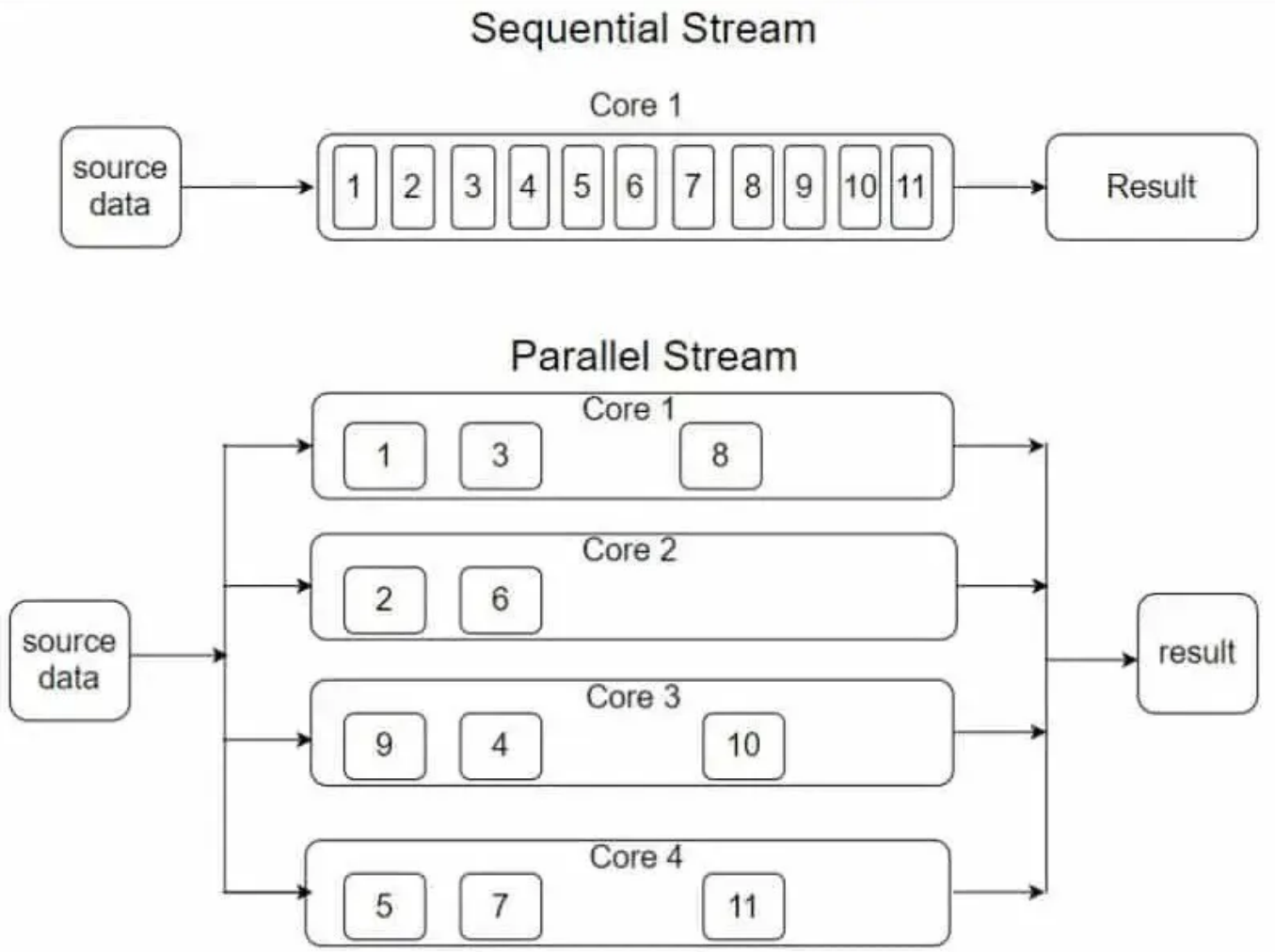
PDP 04-05

Parallel Streams



`Stream` = rappresentazione iterazione di oggetti potenzialmente infinita

- trasformazione da varie sorgenti
- utilizzo di API

Rispetto alle `Collections`:

- performance nell'accesso ai contenuti
  - accesso casuale massimizzato
- downside: mancanza del controllo totale nel recupero

Flag di controllo:

- piena visibilità
- gestire pipeline di esecuzione in modo controllato

Alcune operazioni:

- *short-circuiting*:
  - significa che possono interrompere l'esecuzione della pipeline prima dell'esame di tutti gli elementi
- *stateful*:
  - può necessitare di consumare tutto o gran parte dell'input per poter emettere l'output mantenendo le caratteristiche desiderate (per es. ordinamento).

