University of Oviedo

EN
English

Universidad de Oviedo

School of Computer Science

Runtime

SOFTWARE ARCHITECTURE

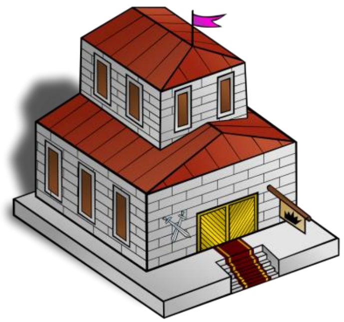
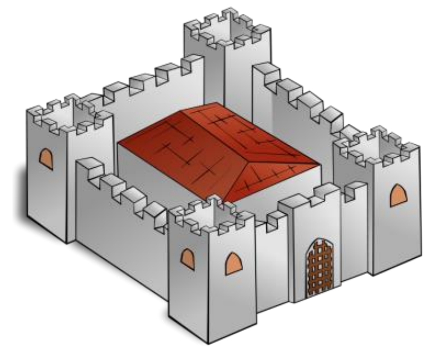
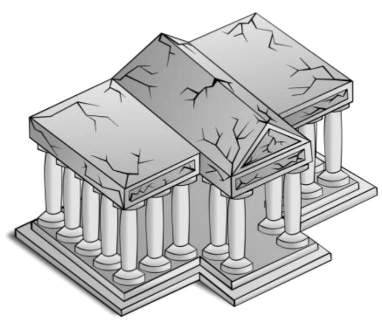
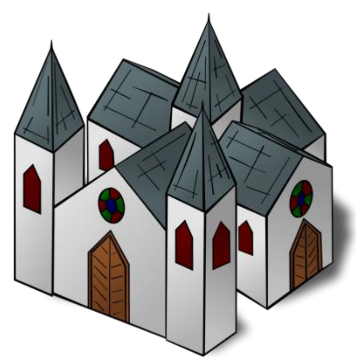
School of Computer Science

Course 2018/2019

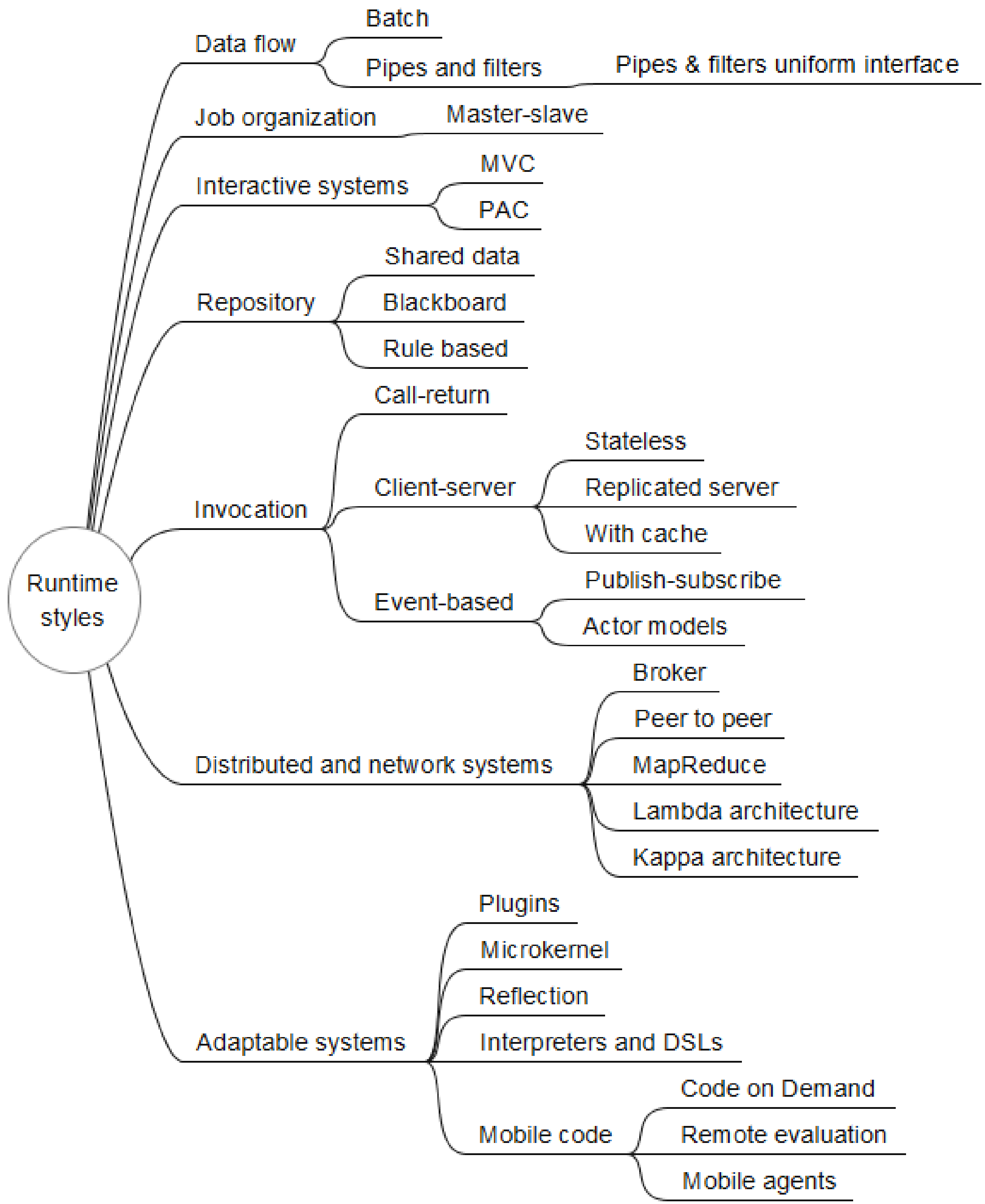
Jose E. Labra Gayo

Runtime behaviour

Also called: Components and connectors


Taxonomy


Runtime styles
Data flow
Batch
Pipes and filters
Pipes & filters uniform interface
Job organization
Master-slave
Interactive systems
MVC
PAC
Repository
Shared data
Blackboard
Rule based
Invocation
Call-return
Client-server
Stateless
Replicated server
With cache
Event-based
Publish-subscribe
Actor models
Distributed and network systems
Broker
Peer to peer
MapReduce
Lambda architecture
Kappa architecture
Adaptable systems
Plugins
Microkernel
Reflection
Interpreters and DSLs
Mobile code
Code on Demand
Remote evaluation
Mobile agents

Data flow

Batch

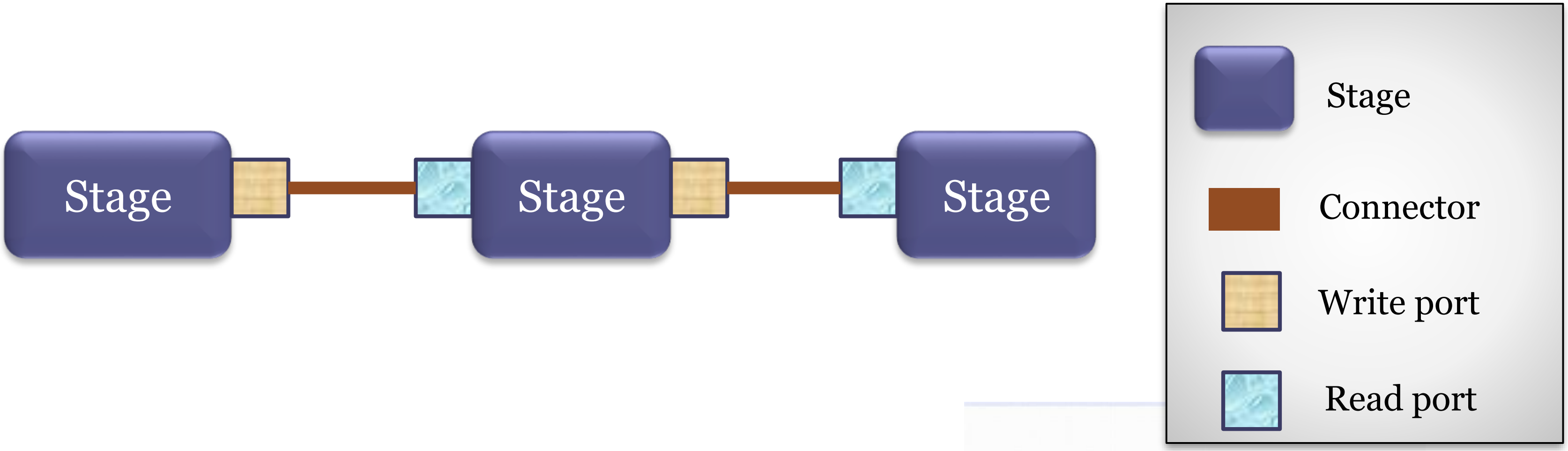
Pipes & Filters

- Pipes & Filters with uniform interface

Batch

Independent programs are executed sequentially

Data is passed from one program to the next



Note

Batch style = grandfather of software architectural styles

Batch

- Elements:
  - Independent executable programs
- Constraints
  - Output of one program is linked to input of the next
  - A program usually waits for the previous one to finish its execution

Batch

Advantages

- Low coupling between components
- Re-configurability
- Debugging
  - It is possible to debug each input independently

Challenges

- It does not offer interactive interface
- Requires external intervention
- No support for concurrency
  - Low throughput
- High latency

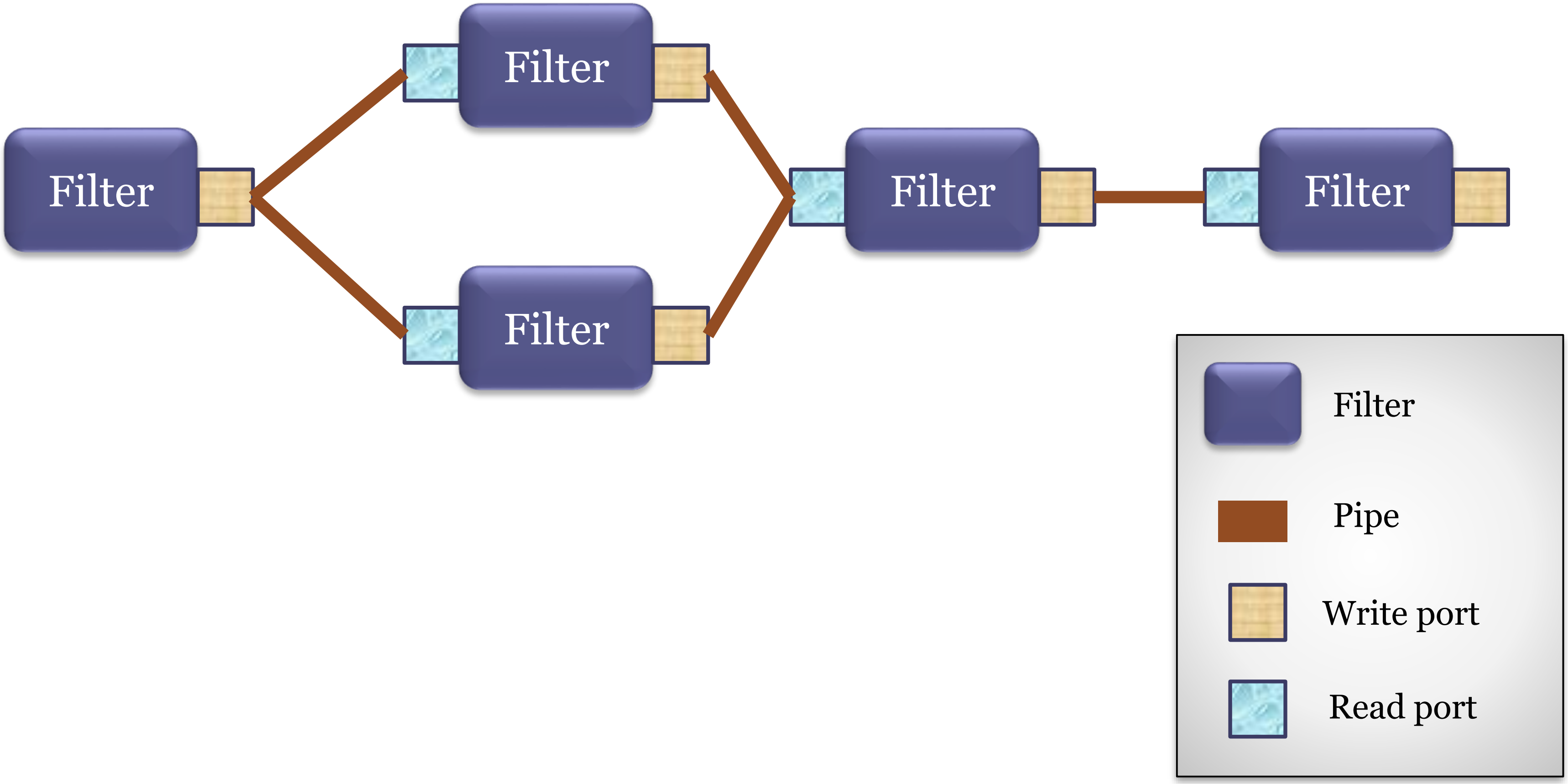
Definitions:

Throughput: rate at which something can be processed.
Example: number of jobs/second

Latency: time delay experienced by a process
Example: 2 seconds

Pipes & Filters

Data flows through pipes and is processed by filters



Pipes & Filters

Elements

- Filter: component that transforms data
  - Filters can be executed concurrently
- Pipe: Takes output data from one filter to the input of another filter
  - Properties: buffer size, data format, interaction protocol

Pipes & Filters

Constraints

- Pipes connect outputs from one filter to inputs of other filters
- Filters must agree on the exchange format they admit

Pipes & Filters

Advantages

- Better understanding of global system
  - Total behavior = sum of each filter behavior
- Reusability:
  - Filters can be recombined
- Evolution and extensibility:
  - It is possible to create/add new filters
  - It is possible to substitute old filters by new ones
- Testability
  - Independent verification of each filter
- Performance
  - It enables concurrent execution of filters

Challenges

- Possible delays in case of long pipes
- It may be difficult to pass complex data structures
- Non interactivity
  - A filter can not interact with its environment

Pipes & Filters

Examples & Applications

- Unix
  - who | wc -l
- Java
  - Clases java.io (PipedReader, PipedWriter)
- Yahoo Pipes

Pipes & Filters - uniform interface

- Variant of Pipes & Filters where filters have the same interface
- Elements
  - The same as in Pipes & Filters
- Constraints
  - Filters must have a uniform interface

Pipes & Filters - uniform interface

Advantages:

- Independent development of filters
- Re-configurability
- Facilitates system understanding

Challenges:

- Performance can be affected if data have to be converted to the uniform interface
  - Marshalling

Pipes & Filters - uniform interface

Examples:

- Unix operating system
  - Programs with a text input (*stdin*) and 2 text outputs (*stdout* y *stderr*)
- Web architecture: REST

Job organization

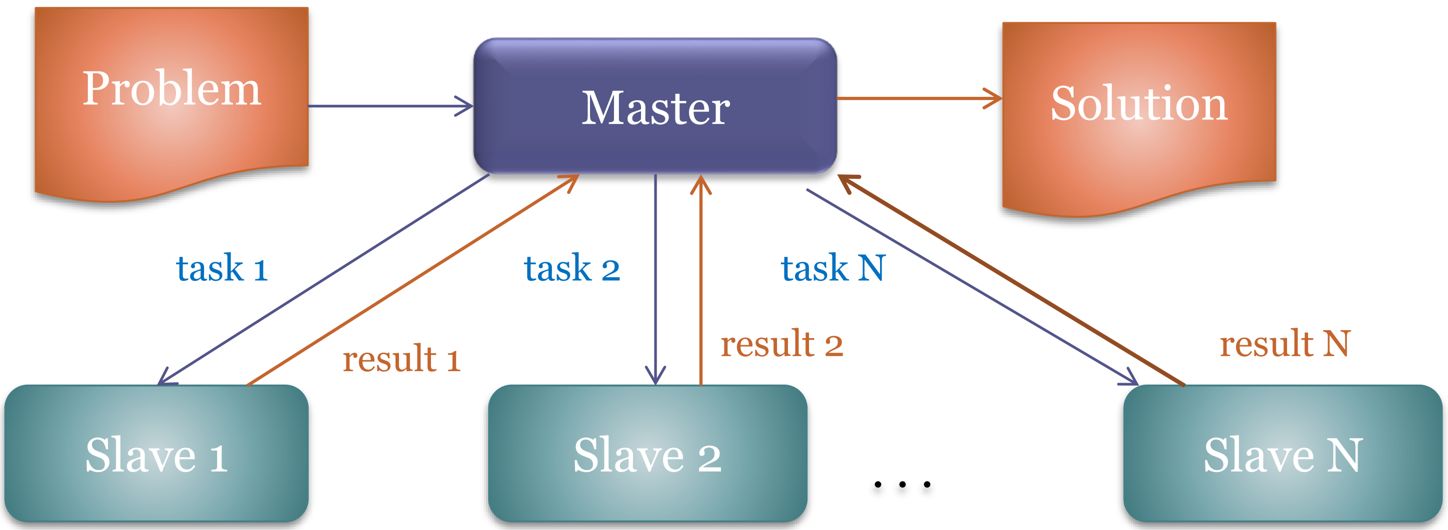
Master-Slave

Master-Slave

Master divides work in sub-tasks

Assigns each sub-task to different nodes

The computational result is obtained as the combination of the slaves results results



Master-Slave

- Elements
  - Master: Coordinates execution
  - Slave: does a task and returns the result
- Constraints
  - Slave nodes are only in charge of the computation
  - Control is done by the Master node

Master-Slave

- Advantages
  - Parallel computation
  - Fault tolerance
- Challenges
  - Difficult to coordinate work between *slaves*
  - Dependency on Master node
  - Dependency on physical configuration

Master-Slave

Applications:

- Process control systems
- Embedded systems
- Fault tolerant systems
- Search systems

