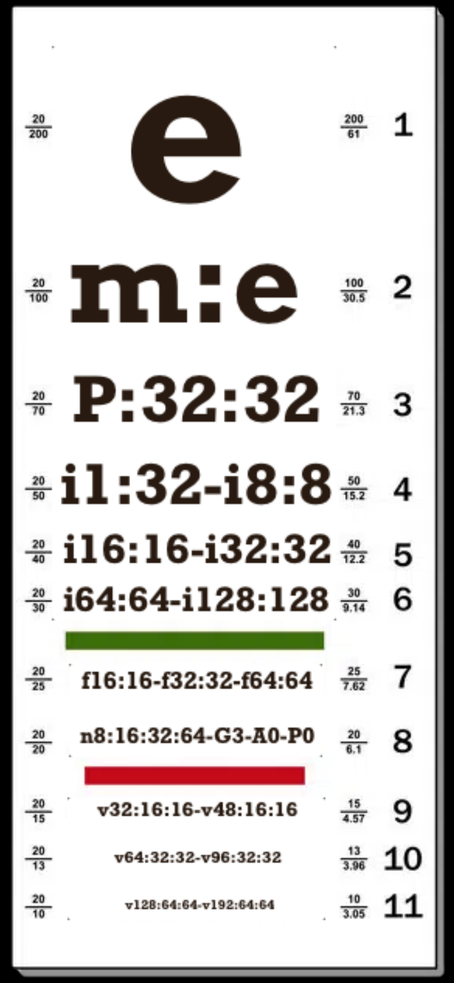
Impossible Perspectives in Data Layout

Defining Data Layouts

- LLVM has per-target Data Layouts
- Alignment constraints and object size rules
- Explicit padding for more elaborate rules



Example: Explicit Padding

```
struct S {
  long long x;
  alignas(32) int y;
};
```

```
%struct.S = type { i64, [24 x i8], i32, [28 x i8] }
```

Example: Explicit Padding

```
struct S {
  long long x;
  alignas(32) int y;
};
```

```
%struct.S = type { i64, [24 x i8], i32, [28 x i8] }
```

Note the padding after the element

HLSL's CBuffer

- HLSL's CBuffers have unusual layout rules
- These were designed around early GPUs
- 16-byte registers with packing rules

That's Right! It goes in a 16-byte register

Example: Scalars and Vectors

- Packed into 16-byte registers
- 1-4 element vectors are element aligned
- Will not cross a register boundary

```
cbuffer cb {
    float a;
    float3 b;
    float3 c;
    float2 d;
    float e;
    float f;
}
```

```
                    offset  size  +pad
cbuffer cb {
    float a;           0      4
    float3 b;          4     12
    float3 c;         16     12  + 4
    float2 d;         32      8
    float e;          40      4
    float f;          44      4
};                            48
```

https://maraneshi.github.io/HLSL-ConstantBufferLayoutVisualizer/

Example: Structs

- Always start on a register boundary
- No implied padding

```
struct S {
  int i;
};
cbuffer cb {
    S a;
    S b;
    int j;
}
```

```
                    offset size +pad
cbuffer cb {
    struct S {
        int i;        0     4
    } a;              0     4 +12
    struct S {
        int i;       16     4
    } b;             16     4
    int j;           20     4
};                         24
```

https://maraneshi.github.io/HLSL-ConstantBufferLayoutVisualizer/

Example: Arrays

- Each element starts on a register boundary
- No padding after last element

```
cbuffer cb {
    float a[3];
    float2 b;
}
```

```
                    offset size +pad
cbuffer cb {
    float a[0];          0    4 +12
    float a[1];         16    4 +12
    float a[2];         32    4
    float2 b;           36    8
};                        44
```

