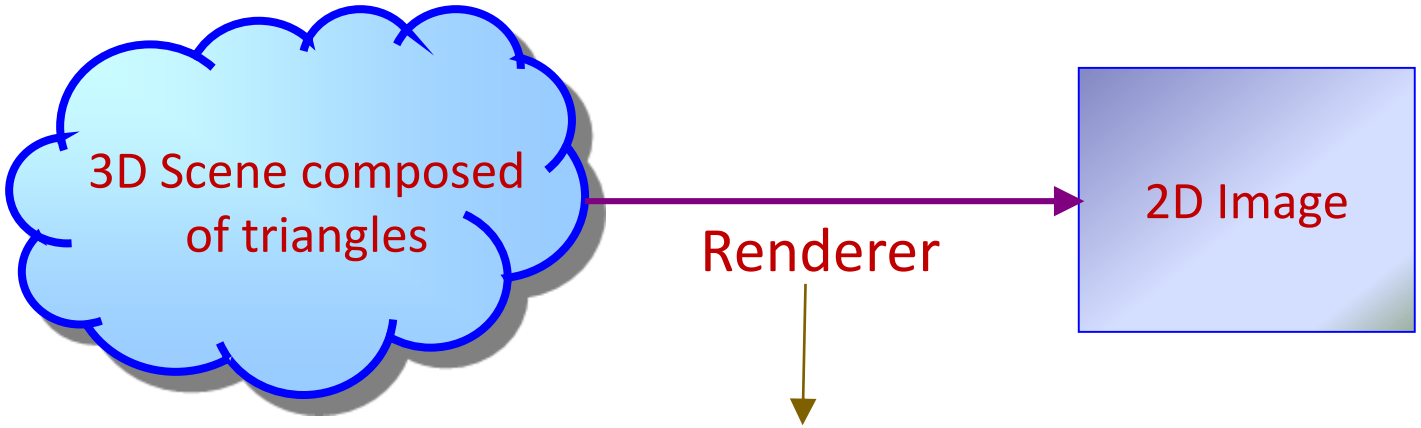
Introduction to Computer Graphics

Introduction to Rendering Pipeline

Prasanna Ghali

Real-Time Rendering

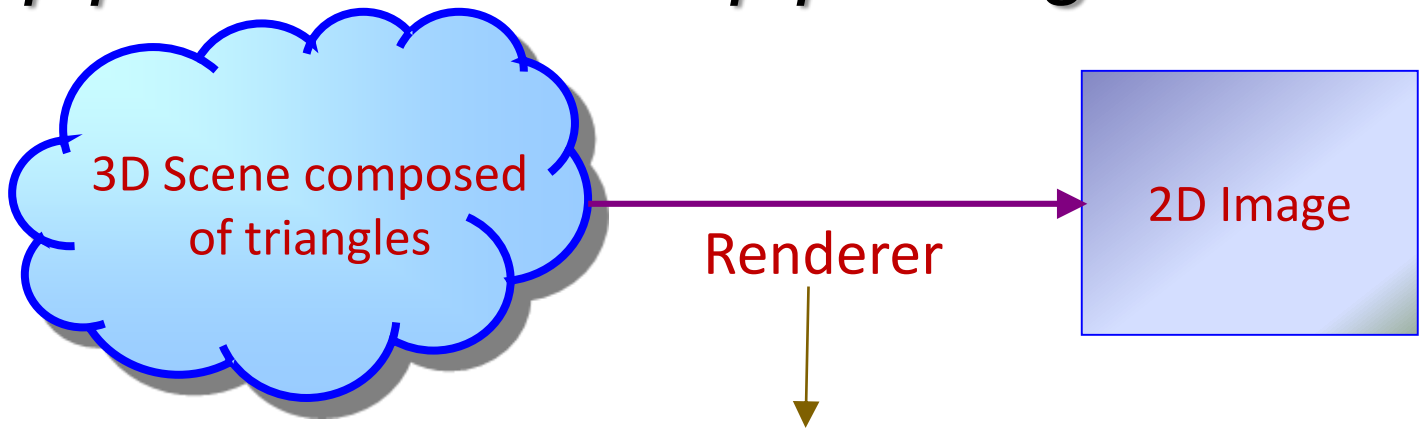
- Computation of 2D images from 3D scene at rate at which user can comfortably interact with scene
- Interactivity rate: ≥ 30 fps



