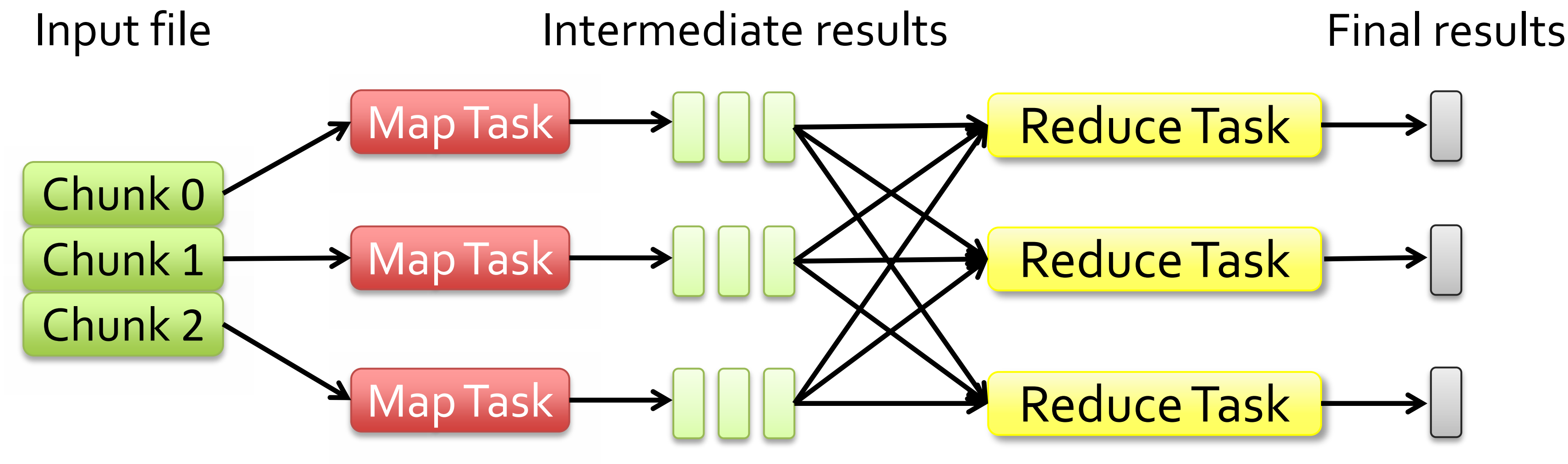
Camdoop

Exploiting In-network Aggregation for Big Data Applications

Paolo Costa
costa@imperial.ac.uk

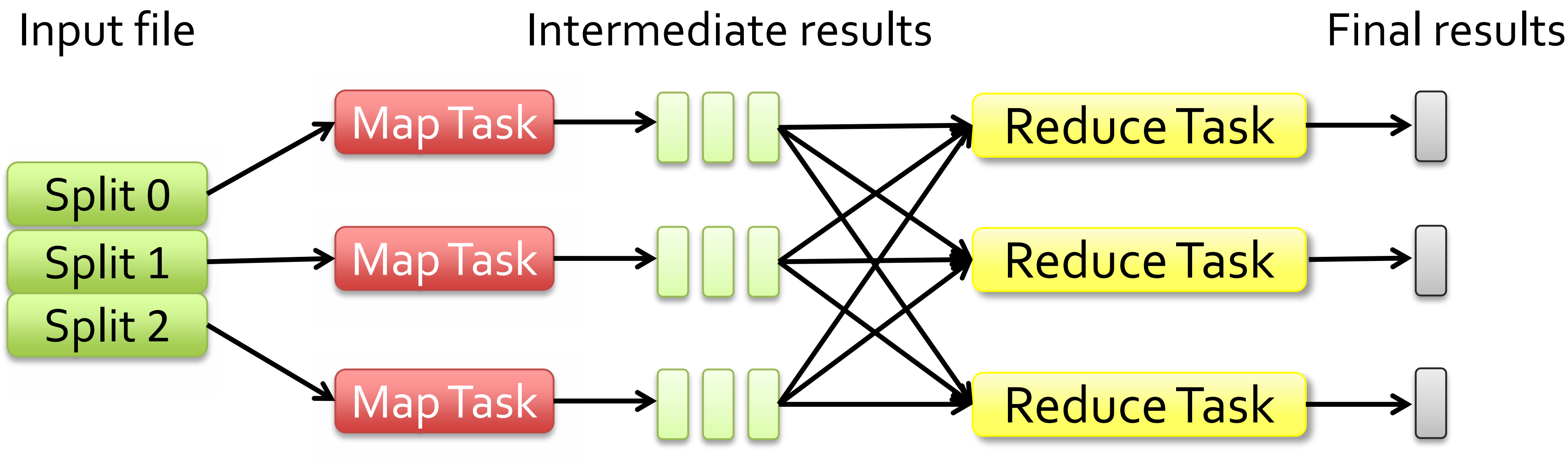
joint work with
Austin Donnelly, Antony Rowstron, and Greg O'Shea (MSR Cambridge)

MapReduce Overview



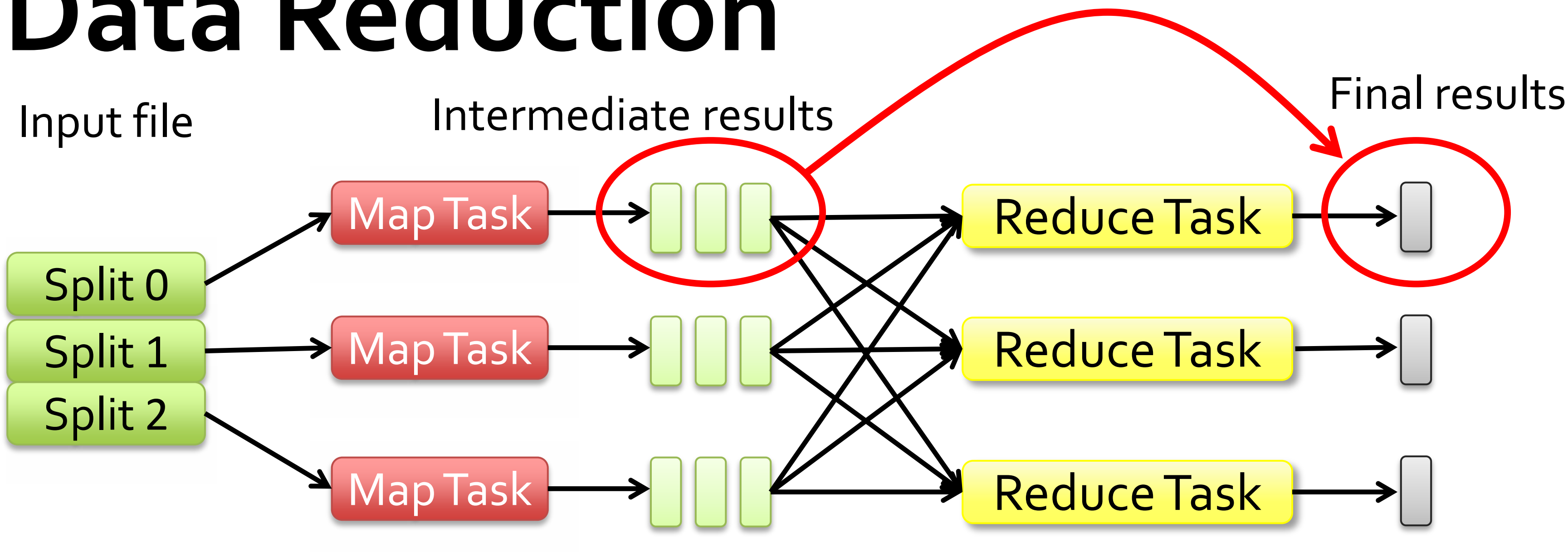
- Map
  - *Processes* input data and *generates* (key, value) pairs
- Shuffle
  - *Distributes* the intermediate pairs to the reduce tasks
- Reduce
  - *Aggregates* all values associated to each key

Problem



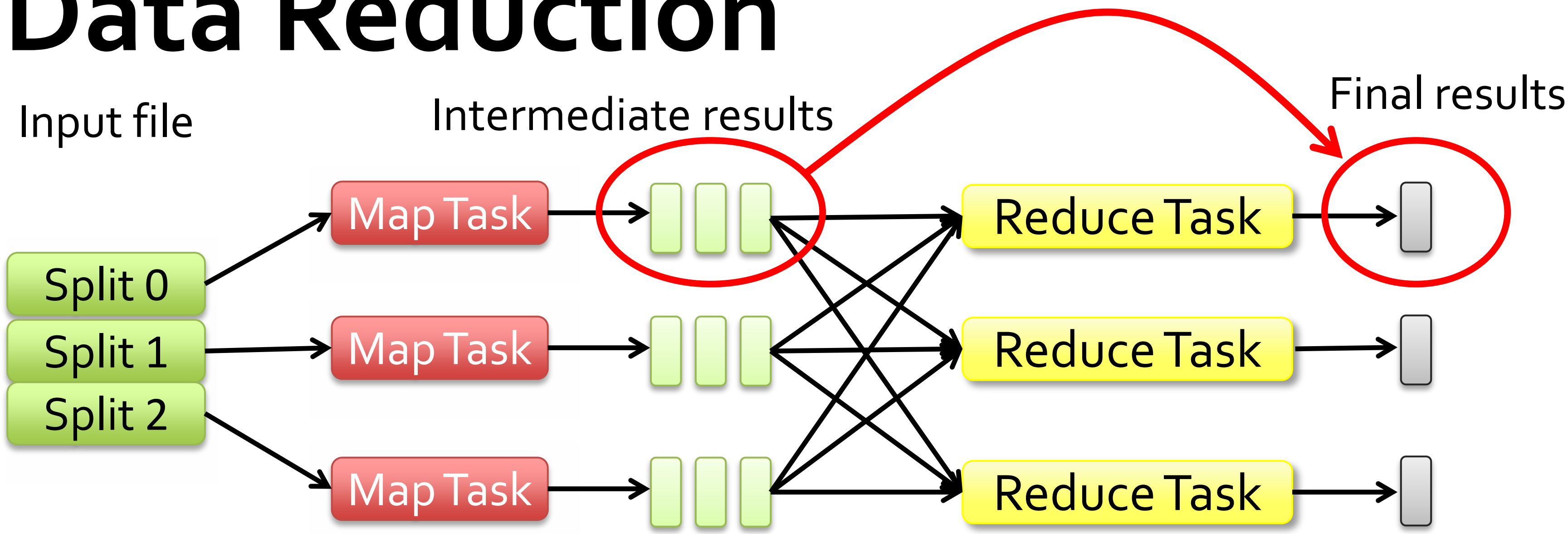
- Shuffle phase is challenging for data center networks
  - All-to-all traffic pattern with *$O(N^2)$* flows
  - Led to proposals for full-bisection bandwidth

Data Reduction



- The final results are typically much smaller than the intermediate results
- In most Facebook jobs the final size is 5.4 % of the intermediate size
- In most Yahoo jobs the ratio is 8.2 %

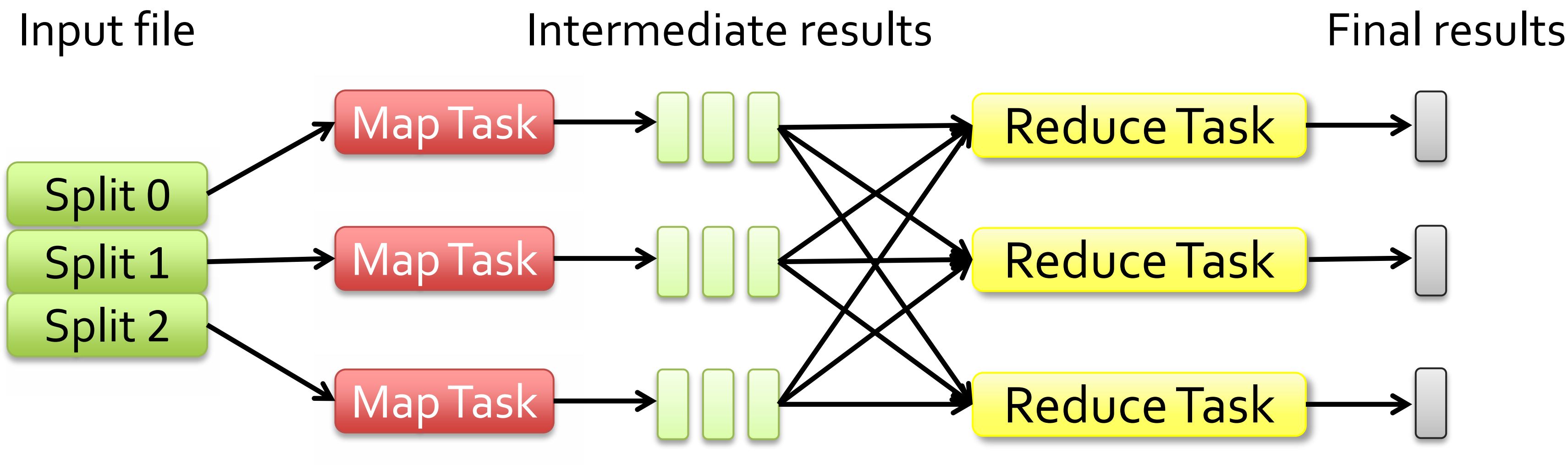
Data Reduction



- The final results are typically much smaller than the intermediate results

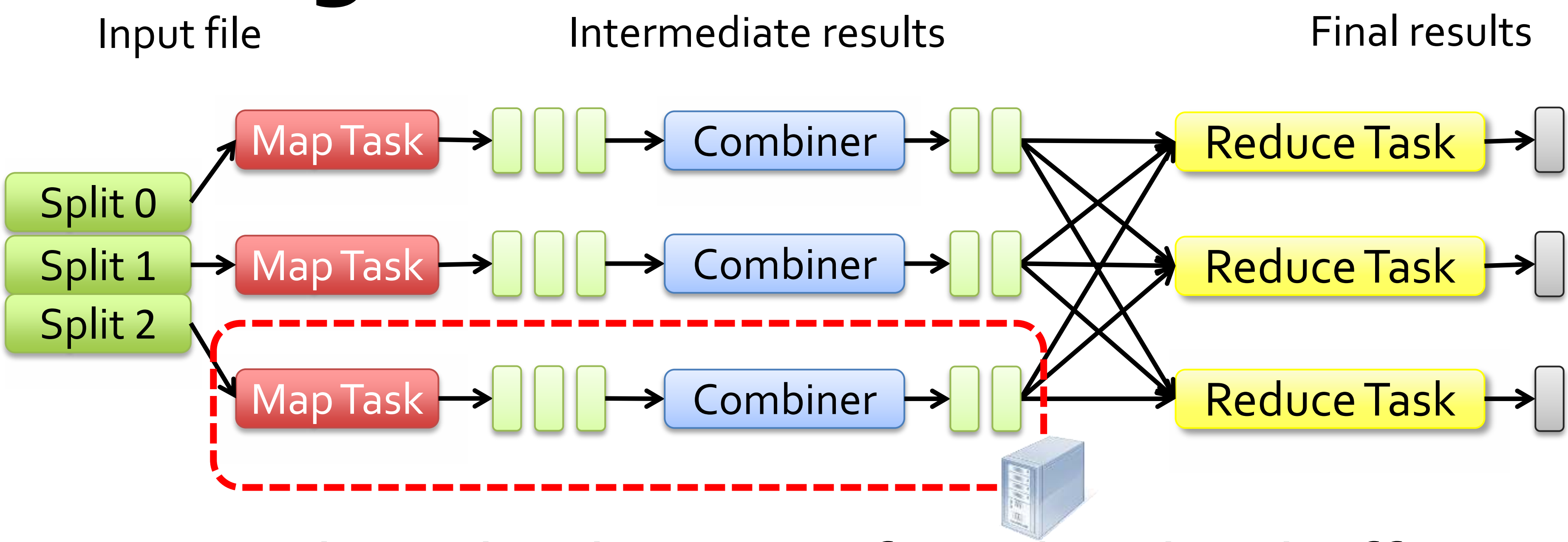
How can we exploit this to reduce the traffic and improve the performance of the shuffle phase?

