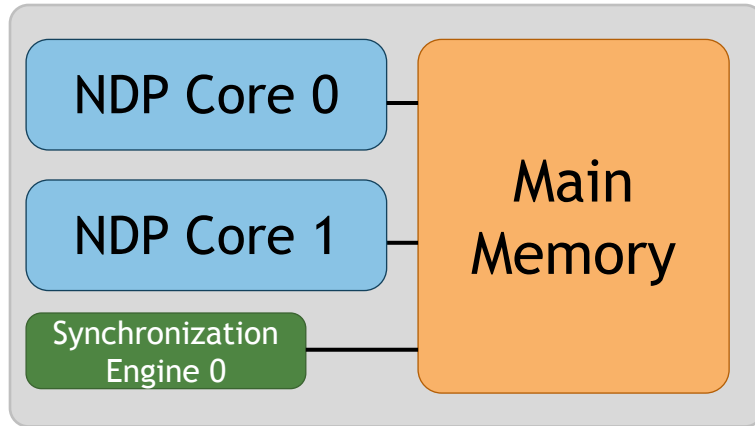
2. Direct Buffering of Variables



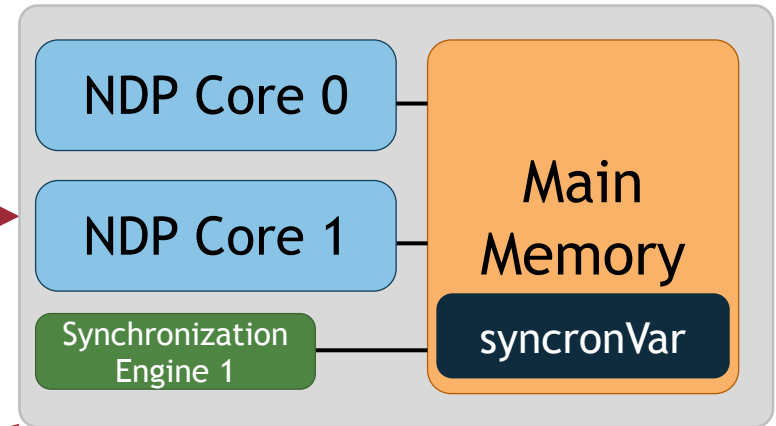
- ✓ No Costly Memory Accesses
- ✓ Low Latency

3. Hierarchical Communication

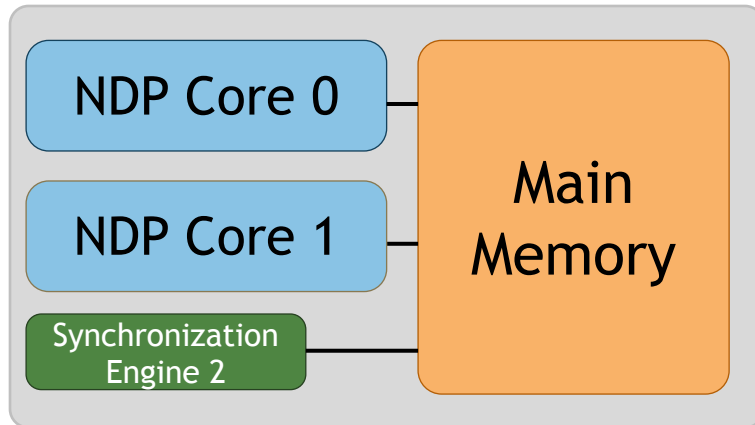
NDP Unit 0



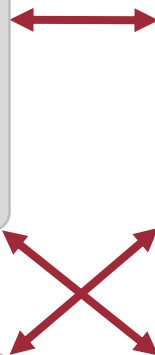
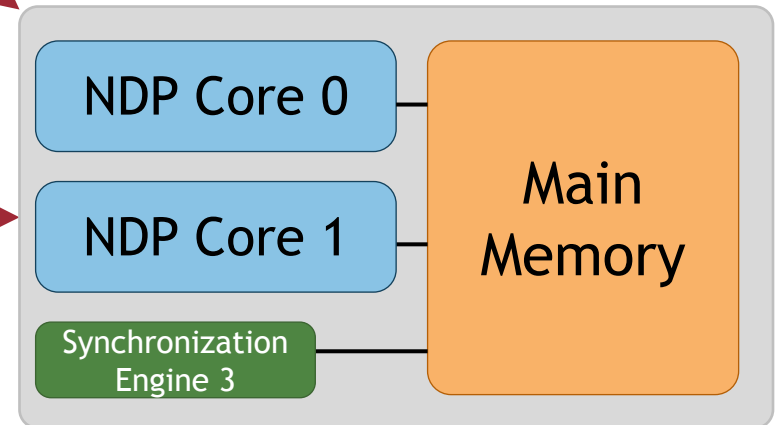
NDP Unit 1



