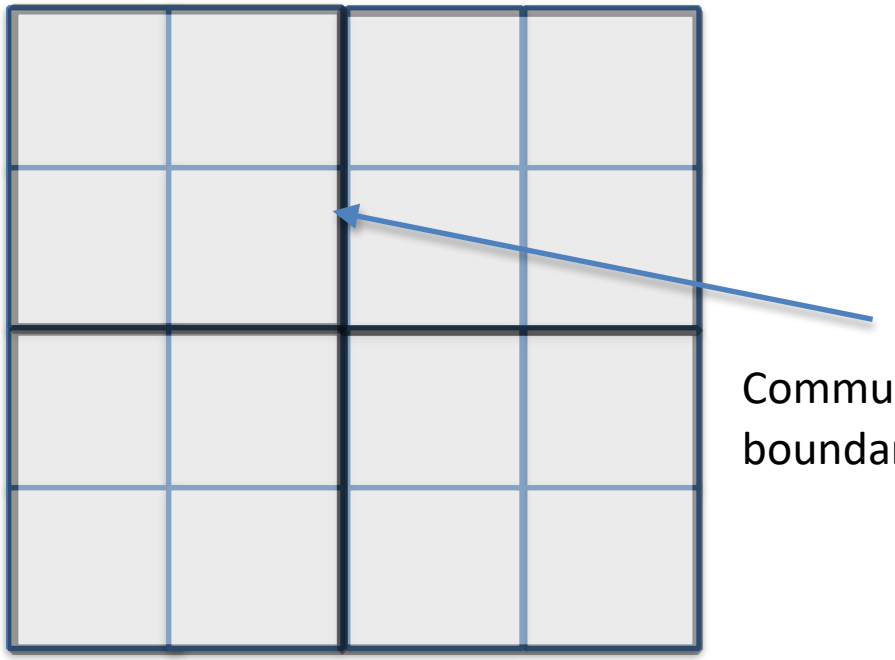
UPPSALA
UNIVERSITET

”Scalable applications”

UPPSALA UNIVERSITET

Tightly coupled applications?

- Common in CSE, typical examples are Partial Differential Equation (PDE) solvers



UPPSALA UNIVERSITET

HPC programming models

- Message Passing Interface (Distributed Memory)
- OpenMP, Pthreads etc. (Shared Memory)
- CUDA, OpenCL (GPGPU)
- FGPA

- Low-latency interprocess communication is critical to application performance
- Specialized hardware/instruction sets.
- “Every cycle counts”, users of HPC become concerned about VM overhead
- Failed tasks causes major problems/performance losses
- Compute infrastructure organized accordingly
- ’High-Performance Computing, HPC cluster resources. Wait in a queue to obtain dedicated direct access to high-bandwidth hardware resources.

UPPSALA UNIVERSITET

High-Throughput Computing (HTC)

While HPC is typically aiming to use a large amount of resources to solve few large, tightly coupled tasks in a short period of time on homogenous and co-localized resources, HTC is typically concerned with

- Solving a large number of (large) problems over long periods of time (days, months, years)
- on distributed, heterogenous resources
- Grid Computing developed largely for this application class
- Task throughput measured over months to years

http://research.cs.wisc.edu/htcondor/htc.html

UPPSALA UNIVERSITET

Many-Task Computing (MTC)

Bridges the gap between HPC and HTC

- Solve a large number of loosely coupled tasks on a large number of resources in a fairly short time
- Can be a mix of independent and dependent tasks
- Throughput typically measured in tasks/s, MB/s etc.
- Often not so clear distinction between HTC and MTC

UPPSALA UNIVERSITET

Vertical vs. Horizontal Scaling

Vertical scaling ("scale-up"):

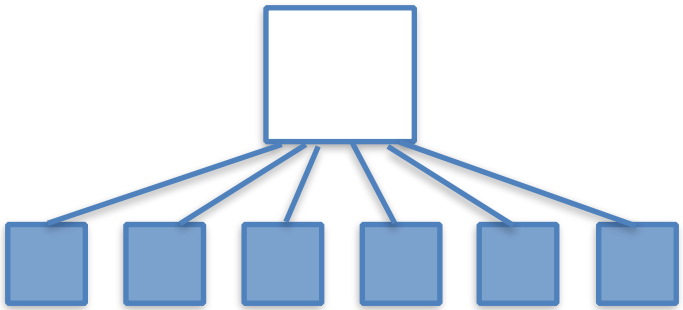
Increase the capacity of your servers to meet application demand.

