OVERNET.

2024 WPC

Event Sourcing in .NET: Casi Reali e Benefici per Applicazioni Enterprise

Andrea Balducci

CTO & CO-FOUNDER NEBULA SRL

andrea.balducci@jarvis.cloud

/in/andreabalducci/

Tips & Tricks

Do e Don’t basati sull’esperienza.

Disclaimer: “secondo la mia esperienza”

Opinionated

Esempi sono basati su NStore, libreria MIT realizzata distillando le esperienze fatte in produzione con NEventStore.

https://github.com/nstoredev/NStore

https://github.com/andreabalducci/wpc2024

Event Sourcing

Pattern architetturale che cattura ogni modifica dello stato dell'applicazione come una sequenza immutabile di eventi.

Questi eventi vengono memorizzati in un log che diventa la **single source of truth** per lo stato del sistema.

Questo approccio consente di ricostruire lo stato attuale o passato dell'applicazione riproducendo la sequenza di eventi registrati.

Evento

Rappresentazione **immutabile** di un cambiamento di stato significativo che si è verificato nel sistema.

Ogni evento descrive cosa **è accaduto** (ad esempio, "Ordine creato" o "Biglietto venduto") e include informazioni rilevanti come timestamp, dati contestuali e identificatori univoci.

```
public record TicketSold(String TicketId, DateTimeOffset SoldAt)
```

Gruppo Utenti Italiani Solution Architect (GUISA)

“Il mondo in cui viviamo non è transazionale.

Se per appendere un quadro al muro stacchi un pezzo di intonaco, non puoi fare rollback.

Devi stuccare.”

Mauro Servienti

ES = Git per lo stato applicativo

Event Sourcing	Git	Analogia
Event	Commit	Modifica di stato (atomica)
Stream	Repository	Sequenza ordinata di modifiche
Store	Git Server	Collection di Sequenze

SaleOne

Creazione dello store, apertura dello stream e aggiunta dei primi eventi.

Lettura sequenziale dello store.

```
var wpc = _streams.Open("WPC");
await wpc.AppendAsync(new SaleStarted(_timeProvider.GetUtcNow()));
await wpc.AppendAsync(new TicketSold("T00001", _timeProvider.GetUtcNow()));
```


```
{
 "Position": 1,
 "PartitionId": "WPC",
 "Index": 1,
 "Payload": {
  "Type": "SaleStarted",
  "StartedAt": "2024-11-27T09:16:10.932975+00:00"
 },
 "OperationId": "6171012a-4d60-4756-8c71-4a6b51d30c4c"
}
```

```
{
 "Position": 2,
 "PartitionId": "WPC",
 "Index": 2,
 "Payload": {
  "Type": "TicketSold",
  "TicketId": "T00001",
  "SoldAt": "2024-11-27T09:16:10.935566+00:00"
 },
 "OperationId": "a560770d-8424-4fb5-829e-c934e139aafb"
}
```

```
var wpc = _streams.Open("WPC");
await wpc.AppendAsync(new SaleStarted(_timeProvider.GetUtcNow()));
await wpc.AppendAsync(new TicketSold("T00001", _timeProvider.GetUtcNow()));
```

```
{
  "Position": 1,
  "PartitionId": "WPC",
  "Index": 1,
  "Payload": {
    "Type": "SaleStarted",
    "StartedAt": "2024-11-27T09:16:10.932975+00:00"
  },
  "OperationId": "6171012a-4d60-4756-8c71-4a6b51d30c4c"
}
```

