Exam Exercises

ACTIVE DATABASES

A. Active Databases (9 p.)

A parcel delivery company has distribution centers around the world, connected by flights. Delivery is requested by inserting tuples into table **PARCEL**, annotated with two timestamps: the request time and the delivery deadline (time by which the parcel should be in the destination center).

Scheduled flights have fixed routes and timing (departure and arrival Ids and timestamps), and a capacity that is expressed in number of transportable parcels. A table DISTANCE stores the distances between all centers.

FLIGHT (FlightId, OriginId, DestinId, DepTime, ArrivalTime, TotalCapacity, AvailableCapacity)

PARCEL (ParcelId, OriginId, DestinId, TimeReceived, DeliveryDeadline, FlagOnTime)

PARCELROUTESTEP (ParcelId, FlightId, FlagFinalStep)

DISTANCE (OriginId, DestinId, Km)

Write a trigger system that manages the creation of the route for the parcels, possibly split into several PARCELROUTESTEPs, according to the following "greedy" strategy. The triggers react to the insertion of a new parcel, and assign it to the first flight with available capacity that either

(i) directly reaches the final destination *f* of the parcel, or
(ii) moves it to a center *c* that is not only closer to *f* than the current location *cl*, but is the closest to *f* among those directly reachable from *cl*.

The triggers must also verify that the TimeReceived of the parcel precedes the DepTime of the chosen flight and, when the delivery is not direct but split over multiple steps, the ArrivalTime of each step must precede the DepTime of the next step.

FlagFinalStep is set to *true* when the step reaches the final parcel destination, while FlagOnTime, initially always set to *true*, must state (at the end of the computation for each parcel) whether it will reach the final destination before the deadline or not.

Also, briefly discuss the termination of the trigger system.

```
CREATE TRIGGER NewParcel
AFTER INSERT INTO Parcel
FOR EACH ROW
BEGIN

    DECLARE _Fid int; DECLARE _Dist int;
    SELECT FlightId INTO _Fid
    FROM Flight
    WHERE OriginId = new.OriginId AND DestinId = new.DestinId
    AND DepTime > new.TimeReceived AND AvailableCapacity > 0
    ORDER BY DepTime ASC
    LIMIT 1;

    IF _Fid IS NOT NULL THEN
        INSERT INTO ParcelRouteStep
        VALUES (new.ParcelId, _Fid, TRUE);

        ….
```

```
  ELSE
    SELECT Km INTO _Dist FROM Distance
    WHERE OriginId = new.OriginId AND DestinId = new.DestinId;

    SELECT FlightId INTO _Fid
    FROM Flight AS F JOIN Distance AS D
    ON D.OriginId = F.DestinId AND D.DestinId = new.DestinId
    WHERE F.OriginId = new.OriginId AND F.DepTime > new.TimeReceived
    AND F.AvailableCapacity > 0 AND D.Km < _Dist
    ORDER BY D.Km ASC
    LIMIT 1;

  IF (_Fid IS NULL) THEN
    SELECT RAISE(ABORT, "no route to destination");
  ELSE
    INSERT INTO ParcelRouteStep
    VALUES (new.ParcelId, _Fid, FALSE);
  END IF;

END IF;
END;
```

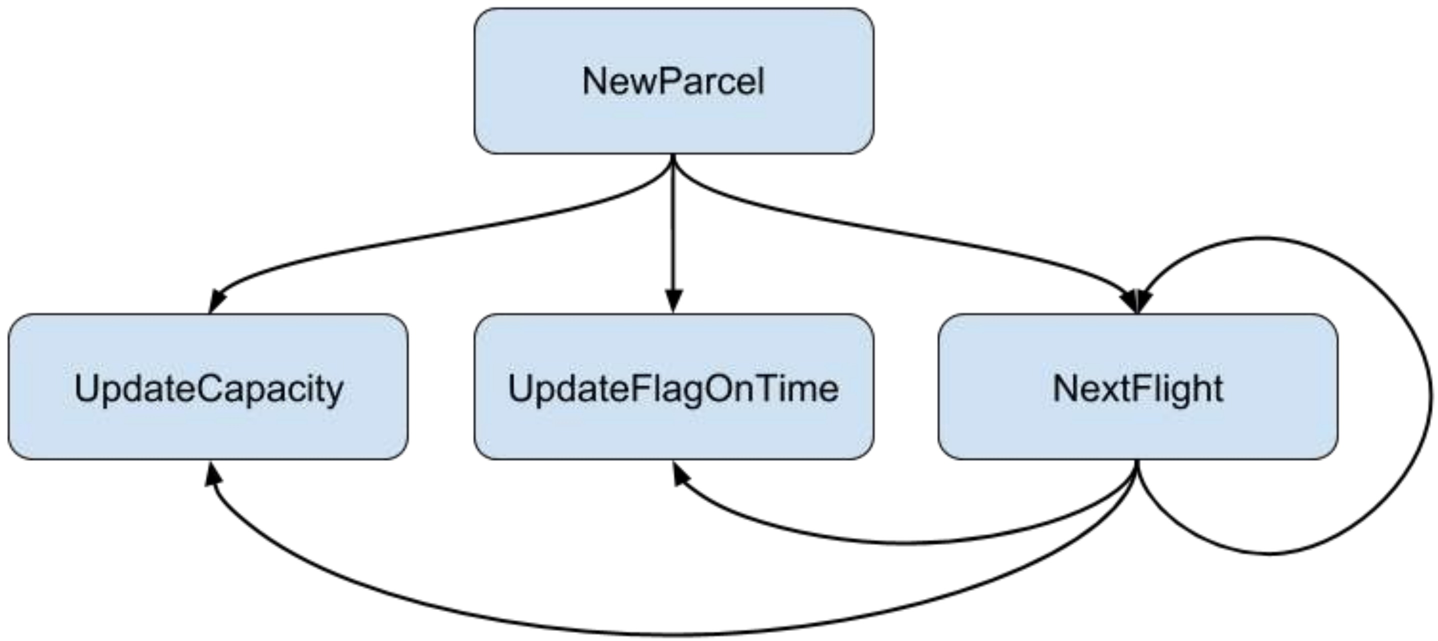
```sql
CREATE TRIGGER UpdateCapacity
AFTER INSERT INTO ParcelRouteStep
FOR EACH ROW
BEGIN

UPDATE Flight
    SET AvailableCapacity = AvailableCapacity - 1
    WHERE FlightId = new.FlightId;

END;
```

```
CREATE TRIGGER UpdateFlagOnTime
AFTER INSERT INTO ParcelRouteStep
FOR EACH ROW
WHEN new.FlagFinalStep = TRUE
BEGIN

  UPDATE Parcel
  SET FlagOnTime = (
    SELECT DeliveryDeadline > (
      SELECT ArrivalTime
      FROM Flight
      WHERE FlightId = new.FlightId
    )
  )
  WHERE ParcelId = new.ParcelId;

END;
```

```
CREATE TRIGGER NextFlight
AFTER INSERT INTO ParcelRouteStep
WHEN new.FlagFinalStep = FALSE
FOR EACH ROW
BEGIN
    DECLARE _Fid int;
    DECLARE _Dist int;
    DECLARE _OriginId int;
    DECLARE _DestinId int;
    DECLARE _ArrivalTime int;

    SELECT DestinId AS _OriginId, ArrivalTime AS _ArrivalTime
    FROM Flight WHERE FlightId = new.FlightId;

    SELECT DestinId AS _DestinId
    FROM Parcel WHERE ParcelId = new. ParcelId ;

   -- from here it's the same as the first trigger, with adjusted variables
      (_OriginId instead of new.OriginId)
```

```
  SELECT FlightId INTO _Fid
  WHERE OriginId = _OriginId AND DestinId = _DestinId
  AND DepTime > _ArrivalTime AND AvailableCapacity > 0
  ORDER BY DepTime ASC LIMIT 1;

  IF _Fid IS NOT NULL THEN
    INSERT INTO ParcelRouteStep VALUES (new.ParcelId, _Fid, TRUE);
  ELSE
    SELECT Km INTO _Dist FROM Distance
    WHERE OriginId = _OriginId AND DestinId = _DestinId;

    SELECT FlightId INTO _Fid FROM Flight AS F JOIN Distance AS D
    ON D.OriginId = F.DestinId AND D.DestinId = _DestinId
    WHERE F.OriginId = _OriginId AND F.DepTime > _ArrivalTime
    AND F.AvailableCapacity > 0 AND D.Km < _Dist
    ORDER BY D.Km ASC LIMIT 1;

    IF _Fid IS NULL THEN
      SELECT RAISE(ABORT, "no route to destination");
    ELSE
      INSERT INTO ParcelRouteStep VALUES (new.ParcelId, _Fid, FALSE);
    END IF;

  END IF;
END;
```

Triggering Graph



There might be a cyclic activation of the *NextFlight* trigger.
Nevertheless, since we always move closer to the destination of the parcel, the termination is guaranteed (if the distances are finite).

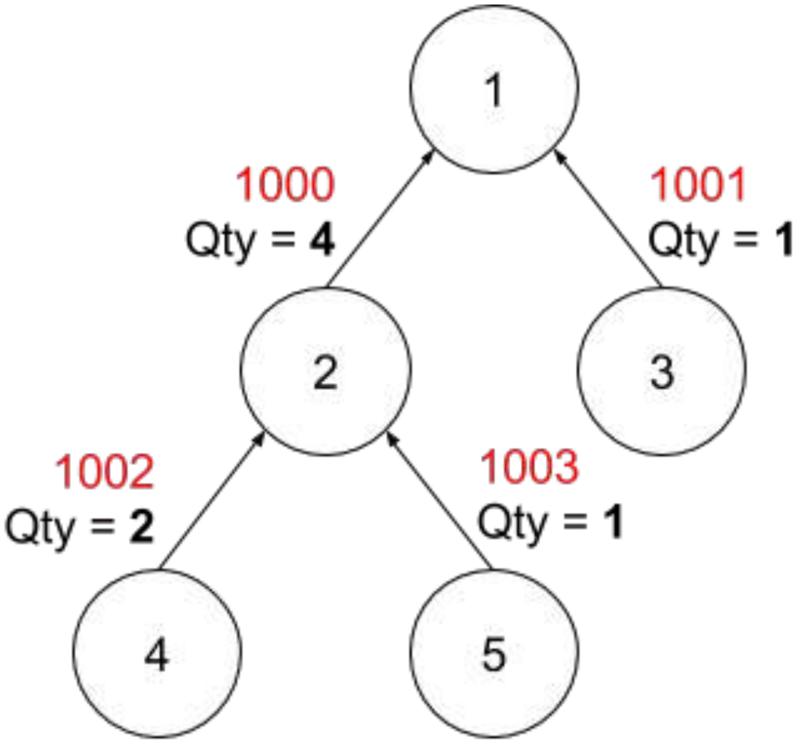
SELECT FlightId **INTO** _Fid … **WHERE** … D.Km < _Dist

Orders (11 / 02 / 2016)

ClientOrder (<u>OrderId</u>, ProductId, Qty, ClientId, TotalSubItems)
ProductionProcess (<u>ProdProcId</u>, ObtainedProdId, StartingProdId, Qty, ProcessDuration, ProductionCost)
ProductionPlan (<u>BatchId</u>, ProdProcId, Qty, OrderId)
PurchaseOrder (<u>PurchaseId</u>, ProdId, Qty, OrderId)

