Sistemas Distribuidos

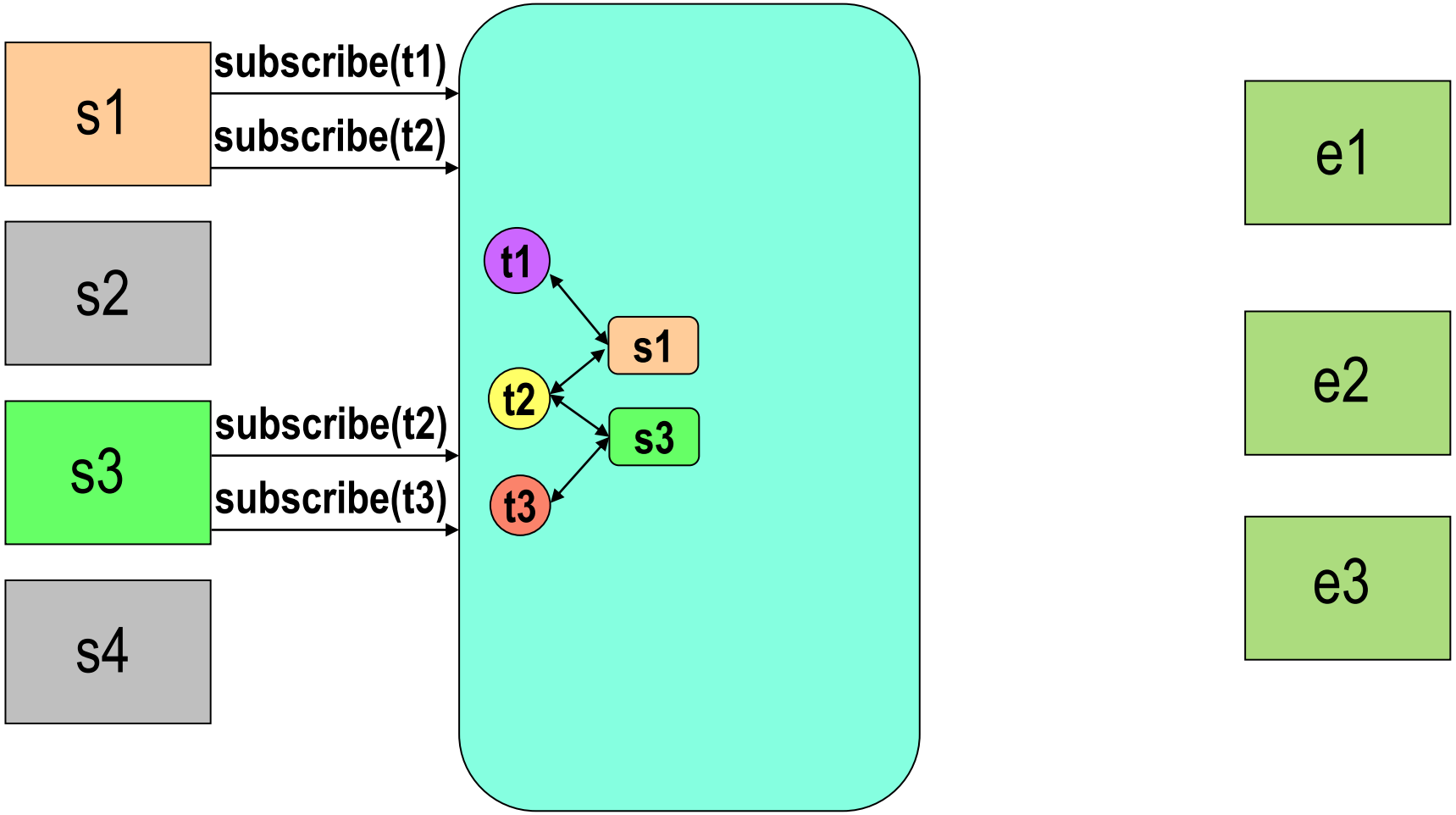
1º práctica individual

EdSuPull

Objetivo

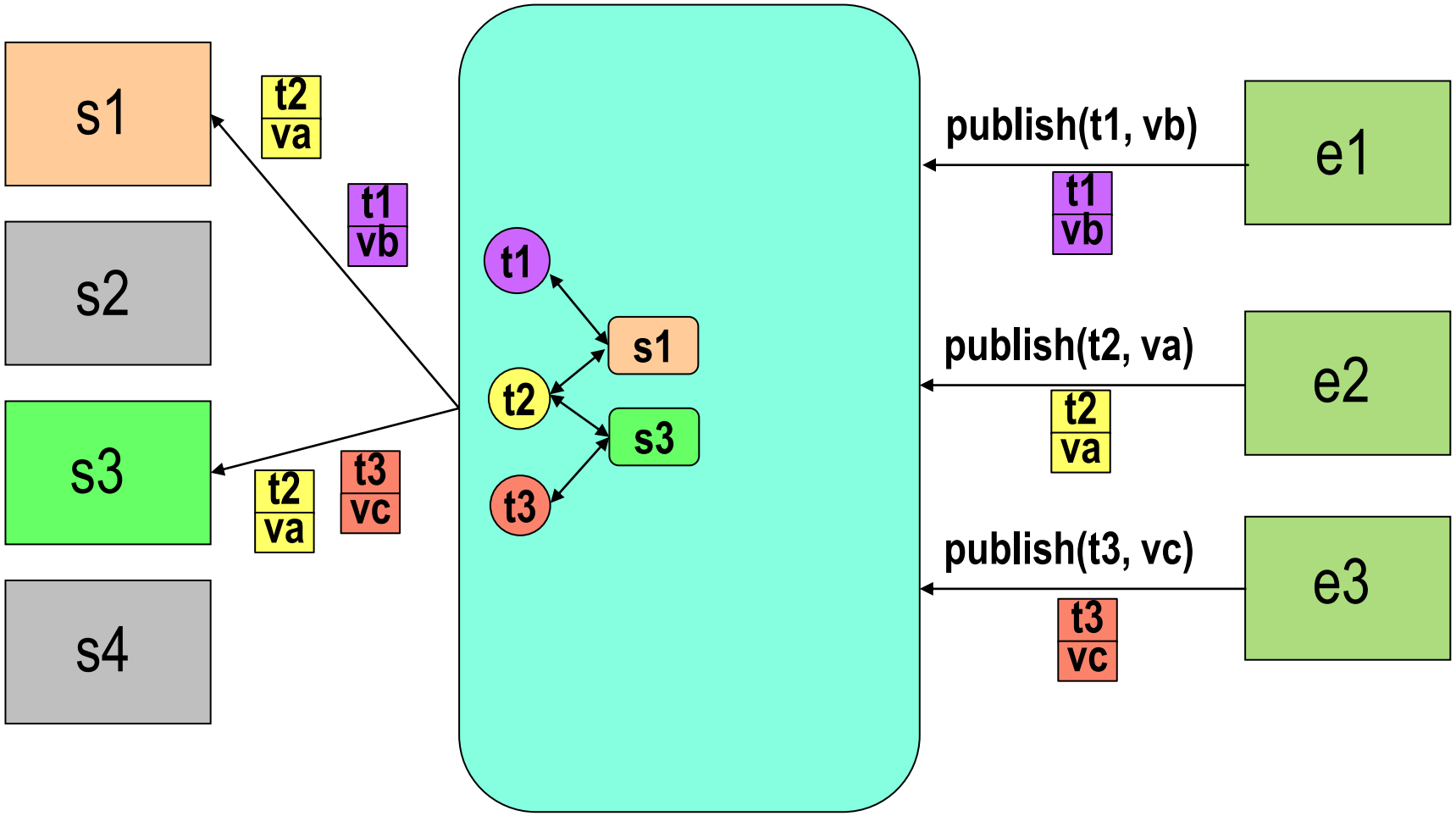
- *EdSu* con esquema *pull*, uso de *broker* y basado en temas
- C y sockets *stream* en entorno heterogéneo
- Servicio concurrente con *threads* dinámicos: 1 por conexión
- Cliente (editor/subscriptor) conexión persistente con *broker*
  - Identificado ante *broker* mediante UUID
- *broker* lee temas de un fichero
- *Evento* = *tema* (*string*; longitud $\leq 2^{32}-1$) y *valor* (puede ser binario)
  - Recuerde asegurar cadenas de caracteres recibidas terminan con nulo
- Uso obligatorio tipos de datos proporcionados: *map*, *set* y *queue*
- No límite en: nº temas, subscriptores y eventos
- *zerocopy*: no copias de nombre de temas ni de eventos
  - Ni múltiples envíos porque producen fragmentación
- Optimizar ancho de banda gastado

Modelo editor/subscriptor (*push*)



