# CS186 Discussion #3

(Joins)

# Keys

superkey

| inst letters | course | first name | last name | grade |
|---|---|---|---|---|
| ab | cs186 | Bob | Smith | B |
| ac | cs186 | Joe | Hellerstein | A+ |
| ab | cs162 | Eric | Brewer | A |
| ad | cs160 | Toby | Brown | C |

# Keys

superkey

| inst letters | course | first name | last name | grade |
|---|---|---|---|---|
| ab | cs186 | Bob | Smith | B |
| ac | cs186 | Joe | Hellerstein | A+ |
| ab | cs162 | Eric | Brewer | A |
| ad | cs160 | Toby | Brown | C |

# Keys

superkey
candidate key

| inst letters | course | first name | last name | grade |
|---|---|---|---|---|
| ab | cs186 | Bob | Smith | B |
| ac | cs186 | Joe | Hellerstein | A+ |
| ab | cs162 | Eric | Brewer | A |
| ad | cs160 | Toby | Brown | C |

# Keys

superkey
candidate key

| inst letters | course | first name | last name | grade |
|---|---|---|---|---|
| ab | cs186 | Bob | Smith | B |
| ac | cs186 | Joe | Hellerstein | A+ |
| ab | cs162 | Eric | Brewer | A |
| ad | cs160 | Toby | Brown | C |

* This is only true if no two people have the same first and last name!

# Last week…

```
SELECT S.name
FROM Students S, Grades G;
```

| S.name | S.id | S.gpa | G.id | G.class | G.grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Bob | 1 | 3.7 | 3 | cs70 | B |
| Bob | 1 | 3.7 | 4 | cs61a | A |
| Bob | 1 | 3.7 | 2 | cs61c | D |
| Bob | 1 | 3.7 | 4 | cs170 | A |
| Sue | 2 | 2.9 | 1 | cs186 | C |
| Sue | 2 | 2.9 | 1 | cs164 | A |
| Sue | 2 | 2.9 | 3 | cs70 | B |
| Sue | 2 | 2.9 | 4 | cs61a | A |

# Last week…

```
SELECT S.name
FROM Students S, Grades G
WHERE S.id = G.id;
```

| S.name | S.id | S.gpa | G.id | G.class | G.grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Sue | 2 | 2.9 | 2 | cs61c | D |
| Ron | 3 | 1.2 | 3 | cs70 | B |
| Al | 4 | 4.0 | 4 | cs61a | A |
| Al | 4 | 4.0 | 4 | cs170 | A |

```sql
SELECT S.name
FROM Students S INNER JOIN Grades G
ON S.id = G.id;
```

## Students

| name | id | gpa |
|---|---|---|
| Bob | 1 | 3.7 |
| Sue | 2 | 2.9 |
| Ron | 3 | 1.2 |
| Al | 4 | 4.0 |
| Sally | 5 | 3.6 |
| Bob | 6 | 2.1 |
| Joe | 9 | 4.0 |

## Grades

| id | class | grade |
|---|---|---|
| 1 | cs186 | C |
| 1 | cs164 | A |
| 3 | cs70 | B |
| 4 | cs61a | A |
| 2 | cs61c | D |
| 4 | cs170 | A |
| 7 | cs61C | F |

**SELECT** S.name
**FROM** Students S **INNER JOIN** Grades G
**ON** S.id = G.id;

Students

| name | id | gpa |
|---|---|---|
| Bob | 1 | 3.7 |
| Sue | 2 | 2.9 |
| Ron | 3 | 1.2 |
| Al | 4 | 4.0 |
| Sally | 5 | 3.6 |
| Bob | 6 | 2.1 |
| Joe | 9 | 4.0 |

Grades

| id | class | grade |
|---|---|---|
| 1 | cs186 | C |
| 1 | cs164 | A |
| 3 | cs70 | B |
| 4 | cs61a | A |
| 2 | cs61c | D |
| 4 | cs170 | A |
| 7 | cs61C | F |

```sql
SELECT S.name
FROM Students S INNER JOIN Grades G
ON S.id = G.id;
```

| name | id | gpa | id | class | grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Ron | 3 | 1.2 | 3 | cs70 | B |
| Al | 4 | 4.0 | 4 | cs61a | A |
| Sue | 2 | 2.9 | 2 | cs61c | D |
| Al | 4 | 4.0 | 4 | cs170 | A |

```
SELECT S.name
FROM Students S LEFT OUTER JOIN Grades G
ON S.id = G.id;
```

| name | id | gpa | id | class | grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Ron | 3 | 1.2 | 3 | cs70 | B |
| Al | 4 | 4.0 | 4 | cs61a | A |
| Sue | 2 | 2.9 | 2 | cs61c | D |
| Al | 4 | 4.0 | 4 | cs170 | A |
| Sally | 5 | 3.6 |  |  |  |
| Bob | 6 | 2.1 |  |  |  |
| Joe | 9 | 4.0 |  |  |  |

**SELECT** S.name
**FROM** Students S **RIGHT OUTER JOIN** Grades G
**ON** S.id = G.id;

| name | id | gpa | id | class | grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Ron | 3 | 1.2 | 3 | cs70 | B |
| Al | 4 | 4.0 | 4 | cs61a | A |
| Sue | 2 | 2.9 | 2 | cs61c | D |
| Al | 4 | 4.0 | 4 | cs170 | A |
|  |  |  | 7 | cs61C | F |

```
SELECT S.name
FROM Students S FULL OUTER JOIN Grades G
ON S.id = G.id;
```

| name | id | gpa | id | class | grade |
|---|---|---|---|---|---|
| Bob | 1 | 3.7 | 1 | cs186 | C |
| Bob | 1 | 3.7 | 1 | cs164 | A |
| Ron | 3 | 1.2 | 3 | cs70 | B |
| Al | 4 | 4.0 | 4 | cs61a | A |
| Sue | 2 | 2.9 | 2 | cs61c | D |
| Al | 4 | 4.0 | 4 | cs170 | A |
| Sally | 5 | 3.6 | | | |
| Bob | 6 | 2.1 | | | |
| Joe | 9 | 4.0 | | | |
| | | | 7 | cs61C | F |

# Simplifying Subqueries

```
SELECT sname
FROM
     (SELECT sid
      FROM Reserves
      EXCEPT
             (SELECT sid
              FROM
                      (SELECT Reserves.sid, PinkBoats.bid
                      FROM Reserves,
                             (SELECT bid
                             FROM Boats
                             WHERE color=’pink’) PinkBoats
                             EXCEPT SELECT sid, bid
                             FROM Reserves)))
R, Sailors S
WHERE R.sid = S.sid;
```

# Simplifying Subqueries

```
WITH Res(sid, bid) AS (SELECT Reserves.sid, PinkBoats.bid
                      FROM Reserves,
                             (SELECT bid
                             FROM Boats
                             WHERE color='pink') PinkBoats
                             EXCEPT SELECT sid, bid
                             FROM Reserves))
SELECT sname
FROM
     (SELECT sid
      FROM Reserves
      EXCEPT
             (SELECT sid
              FROM Res)
R, Sailors S
WHERE R.sid = S.sid;
```

# Worksheet: Advanced SQL Questions:1, 3, 5

Find all album id’s and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output.

Find all album id’s and names for every artist active since 2000 or later. If an artist does not have any albums, you should still include the artist's information in your output.

```
SELECT Ar.artist_id, Ar.artist_name,
Al.album_id, Al.album_name

FROM Artists Ar LEFT OUTER JOIN Albums
Al

ON Ar.artist_id=Al.artist_id

WHERE Ar.first_year_active >= 2000;
```

Find the id and name for each artist who has albums of genre “pop” and “rock”.

# Find the id and name for each artist who has albums of genre “pop” and “rock”.

```
SELECT Ar.artist_id, Ar.artist_name

FROM Albums Al1 INNER JOIN Albums Al2 ON
Al1.artist_id=Al2.artist_id INNER JOIN
Artists Ar ON
Al2.artist_id=Ar.artist_id;

WHERE Al1.genre=“pop" AND
Al2.genre="rock"
```

Find all artists who released songs in 2014 or released songs that spent more than 10 weeks in the top 40.

# Find all artists who released songs in 2014 or released songs that spent more than 10 weeks in the top 40.

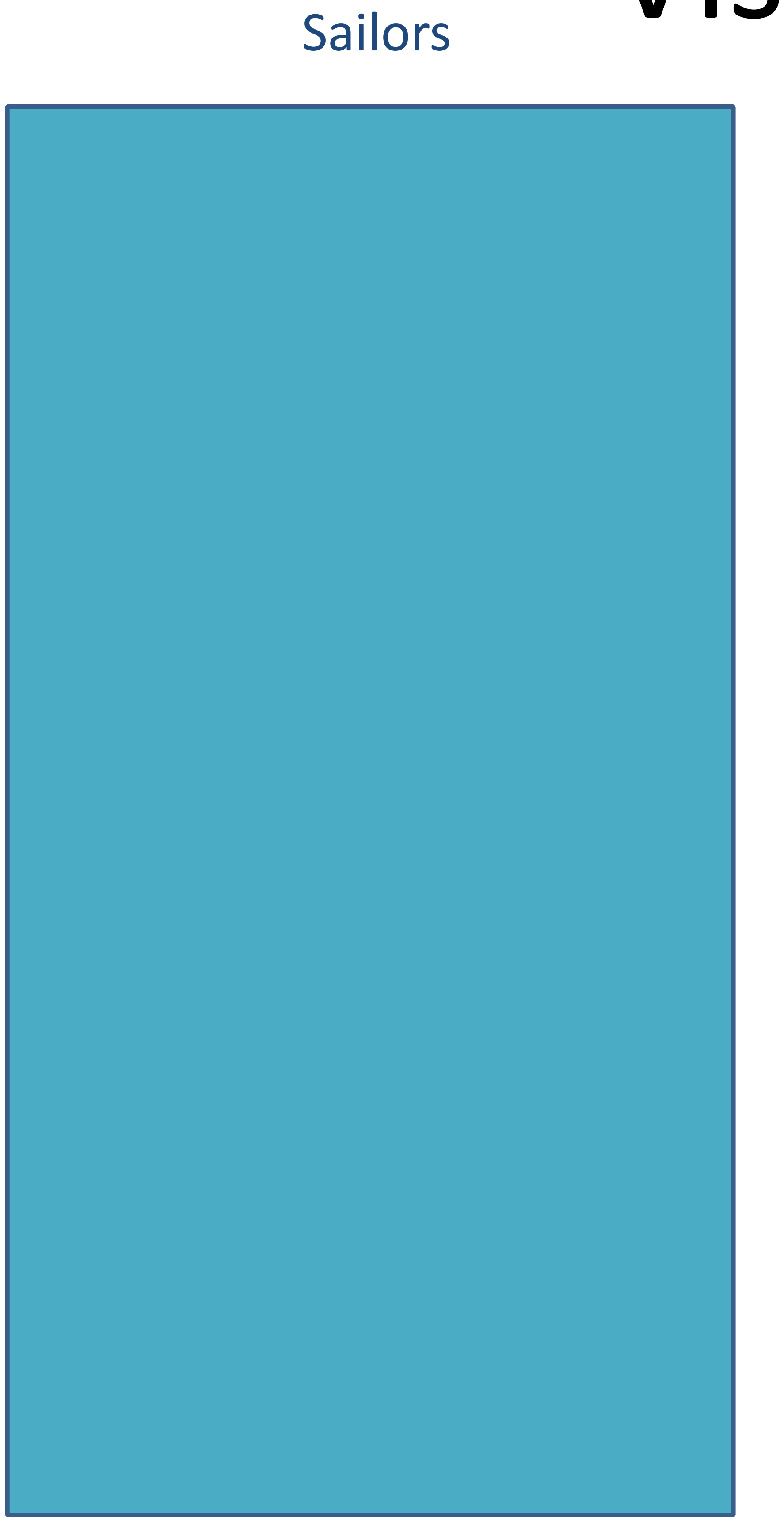
```
WITH Temp(artist_id) AS
    (SELECT Al.artist_id
    FROM Albums Al
    WHERE Al.year_released=2014
    UNION
    SELECT Al.artist_id
    FROM Albums Al
    INNER JOIN Songs S ON Al.album_id=S.album_id
    WHERE S.weeks_in_top_40>10)
SELECT Ar.artist_id, Ar.artist_name
FROM Artists Ar
INNER JOIN Temp T ON Ar.artist_id=T.artist_id;
```

# Join Algorithms

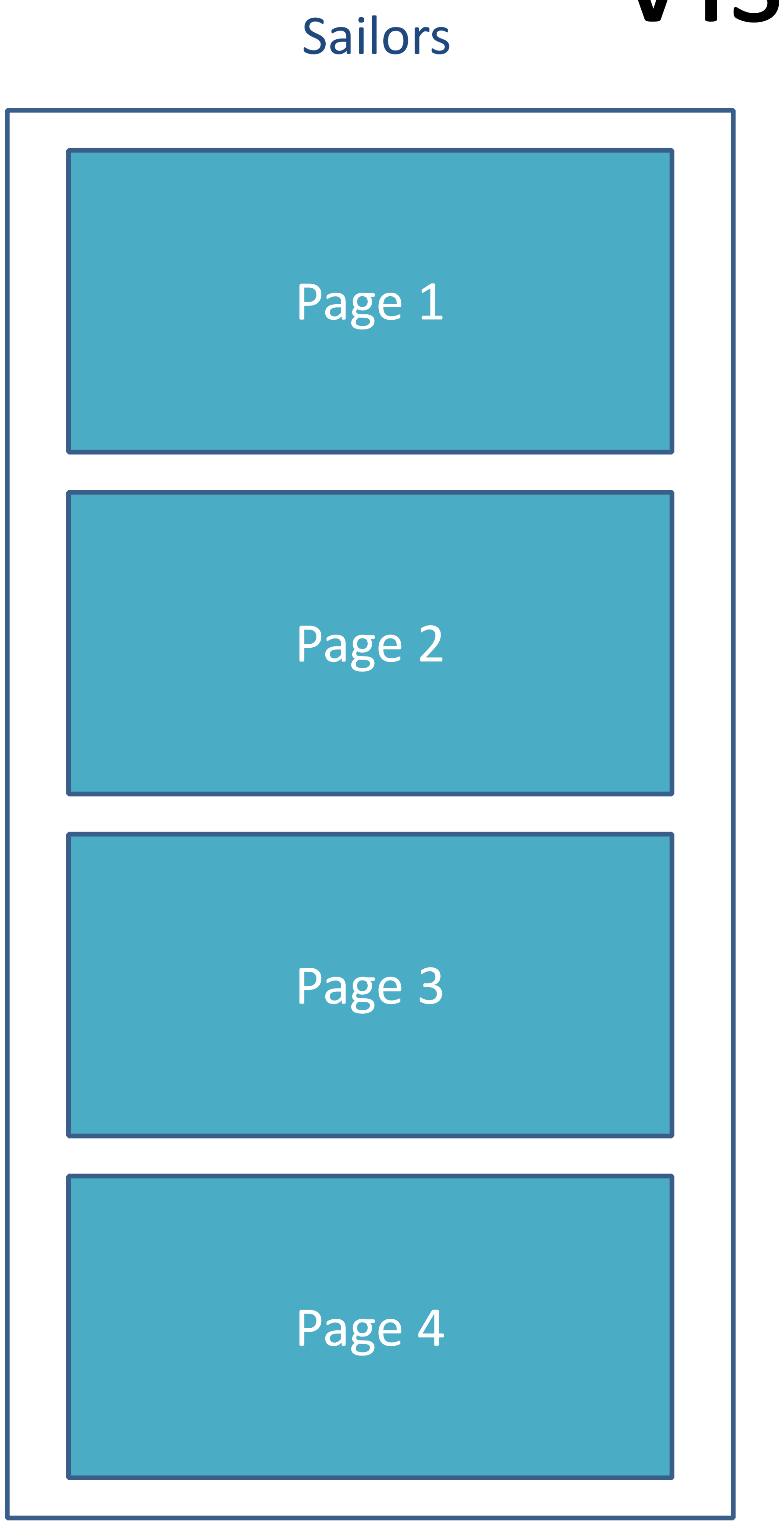
(Slides courtesy of CS186  Fall 2013)

```
SELECT * FROM Sailors S, Reserves R
WHERE S.sid = R.sid;
```

# Visualizations



# Visualizations



Sailors

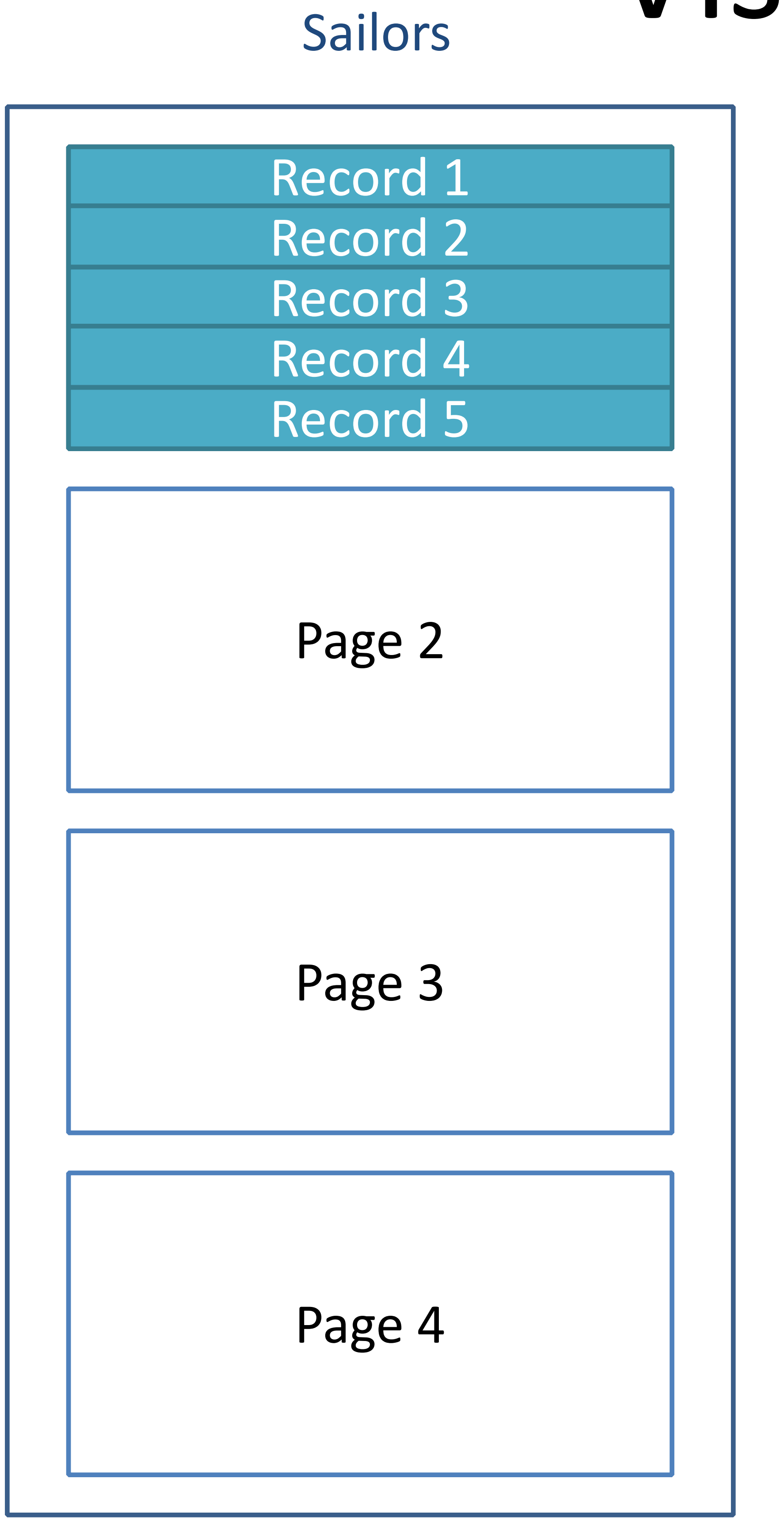
Page 1

Page 2

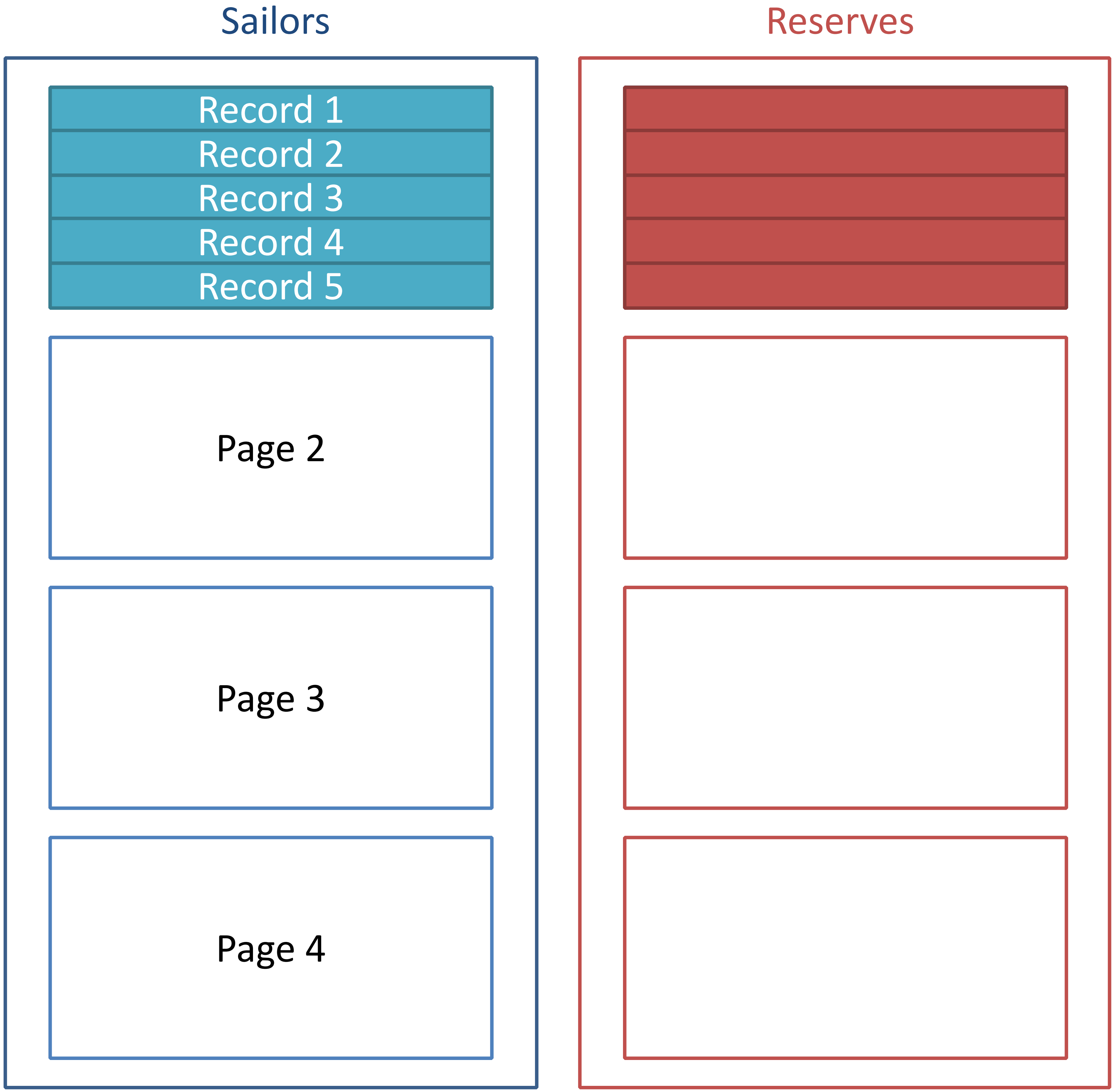
Page 3

Page 4

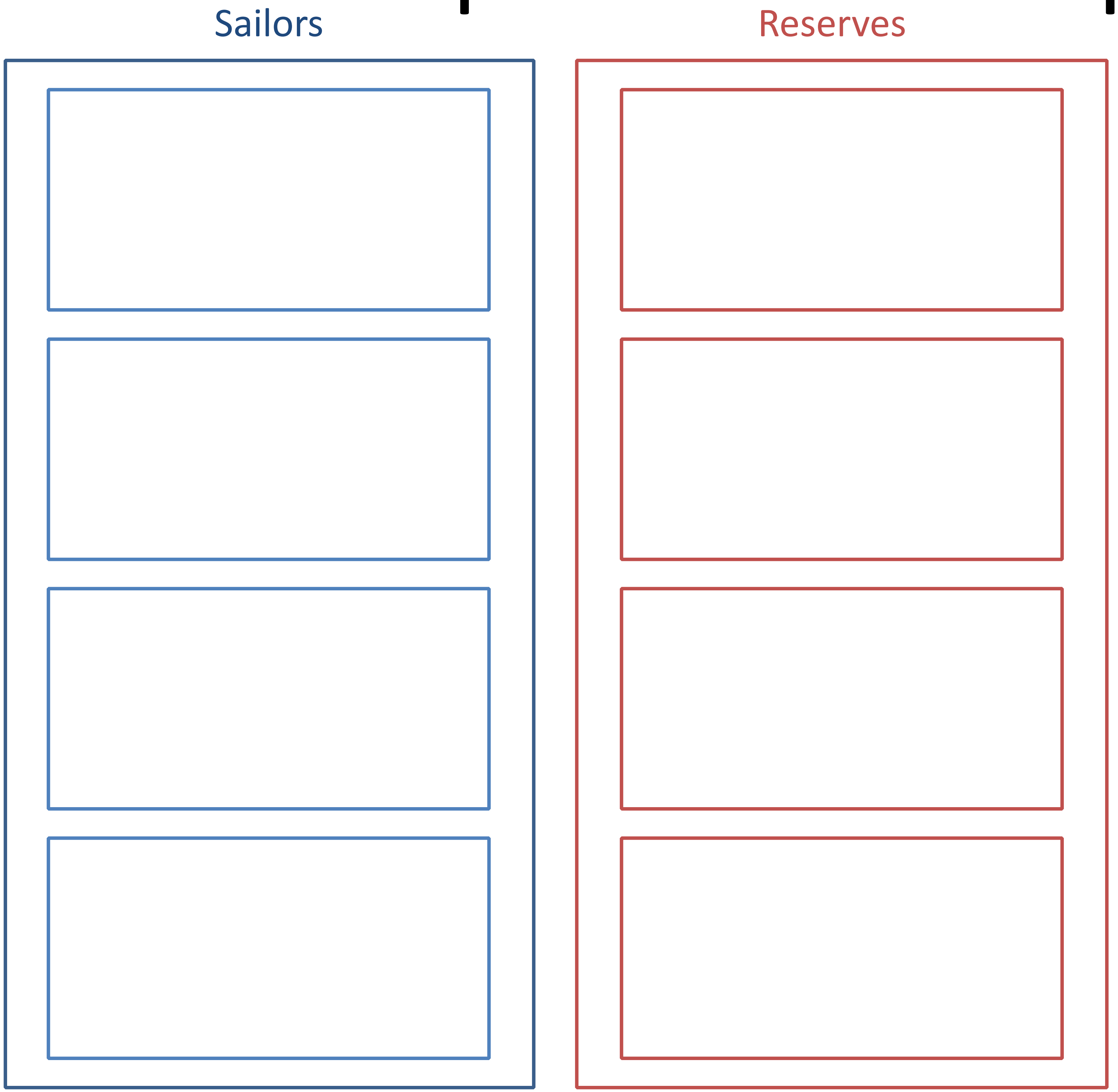
# Visualizations



# Visualizations



# Simple Nested Loops Join



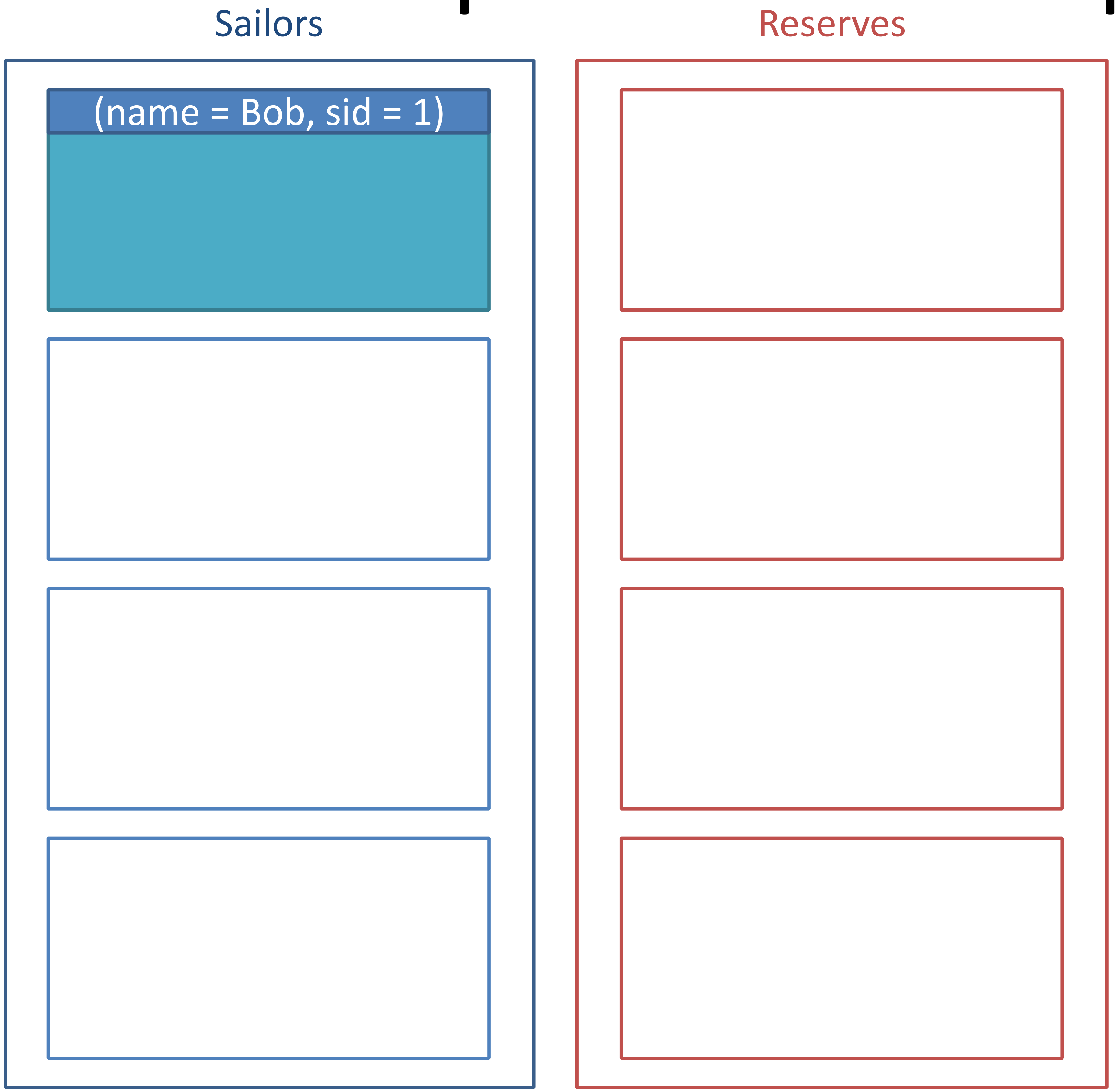
**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

