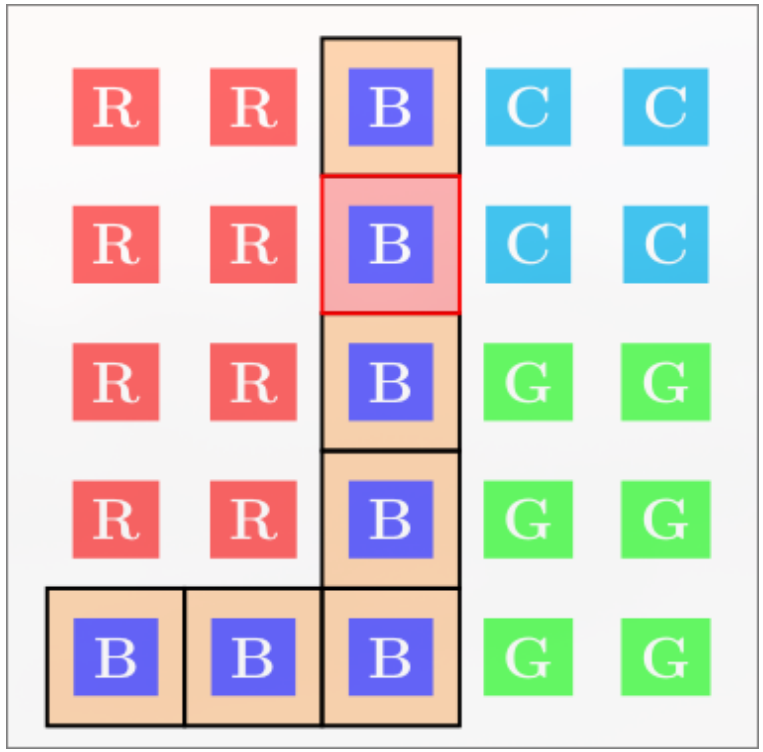
Implementazione Distribuita dell'Algoritmo Flood-Fill

Tam Gabriele, Merlo Filippo

Descrizione del Problema

- Il problema consiste nel ricolorare un'area connessa di una matrice bidimensionale `img[N][M]`.
- Dato un pixel iniziale in posizione `(x, y)` e un nuovo colore `newColor`, l'obiettivo è cambiare il colore del pixel selezionato e dei pixel adiacenti con lo stesso colore iniziale.
- La ricolorazione si propaga in tutte le direzioni (**orizzontale**, **verticale**, **diagonale**).



Requisiti del Sistema

Requisiti Funzionali:

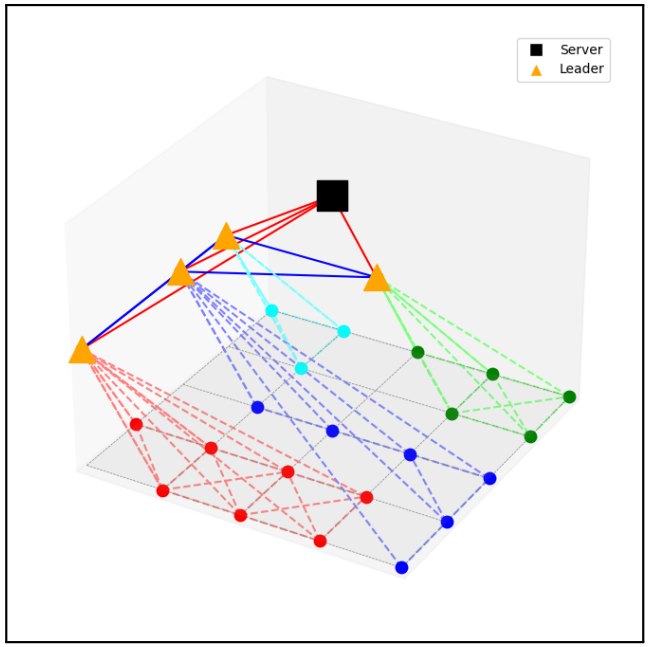
1. Operazione di colorazione distribuita
2. Gestione di richieste concorrenti
3. Comunicazione tra nodi
4. Unione di cluster
5. Tolleranza ai guasti
6. Consistenza globale

Requisiti Non Funzionali:

- Scalabilità
- Efficienza delle comunicazioni
- Robustezza e affidabilità
- Consistenza

