Combining SDFs in a single draw call

https://github.com/PsychedelicOrange/byoe/pull/10

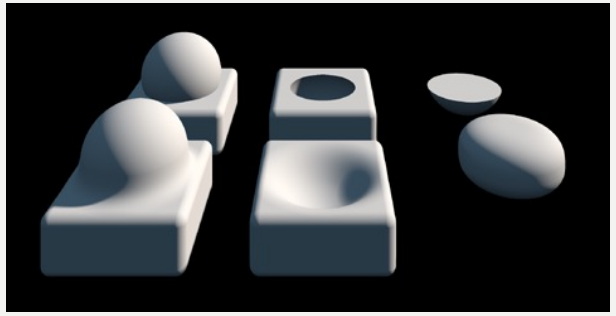
- Phani Srikar

https://github.com/PsychedelicOrange/byoe/pull/10

INTRO: Terminology

- **Primitives:** single SDF function drawn in a single draw call.
- **Object:** Made of one or more Primitives/Objects, more complex structure to combing primitives and complex SDF shapes in a single draw call.
  - Stores a binary hierarchy of node references and combines them using the blend operation

Primitives



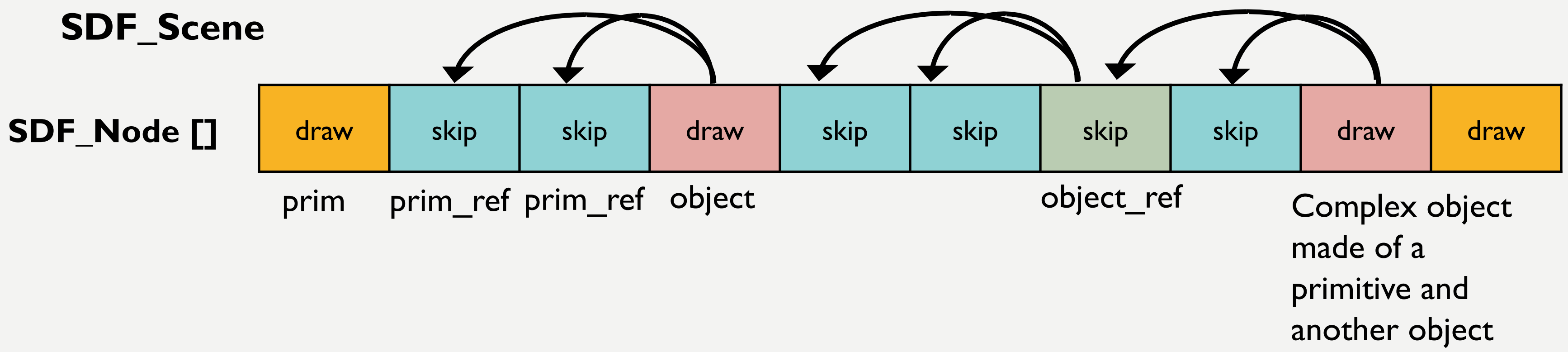
Objects made of one or more primitives/objects

```
typedef struct SDF_Primitive {
    SDF_PrimitiveType type;
    Transform transform;
    SDF_Material material;
} SDF_Primitive;
```

```
typedef struct SDF_Object {
    SDF_BlendType type;
    int prim_a;
    int prim_b;
} SDF_Object;
```

Scene storage: CPU

- Scene stores a linear hierarchy of nodes (primitives/objects) on the CPU, this is uploaded to the GPU
  - Those marked as ref nodes will be skipped from direct draw calls
  - Ref nodes are used for combining nodes of an object using the blend functions from within the raymarching shader

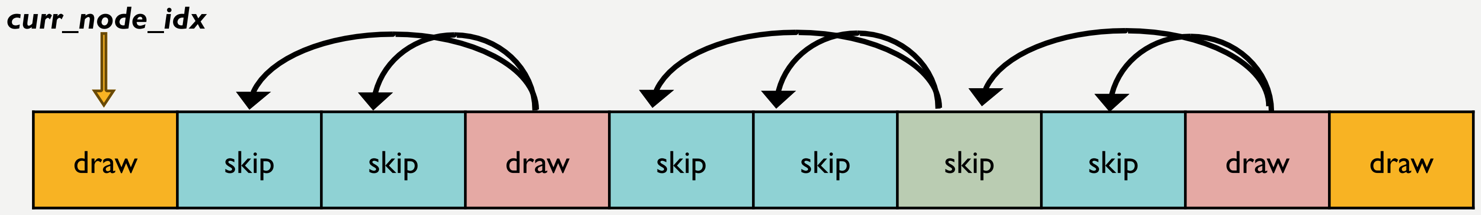


```
typedef enum SDF_NodeType
{
    SDF_NODE_PRIMITIVE,
    SDF_NODE_OPERATION
} SDF_NodeType;

typedef struct SDF_Node
{
    SDF_NodeType type;
    union
    {
        SDF_Primitive primitive;
        SDF_Object operation;
    };
    bounding_sphere bounds;
    bool            is_ref_node;
    bool            is_culled;
} SDF_Node;
```

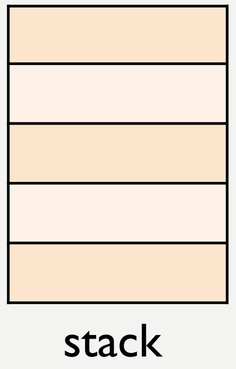
GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

draw 1: primitive



curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

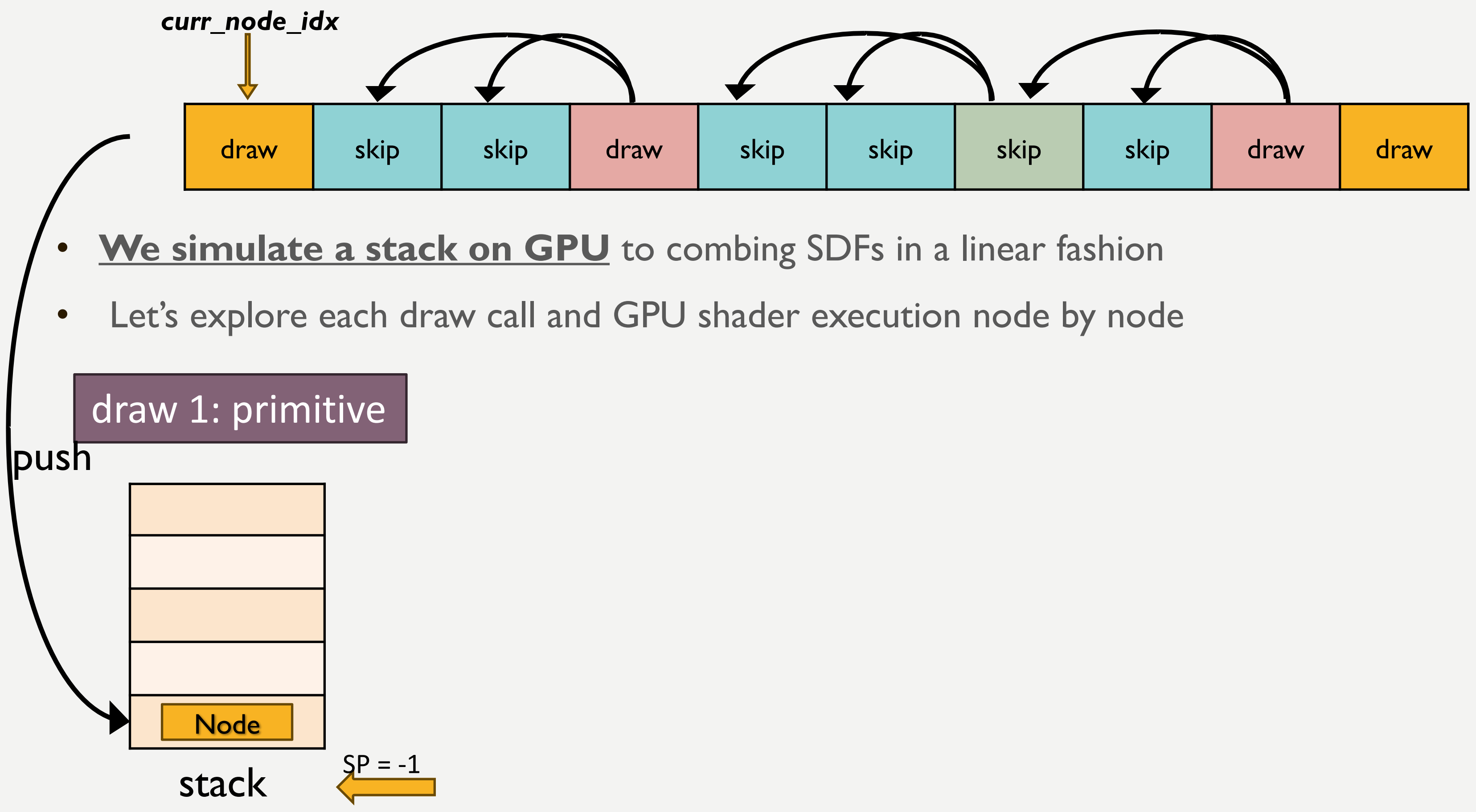
hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

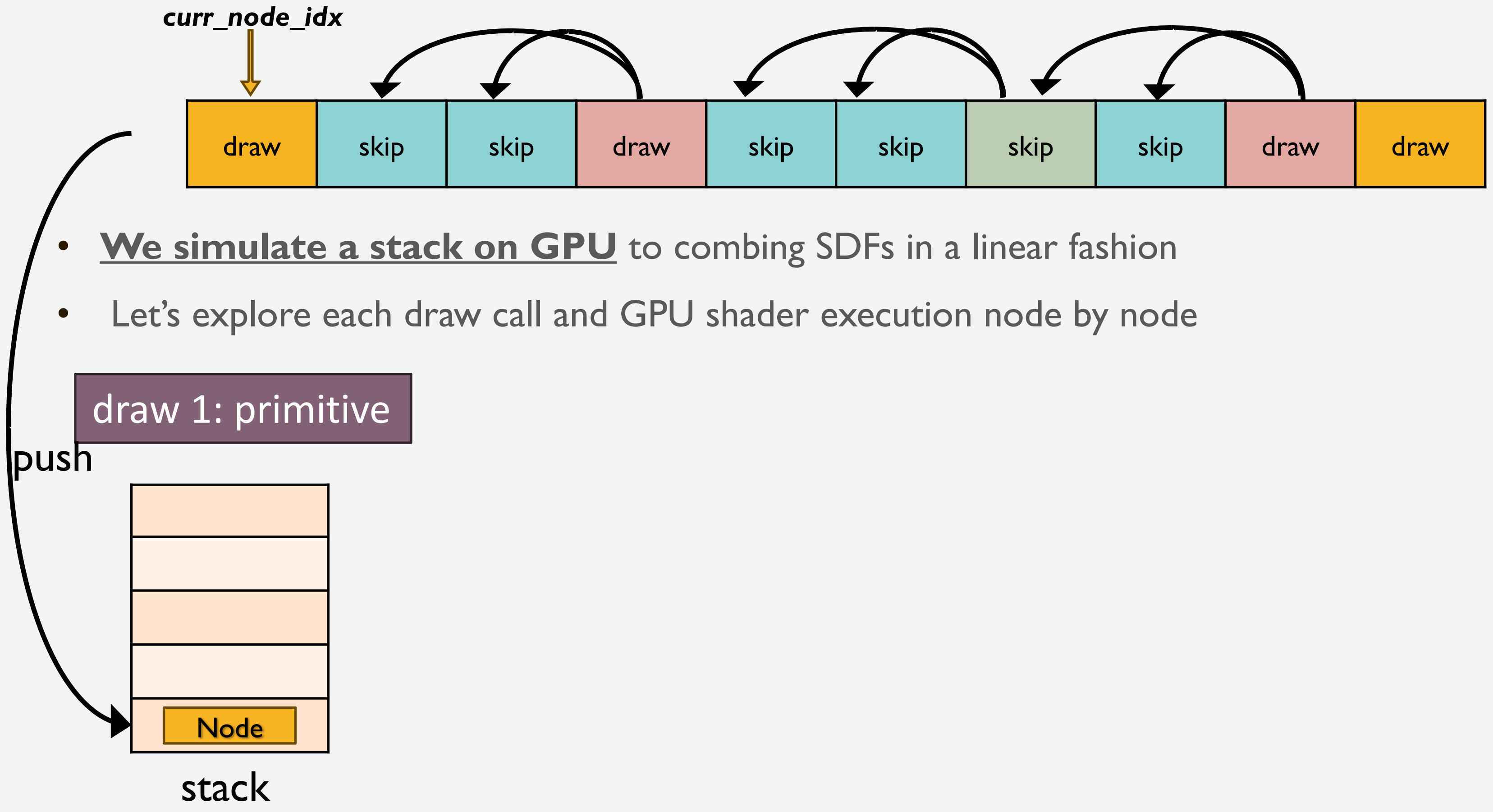
hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

draw 1: primitive

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

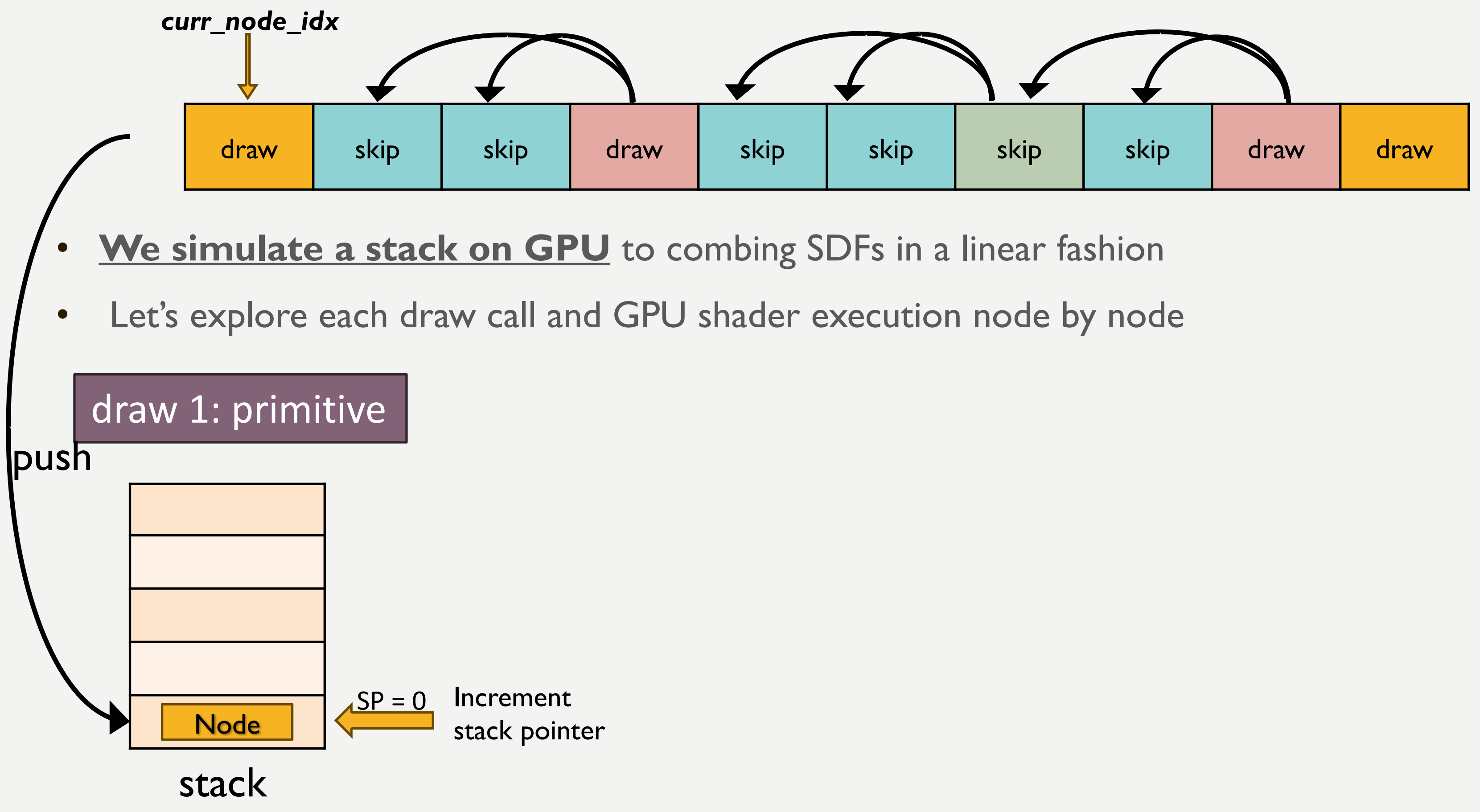
hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

