Memory Architectures in Heterogeneous Multicore SoCs

Axel Jantsch

European Nanoelectronics Design Technology Conference 2012

The Memory Wall

Wulf and McKee predicted in 1995 the impact into the memory wall:

$$t_{\text{avg}} = p \times t_c + (1 - p) \times t_m$$

t_{avg}	...average access latency for one data word
p	...probability of a cache hit
t_c	...access time to the cache
t_m	...access time to main memory

The Memory Wall

Wulf and McKee predicted in 1995 the impact into the memory wall:

$$t_{\text{avg}} = p \times t_c + (1 - p) \times t_m$$

t_{avg}	...average access latency for one data word
p	...probability of a cache hit
t_c	...access time to the cache
t_m	...access time to main memory

Today we navigate at the edge of this wall.

The Memory Wall

Wulf and McKee predicted in 1995 the impact into the memory wall:

$$t_{\text{avg}} = p \times t_c + (1 - p) \times t_m$$

t_{avg}	...average access latency for one data word
p	...probability of a cache hit
t_c	...access time to the cache
t_m	...access time to main memory

Today we navigate at the edge of this wall.

For example the Tilera 64 core chip:

- Raw processor performance: 443 Gops

The Memory Wall

Wulf and McKee predicted in 1995 the impact into the memory wall:

$$t_{\text{avg}} = p \times t_c + (1 - p) \times t_m$$

t_{avg}	...average access latency for one data word
p	...probability of a cache hit
t_c	...access time to the cache
t_m	...access time to main memory

Today we navigate at the edge of this wall.

For example the Tilera 64 core chip:

- Raw processor performance: 443 Gops
- Required memory bandwidth with two cache levels and 95% hit rate: 7.2Gb/s
- Available memory bandwidth: 50 Gb/s

The Memory Wall

Wulf and McKee predicted in 1995 the impact into the memory wall:

$$t_{\text{avg}} = p \times t_c + (1 - p) \times t_m$$

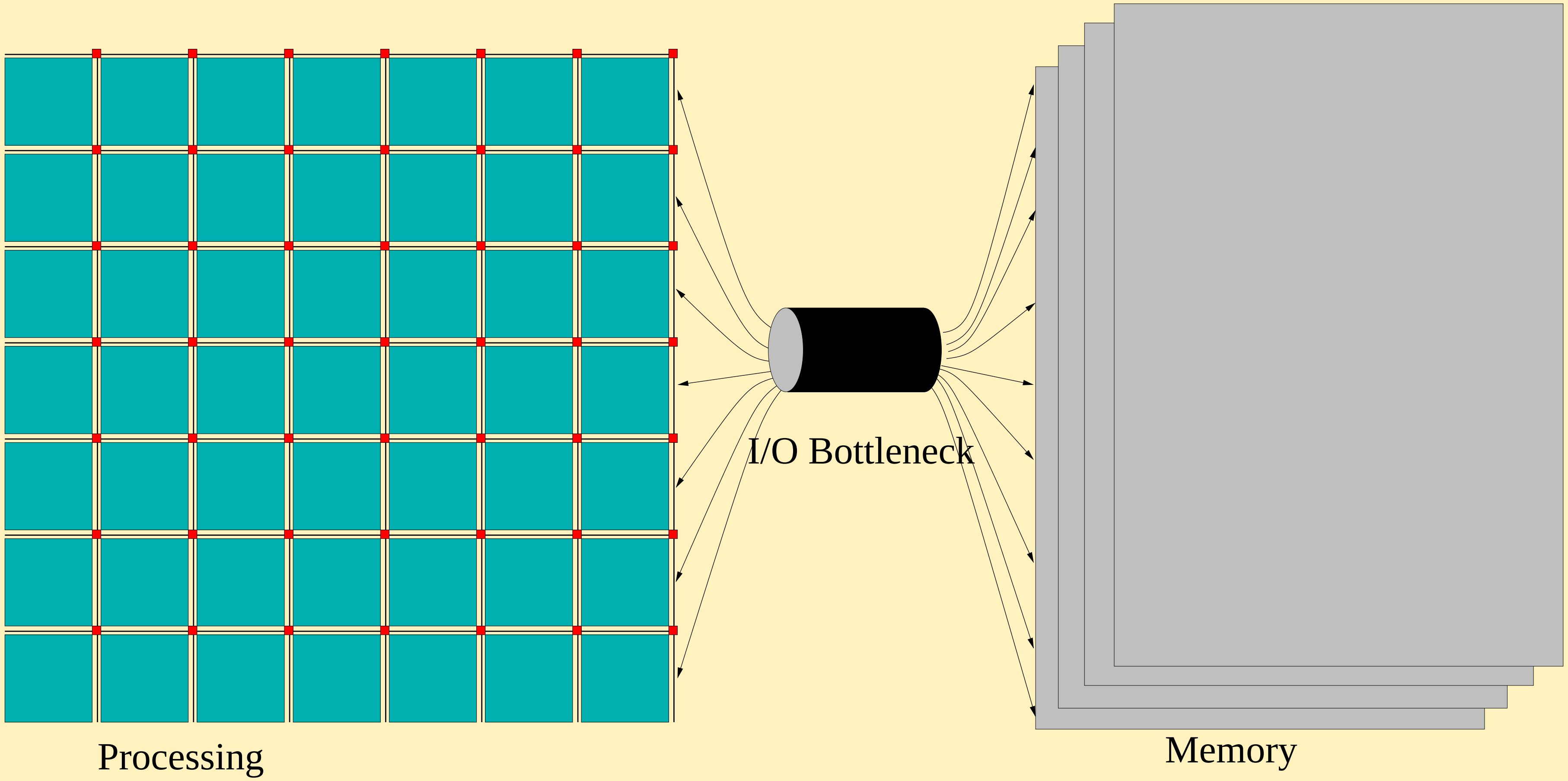
t_{avg}	...average access latency for one data word
p	...probability of a cache hit
t_c	...access time to the cache
t_m	...access time to main memory

Today we navigate at the edge of this wall.

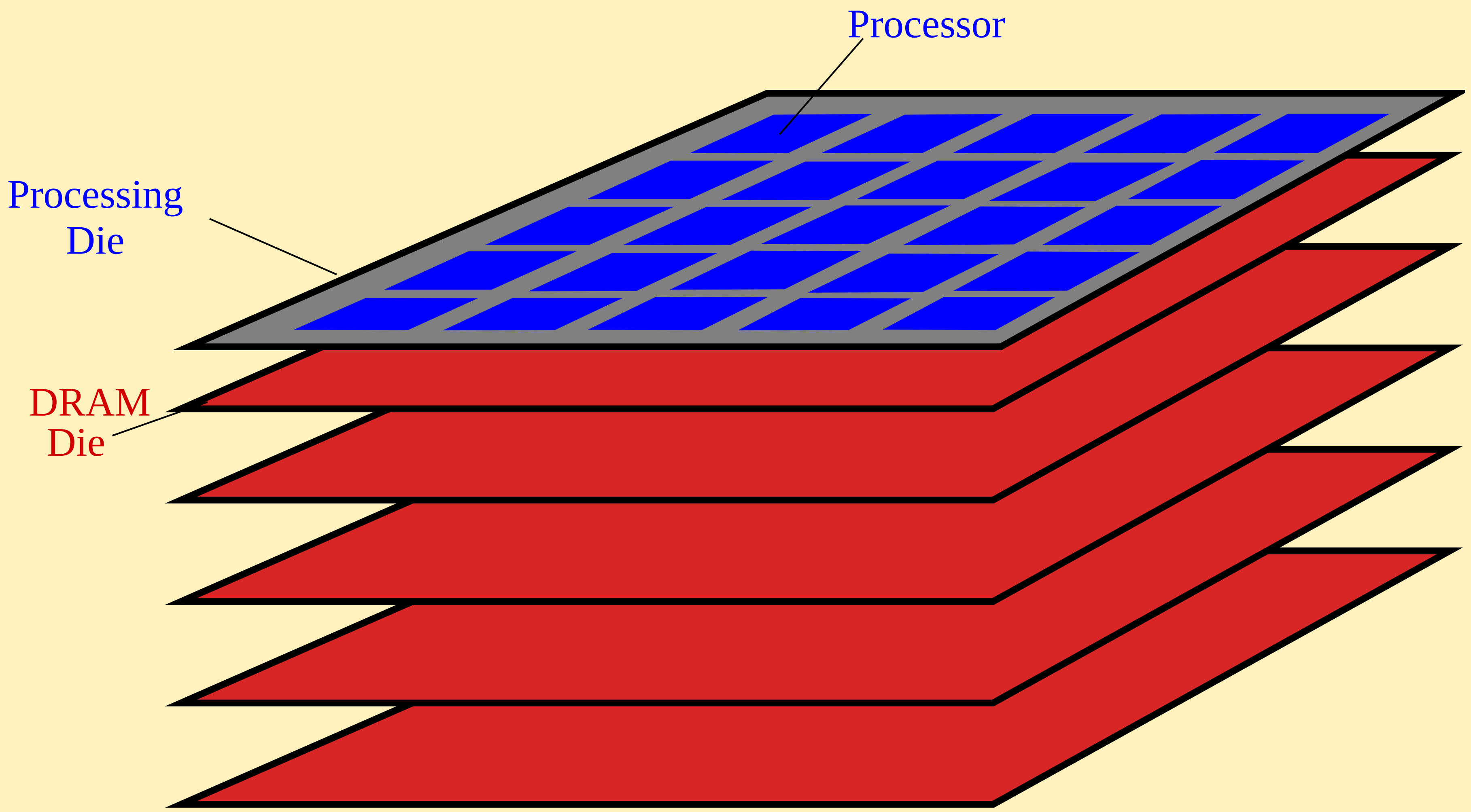
For example the Tilera 64 core chip:

- Raw processor performance: 443 Gops
- Required memory bandwidth with two cache levels and 95% hit rate: 7.2Gb/s
- Available memory bandwidth: 50 Gb/s
- Required memory access latency: 0.625 cycles
- Available average access time: 1.475 cycles