# Simple Nested Loops Join



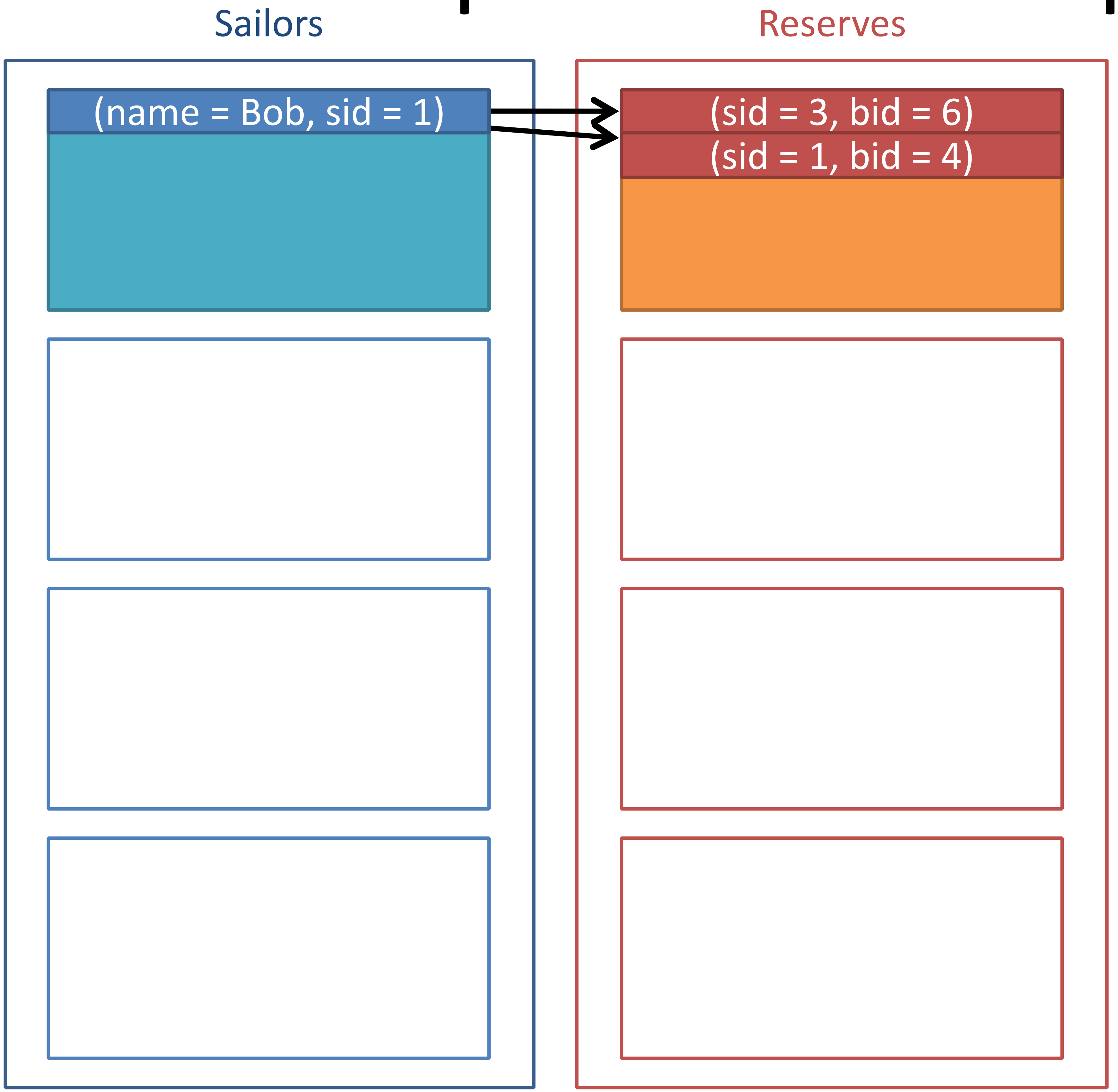
**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

# Simple Nested Loops Join



Sailors

Reserves

(name = Bob, sid = 1)

(sid = 3, bid = 6)

(sid = 1, bid = 4)

**Key idea:**

Take each record of S and match it with each record of R.

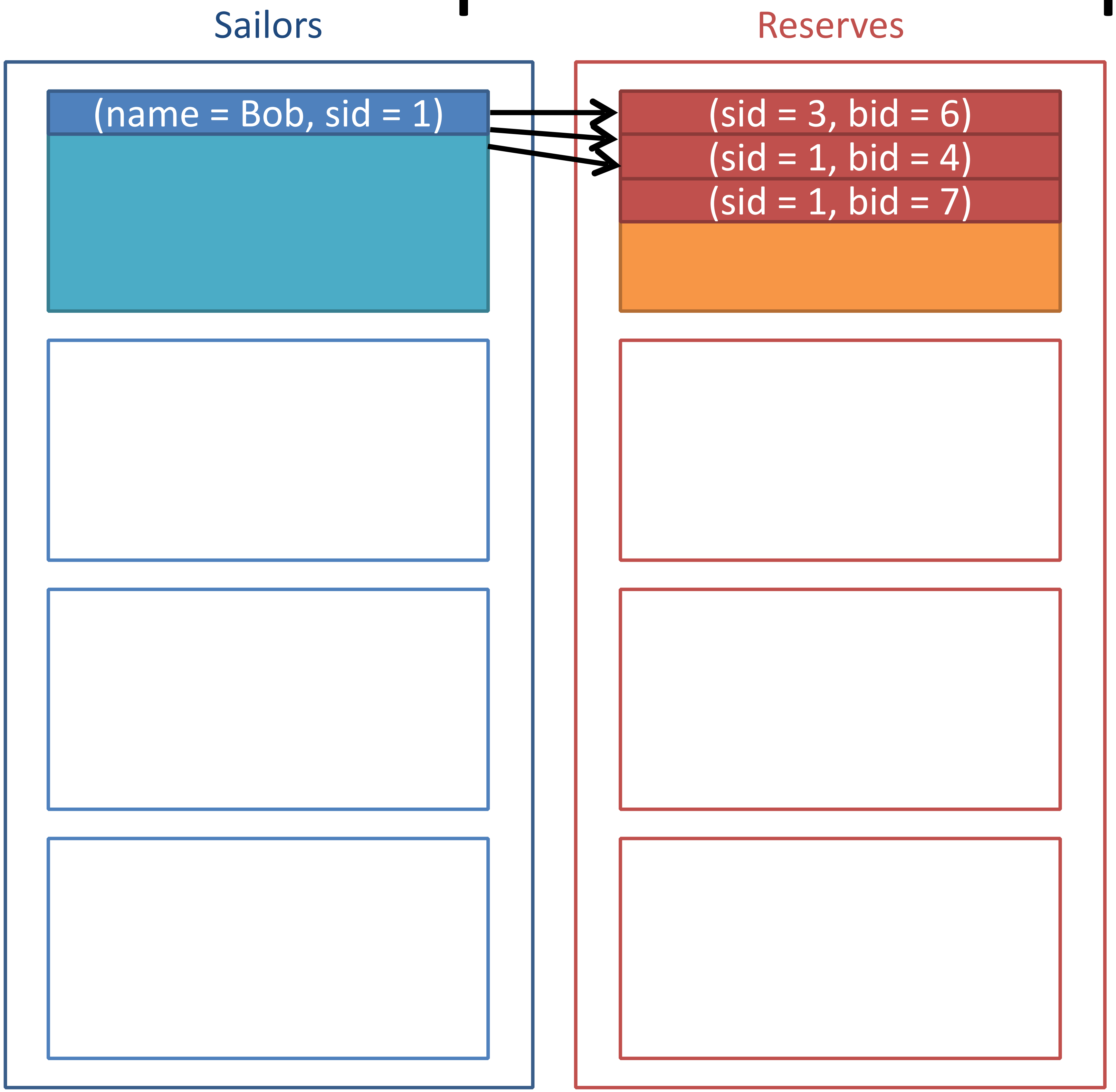
**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)

# Simple Nested Loops Join



Sailors

(name = Bob, sid = 1)

Reserves

(sid = 3, bid = 6)

(sid = 1, bid = 4)

(sid = 1, bid = 7)

**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

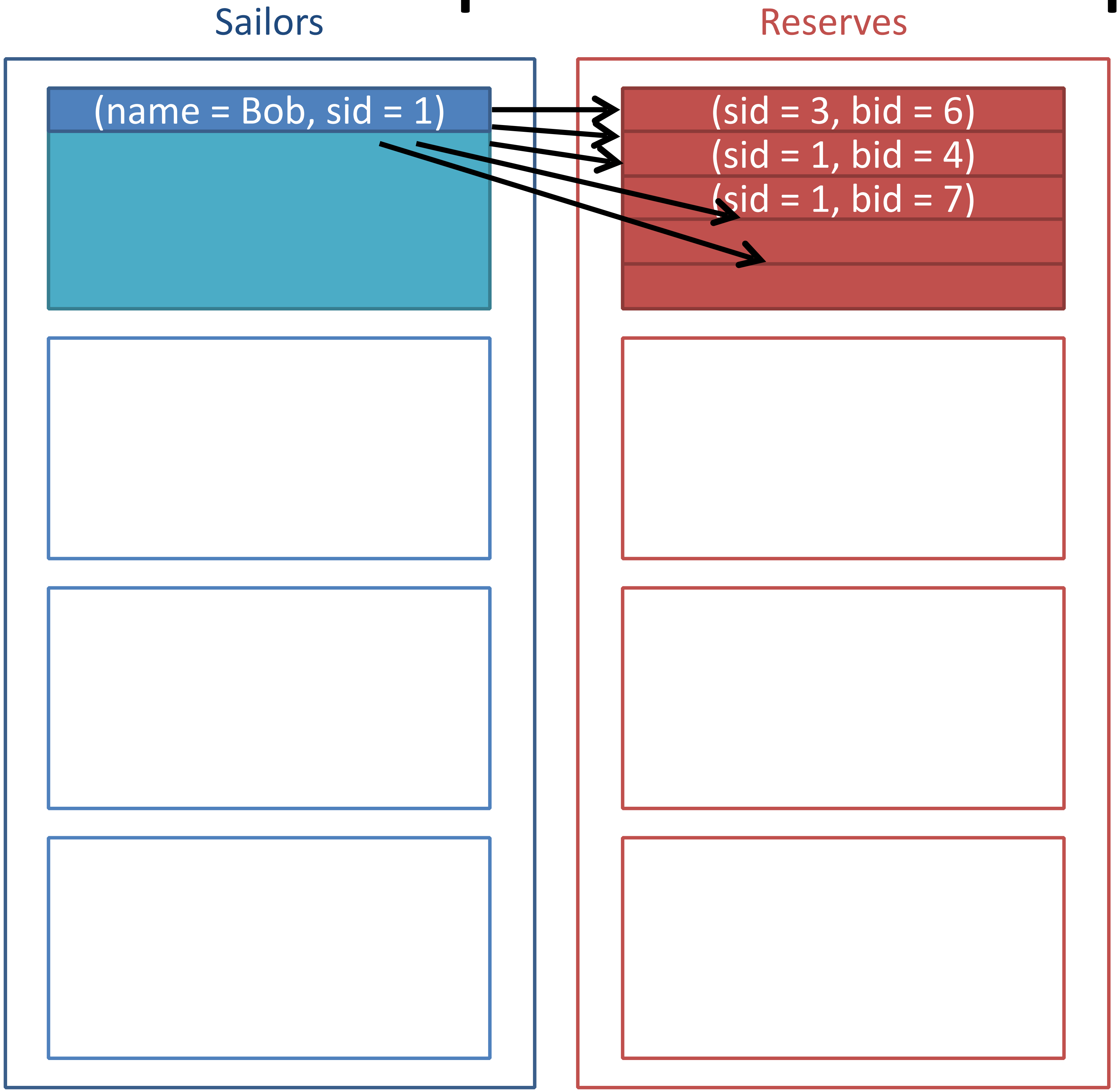
1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



**Key idea:**

Take each record of S and match it with each record of R.

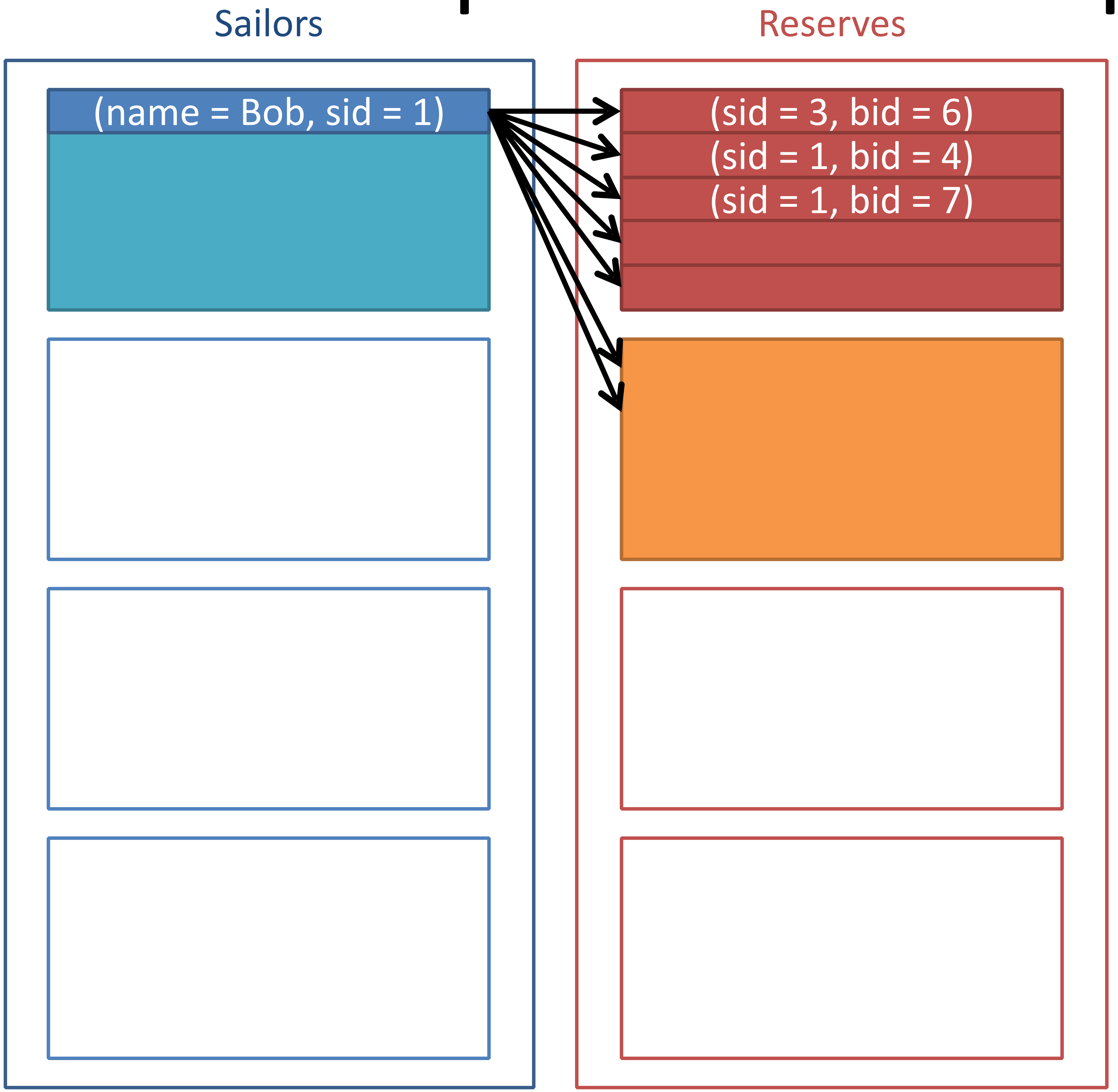
**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



**Key idea:**

Take each record of S and match it with each record of R.

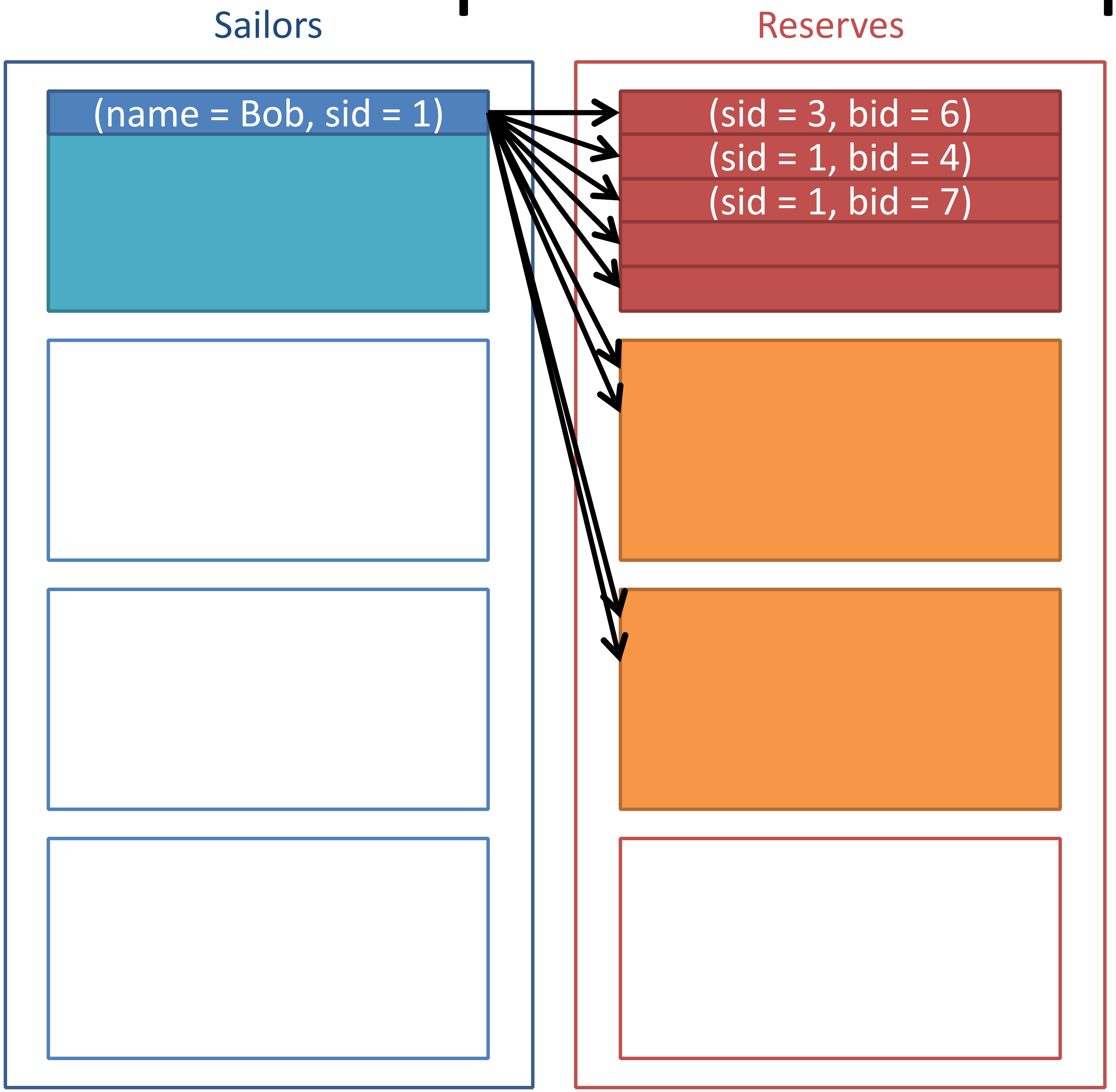
**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



Sailors

(name = Bob, sid = 1)

Reserves

(sid = 3, bid = 6)
(sid = 1, bid = 4)
(sid = 1, bid = 7)

**Key idea:**

Take each record of S and match it with each record of R.

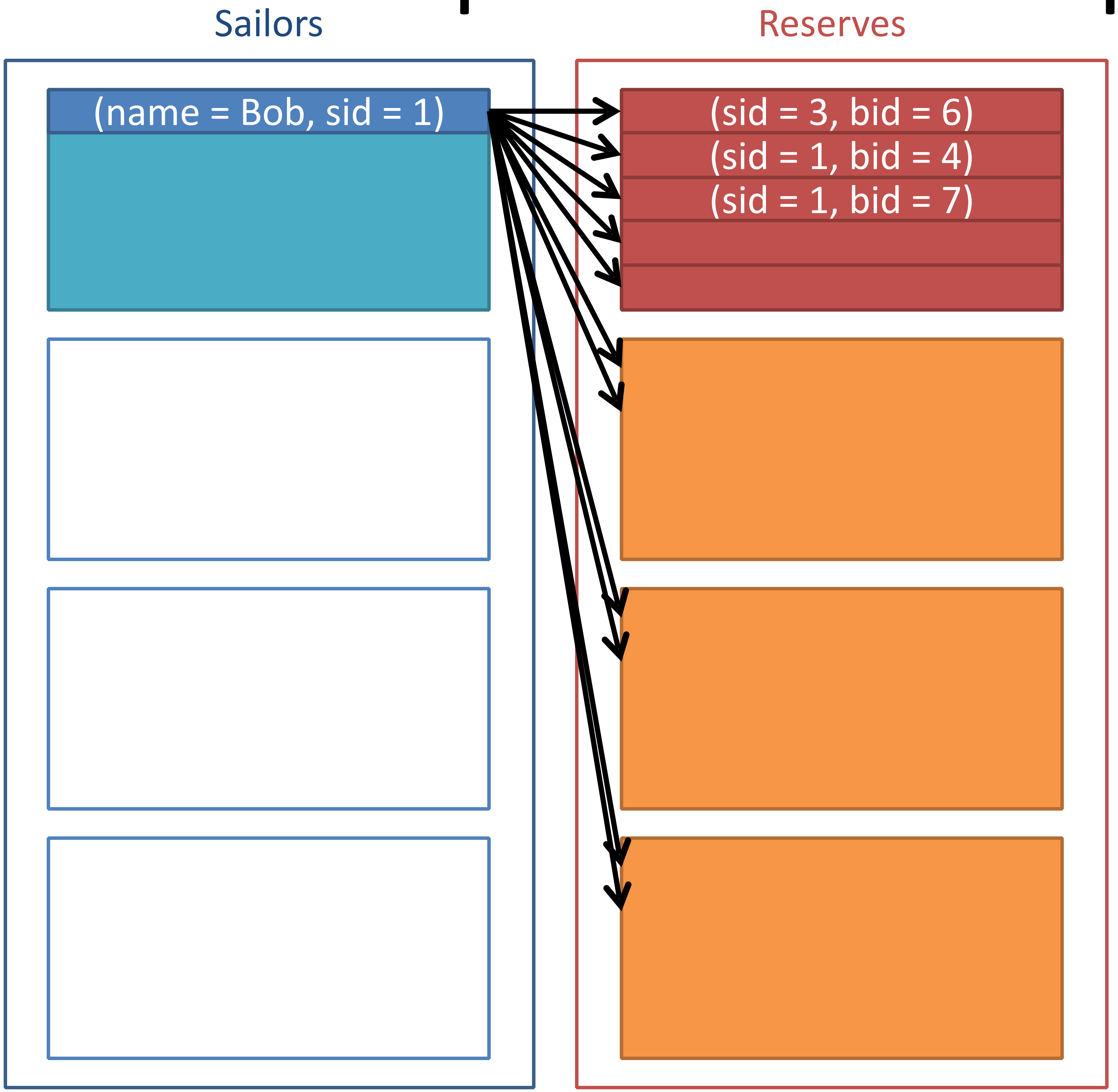
**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



Sailors

(name = Bob, sid = 1)

Reserves

(sid = 3, bid = 6)

(sid = 1, bid = 4)

(sid = 1, bid = 7)

**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

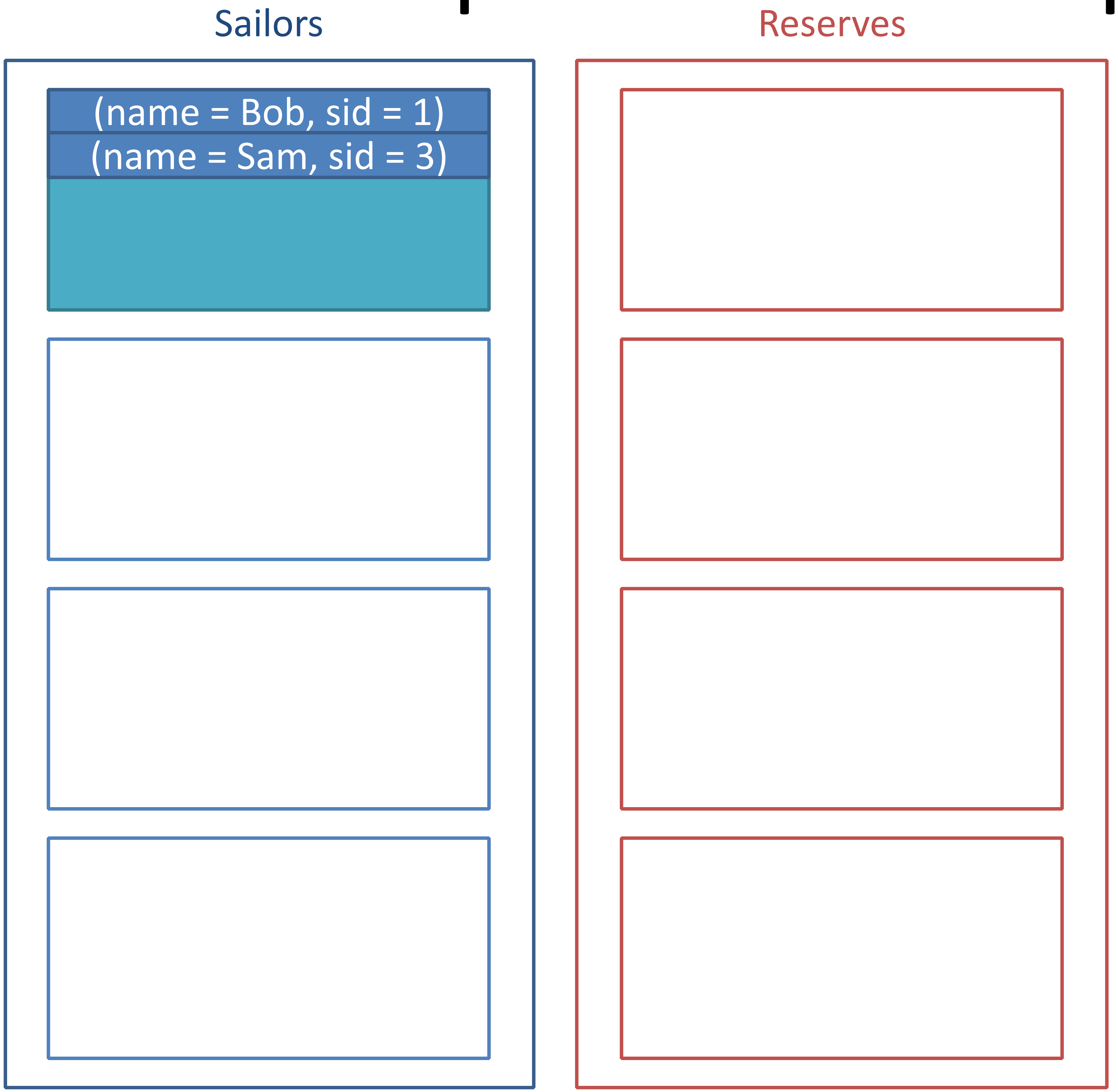
1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



