LINQ Questions

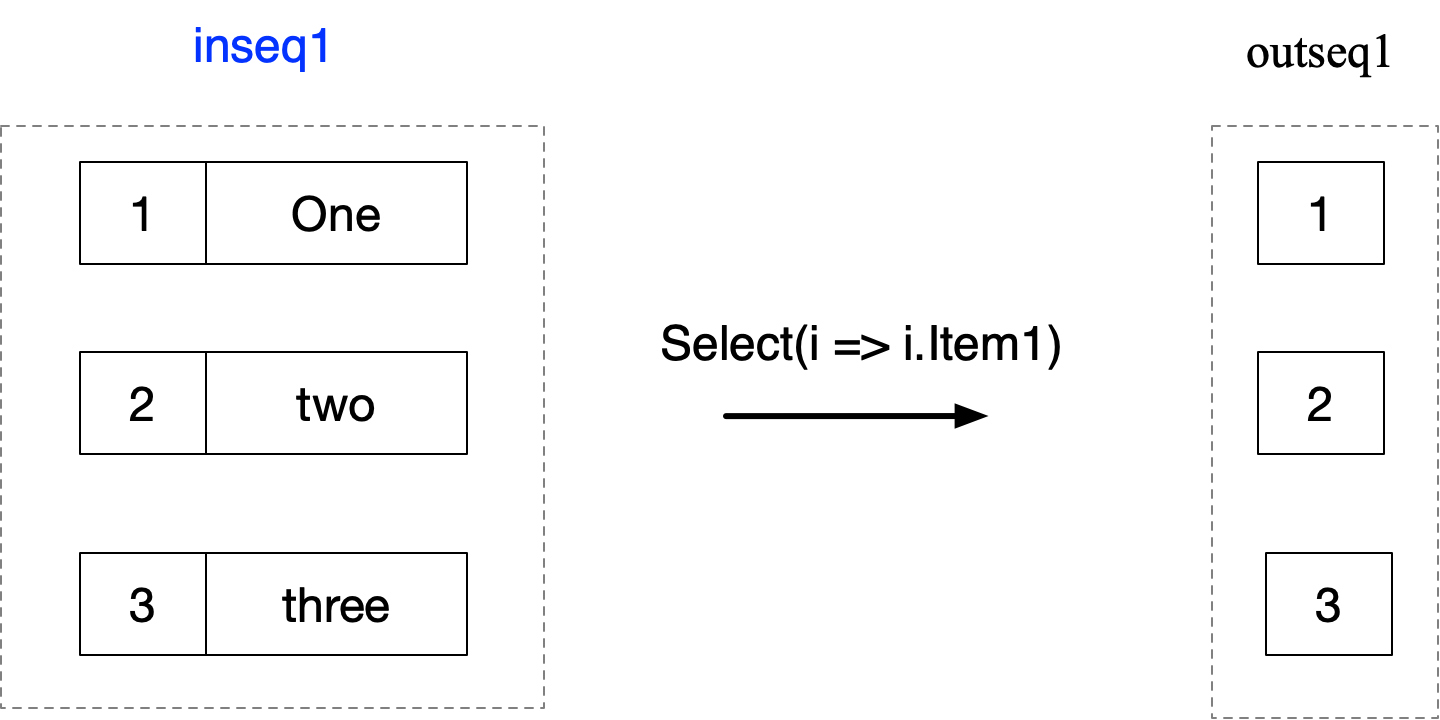
Queries

Hello

Basic Projection

Write query and fluent syntax to perform the following?

var inseq1 = new [] {(1, "one"),(2, "two"),(3, "three")};



var f = inseq1.Select(i => i.Item1);

f.Dump();

var q = from n in inseq1 select n.Item1;