The relational database above supports the production systems of a factory. Table *ProductionProcess* describes how a product can be obtained by (<u>possibly several</u>) other products, which can be themselves obtained from other products or bought from outside.
Build a trigger system that reacts to the <u>insertion of orders</u> from clients and creates new items in *ProductionPlan* or in *PurchaseOrder*, depending on the ordered product, so as to manage the client's order (for the generation of the identifiers, use a function GenerateId()).
The triggers should also update the value of TotalSubItems (initially always set to 0) to describe the number of sub-products (internally produced or outsourced) that are used overall in the production plan deriving from the order.
Also briefly discuss the termination of the trigger system.

sqliteonline: https://goo.gl/Mw4rYB



ProdProc Id	Obtained ProdId	Starting ProdId	Qty
1000	1	2	4
1001	1	3	1
1002	2	4	2
1003	2	5	1

sqliteonline: https://goo.gl/QiOS01

We have to define at least the following triggers:

- **T1 (NewOrder)** reacts to the insertion on ClientOrder and:
  - Adds a record in ProductionPlan if there is a process to build ProductId
  - Adds a record in PurchaseOrder if there is no process to build ProductId

- **T2 (UpdateSubItemsAfterPurchase)** reacts to insertion on PurchaseOrder
  - Sum the ordered Qty to the TotalSubItems of the order

- **T3 (UpdateSubItemsAfterProduction)** reacts to insertion on ProductionPlan
  - Sums the produced Qty to the TotalSubItems of the order

- **T4 (InsertSubProducts)** reacts to insertion on ProductionPlan
  - Adds a record in ProductionPlan if there is a process to build **StartingProdId**
  - Adds a record in PurchaseOrder if there is no process to build **StartingProdId**

- **T1 (NewOrder)** reacts to the insertion on ClientOrder

```
CREATE TRIGGER NewOrder
AFTER INSERT ON ClientOrder
FOR EACH ROW
BEGIN
        IF (EXISTS (SELECT * FROM ProductionProcess
                        WHERE ObtainedProdId = new.ProductId))

                INSERT INTO ProductionPlan
                SELECT GenerateId(), ProdProcId, Qty * new.Qty, new.OrderId
                FROM ProductionProcess
                WHERE ObtainedProdId = new.ProductId;

        ELSE
                INSERT INTO PurchaseOrder VALUES
                (GenerateId(), new.ProductId, new.Qty, new.OrderId);

        END;
END;
```

sqliteonline: https://goo.gl/9QGmtp

- **T1 considerations**:

  - When **new**.ProductId is the ObtainedProdId of a ProductionProcess, we need to insert the records in ProductionPlan to transform its starting products into the obtained product;

  - When **new**.ProductId **isn't** an ObtainedProdId of any ProductionProcess, we need to purchase the ProductId (we are actually re-selling);

  - The production quantity of each Starting Product is **new**.Qty (the number of **new**.ProductId items to produce for the order) * Qty (the number of Starting Products needed to produce one Obtained Product).

- **T2 (UpdateSubItemsAfterPurchase)** reacts to insertion on PurchaseOrder

```
CREATE TRIGGER UpdateSubItemsAfterPurchase
AFTER INSERT ON PurchaseOrder
FOR EACH ROW
BEGIN

        UPDATE ClientOrder
        SET TotalSubItems = TotalSubItems + new.Qty
        WHERE OrderId = new.OrderId;

END;
```

sqliteonline: https://goo.gl/JXiSXC

- **T3 (UpdateSubItemsAfterProduction)** reacts to insertion on ProductionPlan

```
CREATE TRIGGER UpdateSubItemsAfterProduction
AFTER INSERT ON ProductionPlan
FOR EACH ROW
BEGIN

        UPDATE ClientOrder
        SET TotalSubItems = TotalSubItems + new.Qty
        WHERE OrderId = new.OrderId;

END;
```

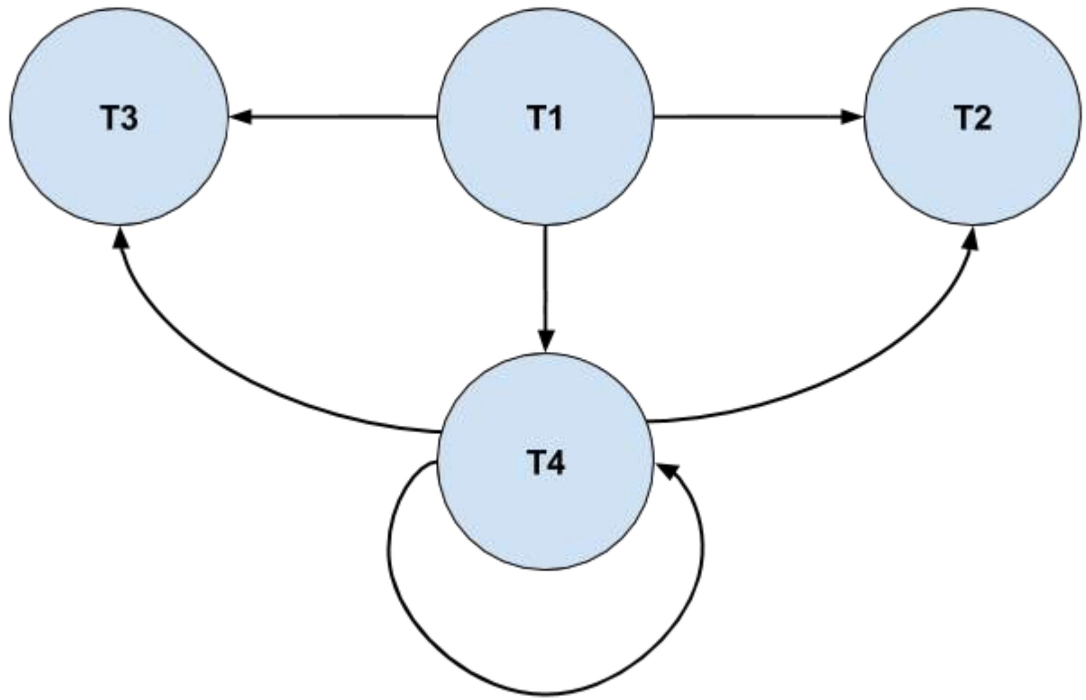
sqliteonline: https://goo.gl/PKyDlJ

- **T4 (InsertSubProducts)** reacts to insertion on ProductionPlan

```
CREATE TRIGGER InsertSubProducts
AFTER INSERT ON ProductionPlan
FOR EACH ROW
BEGIN
        DEFINE S;
        SELECT StartingProdId INTO S
        FROM ProductionProcess WHERE ProdProcId = new.ProdProcId;

        IF (EXISTS (SELECT * FROM ProductionProcess
                    WHERE ObtainedProdId = S))

                INSERT INTO ProductionPlan
                SELECT GenerateId(), ProdProcId, new.Qty * Qty, new.OrderId
                FROM ProductionProcess WHERE ObtainedProdId = S;
        ELSE
                INSERT INTO PurchaseOrder VALUES
                (GenerateId(), S, new.Qty, new.OrderId);
        END;
END;
```

sqliteonline: https://goo.gl/ifrAJO

Termination of the trigger system



- T4 is the only trigger that could be non-terminating
- Nevertheless, if the product hierarchy is well-formed (no cycles), T4 will eventually terminate reaching the leaves.

We can define other (optional and not required) triggers to improve the system:

- **T5 (Validate Order)**
  - Validates TotalSubItems = 0
  - Validates Qty > 0

- **T6 (Delete Order)**
  - Delete all associated PurchaseOrders
  - Delete all associated ProductionPlans

- **T7 (Disable Order Updates)**
  - Permit updates on TotalSubItems
  - Disable updates on other fields

sqliteonline: https://goo.gl/JwhKT2

- **T5 (Validate Order)**

```
CREATE TRIGGER NewOrder_validate
BEFORE INSERT ON ClientOrder
FOR EACH ROW
WHEN ((new.TotalSubItems <> 0) OR (new.Qty <= 0))
BEGIN

        SELECT RAISE(ABORT, "Invalid Order");

END
```

sqliteonline: https://goo.gl/JwhKT2

- **T6 (Delete Order)**

```
CREATE TRIGGER DeleteOrder
AFTER DELETE ON ClientOrder
FOR EACH ROW
BEGIN

        DELETE FROM ProductionPlan
        WHERE OrderId = old.OrderId;

        DELETE FROM PurchaseOrder
        WHERE OrderId = old.OrderId;

END;
```

sqliteonline: https://goo.gl/JwhKT2

- **T7 (Disable Order Updates)**

```
CREATE TRIGGER DisableOrderUpdates
BEFORE UPDATE OF OrderId, ProductId, Qty, ClientId ON ClientOrder
FOR EACH ROW
BEGIN

        SELECT RAISE(ABORT, "Updates on ClientOrder are disabled");

END;
```

sqliteonline: https://goo.gl/JwhKT2

E.1 Siano date le seguenti condizioni di attesa sui nodi di un DBMS distribuito:

Nodo 1: $E_4 \rightarrow t_1$, $t_1 \rightarrow t_2$, $t_2 \rightarrow E_2$

Nodo 2: $E_1 \rightarrow t_2$, $t_2 \rightarrow t_4$, $t_4 \rightarrow E_3$

Nodo 3: $E_2 \rightarrow t_4$, $t_4 \rightarrow t_3$, $t_3 \rightarrow E_4$

Nodo 4: $E_3 \rightarrow t_3$, $t_3 \rightarrow t_1$, $t_1 \rightarrow E_1$

Determinare se si è in presenza di una situazione di blocco critico.

$t_i \rightarrow t_j$

indica un'attesa locale

(t_i attende il rilascio di una ***risorsa*** acquisita da t_j)

$t_i \rightarrow E_n$

indica che t_i attende la ***terminazione*** dell'esecuzione di una sottotransazione $t_?$ sul *nodo* n

(invocazione *sincrona*)

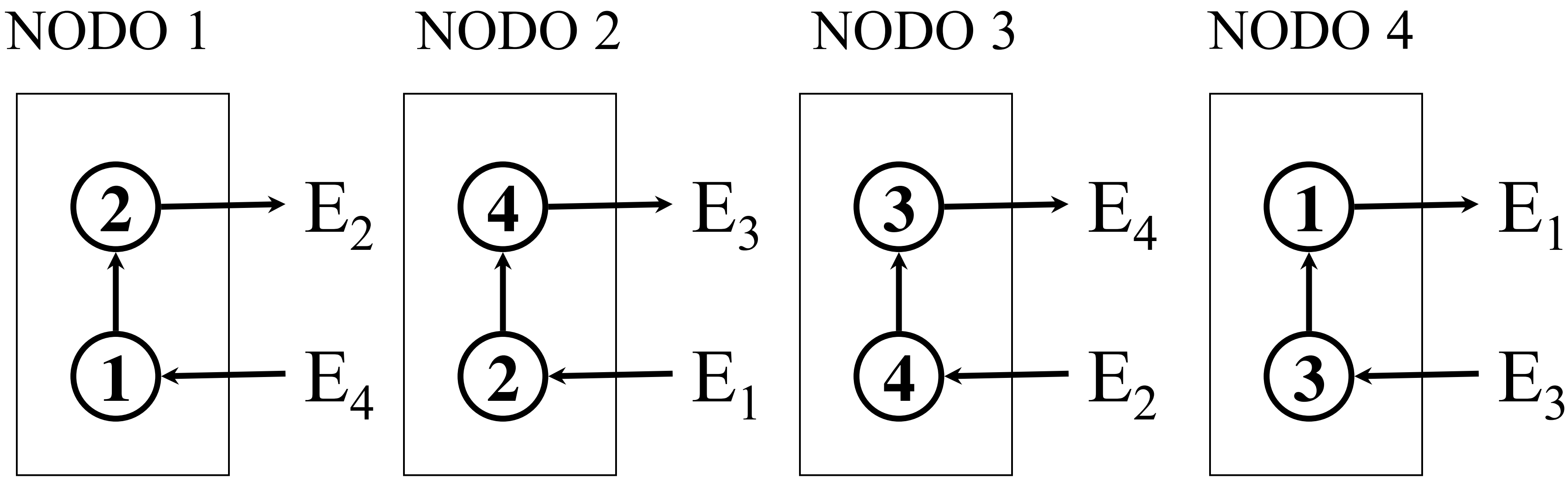
$E_m \rightarrow t_i$

indica che t_i è stata invocata in modo sincrono da una $t_?$ residente sul nodo m

