Summary and Review

Professor Hugh C. Lauer
Professor Thérèse M. Smith

CS-4515, System Programming Concepts

This Course

- Essentially an *in-depth* survey course of the architecture of computers …
- … using an advanced graduate textbook as a guide
- *NOT* trying to prepare you for careers in computer design, …
- … but rather trying to give you an appreciation for all of the complexity, creativity, and diversity already in the computers that you use every day

In the background

- **Moore's Law**

- **Growth rate in transistor count per chip doubles every 18-24 months**
  - That computer you brought to campus as a freshman was too slow by the time you were a junior-senior!

- **However, …**
  - All good things must come to an end …
  - … even Moore's Law
    - … as the press has been reminding us in the past two years (or so)!

Making a chip

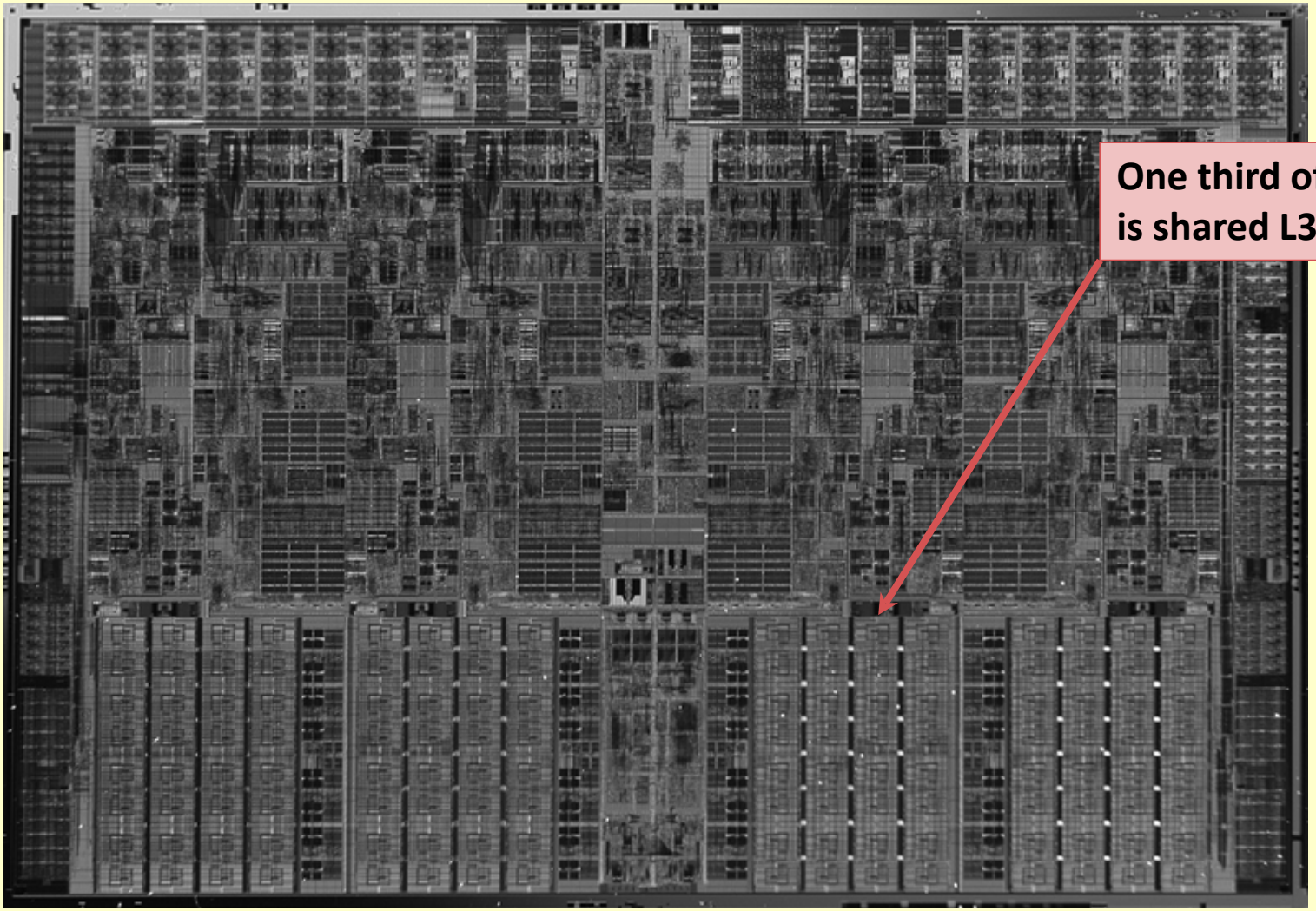


Figure 1.13 Photograph of an Intel Core i7 microprocessor die, which is evaluated in Chapters 2 through 5. The dimensions are 18.9 mm by 13.6 mm (257 mm2) in a 45 nm process. (Courtesy Intel.)