- Add more vCPUs
- Add more RAM/storage

Network a potential performance bottleneck	High pressure on critical component (robustness)

Horizontal scaling ("scale-out"):

Meet increased application demand by adding more servers/storage/etc.



UPPSALA UNIVERSITET

Vertical vs. Horizontal Scaling

Vertical scaling ("scale-up"):

Increase the capacity of your servers to meet application demand.

- Add more vCPUs
- Add more RAM/storage
- Fundamentally limited by HW capacity of the underlying host
- Cost increases sharply for high-end machines

Horizontal scaling ("scale-out"):

Meet increased application demand by adding more servers/storage/etc.

- Cost of adding another machine of the same (commodity) type scales linearly

UPPSALA UNIVERSITET

How to measure performance/ efficiency?

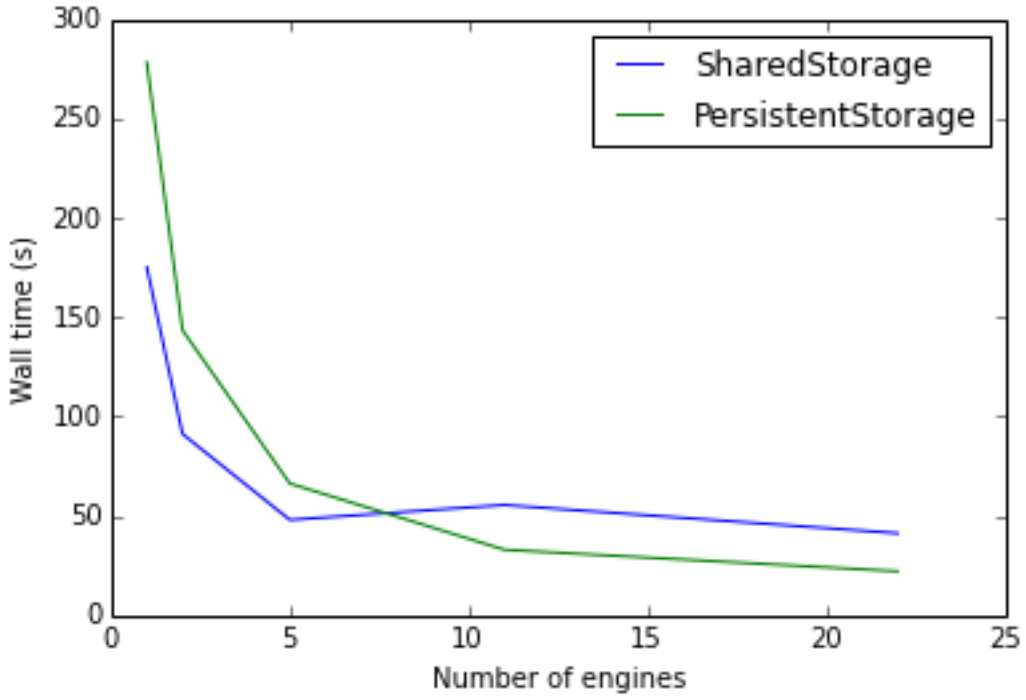
- Traditional HPC: FLOPs, execution time
- Recent HPC/Multicore trend: Energy efficiency (green computing)
- Cloud computing: Same, but in addition cost due to the pay-as-you-go model.
    - “What does my computational experiment cost?”
    - Non-trivial to measure/project
    - Depends on CPU/hours, Number of API requests, Write/Read OPs on Volumes, GB traffic into and out of Object Store etc…

UPPSALA UNIVERSITET

Strong scaling

Strong scaling: How does the execution time decrease as a function of increased resources, *for a fixed problem size?*

How quickly can I solve a problem of this size by adding more HW?

