CUDA - Parallel Computing on GPUs

Richard Membarth

richard.membarth@cs.fau.de

Hardware-Software-Co-Design
University of Erlangen-Nuremberg

19.03.2009

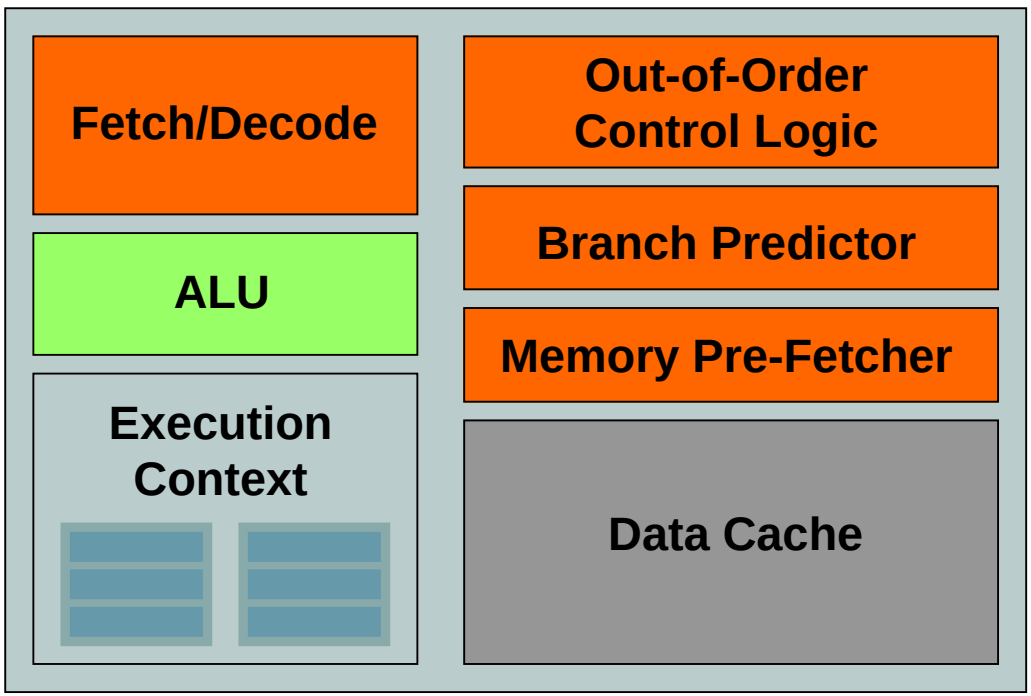
Outline

- Key concepts behind modern graphics architectures
- Understanding these allows to
  - Understand difference between GPU and CPU code
  - Optimize compute kernels
  - Get feeling for what kind of tasks might benefit from GPUs

CPU vs. GPU Hardware

CPU

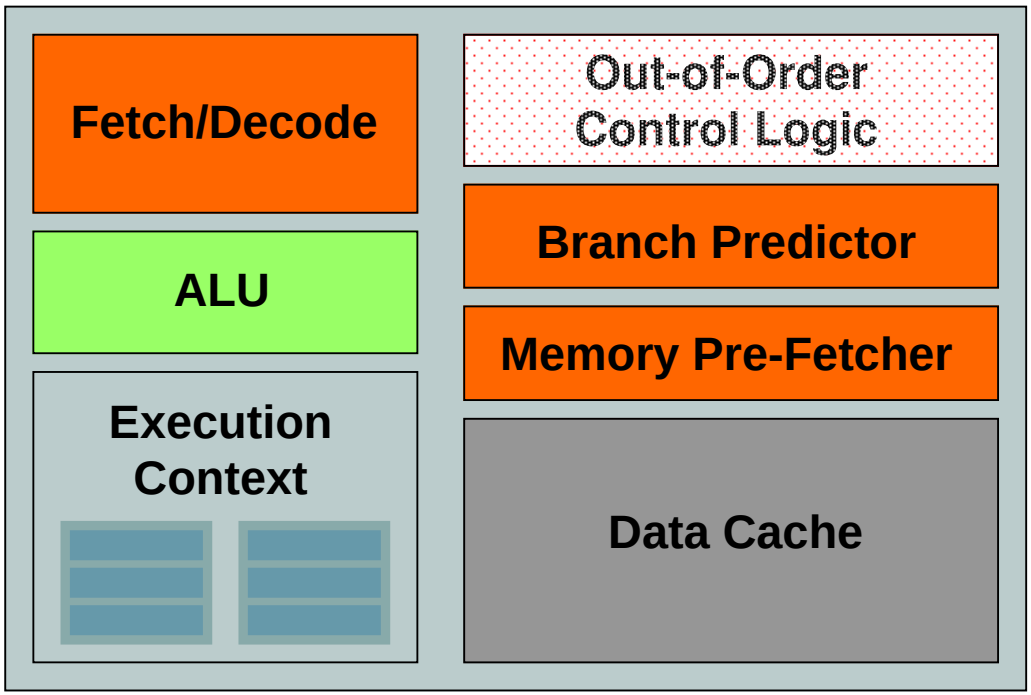
- Little space spent on execution units
- Optimized for latency



CPU vs. GPU Hardware

GPU

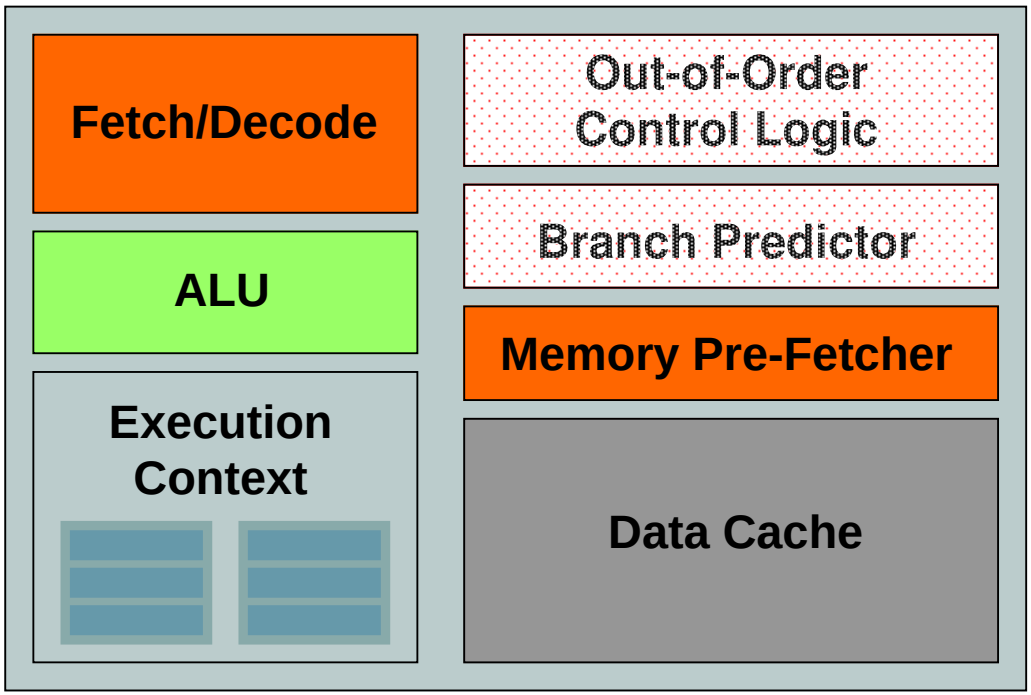
- Remove components that help a single instruction stream run faster



CPU vs. GPU Hardware

GPU

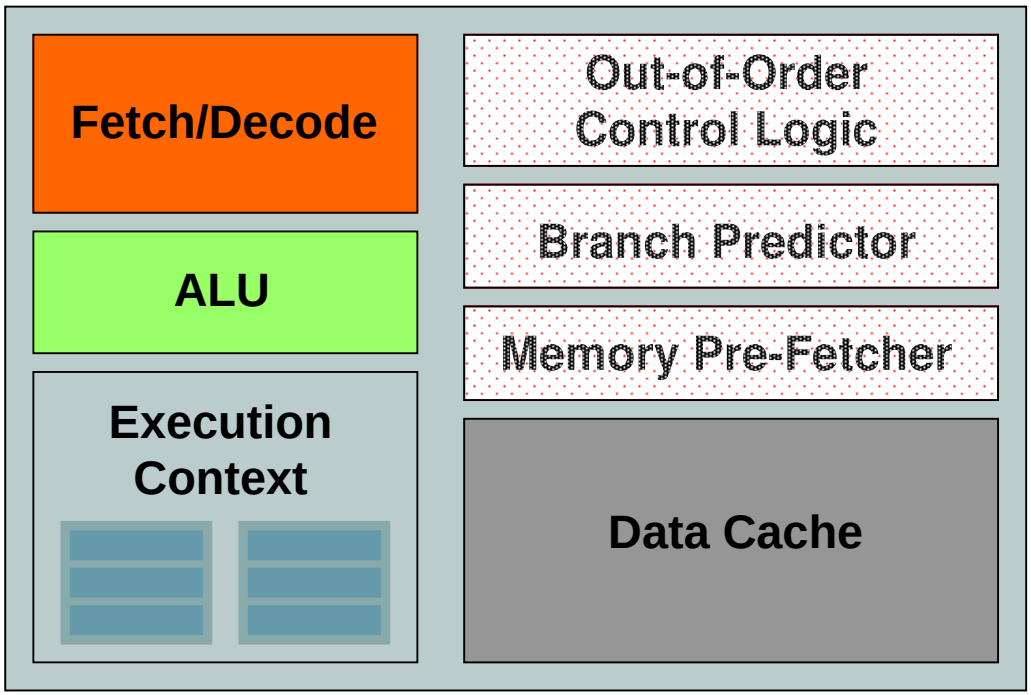
- Remove components that help a single instruction stream run faster



CPU vs. GPU Hardware

GPU

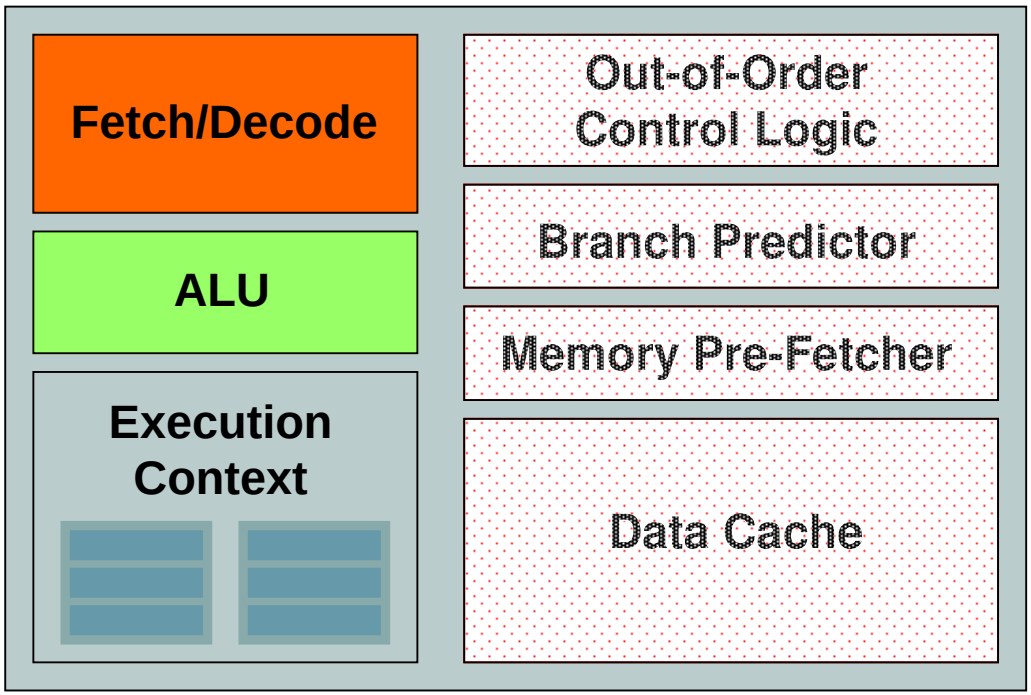
- Remove components that help a single instruction stream run faster



