IDF2011
INTEL DEVELOPER FORUM

Graphics Software Stack Evolution

David Blythe, Chief Graphics Software Architect, Intel

Introduction – Sunil Shenoy, Intel Corporate Vice President
General Manager of Visual Parallel Computing Group

SPCS006

Sponsors of Tomorrow. intel

IDF2011
INTEL DEVELOPER FORUM

Sunil Shenoy

Intel Corporate Vice President
General Manager of Visual Parallel Computing Group

Sponsors of Tomorrow.

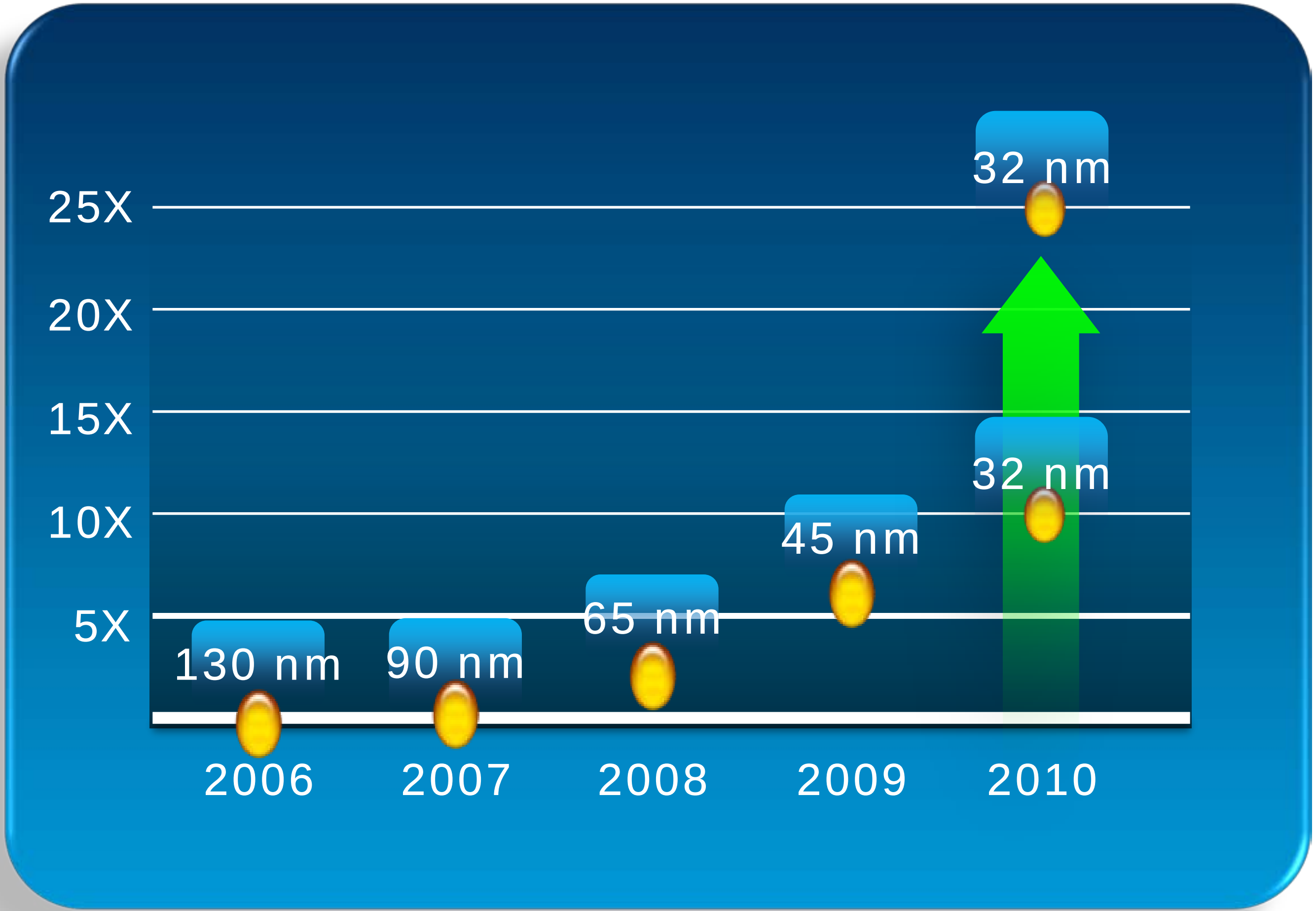
Building a Continuum with Intel® Architecture and Processor Graphics

Desktops Laptops Netbooks Tablets Smartphones Smart TVs Embedded

Its Not Just 3D Games

- Office & Productivity Apps
- Mainstream Games
- High End Games
- 3D Content Creation
- Perceptual Computing
- Multi High Res Display
- Imaging Apps
- High End Video Apps
- Mainstream Video Apps
- Web Video
- Web Games
- Browser Apps & Content (HTML)
- 3D User Interface

Looking at the Last 5 years…



25X
Performance Growth in 5 Years!

About David Blythe

Working at the hardware/software boundary for 25 years:

What	When	
DirectX*10/X11 on workstations	1985-1990	[Grad school]
IRISGL/OpenGL*/OpenGL++ on workstations	1991-2001	[SGI]
OpenGL ES on phones, embedded devices	2002-2003	[Khronos]
DirectX on PCs, phones	2003-2010	[Microsoft]
Processor graphics, media, computing across the continuum	2010-	[Intel]

IDF2011

INTEL DEVELOPER FORUM

David Blythe

Chief Graphics Software Architect

Sponsors of Tomorrow.™ intel

Agenda

- Goals
- Software Stacks and Abstractions
- Graphics Stack Historical Evolution
- Workstation Generation
- PC Generation
- Processor Graphics Generation

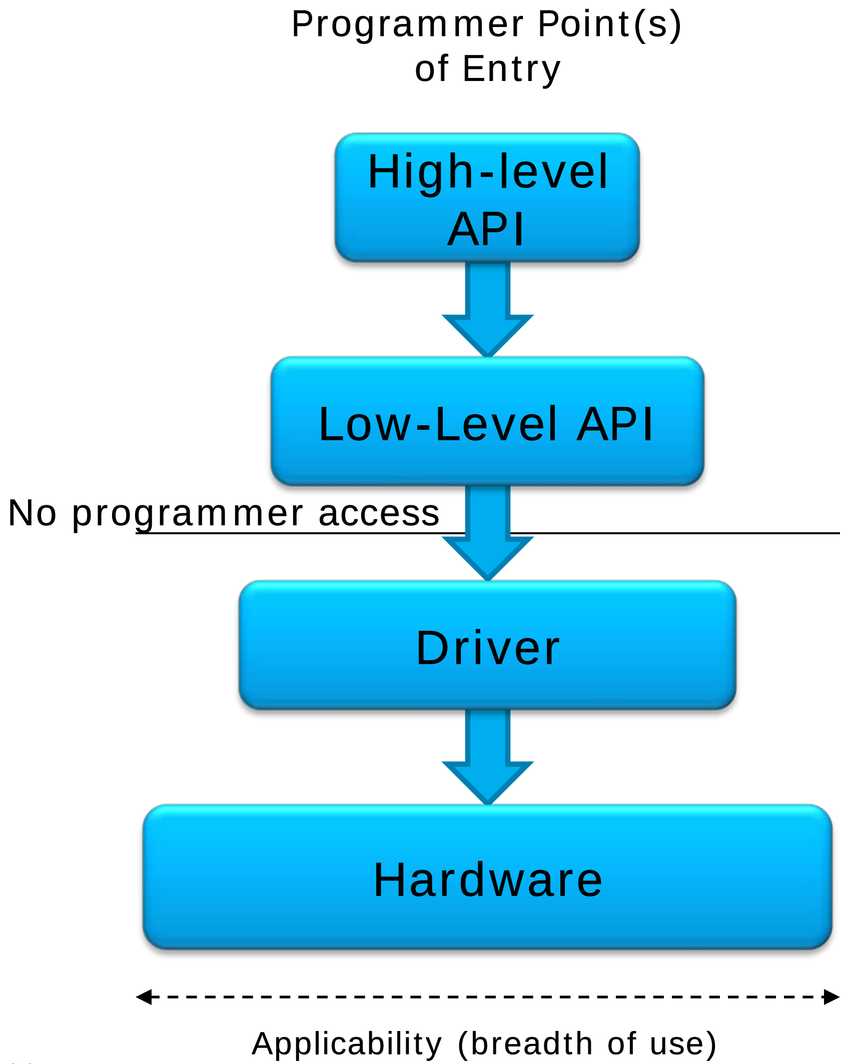
Goals

- Understand architecture of the SW/HW stack
  - Evolution over time
- Understand role of drivers, APIs, etc.
  - Look at evolution, dispel some myths
  - Look at tension points
- Understand how to make them better
  - Define what better means

