Efficient BVH Construction via Approximate Agglomerative Clustering

Yan Gu, Yong He,
Kayvon Fatahalian, Guy Blelloch
Carnegie Mellon University

BVH CONSTRUCTION GOALS

- High quality: produce BVHs of comparable (or better) quality to full-sweep SAH algorithms.

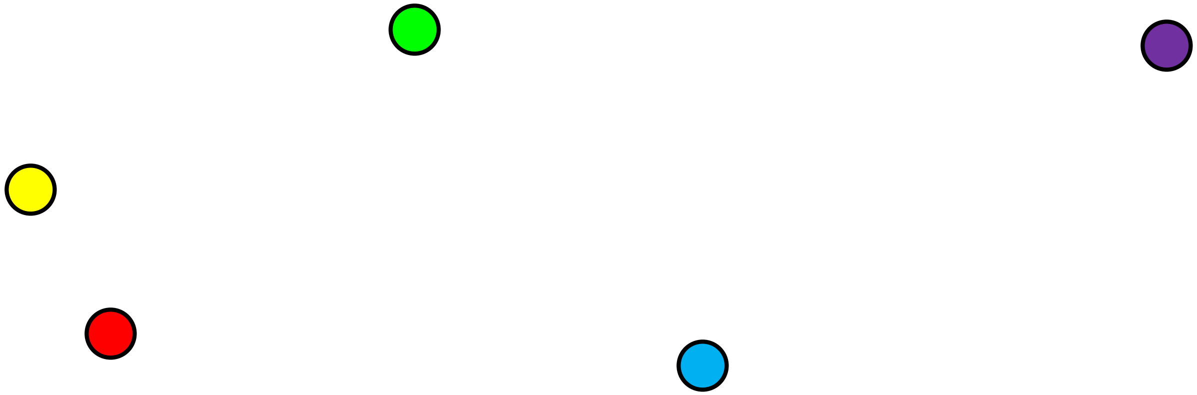
- High performance: faster construction than widely used SAH-based algorithms that use binning.

OUR APPROACH

- An agglomerative clustering (bottom-up) based construction algorithm.
  - Motivated by [Walter et al. 2008] .

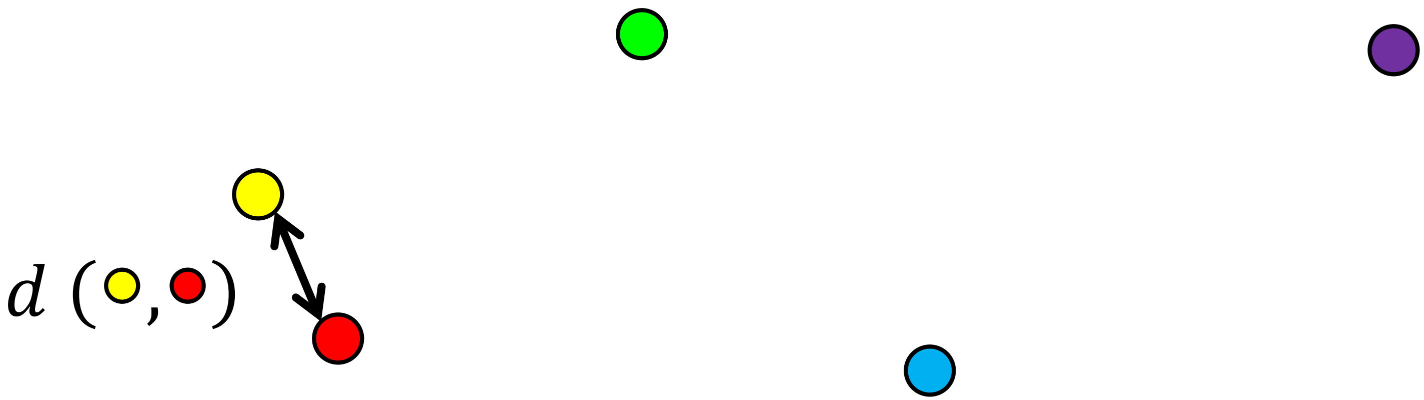
Hierarchical clustering example

Source data points:



HIERARCHICAL CLUSTERING EXAMPLE

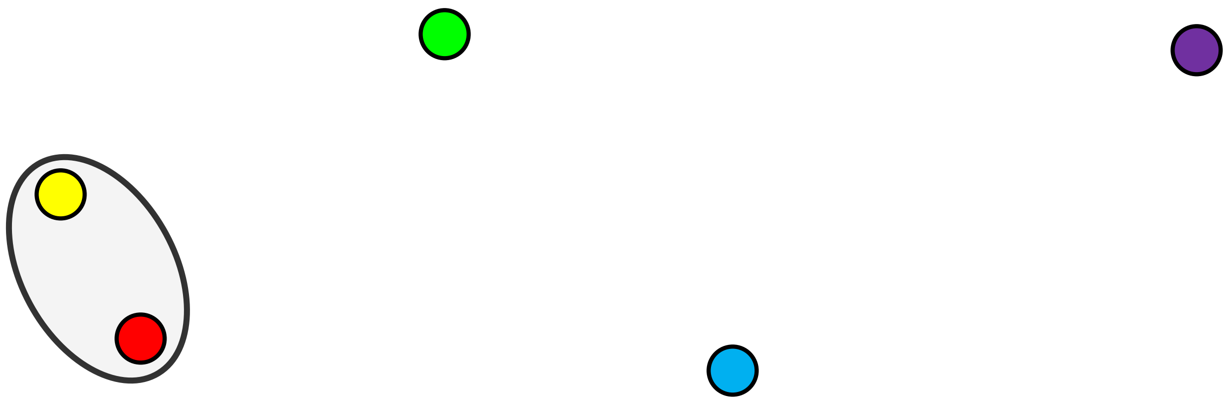
Source data points:



$d(i, j)$ = distance from cluster i to cluster j

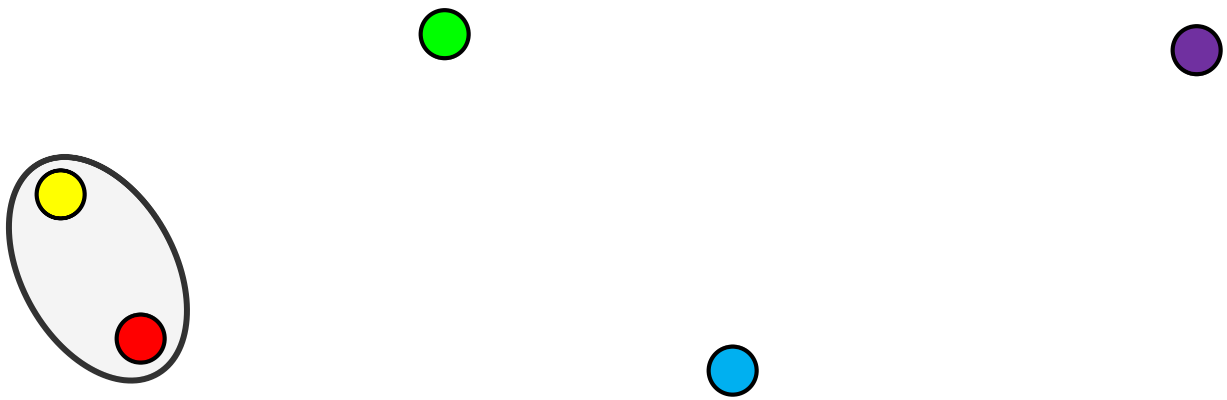
Hierarchical clustering example

Source data points:

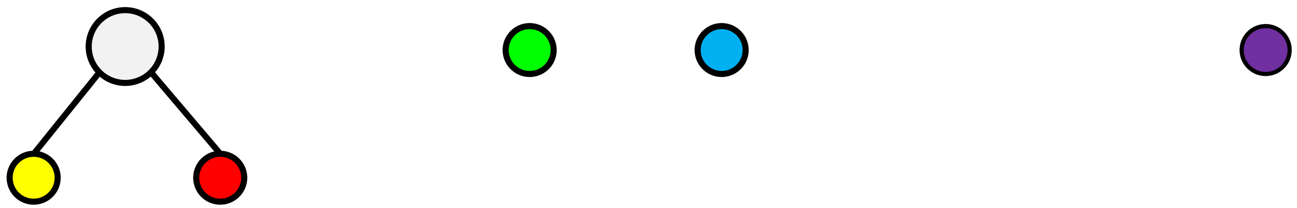


Hierarchical clustering example

Source data points:

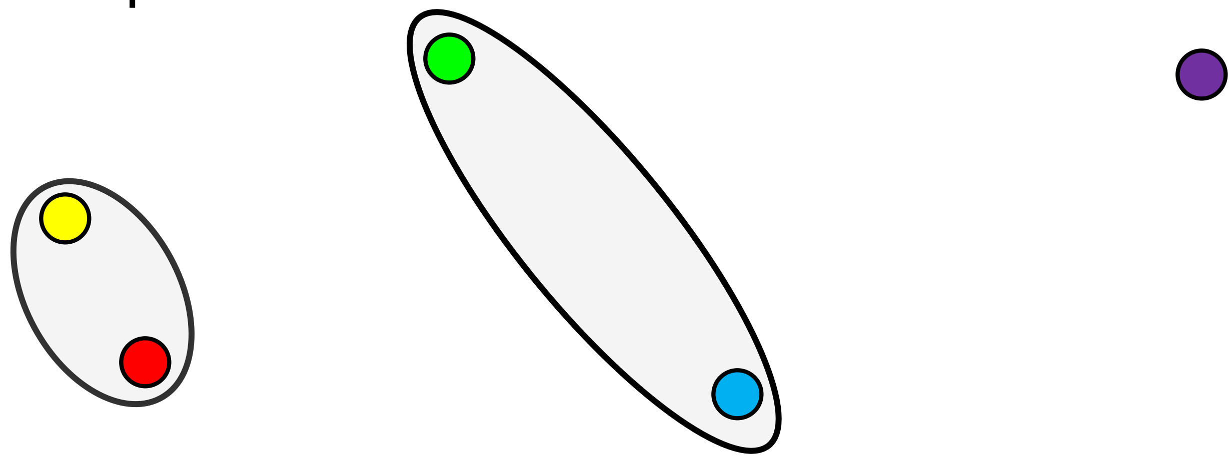


Resulting cluster hierarchy:

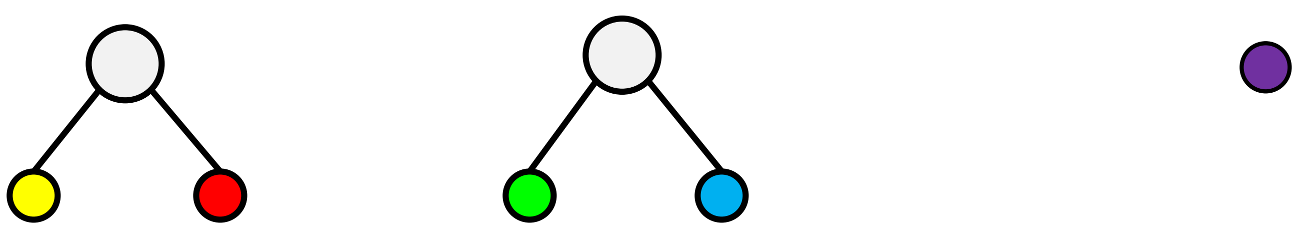


Hierarchical clustering example

Source data points:

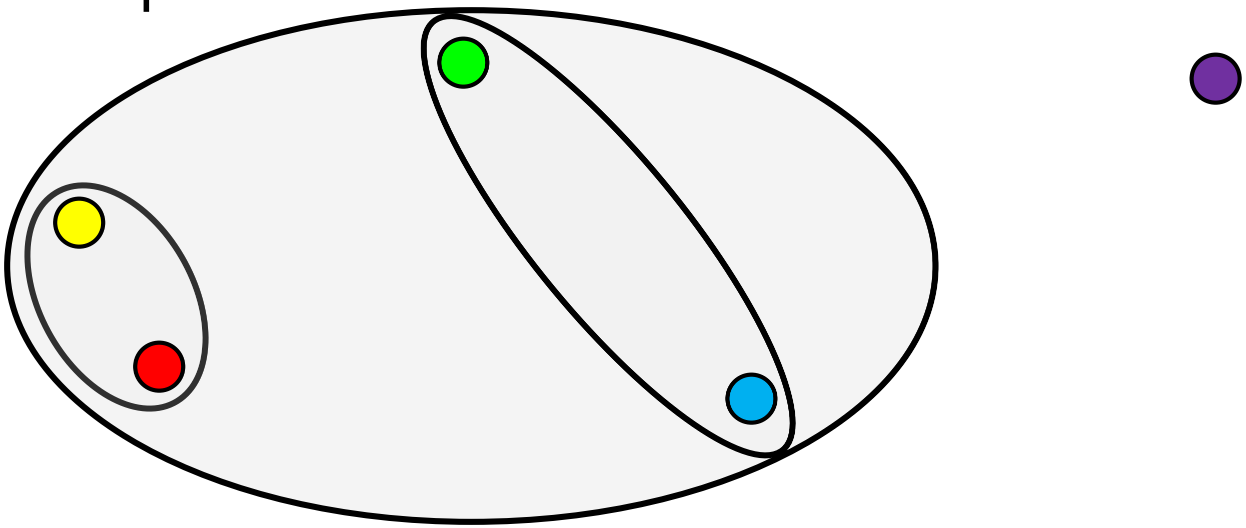


Resulting cluster hierarchy:

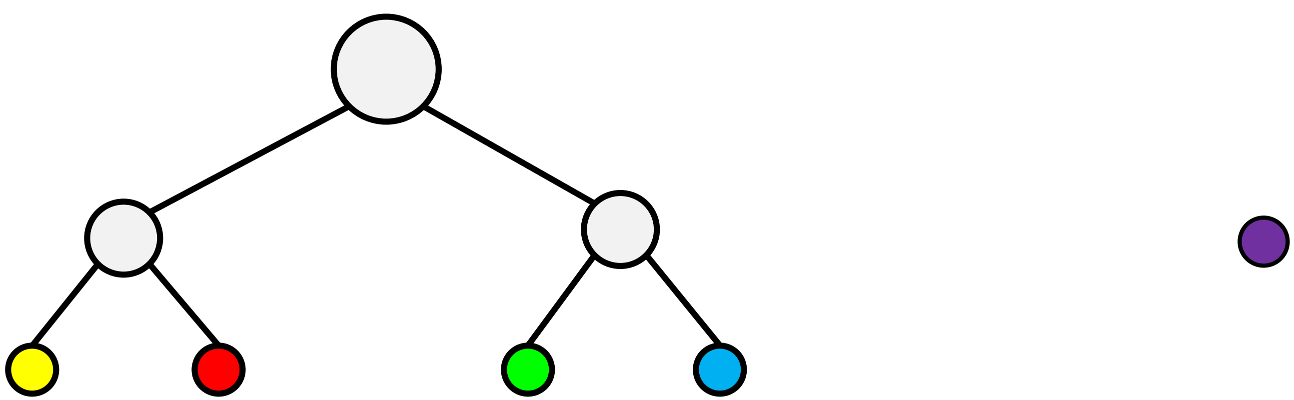


Hierarchical clustering example

Source data points:

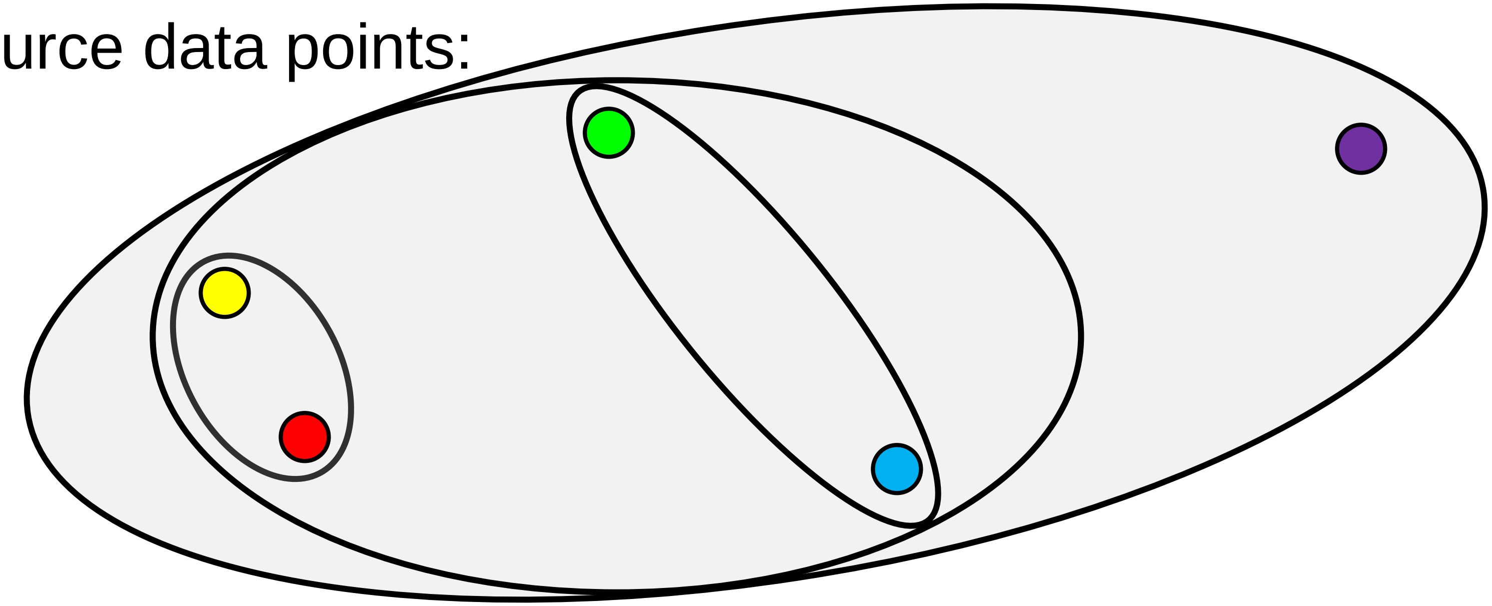


Resulting cluster hierarchy:

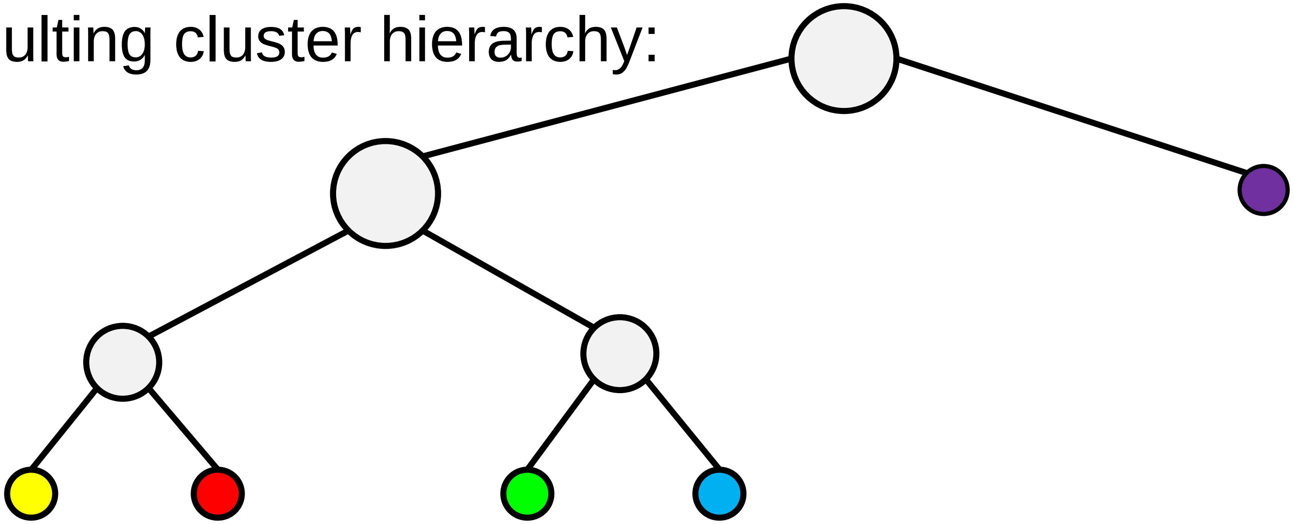


Hierarchical clustering example

Source data points:



Resulting cluster hierarchy:



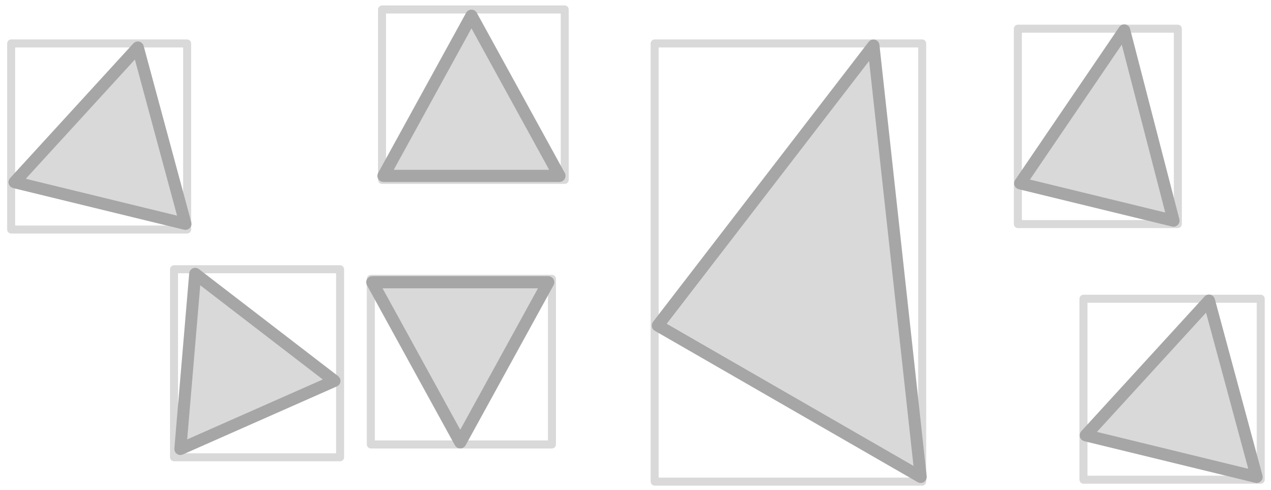
Hierarchical clustering is a general technique for organizing data.

Domain	Clustered primitives
Linguistic	Languages
Image retrieval	Images
Anthropology	Surnames / races
Biology	Genes / species
Social network	People / behaviors

BVH build using agglomerative clustering [Walter et al. 2008]

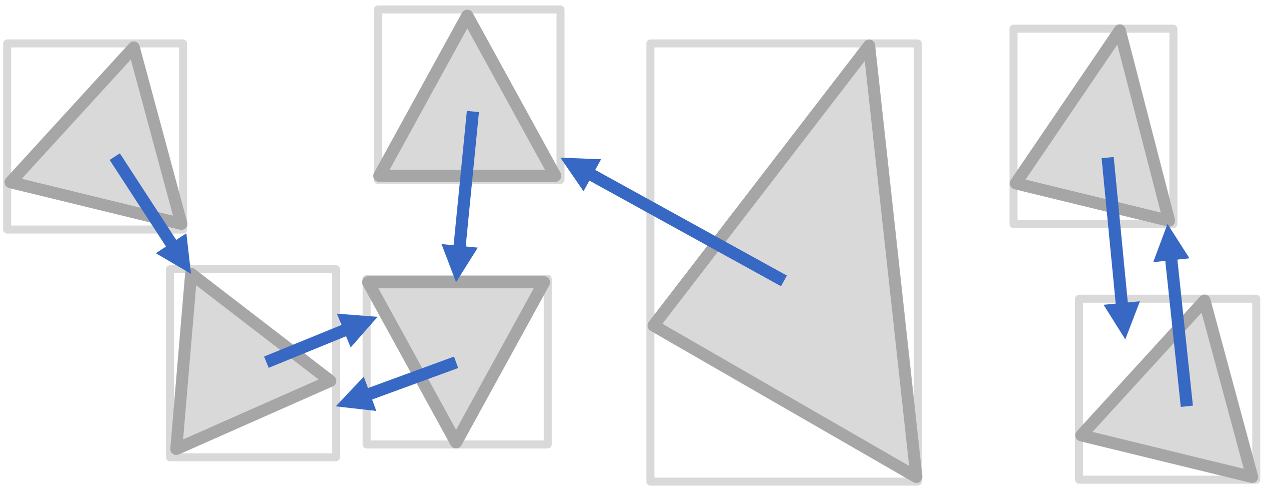
- Elements to cluster = scene primitives

- Distance = surface area of aggregate bounding box



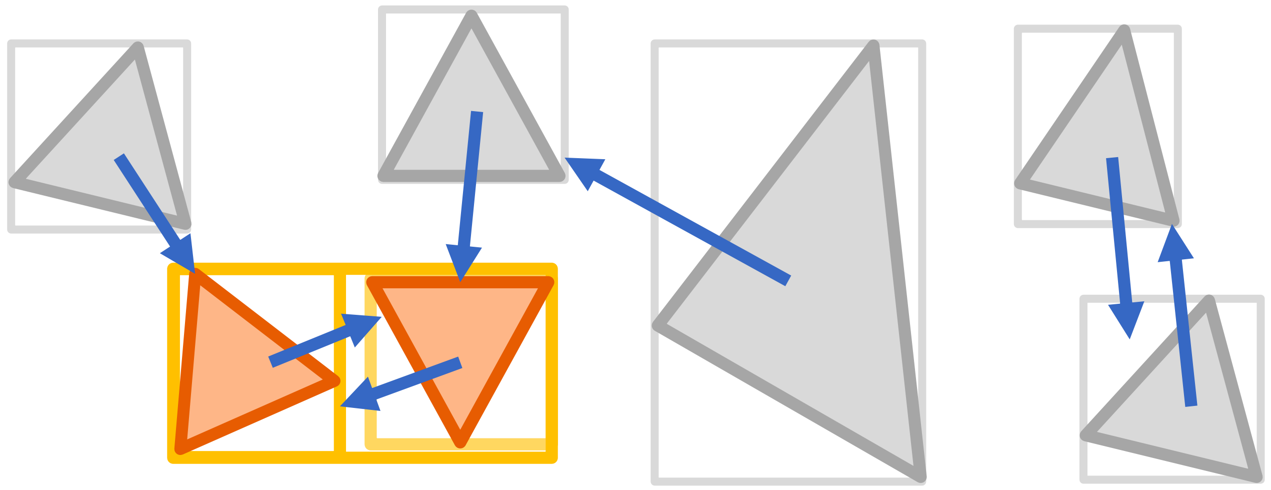
BVH build using agglomerative clustering [Walter et al. 2008]

- Compute the nearest neighbor to each primitive.



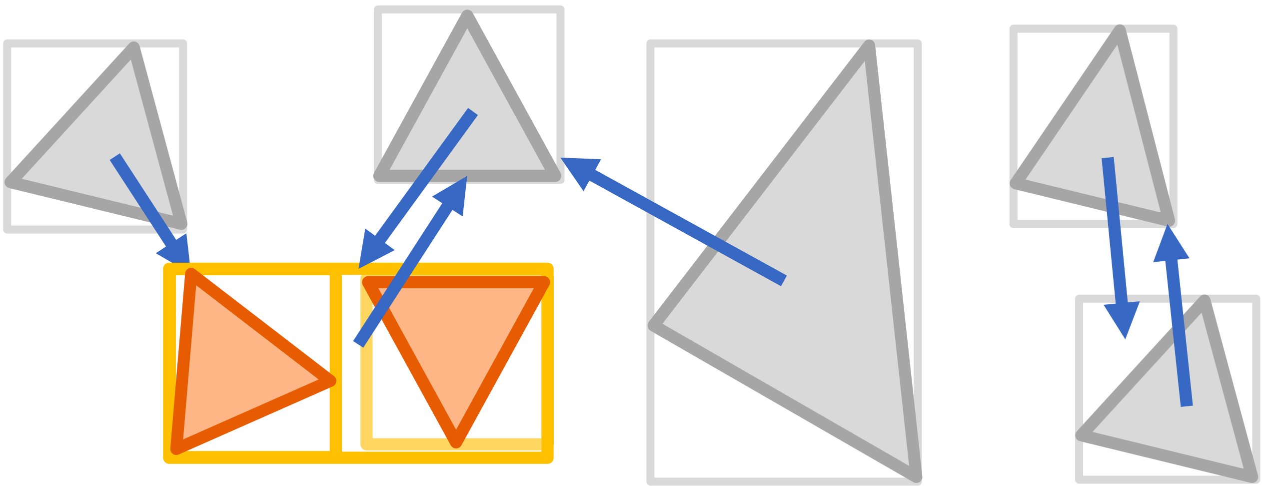
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Find the "closest" pair of primitives and combine them into a cluster.



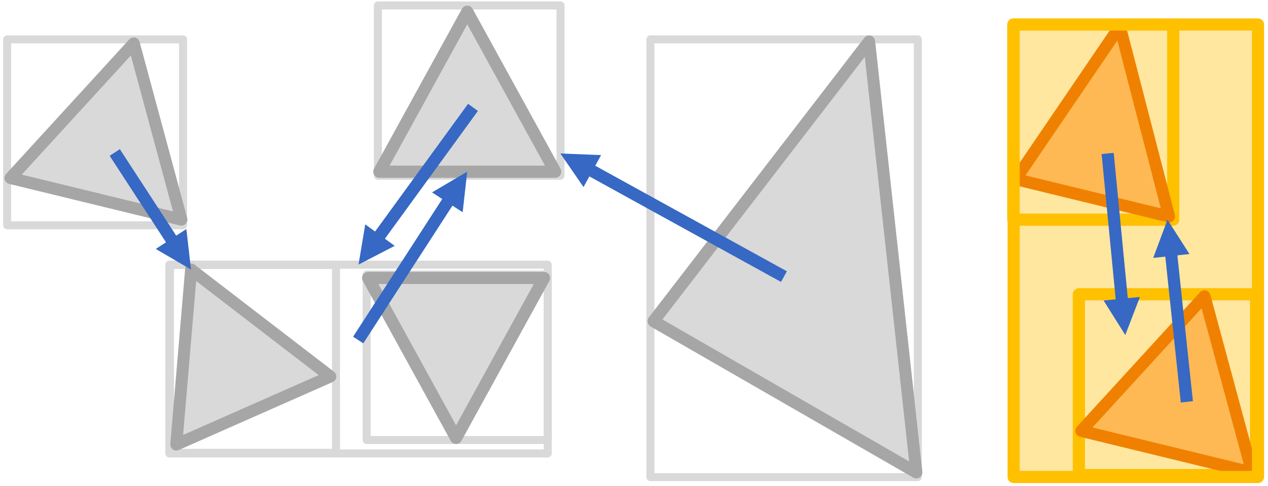
BVH build using agglomerative clustering [Walter et al. 2008]

- Update nearest neighbor links.



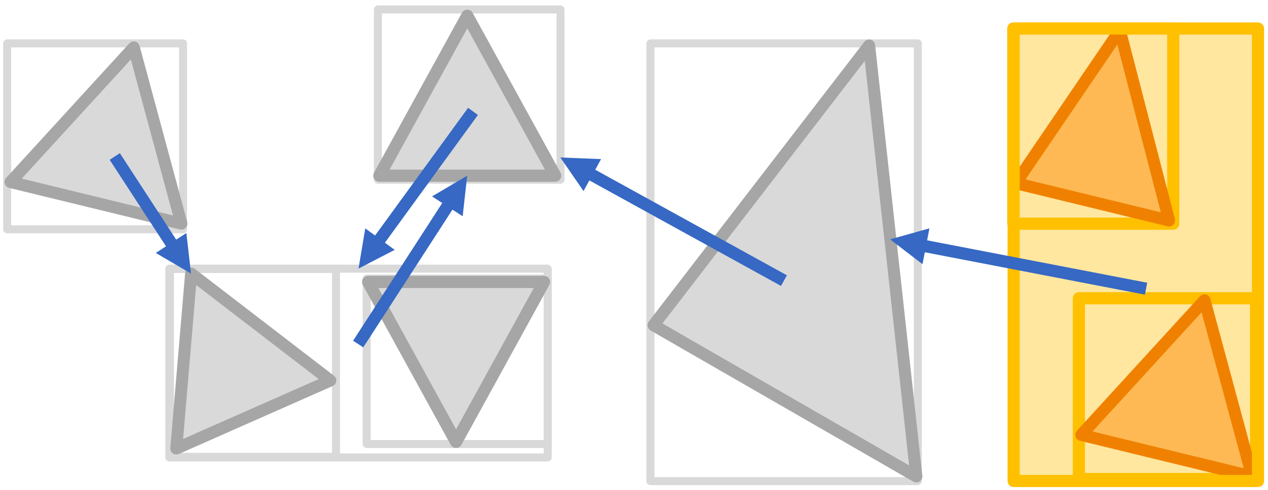
BVH build using agglomerative clustering [Walter et al. 2008]

- Repeat: combine closest remaining clusters.



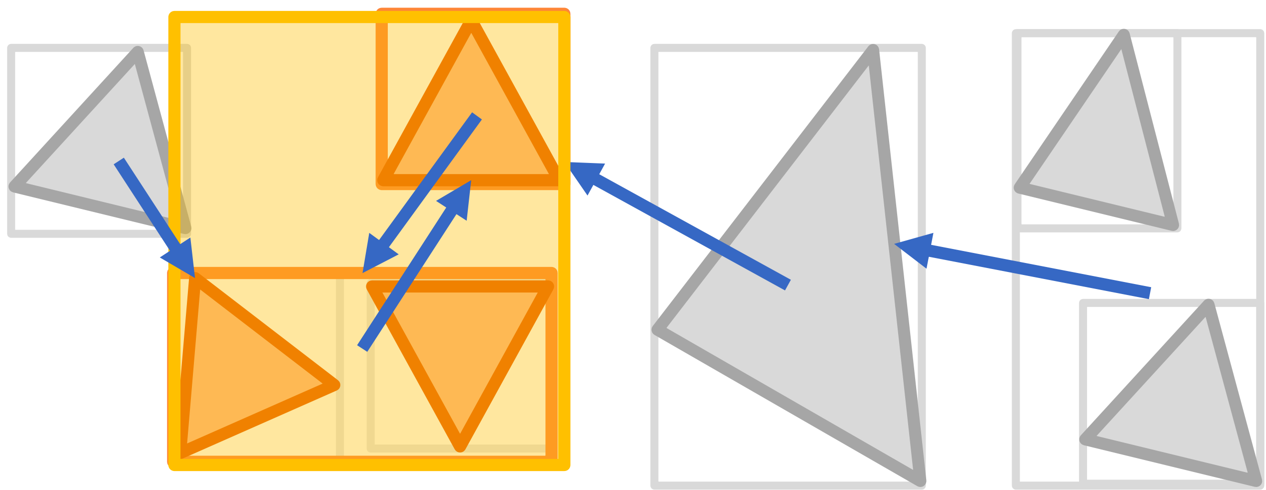
BVH build using agglomerative clustering [Walter et al. 2008]

- Repeat: combine closest remaining clusters.



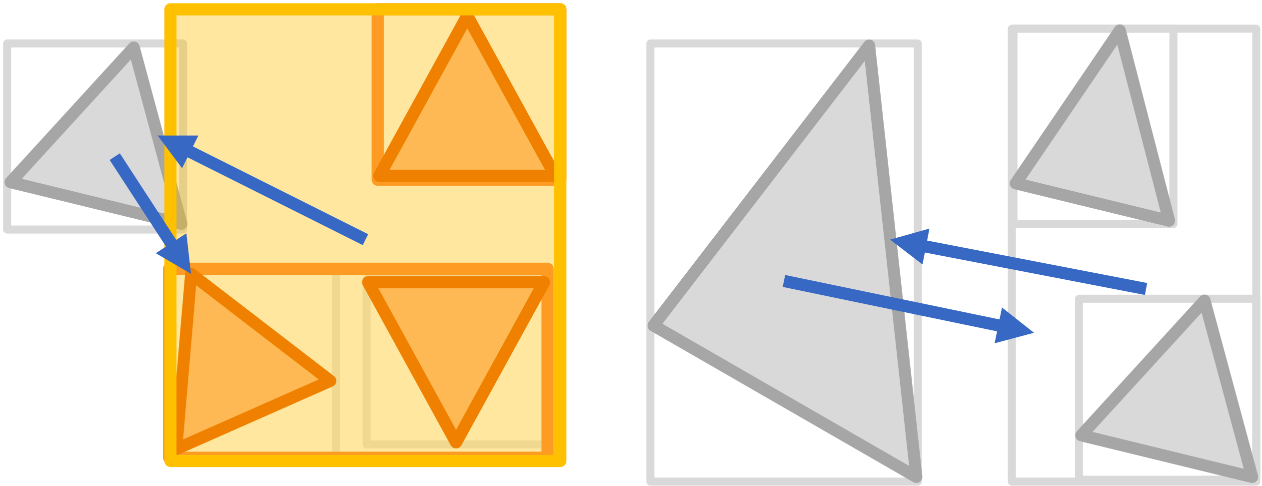
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.



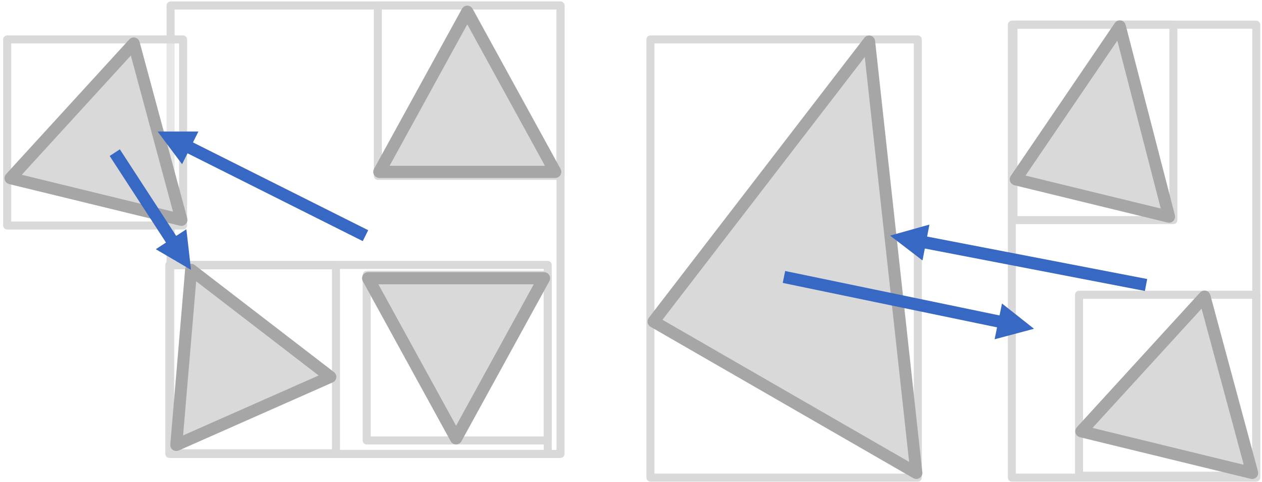
BVH build using agglomerative clustering [Walter et al. 2008]

- Repeat: combine closest remaining clusters.



