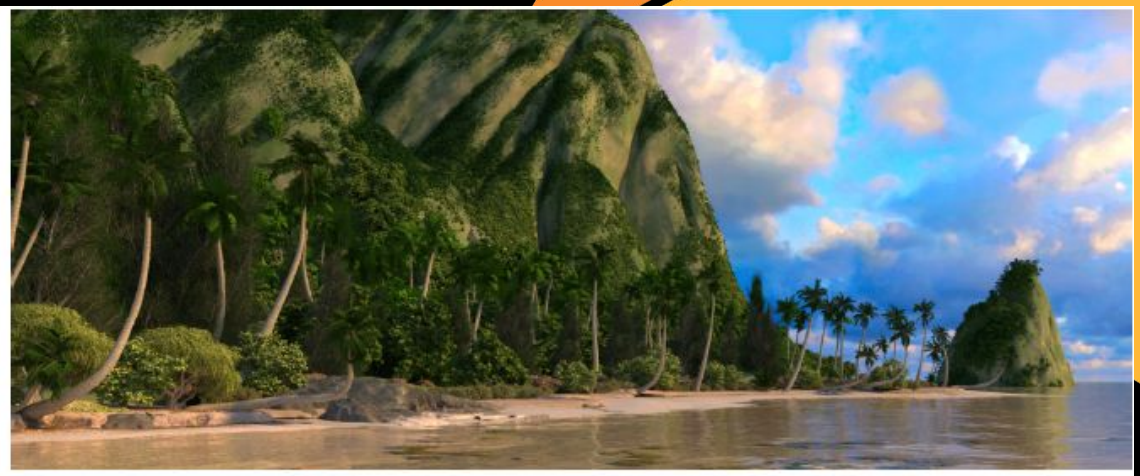
Data Parallel Path Tracing with Object-Hierarchies

Ingo Wald and Steven G Parker

NVIDIA



(Hardware: 4 worker nodes w/ 2×RTX8000, low-end head node, 10-Gigabit Ethernet, screen size 2560 × 1080)

PBRT *landscape*: max 3.7 GB GPU usage, 6.2 FPS (1 path/pixel)
30 K instances, 4.3 B inst. triangles, 370 unique meshes, 500 MB image textures

Disney Moana *island*: max 25 GB GPU usage, 7.9 FPS (1 path/pixel)
39 M instances, 41 B inst. triangles, 7 M unique meshes, 804 MB baked-PTex txt.

About

- Goal: interactive, data-parallel, GPU-enabled, path traced… for Moana-like content
  - Model larger than any one GPU
  - “Production” content
  - Full path tracing

About

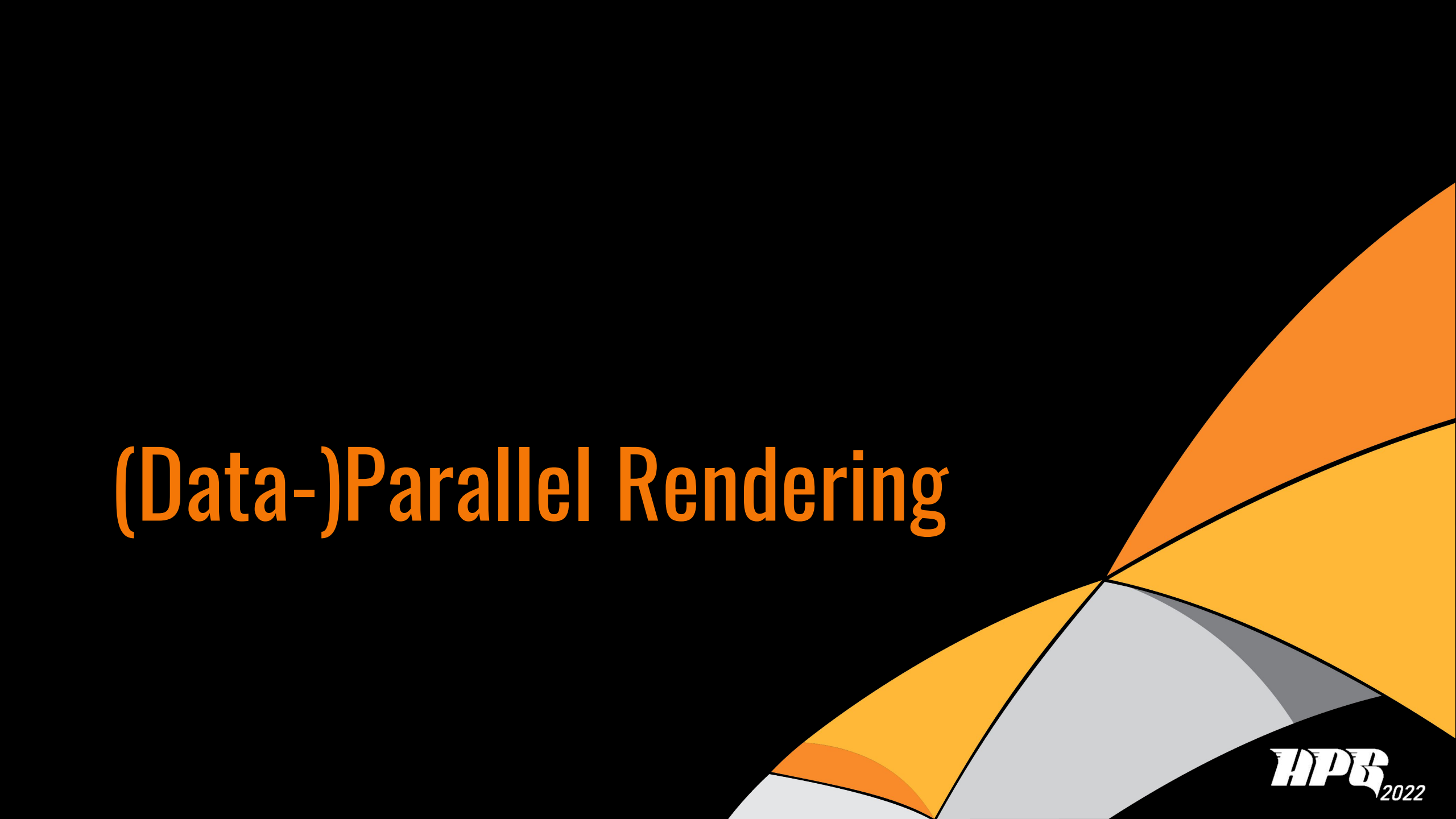
- Paper actually about *three different* things
  1) A *new paradigm* re how to think about partitioning and distributed model content
     - More general: different nodes' content can overlap others, etc
  2) A set of algorithms that realize that
  3) A "system" built on these idea & algorithms
     - interactive, wavefront path tracer, GPU, MPI, …

About

Focus of this talk

- Paper actually about *three different* things
  1) A *new paradigm* re how to think about partitioning and distributed model content
     - More general: different nodes' content can overlap others, etc
  2) A set of algorithms that realize that
  3) A "system" built on these idea & algorithms
     - interactive, wavefront path tracer, GPU, MPI, …

(Data-)Parallel Rendering



(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
  - Every node has only part of the model
    - Usually, because model is too big to fit on a single node/GPU

(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
- Most used today: “sort-last” image compositing
  - But: doesn’t work for secondary rays not for us

(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
- Most used today: “sort-last” image compositing
- When tracing arbitrary rays, and ray needs content that’s on a remote node … two options:
  - Either: fetch model/bvh data to node the ray is on
  - Or: Send (“forward”) ray to the node that has the data

(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
- Most used today: “sort-last” image compositing
- When tracing arbitrary rays, and ray needs content that’s on a remote node … two options:
  - Either: fetch model/bvh data to node the ray is on
  - Or: Send (“forward”) ray to the node that has the data

This talk (see paper for discussion

DPR for Ray Tracing

- Ray forwarding Two key questions:
  - How to partition the scene
  - What nodes should a given ray traverse (and in what order)

- Interesting observations

  a) Not much work on that
    - Some in early times, very little recently

  b) Virtually all prev work assumes spatial partitioning

Ray Forwarding w/ Spatial Partitioning

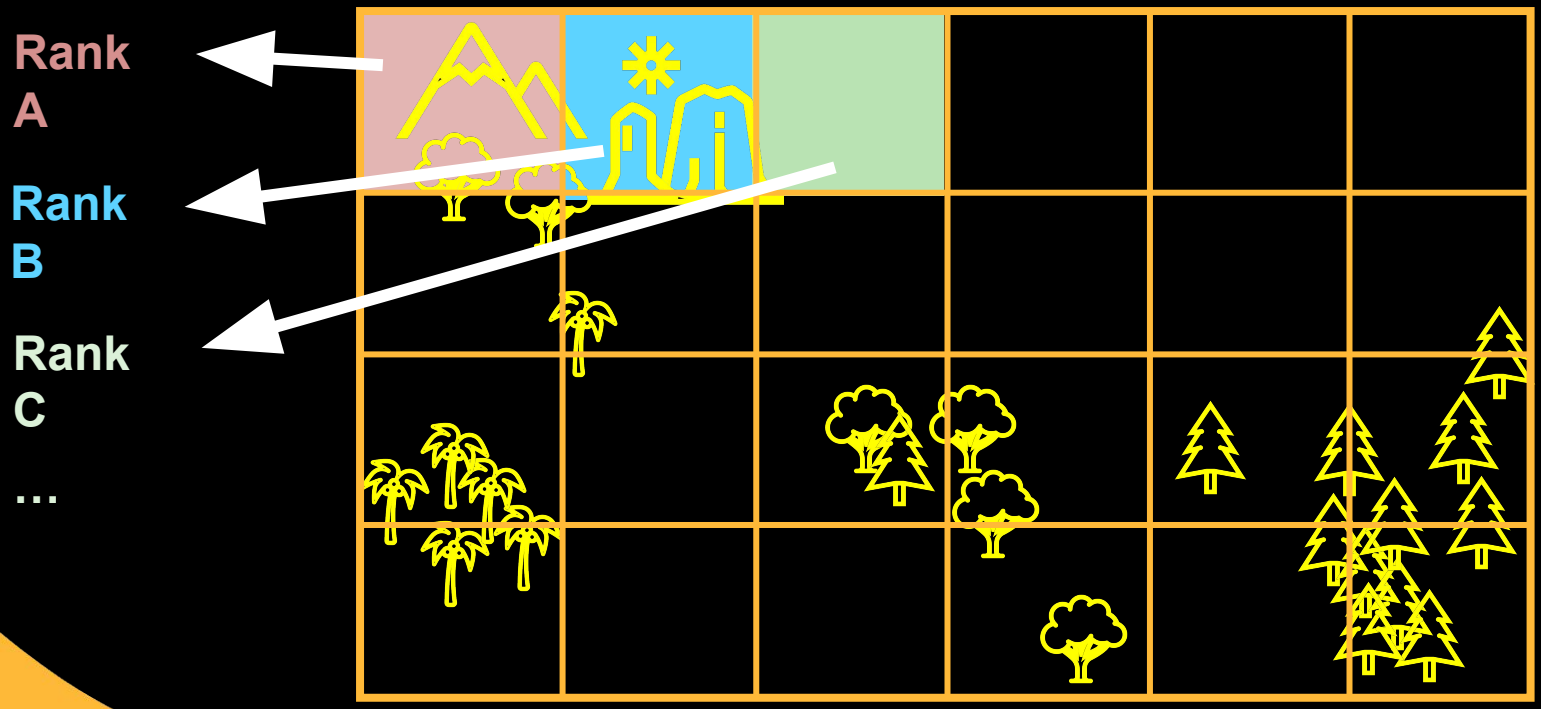
- Partition scene via octree, grid, k-d tree, etc.
- Easy; results in non-overlapping “domains” (one/rank)

Ray Forwarding w/ Spatial Partitioning

- Partition scene via octree, grid, k-d tree, etc.
- Easy; results in non-overlapping “domains” (one/rank)

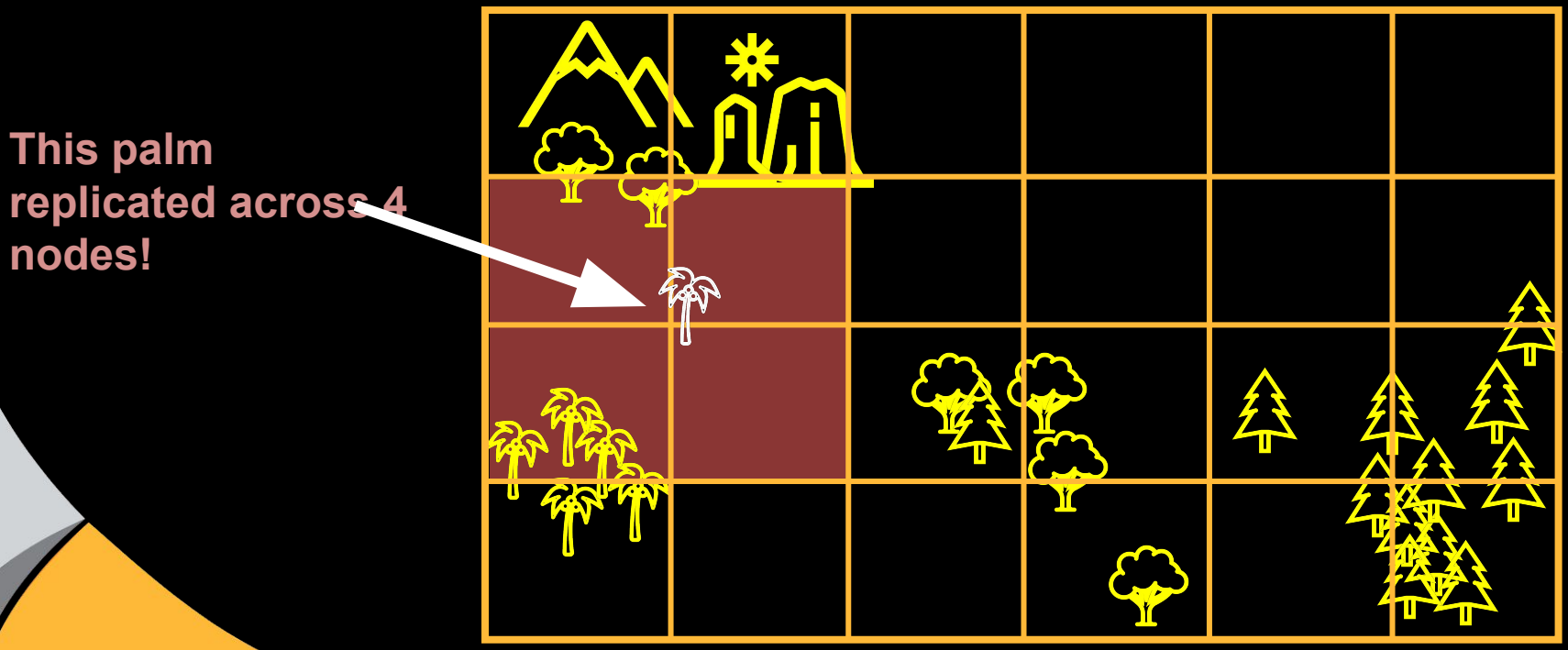
Ray Forwarding w/ Spatial Partitioning

- Partition scene via octree, grid, k-d tree, etc.
- Easy; results in non-overlapping “domains” (one/rank)