Background: Combiners



- To reduce the data transferred in the shuffle, users can specify a combiner function
  - Aggregates the local intermediate pairs
- Server-side only => limited aggregation

Background: Combiners

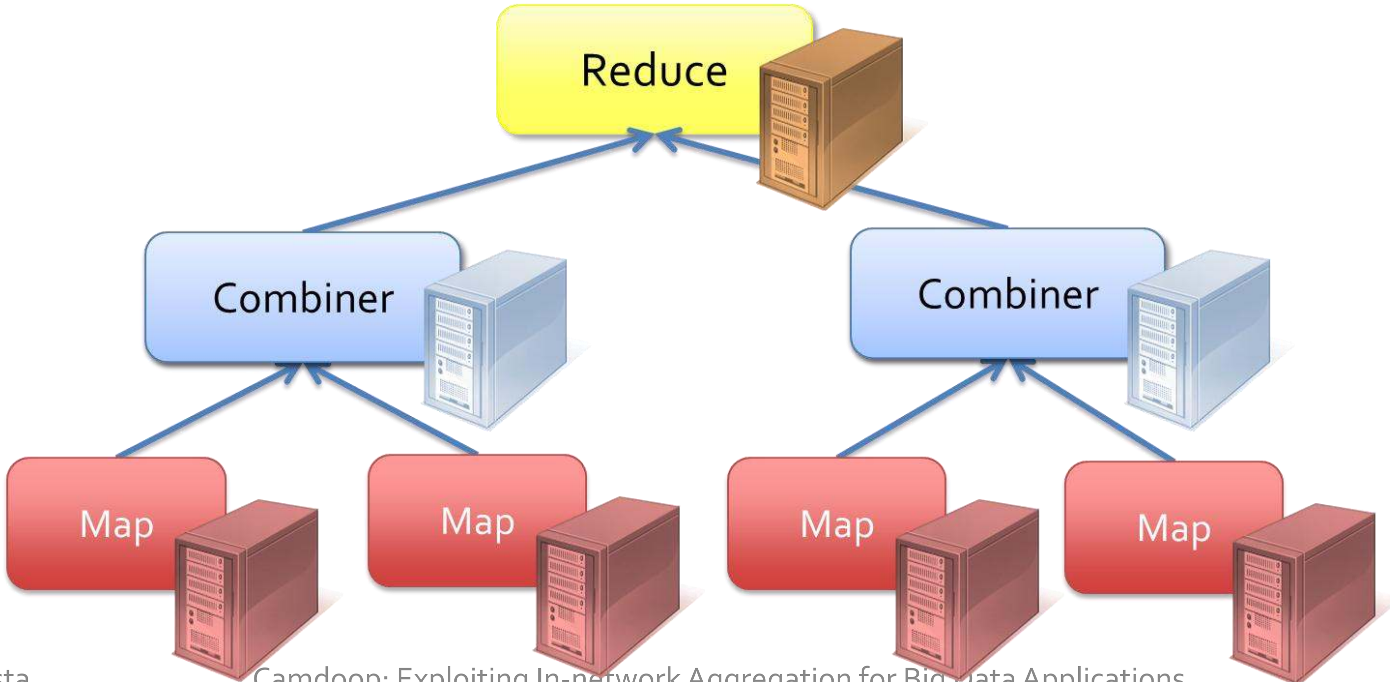


- To reduce the data transferred in the shuffle, users can specify a combiner function
  - Aggregates the local intermediate pairs

- Server-side only => limited aggregation

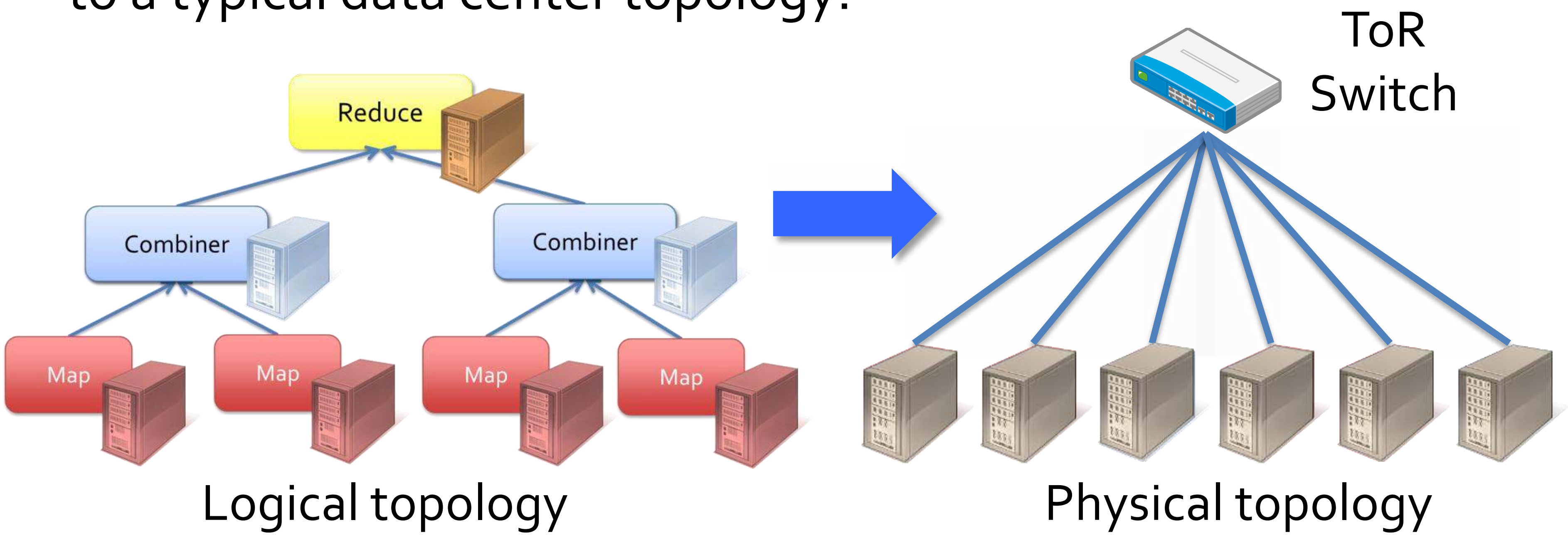
Distributed Combiners

- It has been proposed to use aggregation trees in MapReduce to perform multiple steps of combiners
  - e.g., rack-level aggregation [Yu et al., SOSP'09]



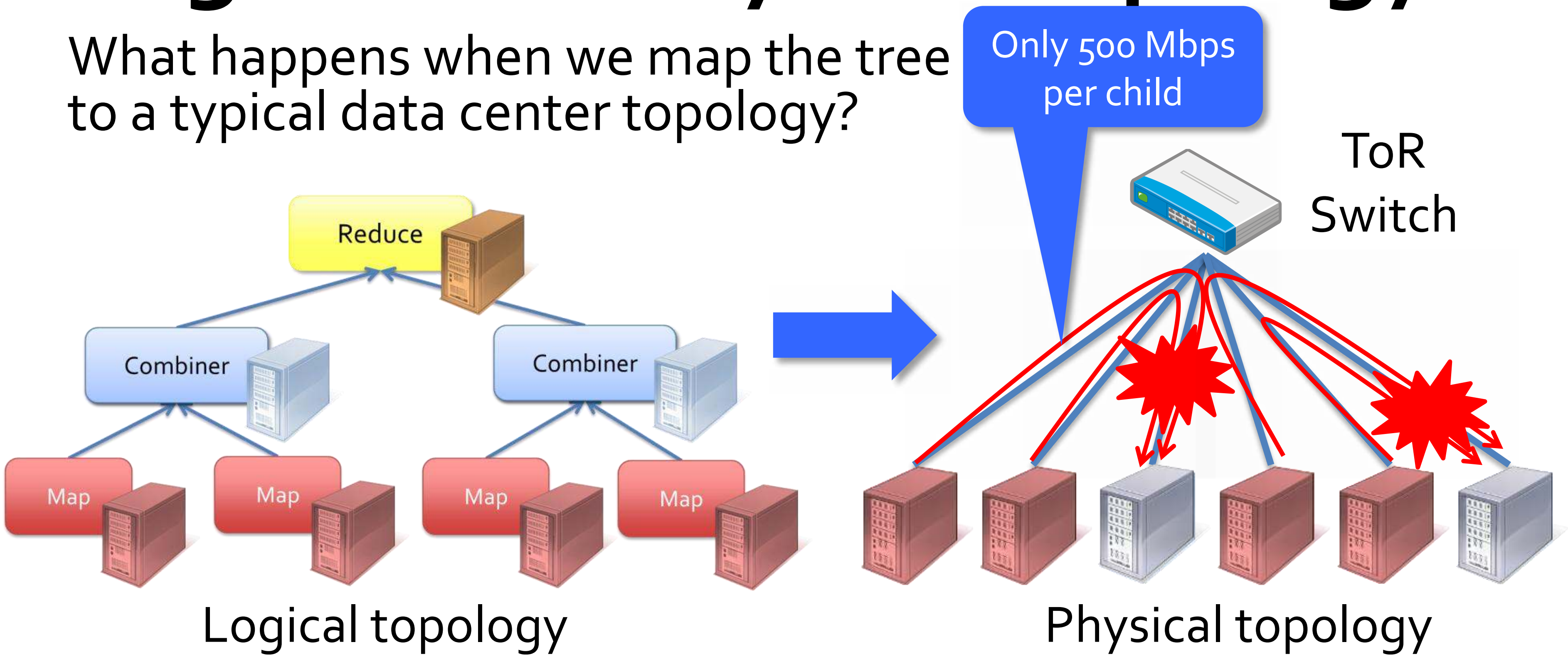
Logical and Physical Topology

What happens when we map the tree
to a typical data center topology?



Logical and Physical Topology

What happens when we map the tree to a typical data center topology?

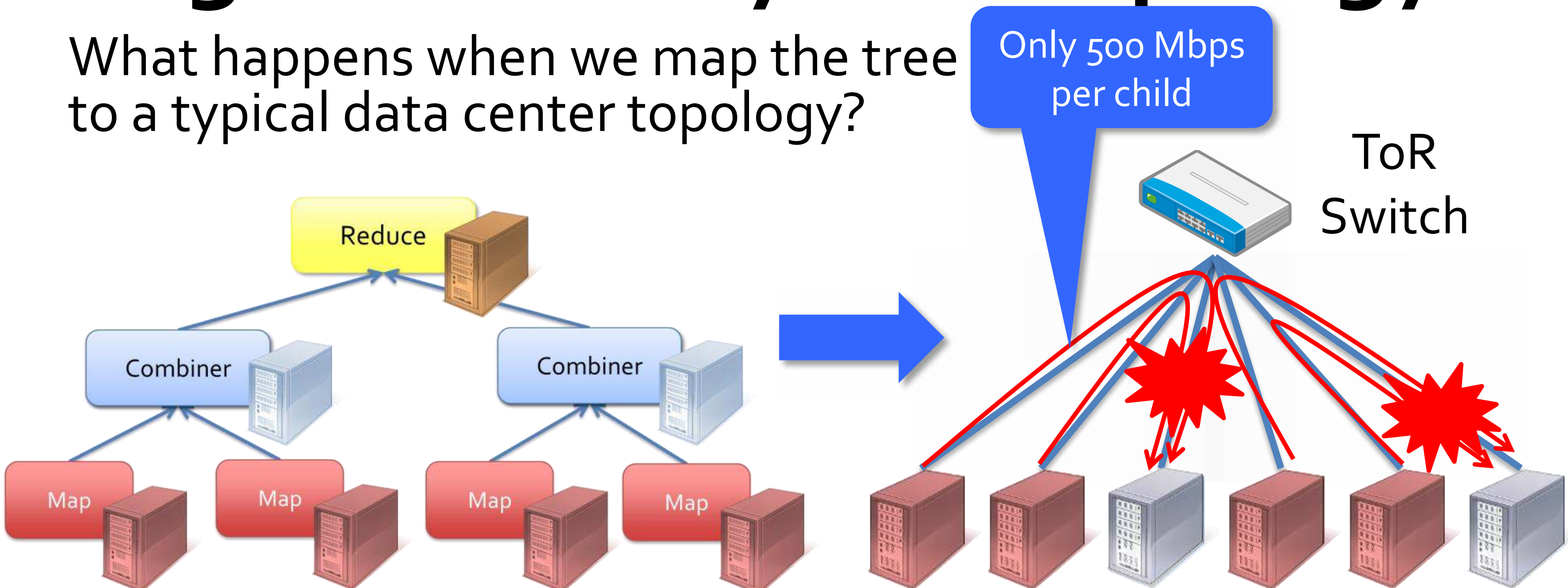


The server link is the bottleneck
Full-bisection bandwidth does not help here

Mismatch between physical and logical topology
Two logical links are mapped onto the same physical link

Logical and Physical Topology

What happens when we map the tree to a typical data center topology?



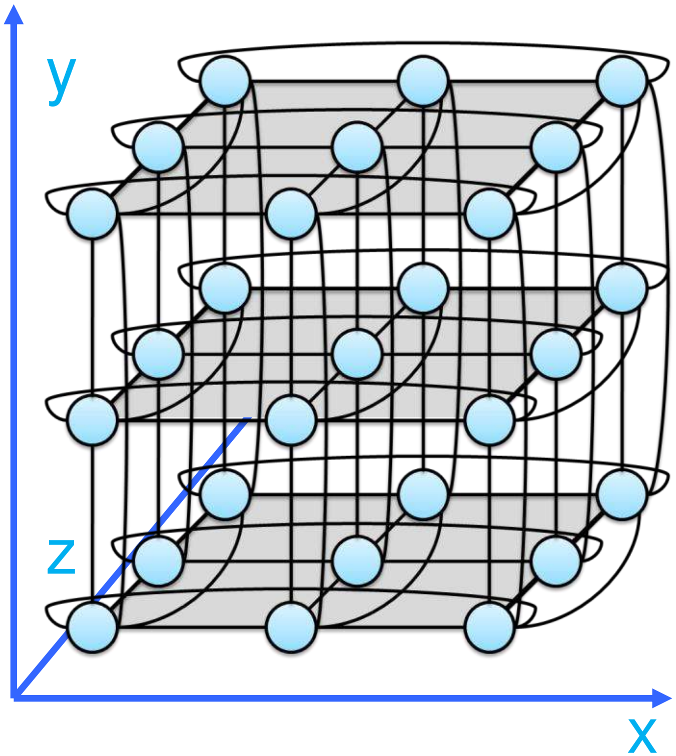
Camdoop Goal

Perform the combiner functions within the network as opposed to application-level solutions

Reduce shuffle time by aggregating packets on path

How Can We Perform In-network Processing?