Sailors

(name = Bob, sid = 1)

(name = Sam, sid = 3)

Reserves

**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

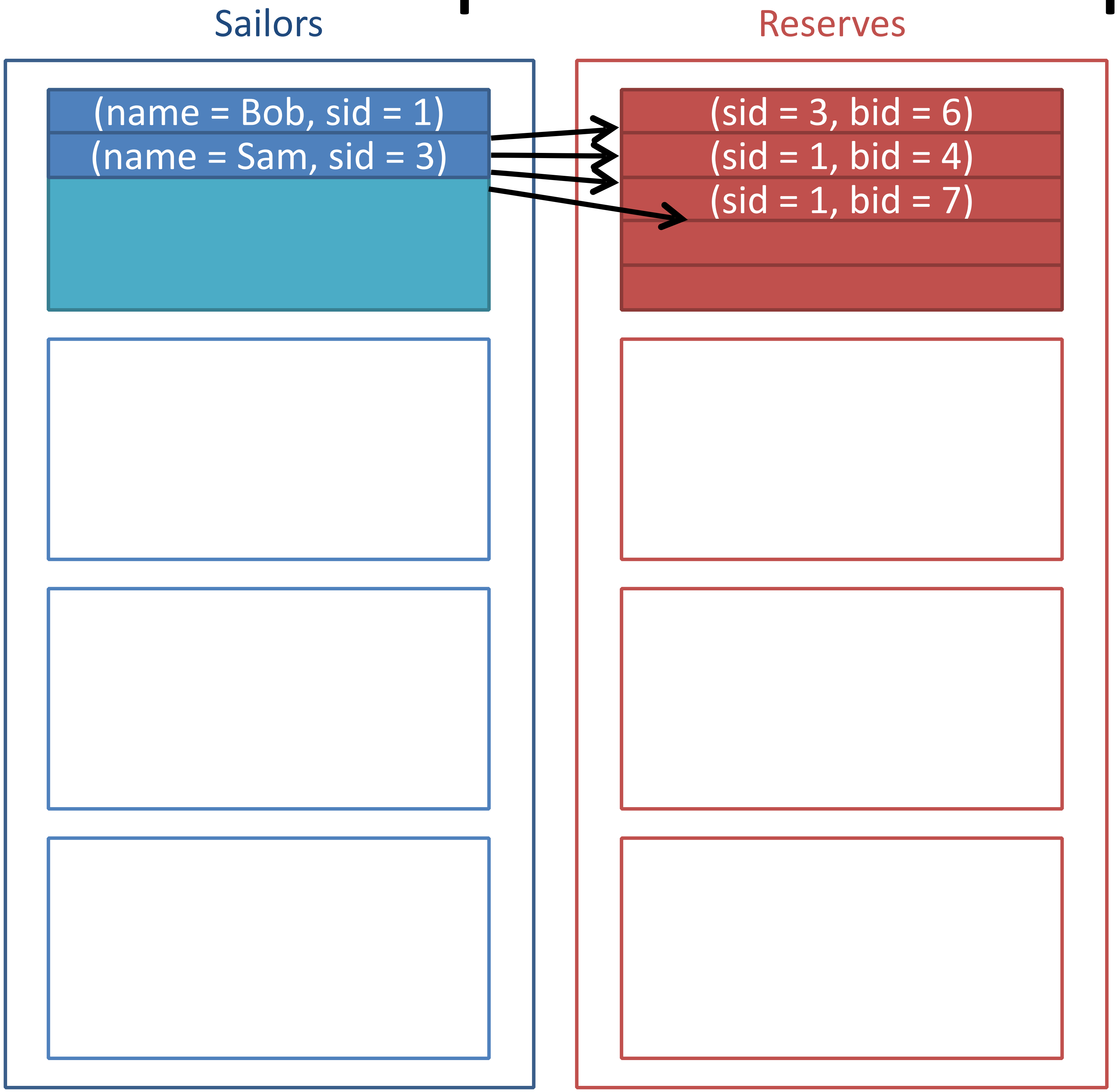
1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

# Simple Nested Loops Join



Sailors

(name = Bob, sid = 1)

(name = Sam, sid = 3)

Reserves

(sid = 3, bid = 6)

(sid = 1, bid = 4)

(sid = 1, bid = 7)

**Key idea:**

Take each record of S and match it with each record of R.

**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**Output:**

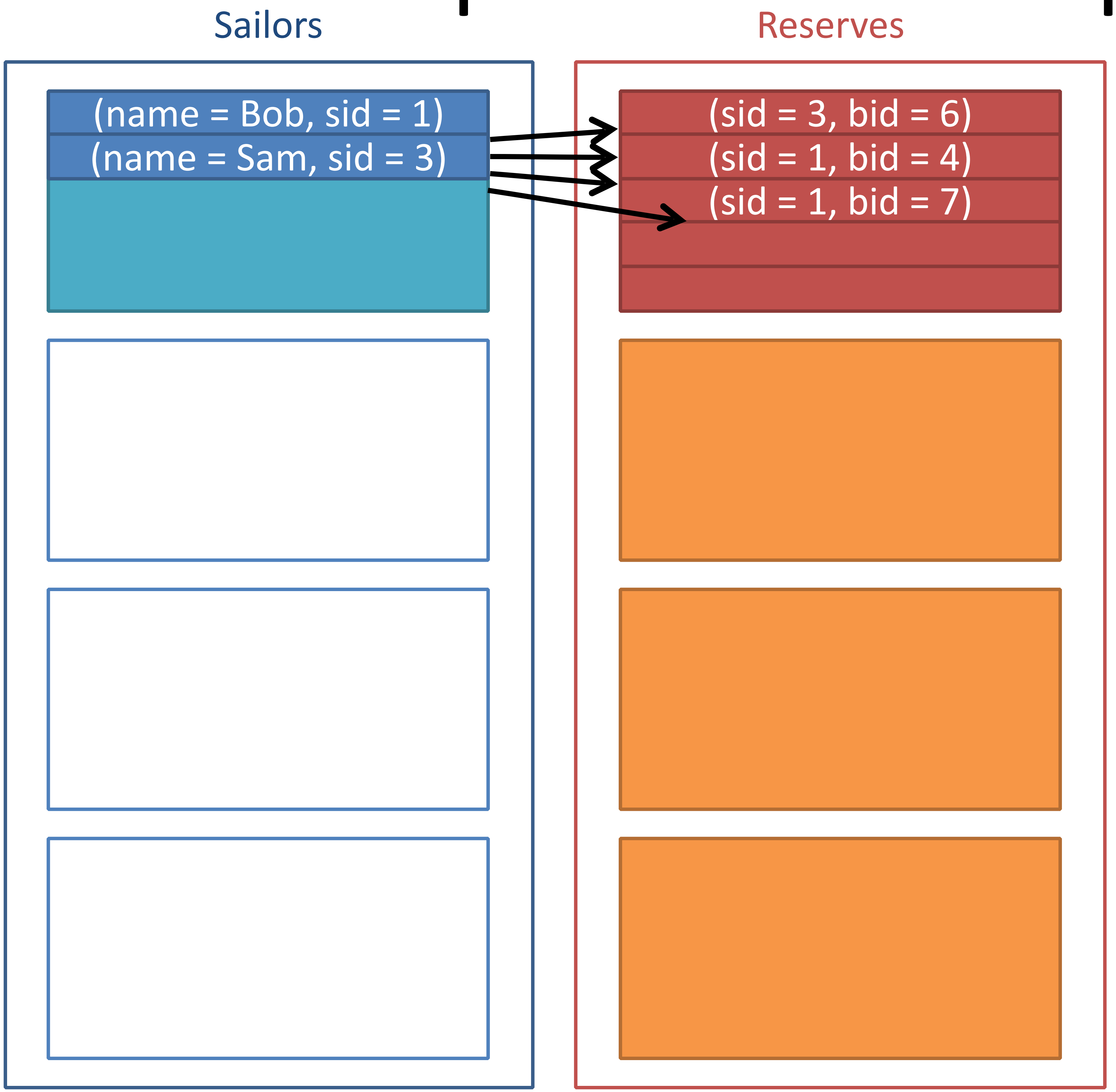
(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

(name = Sam, sid = 3, bid = 6)

**Notation: [S] == "# pages in S" ; |S| == "# tuples in S"**

# Simple Nested Loops Join



**Key idea:**

Take each record of S and match it with each record of R.

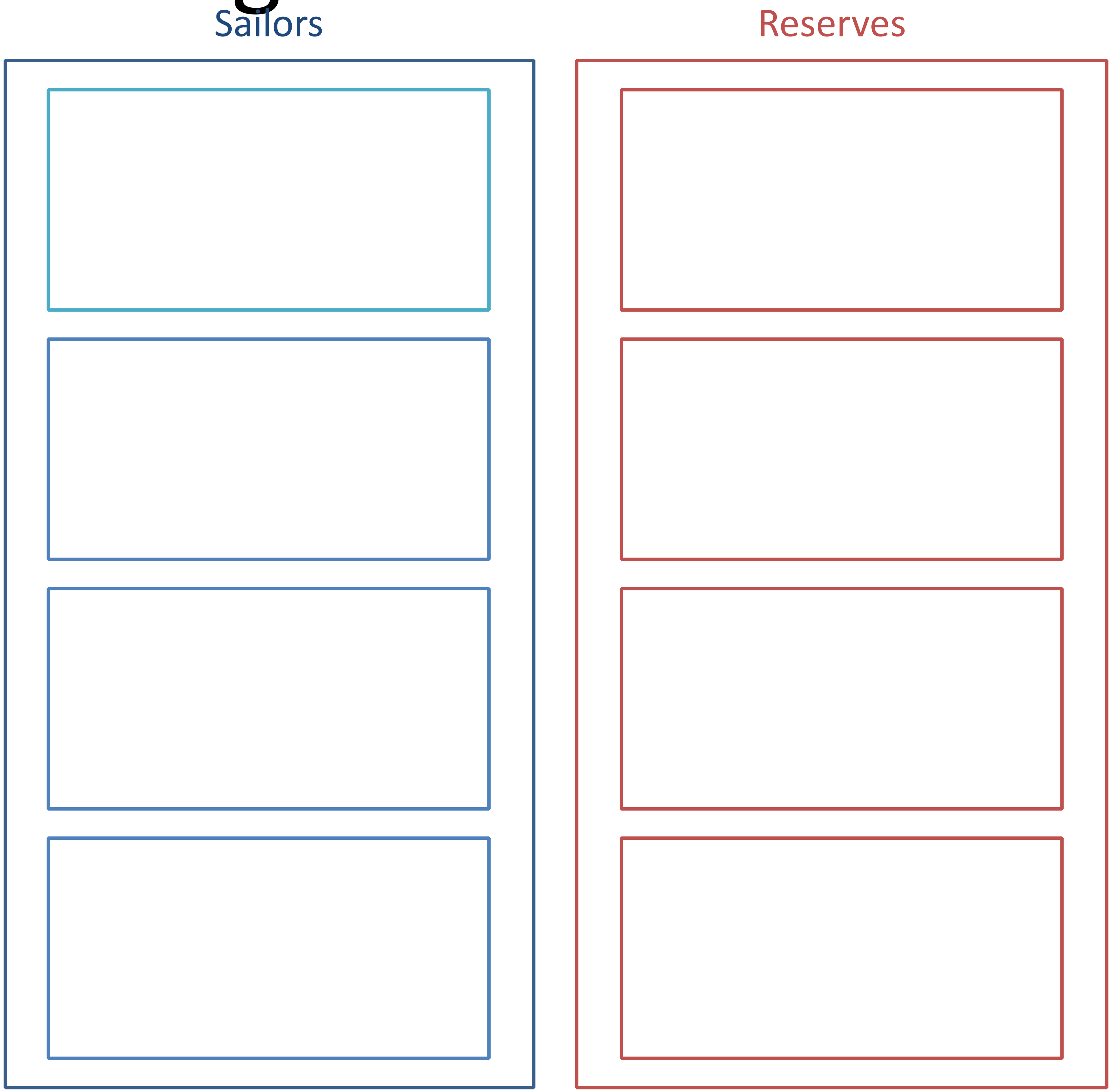
**Steps:**

1. Get tuple of S.
2. Iterate through each tuple in R.

**I/Os:**

[S] + |S|*[R]

# Page-Oriented Nested Loops Join



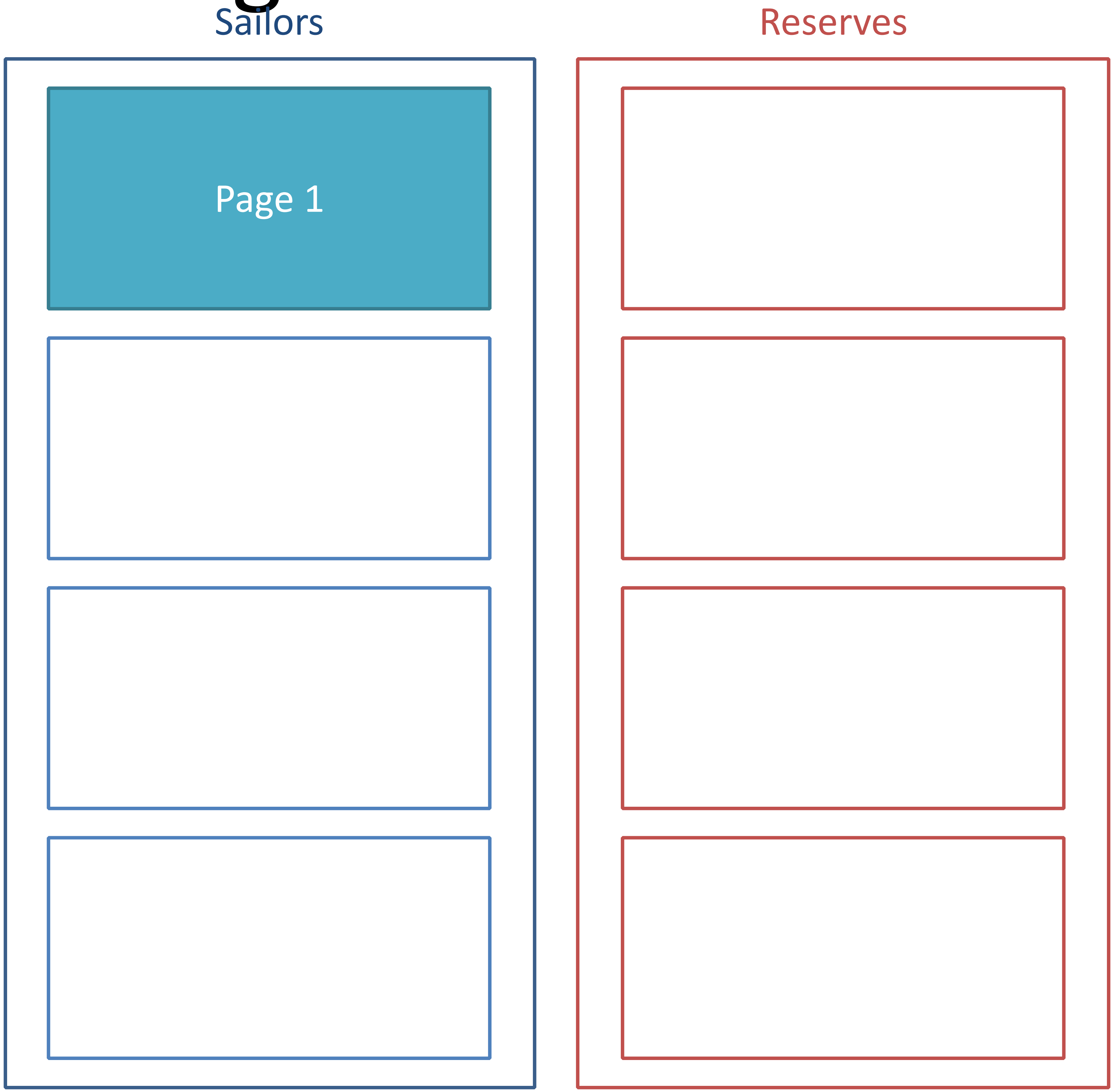
**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Page-Oriented Nested Loops Join



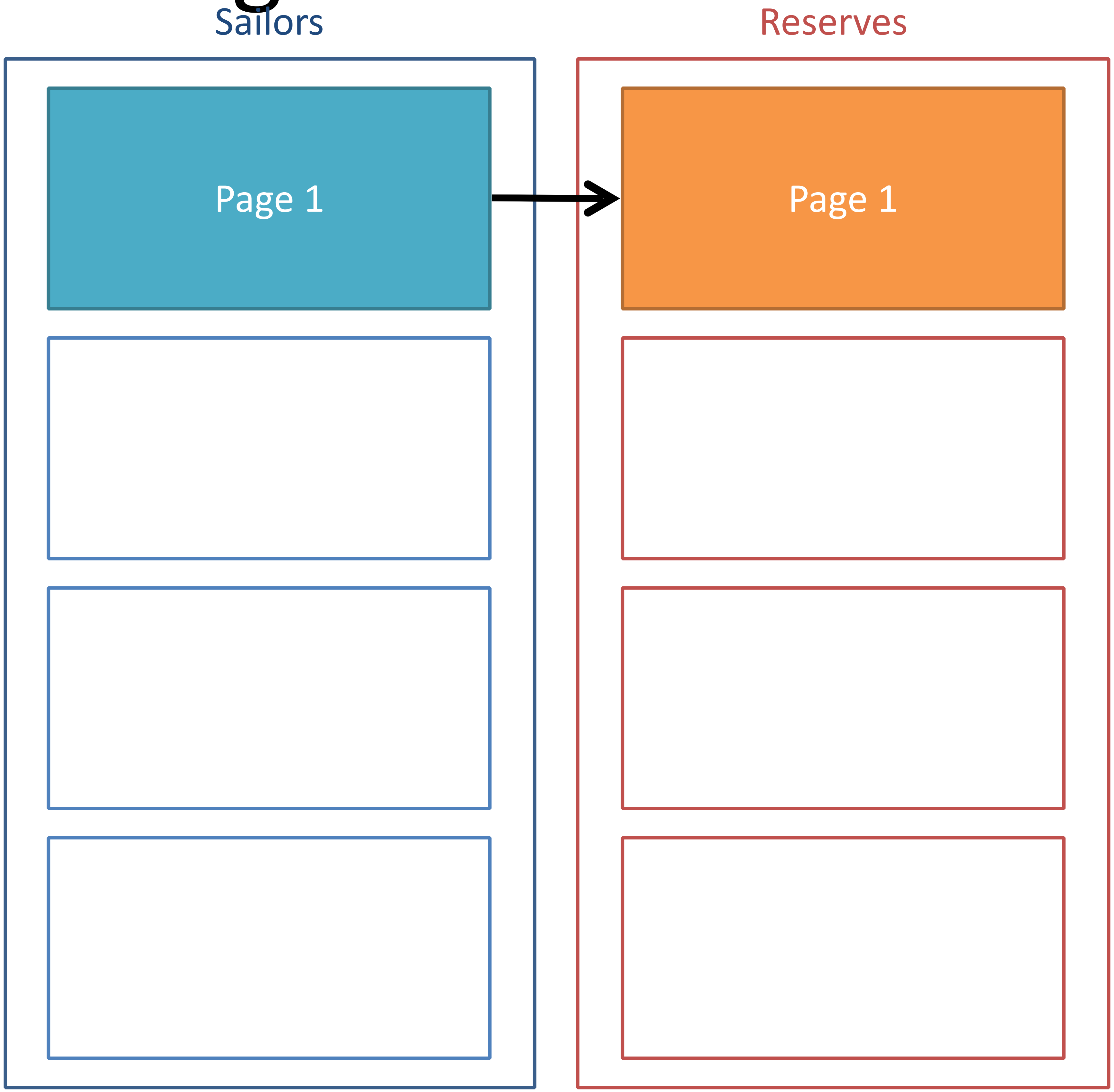
**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Page-Oriented Nested Loops Join



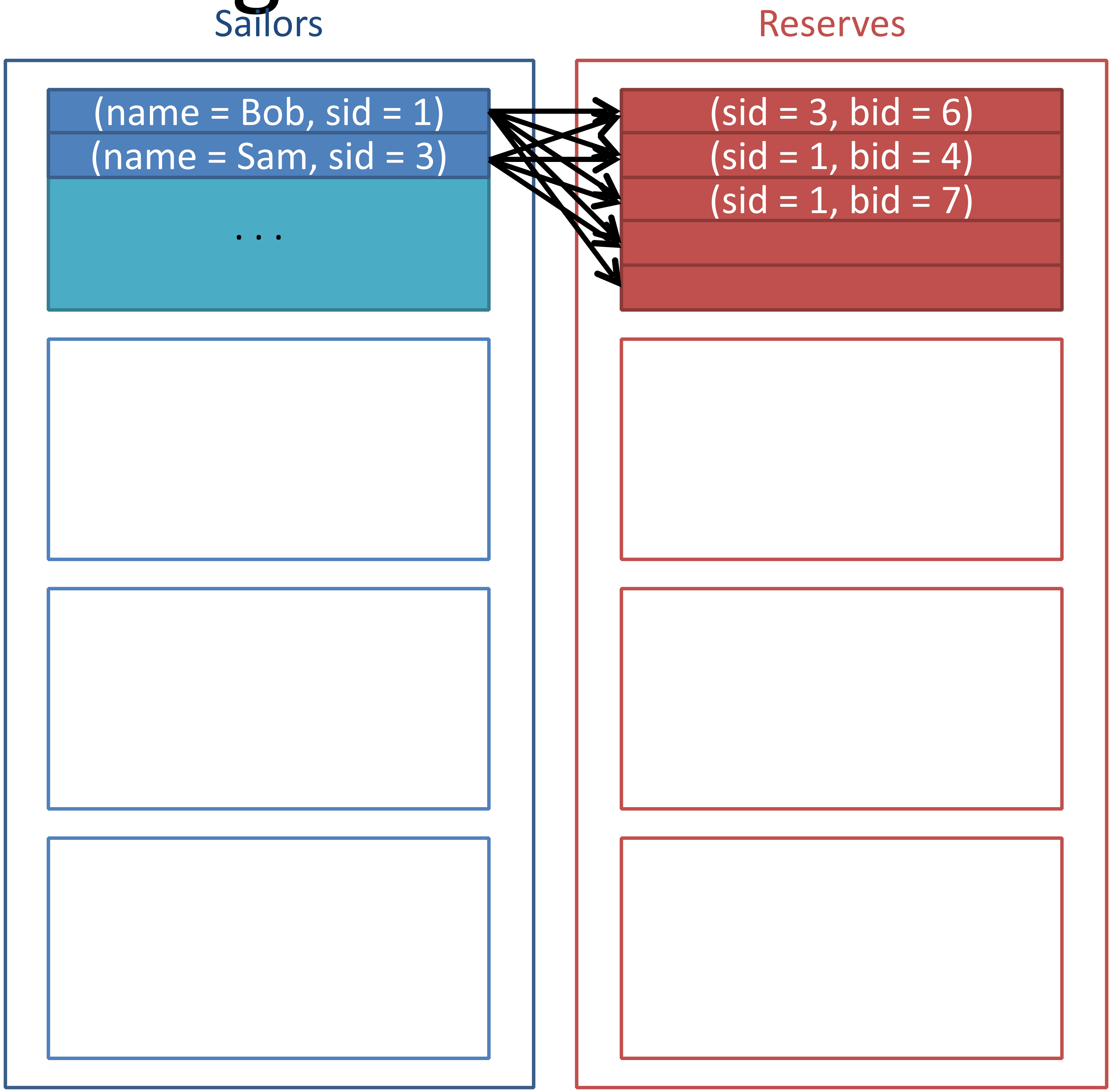
**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Page-Oriented Nested Loops Join



Sailors

(name = Bob, sid = 1)

(name = Sam, sid = 3)

. . .

Reserves

(sid = 3, bid = 6)

(sid = 1, bid = 4)

(sid = 1, bid = 7)

**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

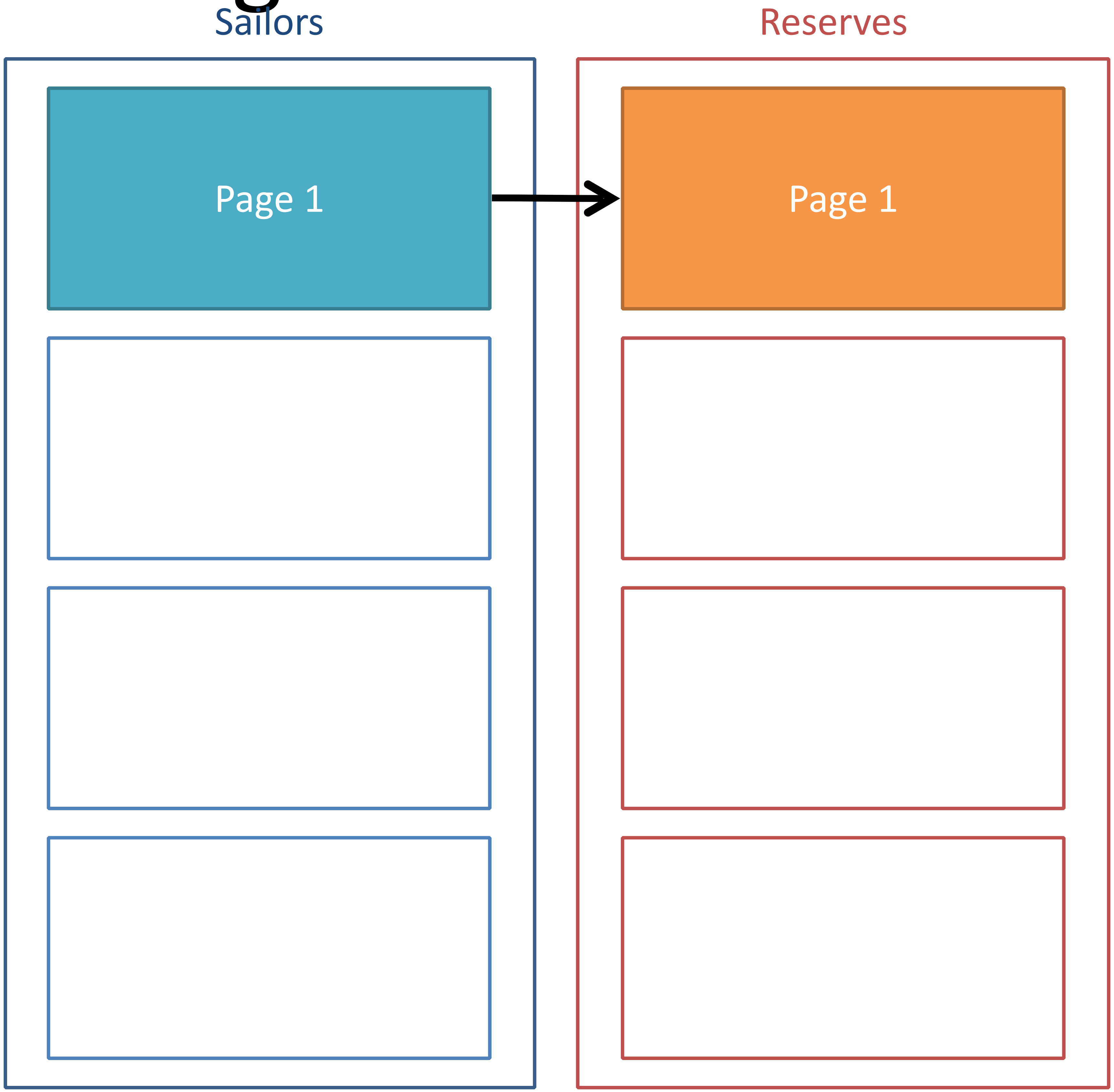
**Output:**

(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

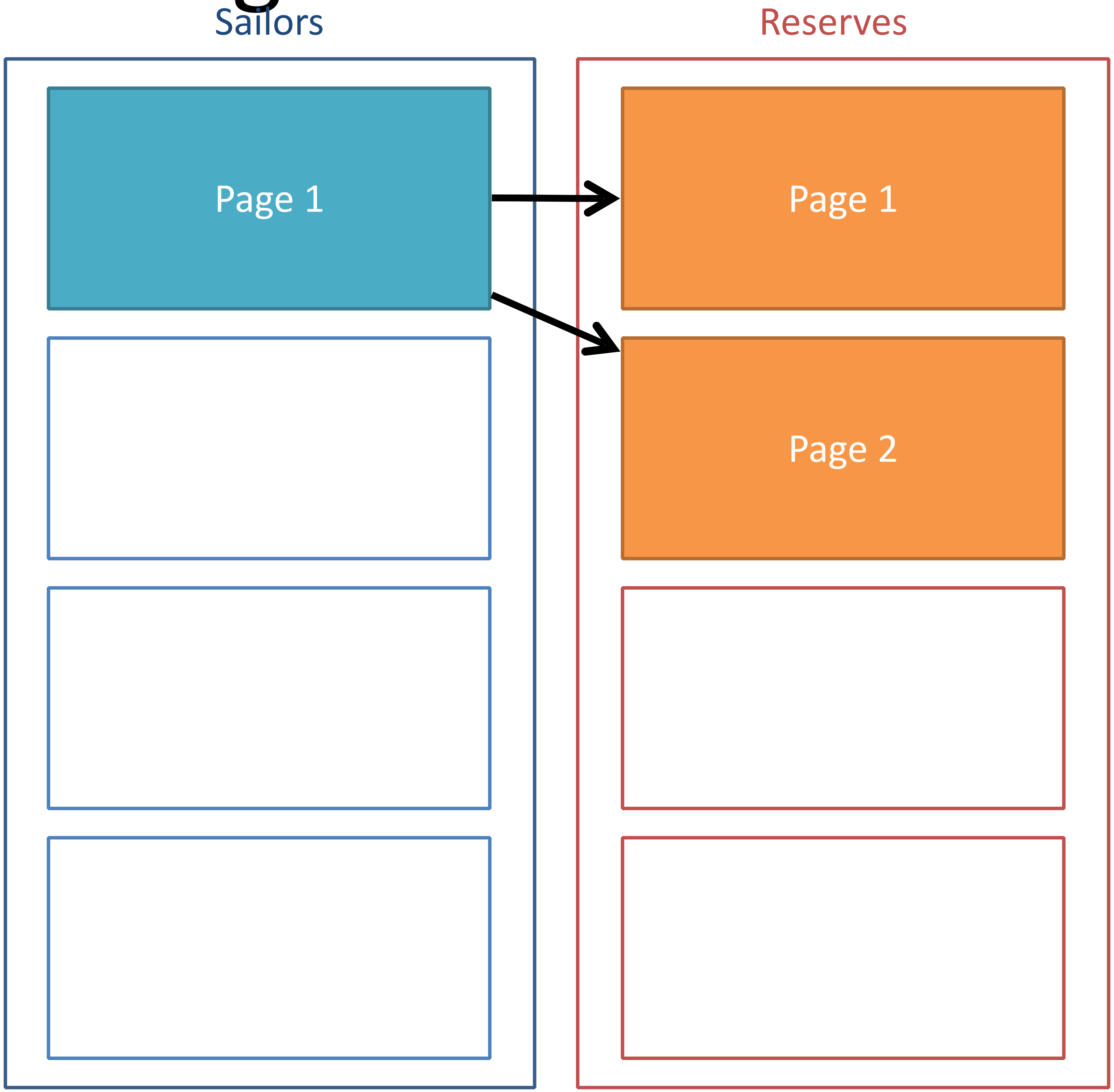
**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