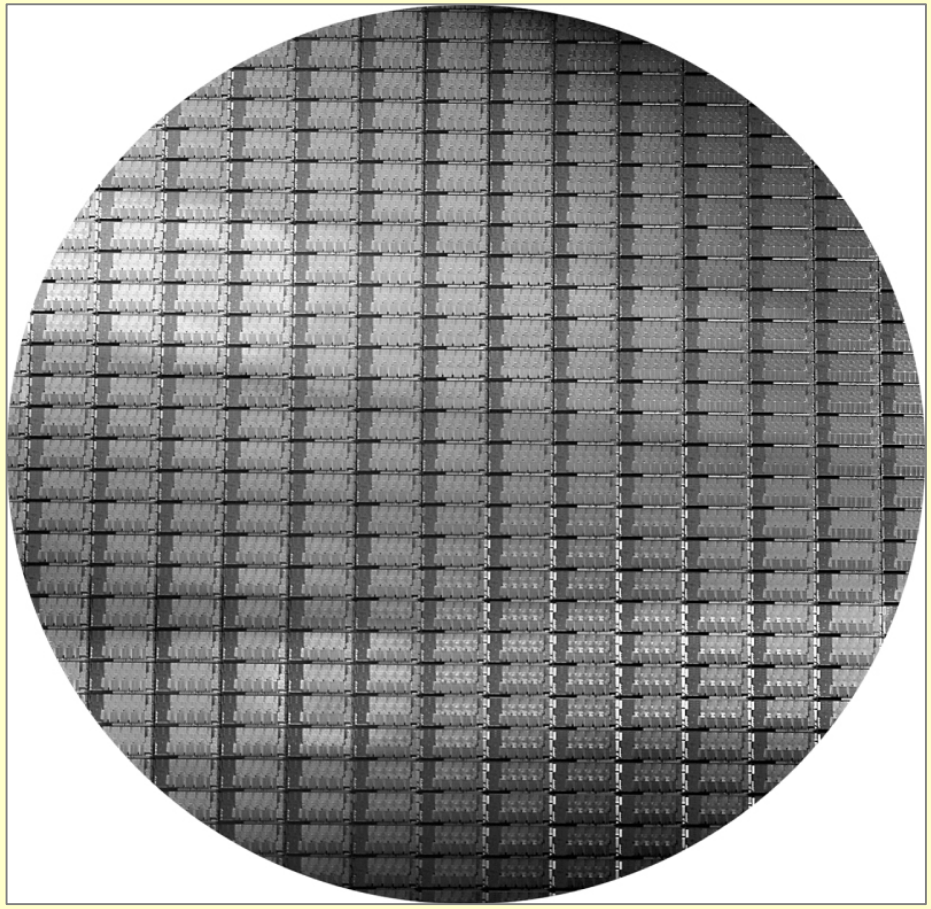


Figure 1.15 This 300 mm wafer contains 280 full Sandy Bridge dies, each 20.7 by 10.5 mm in a 32 nm process. (Sandy Bridge is Intel's successor to Nehalem used in the Core i7.) At 216 mm2, the formula for dies per wafer estimates 282. (Courtesy Intel.)

Cost of Integrated Circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$5000–6000

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

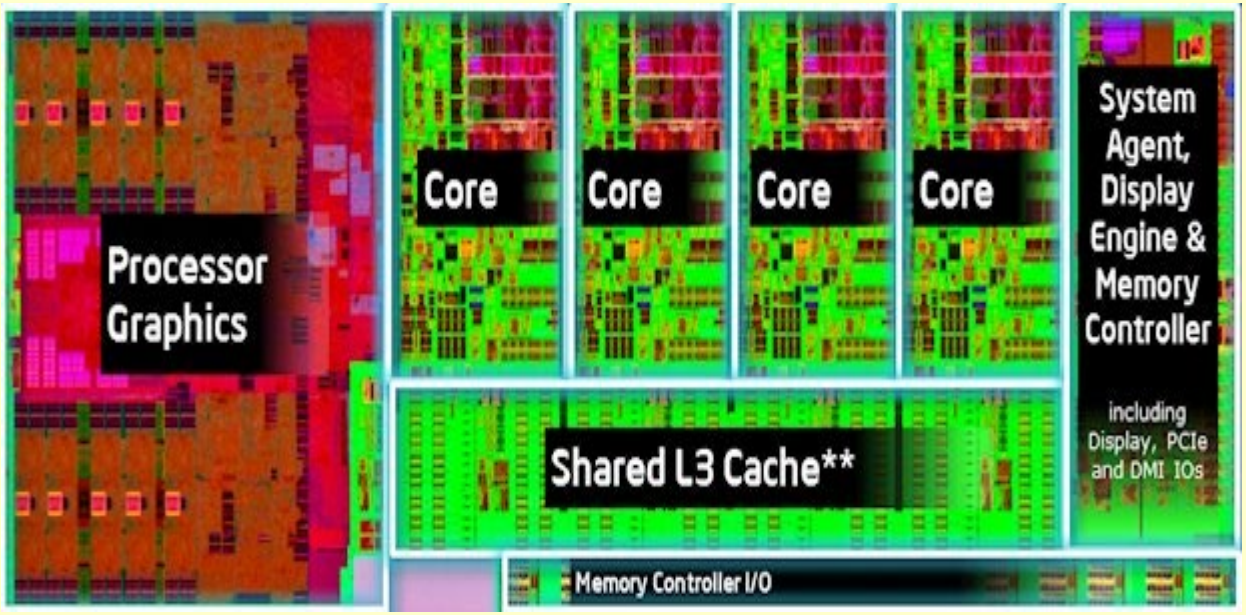
Accounts for partial dies at edge of wafer