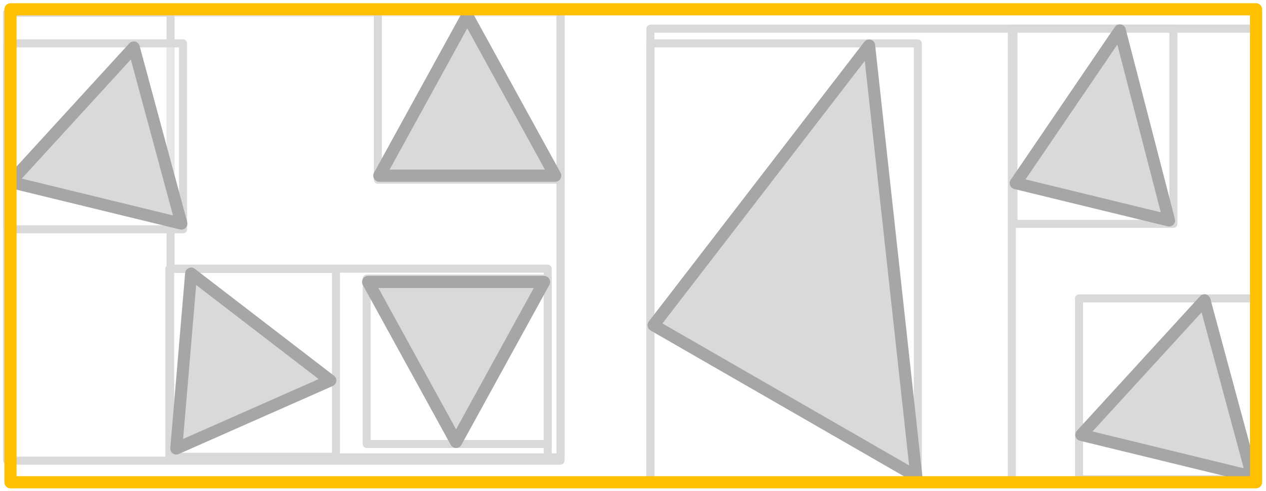
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.



BVH build using agglomerative clustering [Walter et al. 2008]

- Continue until one cluster remains (BVH root).



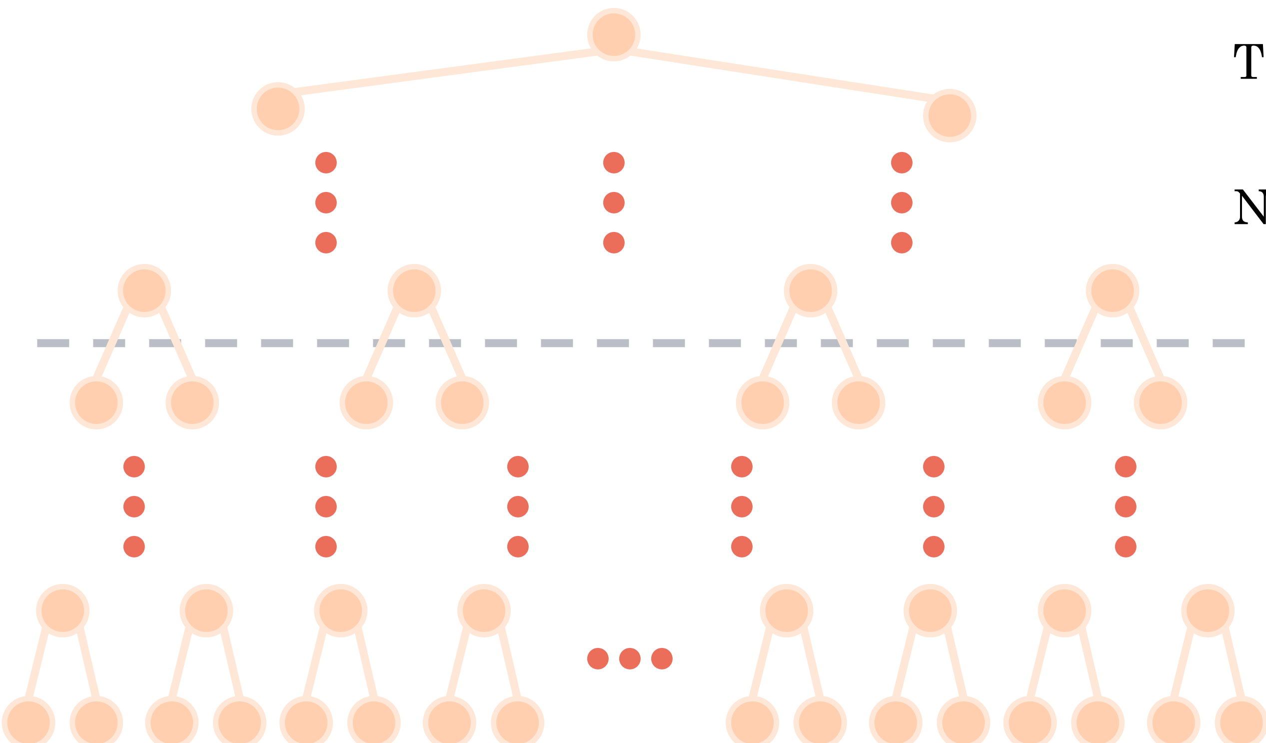
BVH build using agglomerative clustering [Walter et al. 2008]

- Good: often higher quality BVH than sweep builds

- Bad: lower performance than binned builds
  - KD-tree search/update in each clustering step.
  - Data-dependent parallel execution.

Observation

- Most computation occurs at the lowest levels of the BVH of the construction process when the number of clusters is large (near leaves) .



Top $\frac{h}{2}$ levels:

Number of nodes $\approx \sqrt{n}$

Bottom $\frac{h}{2}$ levels:

Number of nodes $\approx n$

CONTRIBUTION

Approximate Agglomerative Clustering (AAC)

- New algorithm for BVH construction that is work efficient, parallelizable, and produces high-quality trees.

OUR MAIN IDEA

- Restrict nearest neighbor search to a small subset of neighboring scene elements.

Our main idea

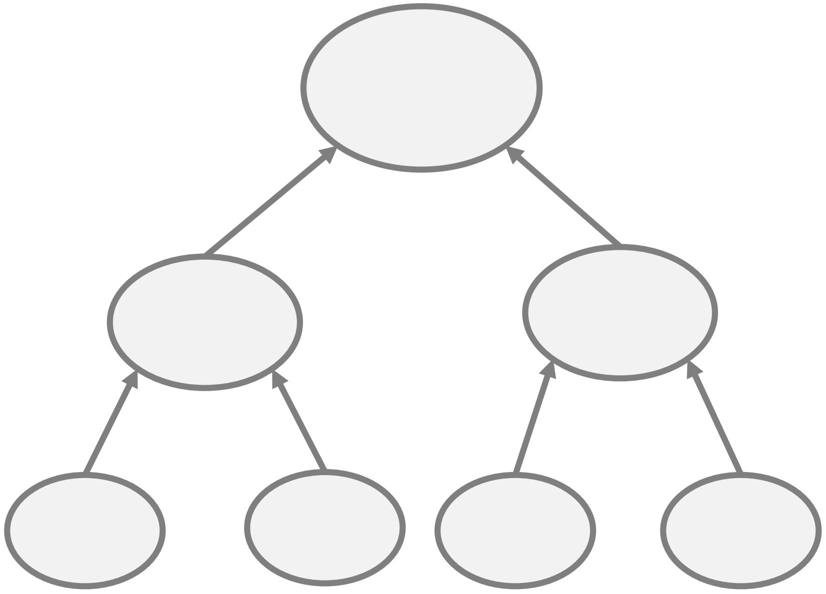
- Restrict nearest neighbor search to a small subset of neighboring scene elements.

OUR MAIN IDEA

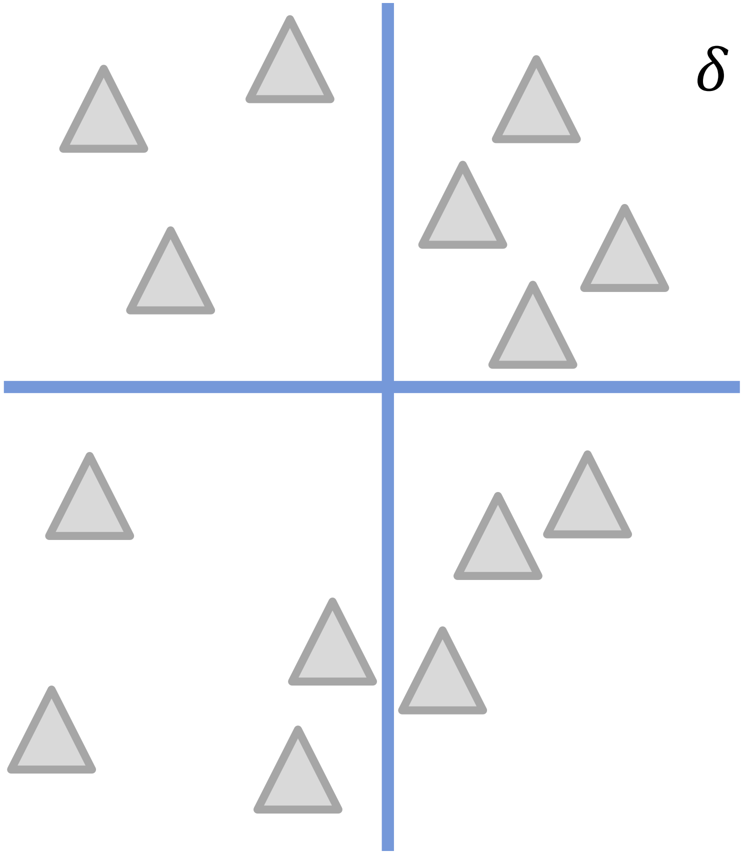
- Restrict nearest neighbor search to a small subset of neighboring scene elements.

PHASE 1: PRIMITIVE PARTITIONING ("DOWNWARD/DIVIDE PHASE")

Computation graph:



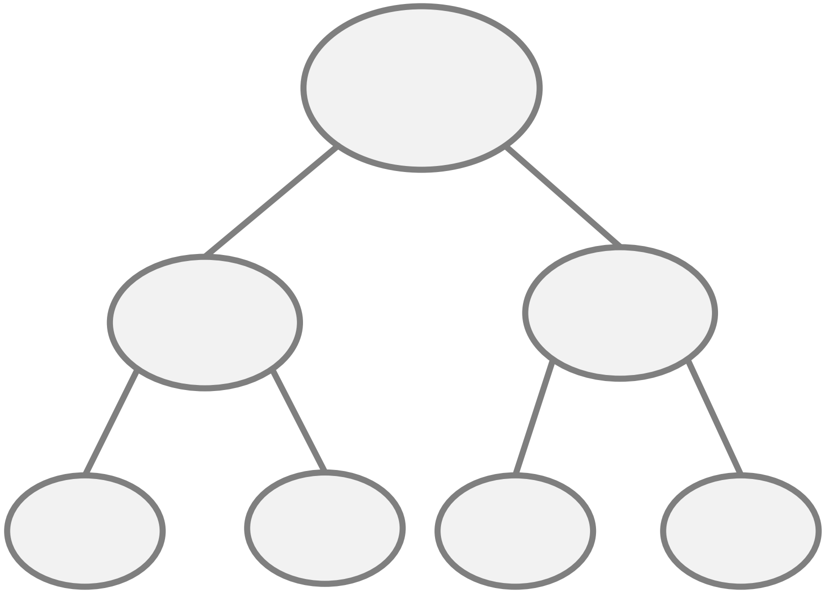
Primitive partitioning:



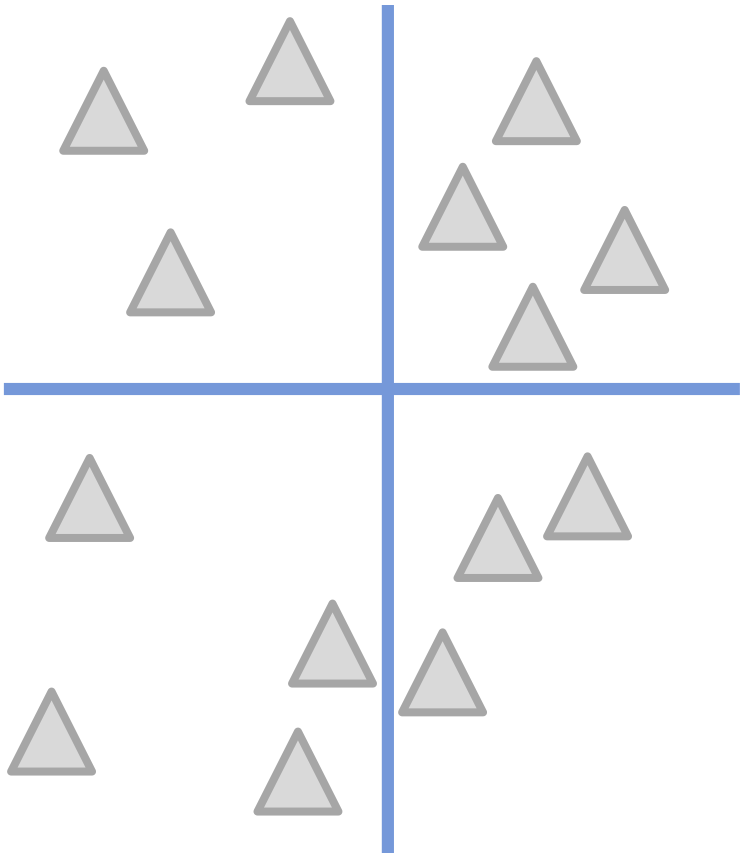
$\delta = 4$

PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:



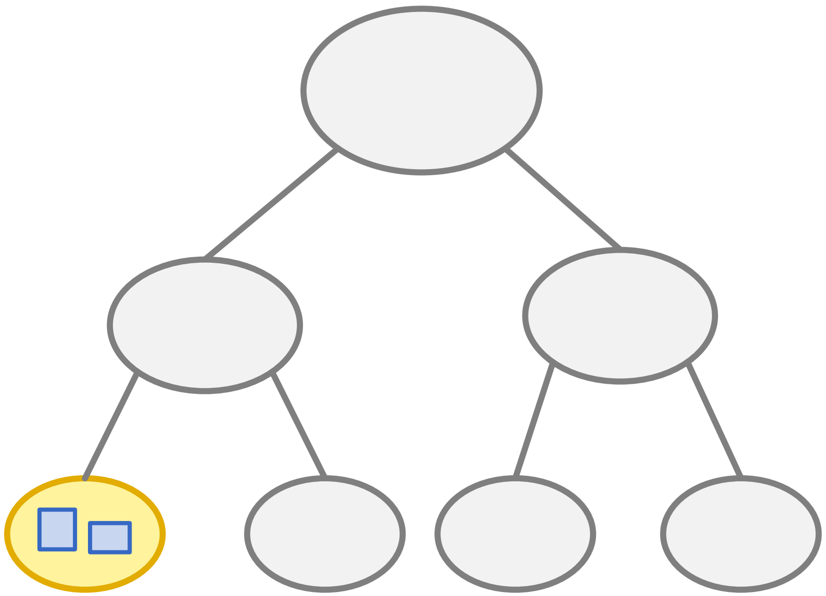
Primitive partitioning:



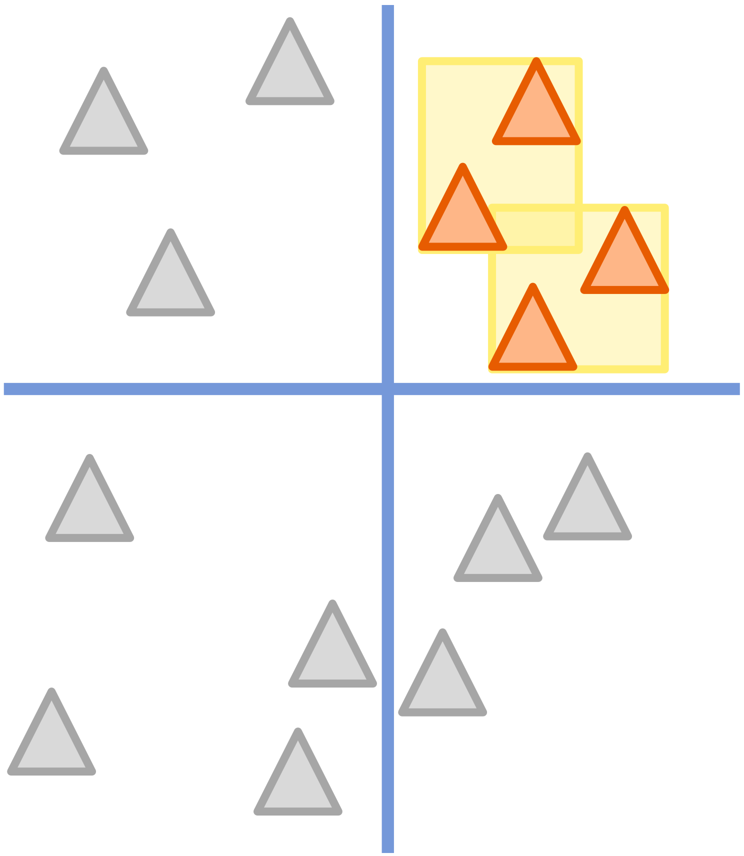
Phase 2: agglomerative clustering ("upward phase")

Computation graph:

Each node = combine input into $f(n)$ clusters



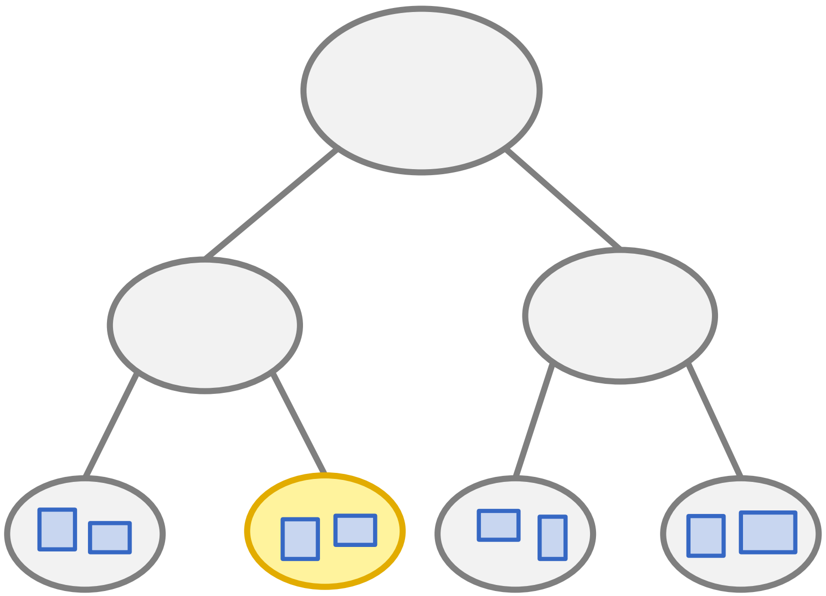
Primitive partitioning:



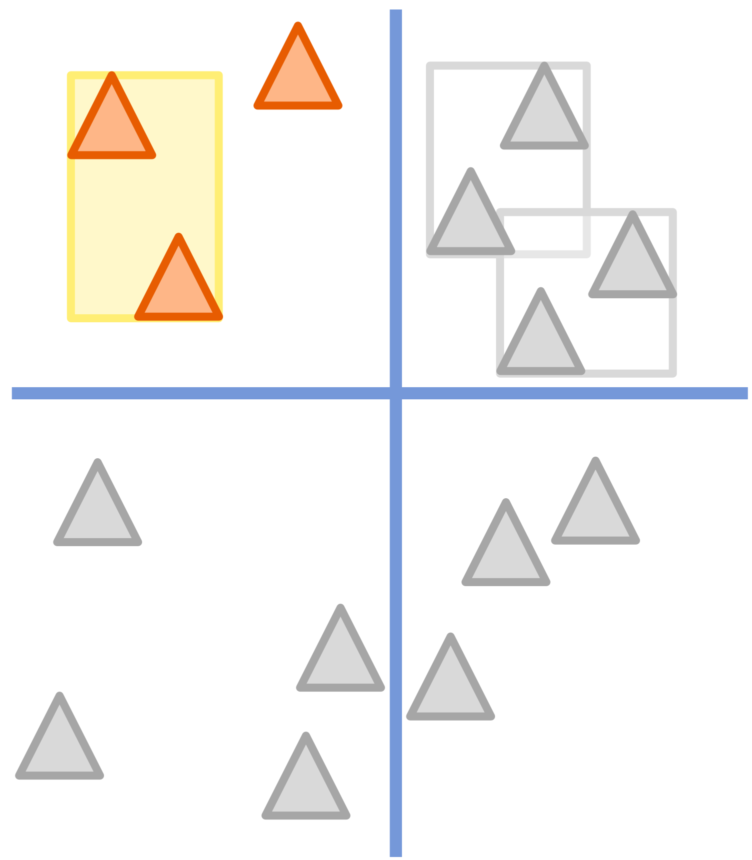
Phase 2: agglomerative clustering ("upward phase")

Computation graph:

Each node = combine input into $f(n)$ clusters



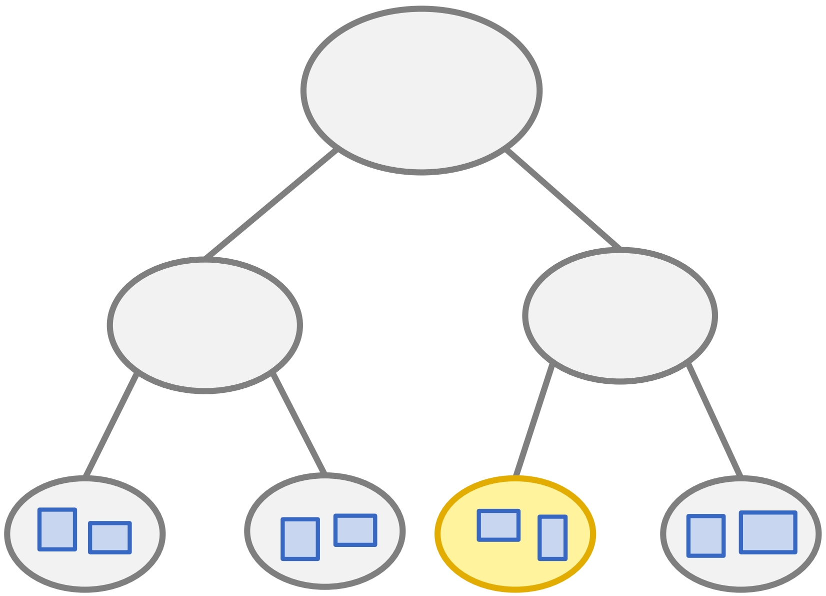
Primitive partitioning:



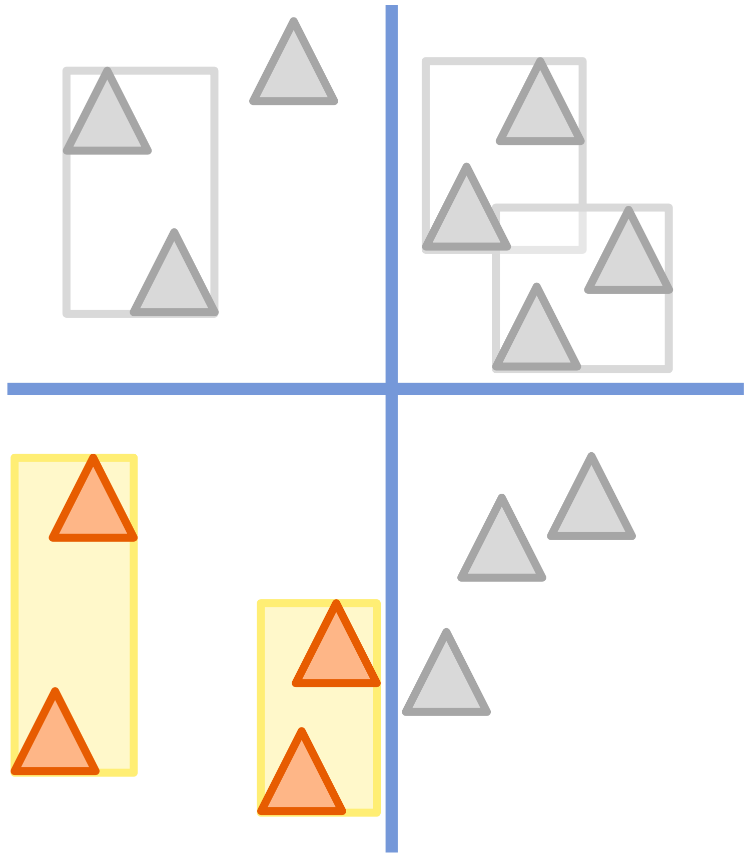
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

Each node = combine input into $f(n)$ clusters



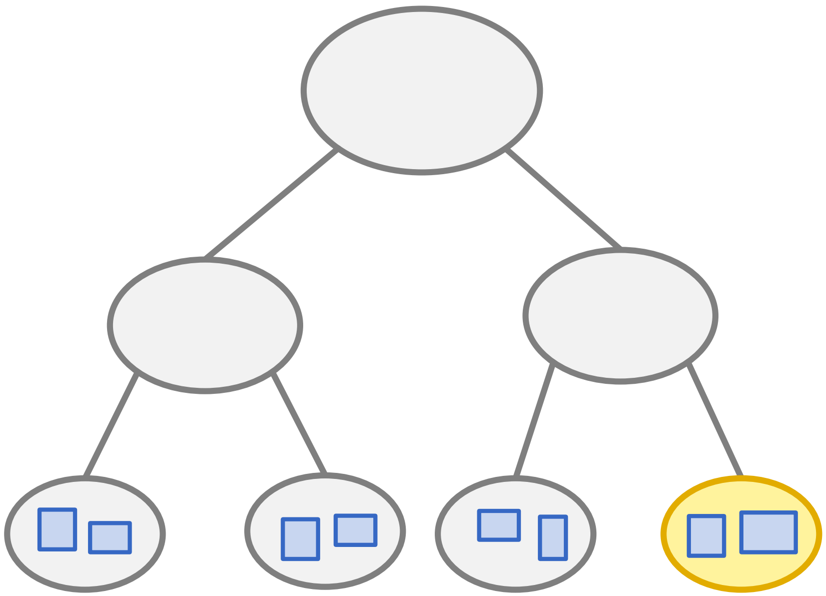
Primitive partitioning:



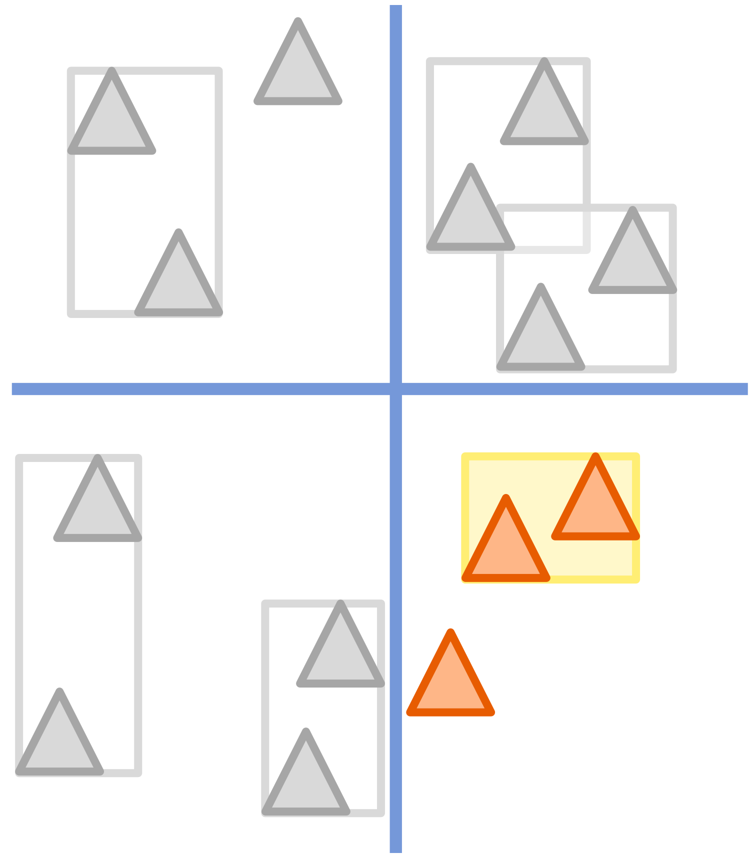
Phase 2: agglomerative clustering (“upward phase”)

Computation graph:

Each node = combine input into $f(n)$ clusters



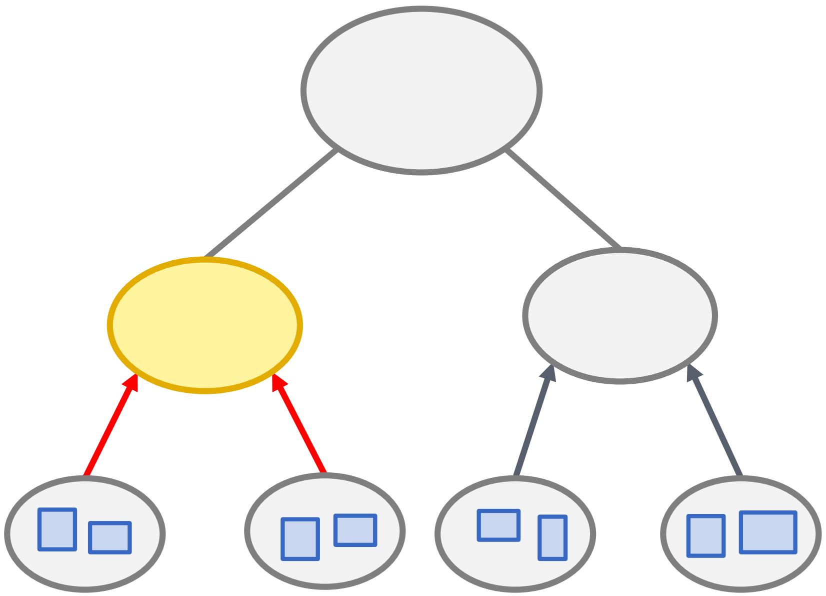
Primitive partitioning:



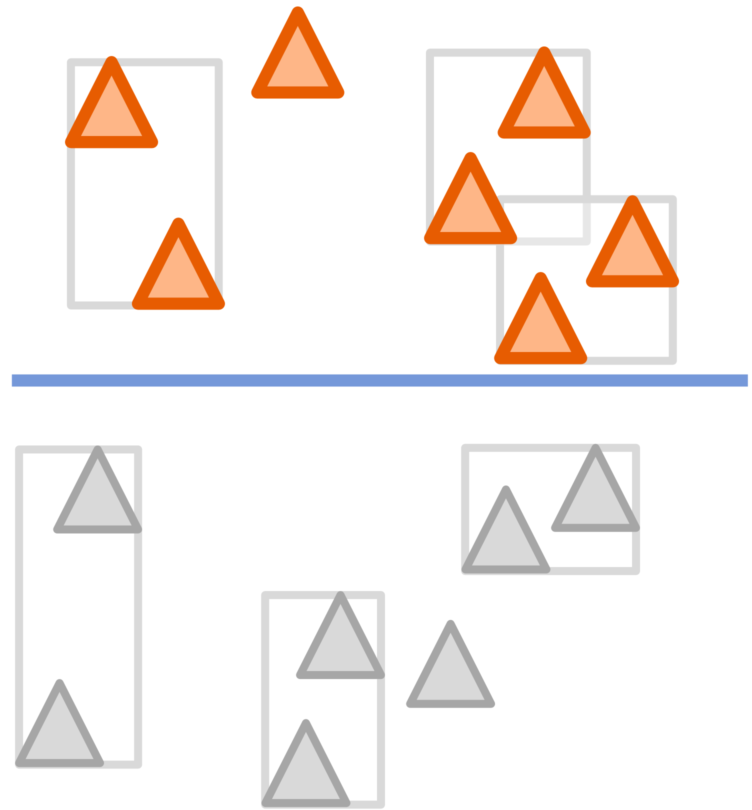
Phase 2: agglomerative clustering ("upward phase")

Computation graph:

Each node = combine input into $f(n)$ clusters



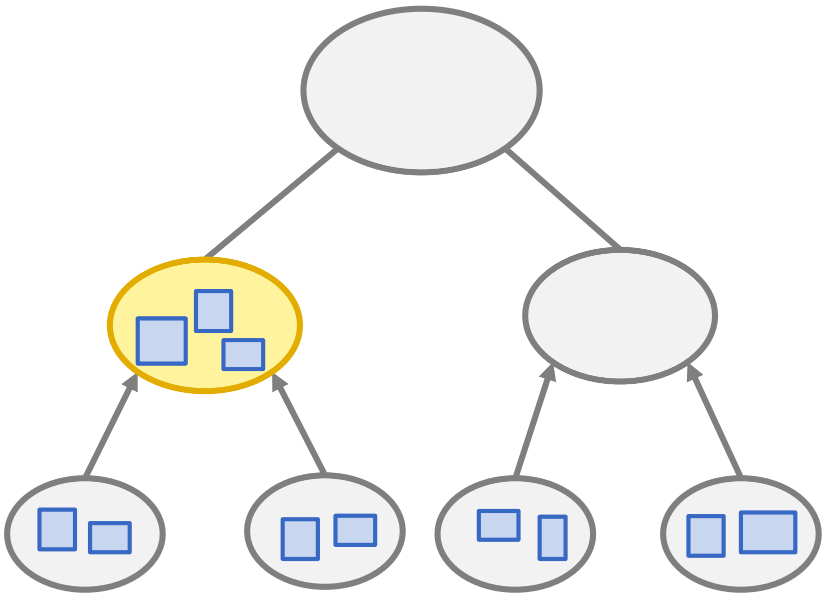
Primitive partitioning:



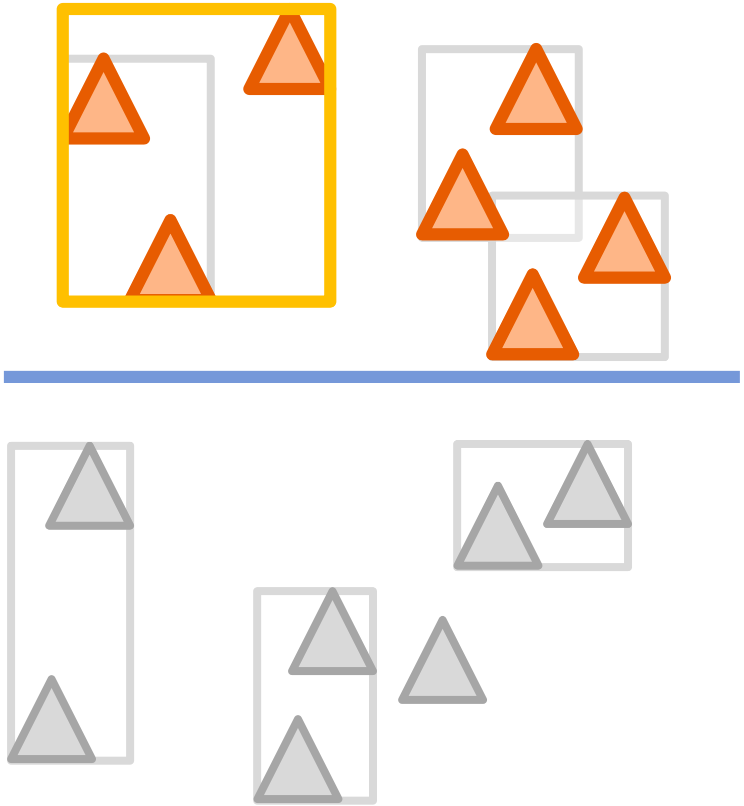
Phase 2: Agglomerative Clustering ("Upward Phase")

Computation graph:

Each node = combine input into $f(n)$ clusters



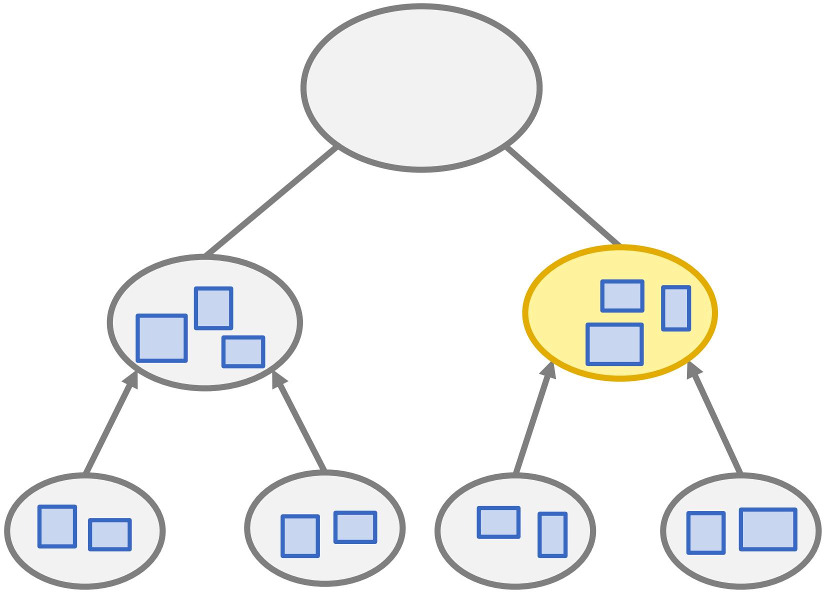
Primitive partitioning:



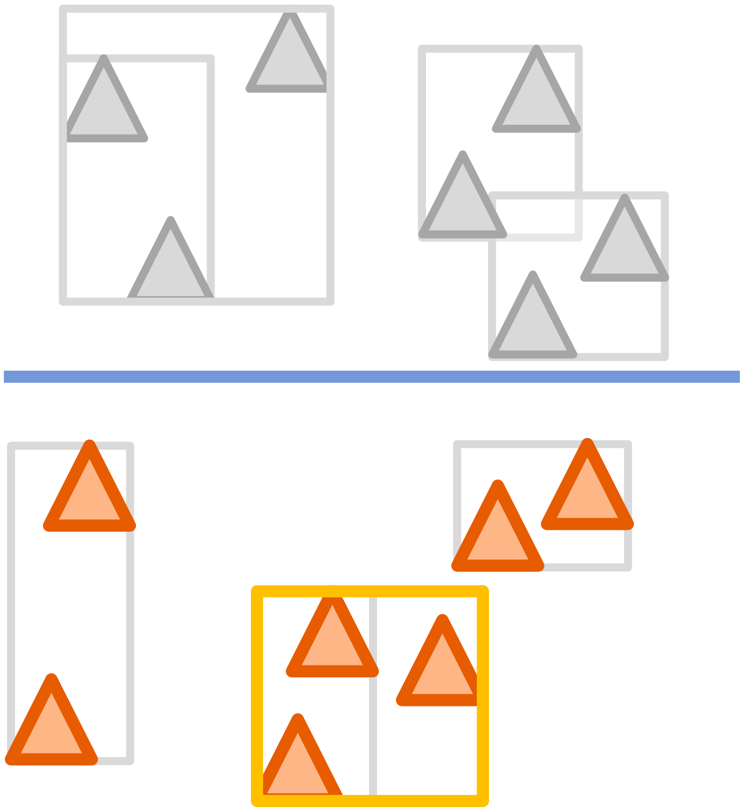
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

Each node = combine input into $f(n)$ clusters

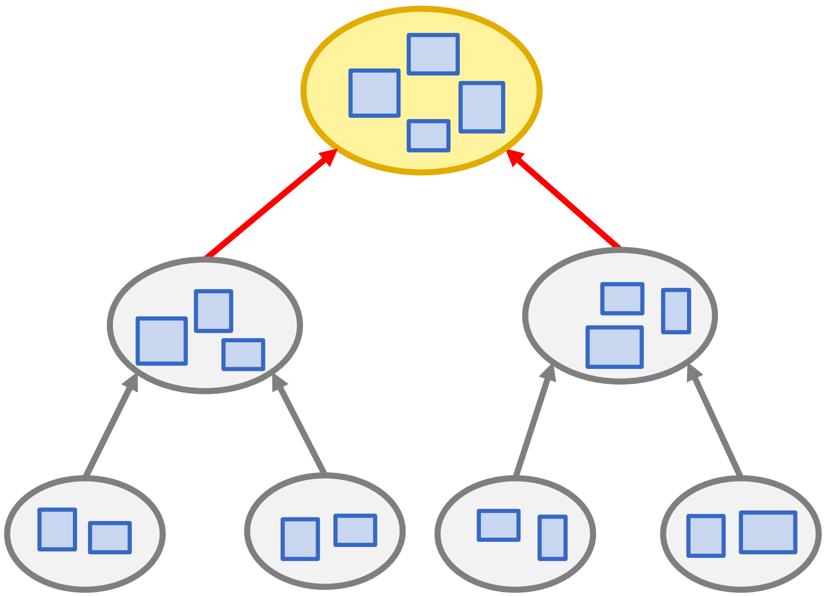


Primitive partitioning:

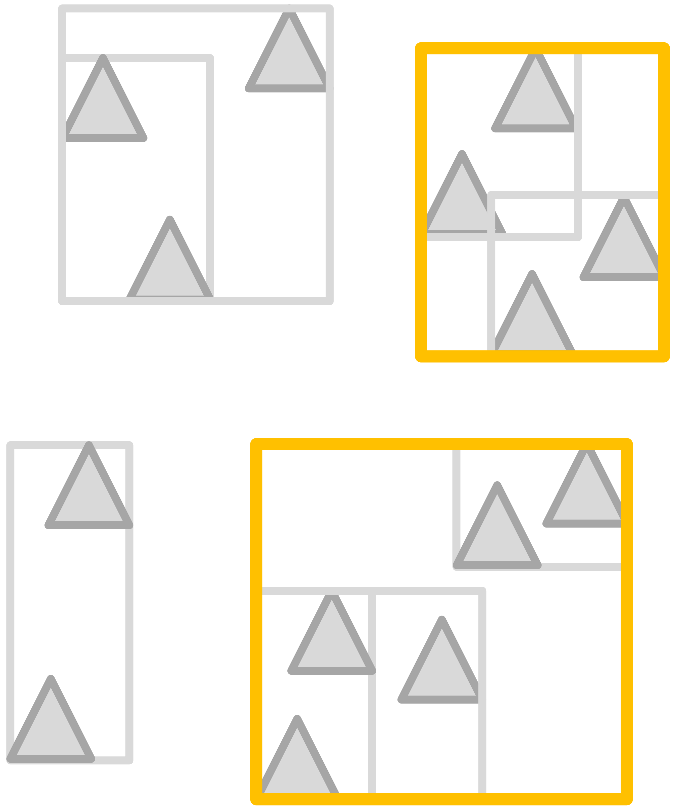


PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

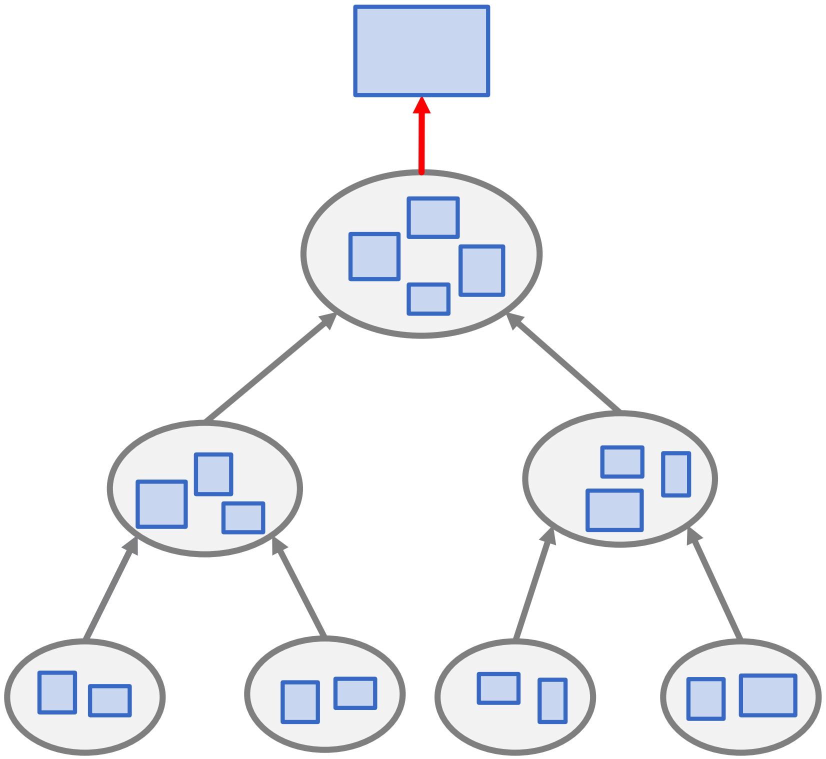


Primitive partitioning:

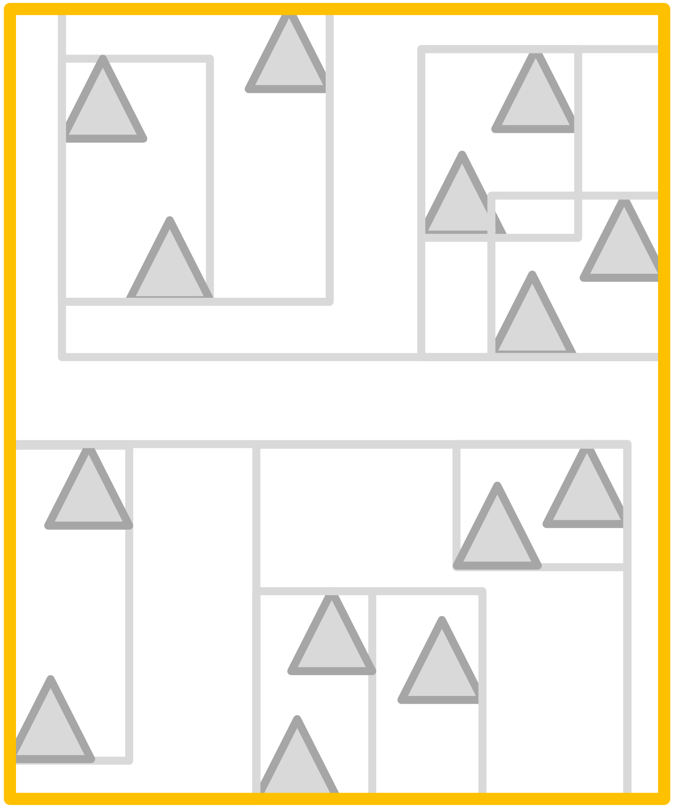


Phase 2: agglomerative clustering ("upward phase")

Computation graph:

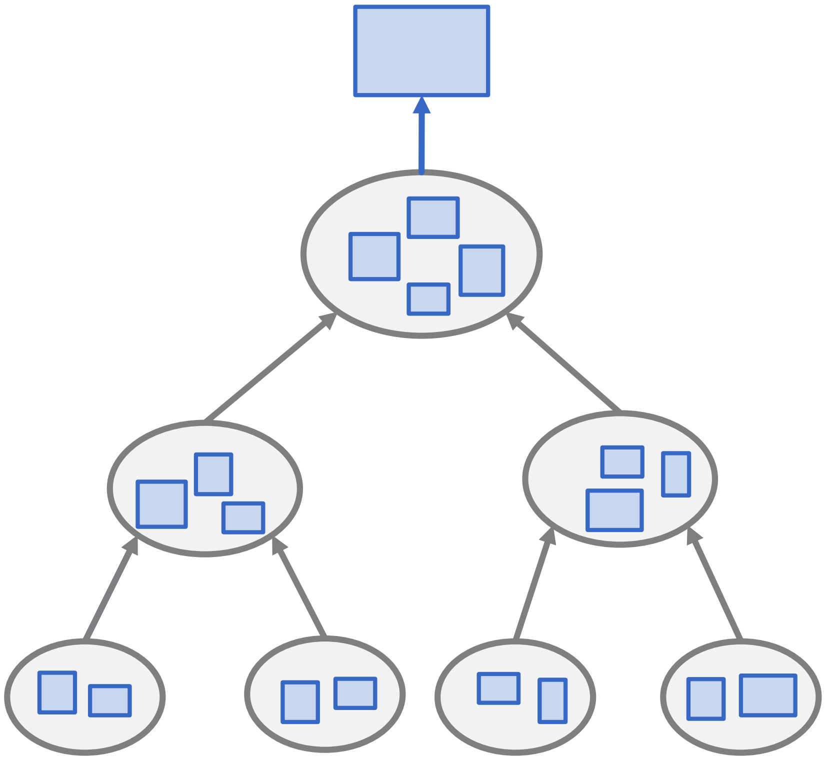


Primitive partitioning:

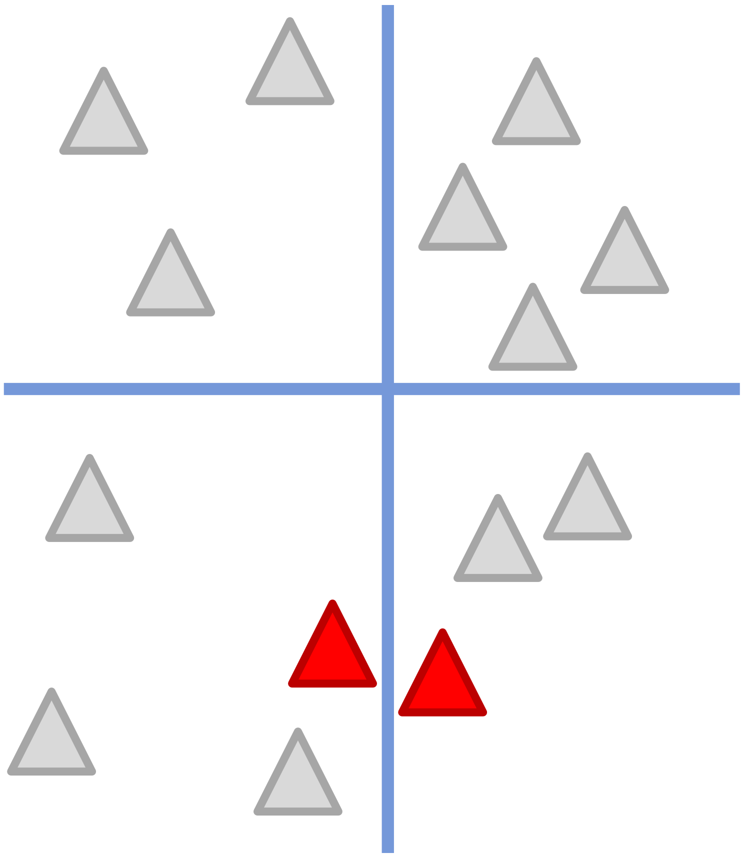


AAC IS AN APPROXIMATION TO THE TRUE AGGLOMERATIVE CLUSTERING SOLUTION.

Computation graph:

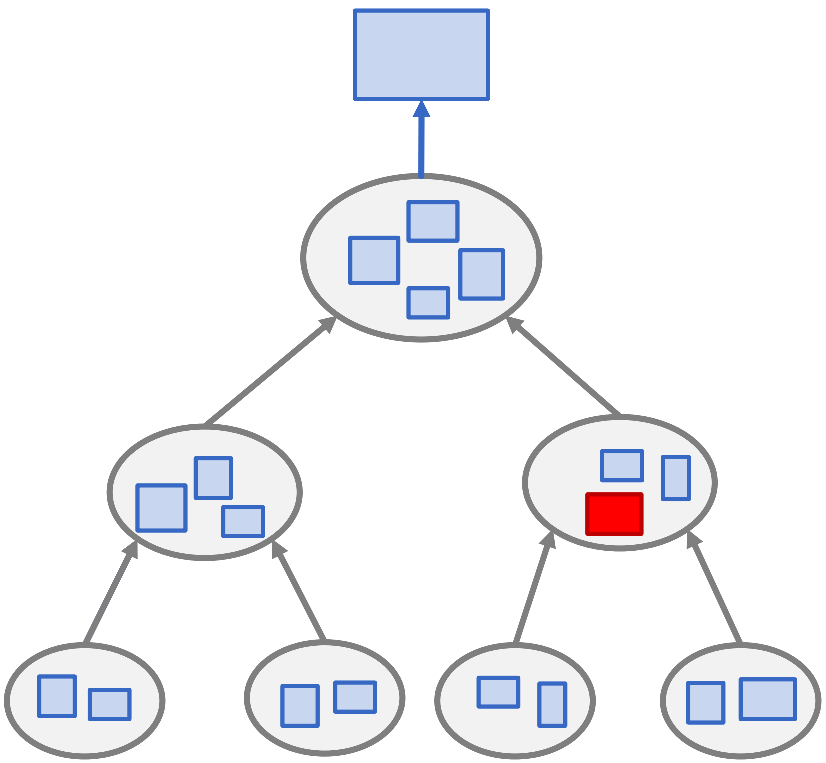


Primitive partitioning:

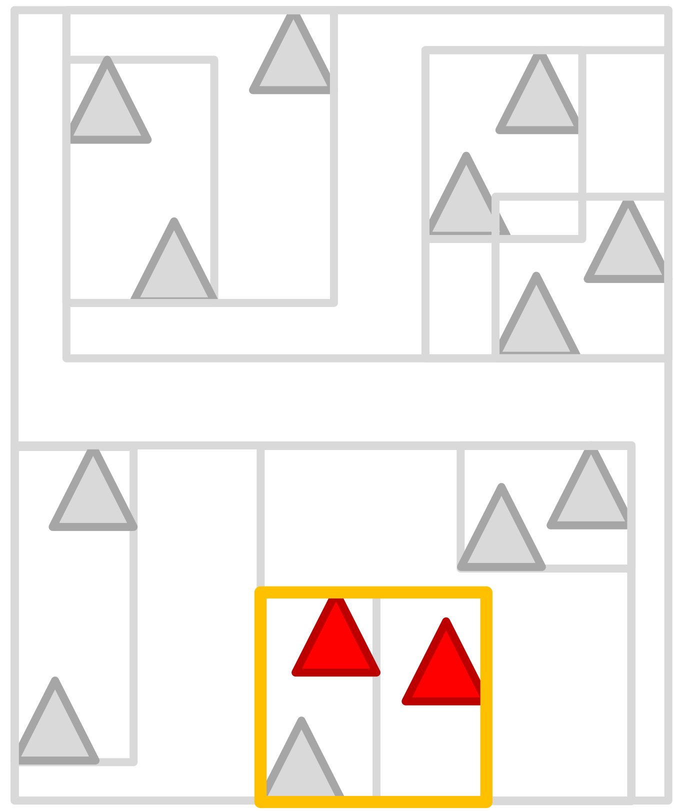


AAC IS AN APPROXIMATION TO THE TRUE AGGLOMERATIVE CLUSTERING SOLUTION.

Computation graph:



Primitive partitioning:

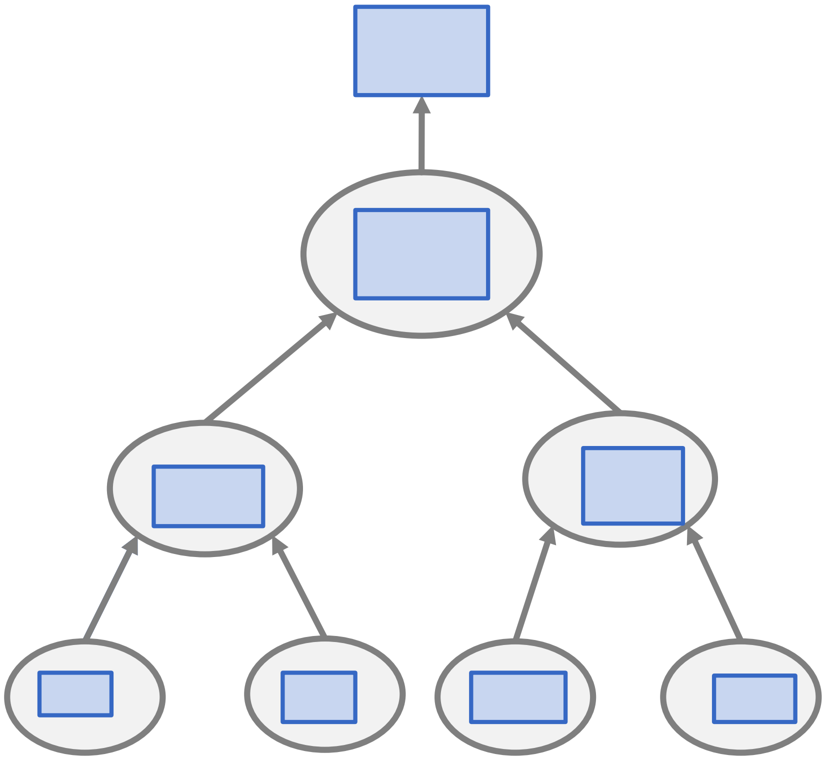


AAC HAS TWO PARAMETERS

- δ: stopping criterion for stop partitioning (maximum of primitives in leaf regions).
- $f(n)$: function that determines the number of clusters to generate in each graph node (n is the number of primitives in the corresponding region.)