Ogni condizione di attesa in cui una sotto-transazione $\mathbf{t_i}$, attivata in remoto da un nodo m, **attende** (*anche transitivamente* a causa della situazione dei lock) un'altra transazione $\mathbf{t_j}$, che a sua volta attende una sotto-transazione remota su un nodo n, è espressa da:

$$E_m \rightarrow t_i \rightarrow t_j \rightarrow E_n$$

Tale è anche la forma del messaggio che si scambiano i nodi del sistema



Nodo 1: $E_4 \rightarrow t_1$, $t_1 \rightarrow t_2$, $t_2 \rightarrow E_2$

Nodo 2: $E_1 \rightarrow t_2$, $t_2 \rightarrow t_4$, $t_4 \rightarrow E_3$

Nodo 3: $E_2 \rightarrow t_4$, $t_4 \rightarrow t_3$, $t_3 \rightarrow E_4$

Nodo 4: $E_3 \rightarrow t_3$, $t_3 \rightarrow t_1$, $t_1 \rightarrow E_1$

L’algoritmo è **distribuito**.
Ogni istanza (in esecuzione su un nodo) comunica ad altre istanze dello stesso algoritmo le sequenze di attesa:

$$E_m \rightarrow t_i \rightarrow t_j \rightarrow E_n$$

I messaggi sono inviati solo “ in avanti”, cioè verso i nodi dove è attiva la sotto-transazione ***attesa*** da t_i, e viene inviato solo se i>j (*puramente convenzionale*)

Per rispondere *simuliamo l'esecuzione* - **asincrona** e **distribuita** - dell'algoritmo di rilevazione dei deadlock. Ogni nodo decide di inviare le condizioni di attesa (da esterno su esterno) che rileva, in base alla convenzione per cui la generica condizione

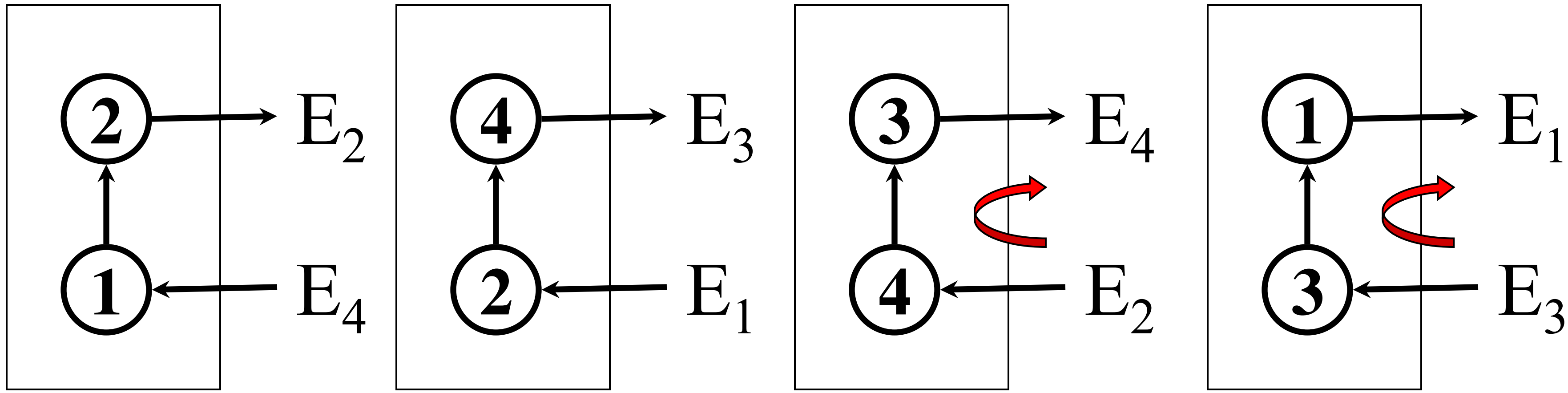
$$E_m \rightarrow t_i \rightarrow t_m \rightarrow t_n \rightarrow t_p \rightarrow t_j \rightarrow E_n$$

si traduce nel messaggio

$$E_m \rightarrow t_i \rightarrow t_j \rightarrow E_n$$

da inviare al nodo **n**, e soltanto se **i > j**

NODO 1 NODO 2 NODO 3 NODO 4

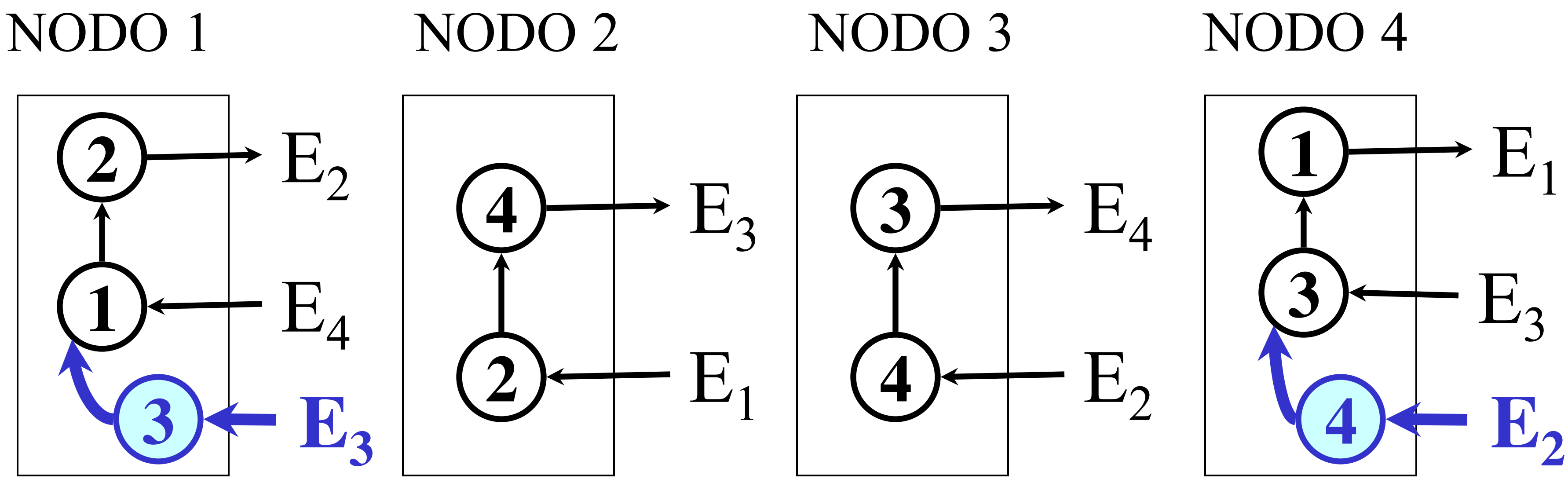


Nulla da inviare

Nulla da inviare

$E_2 \rightarrow t_4 \rightarrow t_3 \rightarrow E_4$ da inviare al nodo 4

$E_3 \rightarrow t_3 \rightarrow t_1 \rightarrow E_1$ da inviare al nodo 1



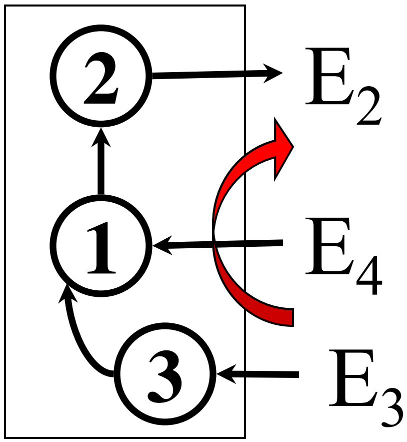
RICEVUTO

$E_3 \rightarrow t_3 \rightarrow t_1 \rightarrow E_1$

RICEVUTO

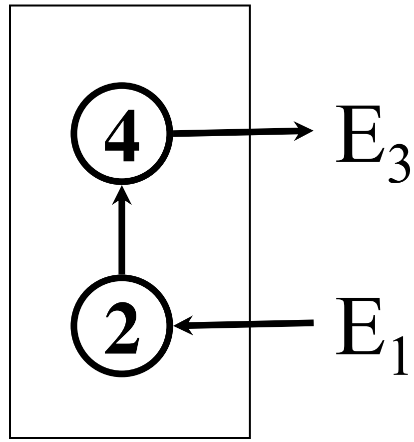
$E_2 \rightarrow t_4 \rightarrow t_3 \rightarrow E_4$

NODO 1



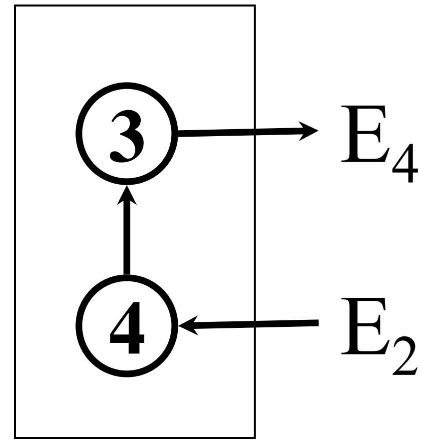
$E_3 \rightarrow t_3 \rightarrow t_2 \rightarrow E_2$ da inviare al nodo 2

