MultiCore Processors

Dr. Smruti Ranjan Sarangi
IIT Delhi

Part I - Multicore Architecture

Part II - Multicore Design

Part III - Multicore Examples

Part I - Multicore Architecture

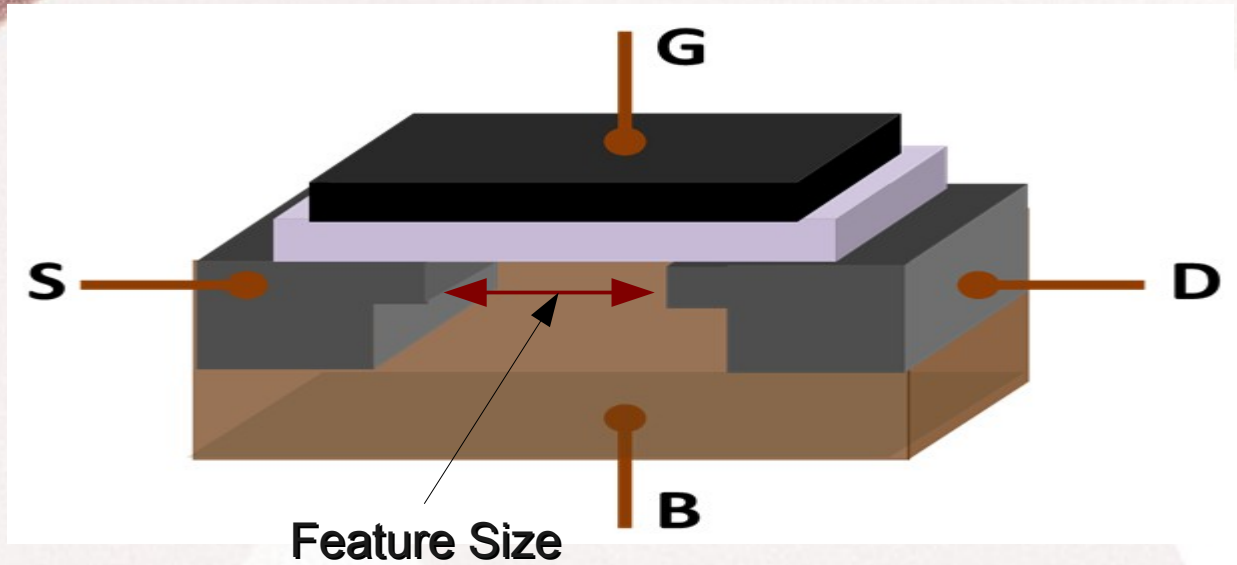
Outline

- Moore's Law and Transistor Scaling
- Overview of Multicore Processors
- Cache Coherence
- Memory Consistency

Moore's Law

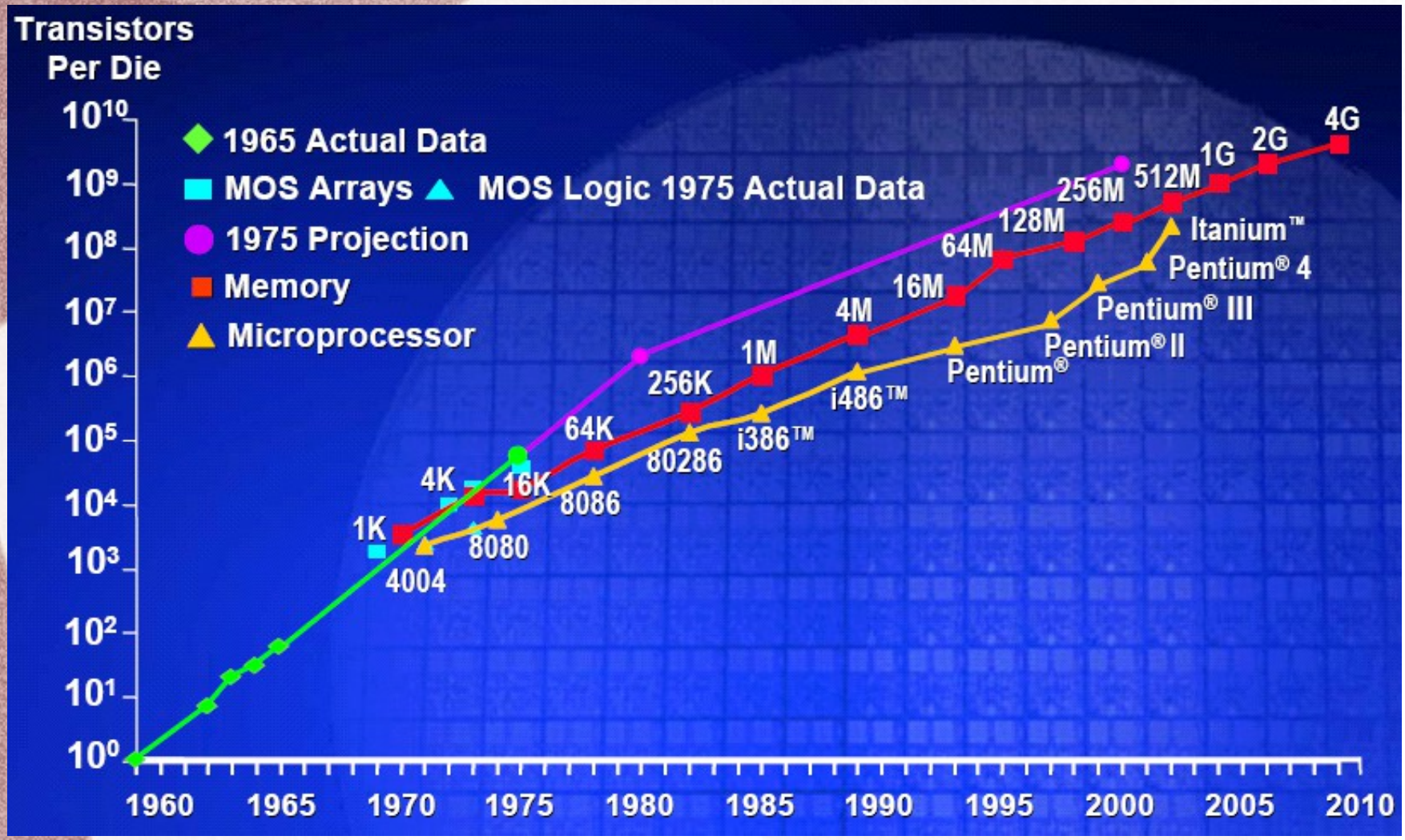
- Intel's co-founder Gordon Moore predicted in 1965 that the number of transistors on chip will double every year
  - Reality Today : Doubles once every 2 years
- How many transistors do we have today per chip?
  - Approx 1 billion
- 2014 – 2 billion
- 2016 – 4 billion

Why do Transistors Double?



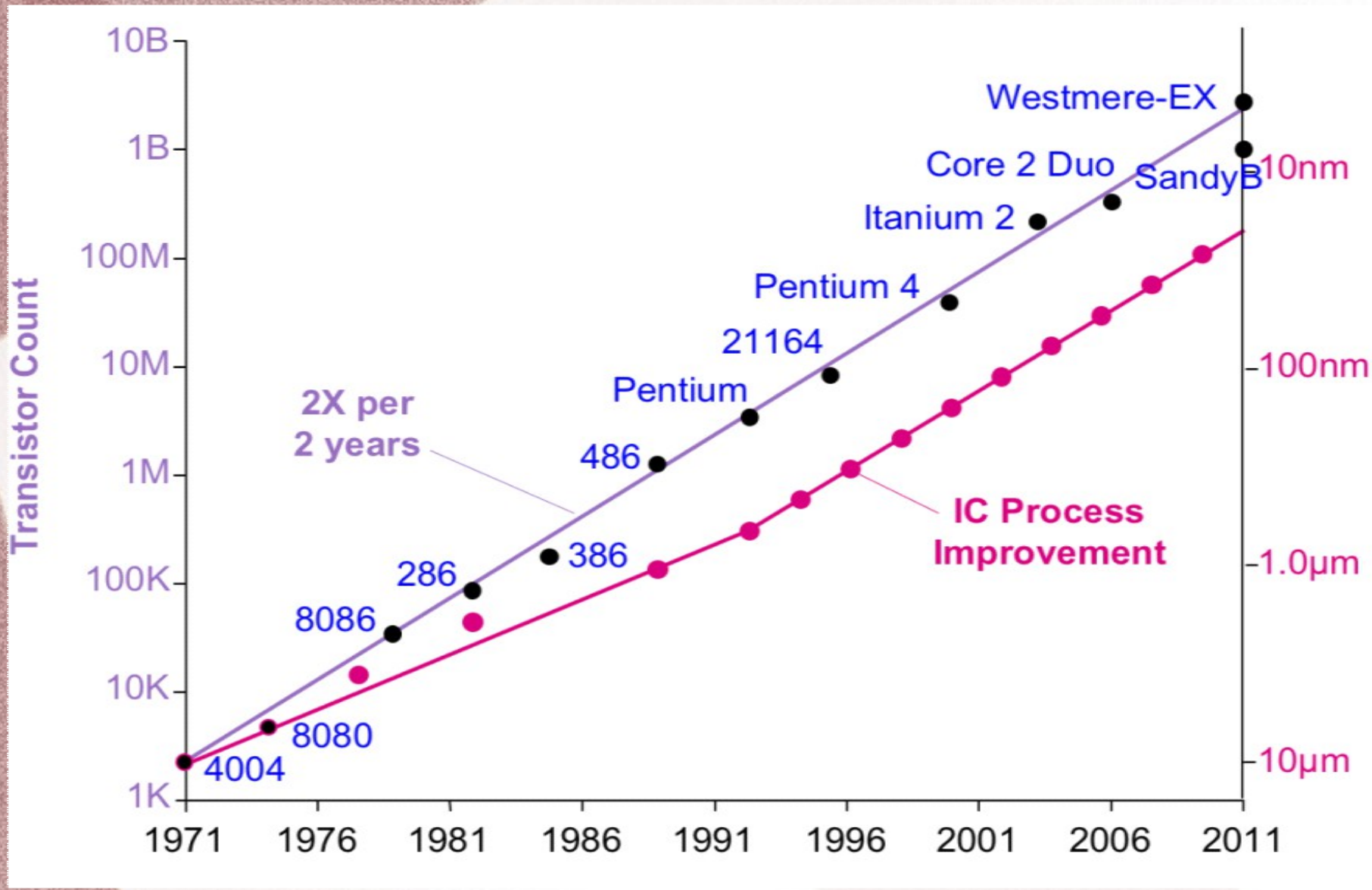
source: wikipedia

- The feature size keeps shrinking by $\sqrt{2}$ every 2 years
- Currently it is 32 nm
- 32 nm → 22 nm → 18 nm →



Number of transistors per chip over time

Source: Yaseen et. al.



Source: Yaseen et. al.

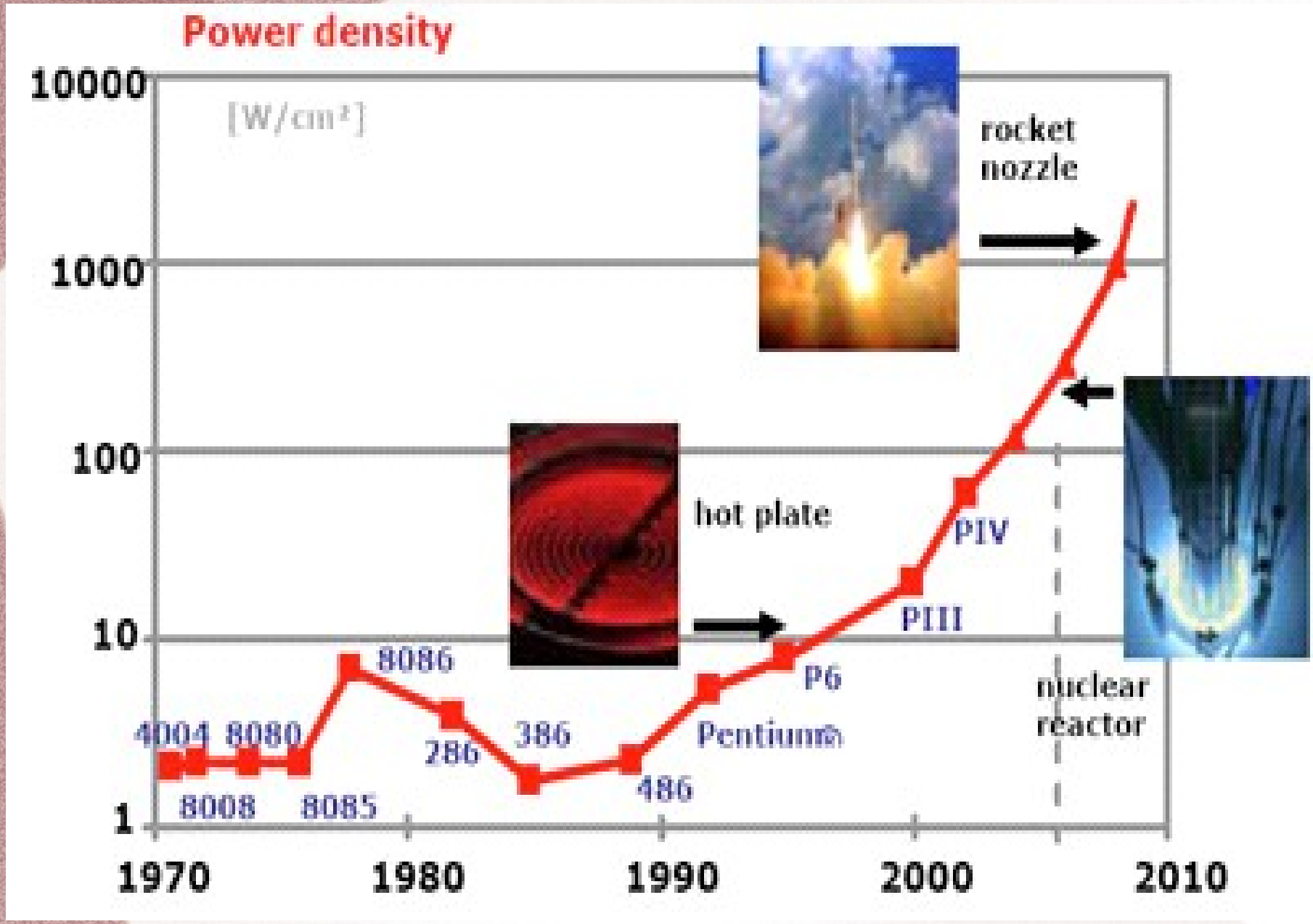
How to Use the Extra Transistors

- Traditional
  - Complicate the processor
    - Issue width, ROB size, aggressive speculation
  - Increase the cache sizes
    - L1, L2, L3
- Modern (post 2005)
  - Increase the number of cores per chip
  - Increase the number of threads per core

What was Wrong with the Traditional Approach?