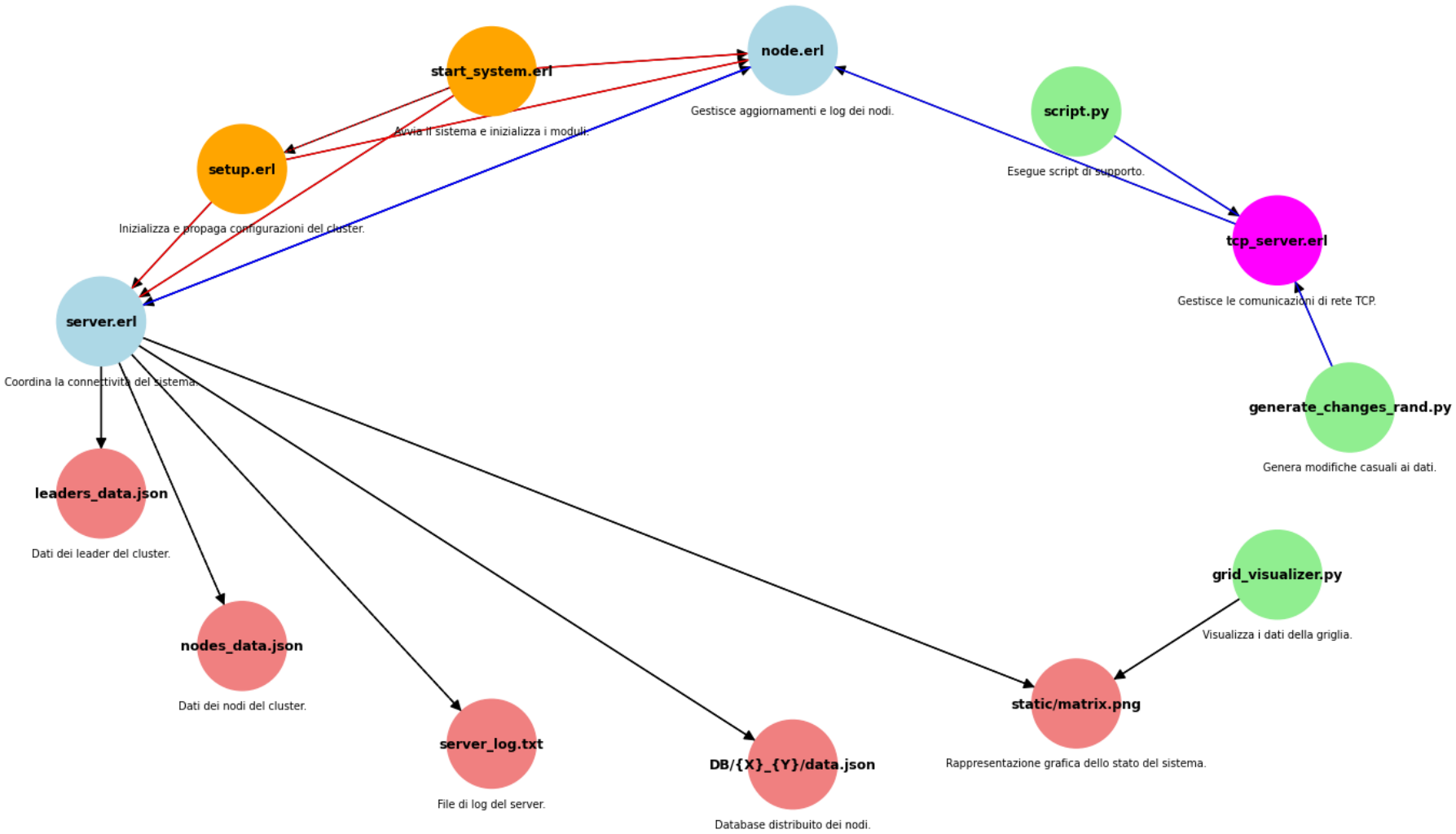
Soluzione Proposta: Idea Principale

- Trattamento dei **cluster** come nodi in un **grafo overlay**.
- Divisione dei nodi in **nodi normali** e **nodi leader**, con comunicazione tra leader.
- Utilizzo di un **server centrale** per la **sincronizzazione**, che gestisce anche il **File log** e lo **stato globale** dei cluster.