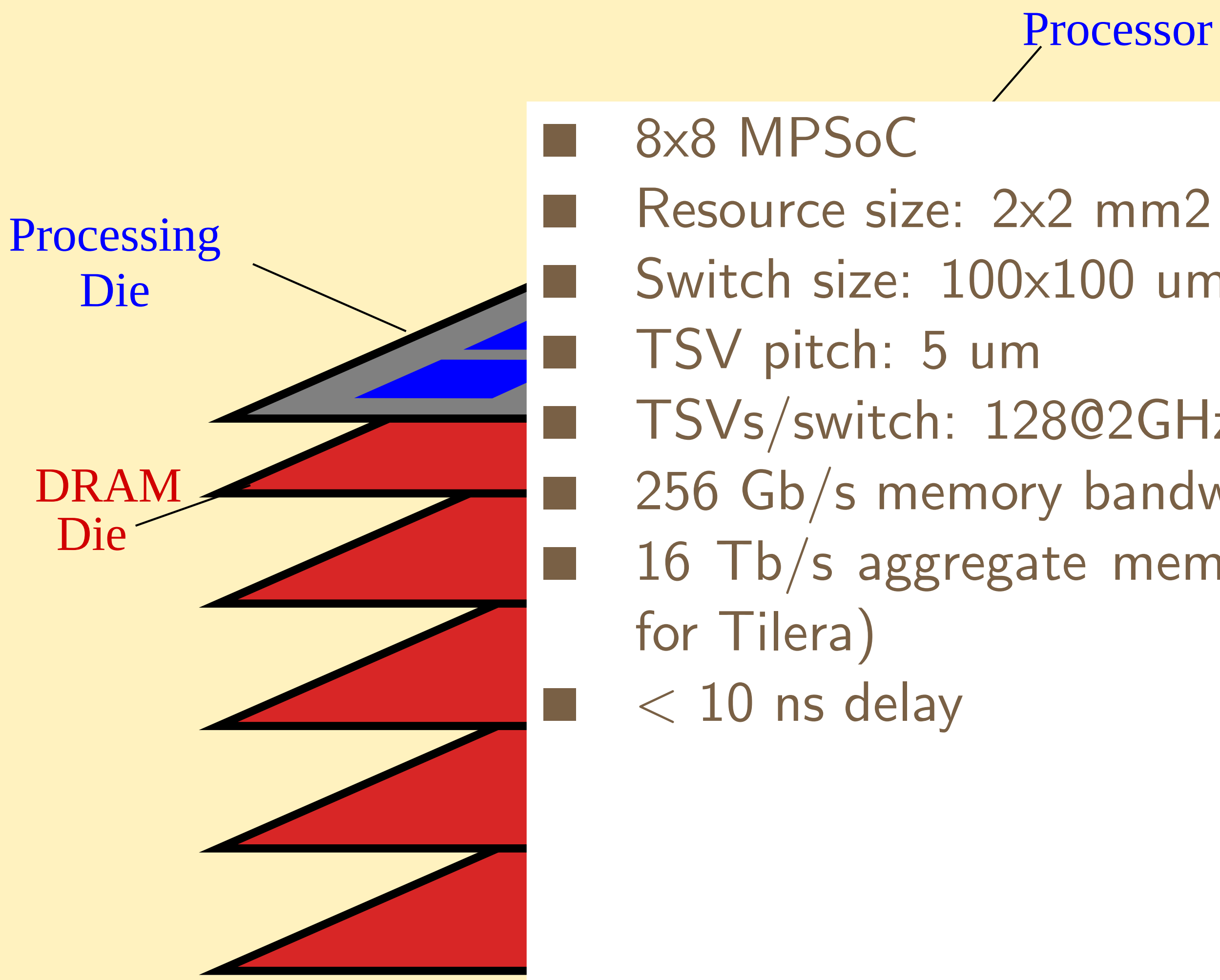
The Memory Access Bottleneck



Memory Bandwidth in 3D ICs

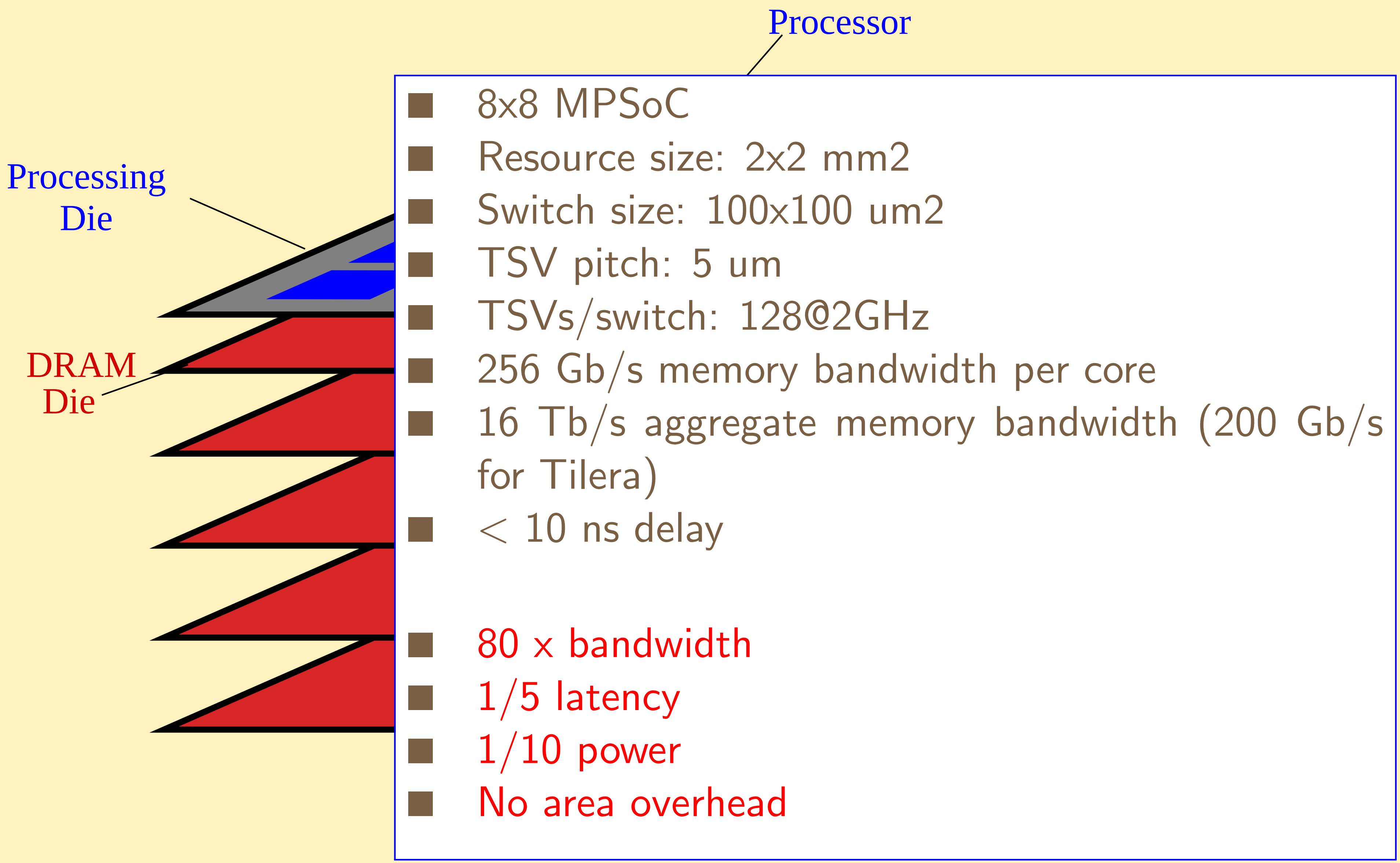


Memory Bandwidth in 3D ICs



- 8x8 MPSoC
- Resource size: 2x2 mm2
- Switch size: 100x100 um2
- TSV pitch: 5 um
- TSVs/switch: 128@2GHz
- 256 Gb/s memory bandwidth per core
- 16 Tb/s aggregate memory bandwidth (200 Gb/s for Tilera)
- $<$ 10 ns delay

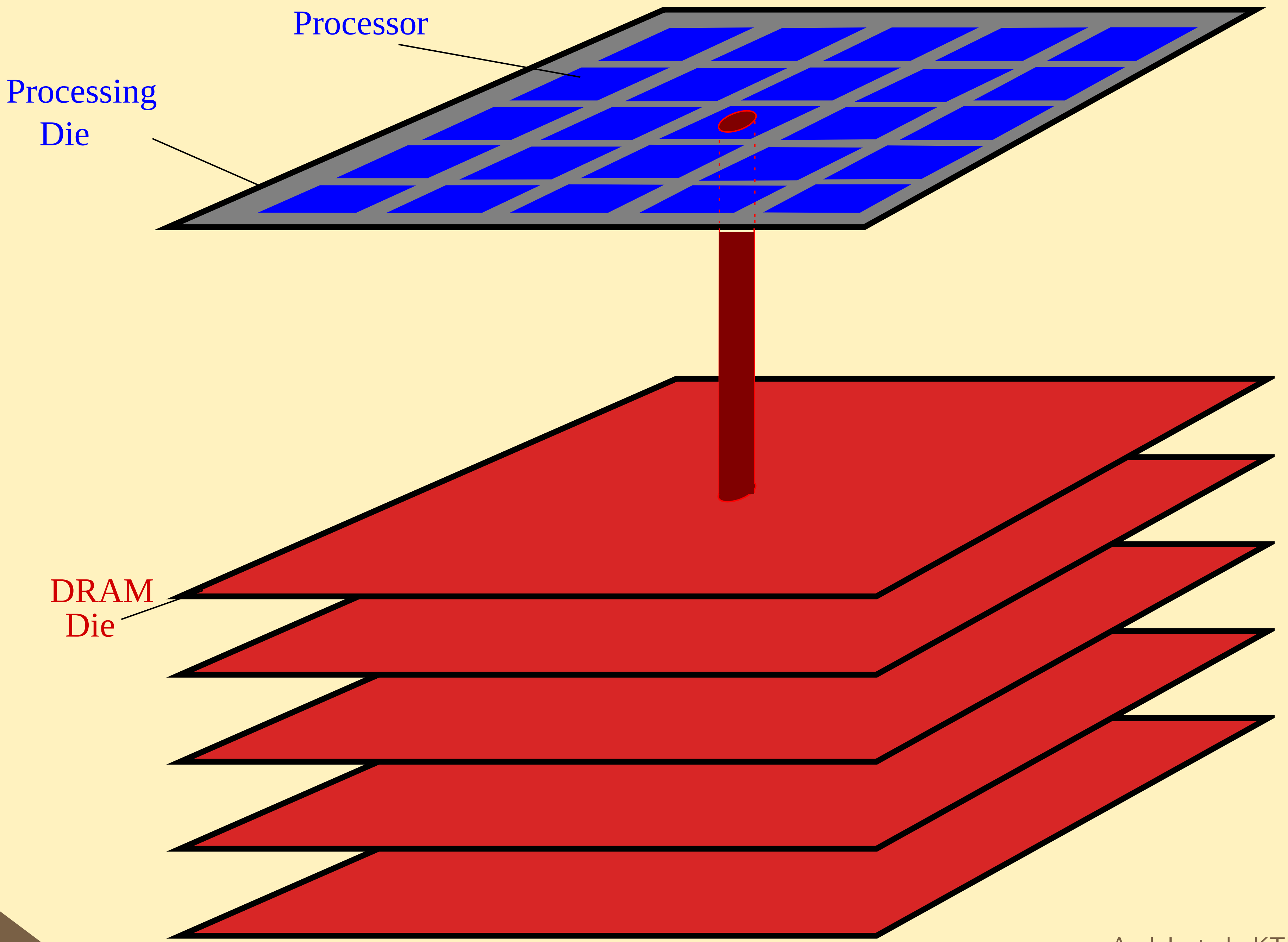
Memory Bandwidth in 3D ICs



- 8x8 MPSoC
- Resource size: 2x2 mm2
- Switch size: 100x100 um2
- TSV pitch: 5 um
- TSVs/switch: 128@2GHz
- 256 Gb/s memory bandwidth per core
- 16 Tb/s aggregate memory bandwidth (200 Gb/s for Tilera)
- $<$ 10 ns delay

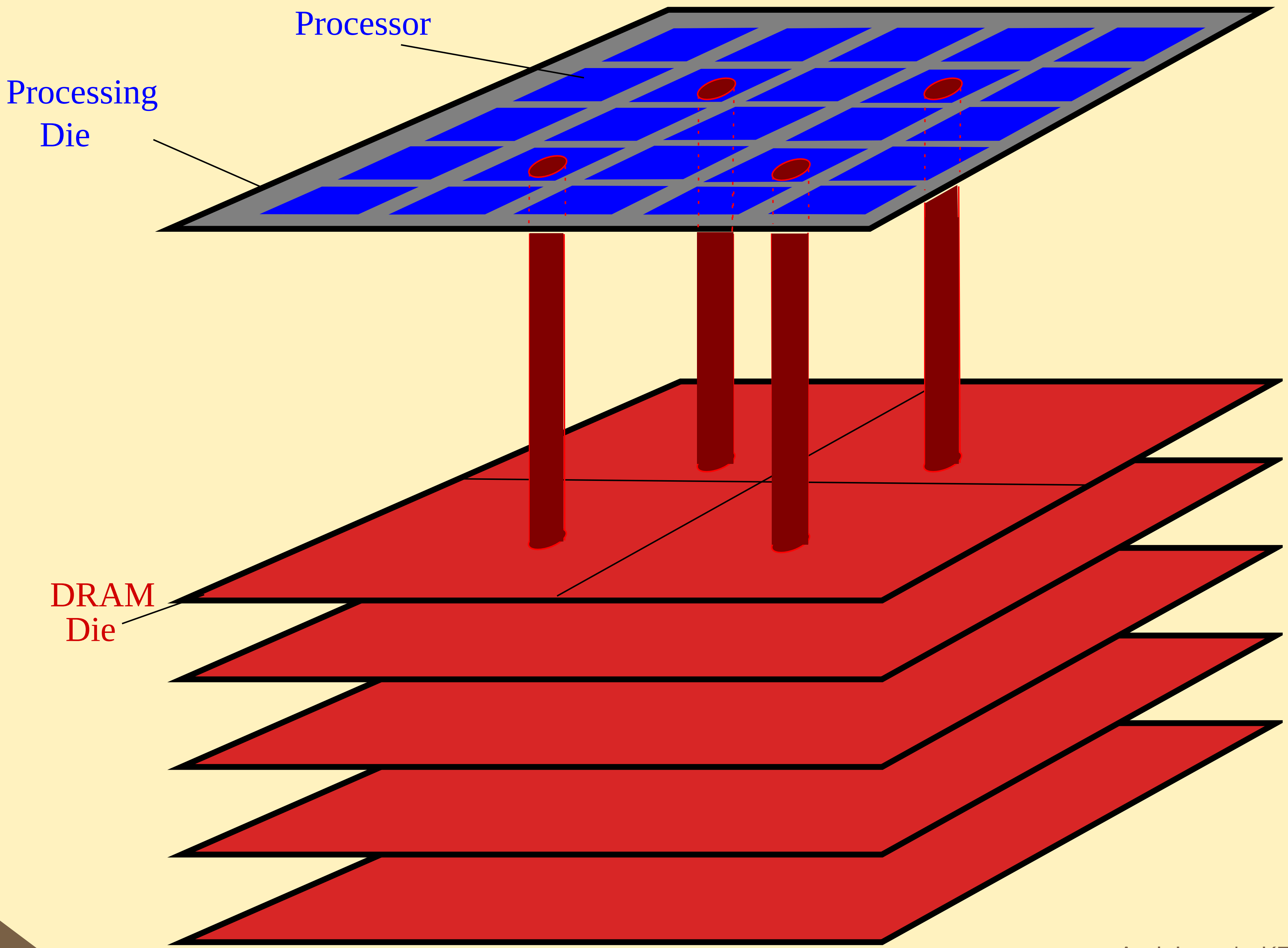
- 80 x bandwidth
- 1/5 latency
- 1/10 power
- No area overhead

Memory Organization in 3D ICs
n cores to 1 port



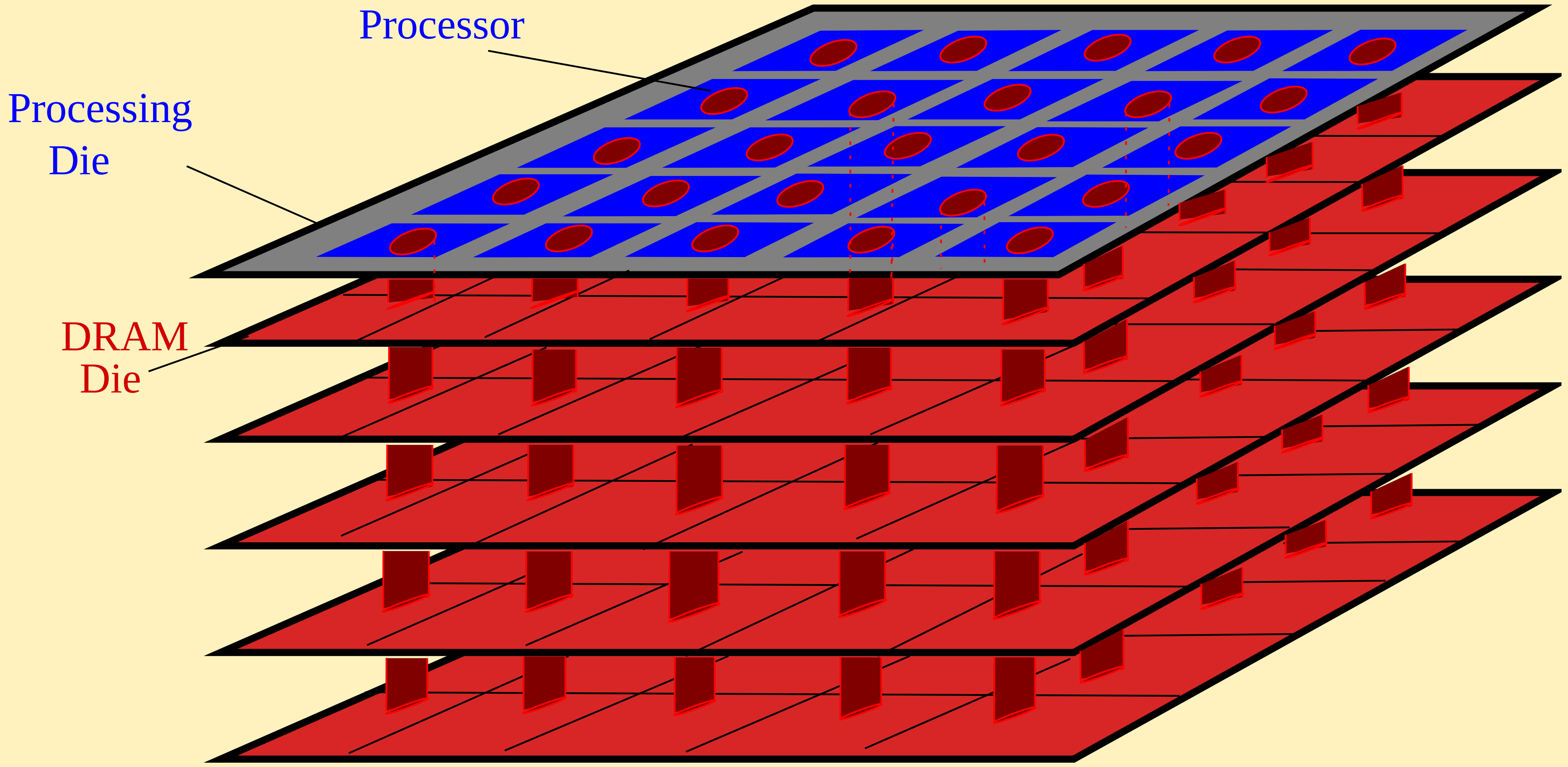
Memory Organization in 3D ICs

n cores to m ports, $n < m$



Memory Organization in 3D ICs

n cores to n ports



Memory Access Protocols

Protocol Characteristics

	Frequency	buswidth	Voltage	max bandiwtdh	Efficiency
DDR3	1066 MHz	32 bit	1.5 V	8.532 GB/s	4.17 GB/s/W
LPDDR2	533 MHz	32 bit	1.2 V	4.264 GB/s	6.25 GB/s/W
Wide I/O	200 MHz	512 bit	1.2 V	12.8 GB/s	25.0 GB/s/W

