GPU Architectures

UCSD CSE 272
Advanced Image Synthesis
Tzu-Mao Li

slides heavily borrowed from Kayvon Fatahalian
https://graphics.stanford.edu/~kayvonf/

The power wall

- Clock rates of CPUs have linear relationship with power consumption
  - reducing voltage helps power consumption, but it's already extremely low these days

$$Power \propto 1/2 \times Capacitive\ load \times Voltage^2 \times Frequency\ switched$$

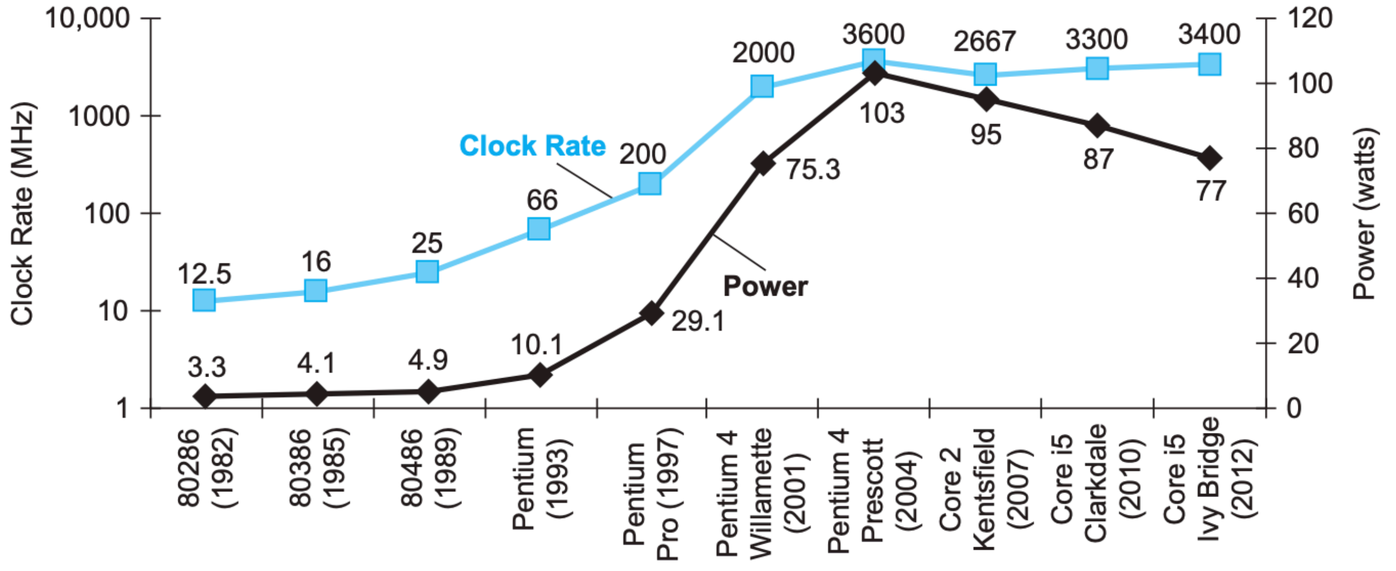


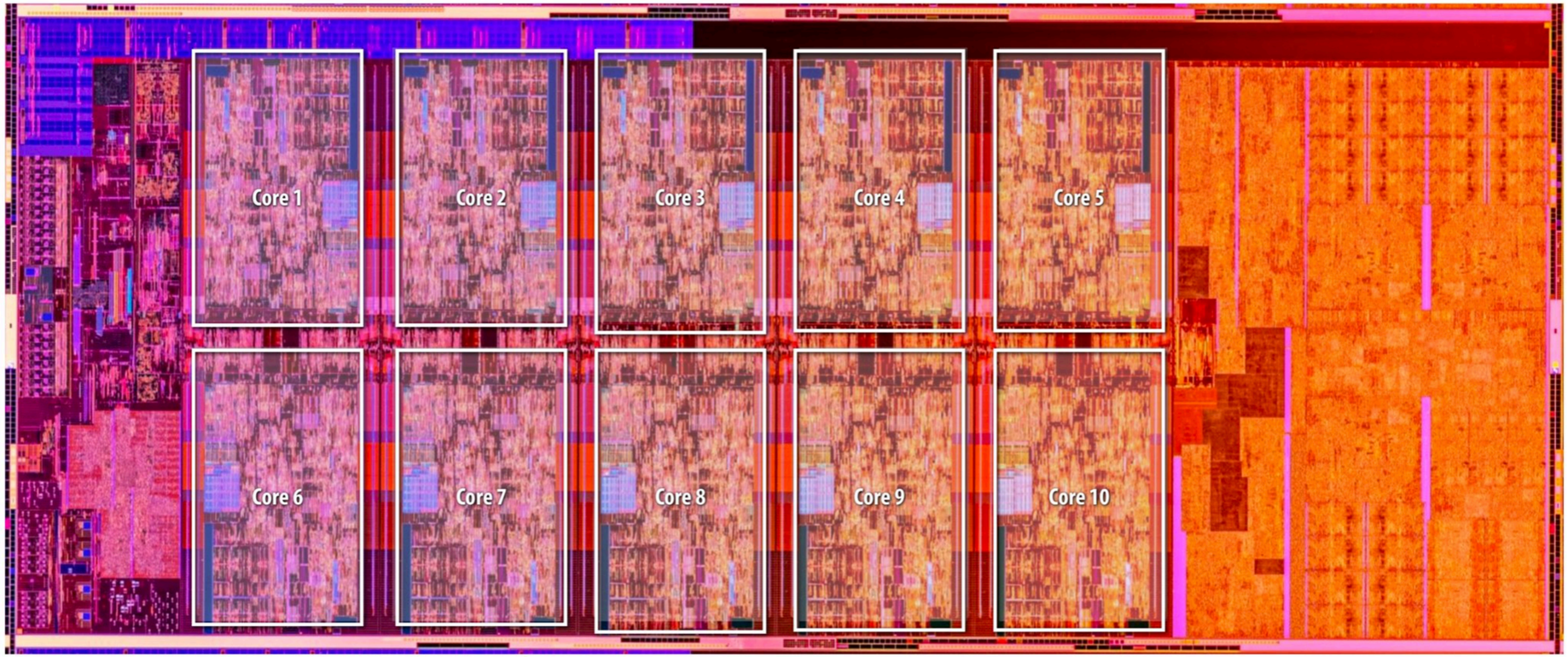
FIGURE 1.16 Clock rate and Power for Intel x86 microprocessors over eight generations and 25 years. The Pentium 4 made a dramatic jump in clock rate and power but less so in performance. The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip. The Core i5 pipelines follow in its footsteps.

from Patternson & Hennessy, "Computer Organization and Design: the hardware/software interface"

Parallelism to the rescue

Example: multi-core CPU

Intel "Comet Lake" 10th Generation Core i9 10-core CPU (2020)



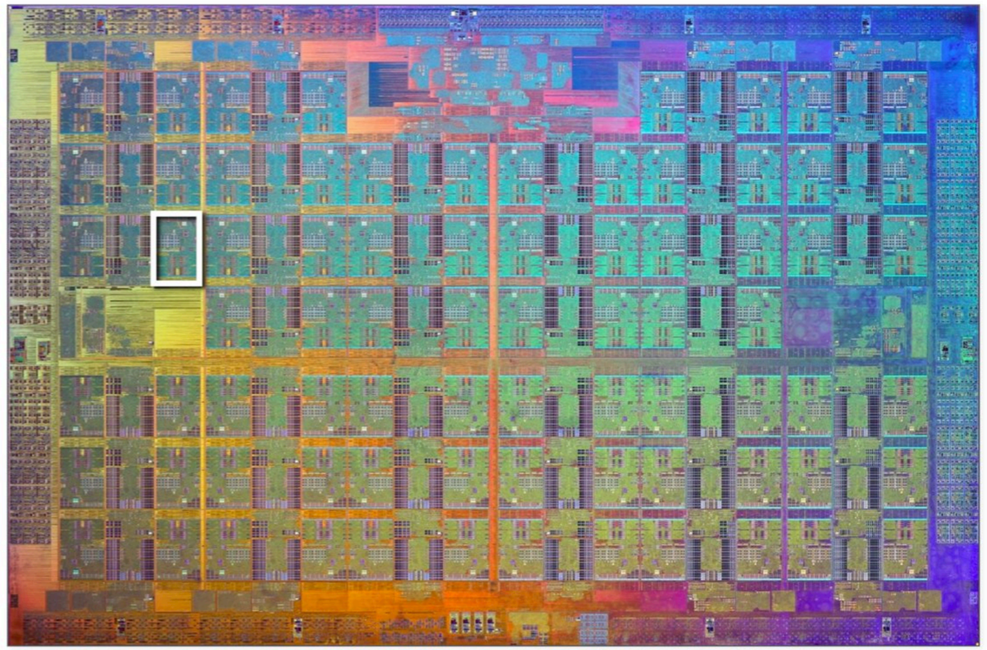
Stanford CS149, Fall 2021

https://gfxcourses.stanford.edu/cs149/fall21/lecture/whyparallelism/slide_52

Parallelism to the rescue

Intel Xeon Phi 7290 (2016)

72 cores (1.5 Ghz)



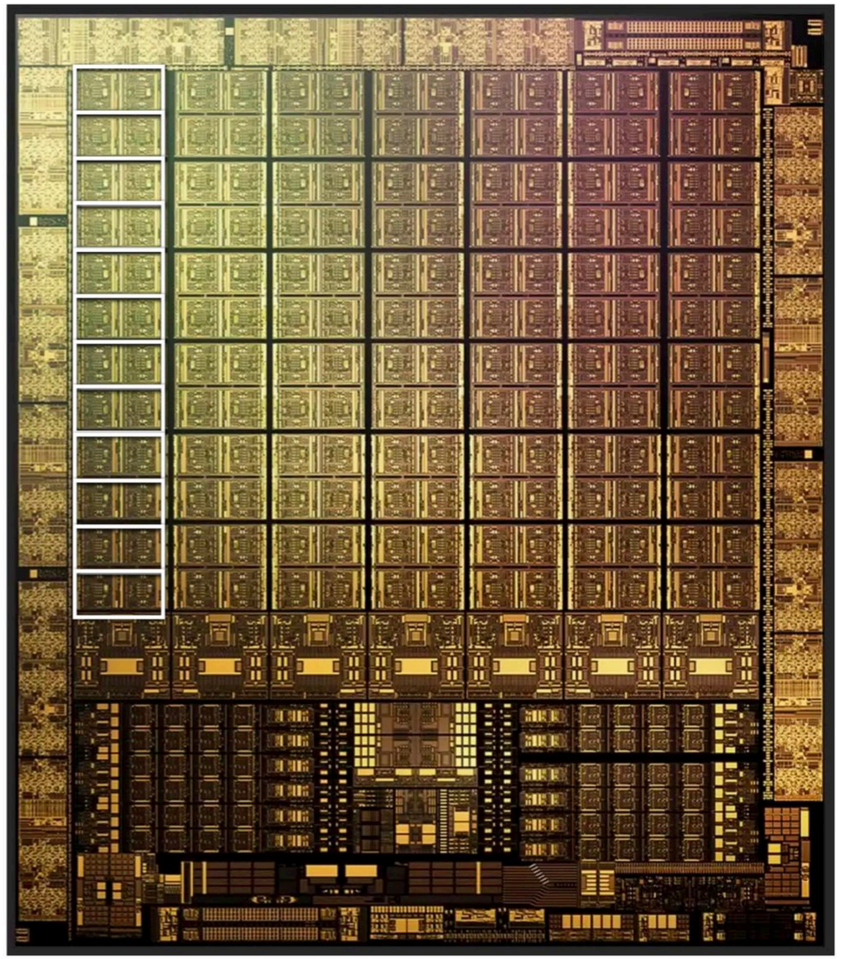
https://gfxcourses.stanford.edu/cs149/fall21/lecture/whyparallelism/slide_54

Parallelism to the rescue

NVIDIA Ampere GA102 GPU

GeForce RTX 3080 (2020)

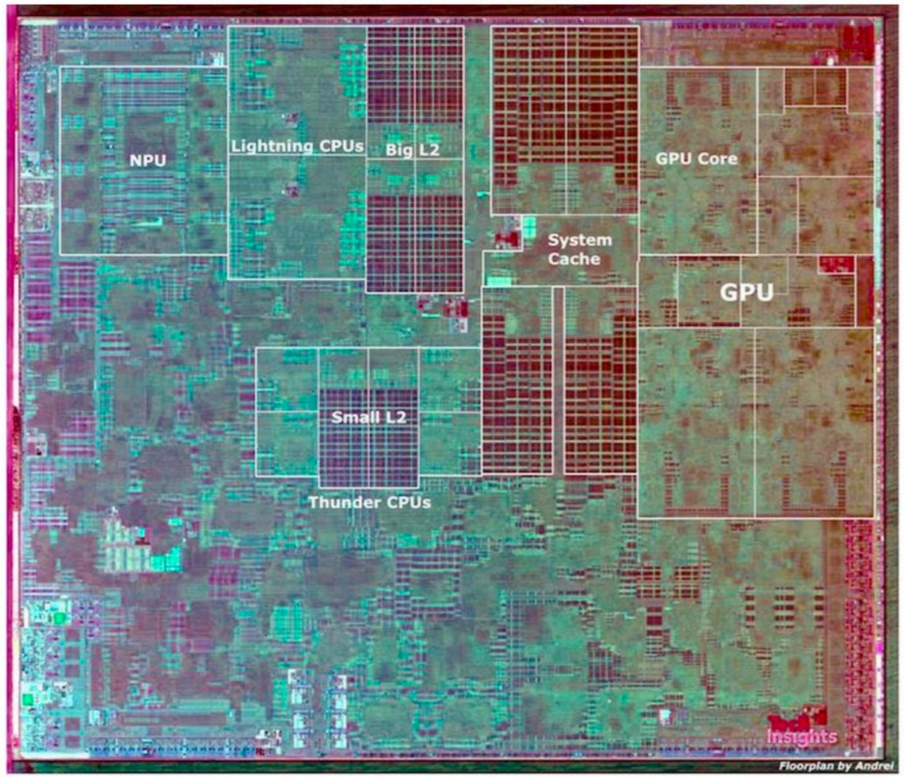
17,408 fp32 multipliers organized in 68 major processing blocks.



Parallelism to the rescue

Mobile parallel processing

Power constraints heavily influence the design of mobile systems



Apple A13 Bionic (in iPhone 11)

2 "big" CPU cores +
4 "small" CPU cores +

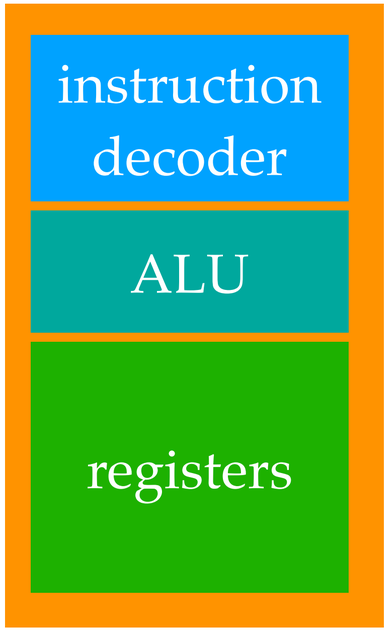
Apple-designed multi-core GPU +
Image processor +
Neural Engine for DNN acceleration +
Motion processor

Image Credit: Anandtech / TechInsights Inc.

https://gfxcourses.stanford.edu/cs149/fall21/lecture/whyparallelism/slide_54

A simple processor

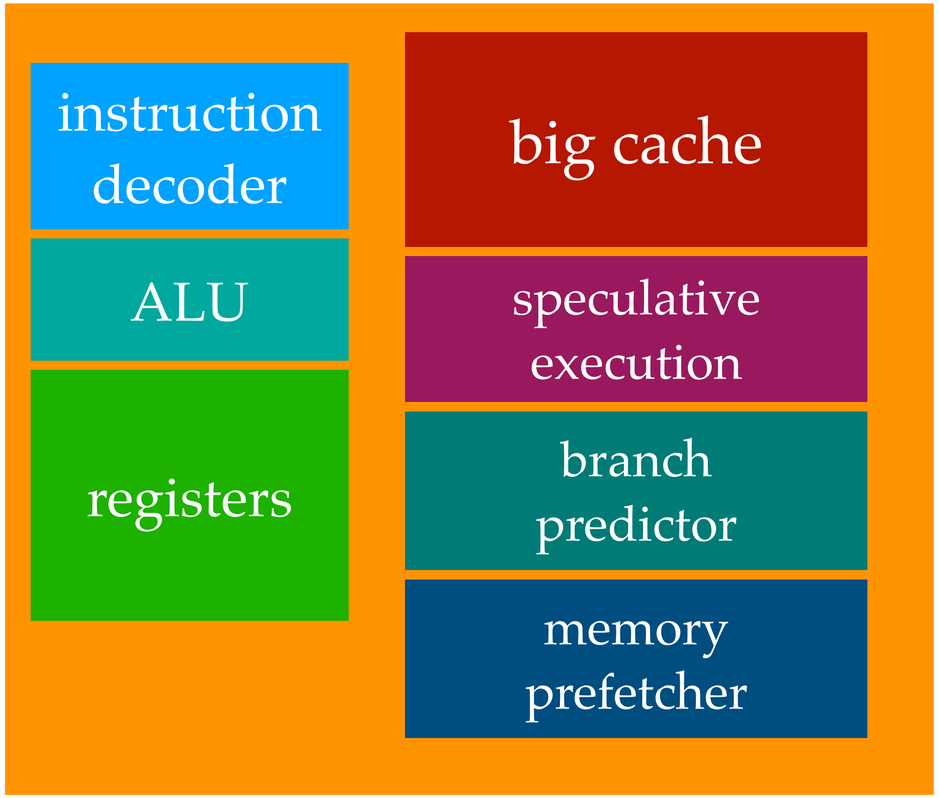
- a processor fetches instructions and execute them in serial order




```
load r0, 5
add r1, r0, r0
mul r1, r0, r1
…
store addr[10], r1
```

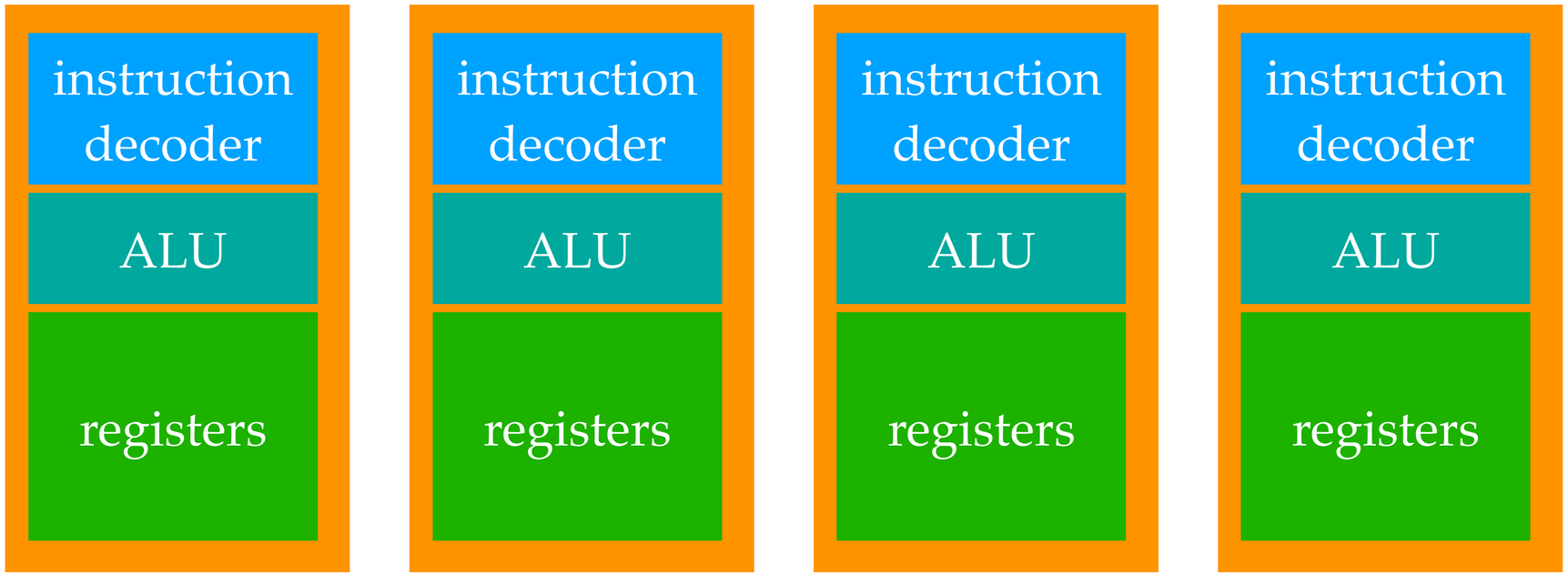
Pre multi-core era processor

lots of logic/transistors to make serial instructions run fast



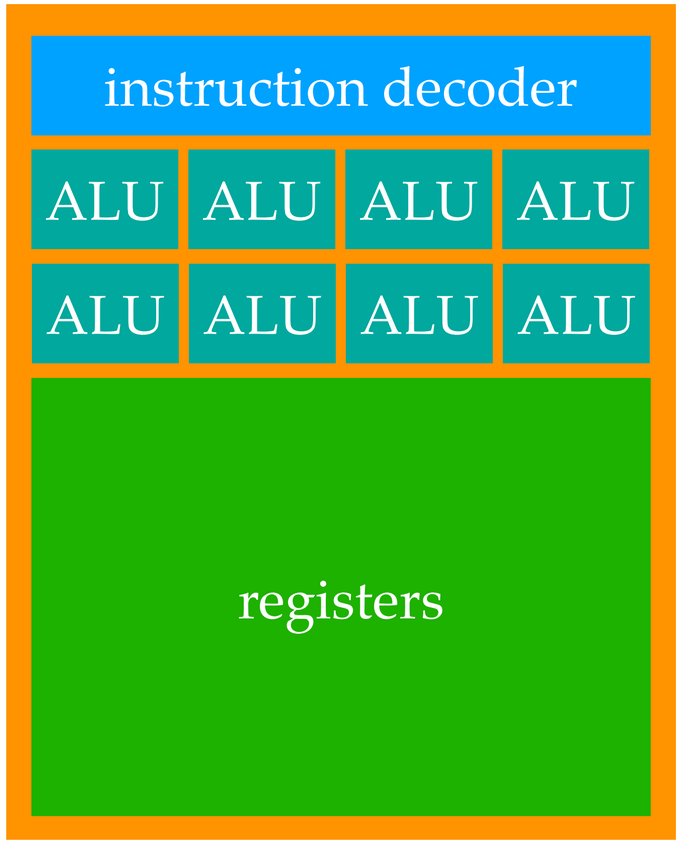
GPU / multi-core processor idea 1

- transistors are used to add more cores, not complex logic
  - each core is more “stupid”, but collectively they are faster



GPU / multi-core processor idea 2

- save transistors for instruction decoding by using the same instructions for many ALUs

