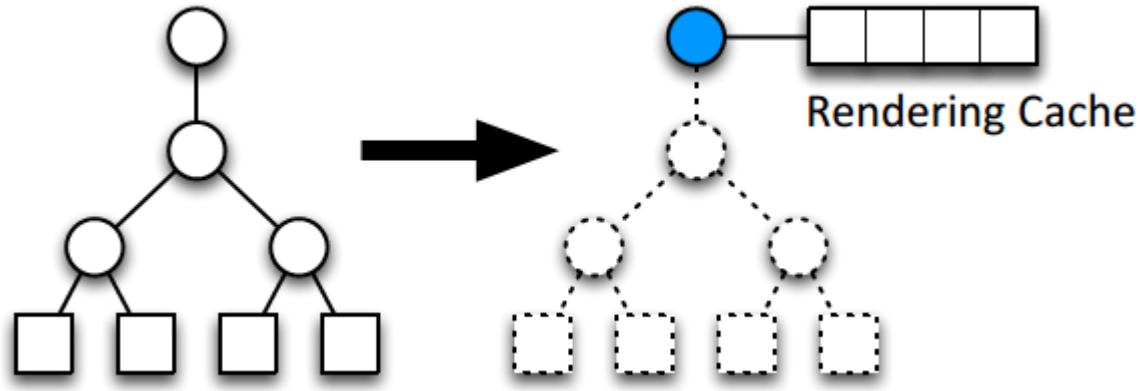



# Lazy Incremental Computation for Efficient Scene Graph Rendering

**Michael Wörister, Harald Steinlechner, Stefan Maierhofer, and Robert F. Tobler**

VRVis Research Center
Vienna, Austria

vrvis

# (Hierarchical) Scenes modelled Scene Graphs

**Regular scene graph rendering algorithm can become inefficient**

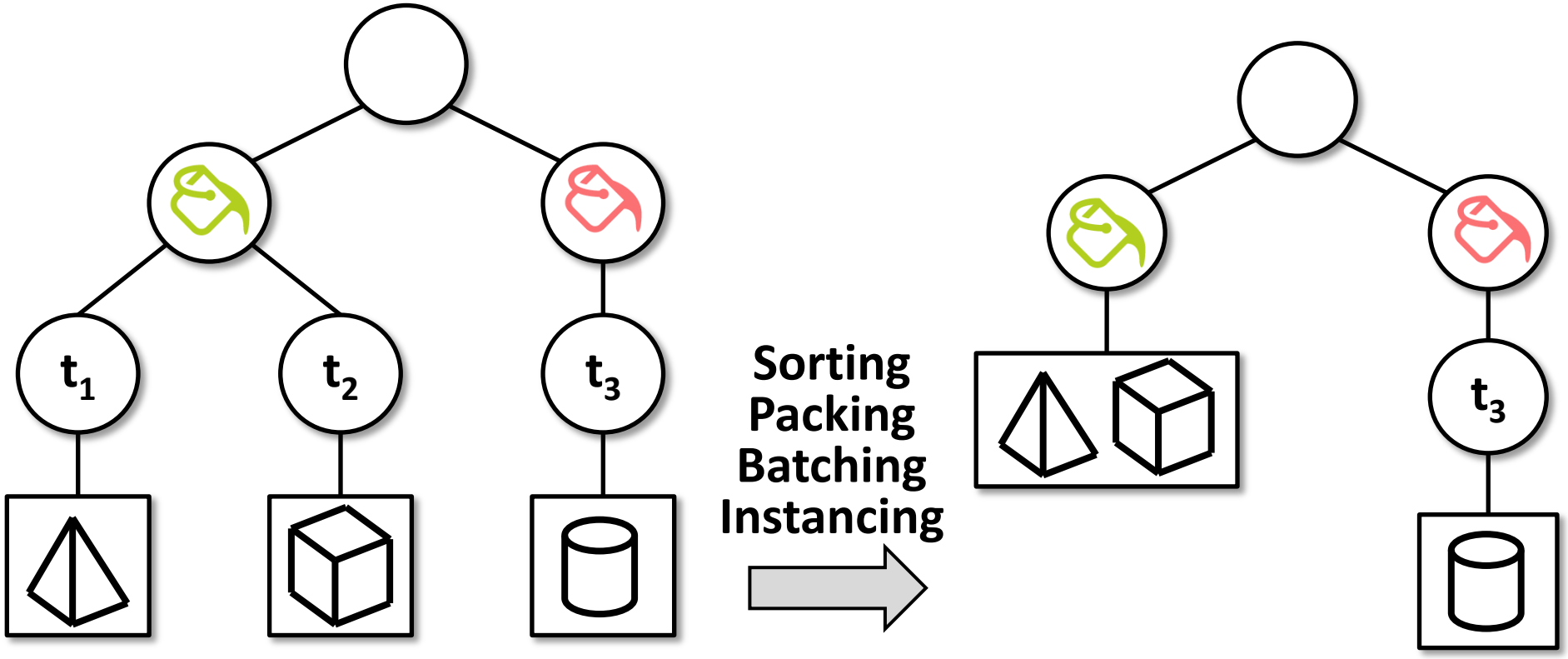
- for large number of nodes / paths

**Profiling shows: much time spent with traversal overhead**

- matrix multiplications
- virtual function calls
- …

**→ scene graph traversal becomes performance bottleneck**

# Scene graph Optimizations



# Optimizations affect Model/Design

## Optimizations leaks into application

- Mutation of modeling datastructure ruins clean semantic view of the scene

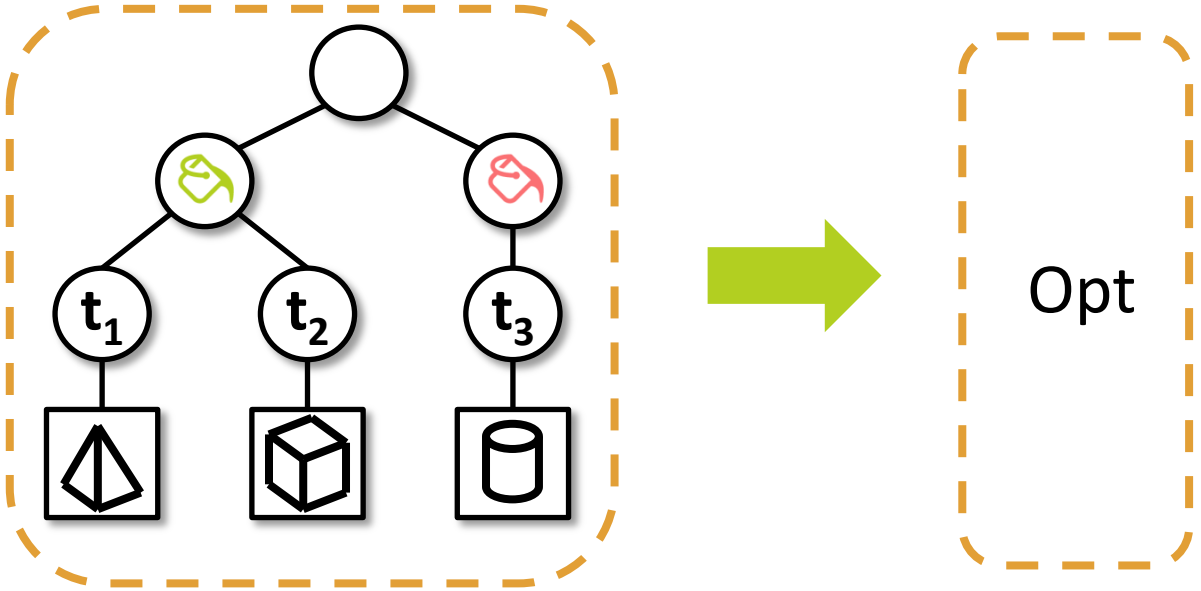
## Interaction of optimization with dynamic scenes

- Some optimizations not valid in general (blending)
- Expensive geometry/texture packing

## How to combine with dynamic scenes? CONFLICT

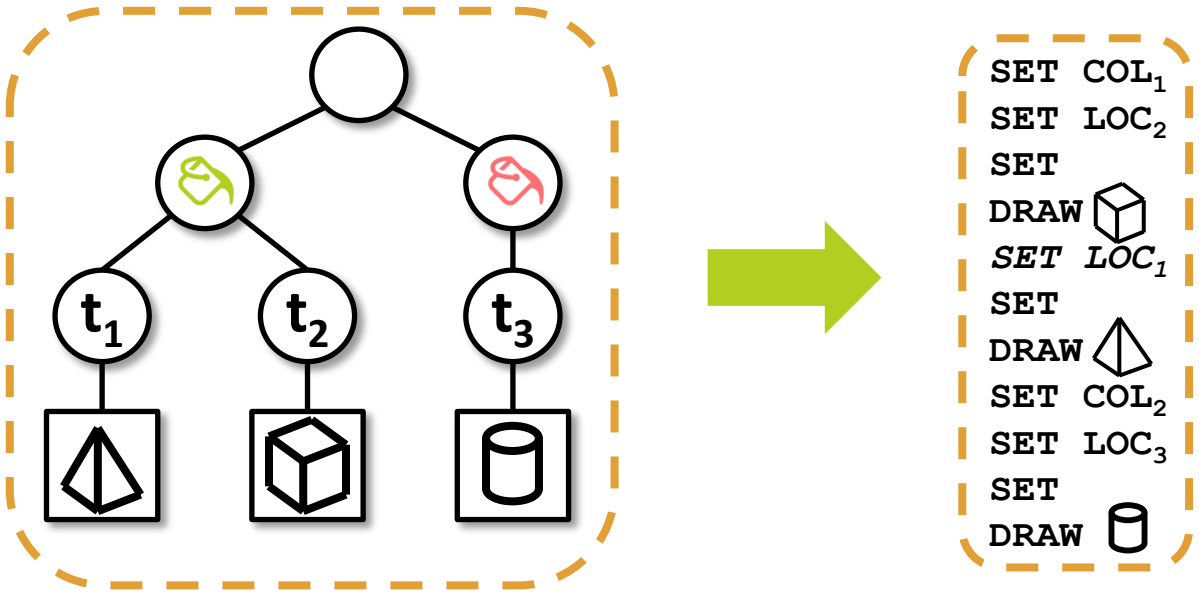
# Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure



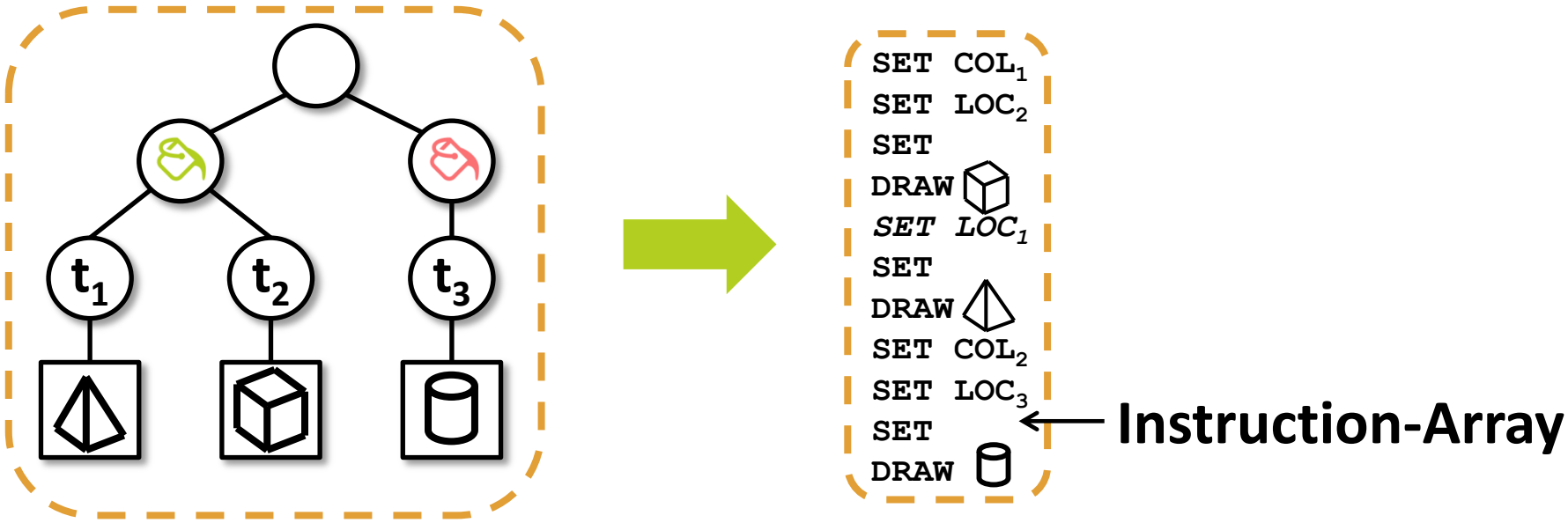
# Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure



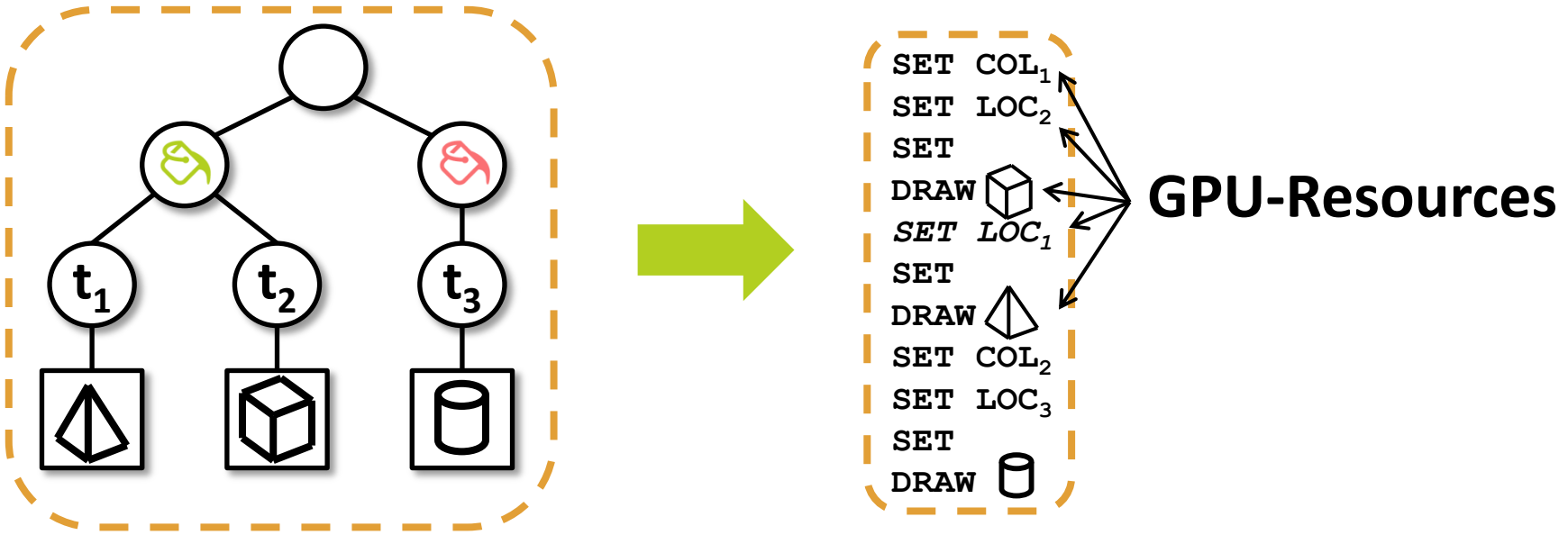
# Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure



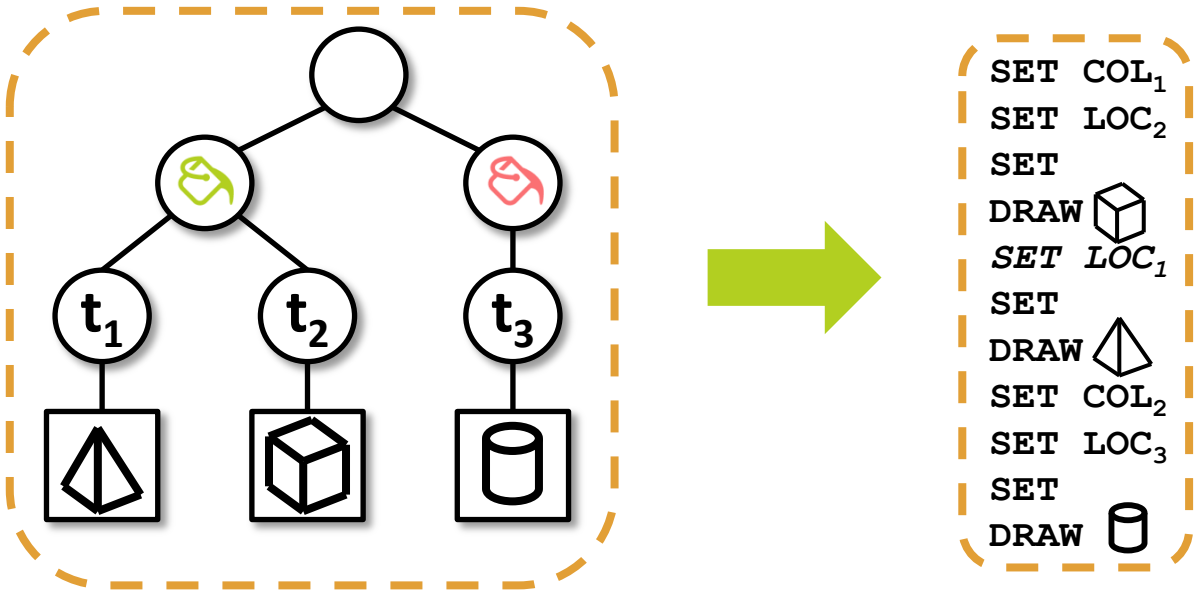
# Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure



# Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure



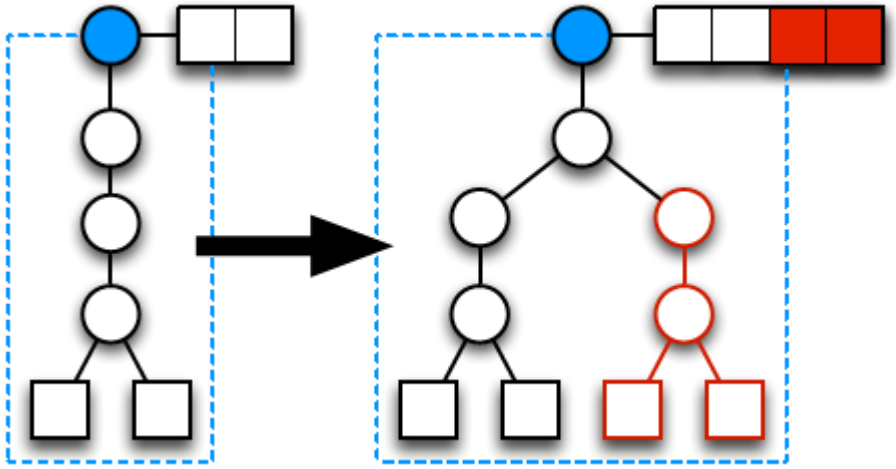

**Keep optimization datastructure in sync!**

# Efficient Synchronization

- Changes in Scene graph
- Modification to the tree or attributes
- Change propagation in: $O(|\ AFFECTED\ |)$
- In-place updates / Structural updates
- In this work: fast In-place updates