NODO 2



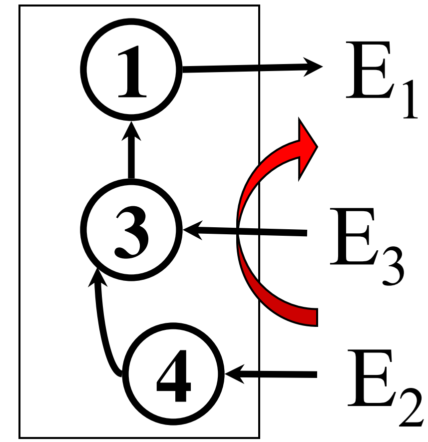
Nulla da inviare

NODO 3

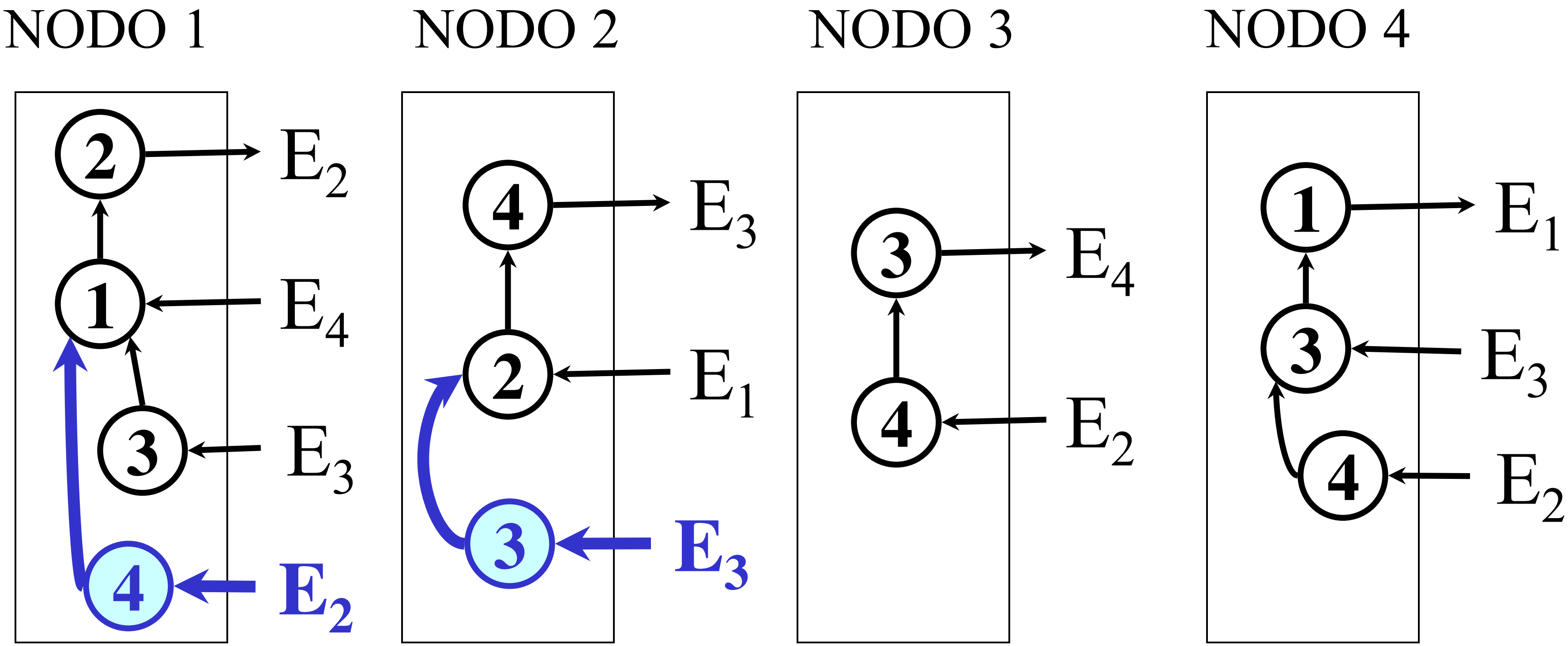


Nulla da inviare

NODO 4



$E_2 \rightarrow t_4 \rightarrow t_1 \rightarrow E_1$ da inviare al nodo 1



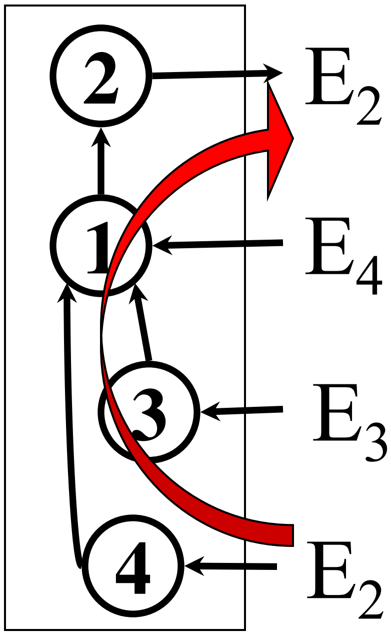
RICEVUTO

$E_2 \rightarrow t_4 \rightarrow t_1 \rightarrow E_1$

RICEVUTO

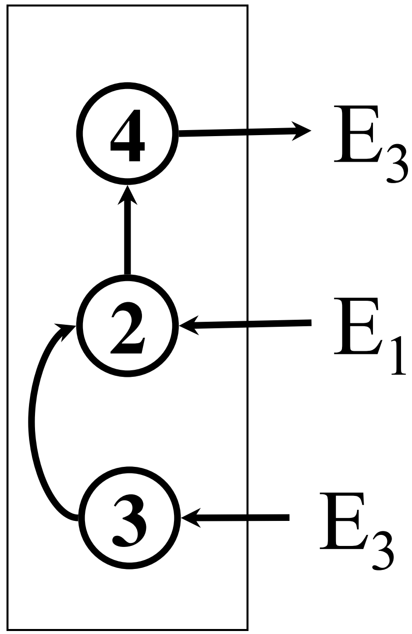
$E_3 \rightarrow t_3 \rightarrow t_2 \rightarrow E_2$

NODO 1



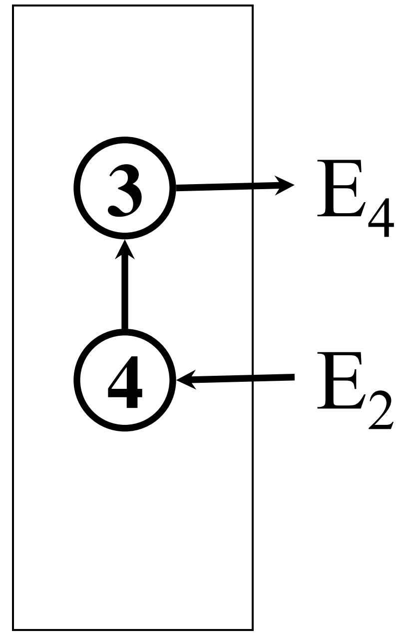
$E_2 \rightarrow t_4 \rightarrow t_2 \rightarrow E_2$ da inviare al nodo 2