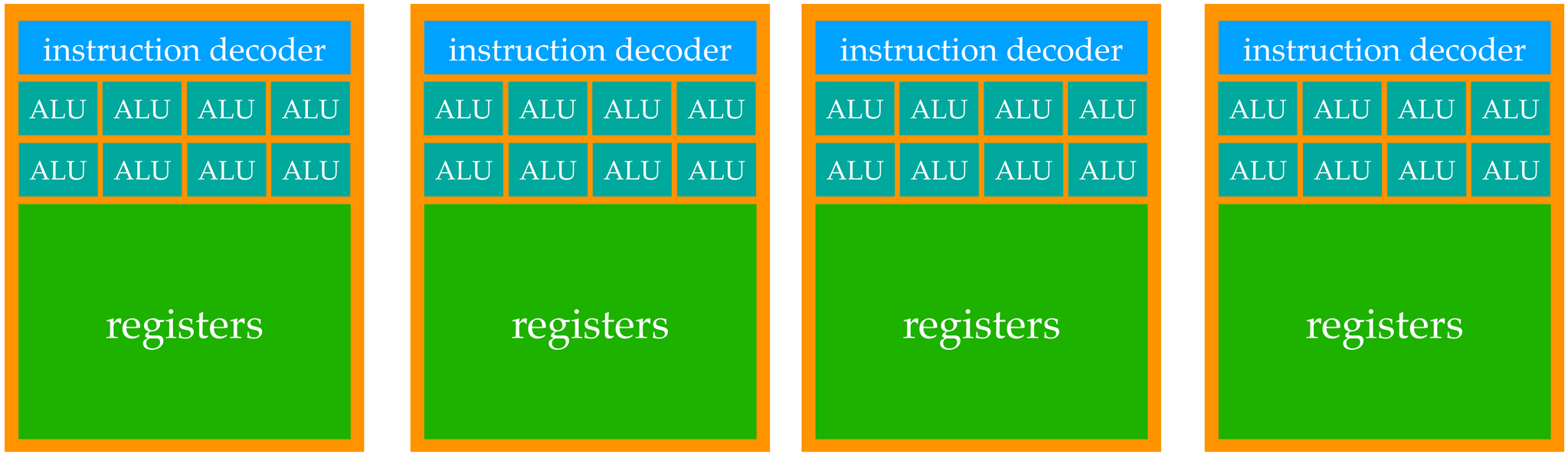



```
load r0, 5
add r1, r0, r0
mul r1, r0, r1
…
store addr[10], r1
```

Single Instruction, Multiple Data (SIMD)
data are processed in parallel

modern GPUs usually have 8-32 ALUs per unit

Each core can do its own thing



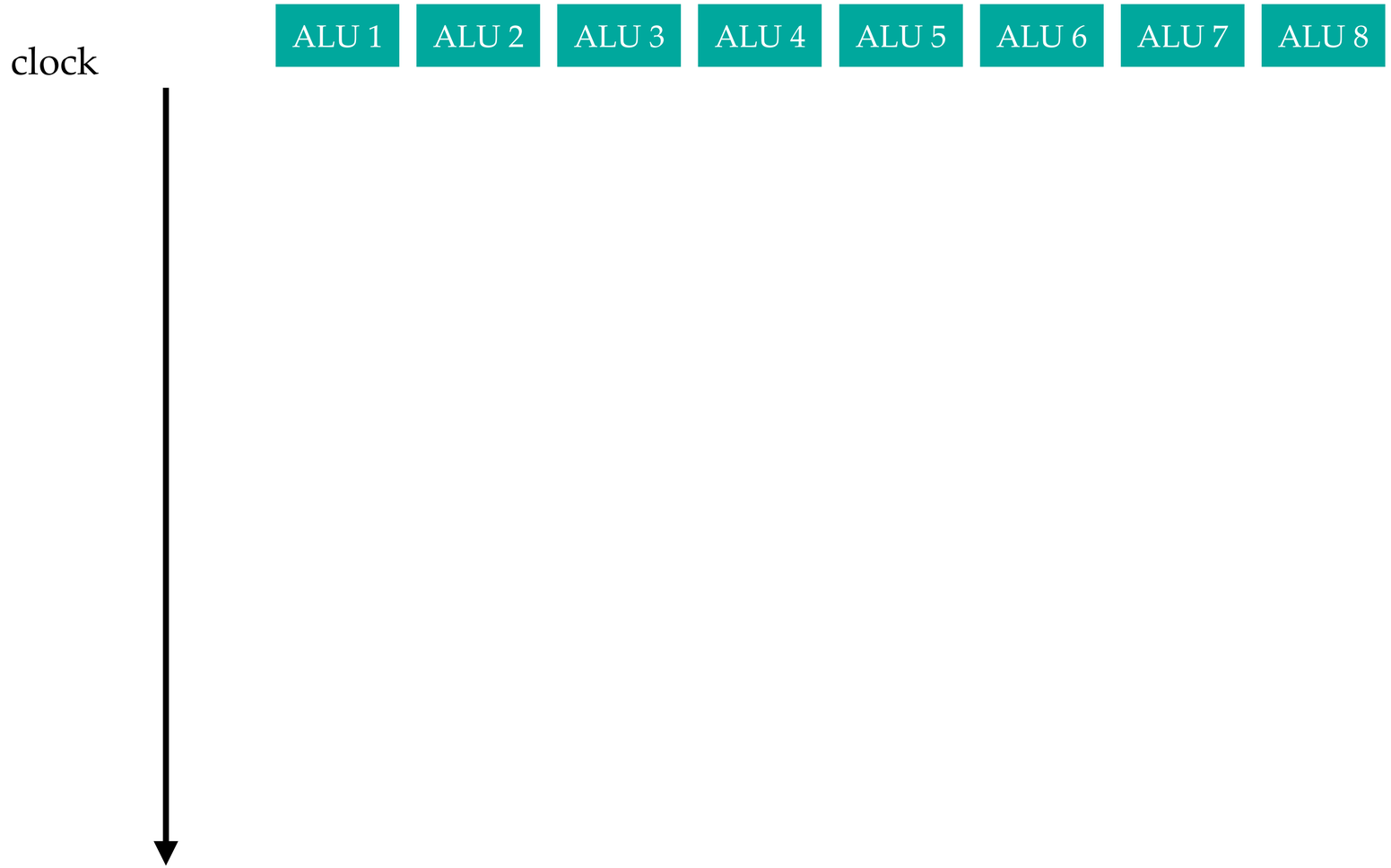
multiply 16 numbers

add 16 numbers

subtract 16 numbers

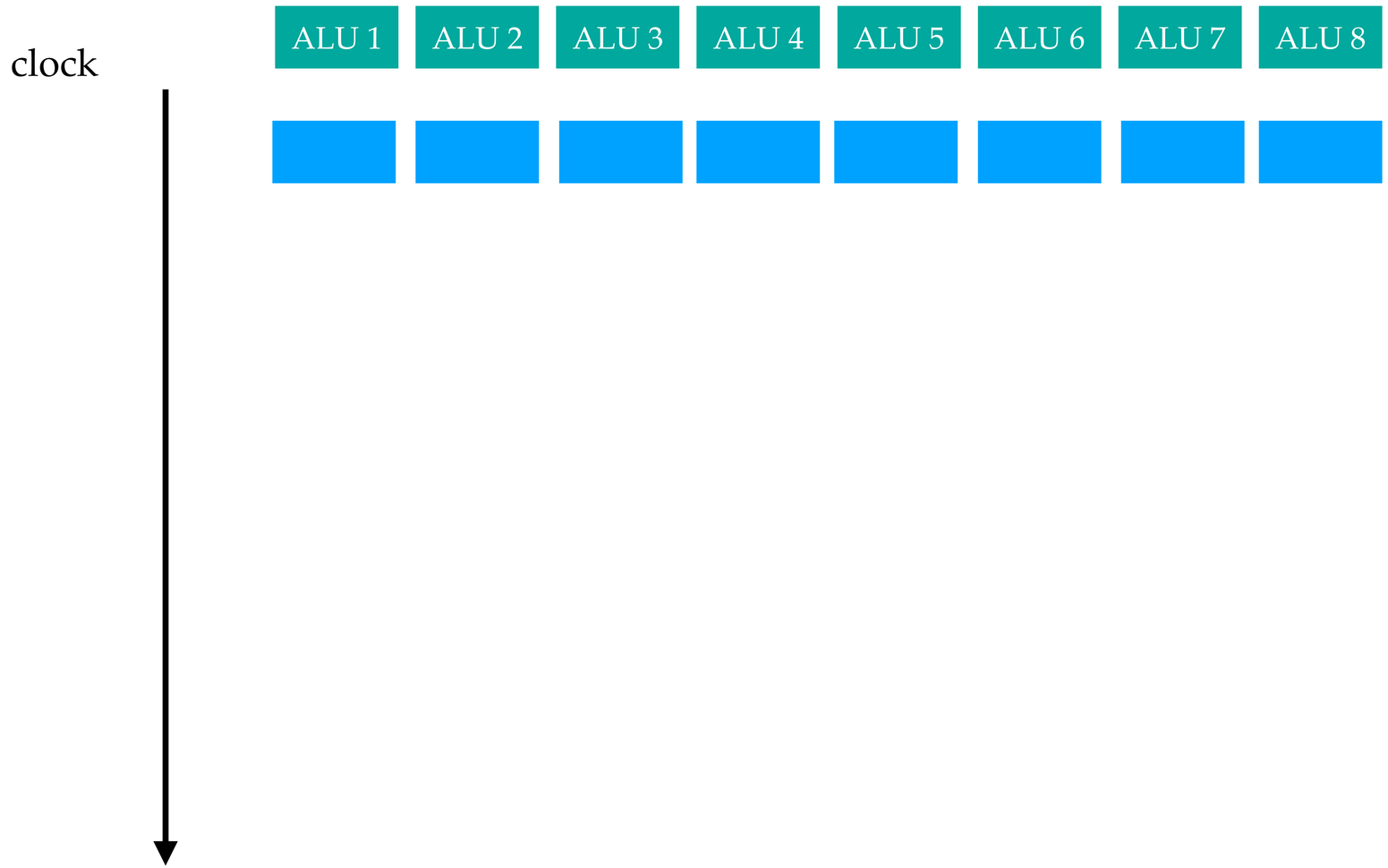
divide 16 numbers

What about conditions?



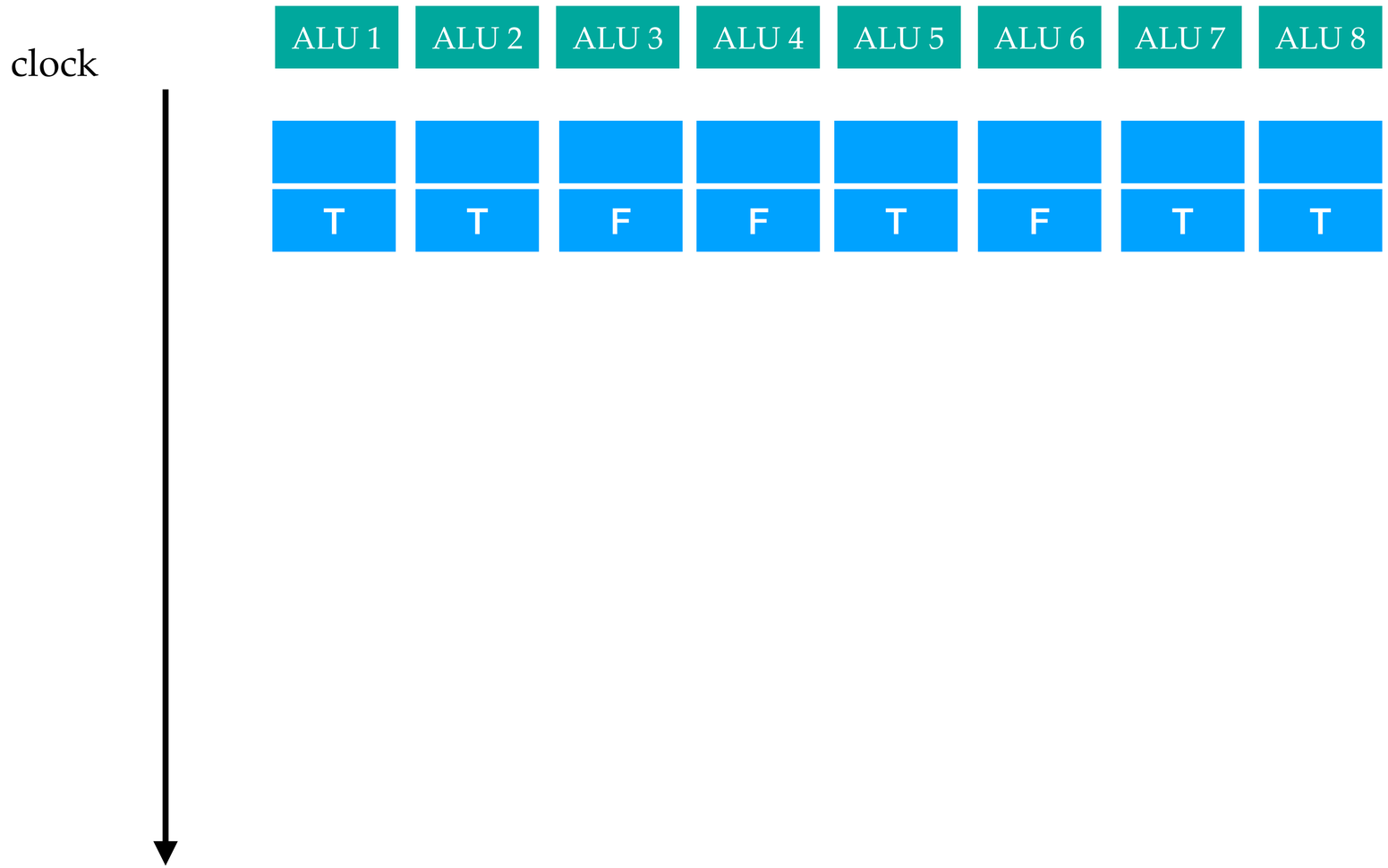

```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

What about conditions?



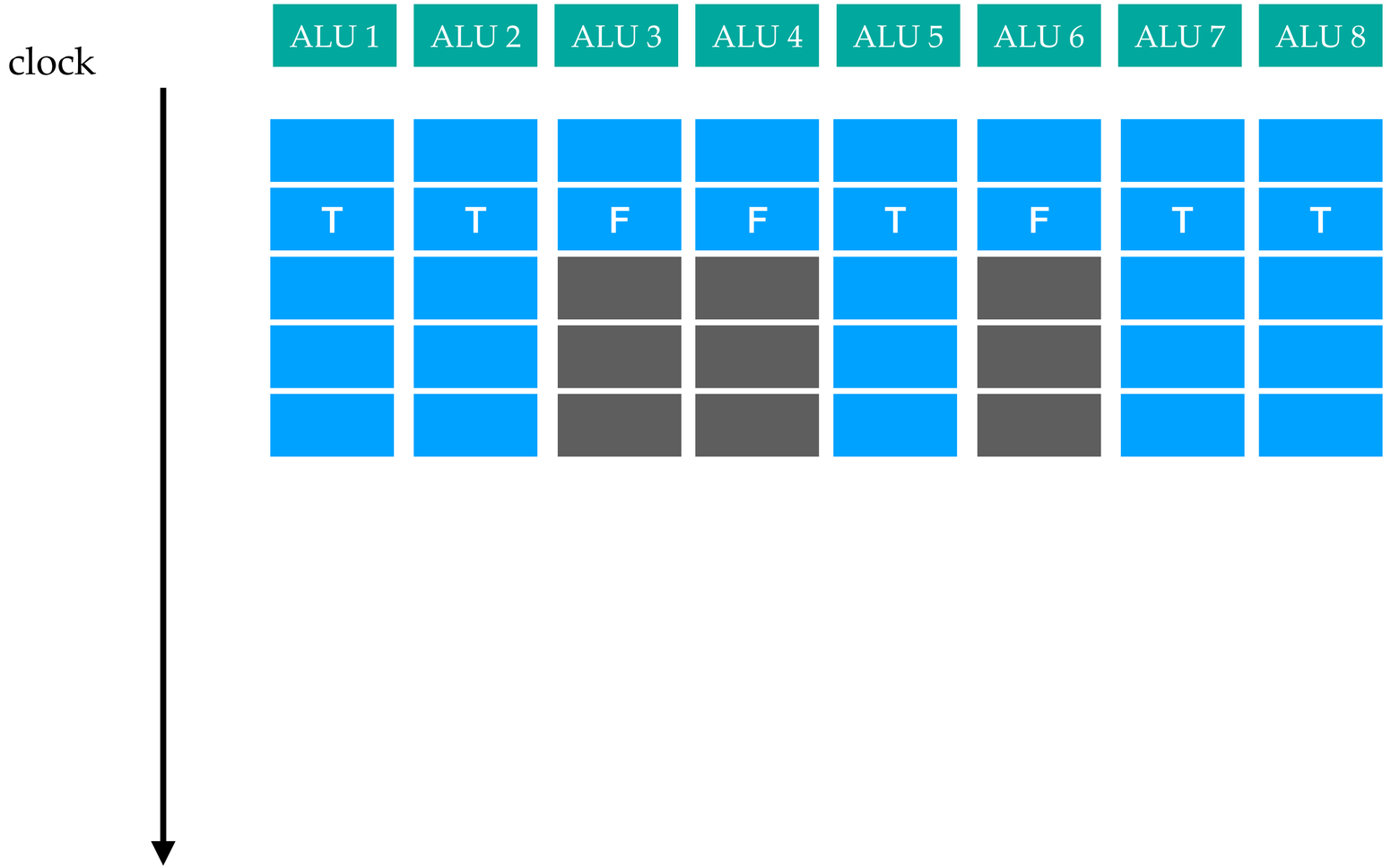

```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

What about conditions?



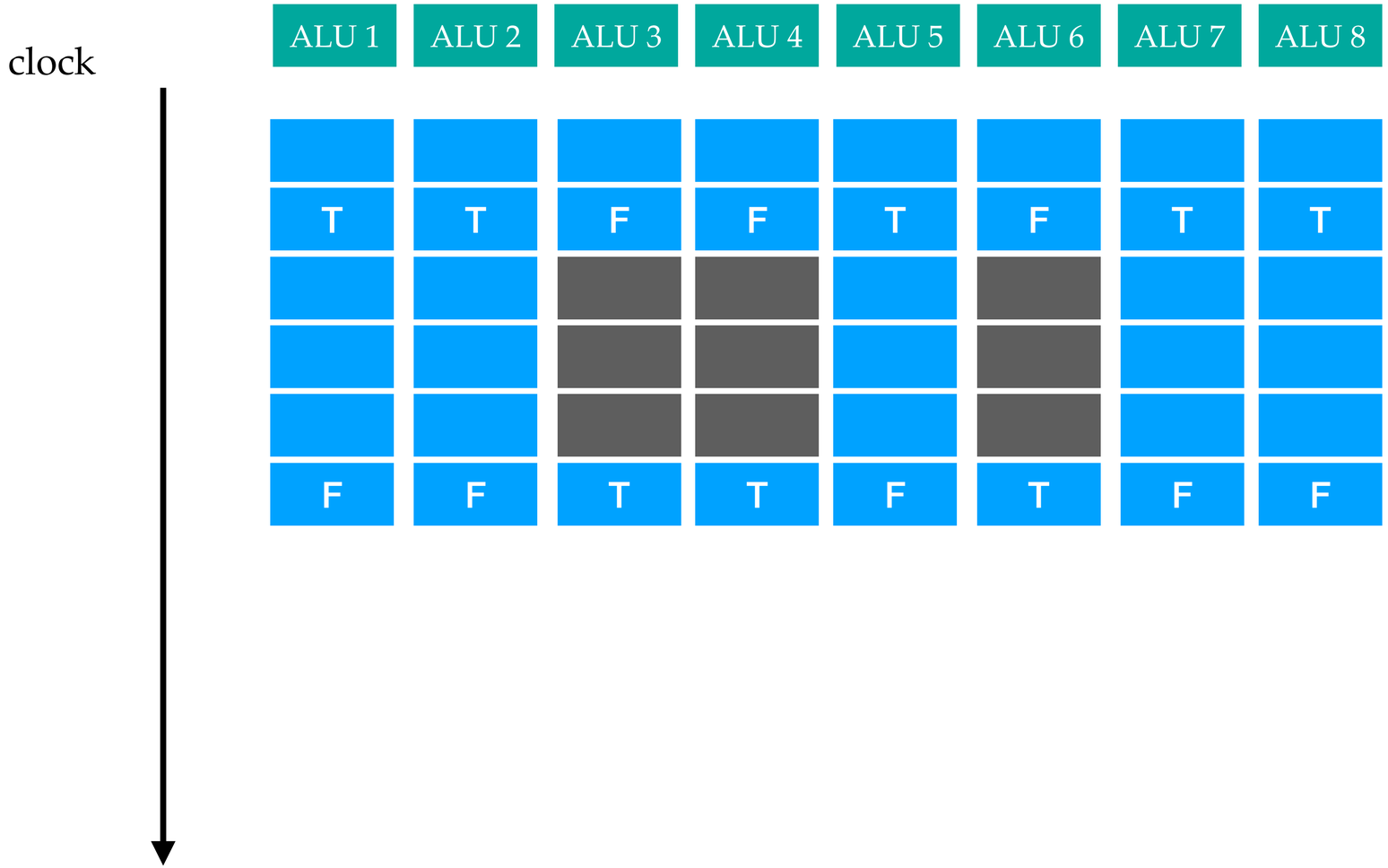

```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

What about conditions?



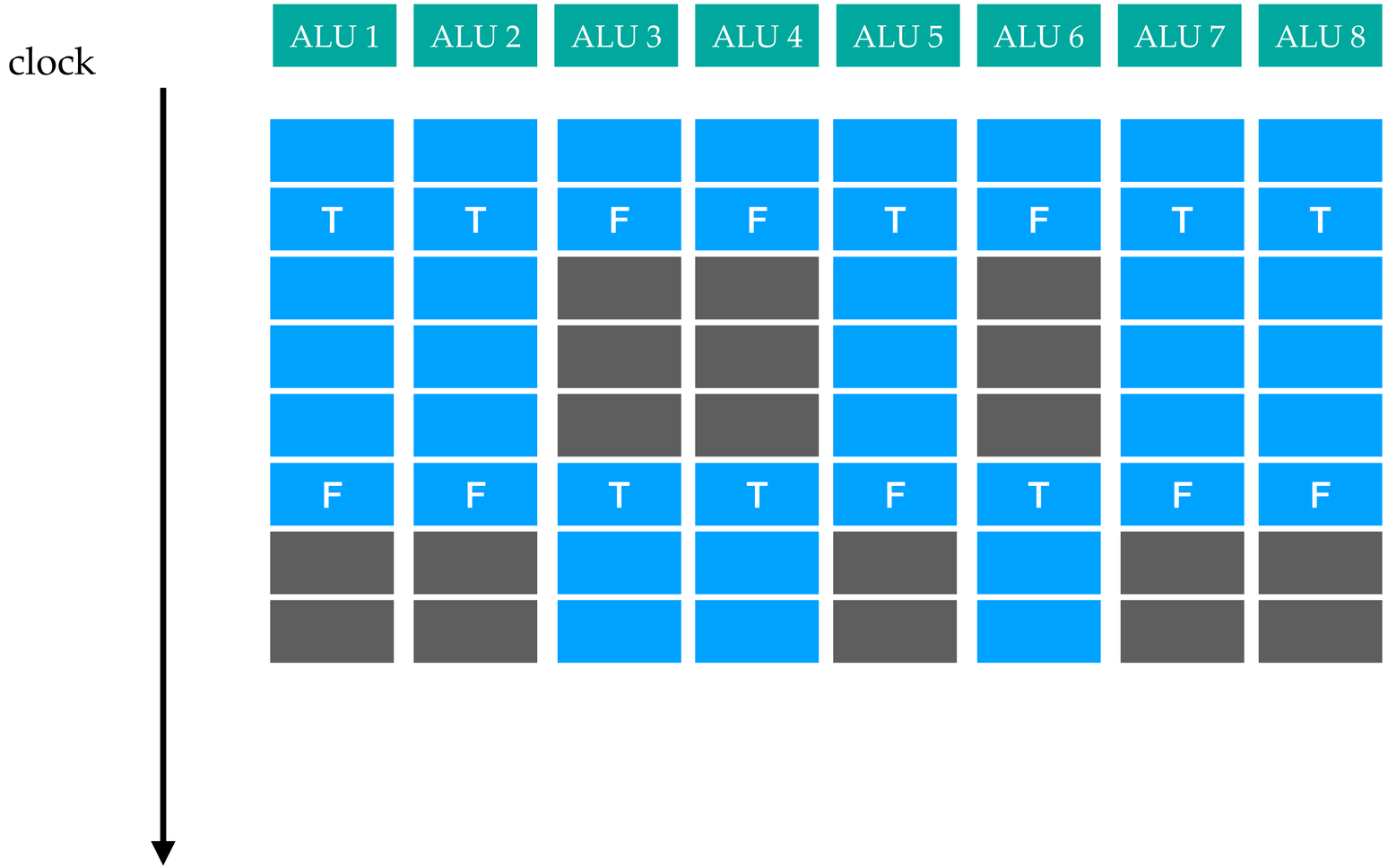