Codice disponibile su: **GitHub - Distributed Flood Fill**

Architettura Proposta



Algoritmi: Setup Iniziale

Scoperta dei Vicini

Input: Coordinate del nodo (x, y) e soglia di vicinanza d_{max}
Output: Lista dei vicini *neighbors*

1. Inizializza *neighbors* $\leftarrow \emptyset$
2. Per ogni nodo n' nel sistema:
   - Calcola la distanza d tra il nodo corrente e n'
   - Se $d \leq d_{max}$, aggiungi n' alla lista *neighbors*

Algoritmi: Setup Iniziale

Formazione dei Cluster

Input: Lista dei vicini *neighbors*

Output: Lista dei nodi del cluster *cluster_nodes*

1. Inizializzazione:
   - Per ogni nodo n nel sistema, imposta $Visited[n]$ a *False*.
2. Identificazione dei cluster:
   - Per ogni nodo n nel sistema, se $Visited[n]$ è *False*:
     - Imposta $Visited[n]$ a *True*.
     - Avvia una ricerca in ampiezza (BFS) da n per individuare tutti i nodi con lo stesso colore.
     - Imposta n come leader del cluster e aggiungilo alla lista *cluster_nodes*.
     - Per ogni nodo n' scoperto durante la BFS:
       1. Aggiungi n' a *cluster_nodes*.
       2. Imposta $Visited[n']$ a *True*.

Algoritmi: Setup Iniziale

Scoperta dei Cluster Adiacenti e Creazione della Rete Overlay

Input: *Lista dei nodi del cluster* (*cluster_nodes*), *Lista dei vicini* (*neighbors*)
Output: *Lista dei cluster adiacenti* (*adj_clusters*)

1. Il leader del cluster comunica con ciascun nodo *n* del proprio cluster:
   - Invia a ciascun nodo *n* la lista *cluster_nodes*, contenente i nodi appartenenti al cluster.
2. Ogni nodo *n* esegue i seguenti passi:
   - Contatta i nodi vicini appartenenti a cluster differenti e richiede la tupla {pid, colore, leader}.
   - Invia l'insieme delle tuple ricevute al proprio leader.
3. Il leader costruisce la lista dei cluster adiacenti, includendo quelli con un colore differente e memorizzando il rispettivo leader.

Algoritmi: Setup Iniziale

Conclusione del Setup

Input: Lista dei nodi

Output: Stato del sistema sincronizzato

1. Per ogni nodo n non leader:
   - Rimuovi informazioni non necessarie da n
   - Mantieni solo: *pid*, (x, y), *leaderID*, e *neighbors*
2. Per ogni leader:
   - Comunica le informazioni del proprio cluster al server
   - Il server salva lo stato globale del sistema

Algoritmo di Cambio Colore - Mittente

Cambio Colore - Mittente

Input:
- Leader del cluster C che desidera cambiare colore
- Nuovo colore *new_color*

Output:
- Invio di update ai cluster adiacenti
- Verifica necessità di merge

1. Richiedi e attendi l'autorizzazione dal server.
2. Aggiorna :
   - Il colore del cluster: $color(C) \leftarrow new\_color$.
   - Il timestamp: $last\_event(C) \leftarrow$ timestamp corrente.
3. Inizializza la variabile *merge_needed* a *False*.
4. Per ogni cluster adiacente C' in $adj\_clusters(C)$:
   - Se $color(C') = new\_color$, imposta $merge\_needed \leftarrow True$.
5. Invia messaggi di aggiornamento del colore ai cluster adiacenti.
6. Se è necessario un merge, avvia l'algoritmo di fusione.
7. Notifica al server la conclusione dell'operazione.

