Pippolo

un DB NOSQL distribuito

Caratteristiche

- Comunicazioni in XML puro
- Alta affidabilità personalizzabile
- Grafo espandibile in qualsiasi momento
- Auto-sincronizzazione dei nuovi Pippoli
- Scritto interamente in C e basato su librerie standard POSIX *(compilabile ovunque senza dipendenze esterne)*

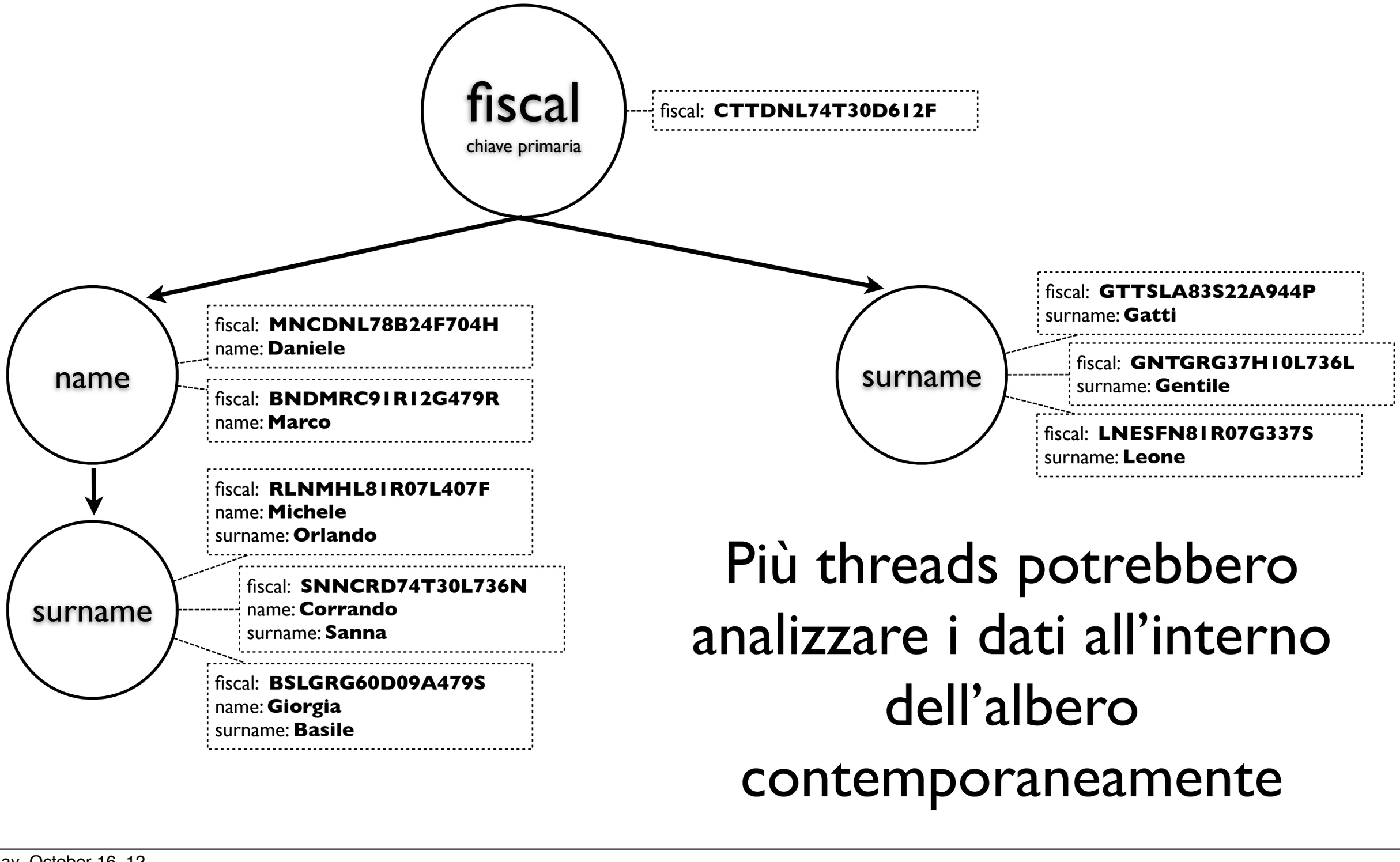
Caratteristiche

- Ogni Pippolo ha un range di copertura in un set di 10 elementi (da 0 a 9)
- Ogni dato deve essere accompagnato da un valore hash (che vada da 0 a 9) calcolato dall'utente in base ad una funzione che discretizzi la chiave primaria
- Solo i nodi che coprono l'hash ricevuto si occuperanno di una azione entrante

