OCC Pinning: Optimizing Concurrent Computations through Thread Pinning

Brahmaiah Gandham **Dr. Praveen Alapati**

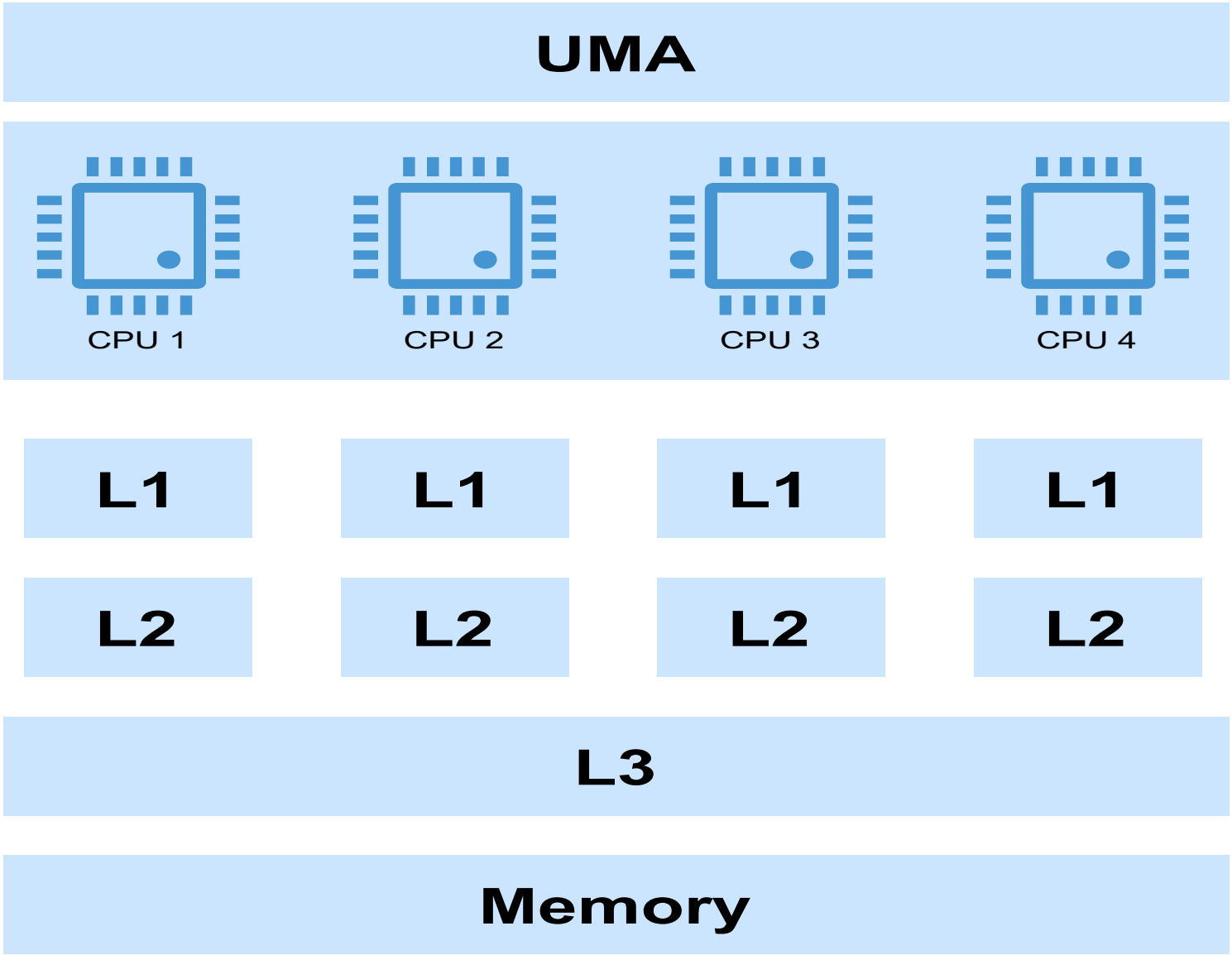
Department of Computer Science and Engineering

