Large Scale Systems Architecture (1)

Grid Computing (M)

Lecture 7

UNIVERSITY
of
GLASGOW



From Distributed Systems to Grids

- What is Grid Computing?
- Why is it different from traditional distributed systems?

- Consider:
  - Analysis of data from the Large Hadron Collider
  - Distributed file system for an office environment

  What key features of the former which make it a grid computing environment rather than a distributed system?
  - Multiple autonomous institutions; collaborative data collection and storage
  - Enormous size of data repository and computational analysis tasks
  - Need for reliability, robustness and data provenance

  Similar systems becoming more common
  - e-science (physics, bioinformatics), oil exploration, financial modelling, etc.

Copyright © 2006 University of Glasgow

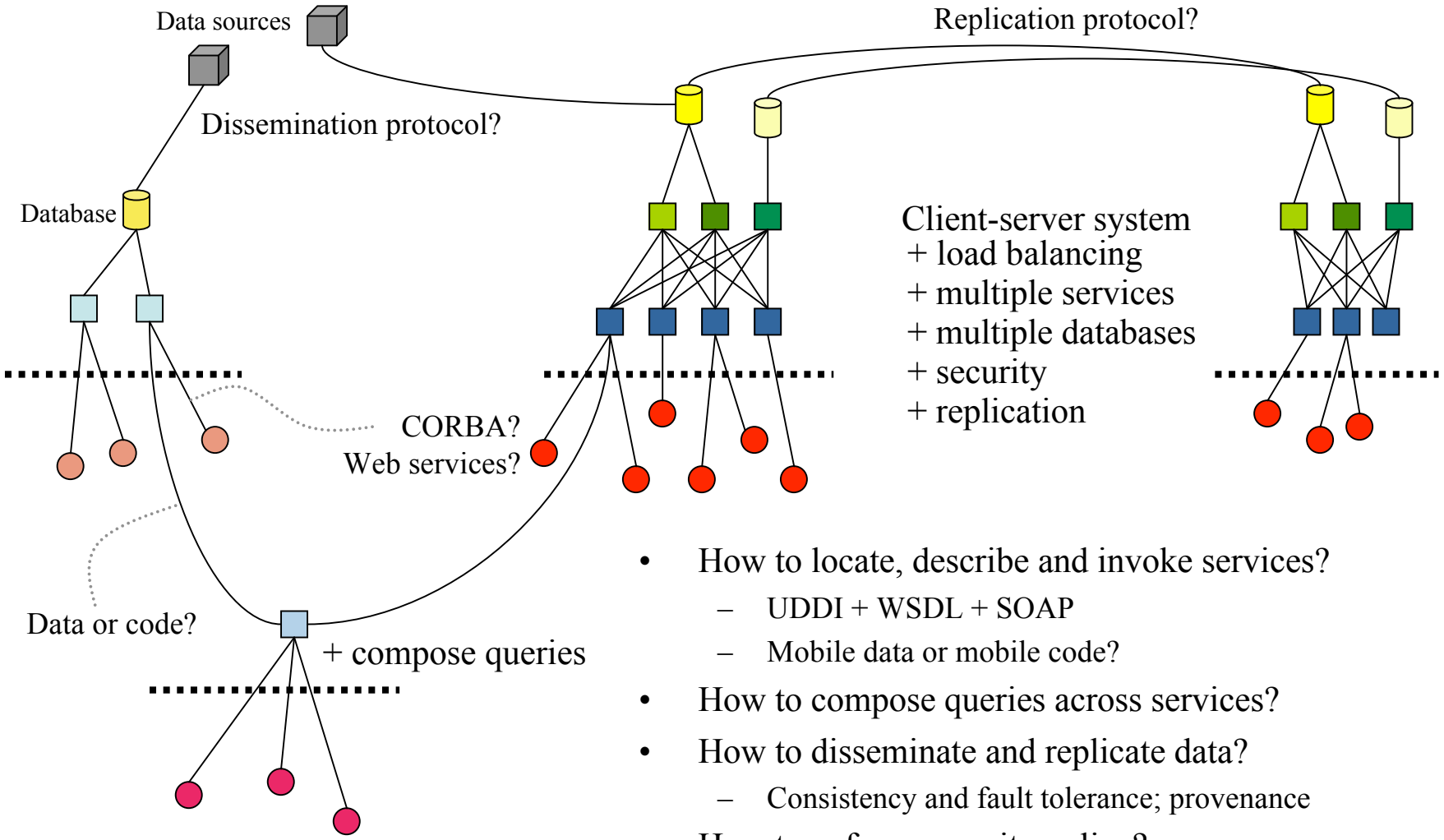
Key Features and Concerns in Grids

- Autonomy
- Scalability
  - Data
  - Communication and computation
  - Organisational
- Heterogeneity
  - Host and network architecture
  - Data format
  - Organisational
- Security
  - Authentication
  - Authorisation
  - Accounting
- Fault Tolerance
  - Correctness and provenance
  - Consistency and repeatability
  - Graceful degradation
  - System resilience vs. fault tolerance

What are the implications on system architecture?

Copyright © 2006 University of Glasgow

Client-Server Systems and the Grid



- How to locate, describe and invoke services?
  - UDDI + WSDL + SOAP
  - Mobile data or mobile code?
- How to compose queries across services?
- How to disseminate and replicate data?
  - Consistency and fault tolerance; provenance
- How to enforce security policy?
  - Authentication, authorisation and accounting
- How to manage, schedule and account for resources?

Copyright © 2006 University of Glasgow

Client-Server Systems and the Grid

- Distributed client-server model represents state of the art in grid computing systems
  - A "Grid Services" architecture
    - Traditional protocols re-purposed for Grid computing
      - Scheduled data dissemination via GridFTP
      - RPC services via UDDI + WSDL + SOAP
      - Distributed batch processing environments

      } Production ready technologies
    - Significant security challenges