- Law of diminishing returns
  - Performance does not scale well with the increase in cpu resources
  - Delay is proportional to size of a cpu structure
  - Sometimes it is proportional to the square of the size
  - Hence, there was no significant difference between a 4-issue and a 8-issue processor
  - Extra caches had also limited benefit because the working set of a program completely fits in the cache
- Wire Delay
  - It takes 10s of cycles for a signal to propagate from one end of a chip to the other end. Wire delays decrease the advantage of pipelining.

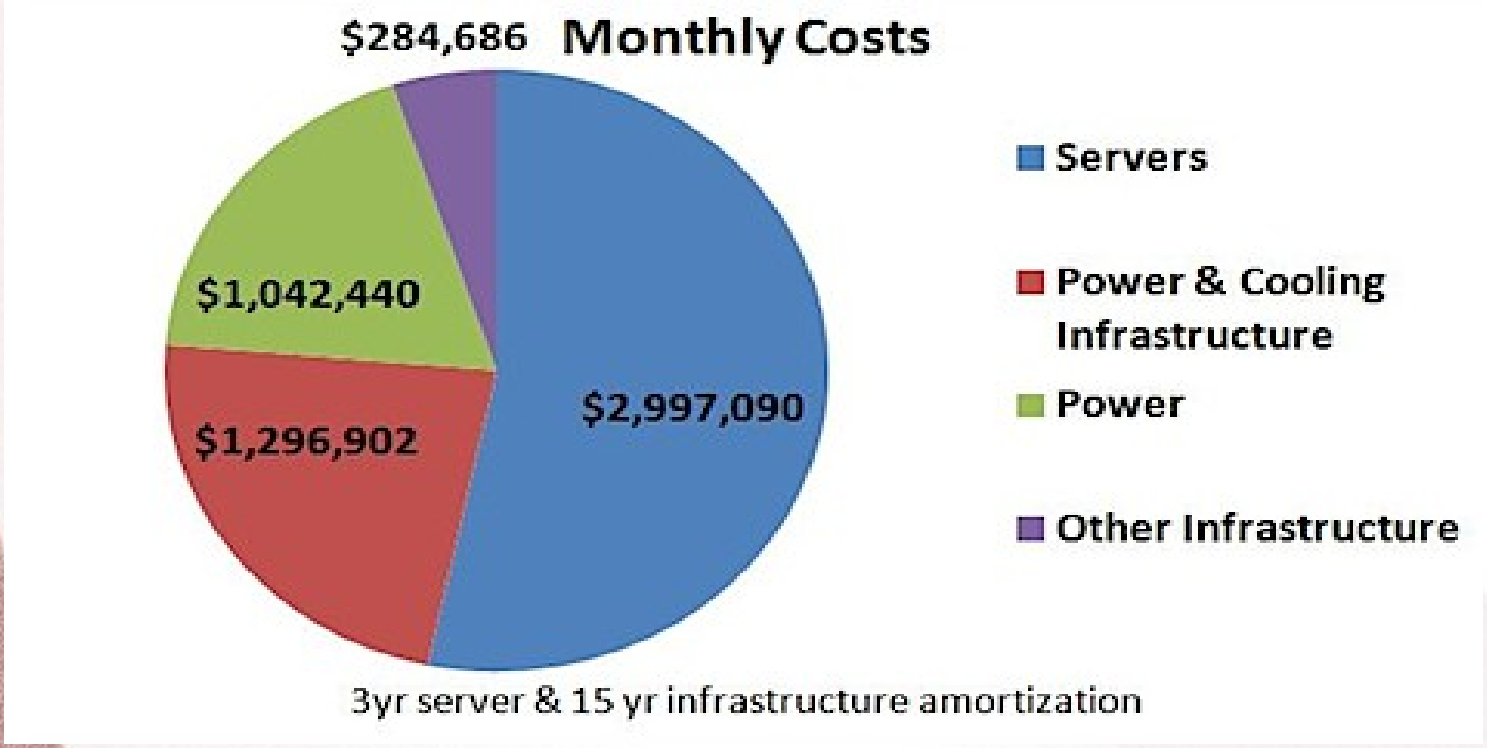
Power & Temperature



Source: Intel Microprocessor Report

Power &Temperature - II

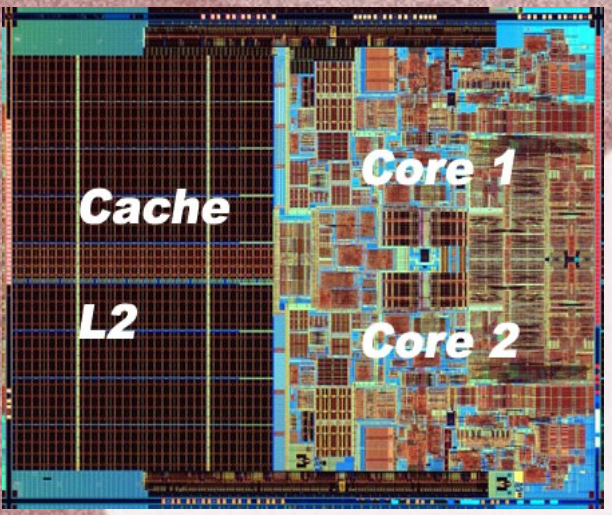
- High power consumption
  - Increases cooling costs
  - Increases the power bill

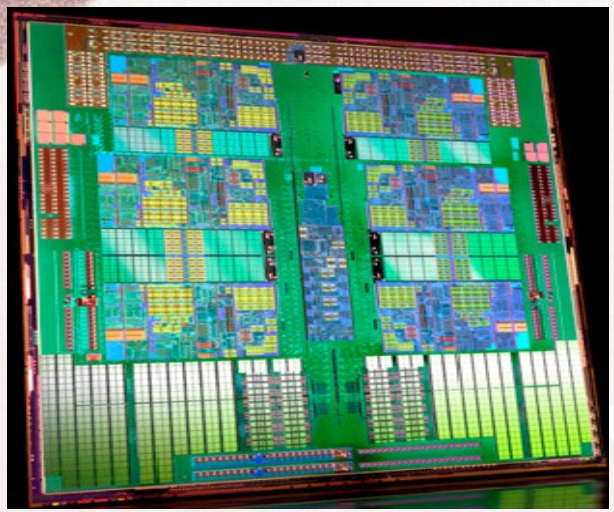


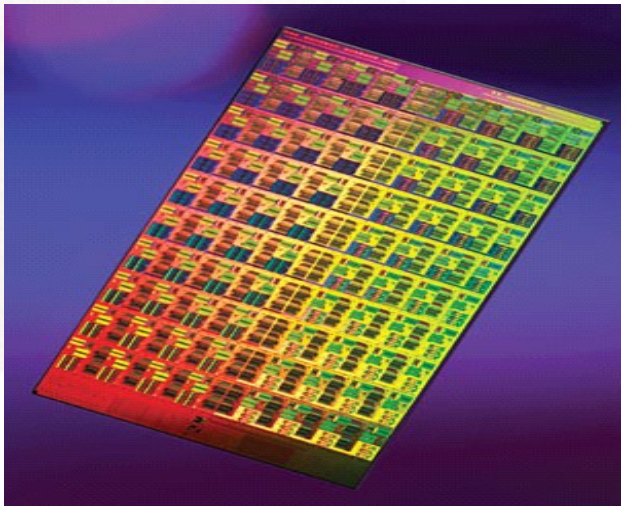
Source: O'Reilly Radar

Intel's Solution: Have Many Simple Cores

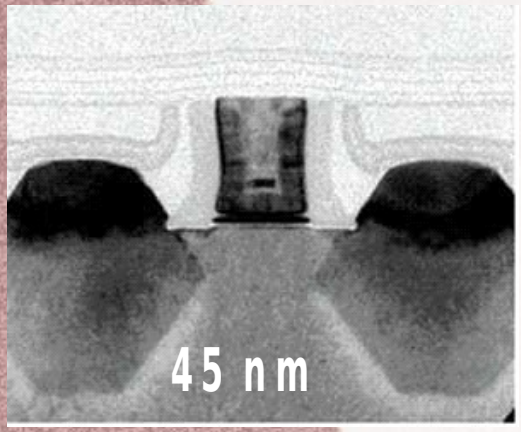
- Some major enhancements to processors
  - Pentium – 2 issue inorder pipeline (5 stage)
  - Pentium Pro – Out of order pipeline (7 stage)
  - Pentium 4 – Aggressive out-of-order pipeline (18 stage) + trace cache
  - Pentium Northwood – 27 stage out-of-order pipeline
  - Pentium M – 12 stage out of order pipeline
  - Intel Core2 Duo – 2 Pentium M cores
  - Intel QuadCore – 4 Pentium M cores

