Simulazione distribuita

Approccio pessimistico

Sommario

- Esempio (simulazione del traffico aereo)
- Simulazione parallela ad eventi discreti
  - Logical processes e messaggi time stamped
  - Vincolo di causalità locale (Local causality constraint), gestione della sincronizzazione
- Algoritmo di Chandy/Misra/Bryant
  - Nozioni fondamentali
  - Versione “banale” dell’algoritmo
  - Utilizzo dei “null messages”

Simulazione basata su eventi

Event handler procedures

Variabili di stato

```
Integer: InTheAir;
Integer: OnTheGround;
Boolean: RunwayFree;
```

```
Arrival
   Event
{
   …
}
```

```
Landed
   Event
{
   …
}
```

```
Departure
   Event
{
   …
}
```

"Applicazione"

"Scheduler"

Event processing loop

Now = 8:45

Pending Event List (PEL)

9:00

9:16

10:10

```
While (simulation not finished)
   E = smallest time stamp event in PEL
   Remove E from PEL
   Now := time stamp of E
   call event handler procedure
```

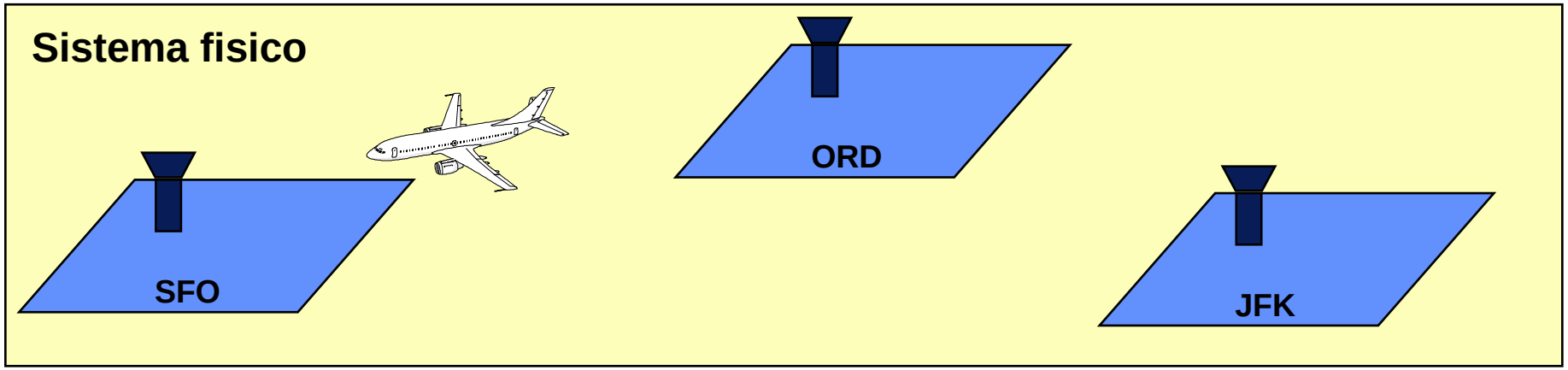
Simulazione ad eventi discreti

- Come possiamo estendere l'esempio ad un insieme di aeroporti?
  - Ogni aeroporto = Logical Process
  - I Logical Processes possono schedulare eventi (spedire messaggi) ad altri LP

Generalizzando:

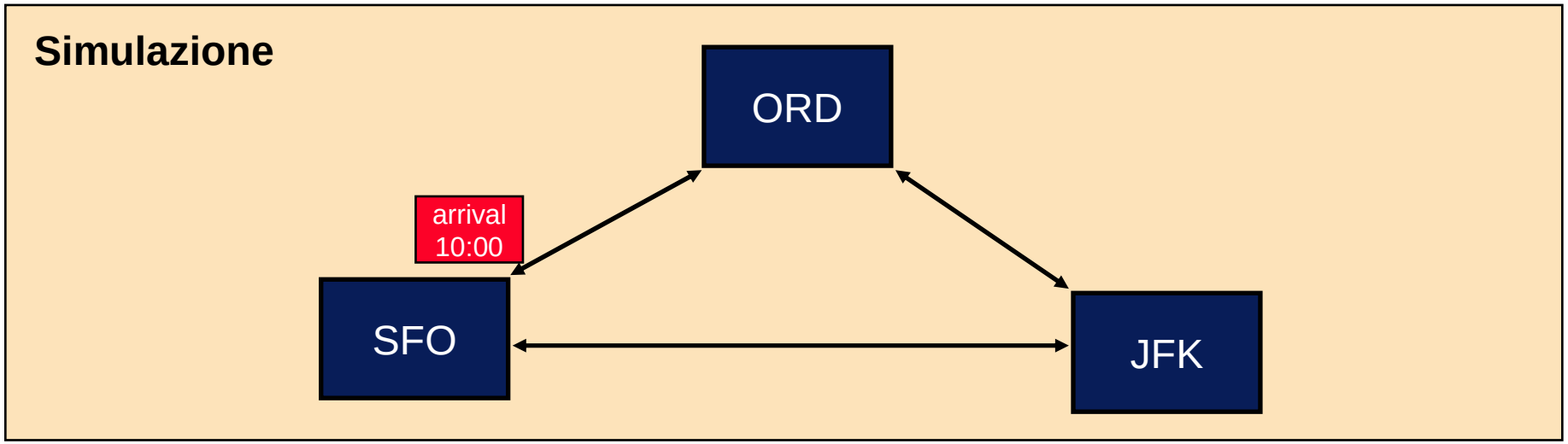
- Sistema fisico = collezione di processi fisici che interagiscono tra loro
- Simulazione distribuita =
  - Collezione di logical processes (LPs)
  - Ogni LP ha il compito di modellare un processo fisico
  - Le interazioni tra processi fisici sono modellate schedulando eventi tra gli LP

Esempio



processo fisico → processo logico

interazioni tra processi fisici → eventi time stamped



tutte le interazioni tra LP devono avvenire per mezzo di messaggi!

Esempio

- Now: **simulation time attuale**
- InTheAir: **numero di aerei in atterraggio o in attesa di atterrare**
- OnTheGround: **numero di aerei già atterrati**
- RunwayFree: **Boolean, vero se la pista è libera**

