# Memory Access Scheduling to Reduce Thread Migrations

SANA DAMANI, PRITHAYAN BARUA, VIVEK SARKAR

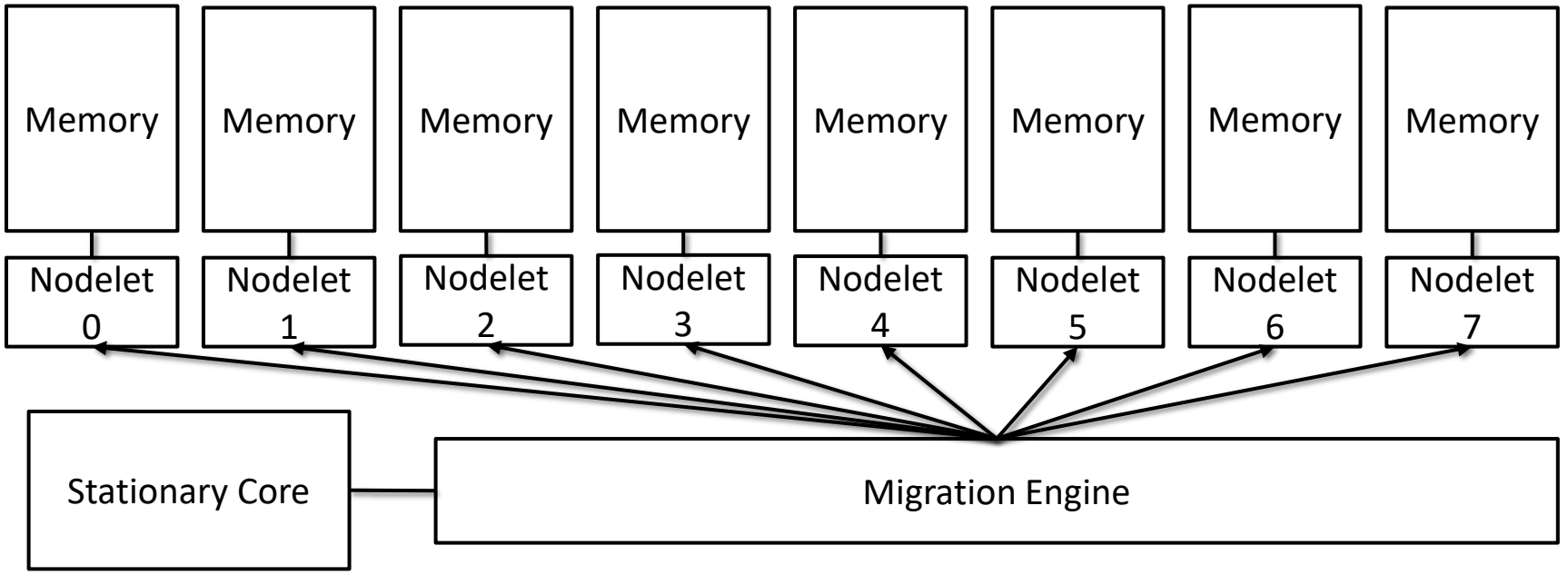
HABANERO EXTREME SCALE SOFTWARE RESEARCH LABORATORY

GEORGIA INSTITUTE OF TECHNOLOGY

HTTP://HABANERO.CC.GATECH.EDU

# EMU Architecture

Migrate small thread context instead of data.




**EMU NODE**

http://lucata.com

Applications with weak data locality:

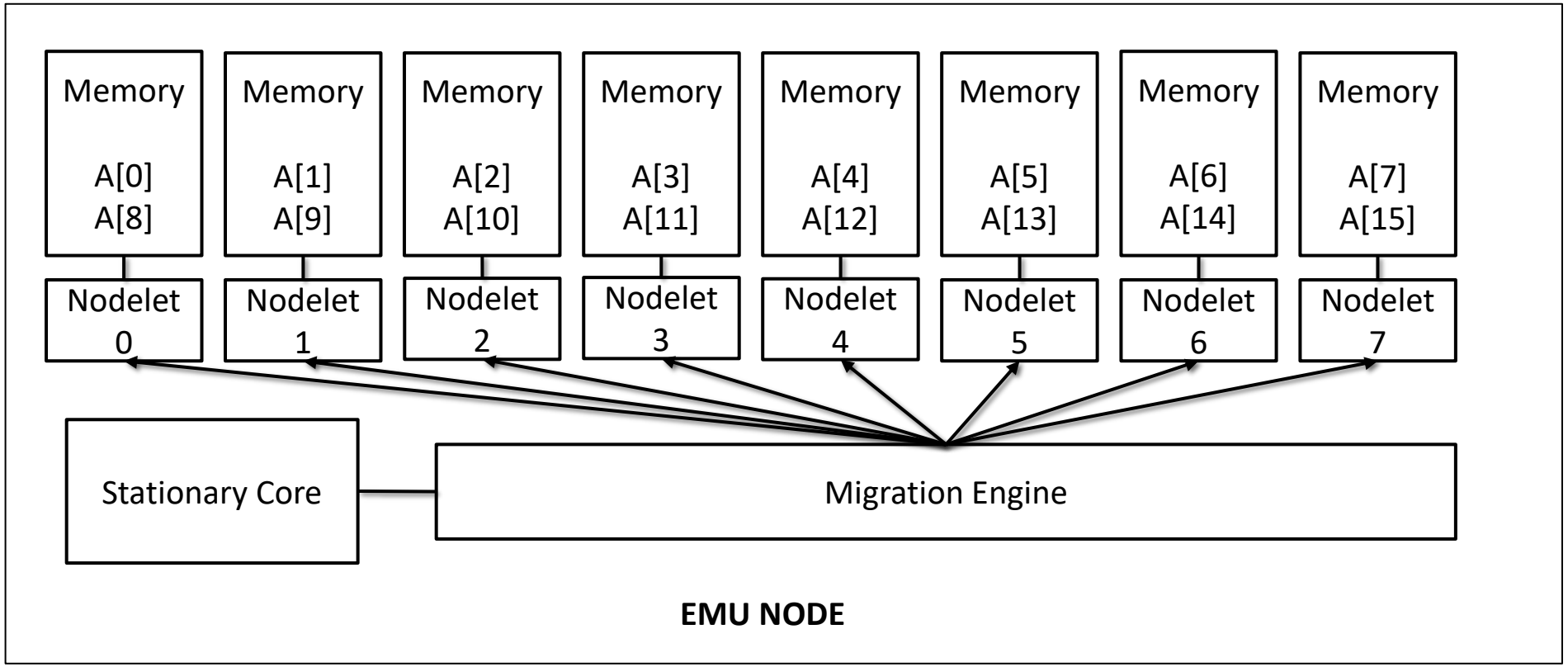
- Graph algorithms
- Sparse matrix applications

# Suboptimal Thread Migrations

S1: LD A[1]

S2: LD A[2]

S3: LD A[9]