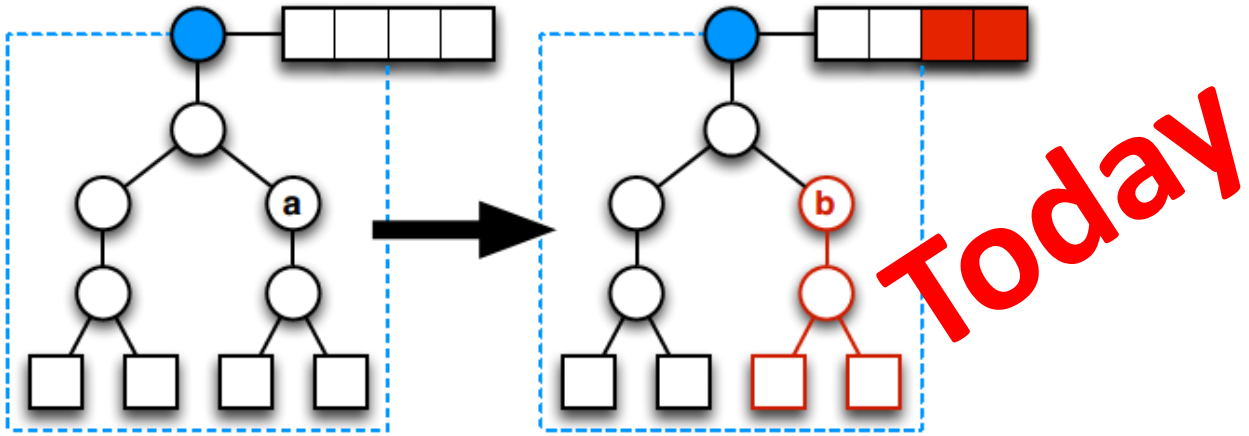
# Efficient Synchronization

- Changes in Scene graph
- Modification to the tree or attributes
- Change propagation in: $O(|\ AFFECTED\ |)$
- In-place updates / Structural updates
- In this work: fast In-place updates

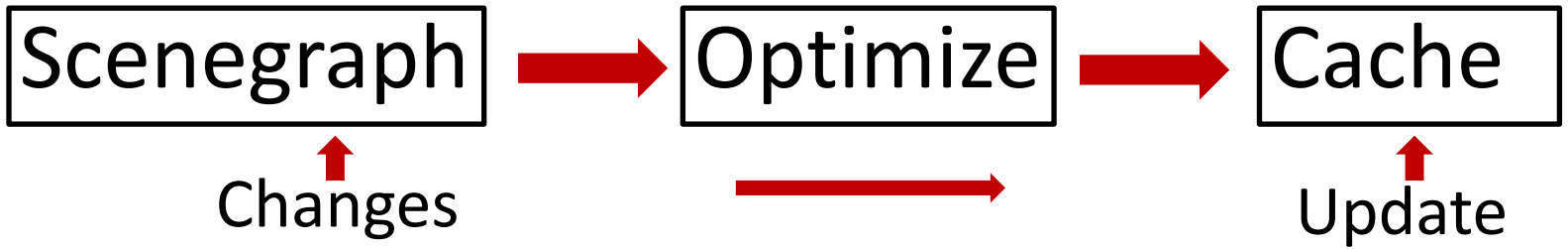


# Efficient Synchronization

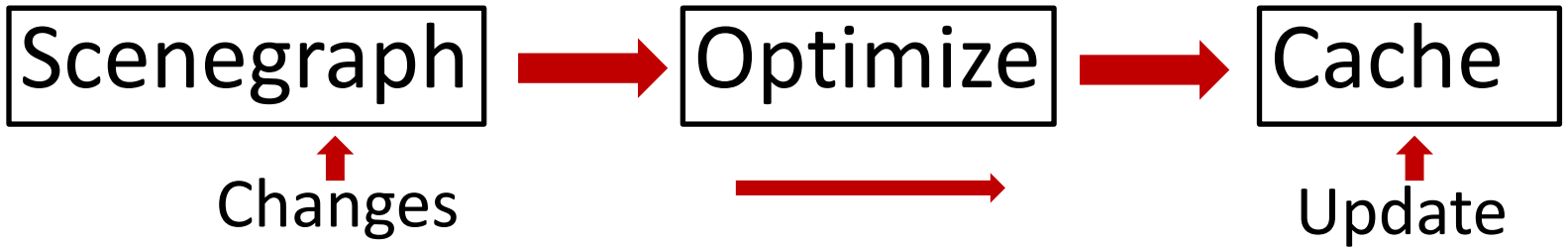
- Changes in Scene graph
- Modification to the tree or attributes
- Change propagation in: $O(|\ AFFECTED\ |)$
- In-place updates / Structural updates
- In this work: fast In-place updates



# Incremental Computation



# Incremental Computation



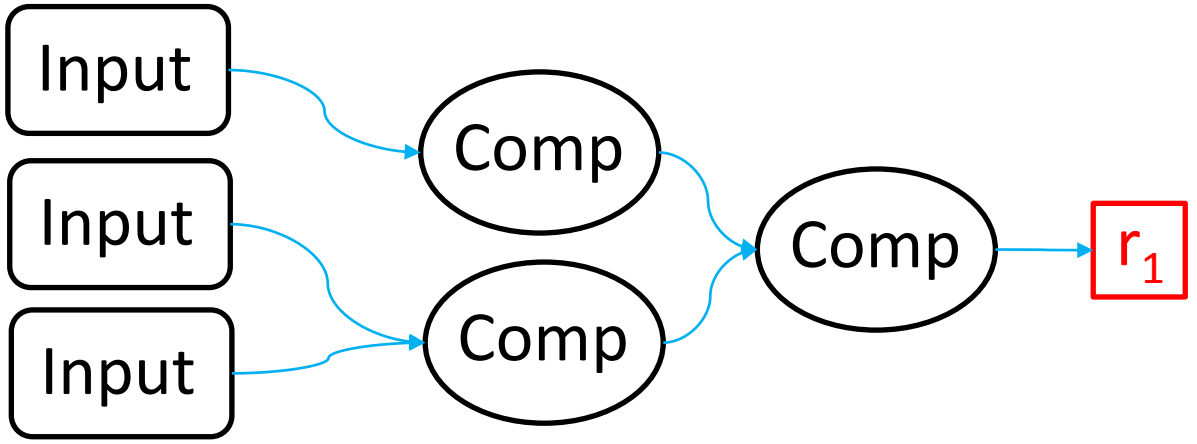

- Given input *x* and *f(x),* find changes of *f(x)* given changes in *x*

- Originally used for Attribute Evaluation for Attribute Grammars
- Builds on **static dependency Graph**

# Dependency Graphs

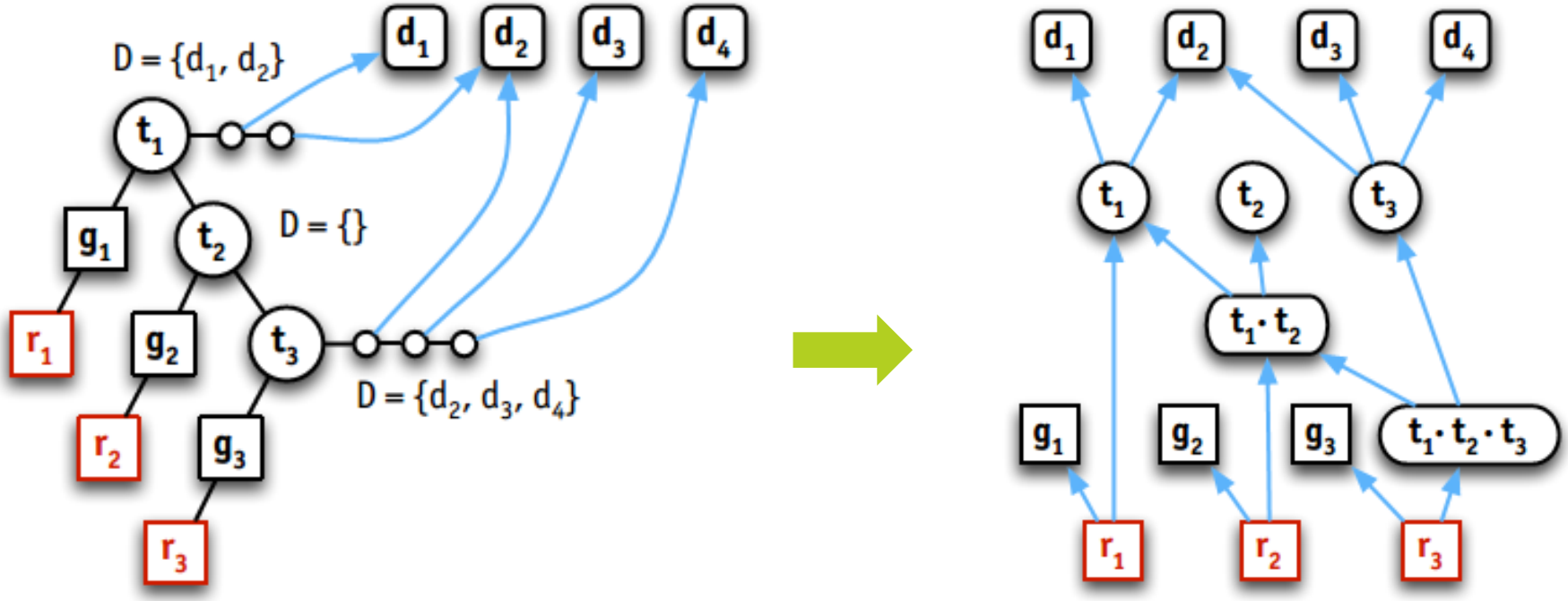
## Dependency Graphs used in

- Build systems (like *make*)
- Compilers (Data/Flow dependencies)
- Visual programming (like Hypergraph, Hypershade)