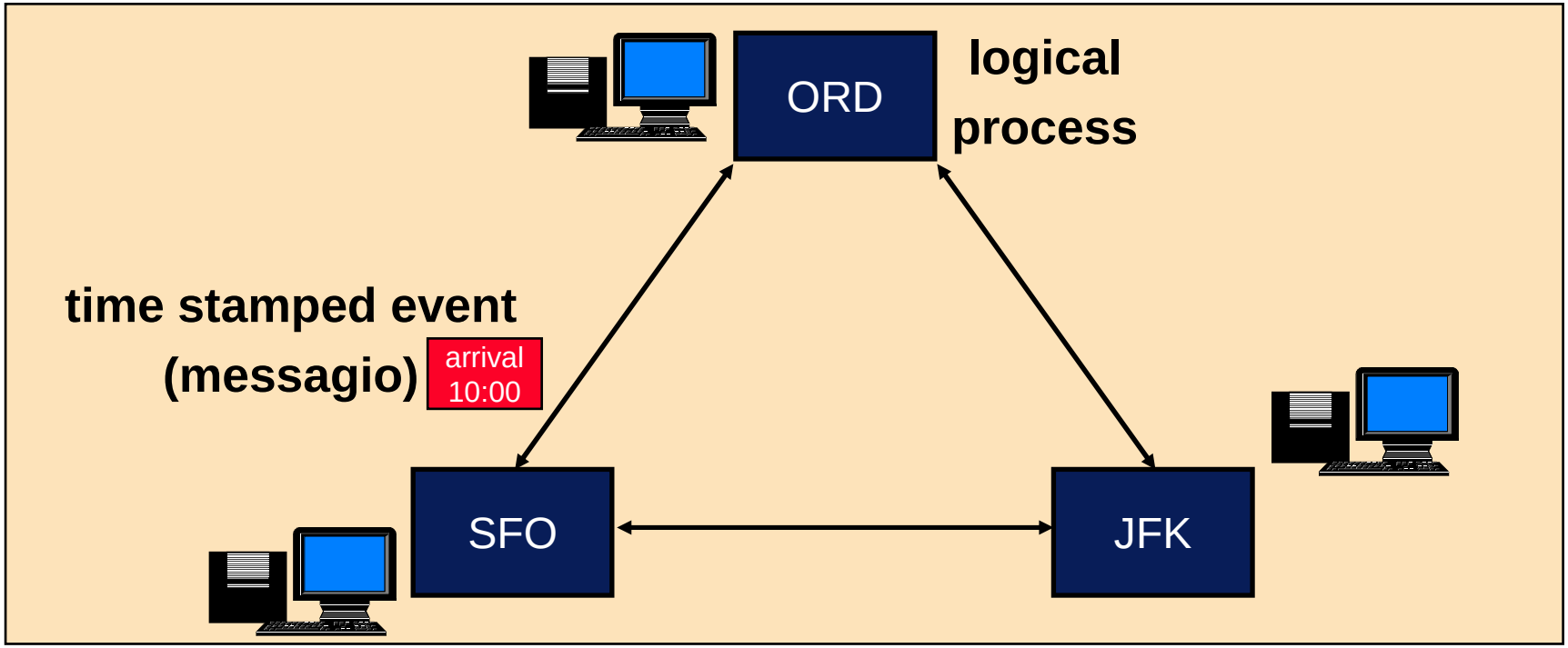
```
Arrival Event:
InTheAir := InTheAir+1;
If (RunwayFree)
   RunwayFree:=FALSE;
   Schedule Landed event (local) @ Now+R;
```

```
Landed Event:
InTheAir:=InTheAir-1;      OnTheGround:=OnTheGround+1;
Schedule Departure event (local) @ Now + G;
If (InTheAir>0) Schedule Landed event (local) @ Now+R;
Else RunwayFree := TRUE;
```

```
Departure Event (D = delay to reach another airport):
OnTheGround := OnTheGround - 1;
Schedule Arrival Event (remote) @ (Now+D) @ another airport
```

