Computer Organization and Structure

Bing-Yu Chen
National Taiwan University

Parallel Processors from Client to Cloud

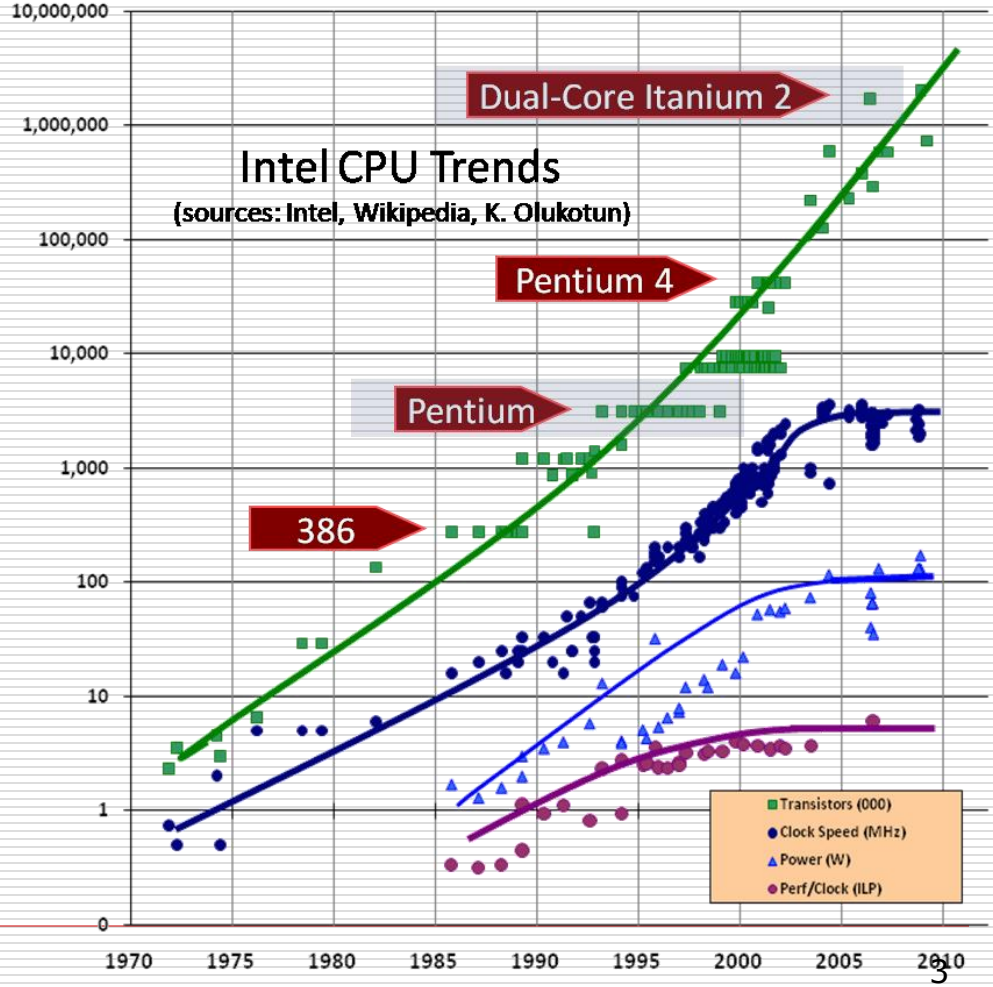
- The Difficulty of Creating Parallel Processing Programs
- Introduction to Graphics Processing Units

Goal of Computing

Faster, faster and faster

Why Parallel Computing?

- Moore's law is dead (for CPU frequency)



Top500 (Nov 2014)

1. Tianhe-2(NUDT)
   - 3,120,000 cores (Intel Xeon E5, Intel Xeon Phi)
2. Titan (Cray)
   - 560,640 cores (Opetron 6274, NVIDIA K20x)
3. Sequoia (IBM)
   - 1,572,864 cores (Power BQC)
4. K computer (Fujitsu)
   - 705,024 cores (Sparc64)
5. Mira (IBM)
   - 786,432 cores (Power BQC)

Introduction

- Goal: connecting multiple computers to get higher performance
  - Multiprocessors
  - Scalability, availability, power efficiency
- Task-level (process-level) parallelism
  - High throughput for independent jobs
- Parallel processing program
  - Single program run on multiple processors
- Multicore microprocessors
  - Chips with multiple processors (cores)

Hardware and Software

- Hardware
  - Serial: e.g., Pentium 4
  - Parallel: e.g., quad-core Xeon e5345
- Software
  - Sequential: e.g., matrix multiplication
  - Concurrent: e.g., operating system
- Sequential/concurrent software can run on serial/parallel hardware
  - Challenge: making effective use of parallel hardware

