High-Performance Graphics 2017
Los Angeles | July 28–30, 2017

TIMELINE SCHEDULING FOR OUT-OF-CORE RAY BATCHING

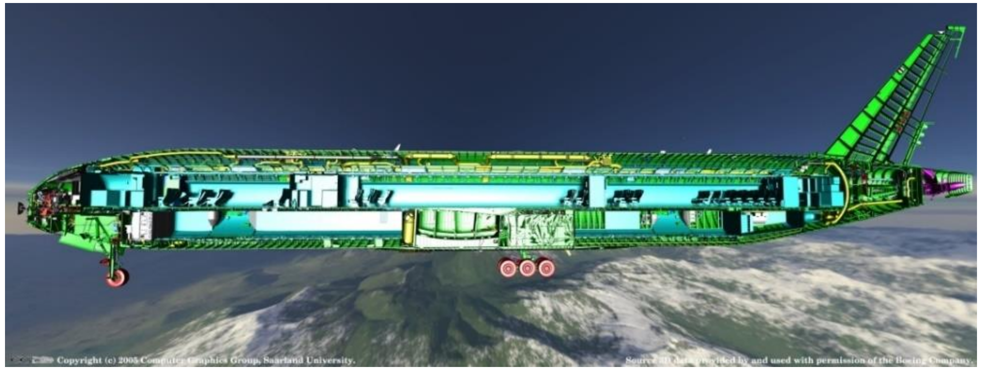
Myungbae Son Sung-Eui Yoon

SGVR Lab
KAIST

KAIST School of Computing

Our Scenario

- Complex scenes
  - Out-of-core model: Too big data!
  - Cannot be stored in main / GPU memory

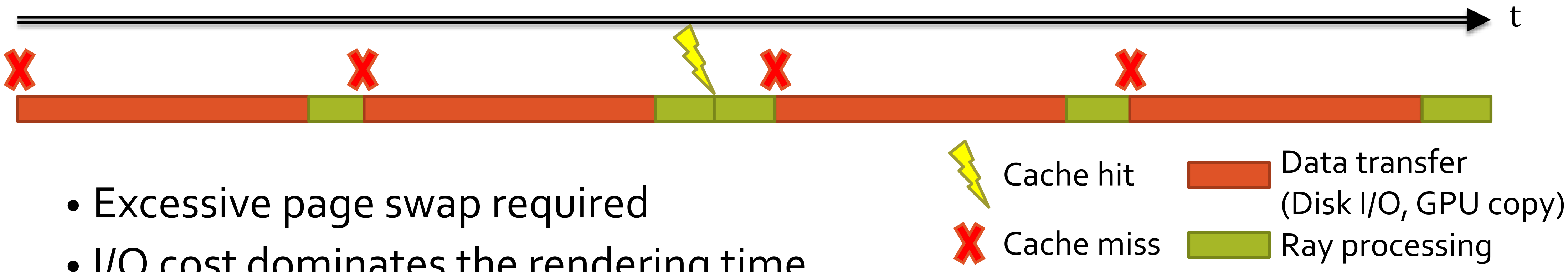


Boeing 777, 366 M tri. (20 GB)

- Complex device configurations
  - Distributed memory cluster system
  - Client-assisted remote rendering
  - Renderfarm of heterogeneous devices

Challenges

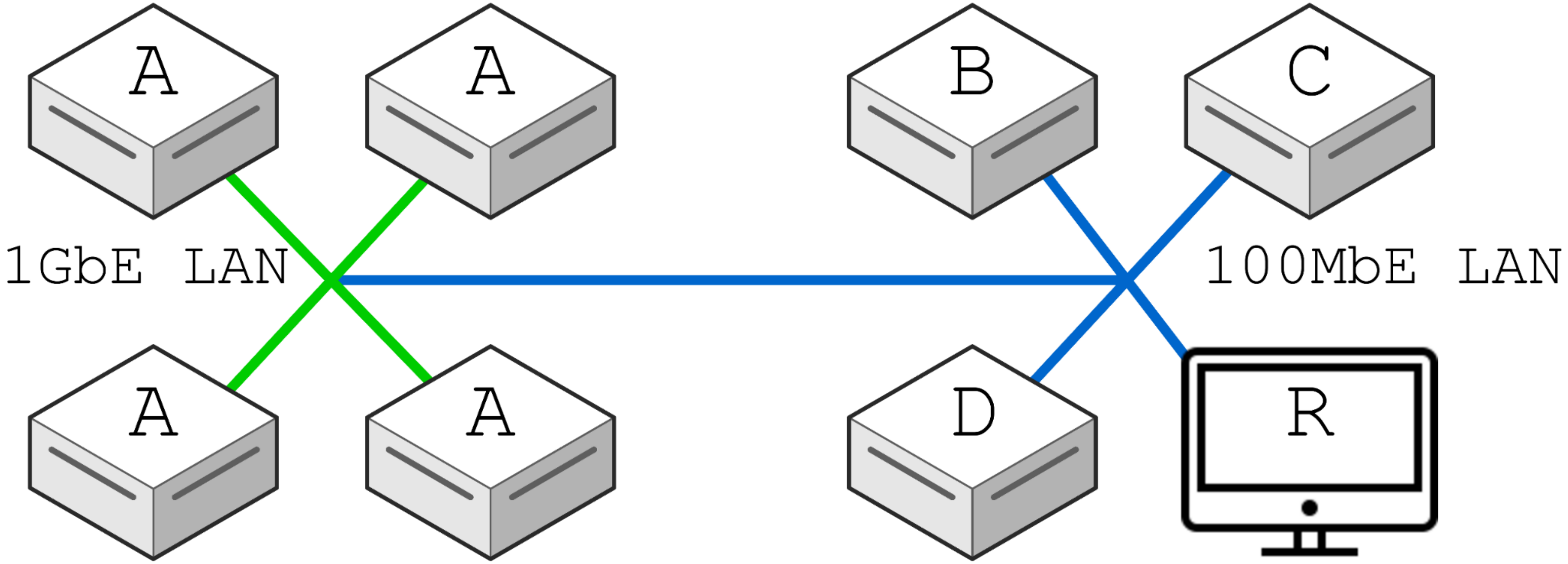
- Massively complex scene
  - Over **96%** of runtime is spent on I/O in naïve BDPT (Boeing777)



  - Excessive page swap required
  - I/O cost dominates the rendering time
- Global Illumination with incoherent rays
  - Efficient ray scheduling is required

Challenges

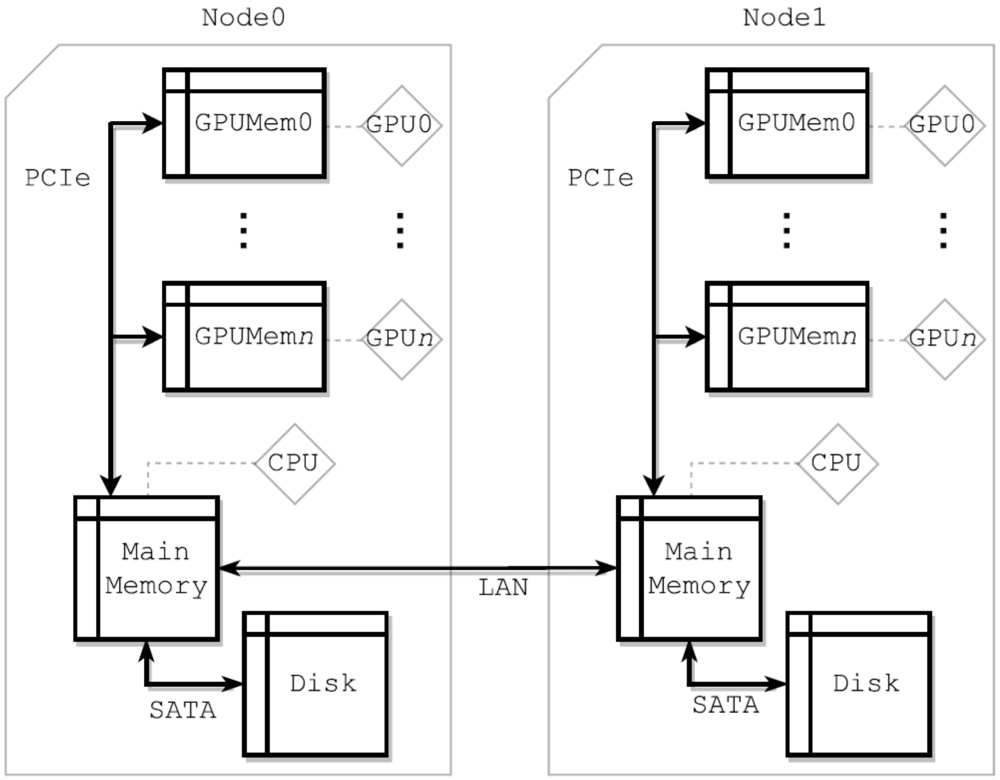
Complex and heterogenenous device configurations...