      - Authentication, authorisation and accounting
      - Transitive trust; public key infrastructure
    - Significant organisational challenges
      - Business process modelling
      - Data provenance and integrity; robustness and fault-tolerance

      } New infrastructure and processes
  - Relatively static and planned infrastructure
- Tutorials will explore such systems
  - Globus toolkit, OGSA-DAI, Condor, PERMIS, Shibboleth, etc.

Copyright © 2006 University of Glasgow

Large Scale Systems Architecture

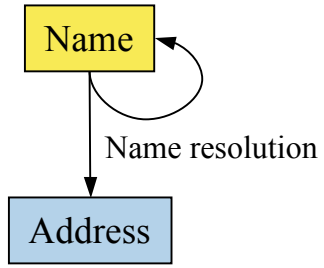
- Consider:
  - What are the fundamental principles of Grids?
  - Are we building these systems the right way?

- Is a distributed client-server model appropriate for the long term?

- Alternative architectures for:
  - Naming and addressing
  - Discovery and routing
  - Availability and robustness
  - Automatic parallelism
  - Timing and ordering constraints
  - Strong vs. weak coupling

Copyright © 2006 University of Glasgow

Naming and Addressing



- Name: an identifier for the object
- Address: the location of an object

- Name resolution gives an address for an object, based on its name
  - Or another name, to resolve recursively
  - Names are sometime also addresses, encoding location information:

    `http://www.ietf.org/rfc/rfc4042.txt`

    `urn:ietf:rfc:4042`

- Questions for discussion:
  - To what extent is it useful to embed location information into names?
  - To what extent is it useful to embed structure into names?
  - To what extent is it useful to have a single naming authority?
  - Advantages/disadvantages of each?

Copyright © 2006 University of Glasgow

Name Ownership

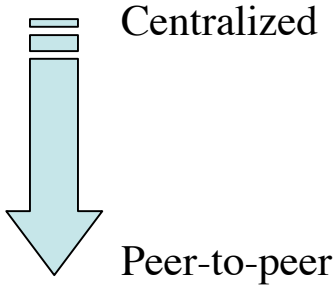
- Who controls the namespace?
  - A hierarchical space
    - Ownership of the root
    - Partitioning the space
      - Autonomy amongst partitions
  - A flat namespace
    - Centralised ownership
    - No ownership
- What is the name allocation policy?
- Political vs. technical issues
  - “Internet governance”
  - Formal vs. informal naming
    - Ontologies for e-science
    - Flickr™ tags for photographs
- What is the trade-off between the desire for a well-defined namespace and the need for organisational autonomy?