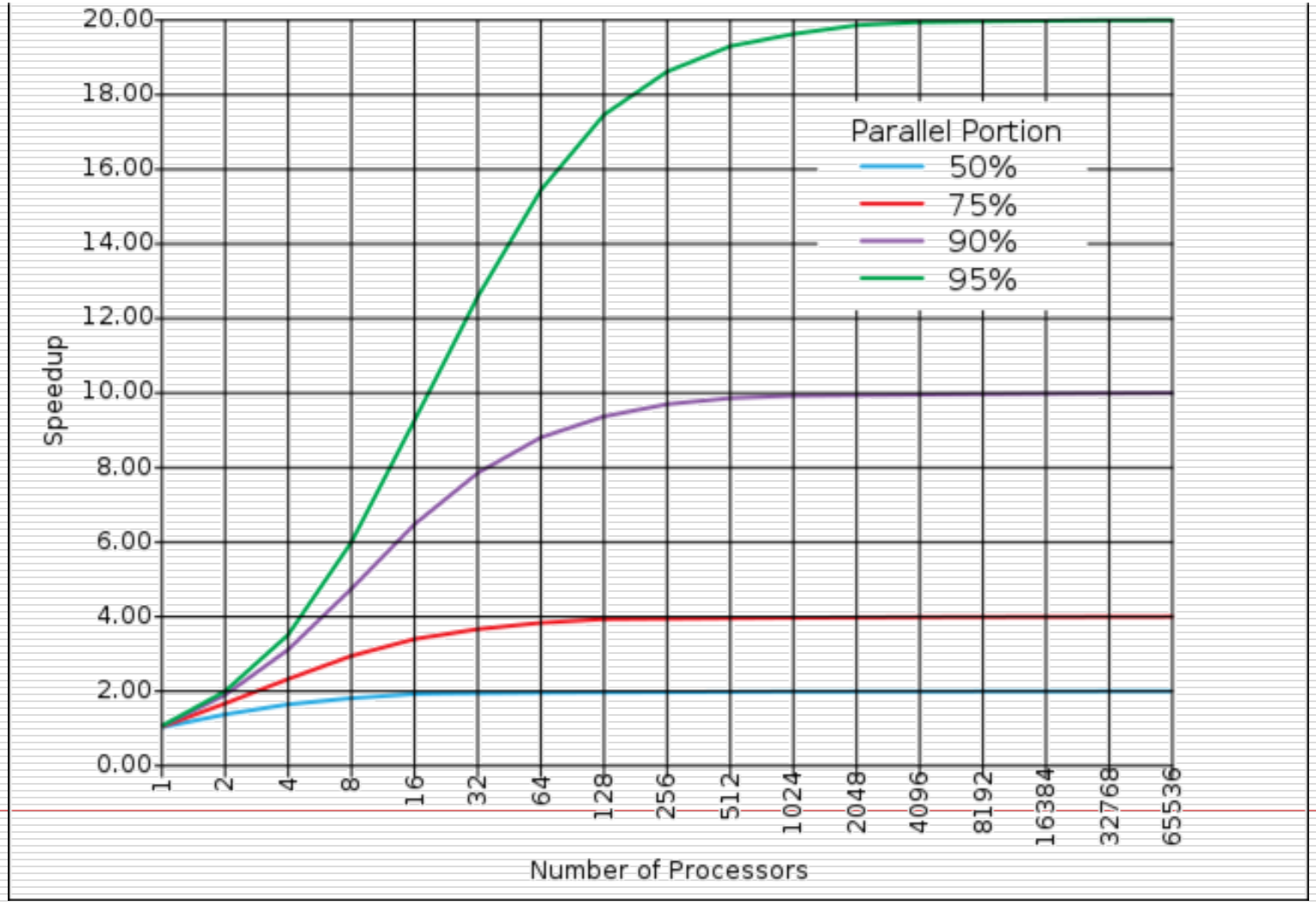
Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement
  - Otherwise, just use a faster uniprocessor, since it's easier!
- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead

Amdahl's Law

- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
  - $T_{new} = T_{parallelizable}/100 + T_{sequential}$
  - $\text{Speedup} = \frac{1}{(1-F_{parallelizable}) + F_{parallelizable}/100} = 90$
  - Solving: $F_{parallelizable} = 0.999$
- Need sequential part to be 0.1% of original time

Amdahl's law


20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Speedup
Parallel Portion
50%
75%
90%
95%
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors

Scaling Example

- Workload: sum of 10 scalars, and 10 × 10 matrix sum
  - Speed up from 10 to 100 processors
- Single processor: Time = (10 + 100) × t_{add}
- 10 processors
  - Time = 10 × t_{add} + 100/10 × t_{add} = 20 × t_{add}
  - Speedup = 110/20 = 5.5 (55% of potential)
- 100 processors
  - Time = 10 × t_{add} + 100/100 × t_{add} = 11 × t_{add}
  - Speedup = 110/11 = 10 (10% of potential)
- Assumes load can be balanced across processors

Scaling Example

- What if matrix size is 100 × 100?
- Single processor: Time = (10 + 10000) × t_{add}
- 10 processors
  - Time = 10 × t_{add} + 10000/10 × t_{add} = 1010 × t_{add}
  - Speedup = 10010/1010 = 9.9 (99% of potential)
- 100 processors
  - Time = 10 × t_{add} + 10000/100 × t_{add} = 110 × t_{add}
  - Speedup = 10010/110 = 91 (91% of potential)
- Assuming load balanced

Graphics Applications

- Movies
- Interactive entertainment
- Industrial design
- Architecture
- Culture heritage

History of Computer Graphics

- 1960, Ivan Sutherland's Sketchpad
  - The beginning of computer graphics
- 1992, OpenGL 1.0
- 1996, Voodoo I
  - The first consumer 3D graphics card
- 1996, DirectX 3.0
  - The first version including Direct3D

