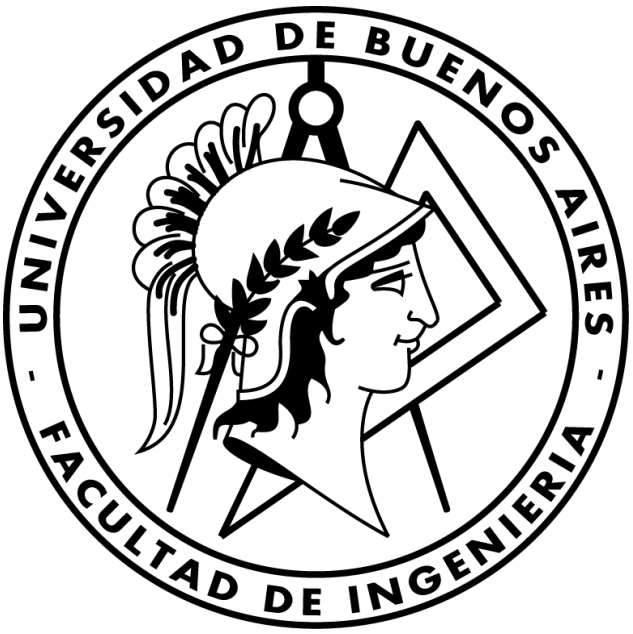



# Documentación Trabajo Práctico 2

Middleware y Coordinación de Procesos

Sistemas Distribuidos I 75.74

Primer Cuatrimestre 2021

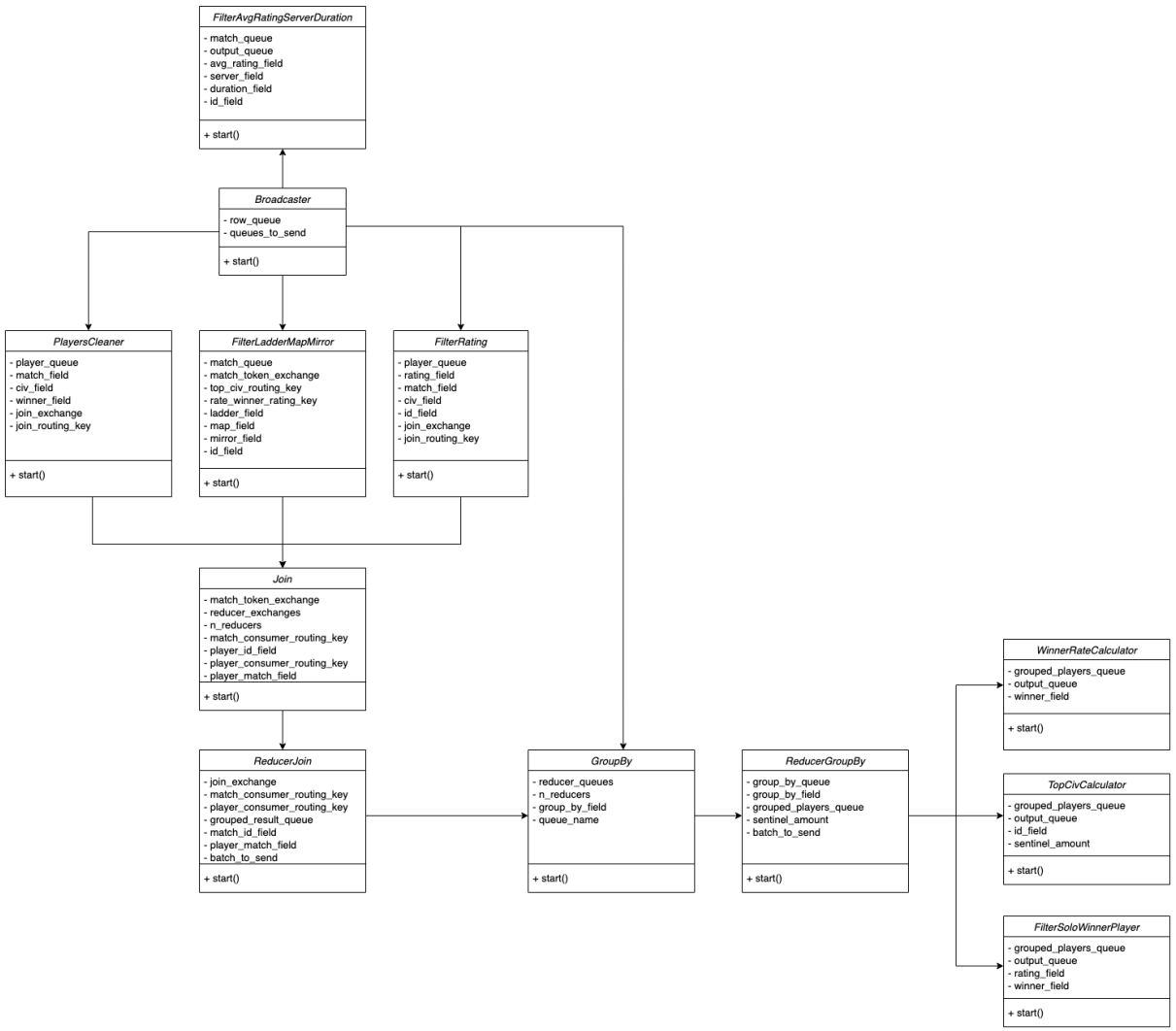
Cecilia Hortas 100687

# Introducción

En este documento se presentarán las distintas vistas de arquitectura para el desarrollo del trabajo práctico. El objetivo del mismo es implementar un sistema distribuido que procese dos datasets del videojuego Age of Empires con el fin de resolver consultas determinadas previamente.

