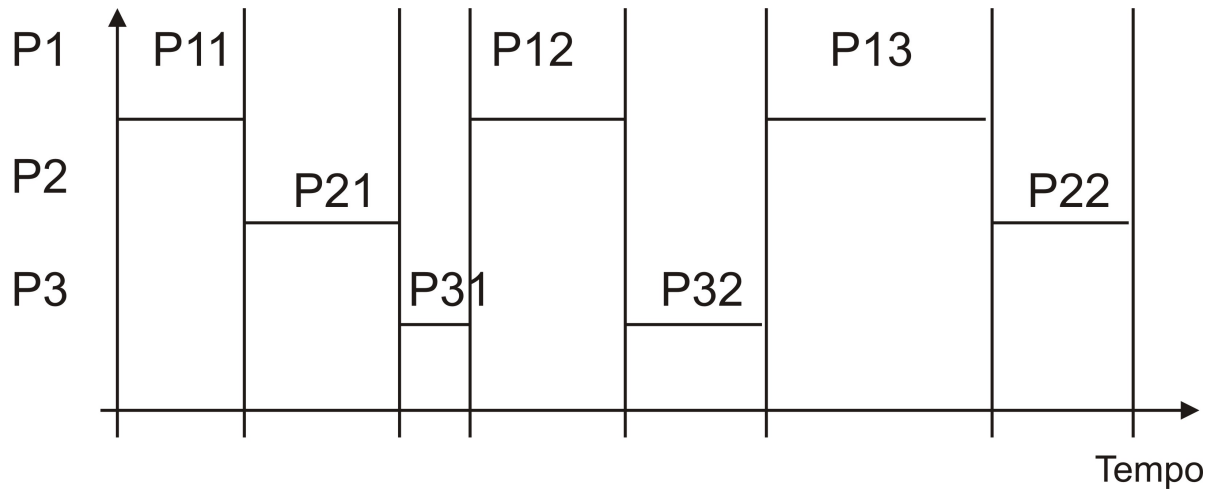
Introduzione alla concorrenza

Indice degli argomenti

- generalitá, interleaving
- grafi delle precedenze: semplificazione, tempi di calcolo
- strumenti linguistici(coroutine,fork-join,cobegin-coend)
- sincronizzazione
- determinatezza

Definizioni

- Interleaving: Nel caso di una sola CPU l'esecuzione procede per intersfogliatura (interleaving) delle istruzioni. Nel caso ci siano piú processi l'esecuzione comprende anche delle sovrapposizioni.



- Context switch: realizza la concorrenza, cioé il meccanismo che consente ai processi di eseguire simultaneamente.

Definizioni

- Processi concorrenti vs. Threads concorrenti
- Atomicitá: esecuzione dall'inizio alla fine senza interferenza
- Contesa (race condition): i processi accedono una risorsa condivisa: il risultato dipende dall'ordine di accesso.
- Attesa indefinita (starvation): non c'e' un blocco ma una attesa indefinita.
- Sezione critica : é una sezione di codice che accede ad una risorsa condivisa con un'altro processo.
- Mutua esclusione (mutual exclusion): una risorsa condivisa tra piú processi puó essere usata solo da un processo alla volta. Non ci possono essere piú processi che usano simultaneamente quella risorsa.
- Stallo (deadlock): blocco di due processi in cui ciascun processo aspetta che l'altro faccia qualcosa prima di eseguire.

Cosé un thread?

Consideriamo ad esempio questo semplice codice sequenziale Java

```
class MulAdd{
    static float mul(float a,float b){
         return a*b;
    }
    static float sum(float a,float b,float c,float d){
         return a+b+c+d;
    }
    public static void main (String [] args){
       float[] X = {1.3f,2.7f,3.14f,7.34f,8.33f,6.212f};
       float[] Y = {3.12f,2.54f,1.23f,5.676f,6.21f,4.21f};

       float a=mul(X[1],Y[1]); float b=mul(X[2],Y[2]); float c=mul(X[3],Y[3]); float d=mul(X[4],Y[4]);
       float ris=sum(a,b,c,d);

       System.out.println("risultato="+ris);
    }
}
```

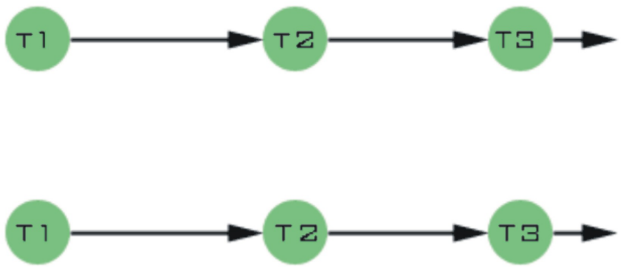
Se i metodi fossero eseguiti in concorrenza −> maggiore efficienza!

– Un thread é simile all'esecuzione automatica dei metodi
– Caratteristica principale: condivisione dell'ambiente tra i diversi thread
– Necessitá principale: sincronizzare i thread!

Grafi delle precedenze

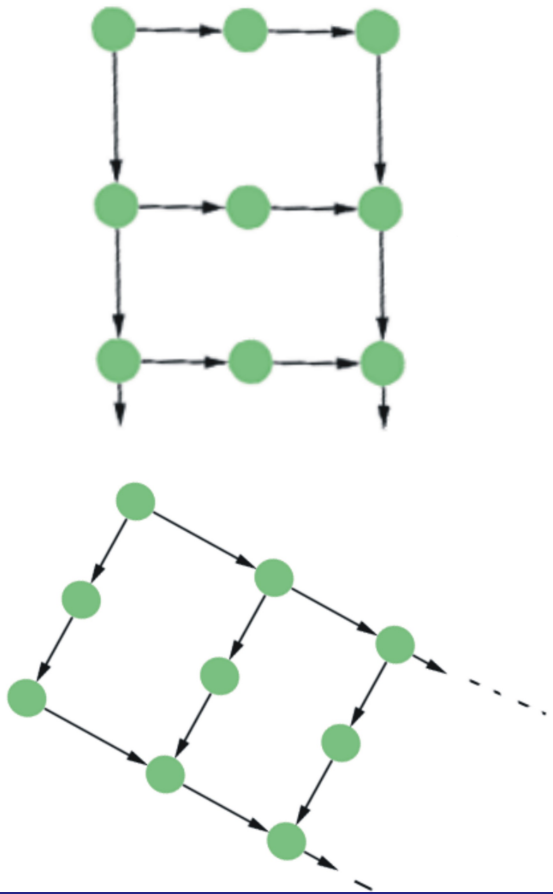
Esempio. Calcolo sequenziale di tre processi: uno legge da un nastro, uno elabora e uno scrive su un nastro

Grafo dei tre processi sequenziali



Grafi delle precedenze

Grafo dei threads concorrenti delle tre elaborazioni



Grafi delle precedenze

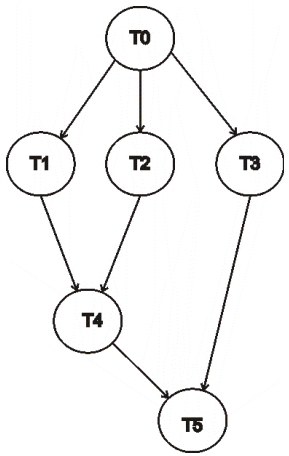
- Grafi costituiti da nodi (unitá computazionale) collegato da archi orientati (relazione tra nodi)
- Gli archi orientati definiscono una relazione d'ordine tra i nodi
- Se il nodo A viene prima del nodo B, allora lattivit B non pu inziare finch lattivit A non terminata.
- Perché sono importanti?
  - modellare e descrivere una applicazione concorrente dividendola tra entitá concorrenti
  - semplificare la applicazione eliminando archi ridondanti
  - massimizzare il parallelismo
  - descrivere il tipo di implementazione
  - uso corretto delle variabili assicurando la determinatezza