Agenda

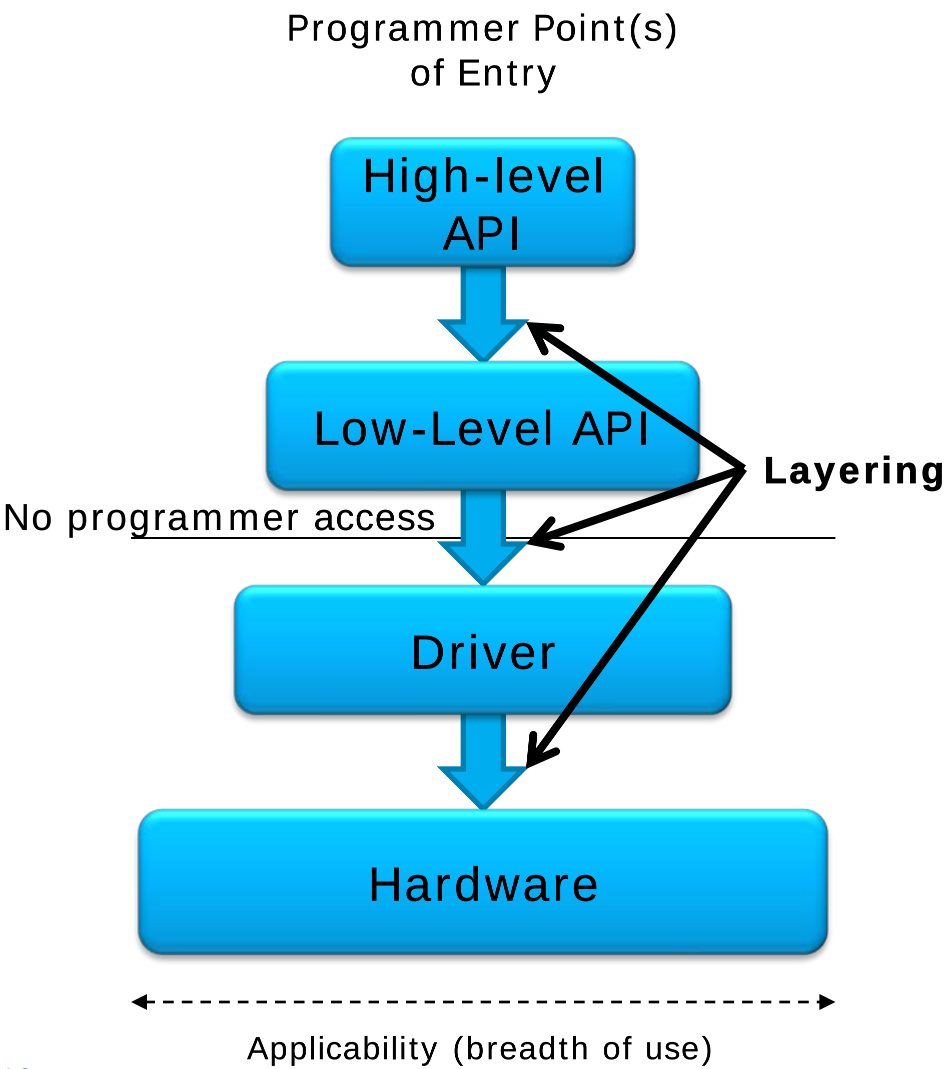
- Goals
- **Software Stacks and Abstractions**
- Graphics Stack Historical Evolution
- Workstation Generation
- PC Generation
- Processor Graphics Generation

Software Stacks and Abstraction



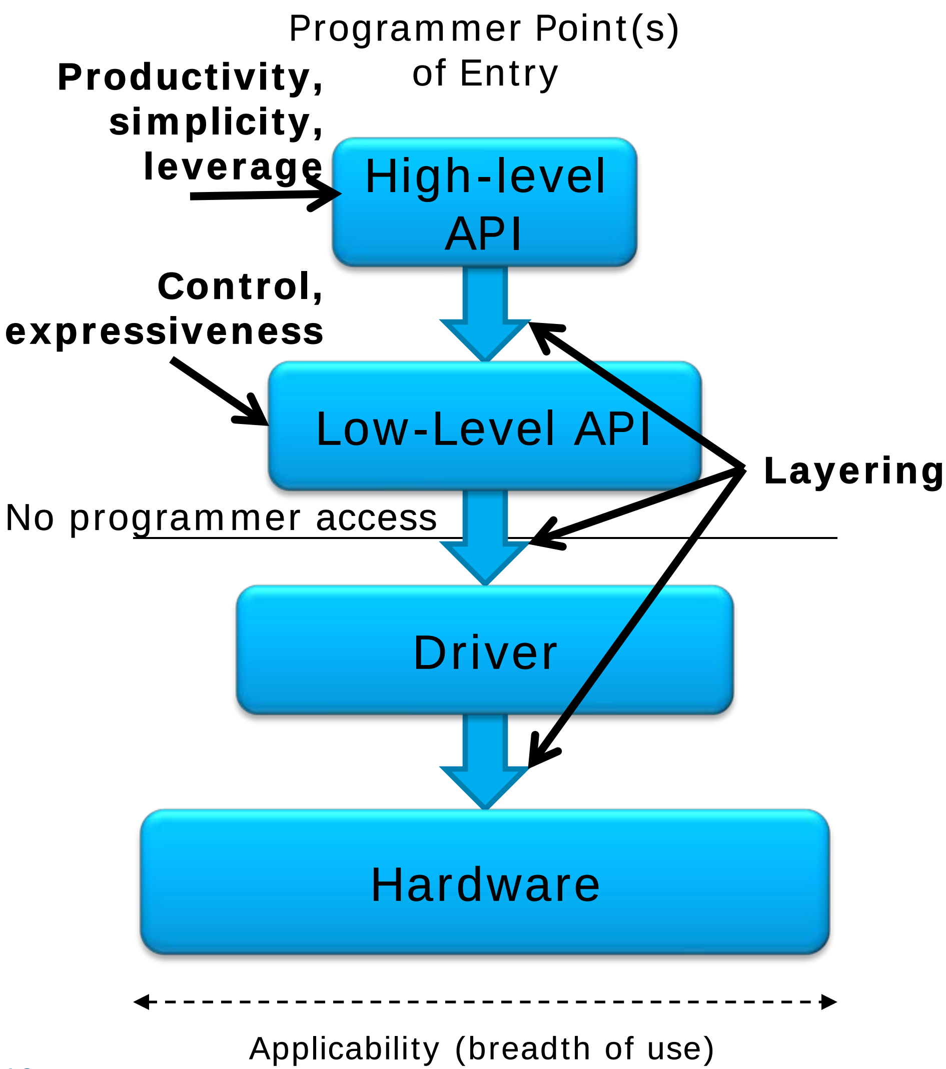
- Abstraction: reusable concepts, high-leverage, targeted

Software Stacks and Abstraction



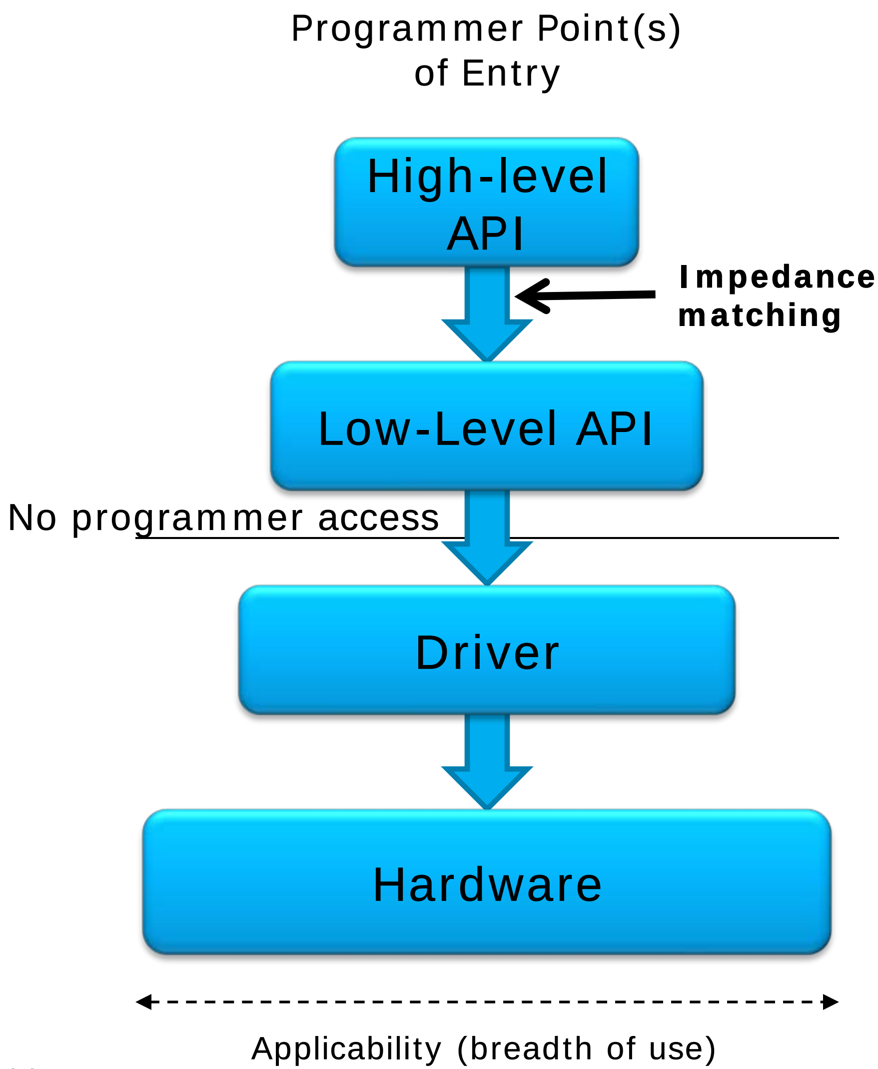
- Abstraction: reusable concepts, high-leverage, targeted
- Build in layers
  - More complex abstractions on top of smaller, less complex abstractions