Caratteristiche

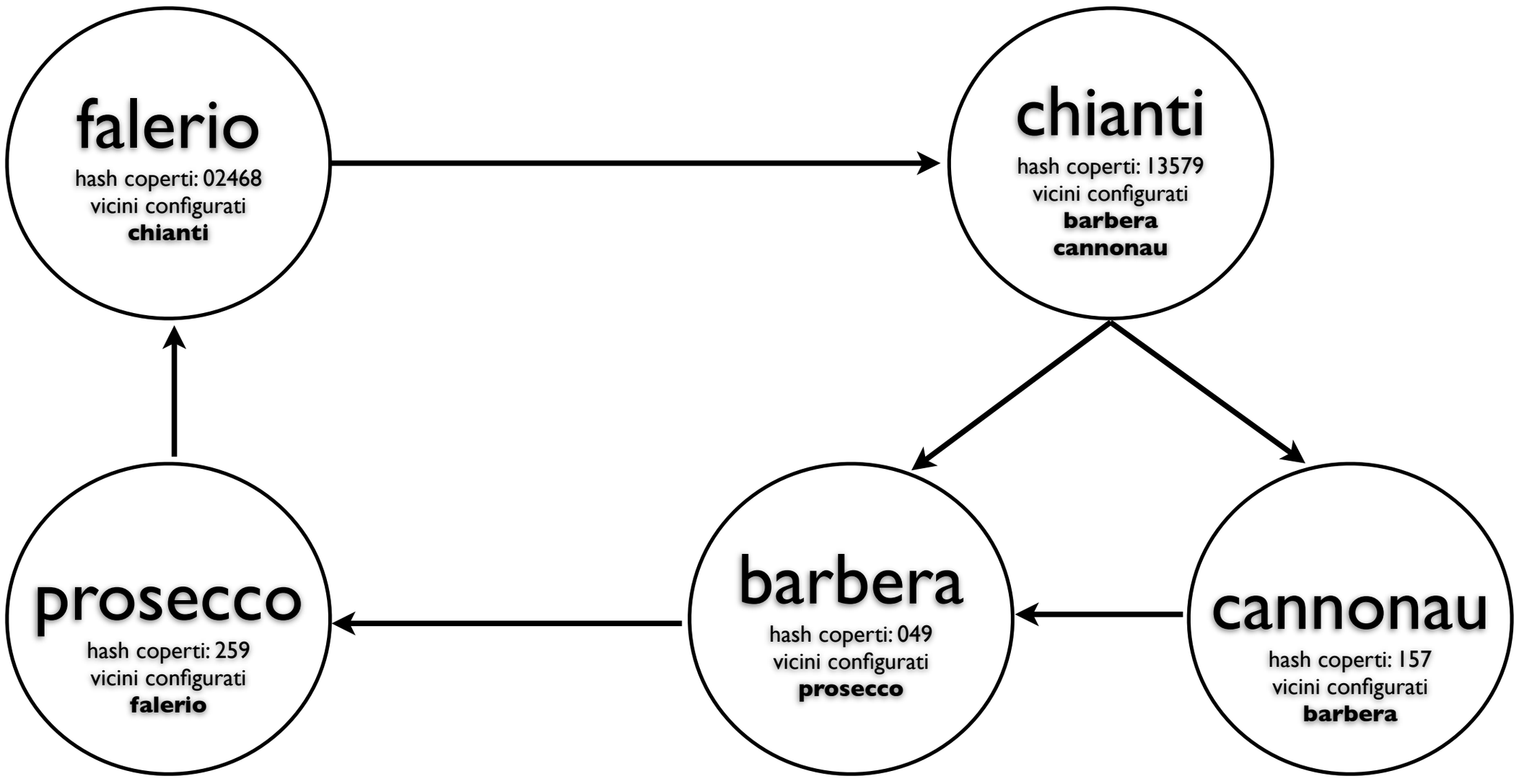
- Dati memorizzati attraverso una struttura ad albero *(possibilità di sfruttare thread multipli che scandiscono l'albero contemporaneamente per selezionare l'output di eventuali get)*
- Interrogazioni attraverso espressioni regolari POSIX
- Open source

