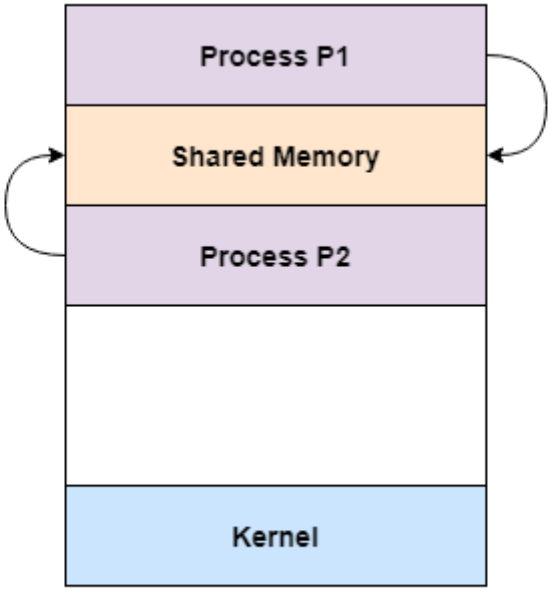
Distributed Systems and Middleware Technologies

Concorrenza, definizione e principi

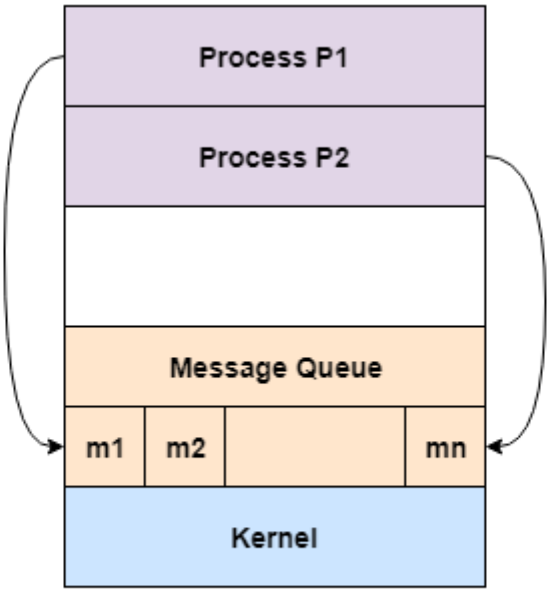
In un sistema moderno possiamo avere piu' unita' computazionali (Computational Units CU) che collaborano e interagiscono per ottenere il risultato finale. Lo scambio di informazioni tra le diverse unita' puo' avvenire in vari modi.

- Shared Memory Model: le informazioni vengono scambiate attraverso una memoria condivisa che puo' essere acceduta da tutti i processi. Puo' creare problemi di sincronizzazione in quanto dobbiamo essere sicuri che i processi non stiano scrivendo sulla stessa locazione di memoria condivisa.



Shared Memory Model

- Message Passing Model: consente lo scambio di informazioni senza essere direttamente collegati attraverso una coda di messaggi. Questo consente di costruire modelli paralleli pur richiedendo un tempo di accesso maggiore.



Message Passing Model

Processi vs Thread

I thread sono dei *processi leggeri* che condividono lo stesso spazio di indirizzamento. Molteplici thread possono essere eseguiti in modo concorrente e il cambio di contesto risulta essere piu' veloce rispetto a quello dei processi.

I processi invece non condividono lo stesso spazio di indirizzamento e di conseguenza devono comunicare attraverso metodi di comunicazione *inter-processo* come il message passing.

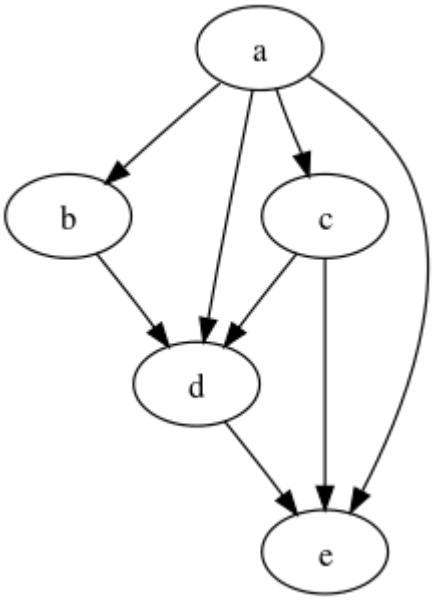
Rappresentazione formale di una computazione concorrente

Definiamo due insiemi:

- A: insieme delle azioni
- PC: insieme dei vincoli di precedenza

La precedenza e' una relazione matematica binaria che vincola l'esecuzione di un'azione ad un'altra.
L'insieme PC e' un insieme *parzialmente ordinato* (Partially Ordered SET, poset).

