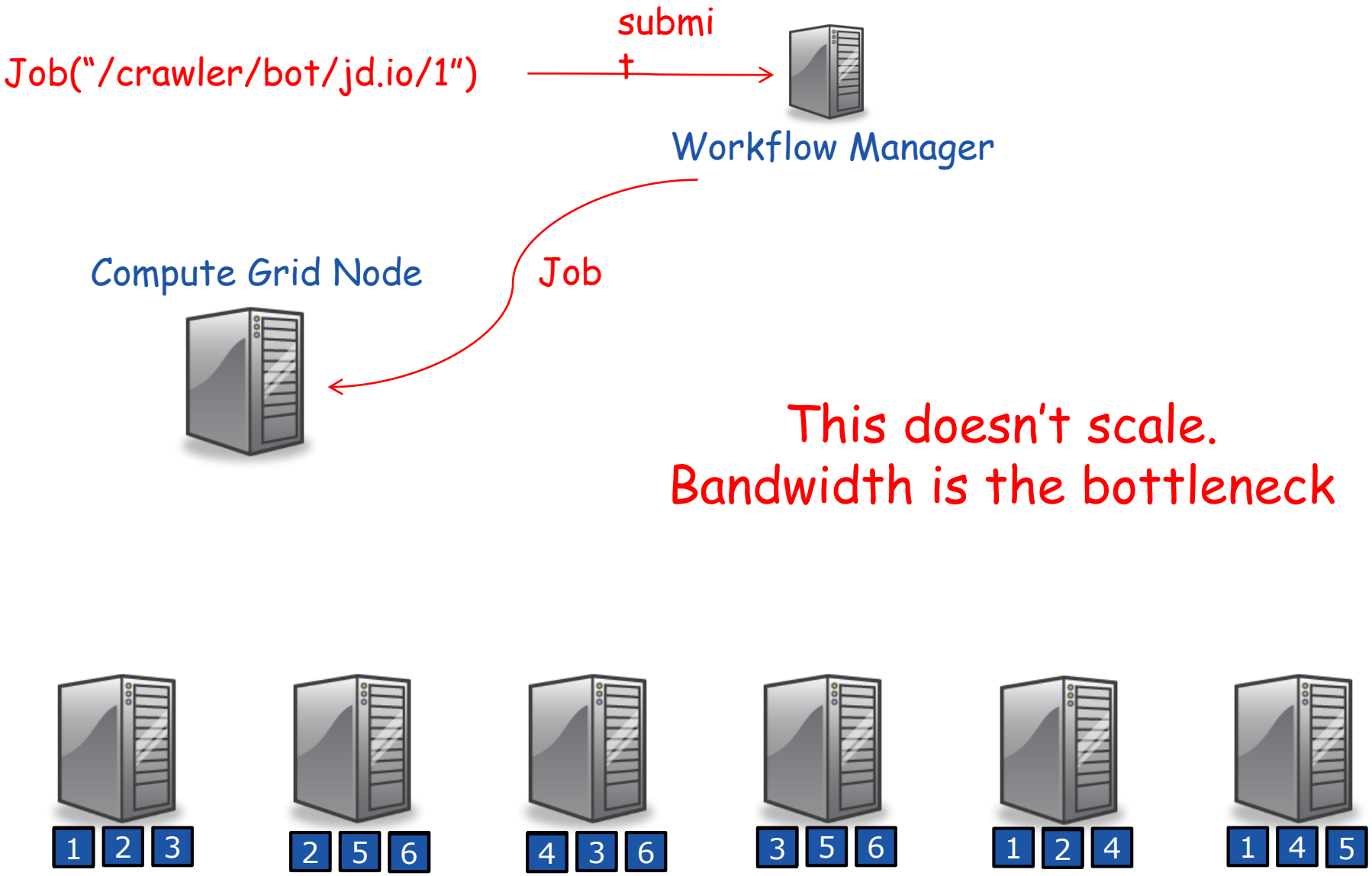
ROYAL INSTITUTE
OF TECHNOLOGY

Managing large clusters resources

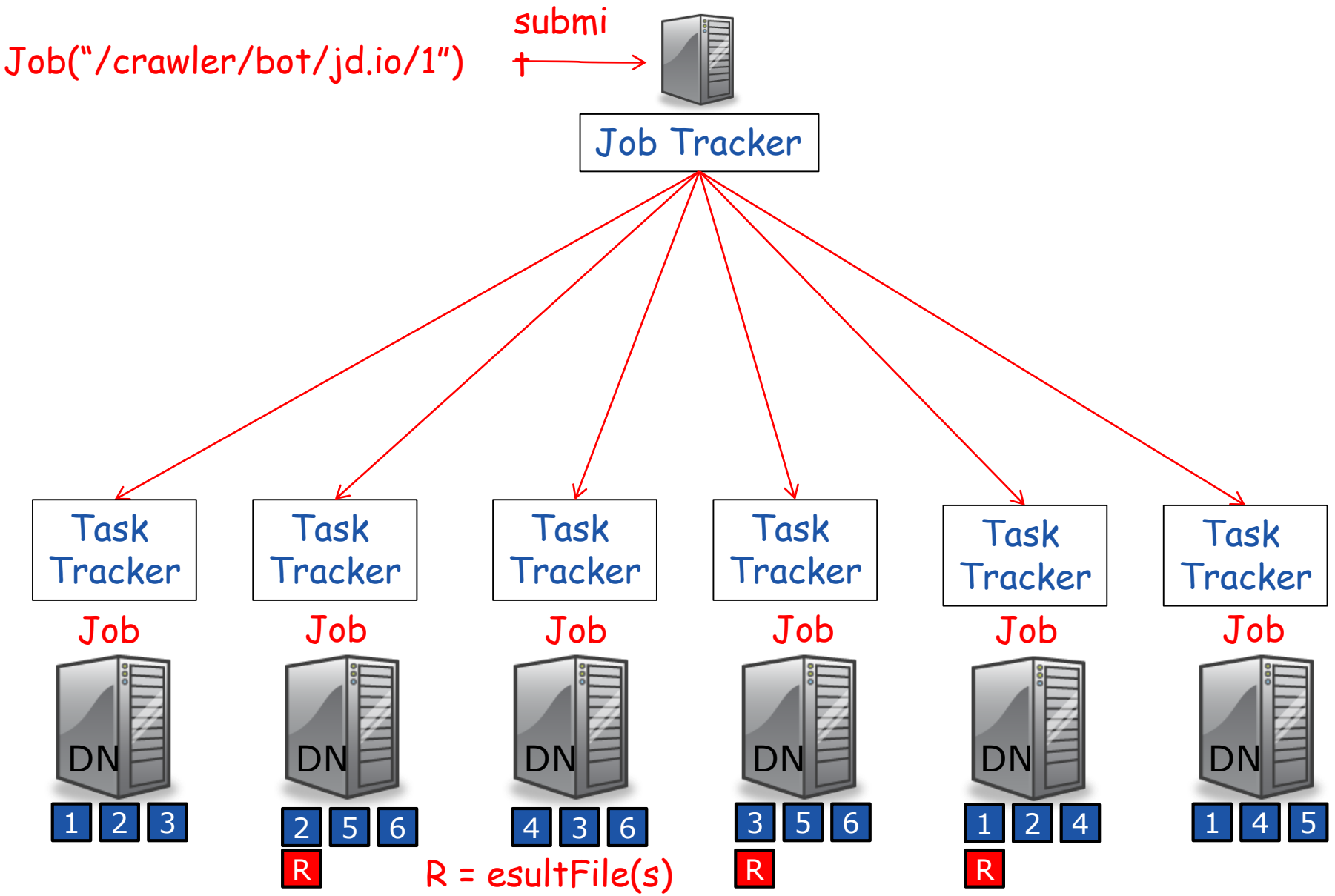
ID2210

Gautier Berthou (SICS)