Cost for 1 TB/s

Protocol	No of ports	No of pads	Power
DDR3	120	3800	240 W
LPDDR2	240	7700	160 W
Wide I/O	80	41000	40 W

From: Denis Dutois and Ahmed Jerraya. "3D Integration Opportunities for Memory Interconnect in Mobile Computing Architectures". In: *Future Fab International* 34 (July 2010)

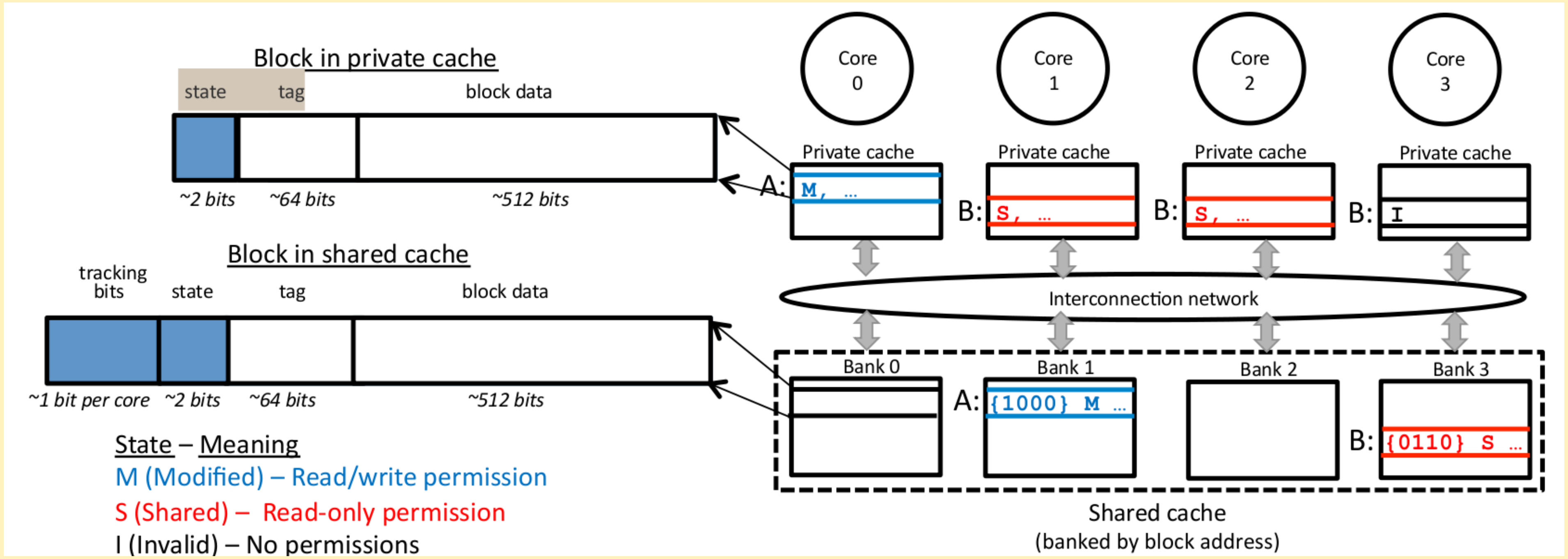
Memory Architecture
Industrial State of the Art

- Custom memory architectures in many SoCs and embedded systems with no shared memory space
- No scalable HW support for shared memory space, cache coherence and consistency (e.g. Intel's 48 core SCC)
- Uniform shared memory space with snooping based cache coherence for small multi-core systems (ARM's Cortex A9)
- No general, flexible, and scalable solution for the many-core era

Memory Architecture Research State of the Art

- Wide range of work on scalable message passing architectures and programming models (Actor based models, dataflow models, MoC)
  Andras Vajda. *Programming Many-core Chips*. Springer, 2011
  Joe Armstrong, Robert Virding, and Mike Williams. *Concurrent Programming in Erlang*. Prentice Hall, 1993
- Non-Uniform Cache Architecture (NUCA) in regular many-core SoCs; introduced in 2002
  C. Kim, D. Burger, and S. Keckler. "An Adaptive, Non-uniform Cache Structure for Wire-Delay Dominated On-Chip Caches". In: *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2002
- Wide range of work on shared memory models, cache coherence and memory consistency
  Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. "Why on-chip cache coherence is here to stay". In: *Comunications of the ACM* (July 2012)

Scalability of Cache Coherence

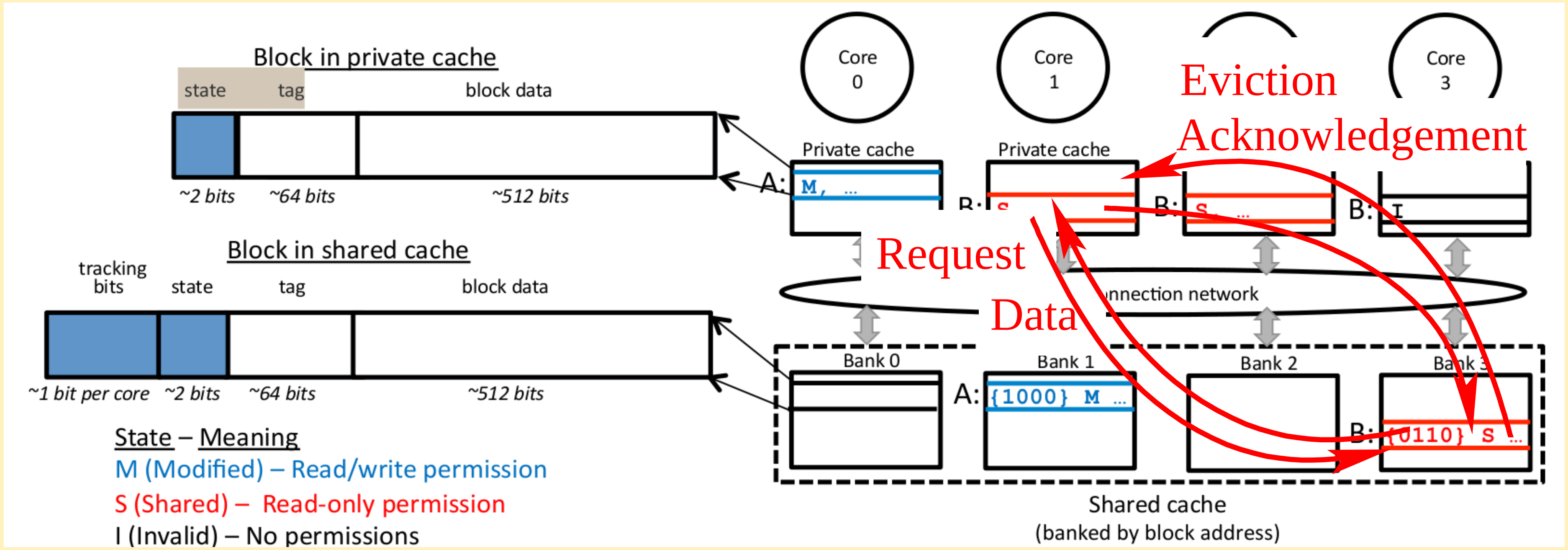


- ≤ 1 core with read and write permission (**Modified**)
- 0 or more readers (**Shared**)
- L1 cache is private; L2 cache is shared
- inclusive cache (all L1s are in L2)
- One L2 bank per core (bandwidth scaling)
- Protocol with eviction and acknowledgment messages

From Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. "Why on-chip cache coherence is here to stay". In: *Comunications of the ACM* (July 2012)

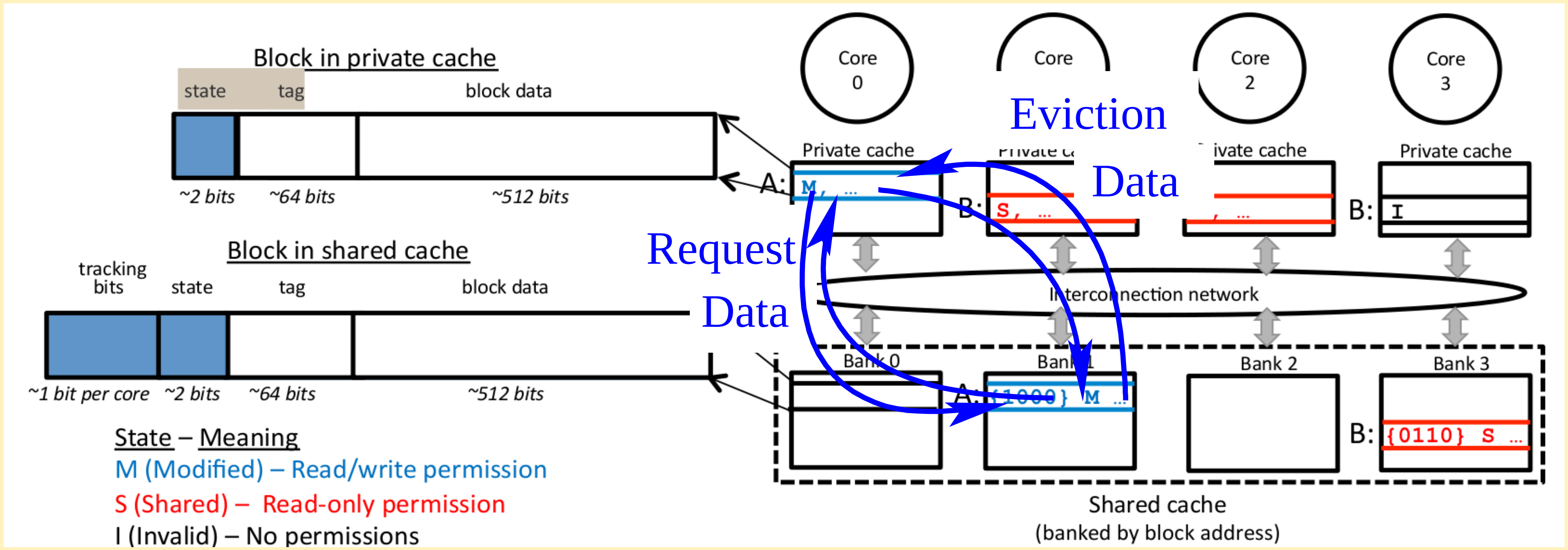
Inexact Tracking
Cache miss on a clean block

$$\begin{array}{lllllll} & (\text{Request} & + & \text{Data}) & + & (\text{Eviction} & + \text{Acknowledgment}) \\ = & (8 & + & (64+8)) & + & (8 & + 8) \\ = & 96\ \text{Bytes/miss} & & & & & \end{array}$$



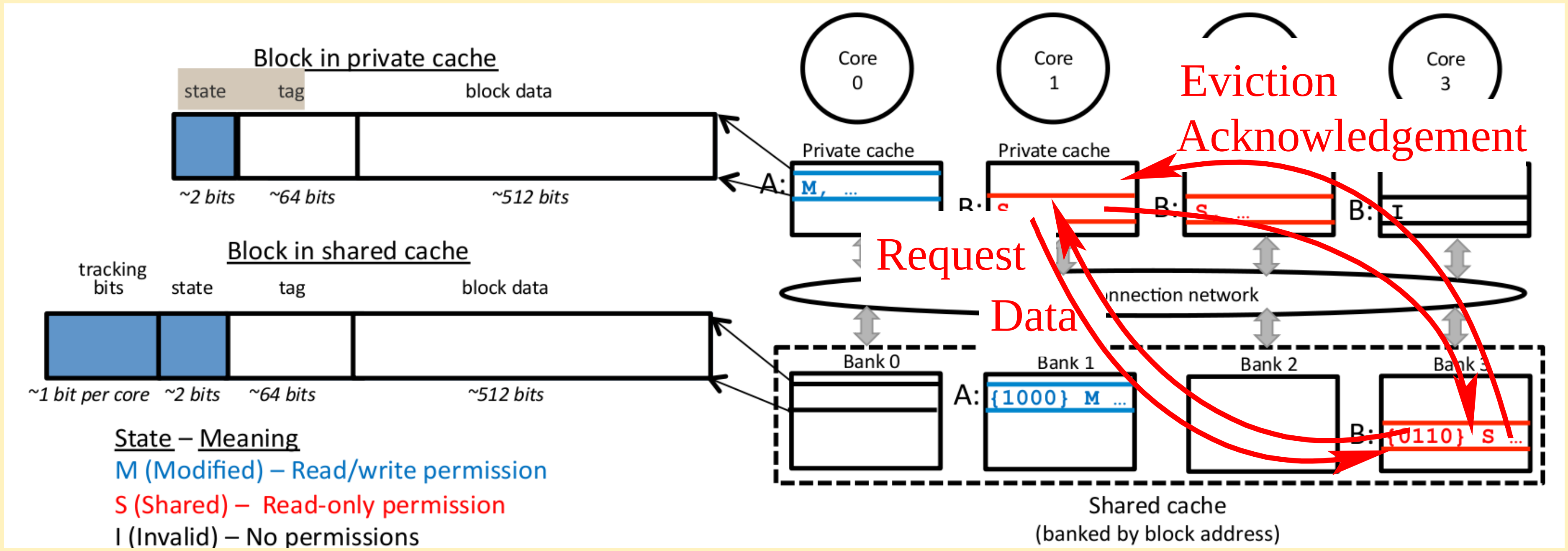
Inexact Tracking
Cache miss on a dirty block

$$\begin{array}{lllllll} & (\text{Request} & + \quad \text{Data}) & + & (\text{Data} & + \text{Acknowledgment}) & + \text{Invalidation} \\ = & (8 & + \quad (64+8)) & + & ((64+8) & +8) & +8n \\ = & 160 + 8n \text{ Bytes/miss} & & & & & \end{array}$$