Ecole Centrale School of Engineering

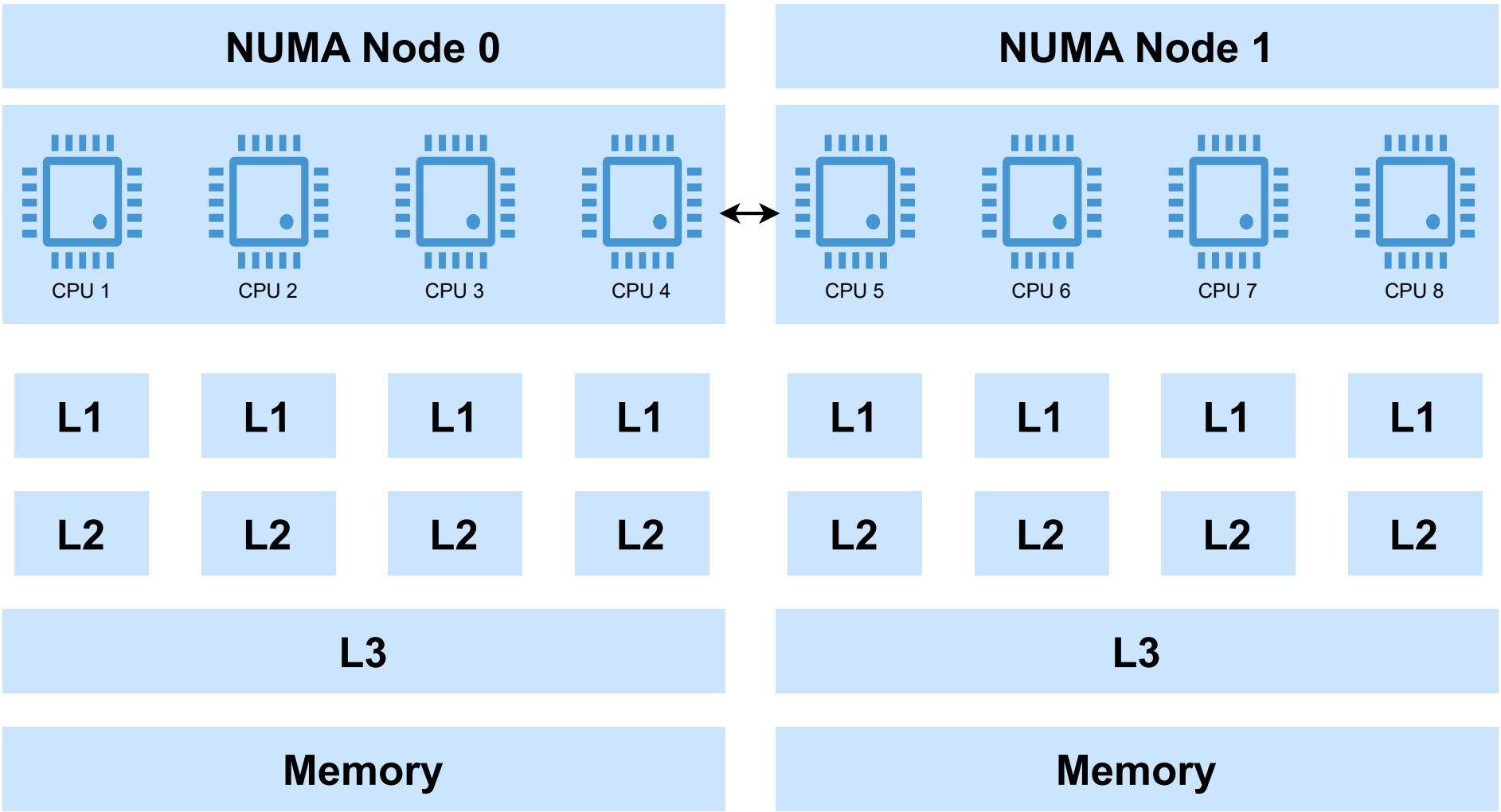
Mahindra University

Outline

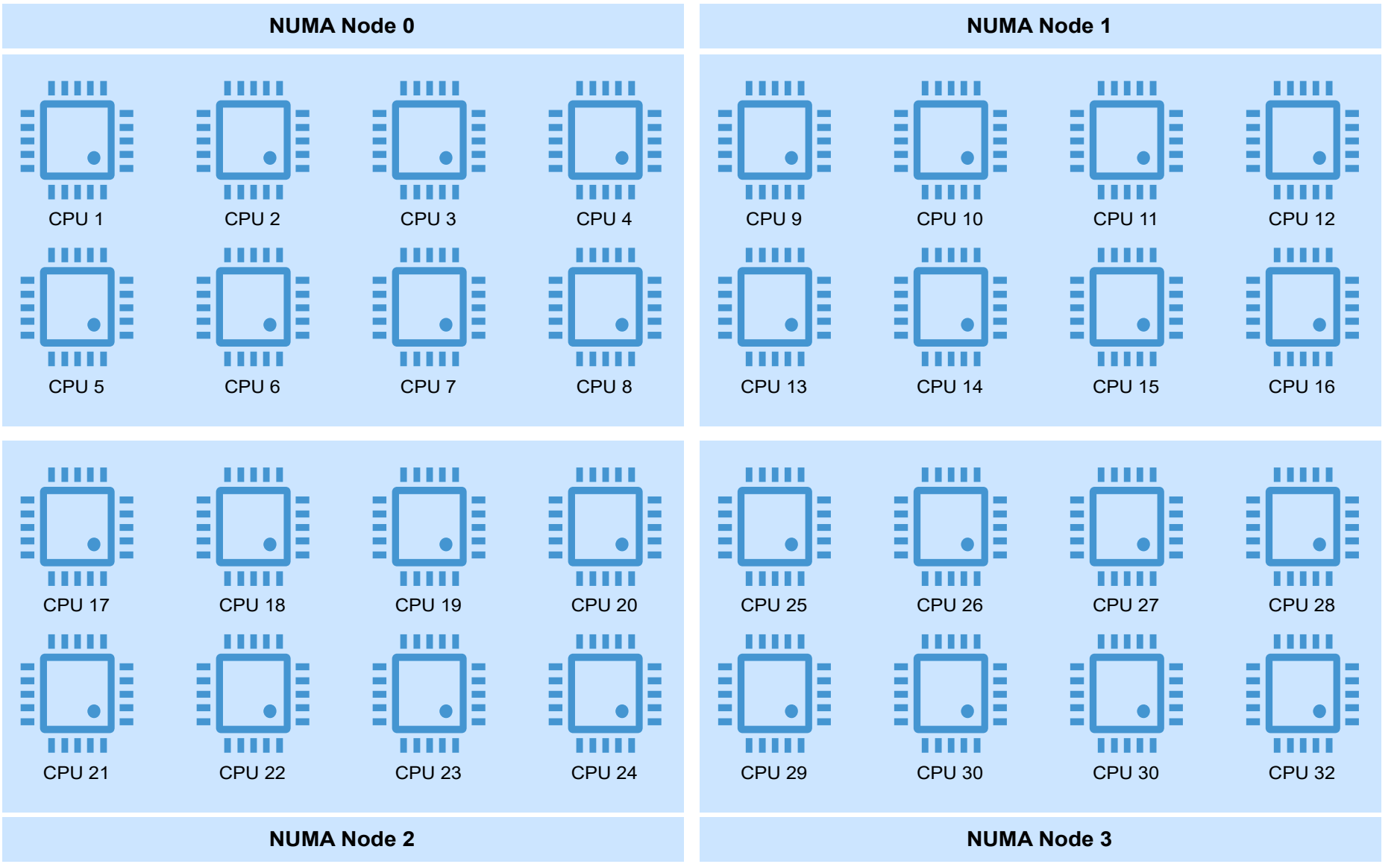
Uniform Memory Access (UMA) Architecture



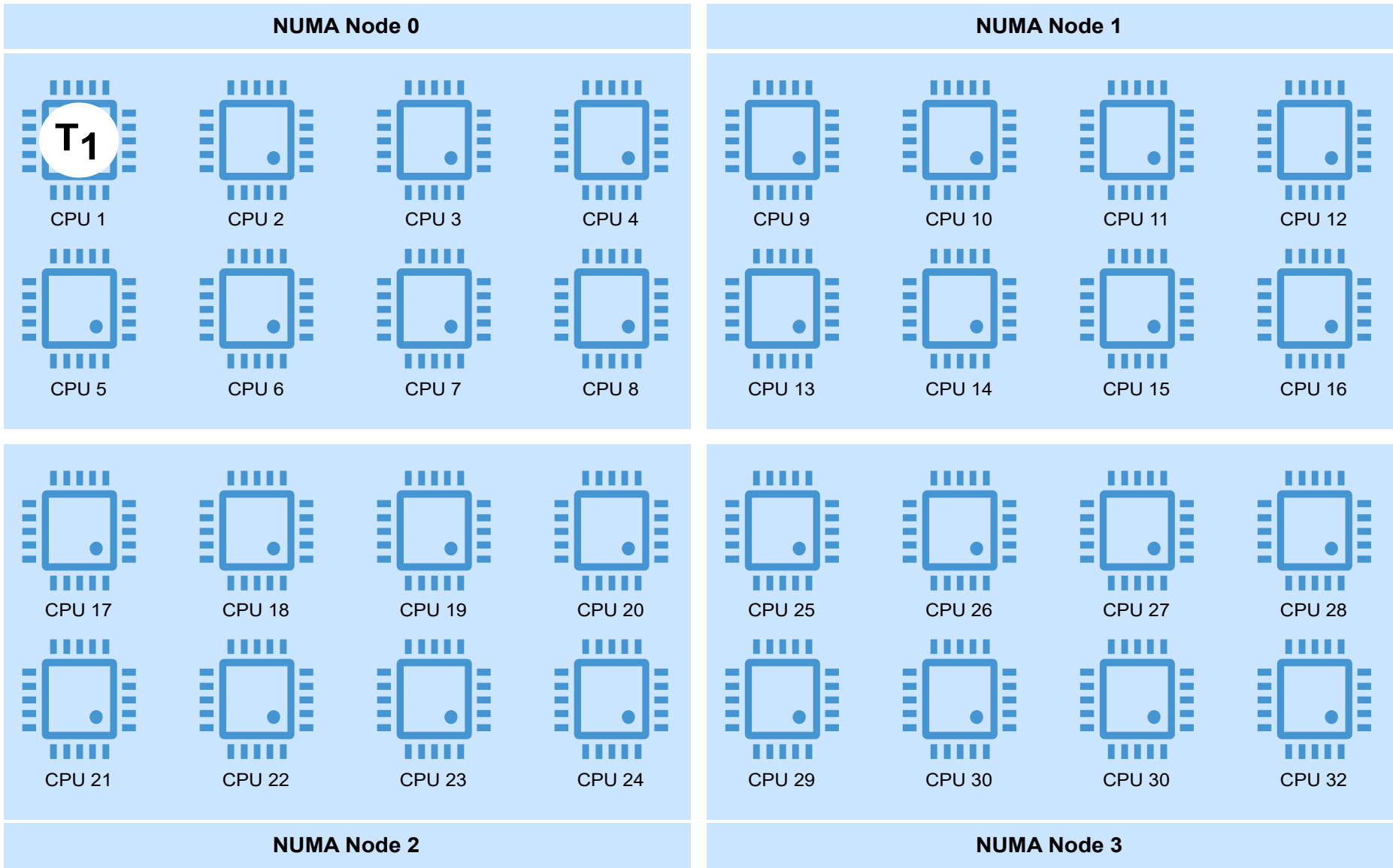
Non-uniform Memory Access (NUMA) Architecture