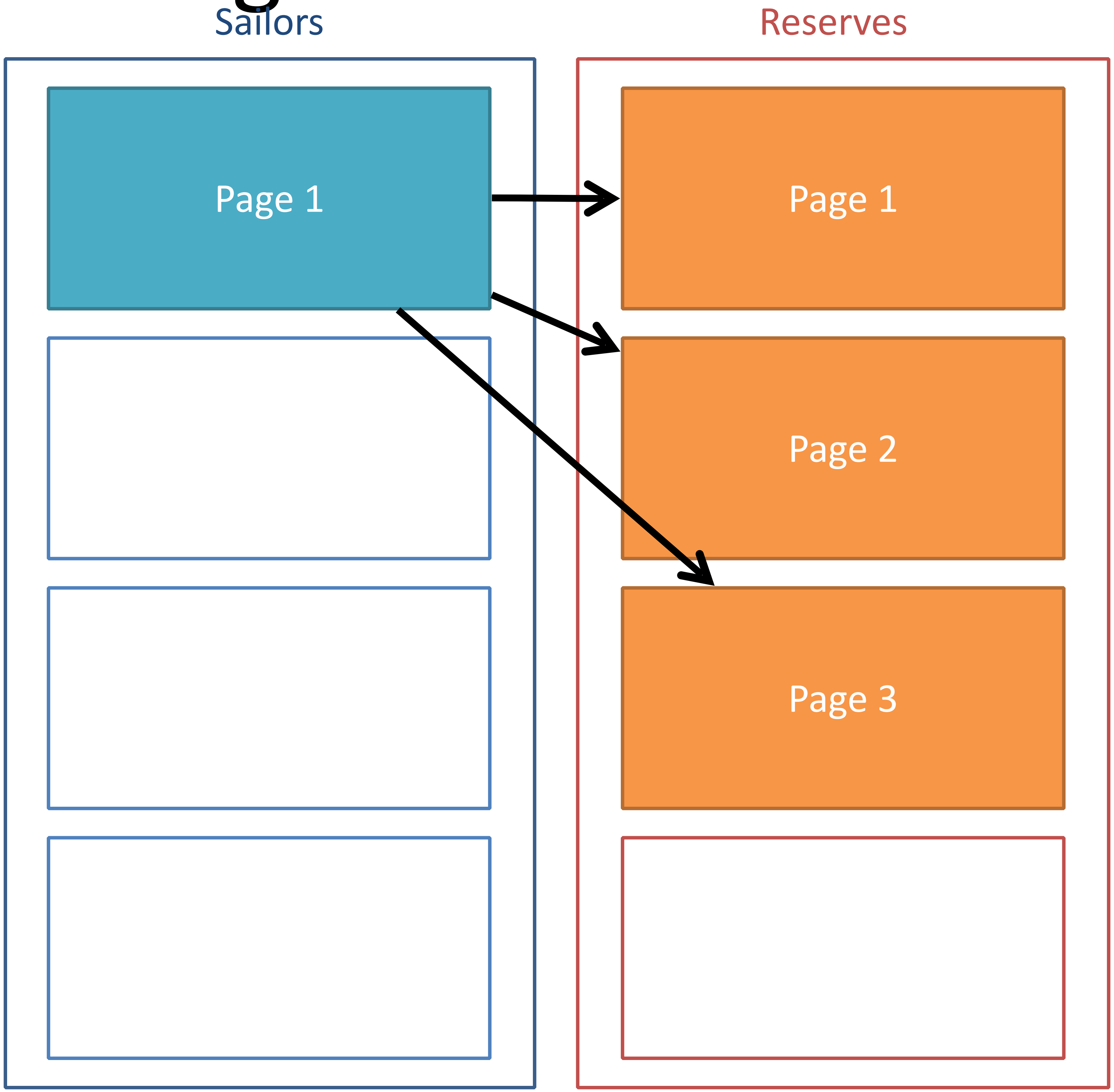
**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

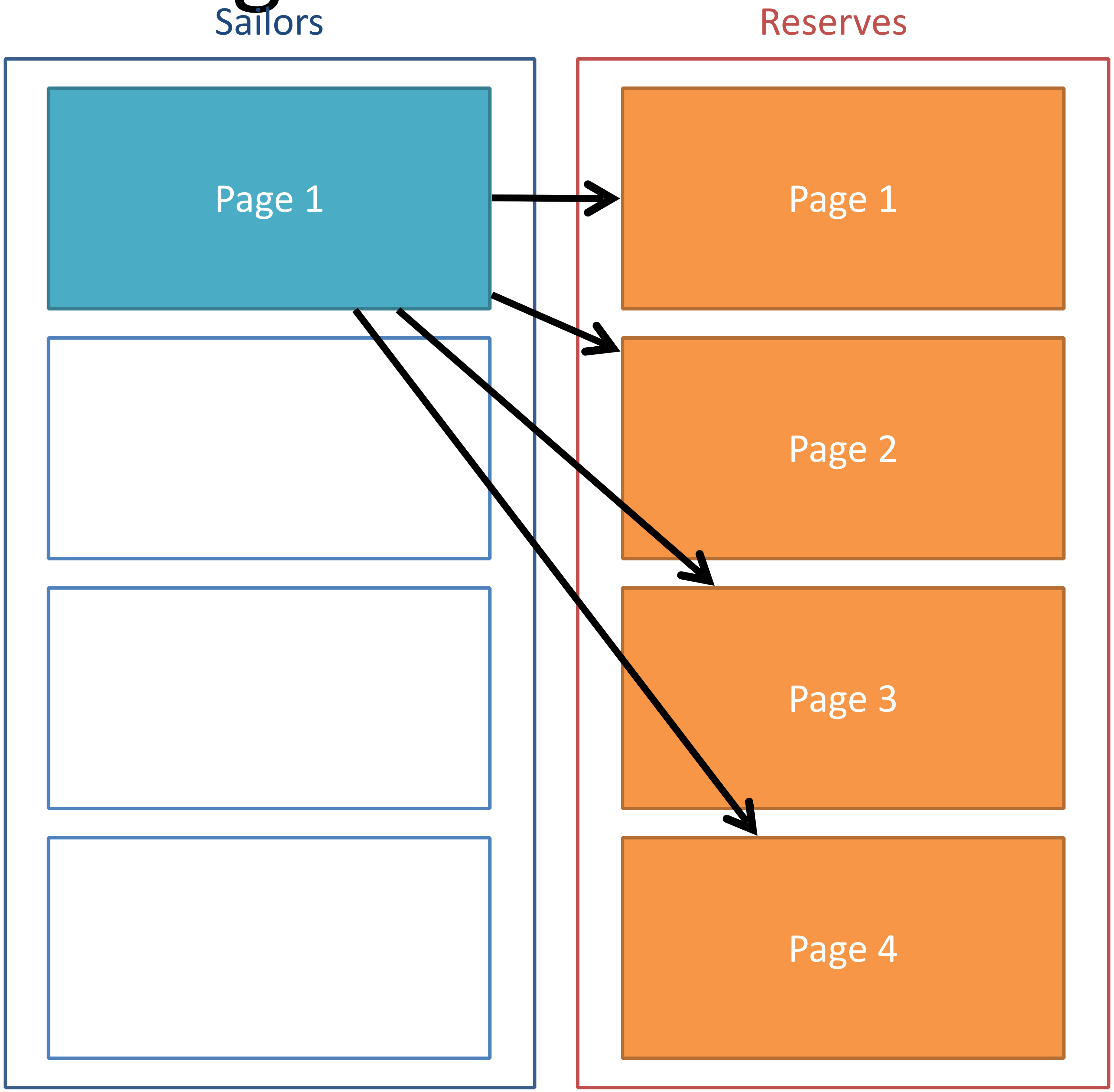
**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

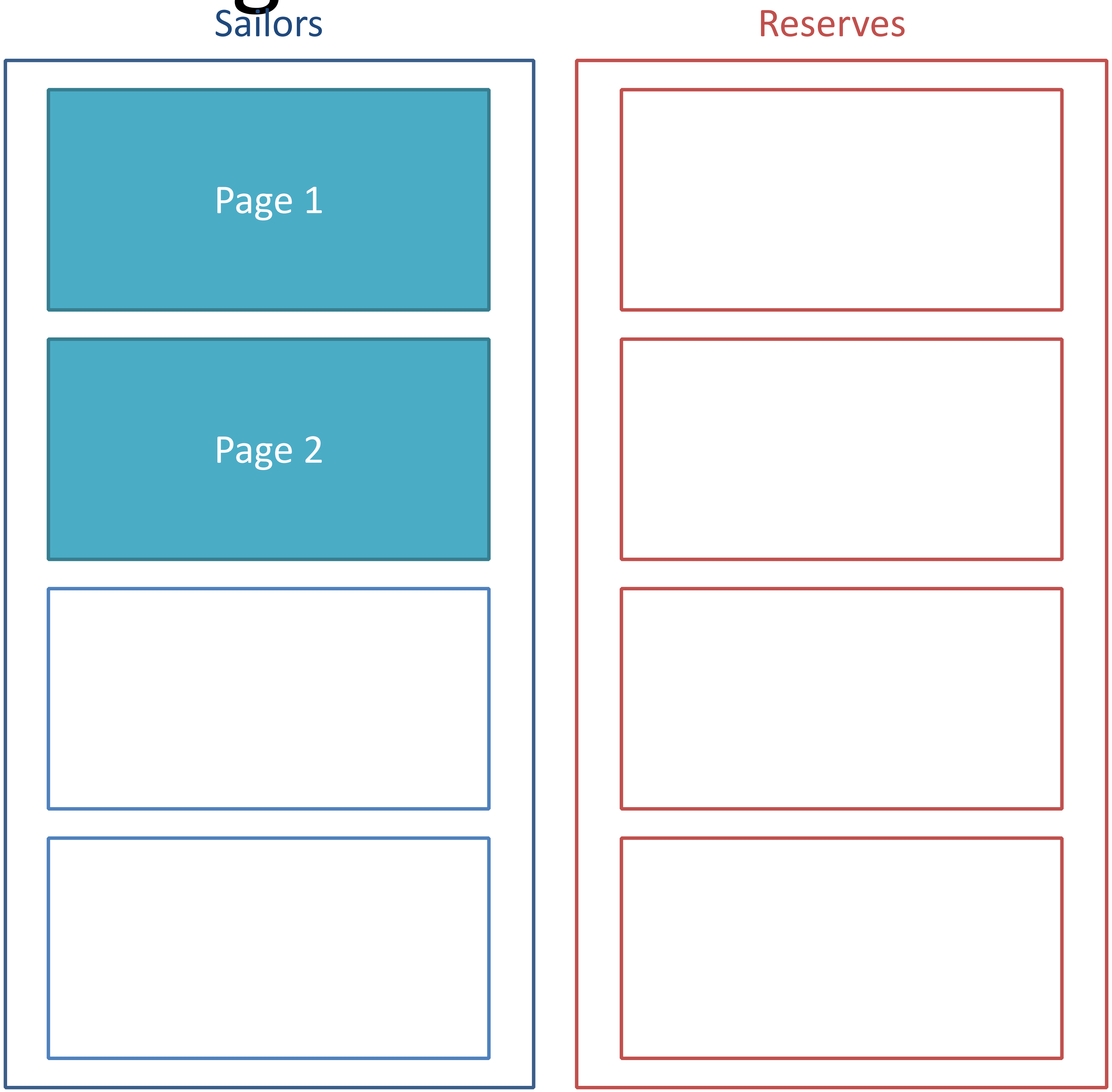
**Output:**

(name = Bob, sid = 1, bid = 4)

(name = Bob, sid = 1, bid = 7)

(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

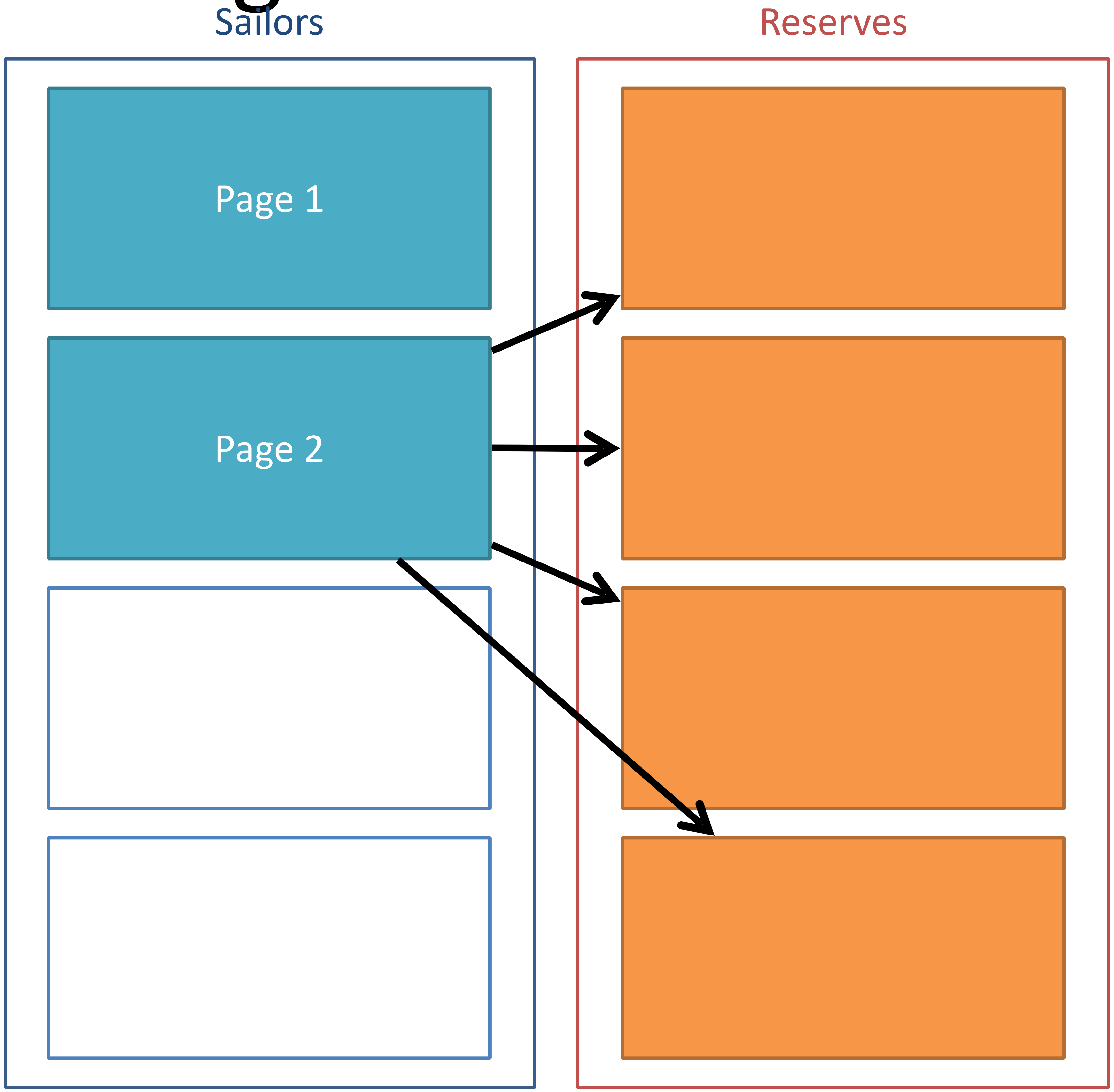
**Steps:**

1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

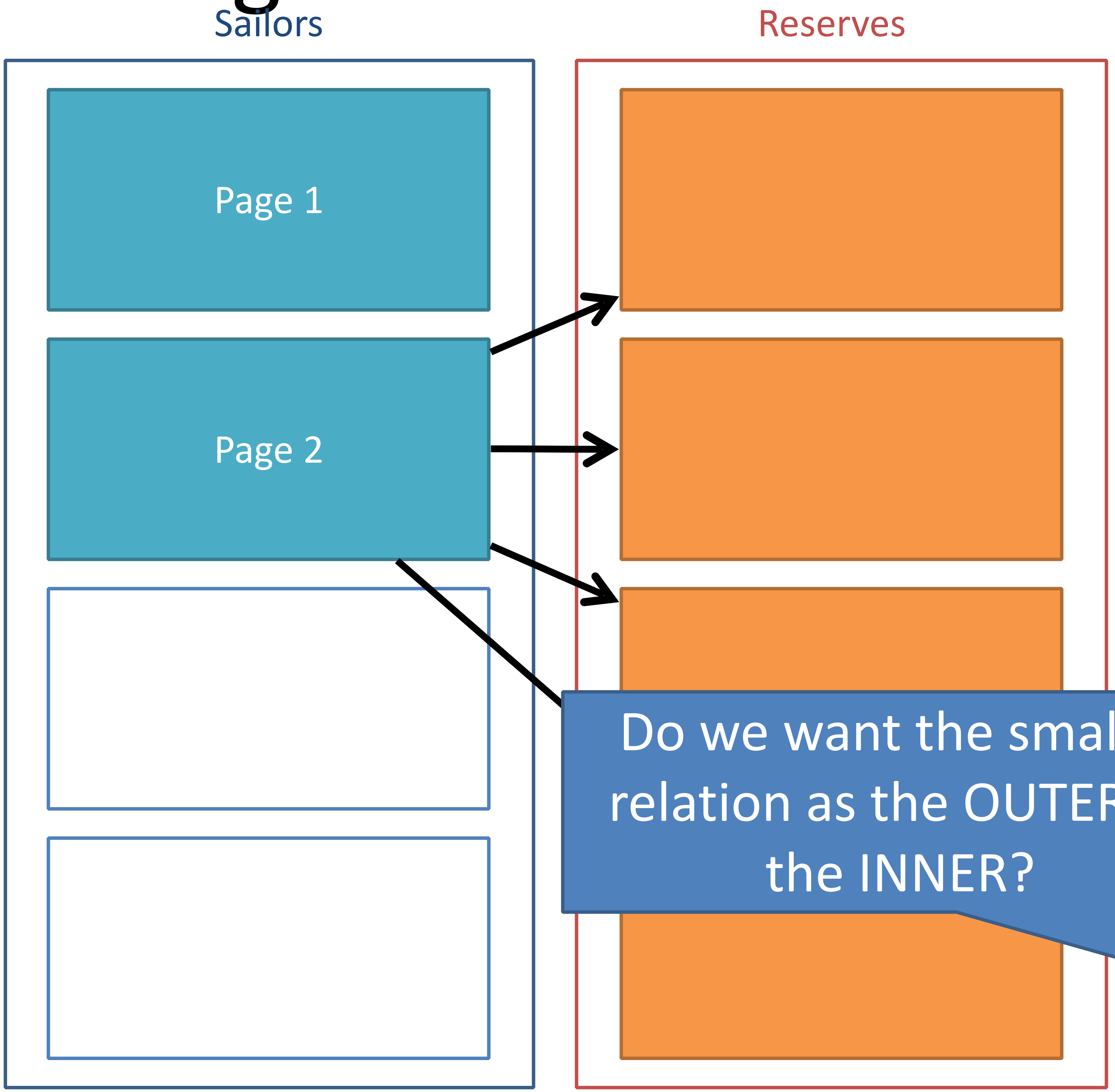
1. Get page of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

**Notation: [S] == “# pages in S” ; |S| == “# tuples in S”**

# Page-Oriented Nested Loops Join



**Key idea:**

Take each page of S and match with each page of R.

**Steps:**

1. Get page of S.
2. Iterate through each page in R.

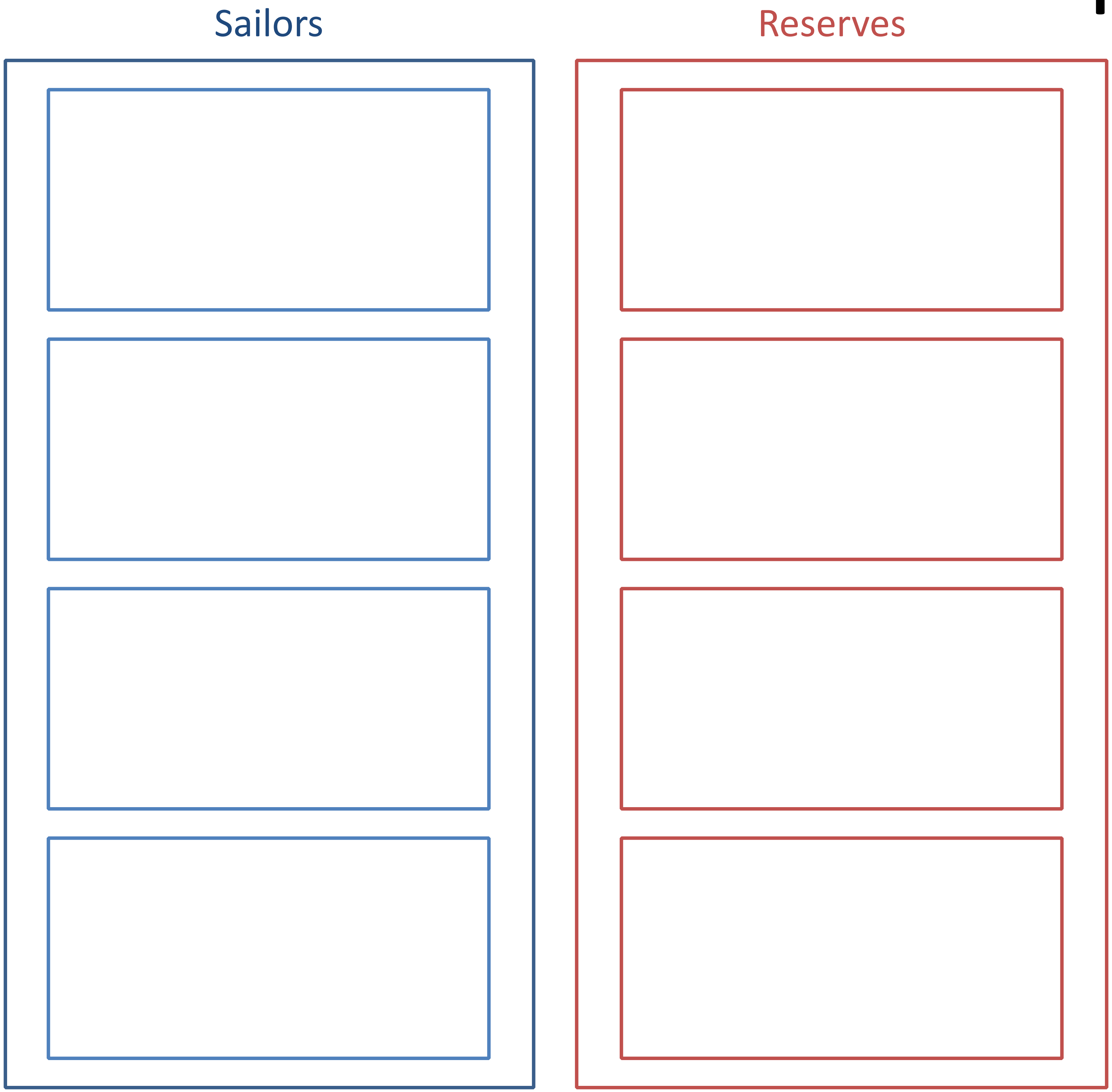
ompare tuples in ach.

Do we want the smaller relation as the OUTER or the INNER?