`Spliterator`:

- flussi paralleli indipendenti
- stimo gli elementi
- prendo le caratteristiche
- li attraverso suddividendo iterazione in vari rami

`Collector`:

- interfaccia per gestire la riduzione di oggetti mutabili
- a prescindere dal tipo di stream e di algoritmo, gestire le strategie di controllo dell'algoritmo

E.g., `ForkJoinPool`:

- interfaccia per gestione di molteplici task
- pool = divide et impera sui singoli task

Blocking factor:

- Intensità del calcolo di un algoritmo
- Un algoritmo con `BF=0`
  - occupa costantemente la CPU.
- Un algoritmo con `BF=1`
  - è costantemente in attesa di I/O

```
#of cores
```

#threads <= ----------

1-BF

Use case: Esempio svolto

- Ricerca concorrente
  - Spliterator
  - Iterazione su tutte le mosse

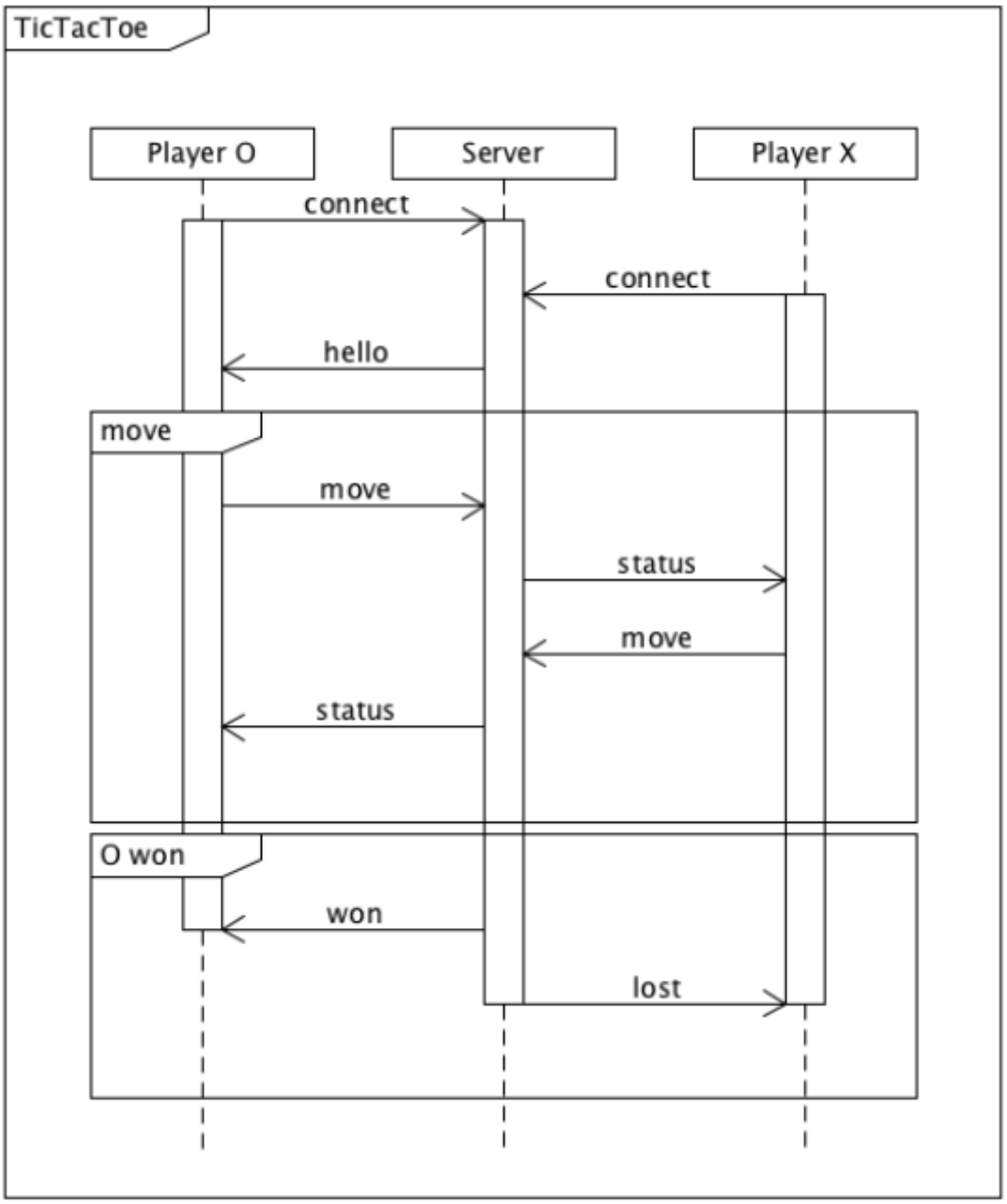
```
public class GameIterator implements Spliterator< Game > {
Queue< Game > current;

public GameIterator(Game seed) {

current = new LinkedList< Game >(seed.moves()); }

...

current.addAll(res.moves());
```