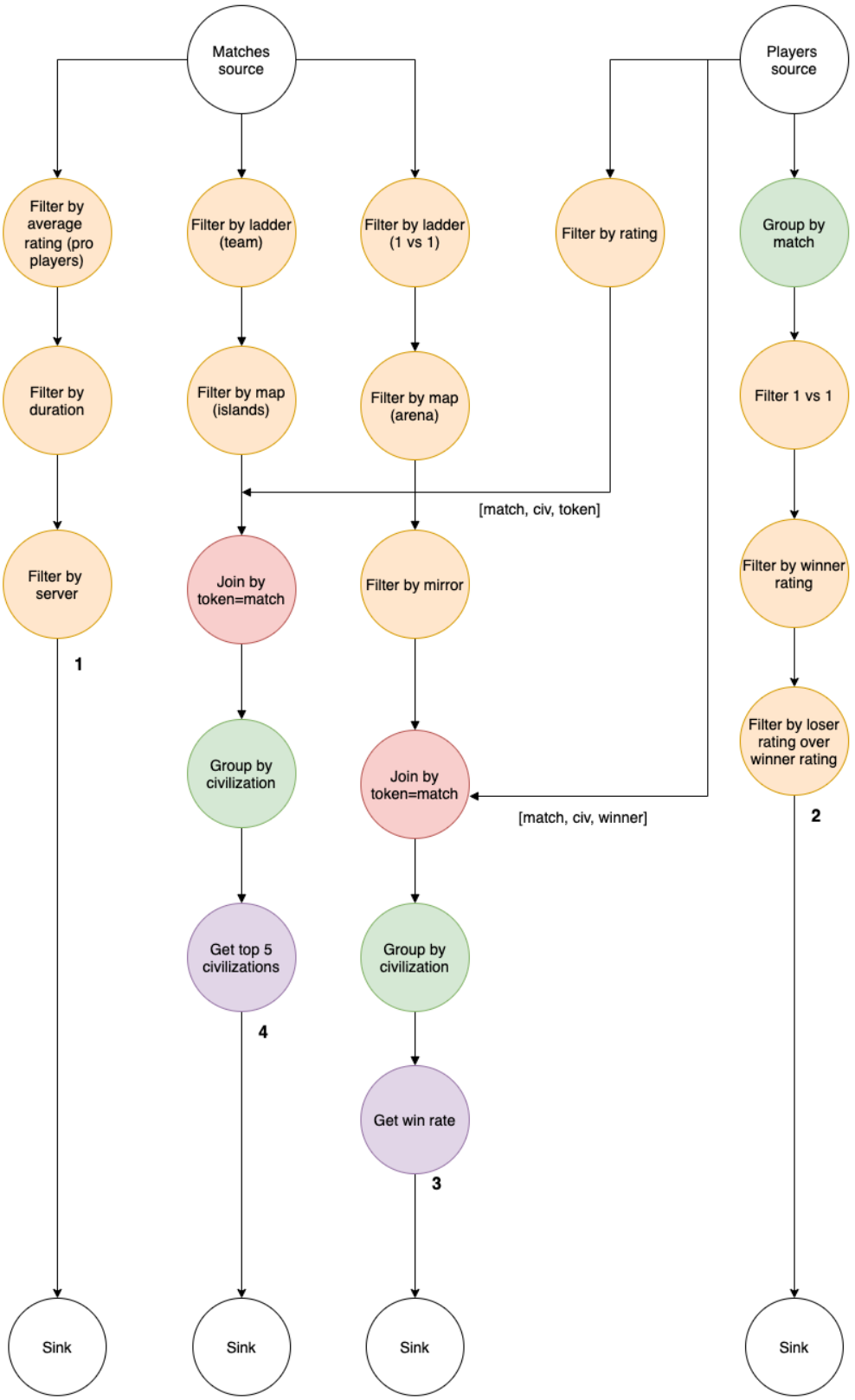
# Vista lógica

## Diagrama de clases




En el diagrama se presentan las clases que conforman el sistema y además se muestran las distintas comunicaciones que existen entre las mismas. Cada flecha del sistema simboliza una comunicación entre las clases involucradas.

DAG

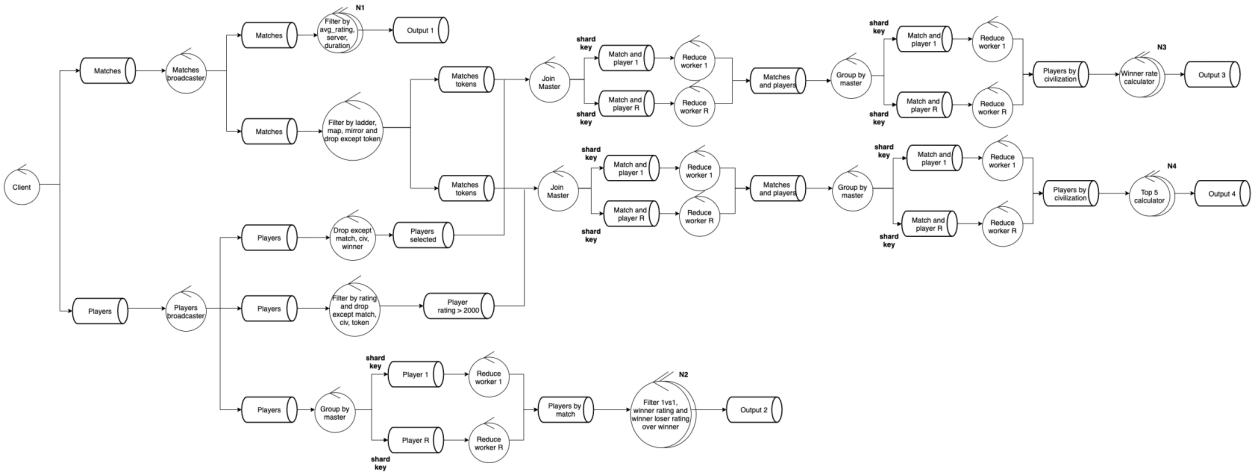



1. Ids from matches that lasted more than two hours by pro players in the servers koreacentral, southeastasia y eastus
2. Ids from matches where the player with less rating (30% difference) won the game in 1v1 games (rating > 1000)
3. Civilization win rate in 1v1 games in map arena
4. Usage rate of every civ used by pro players (rating > 2000) in team matches in map islands

Se presenta el grafo dirigido acíclico con el que se modelaron las consultas del sistema. Se puede observar que hay 4 flujos para resolver las 4 queries requeridas. Se modelan con distintos colores los nodos que se encargan de filtrar, agrupar, juntar y calcular. En la implementación propiamente dicha algunos nodos se acoplaron para reducir el overhead que aportan las colas de rabbit pero en un sistema ideal se hubiera implementado cada uno de los nodos por separado.

# Vista física

## Diagrama de robustez




Se esquematizan en el diagrama los distintos nodos del sistema y cómo se comunican entre sí. Como regla general, cada nodo procesa los datos de entrada y escribe los resultados en una cola.