1

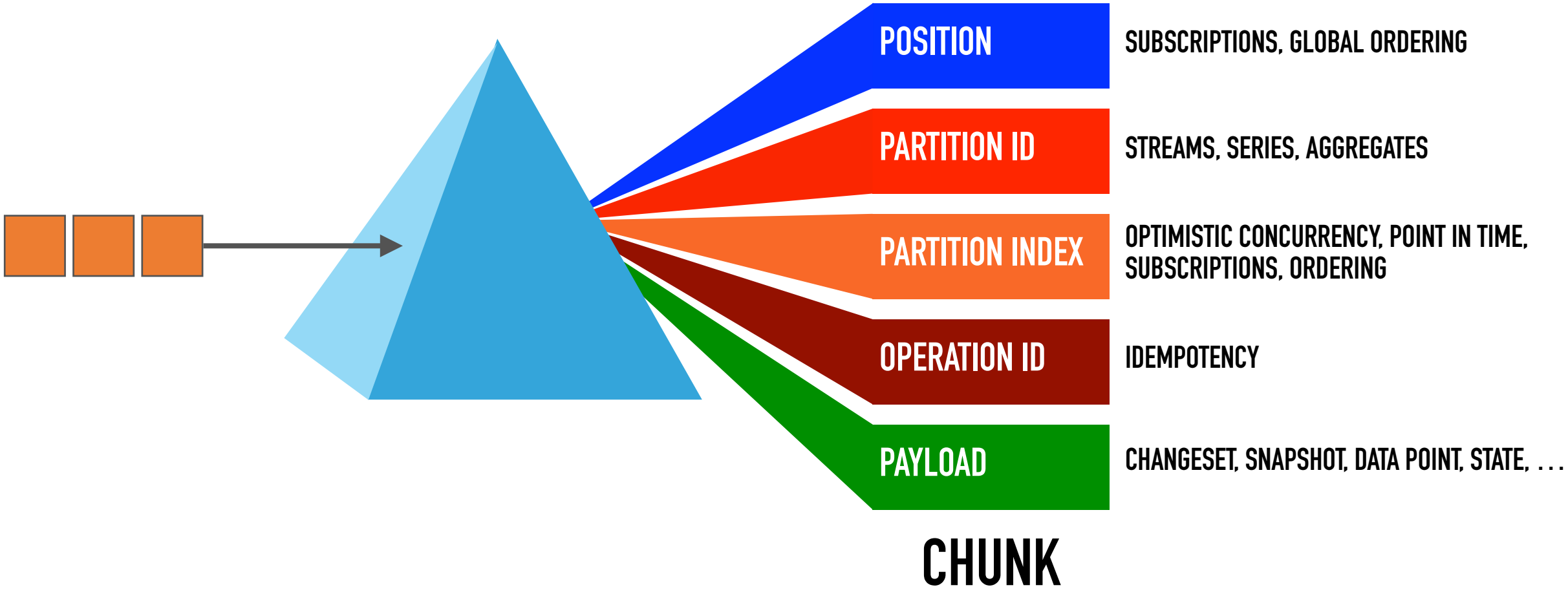
2

3

```
{
  "Position": 2,
  "PartitionId": "WPC",
  "Index": 2,
  "Payload": {
    "Type": "TicketSold",
    "TicketId": "T00001",
    "SoldAt": "2024-11-27T09:16:10.935566+00:00"
  },
  "OperationId": "a560770d-8424-4fb5-829e-c934e139aafb"
}
```