Every frame, real-time renderer *Consumes* triangles to *Produce* corresponding pixels that will collectively form an image in color buffer

Hardware Implementation of Renderer

- Of several techniques available to create image from 3D scene [tessellated into triangles], *pipeline architecture* is ideal to implement real-time renderer in hardware
- What is a *pipeline* or what is *pipelining*?



Every frame, real-time renderer *Consumes* triangles to *Produce* corresponding pixels that will collectively form an image in color buffer

What is a Pipeline?

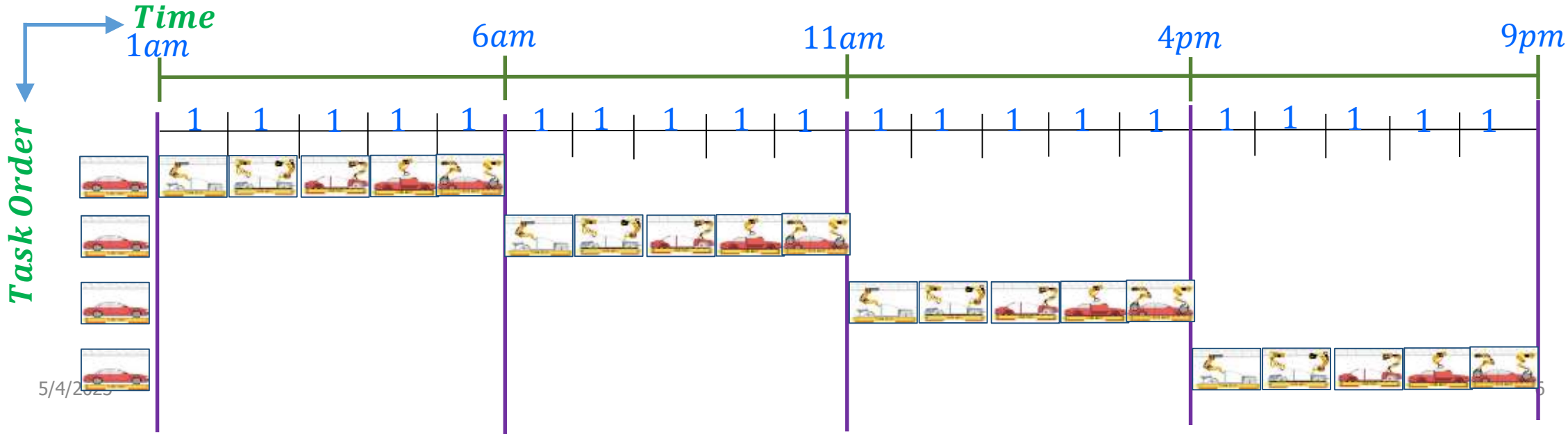
- Pipelining is optimization technique to achieve *temporal parallelism*
  - Concept similar to assembly line where multiple widgets are worked on simultaneously to *speed up* manufacturing of individual widgets
- Subdivide task into sequence of *discrete* subtasks
  - Subtask is executed by specialized hardware *stage*
  - Stages operate *concurrently* to overlap execution of successive tasks
- Assumptions
  - Every stage, by design, takes same amount of time to complete its subtask
  - Pipe is linear, i.e., stage S_j in pipe with stages $\{S_1, S_2, \ldots, S_k\}$ cannot start until all earlier stages $\{S_i \ \forall \ i \leq j\}$ finish

Pipeline Example: Car Manufacturing

- Consider car manufacturing process consisting of five distinct and independent stages [or steps]:
  1. Build chassis [or frame]
  2. Insert engine [into chassis]
  3. Add dashboard, seats, doors, and hood [to frame]
  4. Attach wheels
  5. Paint
- Suppose each stage of car manufacturing process takes 1 hour