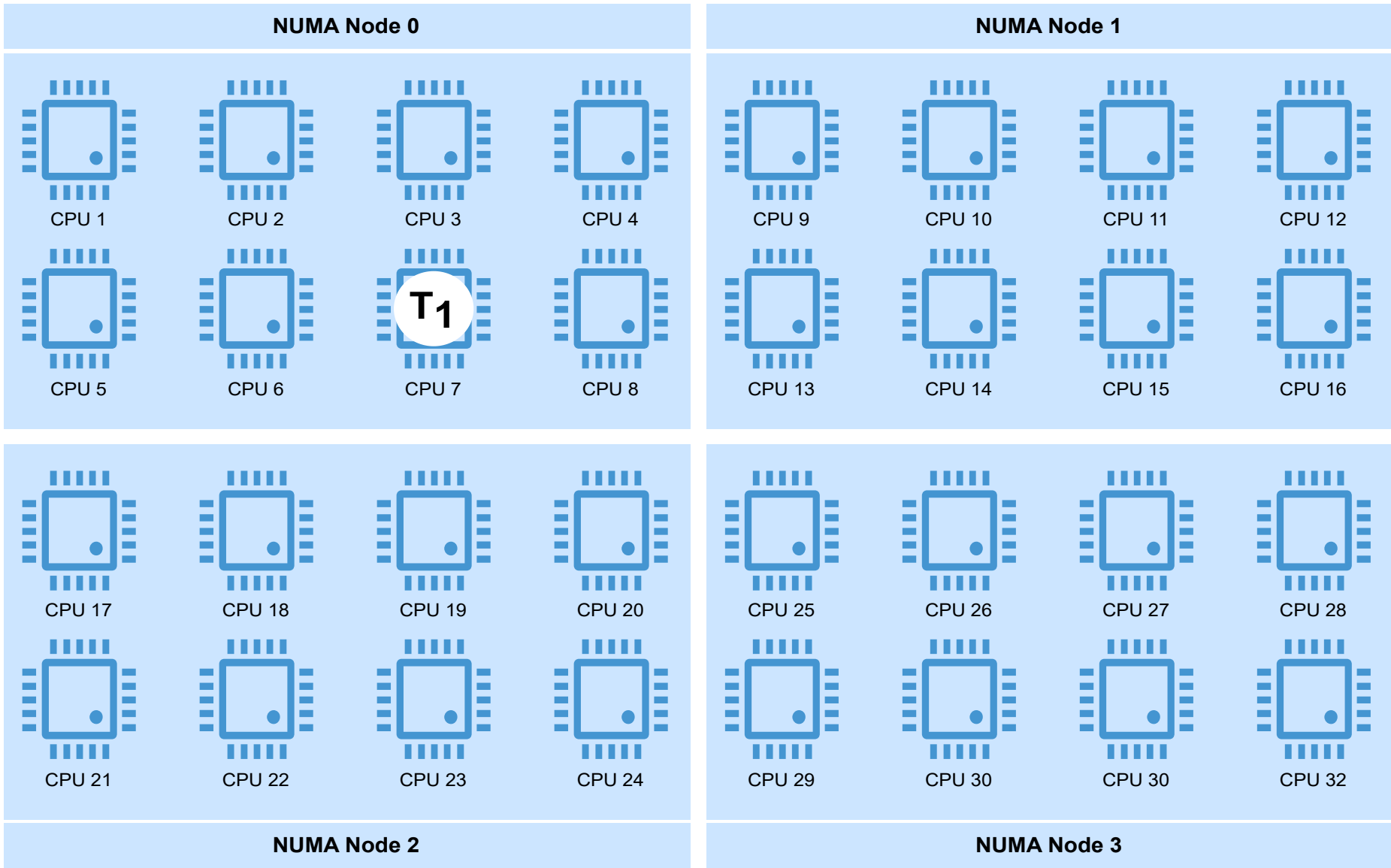
32 - Core Server



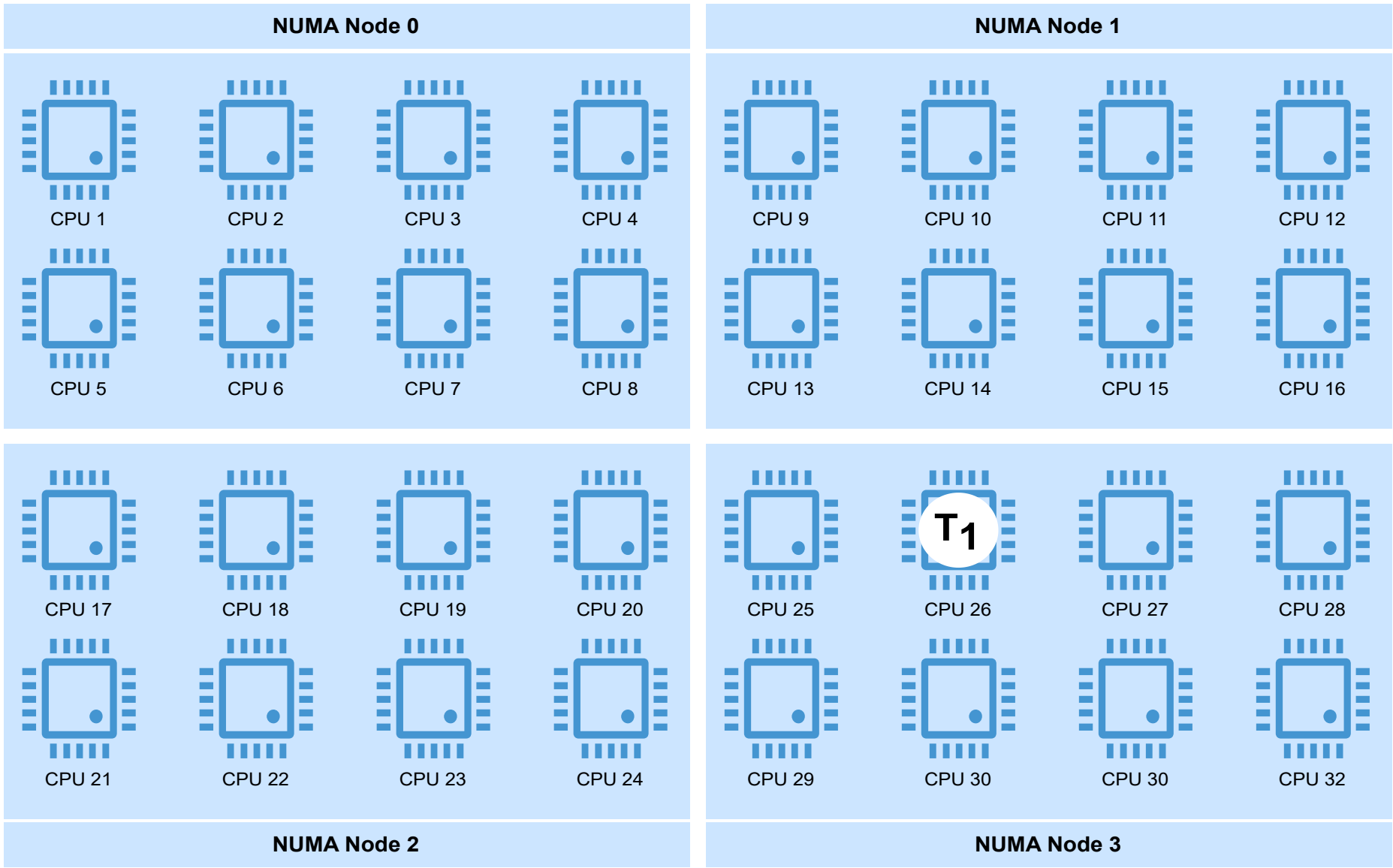
Execution of a Sequential Program



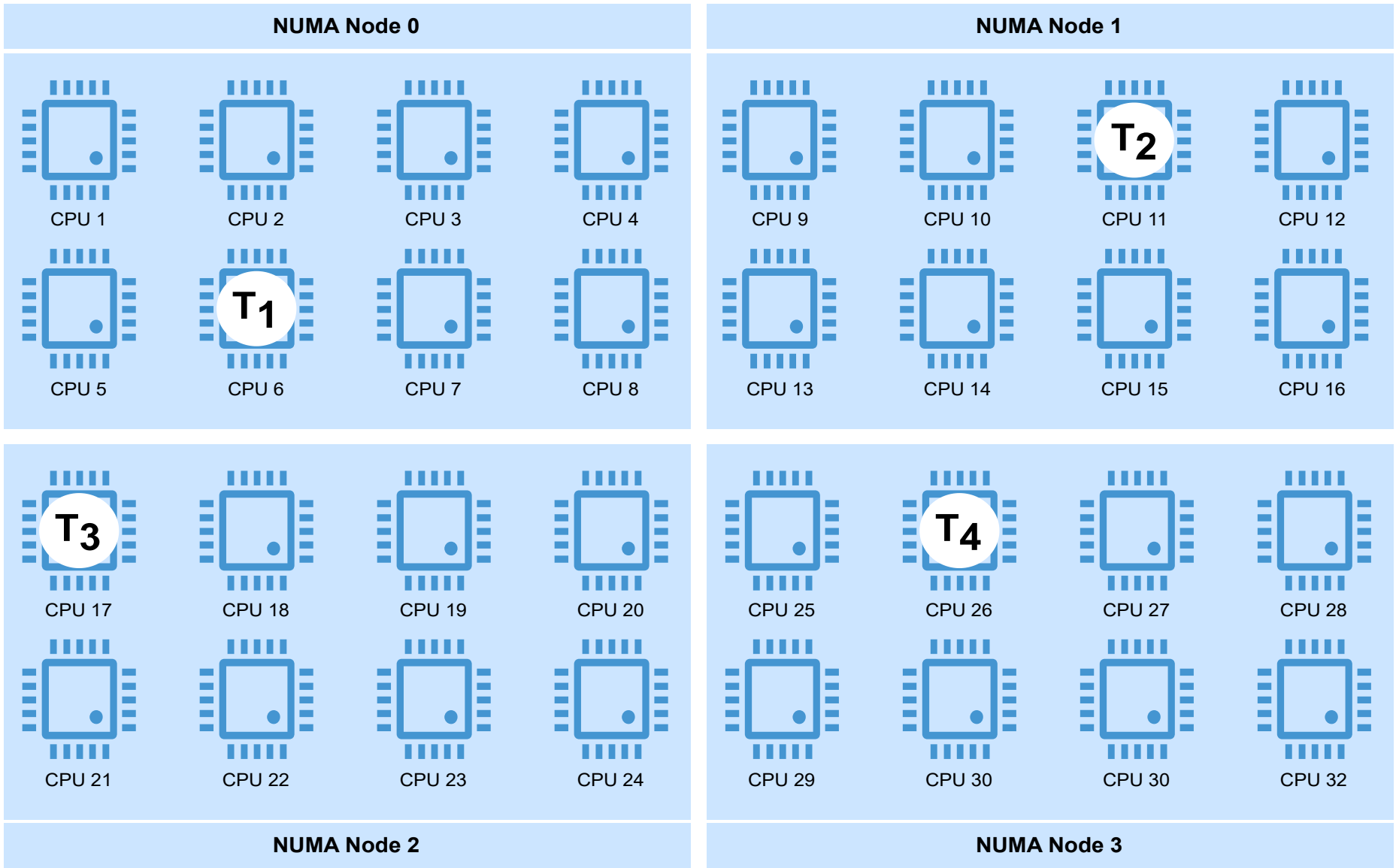