CPU vs. GPU Hardware

GPU

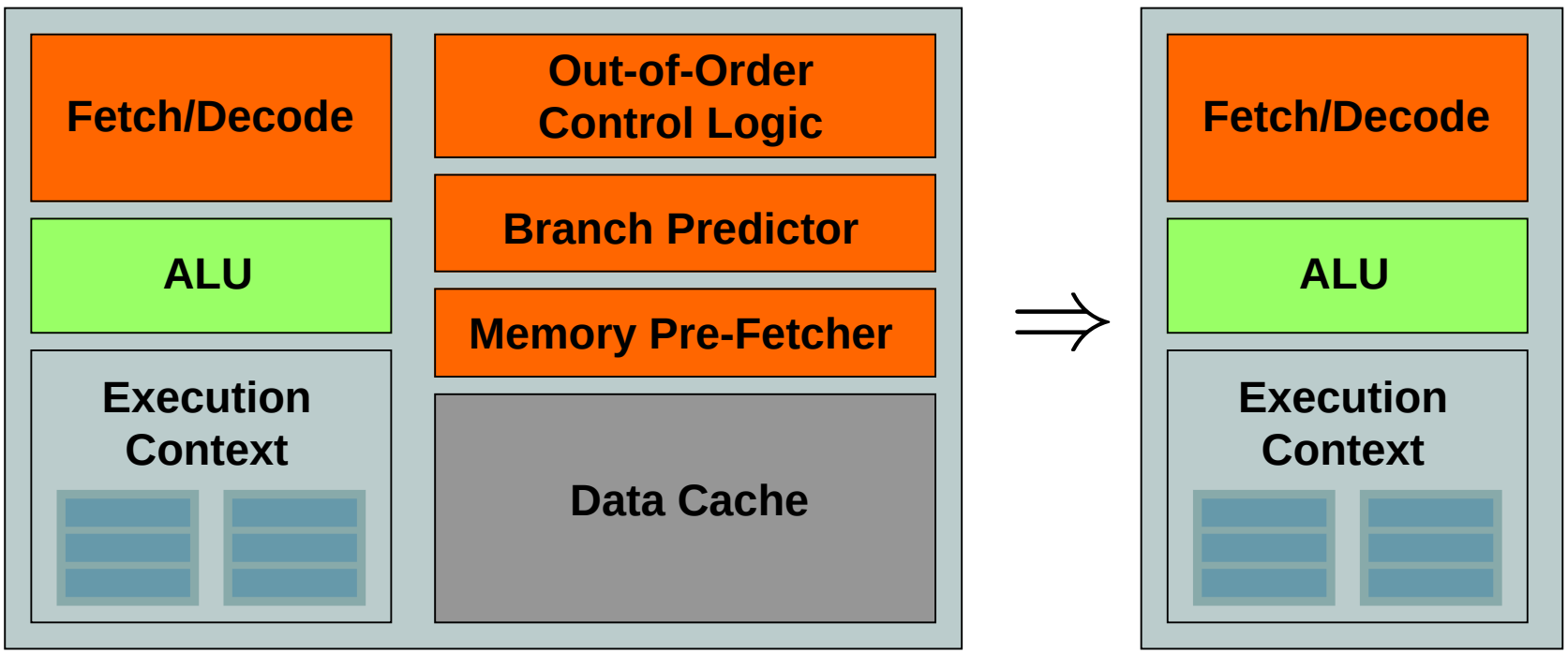
- Remove components that help a single instruction stream run faster



CPU vs. GPU Hardware

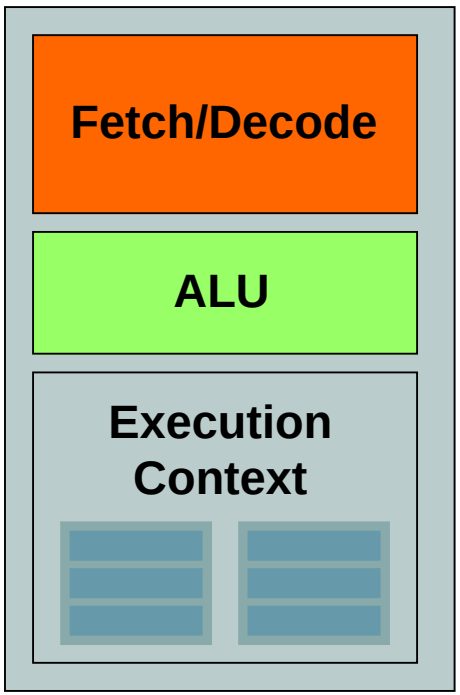
GPU

- Remove components that help a single instruction stream run faster



GPU Parallelism

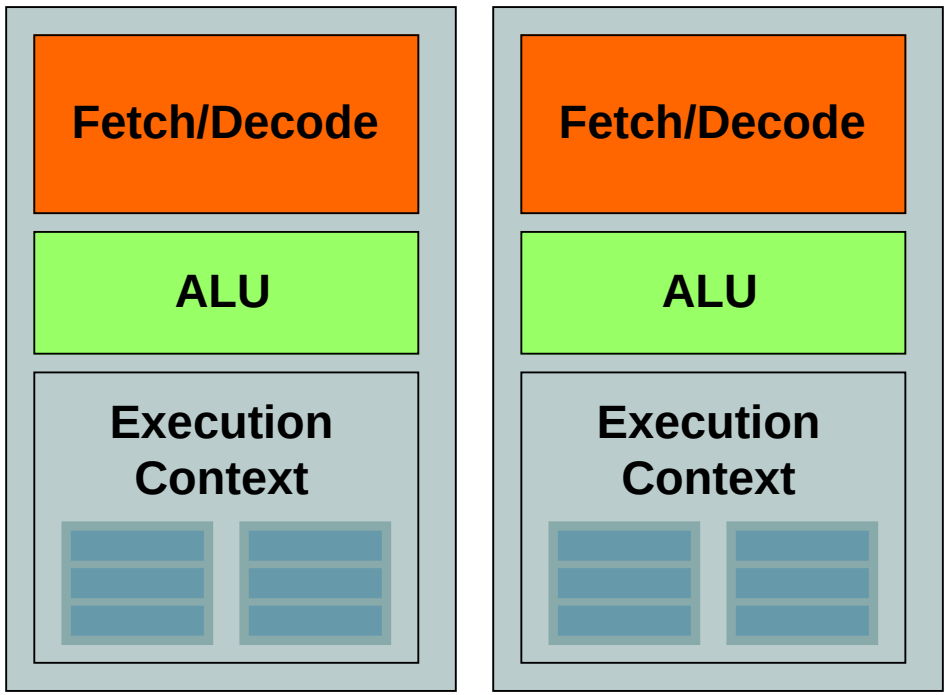
- Shader program on one execution unit




```
add r0.w, -r0.z, c1.w
mul r4.xy, r4, r0.w
cmp r0.w, r0.z, c1.z, c1.w
mov r4.zw, c1
cmp r1.w, -c0.x, r4.z, r4.w
mul r0.w, r0.w, r1.w
cmp r0.xy, -r0.w, r0, r4
mul r4.xy, r0, r0
mul r4.z, r0.y, r0.x
mad r1.xyz, r1, c2.w, c2
mov r0.z, c1.w
```

GPU Parallelism

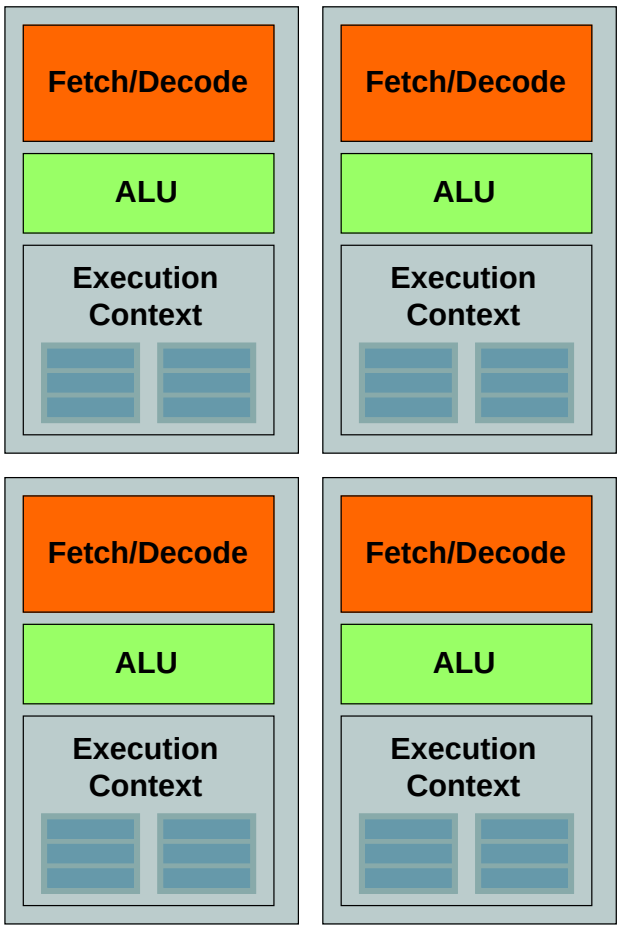
- Use same program for two units




```
add r0.w, -r0.z, c1.w
mul r4.xy, r4, r0.w
cmp r0.w, r0.z, c1.z, c1.w
mov r4.zw, c1
cmp r1.w, -c0.x, r4.z, r4.w
mul r0.w, r0.w, r1.w
cmp r0.xy, -r0.w, r0, r4
mul r4.xy, r0, r0
mul r4.z, r0.y, r0.x
mad r1.xyz, r1, c2.w, c2
mov r0.z, c1.w
```

GPU Parallelism

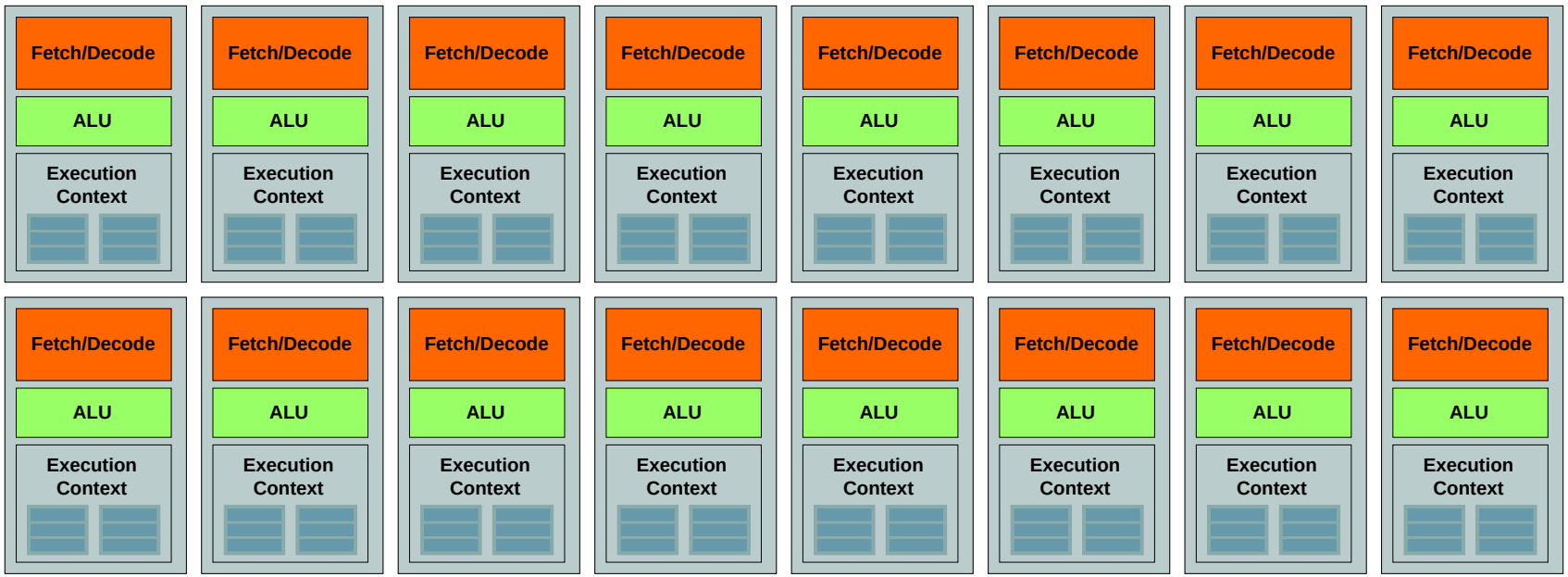
- Use same program for more units




```
add r0.w, -r0.z, c1.w
mul r4.xy, r4, r0.w
cmp r0.w, r0.z, c1.z, c1.w
mov r4.zw, c1
cmp r1.w, -c0.x, r4.z, r4.w
mul r0.w, r0.w, r1.w
cmp r0.xy, -r0.w, r0, r4
mul r4.xy, r0, r0
mul r4.z, r0.y, r0.x
mad r1.xyz, r1, c2.w, c2
mov r0.z, c1.w
```