Copyright © 2006 University of Glasgow

Discovery and Routing

- Given the name of an object, efficiently locate that object
- Desirable features:
  - Scalable to large systems, many objects, many queries
  - Fault tolerant, degrades gracefully
  - Allows unstructured names (to support any type of data)

- Approaches:
  - Centralized name service
  - Distributed hierarchical name service
  - Distributed flooding
  - Distributed hash table

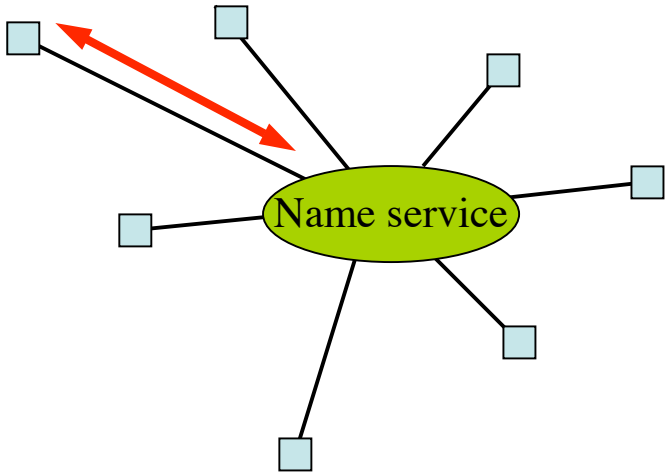


  - What are constraints of each approach?

Copyright © 2006 University of Glasgow

Centralized Name Service

- Nodes advertise names of objects they hold to central name service
- Searches resolved by that service
  - Allows unstructured names; any host can hold any object



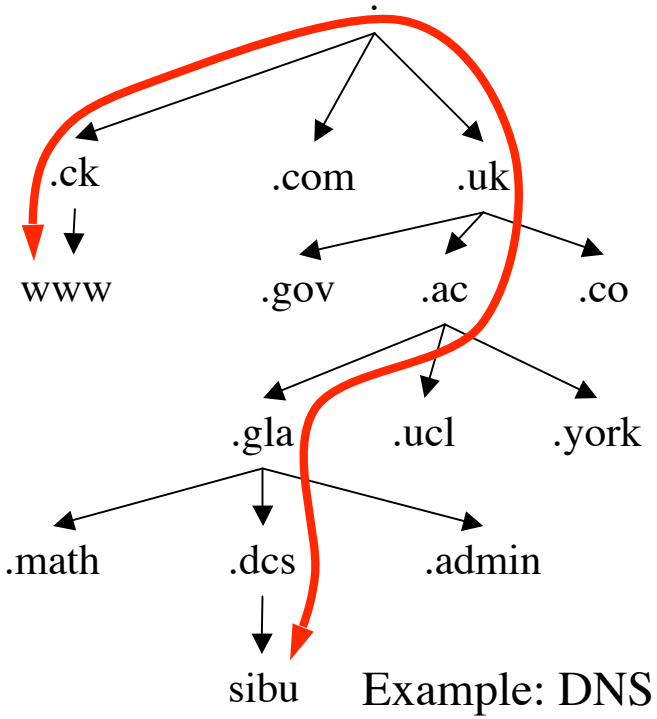
- Problems:
  - Doesn't scale
  - Single point of failure
  - Centralized control

- Widely used: Globus, CORBA, UDDI, etc…

Copyright © 2006 University of Glasgow

Distributed Hierarchical Naming

- Assign hierarchical names to objects
  - Delegate portions of the namespace to different entities/organizations
- Build a search tree
  - Each node knows its parent and children
  - Search for key ascends towards the root, then descends into the tree



Example: DNS

- Assumptions:
  - Structured hierarchical namespace
  - Object ownership delegated to autonomous organizations matching naming hierarchy
    - Scalable name lookup
  - Single root ⇒ centralized control