Challenges

Further down to the processor and memory hierarchy level…

- Different **processors**
- Different **memory channels**
- Different **nodes** and **network**



Goal & Contributions

Design a scheduler for global illumination

- Processes massive models
- Supports variety of computing environments
  - Complex and heterogeneous device configurations

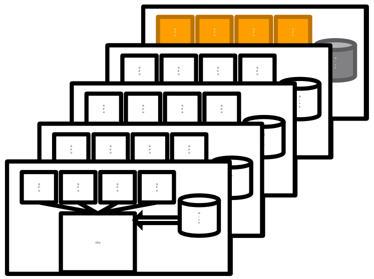
Our contributions

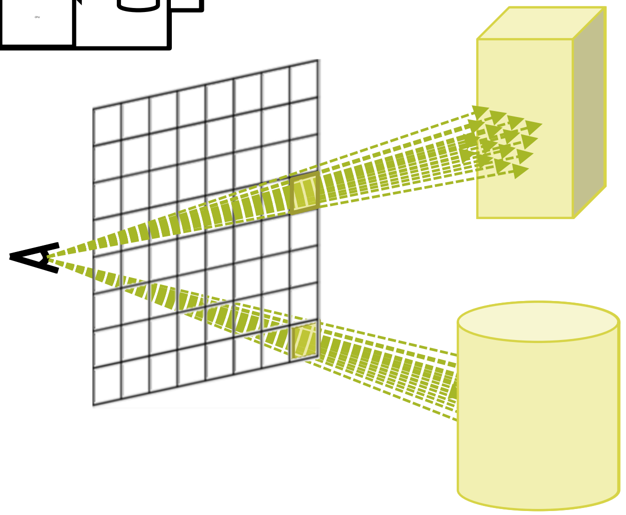
- A modeling technique: device configurations and jobs
- A scheduling algorithm: Greedy Makespan Balancing (GMB)
- An adaptation to path tracer

RELATED WORK

Ray Batching

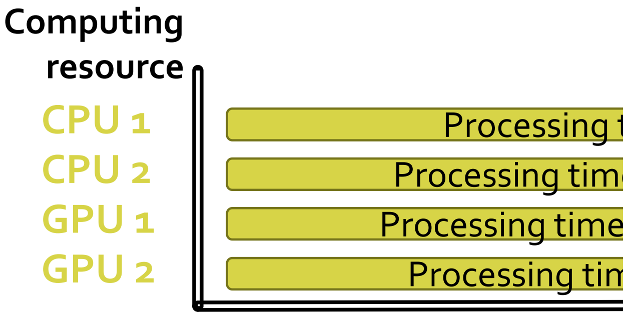
- Ray segments are decomposed into workloads
  - Cost-benefit function [Pharr et al. 1997]
  - Hybrid priority-based optimization [Budge et al. 2009]
  - Cache-oblivious reordering [Moon et al. 2010]
  - Distributed-memory cluster techniques [Navratil et al. 2014]
- Cache is considered and utilized efficiently
- Limitations of prior work
  - Assumes no complex memory hierarchy
  - Hard to scale on multiple nodes
  - No support for heterogeneous devices

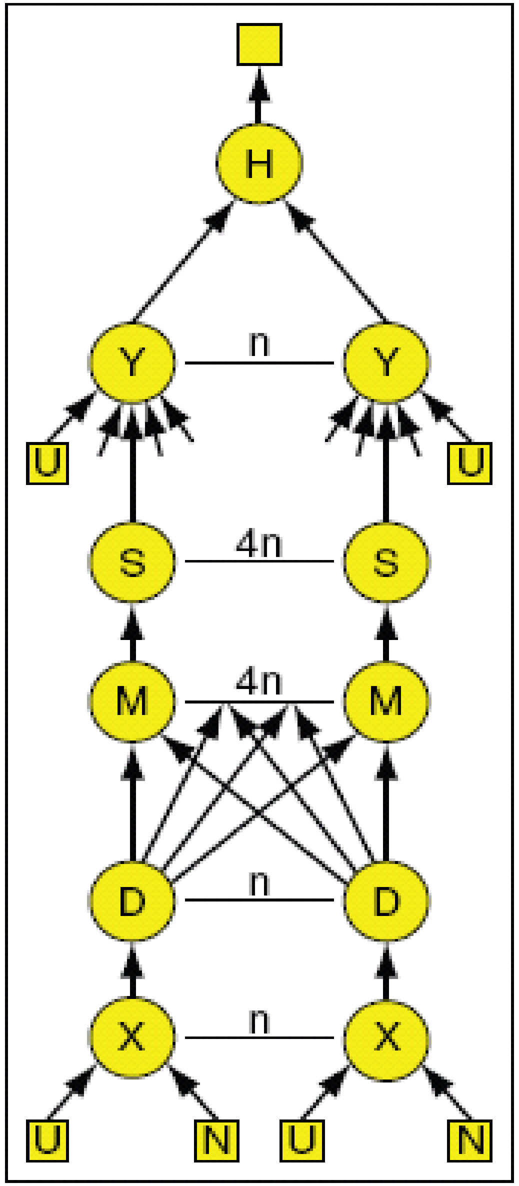




Scheduling & Specification

- General task specification & scheduling
  - LP-based solver[Kim et al. 2012]
  - Dryad[Isard et al. 2007]
  - HEFT, CPOP[Topcuoglu et al. 2002]





- Great scaling on multi-node/task complexity
- Limitations
  - Inefficiencies on dynamic workload
  - Either cache or bandwidth is not considered

OUR APPROACH

Our Approach

- Formulation technique for MC ray tracing jobs
  Device Connectivity Graph (DCG) and Timing Model
- Timeline scheduling and Greedy Makespan Balancing algorithm
  Simple, iterative algorithm that considers utilization and latency hiding
- Adaptation to actual renderer framework
  Out-of-core path tracer

Our Approach

- **Formulation technique for MC ray tracing jobs**
  Device Connectivity Graph (DCG) and Timing Model

- Timeline scheduling and Greedy Makespan Balancing algorithm
  Simple, iterative algorithm that considers utilization and latency hiding

- Adaptation to actual renderer framework
  Out-of-core path tracer

Formulation: Device Connectivity Graph

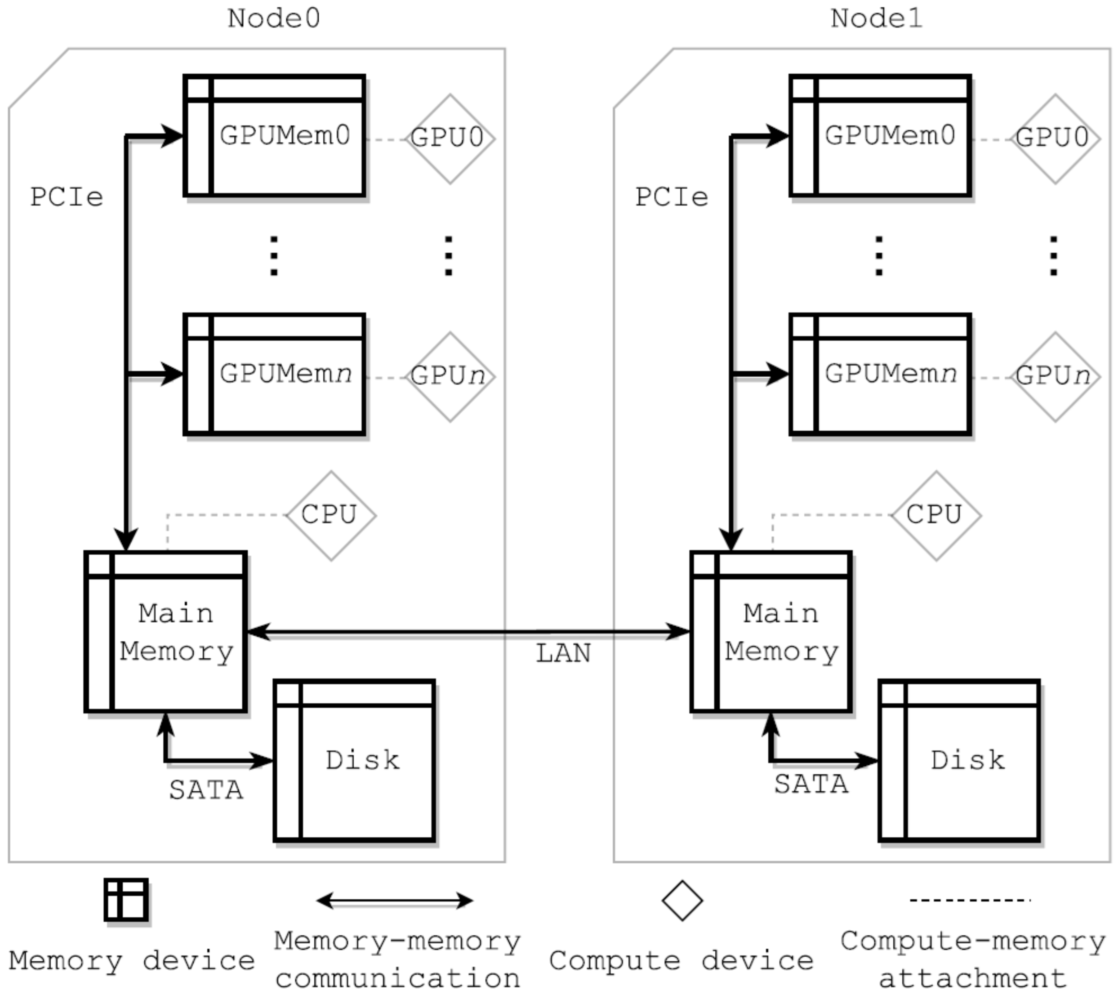
- Graph of memory devices
  - Memory

    Disk storage, RAM, GMEM

  - Connections (Channels)

    PCIe (RAM ↔ GMEM)
    SATA (Disk ↔ RAM)
    LAN (RAM ↔ RAM)