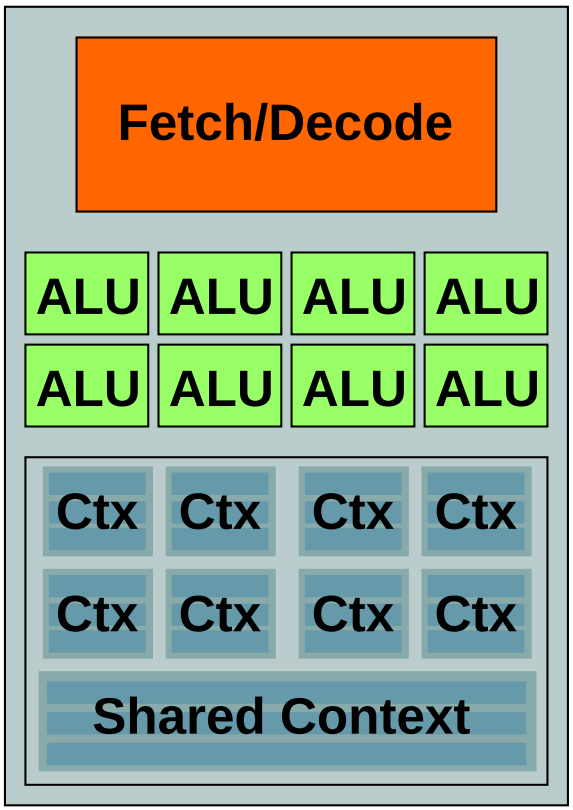
GPU Parallelism

- Use same program for even more units
- Moores Law: GPU parallelism is doubling every year



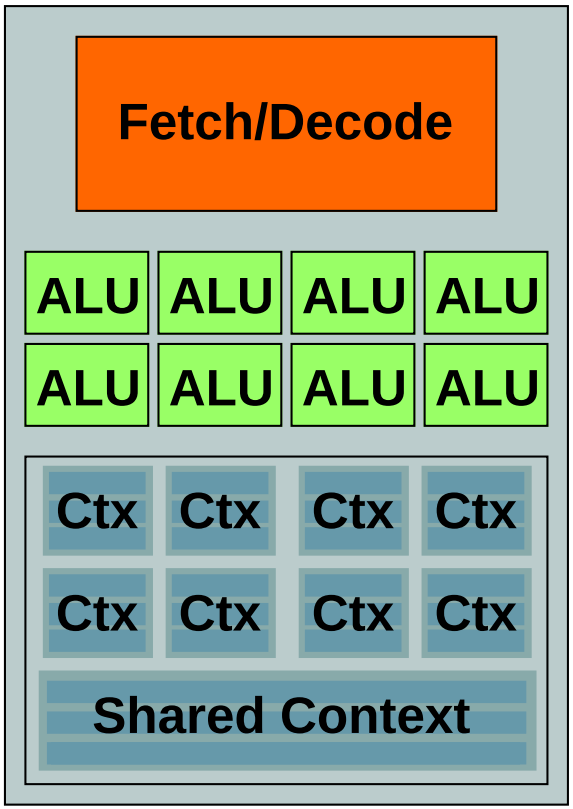
GPU Hardware - SIMD

- Use same instruction unit for more than one stream



GPU Hardware - SIMD

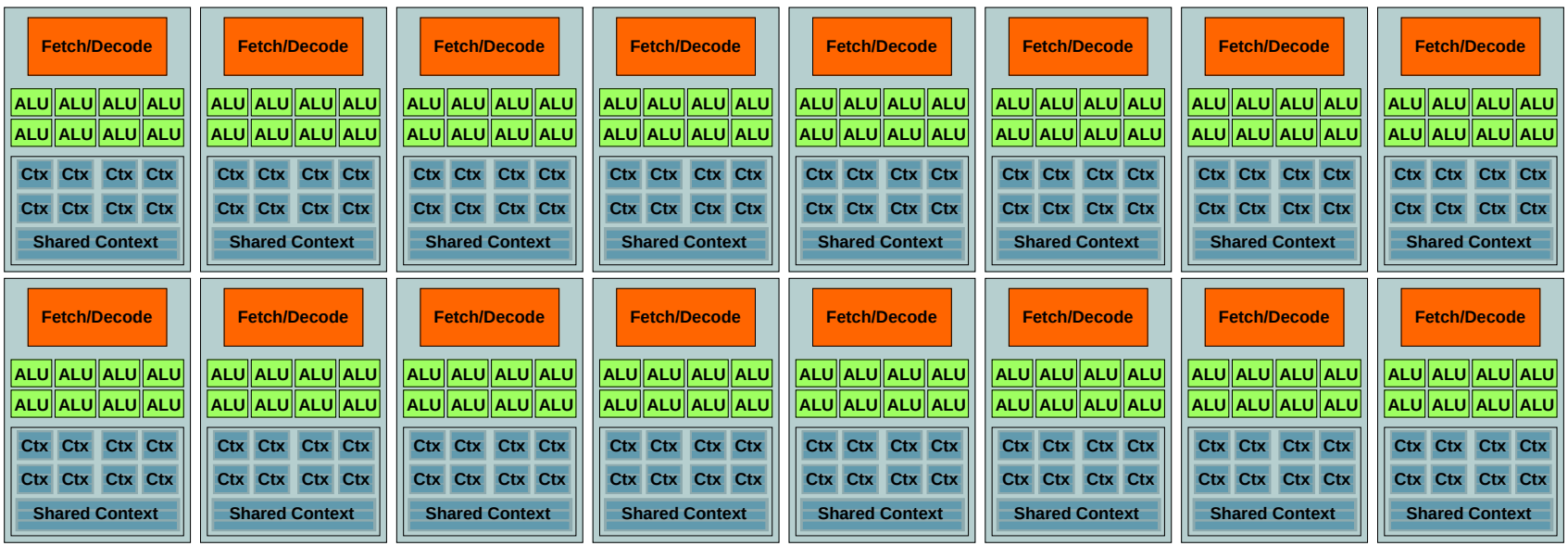
- Use same instruction unit for more than one stream




```
vec_add r0.w, -r0.z, c1.w
vec_mul r4.xy, r4, r0.w
vec_cmp r0.w, r0.z, c1.z, c1.w
vec_mov r4.zw, c1
vec_cmp r1.w, -c0.x, r4.z, r4.w
vec_mul r0.w, r0.w, r1.w
vec_cmp r0.xy, -r0.w, r0, r4
vec_mul r4.xy, r0, r0
vec_mul r4.z, r0.y, r0.x
vec_mad r1.xyz, r1, c2.w, c2
vec_mov r0.z, c1.w
```

GPU Hardware - SIMD

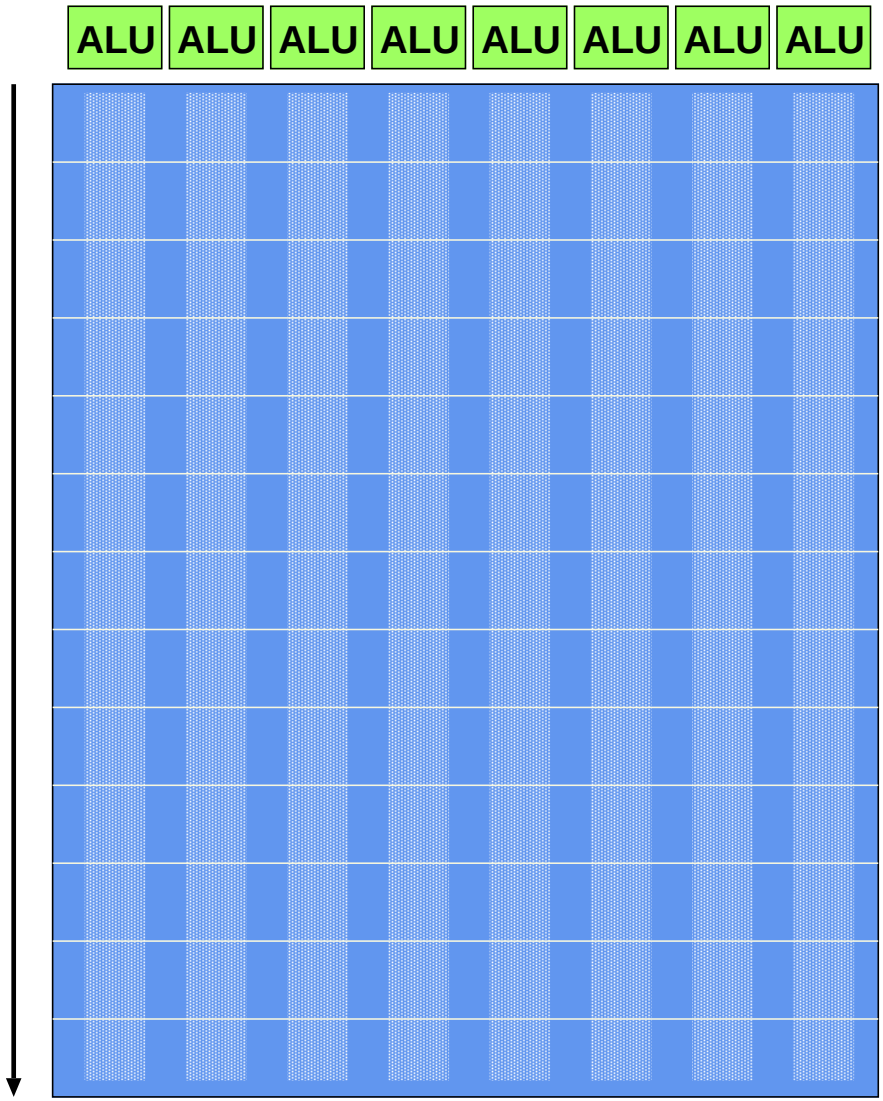
- Use same instruction unit for more than one stream



SIMD Processing

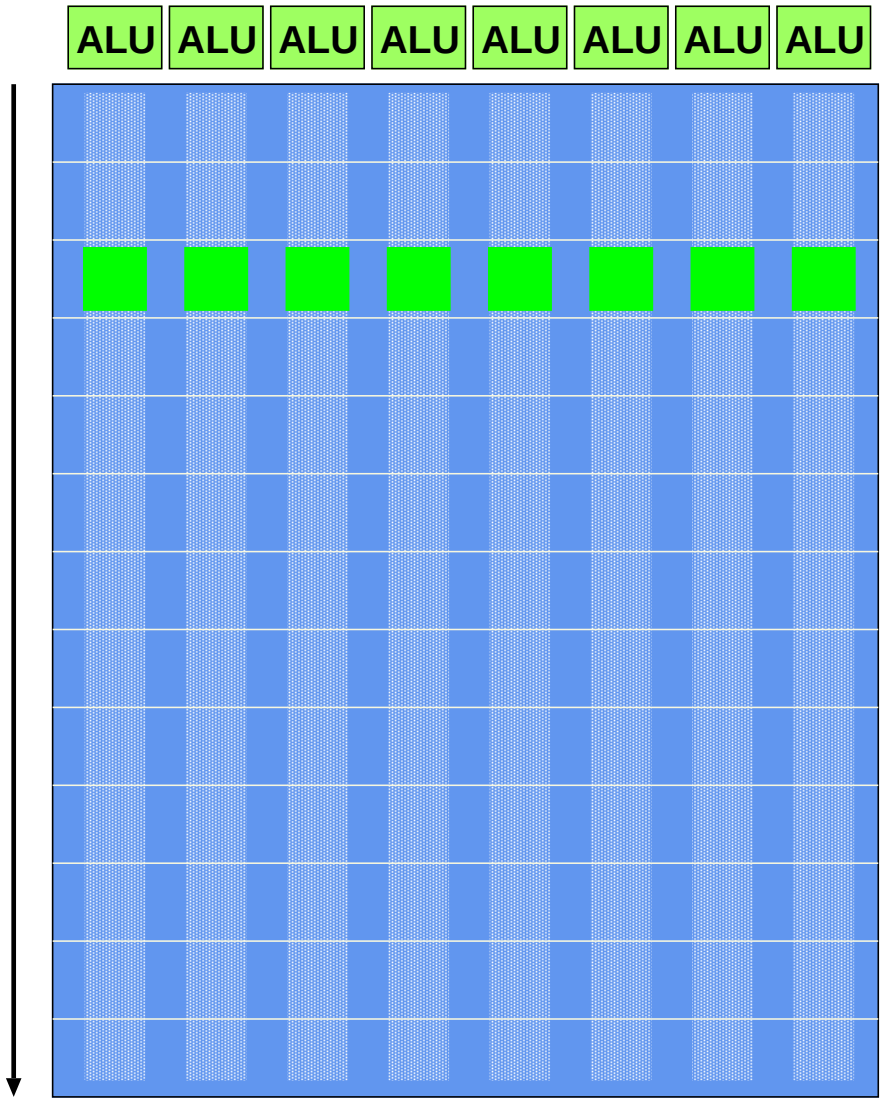
- Explicit vector instructions
  - Intel/AMD/PPC SSE
  - Cell B.E.
  - Intel Larrabee
- Scalar instructions, implicit vectorization
  - NVIDIA GPUs
  - AMD GPUs
  - SPMD alike concept: Single Instruction, Multiple Thread (SIMT)

