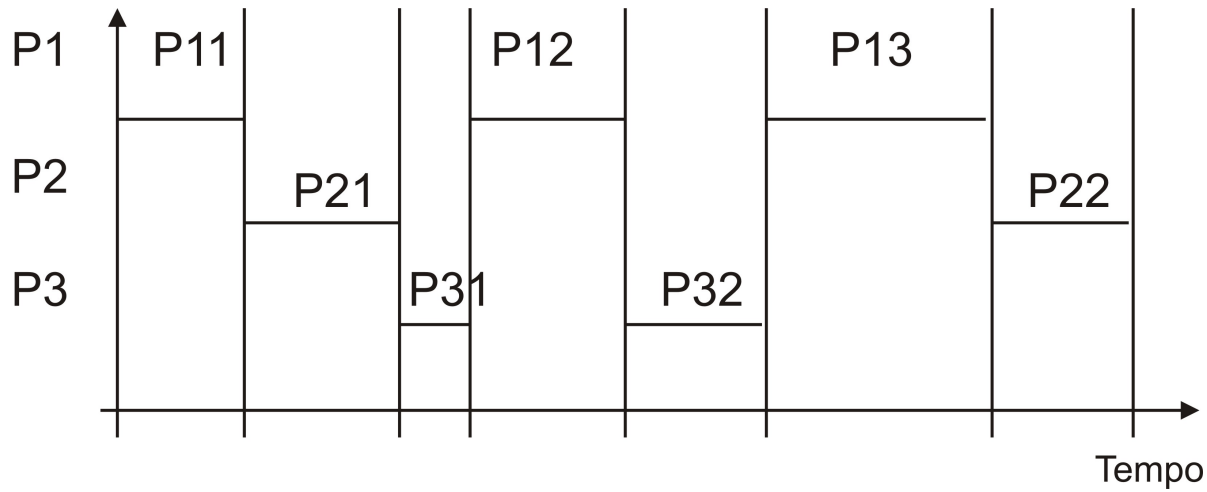
Introduzione alla concorrenza

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

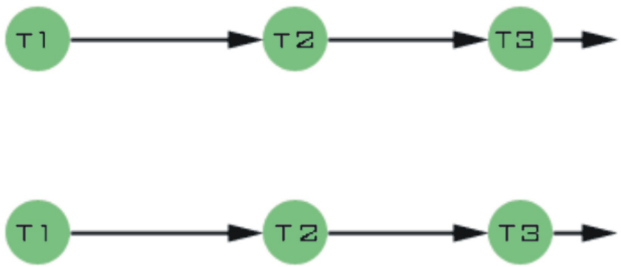
– Caratteristica principale: condivisione dell'ambiente tra i diversi thread

– Necessitá principale: sincronizzare i thread!

Grafi delle precedenze

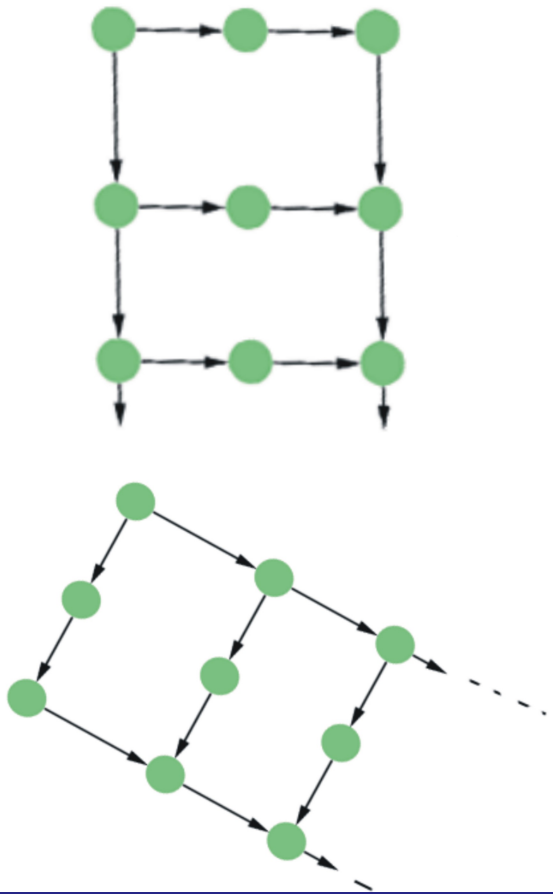
Esempio. Calcolo sequenziale di tre processi: uno legge da un nastro, uno elabora e uno scrive su un nastro