Se puede observar que algunos nodos tienen un número $N_i$ y esto significa que pueden ser replicados en el sistema. El motivo por el cual hay nodos que pueden ser replicados y otros no es por el esquema de coordinación que requiere el envío de un centinela para poder propagar los datos hacia un próximo nodo. Si una cantidad de nodos n lee de una misma cola una cantidad n de centinelas no hay nada que asegure que serían consumidos por el mismo nodo ya que todos están compitiendo por el mismo recurso. De esto podría desprenderse la idea de enviar a cada nodo un centinela pero el mismo podría estar al mismo tiempo procesando otra fila y en consecuencia enviar el centinela antes que la misma por lo que no hay una solución directa a este problema que no implique asumir que estos nodos no pueden ser replicados. Además, como el cuello de botella del procesamiento está en el join y en el group by y los reducers pueden escalarse no se considera que este enfoque no estuviera optimizado para un entorno multicomputing.

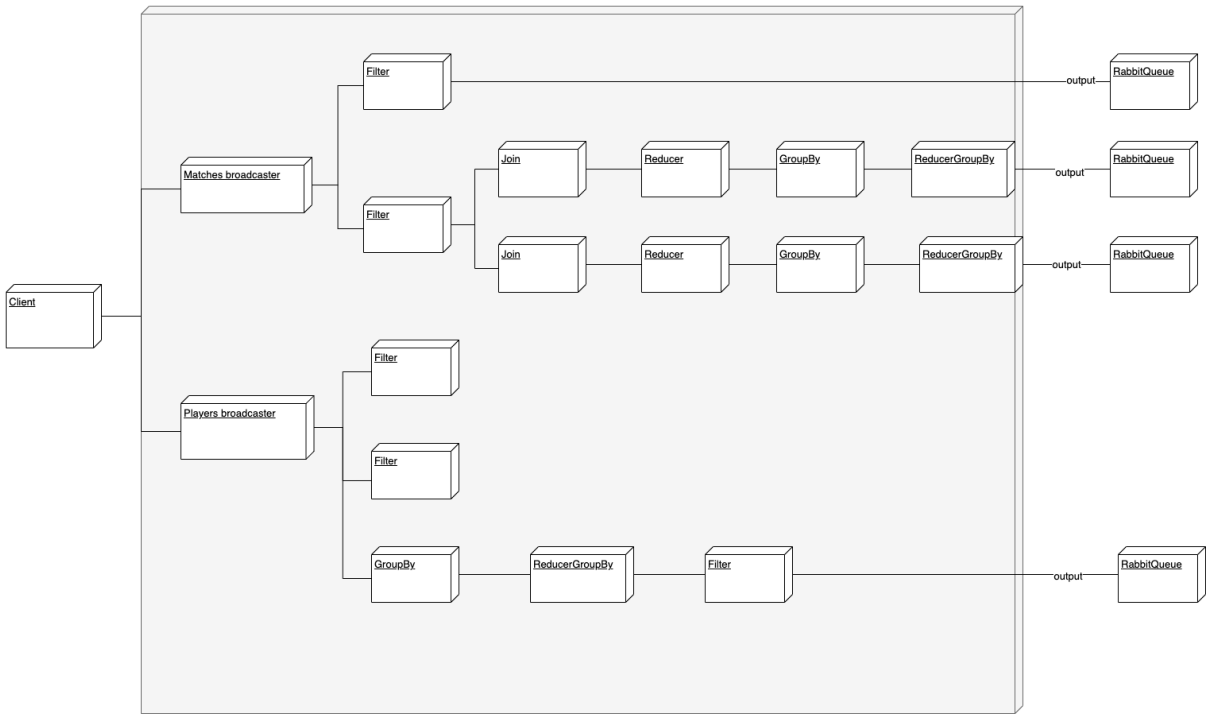
A continuación se procede a explicar la responsabilidad de cada nodo:

- MatchesBroadcaster: se encarga de recibir los partidos por una cola de entrada y propagarlos a las distintas colas que requieran procesar partidos
- PlayersBroadcaster: se encarga de recibir los jugadores por una cola de entrada y propagarlos a las distintas colas que requieran procesar jugadores