Passi per la costruzione di un programma concorrente

- Prima di tutto, Processi concorrenti o Thread concorrenti ?
- Definizione delle precedenze: grafo delle precedenze
- Semplificazione delle precedenze, determinazione del grado di parallelismo, definizione delle funzioni dei moduli
- Determinatezza del grafo (semplice se la concorrenza avviene tra processi)
- Scelta del linguaggio e strumenti linguistici da utilizzare
- Definizione degli strumenti linguistici $\Rightarrow$ modalitá di sincronizzazione
- Formulazione del problema in thread concorrenti
- Definizione delle precedenze $\Rightarrow$ Grafo delle precedenze
- Semplificazione del grafo
- Verifica delle condizioni di determinatezza
- Realizzazione del multithreading (scelta dello strumento linguistico)

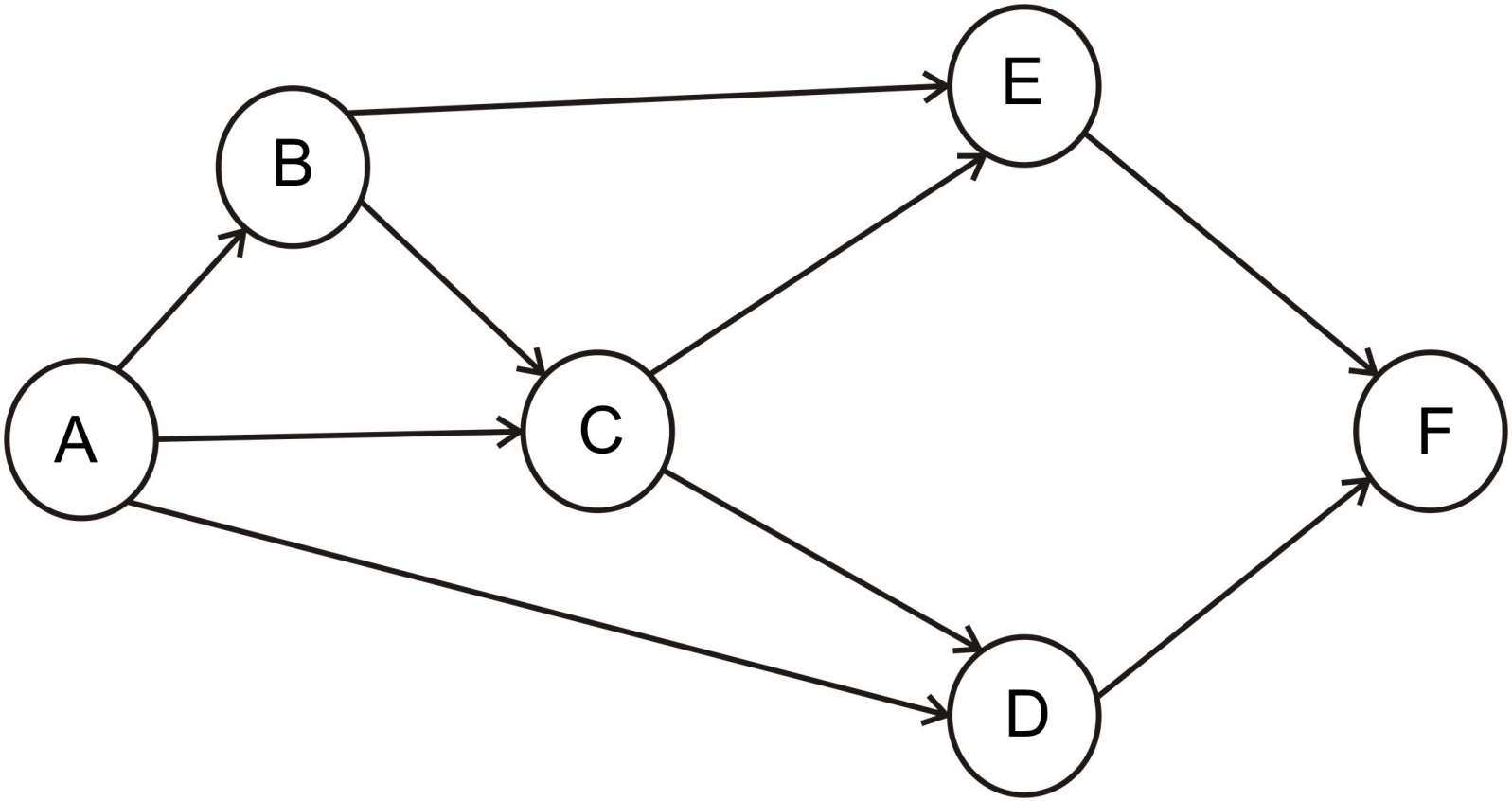
Esempio

- Supponiamo di voler realizzare il seguente grafo:



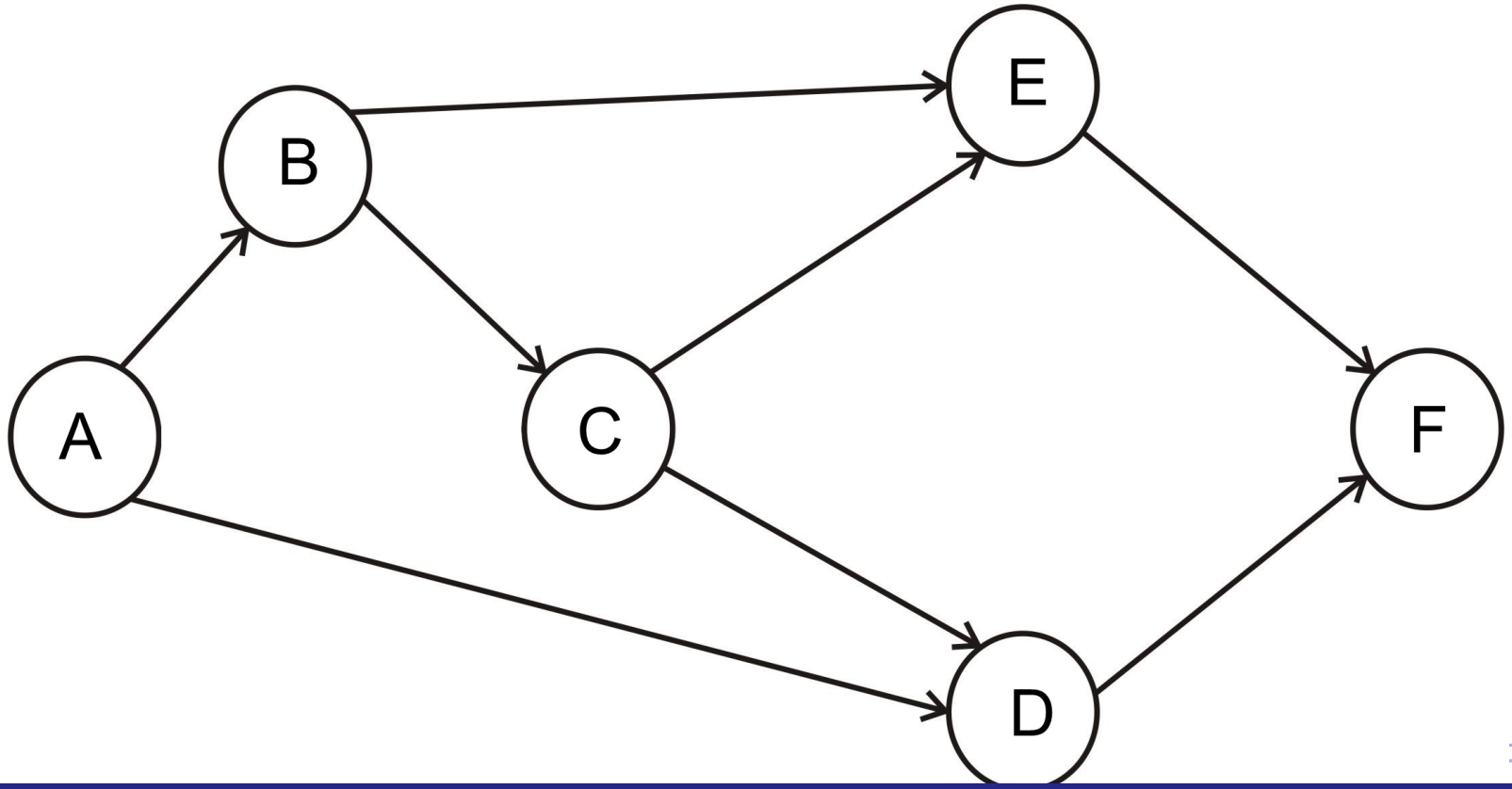
- Unitá computazionali:
  - T0: inizializza le variabili a, b, c, d, e, f
  - T1: $t1 = a + b$
  - T2: $t2 = c + d$
  - T3: $t3 = e/f$
  - T4: $t4 = t1 * t2$
  - T5: $t5 = t4 - t3$
- Il grafo calcola la funzione $(a + b)(c + d) - e/f$