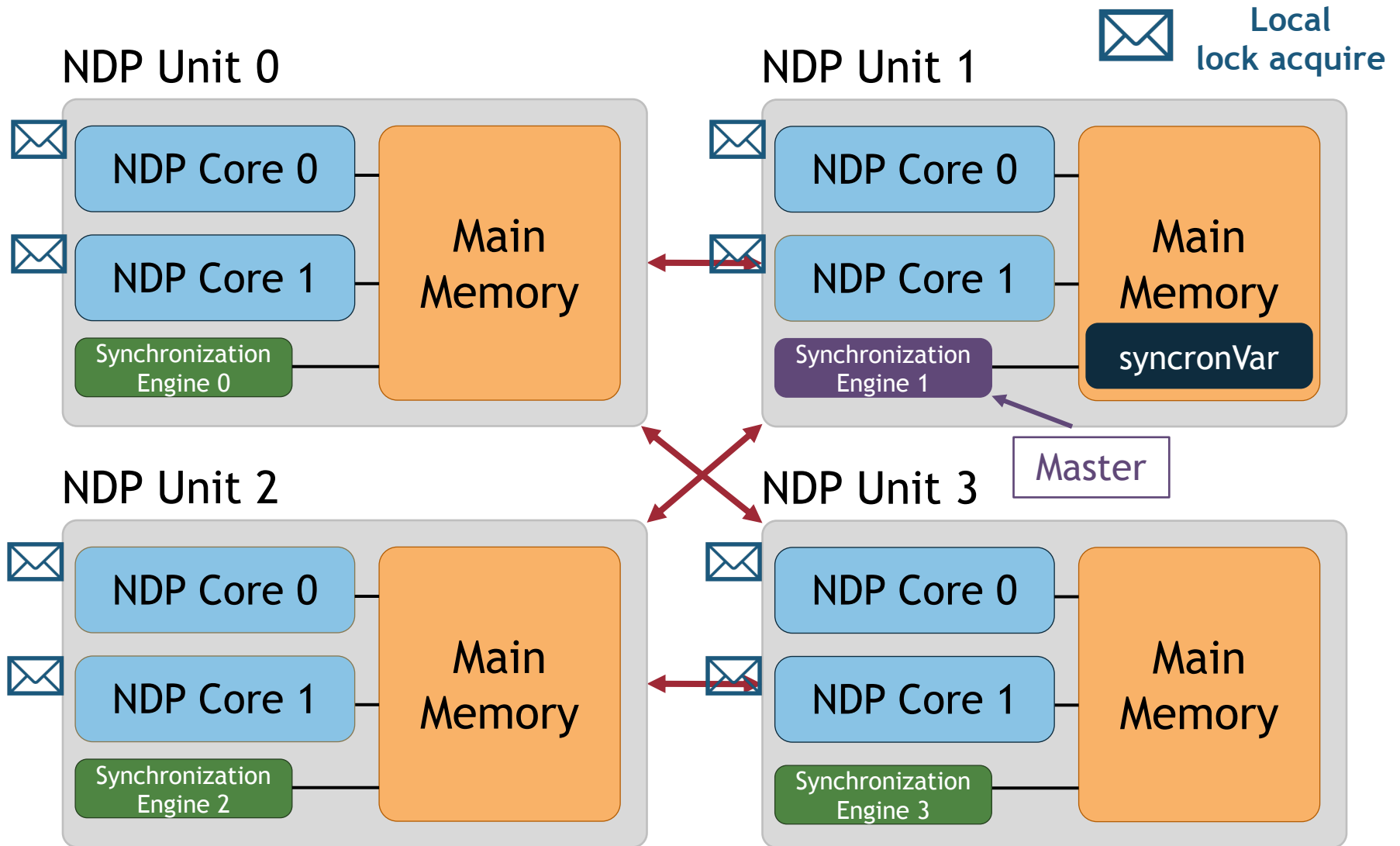
NDP Unit 2



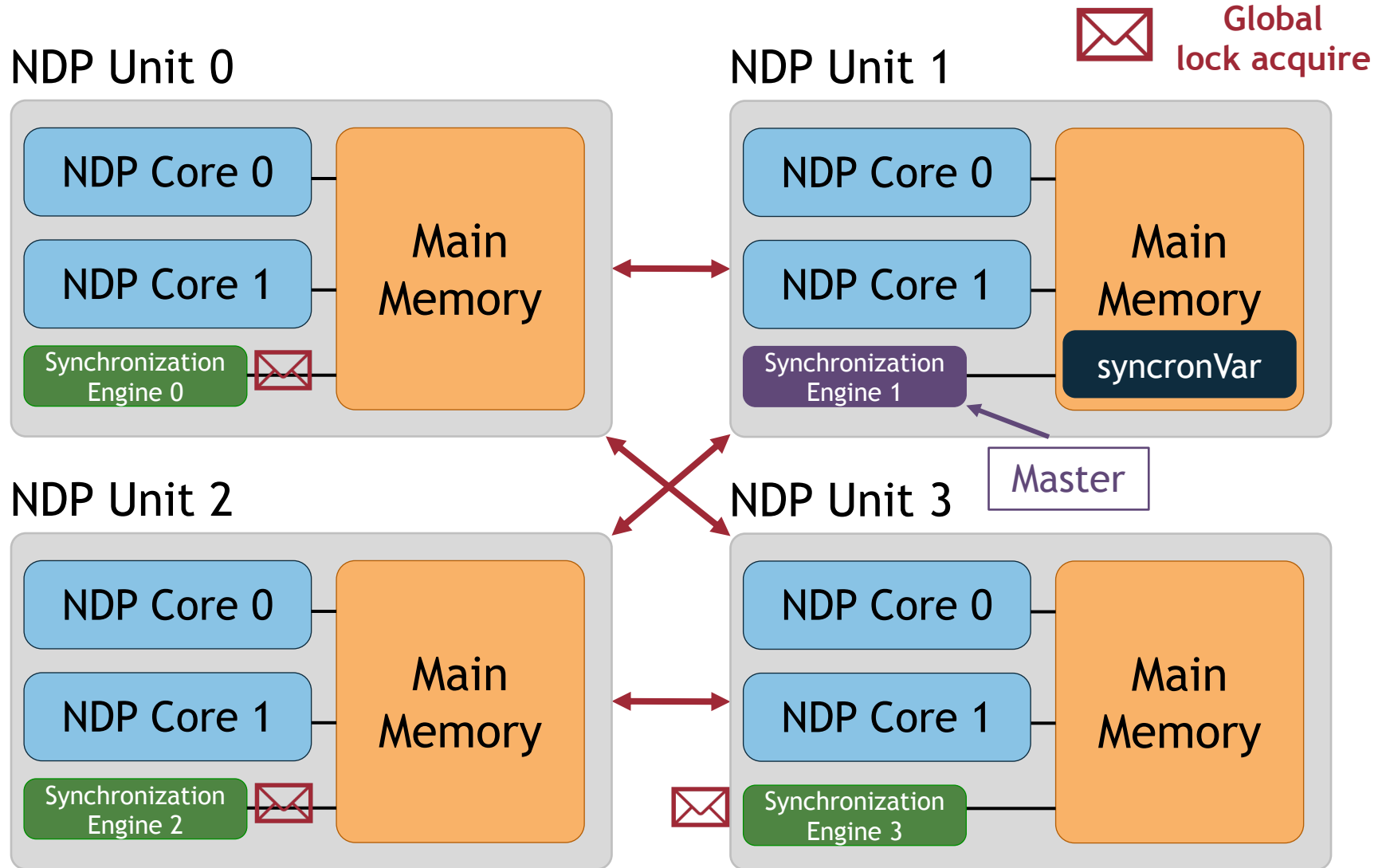
NDP Unit 3



3. Hierarchical Communication

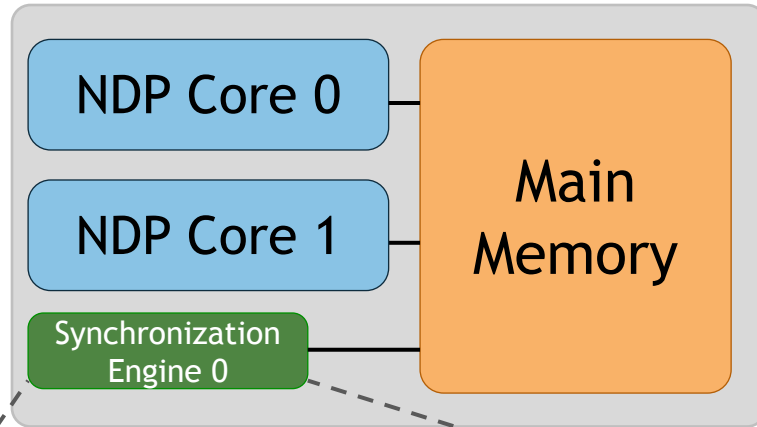


3. Hierarchical Communication

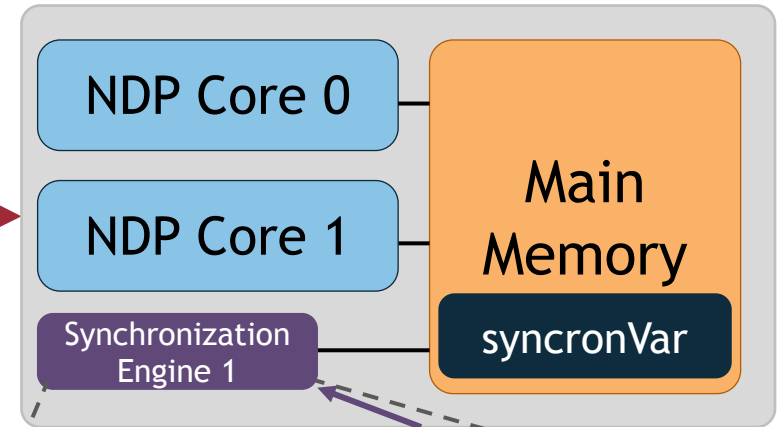


3. Hierarchical Communication

NDP Unit 0

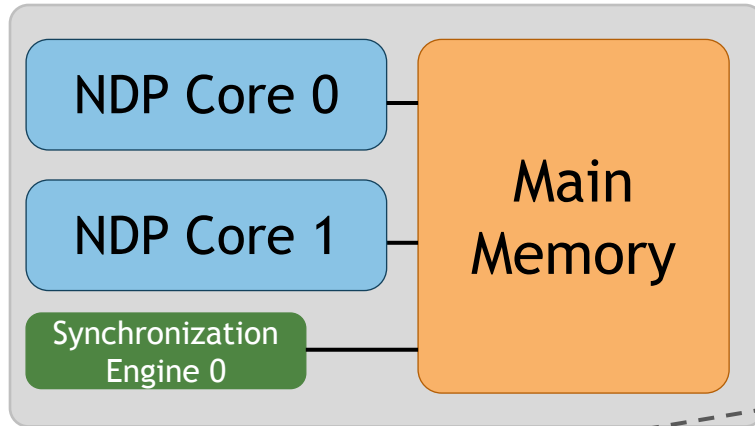


NDP Unit 1

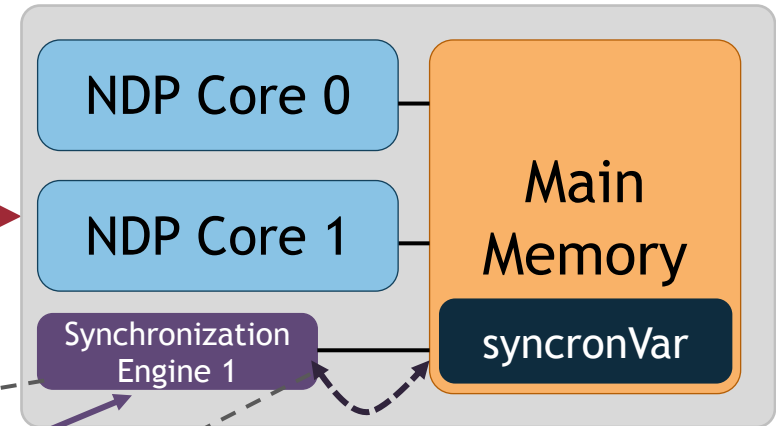


4. Integrated Overflow Management

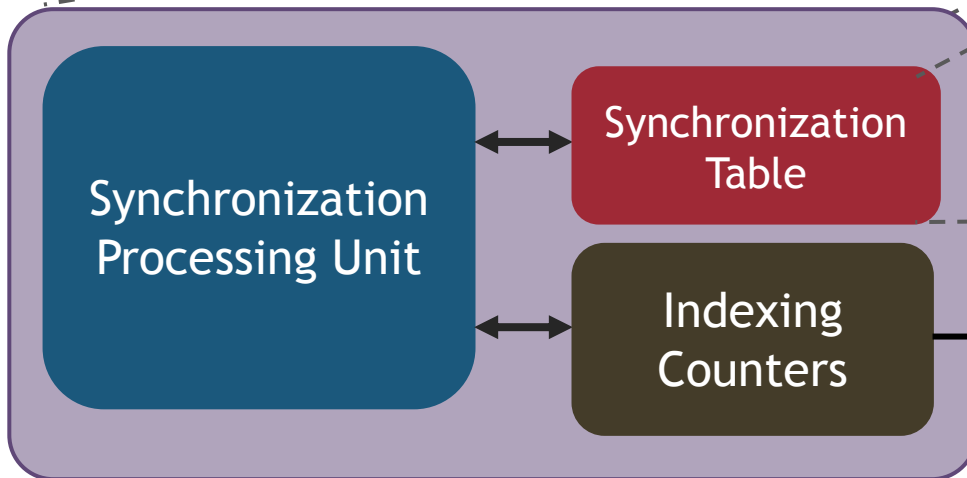
NDP Unit 0



NDP Unit 1



Master



Address	...
0x33A9	...
0x2241	...
0x438C	...
0x6B4A	...

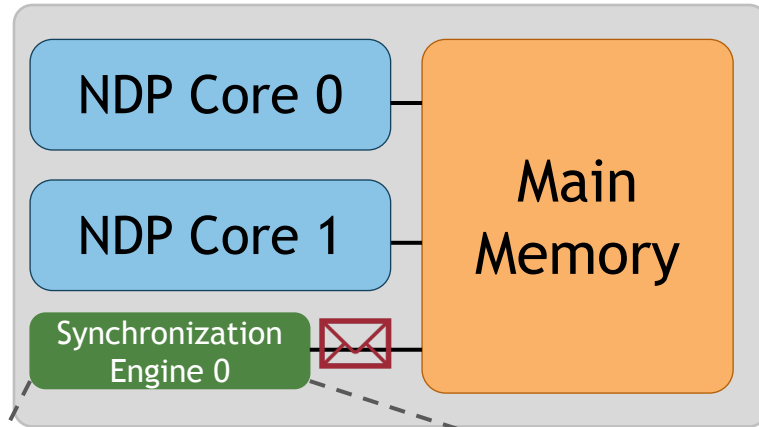
Fully Occupied

synchronVar
Address

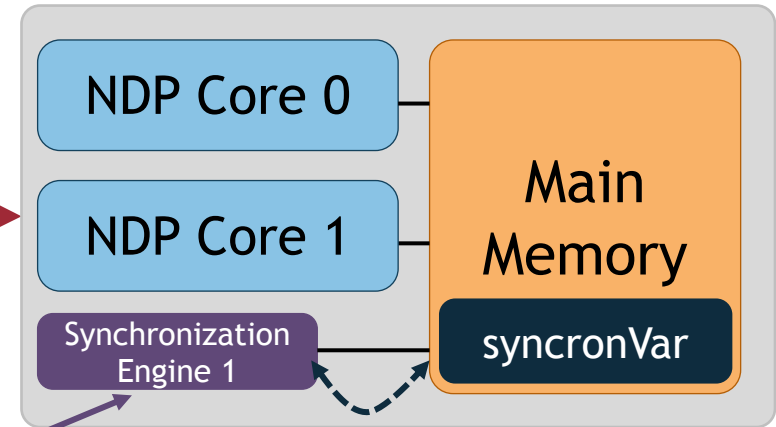
```
Counter0 = 0
Counter1 > 0
Counter2 = 0
Counter3 = 0
```


4. Integrated Overflow Management

NDP Unit 0



NDP Unit 1



 Global overflow lock acquire

Master

Address	...
0x33A9	...
0x2241	...
0x1386	...