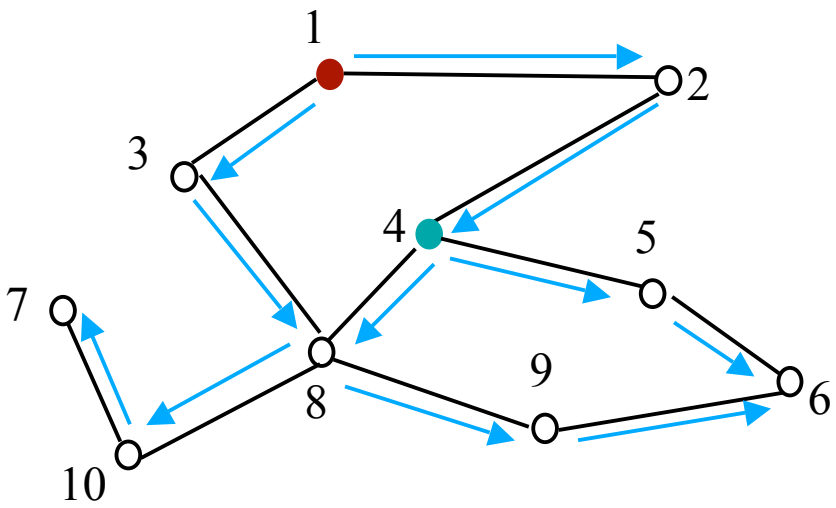
Copyright © 2006 University of Glasgow

Distributed Flooding

- Every node forwards packets to all of its neighbors
  - Lifetime of packets are limited by time-to-live
  - Packets have unique identifiers to detect loops
- Allows unstructured names
- Simple, robust, but generates large amounts of traffic
- Example: Gnutella

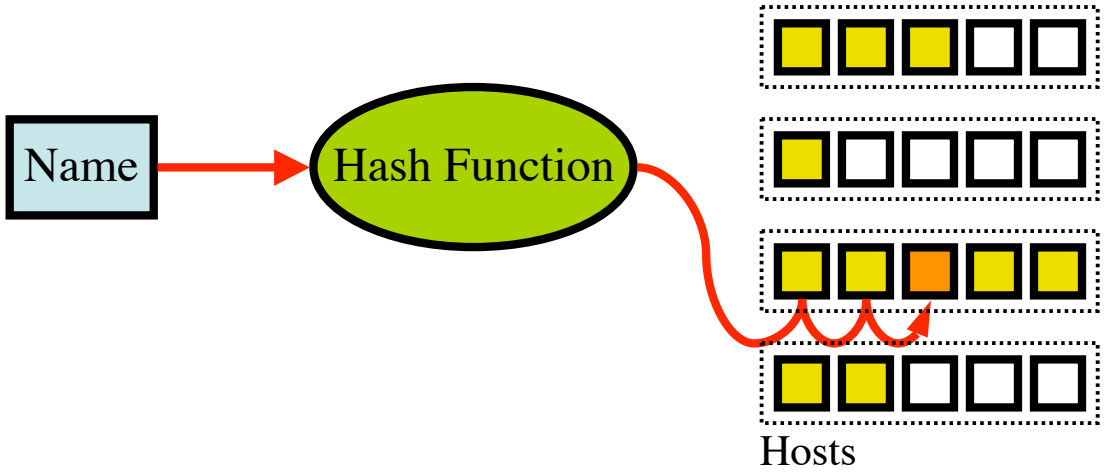
Node 1 initiates search

Data floods network, even after result found at node 4



Copyright © 2006 University of Glasgow

Distributed Hash Table



- Passes the name through a *hash function* which maps it to a next hop host; iterate to locate object
  - Potentially flat unstructured names; location encoded via hash function
- Scaling to large numbers of hosts and names possible
  - e.g. millions of hosts in “file sharing” peer-to-peer networks

- Location independent request routing:
  - Initiator of request *does not know* where named object located