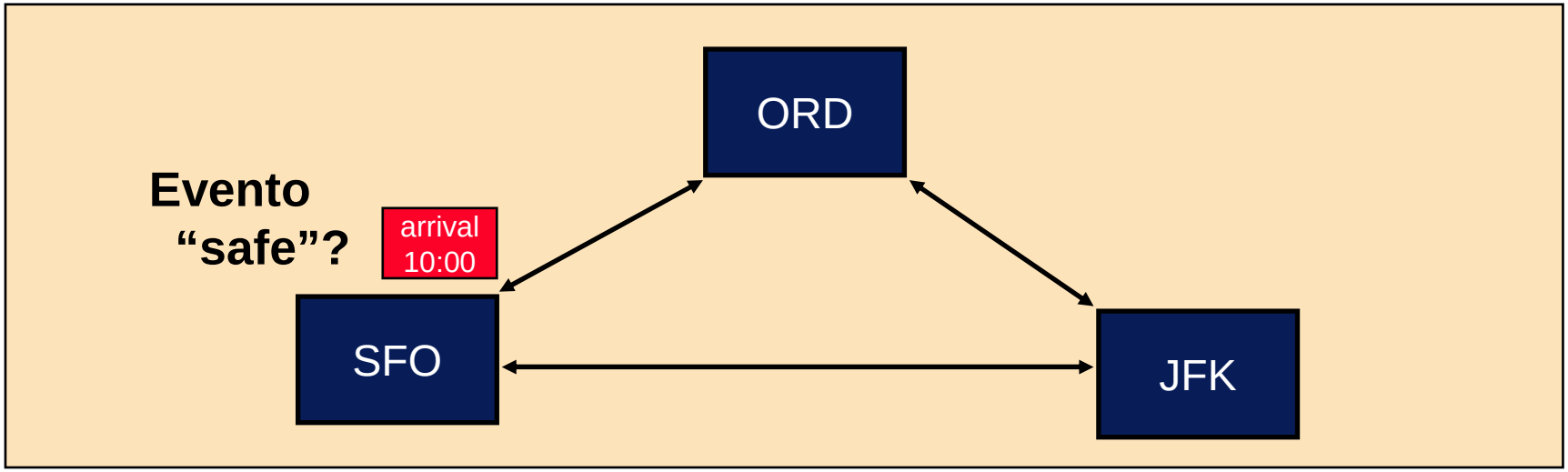
Analisi del modello

- L’approccio ad LP sembra consistente col parallelismo
- Ogni LP può essere associato ad un differente processore
  - Più LP possono essere allocati sulla stessa CPU
- Comunicazione: message passing
  - Tutte le interazioni avvengono attraverso messaggi
  - NON esistono variabili condivise o “comunicazioni esterne”



Vincolo di causalità locale

Data una simulazione ad eventi discreti formata da un insieme di Logical Process (LP) che interagiscono esclusivamente per mezzo di messaggi time stamped, la simulazione rispetta il vincolo di causalità locale se e solo se ogni LP processa gli eventi (messaggi) in ordine non decrescente rispetto al loro time stamp.



Sincronizzazione

Se ogni LP rispetta il vincolo di causalità locale allora la simulazione distribuita otterrà esattamente gli stessi risultati dell'esecuzione sequenziale della simulazione.

La condizione di "rispetto della causalità locale" risulta quindi sufficiente per un'esecuzione causalmente corretta in ambiente distribuito.

Due eventi con uguale time stamp sono definiti simultanei: la condizione di causalità locale non impone vincoli riguardo gli eventi simultanei.

Problema di sincronizzazione: è necessario un algoritmo che garantisca la condizione di sufficienza, ovvero che ogni LP elabori gli eventi secondo il loro ordine di time stamp.

Algoritmi di sincronizzazione

- Approccio conservativo: evitare la violazione del vincolo di causalità locale (banalmente tradotto in “aspetta fino a quando non abbiamo sufficienti informazioni per considerare l’evento safe”)
  - Evitare il deadlock attraverso null-messages (Chandy/Misra/Bryant)
  - Riconoscimento dello stato di deadlock e conseguenti azioni di recovery
  - algoritmi sincroni (ad esempio esecuzione in “round”)

- Approccio ottimistico: il vincolo può essere localmente violato, la violazione va rilevata ed è necessario un recovery basato su meccanismo di rollback
  - Time Warp (Jefferson)
  - altri algoritmi più complessi

Sommario

- Esempio (simulazione del traffico aereo)

- Simulazione parallela ad eventi discreti
  - Logical processes & time stamped messages
  - Vincolo di causalità locale (Local causality constraint), gestione della sincronizzazione