Fully Occupied

- ✓ Low Performance Degradation
- ✓ High Programming Ease

Counter3 = 0

SynCron's Supported Primitives

Lock primitive

- `lock_acquire()`
- `lock_release()`

Barrier

- `ba`
- `ba`

Synchronization Metadata:

- i. Queueing list
- ii. Condition to be satisfied

Semaphore primitive

- `sem_wait()`
- `sem_post()`

Condition variable primitive

- `cond_wait()`
- `cond_signal()`
- `cond_broadcast()`

SynCron's Supported Primitives

Lock primitive

- `lock_acquire()`
- `lock_release()`

Barrier primitive

- `barrier_wait_within_NDP_unit()`
- `barrier_wait_across_NDP_units()`

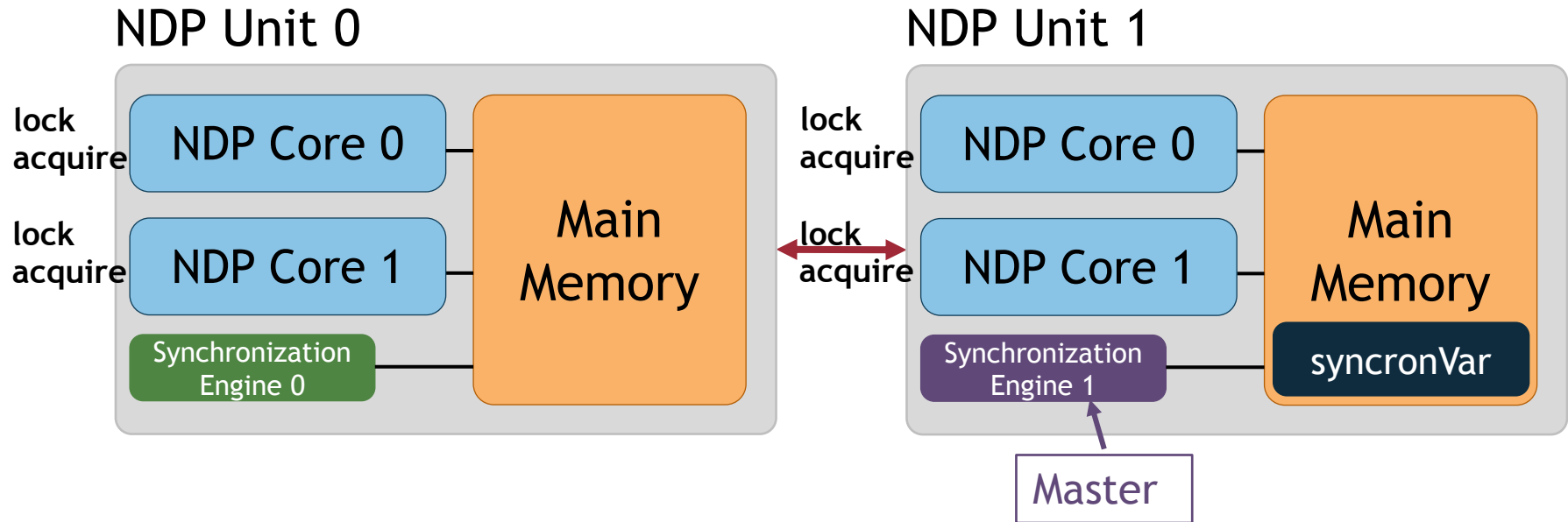
Semaphore primitive

- `sem_wait()`
- `sem_post()`

Condition variable primitive

- `cond_wait()`
- `cond_signal()`
- `cond_broadcast()`

Lock Operation



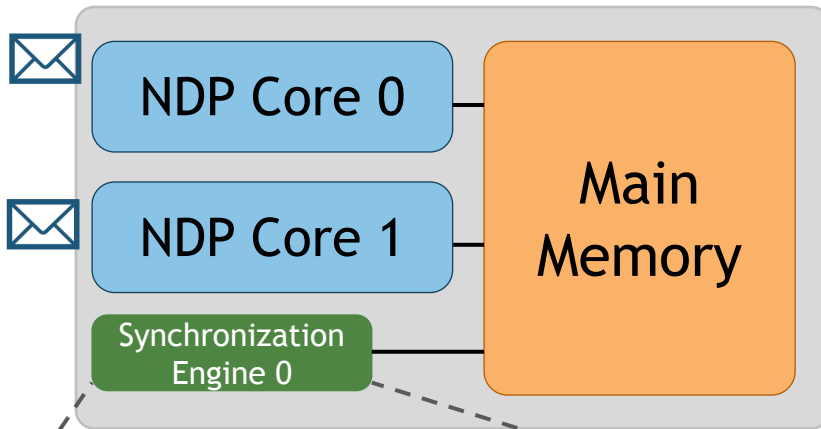
All NDP cores compete for the same lock variable

Lock Operation

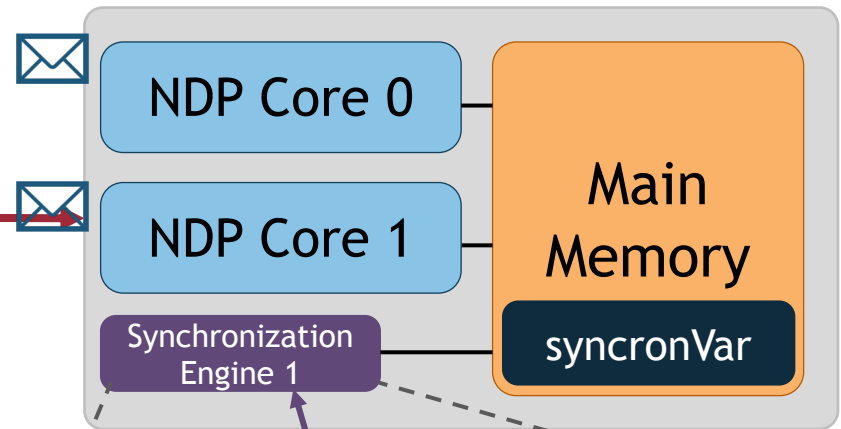


Local
lock acquire

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

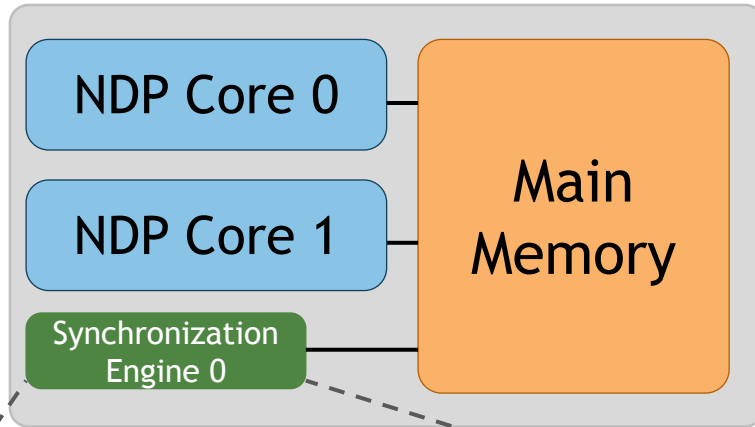
Indexing
Counters

Lock Operation

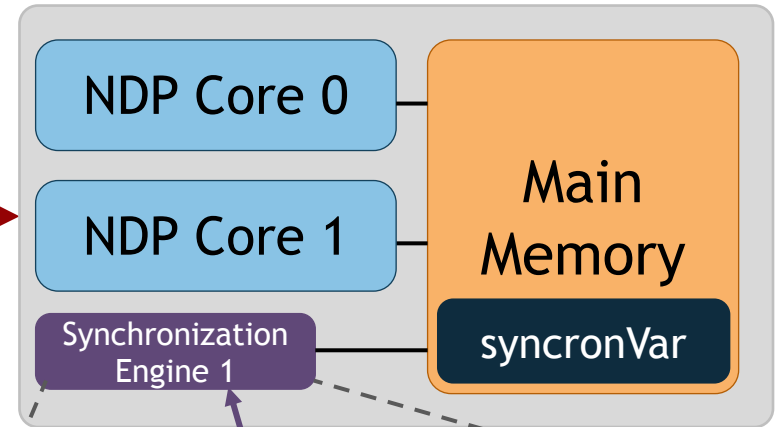


Global
lock acquire

NDP Unit 0



NDP Unit 1



Master



Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

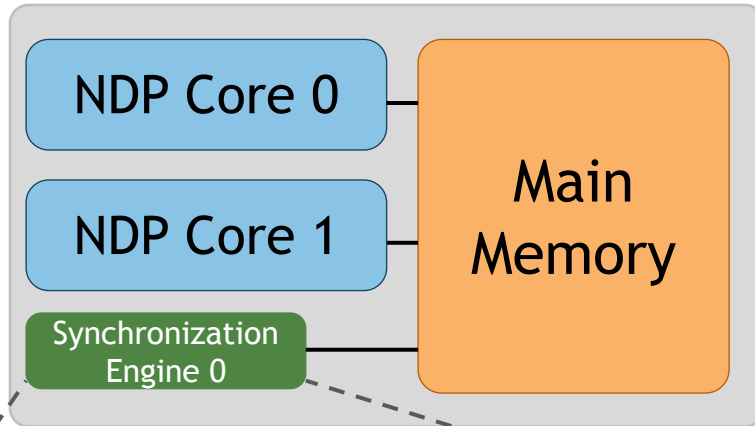
Indexing
Counters

Lock Operation

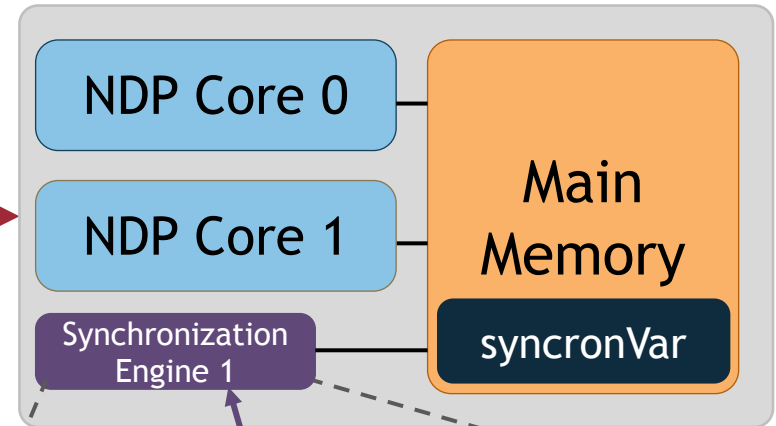


Global
lock acquire

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	01	11	...

