Overcoming Data-Locality: an In-Memory Runtime File System with Symmetrical Data Distribution

Alexandru Uta, Andreea Sandu, **Thilo Kielmann**
a.uta@vu.nl, a.sandu@vu.nl, Thilo.Kielmann@vu.nl

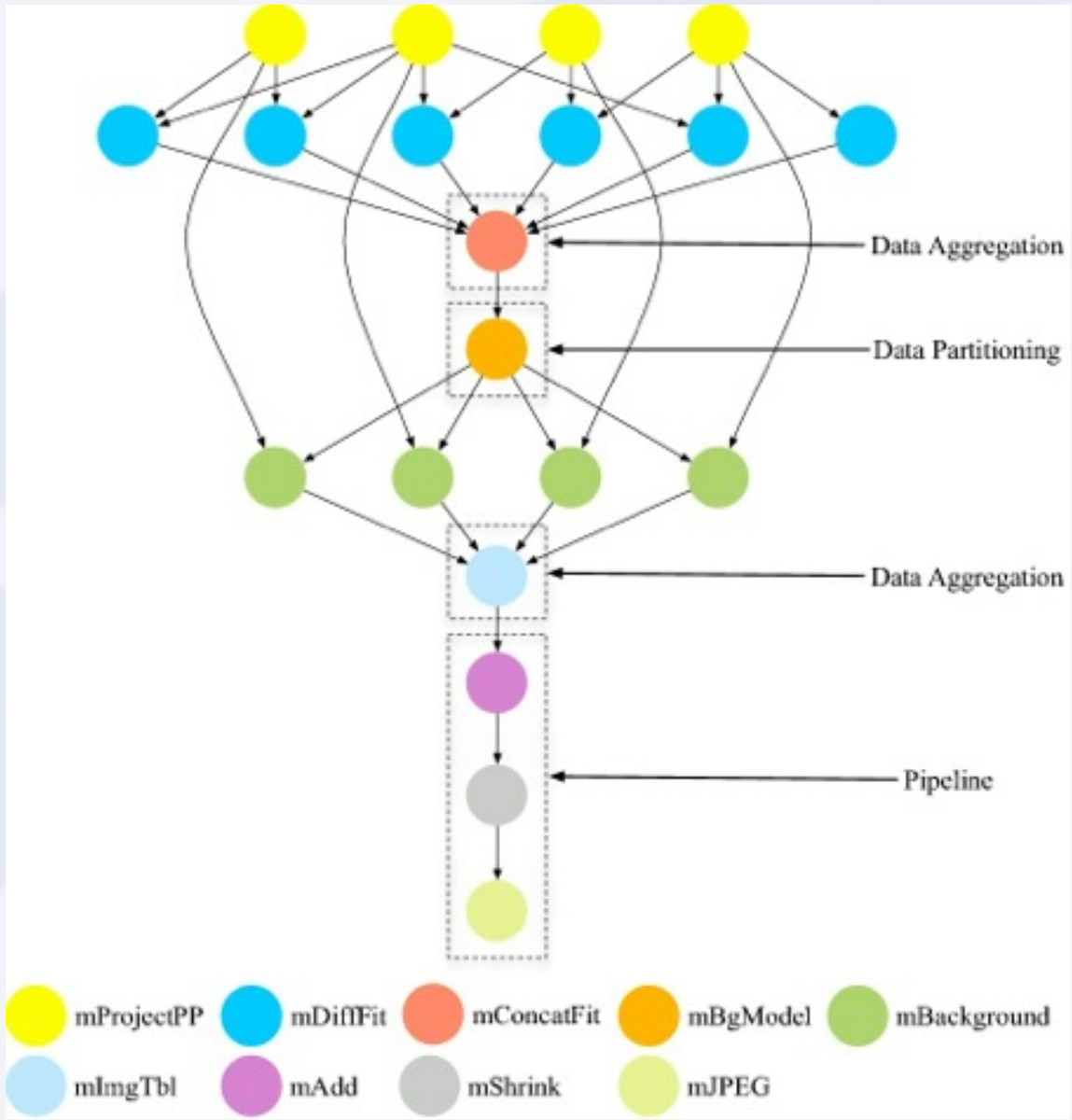
“Data Locality Considered Harmful”

This work has just been reviewed by IEEE Cluster 2014:

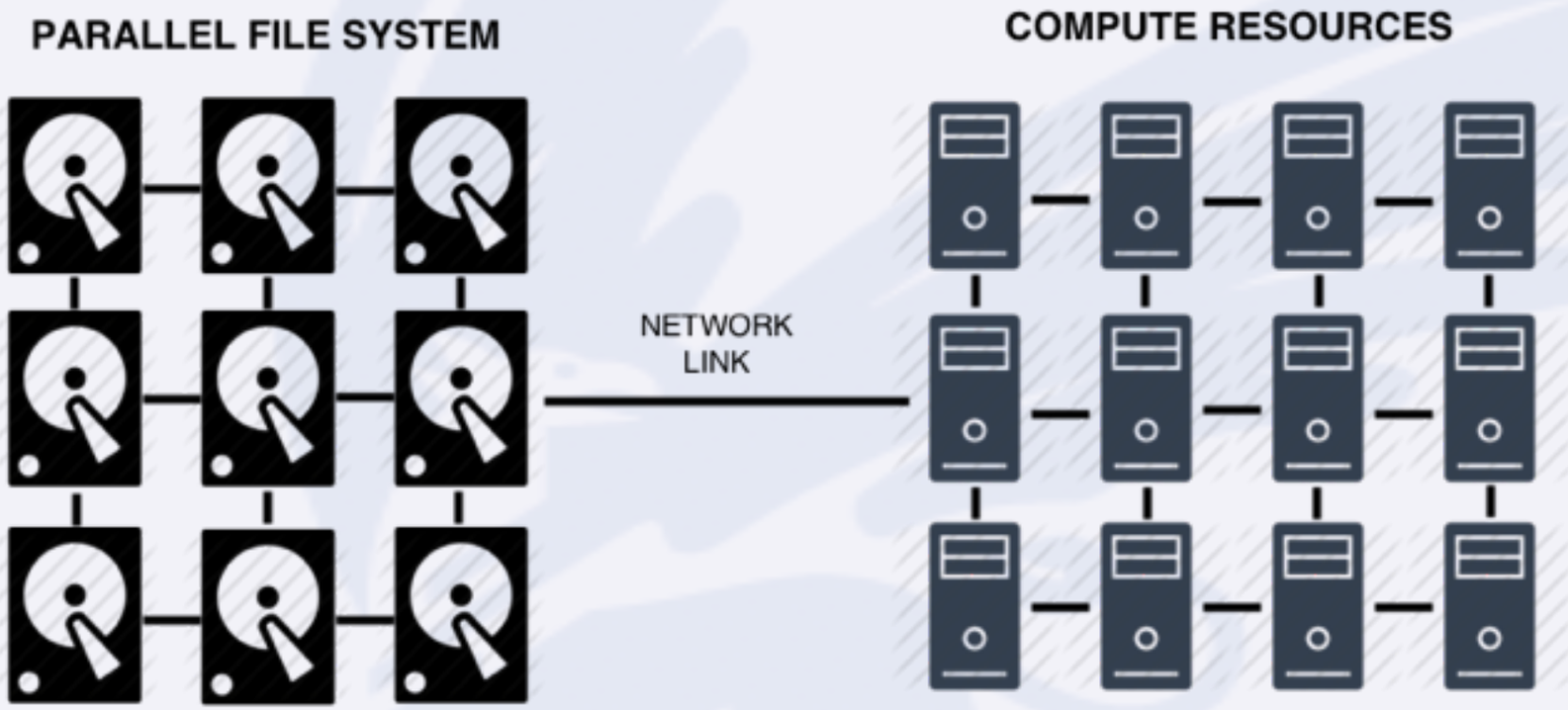
- 2x “Nomination for best paper award”
- 2x “Reject”