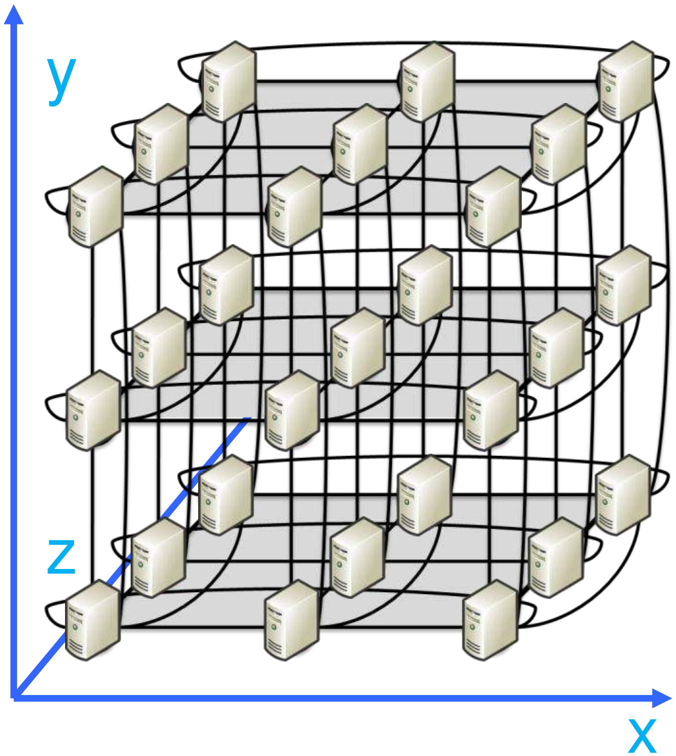
- We exploit CamCube
  - Direct-connect topology
  - 3D torus
  - Uses no switches / routers for internal traffic



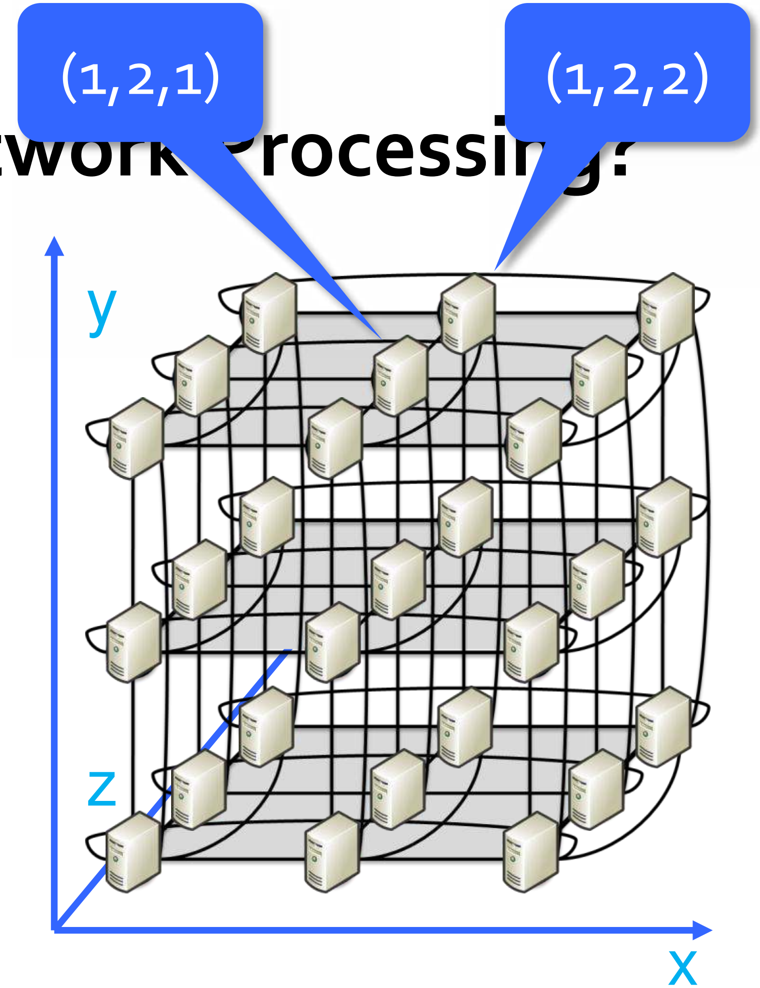
How Can We Perform In-network Processing?

- We exploit CamCube
  - Direct-connect topology
  - 3D torus
  - Uses no switches / routers for internal traffic

- Servers intercept, forward and process packets



How Can We Perform In-network Processing?

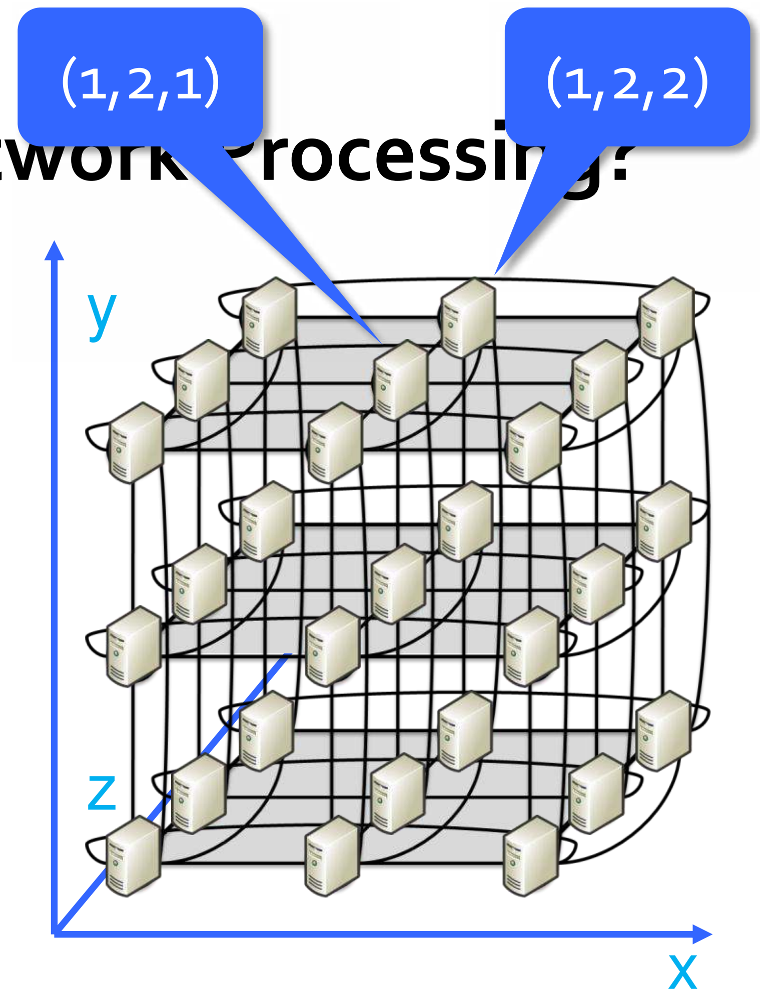


- We exploit CamCube
  - Direct-connect topology
  - 3D torus
  - Uses no switches / routers for internal traffic
- Servers intercept, forward and process packets
- Nodes have `(x,y,z)` coordinates
  - This defines a key-space (=> key-based routing)
  - Coordinates are locally re-mapped in case of failures

How Can We Perform In-network Processing?

- We exploit CamCube
  - Direct-connect topology
  - 3D torus
  - Uses no switches / routers for internal traffic

- Servers intercept, forward and process packets

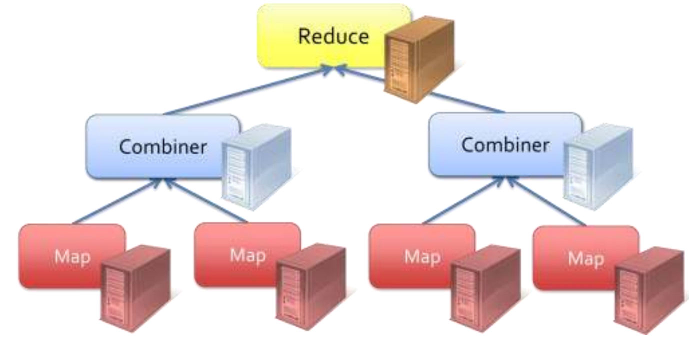


Key property

No distinction between network and computation devices

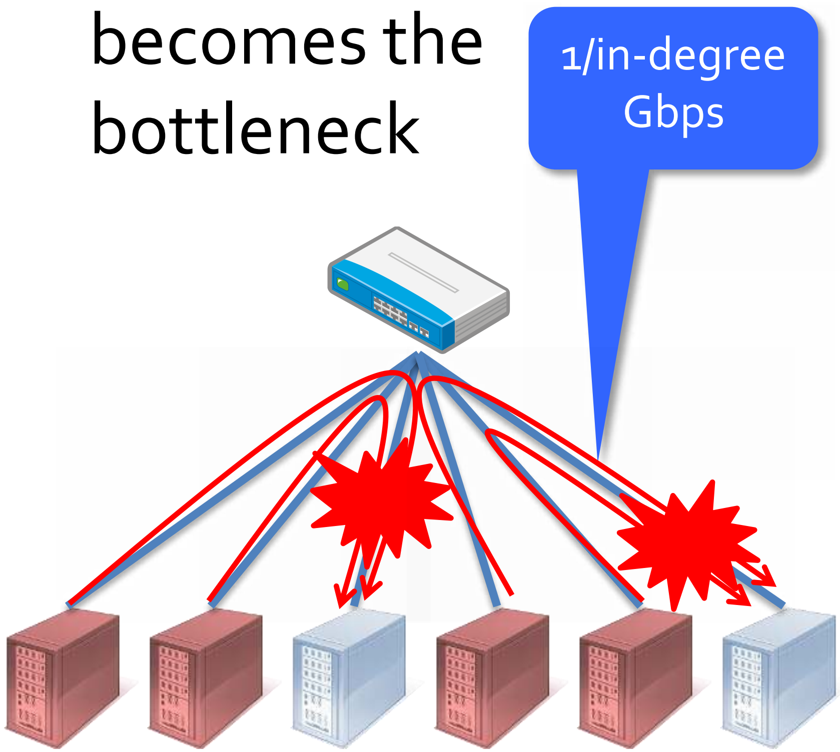
Servers can perform arbitrary packet processing on-path

Mapping a tree…



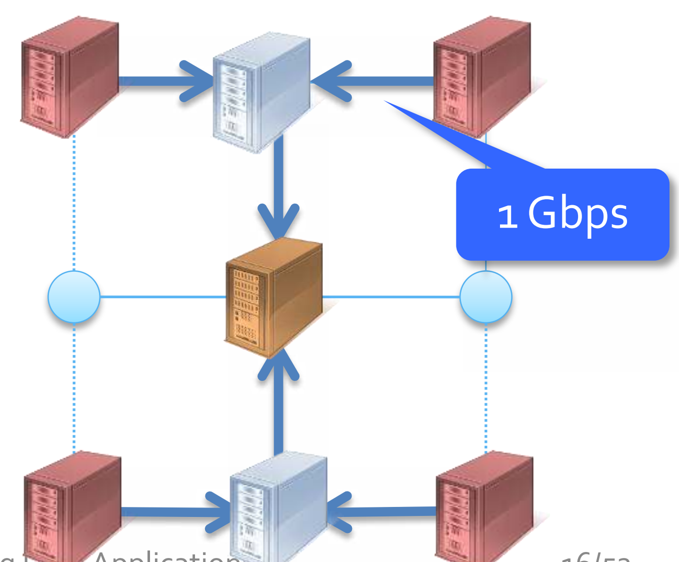
… on a switched topology

- The 1 Gbps link becomes the bottleneck



… on CamCube

- Packets are aggregated on path (=> less traffic)
- 1:1 mapping btw. logical and physical topology

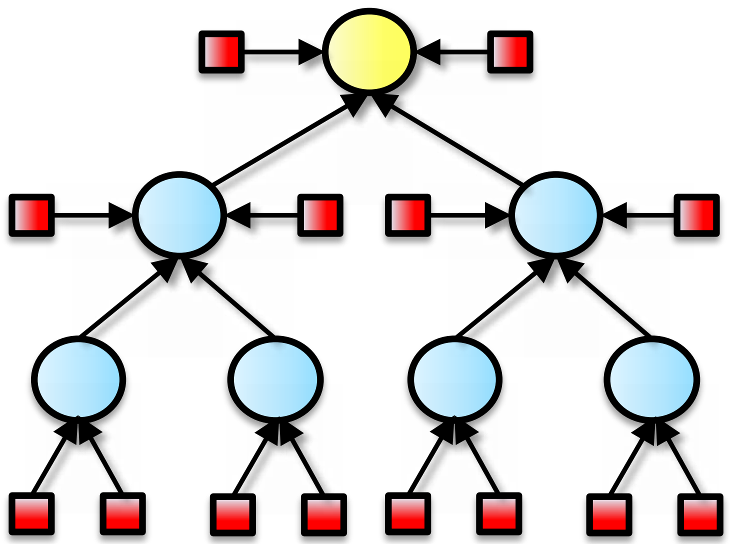


Camdoop Design

Goals

1. No change in the programming model
2. Exploit network locality
3. Good server and link load distribution
4. Fault-tolerance

Design Goal #1

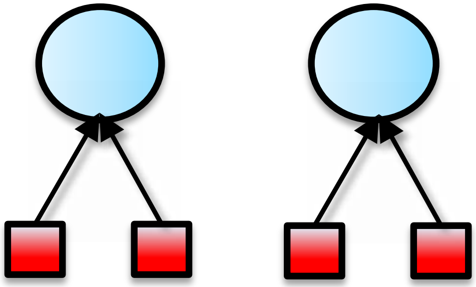


Programming Model

- Camdoop adopts the same MapReduce model
- GFS-like distributed file-system
  - Each server runs map tasks on local chunks
- We use a spanning tree
  - Combiners aggregate map tasks and children results (if any) and stream the results to the parents
  - The root runs the reduce task and generates the final output

Design Goal #2

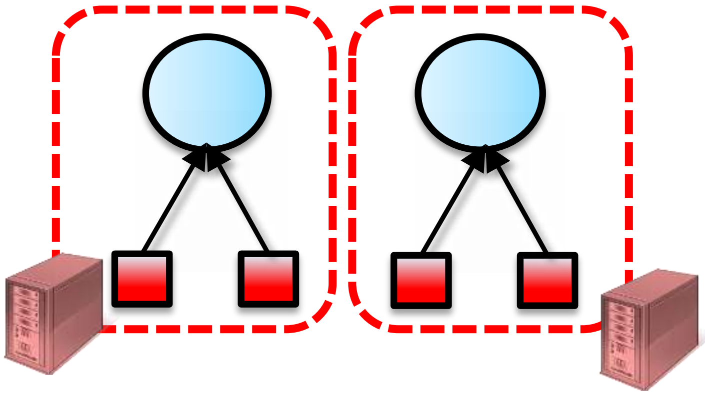
Network locality



How to map the tree nodes to servers?