Algoritmo di Cambio Colore - Destinatario

Cambio Colore - Destinatario

Input:

– Messaggio `color_adj_update`(C_{src}, *new_color*) ricevuto dal cluster C

Output:

– Aggiornamento delle strutture dati del cluster destinatario

1. Per ogni entry ($pid, col, leaderID$) in $adj\_clusters(C)$:
   - Se $leaderID = leader(C_{src})$, aggiorna $col \leftarrow new\_color$
2. Il nodo leader aggiorna eventuali strutture locali
3. Aggiorna $last\_event(C)$ con il timestamp corrente

Algoritmo di Merge - Mittente

Merge - Mittente

Input:

– Leader del cluster C che ha rilevato un cluster confinante C' con lo stesso colore

Output:

– Unione dei due cluster (se accettata) e notifica al server

1. Per ogni C' in $adj\_clusters(C)$:
   - Se $color(C') = color(C)$, invia messaggio `merge_request(C)` a C'

Algoritmo di Merge - Destinatario

Merge - Destinatario

Input:

- Messaggio `merge_request`(C_{src}) ricevuto dal cluster C_{dst}
- Leader del cluster destinatario L_{dst}

Output:

- Unione dei due cluster (se accettata) e aggiornamento strutture dati

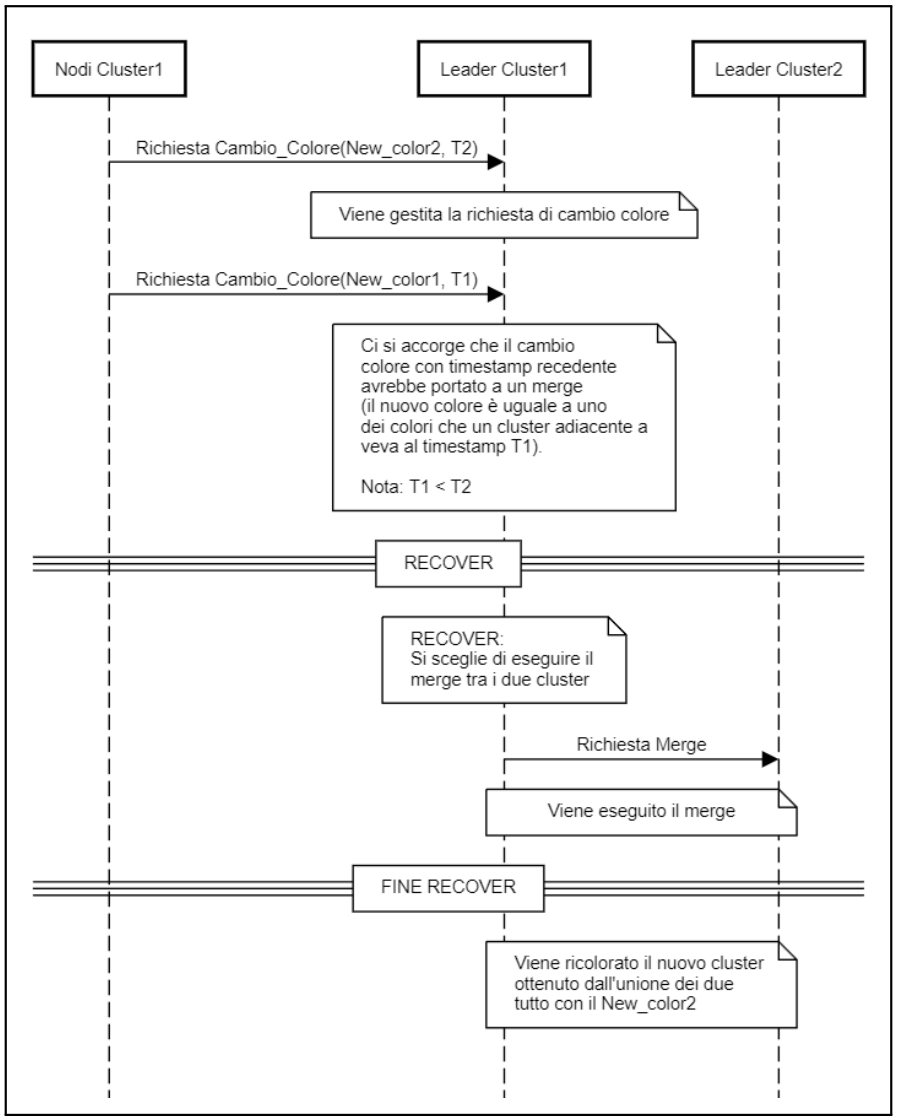
1. Se le politiche interne permettono la fusione:
   - Unisci i cluster: $C_{merged} \leftarrow C_{src} \cup C_{dst}$
   - Determina il nuovo leader L_{new}
   - Aggiorna la lista $adj\_clusters$ per C_{merged}
   - Notifica il server della fusione con `merge_done`(C_{merged})
2. Altrimenti:
   - Invia messaggio di rifiuto `merge_refused`(C_{dst}) a C_{src}