Grafo dei tre processi sequenziali



Grafi delle precedenze

Grafo dei threads concorrenti delle tre elaborazioni

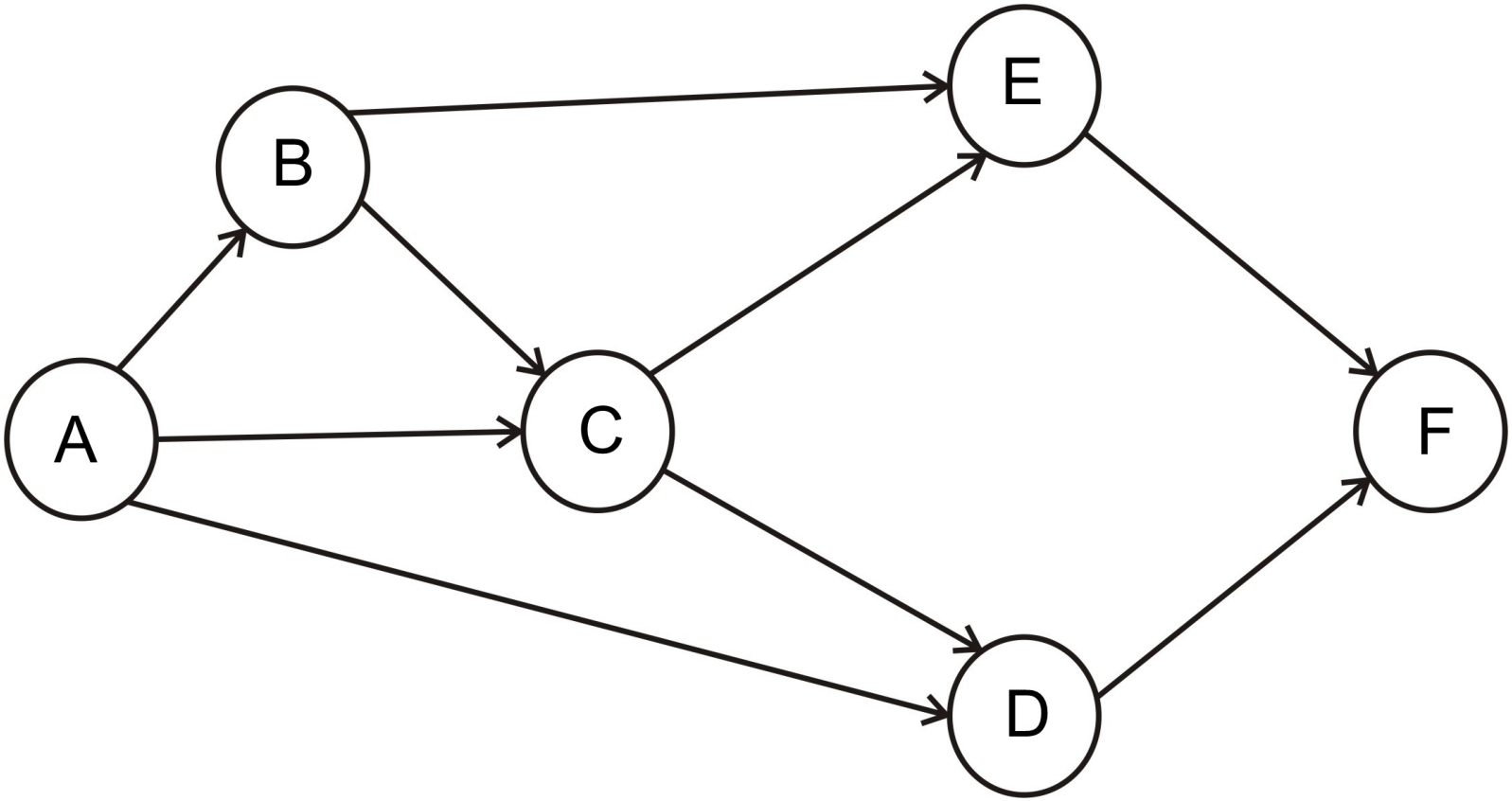


Grafi delle precedenze

Perché sono importanti?
Tipici utilizzi:

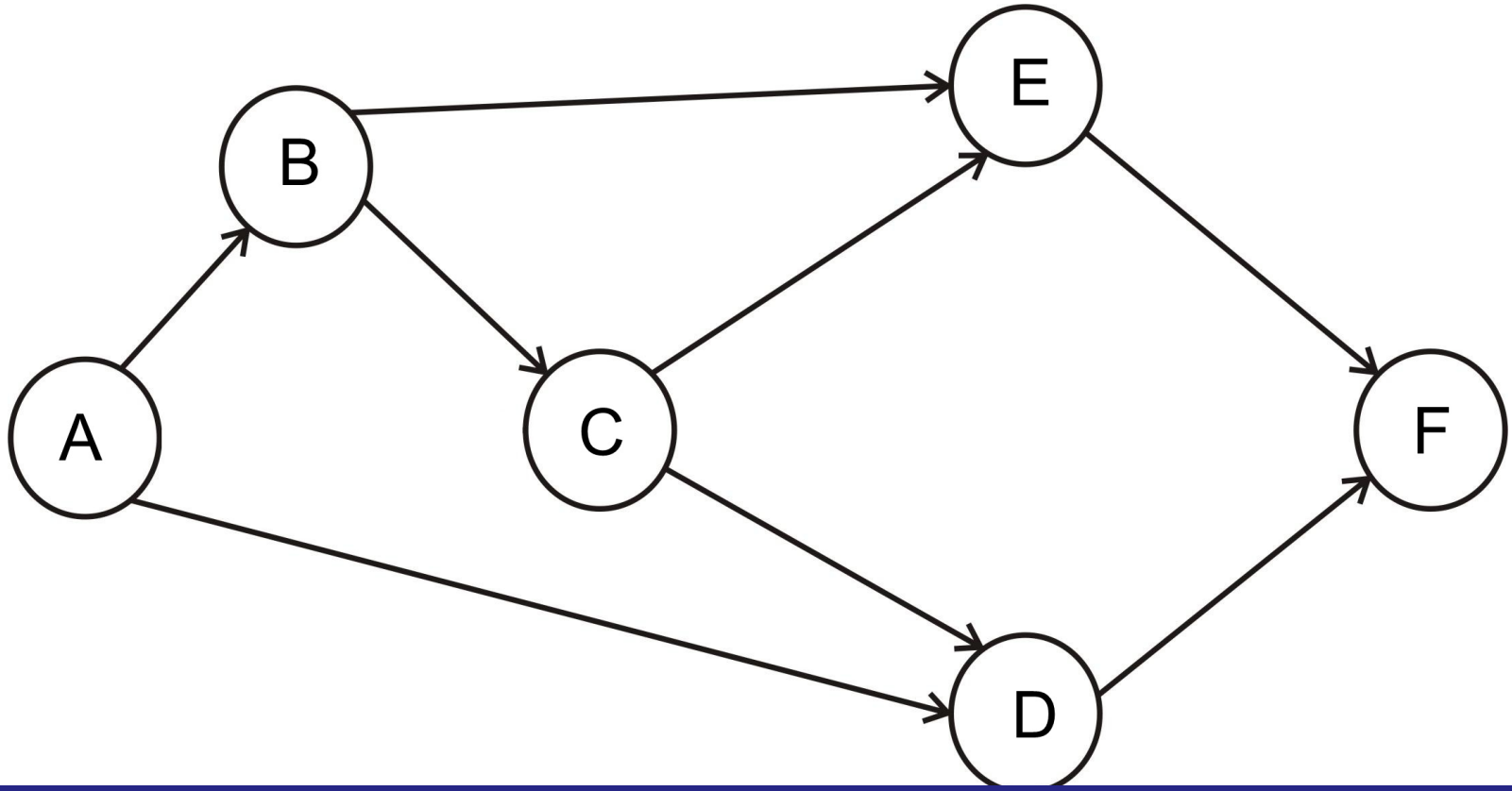
- modellare e descrivere una applicazione concorrente dividendola tra entitá concorrenti
- semplificare la applicazione eliminando archi ridondanti
- massimizzare il parallelismo
- descrivere il tipo di implementazione
- uso corretto delle variabili assicurando la determinatezza