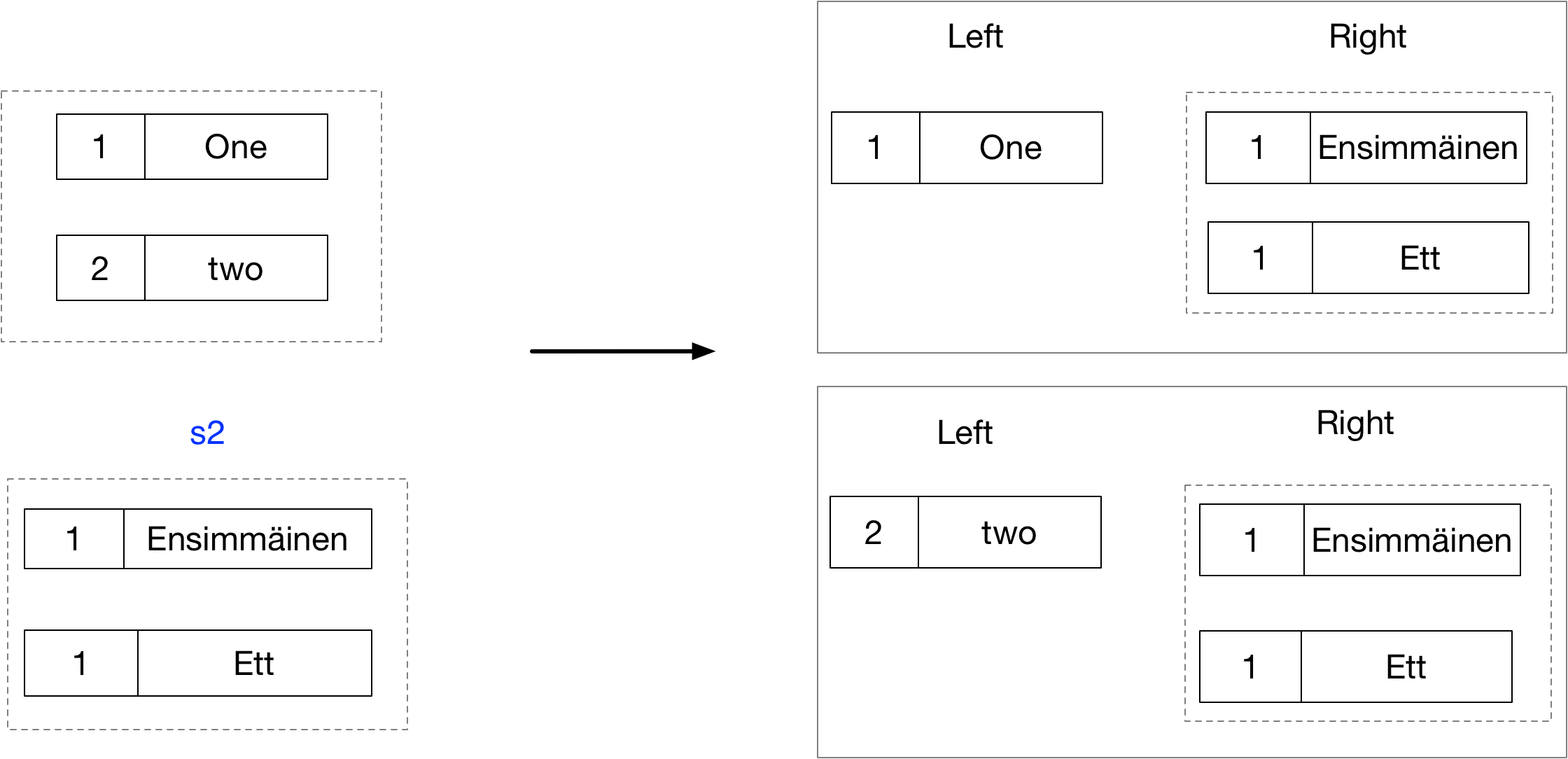
q.Dump();

Cross Product - Hierarchical

Write query and fluent syntax to perform the following?

var s1 = new[] { (1, "one"), (2, "two"), };

var s2 = new[] { (1, "Ensimmäinen"), (1, "Ett") };



var f =s1

.Select(e1 => new {Left=e1, Right=s2});

