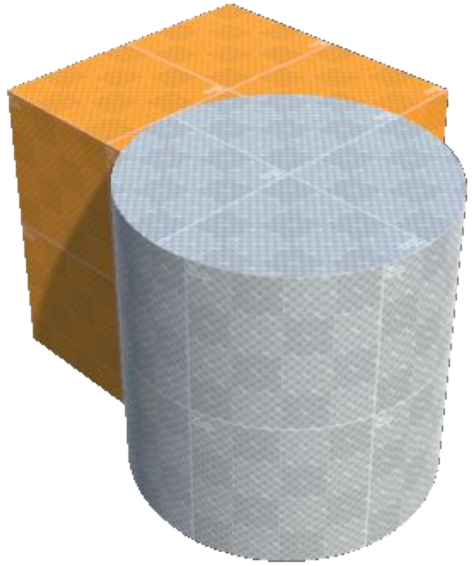
Geometry in milliseconds: Real-time Constructive Solid Geometry

Sander van Rossen
Unity Technologies

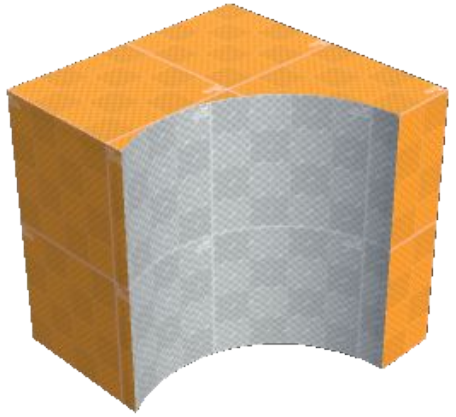
GAME DEVELOPERS CONFERENCE
MARCH 16–20, 2020 | #GDC20

So what *is* Constructive Solid Geometry?

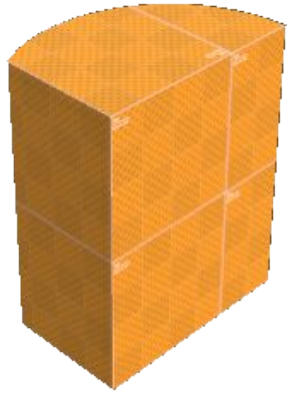
Boolean Operations



Additive

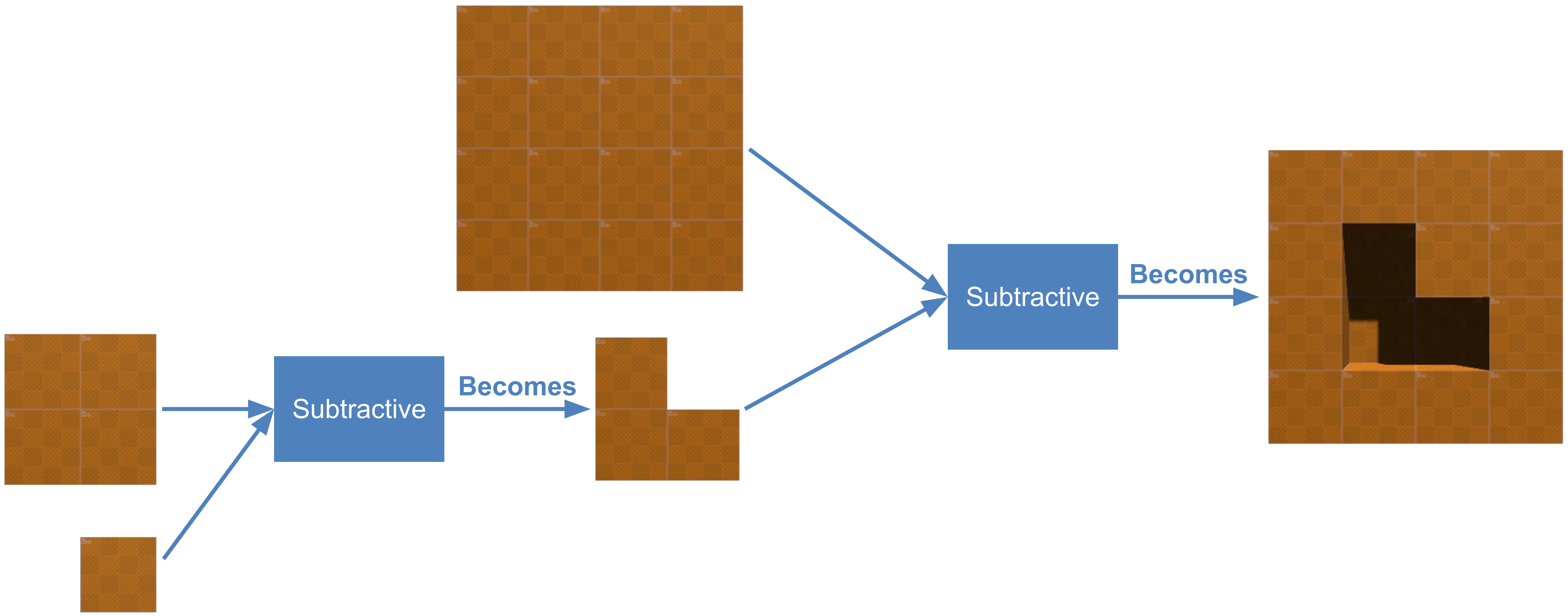


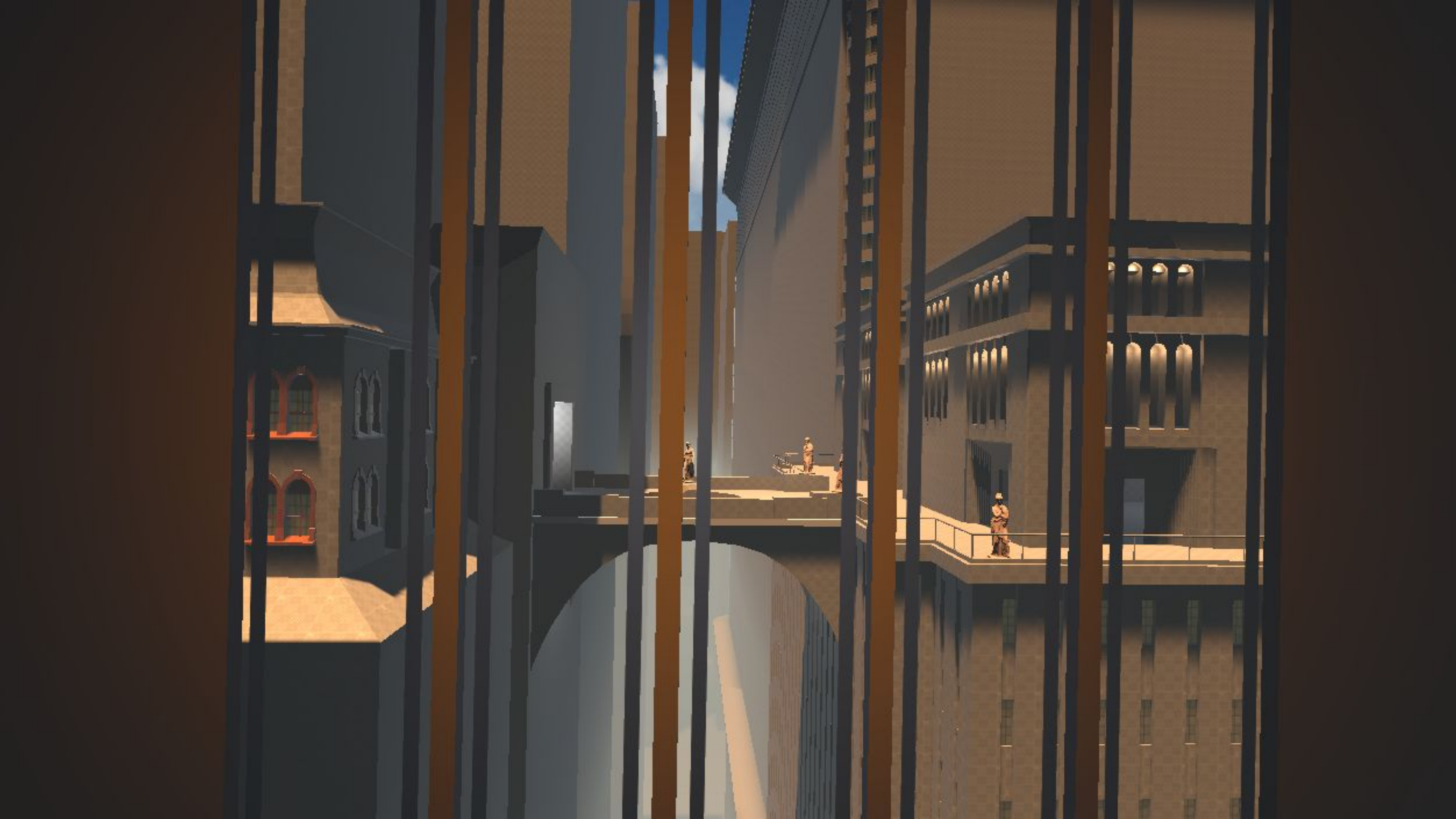
Subtractive



Intersection

CSG Hierarchy





Thank you!

Well,
there’s a little more to it...

Overview

1. History of CSG

The algorithm

2. Iterative updates
3. Intersections
4. Mesh Generation
5. Polygon categories, Routing & Operation tables
6. Putting it all together

First some history

History

- Originated outside of the game industry
  - Used in the CAD industry
- Long history in the game industry
  - Quake/iD tech engines / iD Software
    - Many build on top are still in use today (mostly completely rewritten)
      - Source engine 1-2 / Valve
  - Unreal engine 1-4 / Epic Games
  - Torque, Roblox, and many more

History

- Games with CSG level editors often spawned mod communities
- Some mods turned into full games
  - Counter strike
  - Team Fortress
  - Portal
  - Black Mesa
  - The Stanley Parable
- Many professional level designers started out as modders

History


Unreal GOLD
Unreal TOURNAME
PC DVD
The Orange Box
TEAM FORTRESS 2
GEARS OF WAR 4
TITANFALL 2
STAR WARS JEDI FALLEN ORDER
HALO REACH
THIEF II THE METAL AGE
CALL OF DUTY MODERN WARFARE
ROBLOX
QUAKE
PORTAL 2
APEX LEGENDS
Wolfenstein YOUNGBLOOD
WARFRAME
HALF-LIFE 2
IN THE VALLEY OF GODS
A NEW VIDEO GAME FROM THE MAKERS OF FIREWATCH
Global Offensive 2012
CS GO
BLACK MESA
DOOM 3
system shock 2

History

- Early implementations used Binary Space Partitions (BSP)
  - Scales poorly with number of polygons
    - Unusable beyond a relatively small number
- The tooling build around CSG hasn’t evolved much
  - Unreal, for example, still uses the BSP code Tim Sweeney wrote decades ago

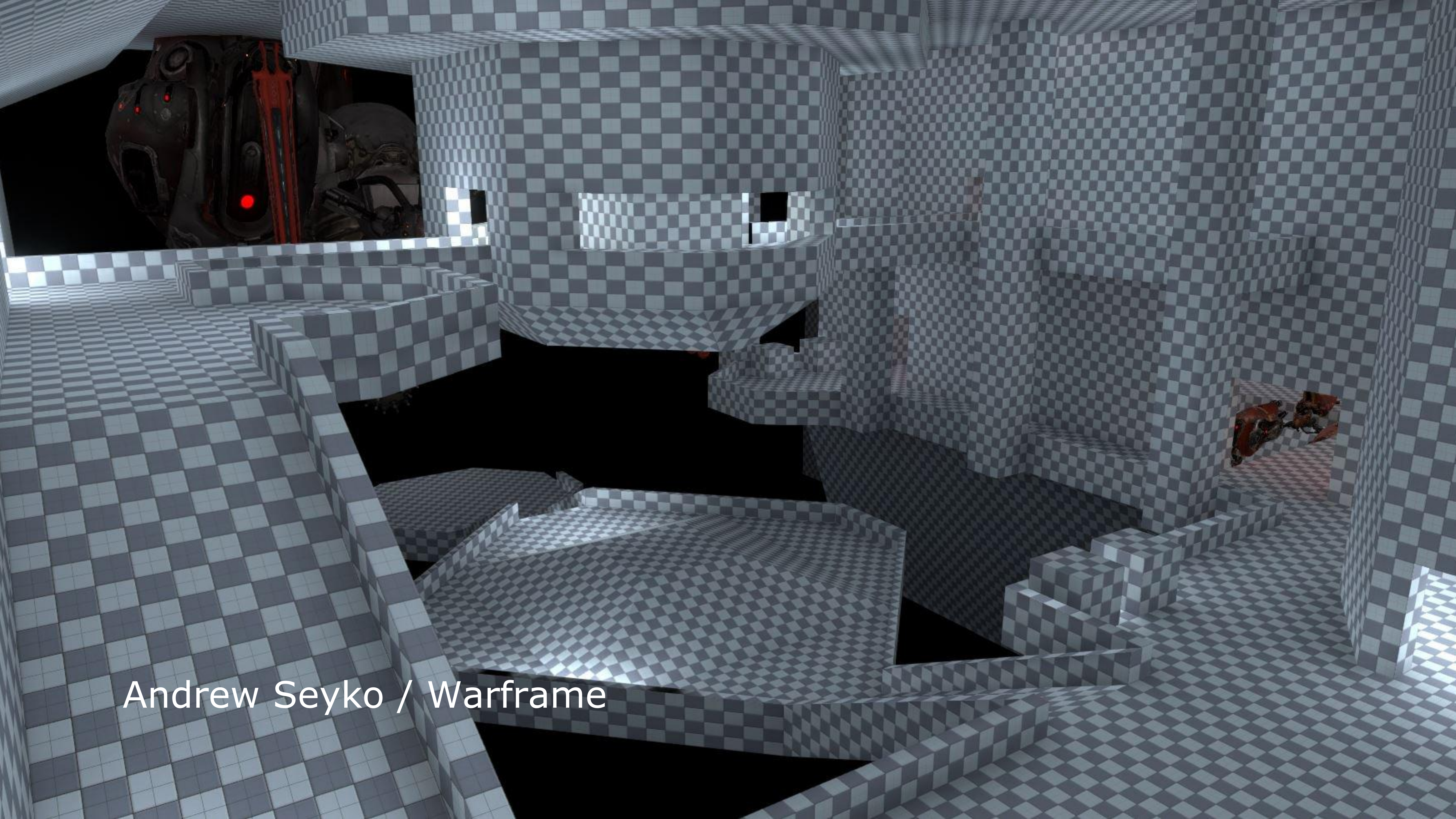
But… why?

Workflow

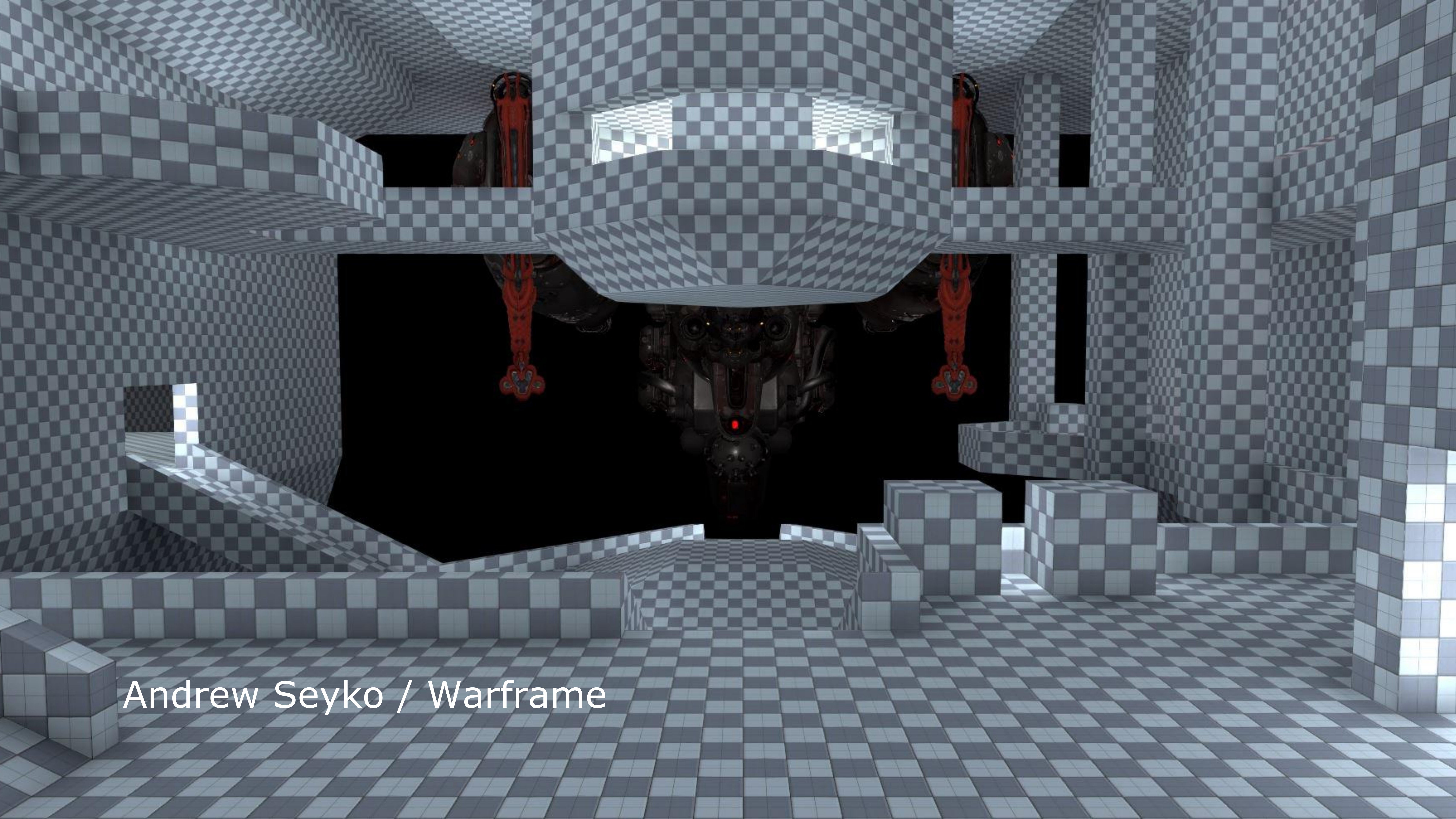
- Fast & non-destructive iteration
  - Brushes can easily be moved around, replaced, hidden/shown
    - Your level geometry will automatically get adjusted
  - Fast to quickly mock/block out levels, test gameplay
  - Easy to try out different game layouts quickly
- Easy to learn / very intuitive / Allows for playful exploration
- Mostly used to design larger outline and flow of levels
  - Complemented with modeled props
  - Sections replaced with pieces of modeled geometry

#blocktober


Michael Barclay @MotleyGrue · Oct 1, 2017
What's up level designers. Level blockouts are art. #blocktober should be a thing. #leveldesign #gamedev #gamedesign #inktober #animtober
28
445
2K



Andrew Seyko / Warframe


Andrew Seyko / Warframe


Andrew Seyko / Warframe

Alex Graner / Apex Legends

Alex Graner / Apex Legends


Alex Graner / Apex Legends

Alex Graner / Apex Legends


Alex Graner / Apex Legends

Workflow

- CSG forces a focus on the large first, details later
  - You can fine tune your game with simpler geometry
    - Before you spend resources on making it pretty
- CSG creates *solid* geometry without gaps, ideal for physics
  - Easy to make invisible infinitely thin gaps in a 3D modeling tool
    - Unlikely for this to happen with CSG
  - Not something you want to worry about during design

Workflow

- CSG is well suited for procedurally generated geometry
  - All geometry created by CSG is physically plausible
  - Can very easily layer geometry by addition & subtraction
    - Allows the user to mix procedural geometry with hand created geometry seamlessly

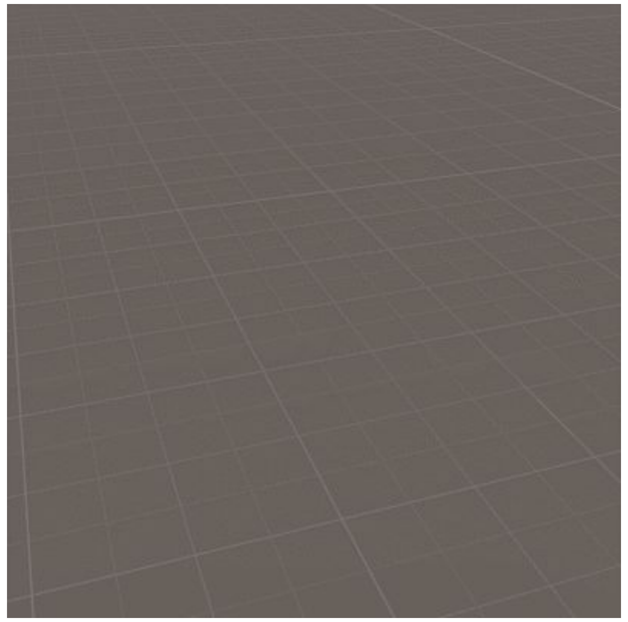
Workflow

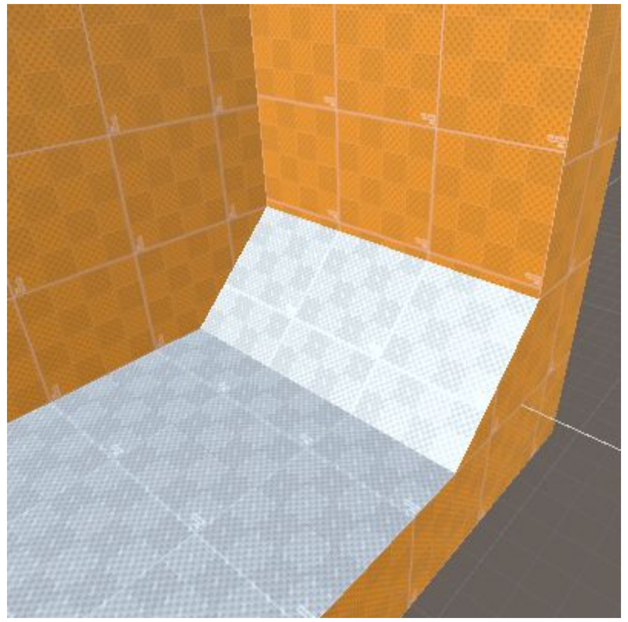
- Level design is not 3D modeling
    - Level designers and 3d artists are two different competencies
    - Level design is not just about what the geometry looks like
    - You always need the best tool for the job
        - You *can* mow your lawn with a scissor, *but why would you?*

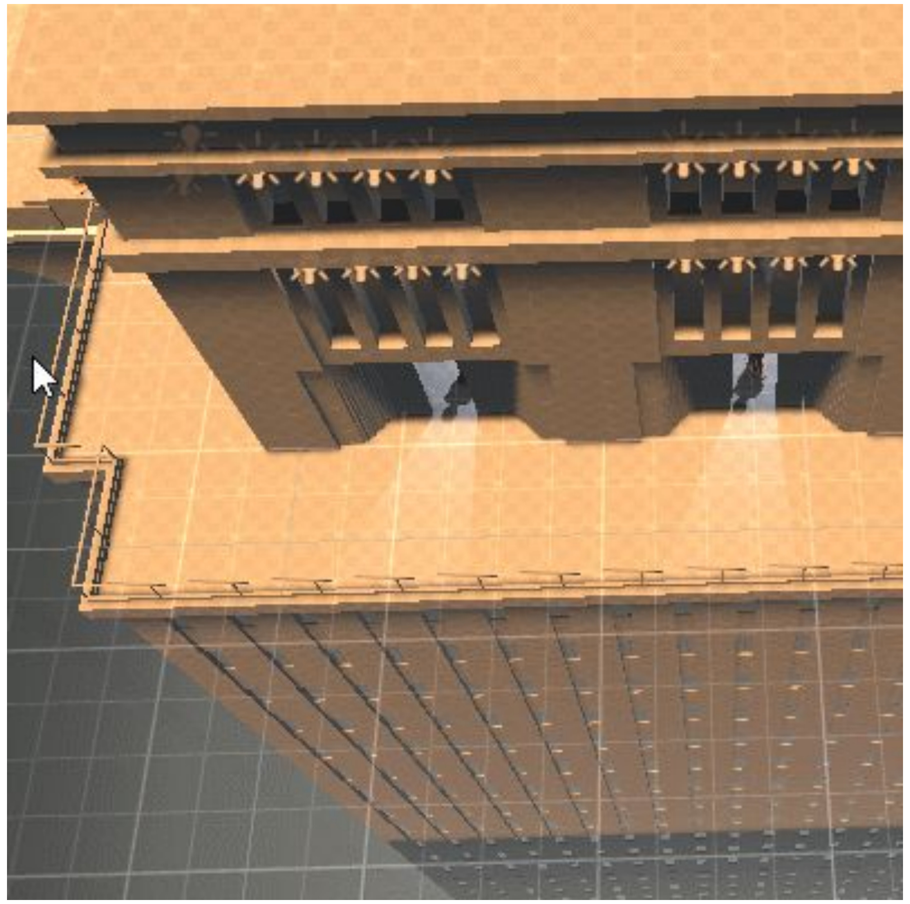
Perception

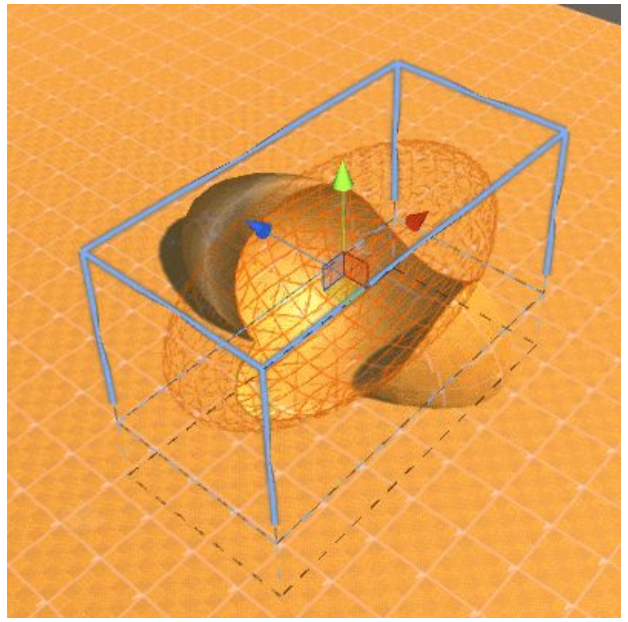
- Sadly, artists often equate CSG with BSP and old tools
    - Most common given reason not to use CSG is "it's slow and blocky"
- Yet, it doesn't have to be this way

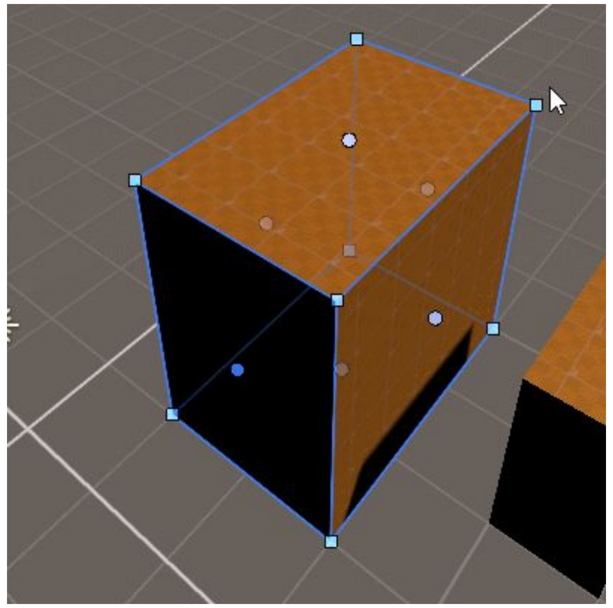
Modern CSG tools

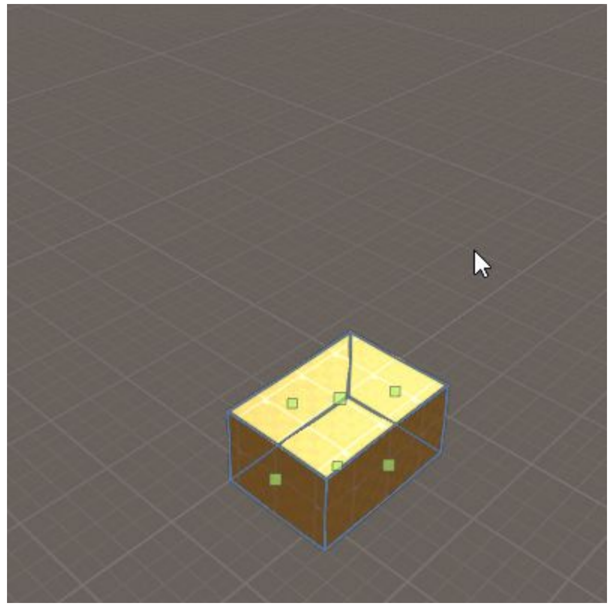


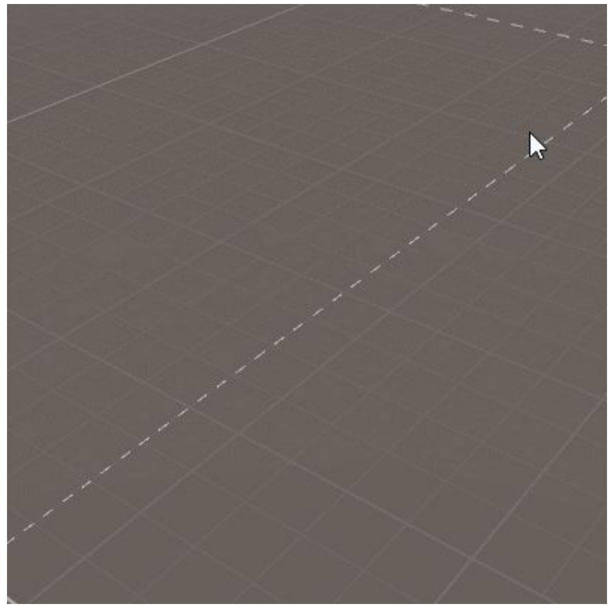


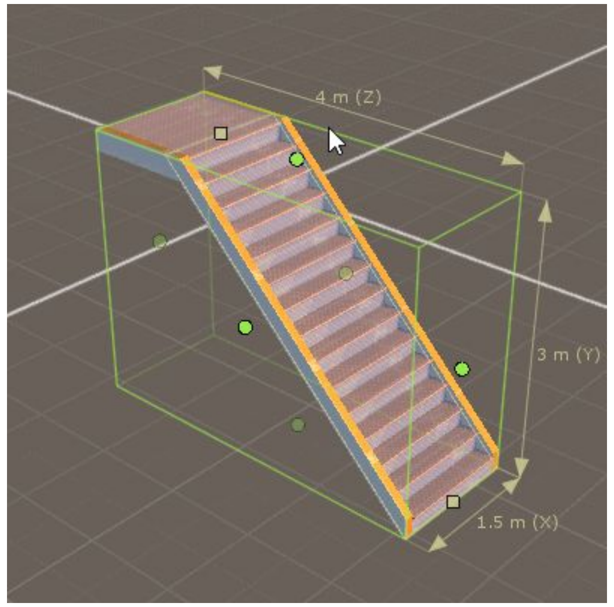


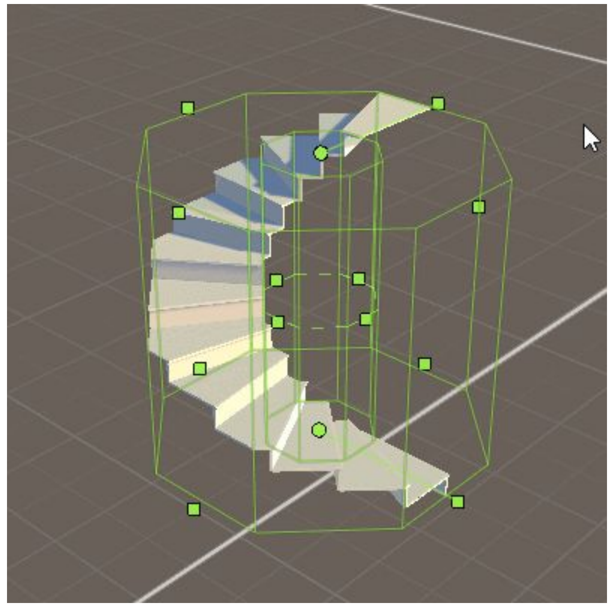












Do I have your attention?

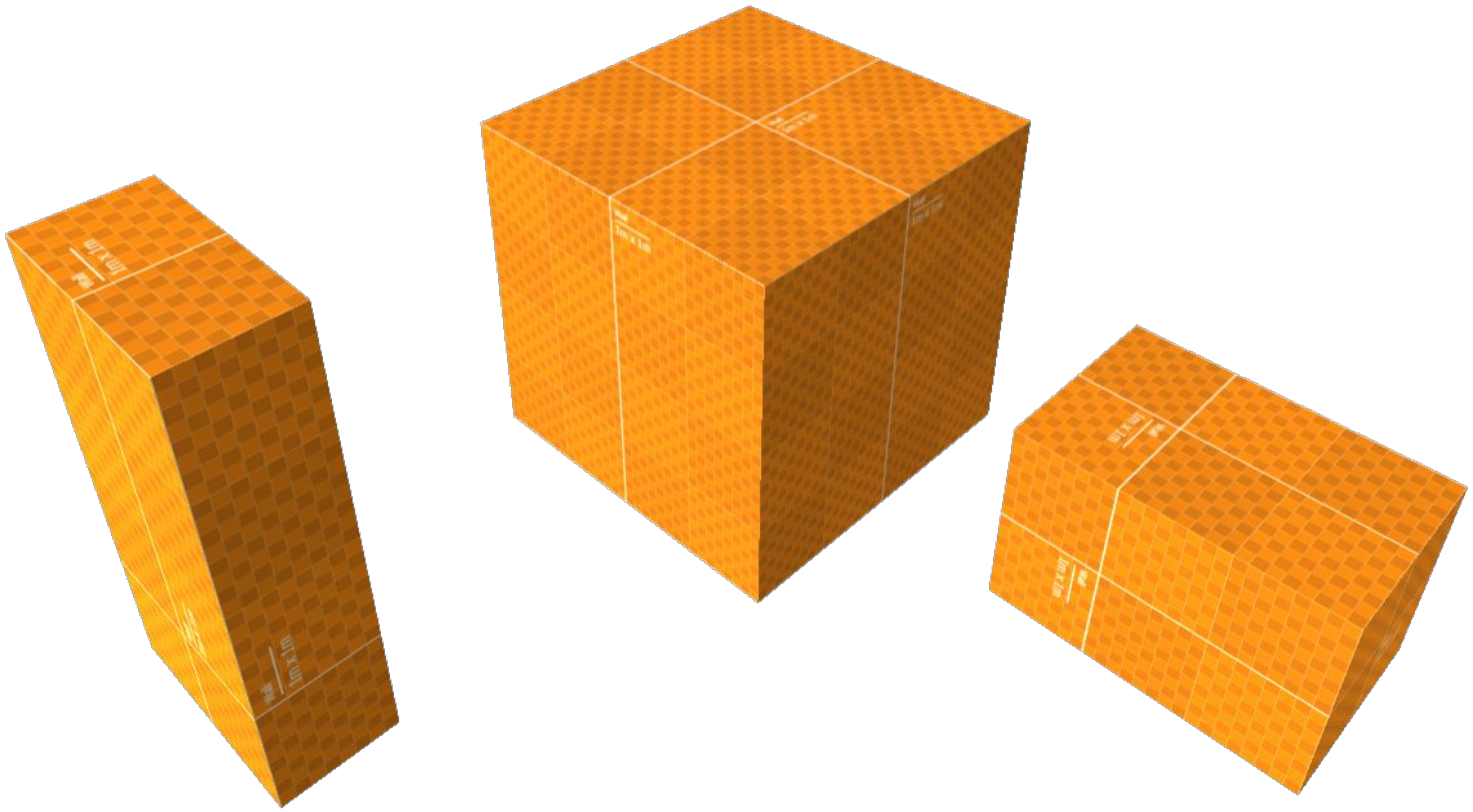
How?