Many-Task Computing

- application processes that communicate through an underlying (distributed/shared) file system
- examples: workflows, parameter sweeps, parallel scripts
- individual tasks typically run in between 0.1 and 10 s
- write-once, read-many access pattern



Cluster File System Setup



Performance limited by:

- network speed/latency
- disk/SSD speed/latency

In-memory Scratch File Systems

- MTC performance depends on file I/O
- There is plenty of RAM in the compute nodes
- There is plenty of network in between
- Why not store the intermediate files in a memory FS?
  - No persistence needed here…
  - Co-locate compute and storage facilities (on all compute nodes)

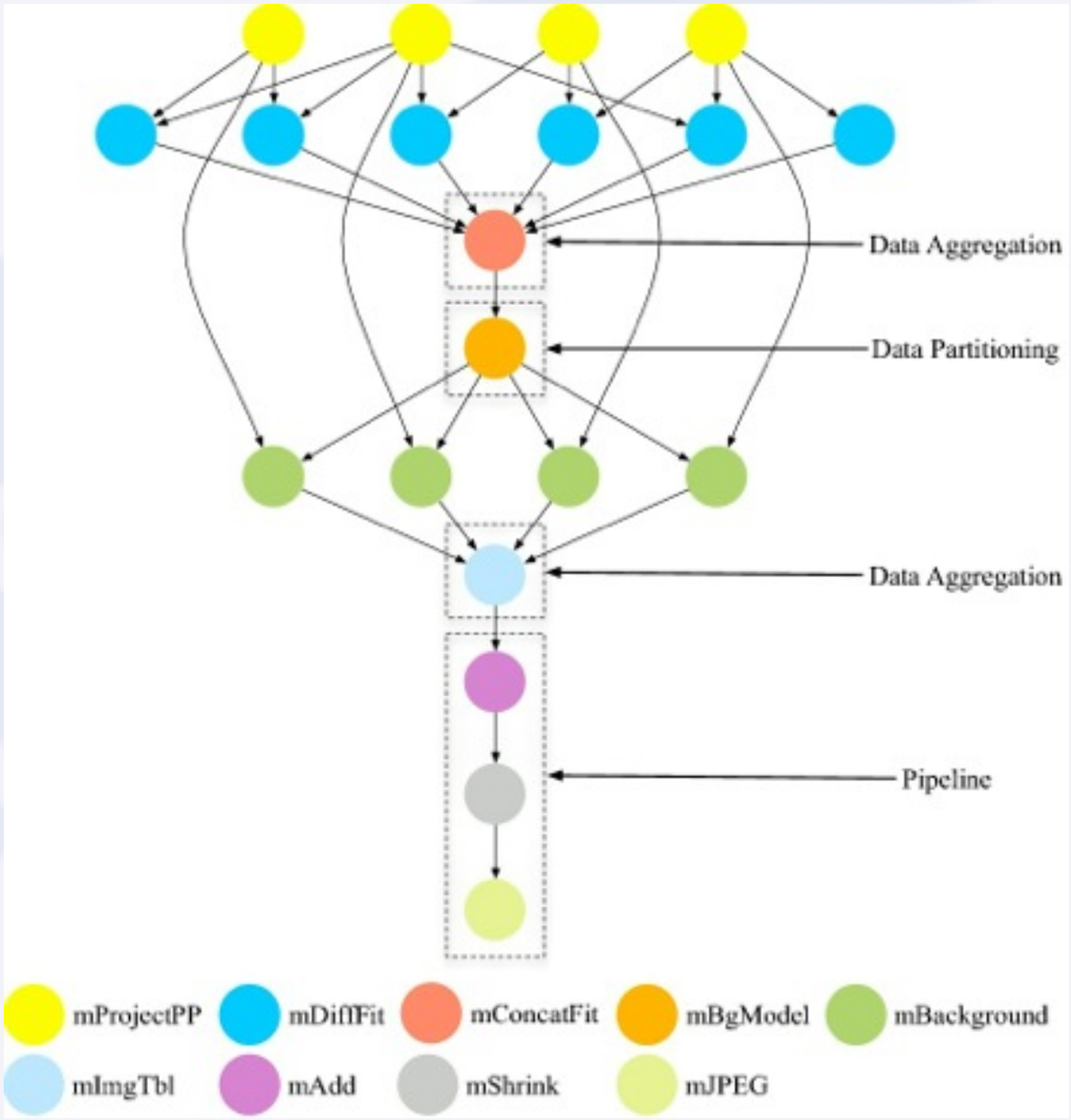
State-of-the-art: AMFS [1]

- In-memory distributed file system for MTC
- Optimizes for data-locality:
  - write into local memory
  - schedule tasks where data resides
  - copy remote file(s) before execution
    - if no local data available
    - if input files from more than one node needed

[1] Zhao Zhang, Daniel S Katz, Timothy G Armstrong, Justin M Wozniak, and Ian Foster. *Parallelizing the execution of sequential scripts*. SC13

Example Workflow: Montage

- Data is created at one or few nodes
- Results are accumulated at one node
- Data aggregation and partitioning in between
- Tasks often read more than one input file

