Predicting Performance

Where to begin?

You're done writing code, now what?

Does it work?

Is it fast?

What does fast mean?

60x faster than the CPU is pretty good

- What are you leaving on the table?
- How close is it to theoretical?

Predicting Performance

It is very useful to predict theoretical performance when working on shaders

Spreadsheets are quite useful for this

- Compute theoretical performance
- Compute pixels per clock, etc
- Easy to see how close an implementation is to peak performance

Quite easy with CAL/Brook+ since you can get the ISA even if you use a high-level language

Breaking down GPGPU performance

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

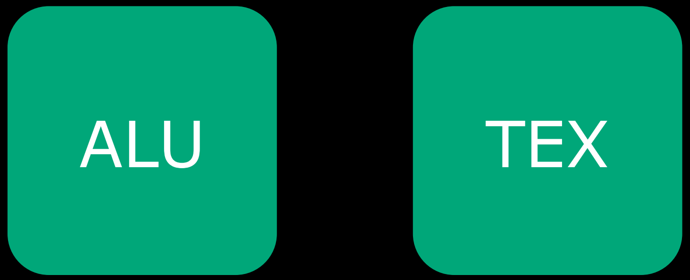
Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

ALU

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering



Breaking down GPGPU performance

AMD Smarter Choice

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

ALU TEX MEM

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Breaking down GPGPU performance

AMD Smarter Choice

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Theoretical performance can be calculated

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Theoretical performance can be calculated

What we know…

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Theoretical performance can be calculated

What we know…

# of ALU, TEX, and Write operations

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Theoretical performance can be calculated

What we know…

# of ALU, TEX, and Write operations
- GPUShaderAnalyzer great for this

Breaking down GPGPU performance

GPGPU applications generally progress through the hardware in a predictable fashion, unlike rendering

Theoretical performance can be calculated

What we know…

# of ALU, TEX, and Write operations

- GPUShaderAnalyzer great for this

What do we need to find out?

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 ALU Shader

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 ALU Shader

$$\frac{(\#pixels)x(\#ALU\ instructions)}{(ALU/clk)x(3D\ engine\ speed)}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 ALU Shader

$$\frac{(\#pixels)x(\#ALU\ instructions)}{(ALU/clk)x(3D\ engine\ speed)} = \frac{(1920x1088)x(1)}{(48)x(625Mhz)}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 ALU Shader

$$\frac{(\#pixels)x(\#ALU\ instructions)}{(ALU/clk)x(3D\ engine\ speed)} = \frac{(1920x1088)x(1)}{(48)x(625Mhz)}$$

$$= 0.07ms$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 TEX Shader

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 TEX Shader

$$\frac{(\#pixels)x(\#TEX\ instructions)}{(TEX/clk)x(3D\ engine\ speed}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 TEX Shader

$$\frac{(\#pixels)x(\#TEX\ instructions)}{(TEX/clk)x(3D\ engine\ speed} = \frac{(1920x1088)x(1)}{(16)x(625Mhz)}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- 1 TEX Shader

$$\frac{(\#pixels)x(\#TEX\ instructions)}{(TEX/clk)x(3D\ engine\ speed} = \frac{(1920x1088)x(1)}{(16)x(625Mhz)}$$

$$= \quad 0.21ms$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- Memory Performance - 1 Byte in 1 Byte out (Copy)

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- Memory Performance - 1 Byte in 1 Byte out (Copy)

$$\frac{(\#pixels)x(in + out\ bits\ per\ pixel)}{(bus)x(memory\ speed)}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- Memory Performance - 1 Byte in 1 Byte out (Copy)

$$\frac{(\#pixels)x(in + out\ bits\ per\ pixel)}{(bus)x(memory\ speed)}$$

$$= \frac{(1920x1088)x(16\ bits)}{(256)x(750\ Mhz\ x\ 2DDR)}$$

Calculating Theoretical Performance

Radeon X1900XT (R580)

- 8:48:16:16 (VER:ALU:TEX:ROP)
- 256 bit memory bus
- 625 MHz Engine / 750 Mhz Memory Clocks

Example:

- Memory Performance - 1 Byte in 1 Byte out (Copy)

$$\frac{(\#pixels)x(in + out\ bits\ per\ pixel)}{(bus)x(memory\ speed)}$$

$$= \frac{(1920x1088)x(16\ bits)}{(256)x(750\ Mhz\ x\ 2DDR)}$$

$$= 0.085\ ms$$

Calculating Theoretical Performance

Calculating Theoretical Performance

Overall Theoretical Performance

- max(ALU,TEX,Memory)
  - each operation happens in parallel
- max(0.07, 0.21, 0.085)
- 0.21 ms - Texture bound
  - Texture units is the limitation

Calculating Theoretical Performance

Overall Theoretical Performance

- max(ALU,TEX,Memory)
  - each operation happens in parallel
- max(0.07, 0.21, 0.085)
- 0.21 ms - Texture bound
  - Texture units is the limitation

Remember, this is only a starting point!

- ALU and TEX calculation is reasonable
  - Actually usually very close
- Memory assumes peak
  - depends on access pattern, etc
- Conditional operations can complicate things

Decoder Ring

GPU	Engine Clock	Memory Clock	Memory Width	ALUs	TEX	ROPs
R600 HD2900	750	850	512	64	16	16
R610 HD2600	700	1100	64	8	4	4
R630 HD2400	800	1100	128	24	8	4
R670 HD3870	750	1150	256	64	16	16

Decoder Ring

GPU	Engine Clock	Memory Clock	Memory Width	ALUs	TEX	ROPs
R600 HD2900	750	850	512	64	16	16
R610 HD2600	700	1100	64	8	4	4
R630 HD2400	800	1100	128	24	8	4
R670 HD3870	750	1150	256	64	16	16

How long would a 1 ALU shader that outputs 1 byte take on R610? What's the bottleneck?

Decoder Ring

GPU	Engine Clock	Memory Clock	Memory Width	ALUs	TEX	ROPs
R600 HD2900	750	850	512	64	16	16
R610 HD2600	700	1100	64	8	4	4
R630 HD2400	800	1100	128	24	8	4
R670 HD3870	750	1150	256	64	16	16

Decoder Ring

GPU	Engine Clock	Memory Clock	Memory Width	ALUs	TEX	ROPs
R600 HD2900	750	850	512	64	16	16
R610 HD2600	700	1100	64	8	4	4
R630 HD2400	800	1100	128	24	8	4
R670 HD3870	750	1150	256	64	16	16

How long would a 2 ALU shader that outputs 1 byte take on R610? What's the bottleneck?