Big Data Processing with No Data Locality



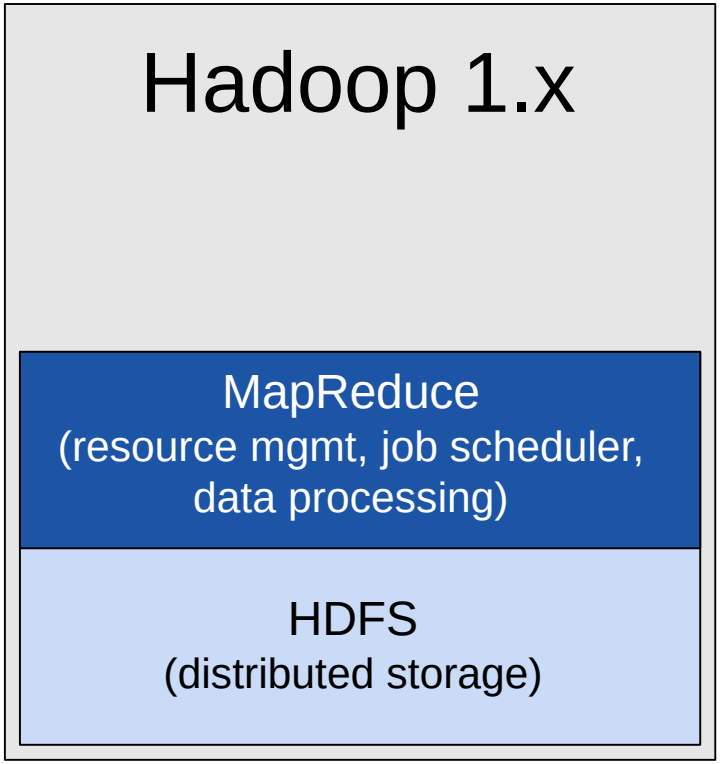
MapReduce – Data Locality



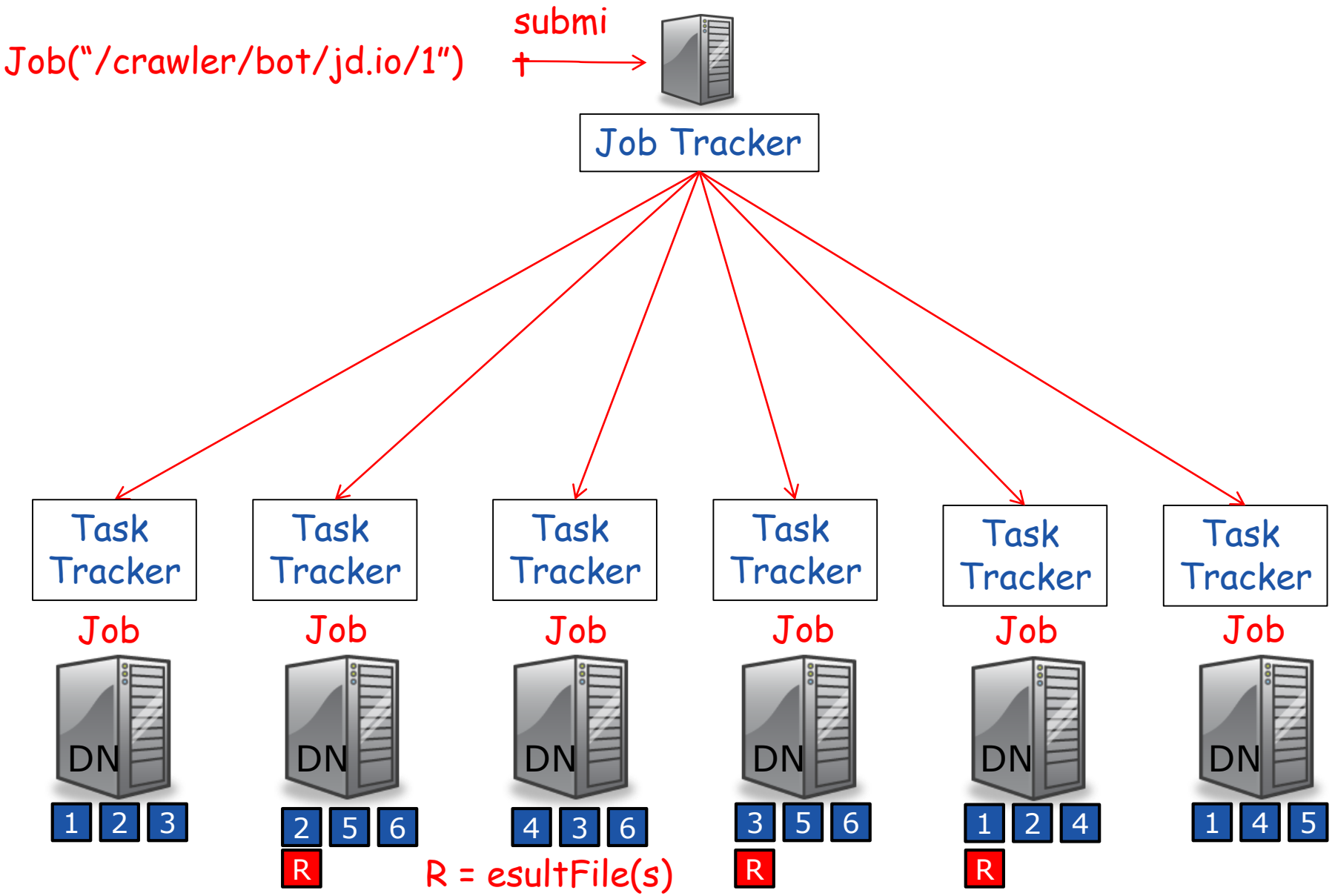
First step

Single Processing Framework

Batch Apps



MapReduce – Data Locality

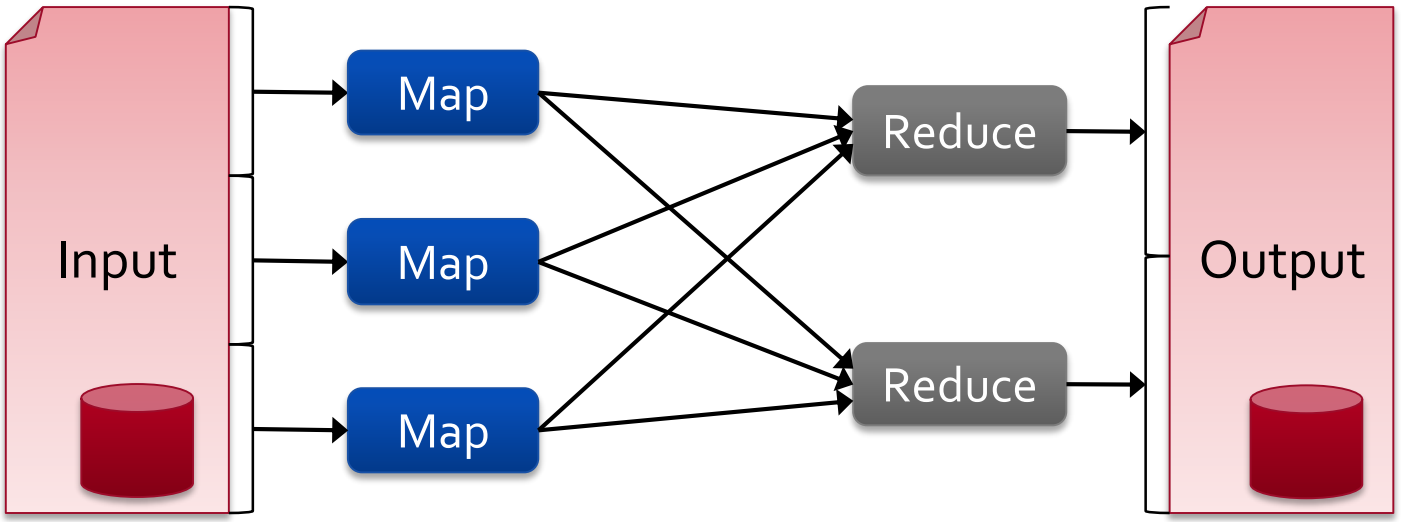


The Job Tracker Job

- Distribute the map and reduce tasks on the nodes of the cluster
- Ensure fairness of the cluster resource attributions
- Track the progress of these tasks
- Authenticate job tenants and make sure that each job is isolated from the others
- Etc.

Limitations of MapReduce [Zaharia'11]

- MapReduce is based on an *acyclic data flow* from stable storage to stable storage.
  - Slow writes data to HDFS at every stage in the pipeline
- Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (machine learning, graphs)
  - **Interactive** data mining tools (R, Excel, Python)



Limitations

- Only one programming model.
- The map reduce framework is not using the cluster at its maximum.
- The job tracker is a bottle neck.
- The job tracker is a single point of failure.

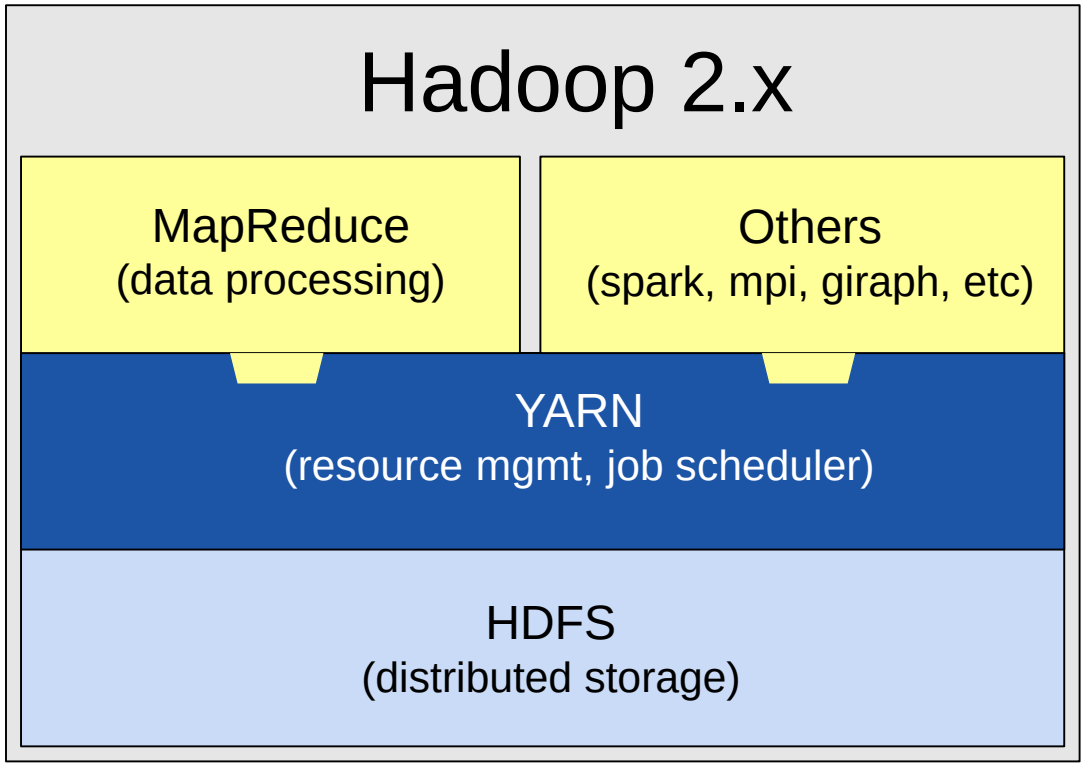
Goals for a new Scheduler

- Being able to run different frameworks
- Scale
- Provide advance scheduling policies
- Run efficiently with different kind of workloads

Second step

Multiple Processing Frameworks

Batch, Interactive, Streaming …



Examples of scheduling Policies

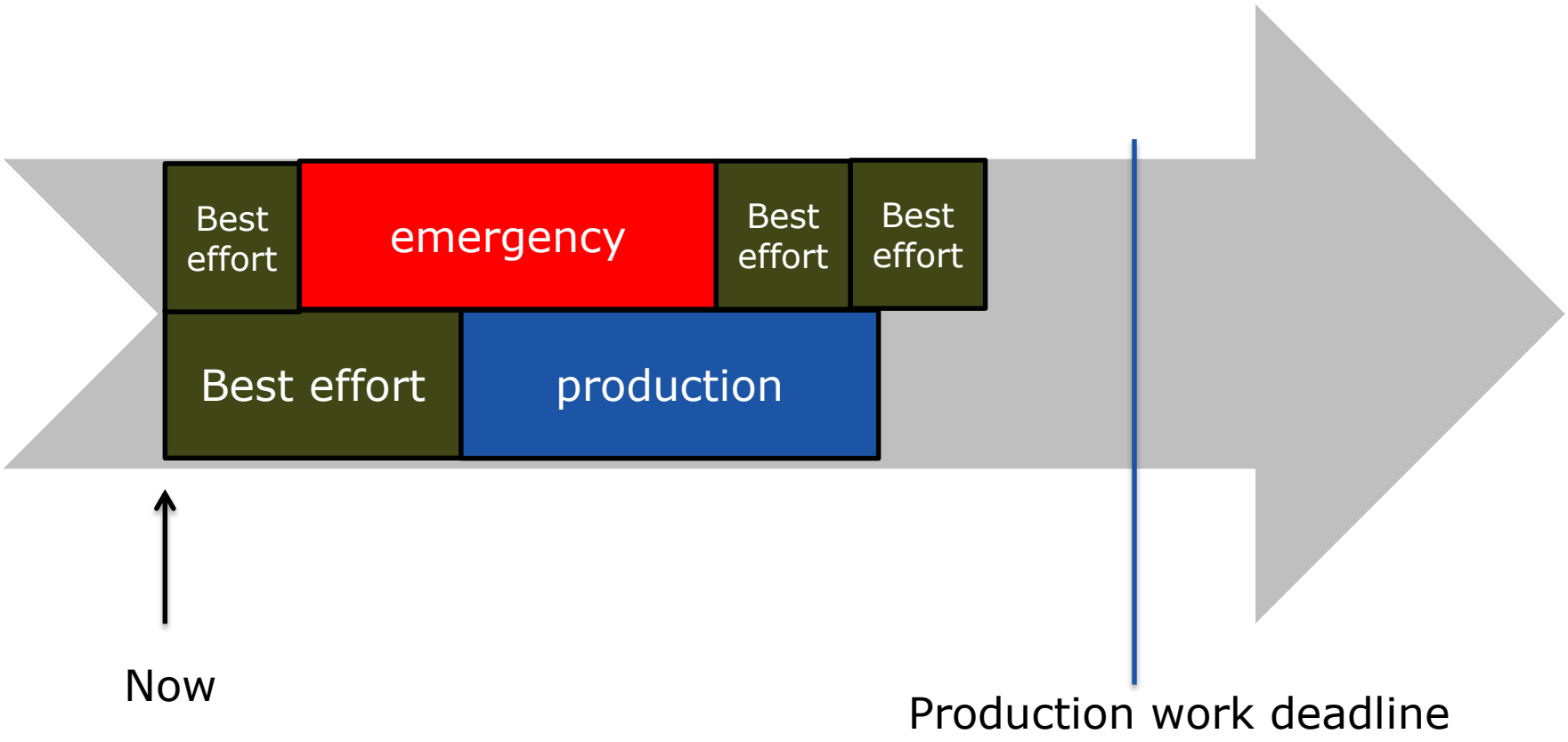
- Capacity scheduler:
  - Applications have different levels of priorities.
- Fair scheduler:
  - Applications have different levels of priorities.
  - Used resources can be preempted.
- Reservation-based scheduler(1):
  - Applications can indicate how long they will run and when they have to be finished.