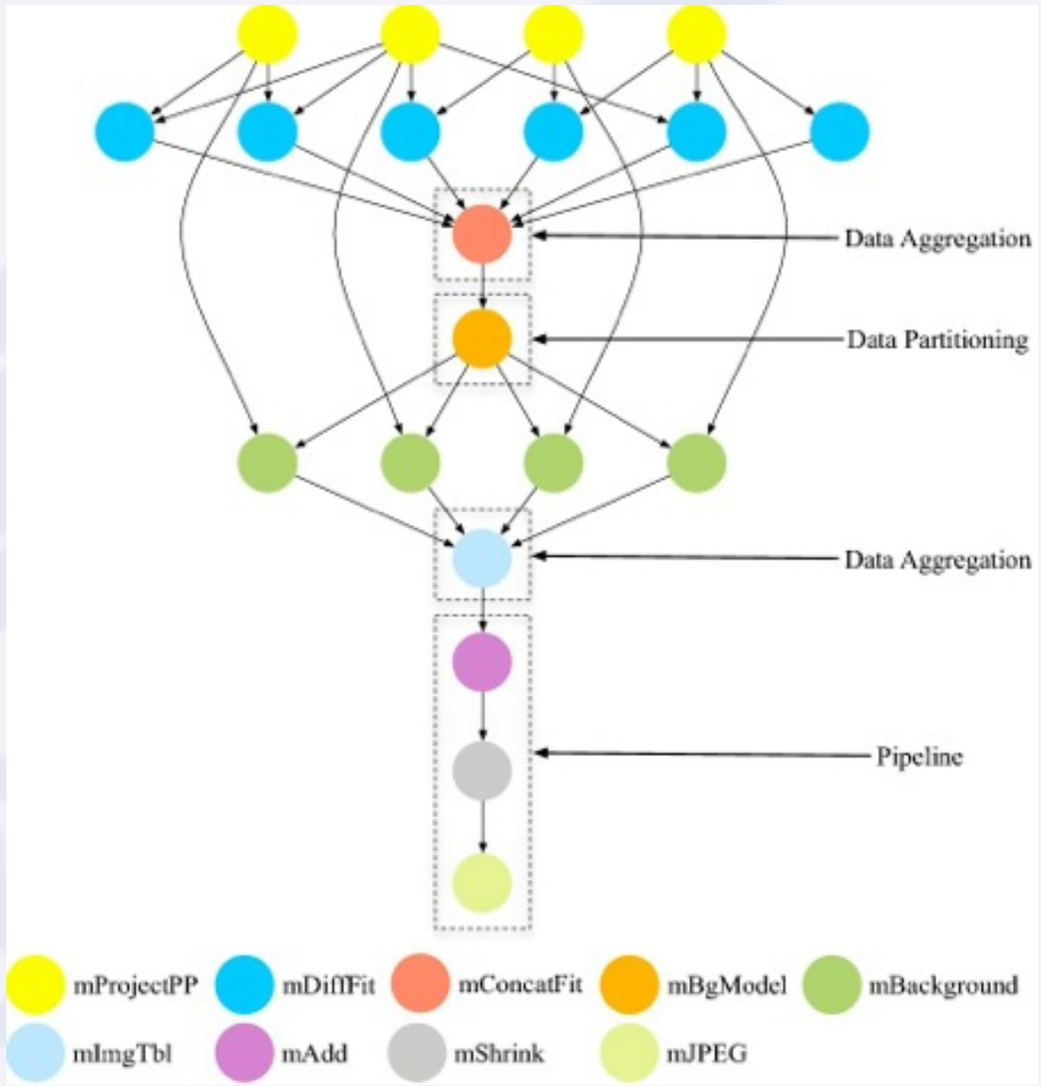


Data Locality-based Scheduling

- Write into local memory:
  - severe data imbalance
- Read from local memory:
  - copy if needed

- Schedule tasks where data resides
  - only few nodes have data in the beginning
- Otherwise, copy remote files before execution
  - create lots of copies of the data
  - for data aggregation, all data needs to be merged onto the aggregating node
    - another imbalance

- Potential problems:
  - slowdown because of data copies
  - failing execution when memory gets exhausted



Overcoming Data Locality

- Remember: the network is fast (QDR IB: 32Gb/s)
- Idea:
  - split files in equal-sized stripes (e.g., 512 KB)
  - store stripes equally distributed across all nodes, based on a hashing function
  - remotely read only stripes that are needed (cache them)

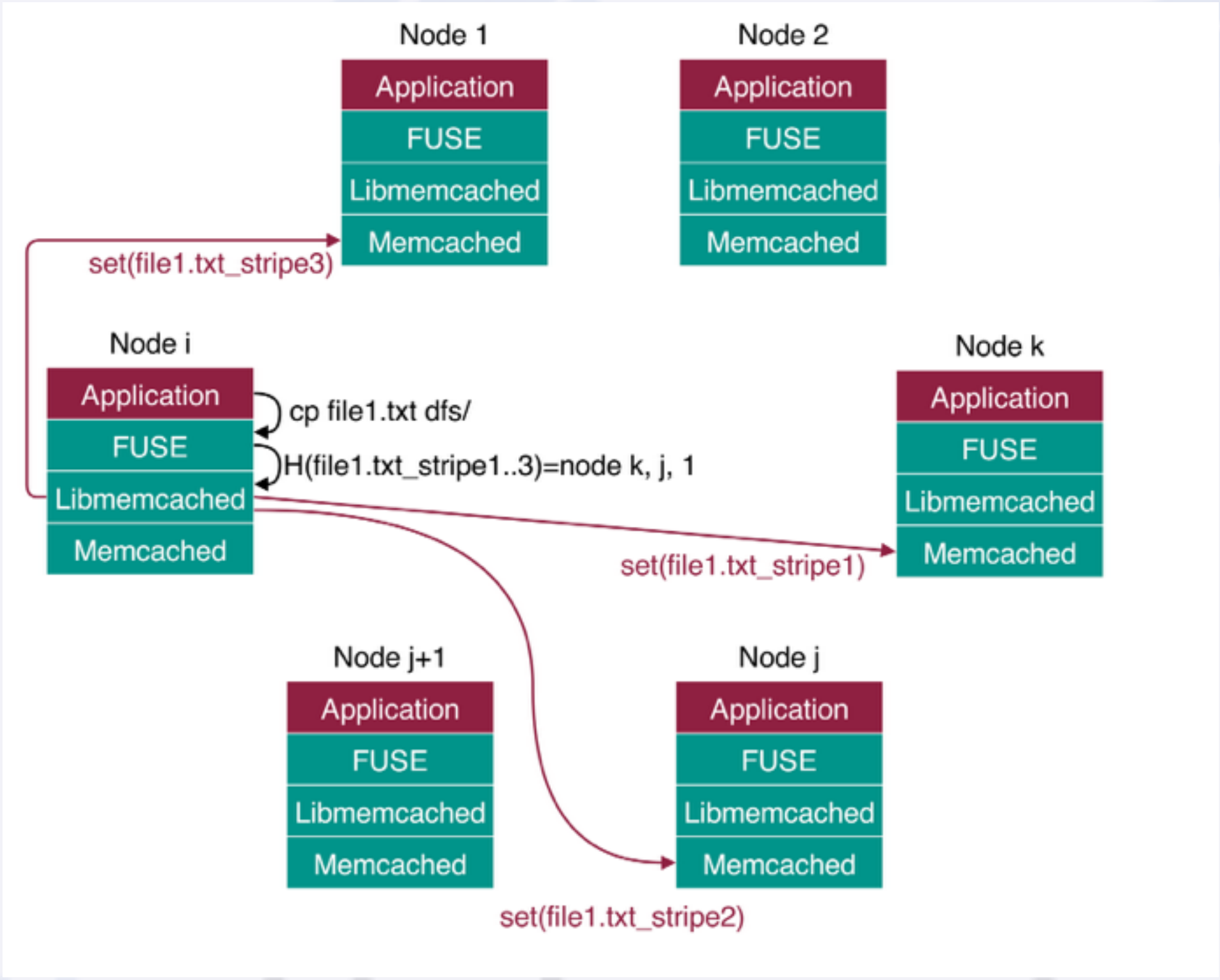
- Semantics: write once / read multiple times
  - that is what MTC/workflows do

MemFS

- Run memcached key-value store on all nodes
- Run FUSE file system and libmemcached to access the memcached servers

- Lots of system optimisation (not shown in this talk)
  - multi-threaded writing and reading
  - buffering and prefetching (for sequential r/w)
  - ...

MemFS



Evaluation

- All experiments run on DAS4 (www.cs.vu.nl/das4/)
  - 72 nodes
  - dual-quad-core Intel E5620 2.4 GHz
  - 24 GB memory
  - QDR Infiniband (32Gb/s)
  - 1GB Ethernet
  - Tests use Infiniband (IPoIB ~ 1.1 GB/s)