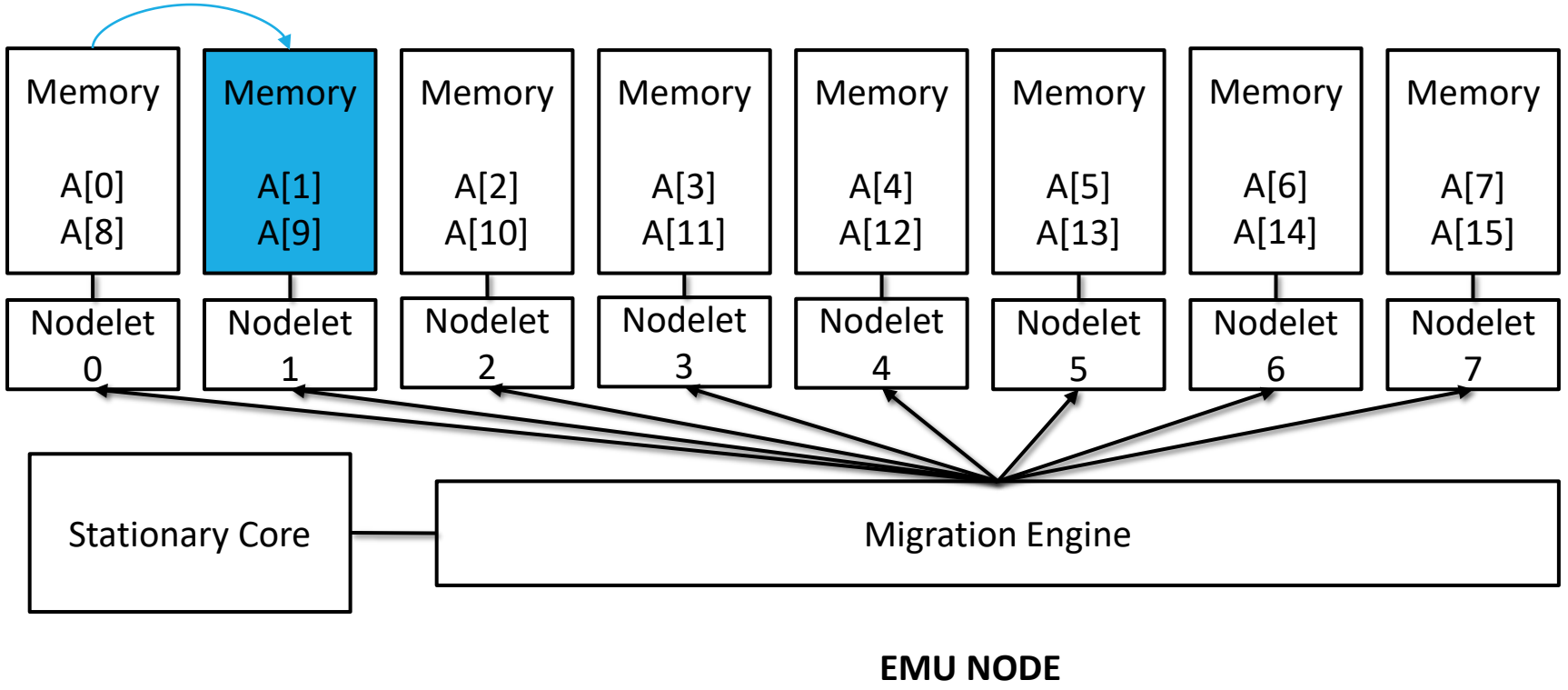


# Suboptimal Thread Migrations

S1: LD A[1]

S2: LD A[2]

S3: LD A[9]