SIMT



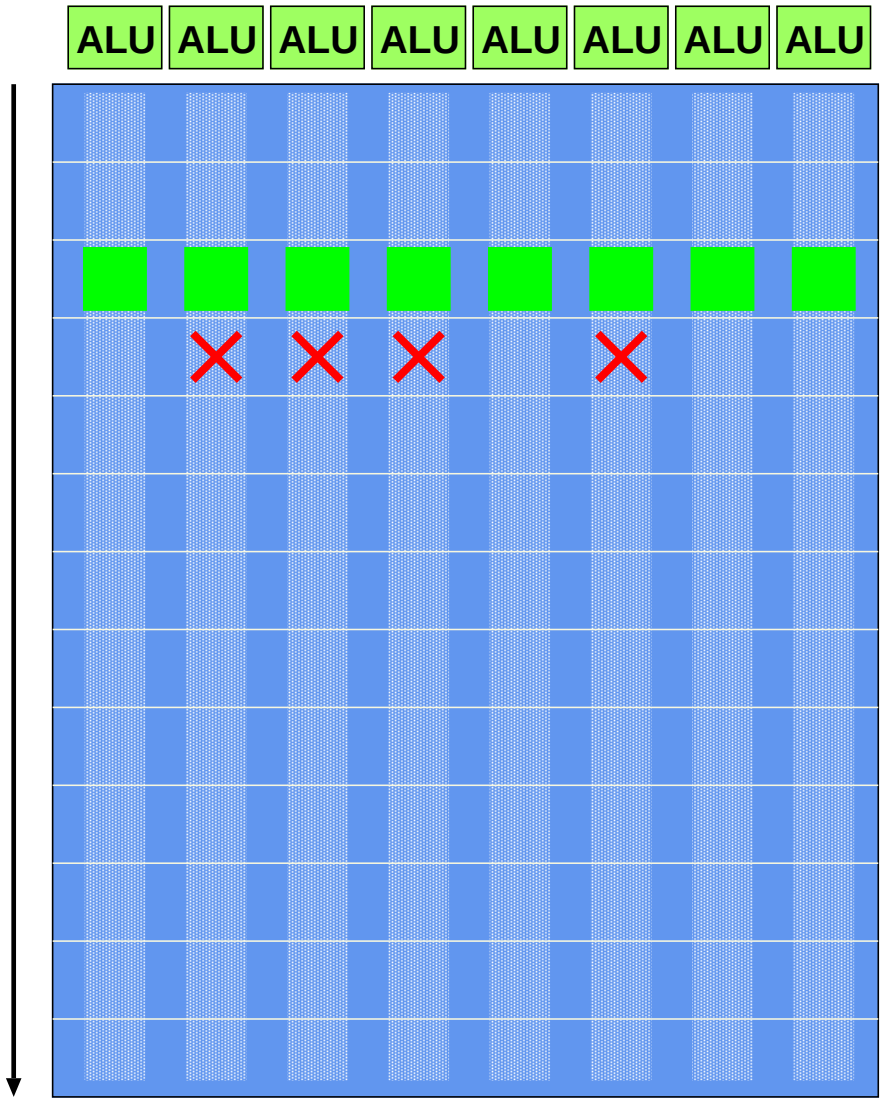

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



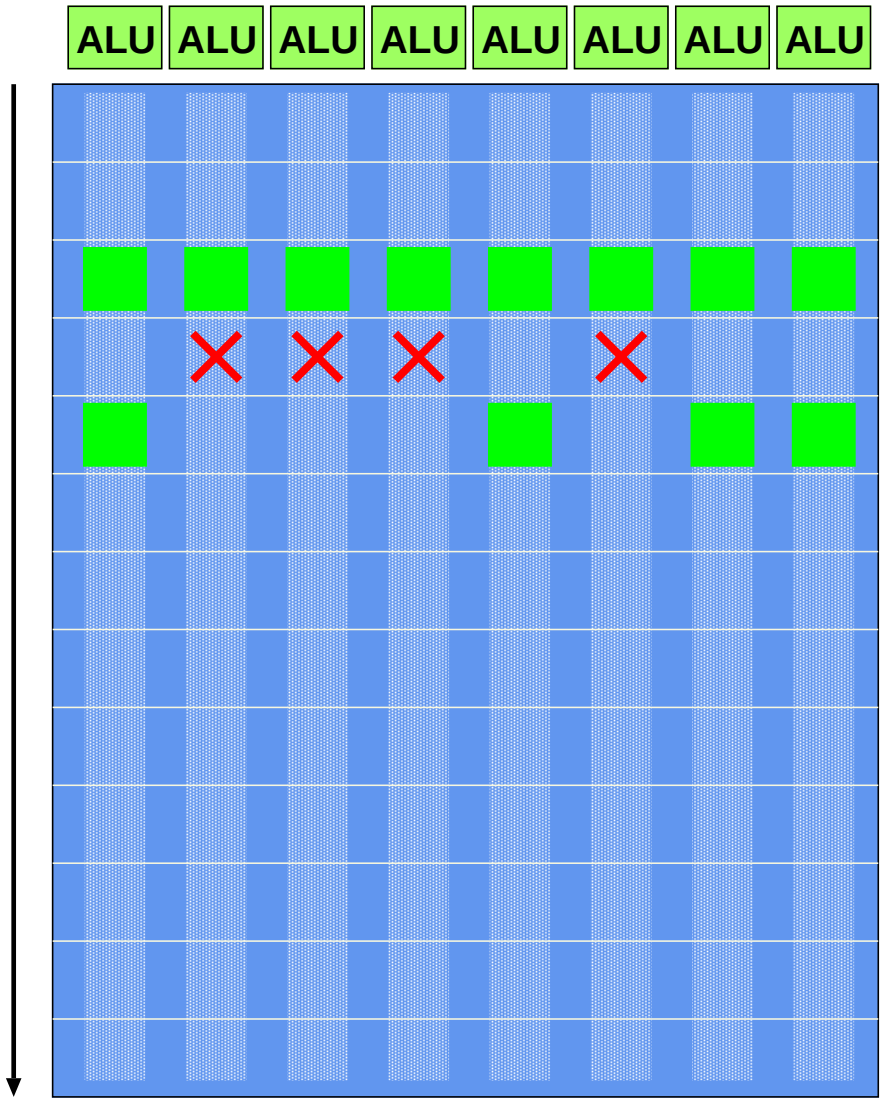

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



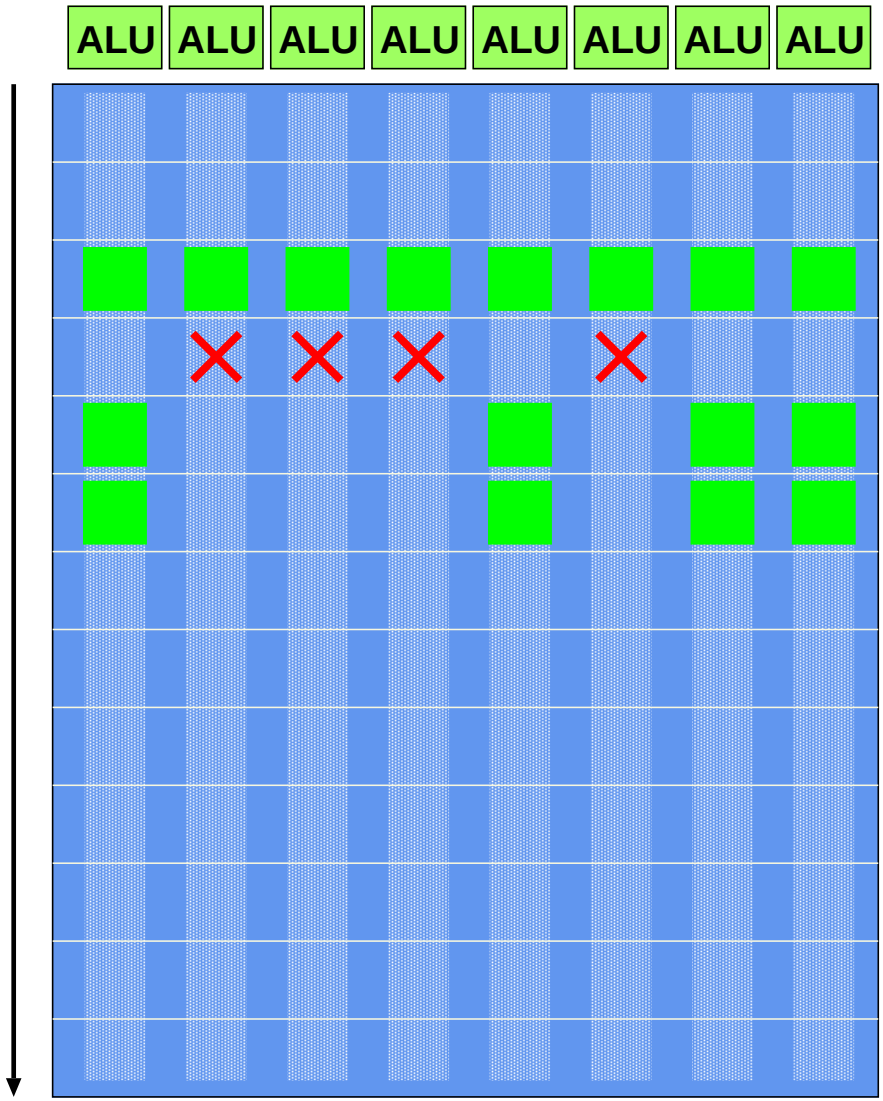

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



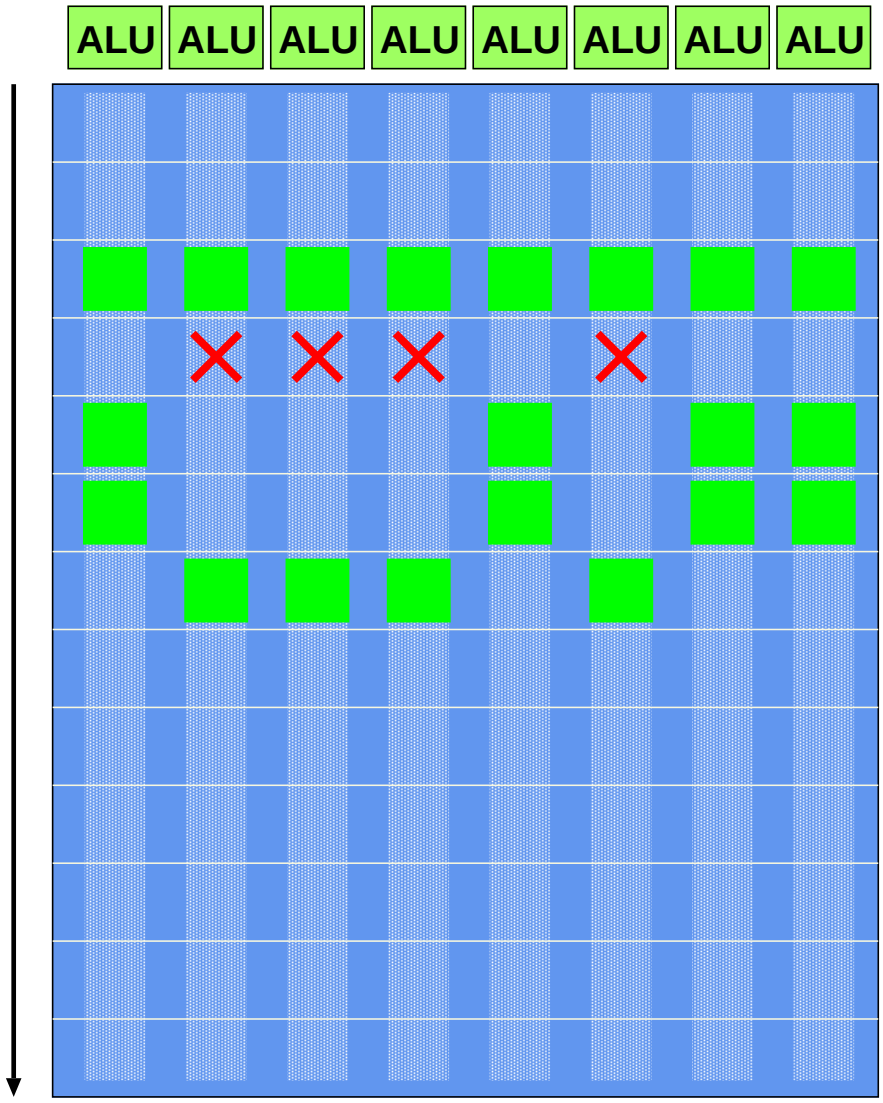