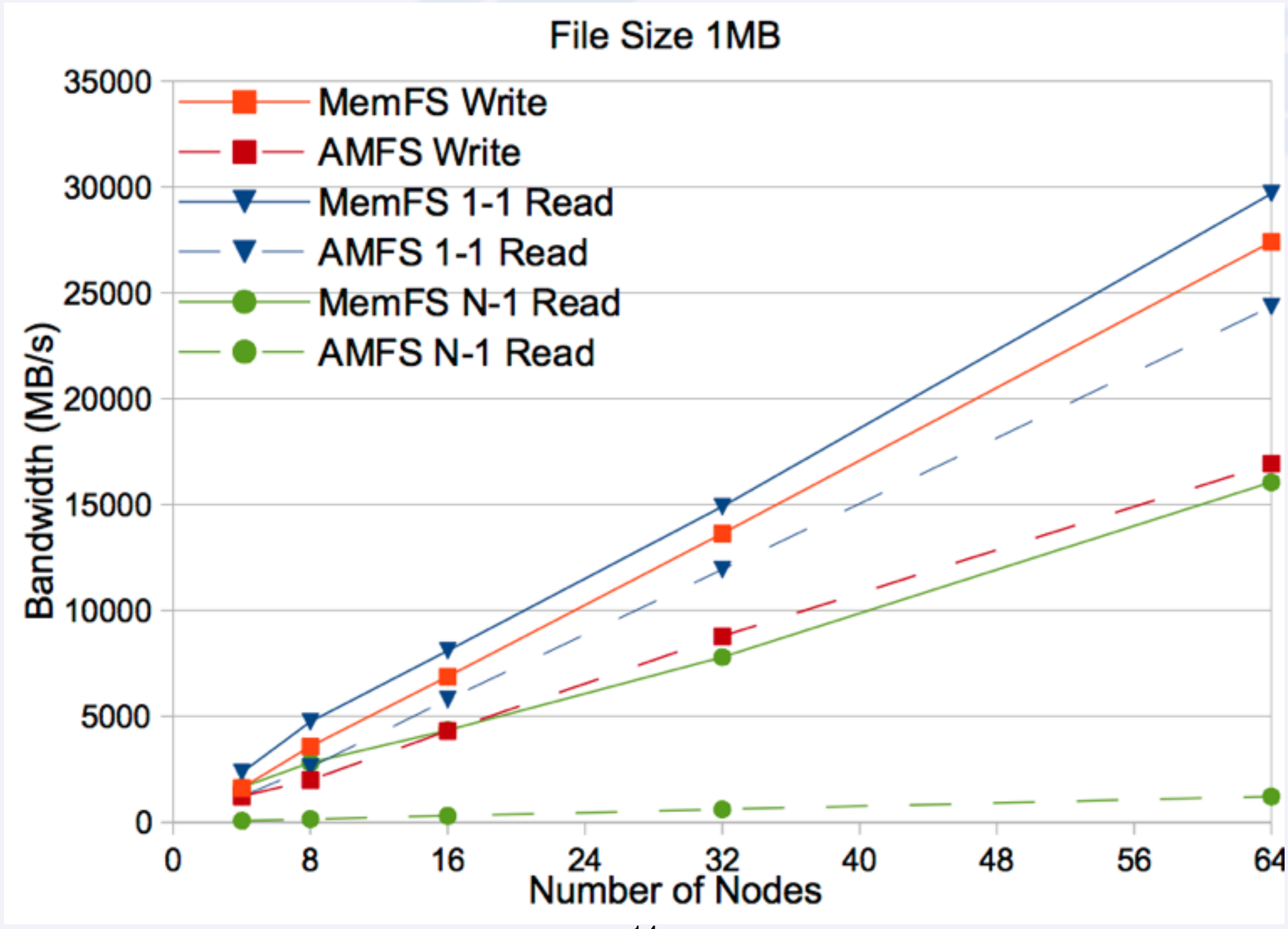
- Micro benchmarks: MTC Envelope using IOZone
- Applications:
  - Montage astronomical image mosaic
  - BLAST gene sequence alignment

MTC Envelope [2]

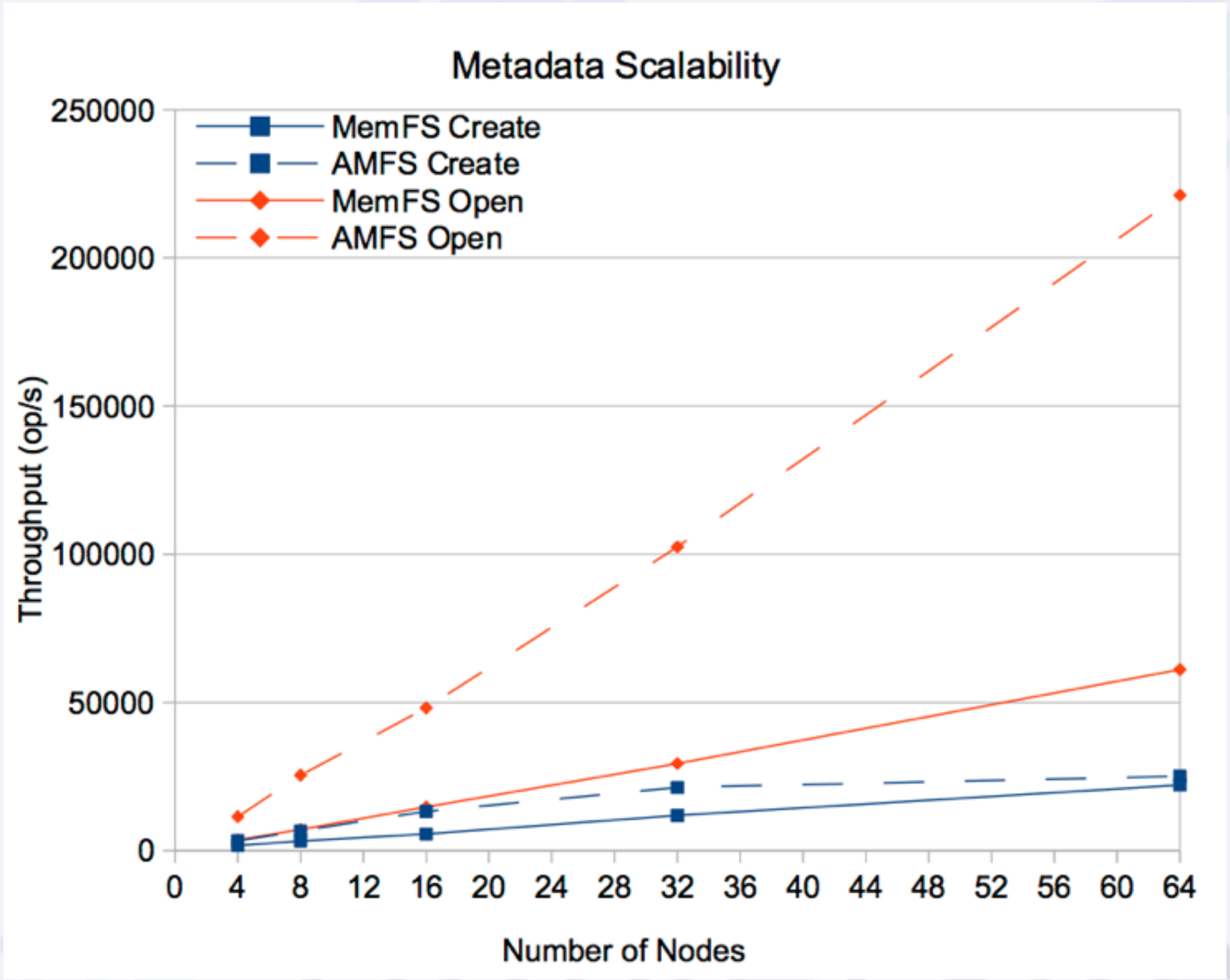
- set of metrics that assess a system's capability to run MTC applications:
  - 1-1 write bandwidth / throughput
  - 1-1 read bandwidth / throughput
  - N-1 read bandwidth / throughput
  - metadata throughput: open, create

[2] Zhang, Z., Katz, D. S., Wilde, M., Wozniak, J. M., & Foster, I. MTC Envelope: Defining the capability of large scale computers in the context of parallel scripting applications. HPDC 13