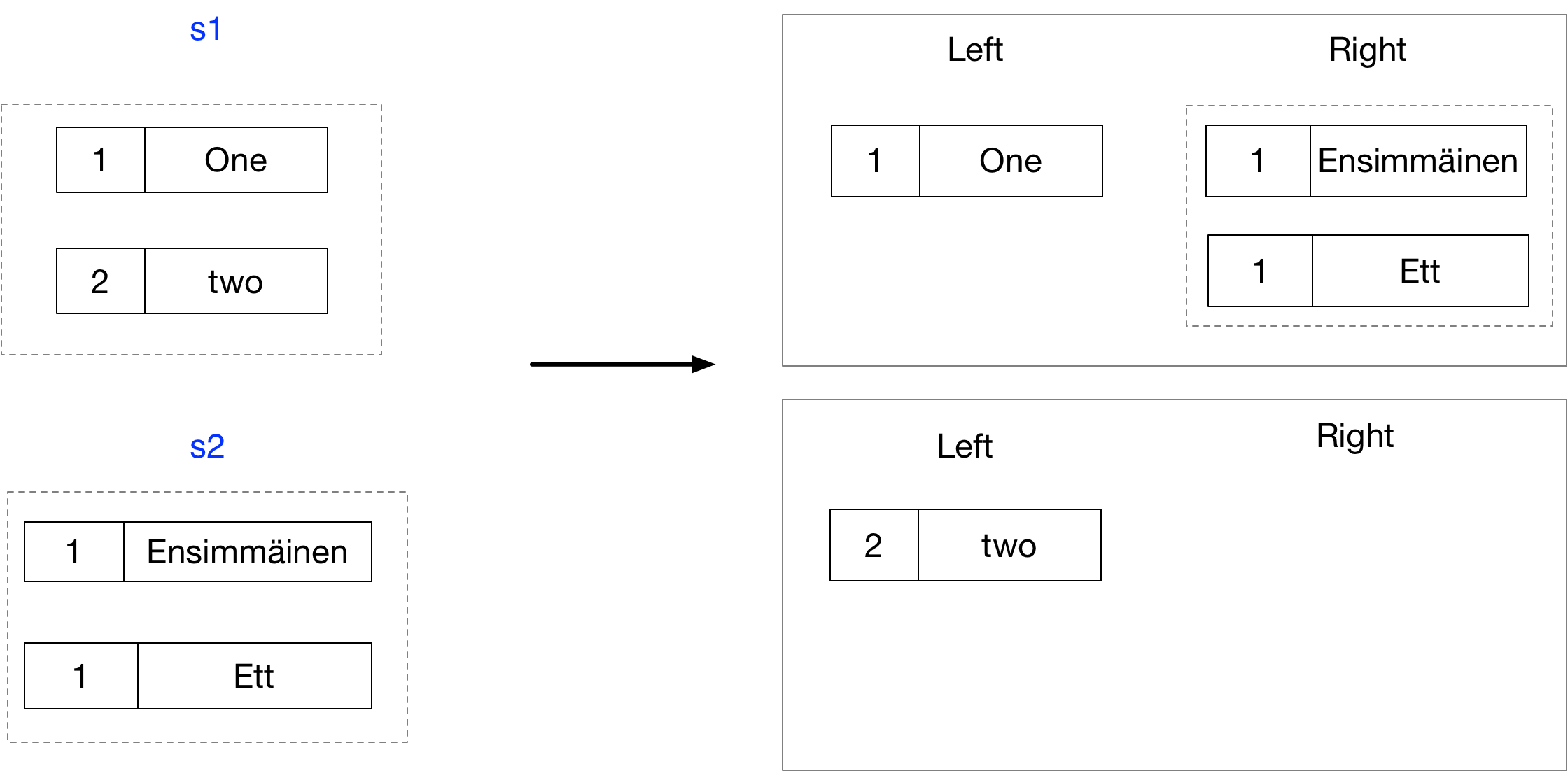
var q = from e1 in s1

select new {Left=e1, Right=s2};

Left Outer Join- Hierarchical

Write query and fluent syntax to perform the following. Write inefficient select based queries, efficient GroupJoin queries, and efficient select+lookup queries?

var s1 = new[] { (1, "one"), (2, "two"), };

var s2 = new[] { (1, "Ensimmäinen"), (1, "Ett") };

// Inefficient because the inner query iterated once per outer element

var f1 = s1.Select(e1 => new { Left = e1, Right = s2.Where(e2 => e1.Item1 == e2.Item1)});

f1.Dump();

// Inefficient because the inner query iterated once per outer element

var q1 = from e1 in s1

select new {Left=e1, Right = from e2 in s2 where e1.Item1 == e2.Item1 select e2 };

q1.Dump();

// Efficient because internally Group Join uses a lookup

var f2 = s1.GroupJoin(s2,e1=>e1.Item1, e2=> e2.Item1,(e1,c) => new {Left=e1, Right =c });

f2.Dump();

// Efficient because internally Group Join uses a lookup

var q2 = from e1 in s1

join e2 in s2 on e1.Item1 equals e2.Item1

into collection

select new {Left=e1, Right =collection };

q2.Dump();

// Efficient - uses lookup

var lookup = s2.ToLookup(e2 => e2.Item1, e2=>e2);

var f3 = s1.Select(e1 => new {Left=e1, Right=lookup[e1.Item1]});

f3.Dump();