Per rappresentare l'esecuzione di un programma concorrente si puo' utilizzare un Direct Acyclic Graph (DAG).



In questo caso i nodi che hanno archi entranti sono soggetti a vincoli di precedenza, mentre quelli con solo archi uscenti non hanno vincoli di precedenza.
In generale posso eseguire due azioni in modo concorrente se seguendo gli archi non posso raggiungere l'altra azione. Ad esempio B e C possono essere eseguite in modo concorrente. Al contrario, le azioni non sono eseguibili in modo concorrente se sono direttamente raggiungibili tramite gli archi.

$\preceq_p$: simbolo usato per l'ordinamento parziale

$\preceq_p = P_p^+$ dove PC^+ e' la **chiusura transitiva**.
La chiusura transitiva di una relazione R è un'altra relazione, tipicamente denotata con R^+, che aggiunge ad R tutti quegli elementi che, pur non essendo necessariamente in relazione direttamente fra loro, possono essere raggiunti da una "catena" di elementi tra loro in relazione. In pratica, la chiusura transitiva di R è la più piccola (in senso insiemistico) relazione transitiva R^+ tale che $R \subset R^+$.

- Non-strict partial order: relazione ordinata che e'

1. riflessiva: a $\preceq$ a
2. transitiva: se $a <= b$ e $b <= c \rightsquigarrow a <= b$
3. anti-simmetrica: se $a <= b$ e $b <= a \rightsquigarrow a = b$ ma non significa che se $a <= b$ e $a \neq b \rightsquigarrow b <= a$, invece $\neg b <= a$.

- Strict partial order: non ho piu' la riflessivita', ma l'irriflessivita' a < a **NOT TRUE**.

Un altro problema che si pone e' che PC non e' necessariamente un insieme ridotto al numero minimo di elementi. Questo vuol dire che possiamo eliminare degli archi. Un metodo per eliminare gli archi superflui e' detto **calcolo del diagramma di Hasse**.
Anziche' effettuare la chiusura transitiva, effettuo la *riduzione transitiva* del partial order.

Voglio trovare il piu' piccolo R tale che $R^+ = P_p^+$ e per i DAG e' unico.

Formalizzazione della definizione di concorrenza

$a_i \in A, a_j \in A$ sono eseguite concorrentemente **se e solo se** ne' $a_i \preceq_p a_j$ ne' $a_j \preceq_p a_i$ valgono, allora $a_i || a_j$.

Bisogna ricordare che la *concorrenza* non e' una relazione transitiva, infatti se $B||C$ e $C||B$ e $B||D$ **NON E' VERO CHE** $C||d$.

Concetto di processo

chain: un path sequenziale all'interno di un DAG viene definito *chain*. Formalmente possiamo dire che e' un sottoinsieme del poset il cui partial order si riduce ad un total order.

total order = strict partial order + totality
$\forall a_i, a_j \in A$ o $a_i \preceq_t a_j$ oppure $a_j \preceq_t a_i$

La *chain* piu' lunga del poset o del diagramma di Hasse e' detto **critical path**. L'esecuzione puo' essere partizionata in chain, ognuna delle quali assegnata ad un *worker*.
Bisogna fare attenzione pero' dato che questo non esula dal rispettare comunque i vincoli di precedenza e per fare questo ci avvaliamo di *synchronization mechanisms*.
Se abbiamo un solo worker? Organizziamo comunque le attivita' in un total order, sempre rispettando i vincoli.