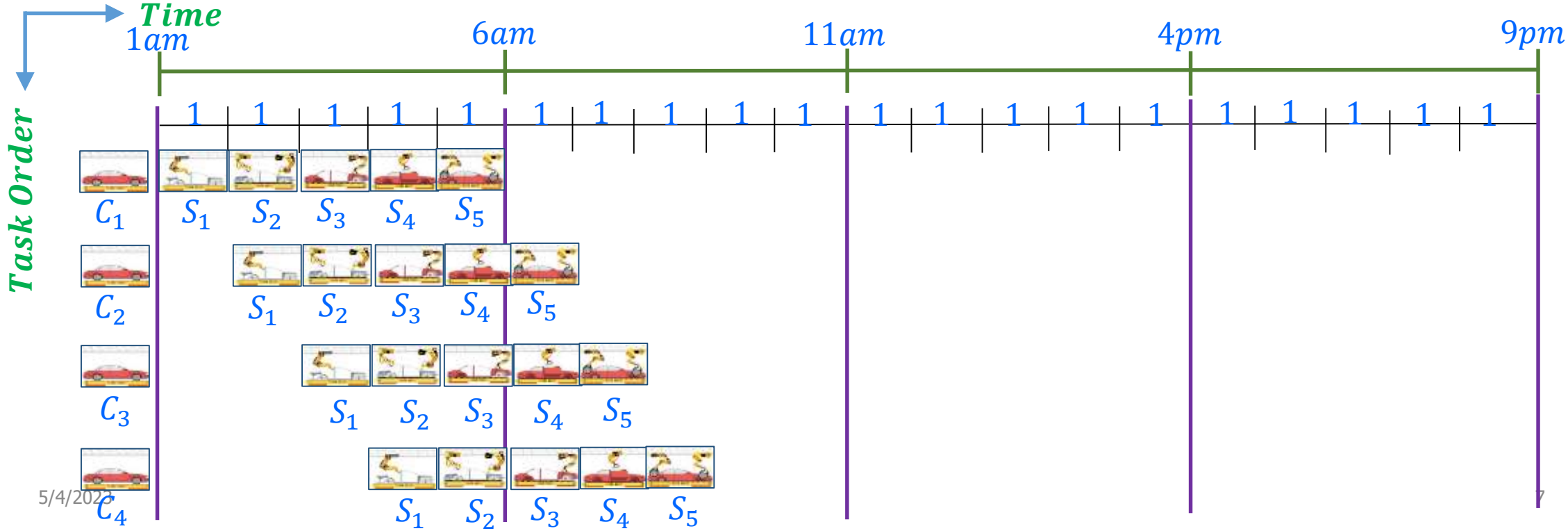
Sequential Car Manufacturing

- *Latency* is time from start to finish to manufacture one car
  - Latency here is 5 hours
- *Throughput* is number of cars manufactured per hour
  - Throughput here is 0.2 cars
- How will pipelining help?



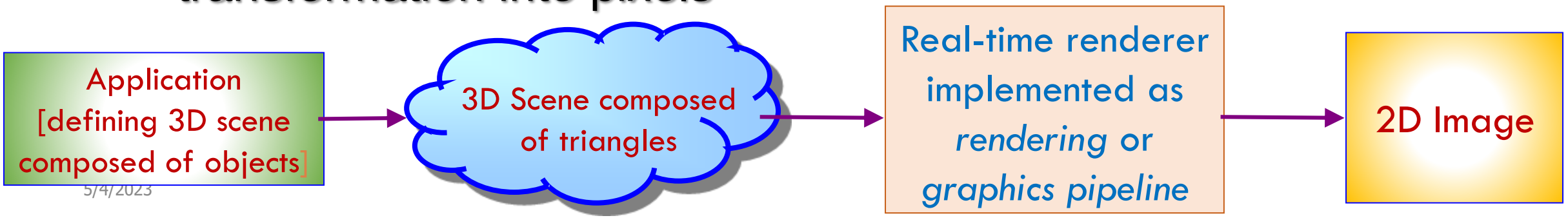
Pipelined Car Manufacturing

- Latency of each manufactured car is still 5 hours!!!
- However, pipelined manufacturing of 4 cars requires only 8 hours!!!
- And, throughput is now 1 car [because new car manufactured every hour]