```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

What about conditions?




```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

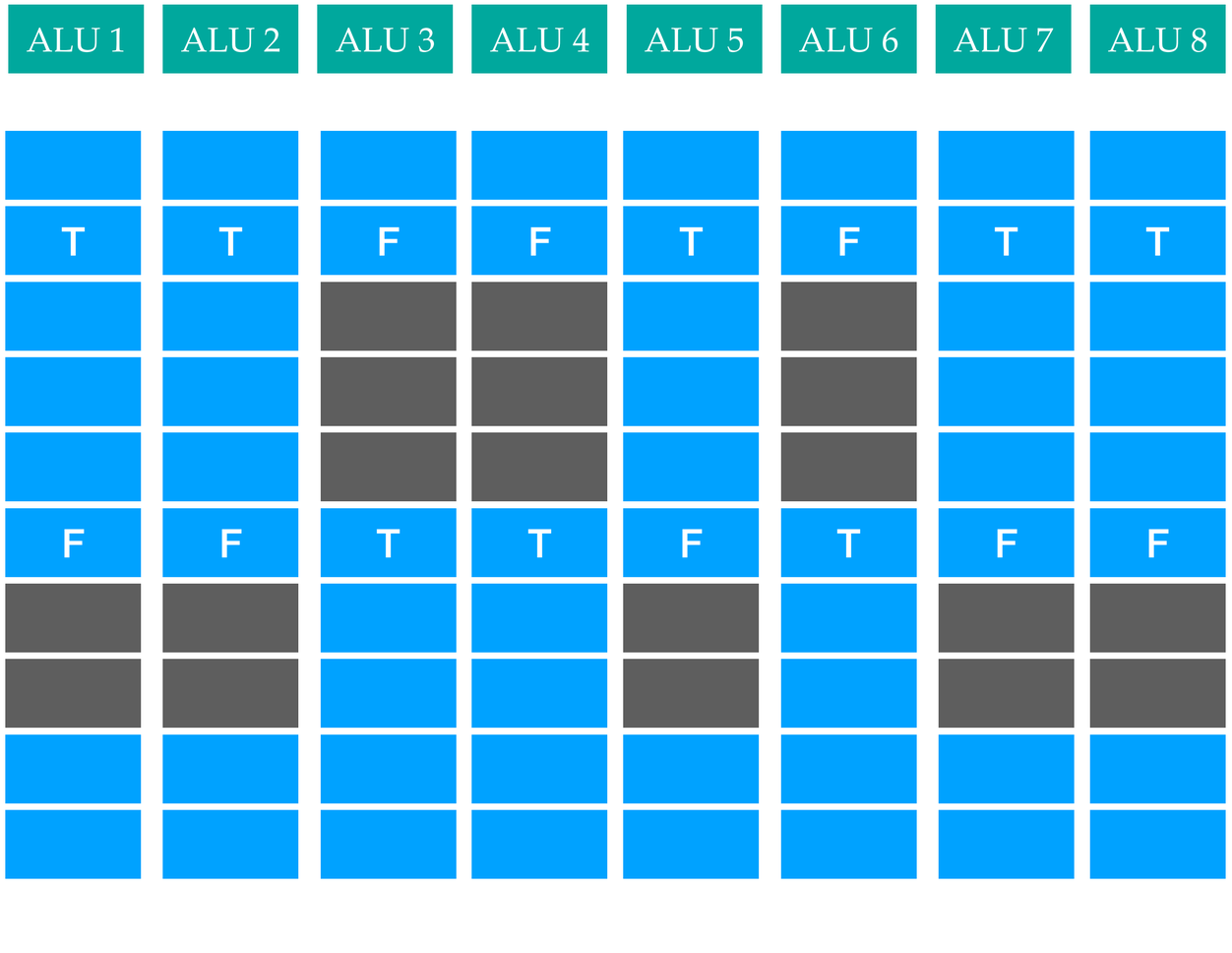
What about conditions?




```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

What about conditions?

clock

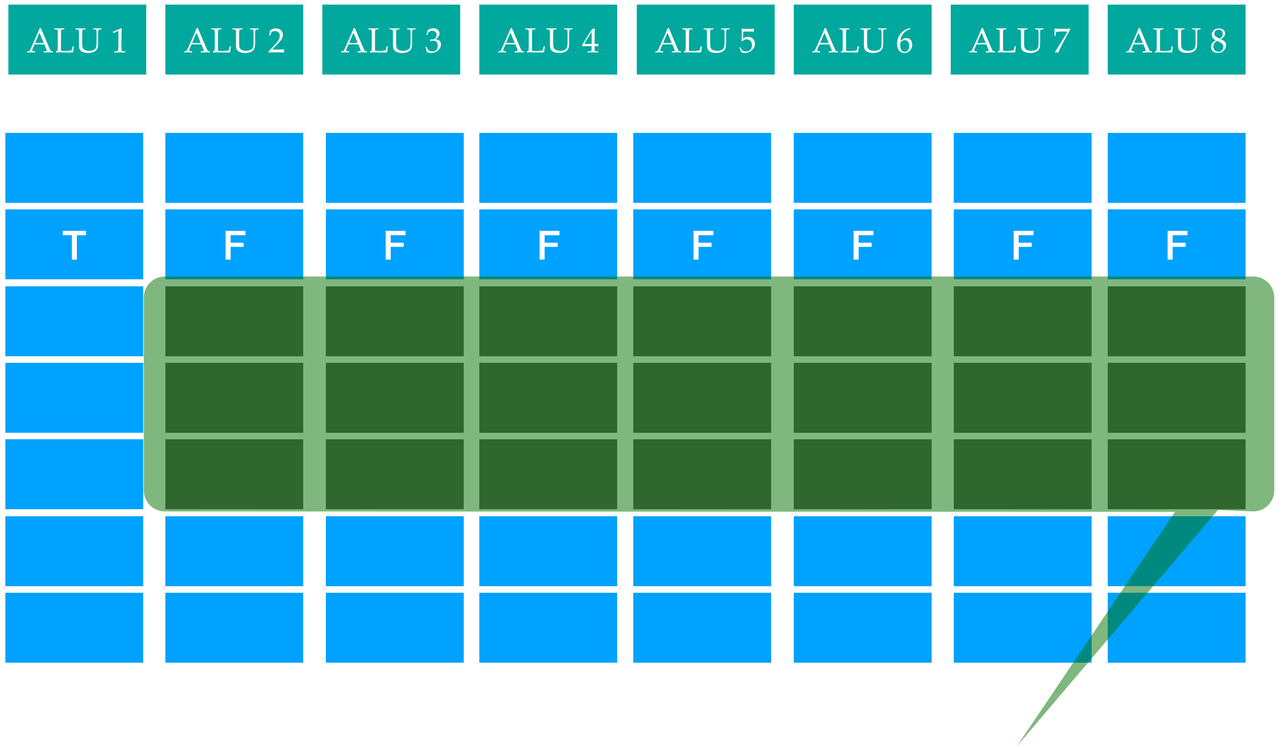



```
t = x[i]
if (t > 0) {
   t = t * t
   t = t * 50;
   t = t + 100;
} else {
   t = t + 30;
   t = t / 10;
}
y[i] = t;
```

some ALUs need to idle and wait for the others!

Thread divergence = bad

clock



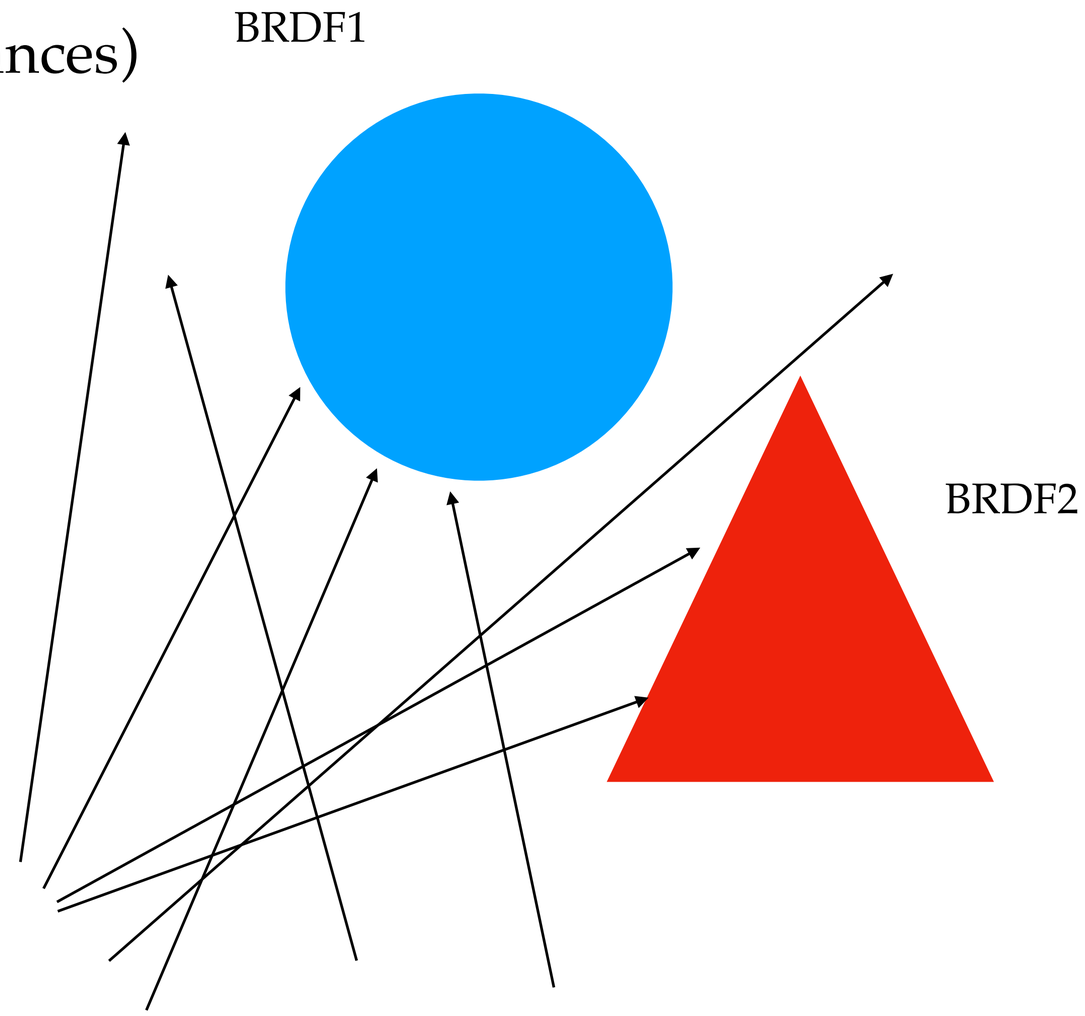

```
t = x[i]
if (t > 10) {
   t = t * t
   t = t * 50;
   t = t + 100;
}
y[i] = t;
```

we're all waiting for you!! \ _ /

12.5% peak performance inside condition

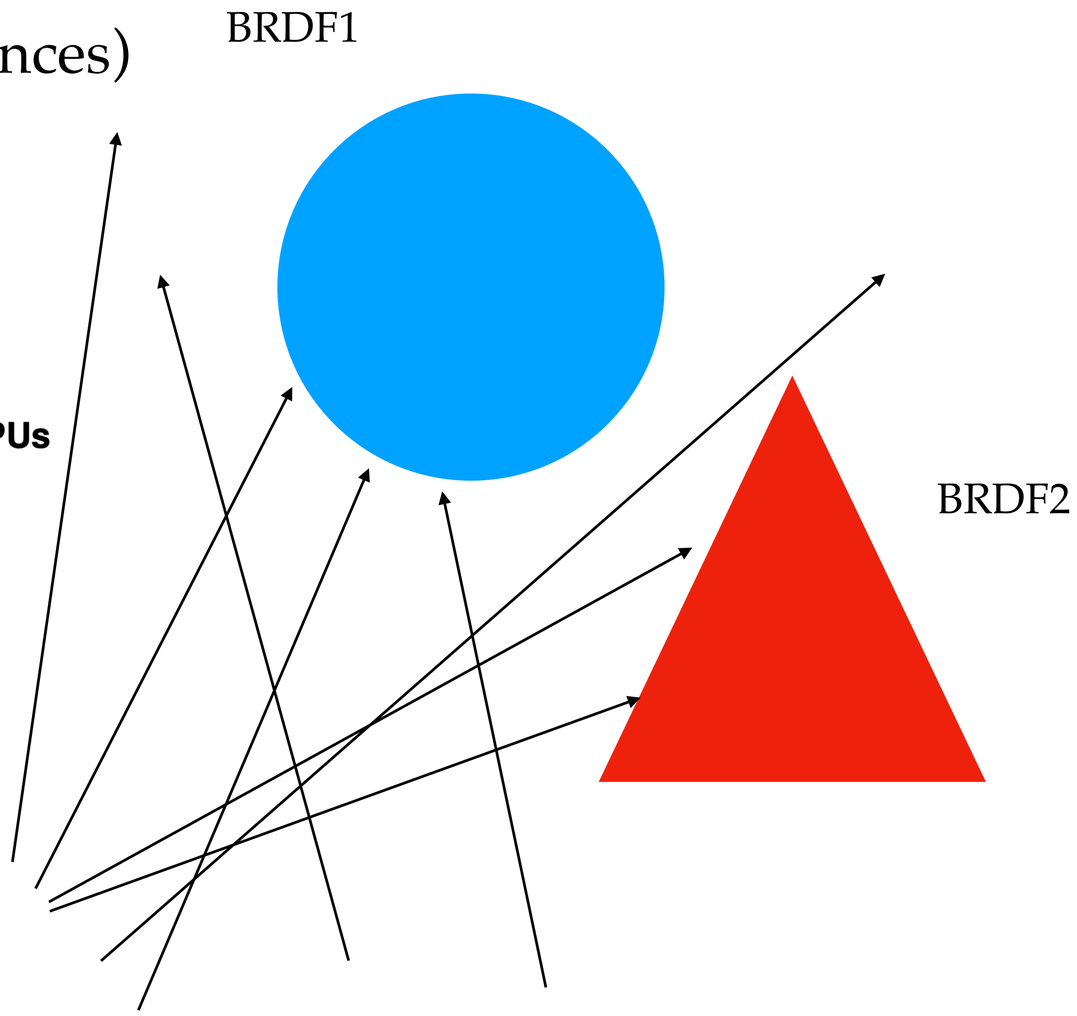
When do we have bad thread divergence?

- ray tracing (especially secondary bounces) is extremely incoherent!



When do we have bad thread divergence?

- ray tracing (especially secondary bounces) is extremely incoherent!



Megakernels Considered Harmful: Wavefront Path Tracing on GPUs

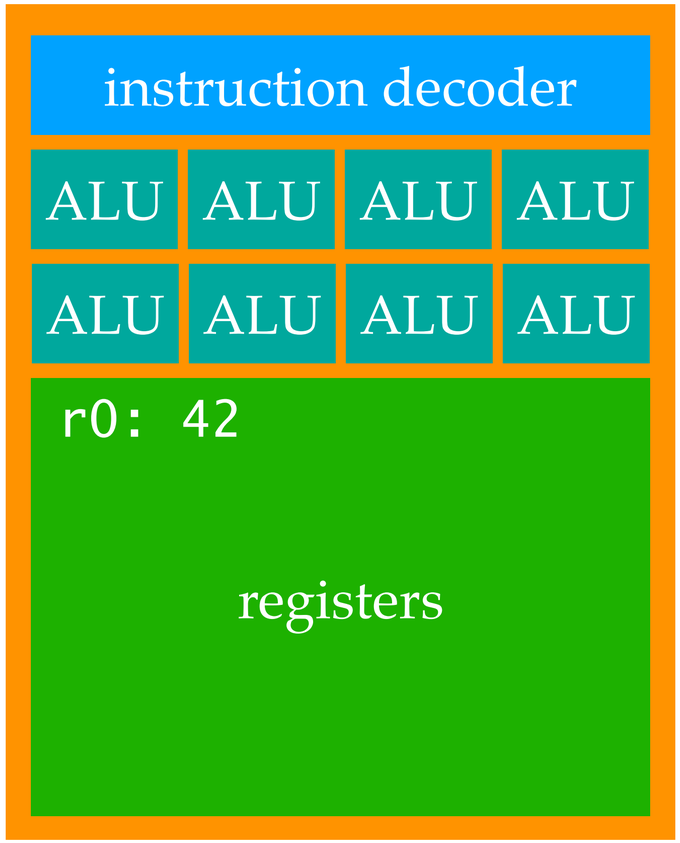
Samuli Laine Tero Karras Timo Aila

NVIDIA*

Progressive Light Transport Simulation on the GPU: Survey and Improvements

TOMÁŠ DAVIDOVIČ
Saarland University and Intel VCI
JAROSLAV KŘIVÁNEK
Charles University in Prague
MILOŠ HAŠAN
Autodesk, Inc.
and
PHILIPP SLUSALLEK
Saarland University, DFKI

Memory access



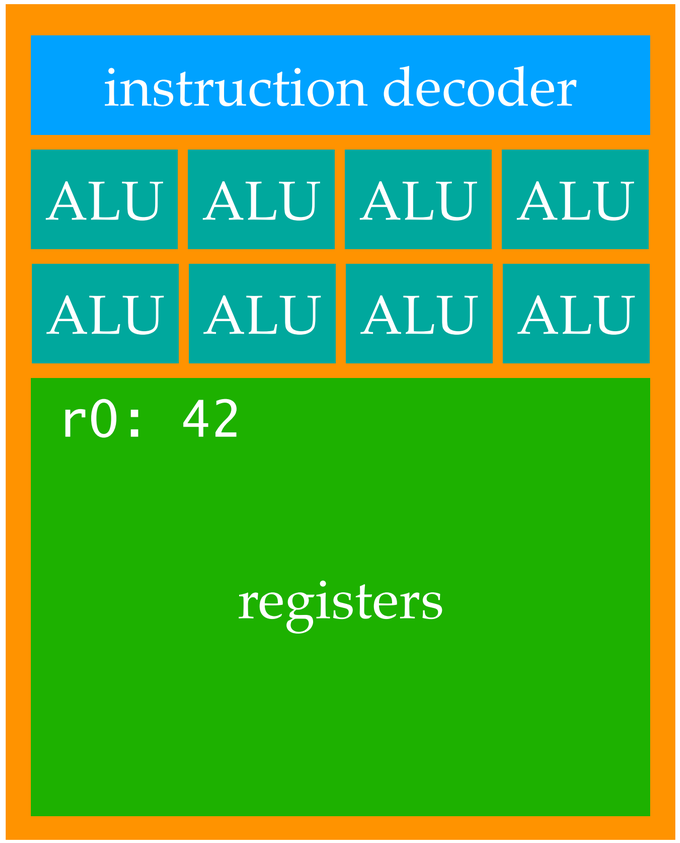

```
load r0, addr[0xff326a38]
```

memory

```
...
0xff326a30: 0
0xff326a34: 100
0xff326a38: 42
...
```

Memory access is expensive!

- often takes hundreds of cycles for loading from the main memory




```
load r0, addr[0xff326a38]
```

memory

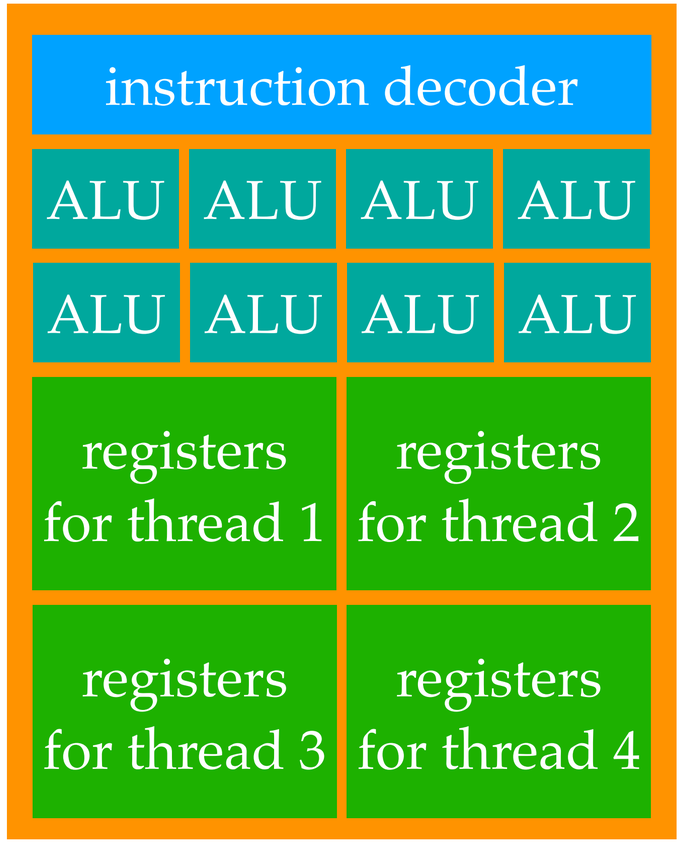
```
…
0xff326a30: 0
0xff326a34: 100
0xff326a38: 42
…
```

Memory access is expensive!