Esempio di grafo da semplificare

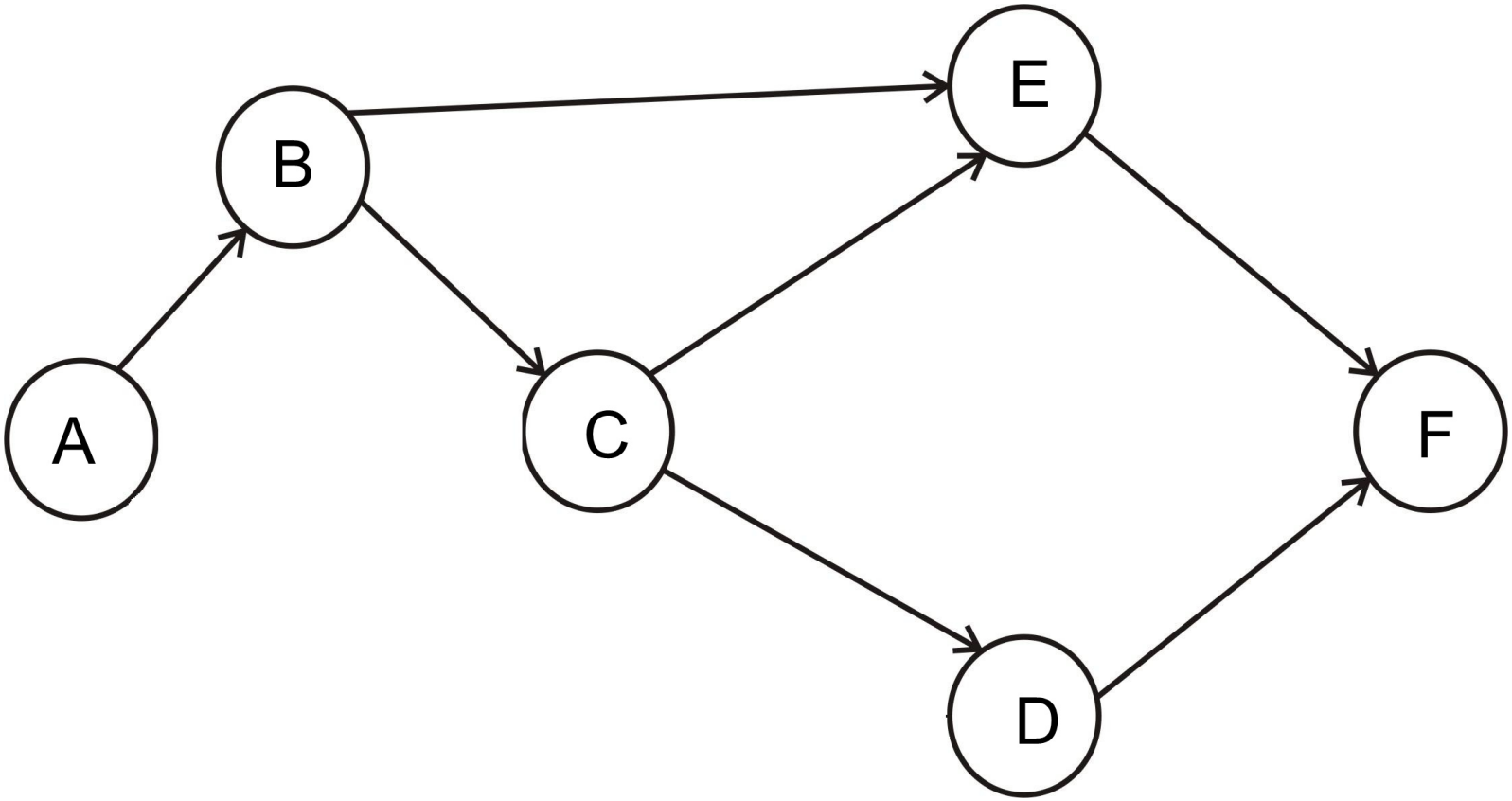


Semplificazione del grafo: eliminazione di precedenze impliciten

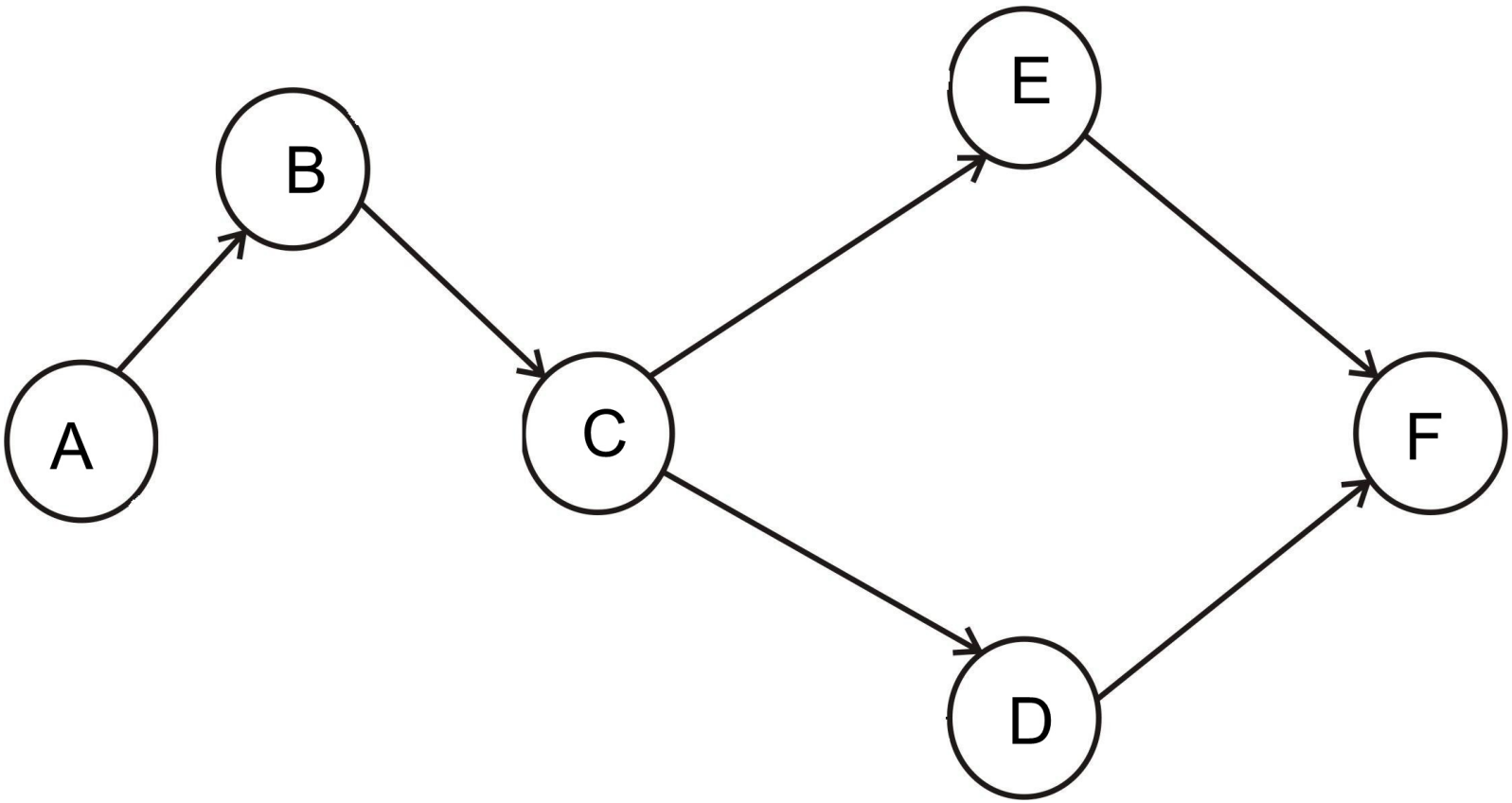
Esempio di grafo da semplificare



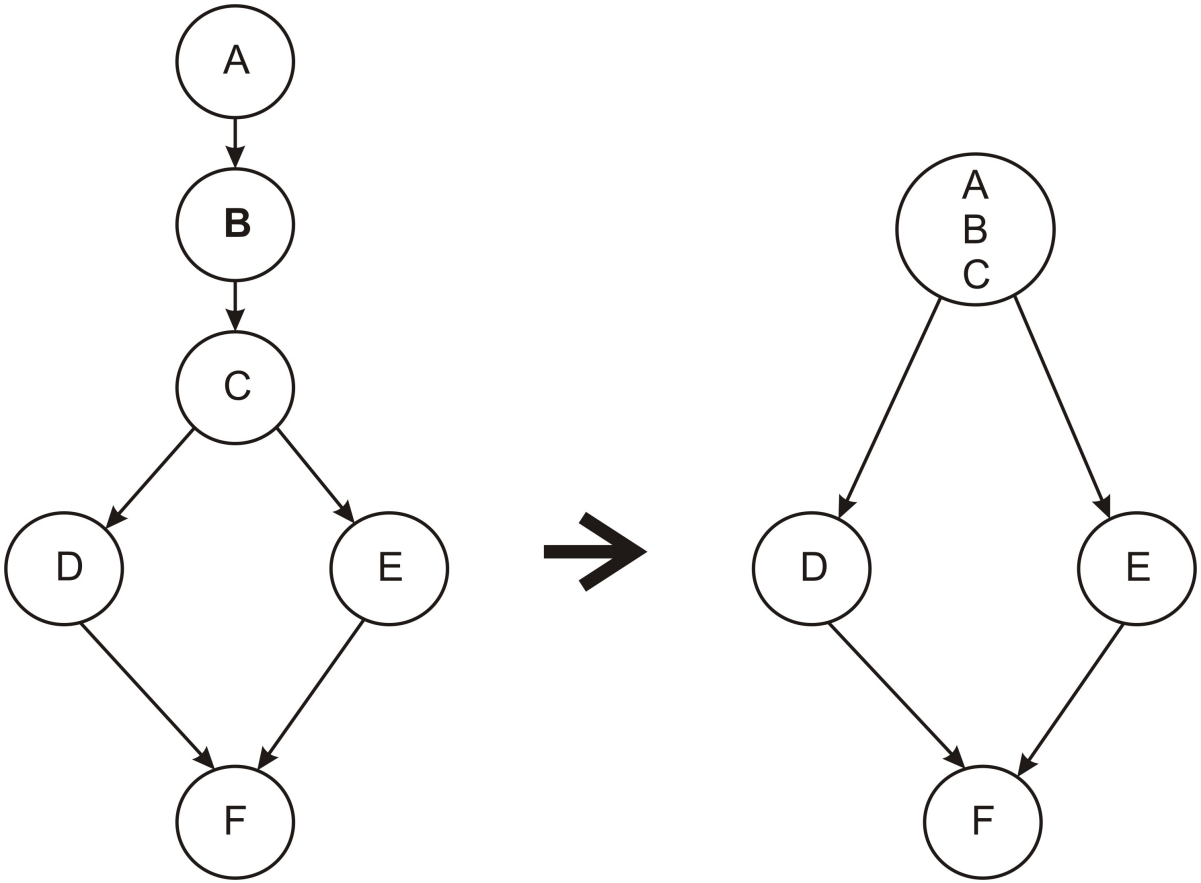
