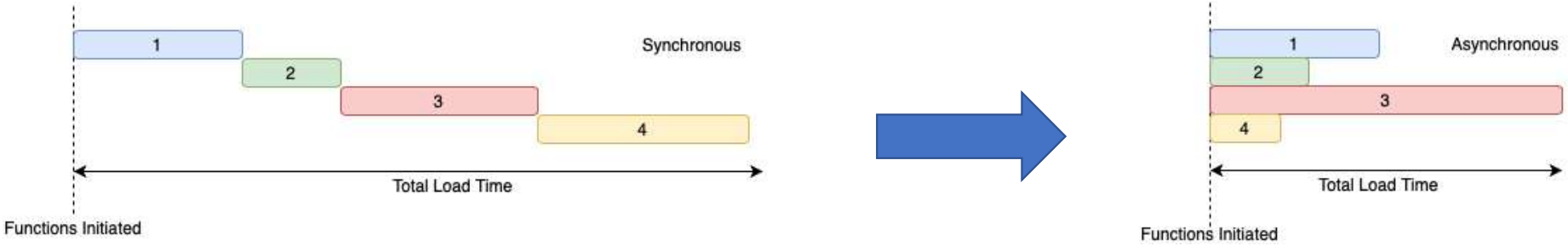
Async/await & MultiThreading

- Un **Processo**, es. ciascuna delle applicazioni console che abbiamo creato, ha a disposizione risorse come memoria e **Thread** ad essa allocati
- Un **Thread** esegue il codice, istruzione per istruzione
- Per impostazione predefinita, ogni processo ha un solo thread e questo può causare problemi quando sia necessario eseguire più di un'attività (**Task**) contemporaneamente

Async Await & MultiThreading

- Esecuzione di codice Asincrono
- Async Await



Async/await

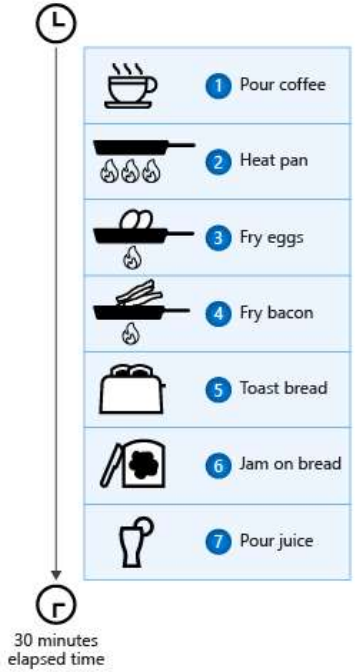
- Per effettuare e semplificare le chiamate asincrone
- Nuove parole chiave introdotte con C# 5
  - `Async`: gestisce la funzione in asincrono
  - `Await`: attende un'operazione asincrona
- Scriviamo codice come se fosse «sincrono»
- Tutto gestito dal compilatore