4

NStore persistence



TYPESYSTEM AS SCHEMA

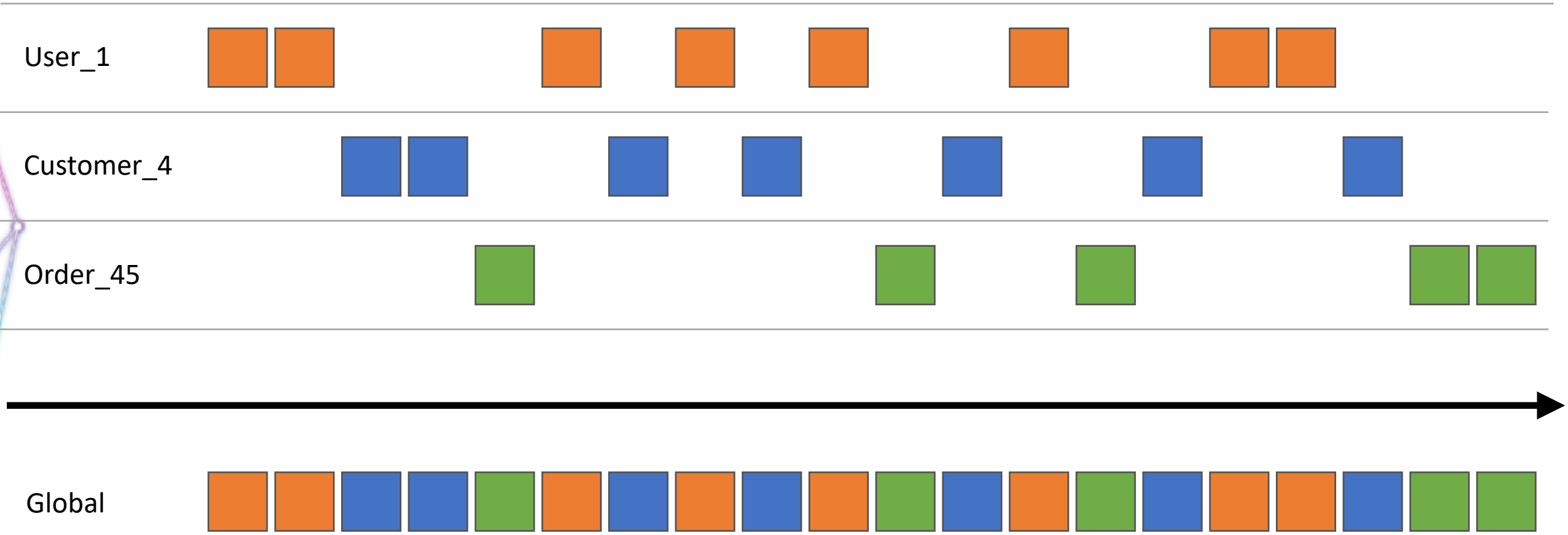
EASY WIN

```
public record TicketSold(
  String TicketId,
  DateTimeOffset SoldAt
);
```

