LECTURE 8

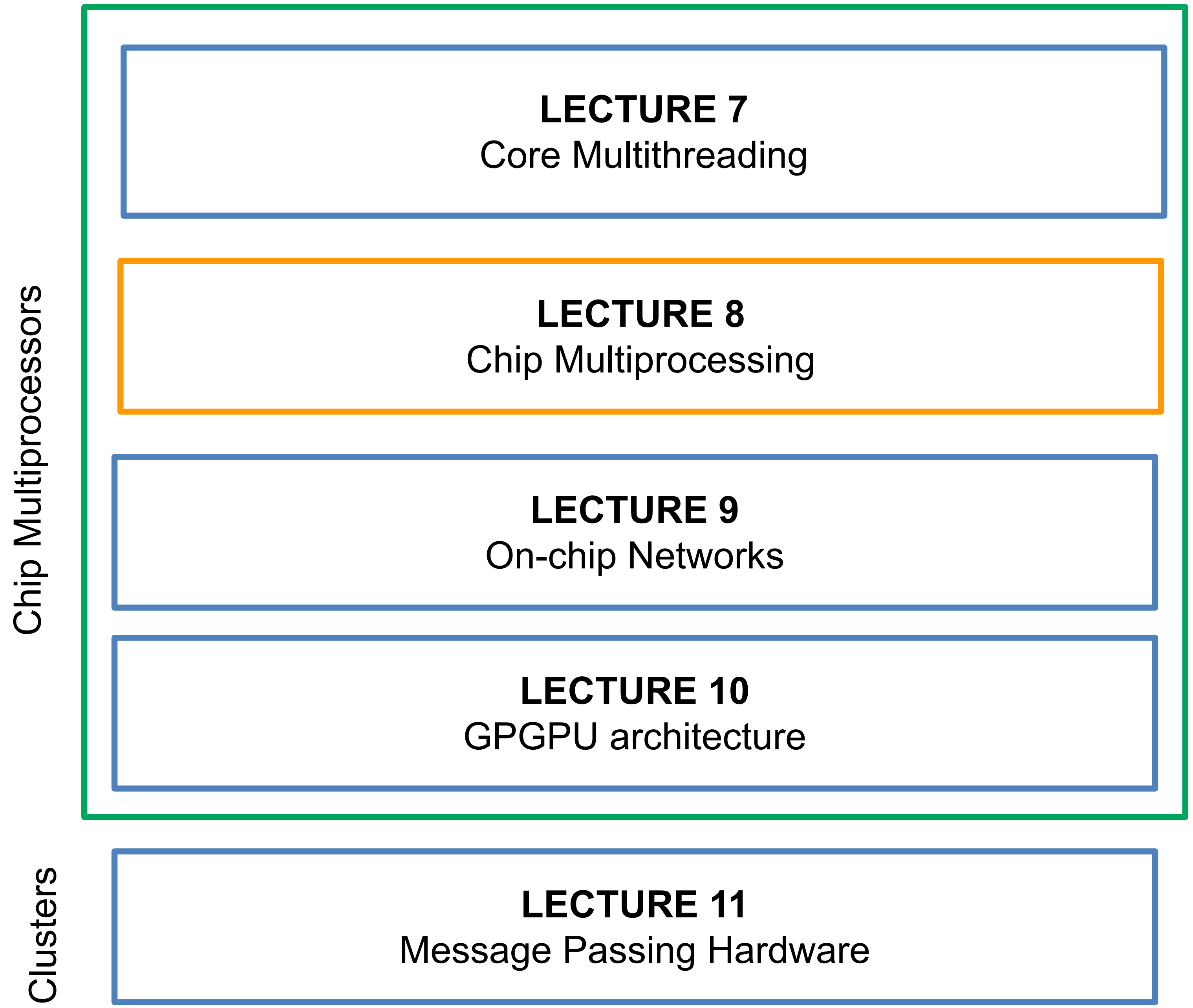
Chip Multiprocessors

Miquel Pericàs
EDA284/DIT361 - 2019/2020

What's cooking

1. **Lectures**
   - Today: **Chip Multiprocessing** (8h-10)
   - Tuesday (Feb 18): Guest Lecture by Ioannis Sourdis on **On-chip Networks** (10h-12h)
2. **Lab session**
   - Friday (8h - 12h @ ED3507), Intro to GEM5
3. **Practice Session**
   - Tuesday (Feb 18) on ccNUMA + Multithreading
   - Exercises will be published in the coming days

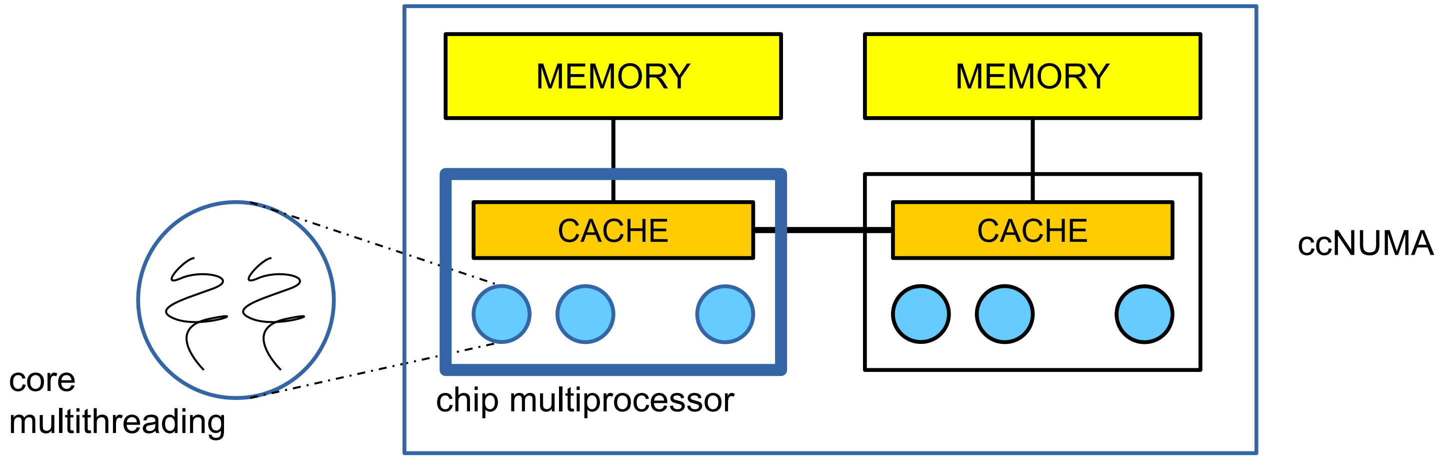
Lectures 7 - 11 Overview



OUTLINE (Lecture 7)

- Heterogeneous CMPs
- CMP Memory Architectures
- CMP Interconnection networks

Chip Multiprocessors (CMPs)



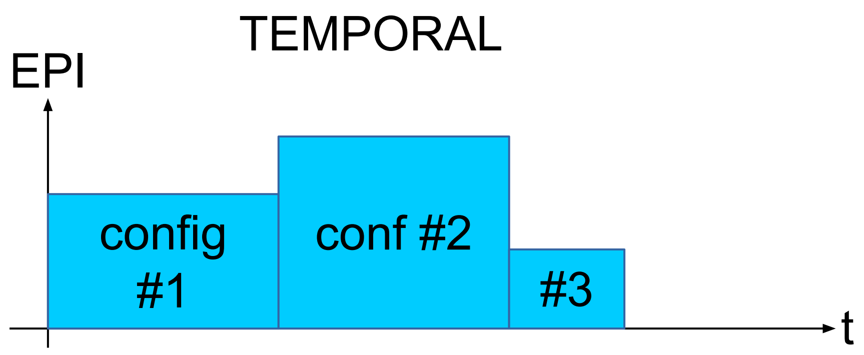
- **CMPs have several desirable properties**
  - Design simplicity: build one core, test it and replicate it
  - Improved power scalability: increases linearly with #cores
  - Low-latency inter-core communication
    - enables fine-grained thread parallelism
    - sharing of data via cache
  - Modularity and customization: different number of cores for different markets

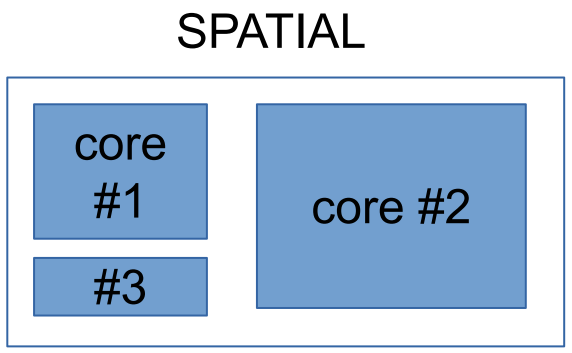
- **CMPs can be homogeneous or heterogeneous**
  - Depends on whether the cores are identical or not
  - Heterogeneity enables further customization and improved energy efficiency

CMPs with heterogeneous cores

- **Workloads have different characteristics**
  - “manycores”: large number of small cores (applications with high thread count)
  - “multicores”: small number of large cores (applications composed of single thread or limited thread count)
  - In practice: mix of workloads
  - Furthermore, most parallel applications have parallel and serial sections (remember: Amdahl's law)

- **Hence, heterogeneity required**
  - temporal: for example, by throttling energy per instruction (EPI)
  - spatial: different types of cores with variation in functionality





CMPs with heterogeneous cores

- **Performance asymmetry (temporal)**
  - using homogeneous cores and DVFS, or processor with mixed cores but same ISA (e.g., in-order + out-of-order cores)
  - variable resources: e.g., adapt size of cache by gating off power to cache banks
  - speculation control (low branch prediction code): throttle the number of in-flight instructions (will reduce activity factor)

Method	EPI Range	Time to vary EPI
DVFS	1:2 to 1:4	100 us, ramp V_{cc}
Variable Resources, eg. turn off cache ways	1:1 to 1:2	1 us, Fill L1
Speculation Control	1:1 to 1:1.4	10 ns, Pipeline flush
Mixed Cores	1:6 to 1:11	10 us, Migrate L2

Amdahl's law in the multicore era

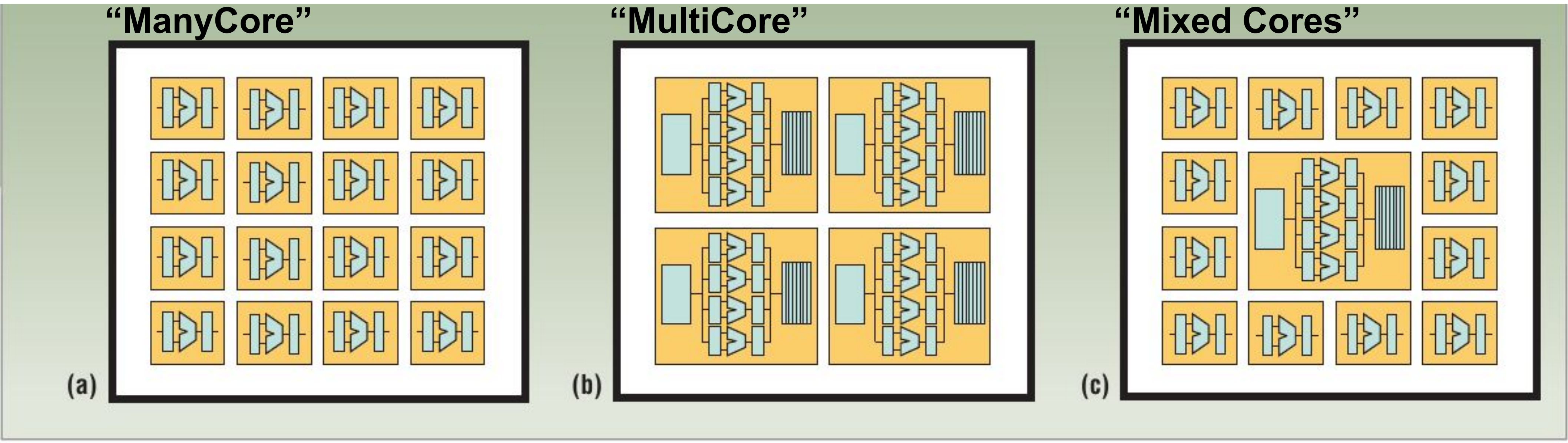


Figure 1. Varieties of multicore chips. (a) Symmetric multicore with 16 one-base core equivalent cores, (b) symmetric multicore with four four-BCE cores, and (c) asymmetric multicore with one four-BCE core and 12 one-BCE cores. These figures omit important structures such as memory interfaces, shared caches, and interconnects, and assume that area, not power, is a chip's limiting resource.