AMD Smarter Choice

The Value of System / Remote Memory

Radial Correction

System Requirements:

- Capture video from one or more cameras.
- Transfer images to GPU
- Convert bayer-pattern images to RGB images
- Remove lens distortions
- Return to host for further processing

System needs to use limited power

- Mobile GPU

Want to minimize correction time

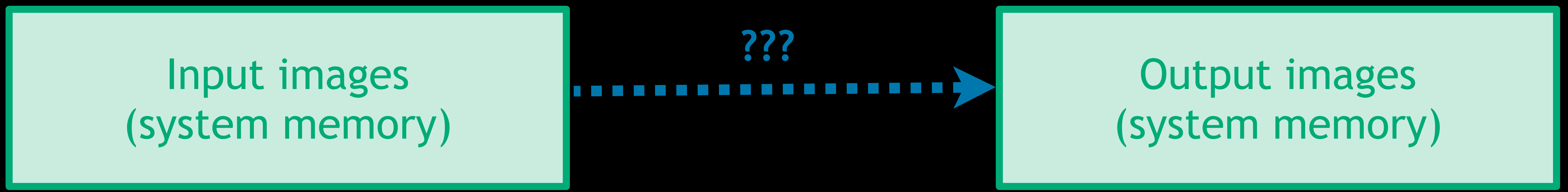
- Images further processed in a large real-time system

Naive Implementation

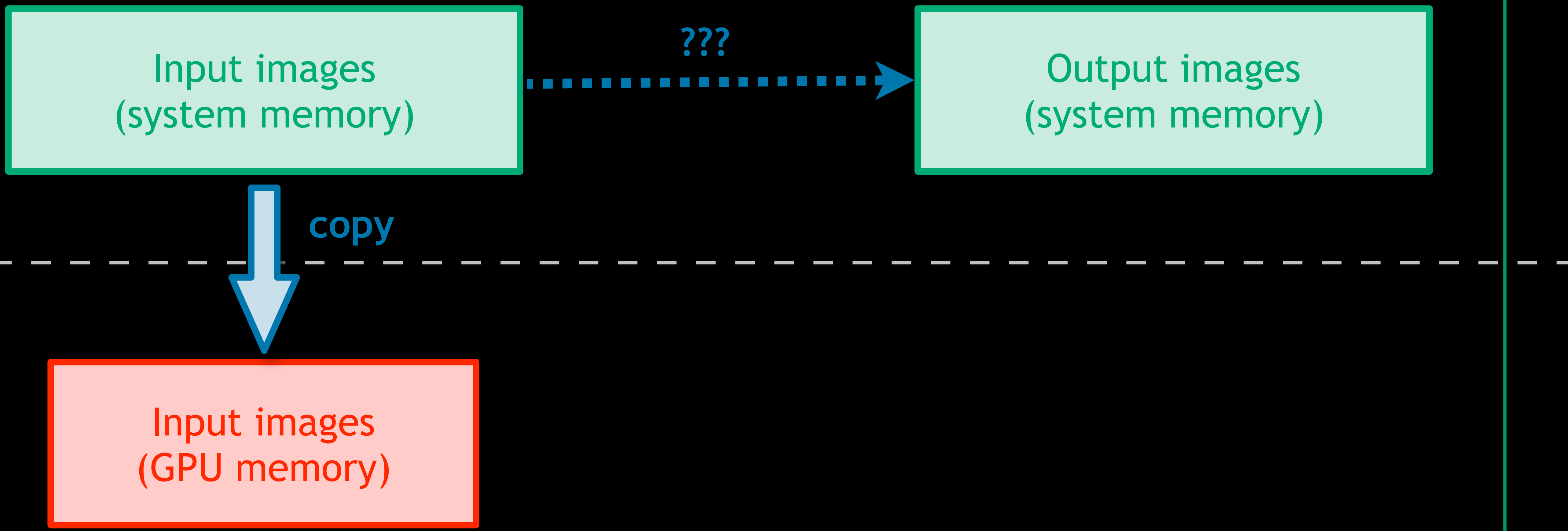
Input images
(system memory)

Output images
(system memory)