draw 1: primitive

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

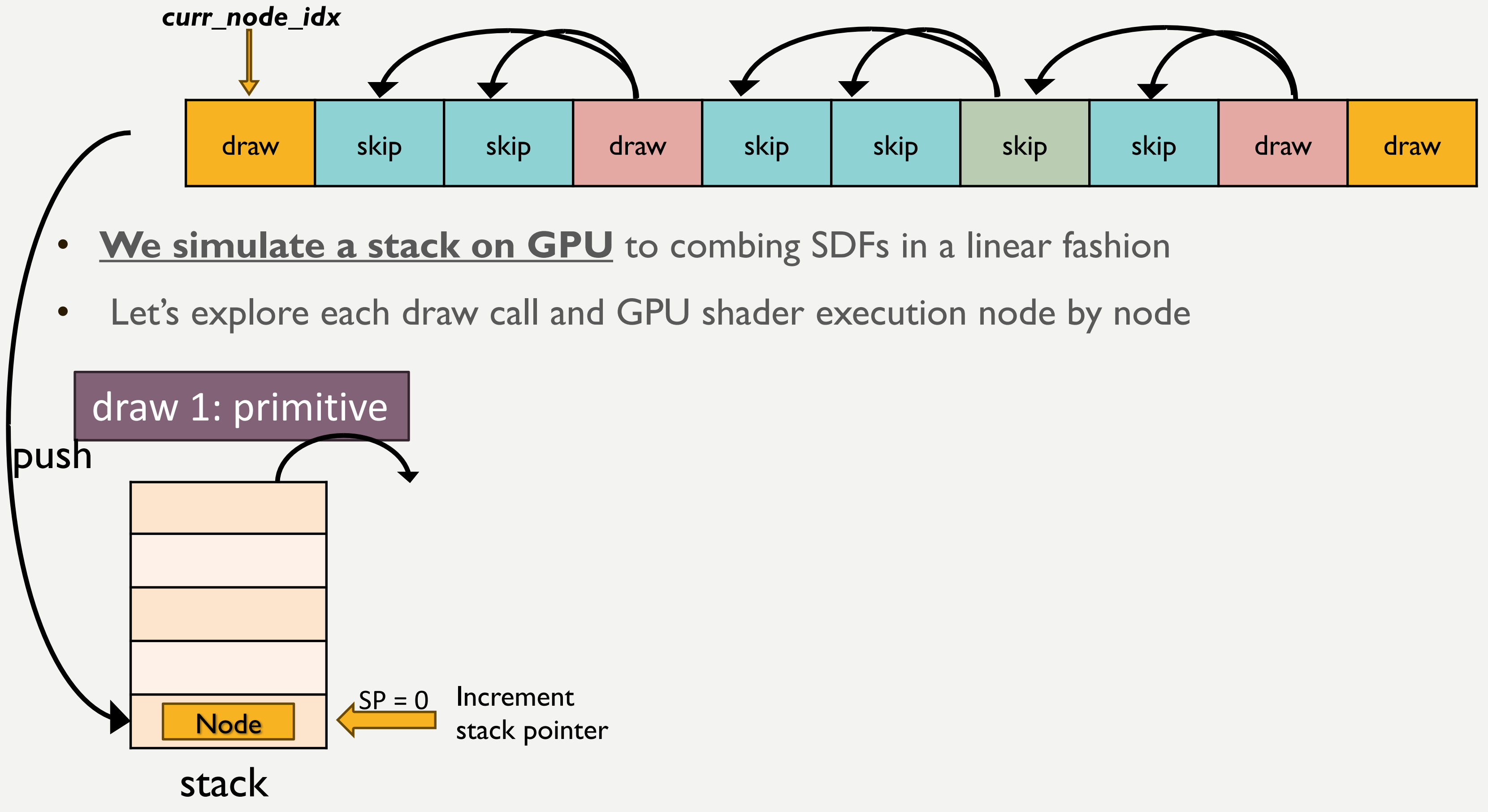
Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation

raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

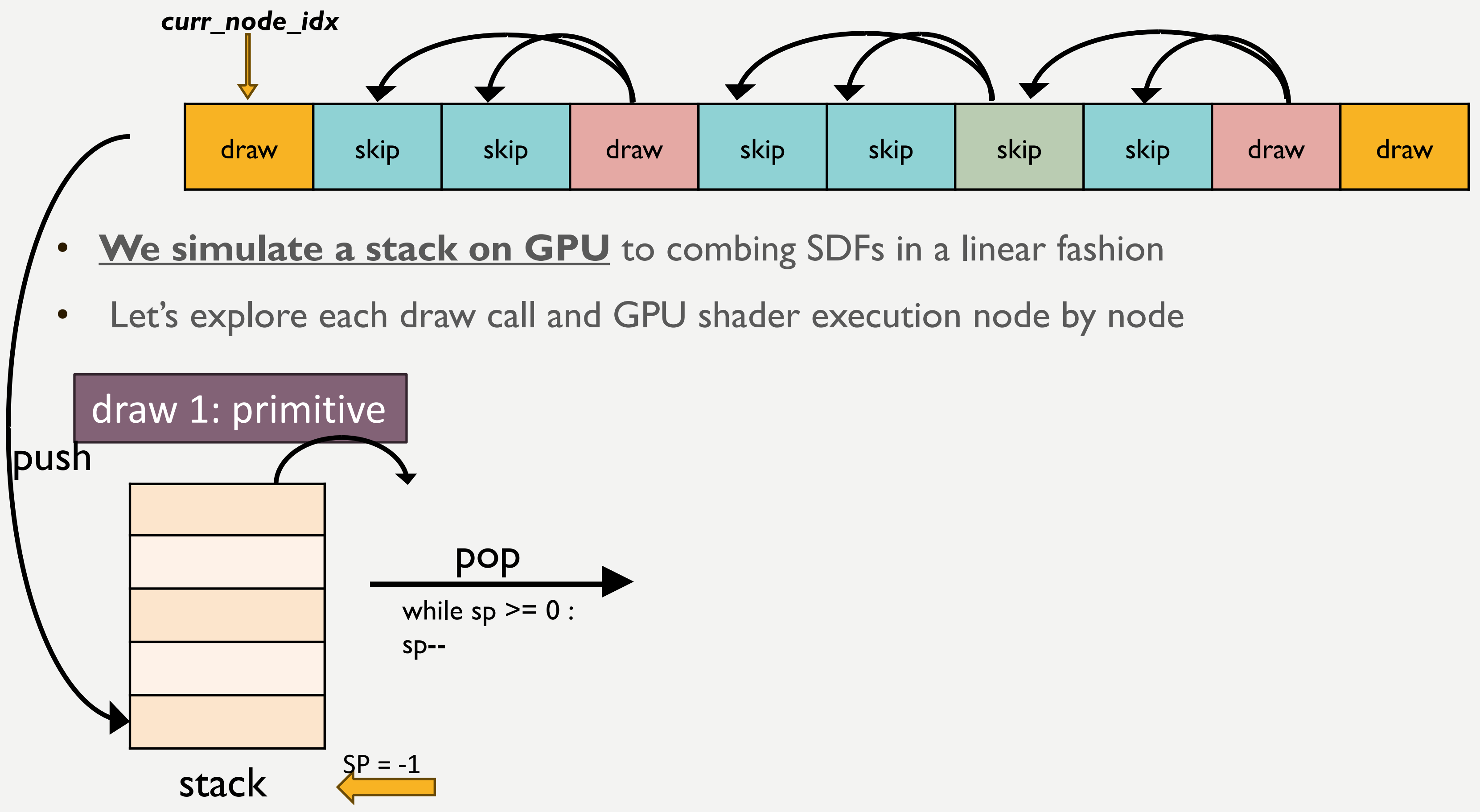
hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

draw 1: primitive

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

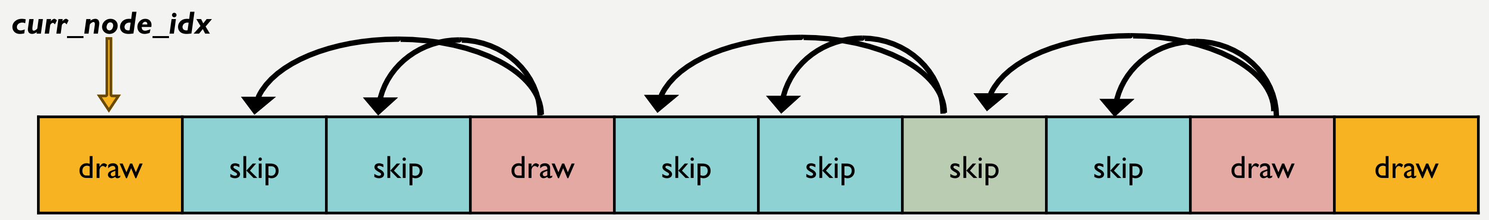
hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

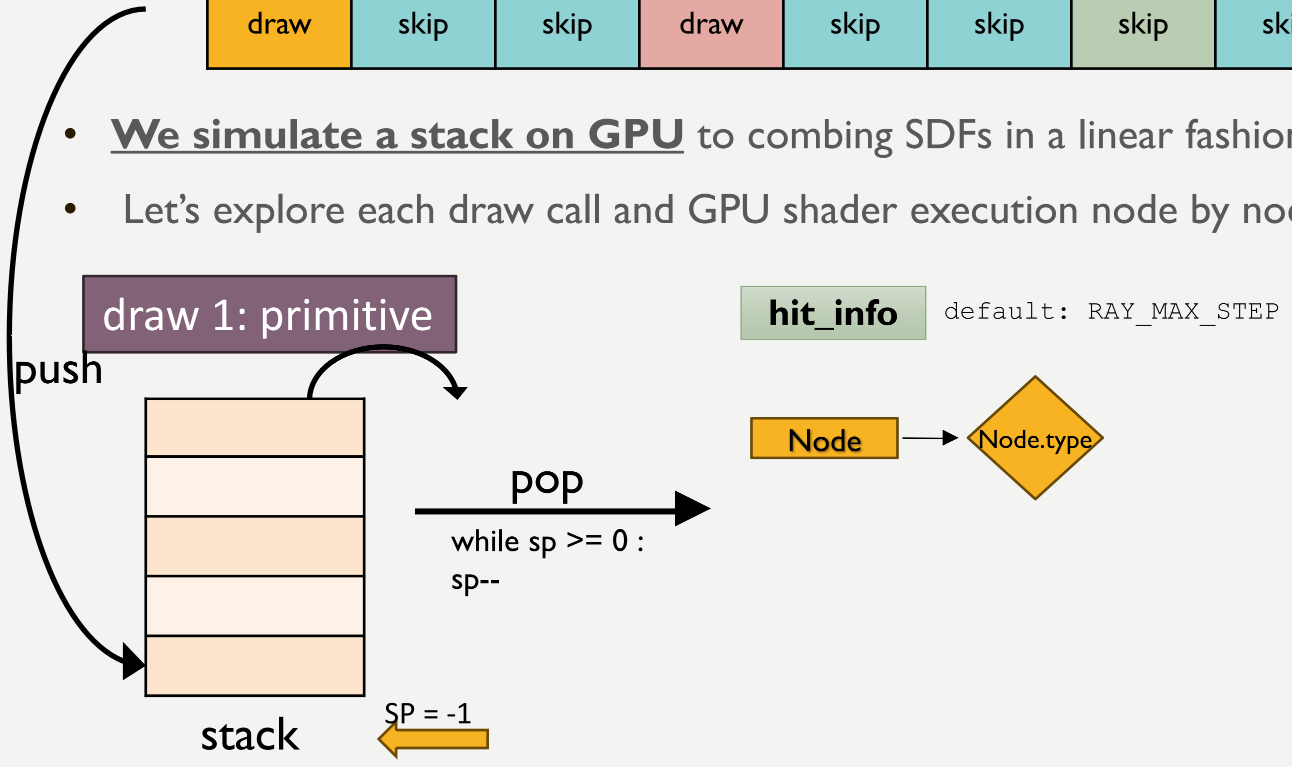
Scene SDF Calculation
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node



curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

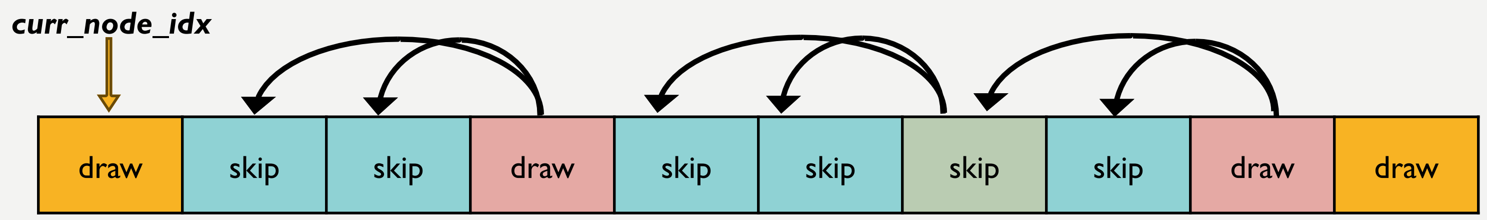
Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation

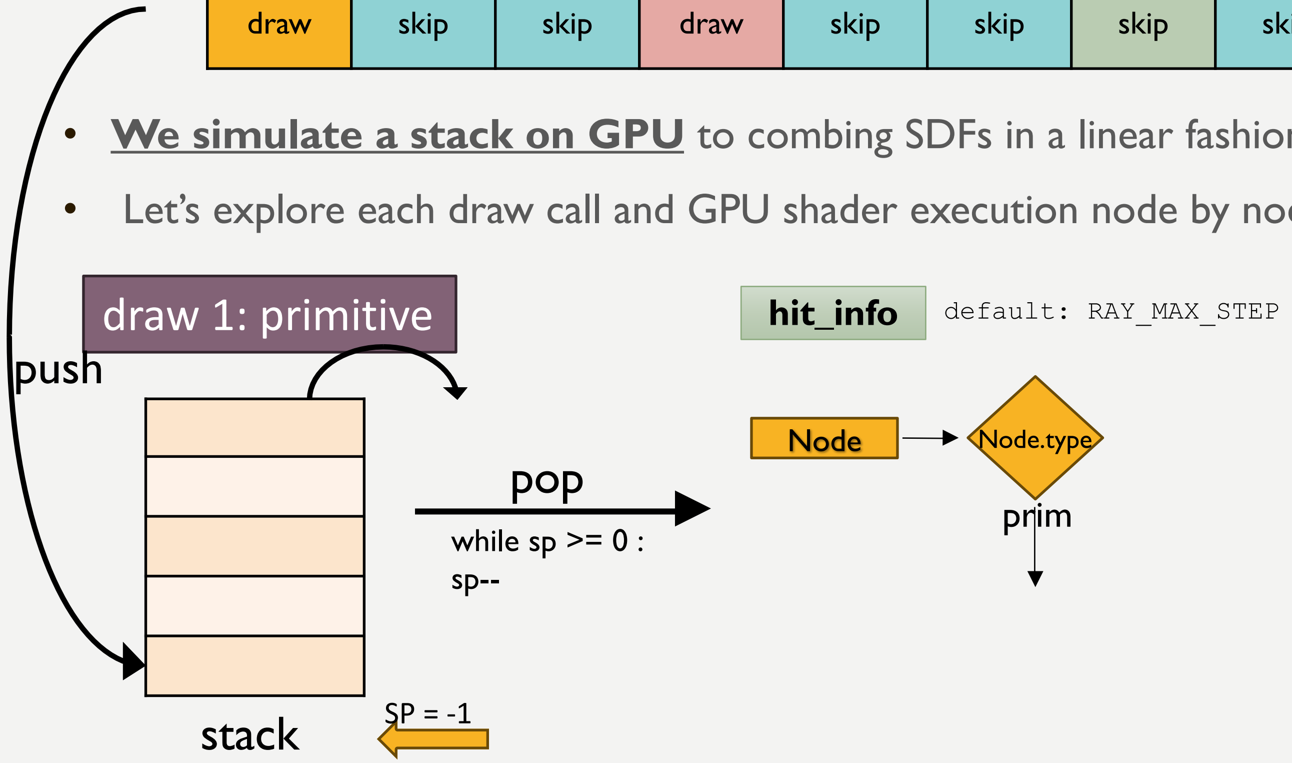
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node



curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

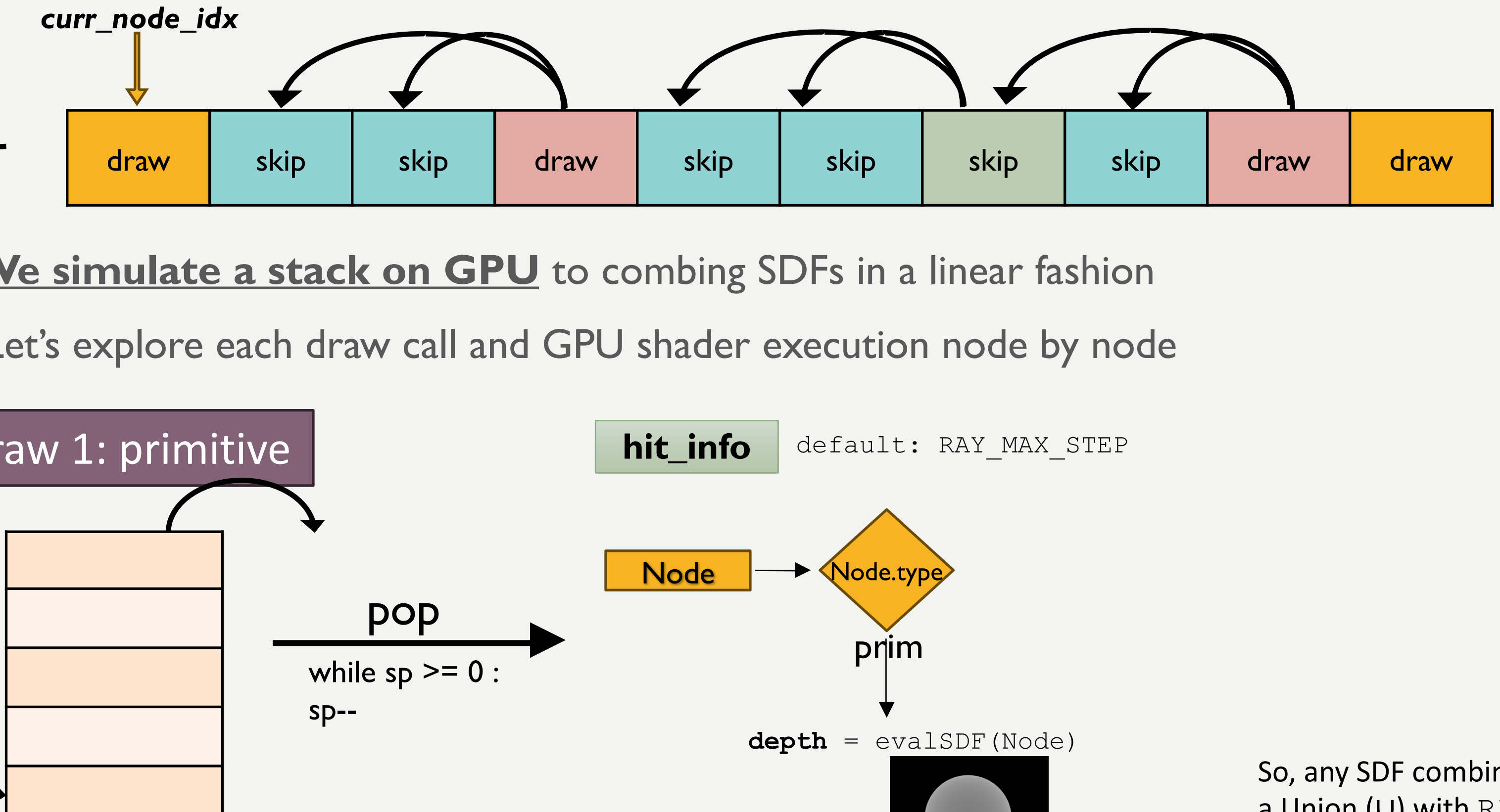
Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