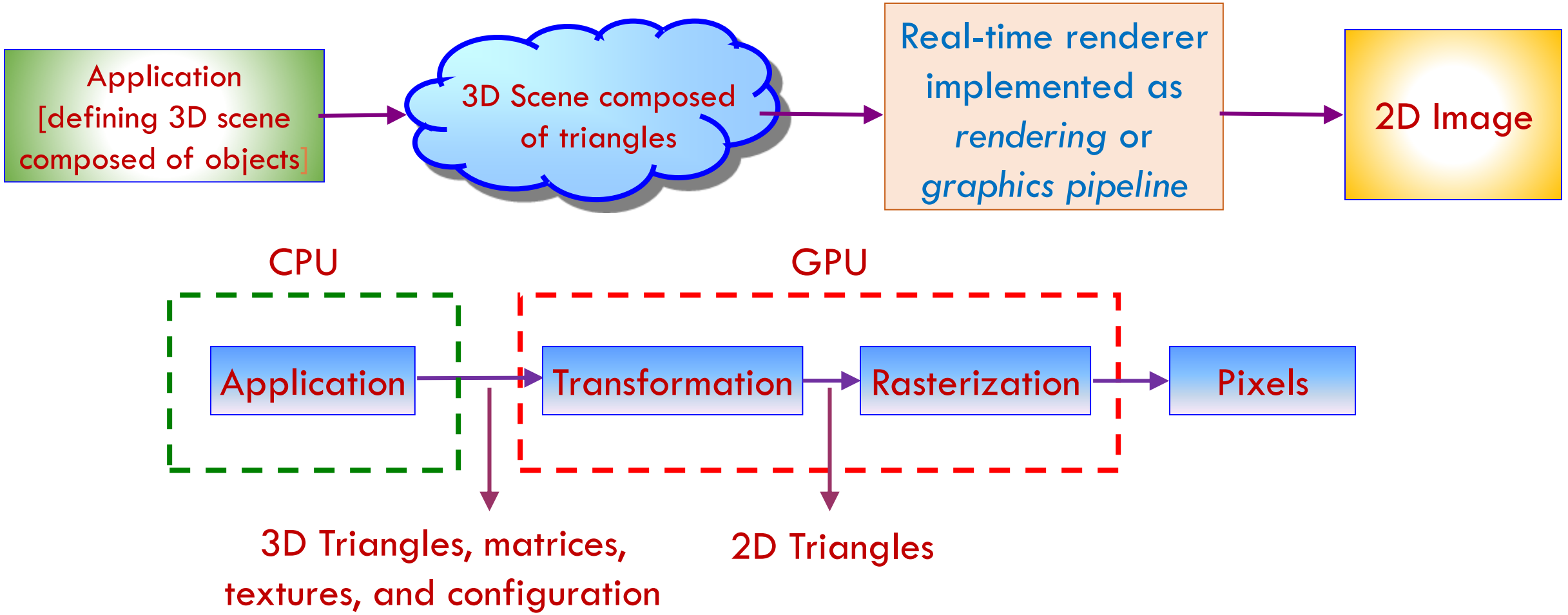


Rendering Pipeline

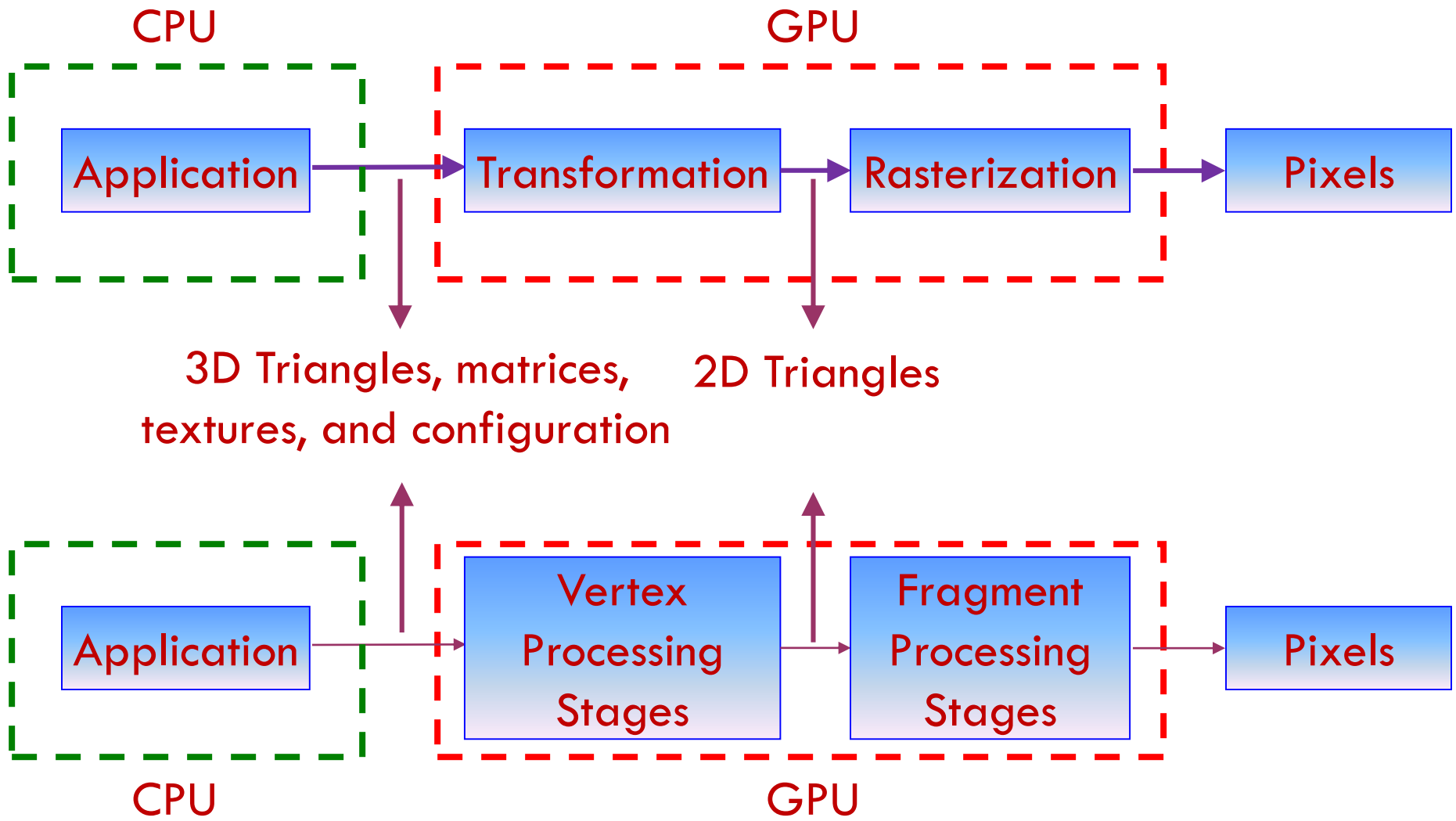
- Real-time rendering – the process of converting many triangles into pixels – is implemented as *rendering pipeline*
  - Efficiently maps consumer-producer model of renderer to hardware
  - Several hardware units perform distinct operations *concurrently* on different triangles at different stages of their transformation into pixels



Fixed Function Rendering Pipeline (1/2)



Fixed Function Rendering Pipeline (2/2)