    …
- Stores bandwidth information



KAIST School of Computing

Formulation: Timing Model

- Assume simple yet efficient linear model on time
  - Job execution

$$T_{EXEC}(d, j, W) = \begin{cases} 0, & if\, W = \emptyset \\ T_{SETUP}(d, j) \\ \quad + T_{RATE}(d, j) \cdot (|w_1|, |w_2|, \ldots), & otherwise \end{cases}$$

  - Data transfer

$$T_{TRANS}(d_i \rightarrow d_j, w) = T_{LAT}(d_i \rightarrow d_j) + \frac{|w|}{T_{BW}(d_i \rightarrow d_j)}$$

- Fitting each parameter ($T_{SETUP}, T_{RATE}, T_{LAT}, T_{BW}$)
  - Use least squares method on test run

Our Approach

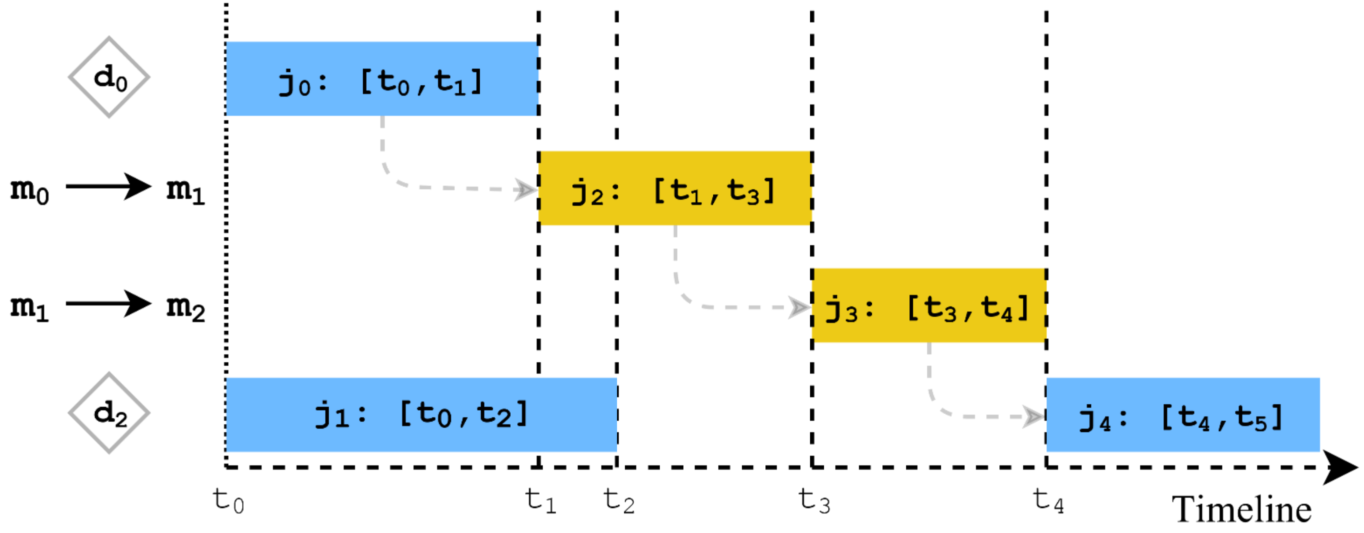
- Formulation technique for MC ray tracing jobs
  Device Connectivity Graph (DCG) and Timing Model

- **Timeline scheduling and Greedy Makespan Balancing algorithm**
  Simple, iterative algorithm that considers utilization and latency hiding

- Adaptation to actual renderer framework
  Out-of-core path tracer

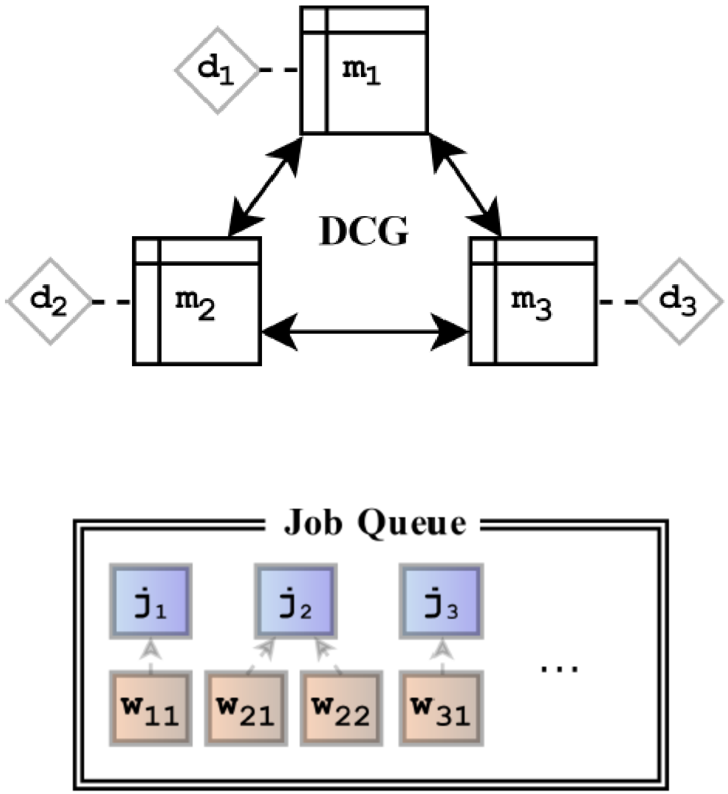
Timeline Scheduling

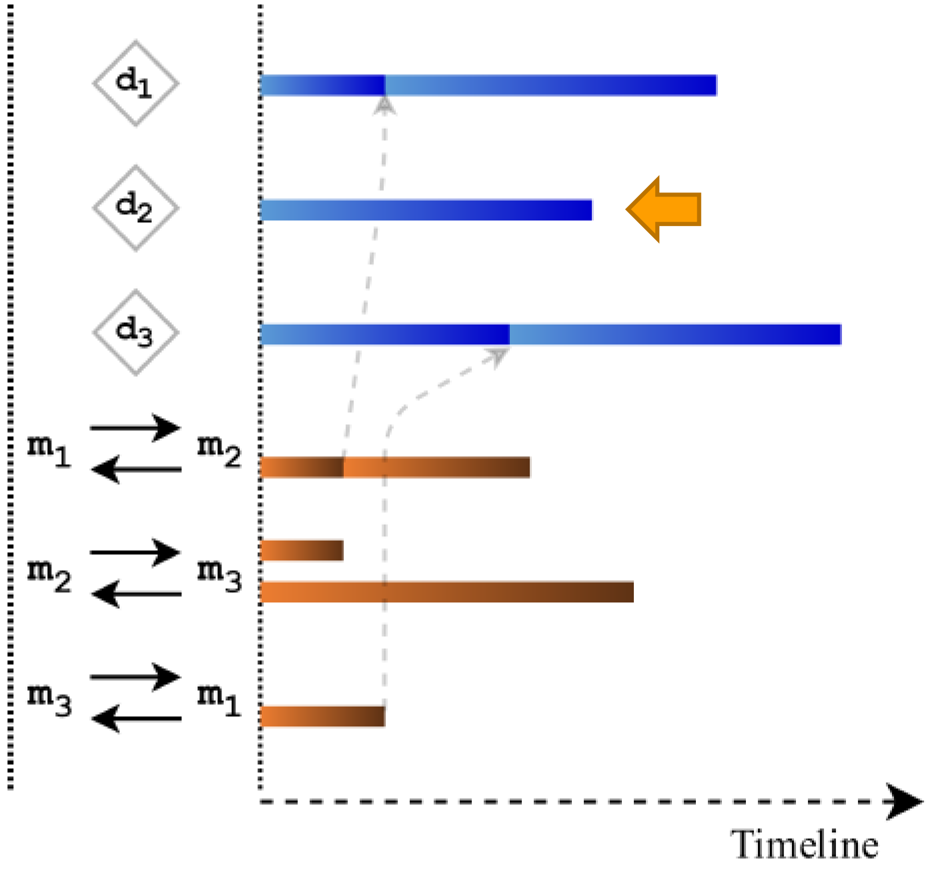
- A representation of schedule with timing constraints
  - For ◇ processors
    Executable jobs are allocated
  - For ↔ memory channels
    Data transfers are allocated
  - Dependencies
    between jobs and fetches