Software Stacks and Abstraction



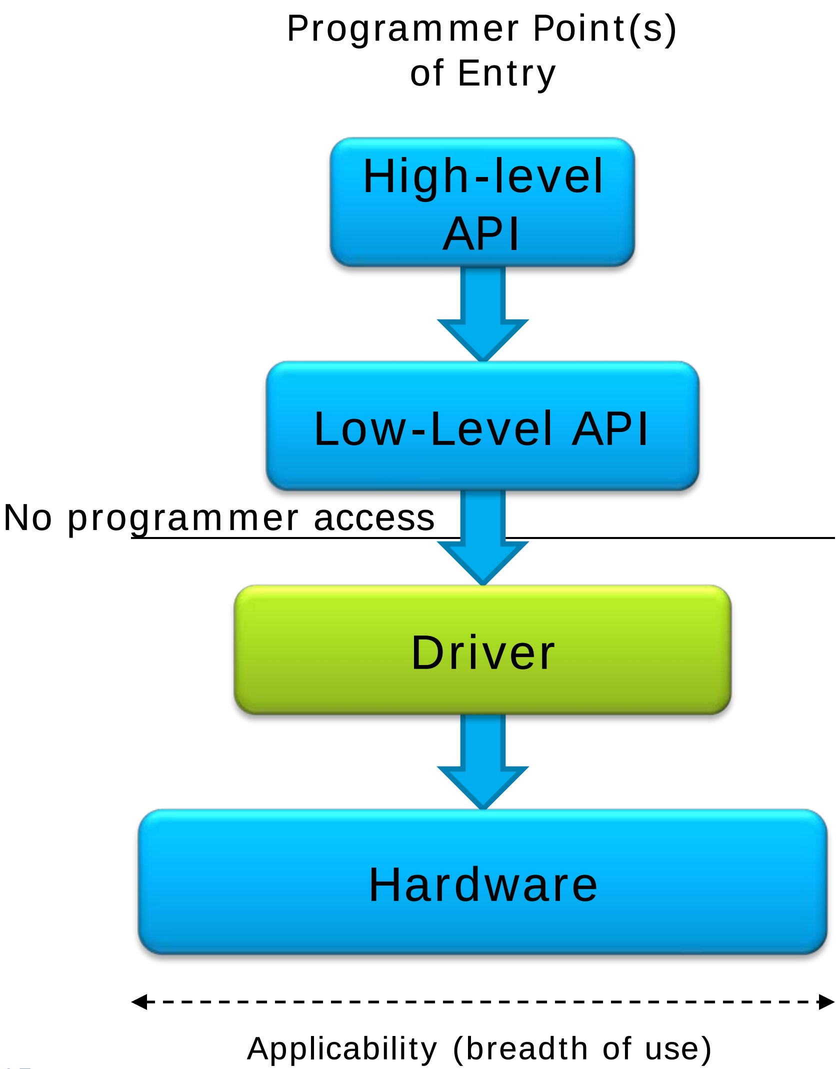
- Abstraction: reusable concepts, high-leverage, targeted
- Build in layers
  - More complex abstractions on top of smaller, less complex abstractions
- Higher-level abstractions are more purpose-specific
  - e.g., draw a line vs draw a menu
  - Multiple purpose-specific abstractions built on the same substrate

Building a Good Stack



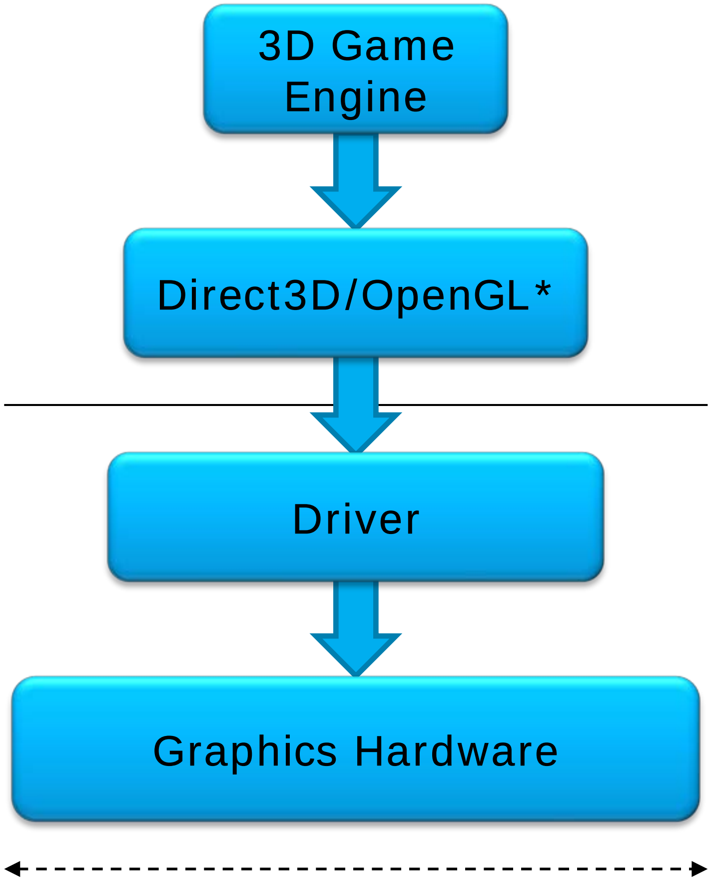
- Abstraction – layers need to have good impedance matches
- Each API must only make promises it can actually keep
  - Otherwise poor performance
  - If the HW is a poor match for low-level API
    - driver does extra work
  - If LL API a poor match for HL API
    - HL API does extra work
- Who is responsible for ensuring good matches?
  - For heterogeneous stacks - disparate standards
    - no one!
  - For closed OS vendor stacks
    - OS vendor

Building a Good Stack - Driver



- Driver really just another API
  - Purpose is to abstract HW into something low-level API can target
  - IHVs implement this layer
    - Responsible for making the HW appear uniform/portable
  - Interface to APIs is hidden to leave flexibility for later evolution
- Hiding implementation detail
  - Leaves room for evolution
  - **Tension on efficiency of abstraction vs. direct access**