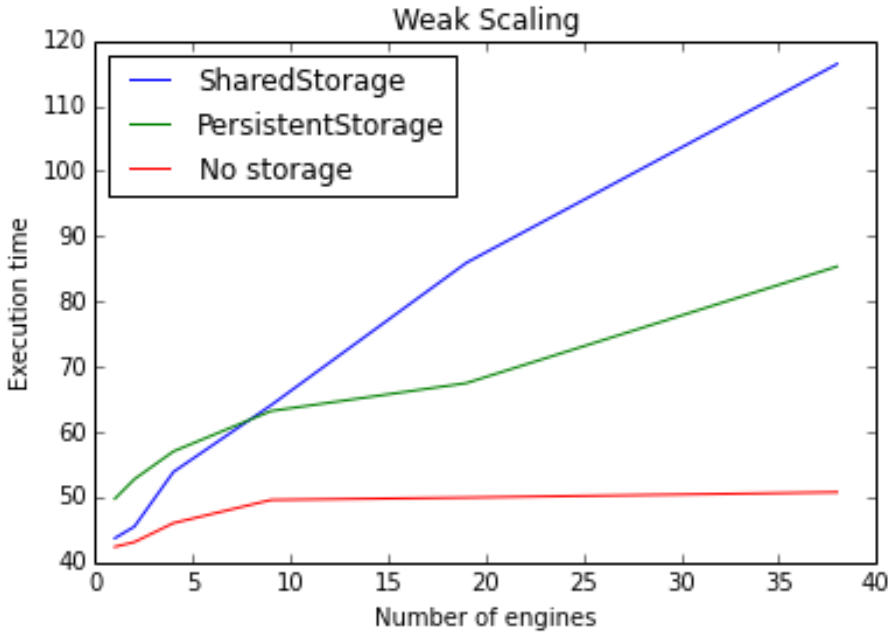


UPPSALA UNIVERSITET

Weak Scaling

Weak scaling: How does the execution time behave as a function of increased resources, *for an increasing problem size? (Fixed amount of work per processor)*

How much can I grow my problem size and still maintain the same execution time?



UPPSALA UNIVERSITET

Key concepts

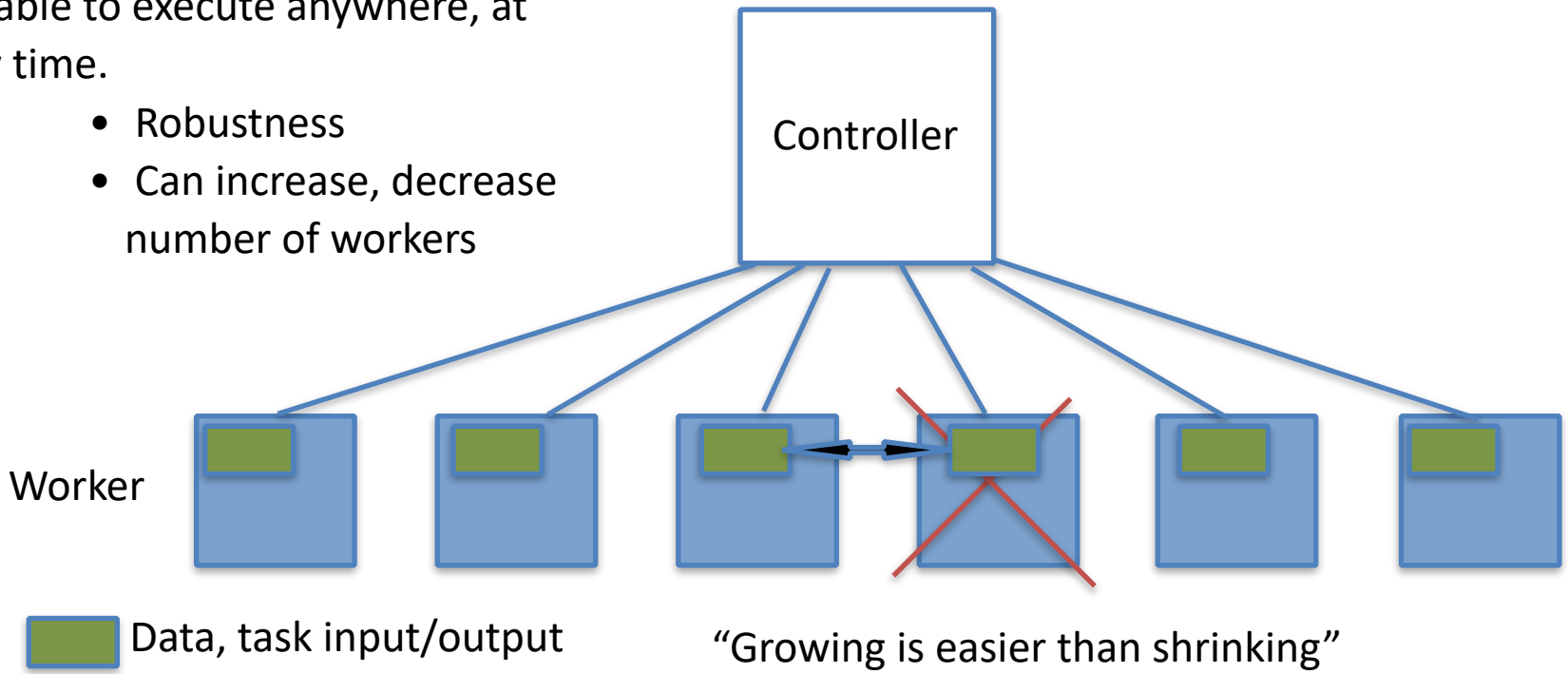
- Vertical scaling
- Horizontal scaling
- Strong scaling
- Weak scaling