Def. schedule: a set of timelines that jobs and fetches are allocated

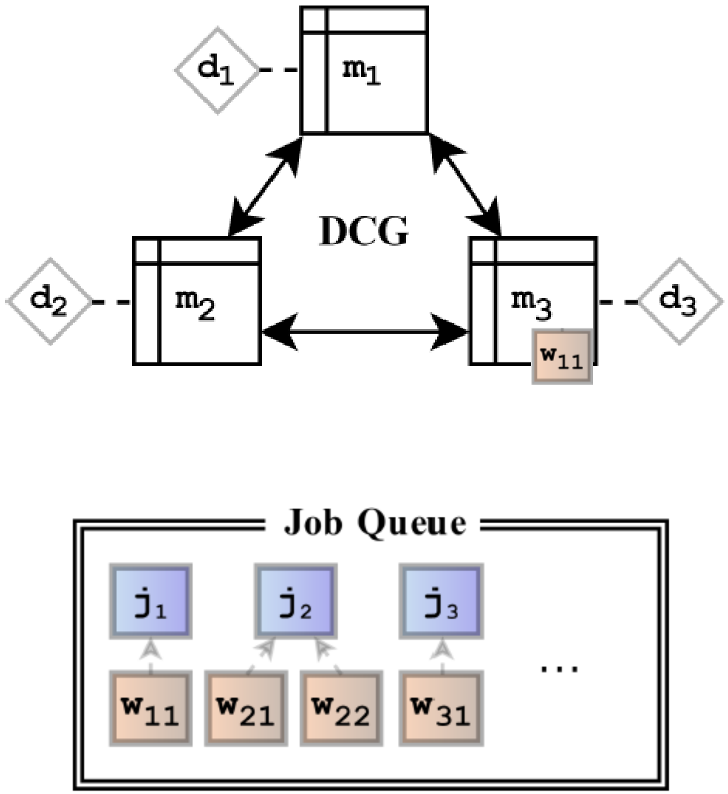
Greedy Makespan Balancing Algorithm

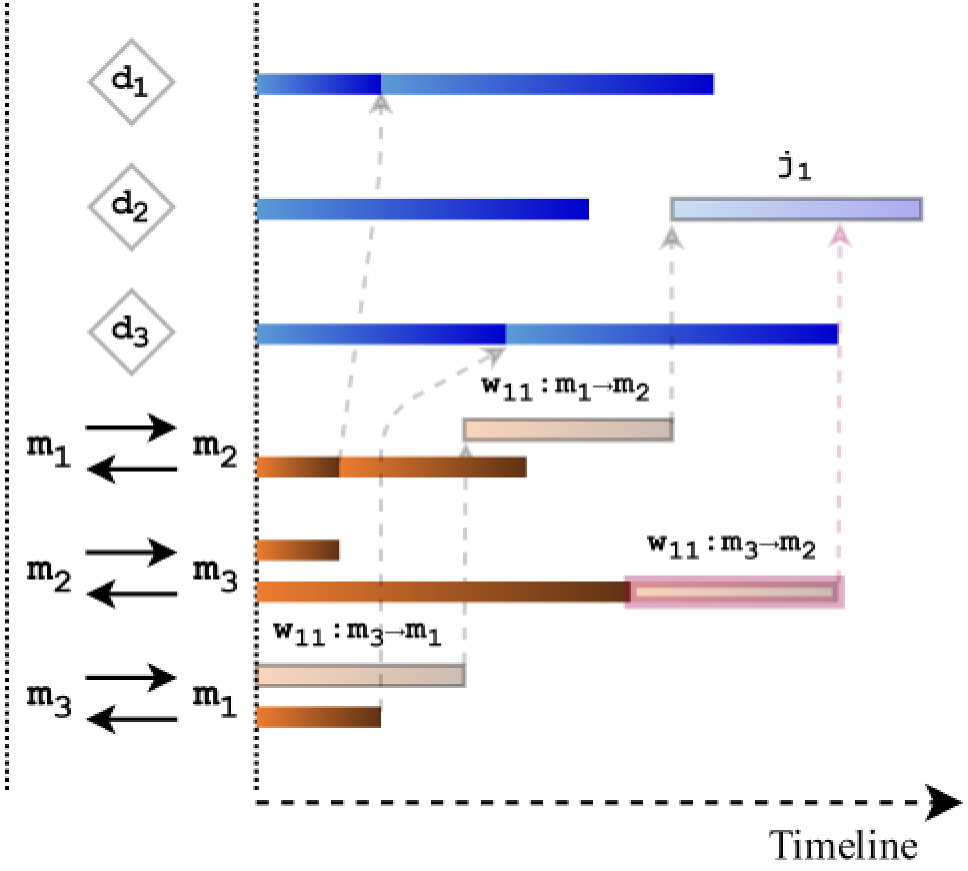




1. Choose least occupied compute device d

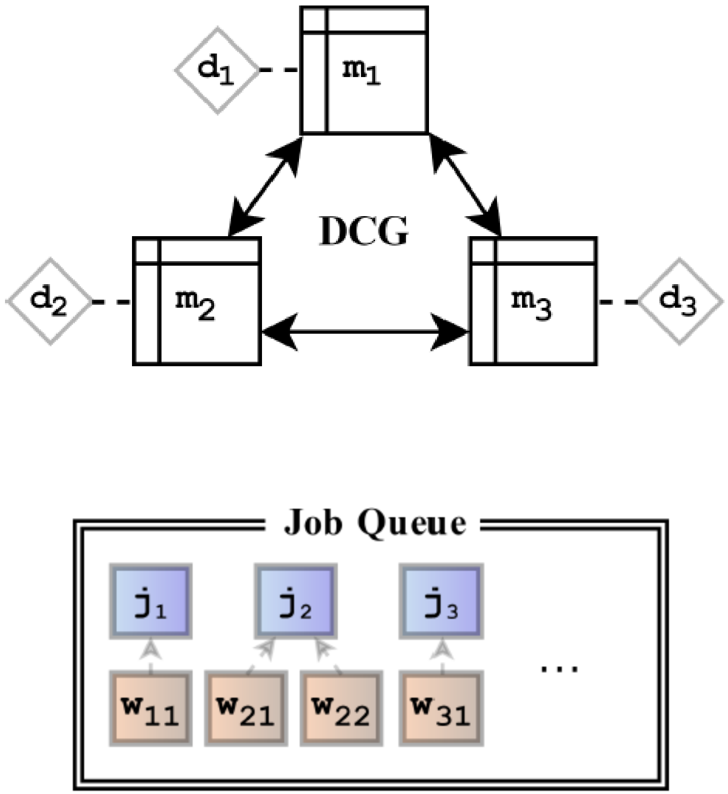
Greedy Makespan Balancing Algorithm

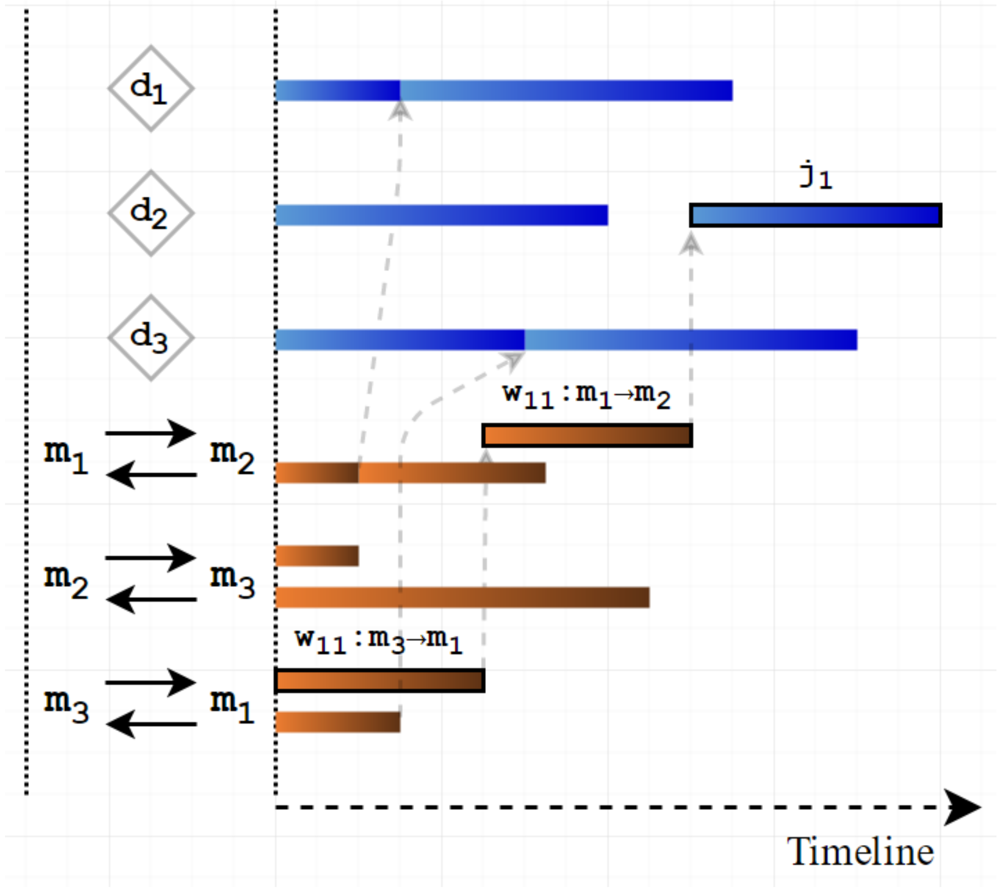




2. Find job j_i that can be run at d as soon as possible

Greedy Makespan Balancing Algorithm





4. Repeat until devices are occupied enough

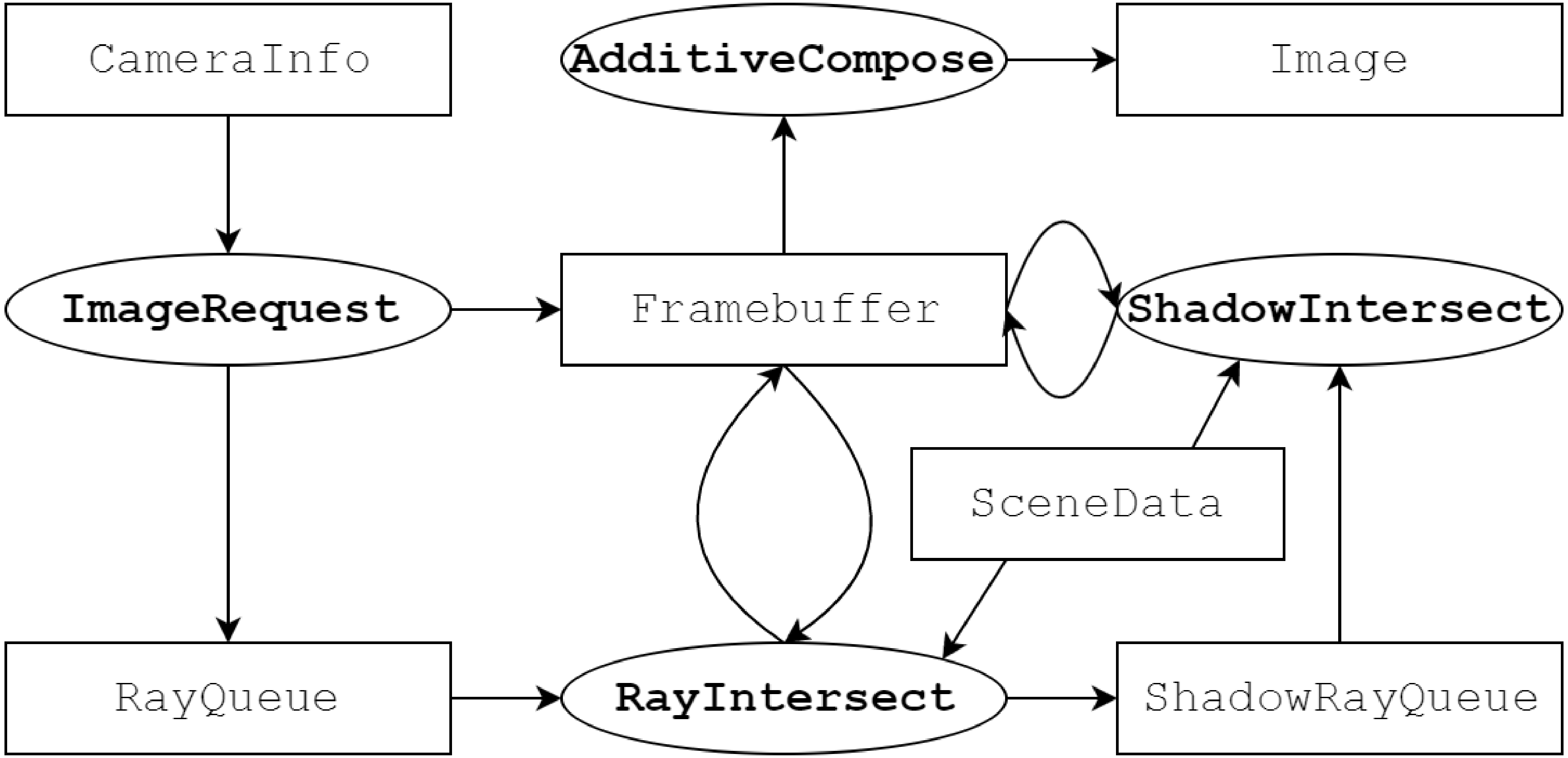
Our Approach

- Formulation technique for MC ray tracing jobs