Memorizzazione



Più threads potrebbero analizzare i dati all'interno dell'albero contemporaneamente

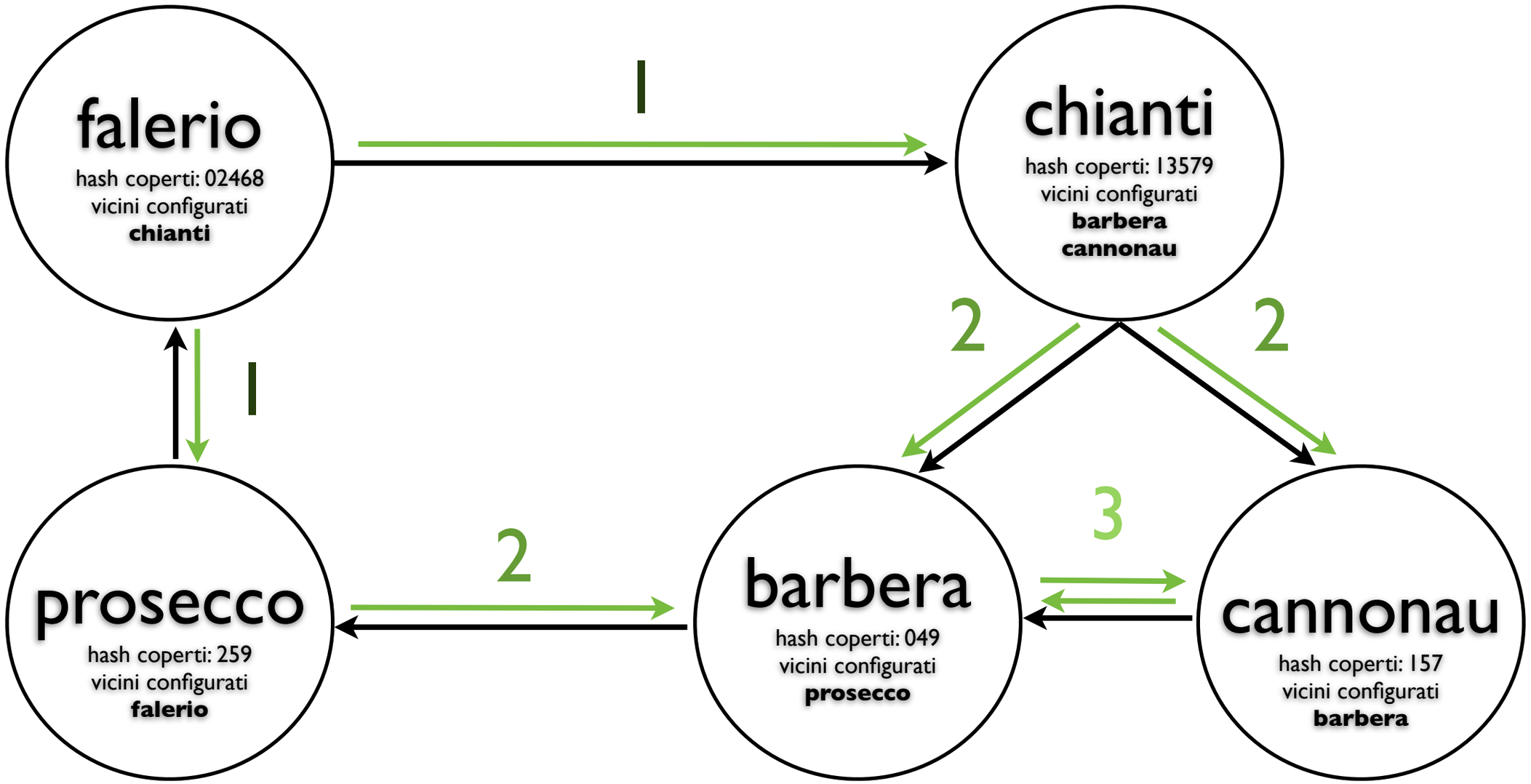
Caso di studio



Algoritmo distribuito: **flooding**

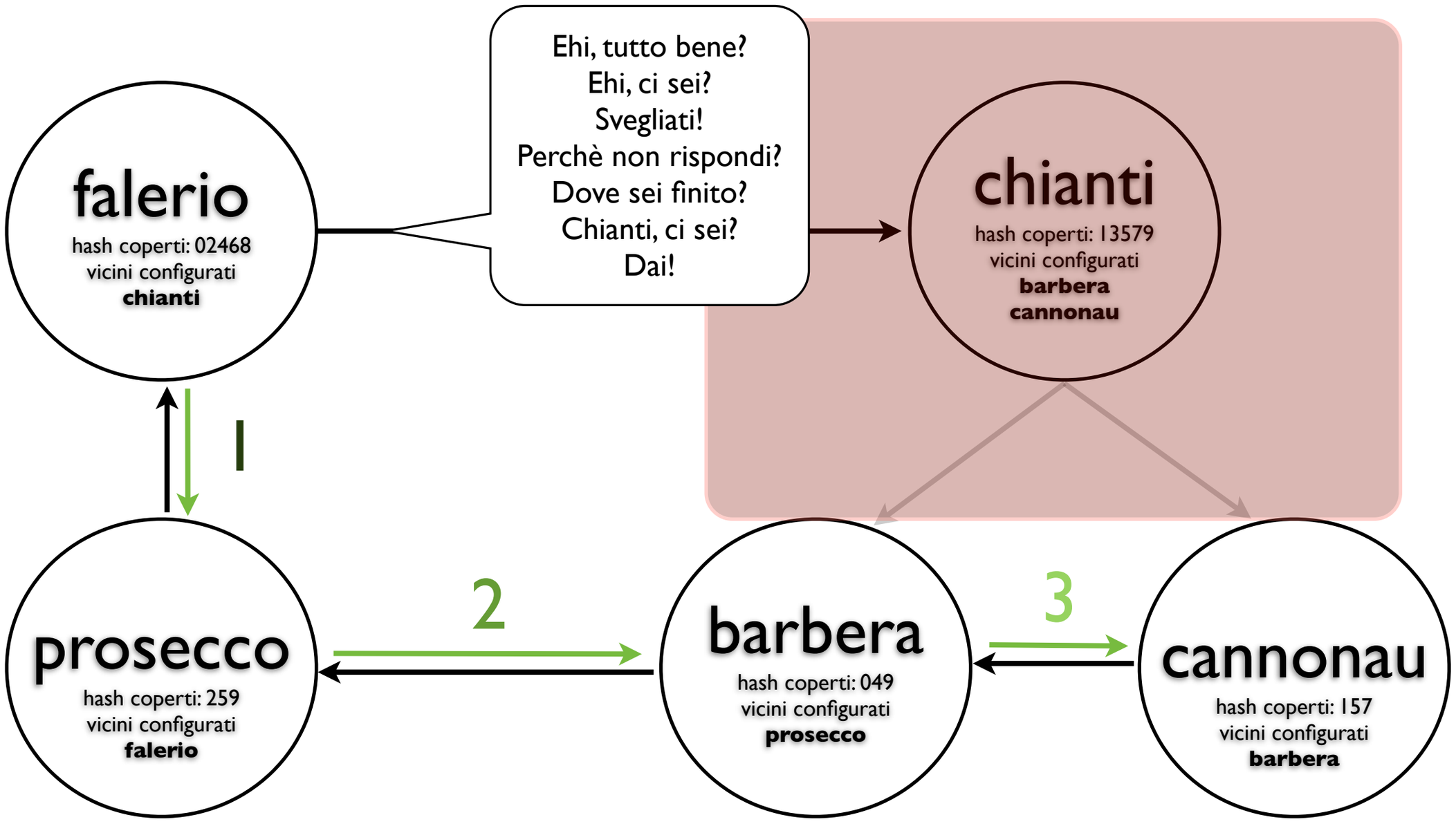
Caso di studio

richiesta di inserimento



Caso di studio

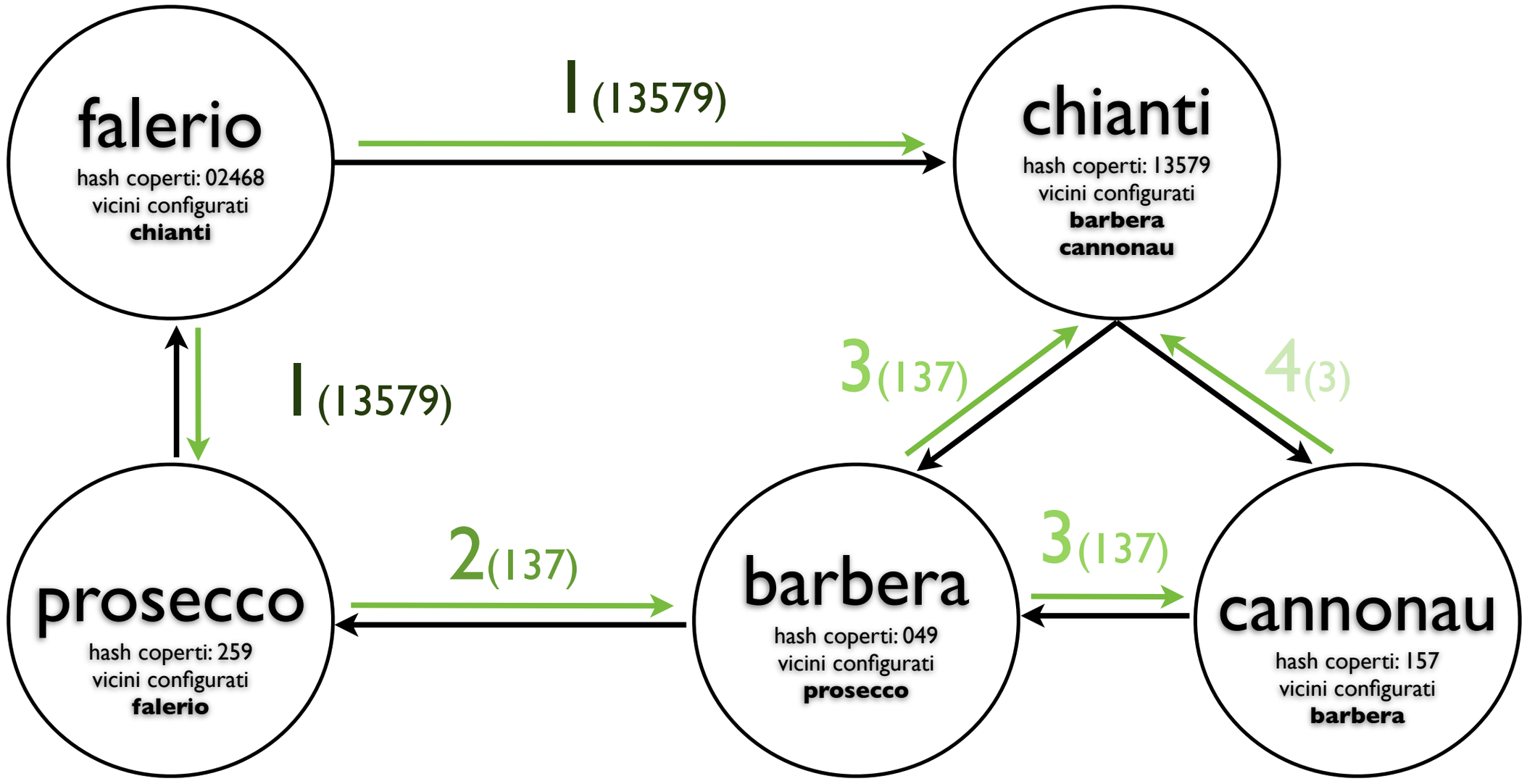
richiesta di inserimento



Caso di studio

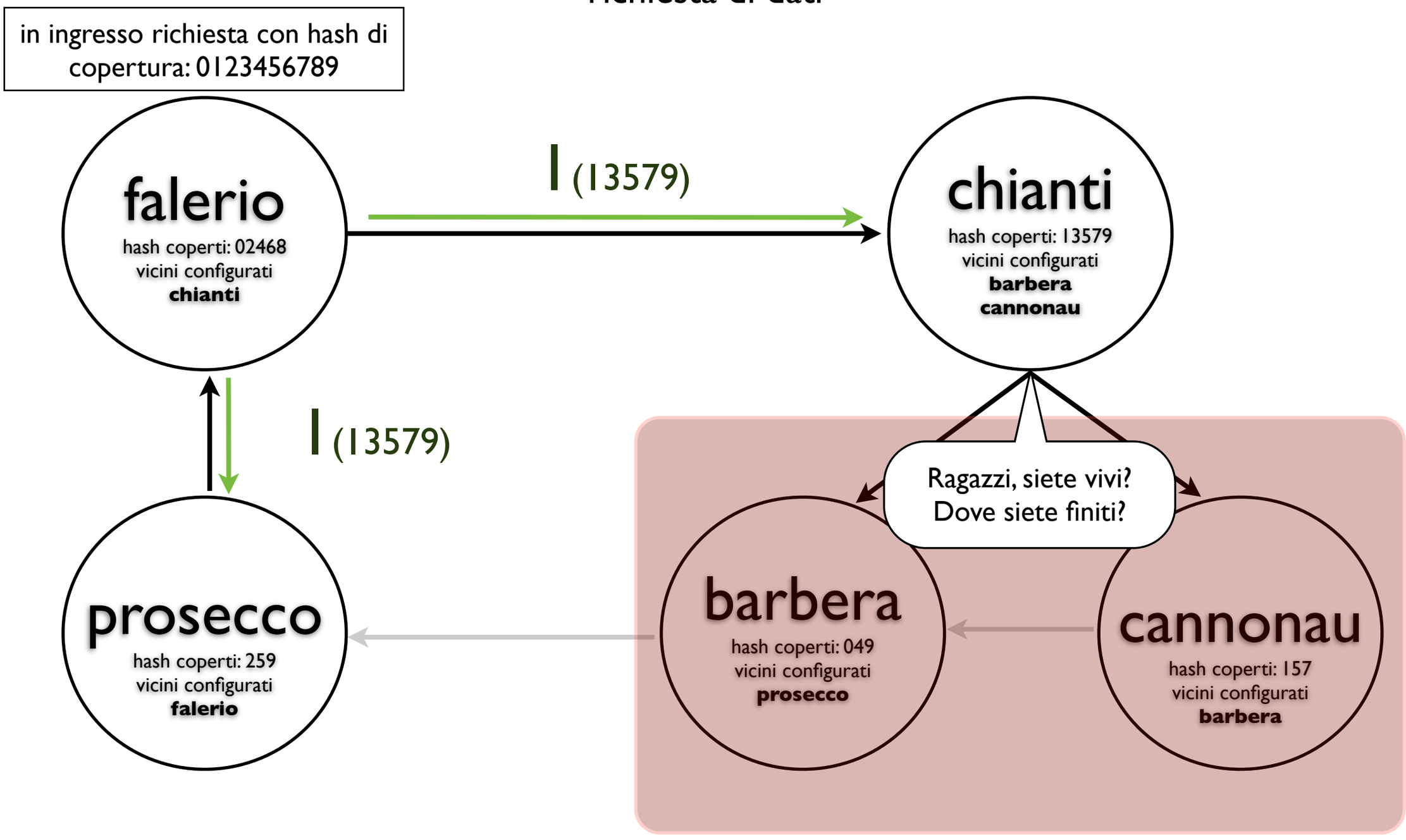
richiesta di dati

in ingresso richiesta con hash di copertura: 0123456789