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



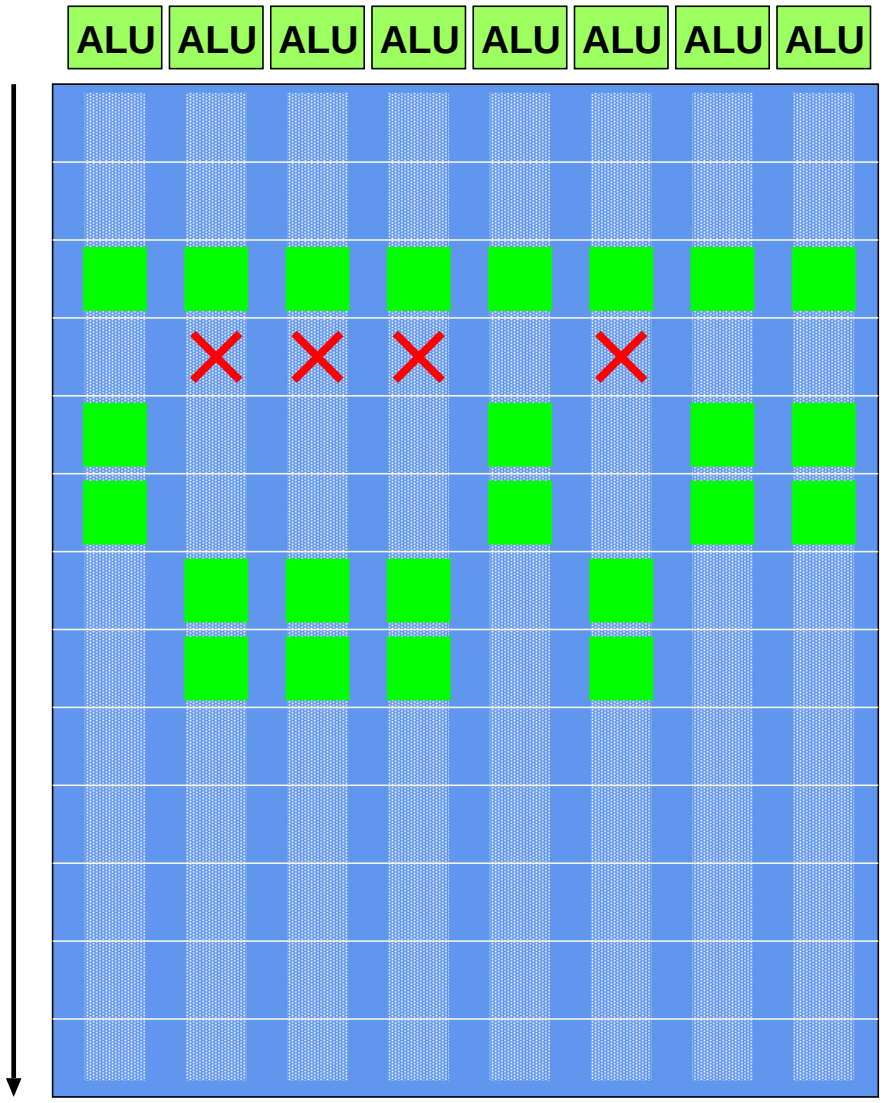

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



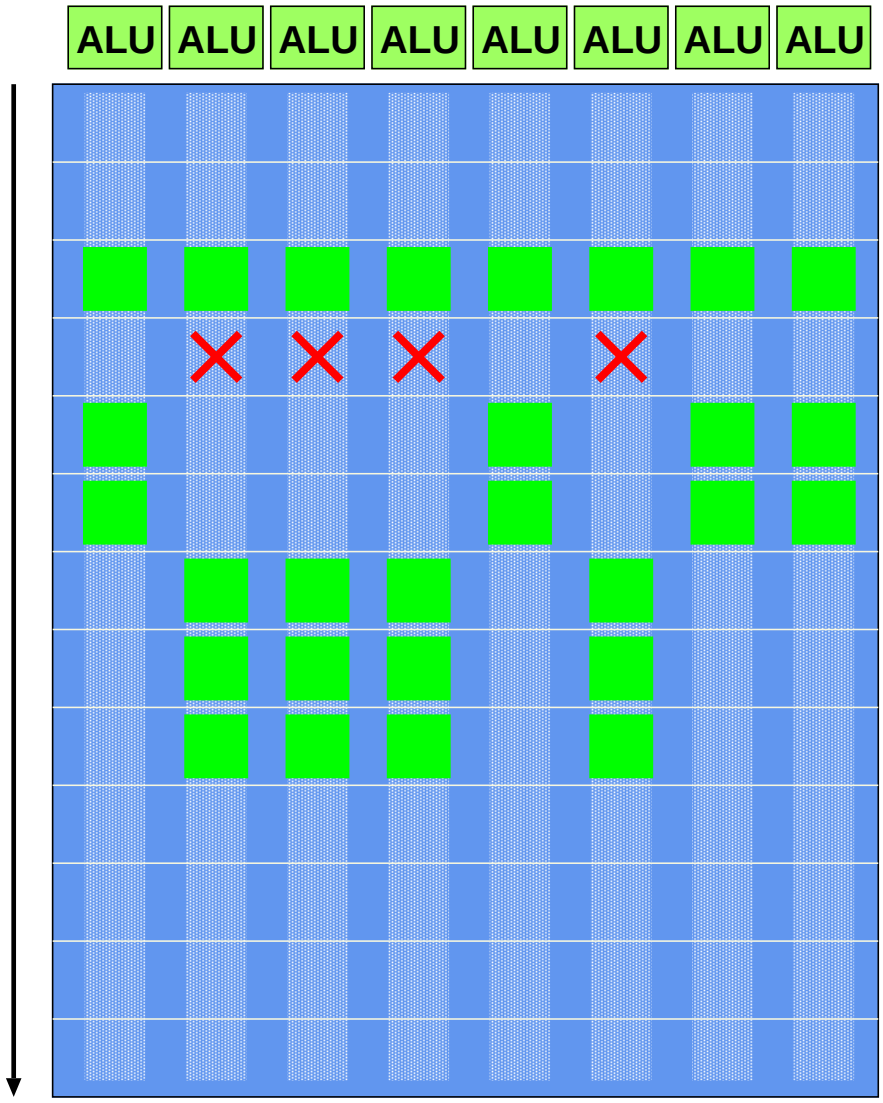

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT




```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

SIMT



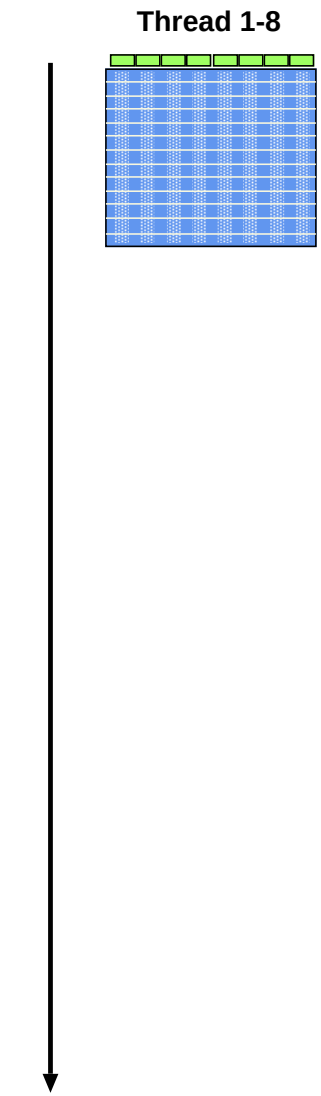