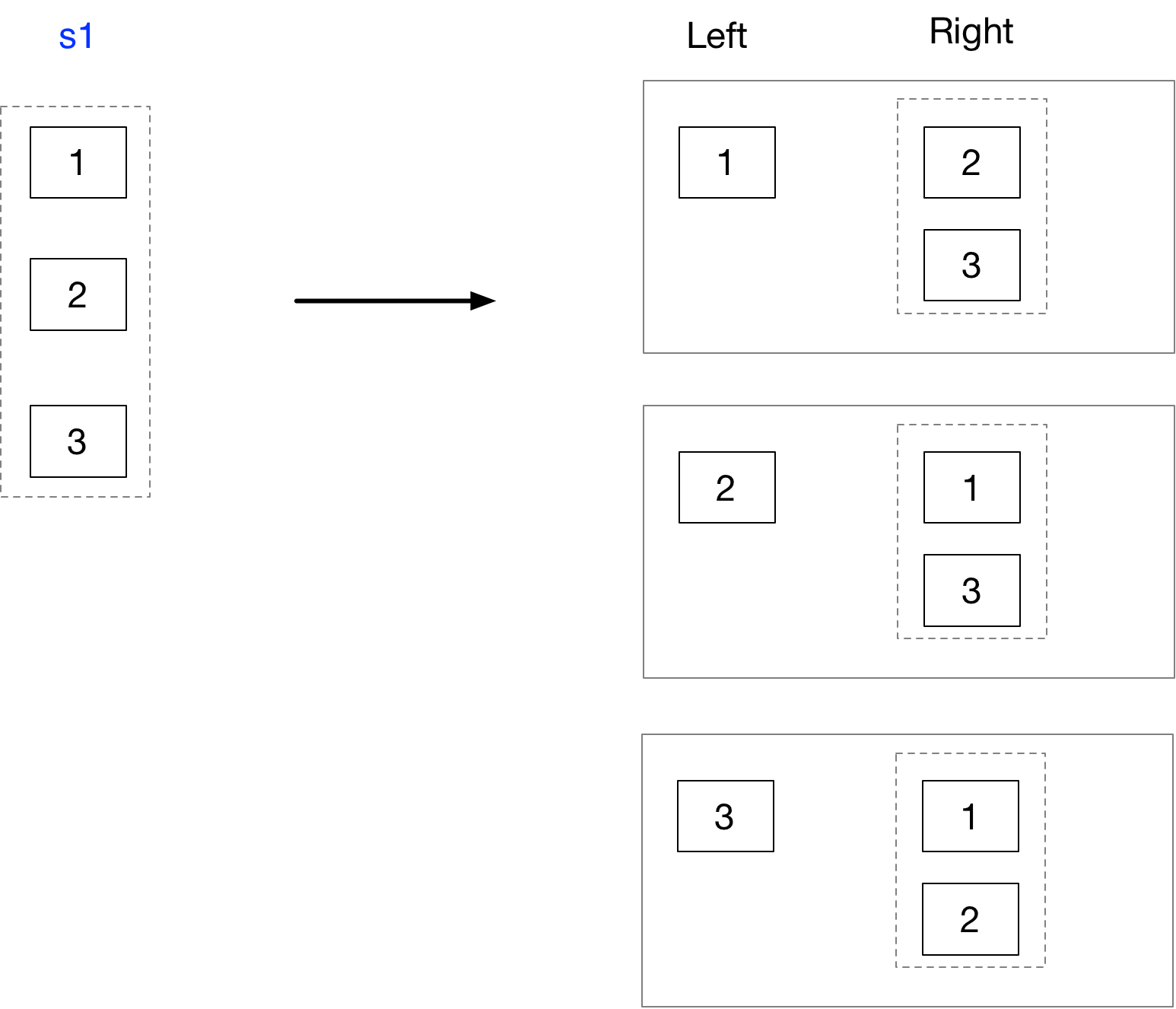
// Efficient - uses lookup

var q3 =from e1 in s1

select new {Left=e1, Right=lookup[e1.Item1]};

Non Equijoin – Hierarchical

Write query and fluent syntax to perform a non-equi join as follows?

var s1 = new[] { 1, 2, 3};