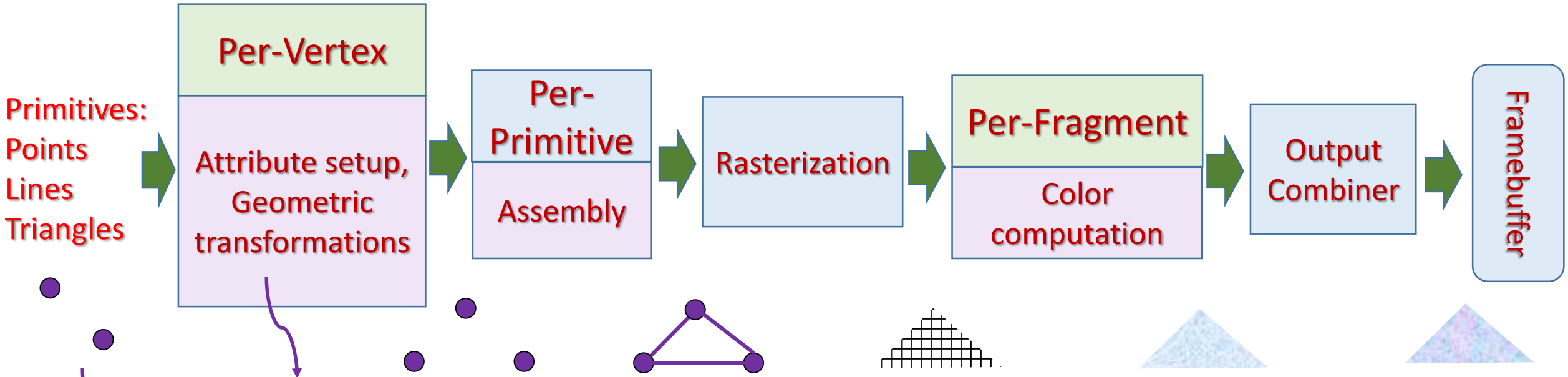
More Elaborate Definition of Pixel

- Set of data that is about to be written to output buffer(s):
  - *Viewport* coordinates [which location in output buffer does pixel correspond to?]
  - Appearance information to color buffer [RGB components]
  - Depth value to depth buffer [for hidden-surface removal to generate correct images and shadow map to generate shadows]
  - Appearance information to texture map [e.g., used in racing game to create image from rear-view camera which can then be used as texture map to be applied on rear-view mirror object]

What is a Fragment?

- Many triangles in 3D scene overlap or occlude other triangles causing some rasterized pixel values to be discarded [i.e., not written to color buffer]
  - Therefore, need to distinguish between pixels that will be written to color buffer and pixels that will be discarded
- We'll relabel rasterized pixels as fragments [potential pixels] and use term pixel to mean corresponding fragment which is determined to be visible
- Fragment will consist of viewport coordinates [where on screen?] plus *interpolants* [values that'll be used to compute color, depth, …]

Programmable Pipeline: More Details



requires matrices

using a triangle
as an example

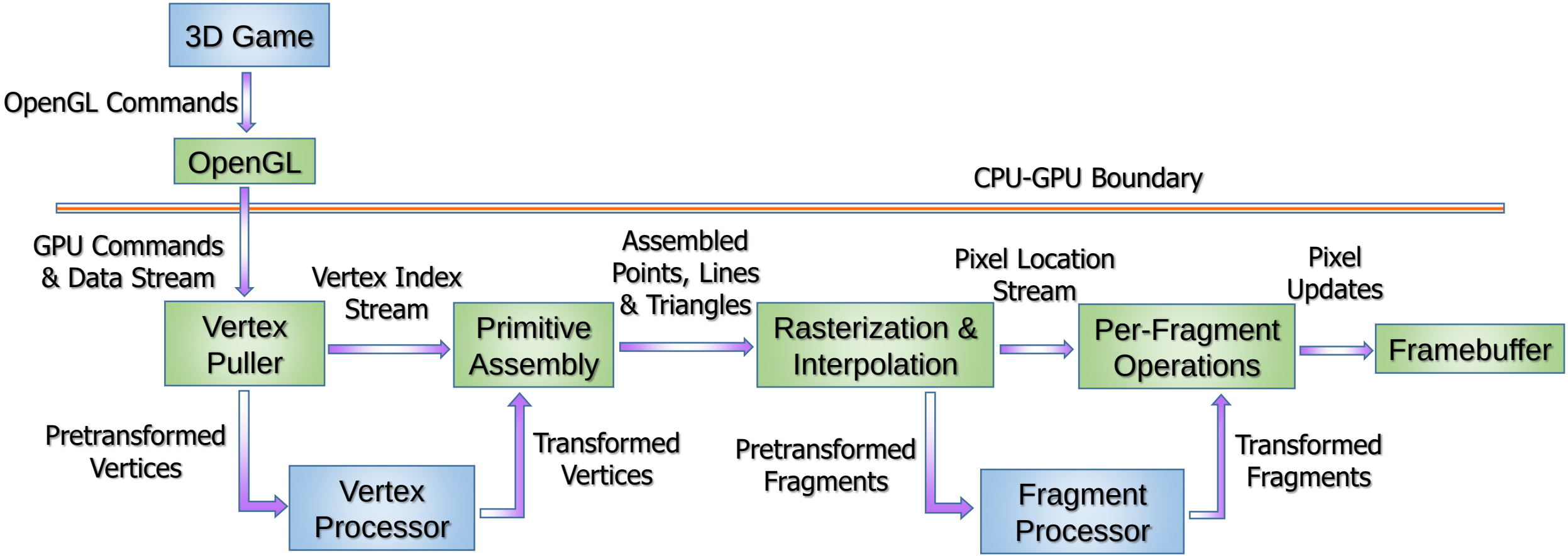
Position coordinates: (x, y, z)