```
<unconditional code>

diff = a[x] - b[x];
if (diff > 0) {
    s = exp(-c * diff);
    d+=s;
} else {
    r = b[x];
    s = exp(-d *a[x]);
    d-=s;
}
```

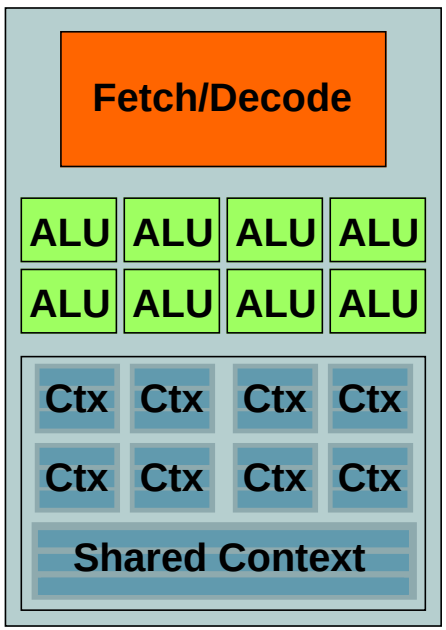
Memory Latency

- No caches on chip
- Access to memory takes a few hundred clock cycles
- Threads perform often only few instructions
- No other instruction can be scheduled due to data dependency

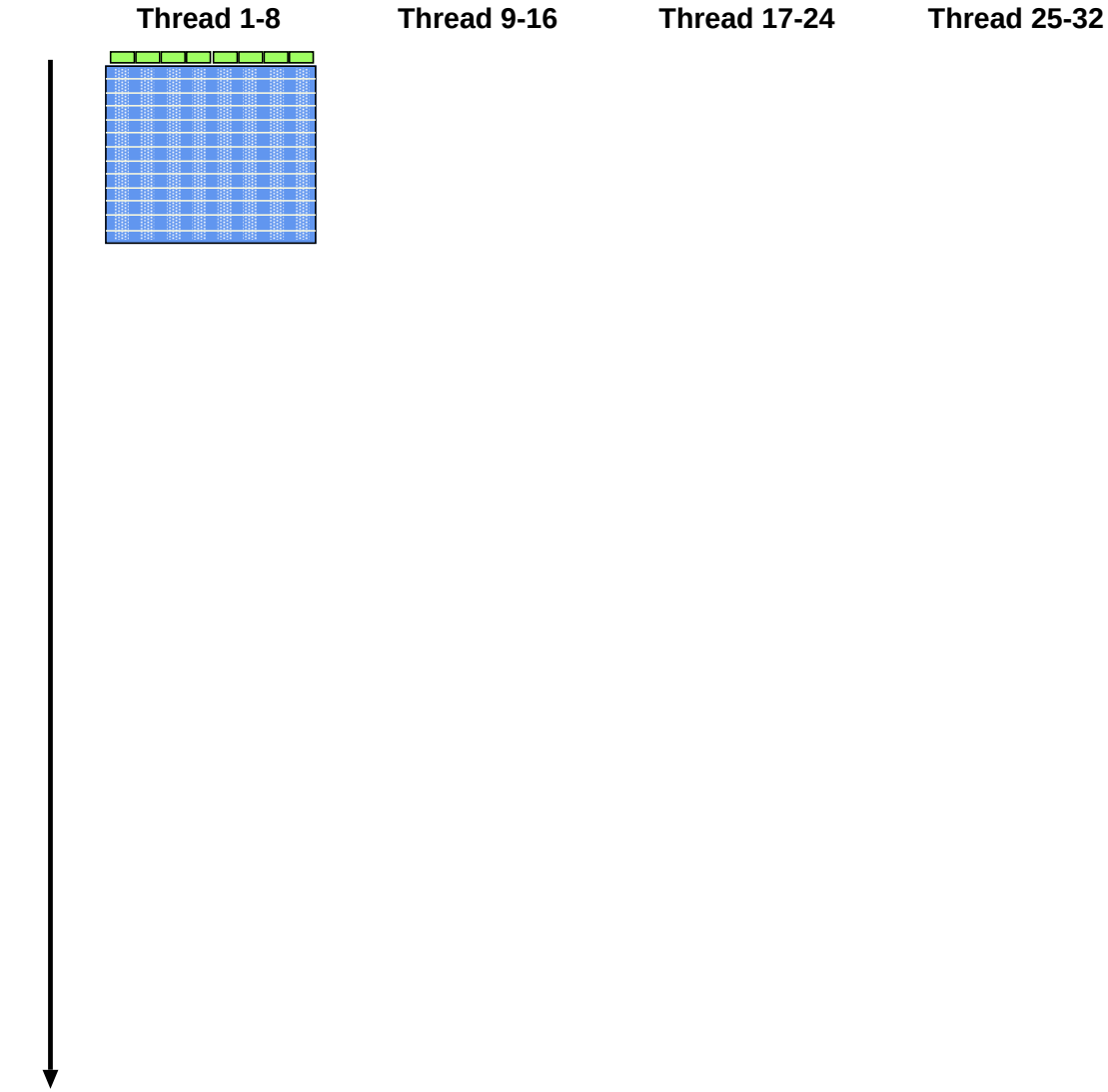
$\Rightarrow$ Stalls!!!

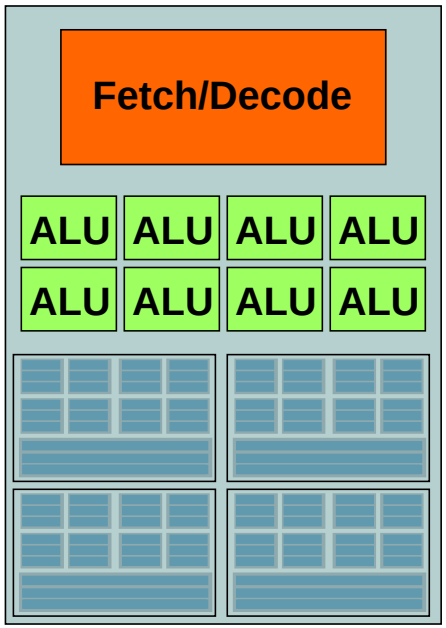
Memory Stalls



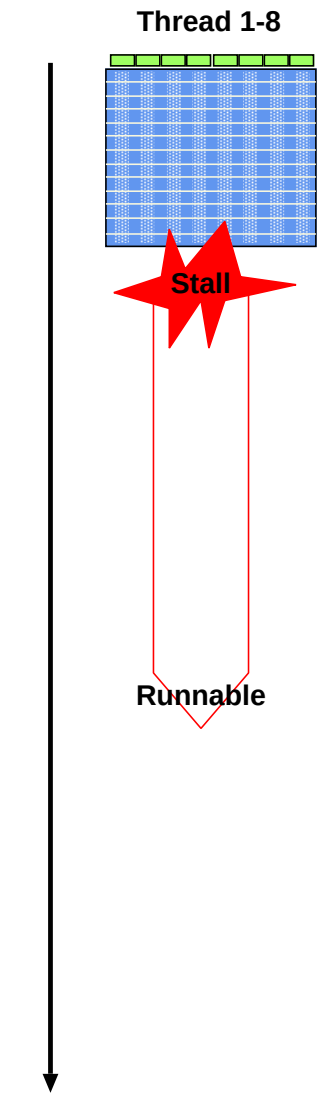


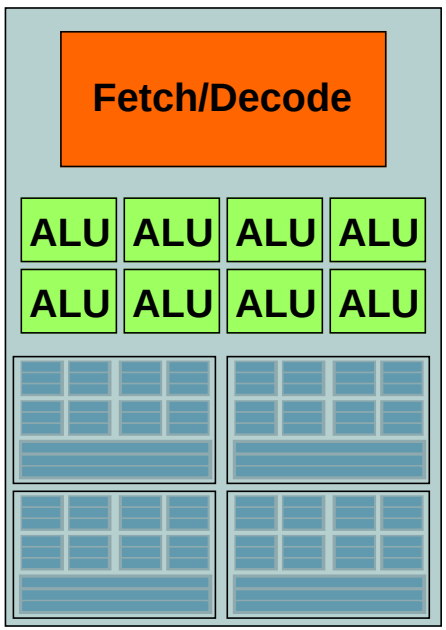
Memory Stalls



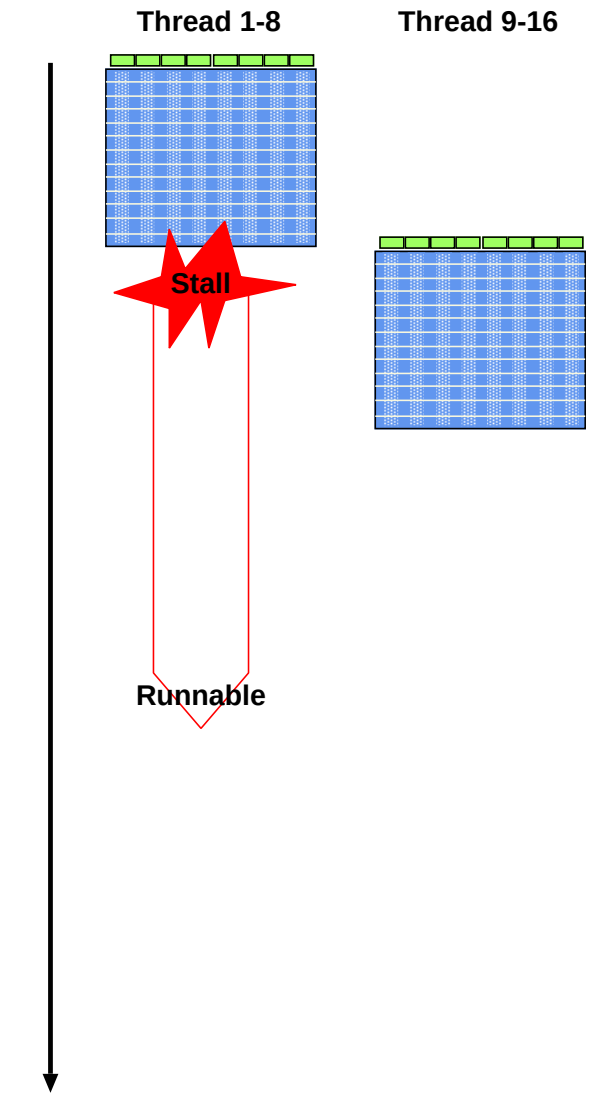


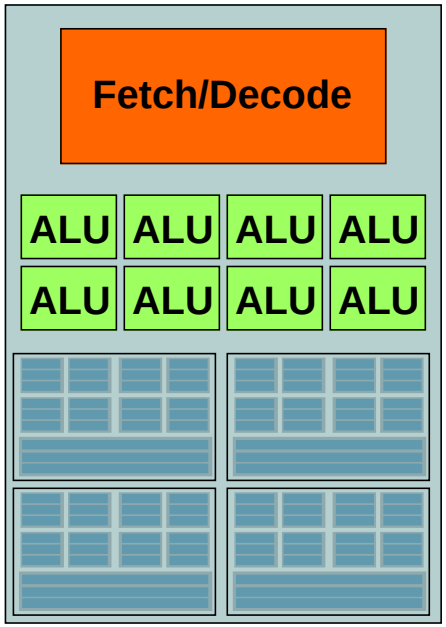
Memory Stalls



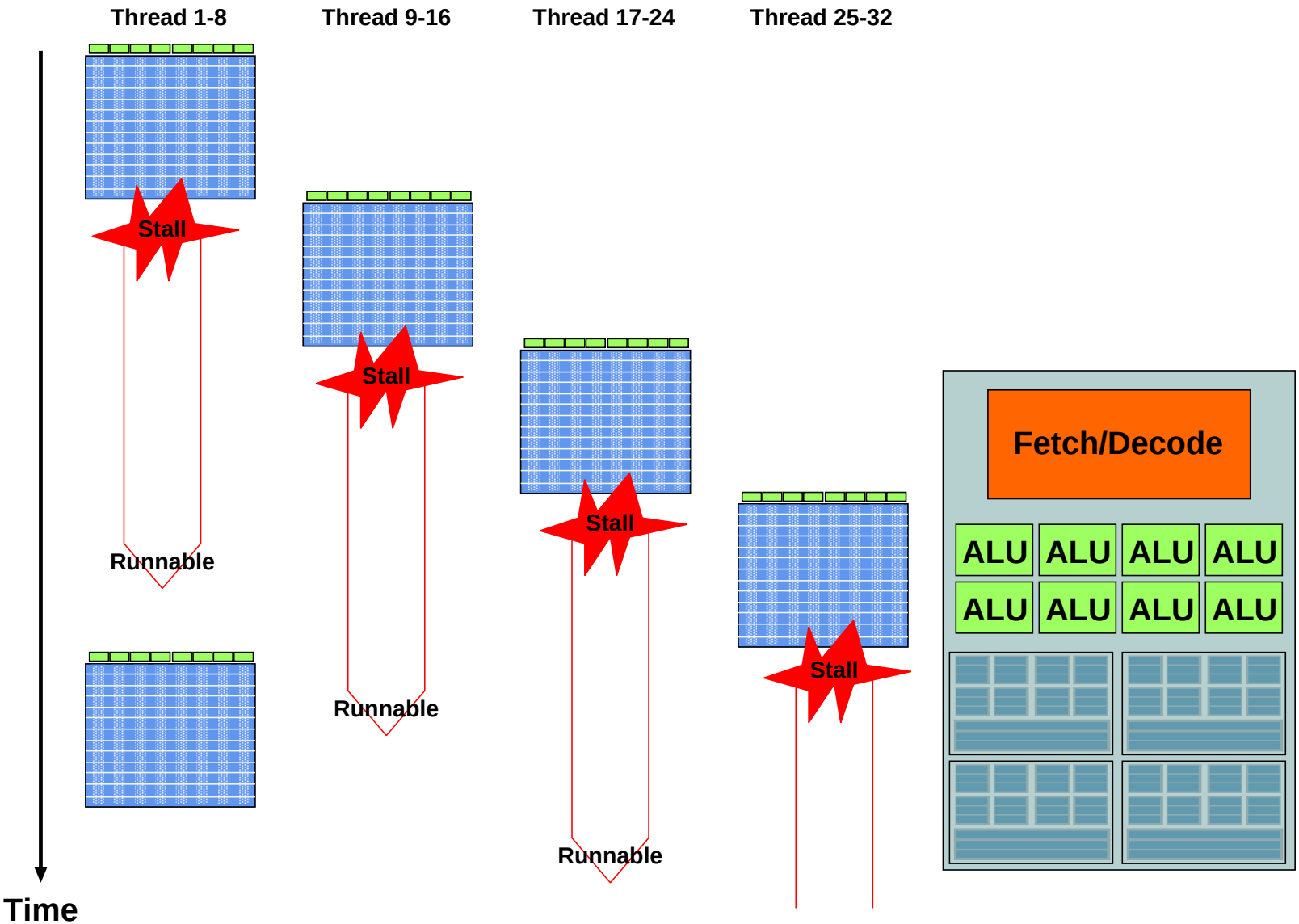


Memory Stalls