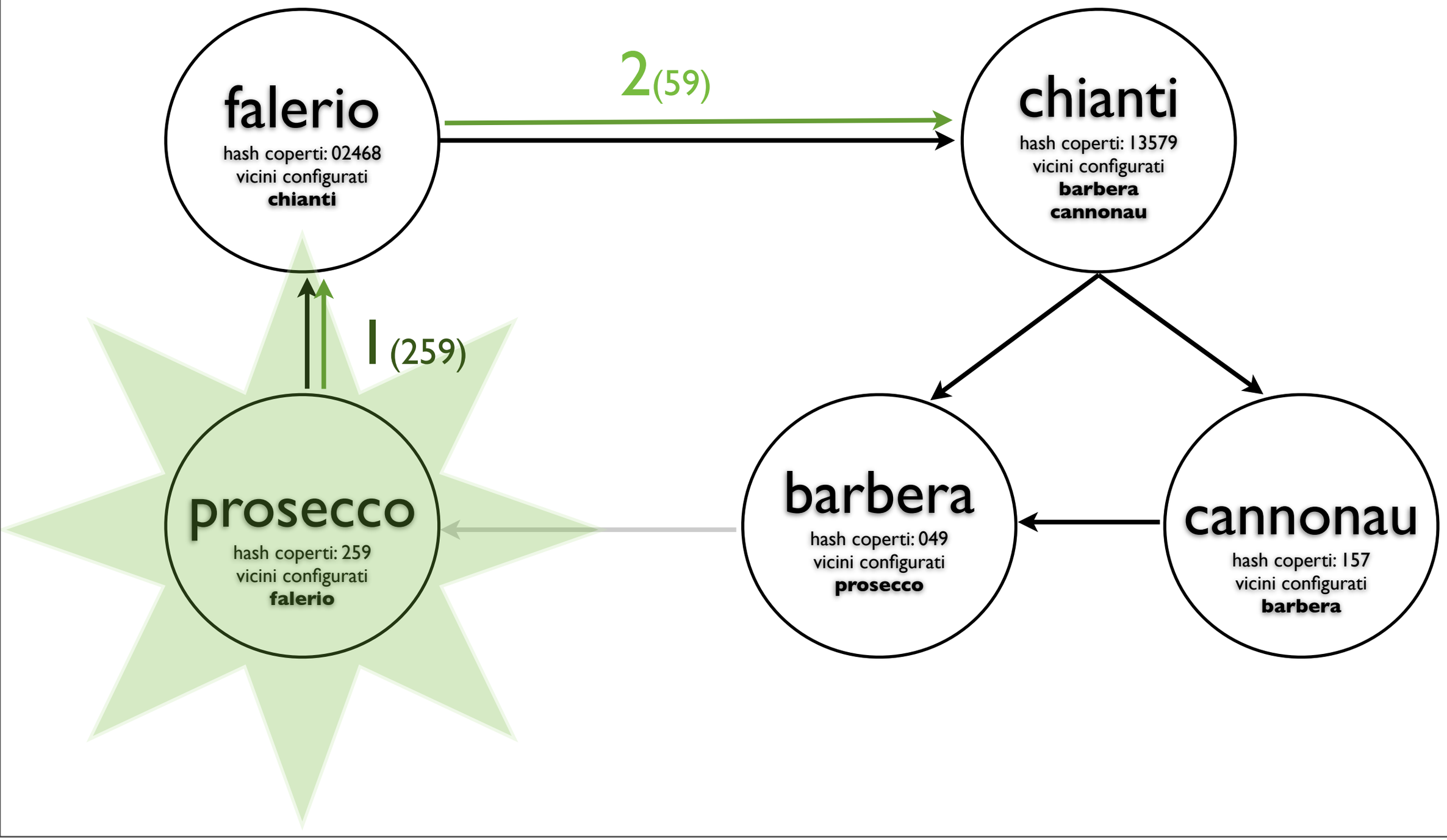
Caso di studio

richiesta di dati



Caso di studio

wake up di un nodo



Implementazioni

pippolapi

- API in C di facile utilizzo con parser XML integrato rapido ed efficiente
- Astrazione totale
- Multipiattaforma (zero dipendenze)
- Facilmente integrabile con qualsiasi progetto

Esempio

inserimento

```c
#include "pippolo_api.h"
void hooker (const char *ID, struct str_xml_node *result) {
    printf("operation %s complete\n", ID);