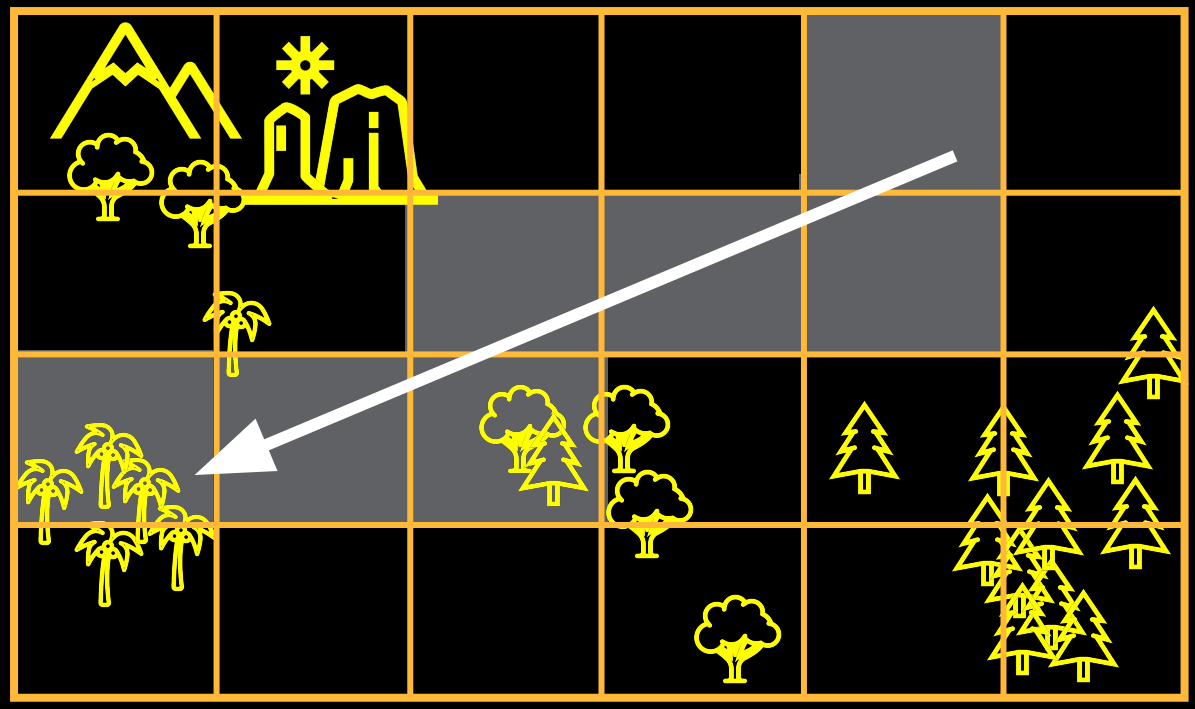
Ray Forwarding w/ Spatial Partitioning

- Partition scene via octree, grid, k-d tree, etc.
- Easy; results in non-overlapping “domains” (one/rank)
  - Prims overlapping more than one domain get replicated



Ray Forwarding w/ Spatial Partitioning

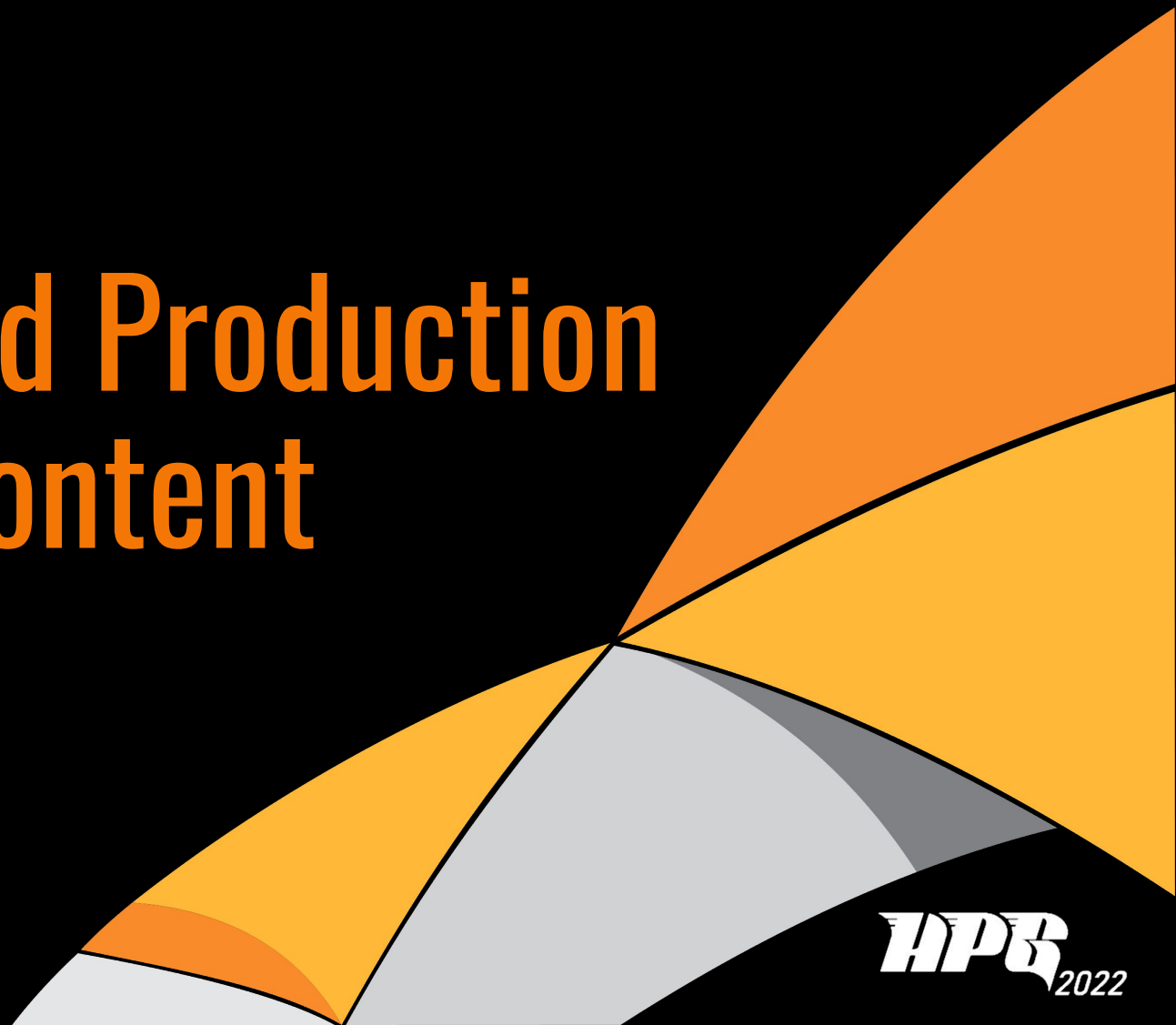
- "Traversing" a ray across ranks is (kind-of) trivial…



Ray Forwarding w/ Spatial Partitioning

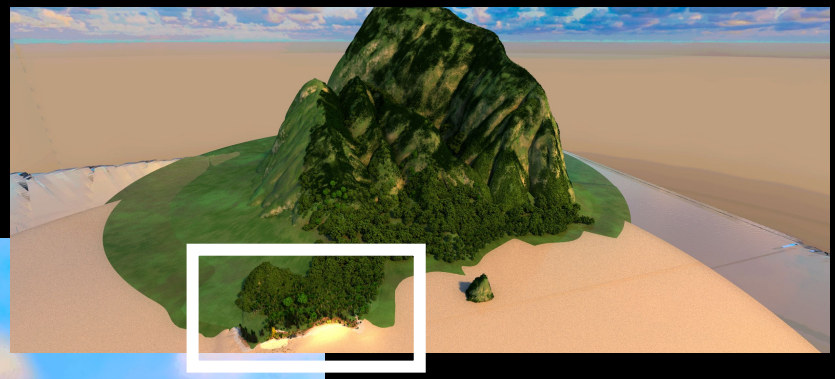
- "Traversing" a ray across ranks is (kind-of) trivial…
- Caveat: "the usual issues" of spatial partitionings
  - Not going into that discussion here …
    - But note: RT has mostly gone from spatial part to object hier.!
  - Main issue: Replication

Ray Forwarding and Production Rendering-Style Content

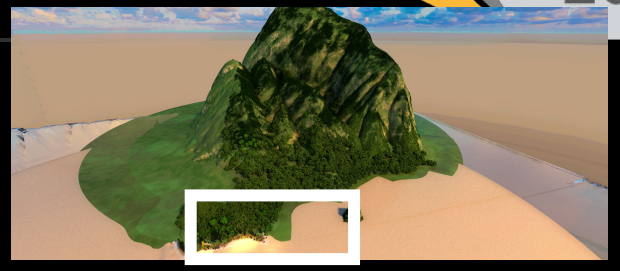


Production Content: “Moana”

- The actual “shot” view
  - Even top right is only part of whole…



Production Content: “Moana”



All those twigs, leaves, etc, are all instances

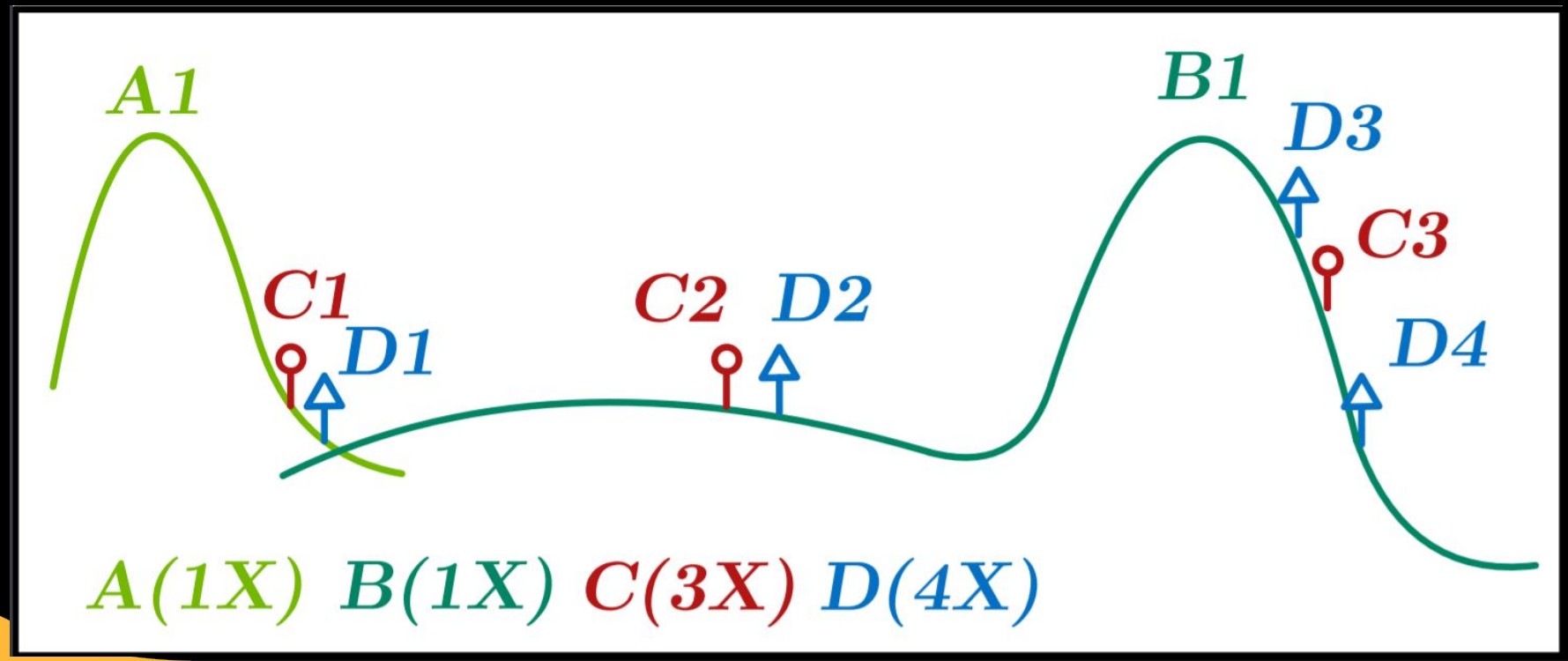
Production Content: “Moana”

- Issues:
  - Some meshes span entire model (or large parts of it)
    - ocean, ocean floor, base of mountain, …
  - Many meshes have instances scattered all over entire model
- Plus: Restrictions on what/how we can split things
  - Instanced Meshes cannot easily be split at all
  - Meshes have shading data (eg: baked textures)

Let’s do a little thought experiment …

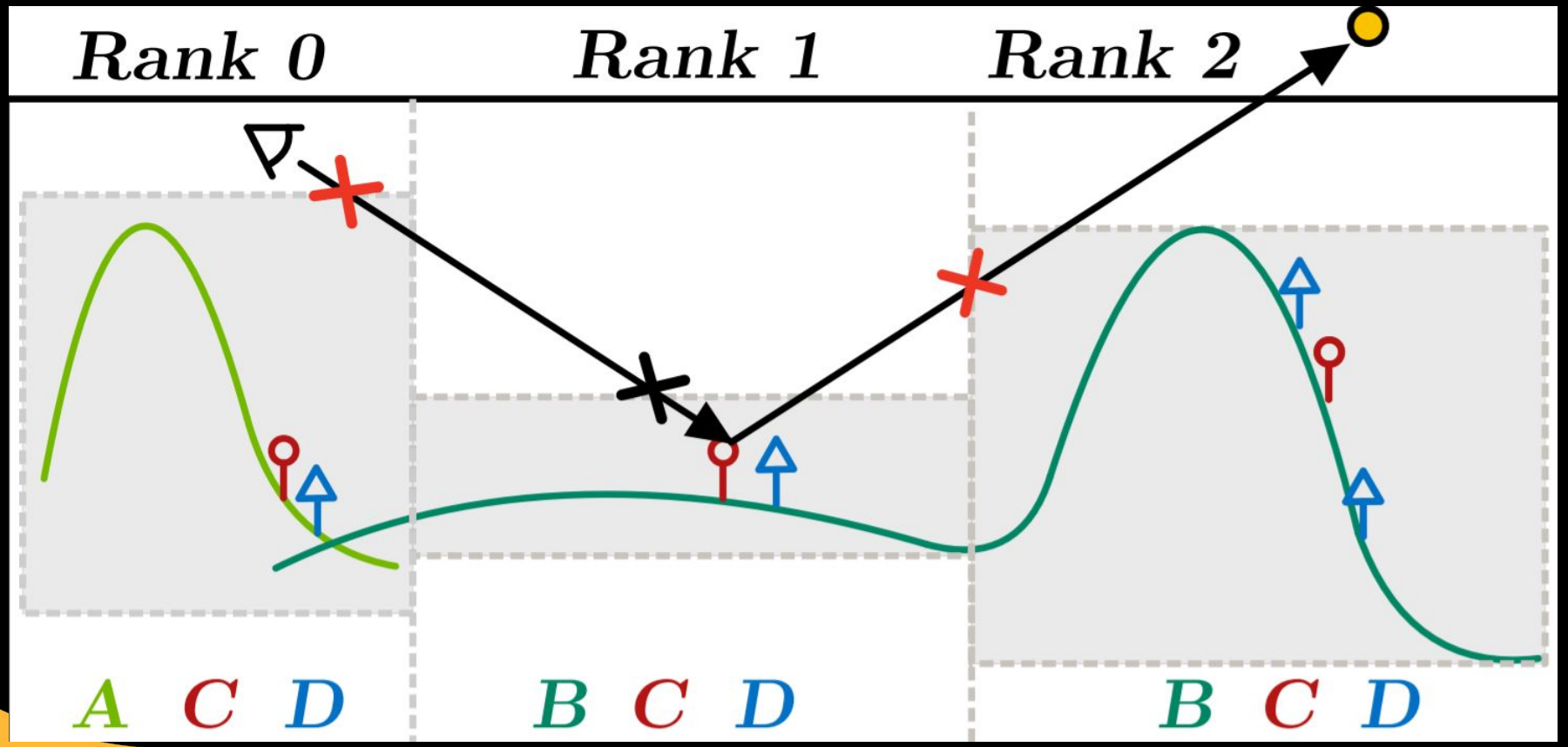
DPRT and Spatial Partition ...