Execution of a Sequential Program



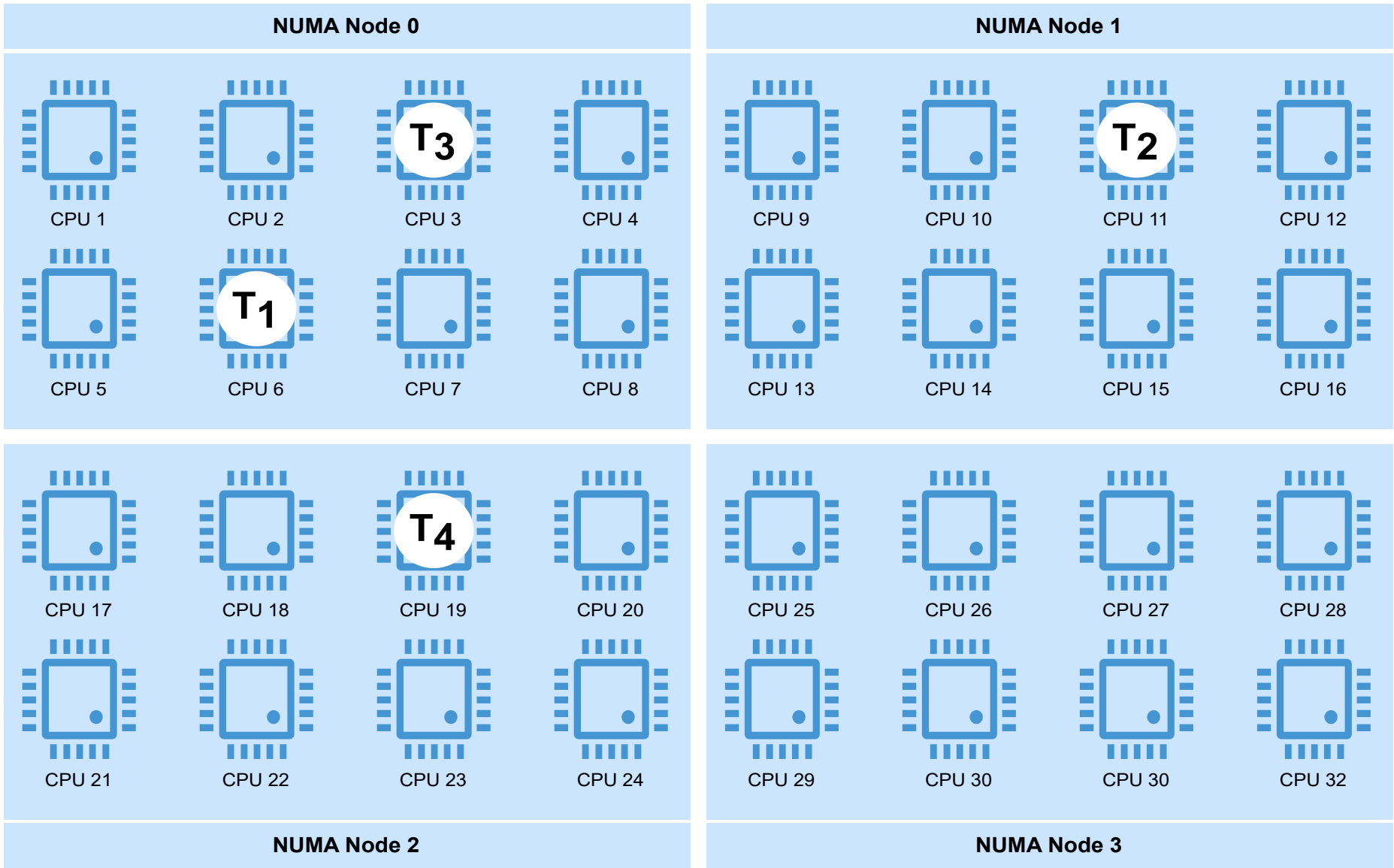
Execution of a Sequential Program



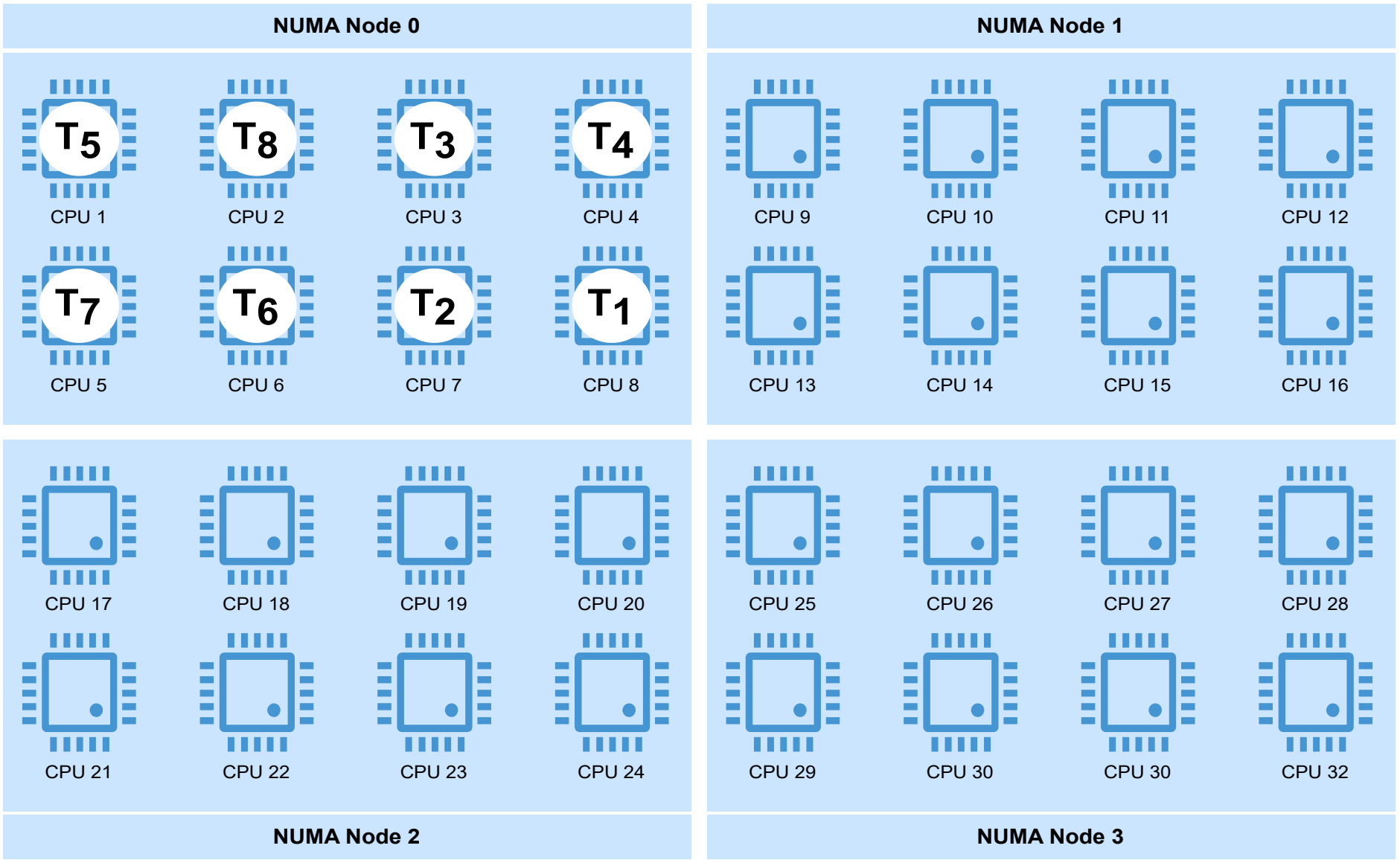
Execution of a Concurrent Program