var q1 = from e1 in s1

select new {

Left=e1,

Right=from e2 in s1

where e2 != e1

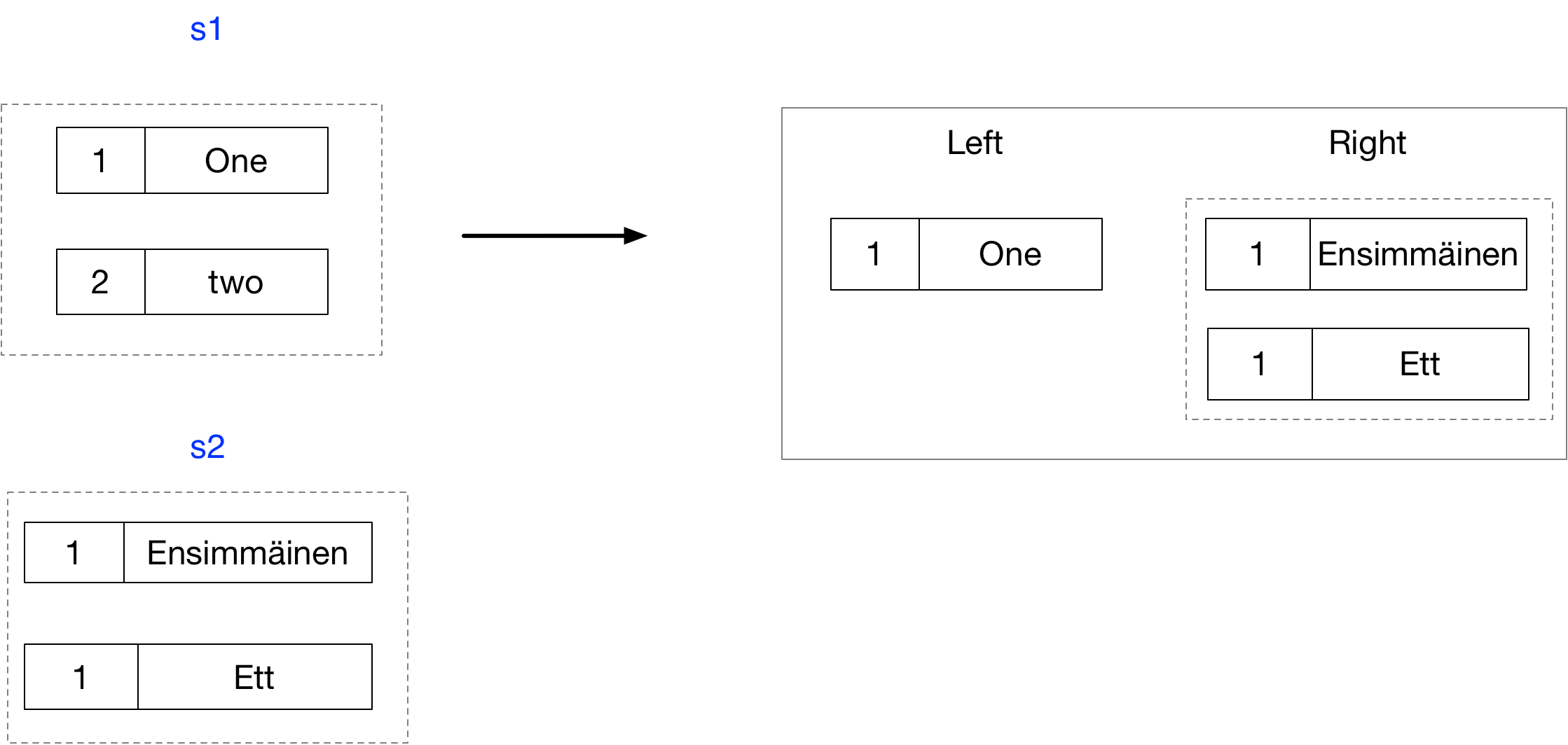
select e2

};

var f1 = s1.Select(e1 => new { Left = e1, Right = s1.Where(e2 => e1 != e2) });

Inner Join – Hierarchical

Write query and fluent syntax to perform a non-equi join as follows?



// Inefficient

var f1 = s1

.Select(e1 => new {Left = e1, Right=s2.Where(e2=>e1.Item1 == e2.Item1)})

.Where(s => s.Right.Any());

// Inefficient

var q1 = from e1 in s1

select new {Left = e1, Right=s2.Where(e2=>e1.Item1 == e2.Item1)}

into r

where r.Right.Any()

select r;

// GroupJoin

var f2 =s1

.GroupJoin(s2, e1=>e1.Item1, e2=>e2.Item1,(e,si)=>new {Left = e, Right=si})

.Where(s => s.Right.Any());

// Group Join

var q2 = from e1 in s1

join e2 in s2 on e1.Item1 equals e2.Item1

into c

select new {Left = e1, Right=c}

into d

where d.Right.Any()

select d;

// Efficient using select and lookup

var lookup = s2.ToLookup(s => s.Item1);

var f3 = s1

.Select(e1 => new { Left = e1, Right = lookup[e1.Item1] })

.Where(s => s.Right.Any());

f3.Dump();