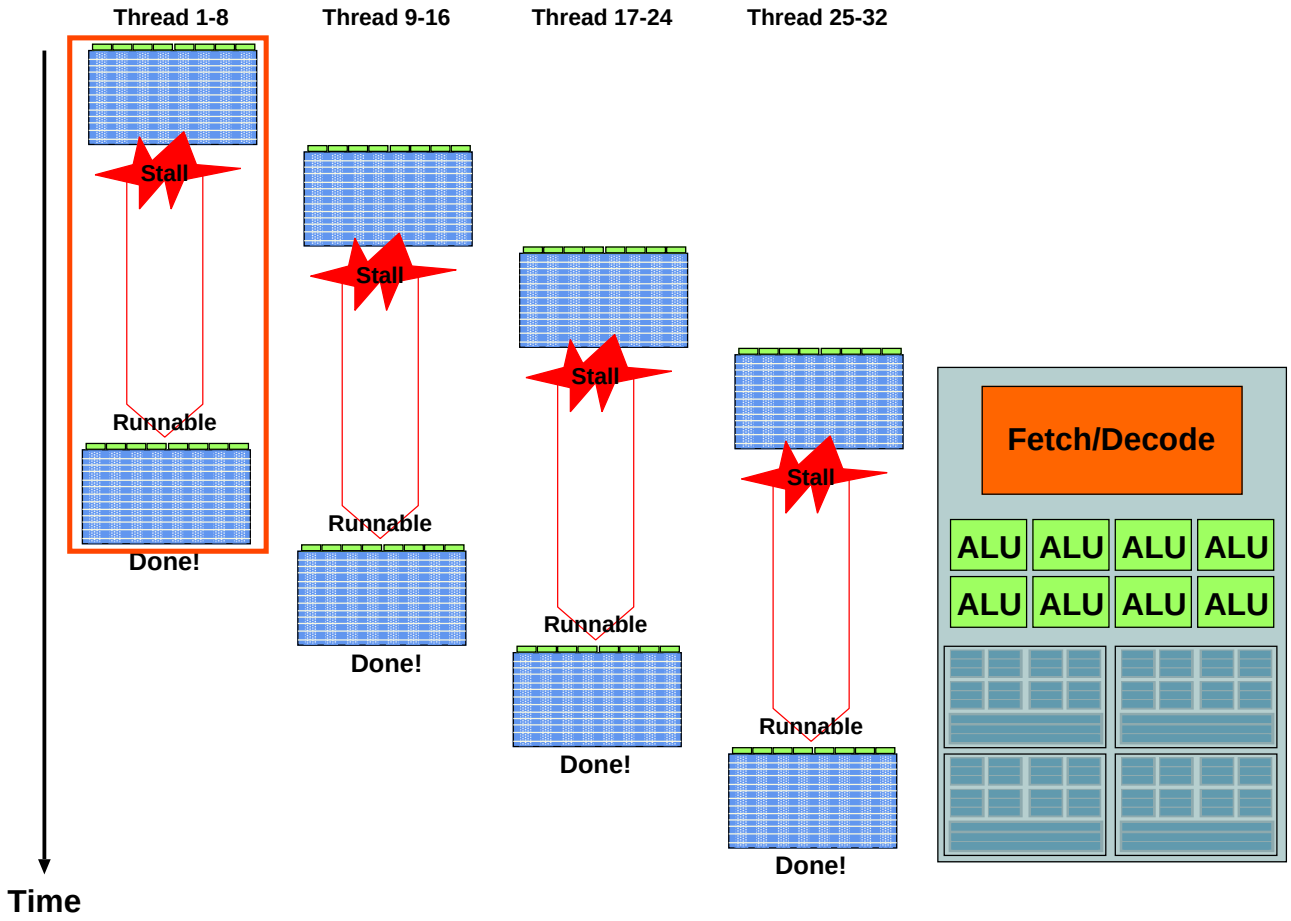




Memory Stalls



GPU Throughput

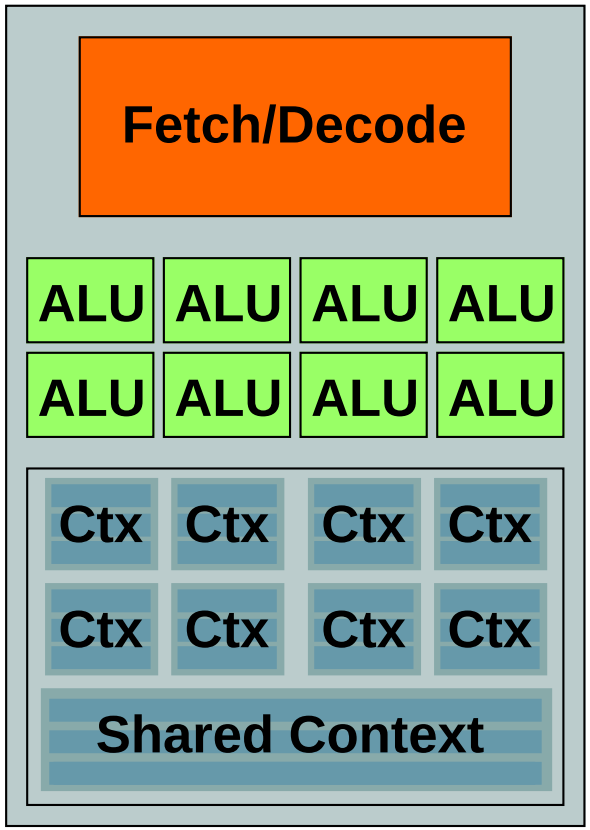


- Increase run-time of one group, maximize throughput of many groups
- GPU: high latency, high throughput

Context storage

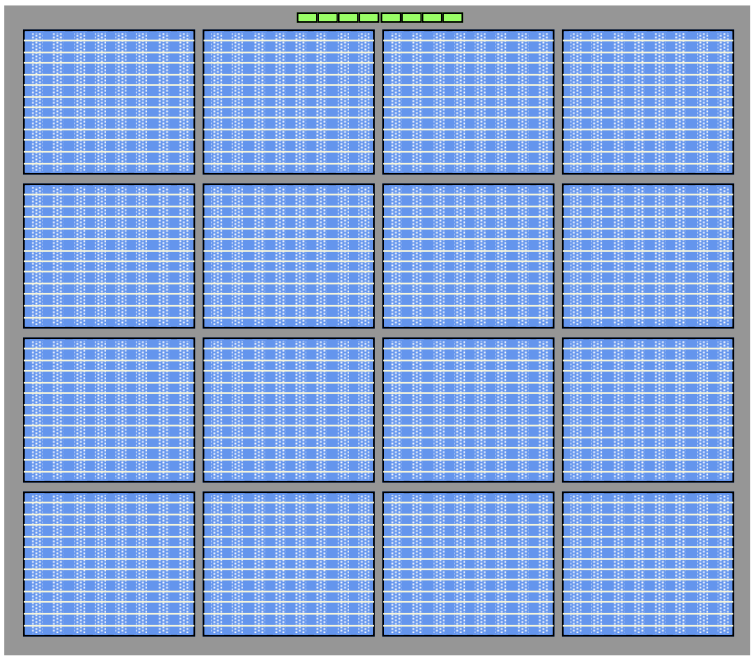
- Large context storage
- Maximal 24 contexts

$\Rightarrow$ Scalability



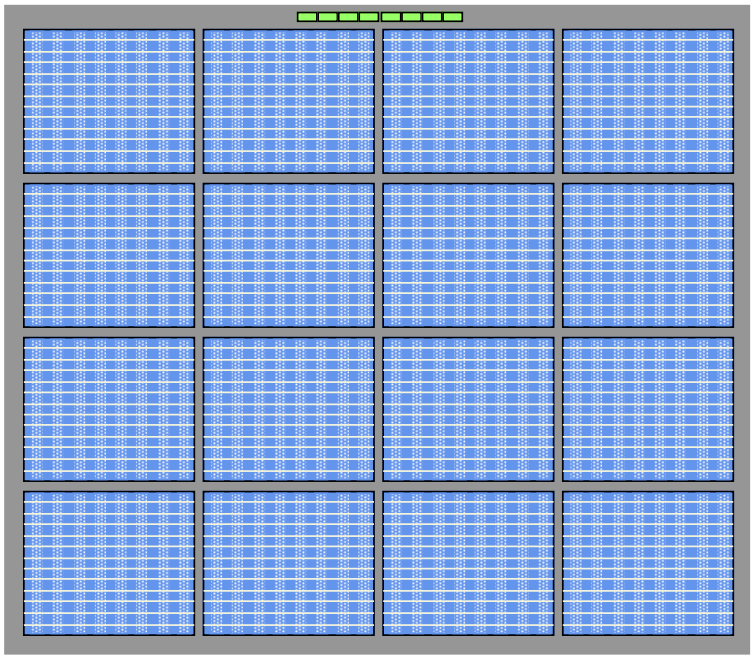
Blocking

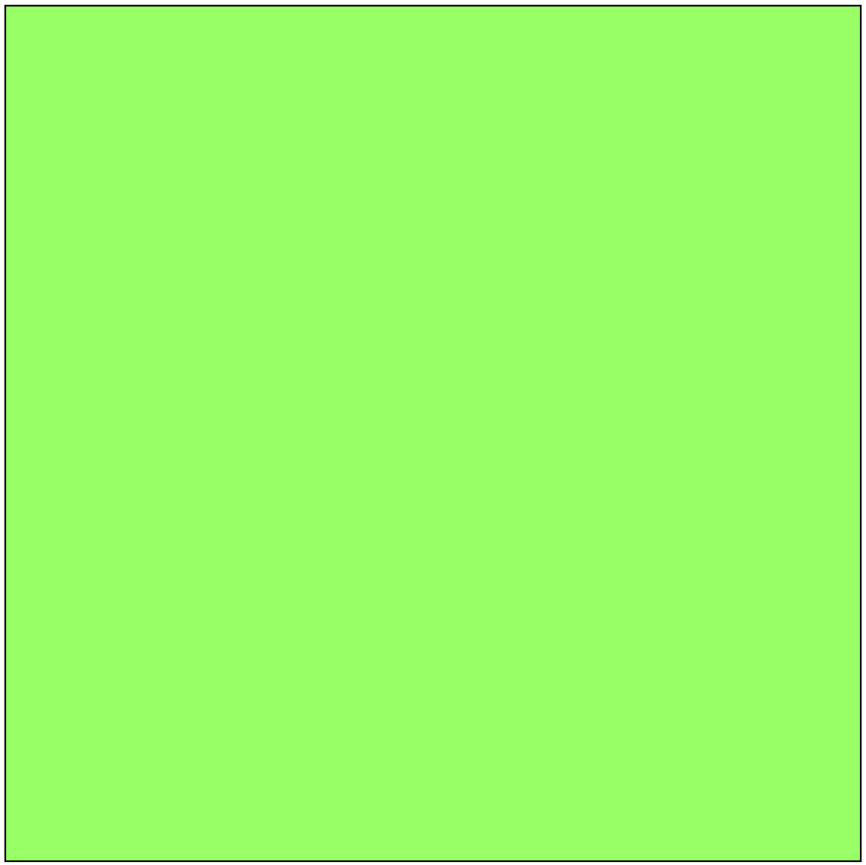
- One core can executes many contexts in parallel



Blocking

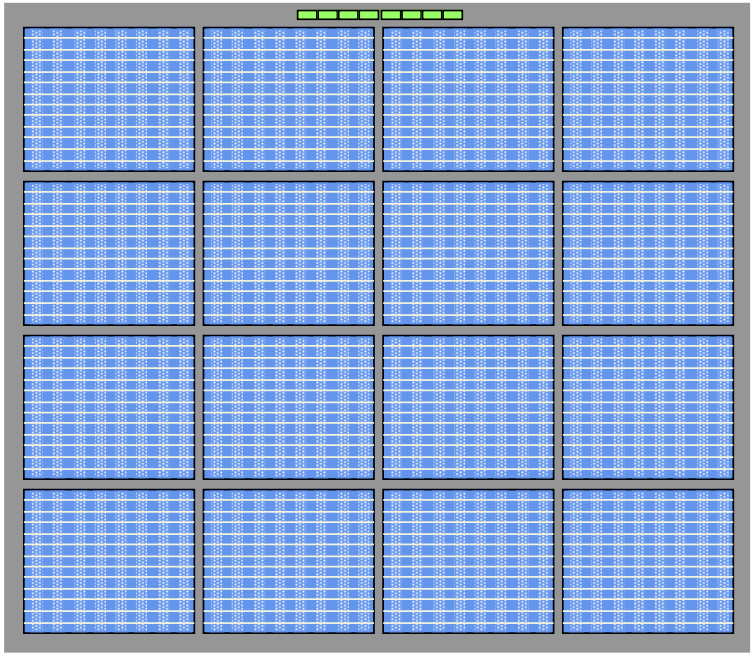
- Data has to be partitioned into independent parts

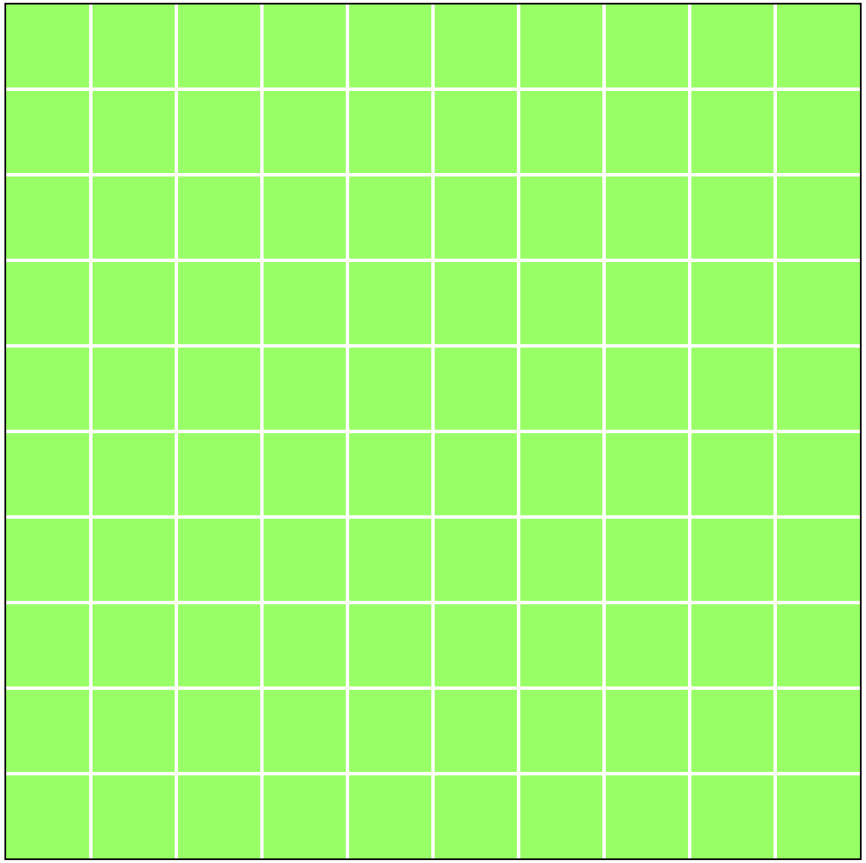




Blocking

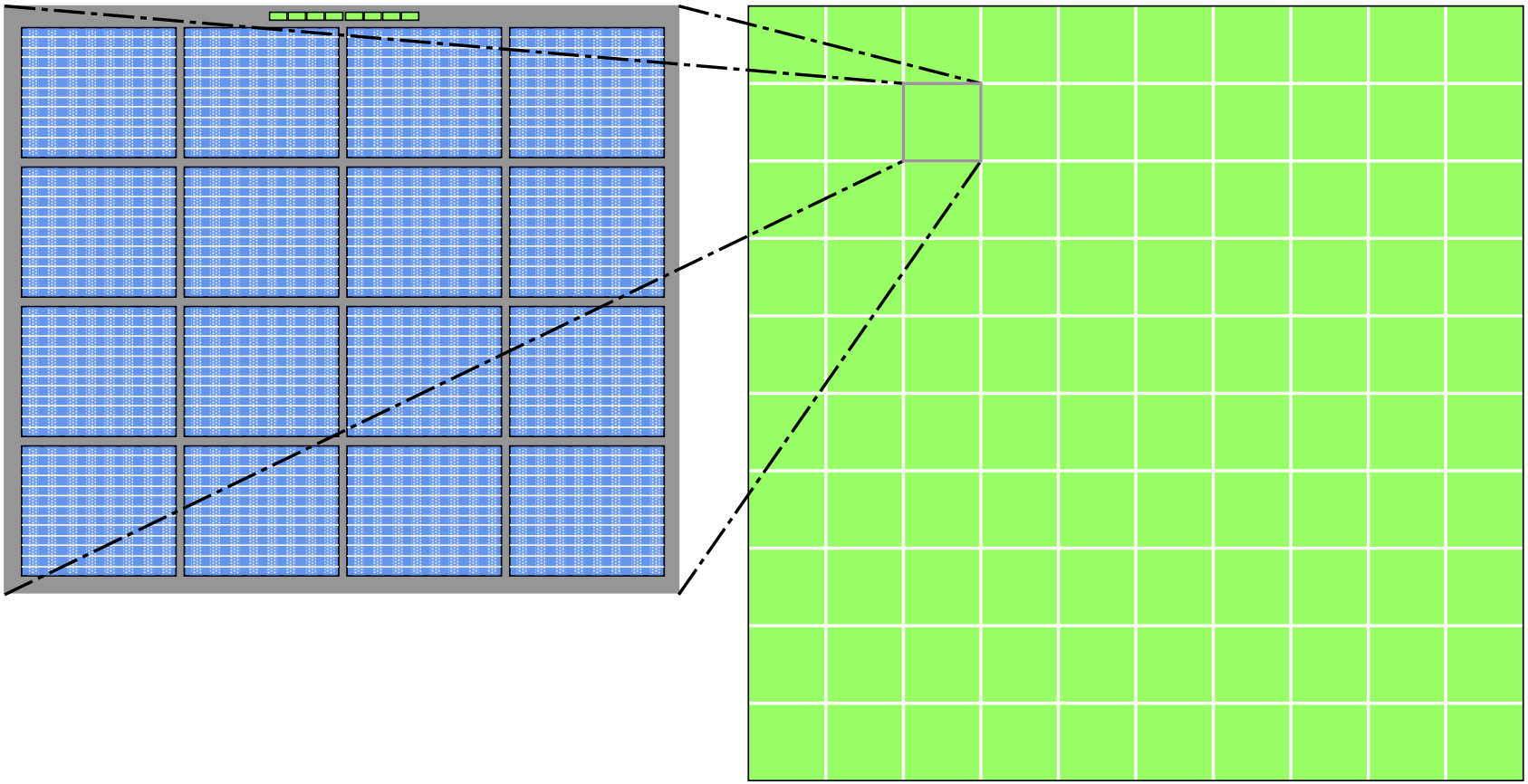
- Data has to be partitioned into independent parts





Blocking

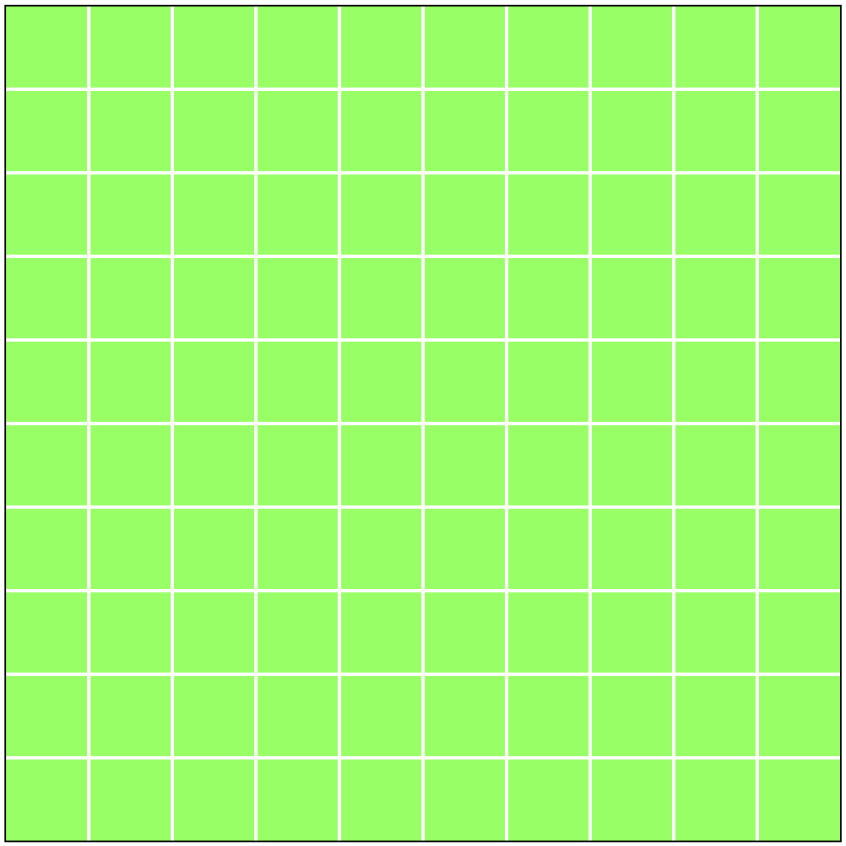