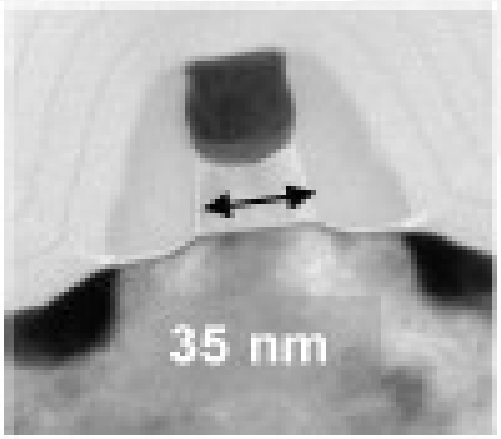


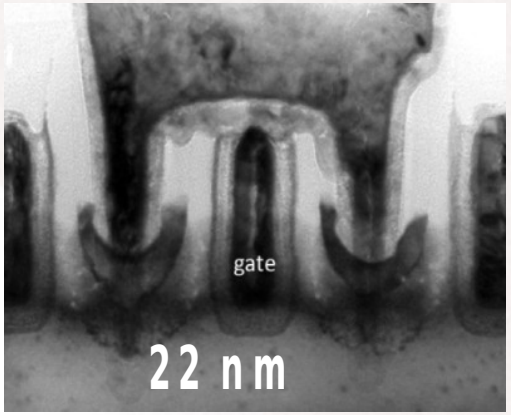


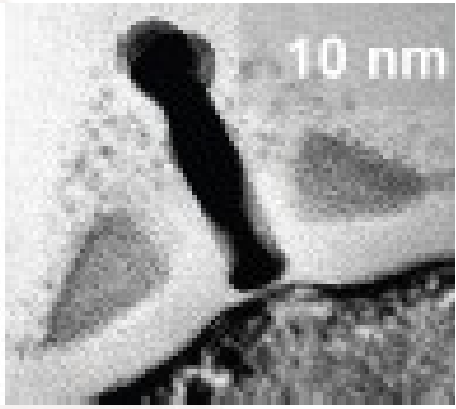


Evolution







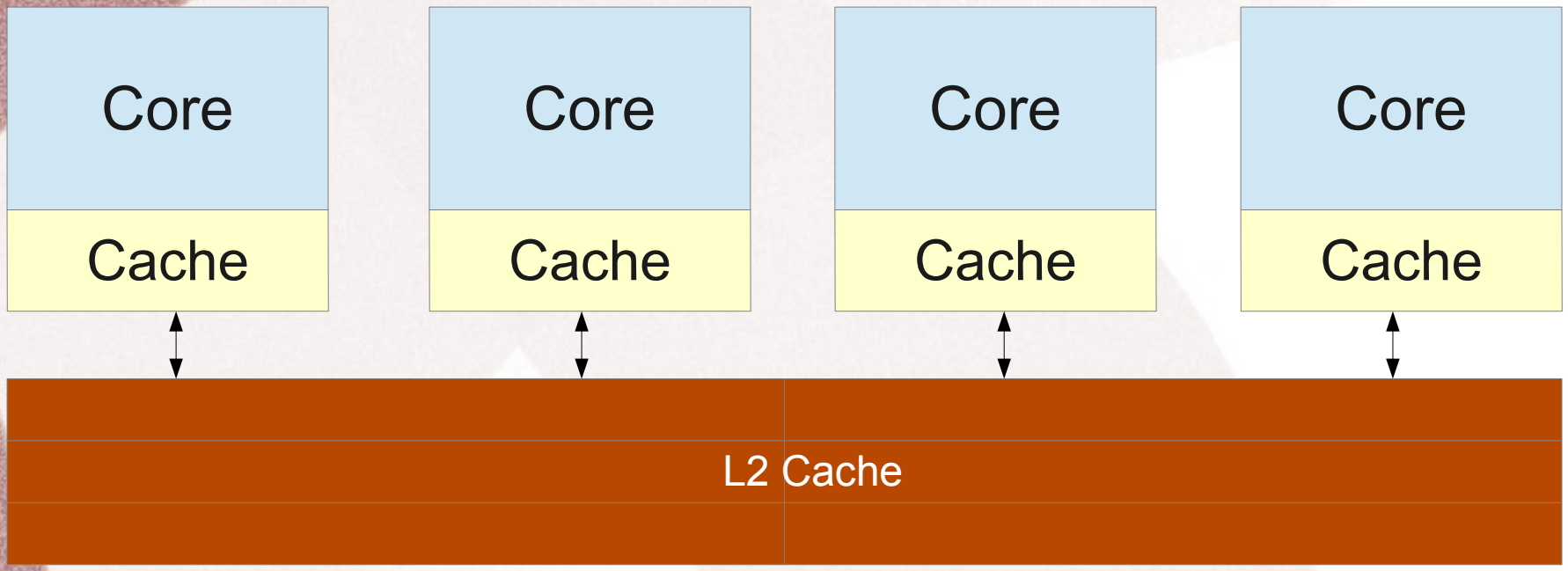


Source: chipworks.com

Outline

- Moore's Law and Transistor Scaling
- Overview of Multicore Processors
- Cache Coherence
- Memory Consistency

What is a Multicore Processor?



- Multiple processing cores, with private caches
- Large shared L2 or L3 caches
- Complex interconnection network
- This is also called symmetric multicore processor

Advantages of Multicores

- The power consumption per core is **limited**. It decreases by about 30% per generation.
- The design has lesser complexity
  - easy to design, debug and verify
- The performance per core increases by about 30% per generation
- <u>The number of cores double every generation.</u>

Multicores

Power

Complexity

Performance per Core

Parallelism

Issues in Multicores

- We have so many cores ...

? How to use them?

ANSWER

1) We need to write effective and efficient parallel programs
2) The hardware has to facilitate this endeavor

Parallel processing is the biggest advantage

Parallel Programs

Sequential

```
for (i=0;i<n;i++)
    c[i] = a[i] * b[i]
```

Parallel

```
parallel_for(i=0;i<n;i++)
    c[i] = a[i] * b[i]
```

What if ???

```
parallel_for(i=0;i<n;i++)
    c[d[i]] = a[i] * b[i]
```

Is the loop still parallel?

1st Challenge: Finding Parallelism

- To run parallel programs efficiently on multi-cores we need to:
  - discover parallelism in programs
  - re-organize code to expose more parallelism
- Discovering parallelism : automatic approaches
  - Automatic compiler analysis
  - Profiling
  - Hardware based methods

Finding Parallelism- II

- Discovering parallelism: manually
  - Write explicitly parallel algorithms
  - Restructure loops to make them parallel

Example

Sequential

```
for (i=0;i<n-1;i++)
    a[i] = a[i] * a[i+1]
```

Parallel

```
a_copy = copy(a)
parallel_for(i=0;i<n-1;i++)
    a[i] = a_copy[i] *
           a_copy[i+1]
```

Tradeoffs

- Compiler Approach
  - Not very extensive
  - Slow
  - Limited utility
- Manual approach
  - Very difficult
  - Much better results
  - Broad utility

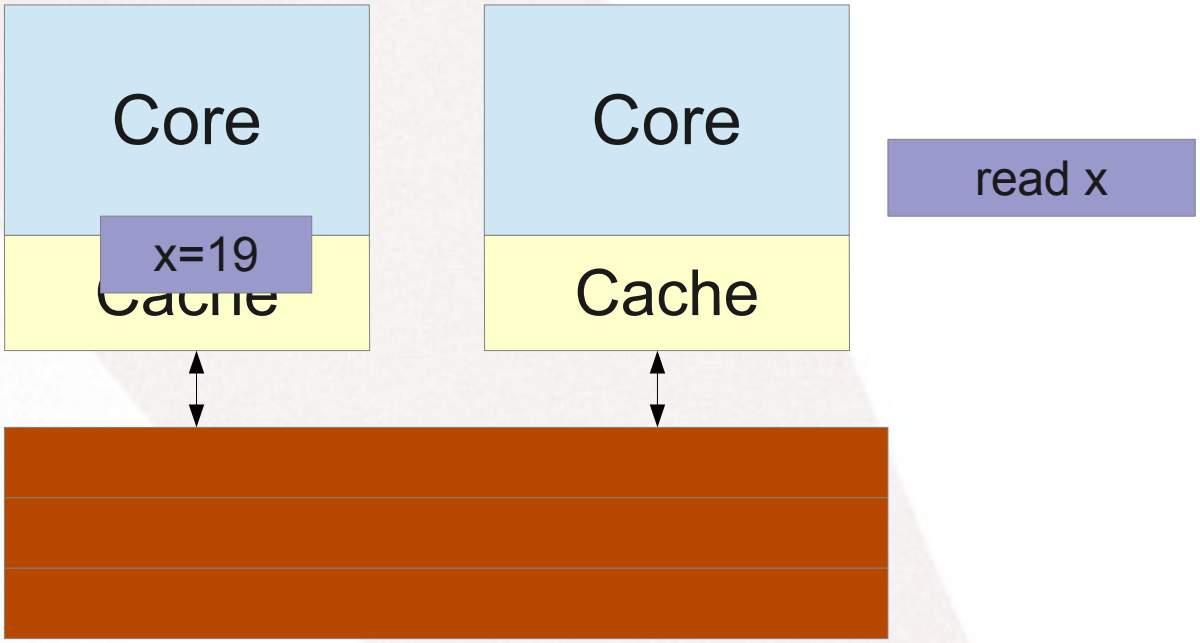
Given good software level approaches, how can hardware help ?

Models of Parallel Programs

- Shared memory
  - All the threads see the same view of memory (same address space)
  - Hard to implement but easy to program
  - Default (used by almost all multicores)
- Message passing
  - Each thread has its private address space
  - Simpler to program
  - Research prototypes

Shared Memory Multicores

- What is shared memory?



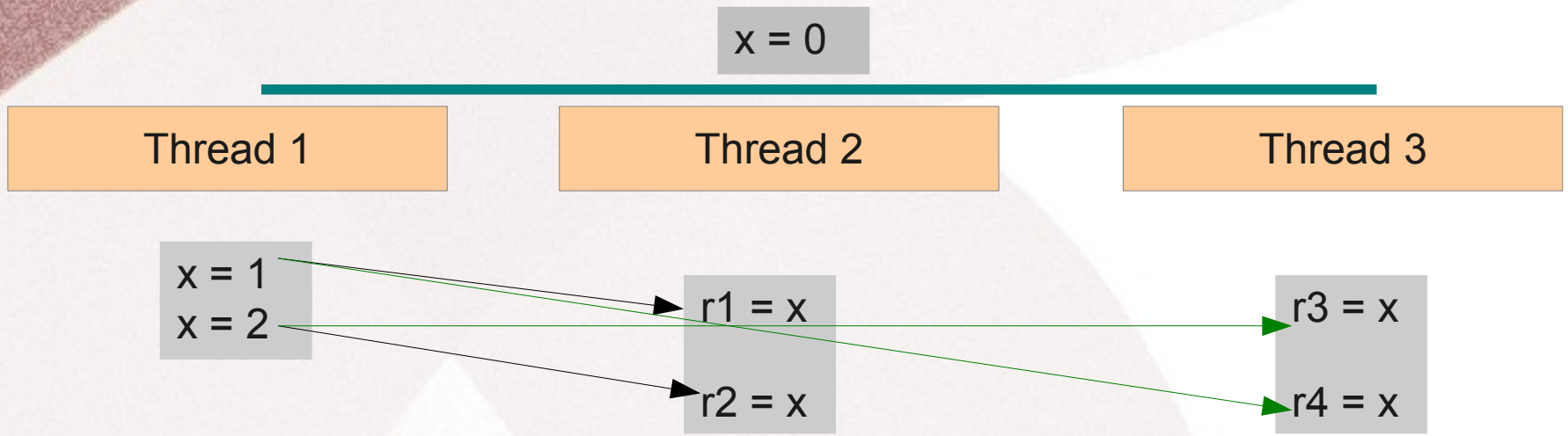
How does it help?

- Programmers need not bother about low level details.
- The model is immune to process/thread migration
- The same data can be accessed/ modified from any core
- Programs can be ported across architectures very easily, often without any modification

Outline

- Moore's Law and Transistor Scaling
- Overview of Multicore Processors
- Cache Coherence
- Memory Consistency

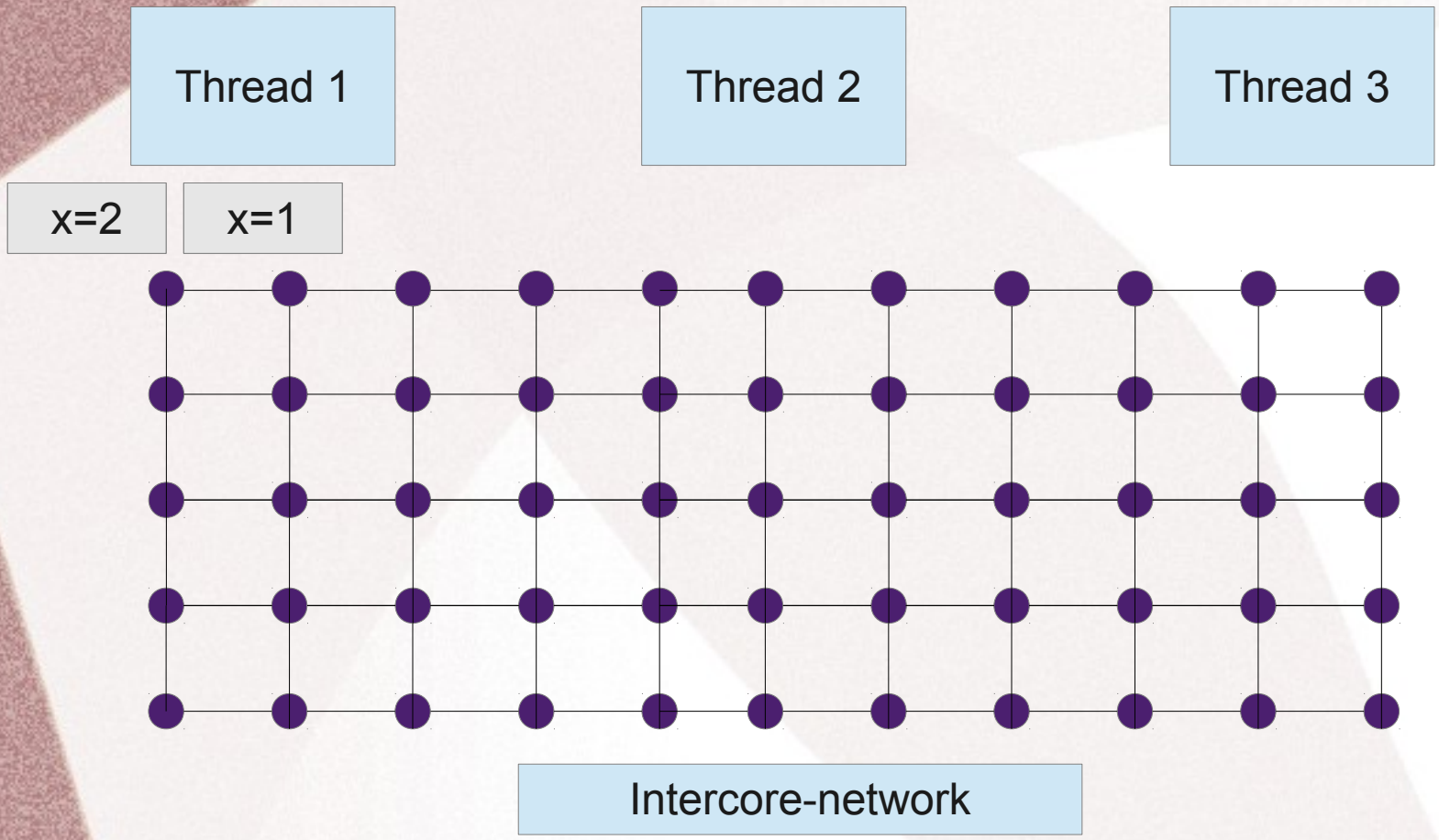
What are the Pitfalls? Example 1



Is the outcome (r1=1, r2=2) (r3=2, r4=1) feasible?