UPPSALA UNIVERSITET

Strive for statelessness of task/ component

Ideally you want your task to be able to execute anywhere, at any time.

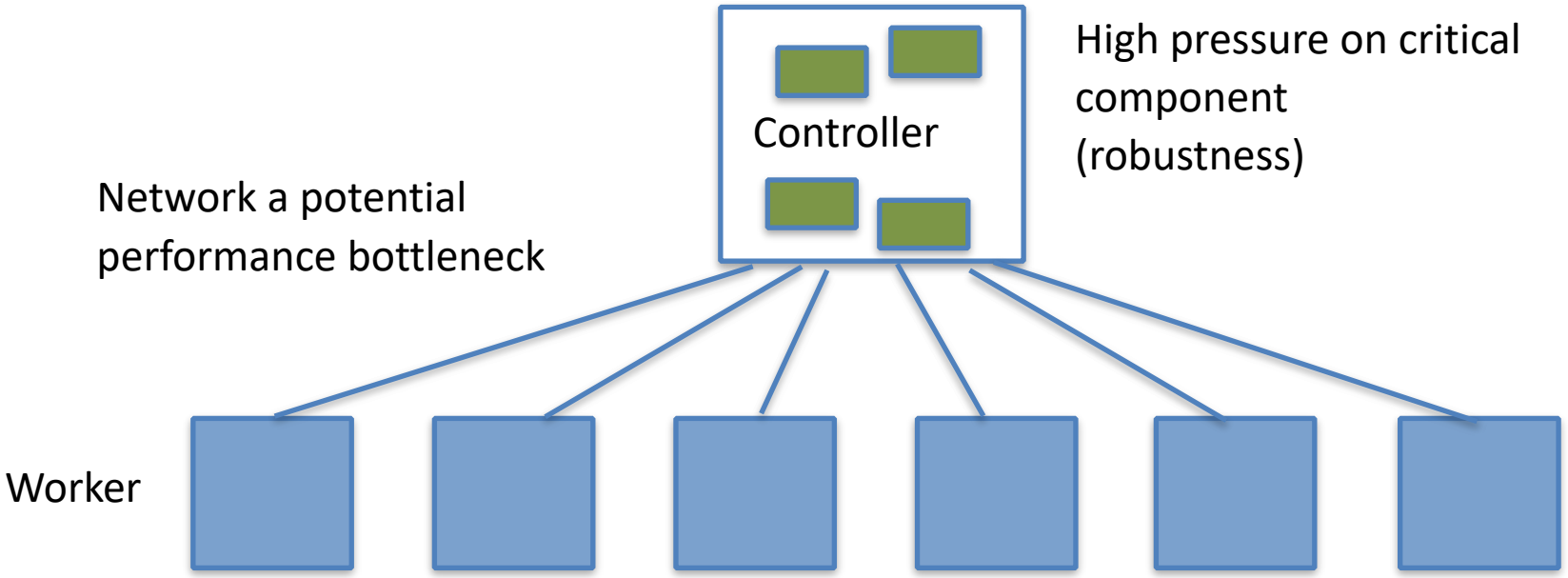
- Robustness
- Can increase, decrease number of workers

