NSTORE

Andrea Balducci

JARVIS FX

Gian Maria Ricci

#ACADEMY #DEV

Per informazioni e proposte di Academy scrivere a info@prxm.it

2021-04-21

<proximo>

GREETINGS TEAM.

Hello.

CAN YOU EXPLAIN WHY YOUR CODE IS FAILING?

People sometimes make mistak

YES. THEY DO. SHALL WE PLAY A GAME?

Love to. How about Event Sourcing?

WOULDN'T YOU PREFER A GOOD GAME OF CHESS?

Later. Let’s play Event Sourcing.

A STRANGE GAME.THE ONLY WINNING MOVE IS NOT TO PLAY.

HOW ABOUT A NICE GAME OF CHESS?

f (input) => output

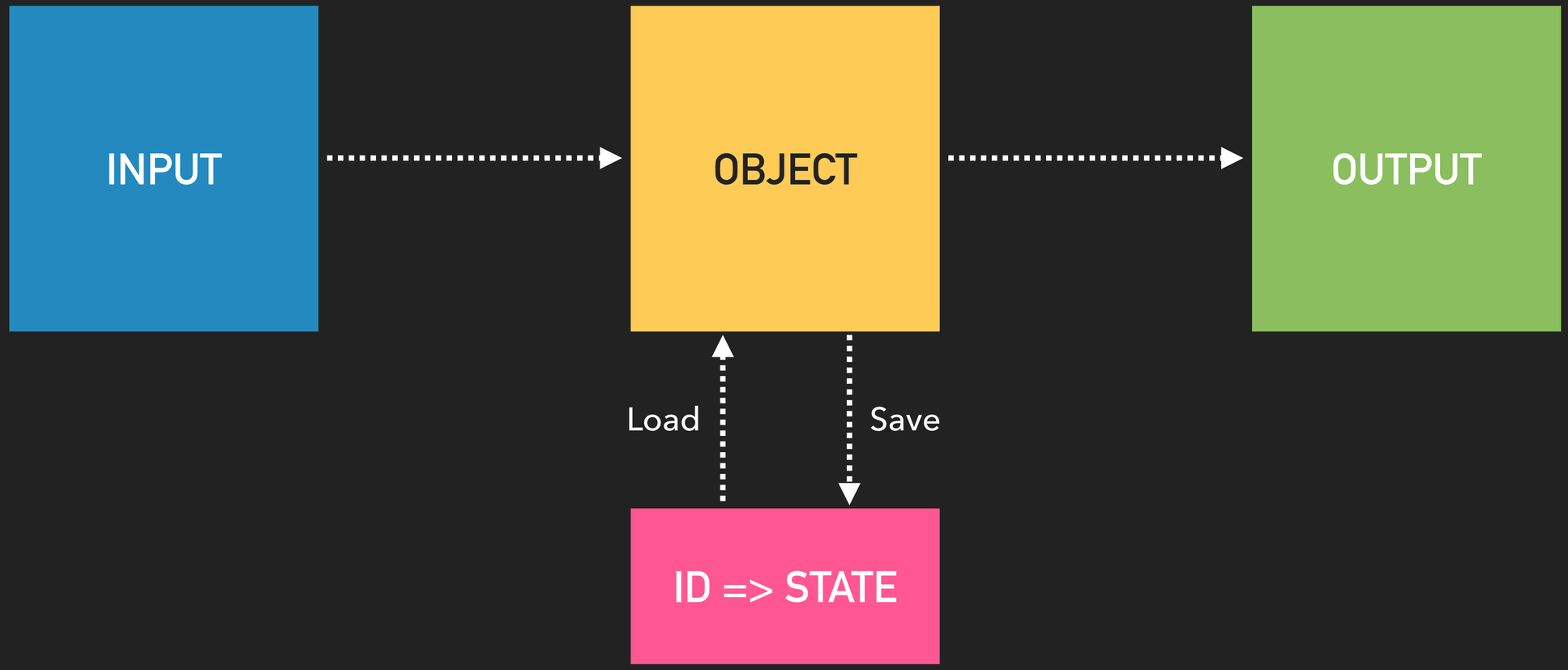
f (input, state) => (output, state)

OOP

f (input, state) => (output, state)

```
class Counter
{
    private int _value = 0;

    public int Add(int delta) => (_value += delta);
}
```