NODO 2



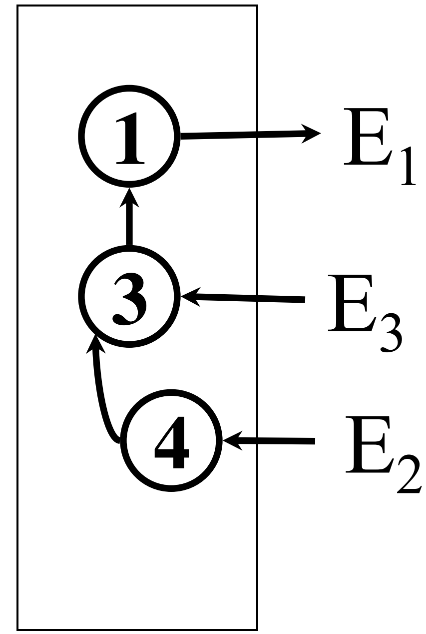
Nulla da inviare

NODO 3

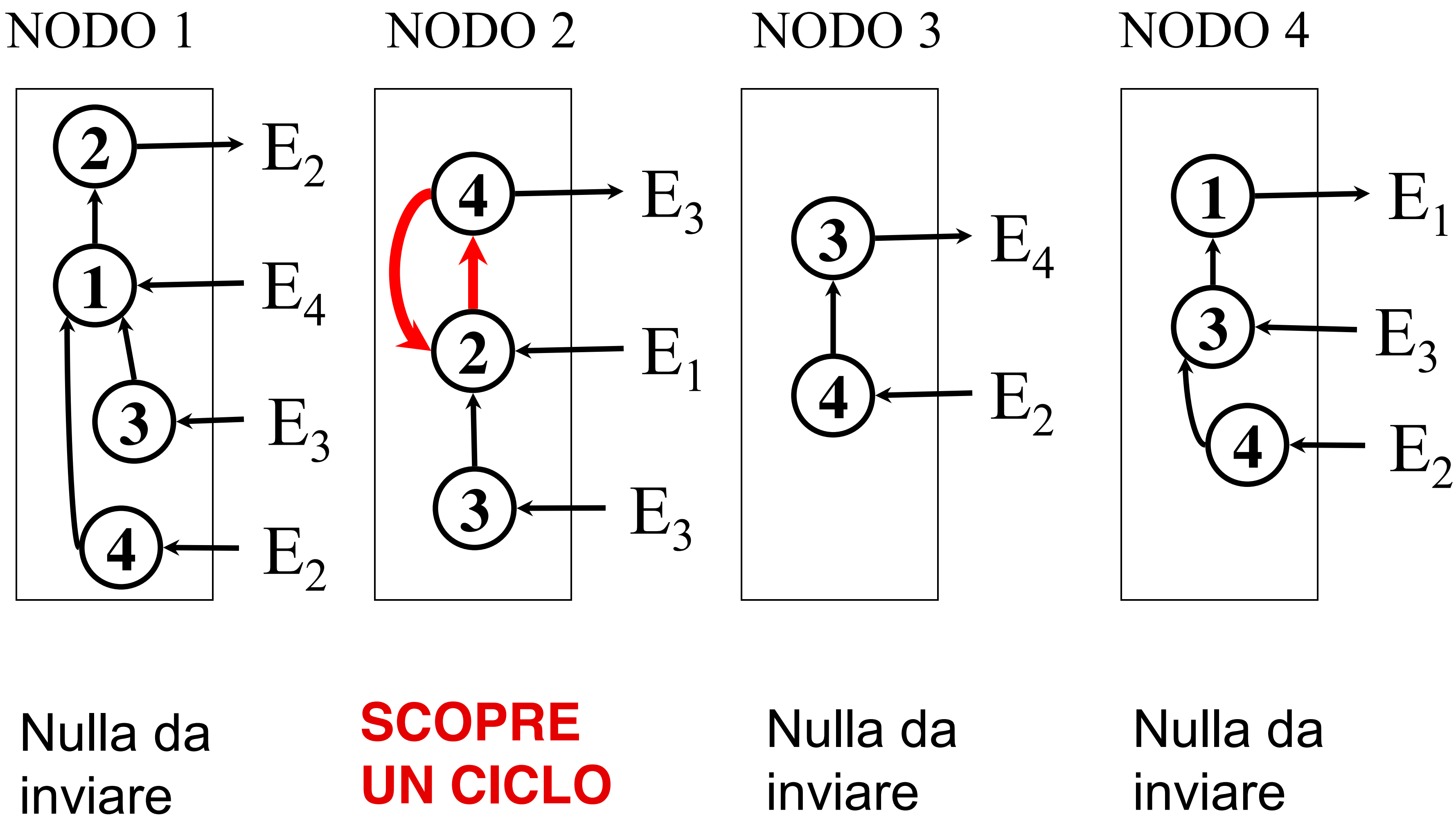


Nulla da inviare

NODO 4



Nulla da inviare



Esiste un blocco critico tra le transazioni t_2 e t_4

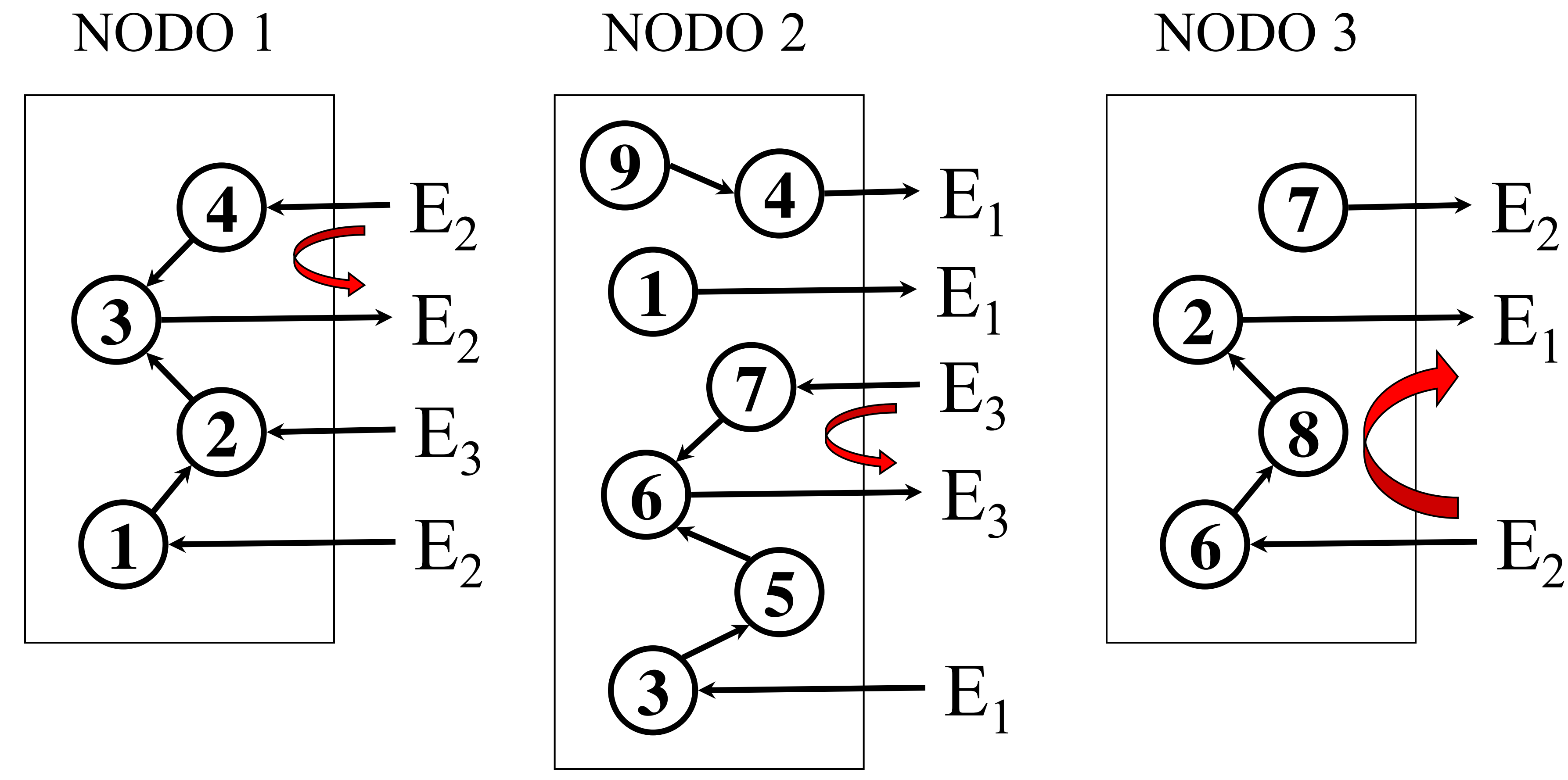
E.2 Dire se le seguenti condizioni di attesa determinano una situazione di blocco critico:

Nodo 1: $E_2 \rightarrow t_1$, $t_1 \rightarrow t_2$, $E_3 \rightarrow t_2$, $t_2 \rightarrow t_3$,
$t_3 \rightarrow E_2$, $E_2 \rightarrow t_4$, $t_4 \rightarrow t_3$

Nodo 2: $E_1 \rightarrow t_3$, $t_3 \rightarrow t_5$, $t_5 \rightarrow t_6$, $t_6 \rightarrow E_3$, $E_3 \rightarrow t_7$,
$t_7 \rightarrow t_6$, $t_9 \rightarrow t_4$, $t_4 \rightarrow E_1$, $t_1 \rightarrow E_1$

Nodo 3: $E_2 \rightarrow t_6$, $t_6 \rightarrow t_8$, $t_8 \rightarrow t_2$, $t_2 \rightarrow E_1$, $t_7 \rightarrow E_2$

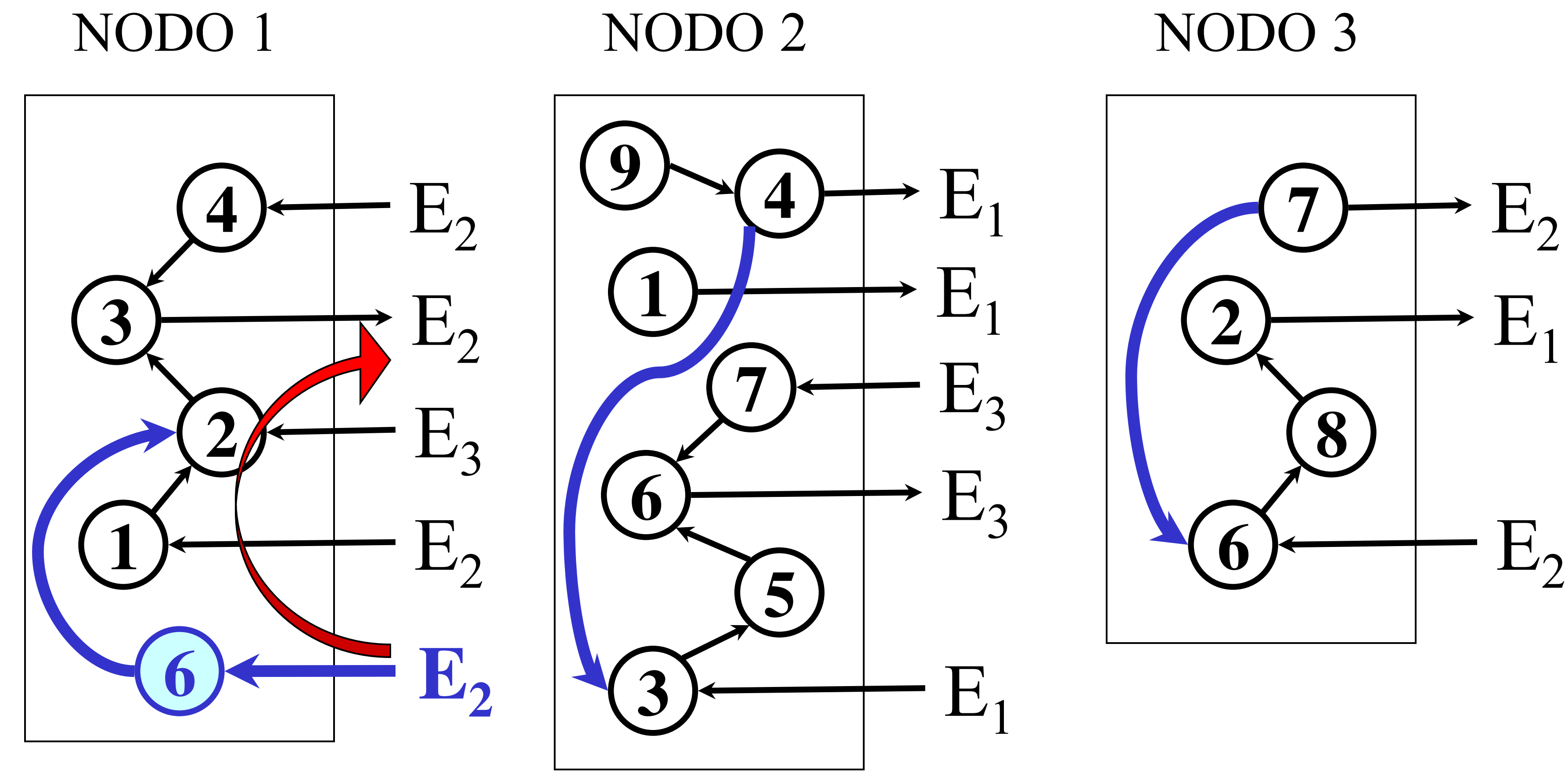
Dove $t_m \rightarrow t_n$ indica un'attesa locale (t_m attende il rilascio di una *risorsa* acquisita da t_n), $t_m \rightarrow E_n$ indica che t_m attende l'esecuzione di una sottotransazione sul *nodo* n (invocazione *sincrona*) e $E_m \rightarrow t_n$ indica che t_n è stata invocata in modo sincrono da una t_i sul nodo m



$E_2 \rightarrow t_4 \rightarrow t_3 \rightarrow E_2$ da inviare al nodo 2

$E_3 \rightarrow t_7 \rightarrow t_6 \rightarrow E_3$ da inviare al nodo 3

$E_2 \rightarrow t_6 \rightarrow t_2 \rightarrow E_1$ da inviare al nodo 1

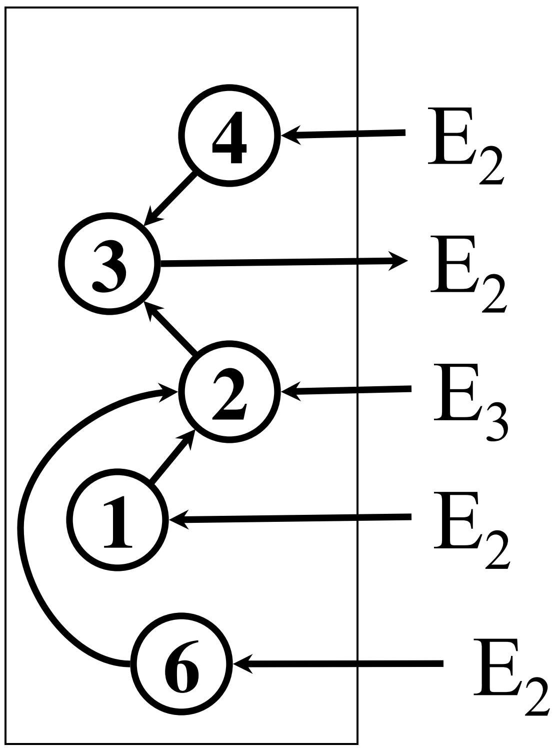


$E_2 \rightarrow t_6 \rightarrow t_3 \rightarrow E_2$ da inviare al nodo 2

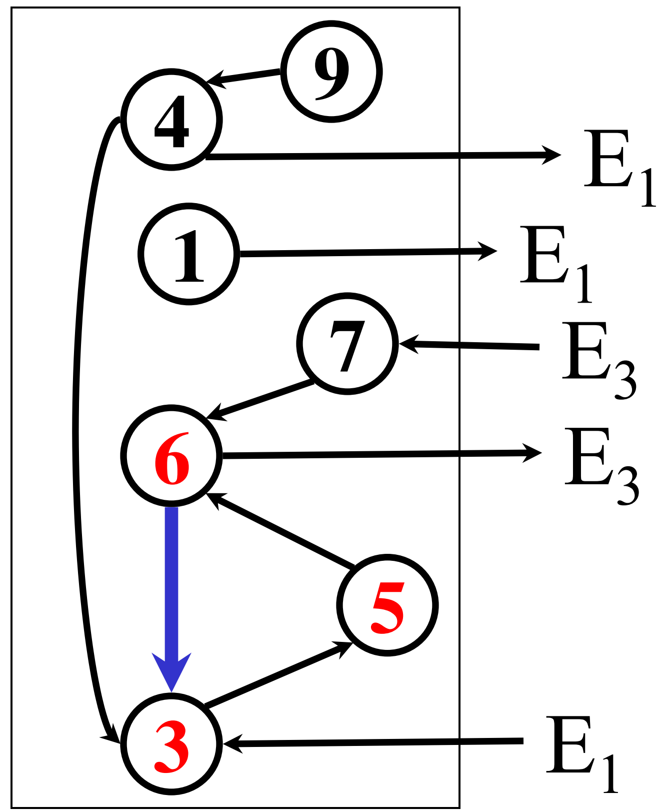
Il messaggio arrivato non causa nuovi messaggi