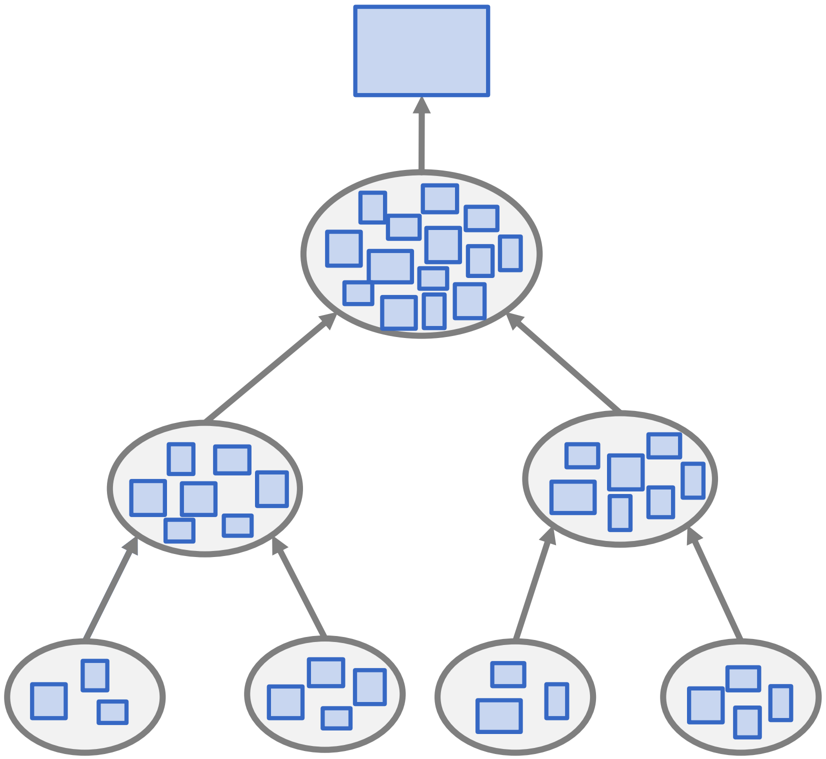
DETERMINING HOW MUCH TO CLUSTER

- $f(n) = 1$: close to spatial bisection BVH.



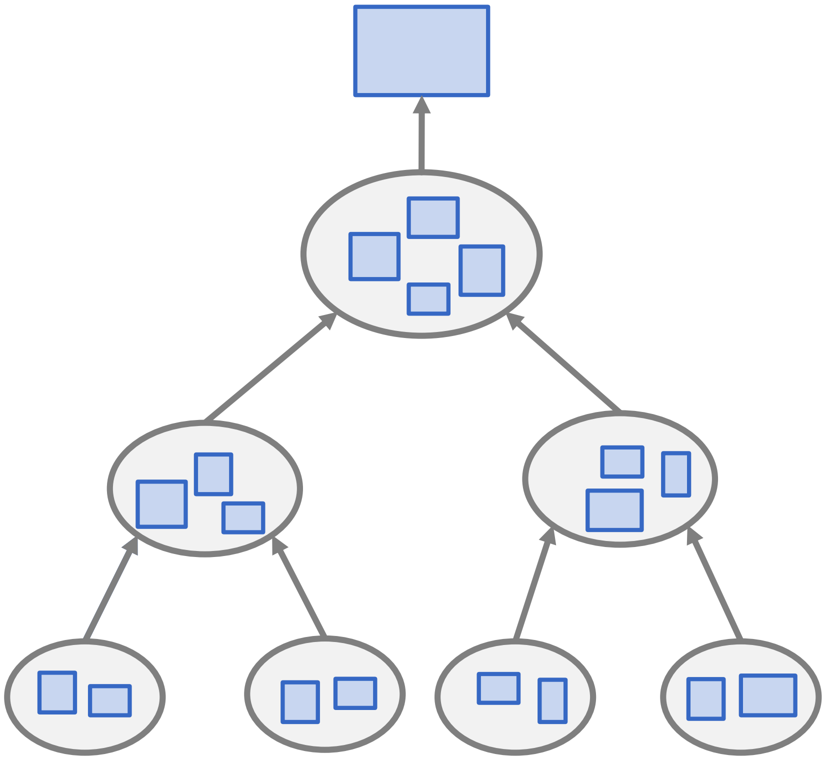
DETERMINING HOW MUCH TO CLUSTER

- $f(n) = n$: all primitives pushed to top of computation graph, AAC solution is same as true agglomerative clustering.



DETERMINING HOW MUCH TO CLUSTER

- We use $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.

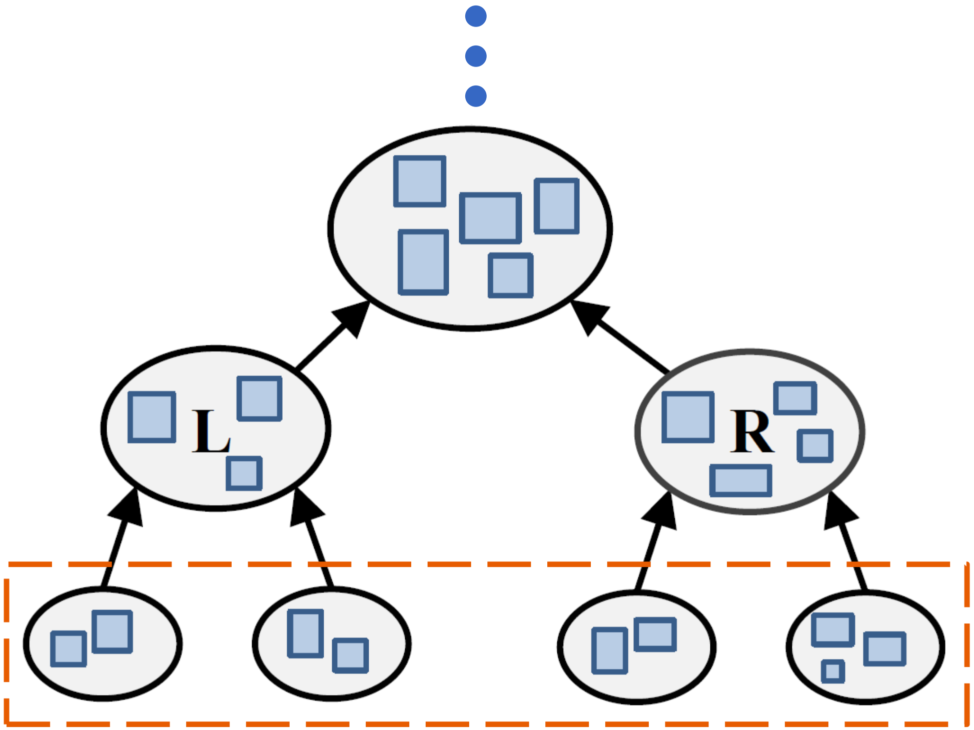


AAC HAS LINEAR TIME COMPLEXITY

- Downward phase is linear.
- Upward clustering phase:
  - Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.
  - Assumptions:
    - δ is a small constant.
    - Time complexity on each graph node is $O(n^2)$ [Olson 1995], where n is the number of input primitives in this node.

COMPLEXITY ANALYSIS

- Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.



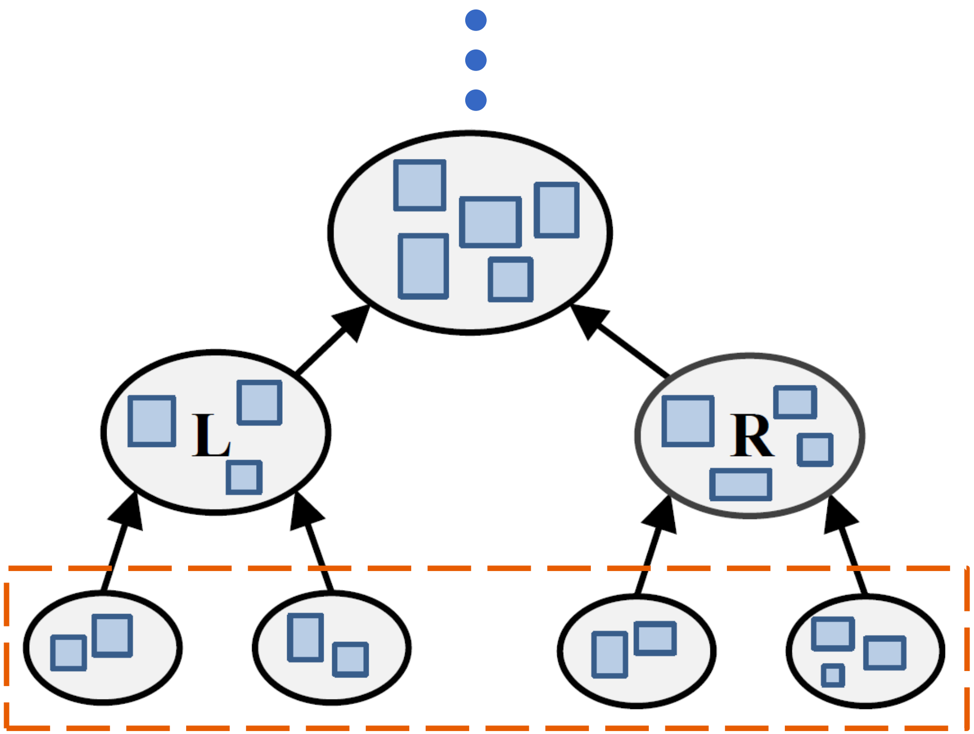
Work done at leaves:
$\frac{N}{\delta}$ nodes, $O(f(\delta)^2) = O(\delta^{2\alpha})$ computation each.

→ $O(N\delta^{2\alpha-1})$ work total.

Complexity analysis

- Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.

Let
$C = N\delta^{2\alpha-1}$



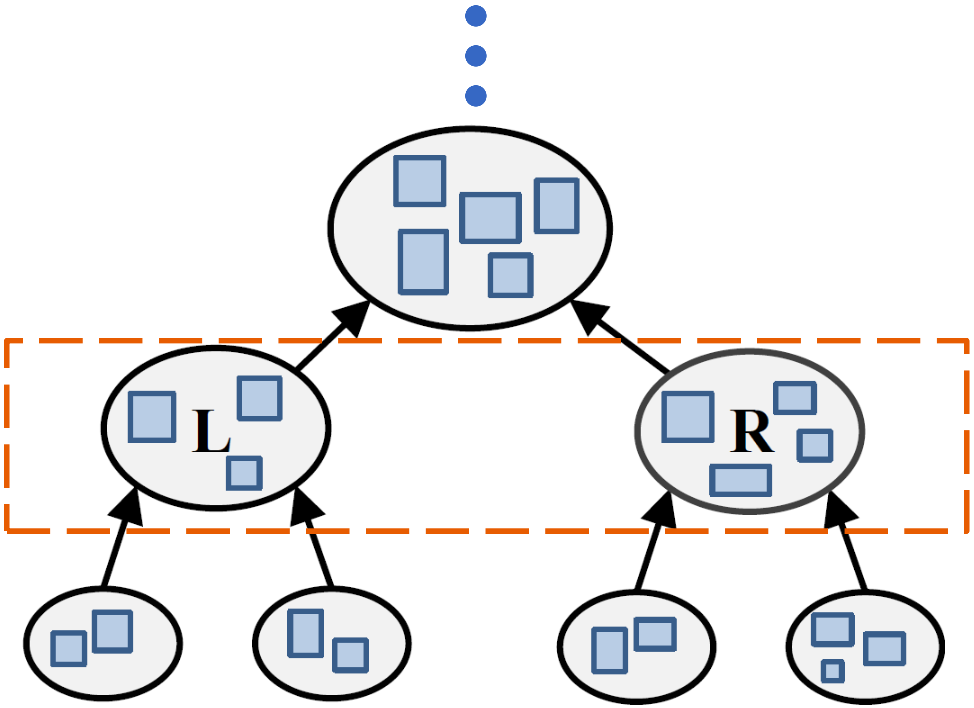
Work done at leaves:
$\frac{N}{\delta}$ nodes, $O(f(\delta)^2) = O(\delta^{2\alpha})$ computation each.

→ $O(C)$ work total.

Linear Time Complexity

- Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.

Let
$C = N\delta^{2\alpha-1}$



$\frac{N}{2\delta}$ nodes,
$O(f(2\delta)^2) = O\left((2\delta)^{2\alpha}\right)$
computation each.

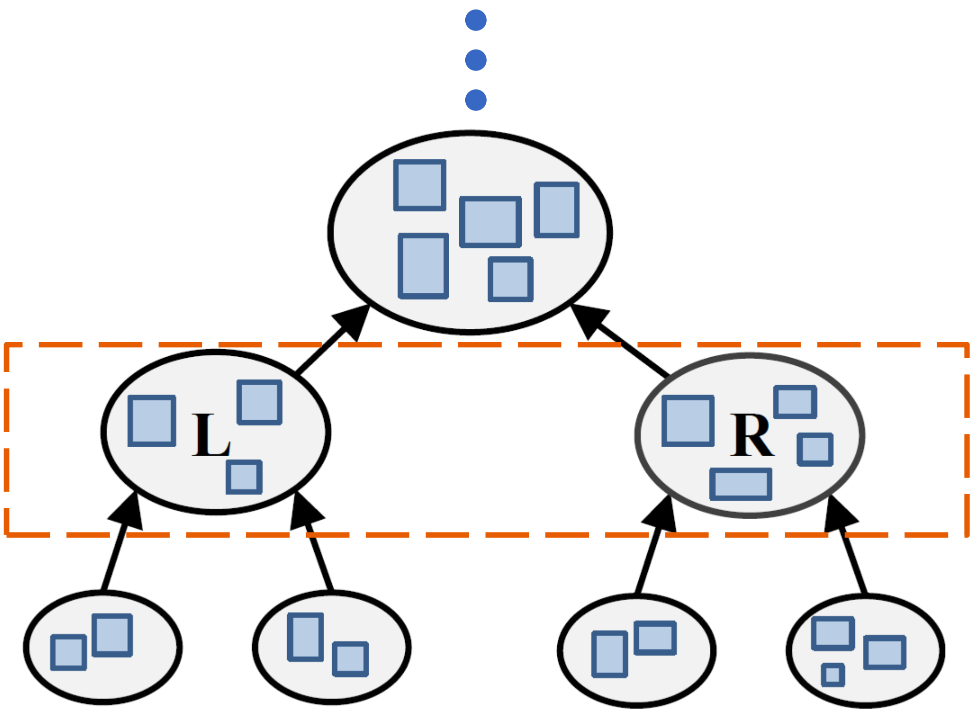
→ $O(N(2\delta)^{2\alpha-1})$ work total.

→ $O(C)$ work total.

Linear Time Complexity

- Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.

Let
$C = N\delta^{2\alpha-1}$
$r = 2^{2\alpha-1} < 1$



$O(C \cdot r)$ work total.

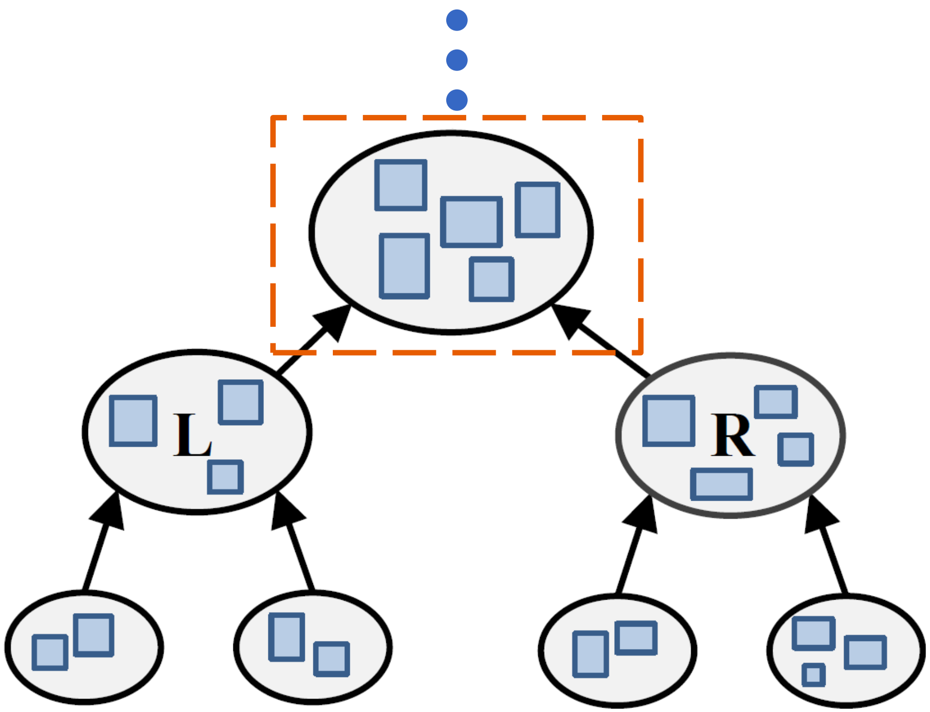
$O(C)$ work total.

Linear Time Complexity

- Let $f(n) = cn^{\alpha}$, where $0 < \alpha < 0.5$.

Geometrically decreasing.