}

int discretizer (const char *value) {
    int result = 0;
    char *ptr = (char *)value;
    while (*ptr)
        result += *ptr++;
    result = (result%10);
    return result;
}

int main (int argc, char *argv[]) {
    if (argc <= 1)
        return 0;
    struct str_record *records = NULL;
    p_node_pippolo_init("vianello");
    p_node_pippolo_add("127.0.0.1", 5090);
    p_node_pippolo_add("127.0.0.1", 7090);
    p_node_pippolo_add("127.0.0.1", 4090);
    pippolo_discretizer = &discretizer;
    p_node_record_add(&records);
    p_node_record_keys_add(&records, "fiscal", "GRSDNL91R12A662K", pippolo_true);
    p_node_record_keys_add(&records, "name", "Daniele", pippolo_false);
    p_node_record_keys_add(&records, "surname", "Grassi", pippolo_false);
    p_node_action("inserimento unico", EDATA_ACTIONS_ADD, records, 10, &hooker);
    while (pippolo_true); /* waiting */
    return 0;
}
```

note dolenti

Cose da implementare

- Salvataggio della knowledge su files e cacheing per ricerca con algoritmo di previsioning
- Attivare multithreading nella ricerca all'interno dell'albero dei dati
- Rimpiazzare il flooding con algoritmo selezionativo (uso porte di servizio)

Cose da implementare

- Autenticazione
- Comunicazione protetta con crittografia asimmetrica (tramite CA interno)
- Controllo nodo tramite CLI
- Permettere il cambio della configurazione direttamente in *runtime* (aggiunta di nuovi nodi o coperture differenti)