Esempio di grafo da semplificare

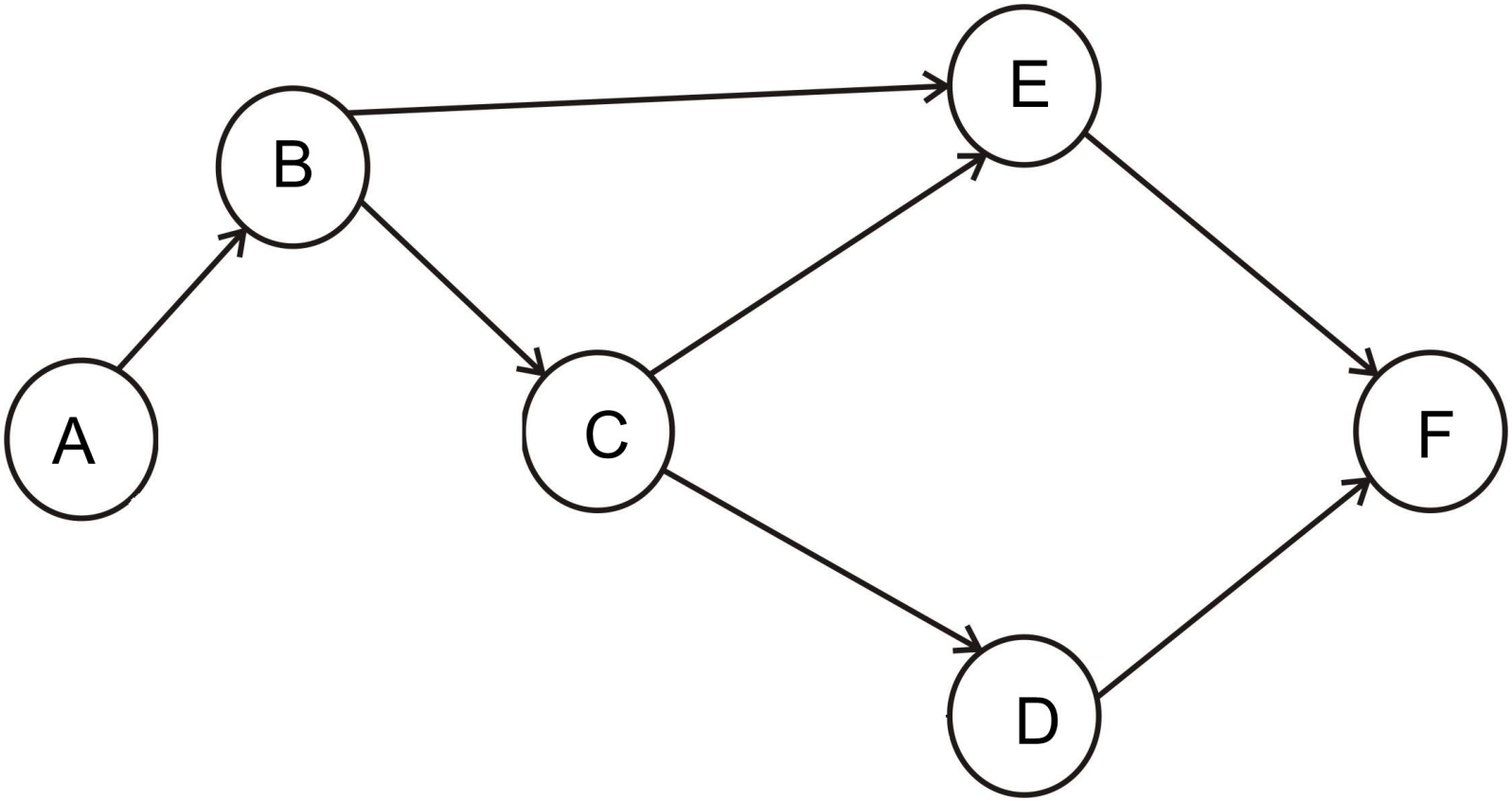


Semplificazione del grafo: eliminazione di precedenze impliciten

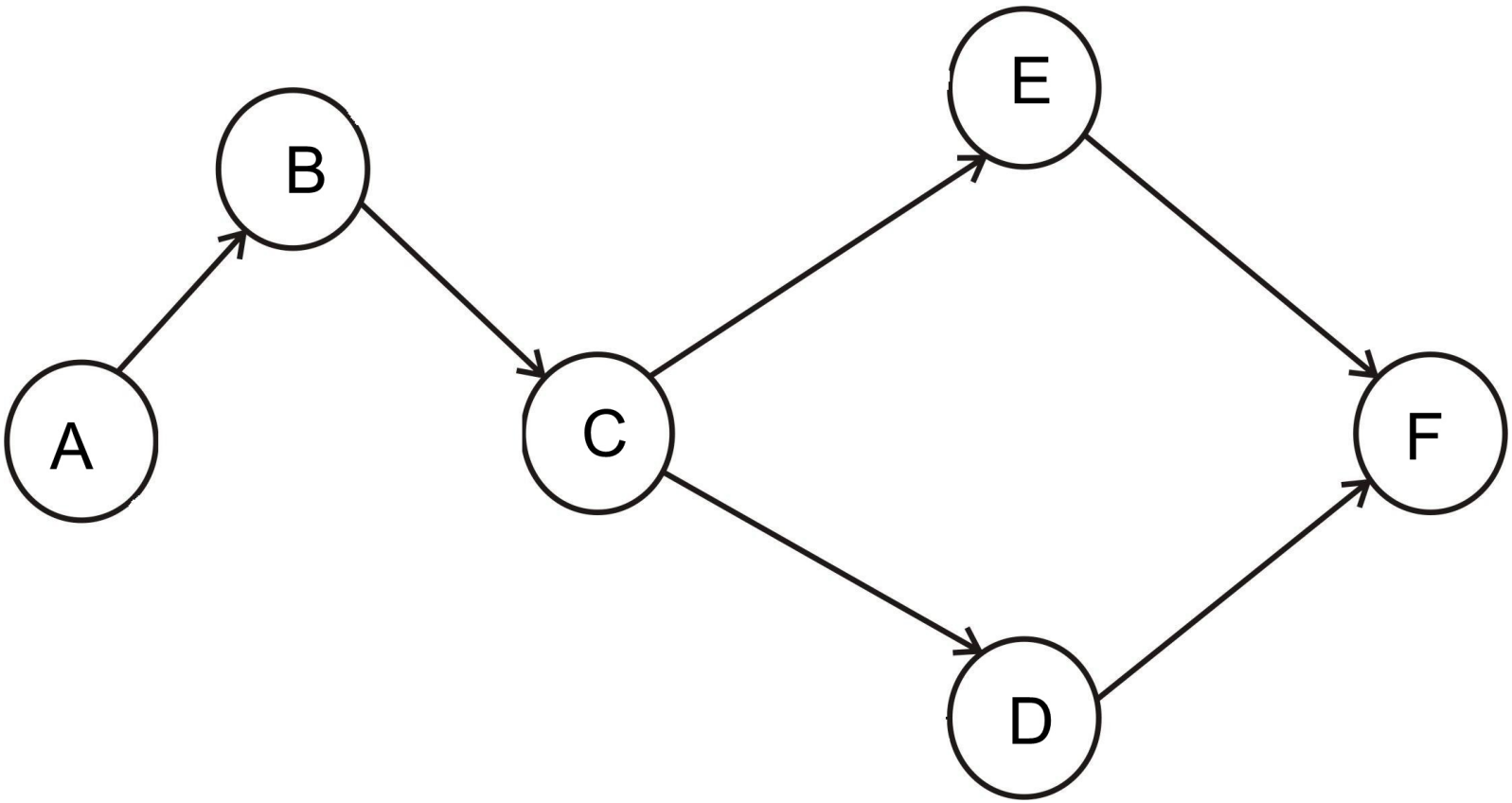
Esempio di grafo da semplificare



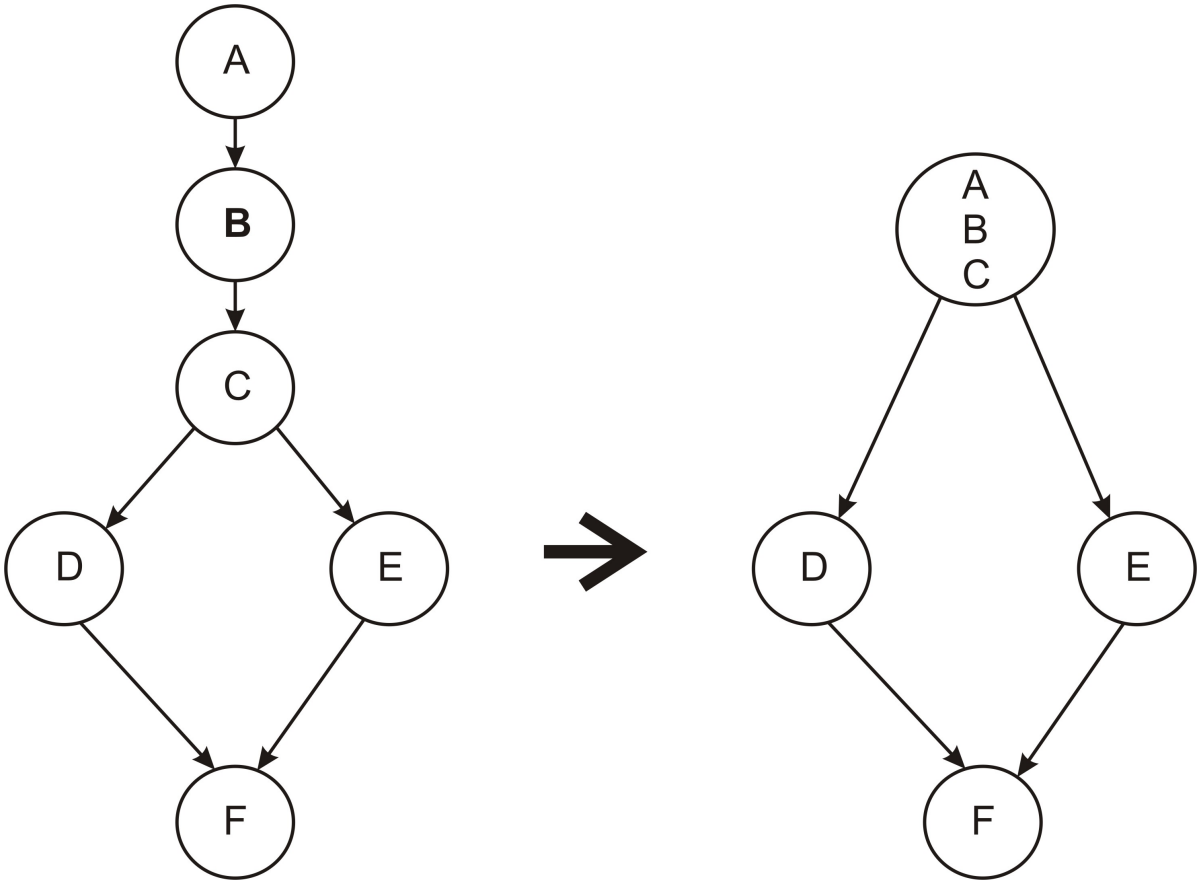
Esempio di grafo da semplificare



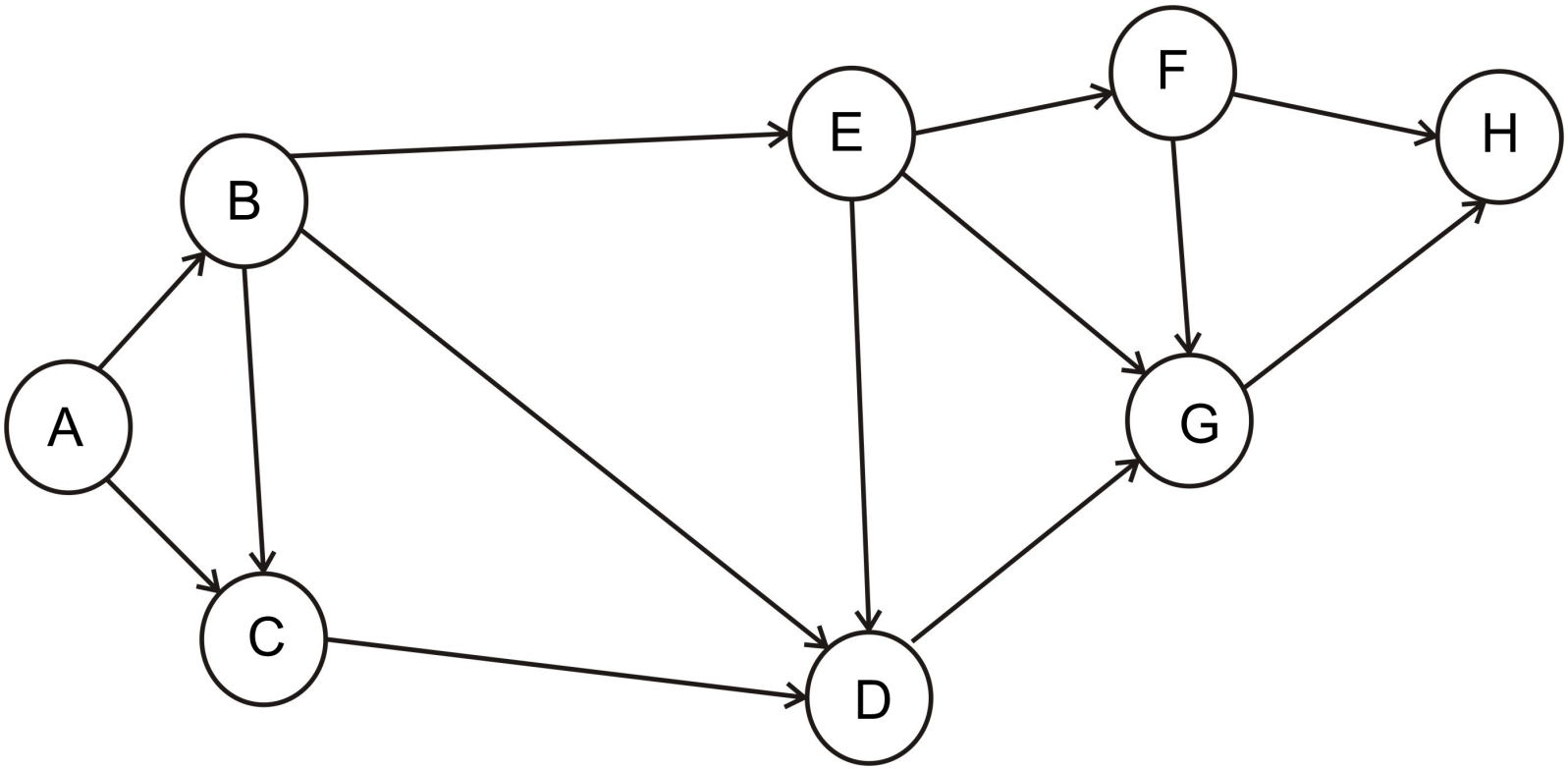