// Efficient using select and lookup

var q3 = from e1 in s1

select new { Left = e1, Right = lookup[e1.Item1] }

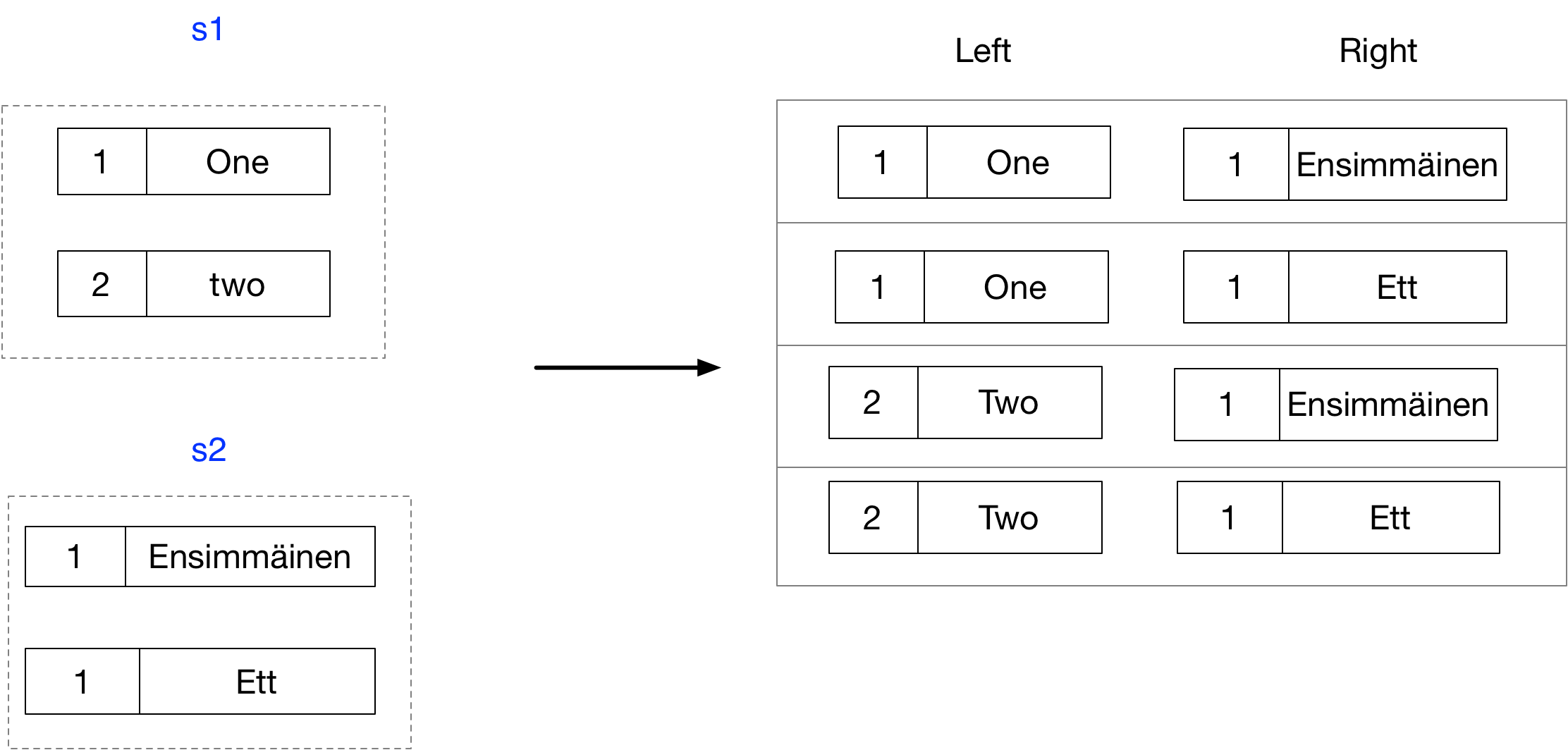
into r

where r.Right.Any()

select r;

Cross Product - Flat

Write query and fluent syntax to perform the following?



var s1 = new[] { (1, "one"), (2, "two"), };

var s2 = new[] { (1, "Ensimmäinen"), (1, "Ett") };

var f =s1

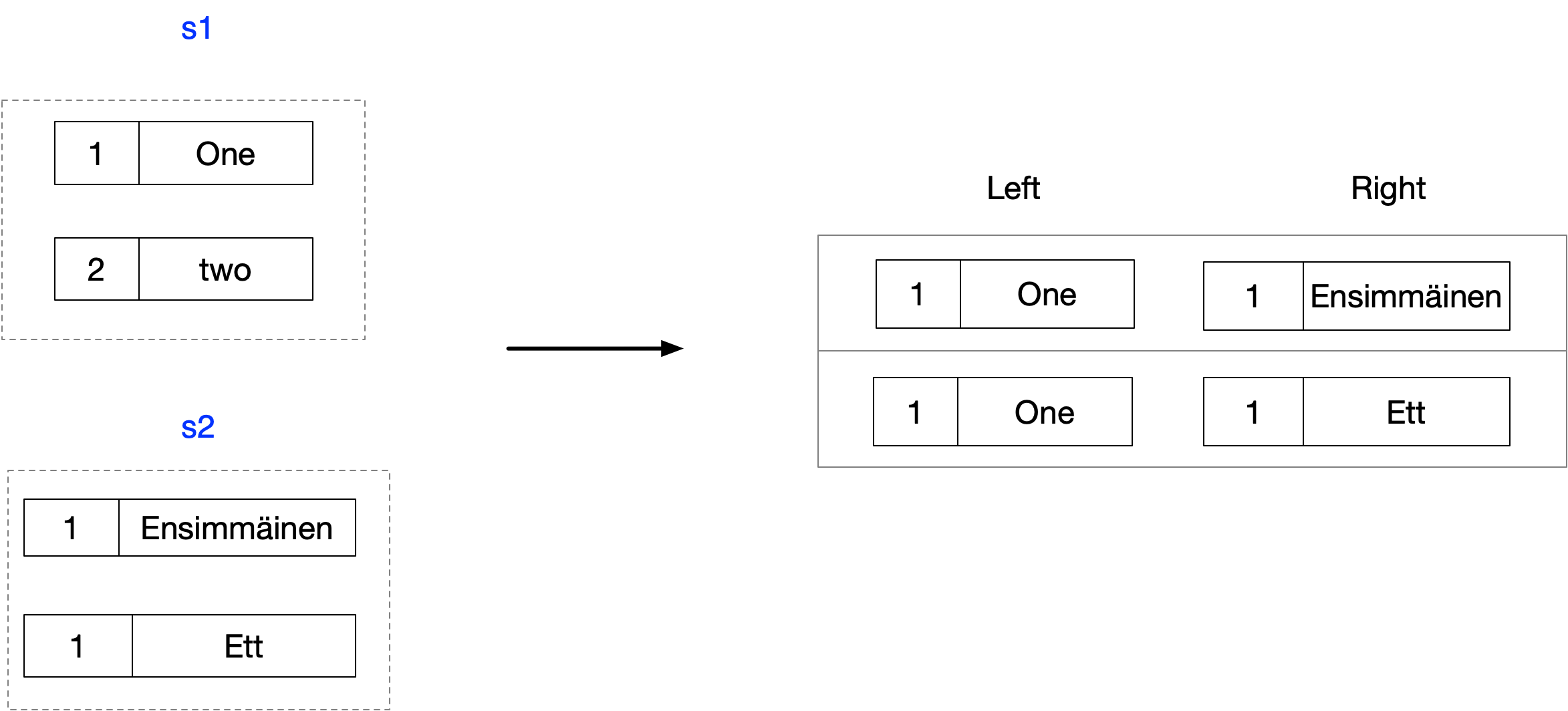
.Select(e1 => new {Left=e1, Right=s2});