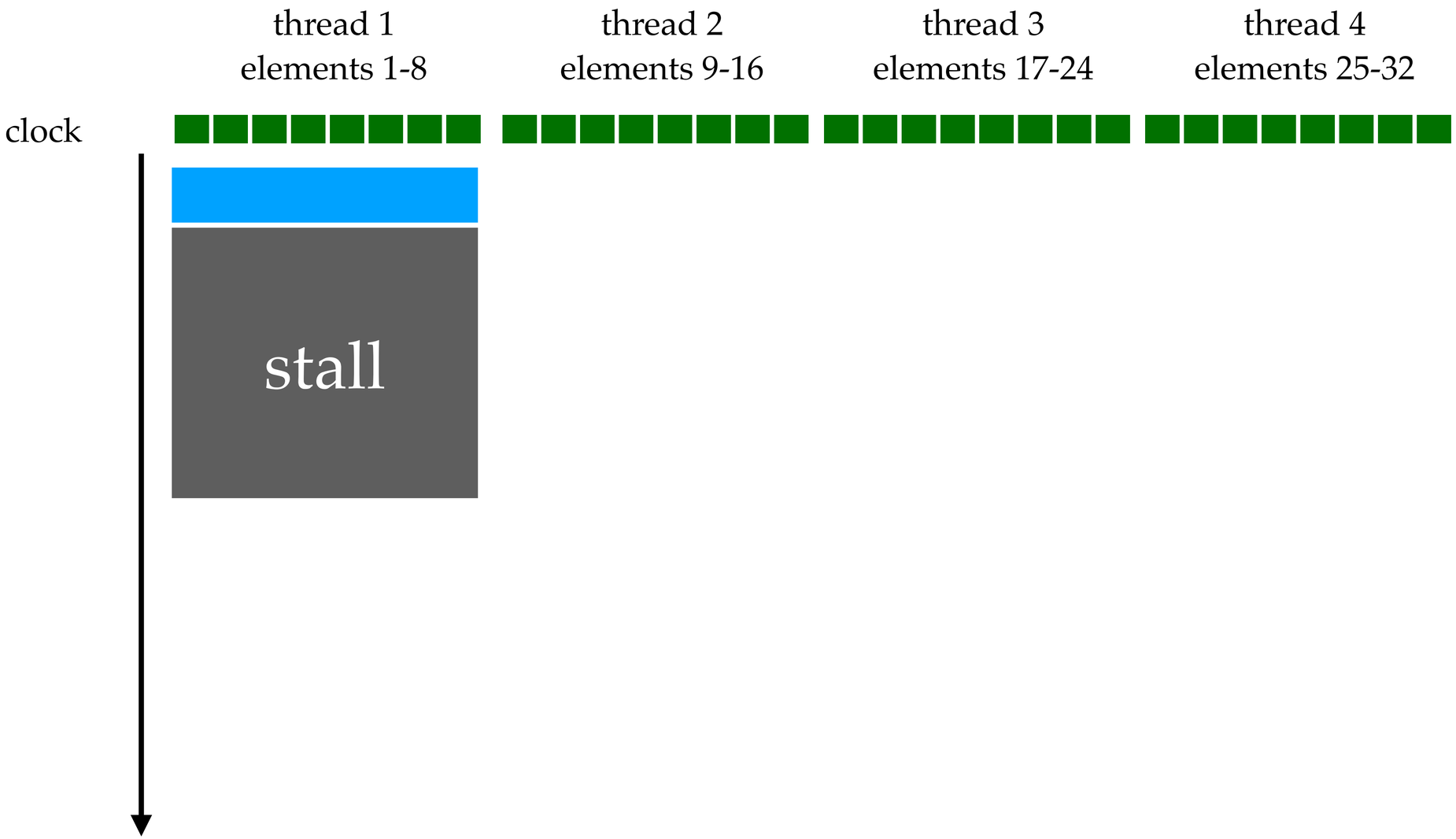
- threads are stalled when accessing from main memory

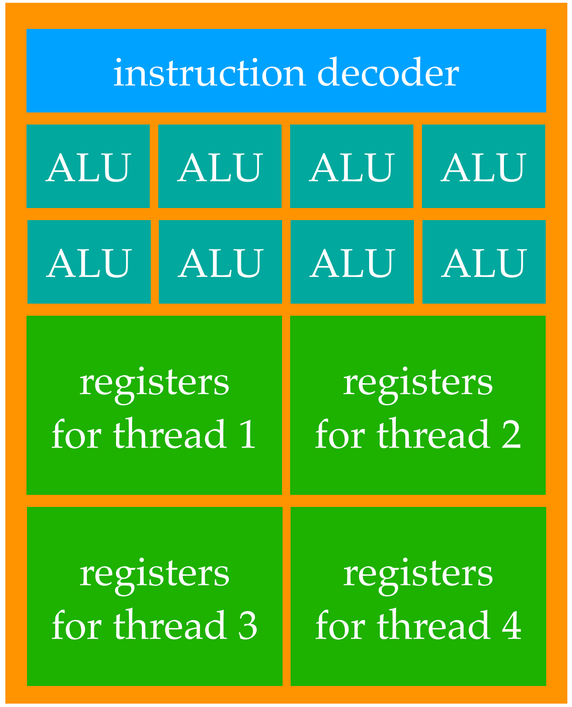
GPU / multi-core processor idea 3

- interleave processing of multiple threads on the same core to hide memory latency
  - move to different threads if you can't make progress on one



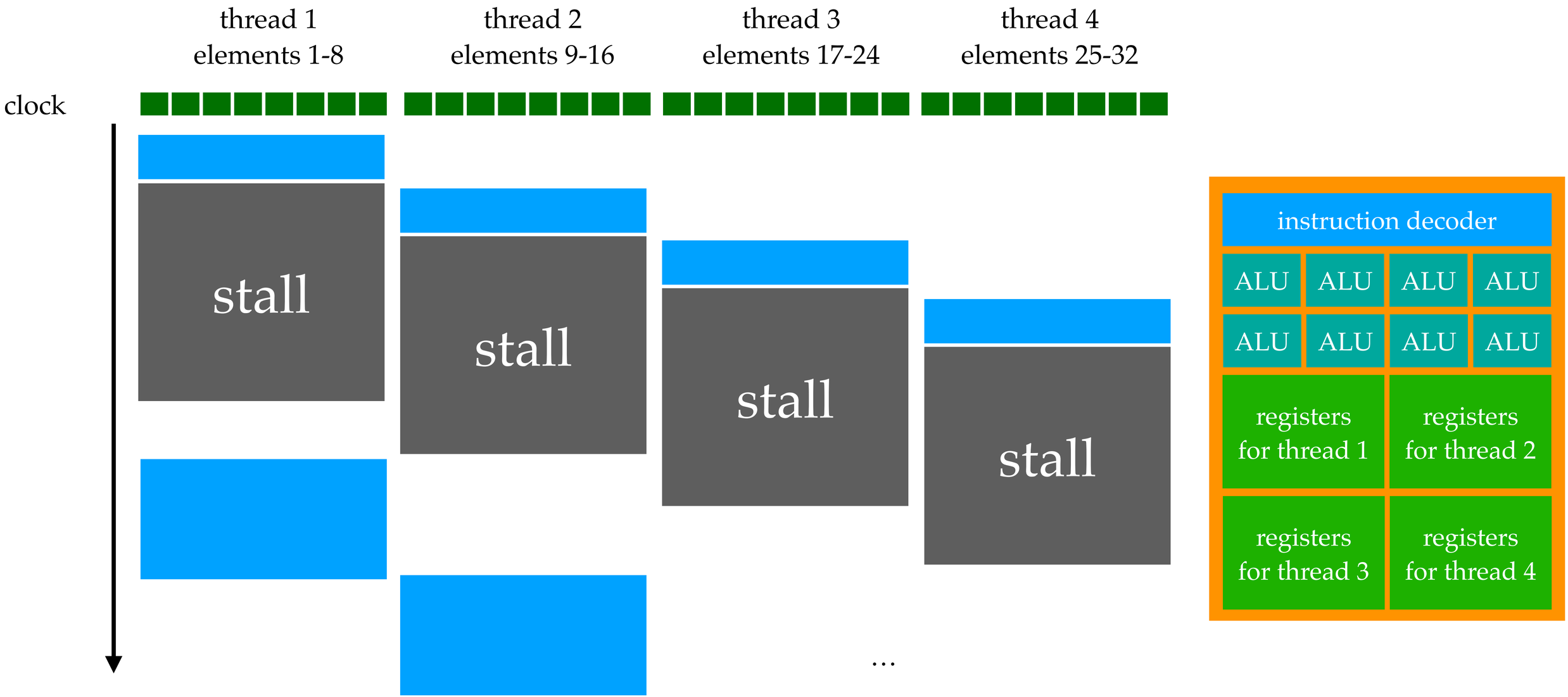
Latency hiding with multi-threading



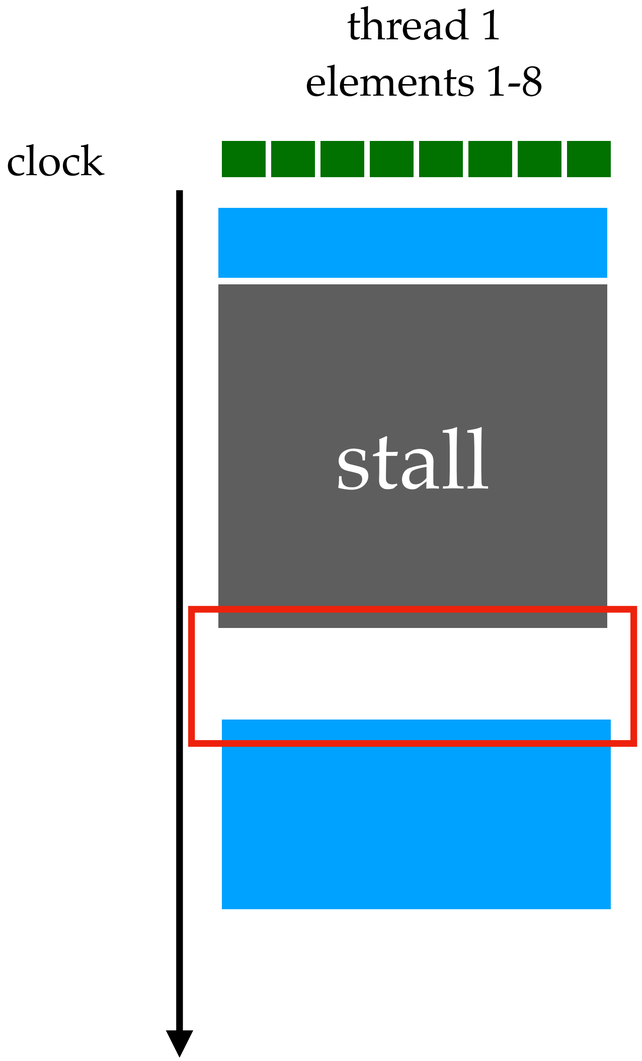


Latency hiding with multi-threading

Latency hiding with multi-threading

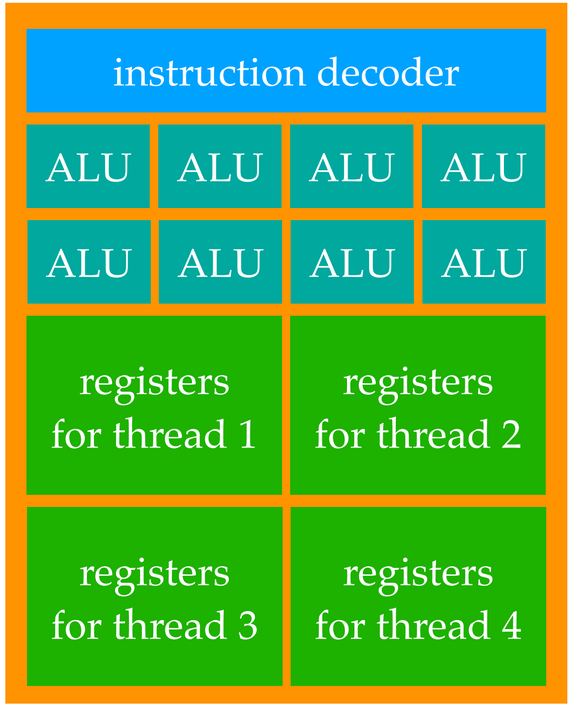


Latency-throughput tradeoff

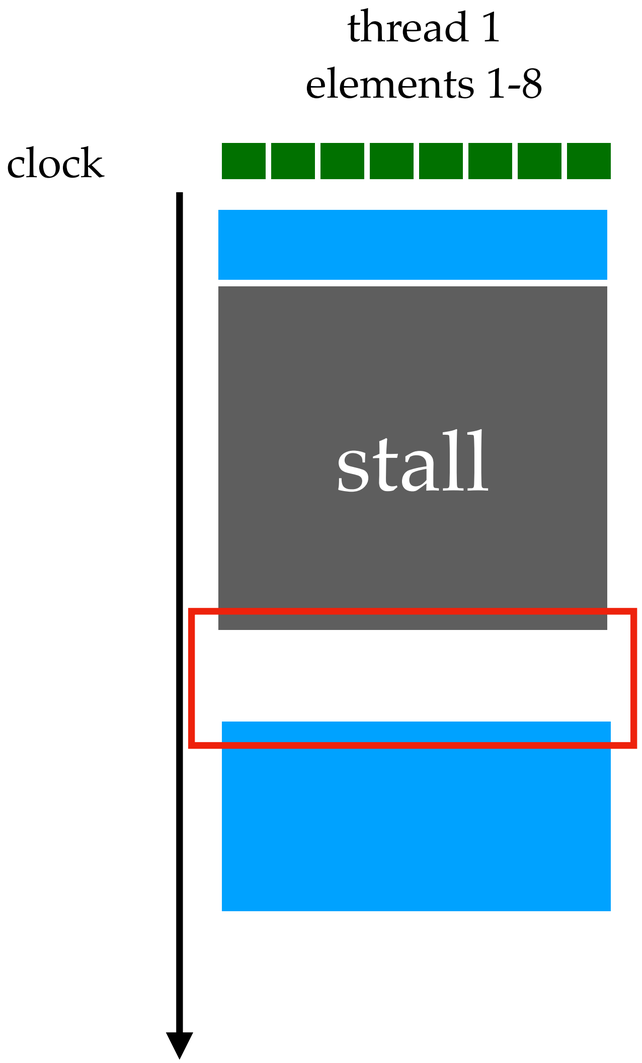


run time for thread 1 is increased,
but the total throughput is improved

thread 1 is waiting for thread 2-4 here



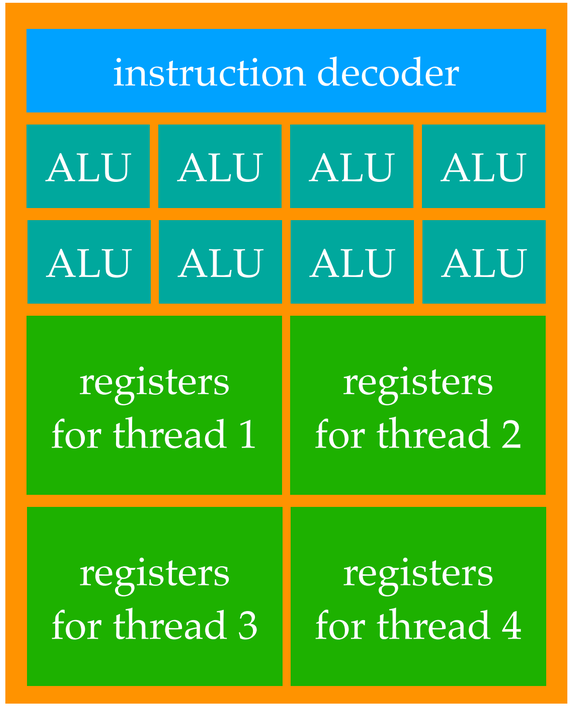
Latency-throughput tradeoff



run time for thread 1 is increased,
but the total throughput is improved

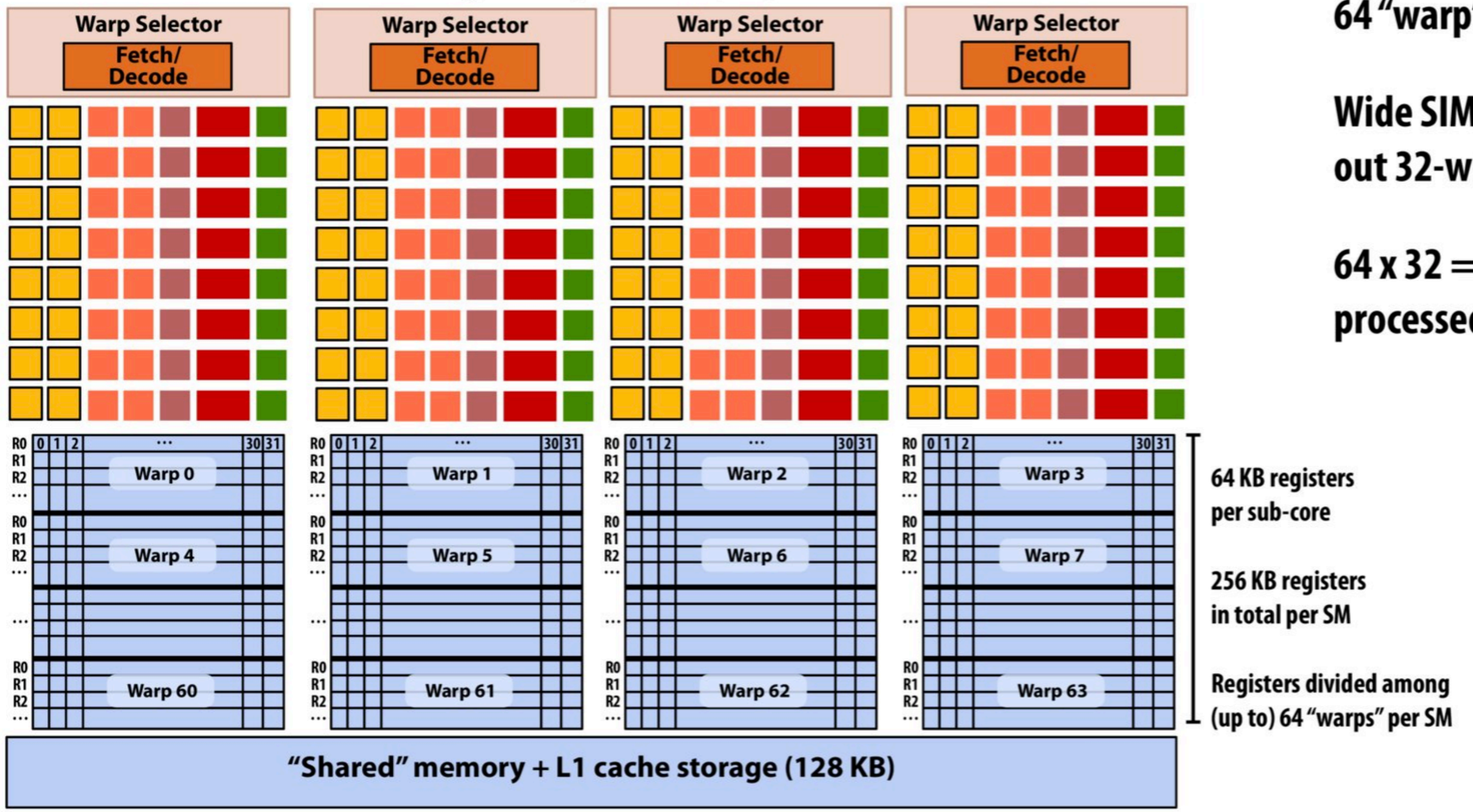
thread 1 is waiting for thread 2-4 here

registers for each thread
are reduced as well



GPUs: extreme throughput-oriented processors

This is one NVIDIA V100 streaming multi-processor (SM) unit



64 "warp" execution contexts per SM

Wide SIMD: 16-wide SIMD ALUs (carry out 32-wide SIMD execute over 2 clocks)

64 x 32 = up to 2048 data items processed concurrently per "SM" core

64 KB registers per sub-core

256 KB registers in total per SM

Registers divided among (up to) 64 "warps" per SM

= SIMD fp32 functional unit, control shared across 16 units (16 x MUL-ADD per clock *)

= SIMD int functional unit, control shared across 16 units (16 x MUL/ADD per clock *)

= SIMD fp64 functional unit, control shared across 8 units (8 x MUL/ADD per clock **)

= Tensor core unit

= Load/store unit

* one 32-wide SIMD operation every 2 clocks

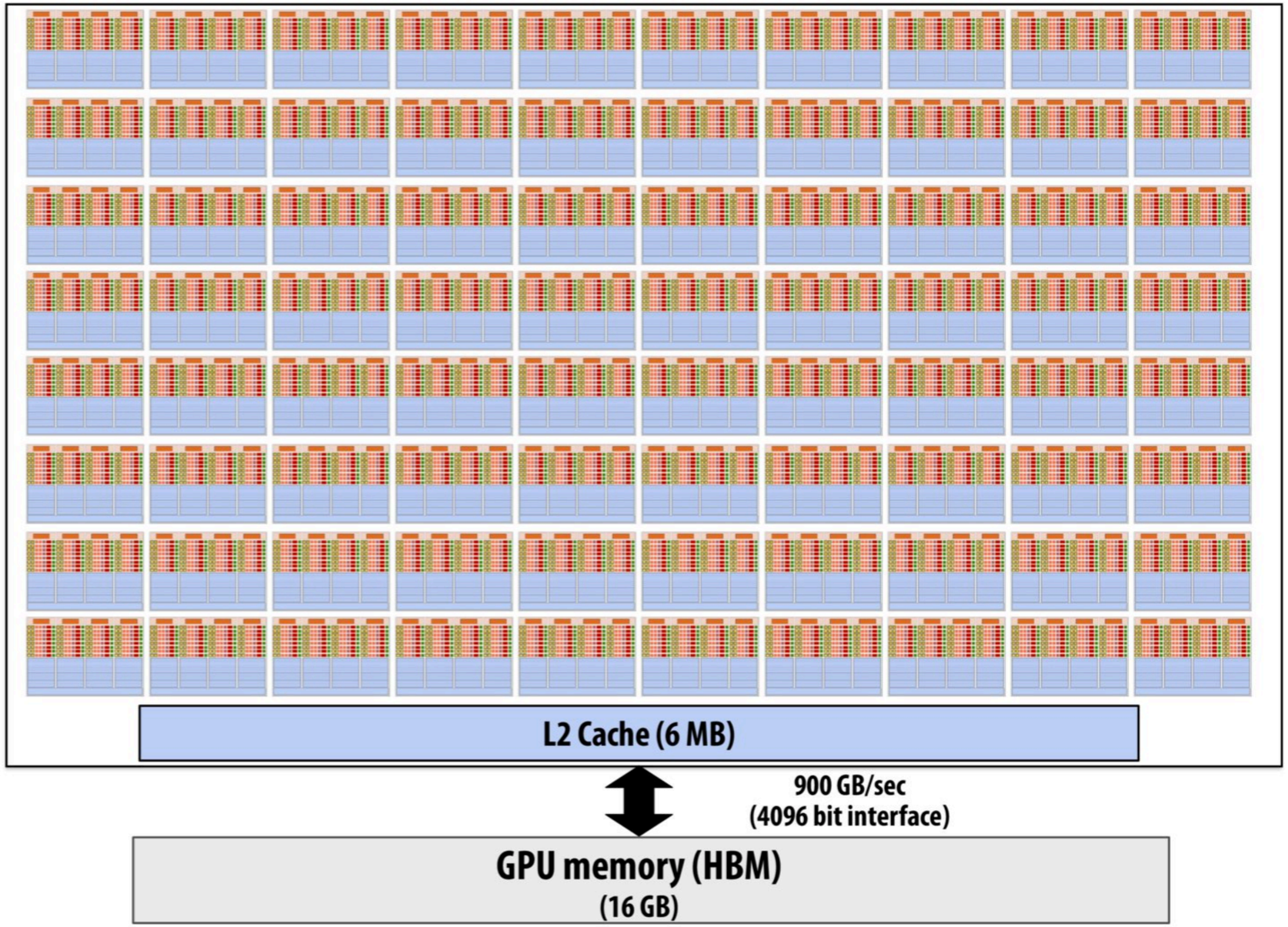
** one 32-wide SIMD operation every 4 clocks