Esempio di grafo da semplificare



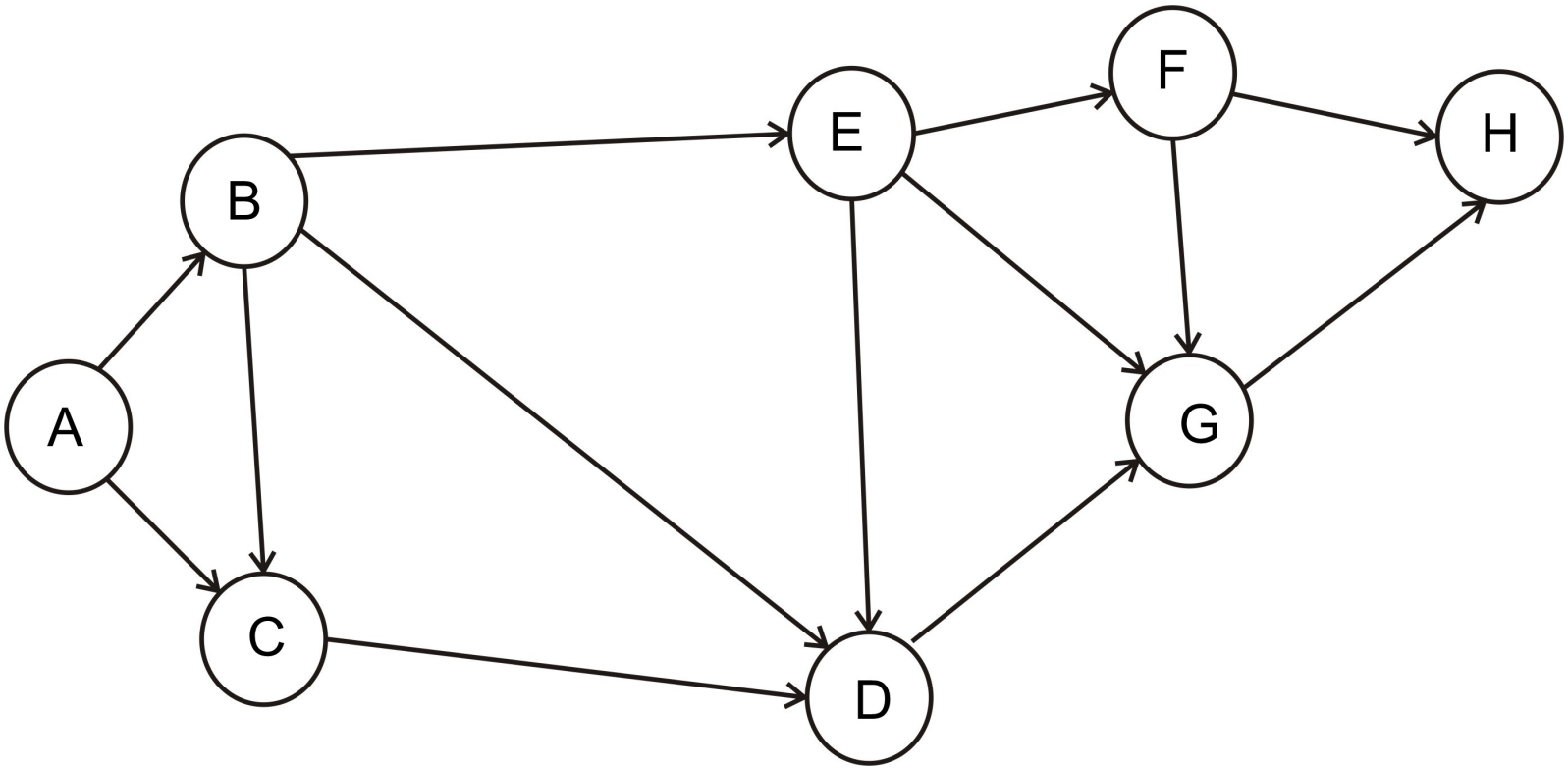
Esempio di grafo da semplificare



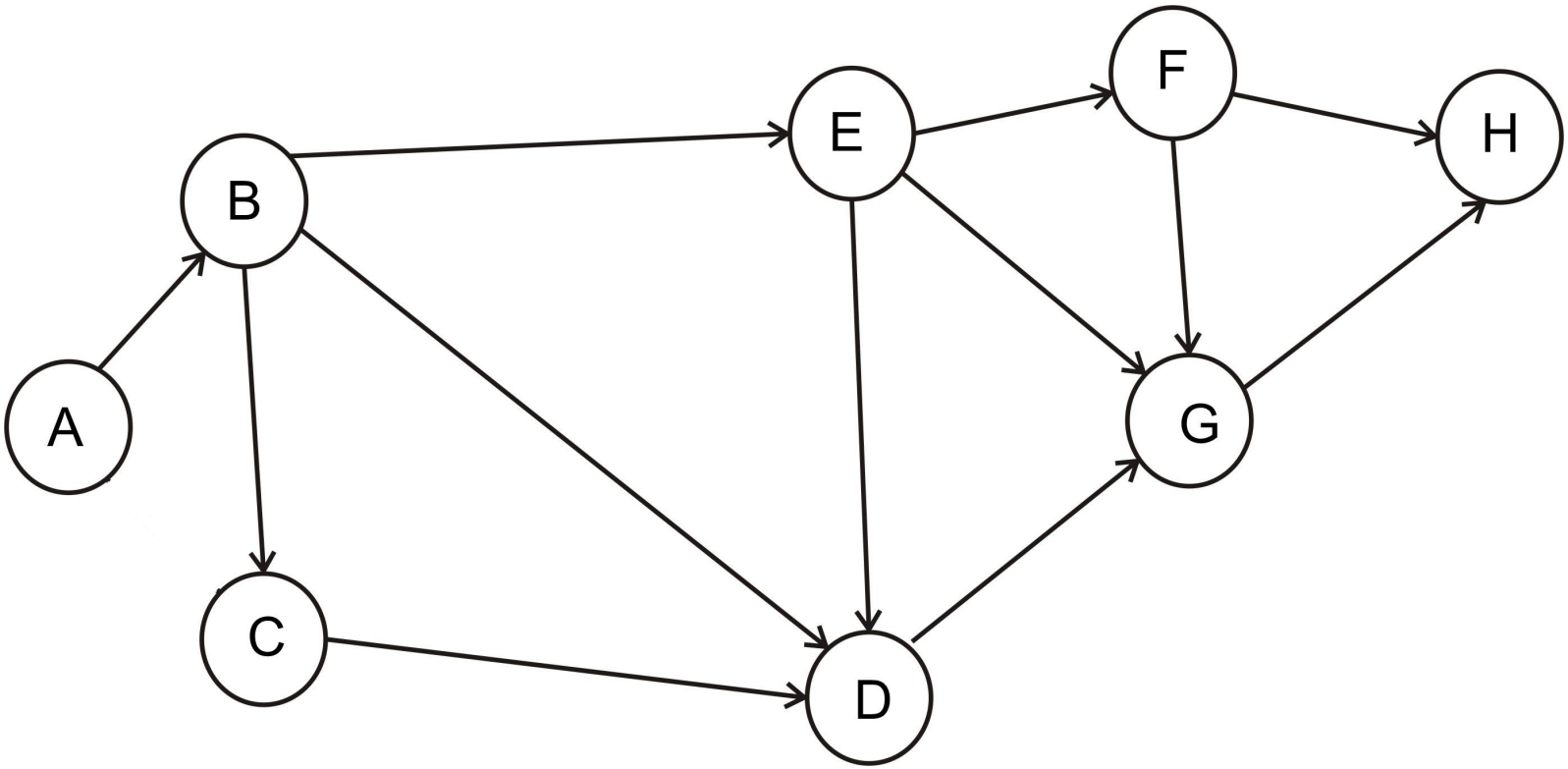
Grafo semplificato



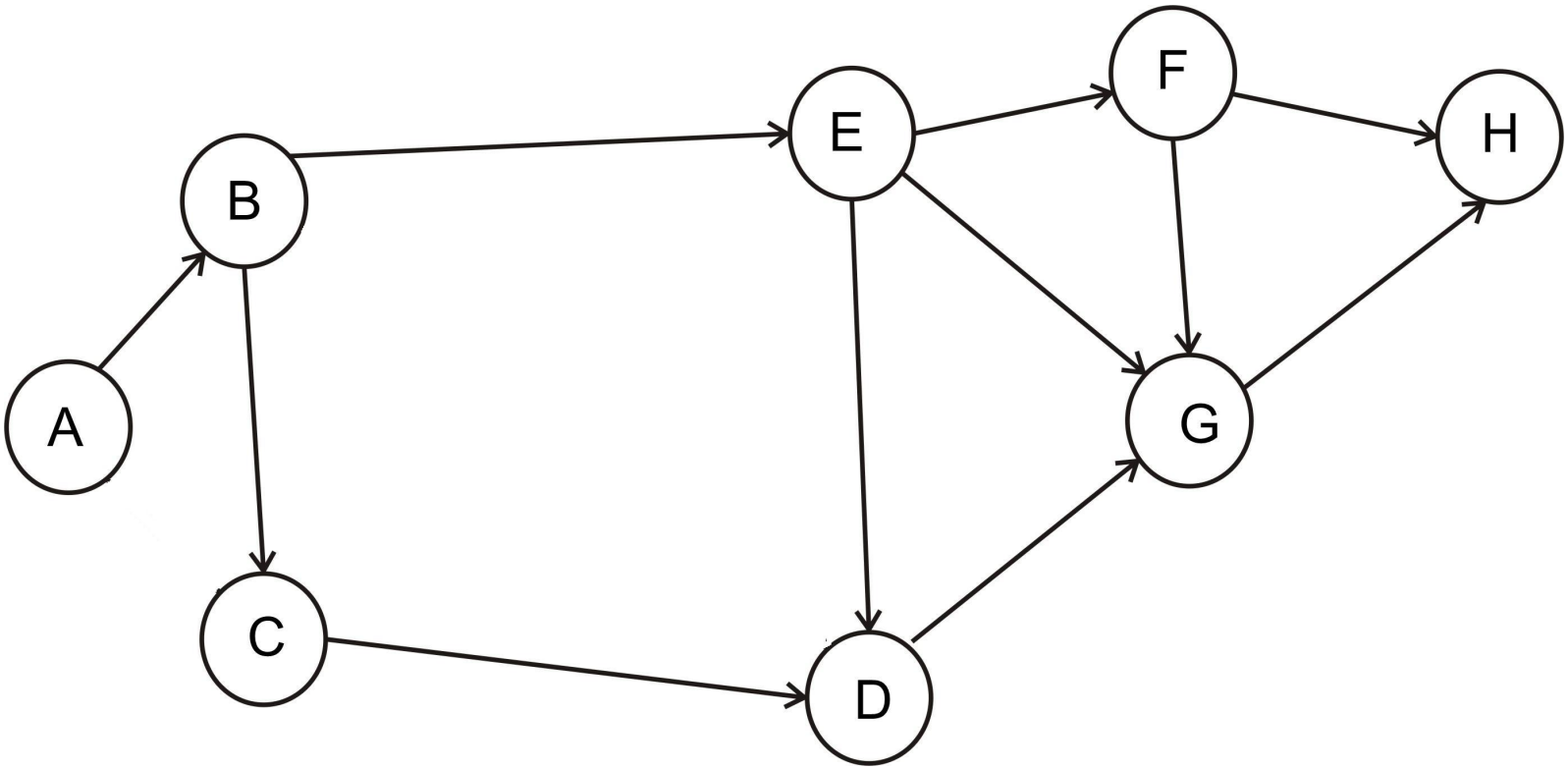
Altro grafo da semplificare