Interactive systems

MVC: Model - view - controller

MVC variants

- PAC: Presentation - Abstraction - Control

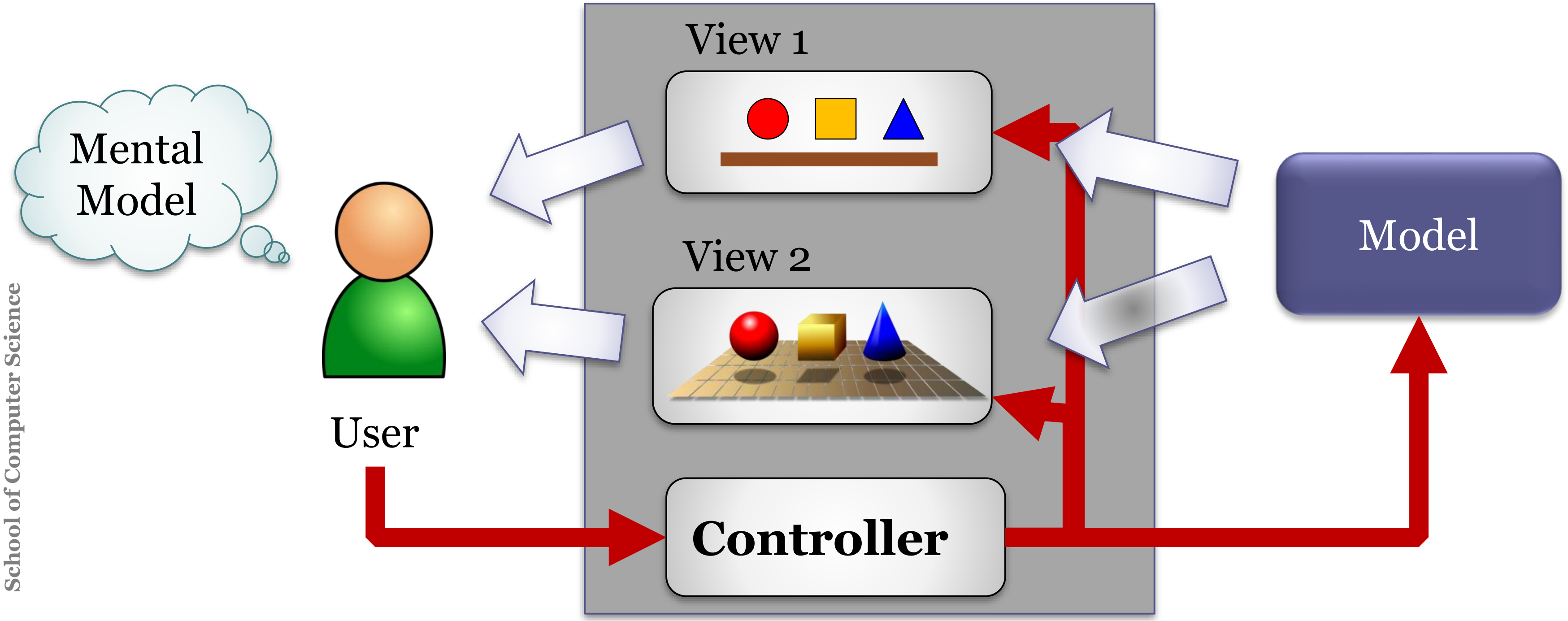
MVC

- MVC: Model - View - Controller
  - Proposed by Trygve Reenskaug (end of 70's)
  - Solution for GUI
  - Controller separates model from the view
  - *"Mental model"* offered through views

MVC

Elements

- Model: represents business logic and state
- View: Offers state representation to the user
- Controller: Coordinates interaction, views and model



MVC

Constraints

- Controller processes user events
  - Creates/removes views
  - Handles interaction
- Views only show values
- Models are independent of controllers/views

MVC

Advantages

- Supports multiple views of the same model
- Views synchronization
- Separation of concerns
  - Interaction (controller), state (model)
- It is easy to create new views and controllers
- Easy to modify *look & feel*
- Creation of generic frameworks

Challenges

- Increases complexity of GUI development
- Coupling between controllers and views
- Controllers/Views should depend on a model interface
- Some difficulties for GUI tools

MVC

Applications

- Lots of web frameworks follow MVC
  - Ruby on Rails, Spring MVC, Play, etc.
- Some variants
  - Push: controllers send orders to views
    - RoR
  - Pull: controllers receive orders from views
    - Play

MVC variants

PAC

Model-View-Presenter

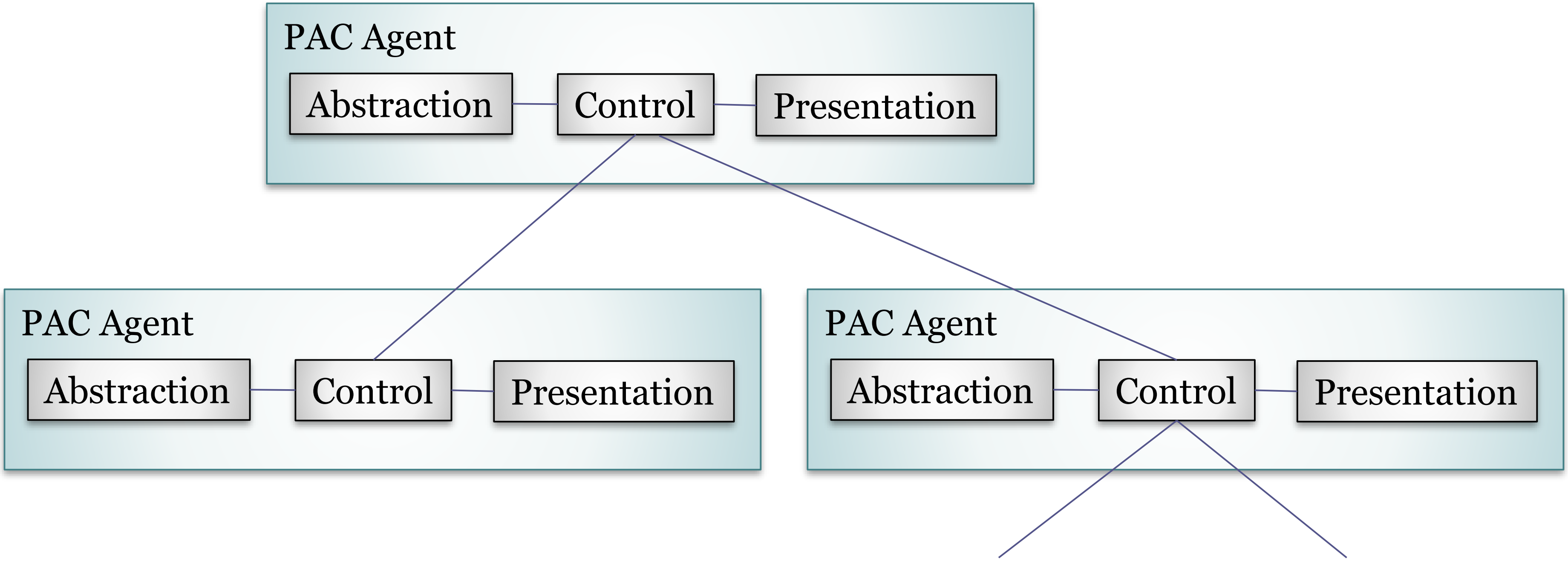
Model View ViewModel

...

PAC

PAC: Presentation-Abstraction-Control

- Hierarchy of agents
- Each agent contains 3 components



PAC

- Elements
  - Agents with
    - Presentation: visualization aspects
    - Abstraction: data model of an agent
    - Control: connects presentation and abstraction components and enables communication between agents
  - Hierarchical relationship between agents
- Constraints
  - Each agent is in charge of some functionality
  - There is no direct communication between abstraction and presentation in each agent
  - Communication through the control component

PAC

Advantages

- Separation of concerns
  - Identifies functionalities
- Support for changes and extensions
  - It is possible to modify an agent without affecting others
- Multitask
  - Agents can reside in different threads, processes or machines

Challenges

- Complexity of the system
  - Too many agents can generate a complex structure which can be difficult tom maintain
- Complexity of control components
  - Control components handle communication
  - Quality of control components is important for the whole quality of the system
- Performance
  - Communication overload between agents

PAC

- Applications
  - Network monitoring systems
  - Mobile robots
  - Drupal is based on PAC
- Relationships
  - This patterns is related with MVC
    - MVC has no agent hierarchy
  - This pattern was re-discovered as Hierarchical MVC

Repository

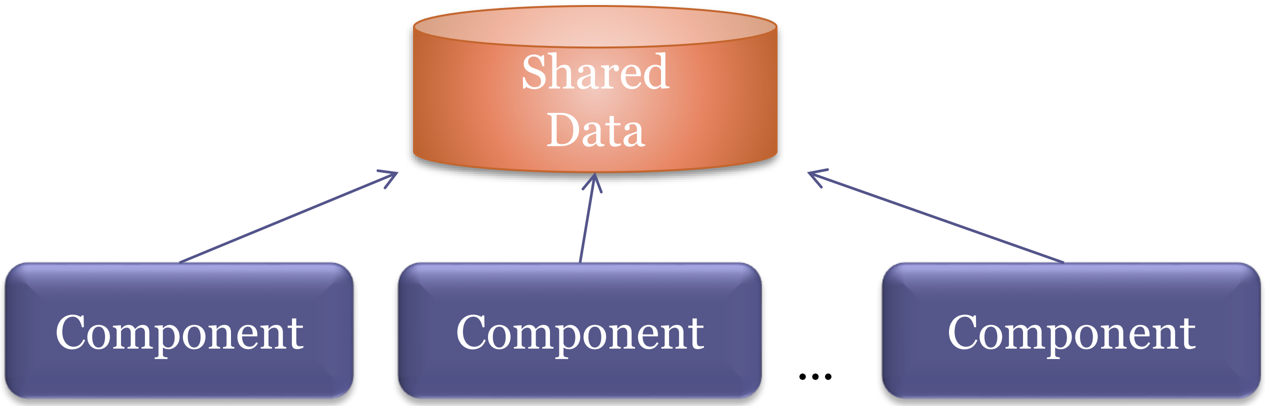
Shared data

Blackboard

Rule based

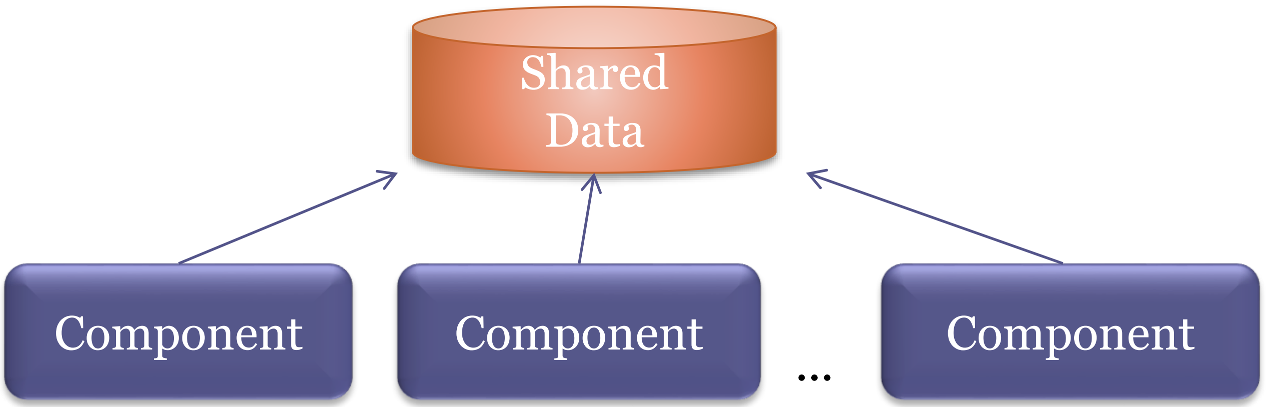
Shared data

- Independent components access the same state
  - Applications based on centralized data repositories



Shared data

- Elements
  - Shared data
    - Database or centralized repository
  - Components
    - Processors that interact with shared data



Shared data

Constraints

- Components interact with the global state
- Components don't communicate between each other
  - Only through shared state
- Shared repository handles data stability and consistency

Shared data

Advantages

- Independent components
  - They don't need to be aware of the existence of other components
- Data consistency
  - Centralized global state
  - Unique *Backup* of all the system state

Challenges

- Unique point of failure
  - A failure in the central repository can affect the whole system
  - Distributing the central data can be difficult
- Possible bottleneck
  - Inefficient communication
  - Problems for scalability
- Synchronization to access shared memory

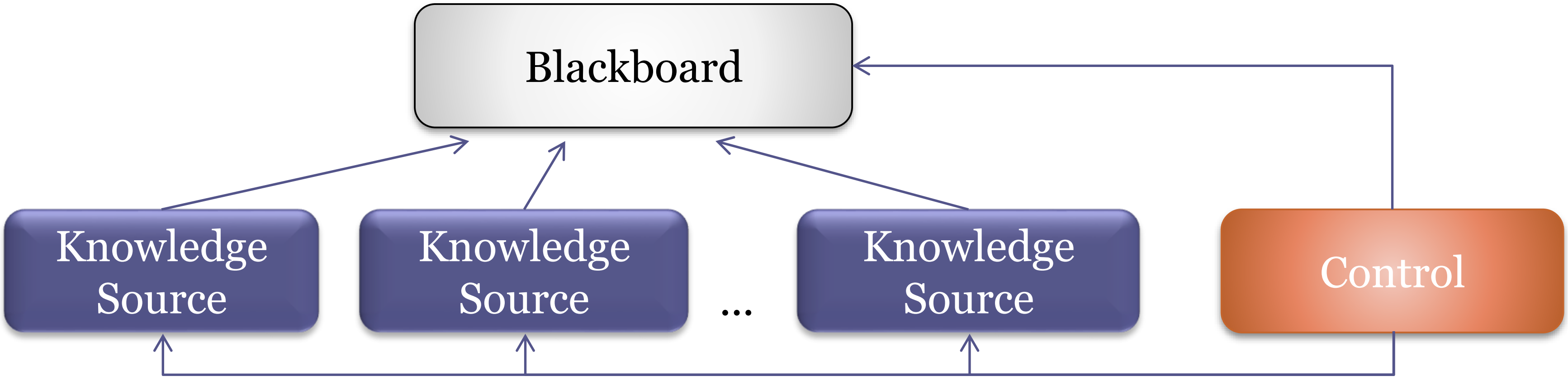
Shared data

- Applications
  - Lots of systems use this approach
- Some variants
  - This style is also known as:
    - Shared Memory, Repository, Shared data, etc.
  - Blackboard
  - Rule based systems

Blackboard

Complex problems which are difficult to solve

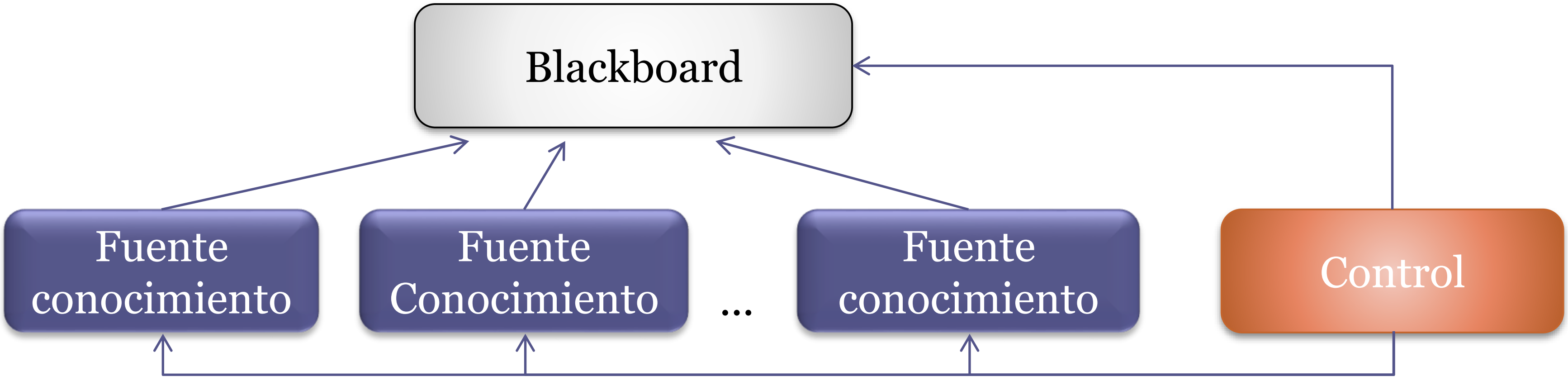
- Knowledge sources solve parts of the problem
- Each knowledge source aggregates partial solutions to the *blackboard*



Blackboard

Elements

- *Blackboard*: Central data repository
- Knowledge source: solves part of the problem and aggregates partial results
- Control: Manages tasks and checks the work state



Blackboard

Constraints

- Problem can be divided in parts
- Each knowledge source solves a part of the problem
- *Blackboard* contains partial solutions that are improving

Blackboard

Advantages

- Can be used for open problems
  - Experimenting
  - Facilitates strategy changes
- Knowledge source can be reused
- Support for fault tolerance

Challenges

- It may be difficult to debug
- No warranty that the right solution will be found
- It may be difficult to establish a control strategy
- Low performance
  - Sometimes it may need to review the incorrect hypothesis
- High development cost
- Implementation of parallelism
  - It is necessary to synchronize blackboard access

Blackboard

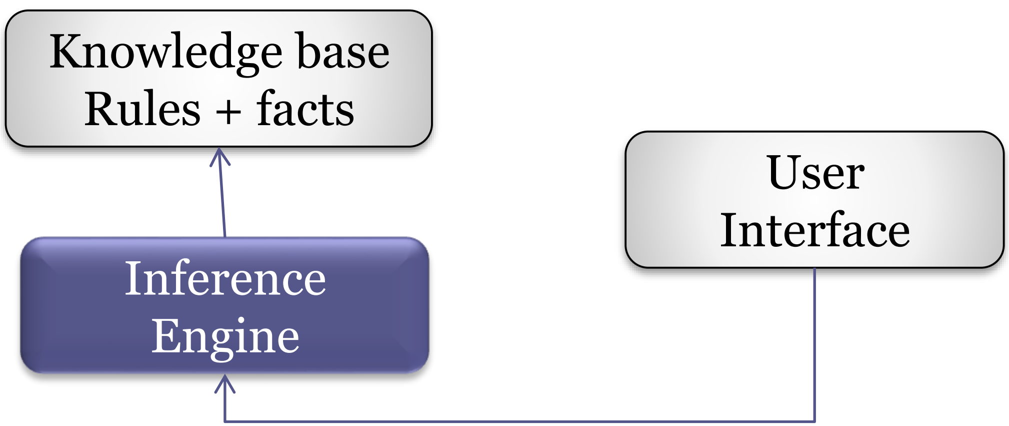
Applications

- Some speech recognition systems
  - HEARSAY-II
- Pattern recognition
- Weather forecasts
- Games
- Analysis of molecular structure
  - Crystalis

University of Oviedo

Rule based systems

- Variant of shared memory
  - Shared memory = Knowledge base
    - Contains rules and facts

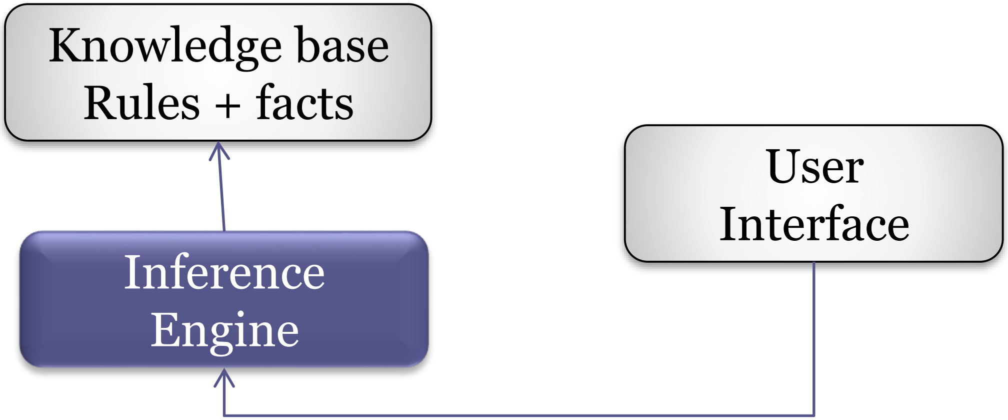


School of Computer Science

Rule based systems

Elements:

- Knowledge base: Rules and facts about some domain
- User interface: Queries/modifies knowledge base
- Inference engine: Answers queries from data and knowledge base



Rule based systems

Constraints:

- Domain knowledge captured in knowledge base
- Limit imperative access to knowledge base
  - It is based on rules like:
    - IF *antecedents* THEN *consequent*
  - Limits expressiveness with regards to imperative languages

Rule based systems

Advantages

- Declarative solution
  - It may be easy to modify the knowledge base
    - Specially tailored to be modified by domain experts
- Separation of concerns
  - Algorithm
  - Domain knowledge
- Reusability

Challenges

- Rules Debugging
- Performance
- Rules creation and maintenance
  - Introspection
  - Automatic rule learning
  - Runtime update of rules

Rule based systems

Applications

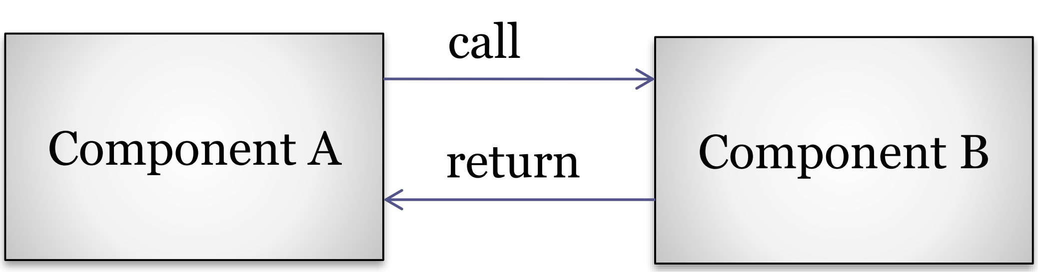
- Expert system
- Production systems
- Rules libraries in Java
  - JRules, Drools, JESS
- Declarative, rule based languages
  - Prolog (logic programming)
- BRMS (Business Rules Management Systems)

Invokation

- Call-return
- Client-Server
- Event based architectures
  - Publish-Subscribe
  - Actor models

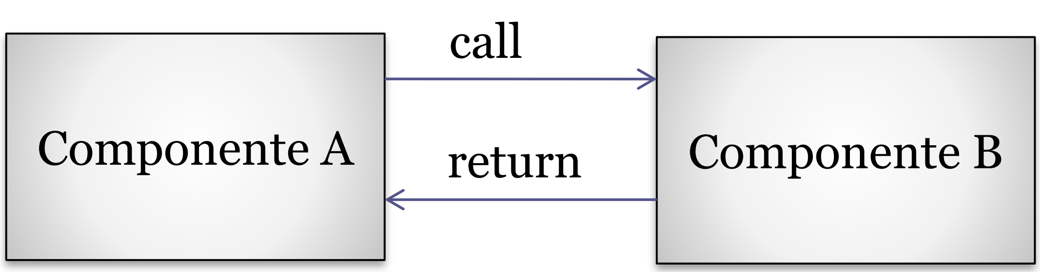
Call-return

A component calls another component and waits for the answer



Call-return

- Elements
  - Component that does the call
  - Component that sends the answer
- Constraints
  - Synchronous communication:
    - The caller waits for the answer

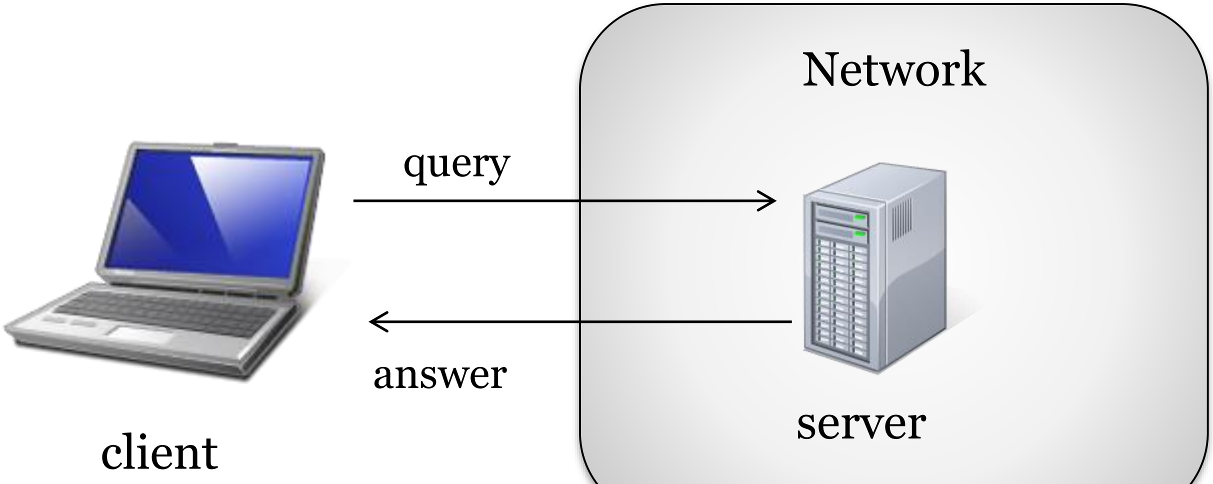


Call-return

- Advantages
  - Easy to implement
- Challenges
  - Problems for concurrent computation
    - If component is blocked waiting for the answer
      - It can be using unneeded resources
  - Distributed environments
    - Little utilization of computational capabilities

Client-Server

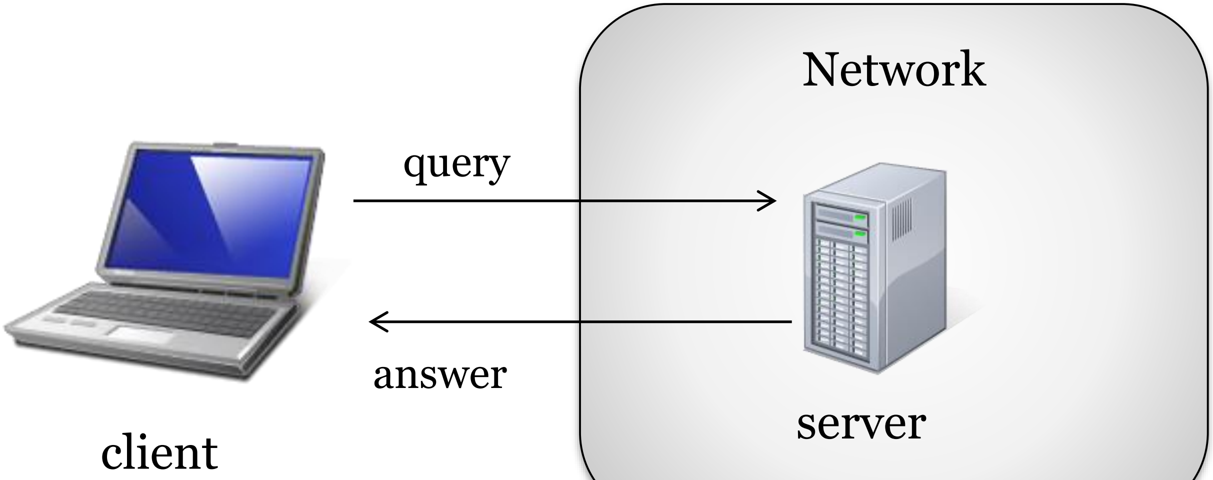
- Variant of layers
  - 2 layers physically separated (2-tier)
- Functionality is divided in several servers
- Clients connect to services
  - Interface query/answer



Client-Server

Elements

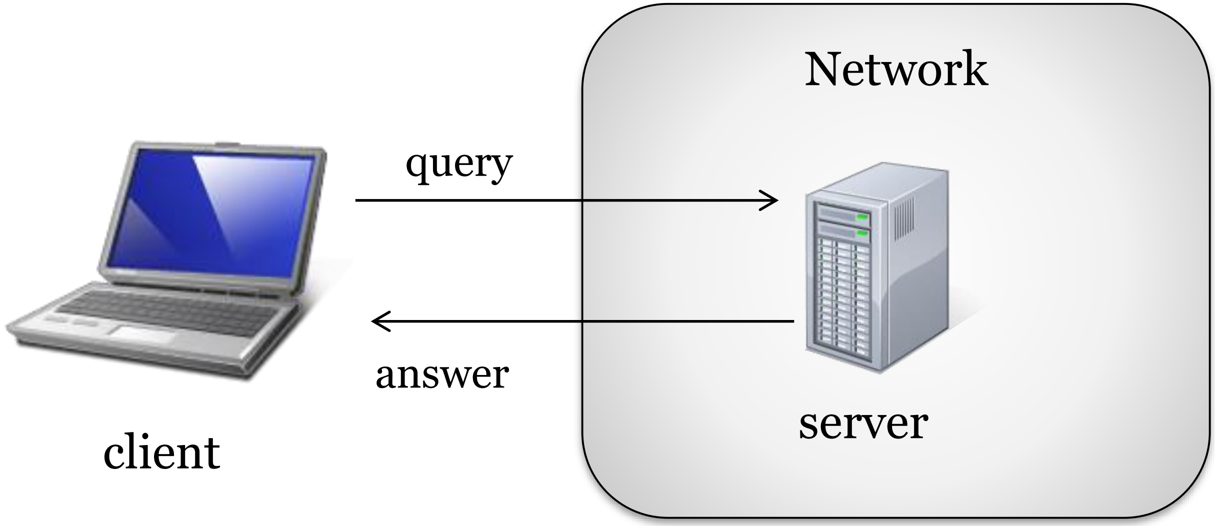
- Server: offers services through a query/answer protocol
- Client: does queries and process answers
- Network protocol: communication management between clients and servers



Client-Server

Constraints

- Clients communicate with servers
  - Not the other way
- Clients are independent from other clients
- Servers don't have knowledge about clients
- Network protocol establishes some communication warranties



Client-Server

Advantages

- Servers can be distributed
- Separation of functionality between clients/servers
  - Independent development
  - Scalability
- General functionality available for all clients
  - Although not all the servers need to offer all the functionality

Challenges

- Each server can be a single point of failure
  - Server attacks
- Unpredictable performance
  - Dependency on the system and the network
  - Problems when servers belong to other organizations
    - How can quality of service be warranted?

Client-Server

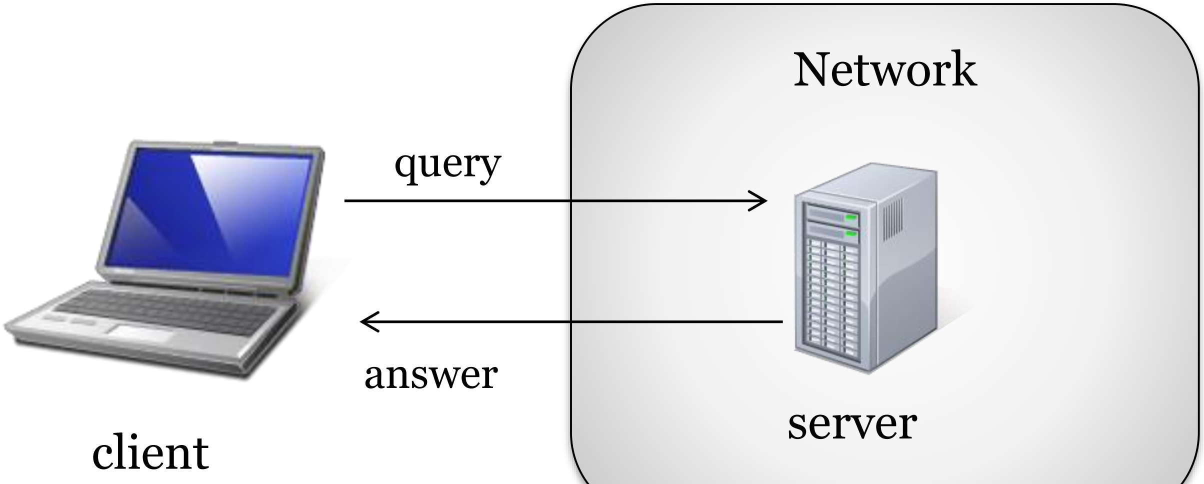
Variants

- Stateless
- Replicated server
- With cache

Client-Server stateless

Constraint

- Server does not store information about clients
- Same query implies same answer



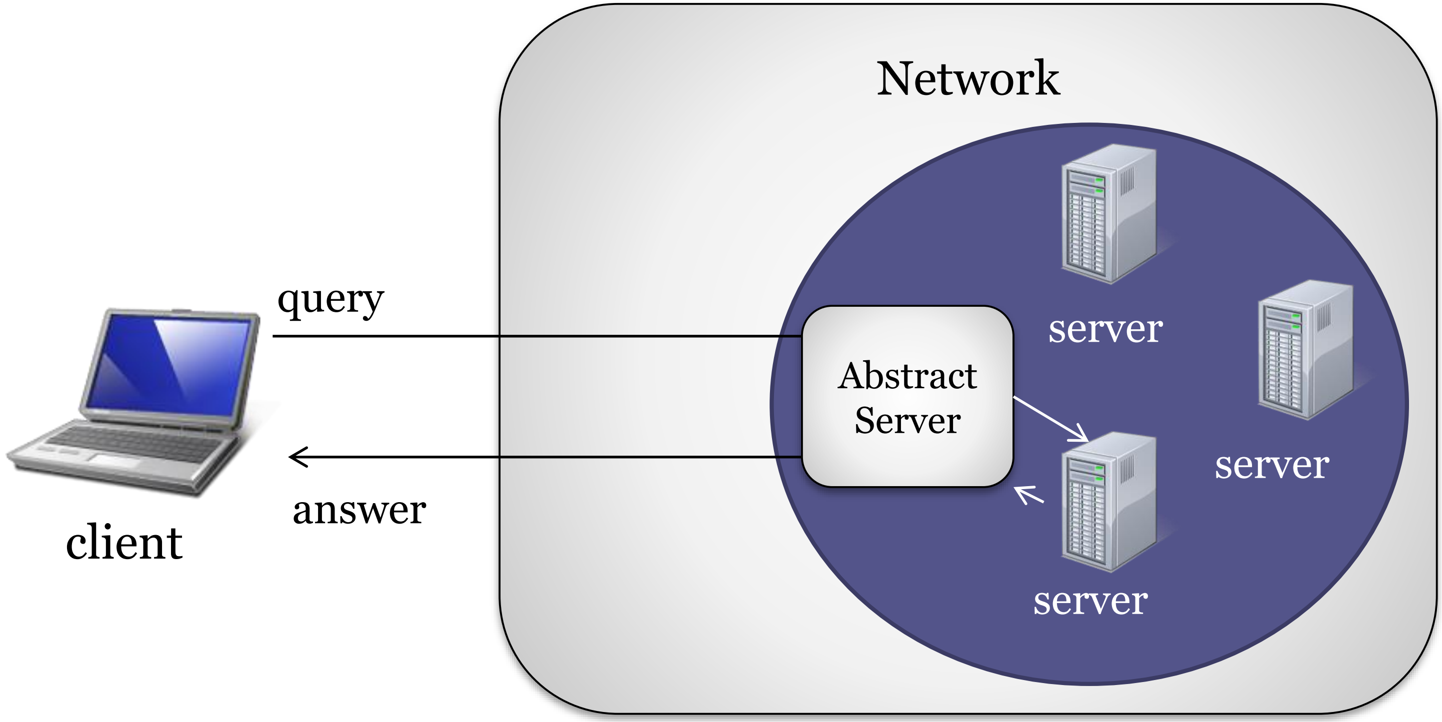
Client-Server stateless

- Advantages
  - Scalability
- Challenges
  - Application state management
    - Client must remember requests
    - Some strategies can be established to handle information between requests

Replicated server

Constraint

- Several servers offer the same service
  - Offer the client the appearance that there is only one server

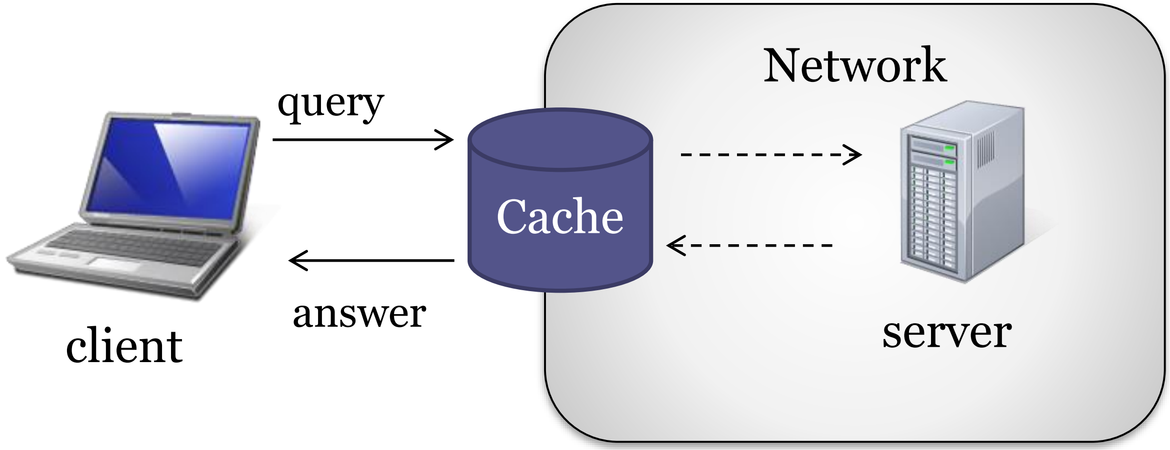


Replicated server

- Advantages
  - Better answer times
  - Less latency
  - Fault tolerance
- Challenges
  - Consistency management between replicated servers
  - Synchronization

Client-server with cache

- Cache = mediator between client/server
  - Stores copies of previous answers to the server
    - When a query is received it return the cached answer without asking the original server



Client-server with cache

Elements:

- Intermediate cache nodes

Constraints

- Some queries are directly answered by the cache node
- Cache node has a policy for answer management
  - Expiration time

Client-server with cache

Advantages:

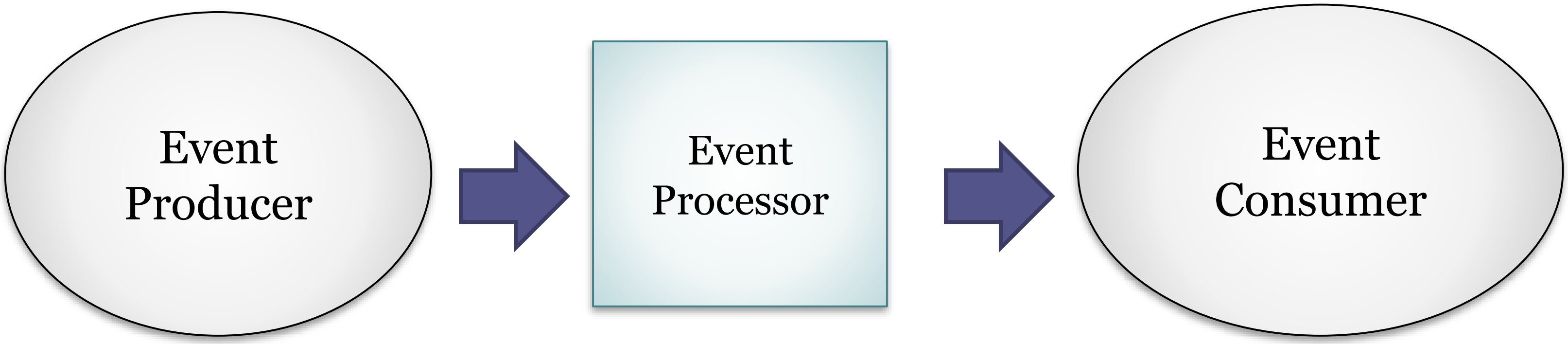
- Less network overload
  - Lots of repeated requests can be stored in the cache
- Less answer time
  - Cached answers arrive earlier