parallel fraction — *total chip resources* — *resources per core (r<n)*

$$\text{Speedup}_{\text{symmetric}}(f,n,r)=\frac{1}{\underbrace{\frac{1-f}{perf(r)}}_{\text{serial}}+\underbrace{\frac{f\cdot r}{perf(r)\cdot n}}_{\text{parallel}}}$$

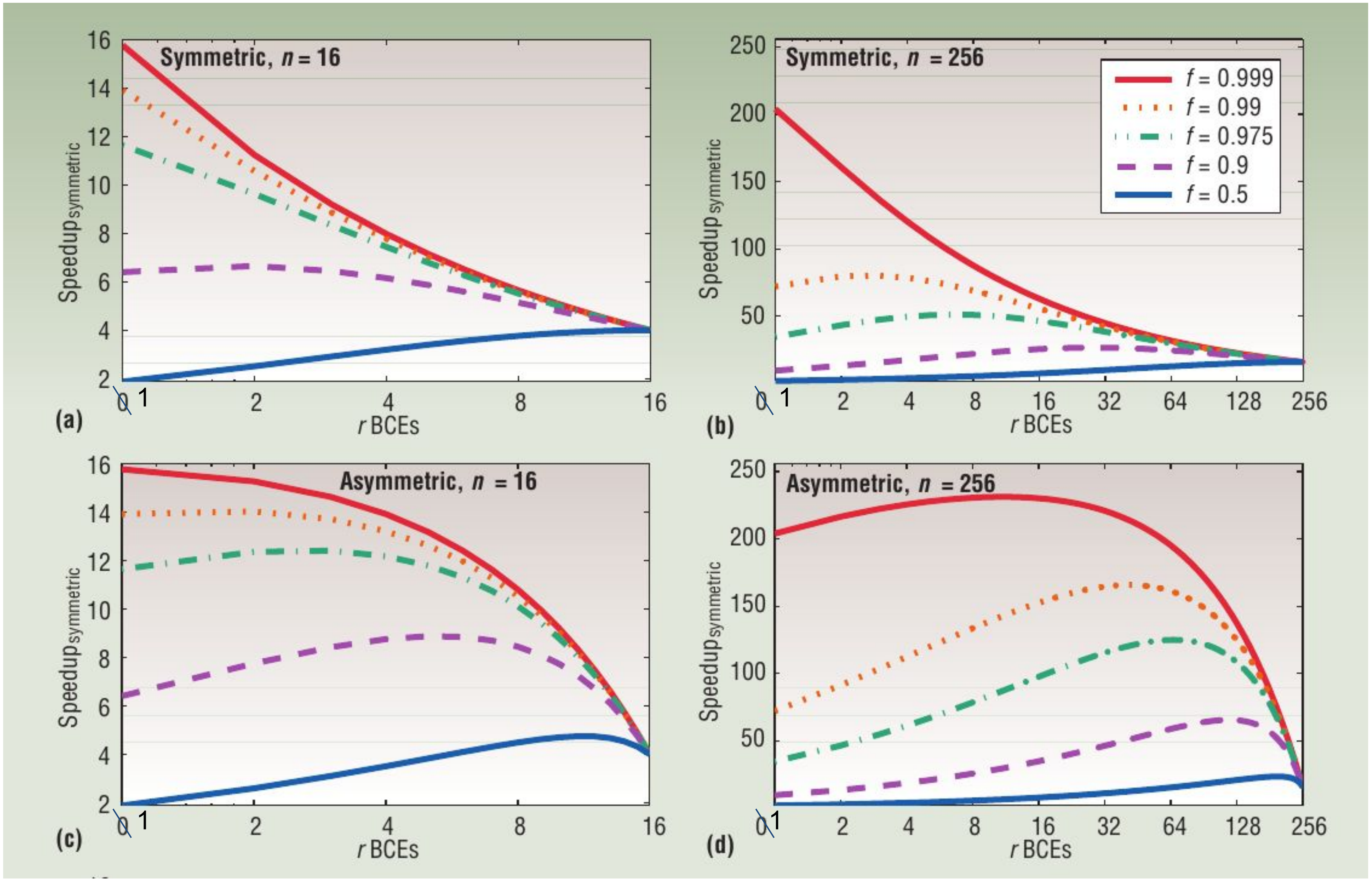
$$\text{Speedup}_{\text{asymmetric}}(f,n,r)=\frac{1}{\underbrace{\frac{1-f}{perf(r)}}_{\text{1 big core (r)}}+\frac{f}{perf(r)+\underbrace{n-r}_{\text{(n-r) x small cores (1)}}}}$$

SYMMETRIC (multicore, manycore)
E.g.: (a) 16 small, (b) 4 large

ASYMMETRIC (mixed)
E.g.: (c) 1 large + 12 small

M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," in Computer, vol. 41, no. 7, pp. 33-38, July 2008.

Amdahl's law in the multicore era



assumption: **`perf(r) = sqrt(r)`** ; known as Pollack's rule

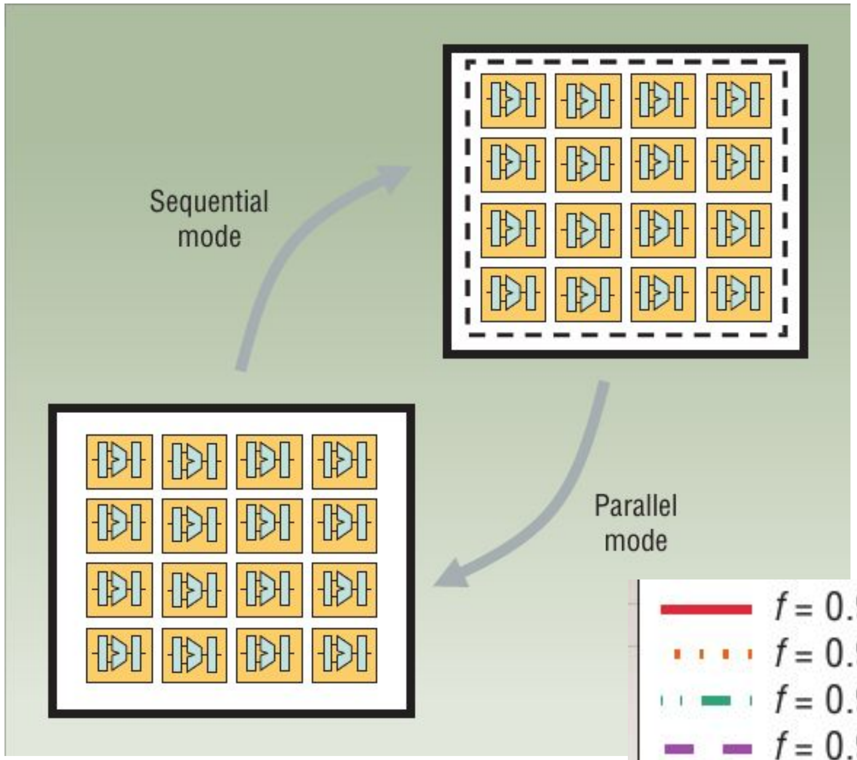
Dynamic multicore

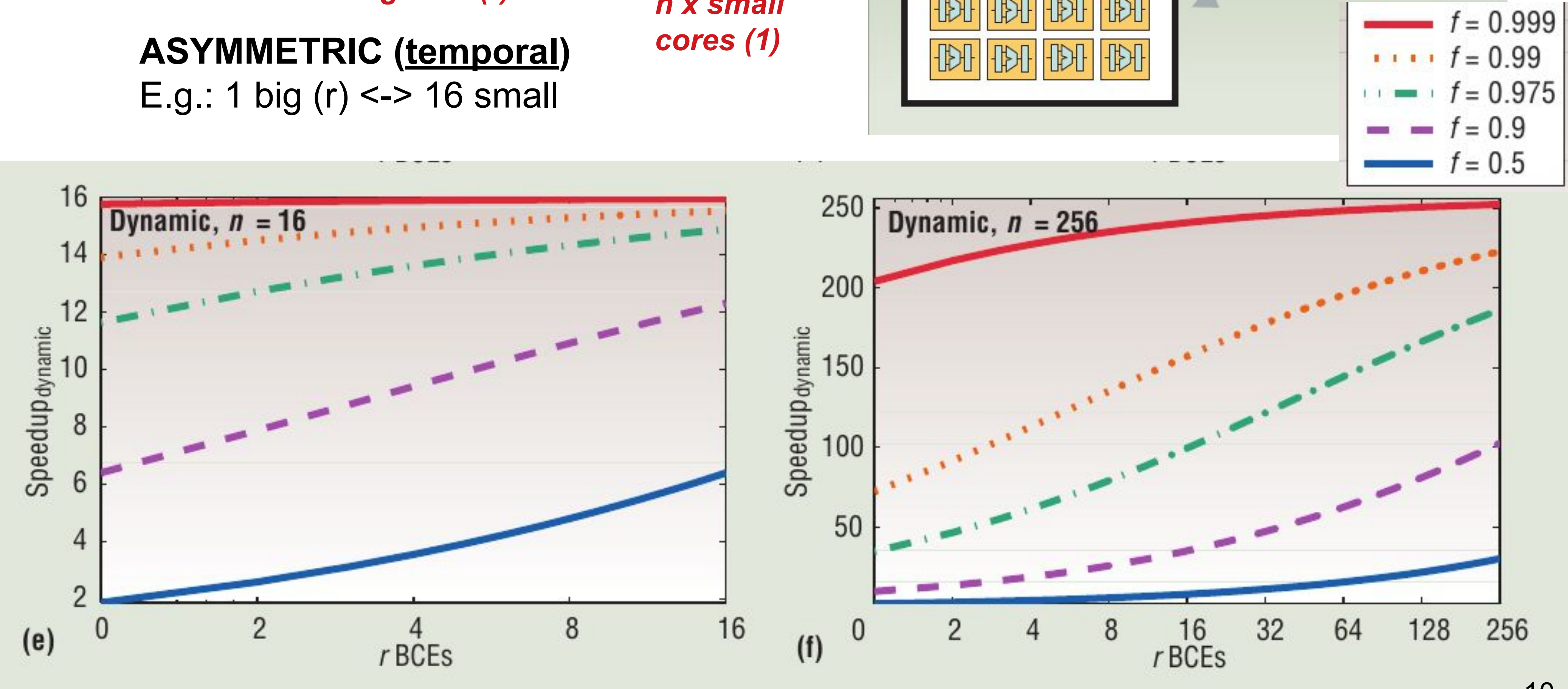
$$\text{Speedup}_{\text{dynamic}}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{n}}$$

1 big core (r)