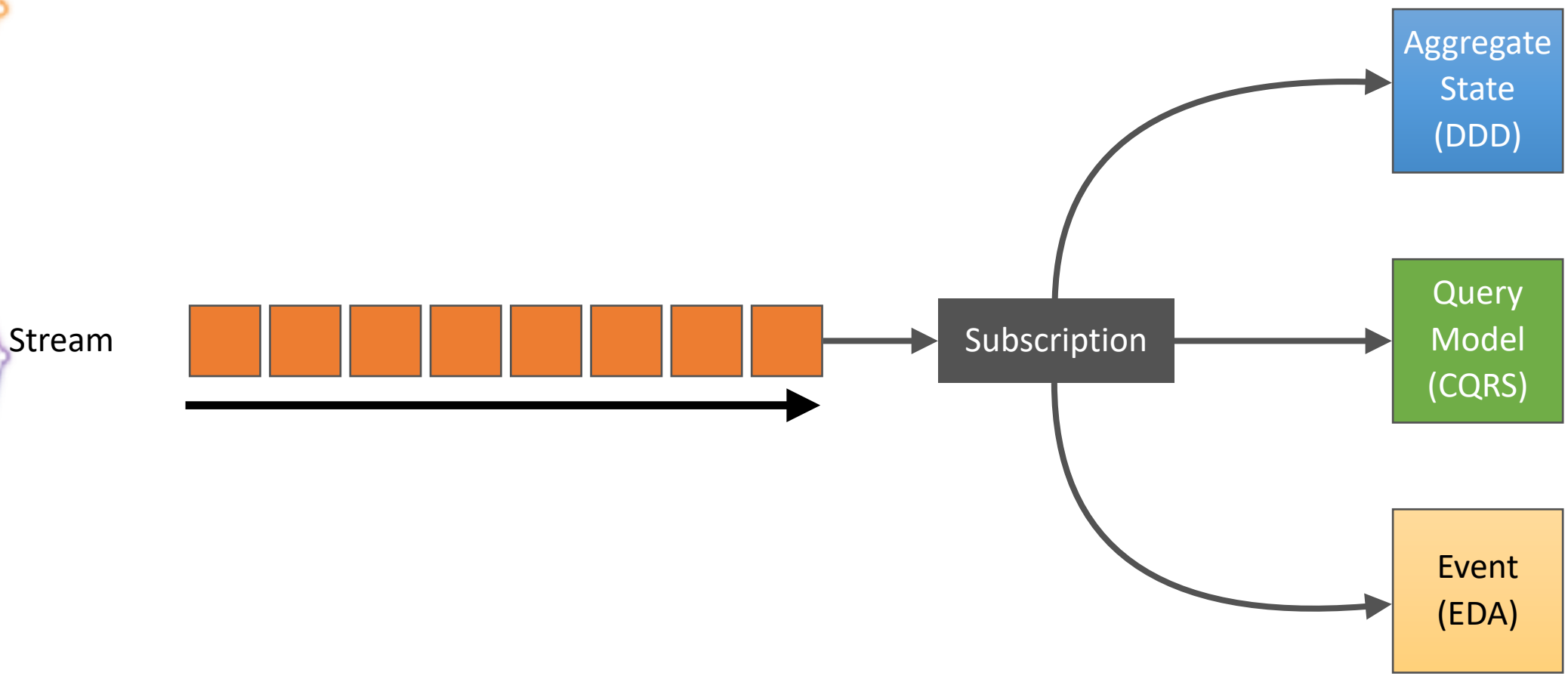
```
{
  "Type": "TicketSold",
  "TicketId": "T00001",
  "SoldAt": "2024-11-27T09:16:10.935566+00:00"
}
```

Poi arriva il momento di fare refactoring del codice con dati immutabili nei database di produzione.

