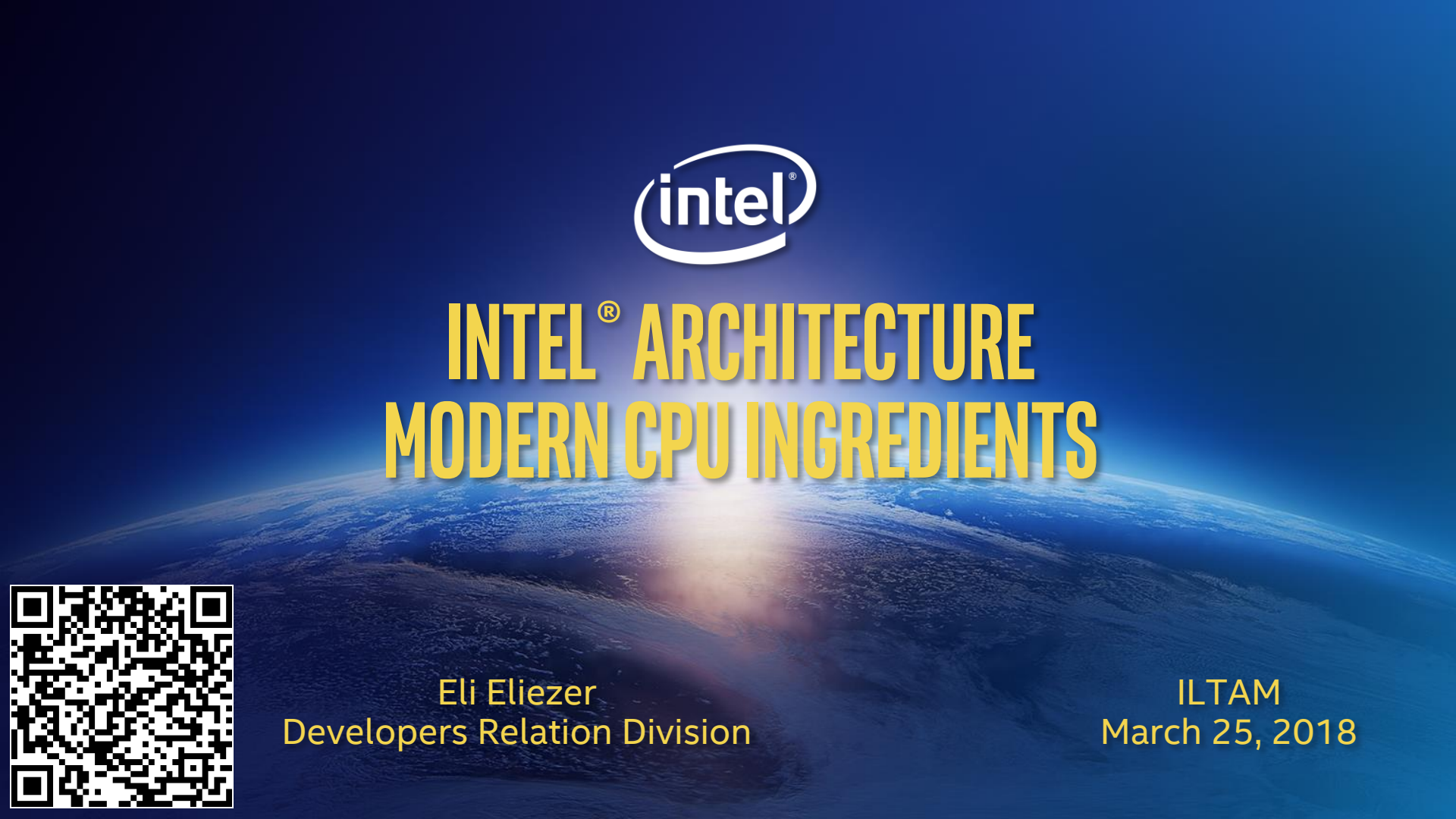


Eli Eliezer
Developers Relation Division

ILTAM
March 25, 2018


intel
CORPORATE OVERVIEW

INTEL'S VISION

Things

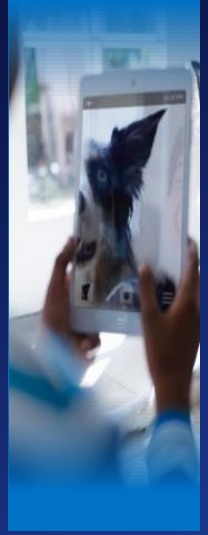
Network

Cloud/Data Center

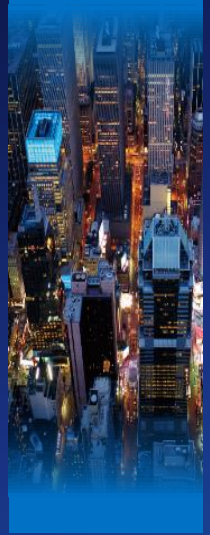
If it's **smart** and **connected**, it's **best** with **Intel**.

INTEL DELIVERS END TO END SOLUTIONS

CLIENT COMPUTING CCG

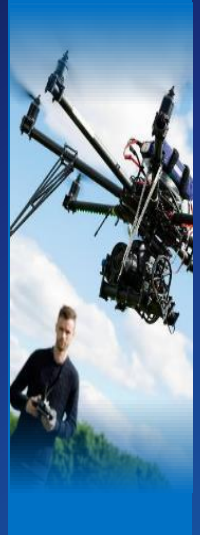


INTERNET OF THINGS IOTG



AUTOMATED DRIVING ADG

NEW TECHNOLOGY NTG

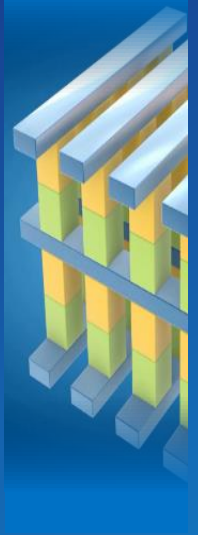


DATA CENTER DCG

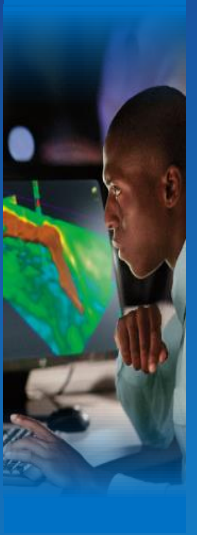
ARTIFICIAL INTELLIGENCE AIPG

COMMUNICATIONS ICDG

NON-VOLATILE MEMORY NSG



PROGRAMMABLE SOLUTIONS PSG



INTEL DELIVERS END TO END SOLUTIONS

Software Tools

Wind River® Simics

Platform Security

Intel® Data Analytics Acceleration Library (Intel® DAAL)

Intel® Context Sensing

Intel® CoFluent™ Technology

Intel® Data Center Manager

IoTivity

Ecosystem Enabling

Intel® Developer Zone

OPEN CONNECTIVITY FOUNDATION™

hadoop

OpenJDK

openstack

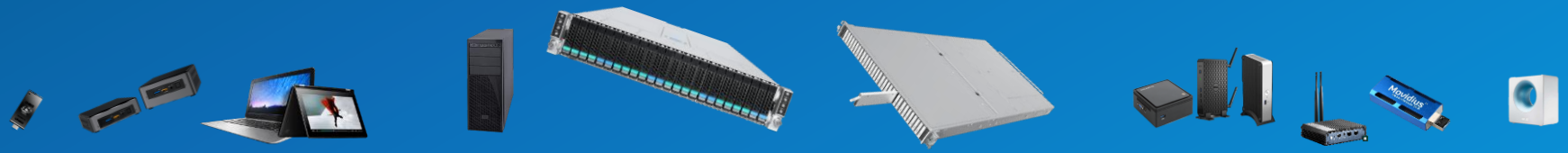
Spark

cloudera

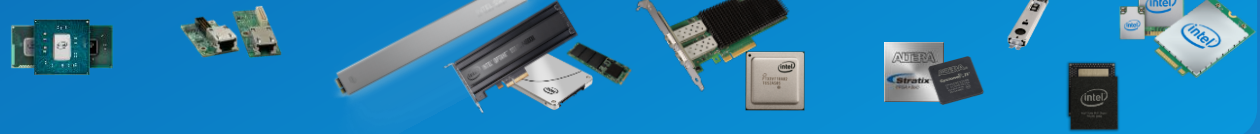
Intel® AI DevCloud

01.ORG INTEL OPEN SOURCE TECHNOLOGY CENTER

Platforms



Silicon



INTEL'S FOUNDATION:
MOORE'S LAW

"The number of transistors on a chip will **double** approximately **every two years**."

MOORE'S "LAW"