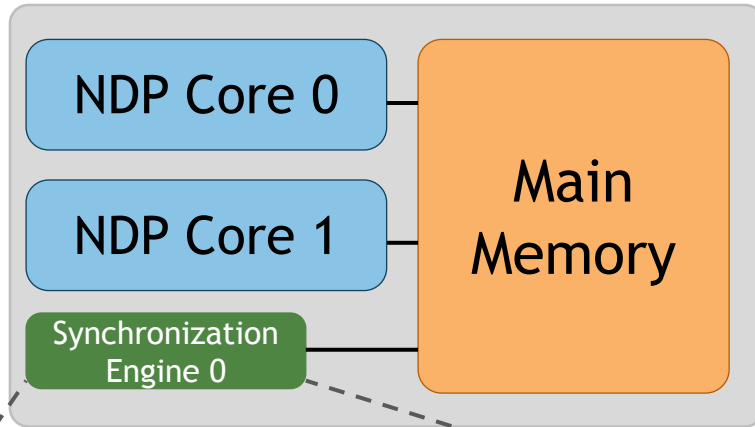
Indexing
Counters

Lock Operation

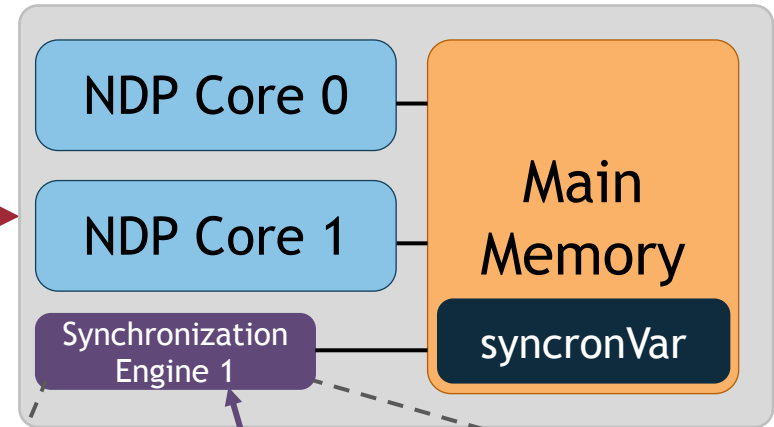


Local
lock grant

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	01	11	...

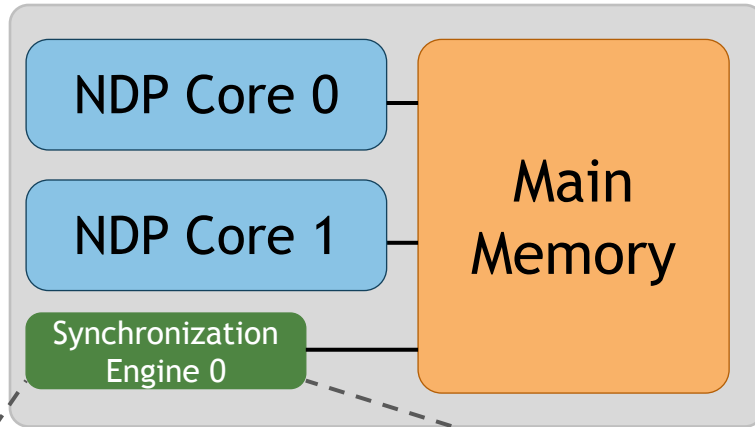
Indexing
Counters

Lock Operation

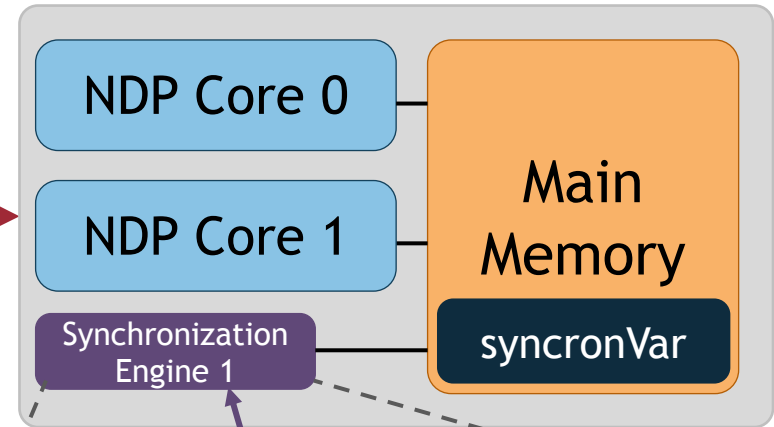


Global
lock grant

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	11	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	01	00	...

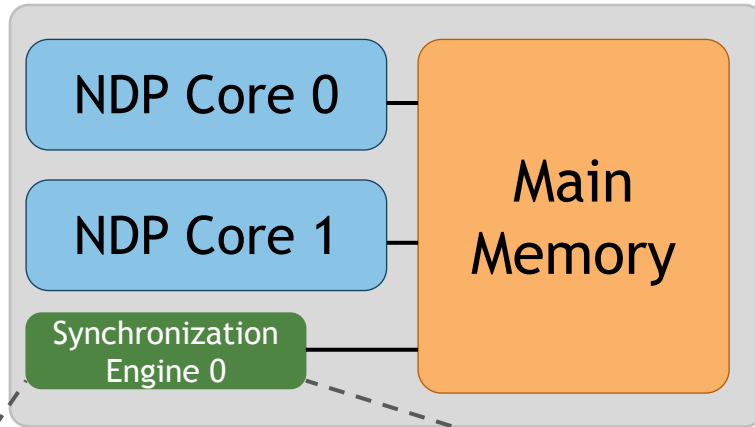
Indexing
Counters

Lock Operation

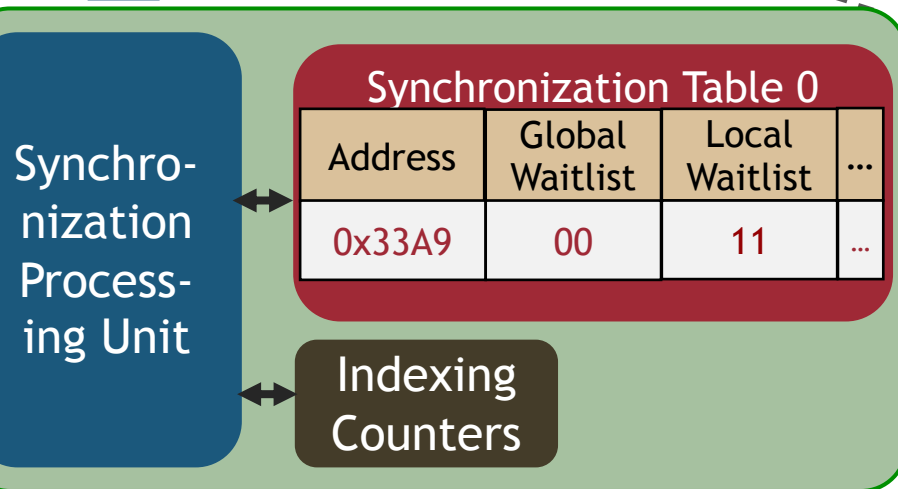
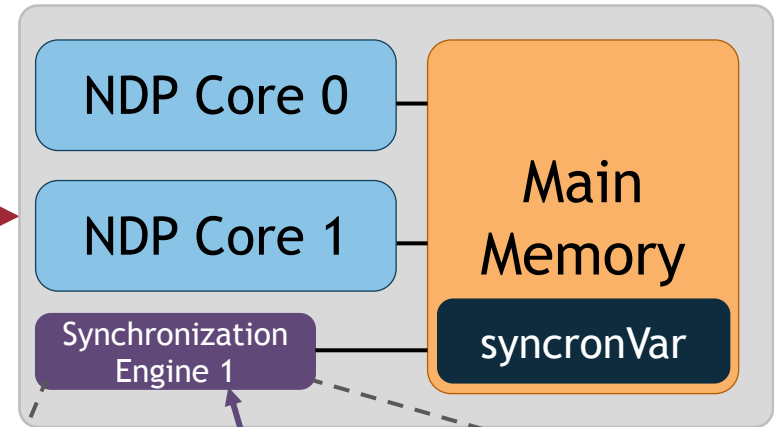


Local
lock grant

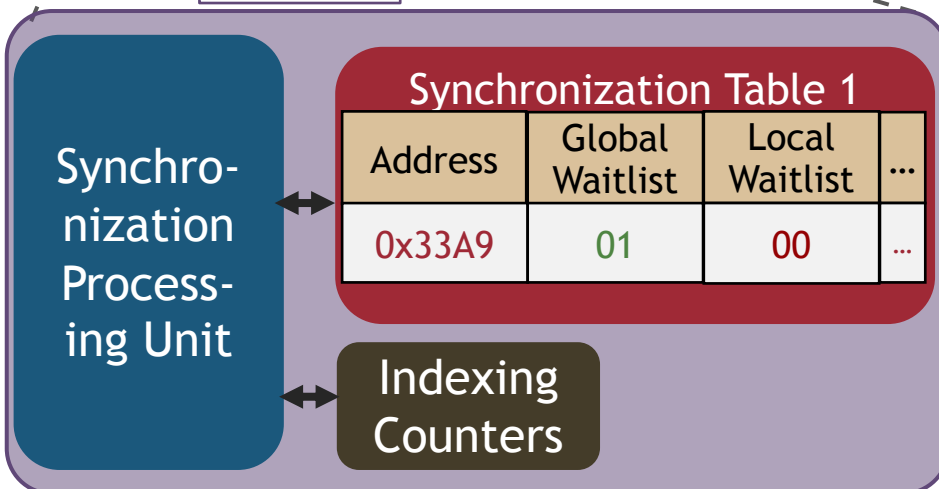
NDP Unit 0



NDP Unit 1

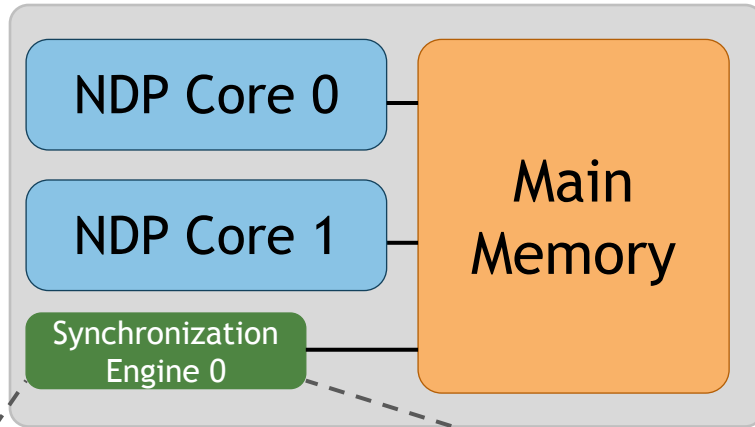


Master

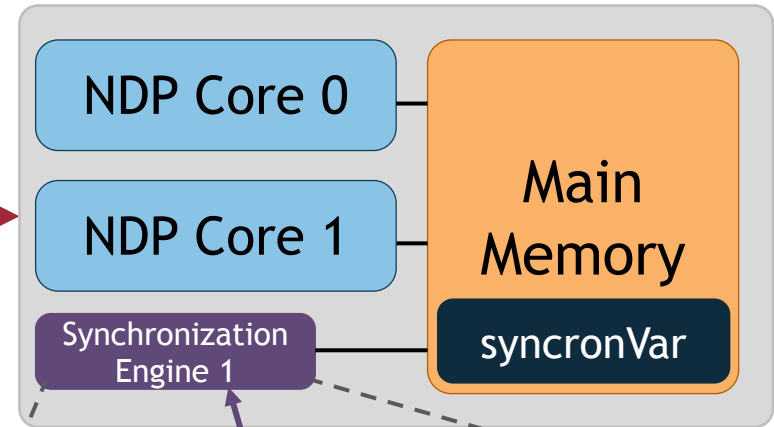


Lock Operation

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	00	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	01	00	...