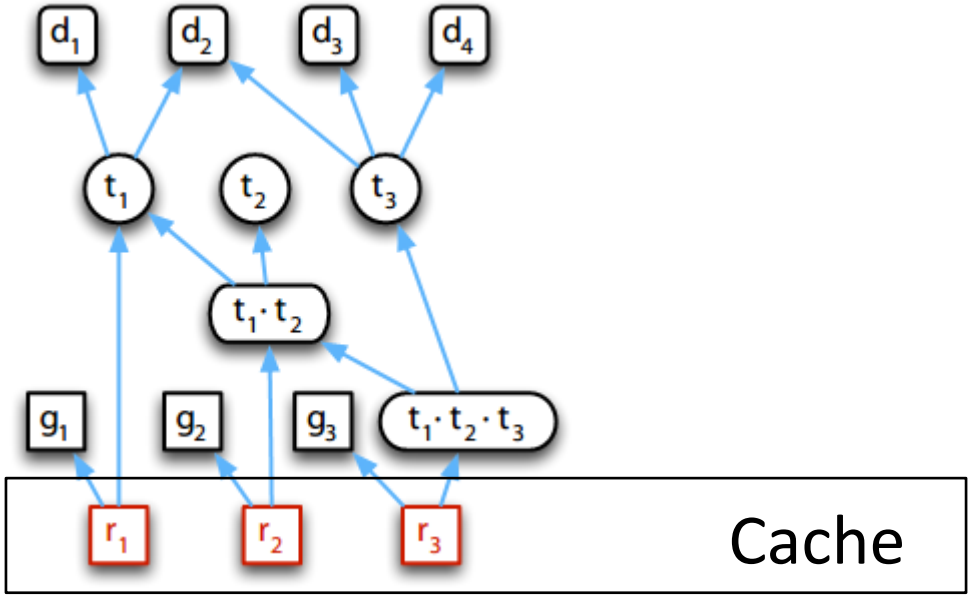
# The Implied Dependency Graph

- Geometry node → Leaf node
- Dependency in Sg → Dependency Node
- Computation → Dependency Node



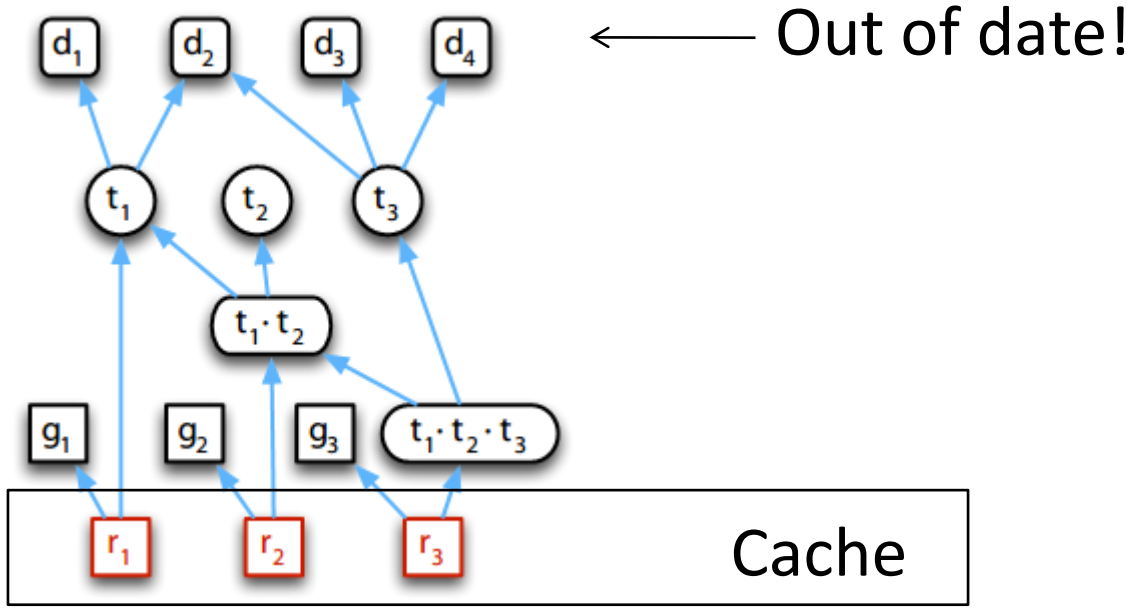
# Towards lazy attribute evaluation

- *Standard Optimal Algorithm* [Reps et al. 1983] not suitable
- Scene graphs are DAGs, parts may be culled
- Demand driven approach by [Hudson 1991]



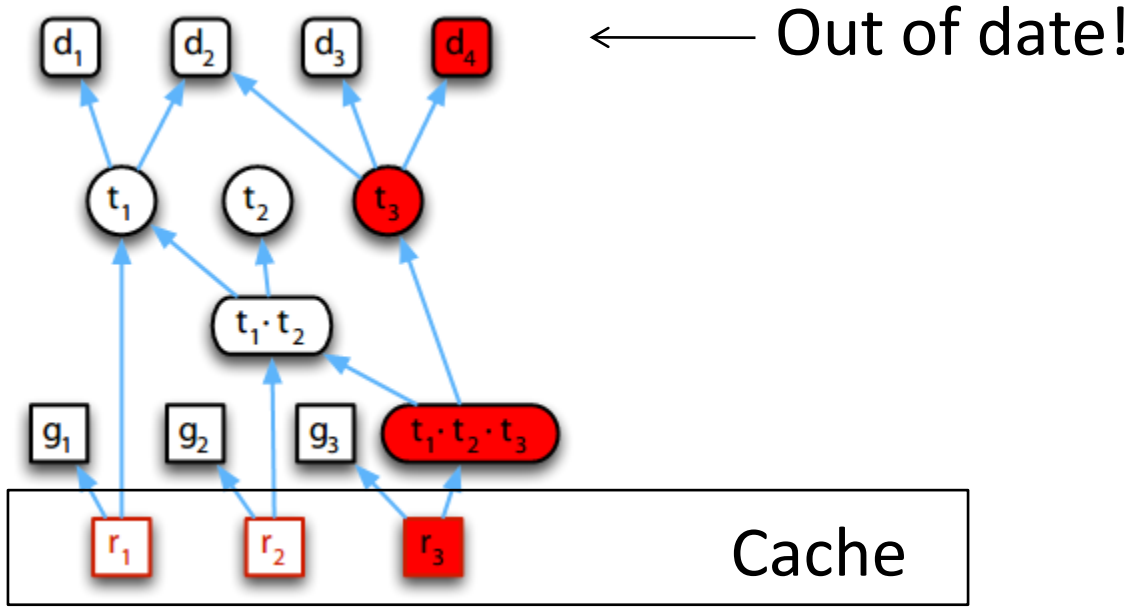
# Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking



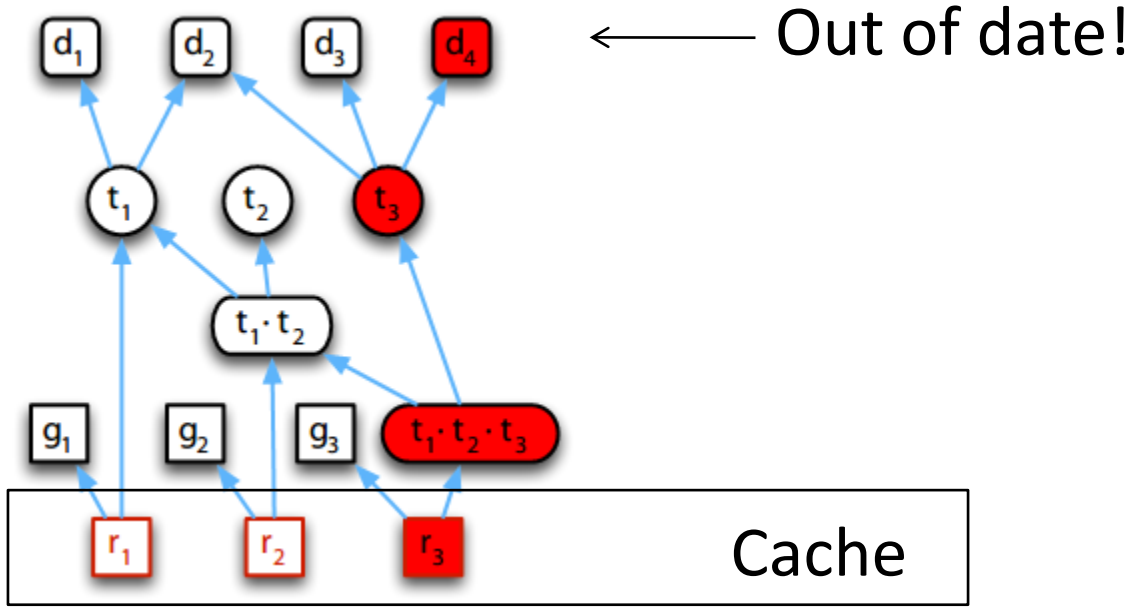
# Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking



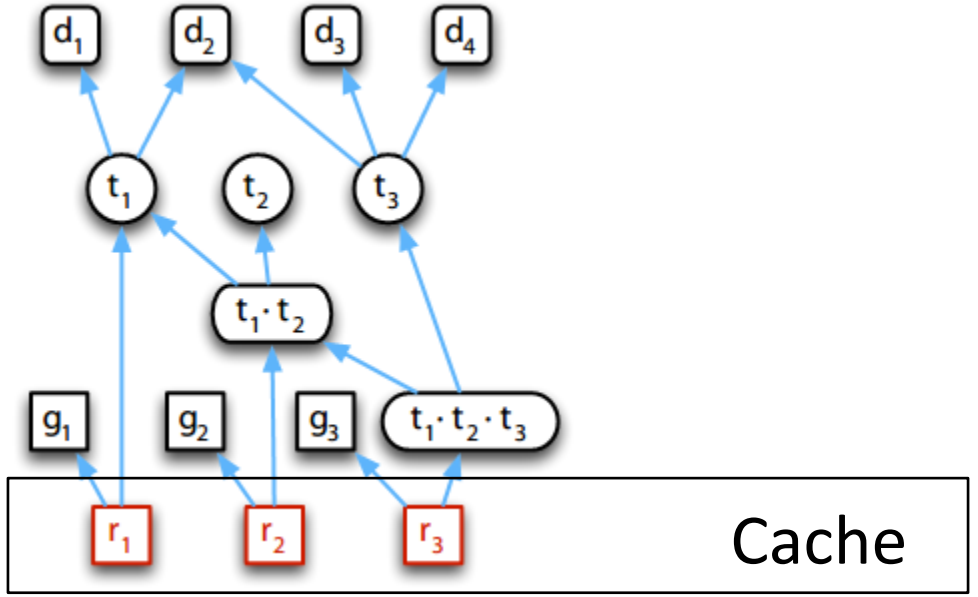
# Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking
- Step 2: Update required values (recompute nodes which are out of date)