Let
$C = N\delta^{2\alpha-1}$
$r = 2^{2\alpha-1} < 1$

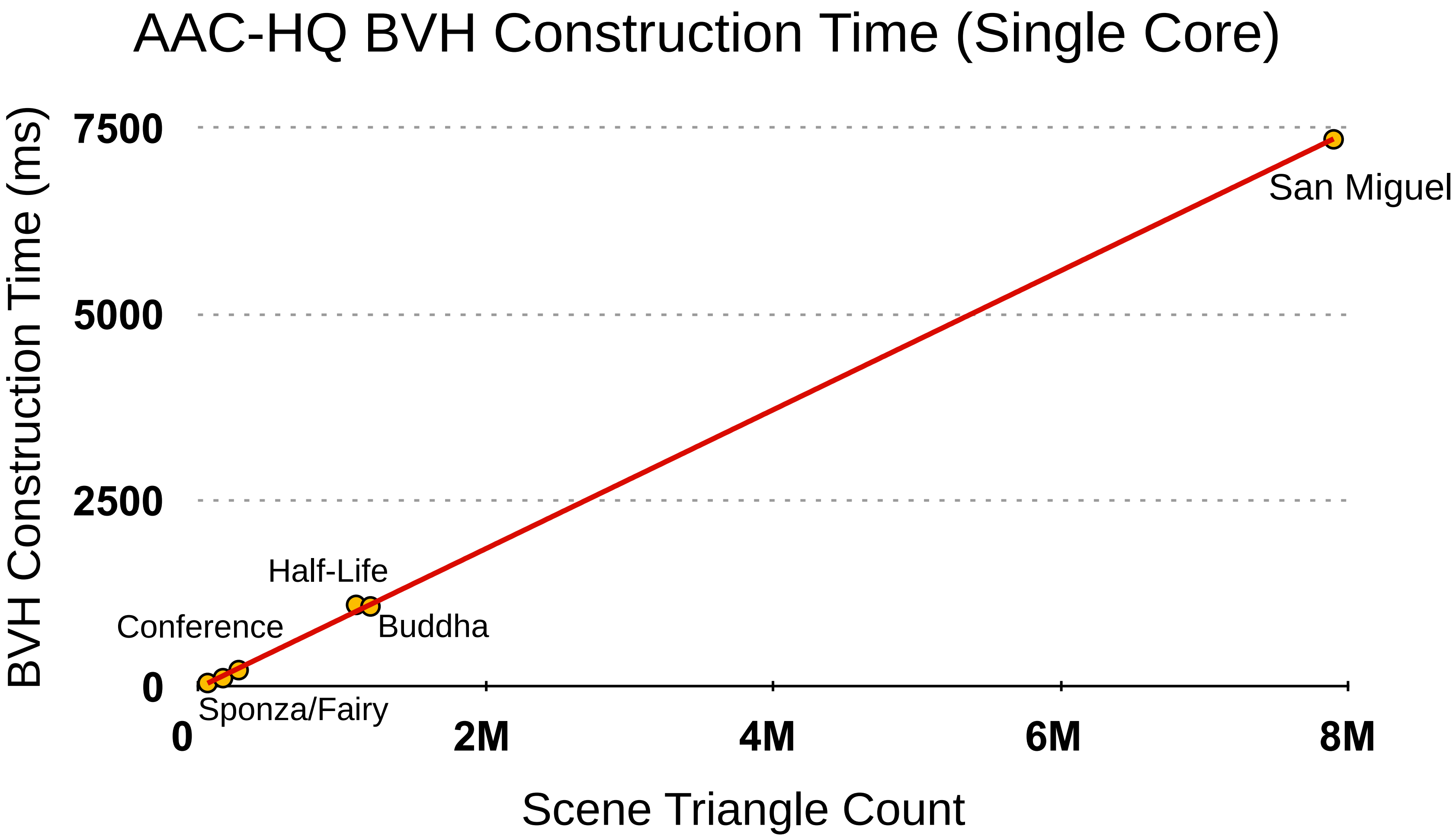


→ $O(C \cdot r^2)$ work total.

→ $O(C \cdot r)$ work total.

→ $O(C)$ work total.

Theory meets practice:
We observe linear scaling with scene size



PARAMETERS

- Trade off BVH quality and construction speed by changing δ and f in the algorithm.
- We proposed 2 sets of parameters:
  - AAC-HQ (high quality): $\delta = 20$, $f(n) = \frac{\delta^{0.6}}{2} \cdot n^{0.4}$;
  - AAC-Fast: $\delta = 4$, $f(n) = \frac{\delta^{0.7}}{2} \cdot n^{0.3}$.

IMPLEMENTATION DETAILS

- Parallelization:
  - Algorithm is divide-and-conquer, so very easy to parallelize.
- Key optimizations possible:
  - Reduce redundant computation of cluster distances;
  - Reducing data movement;
  - Sub-tree flatting for improved tree quality.

EVALUATION

SETUP

- We compared 5 CPU implementations

SAH	A standard top-down full-sweep SAH build [MacDonald and Booth 1990]
SAH-BIN	A top-down “binned” SAH build using at most 16 bins along the longest axis [Wald 2007]
Local-Ord	Locally-ordered agglomerative clustering [Walter et al. 2008]
AAC-HQ	AAC with high quality settings: $\delta = 20$, $f(n) = 3n^{0.4}$
AAC-Fast	AAC configured for performance: $\delta = 4$, $f(n) = 1.3n^{0.3}$

Scenes

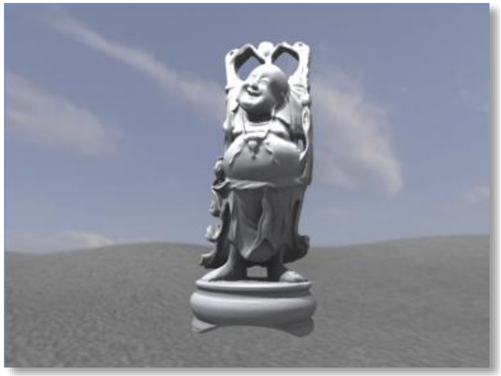
Sponza

Half-Life

Conference

Fairy

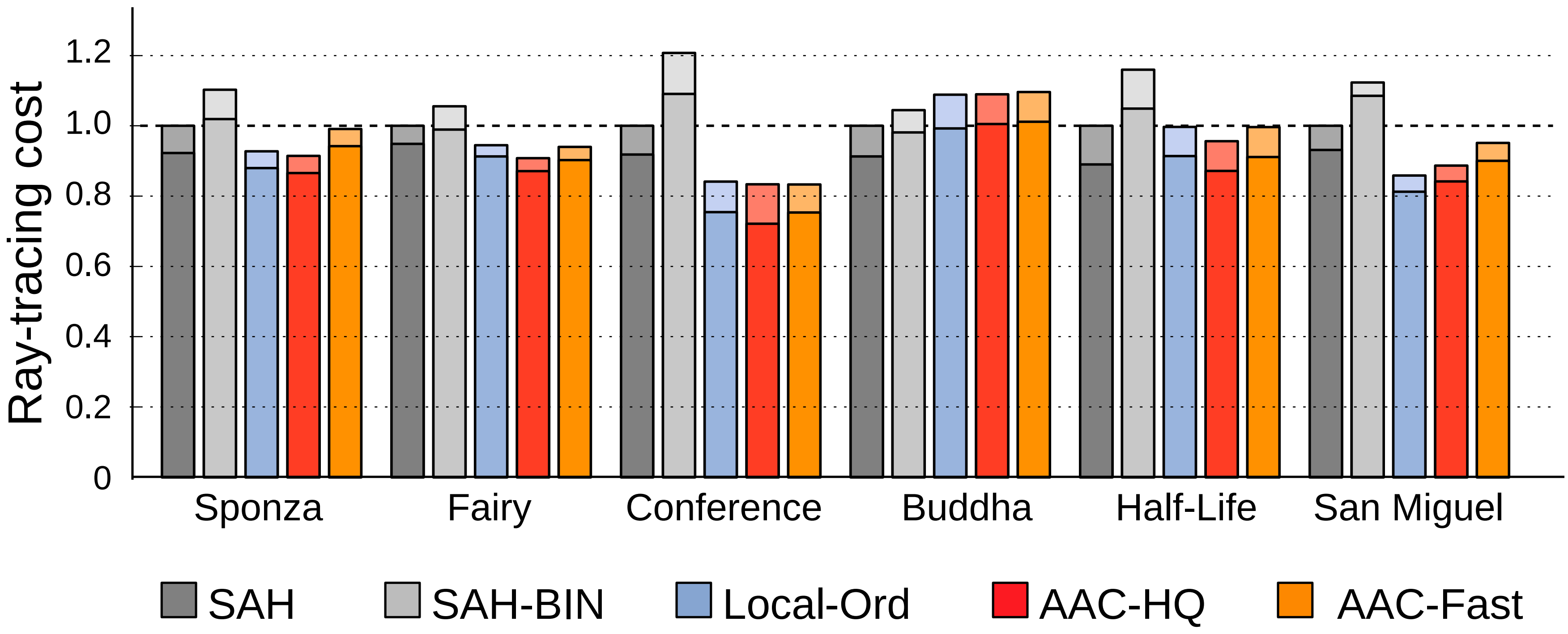
San Miguel



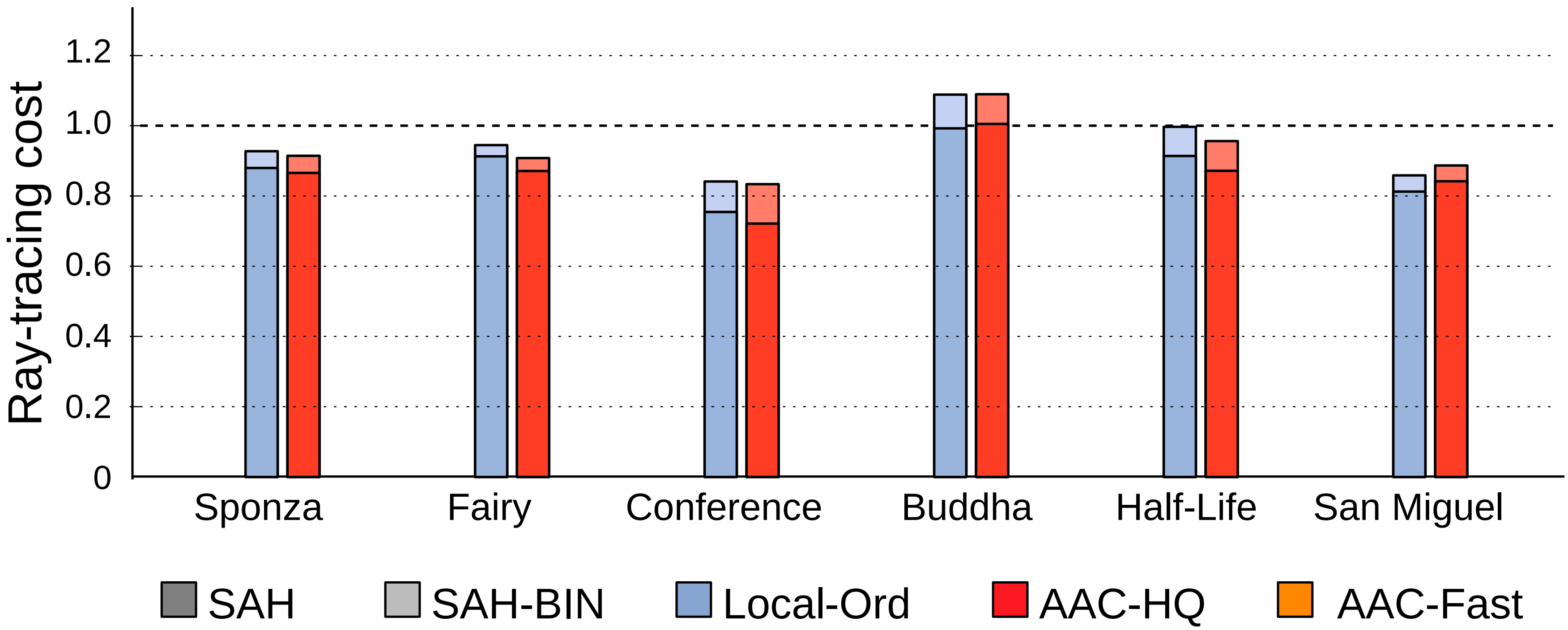
Buddha

TREE COST COMPARISON

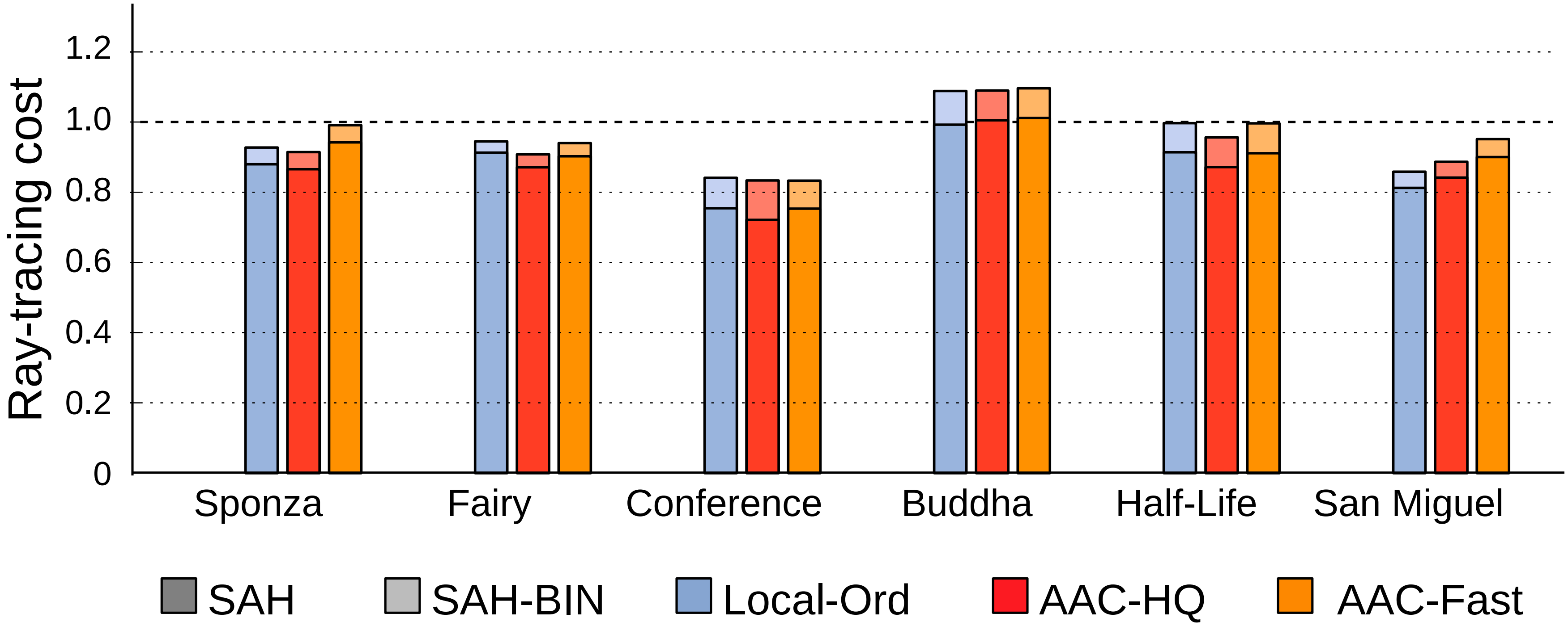
- Cost = number of traversal steps + intersection tests during ray tracing.



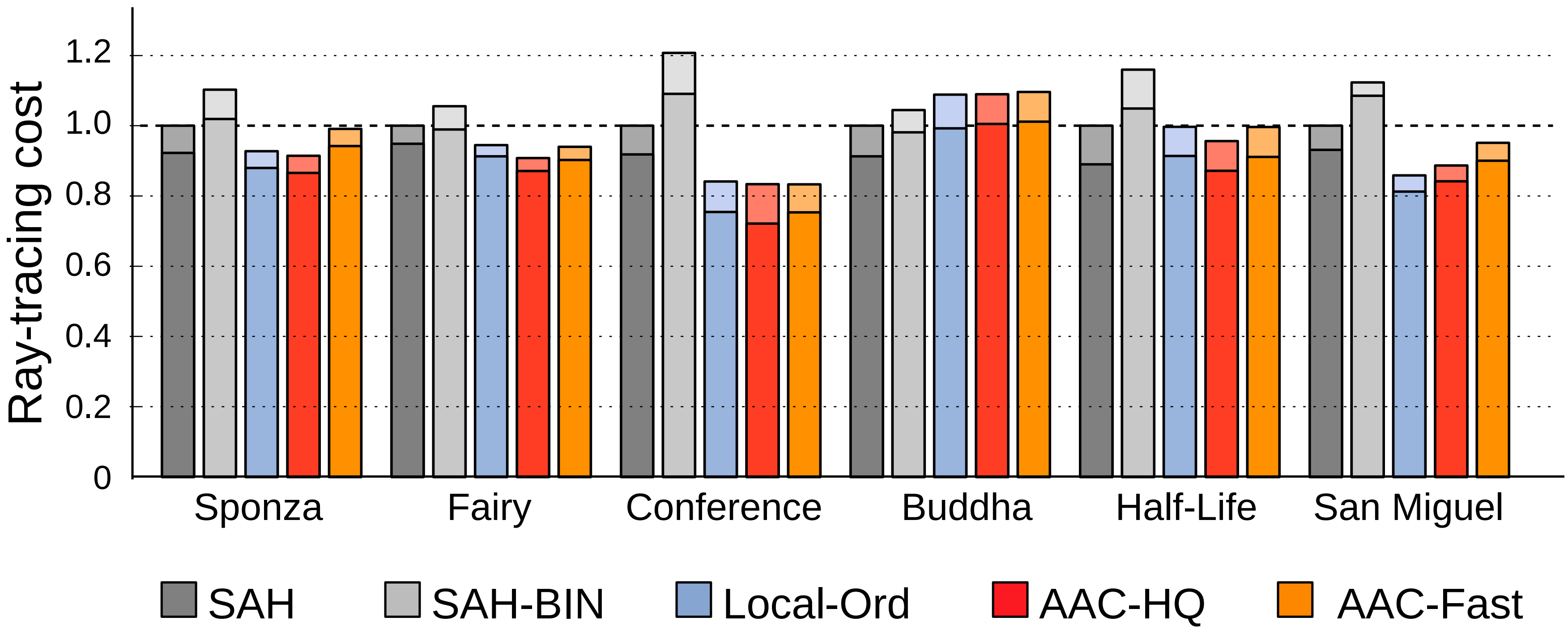
AAC-HQ produces BVHs that have similar cost as those produced by true agglomerative clustering builds.