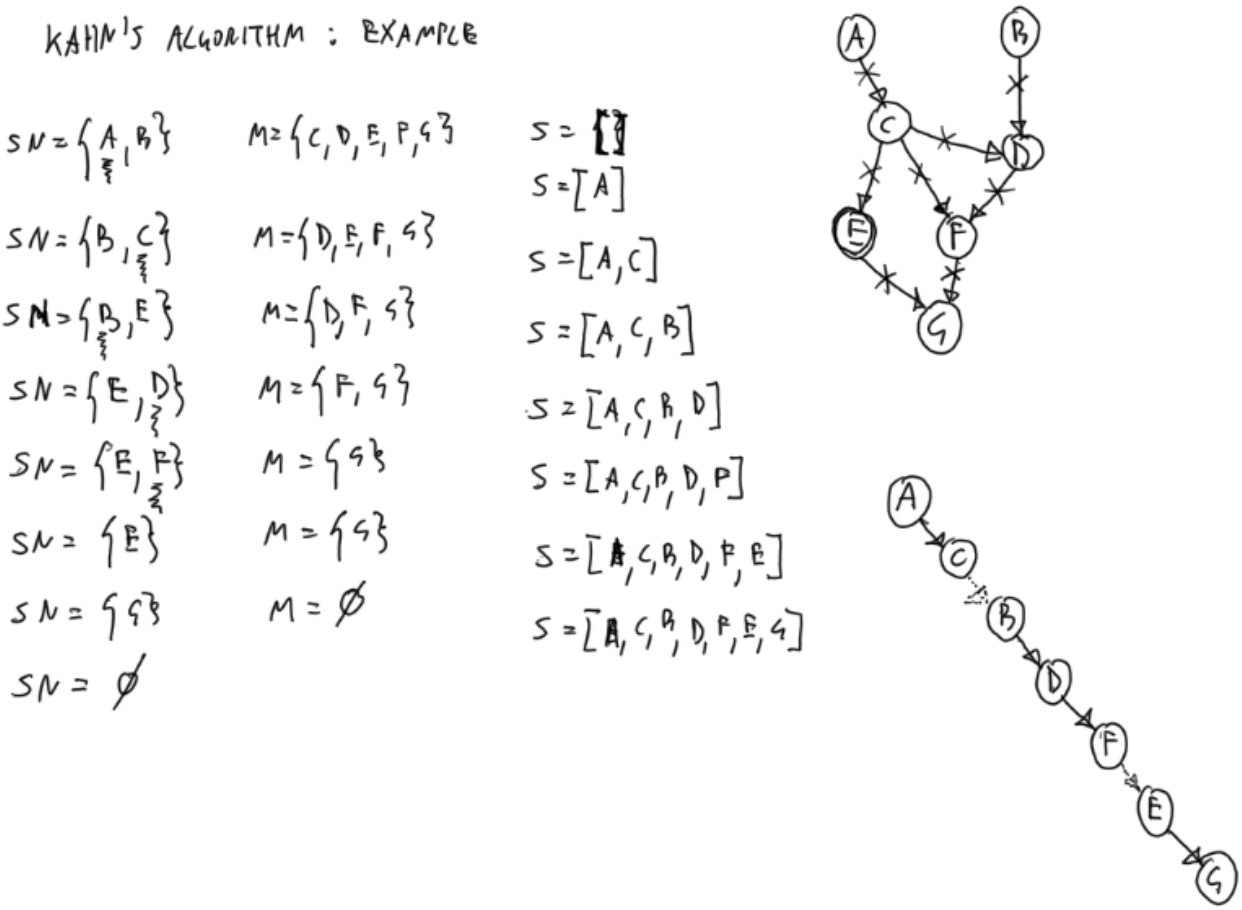
Topological sorting (aka linear extension)

Se $a_i \preceq_p a_j \rightsquigarrow a_i \preceq_t a_j$.

Kahn's algorithm

Pseudocodice:

```
while SN is not empty:
  pick a node from SN
  put it to the tail of SL
  foreach node m reachable directly from n:
    remove correspondent arc from the graph
    if m has no other incoming arcs:
      move m from M to SN
if graph has edges:
  throw error(not a DAG)
else:
  return n
```



True concurrency vs interleaving concurrency

P - programma originale, sequenziale
P' - programma migliorata, parallelo

T(P) - runtime di P
T(P') - runtime di P'

Noi speriamo che $T(P') <= T(P)$.

Vogliamo calcolare lo *speedup*:

$\sigma = \frac{T(P)}{T(P')}$

T(P) = s + p, dove s e' il *contributo puramente sequenziale* e s e' la *parte parallelizzabile*.