  Device Connectivity Graph (DCG) and Timing Model

- Timeline scheduling and Greedy Makespan Balancing algorithm
  Simple, iterative algorithm that considers utilization and latency hiding

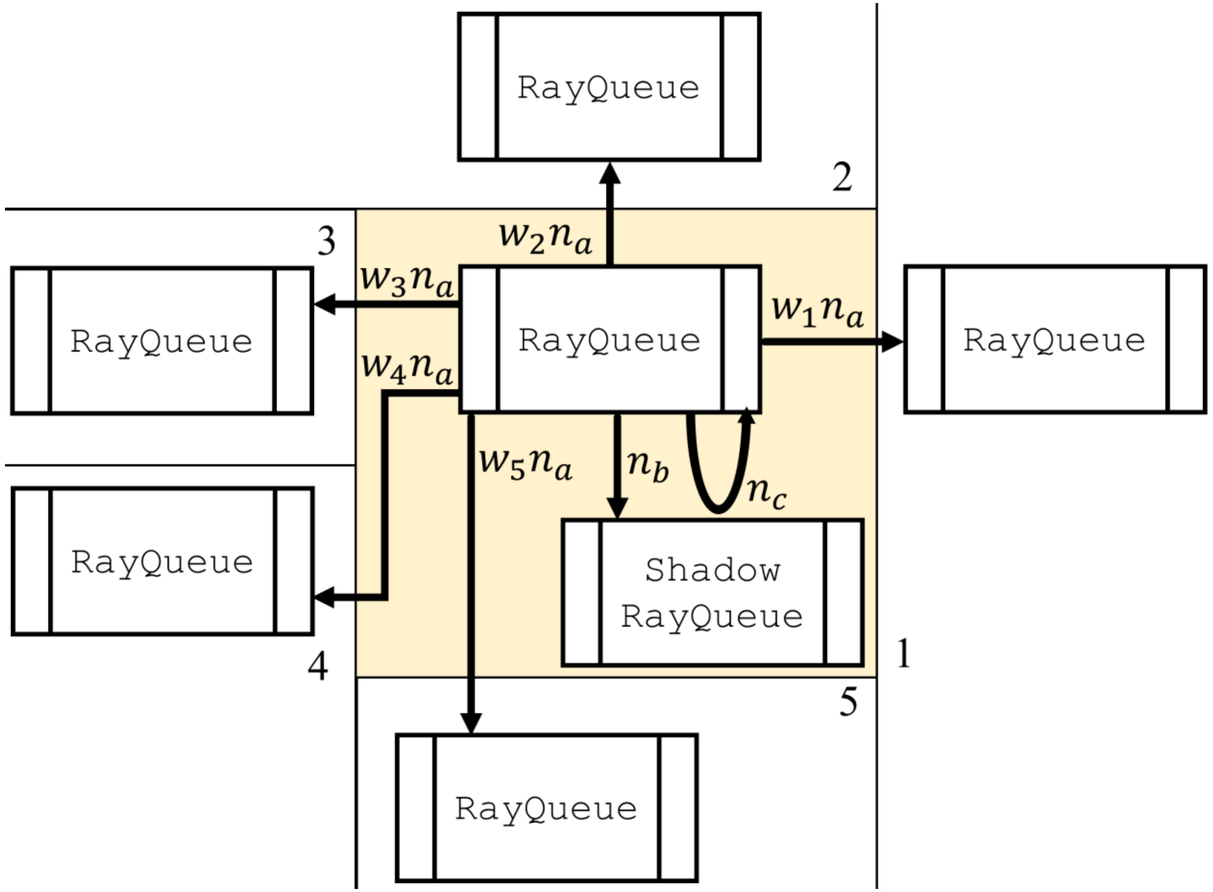
- **Adaptation to actual renderer framework**
  Out-of-core path tracer

Out-of-core Path Tracer Jobs



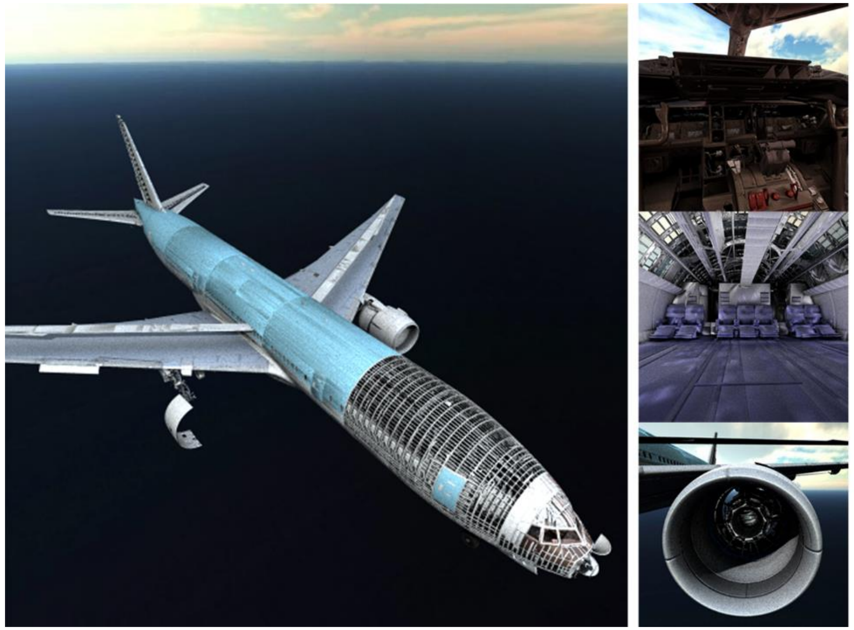
Job Prediction

- Allow more future jobs to be scheduled
  Improved quality of the schedule
- Rays are predicted to be…
  - … propagated to next cell
  - … bounced into secondary ray
  - … terminated with shadow ray
- Expect how much future jobs get spawned