- One core works on such independent blocks



Blocking

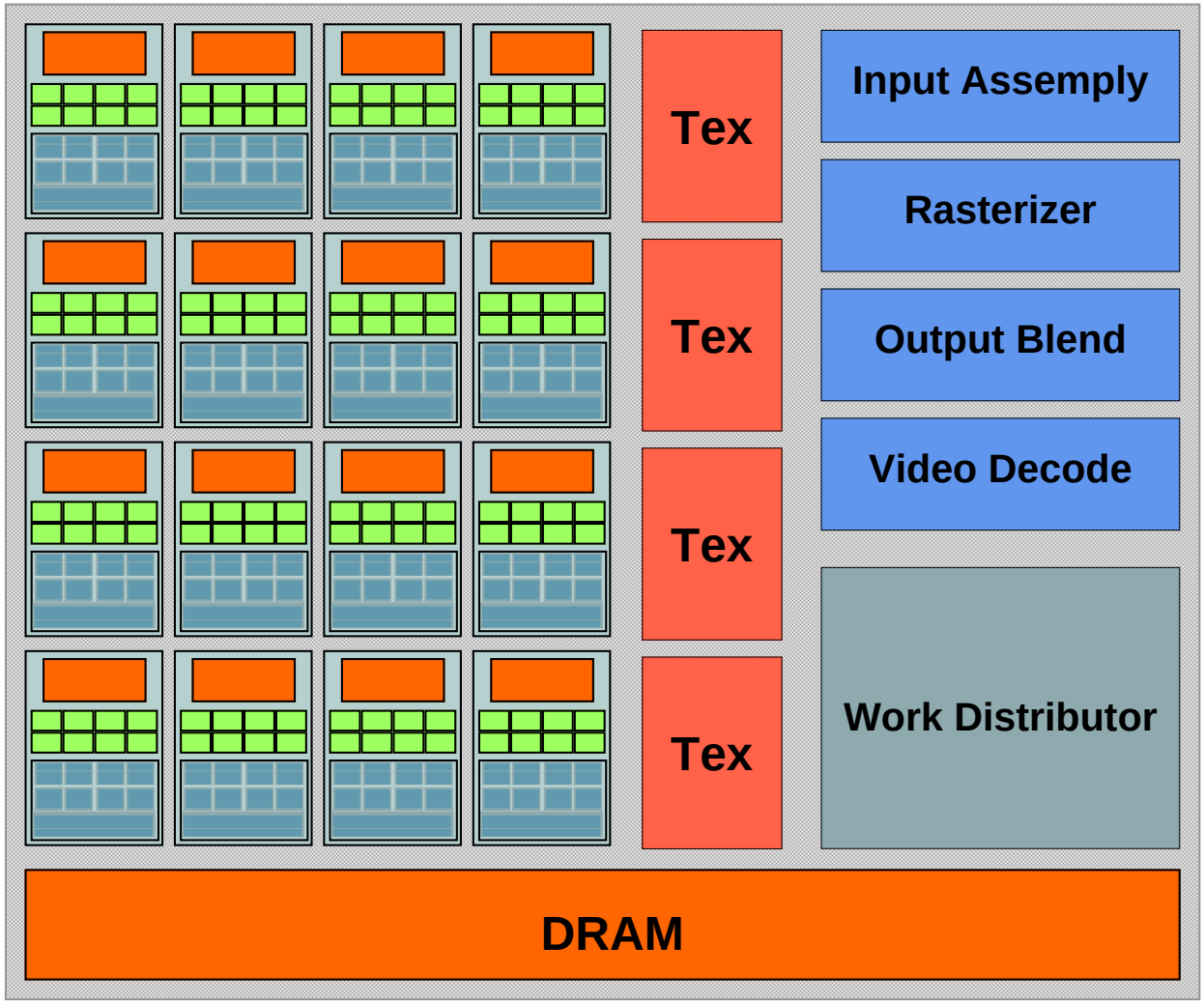
- User defines mapping of blocks to data
- Same kernel is executed on all blocks
- Blocks must be processed independently
- Scheduling of blocks is done by hardware

$\Rightarrow$ Thousands of threads in parallel



GPU Performance

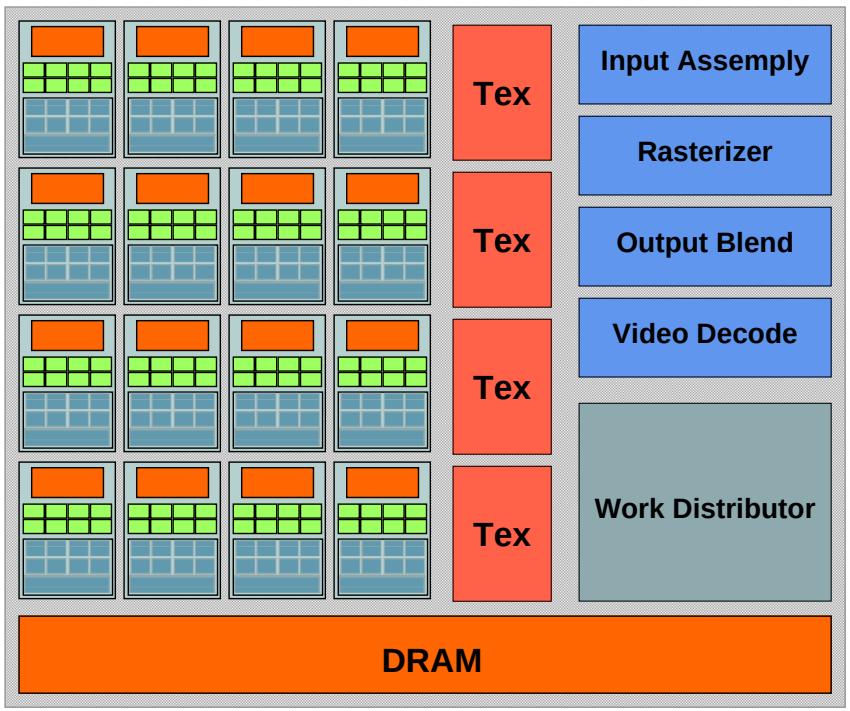
- 16 cores
- 8 ALUs per core
- 128 ALUs
- 1 Mul-Add per cycle
  $\Rightarrow$ 256 GFLOPs (@1GHz)

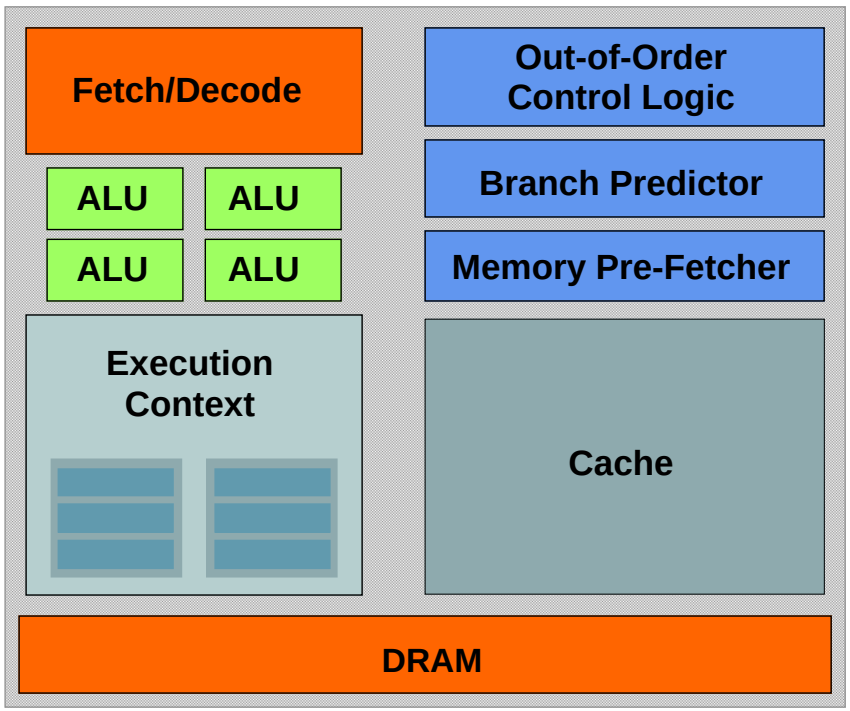


(Tesla C1060: 30 cores, 1.3GHz $\Rightarrow$ 624GFLOPS)

Summary

- Remove components that help a single instruction stream
- Use multiple units in parallel
- Use SIMT
- Schedule multiple blocks (hide latency)





Questions?

Krakow, Pontifical Residency
Courtesy of Robert Grimm