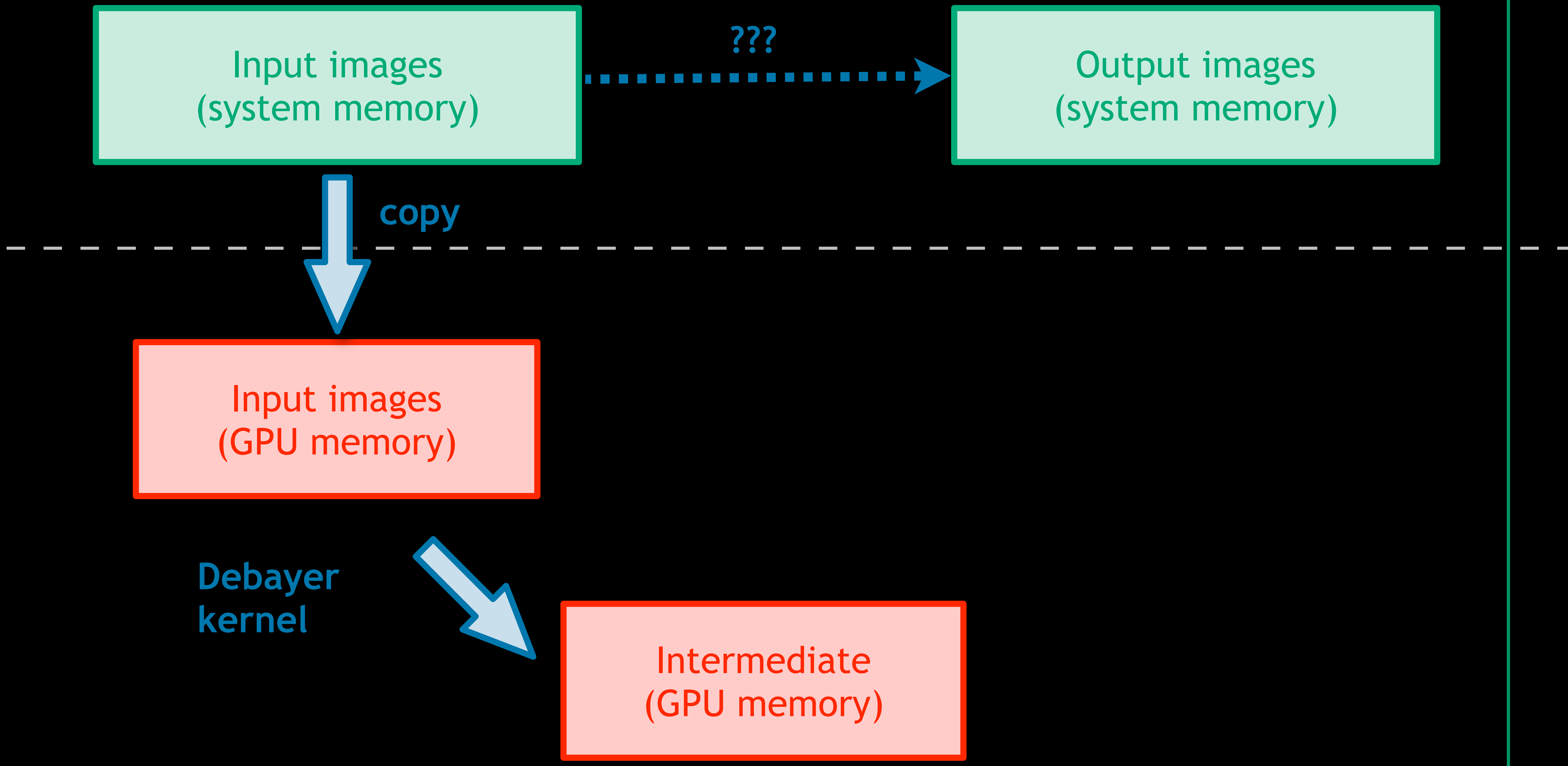
Naive Implementation



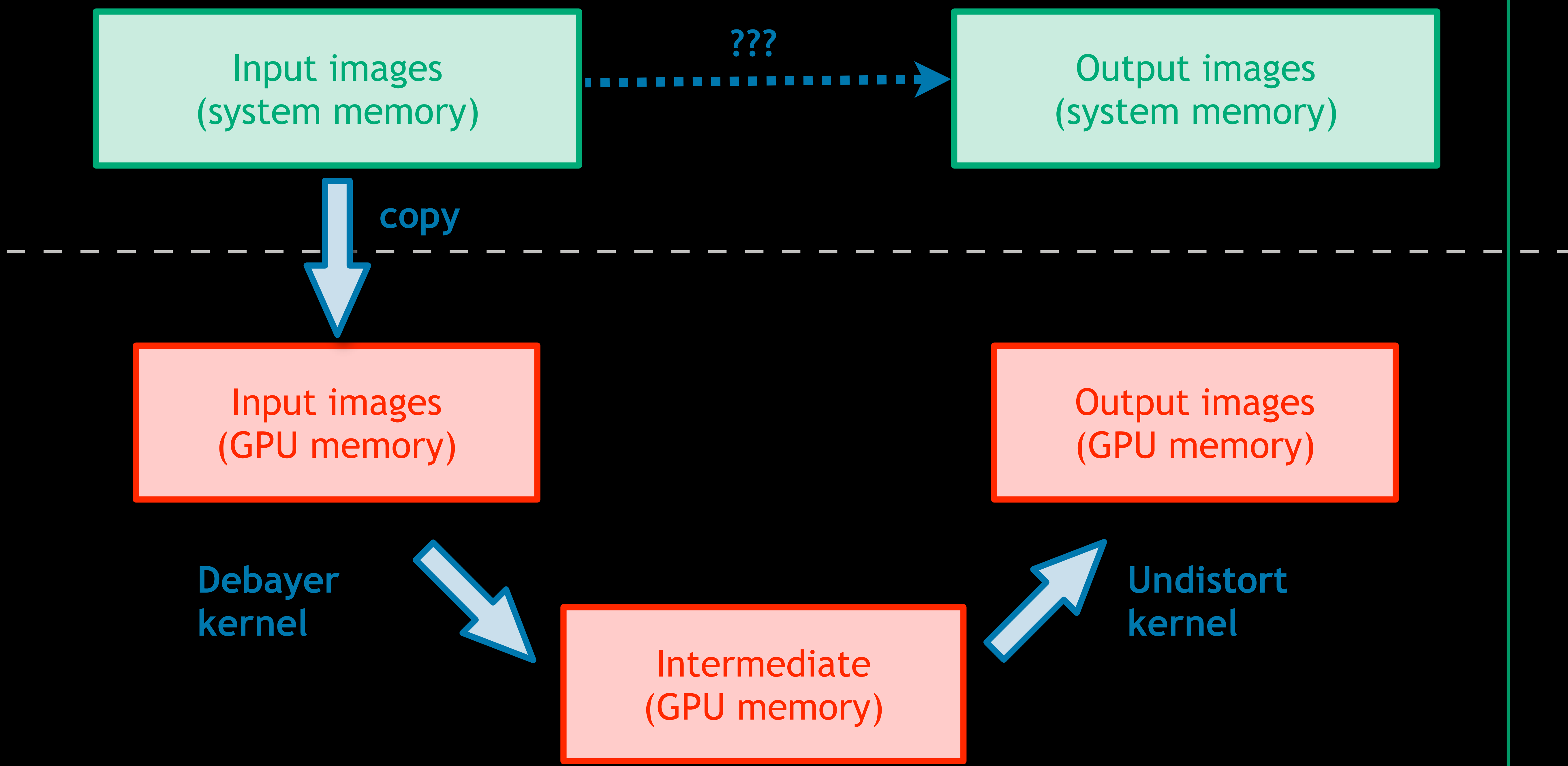
Naive Implementation



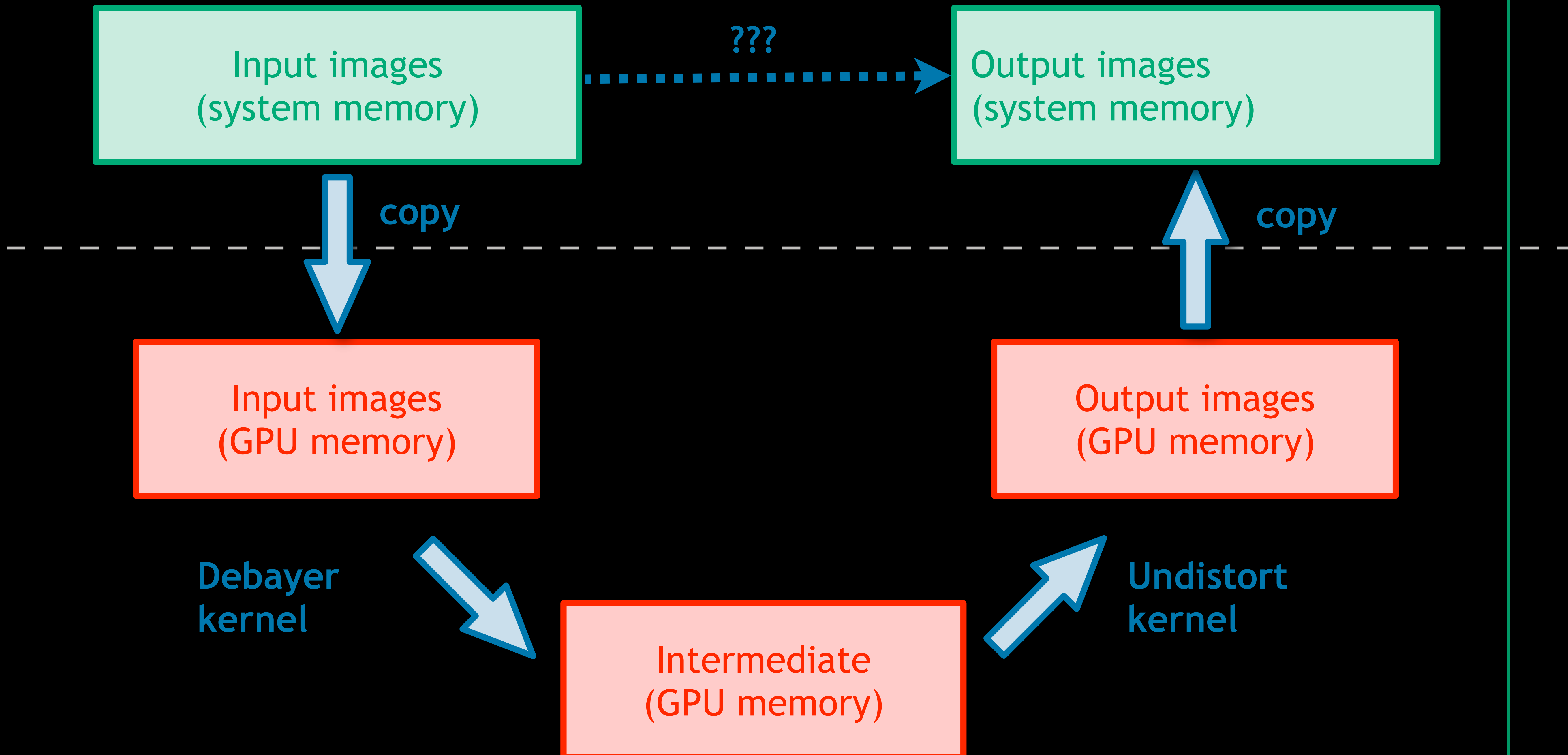
Naive Implementation



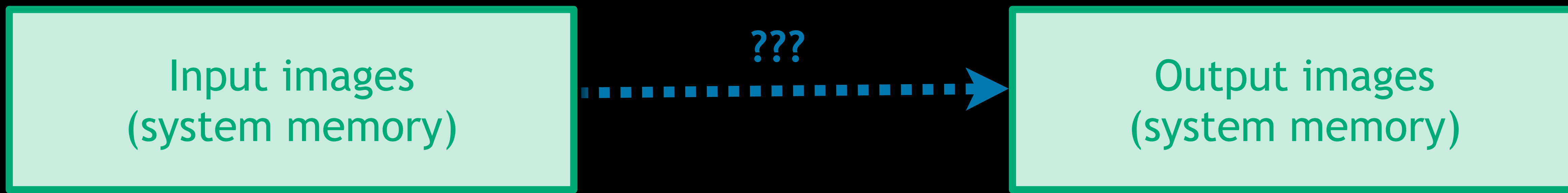