:

[S] + [S]*[R]

# Chunk Nested Loops Join



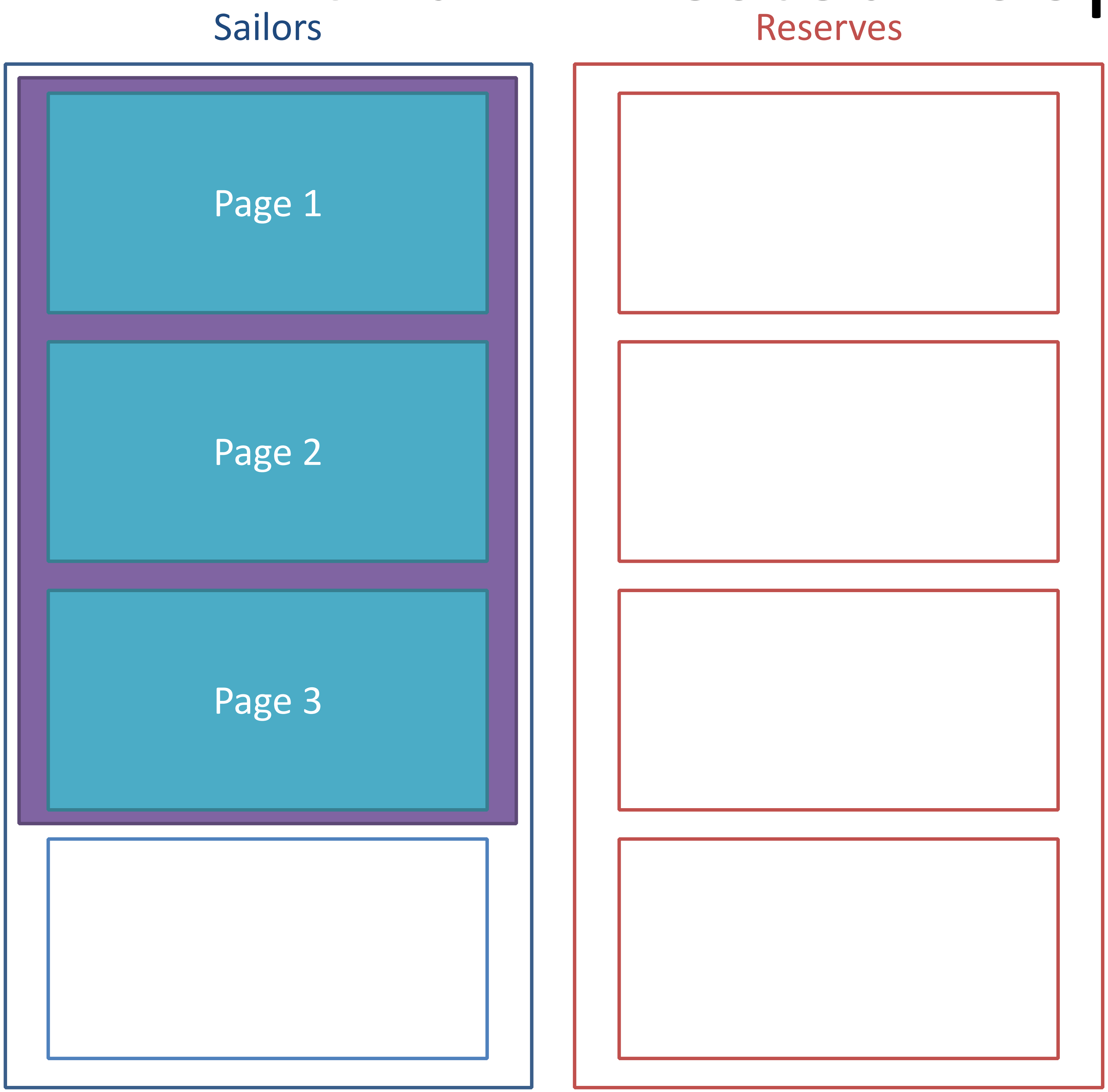
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



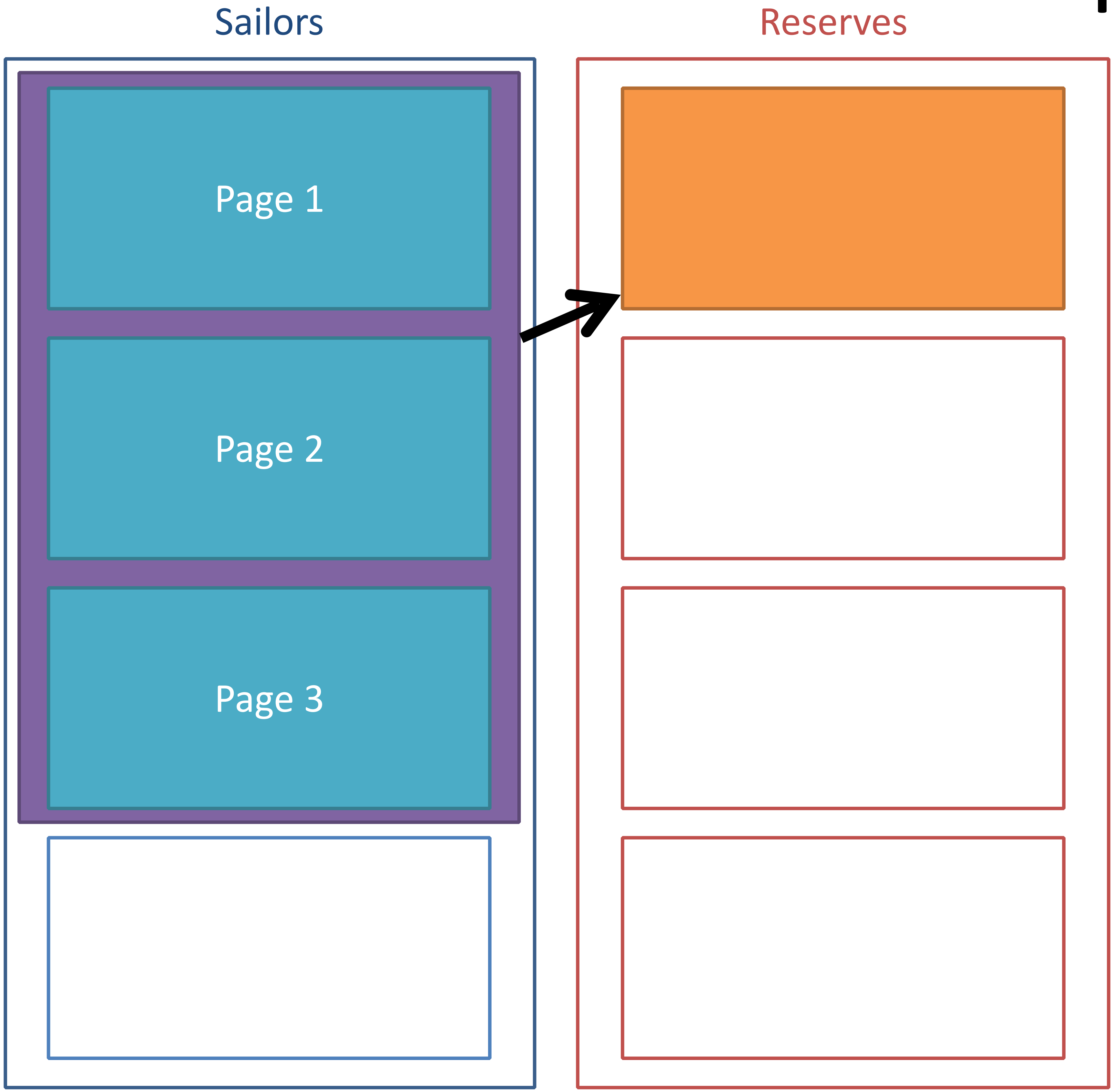
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



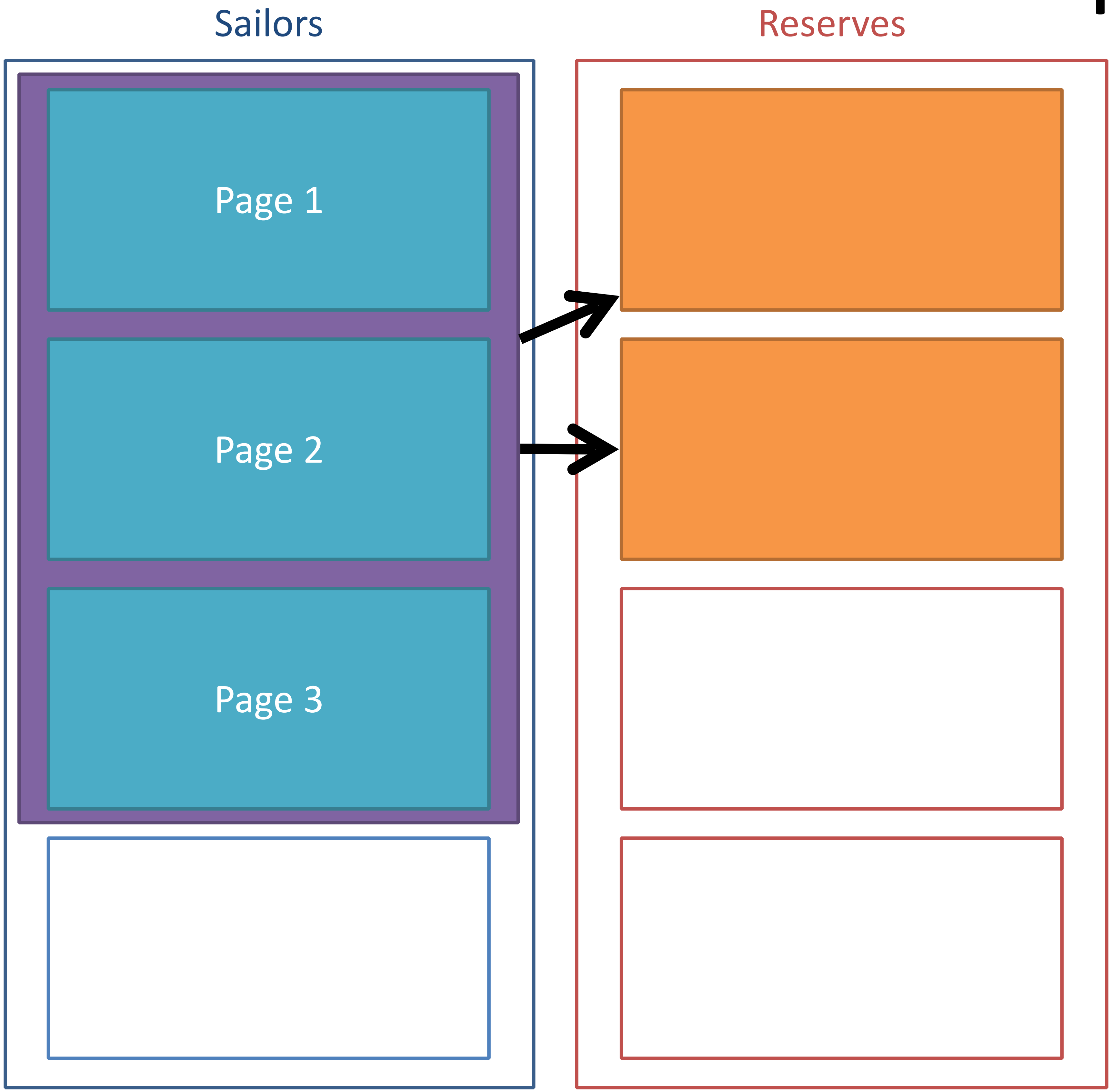
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



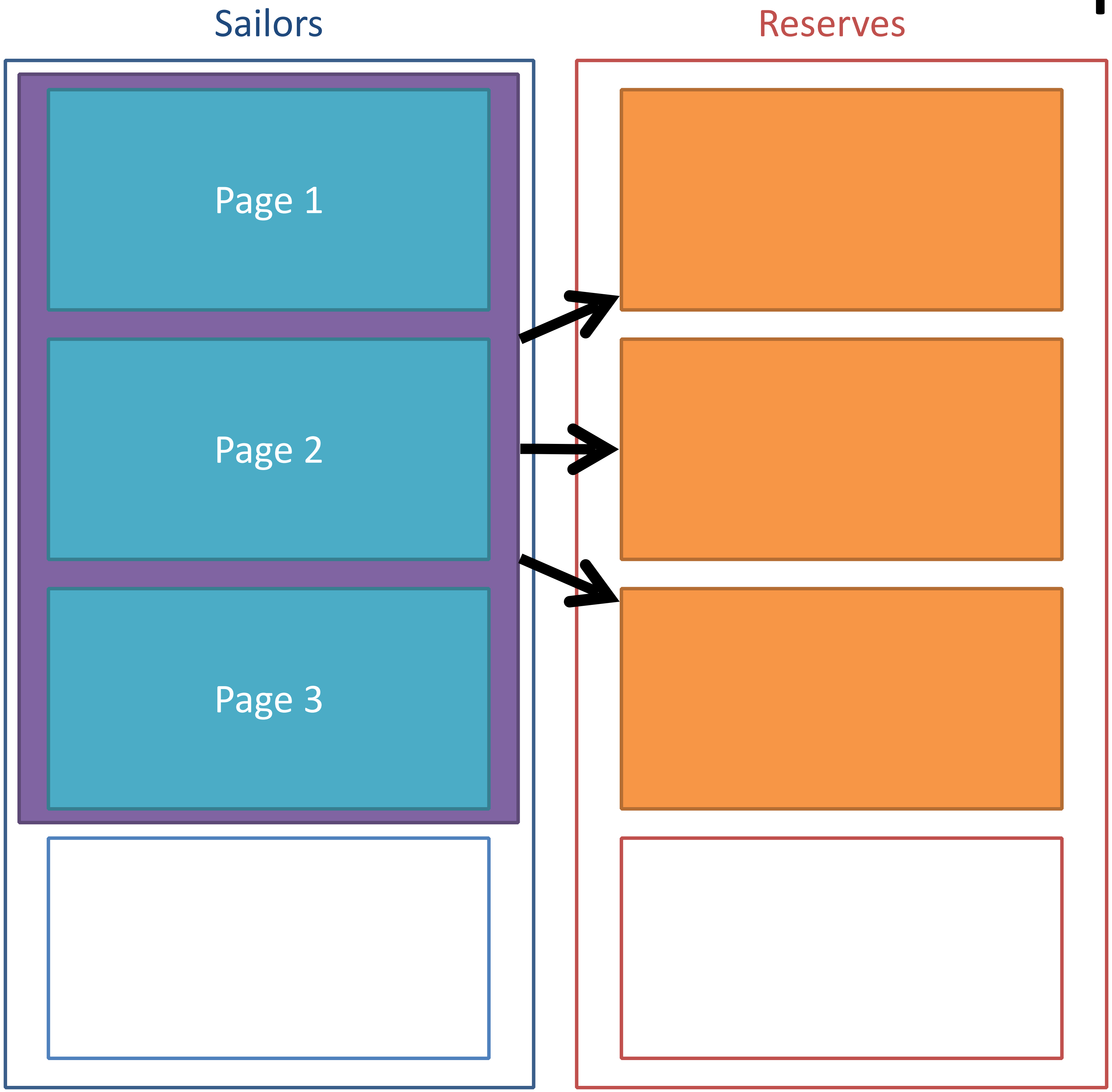
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



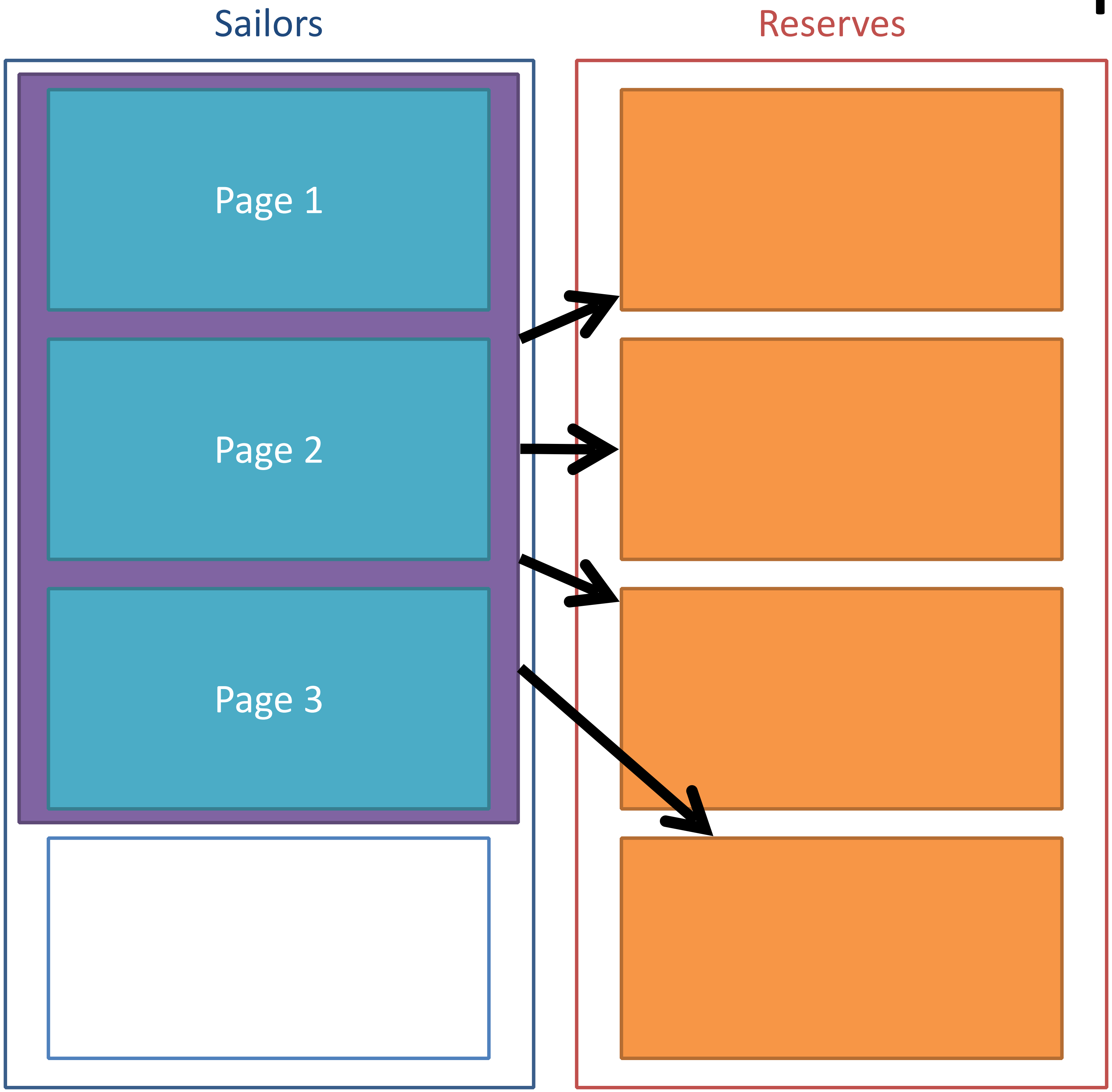
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



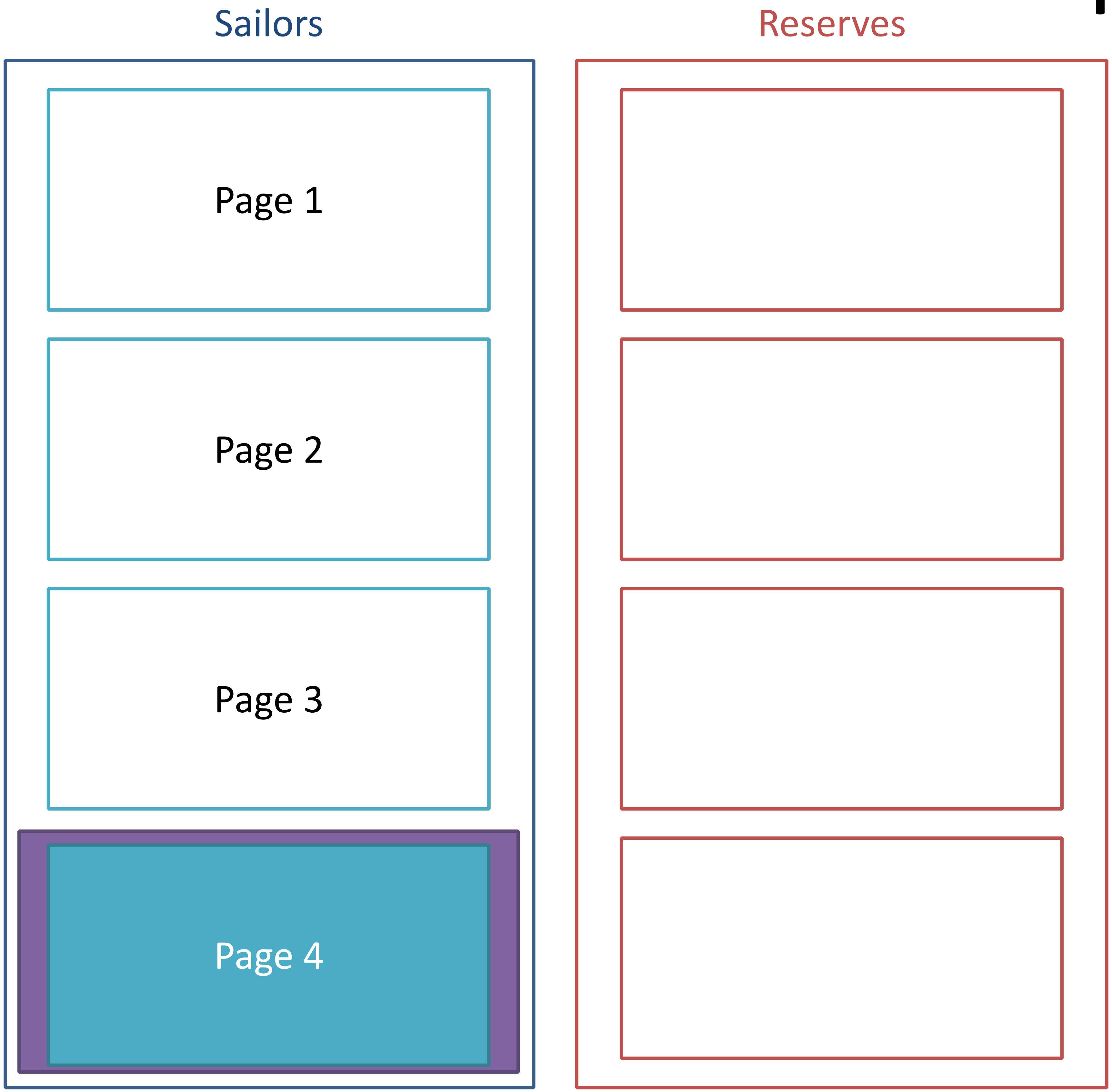
**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

# Chunk Nested Loops Join



**Key idea:**

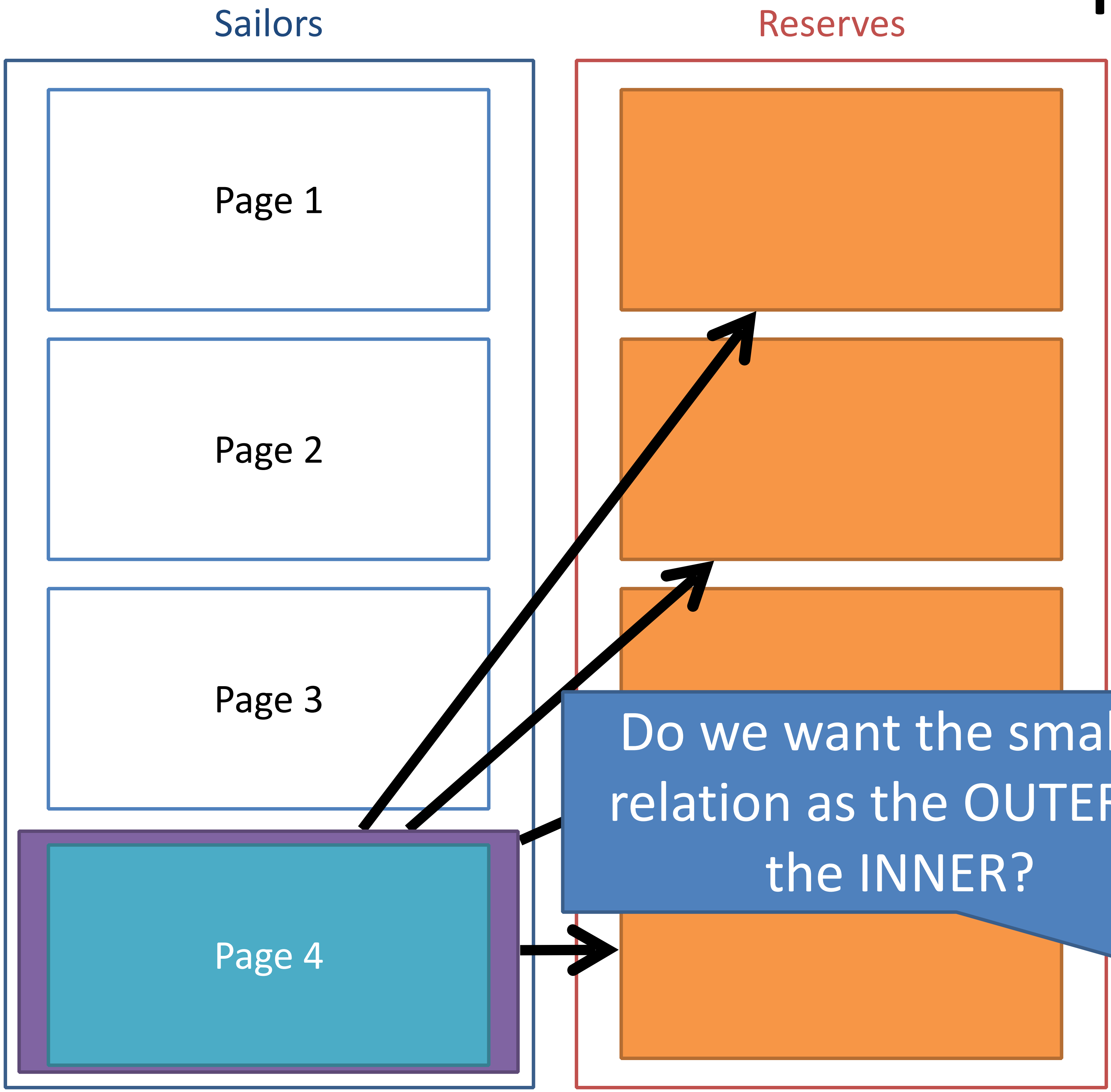
Take **k pages** of S and match with each page of R.

**Steps:**

1. Get **k** pages of S.
2. Iterate through each page in R.
3. Compare tuples in each.

**Notation: [S] == “# pages in S” ; |S| == “# tuples in S”**

# Chunk Nested Loops Join



Do we want the smaller relation as the OUTER or the INNER?

**Key idea:**

Take **k pages** of S and match with each page of R.

**Steps:**

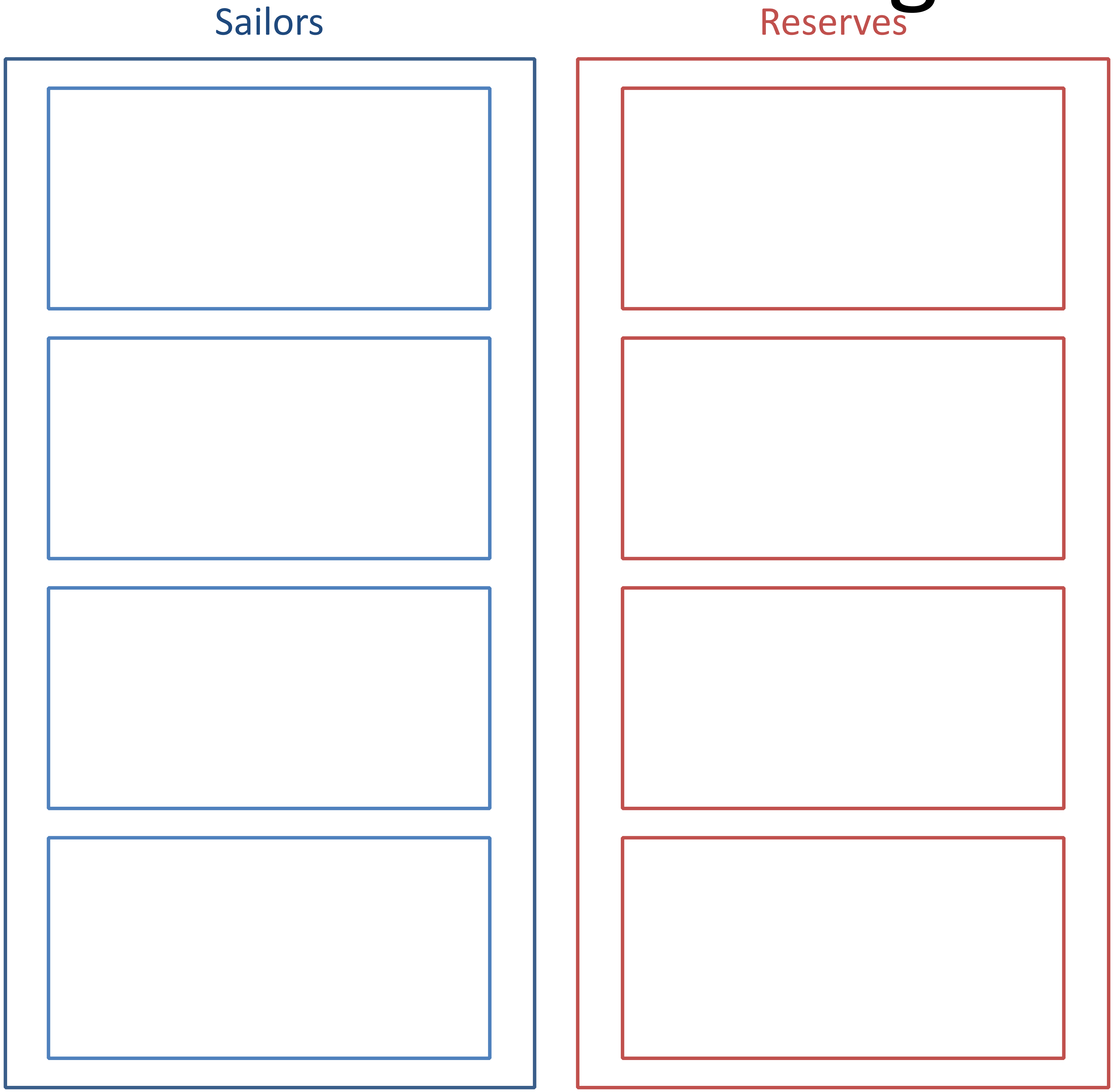
1. Get **k** pages of S.
2. Iterate through each page in R.