APPLICATION STATE



CQRS

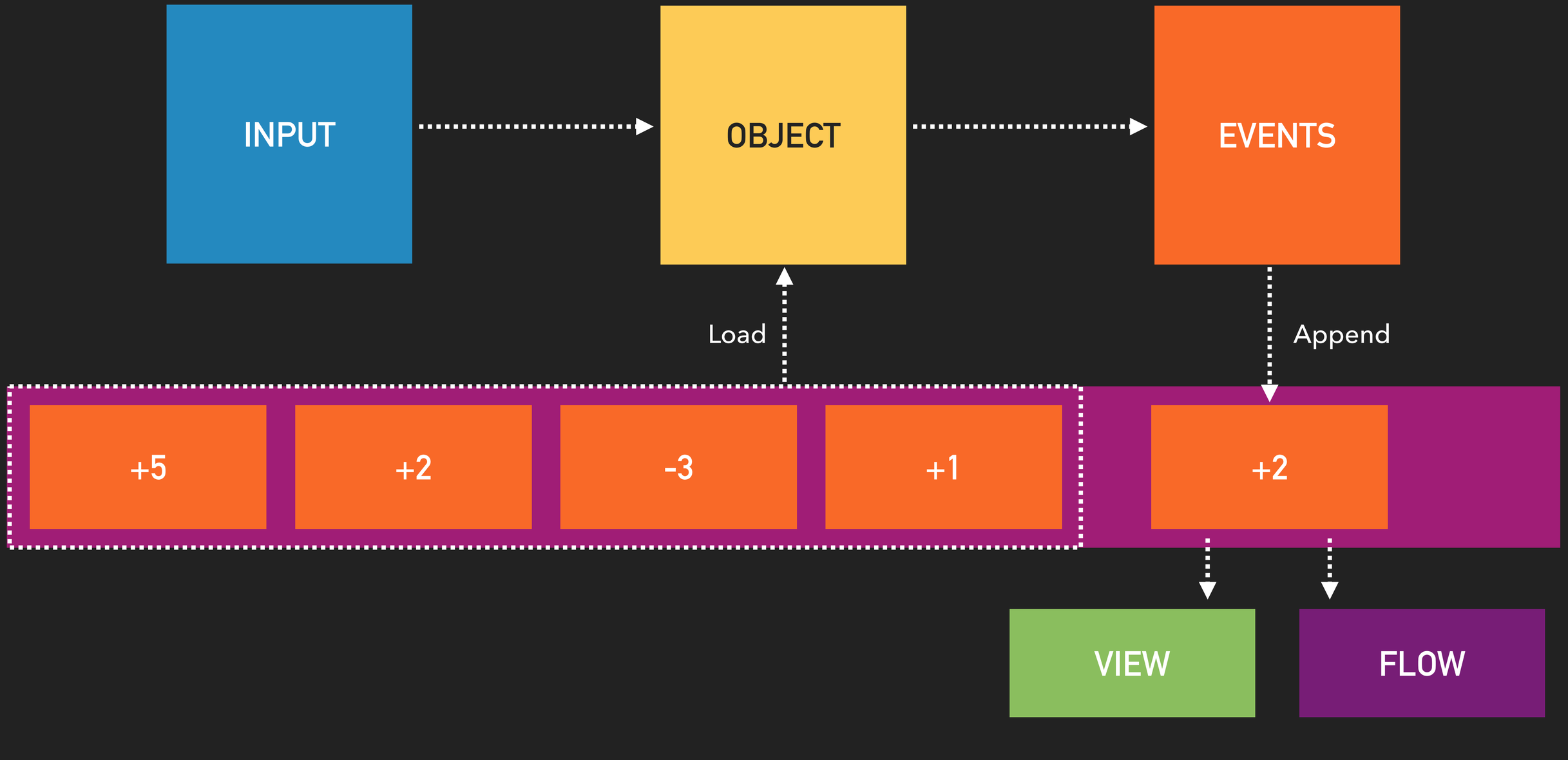
f (input, state) => (output, state)

```
class Counter
{
    private int _value = 0;

    public void Add(int delta) =>_value += delta;

    public int Current() => _value;
}
```

EVENT SOURCING

```
class Counter
{
    private List<(string, int)> _events = new();

    public void Inc(int delta) =>
        _events.Add(new("Added", +delta));

    public void Dec(int delta) =>
        _events.Add(new(“Subtracted", -delta));

    public int Current() =>
        _events.Select(x => x.Item2).Sum();
}
```