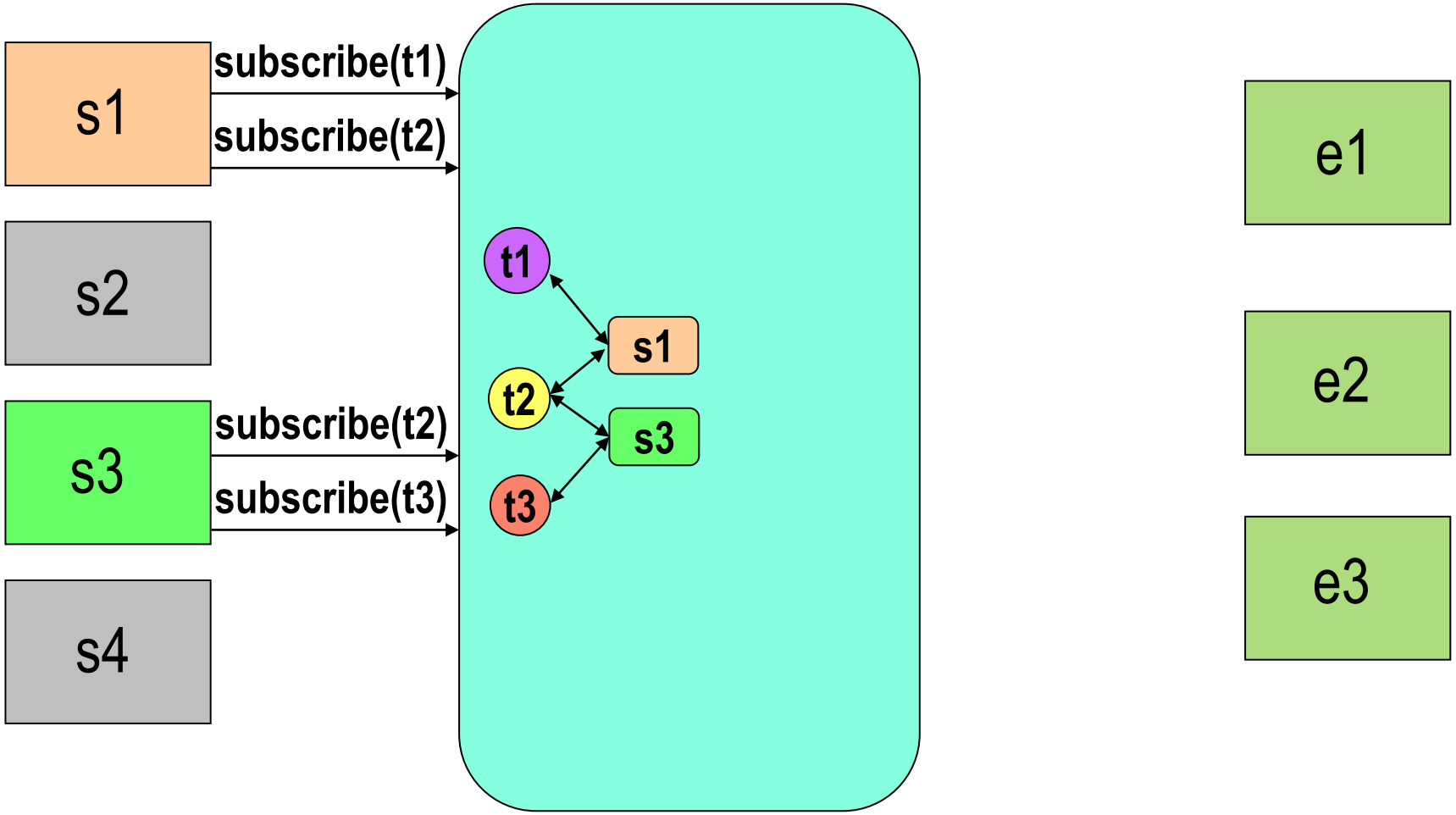
no implementado en la práctica (sí lo fue en cursos pasados)

Modelo editor/subscriptor (*push*)