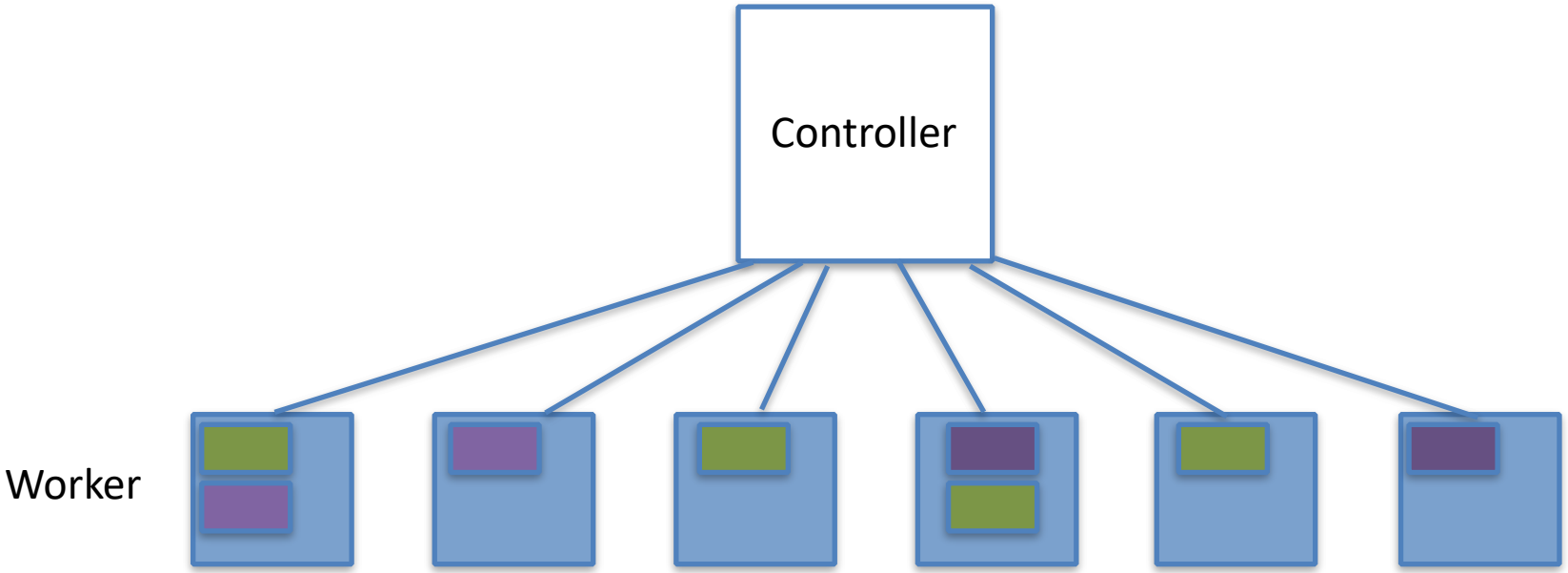


But of course, all computations/applications need data and state. It is not so much about if you store it as it is about where you store it.

UPPSALA UNIVERSITET

Is the controller machine designed for scalable storage?





Robustness/scalability through replication.

Another option for robustness, store replicas of data over multiple hosts in a distributed filesystem.

- HDFS (Hadoop, Big Data)
- Does not decouple compute/storage lifetimes as easily
- Designed for high I/O throughput