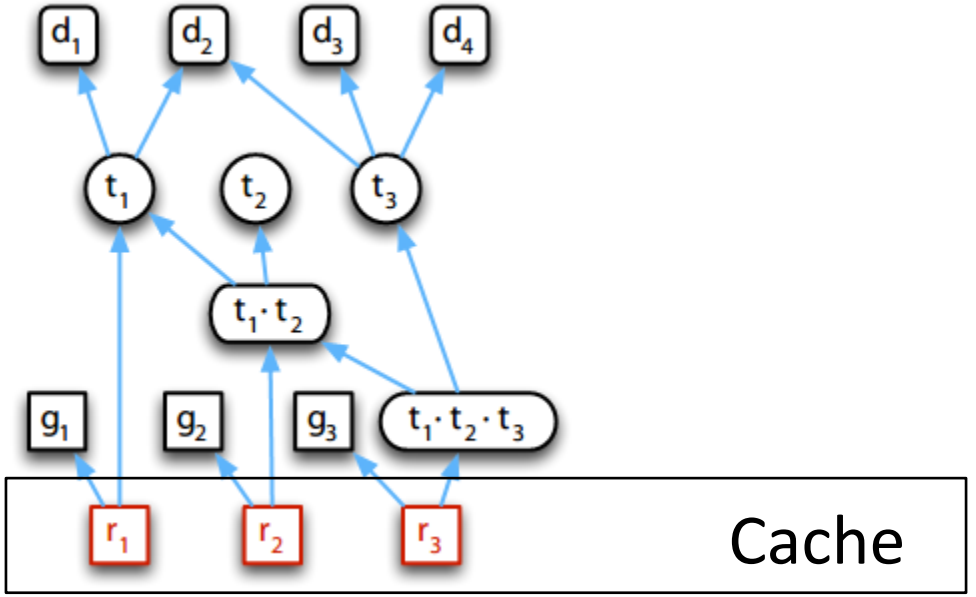
# Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking
- Step 2: Update required values (recompute nodes which are out of date)



# Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking
- Step 2: Update required values (recompute nodes which are out of date)
- Step 3: **Render**



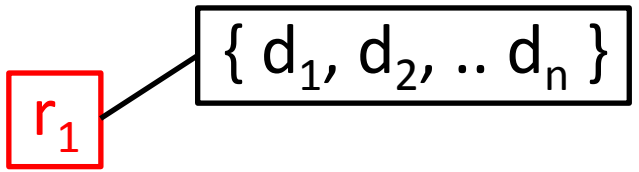
# Not yet there: marking is eager

## Large parts not visible

- Marking not necessary/feasable
- Replace eager marking with **lazy polling**

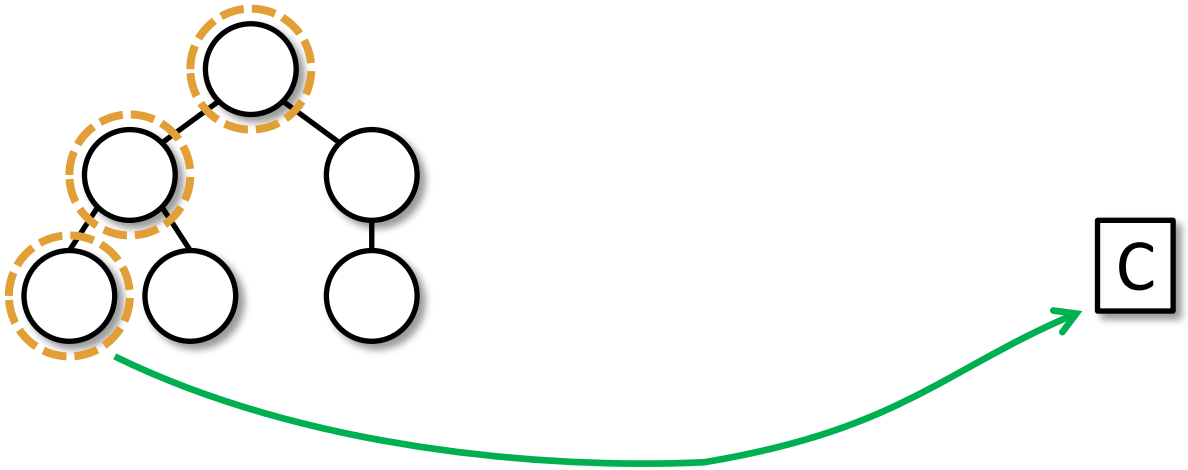
## Keep list of transitive reachable Dependencies

- Check for all predicates directly at cache entry



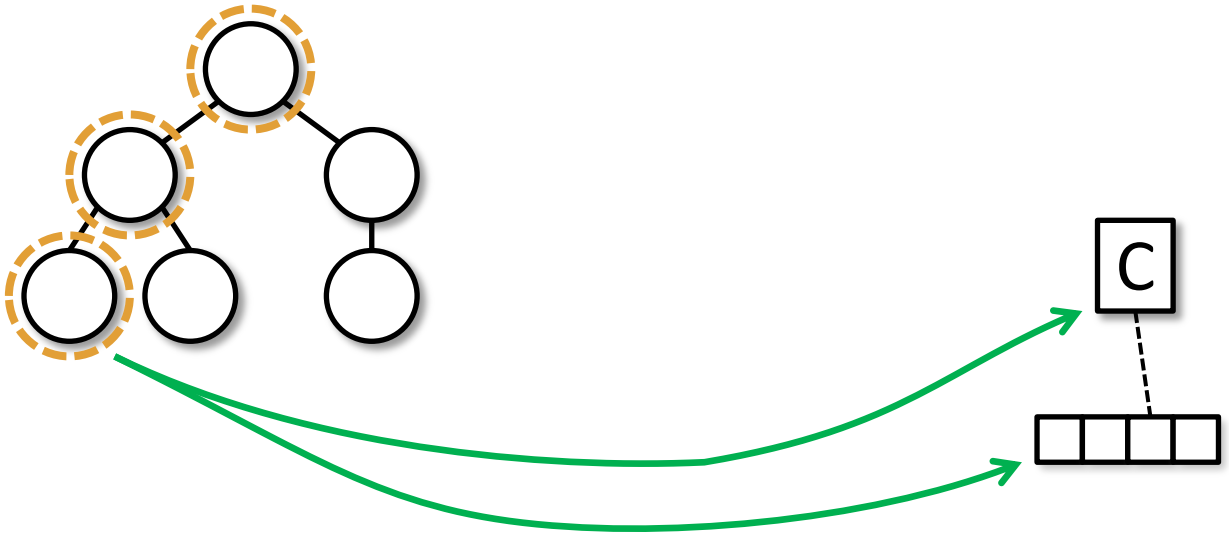
# Building an incremental Render Cache

Create cache entry for instruction parameters (in graphics memory)



# Building an incremental Render Cache

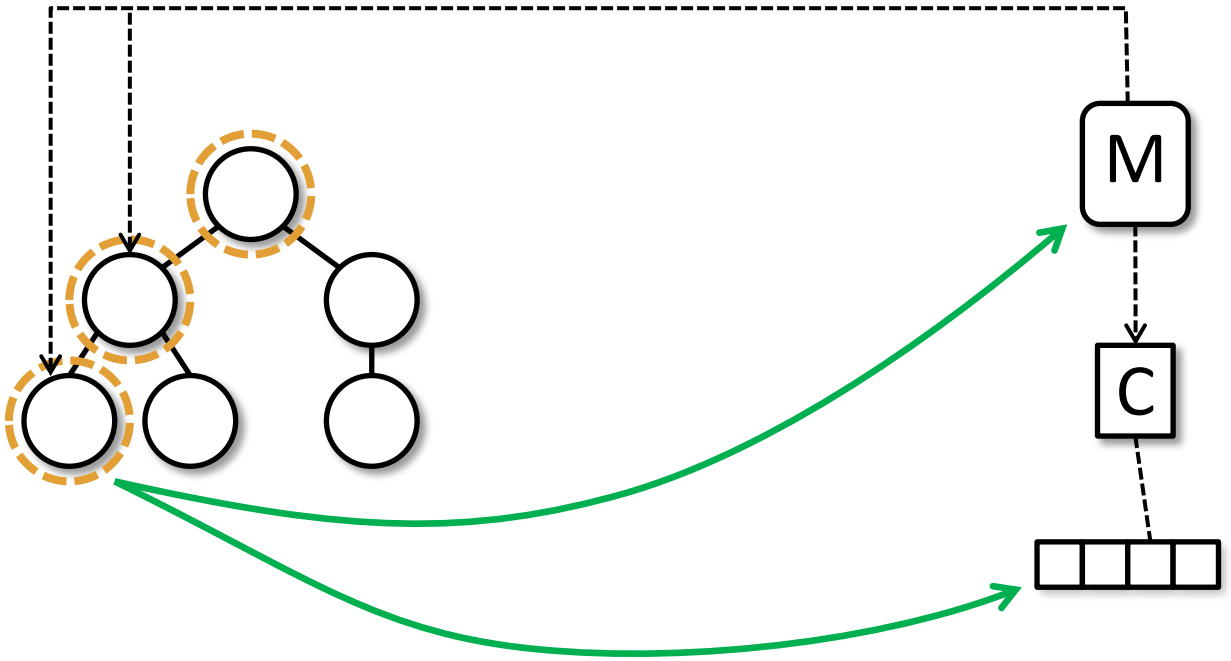
Create instructions that draw the current leaf node