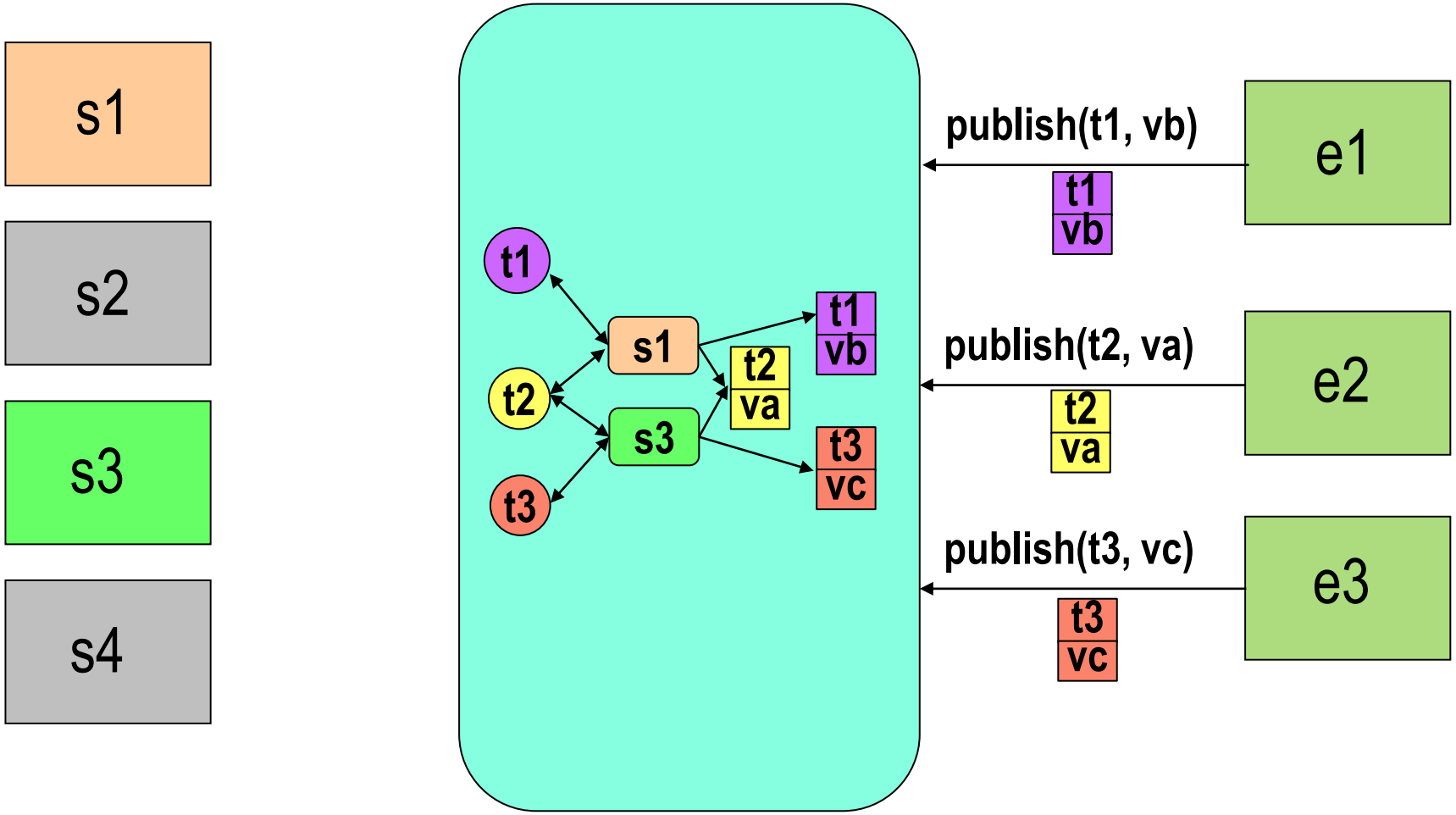
no implementado en la práctica (sí lo fue en cursos pasados)

Modelo editor/subscriptor (*pull*)