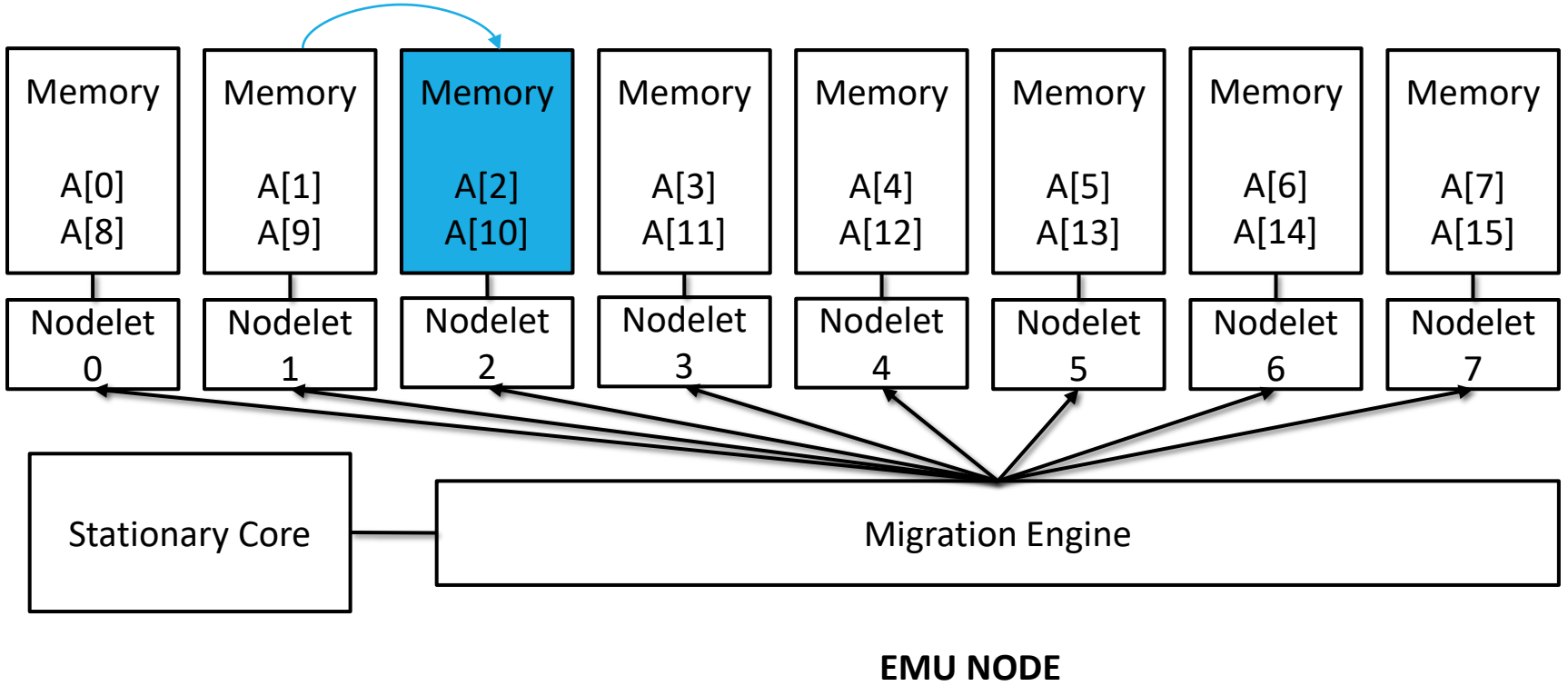


# Suboptimal Thread Migrations

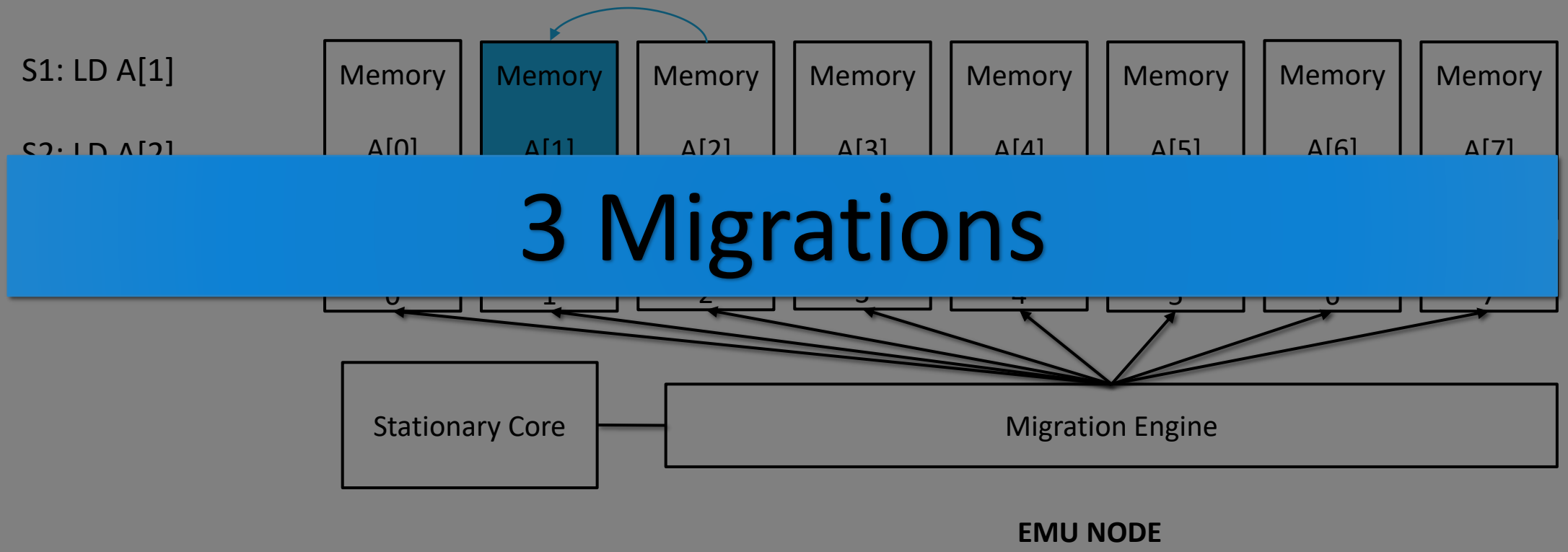
S1: LD A[1]

S2: LD A[2]

S3: LD A[9]



# Redundant Thread Migrations

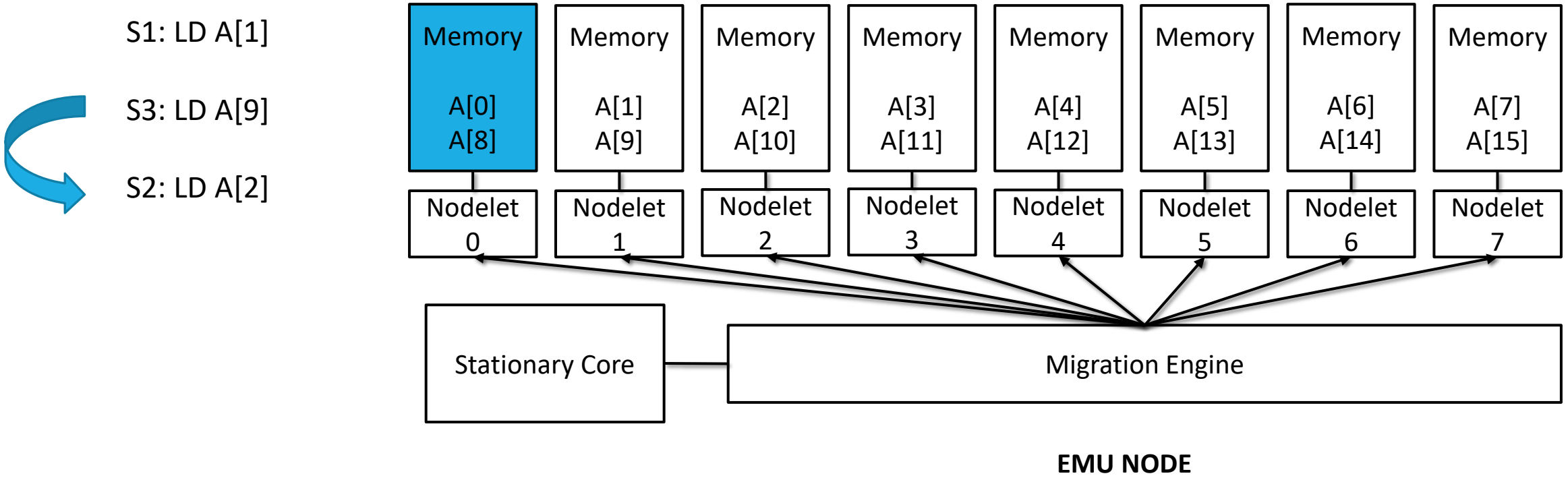


# Memory Access Scheduling

S1: LD A[1]

