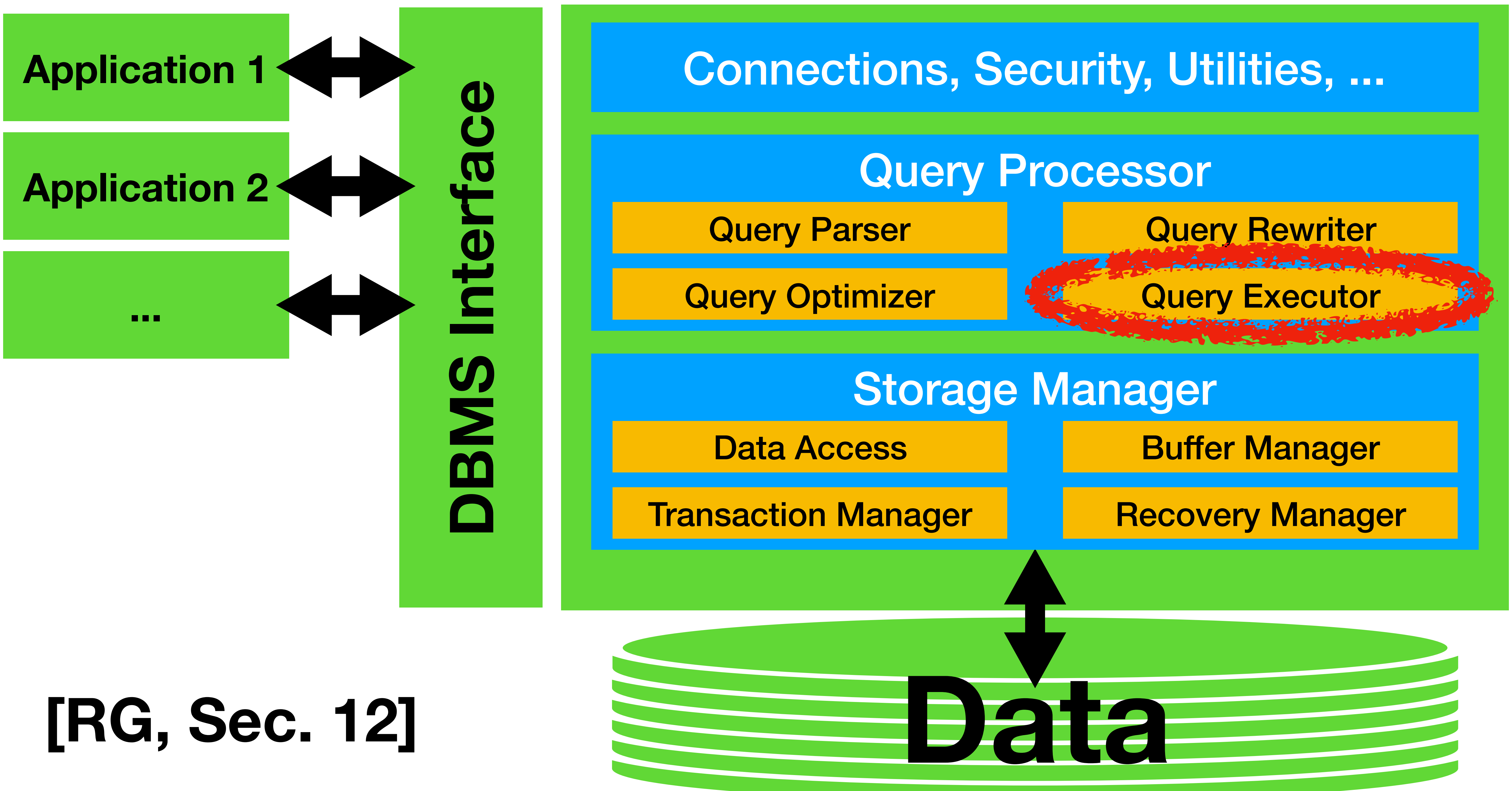
Query Processing Overview

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Database Management Systems (DBMS)



[RG, Sec. 12]

Outlook

- Will discuss how to implement **standard operators**
- Often have **multiple** implementations of same operator
- Can choose **most efficient** implementation at each point
- Will also discuss about **cost estimation**
  - Assumption: cost ~ **number of pages** read/written

Filter Operator (σ)

How to Filter?

- Want to retrieve table rows satisfying a **predicate**
- Simplest option: **scan** all pages, check each entry
- Option if sorted: **binary search** for specific predicates
- Can use **indexes** if available (right table, right key)

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes

Calculate scan cost!

Nr. Entries/Page	?
Nr. Enrollment Pages	?
Total Scan Cost	?

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes

Calculate scan cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Total Scan Cost	6,000

Calculations

- 1000 Bytes per Page/10 Bytes per Entry =
  100 Entries per Page
- 60,000 Students * 10 Enrollments per Student =
  600,000 Enrollments
- 600,000 Enrollments / 100 Entries per Page =
  6,000 Enrollment Pages
- Read Each Enrollment Page Once:
  Total Cost 6,000

What About Output Cost?

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Courses	100
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes

Sorted by CID - Calculate binary search cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Nr. Search Steps	?
Nr. Pages to Scan	?
Total Cost	?

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Courses	100
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes

Sorted by CID - Calculate binary search cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Nr. Search Steps	13
Nr. Pages to Scan	60
Total Cost	~ 73

Where Did We Simplify?

Calculations

- Maximal steps of binary search:
  Ceil(Log2(6,000)) = 13
- 600,000 enrollments partitioned over 100 courses
  - Makes 6,000 enrollments per course (if uniform)
- Search first, then scan all qualifying pages:
  Total cost is 73

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Courses	100
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes
Index Fanout	100

Tree Index with Data on CID - Calculate Access Cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Nr. Inner Node Visits	?
Nr. Leaf Node Visits	?
Total Cost	?

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Courses	100
Nr. Enrollments/Student	10
Size/Enrollment Entry	10 Bytes
Bytes/Page	1,000 Bytes
Index Fanout	100

Tree Index with Data on CID - Calculate Access Cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Nr. Inner Node Visits	2
Nr. Leaf Node Visits	60
Total Cost	62

Calculations

- Tree index root node has 100 children (fanout)
  - 100^2 = 10,000 grand children
- Tree has height 3, need to read 2 inner nodes
- Read results from leaf nodes containing data
  - We have 60 result pages (see before)
- Total cost: 62 pages

Costing Example

SELECT * FROM Enrollment WHERE CID='CS4320'

Property	Value
Nr. Students	60,000
Nr. Courses	100
Nr. Enrollments/Student	10
Size/Entry	10 Bytes
Bytes/Page	1,000 Bytes
Index Fanout	100

Unclustered Tree Index on CID - Calculate Access Cost!

Nr. Entries/Page	100
Nr. Enrollment Pages	6,000
Nr. Inner Node Visits	2
Nr. Leaf Node Visits	60
Nr. Data Pages Read	6,000
Total Cost	6,062

Calculations

- Need to read two inner tree nodes (same as before)
- Leaf nodes now contain references, not data directly
  - Need to read 60 pages of references (same entry size)
- Also, need to read data pages for 6,000 entries
  - Pessimistically assume that each on a different page
  - Hence, need to add 6,000 page reads to total cost

Example Summary

Scan Cost	6,000
Binary Search	73
Index with Data	62
Unclustered Index	6,062

Insights

- Index or sort orders **can** speed up filtering
- However, may **not always** be more efficient
- Need to **calculate cost** of alternatives and compare
    - This is what the **query optimizer** does ...

Multi-Predicate Filtering

- May have to retrieve entries satisfying **two predicates**
- **Scanning** all pages always works
- Can **use index** for first predicate, then **check** second
- Could **merge results** from two indices for both predicates

Join Operators ($\bowtie$)

Join Operators

- Often one of the **most expensive** operations
- Lots of research on different **join operators**
- Some are **more generic** and apply to any join predicate
- Some are **faster** in specific situations
- Some need **less memory** than others
- ...

Page Nested Loop Join

- Load one page after the other from **first (outer) table**
- **For each page** from outer table:
    - Load one page after the other from **second table**
    - For all tuples in memory: **check** and **add** to result

Notations

- **LoadPage**(P): Load page P
- **Pages**(T): Pages of table T
- **Tuples**(P): Tuples of page P

Page Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
 LoadPage(ep)
 For sp in Pages(S):
  LoadPage(sp)
  For et in Tuples(ep), st in Tuples(sp):
   If (et.Sid=st.Sid):
    Output(et ⋈ st)
```

Page Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
 LoadPage(ep)          <- For each page in E
 For sp in Pages(S):
  LoadPage(sp)
  For et in Tuples(ep), st in Tuples(sp):
   If (et.Sid=st.Sid):
    Output(et ⋈ st)
```

For each page in E

Cost = pages in E * load cost

Page Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
  LoadPage(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E (annotation for LoadPage(ep))

For each page in E and each page in S (annotation for LoadPage(sp))

Cost = pages in E * load cost + pages in E * pages in S * load cost

Page Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
  LoadPage(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E (for LoadPage(ep))

For each page in E and each page in S (for LoadPage(sp))

Cost = pages in E * load cost + pages in E * pages in S * load cost + tuples in E * tuples in S * evaluation cost

Page Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
  LoadPage(ep)          ← For each page in E
  For sp in Pages(S):
    LoadPage(sp)        ← For each page in E and each page in S
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

Cost = pages in E * load cost +
pages in E * pages in S * load cost +
tuples in E * tuples in S * evaluation cost

How Much Memory?

- Need space to store current **page from outer table**
- Need space to store **current page from inner table**
- Need **one buffer page** to store output (before disk write)

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Page Nested Loop Cost (Using Enrollment as Outer!)	?

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Page Nested Loop Cost (Using Enrollment as Outer!)	1,000+100 * 1,000 = 101,000

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Page Nested Loop Cost (Using Enrollment as Outer!)	1,000+100 * 1,000 = 101,000

Easy Improvement ... ?

How to Improve Join Operator?

Block Nested Loop Join

- **Page** nested loop: read inner table for each outer **page**
- **Block** nested loop: read inner table for each outer **block**
  - More efficient as block contains **multiple** pages

More Notations

- **PageBlocks**(T, b): Blocks of b pages from T
- **LoadPages**(B): Load pages from block B

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
 LoadPages(ep)
 For sp in Pages(S):
  LoadPage(sp)
  For et in Tuples(ep), st in Tuples(sp):
   If (et.Sid=st.Sid):
    Output(et ⋈ st)
```

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
  LoadPages(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
  LoadPages(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E

Cost = pages in E * load cost

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
  LoadPages(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E

For each block in E and each page in S

Cost = pages in E * load cost + blocks in E * pages in S * load cost

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
  LoadPages(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E

For each block in E and each page in S

*Cost = pages in E * load cost + blocks in E * pages in S * load cost*

Block Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in PageBlocks(E, b):
  LoadPages(ep)
  For sp in Pages(S):
    LoadPage(sp)
    For et in Tuples(ep), st in Tuples(sp):
      If (et.Sid=st.Sid):
        Output(et ⋈ st)
```

For each page in E

For each block in E and each page in S

*Cost = pages in E * load cost + blocks in E * pages in S * load cost*

How Much Memory?

- Need enough space to store **blocks from outer relation**
- Need space to store **one page from inner** relation
- Need one **page to store output** (before writing to disk)

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Buffer for Outer Blocks	10
Block Nested Loop Cost (Using Enrollment as Outer!)	?

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Buffer for Outer Blocks	10
Block Nested Loop Cost (Using Enrollment as Outer!)	1,000+1,000/10 * 100 = 11,000

Index Nested Loop Join

- Idea: have **index on join column** and equality predicate
- **Iterate over pages** of non-indexed (outer) table
- For each outer tuple, **use index** to find matching tuples

More Notations

- **Index**(Predicate): Entries satisfying predicate
- **Tuple**(P, i): i-th tuple on page P

Index Nested Loop Join

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
  LoadPage(ep)
  For et in Tuples(ep):
    For <sp, i> in Index(et.Sid=st.Sid):
      LoadPage(sp)
      Output(et ⋈ Tuple(sp, i))
```

Index Nested Loop Join

$$\bowtie_{E.Sid=S.Sid}$$

```
For ep in Pages(E):
  LoadPage(ep)
  For et in Tuples(ep):
    For <sp, i> in Index(et.Sid=st.Sid):
      LoadPage(sp)
      Output(et ⋈ Tuple(sp, i))
```

Cost = pages in E* load cost + index entries * load cost

How Much Memory?

- Need one page to store current **page from outer table**
- Need one page to store **current page from inner table**
- Need one page as **output buffer** (before disk write)

Alternatives for Equality Joins?

Hash Join

- Want tuples with **same value** in join column
- Same value in join column implies **same hash** value
- Join **Phase 1**
    - **Partition data** by hash values in join columns
        - Make partitions small enough to **fit into memory**
- Join **Phase 2**
    - Join each partition pair (same hash value) **separately**

More Notations

- **Hash**(Tuple): Calculates hash function for tuple
- **Full**(P): Whether page P has no more space left
- **WriteAndClear**(P): Write P to disk and erase

Hash Join: Phase 1

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
  LoadPage(ep)
  For et in Tuples(ep):
    Add et to EB[Hash(et)]
      If (Full(EB[Hash(et)])):
        WriteAndClear(EB[Hash(et)])
```

Hash Join: Phase 1

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
 LoadPage(ep)
 For et in Tuples(ep):
  Add et to EB[Hash(et)]
   If (Full(EB[Hash(et)])):
    WriteAndClear(EB[Hash(et)])
```

← ***For each page in E***

Hash Join: Phase 1

$\bowtie_{E.Sid=S.Sid}$

For ep in Pages(E):

LoadPage(ep) ← *For each page in E*

For et in Tuples(ep):

Add et to EB[Hash(et)]

If (Full(EB[Hash(et)])):

WriteAndClear(EB[Hash(et)]) ← *For each page in E*

Hash Join: Phase 1

$\bowtie_{E.Sid=S.Sid}$

```
For ep in Pages(E):
 LoadPage(ep)
 For et in Tuples(ep):
  Add et to EB[Hash(et)]
   If (Full(EB[Hash(et)])):
    WriteAndClear(EB[Hash(et)])
```

← *For each page in E* (LoadPage)

← *For each page in E* (WriteAndClear)

Cost = pages in E* IO cost * 2

Hash Join: Phase 1

$\bowtie_{E.Sid=S.Sid}$

```
For sp in Pages(S):
  LoadPage(sp)  ← For each page in S
  For st in Tuples(sp):
    Add st to SB[Hash(st)]
     If (Full(SB[Hash(st)])):
      WriteAndClear(SB[Hash(st)])  ← For each page in S
```

Cost = pages in S* IO cost * 2

Hash Join: Phase 2

$\bowtie_{E.Sid=S.Sid}$