- FilterAvgRatingServerDuration: se encarga de filtrar los partidos recibidos por una cola de entrada de acuerdo a su rating promedio, el servidor en el cuál se jugó la partida y su duración. Escribe los resultados en una cola de salida.
- PlayersCleaner: se encarga de purgar los jugadores de los campos que no serán requeridos en la consulta
- FilterLadderMapMirror: se encarga de filtrar los partidos recibidos por una cola de entrada de acuerdo a si son partidos grupales o 1vs1, el mapa en el cual se jugó y si correspondía a civilizaciones diferentes
- FilterRating: se encarga de filtrar los jugadores recibidos por una cola de entrada de acuerdo al rating
- GroupBy: se encarga de hashear los elementos recibidos por una cola de entrada de acuerdo a un campo configurable y enviarlos al reducer que corresponda al resultado obtenido.
- ReducerGroupBy: recibe una serie de campos y se encarga de agruparlos de acuerdo a un campo configurable. Una vez que recibió todos los datos a procesar (detectado por el envío de un centinela), propaga el resultado al próximo nodo que corresponda
- Join: se encarga de hashear los elementos recibidos por una cola de entrada de acuerdo a un campo configurable y enviarlos al reducer que corresponda al resultado obtenido.
- ReducerJoin: recibe una serie de campos y se encarga de juntarlos de acuerdo a un campo configurable. Una vez que recibió todos los datos a procesar (detectado por el envío de un centinela), propaga el resultado al próximo nodo que corresponda
- FilterSoloWinnerPlayer: se encarga de filtrar los jugadores agrupados por partido recibidos por una cola de entrada de acuerdo a si participaron en un partido 1vs1, si el ganador tiene un rating mayor a 1000 y además un 30% menor que el del perdedor
- WinnerRateCalculator: se encarga de recibir los partidos y jugadores agrupados por civilización y calcular el promedio de victorias de cada civilización
- TopCivCalculator: se encarga de calcular cuáles son las civilizaciones más populares