Moore's Law graph, 1965

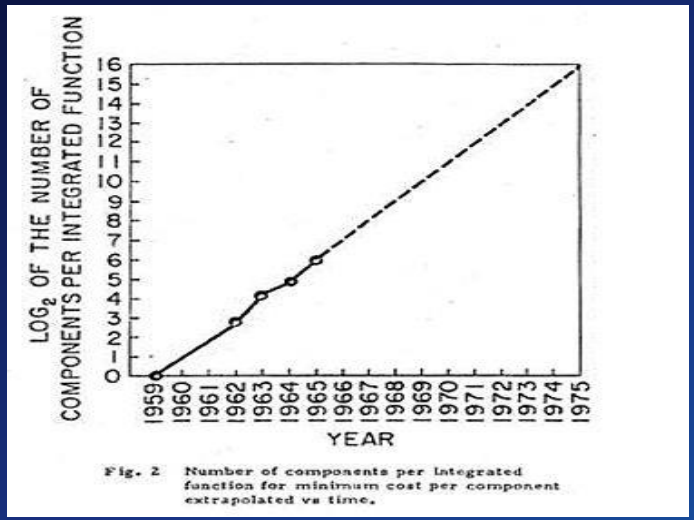
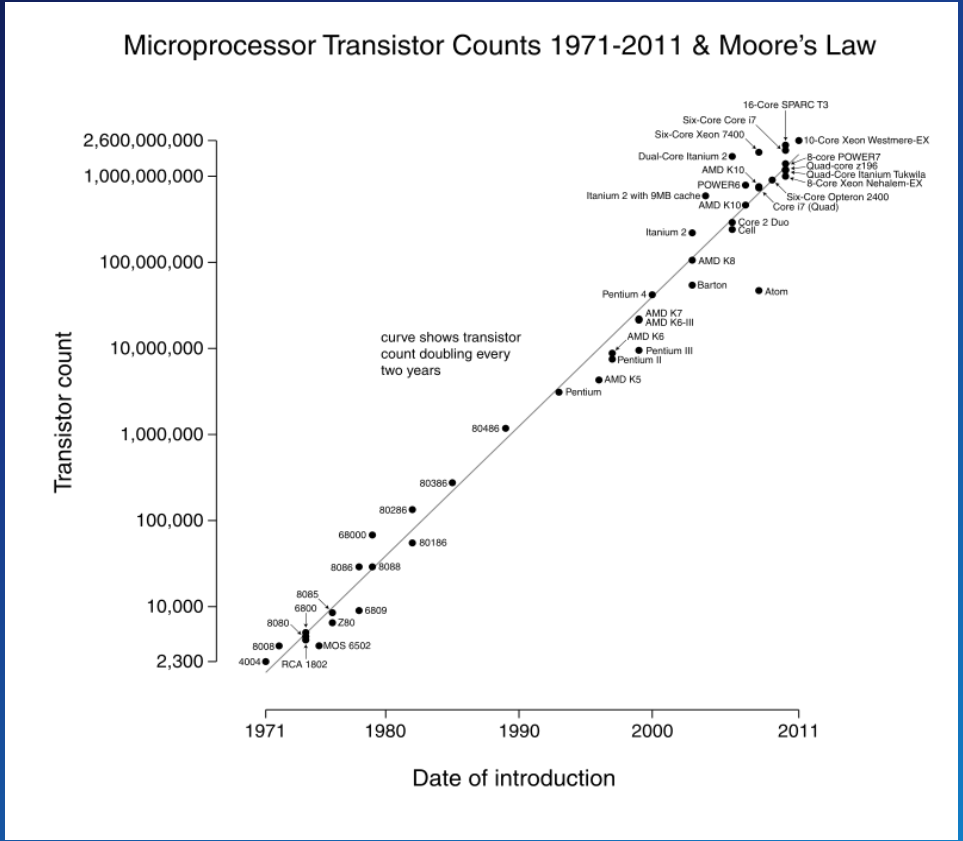
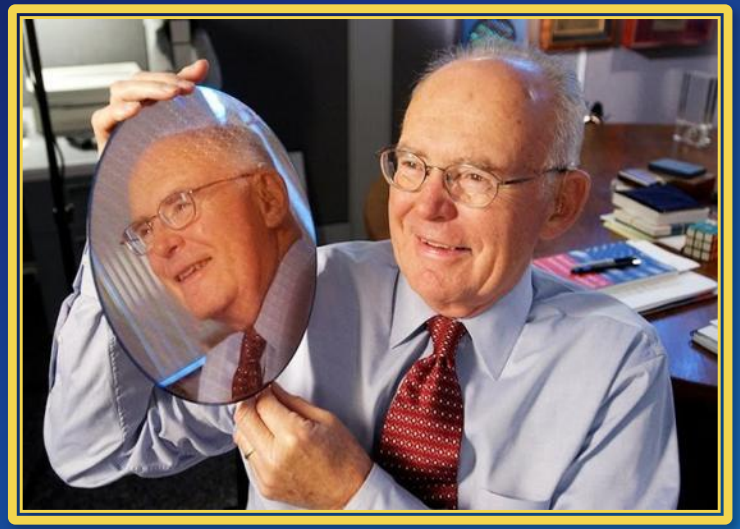


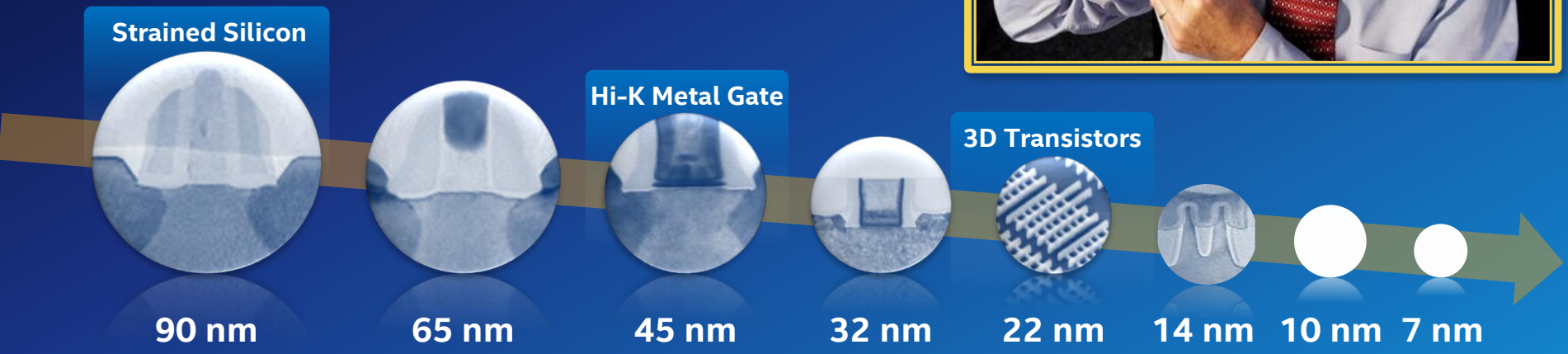
Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.



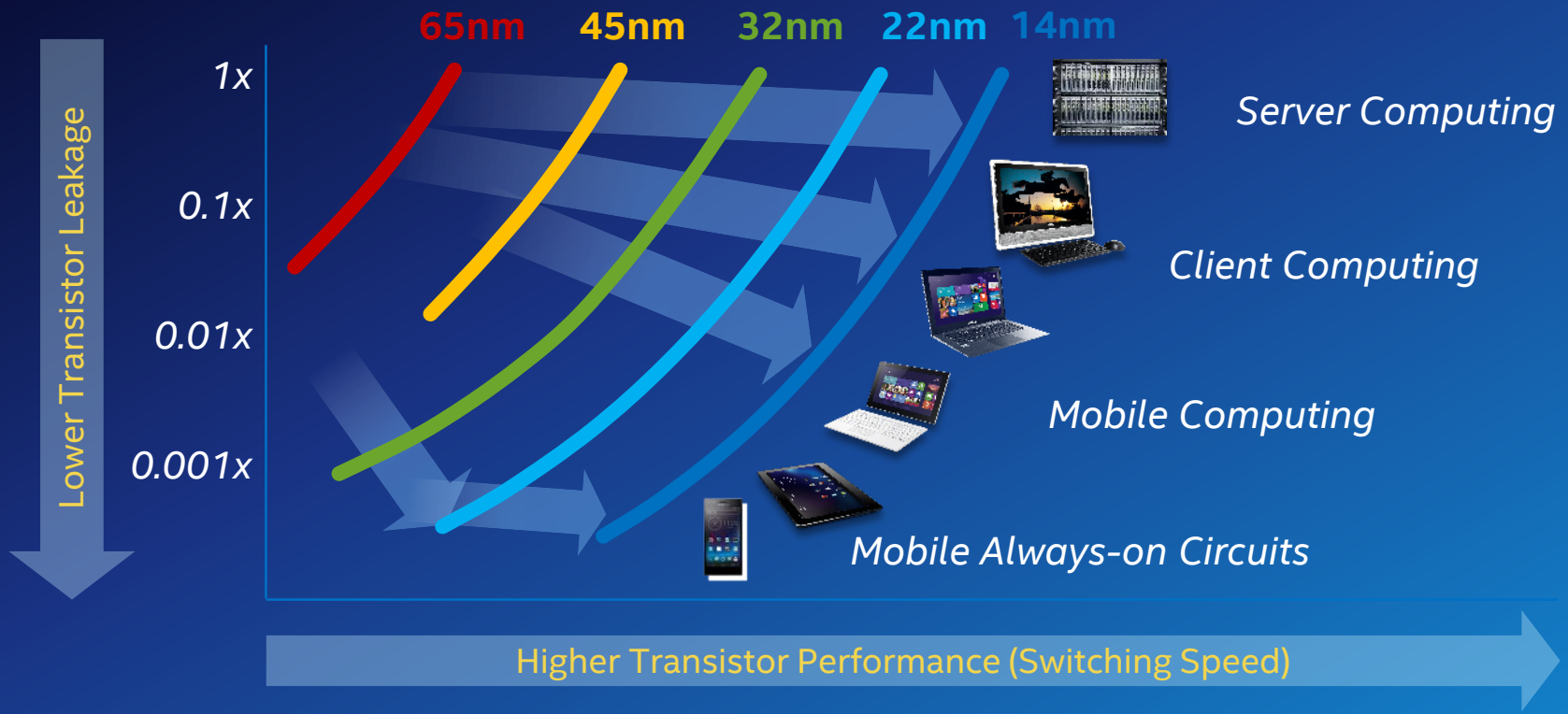
EXECUTING TO MOORE'S LAW

Enabling new devices with higher functionality and complexity while controlling power, cost, and size