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

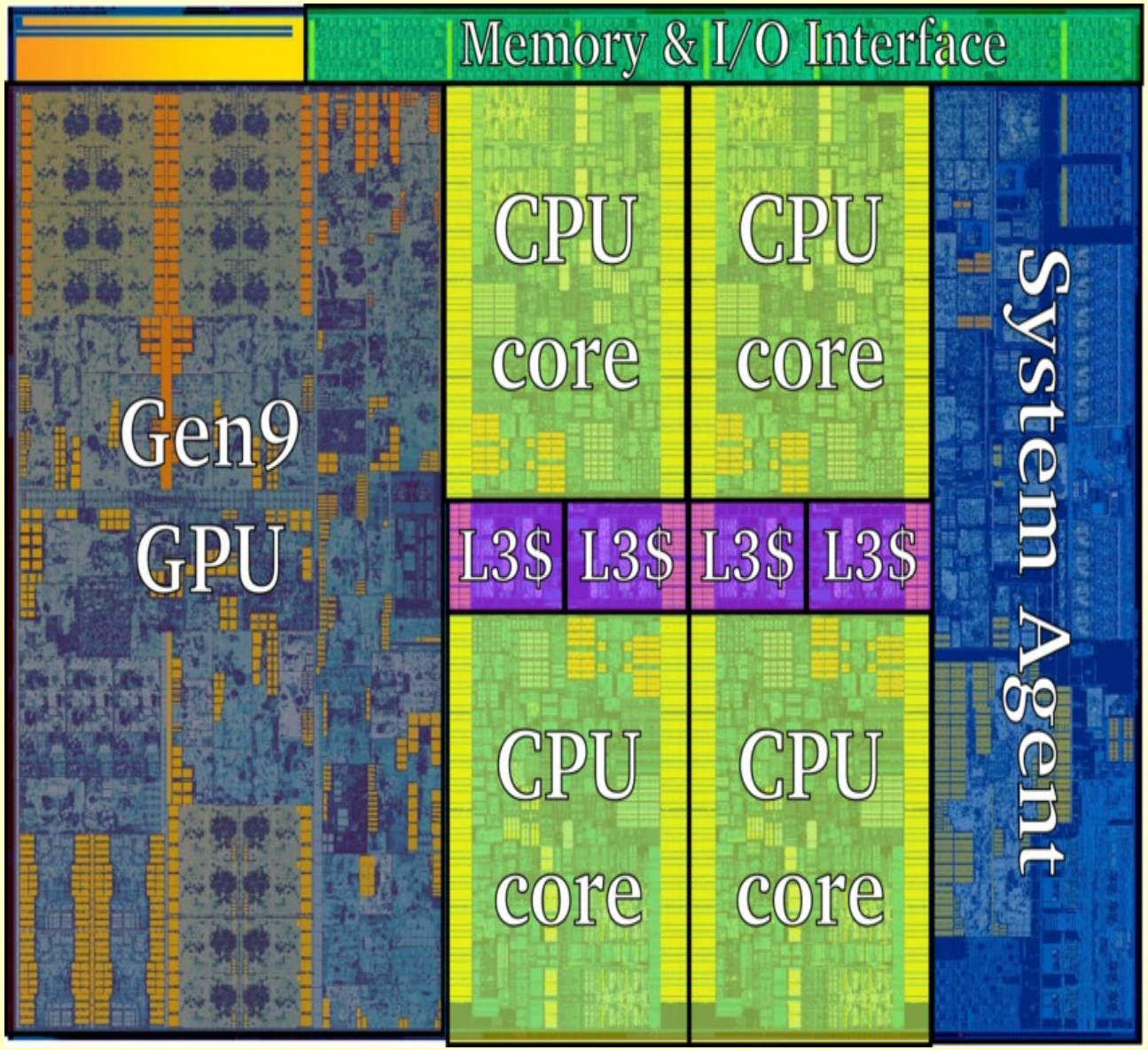
Bose-Einstein formula (empirically derived)

Based on semi-conductor process

Haswell


Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

Sky Lake


Memory & I/O Interface
Gen9
GPU
CPU
core
CPU
core
L3$
L3$
L3$
L3$
CPU
core
CPU
core
System Agent

More Background

- ***Dennard Scaling***
  - 1974 paper by Robert Dennard
  - Power density on semiconductor chips stays constant
  - I.e., as feature size gets smaller, can add more transistors without increasing total power!
  - I.e., a free ride for processor performance & complexity over past several decades!

- **Not even mentioned in Hennessy & Patterson 5th**
  - A brick wall for processor performance in 2017-18
  - Highly featured in Hennessy & Patterson 6th edition!

Comments? Questions? Thoughts?