Naive Implementation



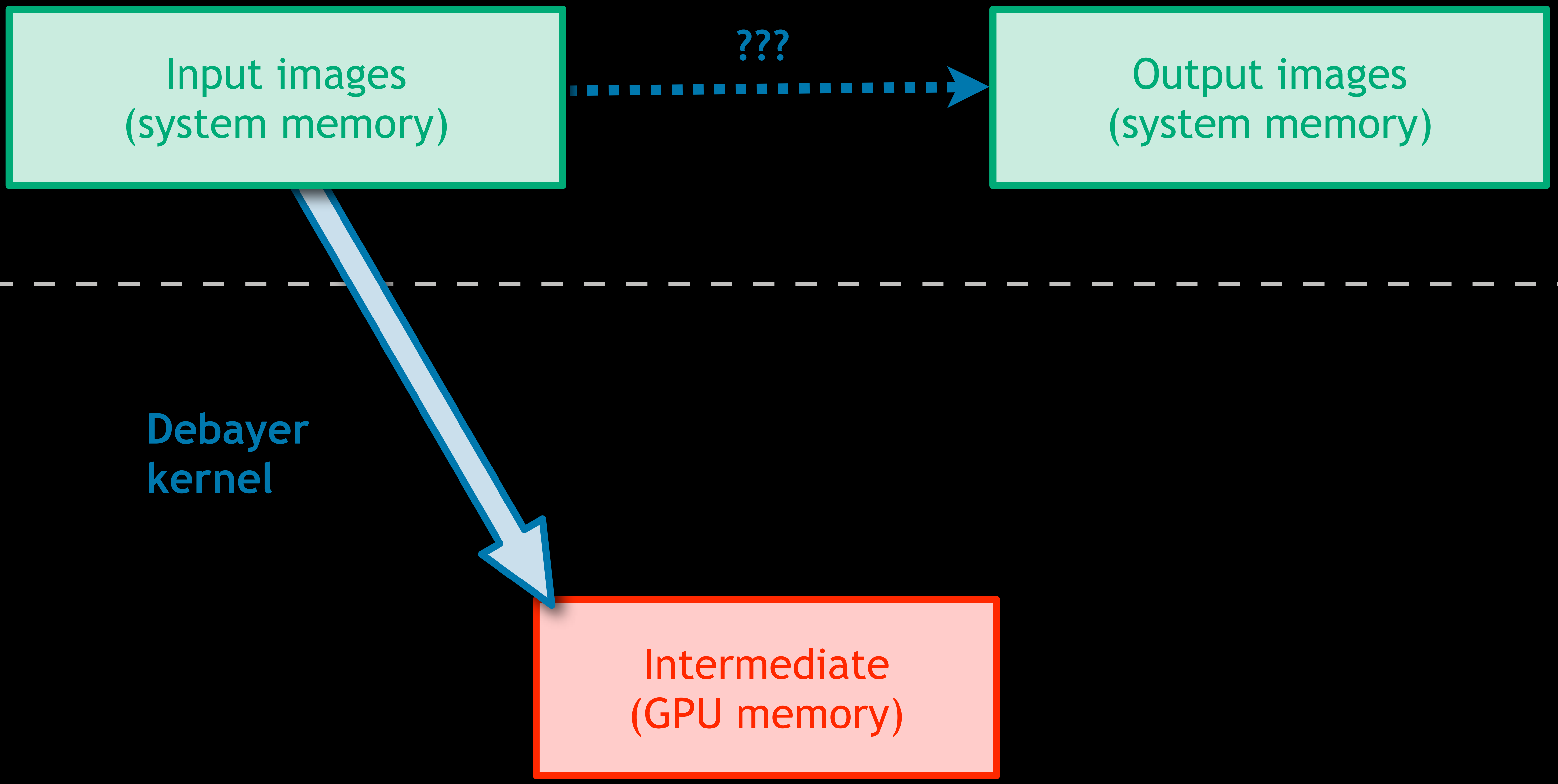
Naive Implementation



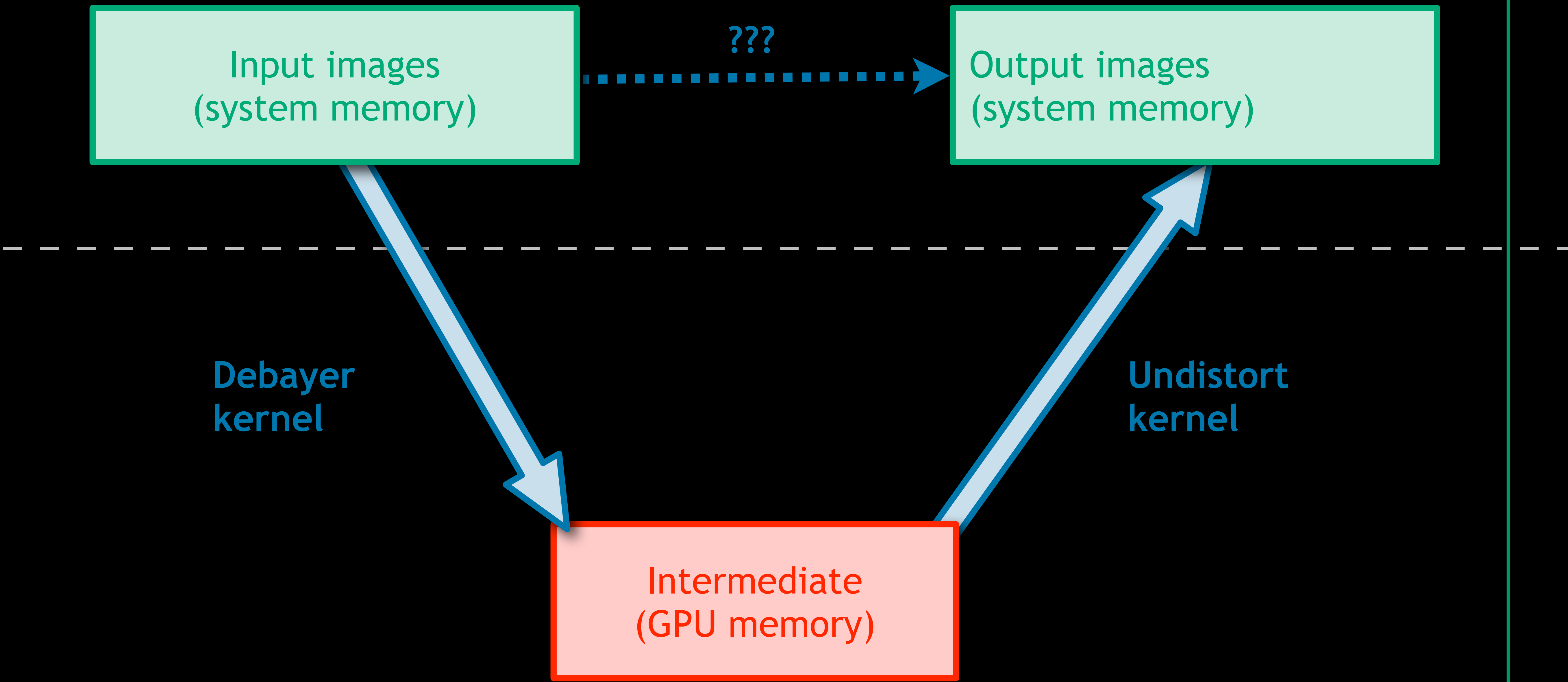
Faster Implementation



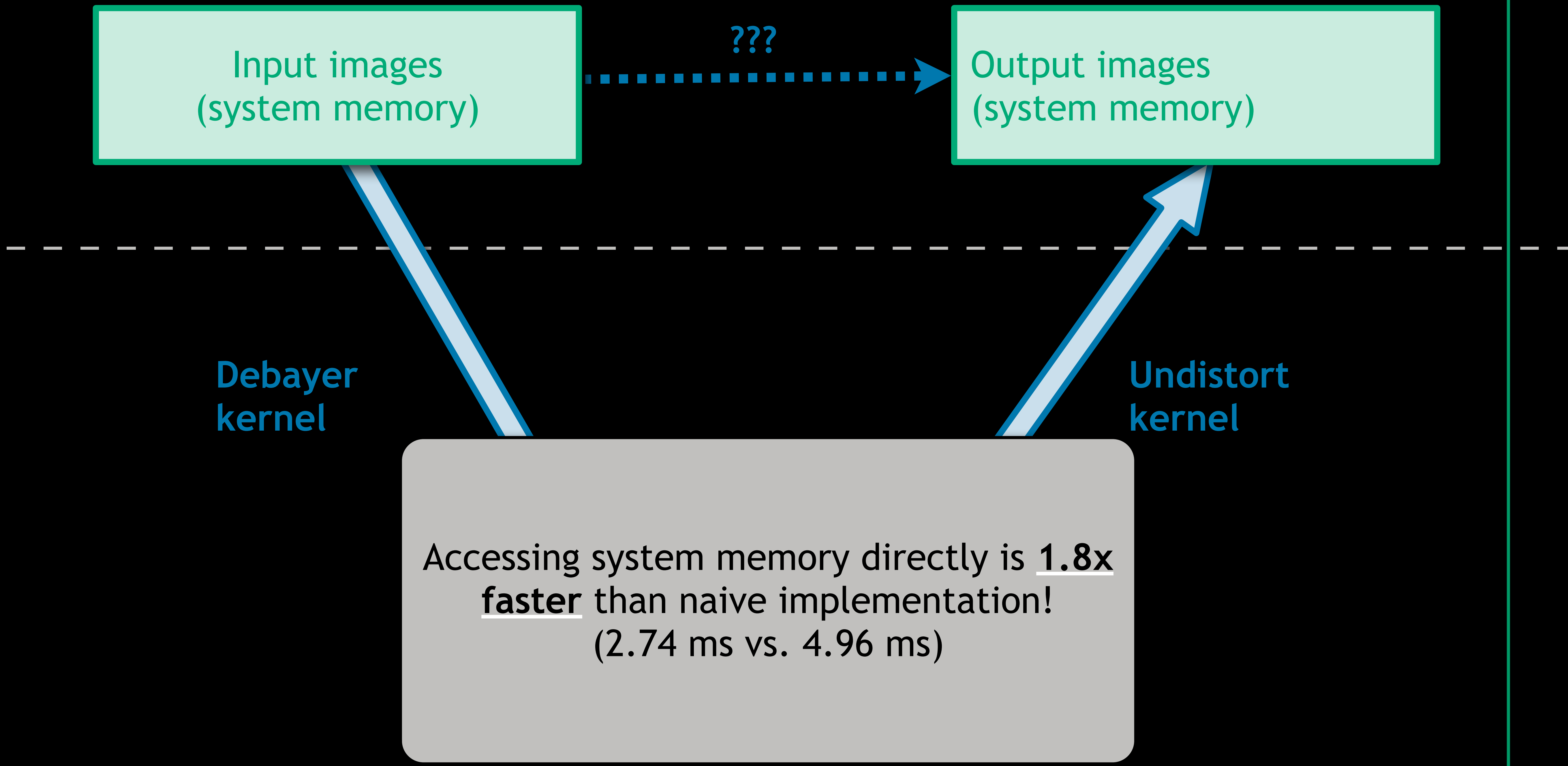