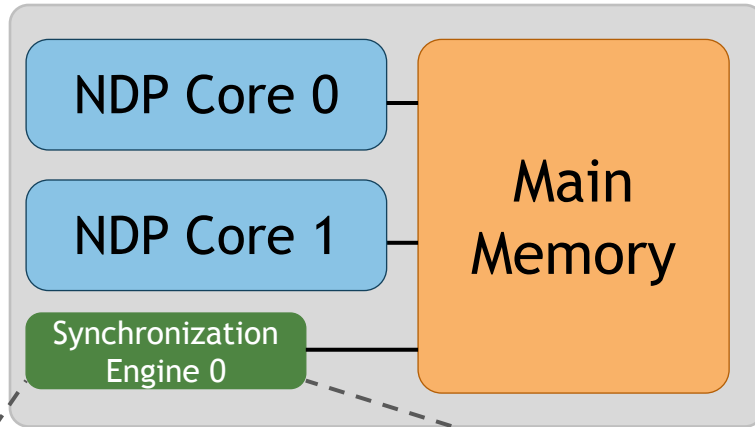
Indexing
Counters

Lock Operation

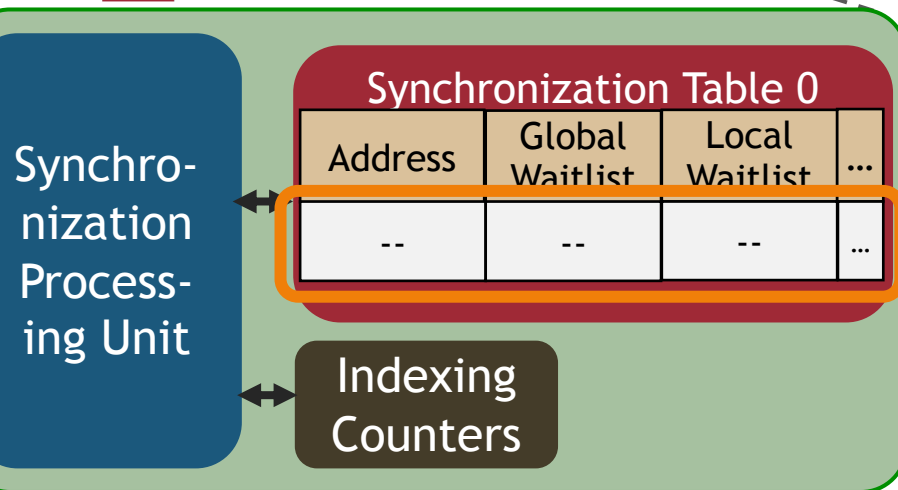
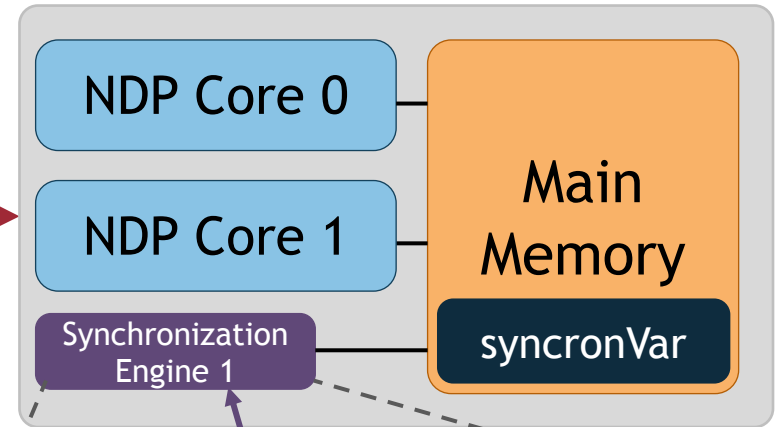


Global
lock release

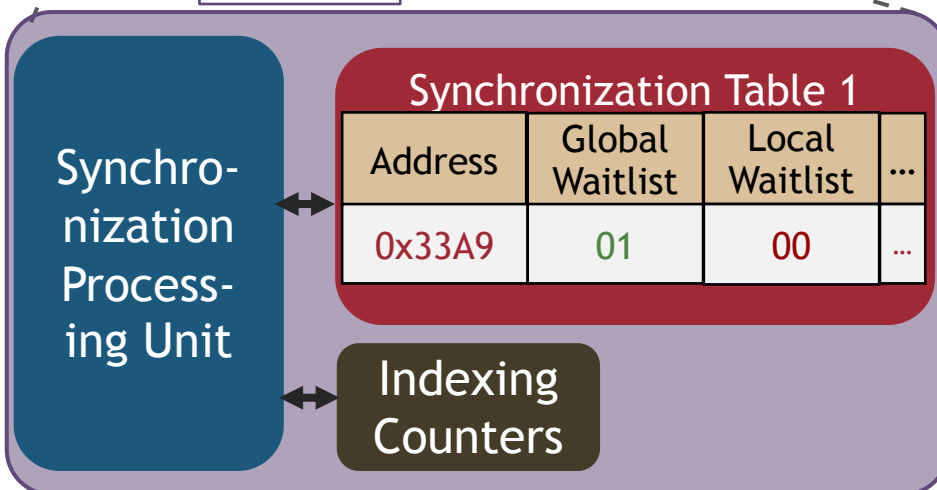
NDP Unit 0



NDP Unit 1



Master

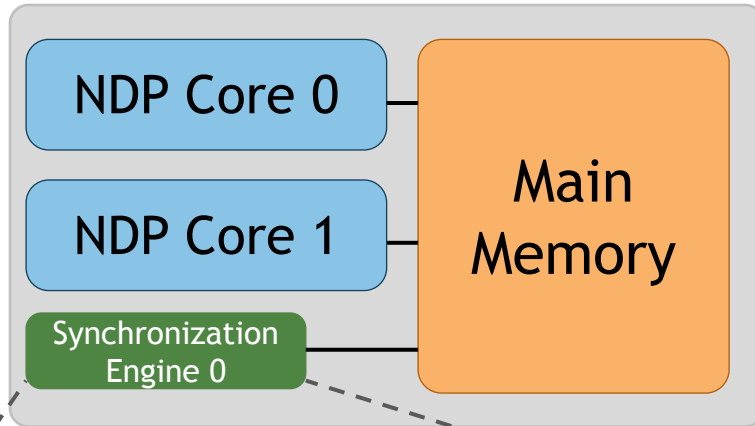


Lock Operation

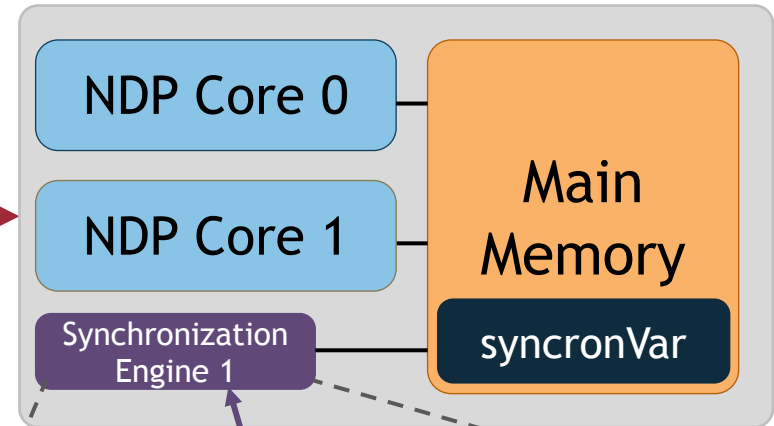


Global
lock release

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
--	--	--	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

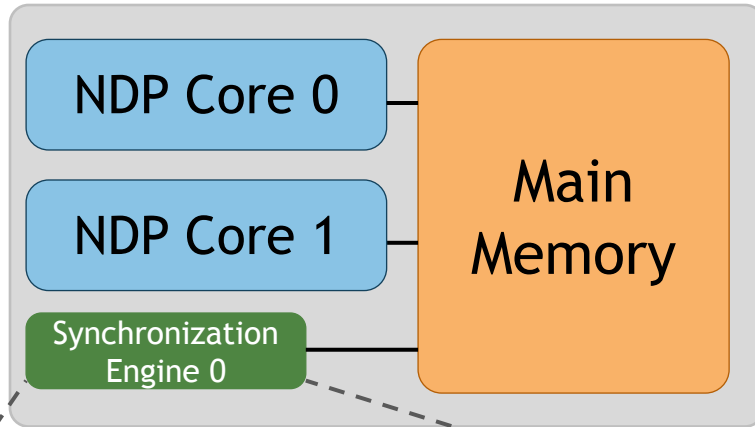
Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
0x33A9	00	00	...

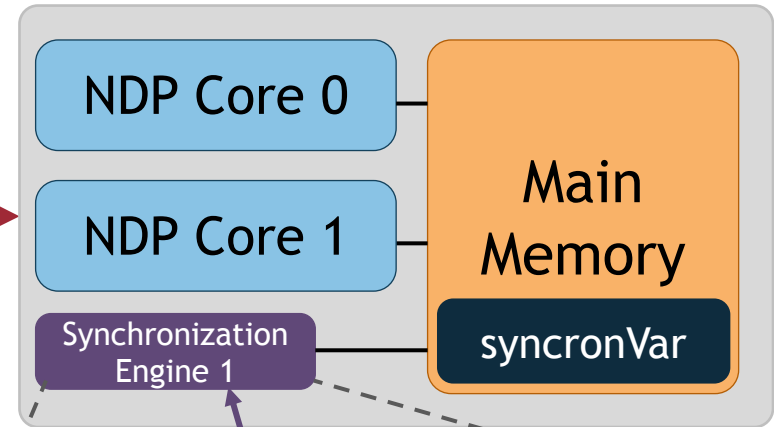
Indexing
Counters

Lock Operation

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

Address	Global Waitlist	Local Waitlist	...
--	--	--	...