https://maraneshi.github.io/HLSL-ConstantBufferLayoutVisualizer/

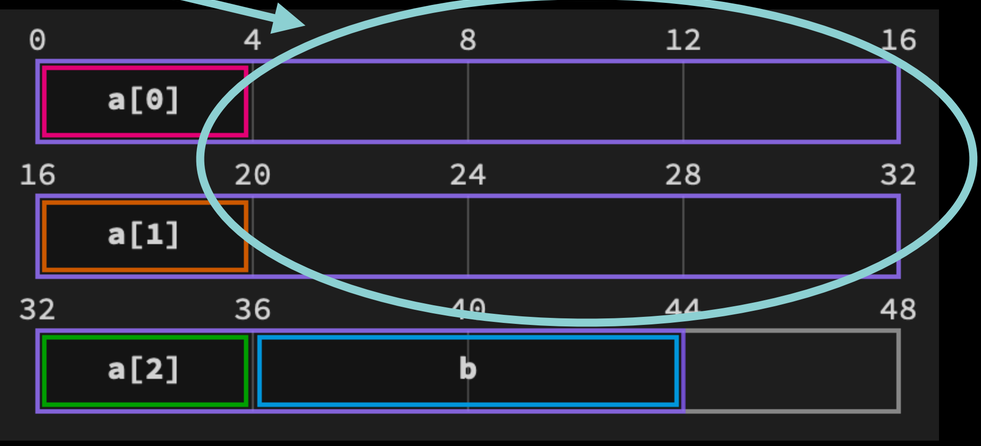
Example: Arrays

- Each element starts on a register boundary
- No padding after last element

```
cbuffer cb {
    float a[3];
    float2 b;
}
```

Padding between elements but not after

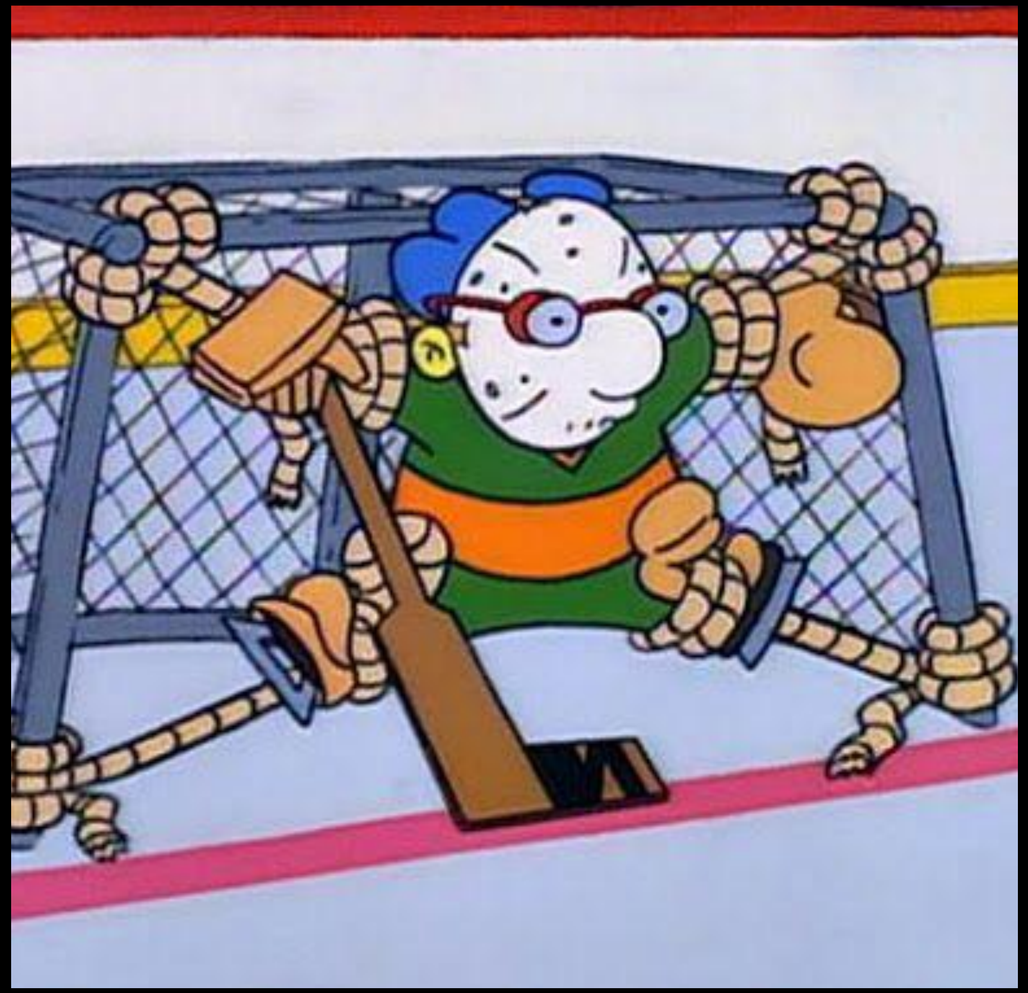
```
                    offset size +pad
cbuffer cb {
    float a[0];          0    4 +12
    float a[1];         16    4 +12
    float a[2];         32    4
    float2 b;           36    8
};                        44
```