Overview

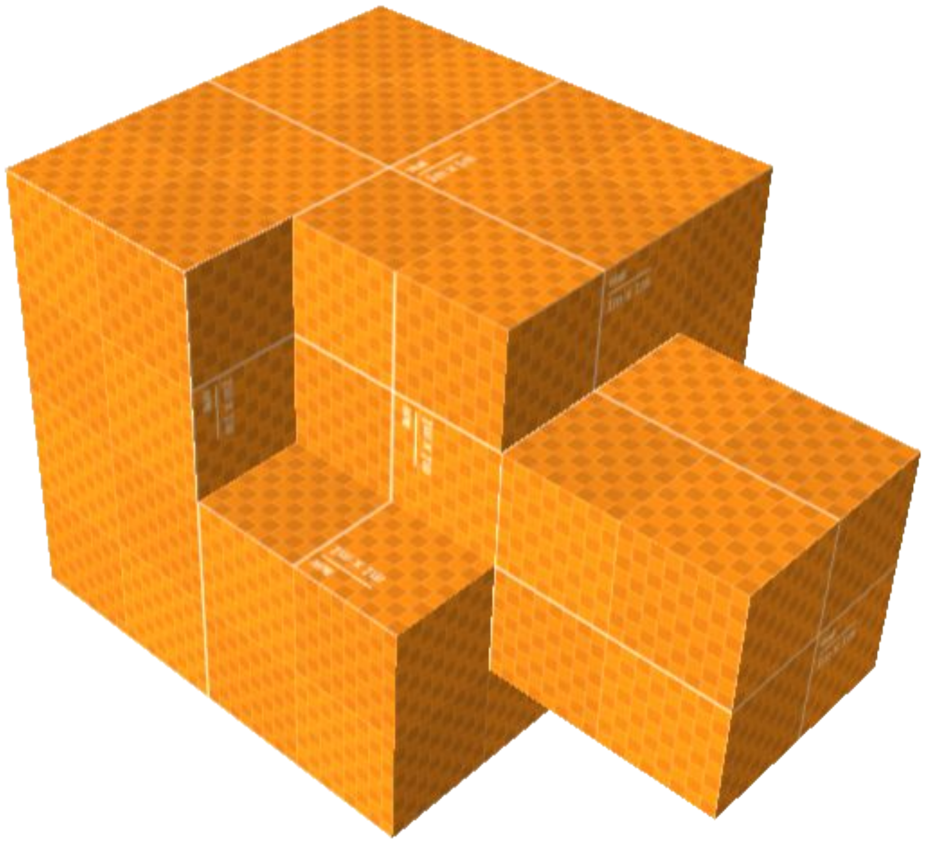
1. History of CSG

The algorithm

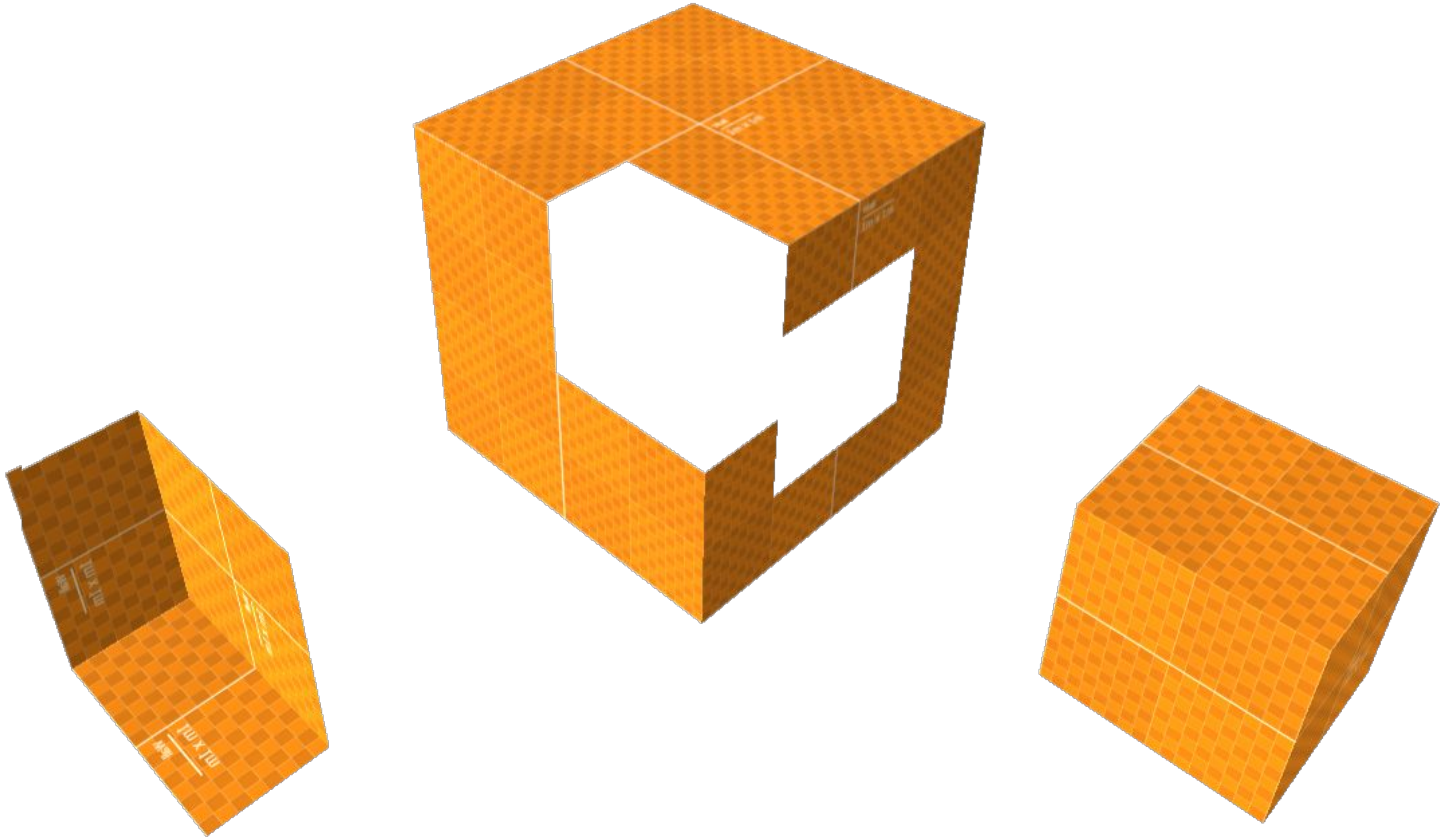
2. **Iterative updates**
3. Intersections
4. Mesh Generation
5. Polygon categories, Routing & Operation tables
6. Putting it all together



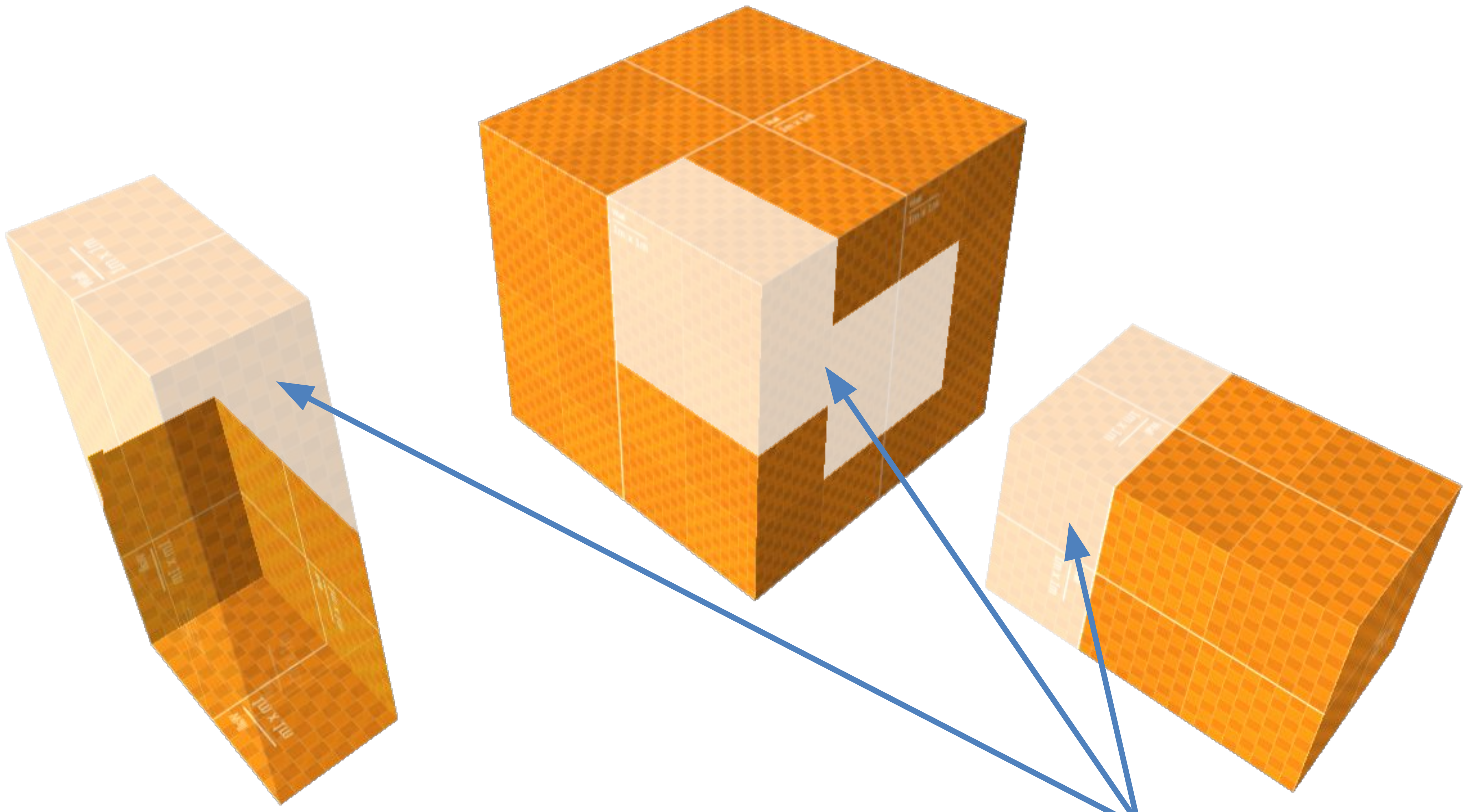
Suppose we perform CSG on some brushes



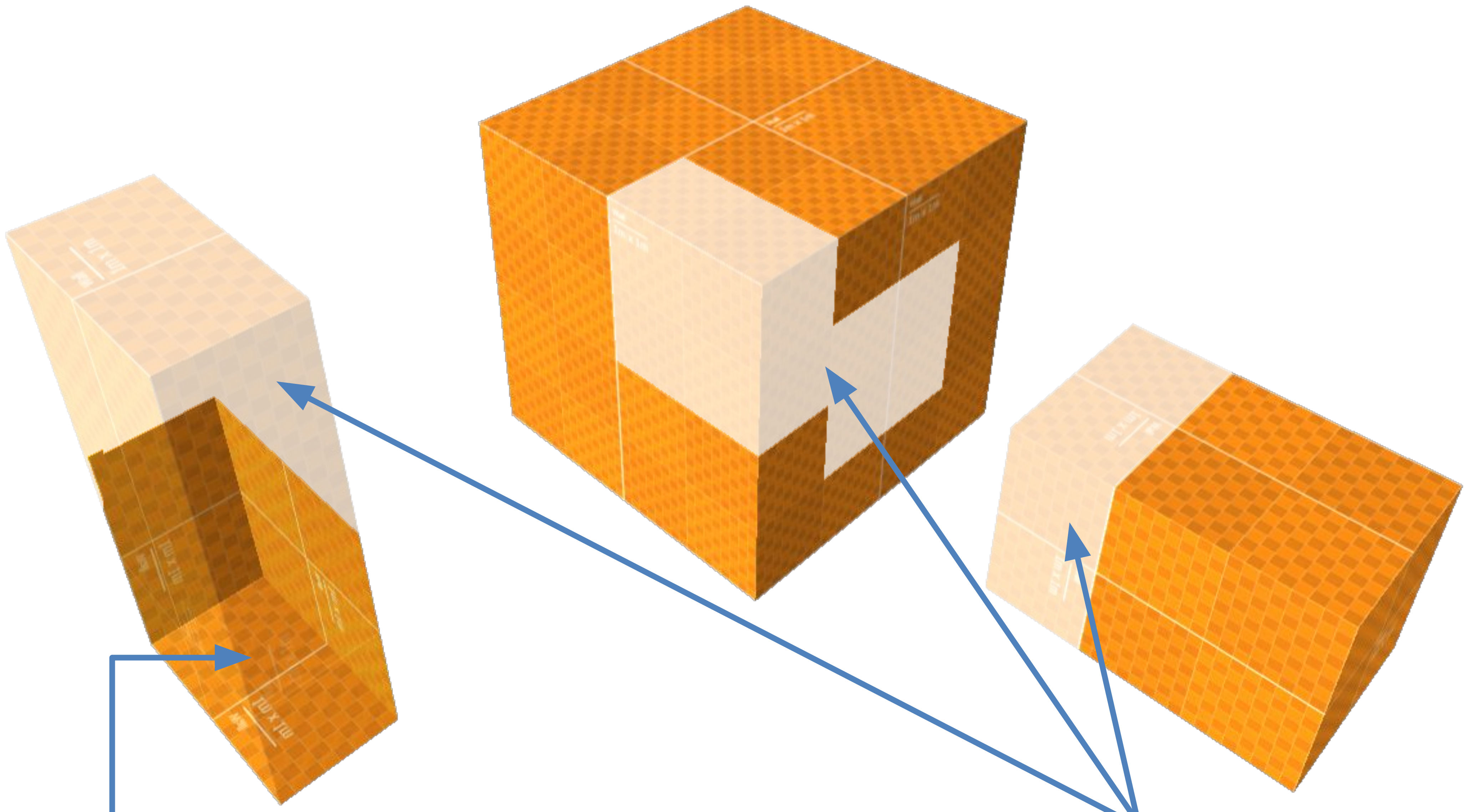
And we create a shape with those brushes,
using a subtractive and an additive operation



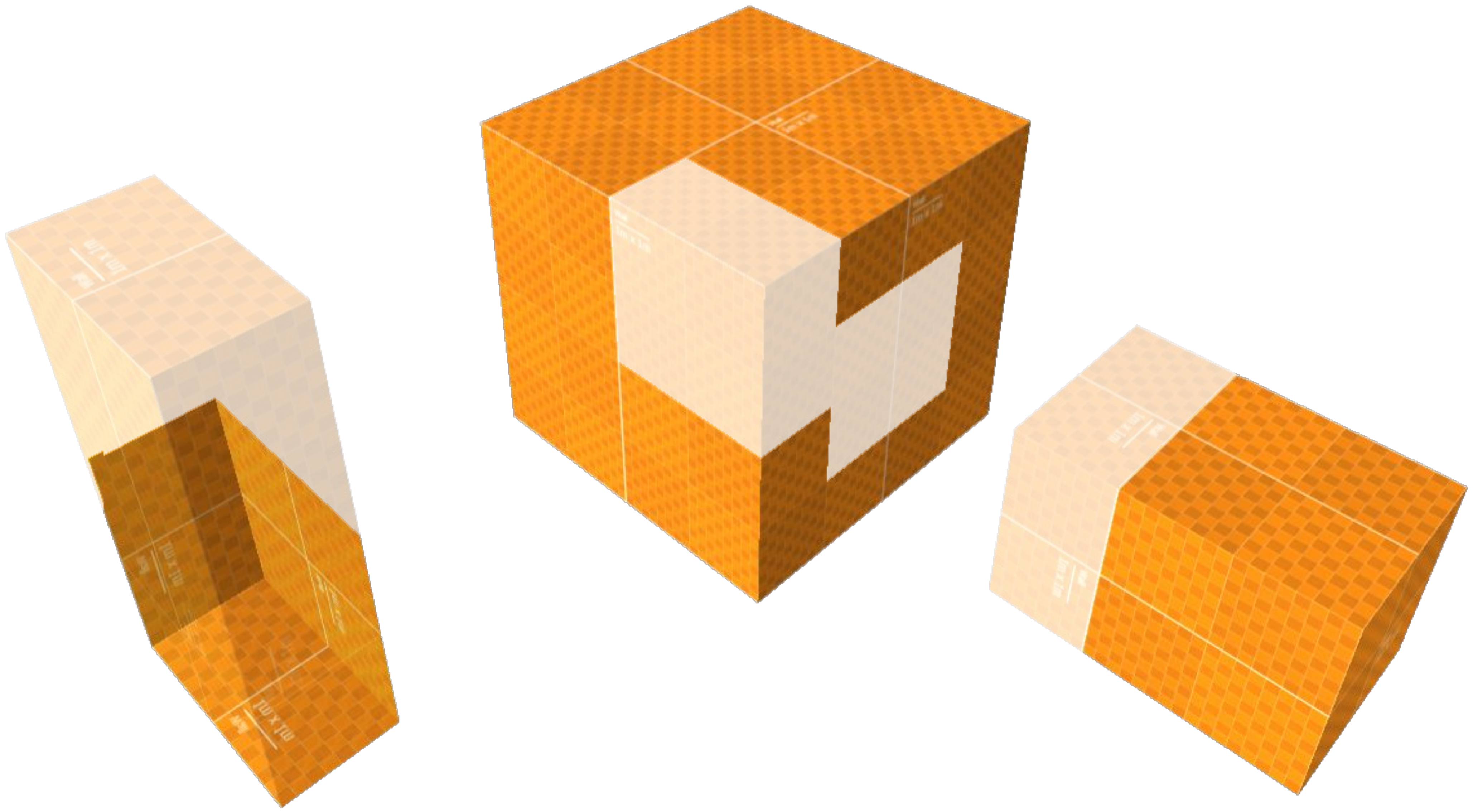
If we look at the contributions from
each individual brush on the final shape



We can see that we only need to **remove**
or **flip the orientation** of polygon pieces ..



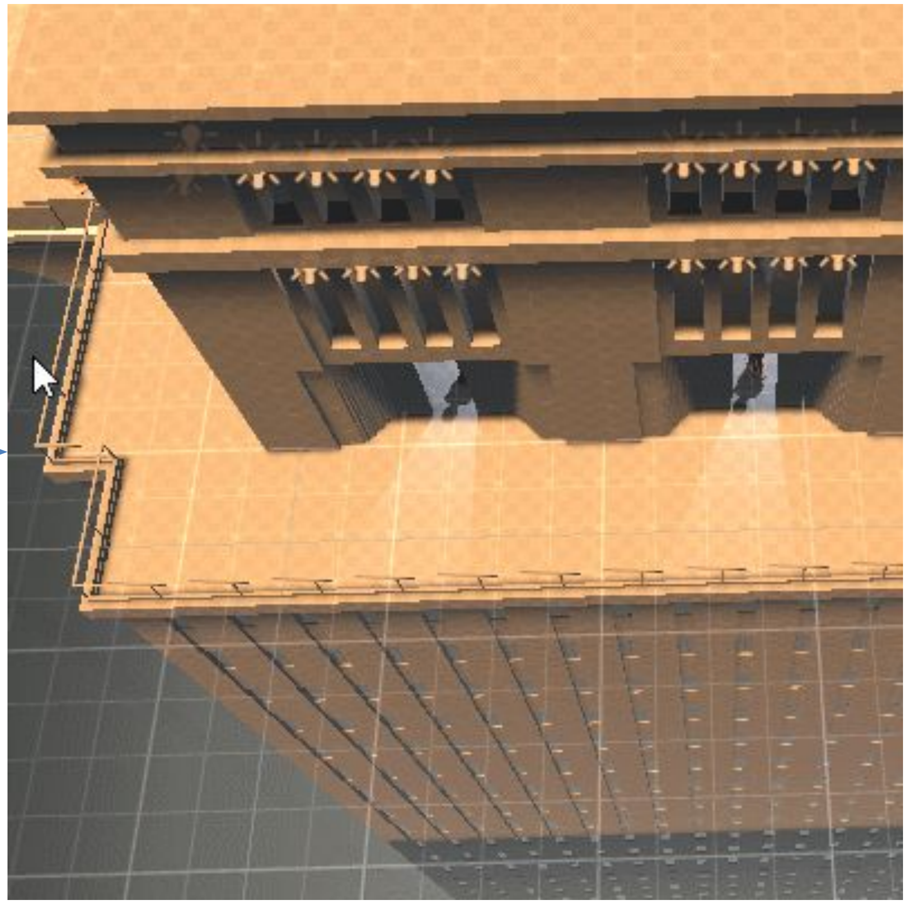
We can see that we only need to **remove**
or **flip the orientation** of polygon pieces ..



We can see that we only need to **remove** or **flip the orientation** of polygon pieces ..
By finding those pieces, we can perform CSG per brush

Performing CSG per brush

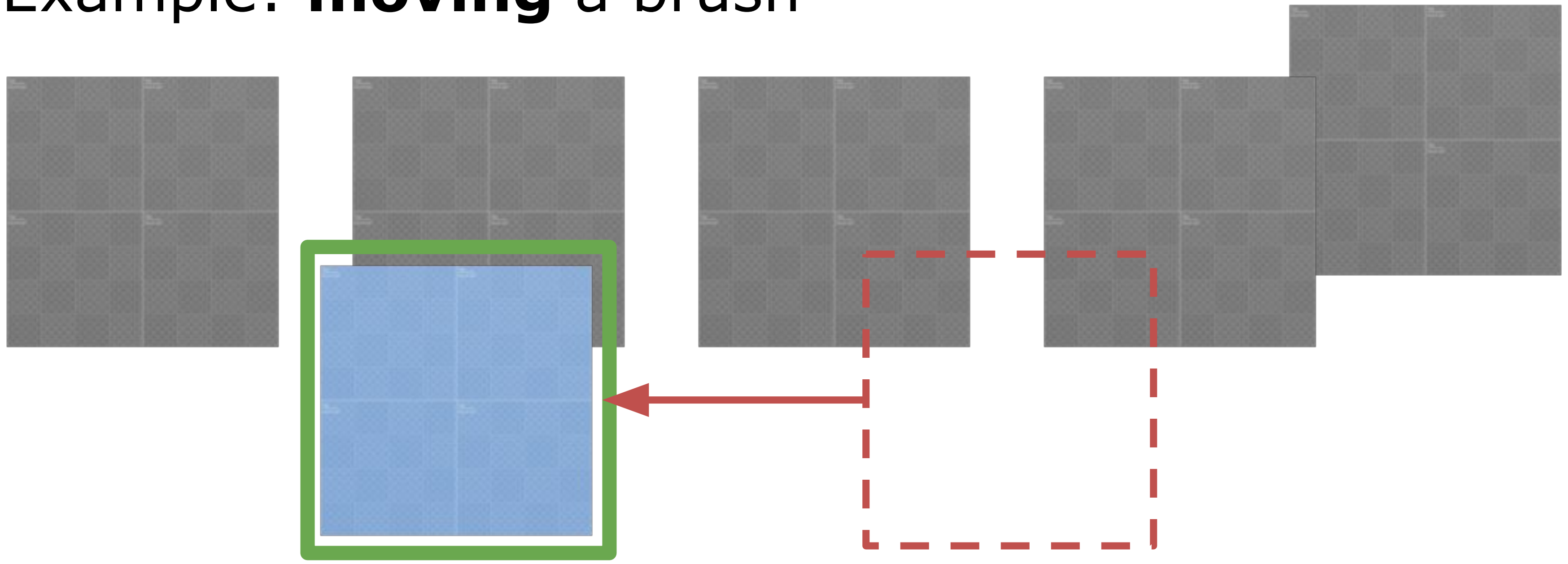
- Allows for **iterative** updates
  - **Makes this possible**



Performing CSG per brush

- Allows for **iterative** updates
    - Only need to **update a brush when its modified**
        - And all brushes that touched/touch it

Example: **moving** a brush

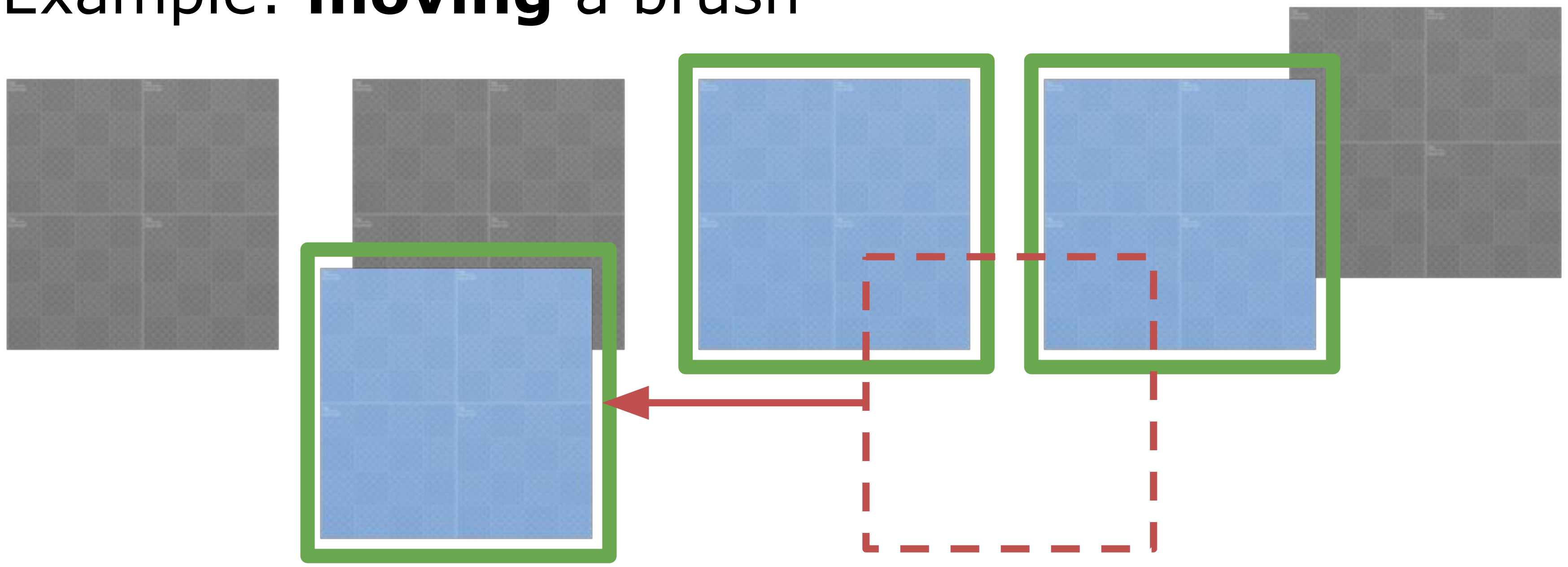


The brush itself is marked dirty

Performing CSG per brush

- Allows for **iterative** updates
  - Only need to **update a brush when its modified**
    - And all brushes that **touched**/touch it

Example: **moving** a brush

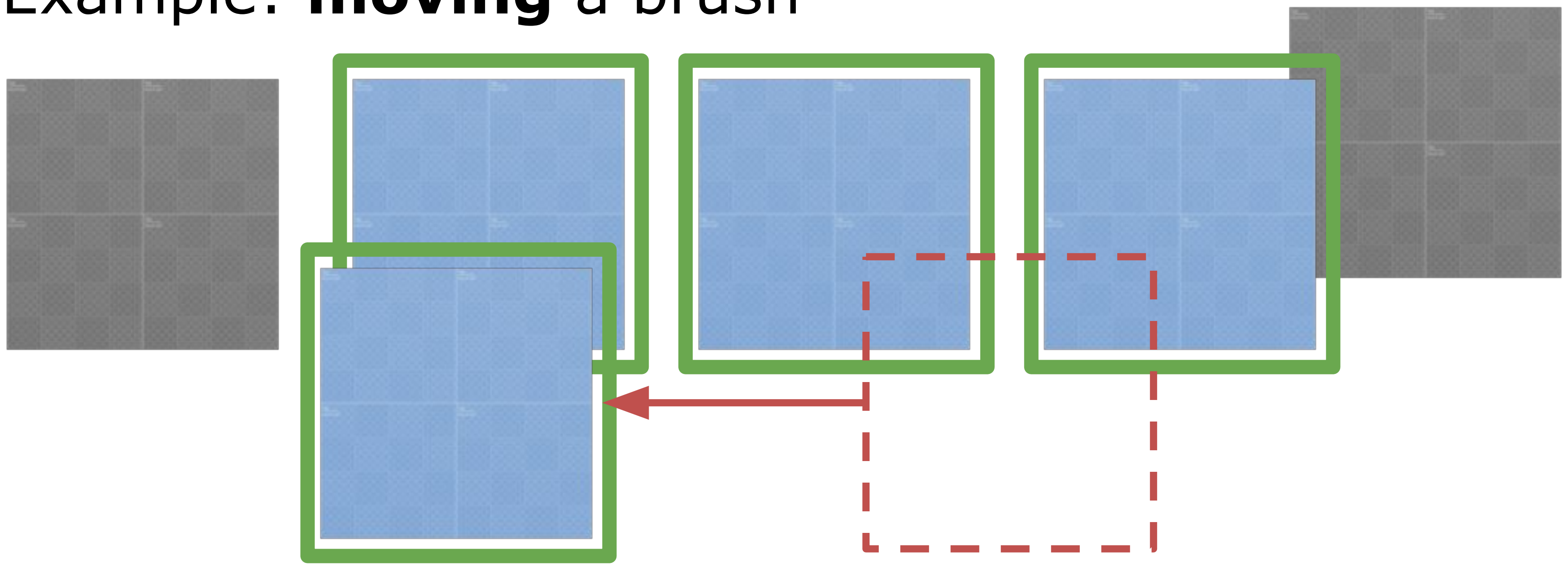


So are those that it touched before the move

Performing CSG per brush

- Allows for **iterative** updates
    - Only need to **update a brush when its modified**
        - And all brushes that **touched**/**touch** it

Example: **moving** a brush

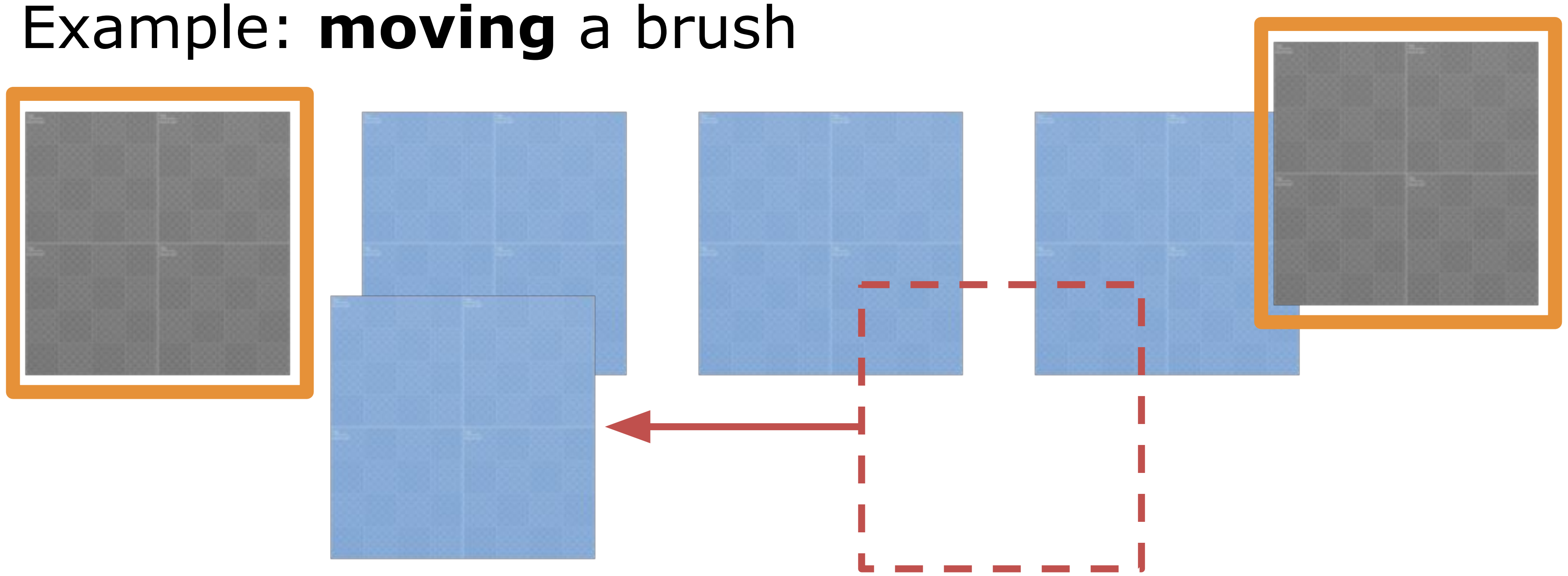


And those that it touches after the move

Performing CSG per brush

- Allows for **iterative** updates
  - Only need to **update a brush when its modified**
    - And all brushes that **touched**/**touch** it
    - But not those that it didn't touch (**can be cached**)

Example: **moving** a brush

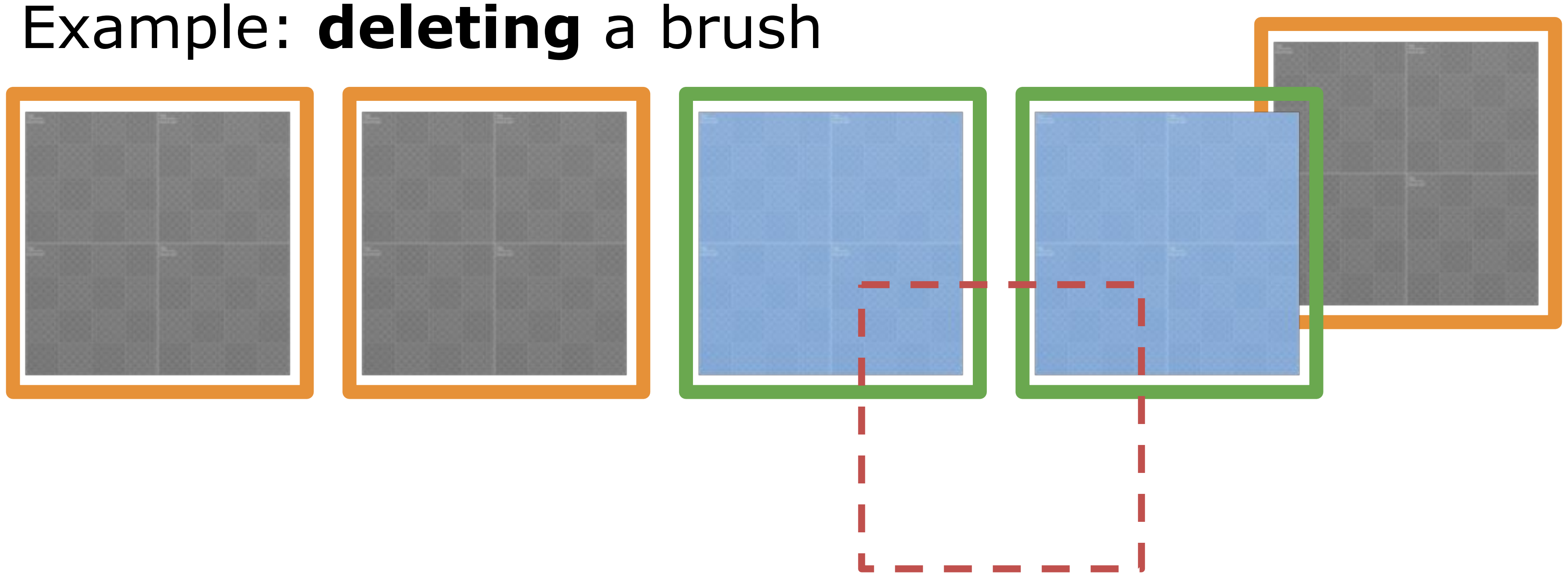


All the other brushes are left unmodified

Performing CSG per brush

- Allows for **iterative** updates
  - Only need to **update a brush when its modified**
    - And all brushes that **touched**/**touch** it
    - But not those that it didn't touch (**can be cached**)

Example: **deleting** a brush

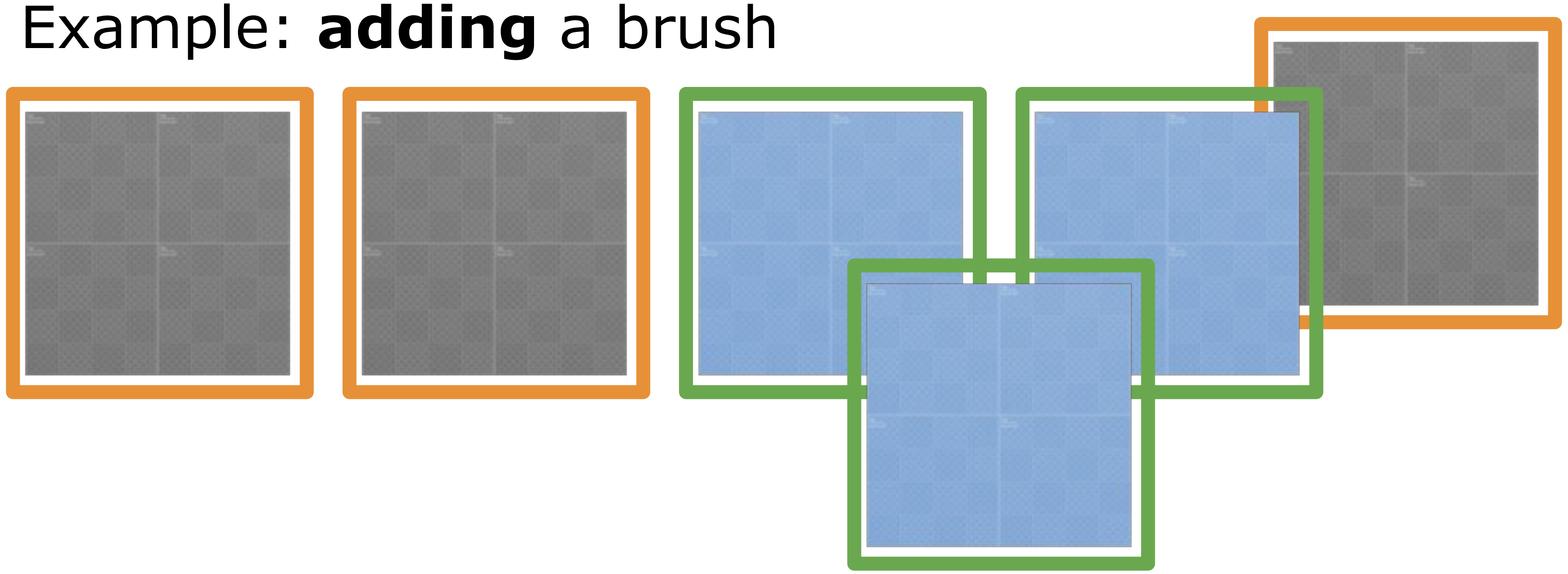


Here we update the brushes it touched before it was deleted

Performing CSG per brush

- Allows for **iterative** updates
  - Only need to **update a brush when its modified**
    - And all brushes that **touched**/**touch** it
    - But not those that it didn’t touch (**can be cached**)

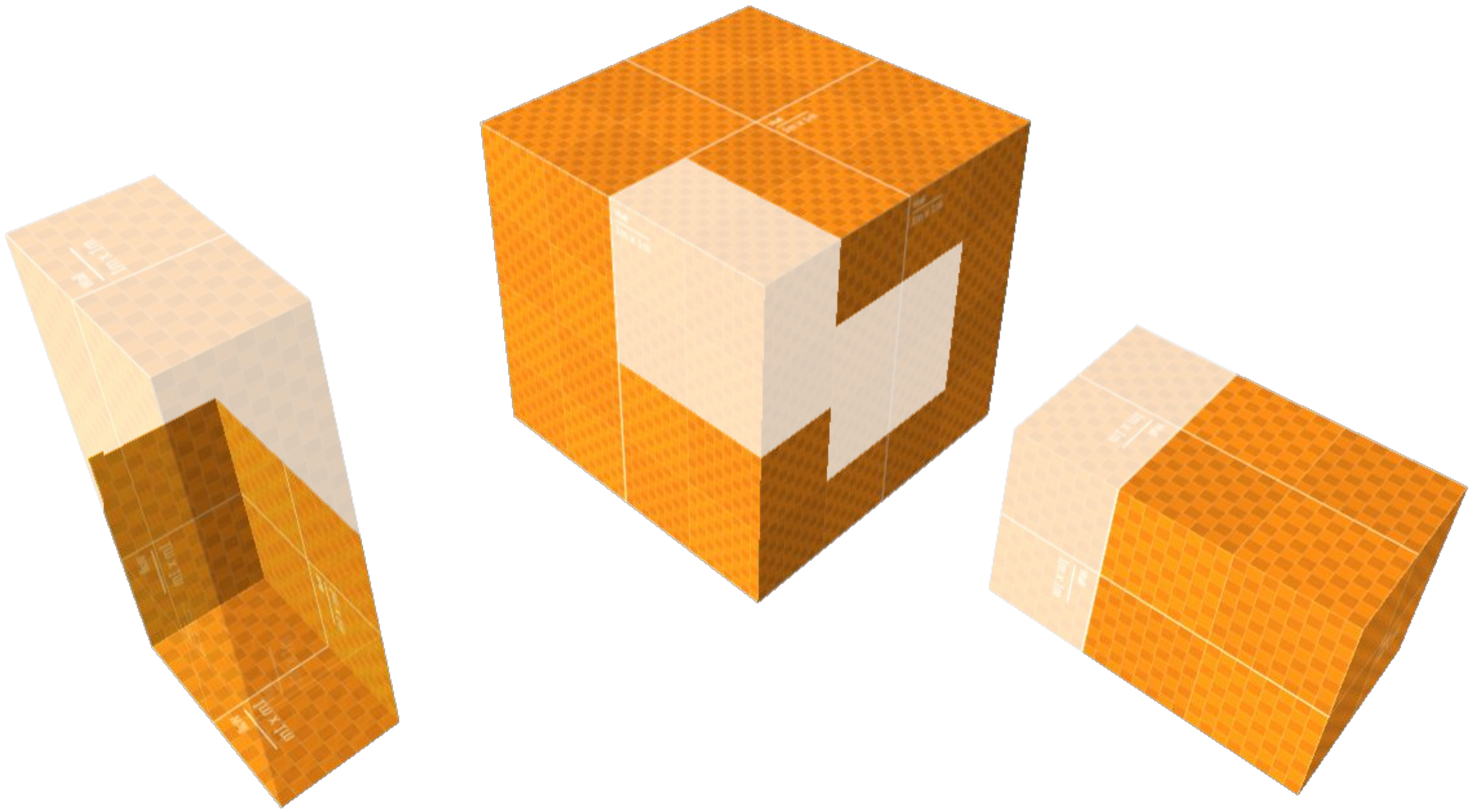
Example: **adding** a brush



And here we update the brushes it touches after creation

Performing CSG per brush

- Allows for **iterative** updates
  - Work can easily be split across **multiple cpu cores**
  - Work per brush doesn’t get too expensive
    - **Scales well** with number of brushes



Remember those polygon pieces?
How do we find them?