Copyright © 2006 University of Glasgow

Discovery and Routing

- Distributed flooding and DHT introduce a new concept:
  - Location of named object implicit in the lookup/routing process
  - Requesting host doesn't know or care where named object resides
  - Interact with an unknown group of objects, which share some characteristic

- Implications for large scale systems:
  - Flexibility
  - Scalability
  - Robustness
  - Powerful programming abstraction
    - Ability to name groups of objects with unknown membership, but common characteristics
    - Ability to name replicas without caring where/how replicated

Copyright © 2006 University of Glasgow

Availability, Robustness, and Coupling

- Location independent naming ⇒ automatic replication trivial

- How is node failure handled?
  - Does it matter?
    - Query will find another replica anyway
    - Computation is atomic, can be repeated
  - Failures inevitable as system size increases

- How are errors repaired? How tightly coupled is the system?
  - How consistent must the data in a grid computing system be?
    - How to resolve incompatible changes after network partition?
    - Probabilistic forward error correction
    - Probabilistic retransmission, avoiding implosion
  - Acceptable to use gossip/epidemic protocols?
  - Distribute data to hosts with some probability?

Copyright © 2006 University of Glasgow

Timing and Ordering Constraints

- What guarantees are provided on operation timing?
  - Total ordering? Partial ordering?
  - Overhead of maintaining a clock?
- Arbitrate and serialise resource access
  - Distributed locking
  - Distributed transactions
- To what extent do traditional locking techniques apply?
  - Web services require explicit locking, ordering
  - Are there programming styles that implicitly order resource access?
    - Lock-free algorithms
    - Functional programming styles

Copyright © 2006 University of Glasgow

Automatic Parallelism

- To what extent does the programmer manage parallelism?
  - Web services model: explicitly code parallelism in RPC requests
    - But perhaps the called services hide parallelism internally?
    - Similar to traditional object-oriented programming
  - Alternatives:
    - Active agents, spread via flooding, etc.
    - Invoke services on named, but unknown group, sharing characteristics

- How to aggregate/process results, avoiding implosion?

Copyright © 2006 University of Glasgow

Peer-to-Peer Systems

- Peer-to-peer and multicast protocols address many of these issues
  - Very large scale (~millions of nodes)
  - Fault tolerant, robust, graceful degradation
  - Weakly coupled
  - Expressive naming architectures
- An alternative to “traditional” computational grids?

- Motivating examples:
  - MapReduce (large scale cluster computing)
  - Astrolabe (distributed monitoring)
  - Ocean Store (distributed file store)
  - …

Copyright © 2006 University of Glasgow

Example: Google MapReduce

- Problem: conceptually simple calculations on large data sets
  - Indexing the web
  - Statistical analysis of large sets of web pages
- Must run across large clusters for reasonable performance
  - Distribute data collection, storage
  - Distribute computational load

- What is the right programming abstraction?
  - The functional *map* and *reduce* operations, implemented as a parallel library

Copyright © 2006 University of Glasgow

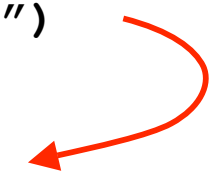
Example: Google MapReduce

```
map(k1, v1) → list(k2, v2)
reduce(k2, list(v2)) → list(v2)
```