```
For h in Hash Values:
 LoadPages(EB[h])          ← For each page in E
 For sp in Pages(SB[h]):
  Load(sp)                 ← For each page in S
  For ep in Pages(EB[h]), st in sp, et in ep:
   If (et.Sid=st.Sid):
    Output(et ⋈ st)
```

Cost = (pages in E in S) * IO cost

How Much Memory?

- **Phase 1**
  - Space to store **current page** read for partitioning
  - Store one buffer page for **each hash bucket**
- **Phase 2**
  - Store all pages from **one hash bucket**
  - Store **current page** from other table bucket
  - One **output buffer** page

How Many Buckets?

- **Constraint in Phase 1**
  - 1 + Nr. Buckets <= Memory
- **Constraint in Phase 2**
  - 2 + Nr. Pages in Smaller Table/Nr. Buckets <= Memory
- **Rule of thumb**
  - Want memory > **Sqrt(Nr. Pages in Smaller Table)**

Example

Property	Value
Enrollment Pages	1,000
Student Pages	100
Available Buffer	11
Hash Join Cost	Sqrt(100)<11 Cost: 3*(100+1,000)

Details on Calculations

- Have enough buffer space to execute join as discussed
    - Rule of thumb: Sqrt(100) = 10 < 11
- Phase 1 reads and writes each input table page once
    - Cost is 2 * (100 + 1,000)
- Phase 2 reads and writes each input table page once
    - However, we do not count the output cost, as usual
    - Therefore, we only count cost 1 * (100 + 1,000)

What If We Lack Memory?

- Number of buffer pages limits number of **output buckets**
- Not enough buckets means **too much data** per bucket