RESULTS

Benchmark scene



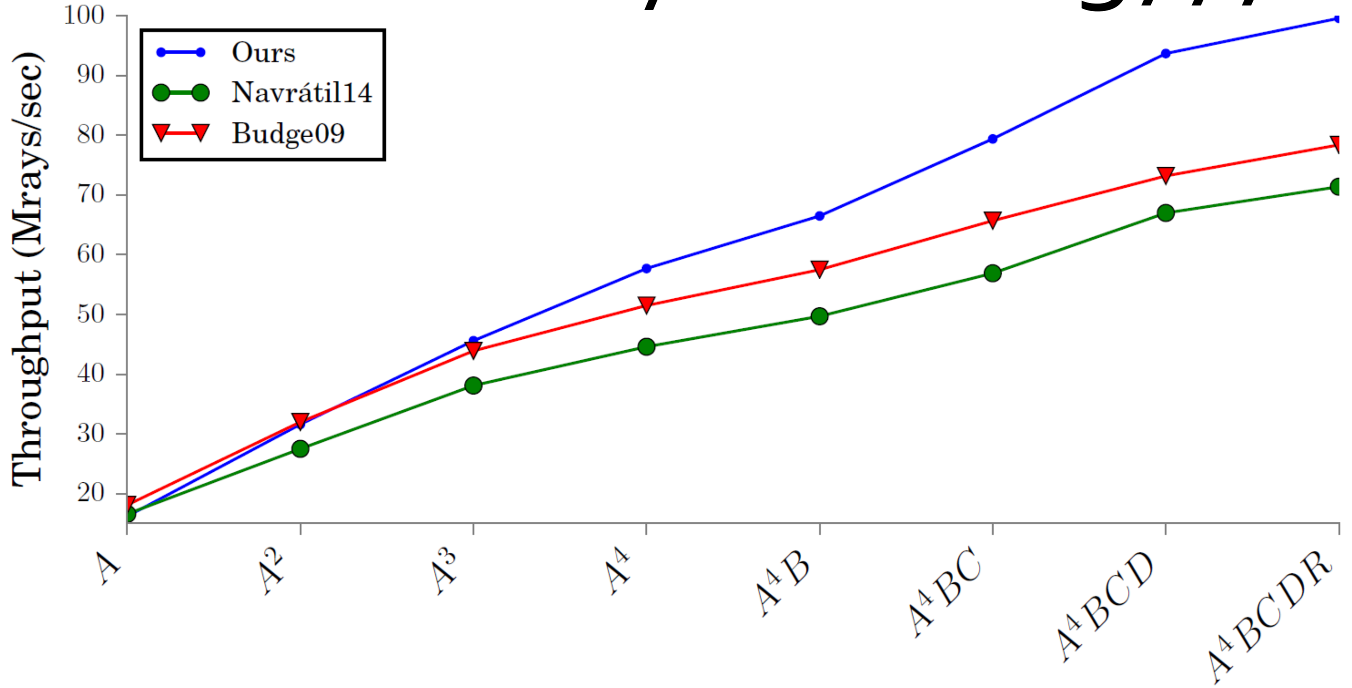
Boeing777 (26.5GB, 496M tri, 5.2sec/img)

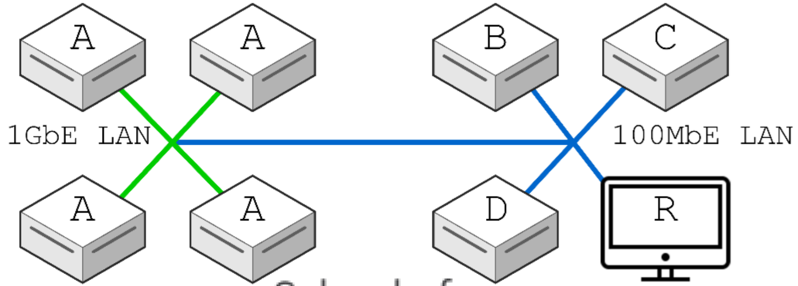
SponzaMuseum (12.3GB, 245M tri, 34.8 sec/img)

$(800 \times 800 \times 32spp \times 60frames)$

- Model preparation
  - Even-sized median-split kdtree, 2^7 / 2^6 subdivision, respectively

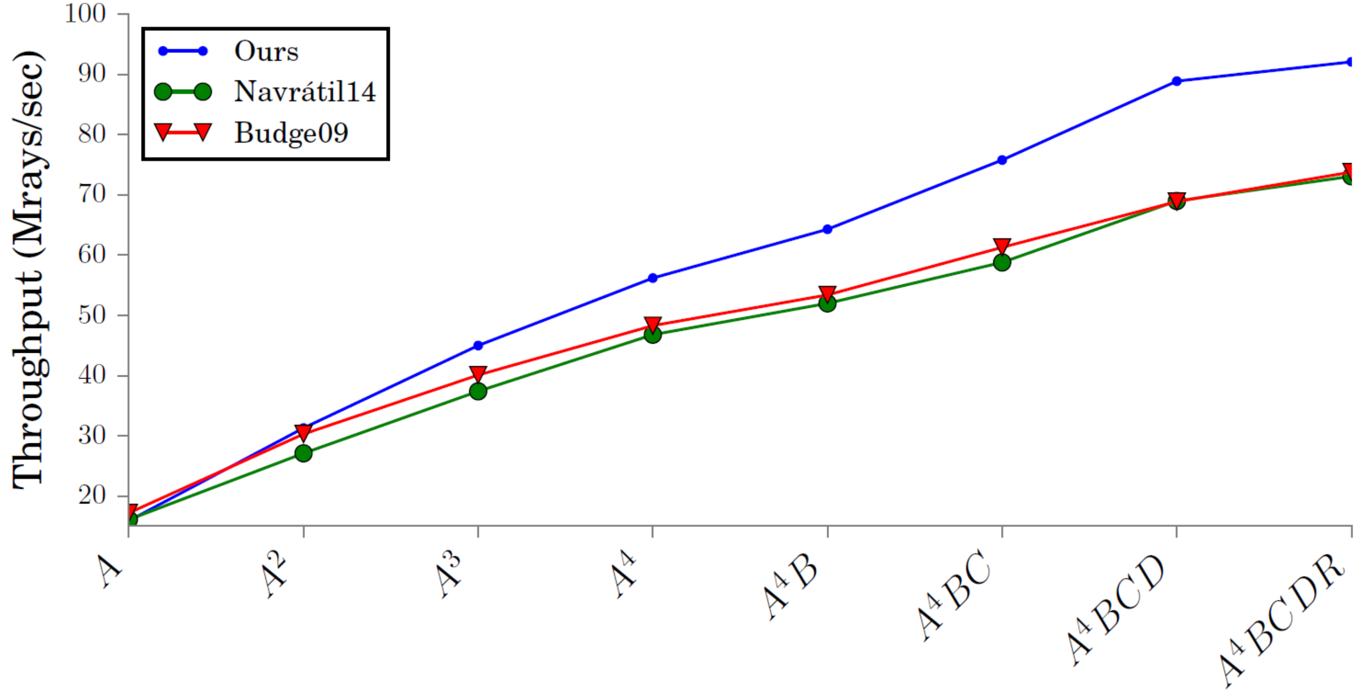
Horizontal Scalability – Boeing777

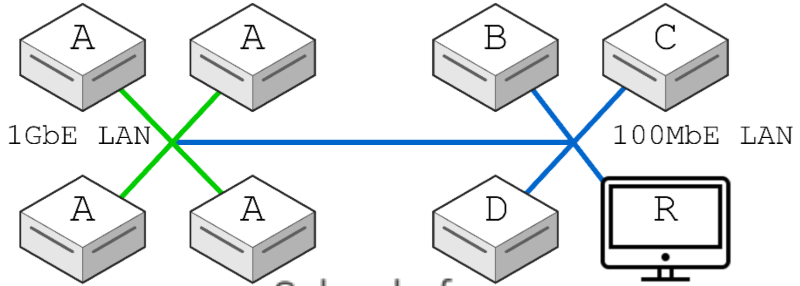




Type	CPU	Main memory	GPU Memory	GPU	Note
A	i7-4770K 3.5GHz octa-core	DDR3 8GB	6GB	GTX Titan	1GbE LAN, 4 nodes
B	i7-4790K 4GHz octa-core	DDR3 8GB	6GB	GTX Titan	
C	Xeon E5-2690 2.9GHz 16-core	DDR3 8GB	6GB	GTX Titan	
D	Xeon E5-2690 2.6GHz 16-core	DDR3 8GB	6GB	GTX Titan X	
R	i7-3770k 3.5GHz quad-core	DDR3 8GB	4GB	GTX980	

Horizontal Scalability – SponzaMuseum

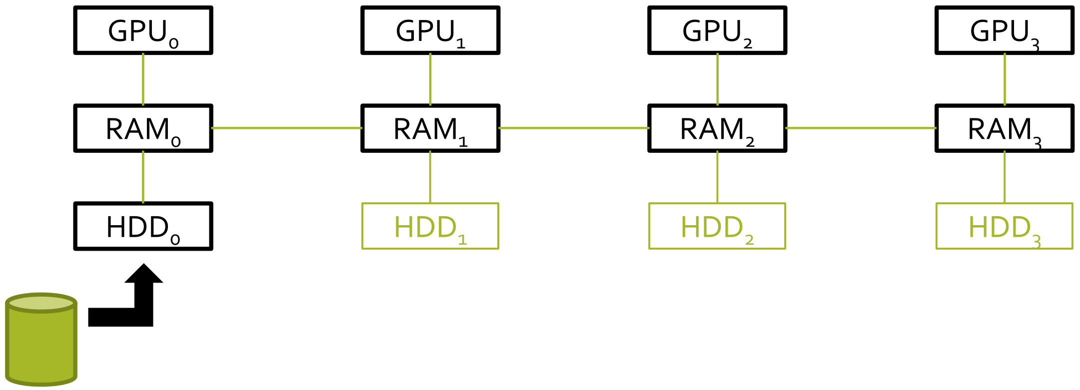




Type	CPU	Main memory	GPU Memory	GPU	Note
A	i7-4770K 3.5GHz octa-core	DDR3 8GB	6GB	GTX Titan	1GbE LAN, 4 nodes
B	i7-4790K 4GHz octa-core	DDR3 8GB	6GB	GTX Titan	
C	Xeon E5-2690 2.9GHz 16-core	DDR3 8GB	6GB	GTX Titan	
D	Xeon E5-2690 2.6GHz 16-core	DDR3 8GB	6GB	GTX Titan X	
R	i7-3770k 3.5GHz quad-core	DDR3 8GB	4GB	GTX980	