How to get performance

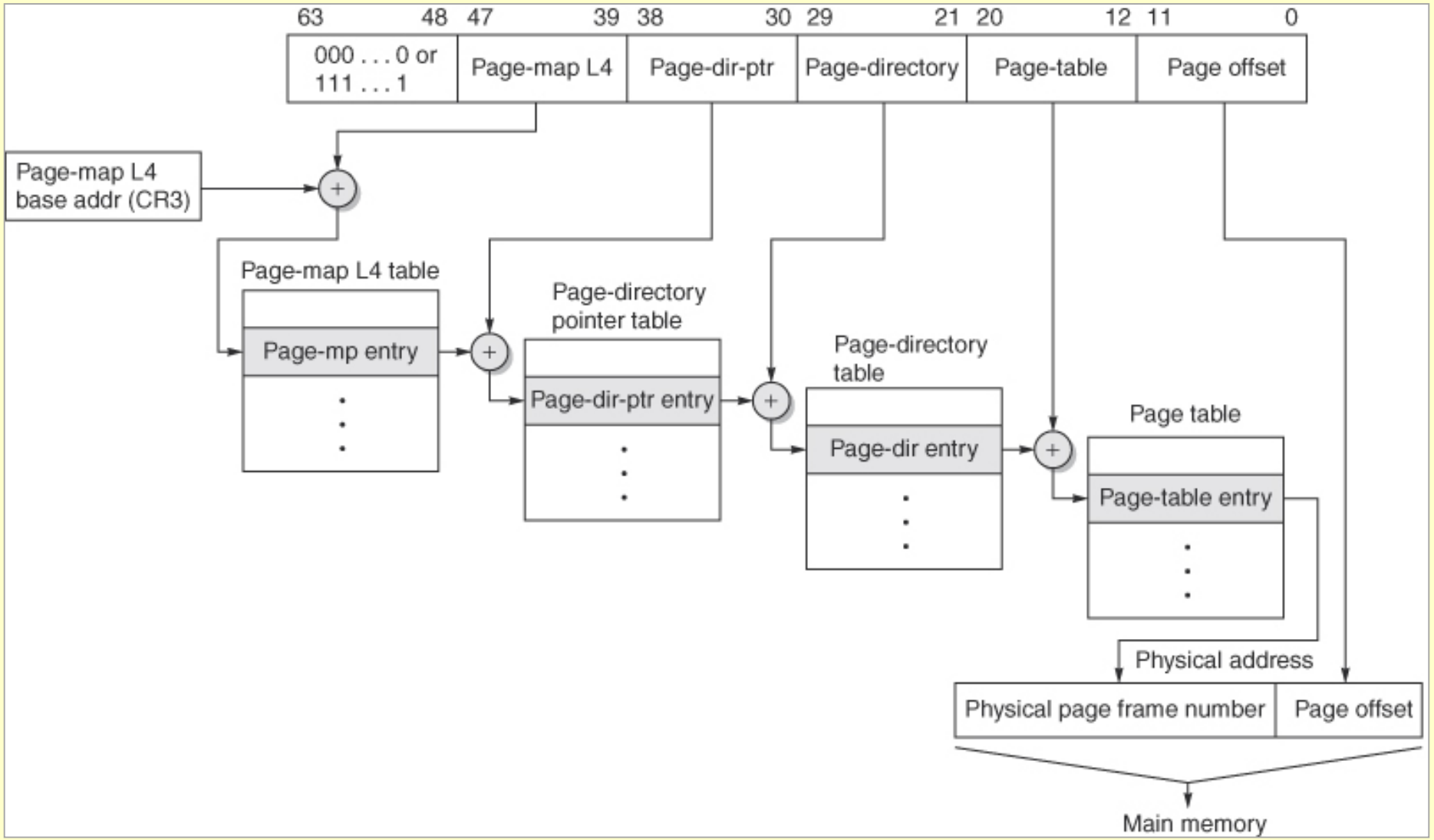
- **Caching**
- **Lots and lots of caching!**
  - Multiple levels — i.e., cach
  - Separate I- and D-caches
  - TLB's, many other specialized hardware buffers, etc.
- **Caching is ubiquitous of *ALL* system design**
- **… hardware *and***
- **…software**

In the Professor's opinion, *caching* is the *most important* system design topic to be taught in all of Computer Science