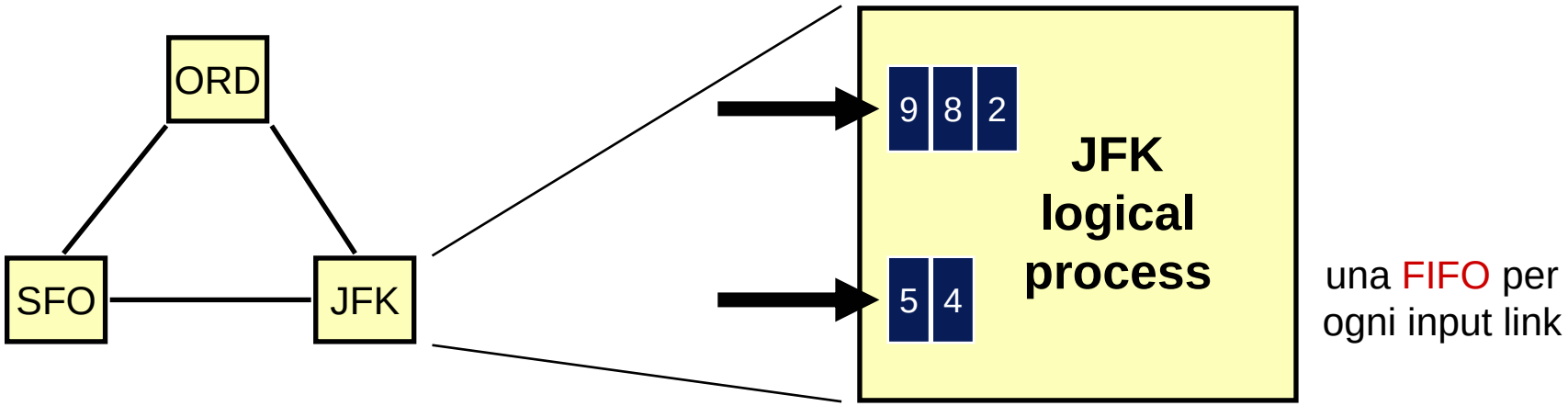
- Algoritmo di Chandy/Misra/Bryant
  - Nozioni fondamentali
  - Versione banale
  - Utilizzo dei “null messages”

Algoritmo “Chandy/Misra/Bryant”

Assunzioni:

- I logical processes (LPs) si scambiano messaggi time stamped
- Topologia statica, nessuna modifica dinamica di link o LP
- I messaggi vengono spediti su ogni link con ordinamento non decrescente in base al time stamp
- La rete è affidabile (non perde messaggi) e preserva l’ordinamento

Nota: queste assunzioni implicano che il time stamp dell’ultimo messaggio ricevuto da un link sia un **lower bound dei time stamp (LBTS)** per i messaggi che verrano ricevuti attraverso quel link

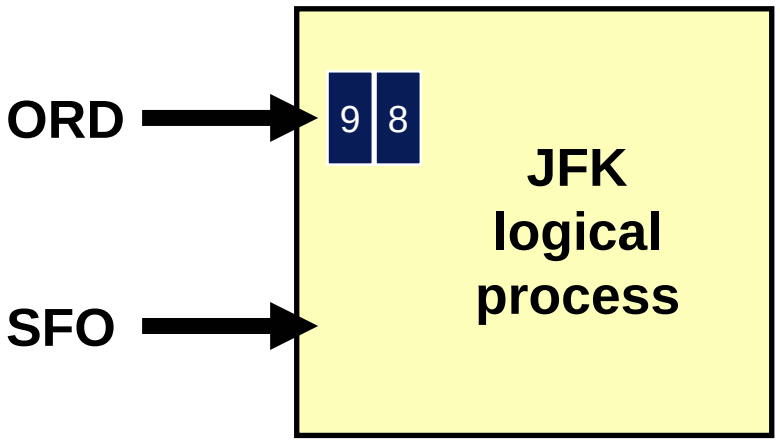


Obiettivo: Assicurare che ogni LP processi gli eventi in ordine di time stamp

Algoritmo conservativo “semplice”

Ogni LP che partecipa alla simulazione esegue in maniera indipendente un’istanza dell’algoritmo.