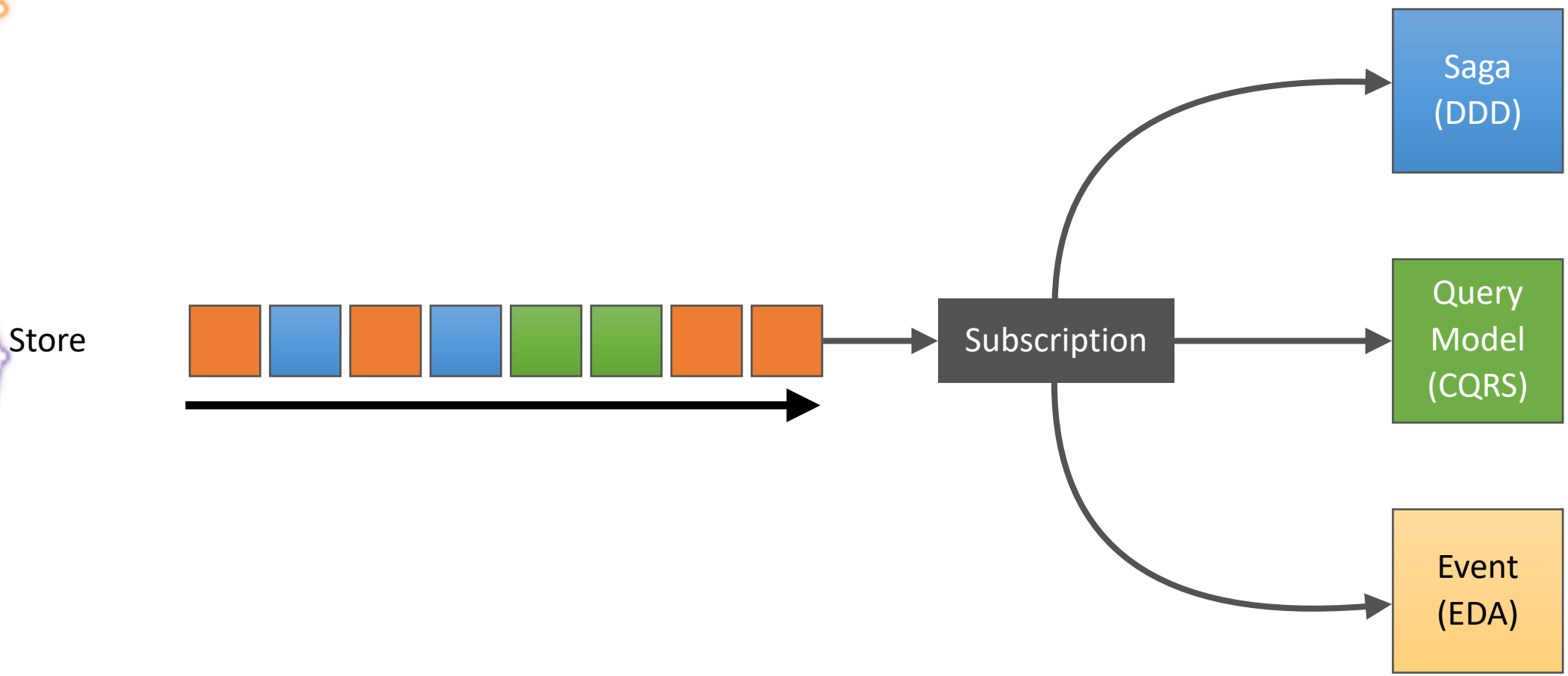
Event Store: database dei fatti