Scene SDF Calculation

raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



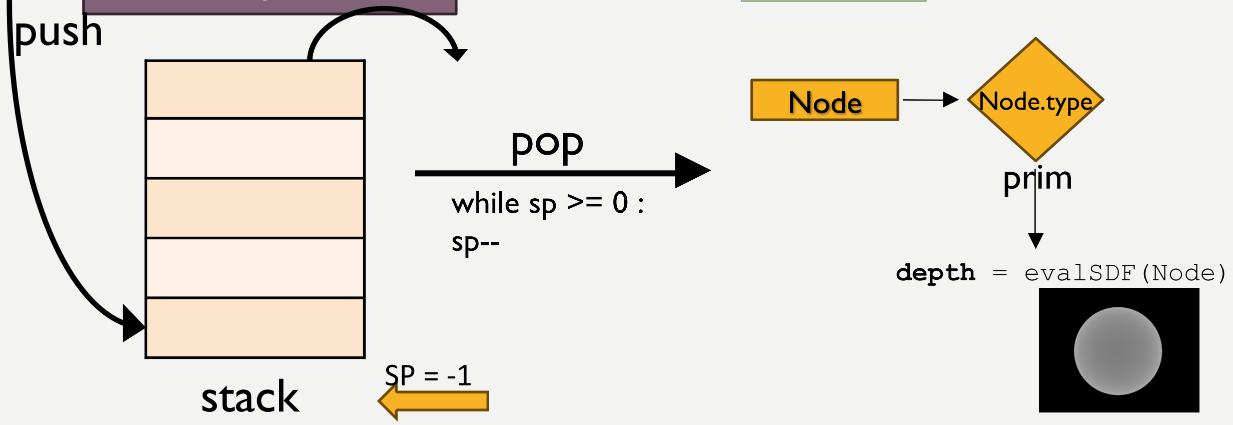
- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

draw 1: primitive



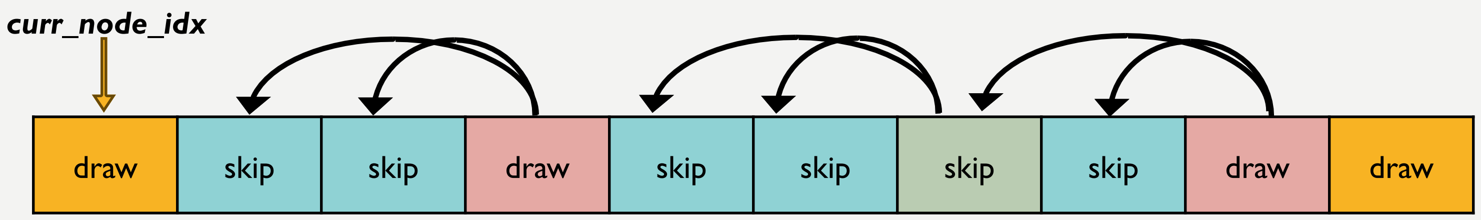
So, any SDF combination starts with a Union (∪) with `RAY_MAX_STEP`, since in this case there's just one, we evaluate and return once sp < 0.

Scene SDF Calculation

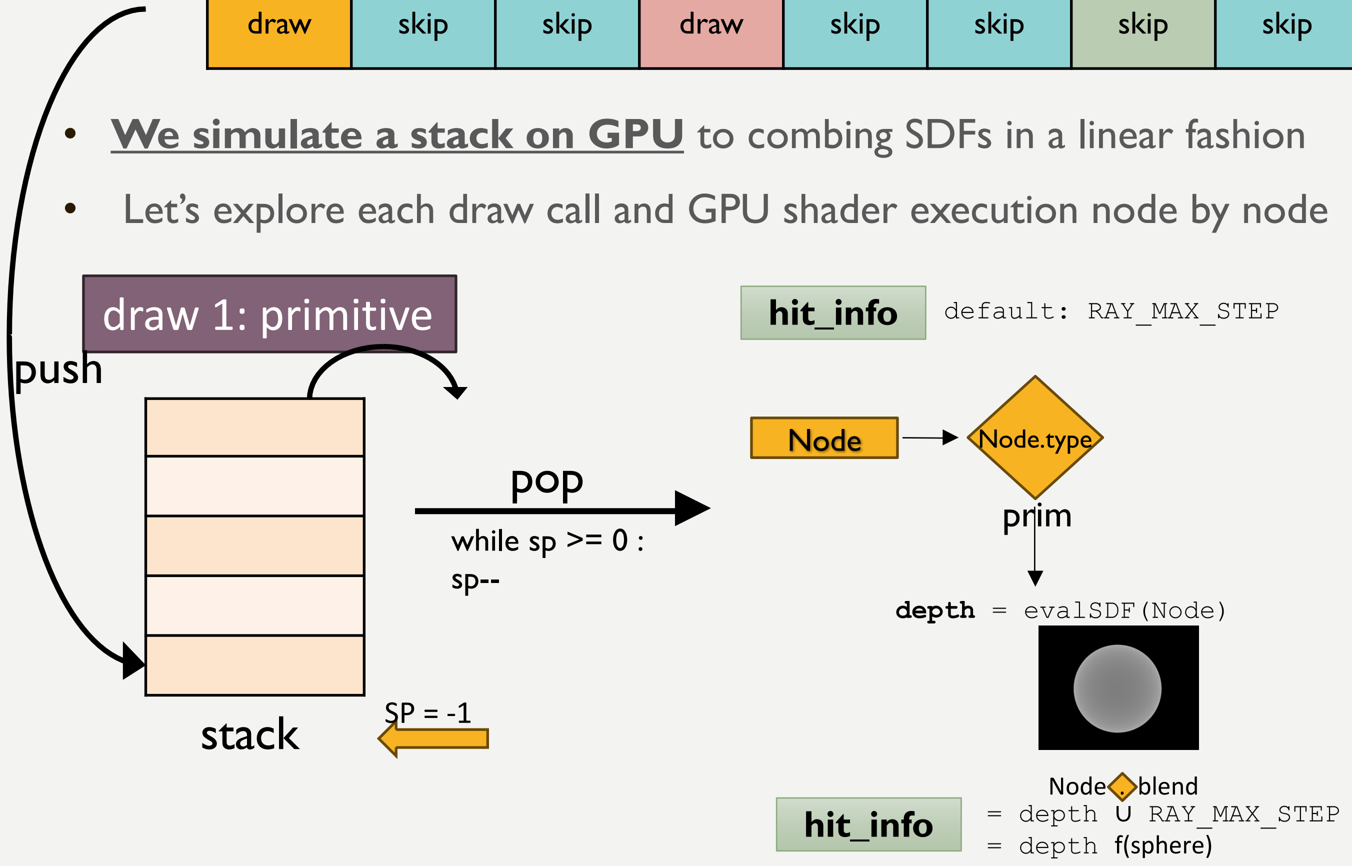
raymarch this function as usual

GPU drawing complex SDFs

- Consider the following scene nodes view from the GPU's POV



- **We simulate a stack on GPU** to combing SDFs in a linear fashion
- Let's explore each draw call and GPU shader execution node by node



So, any SDF combination starts with a Union (∪) with `RAY_MAX_STEP`, since in this case there's just one, we evaluate and return once sp < 0.

Scene SDF Calculation

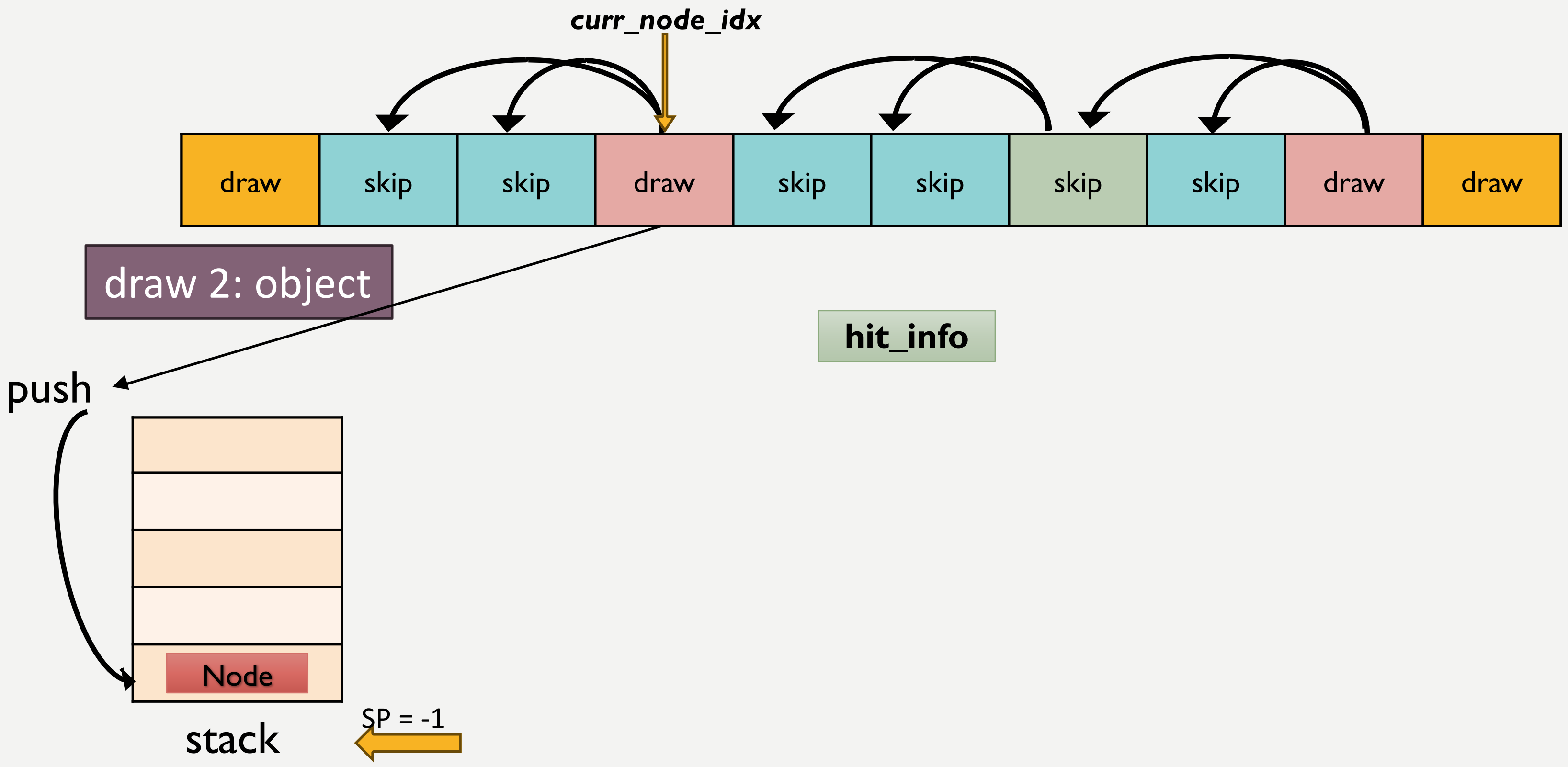
raymarch this function as usual

curr_node_idx:
Tells what node to draw from for the draw call, kind of like bindless access to node data

hit_info:
(global variable outside the while loop) stores the depth and material if the rays hit a surface

Basic ops:
- ***Union:*** *min(a, b)*
- ***Intersection:*** *max(a, b)*
- ***Subtraction:*** *max(a, -b)*
- + Smooth versions of above

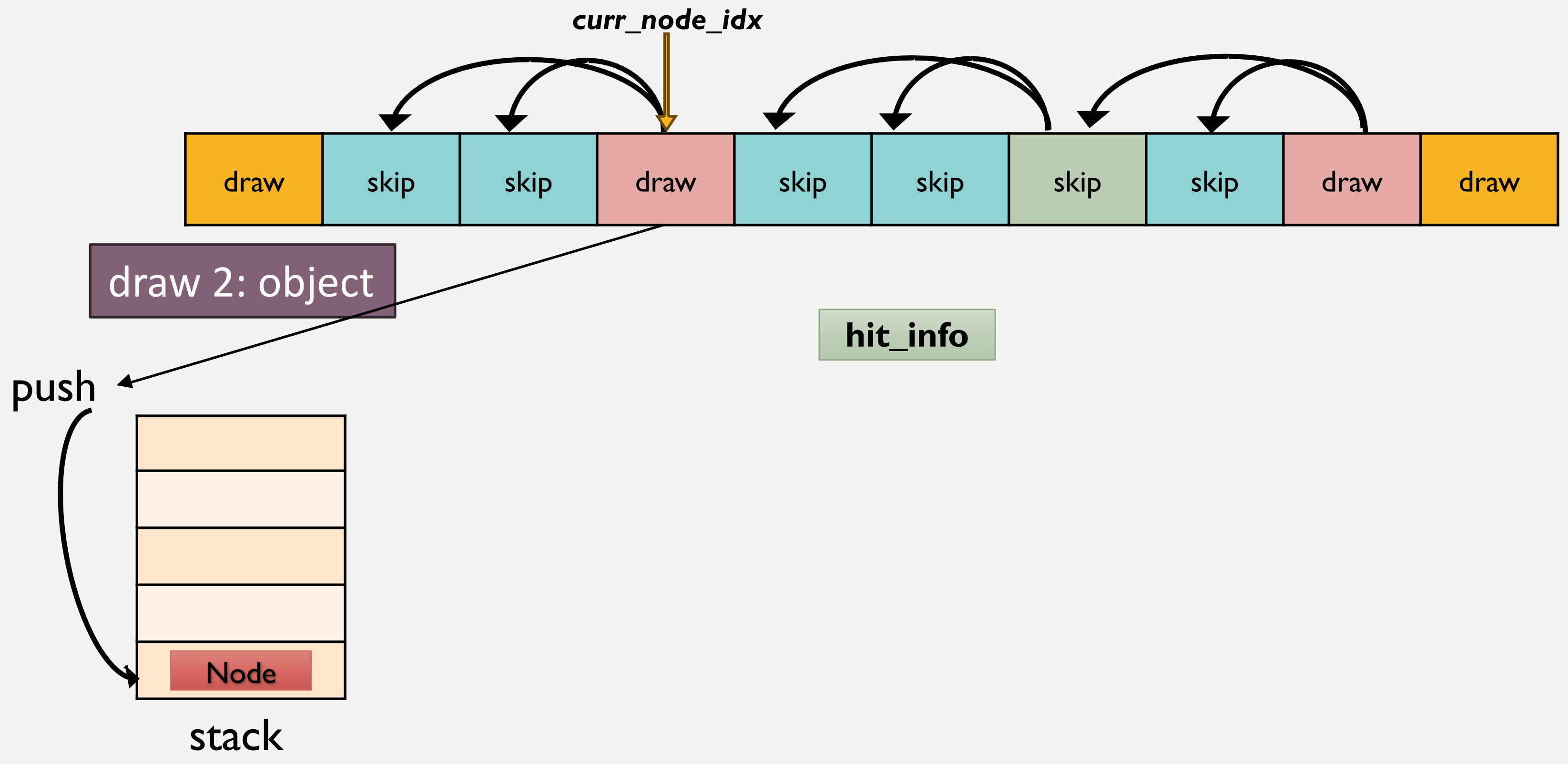
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

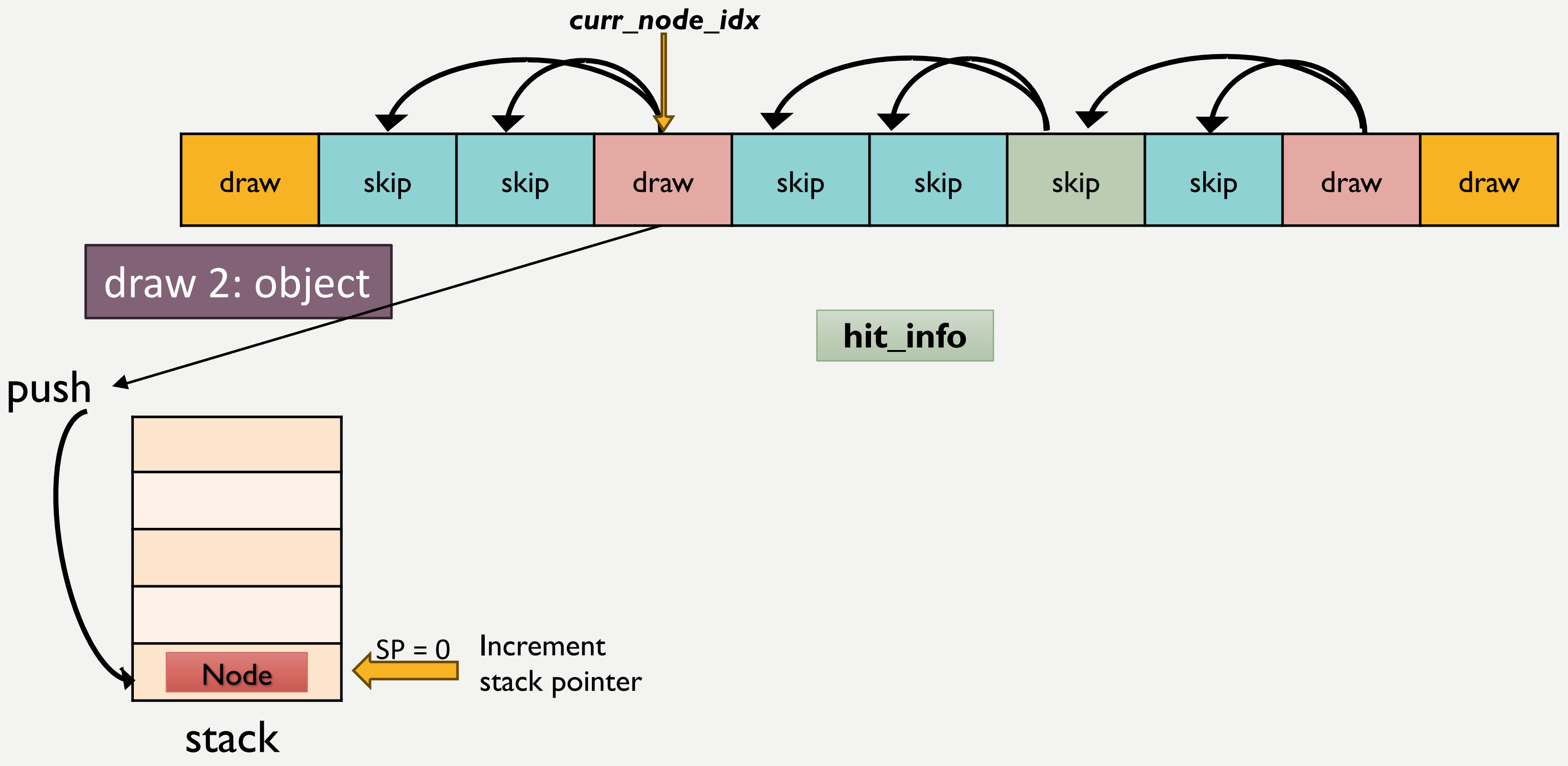
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