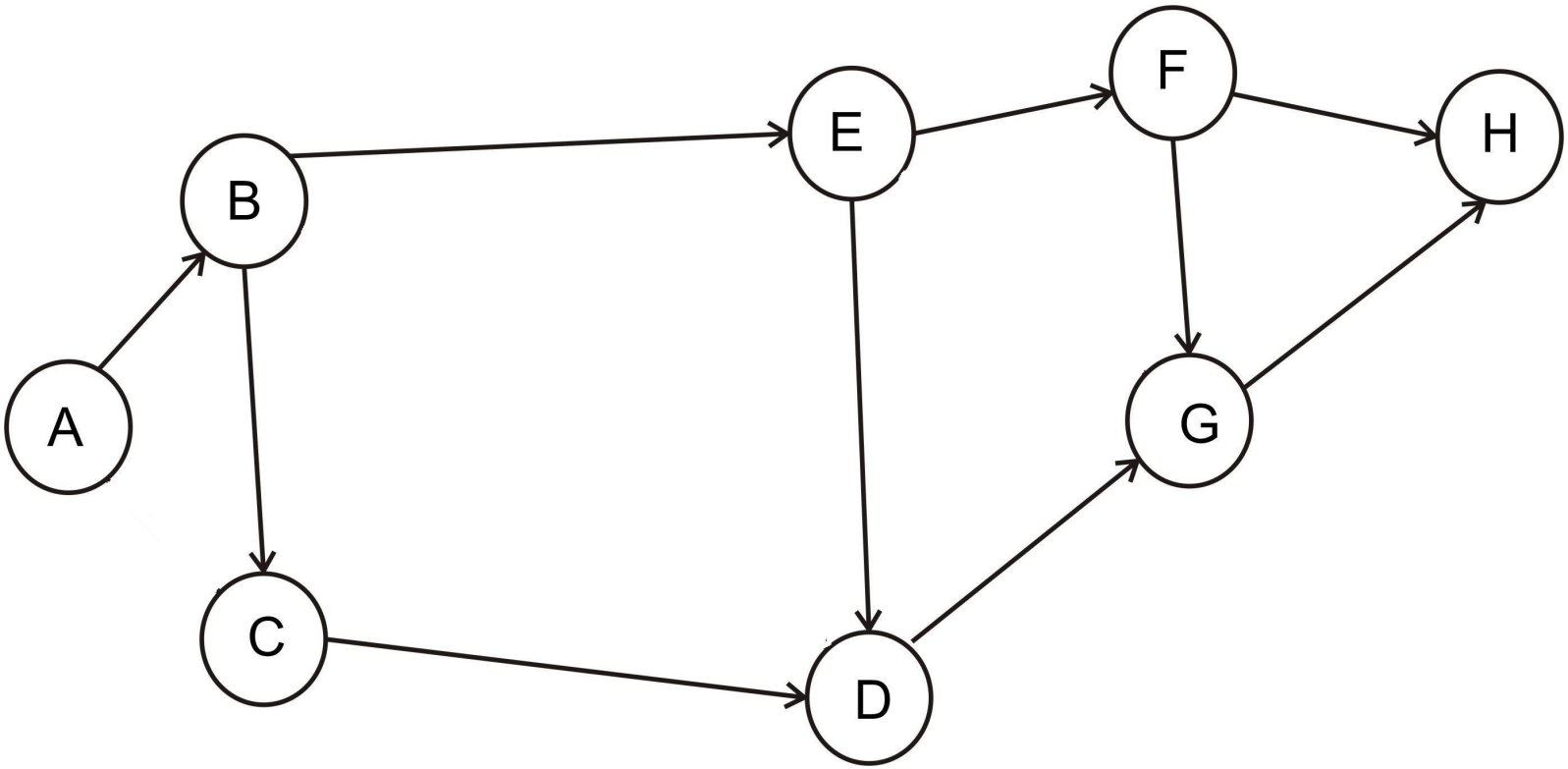
Esempio di grafo da semplificare



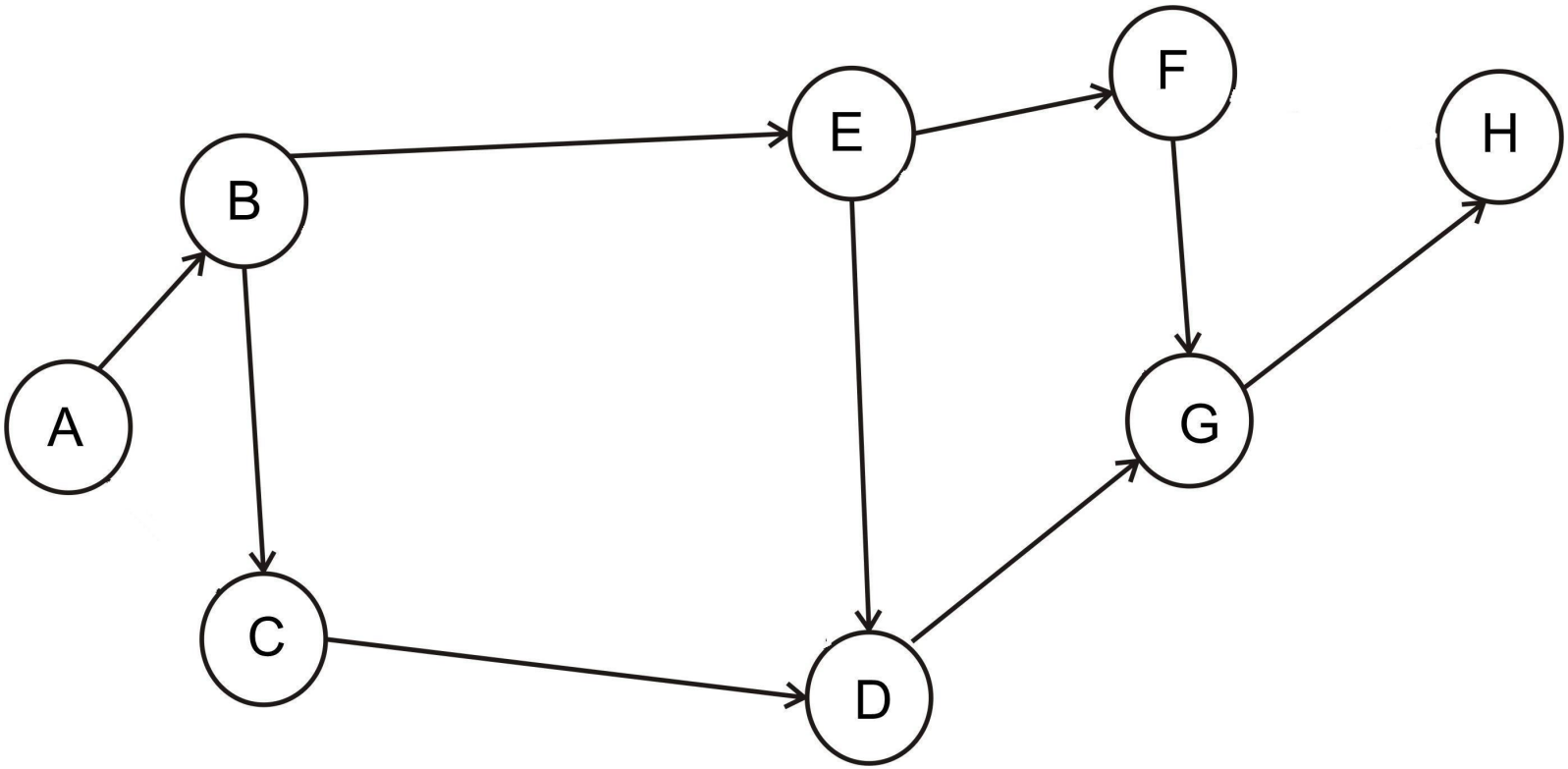
Esempio di grafo da semplificare



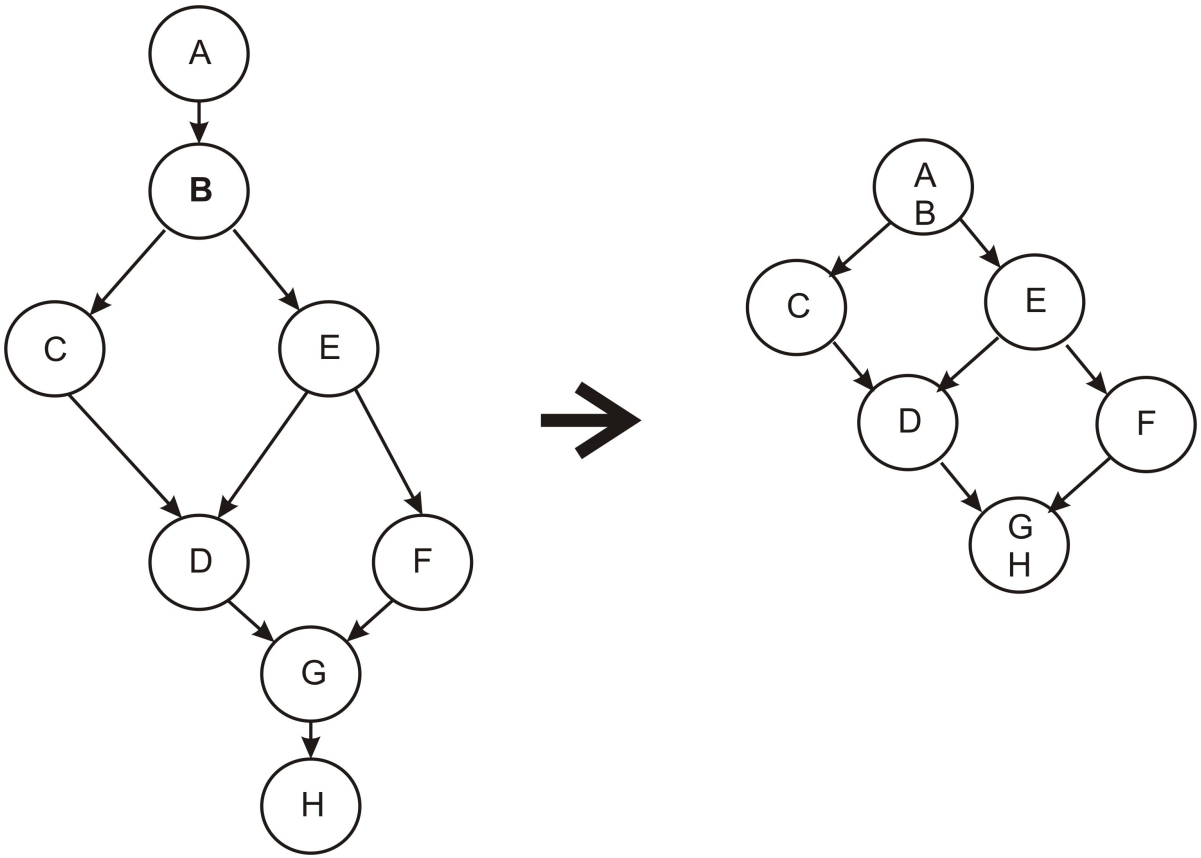
Esempio di grafo da semplificare



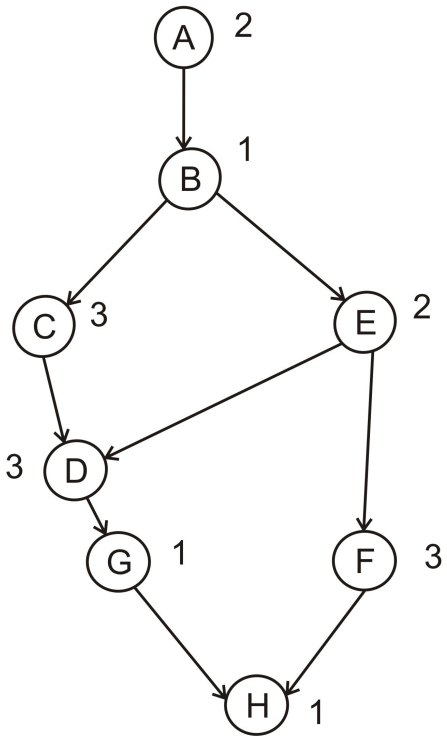
Esempio di grafo da semplificare

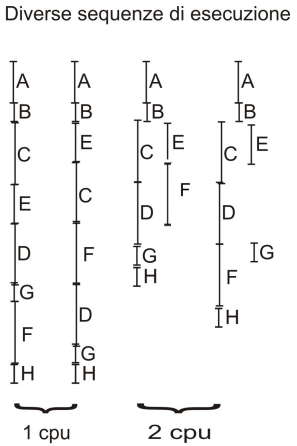


Grafo semplificato



Tempo di calcolo





- una CPU –> tempo di calcolo uguale per ogni sequenza di esecuzione
- piú CPU –> tempo di calcolo dipende dalla schedulazione

Realizzazione della concorrenza 1: modello cobegin-coend

- La concorrenza realizzata con 'cobegin'.
- La sincronizzazione sulla fine della esecuzione con 'coend'

```
cobegin     P1(); P2(); P3();    coend;
```

Che grafico delle precedenze implementa questo codice?