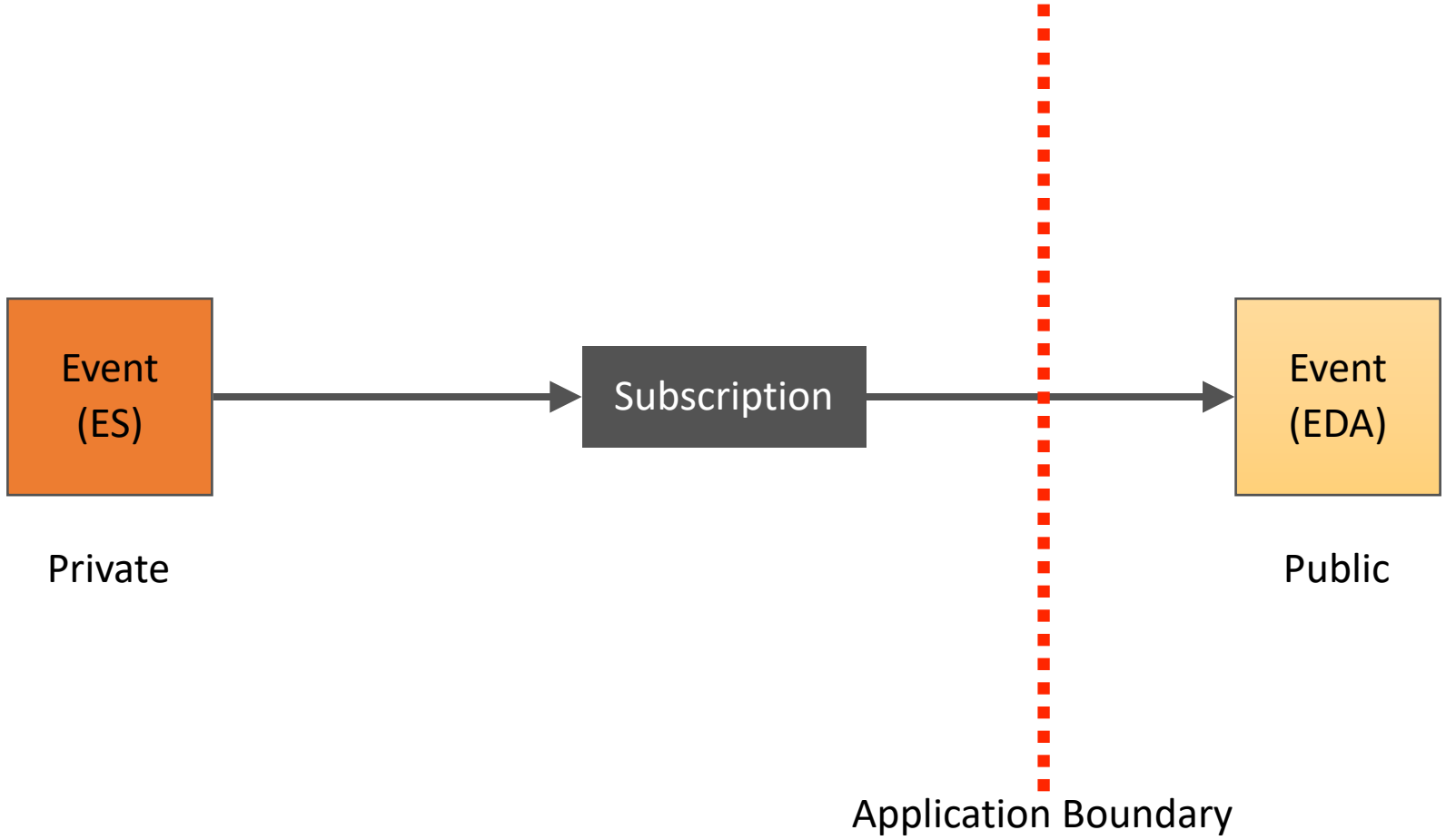
Subscription (stream)