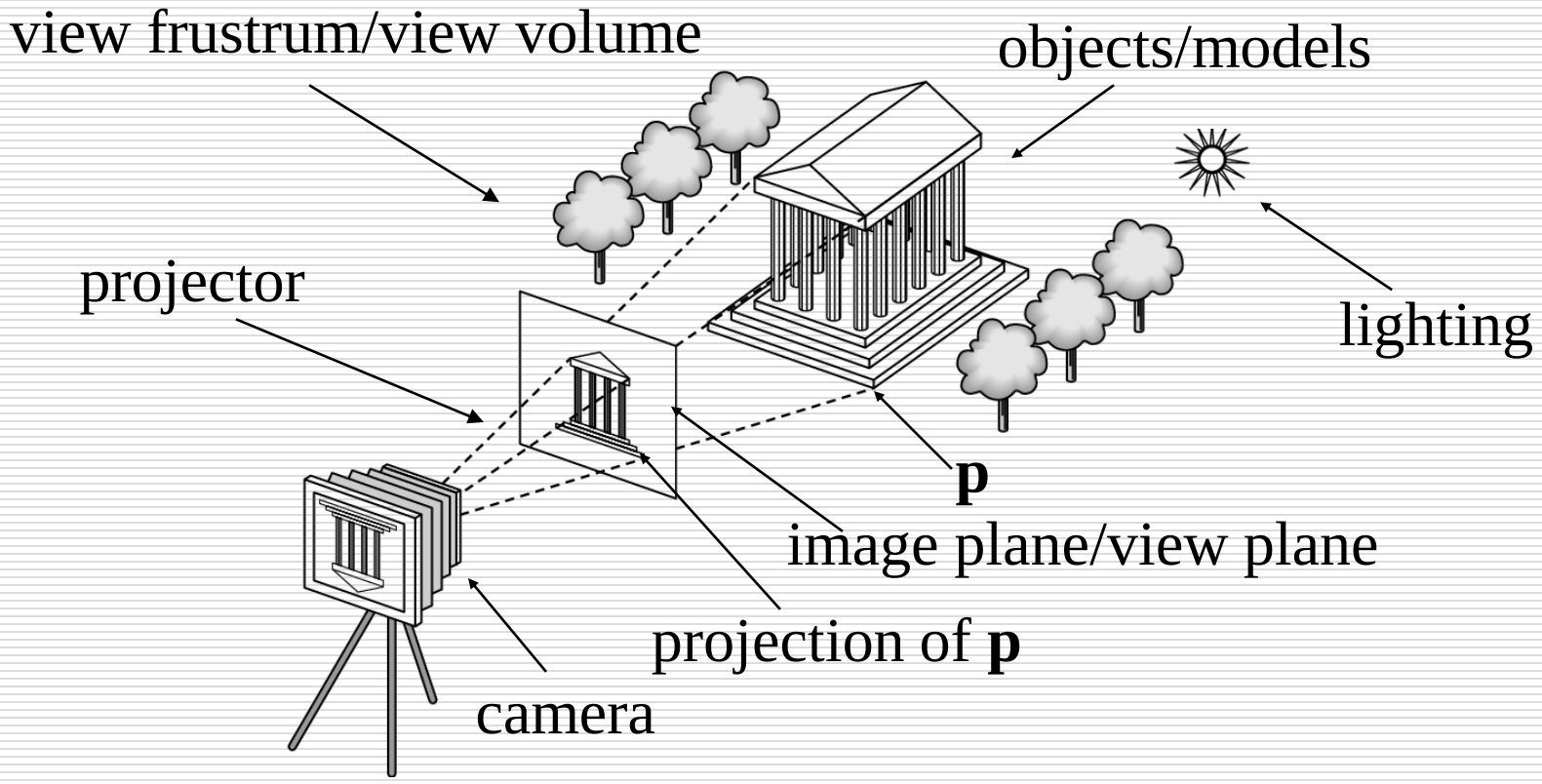
History of Computer Graphics

- 2000, DirectX 8.0
  - The first version supporting HLSL
- 2001, GeForce 3 (NV20)
  - The first consumer GPU
- 2004, OpenGL 2.0
  - The first version supporting GLSL
- 2006, GeForce 8 (G80)
  - The first NVIDIA GPU supporting CUDA
- 2008
  - OpenCL (Apple, AMD, IBM, Qualcomm, Intel, …)

History of GPUs

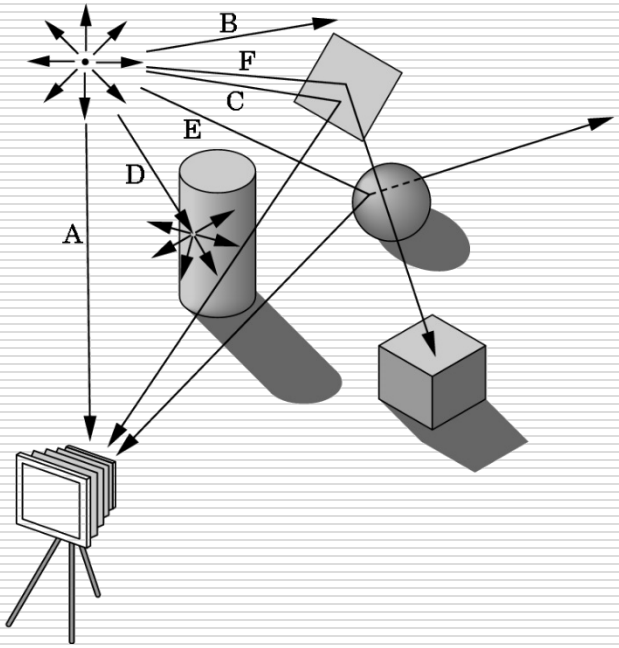
- Early video cards
  - Frame buffer memory with address generation for video output
- 3D graphics processing
  - Originally high-end computers (e.g., SGI)
  - Moore's Law $\Rightarrow$ lower cost, higher density
  - 3D graphics cards for PCs and game consoles
- Graphics Processing Units
  - Processors oriented to 3D graphics tasks
  - Vertex/pixel processing, shading, texture mapping, rasterization