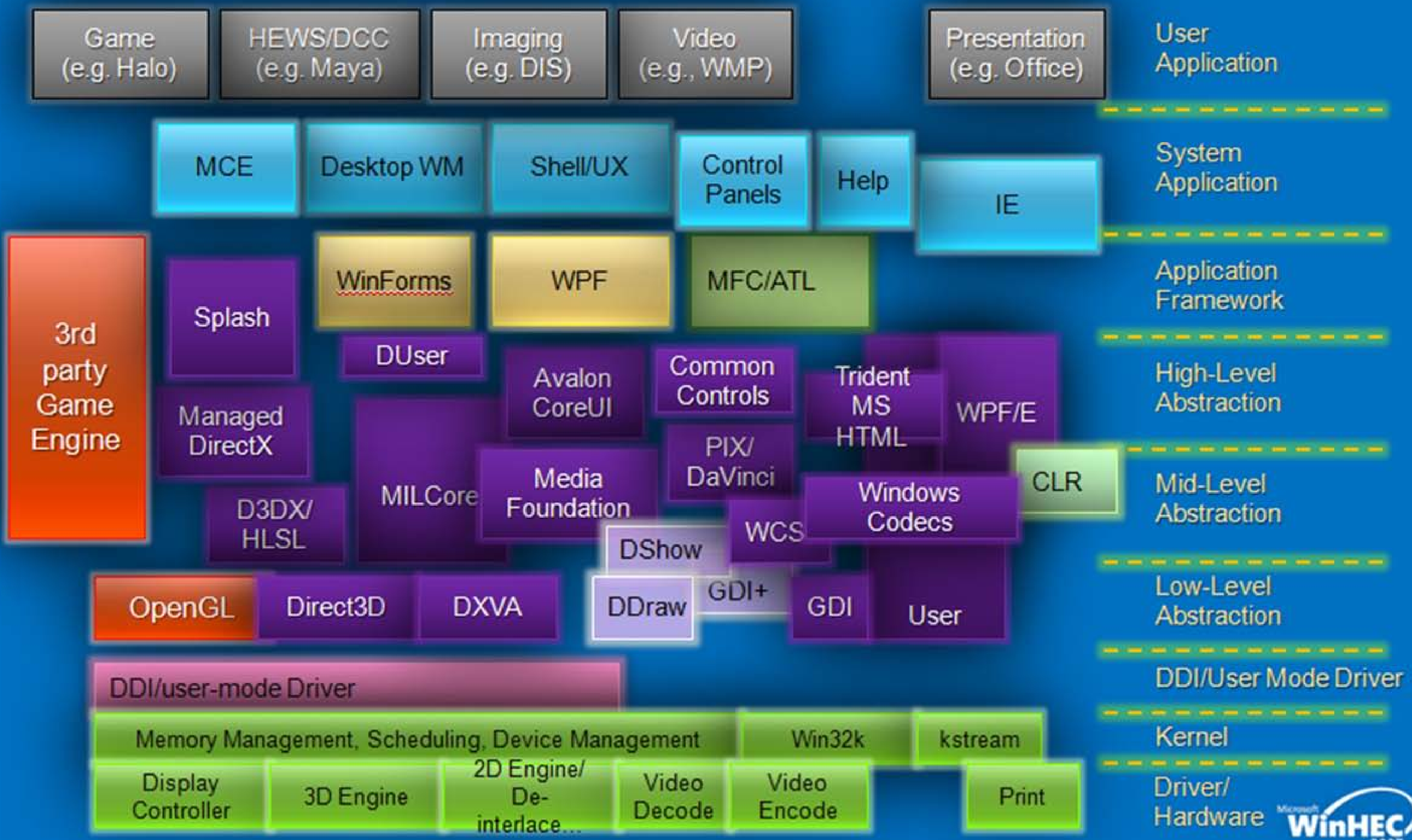
Graphics Stack Example



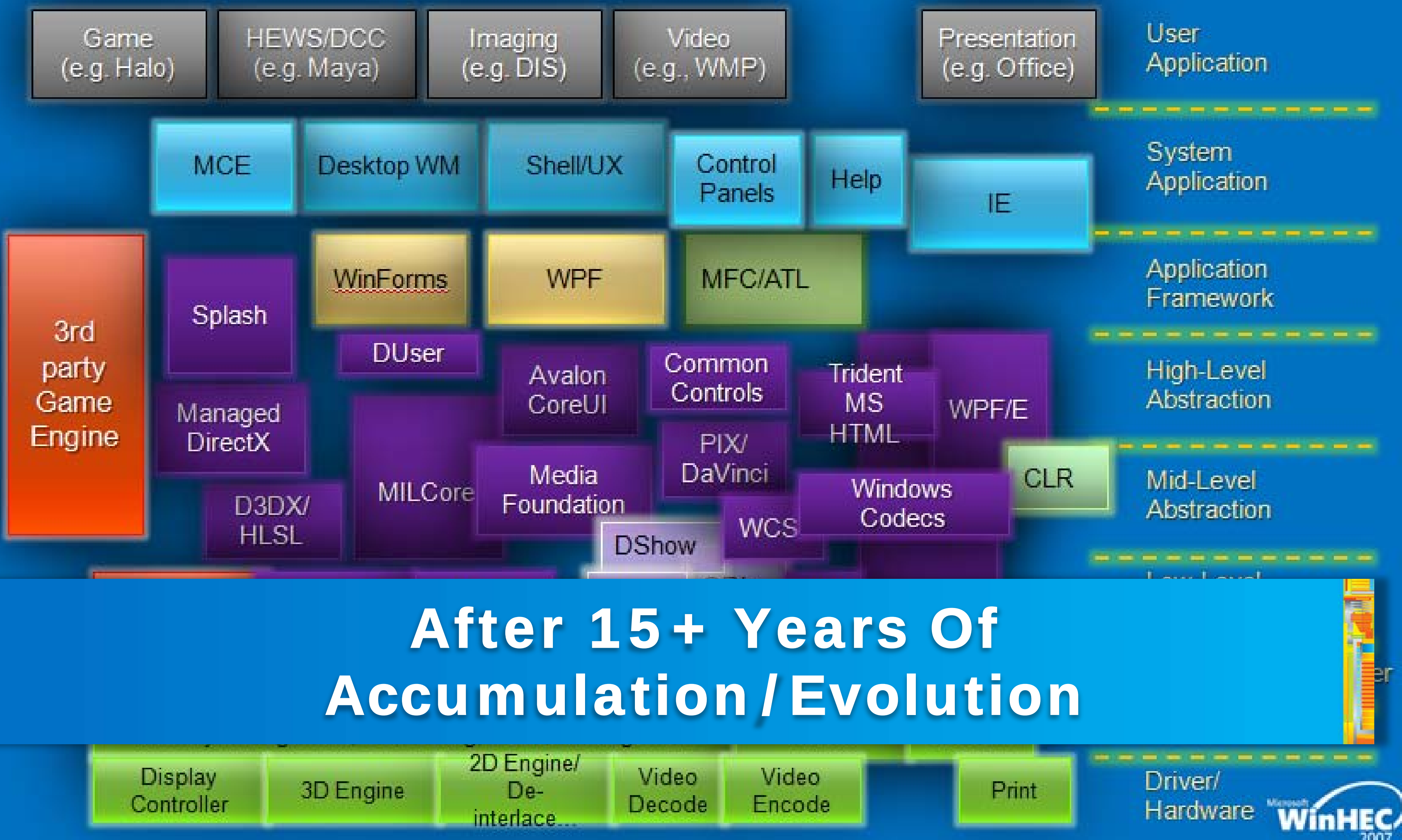
- 3D Game engine is narrow in purpose
  - e.g. PowerPoint*, html browser, ...
  - Focus on higher level concepts
    - Drawing full scenes with shading/illumination
    - Managing level of detail, etc.
- Microsoft* Direct3D much broader
  - Can implement lots of different types of applications with lots of effort
    - e.g., painful to implement GUI
- Driver supports Direct3D runtime
  - Maps API to HW commands
  - Supports resource management between multiple applications
- Repeat this process ...

IDF2011 INTEL DEVELOPER FORUM

Graphics/Gaming/Multimedia/ Presentation Stacks



Graphics/Gaming/Multimedia/ Presentation Stacks



Observations About SW Stacks

- Vertical stacks are necessary
  - Not practical to have everyone program “to the metal”
- Broad usage → hard to replace
  - Large applications can’t easily port to new APIs
  - Lower parts of stack hardest to replace
- Evolution rather than revolution
  - Reduce barriers to adoption of new features
  - Design for extendibility extremely important
- Still room for new APIs
  - But, they need to <u>interoperate</u> with old APIs

Agenda

- Goals
- Software Stacks and Abstractions
- **Graphics Stack Historical Evolution**
- Workstation Generation
- PC Generation
- Processor Graphics Generation

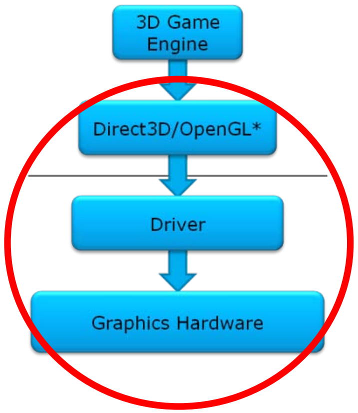
Graphics Stack Evolution

- Learn from the Past!
- Graphics around for a long time
  - More that 50 years!
- No need to go back that far
  - Look at low-level APIs
  - Focus on last couple of decades

Spacewar 1962
computerhistory.org

IDF2011 INTEL DEVELOPER FORUM

Evolution Over Last Few Decades



- Focus on system architecture

  = low-level API + driver + HW

  1990's – "Workstation" generation

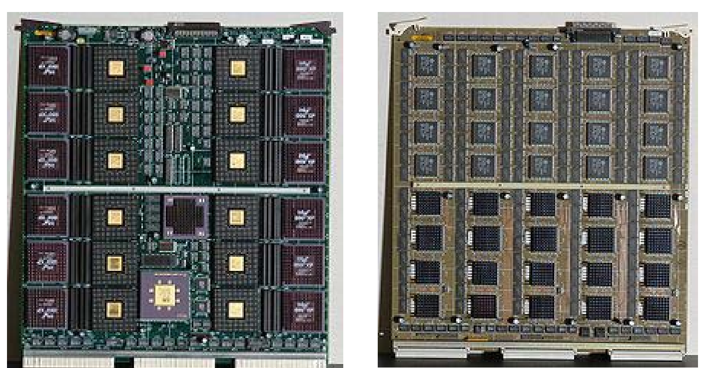
  2000's – "PC" generation

  2010's – "Processor Graphics " generation

- How did/do things work?
- What worked/didn't work?

Agenda

- Goals
- Software Stacks and Abstractions
- Graphics Stack Historical Evolution
- **Workstation Generation**
- PC Generation
- Processor Graphics Generation