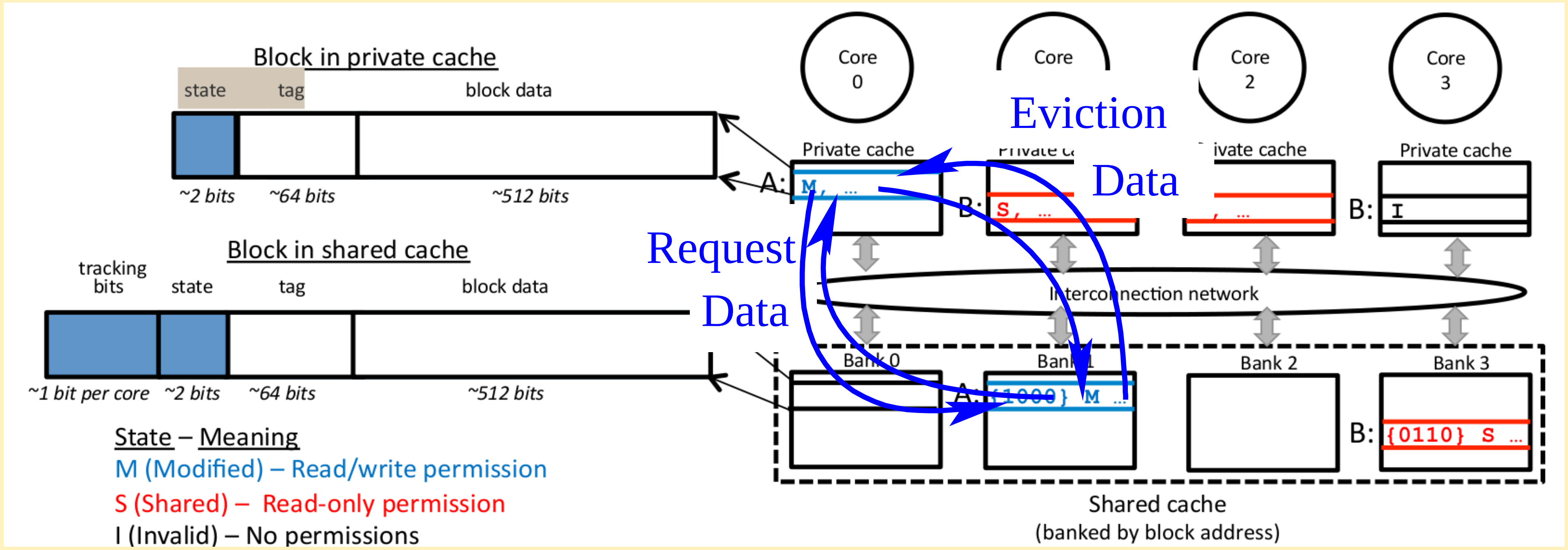
Exact Tracking
Cache miss on a **clean block**

$$\begin{array}{lllllll} & (\text{Request} & + \quad \text{Data}) & + & (\text{Eviction} & + \text{Acknowledgment}) & + \text{Invalidation} \\ = & (8 & + \quad (64+8)) & + & (8 & + 8) & + 8 \\ = & 104 \text{ Bytes/miss} & & & & & \end{array}$$



Exact Tracking
Cache miss on a dirty block

$$\begin{aligned} & (\text{Request} + \text{Data}) + (\text{Data} + \text{Acknowledgment}) \\ = & (8 + (64+8)) + ((64+8) + 8) \\ = & 160 \text{ Bytes/miss} \end{aligned}$$



State – Meaning
M (Modified) – Read/write permission
S (Shared) – Read-only permission
I (Invalid) – No permissions

From Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. “Why on-chip cache coherence is here to stay”. In: *Comunications of the ACM* (July 2012)

Scalability of Cache Coherence

	Clean block	Dirty block
Without coherence	80 Bytes/miss	152 Bytes/miss
With coherence	104 Bytes/miss	160 Bytes/miss
Per-miss traffic overhead	30%	5%

Overhead independent of the number of cores.

Conclusion from the State of the Art

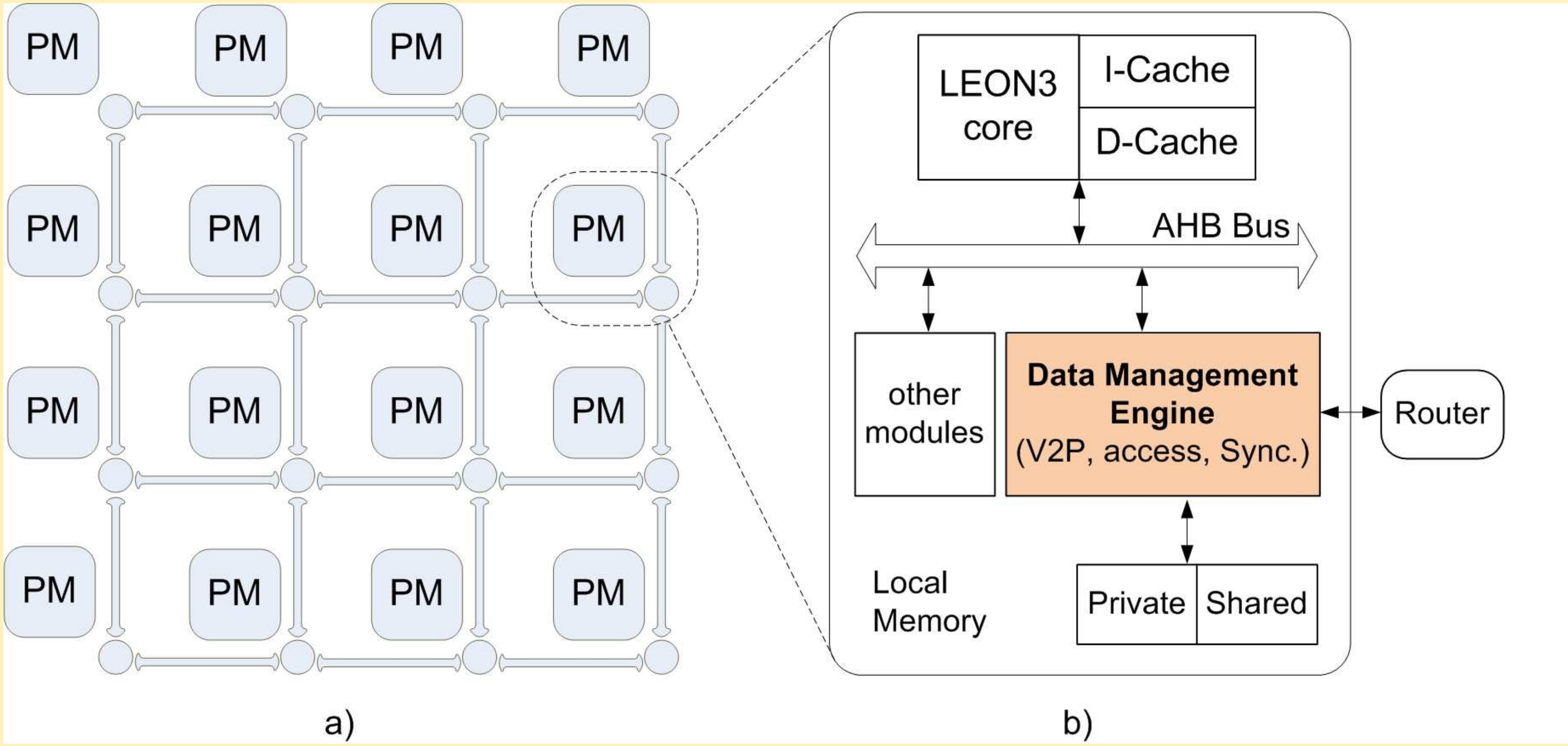
- Both message passing and shared memory models can be implemented in a scalable way
- A variety of communication models and memory architectures will continue to exist
- Optimal model and architecture is highly application dependant
- We need flexible platforms that can be adapted to the application

Data Management Engine

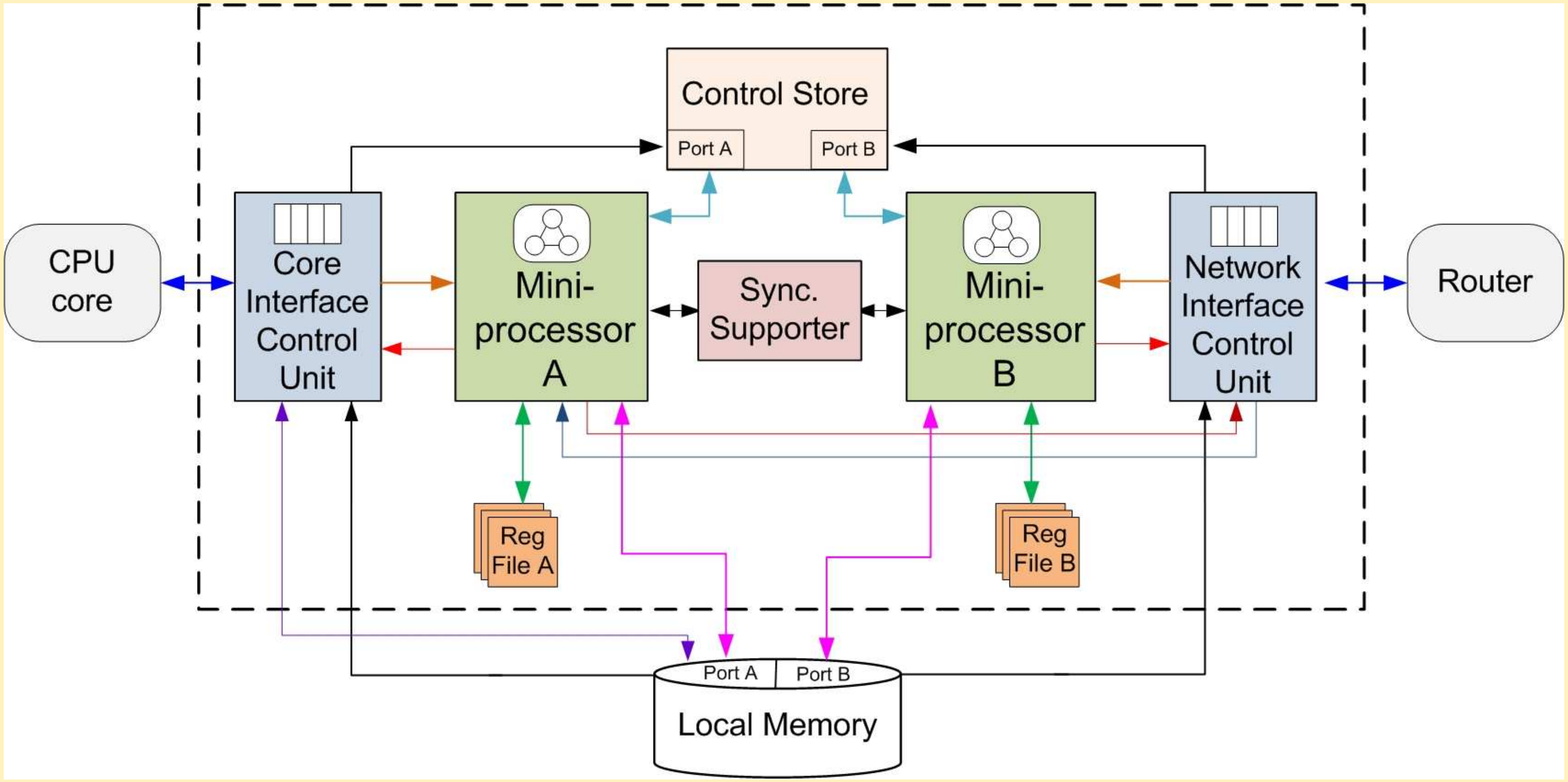
Programmable controller for realizing memory architectures and handling data access

- DMA functions
- Message passing communication
- Shared memory
- Virtual address space
- Cache coherence
- Memory consistency