- Consider a “Moana-like” 2D sketch scene
  - One “rock” (A), one “mountain” (B), two types of trees (C,D), several instances of C and D



DPRT and Spatial Partition ...

- Now let's do some (SAH-like?) spatial partitioning and trace some rays simple, trivial to traverse this

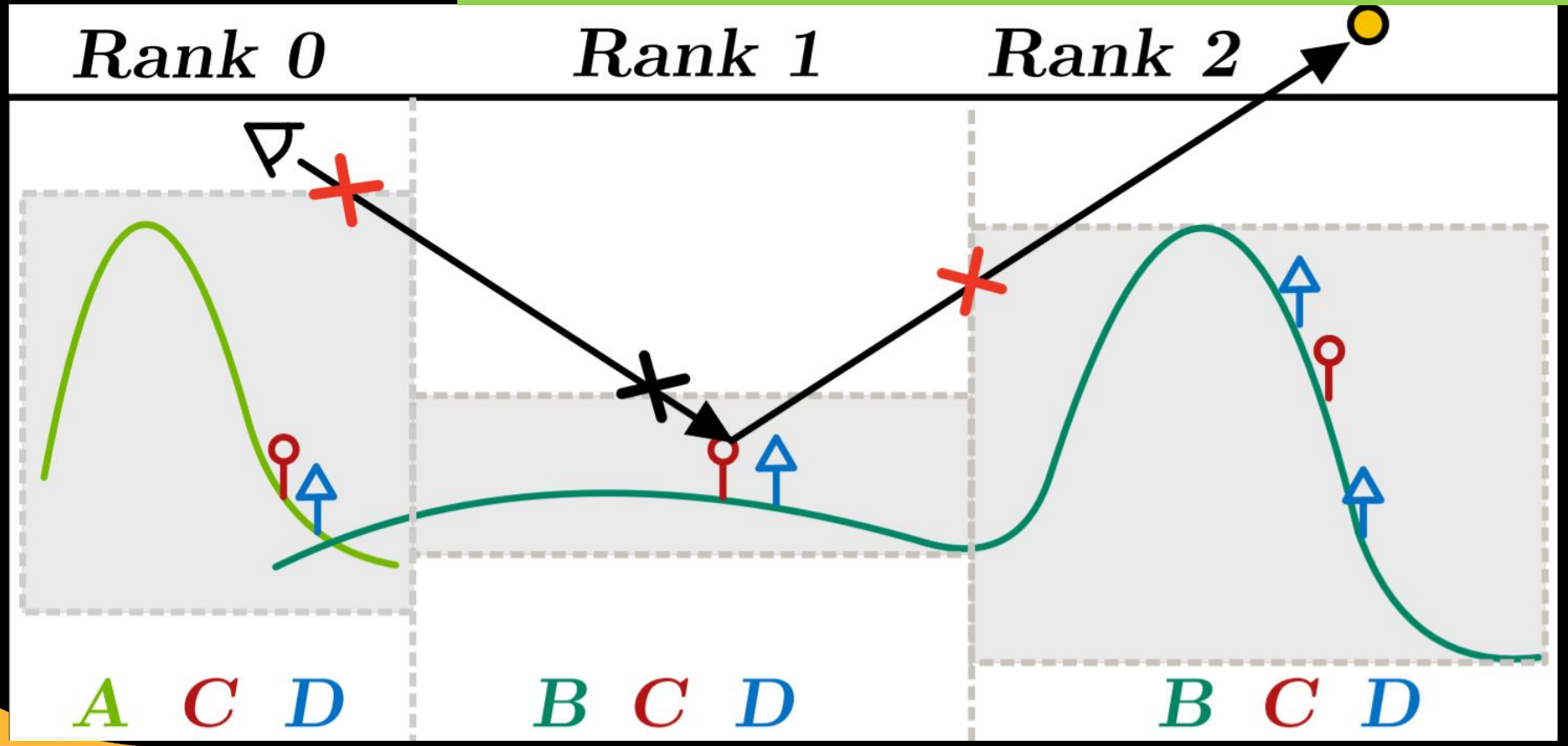


DPRT and Spatial Partition …

- Now let's do some (S
  and trace some rays

BUT: Every rank has (almost) every object!

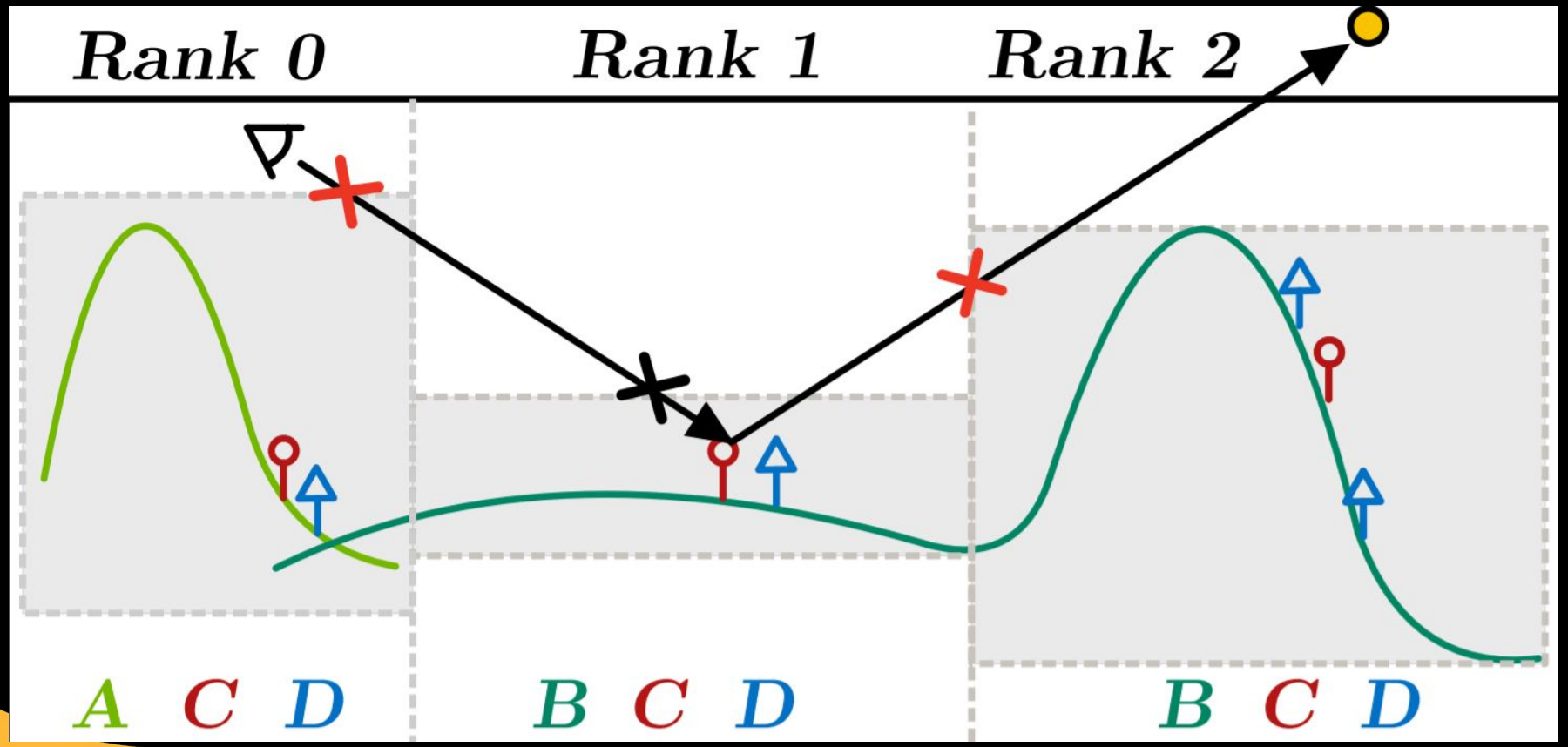
(Note bug in that sketch: R0 actually also has a copy of the green mesh, so has *every* mesh that the un-split model has!)



DPRT and Spatial Partition …

- Now let's do some (SAH-like?) spatial partitioning and trace some ray

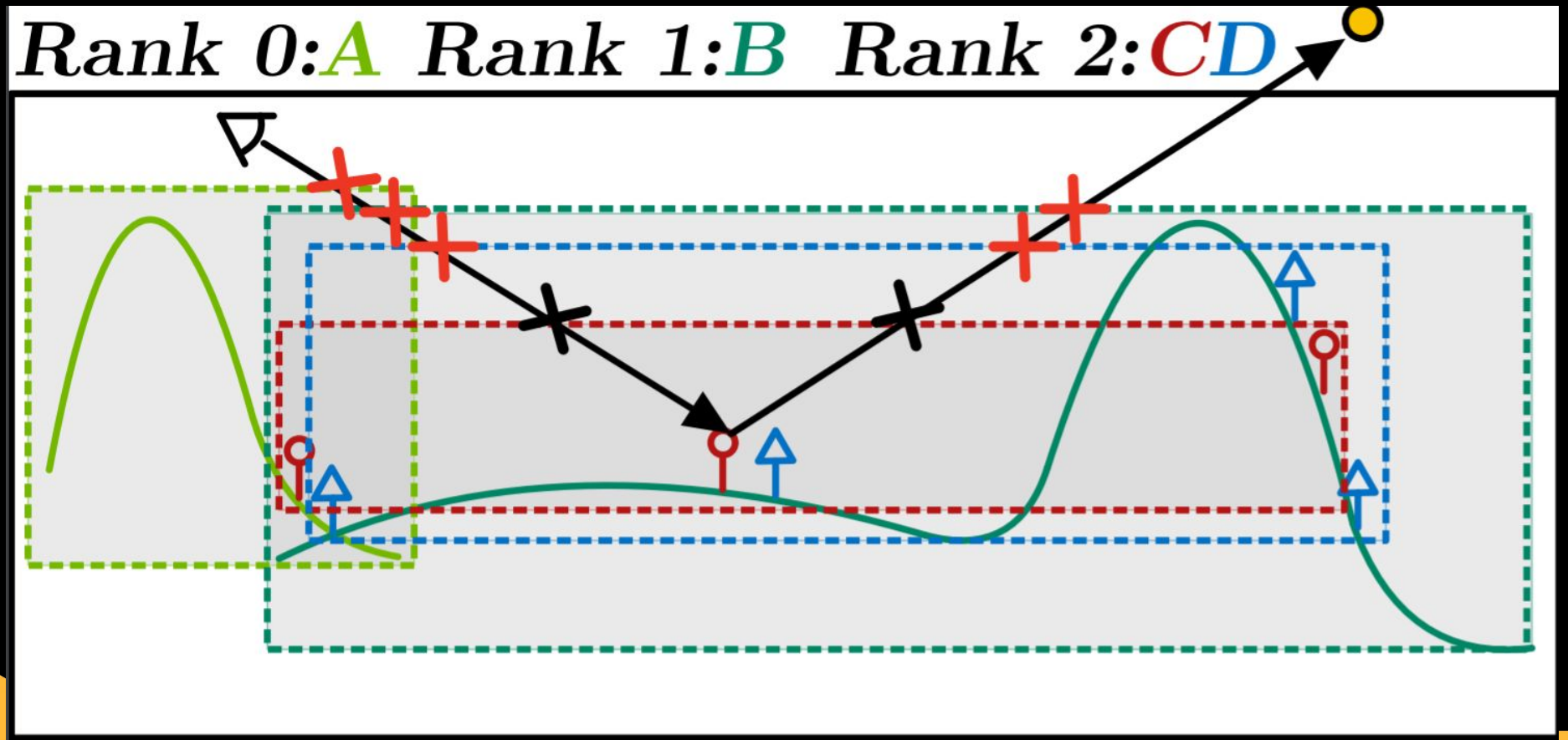
Observation 2: both rays touch two ranks even though they're not even close to R0 or R2!



So, ... What about Object-space Part.?

So, … What about Object-space Part.?