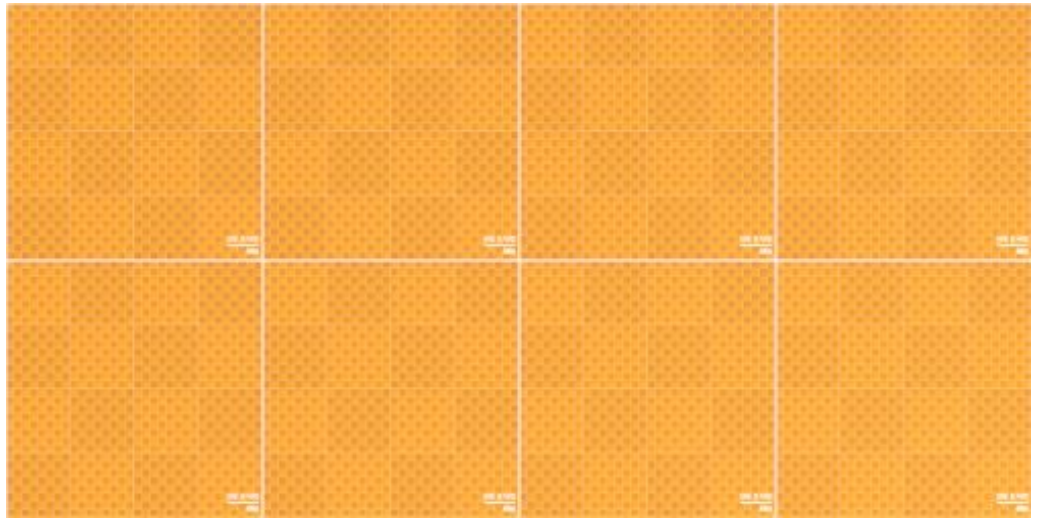
Overview

1. History of CSG

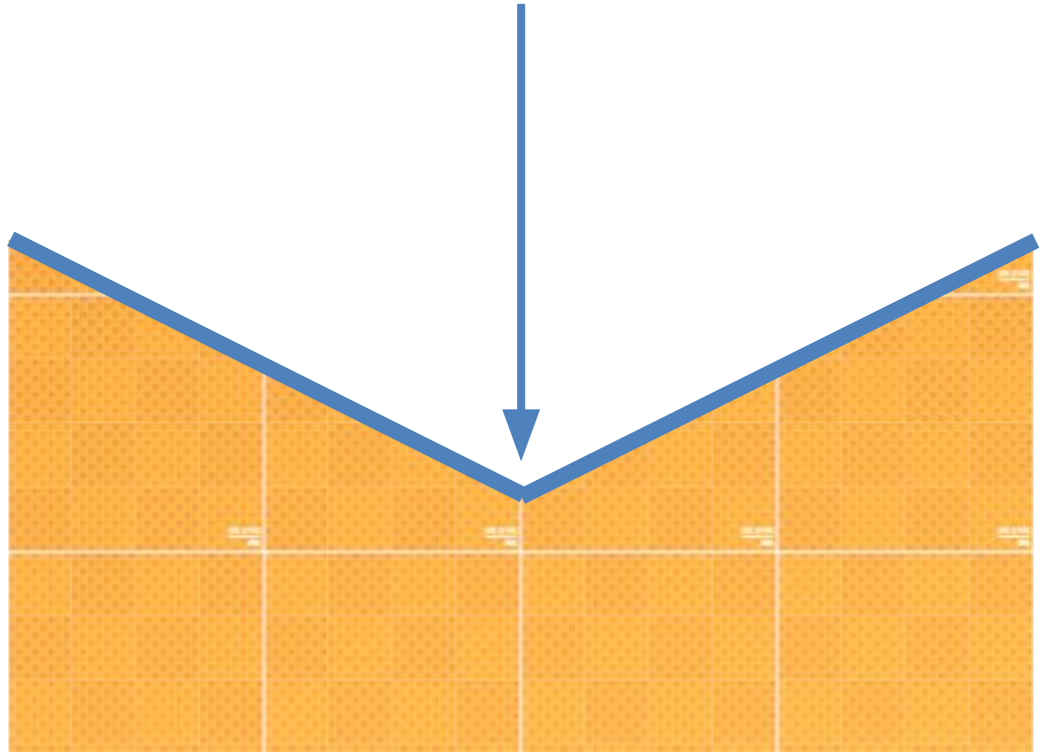
The algorithm

2. Iterative updates
3. **Intersections**
4. Mesh Generation
5. Polygon categories, Routing & Operation tables
6. Putting it all together

Convexity



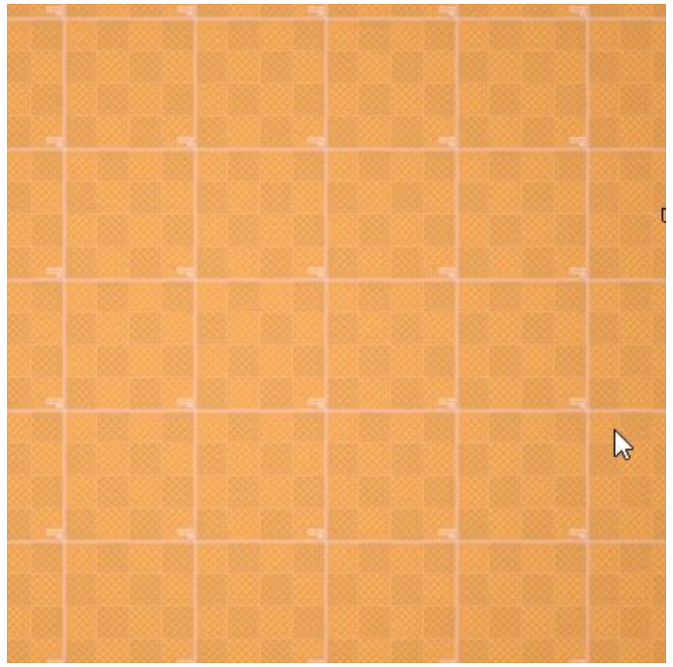
convex



concave

Convex Brushes

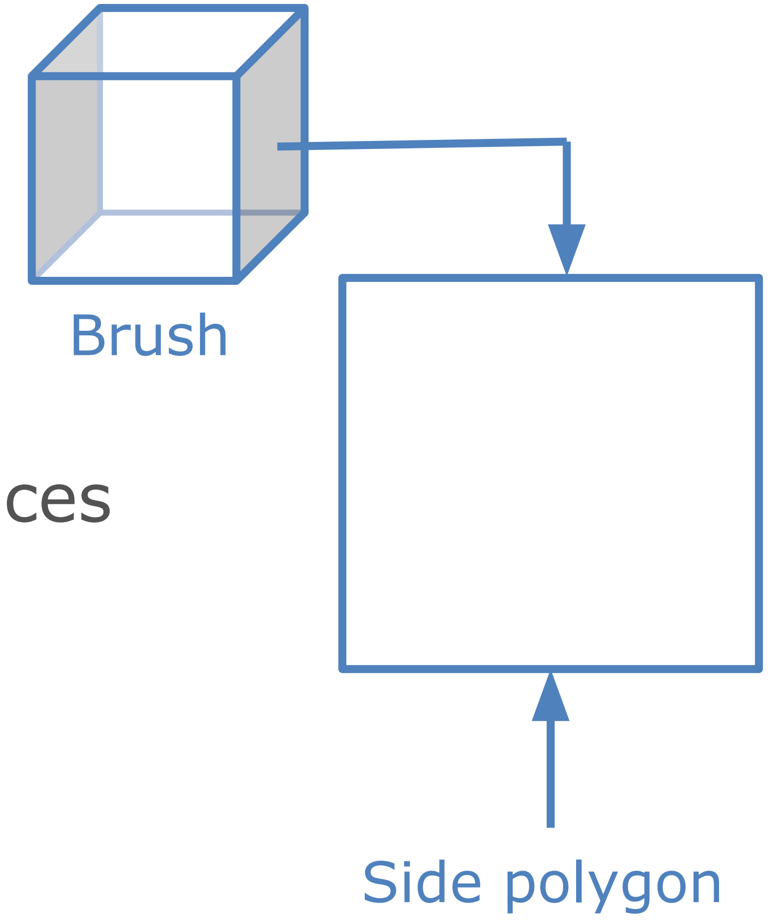
- Can be thought of as an infinite cube sliced multiple times, leaving behind a convex shape
- These “slices” are infinite **planes**
  - Each plane has a facing direction
  - We essentially “remove” everything in front of the planes
- Conceptually convex brushes are “a list of planes”



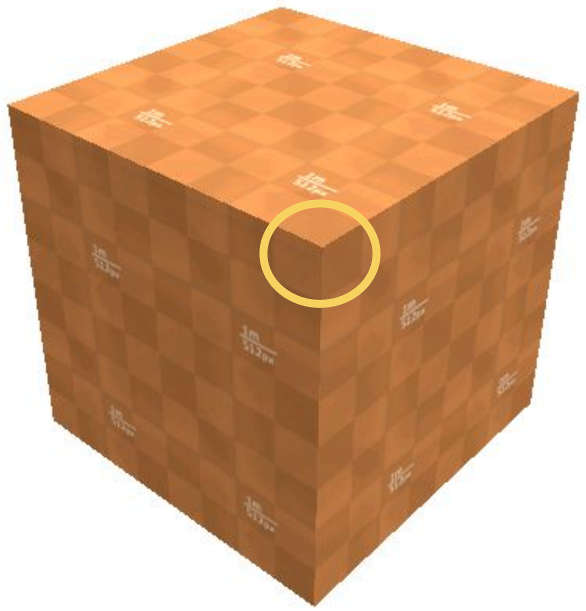
- Convexity is not necessarily a requirement
  - But it makes everything *a lot* simpler & faster
  - You can still build any concave shape from multiple convex shapes

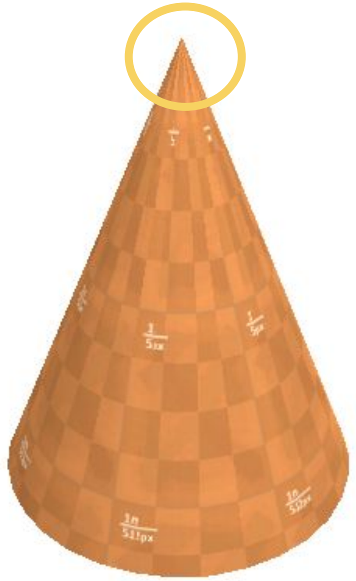
Convex Brushes

- Edges are where **exactly** 2 planes intersect
- Vertices are where ***at least*** 3 planes intersect
- Side polygons are formed between these edges and vertices
  - Each polygon has *a* ***single*** plane going through it



A cube would have just 3 planes intersect at a corner





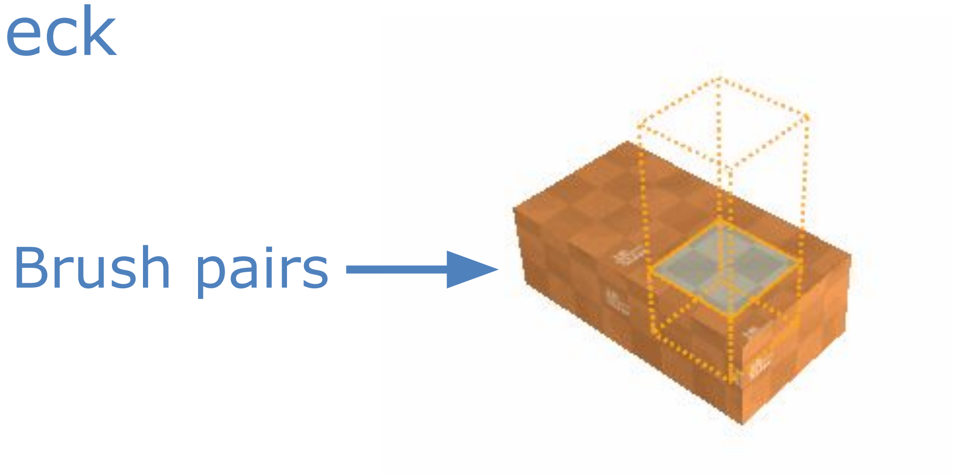
A cone could have an unbounded number of planes intersect at its peak

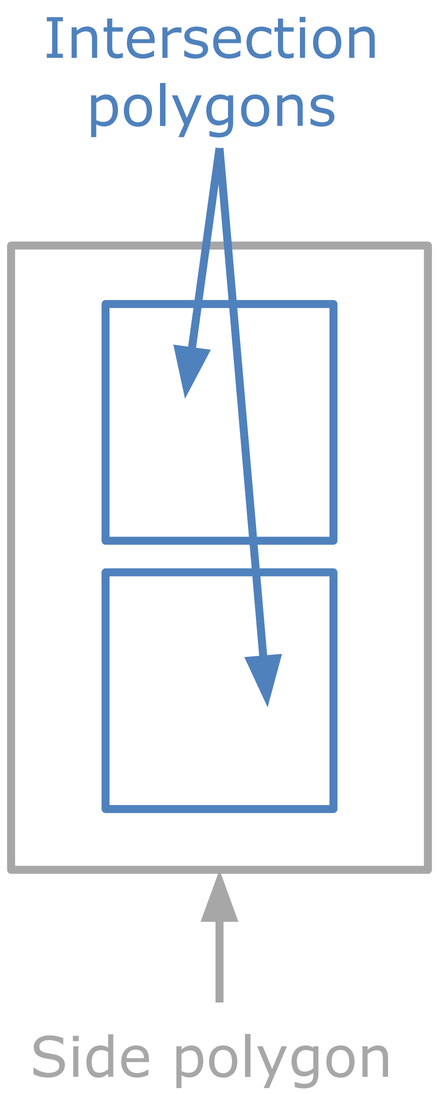
Finding intersecting brushes

- Find intersecting brushes **at insertion time** or **after moving them**
  - Keep in mind that **intersection results are bi-directional**, so you only need to do this once for a pair of brushes.
    - **This ensures identical results**
  - Can use something like hierarchical hashed grids
  - Then, for each potential intersection
    - AABB intersection test
    - Check if vertices of a brush are outside the other brush
- Lots of ways of doing this, this is ***not*** a bottleneck however

Creating **intersection polygons**

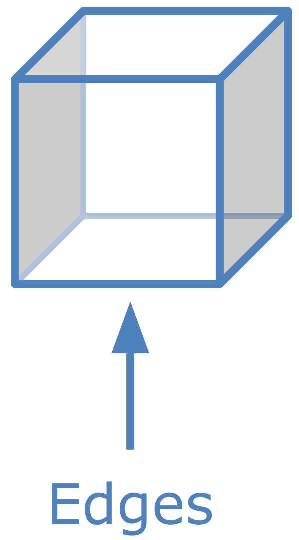
- Process brush **pairs** together
  - Lots of shared information
  - Only consider polygons that intersect with the
    - Use space partition data structures to speed this up
      - Create per brush-shape, can be cached/shared
  - Find polygons that are formed at the intersection between pair
  - Polygons will always be convex if both brushes are convex
  - This ***is*** a bottleneck





Creating **intersection polygons**

- Find all **vertices** of brush
  - Are inside other brush (inside all its other planes)
  - On a plane of the other brush (but inside all its other planes)
- Calculate intersections between brush **edges** with the other brush
  - Find intersection of edge with plane of other brush
    - Intersection vertex must be "inside" all *other* brush planes
  - We can only have 0-2 intersections per edge



Creating **intersection polygons**

- Find all vertices that lie on the same plane on one brush
  - Do not calculate: store plane indices when finding vertices, use those
  - Remember: Our polygon is convex since our brushes are convex

  - Allows us to find **edges** by finding vertex pairs that **share 2 planes**
  - **Connect pairs** by finding **common vertices** between pairs

  - Ensure ordering is correct
    - Calculate normal of vertices (newell's algorithm) and compare with plane normal
      - If dot product between both normals is negative, reverse order of vertices

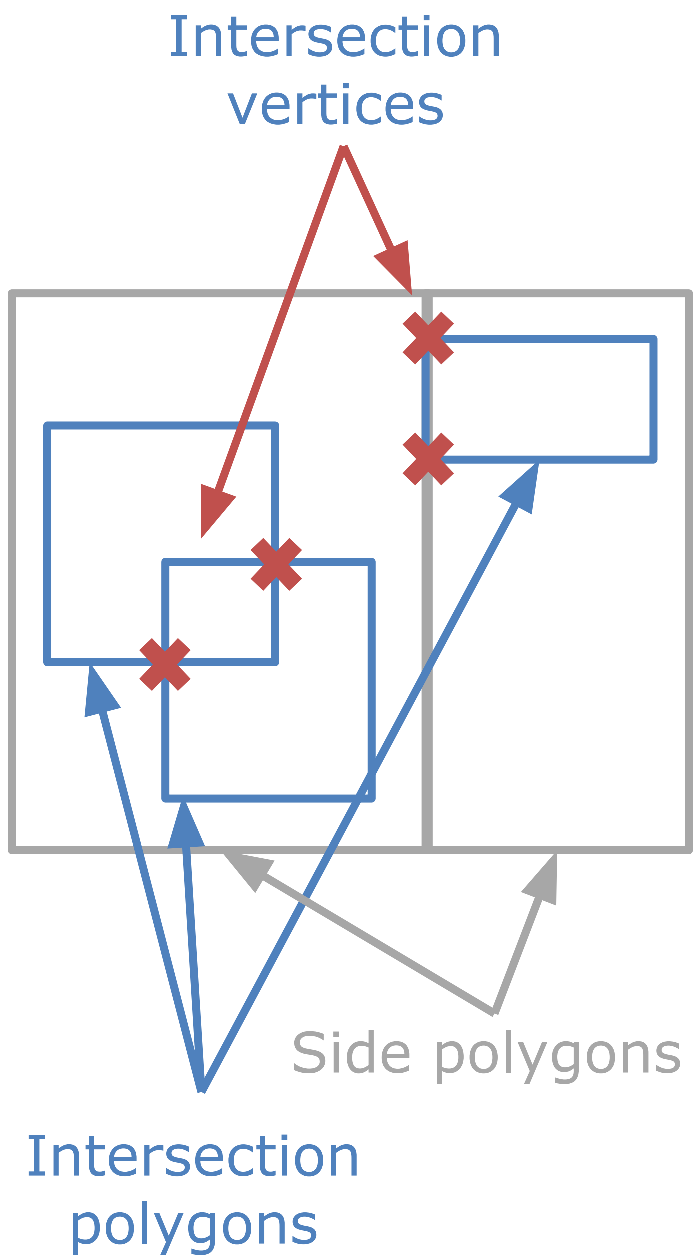
Creating **intersection polygons**

- Store each intersection polygon together with the plane/brush polygon it’s on
- For each intersection polygon
  - Store ***which brush we intersected with***
  - Store an ***interior category*** with this intersection polygon
    - If all the vertices lie on the surface of the other brush,
our category is ***Aligned or Reverse Aligned***
(depending on the orientation of intersecting plane vs side polygon)
    - Otherwise, the polygon is ***Inside***
    - ***Can never be outside***, since this is an intersection
    - We will use this later on in the categorization part

Creating **intersection polygons**

- Find all intersection polygons that **overlap**
  - Add intersection vertex to **both** polygons
    - These polygons are created by intersections with brushes
    - Ensure these vertices are also added to those brushes
      - This avoids gaps

- Do the same with the side polygon the intersection polygons lie on
  - Each edge brush is shared between 2 side polygons on a brush
  - Make sure this vertex exists on both polygons that share edge



Precision

- Make sure that the found vertices are *copied* to the other brush, not recalculated.
  - When the vertices are identical between brushes, there won't be any gaps
    - It *ensures* that the vertices will be 100% identical on all edges
- Note: Snap vertices of intersecting brushes to each other as well, before you do any intersection calculations, for this exact same reason
  - Makes sure vertices are consistent between brushes
- We now have all the vertices we need, we don't need to create any more vertices

Overview

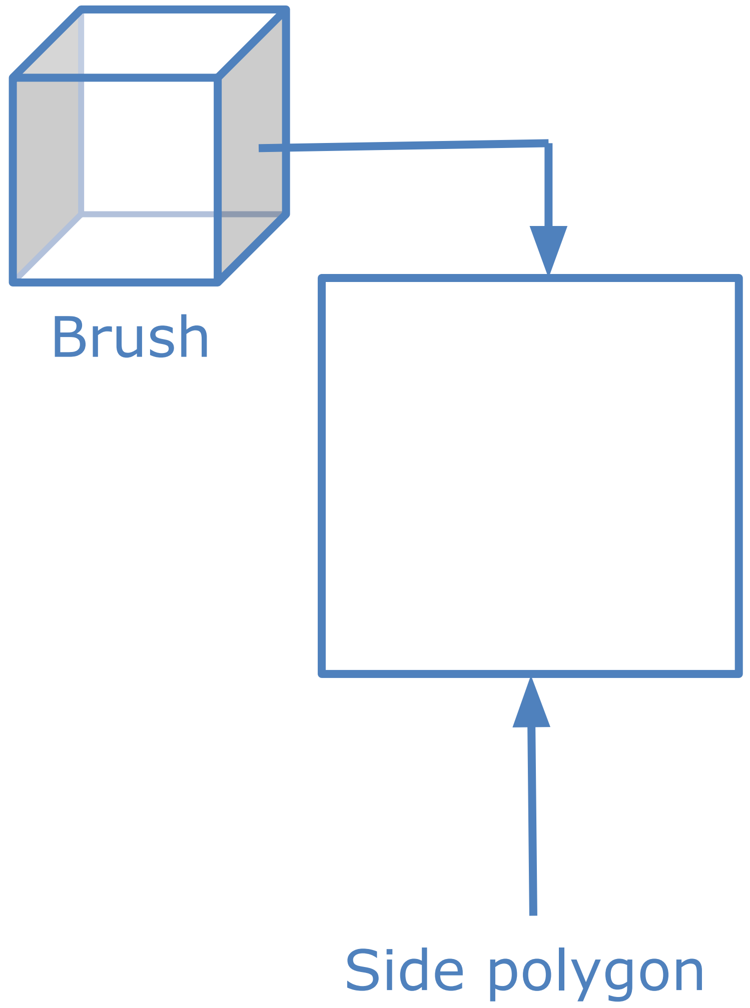
1. History of CSG

The algorithm

2. Iterative updates
3. Intersections
4. **Mesh Generation**
5. Polygon categories, Routing & Operation tables
6. Putting it all together

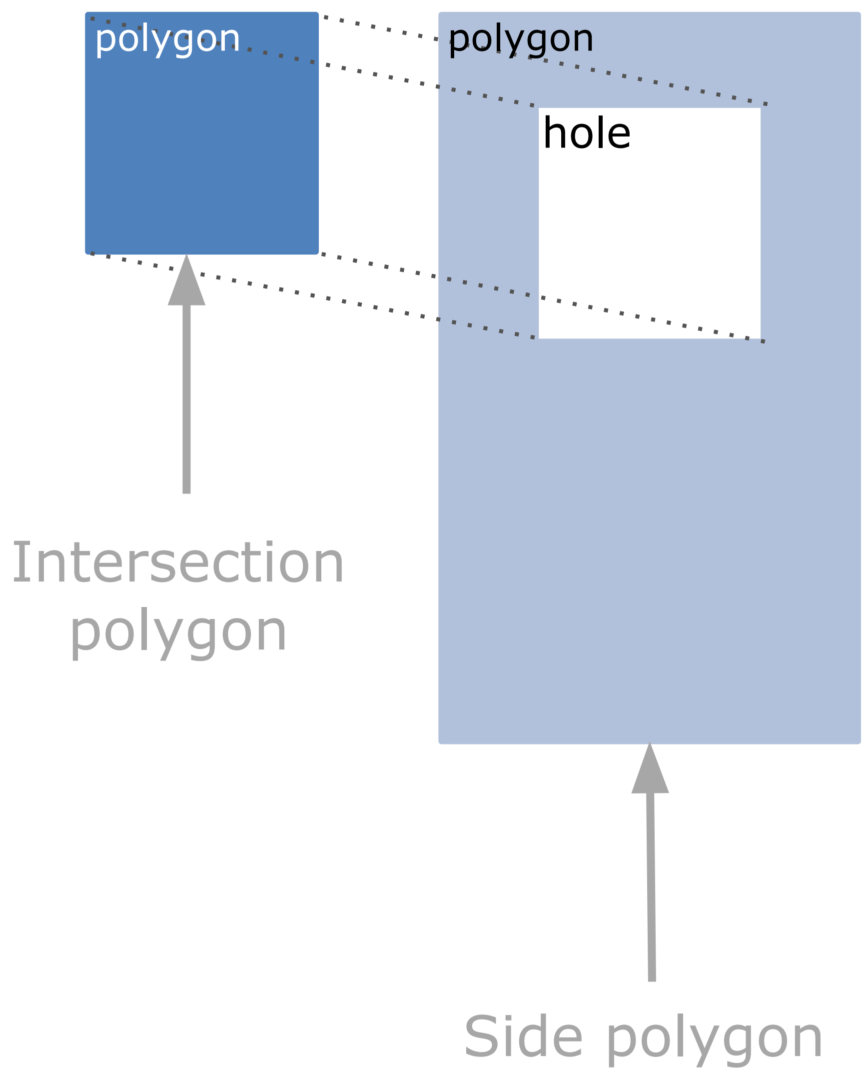
Generating meshes

- We process each brush side separately
  - Here we apply each intersection polygon in order to split our brush side polygon into the pieces that we need



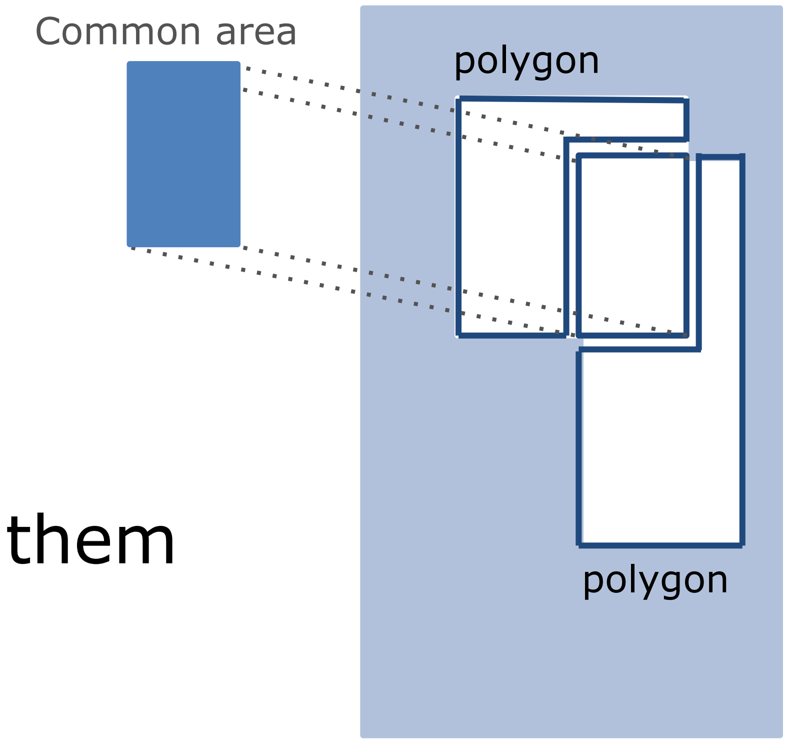
Combining intersection polygons

- Stored as ***both*** a ***hole*** on the current polygon and as a completely ***new polygon***
  - Polygons are triangulated together with its holes



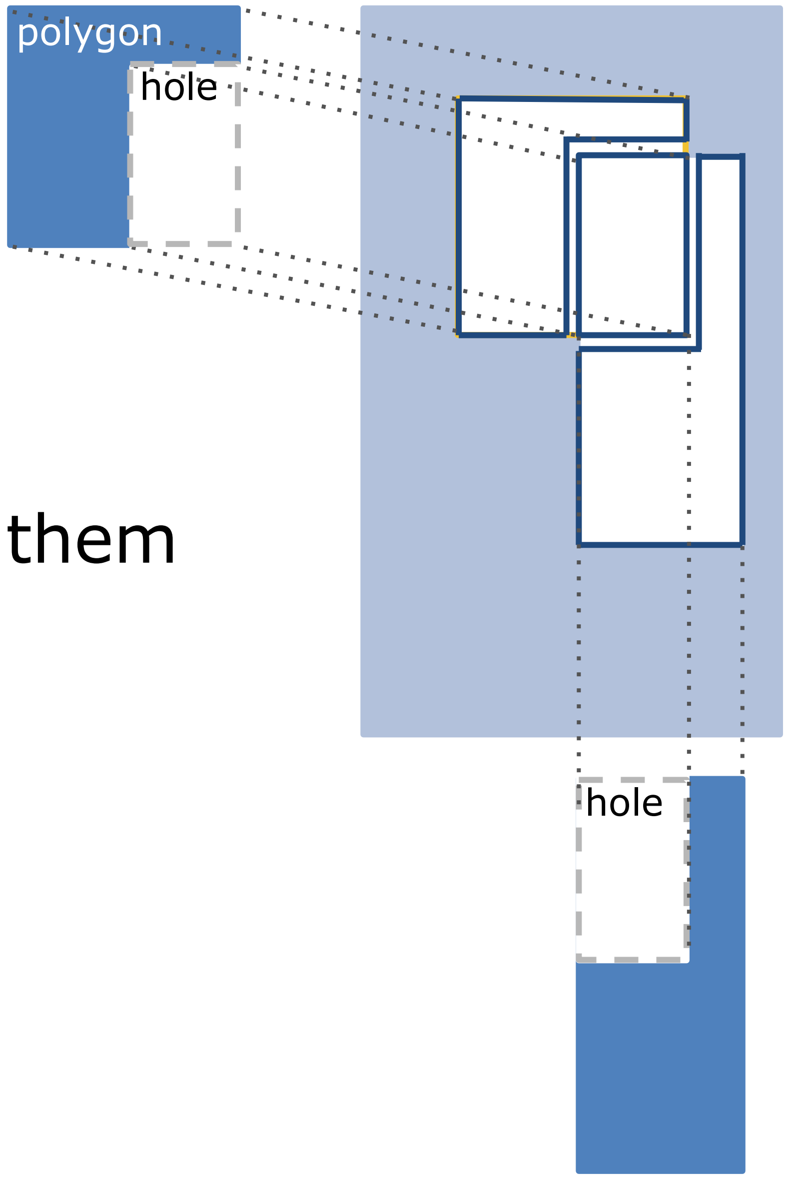
Combining intersection polygons

- Stored as ***both*** a ***hole*** on the current polygon and as a completely ***new polygon***
  - Polygons are triangulated together with its holes
  - Also need to handle ***overlapping polygons***
    - Find ***common area*** between them
      - Find all edges that are inside/on both, combine them
      - Always works if both both polygons are convex



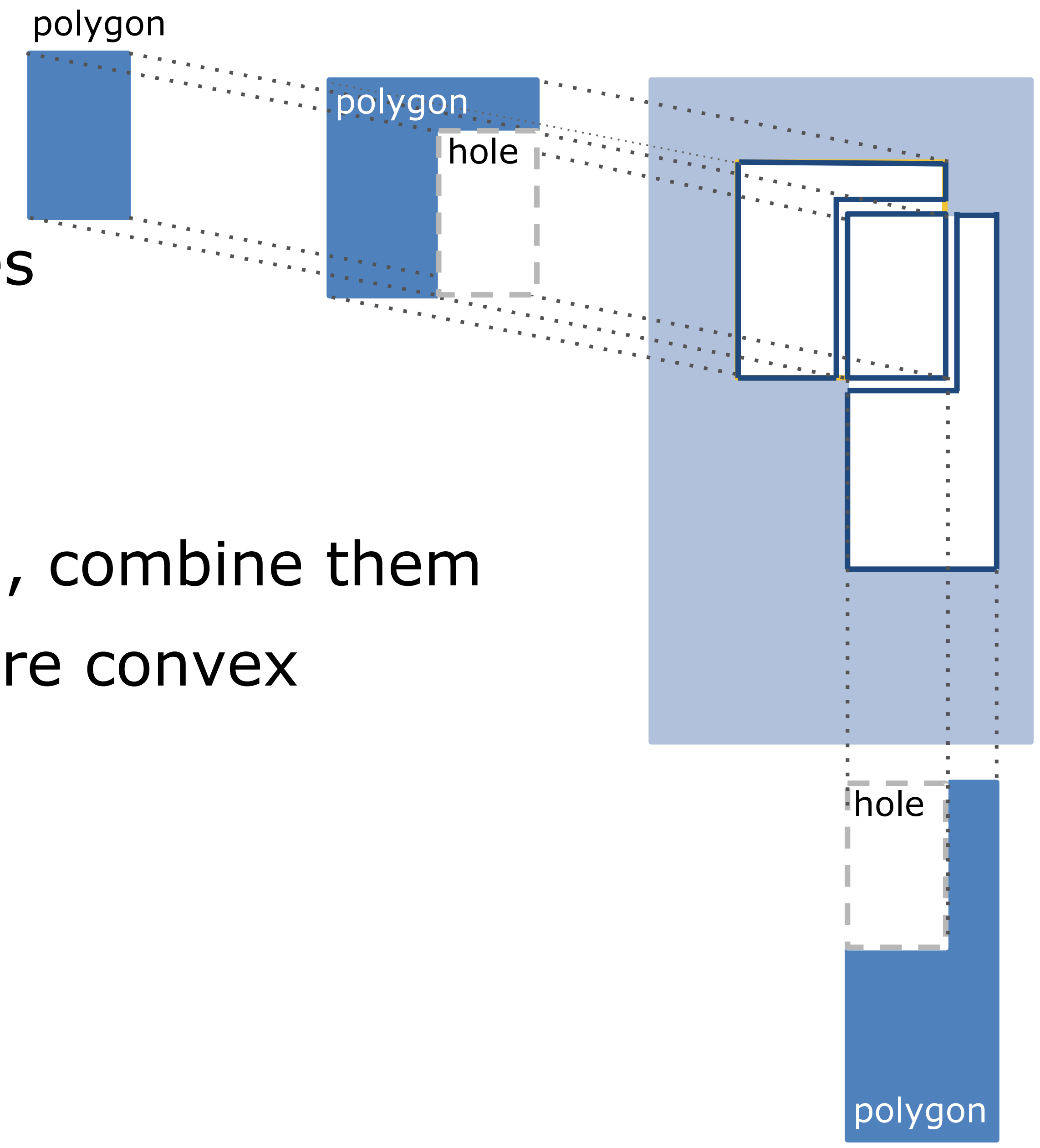
Combining intersection polygons

- Stored as ***both*** a ***hole*** on the current polygon and as a completely ***new polygon***
  - Polygons are triangulated together with its holes
  - Also need to handle ***overlapping polygons***
    - Find ***common area*** between them
      - Find all edges that are inside/on both, combine them
      - Always works if both both polygons are convex
    - Becomes a ***hole on both polygons***



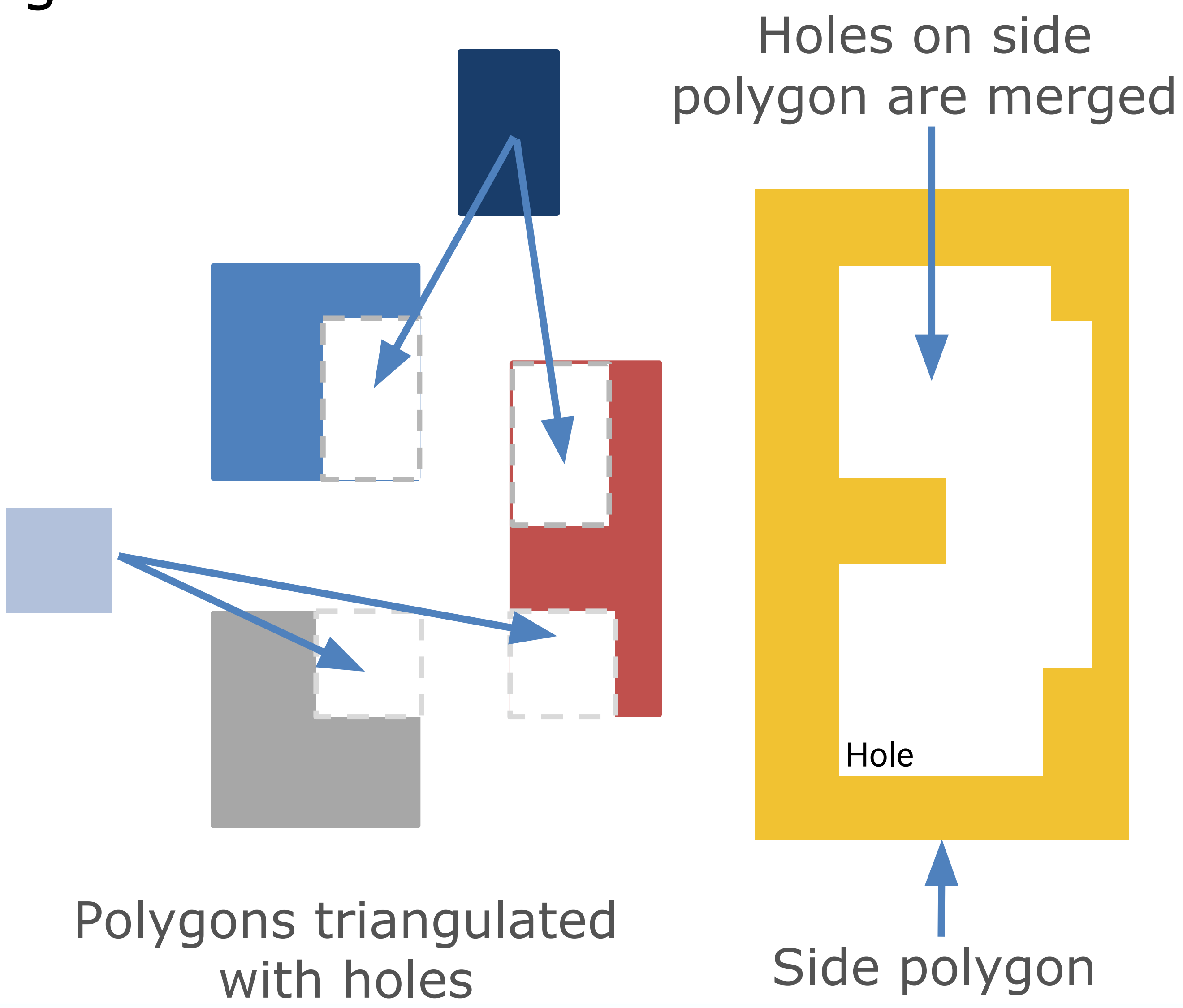
Combining intersection polygons

- Stored as ***both*** a ***hole*** on the current polygon and as a completely ***new polygon***
  - Polygons are triangulated together with its holes
  - Also need to handle ***overlapping polygons***
    - Find ***common area*** between them
      - Find all edges that are inside/on both, combine them
      - Always works if both both polygons are convex
    - Becomes a ***hole on both polygons***
    - and a ***new polygon***



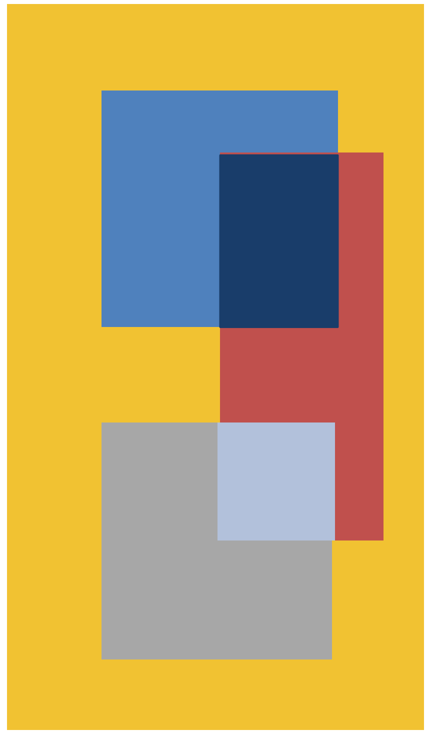
Creating brush meshes

- We triangulate each polygon separately along with its holes
    - Merge the holes by removing overlapping edges and combining all the remaining edges



Creating brush meshes

- We triangulate each polygon separately along with its holes
    - Merge the holes by removing overlapping edges and combining all the remaining edges
    - Each polygon is triangulated using vertex indices
        - Already found all vertices at the beginning



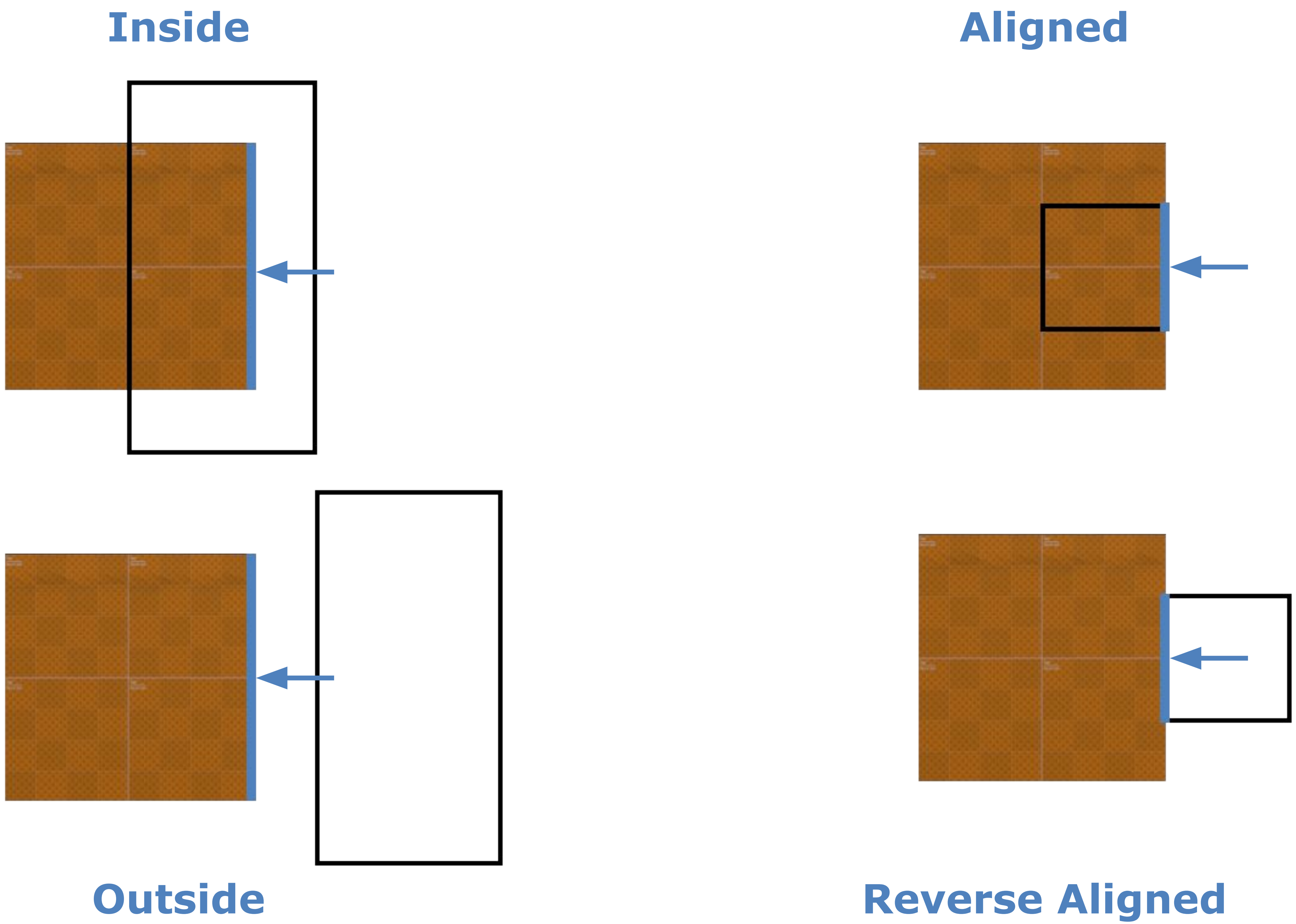
Overview

1. History of CSG

The algorithm

2. Iterative updates
3. Intersections
4. Mesh Generation
5. **Polygon categories, Routing & Operation tables**
6. Putting it all together

Categories



Brush categorization

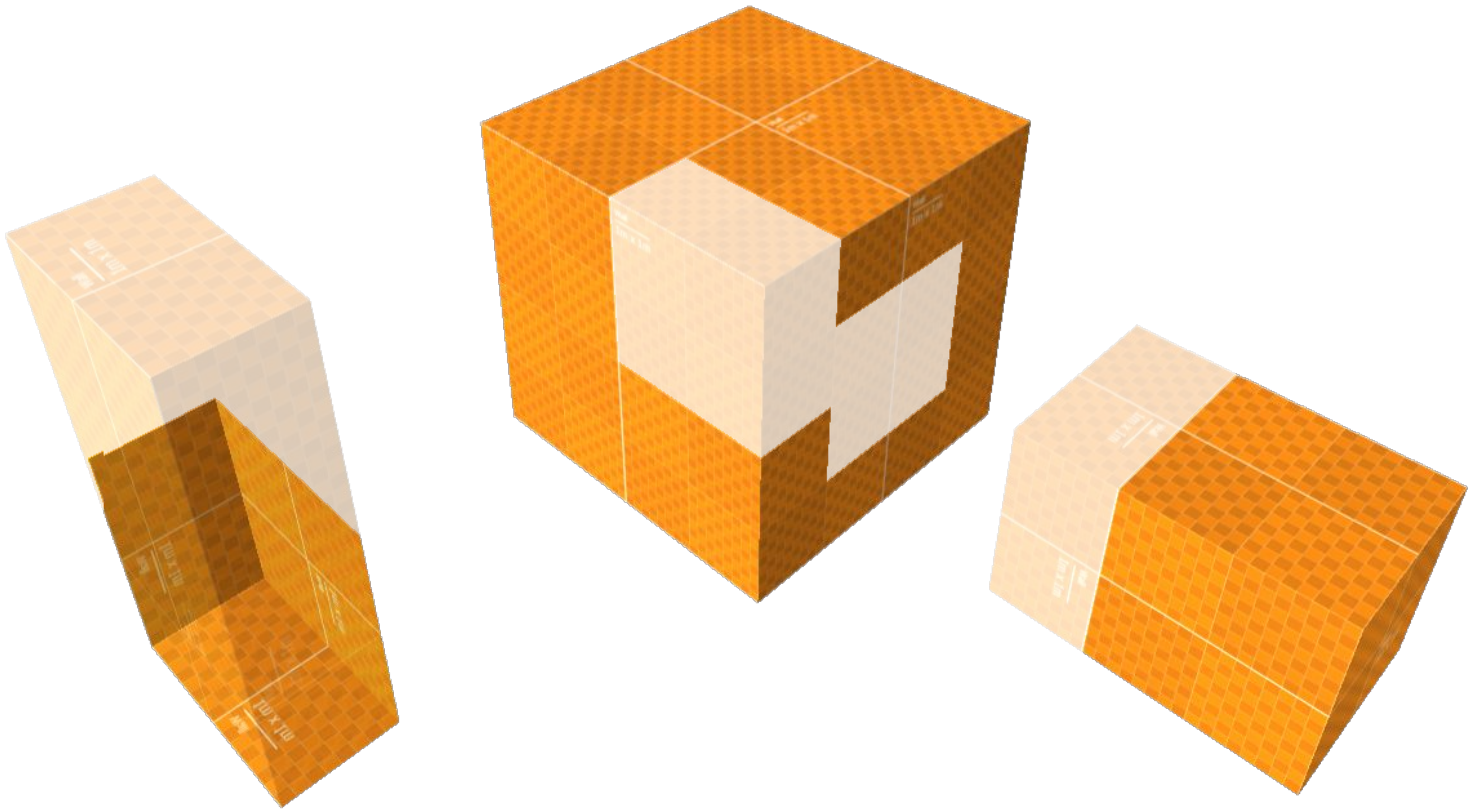
- How to categorize a vertex against a single brush:
  - Calculate distance of a vertex against *each* plane
  - Positive value, compared to *any* plane: it’s **outside** (early out)
  - *Near* zero value: it’s **aligned**
  - Neither outside or aligned to *any* plane: it’s **inside**