# Building an incremental Render Cache

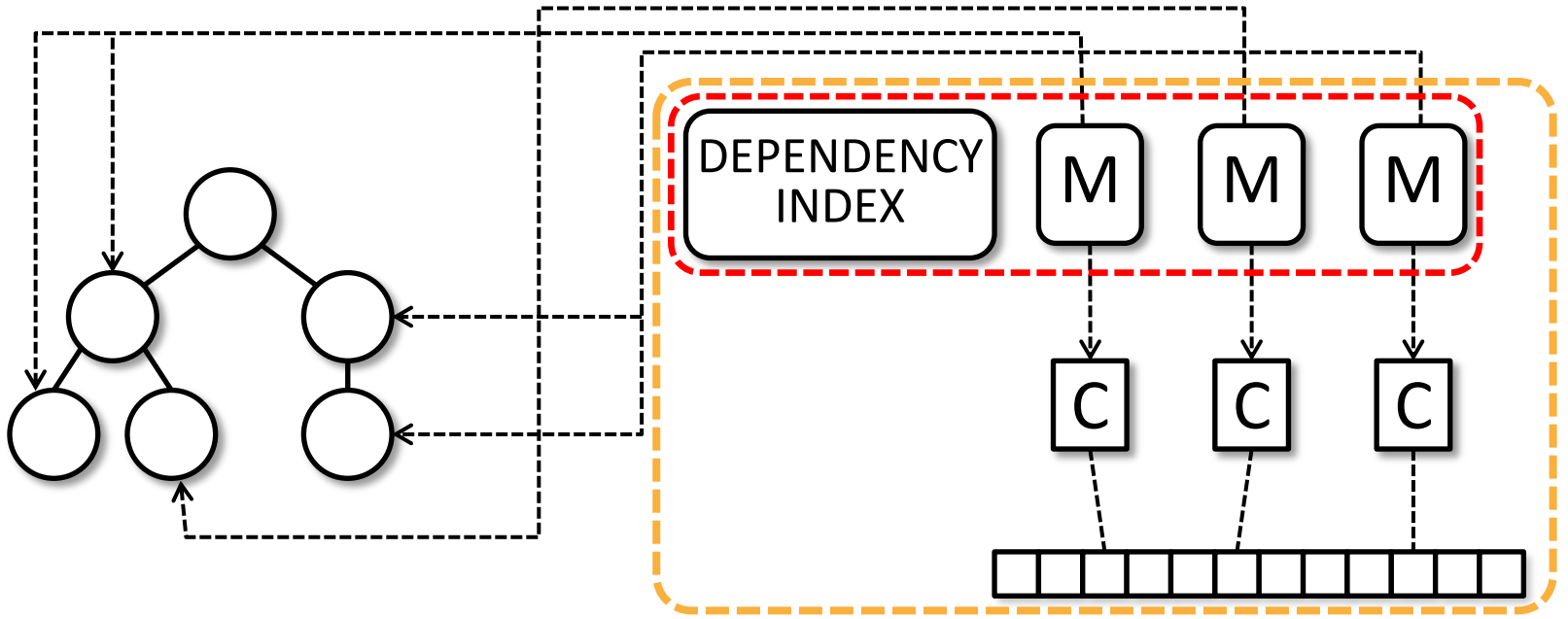
## Create Dependency Metadata entry for cache entry

- Based on type, this entry knows how to update the cache entry using remembered scenegraph nodes



# Building an incremental Render Cache

Create Dependency Index for fast queries of cache entries affected by change



# Solid Foundation for Optimizations

## For static scenes

- State Sorting
- Removal of redundant instructions
- „Super Instructions“
- Generalized Draw Sorting

## For dynamic scenes

- Parallel Cache Update
- Memoized Transformation Matrices

# Evaluation: Worst case

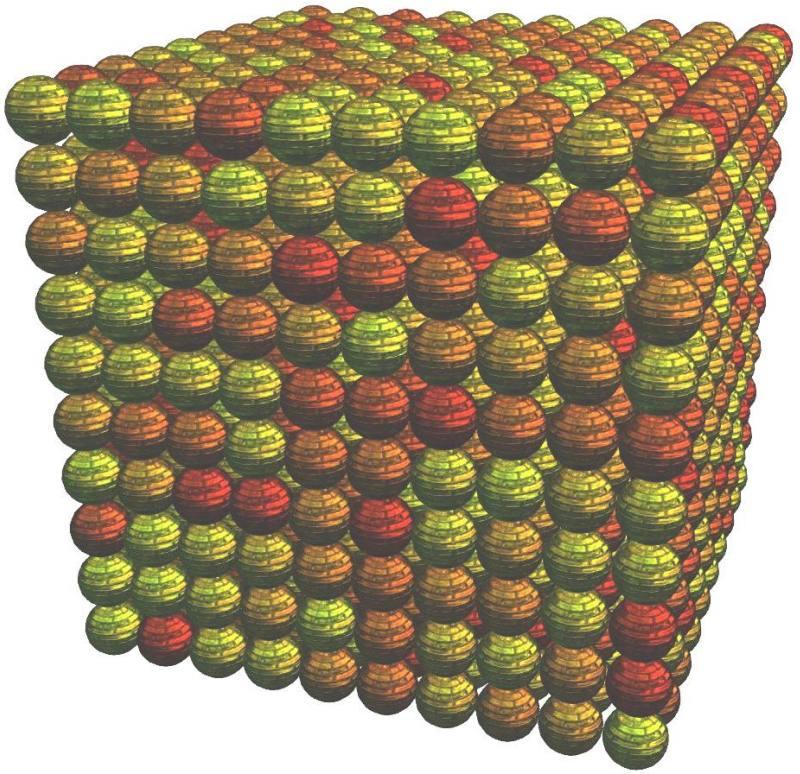
## Simulate worst case

- Distinct buffers, distinct draw calls
- Different shaders, materials etc.

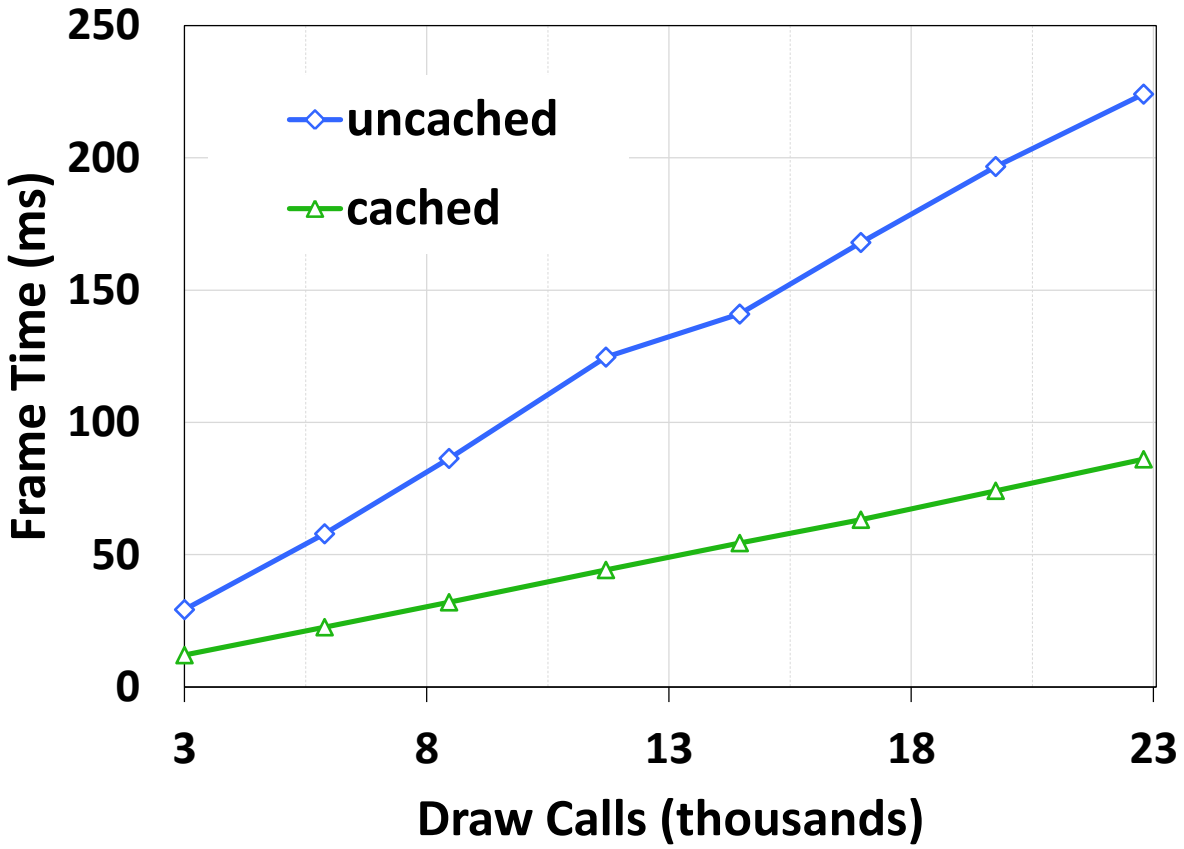
## Many, many draw calls

- Draw call reduction not sufficient (culling)
- Everything is visible
- Huge dependency graph