Esempio di grafo da semplificare



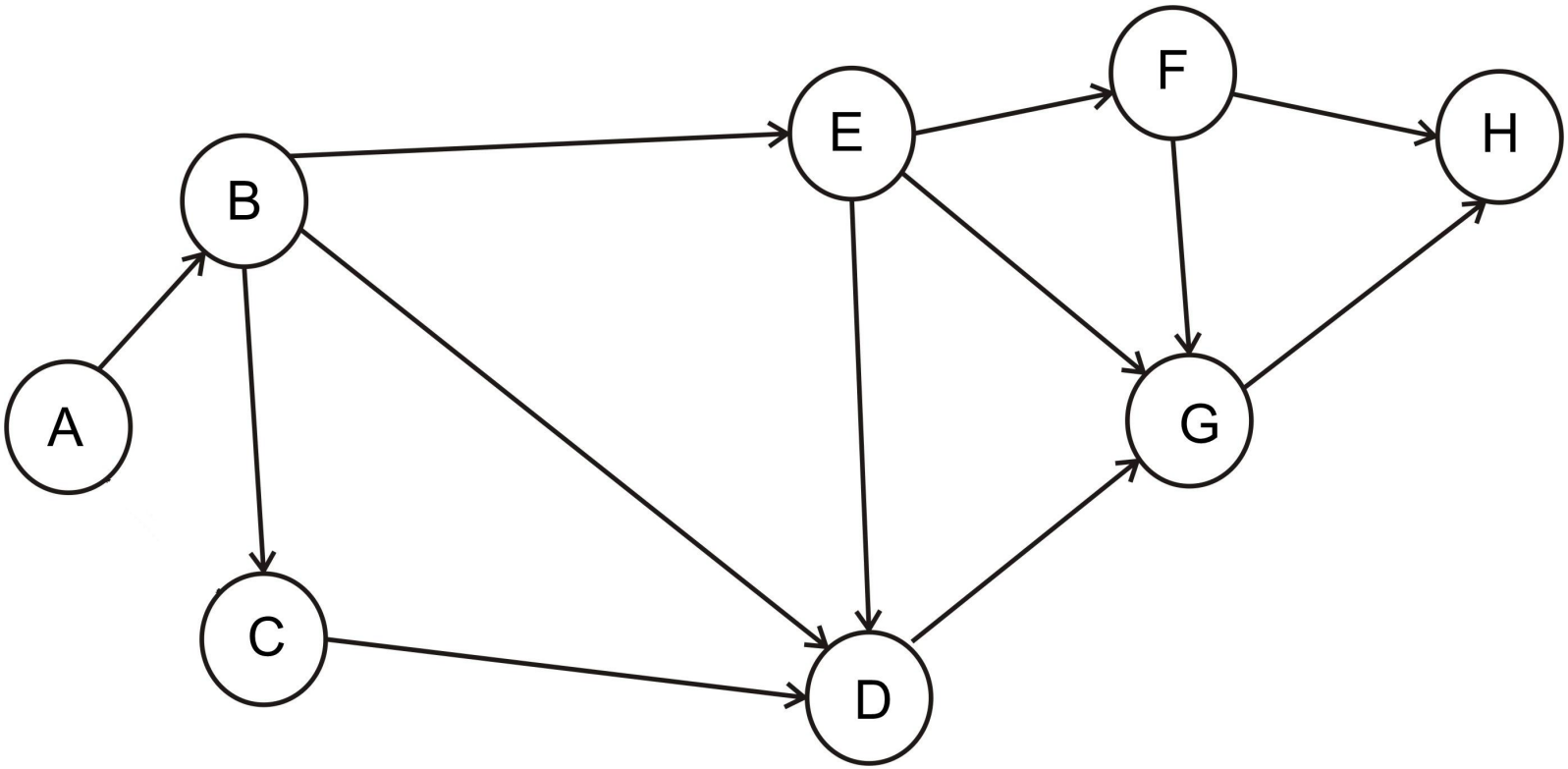
Grafo semplificato



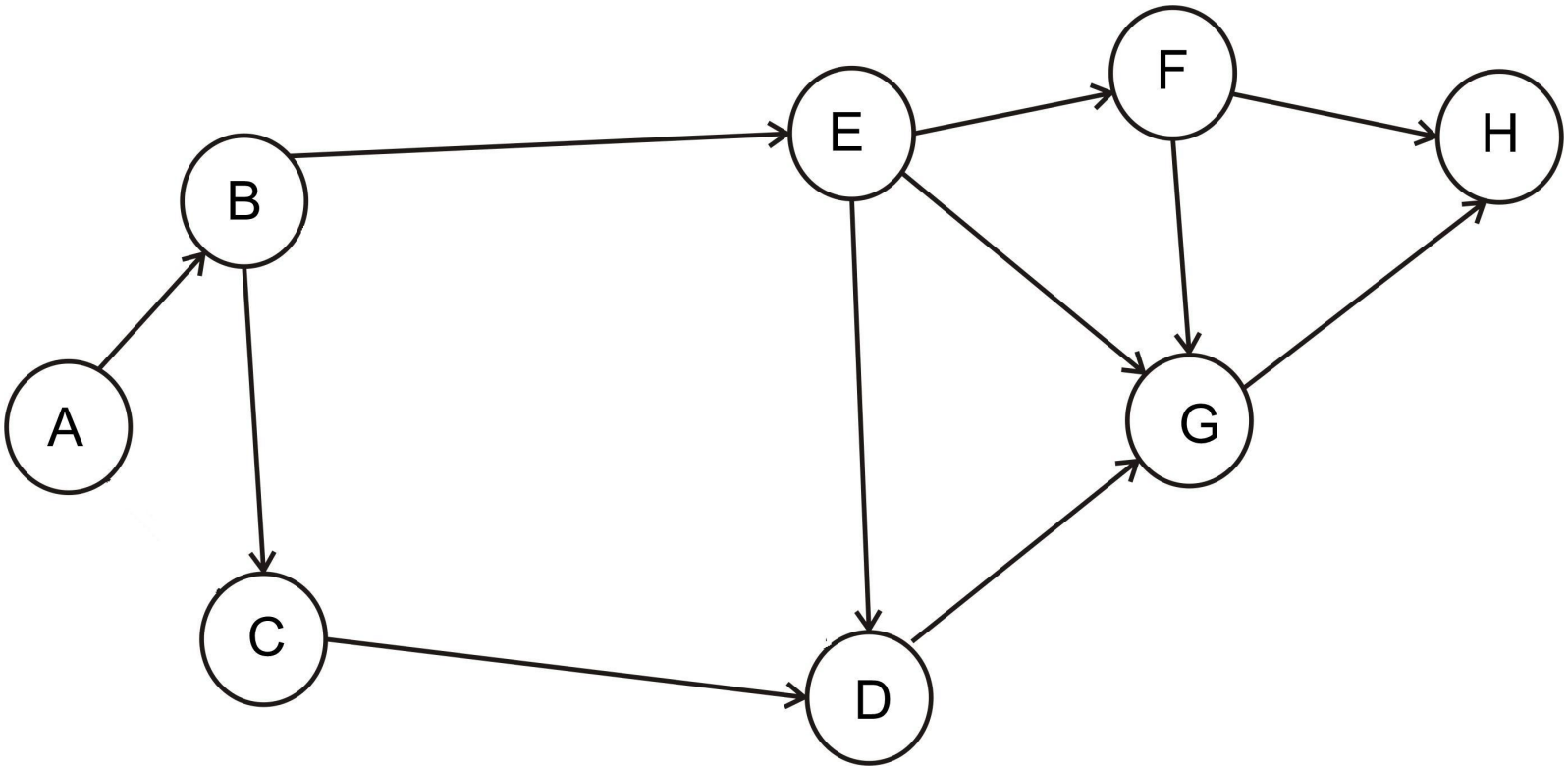
Altro grafo da semplificare



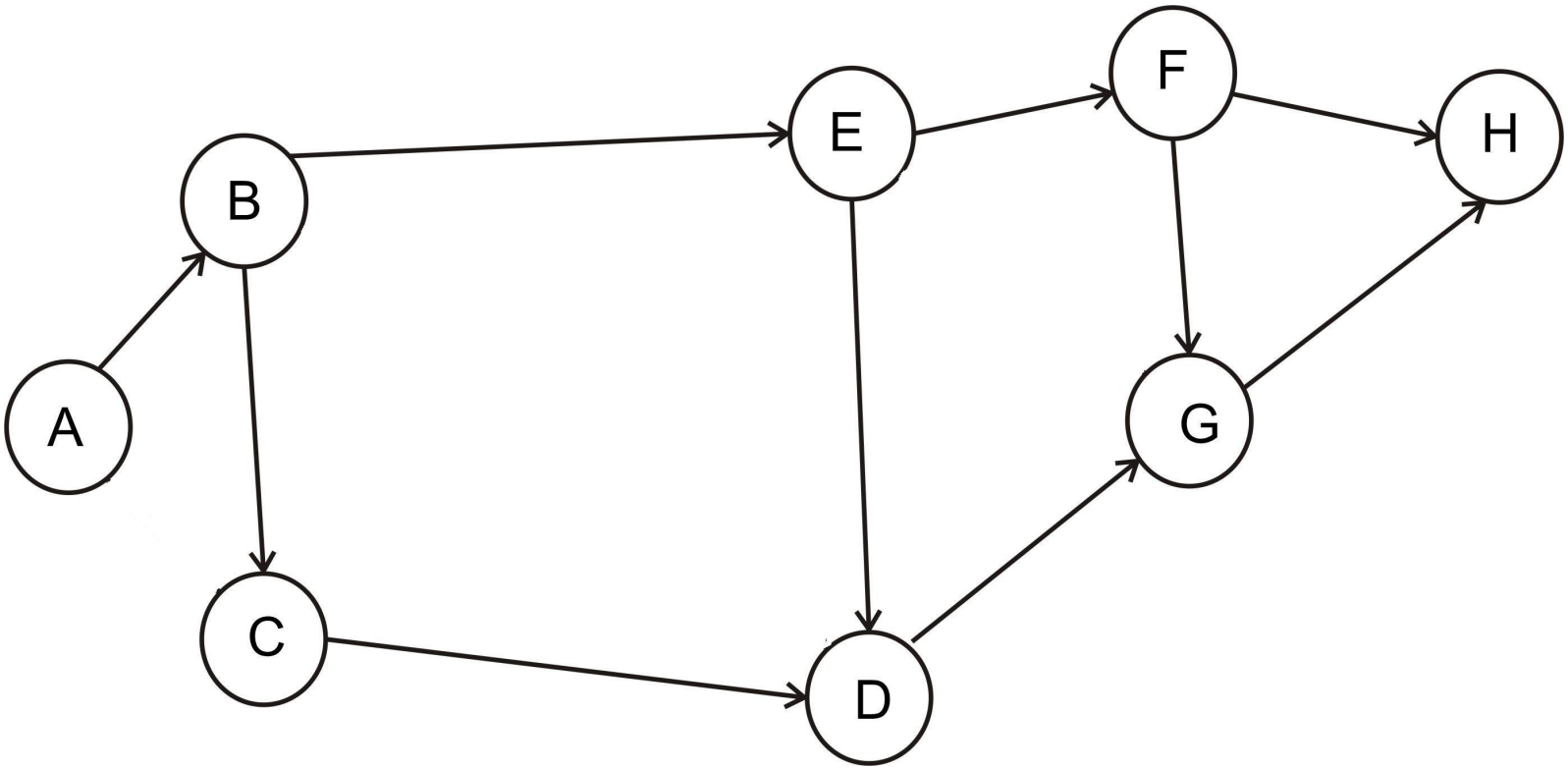
Esempio di grafo da semplificare



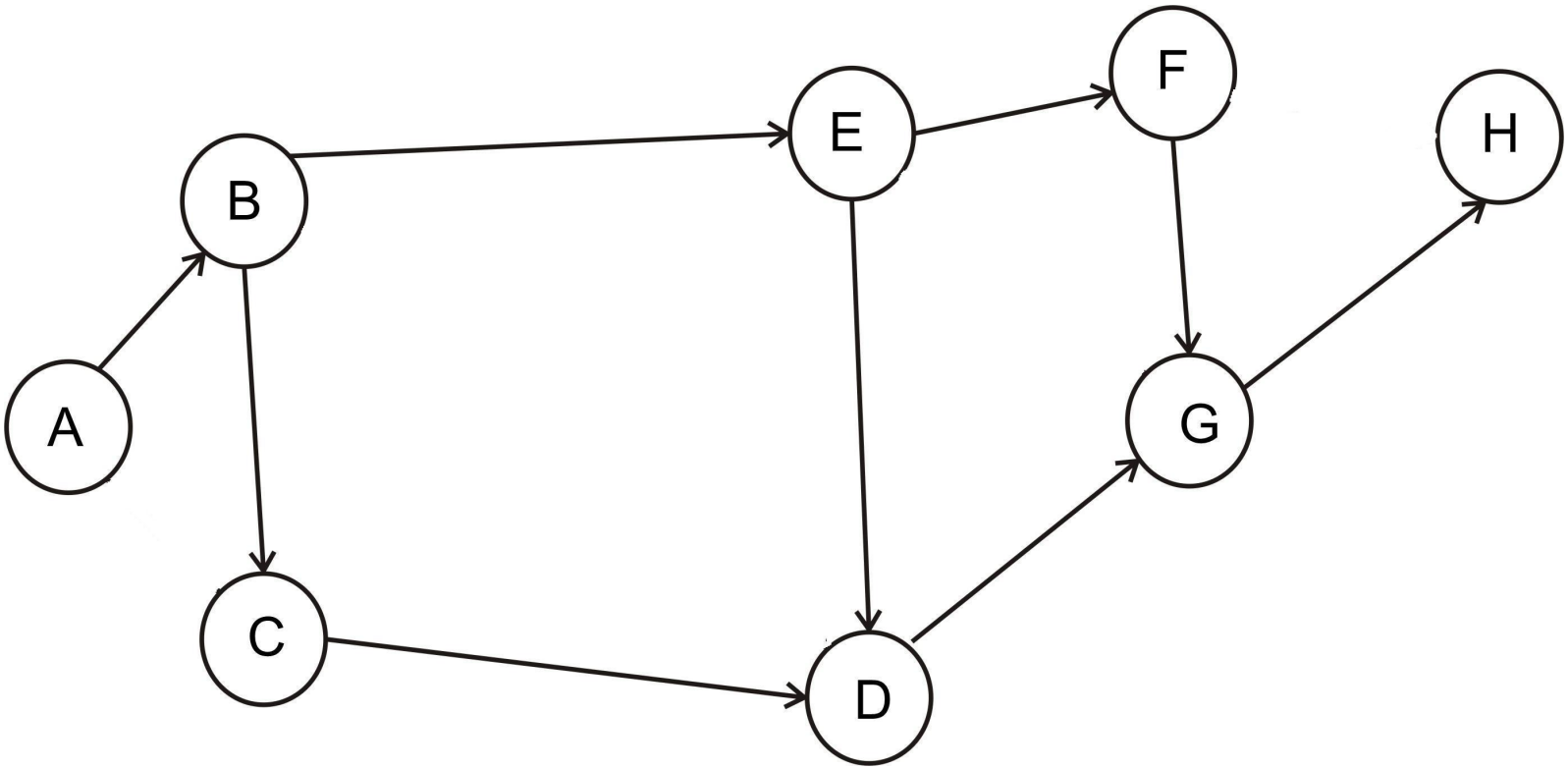
Esempio di grafo da semplificare