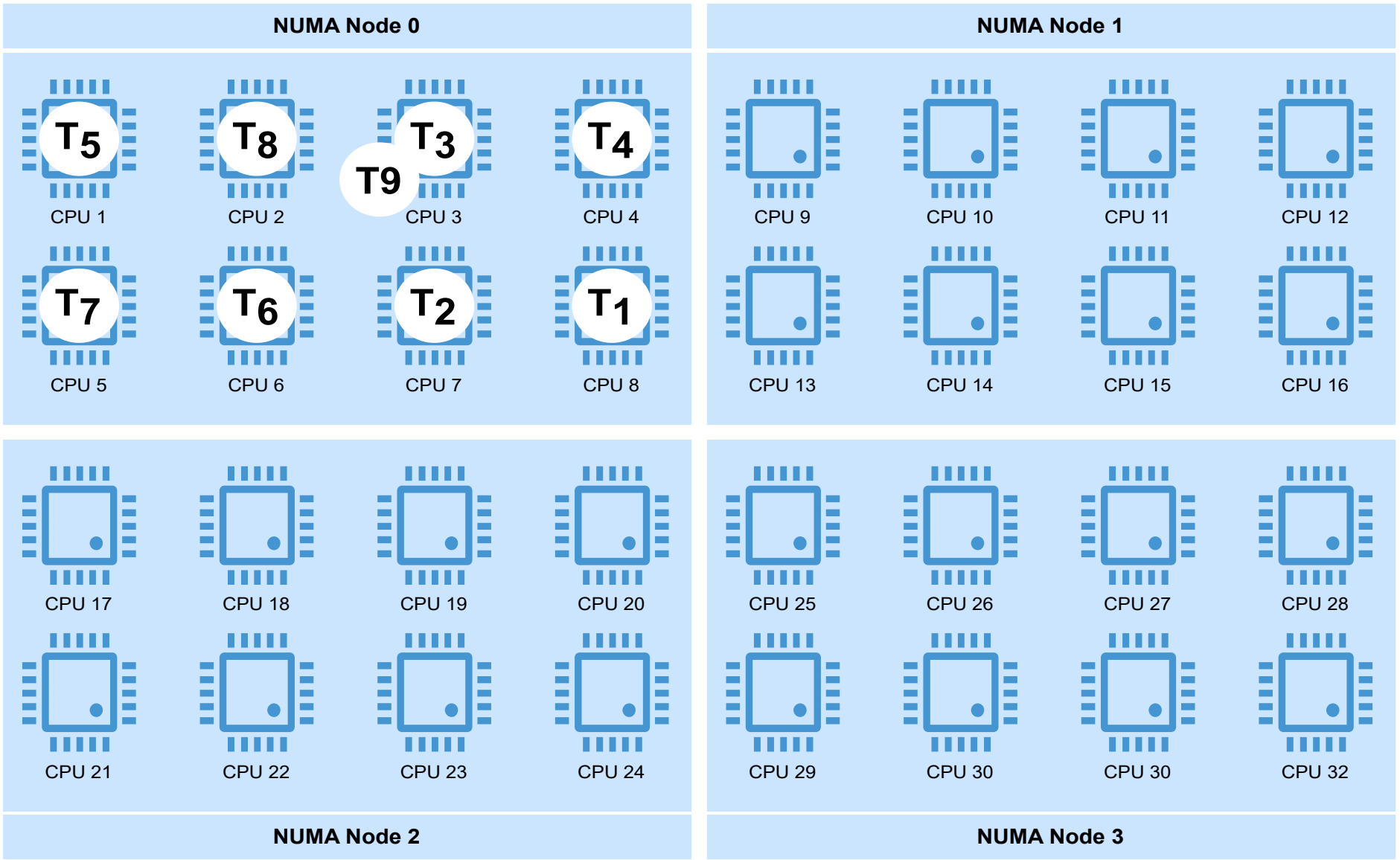
Execution of a Concurrent Program



NUMA Pinning



NUMA Pinning



Problems with NUMA pinning

- Increases average memory access time.