Challenges

- Complexity of configuration
- Expiration policy
- Not appropriate for certain domains
  - When high fidelity of answers is needed
  - Example: real time systems

Event based

EDA (Event-Driven-Architecture)



Event based

Elements:

- Event:
  - Something that has happened (≠ request)
- Event producer
  - Event generator (sensors, systems, ...)
- Event consumer
  - DB, applications, scorecards, ...
- Event processor
  - Transmission channel
  - Filters and transforms events

Event based

Constraints:

- Asynchronous communication
  - Producers generate events at any moment
  - Consumers can be notified of events at any moment
- Relationship one-to-many
  - An event can be sent to several consumers

Event based

Advantages

- Decoupling
  - Producer does not depend on consumer, nor vice versa.
- Timelessness
  - Events are published without any need to wait for the termination of any cycle
- Asynchronous
  - In order to publish an event there is no need to finish any process

Challenges

- Non sequential execution
  - Possible lack of control
- Difficult to debug

Event based

Applications

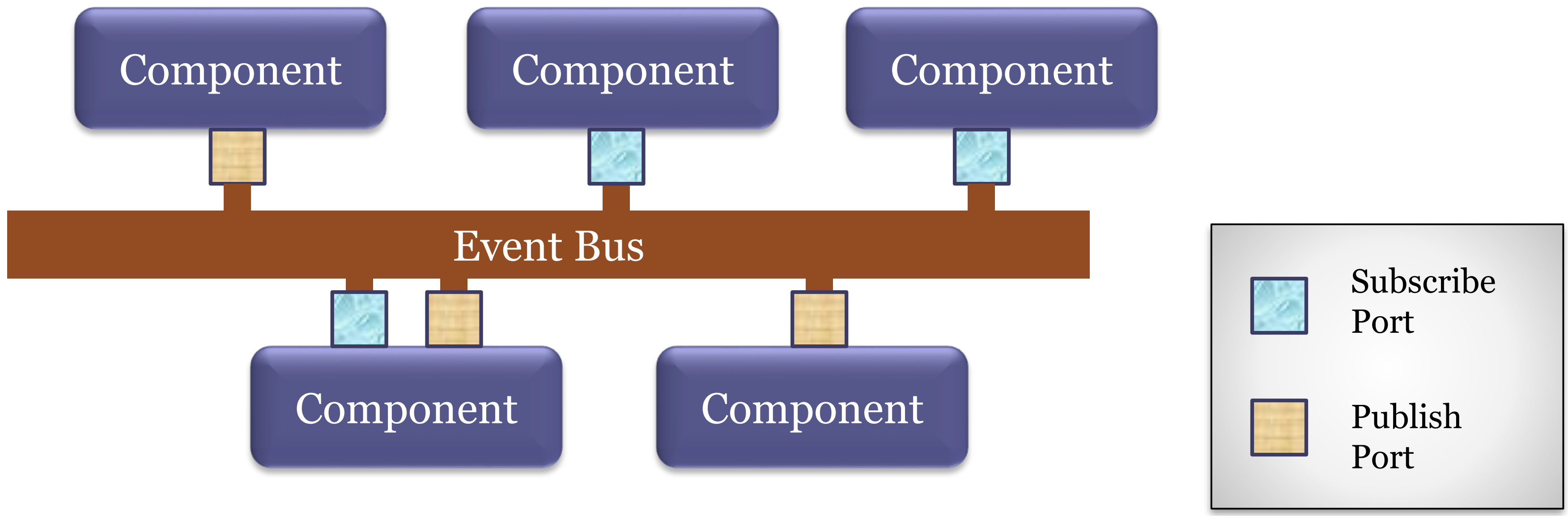
- Event processing networks
- *Event-Stream-Processing (ESP)*
- *Complex-event-processing*

Variants

- Publish-subscribe
- Actor models

Publish-subscribe

Components subscribe to a channel to receive messages from other components



Publish-subscribe

Elements:

- Component:
  - Component that subscribes to a channel
- Publication port
  - It is registered to publish messages
- Subscription port
  - It is registered to receive some kind of messages
- Event bus (message channel):
  - Transmits messages to subscribers

Publish-subscribe

Constraints:

- Separation between subscription/publication port
  - A component may have both ports
- Non-direct communication
  - Asynchronous communication in general
  - Components delegate communication responsibility to the channel

Publish-subscribe

Advantages

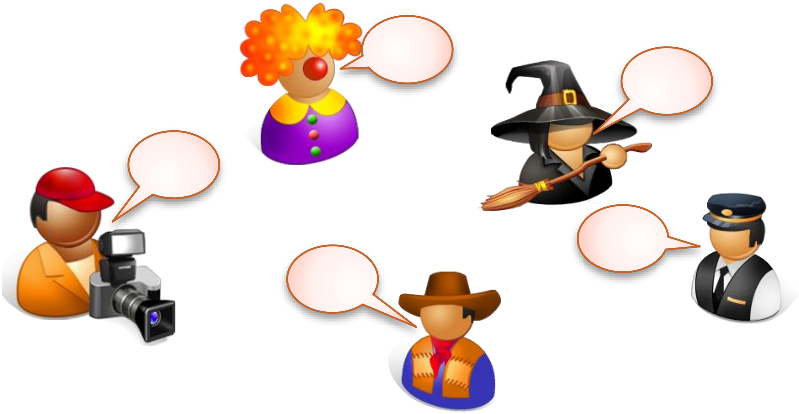
- Communication quality
  - Improves performance
  - Debugging
- Low coupling between components
  - Consumers do not depend on publishers
  - ...nor vice versa...

Challenges

- It adds a new indirection level
  - Direct communication may be more efficient in some domains
- Complex implementation
  - It may require COTS

Actor models

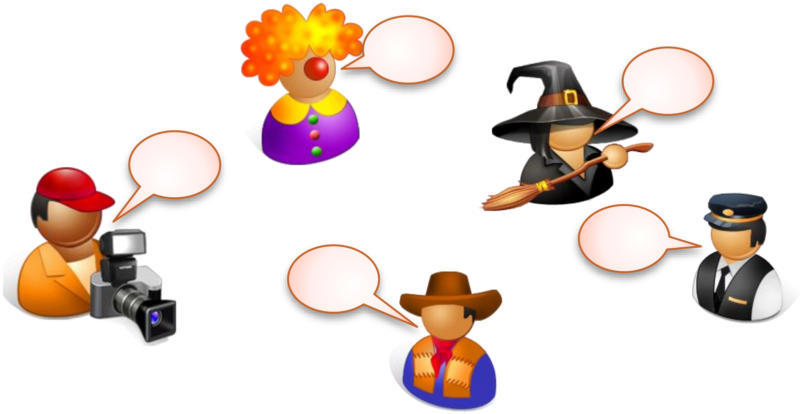
- Used for concurrent computation
  - Actors instead of objects
  - There is no shared state between actors
  - Asynchronous message passing
- Theoretical developments since 1973 (Carl Hewitt)



Actor models

Elements

- Actor: computational entity with state
  - It communicates with other actors sending messages
  - It process messages one by one
- Messages
- Addresses: Identify actors (*mailing address*)



Actor models

Constraints

- An actor can only:
  - Send messages to other actors
    - Messages are immutable
  - Create new actors
  - Modify how it will process next message
- Actors are decoupled
  - Receiver does not depend on sender

Actor models

Constraints (2)

- Local addresses
  - An actor can only send messages to known addresses
  - Because they were given to it or because he created them
- Parallelism:
  - All actions are in parallel
  - No shared global state
  - Messages can arrive in any order

Actor models

Advantages

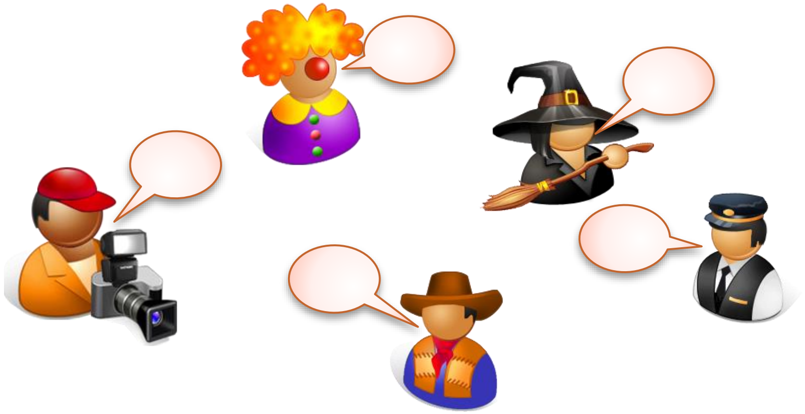
- Highly parallel
- Transparency and scalability
  - Internal vs external addresses
- Non-local actor models
  - Web Services
  - Multi-agent systems

Challenges

- Message sending
  - How to handle arriving messages
- Actor Coordination
- Non-consistent systems by definition

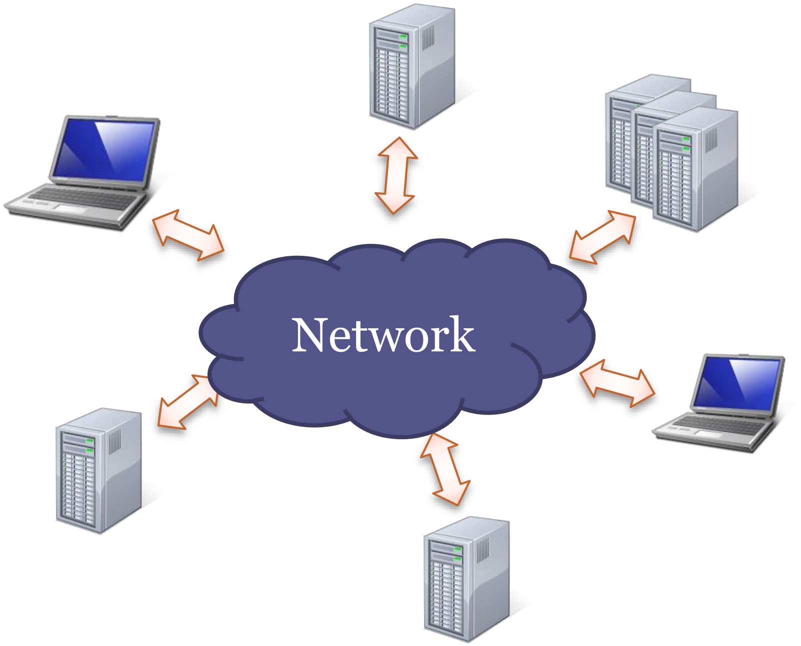
Actor models

- Implementations
  - Erlang (programming language)
  - Akka (library)
- Applications
  - Reactive systems
  - Examples: Ericsson, Facebook, twitter



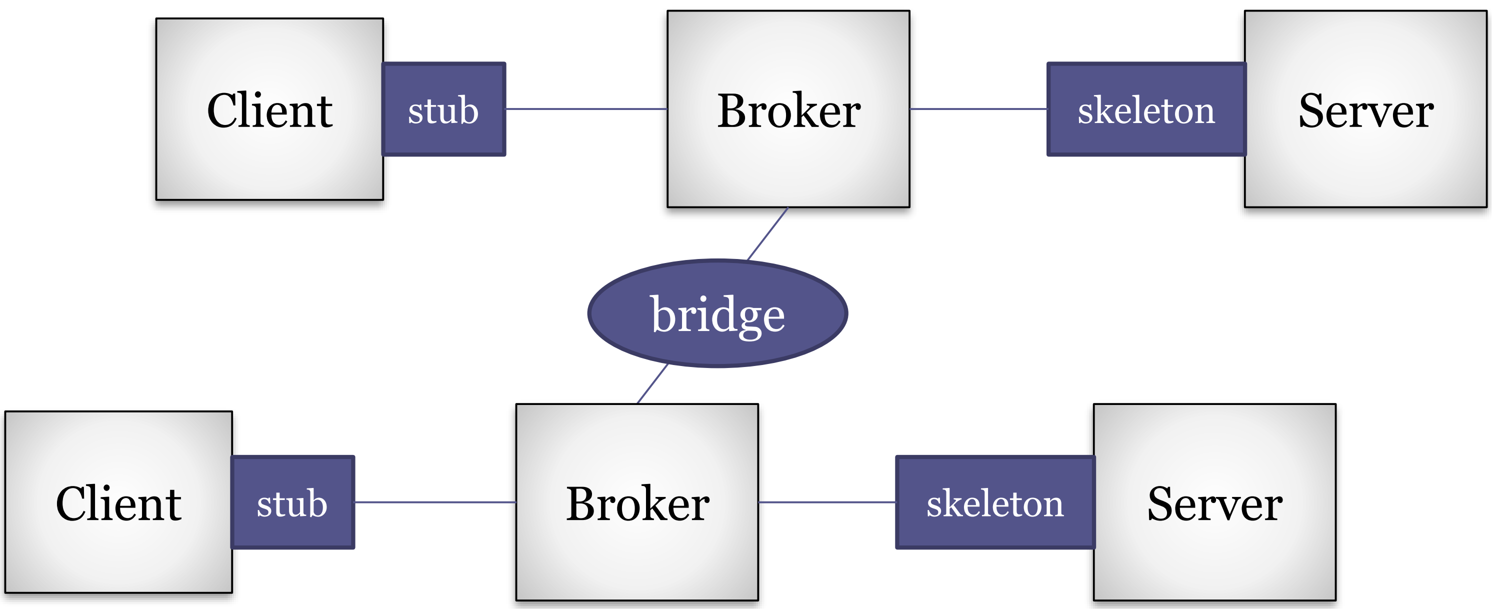
Distributed and network systems

Broker
Peer-to-peer
MapReduce
Lambda arquictecture
Kappa architecture



Broker

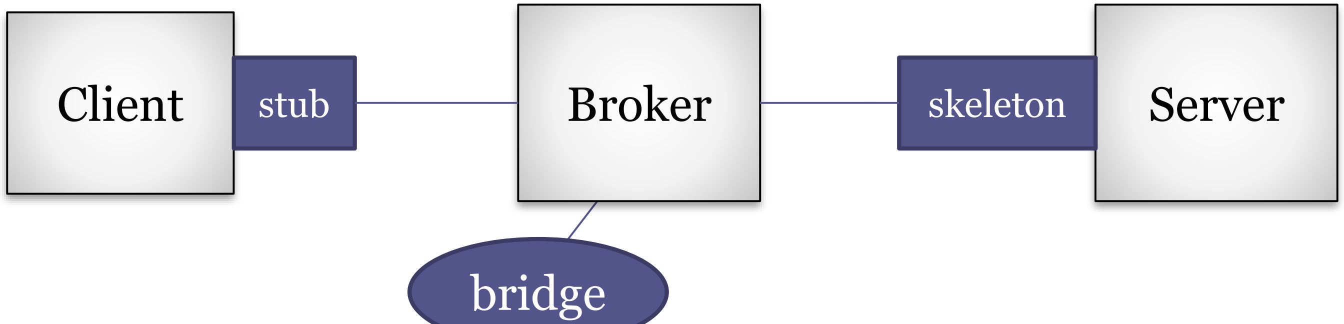
Intermediate node that manages communication between a client and a server



Broker

Elements

- Broker
  - Manages communication
- Client: Sends requests
  - Client Proxy: *stub*
- Server: Returns answers
  - Server Proxy: *skeleton*
- Bridge: Can connect brokers



Broker

Advantages

- Separation of concerns
  - Delegates low level communication aspects to the broker
  - Separate maintenance
  - Reusability
- Servers are independent from clients
- Portability
  - Broker = low level aspects
- Interoperability
  - Using *bridges*

Challenges

- Performance
  - Adds an indirection layer
- Can increase coupling between components
- Broker = single point of failure

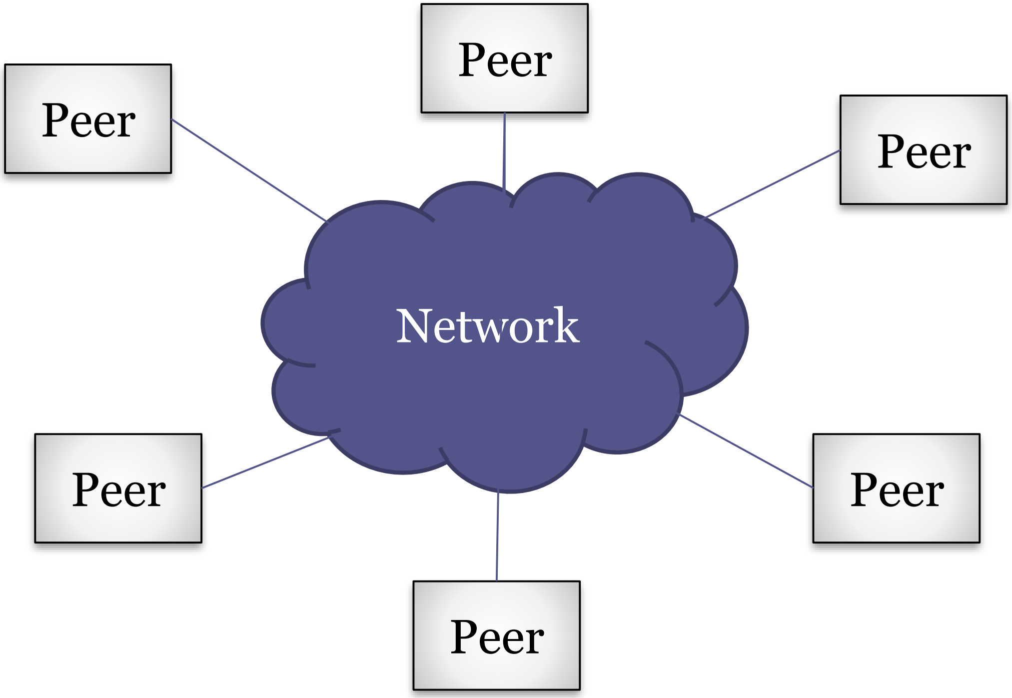
Broker

Applications

- CORBA and distributed systems
- Android uses a variation of Broker pattern

Peer-to-Peer

Equal and autonomous nodes (*peers*) that communicate between them.