Synthetic Camera Model



Ray Tracing and Geometric Optics

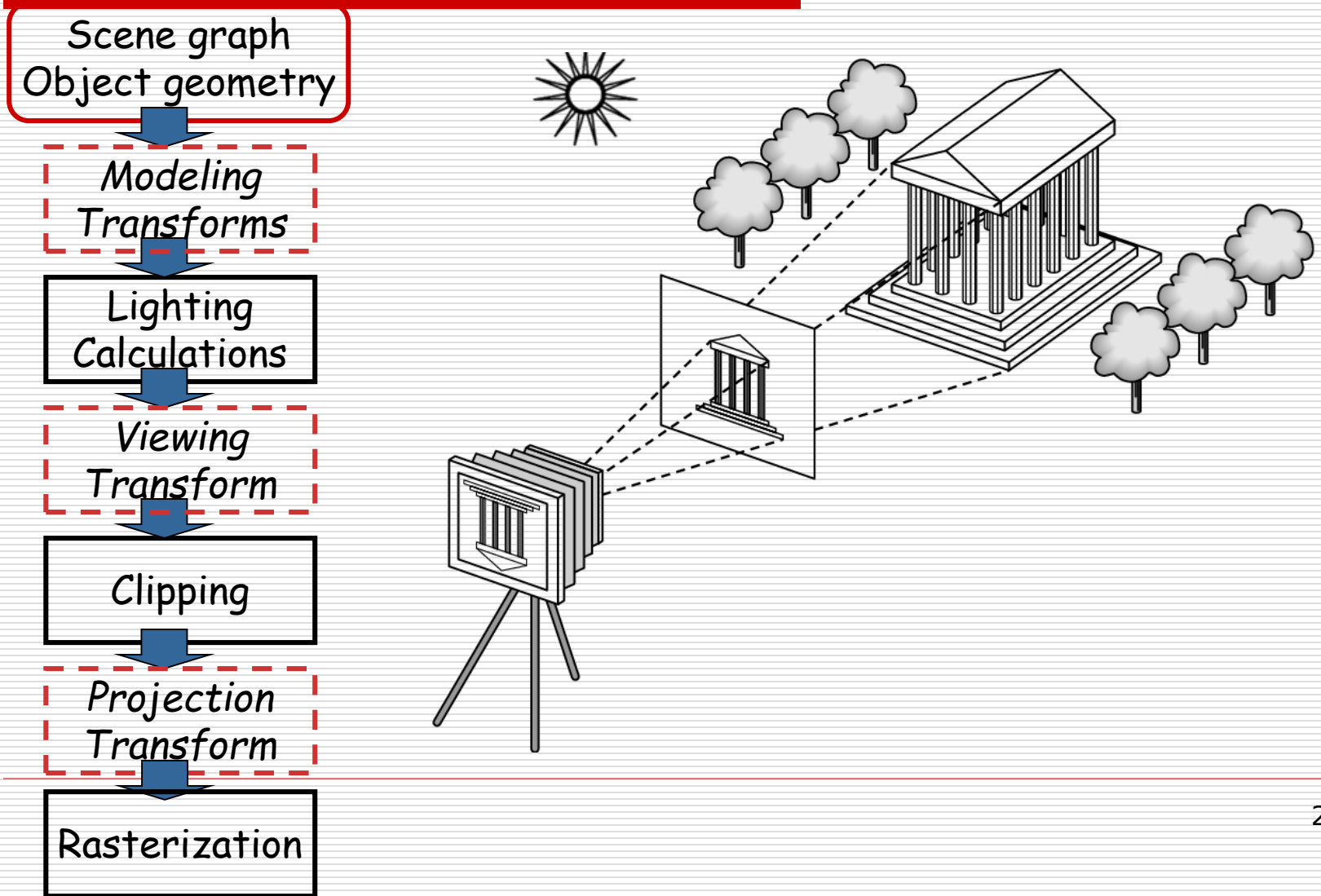
One way to form an image is to follow rays of light from a point source determine which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.



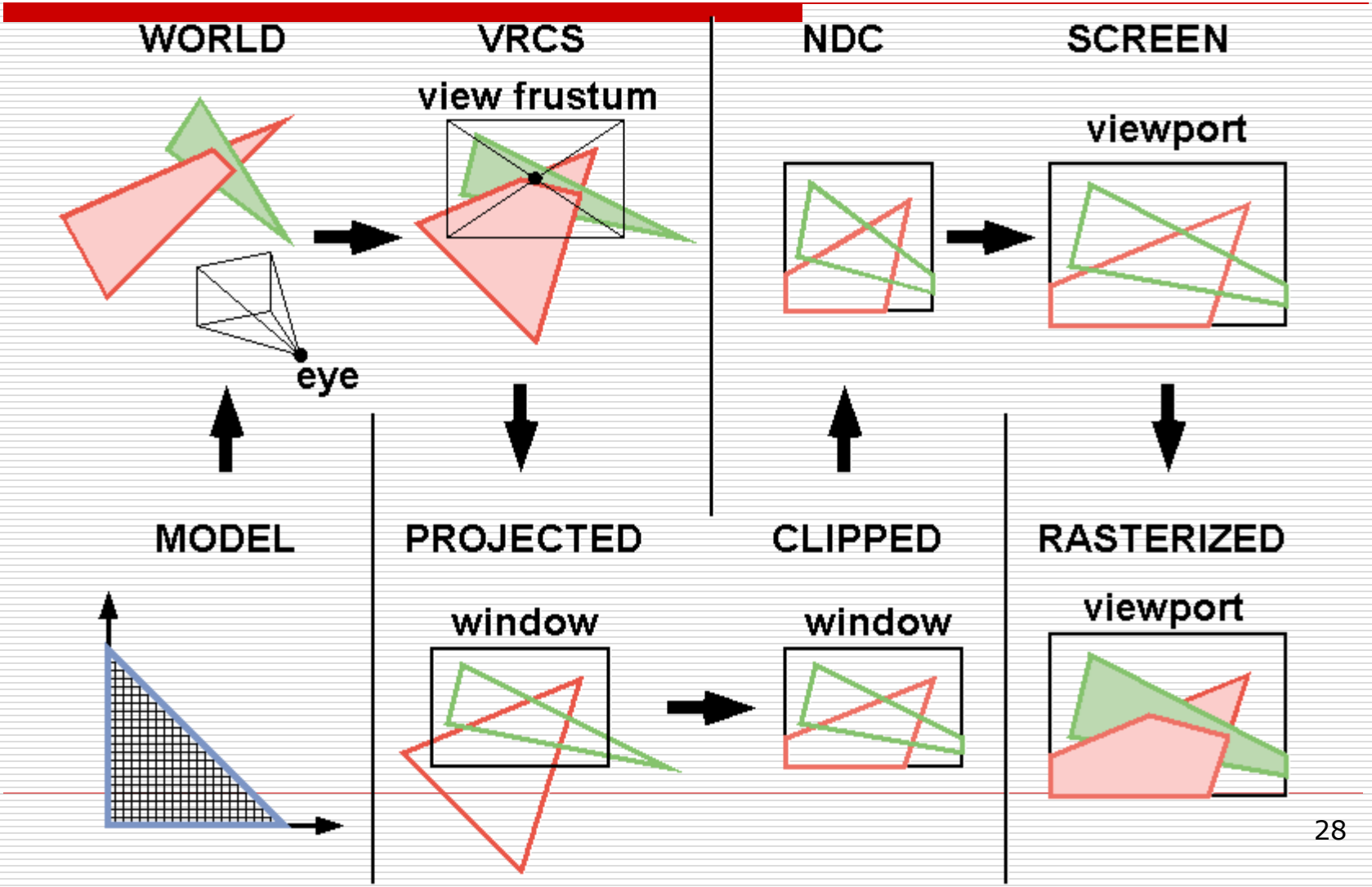
Why not ray tracing?