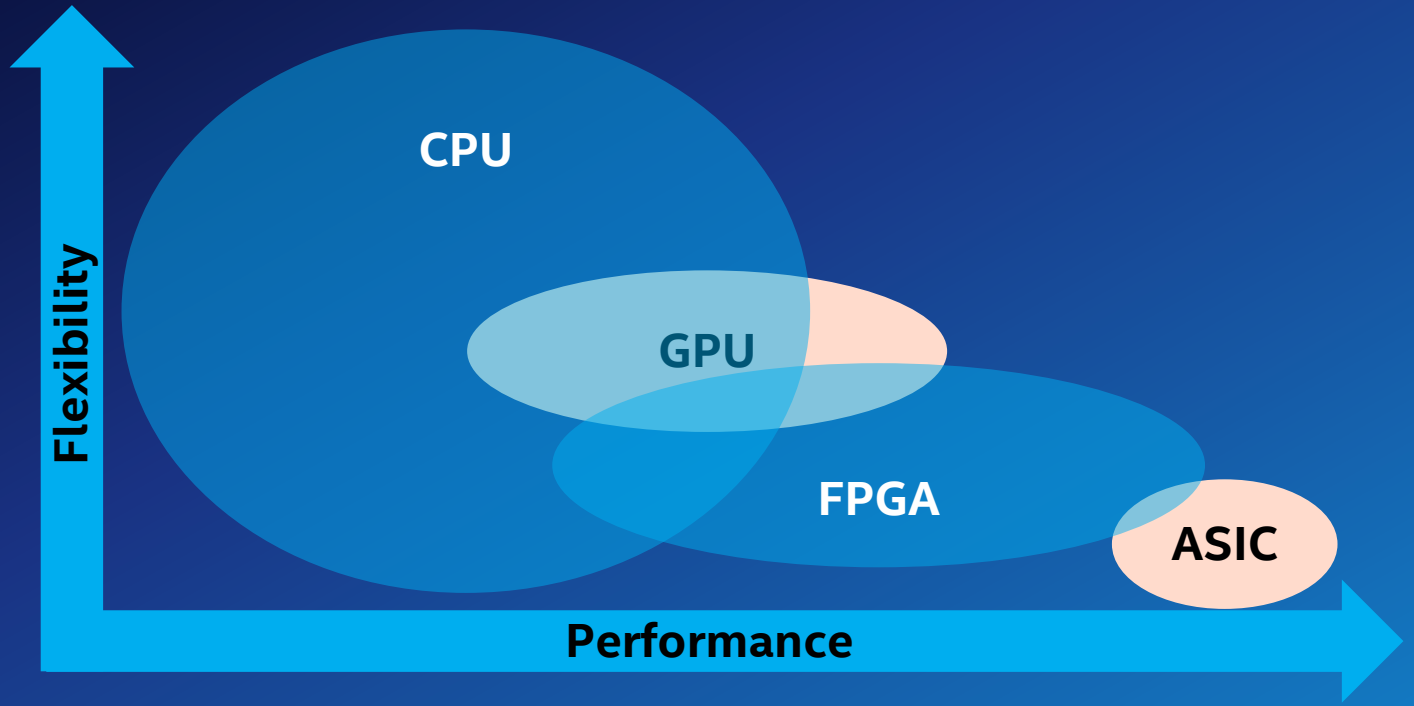




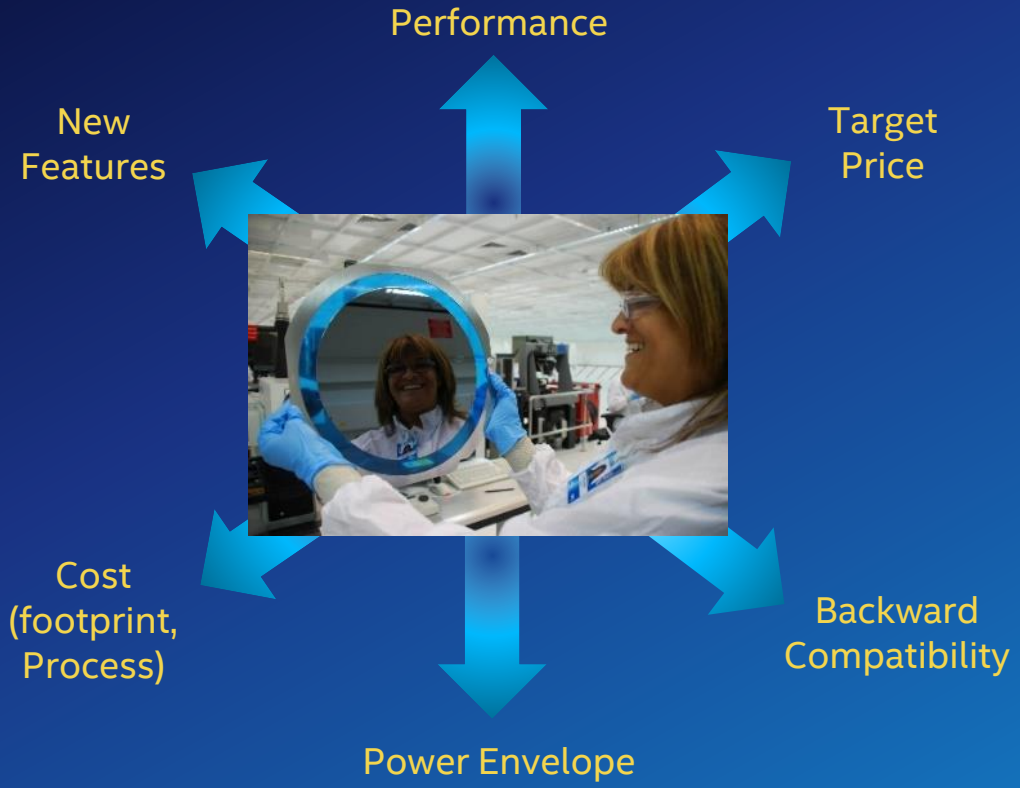
LEADING EDGE PROCESS TECHNOLOGY



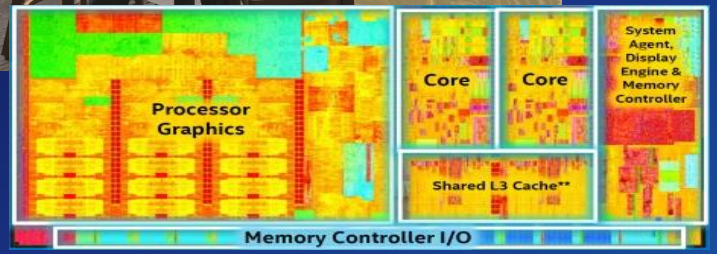
SILICON PRODUCTS

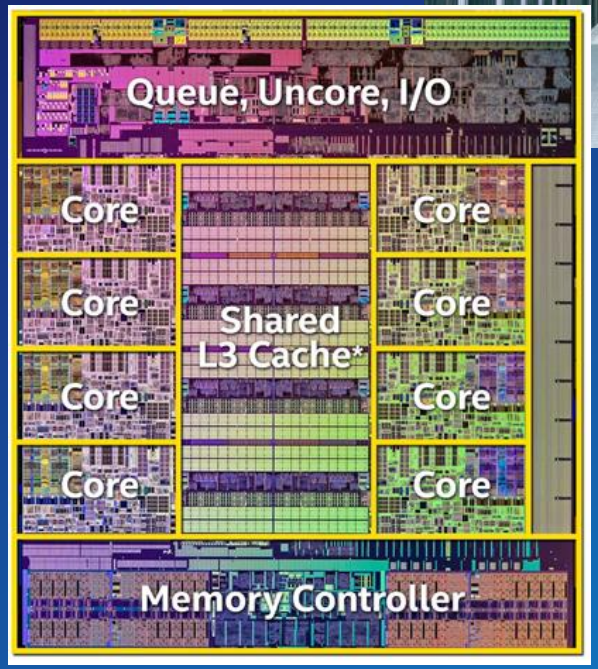


INTEGRATED CIRCUIT DESIGN – TRADEOFFS

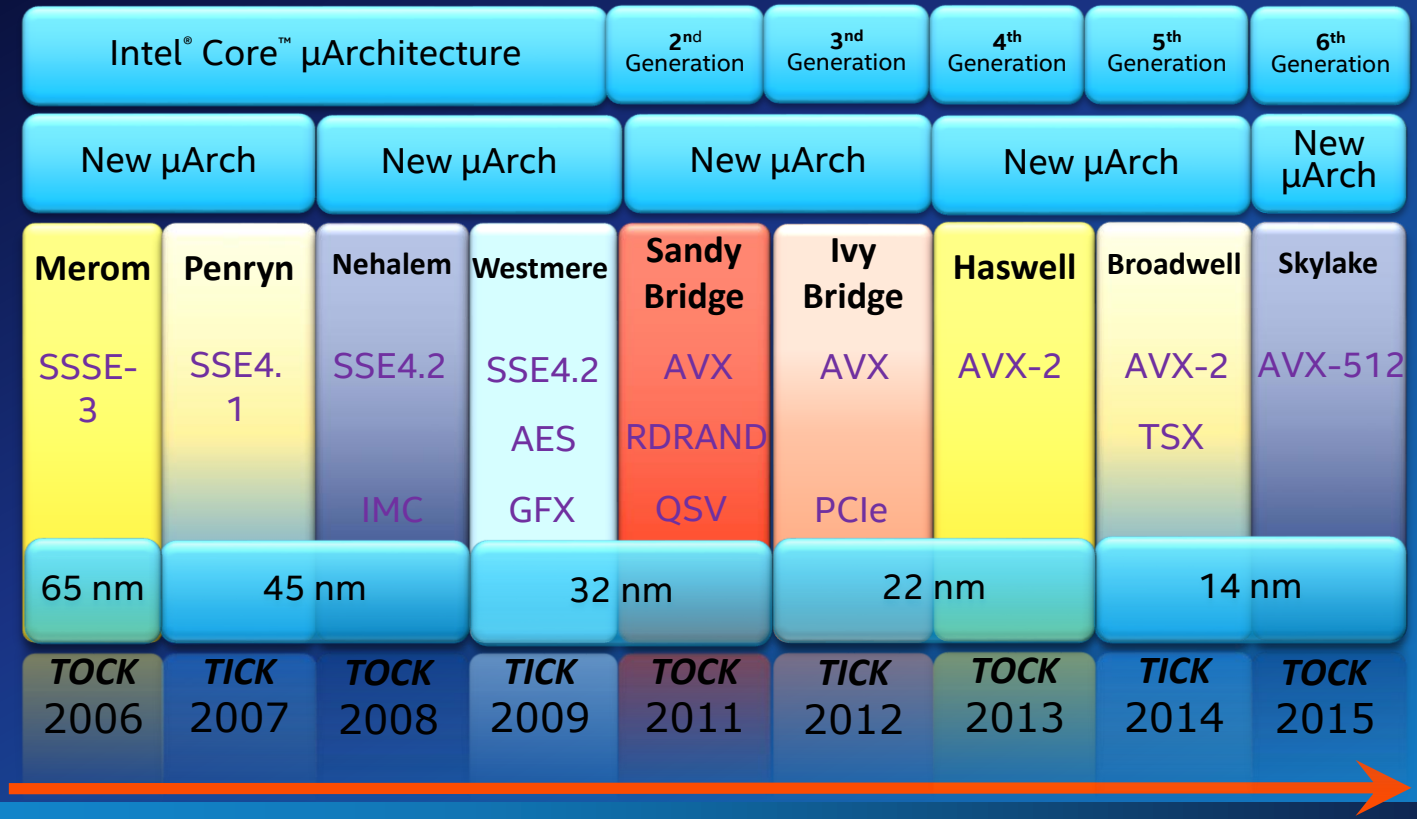


INTEL PROCESSORS





INTEL PROCESSORS

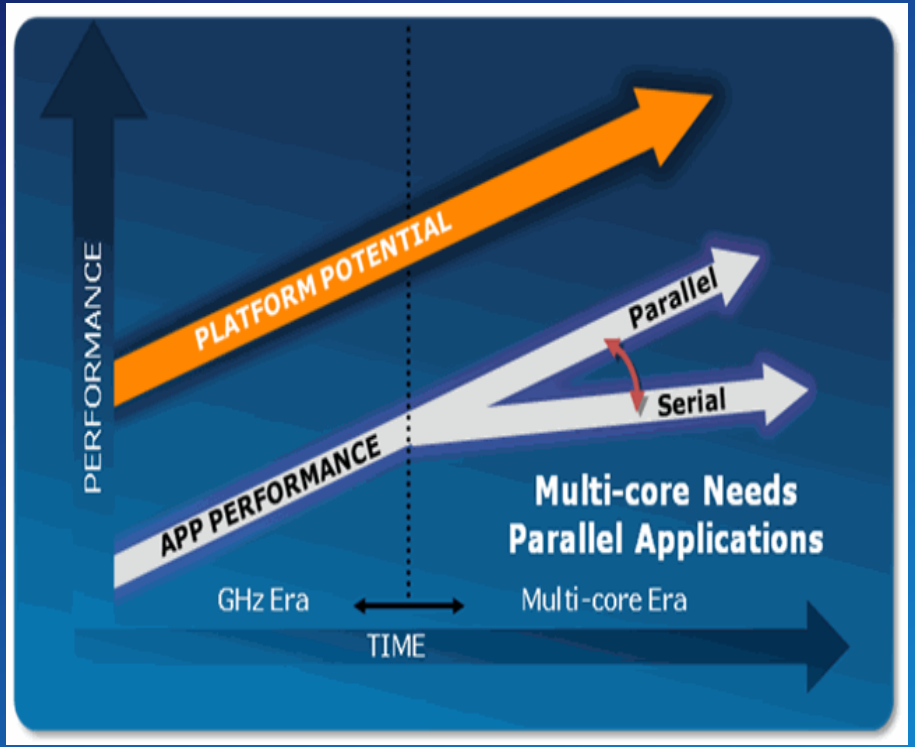


PARALLELISM

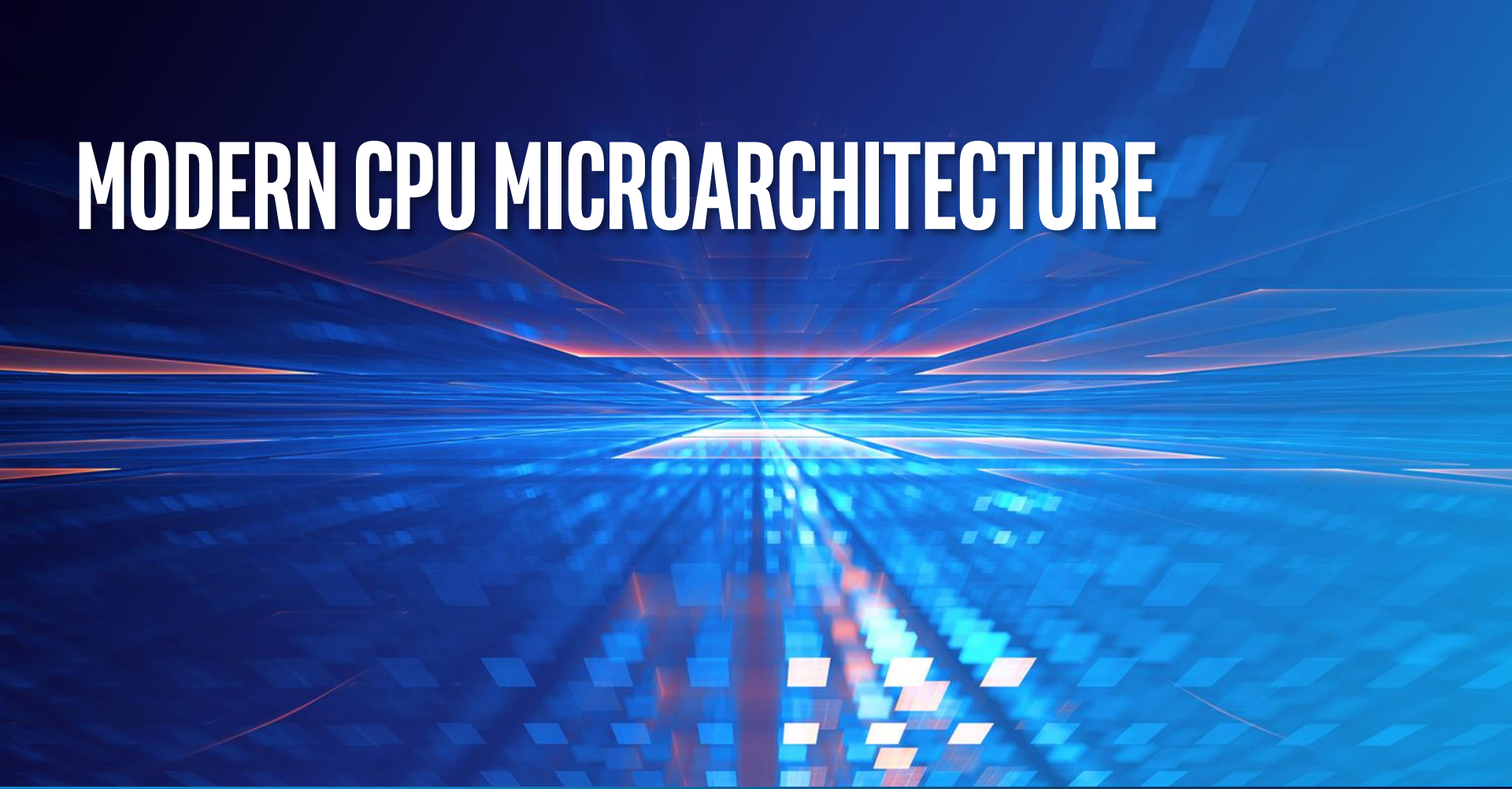
Challenge: Economical operation **frequency of (CMOS) transistors is limited**.
⇨ **No free lunch anymore!**

Solution: Parallelism:

- **Thread level** parallelism (TLP): Multicore
- **Data level** parallelism (DLP): SIMD – AVX
- **Instruction level** parallelism (ILP): Microarchitecture improvements, e.g. Out of Order execution, speculative execution …



MODERN CPU MICROARCHITECTURE



ARCHITECTURE AND MICRO-ARCHITECTURE

Architecture:

Defines the CPU: instructions, registers, data addressing modes....

⇨ **Defines the compilers**

⇨ **RISC/CISC**

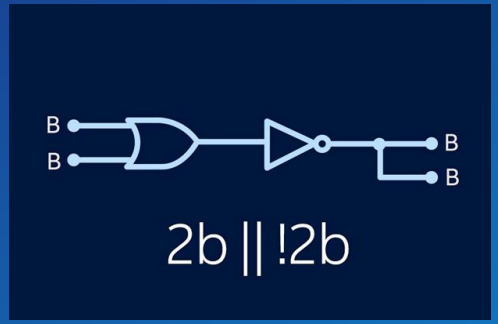
⇨ **x86 is backward compatible**

Micro-architecture:

The implementation “under the hood”: pipeline, cache, BPU, timing ....

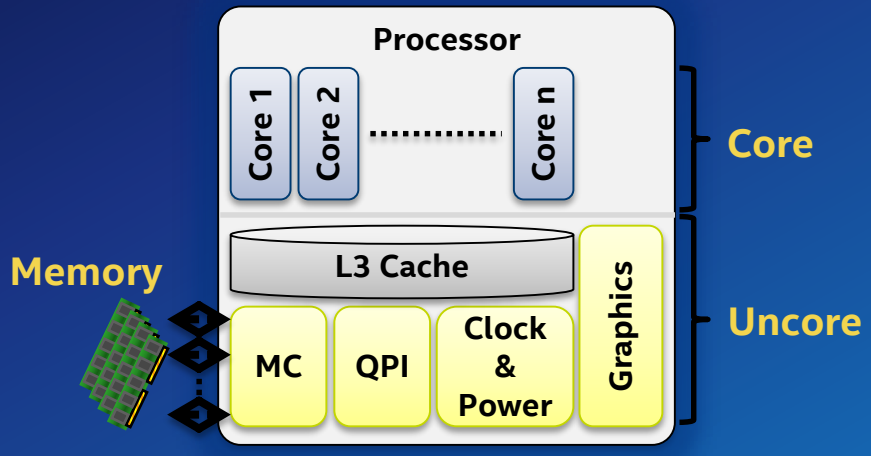
⇨ **“Not visible” to programmer**

```
1: mov $0x1, %rax
2: mov $0x2, %rbx
3: add %rbx, %rax
4: add $0x0, $r8