And many applications, also

Fig B.27

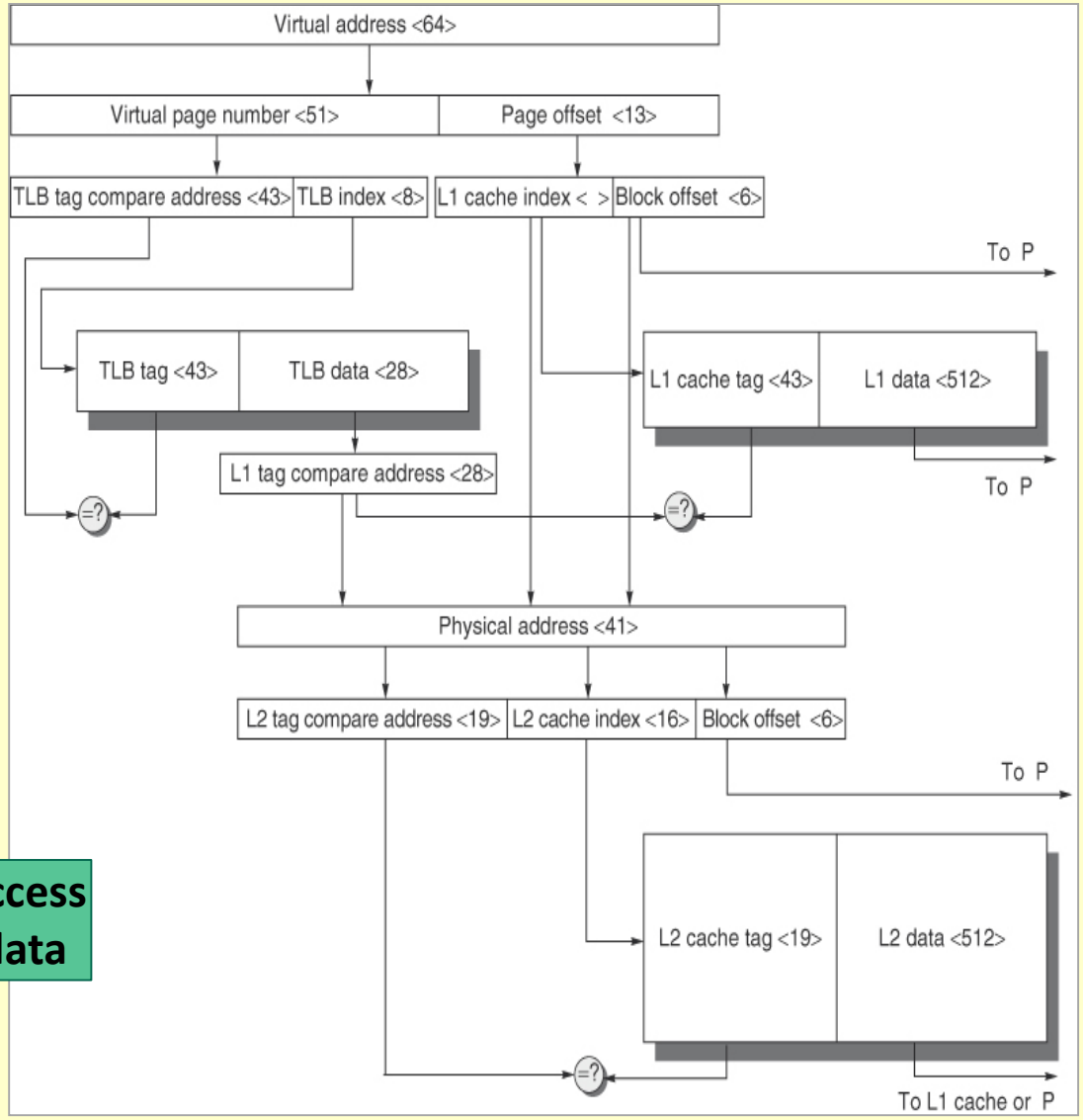
Page table structure — x86_64



Hypothetical architecture

Check L1 cache tag by *physical address*

Why?



Index L1 cache by *virtual address*

Result: 1-cycle access to most active data

Fig. B.25

Caching in Operating Systems

- **All computation happens in *virtual memory***
- **Physical memory is just a cache of virtual memory**
- **All of the OS topics pertaining to virtual memory management get simpler when regarded as caches — e.g.,**
  - Page replacement algorithms
  - Pre-fetch techniques
  - Etc.

Never in my career have I heard others apply caching principals to virtual memory subsystems

Beyond Caching

- Parallelism
- Lots and lots of parallelism
- Doing multiple things at the same time!
- Instruction-level parallelism
- … Data-level parallelism
- … Thread-level parallelism
- … Request-level parallelism