Subscription (store)



Event != Event



SaleMany

Vendita concorrente di biglietti, lo stream garantisce l'ordine sequenziale eliminando i problemi di concorrenza.

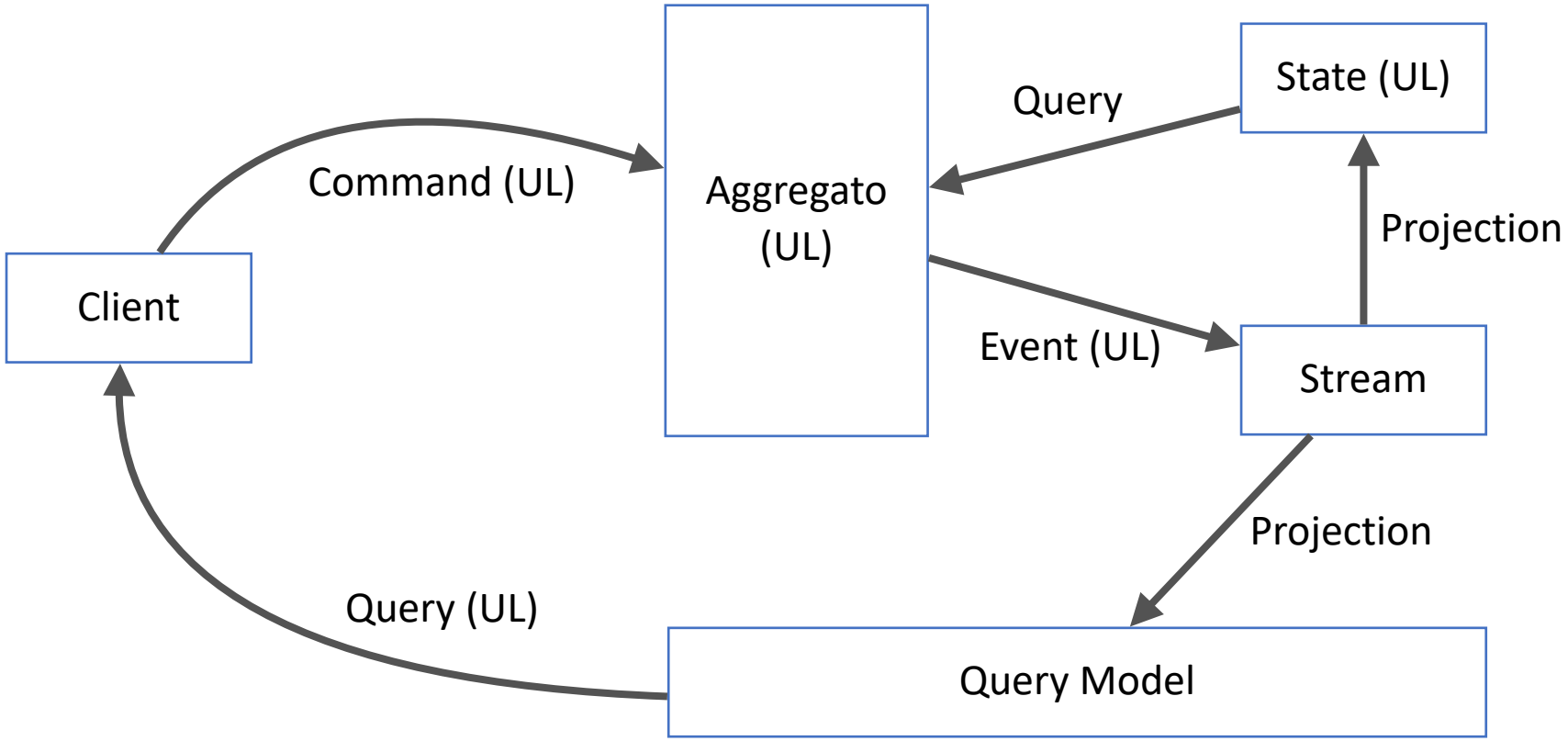
Aggregazioni / Proiezioni.

Nota: La concorrenza viene gestita a livello di storage, eccezioni, fillers & retries. Ottimizzabile con un actor fx.

Aggregato (DDD)

- Come garantisco il rispetto delle regole?
- Come garantisco integrità dei dati?
- Come implemento Ubiquitous Language?

Aggregato (DDD+CQRS+ES)



StoreAggregateSales

Vendita dei biglietti con verifica delle invarianti espresso in UL

Regole in UL

```
var wpc = StoreAggregate.CreateNew("WPC");
wpc.AddTickets(1);
wpc.SaleTicket("T00001");
await _repository.SaveAsync(wpc, "init");
```

```
public void SaleTicket(string ticketId)
{
    if (this.State.CanSale(1))
    {
        this.Emit(new TicketSold(ticketId, TimeProvider.System.GetUtcNow()));
    }
    else
    {
        throw new System.Exception("No more tickets available");
    }
}
```