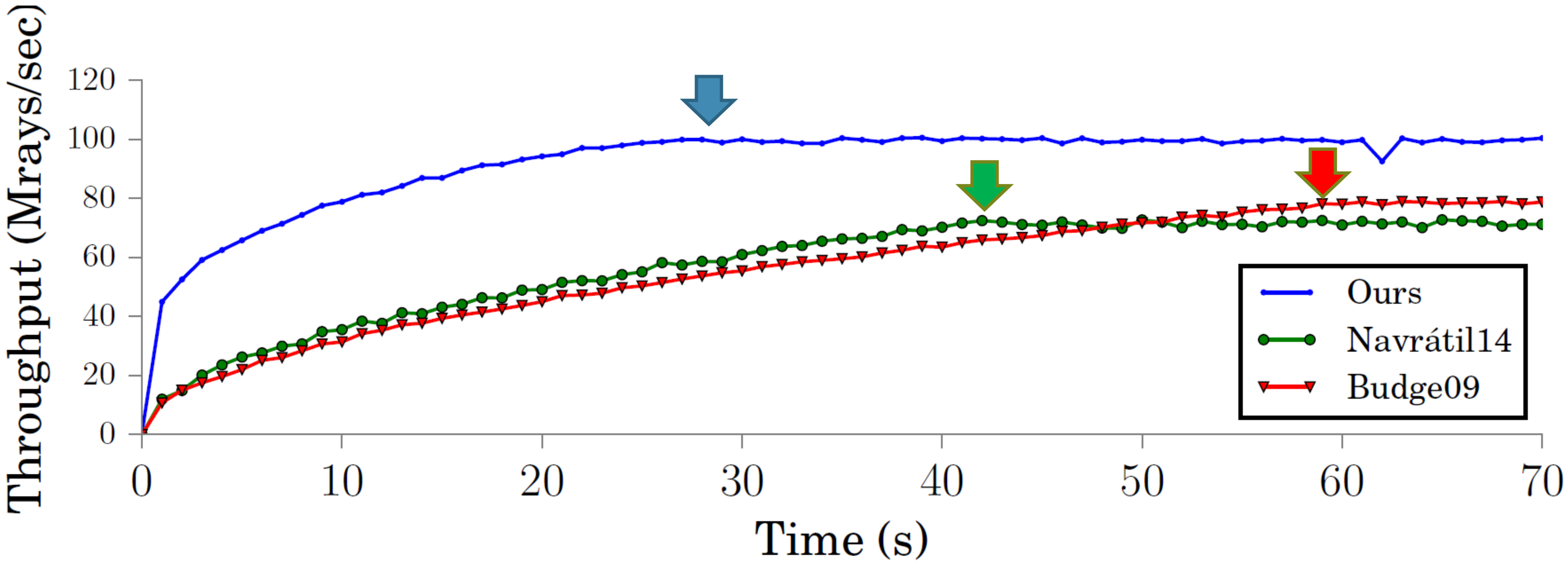
Efficiency on Data Fetching

- Central scene DB scenario



- Initially no data at slave nodes at all
- The master node gives scene data blocks on-demand

Efficiency on Data Fetching



(Boeing777)

Our method converges to peak performance much faster than previous methods

Conclusion

- Presented specification techniques for out-of-core MC ray tracing on arbitrary hardware setup
  - DCG and timing model
- Presented a timeline based scheduling algorithm
  - GMB algorithm
- Applied to the out-of-core path tracer
  - Prediction technique for future rays

Acknowledgement

- Members of KAIST SGVR Lab for discussions

- This work was supported by ICT R&D program of MSIP/IITP [R0126-17-1108].

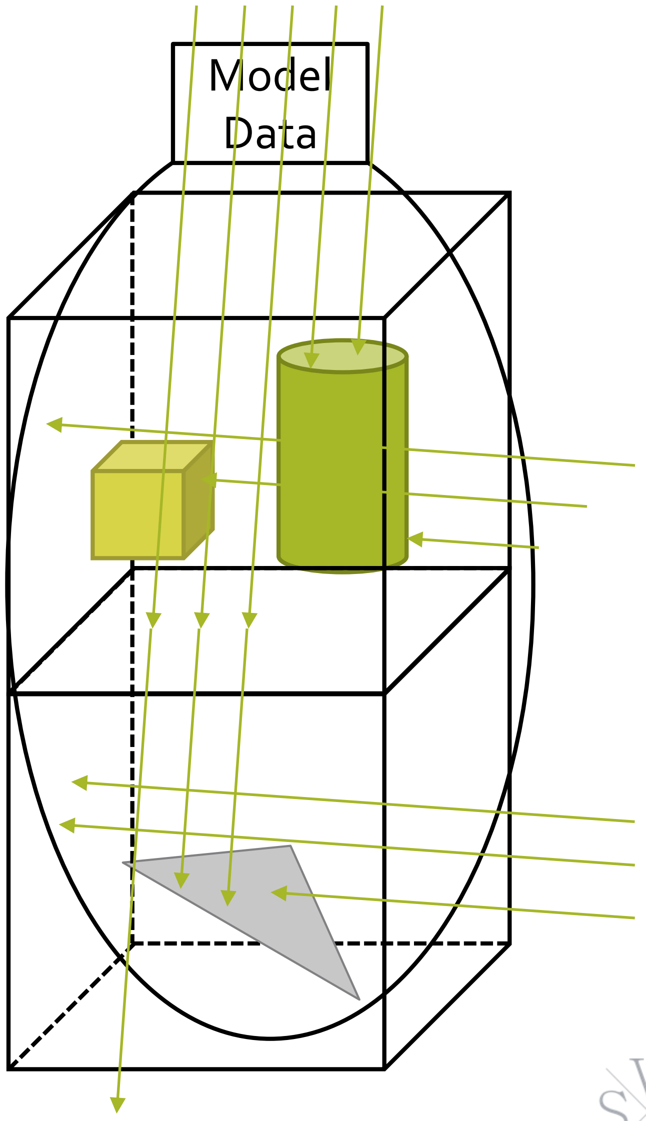
THANK YOU!

Q & A

http://sglab.kaist.ac.kr/GMB/

Ray Batching

- Pseudocode
  1. Sort rays to each model subdivision
  2. Select the subdiv. to be processed
     - Example: subdiv. queued with the highest #rays
  3. Load a subdiv. if not loaded
  4. Process related workloads to that subdiv.
- Ray segments are decomposed into workloads
  - Computational decomposition [Cleary et al. 1986]



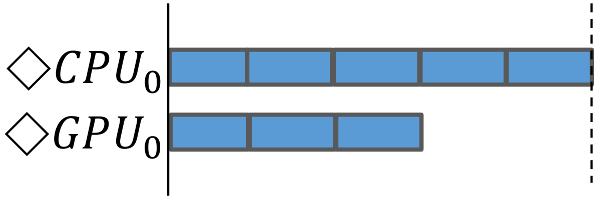
KAIST School of Computing

Formulation Techniques

- To formally specify…
  - (1) How much time to process a job
  - (2) How much time to fetch the required data

Formulation Techniques

- To formally specify…
  - (1) How much time to process a job
  - (2) How much time to fetch the required data

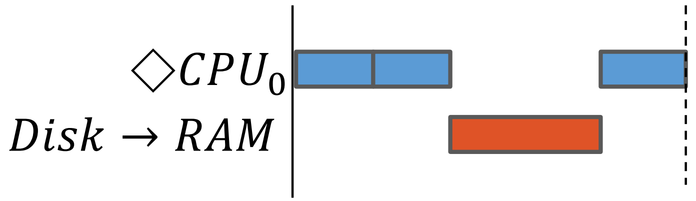


- <u>Load balancing*</u> evens out (1) across devices

  (* How well the jobs are evenly distributed to compute devices?)

Formulation Techniques

- To formally specify…
  - (1) How much time to process a job
  - (2) How much time to fetch the required data



- Latency hiding* is about interleaving (2) while doing (1)
  (* Is the overhead of data fetch invisible?)

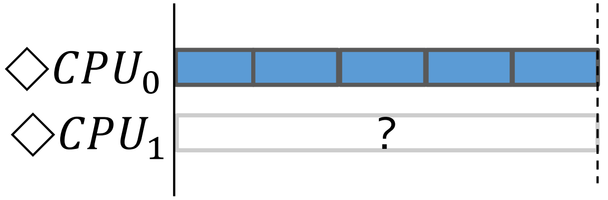
Job Allocation Strategy

- We want to maximize utilization of compute device
- Our strategy: reduce idle time, in following order

Job Allocation Strategy

- We want to maximize utilization of compute device

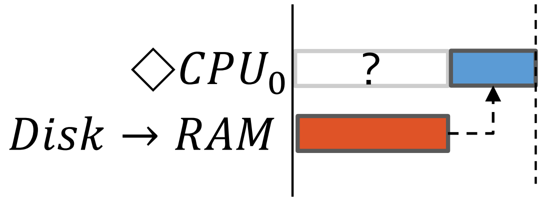
- Our strategy: reduce idle time, in following order
  - **True idle time**: A device is not scheduled nor waiting for a data