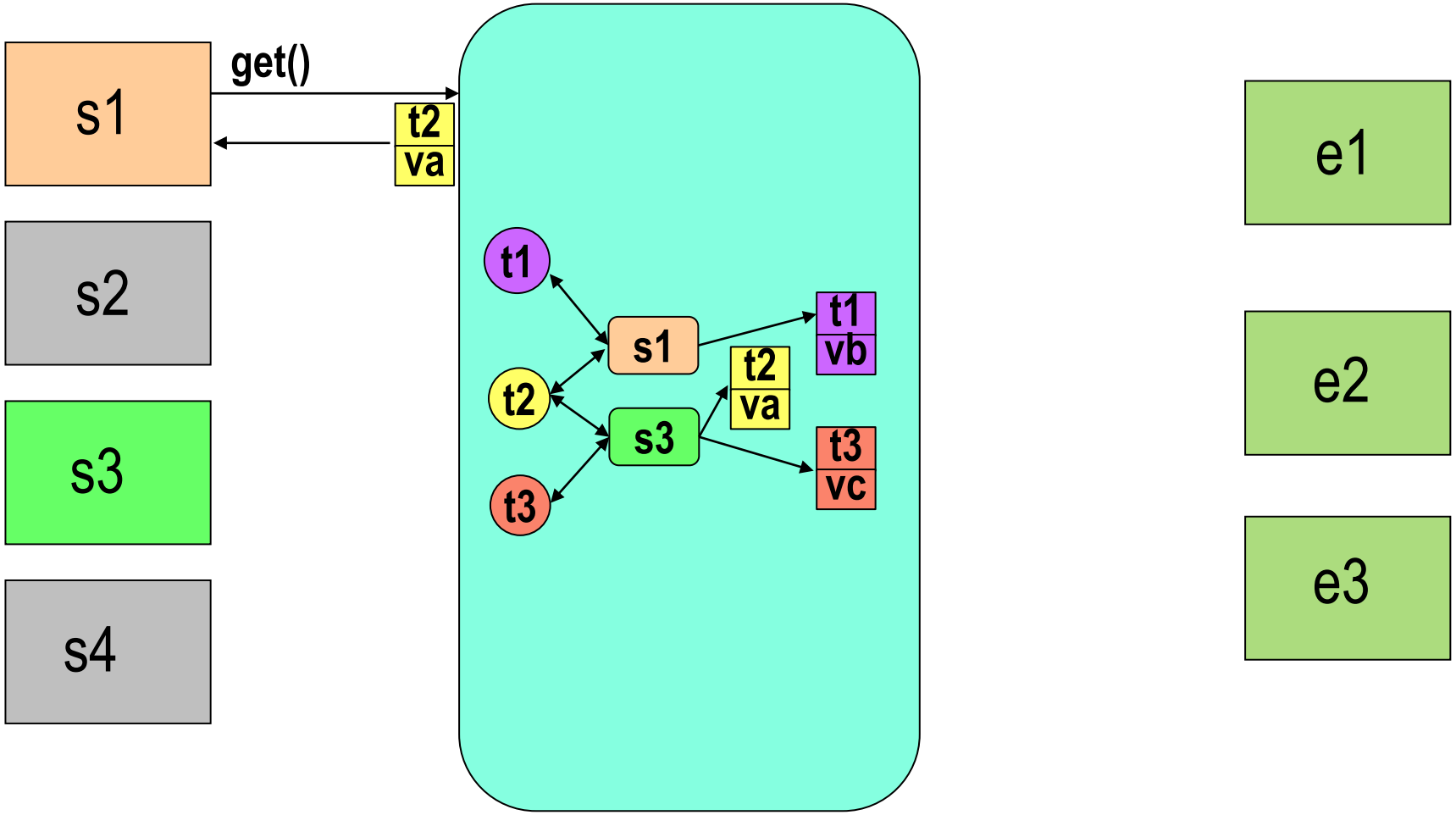
hay que implementarlo en la práctica

Modelo editor/subscriptor (*pull*)



hay que implementarlo en la práctica

Modelo editor/subscriptor (*pull*)



hay que implementarlo en la práctica

API ofrecida a las aplicaciones

int **begin_clnt**(void); // inicio de un cliente
int **end_clnt**(void); // fin del cliente