- Prevents us from **loading one bucket** entirely in Phase 2
- Hence, perform **multiple passes** over data in phase 1
  - In each pass, buckets are partitioned into **sub-buckets**
  - Iterate until data per bucket **fits** into main memory

Sort-Merge Join: Idea

- Also specific to **equality** join conditions
- **Phase 1 (Sort)**
  - **Sort** joined tables on the join column
- **Phase 2 (Merge)**
  - Efficiently **merge** sorted tables together

Join Phase 1: Overview

- Lots of **sorting algorithms** proposed in the literature
- However, typically assume that we access **single** entries
- But random data access can be very **inefficient**
- Hence, want to access **pages** of entries instead
- Need specialized ("**external**") sort algorithms

Algorithm Sketch

- **Step 1**: load chunk of data and **sort**, write back to disk
- **Step 2 .. n**: **merge** sorted runs to produce larger runs
- Each merging step **reduces** number of runs (but longer)
- Finally, have only **one sorted run** left - we're done!

Details on Step 1

- Assume we have **B buffer pages** available
- **Load chunks** of B pages into the buffer
- For each chunk, **sort** by standard sort algorithm
  - Can use standard algorithm as **all data in memory**
- Then, **write** sorted data to hard disk
- A sorted sequence of data is called a "**run**"

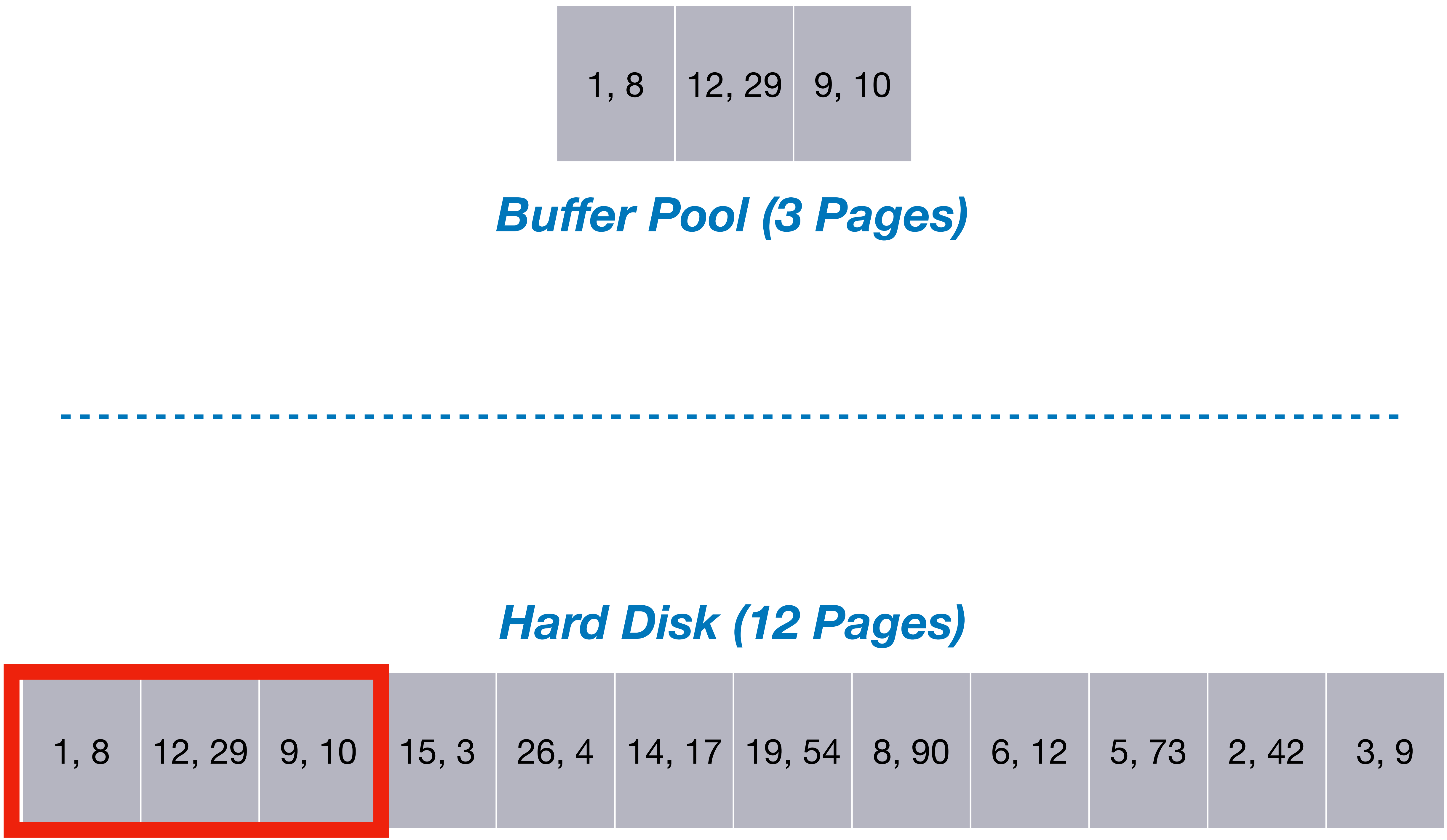
Step 1 Example