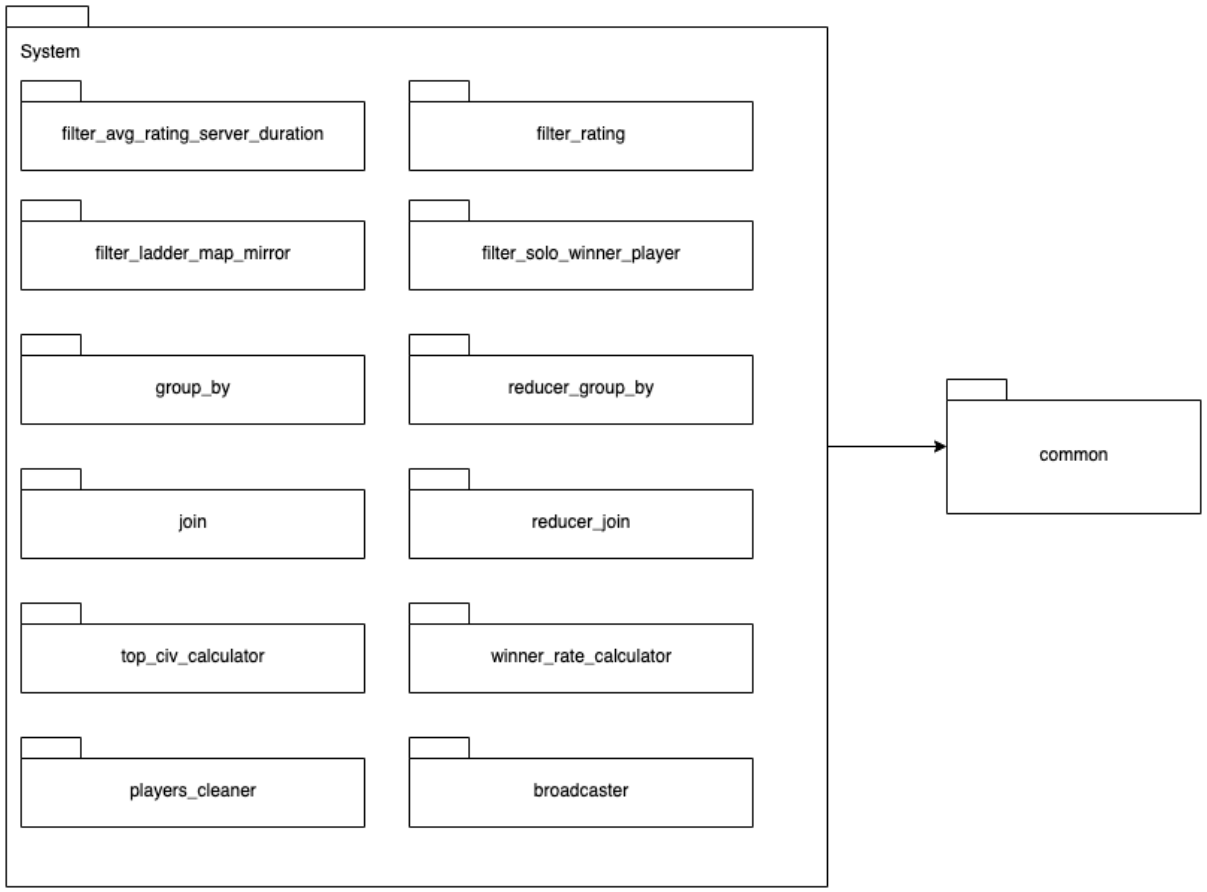
## Diagrama de despliegue

A continuación se presenta el diagrama de despliegue del sistema. Se simplifican los nombres de los distintos nodos ya que guardan semejanza con los del diagrama de robustez. Se busca mostrar en este diagrama qué nodos constituyen el sistema y cuáles son los que ingresan la entrada y consumen la salida.