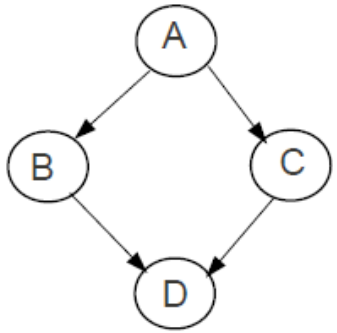
```
begin
  A();
  cobegin
       begin C(); F();  end;
       begin
          B();
          cobegin
             D(); E();
          coend;
       end;
   coend;
   G();
end;
```

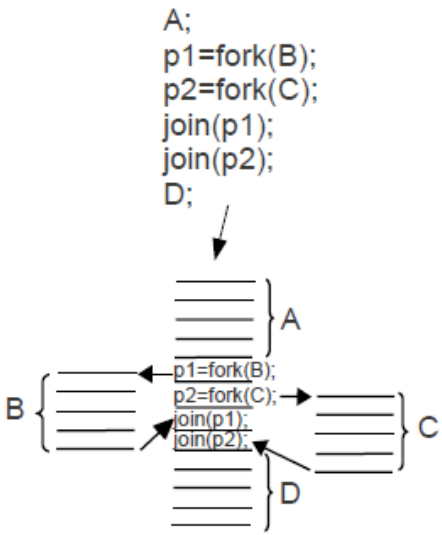
Pseudocodice implementazione con cobegin/coend

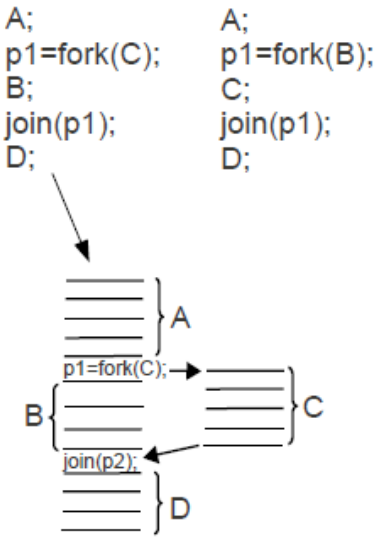
```
cobegin
   begin
      cobegin
         T1(); T2();
      coend
      T4();
   end
   T3();
coend
```

Realizzazione della concorrenza 2: modello fork-join

Concorrenza realizzata con Fork/Join: Join aspetta la terminazione del thread: punto di sincronizzazione



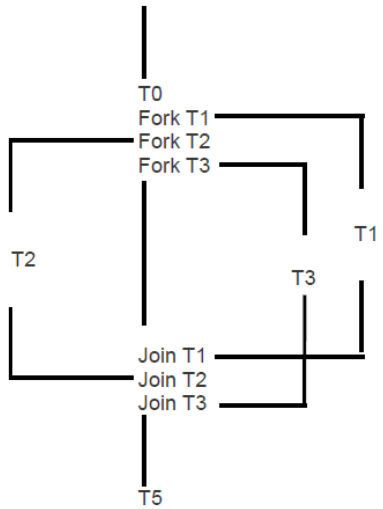




Pseudocodice implementazione con fork/join

```
altra possibilita':                T0();
                                   fork(T1);
fork(processo)          --->       fork(T2);
join(processo                      fork(T3);
                                   join(T1);
                                   join(T2);
                                   join(T3);
                                   T5();
```

Questa implementazione segue lo schema:



Determinatezza: definizioni

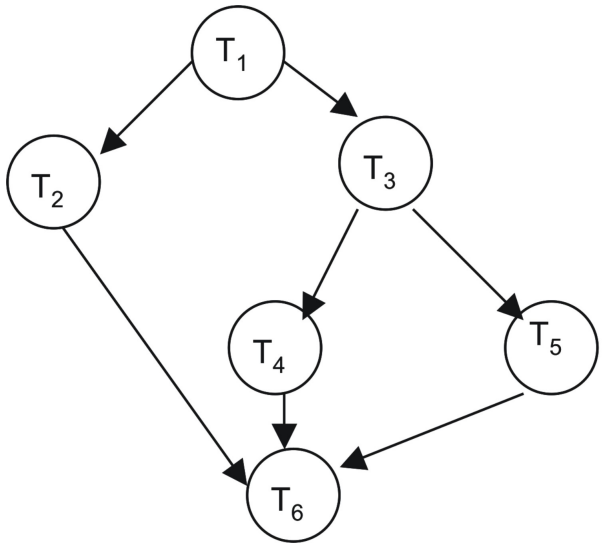
Thread piú efficienti ma possono avere problemi di DETERMINATEZZA.

- insieme di thread: $\Gamma = \{T_1, \ldots, T_n\}$
- eventi fondamentali associati ad un thread: inizio $\overline{T}$ e terminazione $\underline{T}$. $t(\underline{T}) - t(\overline{T}) > 0$ dove $t(.)$ é il tempo del thread
- relazione d'ordine parziale su Γ: $T \prec T'$
- sistema di thread: $C = (\Gamma, \prec)$
- grafo di precedenza: (Γ, archi orientati che rappresentano i vincoli).
- L'arco (T, T') da T a $T' \in$ grafo se e solo se $T \prec T'$ e non esiste
- nodi predecessori, successori;
- nodi immediati predecessori, immediati successori; nodi iniziali e terminali
- nodi indipendenti. T e T' possono essere concorrenti se e solo se indipendenti.
- **Sistema di thread chiuso**
- **Concatenazione, Combinazione parallela** di sistemi di thread

Determinatezza: definizioni

- sequenza di esecuzione α di un sistema di n thread
- Per ogni T in Γ i simboli $\underline{T}$ e $\overline{T}$ appaiono esattamente una volta in α.
- Se $a_i = \overline{T}$ e $a_j = \underline{T}$ allora $i < j$.
- Se $a_i = \underline{T}$ e $a_j = \overline{T}$, dove T $\prec$ T' allora $i < j$.

Esempio: sequenze di esecuzione valide per



possono essere

$$\overline{T_1}\,\underline{T_1}\,\overline{T_2}\,\underline{T_2}\,\overline{T_3}\,\underline{T_3}\,\overline{T_4}\,\underline{T_4}\,\overline{T_5}\,\underline{T_5}\,\overline{T_6}\,\underline{T_6}$$

ma anche

$$\overline{T_1}\,\underline{T_1}\,\overline{T_2}\,\overline{T_3}\,\underline{T_3}\,\overline{T_4}\,\overline{T_5}\,\underline{T_2}\,\underline{T_4}\,\underline{T_5}\,\overline{T_6}\,\underline{T_6}$$

Determinatezza di un sistema di thread

- Chiamamo $V(M_i, \alpha)$ la sequenza di valori scritti nella cella M_i dalla sequenza di esecuzione α.
- Un sistema di thread C é **determinato** se per ogni s_0, $V(M_i, \alpha) = V(M_i, \alpha')\forall i = 1 \dots m$ e per tutte le sequenze di esecuzione.
- Ogni thread ha un dominio D dal quale prende i dati di ingresso e un codominio R nel quale scrive i risultati
- Due thread T e T' sono **noninterferenti** se T é un successore o predecessore di T' oppure se $R_T \cap R_{T'} = R_T \cap D_{T'} = D_T \cap R_{T'} = 0$.
- Un insieme di Thread é **mutualmente non interferente** se Ti e Tj sono nonintereferenti per tutti gli indici i e j $(i \neq j)$.
- Condizione sufficiente per la determinatezza di un sistema di n thread é che il sistema di thread sia mutualmente non interferente.