Brush categorization

- How to categorize a polygon against a single brush:
    - Otherwise
        - If **all** vertices of a polygon are **(reverse) aligned**, then that's the polygons' category.
        - If **any** vertex is inside/outside, it's **inside**/**outside**
            - Some vertices might be aligned with/touch another brush
            - If one vertex of an edge is inside and the other is outside, then it's intersecting the brush
                - We already found all intersections, so this won't happen

Brush categorization

- How to categorize a polygon against a single brush:
  - If it’s aligned
    - Compare normal of polygon to normal of plane
      - Opposite direction: **reverse-aligned**
      - Same direction: **aligned**



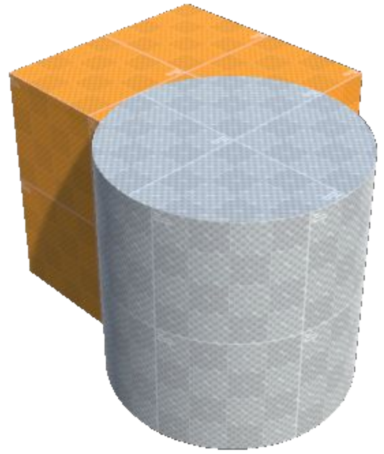
Which polygon piece is what,
to the ***entire generated mesh?***

Use a lookup table to combine categories among multiple brushes

- Find the polygon category for each brush individually
- Combine categories using an operation table
- Note: Polygon does not need to be part of either brush

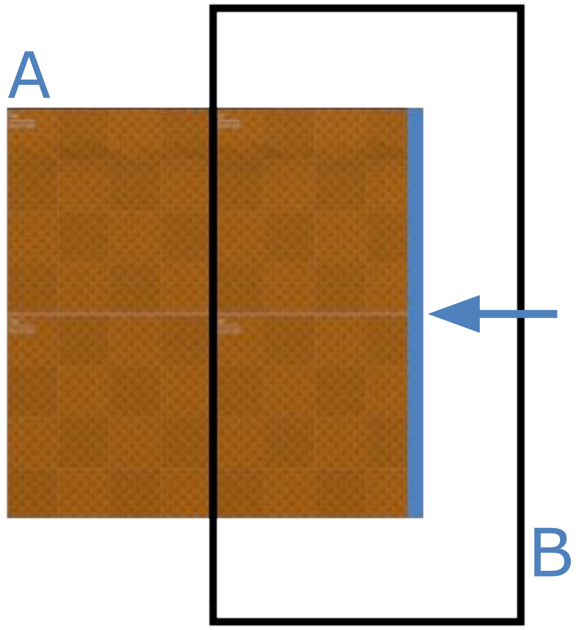
Additive Operation Table		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside

Brush A



Brush B

Use a lookup table to combine categories among multiple brushes

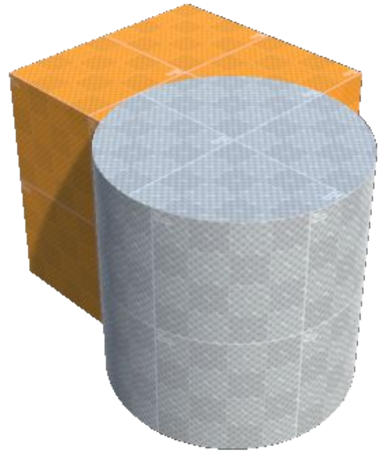


If polygon has the **inside** category for **either** brush, it's inside **both** brushes

Additive Operation Table

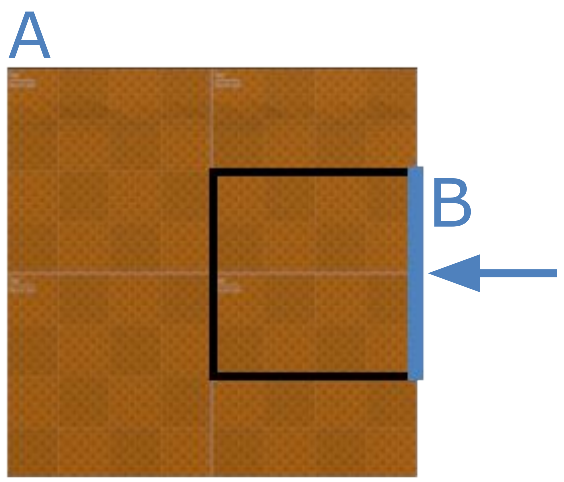
		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside

Brush A



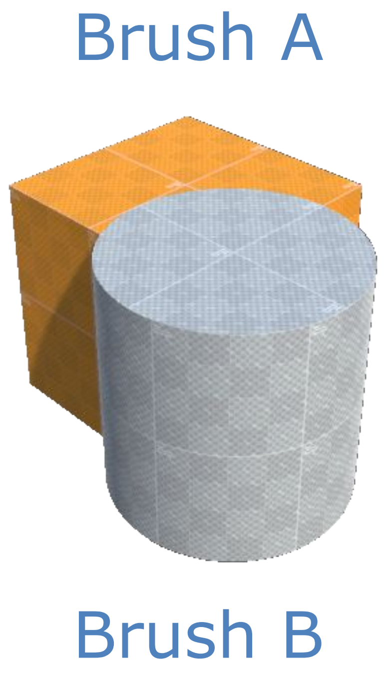
Brush B

Use a lookup table to combine categories among multiple brushes

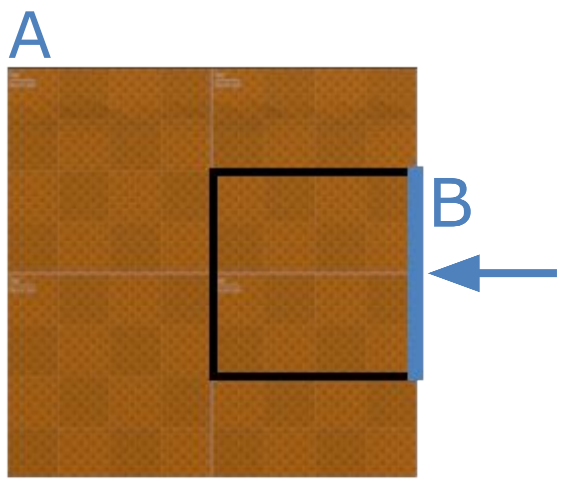


If polygon has the **aligned** category for **both** brushes, it's aligned

Additive Operation Table	Brush A: Inside	Brush A: Aligned	Brush A: Rev-Aligned	Brush A: Outside
Brush B: Inside	Inside	Inside	Inside	Inside
Brush B: Aligned	Inside	Aligned	Inside	Aligned
Brush B: Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
Brush B: Outside	Inside	Aligned	Rev-Aligned	Outside



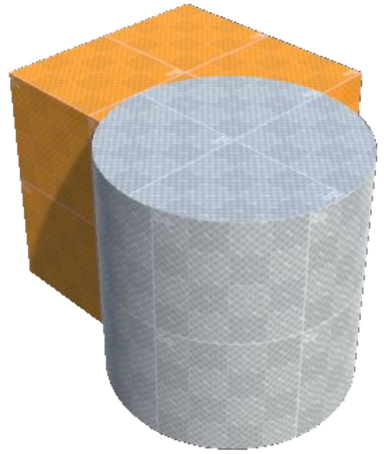
Use a lookup table to combine categories among multiple brushes



If polygon has the **reverse-aligned** category for **both** brushes, it's reverse-aligned

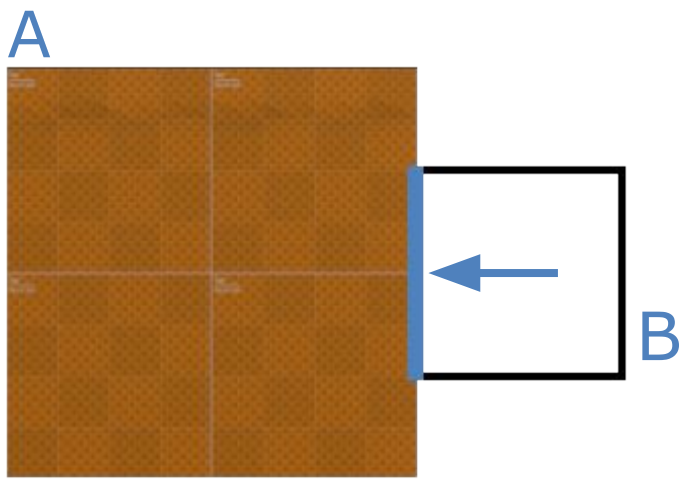
Additive Operation Table		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside

Brush A



Brush B

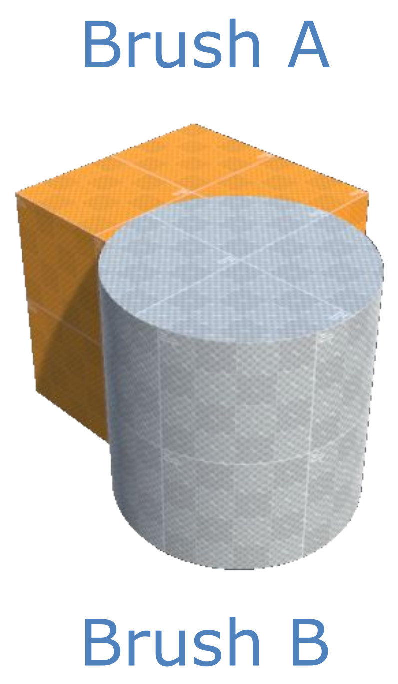
Use a lookup table to combine categories among multiple brushes



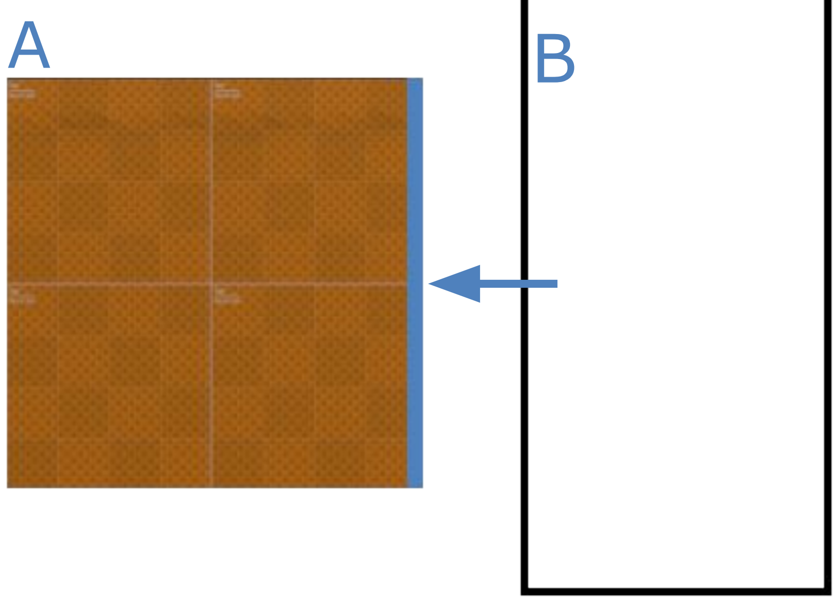
If categories are **reverse-aligned** and **aligned**, the final category is **inside** (surfaces cancel each other out)

Additive Operation Table

		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside



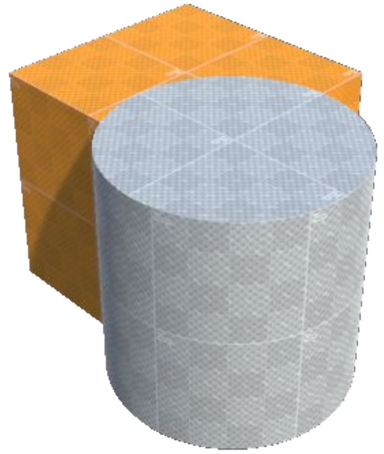
Use a lookup table to combine categories among multiple brushes



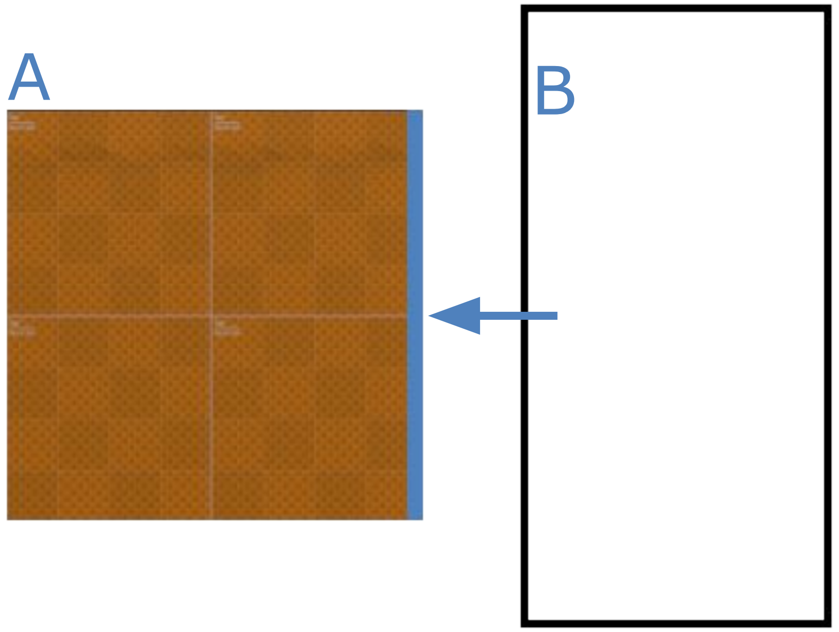
If polygon has the **outside** category for **either** brush, it's the **category of the other brush**

Additive Operation Table

		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside



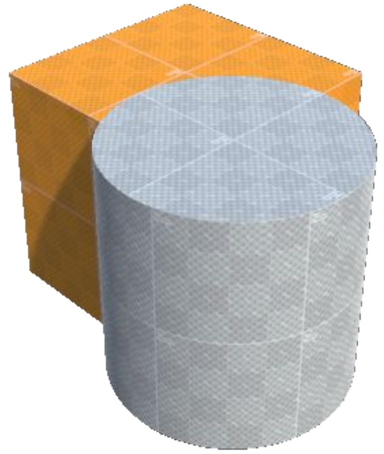
Use a lookup table to combine categories among multiple brushes



If polygon has the **outside** category for **both** brushes, it’s **outside**

Additive Operation Table		Brush A: Inside	Brush A: Aligned	Brush A: Rev-Aligned	Brush A: Outside
Brush B	Inside	Inside	Inside	Inside	Inside
Brush B	Aligned	Inside	Aligned	Inside	Aligned
Brush B	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
Brush B	Outside	Inside	Aligned	Rev-Aligned	Outside

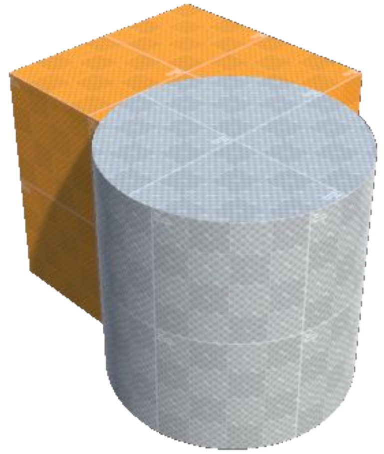
Brush A



Brush B

Use a lookup table to combine categories among multiple brushes

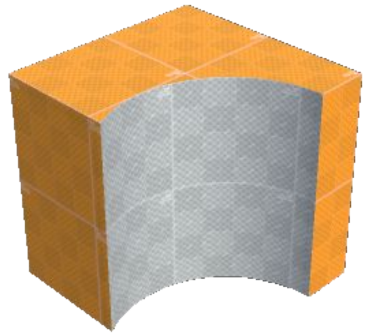
Additive Operation Table		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside



Use a lookup table to combine categories among multiple brushes

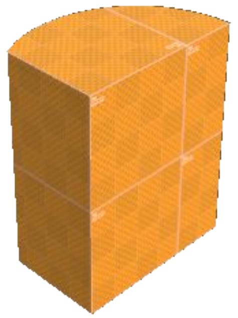
Subtractive Operation Table

		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Outside	Rev-Aligned	Aligned	Inside
	Aligned	Outside	Outside	Aligned	Aligned
	Rev-Aligned	Outside	Rev-Aligned	Outside	Rev-Aligned
	Outside	Outside	Outside	Outside	Outside

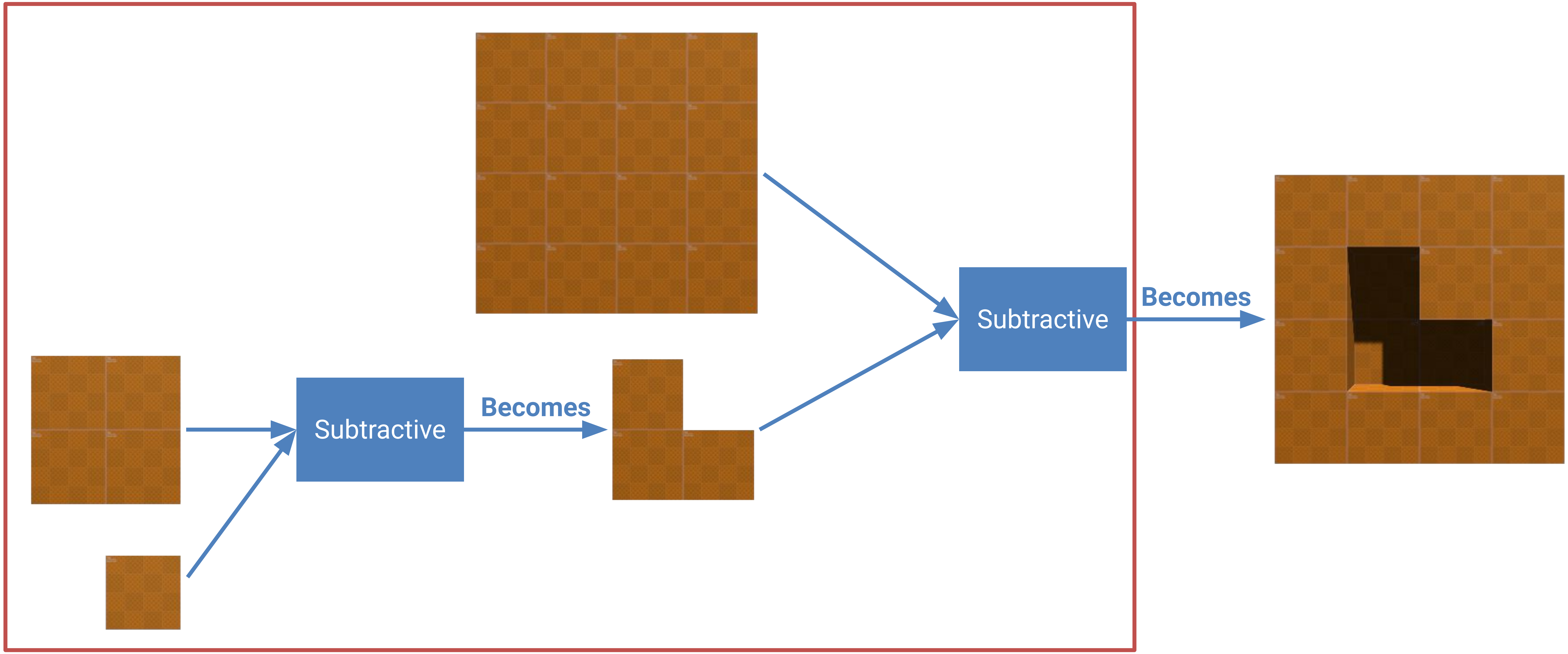


Use a lookup table to combine categories among multiple brushes

Intersecting Operation Table		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Aligned	Rev-Aligned	Outside
	Aligned	Aligned	Aligned	Outside	Outside
	Rev-Aligned	Rev-Aligned	Outside	Rev-Aligned	Outside
	Outside	Outside	Outside	Outside	Outside

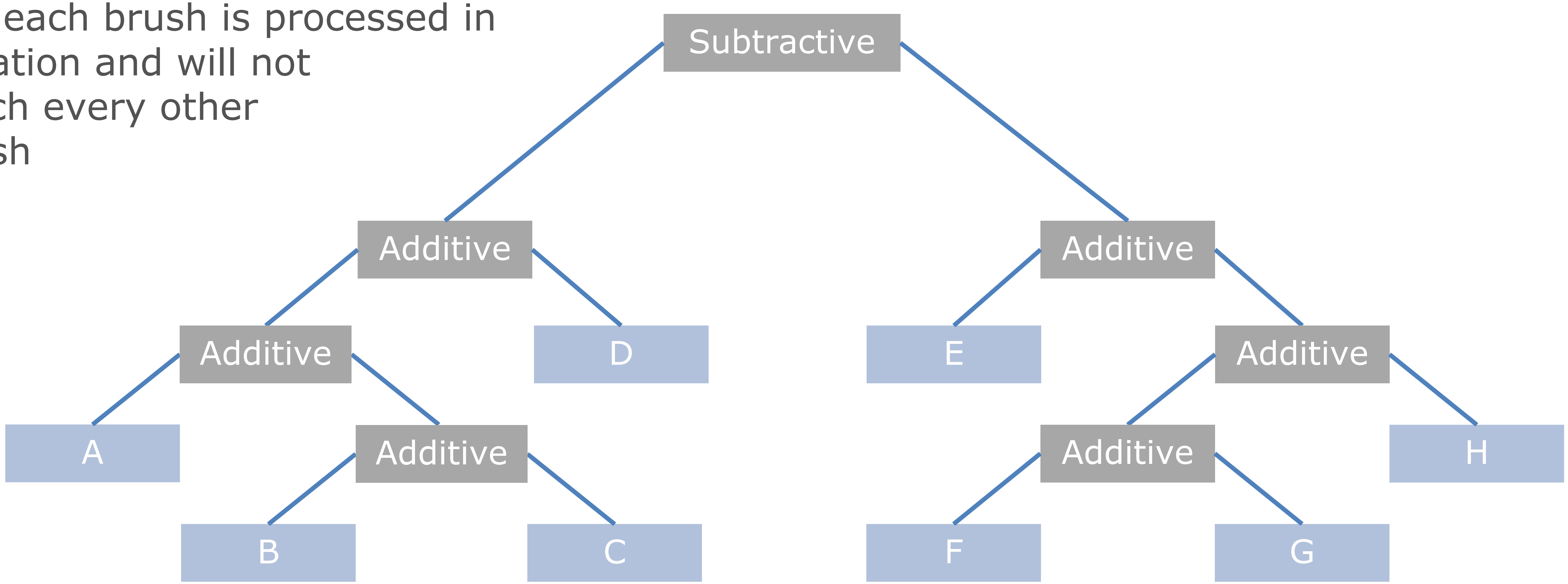


CSG Tree



CSG Tree

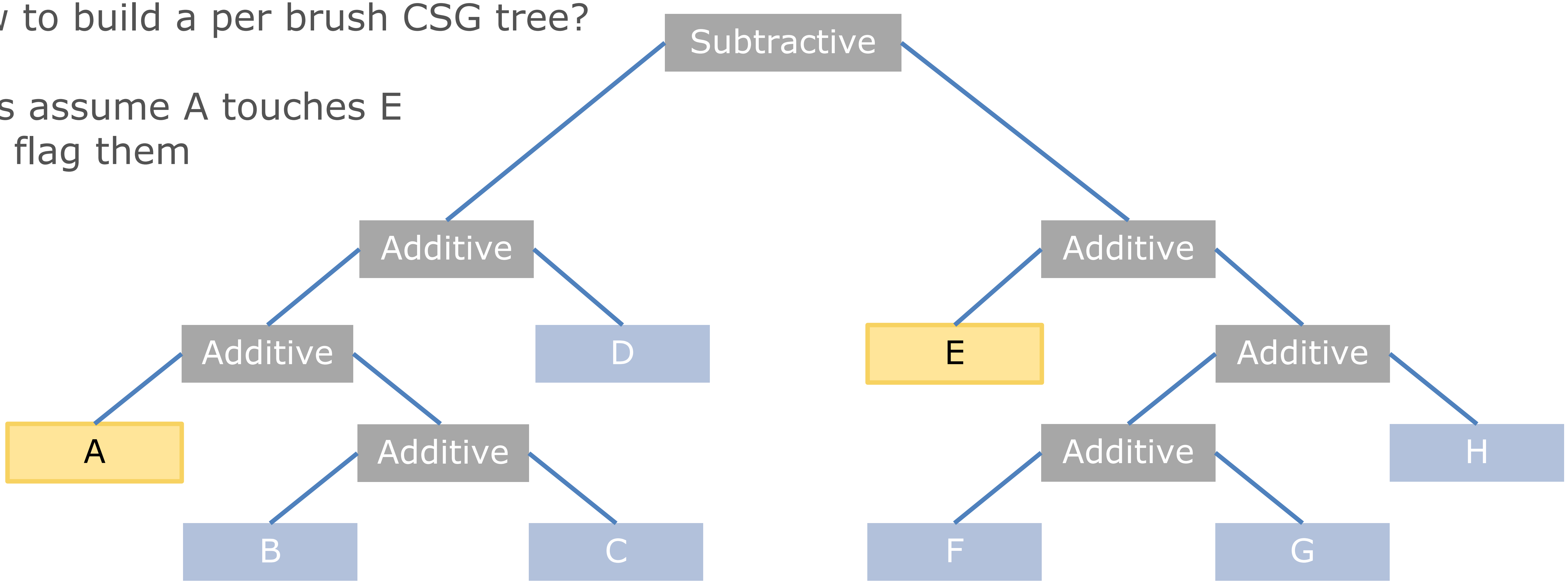
But each brush is processed in isolation and will not touch every other brush



Per brush CSG Tree

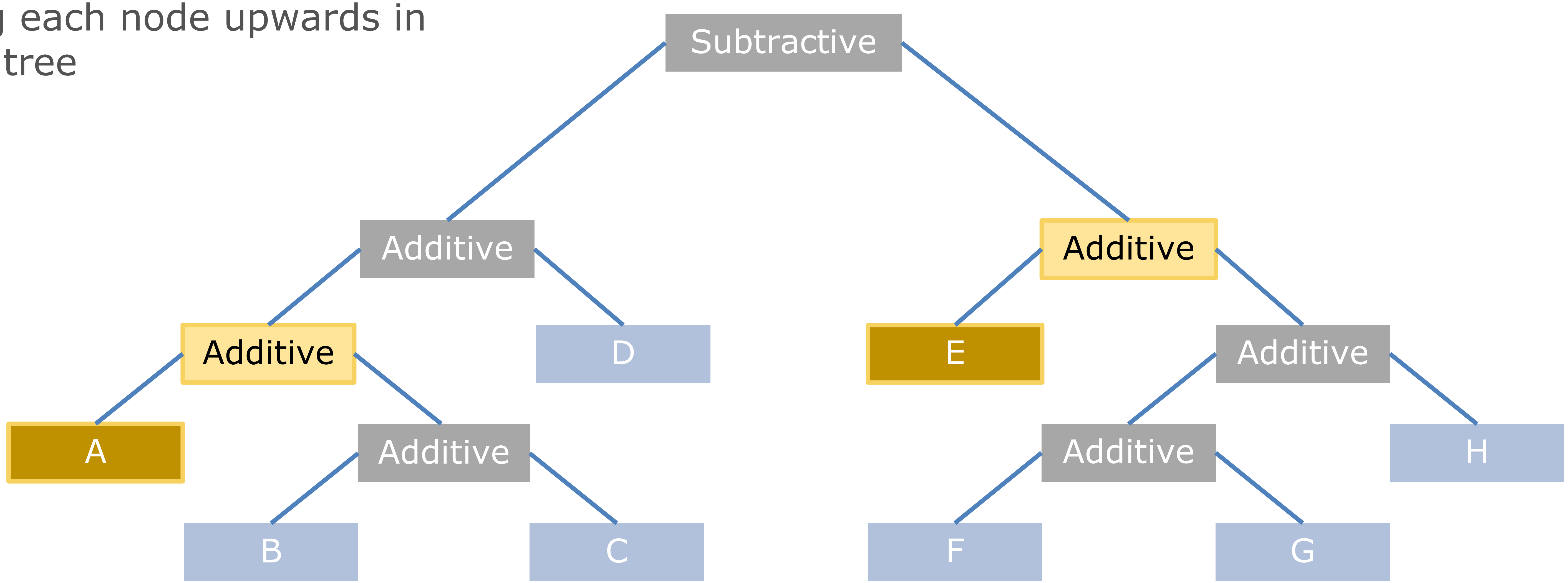
How to build a per brush CSG tree?

Let's assume A touches E
And flag them



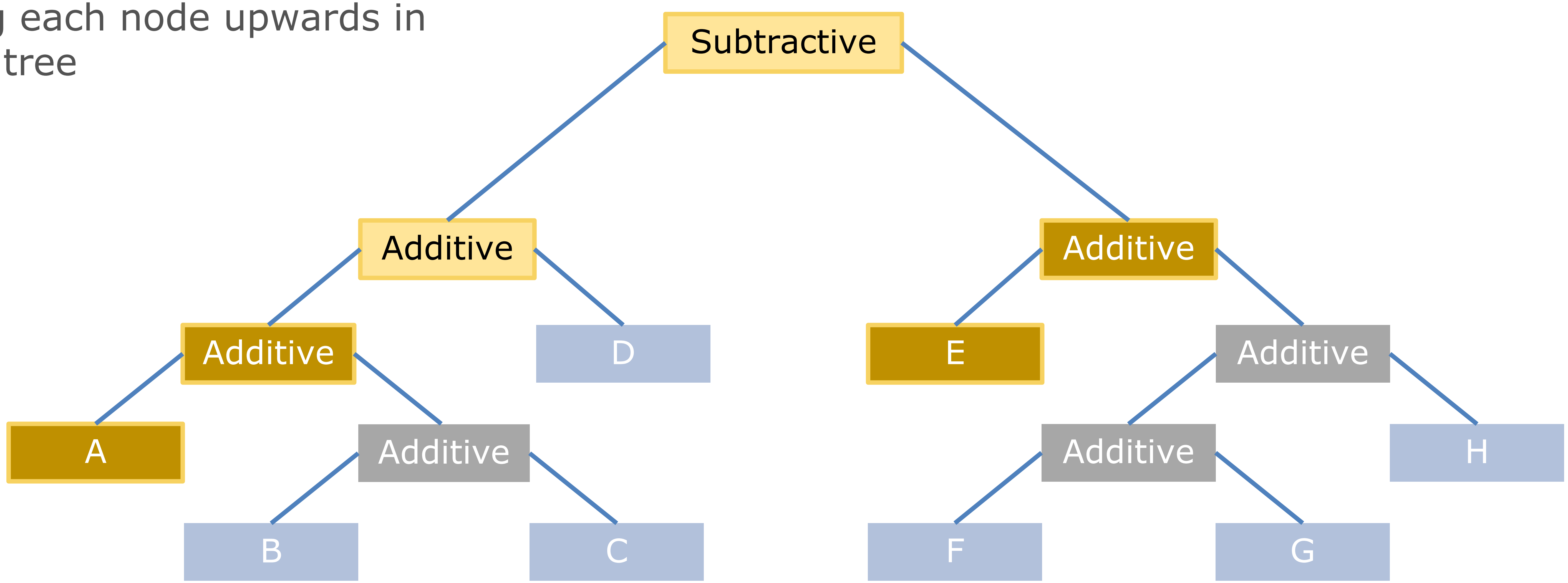
Per brush CSG Tree

Flag each node upwards in the tree



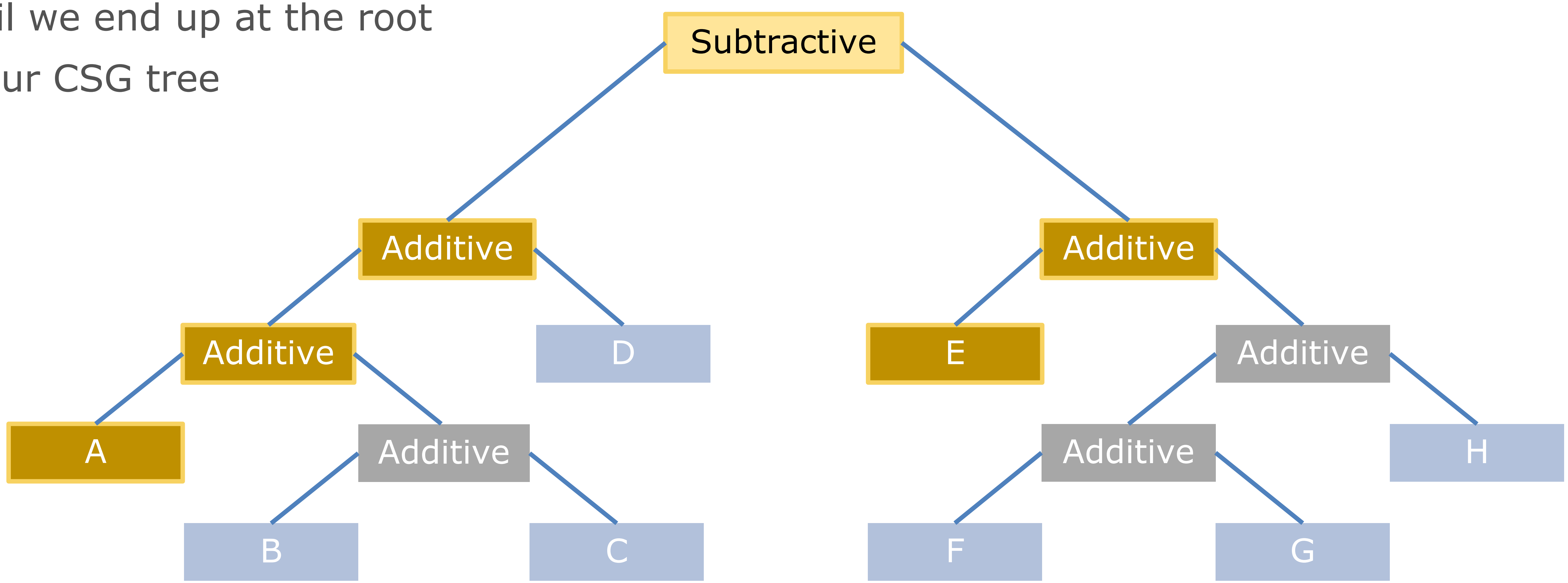
Per brush CSG Tree

Flag each node upwards in the tree



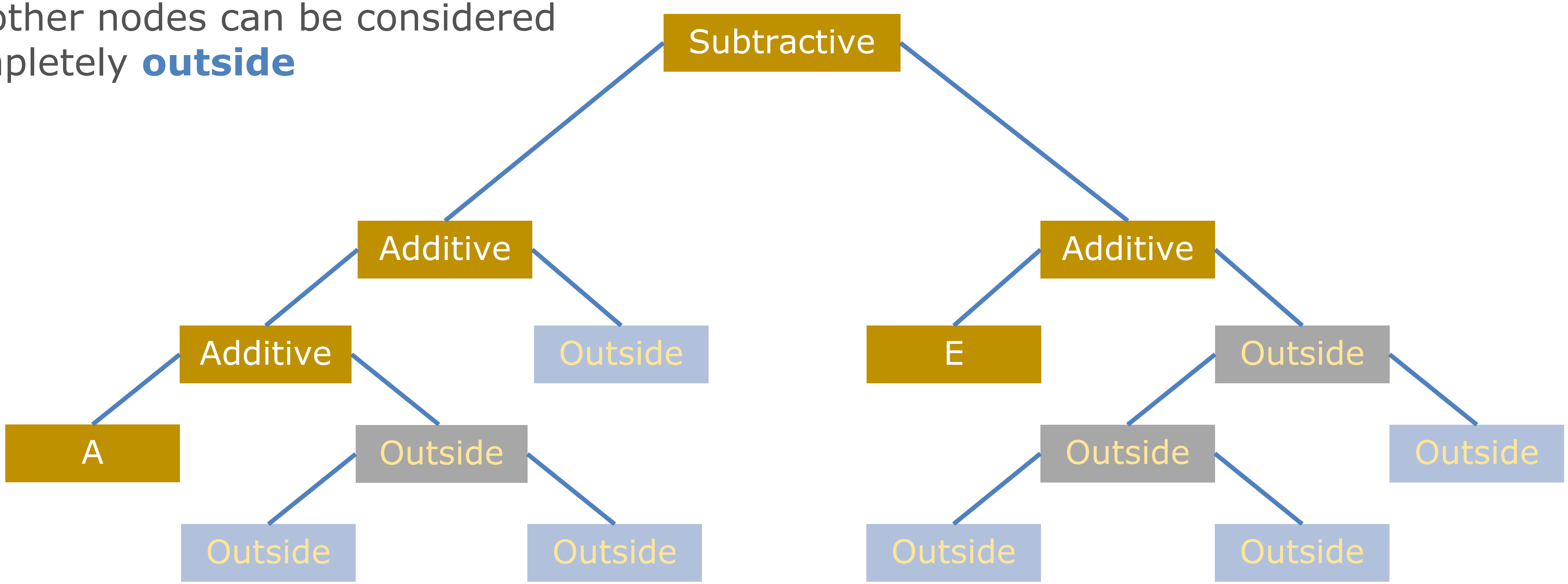
Per brush CSG Tree

Until we end up at the root of our CSG tree



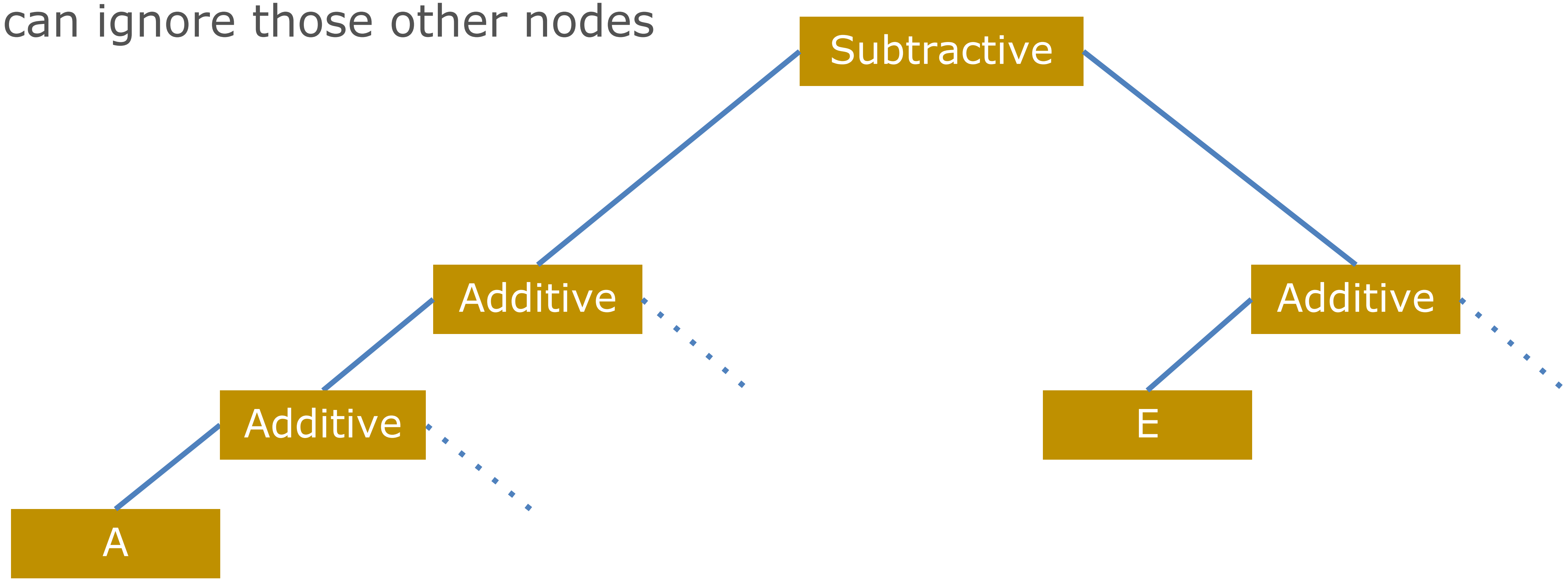
Per brush CSG Tree

All other nodes can be considered completely **outside**

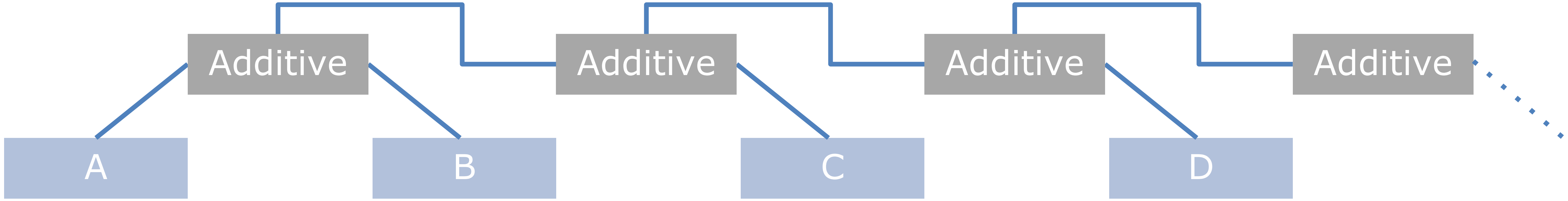


Per brush CSG Tree

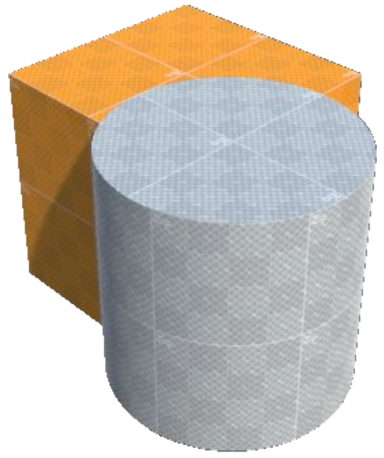
We can ignore those other nodes



Use a lookup table to combine categories among multiple brushes



Additive Operation Table		Left			
		Inside	Aligned	Rev-Aligned	Outside
Right	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside



GAME DEVELOPERS CONFERENCE
MARCH 16–20, 2020 | #GDC20

Boolean operations are **baked** into routing table

using the boolean lookup tables

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside
Brush C	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Aligned	Inside	Aligned
	Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
	Outside	Inside	Aligned	Rev-Aligned	Outside

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

In practice we use **numerical values** instead of symbolic names

Inside = 0
Aligned = 1
Rev-Aligned = 2
Outside = 3

The output of each row becomes an index to a row in the next section

Always be sure to keep the output of the last brush convertible back to categories

Polygon.index == **0**

Lookup **rows** using a category stored on our **polygon**

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Polygon.index == **0**

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

Each **brush** will categorize our polygon, and this category is the **column**

Polygon.index == **0**

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

Using the **brush category** we find the **column**

Using the **polygon index** we find the **row**

Polygon.index == 2

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

The **intersection** of the **column** and **row** leads to the **output index**

Polygon.index == **2**

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

We do this with every brush ...