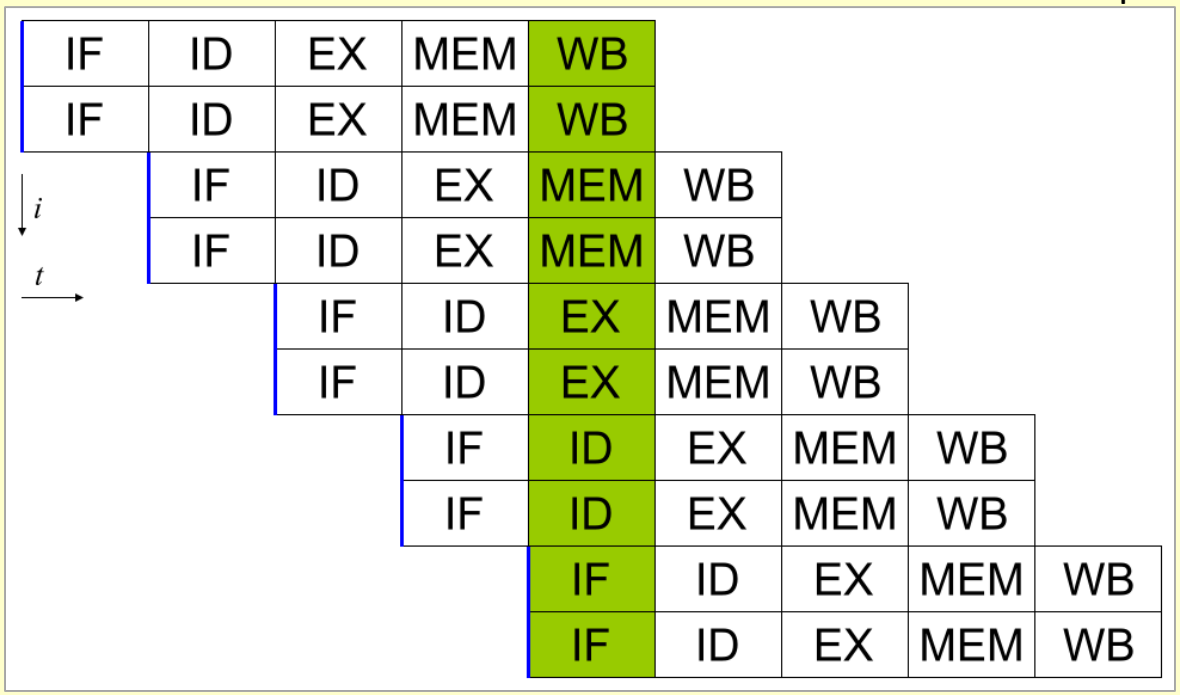
Instruction-level parallelism

- Dates back to 1960s !
- Start one instruction before another has finished
- Out-of-Order execution
- Scoreboards
- … Tomasulo's Algorithm (1960s)
- … Multiple instructions in "flight" at once
- … Forward results to future operations

Definition — "Superscalar"

- **The ability to execute more than one instruction per cycle**
  - In a single processor
  - From a *single* instruction stream

Naïve example



"Superscalarpipeline" by Amit6, Wikipedia

Instruction-Level Parallelism

- **When exploiting instruction-level parallelism, goal is to minimize CPI**
  - Pipeline CPI =
    - Ideal pipeline CPI +
    - Structural stalls +
    - Data hazard stalls +
    - Control stalls
- **Parallelism within basic blocks is limited**
  - Typical size of basic block = 3-6 instructions
  - Must optimize across branches — i.e.,
    - Loops
    - Speculation

Basic Block (definition):–
A straight-line code sequence, with no branches *in* except entry and no branches *out* except exit. (p. 149)

Hazards

Definition:–

Dependence: **A needs output of B to continue**

Manifests itself in *Hazards*

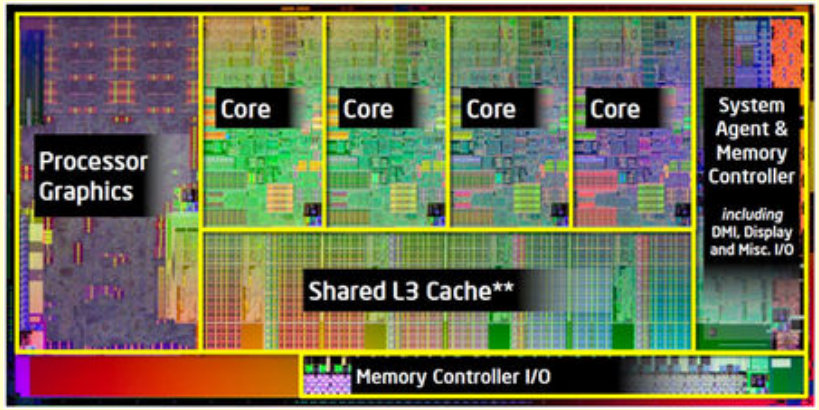
- **RAW — operand to be used does not have a value yet**
- **WAW — Shared register/result written multiple times (in arbitrary order)**
- **WAR — attempt to write before previous value has been read**

Tomasulo's algorithm

- ***Issue*** **instructions in program order**
- **Each instruction waits for operand results from previously-issued instructions**
- **Instructions execute when all operands are available**
- **Instructions are *retired* in program order**
  - To avoid WAW and WAR hazards

What is Thread-Level Parallelism?