EVENT STREAM



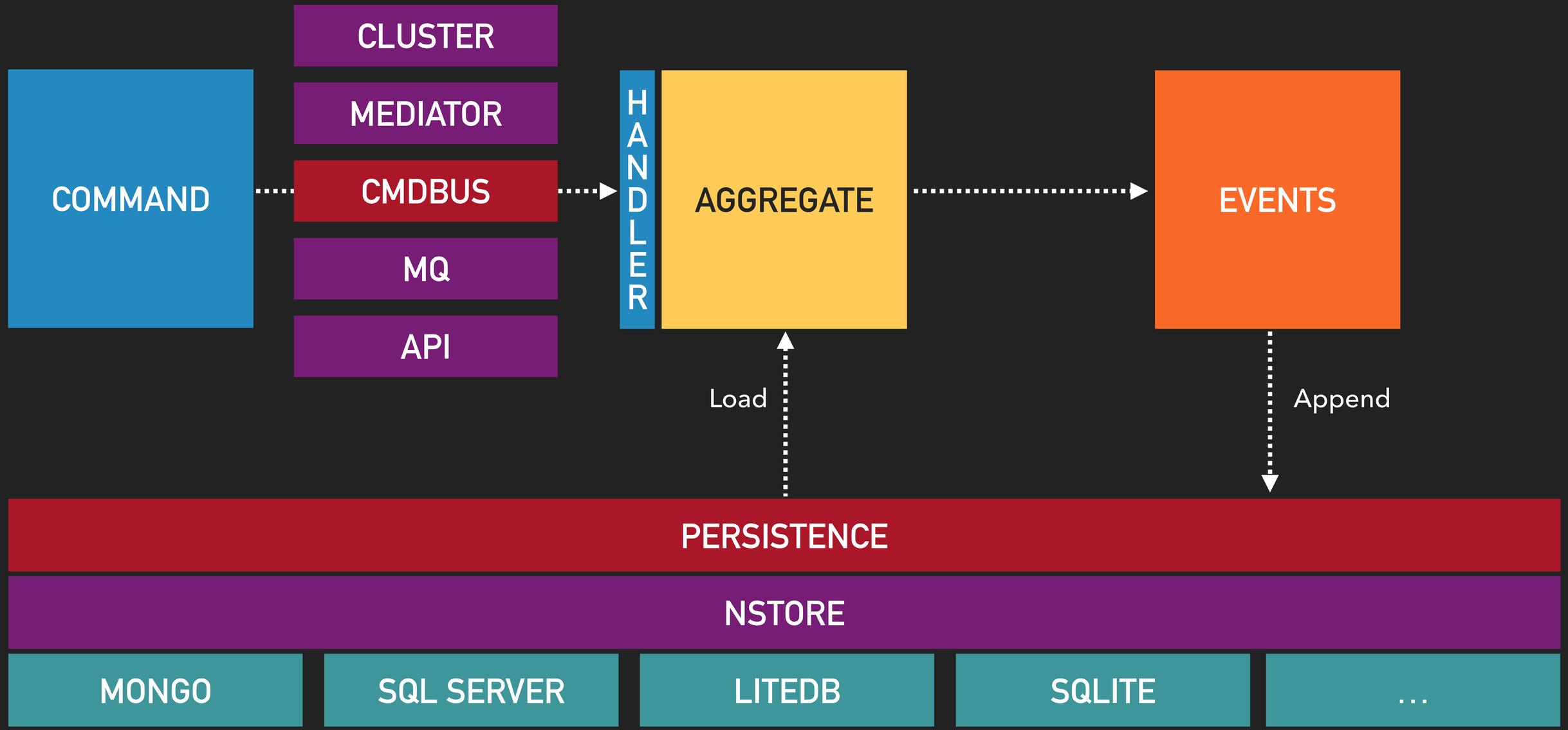
COMMANDS

```
var counter = repository.Get<Counter>(“C_1”);
counter.Inc(10);
repository.Save(counter);


new IncrementCounter(“C_1”, 10);
```

COMMANDS

```
new IncrementCounter(“C_1”, 10);


void Handler(IncrementCounter cmd)
{
    var counter = repository.Get<Counter>(cmd.Id);
    counter.Inc(cmd.Value);
    repository.Save(counter);
}
```