Problems with NUMA pinning

- Increases average memory access time.
- Increases inter-NUMA communication overhead.

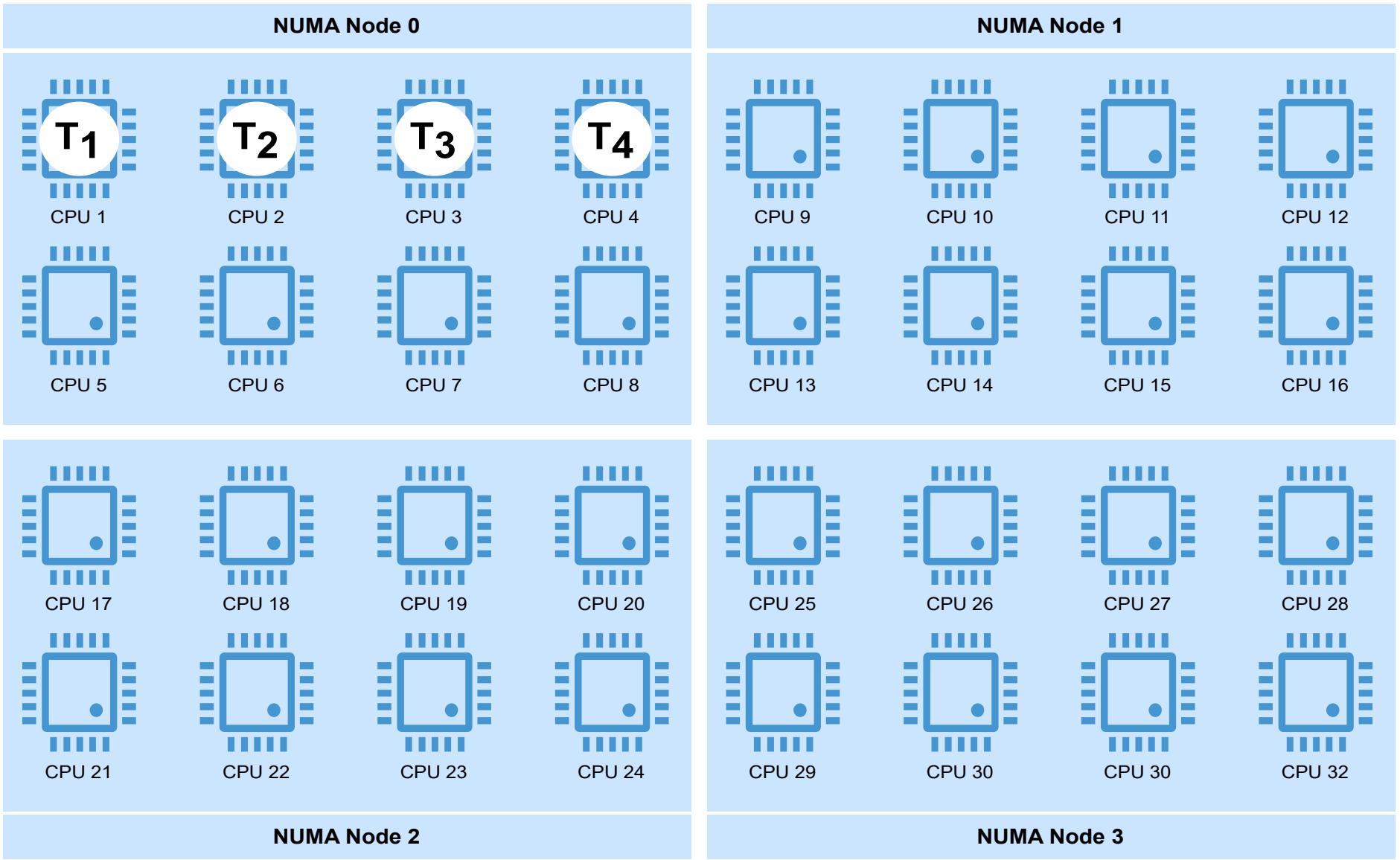
Problems with NUMA pinning

- Increases average memory access time.
- Increases inter-NUMA communication overhead.
- Increases contention resources.

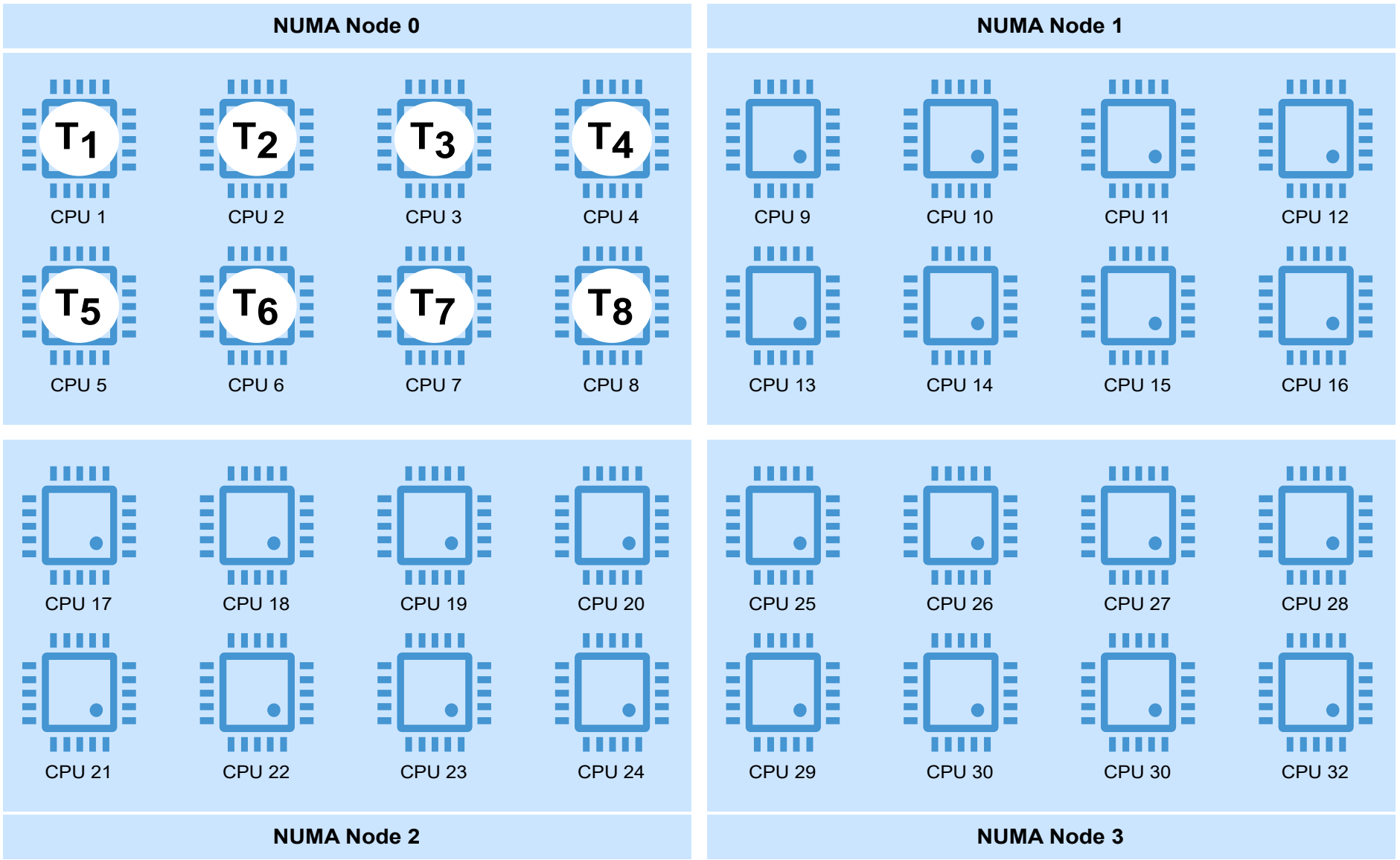
Problems with NUMA pinning

- Increases average memory access time.
- Increases inter-NUMA communication overhead.
- Increases contention resources.
- Load imbalance.