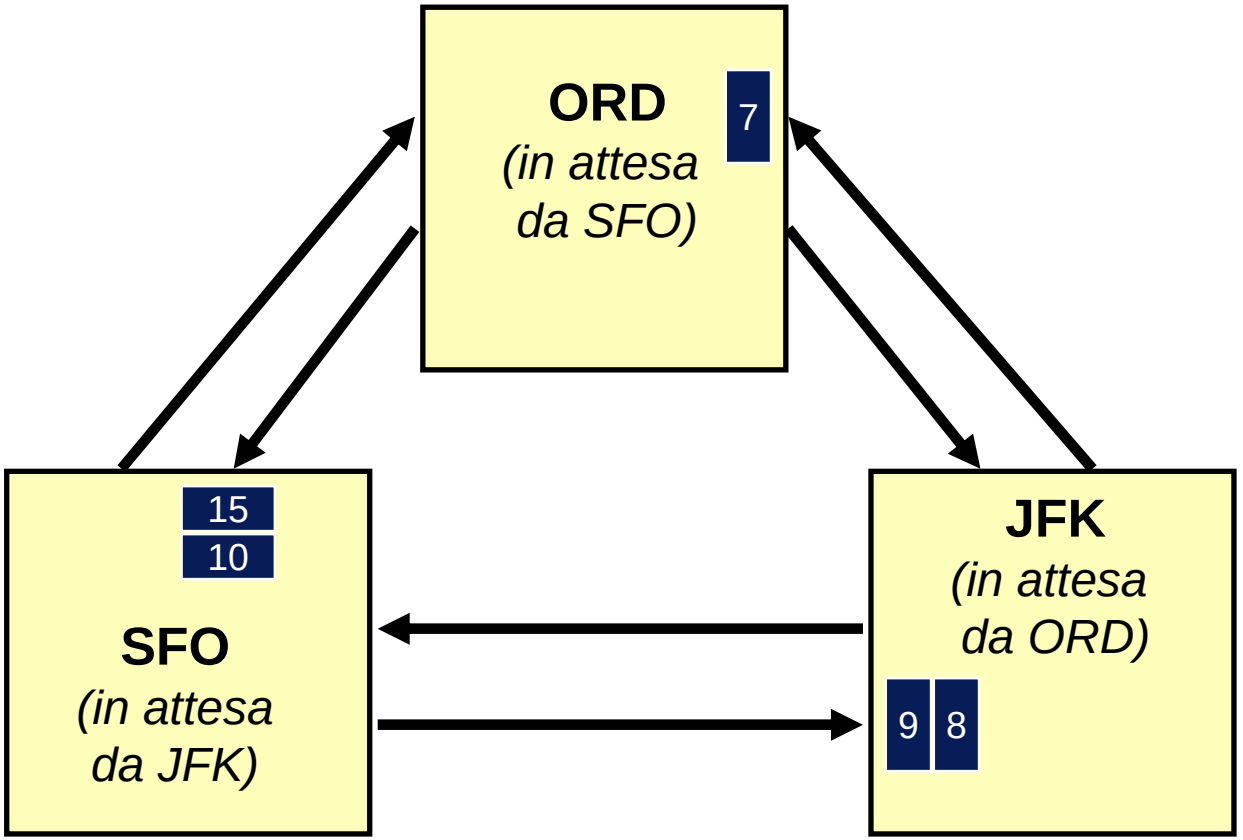
WHILE (simulation is not over)
    wait until each FIFO contains at least one message
    remove smallest time stamped event from its FIFO
    process that event
END-LOOP



- processato l’evento con TS 2
- processato l’evento con TS 4
- processato l’evento con TS 5
- attesa di un evento da SFO

Osservazione: rischiamo il deadlock del sistema!

Esempio di deadlock



Siamo in presenza di una condizione di attesa circolare: tutti gli LP sono in attesa di eventi da altri LP. Nessuno di loro è quindi autorizzato a procedere nella computazione e non esistono eventi esterni che possano sbloccare la situazione: deadlock!