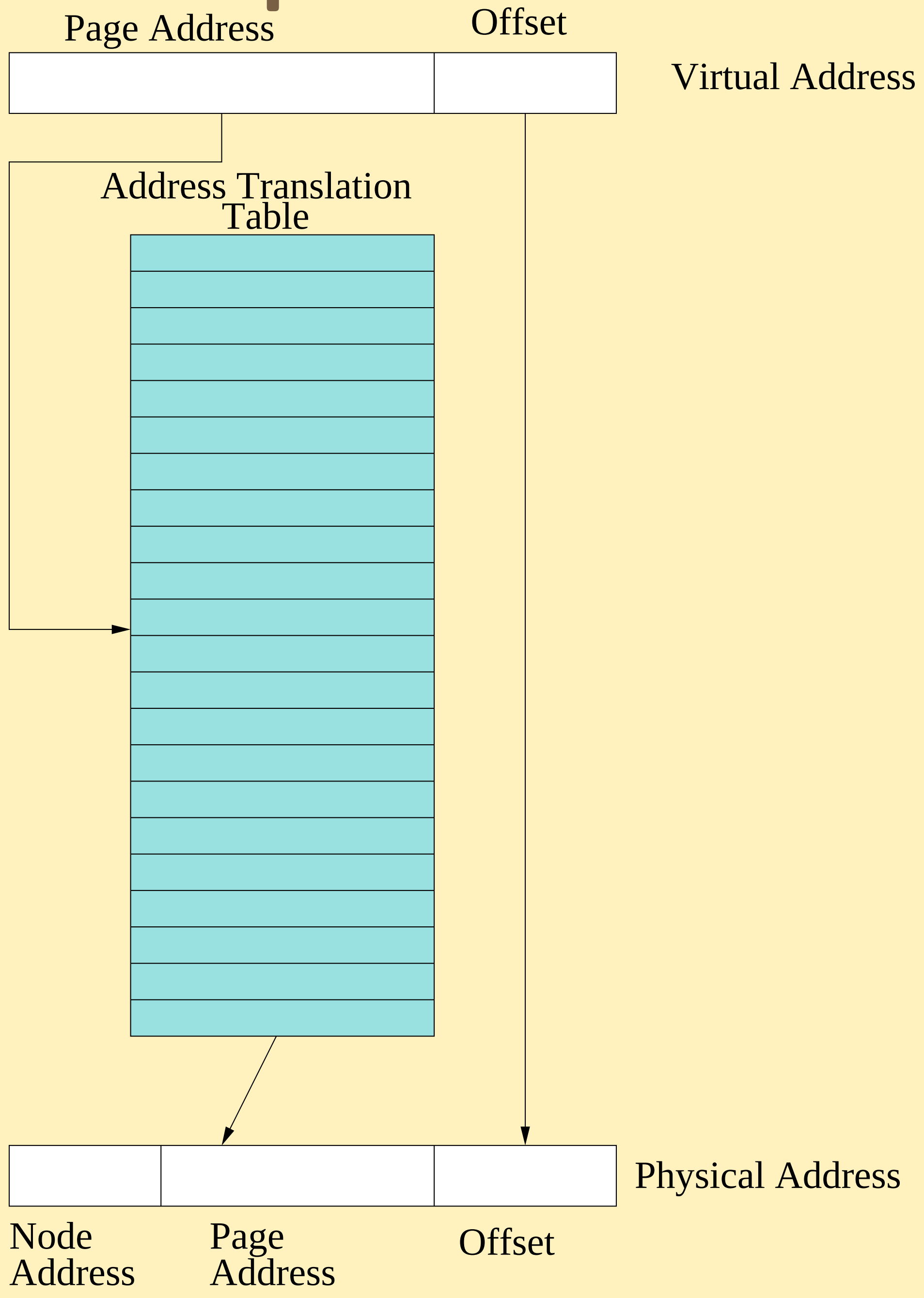
Platform Overview



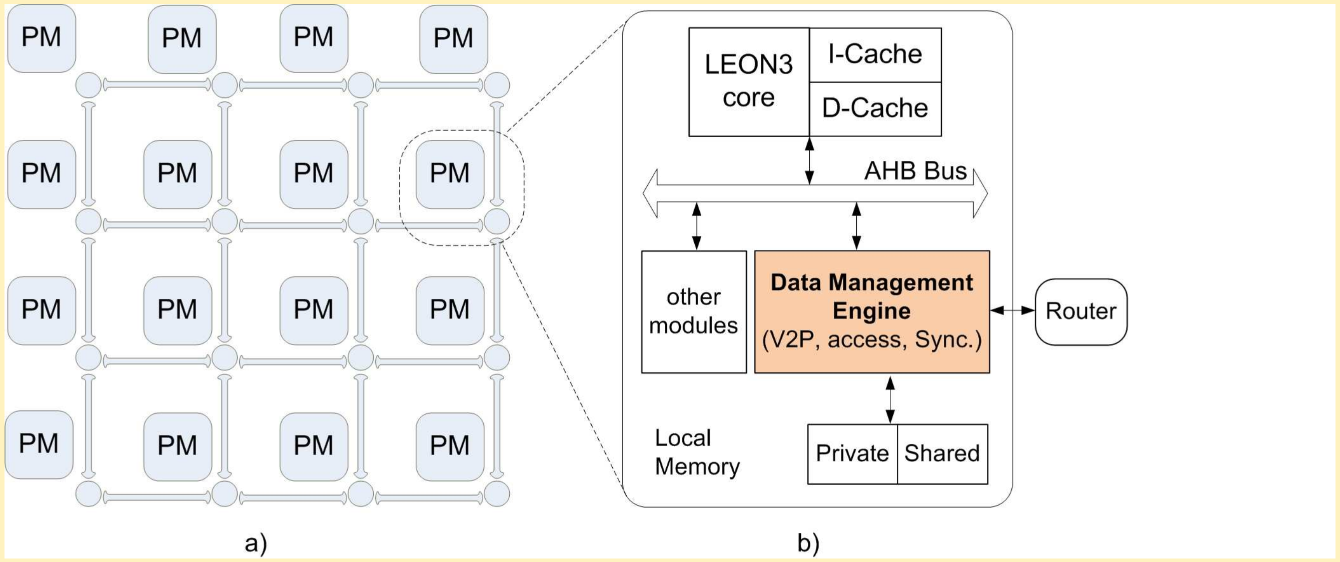
Data Management Engine Architecture



Virtual Address Space Translation



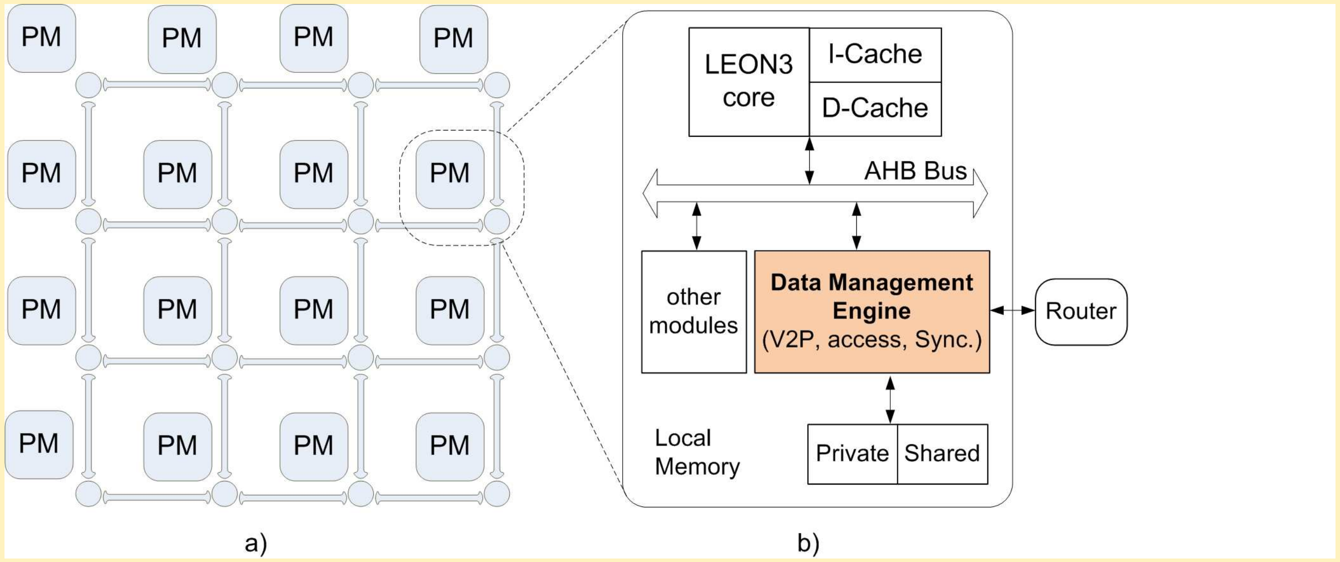
Virtual Address Space Translation



$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	... Size of translation table per node
N	... Number of nodes
M_N	... Memory per node
P_S	... Page size
M_T	... Total memory size

Virtual Address Space Translation

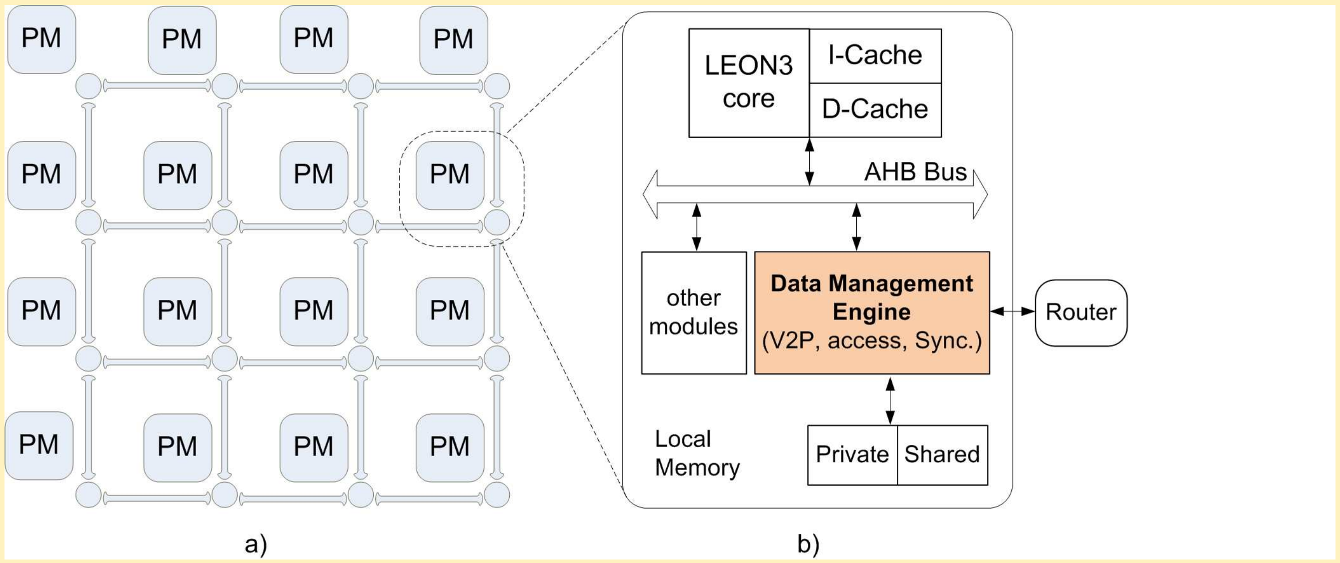


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	...	Size of translation table per node
N	...	Number of nodes
M_N	...	Memory per node
P_S	...	Page size
M_T	...	Total memory size

- 64 nodes - 2^6

Virtual Address Space Translation

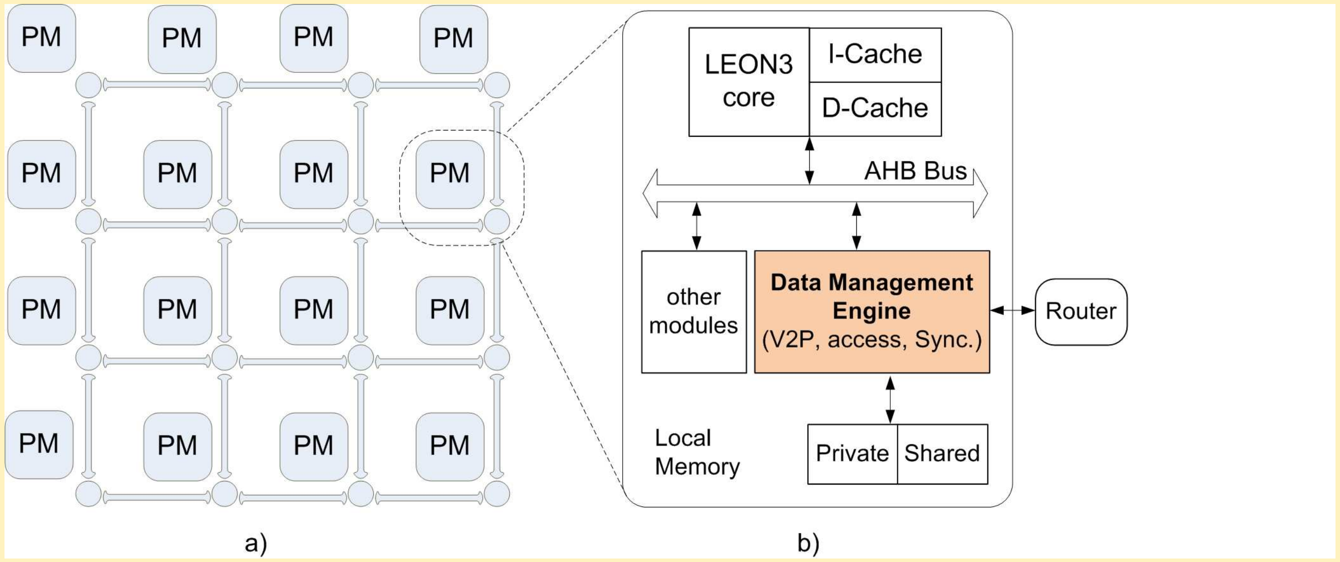


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	...	Size of translation table per node
N	...	Number of nodes
M_N	...	Memory per node
P_S	...	Page size
M_T	...	Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}