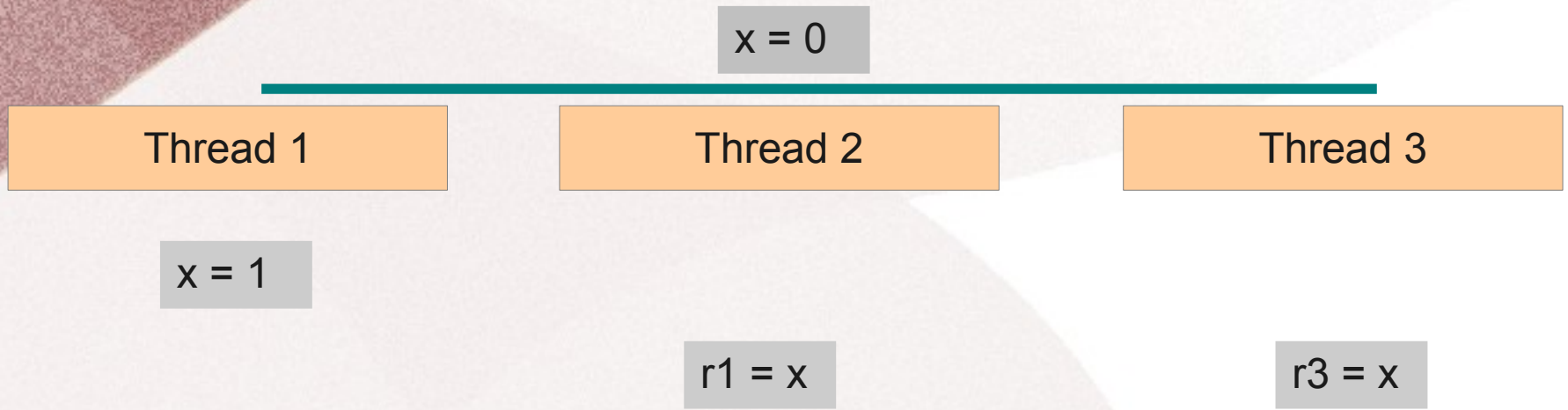
Does it make sense?

Example 1 contd...



Order gets reversed

Example 2



Is the outcome (r1 = 0, r3 = 0) feasible?

Does it make sense?

When should a write from one processor be visible to other processors?

Point to Note

- Memory accesses can be reordered by the memory system.
- The memory system is like a real world network.
- It suffers from bottlenecks, delays, hot spots, and sometimes can drop a packet

How should Applications behave?

- It should be possible for programmers to write programs that make sense
- Programs should behave the same way across different architectures

Cache Coherence

Axiom 1	A write is ultimately visible.
Axiom 2	Writes to the same location are seen in the same order

Example Protocol

Claim

- The following set of conditions satisfy Axiom 1 and 2

Axiom 1

A write is ultimately visible.

Condition 1

Every write completes in a finite amount of time

Axiom 2

Condition 2

Writes to the same location are seen in the same order

At any point of time: a given cache block is either being read by multiple readers, or being written by just one writer

Practical Implementations

- Snoopy Coherence
  - Maintain some state per every cache block
  - Elaborate protocol for state transition
  - Suitable for CMPs with cores less than 16
  - Requires a shared bus
- Directory Coherence
  - Elaborate state transition logic
  - Distributed protocol relying on messages
  - Suitable for CMPs with more than 16 cores

Implications of Cache Coherence

- It is not possible to drop packets. All the threads/cores should perceive 100% reliability
- Memory system cannot reorder requests or responses to the same address
- How to enforce cache coherence
  - Use Snoopy or Directory protocols
  - Reference: Henessey Patterson Part 2

Outline

- Moore's Law and Transistor Scaling
- Overview of Multicore Processors
- Cache Coherence
- Memory Consistency

Reordering Memory Requests in General

x = 0 , y = 0

Thread 1	Thread 2
x = 1 r1 = y	y = 1 r2 = x

Is the outcome (r1=0, r2=0) feasible?

Does it make sense?

- Answer : Depends ….

Reordering Memory Requests

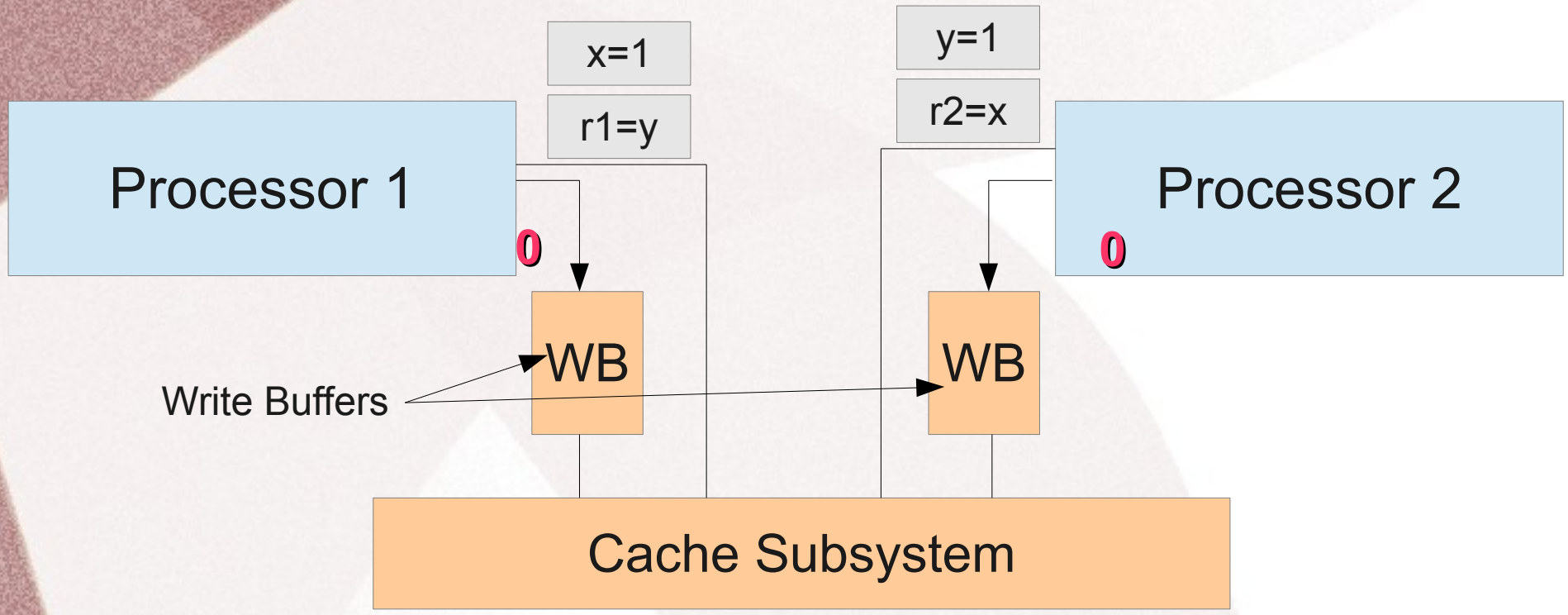
Sources

1 Network on Chip

2 Write Buffer

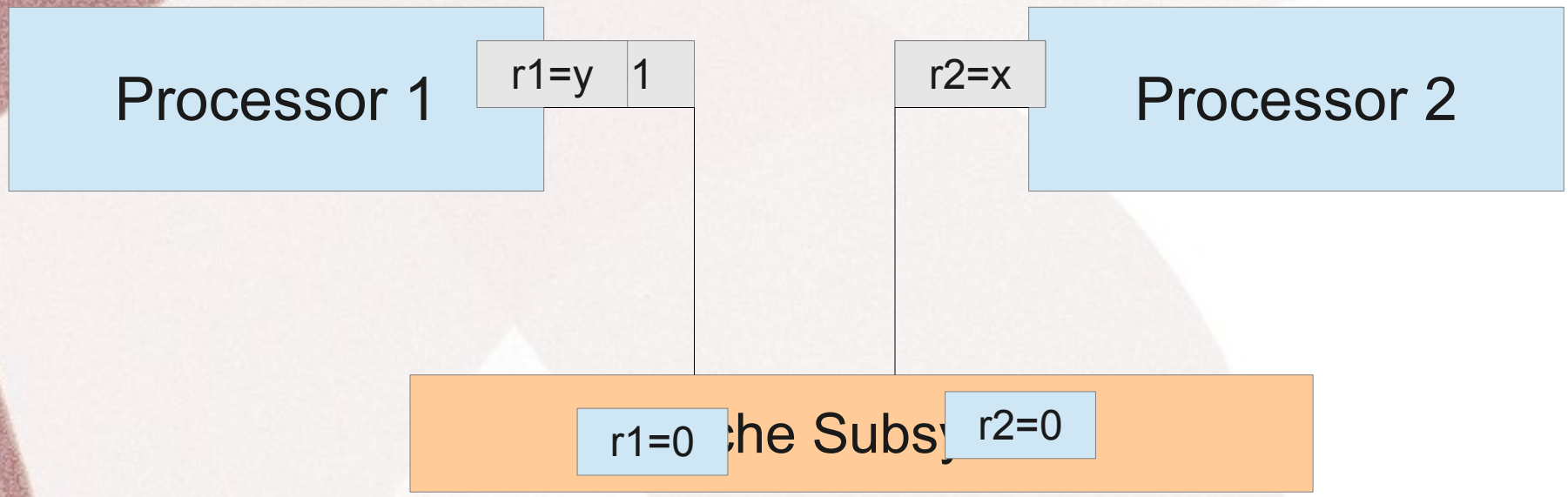
3 Out-of-order Pipeline

Write Buffer



- Write buffers can break W → R ordering

Out of Order Pipeline



- A typical out-of-order pipeline can cause abnormal thread behavior

What is allowed and What is Not?

Same Address	Different Address
Cannot reorder memory Requests	Reordering may be possible
Cache Coherence	Memory Consistency

Memory Consistency Models

	W → R	W → W	R → W	R → R
Sequential Consistency (SC) E.g, MIPS R1000	✓	✓	✓	✓
Total Store Order (TSO) E.g., Intel procs, Sun Sparc V9		✓	✓	✓
Partial Store Order (PSO) E.g., Sparc V8			✓	✓
Weak Consistency IBM Power				
Relaxed Consistency E.g., Research prototypes				

Sequential Consistency is Intuitive

- Definition of sequential consistency
  - If we run a parallel program on a sequentially consistent machine, then the output is equal to that produced by some sequential execution.

Example

x = 0 , y = 0

Thread 1	Thread 2
x = 1 r1 = y	y = 1 r2 = x

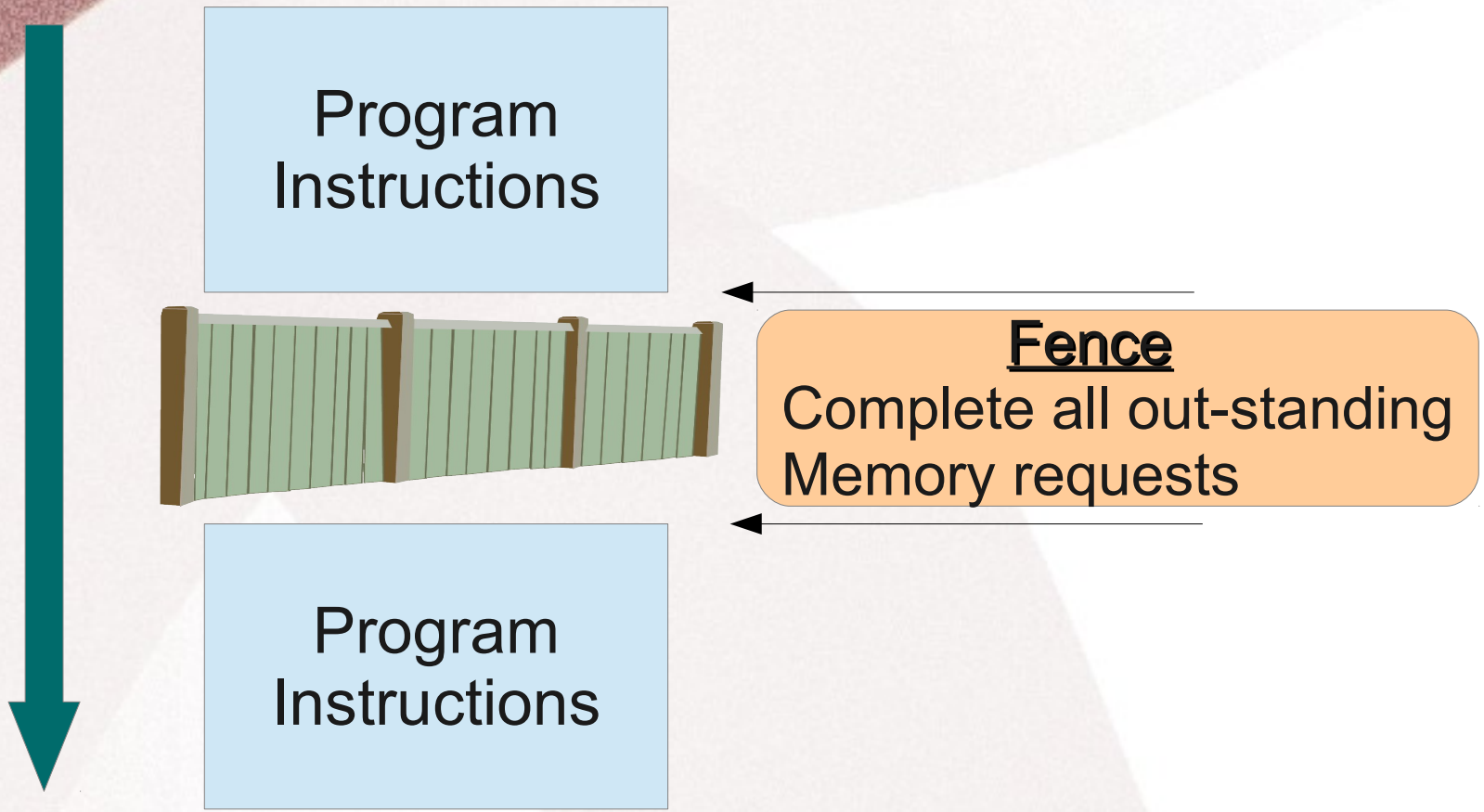
Is the outcome (r1=0, r2=0) feasible?