UPPSALA UNIVERSITET

The reactive manifesto

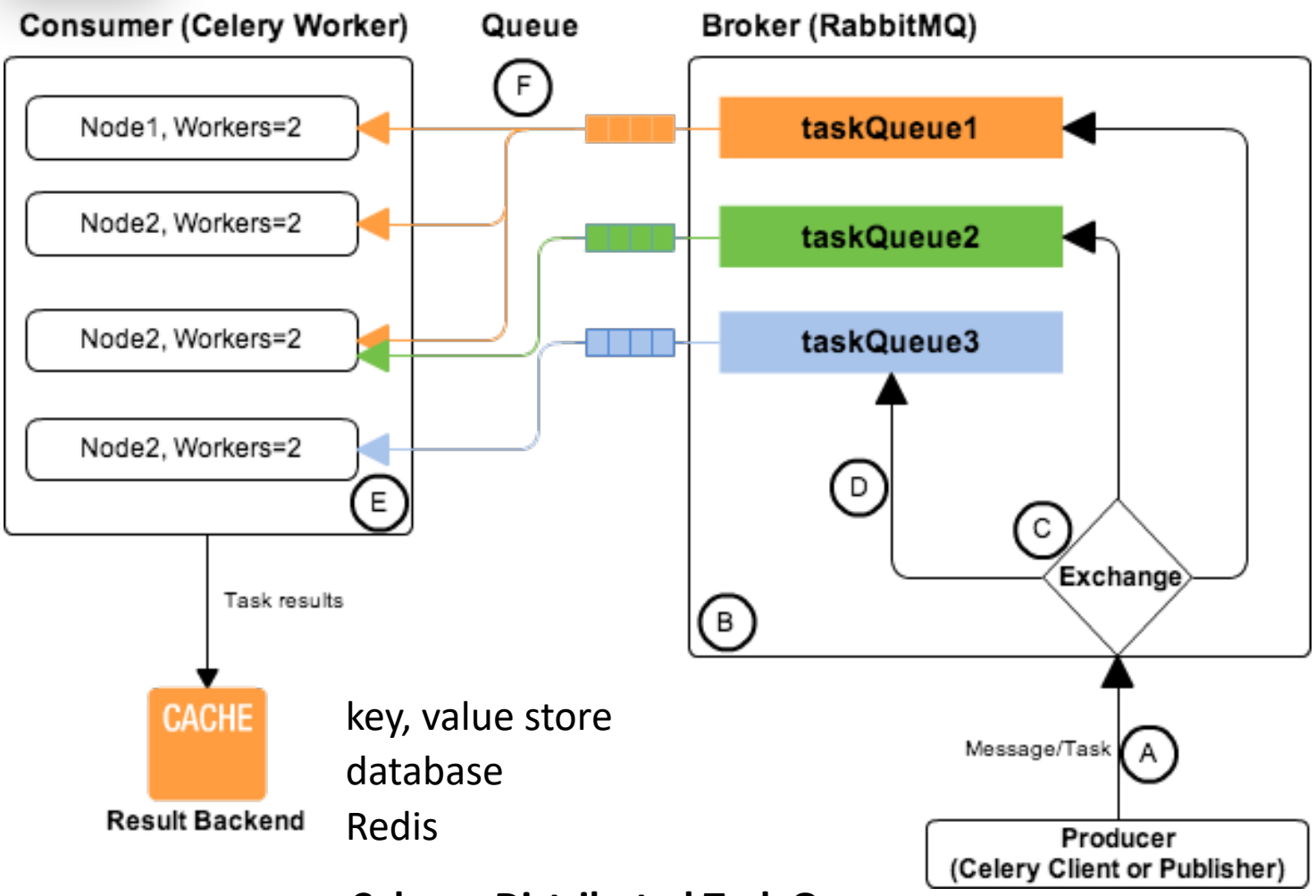
http://www.reactivemanifesto.org/

Reactive systems are:

- Responsive
- Resilient (Stays responsive in events of failure)
- Elastic (avoid contention and central bottlenecks)
- Message Driven (Rely on asynchronous message passing)

Applications designed to adhere to these principles will map well to the cloud computing paradigm - we often call such applications *“cloud native”.*

Example: Celery/RabbitMQ



Celery - Distributed Task Queue

http://abhishek-tiwari.com/post/amqp-rabbitmq-and-celery-a-visual-guide-for-dummies#fn:1

UPPSALA UNIVERSITET

A flexible computational framework using R and Map-Reduce for permutation tests of massive genetic analysis of complex traits

Citation information: DOI 10.1109/TCBB.2016.2527639, IEEE/ ACM Transactions on Computational Biology and Bioinformatics

Authors: Behrang Mahjani, Salman Toor, Carl Nettelblad, Sverker Holmgren

UPPSALA UNIVERSITET

QTL Analysis

- Abbreviation of “ Quantitative Trait Loci ”
- Understanding the relation between genes and traits is a fundamental problem in genetics.
- Such knowledge can eventually lead to e.g. the identification of possible:
  - drug targets,
  - treatment of heritable diseases, and
  - efficient designs for plant and animal breeding.
- QTL mapping is the way to locate such regions in genome.