Faster Implementation



Faster Implementation



Faster Implementation



Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics

Caveat: System Memory

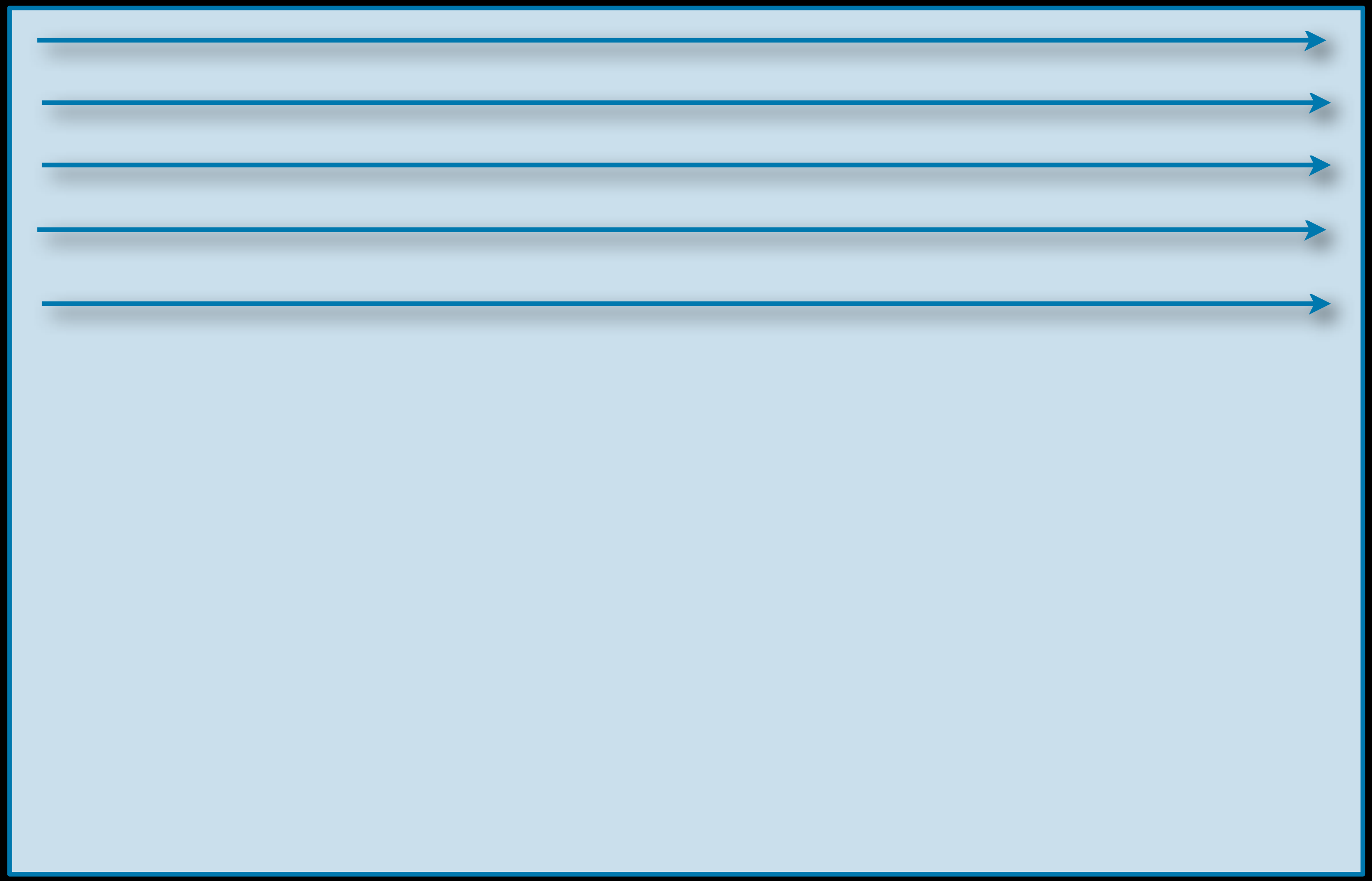
Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics

Caveat: System Memory

Performance is chipset dependent

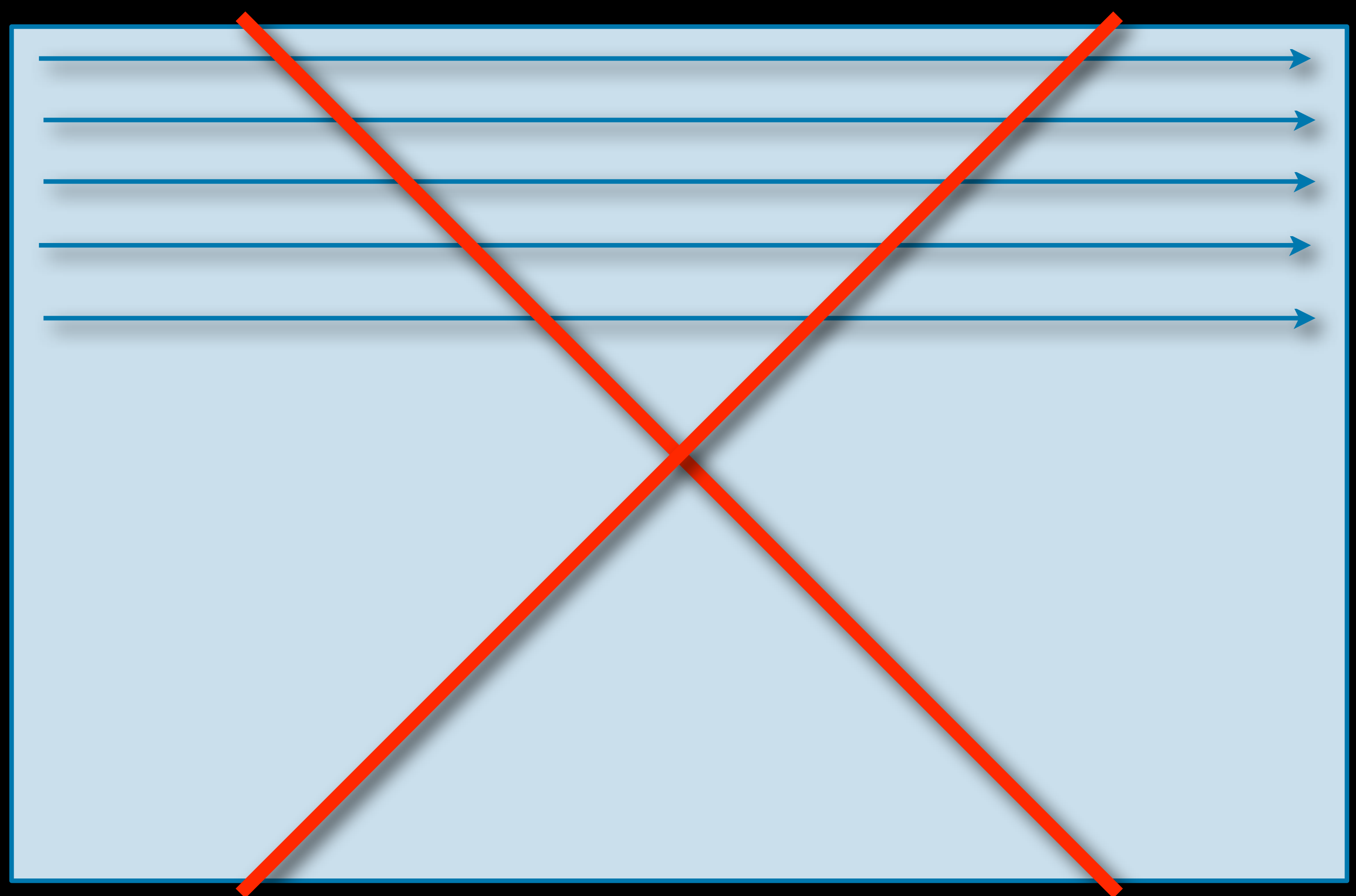
Rasterizers optimized for texture cache performance when rendering graphics



Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics



Caveat: System Memory

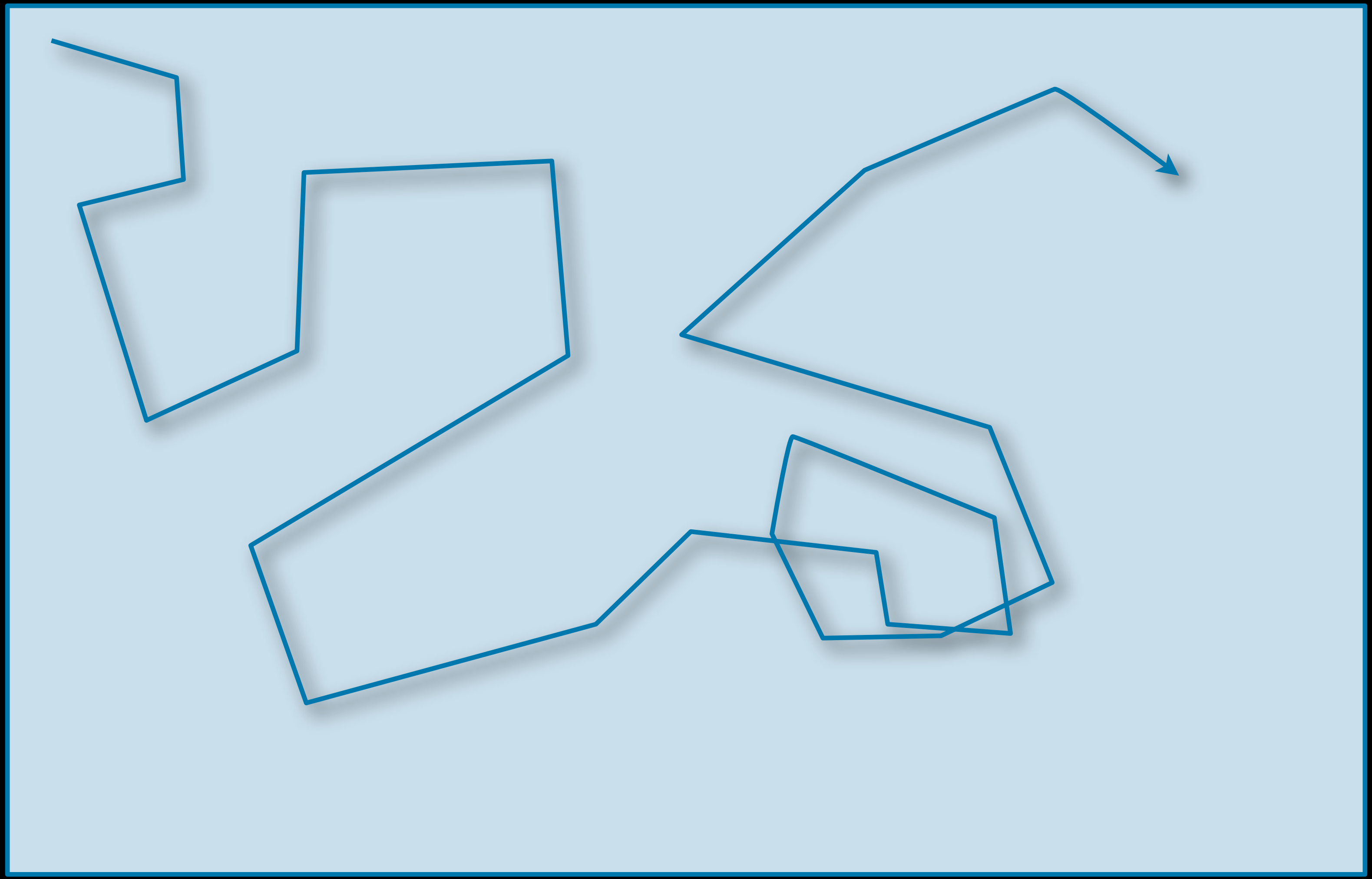
Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics

Caveat: System Memory

Performance is chipset dependent

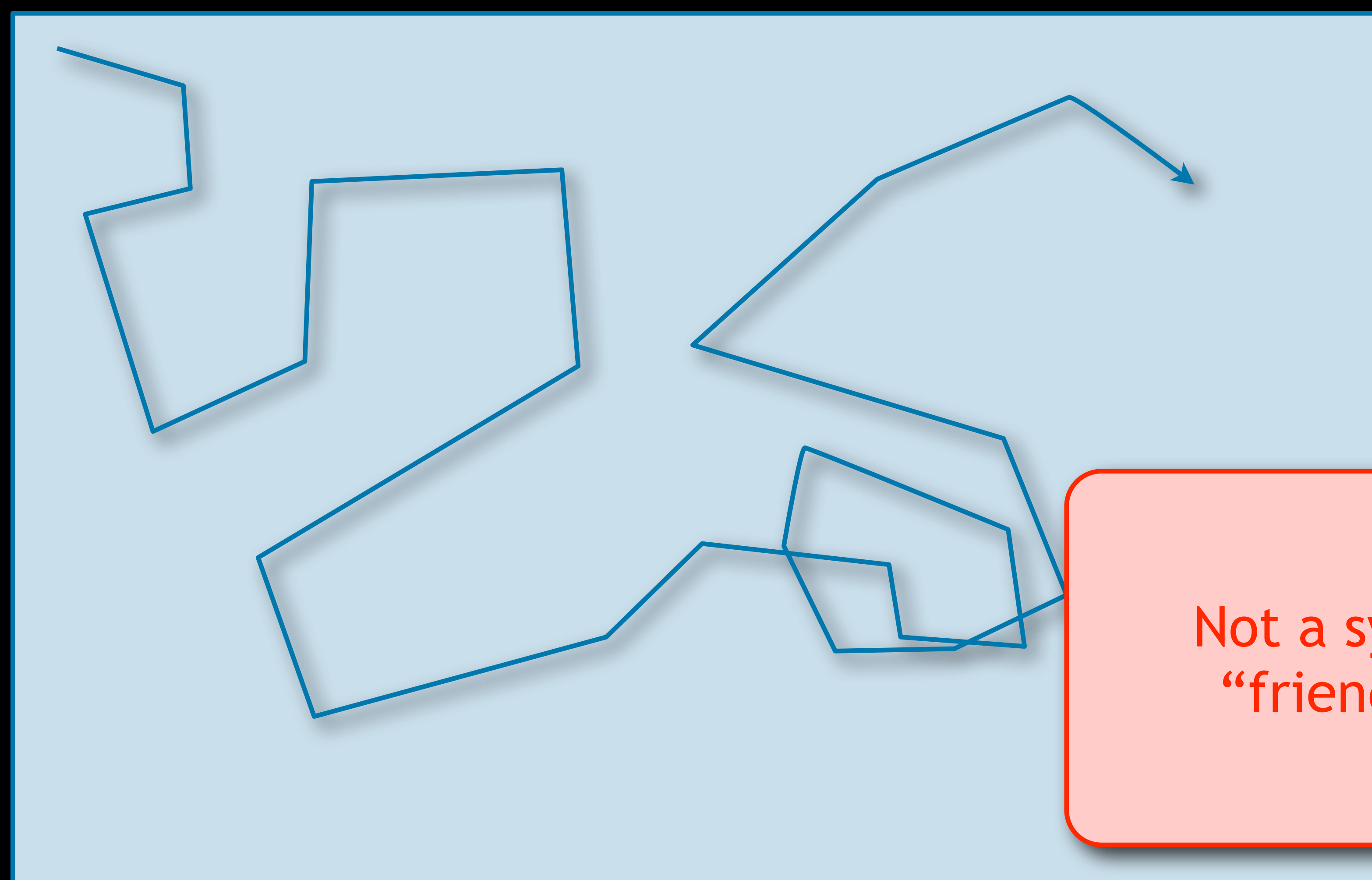
Rasterizers optimized for texture cache performance when rendering graphics