- Answer
  - Sequential Consistency (NO)
  - All other models (YES)

Comparison

- Sequential consistency is intuitive
  - Very low performance
  - Hard to implement and verify
  - Easy to write programs
- Relaxed memory models
  - Good performance
  - Easier to verify
  - Difficult to write correct programs

Solution


Program Instructions
Fence
Complete all out-standing Memory requests
Program Instructions

Make our Example Run Correctly

x = 0 , y = 0

Thread 1	Thread 2
x = 1	y = 1
Fence	**Fence**
r1 = y	r2 = x

Gives the correct result irrespective of the memory consistency model

Basic Theorem

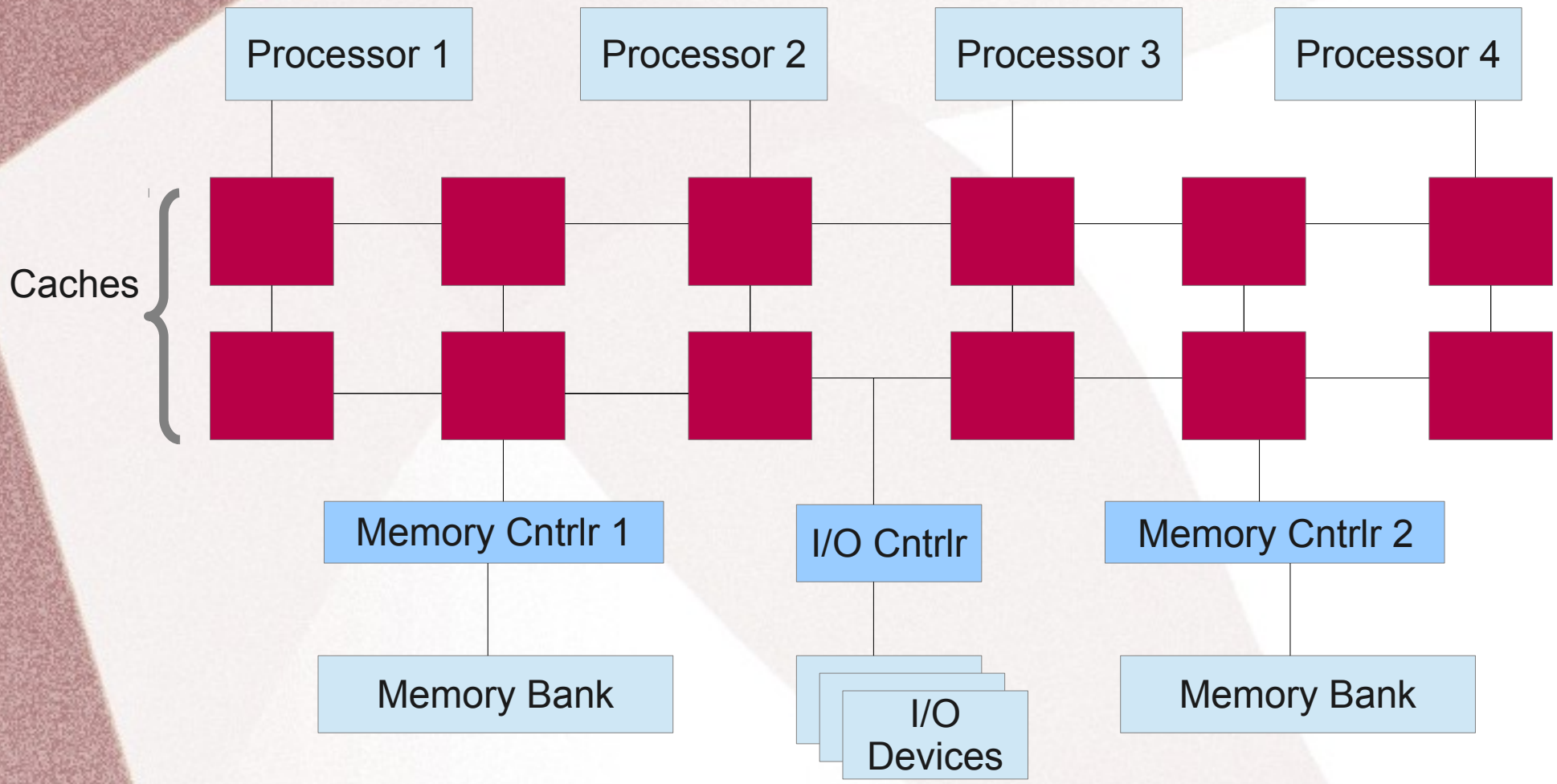
- It is possible to guarantee sequentially consistent execution of a program by inserting fences at the correct places.
- Implications:
  - Programmers need to be multi-core aware and write programs properly
  - Smart compiler infrastructure

Shared Memory: A Perspective

- Implementing a memory system for multicore processors is a non-trivial task
  - Cache coherence (fast, scalable)
  - Memory consistency
    - Library and compiler support
    - Tradeoff: Performance vs simplicity

Part II - Multiprocessor Design

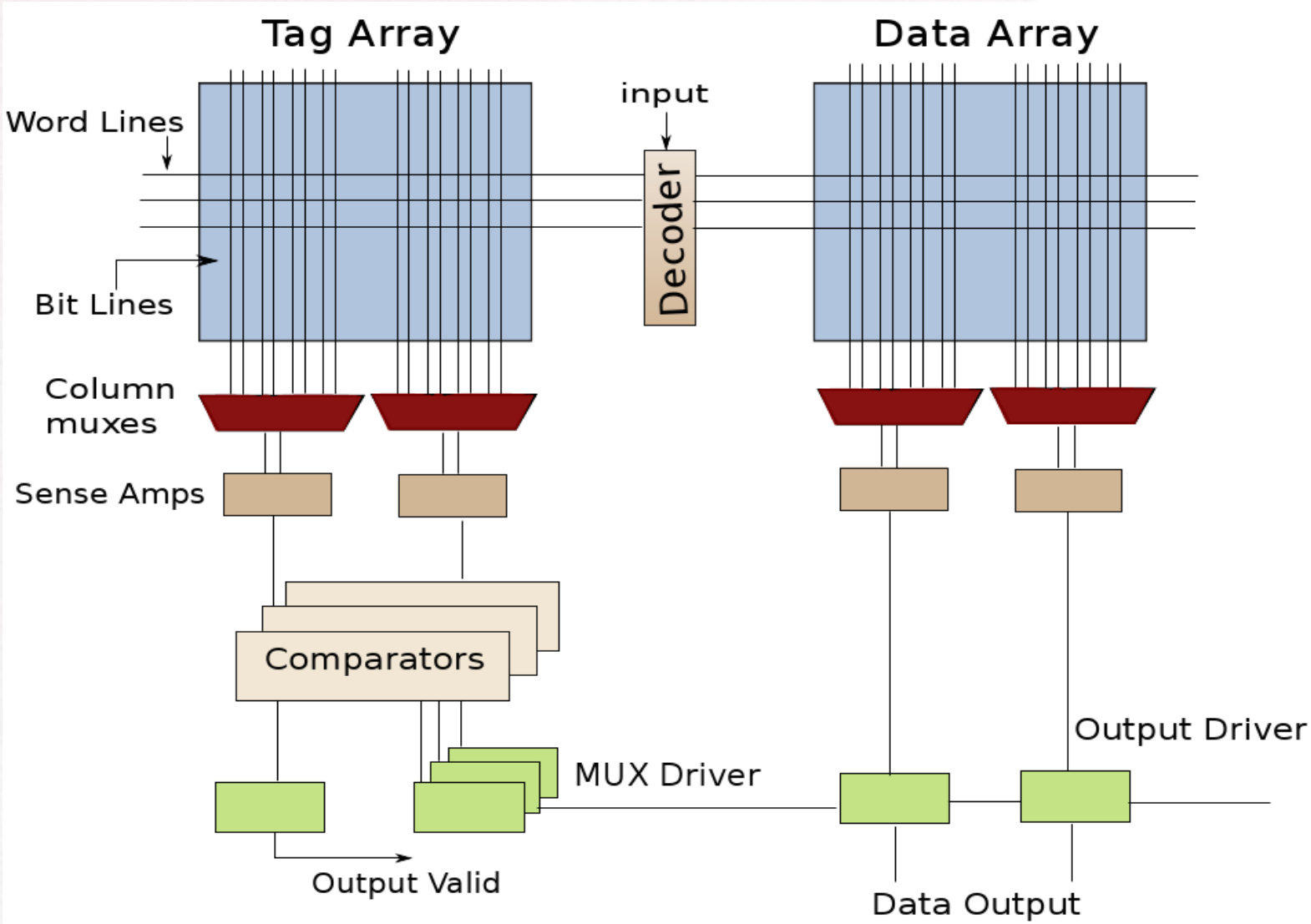
Multicore Organization



Architecture vs Organization

- Architecture
  - Shared memory
  - Cache coherence
  - Memory consistency
- Organization
  - Caches
  - Network on chip (NOC)
  - Memory and I/O controllers

Basics of Multicore Caches - Cache Bank

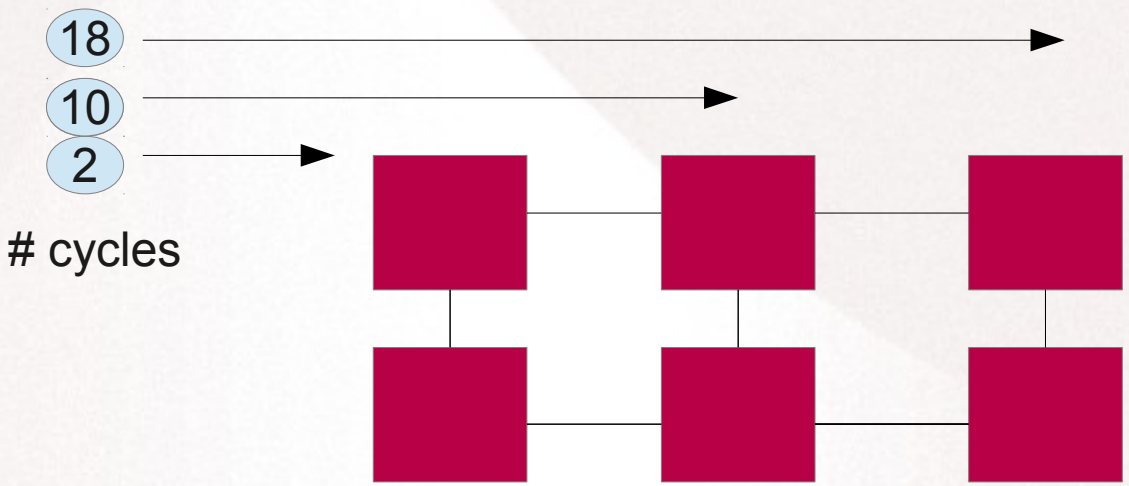


Large Caches

- Multicores typically have large caches (2- 8 MB)
- Several cores need to simultaneously access the cache
- We need a cache that is fast and power efficient
- **DUMB SOLUTION :** Have one large cache
- Why is the solution dumb?
  - Violates the basic rules of cache design