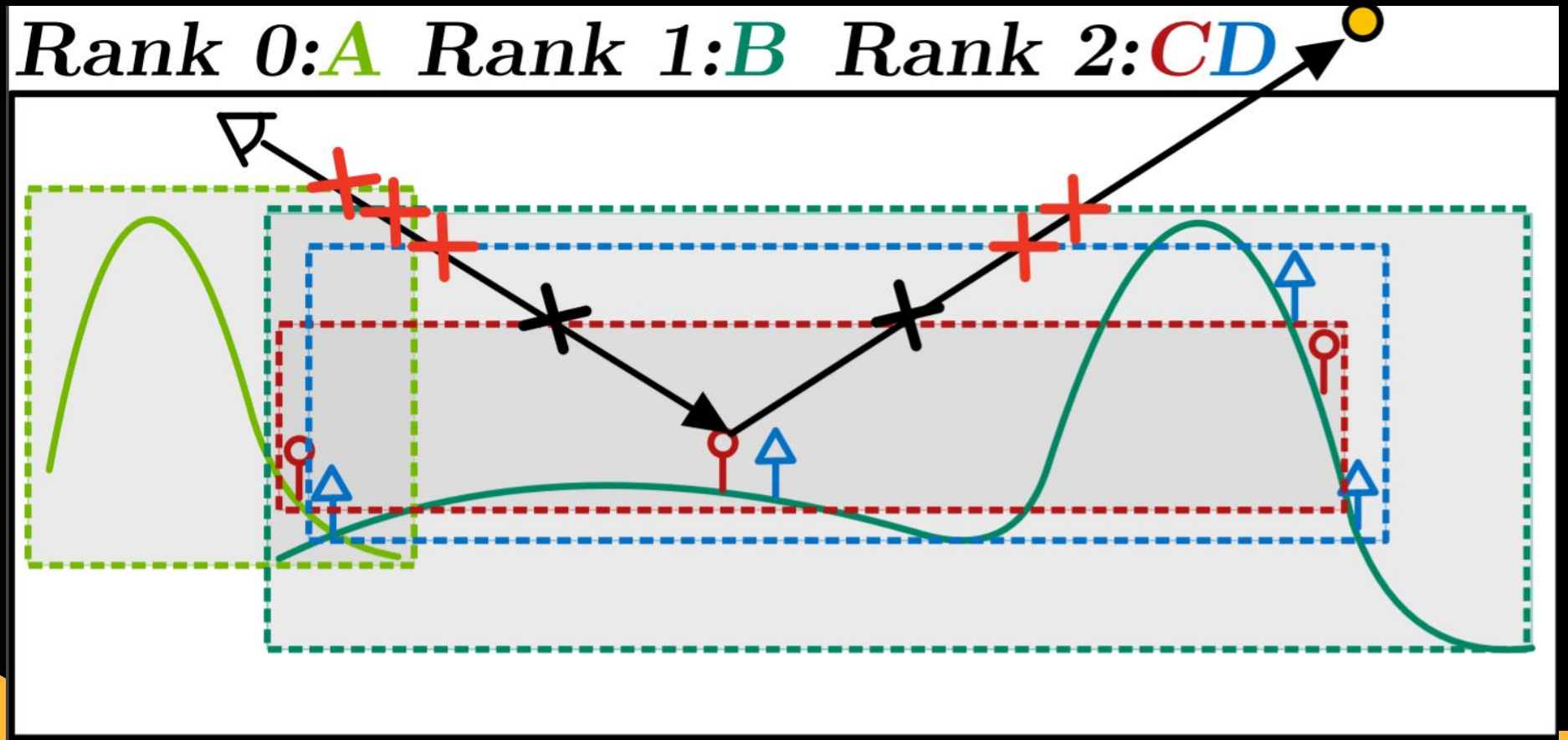
- Partition by object type rather than by space
- Let's put A to rank 0, B to rank 1, and rest to rank 2



So, … What about Obj

- Partition by object type rath
- Let's put A to rank 0, B to rank 1, and rest to rank 2

Much better for partitioning
(▫ every mesh on only one node)
… BUT:
a) How do we even traverse this!?
b) *Really* bad for forwarding: *every ray goes every rank*
(maybe there's a reason nobody ever did that)

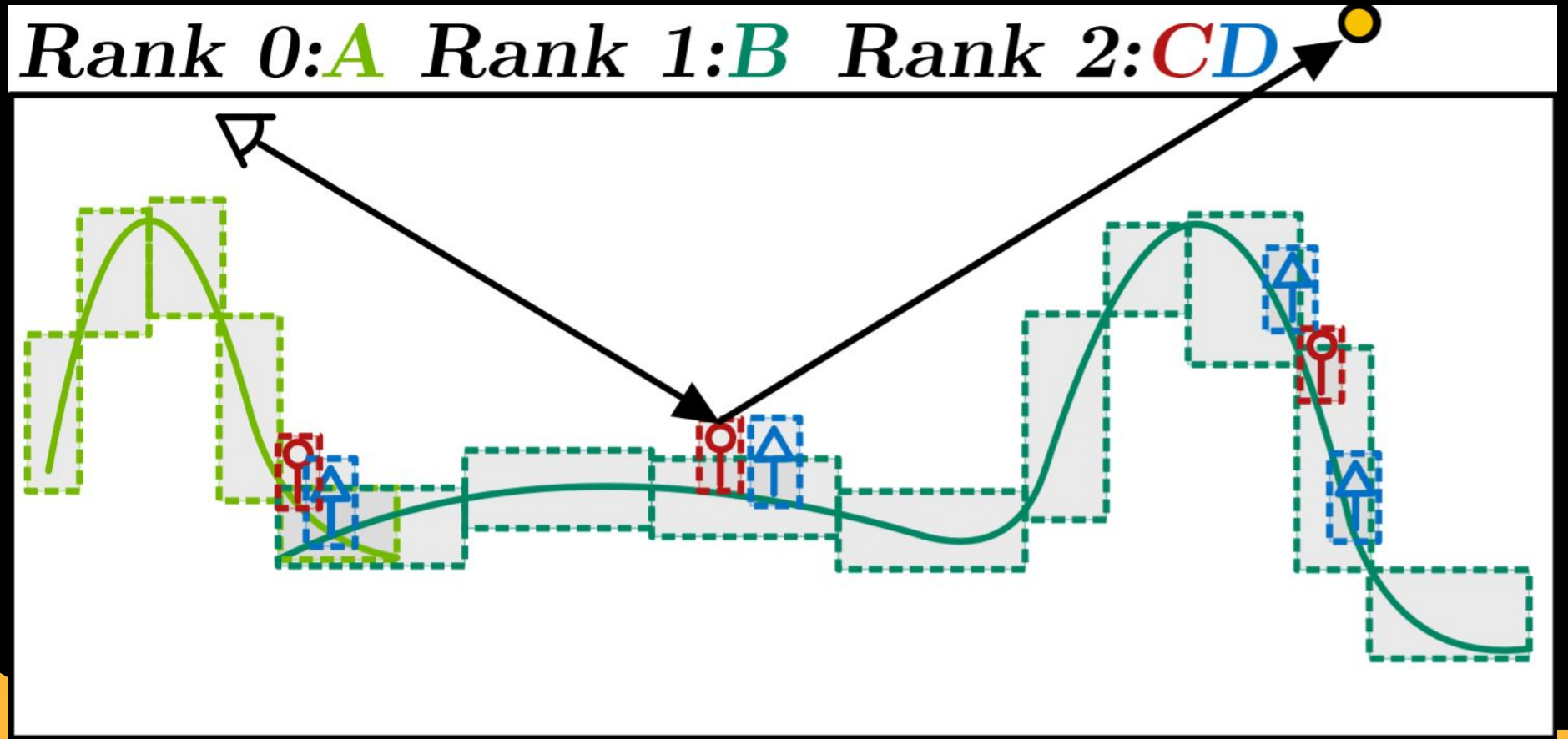


Applying spatial splits & re-braiding eqiv.

- Spatial splits & re-braiding: represent large meshes or instances with multiple smaller, tighter boxes

Applying spatial splits & re-braiding eqiv.

- Spatial splits & re-braiding: represent large meshes or instances with multiple smaller, tighter boxes



Distributed Ray Traversal w/ Object-space Hierarchies

Distr. Ray Traversal in Object Space

- Key idea, Part 1: More general description of distributed content via “proxies”
  - Proxy = region of space describing remote content
- Every proxy has
  - a) 3-D world-space bounding box (region of space)
  - b) List of node(s) that own data behind that box

Proxies

- Naïve: one proxy / inst, each inst on one rank

Proxies

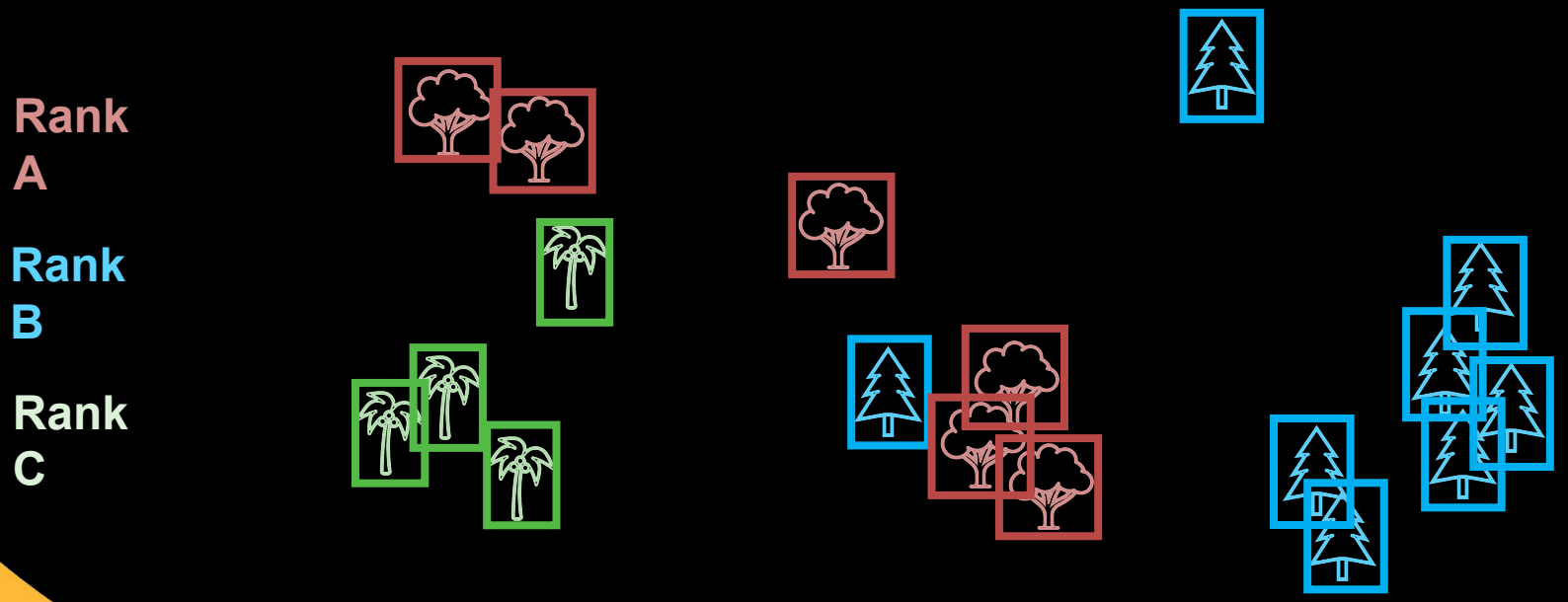
- Naïve: one proxy / inst, each inst on one rank

Proxies

- Naïve: one proxy / inst, each inst on one rank

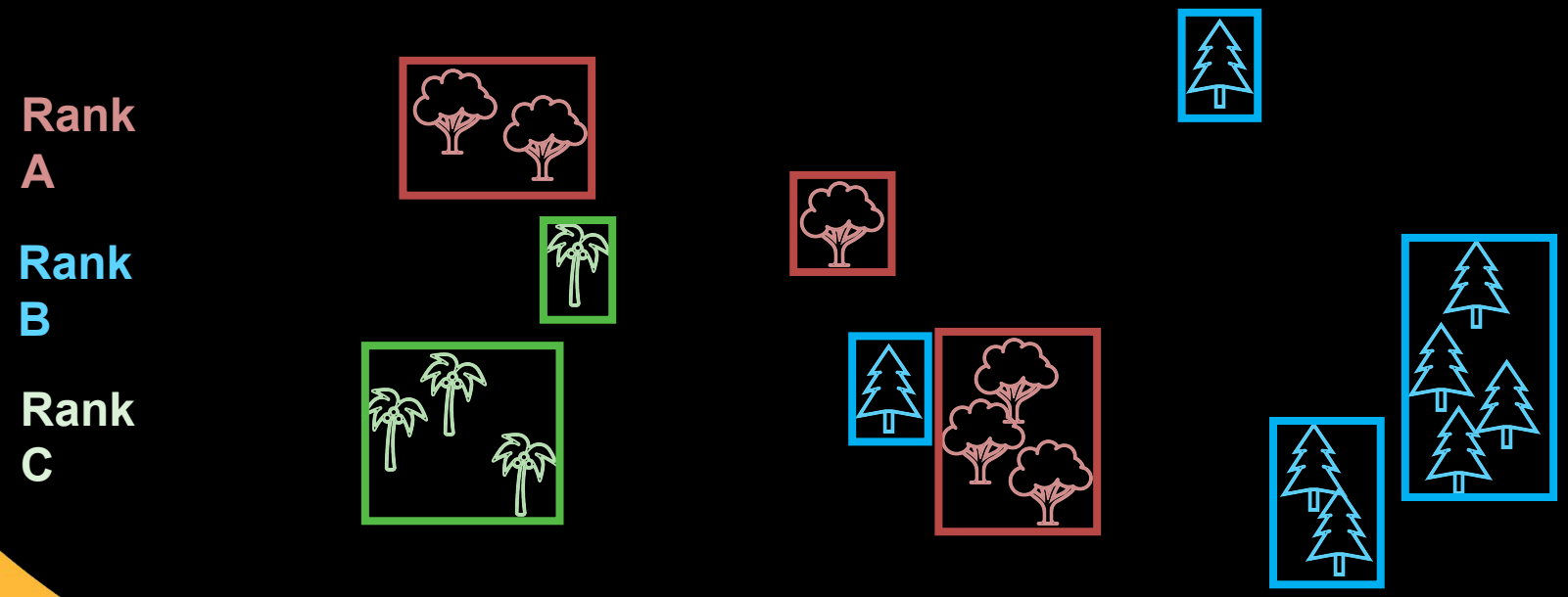
Proxies

- Naïve: one proxy / inst, each inst on one rank



Proxies

- Naïve: one proxy / inst, each inst on one rank
  - But: could also have more than one inst in a proxy



Proxies

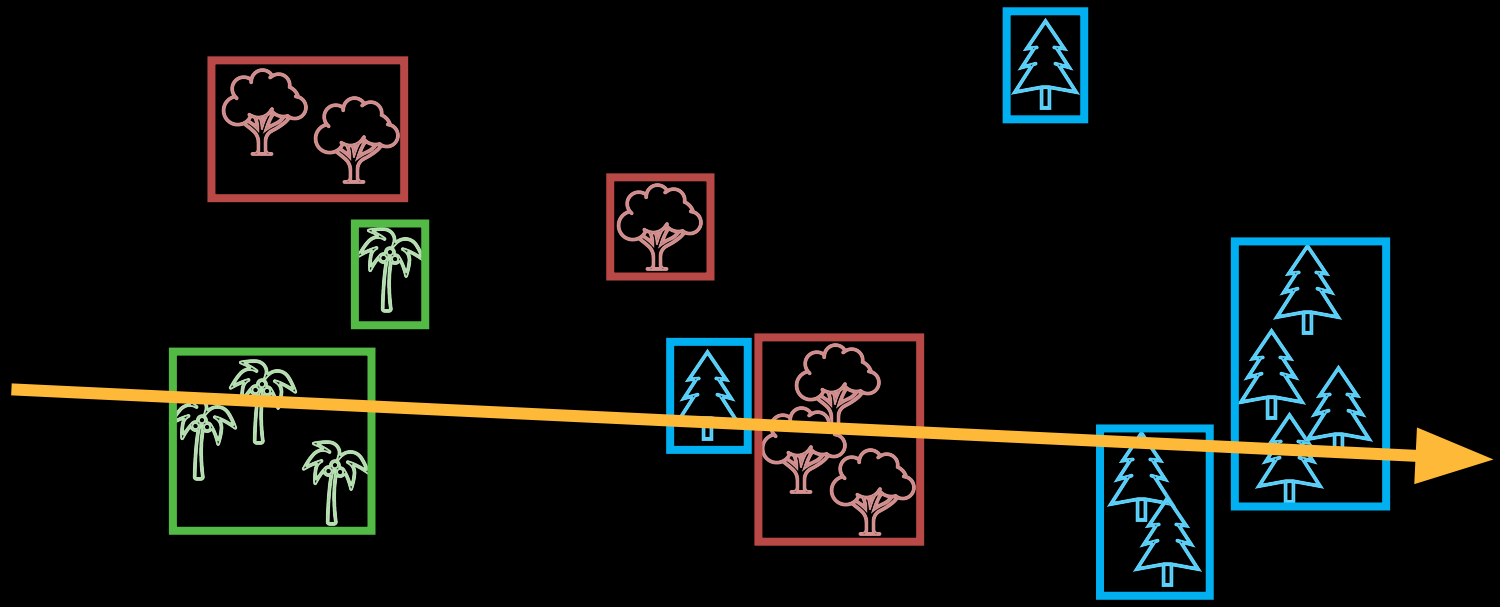
- Naïve: one proxy / inst, each inst on one rank
  - But: could also have more than one inst in a proxy…
  - … or more than one proxy per any geom

Proxies

- Naïve: one proxy / inst, each inst on one rank
  - But: could also have more than one inst in a proxy…
  - … or more than one proxy per any geom

Distributed Ray Traversal

- First idea: traverse from proxy to proxy (stack-free?)