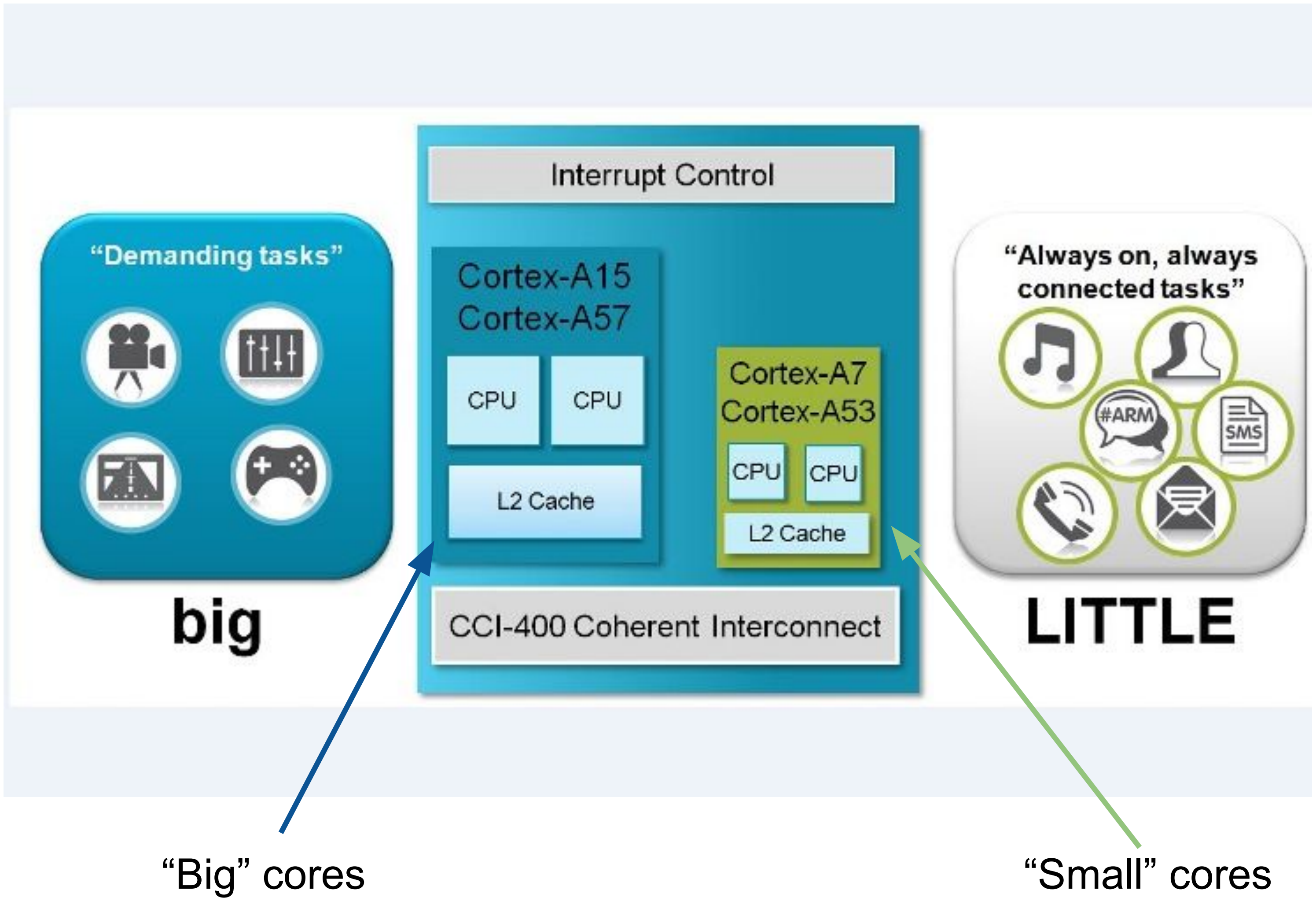
n x small cores (1)

ASYMMETRIC (<u>temporal</u>)
E.g.: 1 big (r) <-> 16 small





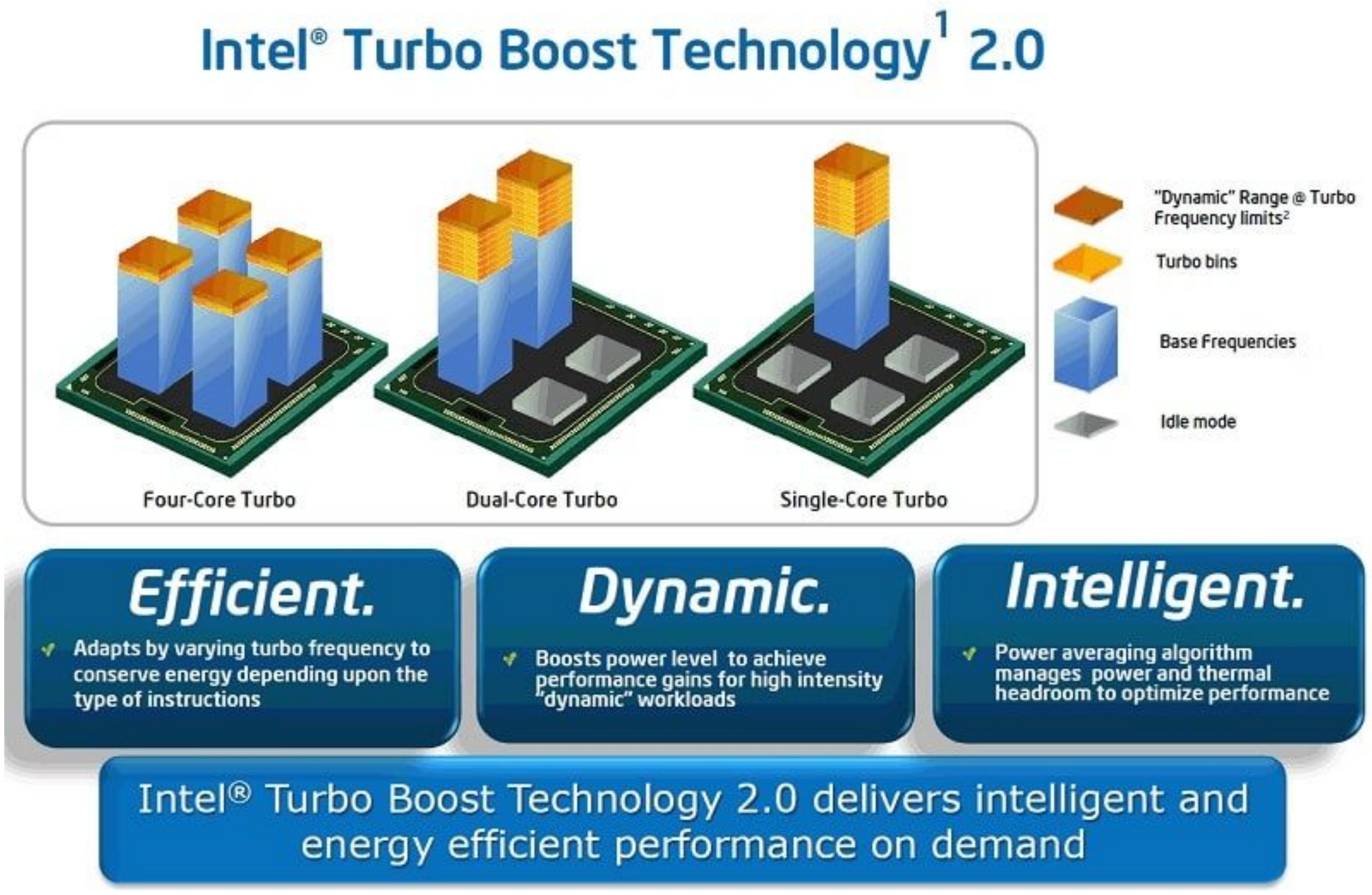
Mixed Cores example:
ARM's big.LITTLE Architecture (2011+)



- Same ISA, different performance/power characteristics
- LITTLE cores are less powerful, but much more efficient

Dynamic multicore example: Intel Turbo Boost (2008+)

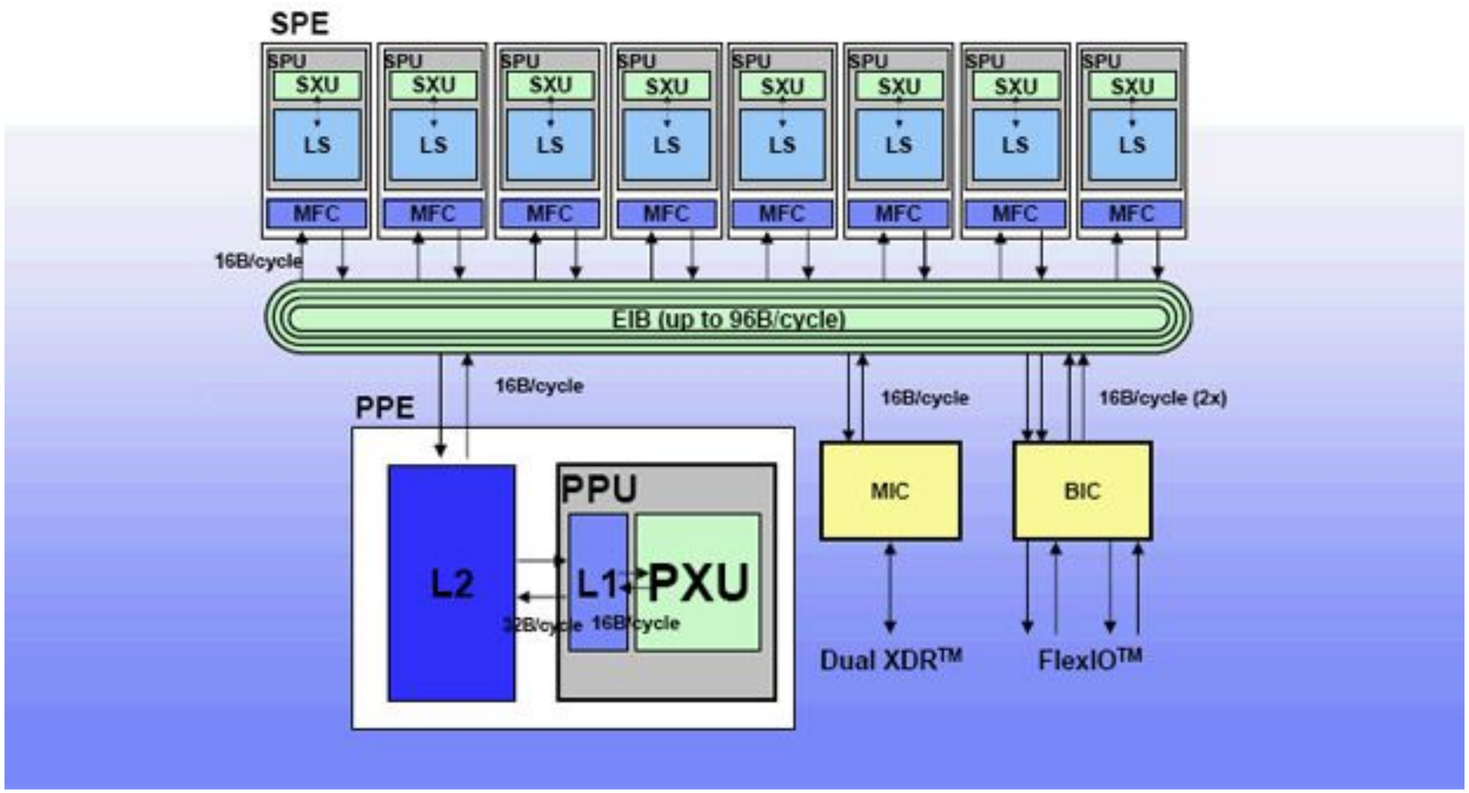
Dynamically scales performance depending on number of running cores and TDP (thermal design power)


Intel® Turbo Boost Technology[1] 2.0
"Dynamic" Range @ Turbo Frequency limits[2]
Turbo bins
Base Frequencies
Idle mode
Four-Core Turbo
Dual-Core Turbo
Single-Core Turbo
Efficient.
✓ Adapts by varying turbo frequency to conserve energy depending upon the type of instructions
Dynamic.
✓ Boosts power level to achieve performance gains for high intensity "dynamic" workloads
Intelligent.
✓ Power averaging algorithm manages power and thermal headroom to optimize performance
Intel® Turbo Boost Technology 2.0 delivers intelligent and energy efficient performance on demand