S3: LD A[9]

S2: LD A[2]

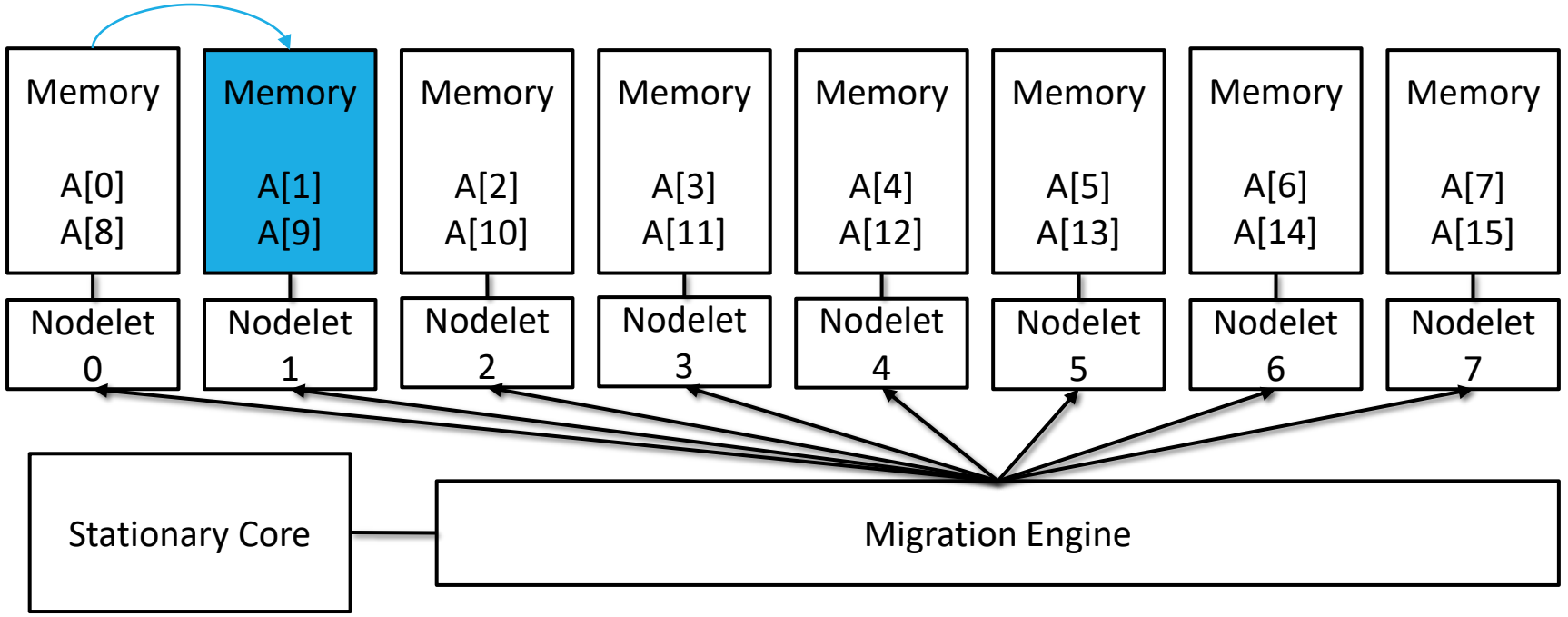


# Memory Access Scheduling

S1: LD A[1]

S3: LD A[9]

S2: LD A[2]