MTC Envelope Bandwidth



MTC Envelope Metadata

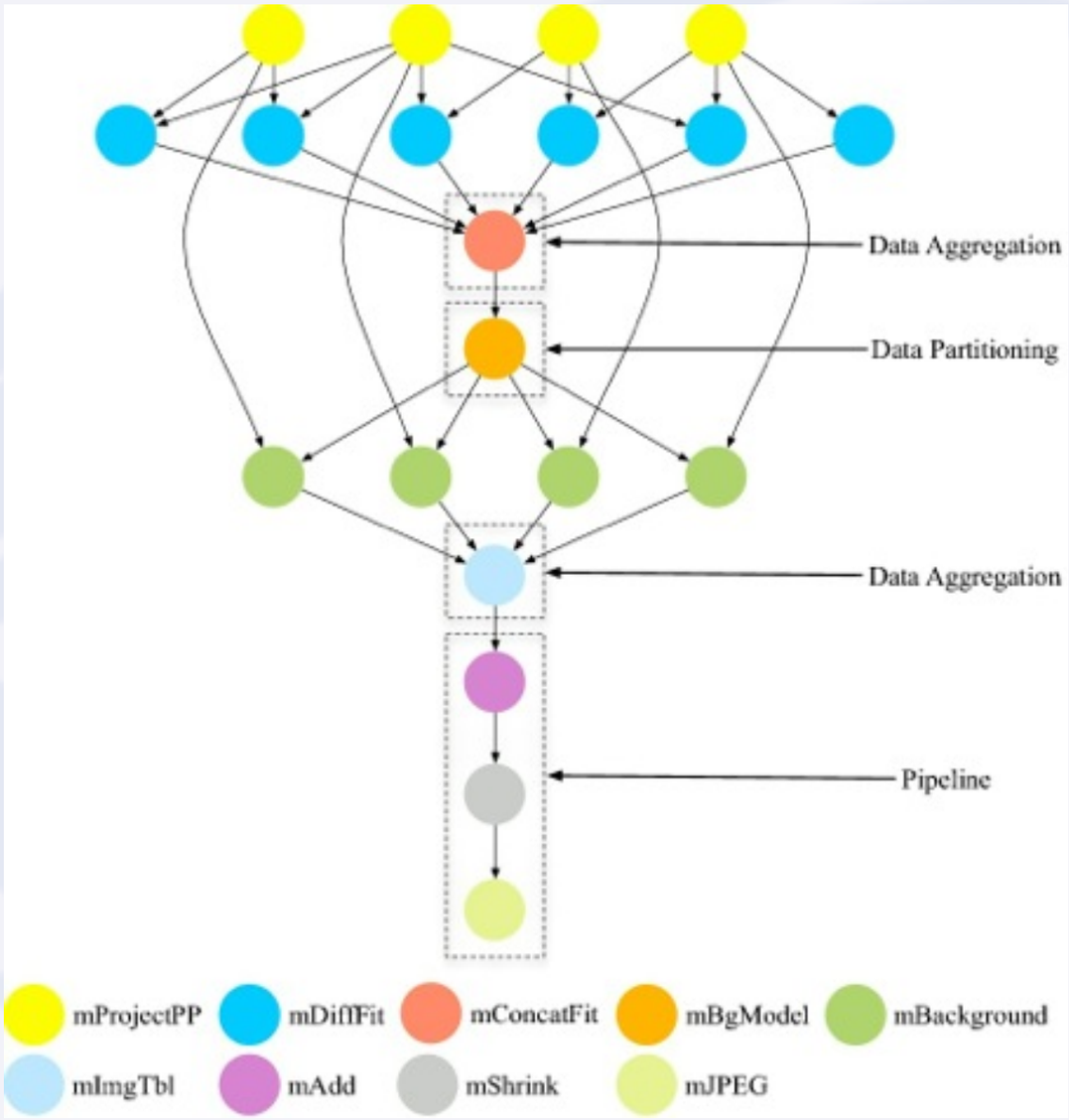


Montage Workflow

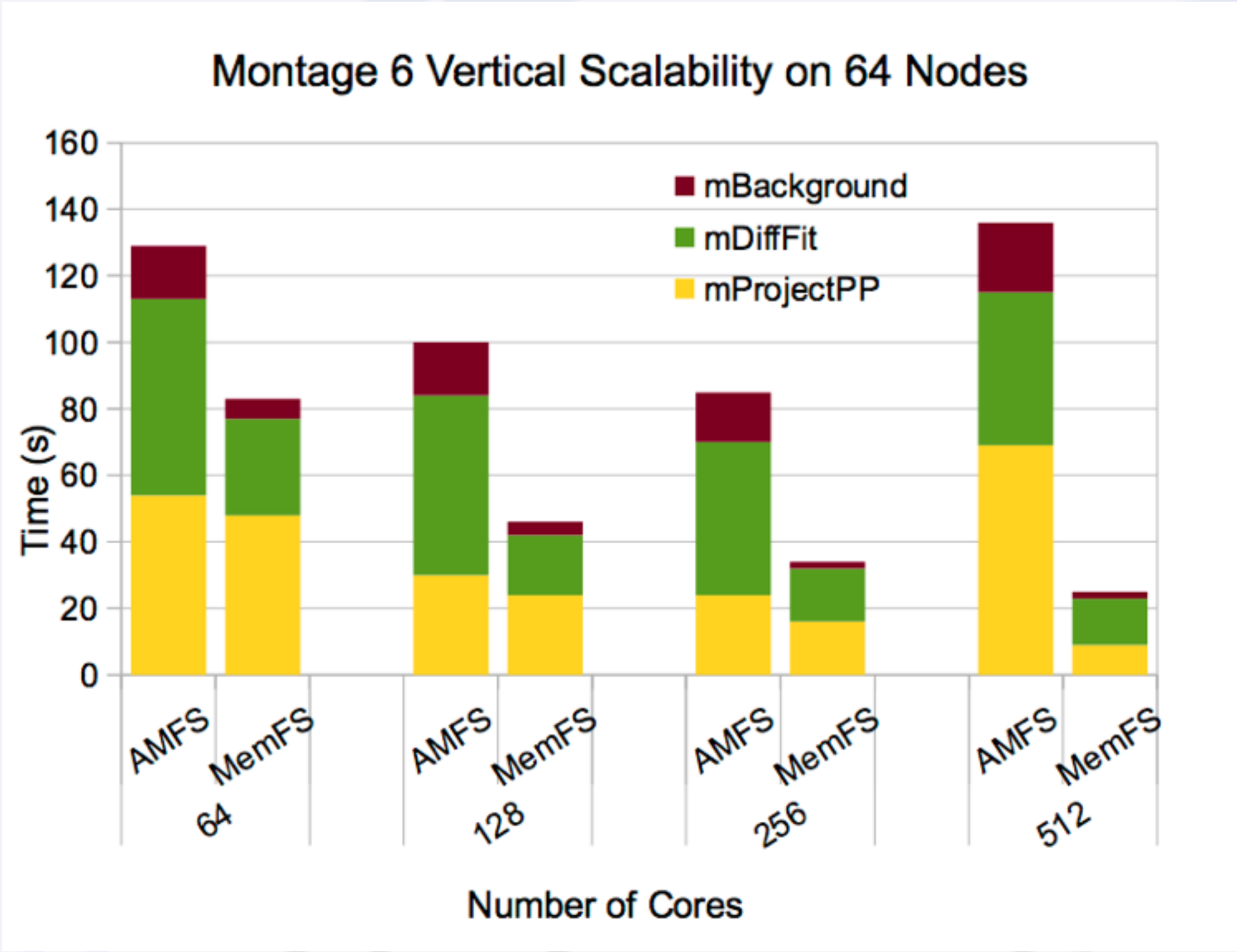
- Astronomical image mosaic engine

- Our use cases:
  - 6x6 (5,9 GB input),
  - 12x12 (20 GB input) degree Montage instance