Compare tuples in each.

s:

[S] + ([S] / k)*[R]

# Sort-Merge Join



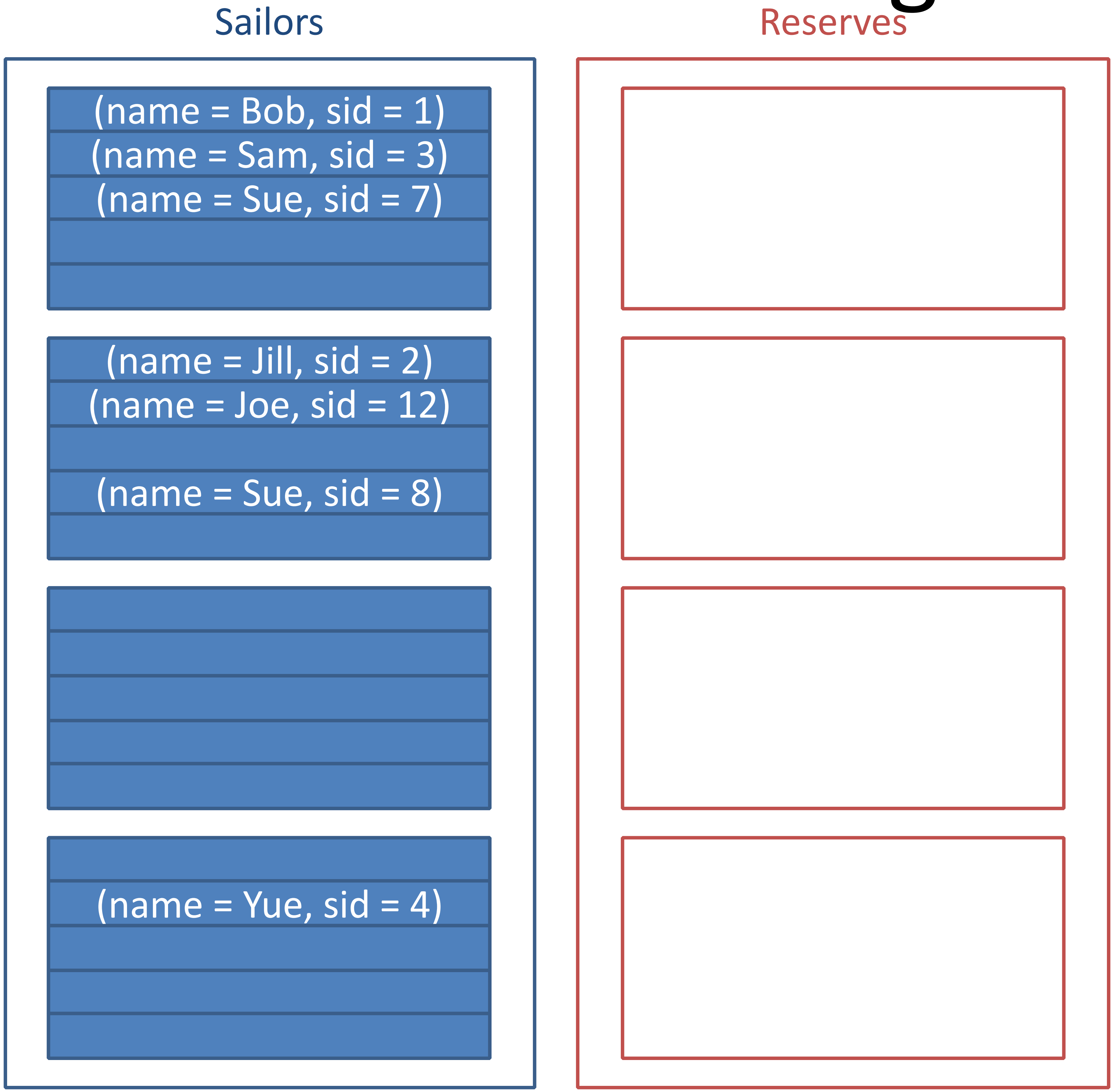
**Key idea:**

Sort S and R, then merge them!

**Steps:**

1. Sort S and R.
2. “Zip” or merge.

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Sam, sid = 3)
(name = Sue, sid = 7)

(name = Jill, sid = 2)
(name = Joe, sid = 12)

(name = Sue, sid = 8)

(name = Yue, sid = 4)

Reserves

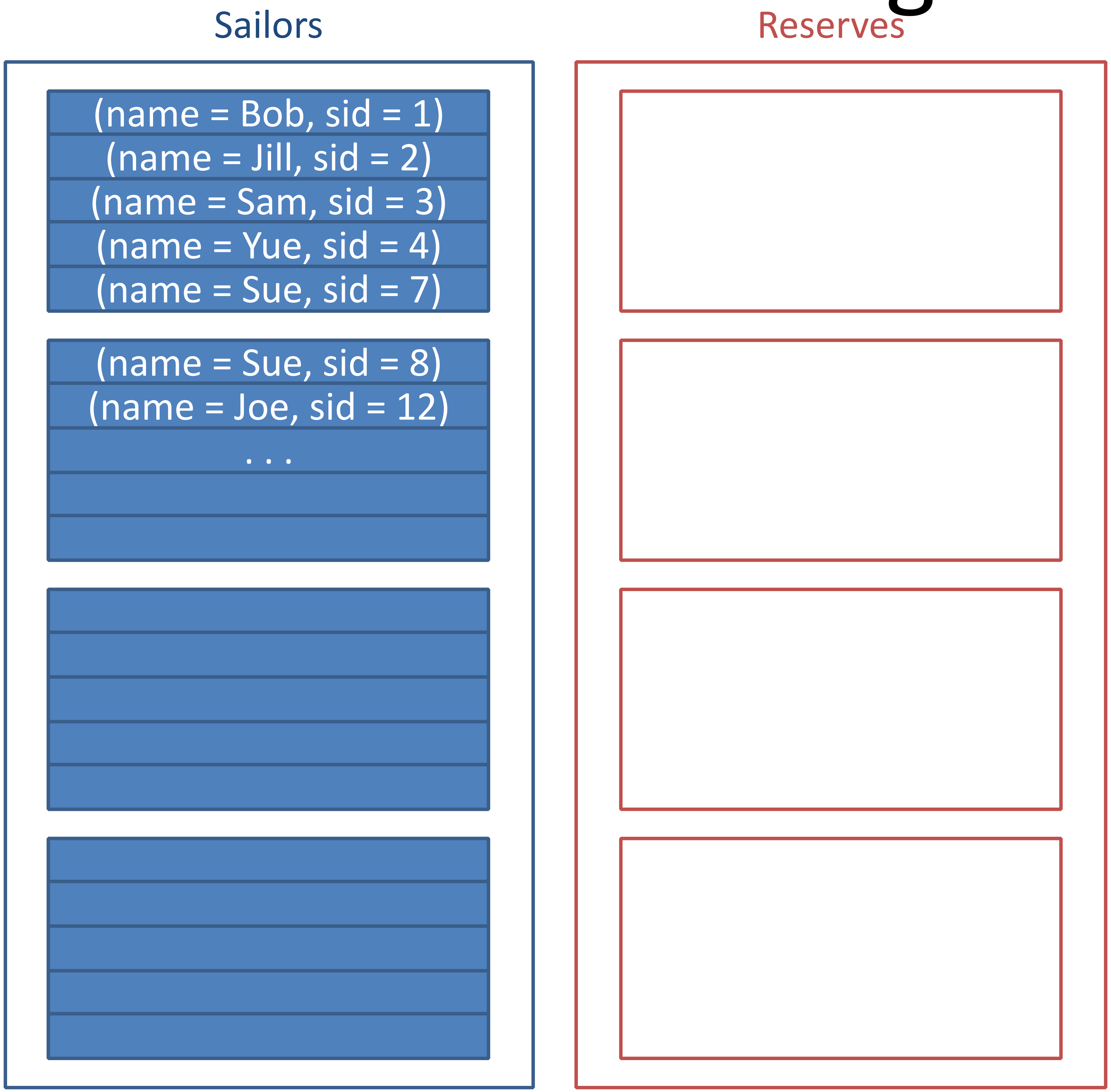
**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

1. Sort S and R.
2. “Zip” or merge.

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

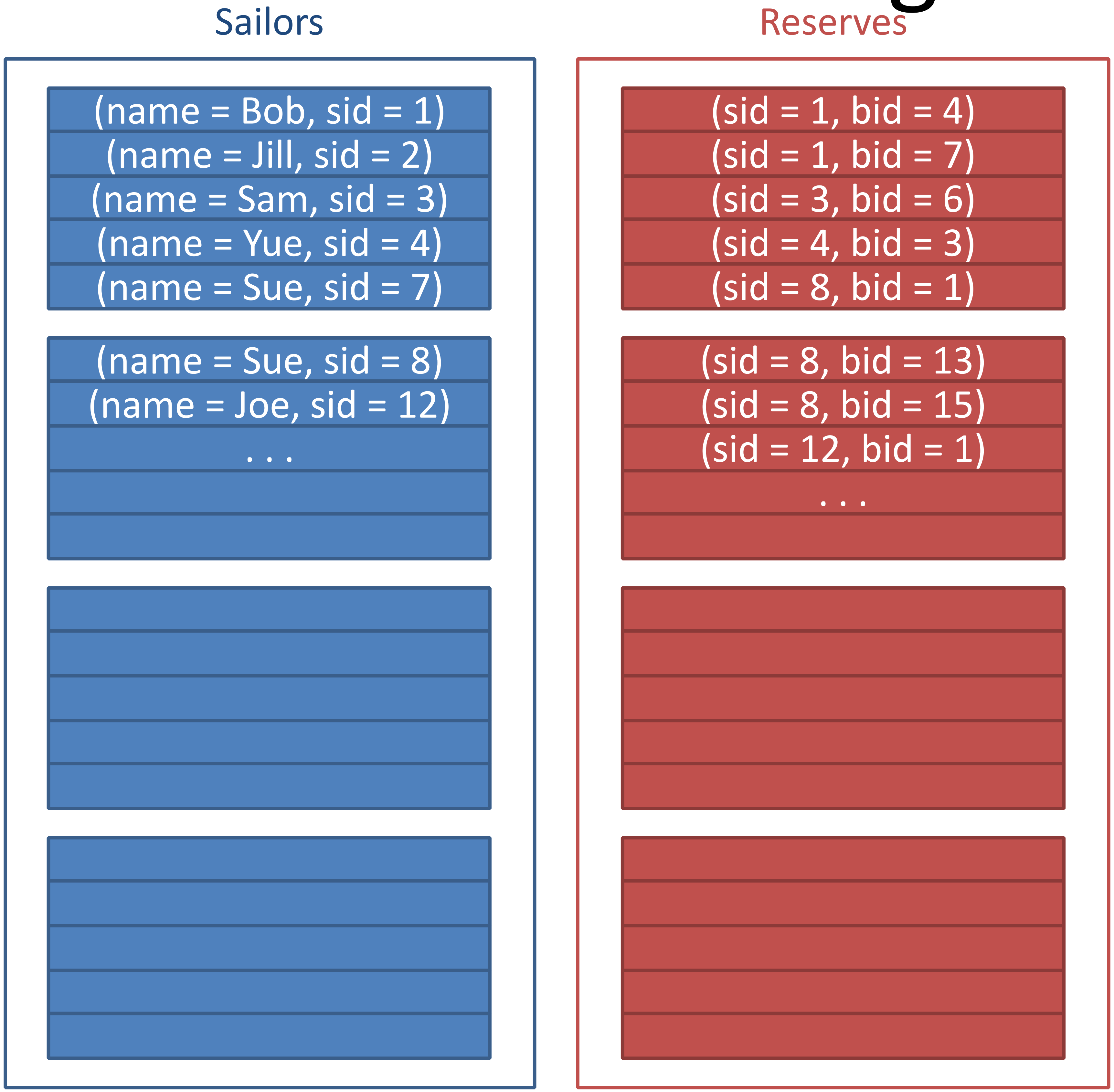
**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

1. Sort S and R.
2. “Zip” or merge.

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

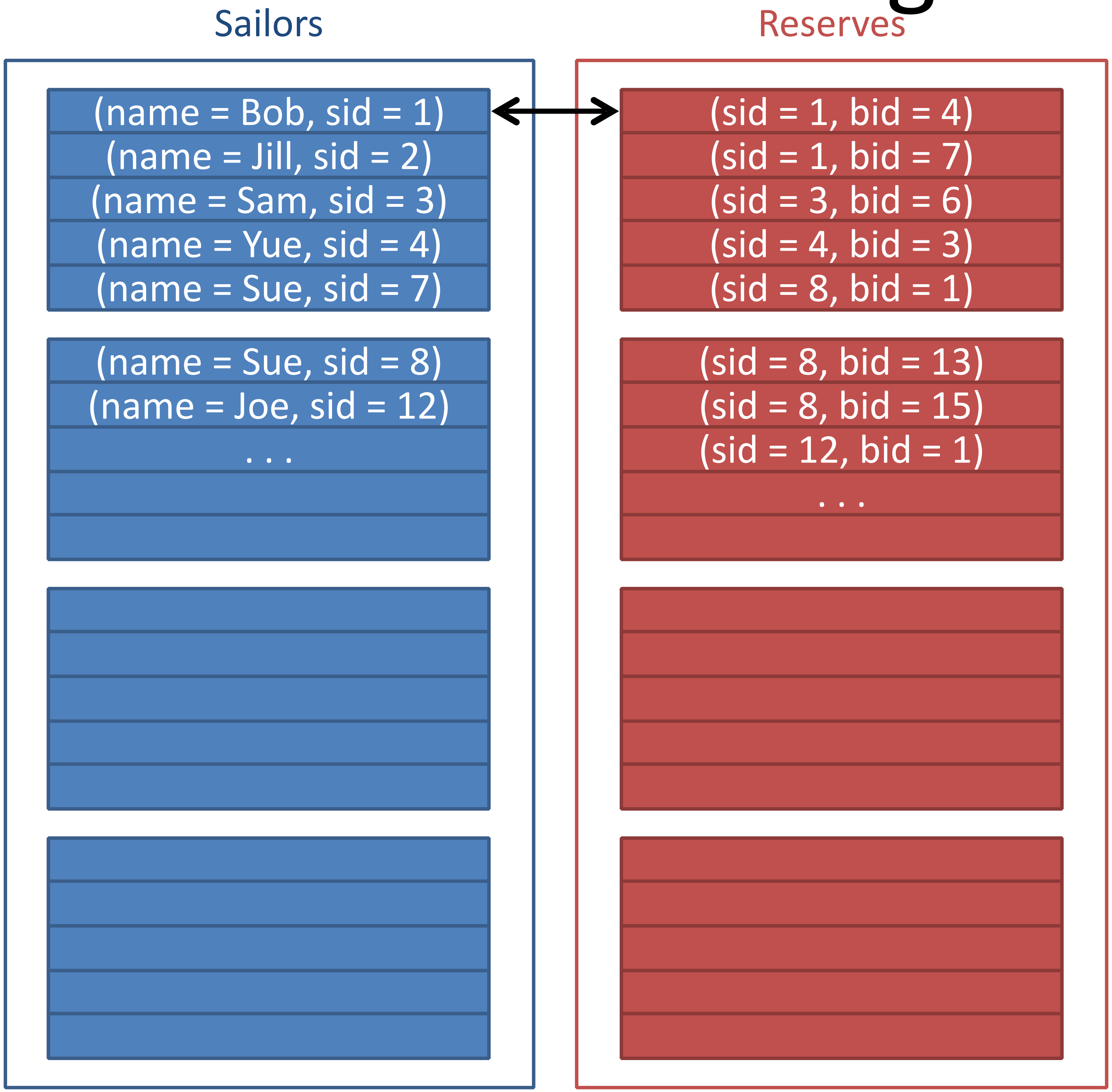
**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