- Ray tracing seems more physically based so why don't we use it to design a graphics system?
- Possible and is actually simple for simple objects such as polygons and quadrics with simple point sources
- In principle, can produce global lighting effects such as shadows and multiple reflections but is slow and not well-suited for interactive applications

The Rendering Pipeline



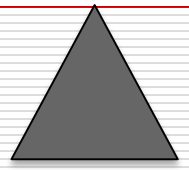
Computer Graphics Rendering



Graphics Pipeline

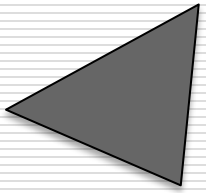
Instructions
States
Data

Input processor



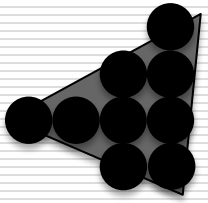
Transforms
Lighting, etc.

Do geometry stuff



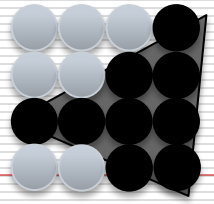
Rasterize
Pixel shading

Do pixel stuff

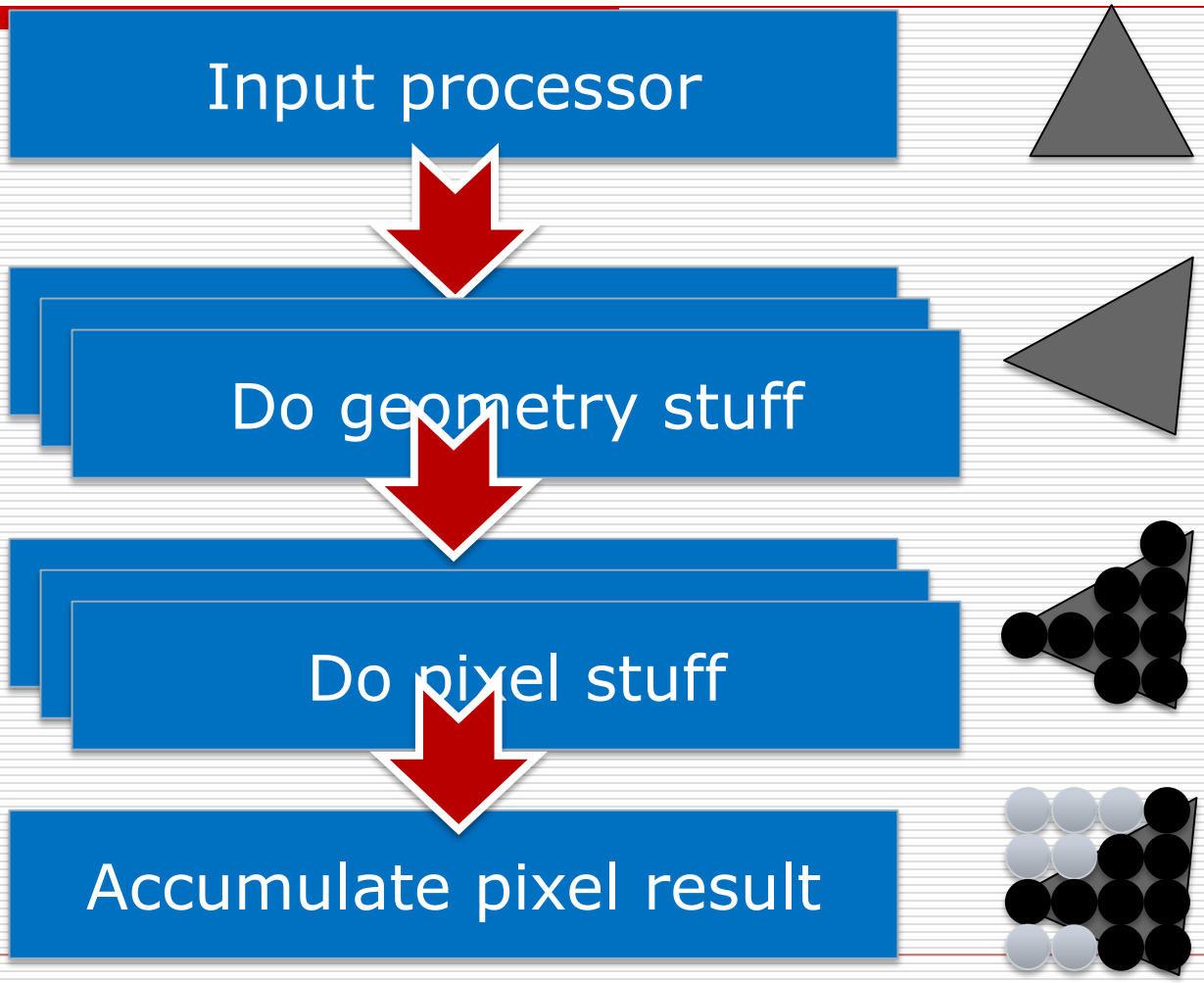


Z-buffer
Transparency