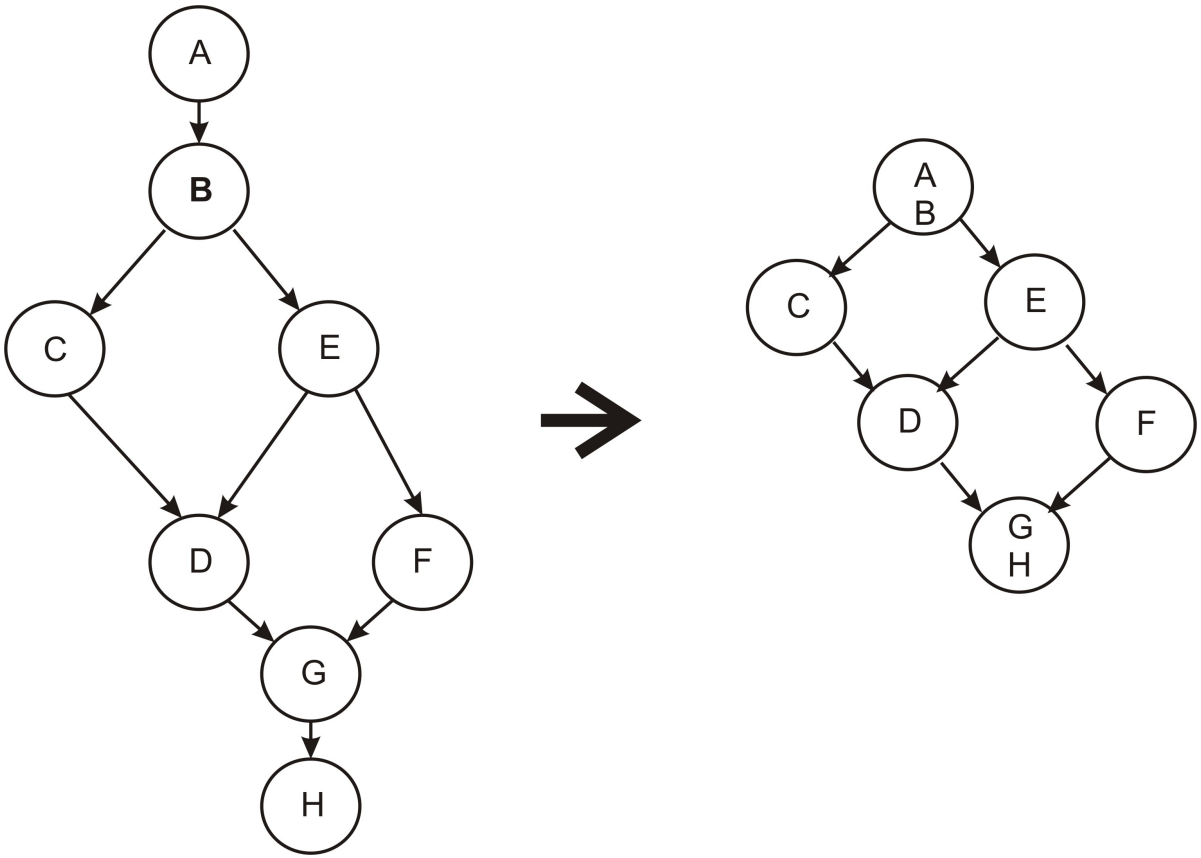
Esempio di grafo da semplificare



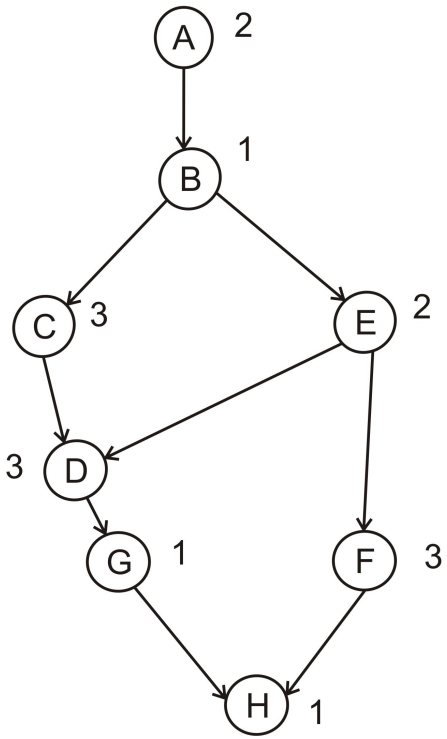
Esempio di grafo da semplificare



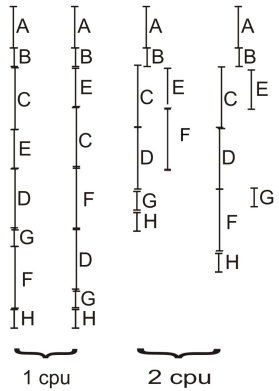
Grafo semplificato



Tempo di calcolo



Diverse sequenze di esecuzione



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