# Vista de desarrollo

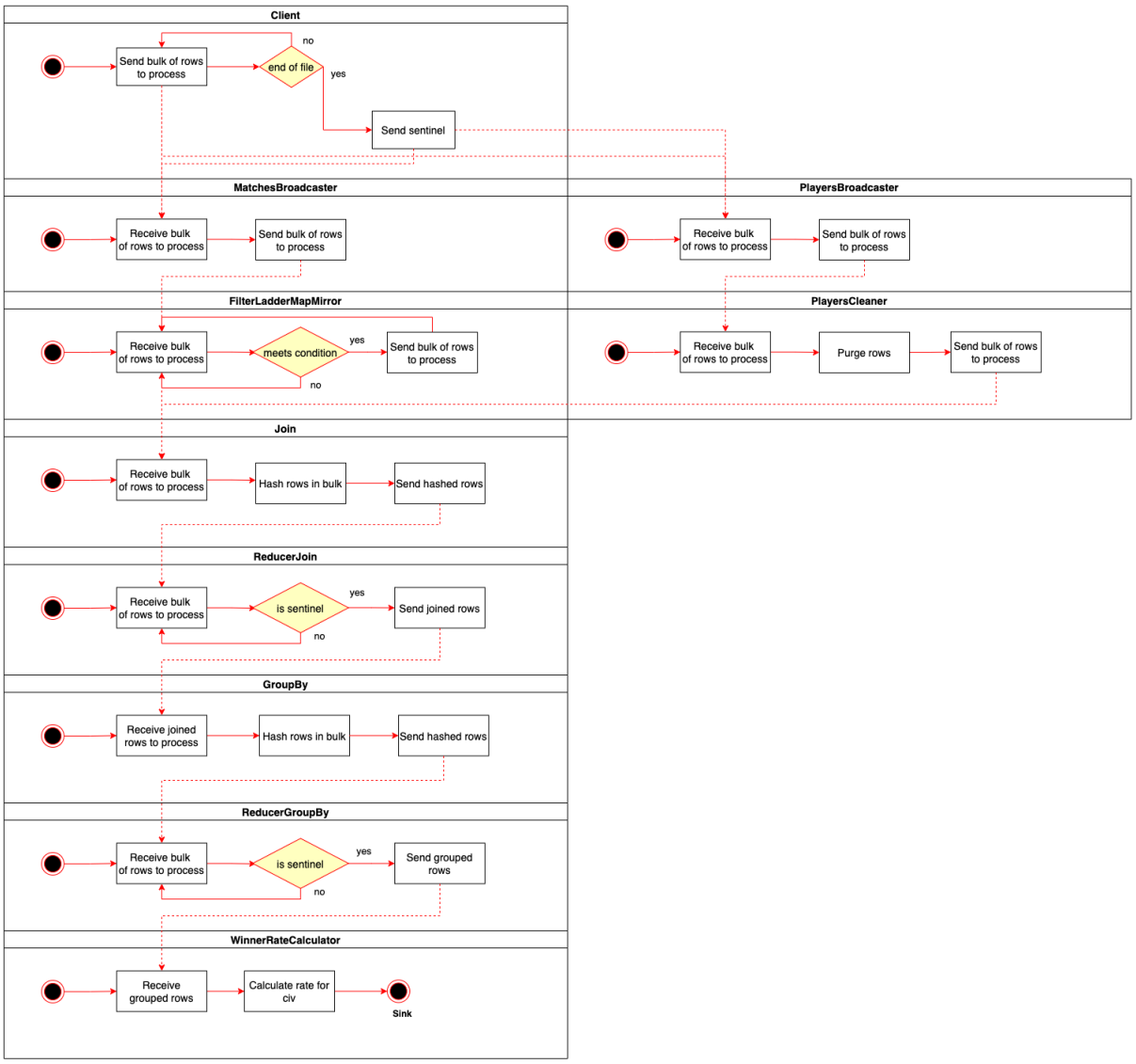
## Diagrama de paquetes



Se expone el diagrama de paquetes para evidenciar la agrupación de los elementos UML del sistema que están relacionados. Lo que se busca mostrar es el uso de un paquete externo llamado common que guarda las distintas funciones de rabbit que se reutilizaron a lo largo del sistema.

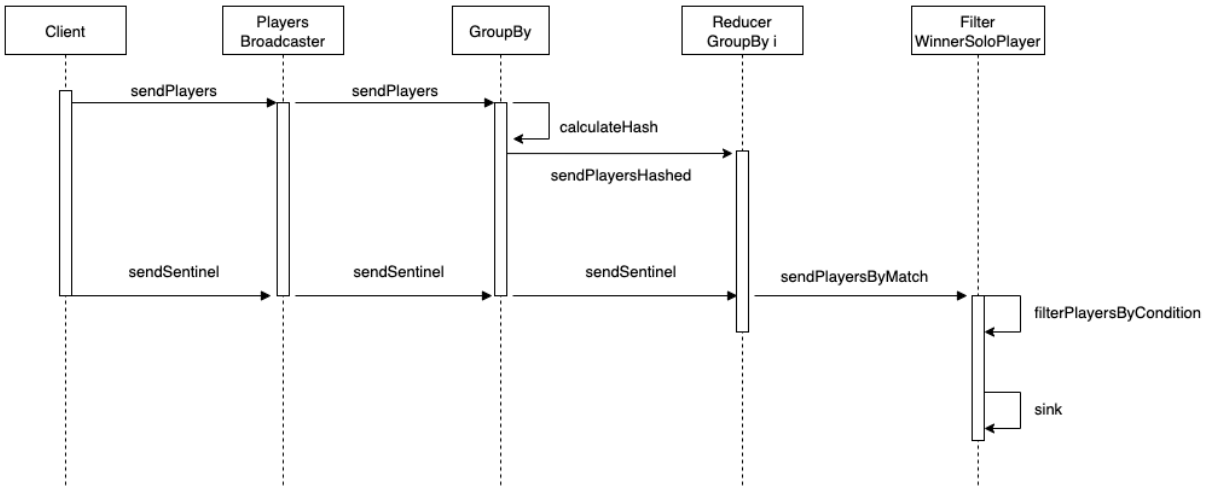
# Vista de procesos

## Diagrama de actividad




En el presente diagrama se esquematizan los distintos pasos que se llevan a cabo para ejecutar la query para hallar el porcentaje de victorias por civilización en partidas 1v1 con civilizaciones diferentes en mapa arena. Se decidió primero filtrar la mayor cantidad de filas posible con el fin de mejorar la performance del join. Por eso primero se eliminan columnas de los jugadores y se filtran los partidos según el mapa, tipo de juego y el campo mirror (civilizaciones diferentes en juego) y luego se realiza el join.

## Diagrama de secuencia




Se expone el diagrama de secuencia para la query que busca obtener los ids de los partidos 1v1 donde el ganador tiene un rating 30% menor al perdedor y el rating del ganador es superior a 1000. Como se puede observar se tuvo que realizar primero la operación para agrupar los jugadores por partida y luego poder realizar los filtros correspondientes para una partida 1v1 de acuerdo a los ratings de los dos jugadores involucrados.