Il messaggio arrivato non causa nuovi messaggi

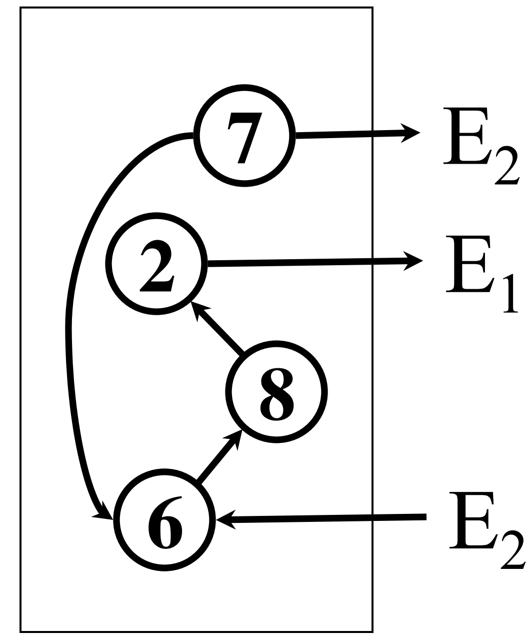
NODO 1

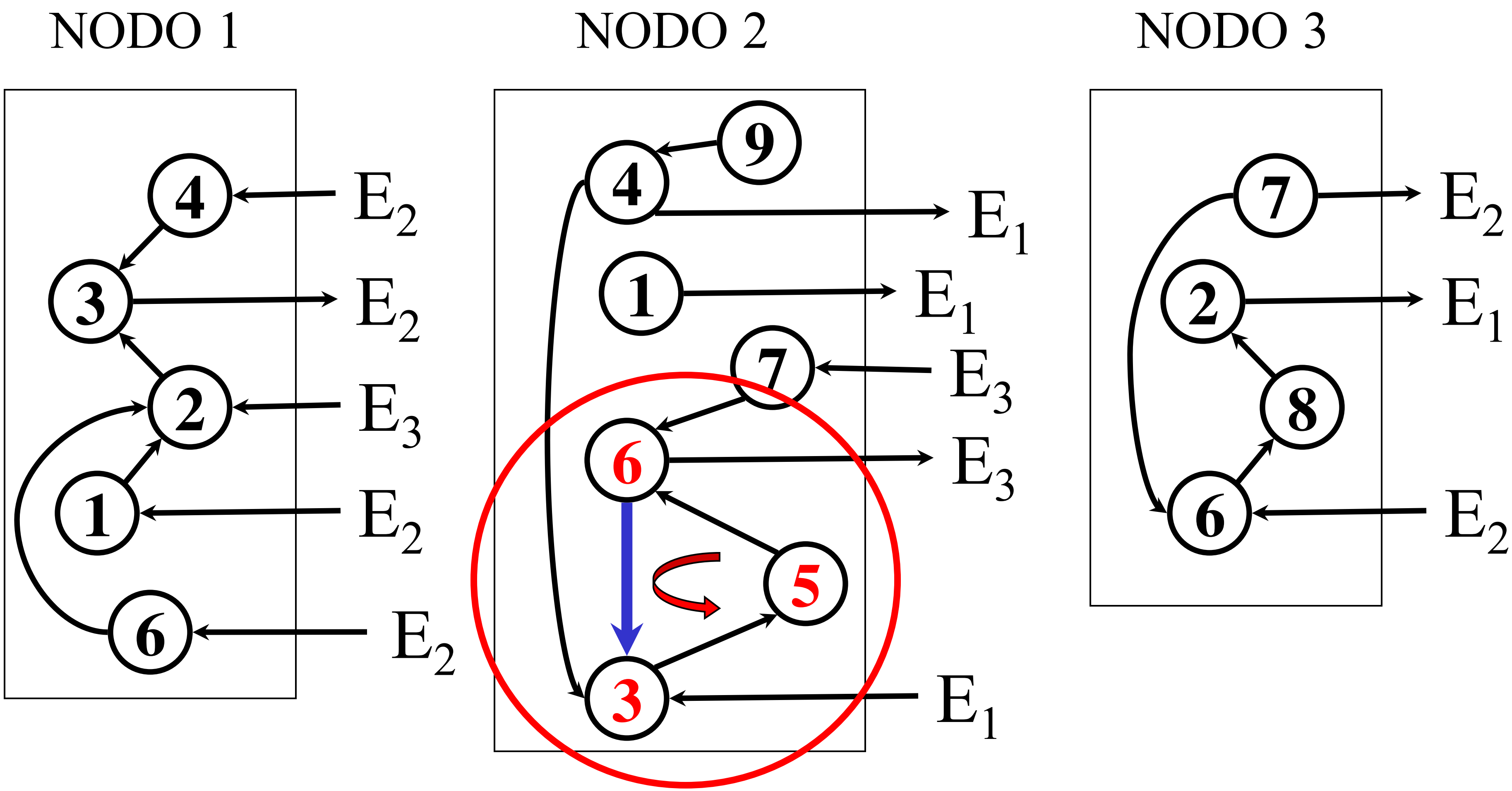


NODO 2



NODO 3





ESISTE UN BLOCCO CRITICO

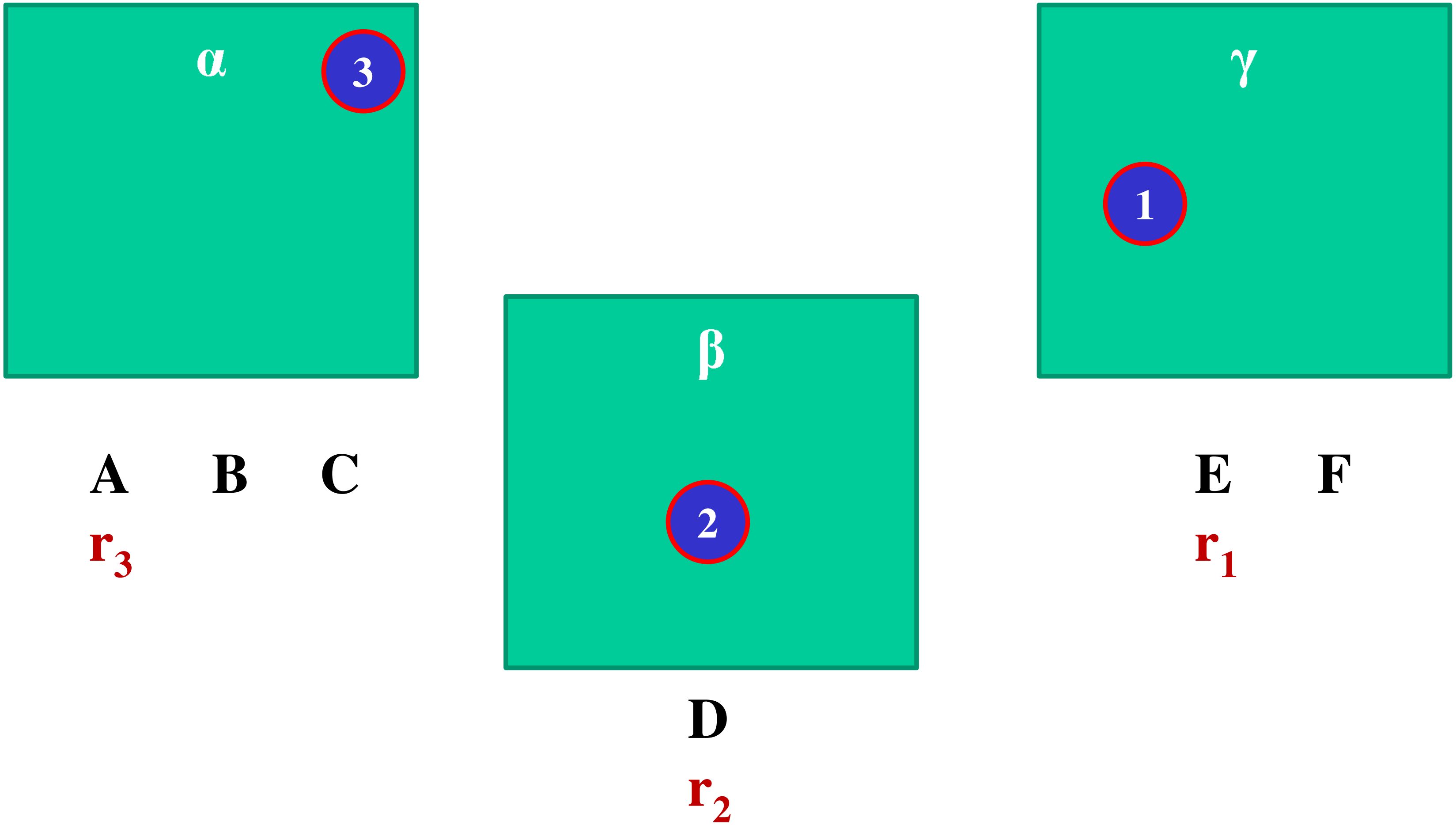
Obermark

Su una base dati distribuita su 3 nodi (**α, β, γ**) sono in esecuzione sei transazioni $\mathbf{T_1 ... T_6}$, che operano sulle risorse **A...F**, così allocate: A,B,C sul nodo α, D sul nodo β e E,F sul nodo γ.

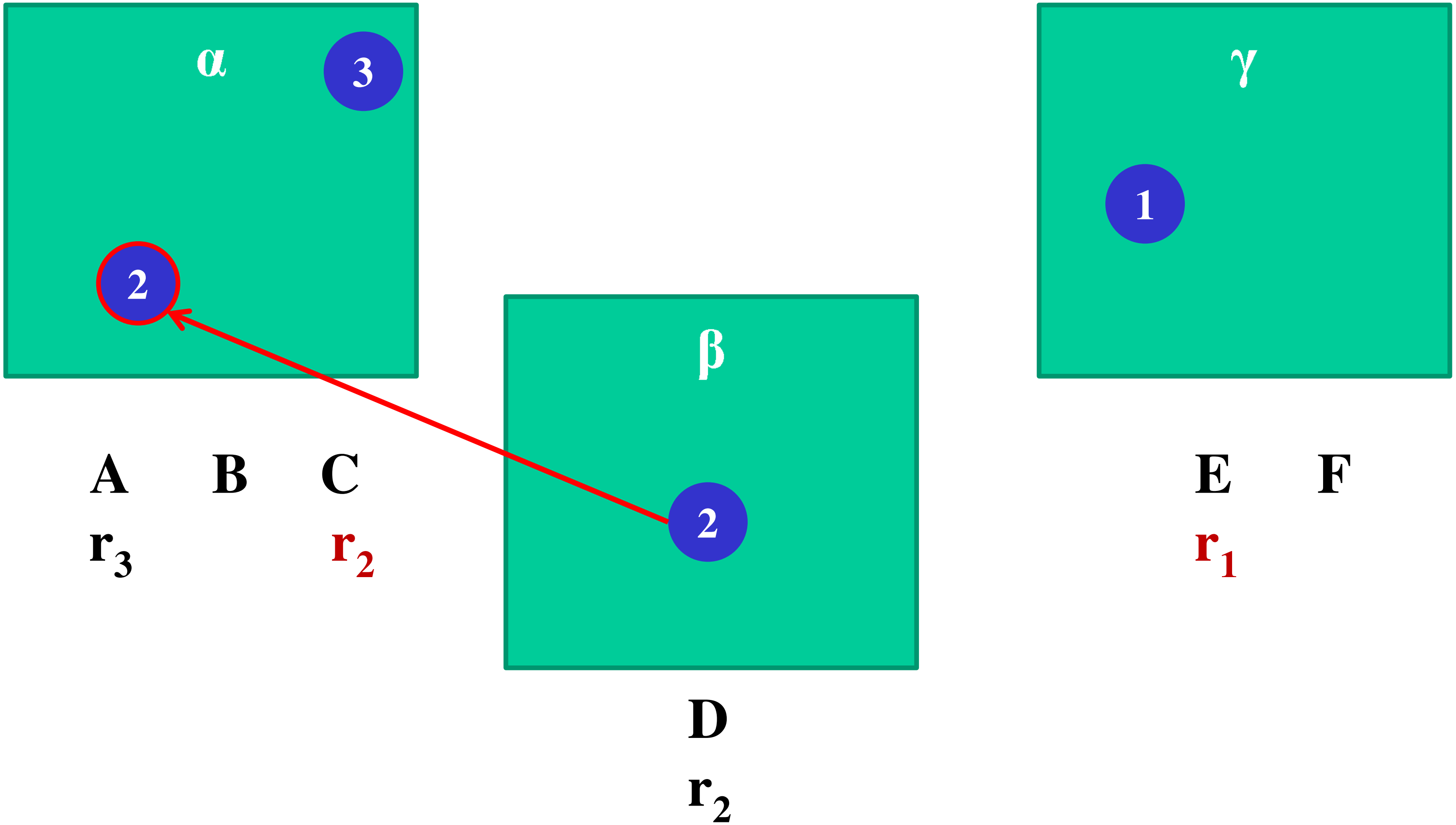
Le operazioni delle transazioni sono state registrate in questo ordine:

$\mathbf{r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)}$

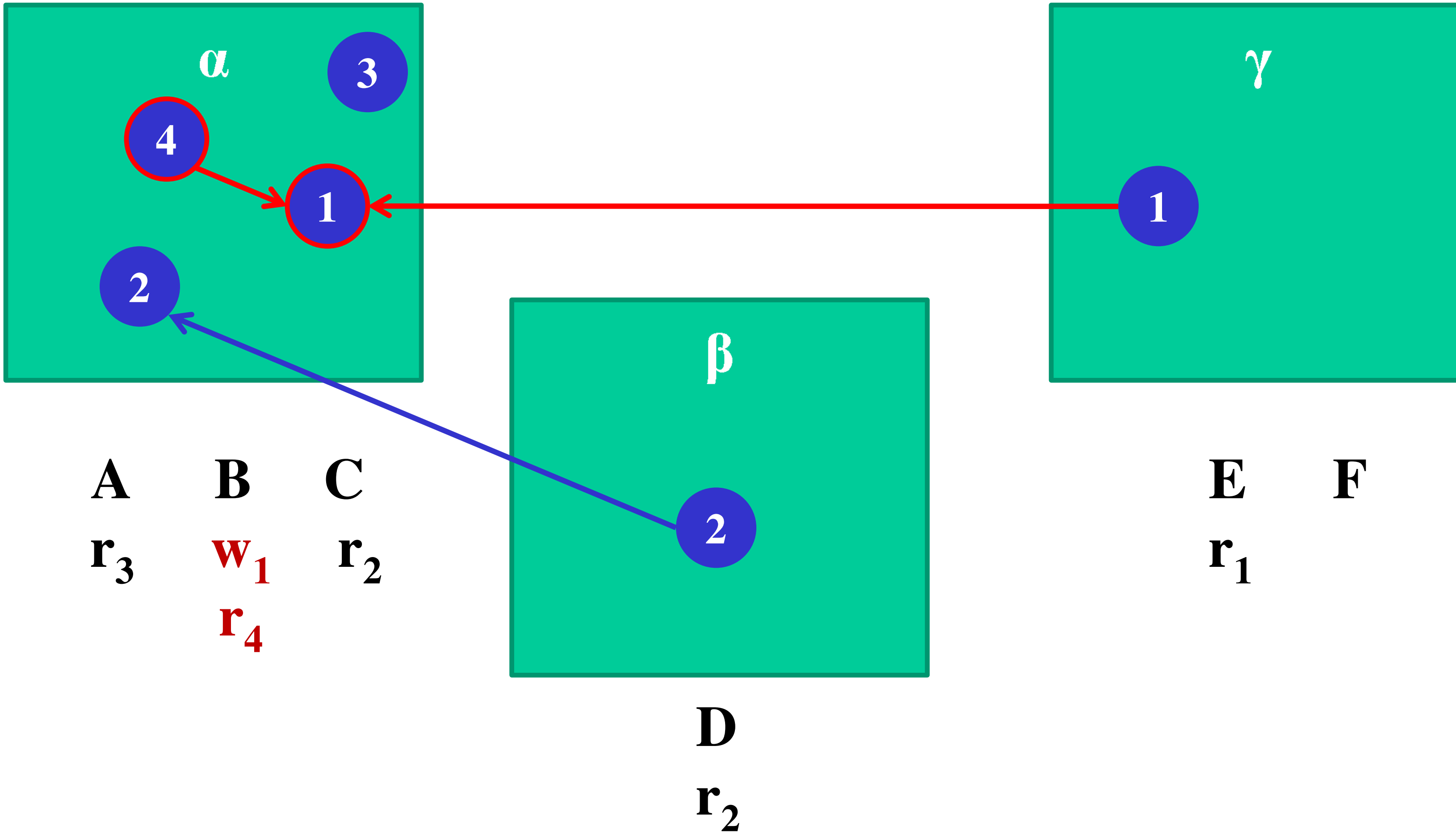
Assumendo che ogni transazione sia iniziata dal nodo su cui si trova la prima risorsa acceduta, e che si verifica l'invocazione di una sotto-transazione quando si accede a una risorsa remota, si costruiscano le condizioni di attesa e le si mostri in forma grafica. Si indichino gli eventuali messaggi da inviare secondo l'algoritmo di Obermark, e se ne simuli l'esecuzione per rilevare eventuali condizioni di deadlock.



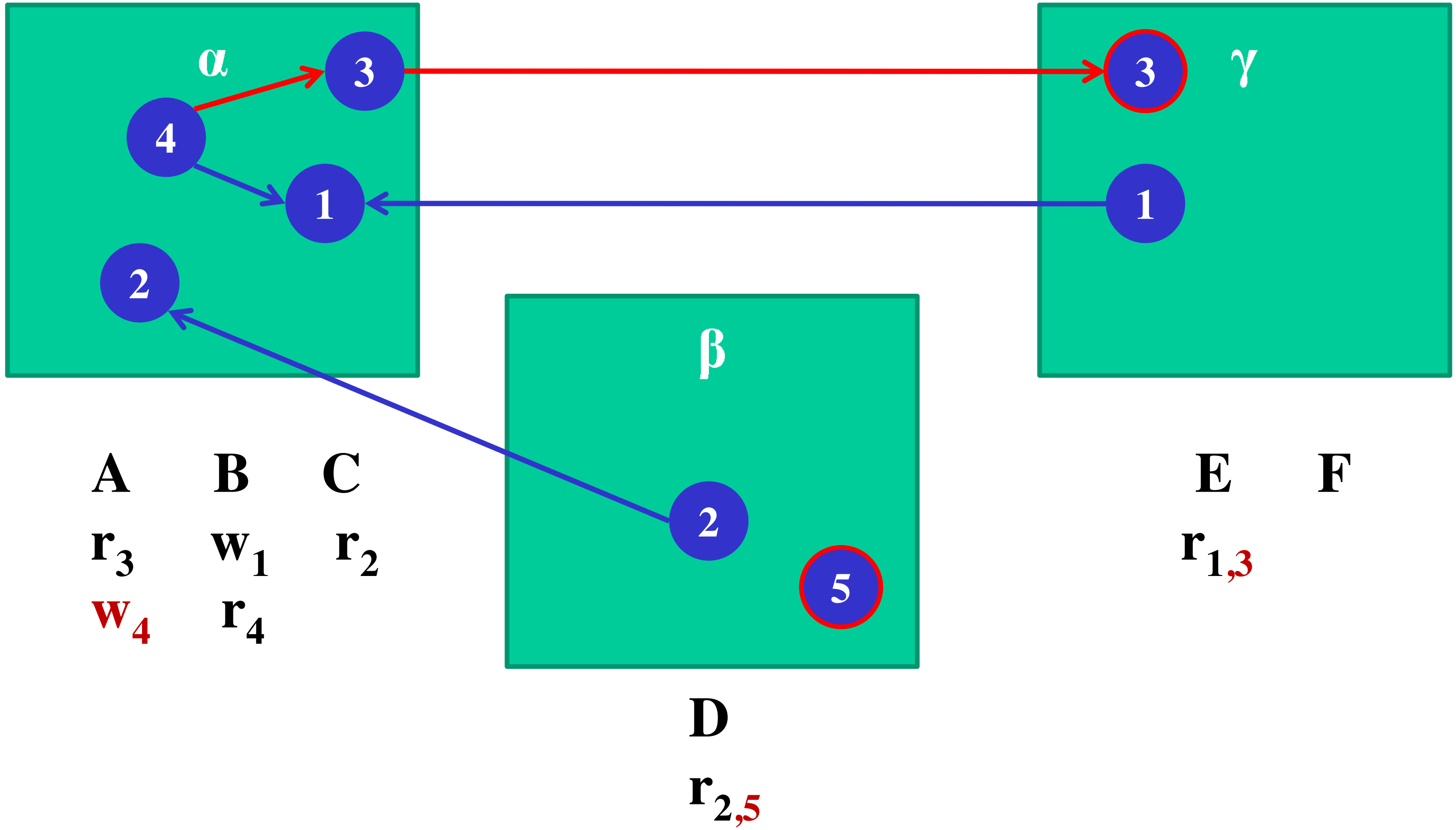
$r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)$