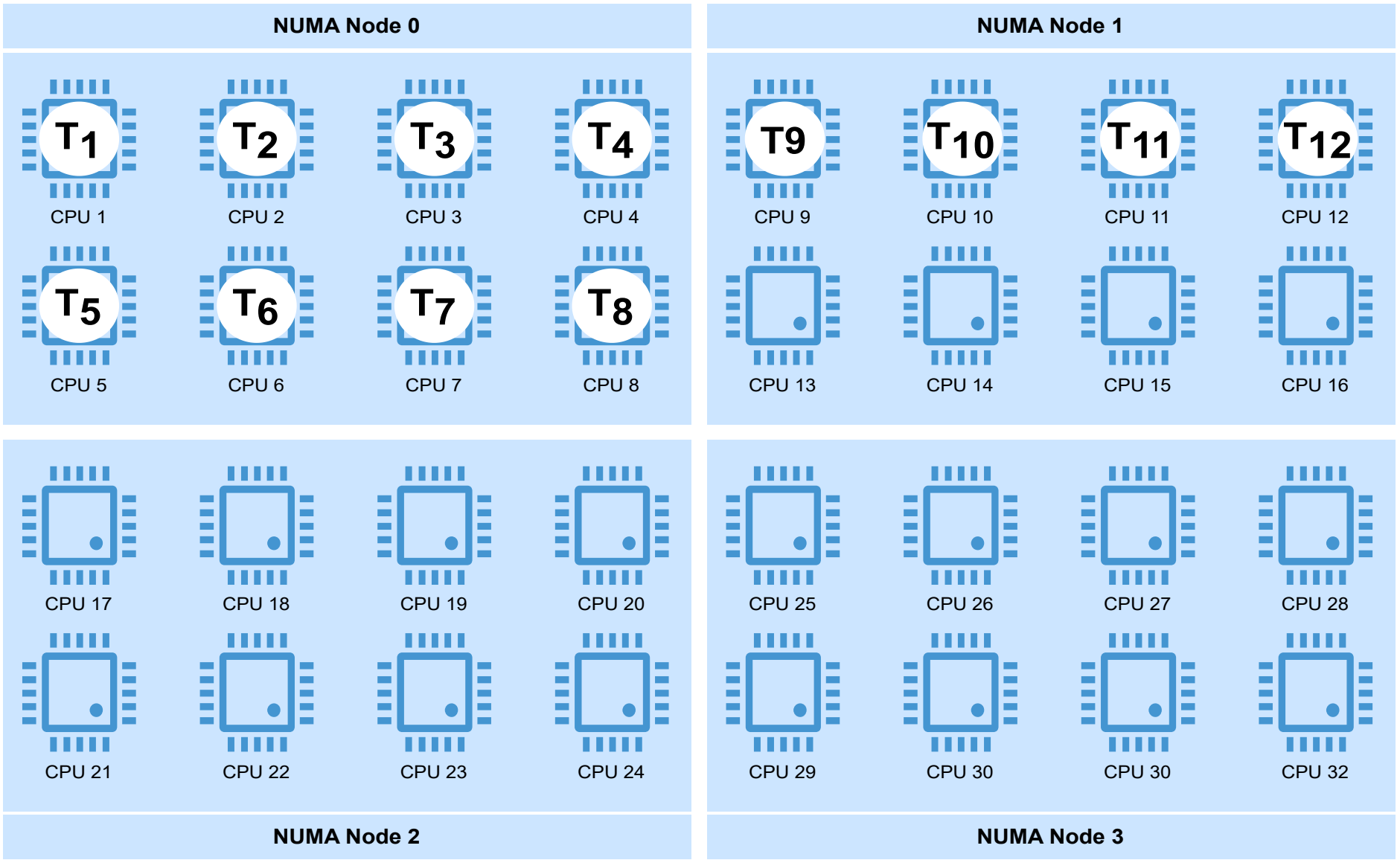
OCC Pinning


NUMA Node 0
T1
T2
T3
T4
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
NUMA Node 1
CPU 9
CPU 10
CPU 11
CPU 12
CPU 13
CPU 14
CPU 15
CPU 16
CPU 17
CPU 18
CPU 19
CPU 20
CPU 21
CPU 22
CPU 23
CPU 24
NUMA Node 2
CPU 25
CPU 26
CPU 27
CPU 28
CPU 29
CPU 30
CPU 30
CPU 32
NUMA Node 3

OCC Pinning


NUMA Node 0
T1
T2
T3
T4
CPU 1
CPU 2
CPU 3
CPU 4
T5
T6
T7
T8
CPU 5
CPU 6
CPU 7
CPU 8
NUMA Node 1
CPU 9
CPU 10
CPU 11
CPU 12
CPU 13
CPU 14
CPU 15
CPU 16
CPU 17
CPU 18
CPU 19
CPU 20
CPU 21
CPU 22
CPU 23
CPU 24
NUMA Node 2
CPU 25
CPU 26
CPU 27
CPU 28
CPU 29
CPU 30
CPU 30
CPU 32
NUMA Node 3

OCC Pinning


NUMA Node 0
T1
T2
T3
T4
CPU 1
CPU 2
CPU 3
CPU 4
T5
T6
T7
T8
CPU 5
CPU 6
CPU 7
CPU 8
NUMA Node 1
T9
T10
T11
T12
CPU 9
CPU 10
CPU 11
CPU 12
CPU 13
CPU 14
CPU 15
CPU 16
CPU 17
CPU 18
CPU 19
CPU 20
CPU 21
CPU 22
CPU 23
CPU 24
NUMA Node 2
CPU 25
CPU 26
CPU 27
CPU 28
CPU 29
CPU 30
CPU 30
CPU 32
NUMA Node 3

Algorithm of OCC Pinning

Algorithm 1 OCC Pinning

```
1: pid ← getCurrentProcessID();
2: threadsInfo ← getThreadsInfoForPID(pid);
3: spids ← extractSPIDs(threadsInfo);
4: for spid in spids do
5:     cpuIndex ← getAndIncrement() % totalCPUs;
6:     setThreadAffinity(spid, cpuIndex);
7: end for
```

OCC Pinning

- Thread pinning assigns specific software threads to dedicated processor cores.

OCC Pinning

- Thread pinning assigns specific software threads to dedicated processor cores.
- It prevents thread thrashing (threads constantly switch cores) and boosts efficiency.

OCC Pinning

- Thread pinning assigns specific software threads to dedicated processor cores.
- It prevents thread thrashing (threads constantly switch cores) and boosts efficiency.
- Leverage operating system mechanisms to identify processes and threads.

OCC Pinning

- Thread pinning assigns specific software threads to dedicated processor cores.
- It prevents thread thrashing (threads constantly switch cores) and boosts efficiency.
- Leverage operating system mechanisms to identify processes and threads.
- Leverage the taskset command to enforce desired thread-to-core mappings.

OCC Pinning

- Thread pinning assigns specific software threads to dedicated processor cores.
- It prevents thread thrashing (threads constantly switch cores) and boosts efficiency.
- Leverage operating system mechanisms to identify processes and threads.
- Leverage the taskset command to enforce desired thread-to-core mappings.
- This effectively overrides the dynamic behavior of the operating system scheduler.