(1) *Reservation-based Scheduling: If you're late don't blame us!*, C. Curino & al., Microsoft tech-report

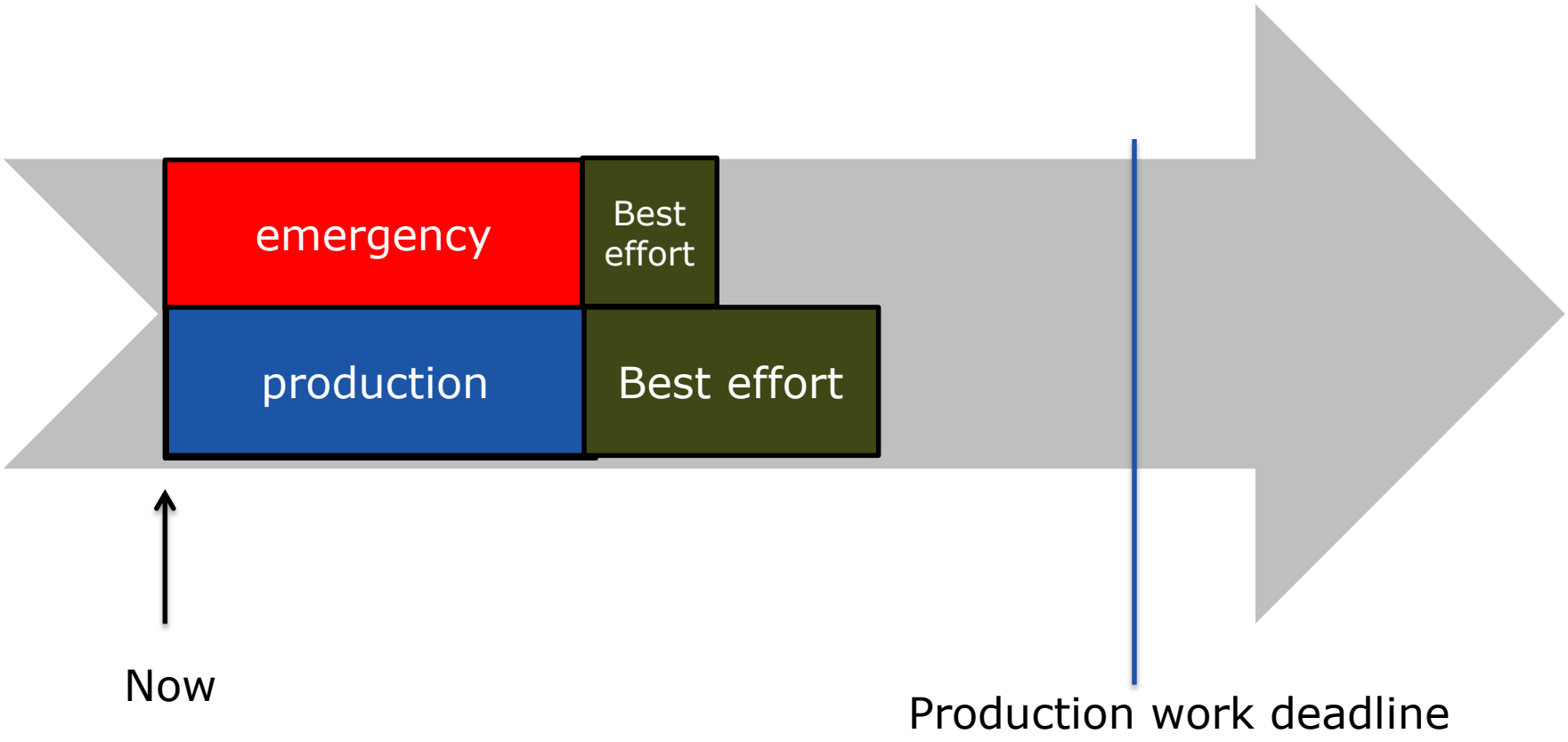
Scenario

- 3 kinds of jobs:
  - Emergency jobs: need to be run as soon as possible.
  - Production jobs: have a deadline, a known running time and are very exigent on the nodes they can be scheduled on.
  - Best effort jobs: interactive jobs that have lower priority, but on which users expect low latency.