Virtual Address Space Translation

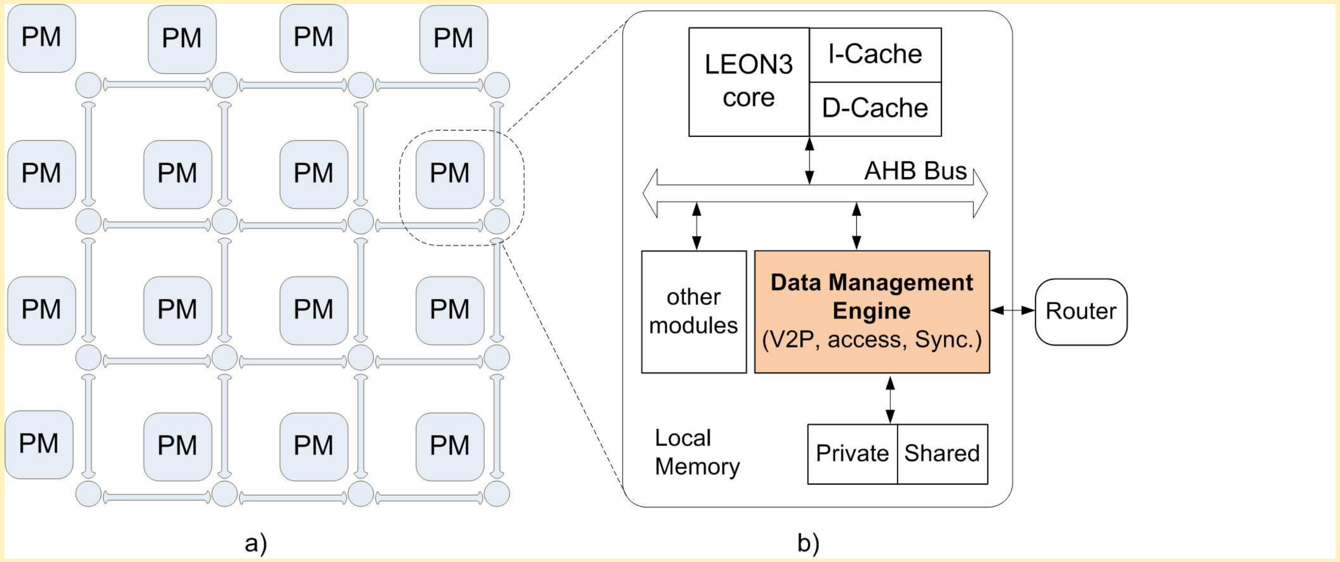


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	... Size of translation table per node
N	... Number of nodes
M_N	... Memory per node
P_S	... Page size
M_T	... Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}
- 1 GB total memory - 2^{30}

Virtual Address Space Translation

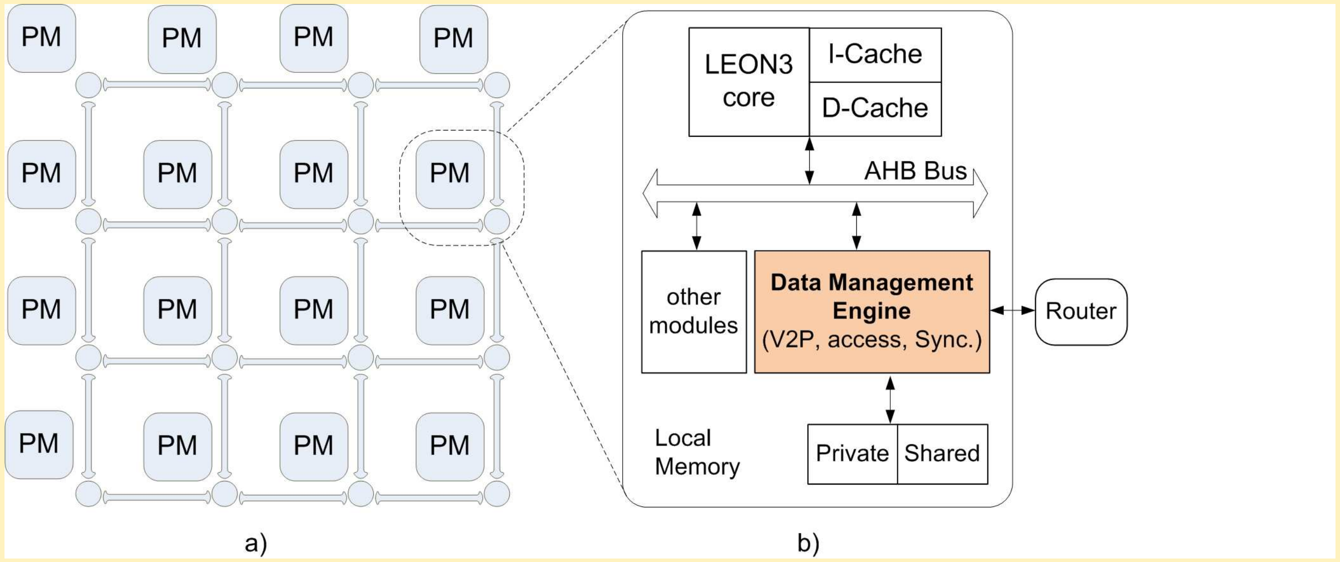


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	... Size of translation table per node
N	... Number of nodes
M_N	... Memory per node
P_S	... Page size
M_T	... Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}
- 1 GB total memory - 2^{30}
- Page size 1KB - 2^{10}

Virtual Address Space Translation

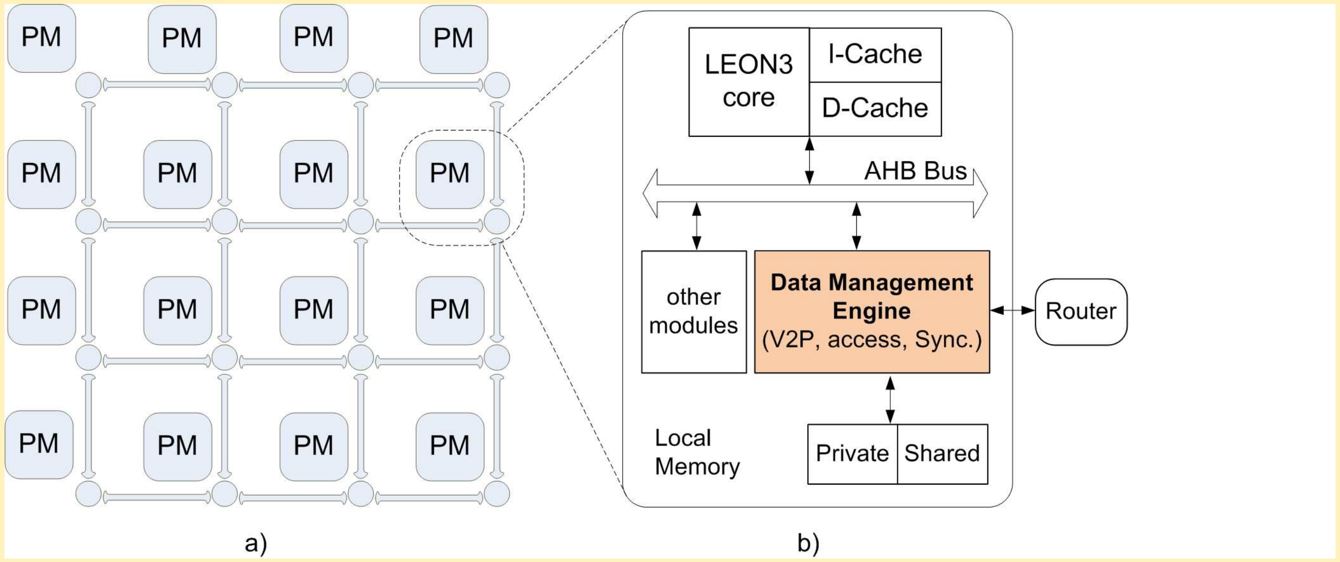


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	... Size of translation table per node
N	... Number of nodes
M_N	... Memory per node
P_S	... Page size
M_T	... Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}
- 1 GB total memory - 2^{30}
- Page size 1KB - 2^{10}
- Translation table: 1 M entries - 2^{20}

Virtual Address Space Translation

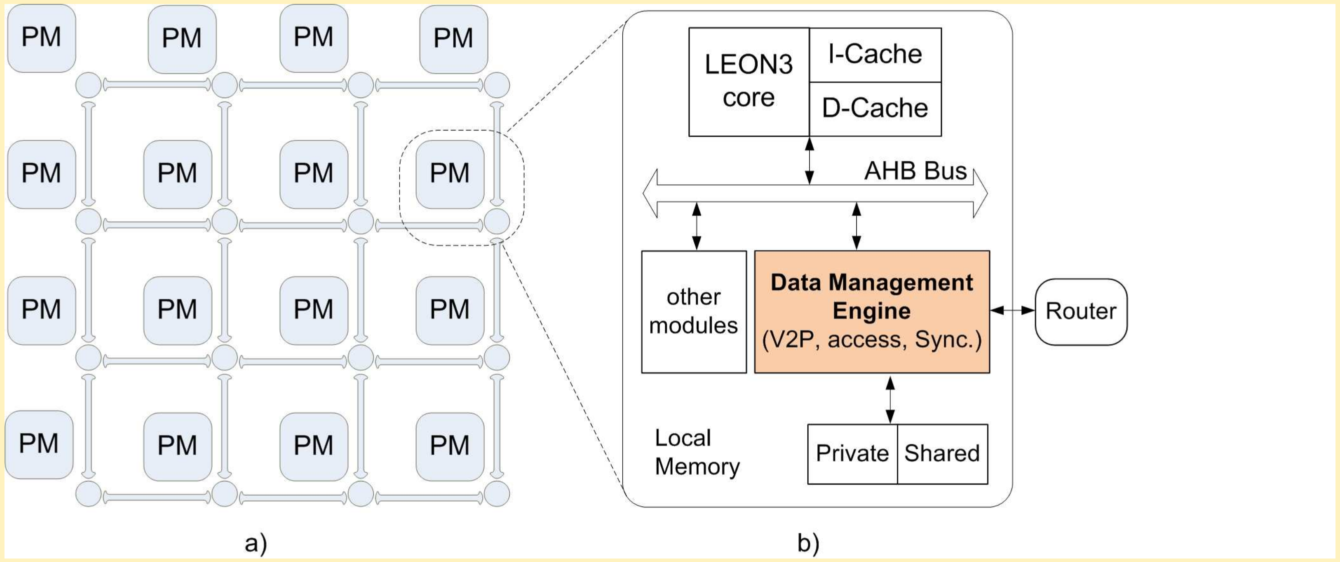


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	...	Size of translation table per node
N	...	Number of nodes
M_N	...	Memory per node
P_S	...	Page size
M_T	...	Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}
- 1 GB total memory - 2^{30}
- Page size 1KB - 2^{10}
- Translation table: 1 M entries - 2^{20}
- 1 entry: 6+14=20 bit $\rightarrow$ 3 Byte

Virtual Address Space Translation

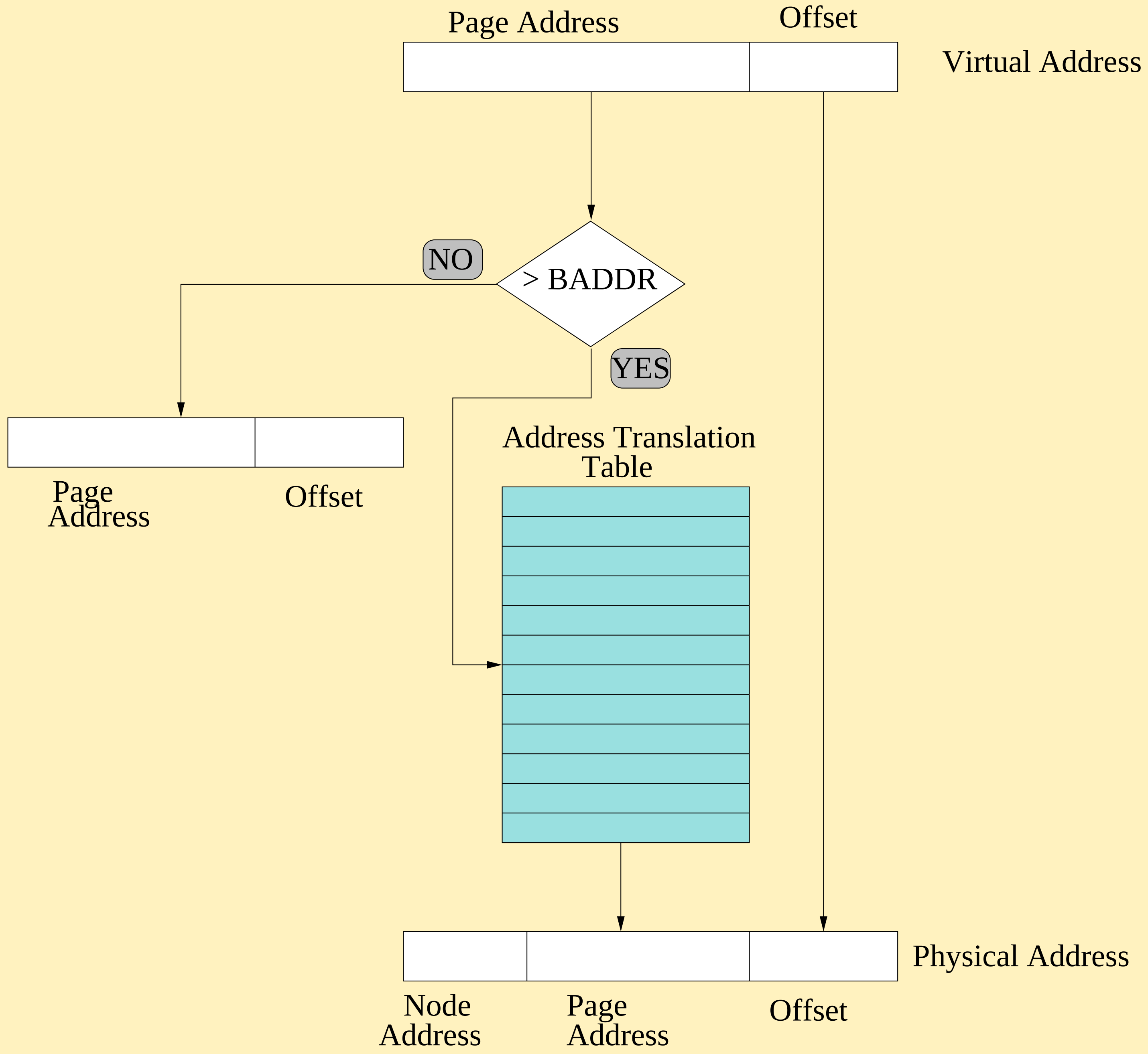


$$S_T = (\log_2 N + \log_2 M_N - \log_2 P_S) \cdot (M_T / P_S)$$

S_T	... Size of translation table per node
N	... Number of nodes
M_N	... Memory per node
P_S	... Page size
M_T	... Total memory size