Indexing
Counters

Synchro-
nization
Process-
ing Unit

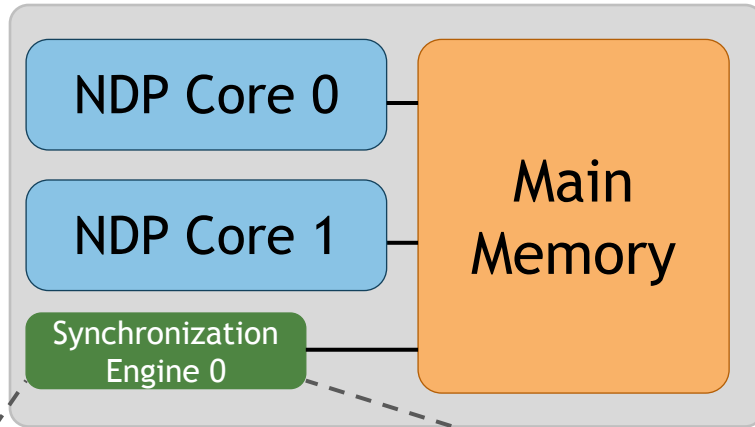
Synchronization Table 1

Address	Global Waitlist	Local Waitlist	...
--	--	--	...

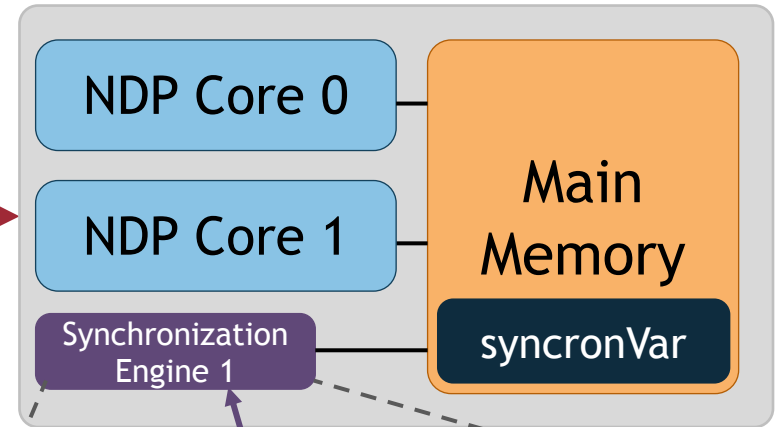
Indexing
Counters

Lock Operation - Overflow

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

FULLY OCCUPIED

Indexing
Counters

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

FULLY OCCUPIED

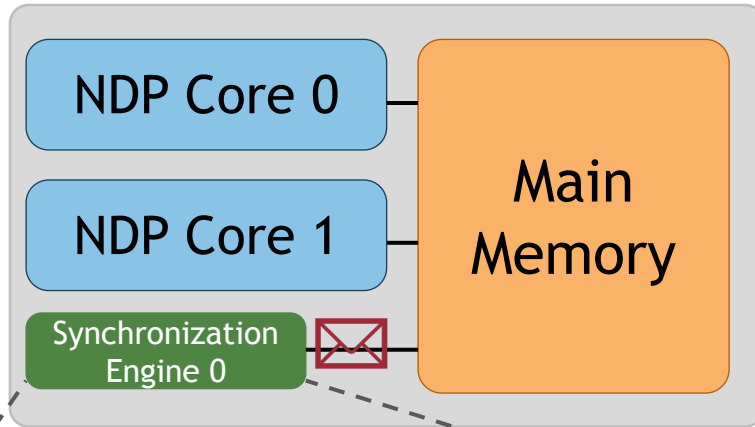
Indexing
Counters

Lock Operation - Overflow

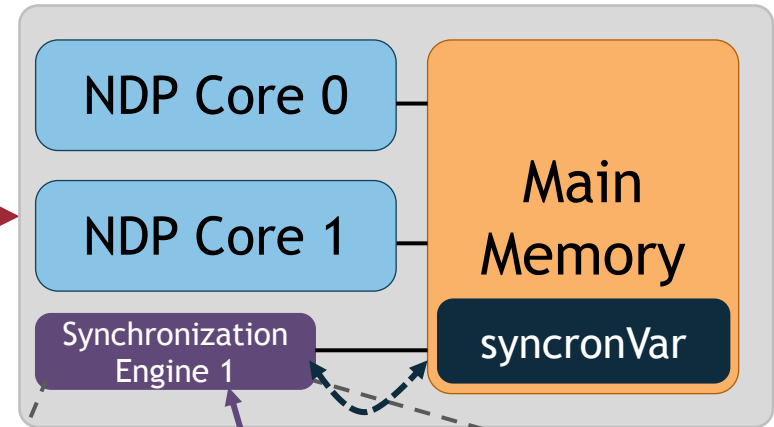


Global
overflow
lock acquire

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

FULLY OCCUPIED

Counter0 = 0

Counter1 > 0

Counter2 = 0

Counter3 = 0

Synchro-
nization
Process-
ing Unit

Synchronization Table 1

FULLY OCCUPIED

Counter0 = 0

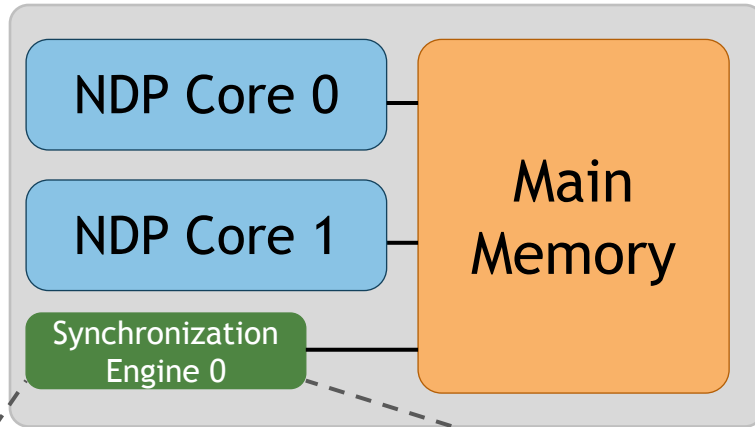
Counter1 > 0

Counter2 = 0

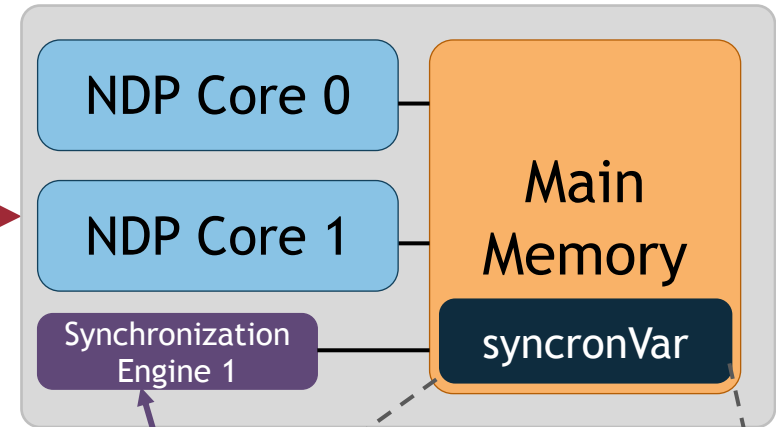
Counter3 = 0

Lock Operation - Overflow

NDP Unit 0



NDP Unit 1



Master

Synchro-
nization
Process-
ing Unit

Synchronization Table 0

FULLY OCCUPIED

Counter0 = 0

Counter1 > 0

Counter2 = 0

Counter3 = 0

```
struct synchronVar {  
    uint16_t Waitlist[2];  
    uint64_t SyncInfo;  
    uint8_t OverflowInfo;  
}
```

Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation

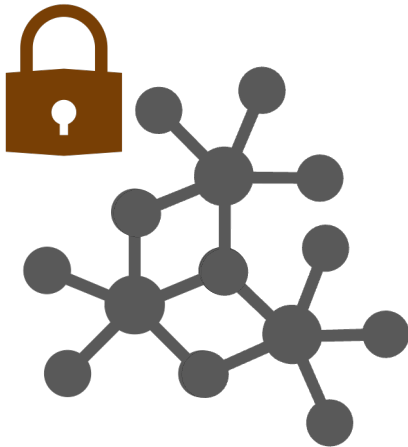
Evaluation Methodology

- 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]

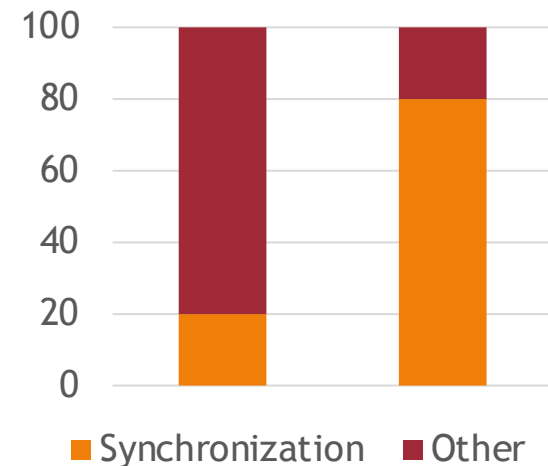
Coarse-grained



Fine-grained



Synchronization Intensity

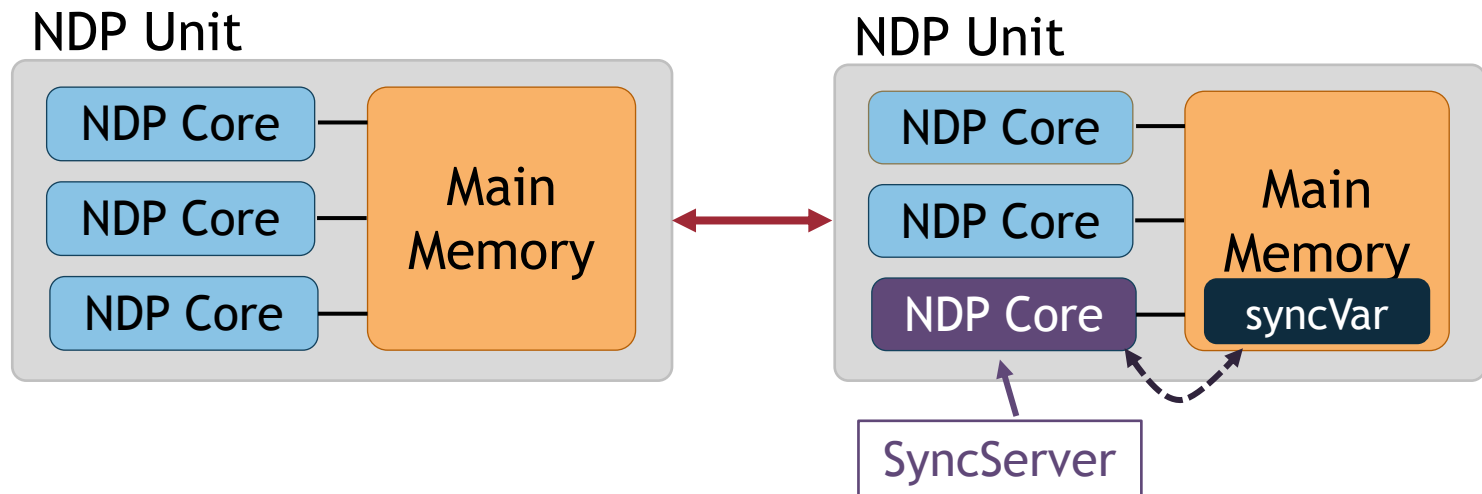


Comparison Points for SynCron

1. SynCron

2. Central [Ahn+, ISCA'15]:

- Synchronization Server: **One NDP core of the NDP system**
- **Centralized** hardware message-passing communication