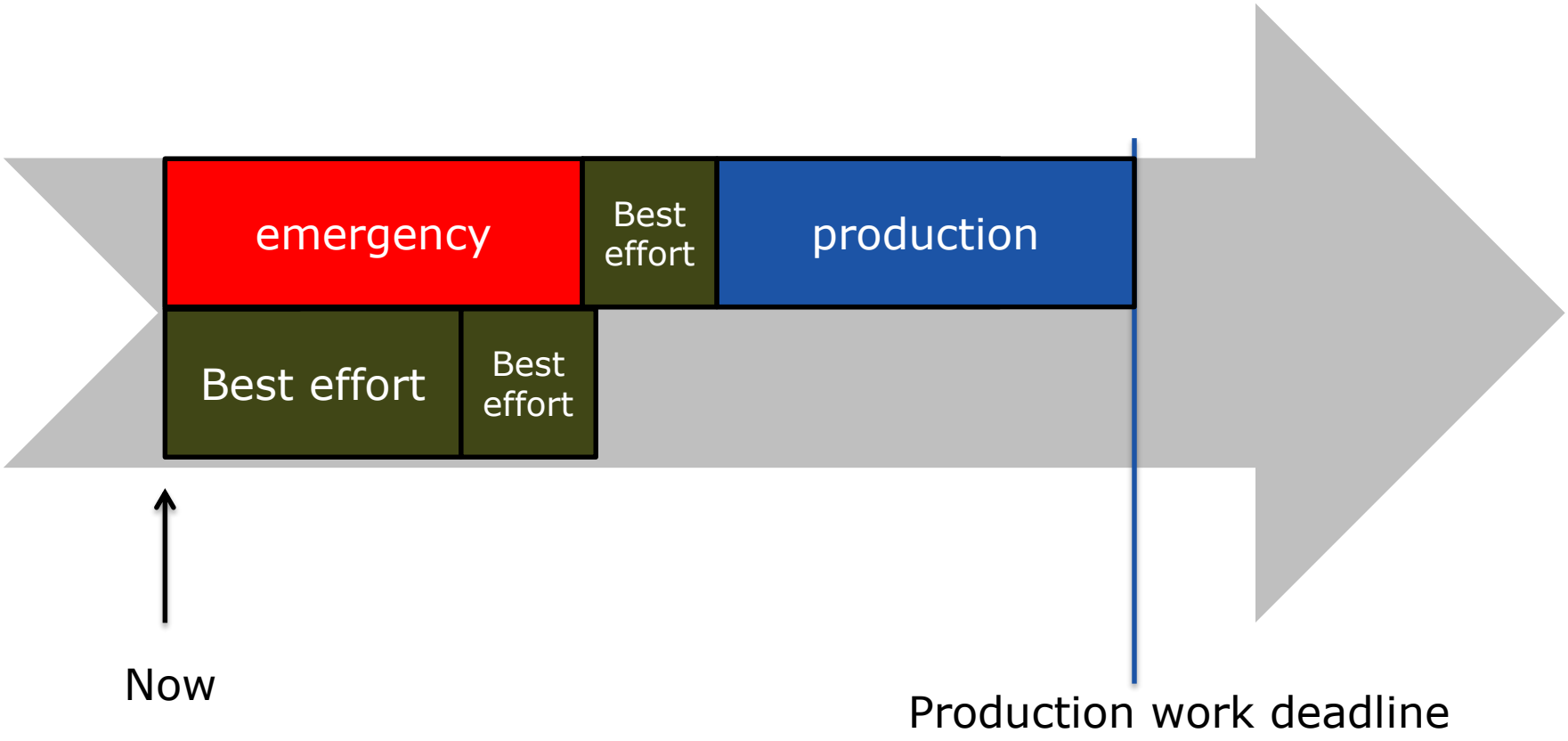
Capacity scheduler



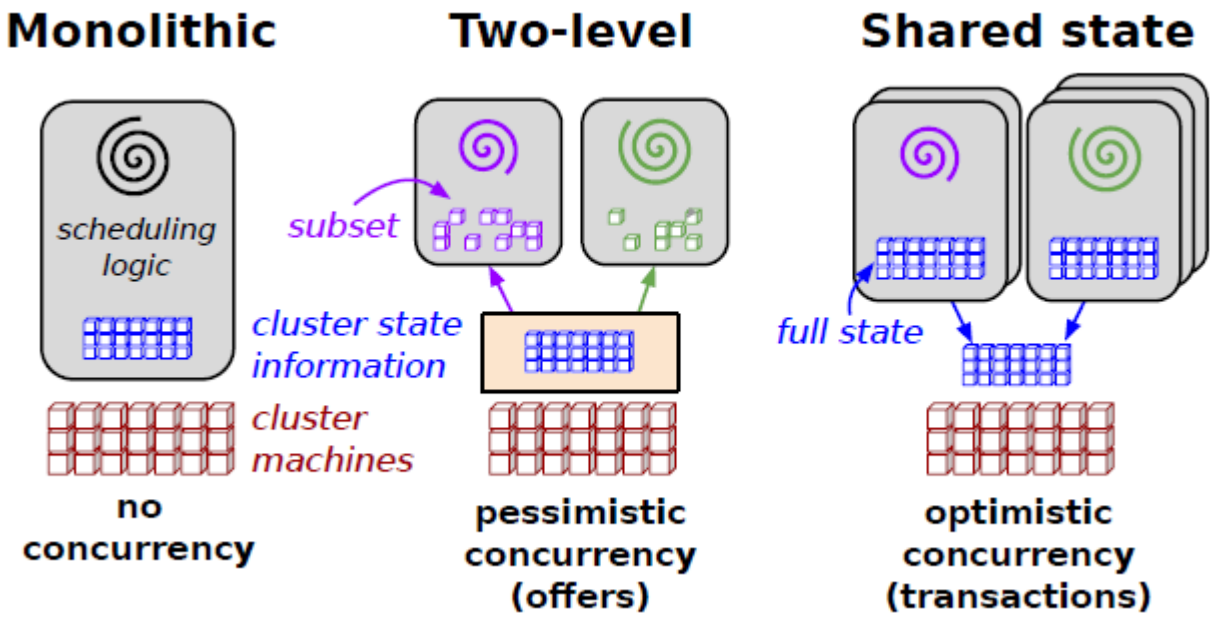
Fair scheduler



Reservation-based scheduler



Scheduler Architectures



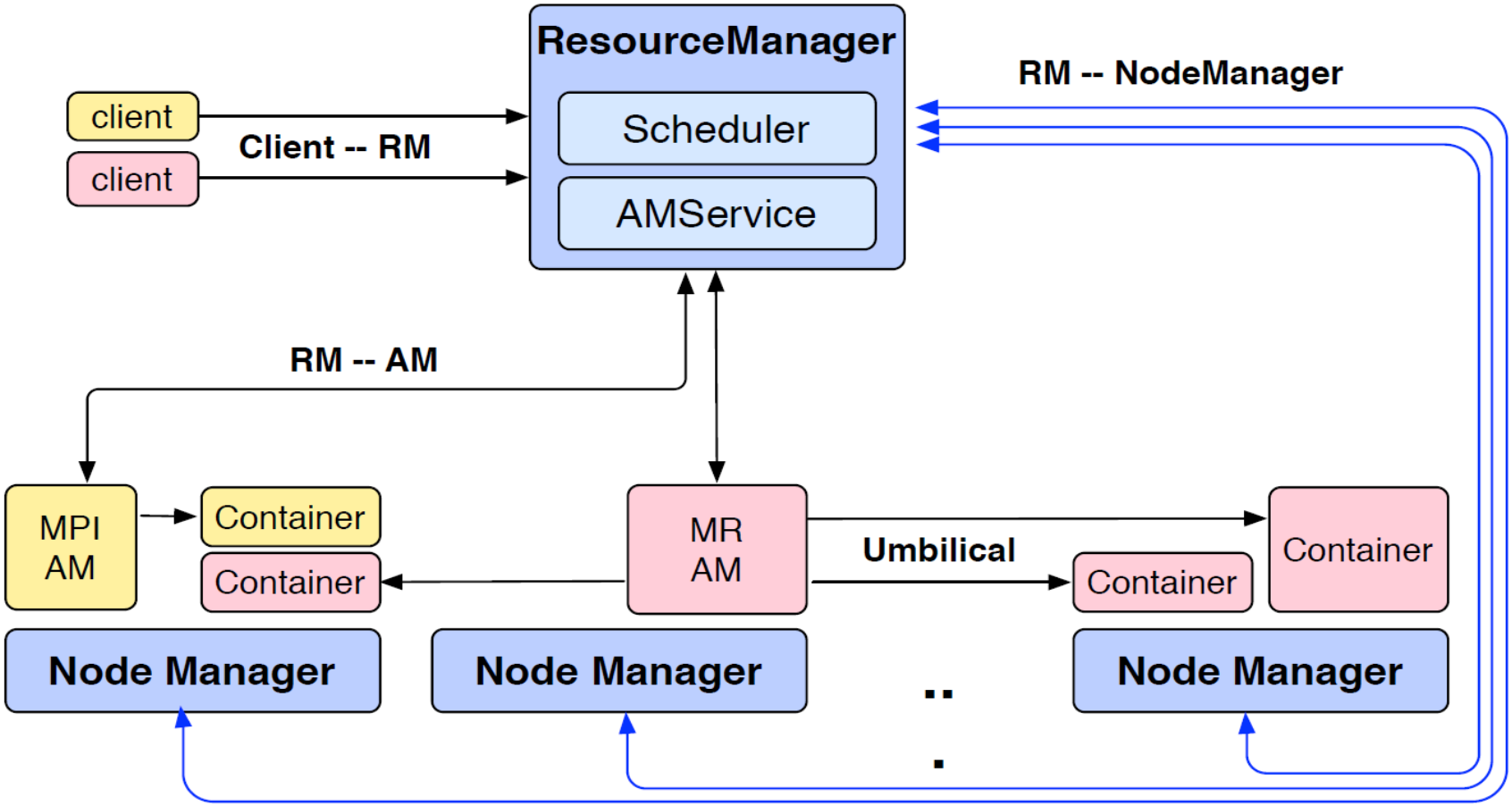
Omega: flexible, scalable schedulers for large compute clusters, Malte Schwarzkopf & al., EuroSys'13

The monolithic Scheduler

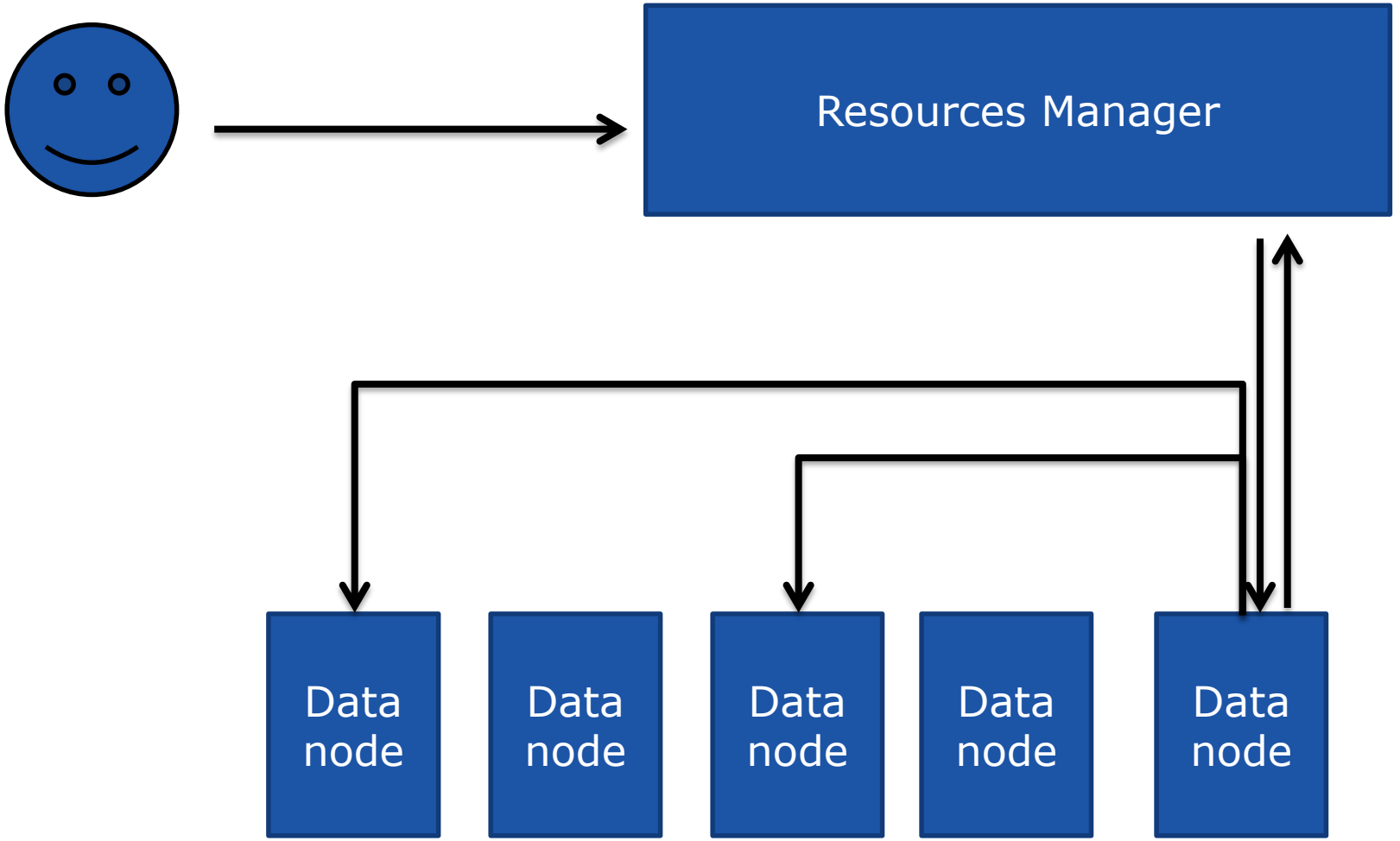
- Yarn:
  - Apache Hadoop YARN: Yet Another Resource Negotiator, V. K. Vavilapalli & al., SoCC'13.

- Borg:
  - Large-scale cluster management at Google with Borg, A. Verma & al., EuroSys'15.