var q = from e1 in s1

select new {Left=e1, Right=s2};

Inner Join - Flat

Write query and fluent syntax to perform the following.?



// Fluent Syntax - Inefficent Group Join

var f1 = s1

.SelectMany(e1 => s2, (e1, e2) => new { e1,e2})

.Where(r => r.e1.Item1 == r.e2.Item1)

.Select(r => new {Left=r.e1, Right=r.e2});

// Query Syntax - Inefficent Group Join

var q1 = from e1 in s1

from e2 in s2

where e1.Item1 == e2.Item1

select new {Left=e1, Right=e2};

//q1.Dump();

// Fluent - Efficient, using Join which internally uses a lookup

var f2 = s1.Join(s2, e1=>e1.Item1, e2=>e2.Item1, (e1,e2)=>new {Left=e1, Right=e2});

// Query - Efficient, using Join which internally uses a lookup

var q2 = from e1 in s1

join e2 in s2 on e1.Item1 equals e2.Item1

select new {Left=e1, Right=e2};

// Fluent - Efficient using SelectMany and Lookup

var lookup = s2.ToLookup(e1 => e1.Item1);

var f3 = s1

.SelectMany(e1 => lookup[e1.Item1], (e1, e2) => new { e1, e2 })

.Select(r => new { Left = r.e1, Right = r.e2 });