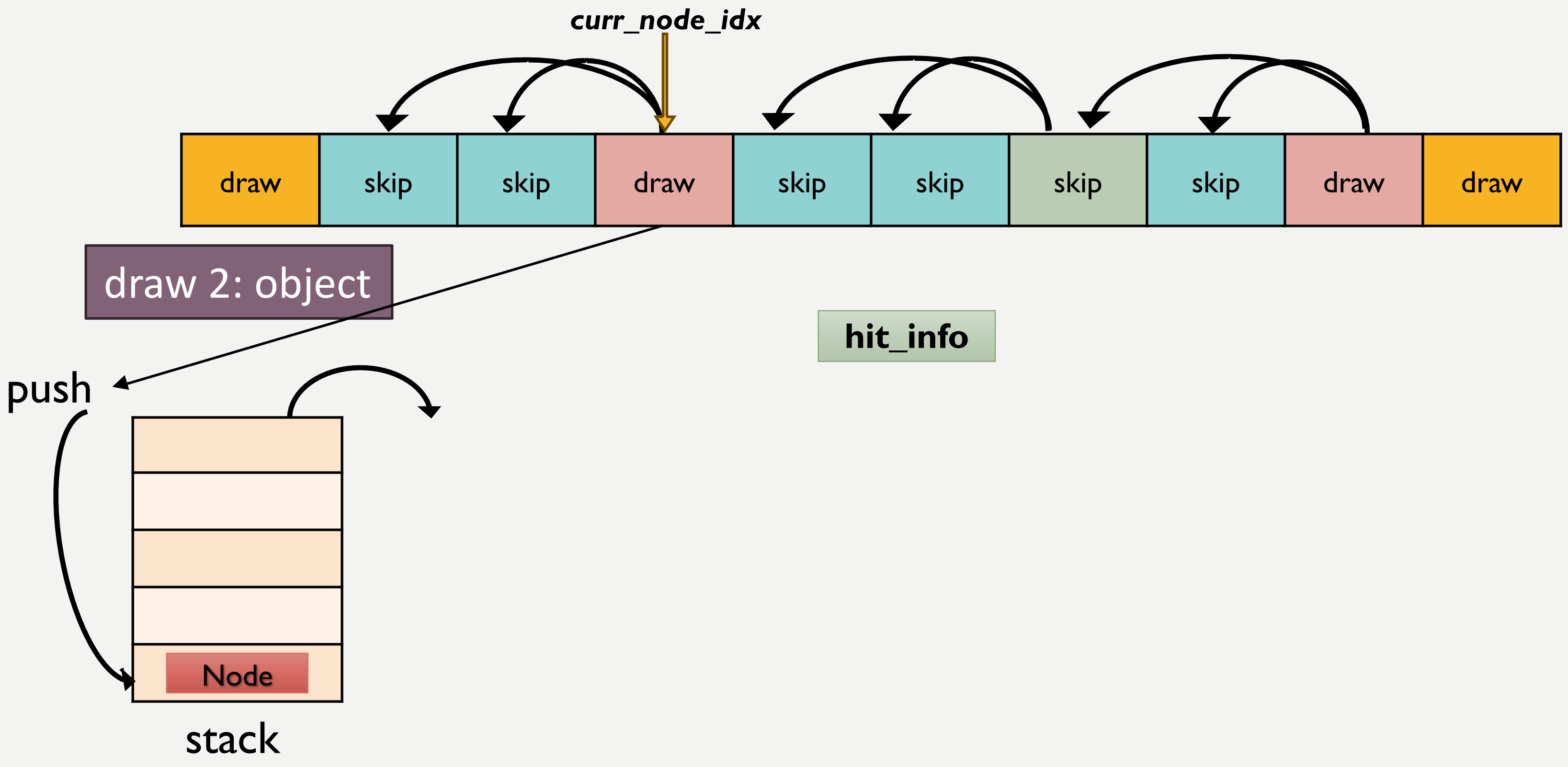
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

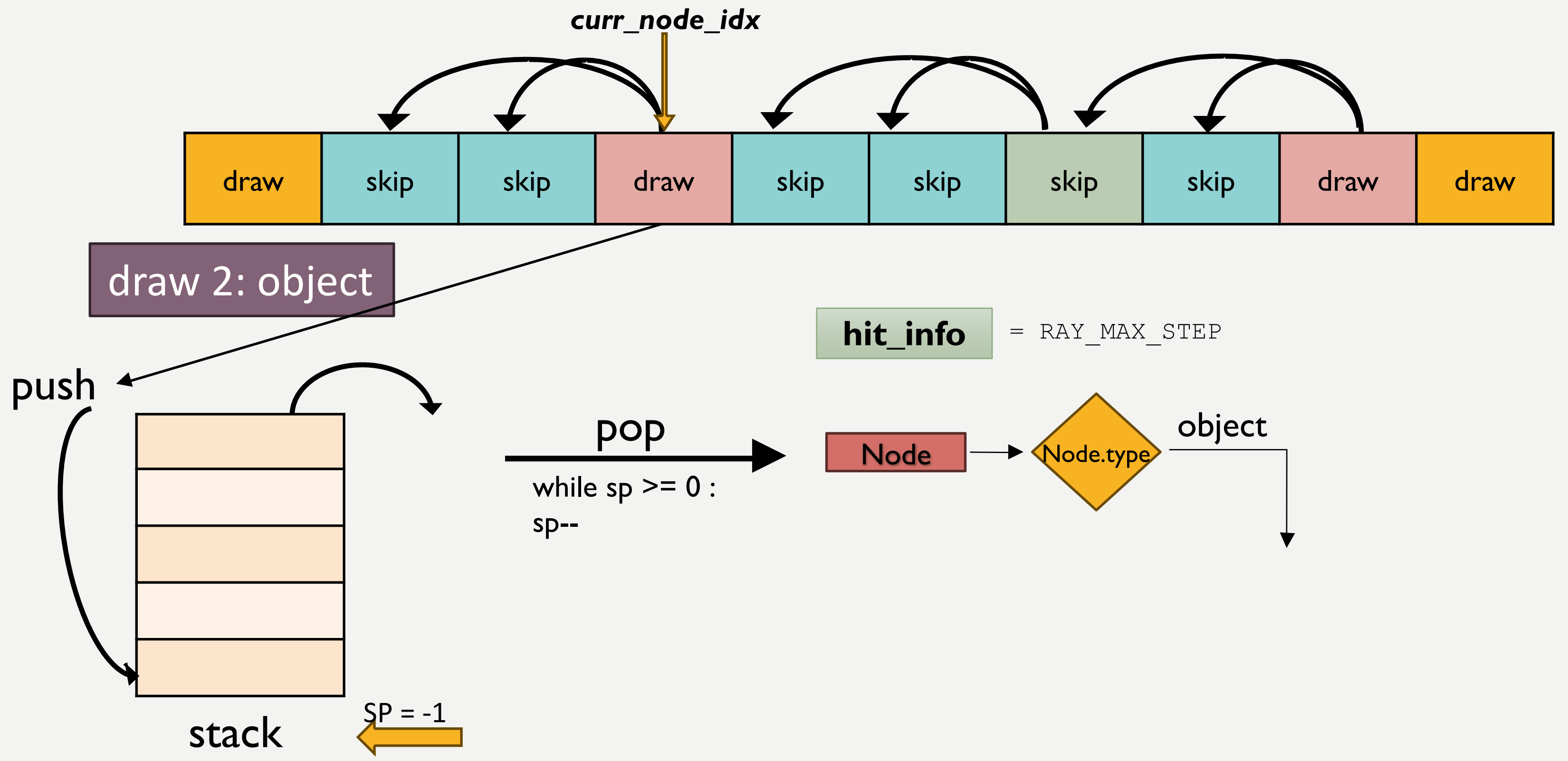
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

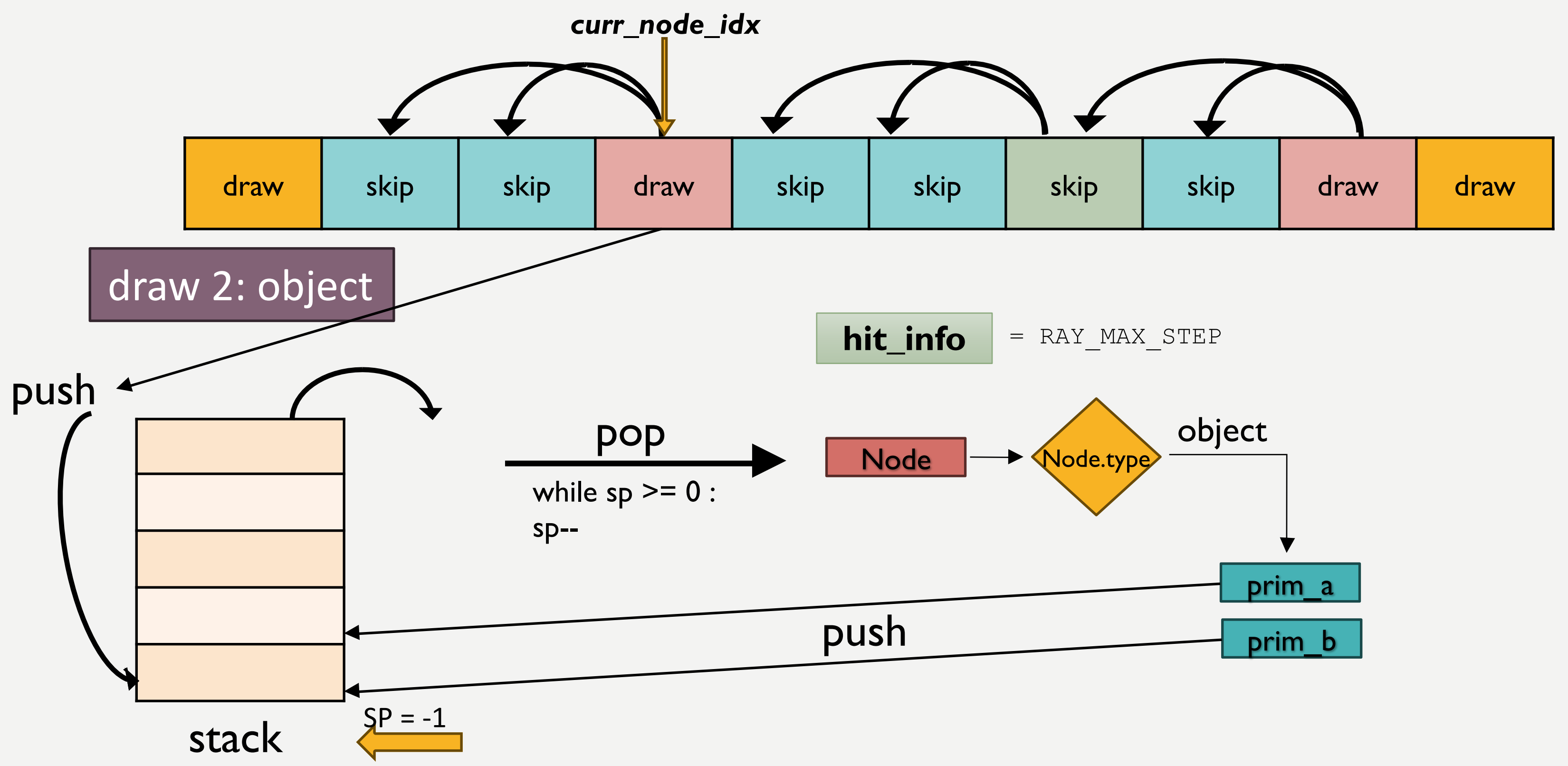
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

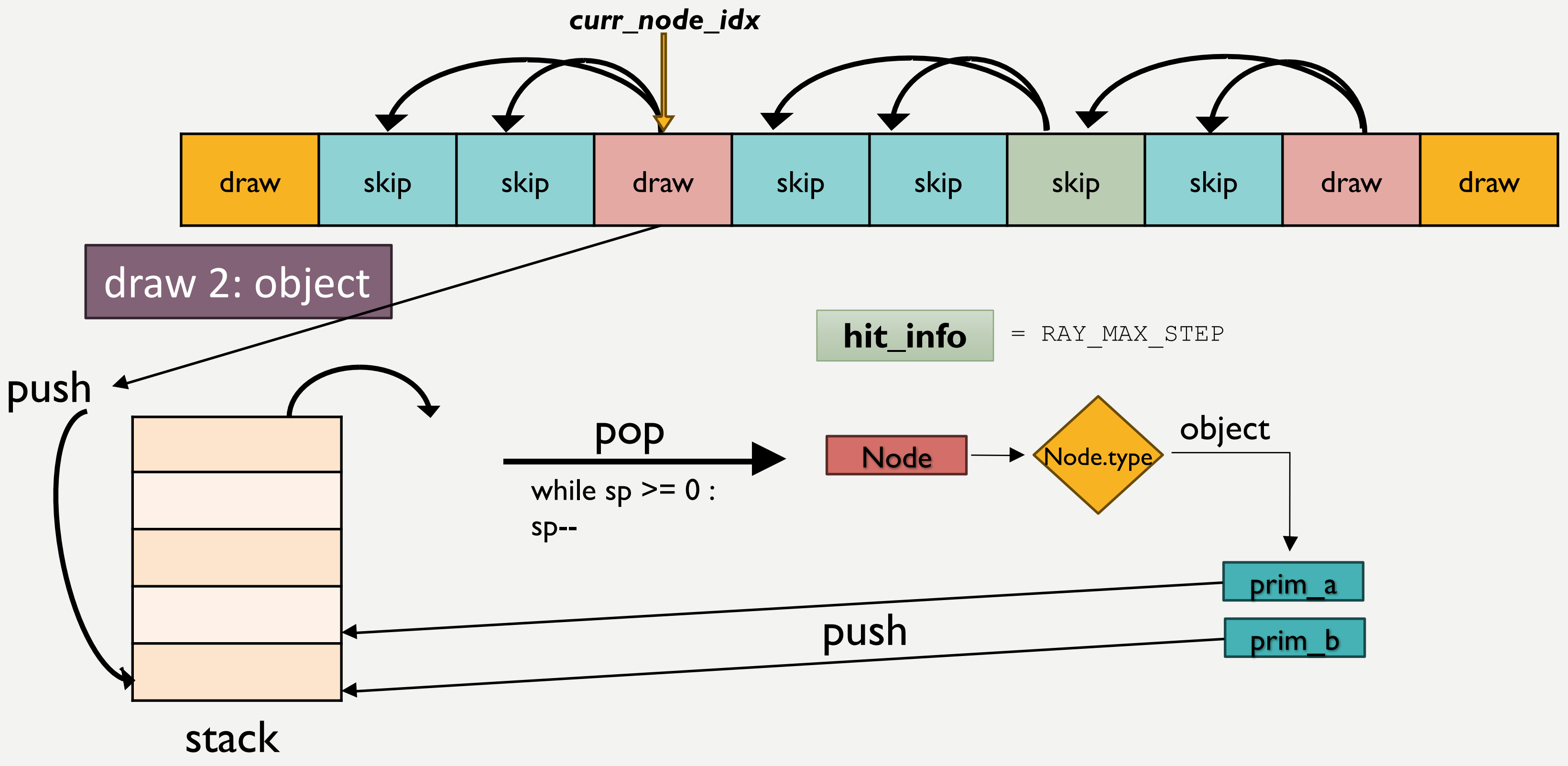
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