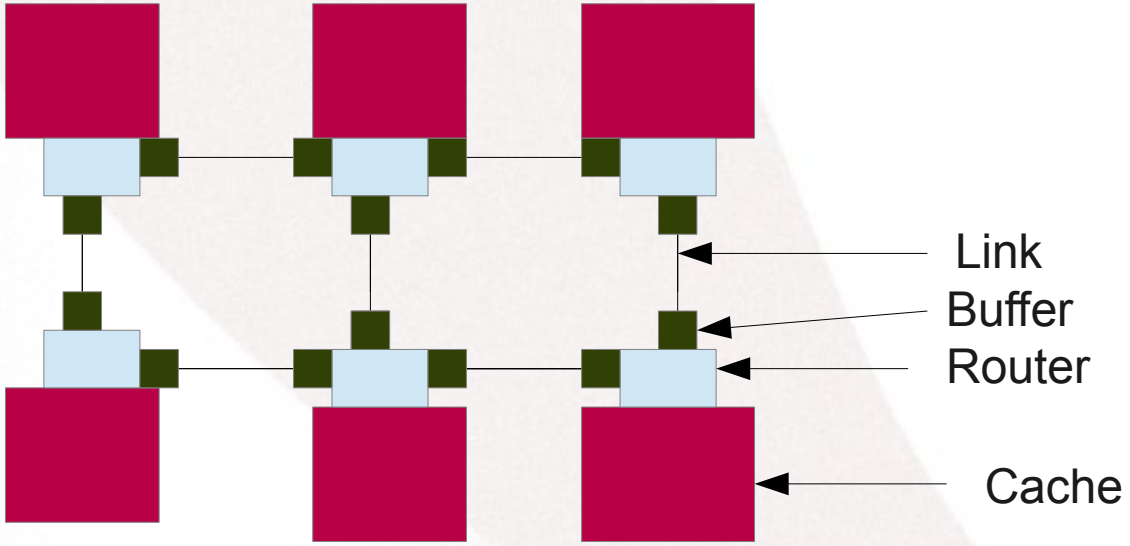
ABCs of Caches

- Delay is proportional to size
- Power is proportional to size
  - Can be proportional to size^2 for very large caches
- Delay is proportional to (#ports)^2
- Wire delay is a major factor



Smart Solution

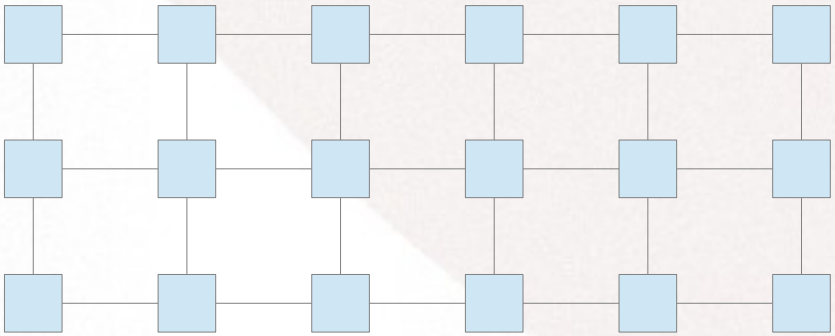
- Create a network of small caches
- Each cache is indexed by unique bits of the address
- Connect the caches using an on-chip network



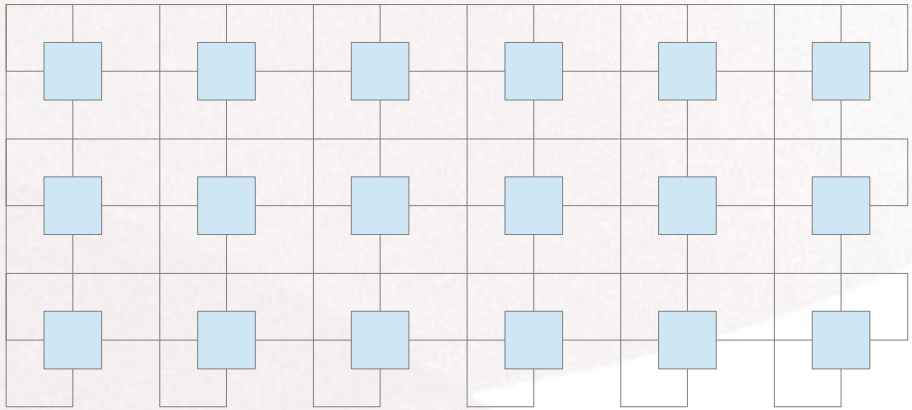
Network Topology

(a) chain

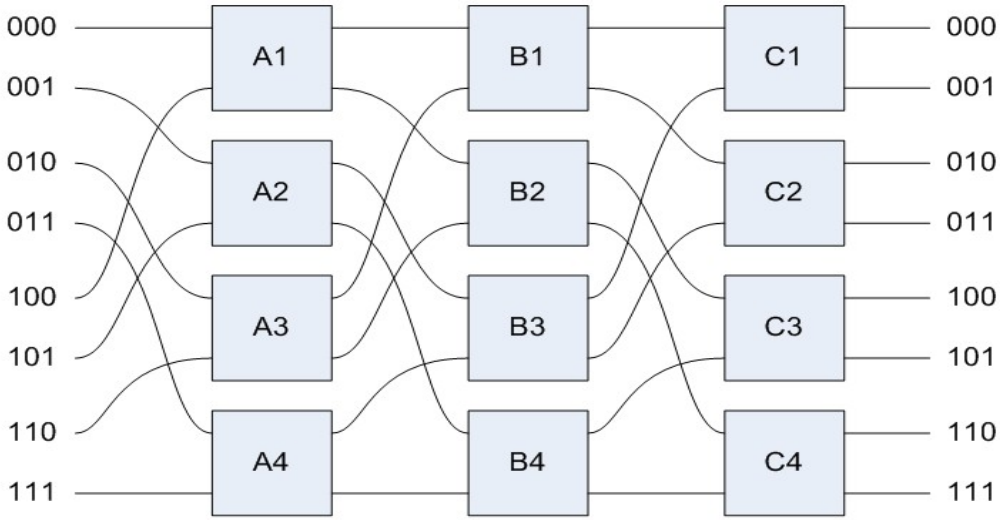
(b) ring



(c) mesh



(d) 2D Torus



(e) Omega Network

Network Routing

- Aims
  - Avoid deadlock
  - Minimize latency
  - Avoid hot-spots
- Major types
  - Oblivious -- fixed policy
  - Adaptive -- takes into account hot-spots and congestion

Oblivious Routing

- X-Y routing
  - First move along the X-axis, and then the Y-axis
- Y-X routing
  - Reverse of X-Y routing

Adaptive Routing

Route from $\underbrace{X\_S, Y\_S}_{\text{source}}$ to $\underbrace{X\_T, Y\_T}_{\text{target}}$

- West-first
  - If X_T ≤ X_S use X-Y routing
  - Else, route adaptively
- North-last
  - If Y_T ≤ Y_S use X-Y routing
  - Else, route adaptively
- Negative-first
  - If (X_T ≤ X_S || Y_T ≤ Y_S) use X-Y routing
  - Else, route adaptively

Flow Control

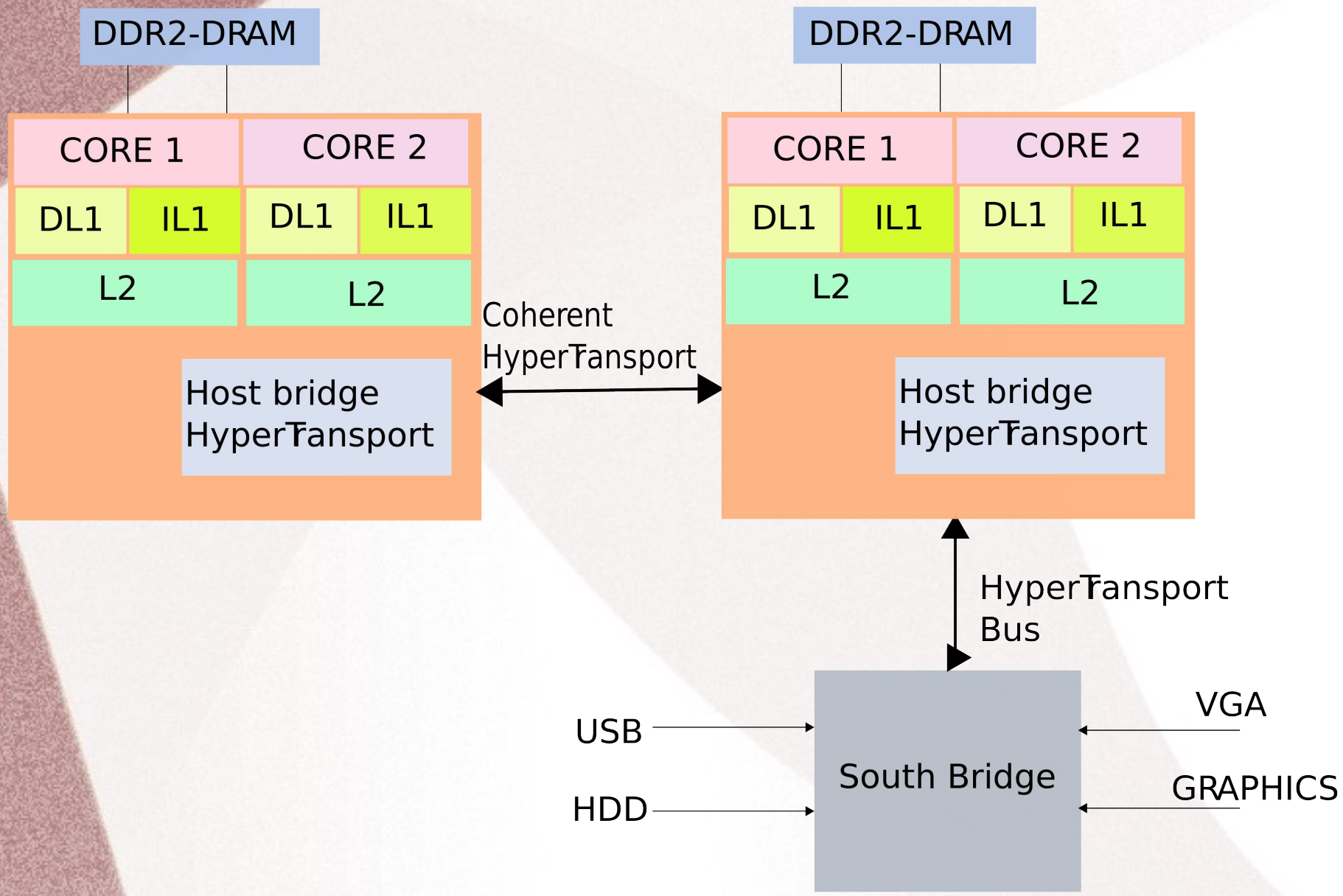
- Store and forward
  - A router stores the entire packet
  - Once it receives all the flits, sends it onward
- Wormhole routing
  - Flits continue to proceed along outbound links
  - They are not necessarily buffered

Flow Control - II

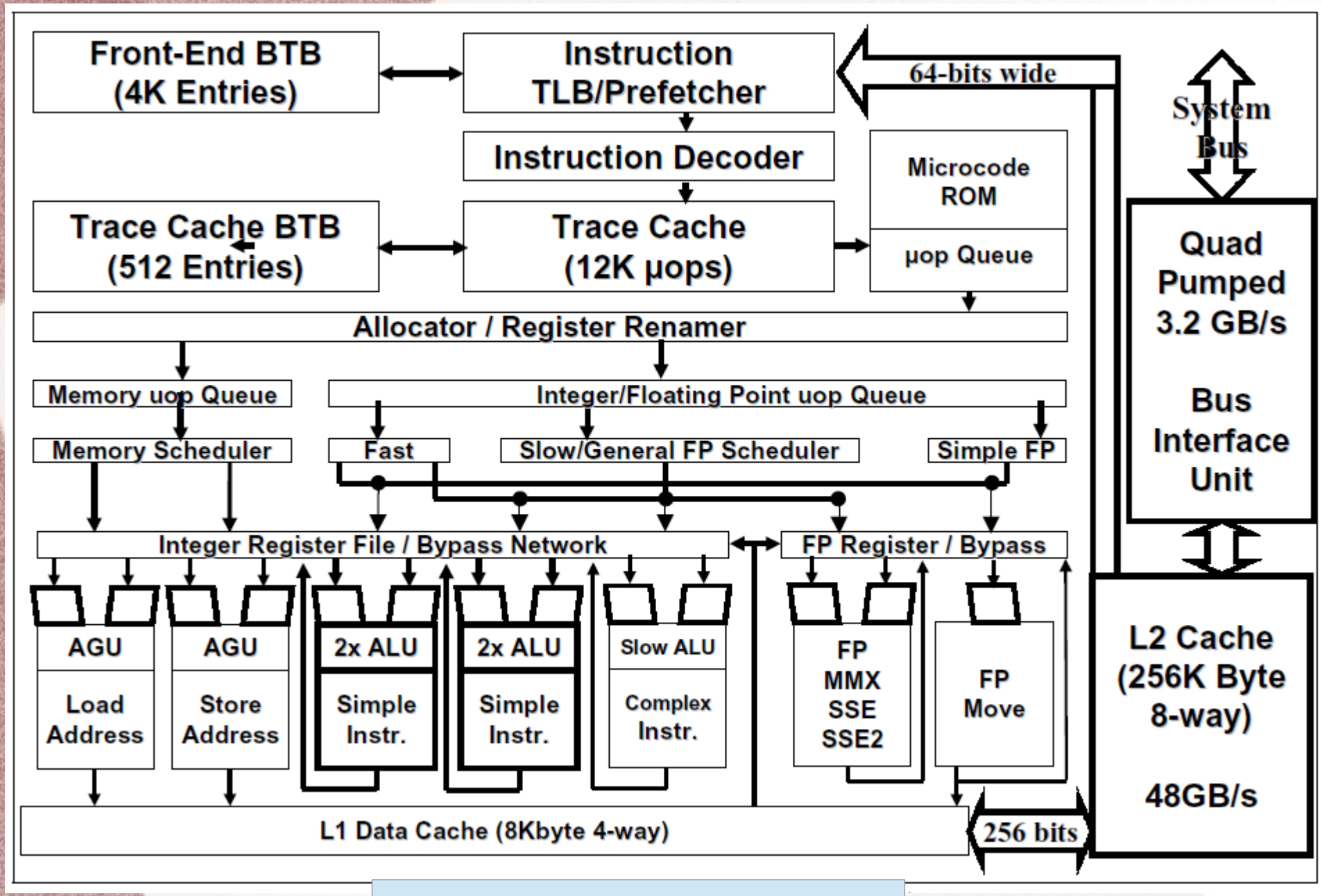
- Circuit switched
  - Resources such as buffers and slots are pre-allocated
  - Low latency, at the cost of high starvation
- Virtual channel
  - Allows multiple flows to share a single channel
  - Implemented by having multiple queues at the routers