Example – word count:

```
map(String key, String value)
    // key: document name
    // value: document contents
    foreach word in value
        EmitIntermediate(word, "1")

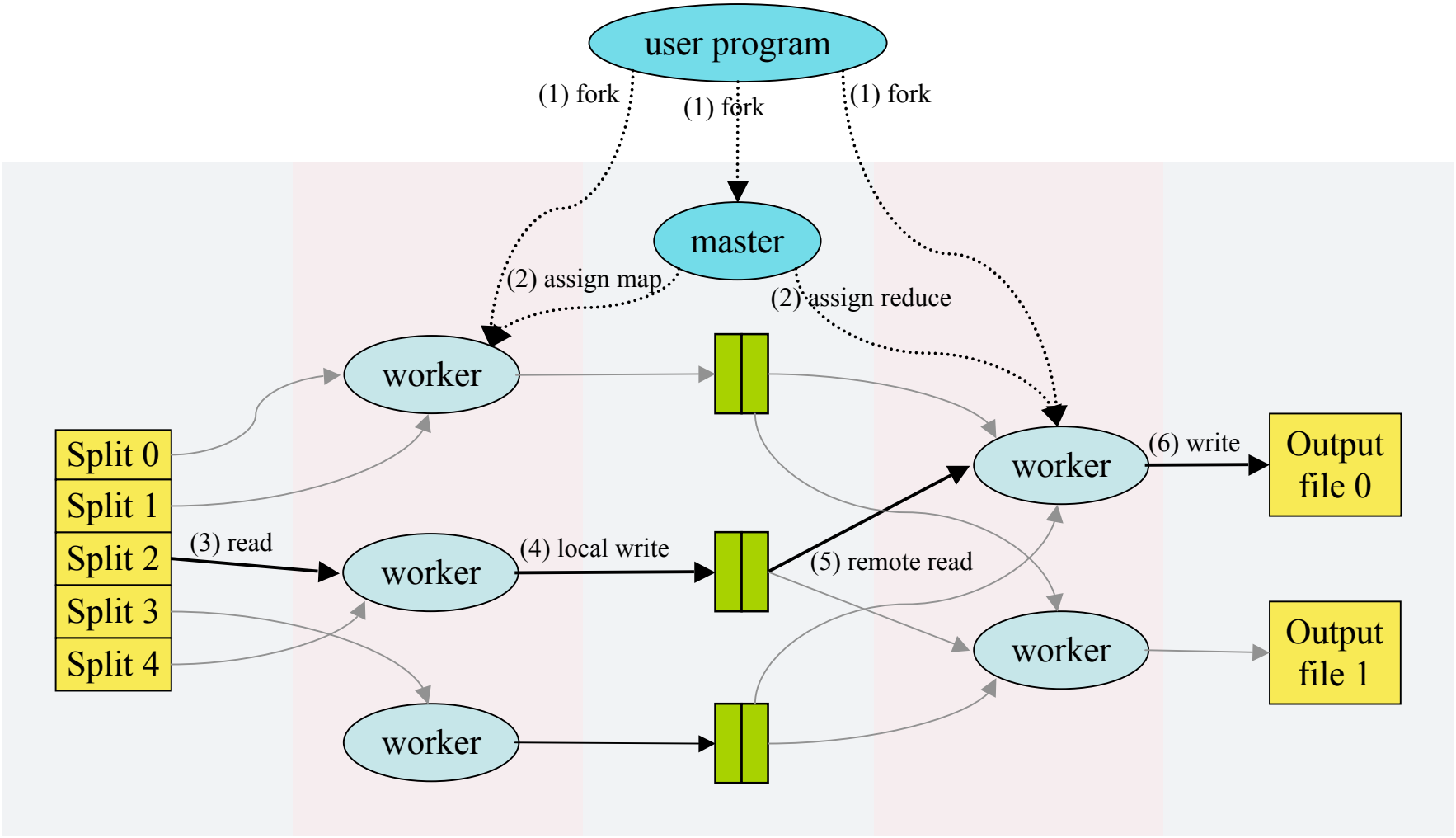
reduce(String key, Iterator values)
    // key: a word
    // values: a list of counts
    int result = 0
    foreach v in values
        result +=ParseInt(v);
    Emit(AsString(result));
```



MapReduce library groups intermediates with equal keys, passes to `reduce()`

Copyright © 2006 University of Glasgow

Example: Google MapReduce



Copyright © 2006 University of Glasgow

Example: Google MapReduce

- Benefits:
  - Automatic parallelism with *no* programmer conceptual effort
  - Very robust fault tolerance model
  - Scales to thousands of nodes; terabytes of data

- Consider:
  - How are inputs, intermediate, worker processes results named?
  - How are results, computations ordered?
  - How are failures handled?

- What can we learn from this design when building other large scale distributed systems?

Copyright © 2006 University of Glasgow

Summary

- From distributed systems to grids
- Naming and its impact on system architecture
- Example:
  - Google MapReduce system
  - An alternative architecture to solve grid-like problems

Copyright © 2006 University of Glasgow