Gestione della Consistenza

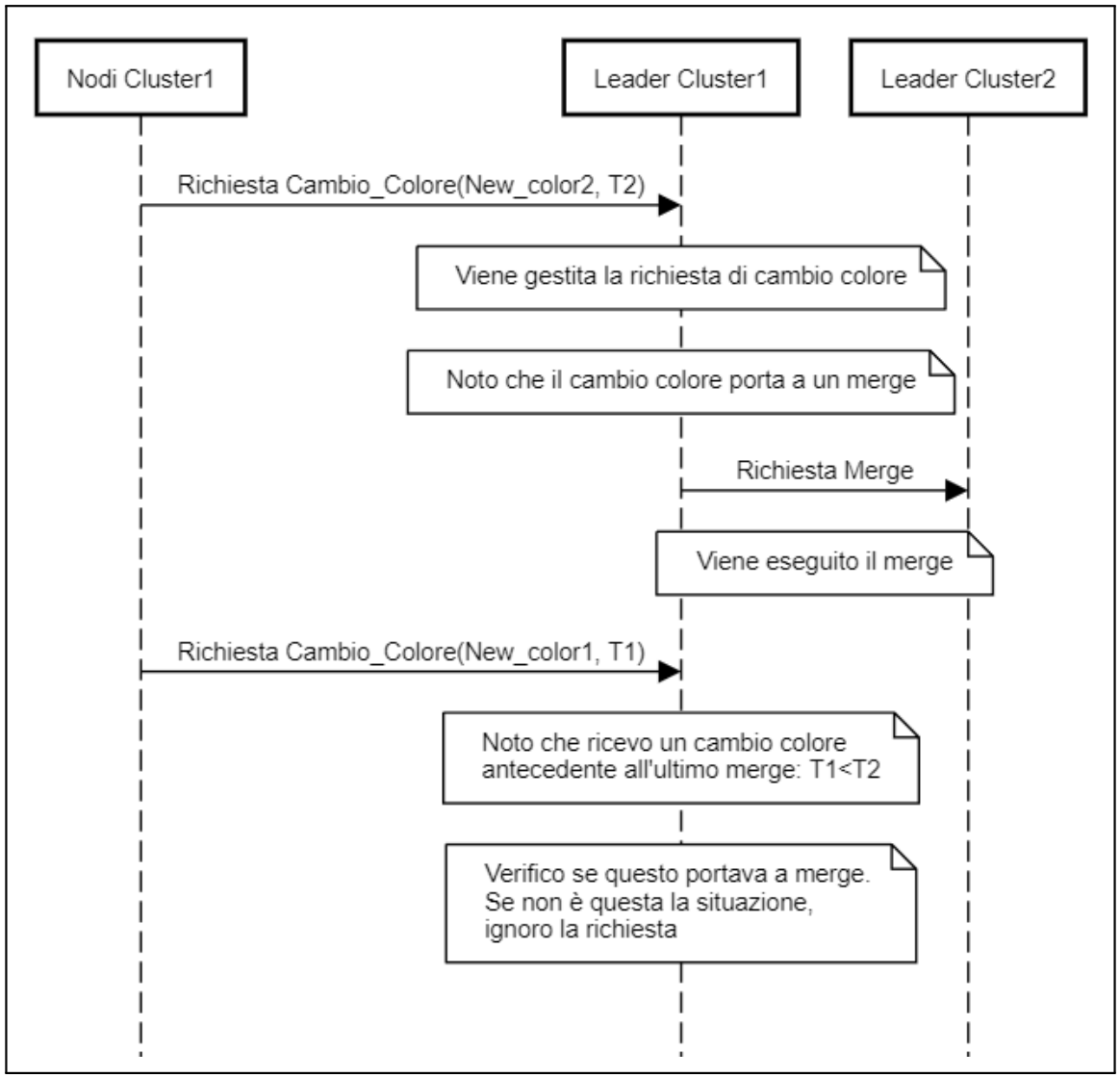
- **Caso 1**: Cambio Colore con Timestamp Anteriore (Merge necessario)
- **Caso 2**: Cambio Colore con Timestamp Anteriore (Merge non necessario)
- **Caso 3**: Richiesta di Cambio colore con Timestamp Anteriore, ricevuta Durante un Merge
- **Caso 4 ("Too Old")**: Gestione di una Richiesta con Timestamp Troppo Vecchio
- **Caso 5 ("Change Color During Merge")**: Richiesta di Cambio Colore Ricevuta Durante un Merge
- **Caso 6 ("Double Merge")**: Doppio Merge tra Cluster Adiacenti con Cambio Colore