Design Goal #2

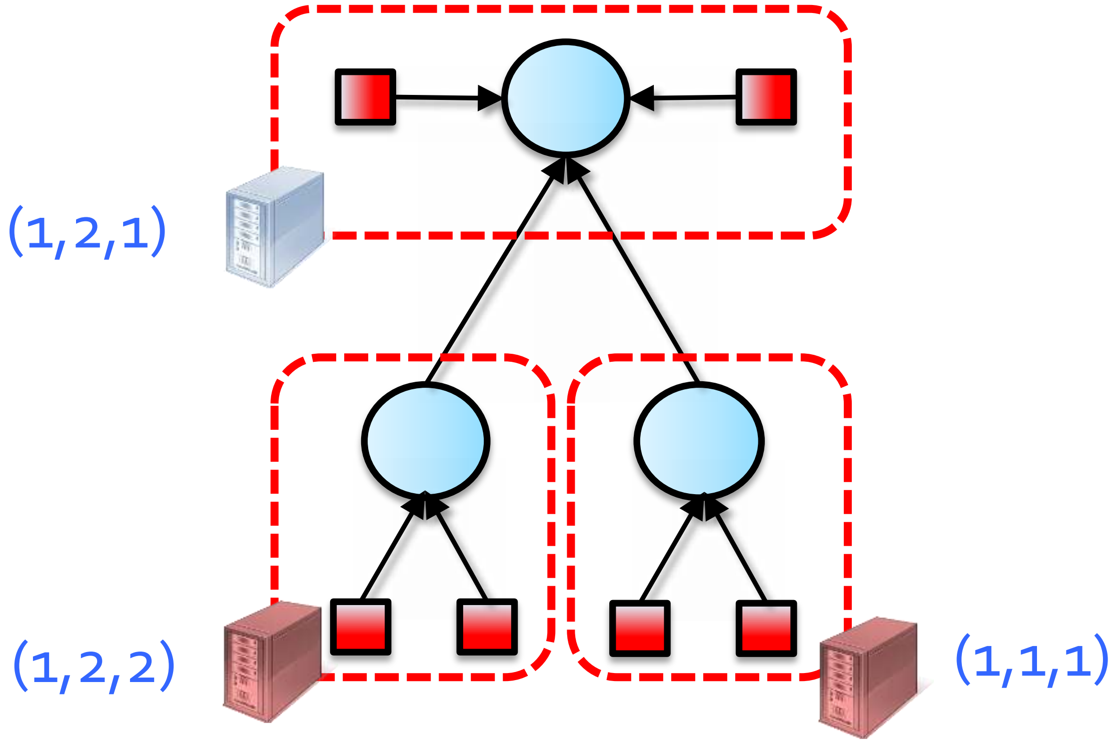
Network locality



Map task outputs are always read from the local disk

Design Goal #2

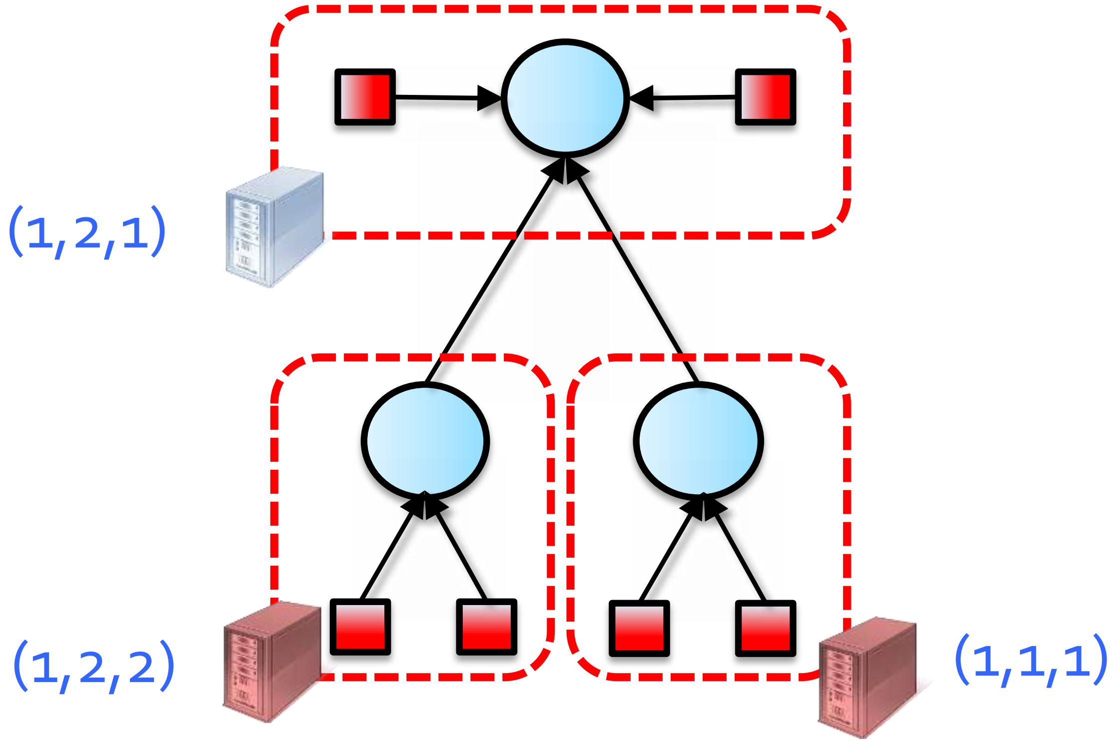
Network locality



The parent-children are mapped on physical neighbors

Design Goal #2

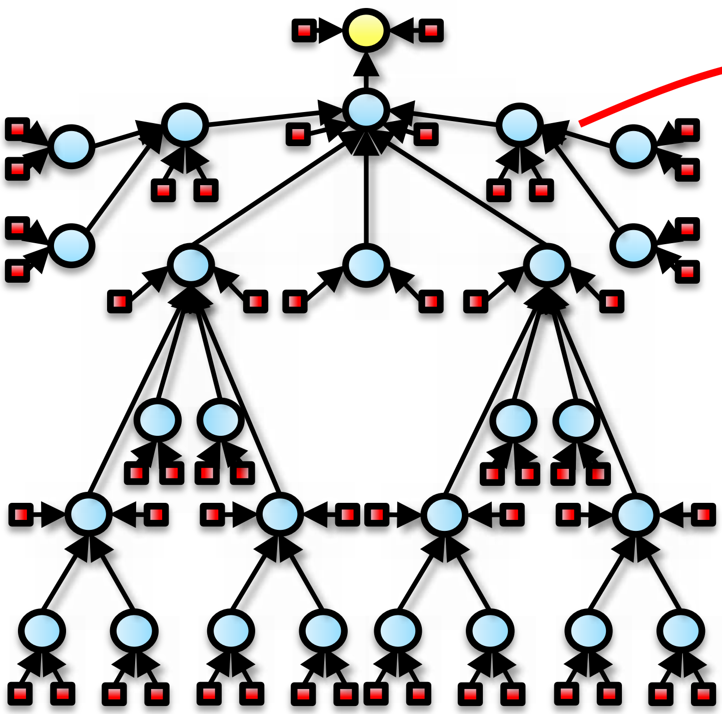
Network locality



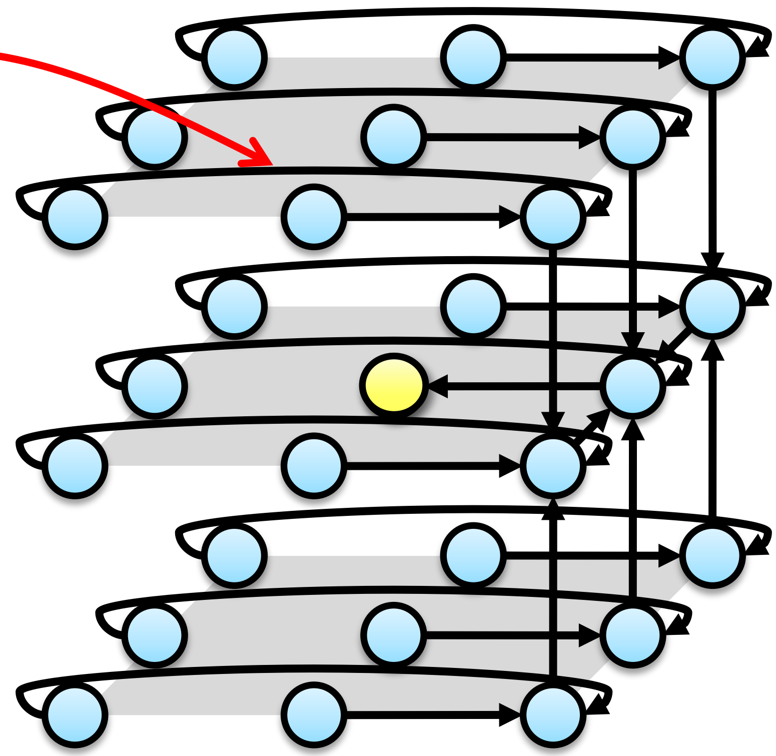
This ensures maximum locality and optimizes network transfer

Network Locality

Logical View



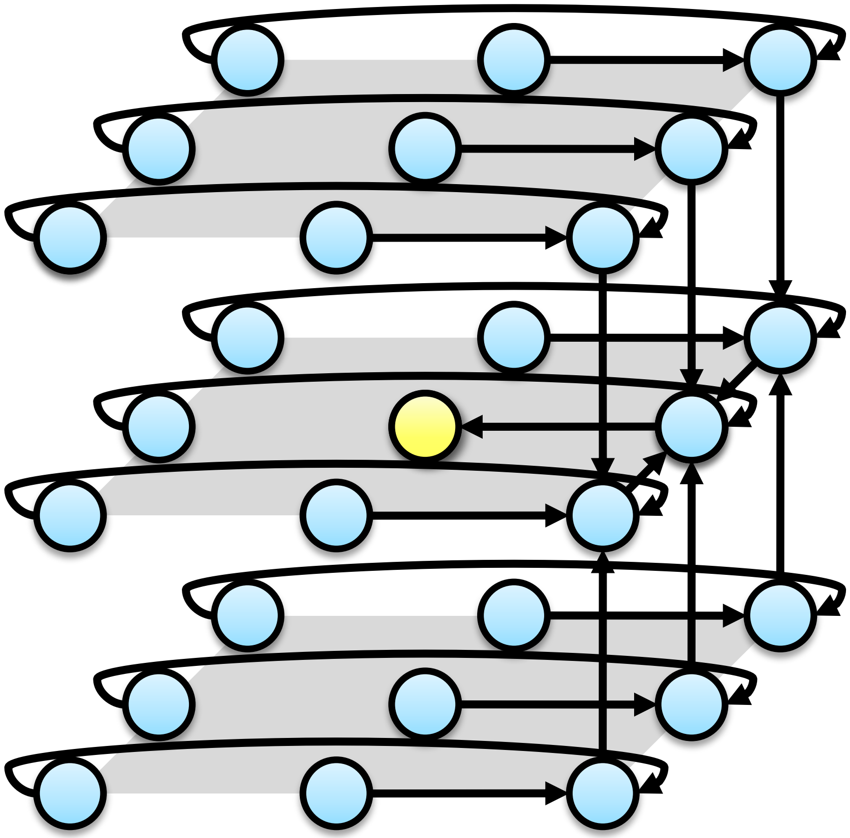
Physical View (3D Torus)



One physical link is used by one and only one logical link

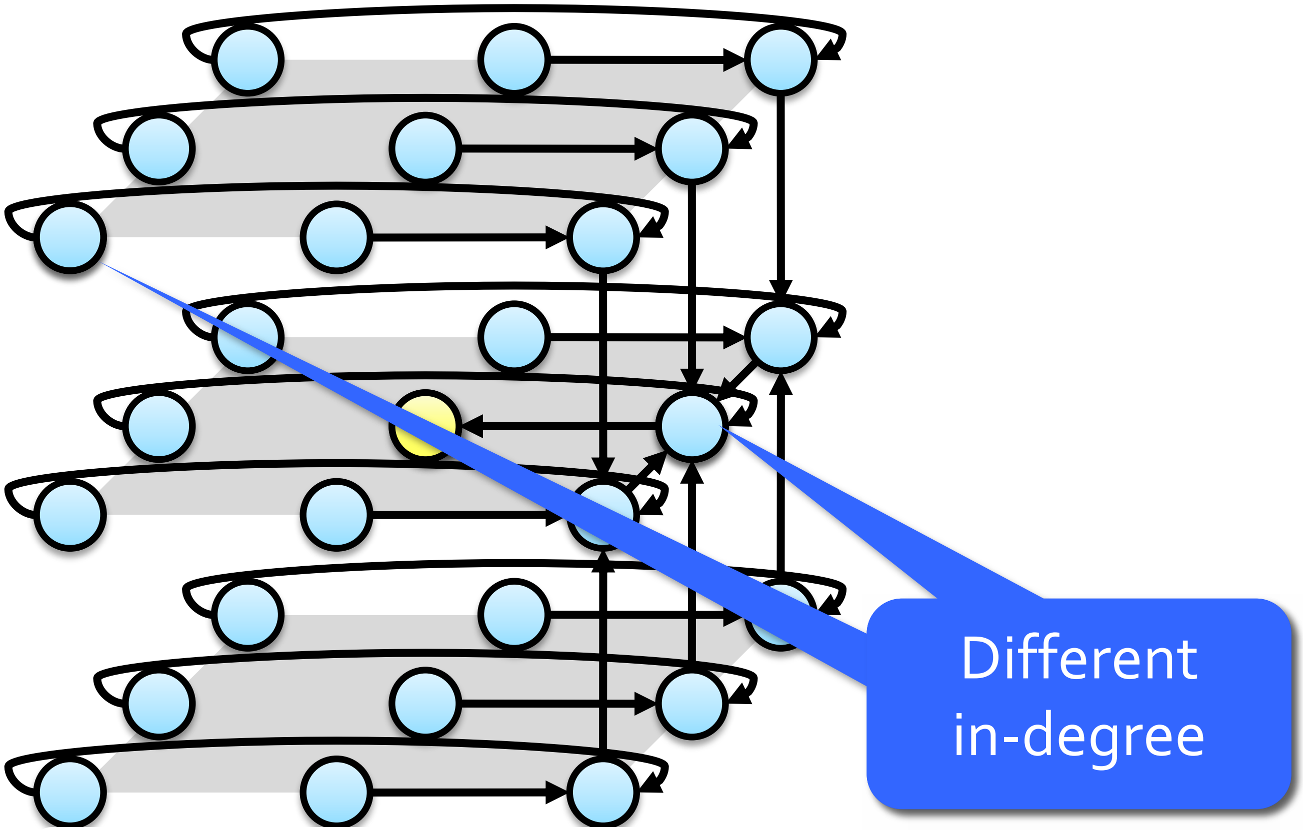
Design Goal #3

Load Distribution



Design Goal #3

Load Distribution

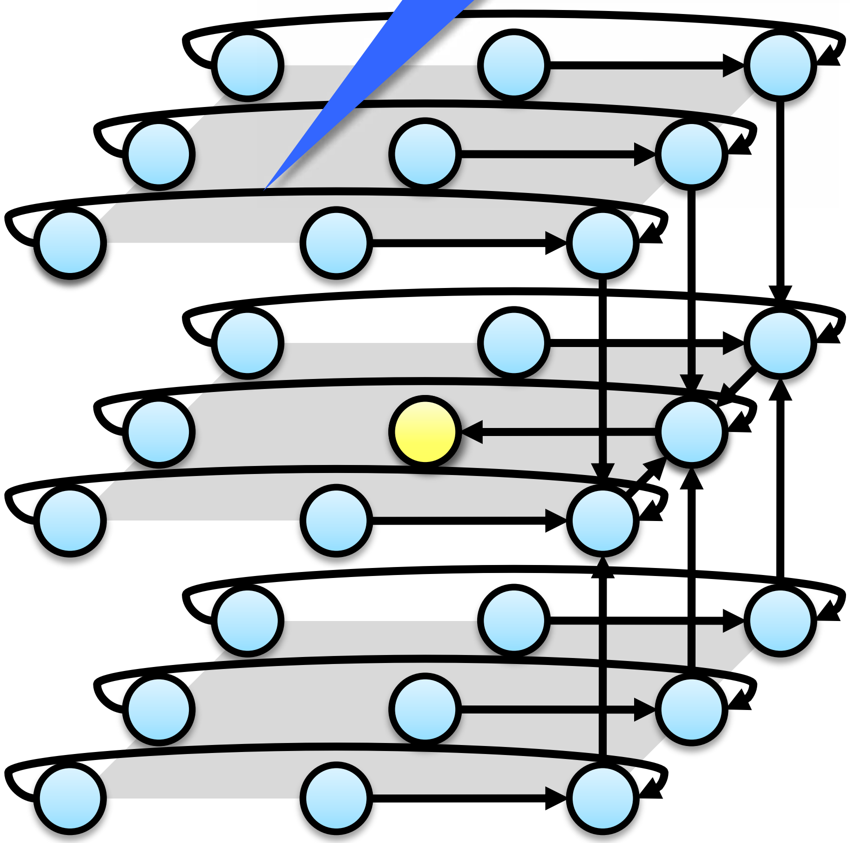


Poor server load distribution

Design #3

Load Distribution

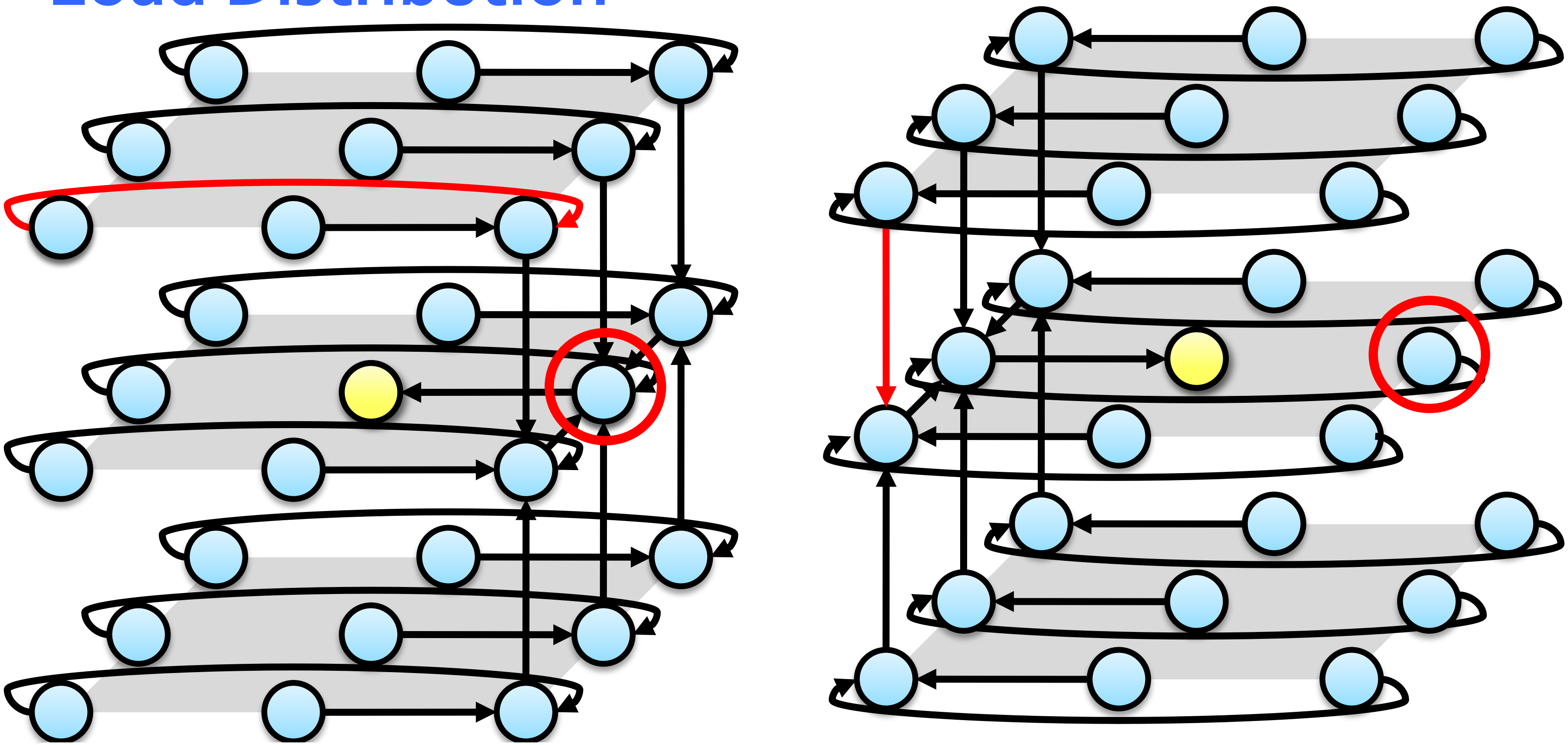
Only 1 Gbps (instead of 6)