Job Allocation Strategy

- We want to maximize utilization of compute device

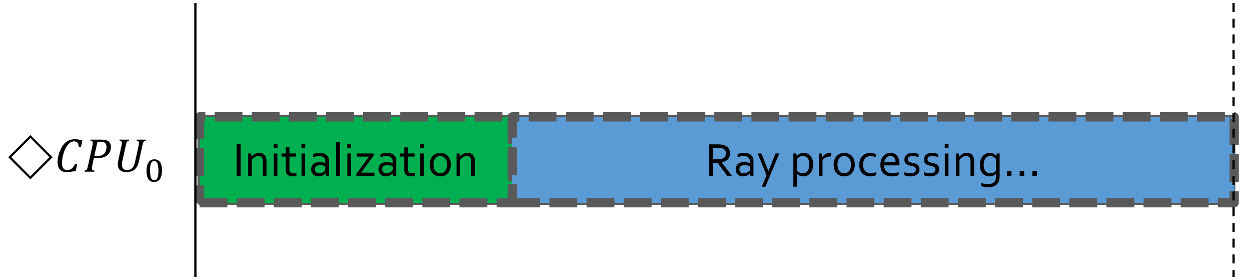
- Our strategy: reduce idle time, in following order
  - **True idle time**: A device is not scheduled nor waiting for a data
  - **Fetching time**: A device is waiting for a data



Job Allocation Strategy

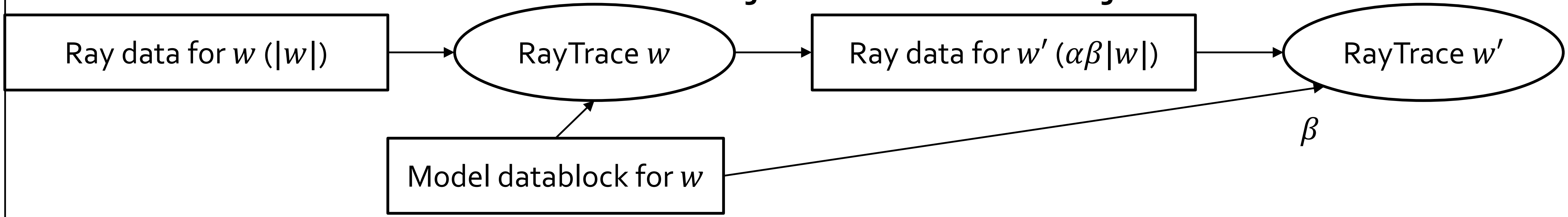
- We want to maximize utilization of compute device

- Our strategy: reduce idle time, in following order
  - **True idle time**: A device is not scheduled nor waiting for a data
  - **Fetching time**: A device is waiting for a data
  - **Setup time**: A device is warming up for processing a job



Adapting to Dynamic Workload

- Prediction: Schedule future jobs with current jobs

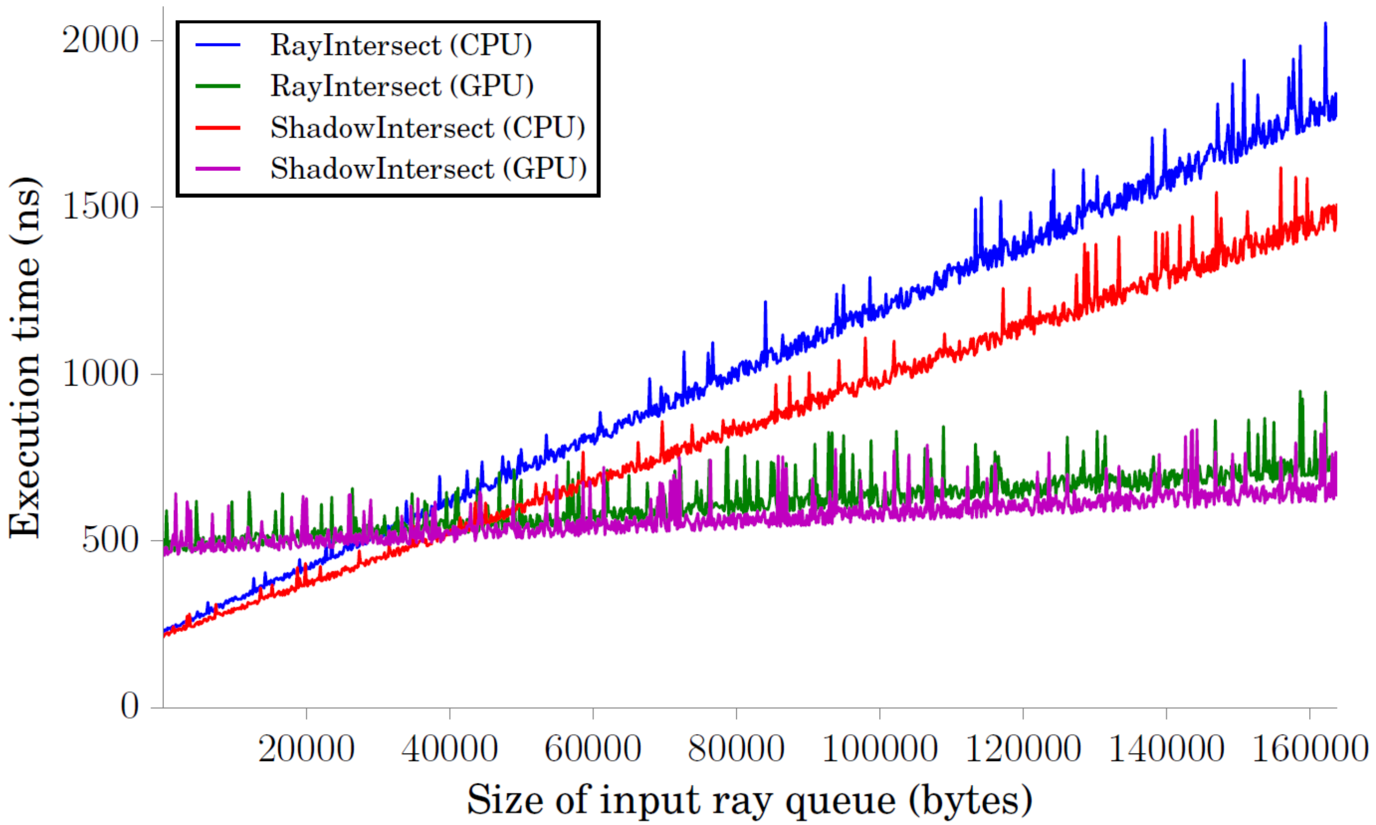


- α is a hit probability within a block
  Just used an empirically correct value (~0.6 on Boeing777, ~0.8 in SponzaMuseum)
- β is an average Russian Roulette pass probability ($= \sum_{r \in w} RR(w)$)
  Determined by averaged acceptance rate

KAIST School of Computing

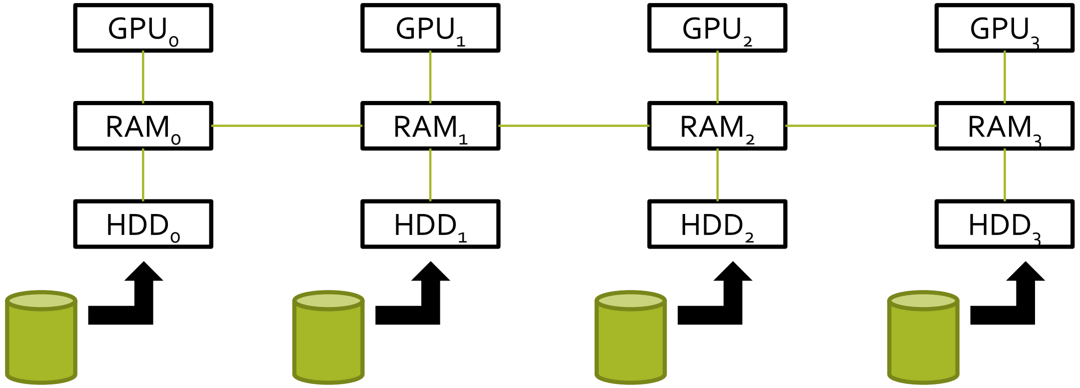
SVRG

Formulation: Timing Model



Centralized Scene DB Structure

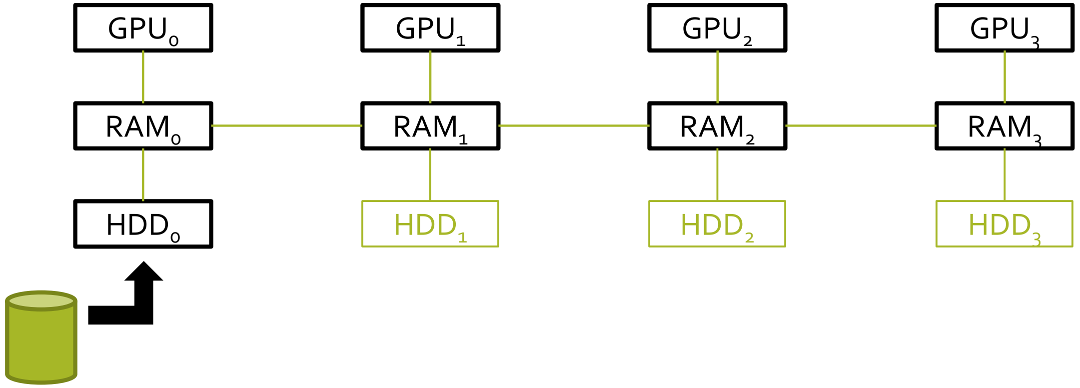
- Decentralized: each node has a copy of the full scene at each HDD from the beginning



- Does not make a diverse data transfer path
  - This is somewhat intended due to simplicity

Centralized Scene DB Structure

- Central Scene DB structure



- Master node gives scene datablock on-demand
- Expected Result: Larger area of applications