---

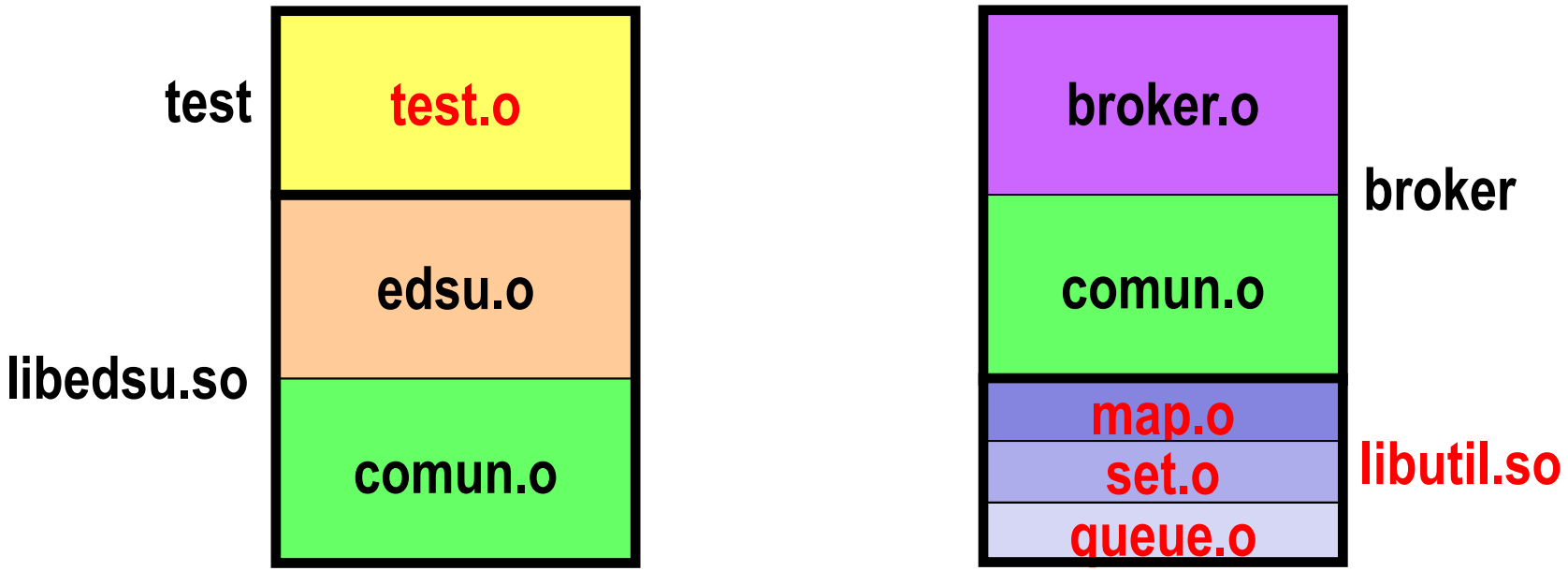
int **subscribe**(const char *tema);
int **unsubscribe**(const char *tema);
int **publish**(const char *tema, const void *evento, uint32_t tam_ev);
int **get**(char **tema, void **evento, uint32_t *tam_evento);

Implementación en *edsu.c*

---

int **topics**(); // cuántos temas existen en el sistema
int **clients**(); // cuántos clientes existen en el sistema
int **subscribers**(const char *tema); // nº subscriptores de ese tema
int **events**(); // nº eventos pendientes de recoger por este cliente

Arquitectura software



Ejecución de pruebas

triqui3: cd broker; make
triqui3: ./broker 12345 ftemas

---

triqui4: cd test; make
triqui4: export BROKER_PORT=12345
triqui4: export BROKER_HOST=triqui3
triqui4: ./test