Codice disponibile su: **GitHub - Distributed Flood Fill**

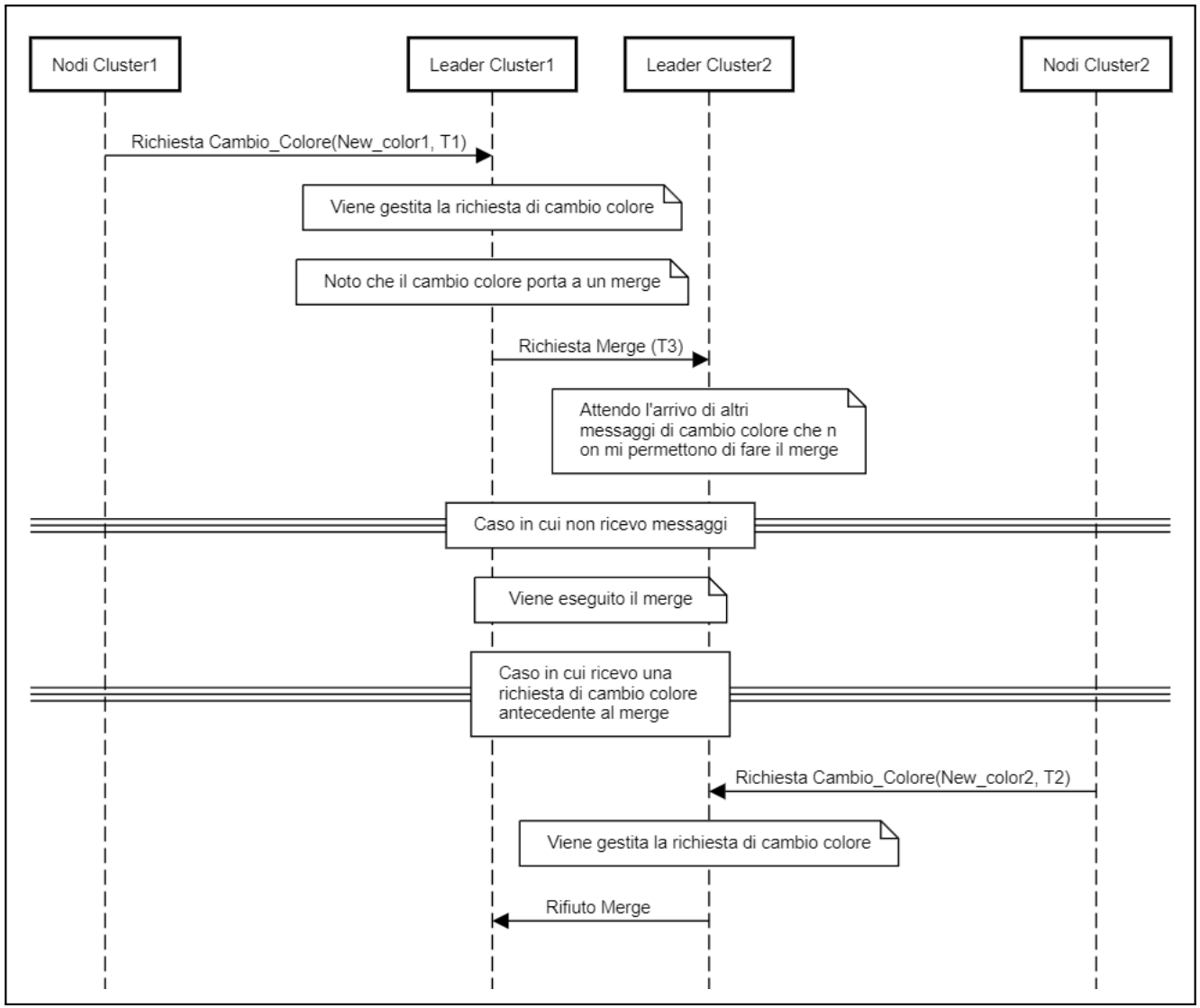
Caso 1: Cambio Colore con Timestamp Anteriore (Merge necessario)