5: add $0x1, $r9
6: add $0x2, $r10
```



PROCESSOR ARCHITECTURE BASICS

- **Core:**
  Processor core's logic:
  - Execution units
  - Core caches (L1/L2)
  - Buffers & registers
  - FP, AVX



- **Uncore:**
  All outside a processor core:
  - Memory controller/channels (MC) and Intel® QuickPath Interconnect (QPI)
  - L3 cache shared by all cores
  - Power management and clocking
  - Optionally: Integrated graphics

⇨ **Only uncore is differentiation within same processor family!**

PROCESSOR ARCHITECTURE BASICS

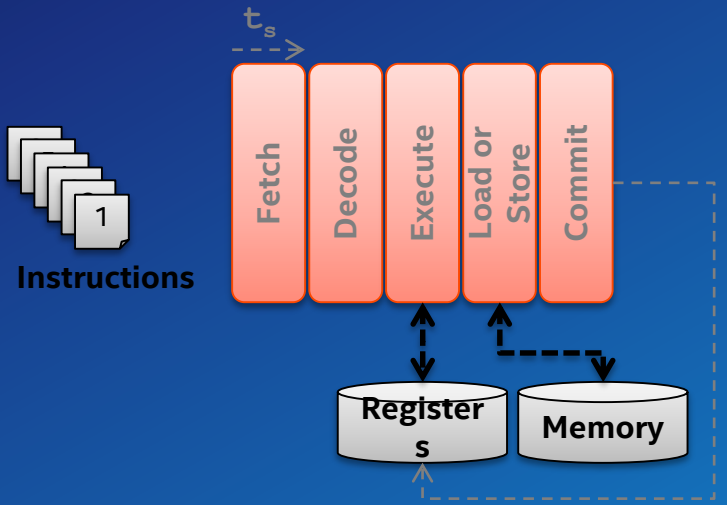
Pipeline Execution

Characteristics of pipeline execution:

- Multiple instructions for the entire pipeline (one per stage)
- Efficient because all stages kept active at every point in time
- Execution time: $n_{instructions} * t_s$

Problem:

- Reality check: What happens if t_s is not constant?

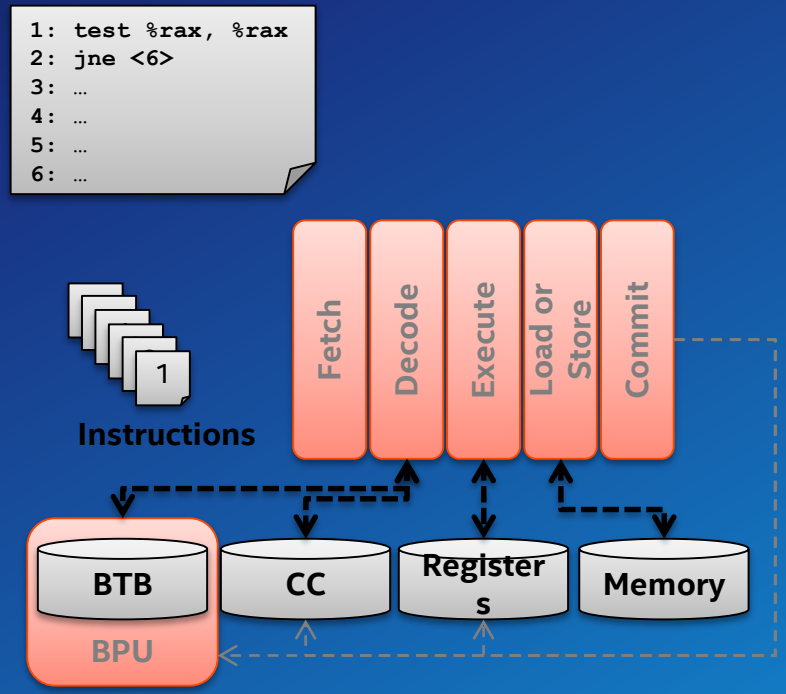


PROCESSOR ARCHITECTURE BASICS

Branch Prediction

Branch prediction reduced pipeline stalls:

- Branch prediction unit (BPU) holds logic to decide whether a branch is likely
- Branch target buffer (BTB) remembers branch targets (state of the BPU logic)
- Correct prediction: no latency
- Incorrect prediction: flush pipeline and start with other target address
- Probability of correct branch prediction can be **more than 90%**!