Buffer Pool (3 Pages)

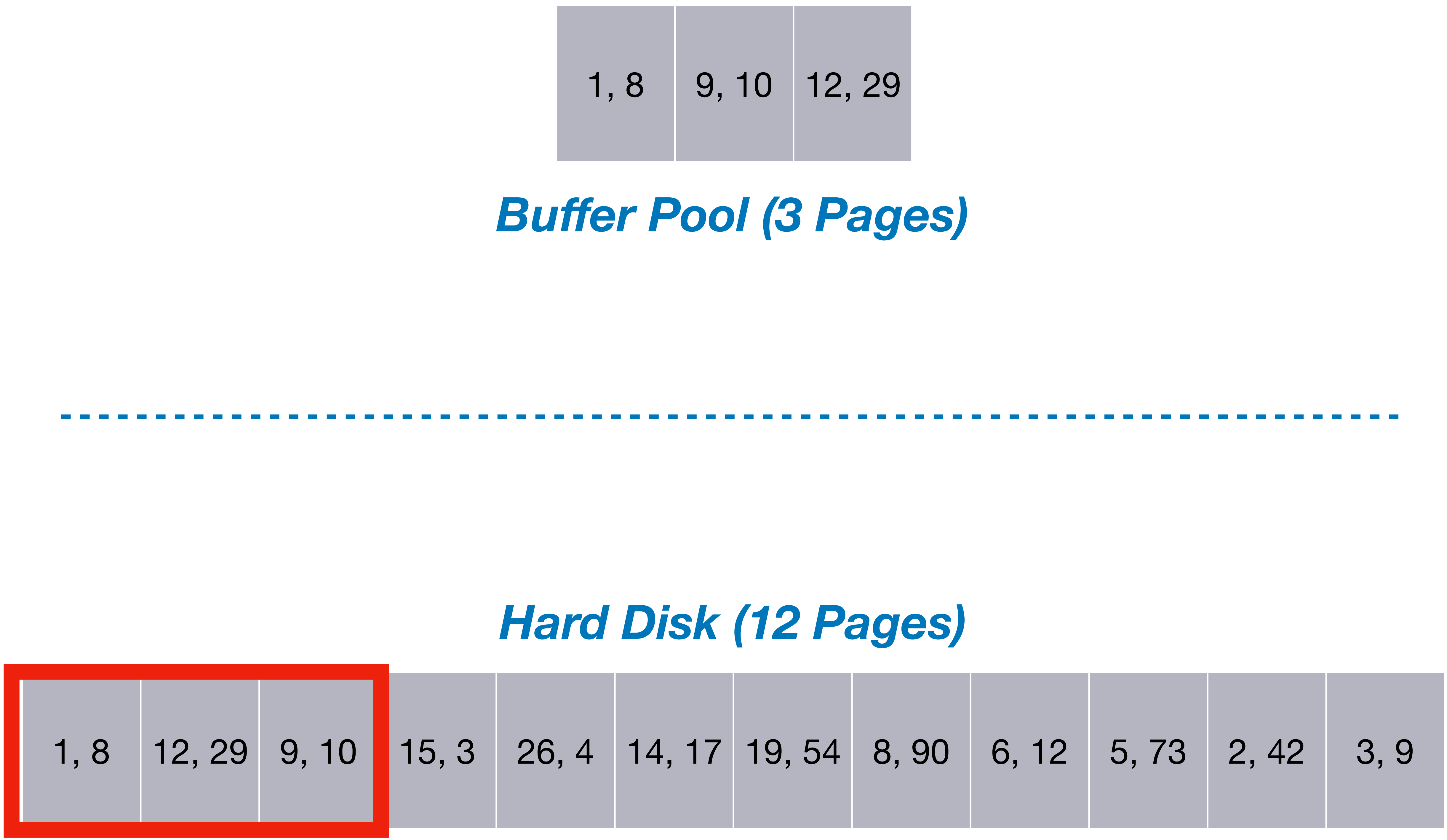
Hard Disk (12 Pages)

1, 8	12, 29	9, 10	15, 3	26, 4	14, 17	19, 54	8, 90	6, 12	5, 73	2, 42	3, 9

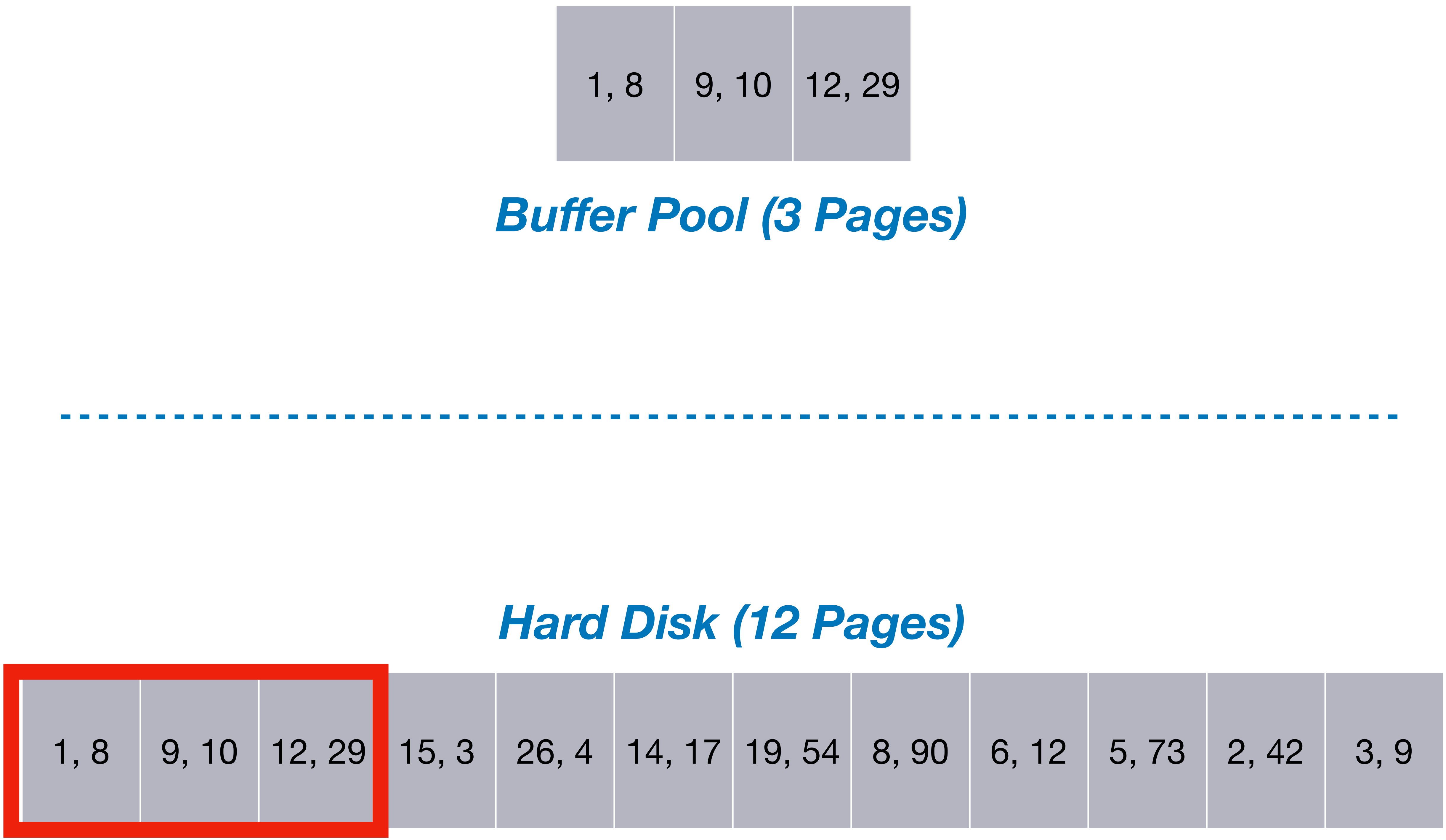
Step 1 Example



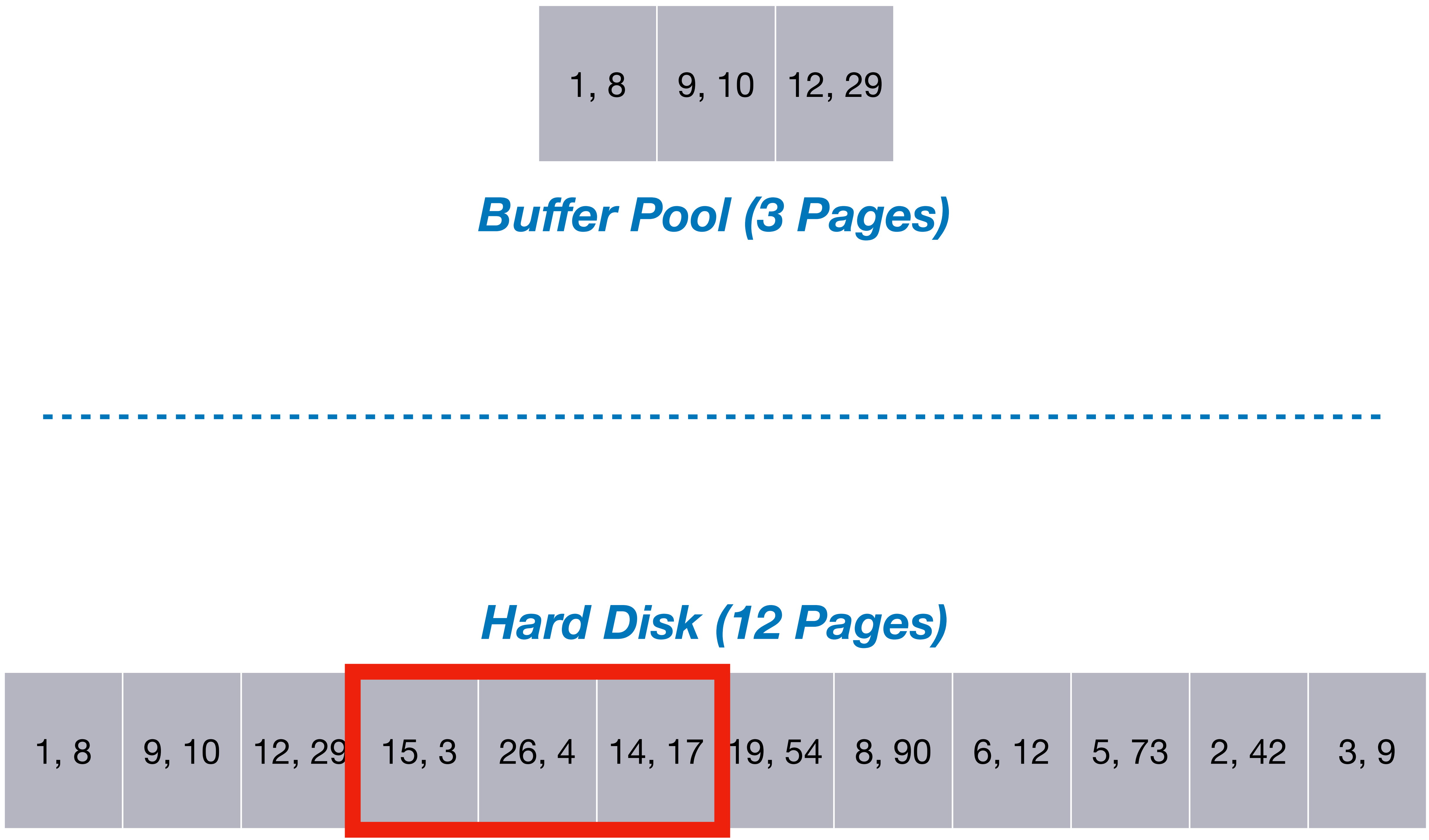
Step 1 Example



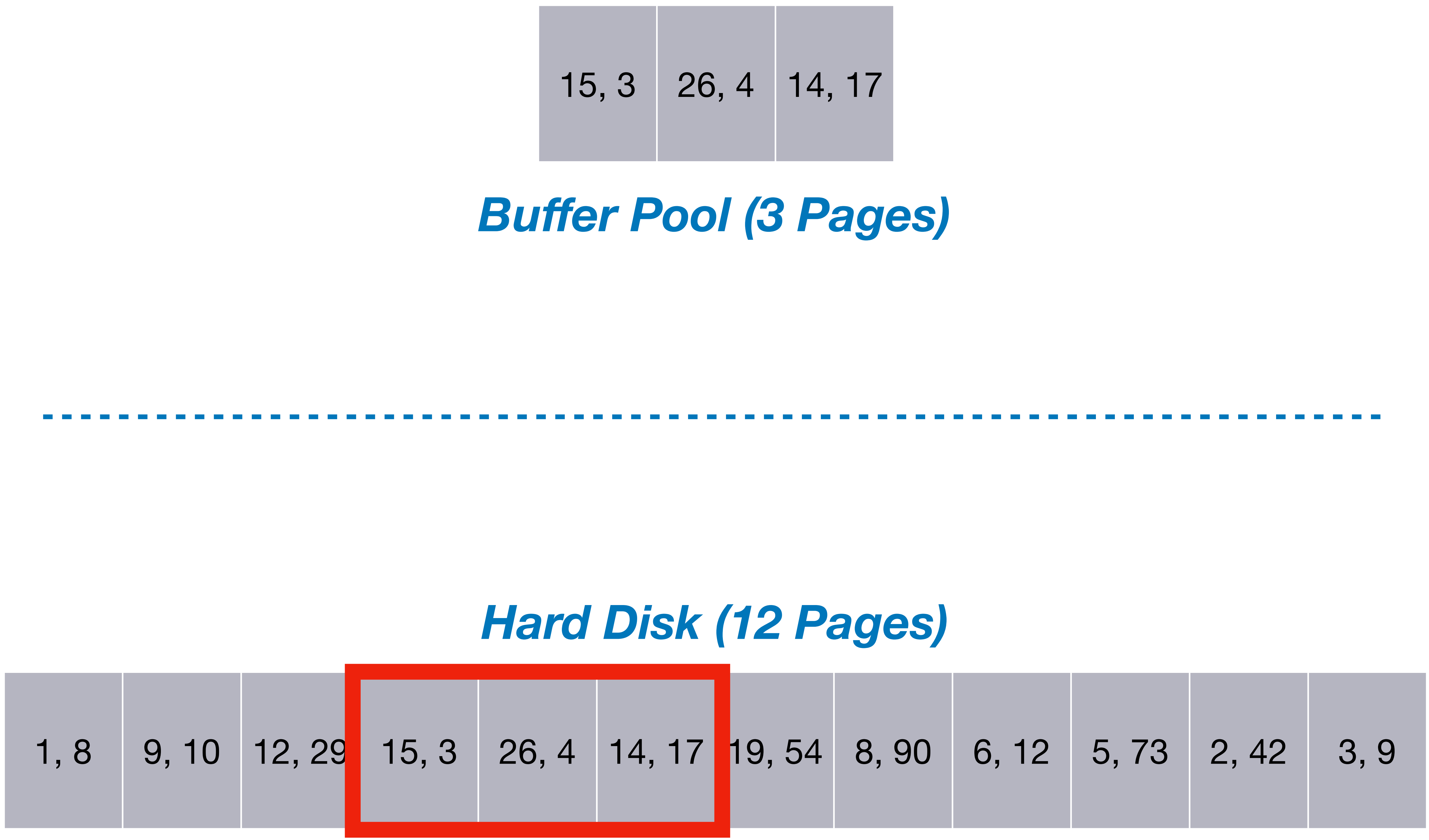
Step 1 Example



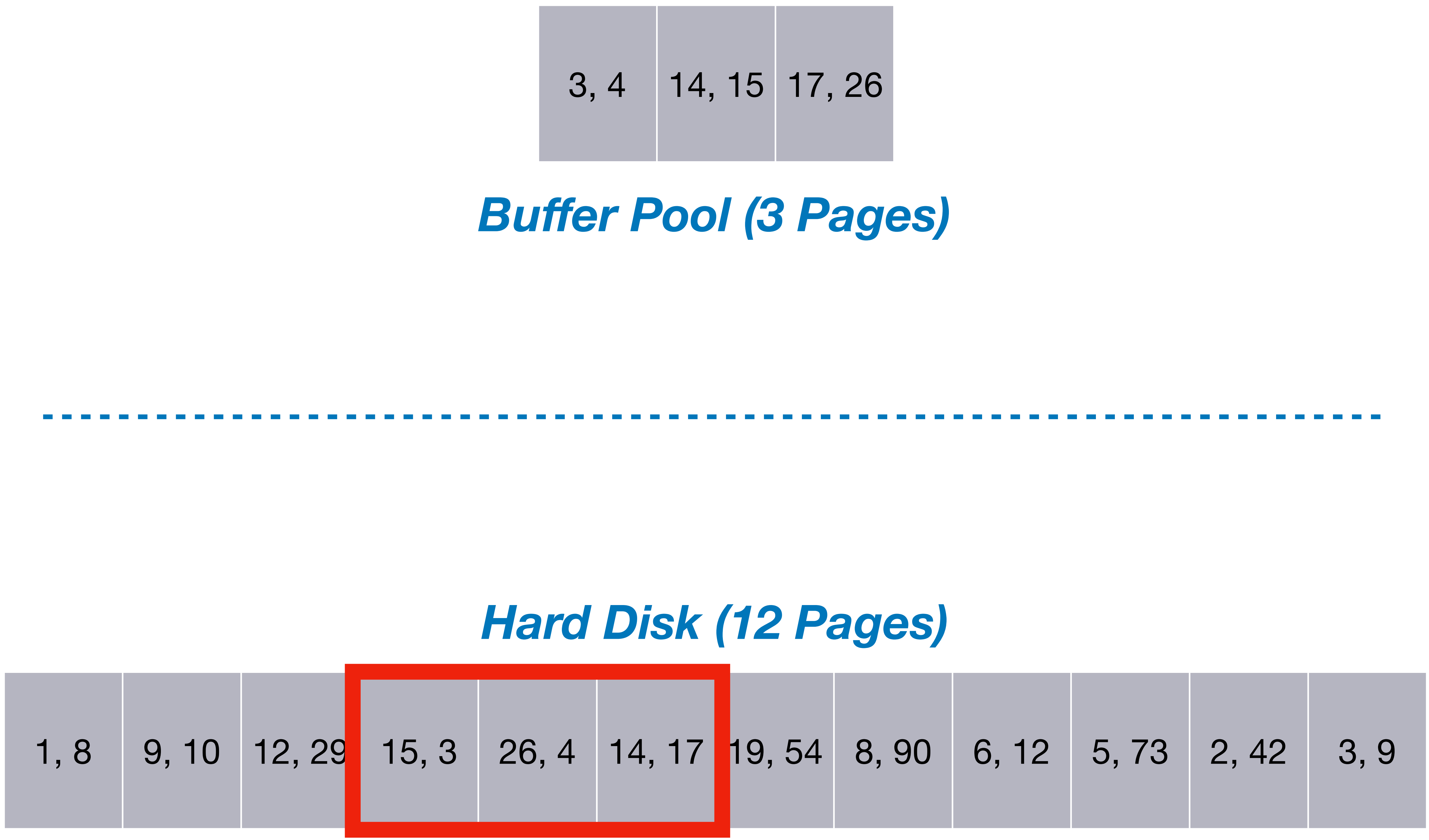
Step 1 Example



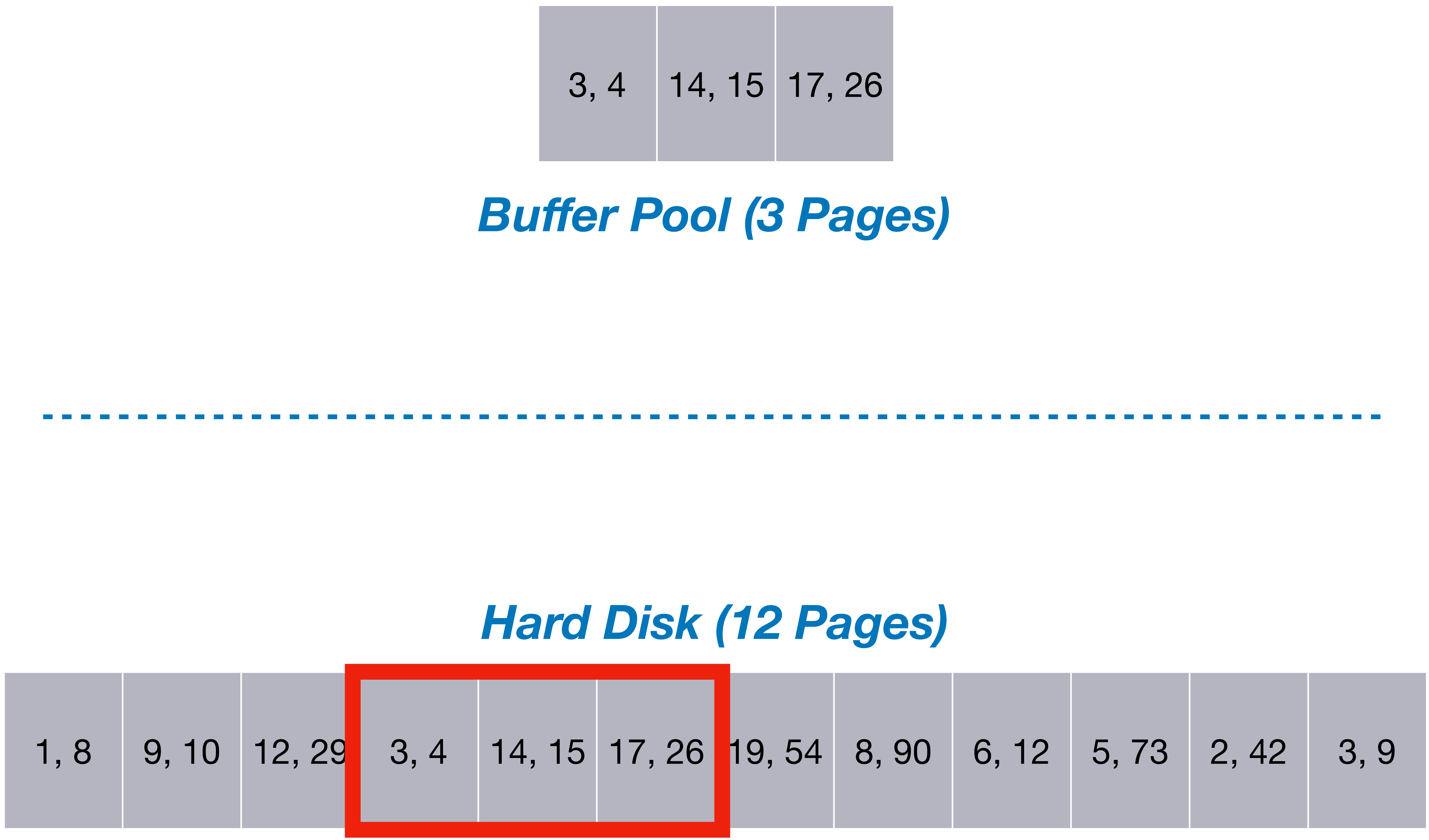
Step 1 Example



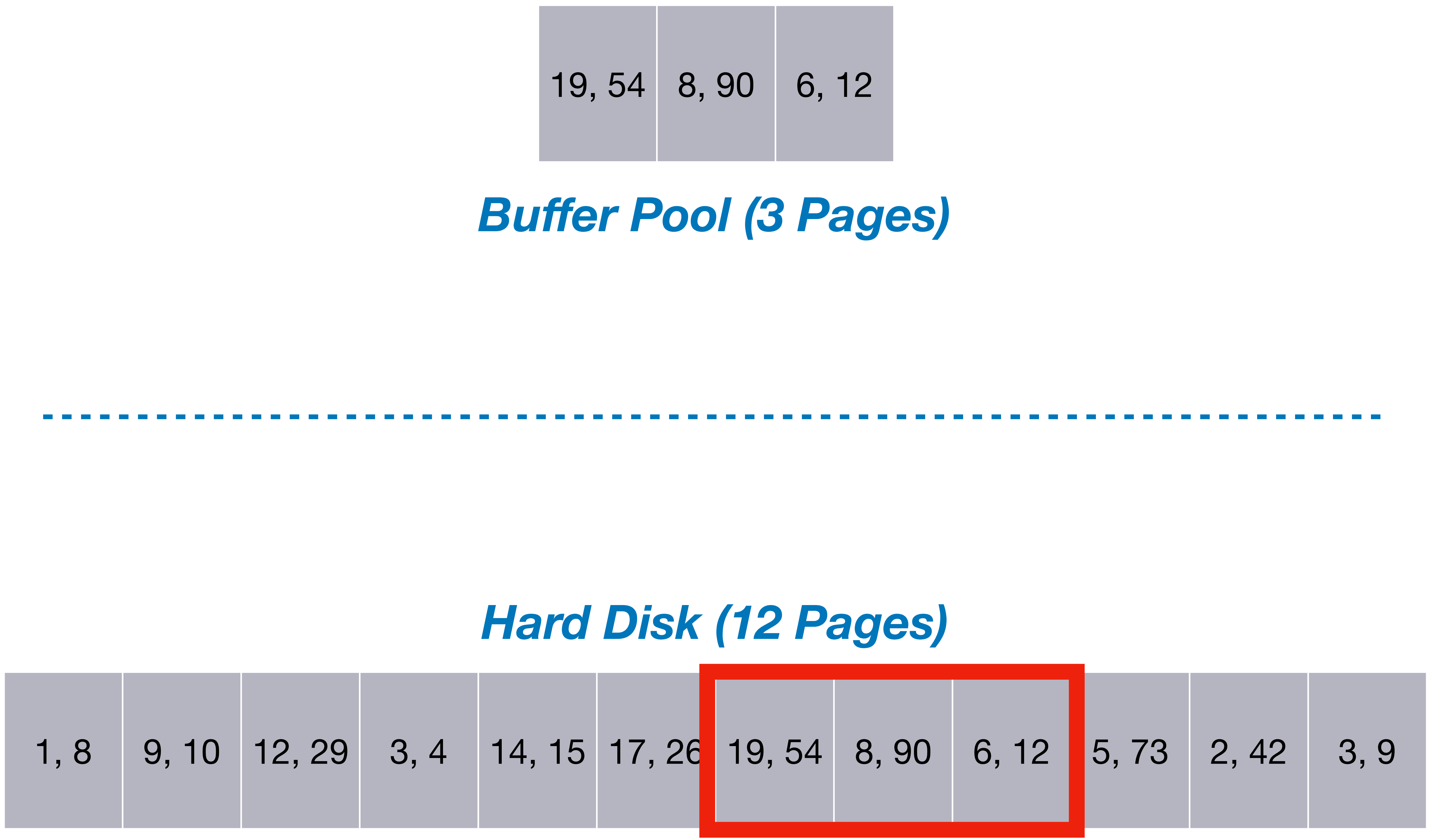
Step 1 Example



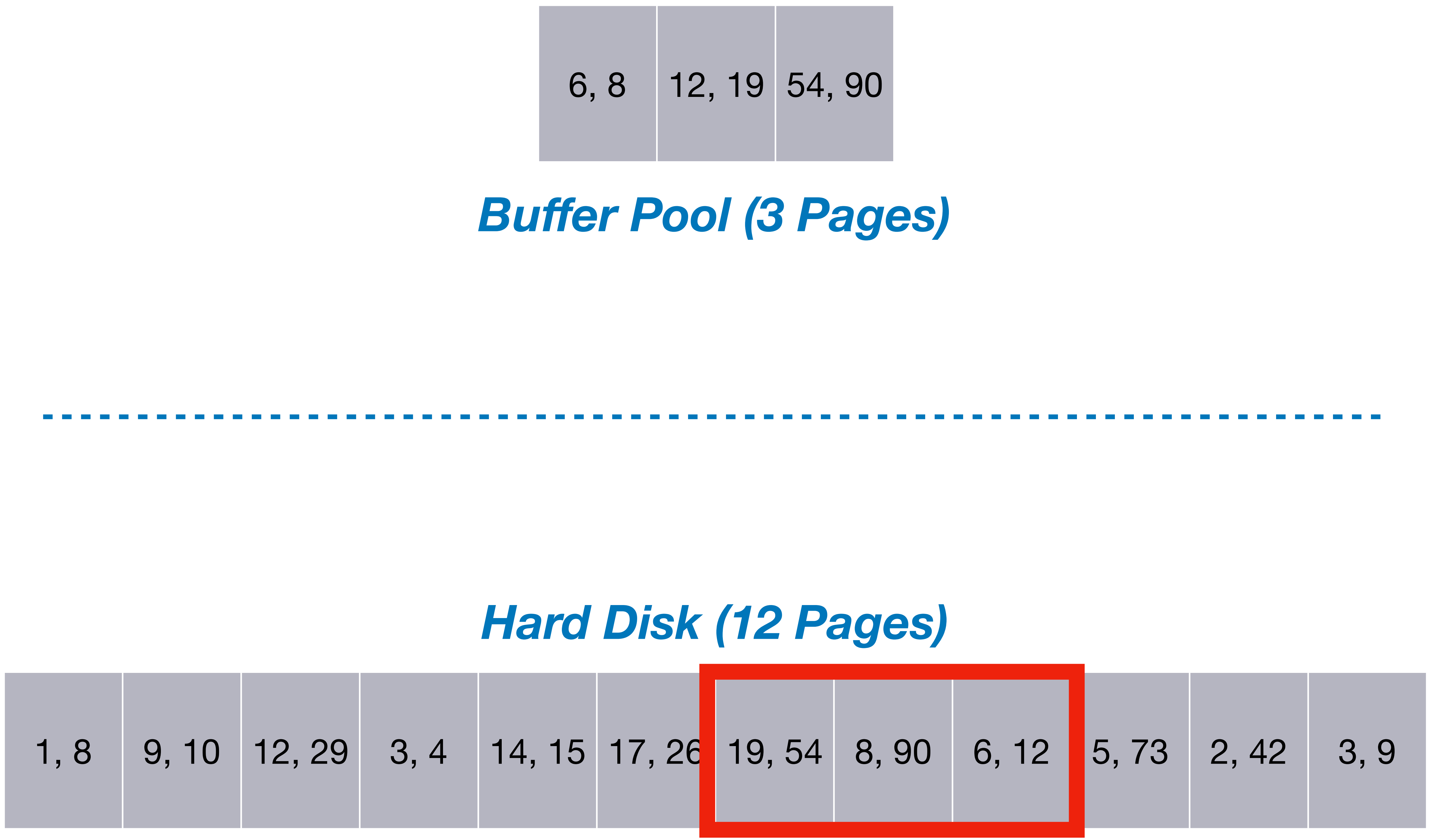
Step 1 Example



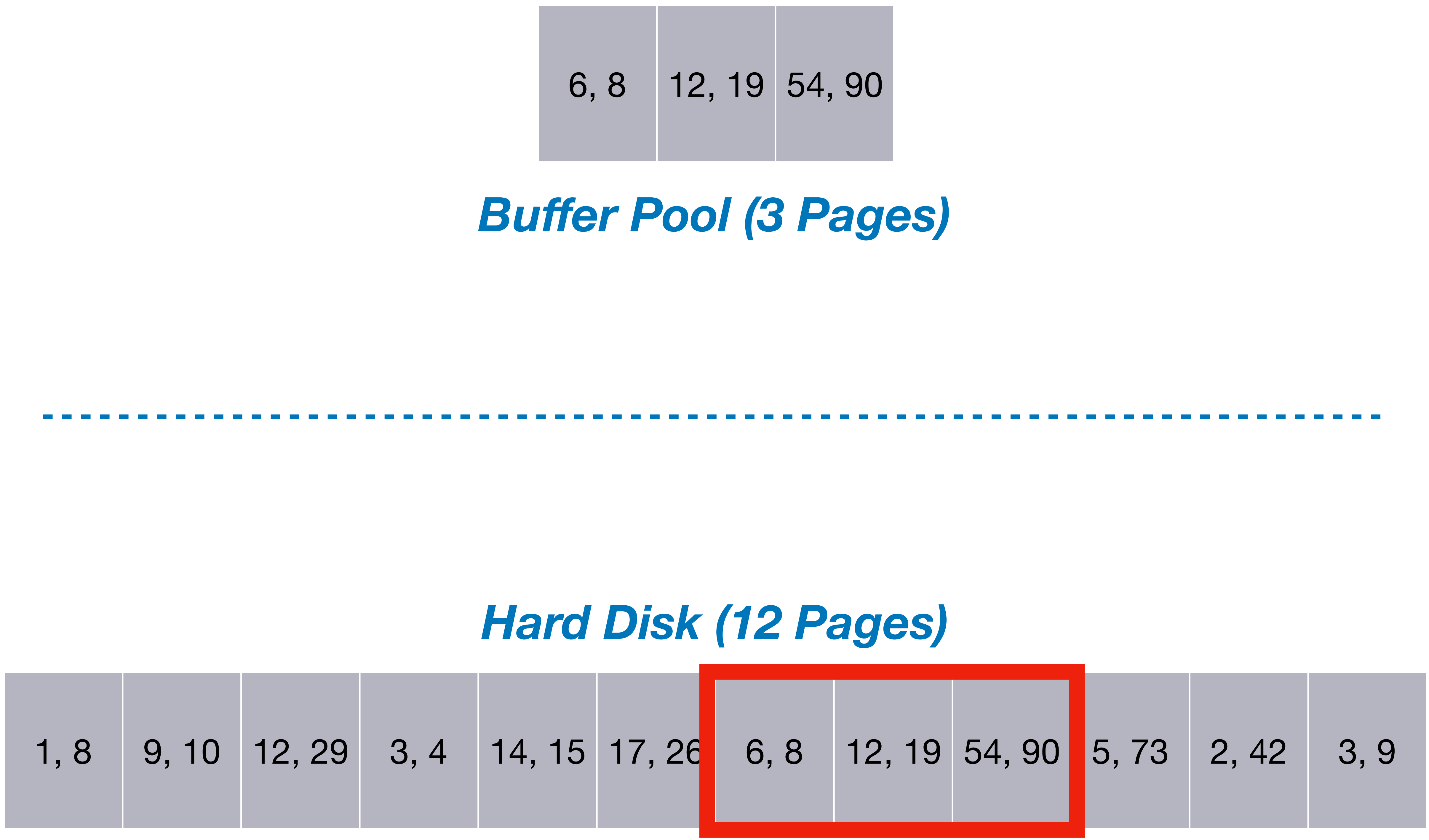
Step 1 Example



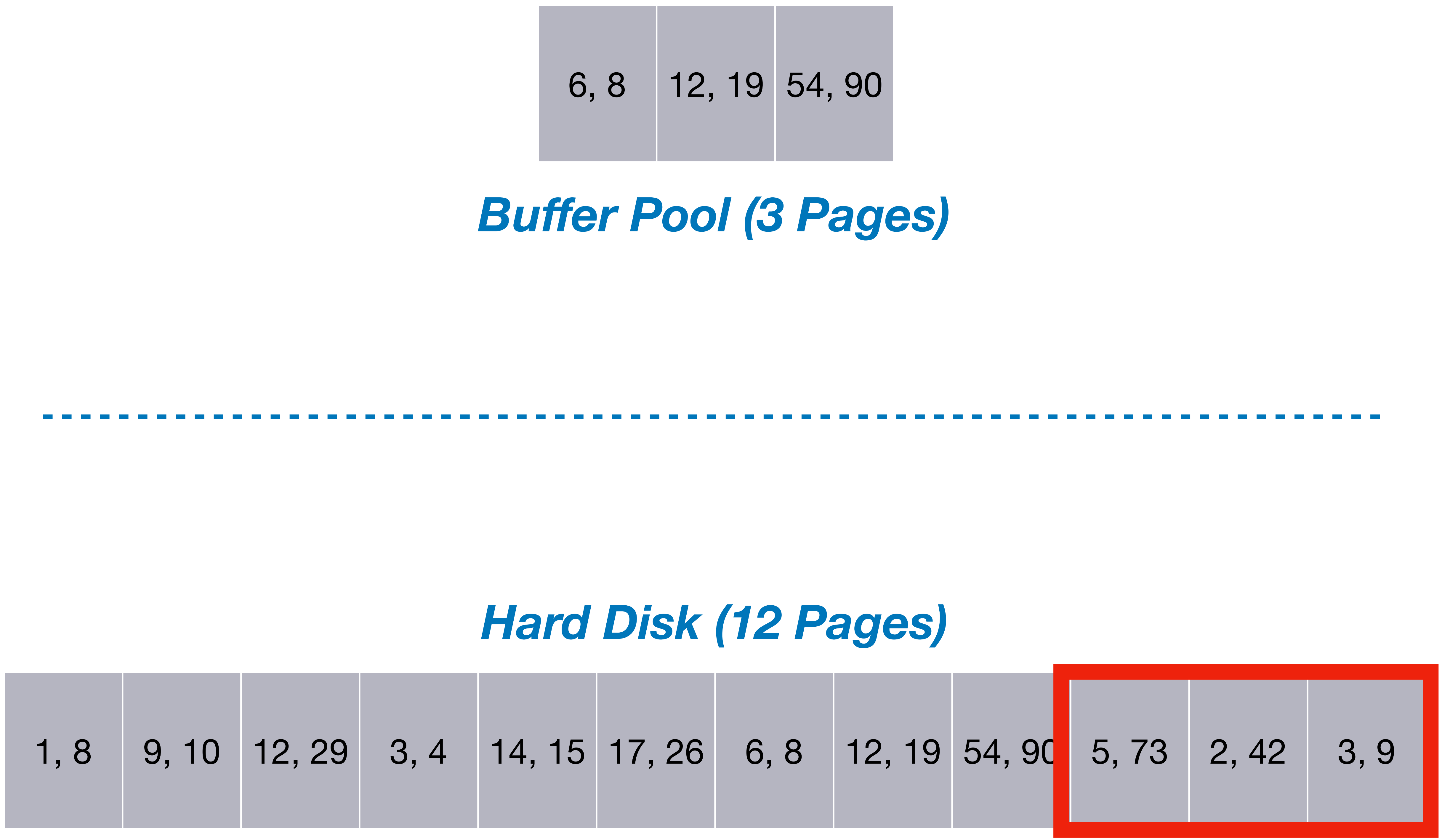
Step 1 Example



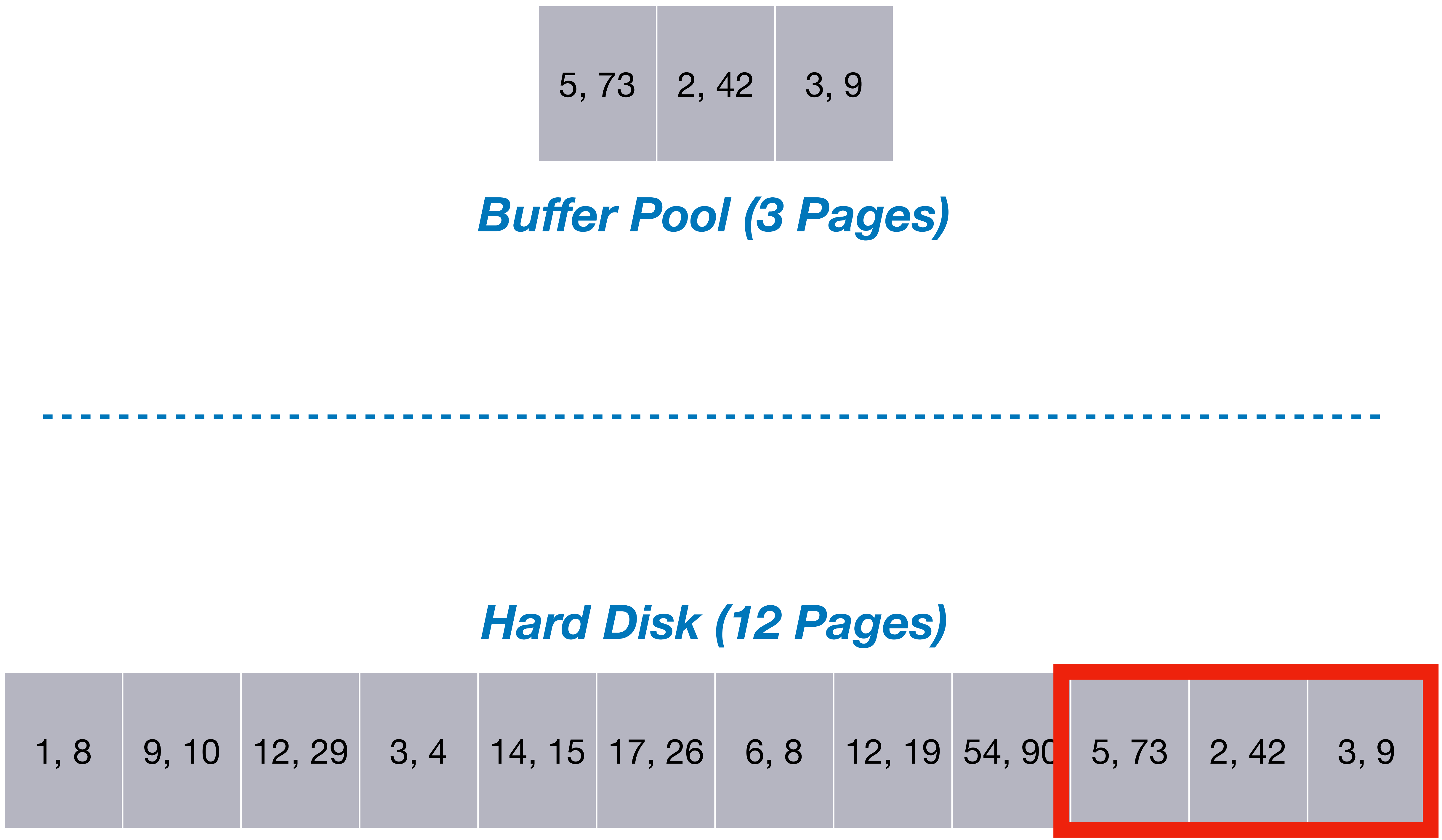
Step 1 Example



Step 1 Example



Step 1 Example



Step 1 Example

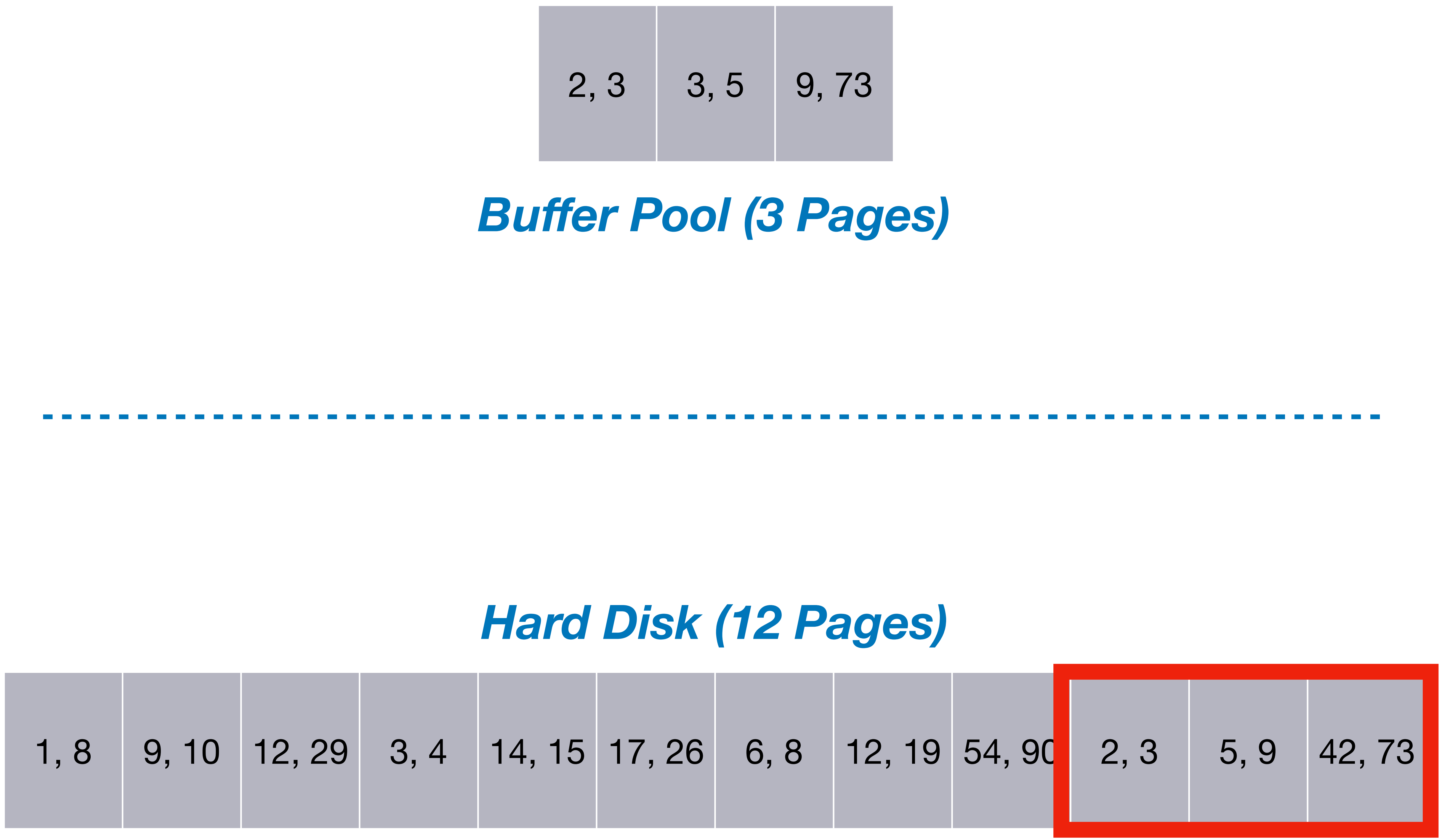
2, 3	5, 9	42, 73

Buffer Pool (3 Pages)

Hard Disk (12 Pages)

1, 8	9, 10	12, 29	3, 4	14, 15	17, 26	6, 8	12, 19	54, 90	5, 73	2, 42	3, 9

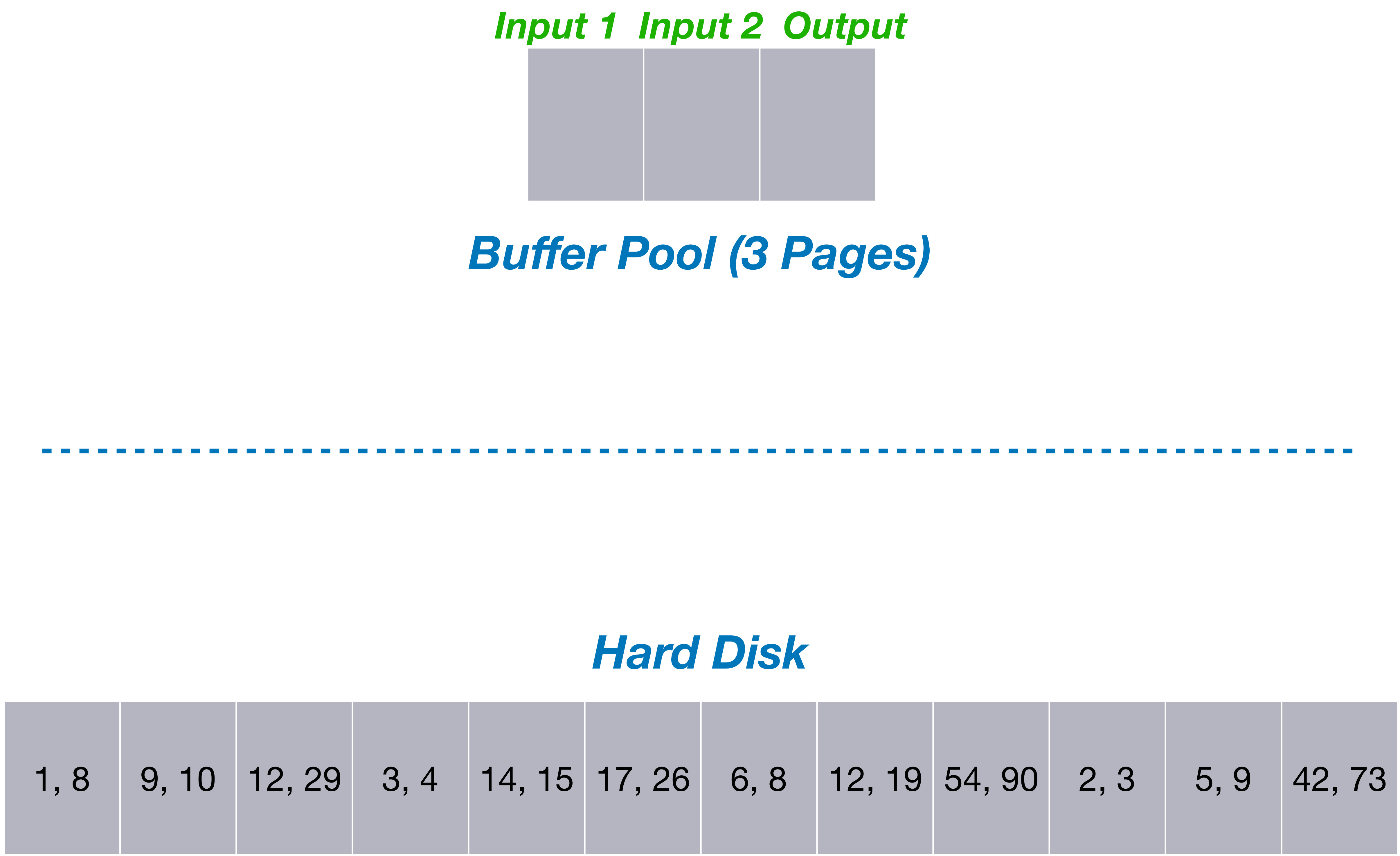
Step 1 Example



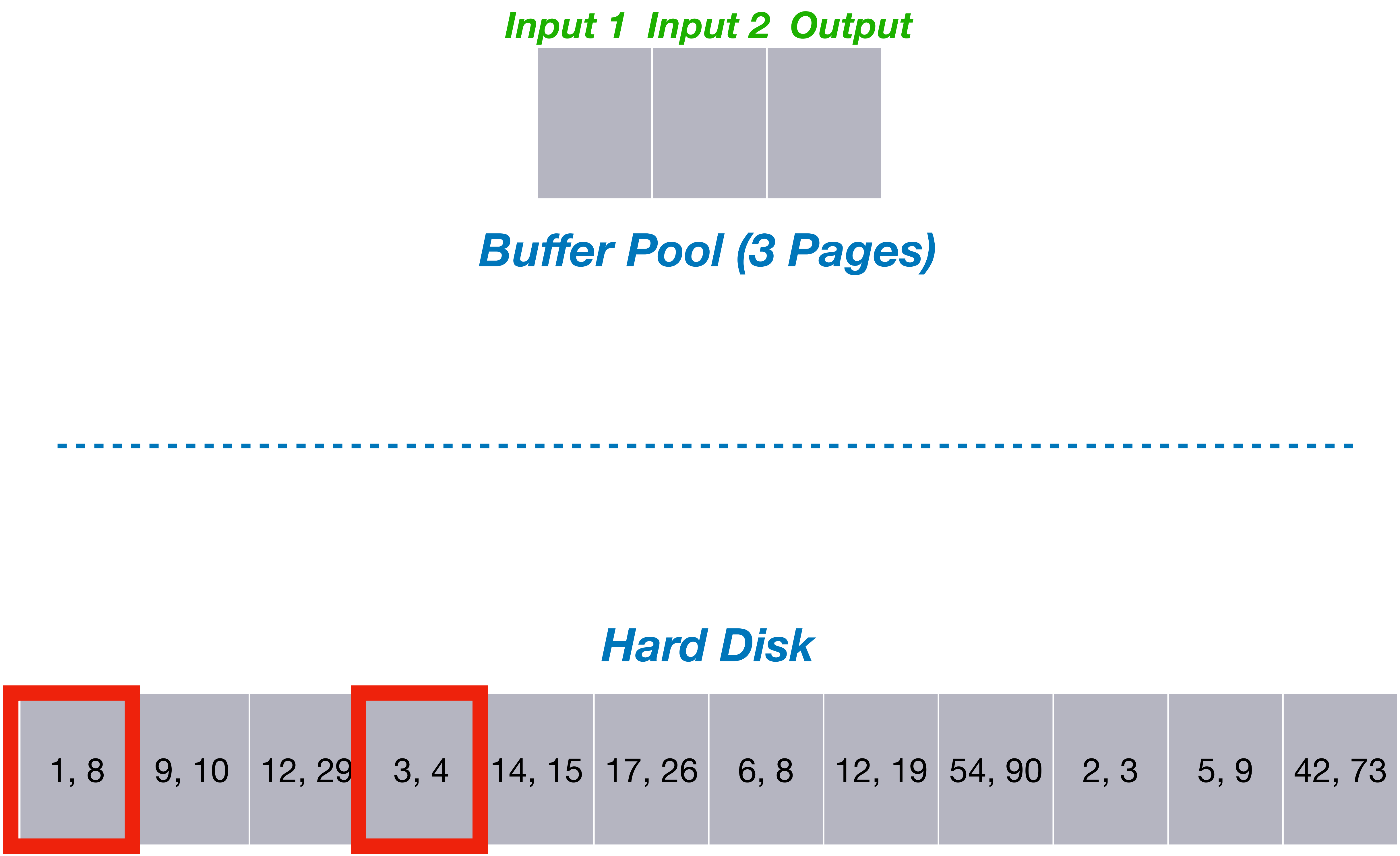
Details on Steps 2 .. n

- (Still have B buffer pages available)
- Enables us to **merge B-1 sorted runs** into one in one step
  - **Load first page** of each sorted run into B-1 pages
  - **Copy minimum entry** in input buffers to output buffer
    - If output buffer full, **write to disk and clear**
  - **Erase minimum** entry from input buffer