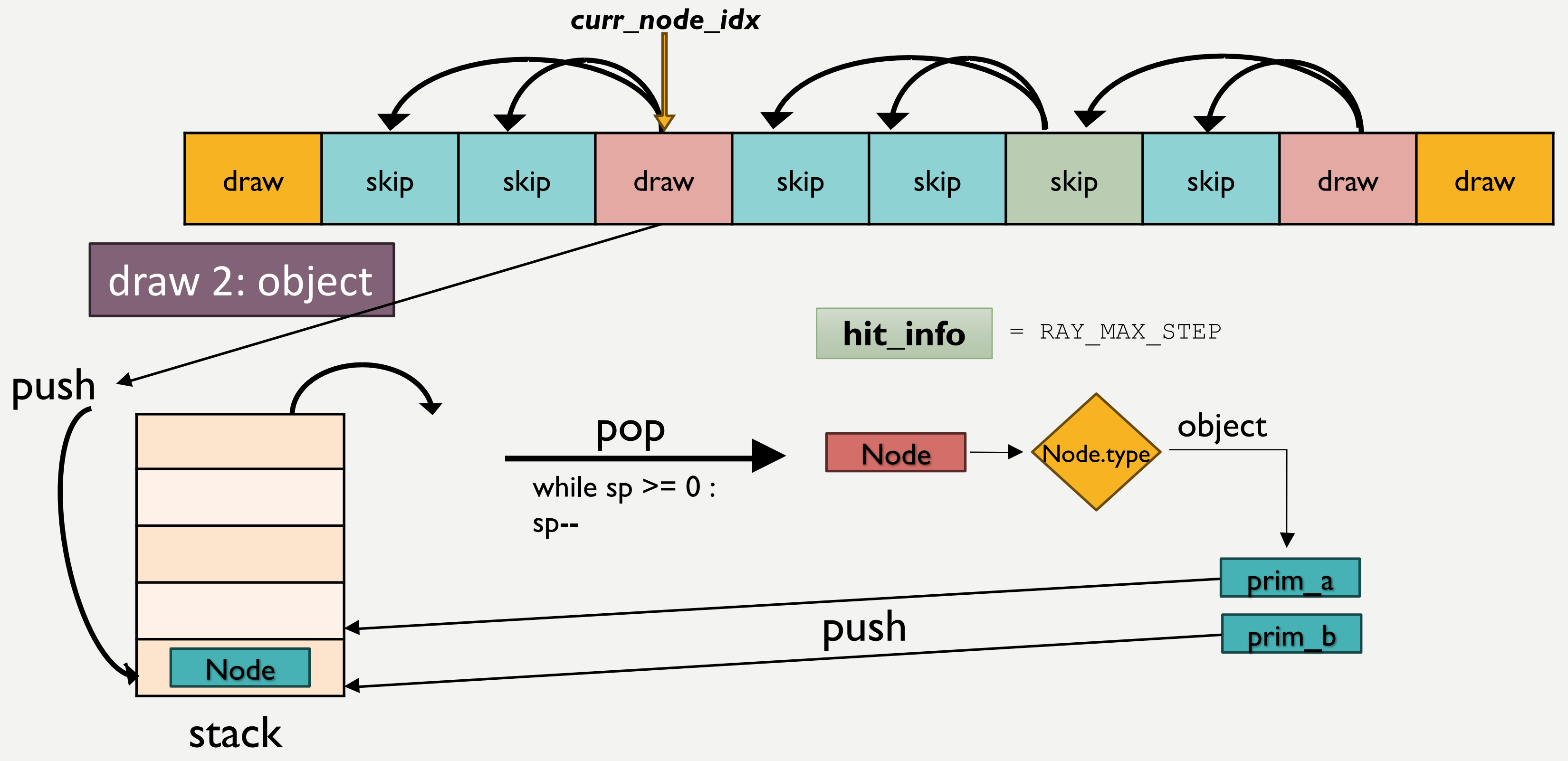
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

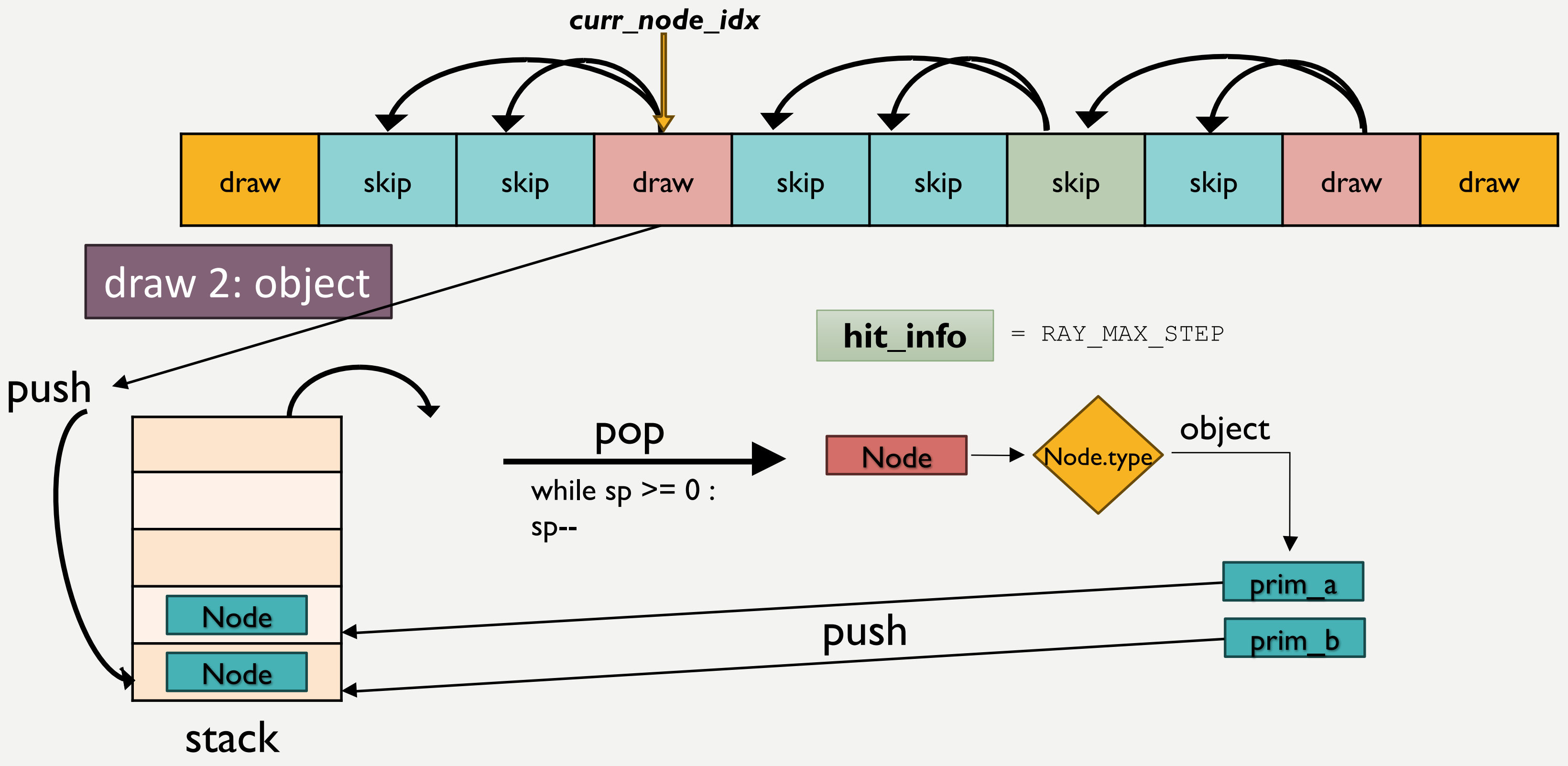
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

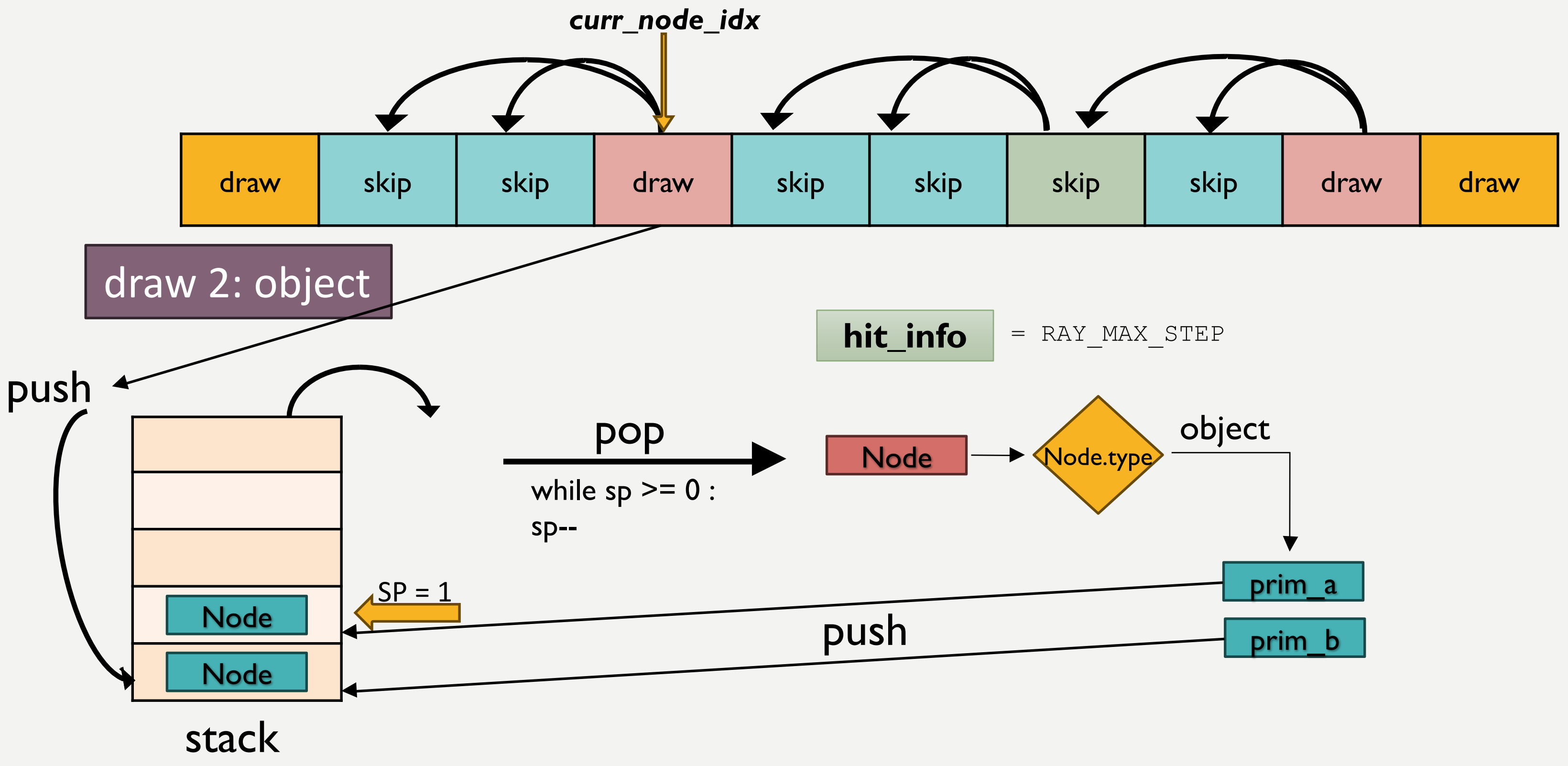
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

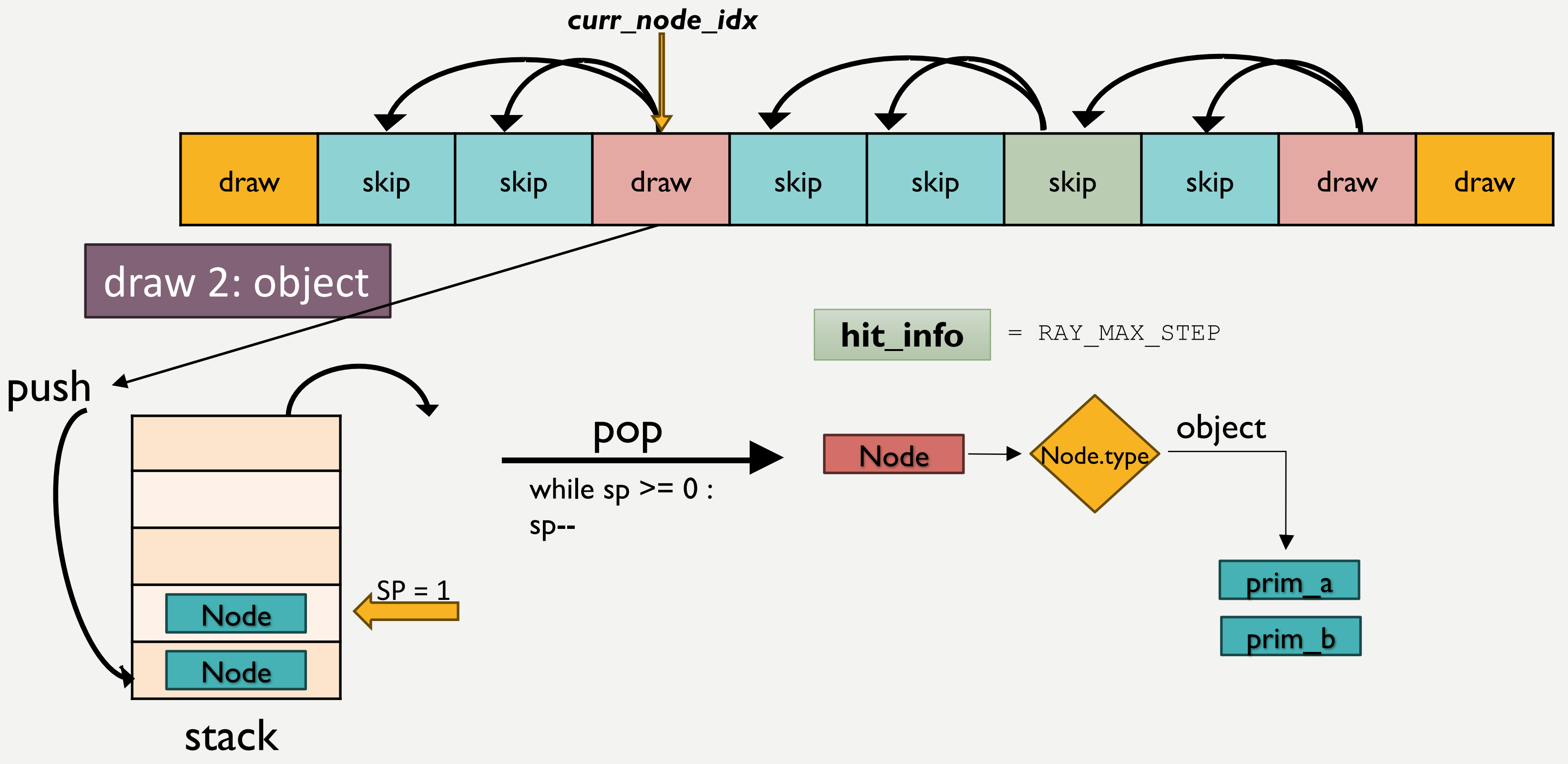
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

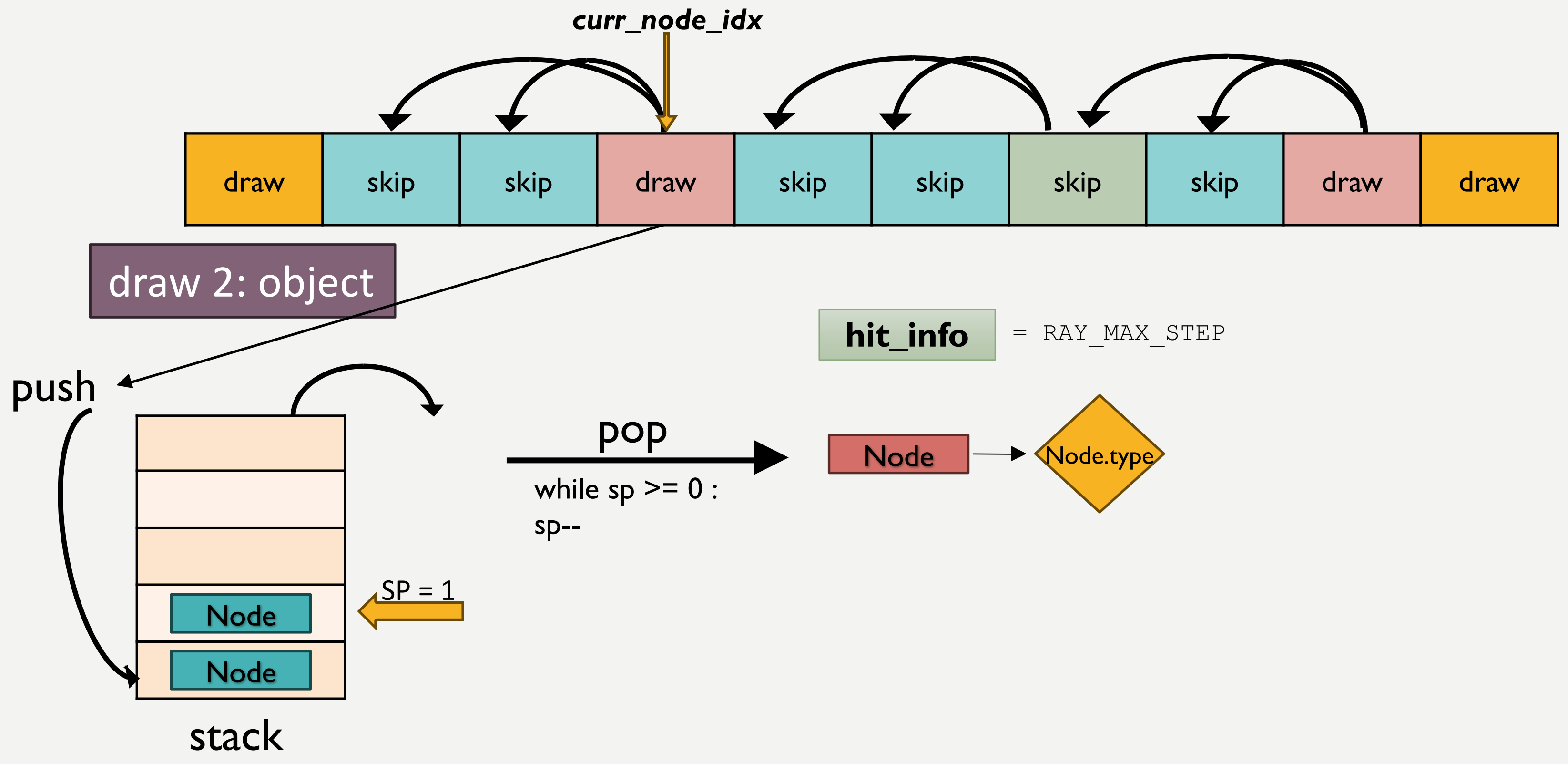
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