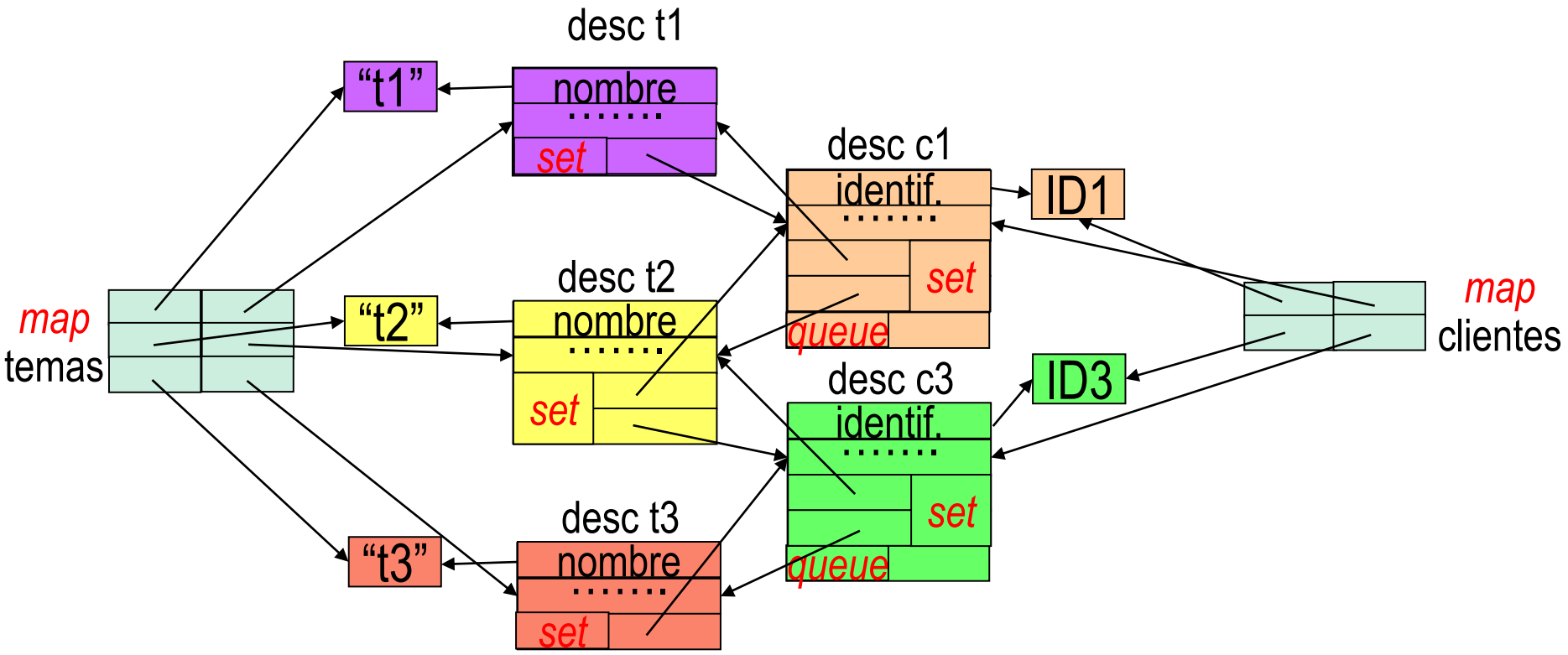
---

triqui2: cd test; make
triqui2: export BROKER_PORT=12345
triqui2: export BROKER_HOST=triqui3
triqui2: ./test

Pautas para el desarrollo de la práctica

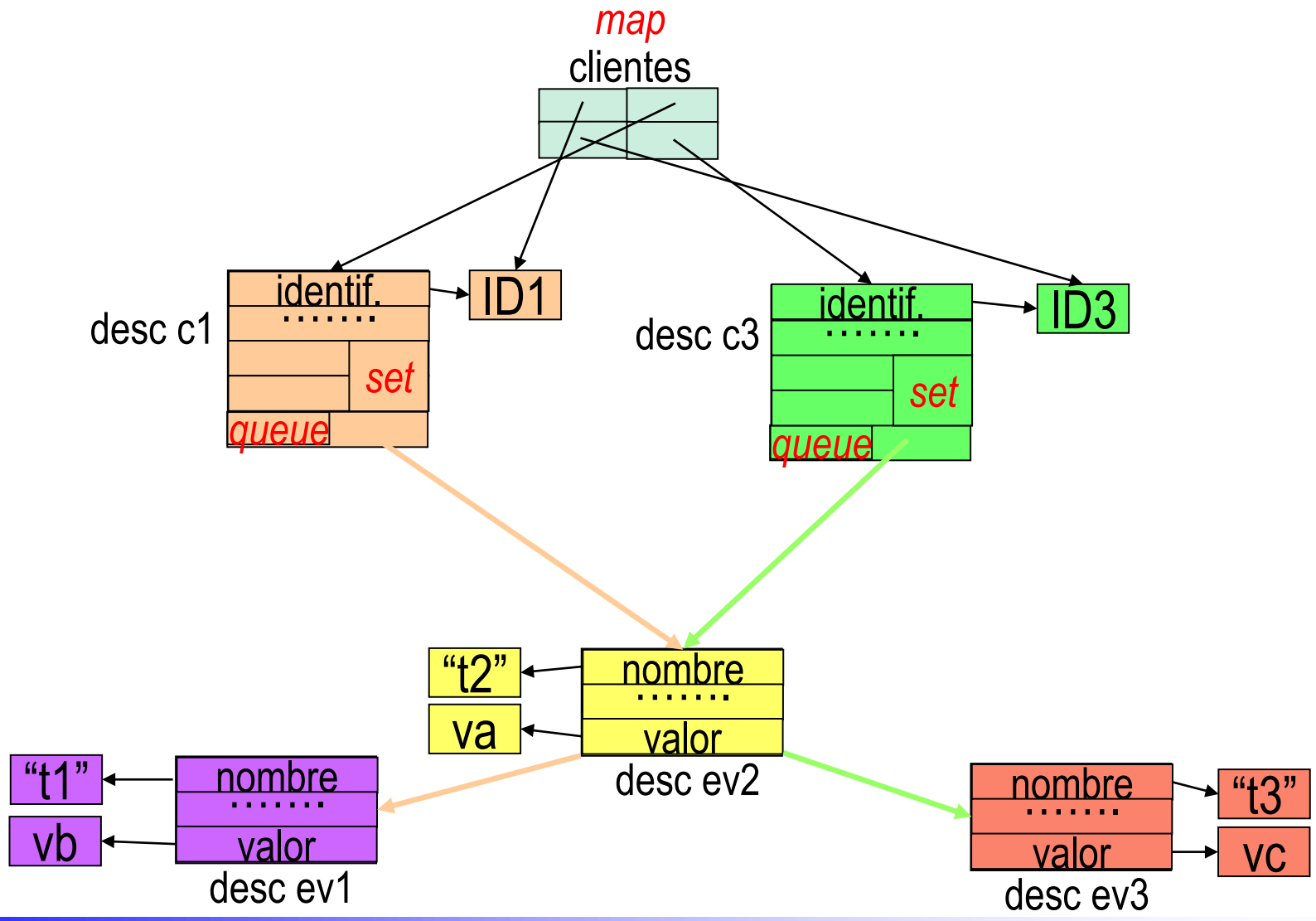
- Gestión de 3 tipos de objetos: temas, clientes y eventos.
  - Recomendación: un descriptor para cada tipo
  - temas y clientes: relación N-M
    - descriptor cliente: ***set*** descriptores de temas a los que está suscrito
    - descriptor tema: ***set*** descriptores de clientes que lo han subscrito
  - ***map*** de clientes: UUID → descriptor de cliente
  - ***map*** de temas: nombre → descriptor de tema
  - Descriptor cliente: ***queue*** de eventos pendientes de recoger
- *demo_integral*: ilustra uso 3 tipos objetos en escenario similar
- Uso de algunos de los ejemplos de sockets estudiados:
  - *broker*: servicio concurrente con *threads* dinámicos (ej. *srv_rev_thr*)
  - Envío de toda la información con una sola operación (*writev*)
    - ejemplos: *cln_rev envio_tam_variable_writev*