Comparison Points for SynCron

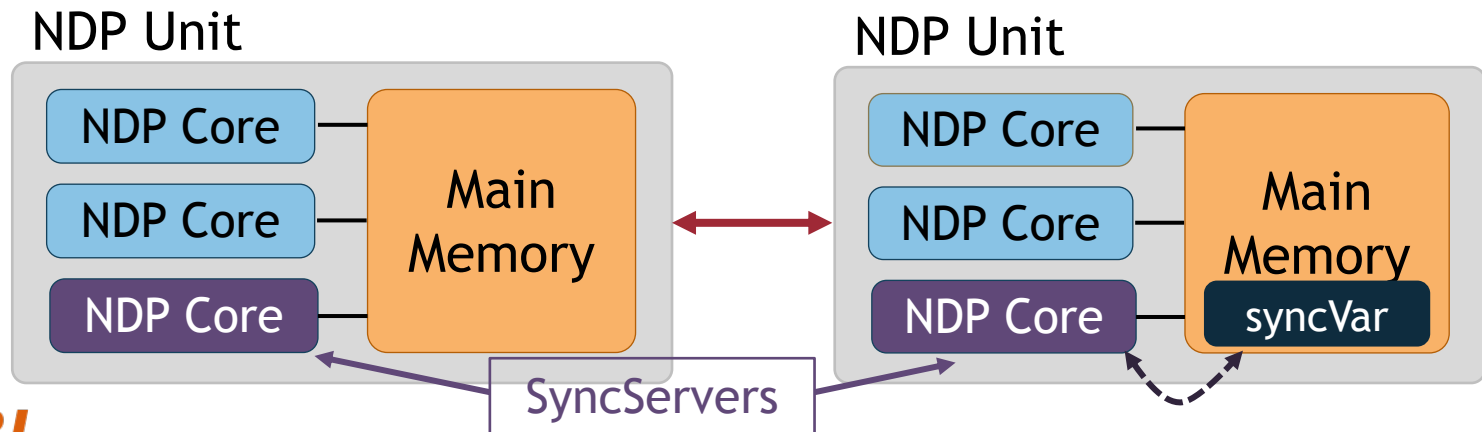
1. SynCron

2. Central [Ahn+, ISCA'15]:

- Synchronization Server: **One NDP core of the NDP system**
- **Centralized** message-passing communication

3. Hier [Gao+, PACT'15 / Tang+, ASPLOS'19]:

- Synchronization Servers: **One NDP core per NDP unit**
- **Hierarchical** message-passing communication



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]:

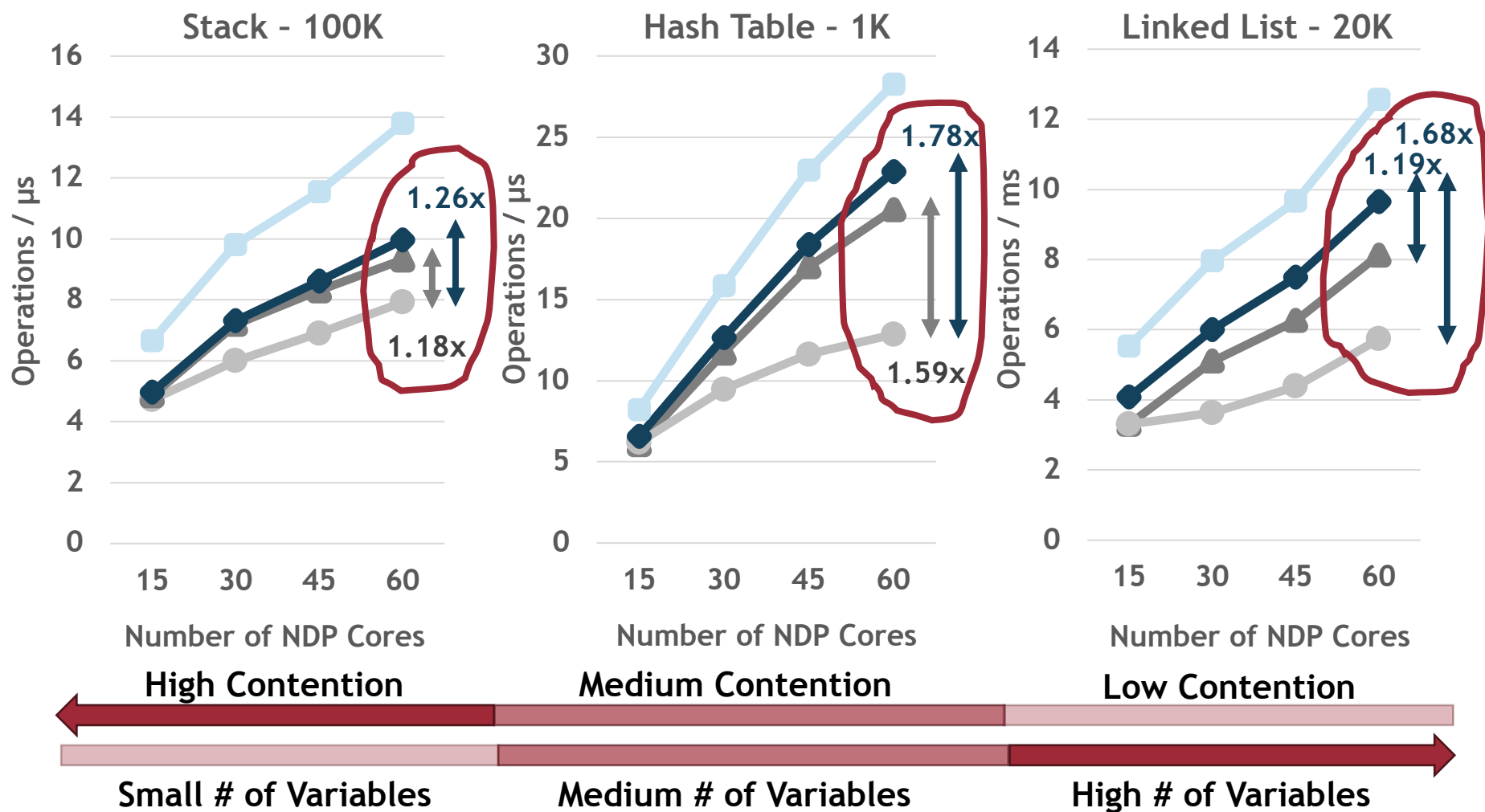
- Synchronization Servers: **One NDP core per NDP unit**
- **Hierarchical** hardware message-passing communication