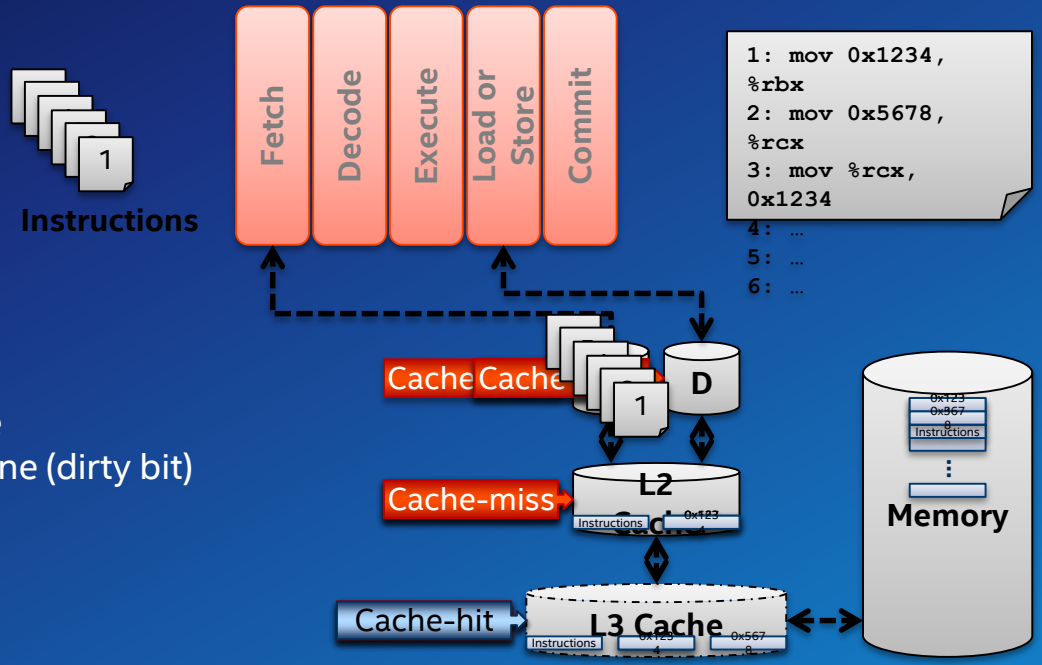


PROCESSOR ARCHITECTURE BASICS

Cache-Hierarchy & Cache-Line

Cache-hierarchy:
- For data and instructions
- Usually inclusive caches
- Races for resources
- Can improve access speed
- Cache-misses & cache-hits

Cache-line:
- Always full 64 byte block
- Minimal granularity of every load/store
- Modifications invalidate entire cache-line (dirty bit)

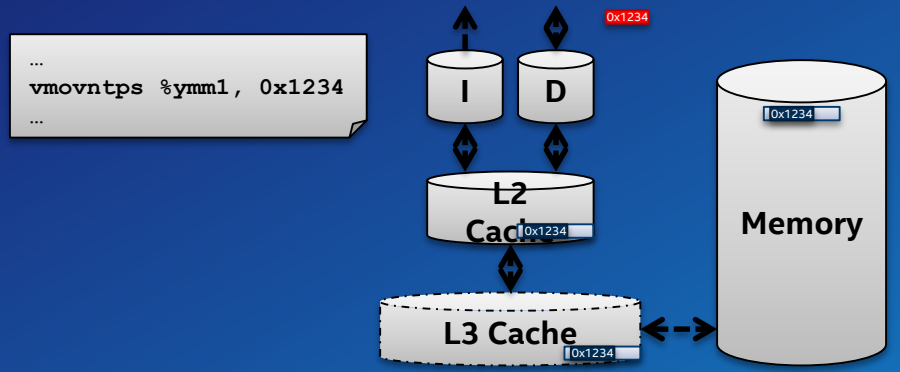


PROCESSOR ARCHITECTURE BASICS

Non-Temporal Stores

Avoids polluting the caches:

- Every data that is written to needs to be loaded into the L1 D-cache first
- For data only written to (once) there is no need to allocate cache lines
  ⇨ **Non-temporal stores** (also called **streaming stores**)
- Non-temporal stores bypass the caches
- Avoids pollution of caches: more cache-lines can be used for other loads/stores
- Cache-lines with this stored data will be evicted from caches
  ⇨ **Evicts other data on such cache-lines, too!**



PROCESSOR ARCHITECTURE BASICS

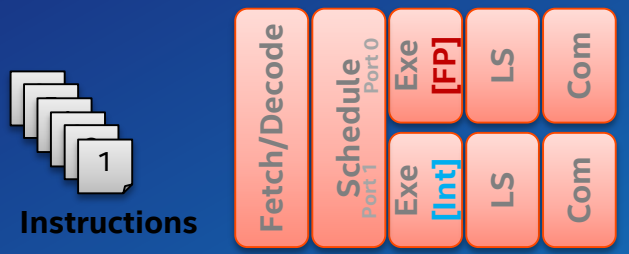
Superscalarity

Characteristics of a superscalar architecture:

- Improves throughput by covering latency (ports are independent)
- Ports can have different functionalities (floating point, integer, addressing, …)
- Requires multiple issue fetch & decode (here: 2 issue)
- Execution time: $n_{instructions} * t_{avg} / n_{ports}$

roblem:

- More complex and prone in case of dependencies
  ⇨ Solution: Out of Order Execution

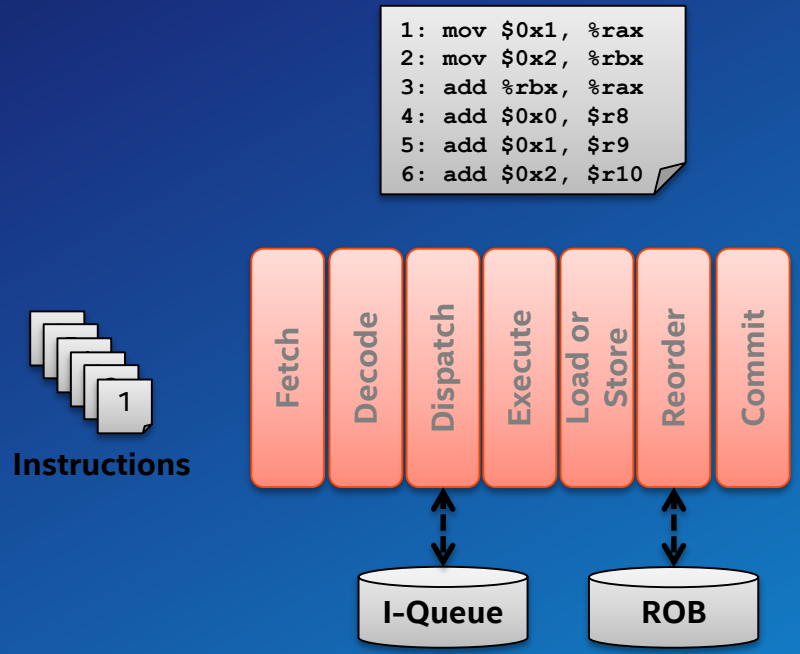


PROCESSOR ARCHITECTURE BASICS

Out of Order Execution

Characteristics of out of order (OOO) execution:

- Instruction queue (I-Queue) moves stalling instructions out of pipeline
- Reorder buffer (ROB) maintains correct order of committing instructions
- Reduces pipeline stalls, but not entirely!
- Speculative execution possible
- Opposite of OOO execution is in order execution

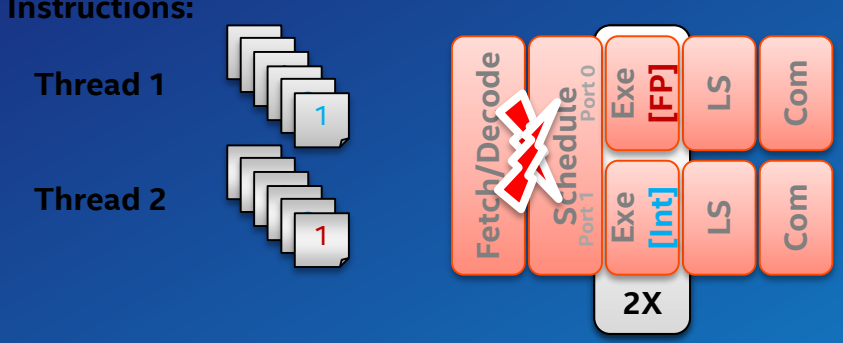


PROCESSOR ARCHITECTURE BASICS

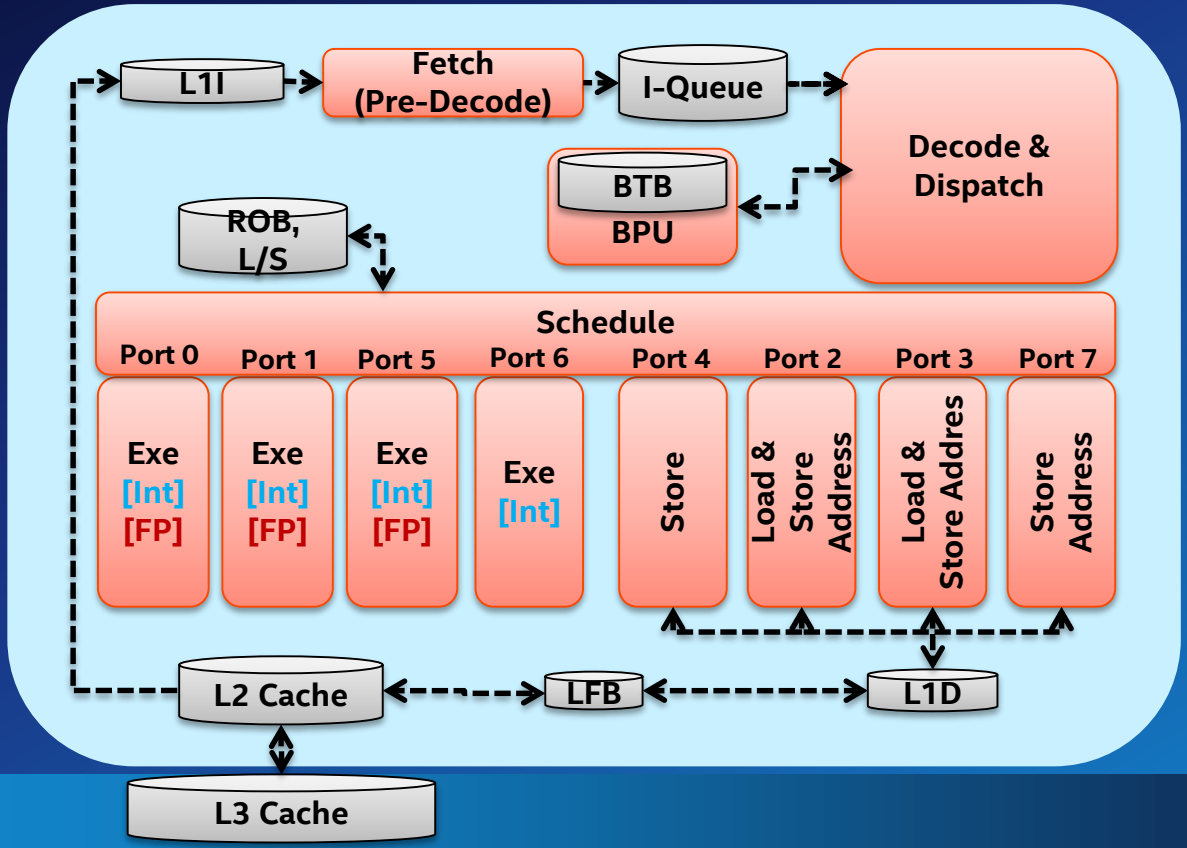
Simultaneous Multithreading Example

Simultaneous Multithreading:

- Requires at least multiple issue and superscalar architecture!
- Multiple threads (instruction streams) executed at once
- Each instruction is scheduled depending on its decoding (which port), e.g. Integer **[Int]** & floating point **[FP]** pipeline
- Multiple identical ports can exist ⇨ scheduler takes care of utilization
- Scheduler is round robin based: fair and no thread starves
- Real world also buffers multiple instructions for scheduling

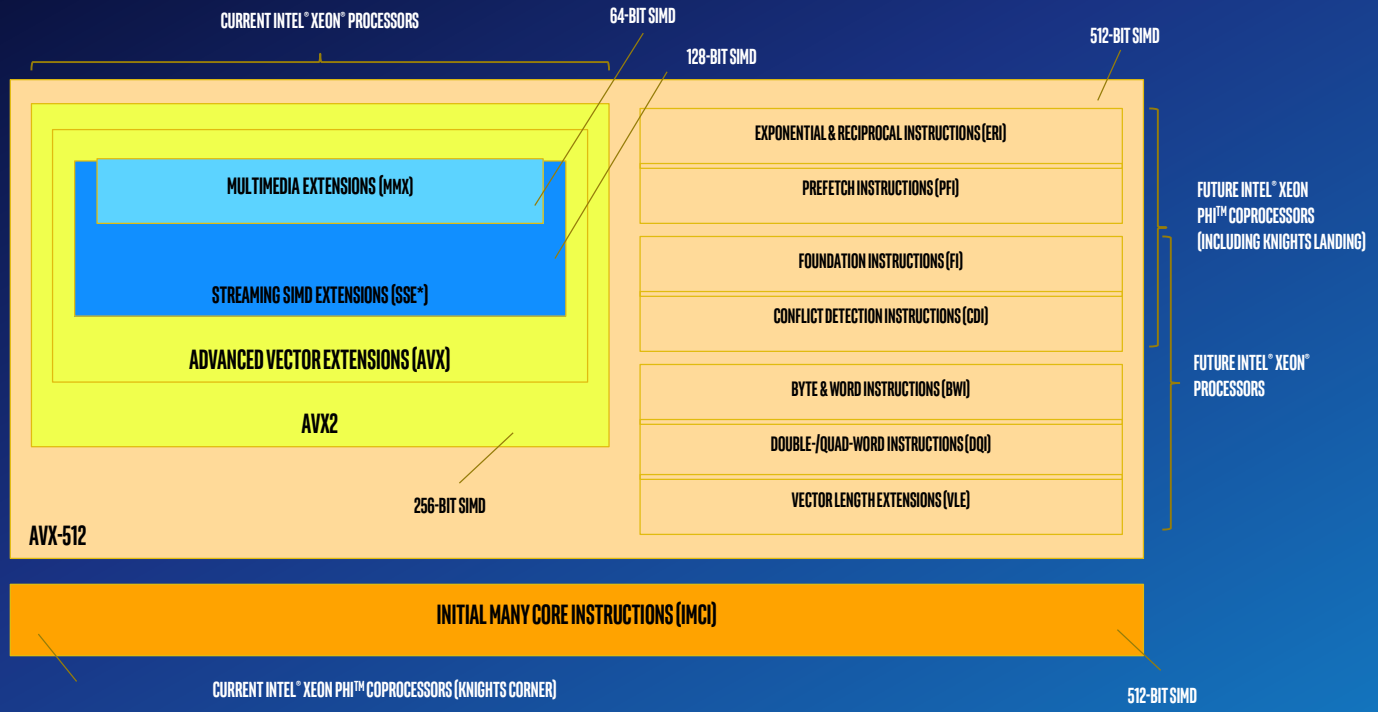


PROCESSOR ARCHITECTURE BASICS



PROCESSOR ARCHITECTURE BASICS

Data parallelism (SIMD)



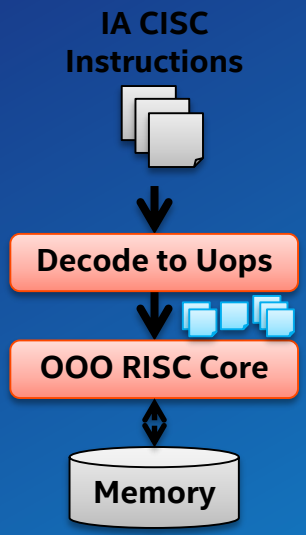
PROCESSOR ARCHITECTURE BASICS

Intel® Microarchitecture ≠ CISC

- Intel processors have a **fast out-of-order RISC core**
- **CISC instruction set** is translated micro-operations (uops)

⇨ **Efficient yet still backward compatible!**

- Micro-fusion:
  - Fuses common sequence of two uops as one uop to improve retirement throughput.
- Macro-fusion:
  - Fuses common sequence of two instructions as one decoded instruction (uop) to increase decoding throughput.
- Loop stream detector:
  - can detect small loops and creates a stream of uops



CAFÉ

eli.eliezer@intel.com

LinkedIn

PROCESSOR ARCHITECTURE BASICS

Performance Killers

Performance Killers

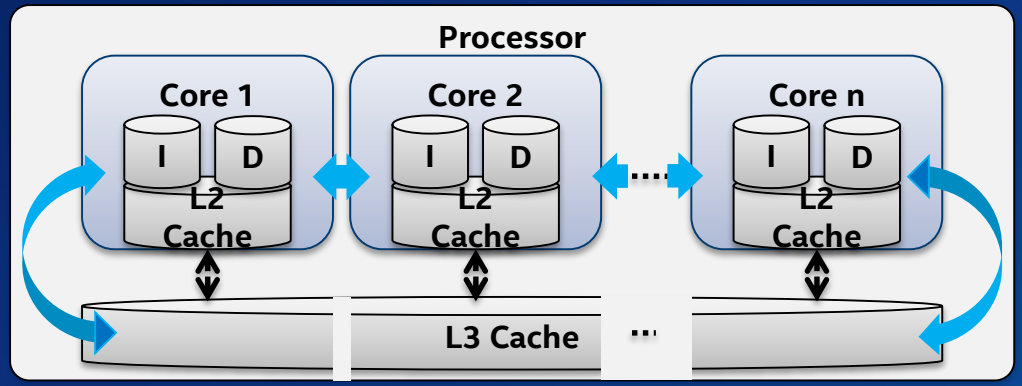
- Cache Misses
- Branch Miss-prediction
- Backend Stalls
- Address Aliasing
- Alignment
- TLB Misses

Potential Killers

- NUMA
- Simultaneous Multithreading (SMT)
- Page misses
- Power Management

PERFORMANCE KILLERS

Cache latencies

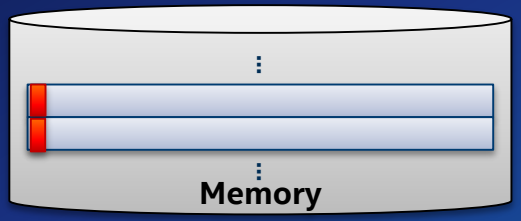


Level	Latency (cycles)	Bandwidth (per core per cycle)	Size
L1-D	4	2x 16 bytes	32KiB
L2 (unified)	12	1x 32 bytes	256KiB
L3 (LLC)	26-31	1x 32 bytes	varies (≥ 2MiB per core)
L2 and L1 D-Cache in other cores	43 (clean hit), 60 (dirty hit)		

PERFORMANCE KILLERS

Cache Misses

```
for (int j = 0; j < 64; j++)
     for (int i = 0; i < 64; i++)
          Item[i][j] = (Item[i][j] - MinValue) * Factor;
```



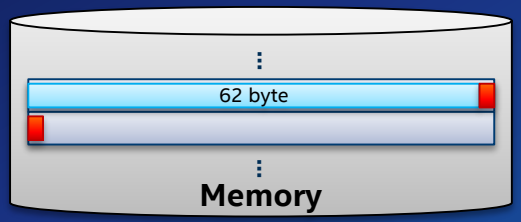
Item[0][0]
Item[1][0]

```
for (int i = 0; i < 64; i++)
     for (int j = 0; j < 64; j++)
          Item[i][j] = (Item[i][j] - MinValue) * Factor;
```

PROCESSOR ARCHITECTURE BASICS

Alignment

```
#pragma pack(2)
struct {
… // 62 byte
int x; // 4 byte
} data;
```



Cache-line n
Cache-line n+1

```
#pragma pack(2)
struct {
… // 62 byte
… // 2 byte
int x; // 4 byte
} data;
```

DATA LAYOUT AOS VS SOA

ARRAY OF STRUCTURES VS STRUCTURE OF ARRAYS

```
// Array of Structures (AoS)
struct coordinate {
    float x, y, z;
} crd[N];
    …
for (int i = 0; i < N; i++)
    … = … f(crd[i].x, crd[i],y, crd[i].z);
```


```
// Structure of Arrays (SoA)
struct coordinate {
    float x[N], y[N], z[N];
} crd;
    …
for (int i = 0; i < N; i++)
    … = … f(crd.x[i], crd.y[i], crd.z[i]);
```

PERFORMANCE KILLERS

Branches

```
for (j = 0; j < FIELDS; j =+2)
    do something …

for (j = 1; j < FIELDS; j =+2)
    do something odd …
```

```
for (j = 0; j < FIELDS; J++)
     if (Field == Even) //if (j mod 2)
     {
          do something …
          Field = Odd;
     }
     else
     {
          do something odd …
          Field = Even;
     }
```

PERFORMANCE KILLERS

Branches

```
BlockPointer = malloc(BlockSize);
if (BlockPointer)
{
     do something …
     free(BlockPointer);
}
else
{
     handle exception …
     break;
}
```

```
BlockPointer = malloc(BlockSize);
if (BlockPointer == NULL)
{
     handle exception …
     break;
}
else
{
     do something …
     free(BlockPointer);
}
```

AUTO VECTORIZATION

not all loops will vectorize

Data dependencies between iterations

- Proven Read-after-Write data (i.e., loop carried) dependencies
- Assumed data dependencies
  - Aggressive optimizations (e.g., IPO) might help

RAW DEPENDENCY

```
for (int i = 0; i < N; i++)
    a[i] = a[i-1] + b[i];
```

Vectorization won't be efficient

- Compiler estimates how better the vectorized version will be
- Affected by data alignment, data layout, etc.