Realizzazione della concorrenza 1: cobegin-coend

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

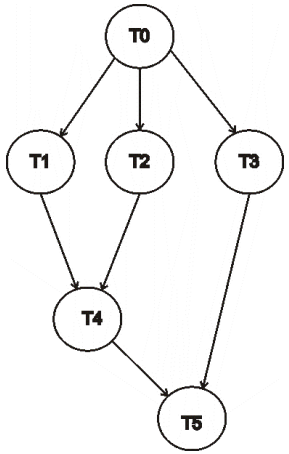
```
cobegin     P1(); P2(); P3();    coend;
```

Che grafico delle precedenze implementa questo codice?

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

Esempio

- Supponiamo di voler realizzare il seguente grafo:



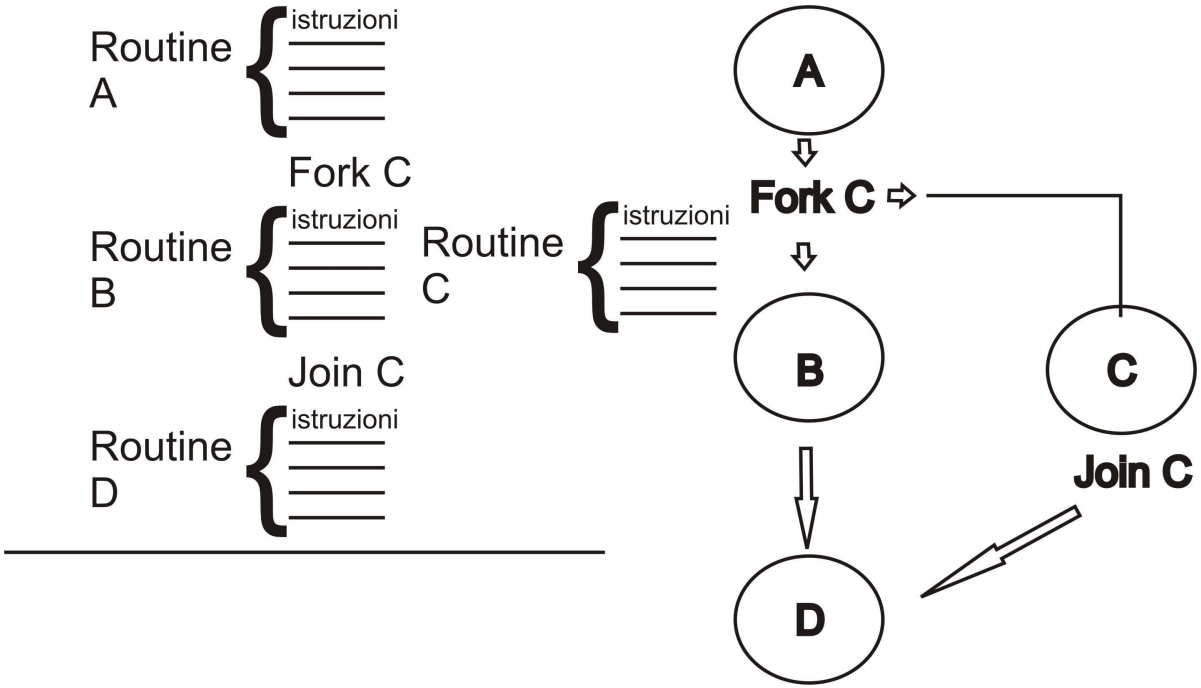
- Unitá computazionali:
  - T0: inizializza le variabili a, b, c, d, e, f
  - T1: $t1 = a + b$
  - T2: $t2 = c + d$
  - T3: $t3 = e/f$
  - T4: $t4 = t1 * t2$
  - T5: $t5 = t4 - t3$
- Il grafo calcola la funzione $(a + b)(c + d) - e/f$