1. Sort S and R.
2. “Zip” or merge.

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

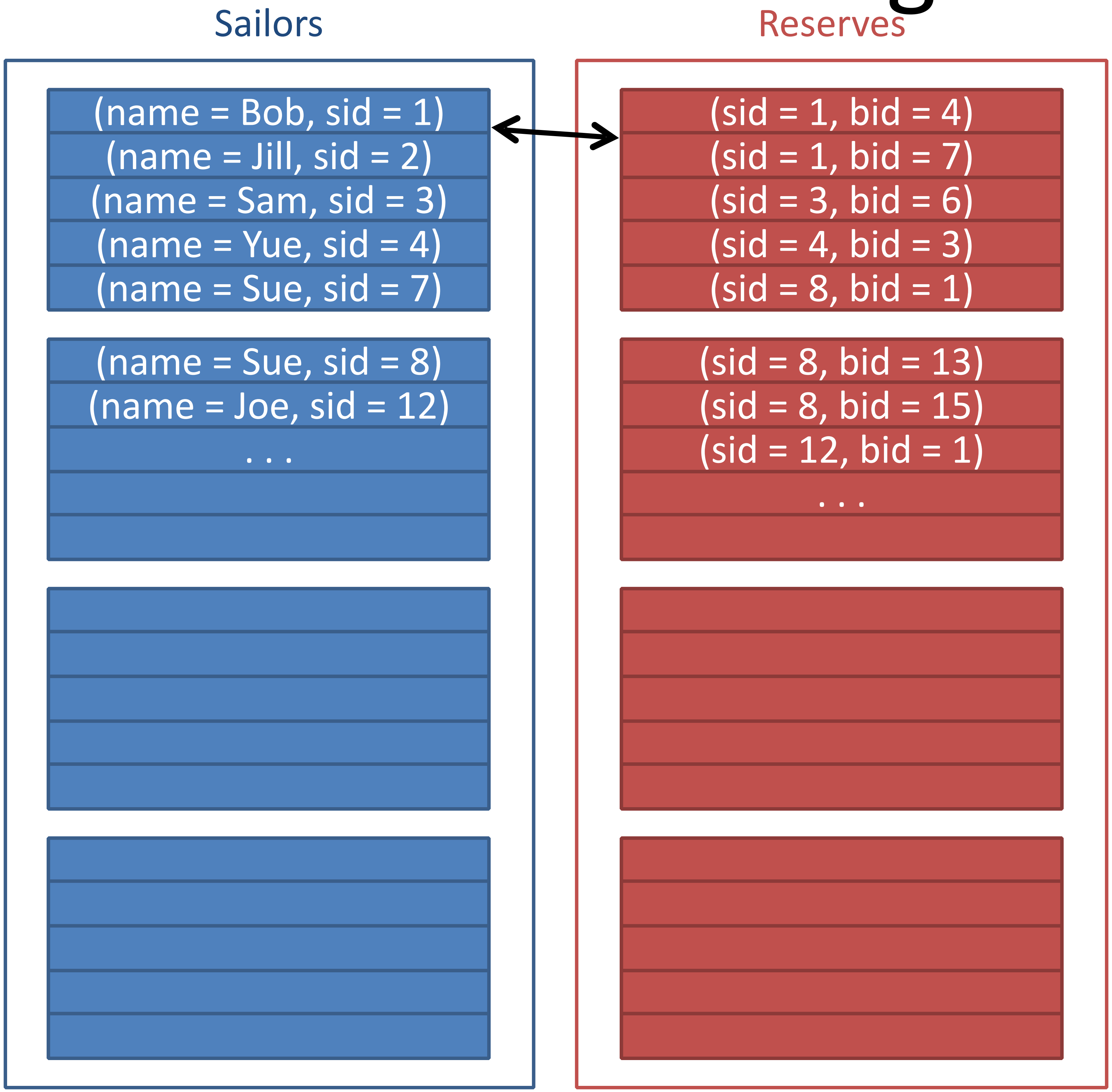
**Steps:**

1. Sort S and R.
2. “Zip” or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

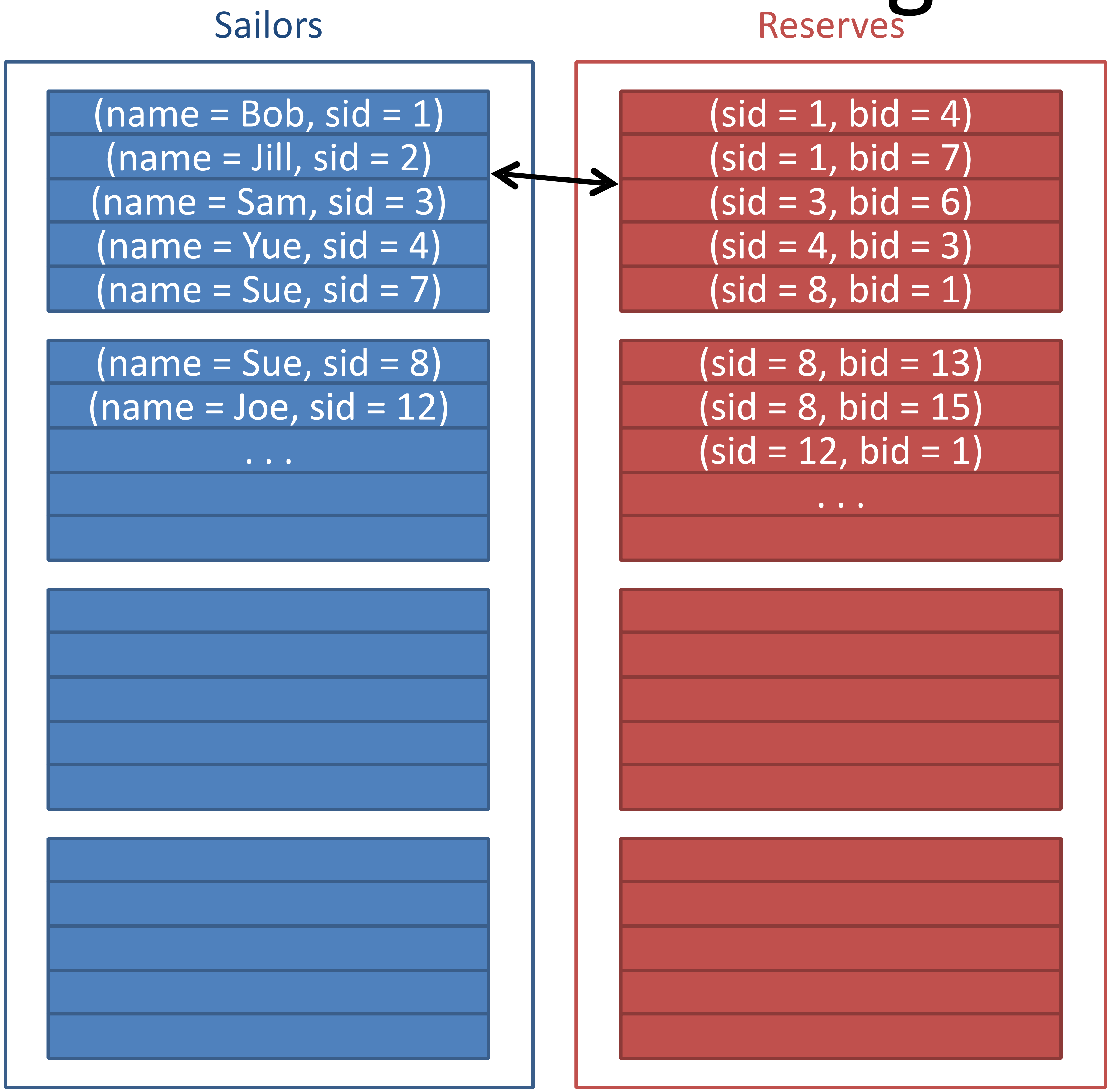
**Steps:**

1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

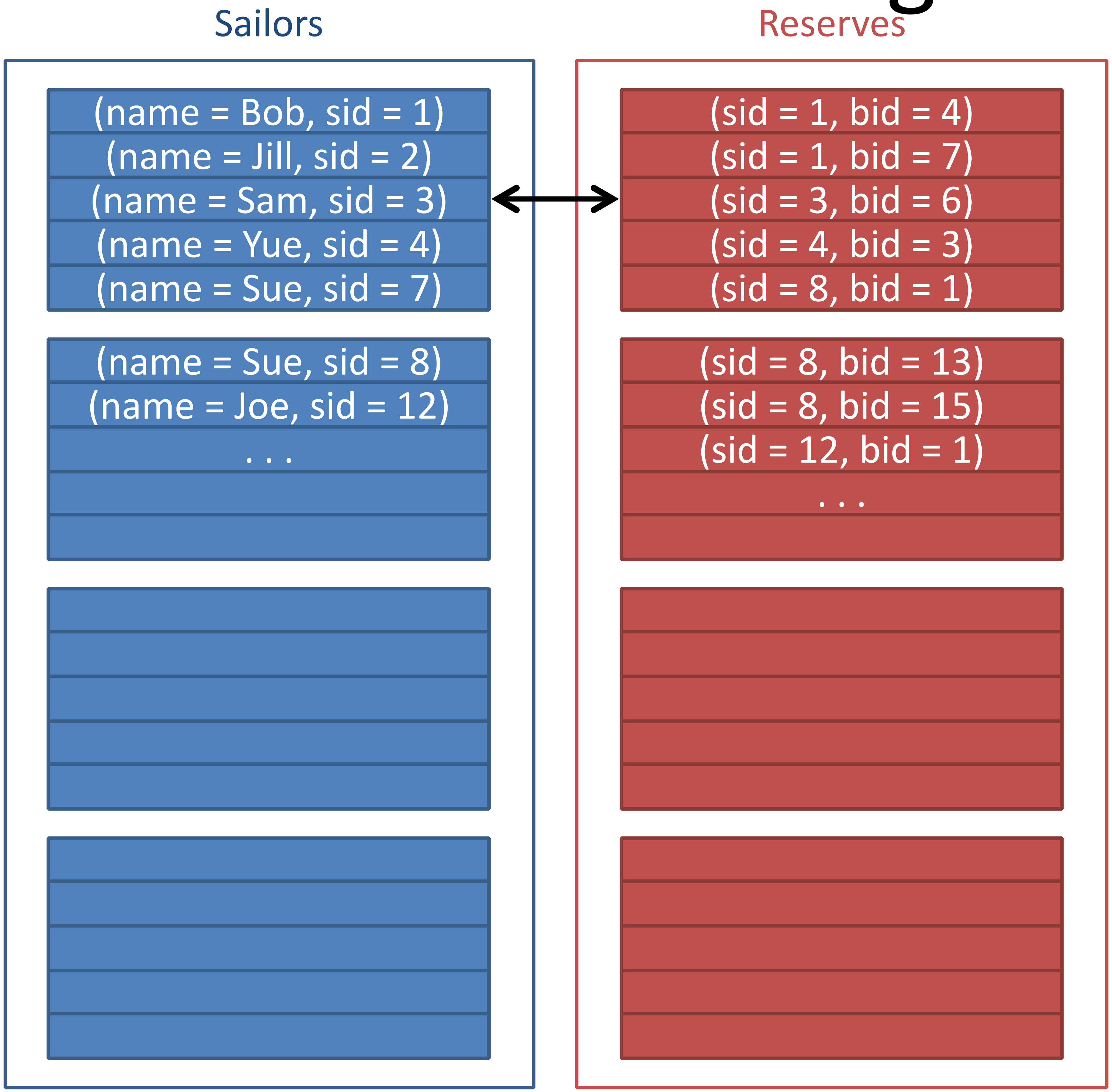
**Steps:**

1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

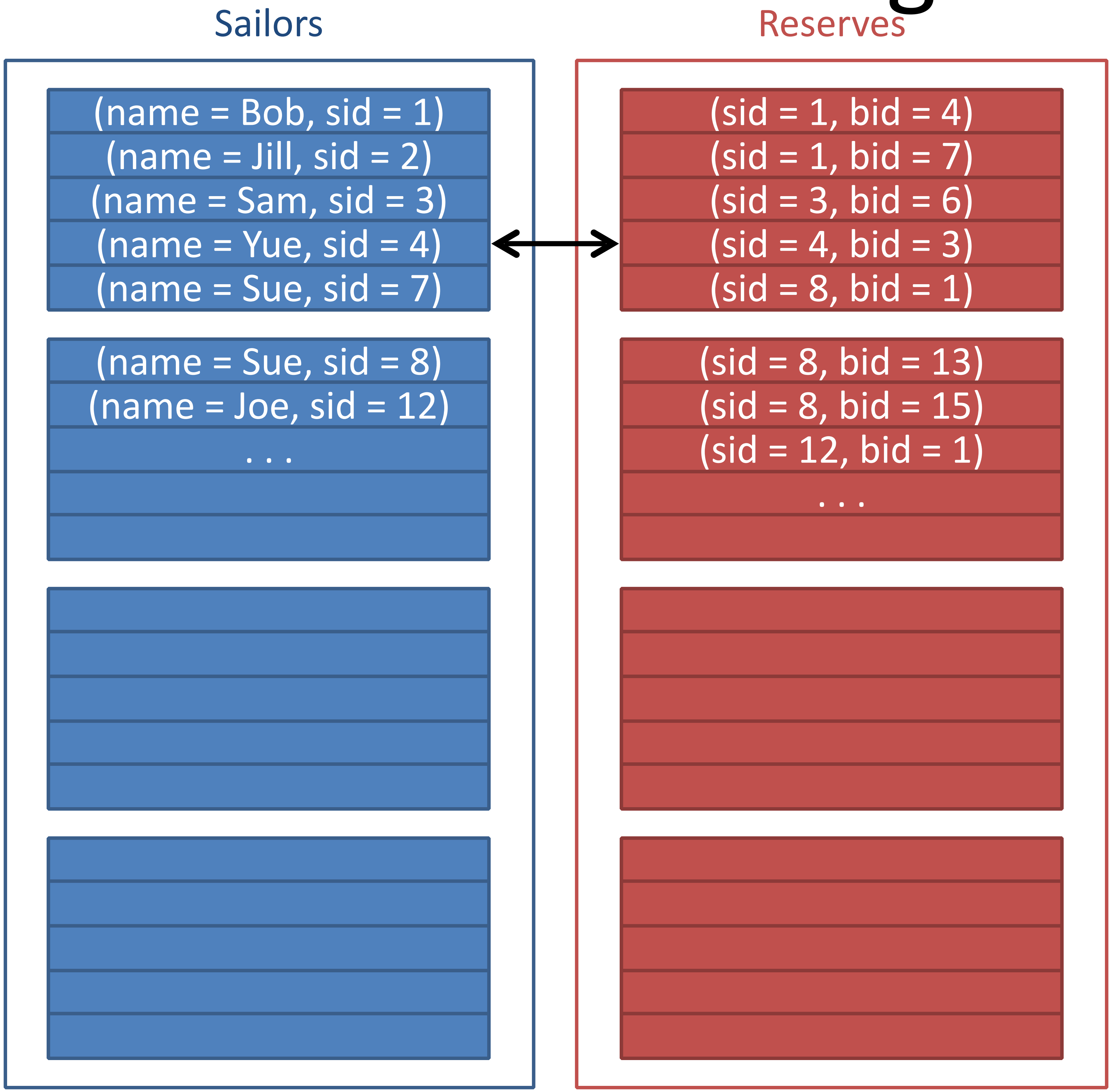
**Steps:**

1. Sort S and R.
2. “Zip” or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)

# Sort-Merge Join



(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

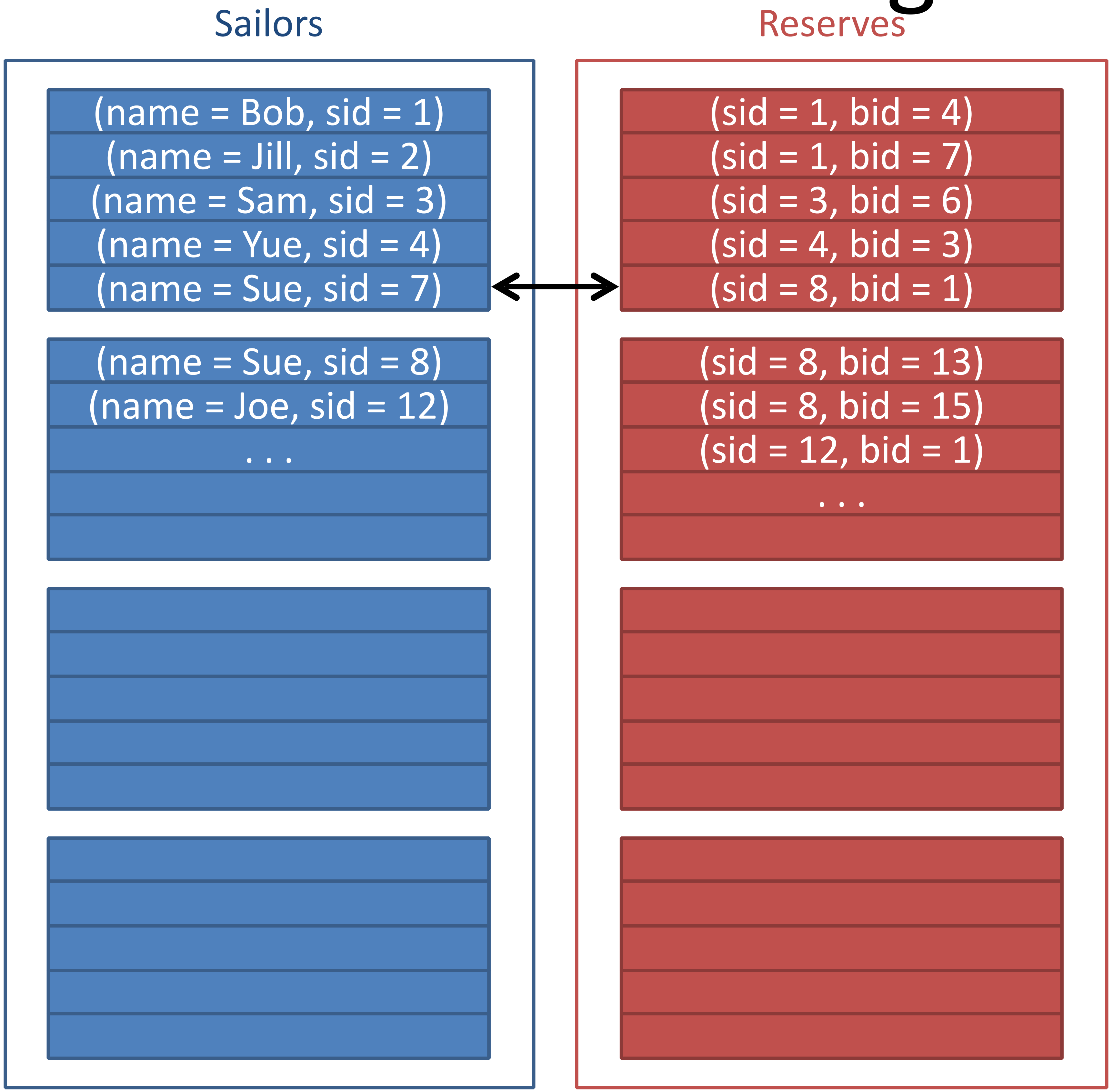
**Steps:**

1. Sort S and R.
2. “Zip” or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

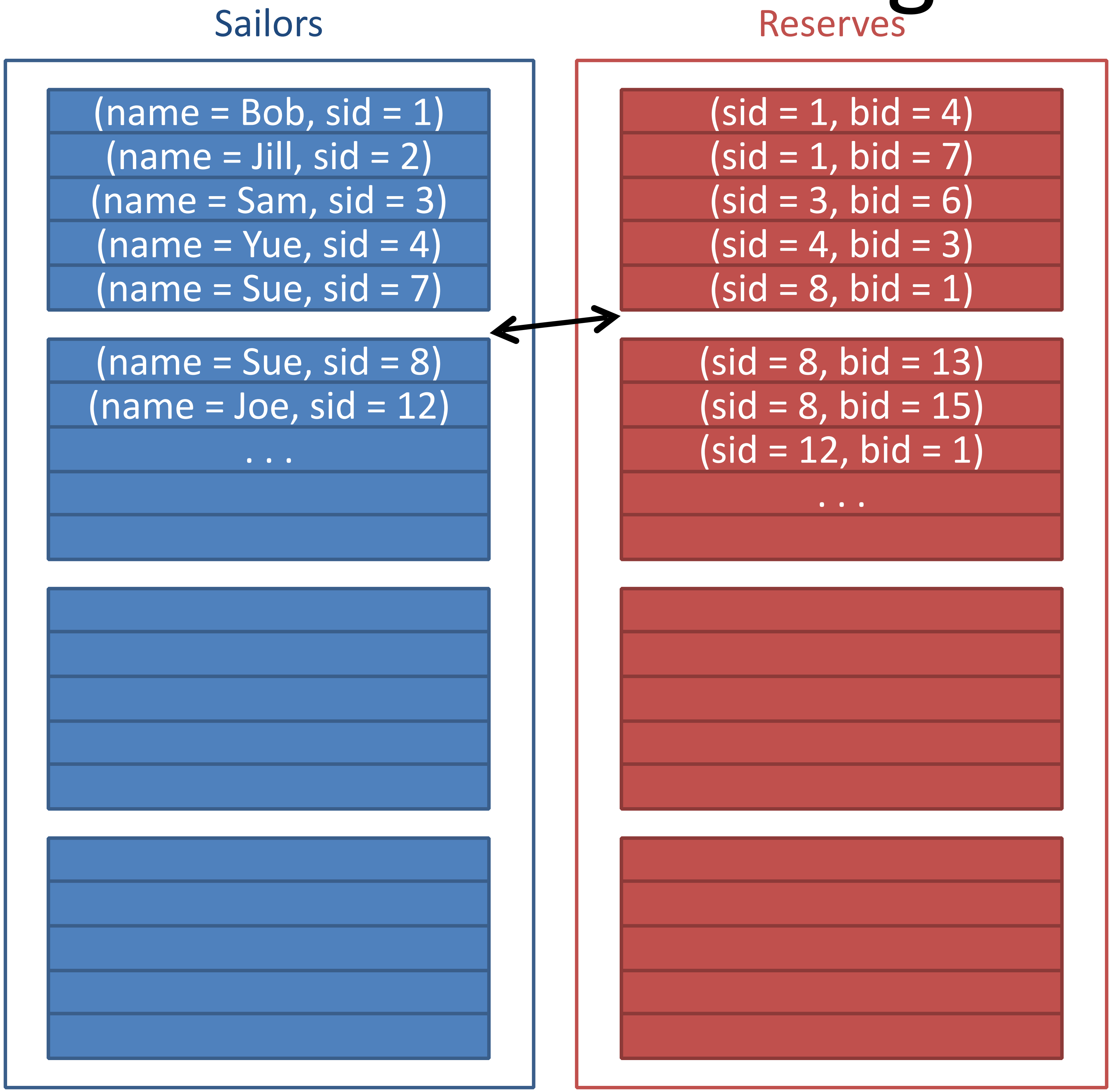
**Steps:**

1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

# Sort-Merge Join



**Key idea:**

Sort S and R **on join column**, then merge them!

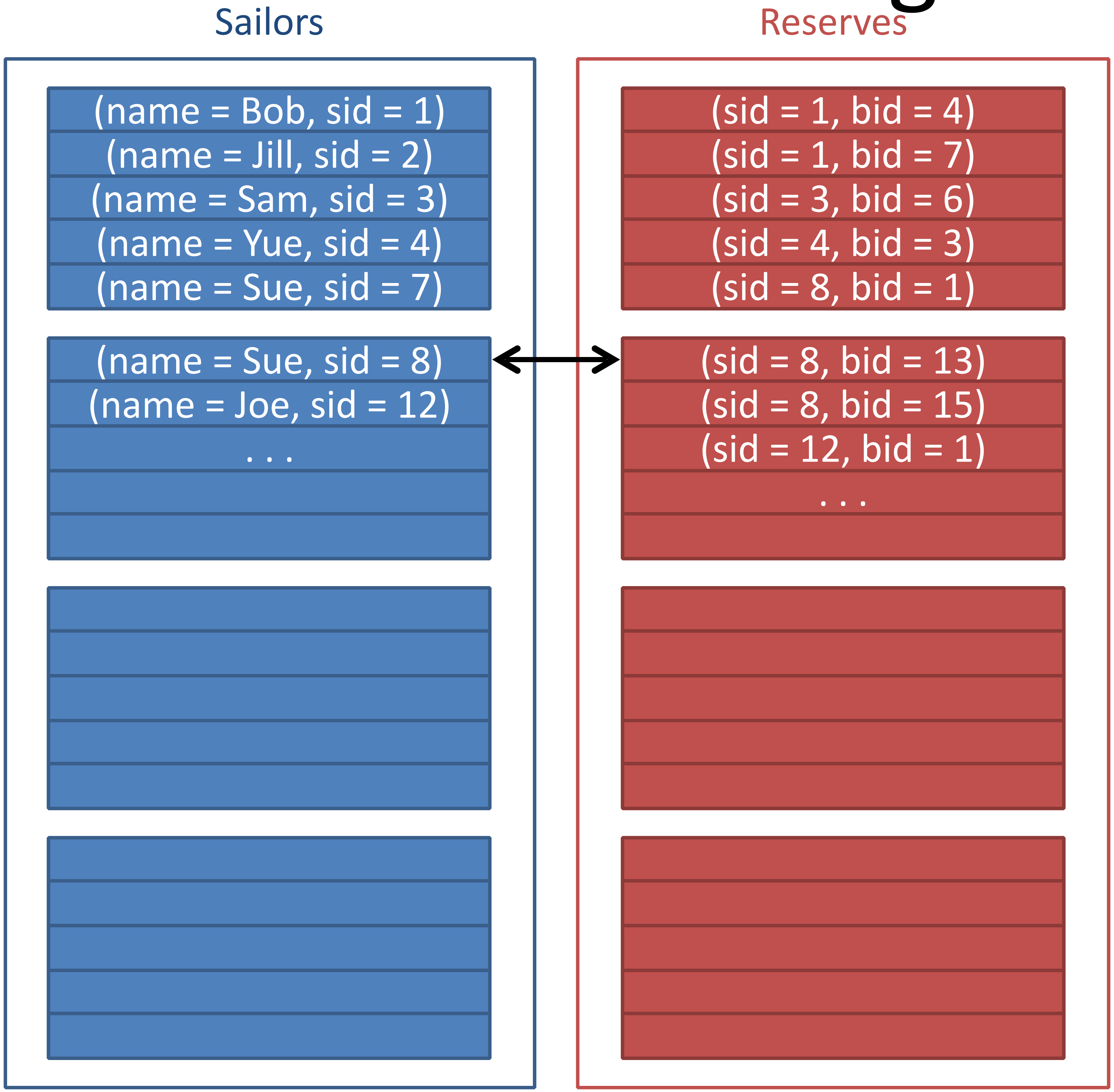
**Steps:**

1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)
(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)
(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

1. Sort S and R.
2. "Zip" or merge.

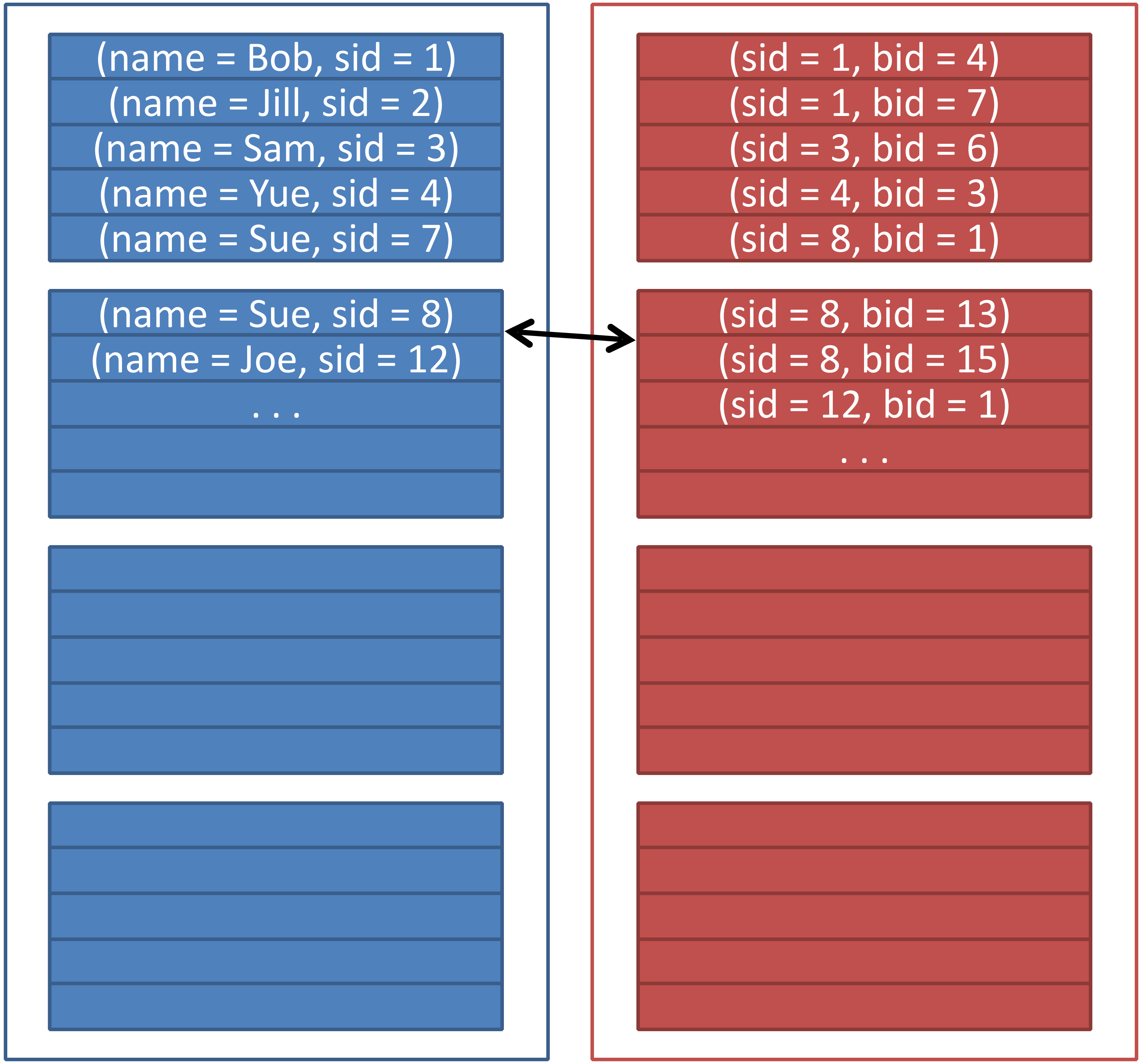
**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

# Sort-Merge Join

Sailors

Reserves



(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

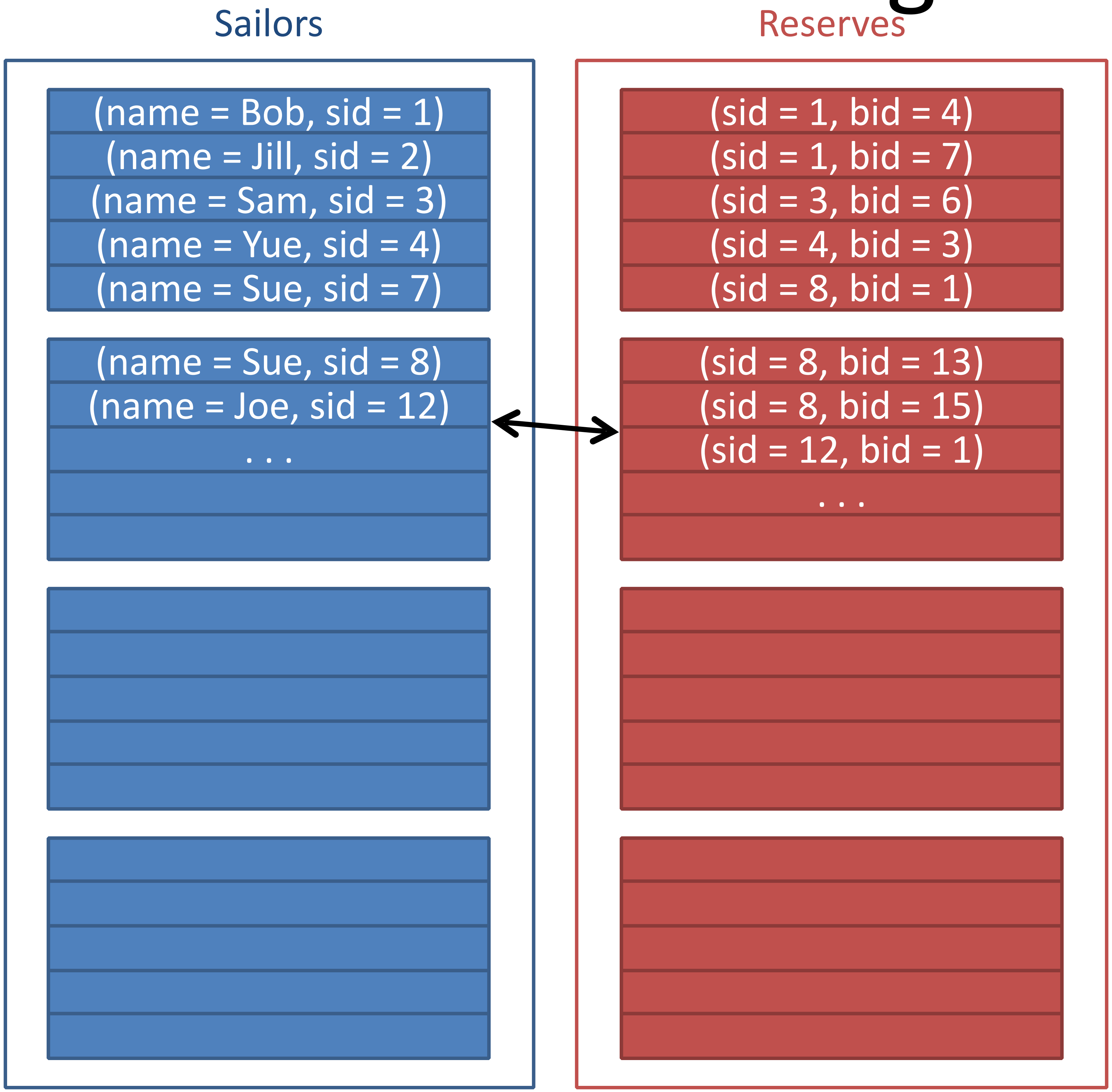
**Steps:**

1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

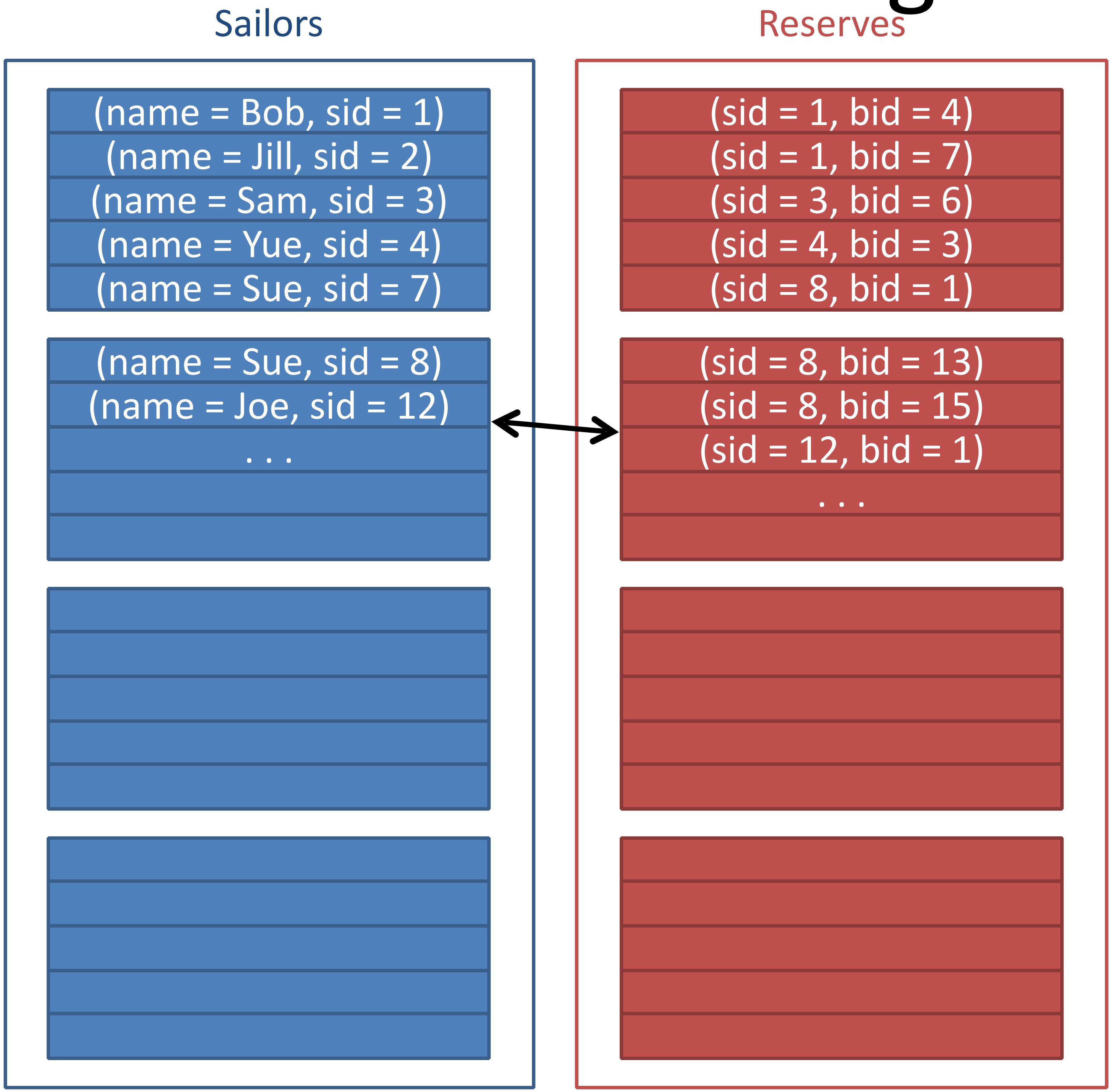
1. Sort S and R.
2. "Zip" or merge.

**Output:**

(name = Bob, sid = 1, bid = 4)
(name = Bob, sid = 1, bid = 7)
(name = Sam, sid = 3, bid = 6)
. . .

Notation: [S] == “# pages in S” ; |S| == “# tuples in S”

# Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

**Key idea:**

Sort S and R **on join column**, then merge them!

**Steps:**

1. Sort S and R.
2. “Zip” or merge.

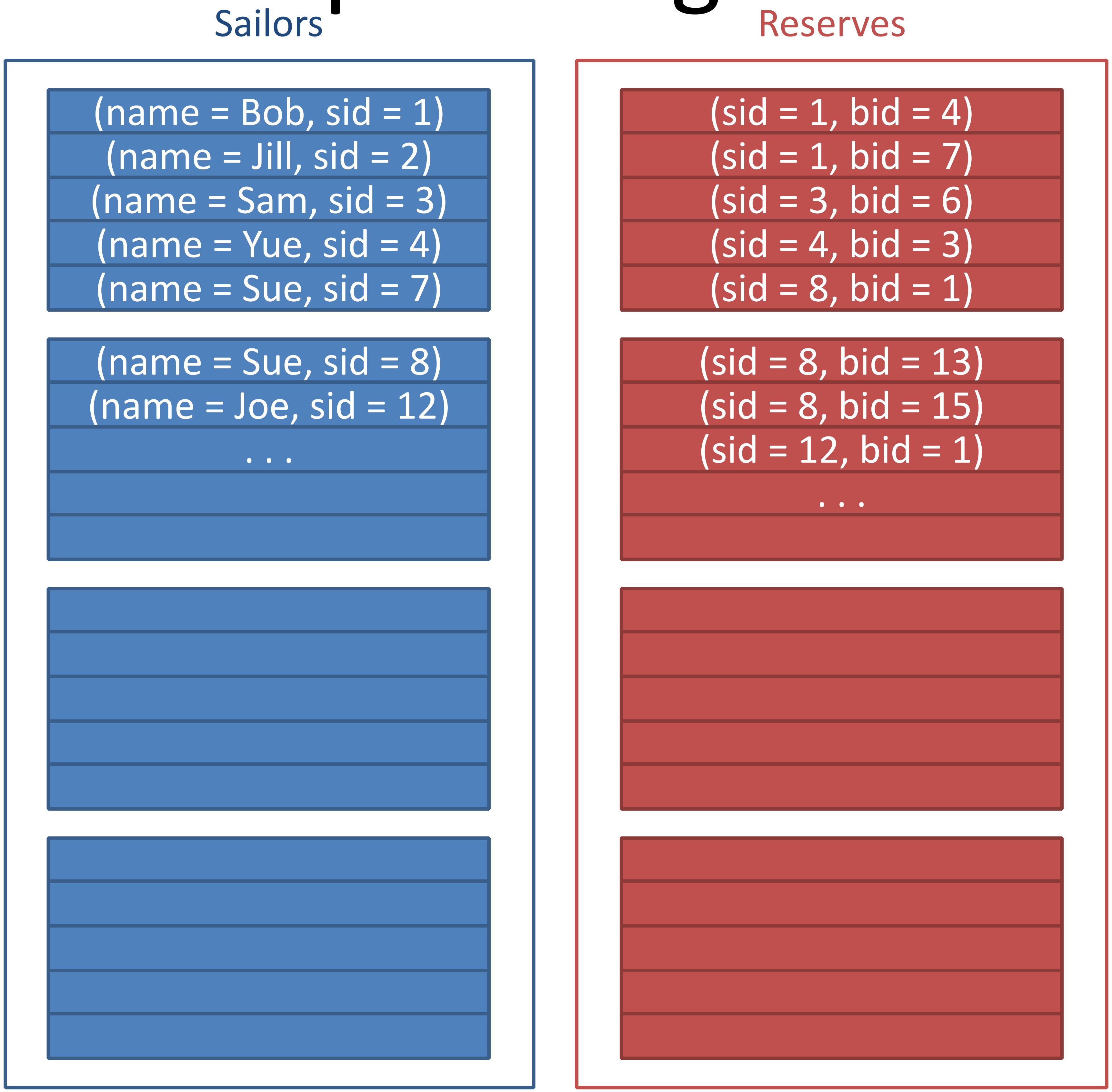
**I/Os:**

~5([S] + [R])

Sorting: 4([S]+[R])

Merging: [S]+[R]

# Optimizing Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Sam, sid = 3)
(name = Yue, sid = 4)
(name = Sue, sid = 7)

(name = Sue, sid = 8)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 3, bid = 6)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 12, bid = 1)
. . .