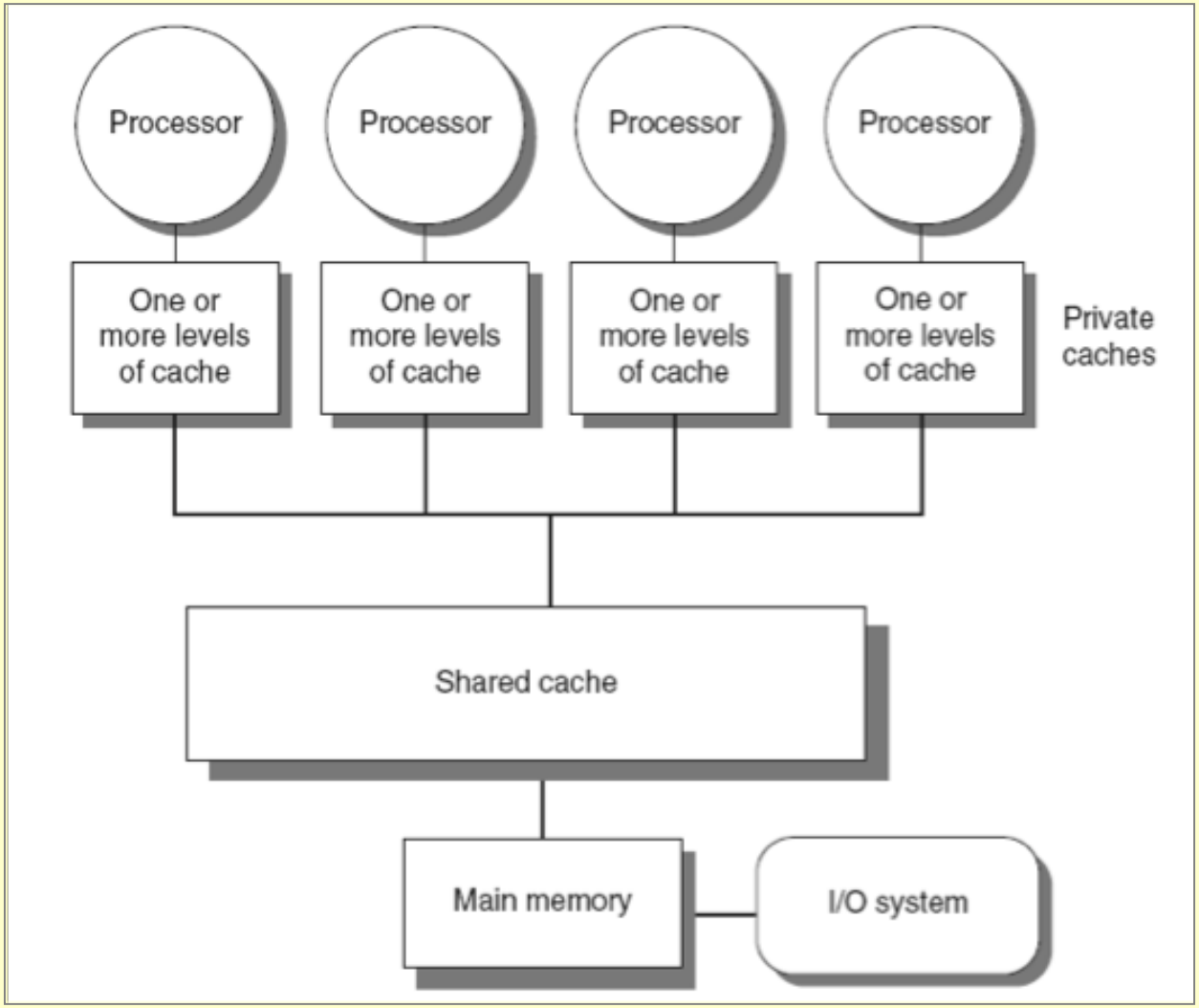
- **Multiple processors**
  - Not an optimization for a single processor
  - Multiple Program Counters
  - Shared Memory
- **MIMD model**
- **Implementation**
  - Multithreaded processor
  - Multicore processor
  - Multi-processor



Why Thread-Level Parallelism?

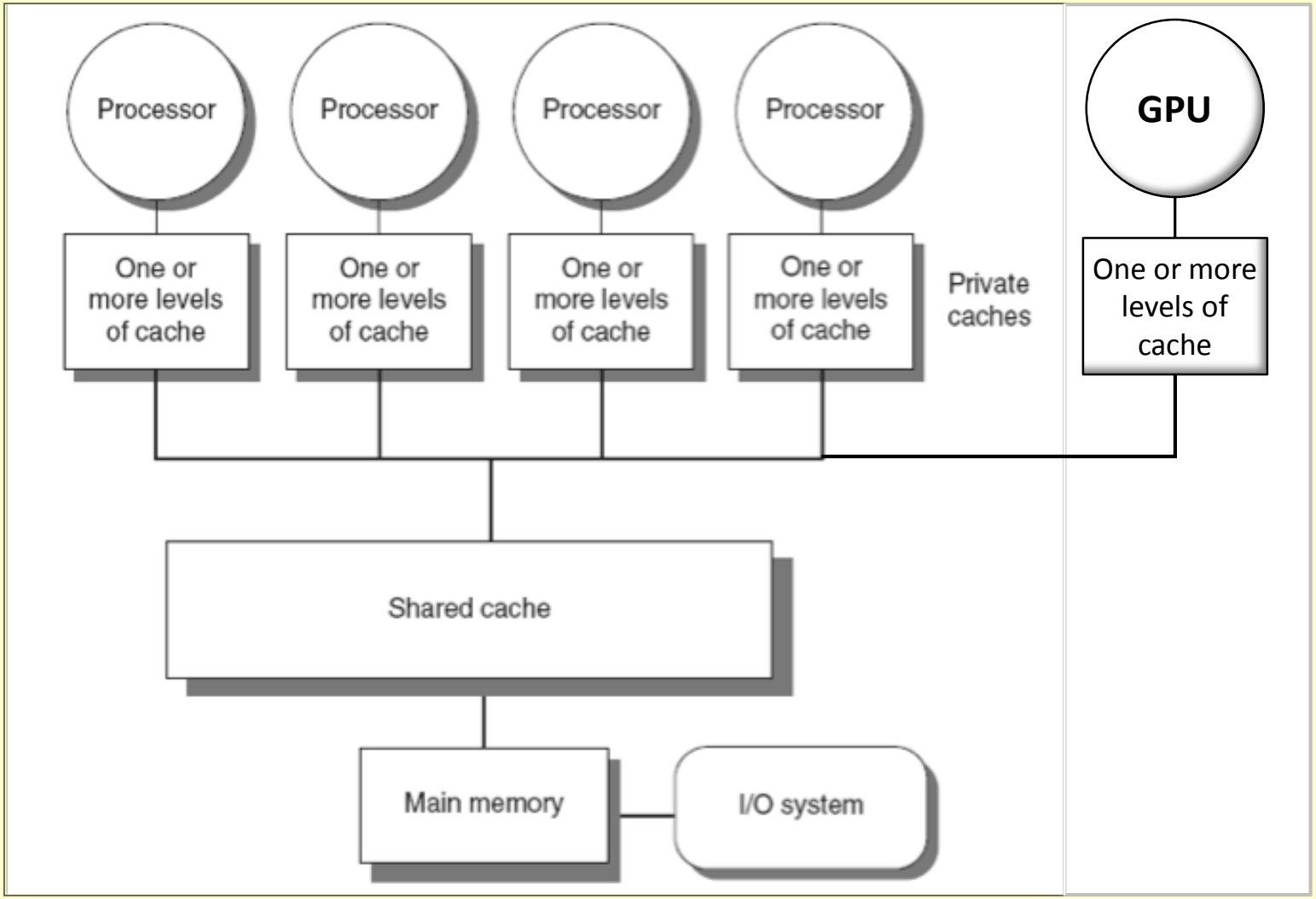
- **Clock Speed Limit**
  - Plateau at 3-4 GHz in ~2005
  - Thermal problems!
  - Increases < 100 Mhz per year
- **ILP Limit**
  - Limited by interdependence of code
  - Diminishing returns