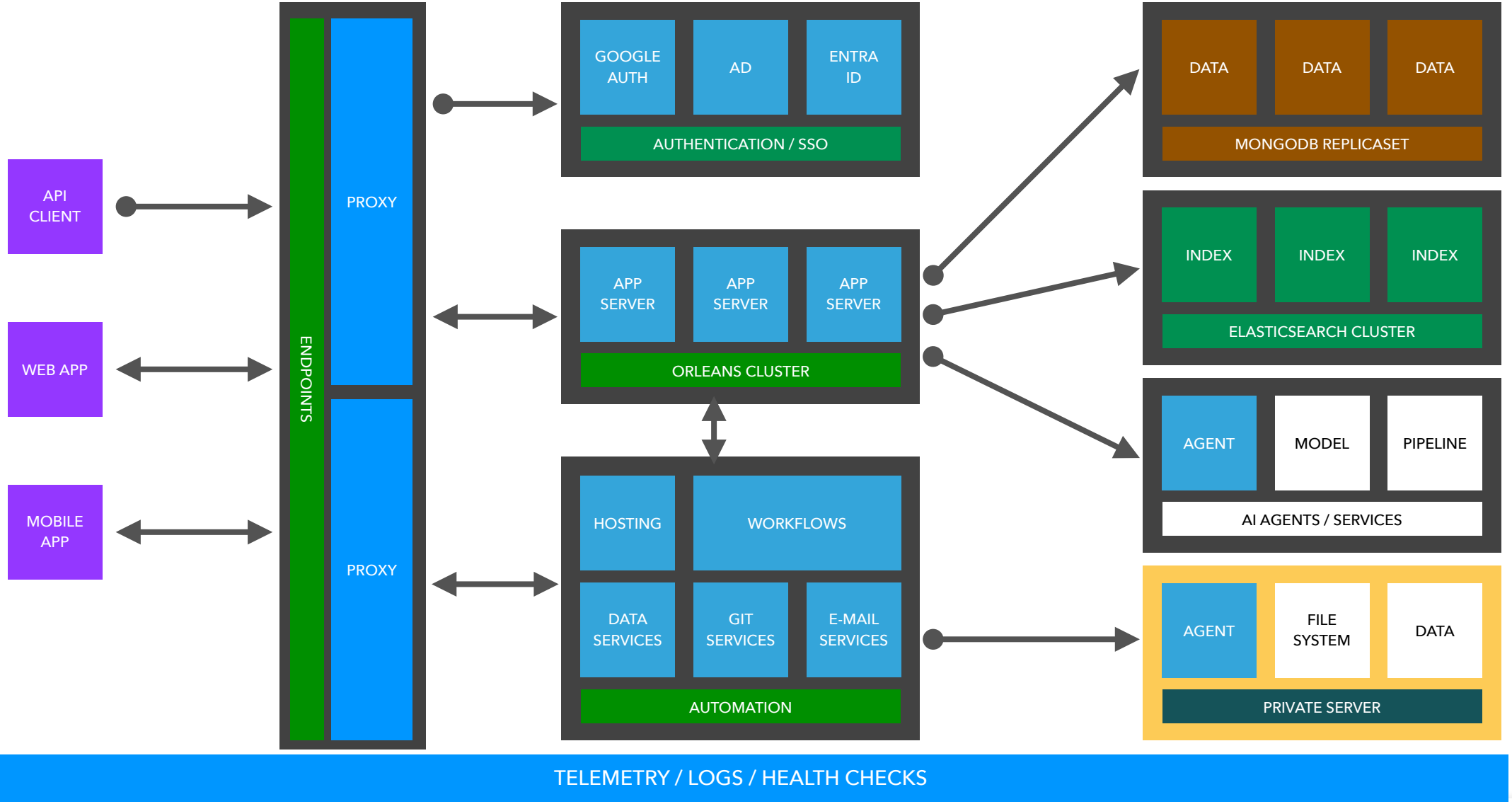
Recap

- L’evento deve essere consistente a livello di informazioni (timestamps)
- Usare il TypeSystem as Schema con attenzione
- Eventi di dominio != eventi di sistema
- Gli stream rappresentano la sequenza in cui sono memorizzati gli eventi, non quella in cui sono accaduti o assumono valore! (modelli n-temporali)
- EventSourcing richiede uno switch mentale non banale
- EventSourcing abilita agilità nel ciclo di vita del prodotto che può diventare un vantaggio competitivo

JARVIS

NStore ⋯⋯▶ Subscriptions

Cluster



Extra

Store

- Time Machine
- Branching
- Client a caldo
- Disacoppiamento Temporale

2024 WPC

26, 27, 28 NOVEMBRE MILANO

Valuta la sessione

GRAZIE!