// Query Syntax - Efficient using SelectMany and lookup

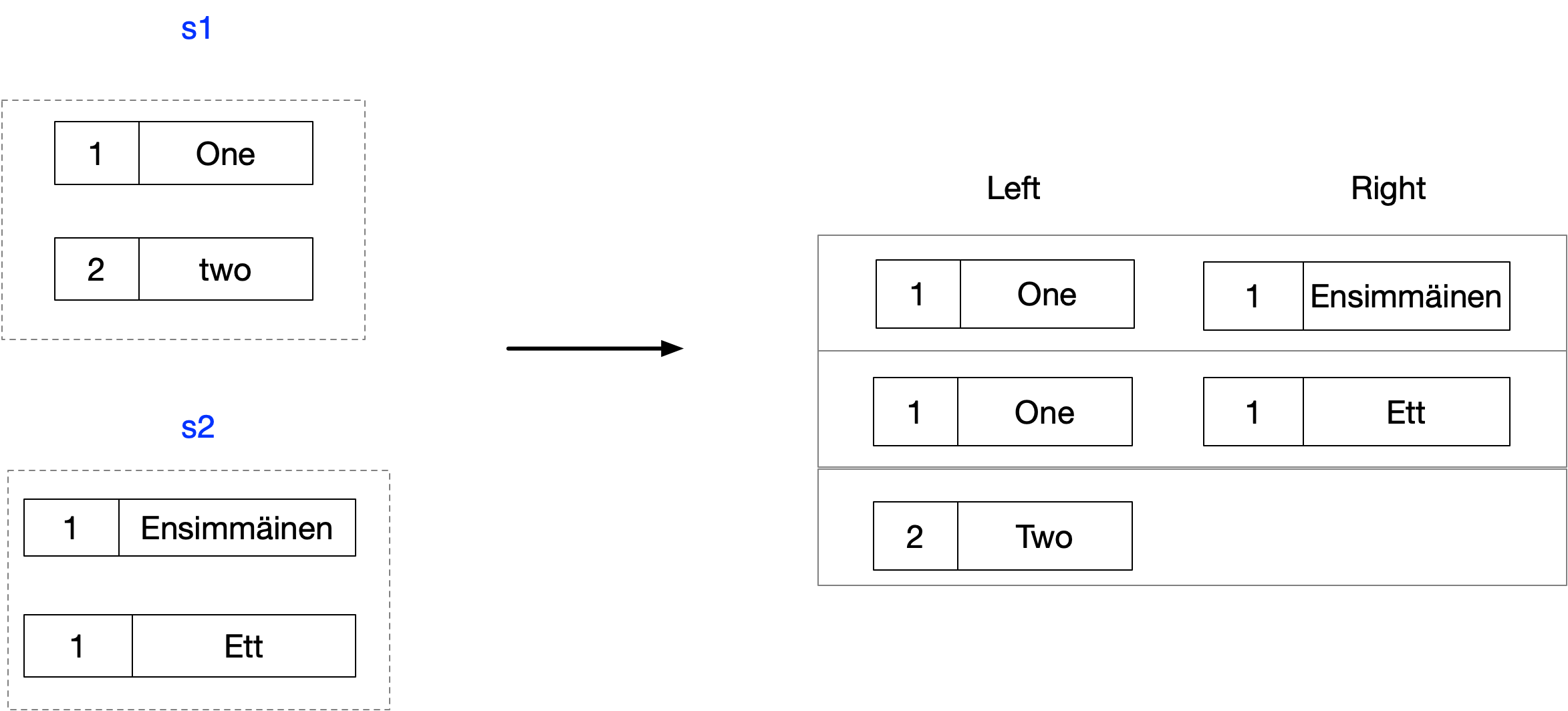
var q3 = from e1 in s1

from e2 in lookup[e1.Item1]

select new { Left = e1, Right = e2 };

Left Outer Join - Flat

Write query and fluent syntax to perform the following.?



/\* First we show how the mechanics work of first doing a group join

\* and then a SelectMany \*/

IEnumerable<((int, string) leftEl, IEnumerable<(int, string)> rightMatches)> joinResults

= Enumerable.GroupJoin(

/\* Left Sequence \*/ s1,

/\* Right Sequence \*/ s2,

/\* Left Key Selector \*/ leftEl => leftEl.Item1,

/\* Right Key Selector \*/ rightEl => rightEl.Item1,

/\* Result Selector \*/ (leftEl, rightMatches) => (leftEl, rightMatches));

IEnumerable<(string Left, string Right)> flattenedResults =

Enumerable.SelectMany(

/\* Source Sequence \*/ joinResults,

/\* Collection Selector \*/ jointElement => jointElement.rightMatches.DefaultIfEmpty(),

/\* Result Selector \*/

(tuple, valueTuple) =>

(Left: tuple.leftEl.Item2, Right: valueTuple.Item2));flattenedResults.Dump();

// Concise Fluent Syntax

IEnumerable<(string Left, string Right)> f1 = s1

.GroupJoin(s2, e1 => e1.Item1, e2 => e2.Item1, (leftEl, rightMatches)

=> (leftEl, rightMatches))

.SelectMany(jointElement

=> jointElement.rightMatches.DefaultIfEmpty(),

(tuple, valueTuple) => (Left: tuple.leftEl.Item2, Right: valueTuple.Item2));

// Concise Query Syntax

IEnumerable<(string Left, string Right)> q2 =

from leftEl in s1

join rightEl in s2 on leftEl.Item1 equals rightEl.Item1 into matches

from match in matches.DefaultIfEmpty()

select (Left: leftEl.Item2, Right: match.Item2);

## Basics

**What is LINQ?**

A language feature that enables us to write type safe queries over any collection that implements IEnumerable<T>

**What inspired LINQ?**

The functional programming paradigm

**What are the basic elements**

* Sequences
* Elements
* Query operators
* Queries

**What do lambda expressions in query operators always operate on?**

Individual elements

**Do query operators alter the input sequence?**

No, they always generate a new sequence

**What does LINQ query comprise?**

A pipeline of operators that accept and return ordered sequences

**What does an SQL query comprise?**

A netwoek of clauses working on unordered sets

**How is deferred execution implemented?**

Query operators provide deferred execution by returning decorator sequences.

**What are the advantages of deferred execution?**

* Decouples construction from execution
* Allows one to construct a query in multiple steps
* You can re-evaluate a query by enumerating it again.

**What are the exceptions that return immediately?**

ToList, ToArray, ToDictionary, ToLookup

Single element or scalar operators such as First or Count

**How do decorator sequences differ from traditional collection classes?**

In general a decorator sequence has no storage of its own to store elements

**What does it have instead?**

A reference to another sequence supplied at runtime

**What happens when you request data from a decorator?**

It must in turn ask for data from its wrapped input sequence

**What happens when you chain query operators?**

A chain of decorators are created

**What happens when you enumerate a query?**

You query the original input sequence transformed through a layering chain of decorators

**What happens if you call ToList() on query?**

The whole chain is collapsed into a single list

emp + b);}