https://maraneshi.github.io/HLSL-ConstantBufferLayoutVisualizer/

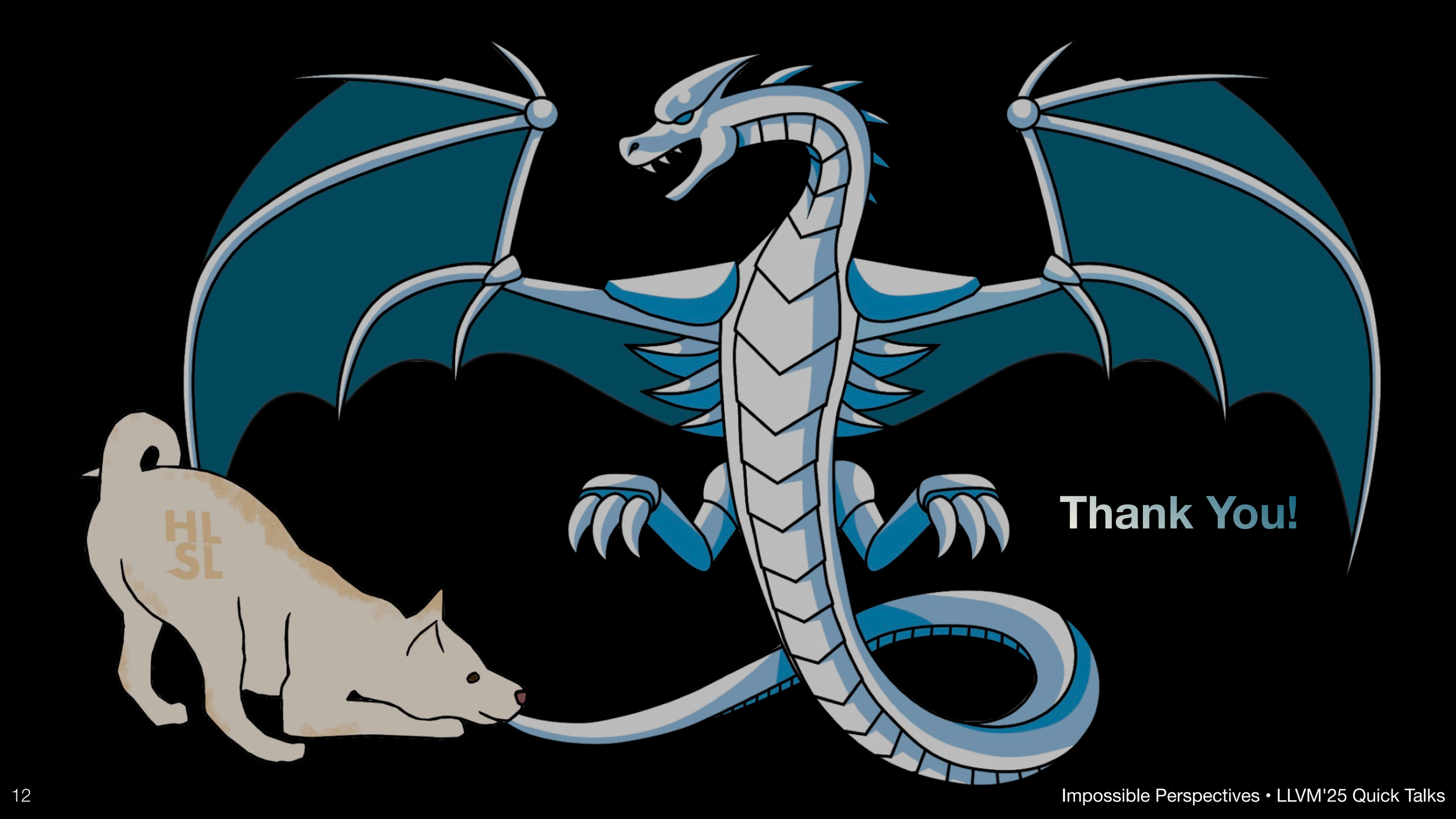
Implicit vs Explicit Padding

- `[ n x i8 ]` is ambiguous
- Need original type for GPU driver metadata
- The SPIR-V type encodes index and offset
- We used `target("Padding", n)`



Potential Outside of HLSL

- Should we have a first class `pad8` type?
- Full control of layouts (vk::offset)
- Scalar replacement of aggregates (SROA)
- Transformation validation (alive)
- Sanitizers



Thank You!