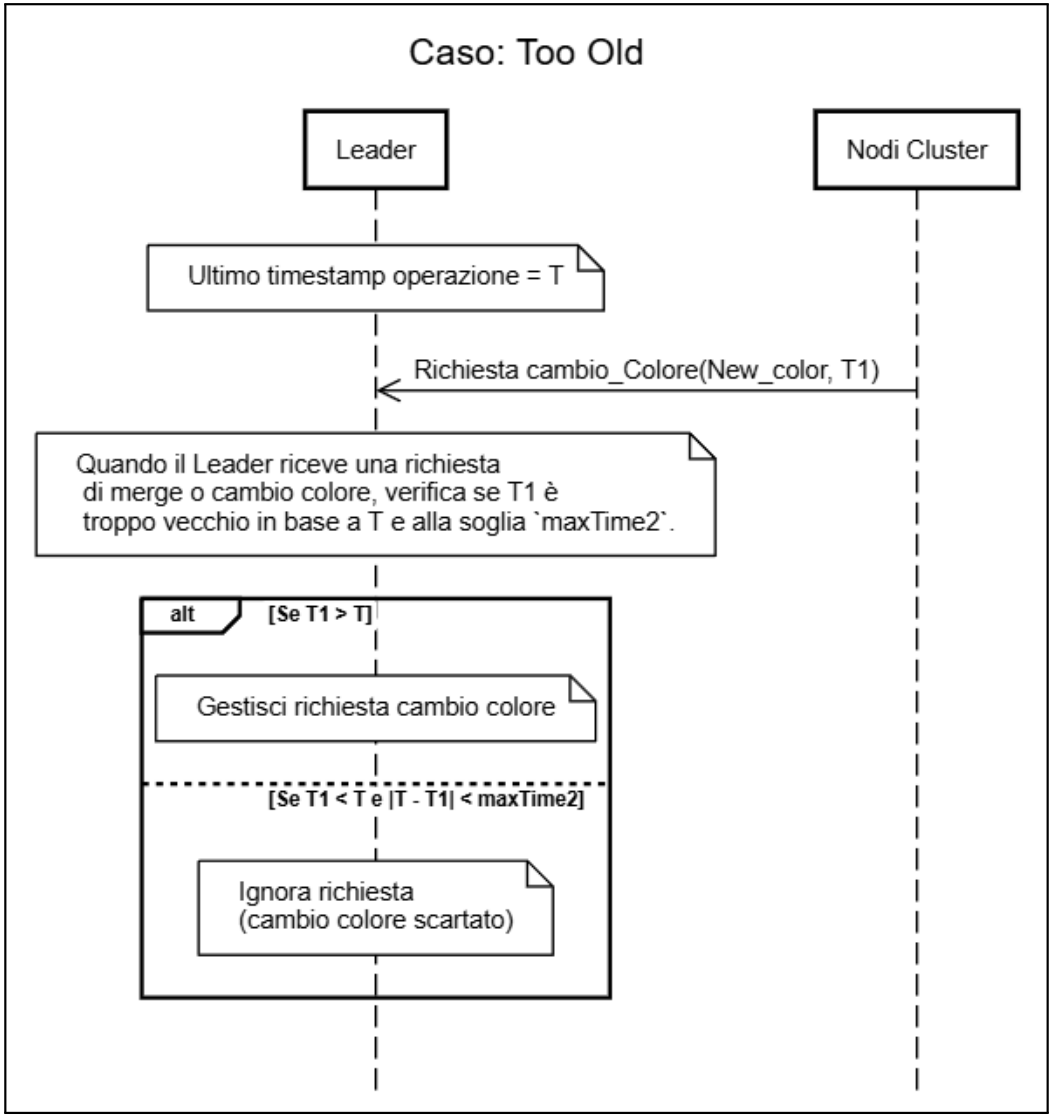
Caso 2: Cambio Colore con Timestamp Anteriore (Merge non necessario)