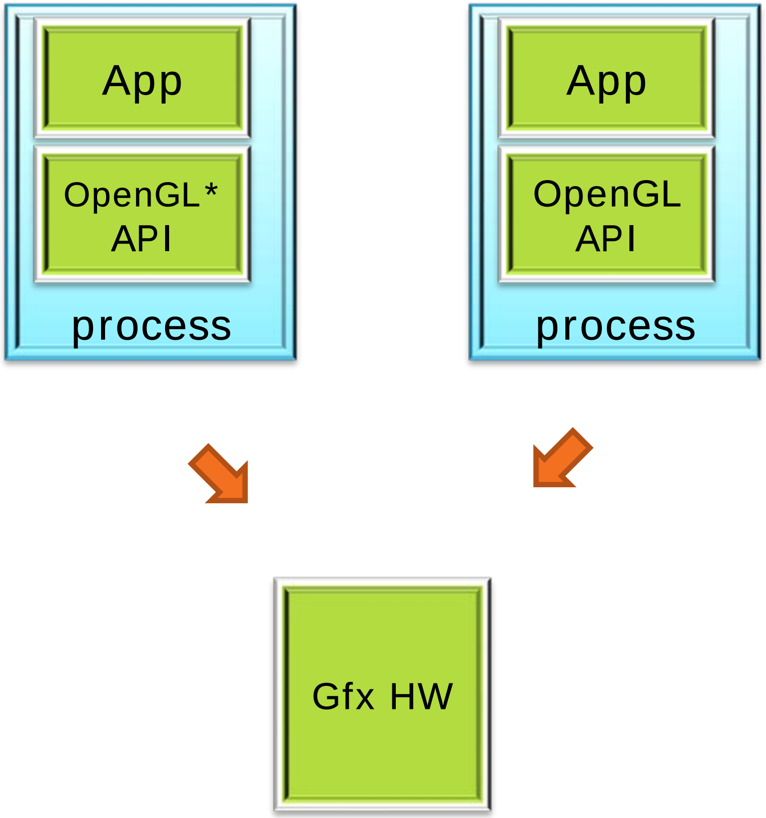
SGI* Reality Engine 1993

1990's Workstation Generation

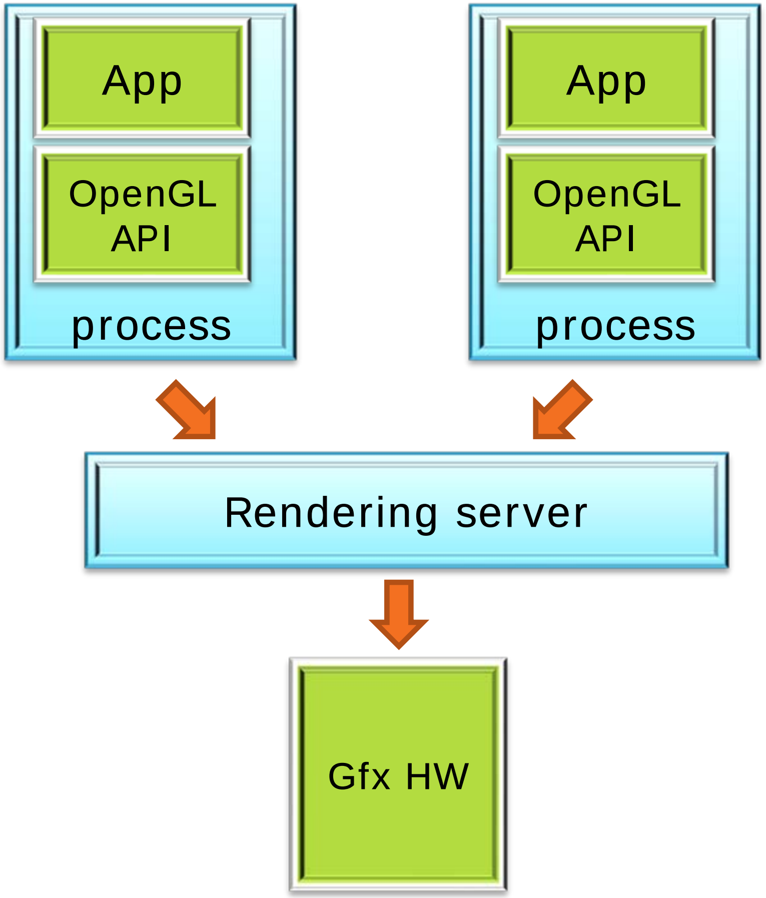
Fixed-Function, Immediate Mode APIs

- Fixed-function – performs a specific task
  - E.g., transform a vertex, shade a pixel
  - No programmability
- Immediate mode vs. declarative
  - Specify "how" to draw, step by step (GDI, OpenGL*)
  - Rather than "what" to draw (PHIGS, OpenInventor*)
  - Can implement declarative on top of immediate
    - Not vice-versa!
- Direct vs. Indirect rendering

Path to Hardware



Direct Rendering



“Indirect” Rendering

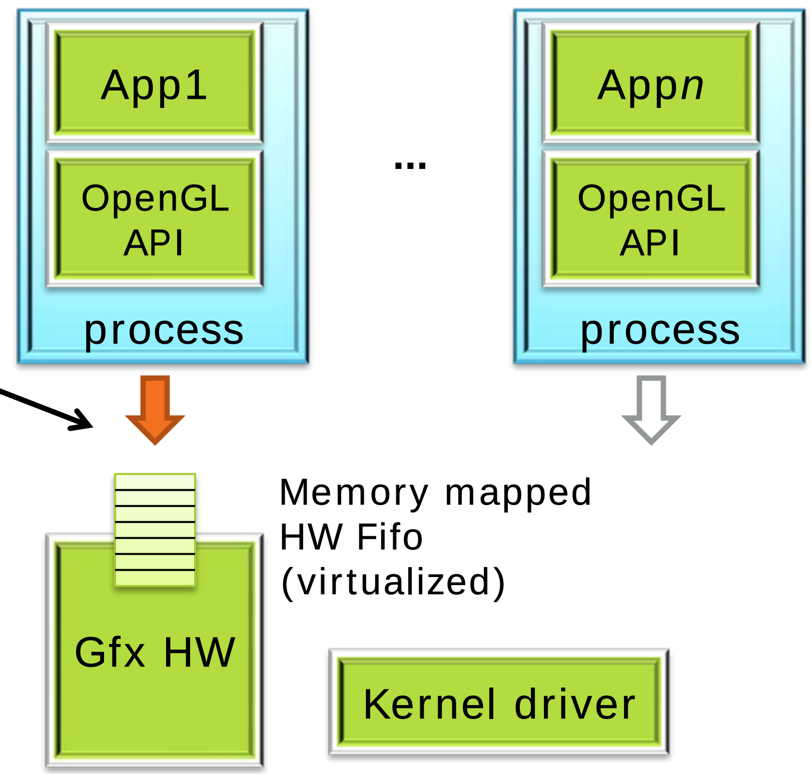
Workstation System Architecture

- OpenGL* - 3 layers:
  - API/library: command translation & submission
    - Assume HW validates command integrity
  - Kernel driver: global (machine wide) resource mgmt, app virtualization
    - context creation, memory allocation (textures, render targets), display, ...
  - HW: command execution

Command Submission

glVertex3fv():

- 1 jmp
- 3 loads
- 2 qw stores
- return

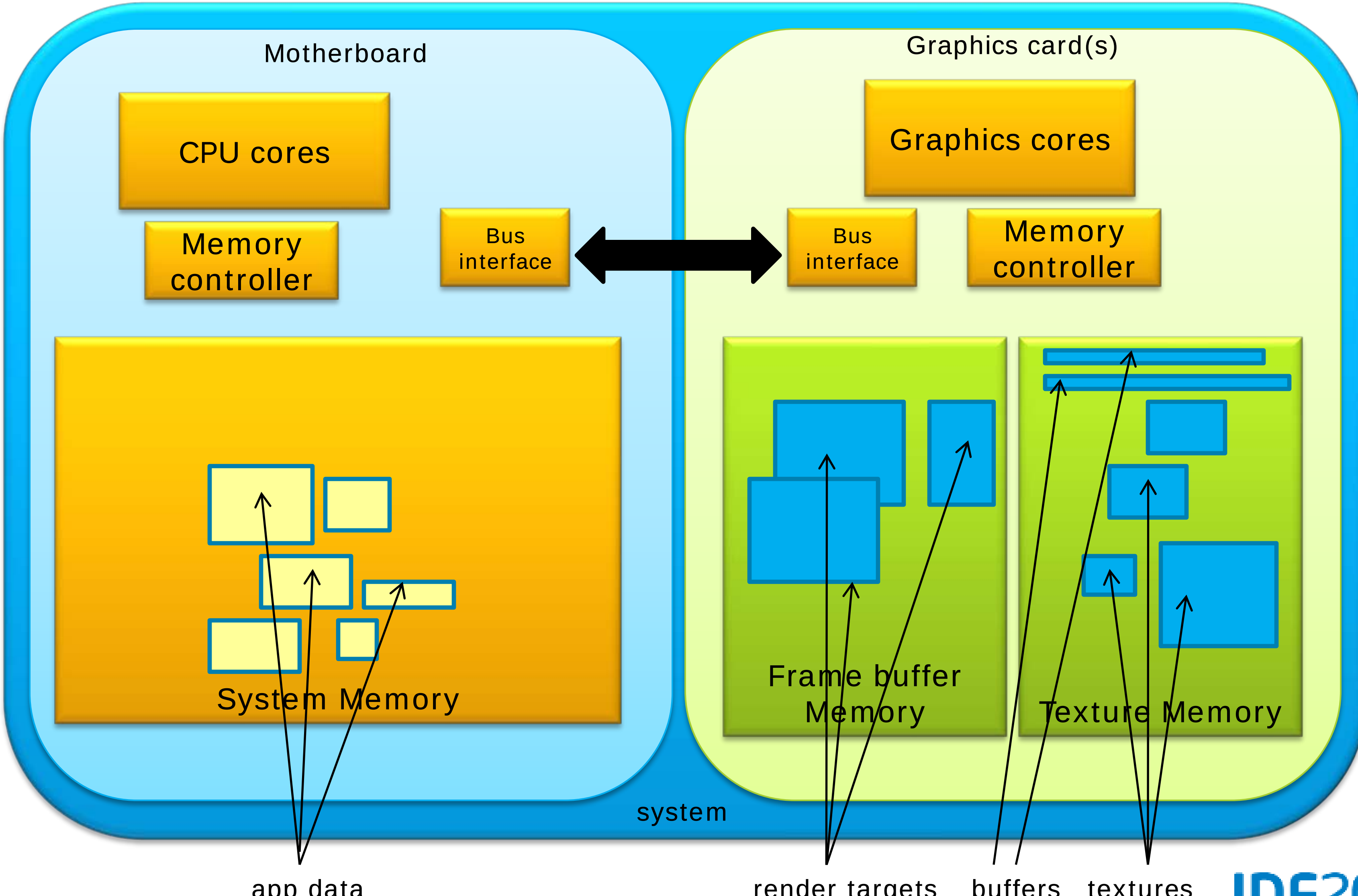


1993 OpenGL on Unix* workstation

OpenGL*

- Command stream sent to HW
  - State setting commands (color, lighting, texture maps)
  - Drawing commands (points, lines, triangles, images, ...)
- Best if 1:1 mapping API ↔ HW
  - Driver does (simple) translation of API commands to HW commands
- Pressure around overhead of managing state
  - Record state on host, defer state processing until draw time
    - E.g., bind a texture is slowest command

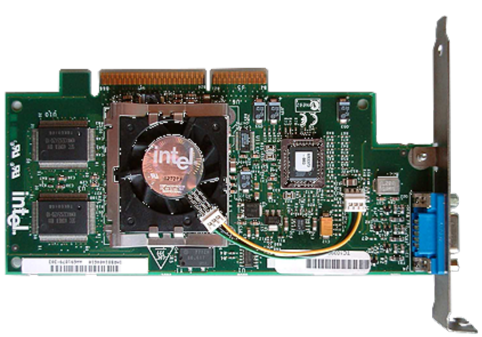
Workstation Resource Management



IDF2011
INTEL DEVELOPER FORUM

Agenda

- Goals
- Software Stacks and Abstractions
- Graphics Stack Historical Evolution
- Workstation Generation
- **PC Generation**
- Processor Graphics Generation