Distributed Ray Traversal

- First idea: traverse from proxy to proxy (stack-free?)

- But: *Lots* of issues with this …
  - Stability (not going into that here)
  - Possibly *tons* of proxies along a ray
  - Rays can actually go back-and-forth between same node(s)

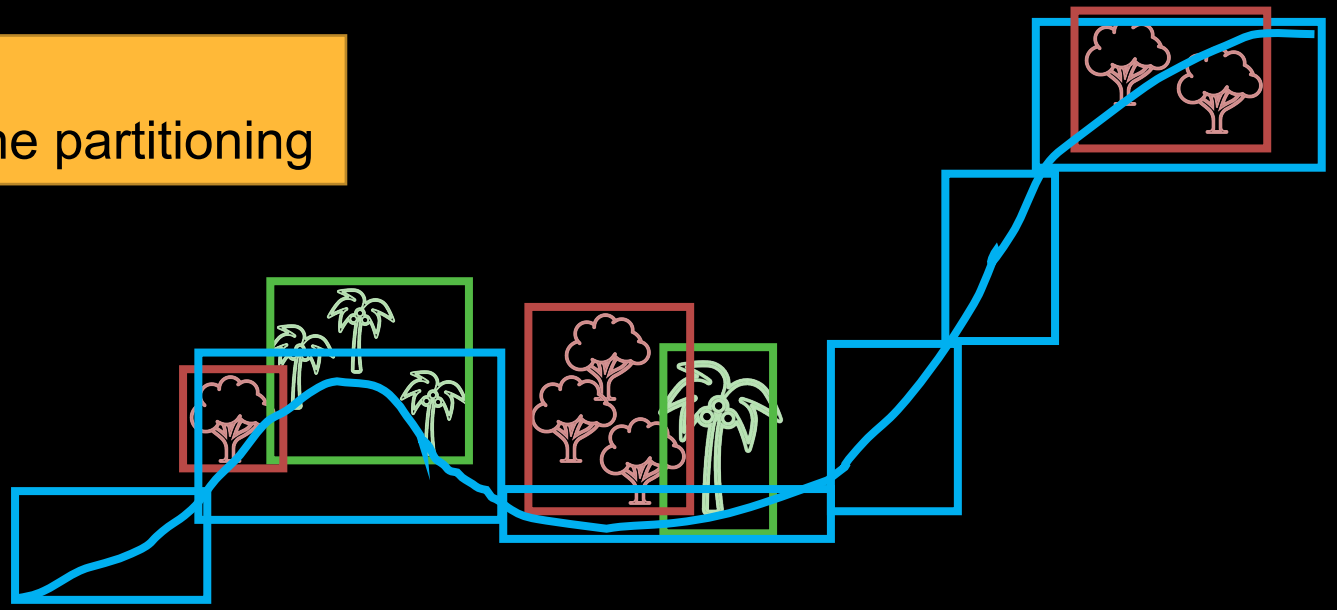
Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

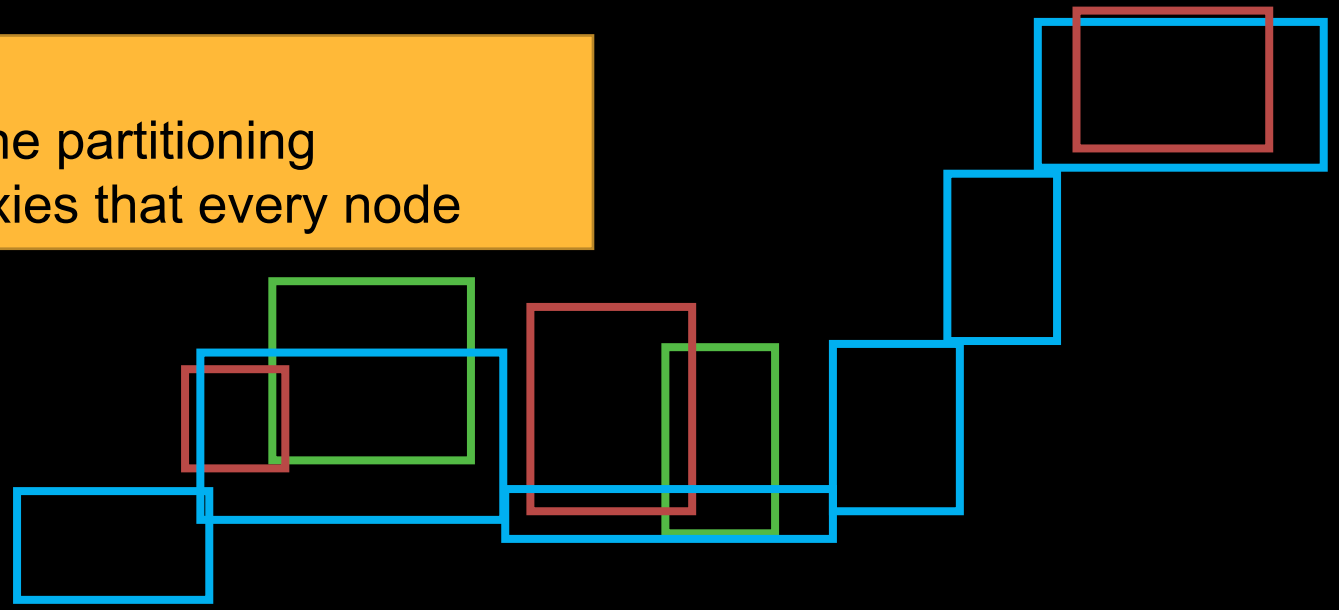
Example:
- Take this scene partitioning



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

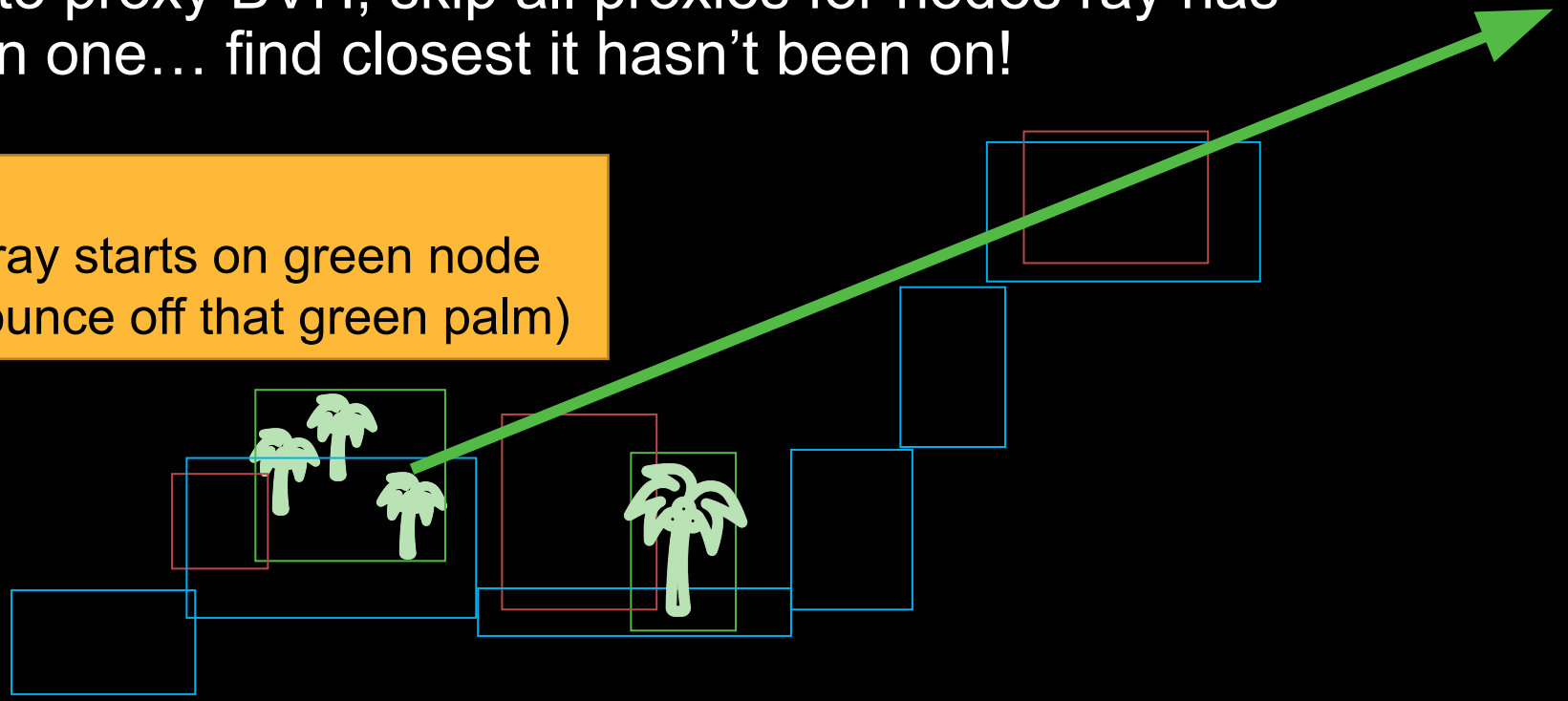
Example:
- Take this scene partitioning (ie, these proxies that every node



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

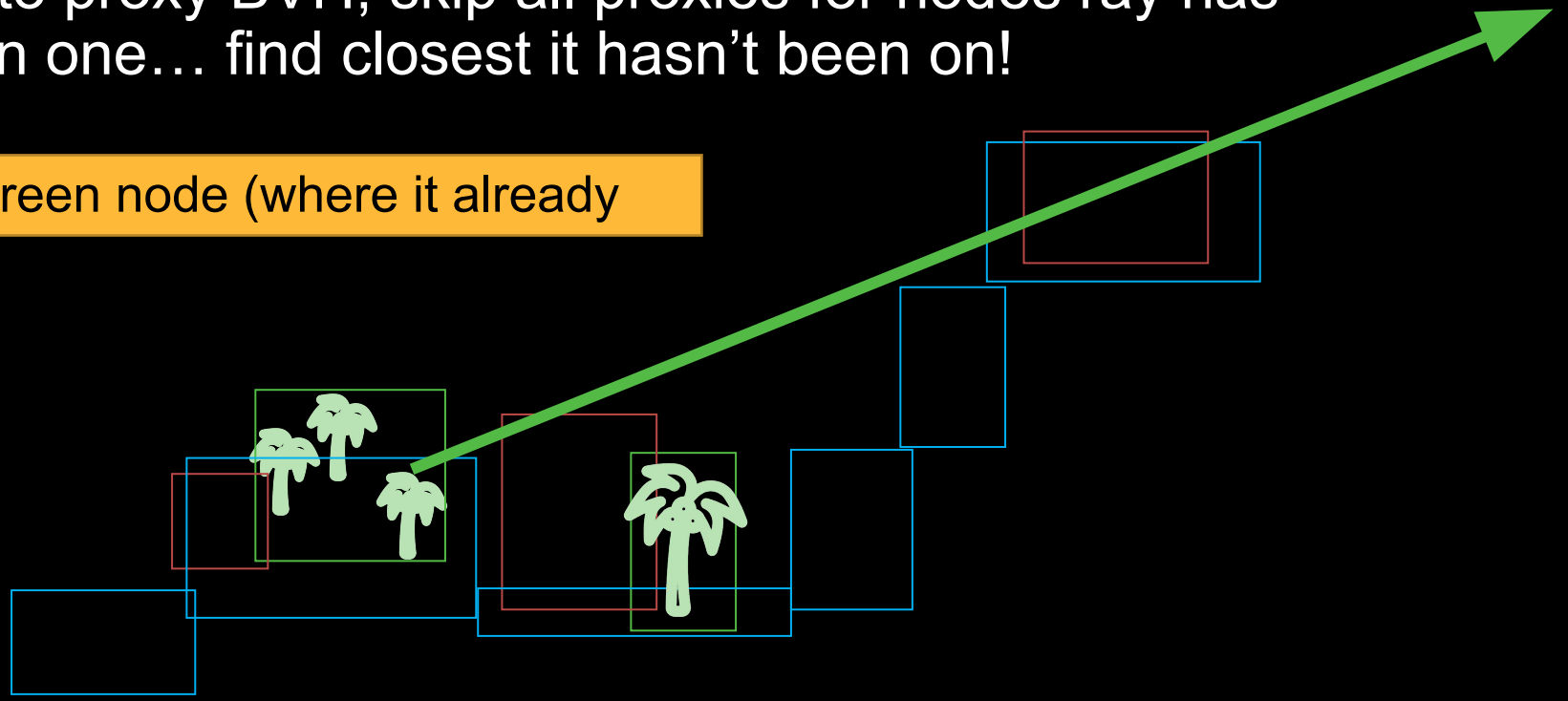
Example:
- Assume new ray starts on green node (eg, diffuse bounce off that green palm)



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

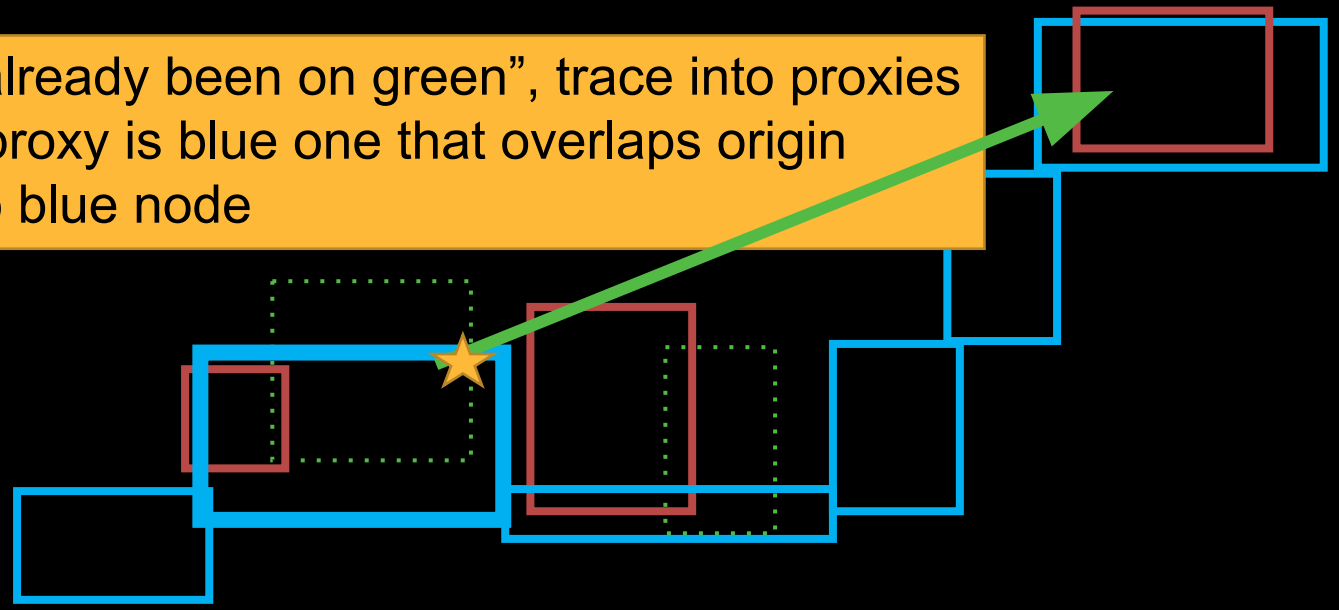
1) Trace ray on green node (where it already