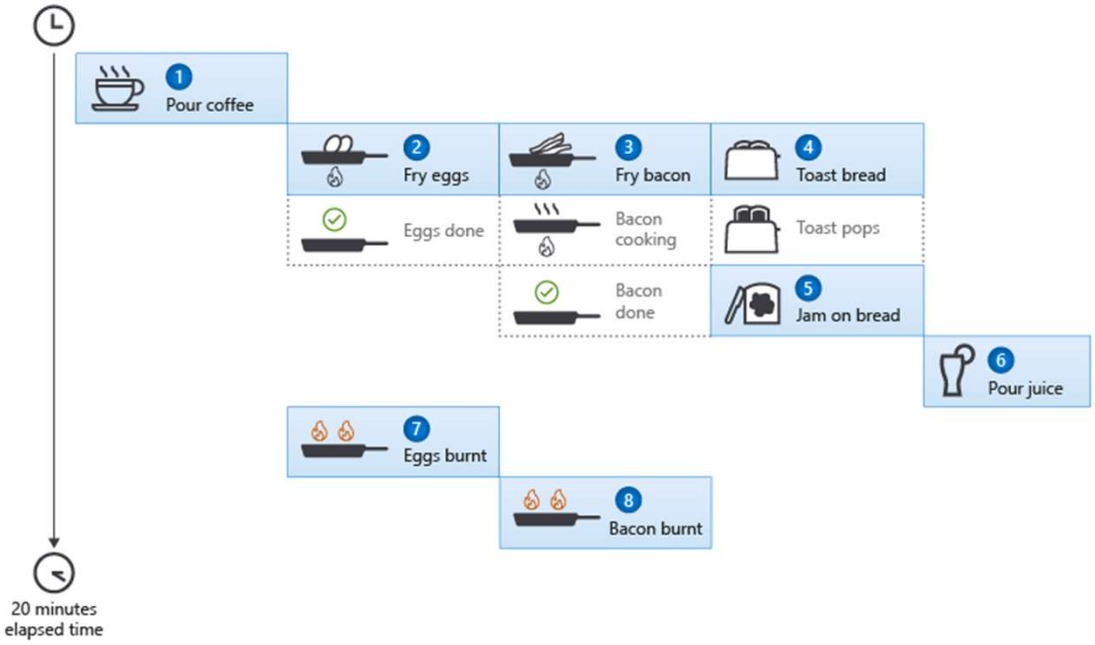
Async/await

- Utilizzabile con tutti i metodi `***Async`
  - Restituiscono un riferimento all'operazione, non il risultato
- Normale gestione eccezioni
  - Costrutto `try/catch/finally`
- Gestione automatica delle problematiche di threading
  - Prima era demandato ad ogni classe (es. WebClient)
  - Per scalare su web e per UI fluide su client

Demo

Demo – Versione sincrona





Esercitazione 4

Riprendere l'esercizio precedente:

- Aggiungere all'interfaccia IFileSerializable i metodi asincroni
    - SaveToFileAsync
    - LoadFromFileAsync
- Implementarli per la classe Shape e le sue classi derivate (sostituire i metodi sincroni di StreamReader / StreamWriter con quelli asincroni)

Per verificare il corretto funzionamento delle classi realizzate, utilizzare il metodo asincrono nel metodo Main (attenzione alla firma di Main …).

Delegate

- I **delegate** sono l'equivalente .NET dei puntatori a funzione del C/C++ unmanaged, ma hanno il grosso vantaggio di essere tipizzati
- In C# lo si dichiara con la parola chiave **delegate**

```
delegate void MyDelegate(int i);
```

- Il compilatore crea di conseguenza una classe che deriva da **System.Delegate** oppure **System.MulticastDelegate** (di nome **MyDelegate**)
- Queste due classi sono speciali e solo il compilatore può derivarle

Delegate

- Da programma il delegate viene istanziato passandogli nel costruttore il nome del metodo di cui si vuole creare il delegate.

```
MyDelegate del = new MyDelegate(MyMethod);
```

→

```
void MyMethod(int i) { }
```

- L'istanza può finalmente essere invocata

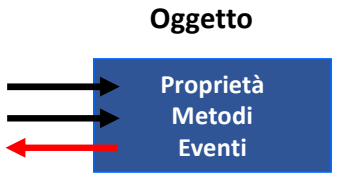
```
del(5); // esegue MyMethod (integer)
```

Demo

Delegati

Eventi

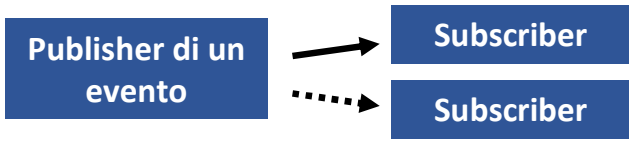
- Un **<u>evento</u>** è un membro che permette alla classe di inviare notifiche verso l'esterno
- L'evento mantiene una lista di *subscriber* che vengono iterati per eseguire la notifica
- Tipicamente sono usati per gestire nelle Windows Forms le notifiche dai controlli all'oggetto container (la Form)



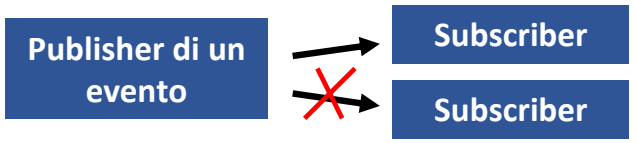
- Si parla di:
  - *Publisher* Inoltra gli eventi a tutti i subscriber
  - *Subscriber* Riceve gli eventi dal publisher

Eventi

- Ciascun subscriber deve essere aggiunto alla lista del publisher (*subscribe*) oppure rimosso (unsubscribe).




```
c.MyEvent += f;
```




```
c.MyEvent -= f;
```

Demo

Eventi

Esercitazione 5

Realizzare un'applicazione Console che consenta di gestire l'evento di invio di messagistica utilizzando diverse forme (email, VoIP, Whatsapp).

Per l'invio dell'email aggiungere il Provider (Outlook, Gmail, ecc)

Per l'invio del messaggio Whatsapp indicare se si tratta di un gruppo.

All'interno del messaggio deve essere inserito:

- Il testo del messaggio (string)
- una data di ricezione (DateTime)
- Il mittente (string)

Utilizzare l'approccio Publish-Subscriber visto a lezione.