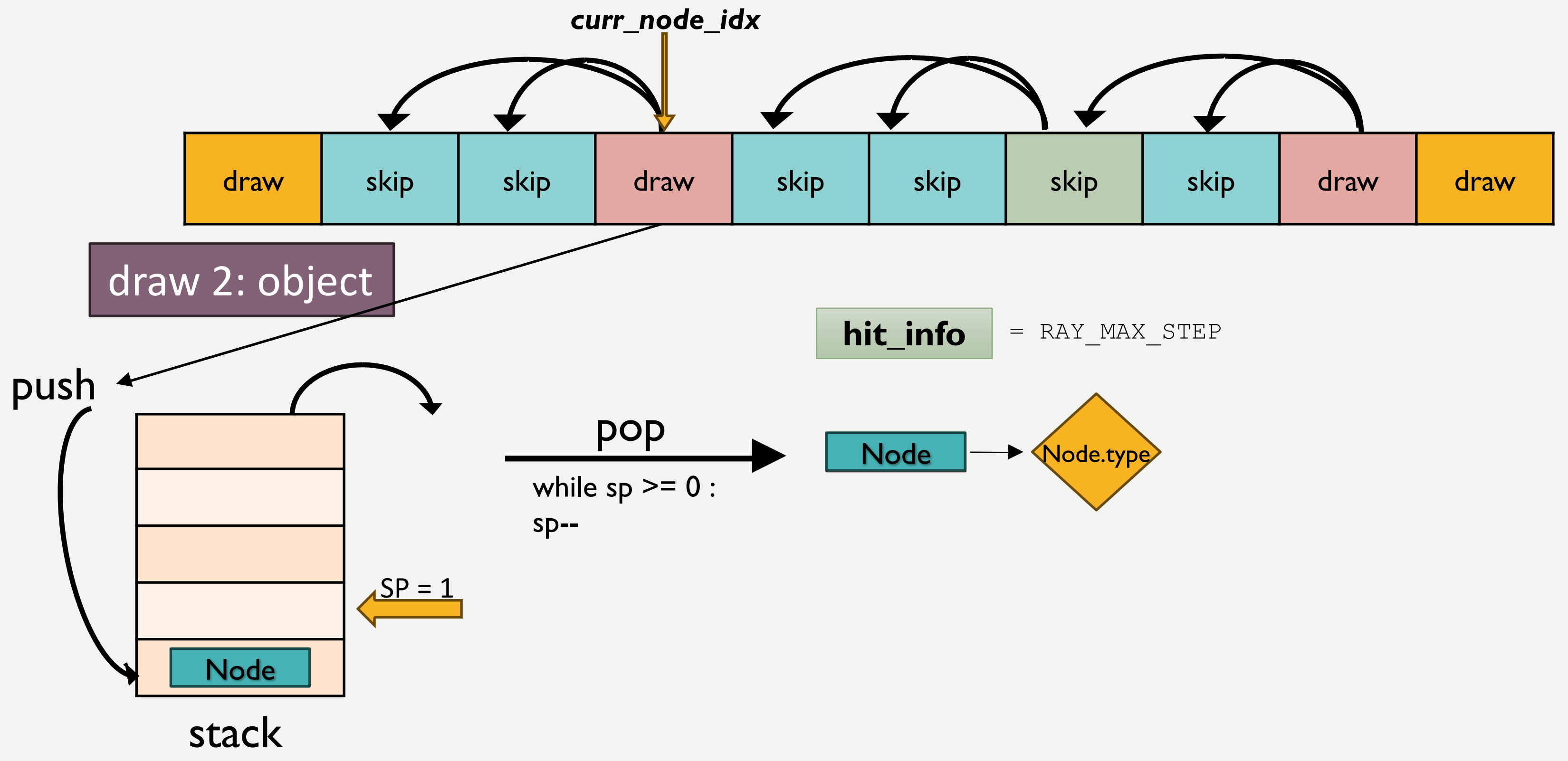
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

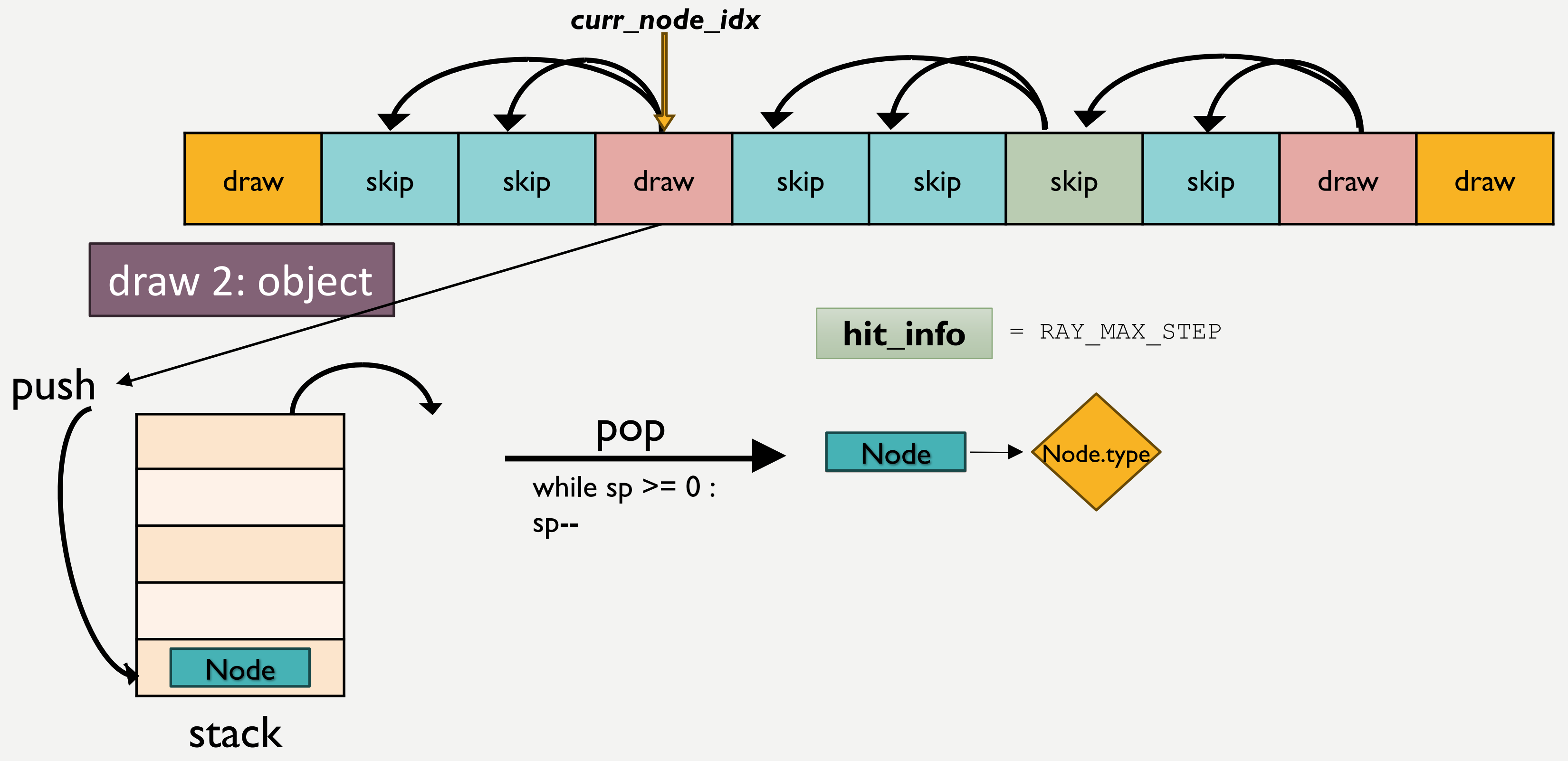
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

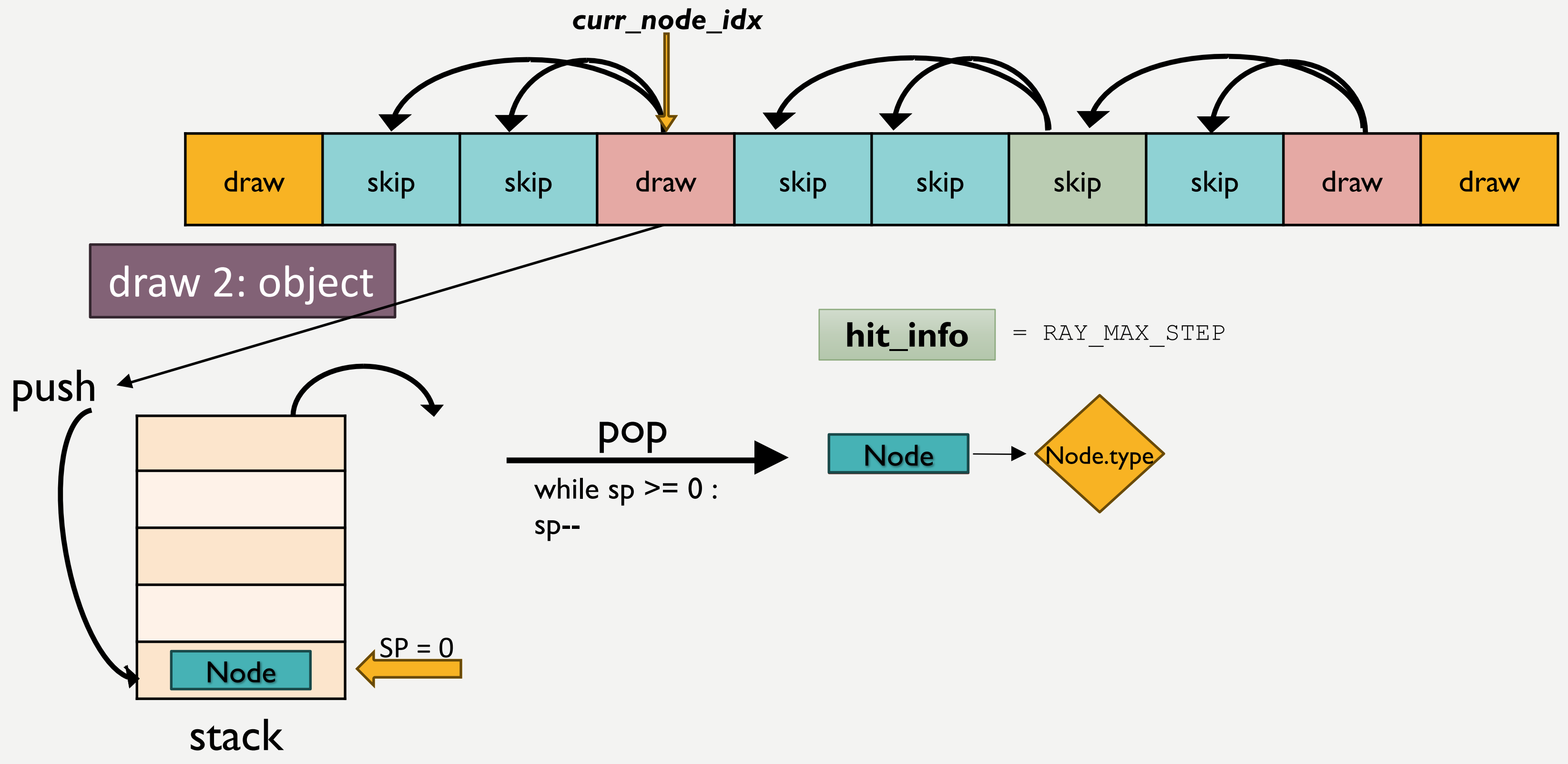
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

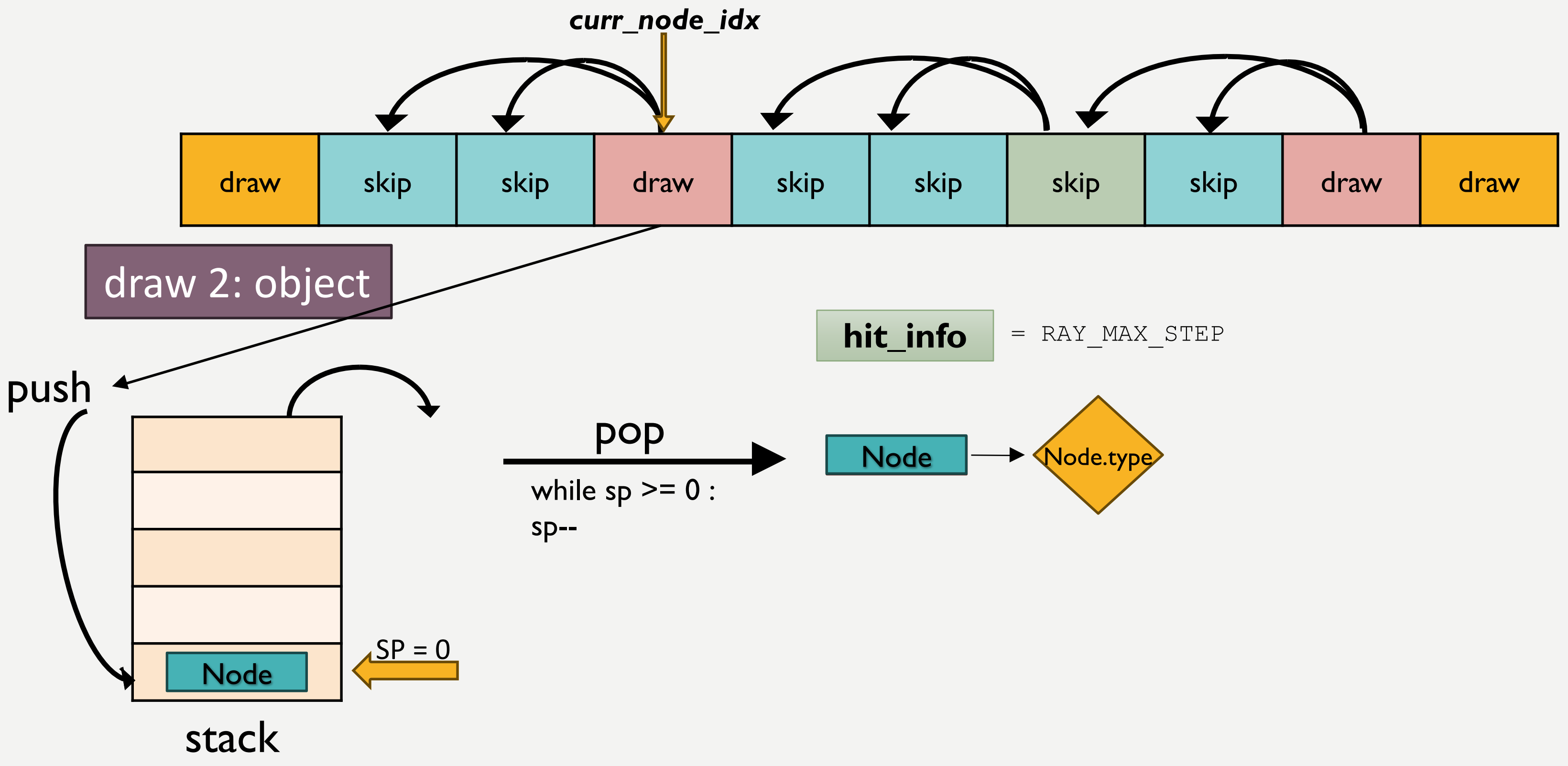
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

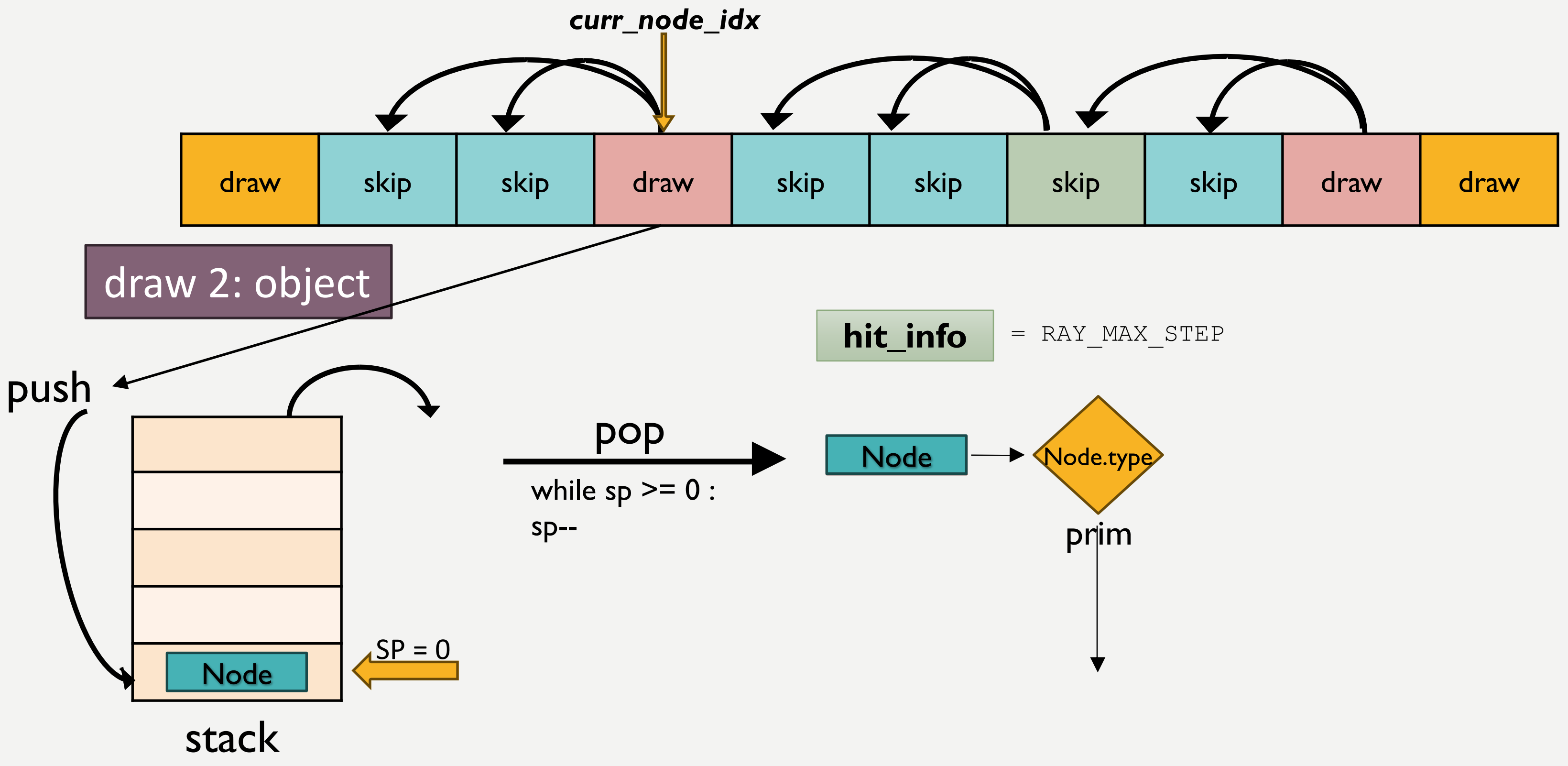
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