PORTS & ADAPTERS



CQRS + ES

Action

COMMAND

Mutation

EVENT EVENT EVENT EVENT EVENT

Timeline

Write

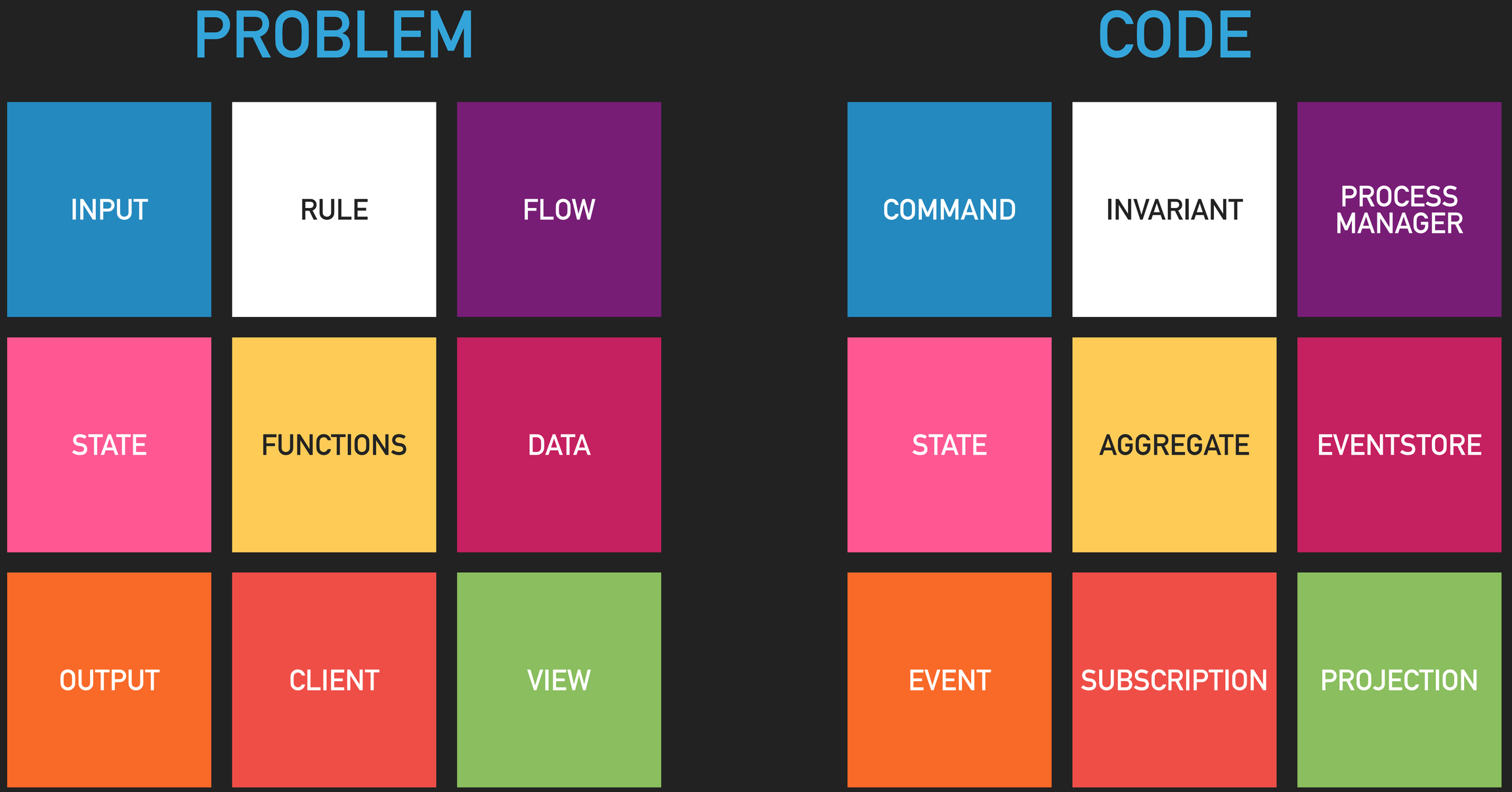