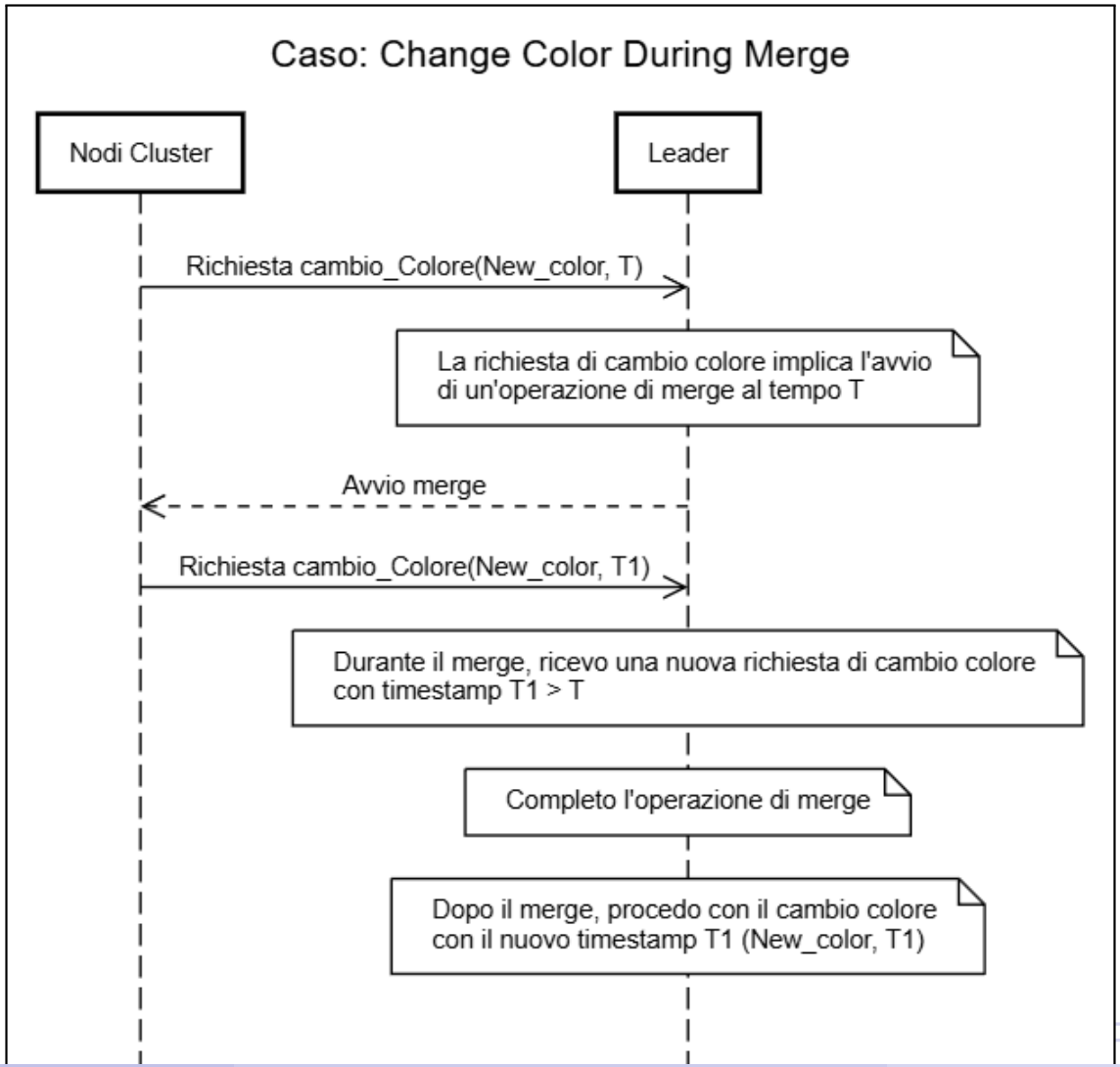
Caso 3: Richiesta di Cambio colore con Timestamp Anteriore, ricevuta Durante un Merge



Caso 4 ("Too Old"): Gestione di una Richiesta con Timestamp Troppo Vecchio



Caso 5 ("Change Color During Merge"): Richiesta di Cambio Colore con Timestamp Superiore, Ricevuta Durante un Merge



Caso 6 ("Double Merge"): Doppio Merge tra Cluster Adiacenti con Cambio Colore