Caveat: System Memory

Performance is chipset dependent

Rasterizers optimized for texture cache performance when rendering graphics



Not a system memory "friendly" traversal

“Forcing” Raster pattern

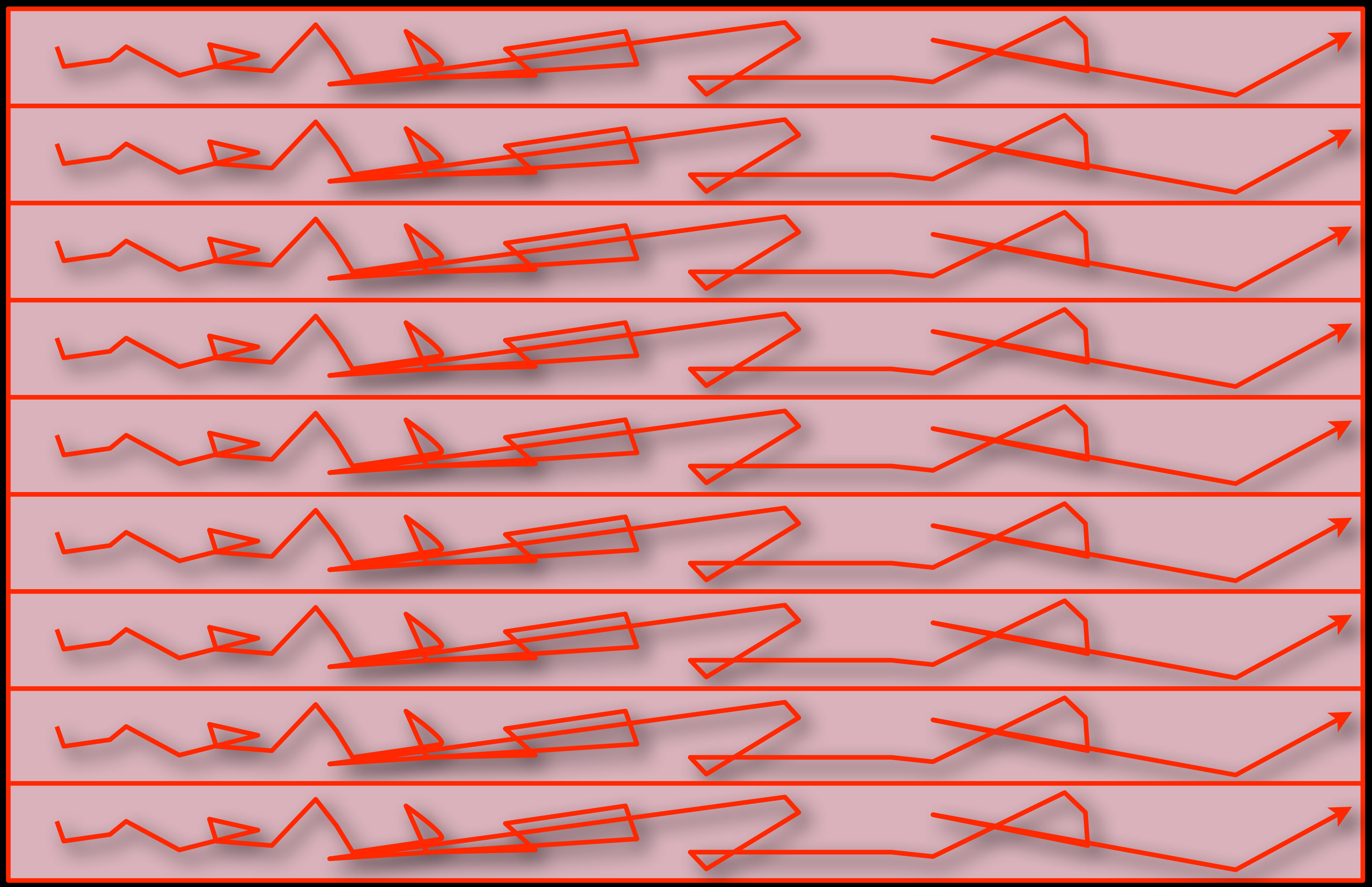
“Force” the rasterizer to be more friendly with system memory traversal

- Use strips of geometry (CTM can directly *stamp* quads)

"Forcing" Raster pattern

"Force" the rasterizer to be more friendly with system memory traversal

- Use strips of geometry (CTM can directly *stamp* quads)



Effect of "Forcing" Raster pattern

AMD
Smarter Choice

2048x2048 float32x4 "copy" shader

- Reads input in local GPU memory, writes to system memory
- RD580 with an R580

Full screen quad time = 45.51 ms

- ~1.5 GB/sec *readback*

"Raster-Blocks" time = 26.53 ms

- ~2.5 GB/sec *readback*

Technique could also be used to optimize shaders with non-standard to local memory accesses

Need to be aware of how threads are assigned to wavefronts / warps / vectors

Effect of “Forcing” Raster pattern

2048x2048 float32x4 “copy” shader

- Reads input in local GPU memory, writes to system memory
- RD580 with an R580

Full screen quad time = 45.51 ms

- ~1.5 GB/sec *readback*

“Raster-Blocks” time = 26.53 ms

- ~2.5 GB/sec *readback*

1.7x faster

Technique could also be used to optimize shaders with non-standard to local memory accesses

Need to be aware of how threads are assigned to wavefronts / warps / vectors