Peer-to-Peer

- Elements
  - Computational nodes: *peers*
    - They contain their own state and control thread
  - Network protocol
- Constraints
  - There is no main node
  - All peers are equal

Peer-to-Peer

Advantages

- Decentralized information and control
- Fault tolerance
  - There is no single point of failure
  - A failure in one peer does not compromise the whole system

Challenges

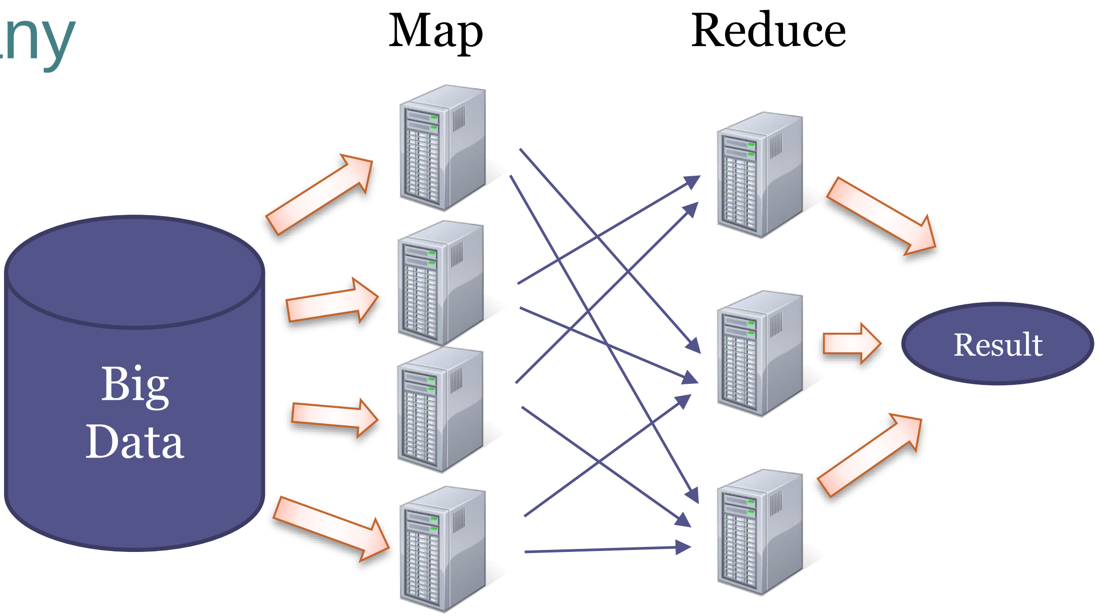
- Keeping the state of the system
  - Complexity of the protocol
- Bandwidth Limitations
  - Network and protocol latency
- Security
  - Detect malicious *peers*

Peer-to-Peer

- Popular applications
  - Napster, BitTorrent, Gnutella, ...
  - This architecture style is not only to share files
    - e-Commerce (B2B)
    - Collaborative systems
    - Sensor networks
    - ...
- Variants
  - Super-peers

MapReduce

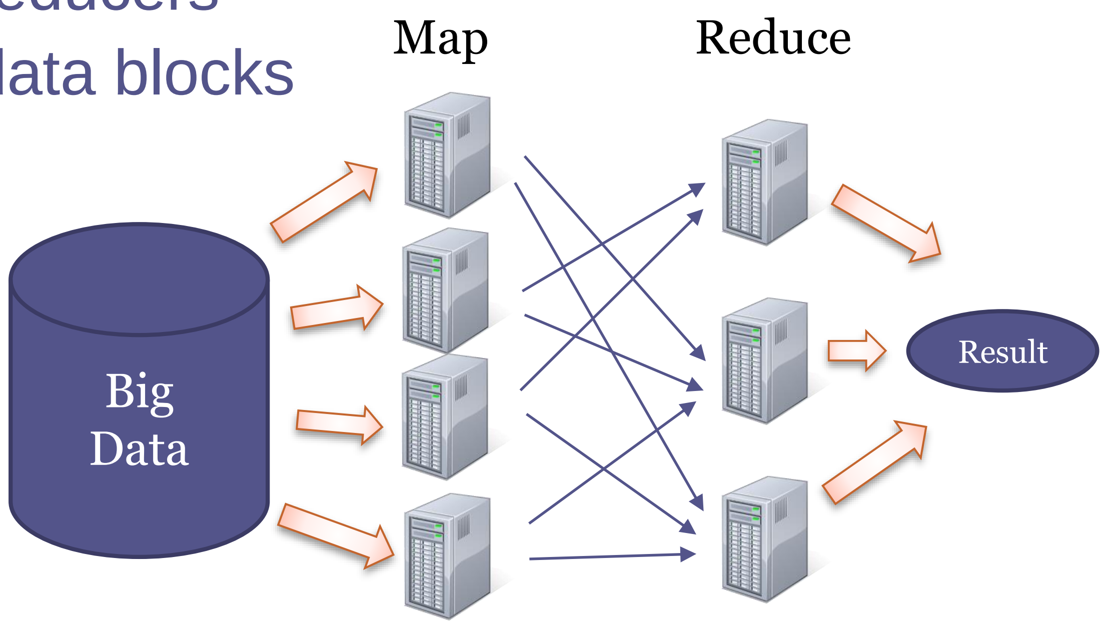
- Proposed by Google
  - Published in 2004
  - Internal implementation by Google
- Goal: big amounts of data
  - Lots of computational nodes
  - Fault tolerance
  - Write-once, read-many
- Style composed of:
  - Master-slave
  - Batch



MapReduce

Elements

- *Master* node: Controls execution
    - Node table
    - It manages replicated file system
- *Slave nodes*
    - Execute mappers, reducers
    - Contain replicated data blocks



MapReduce - Scheme

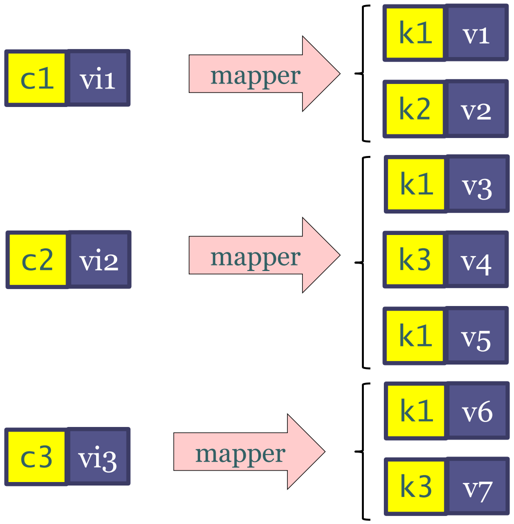
- Inspired by functional programming
  - 2 components: mapper and reducer
- Data are divided for their processing
- Each data is associated with a key
  - Transforms `[(key1,value1)]` to `[(key2,value2)]`

Input:
`[(key1,value1)]`

Output:
`[(key2,value2)]`

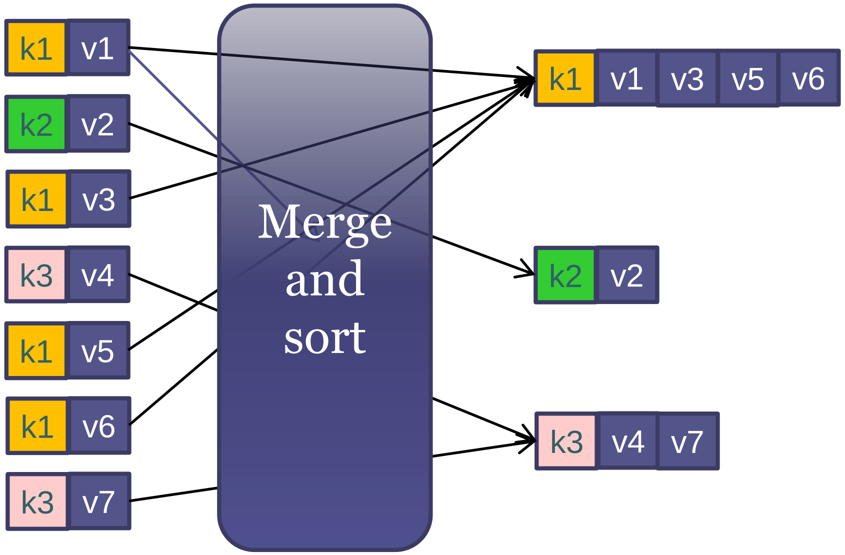
MapReduce
Type of mapper

mapper: `(Key1, Value1)` → `[(Key2,Value2)]`



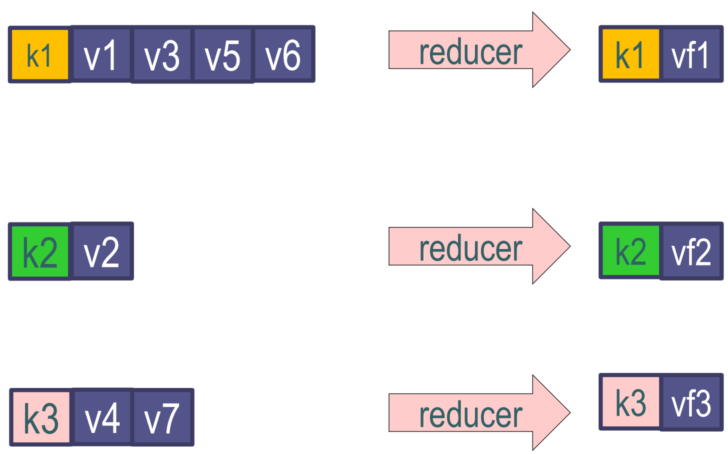
MapReduce - Merge and sort

System merges and sorts intermediate results according to the keys

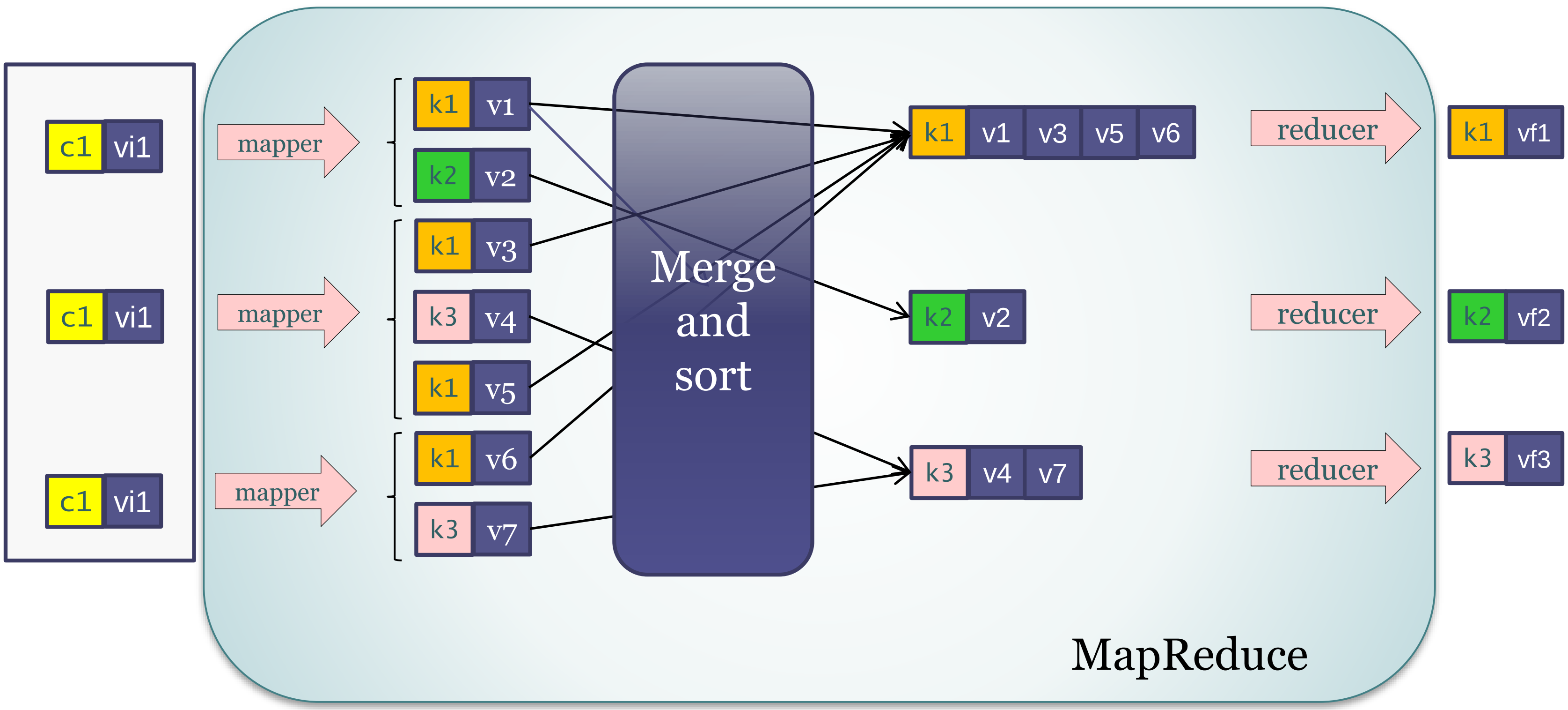


MapReduce
Type of reducers

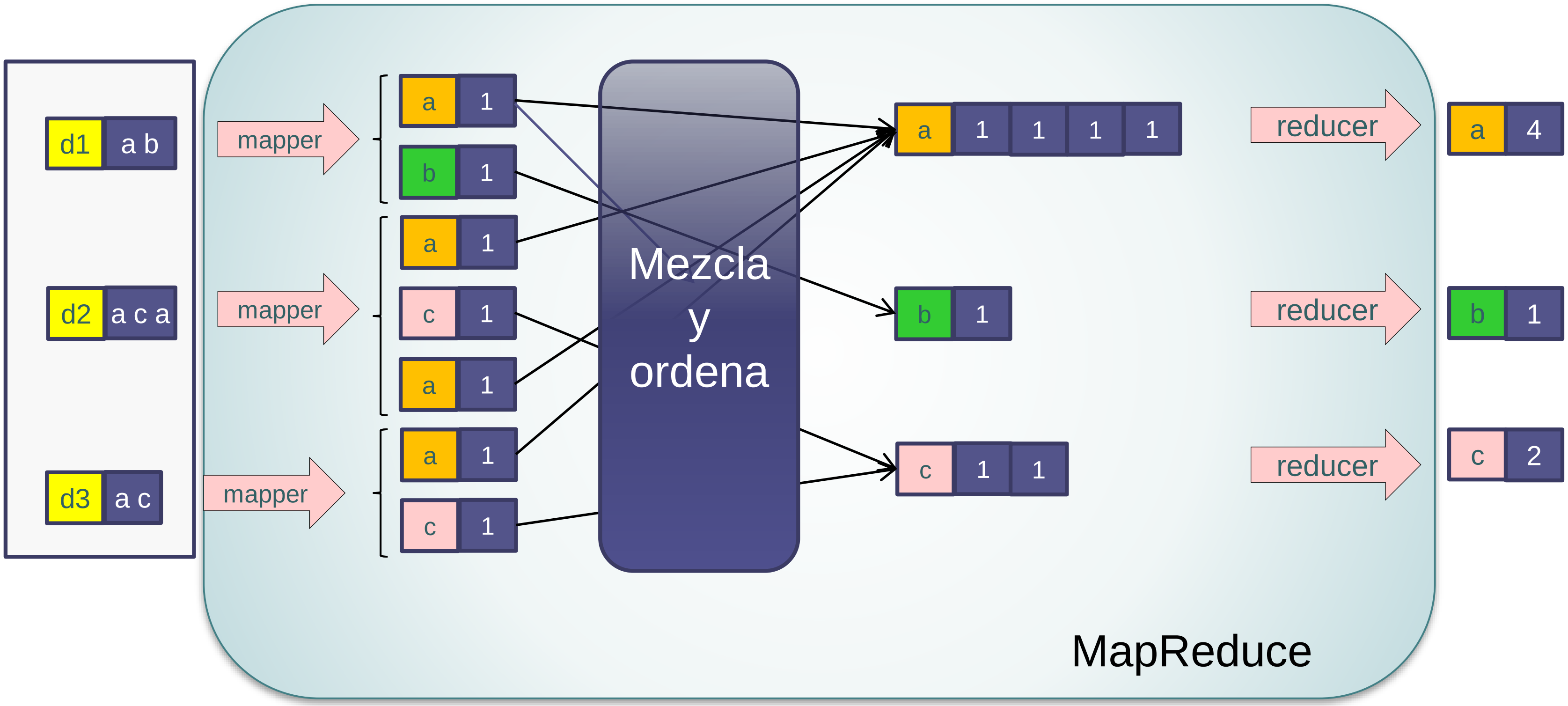
reducer: `(Key2, [Value2])` → `(Key2,Value2)`



MapReduce - general scheme



MapReduce - count words




```
// return each work with 1
mapper(d,ps) {
 for each p in ps:
   emit (p, 1)
}
```

```
// sum the list of numbers of each word
reducer(p,ns) {
  sum = 0
  for each n in ns { sum += n; }
  emit (p, sum)
}
```

MapReduce - execution environment

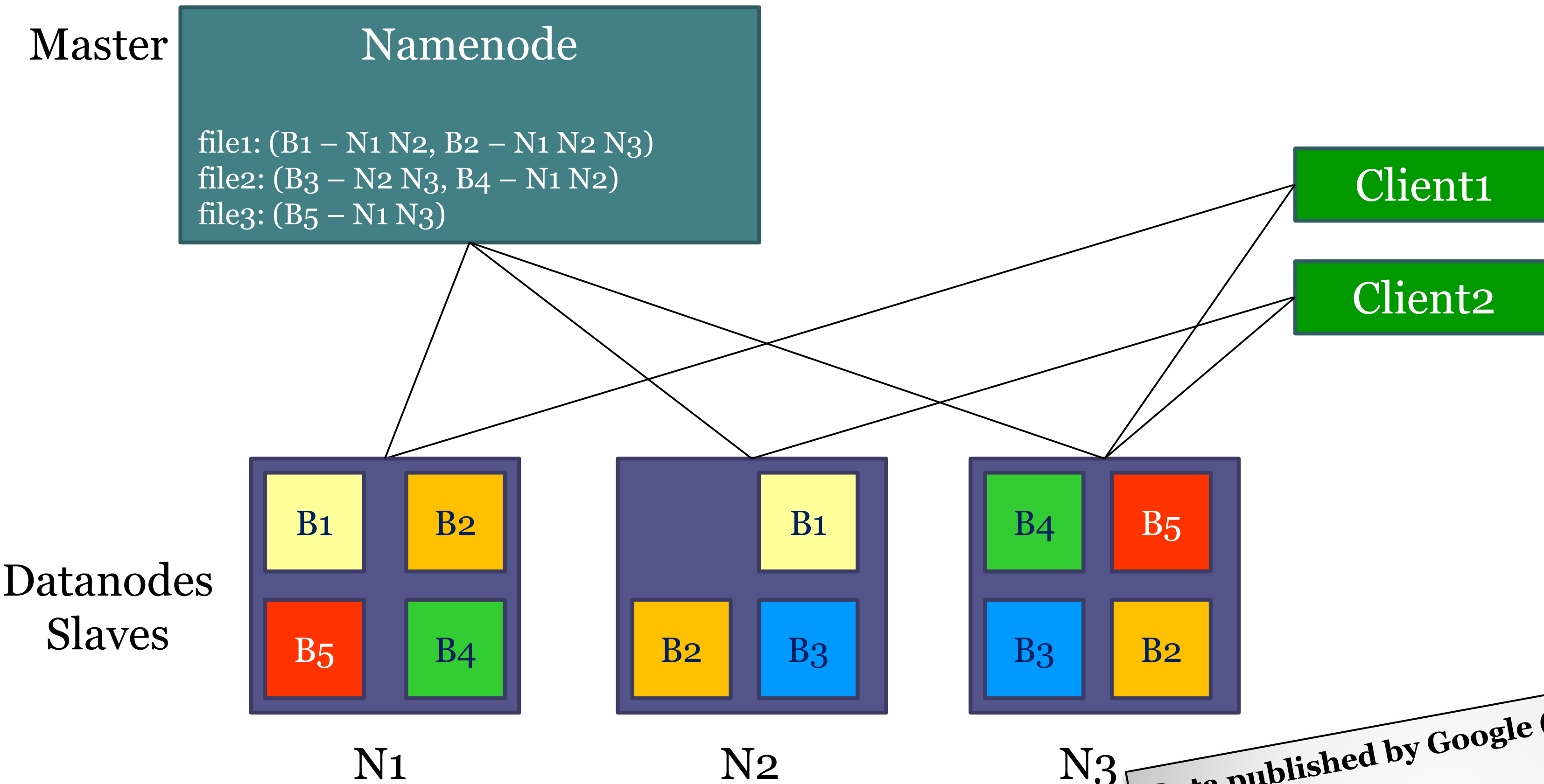
Execution environment is in charge of:

- Planning: Each job is divided in tasks
- Placement of data/code
  - Each node contains its data locally
- Synchronization:
  - *reduce* tasks must wait *map* phase
- Error and failure handling
  - High tolerance to computational nodes failures

MapReduce - File system

- Google developed a distributed file system - GFS
- Hadoop created HDFS
  - Files are divided in chunks
  - 2 node types:
    - Namenode (master), datanodes (data servers)
  - Datanodes store different chunks
    - Block replication
  - *Namenode* contains metadata
    - Where is each chunk
    - Direct communication between clients and datanodes

MapReduce - File system



Data published by Google (2007)
200+ clusters
Lots of clusters 1000+ machines
Pools with thousands of clients
4+ PB
HW/SW fault tolerance

MapReduce

Advantages

- Distributed computations
  - Split input data
    - Replicated repository
  - Fault tolerant
    - Hardware/software heterogeneous
- Large amount of data
  - Write-once. Read-many

Challenges

- Dependency on master node
- Non interactivity
- Data conversion to MapReduce
  - Adapt input data
  - Convert output data

MapReduce: Applications

Lots of applications:

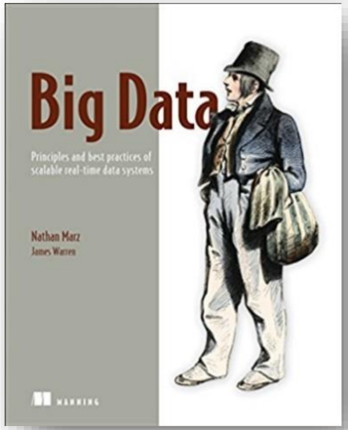
- Google, 2007, 20petabytes/day, around 100,000 mapreduce jobs/day
- PageRank algorithm can be implemented as MapReduce
- Success stories:
  - Automatic translation, similarity, sorting, ...
- Other companies: last.fm, facebook, Yahoo!, twitter, etc.

MapReduce: Applications

- Implementations
  - Google (internal)
  - Hadoop (*open source)*
  - …
- Libraries
  - Hive (Hadoop): query language inspired by SQL
  - Pig (Hadoop): specific language that can define data flows
  - Cascading: API that can specify distributed data flows
  - Flume Java (Google)
  - Dryad (Microsoft)

Lambda architecture

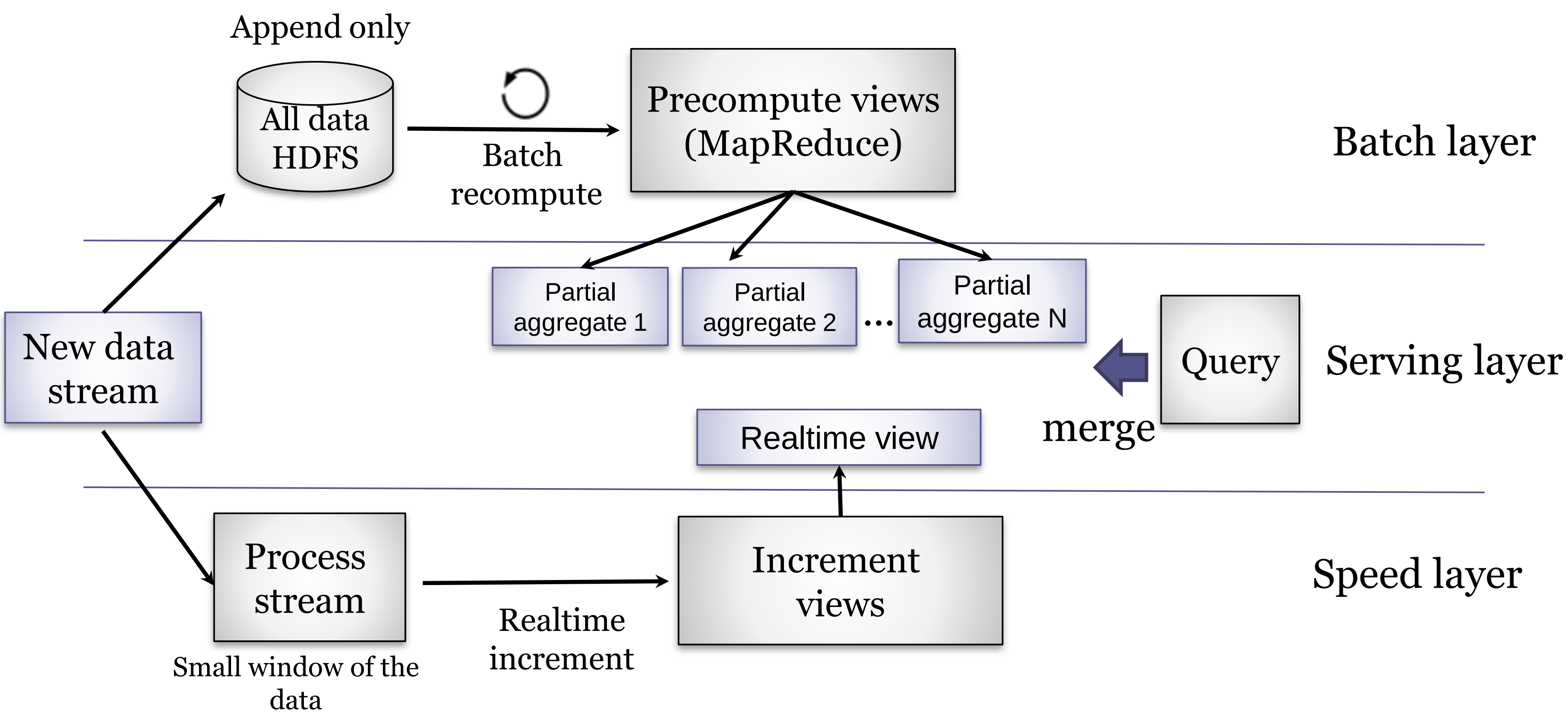
- Handle Big Data & real time analytics
- Proposed by Nathan Marz, 2011
- 3 layers
  - Batch layer: precomputes all data with MapReduce
    - Generates partial aggregate views
    - Recomputes from all data
  - Speed layer: real time, small window of data
    - Generates fast real time views
  - Serving layer: handles queries
    - Merges the different views



Lambda architecture

λ

Combines Real time with batch processing



Lambda architecture

λ

Constraints

- All data is stored in the batch layer
- The batch layer precomputes views
- The results of the speed layer may not be accurate
- Serving layer combines precomputed views
- The views can be simple DBs for querying

Lambda architecture

λ

Advantages

- Scalability (Big data)
- Real time
- Decoupling
- Fault tolerant
- Keep all input data
- Reprocessing

Challenges

- Inherent complexity
- Merging views can be innacurate
  - Losing some events

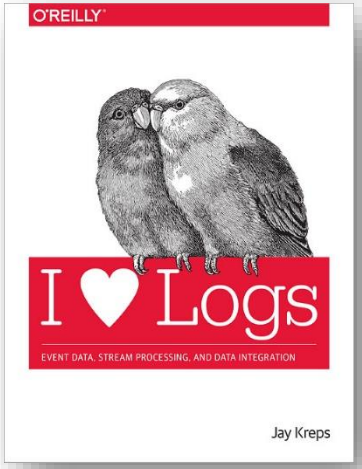
Lambda architecture

λ

- Applications
  - Spotify, Alibaba, …
- Libraries
  - Apache Storm
  - Netflix Suro project

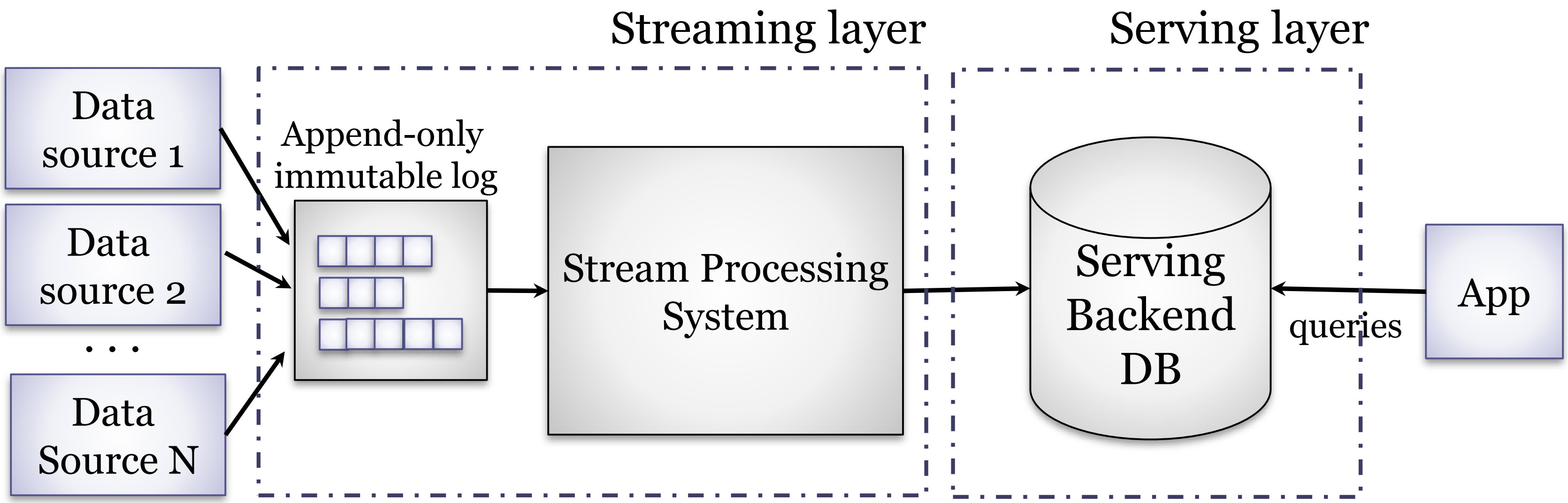
Kappa architecture

- Proposed by Jay Krepps (Apache Kafka), 2013
- Handle Big data & Real time with logs
  - Simplifies Lambda architecture
    - Removes the batch layer
  - Based on a distributed ordered log
    - Replicated cluster
    - The log can be very large



Kappa architecture

Diagram



Kappa architecture

Constraints

- The event log is append-only
- The events in the log are immutable
- Stream processing can request events at any position
  - To handle failures or doing recomputations

University of Oviedo

Kappa architecture

Advantages

- Scalable (big data)
- Real time processing
- Simpler than lambda
  - No batch layer

Challenges

- Space requirements
  - Duplication of log and DB
  - Log compaction
- Ordering of events
- Delivery paradigms
  - At least once
  - At most once (it may be lost)
  - Exactly once

School of Computer Science

Kappa architecture

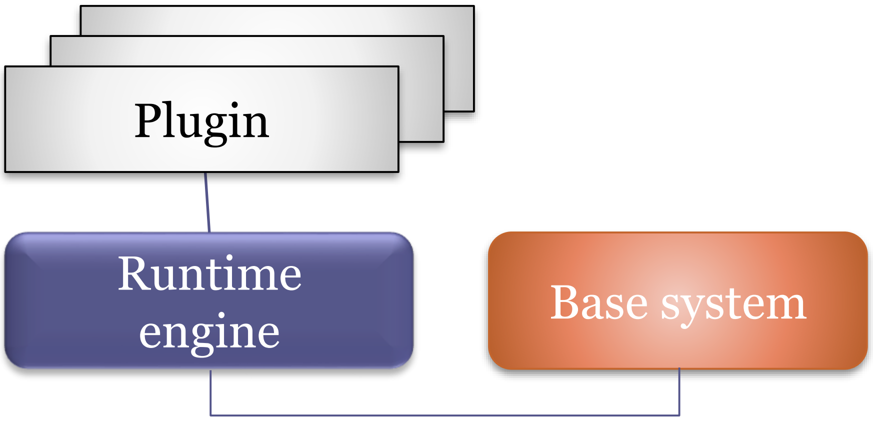
- Applications & libraries
  - Apache Kafka
  - Apache Samza
  - Spark Streaming
- LinkedIn

Adaptable Systems

Plugins
Microkernel
Reflection
Interpreters and DSL
Mobile code
- Code on demand
- Remote evaluation
- Mobile agents

Plugins

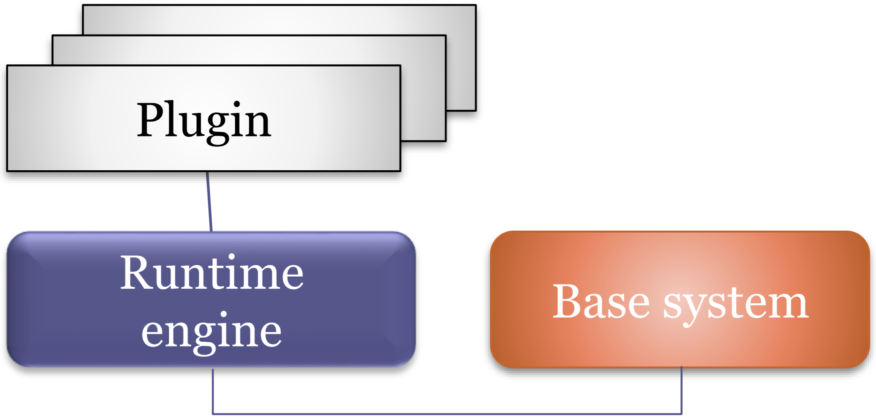
It allows to extend the system using plugins that add new functionality



Plugins

Elements

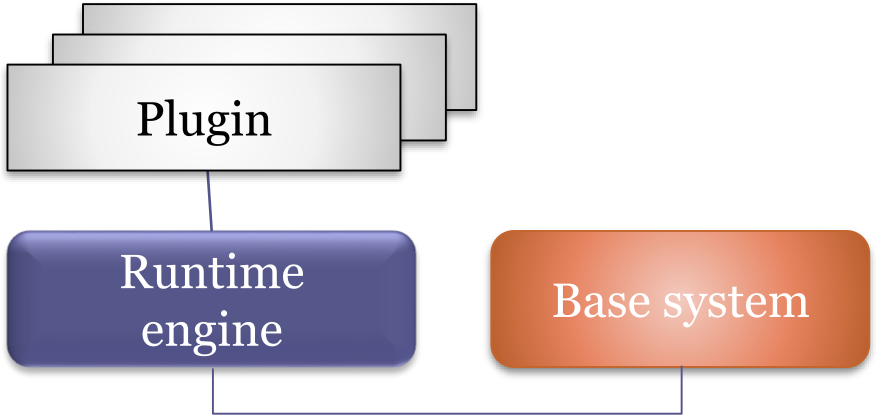
- Base system:
  - System that allows plugins
- *Plugins:* Components that can be added/removed dynamically
- Runtime engine:
  - Starts, localizes, initializes, executes, and stops plugins



Plugins

Constraints

- Runtime engine manages plugins
- System can add/remove plugins
- Some plugins can depend on other plugins
  - The plugin must declare dependencies and the exported API



Plugins

Advantages

- Extensibility
  - Application can get new functionalities in some ways that were not foreseen by the original developers
- Customization
  - Application can have a small kernel that is extended on demand

Challenges

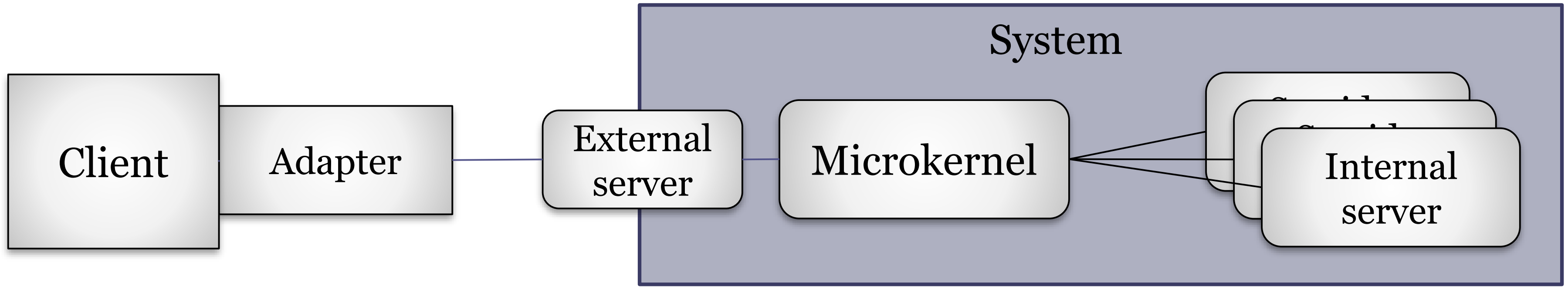
- Consistency
  - Plugins must be added to the system in a sound way
- Performance
  - Delay searching/configuring plugins
- Security
  - Plugins made by third parties can compromise security
- Plugin management and dependencies

Plugins

- Examples
  - Eclipse
  - Firefox
- Technologies
  - Component systems: OSGi

Microkernel

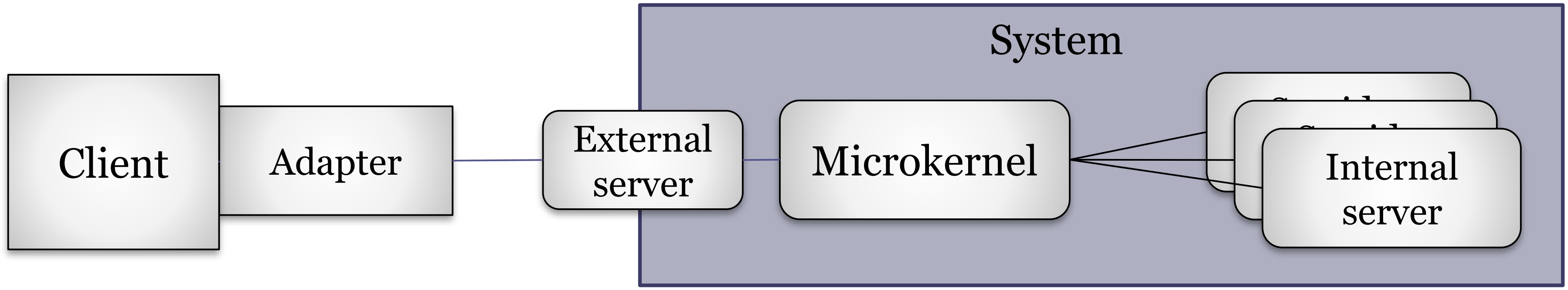
- Identify minimal functionality in a microkernel
- Extra functionality is added using internal servers
- External server handles communication with other systems