Poor bandwidth utilization

Design Goal #3

Load Distribution

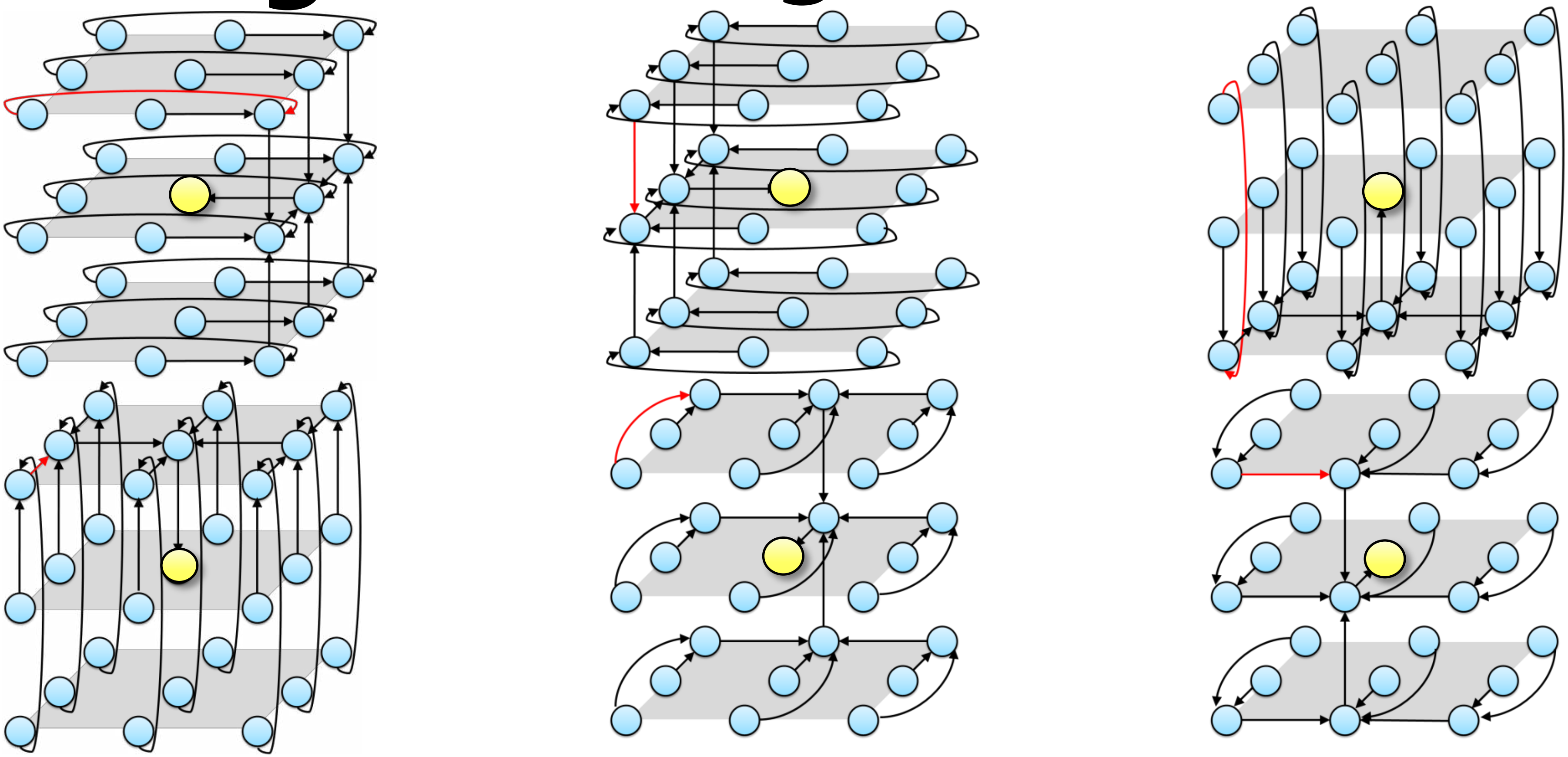


Solution: stripe the data across disjoint trees

✓Different links are used

✓Improves load distribution

Design Goal #3



Solution: stripe the data across 6 disjoint trees

✓All links are used => (Up to) 6 Gbps / server

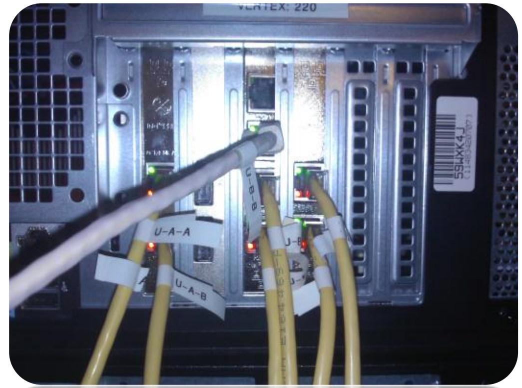
✓Good load distribution

Design Goal #4

Fault-tolerance

- The tree is built in the coordinate space
  - CamCube remaps coordinates in case of failures
- Details in the paper

Evaluation



Testbed

- 27-server CamCube (3 x 3 x 3)
- Quad-core Intel Xeon 5520 2.27 Ghz
- 12GB RAM
- 6 Intel PRO/1000 PT 1 Gbps ports
- Runtime & services implemented in user-space

Simulator

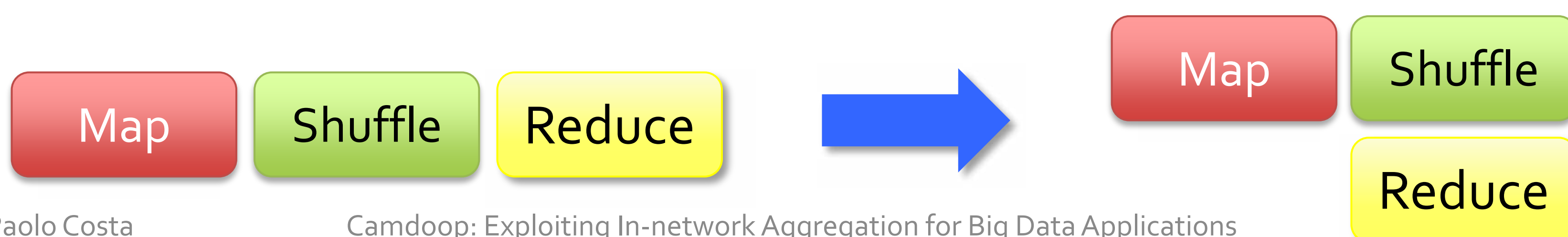
- Packet-level simulator (CPU overhead not modelled)
- 512-server (8x8x8) CamCube

Evaluation

Design and implementation recap

	Camdoop
Shuffle & reduce parallelized	✓

- Reduce phase is parallelized with the shuffle phase
  - Since all streams are ordered, as soon as the root receive at least one packet from all children, it can start the reduce function
  - No need to store to disk intermediate results on reduce servers



Evaluation

Design and implementation recap

	Camdoop
Shuffle & reduce parallelized	✓
CamCube	✓
Six disjoint trees	✓
In-network aggregation	✓

Evaluation

Design and implementation recap

	Camdoop	TCP Camdoop (switch)
Shuffle & reduce parallelized	✓	✓
CamCube	✓	✗
Six disjoint trees	✓	✗
In-network aggregation	✓	✗

- TCP Camdoop (switch)
  - 27 CamCube servers attached to a ToR switch
  - TCP is used to transfer data in the shuffle phase

Evaluation

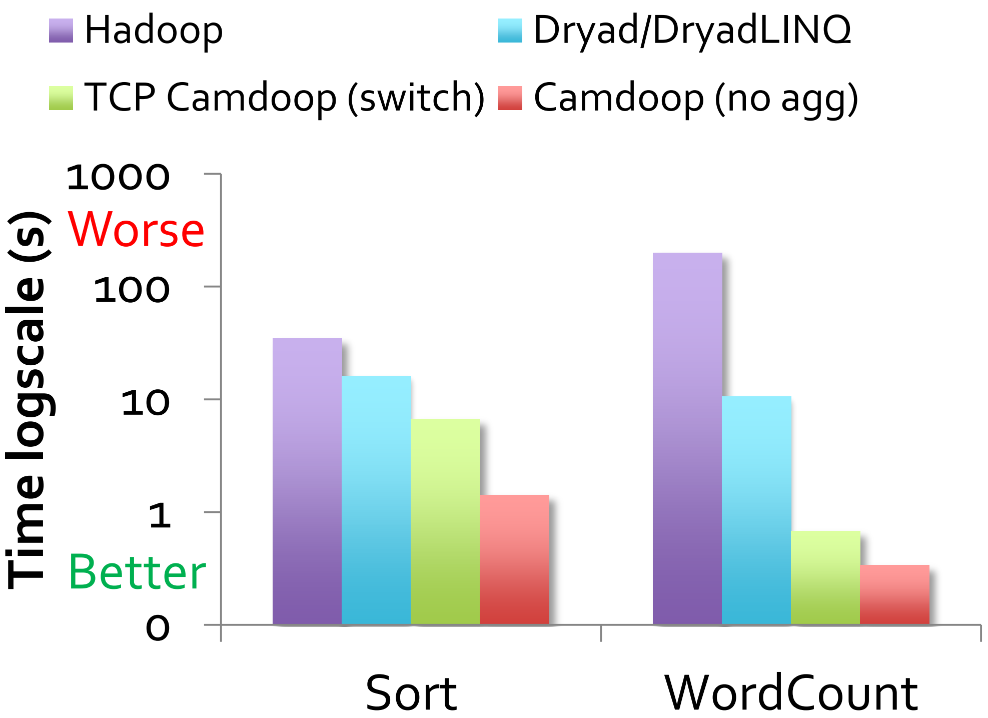
Design and implementation recap

	Camdoop	TCP Camdoop (switch)	Camdoop (no agg)
Shuffle & reduce parallelized	✓	✓	✓
CamCube	✓	✗	✓
Six disjoint trees	✓	✗	✓
In-network aggregation	✓	✗	✗

- Camdoop (no agg)
  - Like Camdoop but without in-network aggregation
  - Shows the impact of just running on CamCube

Validation against Hadoop & Dryad

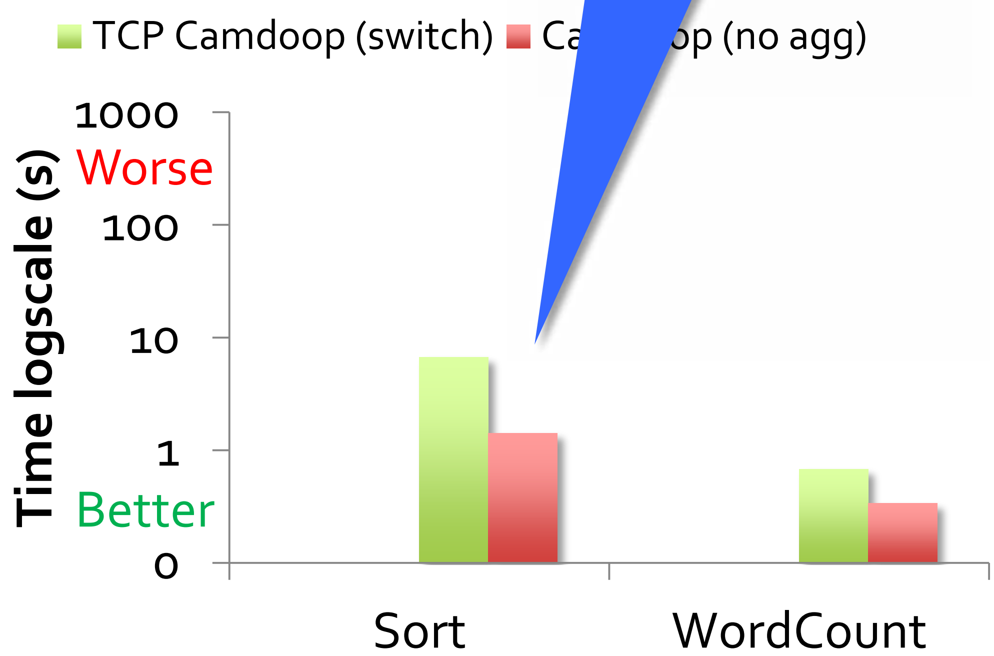
- Sort and WordCount
- Camdoop baselines are competitive against Hadoop and Dryad
- Several reasons:
  - Shuffle and reduce parellized
  - Fine-tuned implementation



Validation against Hadoop

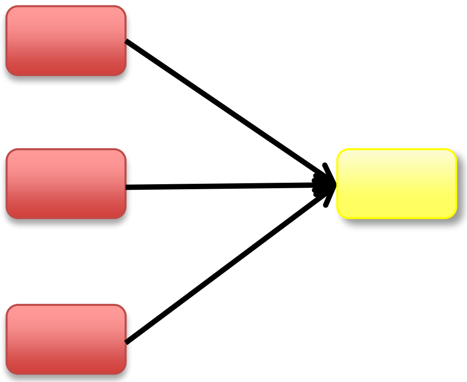
We consider these as our baselines

- Sort and WordCount
- Camdoop baselines are competitive against Hadoop and Dryad
- Several reasons:
  - Shuffle and reduce parellized
  - Fine-tuned implementation