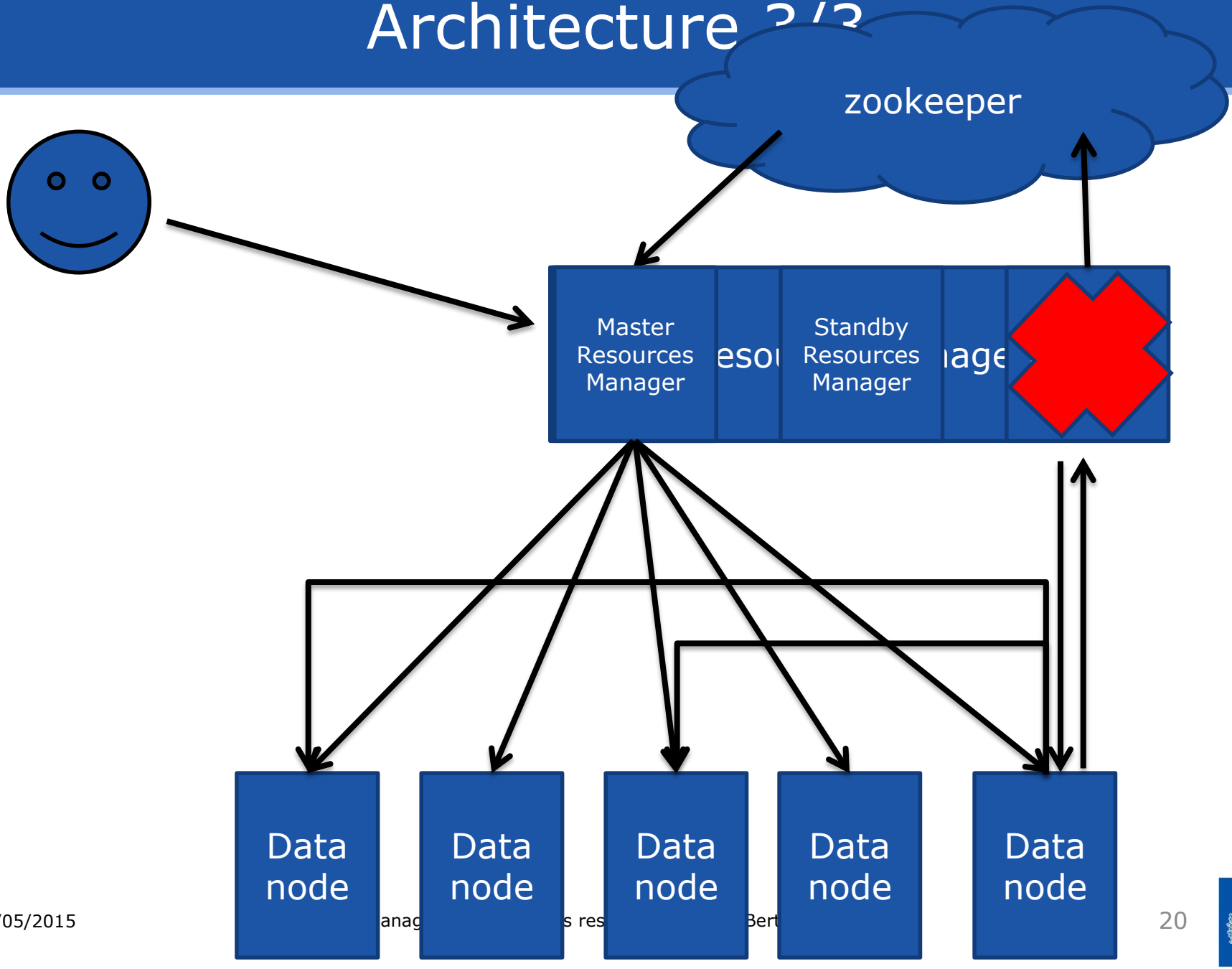
Architecture 1/3



Architecture 2/3



Architecture 2/3



Pros and Cons

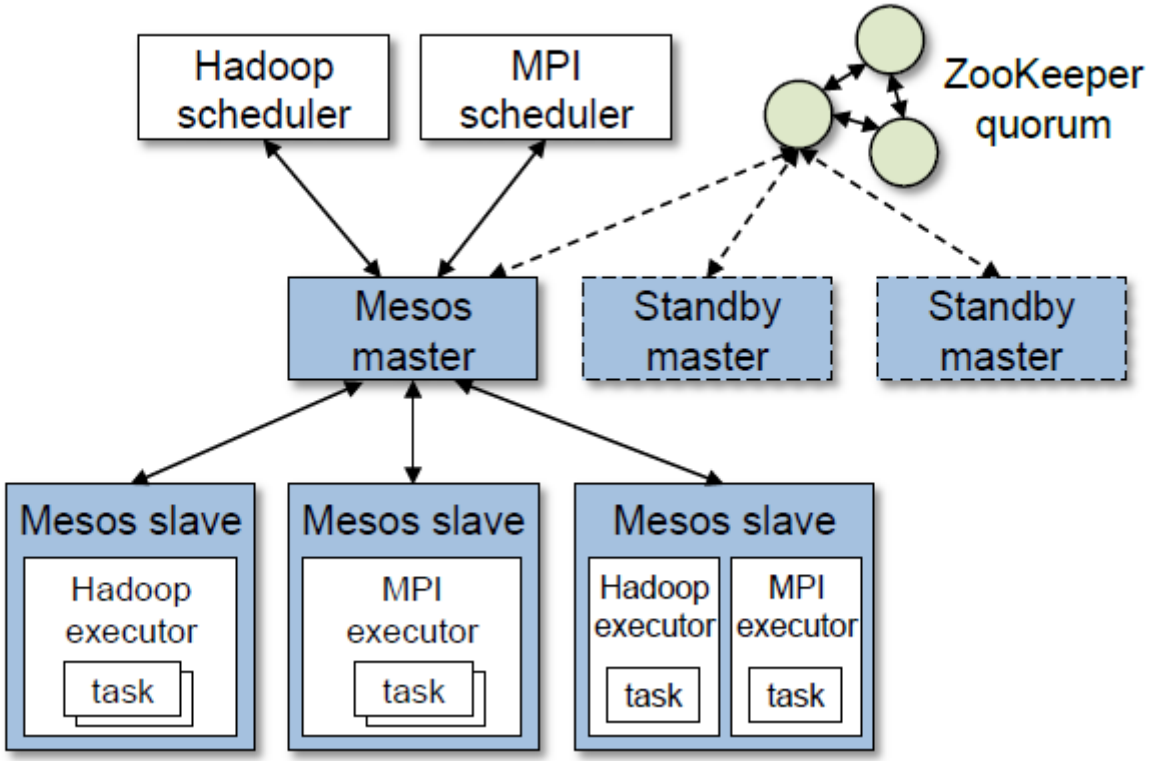
- Pros:
  - Fine knowledge of the state of the cluster state -> optimal use of the cluster resources.
  - Easy to implement new scheduling policies.

- Cons:
  - Bottle neck.
  - The failure of the master scheduler has a big impact on the cluster usage.

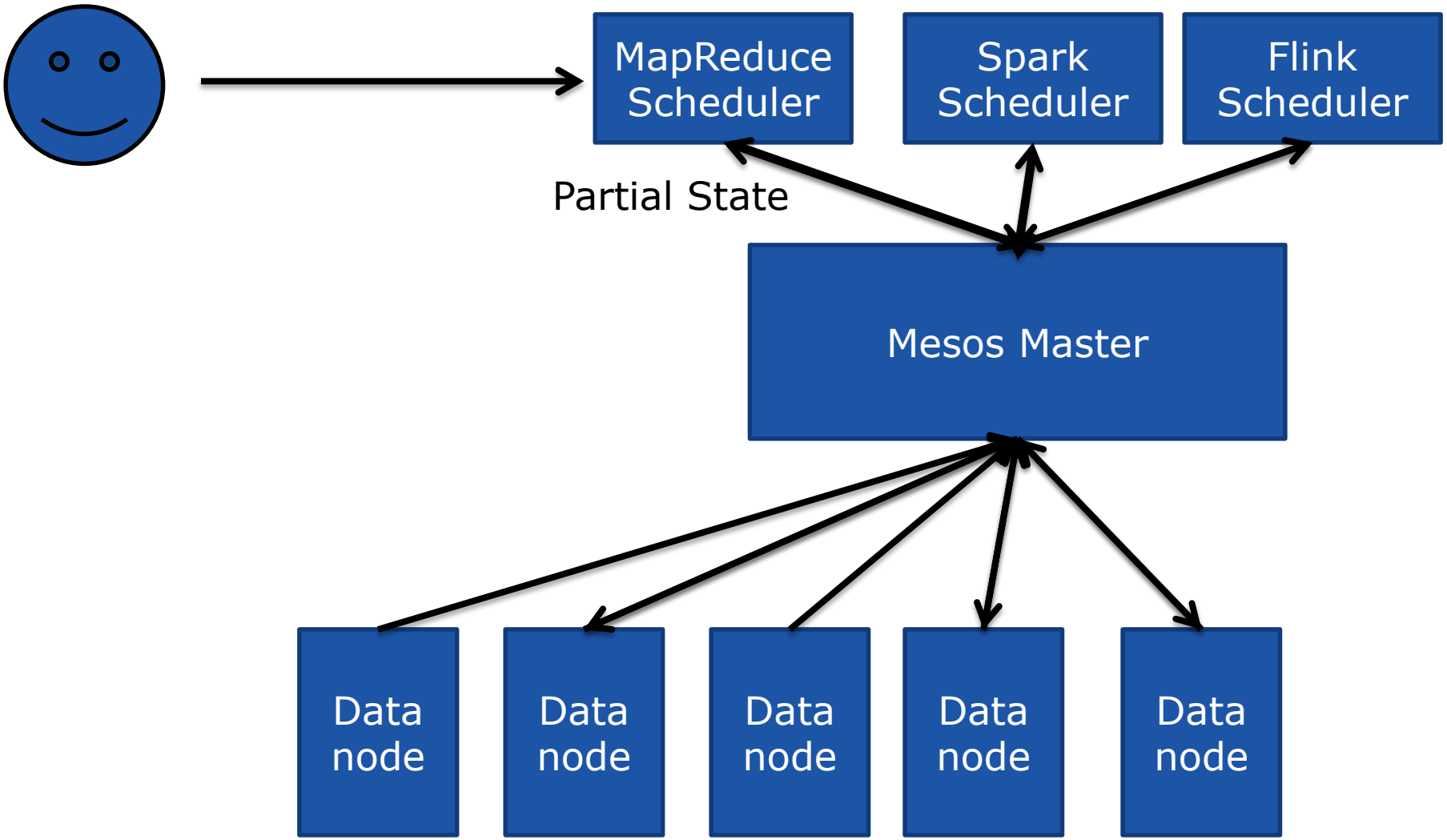
Two level Scheduler

- *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*, B. Hindman & al., NSDI'11

Architecture 1/2



Architecture 2/2



Pros and Cons

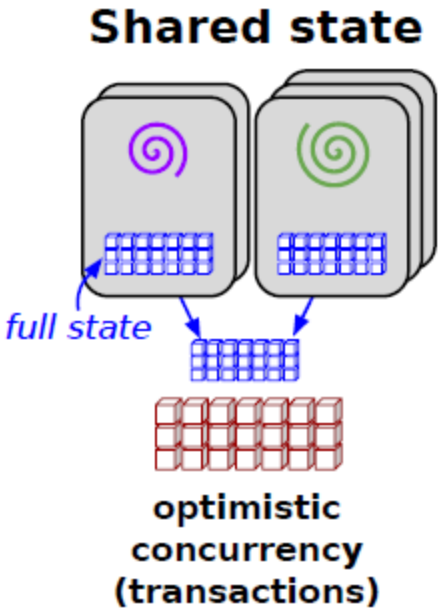
- Pros:
  - Scale out by adding schedulers.
  - Concurrent scheduling of tasks.

- Cons:
  - Suboptimal use of the cluster. Especially when there exist long running tasks.