CMPs with heterogeneous cores

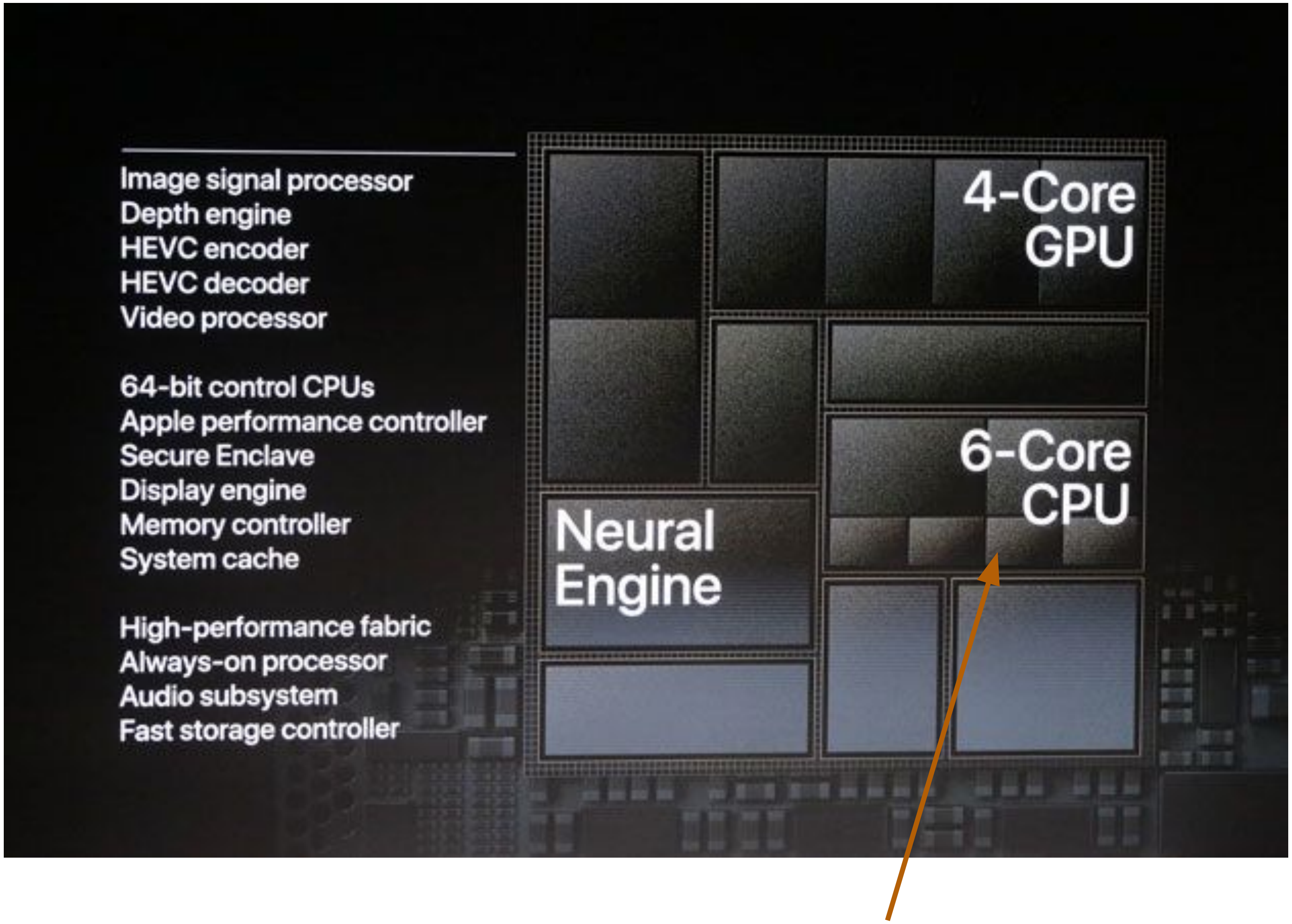
- **Functional asymmetry (spatial)**
  - **Use heterogeneous cores**
    - e.g., GP cores, graphics processors, cryptography, vector cores, floating-point co-processors
    - heterogeneous cores may be programmed differently (both high level and ISA)
    - mechanisms must exist to transfer activity for one core to another
    - **fine-grain**: in the case of floating point co-processor, use ISA extension (e.g, floating point instruction, vector extensions..)
    - **coarse grain**: transfer the computation from one core to another using APIs (e.g. CUDA, OpenACC..)
  - **Goals:**
    - save power by using cores with different power/performance characteristics for different phases of execution
    - higher performance
    - better usage of increased chip area at limited power

Functional Asymmetry Example: IBM Cell Processor (2006)



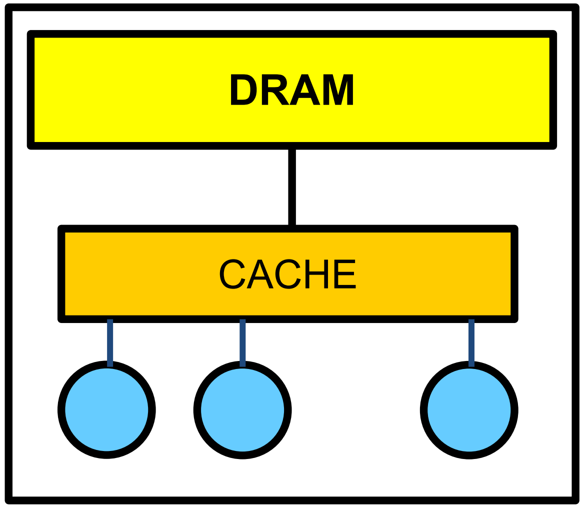
- **One PowerPC processing element (PPE)**
  - 2-way SMT PowerPC core
- **Plus 8 synergistic processing elements (SPEs)**
  - SPE is 2-issue in-order processor
  - two SIMD instructions can be issued in each cycle (vectors)
  - no coherence support between SPE and PPE (data transfers are explicitly programmed)
  - PPE and SPE execute different ISAs

Functional Asymmetry Example: Apple A12 Bionic (2018)

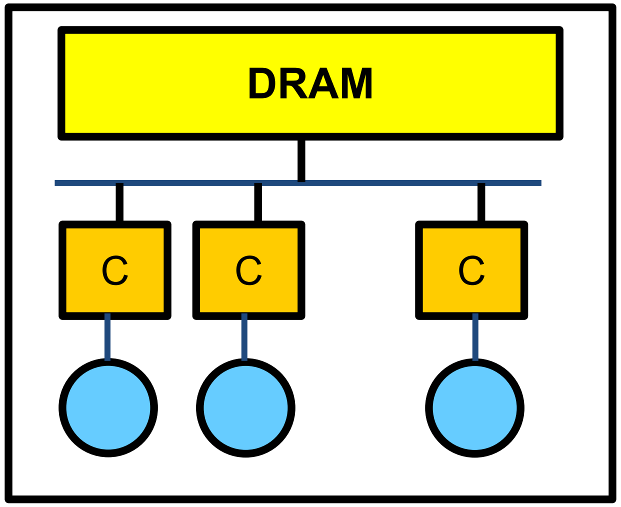


2x Vortex (High Performance) +
4x Tempest (Energy Efficient)

Design options for Last Level Caches



shared cache



private cache

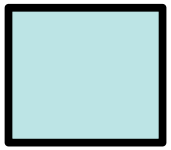
- **Private Cache**
  - Preferable when shared cache latency is too large
  - Provides stronger isolation between cores
    - No destructive interference possible (per-core capacity is statically partitioned)
    - But also no constructive sharing
    - Good for multi-user workloads (provides fairness)

Shared vs Private cache

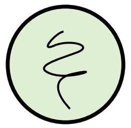
- **In general shared cache preferable**
  - No coherence problem
  - Dynamic re-allocation of capacity among cores (LRU)
  - In the presence of sharing, effective capacity is larger than private
  - Two scenarios: <u>constructive</u> (aka “cooperative”) sharing or <u>destructive</u> interference
- **Constructive sharing** (code and data shared among cores): GOOD
  - effective size is greater because of sharing
  - prefetching effects: one core loads data used by a different core
- **Destructive interference** (no sharing): BAD
  - threads do not help each others
  - threads sharing the cache compete with other threads for capacity
  - as the thread count increases, the amount of cache per thread decreases!
  - both misses and contention increase