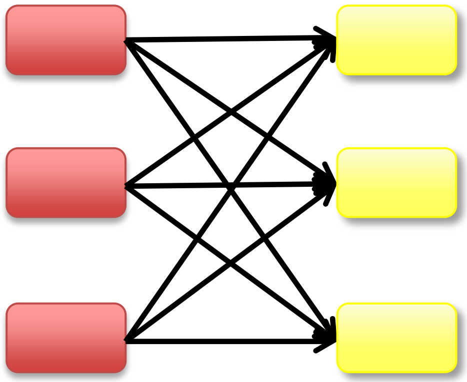


Parameter Sweep

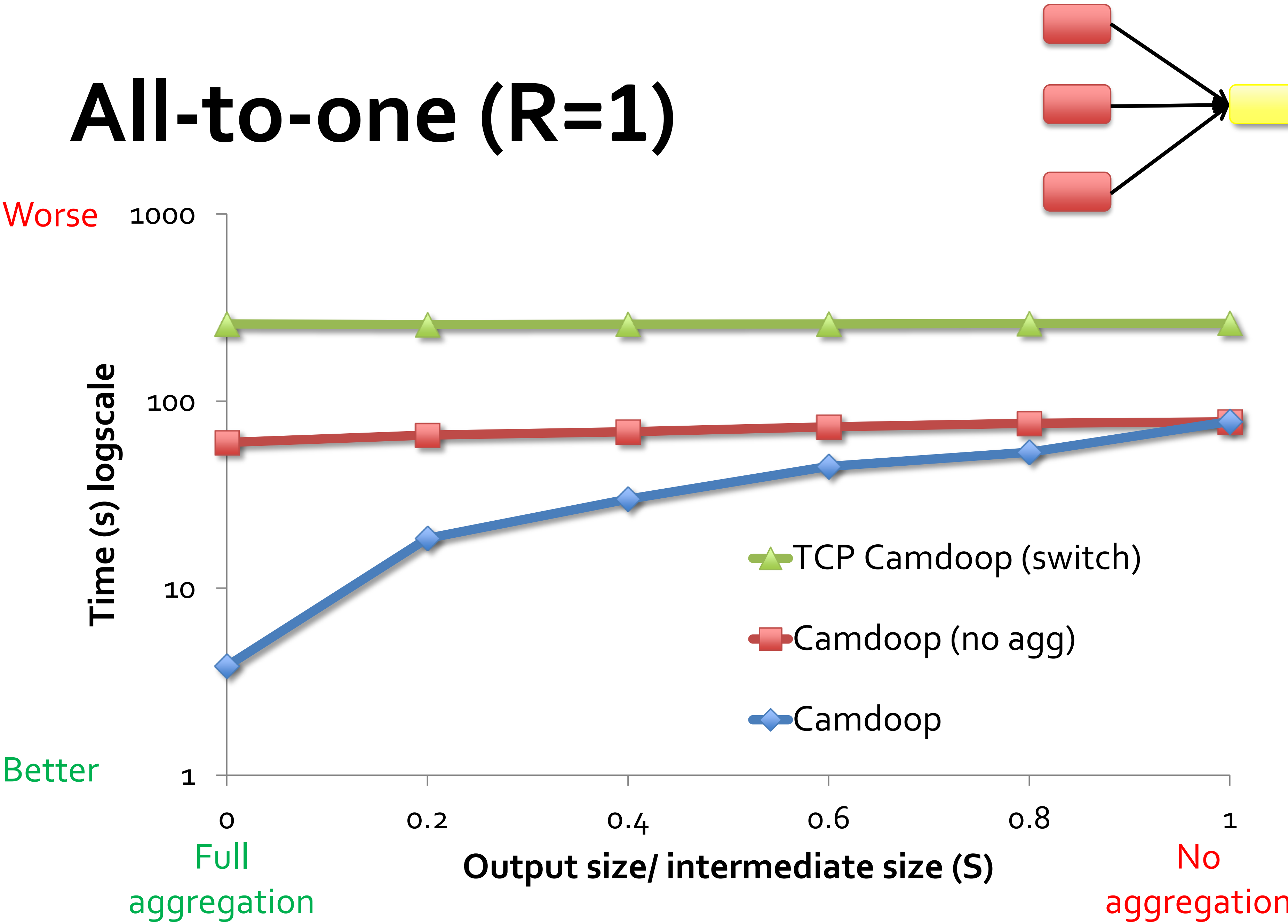
- Output size / intermediate size (S)
  - S=1 (no aggregation)
    - Every key is unique
  - S=1/N ≈ 0 (full aggregation)
    - Every key appears in all map task outputs
  - We use synthetic workloads to explore different value of S
    - Intermediate data size is 22.2 GB (843 MB/server)

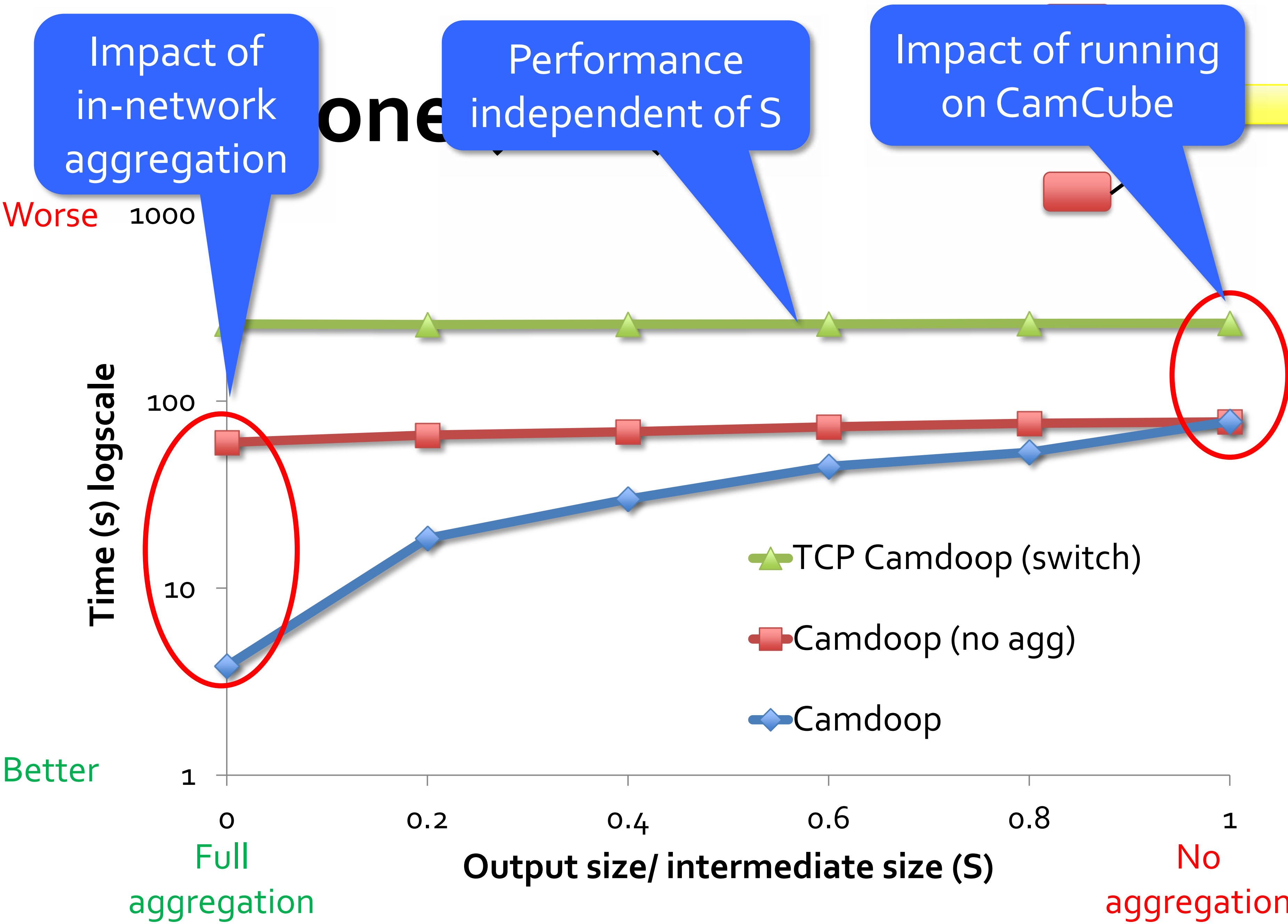


- Reduce tasks (R)
  - R= 1 (all-to-one)
    - E.g., Interactive queries, top-K jobs
  - R=N (all-to-all)
    - Common setup in MapReduce jobs
    - N output files are generated

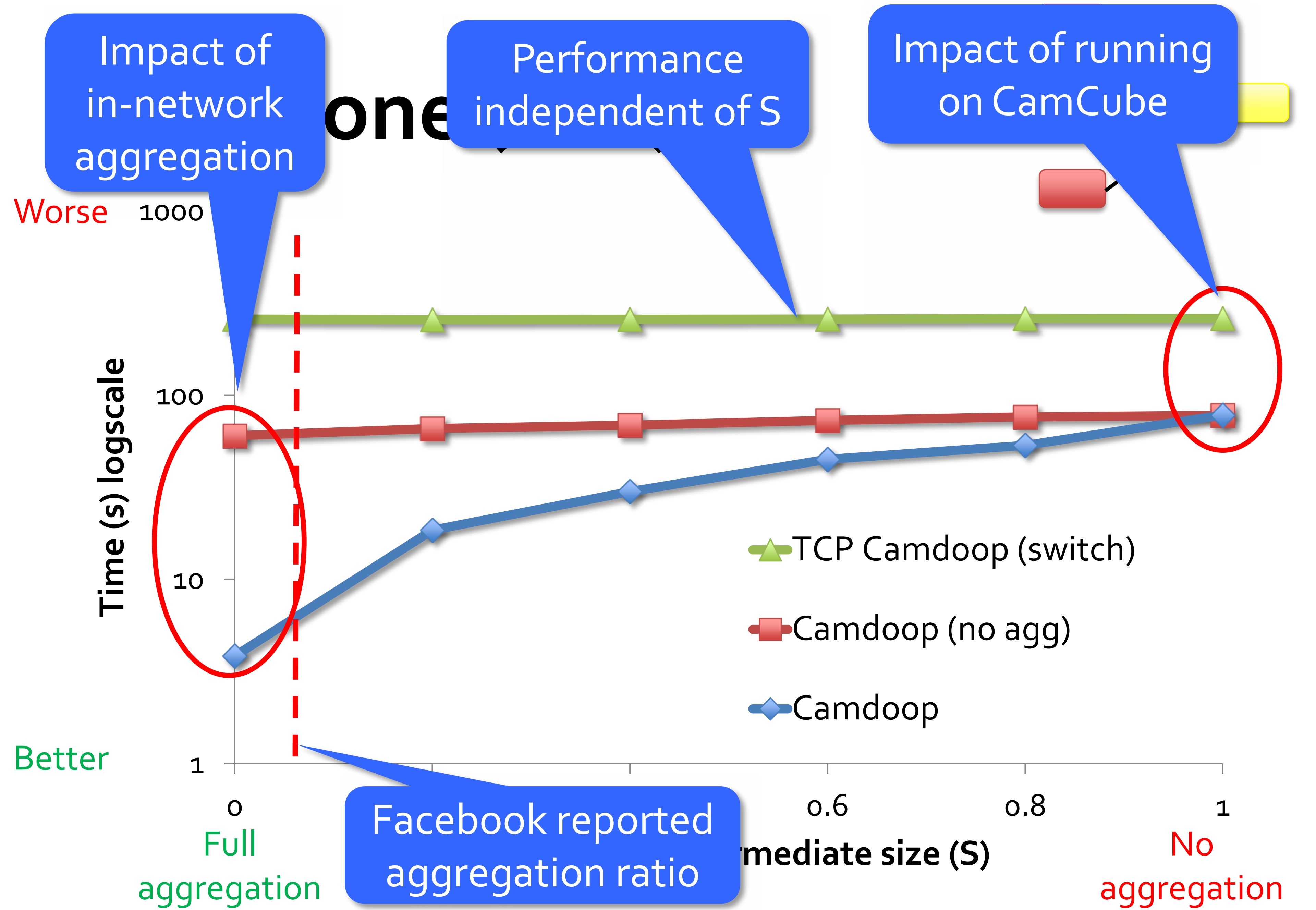


All-to-one (R=1)

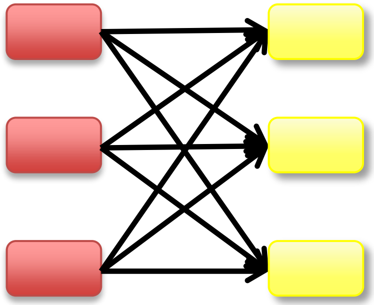

Worse
1000
100
10
1
Better
Time (s) logscale
TCP Camdoop (switch)
Camdoop (no agg)
Camdoop
0
0.2
0.4
0.6
0.8
1
Full aggregation
Output size/ intermediate size (S)
No aggregation

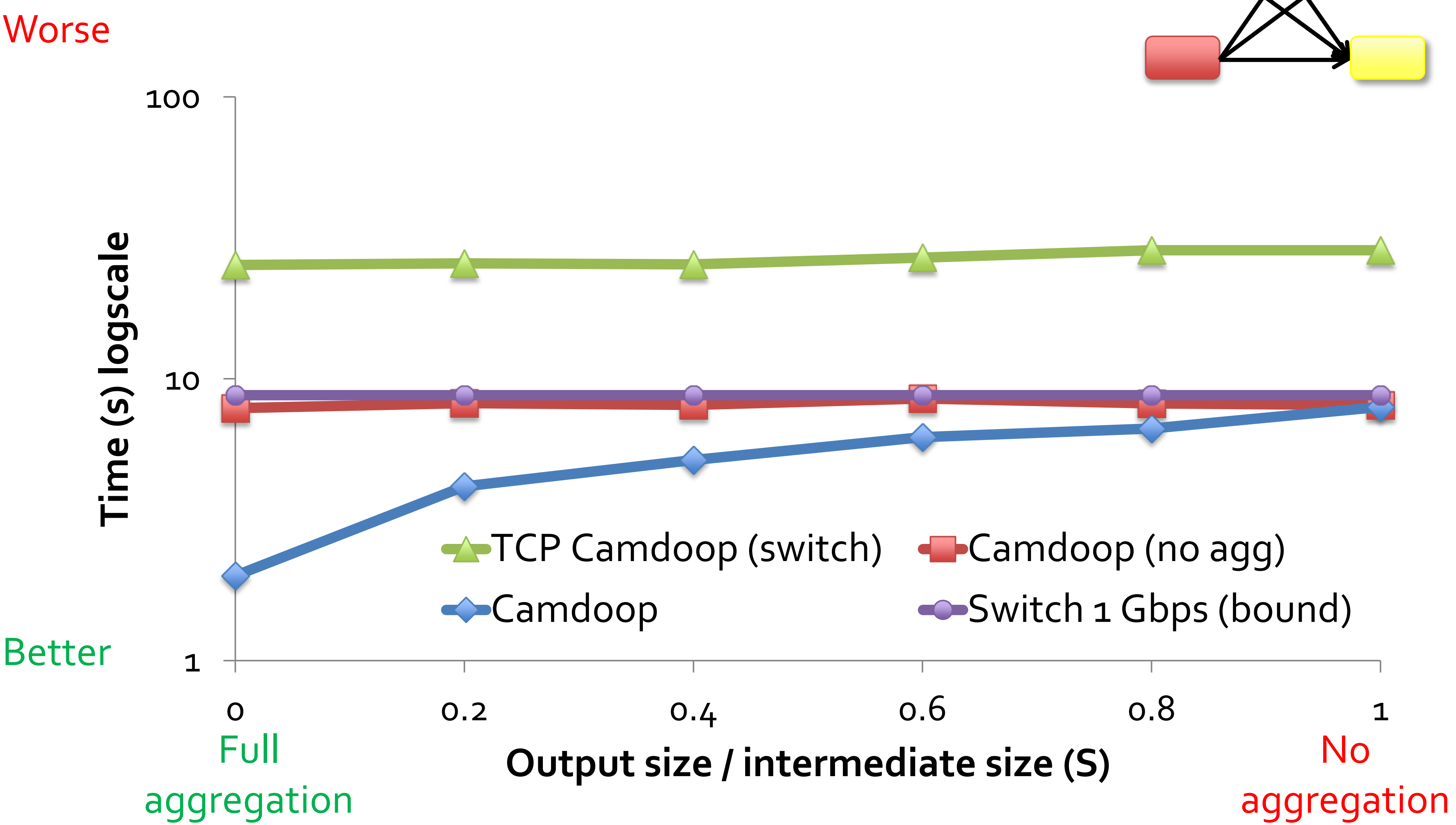


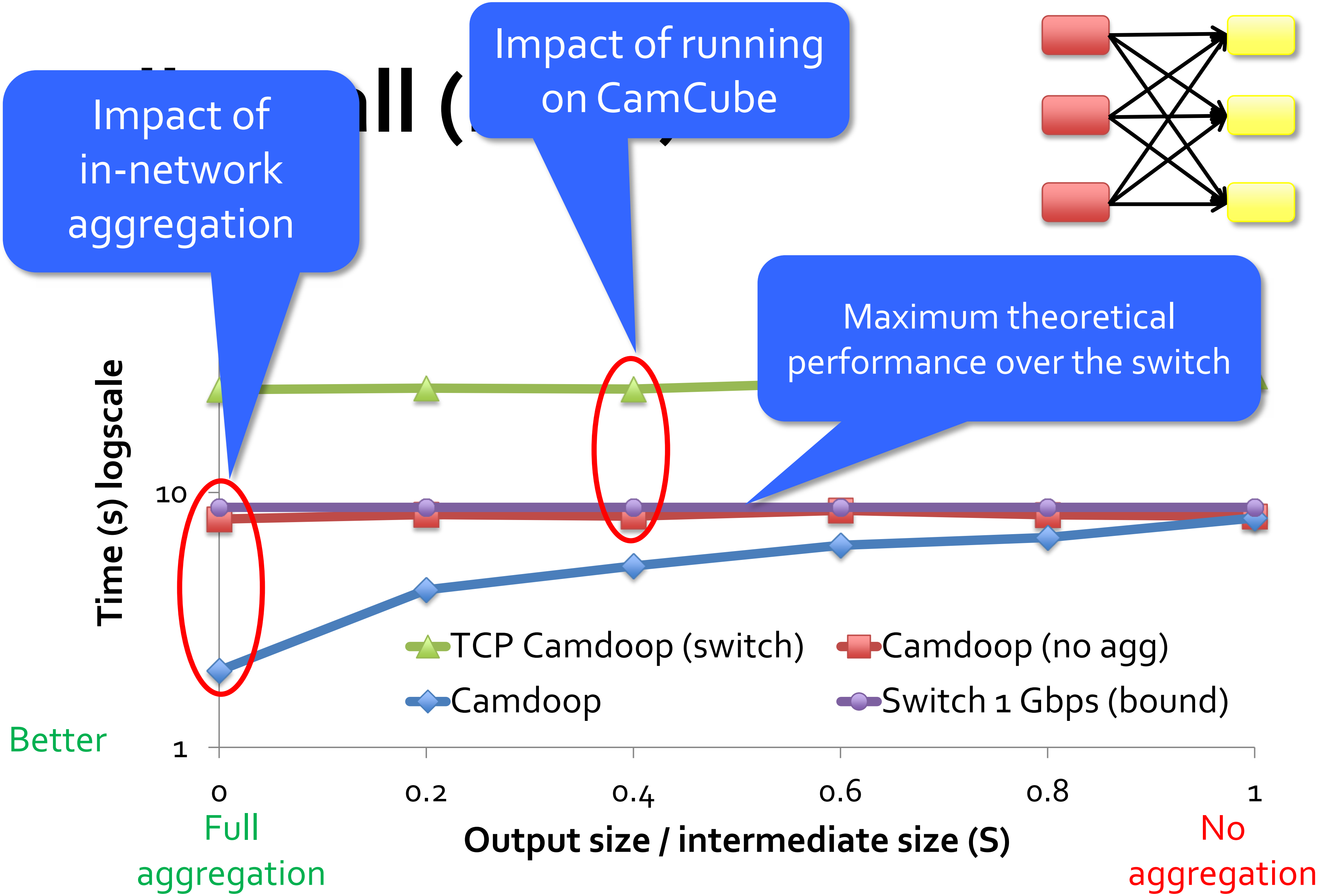
Impact of in-network aggregation
one
Performance independent of S
Impact of running on CamCube
Worse
Better
1000
100
10
1
Time (s) logscale
0
0.2
0.4
0.6
0.8
1
Output size/ intermediate size (S)
Full aggregation
No aggregation
TCP Camdoop (switch)
Camdoop (no agg)
Camdoop