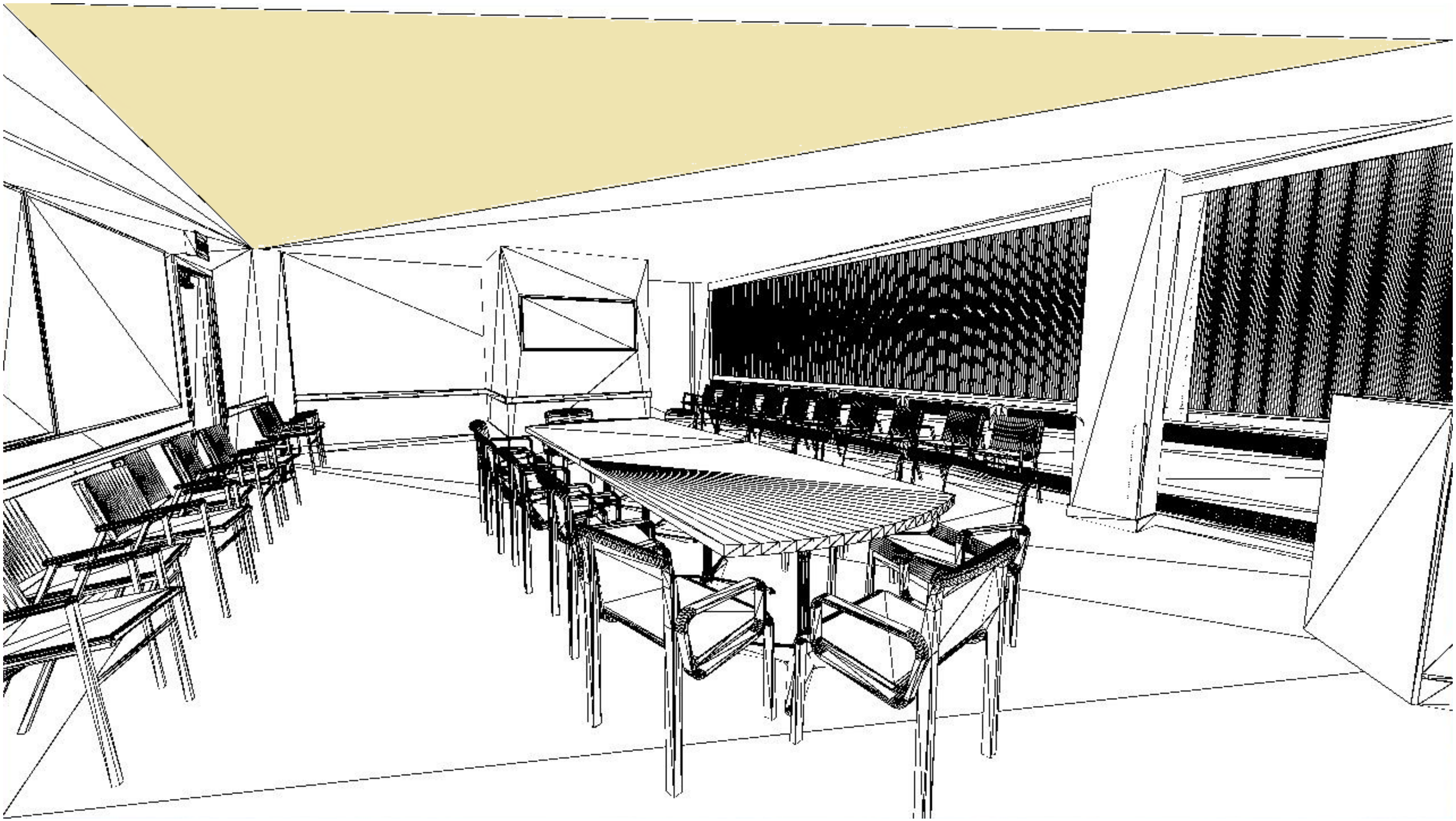
AAC-Fast produces BVHs with equal or lower cost than the **full sweep build** in all cases except Buddha.



AAC-Fast produces BVHs with equal or lower cost than the **full sweep build** in all cases except Buddha.

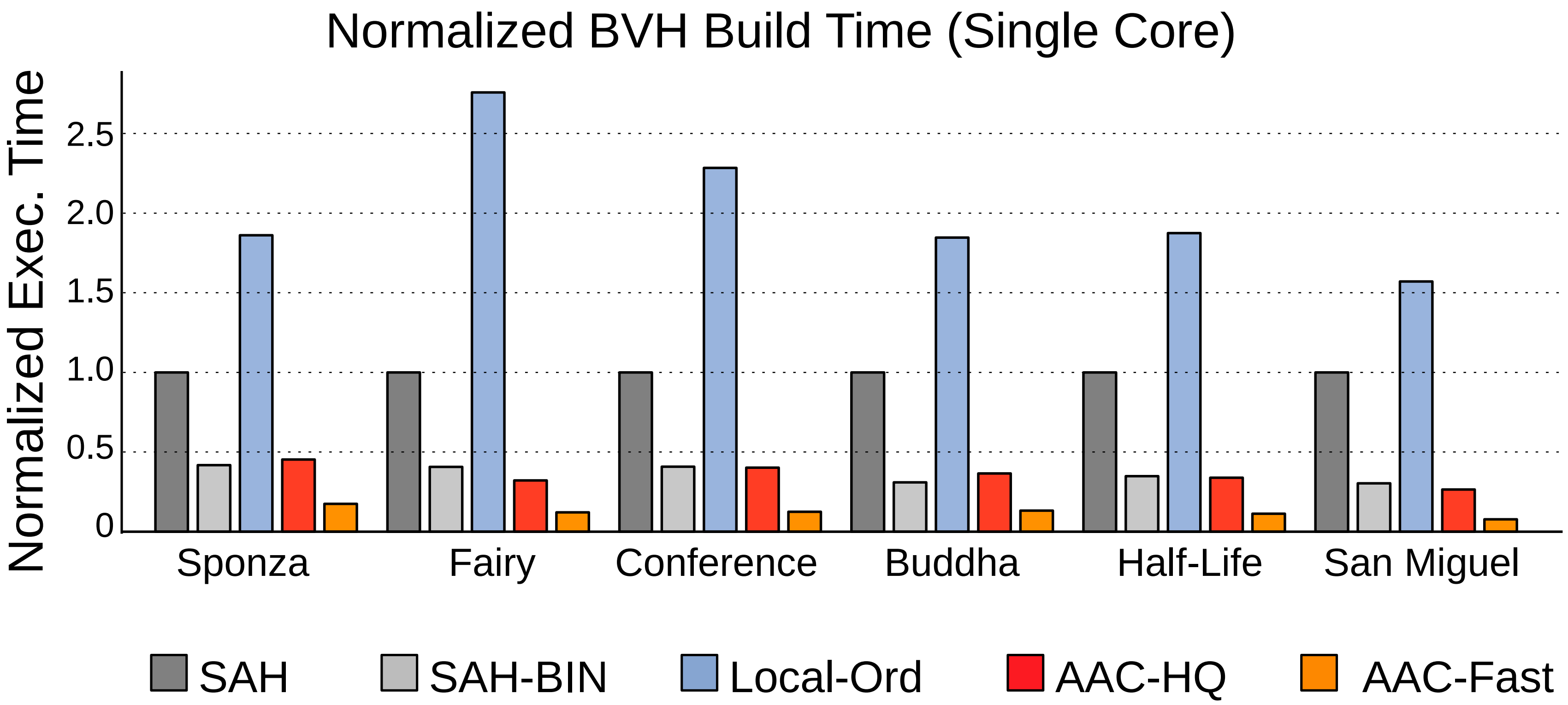


AAC IS ABLE TO MAKE PARTITIONS THAT ARE NOT DETERMINED BY PARTITION PLANES.



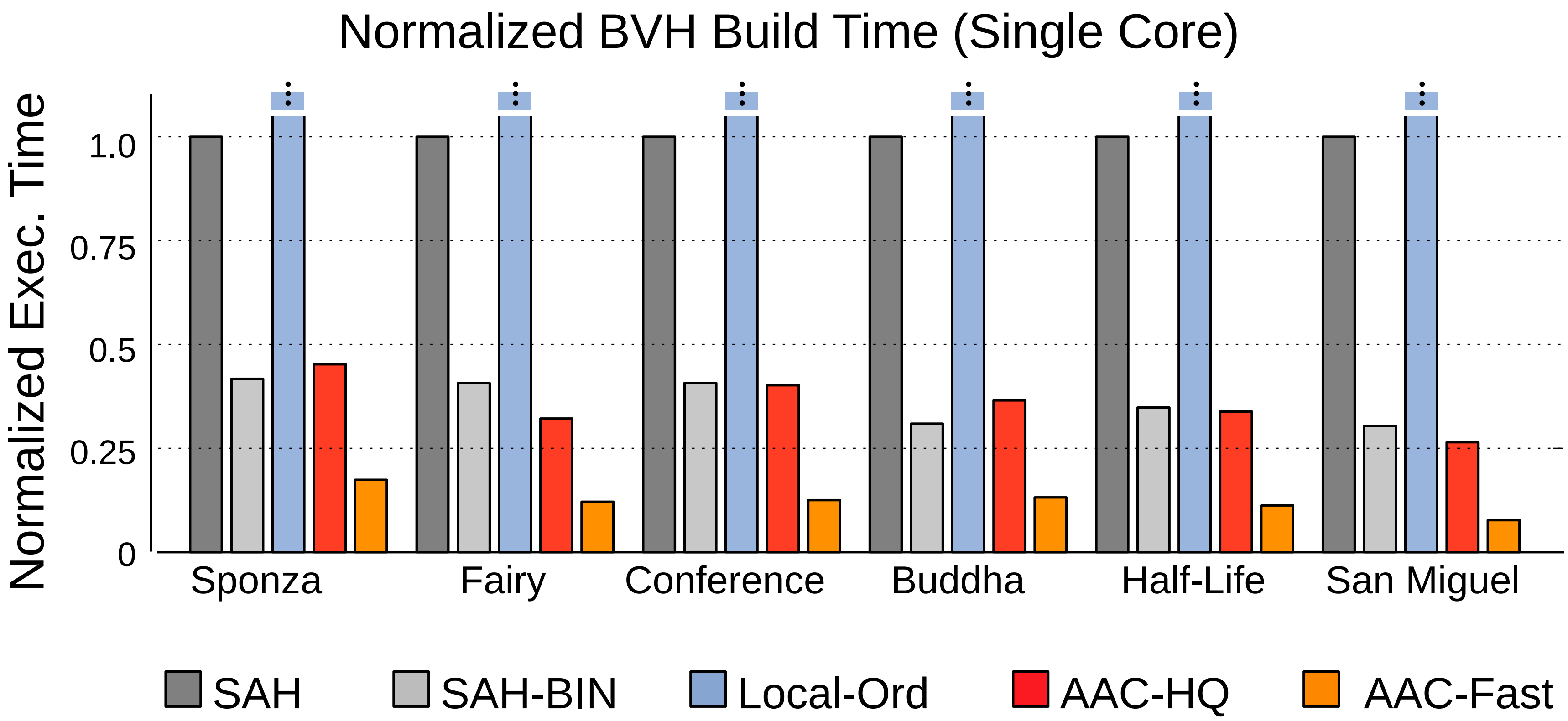
BVH CONSTRUCTION TIME (SINGLE CORE)

- AAC-HQ build times are five to six times lower than Local-Ord (while maintaining comparable BVH quality)



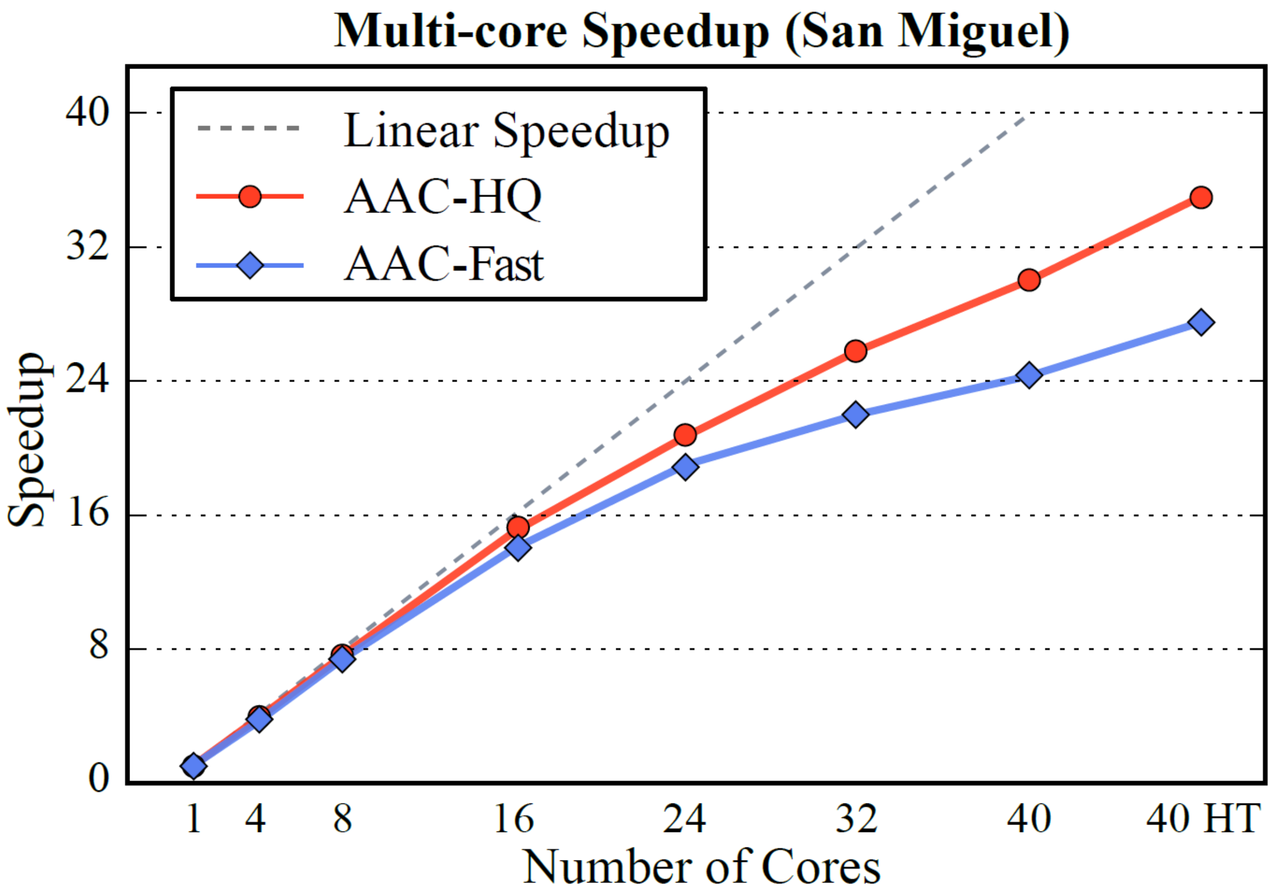
AAC-HQ build times are comparable to SAH-BIN

AAC-Fast build times up to four times faster than SAH-BIN



AAC PARALLEL EXECUTION SPEEDUP

AAC-HQ achieves nearly linear speedup out to 16 cores, and a 34× speedup on 40 cores



AAC 32-CORE SPEEDUP

AAC Build Execution Times (milliseconds) and Parallel Speedup

		AAC-HQ			AAC-Fast		
	Tri Count	**1 core**	**32 cores**		**1 core**	**32 cores**	
Sponza	67 K	52	2	**(24.0)**	20	1	**(21.5)**
Fairy	174 K	117	5	**(24.5)**	44	2	**(22.4)**
Conference	283 K	225	10	**(23.6)**	70	4	**(19.4)**
Buddha	1.1 M	1,101	43	**(25.8)**	397	16	**(24.0)**
Half-Life	1.2 M	1,080	42	**(25.7)**	359	15	**(22.8)**
San Miguel	7.9 M	7,350	298	**(24.6)**	2,140	99	**(21.6)**

Summary

- AAC algorithm: BVH construction via an approximation to agglomerative clustering of scene primitives
  - Comparable quality BVH to full sweep SAH build
  - Up to four-times faster than binned SAH build
  - Amenable to parallelism on many-core CPUs

Similarity to Karras13 (next talk)

- Fast initial organization of scene primitives via Morton codes
  - AAC: to define constraints on clustering
  - Karras13: to define initial BVH
- “Brute-force” optimization of local sub-structures
  - AAC: brute-force local clustering in each node
  - Karras13: brute-force enumeration of treelet structures
  - In both: more flexible partitions than defined by spatial partition plane
- AAC does not address triangle splitting

Looking forward

- Have not yet explored parallelization of AAC on GPUs
- Post-process BVH optimizations can be applied on a smaller set of clusters generated by AAC
- Clustering in low dimensional space has many other applications in computer graphics including:
  - Lighting (e.g., Light Cuts)
  - N-body simulation
  - Collision detection

Thank you

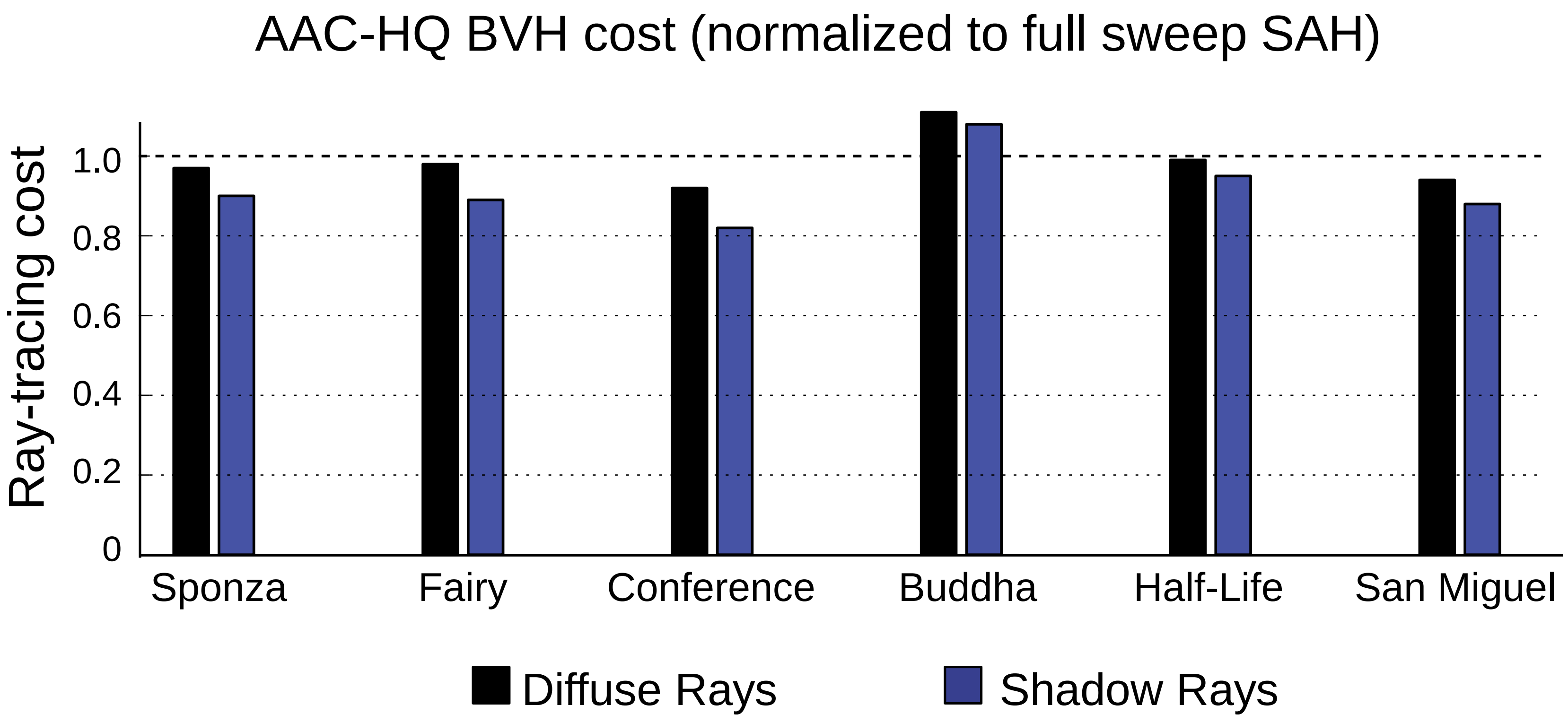
We acknowledge the support of:

The National Science Foundation (CCF-1018188)

Intel Labs Academic Research Office

NVIDIA corporation

BVHs produced by AAC methods realize greater benefit for shadow rays than diffuse bounce rays.



WHY AAC PERFORMS WORSE FOR BUDDHA.