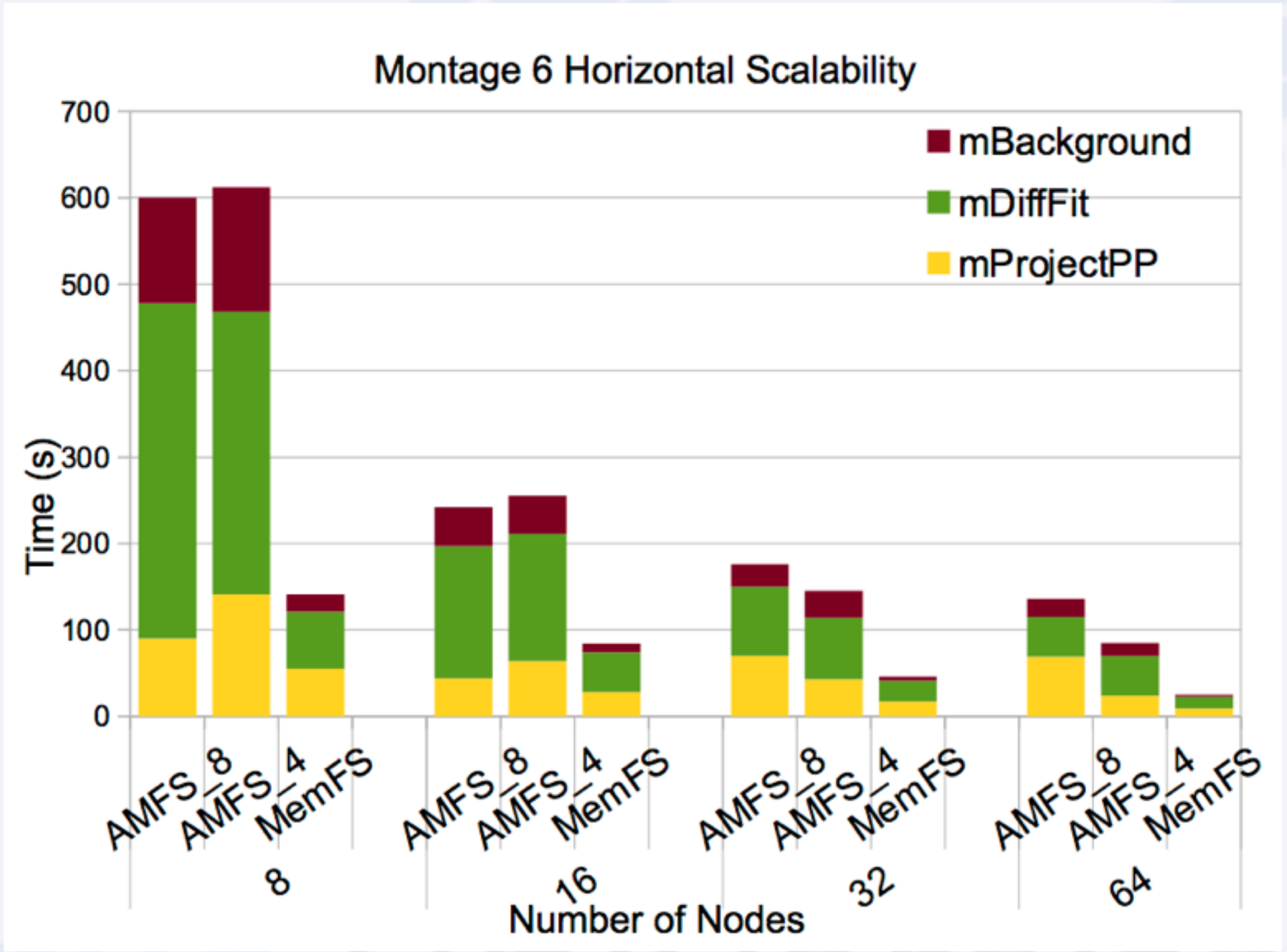
- We assessed performance, memory usage and vertical/ horizontal scalability