Reaction

VIEW

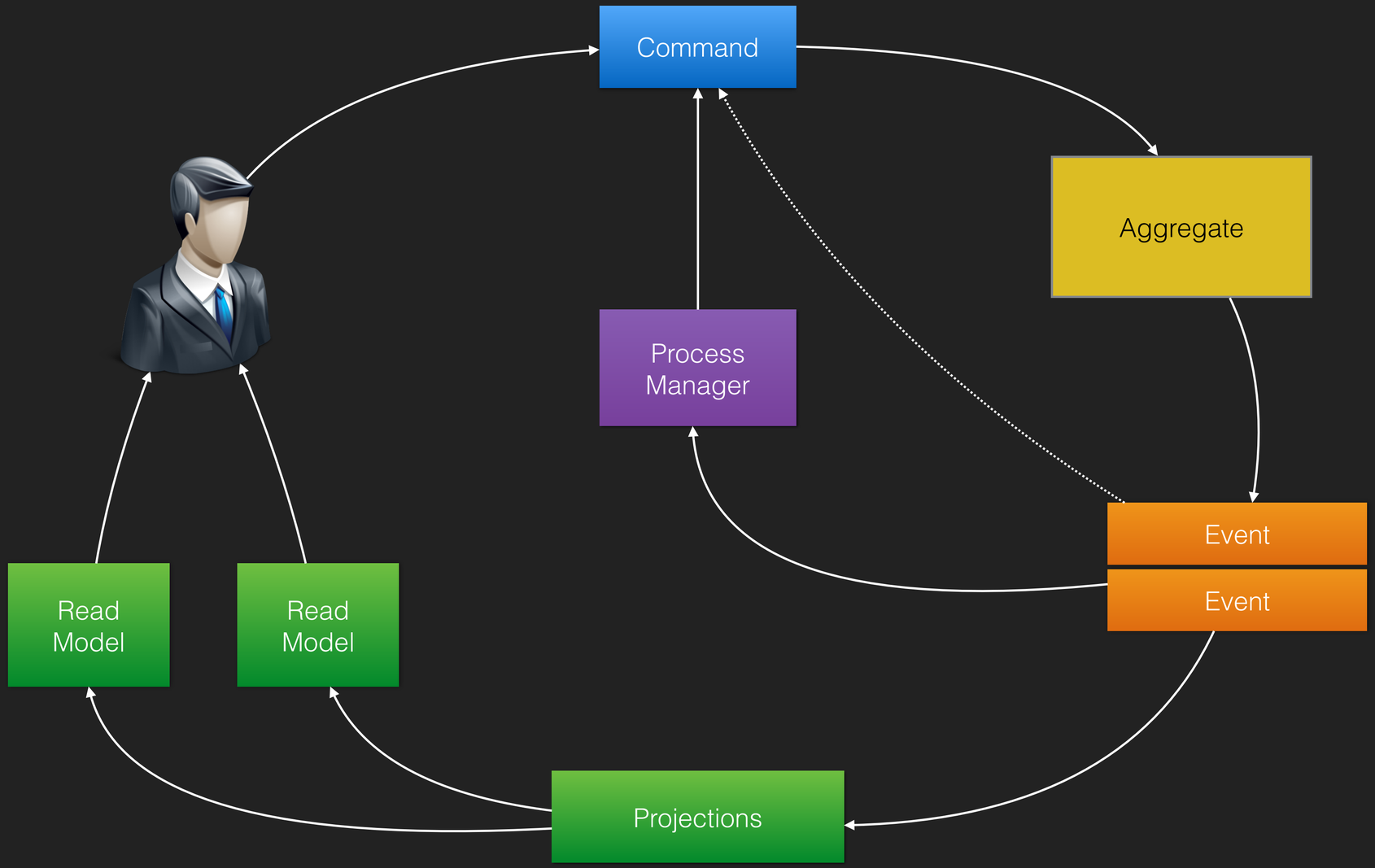
FLOW

SUBSCRIPTION

Read

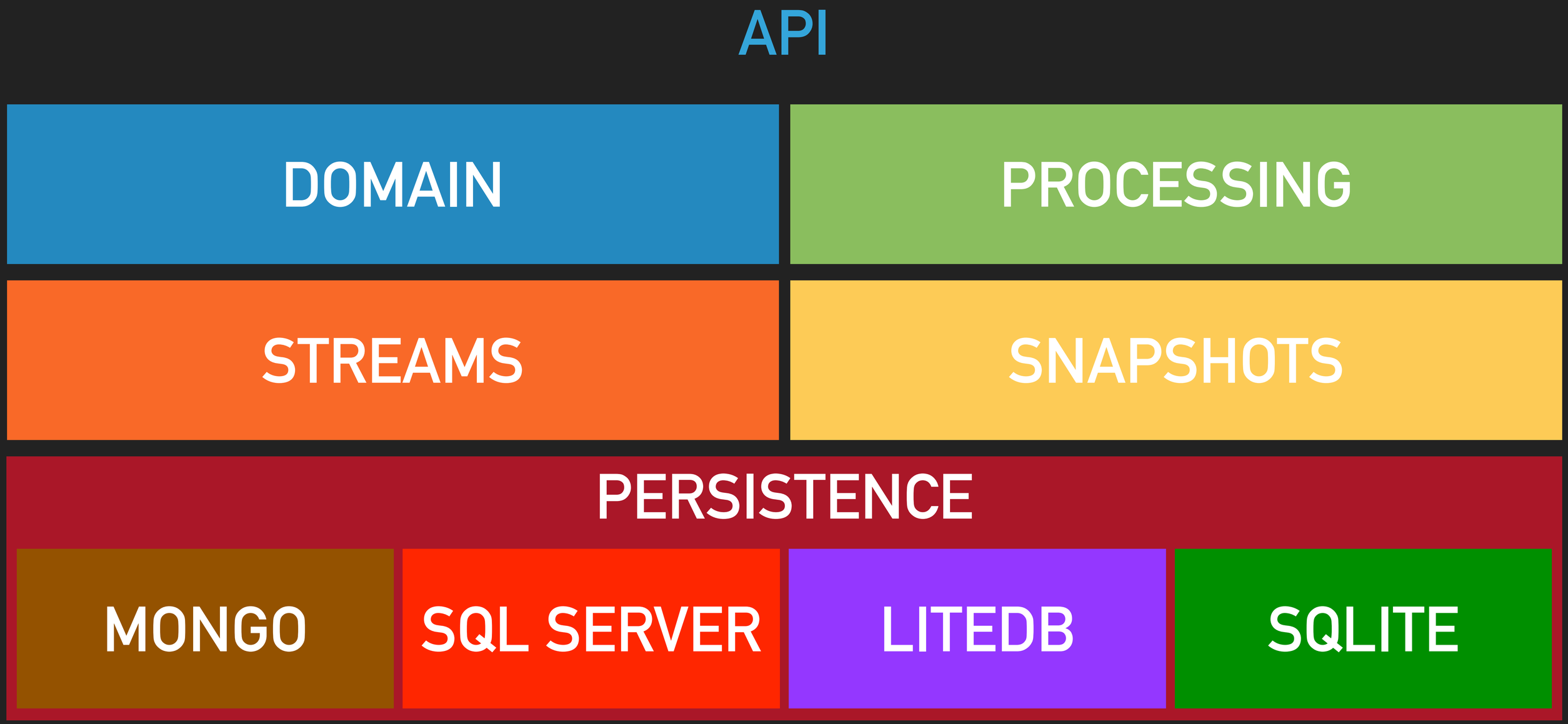

PROBLEM
INPUT
RULE
FLOW
STATE
FUNCTIONS
DATA
OUTPUT
CLIENT
VIEW
CODE
COMMAND
INVARIANT
PROCESS MANAGER
STATE
AGGREGATE
EVENTSTORE
EVENT
SUBSCRIPTION
PROJECTION