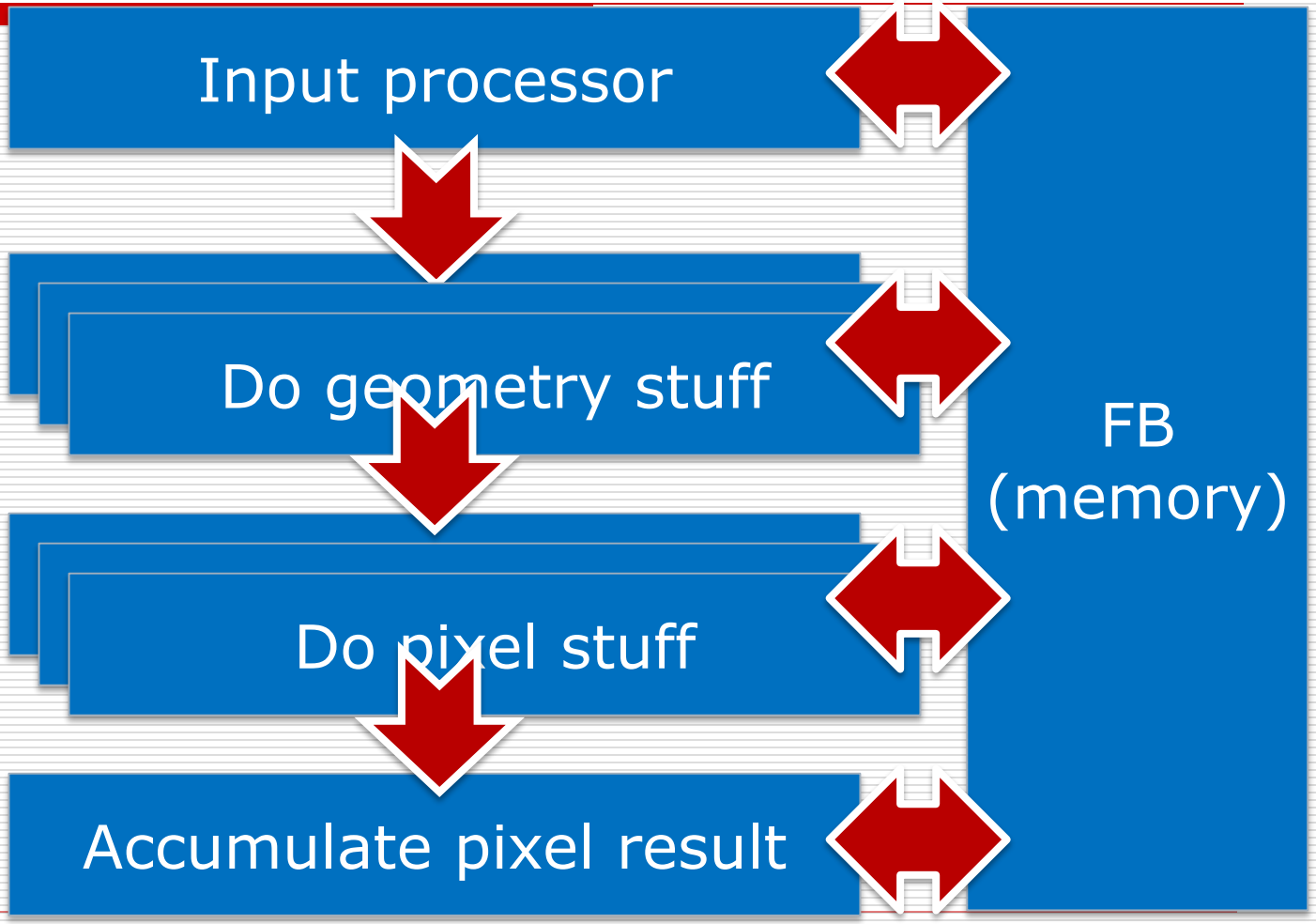
Accumulate pixel result



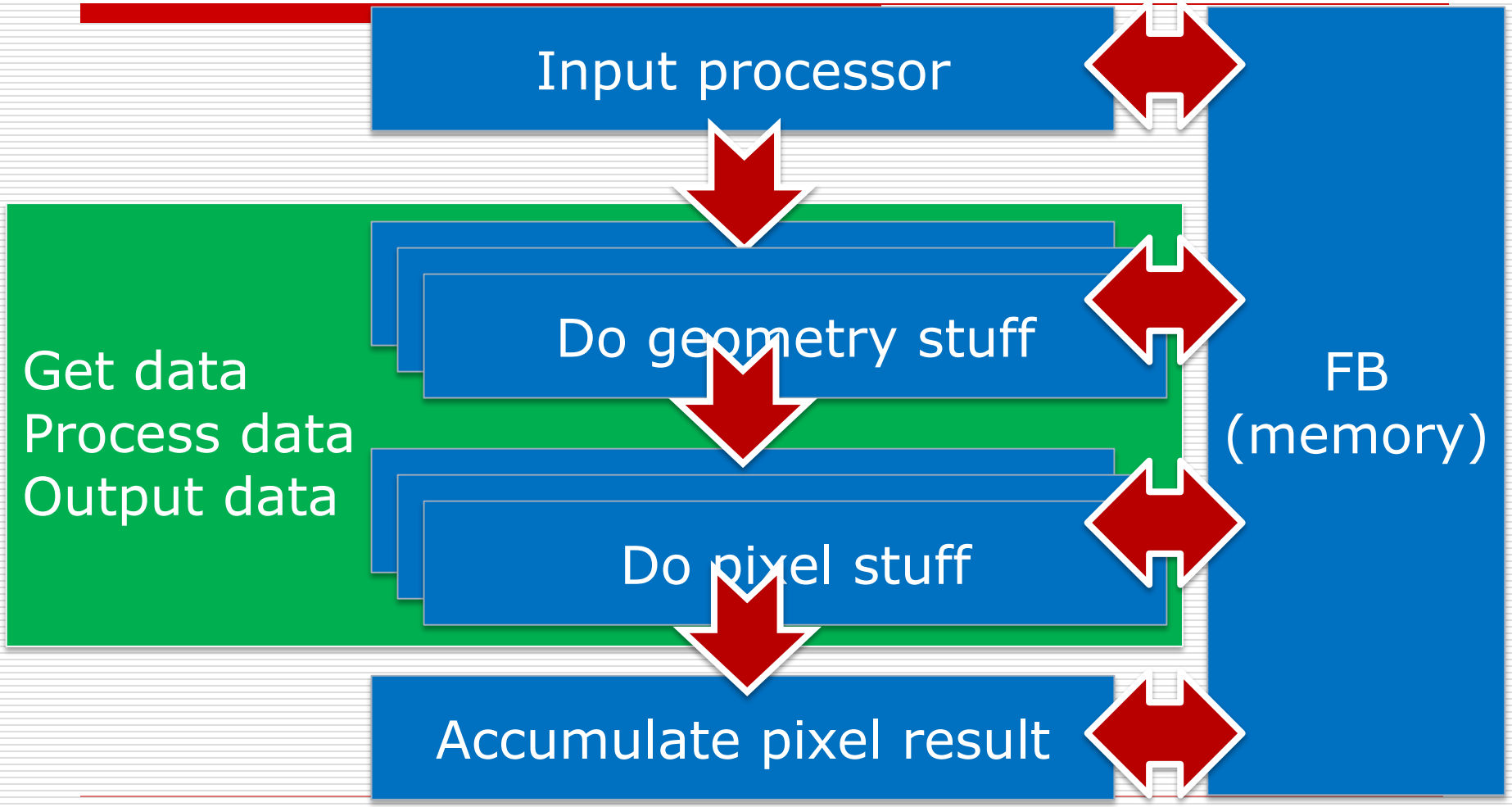
Make it faster



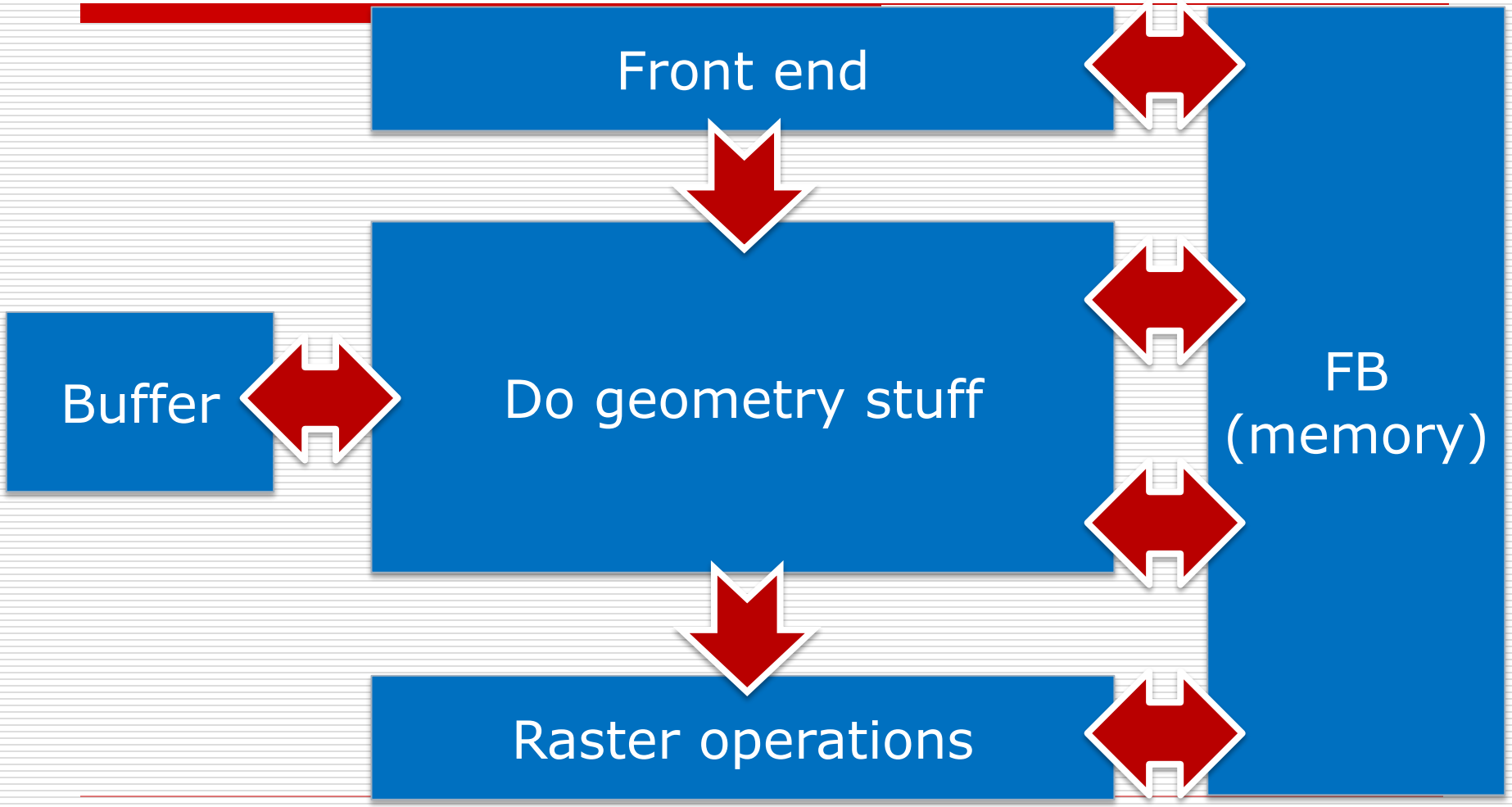
Add Frame Buffer Support



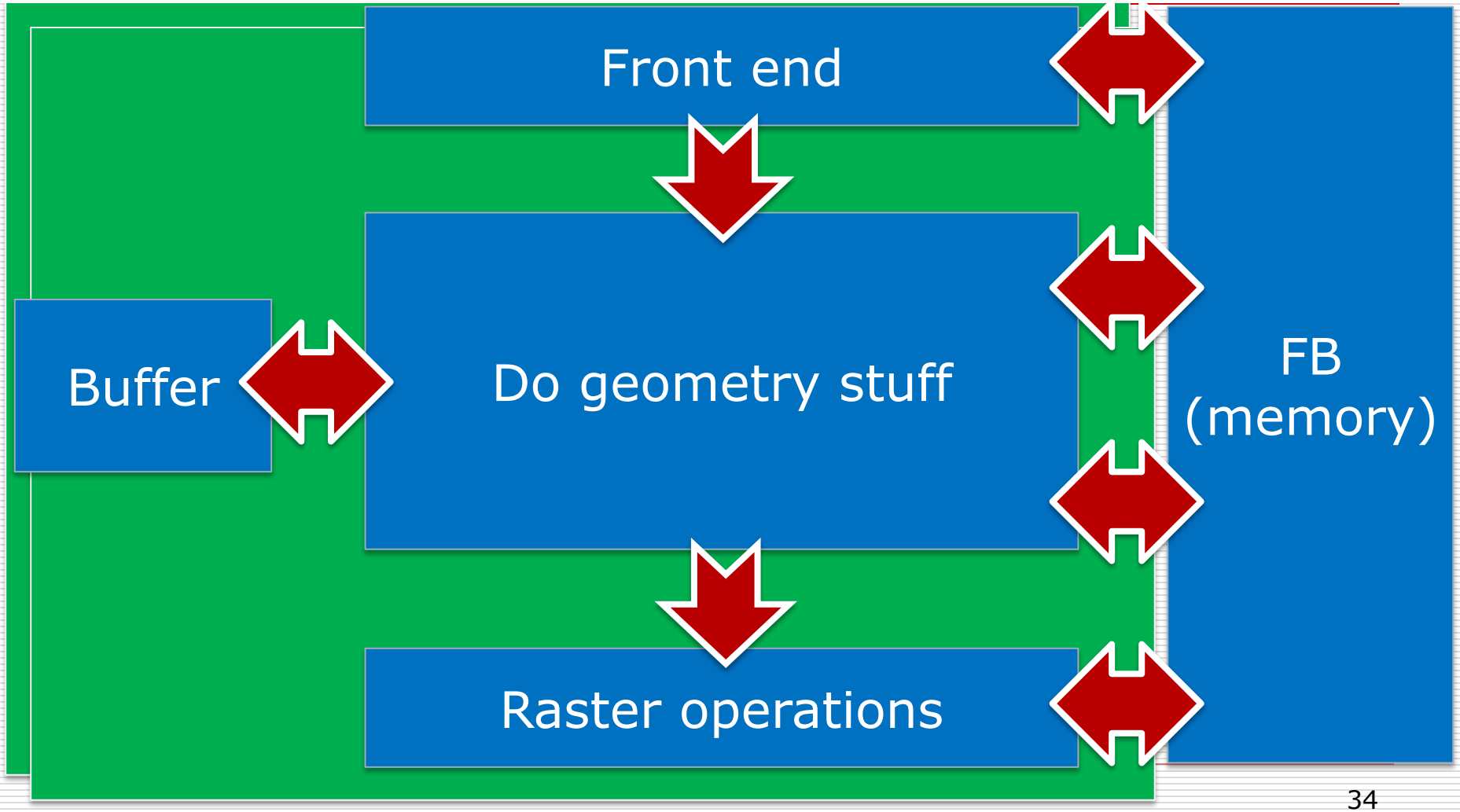
Add Programmability



Uniform Shader



Scaling it up again



GPU Architectures

- Processing is highly data-parallel
  - GPUs are highly multithreaded
  - Use thread switching to hide memory latency
    - Less reliance on multi-level caches
  - Graphics memory is wide and high-bandwidth
- Trend toward general purpose GPUs
  - Heterogeneous CPU/GPU systems
  - CPU for sequential code, GPU for parallel code
- Programming languages/APIs
  - DirectX, OpenGL
  - C for Graphics (Cg), High Level Shader Language (HLSL)
  - Compute Unified Device Architecture (CUDA)