Uso de estructuras de datos (sin eventos)



Relación N-M entre temas y clientes

Uso estructuras datos: detalle eventos



Etapas de la práctica

1. Inicio del cliente (1,5 puntos)
2. Lectura de temas (1 puntos)
3. Subscripción (1,5 puntos)
4. Publicación (1,5 puntos)
5. Operación *get* (1,5 puntos)
6. Baja de subscripción (1,5 puntos)
7. Fin del cliente (1,5 puntos)

Trabajo optativo extra (1 punto + sobre nota final, si esta es ≥ 5)

- *get* bloqueante (*long polling*)
- esquema concurrente ⟶ basado en eventos
  - similar a ejemplo *srv_rev_evn* (usa *epoll*).

Etapa 1: Inicio del cliente

- Crea conexión persistente
- *Broker* crea descriptor de cliente identificado por UUID
- Incluye entrada en mapa de clientes
- Operación *clients* facilita depuración y evaluación

Etapa 2: Lectura de temas

- Antes del bucle de servicio
- *Broker* lee fichero de temas
- Incluye entradas en mapa de temas
- Operación *topics* facilita depuración y evaluación

Etapa 3: Subscripción

- Cliente se añade a conjunto de subscriptores del tema
- Tema se añade a conjunto de subscripciones del cliente
- Operación *subscriptors* facilita depuración y evaluación

Etapa 4: Publicación

- Se crea descriptor de evento
- Por cada cliente subscriptor
  - Se añade evento al final de la cola de cada cliente
- Operación *events* facilita depuración y evaluación

Etapa 5: Operación *get*

- Extrae evento del inicio de la cola de ese cliente
- Solo se puede liberar información del evento cuando
  - haya sido entregado a todos los clientes involucrados
- En cliente, *get* reserva memoria dinámica requerida
  - Aplicación debe liberarla

Etapa 6: Baja de subscripción

- Cliente se elimina de conjunto de subscriptores del tema
- Tema se elimina de conjunto de subscripciones del cliente
- No afecta a la cola de eventos

Etapa 7: Fin de cliente

- Se da de baja cliente de todos los temas subscritos
- Se elimina entrada en el mapa de clientes
- Se libera descriptor de cliente:
  - Liberando el conjunto de temas subscritos
  - Descartando los eventos encolados