Microkernel

Elements

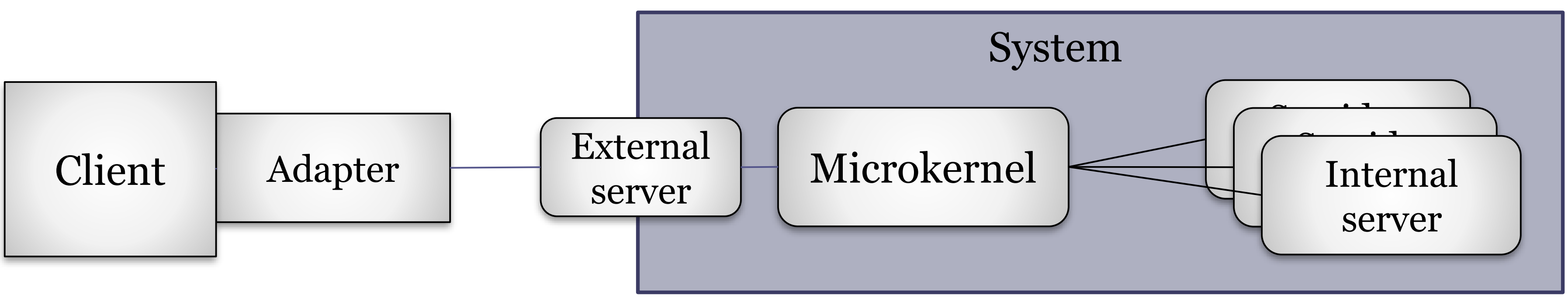
- Microkernel: Minimal functionality
- Internal server: Extra functionality
- External server: Offers external API
- Client: External application
  - Adapter: Component that establish communication with external server



Microkernel

Constraints:

- *Microkernel* implements only minimal functionality
- The rest of the functionality is implemented using internal servers
- Communication with clients by external servers



Microkernel

Advantages

- Portability
  - It is only needed to port the kernel
- Flexibility and extensibility
  - Adding new functionality with new internal servers
- Security and reliability
  - Critical parts of the system are encapsulated
  - Errors in external parts don't affect the microkernel

Challenges

- Performance
  - A monolithic can be more efficient
- Design complexity
  - Identify components in the microkernel
    - It may be difficult to separate parts to internal servers
- Unique point of failure
  - If microkernel fails, the whole system may fail

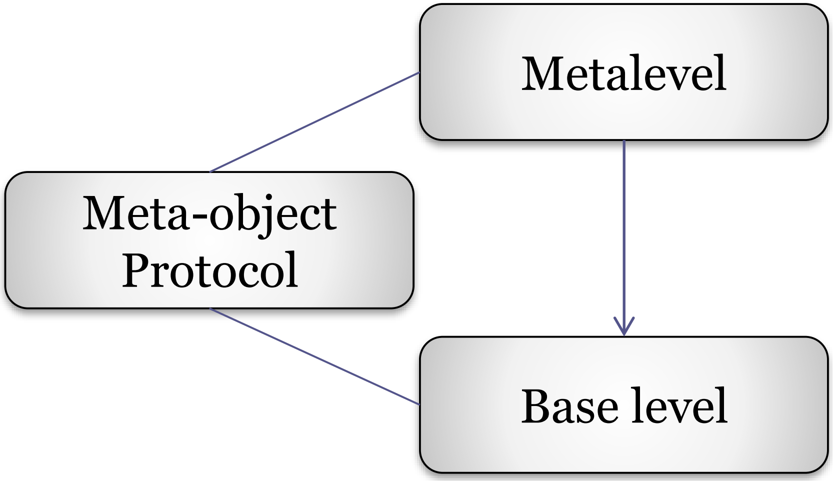
Microkernel

- Applications
  - Operating systems
  - Games
  - Editors

Reflection

It allows to change the structure and behavior of an application dynamically

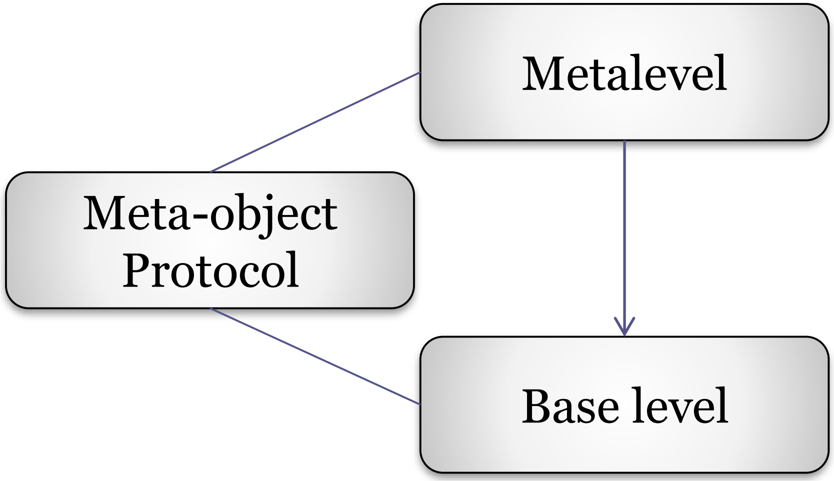
Systems that can modify themselves



Reflection

Elements

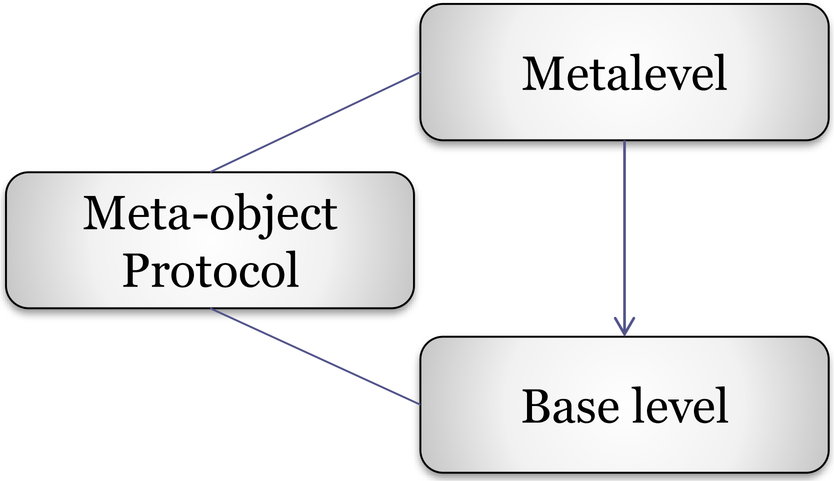
- Base level: Implements application logic
- Metalevel: Aspects that can be modified
- Metaobject protocol: Interface that can modify the metalevel



Reflection

Constraints

- Base level uses metalevel aspects for its behavior
- At runtime, it is possible to modify the metalevel using the metaobject protocol



Reflection

Advantages

- Flexibility
  - Adapt to changing conditions
  - Change behavior of running system without changing source code or stopping execution

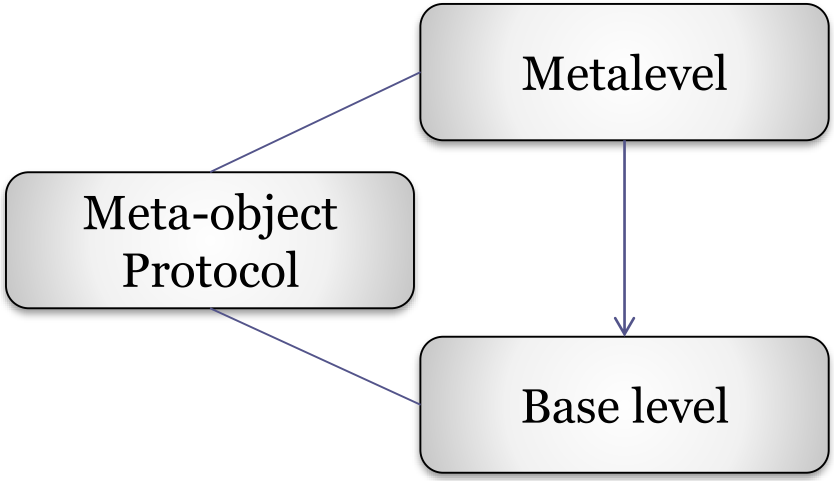
Challenges

- Implementation
  - Not all languages enable meta-programming
  - More difficult to combine with static type systems
- Performance
  - It may be necessary to do some optimizations to limit reflection
- Security:
  - Consistency maintenance

Reflection

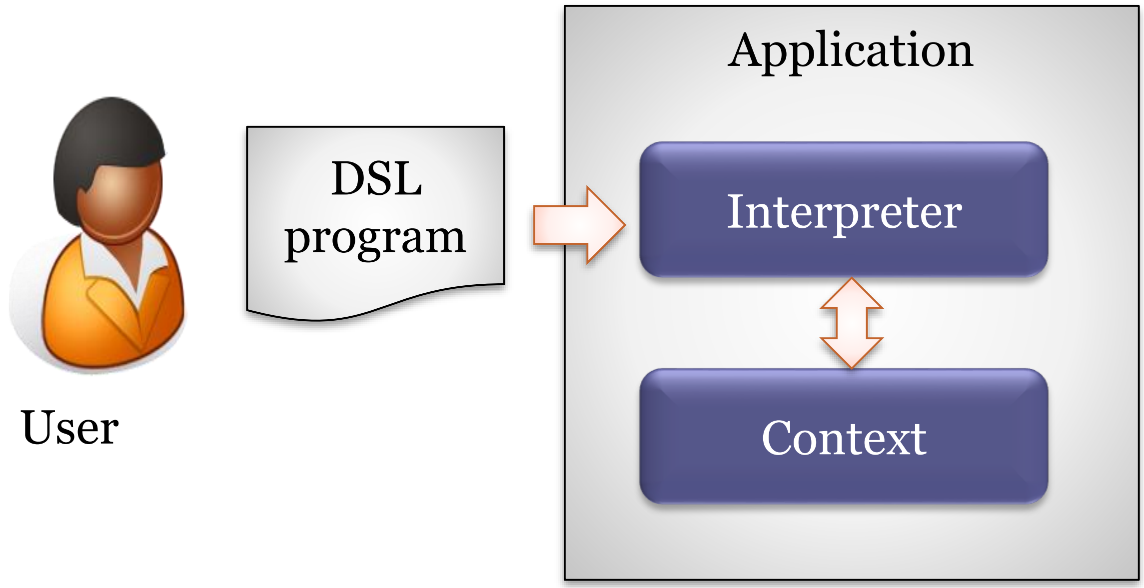
Applications

- Most dynamic languages support reflection
  - Scheme, CLOS, Ruby, Python, ...
- Intelligent systems
- Self-modifiable code



Interpreters and DSLs

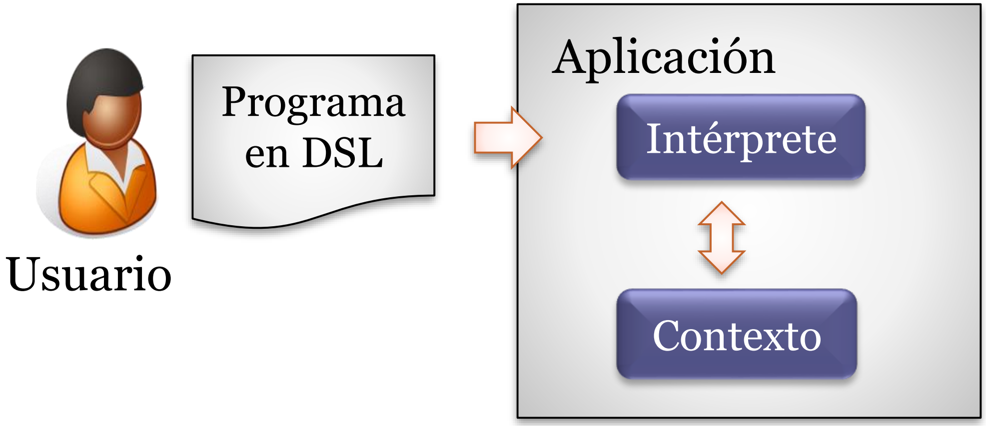
Include a domain specific language (DSL) that is interpreted by the system



Interpreters and DSLs

Elements

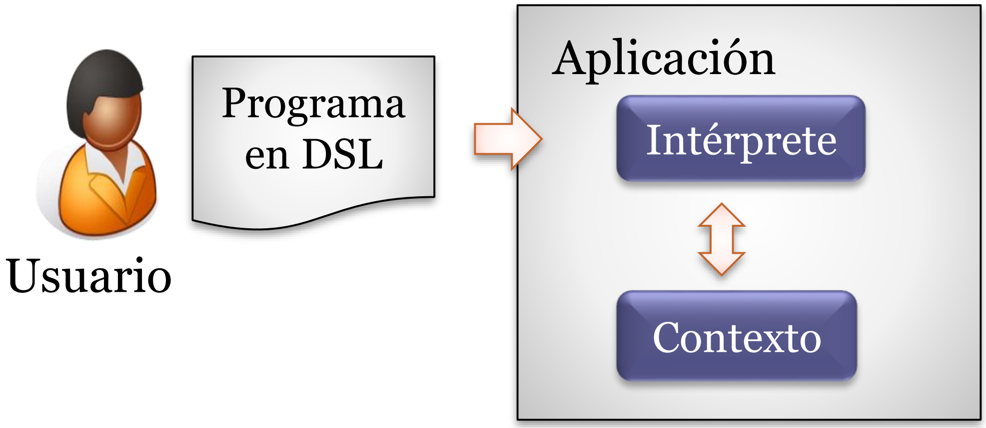
- Interpreter: Module that executes the program
- Program: Written in the DSL
  - DSL can be designed so the end user can write programs
- Context: Environment where the program is executed



Interpreters and DSLs

Constraints

- Interpreter runs the program interacting with the context
- It is necessary to define a DSL
  - Syntax (grammar, parsing,...)
  - Semantics (behavior)



Interpreters and DSLs

Advantages

- Flexibility
  - Adapt application behavior to user needs
- Usability
  - End users can write their own programs
- Adaptability
  - Easy to adapt to unforeseen situations

Challenges

- Design of the DSL
- Complexity of implementation
  - Interpreter
  - Separation of context/interpreter
- Performance
  - Possible programs may be not optimal
- Security
  - Handle wrong programs

Interpreters and DSLs

Variants:

- Embedded DSLs

Embedded DSLs

- Embedded DSLs
- Domain specific languages that are embedded in general purpose host languages
  - This technique is popular in soma languages like Haskell, Ruby, Scala, etc.

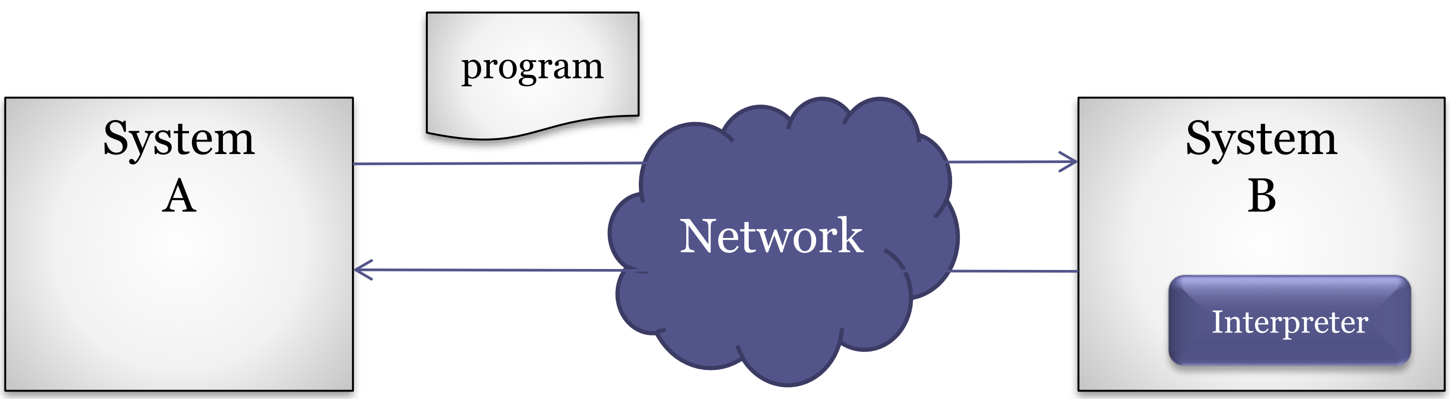
Embedded DSLs

- Advantages:
  - Reuse of host language syntax
  - Access to libraries and IDEs of host language
- Challenges
  - Separation between DSL and host language
  - End users may have too many expressivity

Mobile code

Code that is transferred from one machine to another

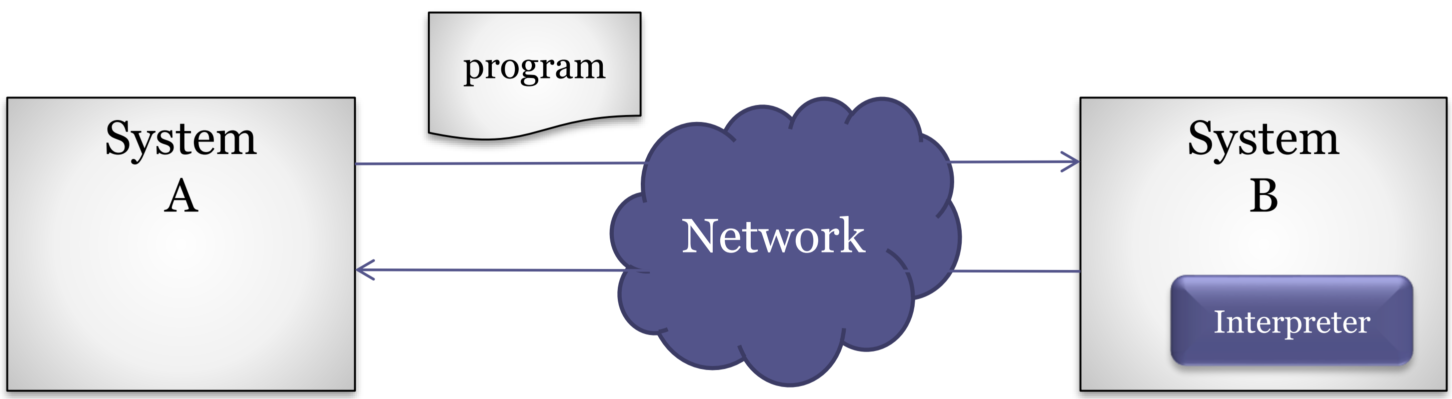
- System A sends a program to be run by system B
- System B must contain an interpreter for the language in which the program is written