SMP



Modern multicore processors

SMP — non-uniform access patterns



Thread-level Parallelism

- Multiple processors ⇒
- Multiple caches ⇒
- (Possibly) Multiple main memories
- All under the control of a single operating system
- … with many multithreaded applications

Coherency

- **What is coherency?**
- **Why do we need it?**
- **Answers:–**
  - Applications are multi-threaded ⇒ all threads must have coherent view of application data
  - Operating systems are multi-threaded ⇒ all threads must have coherent view of OS data structures
    - E.g., Ready queue, task lists, memory management tables, etc.

Definition

- ***Wait-free***
  - Implementation of a “concurrent” object that guarantees that any process/thread can complete an operation in a finite number of steps
  - No process/thread can be prevented from completing the operation by undetected halting or failures of another process
  - No process/thread can be prevented from completing the operation by arbitrary variations in speed

Compare-and-swap()

- **Equivalent to *LL* + *SC***
  - I.e., Compare-and-swap() can be implemented by LL + SC
  - LL + SC can be implemented by Compare-and-swap()
- **Included in Intel architectures as *CMPEXCH***
- **Included in all IBM architectures since System 370**
  - Early 1970s
- **(*LL* + *SC*) & *CMPEXCH* depend crucially on cache coherency mechanisms**

Meanwhile …

Data parallelism

- **Vector processors**
  - SIMD
  - Cray and Cray-like architectures
- **GPUs**
  - Many threads, many lanes per thread

GPU Pipelining

- **Like CPU except shared mem has coordinated reads**
  - Bandwidth of shared memory = 32*#banks per cycle
  - Optimize for data parallelism
- **Small registers in SIMTs, tiers of sharing**
- **Address Coalescing**
  - Because of the shared overhead
  - Make aligned regions fetch into corresponding threads
  - Aligned & unpermuted regions

Asynchronous processors

ORDVAC / ILLIAC I (1952)

- ILLIAC - Illinois Automatic Computer
- ORDVAC - Ordnance Discrete Variable Automatic Computer
- Developed concurrently, used same instruction set
- Based on IAS computer by John von Neumann
- ADD - 72µs
  MUL - 732µs

ORDVAC Mainframe

Warehouse-scale computers (WSCs)

- Provide Internet services
  - Search, social networking, online maps, video sharing, online shopping, email, cloud computing, etc.
- Differences from HPC "clusters"
  - Clusters have higher performance processors and network
  - Clusters emphasize thread-level parallelism, WSCs emphasize request-level parallelism
- Differences from datacenters
  - Datacenters consolidate different machines and software into one location
  - Datacenters emphasize virtual machines and hardware heterogeneity in order to serve varied customers

Also

- Lots of other stuff
- Homework exercises similar to a design course

On the Horizon

- ***Deep Learning Networks***
  - *Aka* Neural Nets
- ***Lots of multiplications!***
- **Domain Specific Architectures**
  - Specialized for “training” and operation
- **On-line & Cloud Services**
  - Google Cloud Machine Learning Engine

Neural Net

- **Array of input values**
  - 1-, 2-, or more-dimensional
- **Array of *weights* to calculate a new "layer" from input**
- **Multiple layers, each derived from previous layer using own weights**
- ***Training* protocols to derive weights from sample inputs**

Specialized architectures

- **Support many layers**
- **Support large arrays of inputs**
- **High speed recognition of**
  - Images
  - Patterns
  - Objects
  - … Etc.

Questions? Comments? Thoughts?

Thank you for your attention. Thank you for your participation.