UPPSALA UNIVERSITET

Analysis Framework

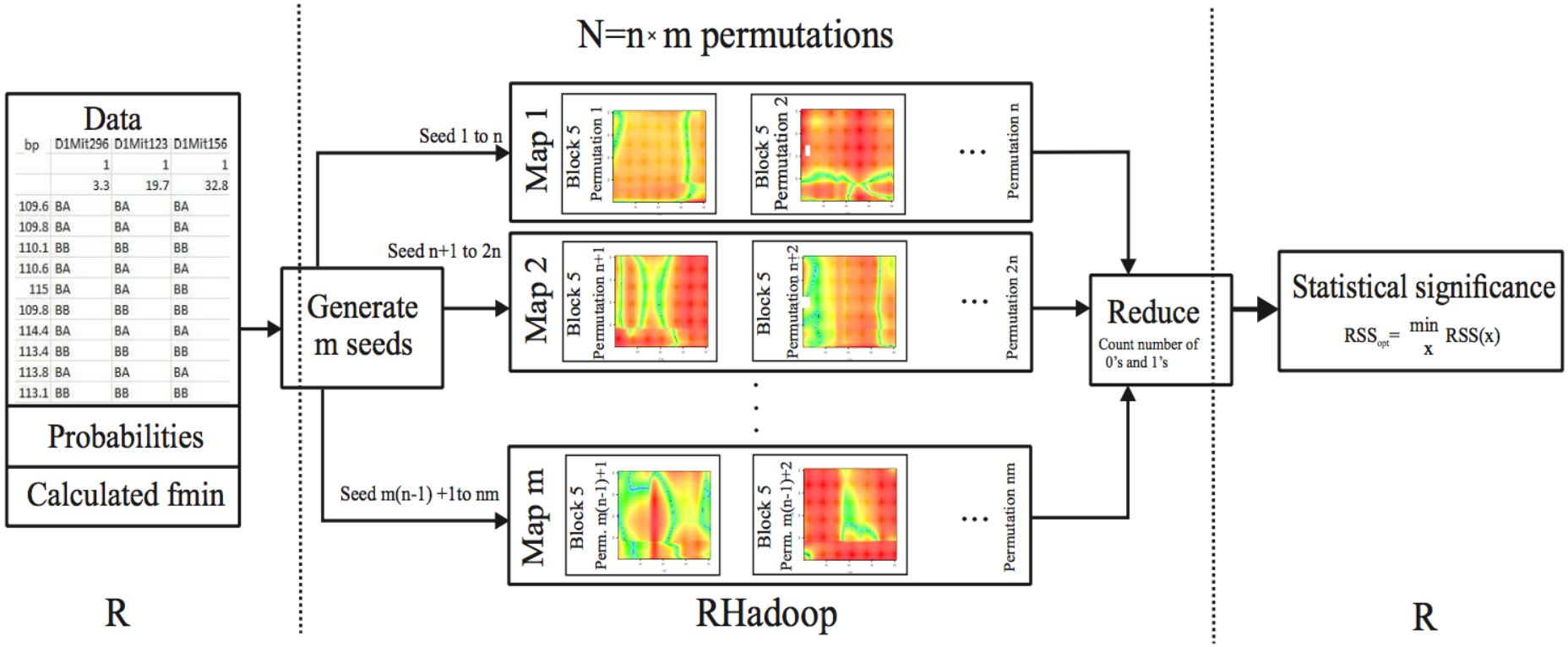


Fig. 4: Distributed computing of the significance level using map-reduce setting; Details of block 4

UPPSALA UNIVERSITET

Results

TABLE 1: 10^4 permutations, PruneDIRECT is using the global minimum from true QTL

	Exhuastive	DIRECT	PruneDIRECT
Total time (hours)	70	15	8
Average of FE.*	531441	16867	63
Standard division of FE.*	0	2239	1555

*FE. is the number of function evaluations.

TABLE 2: 2×10^5 permutations, PruneDIRECT is using the global minimum from true QTL

	PruneDIRECT
Total time (hours)	15
Average of FE.*	62
Standard division of FE.*	1650

*FE. is the number of function evaluations.

TABLE 3: 640 permutations to analyze the scalability of the proposed framework.

Number of virtual CPUs	1	4	8	16	32
Total time in hours	39.9	11.02	5.83	2.88	1.50