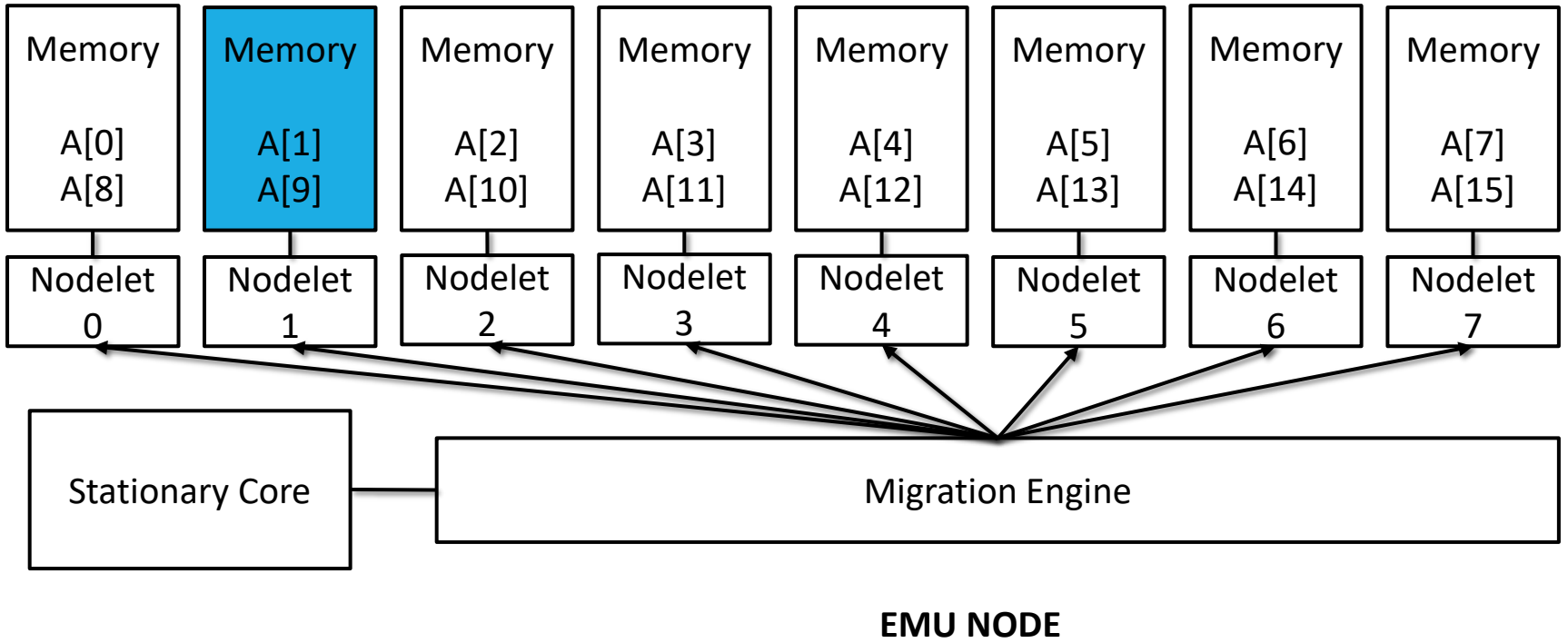


# Memory Access Scheduling

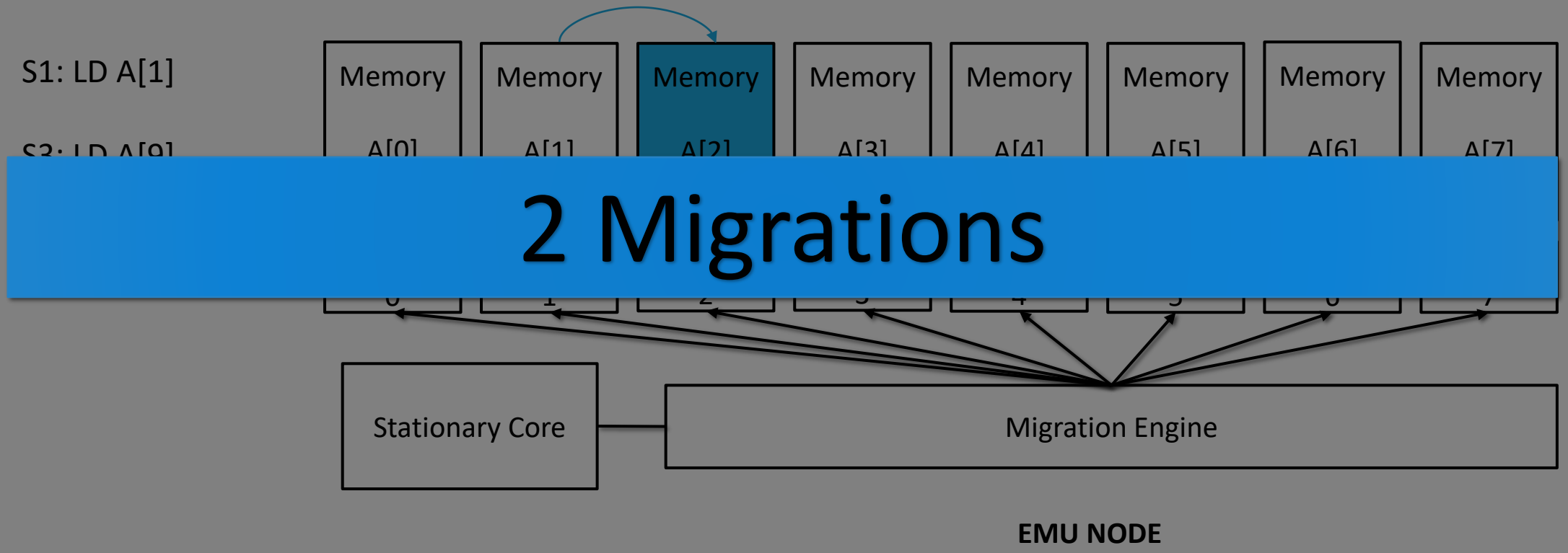
S1: LD A[1]