Montage 6 Results



Montage 6 Results



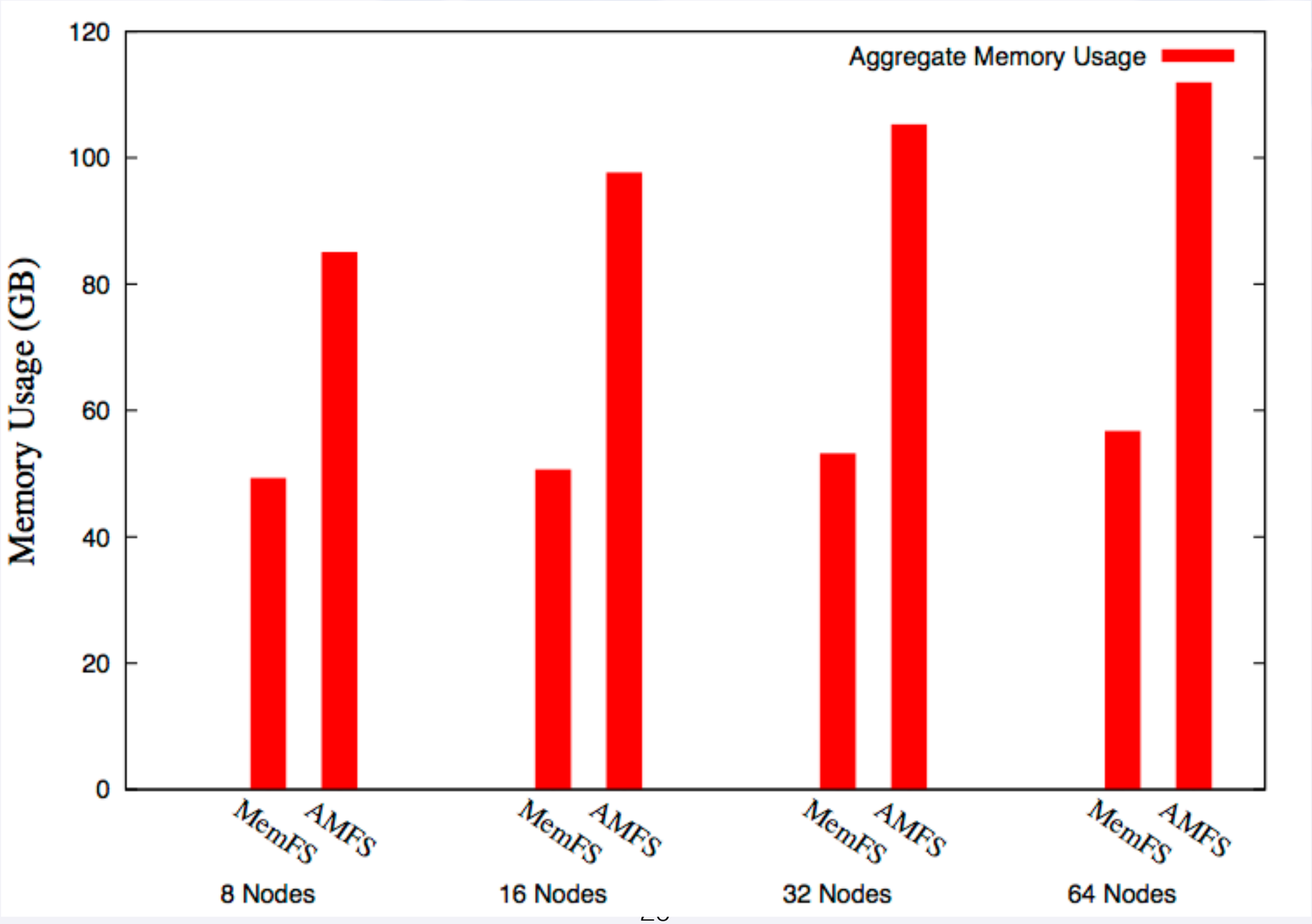
Montage 6 Results

AMFS Memory Distribution for Montage 6

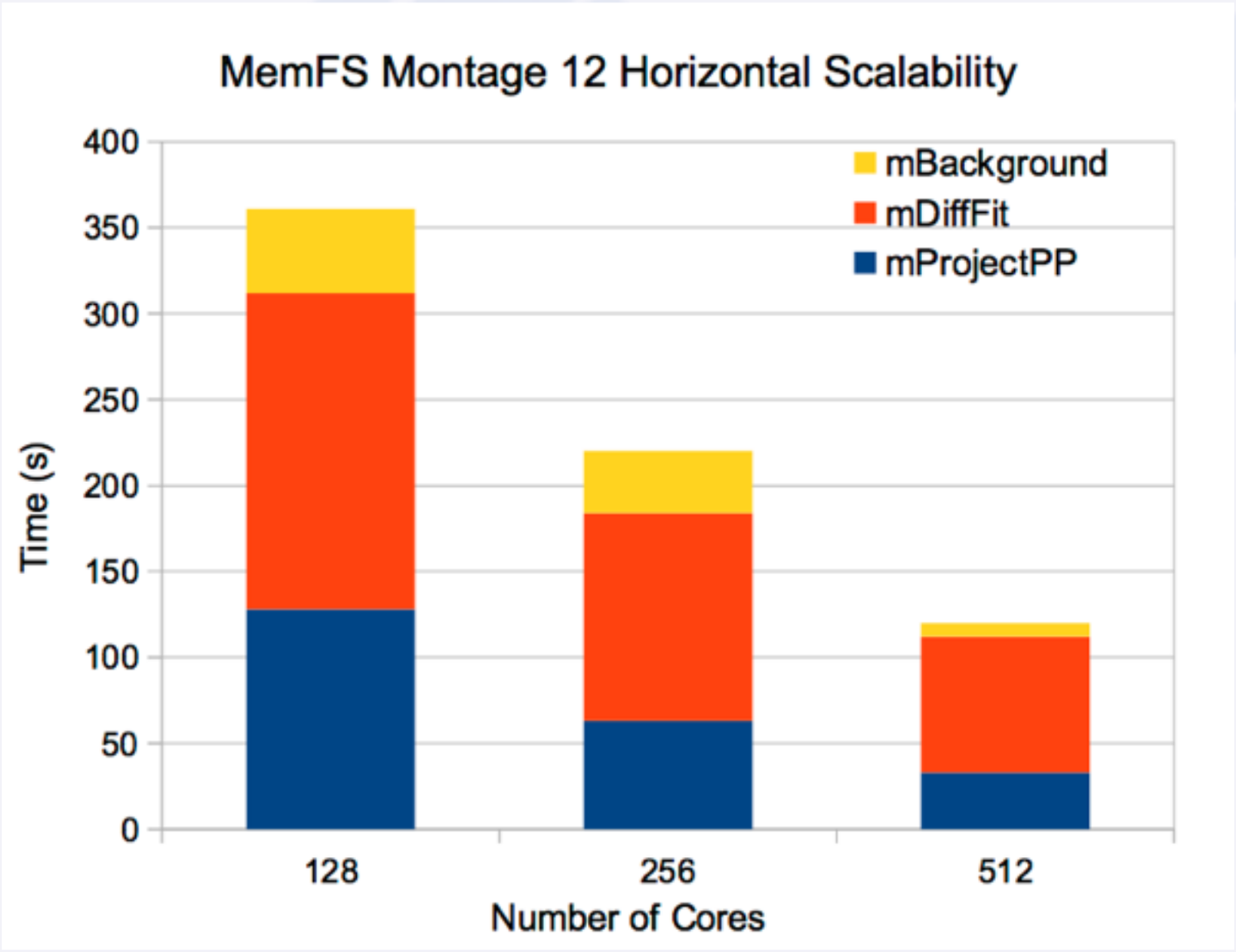
Number of Nodes	Scheduler Node	Other Nodes
8	19 GB	9.5 GB
16	17 GB	5.5 GB
32	16 GB	3 GB
64	16 GB	1.8 GB

- AMFS does not scale up to 8 cores per node because in the 64 node case, there is less data-locality
- the scheduler node becomes a centralized bottleneck