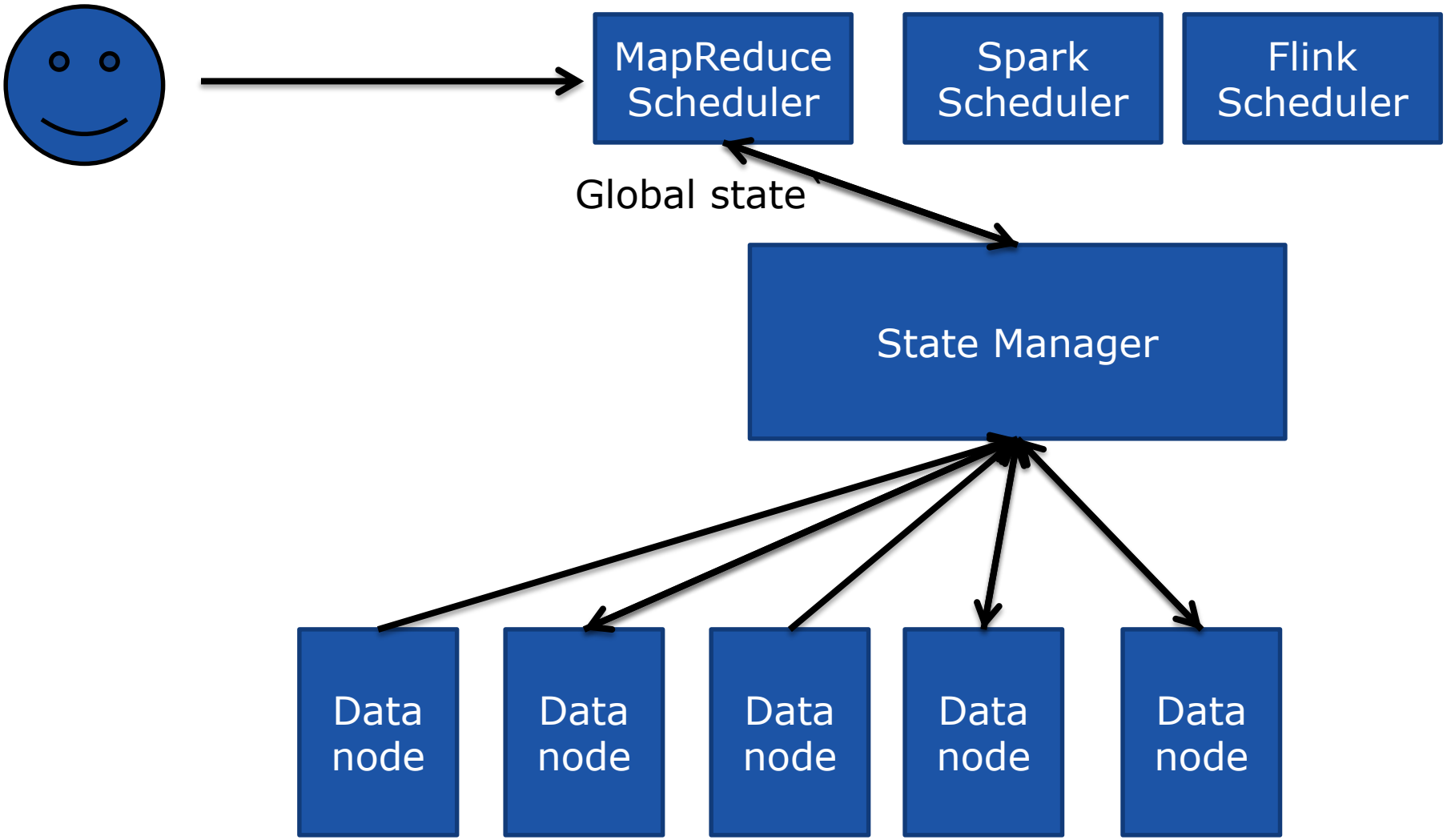
Shared State Scheduler

- *Omega: flexible, scalable schedulers for large compute clusters*, M. Schwarzkopf & al. EuroSys'13

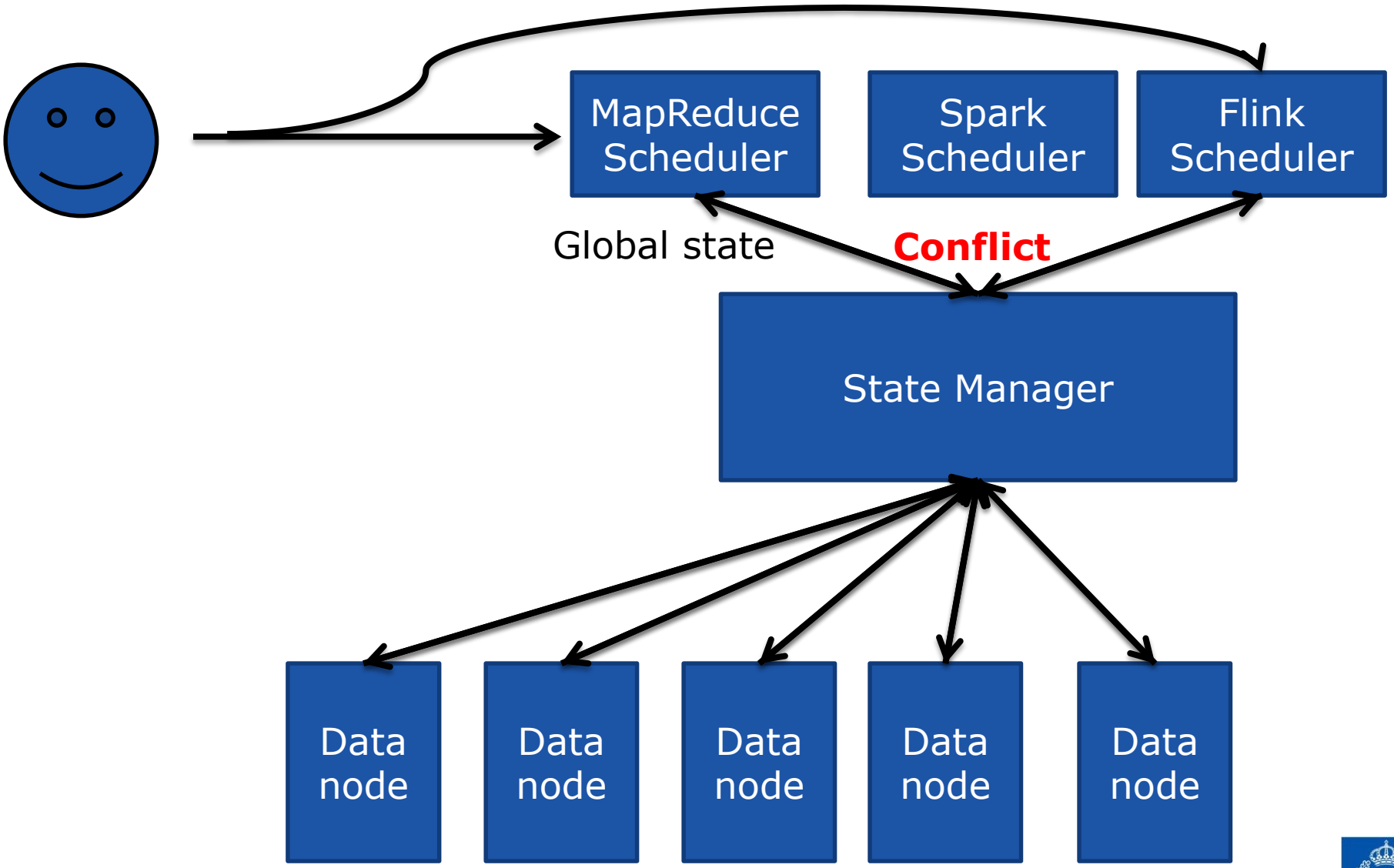
Architecture 1/2



Architecture 2/2



Architecture 2/2



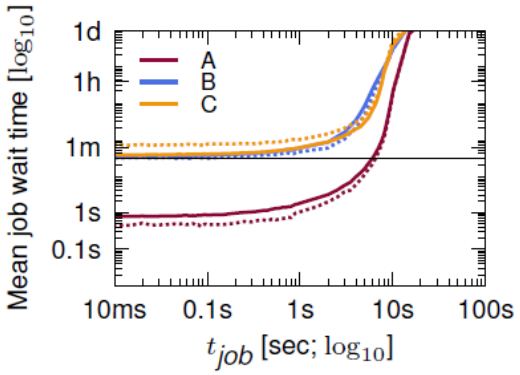
Pros and Cons

- Pros
  - Scalable.
  - Good use of the cluster resources.
- Cons
  - Unpredictable interaction between the different schedulers' policies.

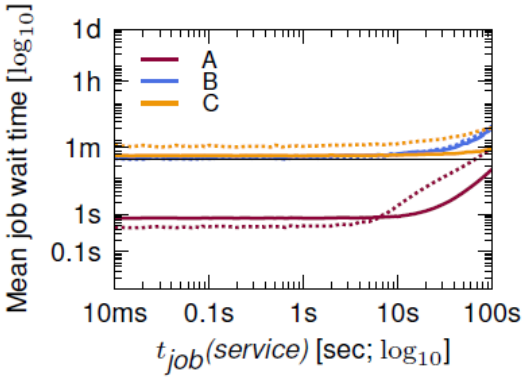
Comparison

Approach	*Resource choice*	*Interference*	*Alloc. granularity*	*Cluster-wide policies*
Monolithic	all available	none (serialized)	global policy	strict priority (preemption)
Statically partitioned	fixed subset	none (partitioned)	per-partition policy	scheduler-dependent
Two-level (Mesos)	dynamic subset	pessimistic	hoarding	strict fairness
Shared-state (Omega)	all available	optimistic	per-scheduler policy	free-for-all, priority preemption

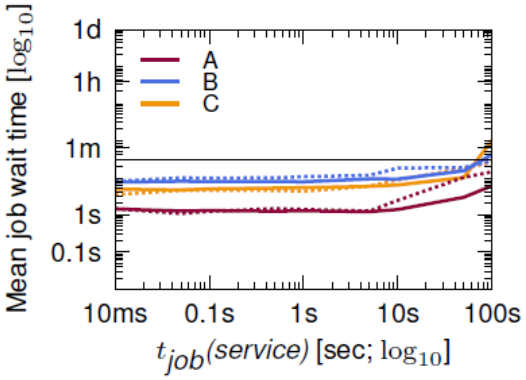
Performance comparison 1/



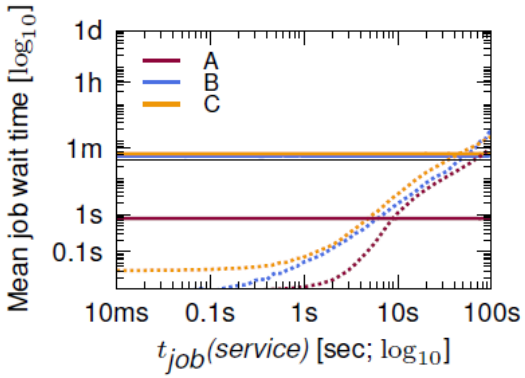
Simple monolithic



Advence monolithic



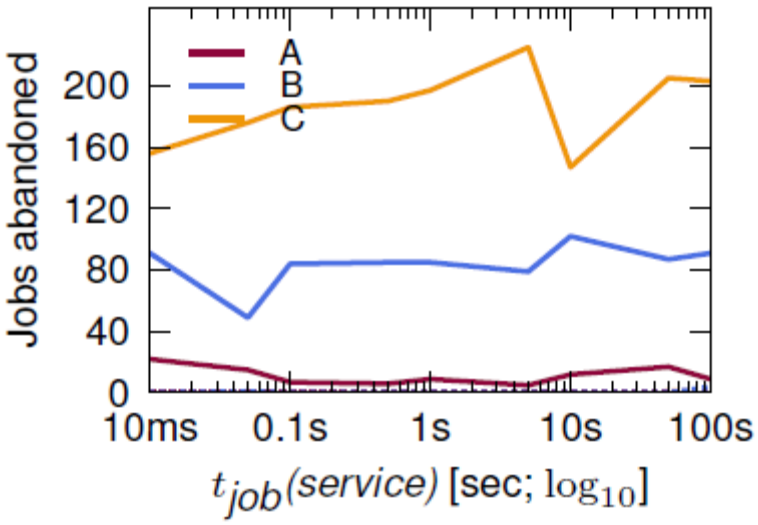
Two-level



Shared state

Performance Comparison 2/

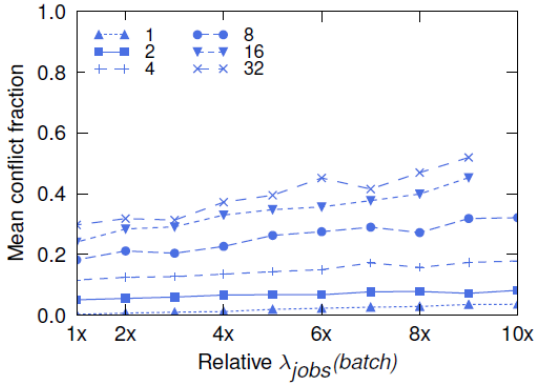
What the previous evaluation does not show about the Two-level scheduling:



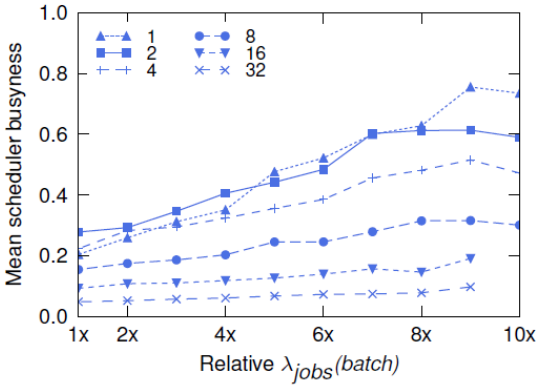
(c) Unscheduled jobs.