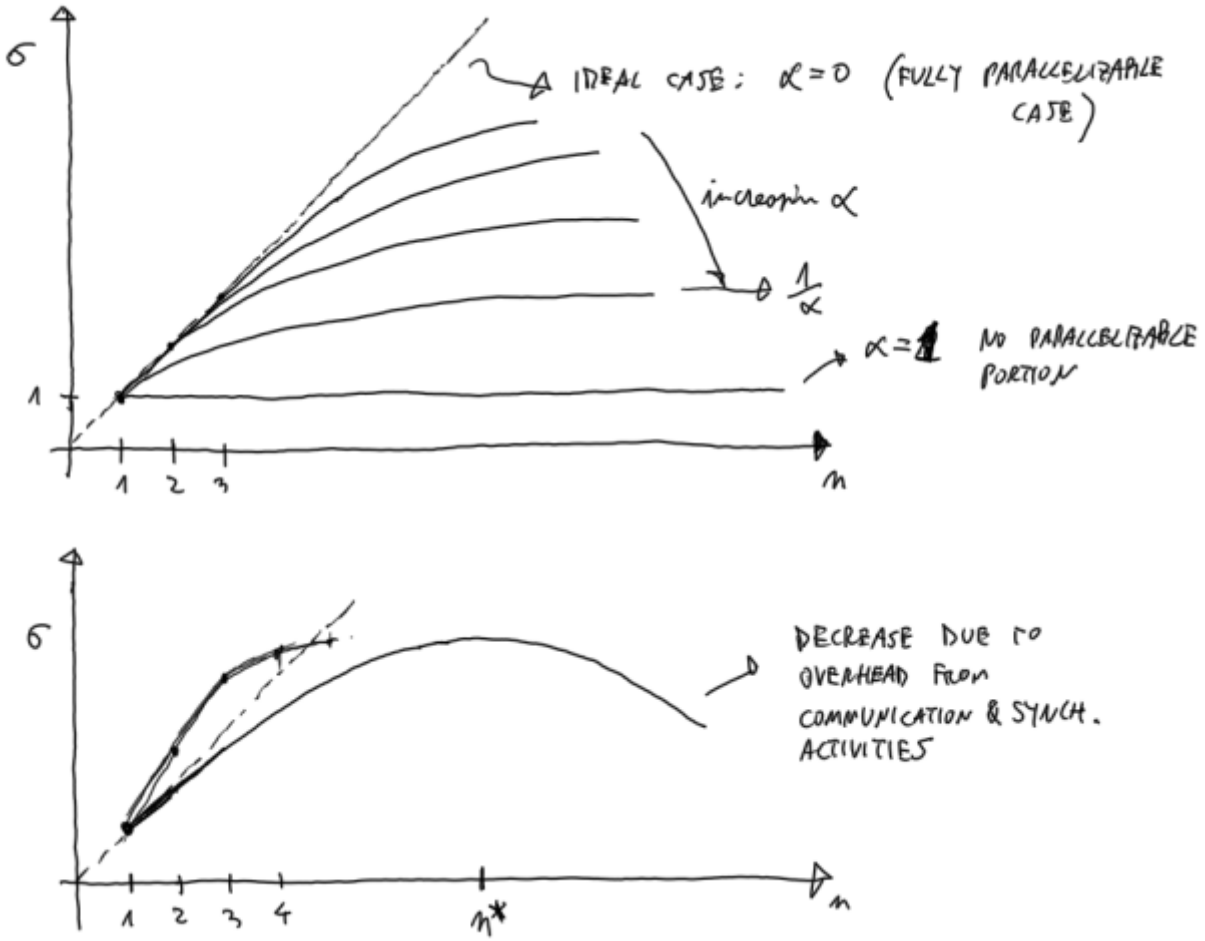
Sequential portion $\alpha = \frac{s}{s+p}$

Supponiamo di avere n unita' computazionali $p \rightsquigarrow \frac{p}{n}$ e quindi possiamo dire che lo *speedup* e' dato dalla **Legge di Amdahl**:

$\sigma(n, \alpha) = \frac{1}{\alpha + \frac{1-\alpha}{n}}$

dove

$\lim_{n \to \infty} \sigma(n, \alpha) = \frac{1}{\alpha}$



Dai grafici riportati notiamo come aumentare le unita' computazionali all'infinito non e' utile, anzi aumentarne troppo il numero puo' portare a degli svantaggi, dato che aumenta l'overhead causato dalla necessita' di comunicazione tra le varie unita'.