S3: LD A[9]

S2: LD A[2]



# Memory Access Scheduling



# Memory Access Scheduling

Migrating threads instead of data helps applications with weak data locality

Redundant thread migrations result in slowdowns

Memory access scheduling:

- Identify and group co-located accesses to reduce thread migrations
- Maintain dependences

# Design: Analysis

Goal: Identify co-located accesses

- Layout analysis:
  - A: 1D
- Stride analysis:
  - A[1], A[9]: nodelet 1
  - A[2]: nodelet 2
- Dependence graph

# Design: Transform

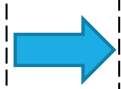
Goal: Memory Access Scheduling to group co-located accesses

- Integer Linear Programming
- Greedy Scheduler

# Interaction with Loop Unrolling

```
for (int i = 0; i < N; i++)
{
      for (int j = 0; j < N; j++)
      {
            S1: LD A[i][j]   // N0
            S2: LD B[j]      // N1
      }
}
```

```
for (int i = 0; i < N; i++)
{
      for (int j = 0; j < N; j+=2)
      {
            LD A[i][j]     // N0
            LD B[j]        // N1
            LD A[i][j+1]   // N0
            LD B[j+1]      // N2
      }
}
```



```
for (int i = 0; i < N; i++)
{
      for (int j = 0; j < N; j+=2)
      {
            LD A[i][j]     // N0
            LD A[i][j+1]   // N0
            LD B[j]        // N1
            LD B[j+1]      // N2
      }
}
```