Constructive vs Destructive sharing

Application

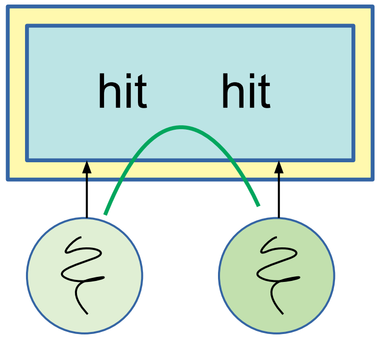


Working Set
:= amount of memory that a process accesses



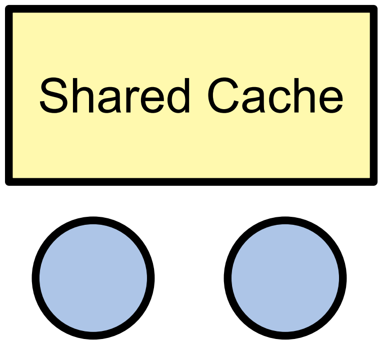
Thread

Constructive Sharing



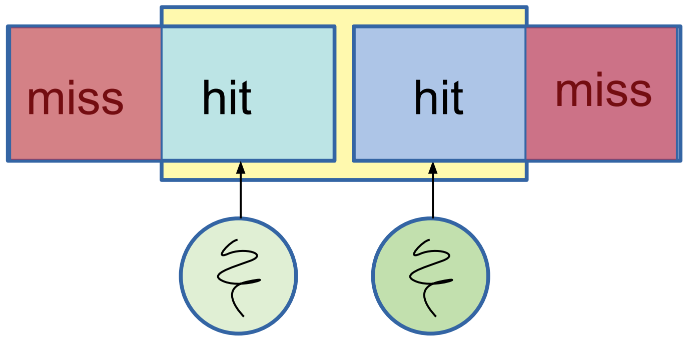
prefetching effect

Architecture



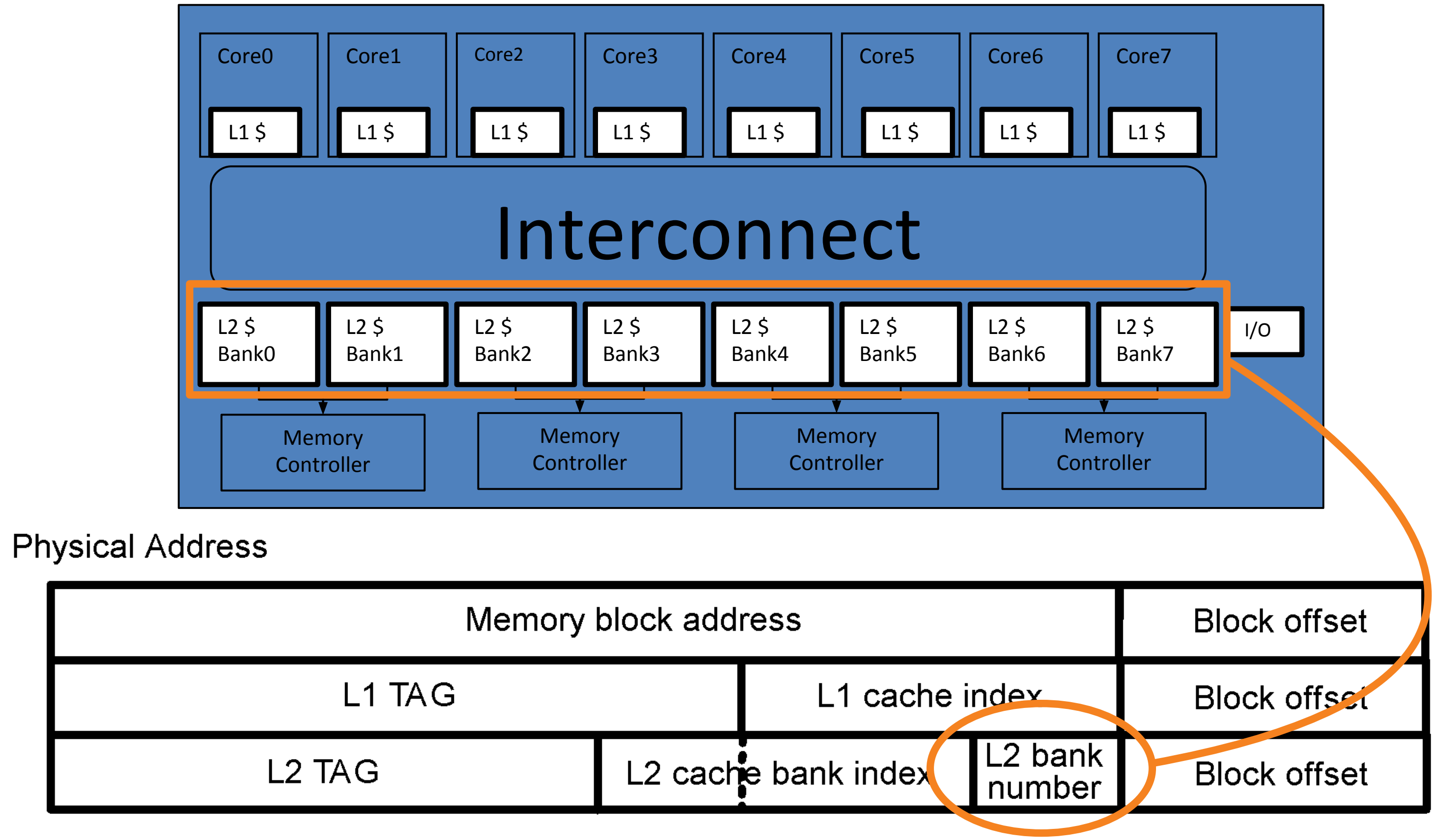
Dual-core Processor with Shared Cache

Destructive Sharing / Interference



threads competing for shared cache space

Shared Cache architecture

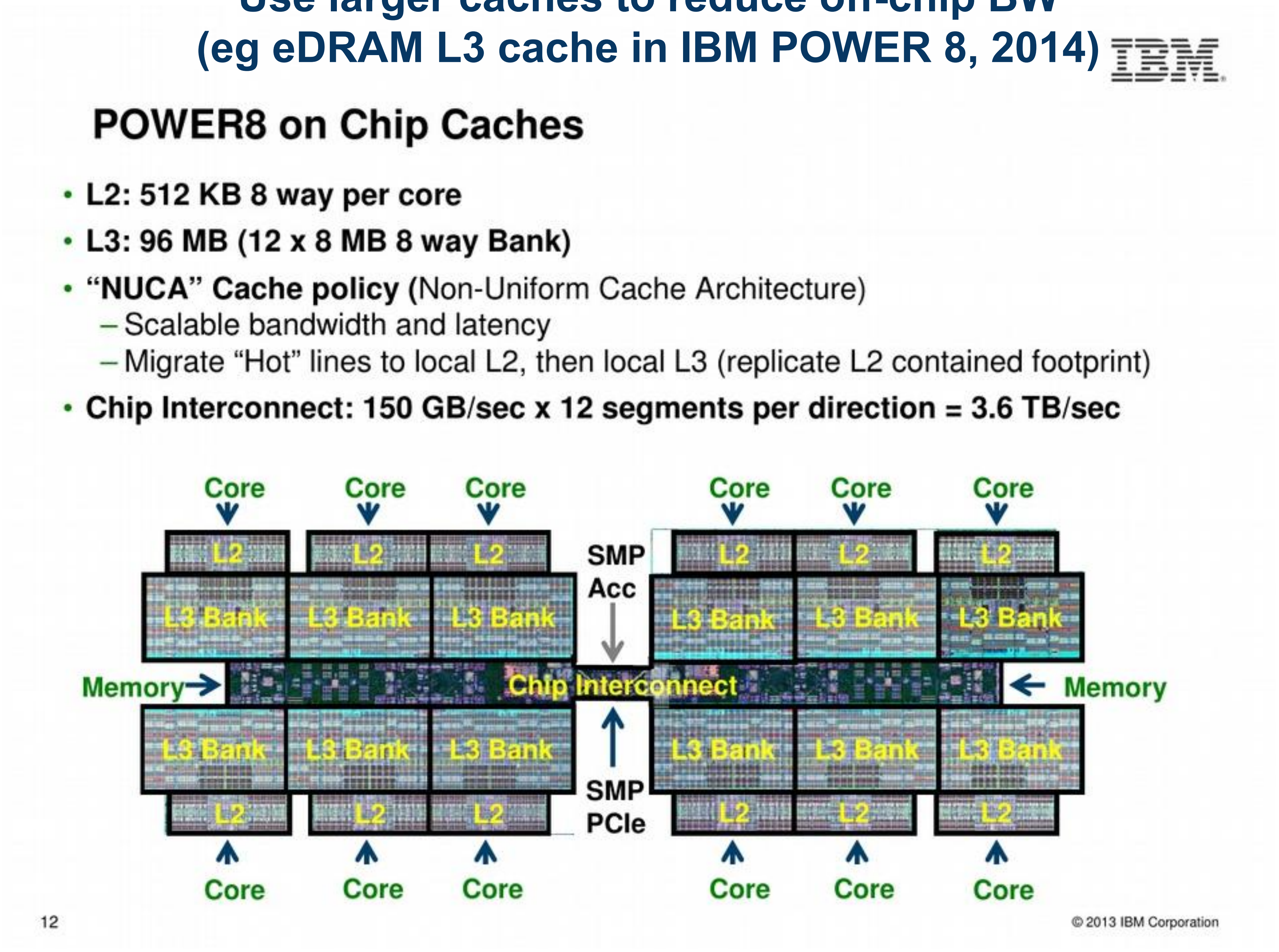


- **S-NUCA** (static non-uniform cache architecture): latency depends on core<->bank
- Due to dynamic packet routing in CMP NoCs, coherence usually enforced by point-to-point directory protocols
- **Directory can use a full presence bit vector per L2 line**
  - N+1 bits for each L2 line (sharers + dirty bit)
  - **EXAMPLE**: 64B cache line, 16 cores → overhead = 17 / (64x8) = ~3%

CMP cache and BW considerations

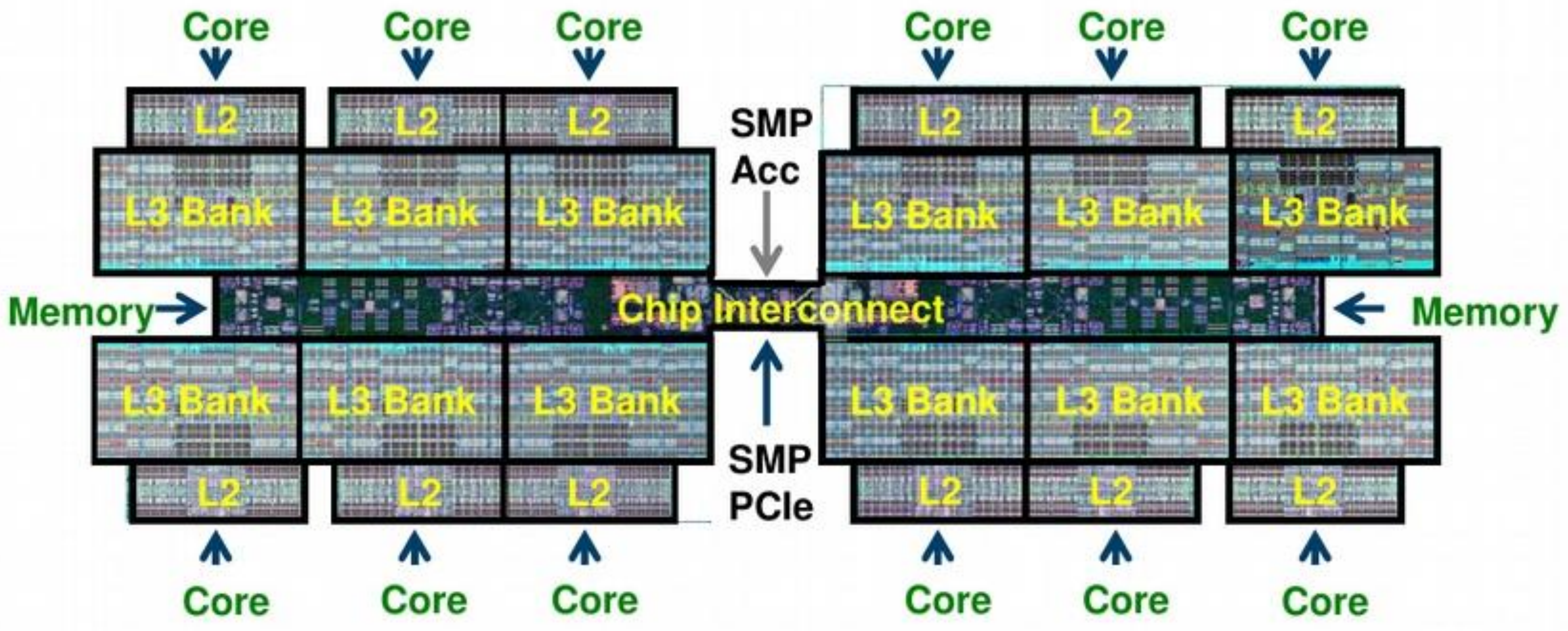
- **Off-chip bandwidth is a critical resource in CMP**
  - limited by pin count, frequency
  - BW requirements grow with number of cores
  - if number of off-chip memory accesses per cycle > memory channels, then queuing delays lead to superlinear memory access latency.
    - Leads to “work-time inflation” → each individual thread needs more cycles while waiting for memory
    - Adding more cores to an already “memory-bound” problem just increases average memory latency. The total throughput remains constant or even decreases.
- **Memory wall problem replaced by bandwidth problem. Potential solutions:**
  - Reduce BW requirement of threads (HW: larger caches, SW: locality-aware programming)
  - Augment available off-chip bandwidth

Use larger caches to reduce off-chip BW (eg eDRAM L3 cache in IBM POWER 8, 2014)

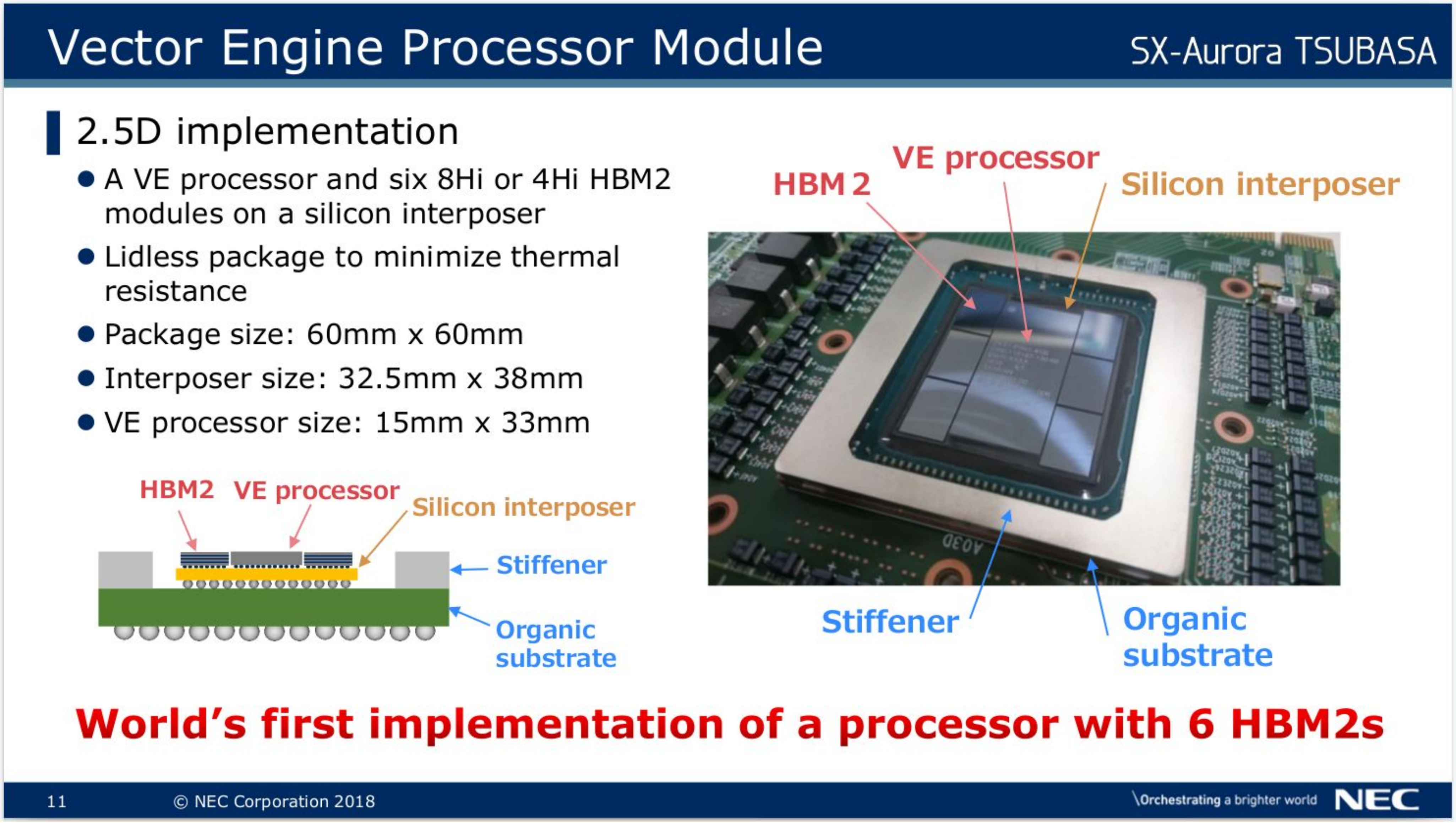


POWER8 on Chip Caches

- L2: 512 KB 8 way per core
- L3: 96 MB (12 x 8 MB 8 way Bank)
- "NUCA" Cache policy (Non-Uniform Cache Architecture)
  - Scalable bandwidth and latency
  - Migrate "Hot" lines to local L2, then local L3 (replicate L2 contained footprint)
- Chip Interconnect: 150 GB/sec x 12 segments per direction = 3.6 TB/sec



..And high BW off-chip memory (e.g., HBM2 in SX-Aurora TSUBASA, 2018)