    - If input buffer becomes empty, **load next page**

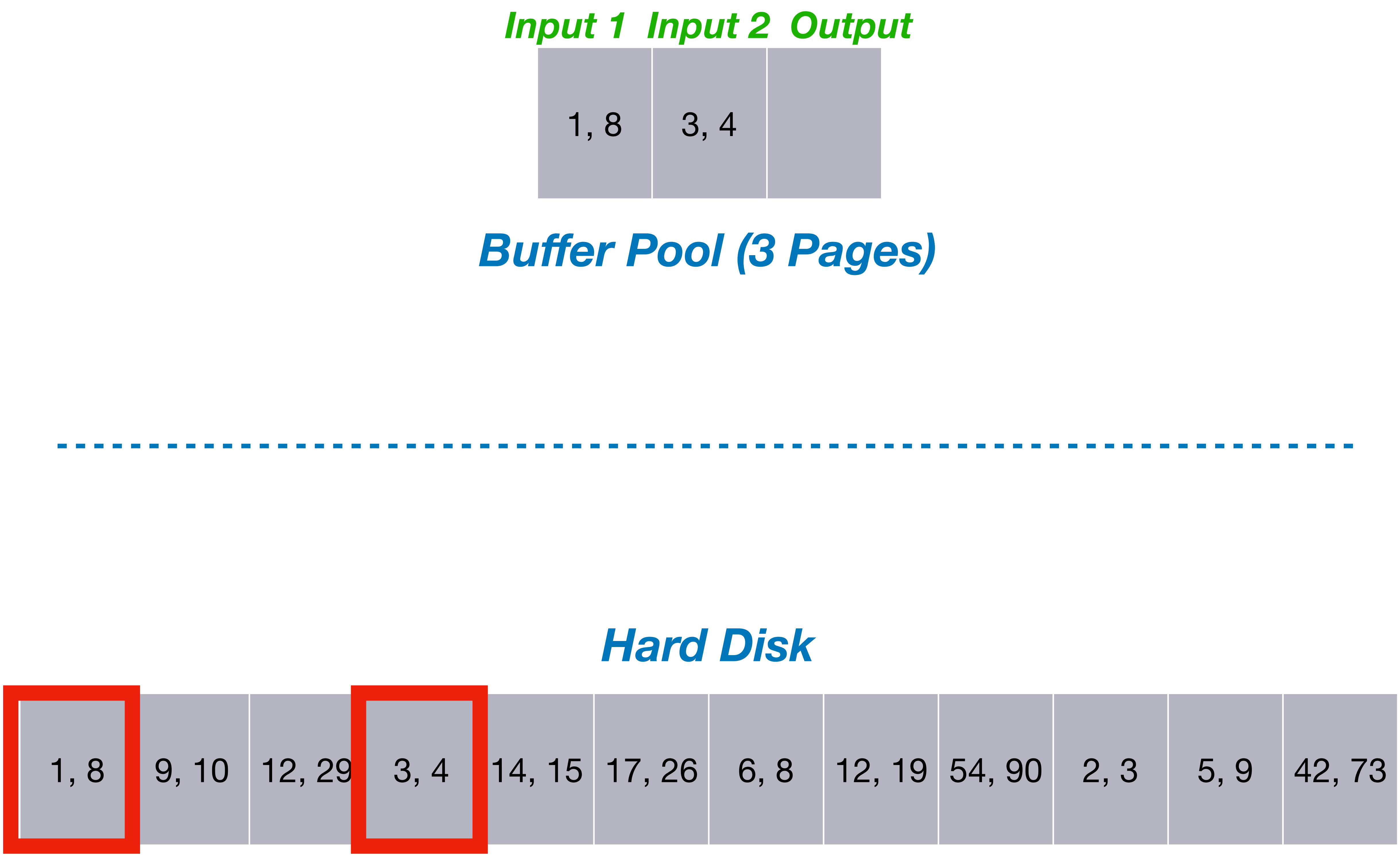
Step 2 Example



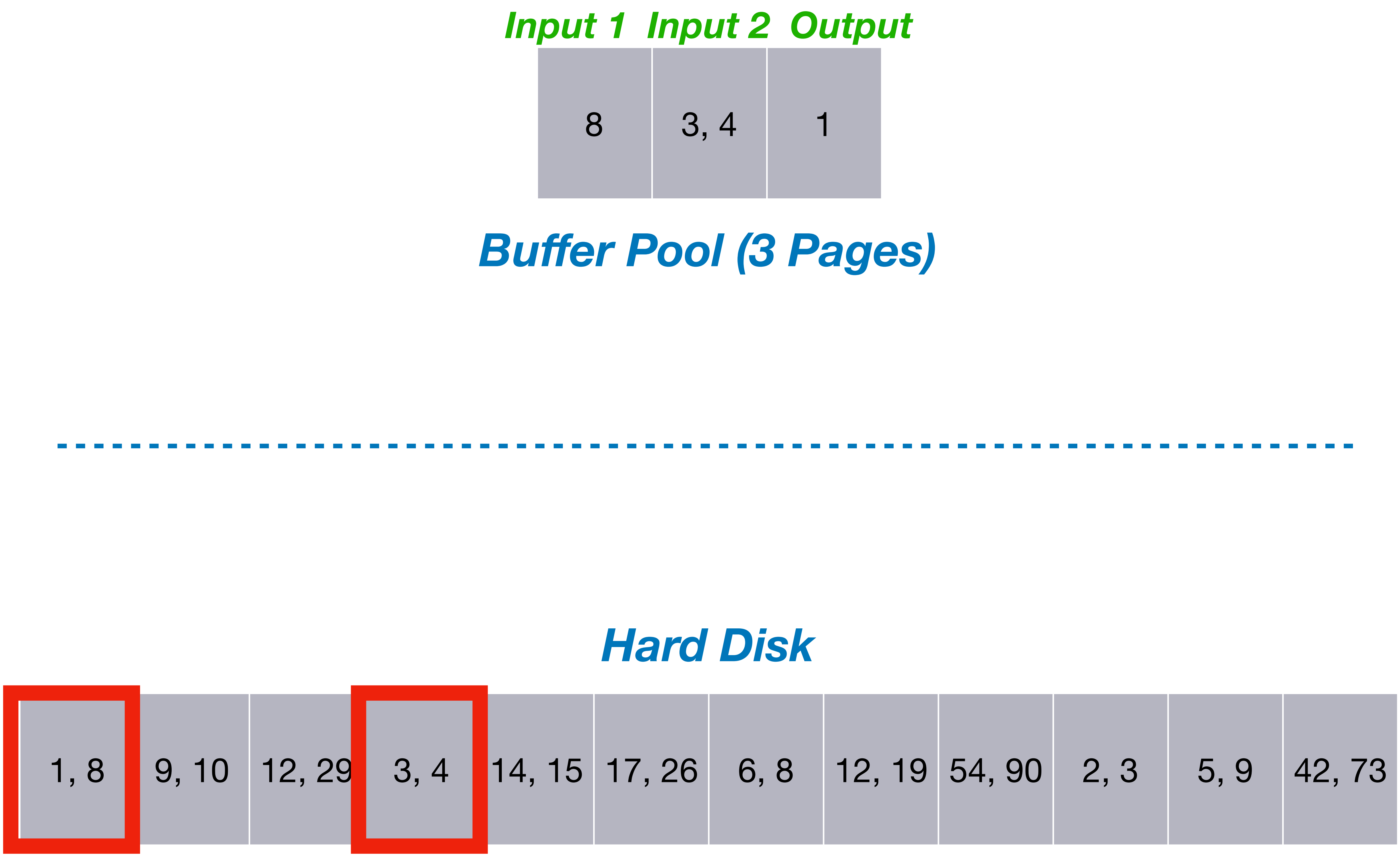
Step 2 Example



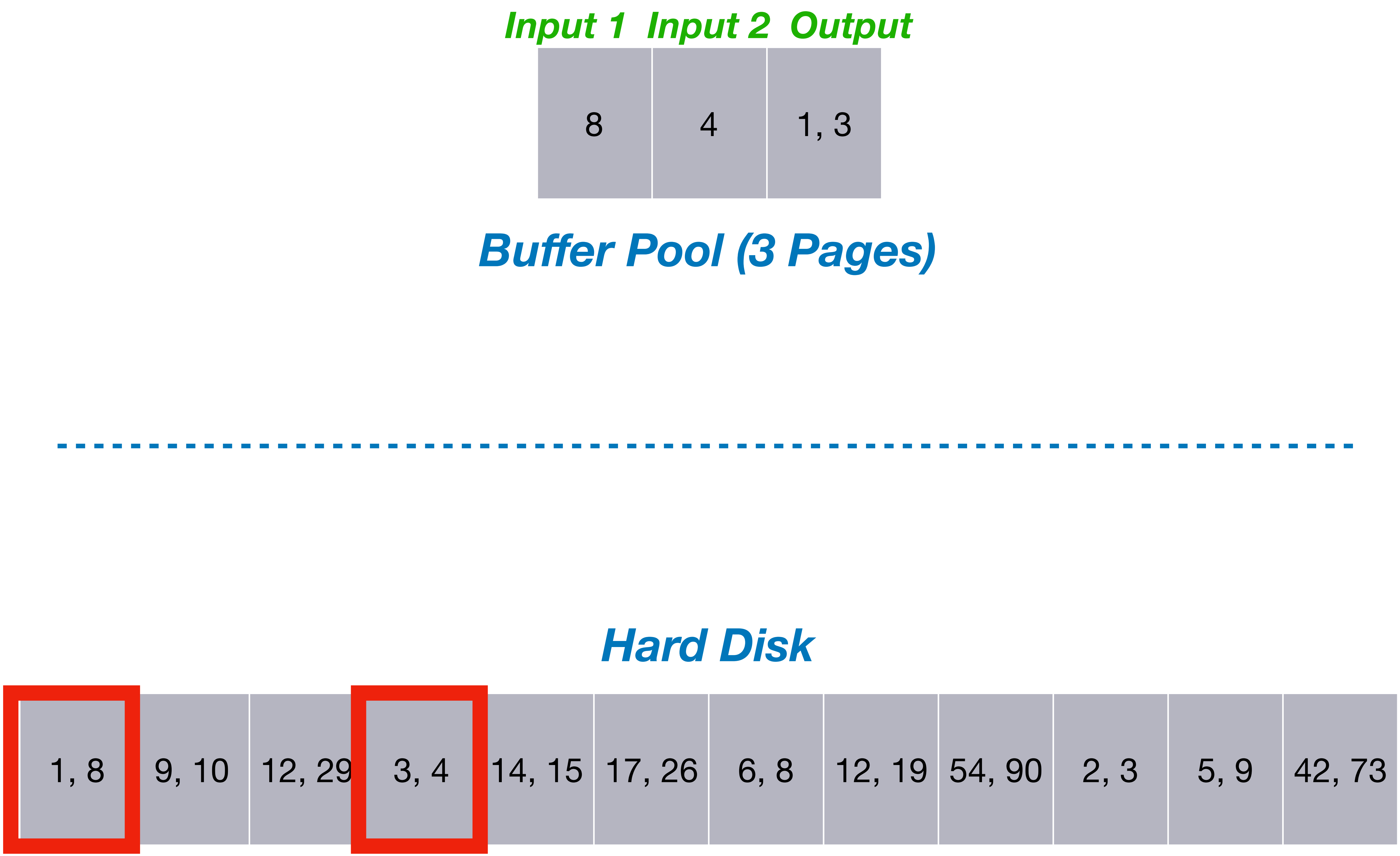
Step 2 Example



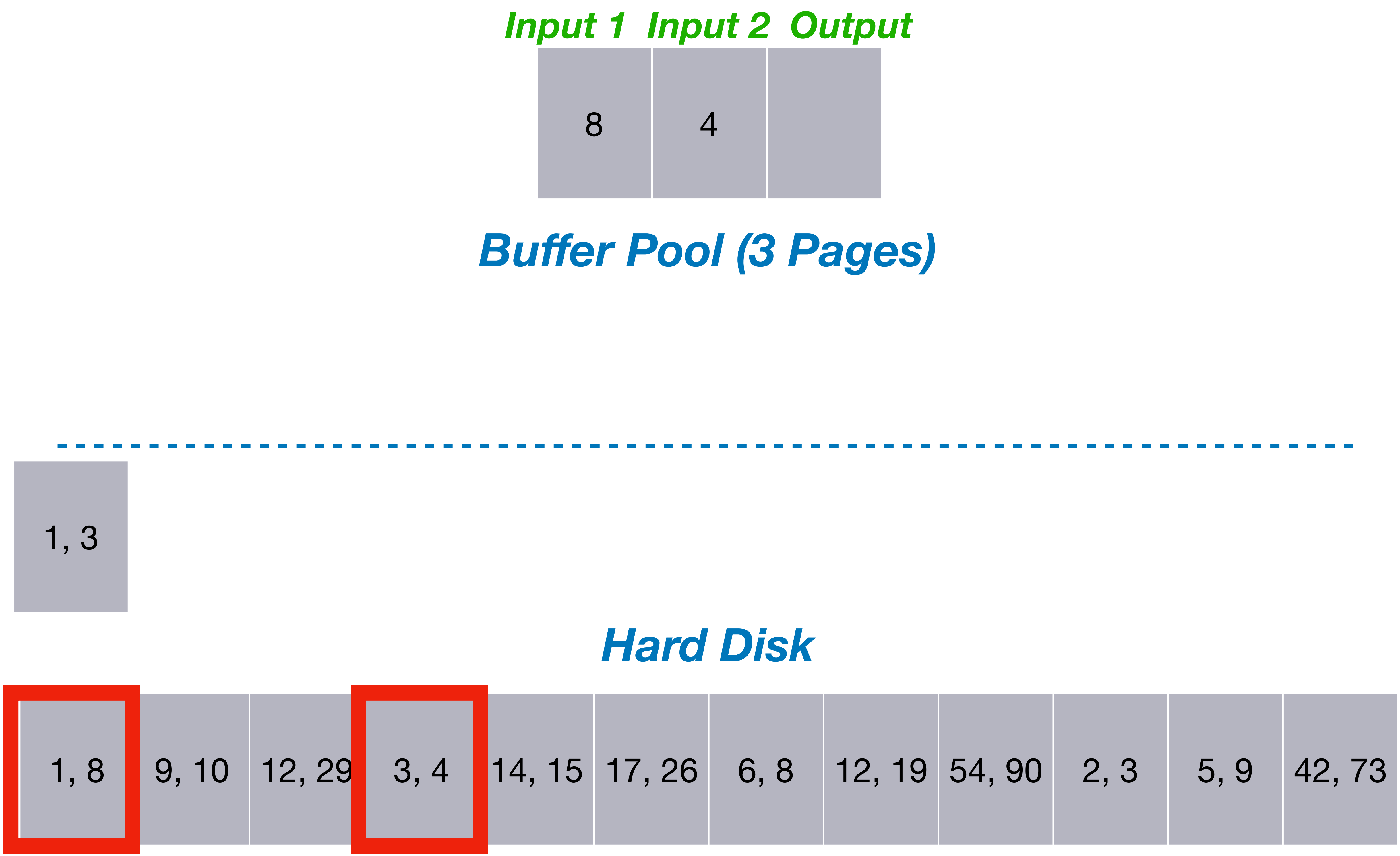
Step 2 Example



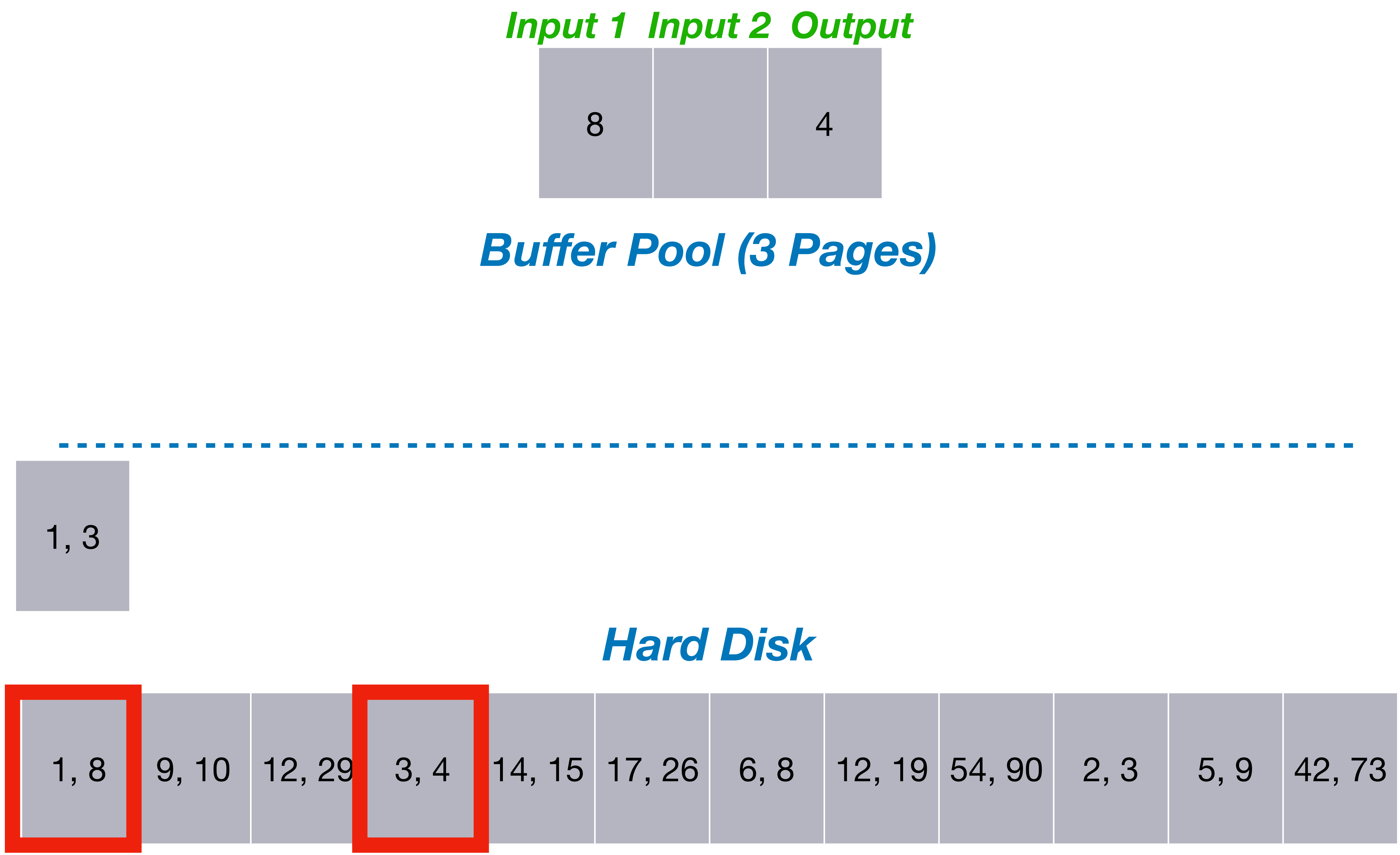
Step 2 Example



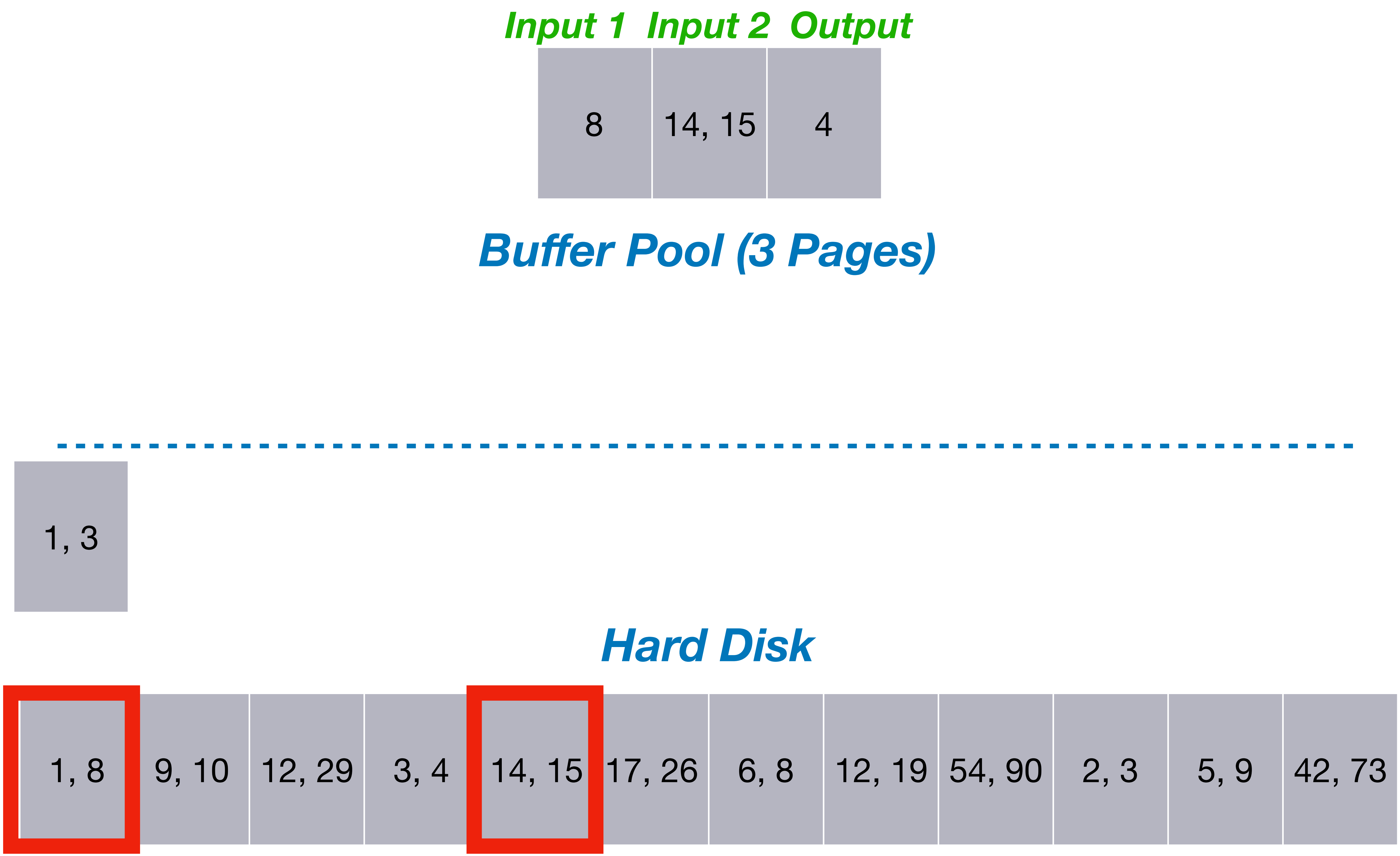
Step 2 Example



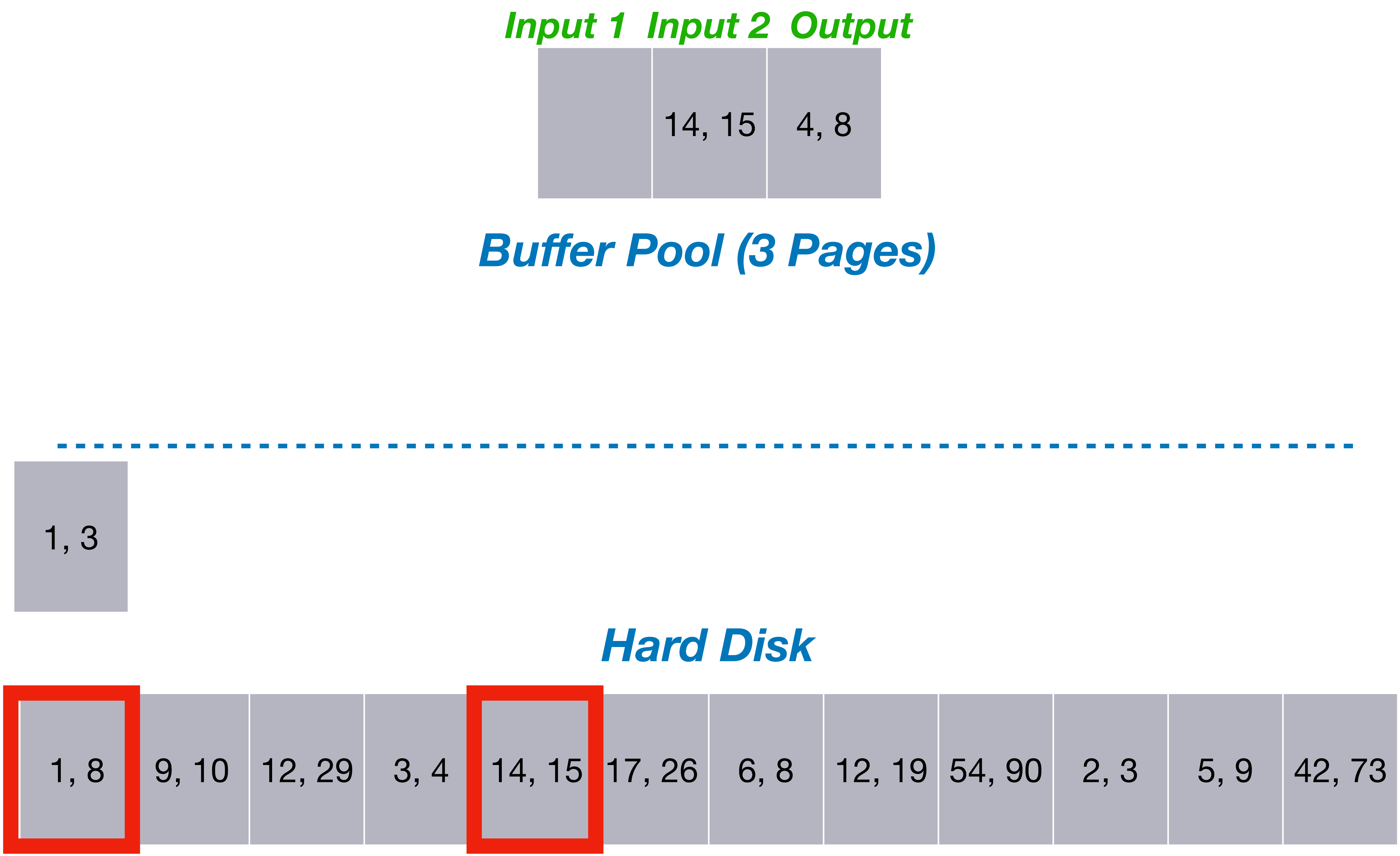
Step 2 Example



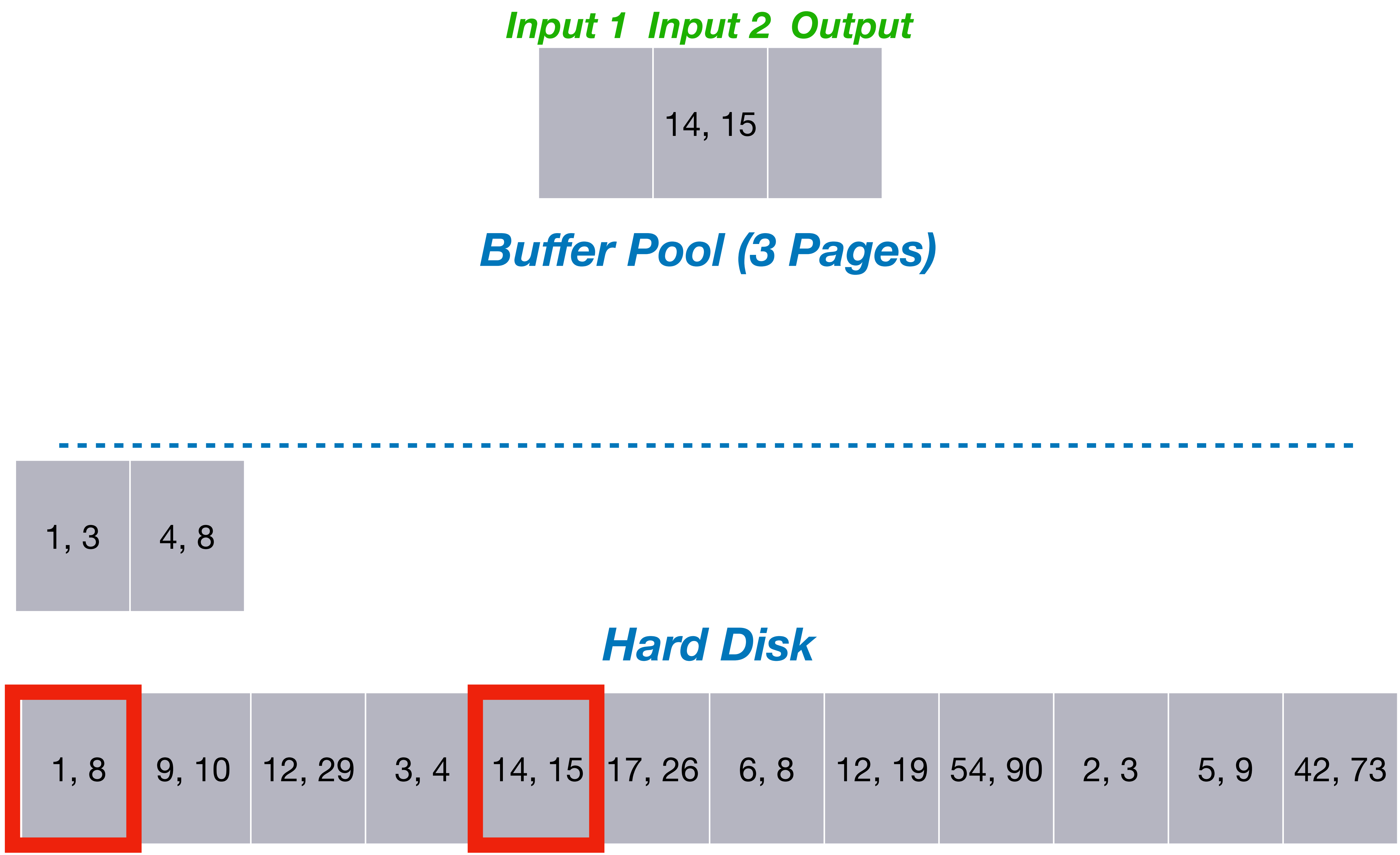
Step 2 Example



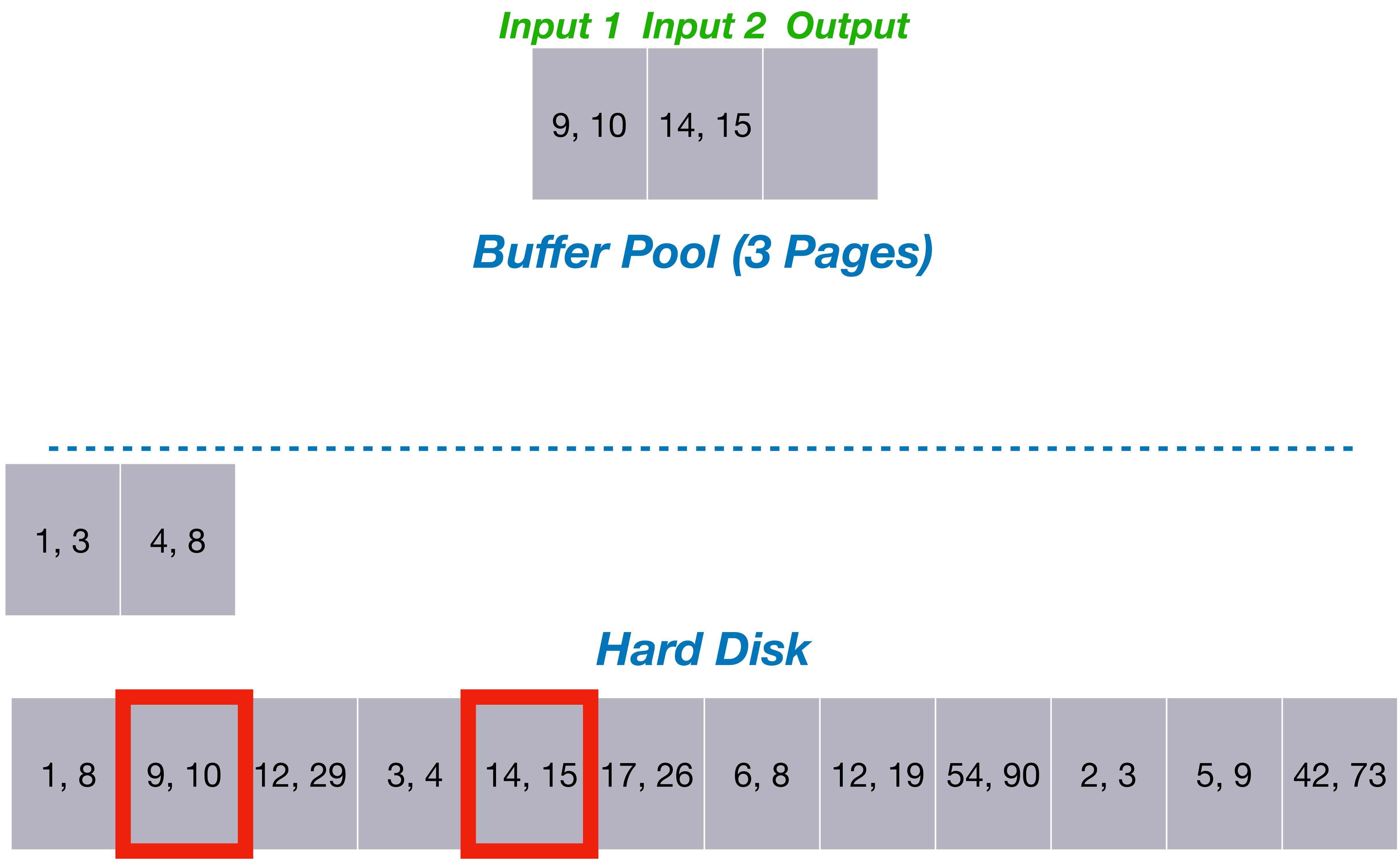
Step 2 Example



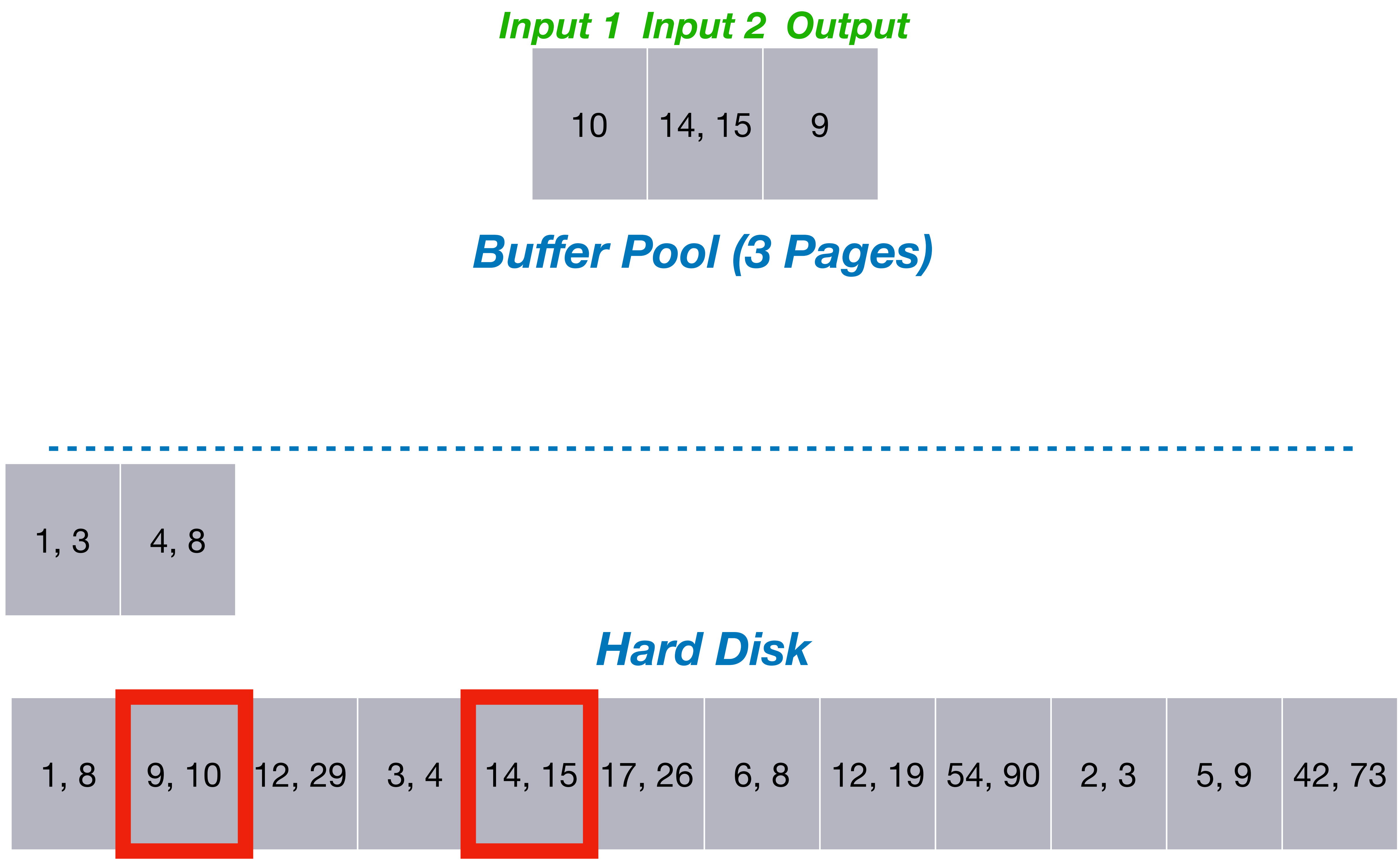
Step 2 Example



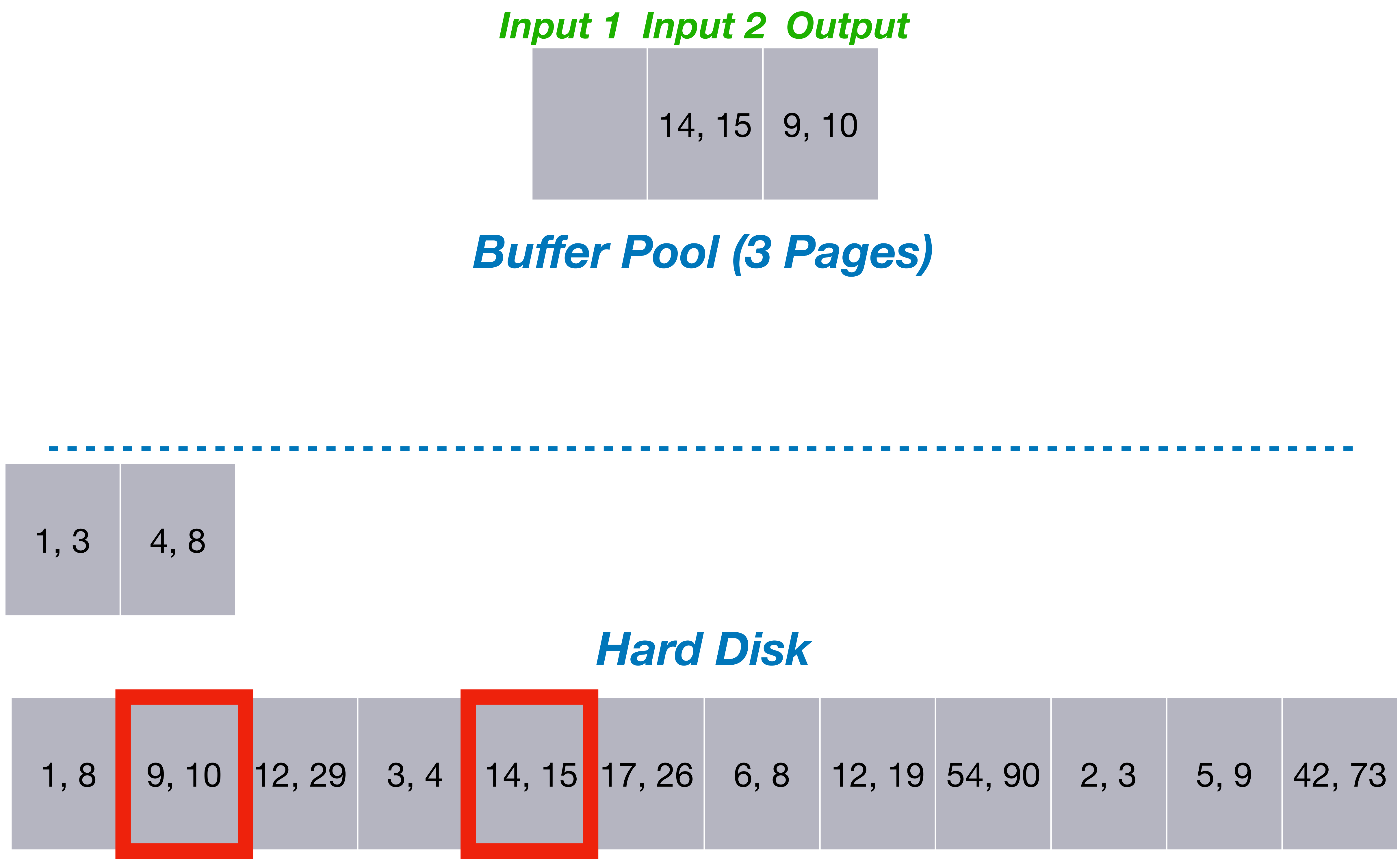
Step 2 Example



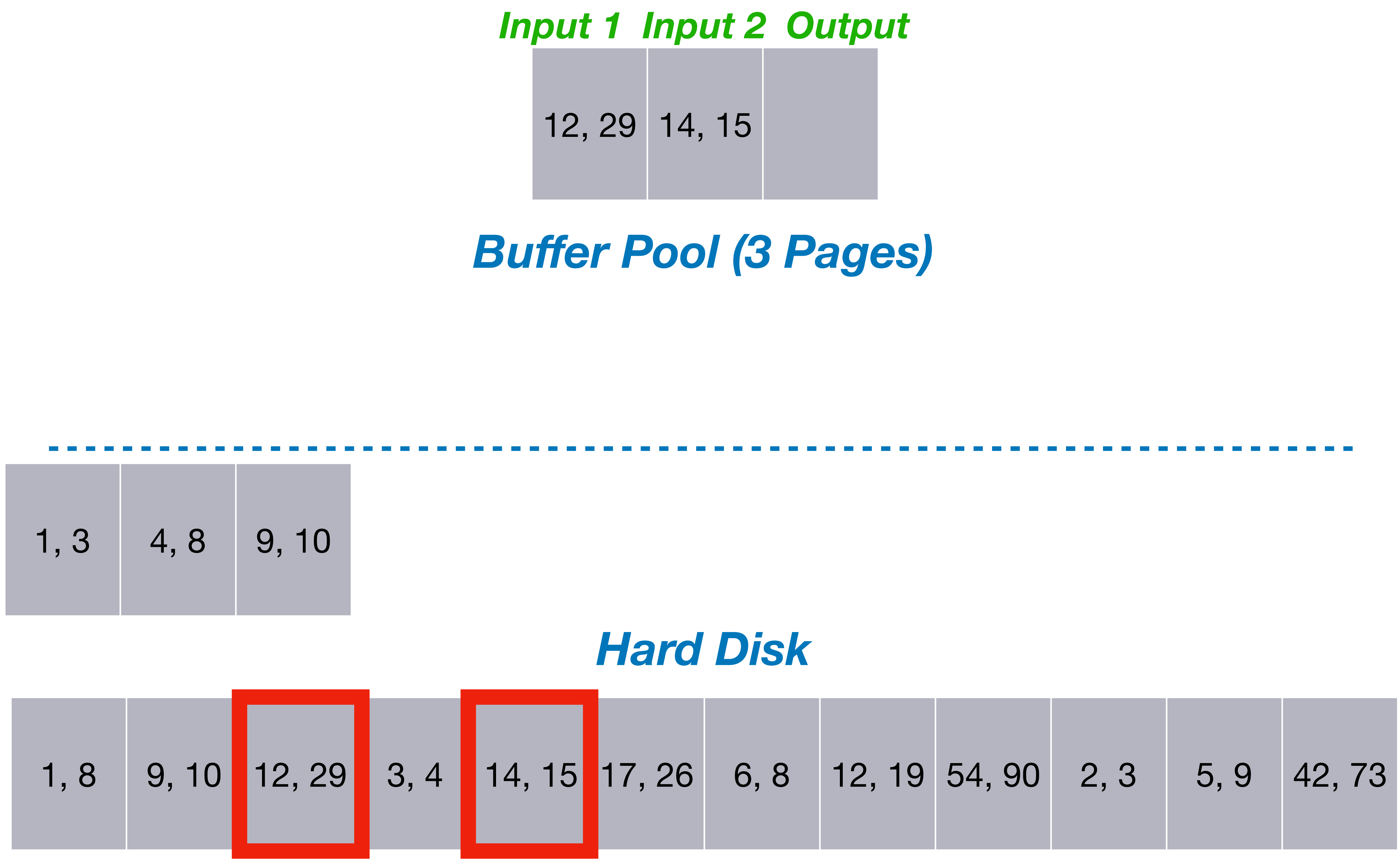
Step 2 Example



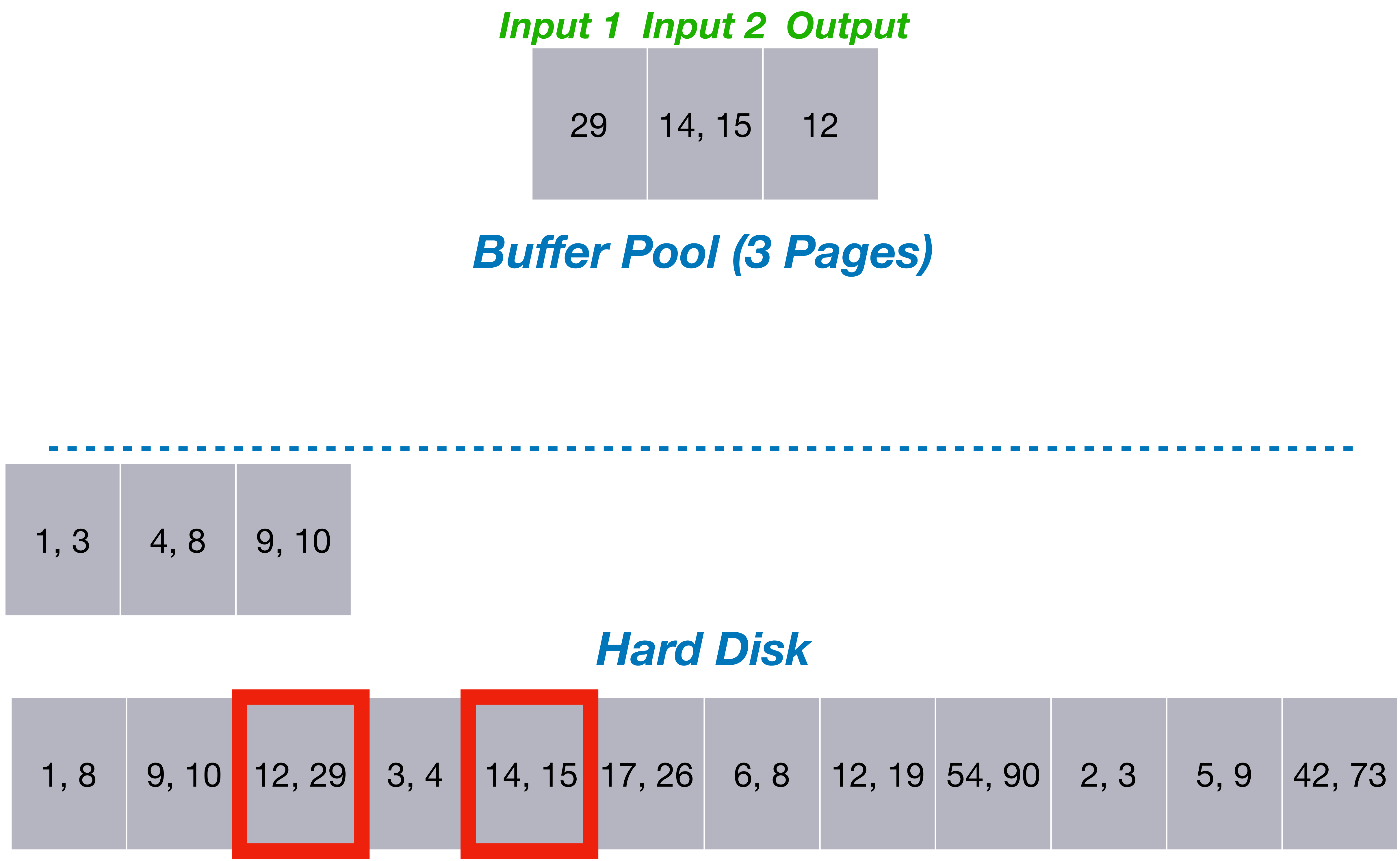
Step 2 Example



Step 2 Example



Step 2 Example



■ ■ ■

Example Summary

- Have **12 pages** to sort with **3 buffer** pages
- First step: produce **4 sorted runs** of **length 3**
- Can **merge 2 runs** in each merge step
- Second step: produce **2 sorted runs** of **length 6**
- Third step: produce **1 sorted run** of **length 12**

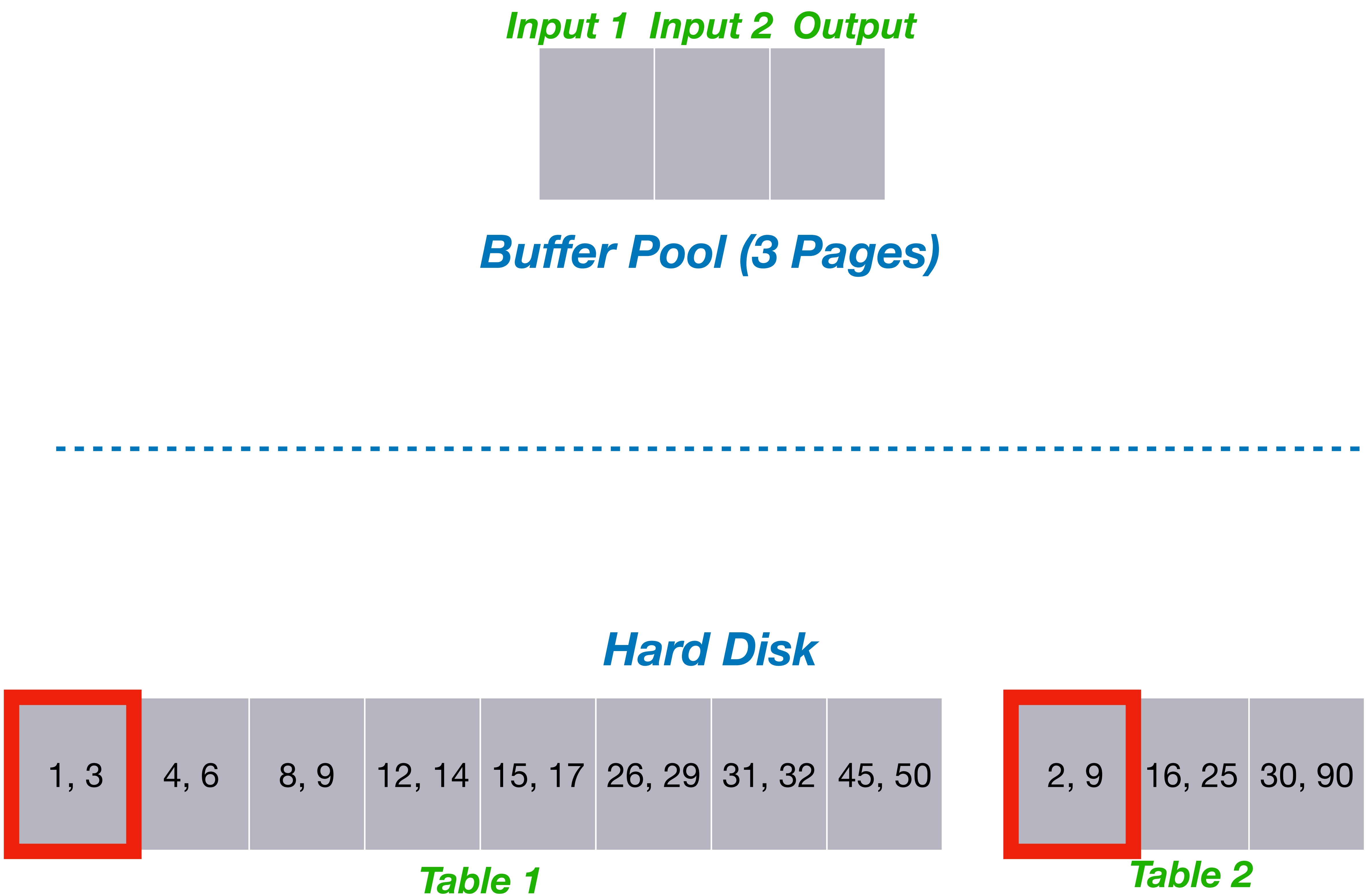
Cost Analysis (Phase 1)

- Multiple **sorting passes**, we read and write data once in each
  - Cost per pass is **2 * N (N is number of pages)**
- **How many steps** must we make with B buffer pages?
  - **First step** produces runs of length B