- 64 nodes - 2^6
- 16 MB/node - 2^{24}
- 1 GB total memory - 2^{30}
- Page size 1KB - 2^{10}
- Translation table: 1 M entries - 2^{20}
- 1 entry: 6+14=20 bit $\rightarrow$ 3 Byte
- $\Rightarrow$ 3MB/node (18.75%) for translation table

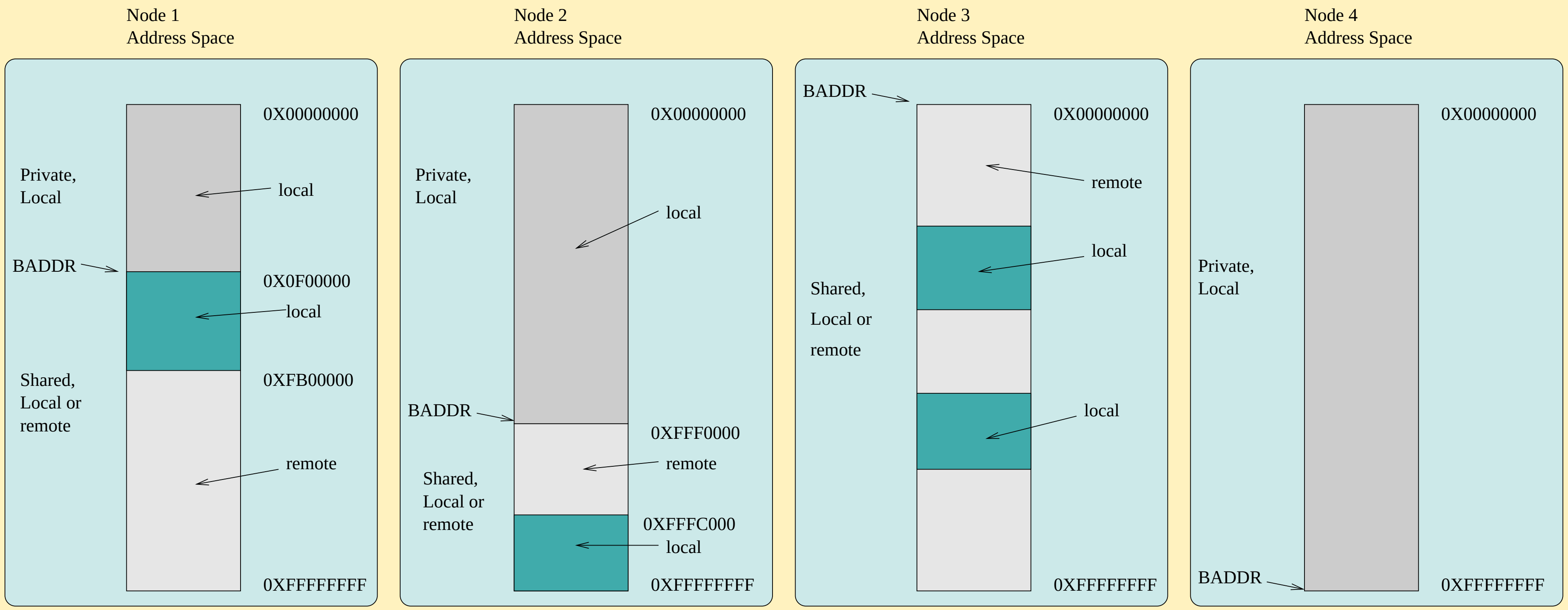
DME Virtual Address Translation



Supported Memory Partitions

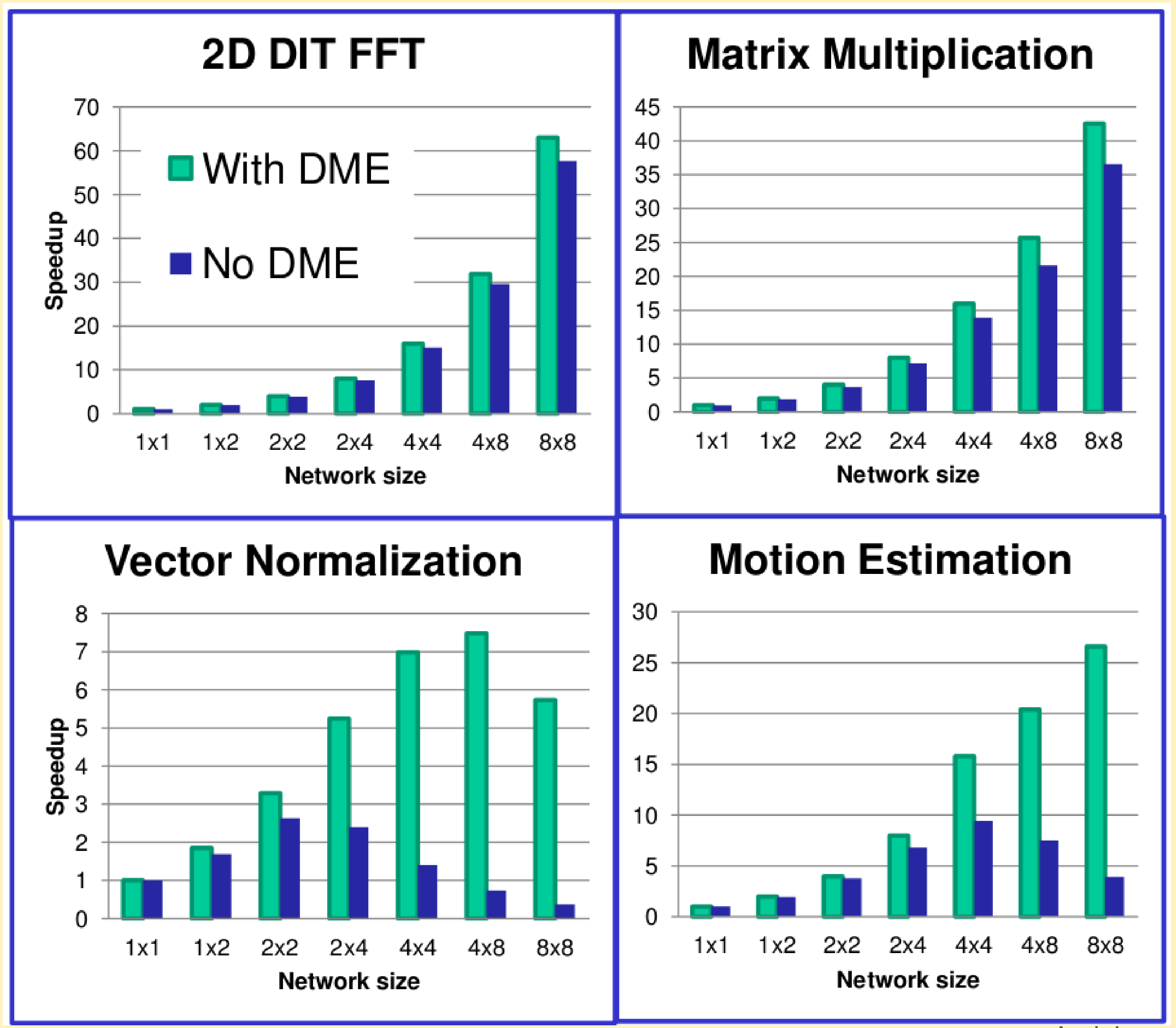
local	private	physical	Supported
local	private	virtual	-
local	shared	physical	-
local	shared	virtual	Supported
remote	private	physical	-
remote	private	virtual	-
remote	shared	physical	-
remote	shared	virtual	Supported