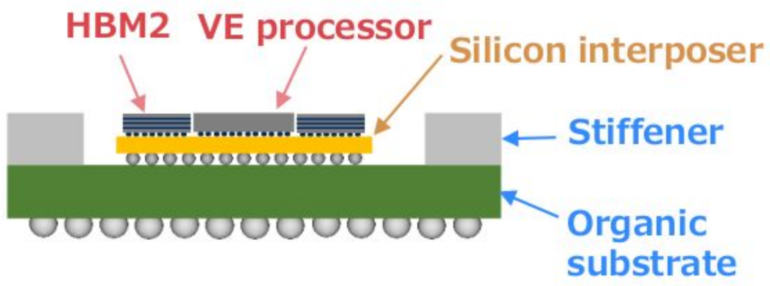


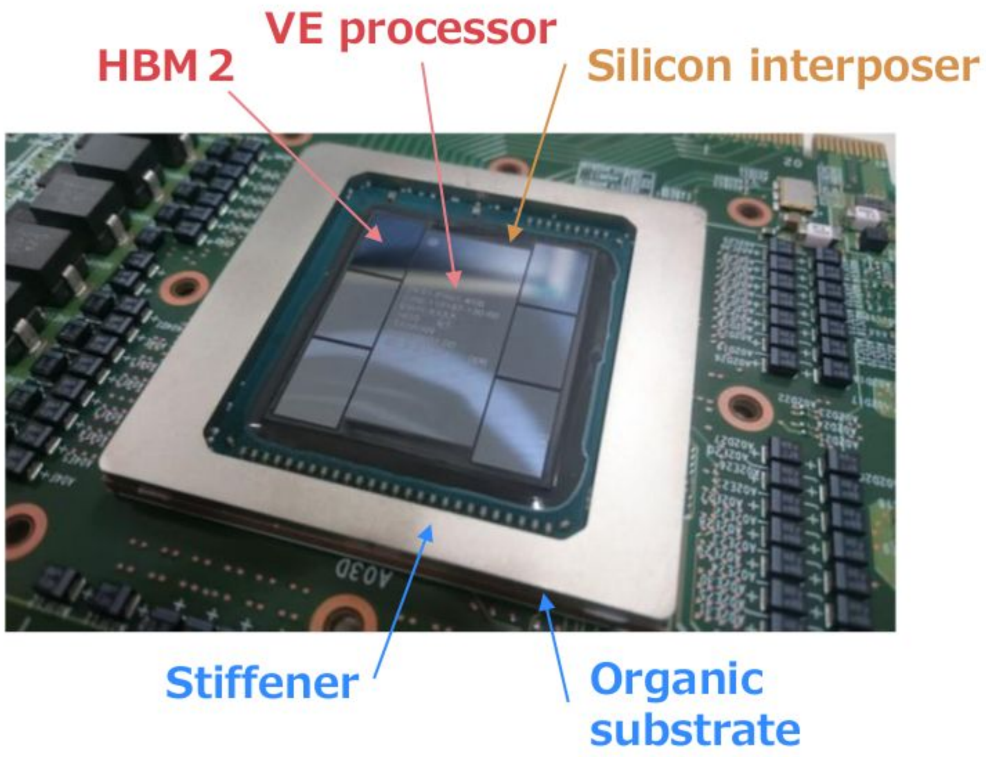
Vector Engine Processor Module

SX-Aurora TSUBASA

2.5D implementation

- A VE processor and six 8Hi or 4Hi HBM2 modules on a silicon interposer
- Lidless package to minimize thermal resistance
- Package size: 60mm x 60mm
- Interposer size: 32.5mm x 38mm
- VE processor size: 15mm x 33mm

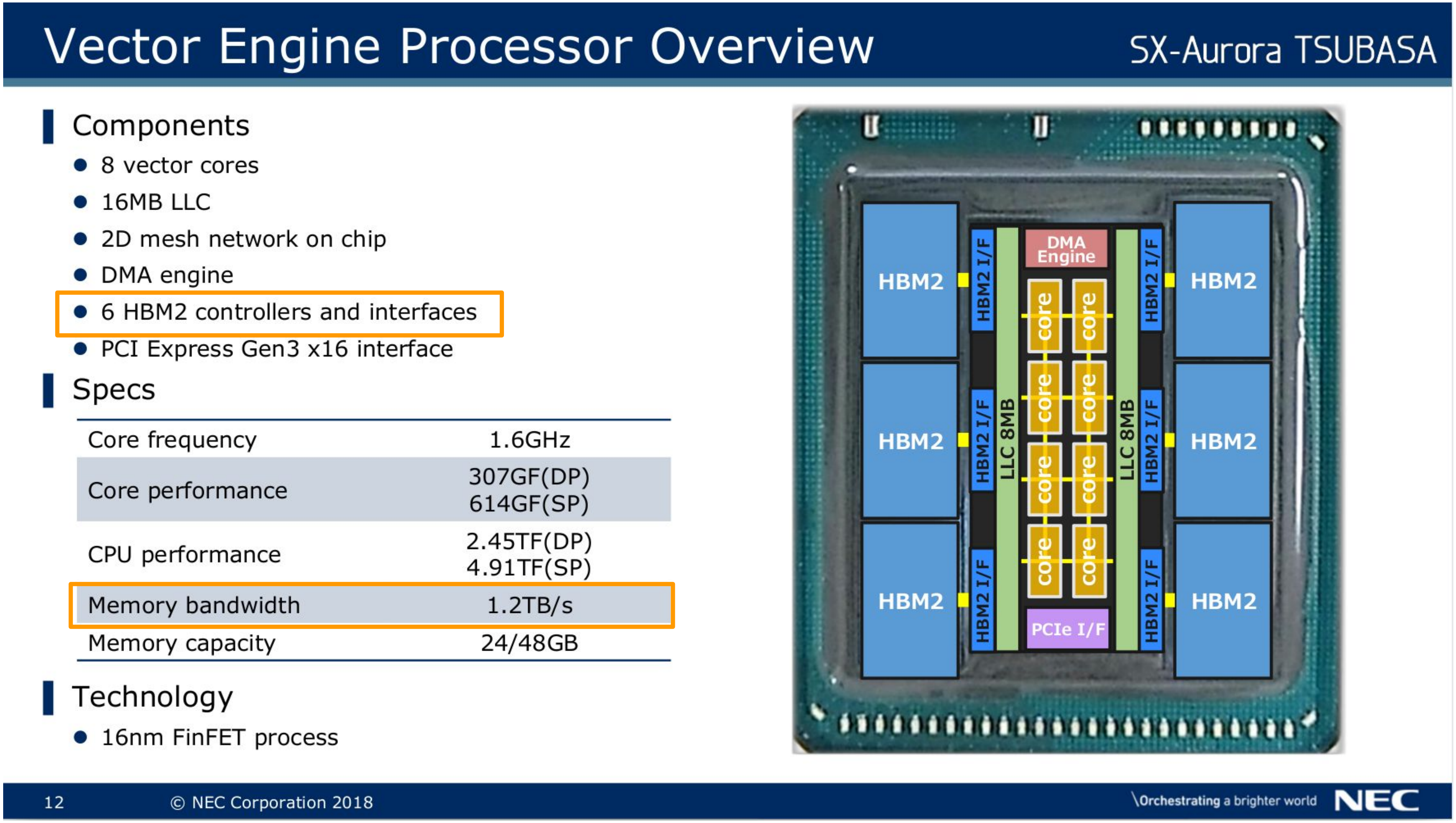




World's first implementation of a processor with 6 HBM2s

11 © NEC Corporation 2018 \Orchestrating a brighter world NEC

..And high BW off-chip memory caches (e.g., HBM2 in SX-Aurora TSUBASA, 2018)



Vector Engine Processor Overview

SX-Aurora TSUBASA

Components

- 8 vector cores
- 16MB LLC
- 2D mesh network on chip
- DMA engine
- 6 HBM2 controllers and interfaces
- PCI Express Gen3 x16 interface

Specs

Core frequency	1.6GHz

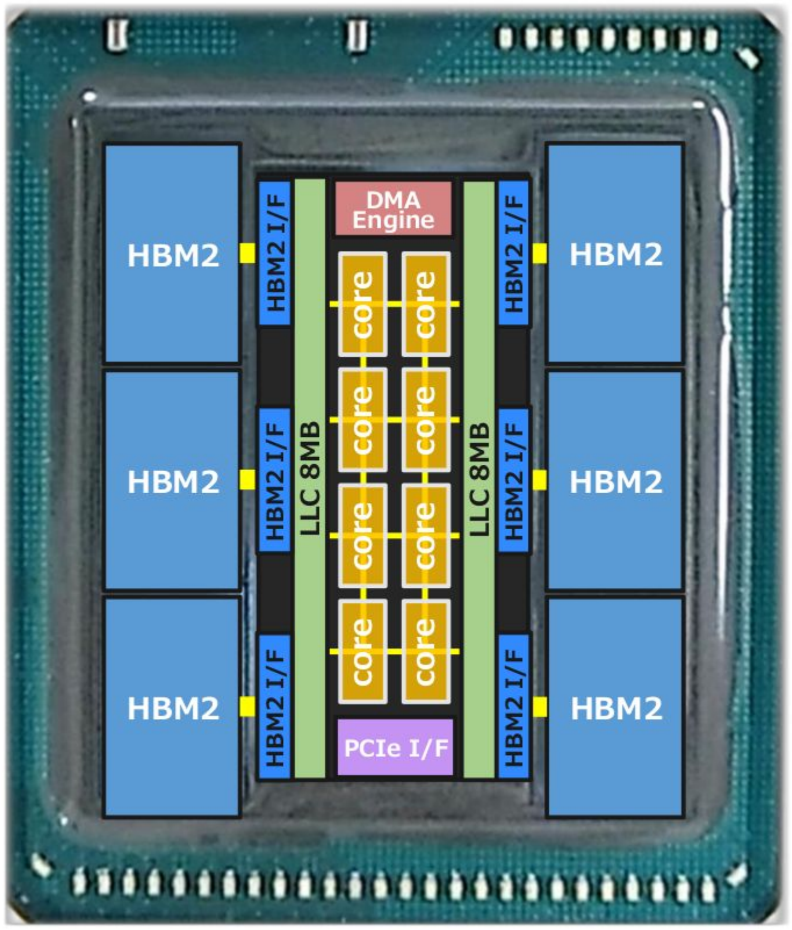
Core performance	307GF(DP)
	614GF(SP)

CPU performance	2.45TF(DP)
	4.91TF(SP)