2000's – PC Generation

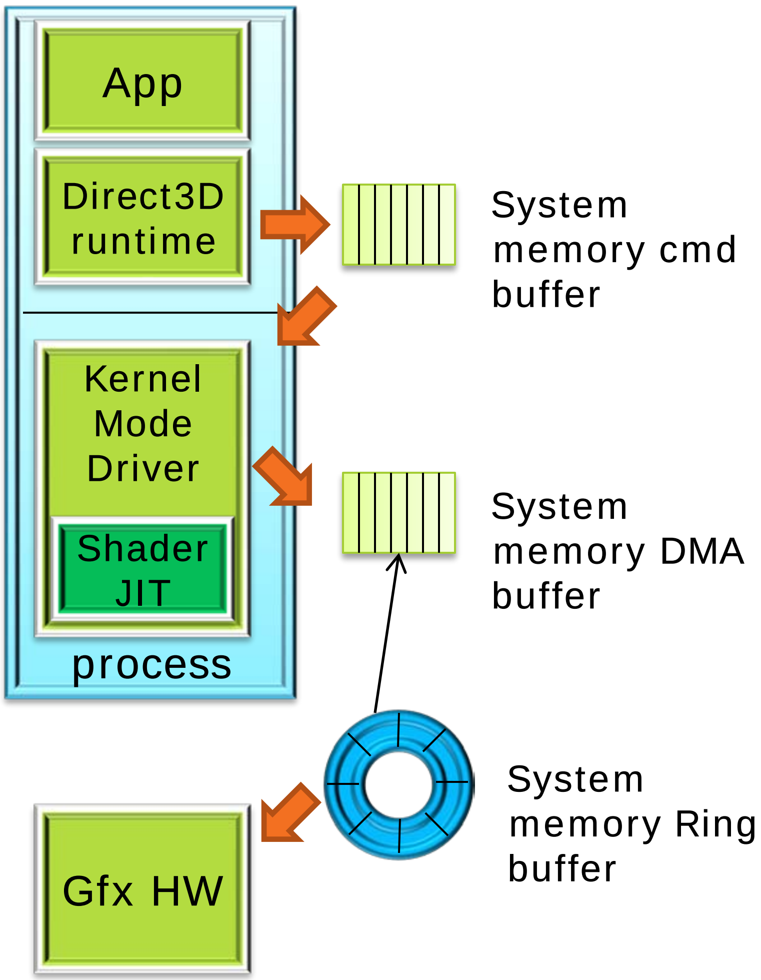
Rapid Feature Additions:

- Programmable vertex/pixel processing
  - New online JIT compilation stage
  - Runtime manages compiled shaders
  - Offline compiles to intermediate representation
    - E.g., HLSL
- Render to texture, lots of memory objects
  - Much more memory state to manage
- More logical pipeline stages
  - Tessellation, geometry shader
- Early "compute" APIs
  - DirectCompute, OpenCL*

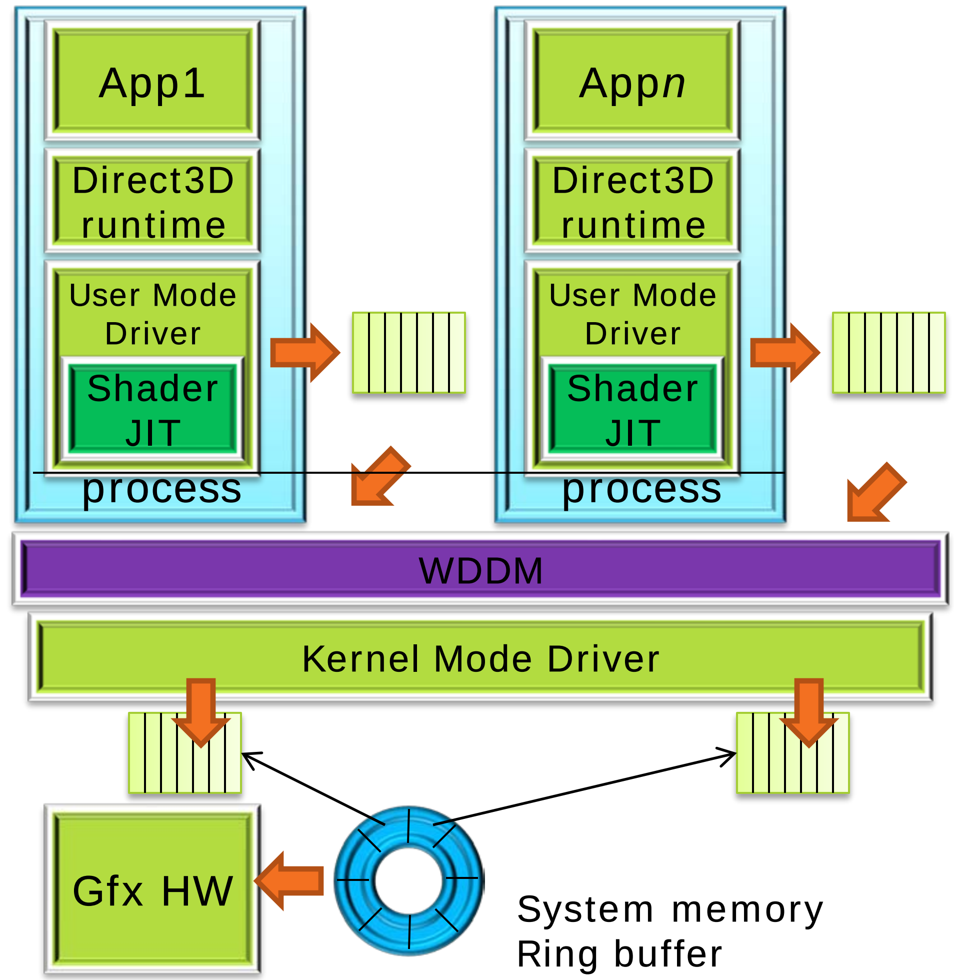
Changing System Requirements

- New challenges
  - Power management
  - Dynamic display configuration
  - Security/robustness, content protection
  - Hardware diversity
- Reuse 3D APIs for desktop rendering
  - Transition from “single app” to “multi-app”
- Virtualization of HW resources done in software
  - Oversubscribed memory → swapping/paging
  - Processing → time slicing
- Isolate applications from one another
  - i.e., enforce OS process boundaries

Buffer-based Command Submission



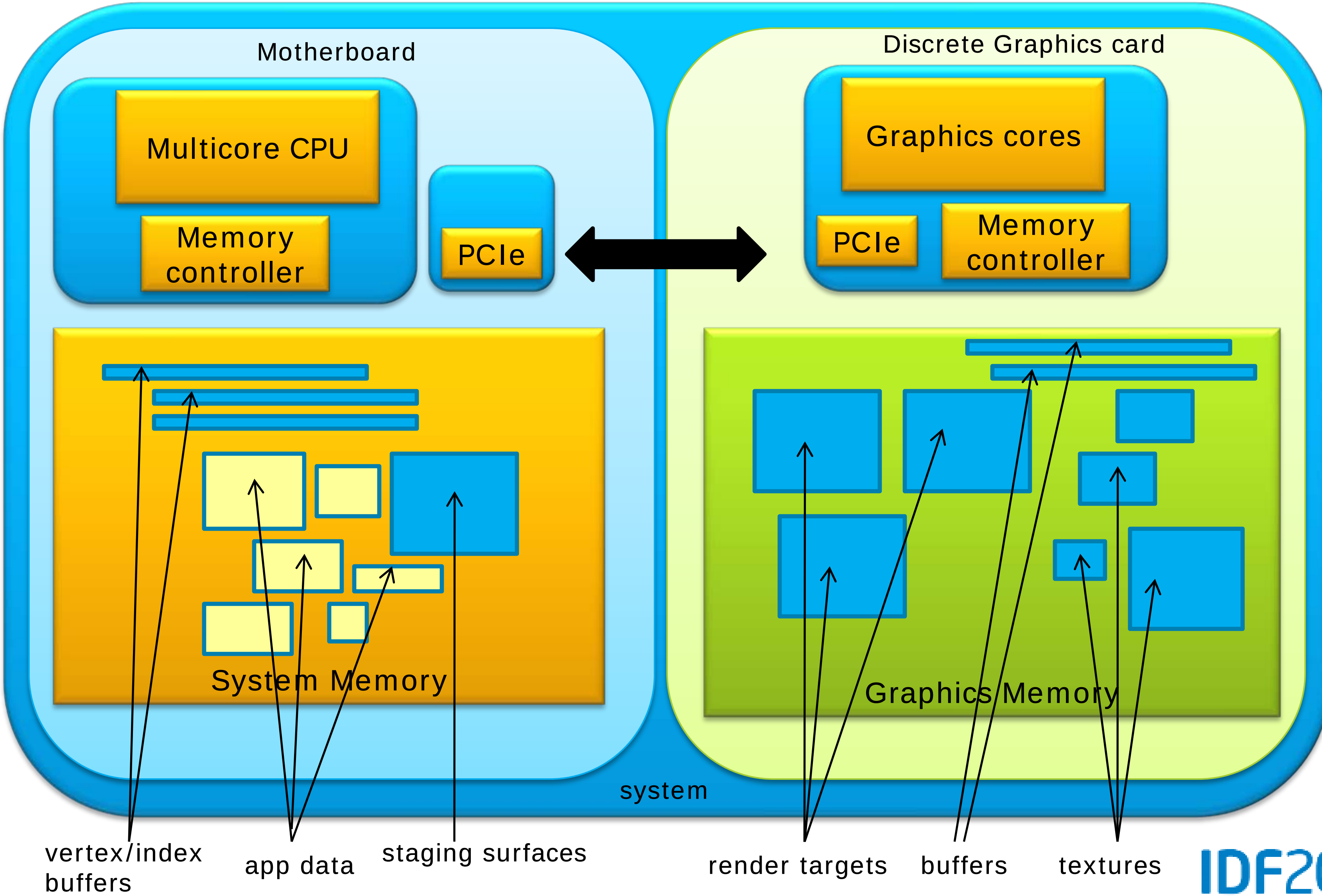
2003 Direct3D Microsoft* Windows*



2007 Direct3D Microsoft* Windows*

IDF2011
INTEL DEVELOPER FORUM

Resource Management



What Works

- Driver/low-level API factoring
  – APIs abstract driver model changes
  – Little driver-motivated API change
- Good compatibility over 15-20 years
  – E.g., Microsoft* Windows* still running Microsoft DirectX*7,8,9 ... apps
  – OpenGL* 1.0 (1993) apps still running
  – Not free, extra HW, SW work required
- Supports significant HW evolution
  – New API versions to expose new features
- “Resets” employed to address layer mismatches
  – E.g., major driver model change for Microsoft Windows Vista*
  – These are expensive for the ecosystem