Address Space Management

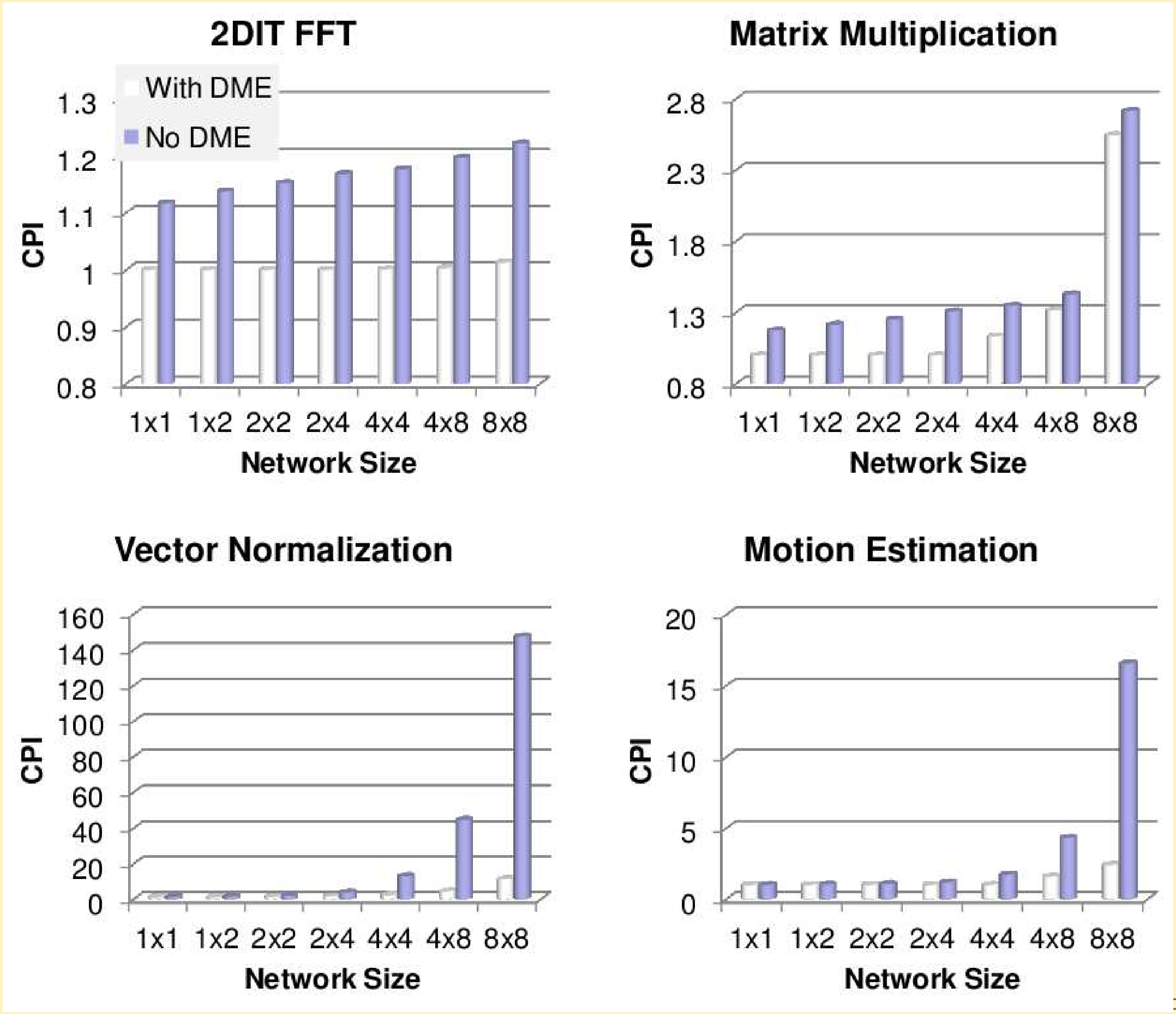


Experiments and Results

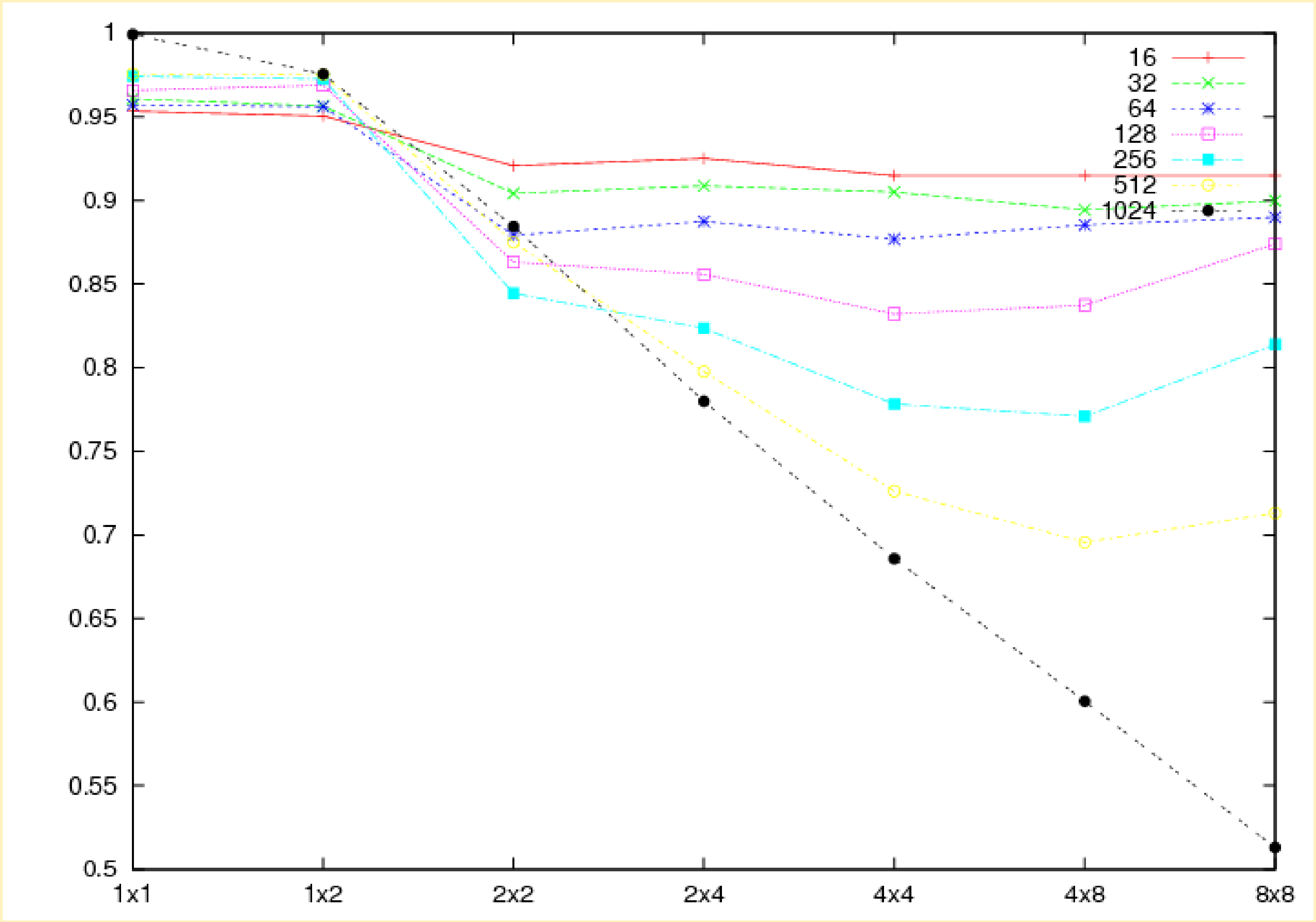
DME Speedup Comparison



DME Cycle per Instruction Comparison

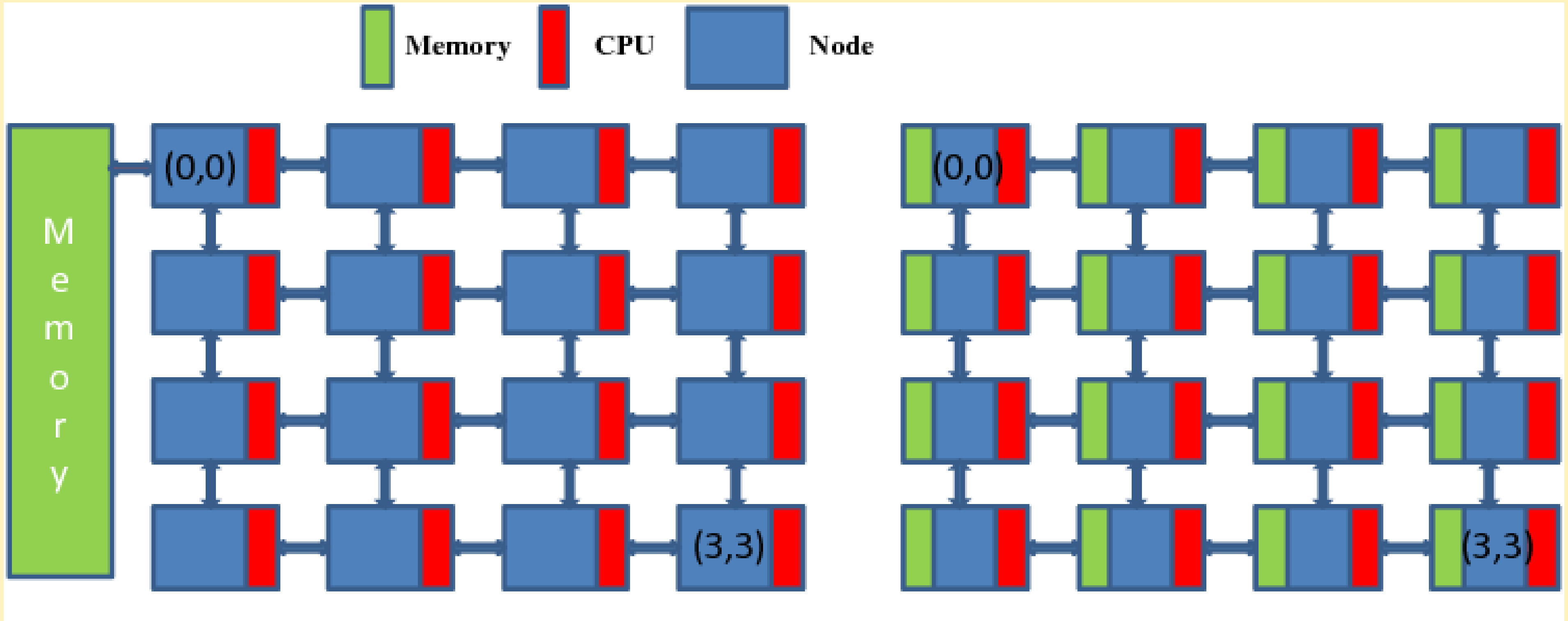


Scalability of Relaxed Memory Consistency



Execution time of Release Consistency normalized to Sequential Consistency for a BitCount example.

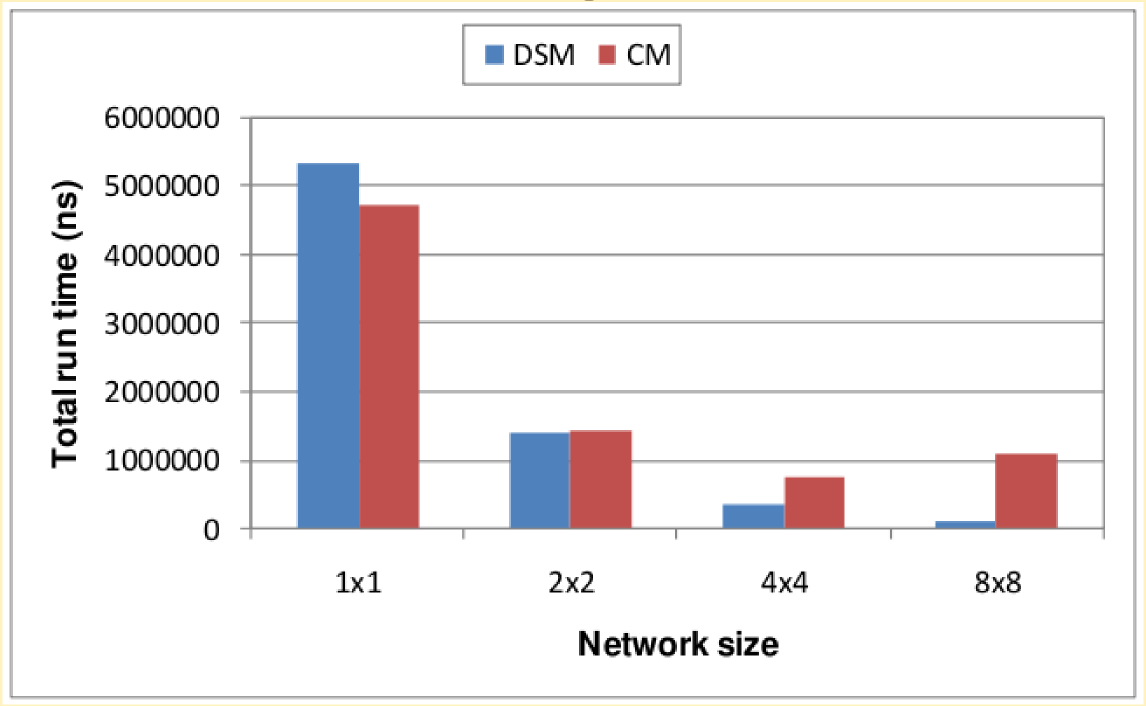
Experiment: Central vs. Distributed Shared Memory

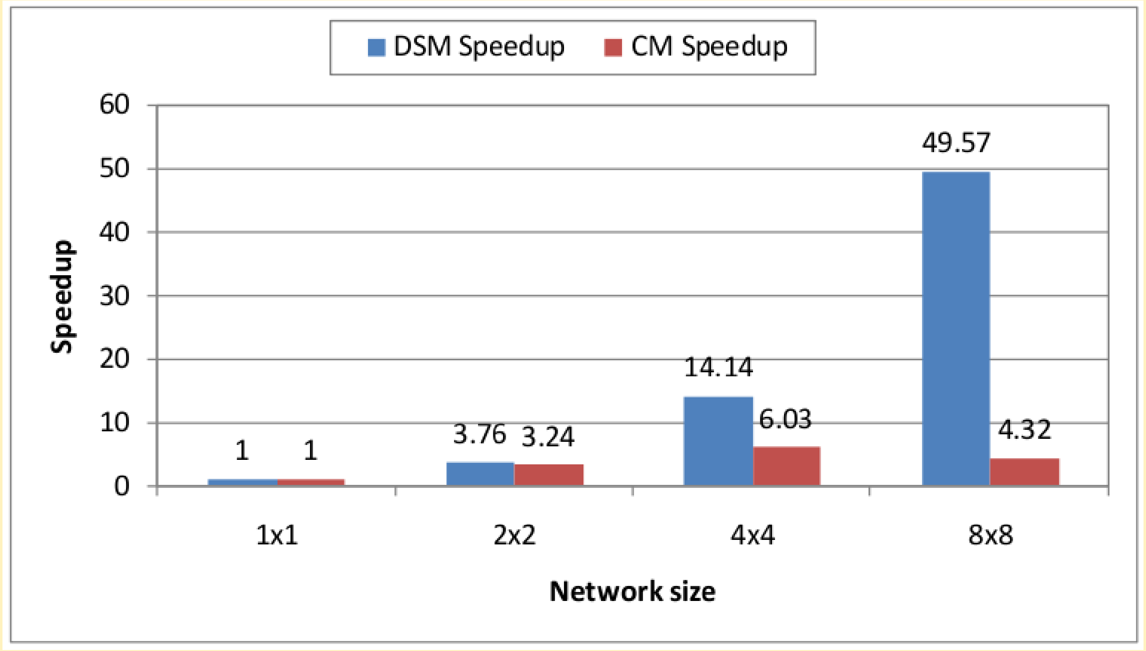


(a) Centralized shared memory (CM)

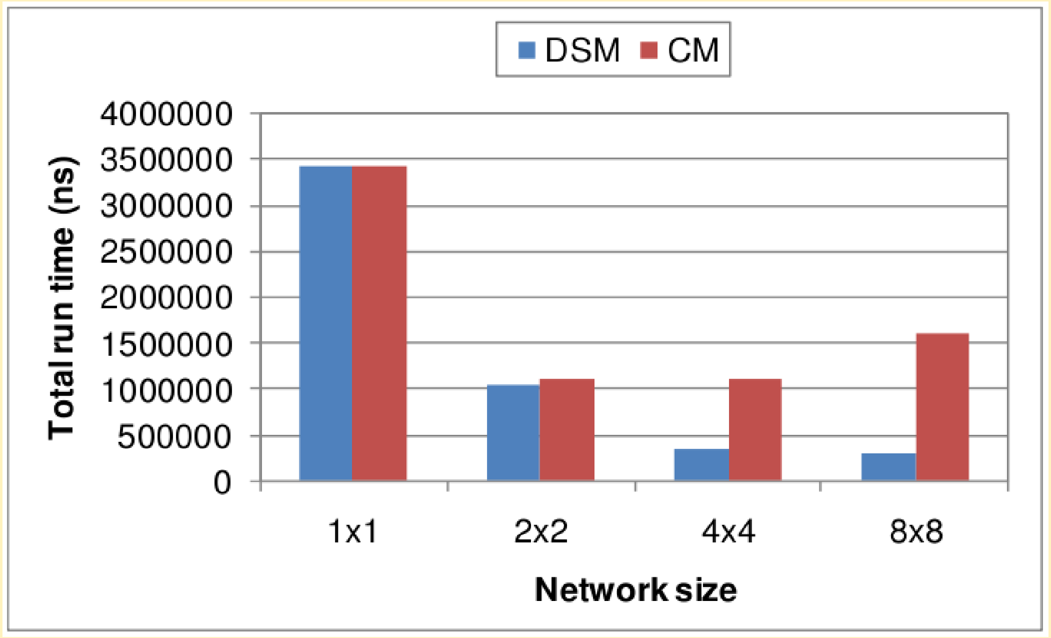
(b) Distributed shared memory (DSM)

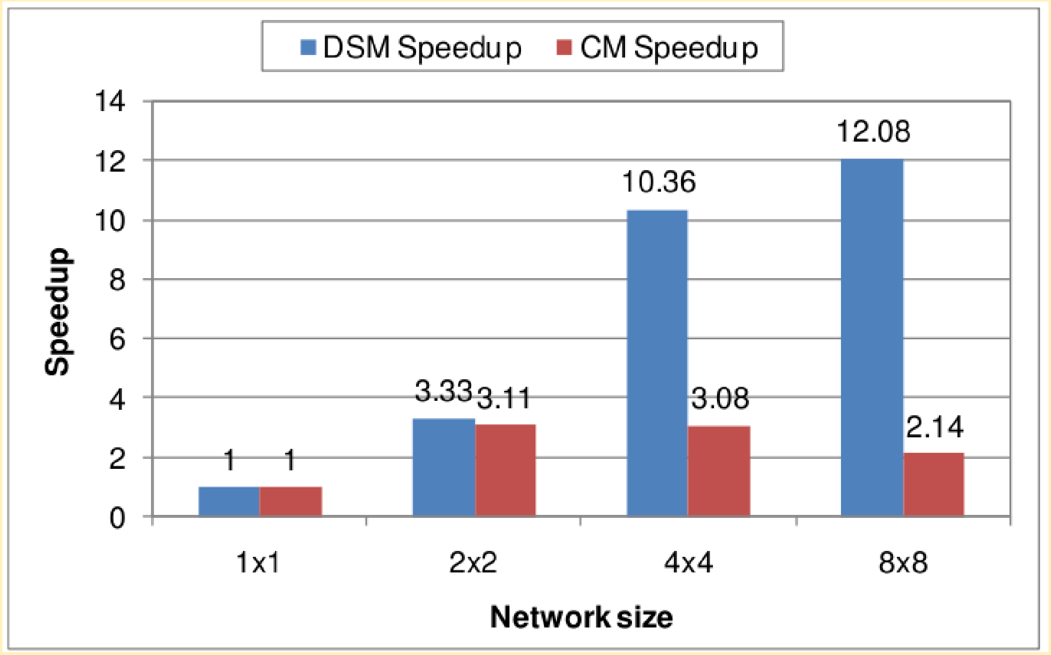
Central vs. Distributed Shared Memory: Matrix Multiplication





Central vs. Distributed Shared Memory: FFT





Summary

- After computation (multi-core) and communication (NoC), memory access must be parallelized, made flexible and scalable
- DME parallizes memory handling
- DME supports
  - Central/distributed memory
  - Private/shared
  - Physical/virtual address space
- DME features
  - Synchronization support
  - Cache coherence protocols
  - Memory consistency support
  - Message passing communication
  - Dynamic memory allocator