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

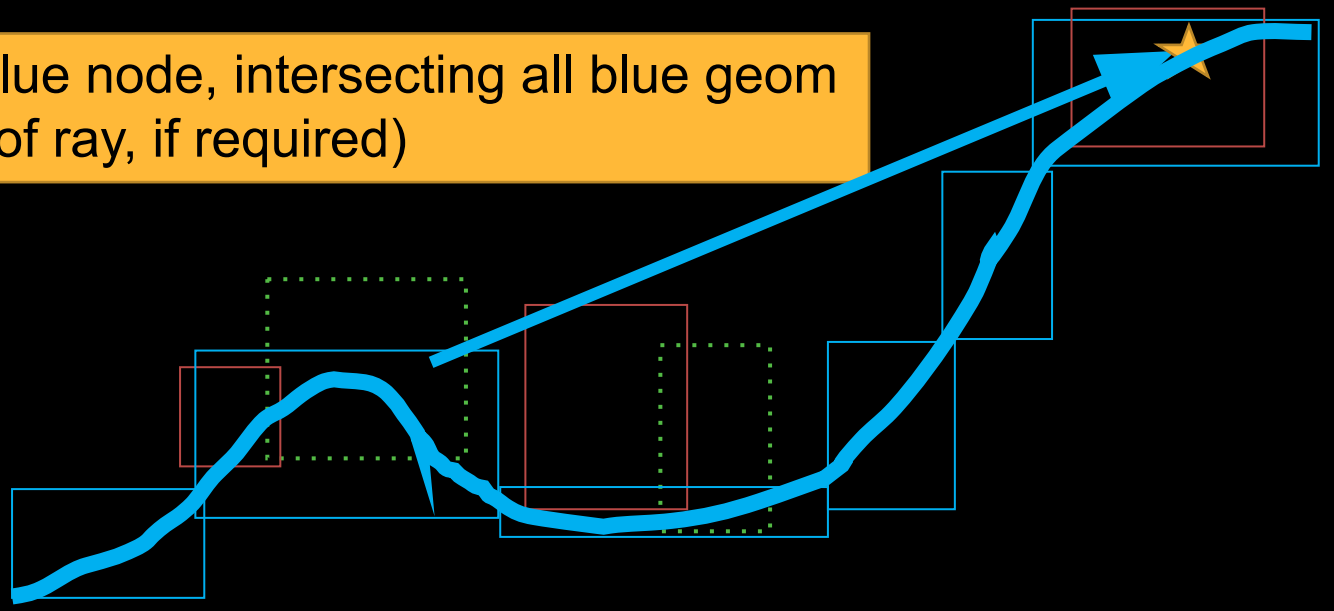
2) Mark ray as “already been on green”, trace into proxies
- Closest “live” proxy is blue one that overlaps origin
- Needs to go to blue node



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

3) Trace ray on blue node, intersecting all blue geom (update hit point of ray, if required)



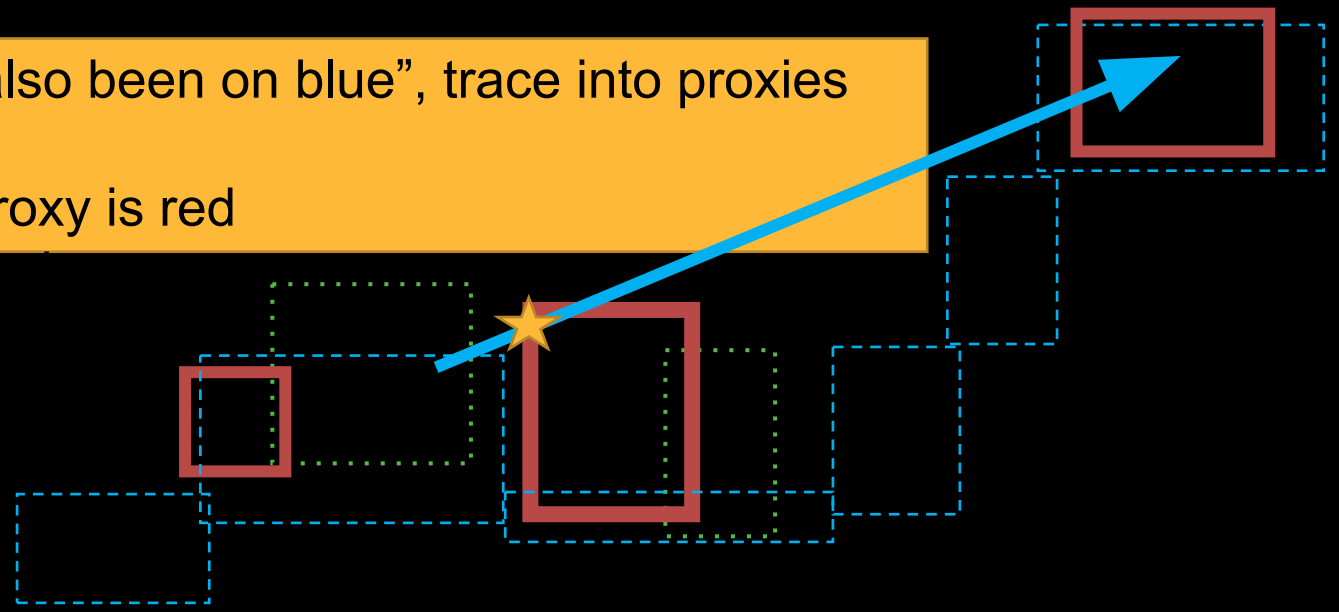
Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

4) Mark ray as “also been on blue”, trace into proxies

…

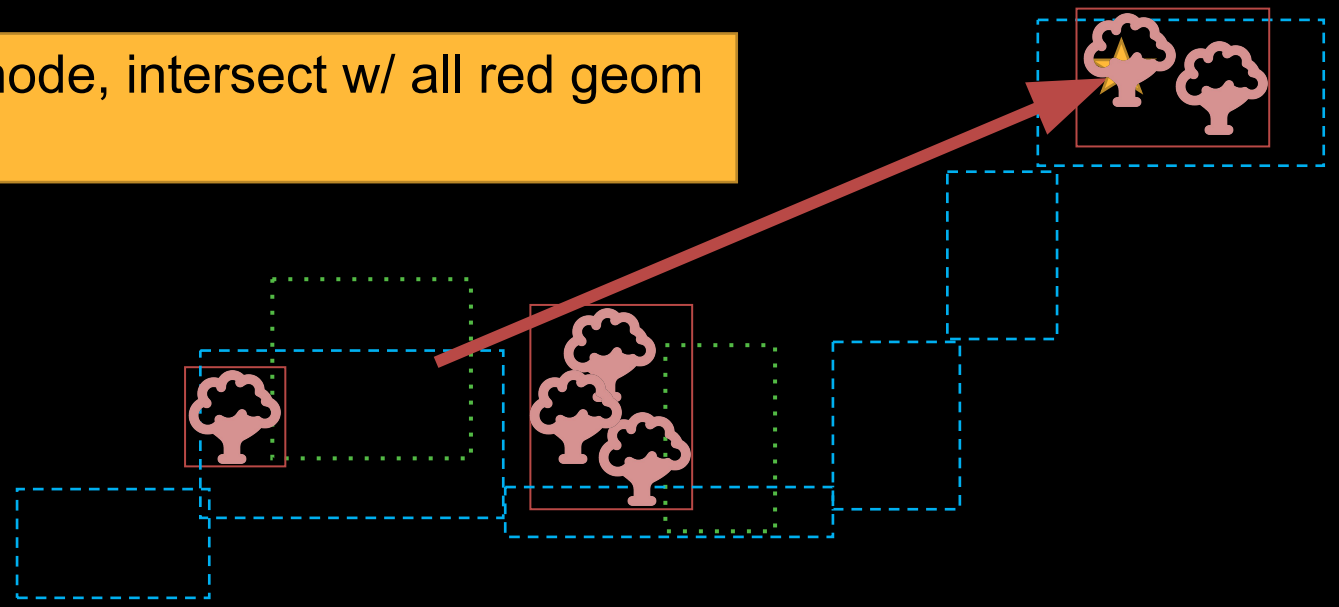
 Closest live proxy is red



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

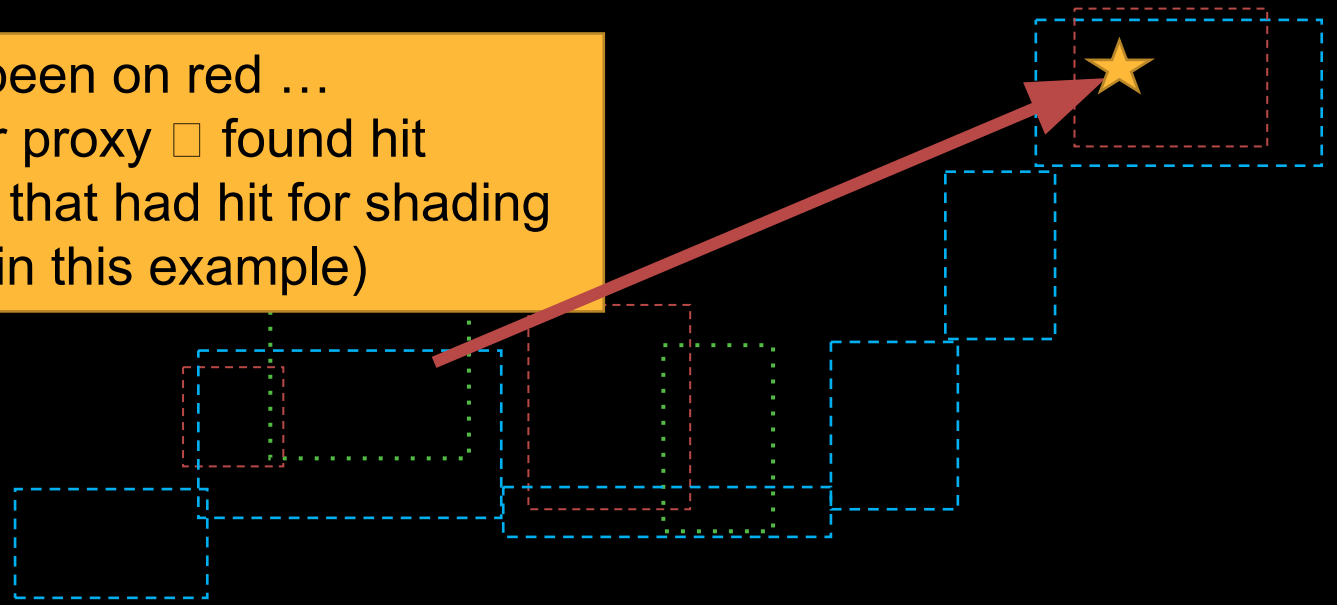
5) Trace on red node, intersect w/ all red geom (update hit point)



Distributed Ray Traversal

- Better: always get next “not yet travrsed node” proxy
  - Build BVH over proxies
  - Have each ray track which nodes it’s already been on
  - Trace ray into proxy BVH, skip all proxies for nodes ray has already been one… find closest it hasn’t been on!

6) Mark as also been on red …
 no other closer proxy found hit
  Move to node that had hit for shading (already on it in this example)



Paper: Putting that idea in practice …

- How to …
  - …do the actual scene partitioning
  - …track previously visited nodes
  - … build a full system w/ that
  - …
- No details here – see the paper …

Results (Teaser)

- Object-space splits way better for partitioning
  - Generally (way) less data per node (no repl, duh!)

  - Can reach same memory budget w/ less splits/nodes!
    - Cheaper *and* faster
  - Spatial often can't get below certain budget *at all*

  - And way more *general* where content can be

Results (Teaser)

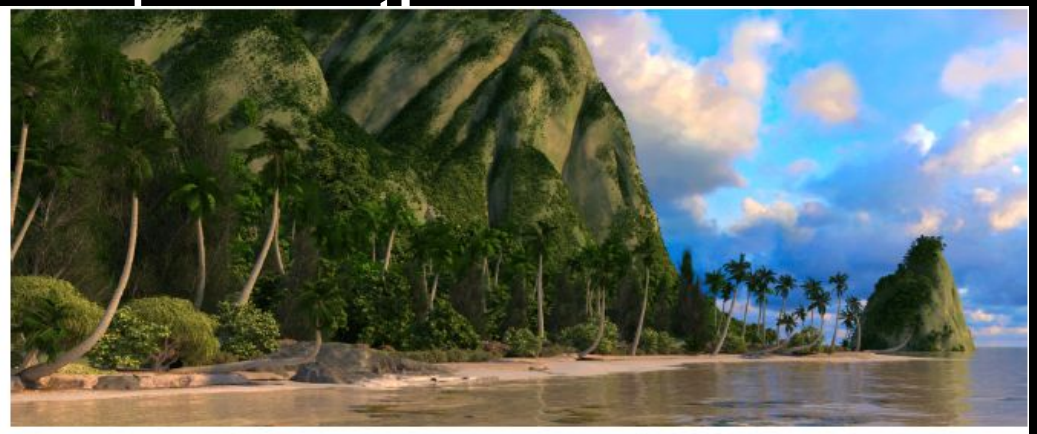
- Object-space splits way better for partitioning
- With good partitioning, also *faster* than spatial
  - Tight proxies = little forwarding = faster

Results (Teaser)

- Object-space splits way better for partitioning
- With good partitioning, also *faster* than spatial
- Proxies can *also* represent classical spatial partitions
  - Not “either/or”, but super-set of state of the art

Results (Teaser)

- Object-space splits way better for partitioning
- With good partitioning, also *faster* than spatial
- Proxies can *also* represent classical spatial partitions
-



(Hardware: 4 worker nodes w/ 2×RTX8000, low-end head node, 10-Gigabit Ethernet, screen size 2560 × 1080)

PBRT *landscape*: max 3.7 GB GPU usage, 6.2 FPS (1 path/pixel)	Disney Moana *island*: max 25 GB GPU usage, 7.9 FPS (1 path/pixel)
30 K instances, 4.3 B inst. triangles, 370 unique meshes, 500 MB image textures	39 M instances, 41 B inst. triangles, 7 M unique meshes, 804 MB baked-PTex txt.