Accumulated Problems

- Overhead in submitting commands to HW
  - Multiple memory writes
  - Driver transition from ring 3 to ring 0
- Latency and bandwidth constraints in moving data
- Complicated resource management
  - Which resources are in graphics memory vs. CPU memory?
  - Resource contents marshaled between memories
- Increasingly complicated API additions to overcome these problems

Applications are complex to write
Overheads limit scenarios

Agenda

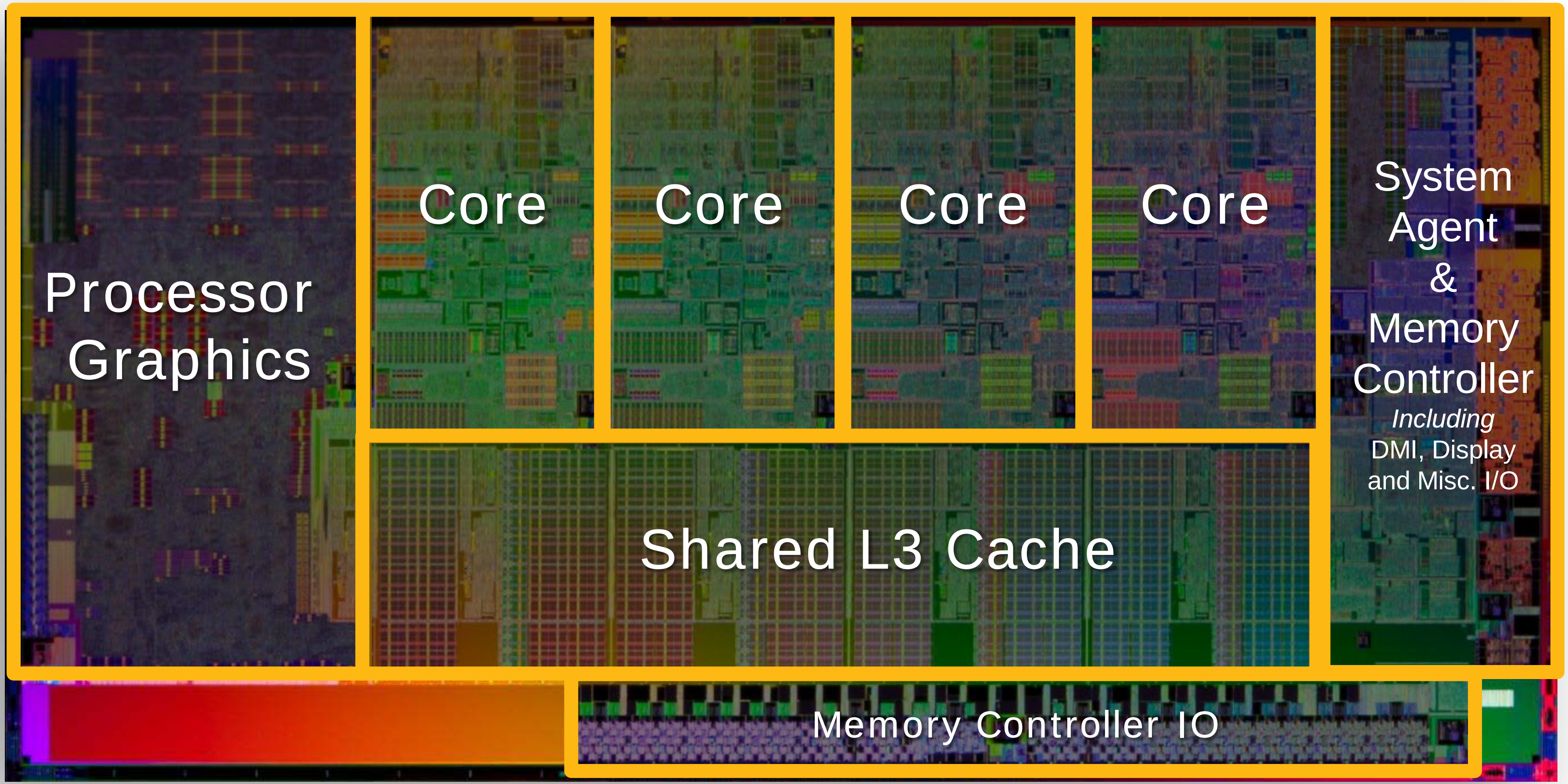
- Goals
- Software Stacks and Abstractions
- Graphics Stack Historical Evolution
- Workstation Generation
- PC Generation
- **Processor Graphics Generation**

2nd Generation Intel® Core™ processors (Sandy Bridge)

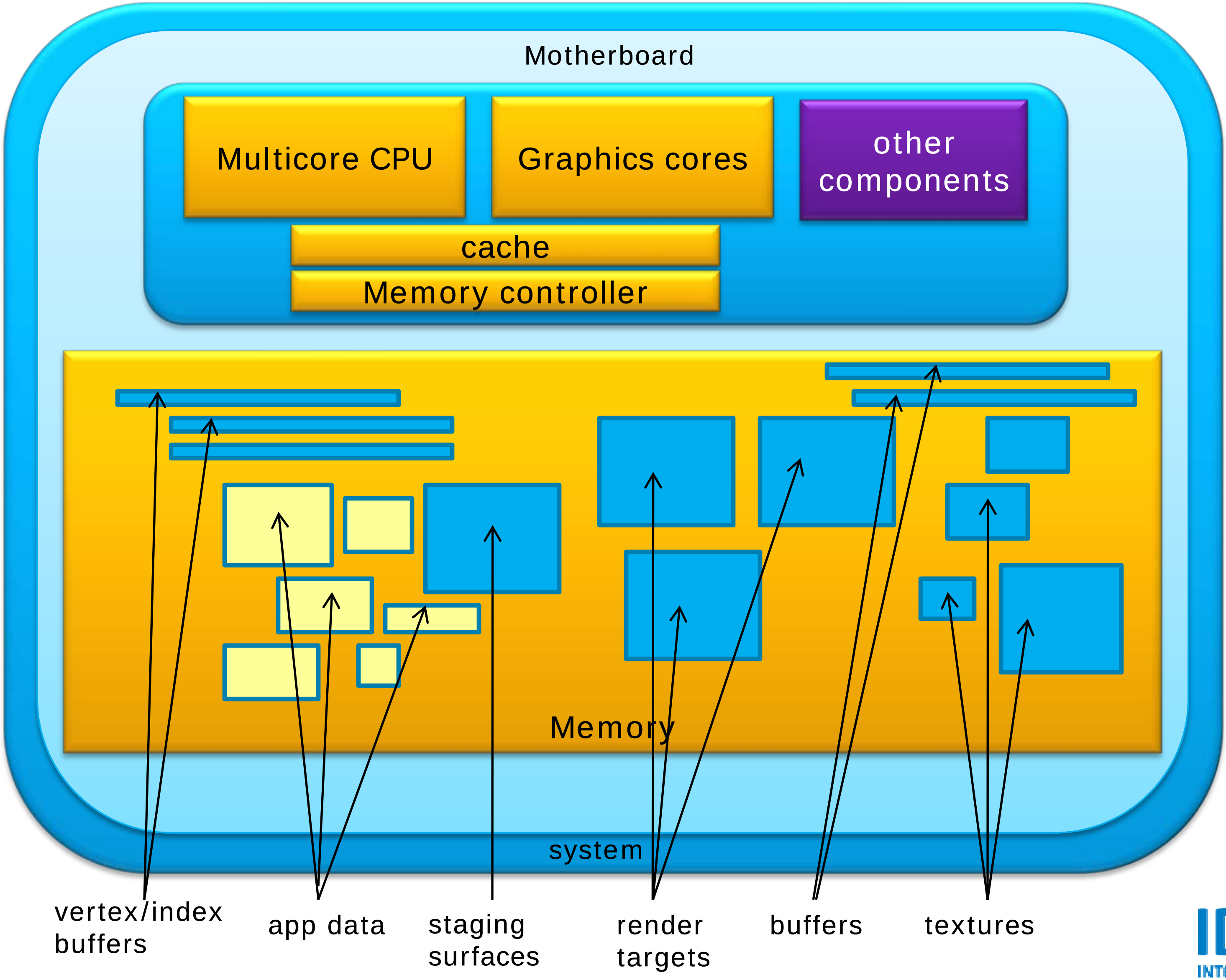
2010's – Processor Graphics Generation

- Graphics hardware acceleration ubiquitous
  - workstation, desktop, notebook, slate, phone
- On-die integration of CPU, Graphics, ...
  - Rename on-die graphics to **Processor Graphics**
  - Additional integration → system on a chip (SOC)
- What are the opportunities and challenges?