Experimental System

Hardware:

- Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz
- RAM: 512GB RAM
- CPU(s): 96
- Thread(s) per core: 2
- Core(s) per socket: 24
- Socket(s): 2
- L1d cache: 32K, L1i cache: 32K, L2 cache: 1MB, L3 cache: 36MB

Software:

- Ubuntu 22.04 LTS
- JDK Runtime version 19.02

Experimental Evaluation

Pinning configurations:

- OCC Pinning
- NUMA Pinning
- No Pinning

Concurrent Data structures:

- Contention Adapting Binary Search Tree [1]
- Striped Hash Set [2]
- Lazy List [3]

Workload representation:

- xC-yI-zD

Workloads

- Read-only workload (100C-0I-0D).

Workloads

- Read-only workload (100C-0I-0D).
- Read dominant workload (70C-20I-10D).

Workloads

- Read-only workload (100C-0I-0D).
- Read dominant workload (70C-20I-10D).
- Balanced workload (50C-25I-25D).

Workloads

- Read-only workload (100C-0I-0D).
- Read dominant workload (70C-20I-10D).
- Balanced workload (50C-25I-25D).
- Update dominant workload (30C-35I-35D).

Workloads

- Read-only workload (100C-0I-0D).
- Read dominant workload (70C-20I-10D).
- Balanced workload (50C-25I-25D).
- Update dominant workload (30C-35I-35D).
- Update only workload (0C-50I-50D).

Experimental Evaluation

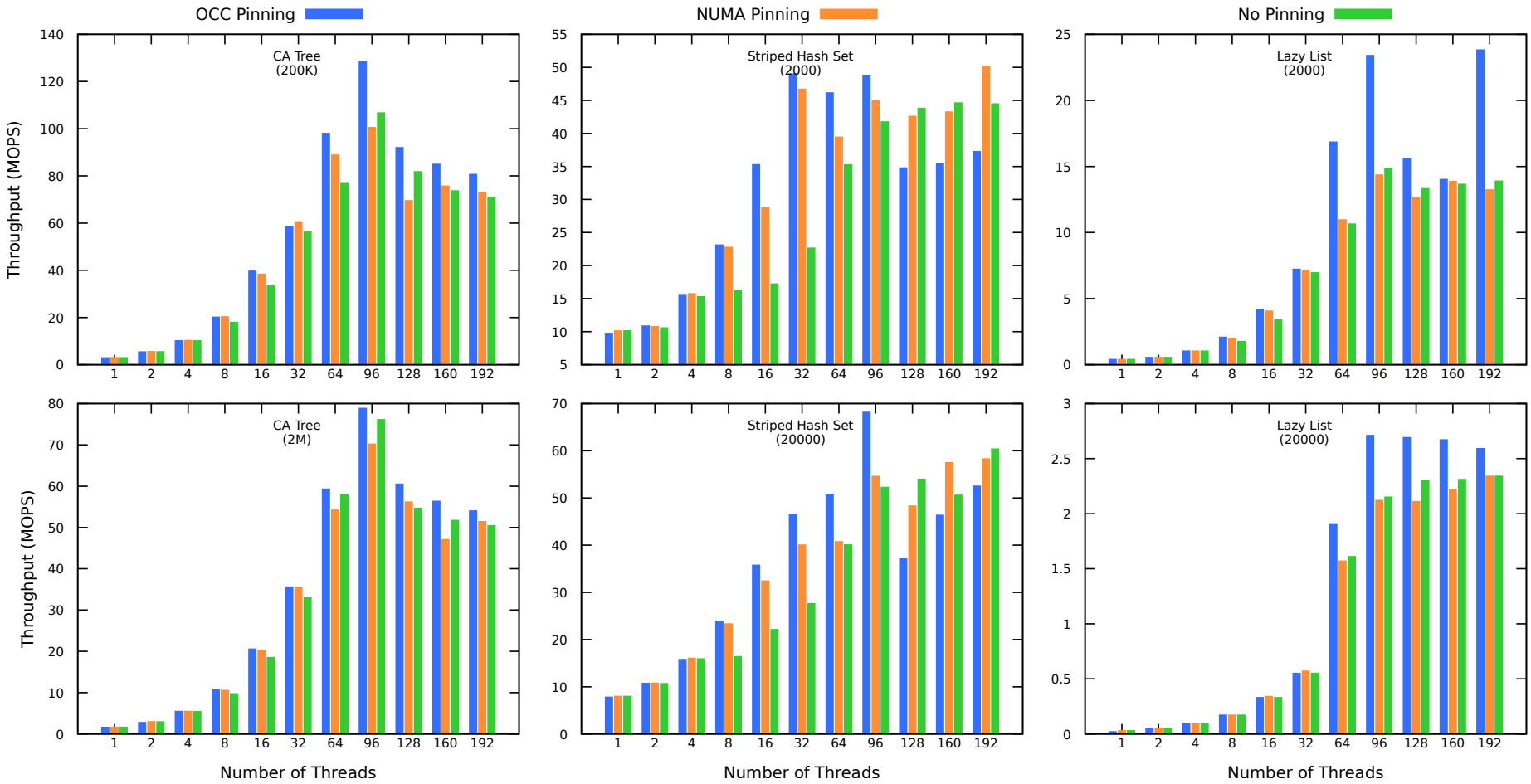


Figure 1: Average throughput of a CA tree, striped hash set, and lazy list implementations.

Experimental Evaluation

CA Tree

- OCC pinning achieved an average speedup of **1.10**× compared to both NUMA pinning and No pinning.

Striped Hash Set

- OCC pinning achieved an average speedup of **1.25**× compared to both NUMA pinning and No pinning.

Lazy List

- OCC pinning achieved an average speedup of **1.26**× compared to both NUMA pinning and No pinning.

Conclusion

- In the NUMA world, the OCC pinning strategy helps for efficient mapping between software threads and hardware threads.

Conclusion

- In the NUMA world, the OCC pinning strategy helps for efficient mapping between software threads and hardware threads.
- This approach achieves better resource utilization, minimizes interference, and cache misses, and outperforms traditional scheduling policies.

Conclusion

- In the NUMA world, the OCC pinning strategy helps for efficient mapping between software threads and hardware threads.
- This approach achieves better resource utilization, minimizes interference, and cache misses, and outperforms traditional scheduling policies.
- It optimizes performance by utilizing OS integration, granular affinity binding, and distributing threads across CPU cores.

Conclusion

- In the NUMA world, the OCC pinning strategy helps for efficient mapping between software threads and hardware threads.
- This approach achieves better resource utilization, minimizes interference, and cache misses, and outperforms traditional scheduling policies.
- It optimizes performance by utilizing OS integration, granular affinity binding, and distributing threads across CPU cores.
- It paves the way for realizing the full potential of concurrent applications.

Conclusion

- In the NUMA world, the OCC pinning strategy helps for efficient mapping between software threads and hardware threads.
- This approach achieves better resource utilization, minimizes interference, and cache misses, and outperforms traditional scheduling policies.
- It optimizes performance by utilizing OS integration, granular affinity binding, and distributing threads across CPU cores.
- It paves the way for realizing the full potential of concurrent applications.
- Future work includes the potential extension of this strategy to GPUs.

References


Konstantinos Sagonas and Kjell Winblad. 2015. Contention Adapting Search Trees. In 2015 14th International Symposium on Parallel and Distributed Computing. 215–224.


Steve Heller, Maurice Herlihy, Victor Luchangco, Mark Moir, William N. Scherer, and Nir Shavit. 2006. A Lazy Concurrent List-Based Set Algorithm.


Maurice Herlihy and Nir Shavit. 2012. The Art of Multiprocessor Programming, Revised Reprint (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.


The Linux Foundation. 2022. numactl(8) - Linux man page. https://man7.org/linux/man-pages/man8/numactl.8.html.

Thank You ☺