Part III -- Examples

AMD Opteron

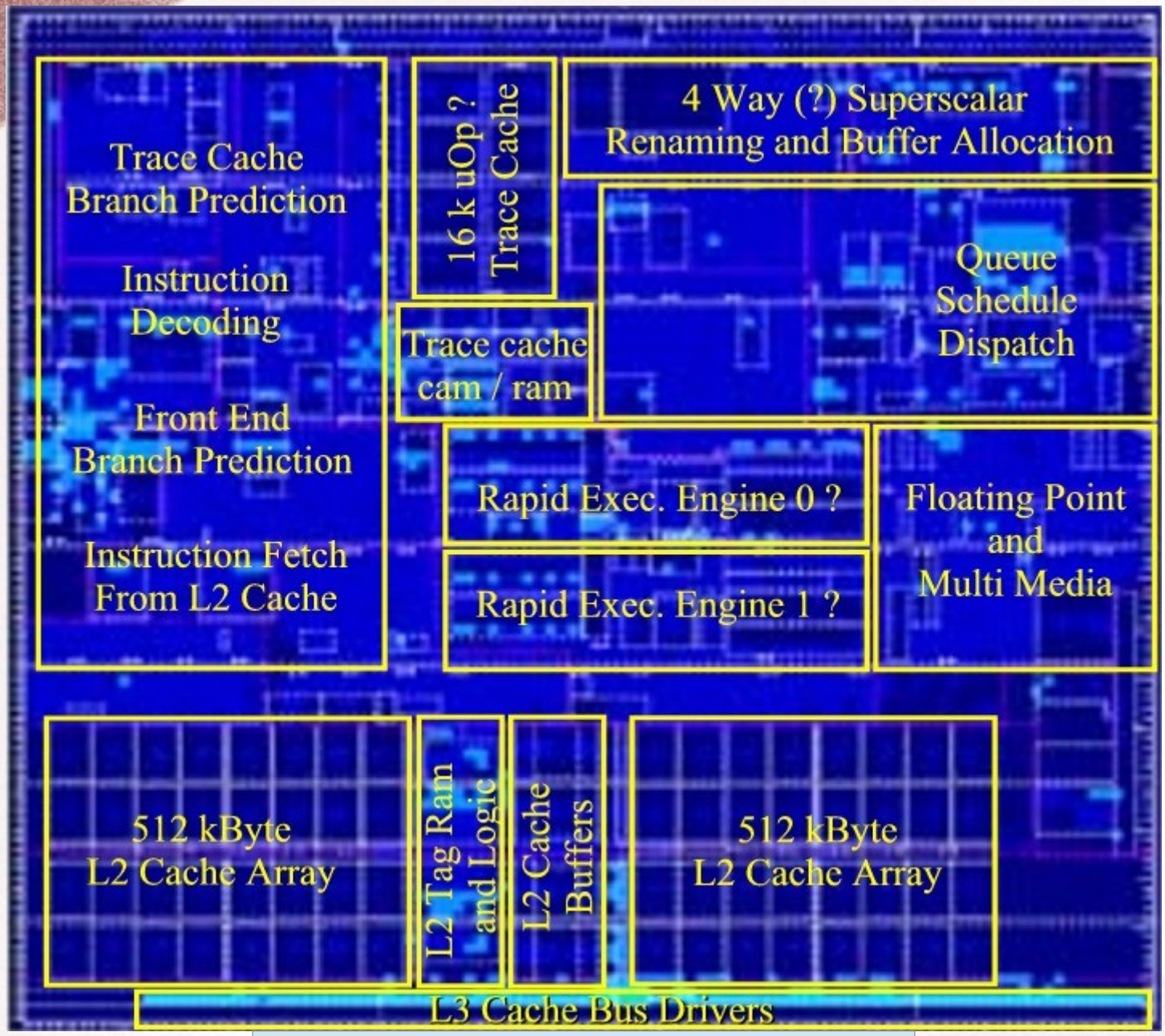

DDR2-DRAM
CORE 1
CORE 2
DL1
IL1
DL1
IL1
L2
L2
Host bridge
HyperTransport
Coherent
HyperTransport
DDR2-DRAM
CORE 1
CORE 2
DL1
IL1
DL1
IL1
L2
L2
Host bridge
HyperTransport
HyperTransport
Bus
USB
HDD
South Bridge
VGA
GRAPHICS

Intel - Pentium 4



source: Intel Techology Docs

Intel Prescott

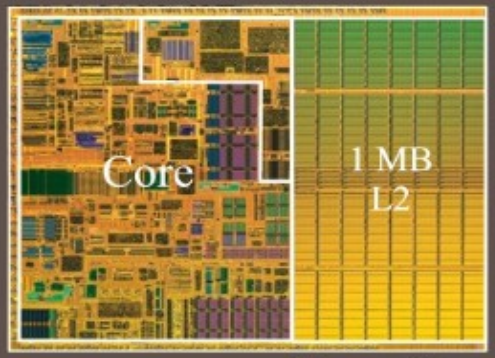


source: chip-architect.com

Intel Core family picture. March.28.2007
Banias / Dothan / Yonah / Merom / Penryn

All dies at the same absolute scale. L2 sram area numbers exclude L2 tags

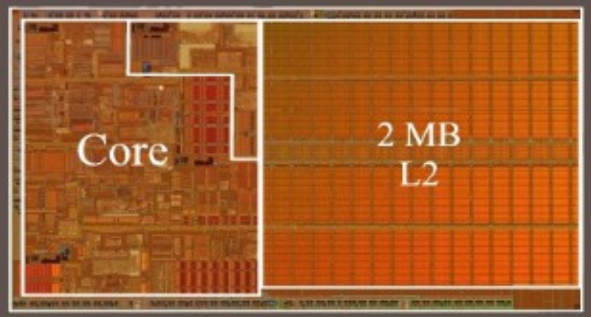
Intel Banias SC 130 nm



10.5 mm x 7.9 mm

Die: 82.8 mm2
Core: 39 mm2, L2 sram: 29 mm2/MB

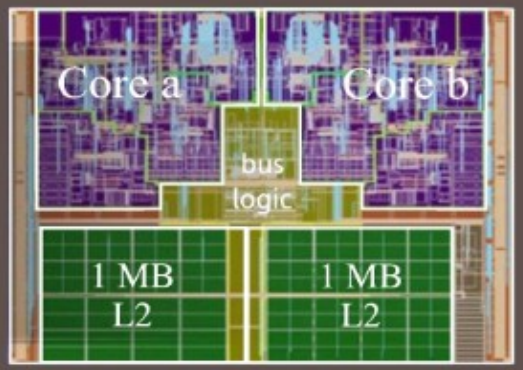
Intel Dothan SC 90 nm



12.6 mm x 6.9 mm

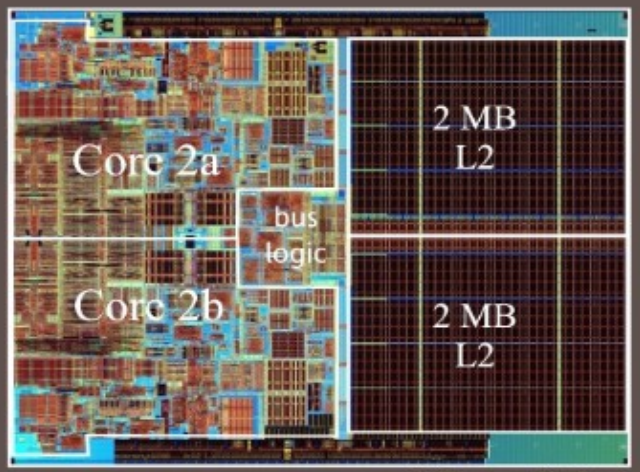
Die: 87.7 mm2
Core: 28 mm2, L2 sram: 19.5 mm2/MB