Montage 6 Results



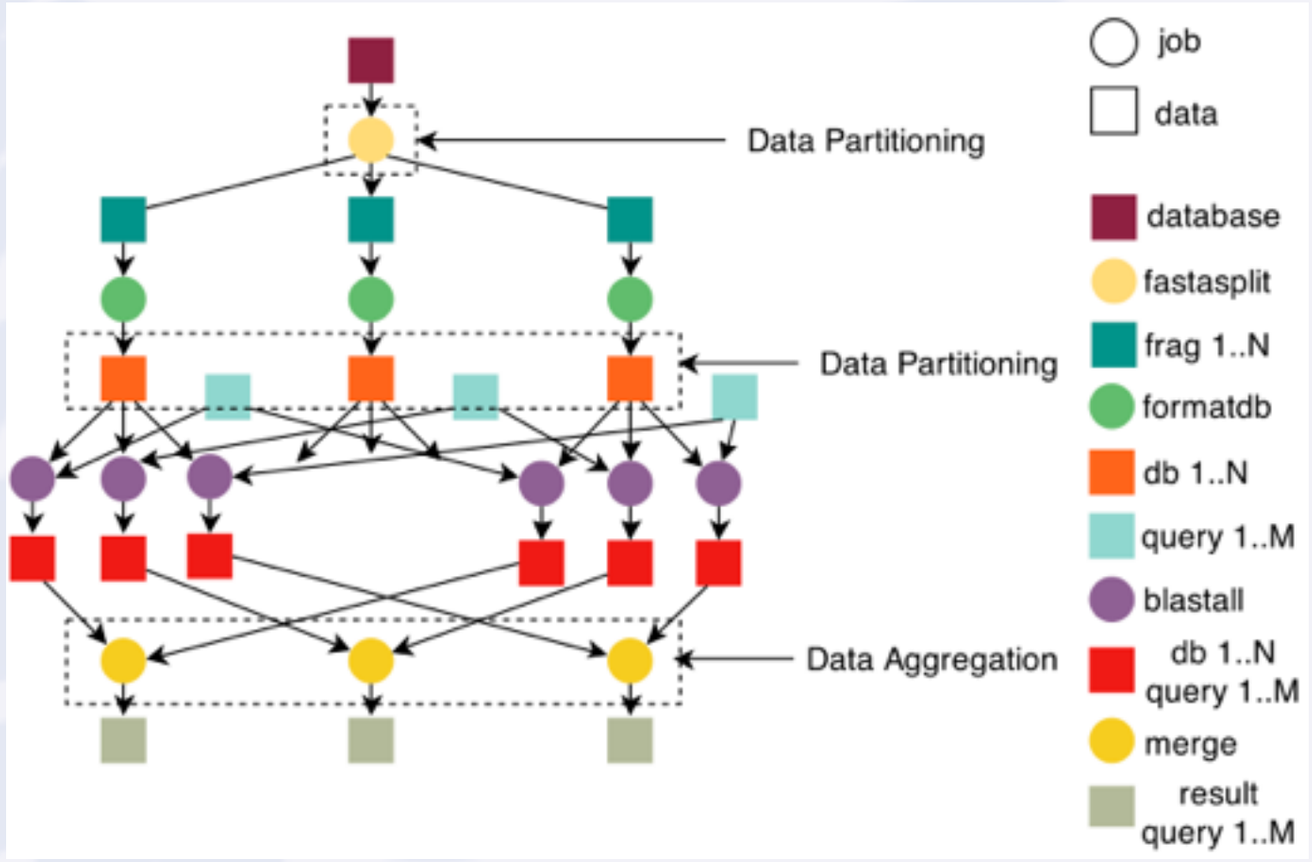
Montage12 Results



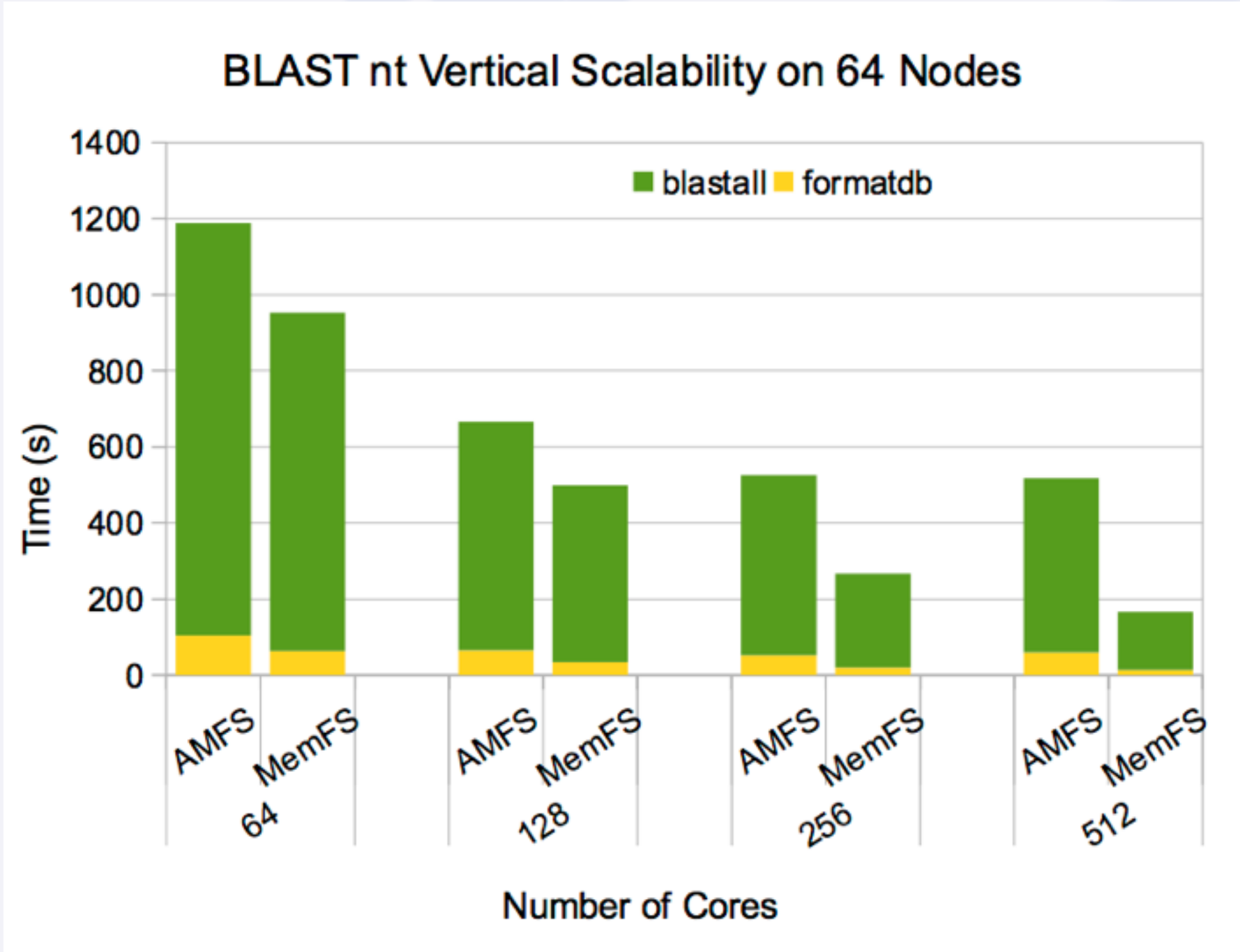
AMFS cannot run this use case;
the data does not fit into the scheduler node

BLAST Workflow

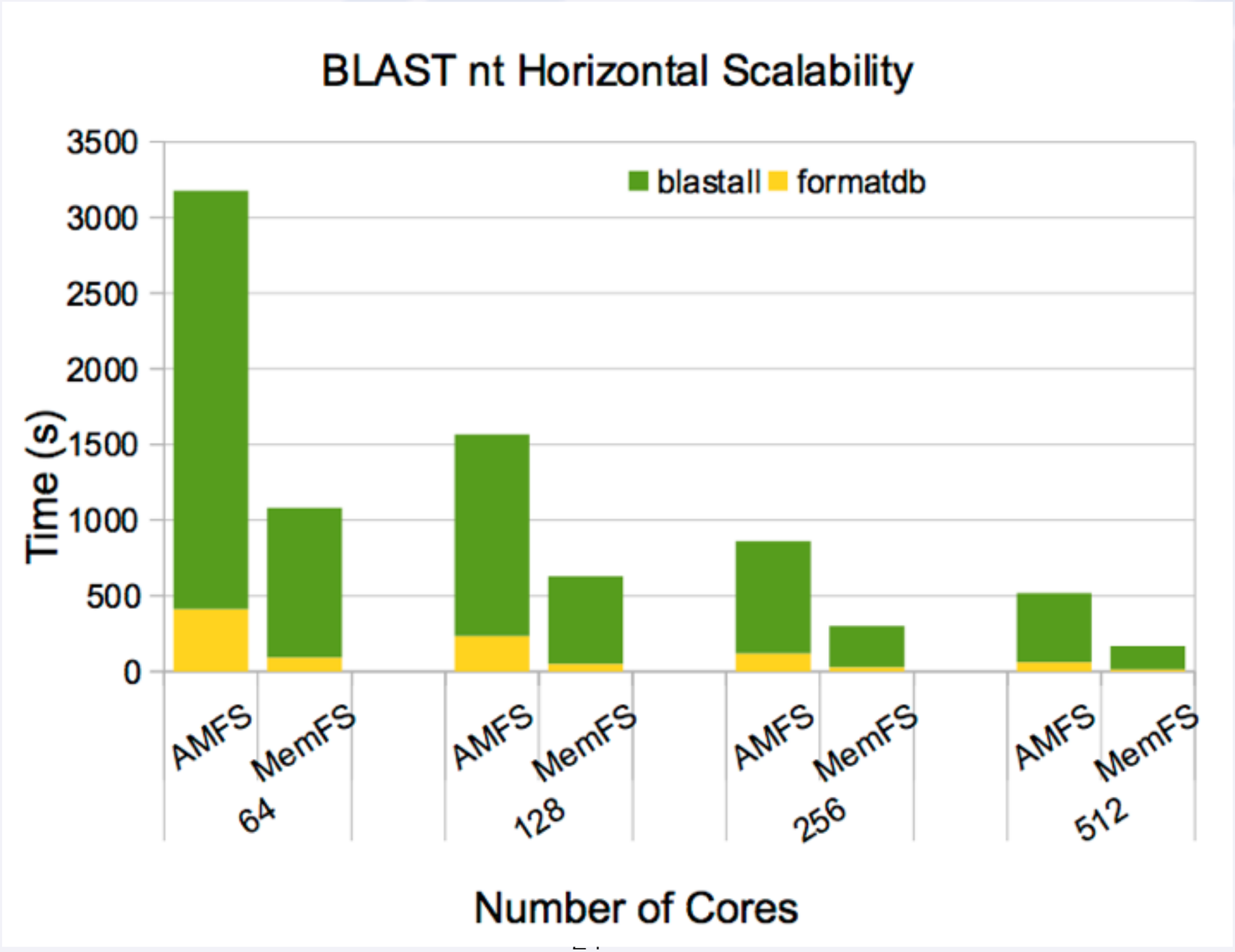
- bioinformatics app
- searches for gene sequences in a database
- input *nt* database (57 GB input)
- assessed performance, horizontal/vertical scalability



BLAST Results



BLAST Results



Conclusions

- It is better to equally distribute large data
  - Better utilisation of memory capacity
  - Can run larger problems
  - Better balancing helps speeding up

- MemFS implements a distributed hash table overlay using memcached/libmemcached

- MemFS scales welly, both horizontally and vertically

Future Research Directions

- decreasing CPU load for better vertical scalability: using native RDMA, or kernel file system

- supporting malleability/elasticity
  - e.g., scaling out when memory exceeded

- develop scheduler that exploits malleability to:
  - save cost
  - increase/decrease aggregate throughput
  - increase/decrease system capacity
  - save power