The End …

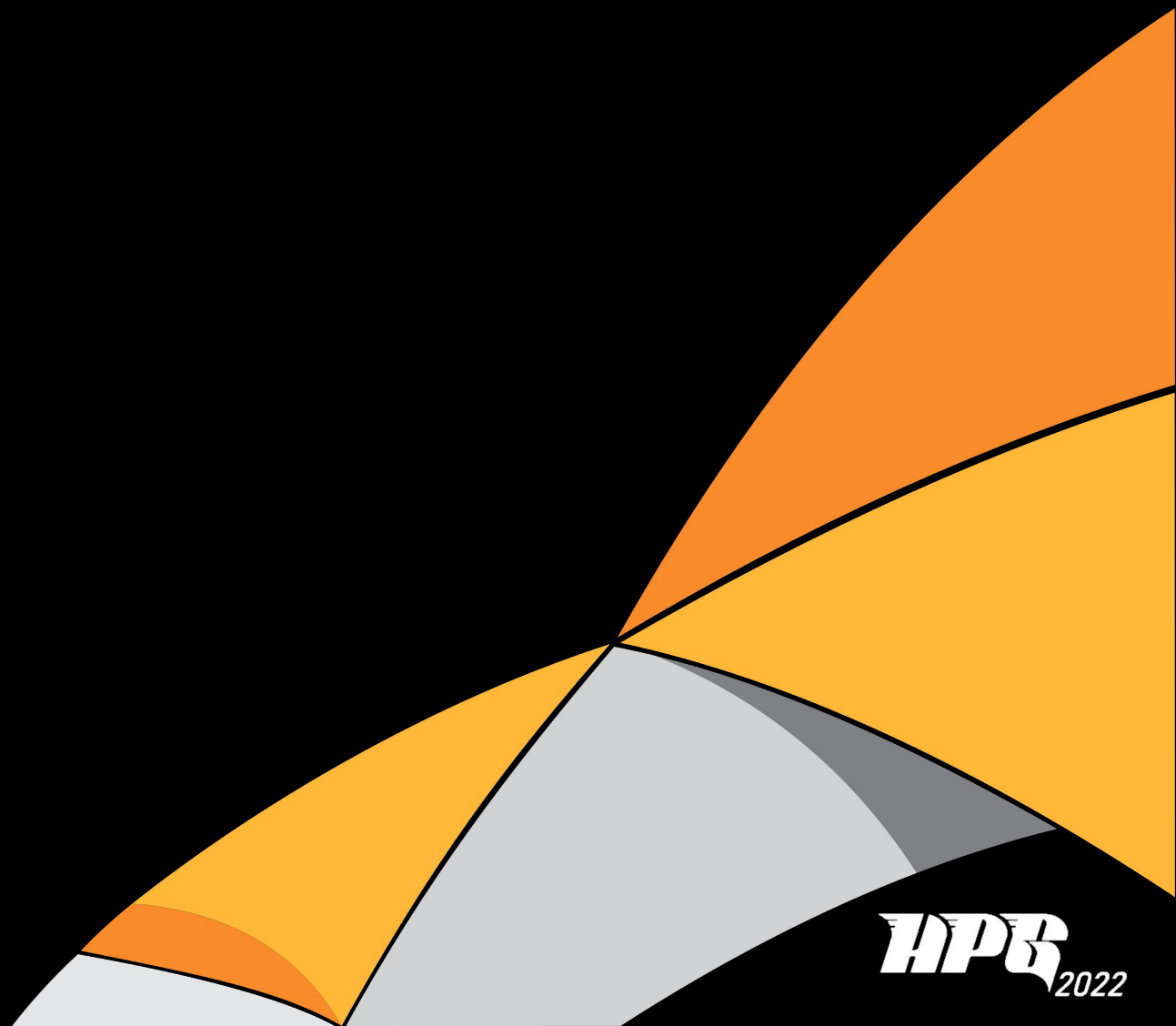
- Many details I glossed over □ paper
- Some remaining issues not solved yet … □ paper
- Questions?

(Hardware: 4 worker nodes w/ 2×RTX8000, low-end head node, 10-Gigabit Ethernet, screen size 2560 × 1080)

PBRT *landscape*: max 3.7 GB GPU usage, 6.2 FPS (1 path/pixel)
30 K instances, 4.3 B inst. triangles, 370 unique meshes, 500 MB image textures

Disney Moana *island*: max 25 GB GPU usage, 7.9 FPS (1 path/pixel)
39 M instances, 41 B inst. triangles, 7 M unique meshes, 804 MB baked-PTex txt.

Out-takes



Proxies

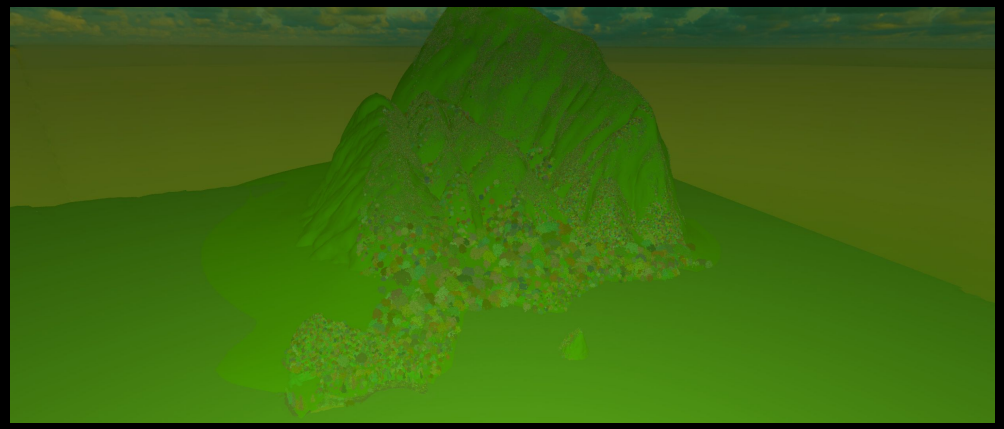
- Naïve: one proxy / inst, each inst on one rank
  - But: could also have more than one inst in a proxy…
  - … or more than one proxy per any geom…
  - … or even replicate some insts to more than one rank
- Eventually: *Very* general
  - 3D box that contains “something”…
    - A mesh, an instance, part of a mesh or instance, …. *something*
  - … and that something is available on any of the listed node(s)

Paper: Putting that idea in practice …

- How to do the actual scene partitioning
  - Several options in the paper (incl one “good” one)
  - But: Certainly not fully solved problem yet
    - Surprisingly tricky problem …
    - Anybody looking for a challenging graphics problem?
- Build a system with GPUs, OptiX, CUDA, MPI, …
  - Data parallel “closest hit” operator (what we sketched)
  - Wavefront path tracer (on top of that)
- Performance evaluation and discussion

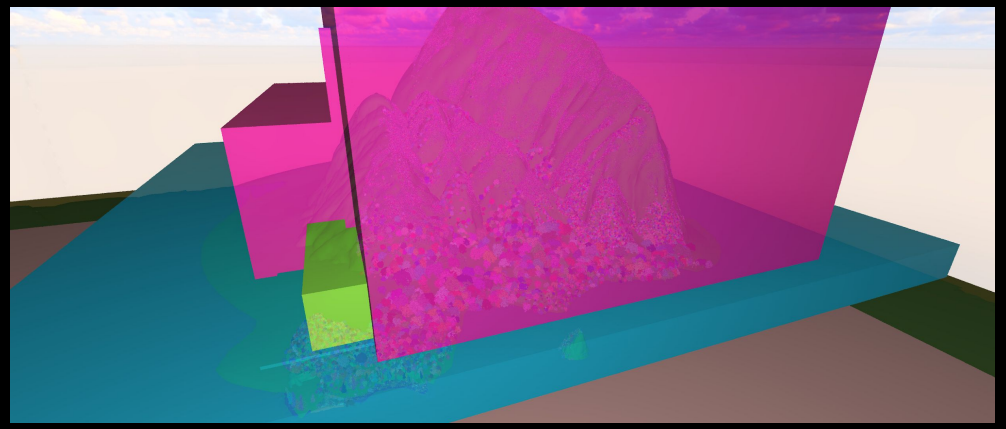
 No time here for details: go read the paper!

Proxies in practice (Moana)



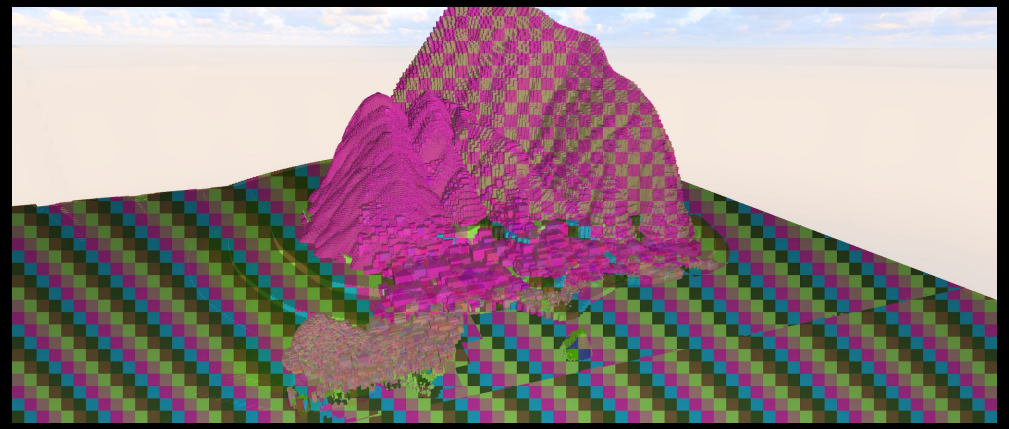
Proxies – spatial only

- domain boxes are *huge:* lots of forwards



Proxies – object hier. naïve

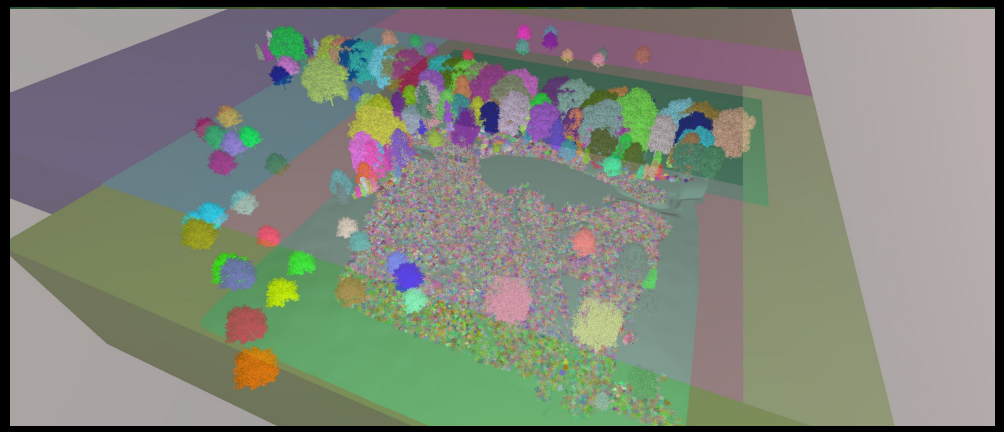
- some proxies still huge: lots of overlap, lots of fwds



Proxies – “best” (striped means “replicated”)

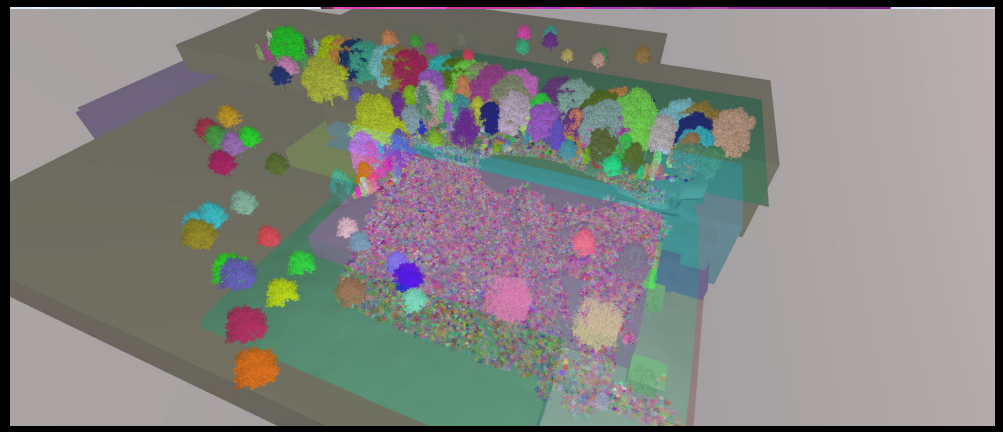
- first proxy is often the correct node already

Proxies in practice (PBRT Landscape)



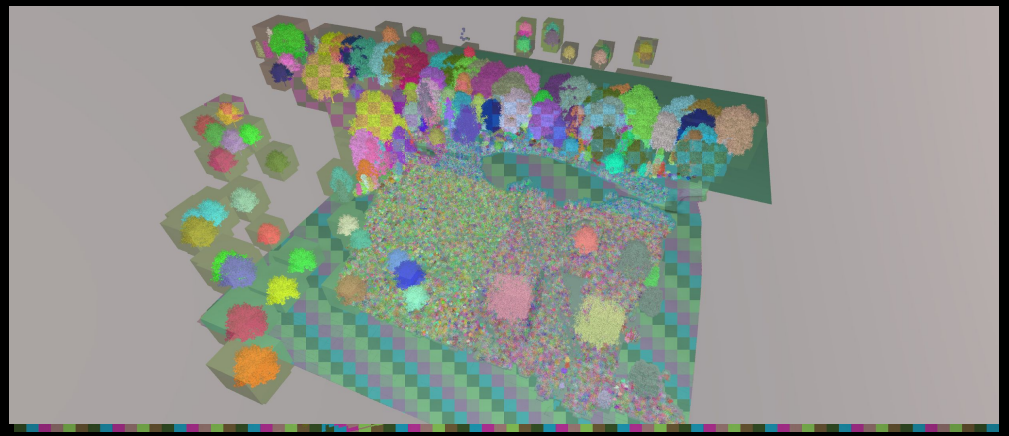
Proxies – spatial only

- domain boxes are *huge:* lots of forwards



Proxies – object hier. naïve

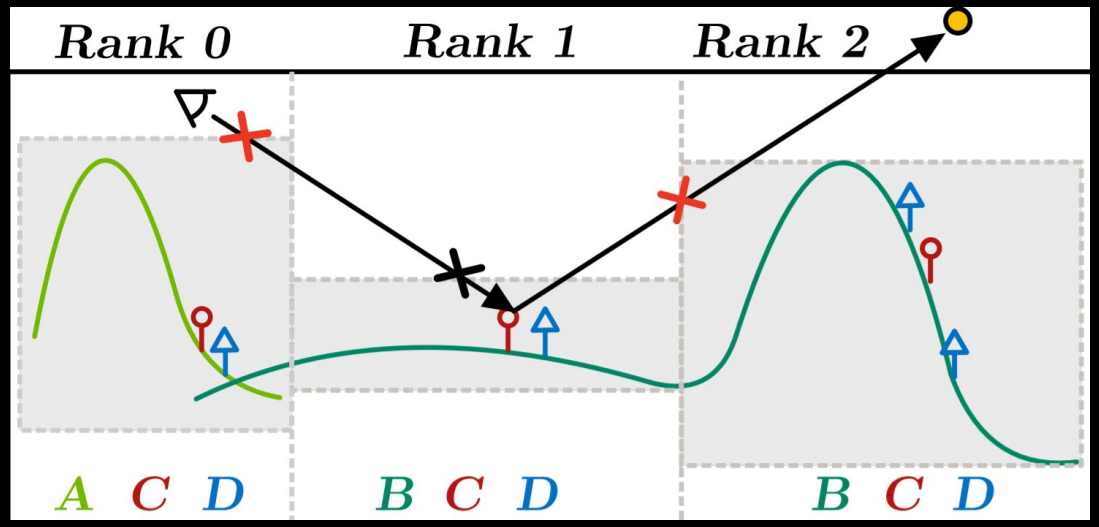
- some proxies still huge: lots of overlap, lots of fwds