Pseudocodice implementazione con cobegin/coend

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

Realizzazione della concorrenza 2: fork-join

Concorrenza realizzata con Fork/Join

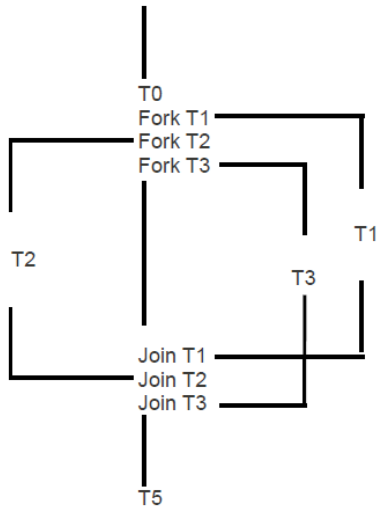


Join é un punto di sincronizzazione: aspetta la terminazione del thread

Pseudocodice implementazione con fork/join

```
T0();
fork(T1);
fork(T2);
fork(T3);
join(T1);
join(T2);
join(T3);
T5();
```

Questa implementazione segue lo schema:



Strumenti base per la creazione dei processi in linguaggio C

- **int fork();**
  Crea un nuovo processo. Al processo padre ritorna il PID del figlio (il padre puo avere piu figli) Al processo figlio ritorna 0 (un processo puo avere 1 solo padre (getppid())
- **void exit();**
  Chiude tutti i descrittori aperti, rilascia memoria.
- **pid_t wait(int *status);**
  blocca il chiamante finche un figlio termina
- **pid_t waitpid(pid_t pid, int *status, int options);**
  blocca il padre finche termina pid
- **int execlp(char *file, char *arg0, ..., char *argN);**
  invoca un altro programma, sovrascrivendo lo spazio di memoria di un processo con una copia di un programma eseguibile. NB: L'INDIRIZZO DI RITORNO E' PERSO!

Creazione/attesa dei processi in Linux

```
t1=fork();
if(t1==0)
   exec(T1);
else{
   t2=fork();
   if(t2==0)
       exec(T2);
   else {
       t3=fork();
       if(t3==0)
           exec(T3);
       else{
           waitpid(t1); waitpid(t2);
           t4=fork();
           if(t4==0)
                exec(T4);
           else{
                waitpid(t4); waitpid(t3);
                t5=fork();
                if(t5==0)
                    exec(T5);
                else{
                    waitpid(t5);
                }
           }
       }
   }
}
```

Implementazione con i thread di Java

Alcuni strumenti base per la concorrenza in Java:

- public void start(): inizia l'esecuzione del thread.
- public void sleep(long ms): attesa per ms millisecondi.
- public void join(): attende la terminazione di un thread.
- public int setPriority(int priority): cambia la prioritá di un thread.
- public int getId(): ritorna l'ID del thread.
- public void yield(): il thread corrente esegue un altro thread.
- public void stop(): ferma il thread.

Pesudocodice:

```
t0.start(); t0.join();
t1.start(); t2.start(); t3.start();
t1.join(); t2.join();
t4.start(); t4.join();
t3.join();
t5.start(); t5.join();
```

Codice reale Java

```
import java.lang.*; import java.util.*;
public class prog {
    public static void main(String[] args) {
      data item= new data(1,2,3,4,5,6);//3*7-5/6=20.16666
      P1 p1=new P1(item); // istanze
      P2 p2=new P2(item);
      P3 p3=new P3(item);
      P4 p4=new P4(item);
      P5 p5=new P5(item);
      p1.start(); // calcolo concorrente
      p2.start();
      p3.start();
      try { p1.join(); } catch(InterruptedException e) {System.out.println("Errore P1");}
      try { p2.join(); } catch(InterruptedException e) {System.out.println("Errore P2");}
      p4.start();
      try { p3.join(); } catch(InterruptedException e) {System.out.println("Errore P3");}
      try { p4.join(); } catch(InterruptedException e) {System.out.println("Errore P4");}
      p5.start();
      try { p5.join(); } catch(InterruptedException e) {System.out.println("Errore P5");}
      p1.stop();  p2.stop();  p3.stop();
      p4.stop();  p5.stop();
      System.out.println(" Fine programma ");
    }
}
```

Determinatezza: definizioni

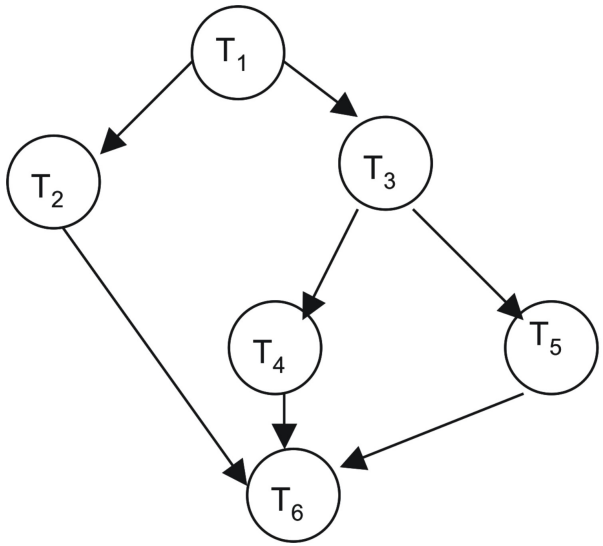
Thread piú efficienti ma possono avere problemi di DETERMINATEZZA.

- 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, $V(M_i, \alpha) = V(M_i, \alpha')\forall i = 1 \ldots m$ 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.

Esempio di Indeterminatezza

Thread T1

```
void echo ( )
{
in = getchar( );
out = in;
cout << out;
}
```

Thread T2

```
void proc ( )
{
out = in + "...";
cout << out;
}
```

Thread T3

```
void stam ( )
{
out = in + "---";
cout << out;
}
```

Eseguendo $\overline{T_1}\underline{T_1}\,\overline{T_2}\underline{T_2}\,\overline{T_3}\underline{T_3}$ e scrivendo per esempio "abc" si ha in uscita la scritta: "abcabc...abc—" ma eseguendo
$\overline{T_1}\underline{T_1}\,\overline{T_2\,T_3}\underline{T_3\,T_2}$
si ha "abcabc...abc—" e se $\overline{T_1}\underline{T_1}\,\overline{T_3\,T_2}\underline{T_3\,T_2}$

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)