4. Ideal

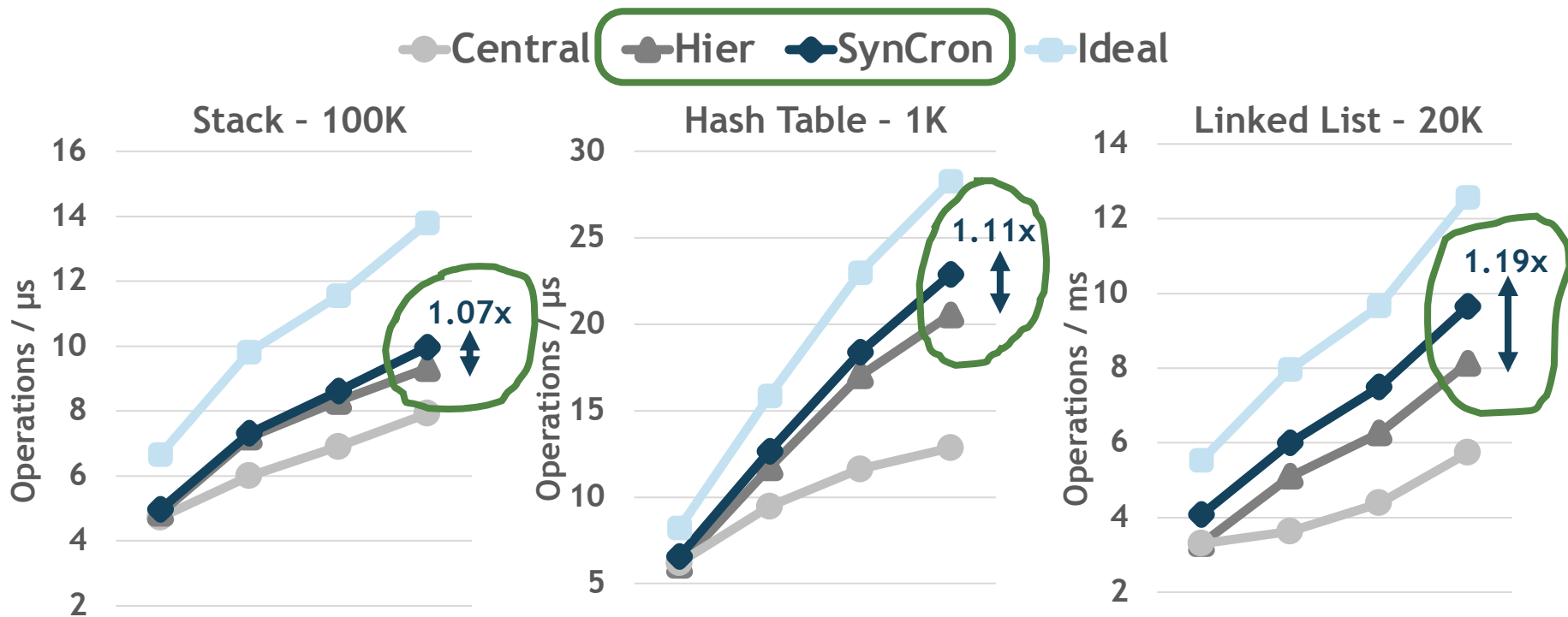
- **Zero overhead** for synchronization

Throughput of Pointer Chasing

Central Hier SynCron Ideal

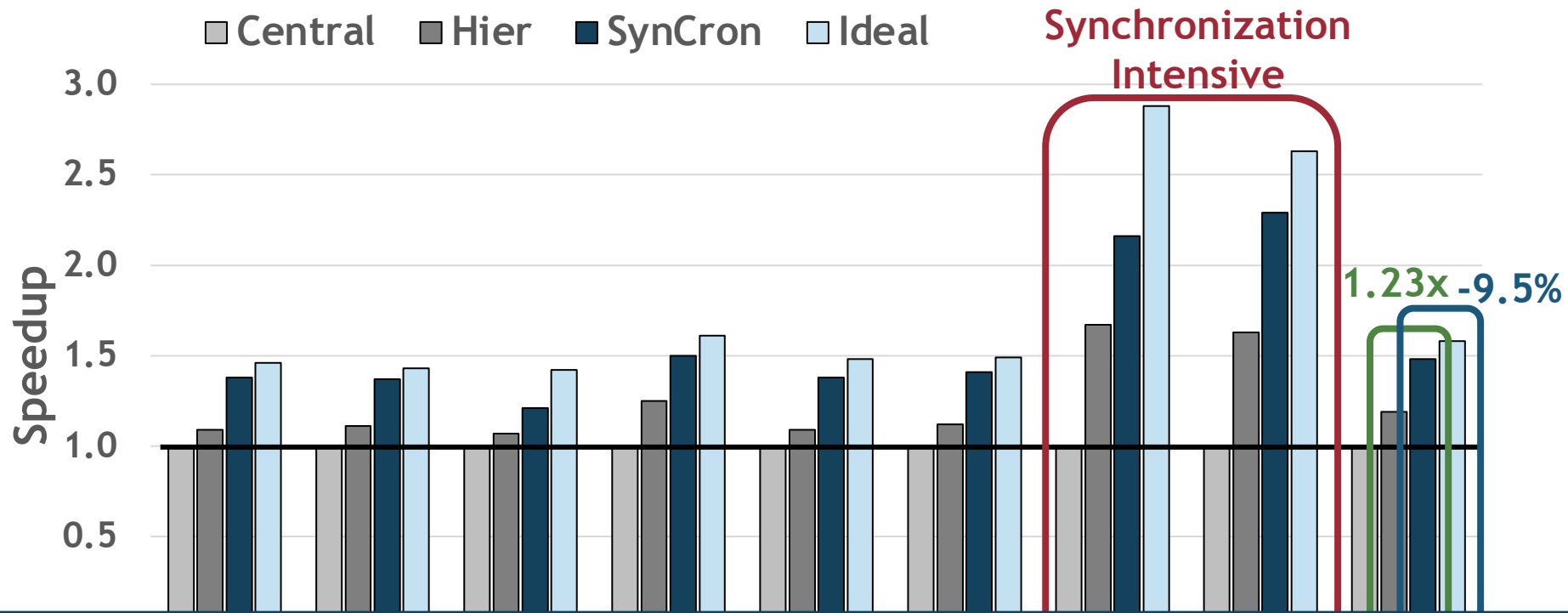


Throughput of Pointer Chasing



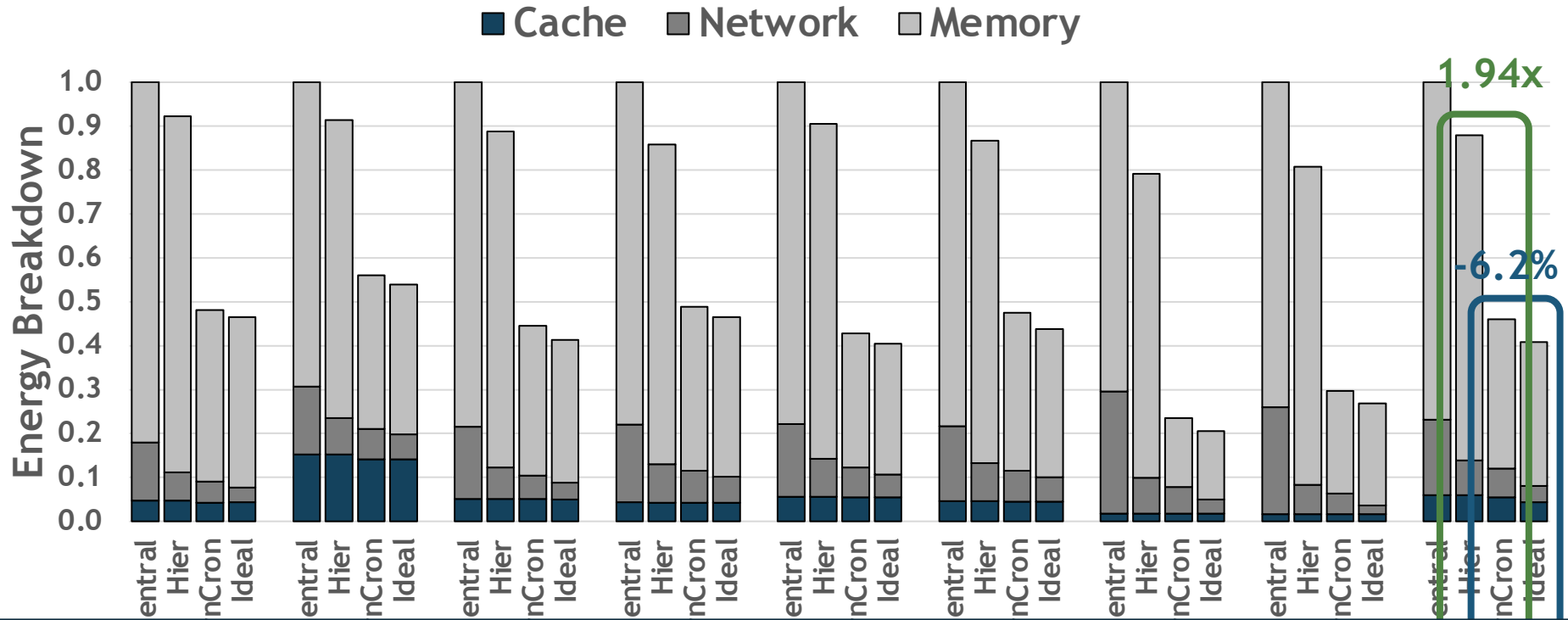
SynCron achieves the **highest throughput** under all contention scenarios

Speedup in Real Applications



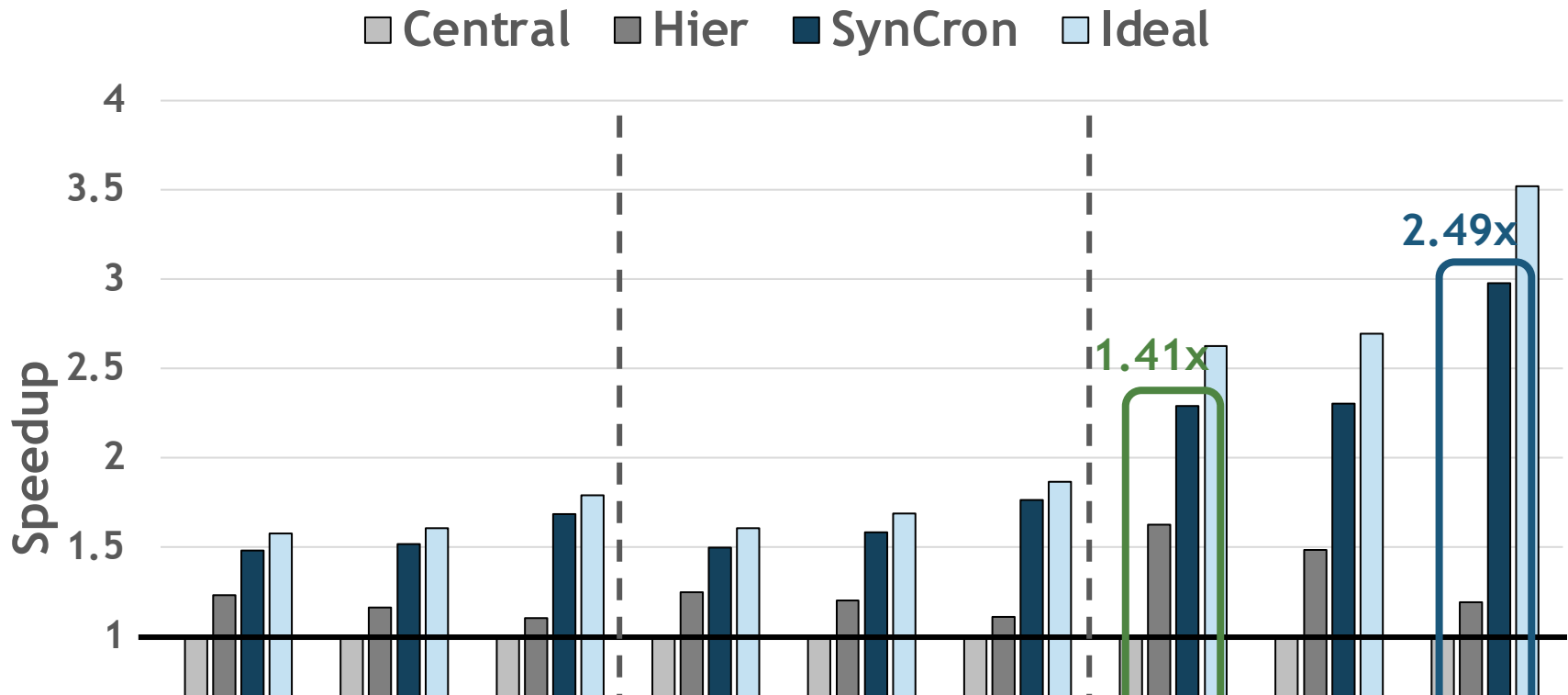
SynCron **performs best** across all **real** applications

System Energy in Real Applications



SynCron reduces system **energy significantly**

Memory technologies



SynCron is **orthogonal**
to the memory technology used

Area and Power Overheads

	Synchronization Engine	ARM Cortex A7
Technology	40nm	28nm
Area	Total: 0.0461mm ²	Total: 0.45mm ²
Power	2.7mW	100mW

SynCron has **low area** and **power** overheads

Sensitivity Studies

- 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



P&S Processing-in-Memory
14.12.2021

Christina Giannoula

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

SAFARI **ETH** zürich

P&S Processing-in-Memory

Synchronization Support for Processing-in-Memory Architectures

Dr. Juan Gómez Luna

Prof. Onur Mutlu

ETH Zürich

Fall 2021

14 December 2021