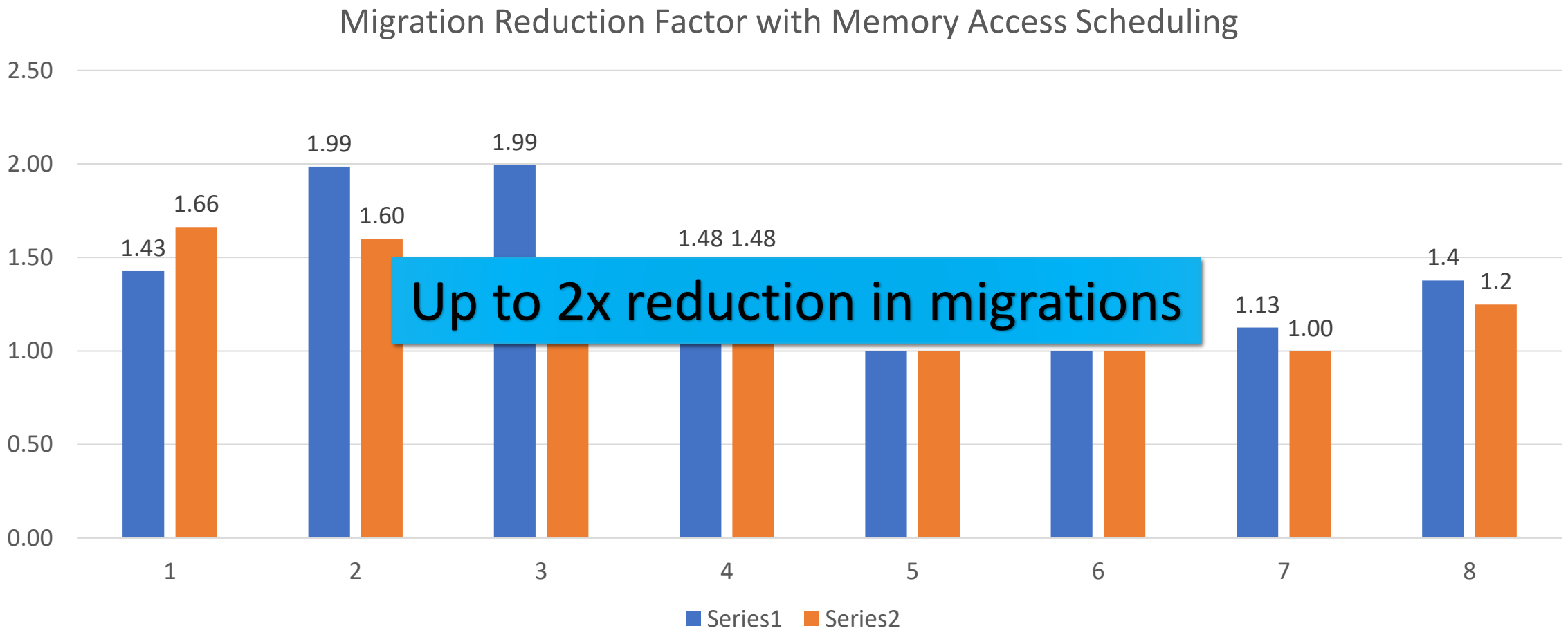
Base
2N*N

Loop Unroll
2N*N

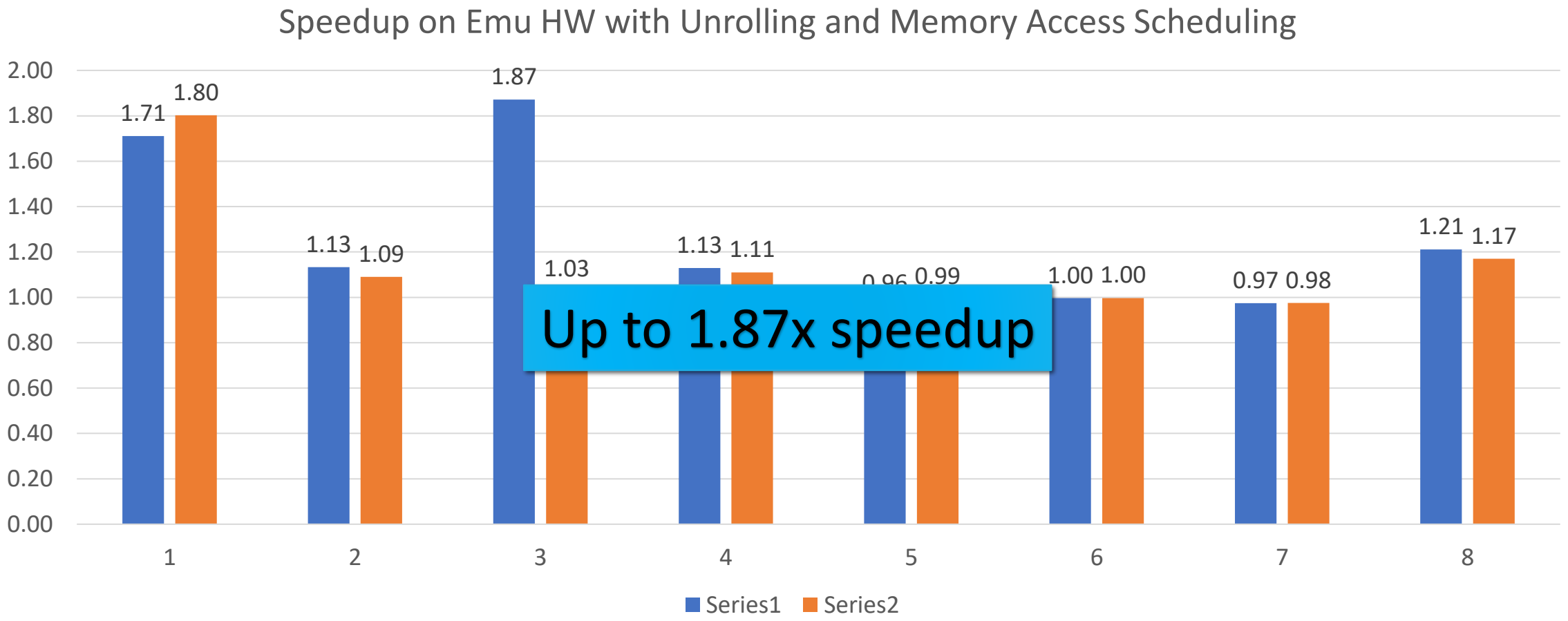
Memory Access Scheduling
3/2*N*N

# Experimental Setup

| | |
|---|---|
| **Performance Metric** | Thread Migrations, Speedup |
| **EMU Configuration** | 1 Node, 8 Nodelets |
| **Per-Thread Registers** | 16 |
| **Compiler** | LLVM-Cilk |
| **Migration Profiling** | emusim |
| **Applications** | Linear Algebra, Polybench |
| **Unroll factor** | 2 |




# Reduction in Migrations


Speedup on Emu HW with Unrolling and Memory Access Scheduling
2.00
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
1.71
1.80
1.13
1.09
1.87
1.03
1.13
1.11
0.96
0.99
1.00
1.00
0.97
0.98
1.21
1.17
Up to 1.87x speedup
1
2
3
4
5
6
7
8
Series1
Series2

# Speedup

# Conclusion

| Benefits | Cost & Challenges |
| --- | --- |
| • Identify & group co-located accesses<br>• Reduce thread migrations | • Register pressure, spills<br>• Compile time overhead<br>• Indirect accesses, pointers |

# Thank You

CONTACT: SDAMANI@GATECH.EDU