  - **Second step** produces runs of length (B-1) * B
  - **Third step** produces runs of length (B-1) * (B-1) * B ...
  - Stop once $(B-1)^{steps-1} * B \geq N$, after $1+Ceil(log_{B-1}(N/B))$ steps

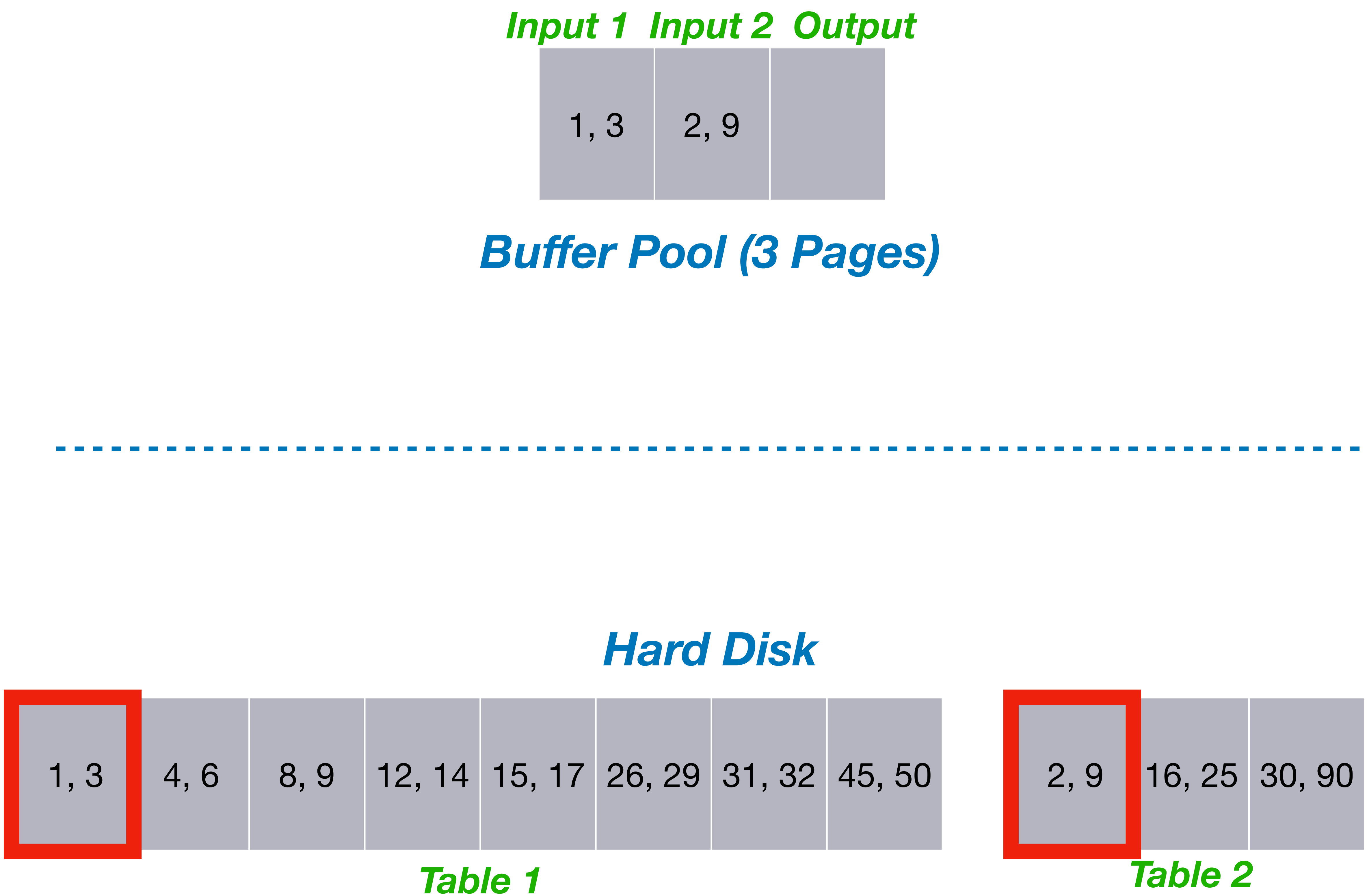
Join Phase 2: Overview

- (Have **sorted** both input tables by their join column)
- **Load first page** of both sorted tables into memory
- **Find matching tuples** and add to join result output
- **Load next page** for table with smallest last entry
- **Keep doing** until no pages left for one table

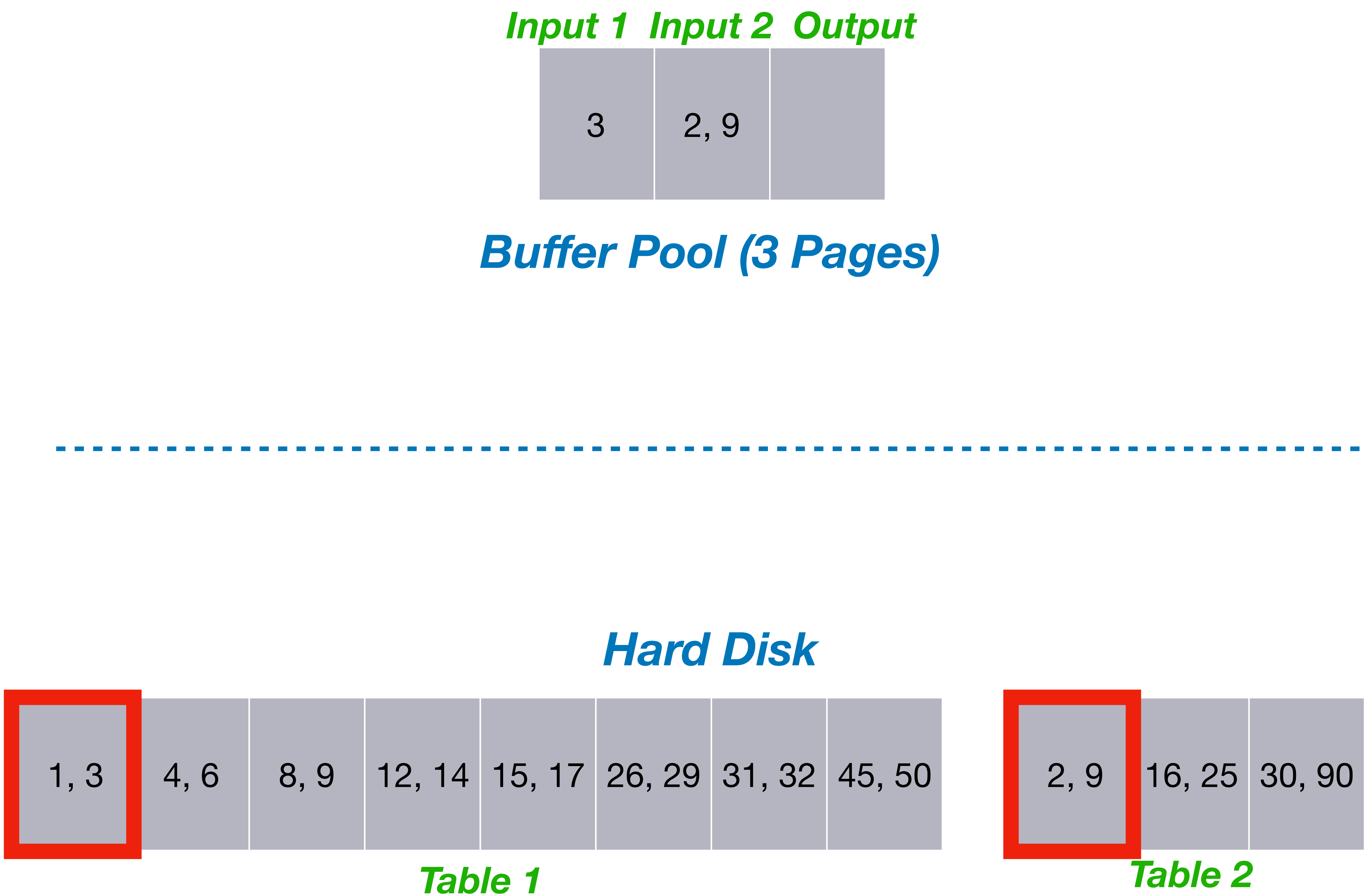
Join Phase 2 Example



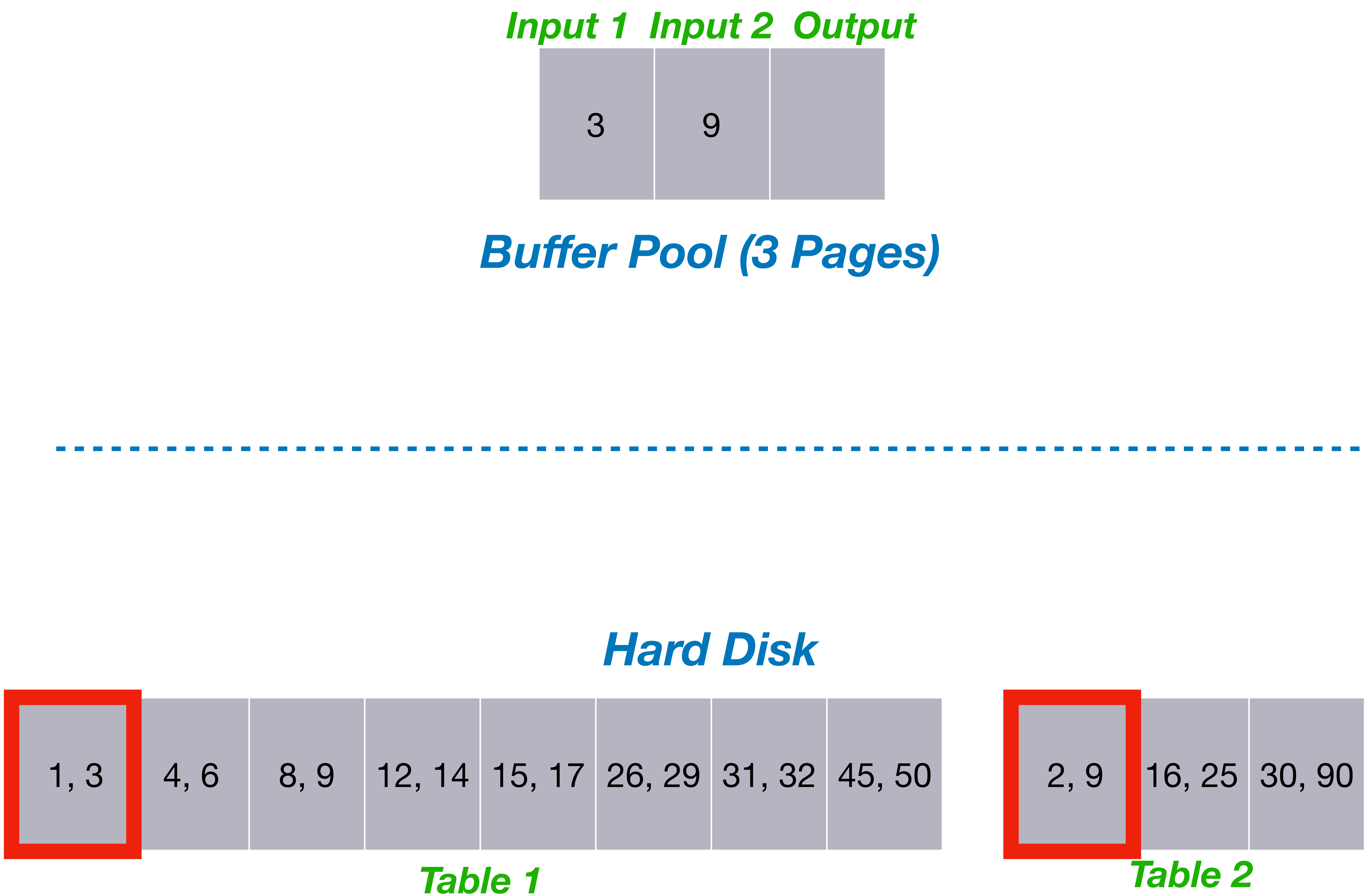
Join Phase 2 Example



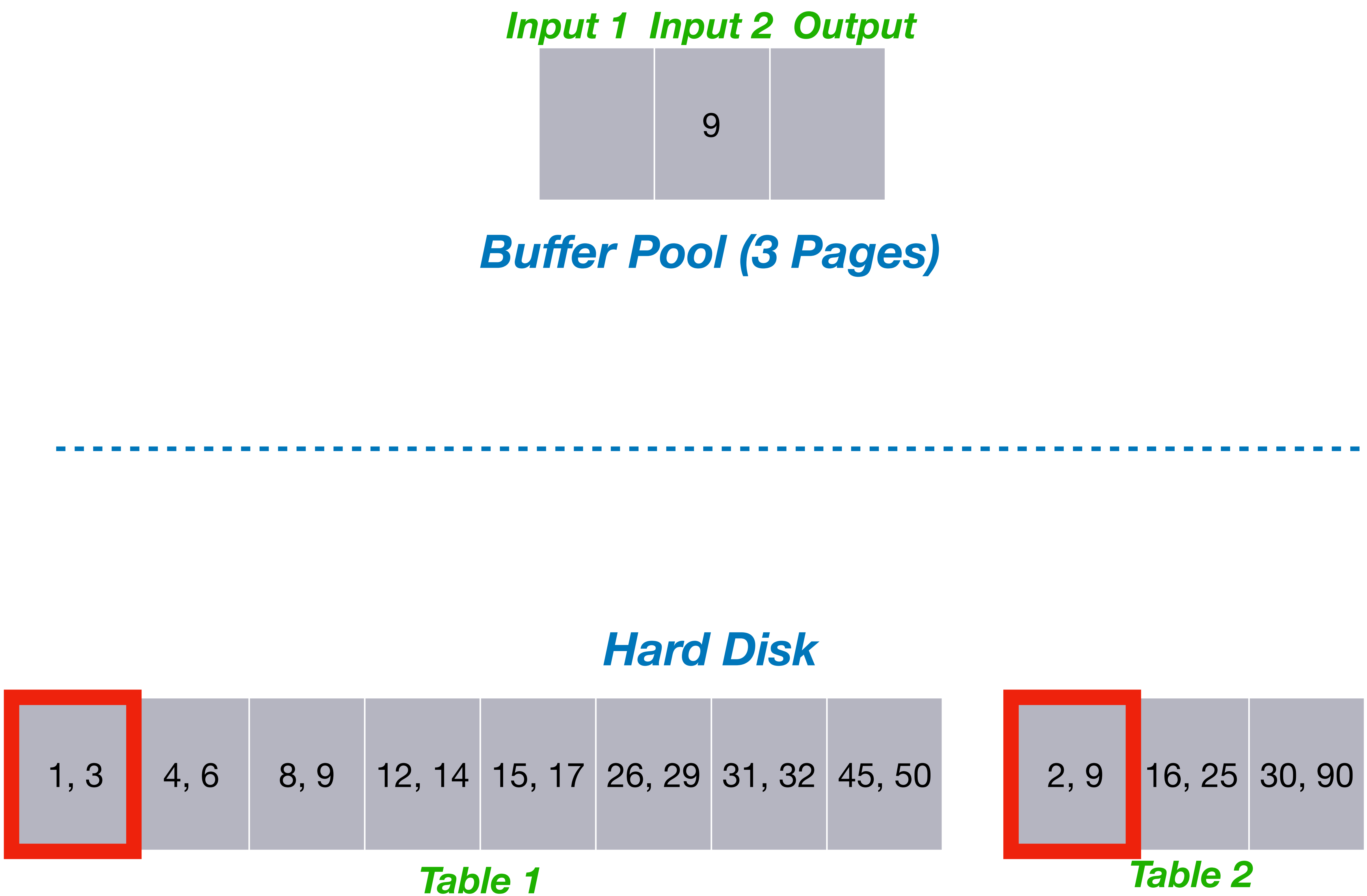
Join Phase 2 Example



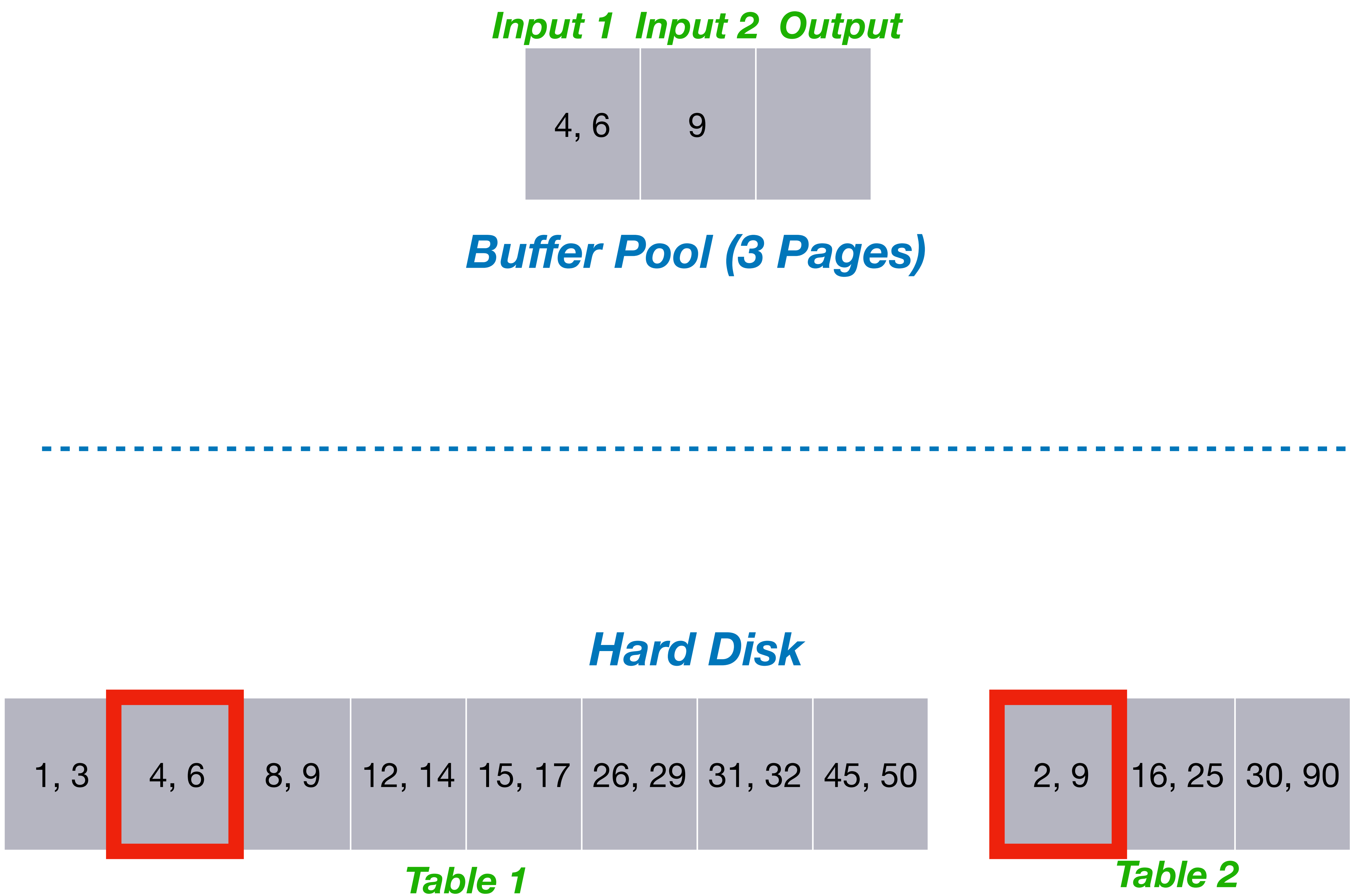
Join Phase 2 Example



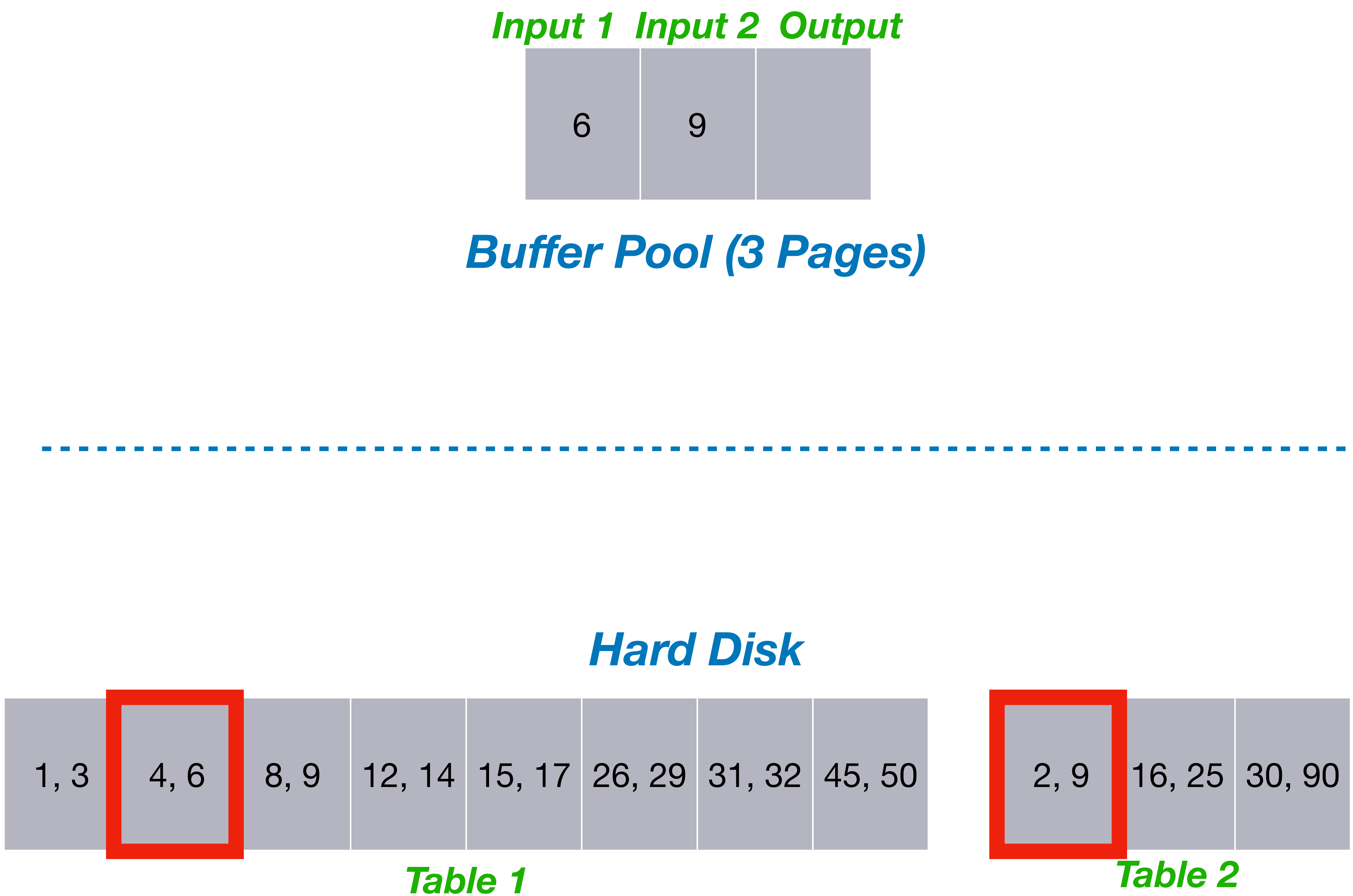
Join Phase 2 Example



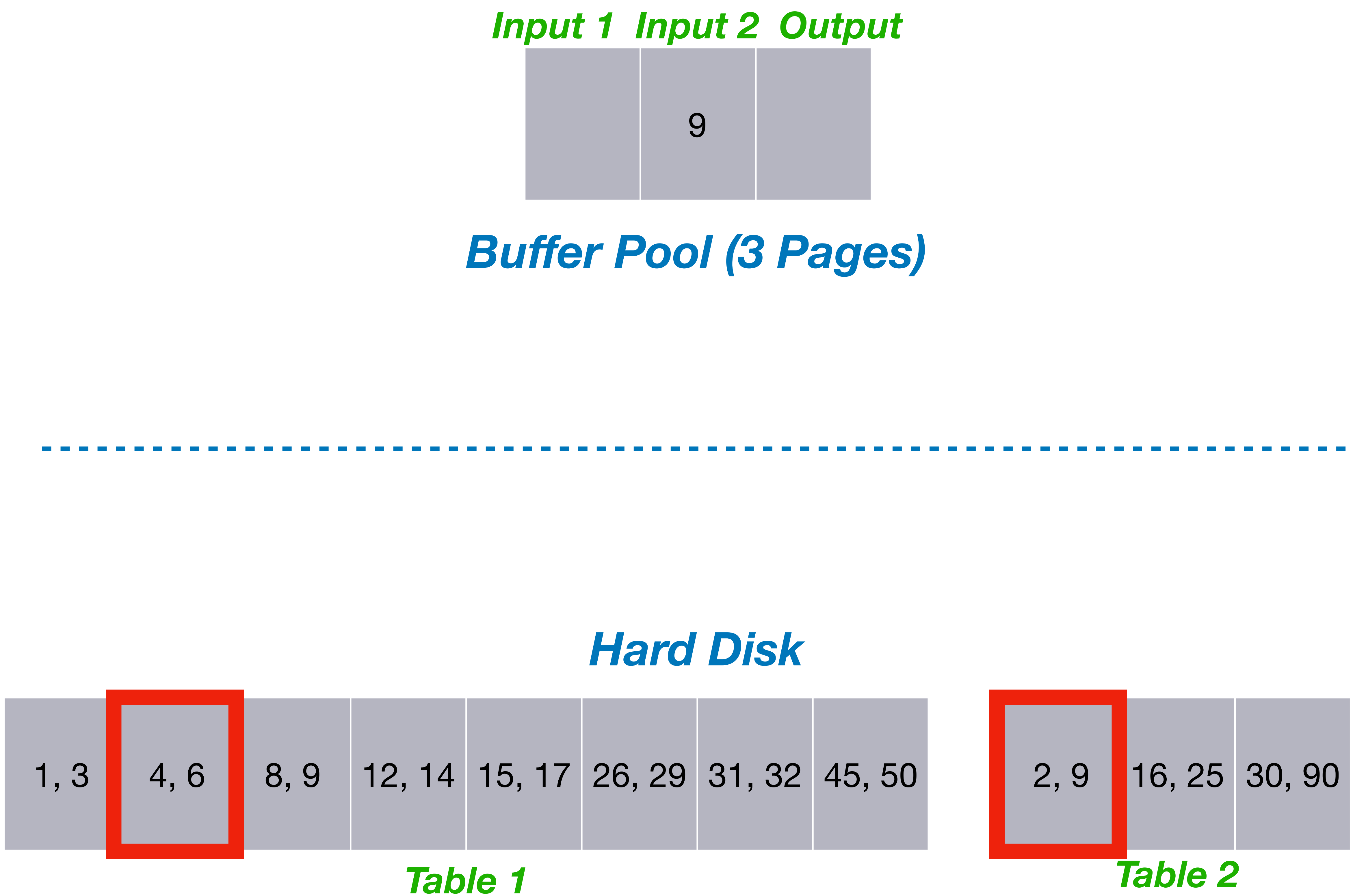
Join Phase 2 Example



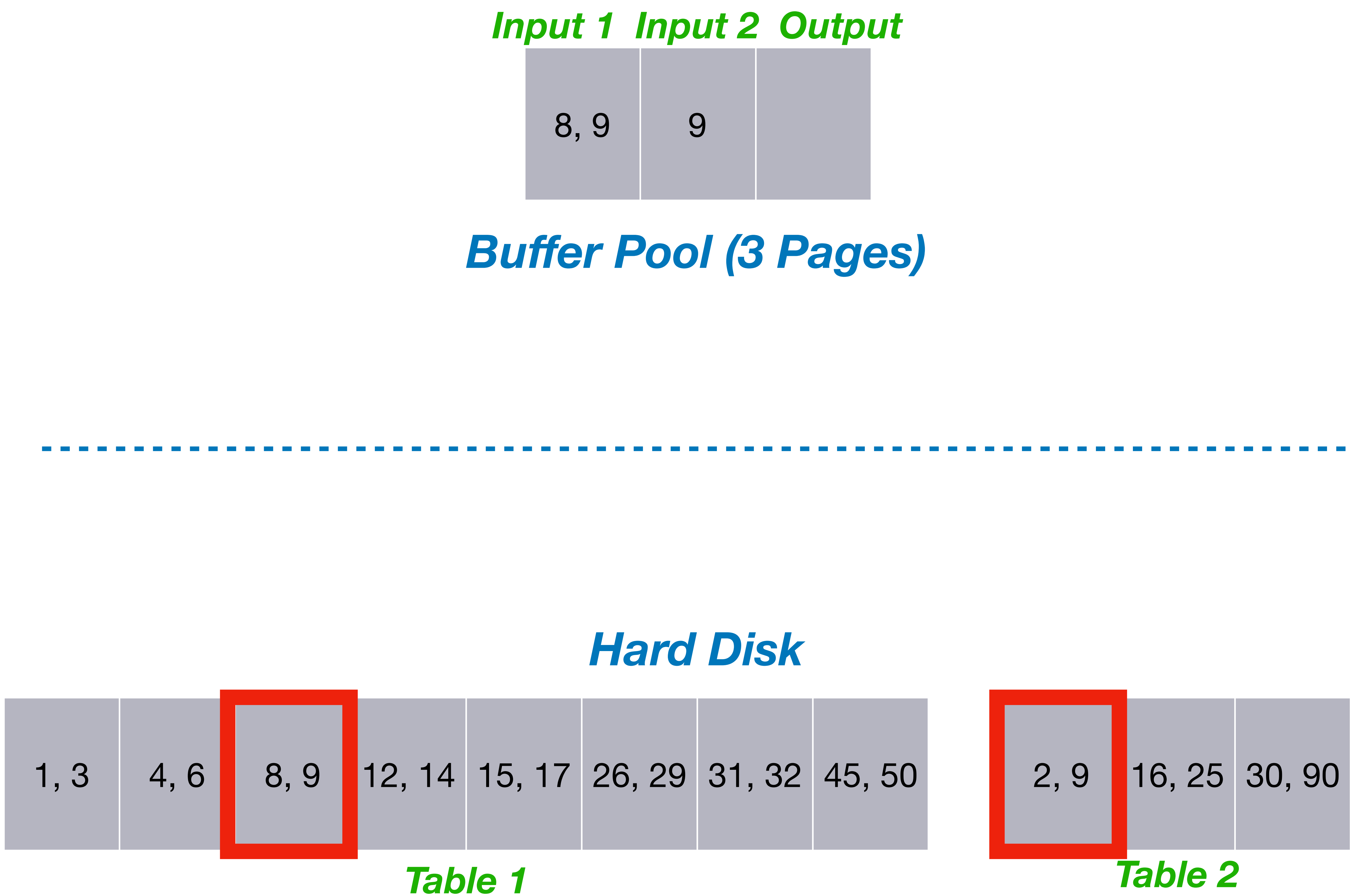
Join Phase 2 Example



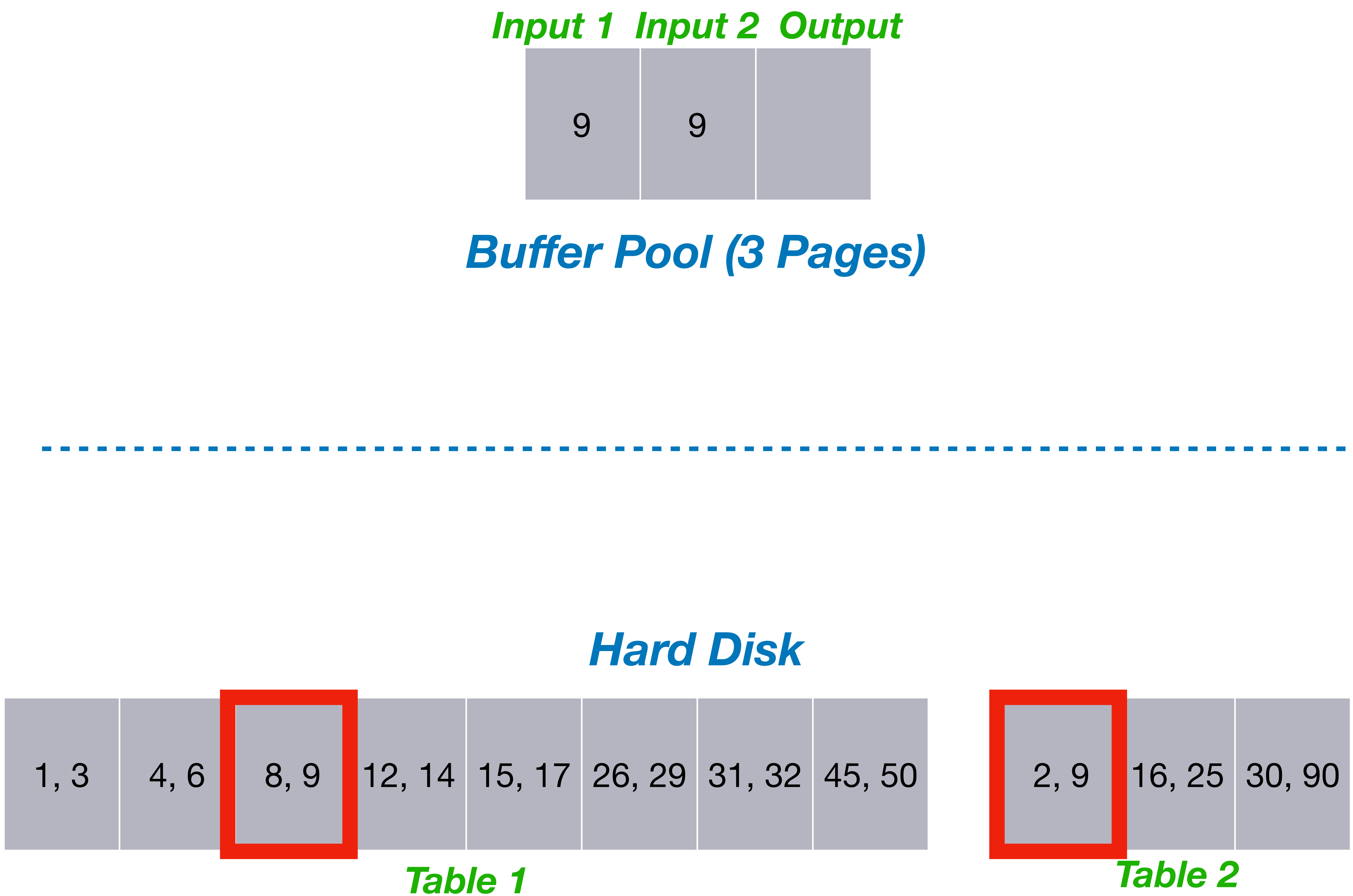
Join Phase 2 Example



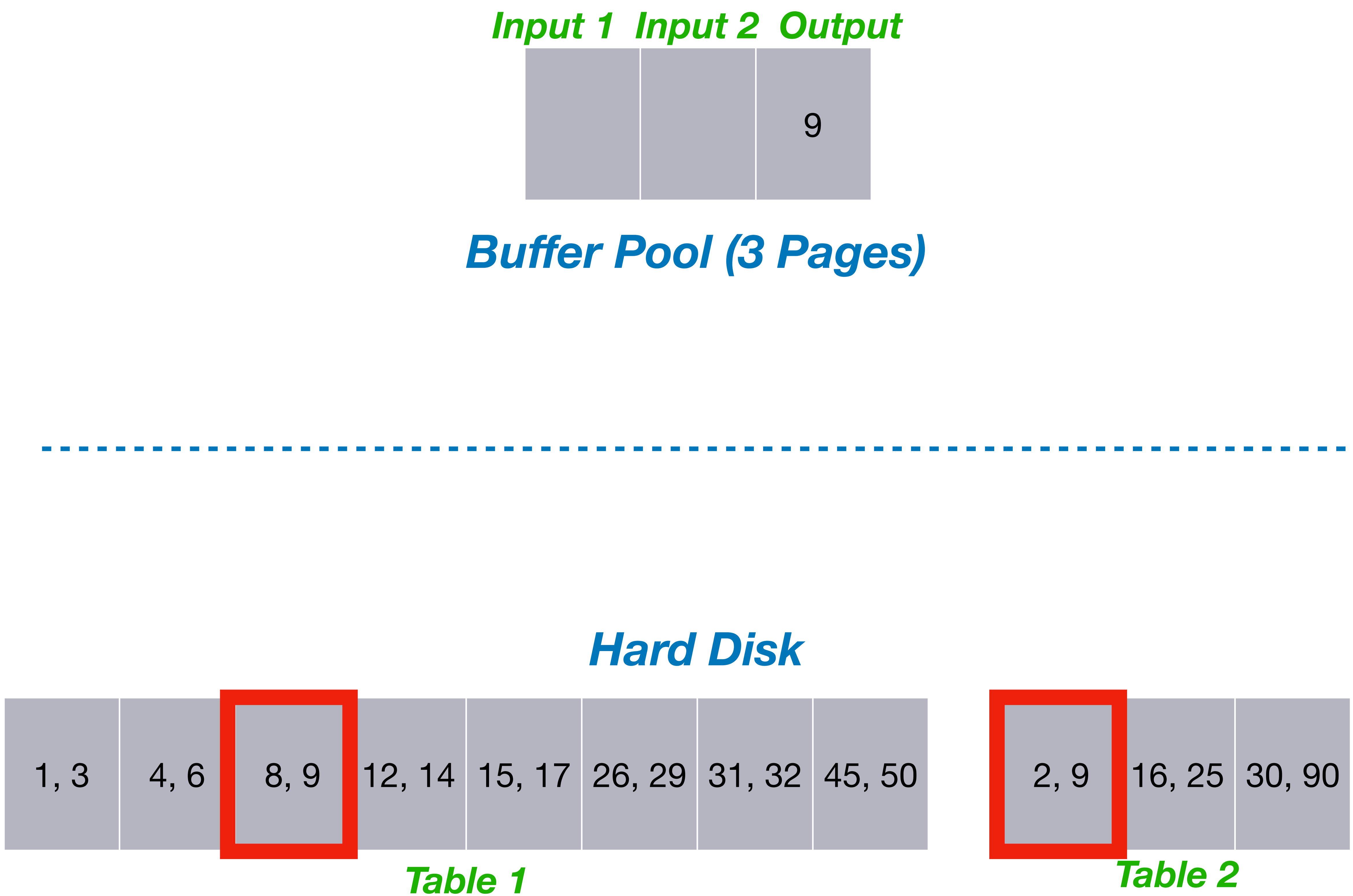
Join Phase 2 Example



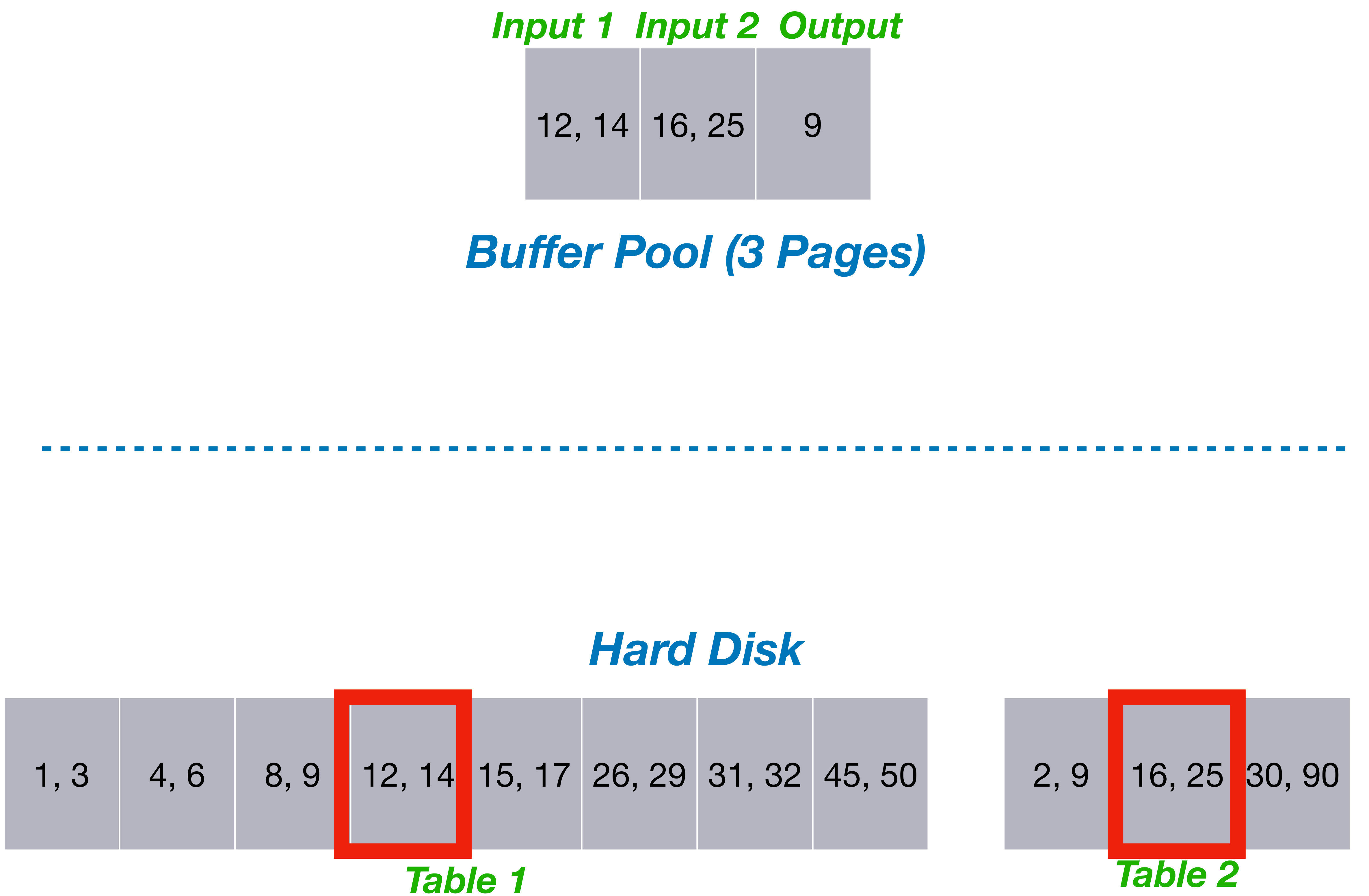
Join Phase 2 Example



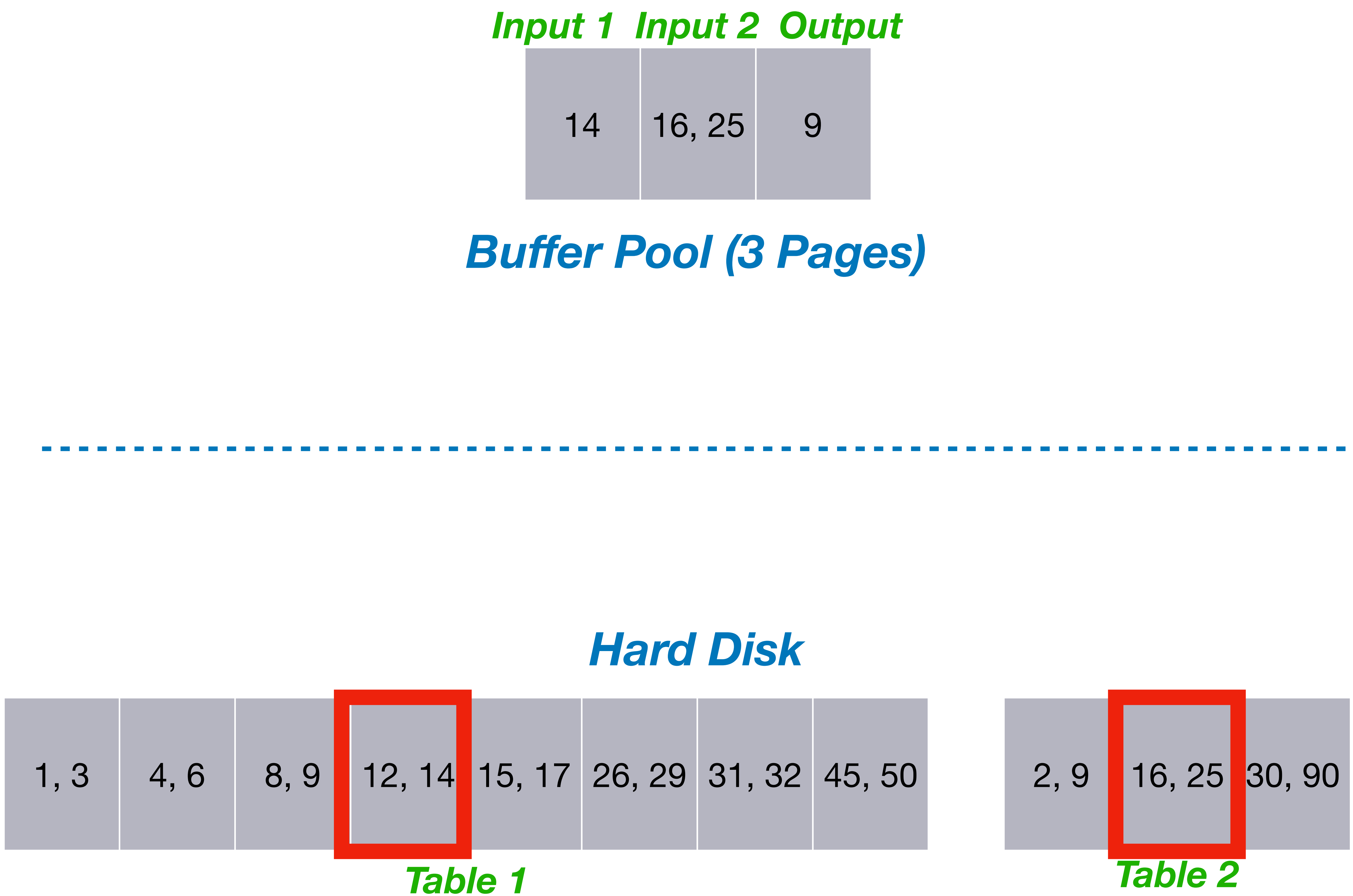
Join Phase 2 Example



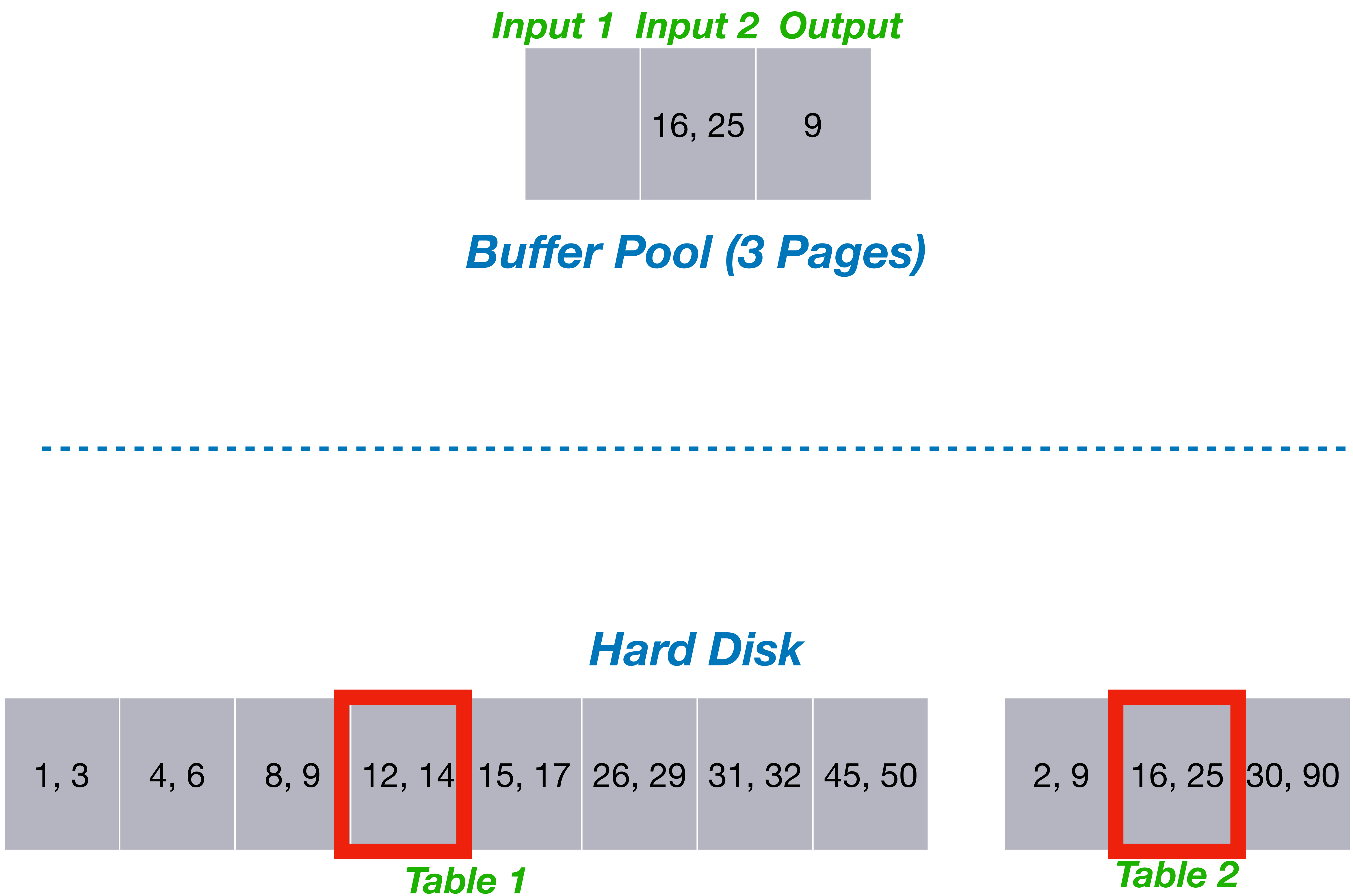
Join Phase 2 Example



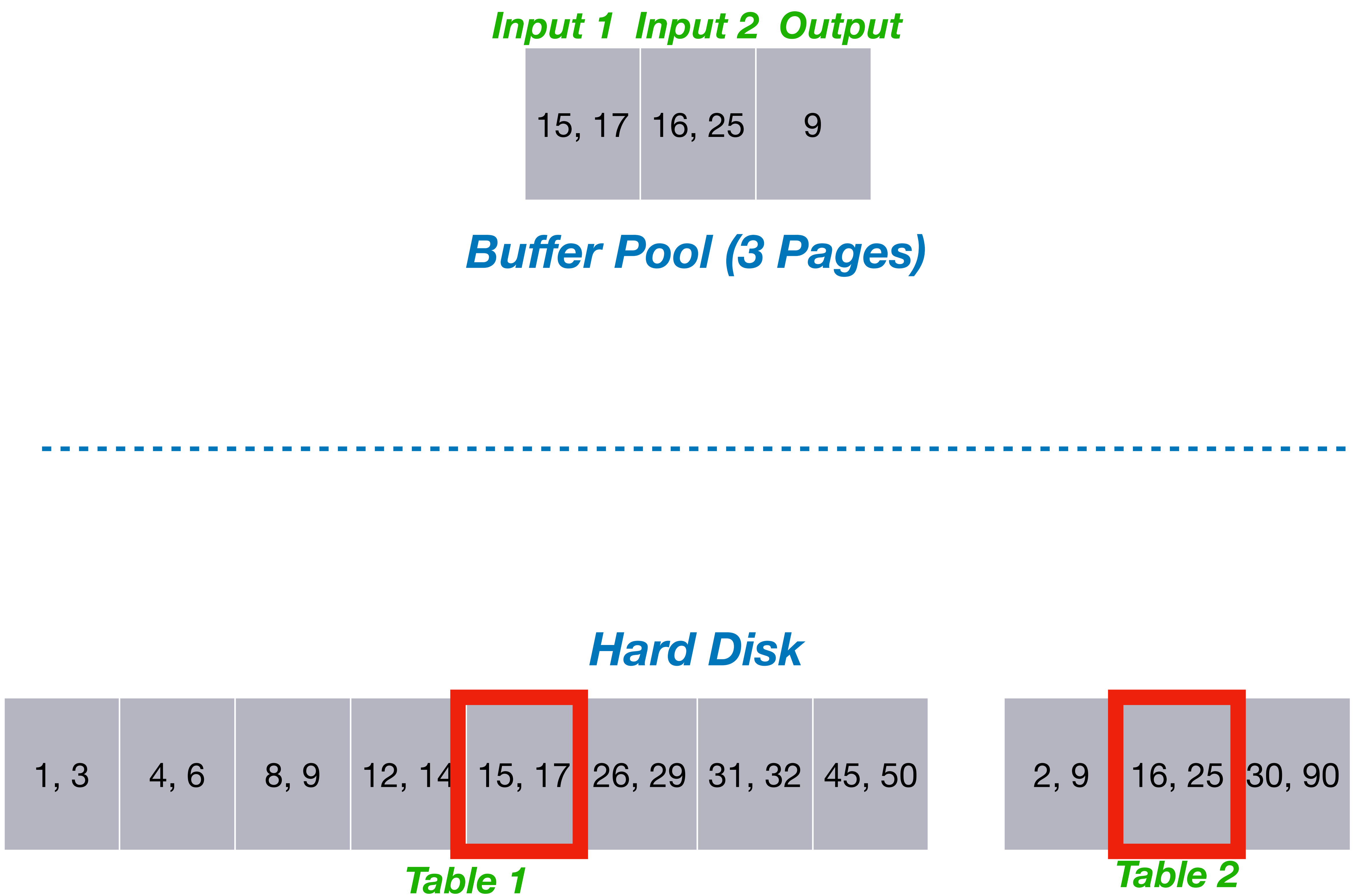
Join Phase 2 Example