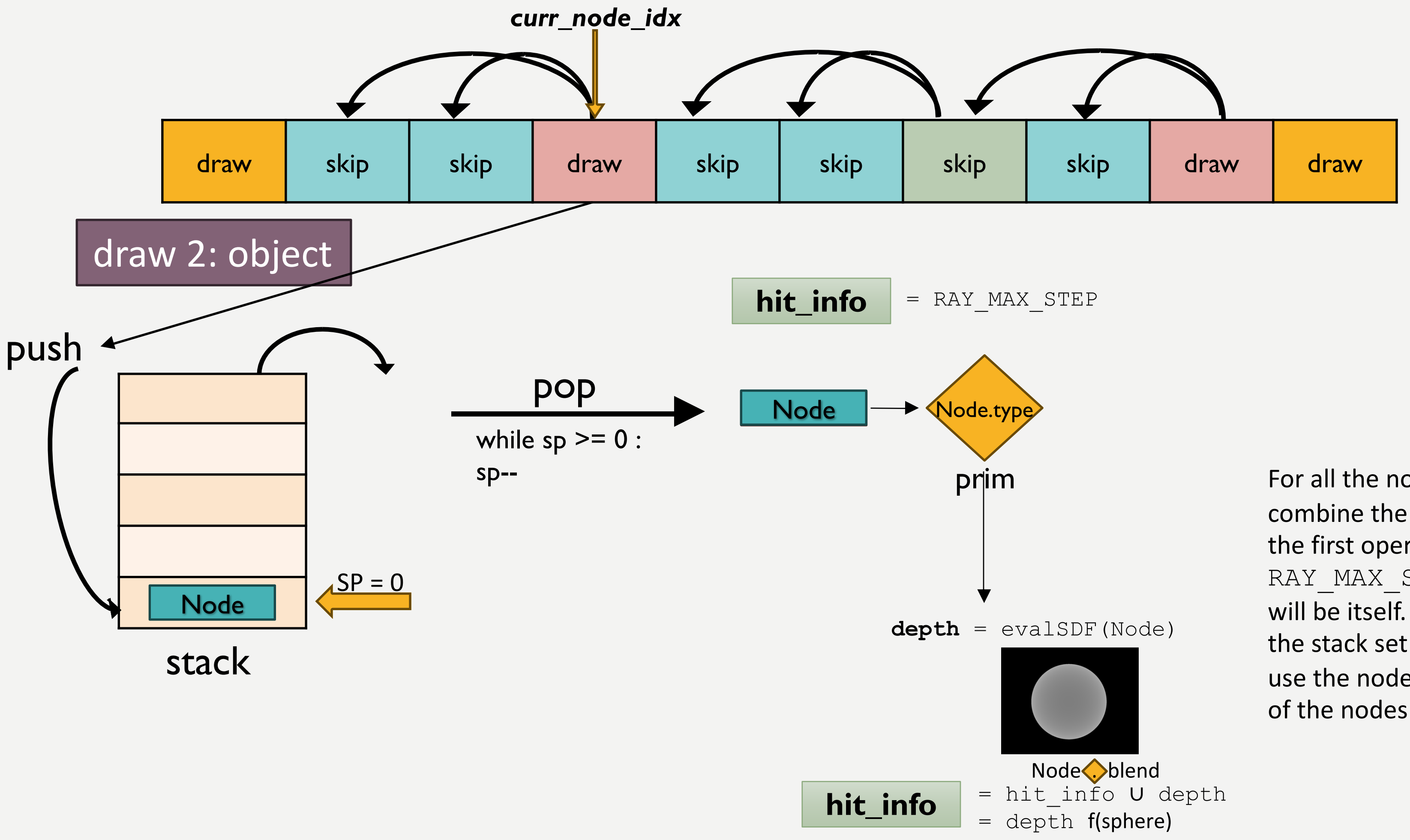
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

GPU drawing complex SDFs

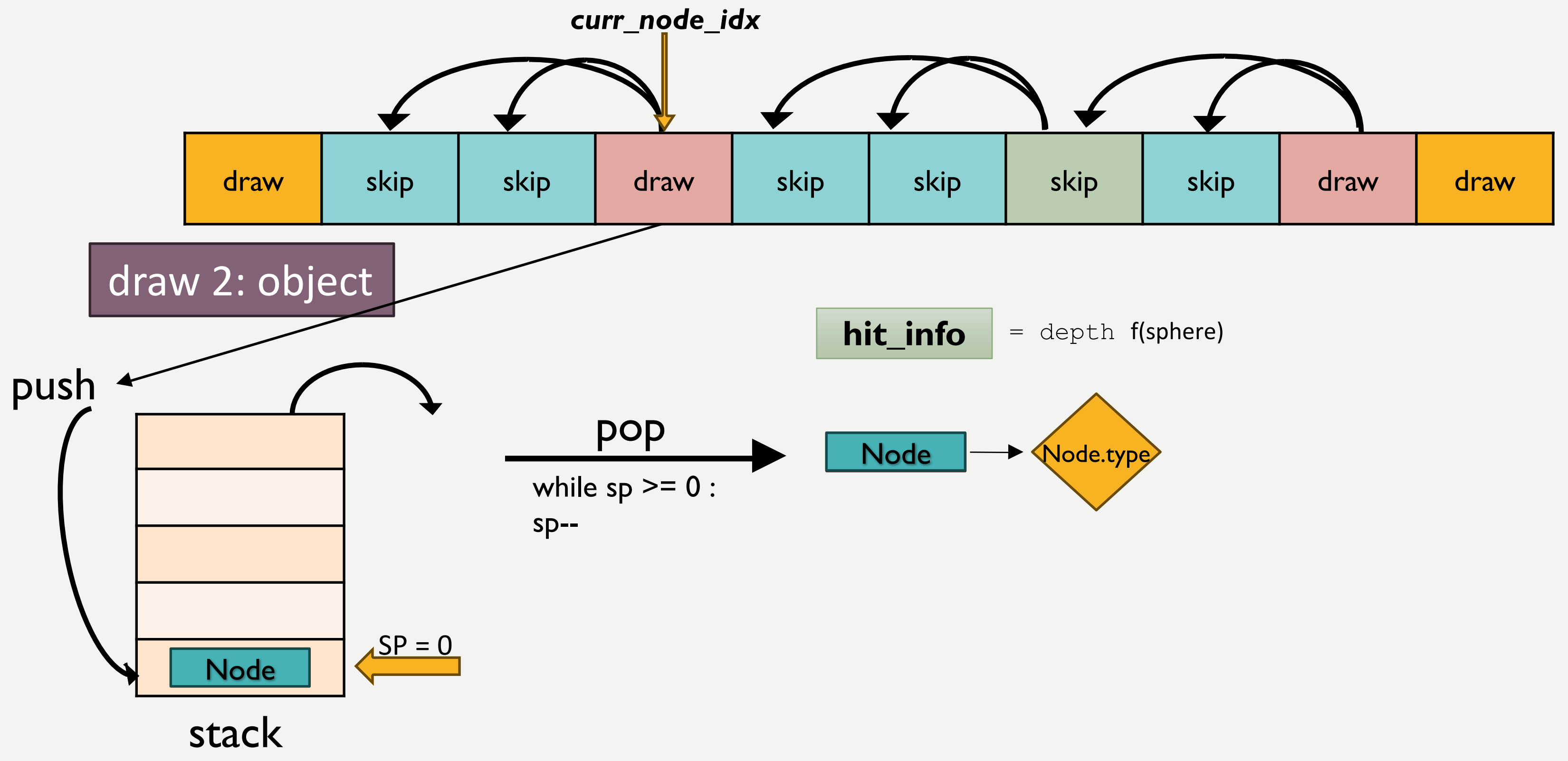


For all the nodes in the stack, as we combine the SDFs node after node, the first operations is a Union (∪) with `RAY_MAX_STEP`, basically the value will be itself. So, for the first node in the stack set the node.blend as ∪ and use the node.blend for the remaining of the nodes.

Scene SDF Calculation

raymarch this function as usual

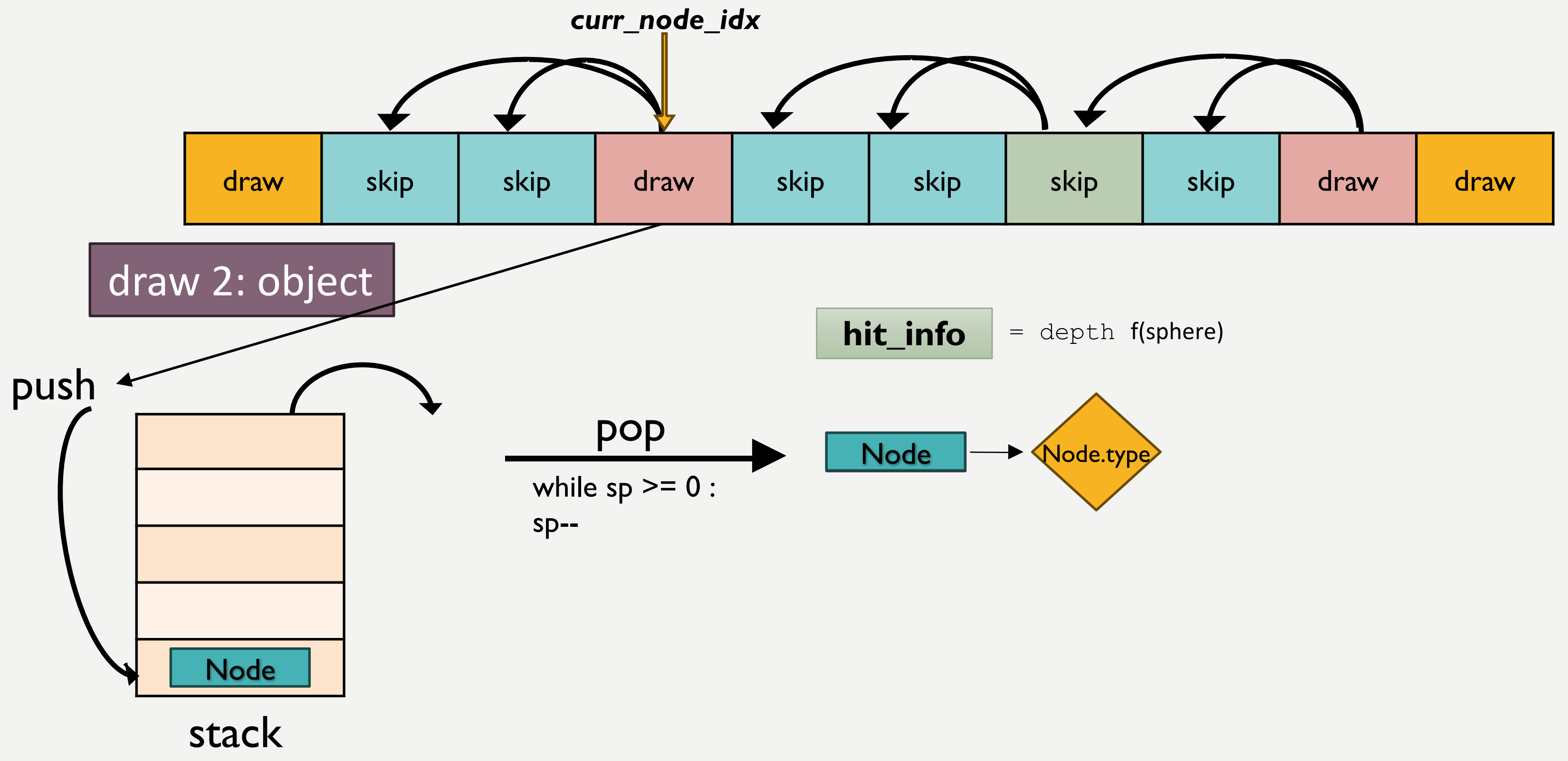
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

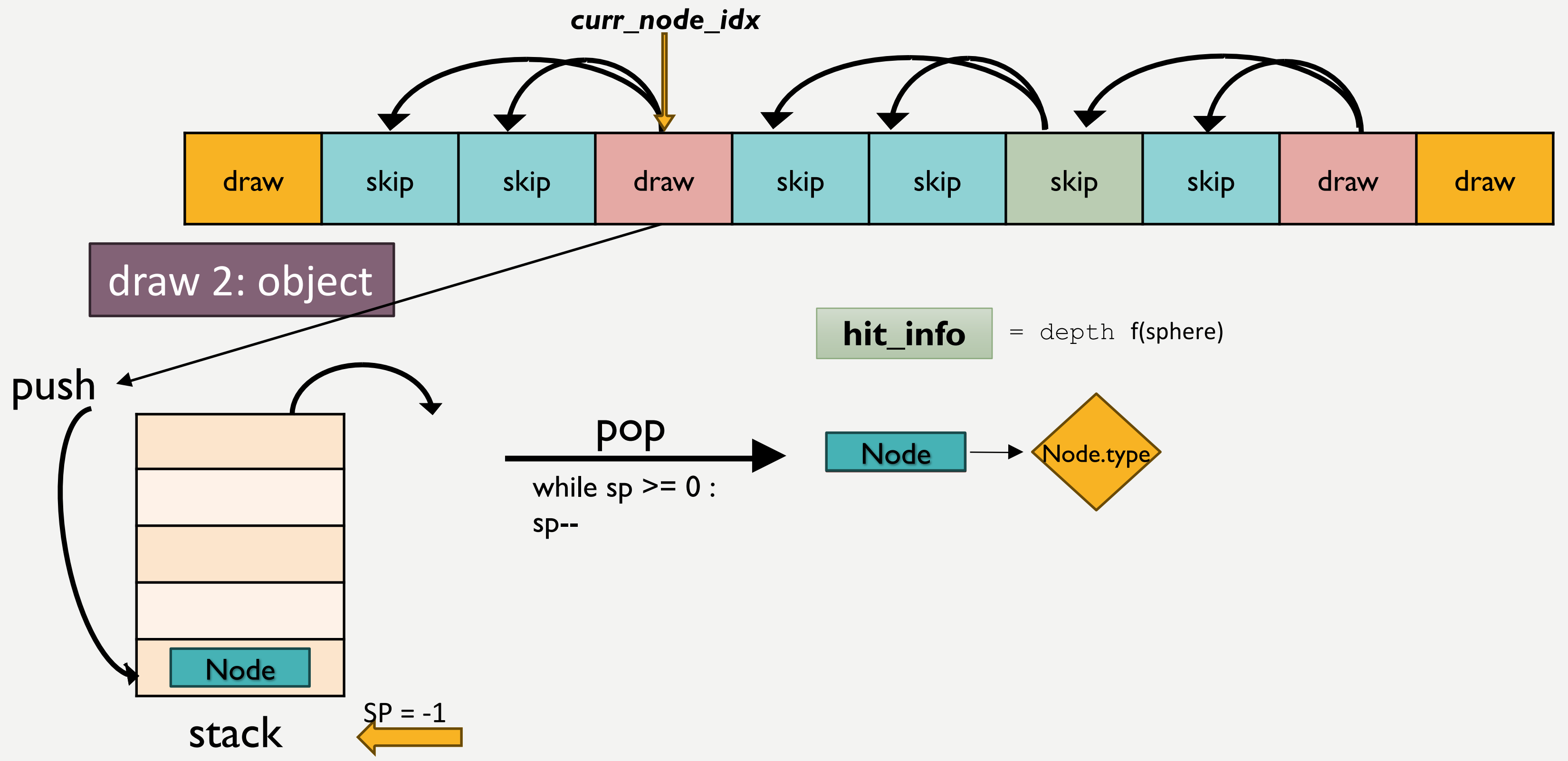
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

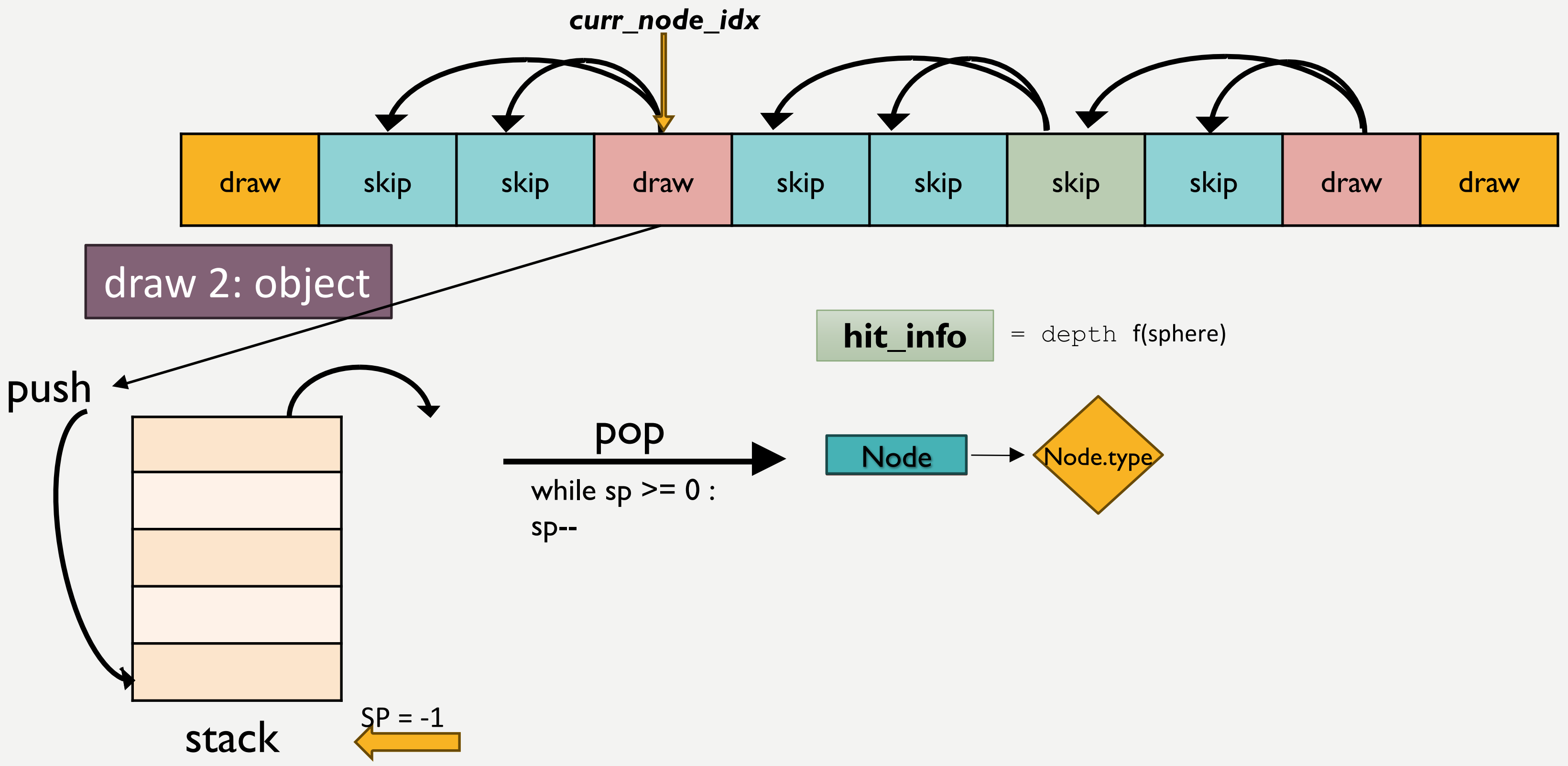
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