INEFFICIENT VECTORIZATION

```
for (int i = 0; i < N; i++)
    a[c[i]] = b[d[i]];
```

Unsupported loop structure

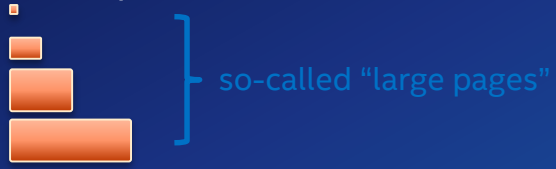
- While-loop, for-loop with unknown number of iterations
- Complex loops, unsupported data types, etc.
- (Some) function calls within loop bodies
  - Not the case for SVML functions

FUNCTION CALL WITHIN LOOP BODY

```
for (int i = 0; i < N; i++)
    a[i] = foo(b[i]);
```

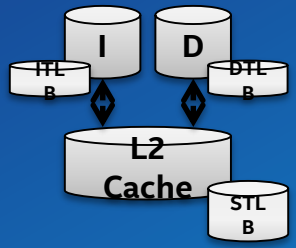
PAGING AND TLB

- “page” maps linear address range to physical memory
- Page size is constant across entire memory
- Different page sizes possible (depends on HW):
  - 4 kB
  - 64 kB (non IA)
  - 2 or 4 MB
  - 1 GB



Transaction Lookaside Buffer (TLB):

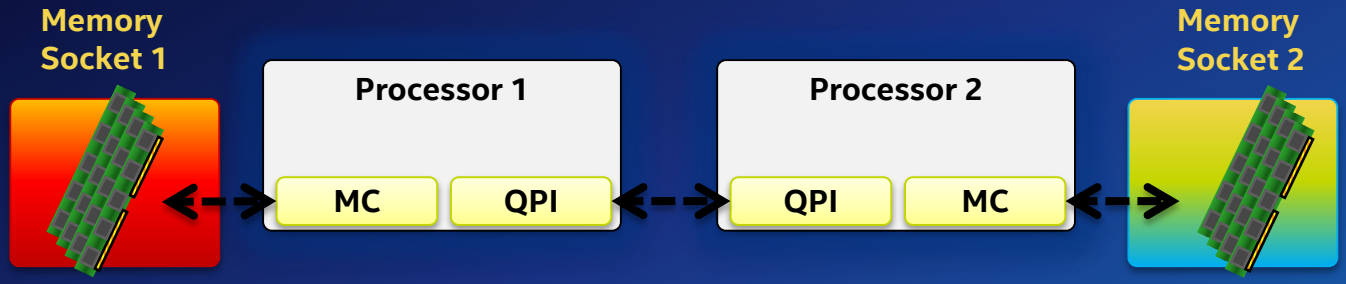
- Translates physical to linear addresses
- Related to cache hierarchy:
  - L1 Instruction TLB (ITLB)
  - L1 Data TLB (DTLB)
  - Unified TLB for L2 (STLB)
- Limited amount of entries decreasing with growing page sizes



- Any page that is not in the TLB causes a penalty due to calculation (~7 cycles)

PROCESSOR ARCHITECTURE BASICS

UMA and NUMA Explained



- **UMA (Uniform Memory Access, non-NUMA):**
  - Addresses interleaved across memory nodes **by cache line**
  - Accesses may or may not have to cross QPI link
    - ⇨ **Memory access is distributed evenly between Processors**

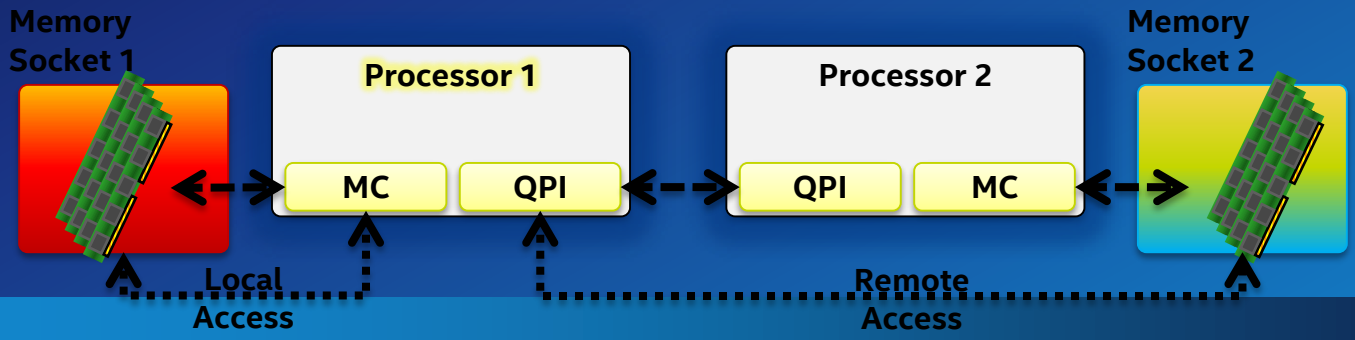
- **NUMA (Non Uniform Memory Access):**
  - Addresses not interleaved across memory nodes by cache line
  - Each processor has direct access to contiguous block of memory
    - ⇨ **Provides peek performance but requires special handling**

PROCESSOR ARCHITECTURE BASICS

Enabling NUMA

- Improve accesses to local memory vs. remote memory
  - Remote memory access **latency** ~1.7x greater than local memory
  - Local memory **bandwidth** can be up to ~2x greater than remote

- Ensure OS, middleware, VM .. All NUMA aware
- Ensure 3rd party components support affinity mapping
- Thread affinity is required
- Explicit allocation with NUMA aware libraries, e.g. `libnuma` (Linux*)
- Bind **memory ⇔ (SW) thread**, and **(SW) thread ⇔ processor**



CLOCK RATE AND POWER GATING

Control power utilization and performance:

- Clock rate:
  - Idle components can be clocked lower to save energy:
    ⇨ **Intel SpeedStep®**
  - If thermal specification allows, components can be over-clocked to gain performance :
    ⇨ **Intel® Turbo Boost Technology**
- Power gating: Turn off components to save power
- P-state (Processor State): lower latency and combined with Intel SpeedStep®
- C-state (Chip State): longer latency, more aggressive static power reduction

BEFORE YOU START PERFORMANCE ANALYSIS:

- Expectations: Set reasonable goals\targets
  - CPU can issue up to 4 instruction per cycle
    - Multiply by clock speed will give a ball park figure (per second)
  - Application:
    - Get estimation of the amount of instruction
      - Not all instructions born equal …
    - Set target accordingly
  - The 80\20% rules usually apply:
    - 80% of the gain will take 20% of the time

BEFORE YOU START PERFORMANCE ANALYSIS:

- Verify the system is in good shape

  - Hardware Configuration
    - CPU – the right SKU (frequency, cache, QPI)
    - Disks (SATA 3, no failures, RAID …)
    - Memory (amount, type, speed, all channels …)

  - BIOS configuration
    - Turn off Hyper Threading
    - Turn off “autonomous” HW
      - Turbo Boost
      - Power Management
      - Standby States
    - NUMA

BEFORE YOU START PERFORMANCE ANALYSIS:

- Verify the software stack is right:

  - System
    - OS (version, patches, 32\64 bits …)
    - Drivers
    - Middleware ….

  - Software
    - No bugs … passed all functional testing
    - Compiler (version , 32\64 bits, optimization)
    - 3rd party's software: libraries …


intel
INTEL® SOFTWARE PERFORMANCE
USING THE RIGHT TOOLS

INTEL® SOFTWARE DEVELOPMENT PRODUCTS

Technical Computing

Improve application performance, scalability and reliability

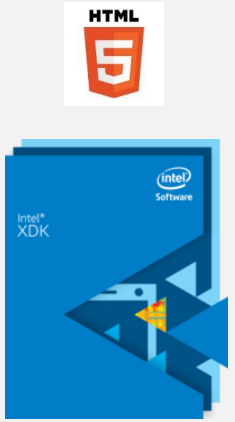
Performance Client

Video streaming performance and Immersive interactivity for multimedia apps

System

Create fast, efficient embedded & mobile devices/systems

Web App



Deploy apps on multiple platforms using one codebase

INTEL PARALLEL STUDIO XE SUITES

Naming and Configuration

Intel® Parallel Studio XE 2016 Composer Edition 	Intel® Parallel Studio XE 2016 Professional Edition 	Intel® Parallel Studio XE 2016 Cluster Edition 
Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*
	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE Intel® MPI Library Intel® Trace Analyzer and Collector
Bundle or Add-on: Rogue Wave IMSL* Library	Add-on: Rogue Wave IMSL* Library	Add-on: Rogue Wave IMSL* Library

Additional configurations including, floating and academic, are available at: http://intel.ly/perf-tools

INTEL COMPILER

Always supports Intel's latest processors

- Usually takes Tick or Tock for the other compilers

Auto Dispatcher

- Create one binary with several code path's optimized to all\selected Intel's micro-architectures
- Runs the appropriate path after recognizing the underlying processor

Cilk++ – Auto Thread Parallelism

- cilk_spawn
- cilk_sync
- cilk_for -

Array Notation – Auto Thread Parallelism (Data Vectorization)

- Supporting the evolution of industry standards of ***OpenMP, MPI, TBB, Fortran and C++*** Intel® Compilers & performance libraries
- Latest tool and operating system environments
  further enhanced AVX-512 support, Skylake optimizations, Knights Landing support, Windows 10 support, ...