https://gfxcourses.stanford.edu/cs149/fall21/lecture/multicorearch/slide_82

NVIDIA V100

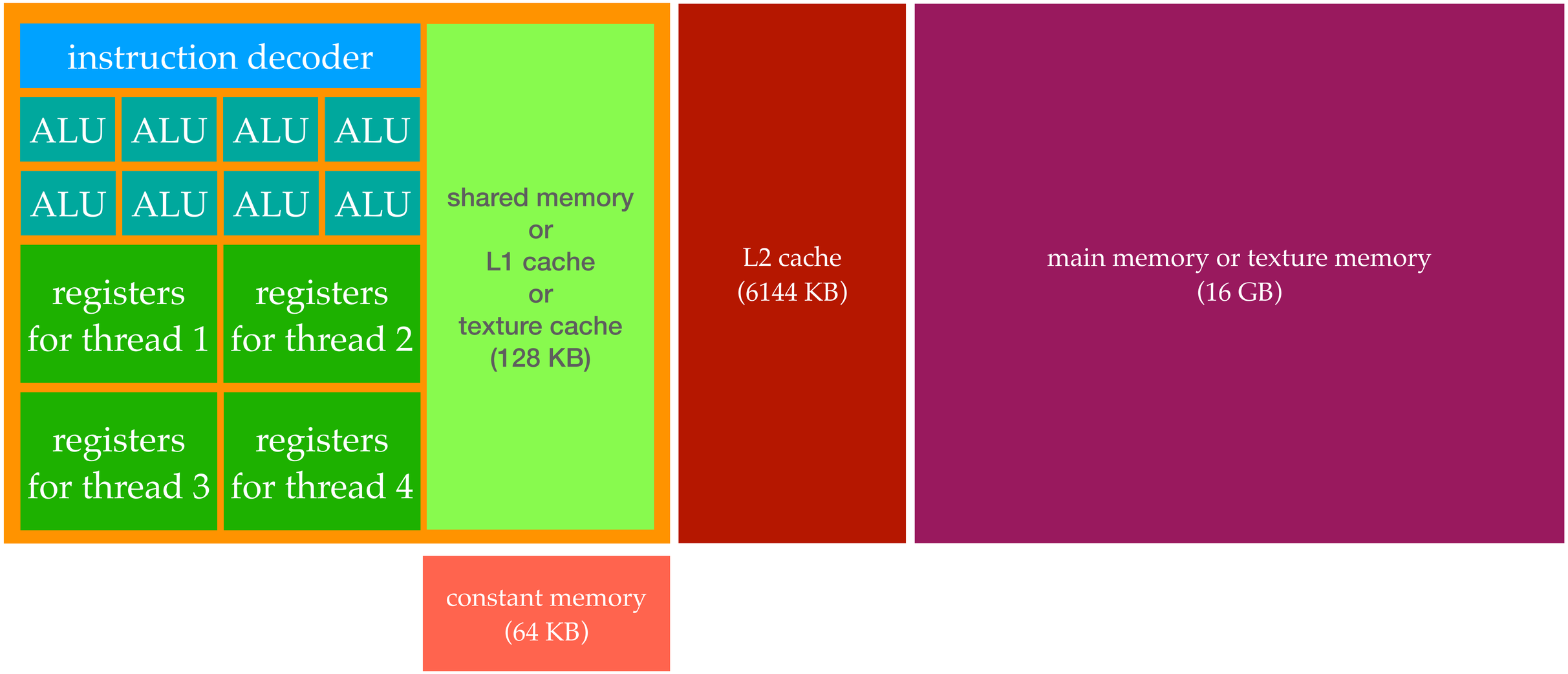
There are 80 SM cores on the V100:

That's 163,840 pieces of data being processed concurrently to get maximal latency hiding!



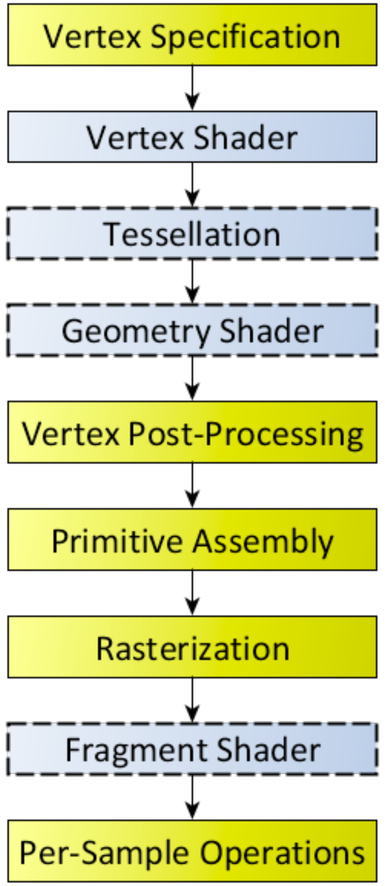
https://gfxcourses.stanford.edu/cs149/fall21/lecture/multicorearch/slide_83

GPU memory hierarchy



What about the graphics stuff?

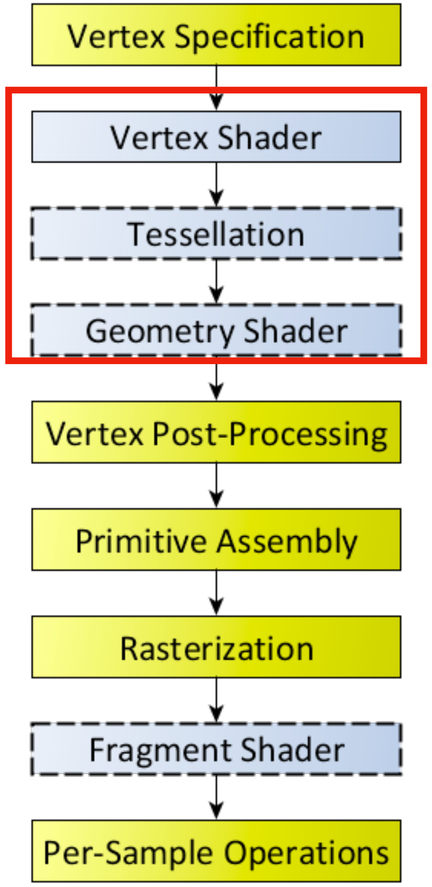
the graphics pipeline

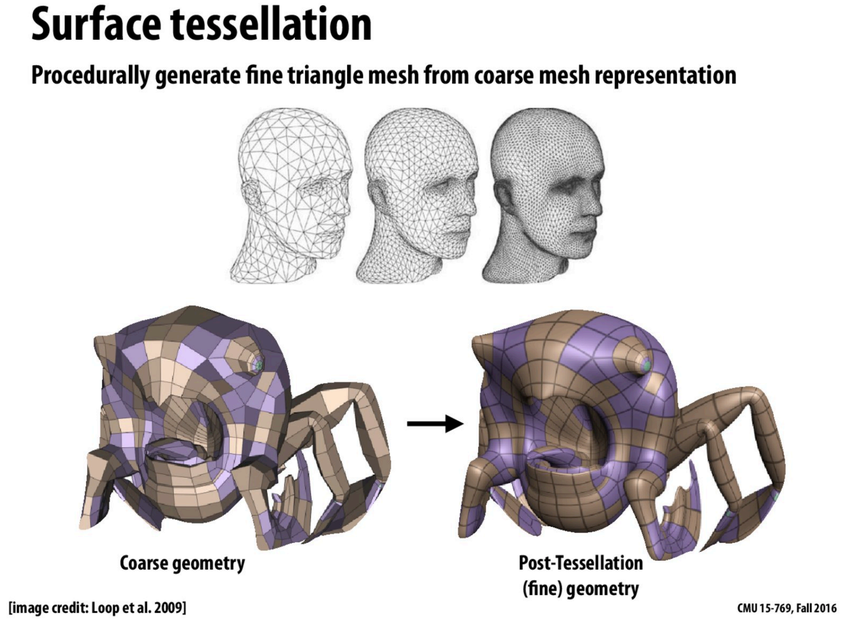


https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

What about the graphics stuff?

the graphics pipeline



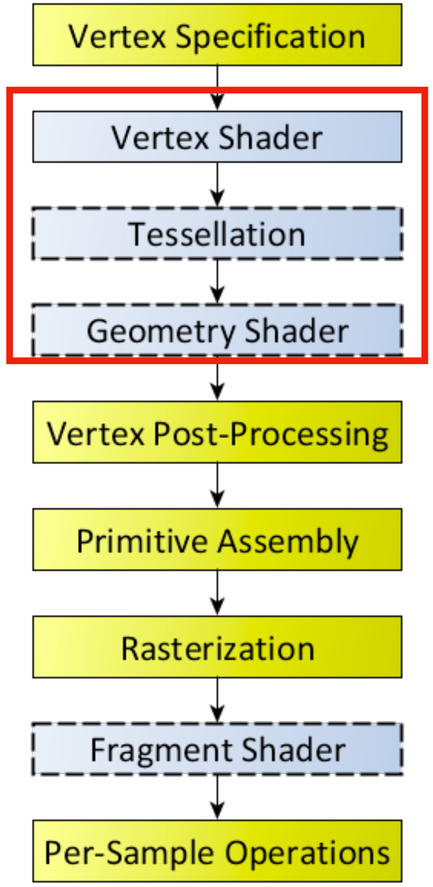


http://graphics.cs.cmu.edu/courses/15769/fall2016/lecture/schedulinggfx/slide_006

https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

What about the graphics stuff?

the graphics pipeline

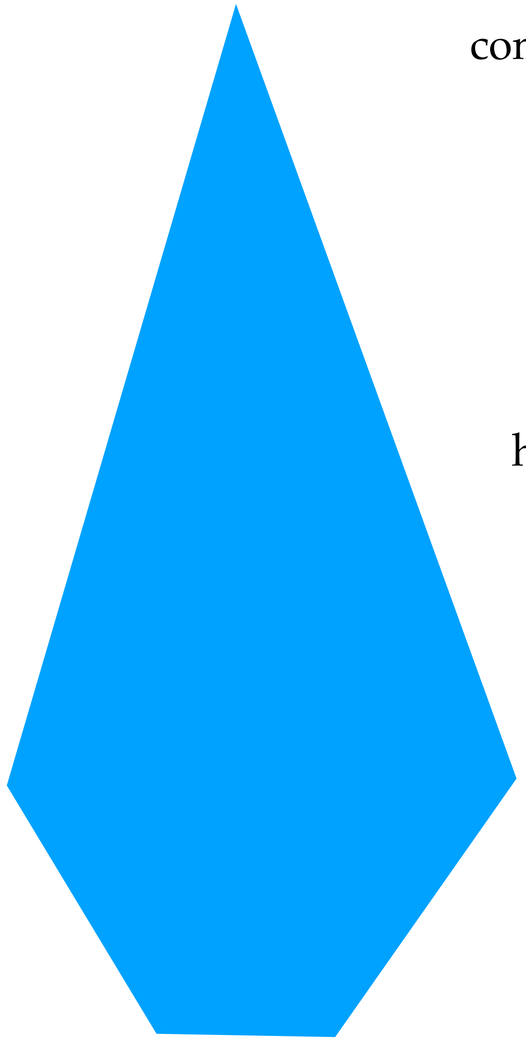


can be combined into a “mesh shader” (Nvidia Turing)

https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

Graphics workload: diamond structure

amount of data generated

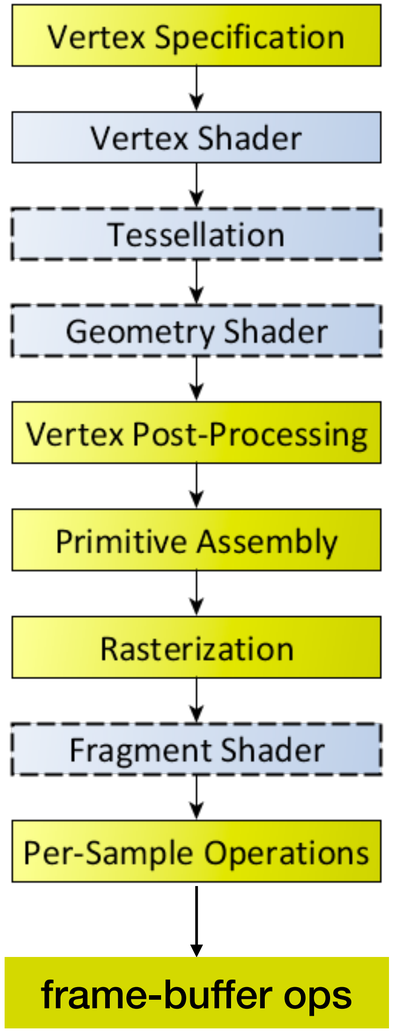


compact geometry

high-res mesh

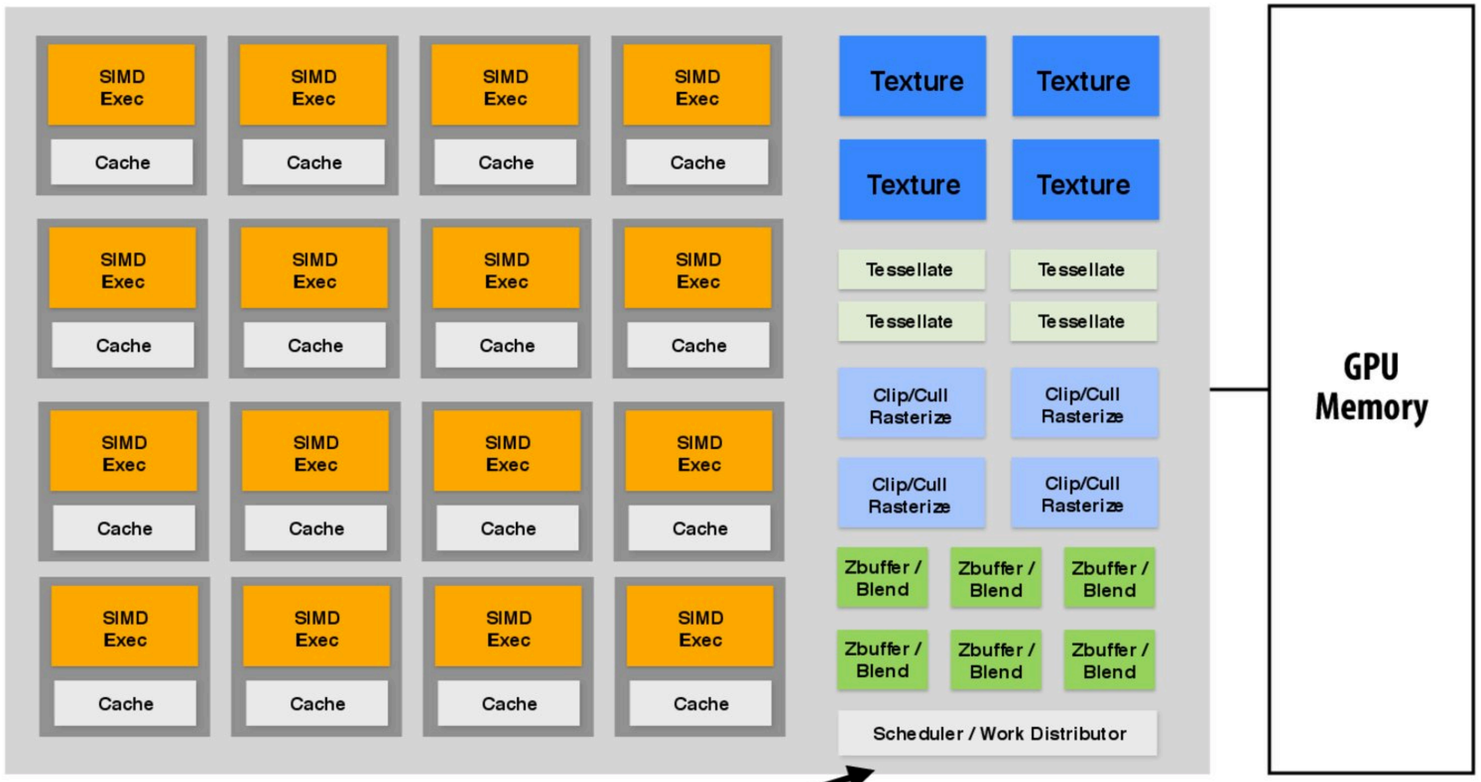
fragments

pixels



https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

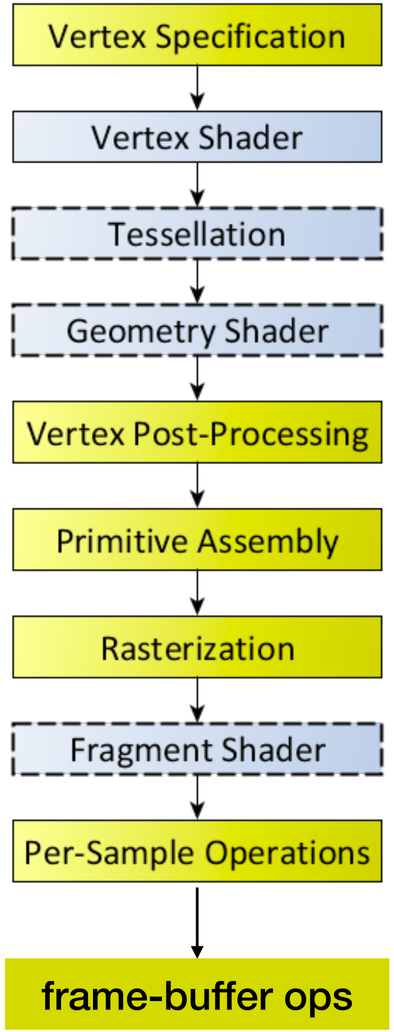
Parallelizing graphics pipeline

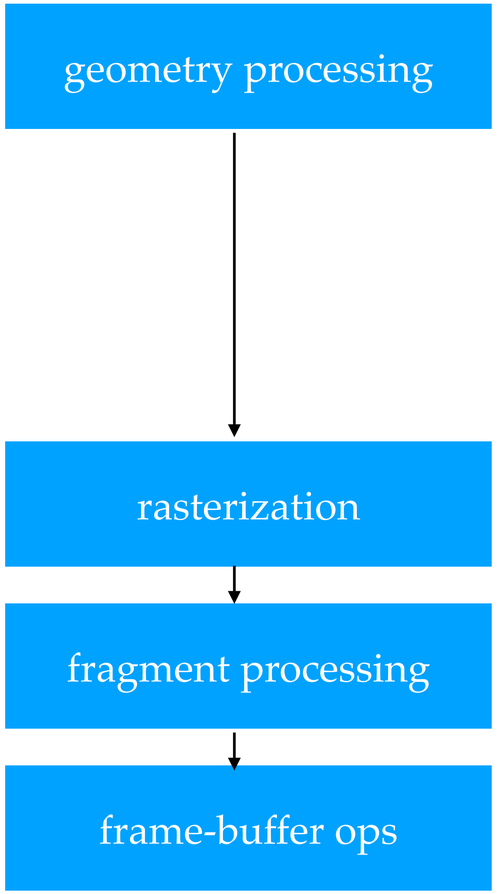


We're now going to talk about this scheduler

Simplifying a bit

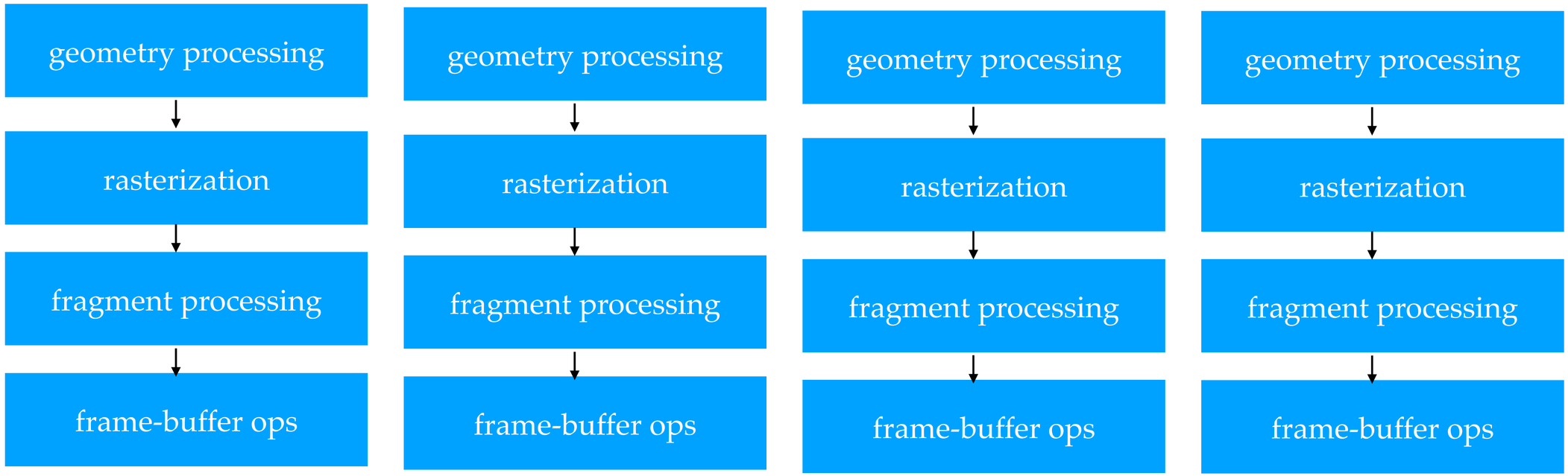
the graphics pipeline





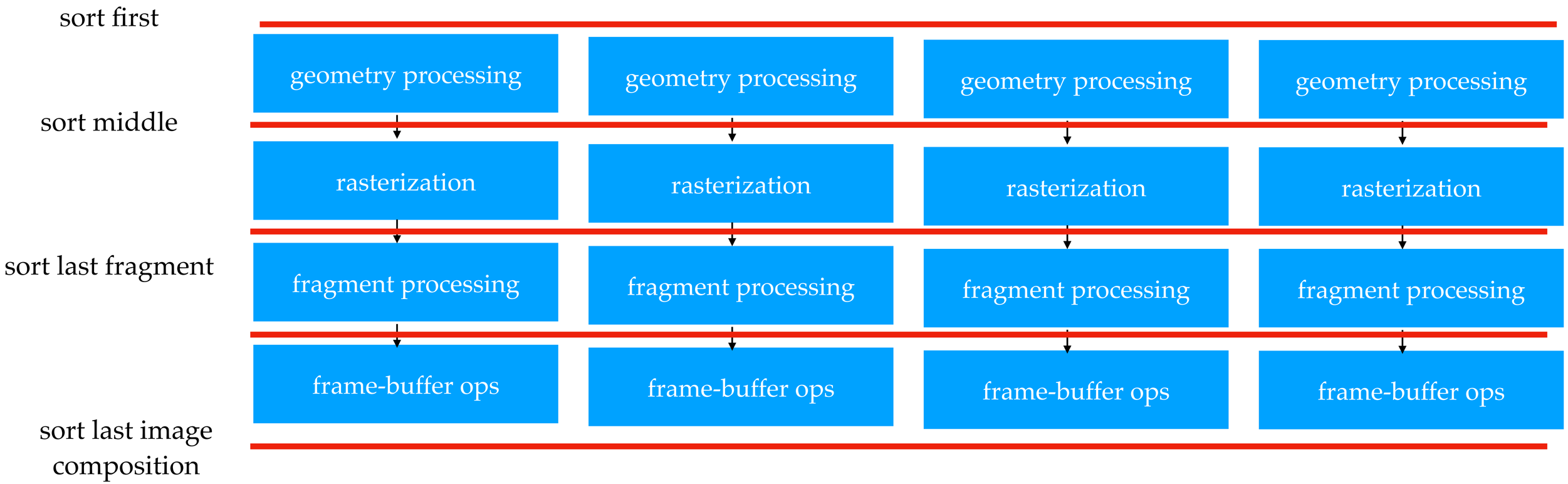
https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