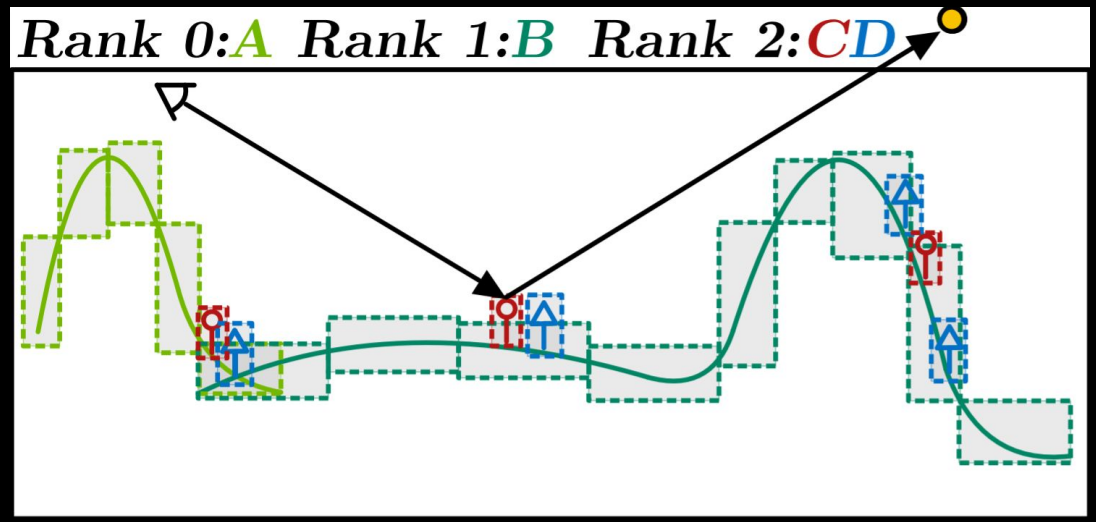


Proxies – “best” (striped means “replicated”)

- first proxy is often the correct node already

Spatial vs Object with Spatial Split?

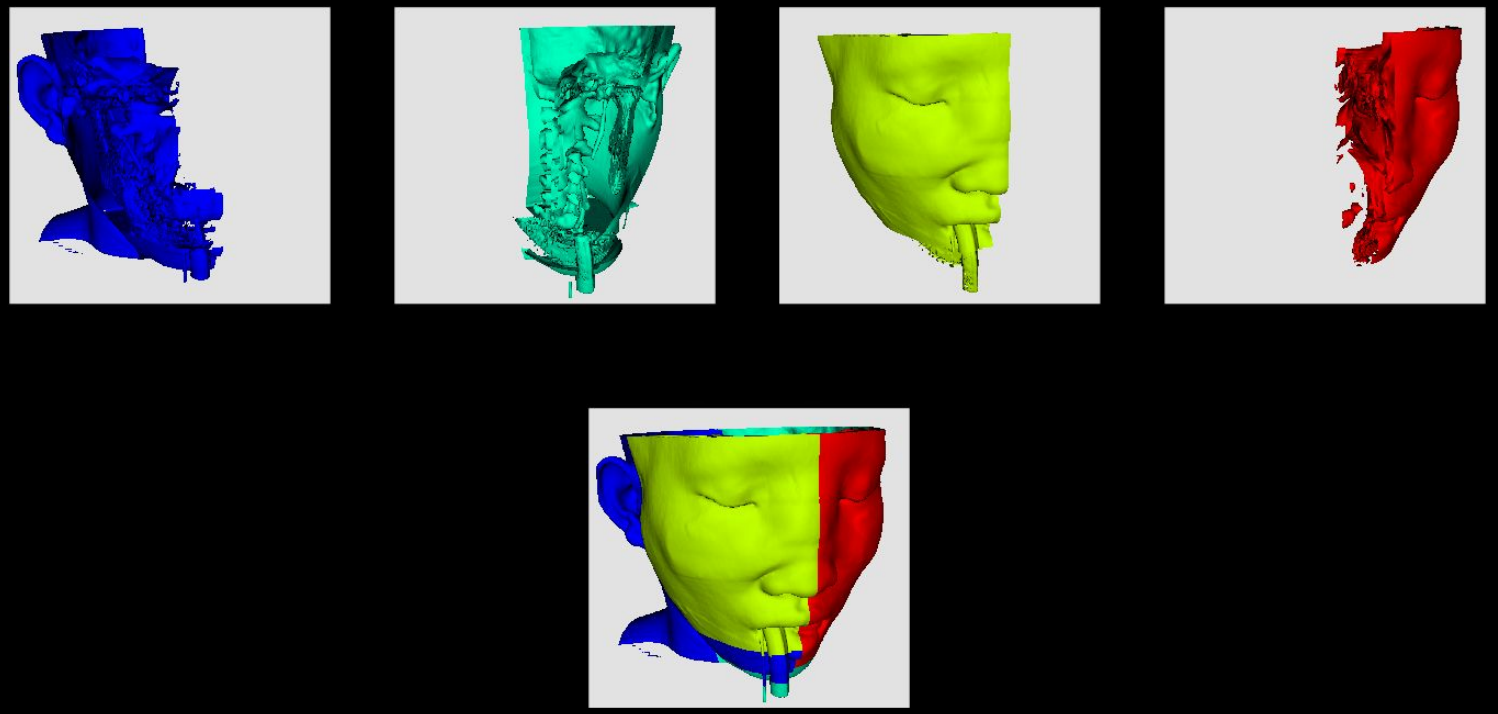




- In this motivational example: actually better in partitioning and ray forwards!
- Question: “How”.
  - In particular, how do we traverse this?

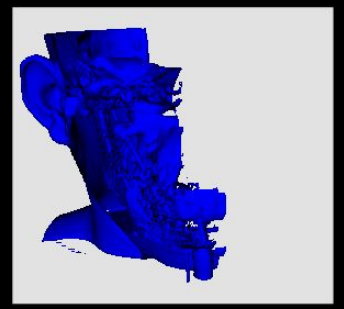
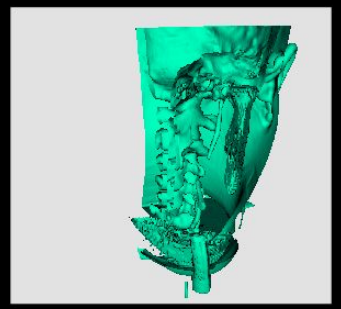
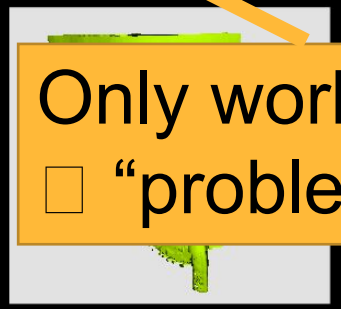
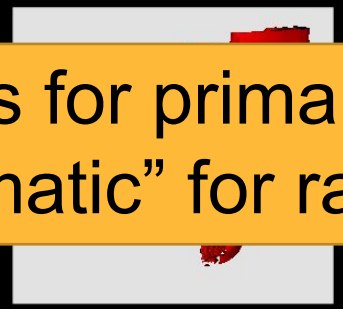
(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
- Most used today: “sort-last” image compositing
  - Easy, fast, well understood, widely used in Sci-Vis

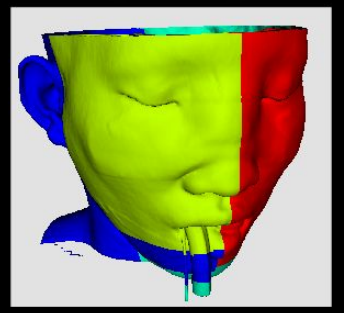


(Data-)Parallel Rendering : How ?

- Data-parallel rendering: mult.nodes work together
- Most used today: “sort-last” image compositing
  - Easy, fast, well understood, widely used in Sci-Vis

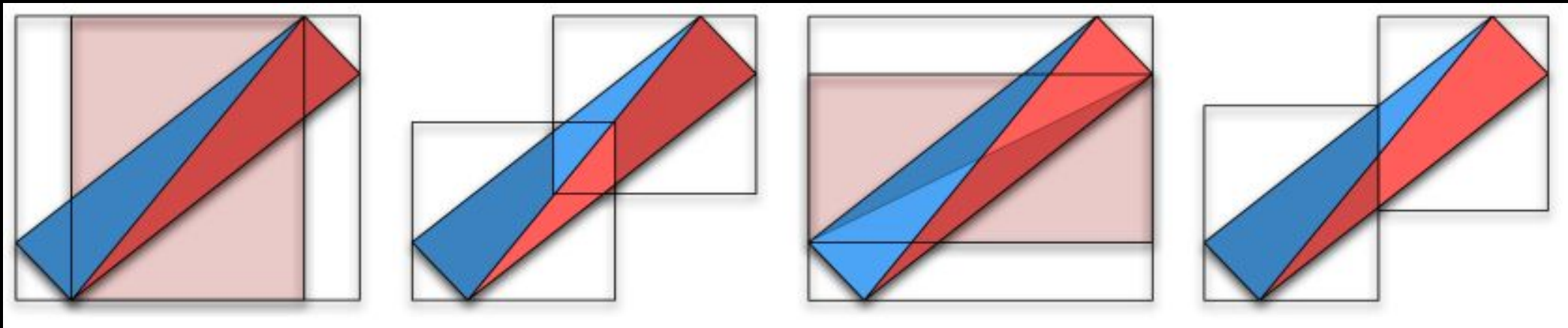





Only works for primary rays
 “problematic” for ray/path tracing



Huh – overlap of large objects …

- Same problem as large tris in BVH building
  - E.g.: Stich et al, Spatial Splits in Bounding Volume Hierarchies:
    - don’t split object itself, but represent with multiple, tighter-fitting boxes!



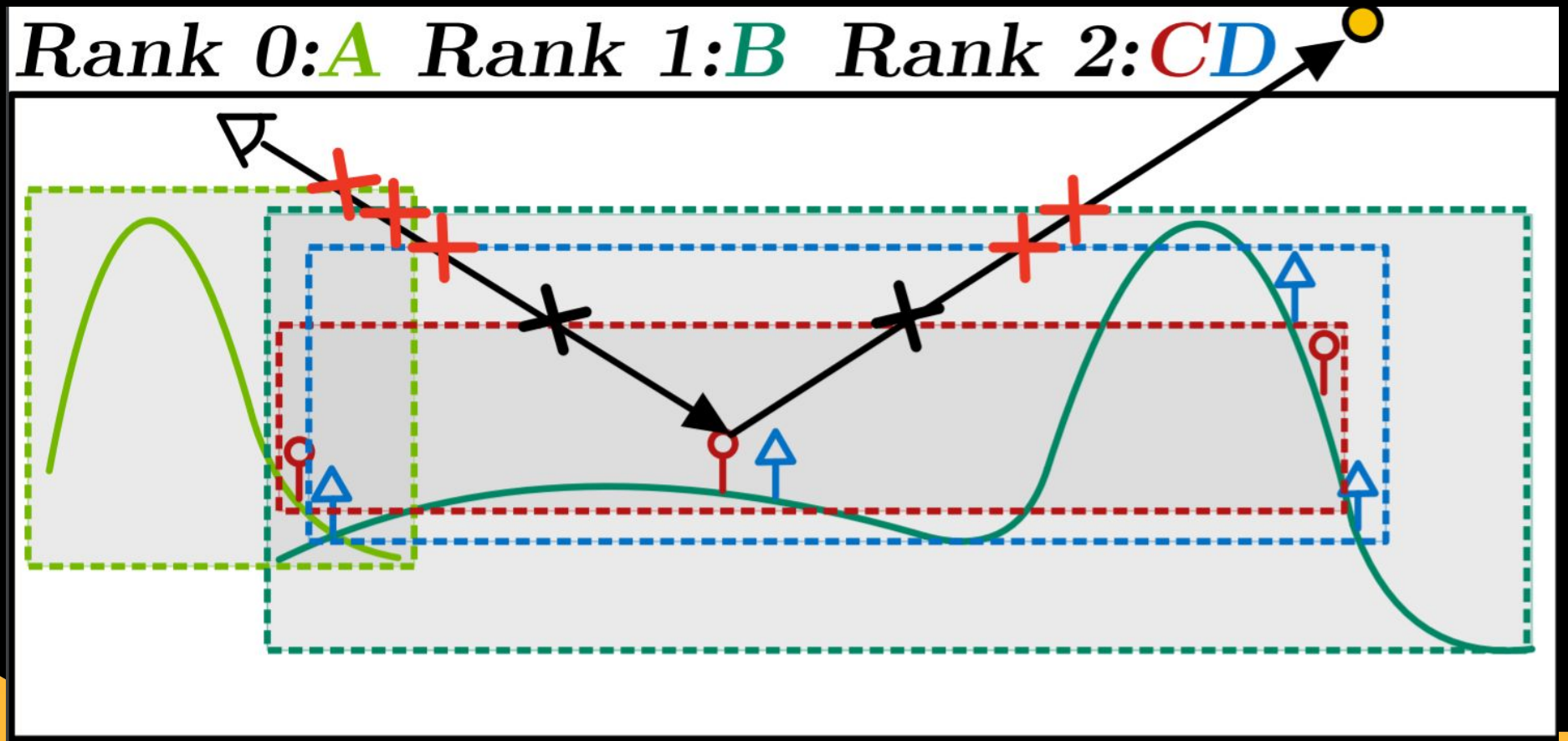
  - Also “re-braiding”: similar approach for instances

- Can do exactly the same on partitioning problem

Applying spatial splits & re-braiding eqiv.

- Let's simply put A to rank 0, B to r
  others to rank 2 …

"pure" object space w/ large instance bounding boxes



Proxies

- Naïve: one proxy / inst, each inst on one rank
  - But: could also have more than one inst in a proxy…
  - … or more than one proxy per any geom…
  - … or even replicate some insts to more than one rank