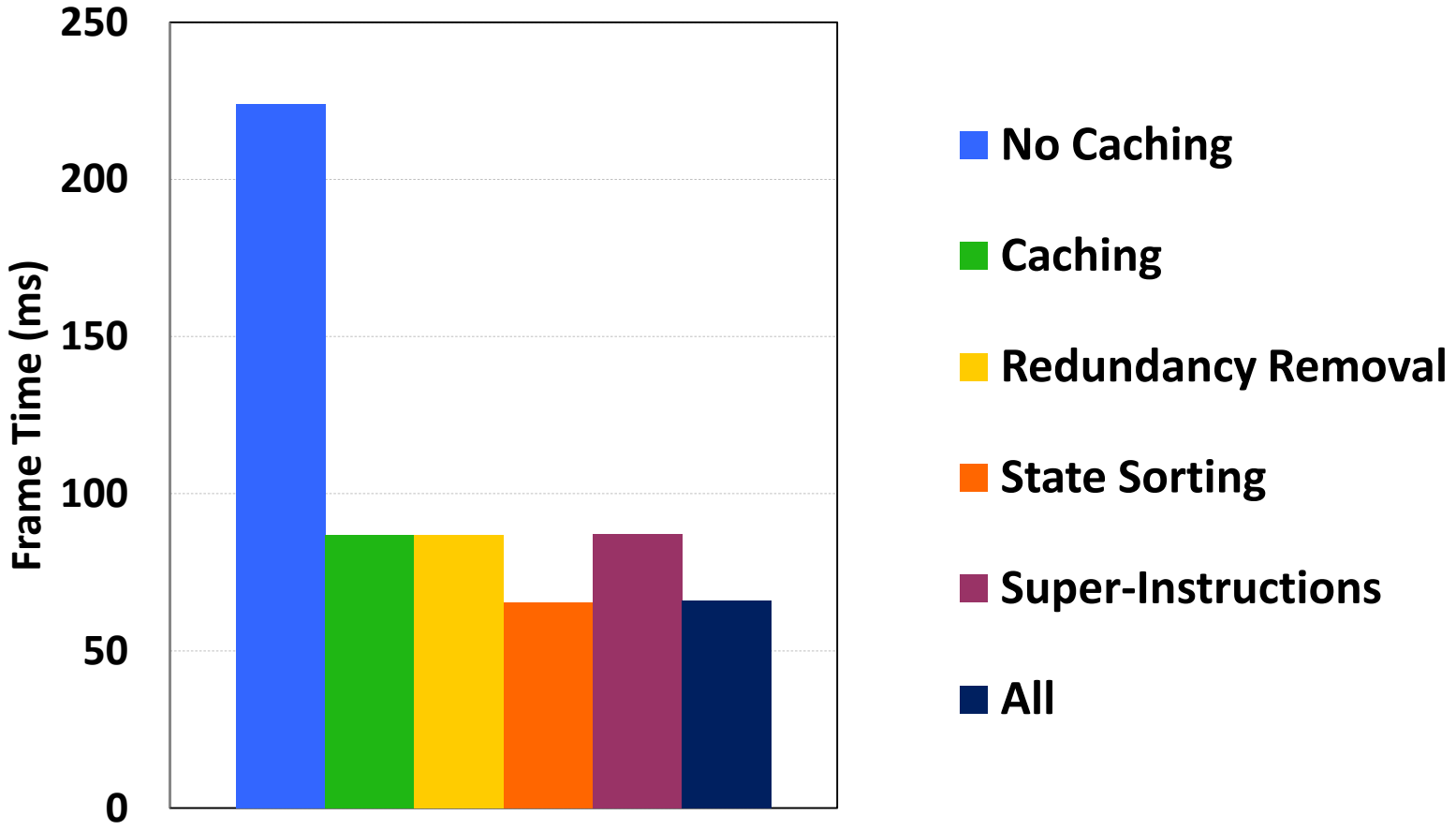
## GPU load static (poly counts)