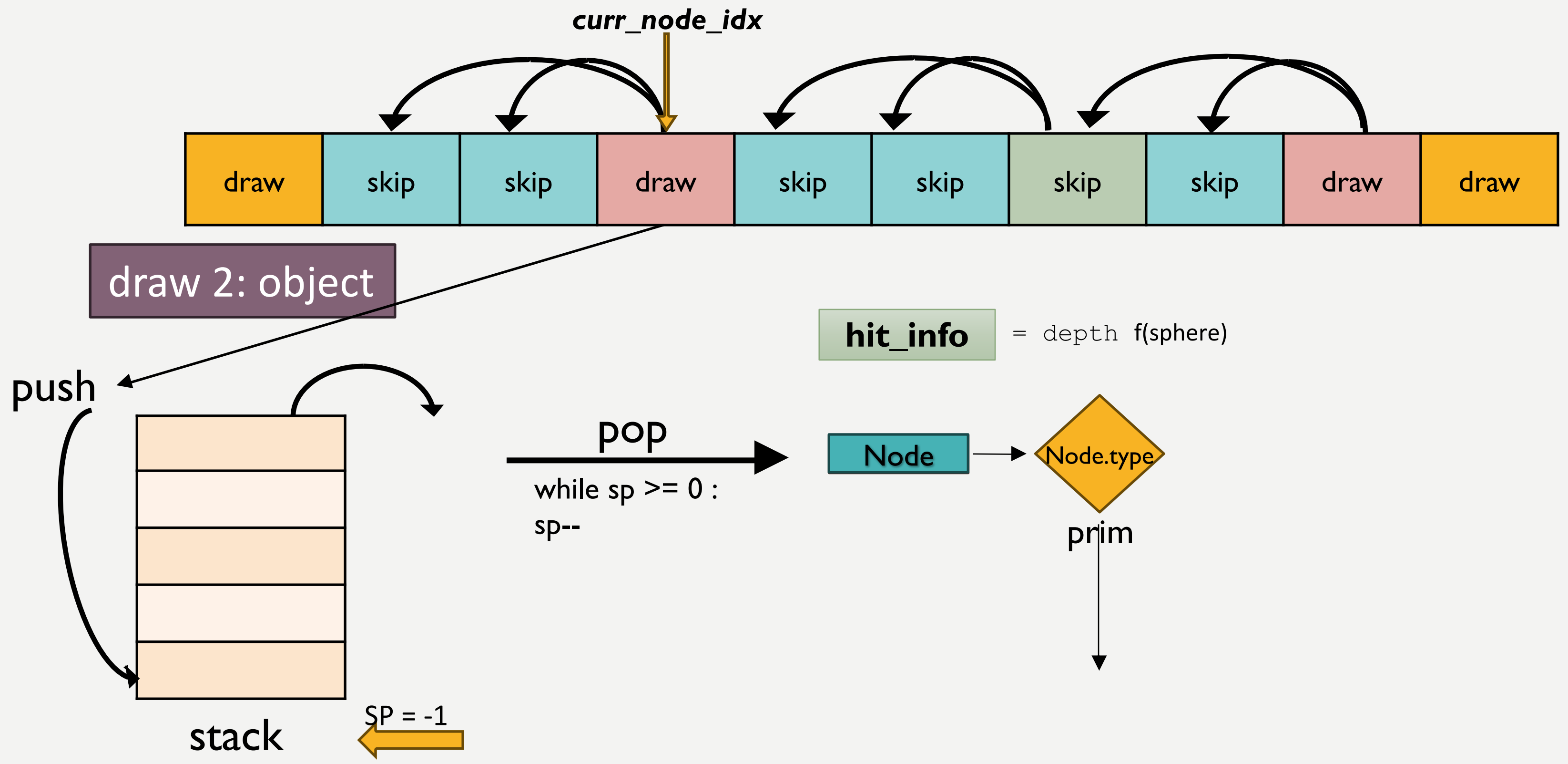
GPU drawing complex SDFs



Scene SDF Calculation

raymarch this function as usual

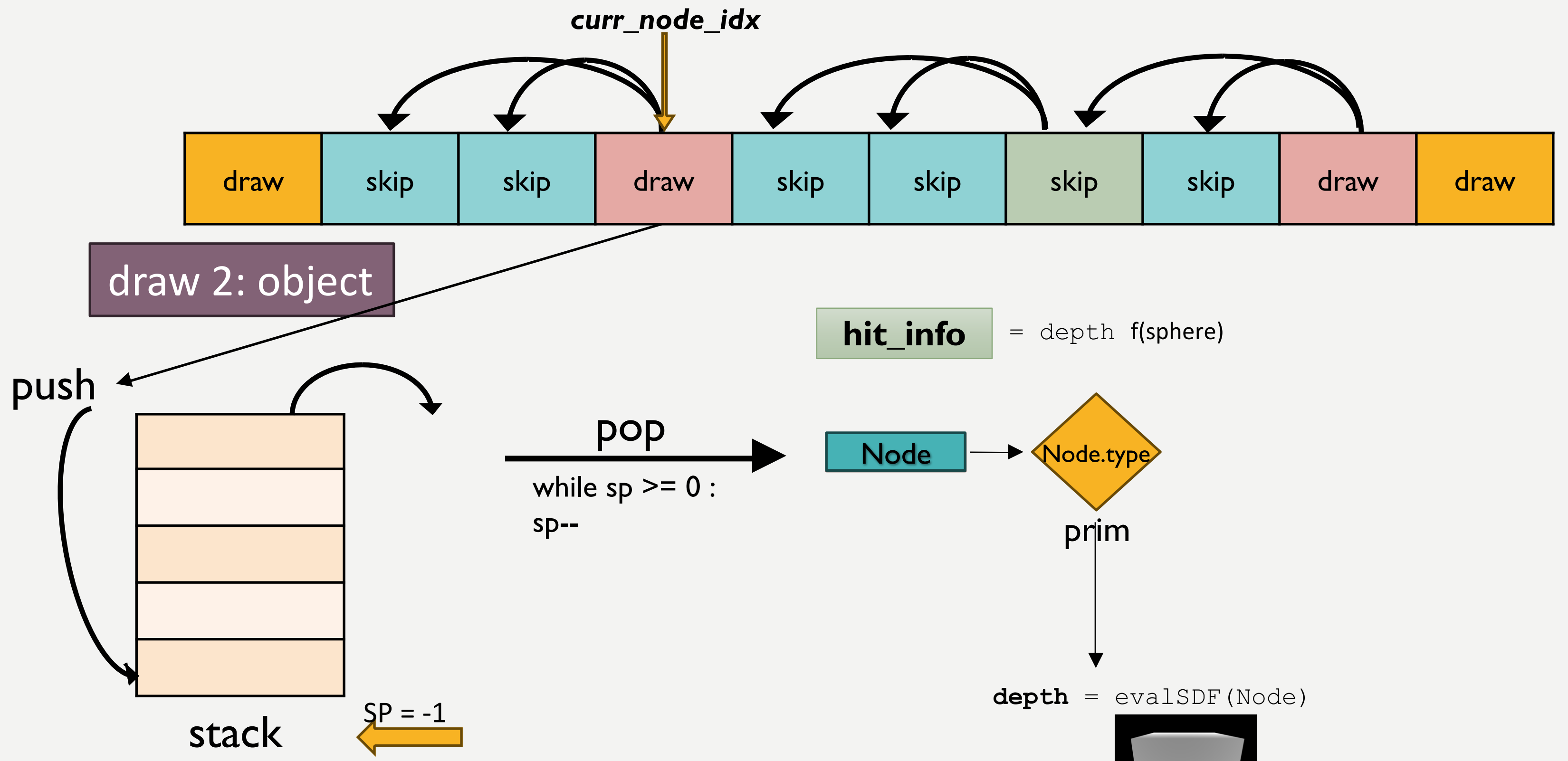
GPU drawing complex SDFs



Scene SDF Calculation

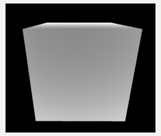
raymarch this function as usual

GPU drawing complex SDFs



hit_info = depth f(sphere)

depth = evalSDF(Node)

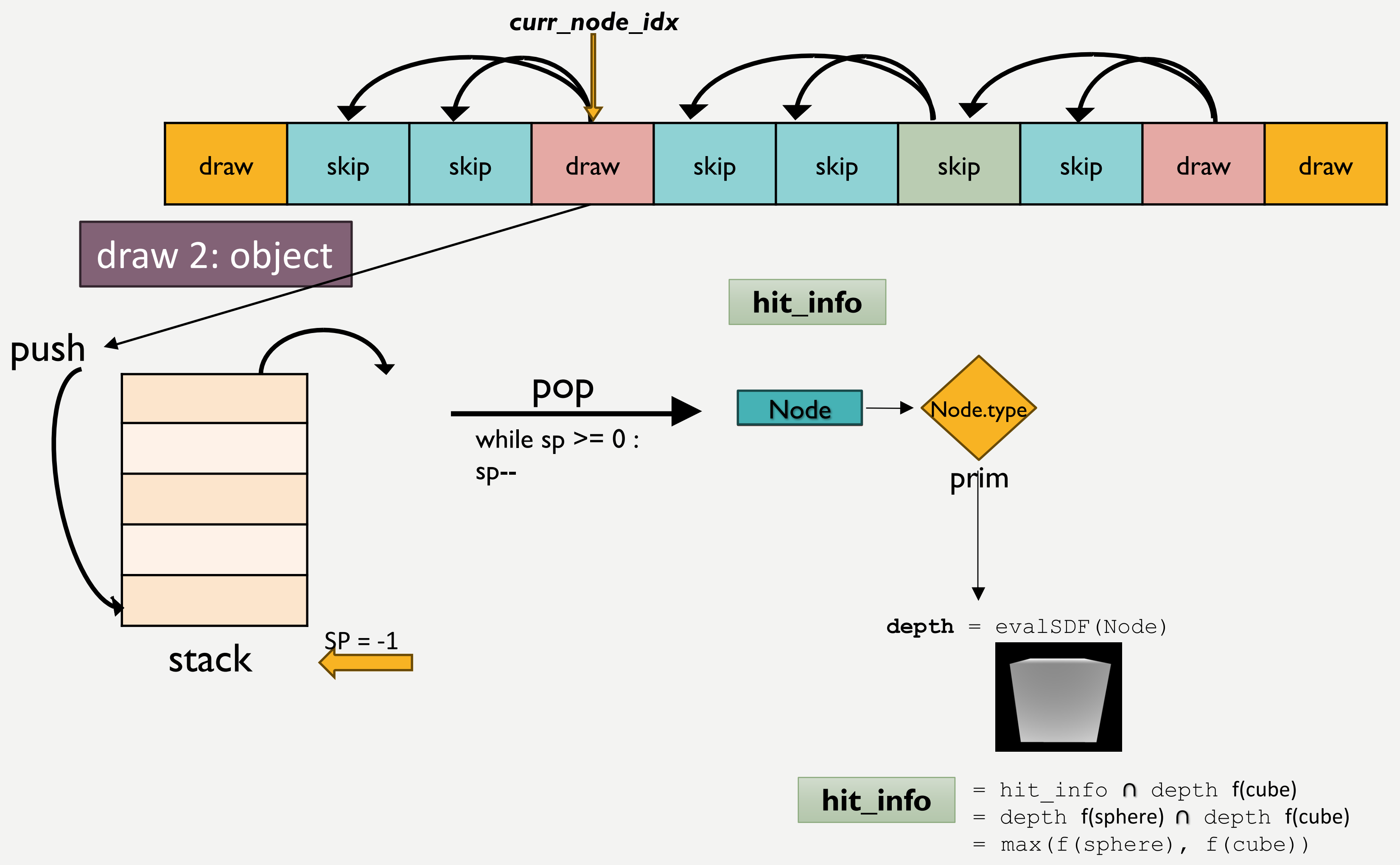


hit_info = hit_info ∩ depth f(cube)
= depth f(sphere) ∩ depth f(cube)
= max(f(sphere), f(cube))

Scene SDF Calculation

raymarch this function as usual

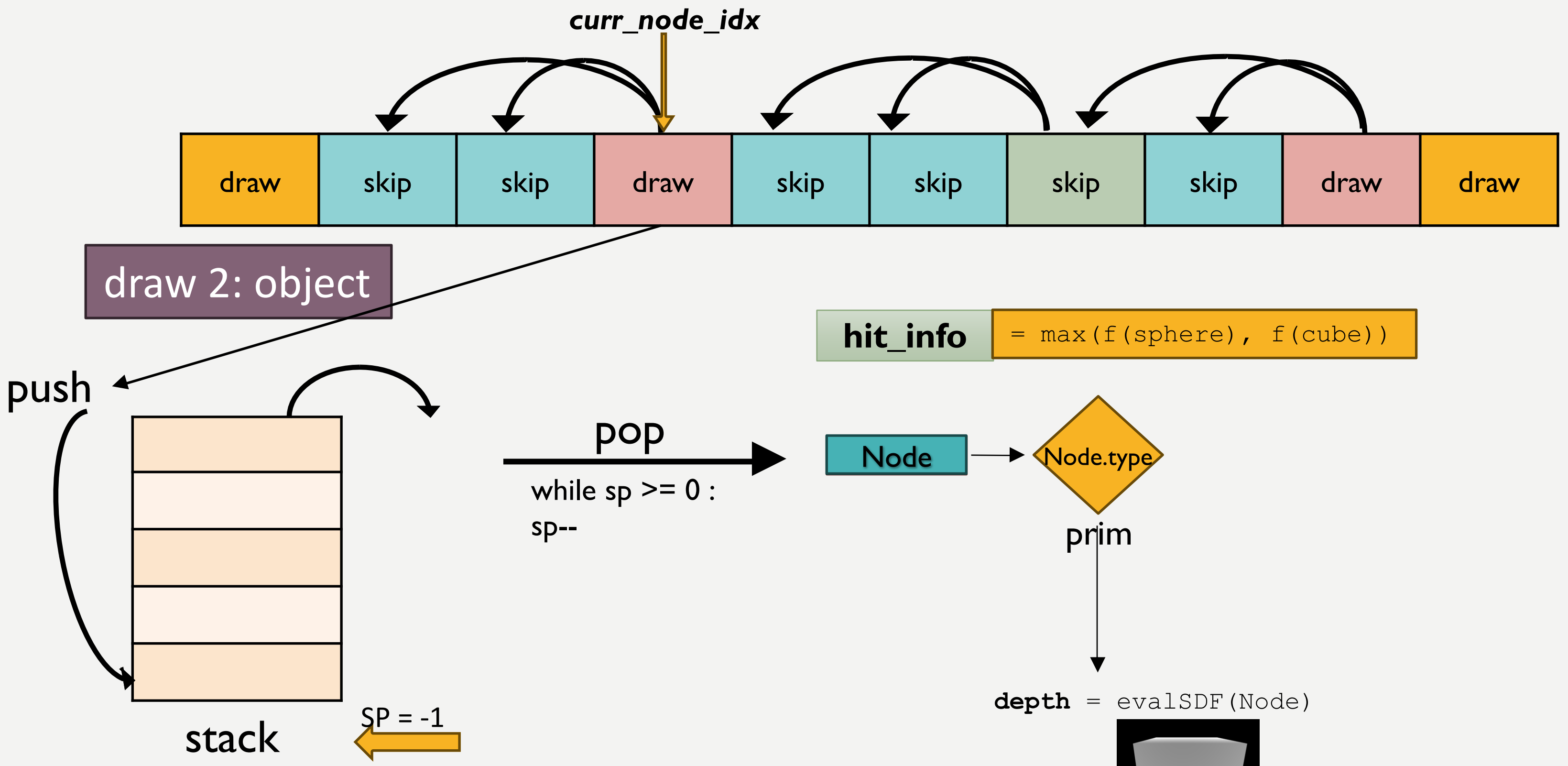
GPU drawing complex SDFs



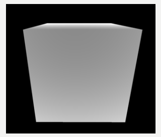
Scene SDF Calculation

raymarch this function as usual

GPU drawing complex SDFs



depth = evalSDF(Node)



hit_info
= hit_info ∩ depth f(cube)
= depth f(sphere) ∩ depth f(cube)
= max(f(sphere), f(cube))

Scene SDF Calculation

raymarch this function as usual

Optimizing Ray Marching shader

- **PS to CS:** Moving the shader from a pixel shader to compute shader will results in a faster shader because a CS doesn’t follow ordered exports and can exit early if some rays don’t hit anything and can perform texture writes in any arbitrary order.
- **Improving Occupancy:**