Mobile code

Elements

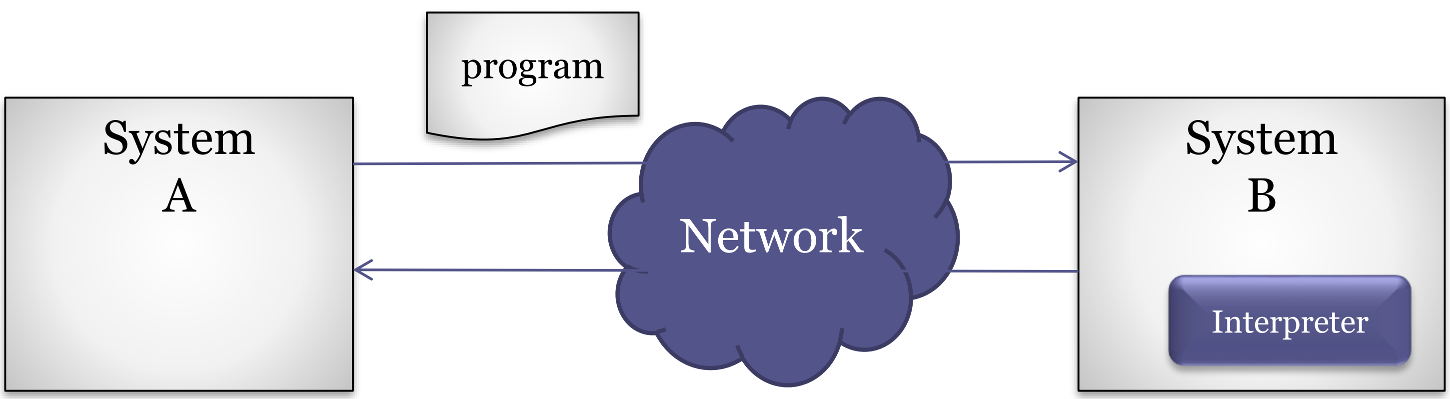
- Interpreter: Runs the code
- Program: Program that is transferred
- Network: Transfers the program



Mobile code

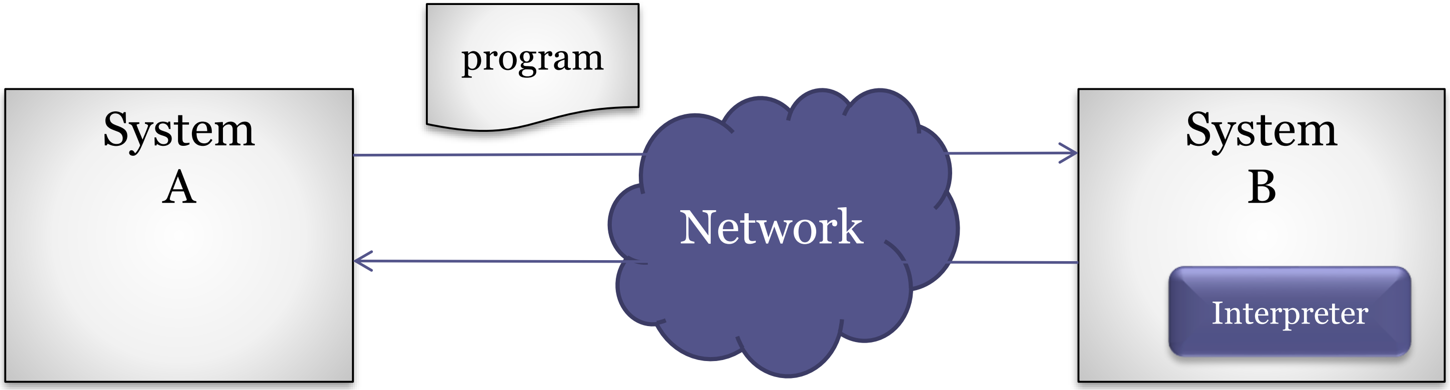
Constraints

- The program must be run in the receiver system
- The network protocol transfers the program



Mobile code

- Advantages
  - Flexibility and adaptability to new environments
  - Parallelism
- Challenges
  - Complexity of implementation
  - Security



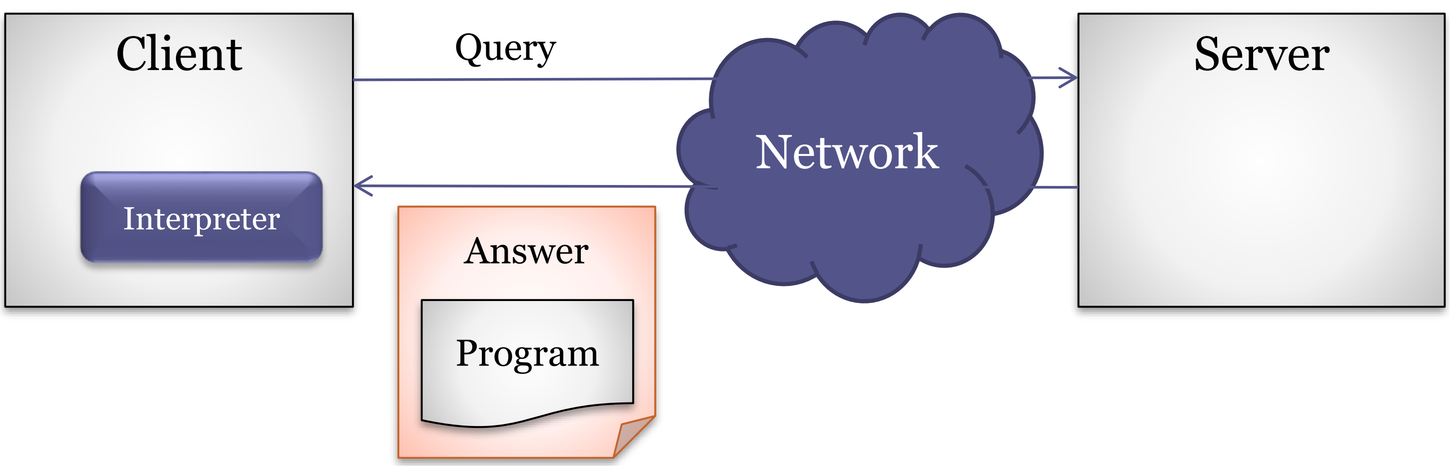
Mobile code

Variants

- Code on demand
- Remote evaluation
- Mobile Agents

Code on demand

- Code is downloaded and run by the client
- Combination between mobile code and client-server
- Example:
  - ECMAScript



Code on demand

- Elements
  - Client
  - Server
  - Code that is transferred from server to client
- Constraints
  - Code resides or is generated by the server
  - It is transferred to the client when it asks for it
  - It is run by the client
    - Client must have an interpreter for the corresponding language

Code on demand

Advantages

- Improves user experience
- Extensibility: Application can add new functionalities that were not foreseen
  - No need to install or download a whole application
  - Always *Beta*
- Adaptability to client environment

Challenges

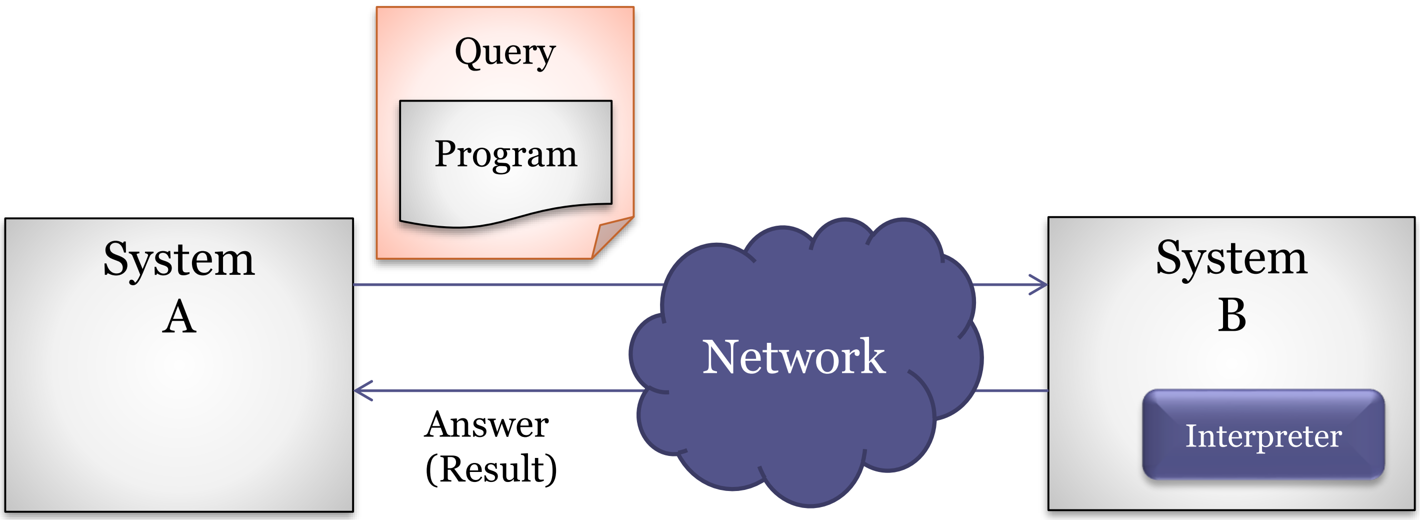
- Security
- Coherence
  - It may be difficult to ensure an homogeneous behavior in different types of clients
    - Client can even decide not to run the program
      - Reminder: Responsive design

Code on demand

- Applications:
  - RIA (Rich Internet Applications)
    - HTML5 standardizes a lot of APIs
    - Improves coherence between clients
- Variants
  - AJAX
    - Initially: *Asynchronous Javascript and XML*
    - The program that is running at the client side sends asynchronous requests to the server without stopping its running

Remote evaluation

- System A sends program to system B to be run and obtain its results



Remote evaluation

- Elements
  - Sender: Does the query including the program
  - Receiver: Runs the program and returns the results
- Constraints
  - Receiver runs the program
    - It must contain some interpreter of the program language or the program could be in machine code
  - Network protocol transfers program and results

Remote evaluation

- Advantages
  - Exploits capabilities of third parties
    - Computational capabilities, memory, resources, etc.
- Challenges
  - Security
    - Untrusted code
    - Virus = variant of this style
  - Configuration

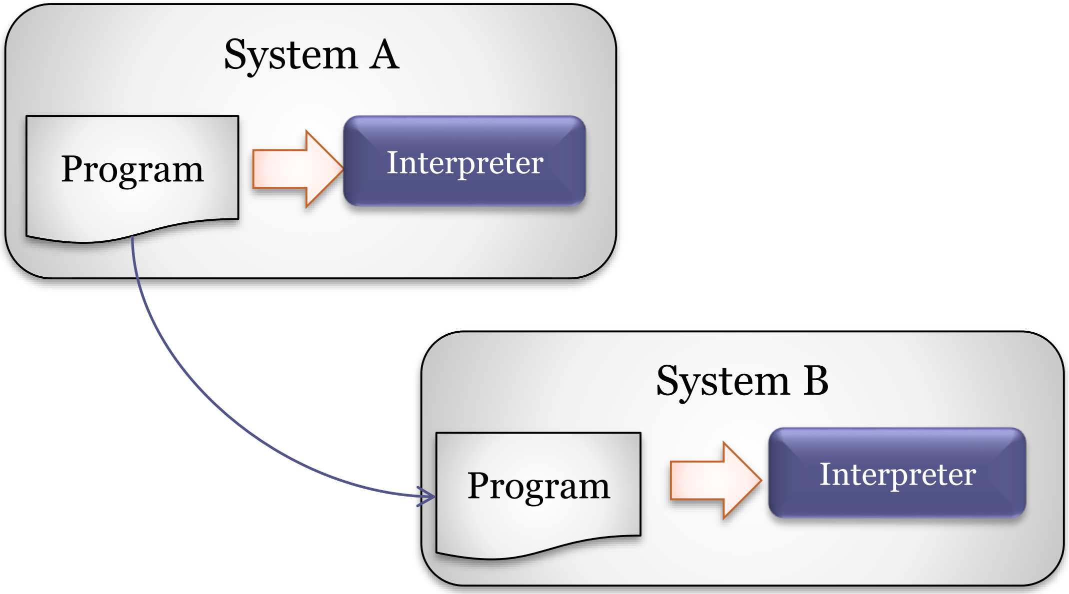
Remote evaluation

Example:

- Volunteer computation
  - SETI@HOME
    - It was the basis for the BOINC system
      - Berkeley Open Infrastructure for Network Computing
  - Other projects: Folding@HOME, Predictor@Home, AQUA@HOME, etc.

Mobile agents

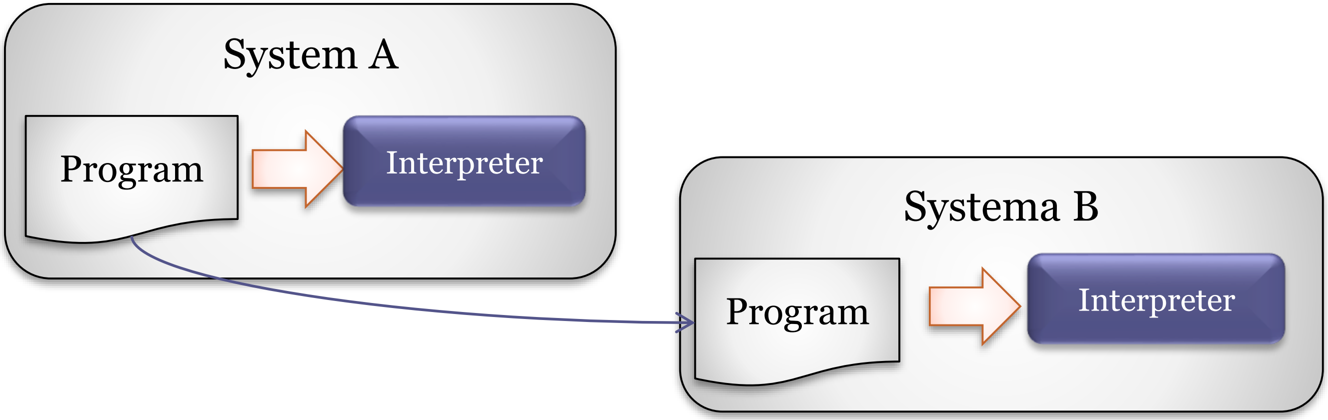
- Code and data can move from one machine to another to be run
  - The process takes its state from machine to machine
  - Code can move autonomously



Mobile agents

Elements

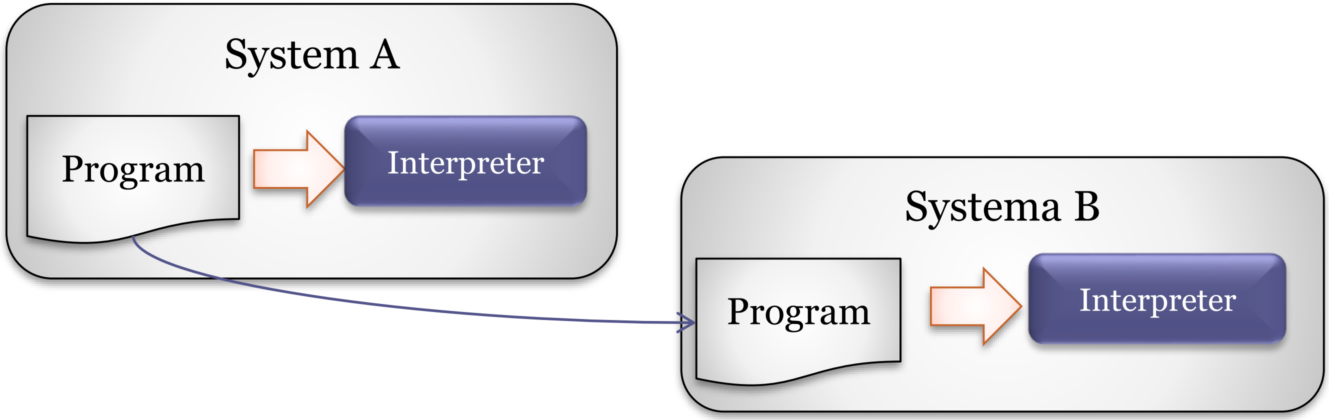
- Mobile agent: Program that travels and is run from one machine or another autonomously
- System: Execution environment where the mobile agents are run
- Network protocol: transfers state between agents



Mobile agents

Constraints

- Systems host and run mobile agents
- Mobile agents can decide to change its running from one system to another
  - They can communicate with other agents



Mobile agents

Advantages

- It can reduce network traffic
  - Code blocks that are run are transmitted
- Implicit parallelism
- Fault tolerance to network failures
- Agents can be conceptually simple
  - Agent = independent unit of execution
  - It is possible to create mobile agent systems
  - Emergent behaviour
- Adaptability to environtment changes
  - Reactive and learning systems

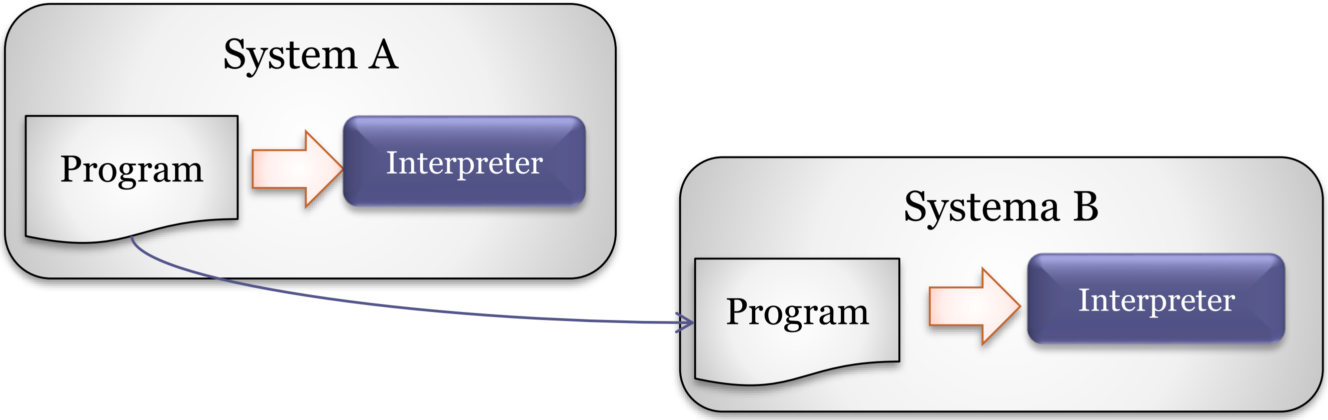
Challenges

- Complexity of configuration
- Security
  - Malicious or incorrect code

Mobile agents

Challenges

- Complexity of configuration
- Security
  - Malicious or incorrect code



Mobile agents

- Applications
  - Information retrieval
    - Web crawlers
  - Peer-to-peer systems
  - Telecommunications
  - Remote control and monitoring
- Systems:
  - JADE (Java Agent DEvelopment framework)
  - IBM Aglets

End of presentation