Intel Yonah DC 65 nm



Die: 91.0 mm2
Core: 19.7 mm2, L2 sram: 12.4 mm2/MB

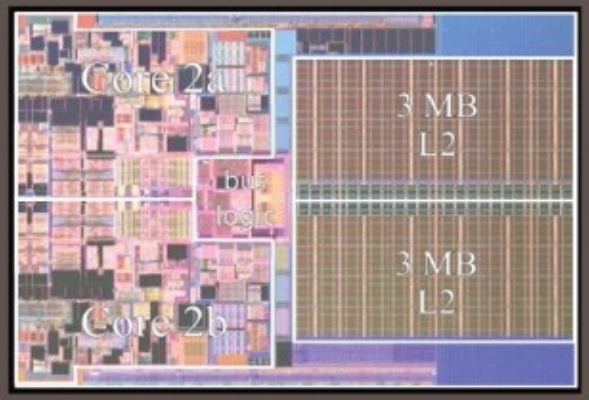
Intel Merom DC 65 nm



13.67 mm x 10.47 mm

Die: 143 mm2
Core: 31.5 mm2, L2 sram: 12.4 mm2/MB

Intel Penryn DC 45 nm

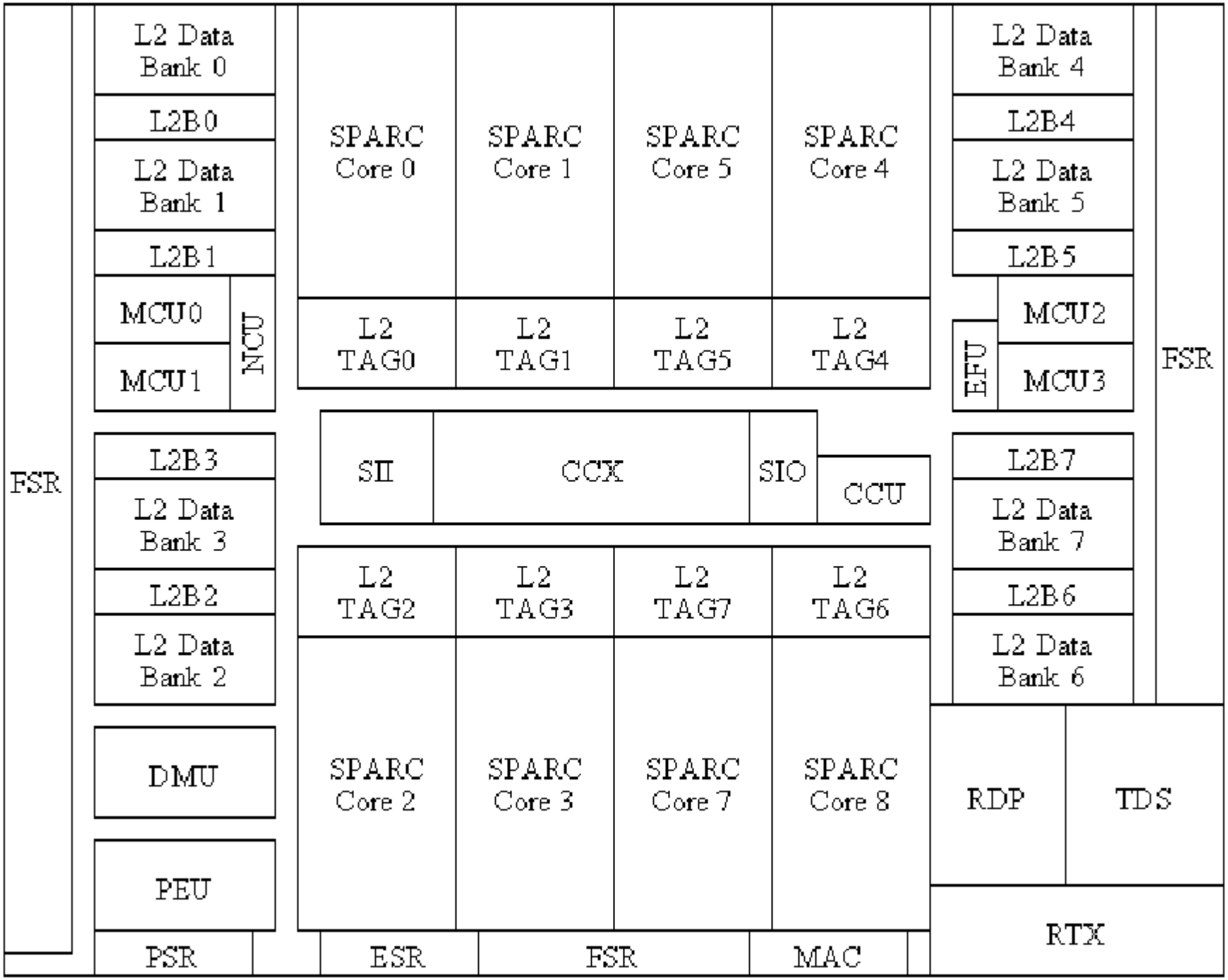


12.3 mm x 8.6 mm

Die: 107 mm2
Core: 22.0 mm2, L2 sram: 6.0 mm2/MB

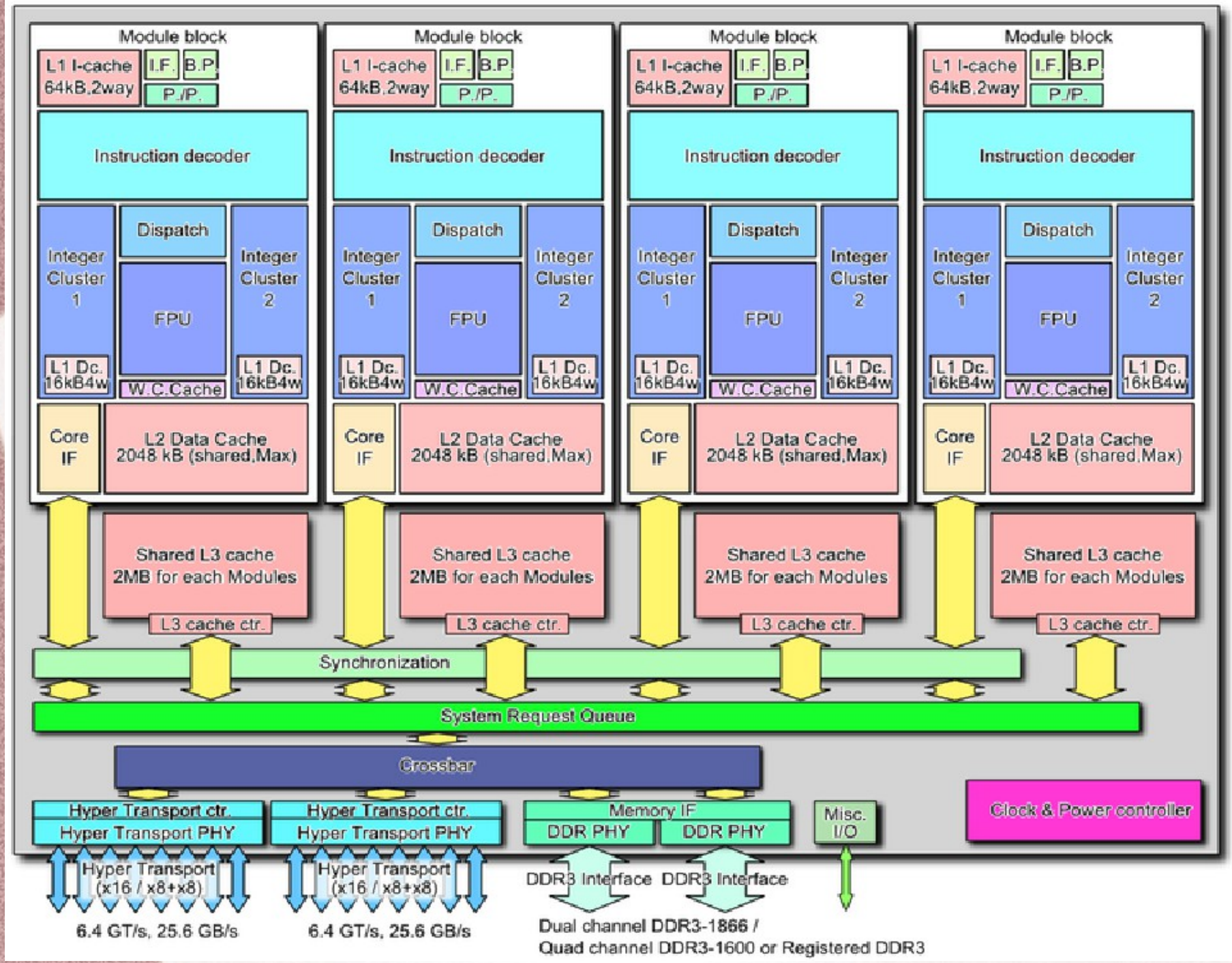
source: chip-architect.com

UltraSparc T2



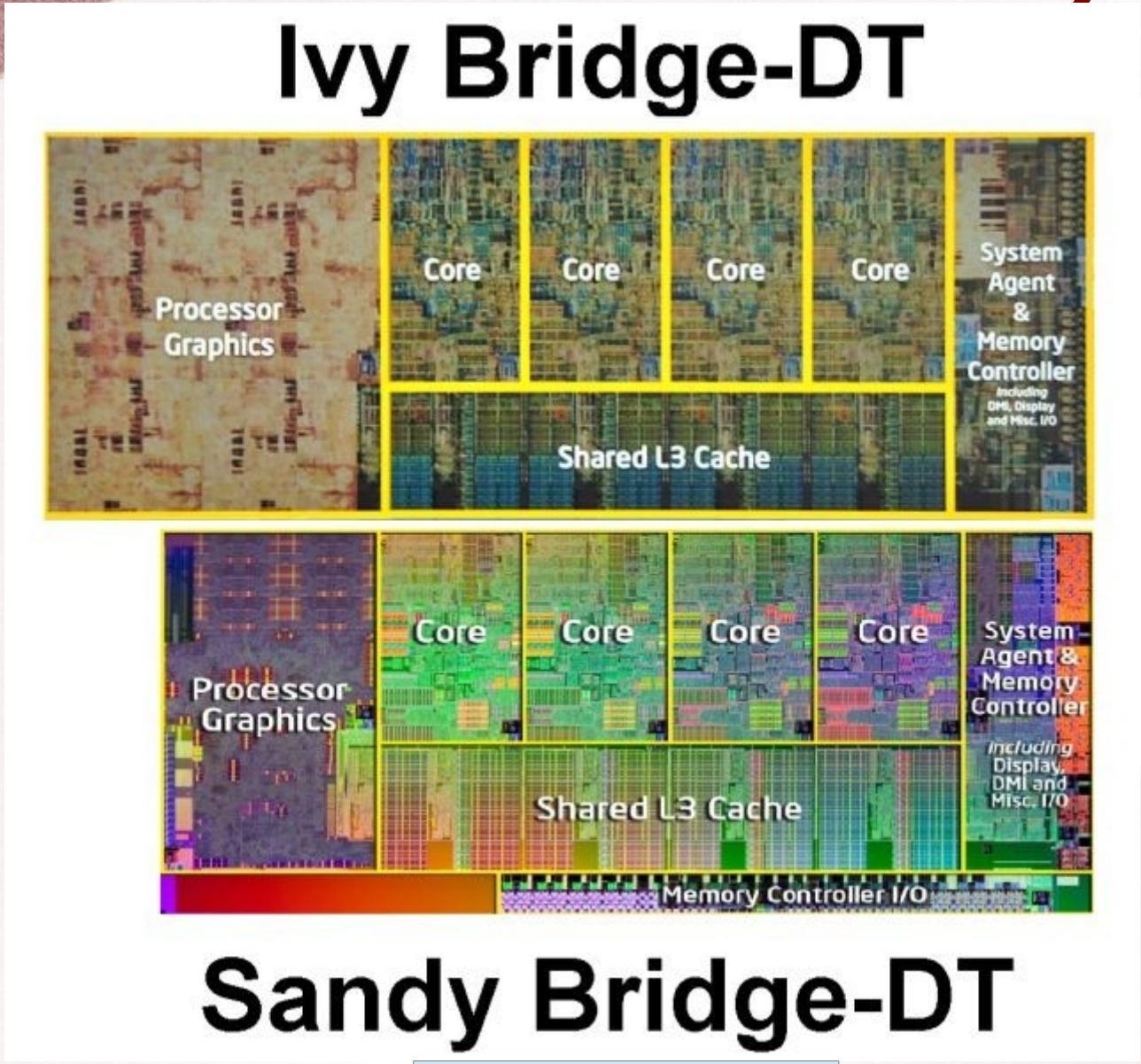
source: wikipedia

AMD: Bulldozer



source: wikipedia

Intel Sandybridge & Ivybridge



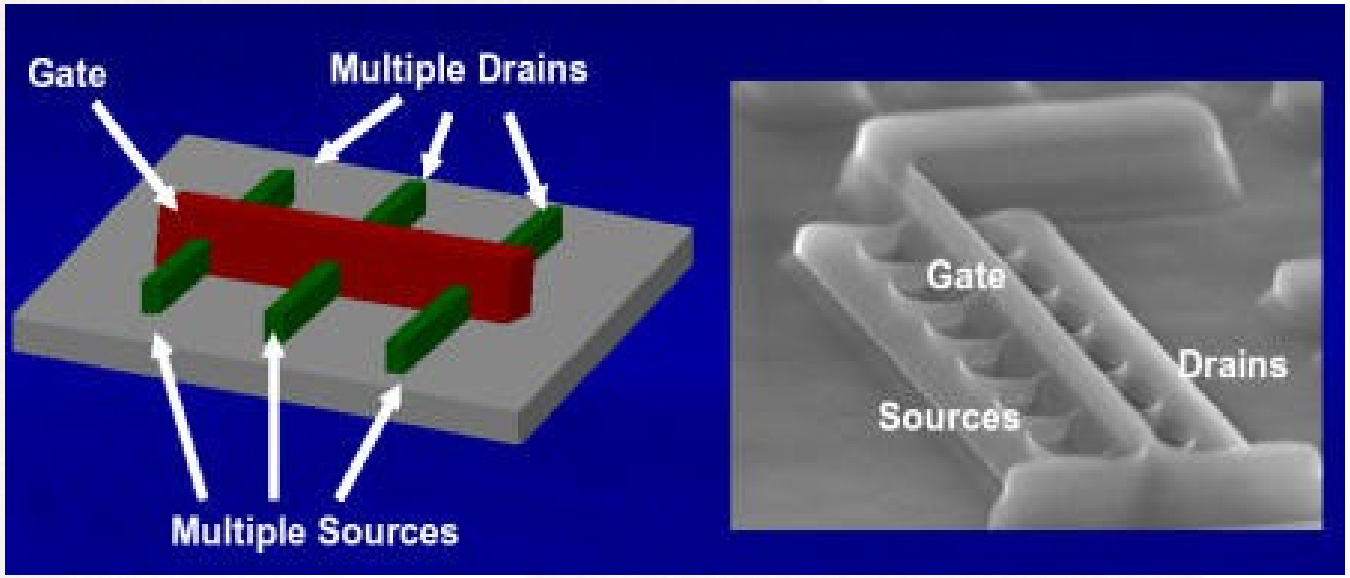
source: itportal.com

Intel Ivybridge

- Micro-architecture
  - 32KB L1 data cache + 32 KB L1 instruction cache
  - 256 KB coherent L2 cache per core
  - Shared L3 cache (2 - 20 MB)
  - 256 bit ring interconnect
  - Upto 8 cores
  - Roughly 1.1 billion transistors
  - Turbo mode - Can run at an elevated temperature for upto 20s.
  - Built-in graphics processor

Revolutionary Features

- 3D Tri-gate Transistors based on FinFets



source: wikipedia

- Faster operation
- Lower threshold voltage and leakage power
- Higher drive strength

Enhanced I/O Support

- PCI Express 3.0
- RAM: 2800 MT/s
- Intel HD Graphics, DirectX 11, OpenGL 3.1, OpenCL 1.1
- DDR3L
- Configurable thermal limits
- Multiple video playbacks possible

Security

- RdRand instruction
  - Generates pseudo random numbers based on truly random seeds
  - Uses an on-chip source of randomness for getting the seed
- Intel vPRO
  - Possible to remotely disable processors or erase hard disks by sending signals over the internet or 3G
  - Can verify identity of users/ environments for trusted execution

Thank You

Slides can be downloaded at:

source: http://www.cse.iitd.ac.in/~srsarangi/files/drdo-pres-july18-2012.odp
pdf : http://www.cse.iitd.ac.in/~srsarangi/files/drdo-pres-july18-2012.pdf