Memory bandwidth	1.2TB/s
Memory capacity	24/48GB

Technology

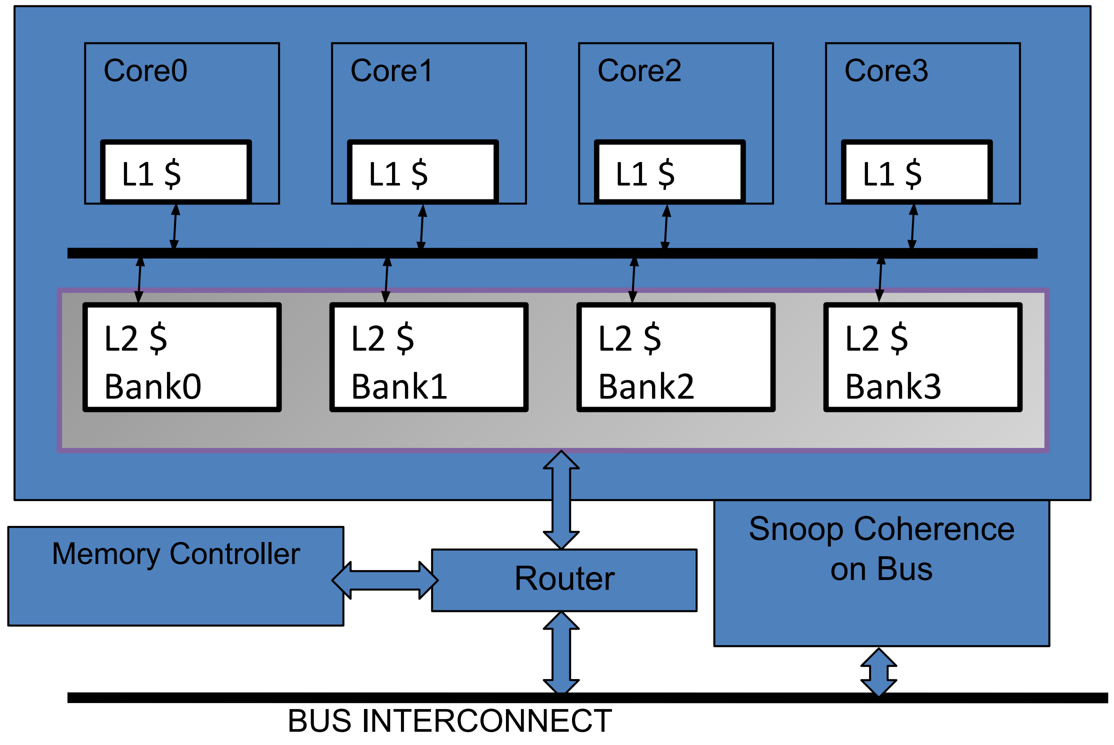
- 16nm FinFET process



12 © NEC Corporation 2018

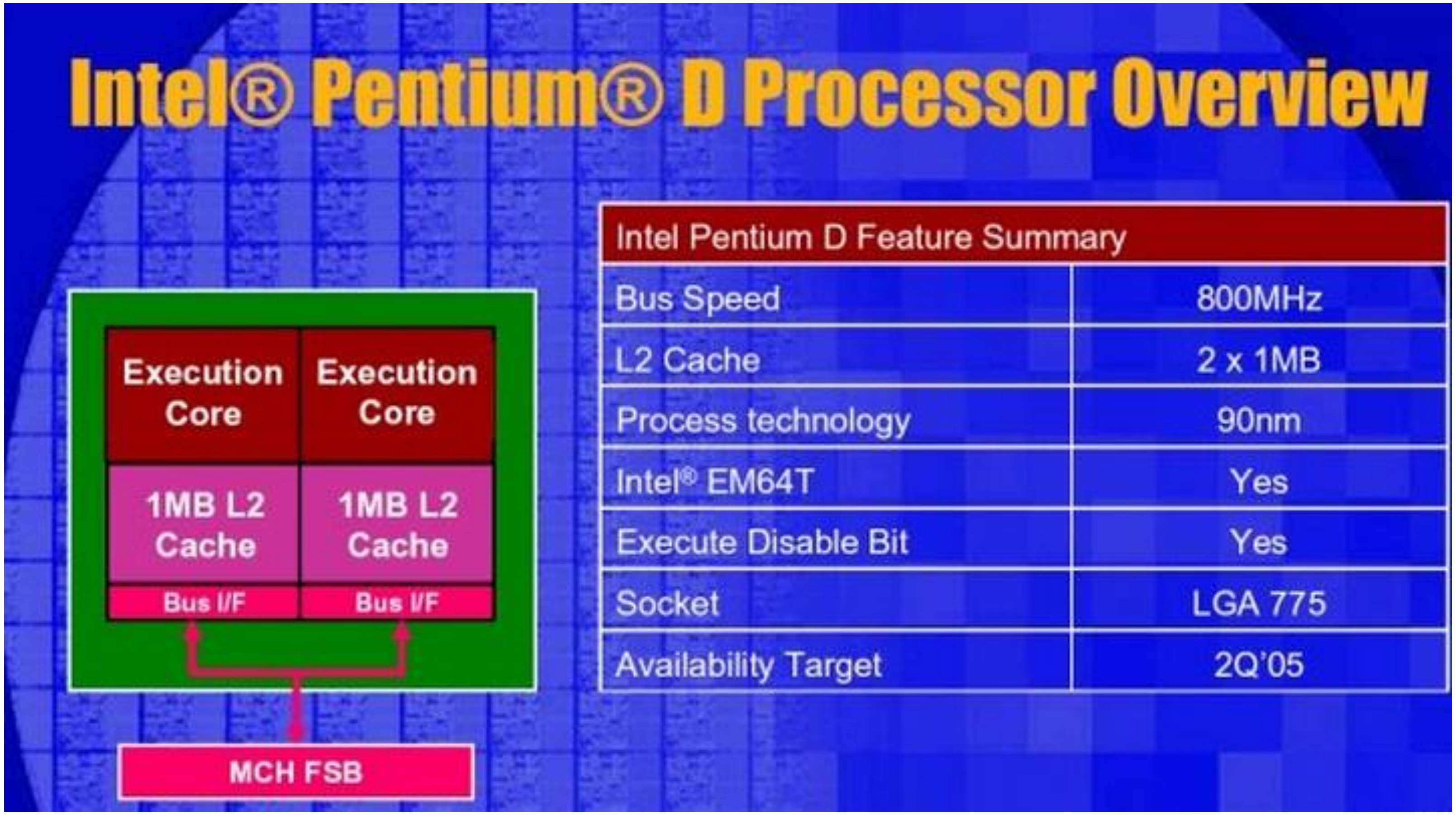
\Orchestrating a brighter world NEC

Bus-based CMPs



- **cores share L2 cache banks through a shared bus**
  - coherence is maintained between L1s by bus snooping
- **similar to SMP except that memory is replaced by L2**
  - inclusion is enforced between L2 and L1s
  - main difference is that a miss can happen in L2
  - in this case the on-chip protocol must be able to deal with variable latencies
- **Example:** early generation CMPs such as Pentium IV dual core processor

Intel Pentium D Dual Core (2005)

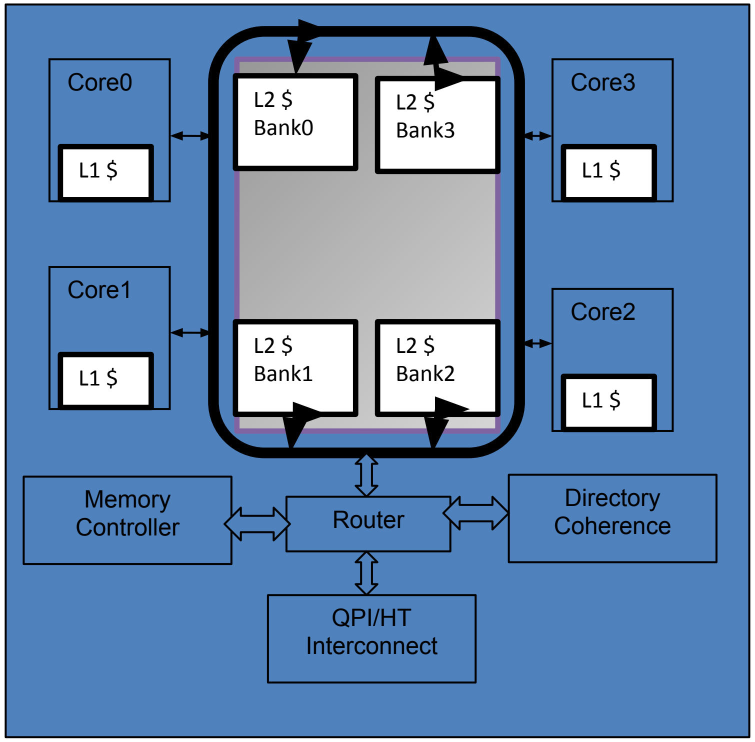


Intel® Pentium® D Processor Overview

Intel Pentium D Feature Summary	
Bus Speed	800MHz
L2 Cache	2 x 1MB
Process technology	90nm
Intel® EM64T	Yes
Execute Disable Bit	Yes
Socket	LGA 775
Availability Target	2Q'05

Ring-based CMPs

- **Nodes (core + L2 bank) are connected through a ring**
  - Multiple requests in progress on different links
    - Better bandwidth and scalability
  - Packets are routed by additional logic in each node (routers)
    - Increases complexity and introduces area overhead
  - Inter-core latency depends on number of hops
- **Rings can be clocked much faster than buses**
  - point-to-point links instead of global interconnect (e.g. bus)

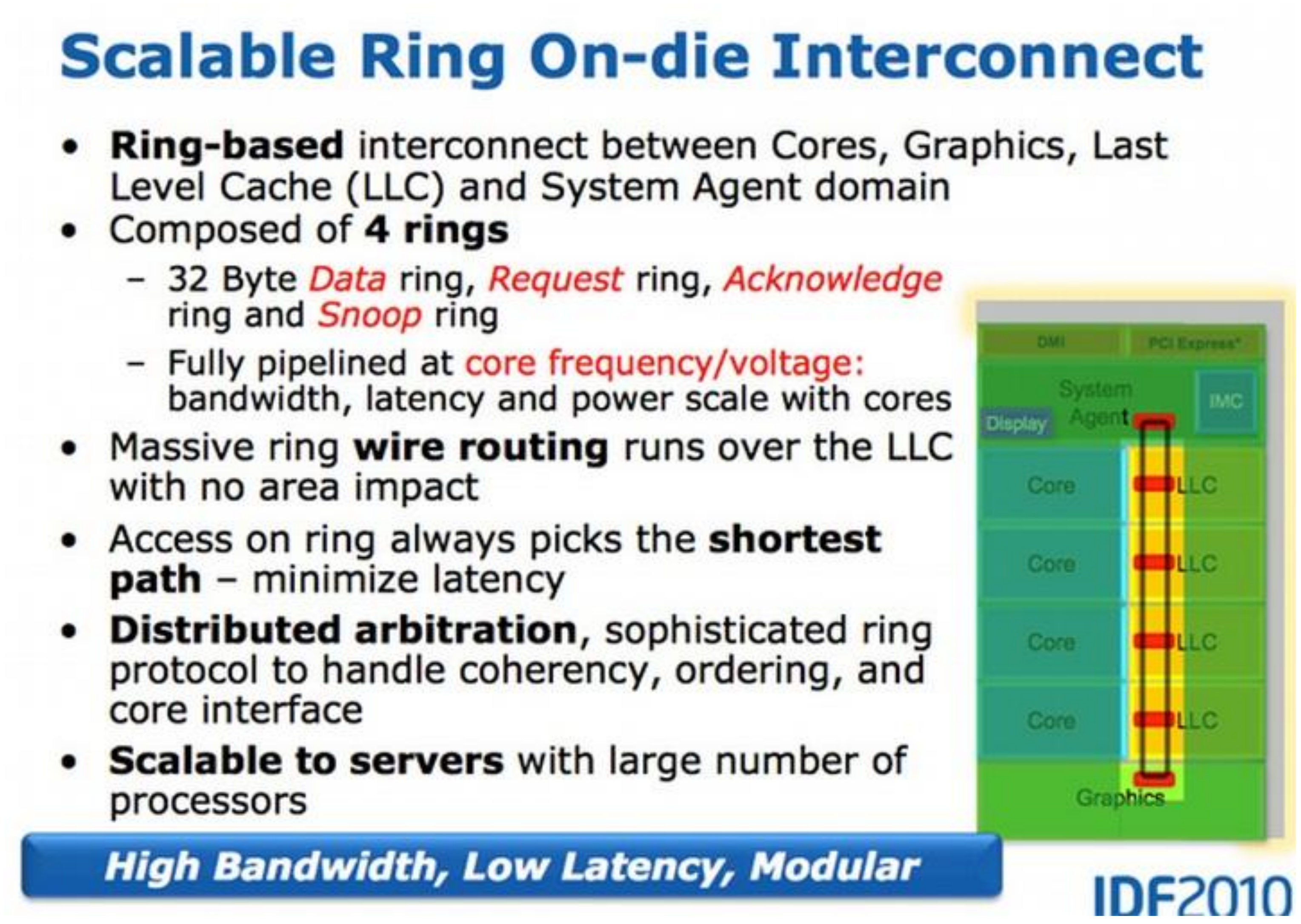


Cache coherence in Ring-based CMPs

- **Snooping protocols**
  - coherence requests visit (snoop) every node, including cores and L2 banks
  - requests “hop” around the ring, from link to link to broadcast to all nodes
  - responses (e.g., missing blocks, acks) are inserted in the ring
  - owner (L2 or dirty node) replies with the block on a miss
  - a coherence transaction takes one trip around the ring (constant time)

- **Directory-based protocols**
  - Each node (core plus L2 cache bank) is responsible for a range of addresses where the global state of the block is stored (presence bits, dirty bit)
  - Requests go first to home node
  - If home node is not the owner, then the request is forwarded to dirty node
  - If dirty node is between requester and home then one more round trip is needed (unless the ring network is bidirectional)

Example of Ring-based CMP: Intel Sandy Bridge (2011)