Color components: (r, g, b)

Texture coordinates: (s, t)

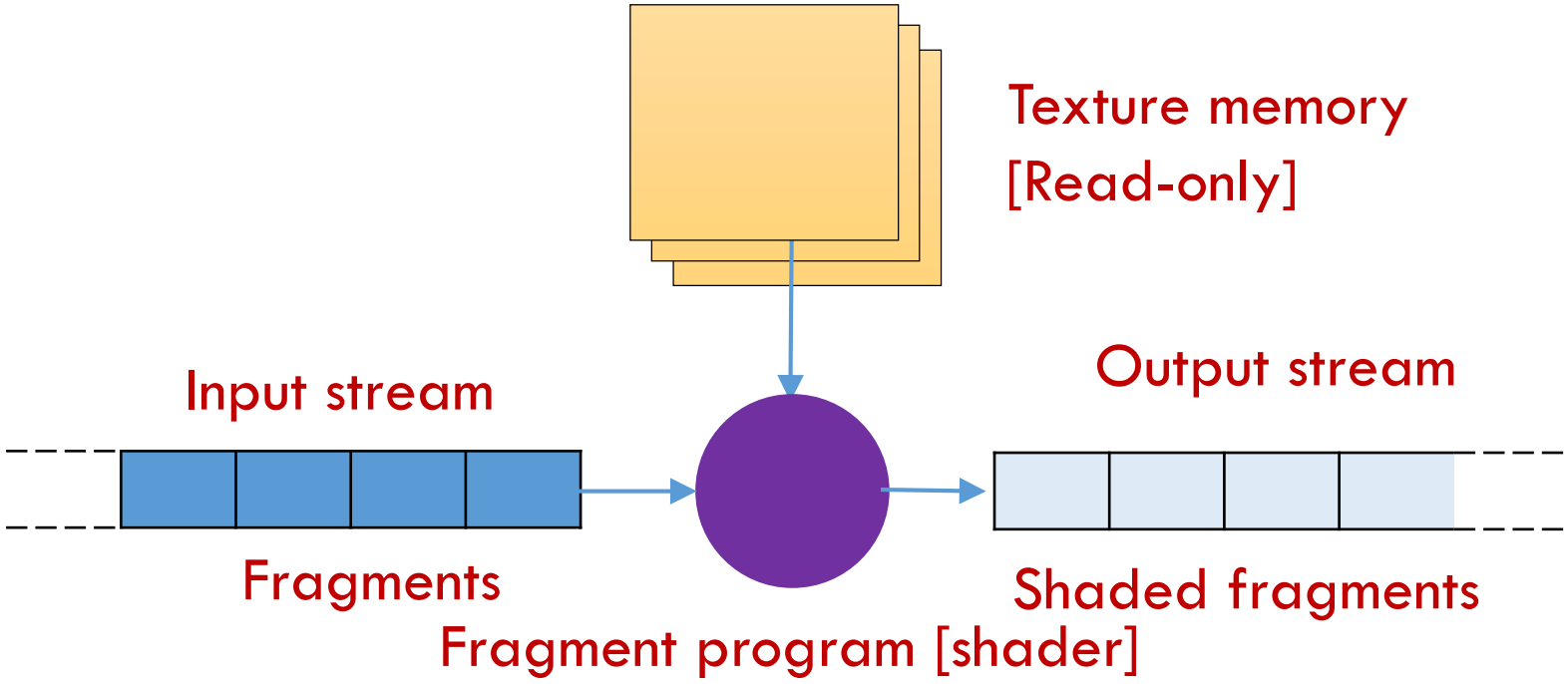
} Per-vertex attributes

Our Simplified Programmable Pipe



Shader

Shader is written to operate on a single vertex or fragment and is executed for every vertex or fragment

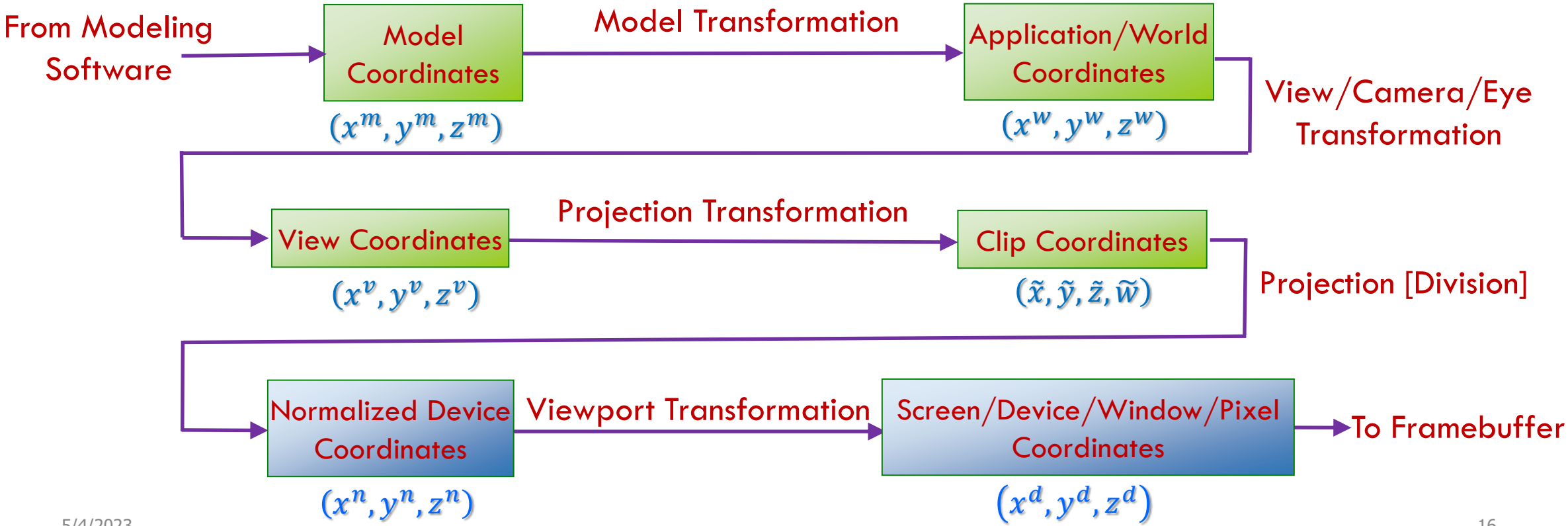


Fragment program as a stream processor

[Single Program Multiple Data processor]

3D Rendering Pipe: Coordinate Systems

- Another view of rendering pipeline: different coordinate systems that 3D objects [and their primitives] must transition thro' in rendering pipeline



2D Rendering Pipe: Coordinate Systems

- Rendering pipeline for 2D applications: different coordinate systems that 2D objects [and their primitives] must transition thro' in rendering pipeline