Assume we have four threads



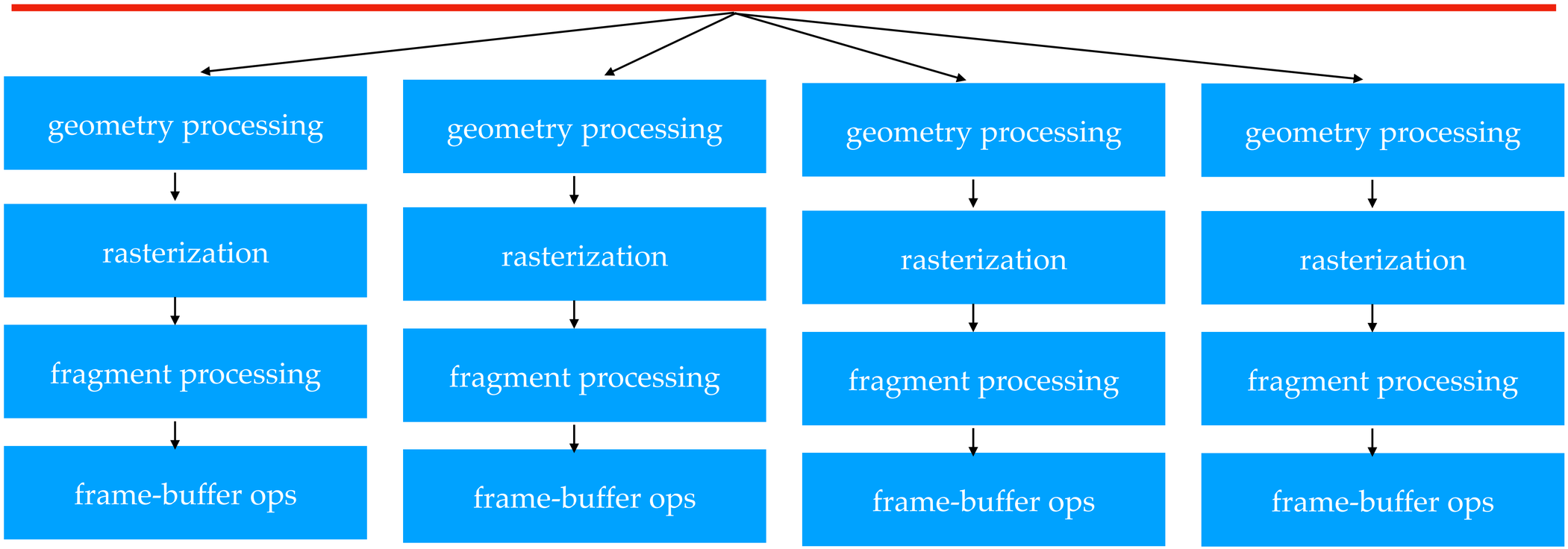
Molnar's sorting taxonomy

- how do we assign tasks to each thread?



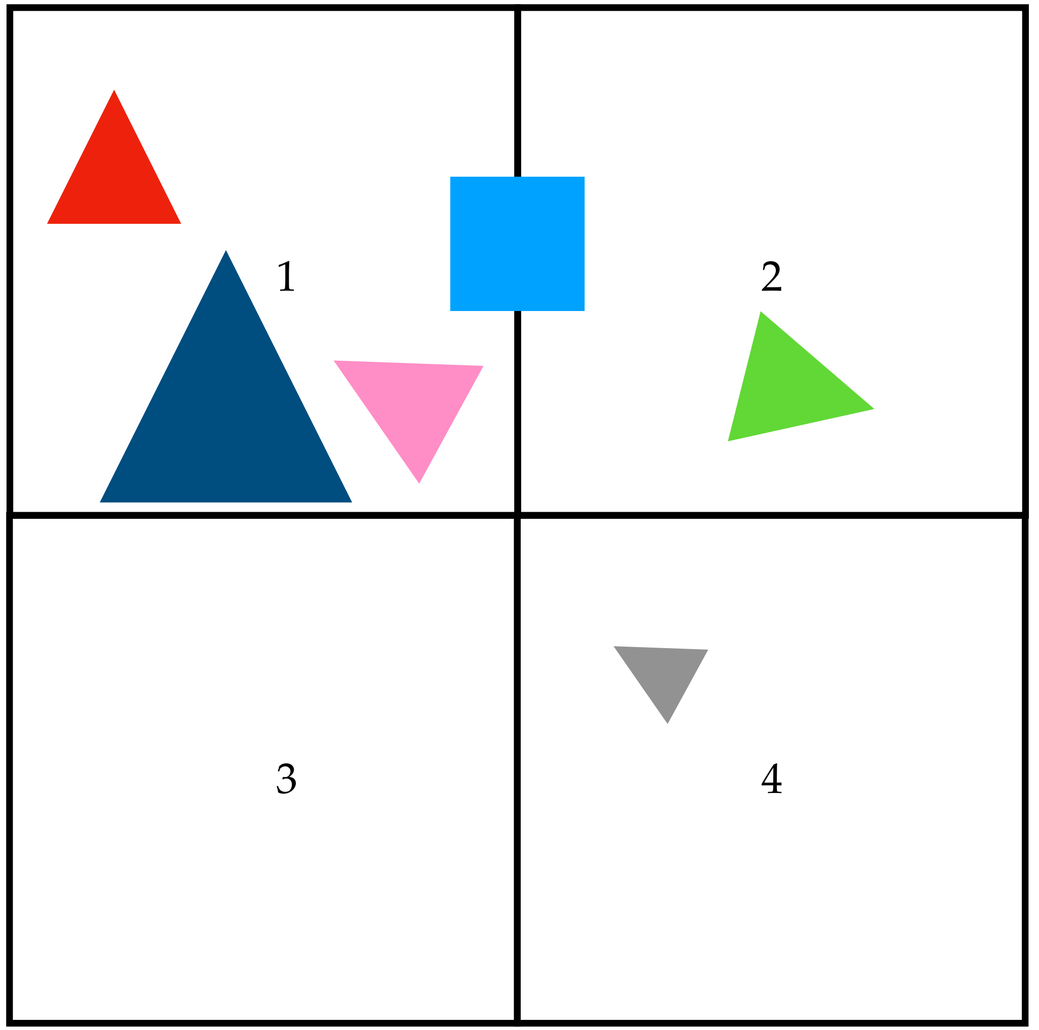
Sort first

- assign each pipeline for a region of the output image
- do minimal amount of work (compute screen-space positions of triangles) to determine which regions each primitive overlap



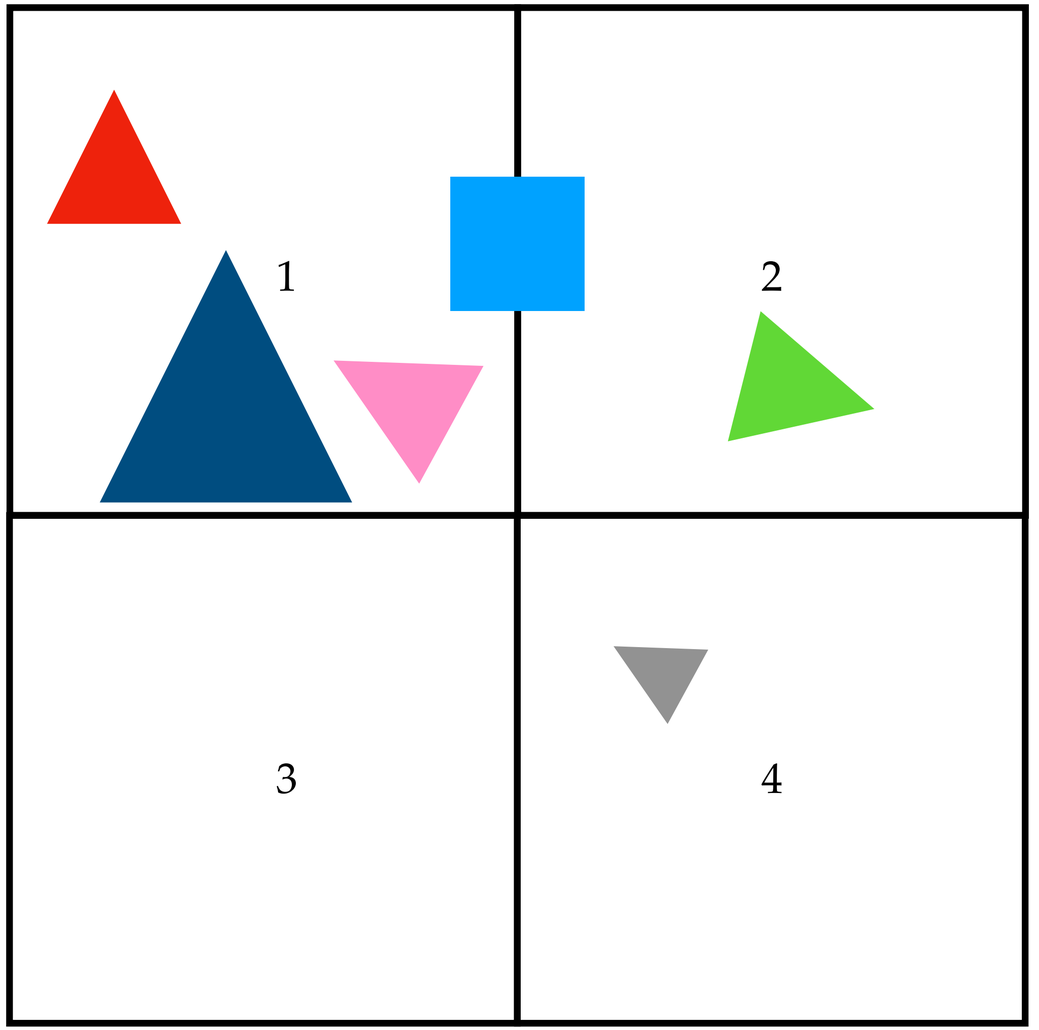
Sort first

- pros:
- cons:



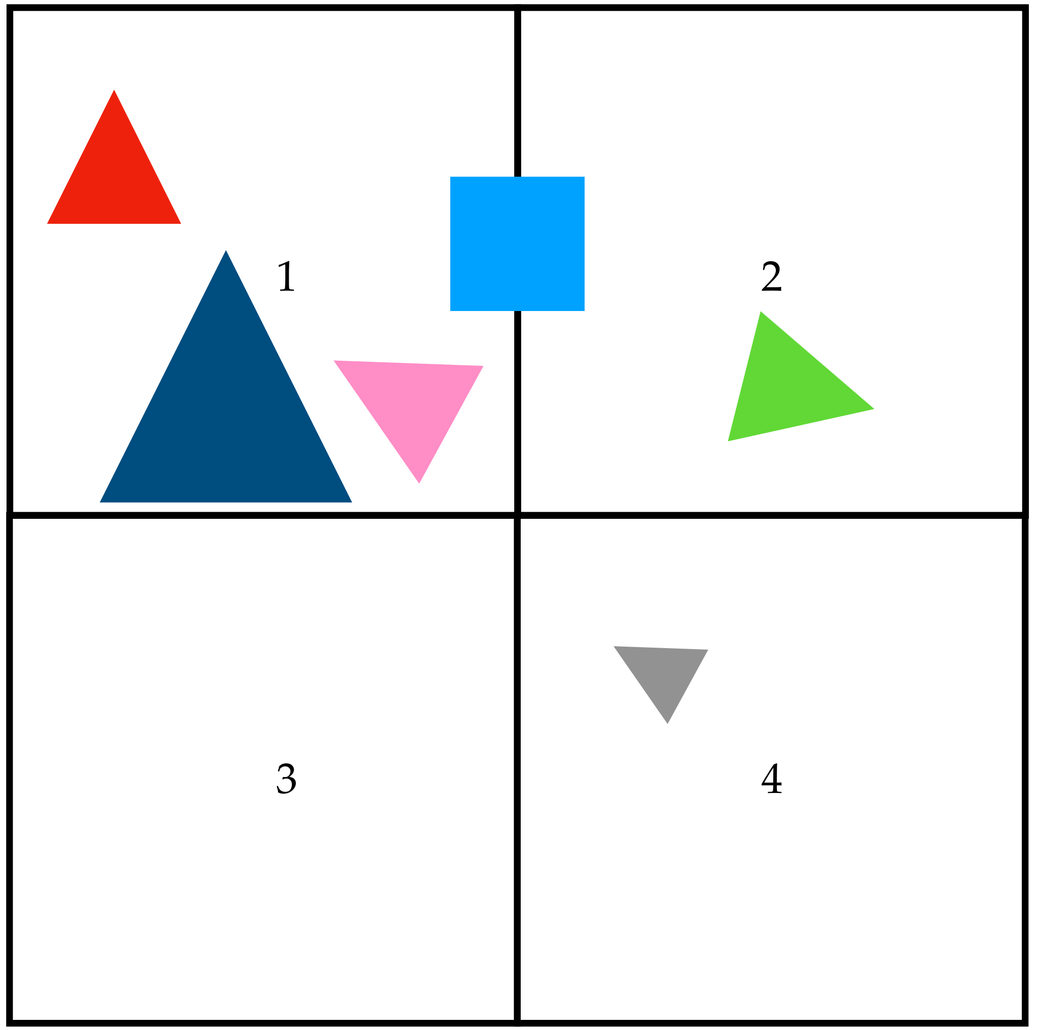
Sort first

- pros:
  - simple parallelization, no communication
  - early occlusion culling
- cons:



Sort first

- pros:
  - simple parallelization, no communication
  - early occlusion culling
- cons:
  - work balance
  - preprocessing time
  - duplicate work



Examples of sort first processors

WireGL: A Scalable Graphics System for Clusters

Greg Humphreys* Matthew Eldridge* Ian Buck* Gordon Stoll† Matthew Everett* Pat Hanrahan*

*Stanford University †Intel Corporation

Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters

Greg Humphreys* Mike Houston* Ren Ng* Randall Frank† Sean Ahern† Peter D. Kirchner‡
James T. Klosowski‡

*Stanford University †Lawrence Livermore National Laboratory ‡IBM T.J. Watson Research Center

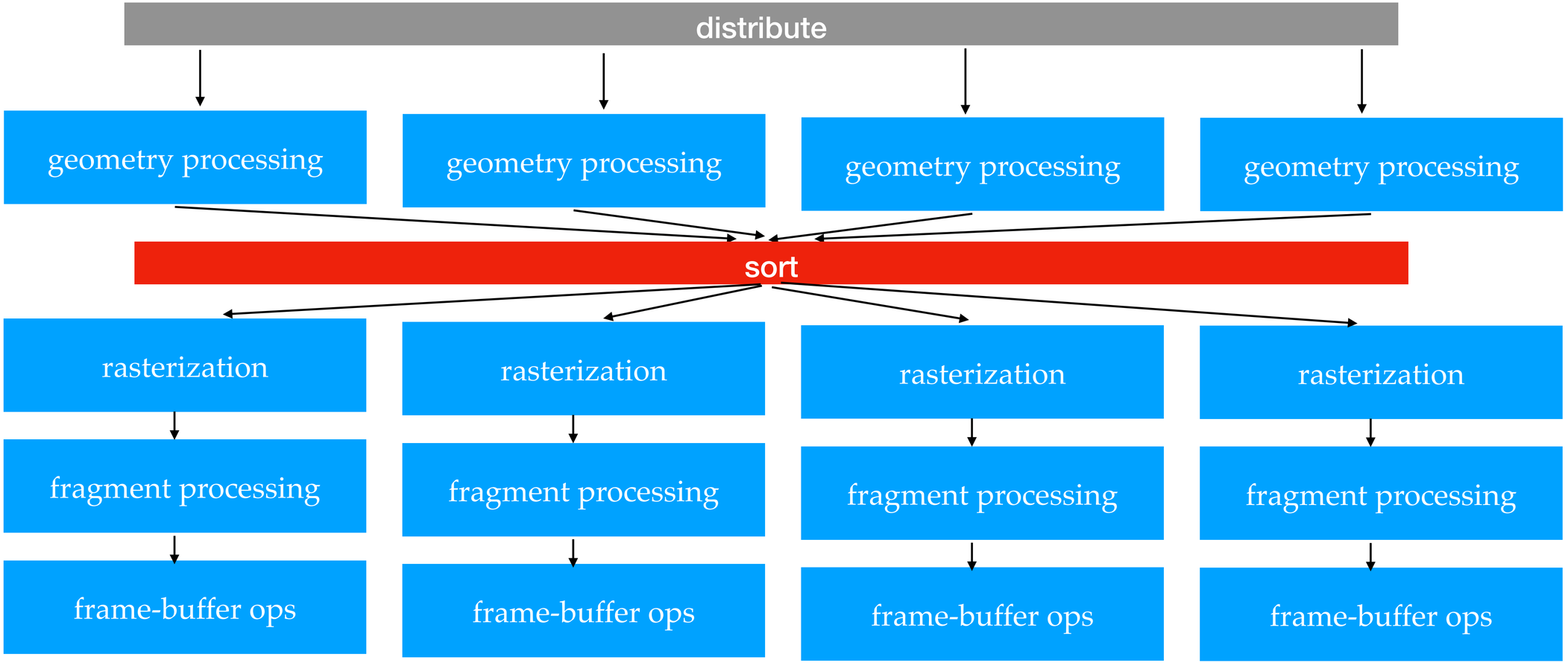
The Reyes Image Rendering Architecture

Robert L. Cook
Loren Carpenter
Edwin Catmull

Pixar
P. O. Box 13719
San Rafael, CA 94913

Sort middle

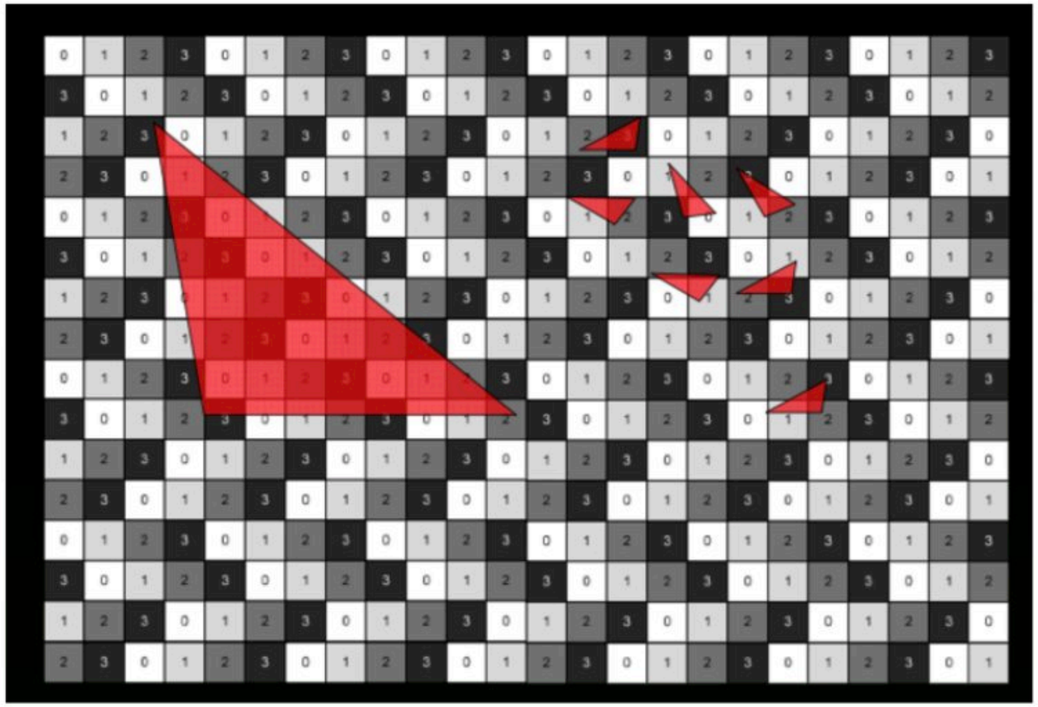
- distribute primitives to pipelines (e.g., round robin)
- assign each rasterizer a region of the image
- assign primitives to rasterizers based on region overlap



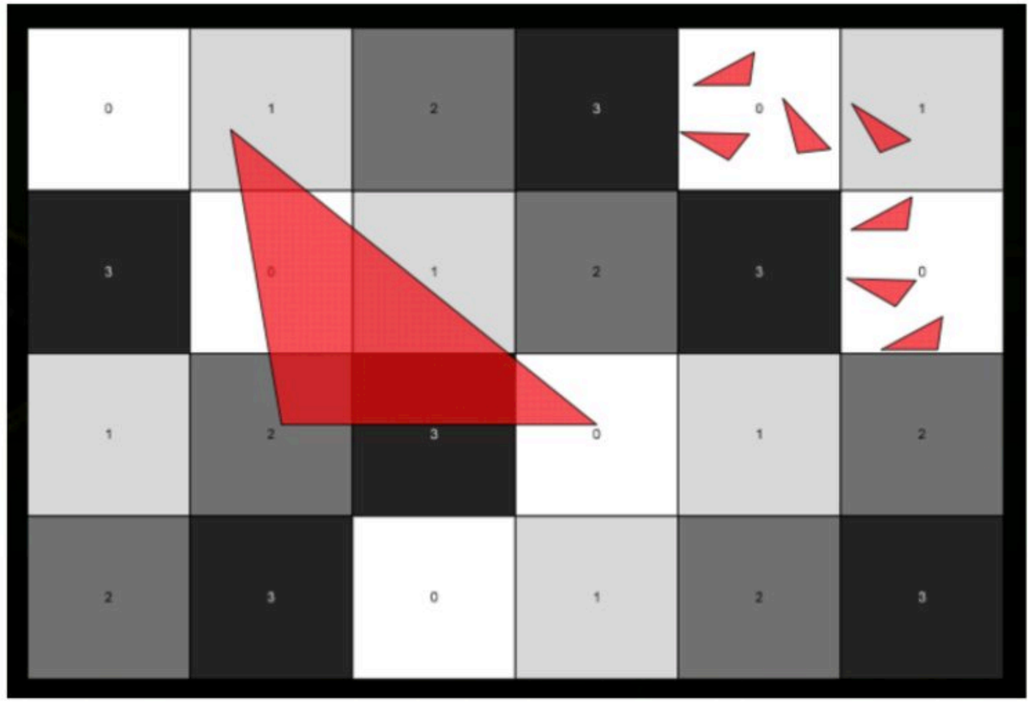
How do we assign regions on image?