Performance Comparison 3/

Trying to handle more batch jobs in Omega by running several batch schedulers in parallel.



(a) Mean conflict fraction.



How to write an accepted paper.

Sum up

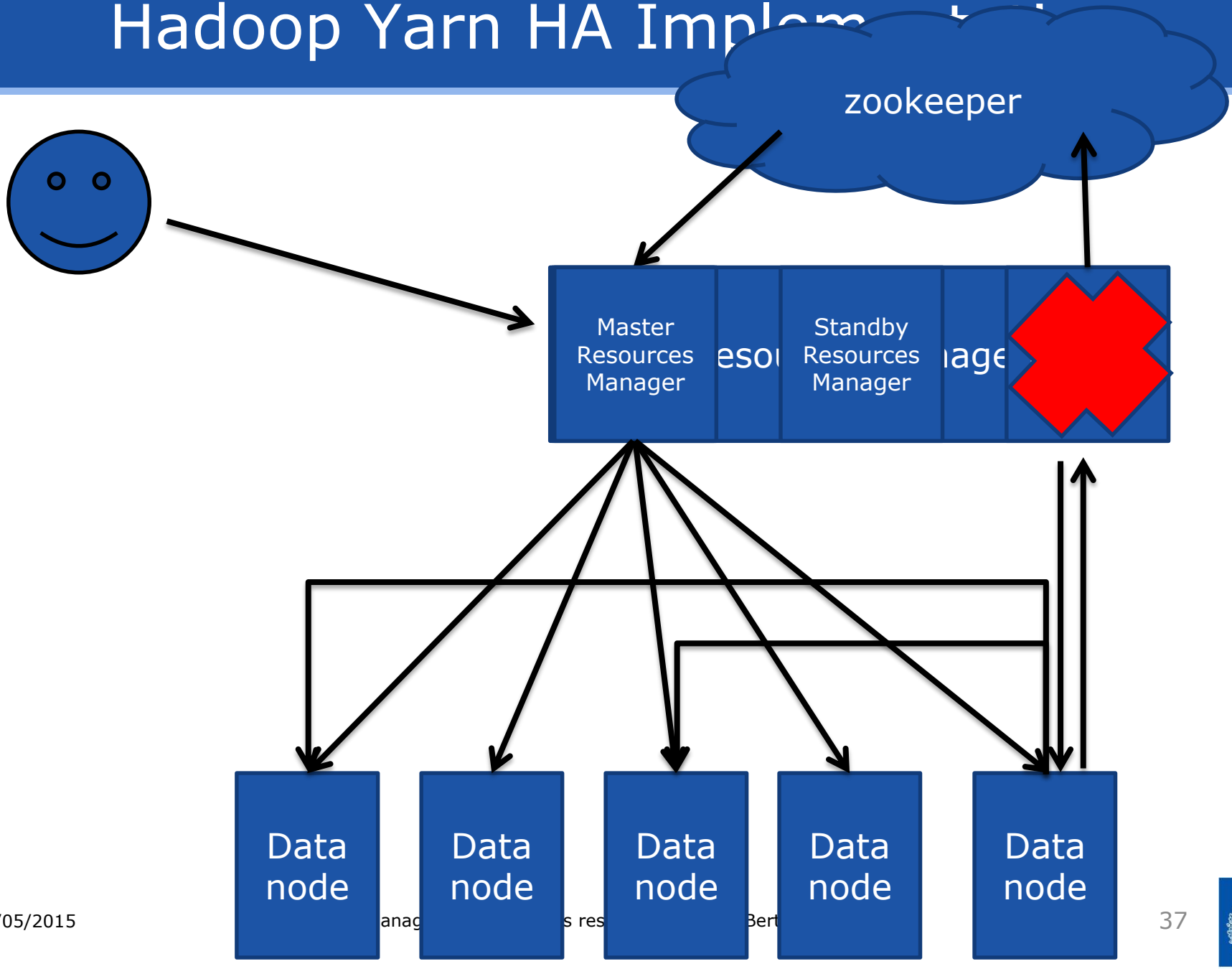
- Two-Level and Shared state Schedulers scale better.
- Shared state Schedulers use the cluster resources more optimally than Two-level Schedulers.
- Monolithic Scheduler are a potential Bottleneck.

- But as Monolithic schedulers are easier to design, allow finer allocation of resources and more advance scheduling policies, they are the ones used in practice.