Condizioni necessarie al deadlock

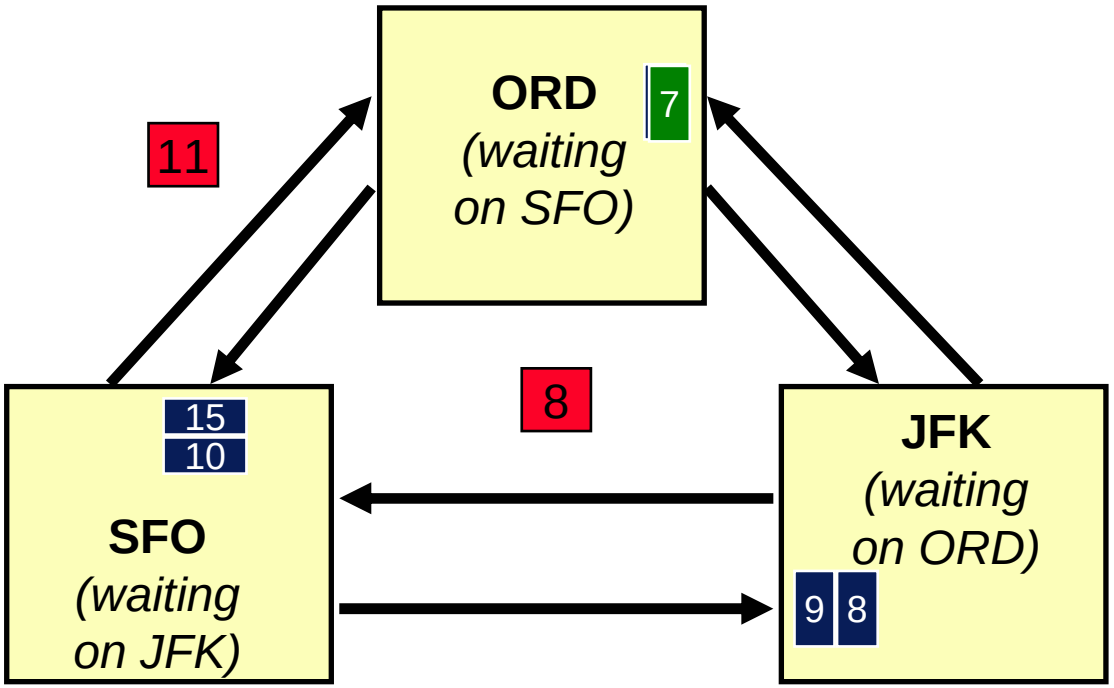
La caratterizzazione del deadlock prevede che debbano sussistere contemporaneamente 4 condizioni:

- Mutua esclusione
- Hold & Wait (assenza di rilascio)
- Assenza di pre-emption
- Attesa circolare

Nel caso preso in esame, le condizioni sono tutte verificate quindi siamo effettivamente in presenza di un deadlock. Come possiamo evitarlo?

Evitare il deadlock: messaggi “null”

Ogni LP spedisce messaggi dal contenuto “null” con lo scopo di specificare il “lower bound” al time stamp dei suoi messaggi futuri.



Assumiamo un **ritardo minimo** tra gli aeroporti quantificato in 3 unità di tempo:

- JFK inizialmente è al tempo 5
- JFK spedisce “null” a SFO con Time Stamp (TS) 8
- SFO spedisce “null” a ORD con TS 11
- ORD è sbloccato, processa il messaggio con TS 7

Messaggi nulli: algoritmo

Ogni LP esegue indipendentemente un'istanza dell'algoritmo

```
WHILE (simulation is not over)
   wait until each FIFO contains at least one message
   remove smallest time stamped event from its FIFO
   process that event
   send null messages to neighboring LPs with time stamp indicating
   a lower bound on future messages sent to that LP (current time
   plus lookahead)
END-LOOP
```

Da notare come l'algoritmo sia basato interamente sul concetto di lookahead! Il lookahead è una caratteristica intrinseca del modello.

Conclusioni

- Esempio (simulazione del traffico aereo)
- Simulazione parallela ad eventi discreti
  - Logical processes e messaggi time stamped
  - Vincolo di causalità locale (Local causality constraint), gestione della sincronizzazione
- Algoritmo di Chandy/Misra/Bryant
  - Nozioni fondamentali
  - Versione banale
  - Utilizzo dei “null messages”