Polygon.index == **0**

Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

We do this with every brush ...

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

Routing table

Polygon.index == **0**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Until we find the final index ...

Polygon.index == **0**
(final index)

Inside = 0
Aligned = 1
Rev-Aligned = 2
Outside = 3

The final index can be converted back to a category

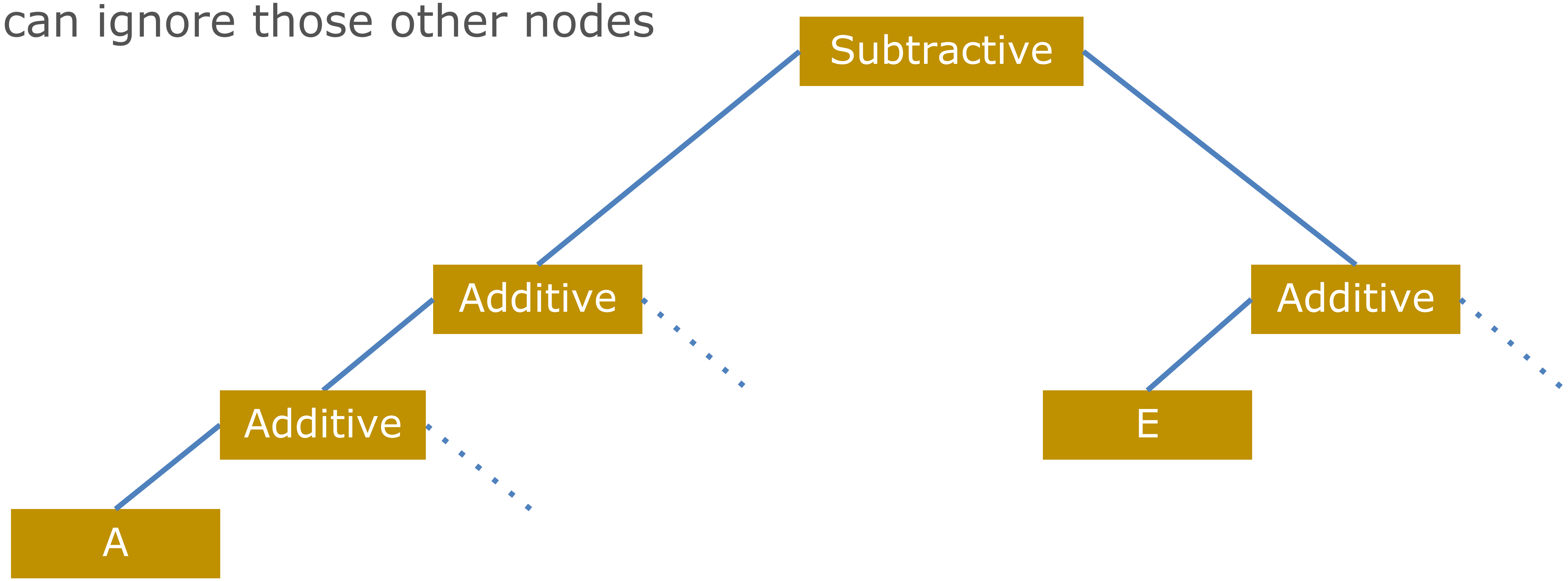
Routing table

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Example
Brush A: **Rev-Aligned**
Brush B: **Aligned**
Brush C: **Outside**

Per brush CSG Tree

We can ignore those other nodes

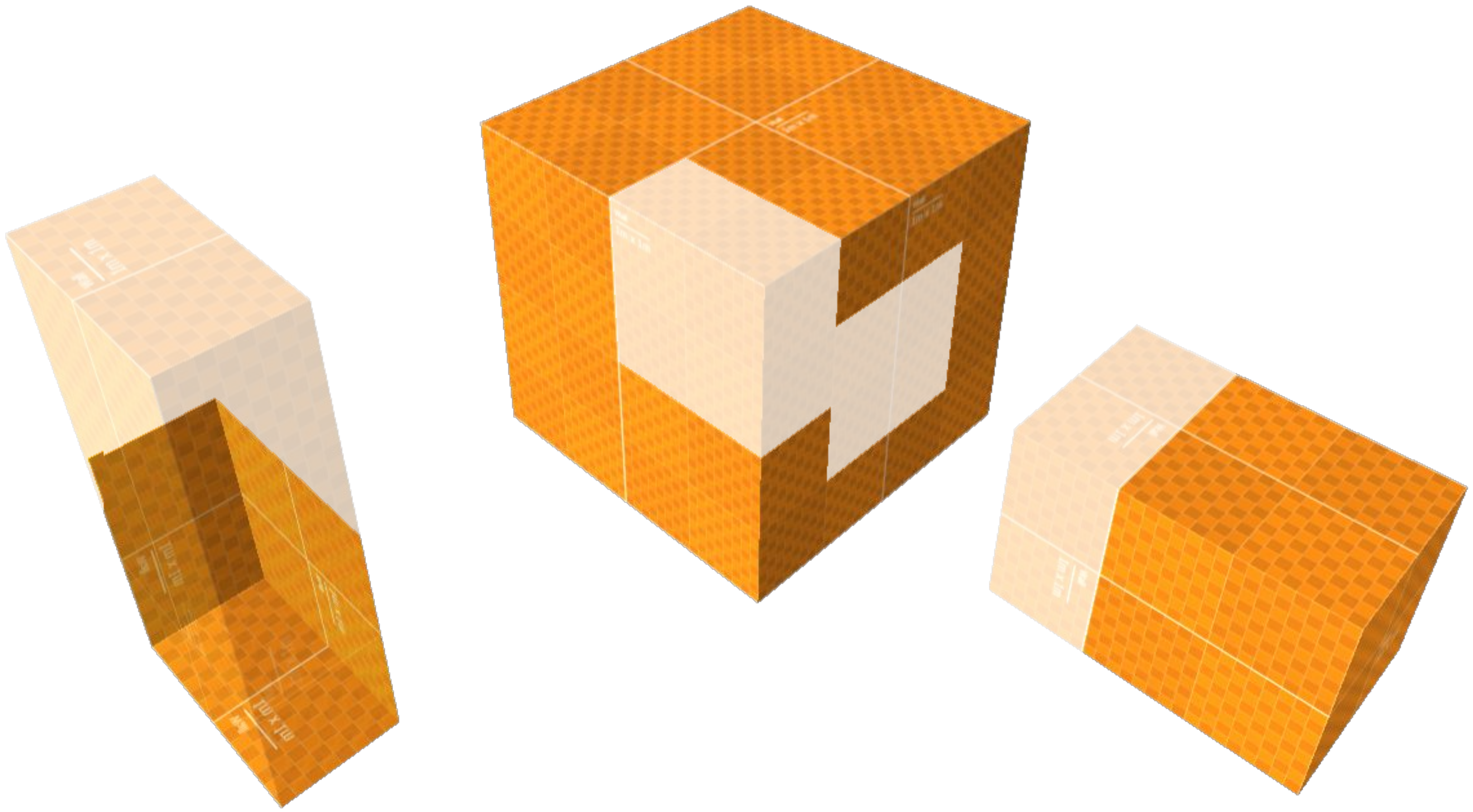


Routing tables are build **individually** for each brush

For example, this table is the routing table for **brush B**

Routing table **for brush B**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3



Which polygon piece is what,
to the ***entire generated mesh?***

Routing table **for brush B**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

All categories of the brush this routing table is made for will always be **aligned**

Brush B can be **optimized** away

Routing table **for brush B**

All categories of the brush this routing table is made for will always be **aligned**

Brush B can be **optimized** away

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	2	3
~~Brush B~~	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Routing table **for brush B**

	Polygon Index	Inside	Aligned	Rev Aligned	Outside
Brush A	0	0	1	0	1
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

All outputs of **brush B** Are rewired to the output of **brush A**

This allows us to **remove brush B**

Not all outputs of **brush A** lead to **brush C**

We can remove outputs that we'll never use

Routing table **for brush B**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	0	1
Brush C	0	0	0	0	0
	1	0	1	0	1
	~~2~~	0	0	2	2
	~~3~~	0	1	2	3

Routing table **for brush B**

Not all outputs of **brush A** lead to **brush C**

We can remove outputs that we'll never use

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	0	1
Brush C	0	0	0	0	0
	1	0	1	0	1

Routing table **for brush B**

When optimizing routing tables make sure all indices are **sequential** and **start with 0**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	1	3	1	3
Brush B	1	0	1	0	1
	3	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush D	0	0	0	0	0
	1	0	1	0	1

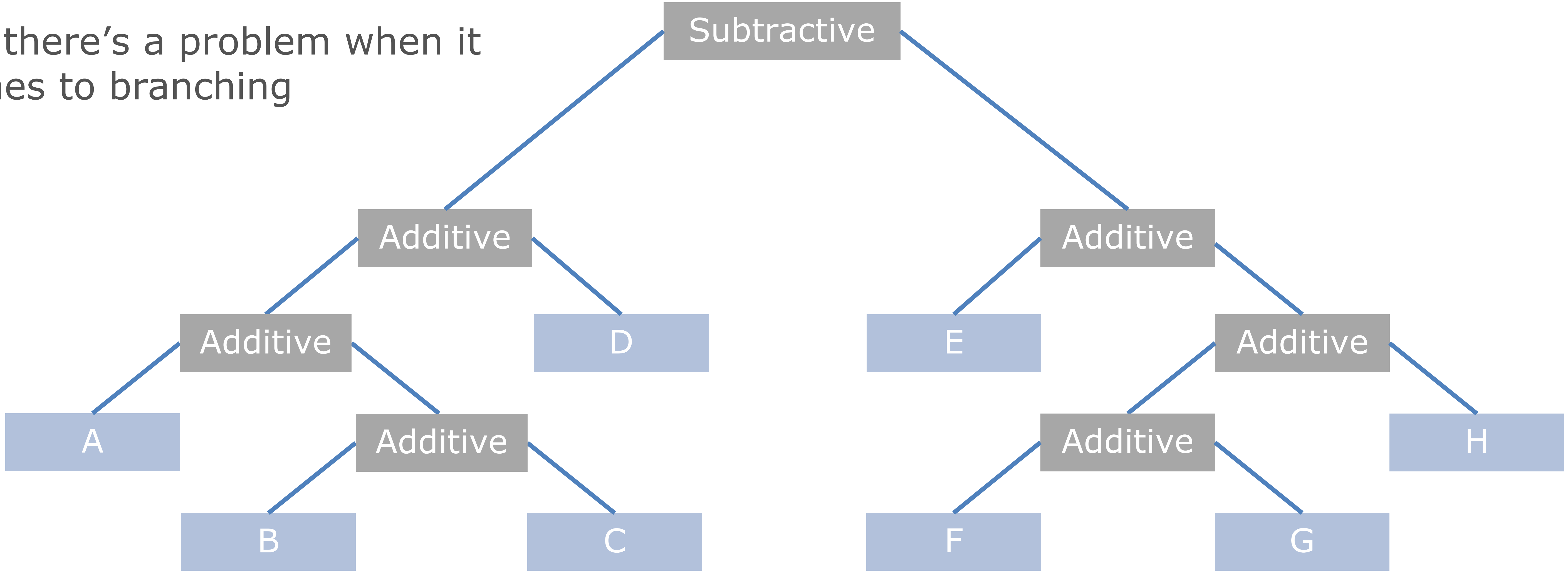
Routing table **for brush B**

Remap outputs of previous brush to new rows to **fix gaps**

	Polygon Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	0	0	1	0	1
Brush B	0	0	1	0	1
	1	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
Brush D	0	0	0	0	0
	1	0	1	0	1

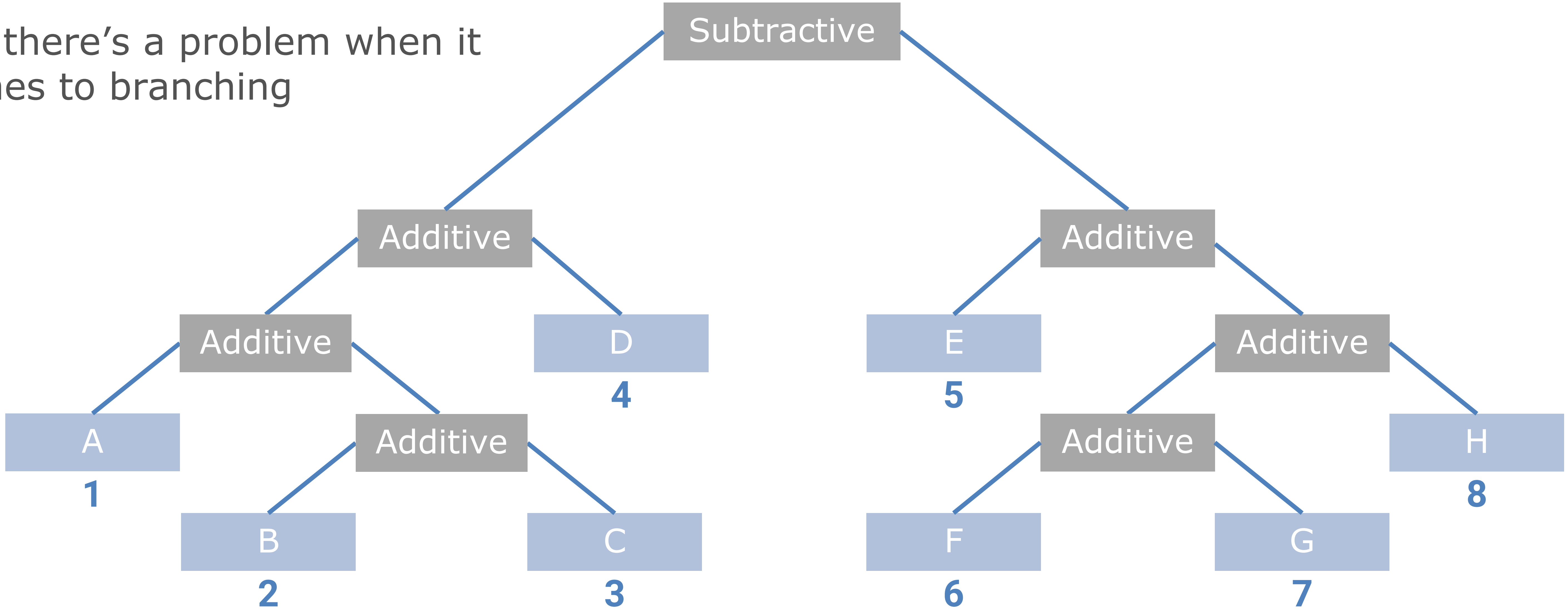
Branching

But there’s a problem when it comes to branching



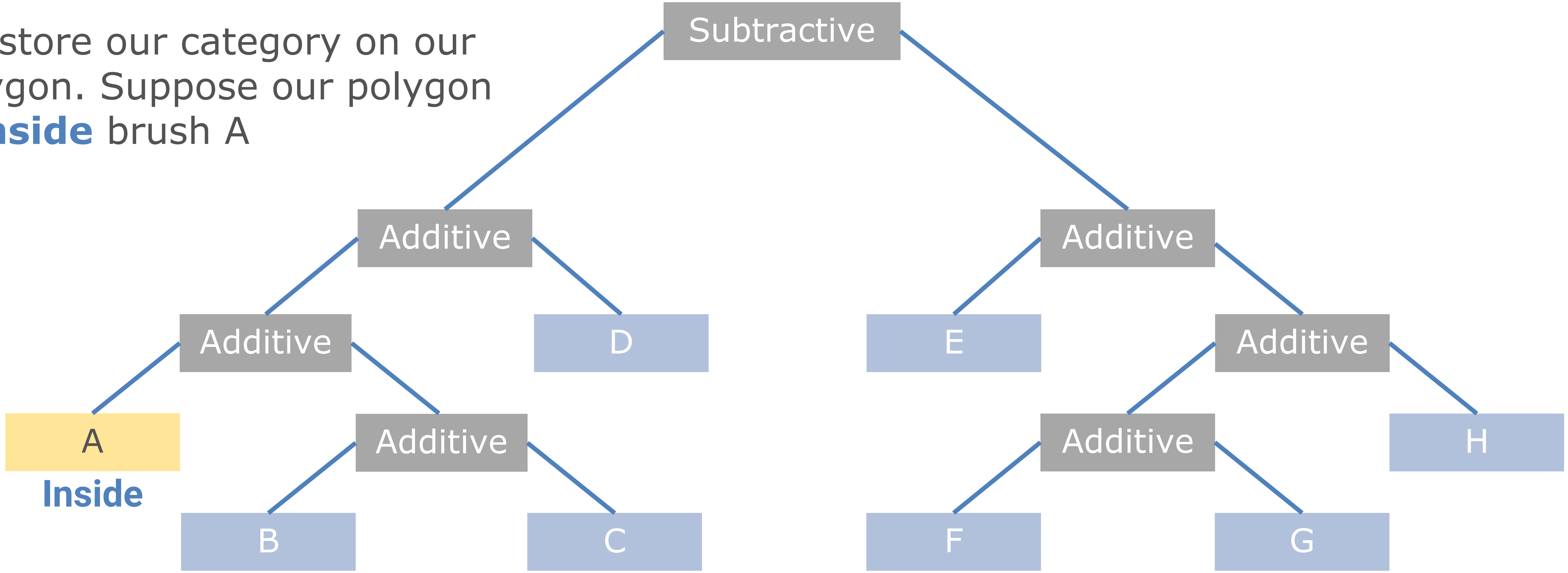
Branching

But there’s a problem when it comes to branching



Branching

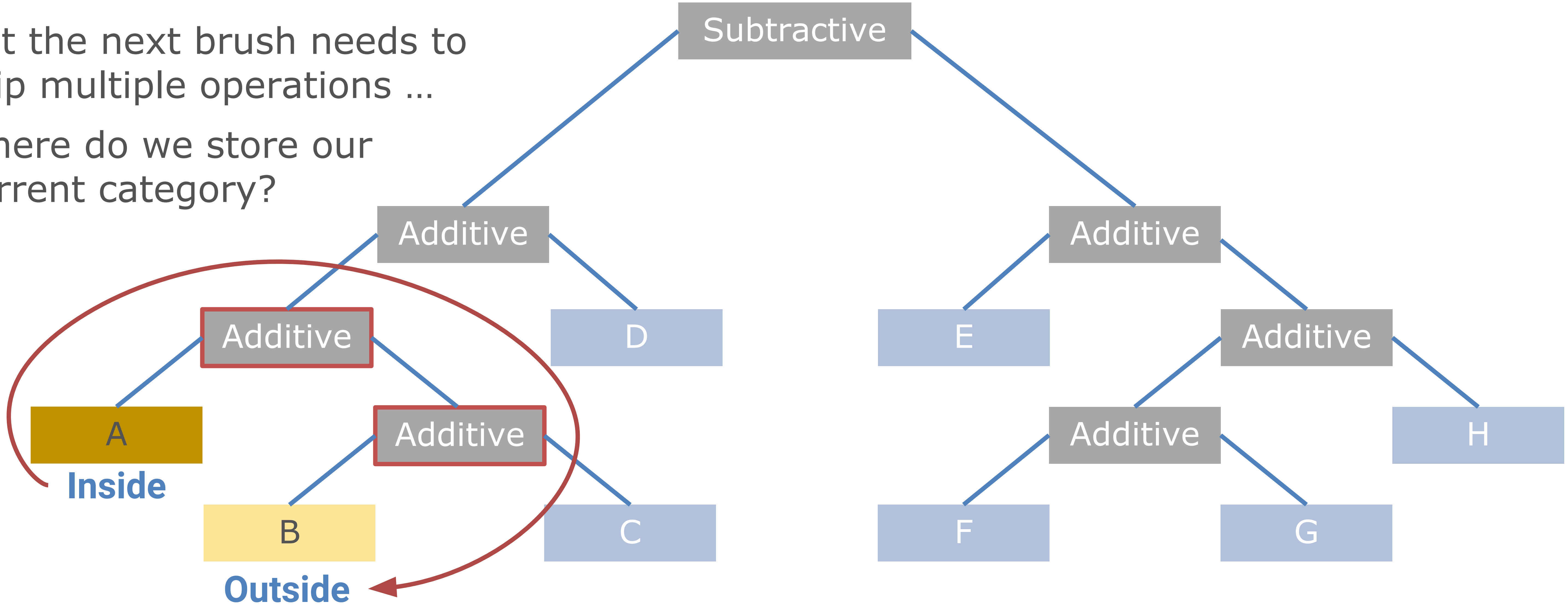
We store our category on our polygon. Suppose our polygon is **inside** brush A



Branching

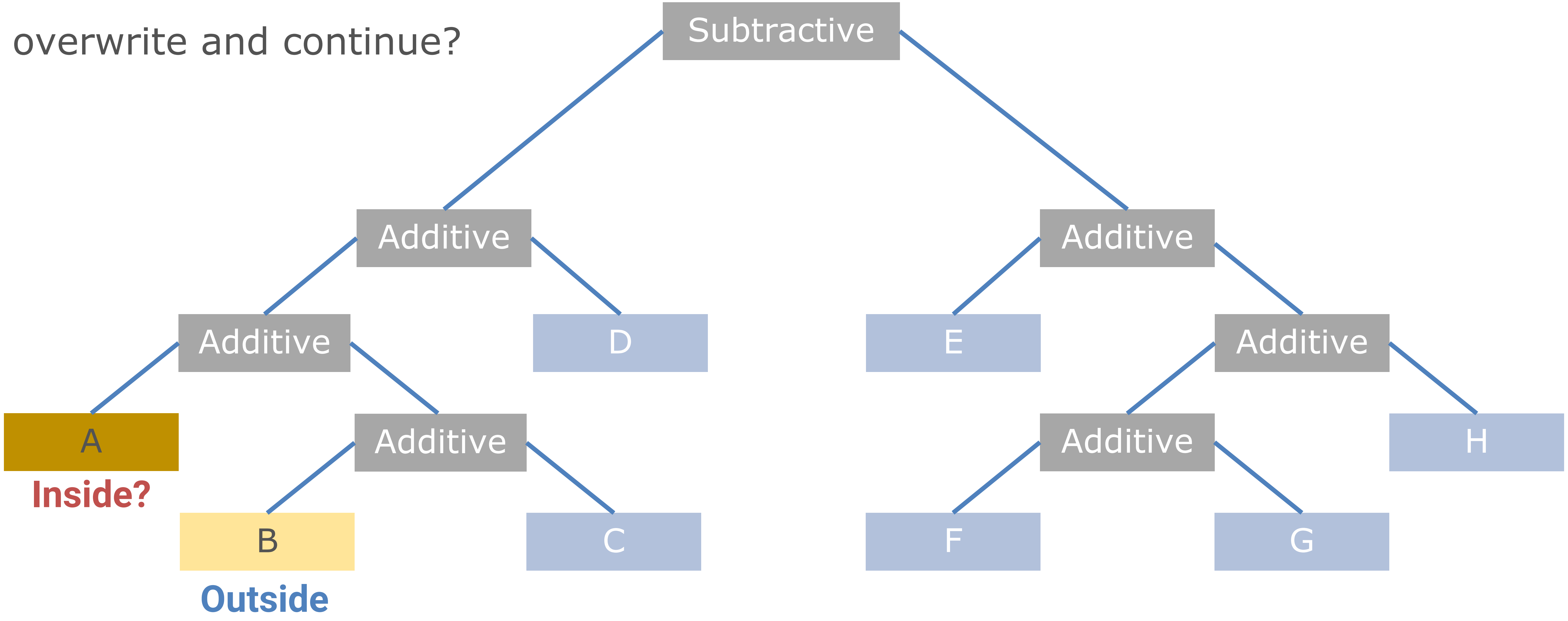
But the next brush needs to skip multiple operations …

Where do we store our current category?



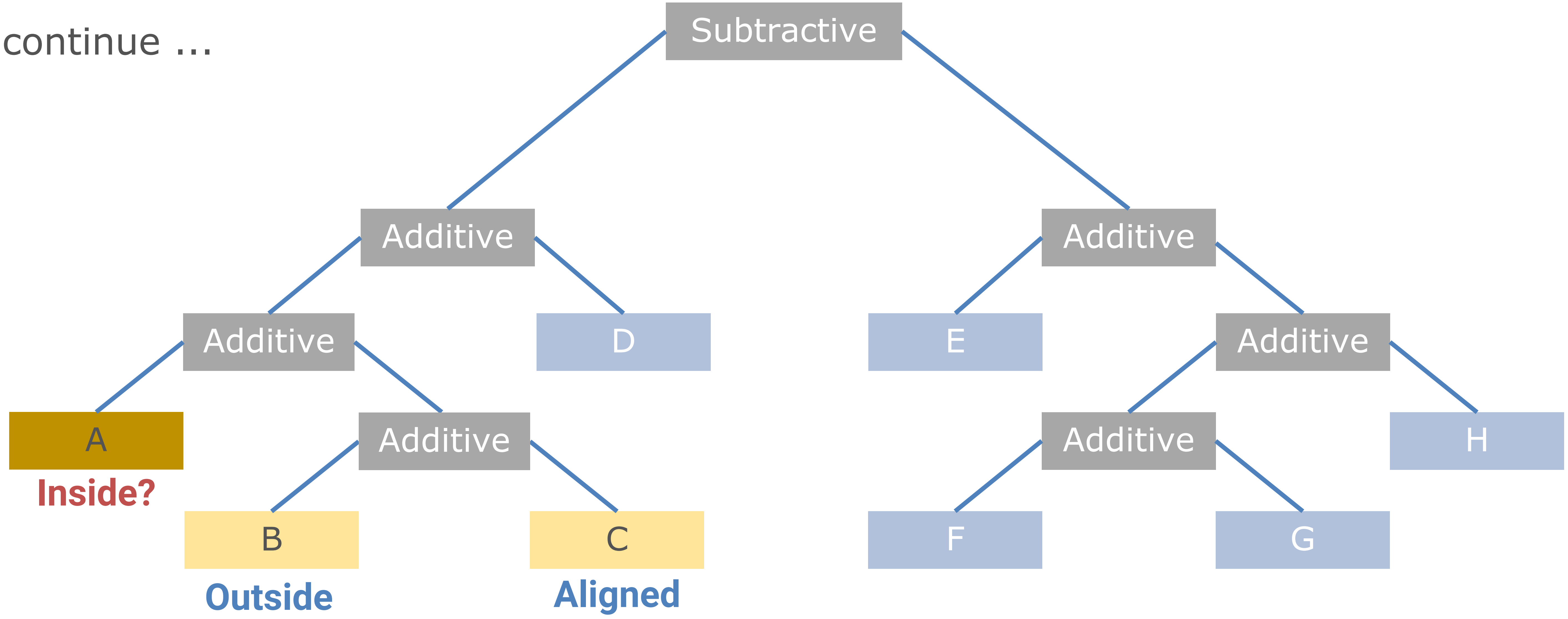
Branching

Just overwrite and continue?



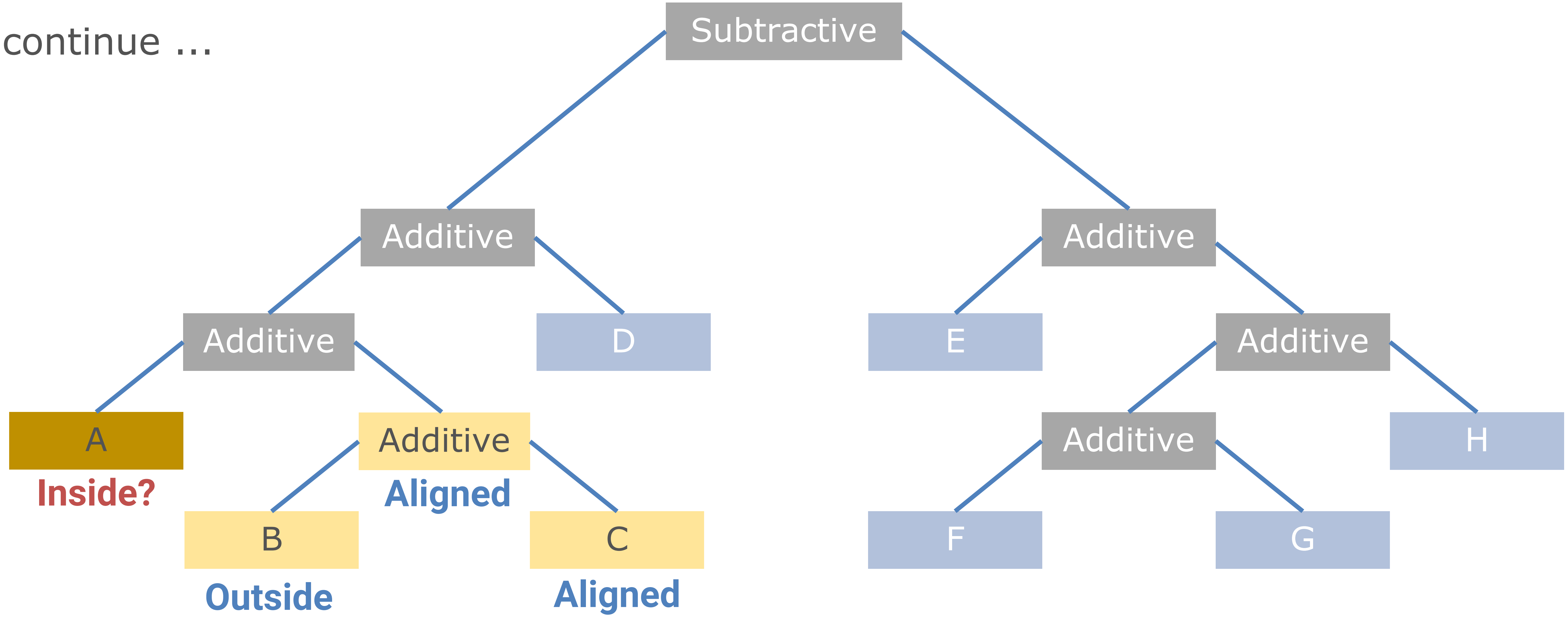
Branching

We continue ...



Branching

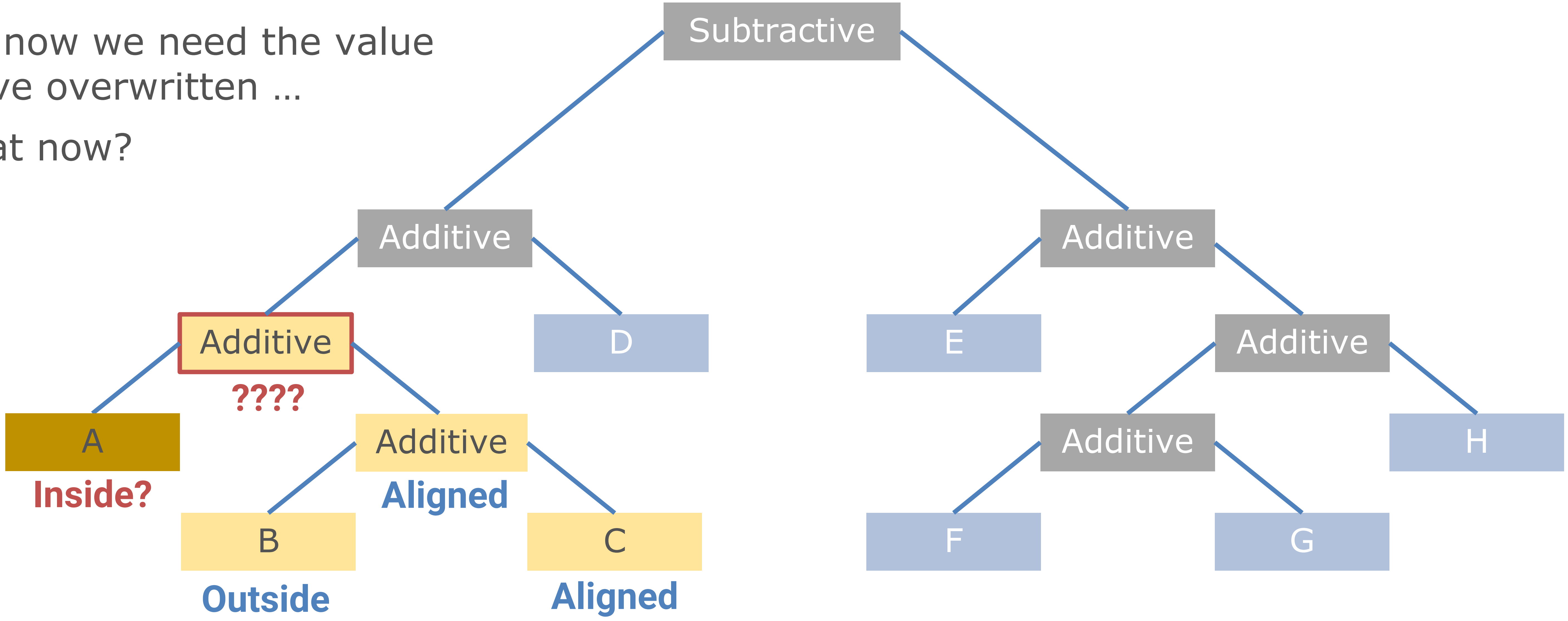
We continue …



Branching

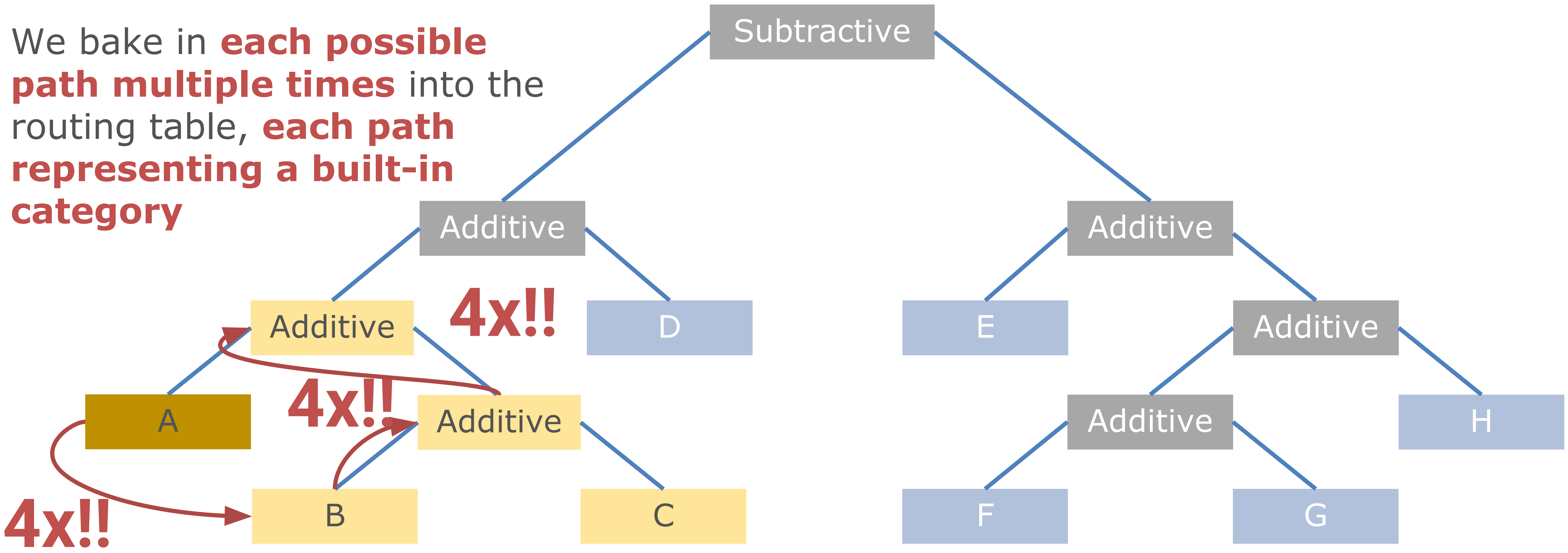
But now we need the value we've overwritten …

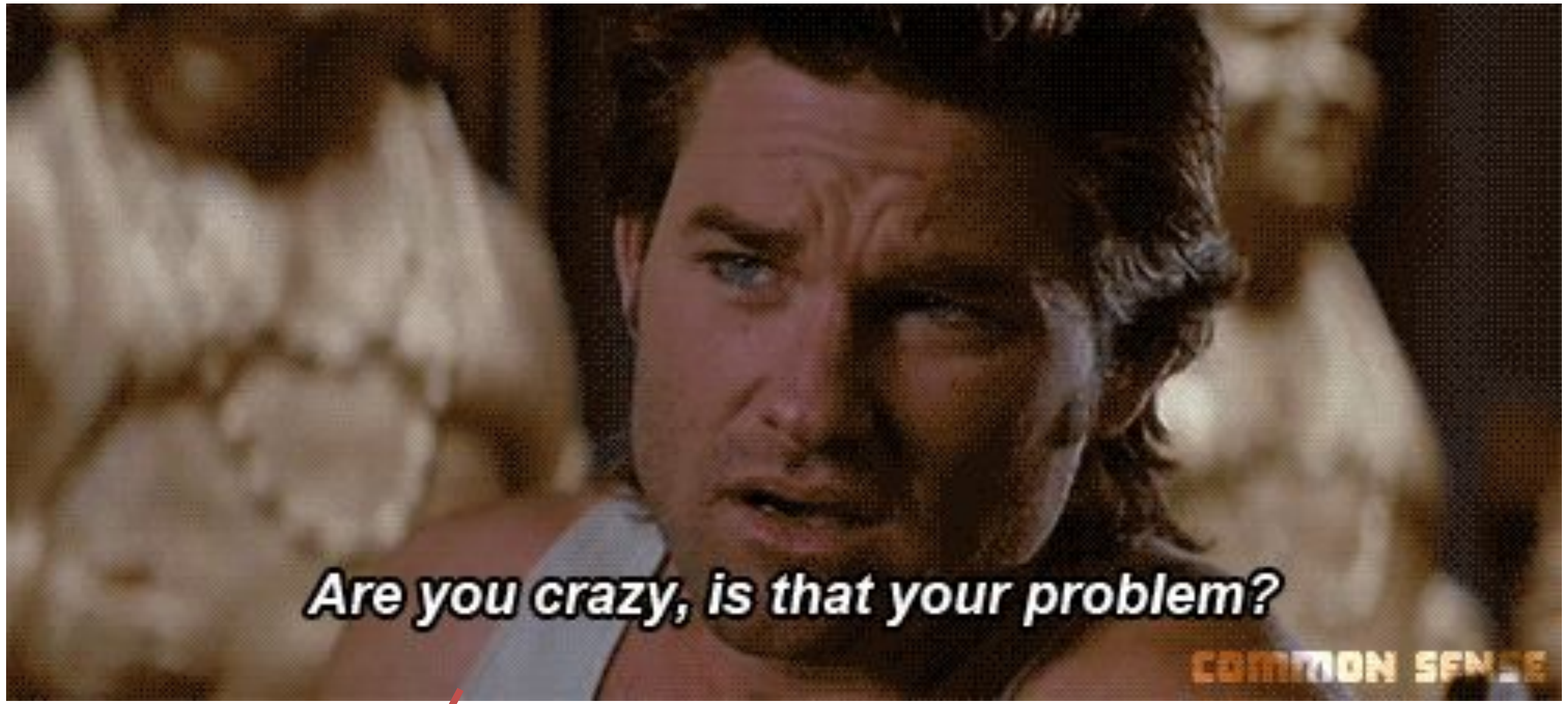
What now?



Branching

We bake in **each possible path multiple times** into the routing table, **each path representing a built-in category**

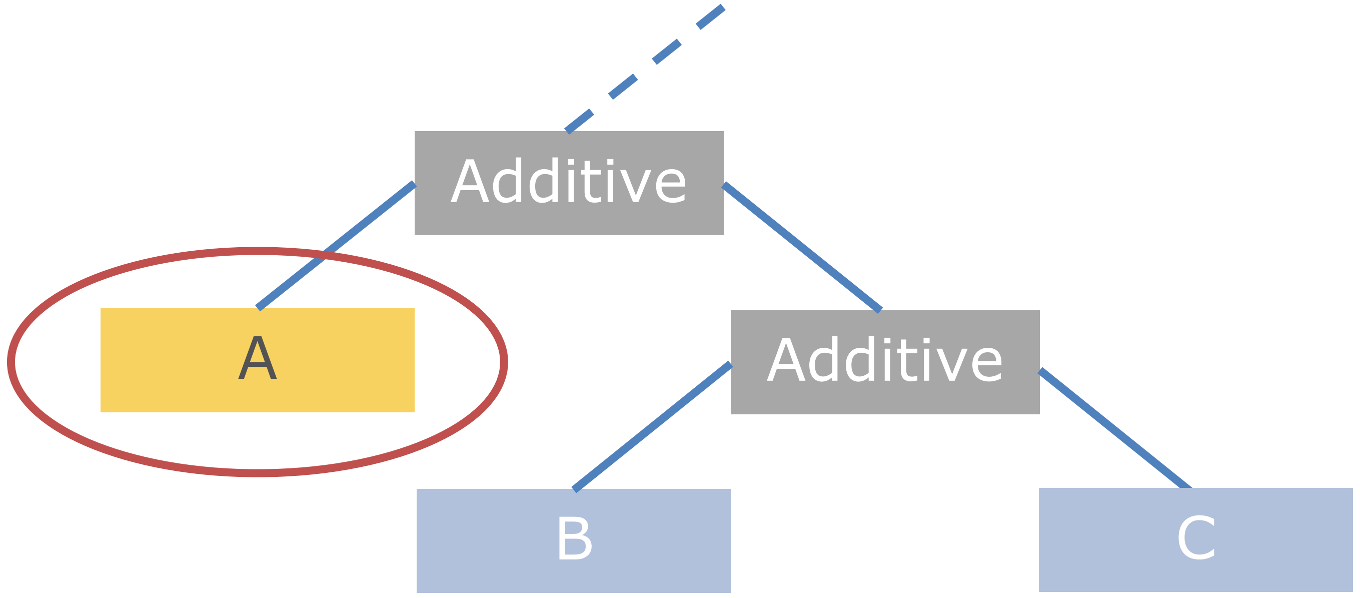



Are you crazy, is that your problem?