Impact of in-network aggregation
Performance independent of S
Impact of running on CamCube
Worse
Better
1000
100
10
1
Time (s) logscale
TCP Camdoop (switch)
Camdoop (no agg)
Camdoop
0
0.6
0.8
1
Full aggregation
No aggregation
rmediate size (S)
Facebook reported aggregation ratio

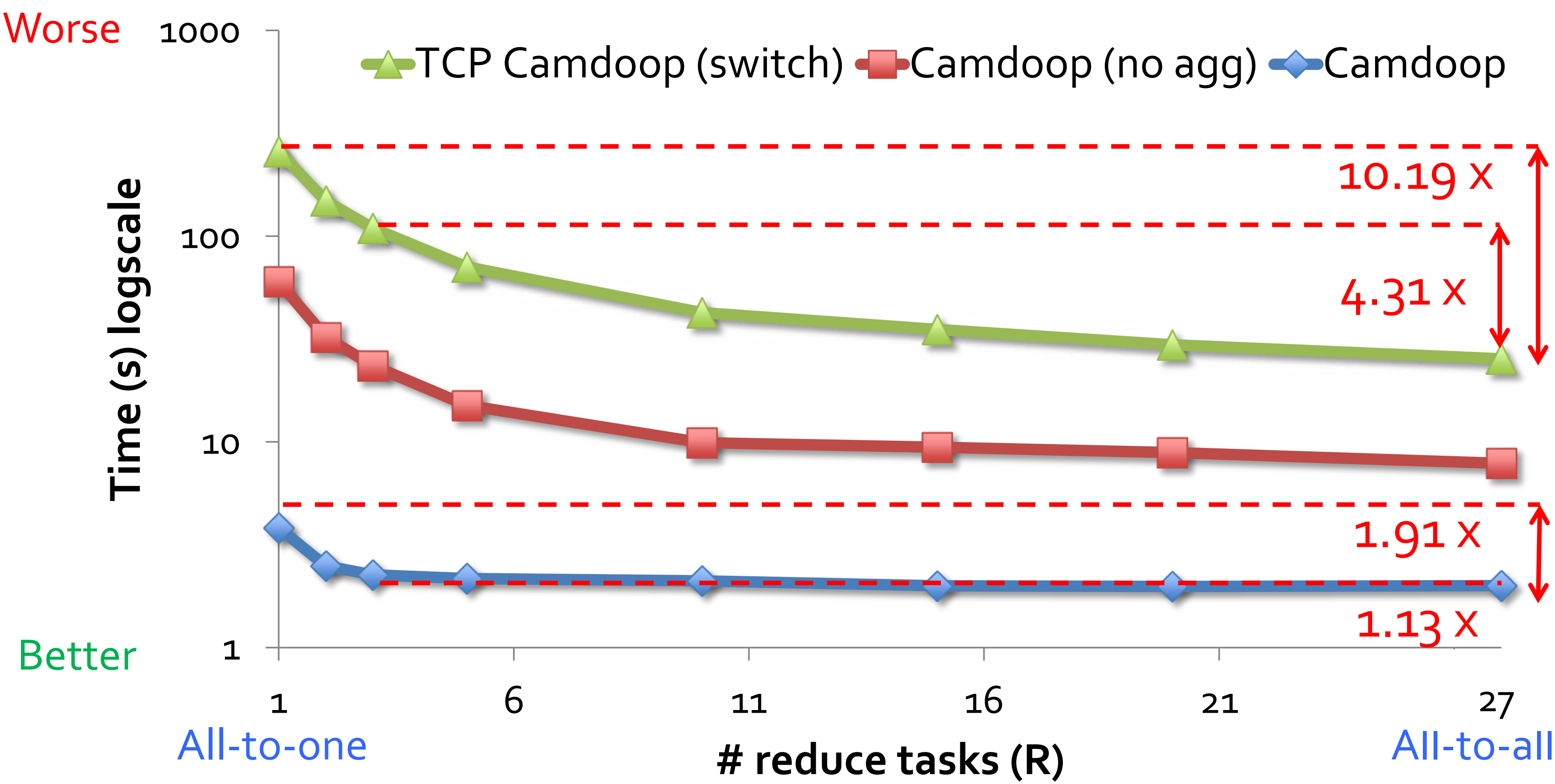
All-to-all (R=27)



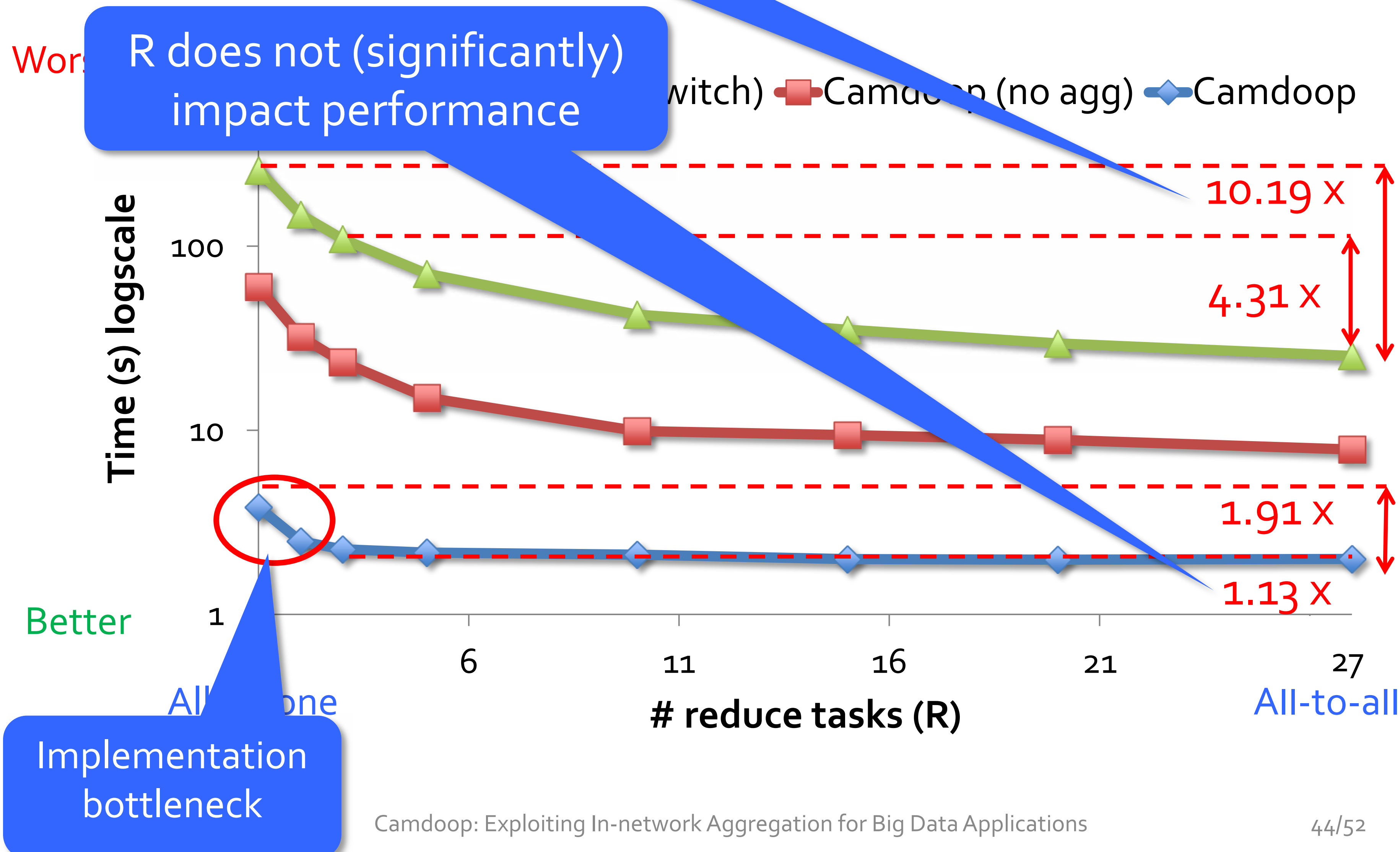



Impact of in-network aggregation
Impact of running on CamCube
Maximum theoretical performance over the switch
Time (s) logscale
10
1
Better
0
0.2
0.4
0.6
0.8
1
Full aggregation
Output size / intermediate size (S)
No aggregation
TCP Camdoop (switch)
Camdoop (no agg)
Camdoop
Switch 1 Gbps (bound)

Number of reduce tasks (S=0)



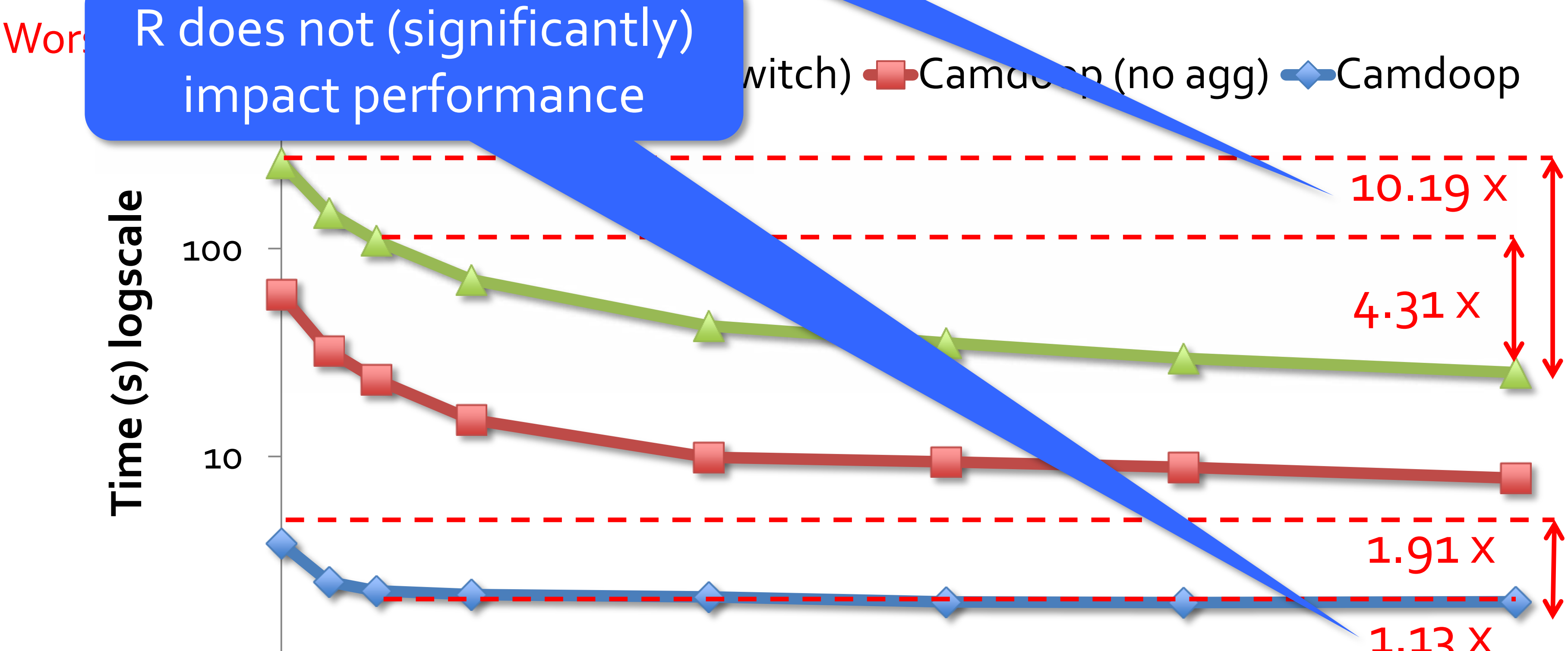
Numbe ce tasks (S=0)



Numbe… ce tasks (S=0)

Performance depends on R

R does not (significantly) impact performance

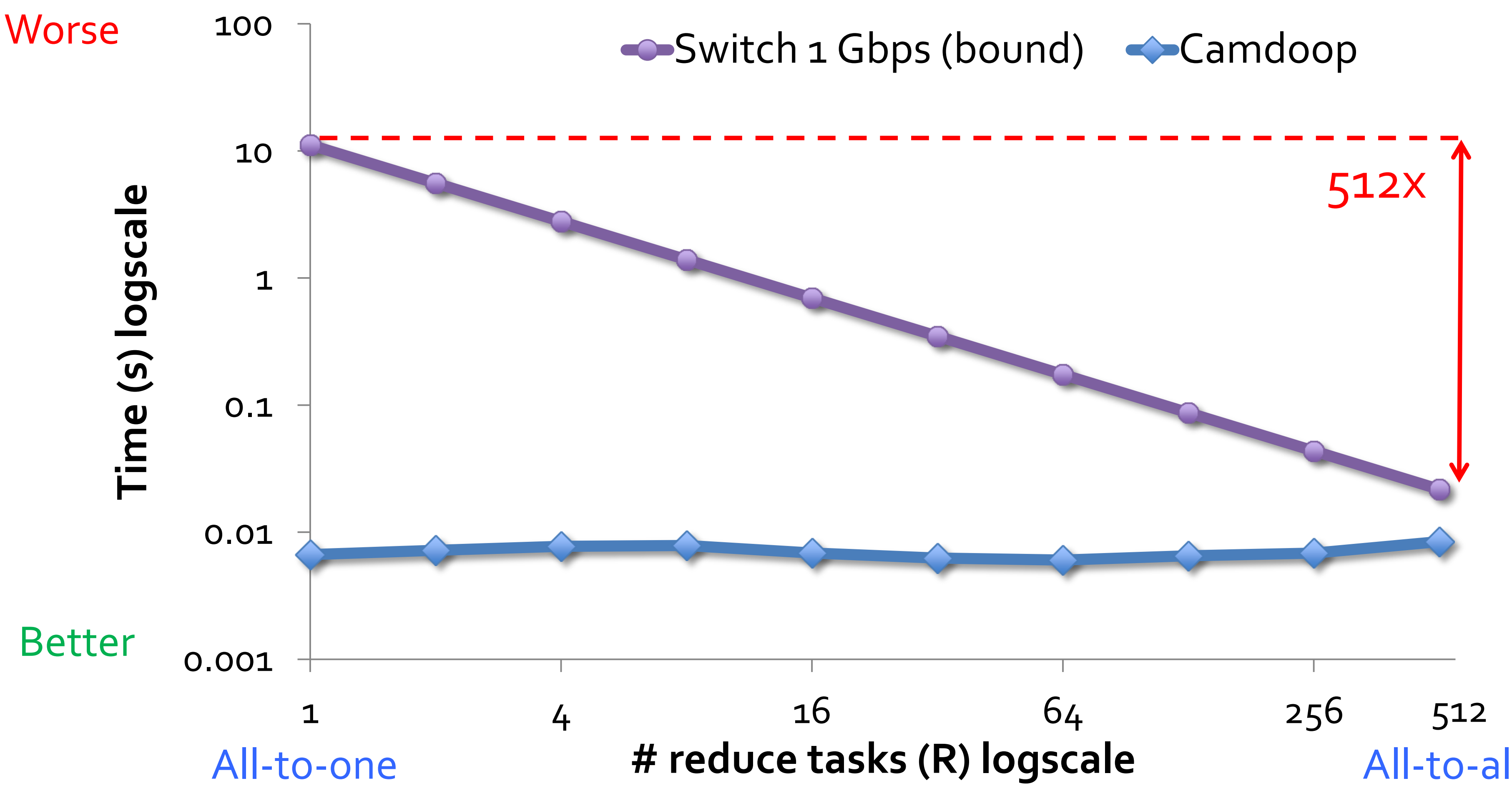


All resources are used even when R = 1

Camdoop decouples the job execution time from the number of output files generated