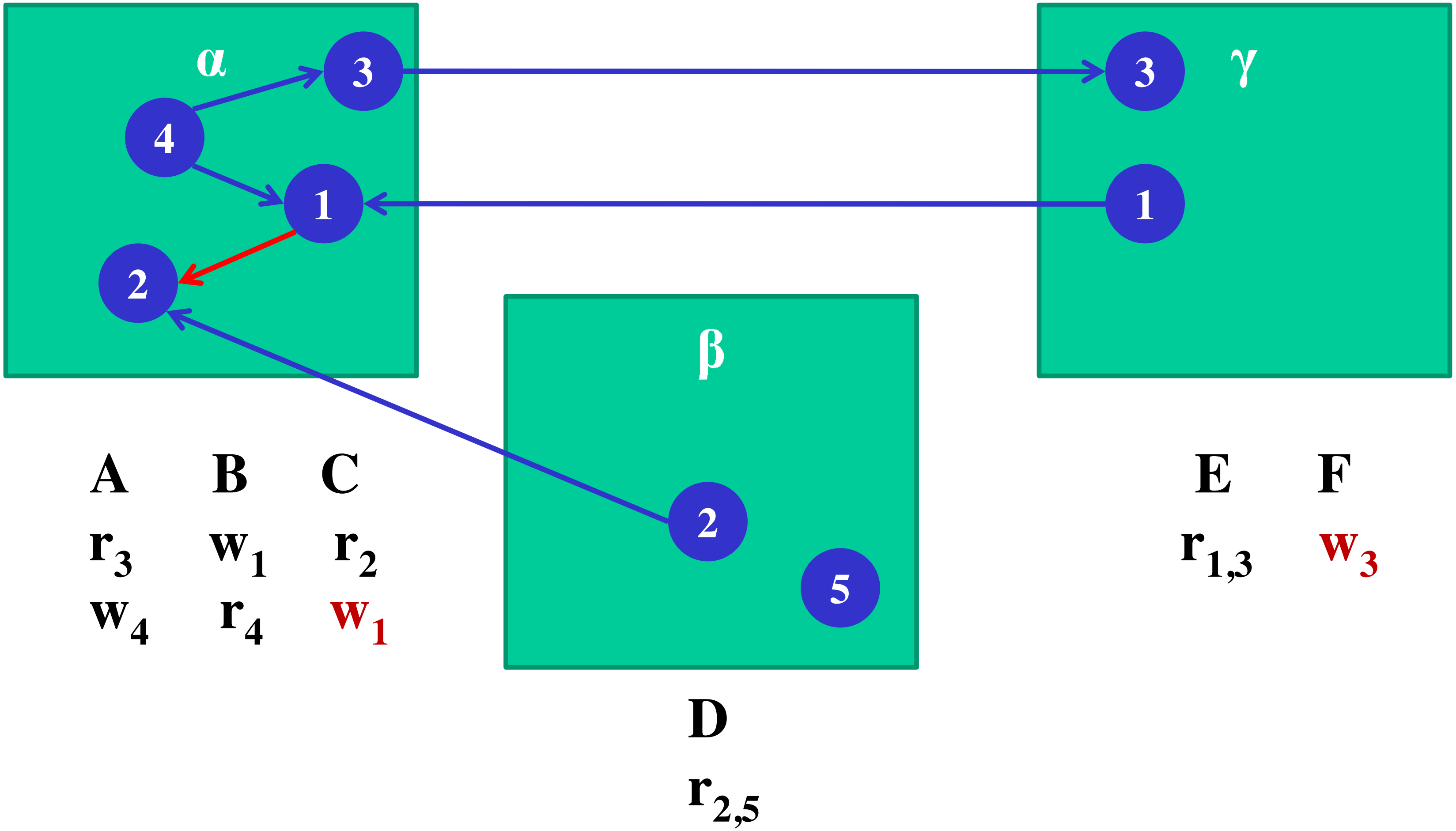
$r_1(E)$, $r_2(D)$, $r_3(A)$, $r_2(C)$, $w_1(B)$, $r_4(B)$, $w_4(A)$, $r_3(E)$, $r_5(D)$, $w_1(C)$, $w_3(F)$, $r_6(D)$, $w_5(E)$, $w_6(D)$



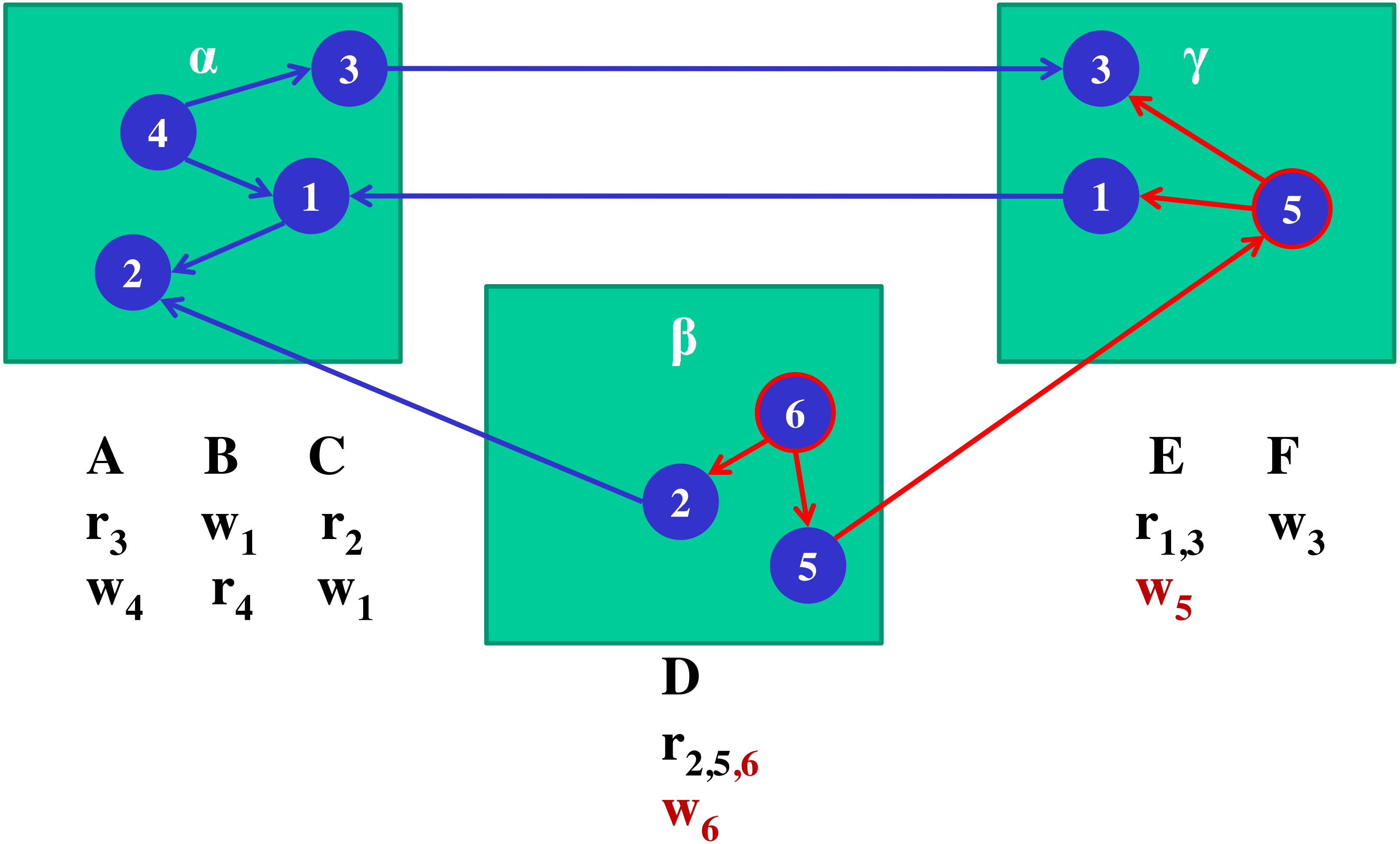
$r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)$



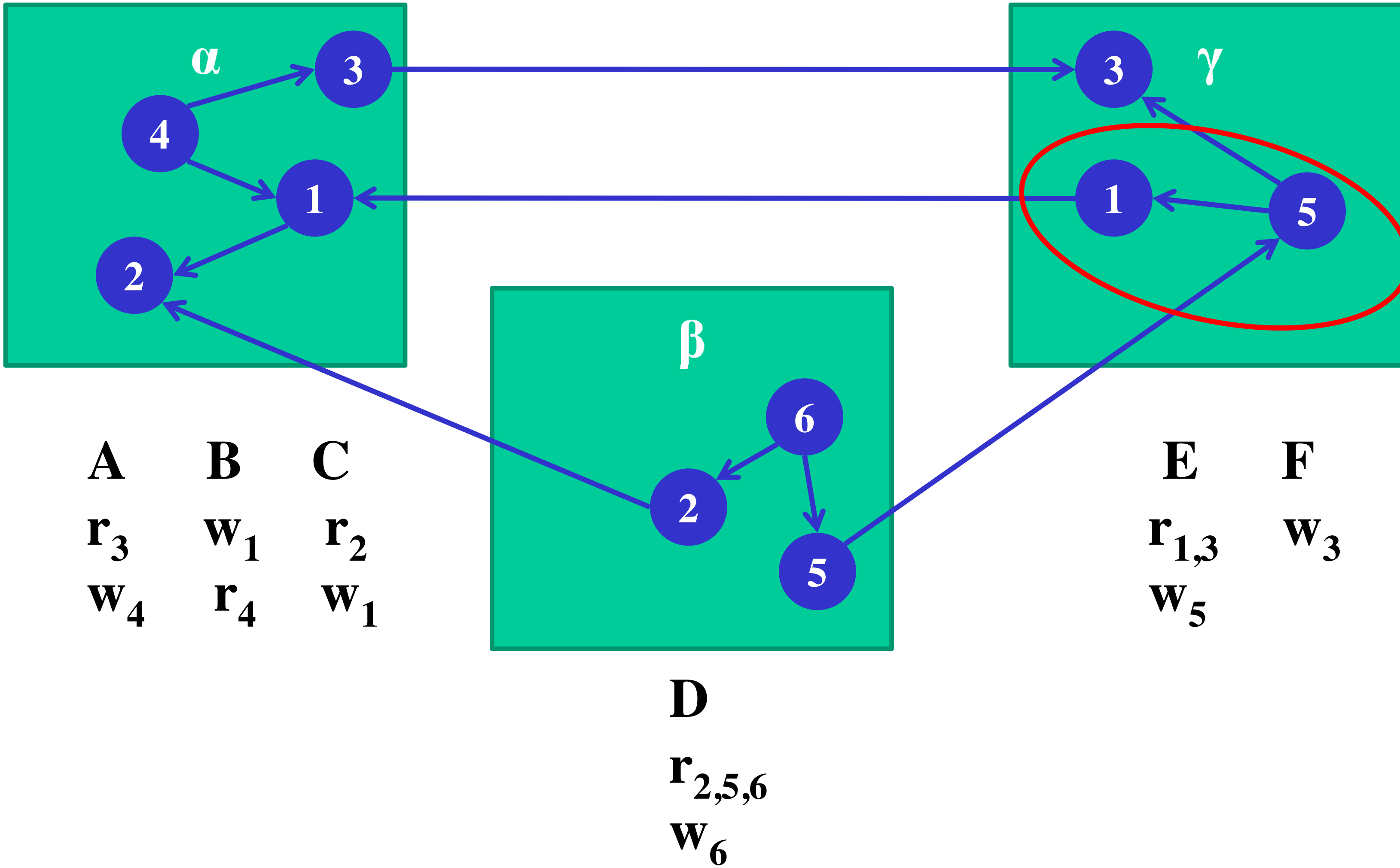
$r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)$



$r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)$



$r_1(E), r_2(D), r_3(A), r_2(C), w_1(B), r_4(B), w_4(A), r_3(E), r_5(D), w_1(C), w_3(F), r_6(D), w_5(E), w_6(D)$



Message: $E_\beta \rightarrow T_5 \rightarrow T_1 \rightarrow E_\alpha$ to node α

No deadlock is found

UniBG Security Lab

Unibg Security Lab is the Computer Security Team at Università degli Studi di Bergamo.

Its work focuses on several areas in computer science such as systems security (UNIX/Linux security), mobile security (especially Android security and malware analysis), information systems, database technology (data warehouses, workflow management systems), Web, emerging technologies and information security (security for databases, access control, secure reputation in P2P networks, data outsourcing and privacy).

The team is often involved in european projects, and is currently working on cloud security technologies with the EscudoCloud project. The recent work on Android security allowed the team to obtain two Google Awards during the last three years. Last but not least, some of the members usually take part in CTF competitions.

We are always looking for smart, hardworking thesis students. If you are interested in computer security, come talk to us!

Source Code

The source code of our open source projects is available at: https://github.com/unibg-seclab/ .

Star the projects on GitHub to receive updates on future releases.

Acknowledgements

Google Award

Winter 2016

The APM project won a Google Award in Winter 2016 batch.

http://seclab.unibg.it