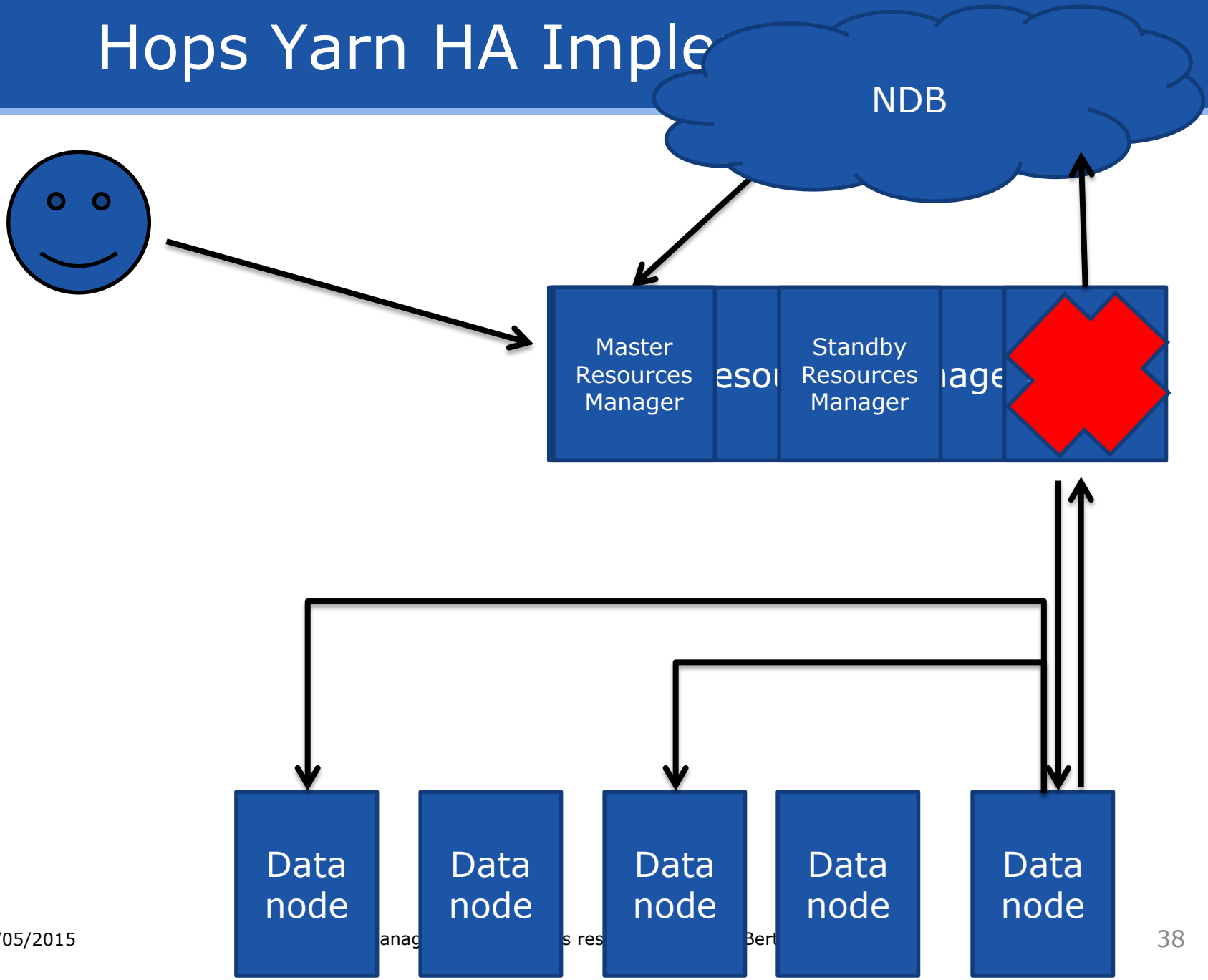
Making Yarn more scalable

- HOPS YARN: a one and a half level scheduler

Hadoop Yarn HA Implem



Hops Yarn HA Imple

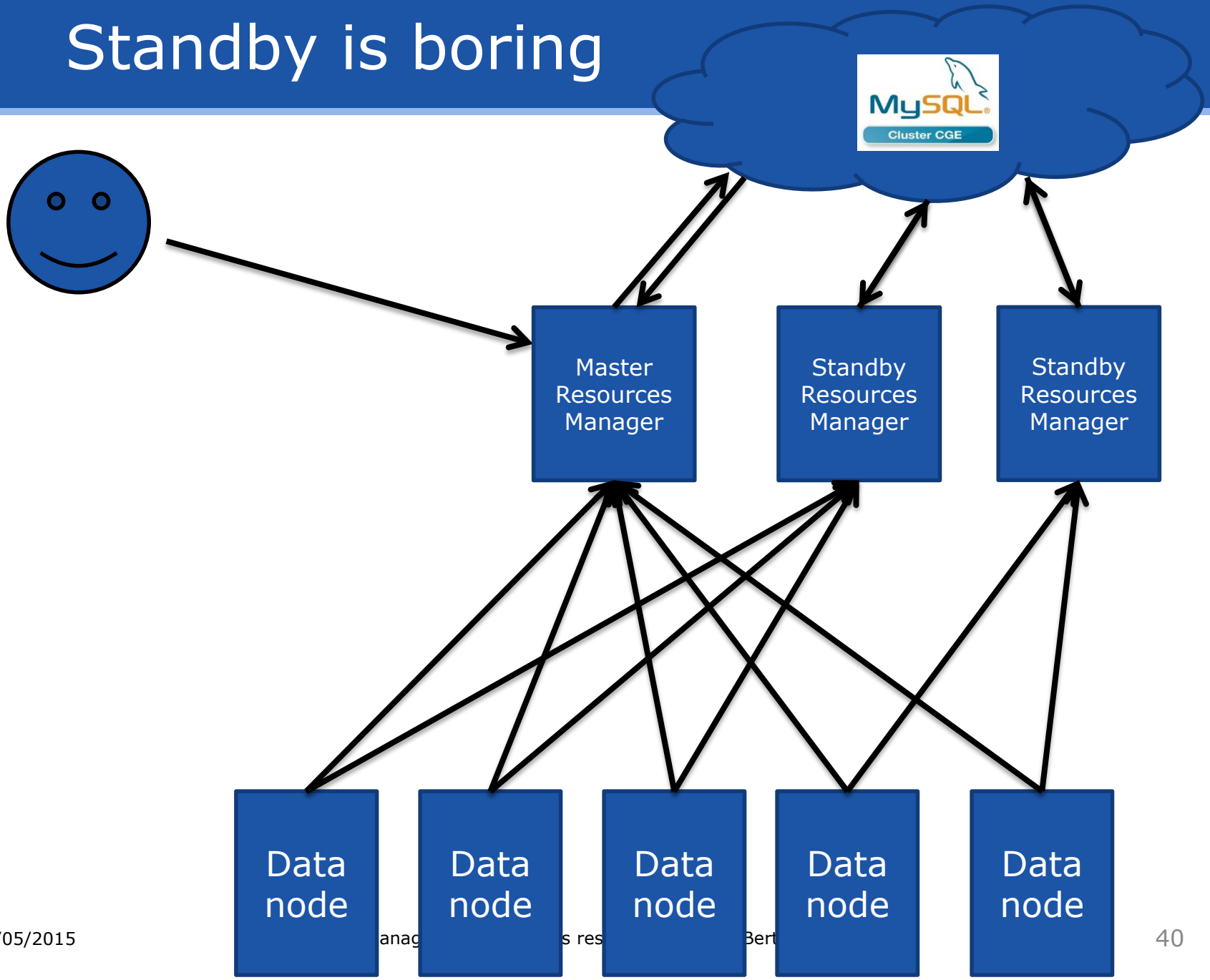


MySQL Cluster (NDB) – Shared Nothing DB

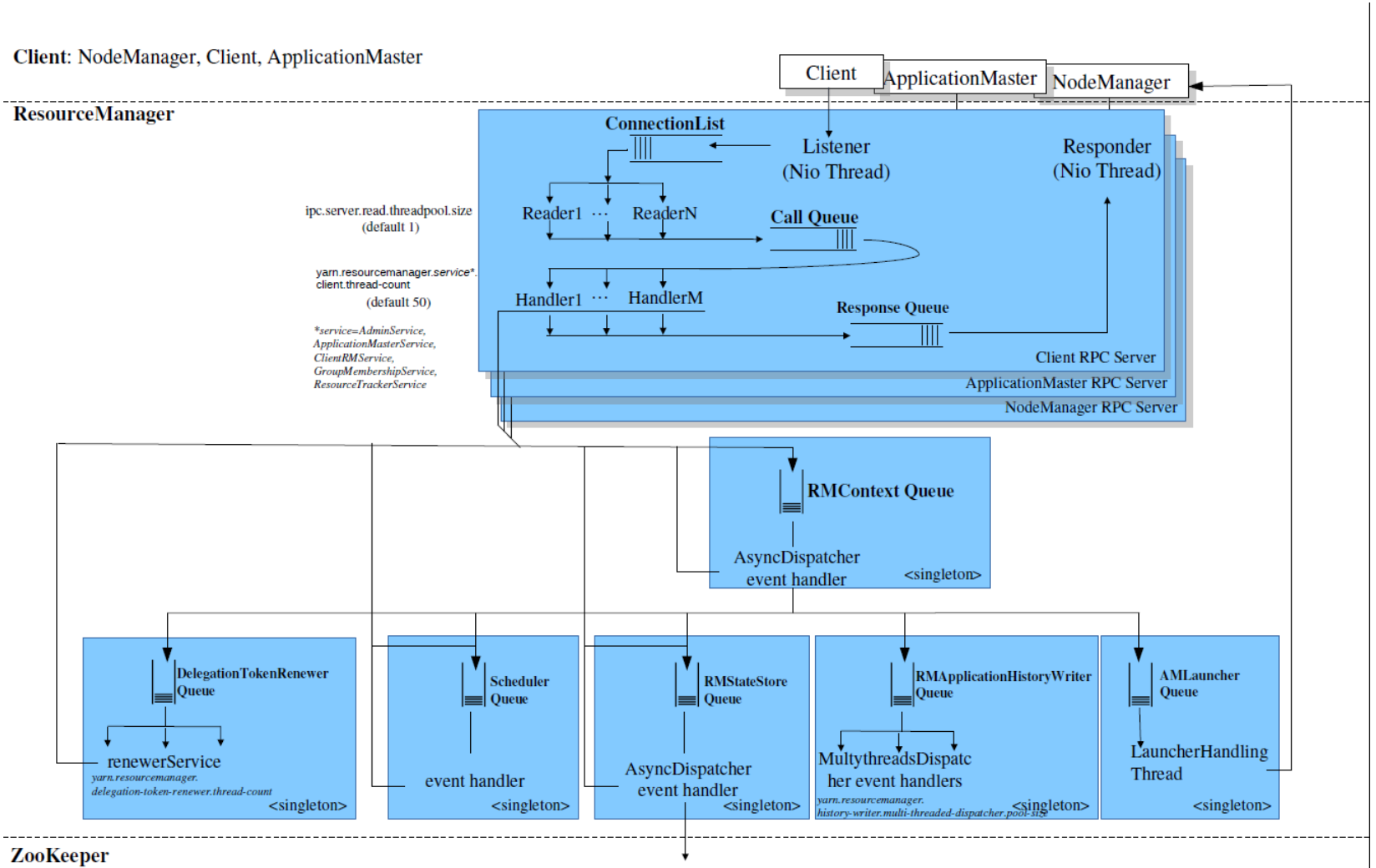
- Distributed, In-memory
- 2-Phase Commit
  - Replicate DB, not the Log!
- Real-time
  - Low TransactionInactive timeouts
- Commodity Hardware
- Scales out
  - Millions of transactions/sec
  - TB-sized datasets (48 nodes)
- Split-Brain solved with Arbitrator Pattern
- SQL and Native Blocking/Non-Blocking APIs

30+ million update transactions/second on a 30-node cluster

Standby is boring



Dificulties 1/2



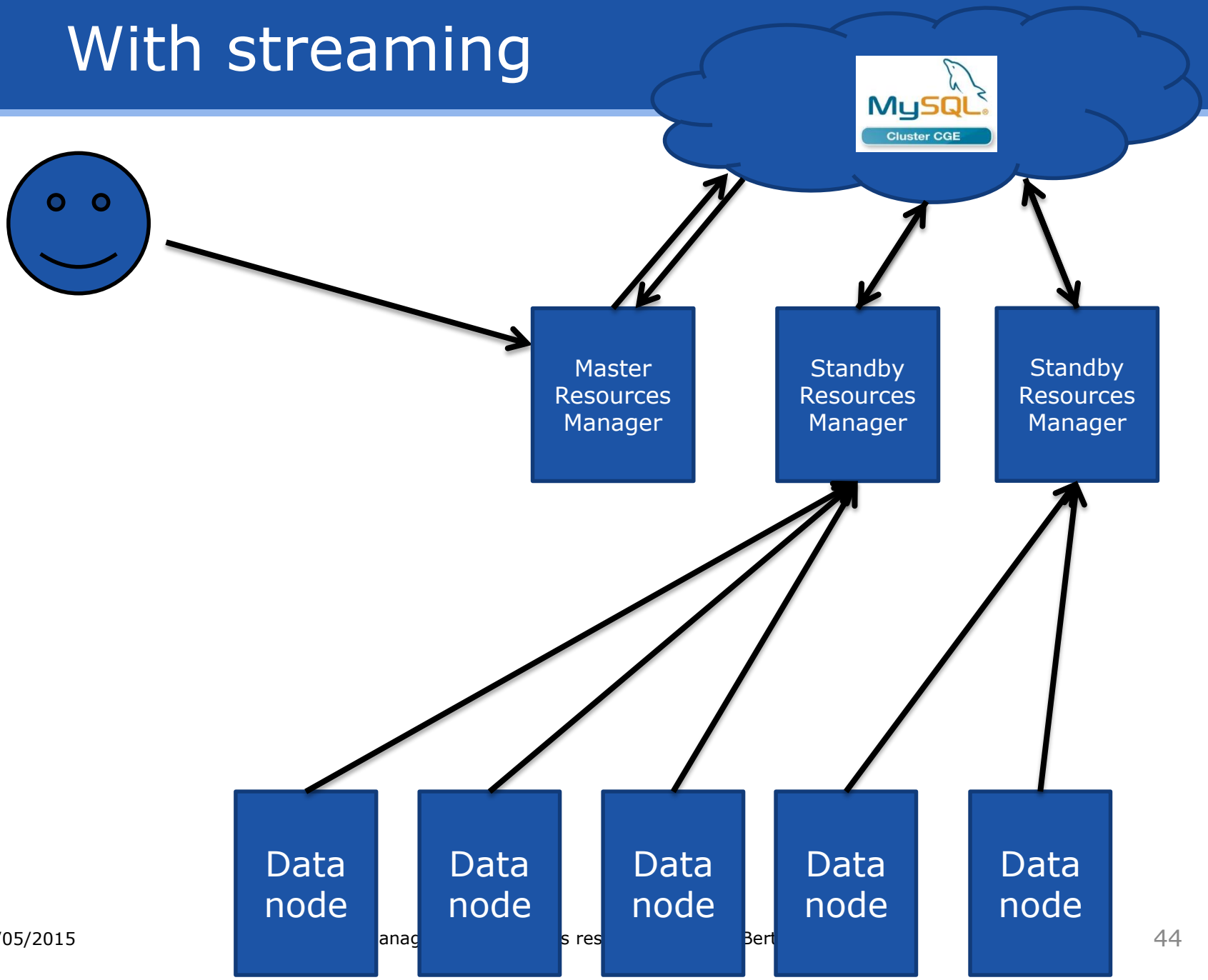
Difficulties 2/2

- Pulling from the database when the state is needed is inefficient.
- Having an independent thread that regularly pull from the database is difficult to tune and cause lock problems.

Solution

- Luckily NDB has an event API.

With streaming



Conclusion

- There exists three architectures for large cluster resource scheduling:
  - Monolithic
  - Two-levels
  - Shared State
- Each of these architectures has pros and cons.
- The monolithic architectur is the one presently used because it is easyer to use and develop.

- At KTH and SICS we are exploring the possibilities for a new architecture ensuring more scalability while keeping the advantages of the monolithic architecture.

References

- *Reservation-based Scheduling: If you're late don't blame us!*, C. Curino & al., Microsoft tech-report
- *Omega: flexible, scalable schedulers for large compute clusters*, Malte Schwarzkopf & al., EuroSys'13
- *Apache Hadoop YARN: Yet Another Resource Negotiator*, V. K. Vavilapalli & al., SoCC'13.
- *Large-scale cluster management at Google with Borg*, A. Verma & al., EuroSys'15.
- *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*, B. Hindman & al., NSDI'11