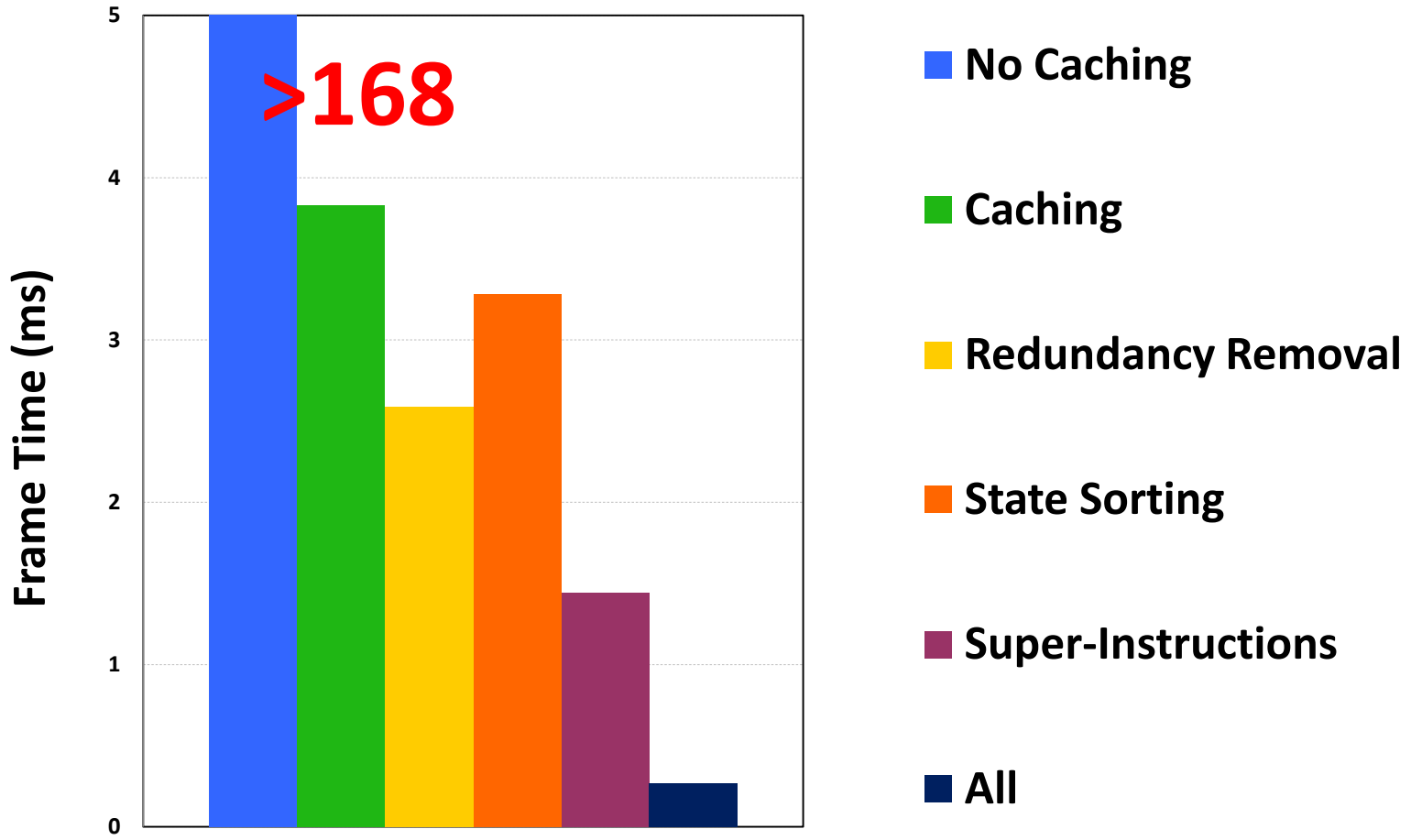
# Static Scenes



# Optimizations: Factor 2.5

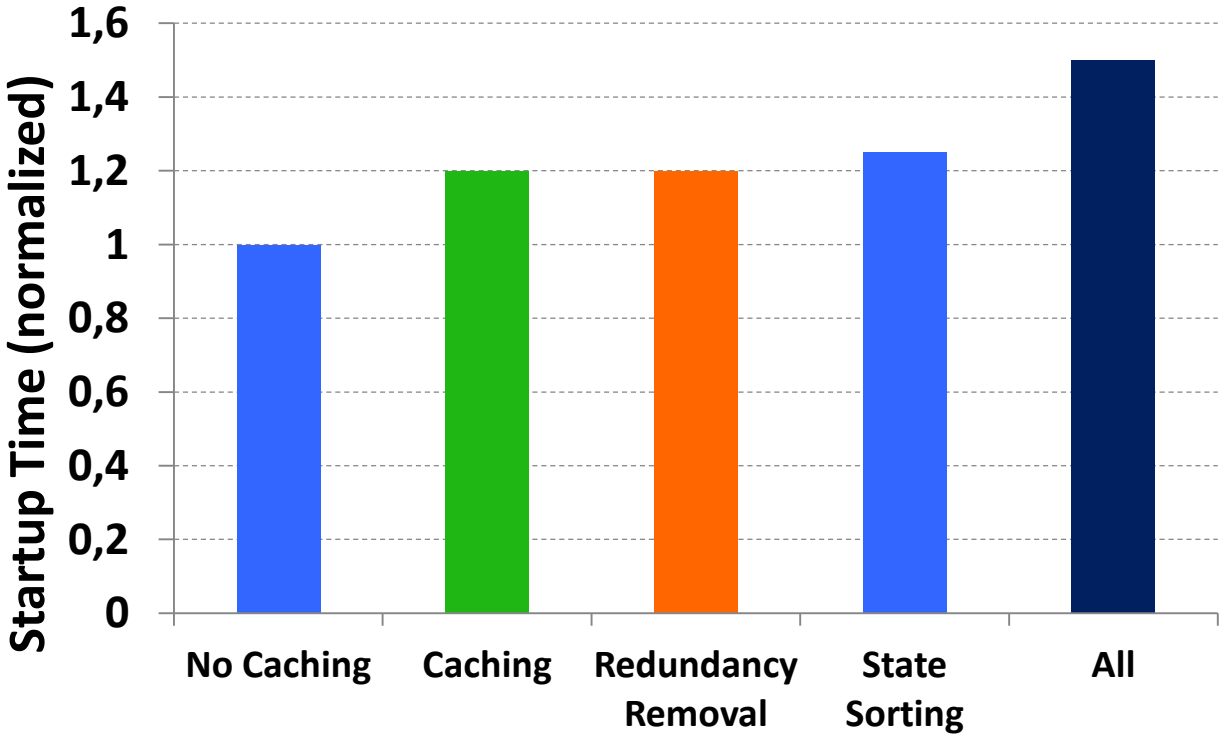


# CPU Optimizations: Huge improvement

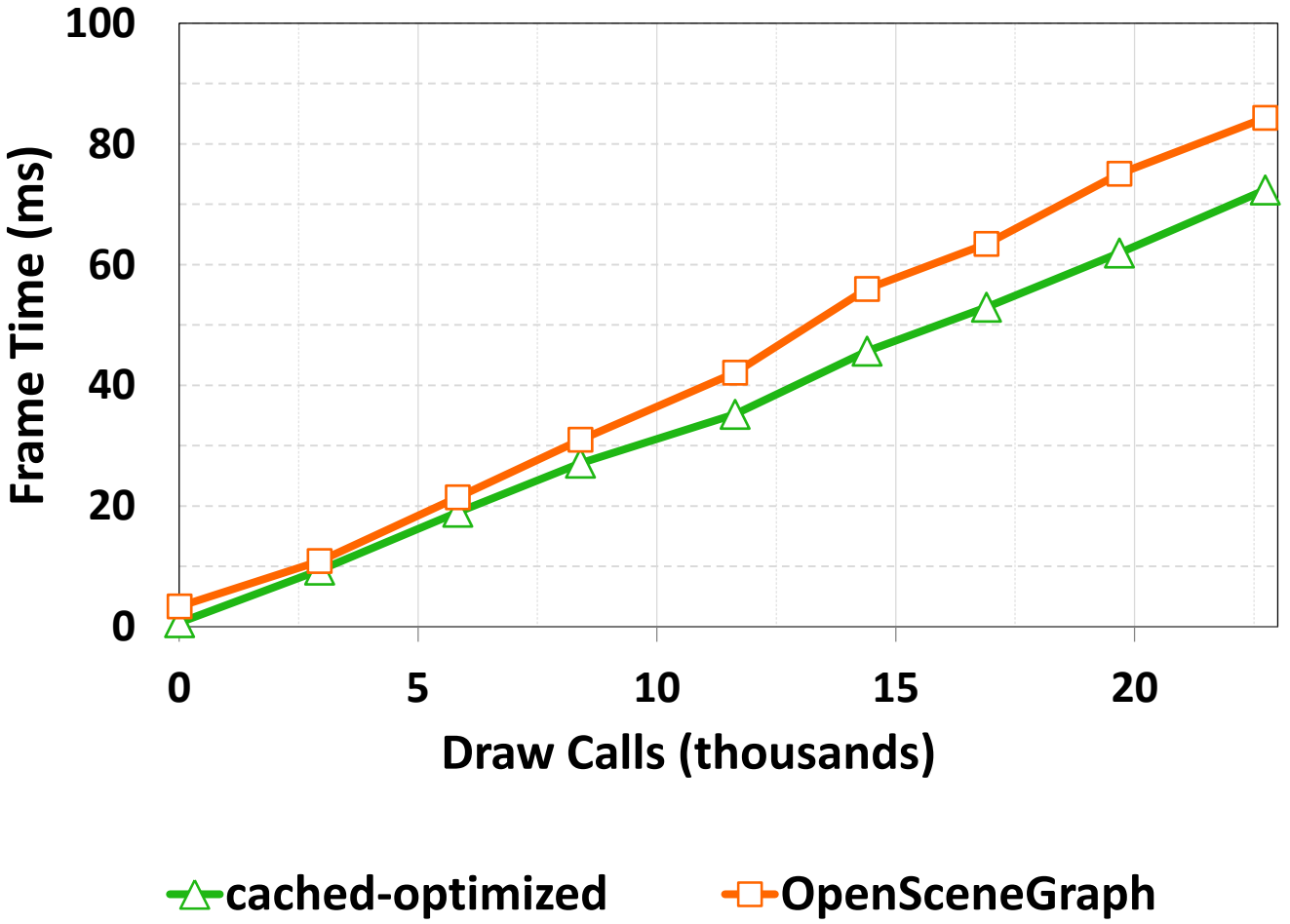


# What are the costs?

- Test scene with 22k objects, 224MB memory, 669MB graphics memory
- Additional 3MB main memory (dependencies) + additional 3MB graphics memory for caching (buffers)



# OpenSceneGraph Comparison



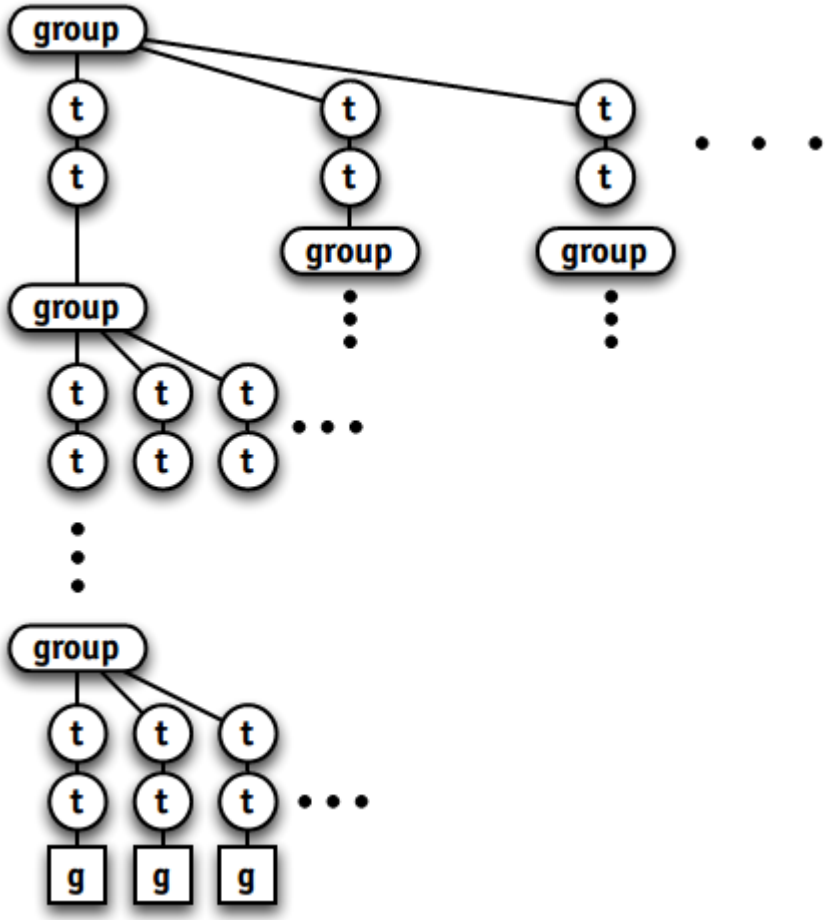
# Dynamic Scene Setup

## Octree structure

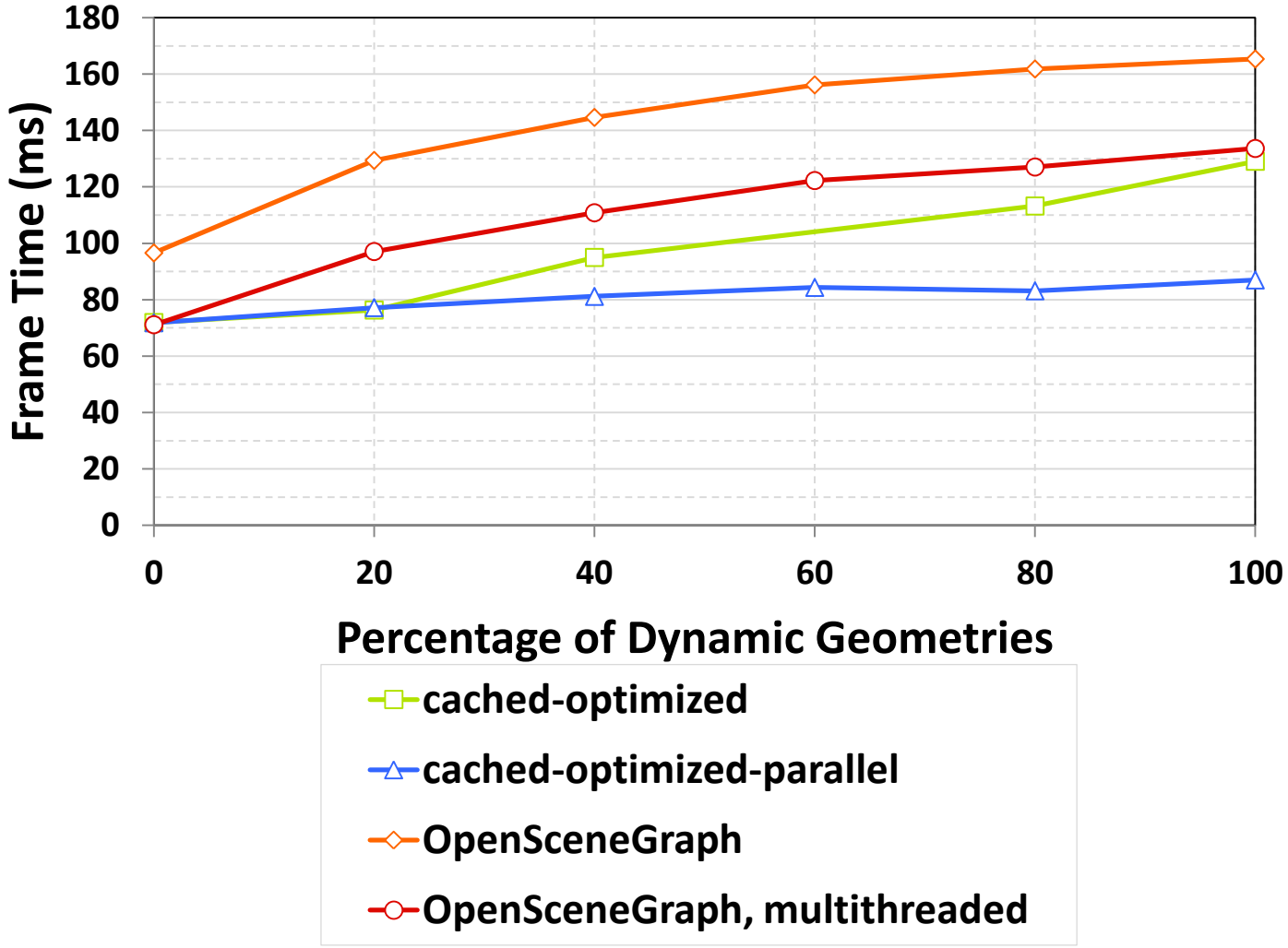
- 2  trafo nodes each level
- depth: 5, some leafs empty

## Percentage of dynamic objects

- randomized, some trafos dynamic
- varying from 0 – 100 percent changing trafo nodes



# Dynamic Scenes



# Future Work

## Achieved: efficient rendering of high level scene graph

## Structural scene graph changes

- add/remove/change arbitrary nodes
- caches need to built from scratch for this type
- improve/generalize incremental model

## Improved runtime system

- optimization at runtime / on demand
- automatic placement of render caches

# Thanks to…

**…my colleagues at the VRVis Research Center, especially:**

- Georg Haaser
- Michael Schwärzler
- Christian Luksch

# Thank you for your attention!

Please visit us at

**http://www.VRVis.at/**

vr|vis