Early Example: Intel® Microarchitecture (Sandy Bridge)



Processor Graphics View



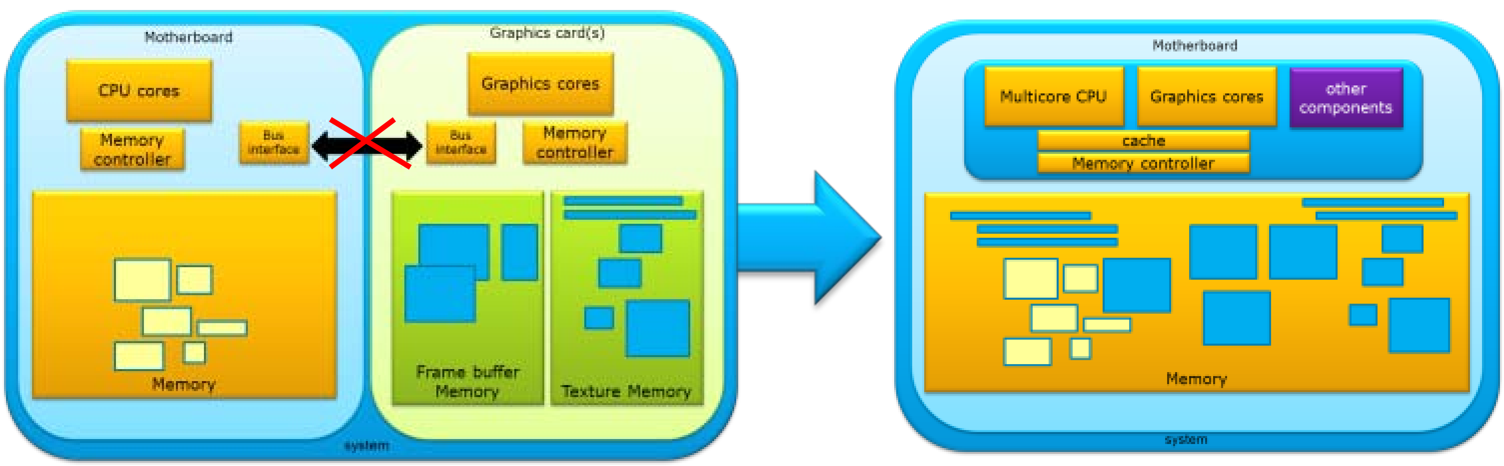
IDF2011 INTEL DEVELOPER FORUM

The Processor Graphics Opportunity

- Integration is efficiency
  - Higher bandwidths, lower latencies
- 3 classes of opportunities:
  - Improve data access and efficiency
  - Reduce command submission/return overhead
  - Agile allocation of power

More Efficient Data Management

- Graphics and other application resources in same physical memory
  - No external bus to cross
  - Move to same process address space
  - Remove copies and copying (marshaling)



IDF2011 INTEL DEVELOPER FORUM

Thinking Bigger

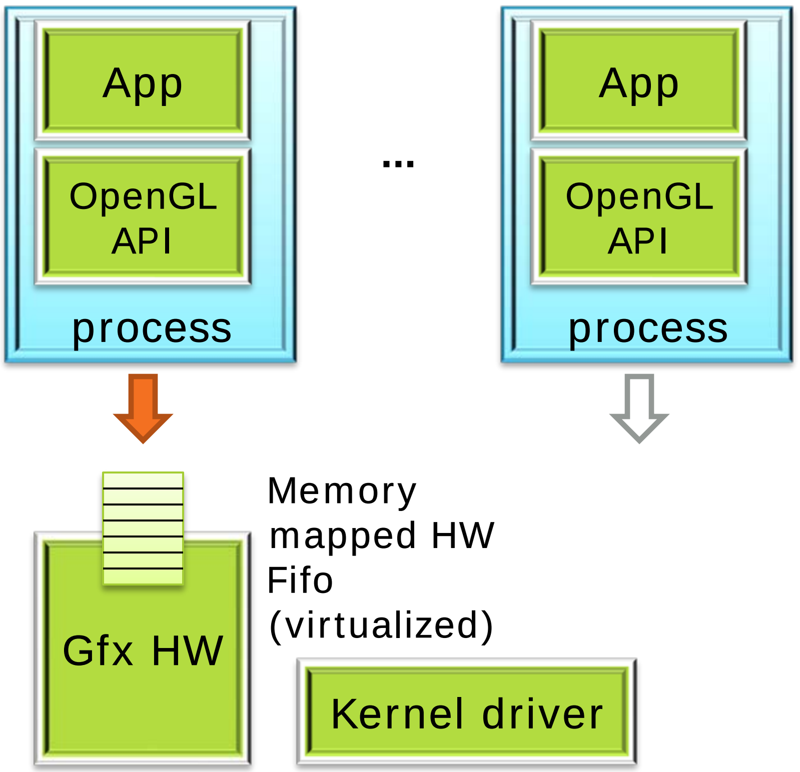
Eliminate memory allocation from APIs

- Use OS APIs instead
  - Malloc, HeapAlloc, ...
- Simplify graphics APIs to not create separate surfaces, vertex buffers, etc.
  - Flashback to workstation generation!
- Make surface “mapping” or “locking” a cheap operation
  - *“ ... what I most want to see is direct surfacing of the memory ...” - John Carmack*

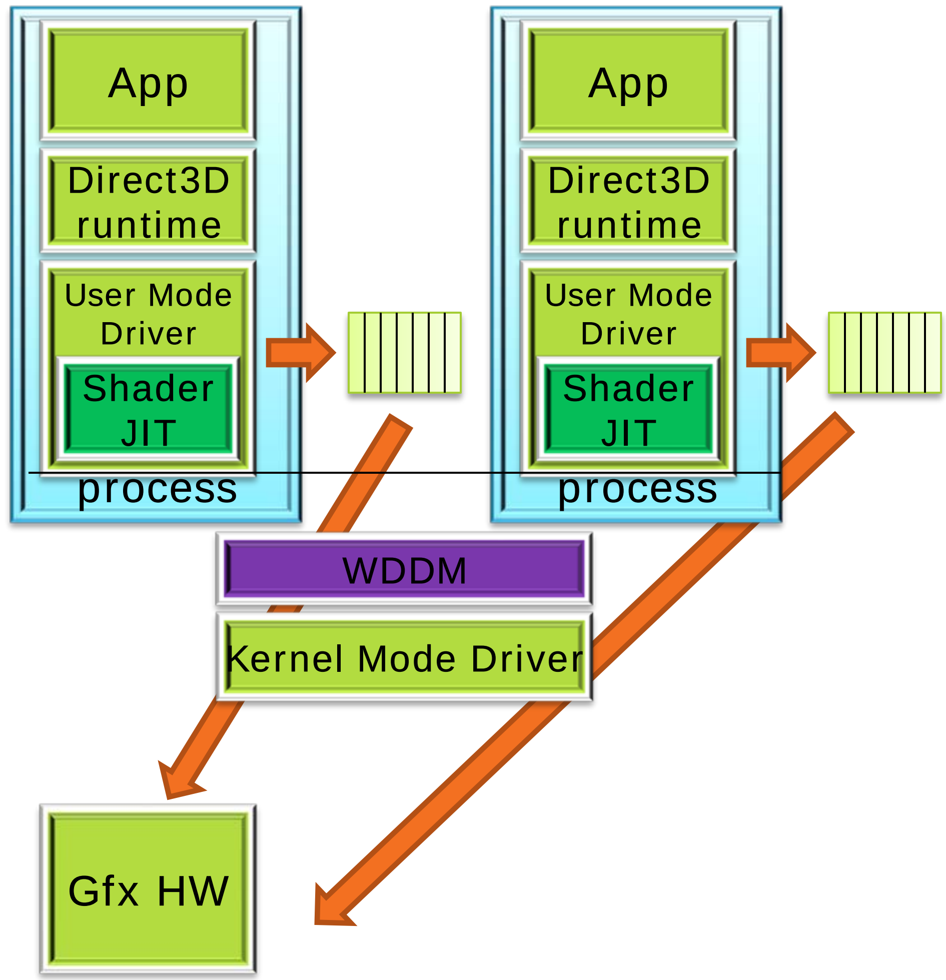
Command Submission Efficiency

- Previous attempts to improve efficiency
  - Direct3D 10 – refactor API
  - Increase amount of work per API call
- New opportunity:
  - Leverage shared physical memory
  - Avoid multiple copies of command buffers
- Can never compromise security, robustness
  - Must validate commands from untrusted ring 3

Can We Go Back to 1993?



1993 OpenGL* on Unix* workstation



Perhaps something like this?

Power Sharing

- Integration allows “big picture” power management
- Shift power between CPU, Graphics, other components

Power Allocation

- Track work, power, temperature
  - Adjust frequency, voltage on demand – power = $C_{dyn} * f * v^2$
  - Just need clever software
- Maybe add APIs to enable application hints
- Take further advantage of power "lulls"
  - Accumulate thermal credit for periods of low activity
  - E.g., Turbo = boost frequency beyond base
- New importance on running "lean" CPU code
  - No busy waiting, inefficient algorithms, ...
  - Untuned code is wasted power

API Evolution for Processor Graphics

- Change memory management model
  - Use OS memory allocation
- Change command submission model?
  - API is okay, underlying implementation is bad
- Better async communication between CPU and graphics
  - Are programmers comfortable with async programming?
- Application-assisted power management
  - Annotate CPU and graphics work phases

Summary

- Graphics API stack has evolved a lot
  - Useful insights from prior implementations
- Large, successful ecosystem built around current stack
- Significant improvement opportunities remain
  - Power management
  - Command transport
  - Memory management
- We can tackle these in Processor Graphics generation
  - Not in one big jump, but through a collection of changes

Call to Action

Processor Graphics generation is upon us

– Time to experiment and learn:

  - Power interactions between CPU and Graphics
  - Unified memory hierarchy

– And more to come ...

Legal Disclaimer

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the second quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should,” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including supply constraints and other disruptions affecting customers; customer acceptance of Intel’s and competitors’ products; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Potential disruptions in the high technology supply chain resulting from the recent disaster in Japan could cause customer demand to be different from Intel’s expectations. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; product mix and pricing; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. The majority of Intel’s non-marketable equity investment portfolio balance is concentrated in companies in the flash memory market segment, and declines in this market segment or changes in management’s plans with respect to Intel’s investments in this market segment could result in significant impairment charges, impacting restructuring charges as well as gains/losses on equity investments and interest and other. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting us from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the report on Form 10-Q for the quarter ended April 2, 2011.