Programmazione distribuita

Motivazioni:

- Affidabilità
- Suddivisione del carico
- Diffusione

Caratteristiche algoritmo distribuito:

- Concorrenza dei componenti

- Totale asincronia
- Fallimenti imperscrutabili

RPC - *Remote Procedure Call*

- Stub: scambio tramite procedure di messaggi

Serializzazione = Marshalling

- il metodo con cui un oggetto viene predisposto per la trasmissione in un messaggio
- usare un meccanismo di codifica che prende direttamente l'oggetto e lo traduce in messaggio per eseguire il passo di *marshalling*

Contrario: *deserializzare* corrisponde al passo di *unmarshalling*

Le problematiche che la serializzazione deve affrontare sono:

- gestire il cambiamento strutturale delle classi
- serializzare grafi di oggetti
- indicare oggetti che non possono/non devono essere serializzati
- assicurare l'integrità e l'affidabilità dei dati serializzati
- rendere *marshalling*/*unmarshalling* efficienti in tempo e spazio

Le classiche primitive del modello TCP/IP:

- Sockets (Connessioni TCP)
- Datagrams (Pacchetti UDP)

Librerie:

- java.nio
- HttpClient

Framework:

- gRPC
- OkHTTP

Java EE = Standard di realizzazione di quanto sopra

Primitive di networking

Le astrazioni di base che abbiamo a disposizione per la comunicazione fra più JVM corrispondono direttamente alle caratteristiche del protocollo TCP/IP:

- Sockets
- Datagrams

Un `Socket` è una astrazione per la comunicazione bidirezionale punto-punto fra due sistemi-

- Un *client* `Socket` è un socket per iniziare il collegamento verso un'altra macchina
- Un *server* `Socket` è un socket per attendere che un'altra macchina ci chiami

Lato client, lo diventa appena il collegamento (l'handshake TCP/IP) è completato.

Lato server, viene ritornato quando un collegamento viene ricevuto e completato.

Questi stream sono sottoposti a diverse regole:

- sono thread-safe, ma un solo thread può scrivere o leggere per volta, pena eccezioni
  - accesso atomico alle informazioni
- i buffer sono limitati, ed in alcuni casi i dati in eccesso possono essere silenziosamente scartati
- lettura e scrittura possono bloccare il thread
- alcune connessioni possono avere caratteristiche particolari (per es. urgent data)

Una volta terminato l'uso, il `Socket` va chiuso esplicitamente.

Un `Datagram` è un'astrazione per l'invio di un pacchetto di informazioni singolo verso una destinazione o verso più destinazioni.

- Il concetto di connessione è diverso rispetto al socket, e non c'è garanzia di ricezione o ordinamento in arrivo

Con i Datagram la logica del protocollo è differente. Abbiamo a disposizione:

- la dimensione del messaggio nota (e quindi l'informazione di ricezione completa)
- la possibilità di inviare messaggi a più indirizzi contemporaneamente (multicast)

Stream Socket:

- Dedicated & end-to-end channel between server and client.
- Use TCP protocol for data transmission.

- Reliable and Lossless.
- Data sent/received in the similar order.
- Long time for recovering lost/mistaken data

Datagram Socket:

- Not dedicated & end-to-end channel between server and client.
- Use UDP for data transmission.
- Not 100% reliable and may lose data.
- Data sent/received order might not be the same.
- Don't care or rapid recovering lost/mistaken data.

Una *URL* ha un formato complesso ed in grado di esprimere molte cose (protocolli, porte richieste, indirizzi remoti, file, jar, ecc.)

`HttpClient` più versatile ed in grado di fornire maggiore controllo su come le richieste vengono effettuate