- fragment interleaving in NVIDIA Fermi

fine granularity interleaving

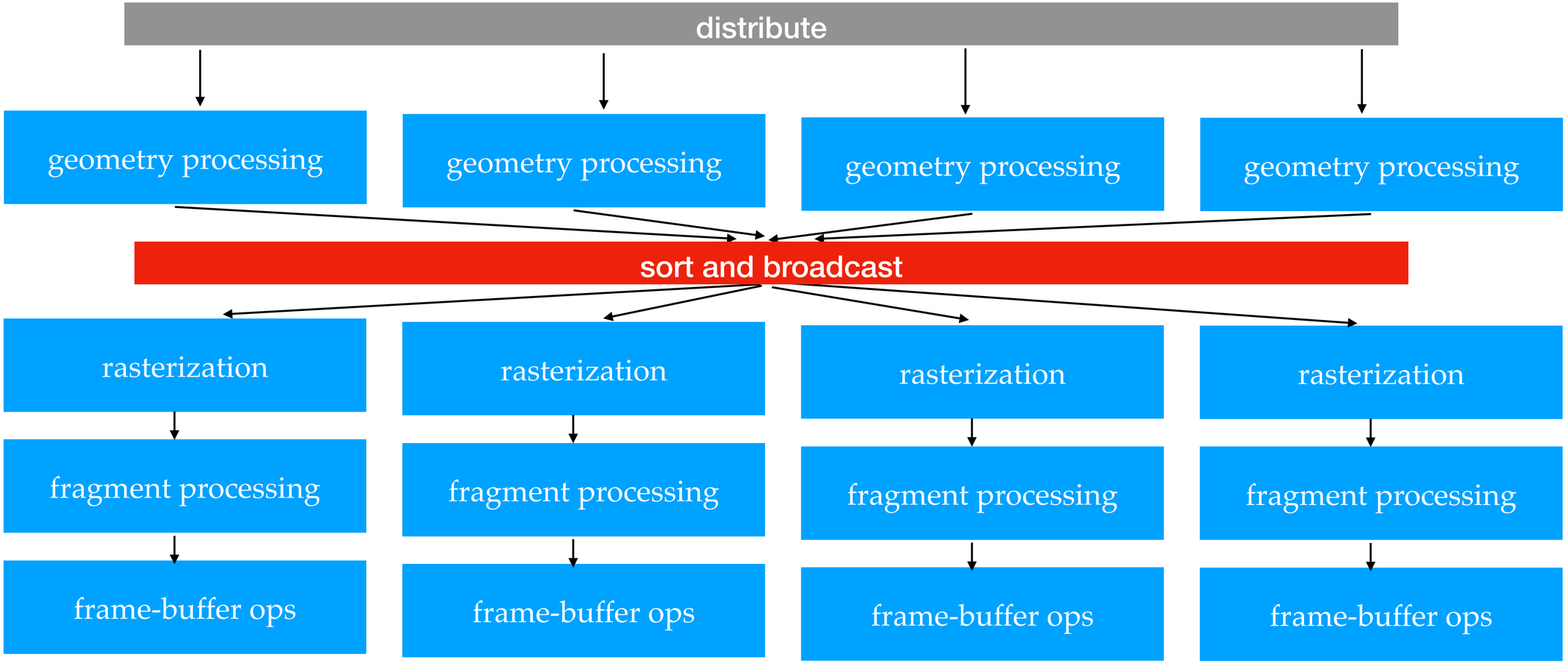


coarse granularity interleaving



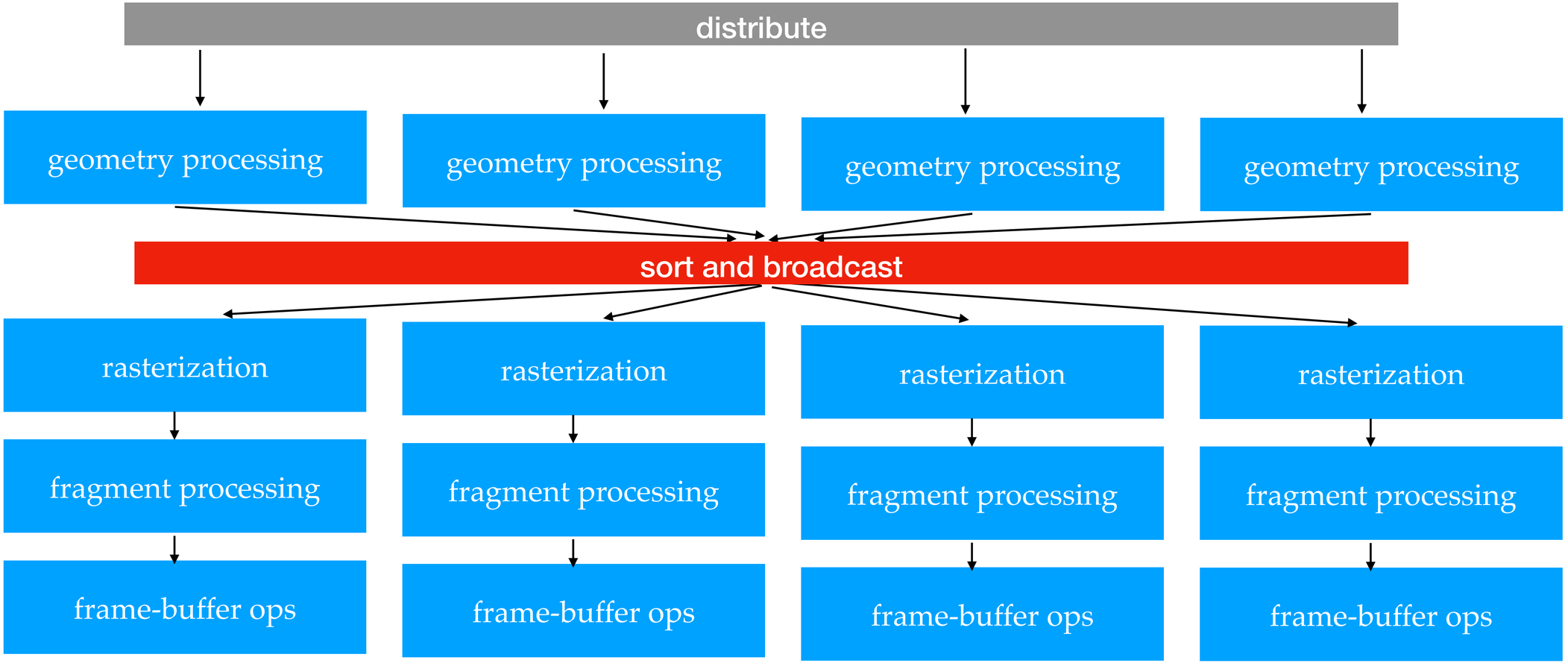
Sort middle interleaved

- pros:
- cons:



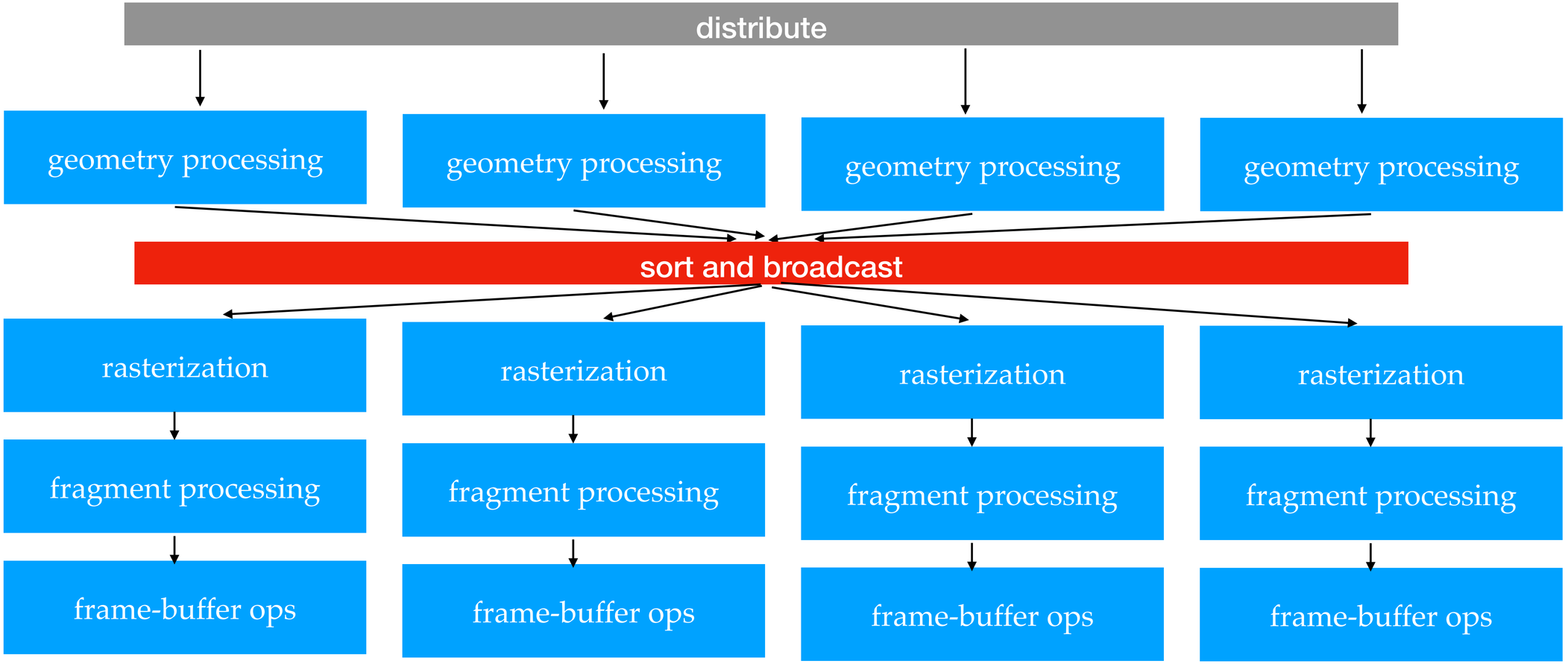
Sort middle interleaved

- pros: workload balance, no duplicate work for geometry processing
- cons:



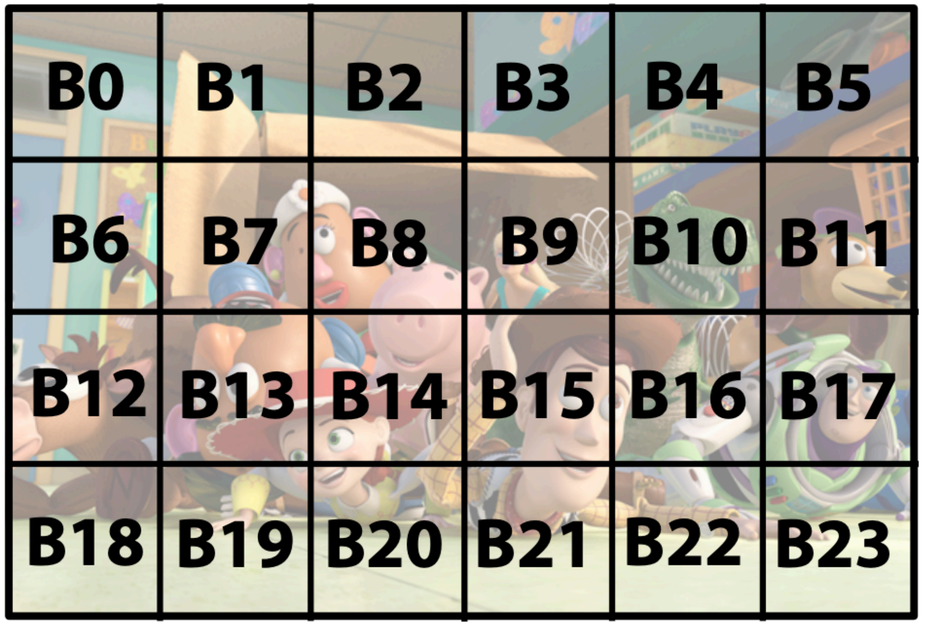
Sort middle interleaved

- pros: workload balance, no duplicate work for geometry processing
- cons: communication cost (larger cost if meshes are highly tessellated)
  duplicate work for rasterization



Sort middle tiled

Interleaved (static) assignment of screen tiles to processors

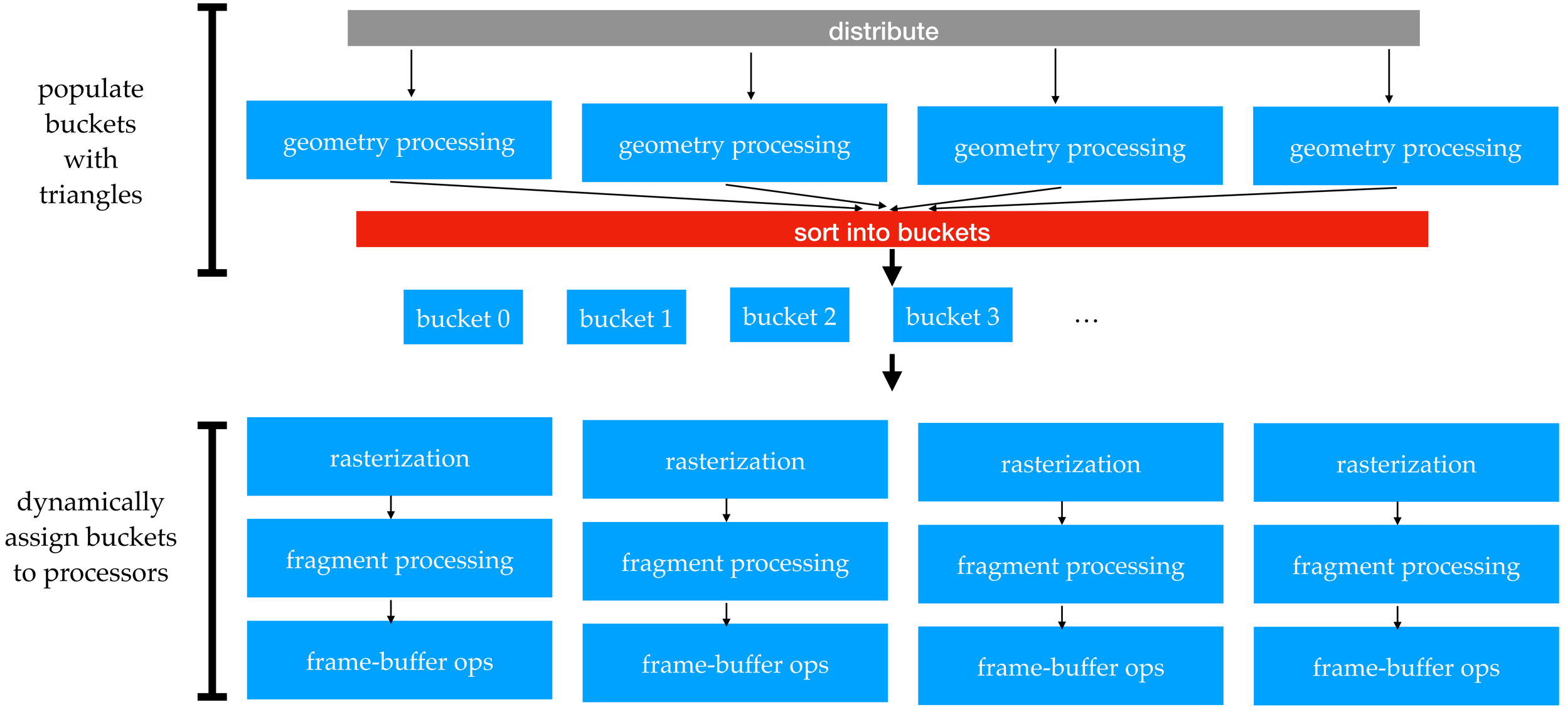


Assignment to buckets

List of buckets is a work queue. Buckets are dynamically assigned to processors.

https://graphics.stanford.edu/courses/cs348v-18-winter/lectures/17_schedulinggfx.pdf

Sort middle tiled



Sort middle tiled

- pros:
- cons:

Sort middle tiled

- pros:
  - good load balance
  - low bandwidth requirement
- cons:
  - slightly more overhead compared to sort middle interleaved
  - duplicate work for rasterization

Most recent mobile GPUs are sort-middle tiled

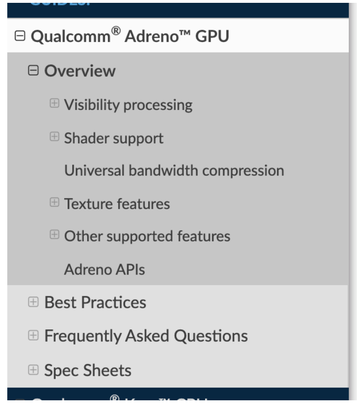
- due to the low memory bandwidth requirement if tile sizes are roughly the sizes of polygons

A look at the PowerVR graphics architecture: Tile-based rendering

Tile based rendering

Mali GPUs use *tile-based deferred* rendering.

The Mali GPU divides the framebuffer into tiles and renders it tile by tile. Tile based rendering is efficient because values for pixels are computed using an on-chip memory. This technique is ideal for mobile devices because it requires less memory bandwidth and less power than traditional rendering techniques.



Tile-based rendering

To optimize rendering for low-power and memory-bandwidth-limited devices, Adreno GPUs use a tiled-based rendering architecture. This rendering mechanism breaks the scene frame buffer into small rectangular regions for rendering. Region sizes are automatically determined so that they are optimally rendered using local, low-latency memory on the GPU (referred to as GMEM), rather than using a bandwidth-limited bus to system memory.

The deferred mode rendering mechanism of the Adreno GPU also uses a tile-based rendering architecture. It implements a binning approach to create bins of primitives that are processed in each tile.

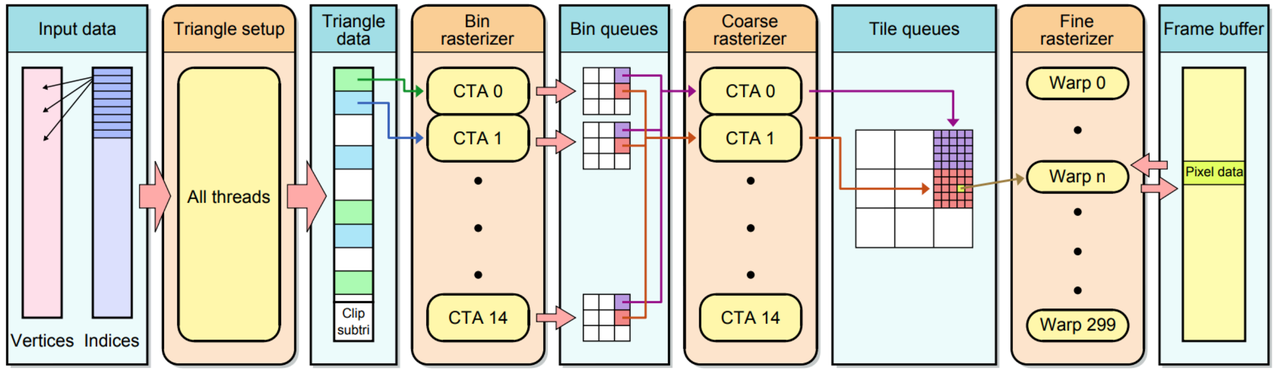
The Adreno GPU divides a frame into bins and renders them one at a time. During rendering.it uses on-chip high performance Graphics Memory (GMEM) to avoid the cost of going to system memory.

In the image below, you see the two passes that are performed over the graphic primitives (Binning and Rendering). In this example there are three triangles that will be rendered in the frame buffer. The Binning Pass marks which bins a triangle is visible in (visiblty stream). This stream is stored to system memory.

Software GPU rasterizers are often sort-middle tiled

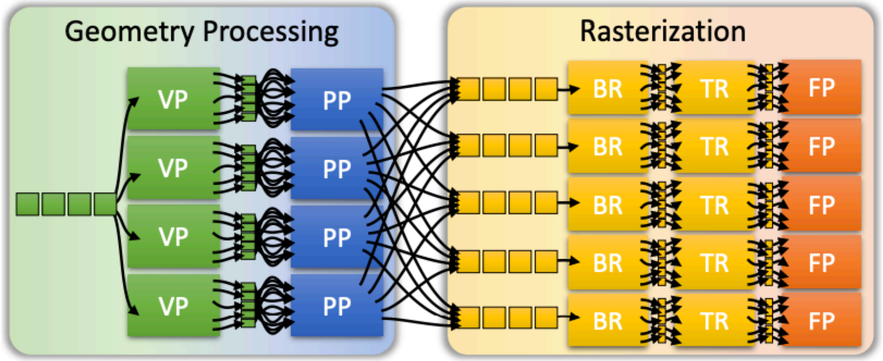
High-Performance Software Rasterization on GPUs

Samuli Laine Tero Karras
NVIDIA Research*



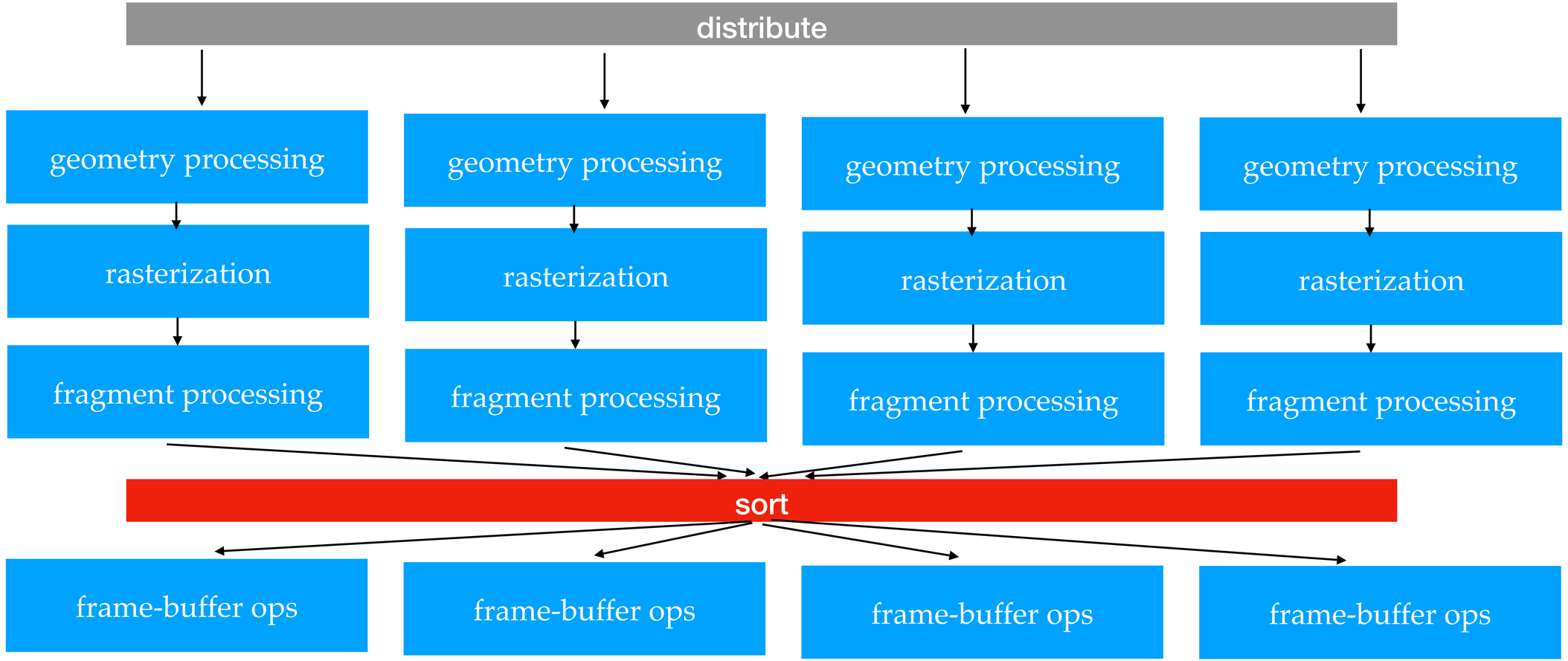
A High-Performance Software Graphics Pipeline Architecture for the GPU