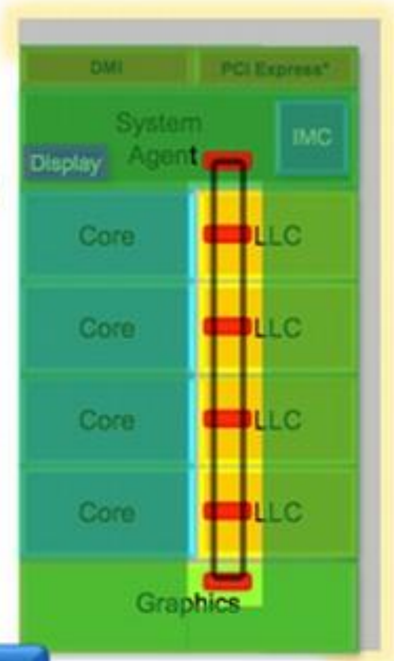


Scalable Ring On-die Interconnect

- **Ring-based** interconnect between Cores, Graphics, Last Level Cache (LLC) and System Agent domain
- Composed of **4 rings**
  - 32 Byte *Data* ring, *Request* ring, *Acknowledge* ring and *Snoop* ring
  - Fully pipelined at core frequency/voltage: bandwidth, latency and power scale with cores
- Massive ring **wire routing** runs over the LLC with no area impact
- Access on ring always picks the **shortest path** – minimize latency
- **Distributed arbitration**, sophisticated ring protocol to handle coherency, ordering, and core interface
- **Scalable to servers** with large number of processors

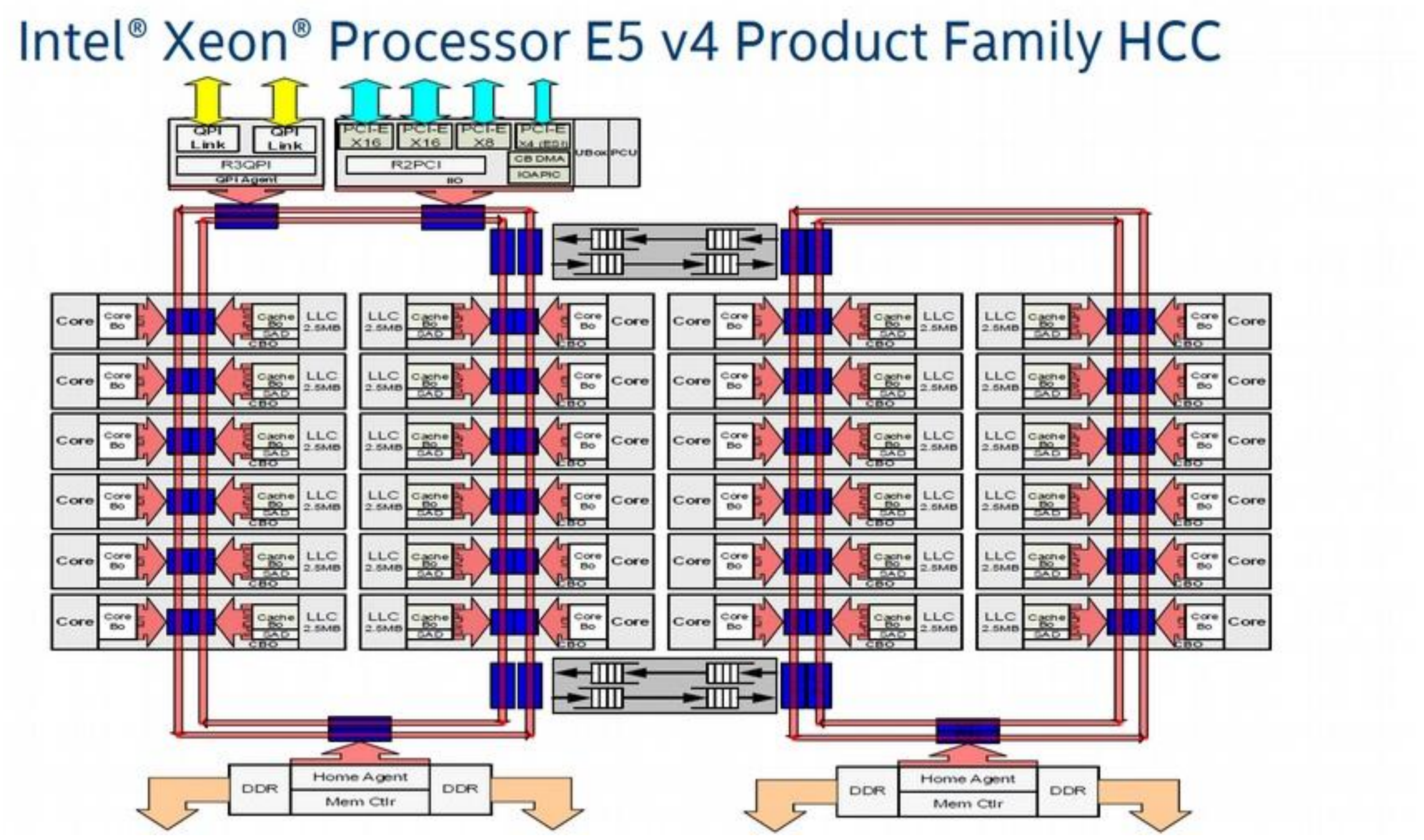
High Bandwidth, Low Latency, Modular

IDF2010



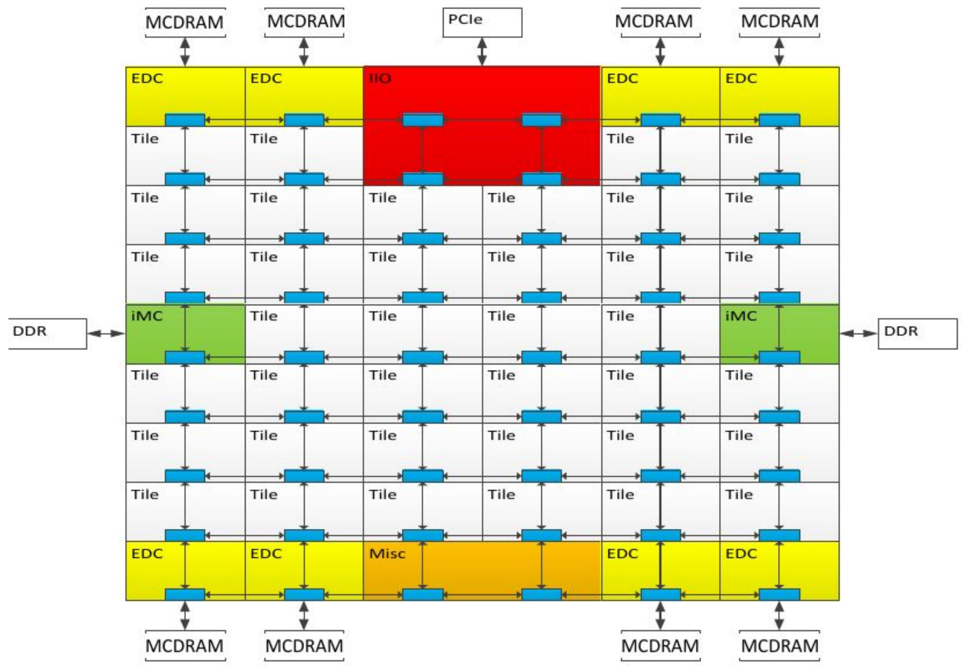
Scalability of rings

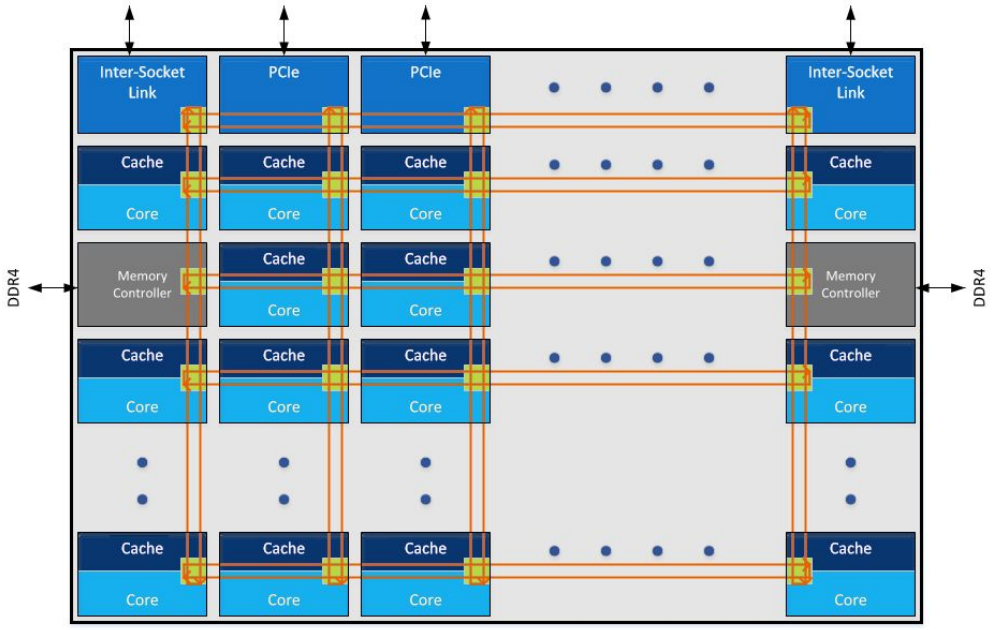
- Rings have two problems: 1) Latency grows linearly with number of cores, (2) Rings do not match common chip layouts
- **One option:** connect multiple rings via routers (like Xeon E5 v4)
    - Complex. Improves layout, but does not solve scalability issue (latency, BW)



Mesh Interconnects

- Mesh networks have higher bisection bandwidth
- Also a better match with chip-level layout





Intel Knights Landing (2016)

Intel Skylake-SP (2017)

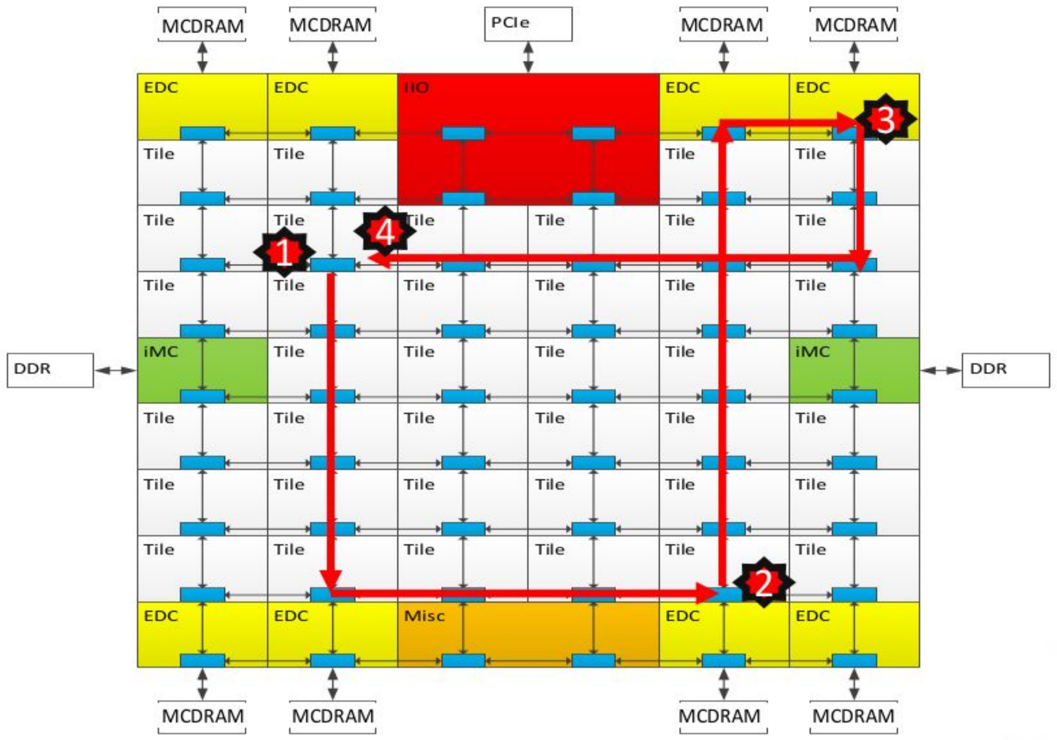
Interconnection network	Switch degree	Network diameter	Bisection width	Network size
Ring	2	N/2	2	N
n-by-n mesh	4	2(n-1)	n	$N=n^2$

Learn more about on-chip networks in the guest lecture next Tuesday! :)

Cache coherence with Mesh Interconnects

- **Snooping**: requires reaching farthest node. Variable latency, must assume worst case
- **Directory**: Similar solution to ring-based protocol, but packets proceed directly through network to destination node and back.
  - Example: Intel Knights Landing

Cluster Mode: All-to-All



Address uniformly hashed across all distributed directories

No affinity between Tile, Directory and Memory

Most general mode. Lower performance than other modes.

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

14

SUMMARY (Lecture 8)

- Heterogeneous CMPs
- CMP Memory Architectures
- CMP Interconnection networks