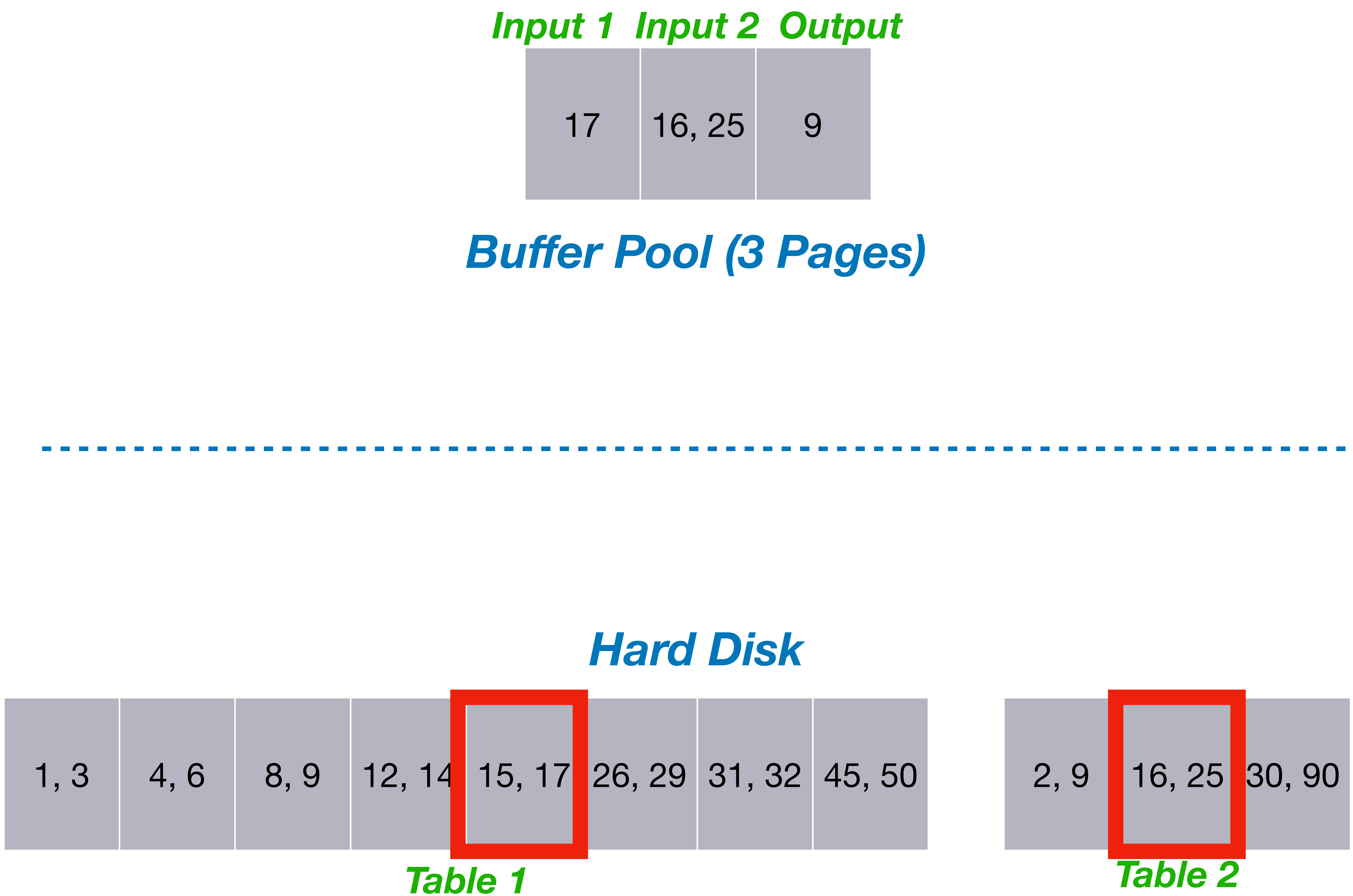
Join Phase 2 Example



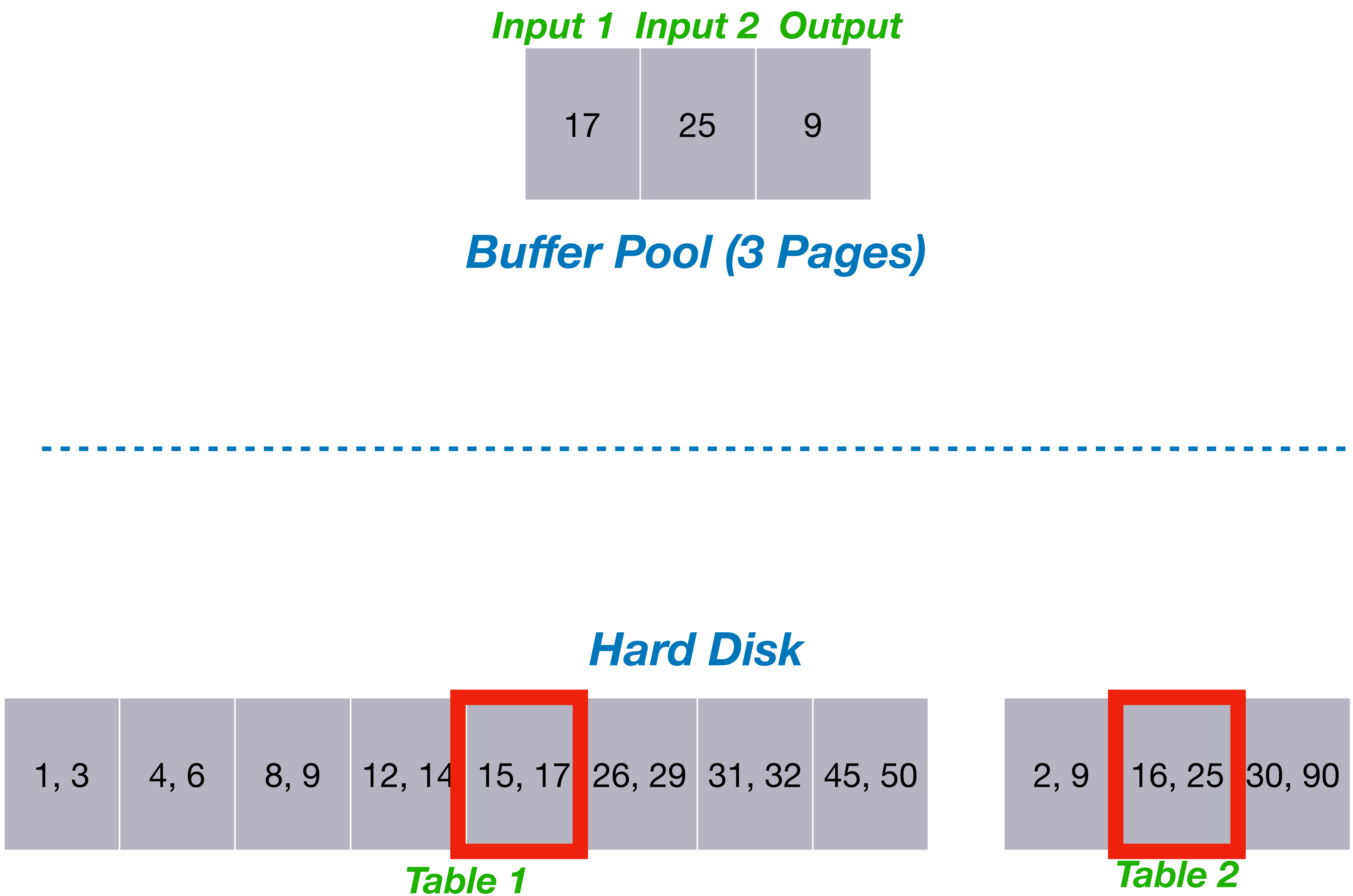
Join Phase 2 Example



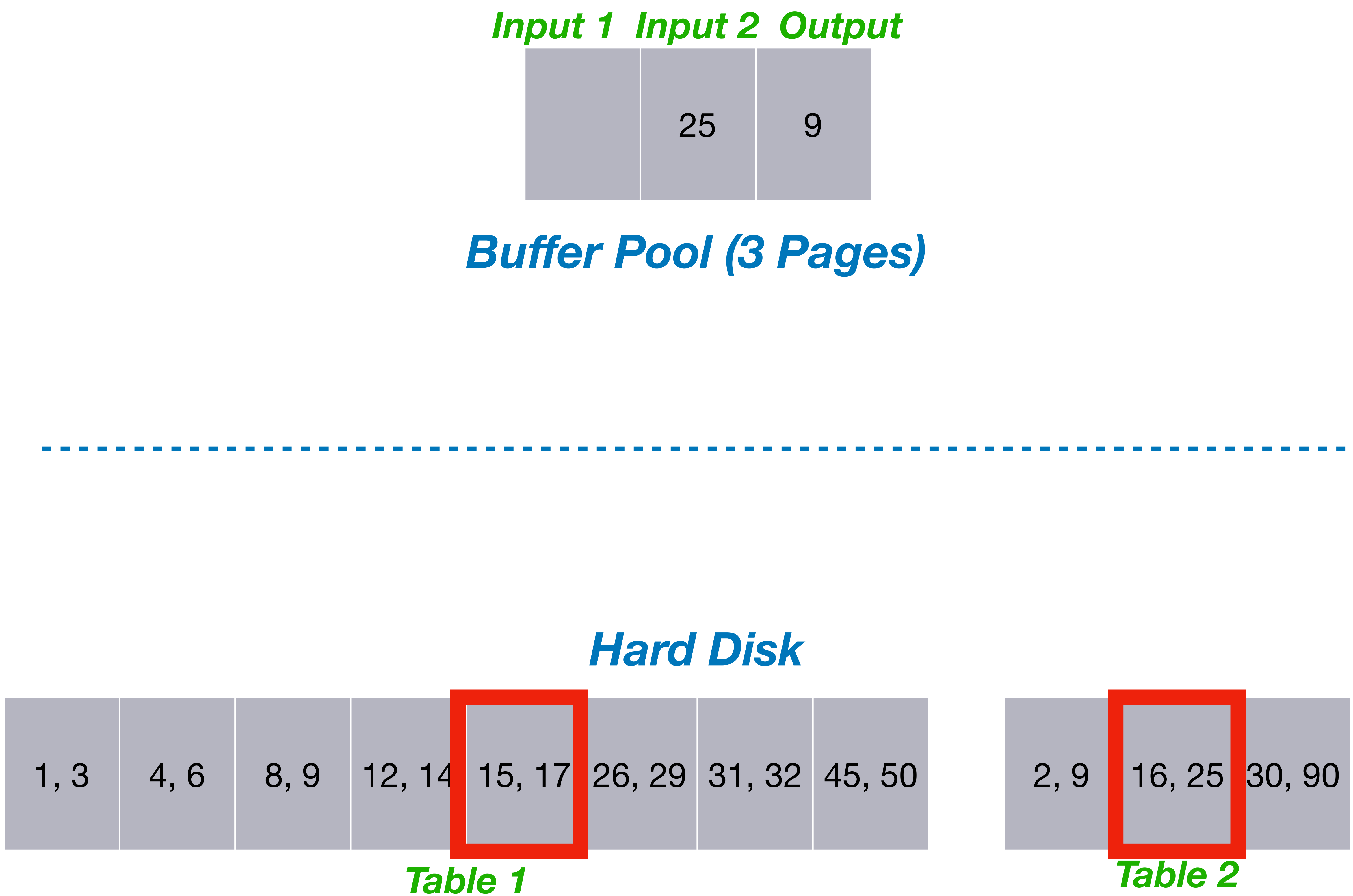
Join Phase 2 Example



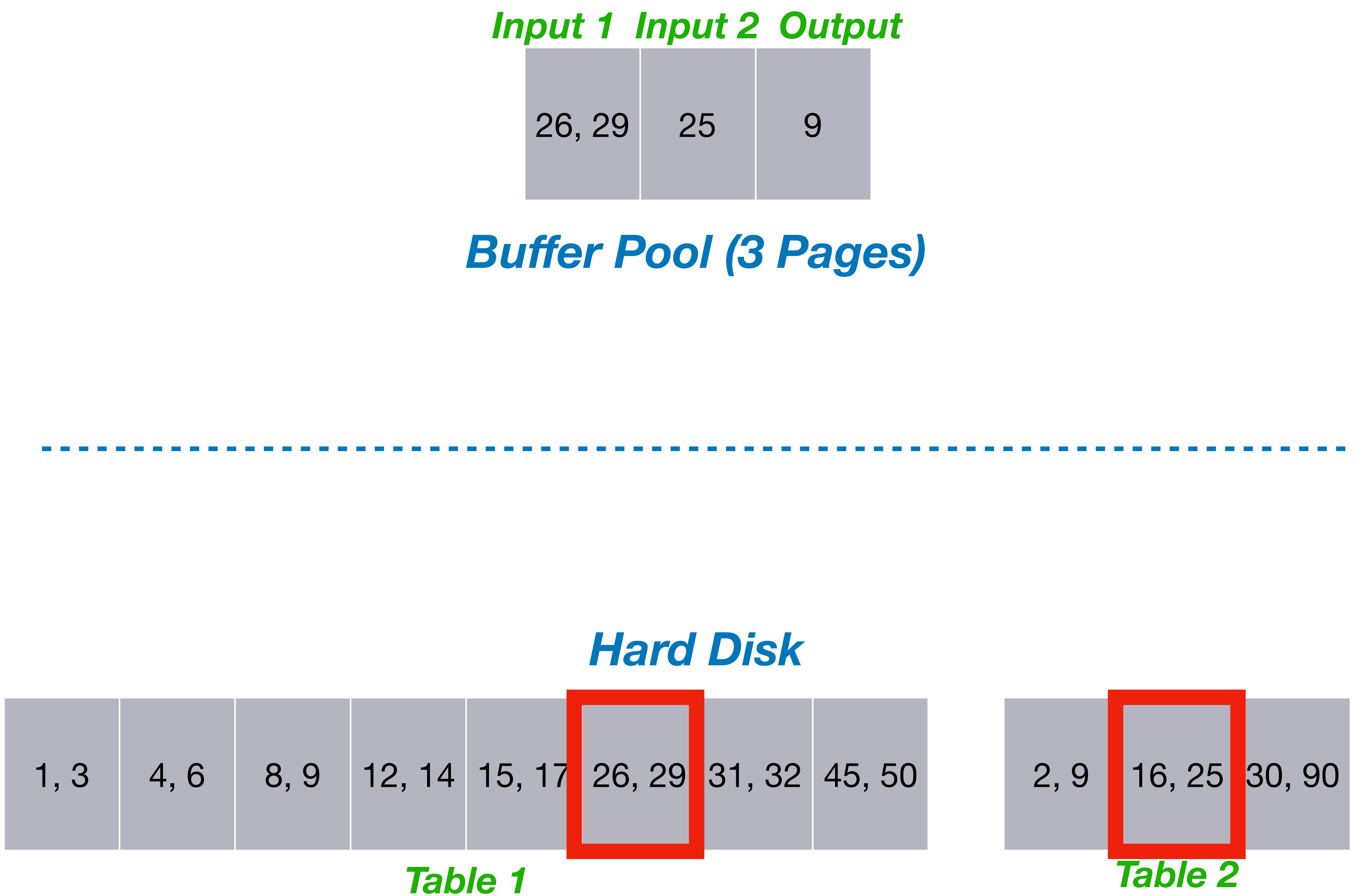
Join Phase 2 Example



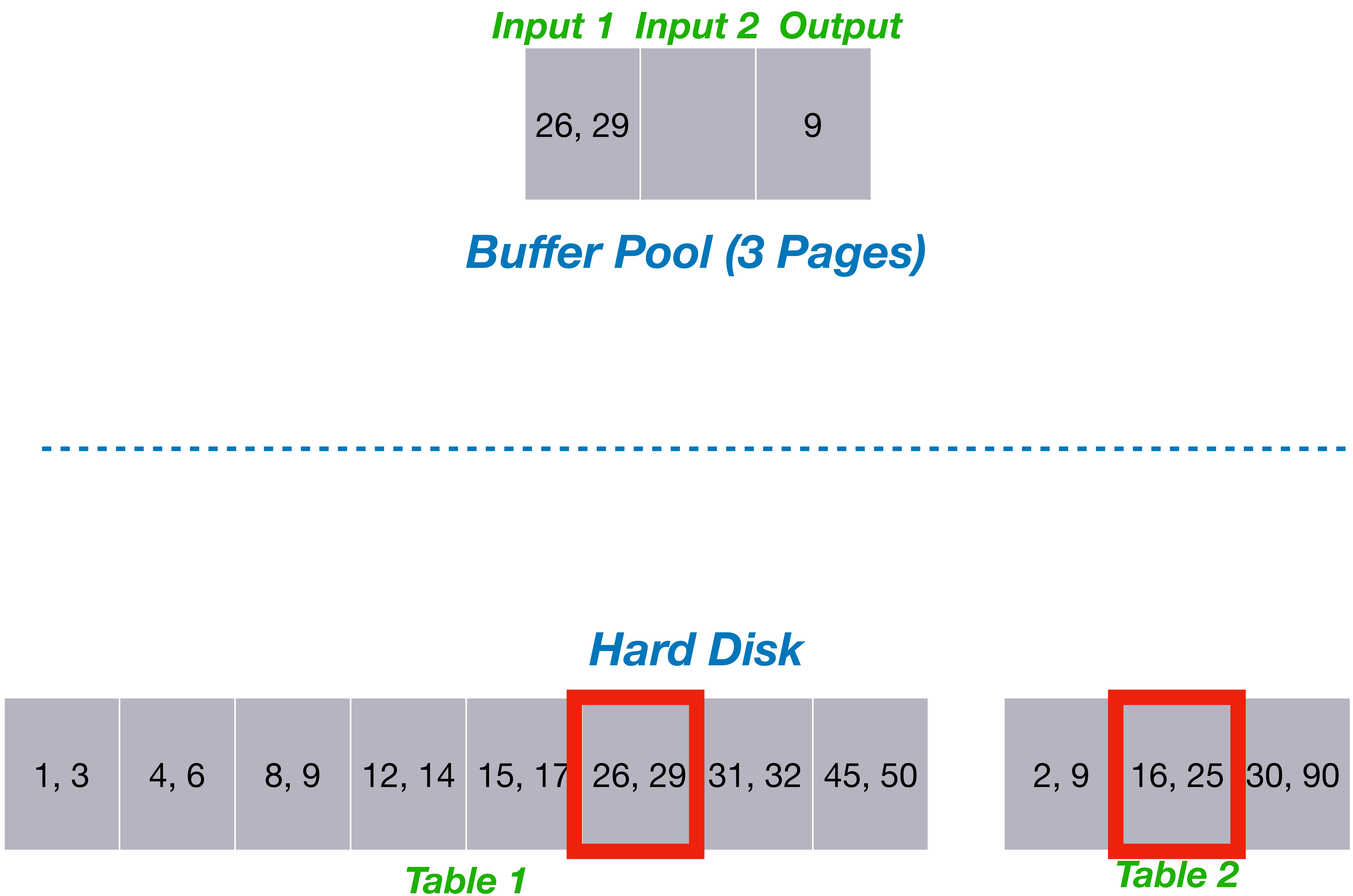
Join Phase 2 Example



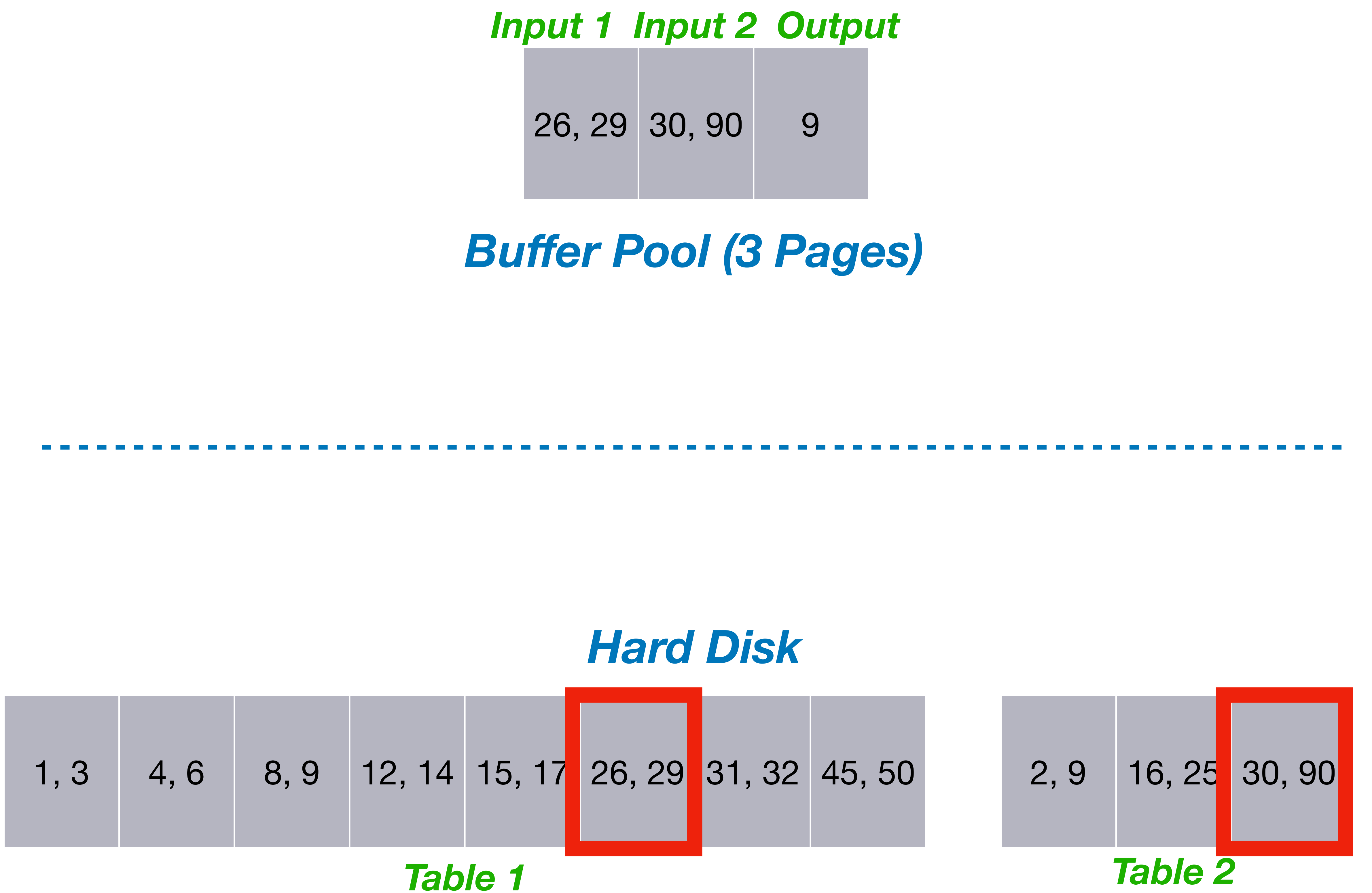
Join Phase 2 Example



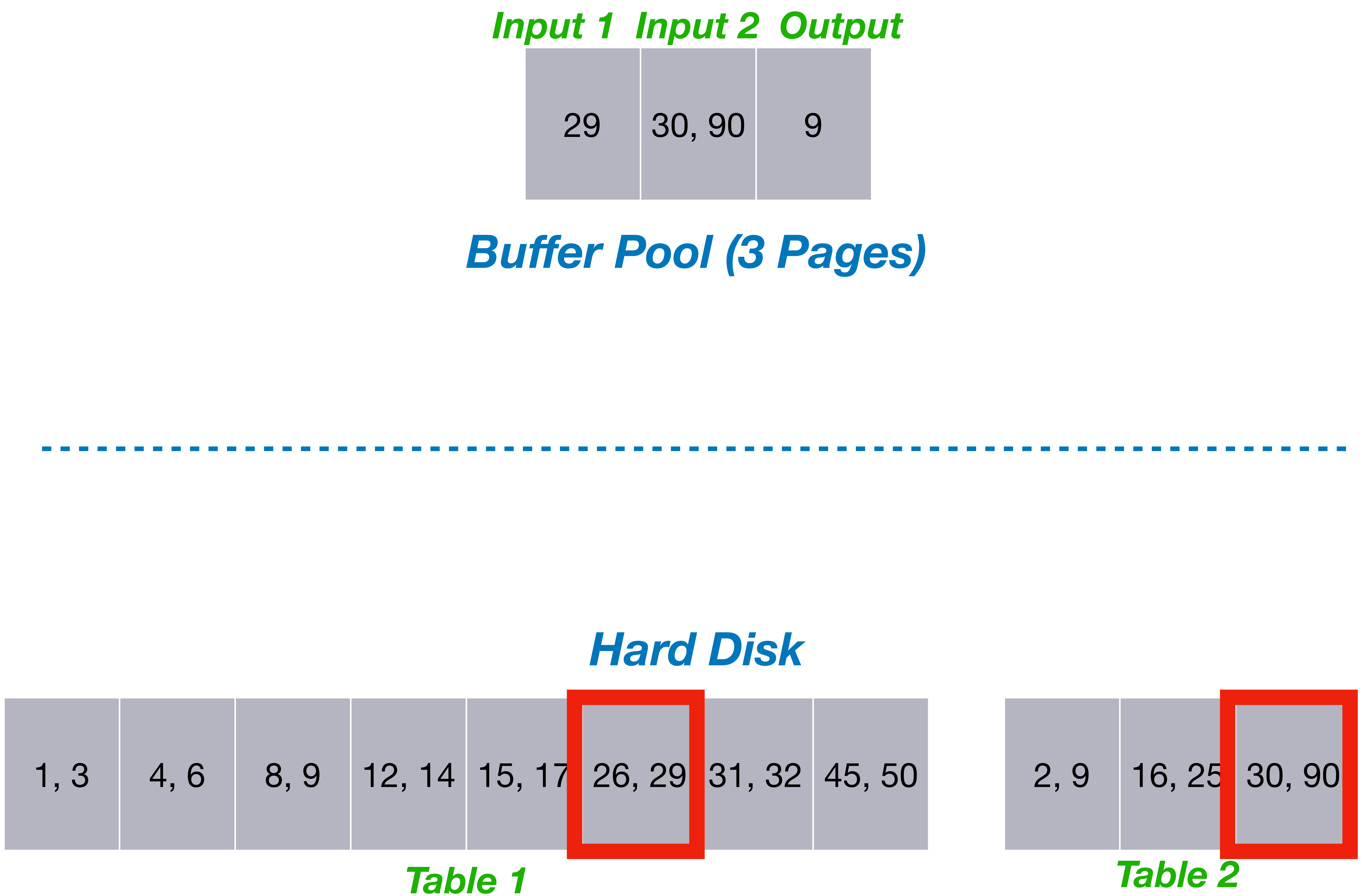
Join Phase 2 Example



Join Phase 2 Example



Join Phase 2 Example



Handling Many Duplicates

- May have duplicates over **multiple pages**
- **Must revert** to first page with duplicate whenever we load new page from other table
- This makes the join more **expensive**

Cost Analysis (Phase 2)

- For now: assume that all **duplicate entries** on same page
  - Duplicate entry: **same value in join column**
- Means that each input page is **only read once**
- Cost is proportional to **number of input pages**
  - I.e., Pages from **both** input tables

Total Join Cost

- Two input tables with M and N pages, B buffer pages
- First phase has cost
  - **$2*M*(1+Ceil(log_{B-1}(M/B)))$** for sorting table 1
  - **$2*N*(1+Ceil(log_{B-1}(M/B)))$** for sorting table 2
- Second phase has cost
  - **M+N** (we don't count cost for writing output!)

How Much Memory?

- First phase: try to exploit **all buffer pages**
  - More buffer means less merging passes!
- Second phase: only exploit **three buffer pages**