INTEL CILK PLUS

C++ Parallelism: Taking Advantage of Intel® Cilk™ Plus Keywords

Before: Serial Code Fragment

```
void Search( const Board& b ) {
  ...
    } else {
     Search( Board(b,i,0) );
     Search( Board(b,i,1) );
    }}}
```

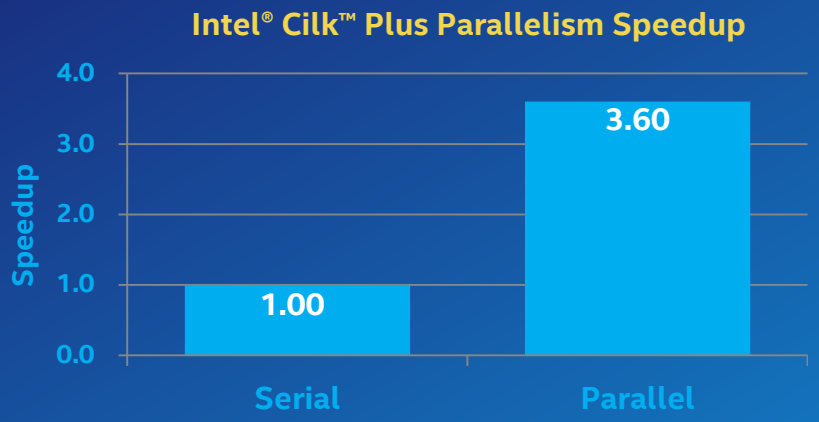
30.2 Seconds

After: Parallel Code Fragment, fork/join task parallel

```
void Search( const Board& b ) {
  ...
    } else {
     cilk_spawn Search( Board(b,i,0)
);
     Search( Board(b,i,1) );
    }}}
```

8.4 Seconds

Real example code produced the results on the right



INTEL SOFTWARE ANALYSIS TOOLS

Intel® VTune™ Amplifier XE

Performance Profiler

Intel® Inspector XE

Memory & Thread Debugger

Intel® Advisor XE

Threading Design & Prototyping

TUNE APPLICATIONS FOR SCALABLE MULTICORE PERFORMANCE

Intel® VTune™ Amplifier XE Performance Profiler

Get the Data You Need

- Hotspot (Statistical call tree), Call counts (Statistical)
- Thread Profiling – Concurrency and Lock & Waits Analysis
- Cache miss, Bandwidth analysis[1]
- GPU Offload and OpenCL* Kernel Tracing on Windows

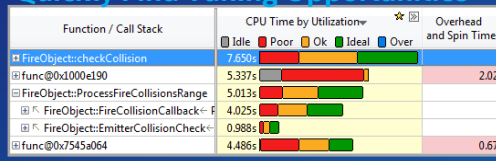
Find Answers Fast

- View Results on the Source / Assembly
- OpenMP Scalability Analysis, Graphical Frame Analysis
- Filter Out Extraneous Data - Organize Data with Viewpoints
- Visualize Thread & Task Activity on the Timeline

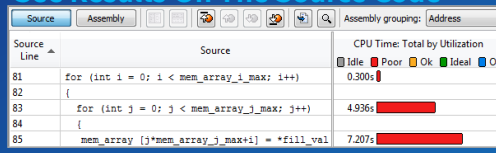
Easy to Use

- No Special Compiles – C, C++, C#, Fortran, Java, ASM
- Visual Studio* Integration or Stand Alone on Windows* or Linux*
- Graphical Interface & Command Line
- Local & Remote Data Collection
- New! Analyze Windows* & Linux* data from OS X*[2]

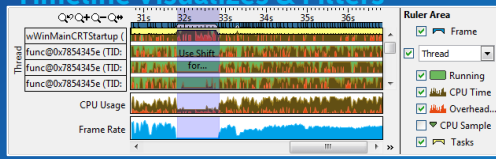
Quickly Find Tuning Opportunities



See Results On The Source Code



Timeline Visualizes & Filters



[1] Events vary by processor. [2] No data collection on OS X*

FIND & DEBUG MEMORY & THREADING ERRORS

Intel® Inspector XE – Memory & Thread Debugger

Correctness Tools Increase ROI By 12%-21%1

- Errors found earlier are less expensive to fix
- Several studies, ROI% varies, but earlier is cheaper

Diagnosing Some Errors Can Take Months

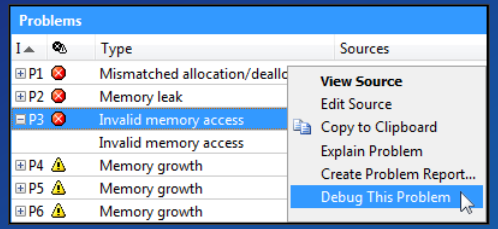
- Races & deadlocks not easily reproduced
- Memory errors can be hard to find without a tool

Debugger Integration Speeds Diagnosis

- Breakpoint set just before the problem
- Examine variables & threads with the debugger

[1] **Cost Factors – Square Project Analysis**
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project Results

Debugger Breakpoints



Part of Intel® Parallel Studio
For Windows* and Linux* From $1,599

Intel® Inspector XE dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.

Peter von Kaenel, Director, Software Development, ***Harmonic Inc.***

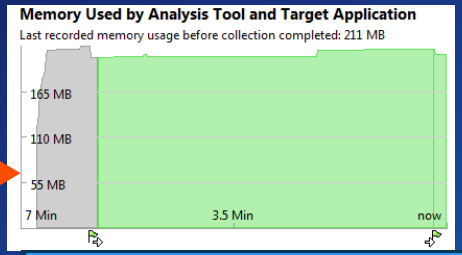
http://intel.ly/inspector-xe

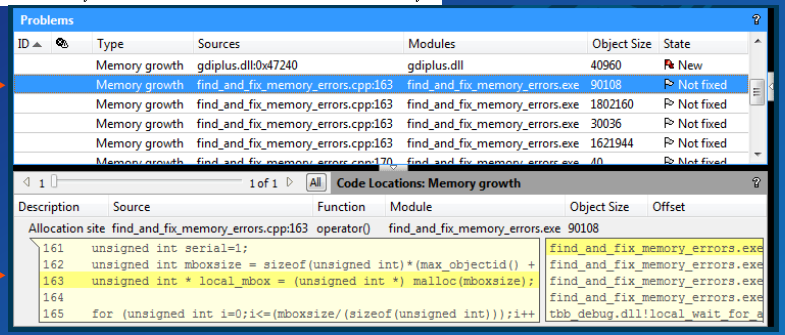
INCREMENTALLY DIAGNOSE MEMORY GROWTH

Intel® Inspector XE

As your app is running…

- Memory usage graph plots memory growth
- Select a cause of memory growth
- See the code snippet & call stack





DATA DRIVEN THREADING DESIGN

Intel® Advisor XE – Thread Prototyping for Software Architects

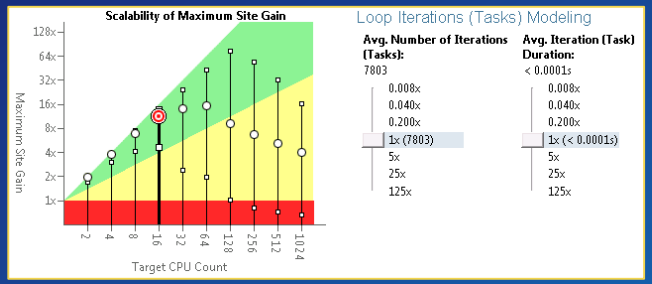
Have you:

- Tried threading an app, but seen little performance benefit?
- Hit a "scalability barrier"? Performance gains level off as you add cores?
- Delayed a release that adds threading because of synchronization errors?

Breakthrough for threading design:

- Quickly prototype multiple options
- Project scaling on larger systems
- Find synchronization errors before implementing threading
- Separate design and implementation -Design without disrupting development

Add Parallelism with Less Effort, Less Risk and More Impact



Part of Intel® Parallel Studio XE
For Windows* and Linux* From $1,599

"**Intel® Advisor XE** has allowed us to quickly prototype ideas for parallelism, saving developer time and effort"

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

http://intel.ly/advisor-xe

DESIGN THEN IMPLEMENT

Intel® Advisor XE Thread Prototyping

Design Parallelism

- No disruption to regular development
- All test cases continue to work
- Tune and debug the design before you implement it

1) Analyze it.

2) Design it.
(Compiler ignores these annotations.)

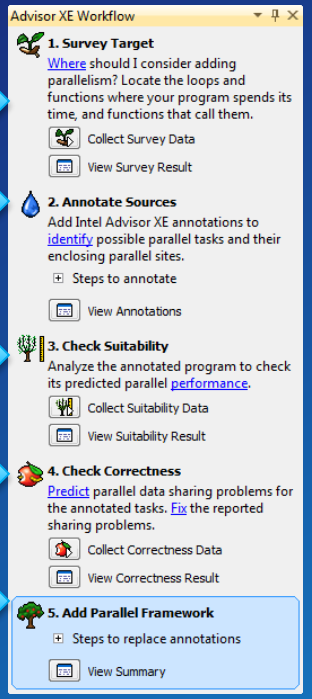
3) Tune it.

4) Check it.

Implement Parallelism

5) Do it!

Less Effort, Less Risk, More Impact




LIBRARIES

INTEL® THREADING BUILDING BLOCKS

What

- Widely used C++ template library for task parallelism.
- Features
- Parallel algorithms and data structures.
- Threads and synchronization primitives.
- Scalable memory allocation and task scheduling.

Also available as open source at threadingbuildingblocks.org

https://software.intel.com/intel-tbb

Benefit

- Rich feature set for general purpose parallelism.
- Available as an open source and a commercial license.
- Supports C++, Windows*, Linux*, OS X*, other Os's.
- Commercial support for Intel® Atom™, Core™, Xeon® processors, and for Intel® Xeon Phi™ coprocessors

INTEL® IPP PROGRAMMING DOMAINS

Signal Processing	Image Processing	Computer Vision	String Processing	Data Compression	Cryptography*
One dimensional input data processing	2D input data processing including color conversion support	Includes optimizations that accelerate common OpenCV functions	String manipulation and regular expression functionality	Huffman, VLC and Dictionary compression techniques	Support for standard cryptographic algorithms

Domains previously marked for deprecation are no longer in the default IPP package. Please see the IPP documentation for additional details.

* Cryptography domain may not be available in all geographies

OPTIMIZED MATHEMATICAL BUILDING BLOCKS

Intel® Math Kernel Library

Linear Algebra

- BLAS
- LAPACK
- Sparse Solvers

Fast Fourier Transforms

- Multidimensional
- FFTW interfaces
- Cluster FFT

Vector Math

- Trigonometric
- Hyperbolic
- Exponential, Log
- Power / Root

Vector RNGs

- Congruential
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

Summary Statistics

- Kurtosis
- Variation coefficient
- Order statistics
- Min/max
- Variance-covariance

And More

- Splines
- Interpolation
- Trust Region
- Fast Poisson Solver

INTEL® MPI LIBRARY

What	• Intel's High Performance MPI Library
Why	• Scale Performance – Tuned for Latest Intel Architectures • Scale Forward – Multicore and Manycore Ready • Scale Efficiently – Flexible Fabric Selection & Compatibility
How	• Standards Based – Built on Open Source MPICH Implementation • Sustained Scalability – Tuning for Low Latencies, High Bandwidth & Increased Processes • Multi Fabric Support – Supports Popular High Performance Networking Fabrics

PARALLEL PROGRAMMING BOOKS

Xeon and Xeon Phi Code Modernization, Vertical Case Studies, Tips and Tricks, Parallel programming concepts/methods

USEFUL LINKS

Register for Webinar series:

https://software.intel.com/en-us/articles/intel-software-tools-technical-webinar-series

Intel Software Tools Home page:

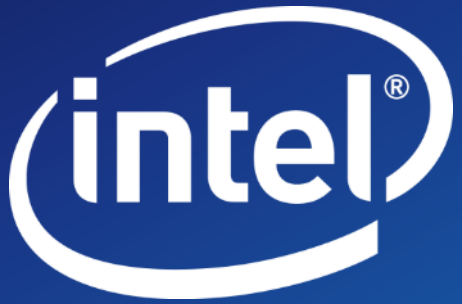
www.intel.com/software/products

Experimental Intel Software Technologies:

whatif.intel.com

Intel Registration Centre (IRC):

https://registrationcenter.intel.com/


intel®