PATTERN

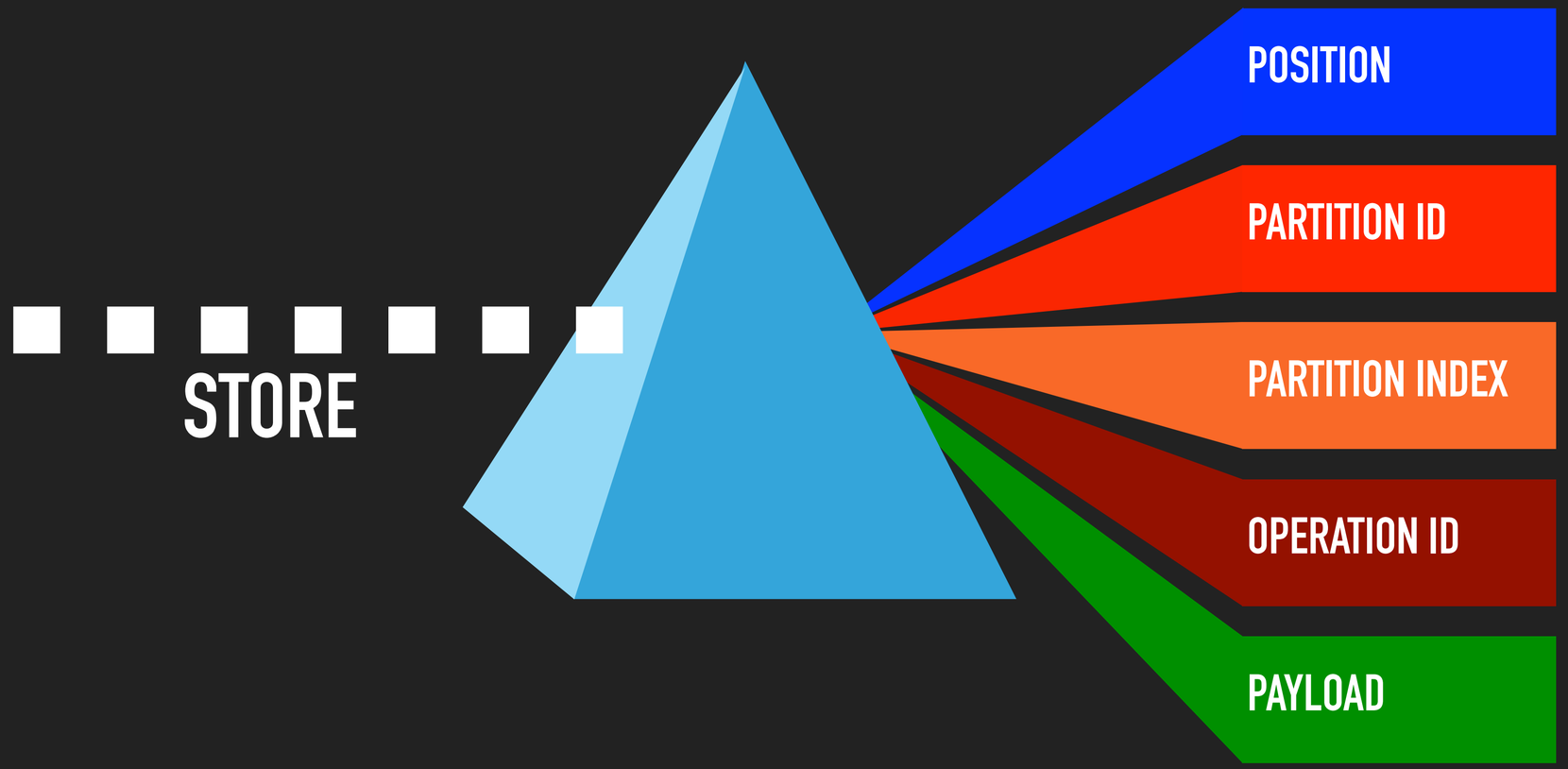

Command
Aggregate
Process Manager
Event
Event
Read Model
Read Model
Projections

(OPINIONATED) EVENT SOURCING LIBRARY

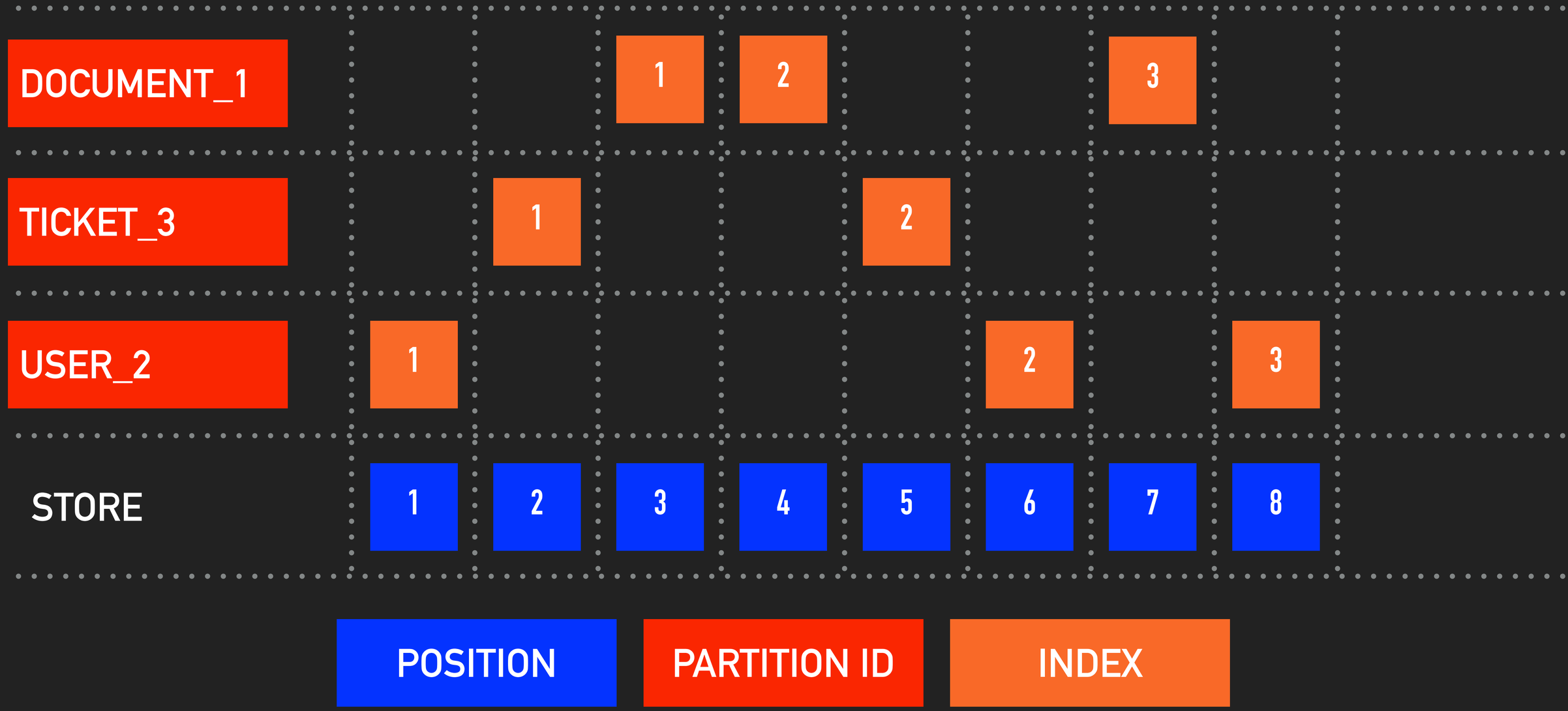
NSTORE


API
DOMAIN
PROCESSING
STREAMS
SNAPSHOTS
PERSISTENCE
MONGO
SQL SERVER
LITEDB
SQLITE

NSTORE LAYER 0 API - PERSISTENCE DESIGN



NSTORE LAYER 0 API - PERSISTENCE LAYOUT



```
namespace NStore.Core.Persistence
{
    /// Chunk is the atomic element of the store.
    /// The store is a global ordered sequence of chunks
    public interface IChunk
    {
        /// Global position
        long Position { get; }

        /// Partion / stream / aggregate id
        string PartitionId { get; }

        /// Chunk index for a given partition
        long Index { get; }

        /// Data
        object Payload { get; }

        /// Idempotency Key, unique for a given partition / stream / aggregate
        string OperationId { get; }
    }
}
```

https://github.com/nstoredev/NStore/blob/develop/src/NStore.Core/Persistence/IChunk.cs

NSTORE LAYER 1 API - STREAMS

```
public record Favorited(string UserId, DateTime When);

static async Task Main()
{
    // StreamsFactory setup
    var streams = new StreamsFactory(new InMemoryPersistence());

    // Open the stream in r/w
    var post = streams.Open("post/123");

    // Write to stream
    await post.AppendAsync(new Favorited("users/200", DateTime.UtcNow));
    await post.AppendAsync(new Favorited("users/404", DateTime.UtcNow));
    await post.AppendAsync(new Favorited("users/200", DateTime.UtcNow));

    // Read the stream from start
    await post.ReadAsync(chunk =>
    {
        Console.WriteLine($"{chunk.PartitionId} #{chunk.Index} => {chunk.Payload}");
        return Subscription.Continue;
    });
}
```

NSTORE LAYER 1 API - STREAM PROCESSING

```
// Aggregation logic
public class UniqueFavs