  - One for first input, one for second input, one output

How Much Memory?

- First phase: try to exploit **all buffer pages**
    - More buffer means less merging passes!
- Second phase: only exploit **three buffer pages**
    - One for first input, one for second input, one output

Seems Sub-Optimal!

Refined Sort-Merge Join

- Idea: can merge **more than two** sorted tables in phase 2
- Hence, **do not need to sort** tables completely in phase 1
- Means we can **save steps** (i.e., passes over the data)
- **First phase**: only sort data chunks that fit into memory
- **Second phase**: join all sorted chunks together (one step)

Refined Join Details

- Assume B buffer pages, tables with N and M pages
- **First phase**: load chunks of B pages, sort, write back
  - We now have (N+M)/B sorted chunks on disk
- **Second phase**: merge B-1 sorted chunks together
  - Can sort entries in-memory to find matches
- Cost is **2*(M+N)** (Phase 1) **+ 1 * (M+N)** (Phase 2)

How Much Memory?

- Again, B buffer pages, input sizes are M and N
- Have **(N+M)/B** sorted runs after first phase
- Need **B-1 ≥ (N+M)/B** to merge them in one step
- Rule of thumb if N>M: need **B ≥ 2*Sqrt(N)**

R-SMJ vs. Hash Join

	Hash Join	Refined Sort-Merge Join
Time	3 * Input Size	3 * Input Size
Memory	**> Sqrt(Smaller Table Size)**	> 2 * Sqrt(Larger Table Size)
Parallelization	**Advantage**	
Skew-Resistance		**Advantage**