Behavior at scale (simulated)

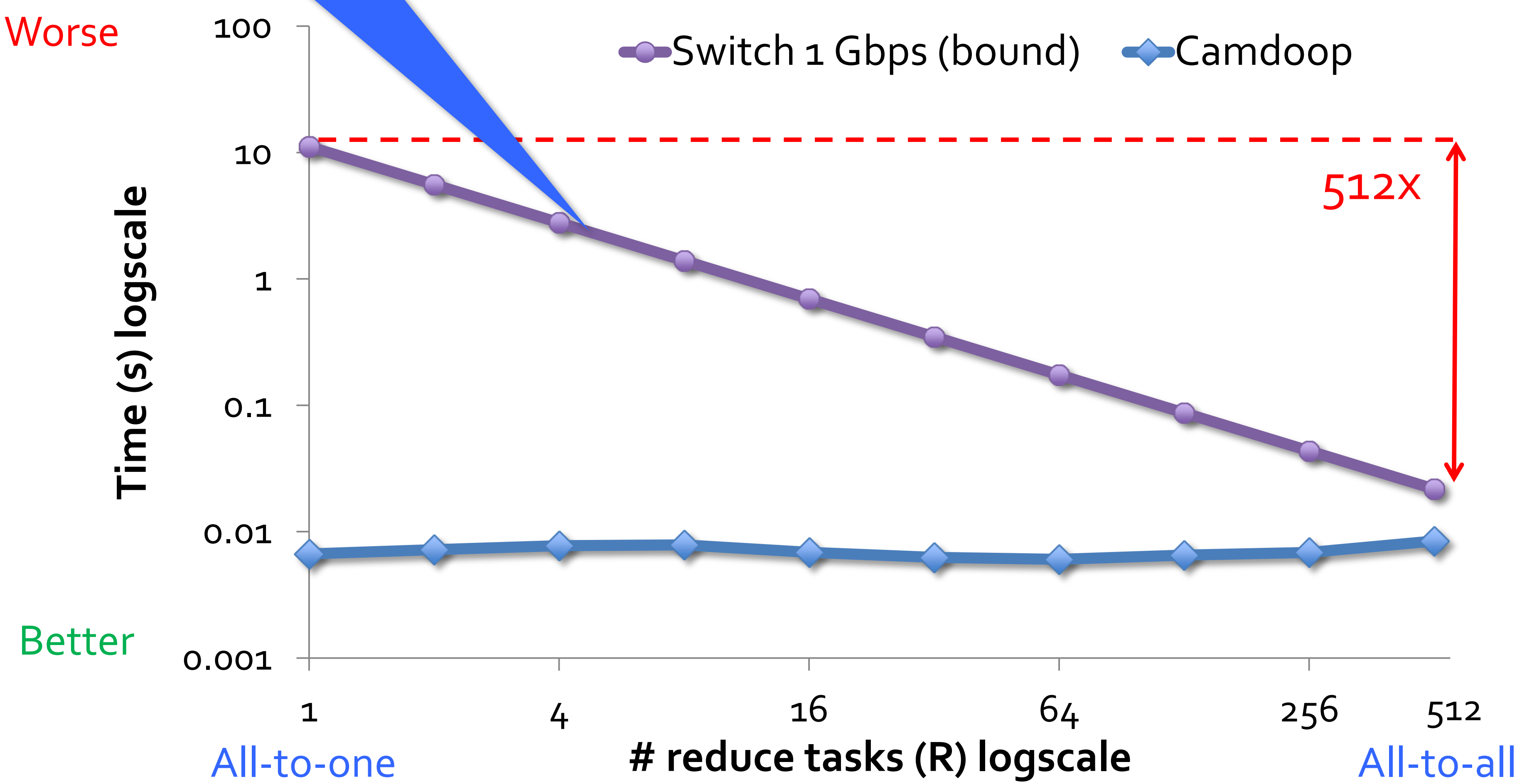
N=512, S= 0



...or at scale (simulated)

This assumes full-bisection bandwidth

N=512, S= 0

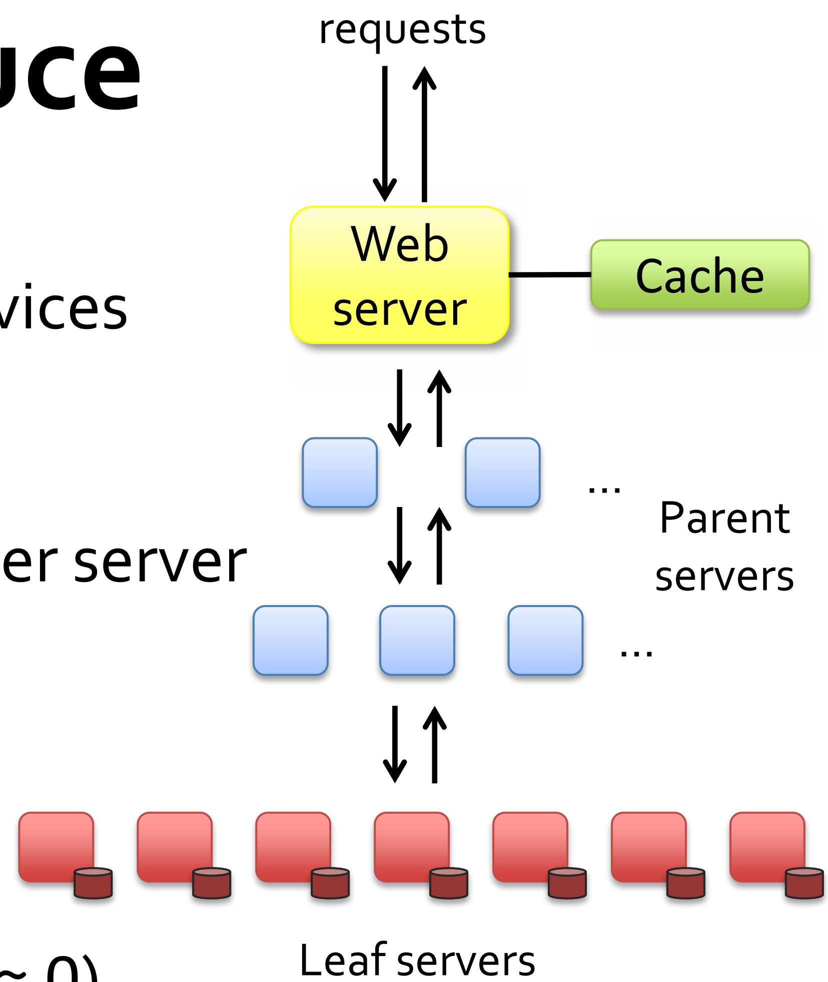


Beyond MapReduce

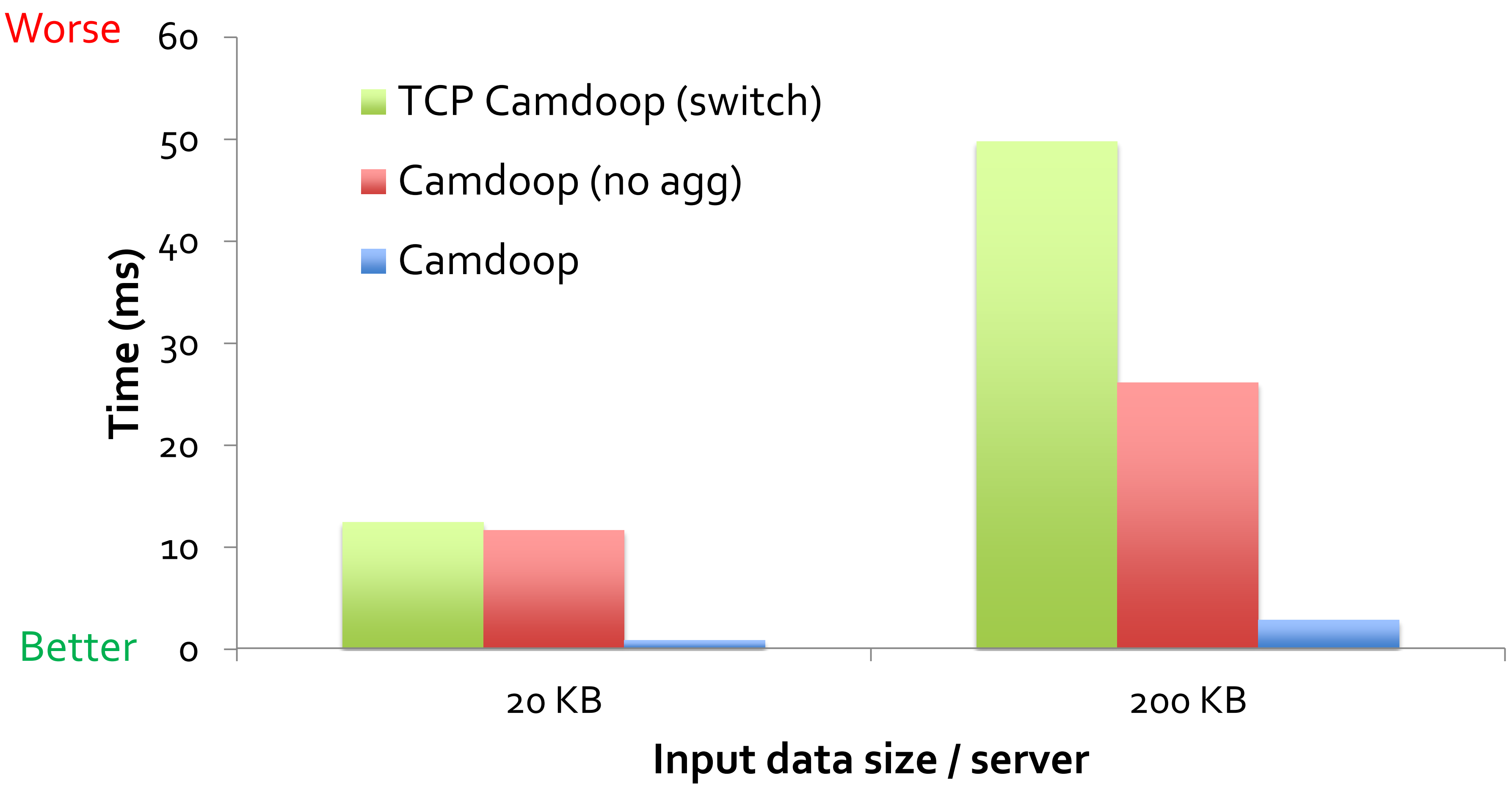
- *More experiments (failures, multiple jobs,…) in the paper*

Beyond MapReduce

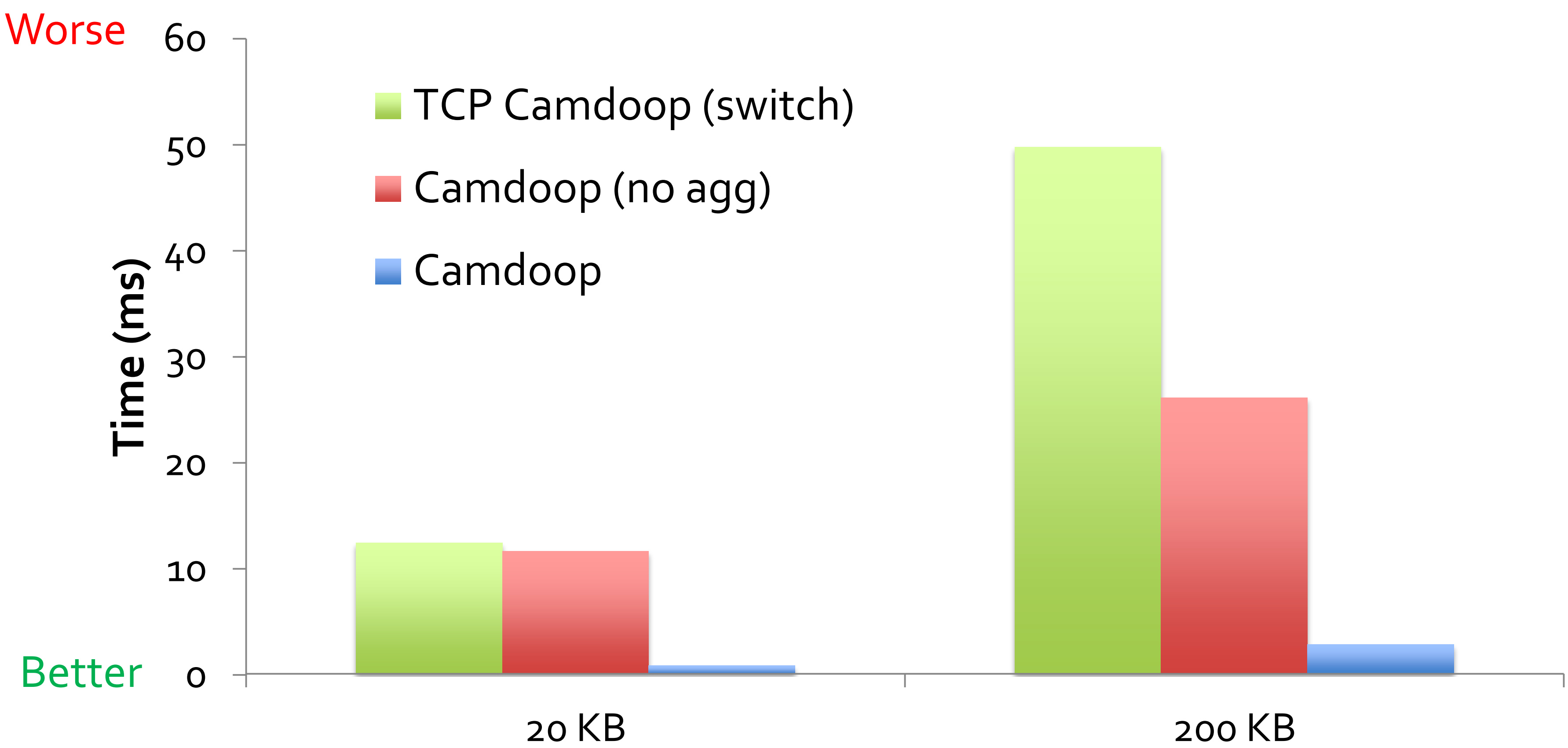
- The partition-aggregate model also common in interactive services
  - e.g., Bing Search, Google Dremel
- Small-scale data
  - 10s to 100s of KB returned per server
- Typically, these services use one reduce task (R=1)
  - Single result must be returned to the user
- Full aggregation is common (S ≈ 0)
  - E.g., N servers generate their best *k* responses each and the final result contains the best *k* responses



Small-scale data (R=1, S=0)



Small-scale data (R=1, S=0)



In-network aggregation can be beneficial also for (small-scale data) interactive services

Conclusions

- Camdoop
  - Explores the benefits of in-network processing by running combiners within the network
  - No change in the programming model
  - Achieves lower shuffle and reduce time
  - Decouples performance from the # of output files

- A final thought: *how would Camdoop run on this?*
  - AMD SeaMicro – a 512-core cluster for data centers using a 3D torus
  - Fast interconnect: 5 Gbps / link