{
    private readonly HashSet<string> _users = new();
    public int Count => _users.Count;

    private void On(Favorited fav) => _users.Add(fav.UserId);
}

// Stream processing
var favs = await post.AggregateAsync<UniqueFavs>();
Console.WriteLine($"{favs.Count} users added '{post.Id}' as favorite");
```

NSTORE LAYER 2 API - DOMAIN

```
public class Room : Aggregate<RoomState>
{
    public void EnableBookings()
    {
        if(!this.State.BookingsEnabled)
            Emit(new BookingsEnabled(this.Id));
    }

    public void AddBooking(DateRange dates)
    {
        if (State.IsAvailableOn(dates)){
            Emit(new RoomBooked(this.Id, dates));
        } else {
            Emit(new BookingFailed(this.Id, dates));
        }
    }
}
```

NSTORE LAYER 2 API - DOMAIN

```
public class RoomState
{
    private IList<DateRange> _reservations = new List<DateRange>();
    public bool BookingsEnabled { get; private set; }

    private void On(BookingsEnabled e)
    {
        this.BookingsEnabled = true;
    }

    private void On(RoomBooked e)
    {
        _reservations.Add(e.Dates);
    }

    public bool IsAvailableOn(DateRange range)
    {
        return !_reservations.Any(range.Overlaps);
    }
}
```

NSTORE LAYER 2 API - DOMAIN

```
var repository = new Repository(
    AggregateFactory,
    Streams,
    Snapshots
);

var room = await repository.GetByIdAsync<Room>("room/2");

room.EnableBookings();

await repository.SaveAsync(room, “init");
```

NSTORE LAYER 2 API - DOMAIN

```
{
    "_id": NumberLong(1),
    "PartitionId": “room/2",
    "Index": NumberLong(1),
    "Payload": {
        "_t": "Changeset",
        "Events": [
            {
                "_t": "BookingsEnabled",
                "_id": “room/2"
            }
        ],
        "AggregateVersion": NumberLong(1),
        "Headers": {}
    },
    "OperationId": "init"
}
```