It’s not as bad as it may seem™

Routing table

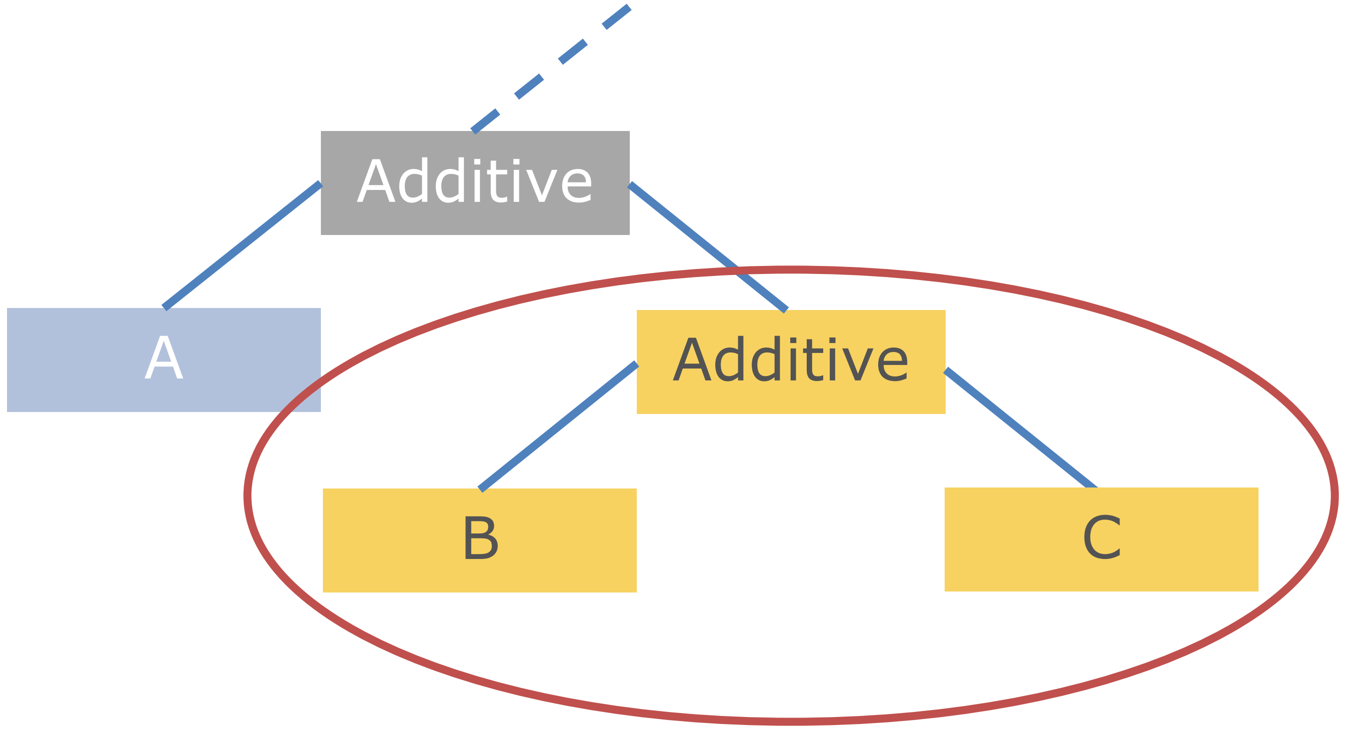
	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3



Routing table

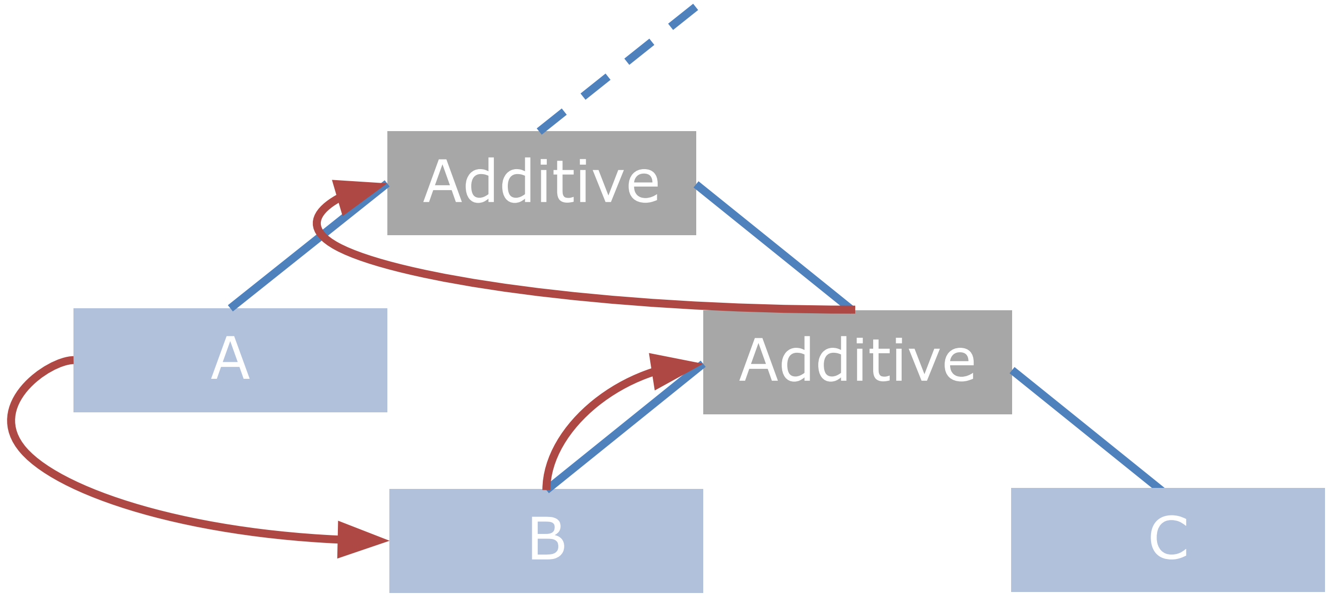
	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush B	0	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3



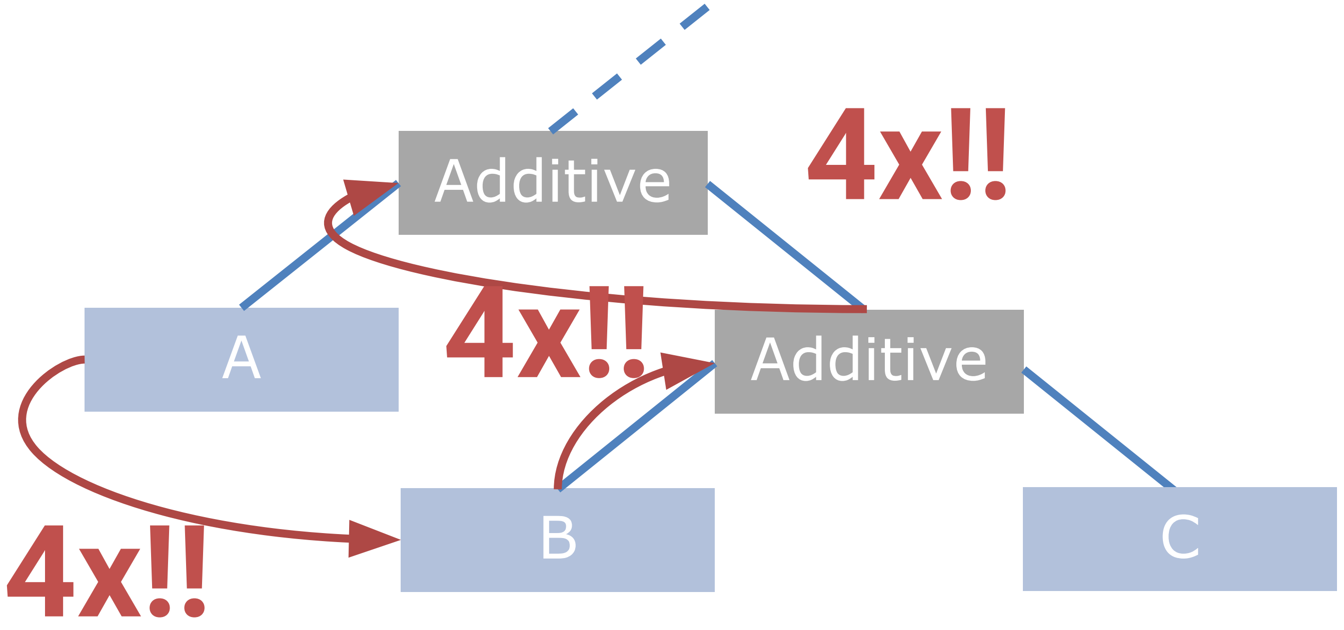
Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	1	2	3
Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3



Routing table

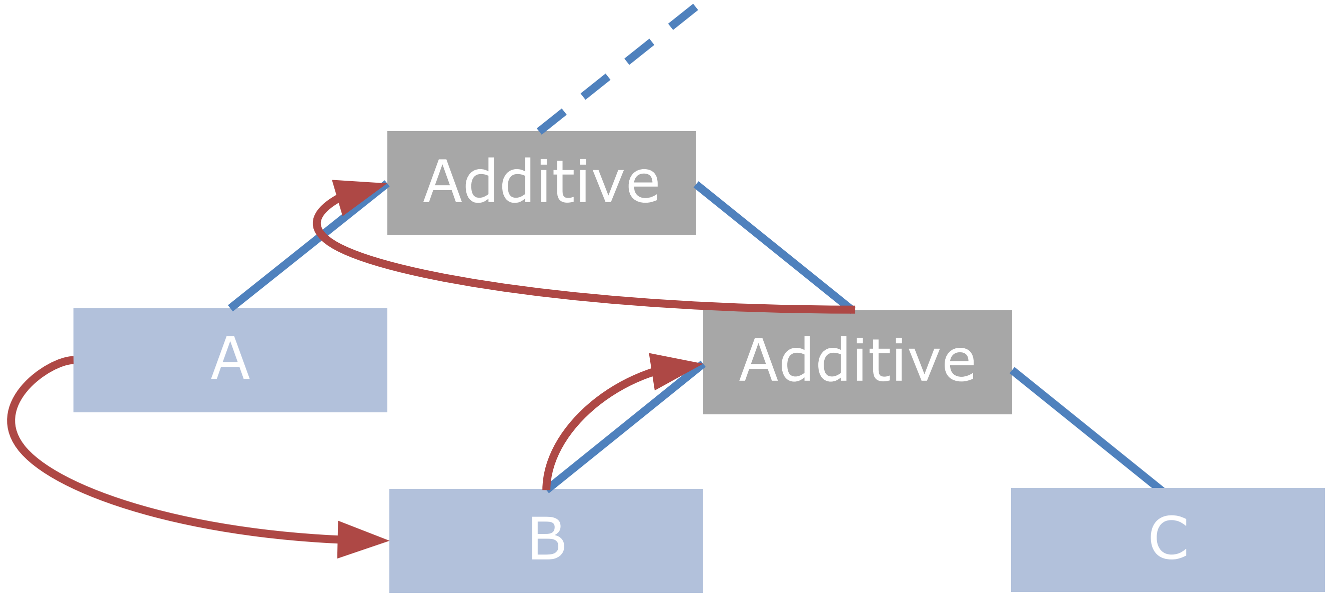
		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)						
(Rev-Aligned)						
(Outside)						
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)						
(Rev-Aligned)						
(Outside)						



Duplicate all rows of all brushes in second routing table, once for each category

Routing table

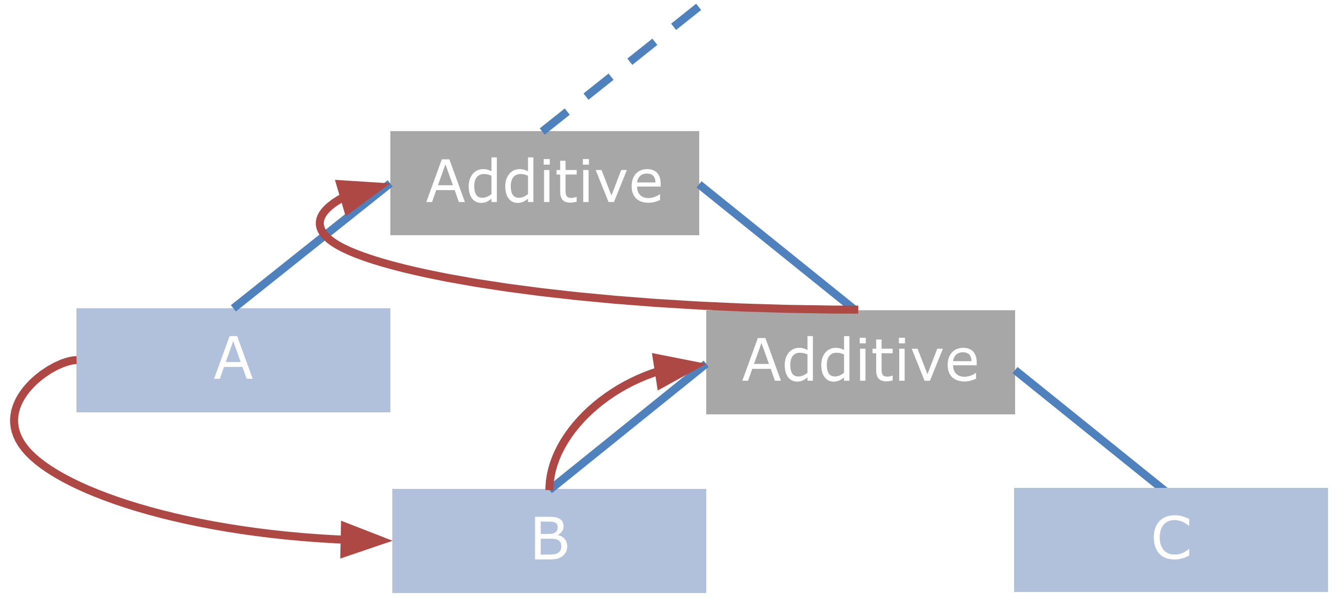
	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	0	0	1	2	3
(Rev-Aligned)	0	0	1	2	3
(Outside)	0	0	1	2	3
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)					
(Rev-Aligned)					
(Outside)					



Duplicate all rows of all brushes in second routing table, once for each category

Routing table

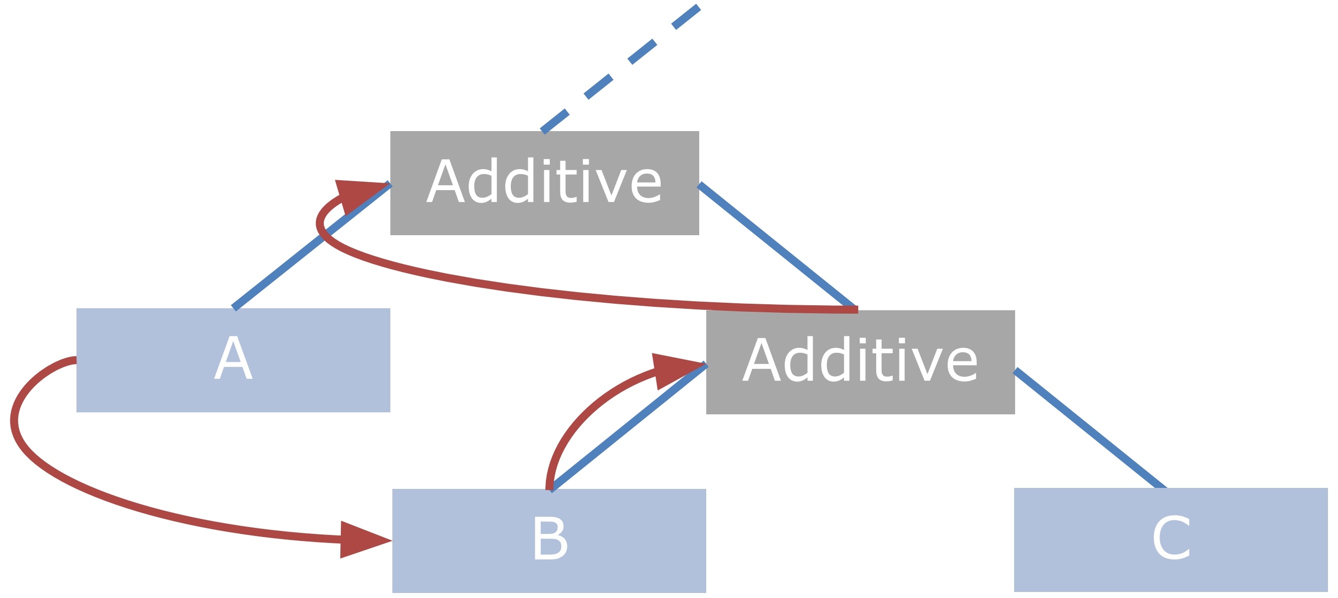
		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	0	1	2	3
(Rev-Aligned)		2	0	1	2	3
(Outside)		3	0	1	2	3
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)						
(Rev-Aligned)						
(Outside)						



Give each row an unique index

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)					
(Rev-Aligned)					
(Outside)					



Make every output unique,
add 4 for each duplicated row

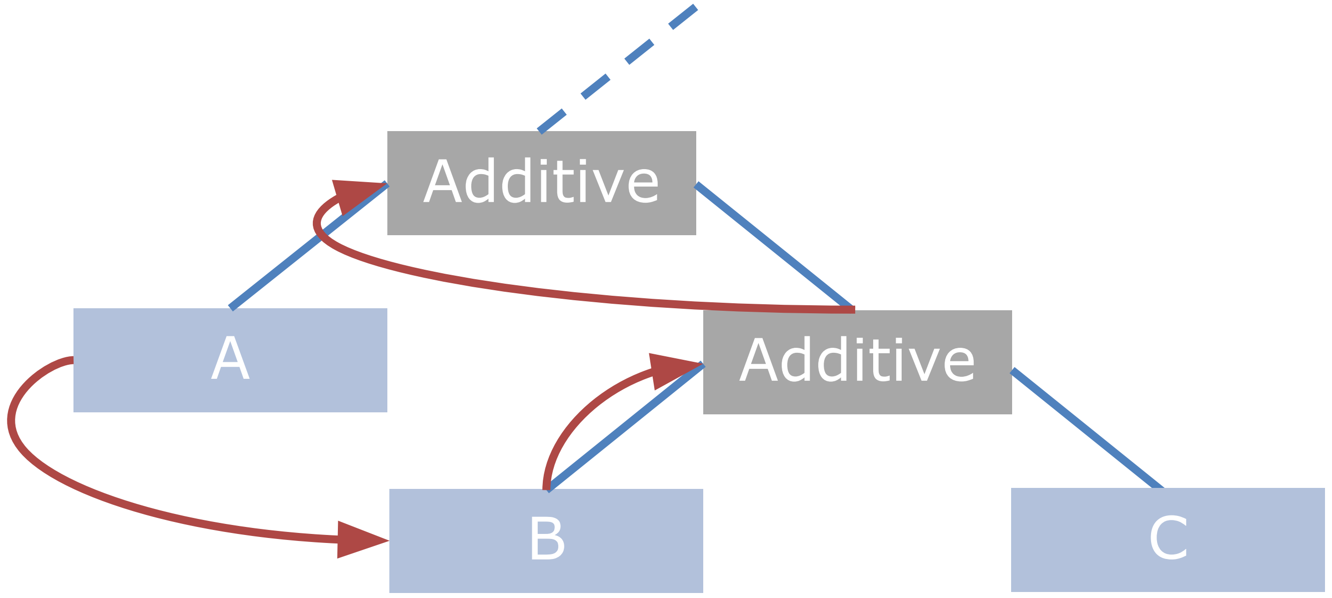
This gives is unique sequential
values for all outputs

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

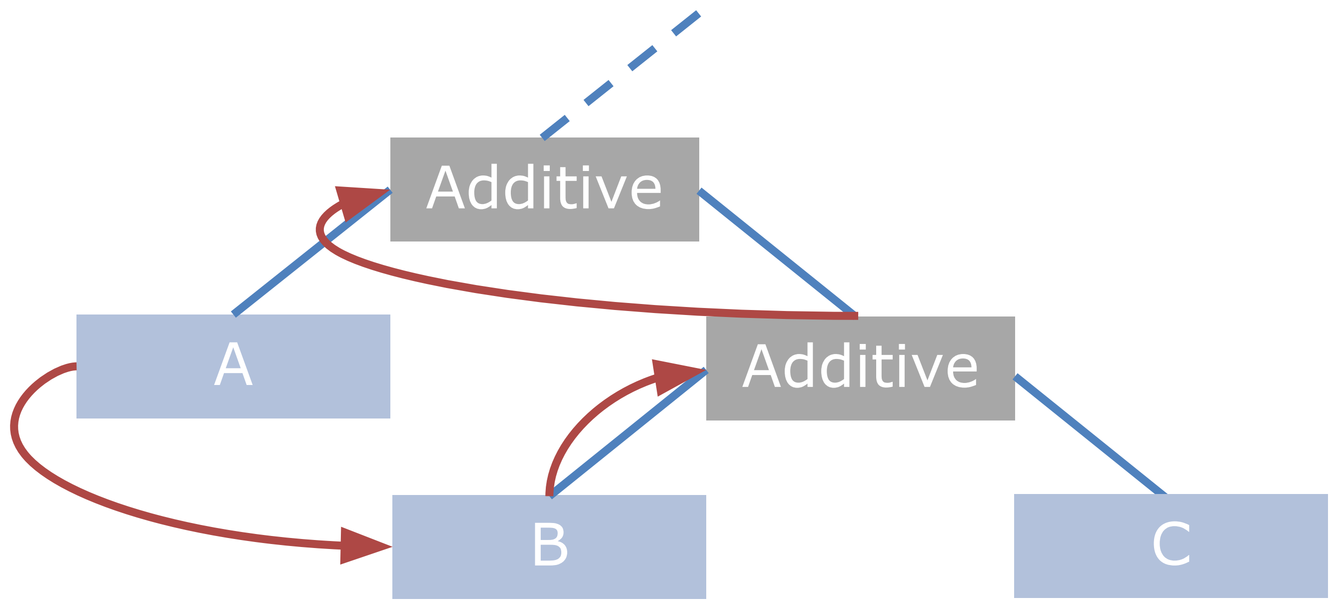
last brush in table

final output
(don't modify output values)



Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3

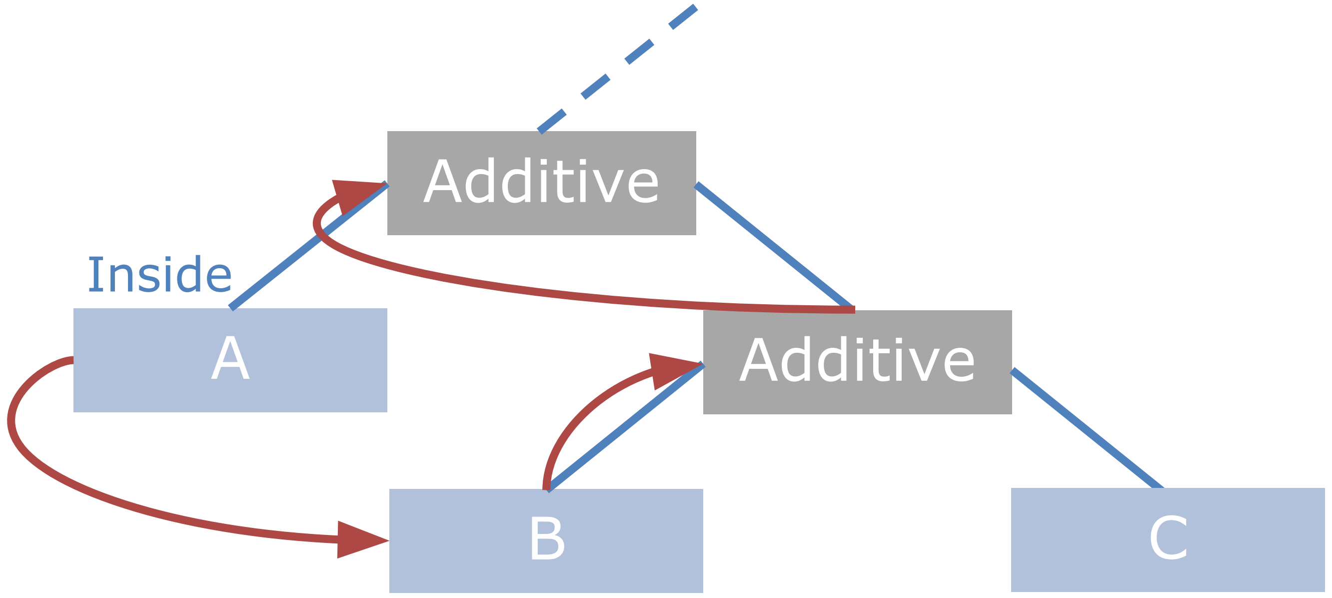


Each path from brush A represents a brush A category

GAME DEVELOPERS CONFERENCE
MARCH 16–20, 2020 | #GDC20

Routing table

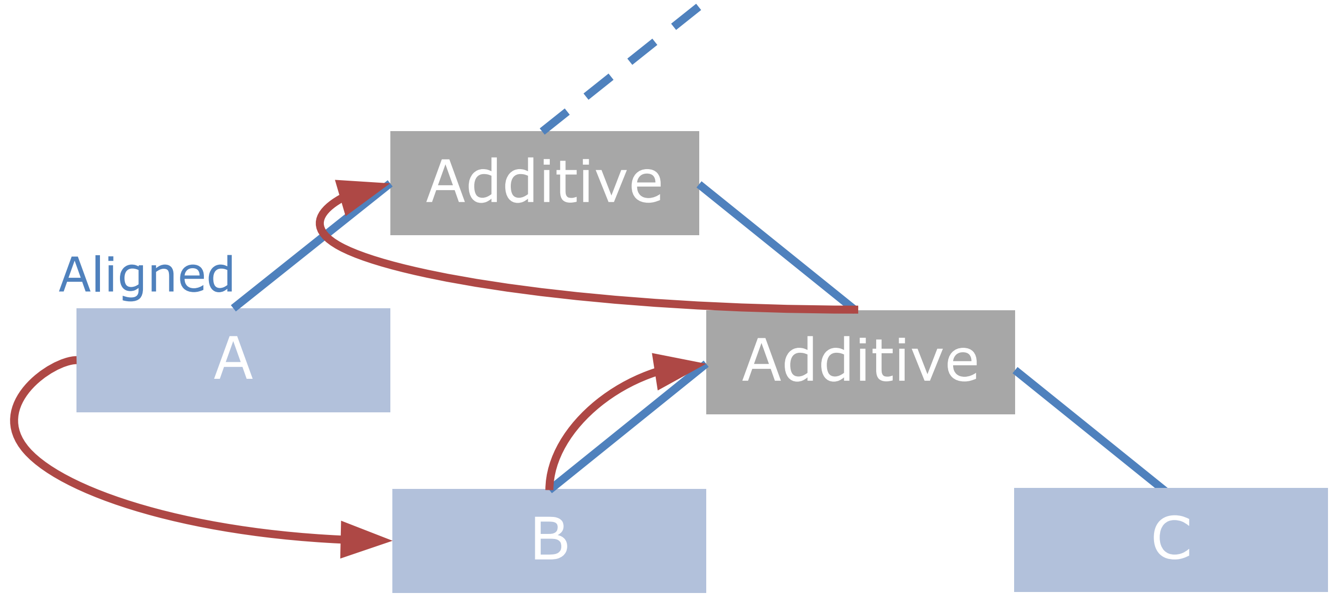
		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3



Each path from brush A represents a brush A category

Routing table

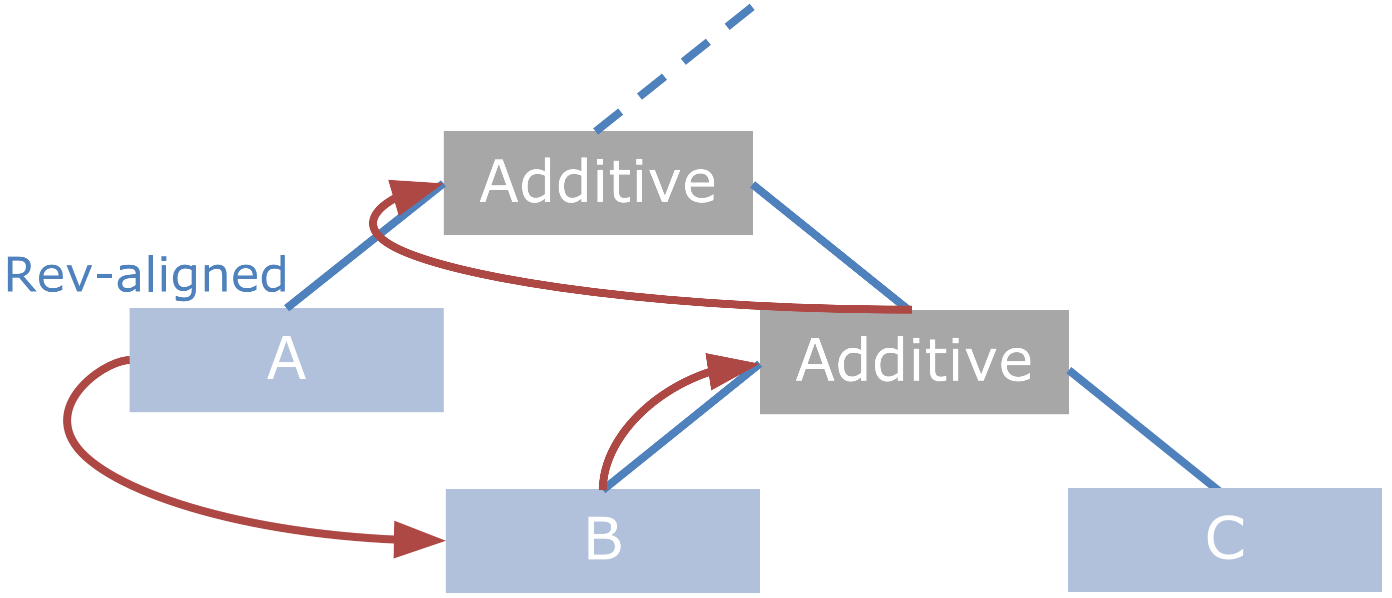
	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3



Each path from brush A represents a brush A category

Routing table

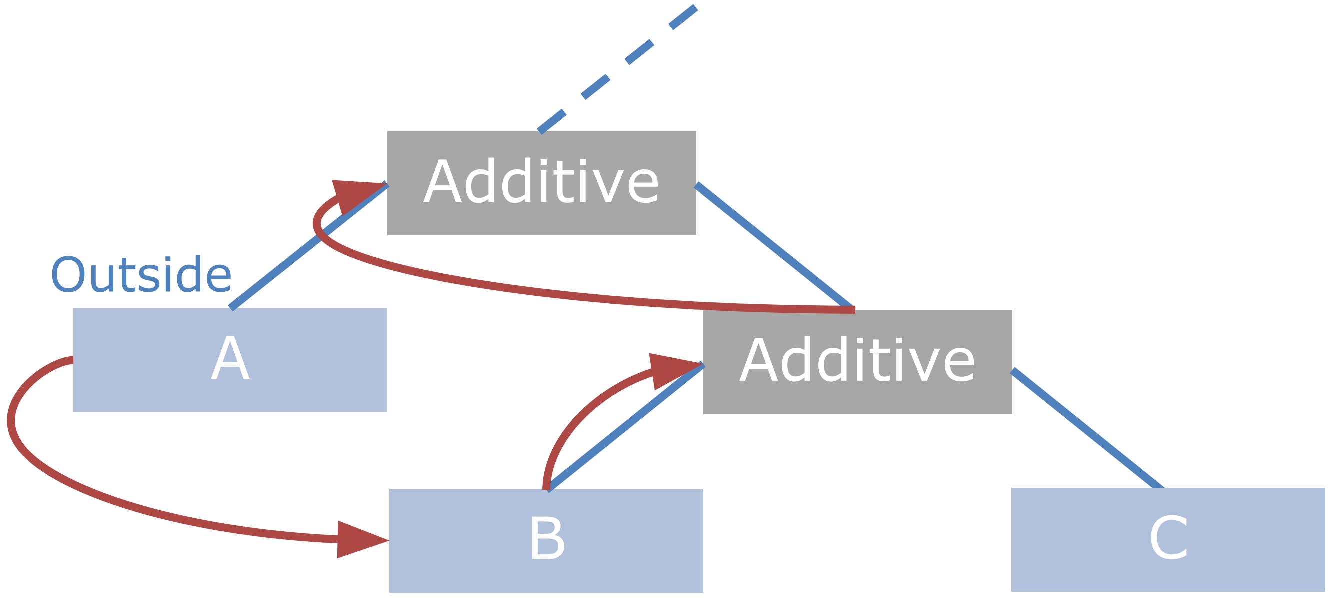
		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3



Each path from brush A represents a brush A category

Routing table

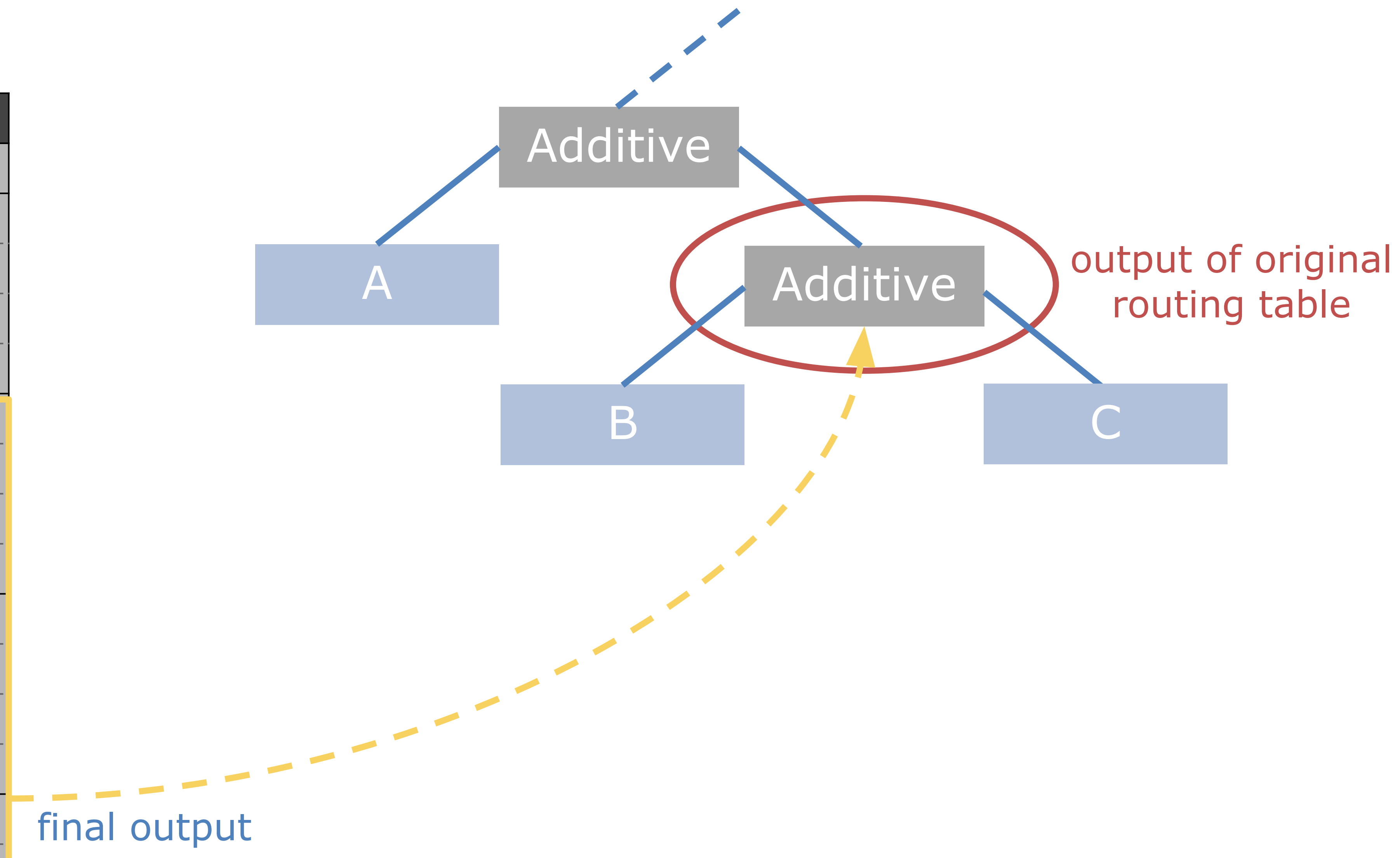
	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3



Each path from brush A represents a brush A category

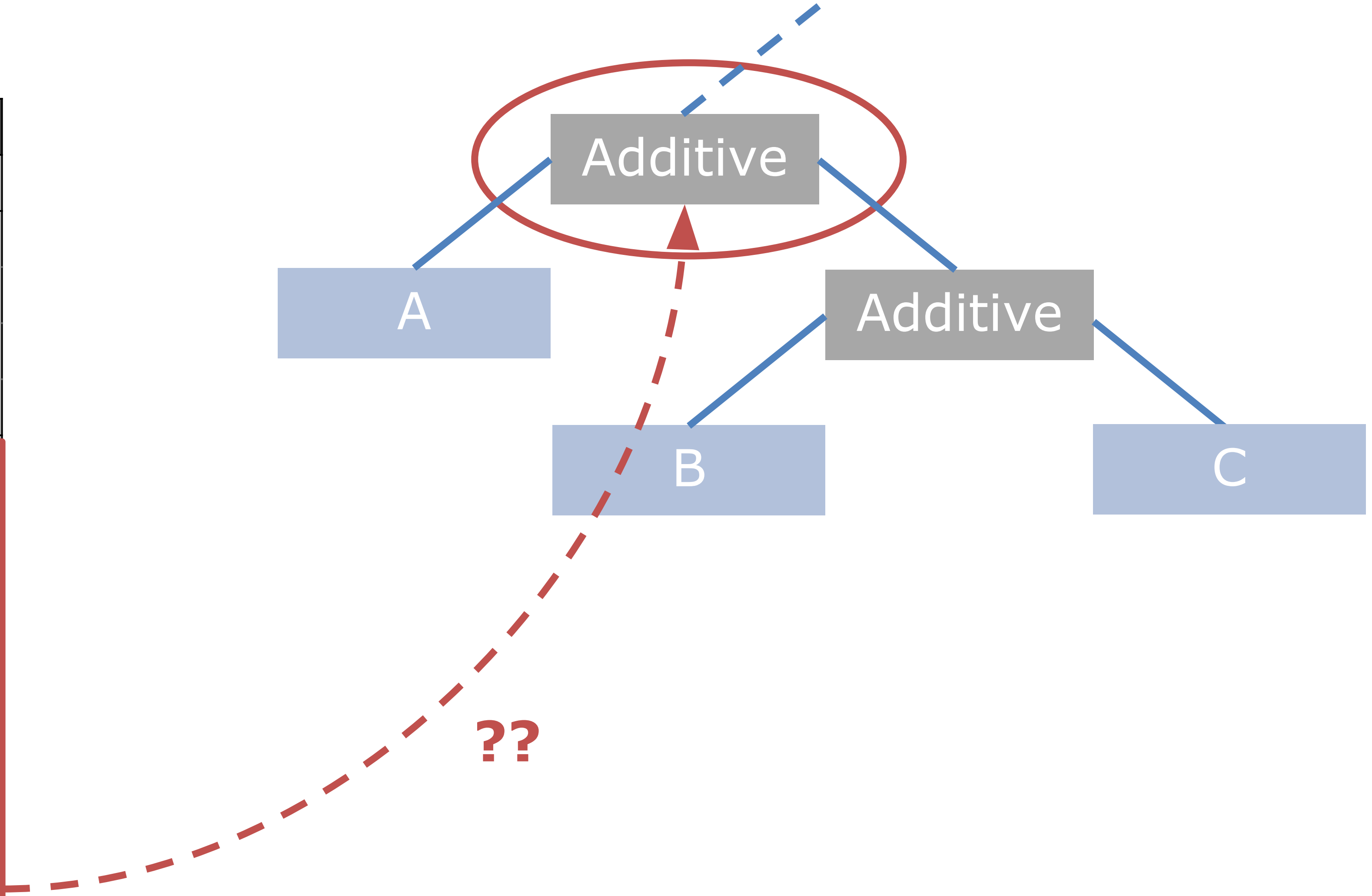
Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3



Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

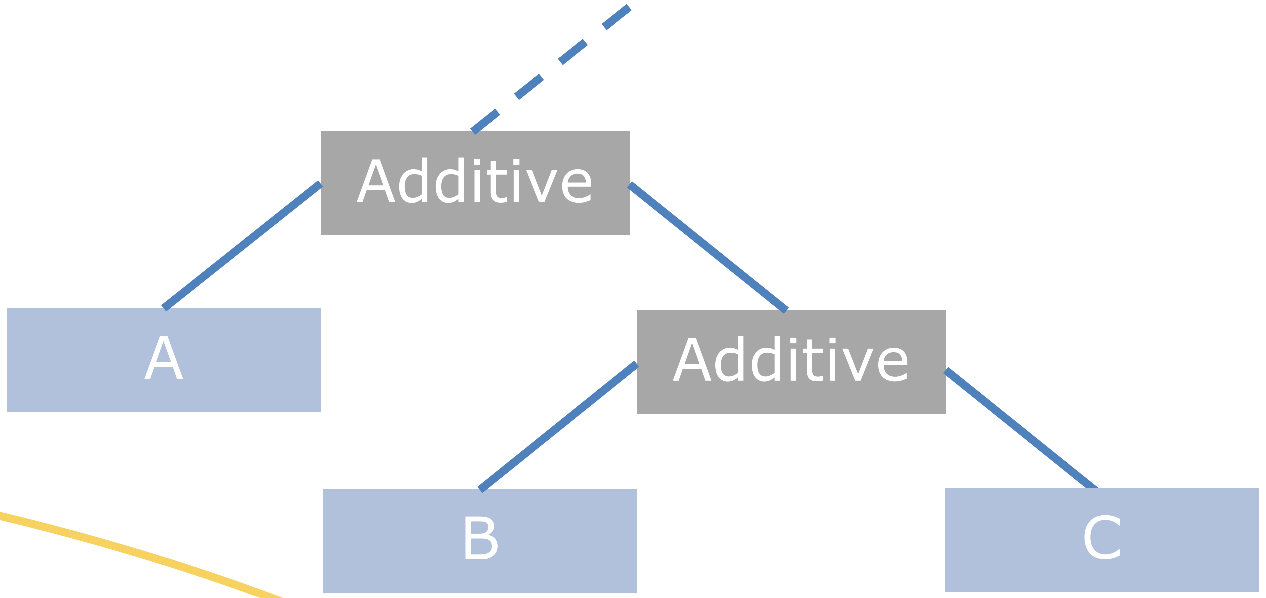


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
Inside	Inside	Inside	Inside	Inside
Aligned	Inside	Aligned	Inside	Aligned
Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
Outside	Inside	Aligned	Rev-Aligned	Outside

Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3

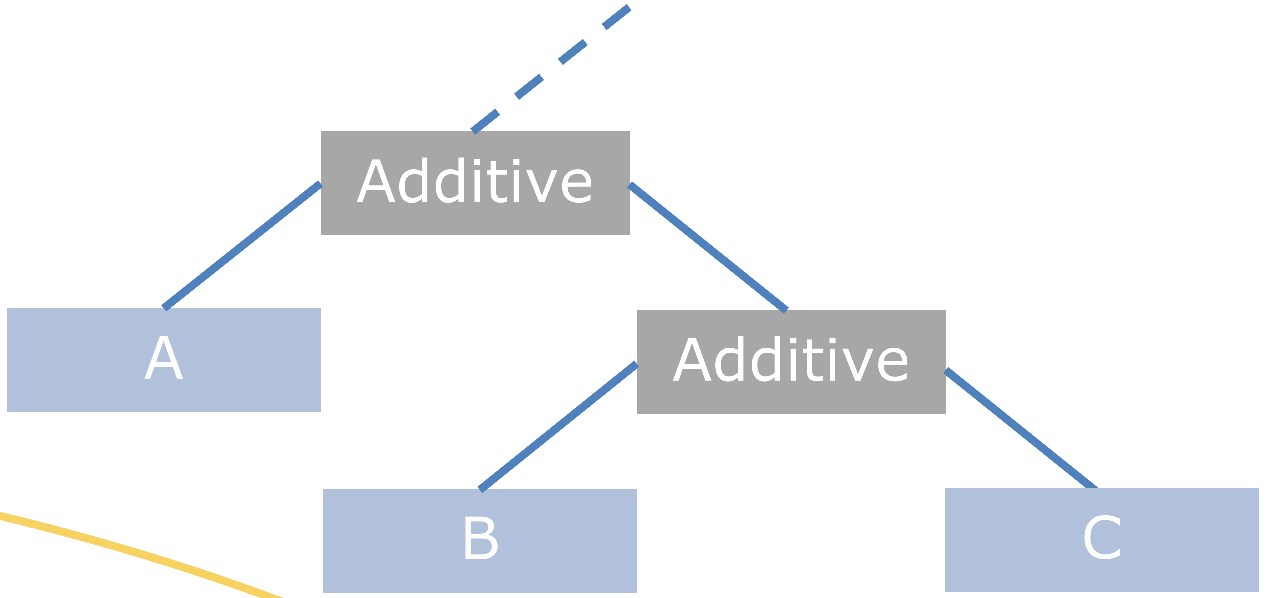


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
Inside	Inside	Inside	Inside	Inside
Aligned	Inside	Aligned	Inside	Aligned
Rev-Aligned	Inside	Inside	Rev-Aligned	Rev-Aligned
Outside	Inside	Aligned	Rev-Aligned	Outside

Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3

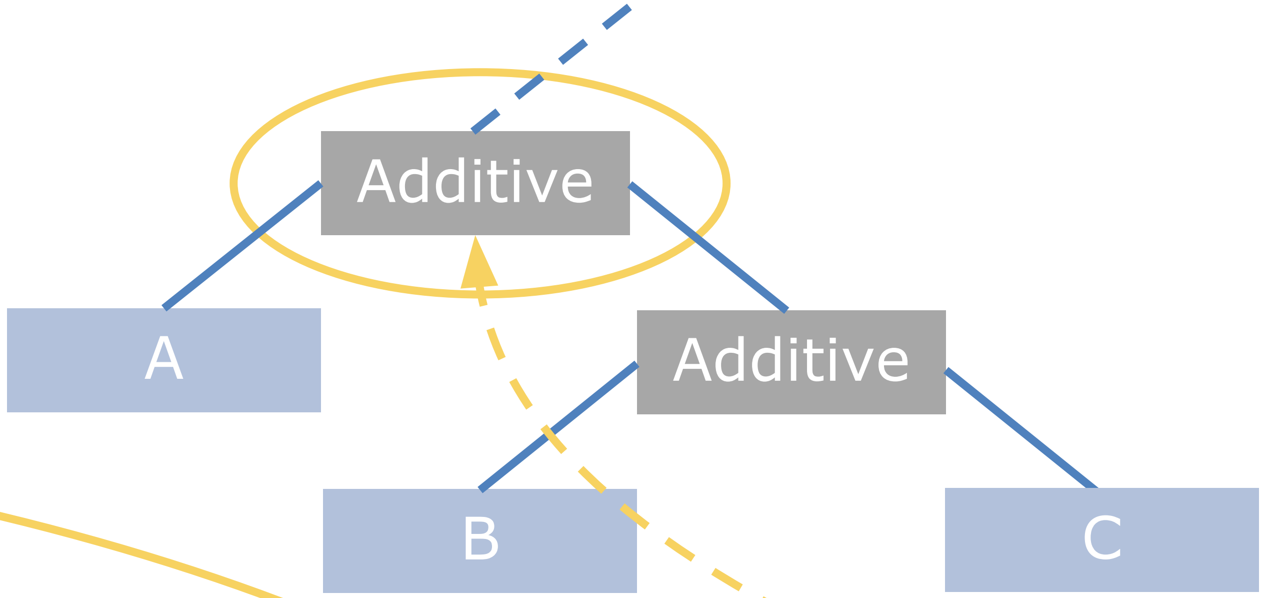


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	1	0	0
		2	0	0	2	2
		3	0	1	2	3
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3



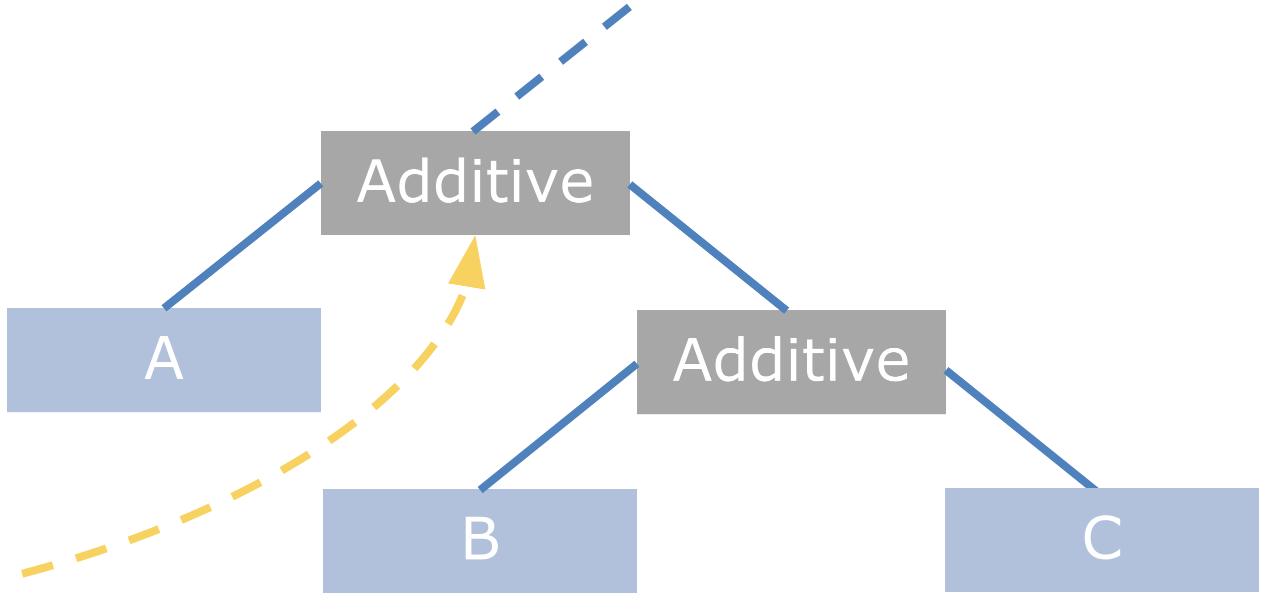
final output

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Additive operation table

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3



final output

write

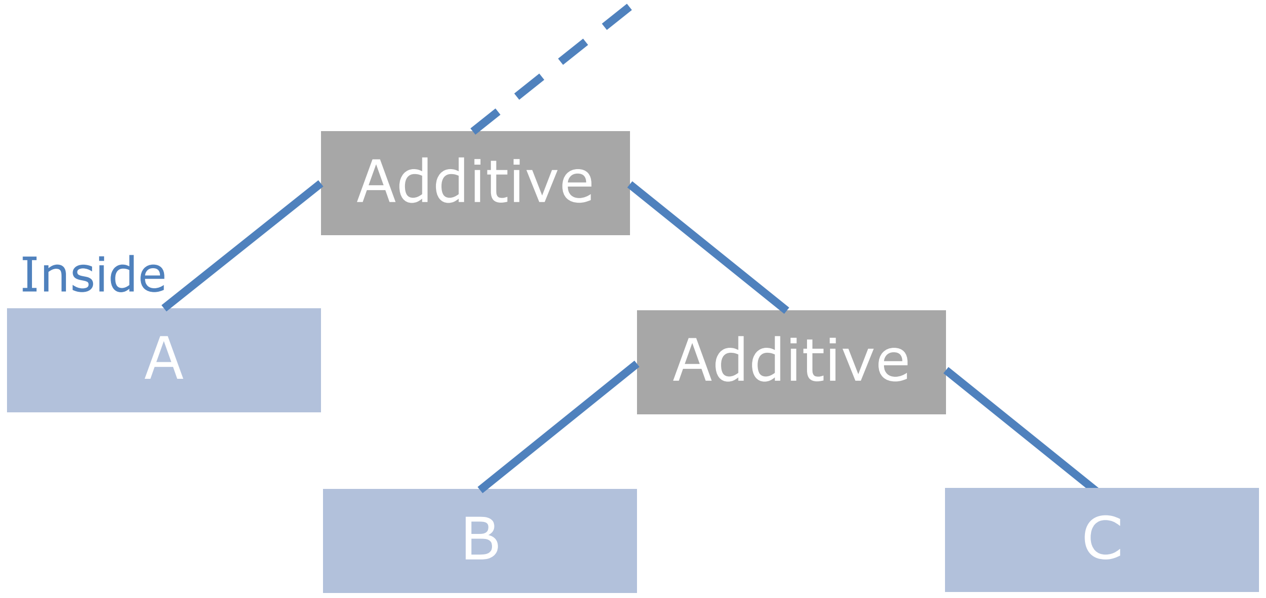
read

Additive
operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	4	5	6	7
(Aligned)	1	8	9	10	11
(Rev-Aligned)	2	12	13	14	15
(Outside)	3	16	17	18	19
(Inside) Brush C	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

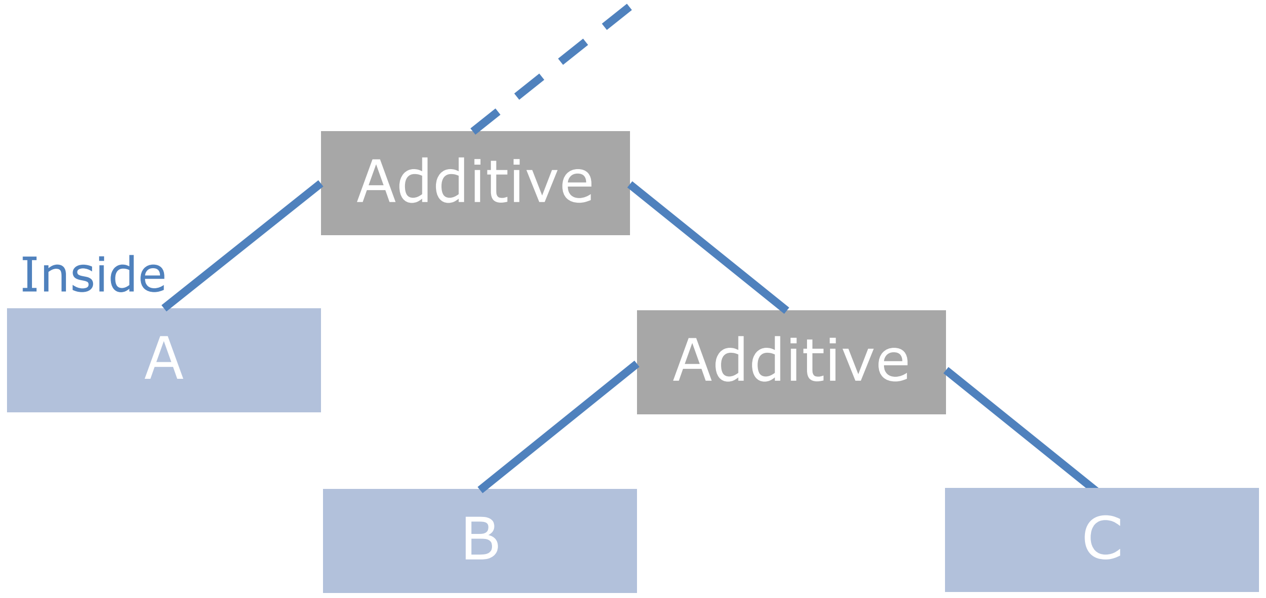


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	4	5	6	7
(Aligned)	1	8	9	10	11
(Rev-Aligned)	2	12	13	14	15
(Outside)	3	16	17	18	19
(Inside) Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	2	2
	7	0	1	2	3
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

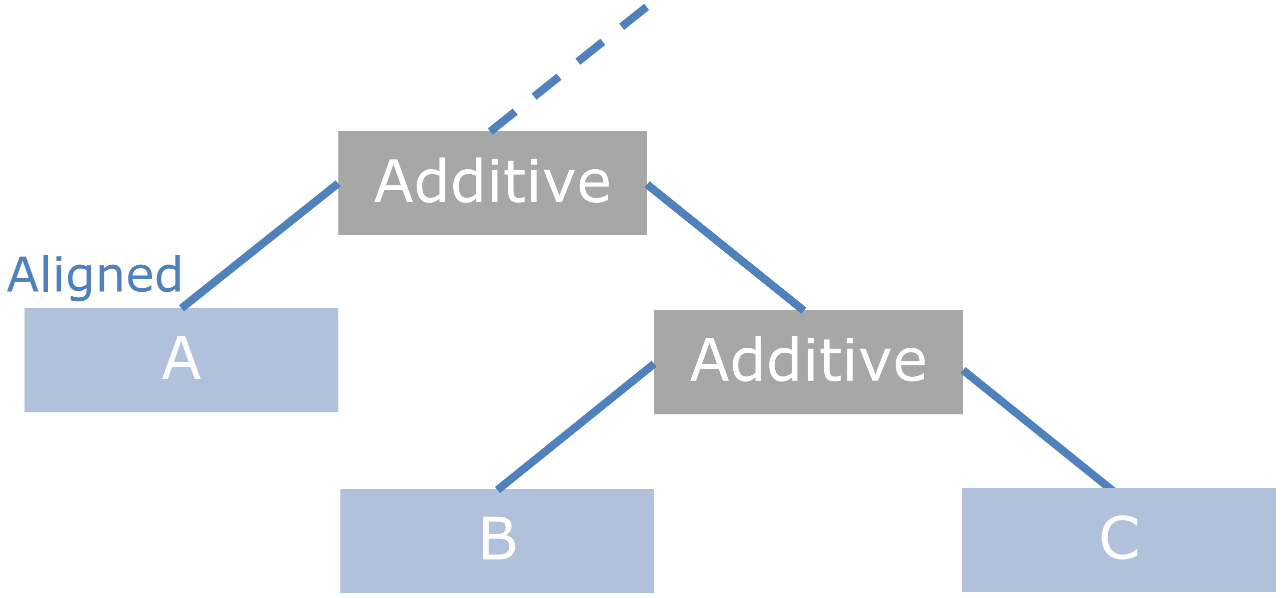


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	0	0	0
		2	0	0	0	0
		3	0	0	0	0
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	2	2
		7	0	1	2	3
(Rev-Aligned)		8	0	0	0	0
		9	0	1	0	0
		10	0	0	2	2
		11	0	1	2	3
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3

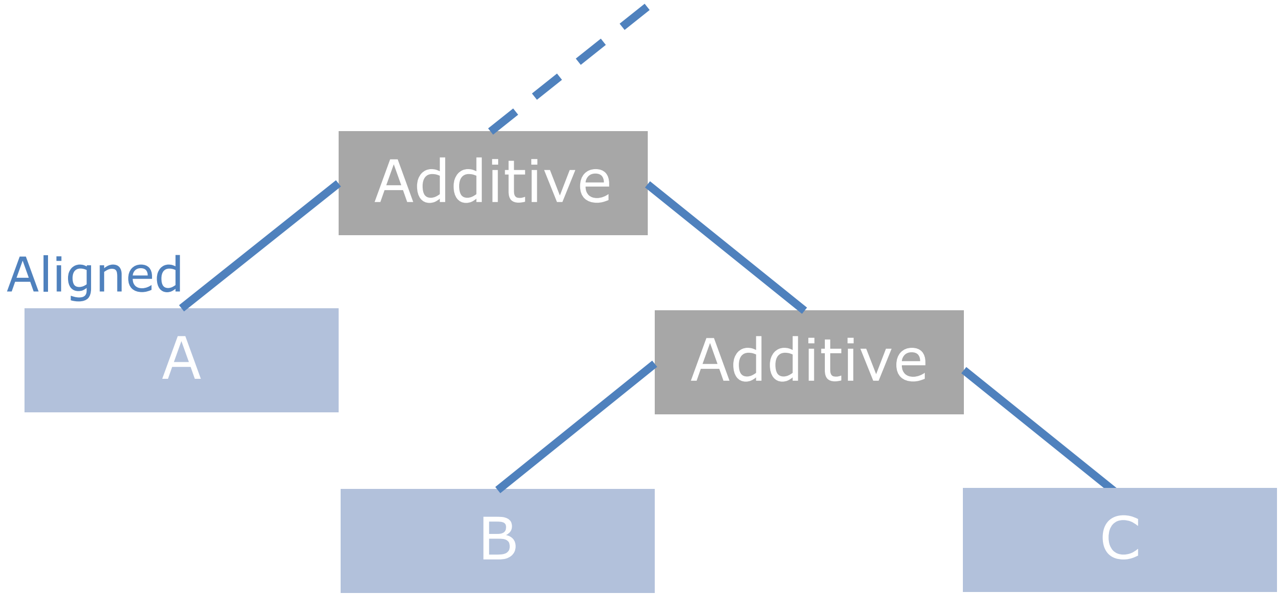


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

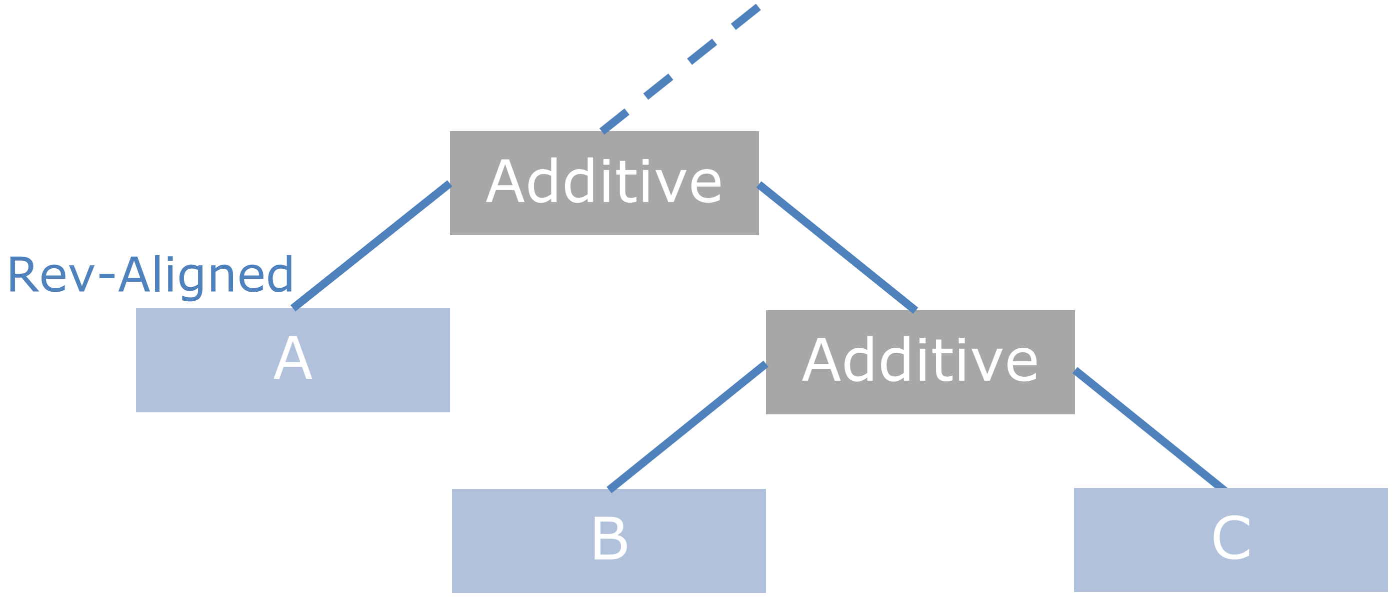


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
(Rev-Aligned)	8	0	0	0	0
	9	0	1	0	0
	10	0	0	2	2
	11	0	1	2	3
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

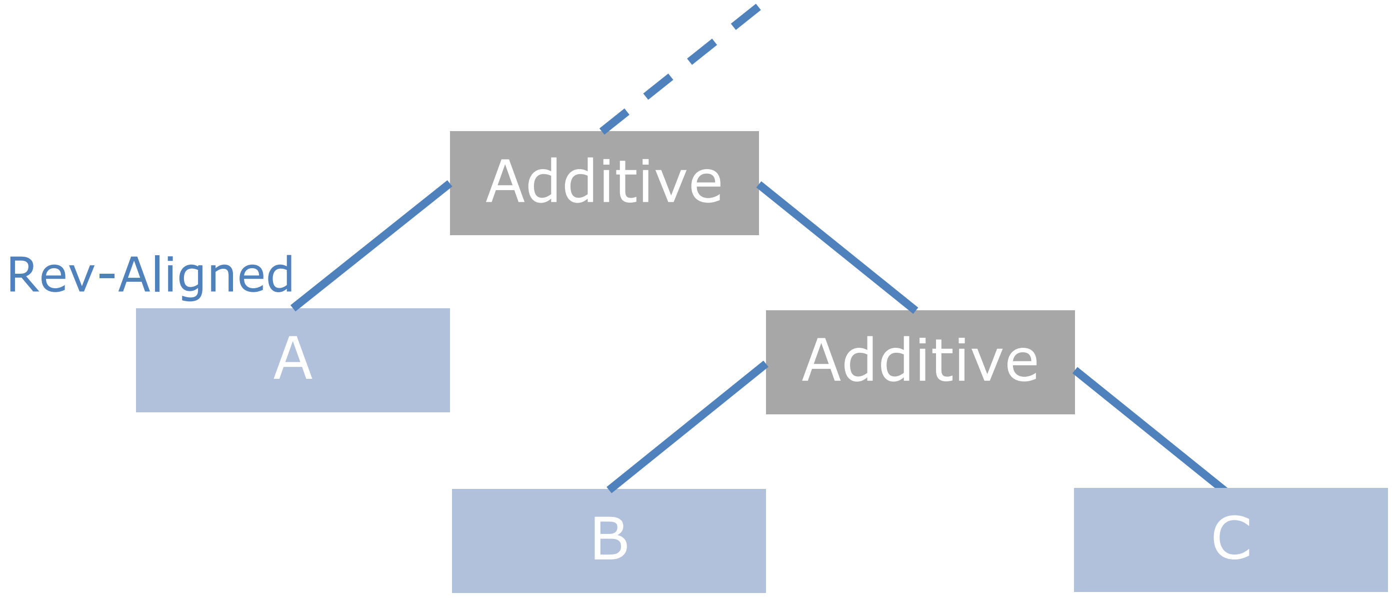


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
(Rev-Aligned)	8	0	0	0	0
	9	0	0	0	0
	10	0	0	2	2
	11	0	0	2	2
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

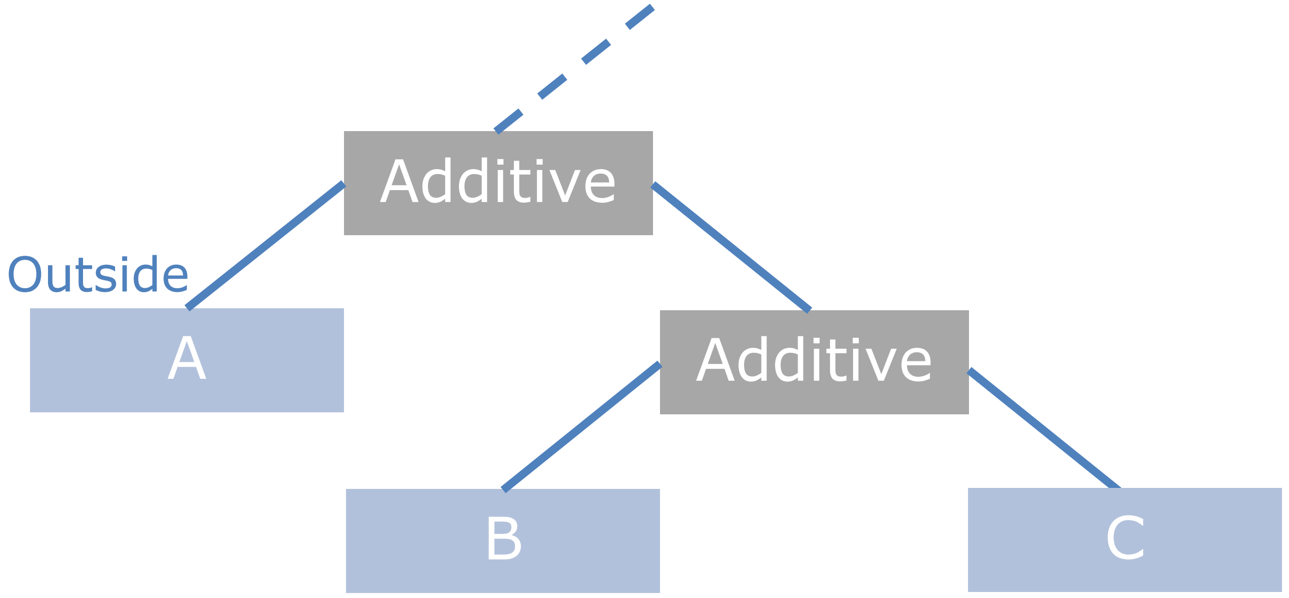


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
(Inside) Brush B	0	0	1	2	3
(Aligned)	1	4	5	6	7
(Rev-Aligned)	2	8	9	10	11
(Outside)	3	12	13	14	15
(Inside) Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
(Aligned)	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
(Rev-Aligned)	8	0	0	0	0
	9	0	0	0	0
	10	0	0	2	2
	11	0	0	2	2
(Outside)	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

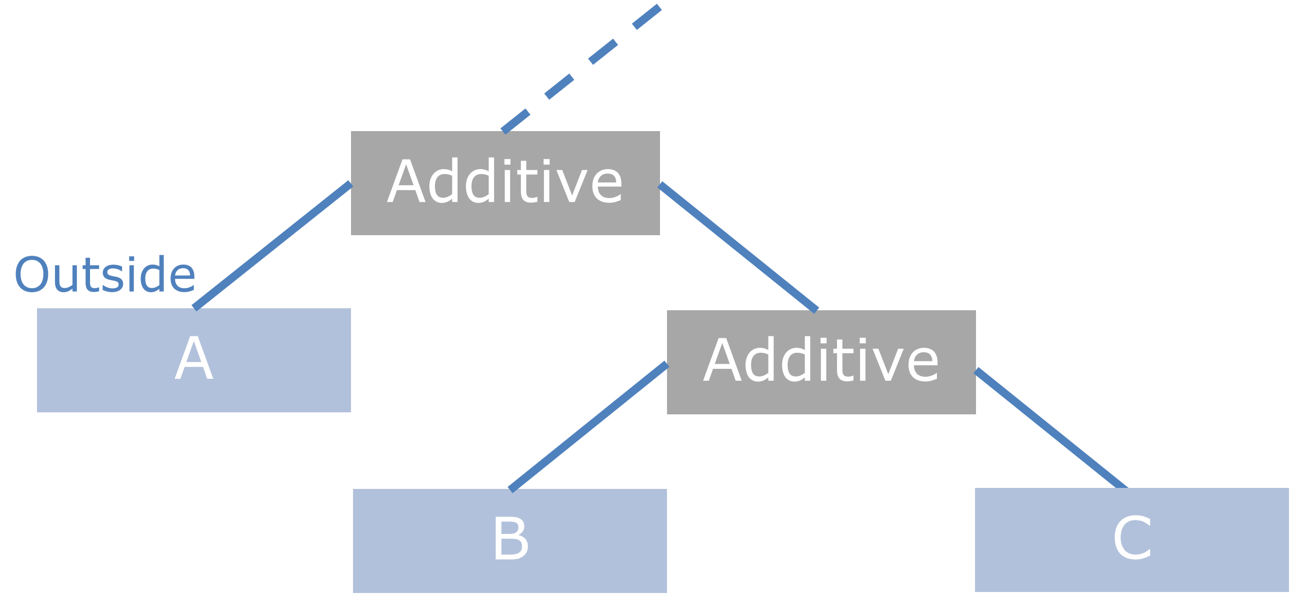


Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

		Index	Inside	Aligned	Rev-Aligned	Outside
	Brush A	-	0	1	2	3
(Inside)	Brush B	0	0	1	2	3
(Aligned)		1	4	5	6	7
(Rev-Aligned)		2	8	9	10	11
(Outside)		3	12	13	14	15
(Inside)	Brush C	0	0	0	0	0
		1	0	0	0	0
		2	0	0	0	0
		3	0	0	0	0
(Aligned)		4	0	0	0	0
		5	0	1	0	0
		6	0	0	0	0
		7	0	1	0	1
(Rev-Aligned)		8	0	0	0	0
		9	0	0	0	0
		10	0	0	2	2
		11	0	0	2	2
(Outside)		12	0	0	0	0
		13	0	1	0	0
		14	0	0	2	2
		15	0	1	2	3



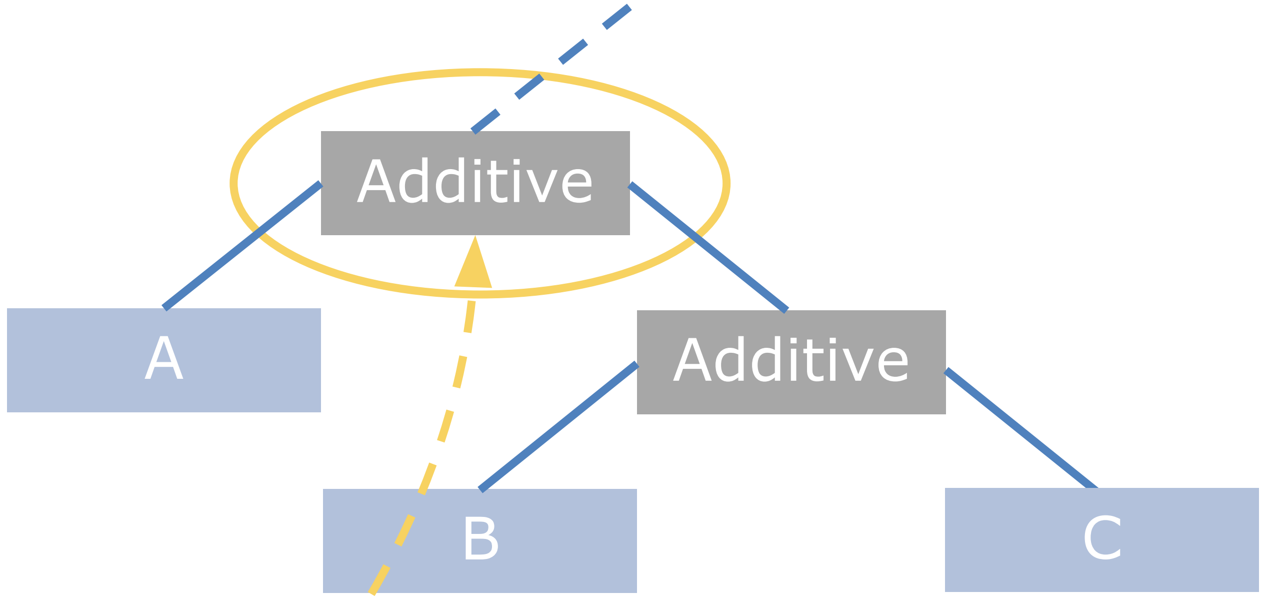
Additive operation table

	Inside	Aligned	Rev-Aligned	Outside
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	1	2	3
	1	4	5	6	7
	2	8	9	10	11
	3	12	13	14	15
Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
	8	0	0	0	0
	9	0	0	0	0
	10	0	0	2	2
	11	0	0	2	2
	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

final output



Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	1	2	3
	1	4	5	6	7
	2	8	9	10	11
	3	12	13	14	15
Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
	8	0	0	0	0
	9	0	0	0	0
	10	0	0	2	2
	11	0	0	2	2
	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	1	2	3
	1	4	5	6	7
	2	8	9	10	11
	3	12	13	14	15
Brush C	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0
	5	0	1	0	0
	6	0	0	0	0
	7	0	1	0	1
	8	0	0	0	0
	9	0	0	0	0
	10	0	0	2	2
	11	0	0	2	2
	12	0	0	0	0
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	1	2	3
	1	4	5	6	7
	2	8	9	10	11
	3	12	13	14	15
Brush C	0	0	0	0	0
	1				
	2				
	3				
	4				
	5	0	1	0	0
	6				
	7	0	1	0	1
	8	-	-	-	-
	9	-	-	-	-
	10	0	0	2	2
	11	0	0	2	2
	12	-	-	-	-
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	5	0	7
	2	0	0	10	11
	3	0	13	14	15
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	5	0	1	0	0
	-	-	-	-	-
	7	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	10	0	0	2	2
	11	0	0	2	2
	-	-	-	-	-
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	5	0	7
	2	0	0	10	11
	3	0	13	14	15
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	5	0	1	0	0
	-	-	-	-	-
	7	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	10	0	0	2	2
	11	0	0	2	2
	-	-	-	-	-
	13	0	1	0	0
	14	0	0	2	2
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	5	0	7
	2	0	0	10	10
	3	0	13	10	15
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	5	0	1	0	0
	-	-	-	-	-
	7	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	10	0	0	2	2
	-	-	-	-	-
	-	-	-	-	-
	13	0	1	0	0
	-	-	-	-	-
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	5	0	7
	2	0	0	10	10
	3	0	13	10	15
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	5	0	1	0	0
	-	-	-	-	-
	7	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	10	0	0	2	2
	-	-	-	-	-
	-	-	-	-	-
	13	0	1	0	0
	-	-	-	-	-
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	5	0	7
	2	0	0	10	10
	3	0	5	10	15
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	5	0	1	0	0
	-	-	-	-	-
	7	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	10	0	0	2	2
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	15	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	2
	2	0	0	3	3
	3	0	1	3	4
Brush C	0	0	0	0	0
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	1	0	1	0	0
	-	-	-	-	-
	2	0	1	0	1
	-	-	-	-	-
	-	-	-	-	-
	3	0	0	2	2
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	-	-	-	-	-
	4	0	1	2	3

Routing table

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	2	3
Brush B	0	0	0	0	0
	1	0	1	0	2
	2	0	0	3	3
	3	0	1	3	4
Brush C	0	0	0	0	0
	1	0	1	0	0
	2	0	1	0	1
	3	0	0	2	2
	4	0	1	2	3

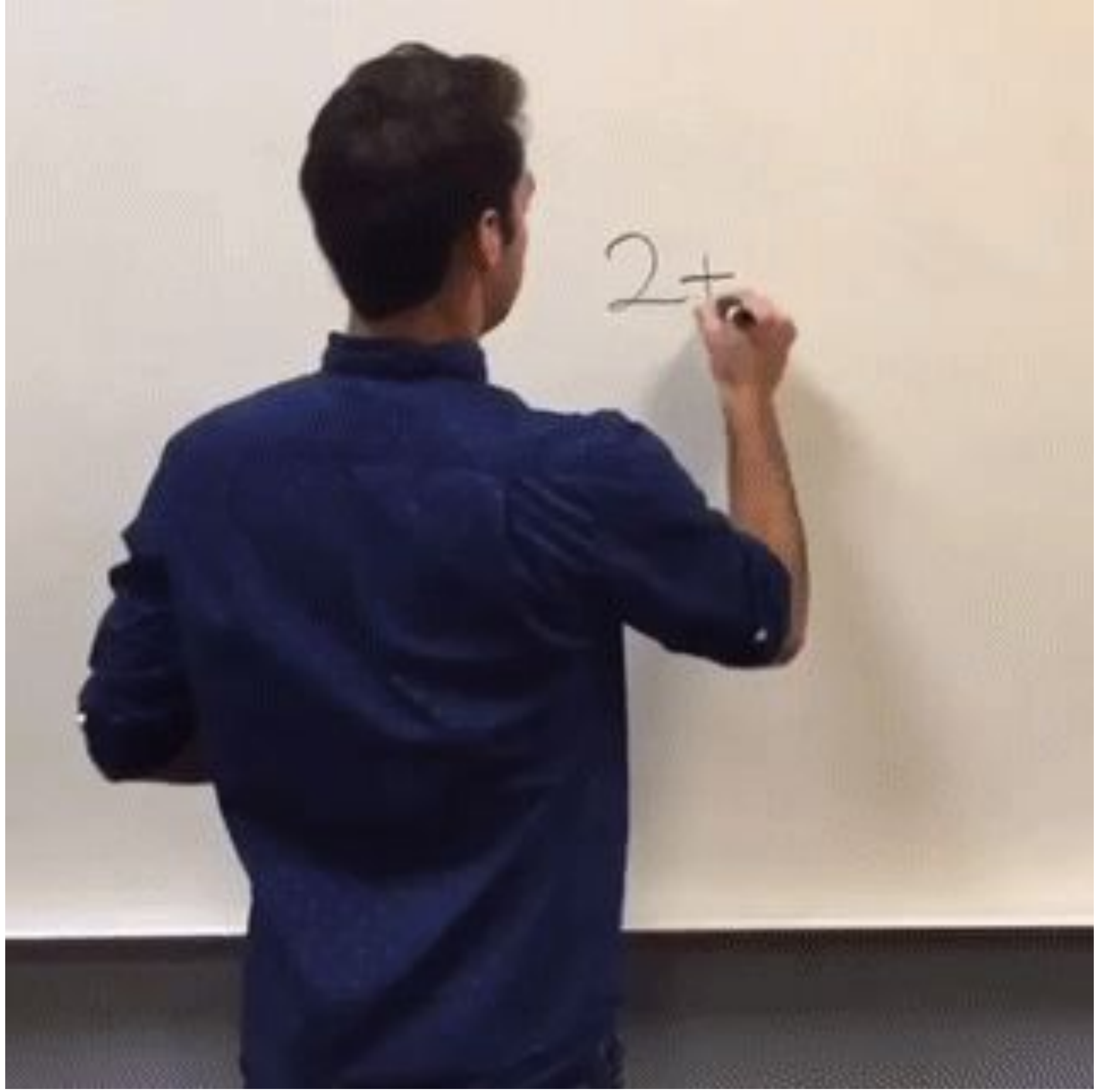
Compact routing table

Cacheable per brush

Number of rows per brush < 255

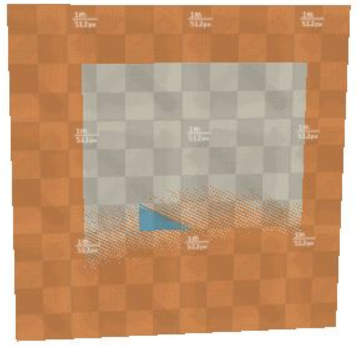
- 4 output values, 0-3, 2 bits * 4 = 8
- Row can be stored as 4 bytes
- More than 6 rows is rare
- Not all row output combinations make sense, or can be generated by operations
  - Theoretical maximum is probably *a lot* lower

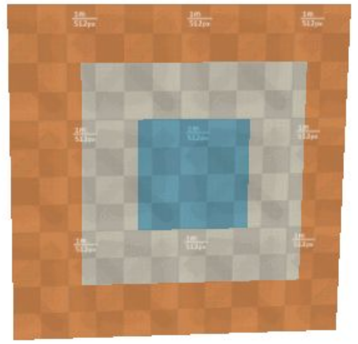
V= 1
$\lim(1+\frac{1}{n})^n = e$


2+

Brush categorization

Can have **multiple** brushes overlapping on the **same polygon area**





Solution: Make every brush **remove** the area of the **previous** brushes

Brush categorization

Solution:
Switch to variation of our operation tables that removes polygons that overlap by returning the **outside** category.

We use this on each brush beyond the brush the routing table belongs to.
Note: we keep using the original operation tables when combining routing tables

	Additive Operation Table	Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Inside	Inside	Inside
	Aligned	Inside	Outside	Inside	Aligned
	Rev-Aligned	Inside	Inside	Outside	Rev-Aligned
	Outside	Inside	Outside	Outside	Outside

Used to be Aligned

Used to be Rev-Aligned

Brush categorization

Solution:
Switch to variation of our operation tables that removes polygons that overlap by returning the **outside** category.