MICHAEL KENZEL, BERNHARD KERBL, and DIETER SCHMALSTIEG, Graz University of Technology, Austria
MARKUS STEINBERGER, Graz University of Technology, Austria and Max Planck Institute for Informatics, Germany



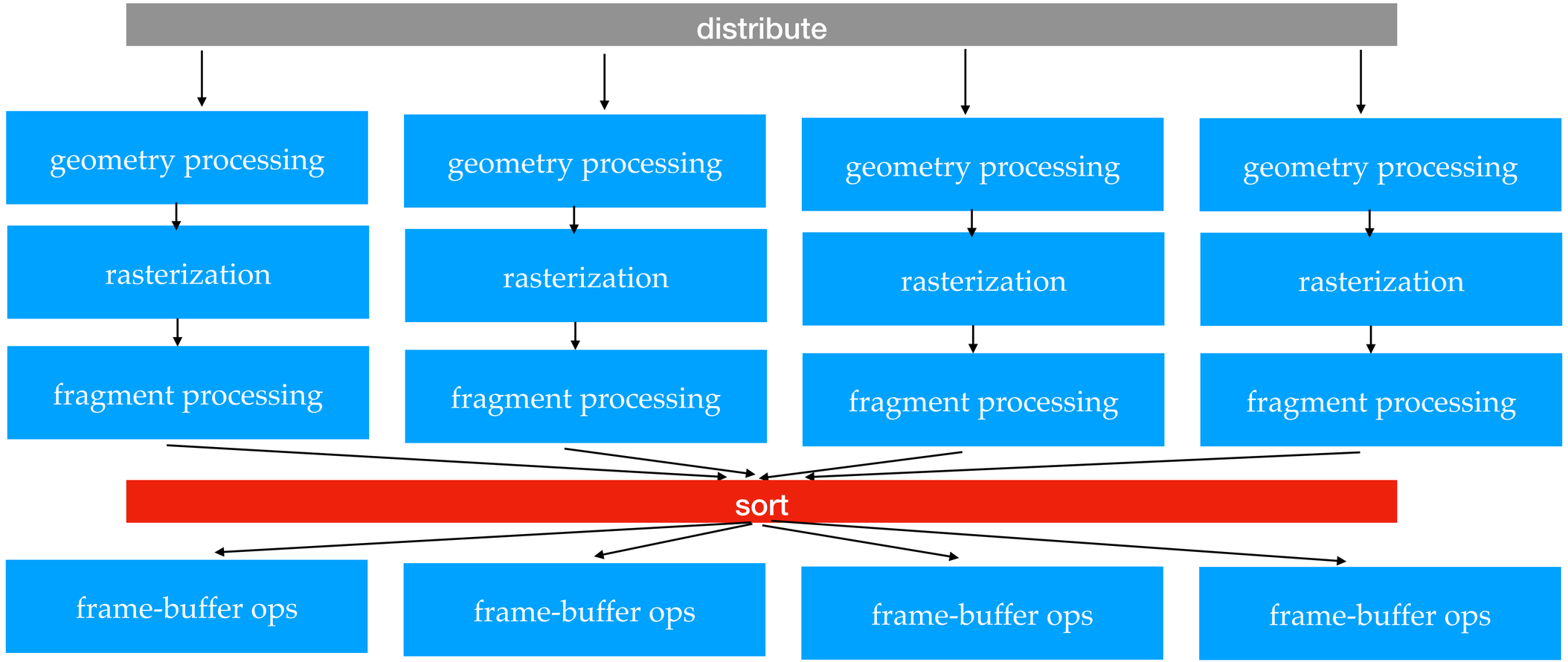
Sort last fragment

- distribute primitives to pipelines (e.g., round robin)
- sort fragments based on their (x, y) positions



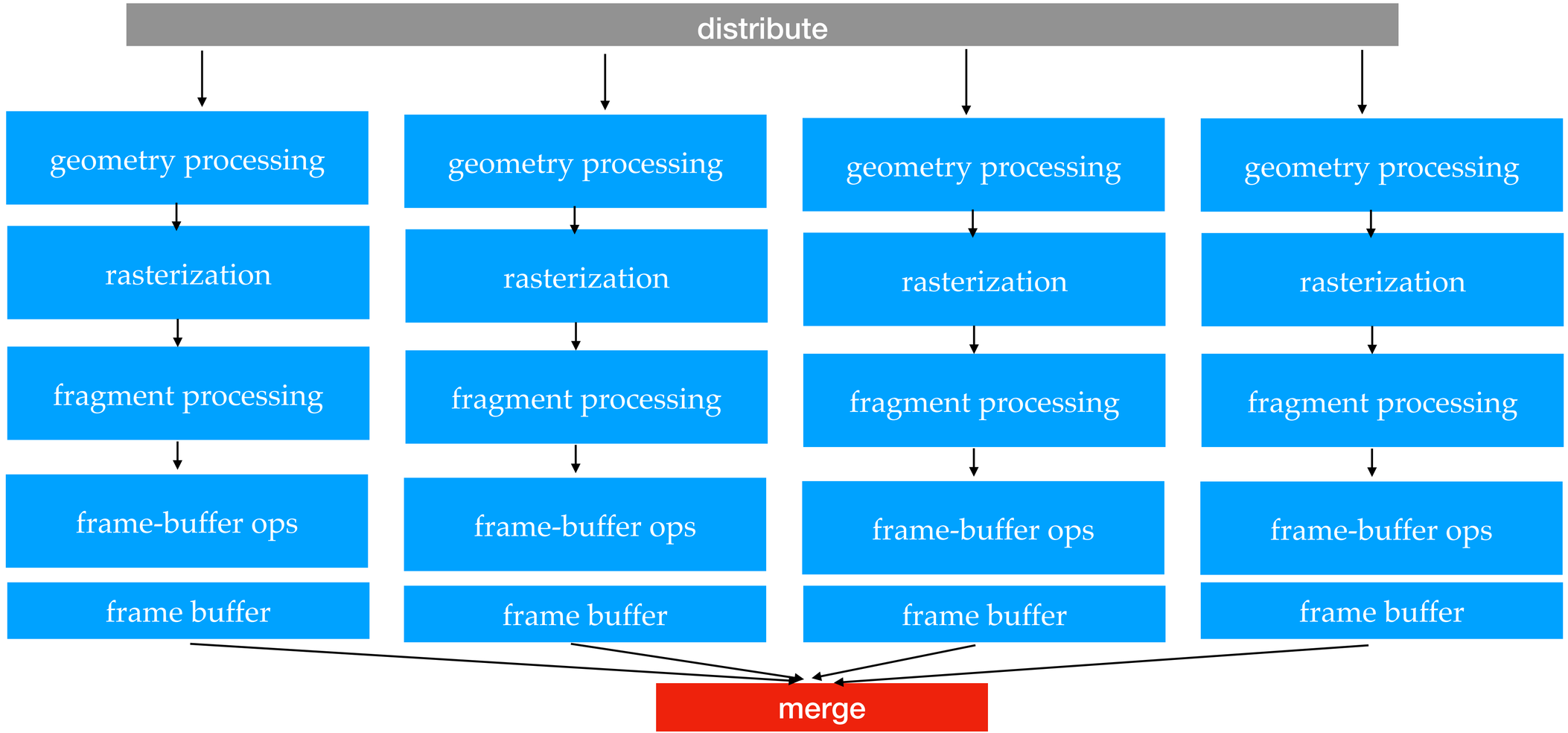
Sort last fragment

- pros: no duplicate work, good work balance for frame-buffer ops
- cons: early z culling is tricky, bad sorting memory use (many more fragments than triangles)

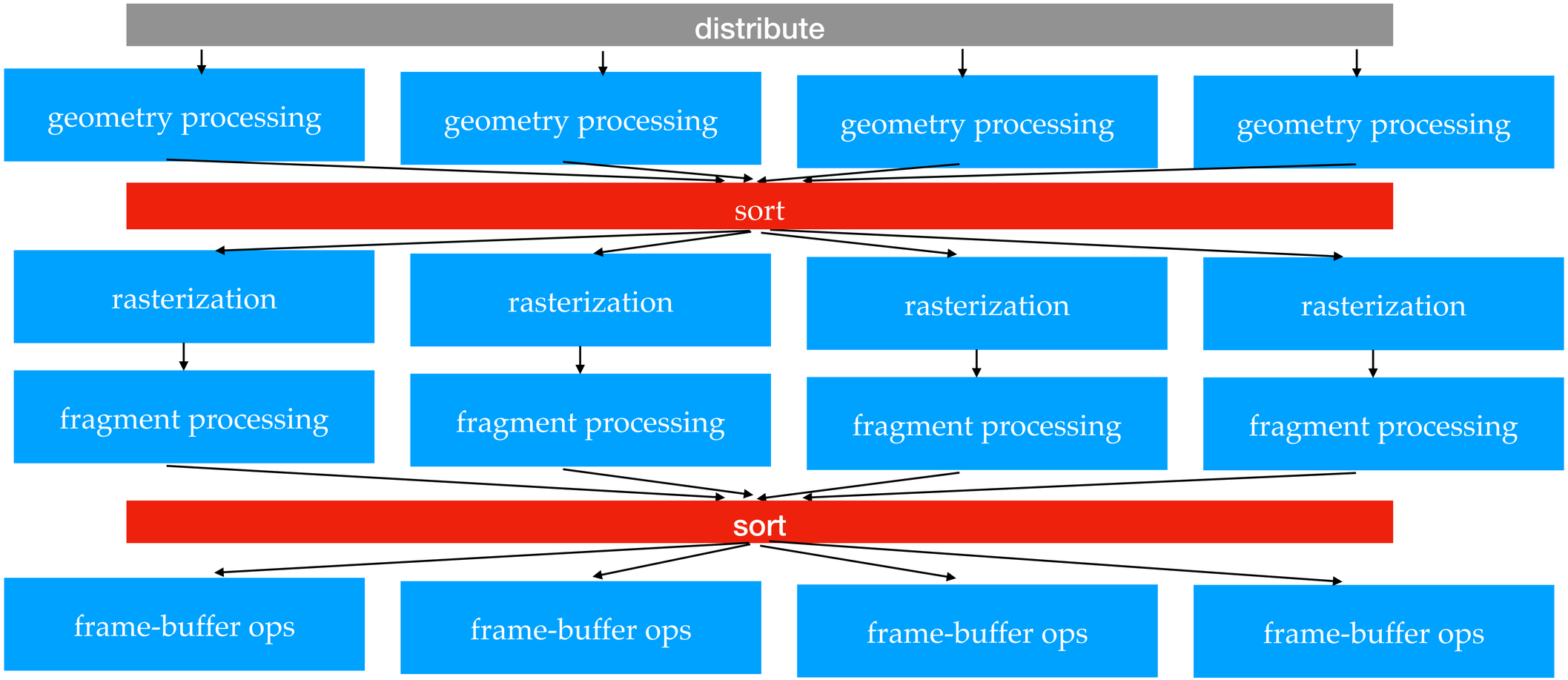


Sort last image composition

- each pipeline render some geometry
- merge them in the end based on depth



Modern GPUs: sort everywhere

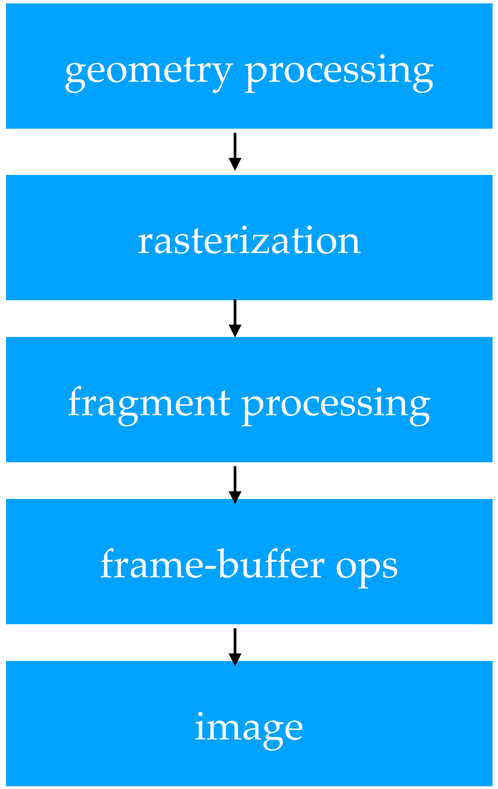


Pomegranate: A Fully Scalable Graphics Architecture

Matthew Eldridge Homan Igehy Pat Hanrahan

Stanford University*

Where should lighting be done?



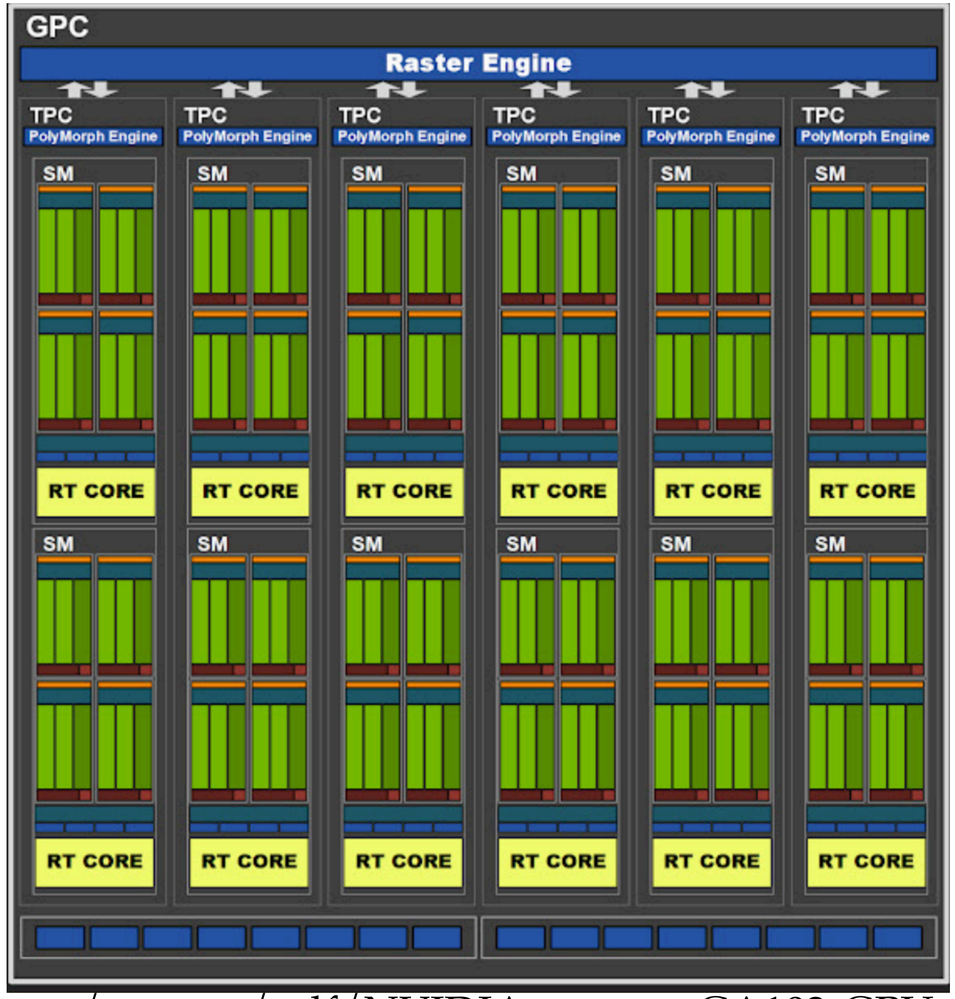
do lighting here: Gouraud shading

do lighting here: Phong shading

do lighting here: deferred shading

Ray tracing

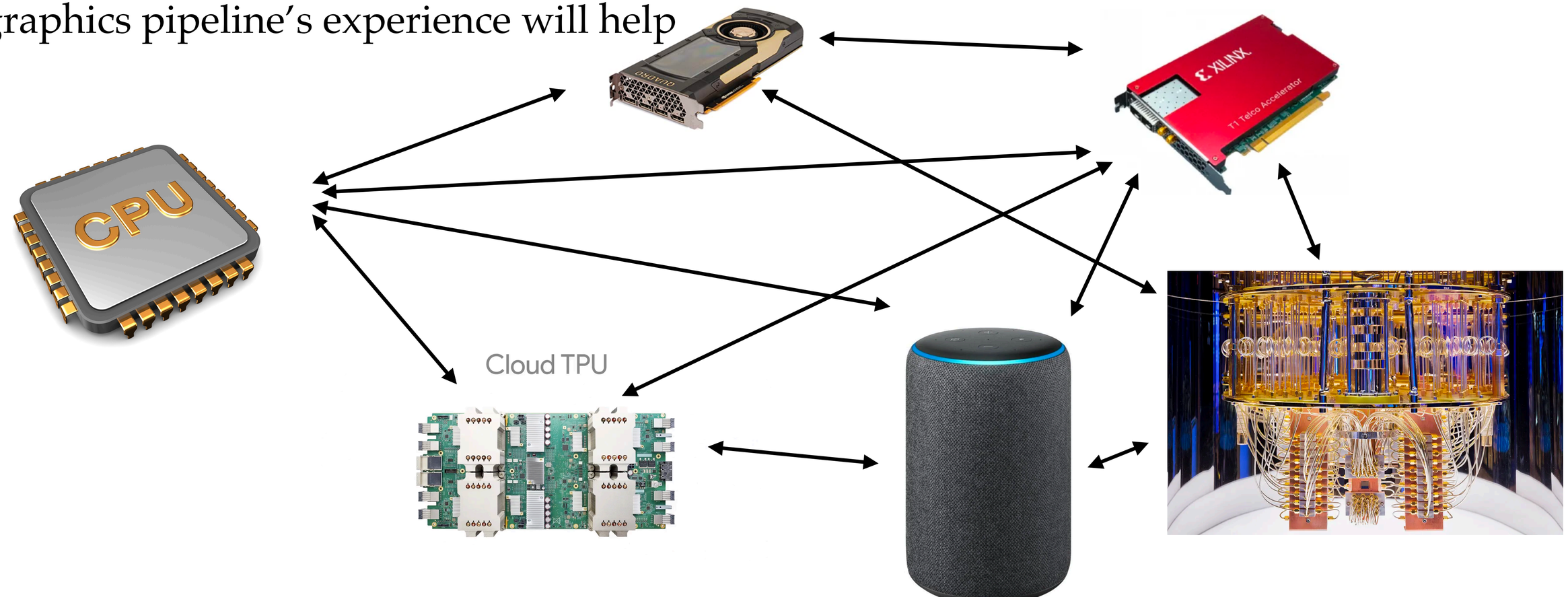
- very little detail has been released
- hardware is only responsible for BVH traversal & scene geometry



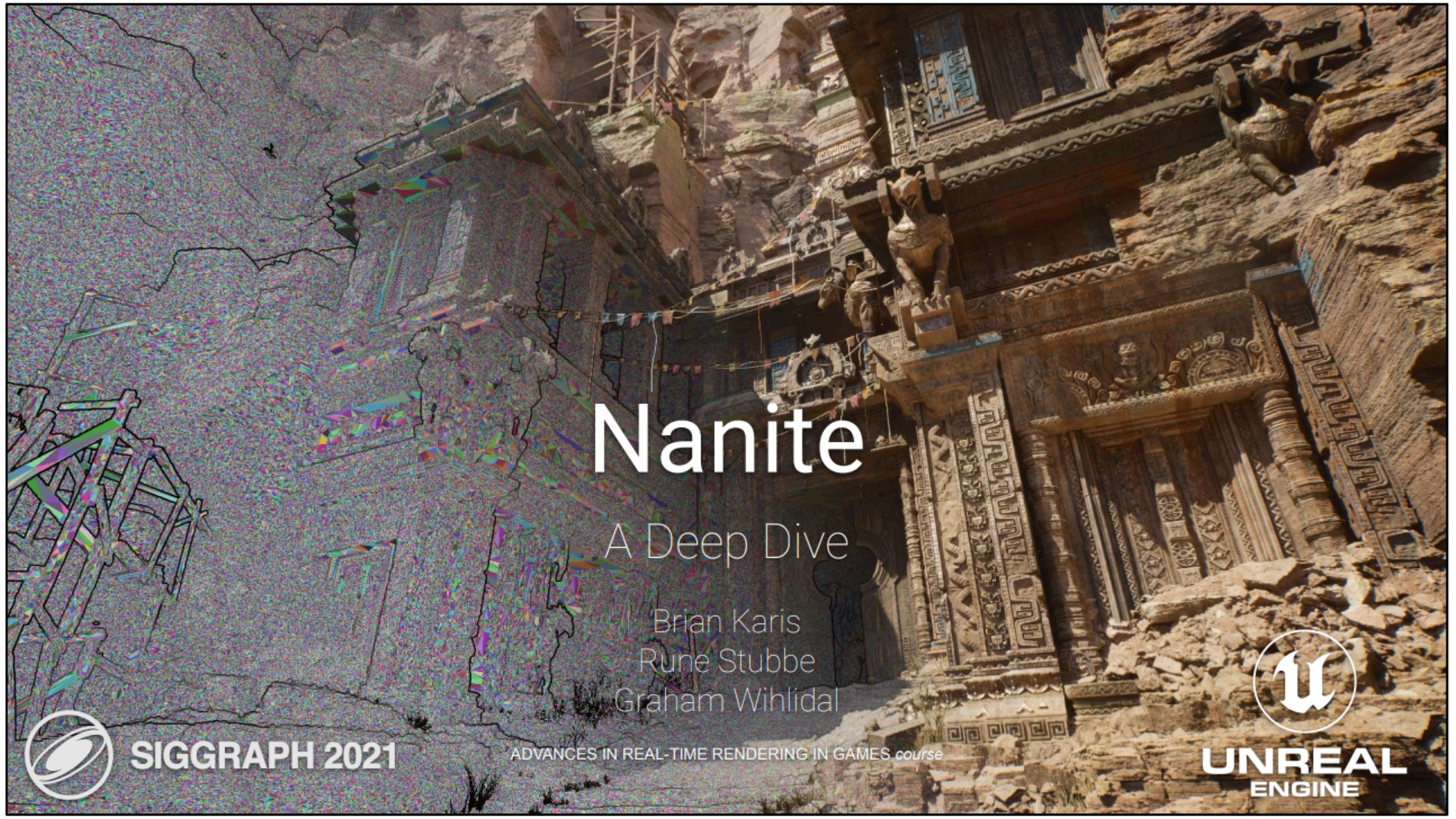
https://images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf

The dawn of Moore's law = domain-specific hardware

- writing software / systems for heterogeneous hardware will become a necessary skill
- graphics pipeline's experience will help



Next: Nanite


Nanite
A Deep Dive
Brian Karis
Rune Stubbe
Graham Wihlidal
SIGGRAPH 2021
ADVANCES IN REAL-TIME RENDERING IN GAMES course
UNREAL
ENGINE