We use this on each brush beyond the brush the routing table belongs to.
Note: we keep using the original operation tables when combining routing tables

Subtractive Operation Table		Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Outside	Outside	Outside	Inside
	Aligned	Outside	Outside	Outside	Aligned
	Rev-Aligned	Outside	Outside	Outside	Rev-Aligned
	Outside	Outside	Outside	Outside	Outside

Used to be Aligned

Used to be Rev-Aligned

Brush categorization

Solution:
Switch to variation of our operation tables that removes polygons that overlap by returning the **outside** category.

We use this on each brush beyond the brush the routing table belongs to.
Note: we keep using the original operation tables when combining routing tables

	Intersecting Operation Table	Brush A			
		Inside	Aligned	Rev-Aligned	Outside
Brush B	Inside	Inside	Outside	Outside	Outside
Brush B	Aligned	Aligned	Outside	Outside	Outside
Brush B	Rev-Aligned	Rev-Aligned	Outside	Outside	Outside
Brush B	Outside	Outside	Outside	Outside	Outside

Used to be Aligned

Used to be Rev-Aligned

Overview

1. History of CSG

The algorithm

2. Iterative updates
3. Intersections
4. Mesh Generation
5. Polygon categories, Routing & Operation tables
6. **Putting it all together**

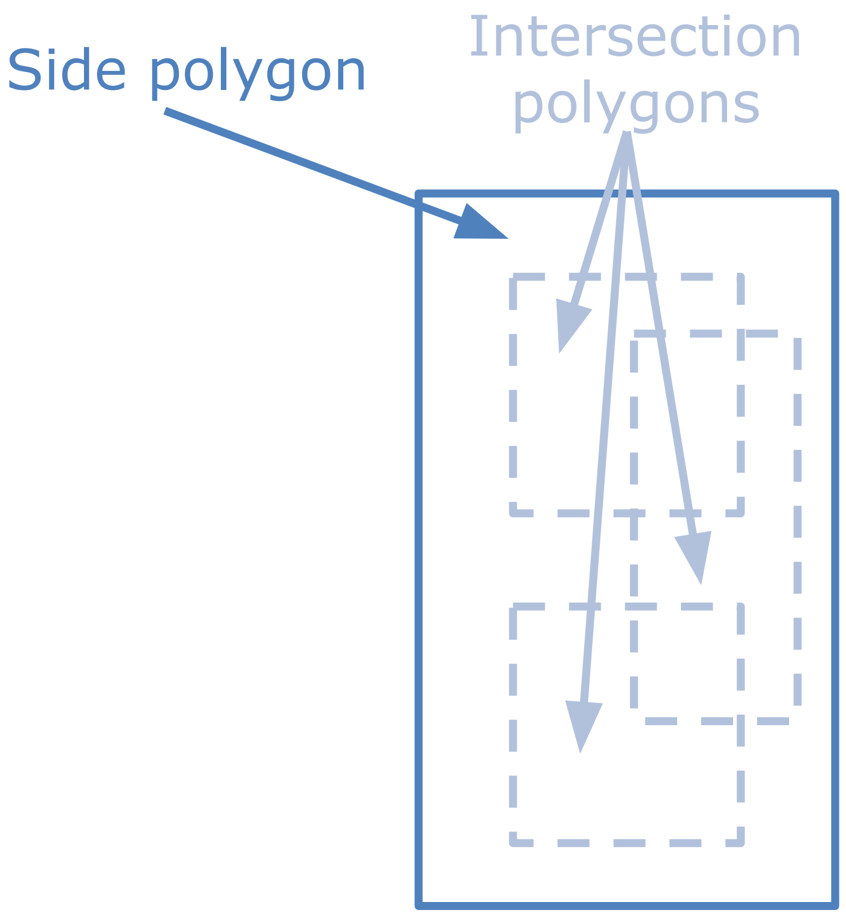
Using the routing table

For each brush in the CSG tree, loop through the **brushes on <u>its own</u> routing table**

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Using the routing table

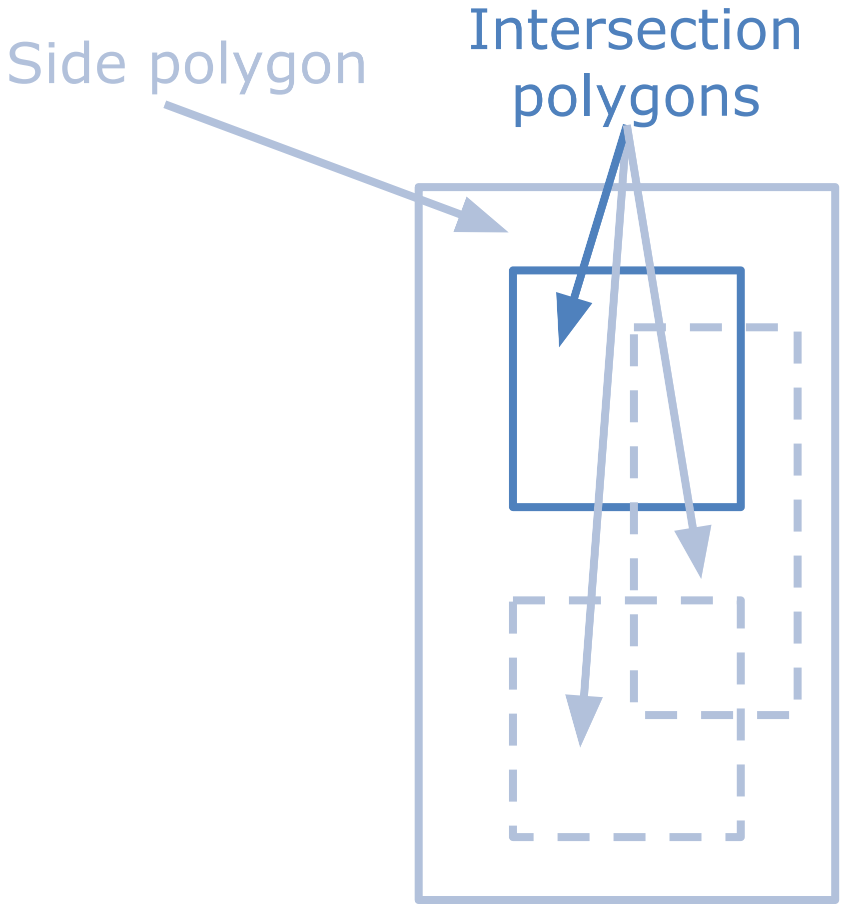
We do this for each ***side polygon*** of the brush we're processing



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Using the routing table

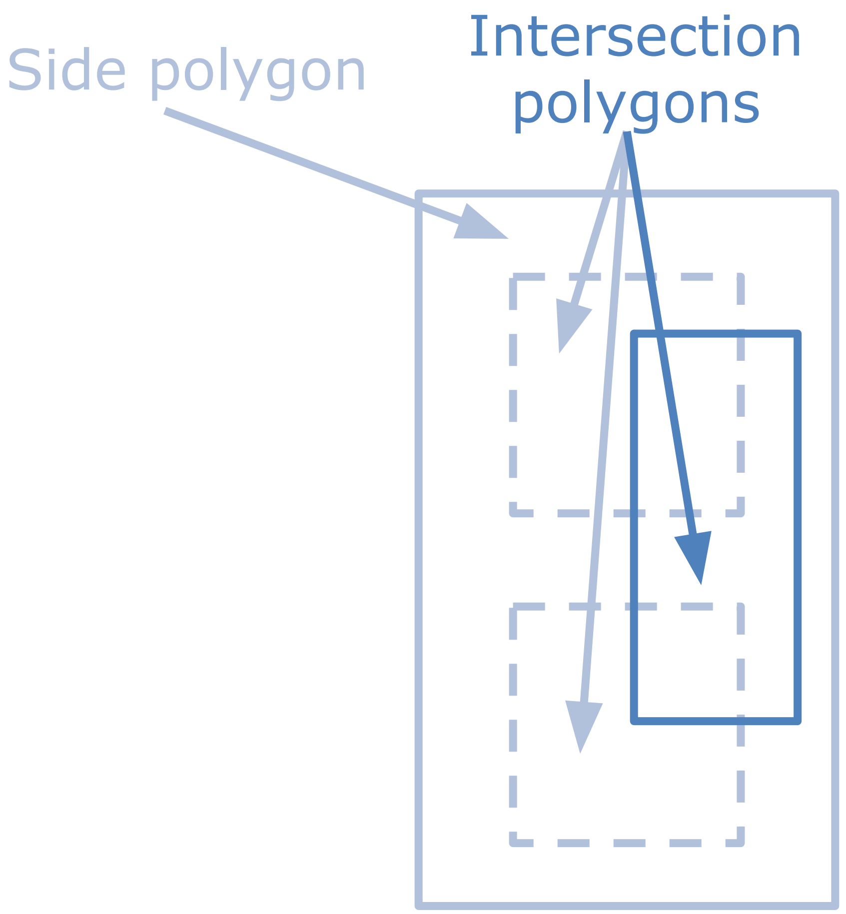
Each ***intersection polygon*** represents an ***intersection*** between the processed brush and a brush that's represented in the routing table



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Using the routing table

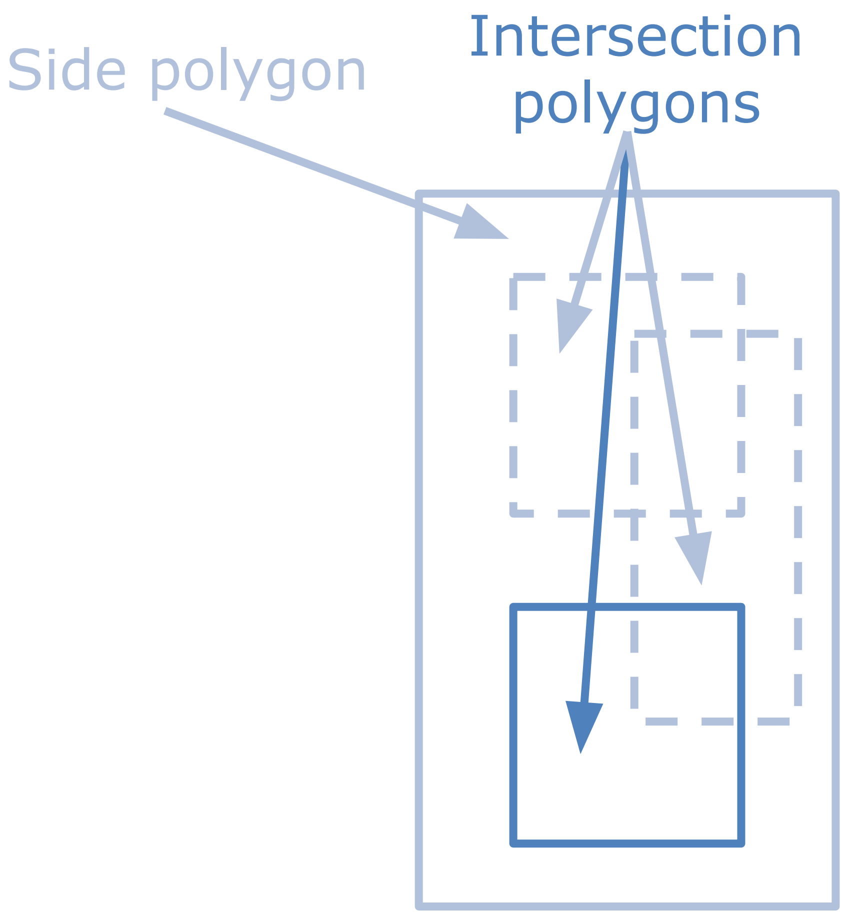
Each ***intersection polygon*** represents an ***intersection*** between the processed brush and a brush that's represented in the routing table



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Using the routing table

Each ***intersection polygon*** represents an ***intersection*** between the processed brush and a brush that's represented in the routing table



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use ***interior category*** (inside polygon)

Determine which polygons to show

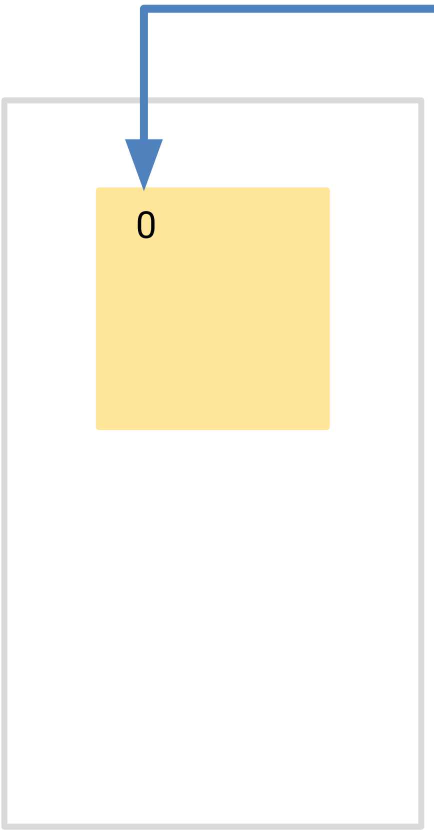
- Use categories on intersection polygons
    - Use ***interior category*** (inside polygon)

Interior intersection polygon:
Rev-Aligned

	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
    - Use interior category (inside polygon)

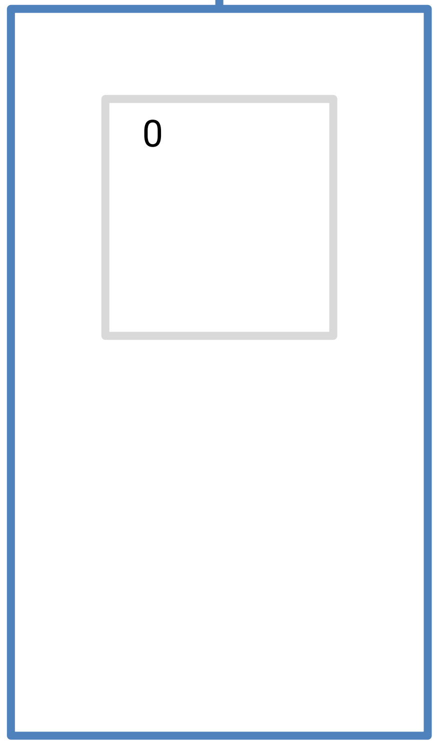


	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - ***Outside*** for everything else

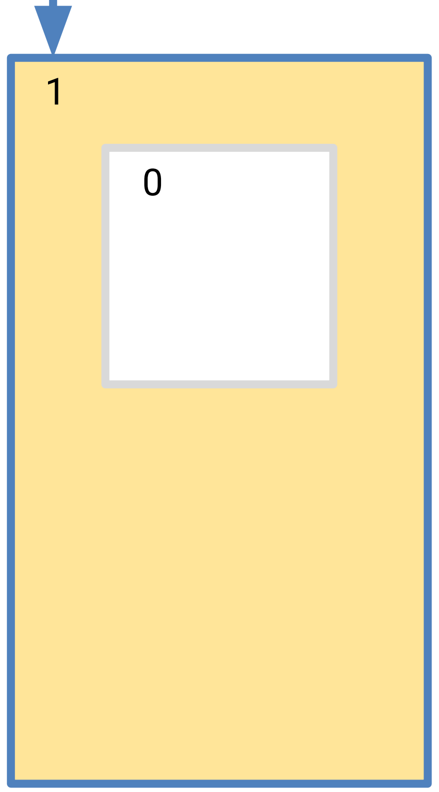
Outside intersection category:
Outside (always)



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

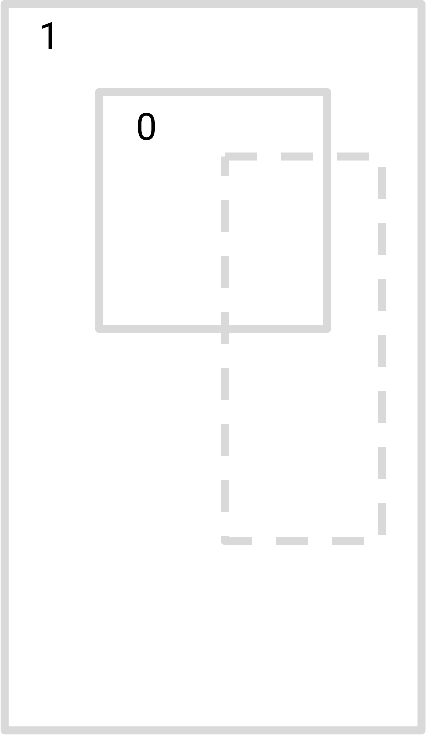
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

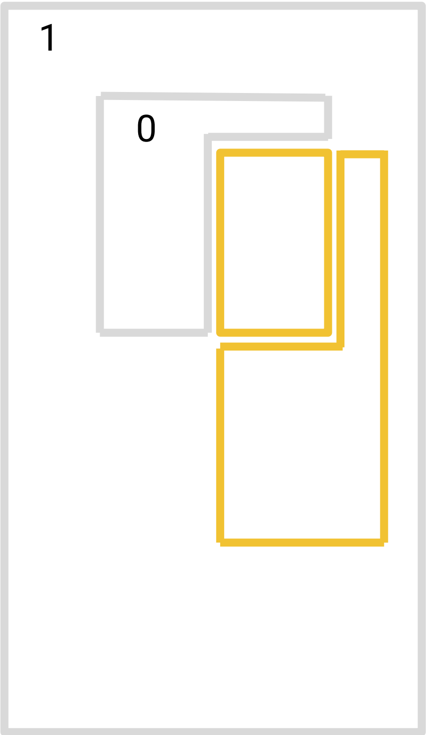
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

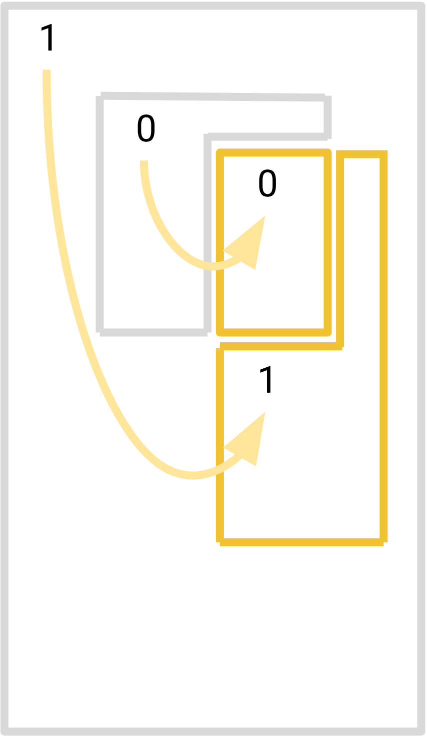
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else

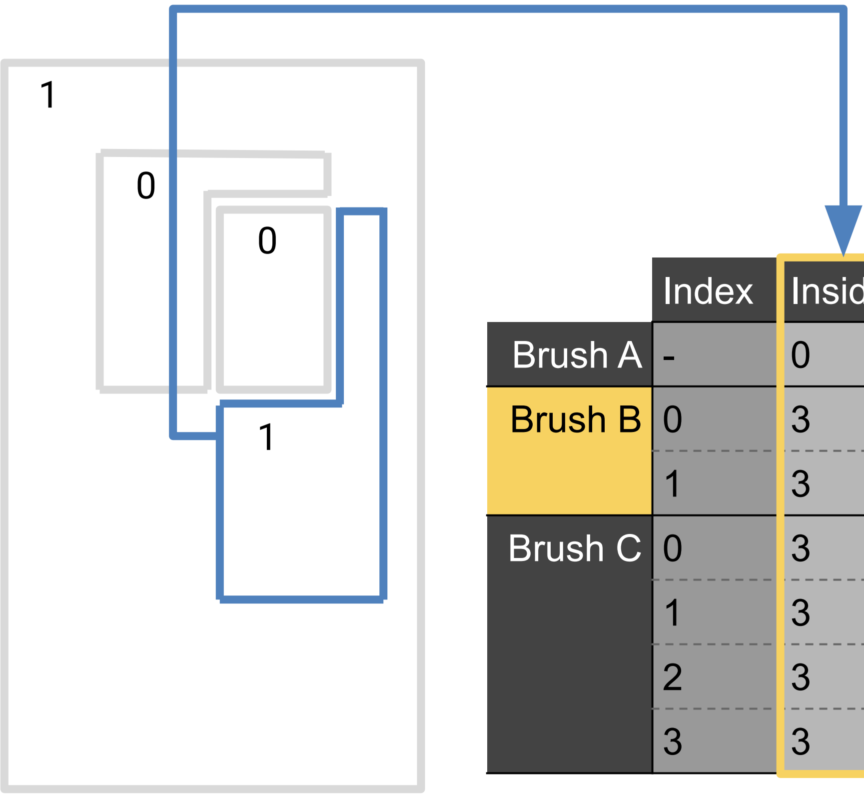


	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else

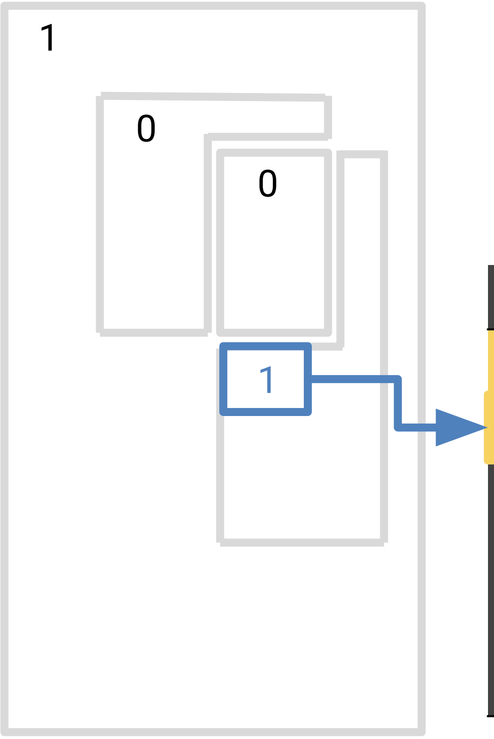
Interior intersection polygon
Aligned



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

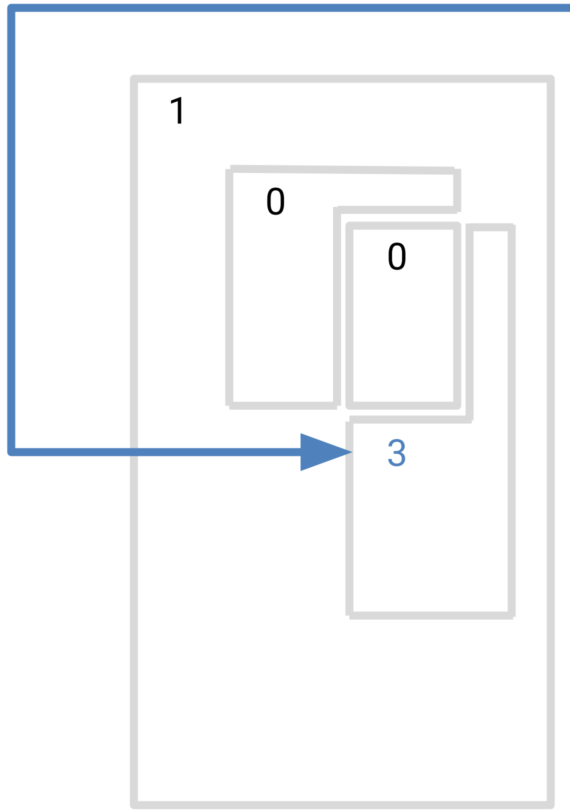
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

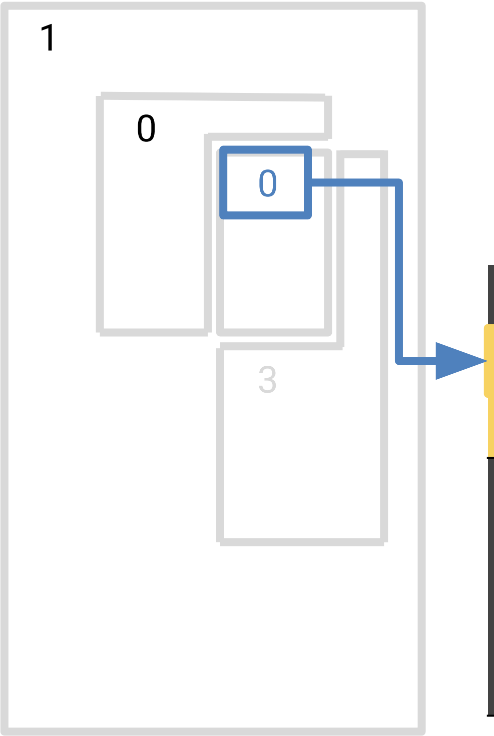
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

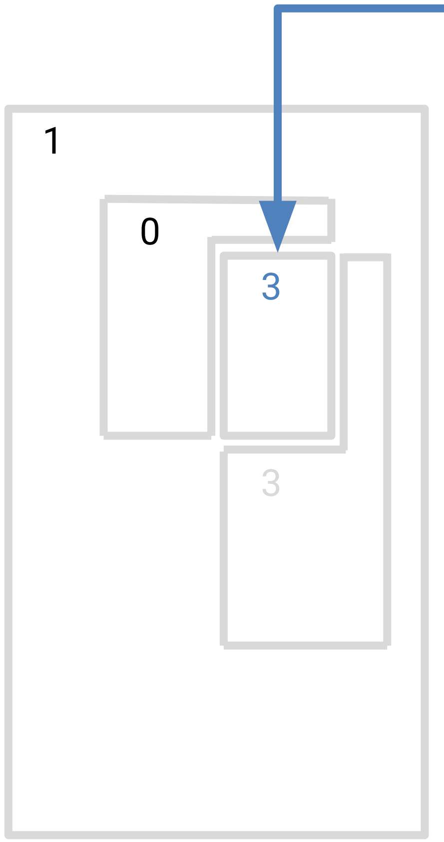
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else

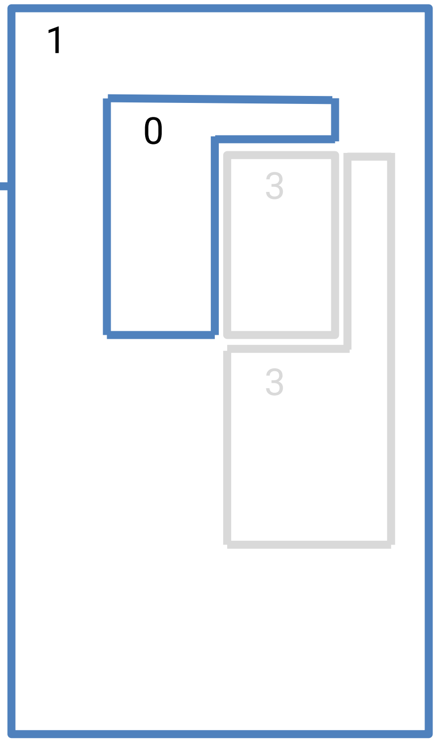


	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else

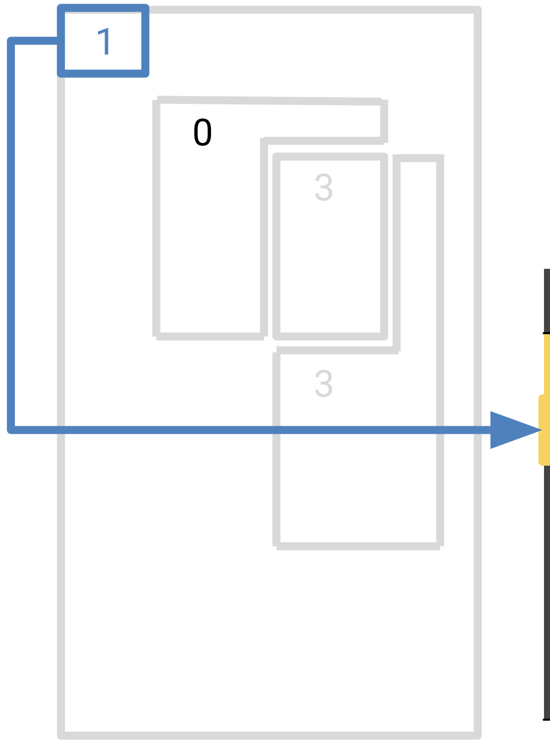
Outside intersection polygon:
Outside (always)



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

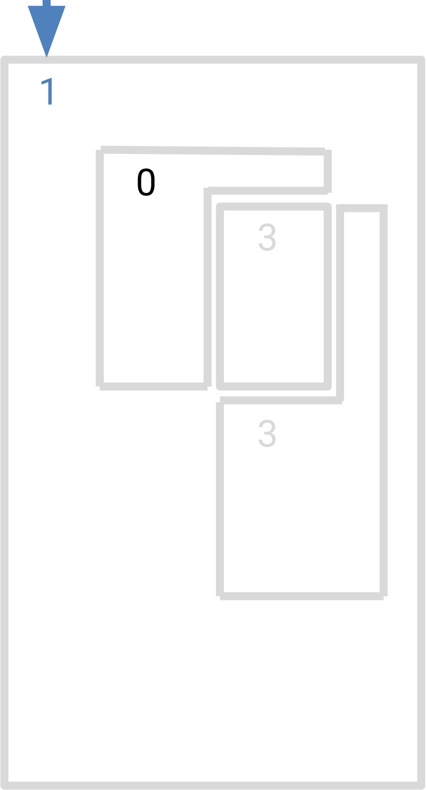
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

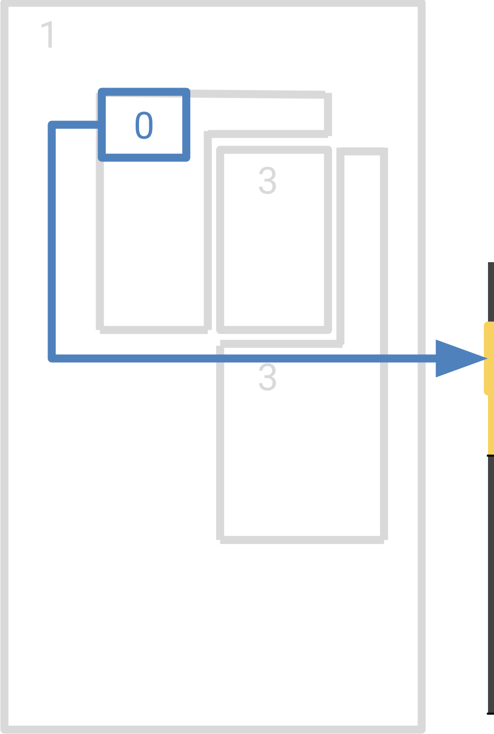
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

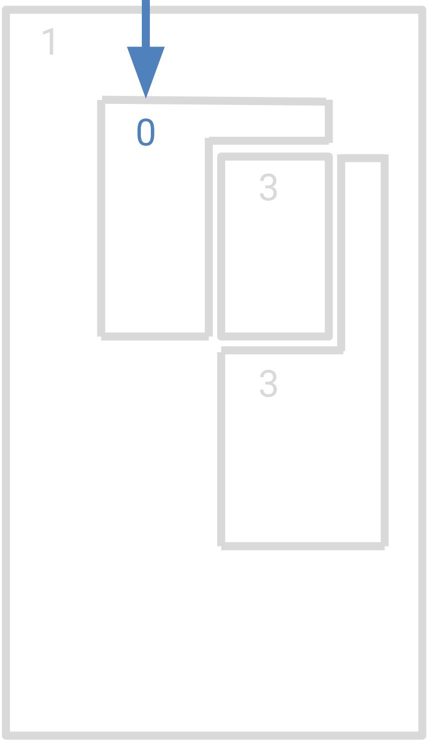
- Use categories on intersection polygons
    - Use interior category (inside polygon)
    - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

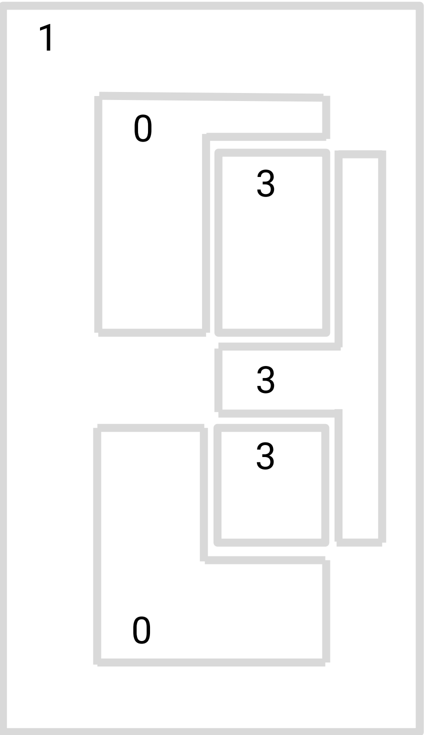
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

Determine which polygons to show

- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else

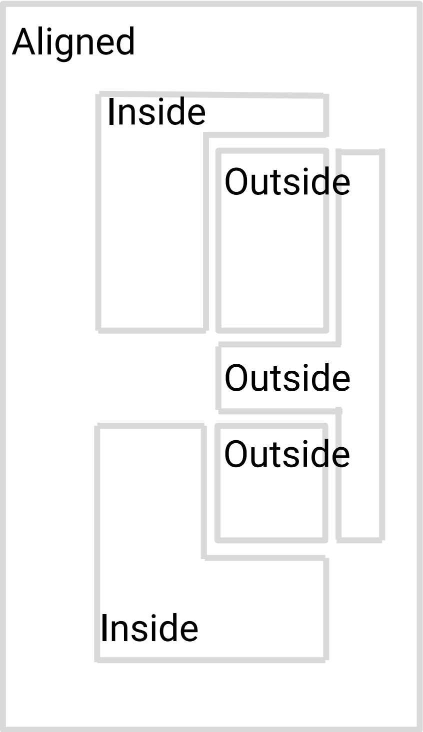


	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

GAME DEVELOPERS CONFERENCE
MARCH 16–20, 2020 | #GDC20

Determine which polygons to show

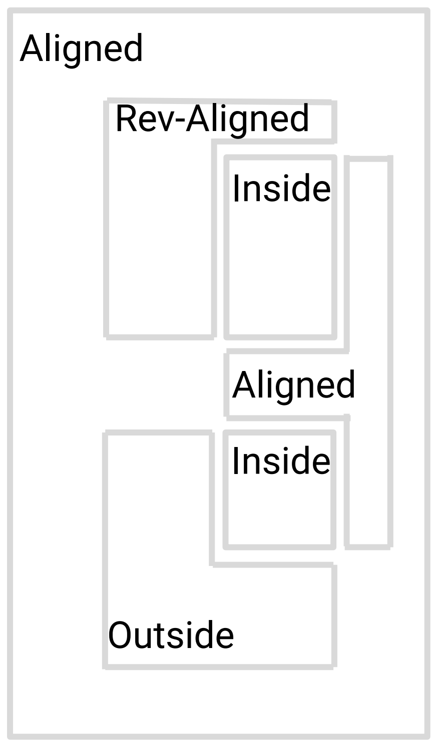
- Use categories on intersection polygons
  - Use interior category (inside polygon)
  - *Outside* for everything else



	Index	Inside	Aligned	Rev-Aligned	Outside
Brush A	-	0	1	0	1
Brush B	0	3	2	1	0
	1	3	0	1	1
Brush C	0	3	2	1	0
	1	3	0	1	1
	2	3	2	0	2
	3	3	3	3	3

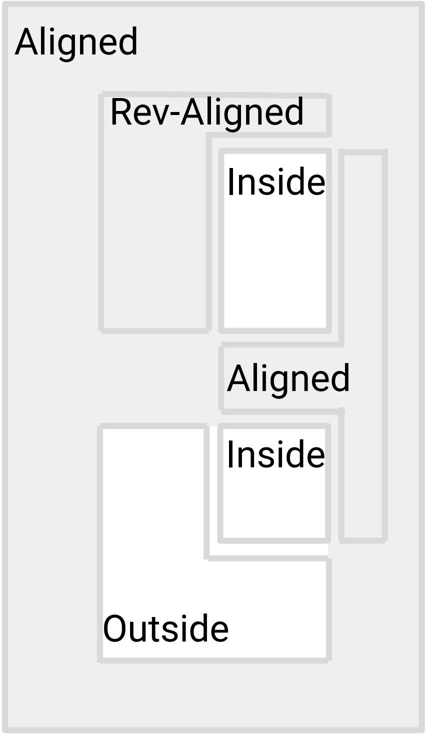
Creating meshes

- Final polygon category will determine if it's kept or discarded
  - Remember, we only keep (reverse-)aligned polygons



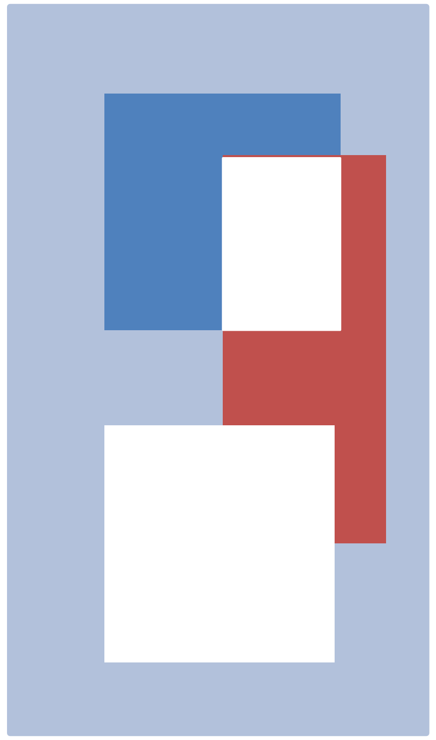
Creating meshes

- Final polygon category will determine if it’s kept or discarded
  - Remember, we only keep (reverse-)aligned polygons



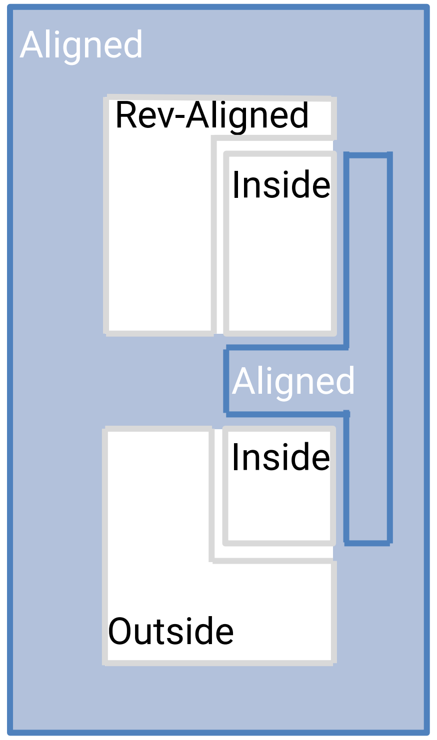
Creating meshes

- Final polygon category will determine if it’s kept or discarded
  - Remember, we only keep (reverse-)aligned polygons



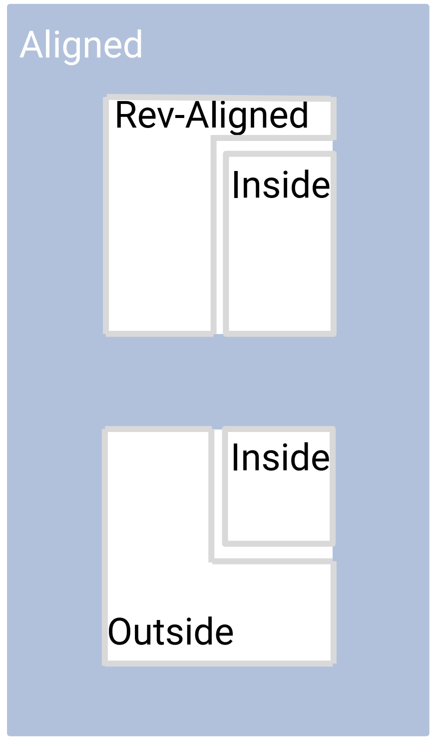
Creating meshes

- Final polygon category will determine if it's kept or discarded
  - Remember, we only keep (reverse-)aligned polygons
  - We can also combine all polygons that have the same category
    - Remove edges with same indices, but opposite order
    - Identical edges should have their copies removed
    - Combine remaining edges



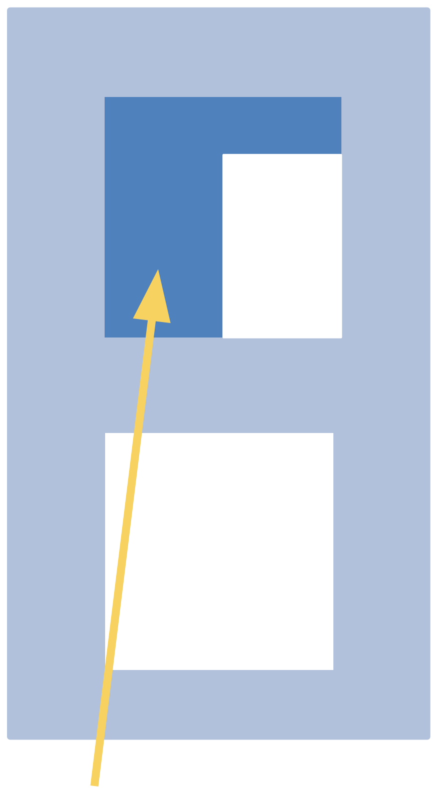
Creating meshes

- Final polygon category will determine if it’s kept or discarded
  - Remember, we only keep (reverse-)aligned polygons
  - We can also combine all polygons that have the same category
    - Remove edges with same indices, but opposite order
    - Identical edges should have their copies removed
    - Combine remaining edges



Creating meshes

- Final polygon category will determine if it's kept or discarded
  - Remember, we only keep (reverse-)aligned polygons
  - We can also combine all polygons that have the same category
    - Remove edges with same indices, but opposite order
    - Identical edges should have their copies removed
    - Combine remaining edges
- We only triangulate the polygons that we keep / after merging
- All reverse aligned polygons need to be flipped around
  - Reverse vertex index order



Reverse Aligned polygons need to be **flipped**

The payoff

- Scalable way of building geometry
- Iterative updates
  - Everything we can do per brush, we can cache per brush!
- Updates can be easily split across multiple cores!

Thank you!

References

Chisel https://github.com/RadicalCSG/Chisel.Prototype

Realtime CSG https://github.com/LogicalError/realtime-CSG-for-unity/