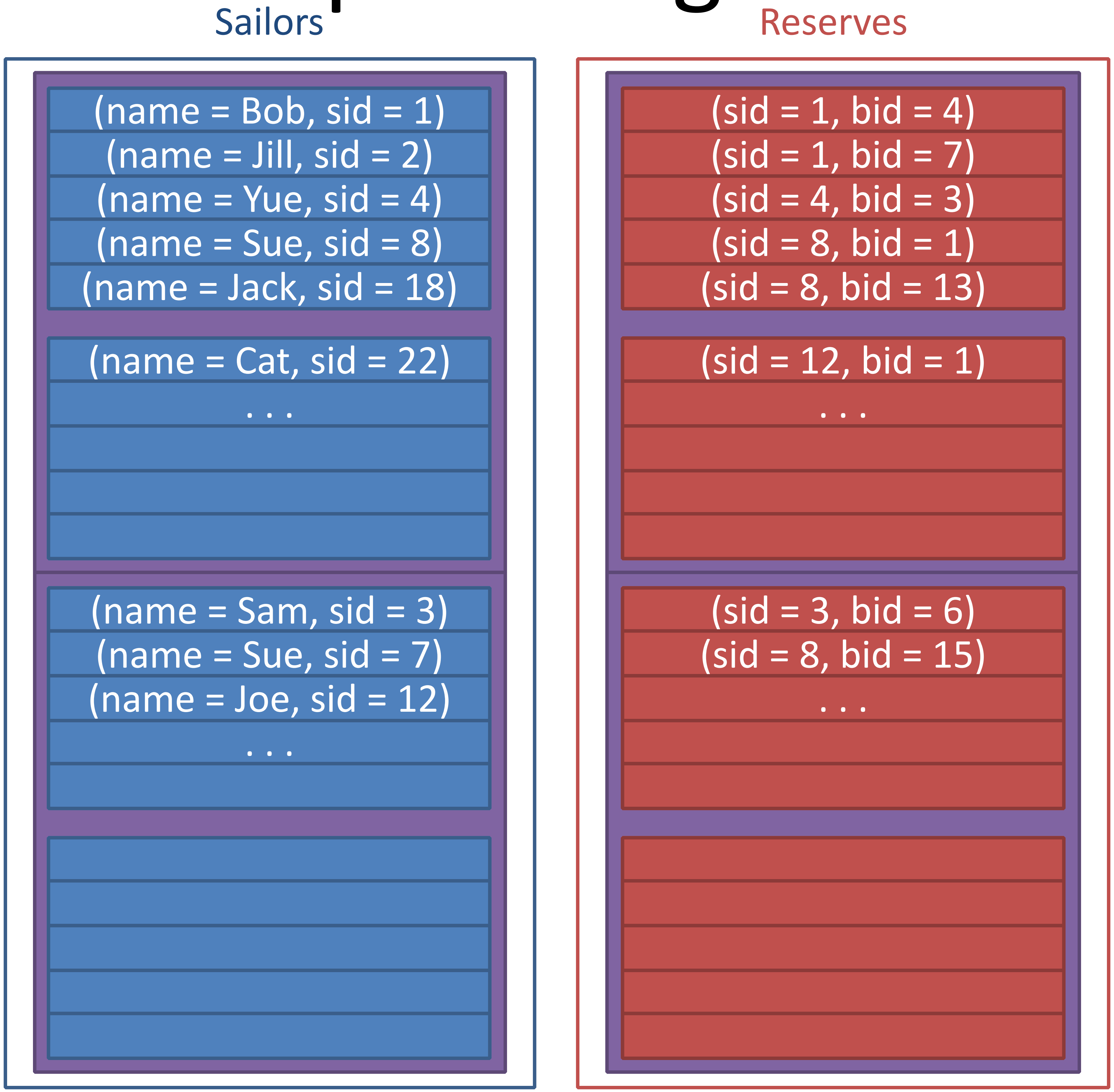
**Key idea:**

Internal Sort on both. Perform merge on all runs!

**Steps:**

1. Internal sort S and R. (Pass 0)
2. Merge all runs.

# Optimizing Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Yue, sid = 4)
(name = Sue, sid = 8)
(name = Jack, sid = 18)

(name = Cat, sid = 22)
. . .

(name = Sam, sid = 3)
(name = Sue, sid = 7)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 4, bid = 3)
(sid = 8, bid = 1)
(sid = 8, bid = 13)

(sid = 12, bid = 1)
. . .

(sid = 3, bid = 6)
(sid = 8, bid = 15)
. . .

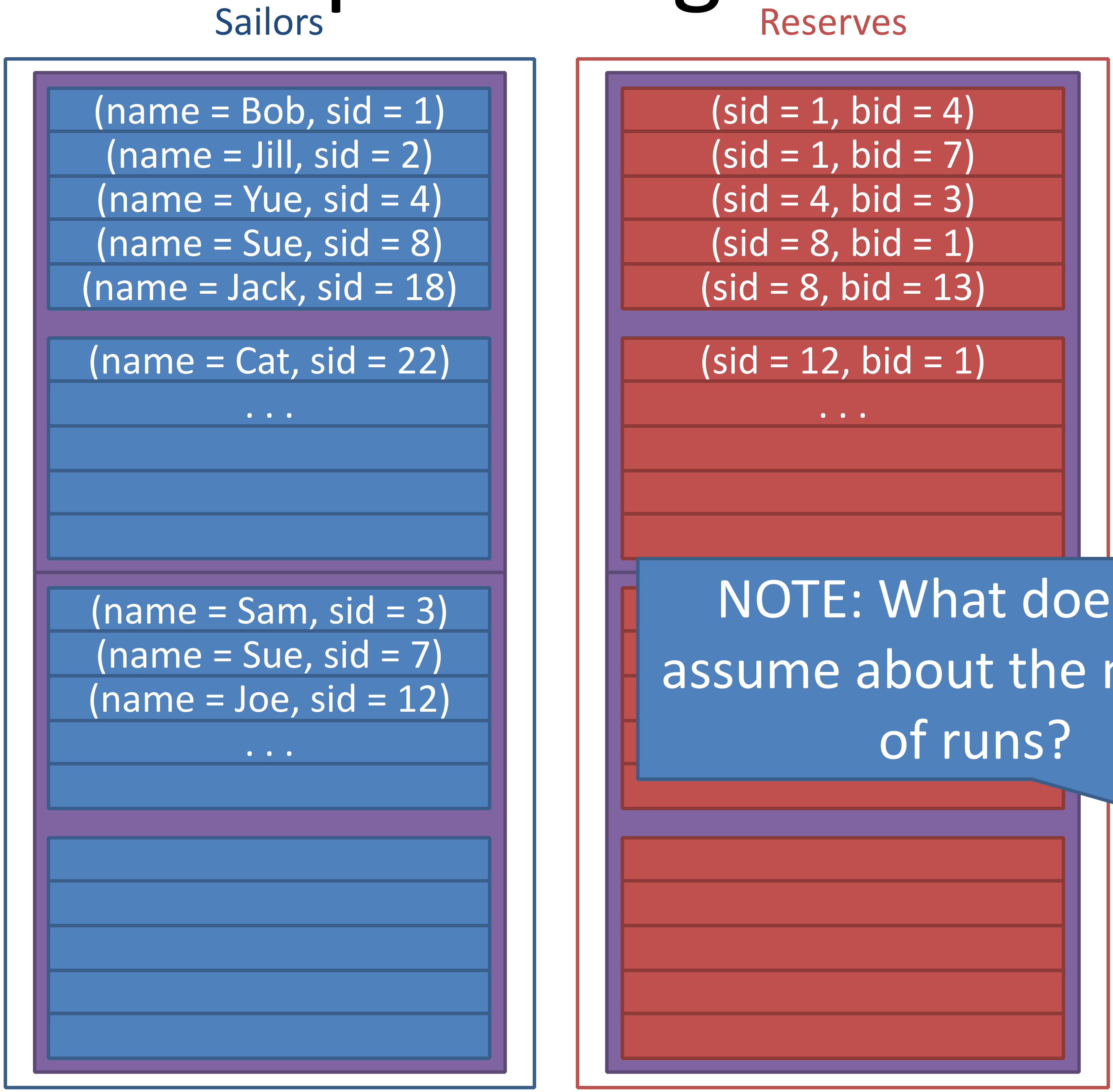
**Key idea:**

Internal Sort on both. Perform merge on all runs!

**Steps:**

1. Internal sort S and R. (Pass 0)
2. Merge all runs.

# Optimizing Sort-Merge Join



Sailors

(name = Bob, sid = 1)
(name = Jill, sid = 2)
(name = Yue, sid = 4)
(name = Sue, sid = 8)
(name = Jack, sid = 18)

(name = Cat, sid = 22)
. . .

(name = Sam, sid = 3)
(name = Sue, sid = 7)
(name = Joe, sid = 12)
. . .

Reserves

(sid = 1, bid = 4)
(sid = 1, bid = 7)
(sid = 4, bid = 3)
(sid = 8, bid = 1)
(sid = 8, bid = 13)

(sid = 12, bid = 1)
. . .

**Key idea:**

Internal Sort on both. Perform merge on all runs!

**Steps:**

1. Internal sort S and R. (Pass 0)

ge all runs.

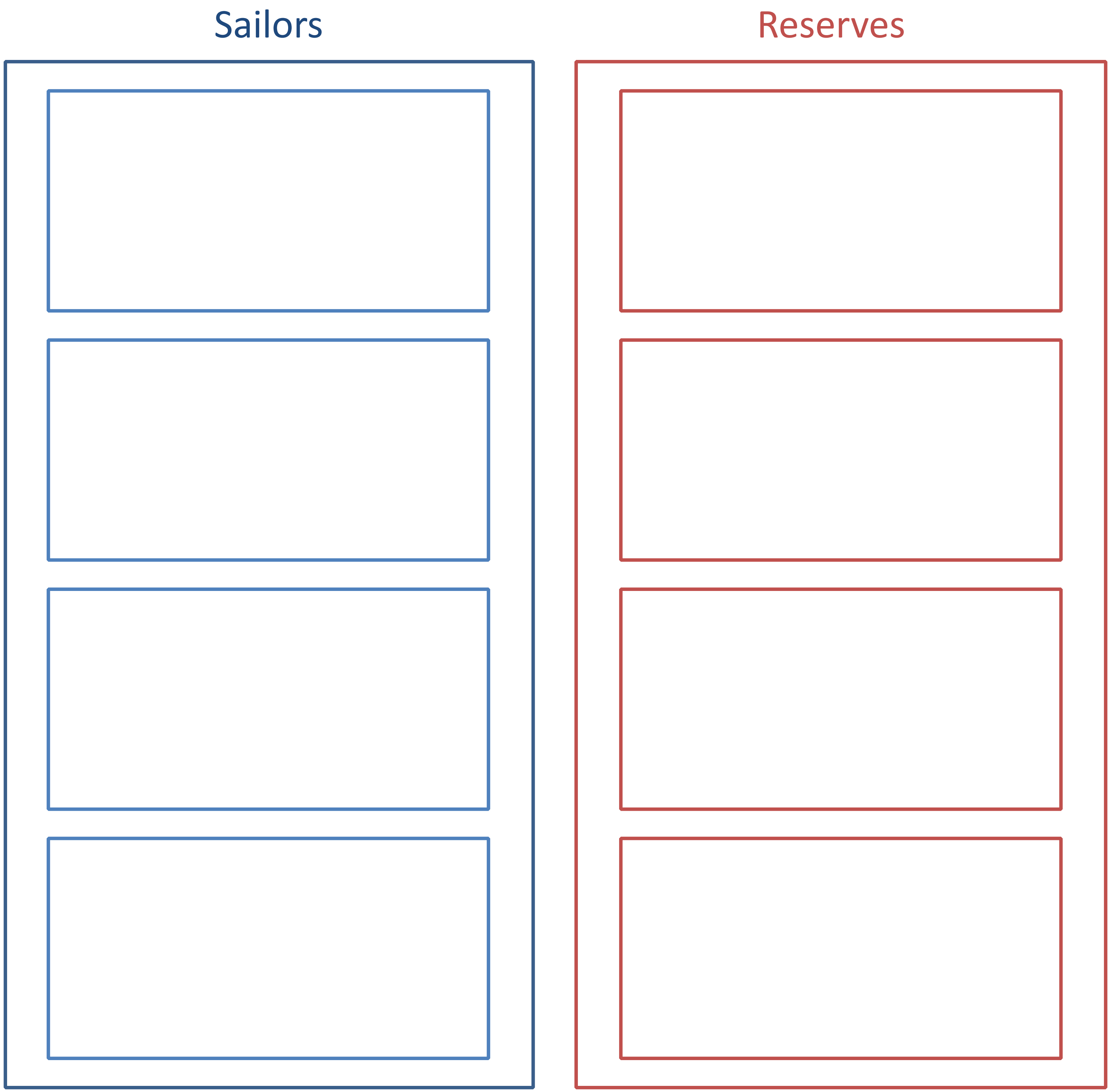
~3([S] + [R])

Pass 0: 2([S]+[R])

Merging: [S]+[R]

NOTE: What does this assume about the number of runs?

# Hash-Join



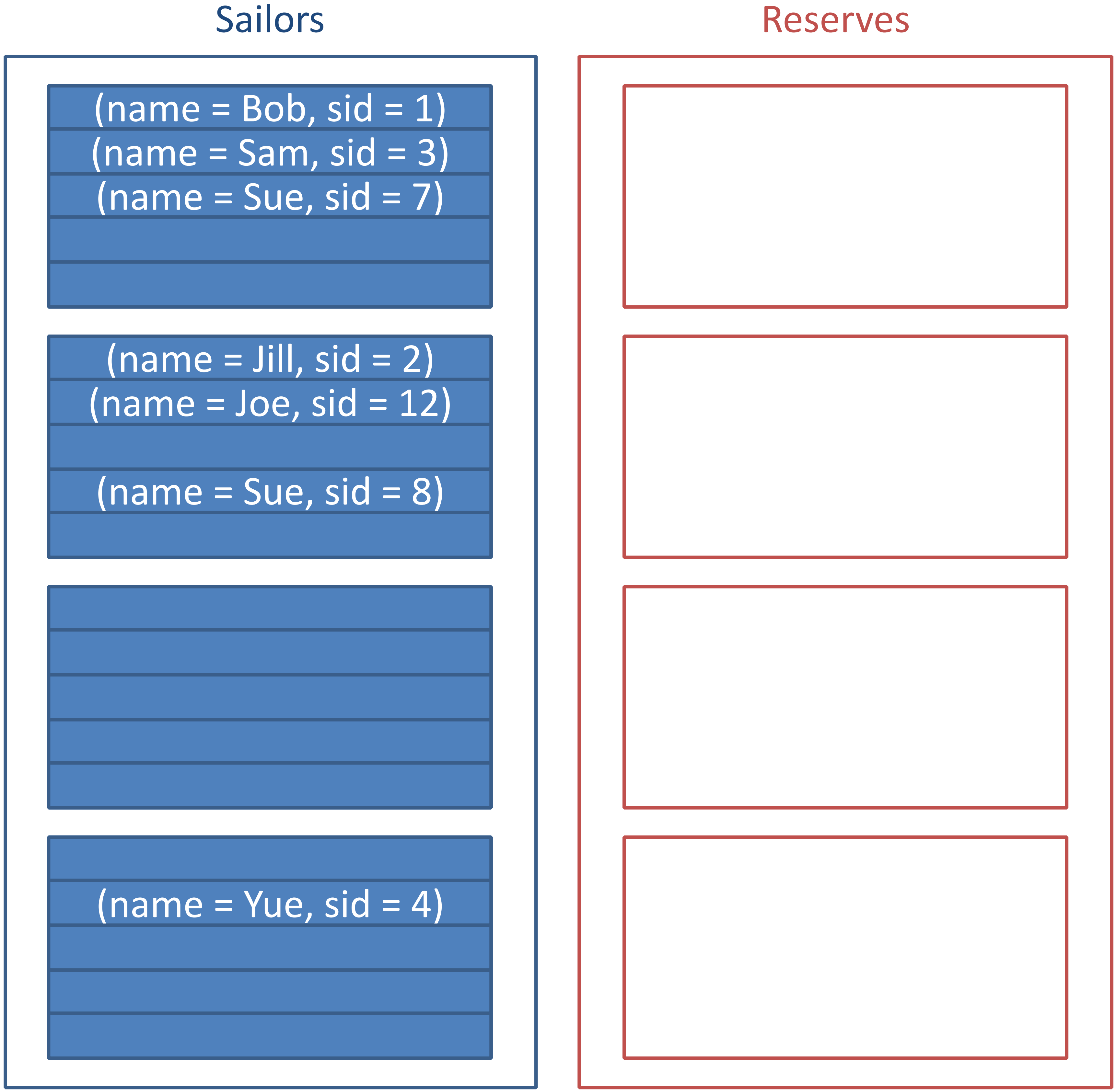
**Key idea:**

Partition S and R using same hash fn, then collect same partitions

**Steps:**

1. Partition S and R
2. Re-Hash, collect

# Hash-Join



Sailors

(name = Bob, sid = 1)
(name = Sam, sid = 3)
(name = Sue, sid = 7)

(name = Jill, sid = 2)
(name = Joe, sid = 12)

(name = Sue, sid = 8)

(name = Yue, sid = 4)

Reserves

**Key idea:**

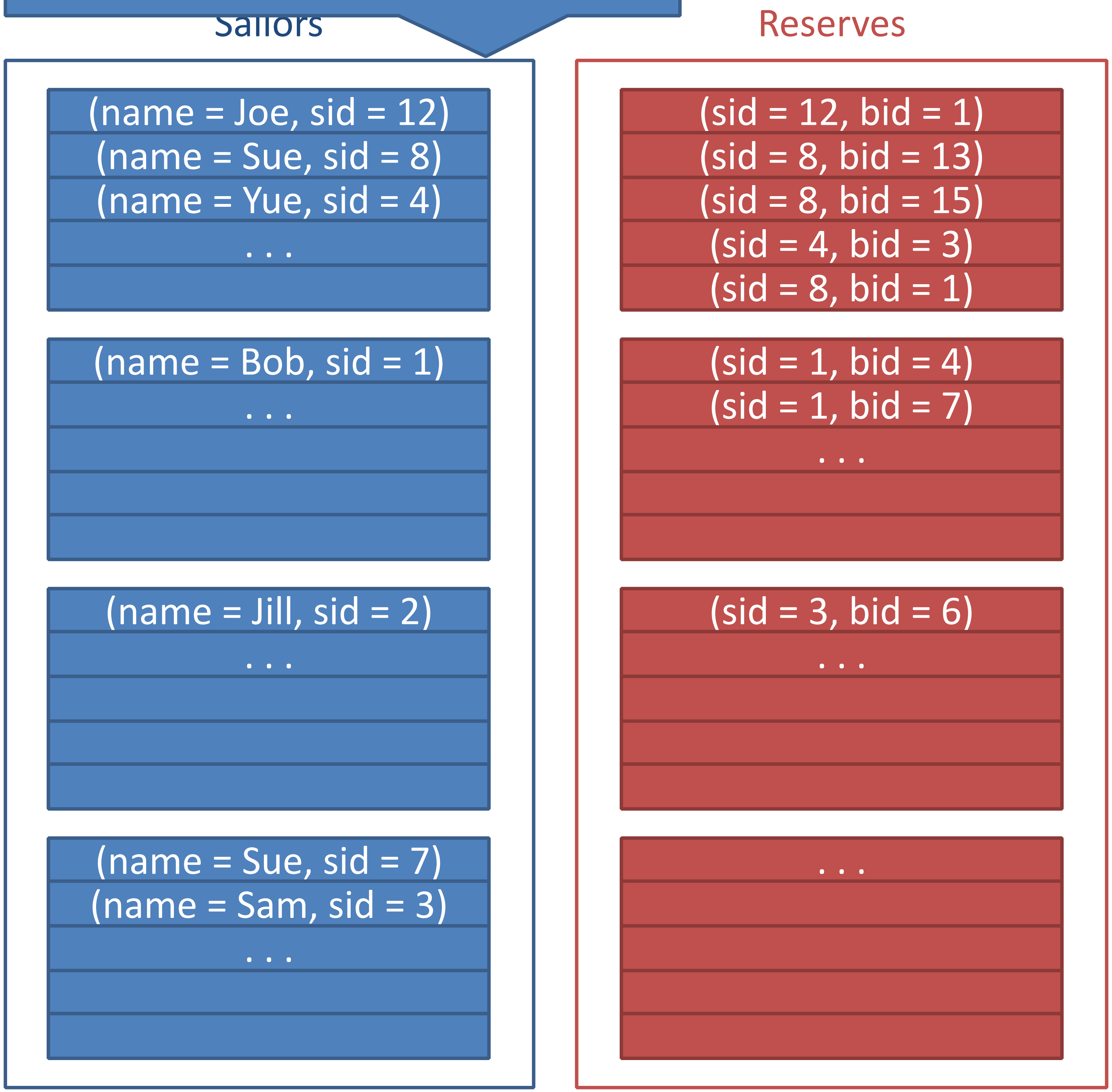
Partition S and R using same hash fn, then collect same partitions

**Steps:**

1. Partition S and R
2. Re-Hash, collect

Hash function: **sid mod 4**

# Hash-Join



Sailors

(name = Joe, sid = 12)
(name = Sue, sid = 8)
(name = Yue, sid = 4)
. . .

(name = Bob, sid = 1)
. . .

(name = Jill, sid = 2)
. . .

(name = Sue, sid = 7)
(name = Sam, sid = 3)
. . .

Reserves

(sid = 12, bid = 1)
(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

(sid = 1, bid = 4)
(sid = 1, bid = 7)
. . .

(sid = 3, bid = 6)
. . .

. . .

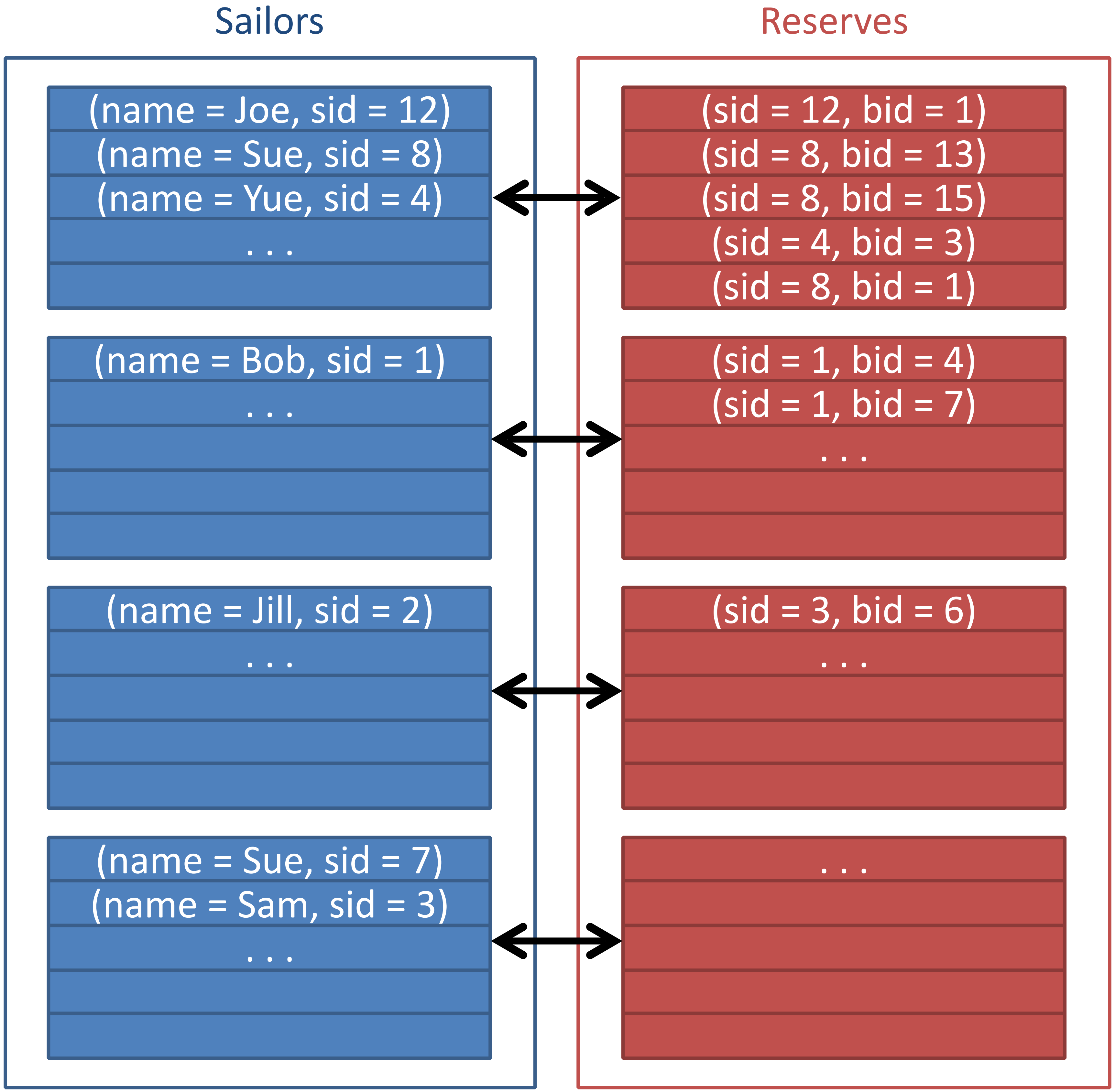
**Key idea:**

Partition S and R using same hash fn, then collect same partitions

**Steps:**

1. Partition S and R
2. Re-Hash, collect

# Hash-Join



| Sailors | Reserves |
|---|---|
| (name = Joe, sid = 12)<br>(name = Sue, sid = 8)<br>(name = Yue, sid = 4)<br>. . . | (sid = 12, bid = 1)<br>(sid = 8, bid = 13)<br>(sid = 8, bid = 15)<br>(sid = 4, bid = 3)<br>(sid = 8, bid = 1) |
| (name = Bob, sid = 1)<br>. . . | (sid = 1, bid = 4)<br>(sid = 1, bid = 7)<br>. . . |
| (name = Jill, sid = 2)<br>. . . | (sid = 3, bid = 6)<br>. . . |
| (name = Sue, sid = 7)<br>(name = Sam, sid = 3)<br>. . . | . . . |

**Key idea:**

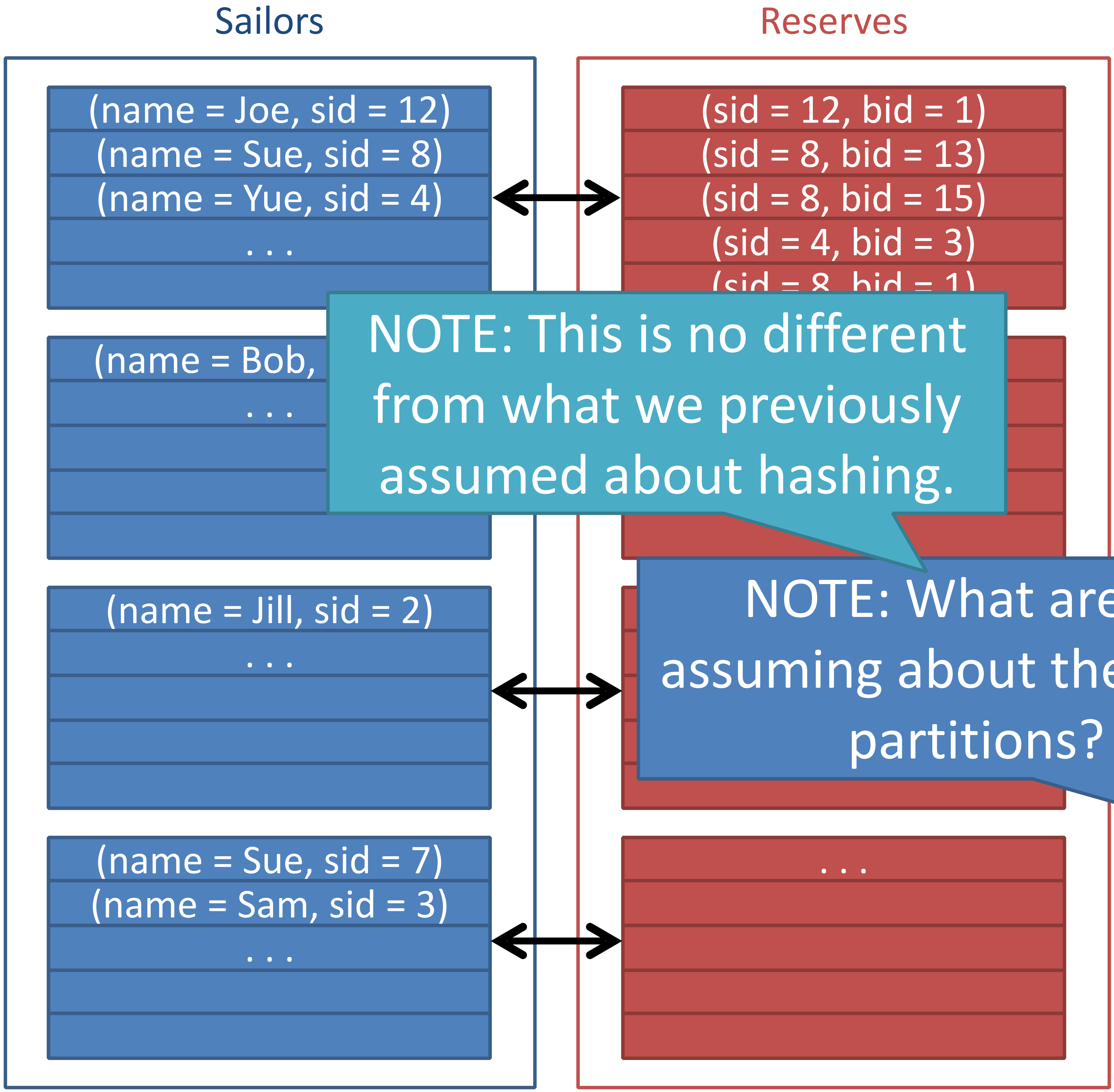
Partition S and R using same hash fn, then collect same partitions

**Steps:**

1. Partition S and R
2. Re-Hash, collect

**Notation: [S] == “# pages in S” ; |S| == “# tuples in S”**

# Hash-Join



Sailors

(name = Joe, sid = 12)
(name = Sue, sid = 8)
(name = Yue, sid = 4)
. . .

(name = Bob,
. . .

(name = Jill, sid = 2)
. . .

(name = Sue, sid = 7)
(name = Sam, sid = 3)
. . .

Reserves

(sid = 12, bid = 1)
(sid = 8, bid = 13)
(sid = 8, bid = 15)
(sid = 4, bid = 3)
(sid = 8, bid = 1)

. . .

NOTE: This is no different from what we previously assumed about hashing.

NOTE: What are we assuming about the size of partitions?

**Key idea:**

Partition S and R using same hash fn, then collect same partitions

**Steps:**

1. Partition S and R

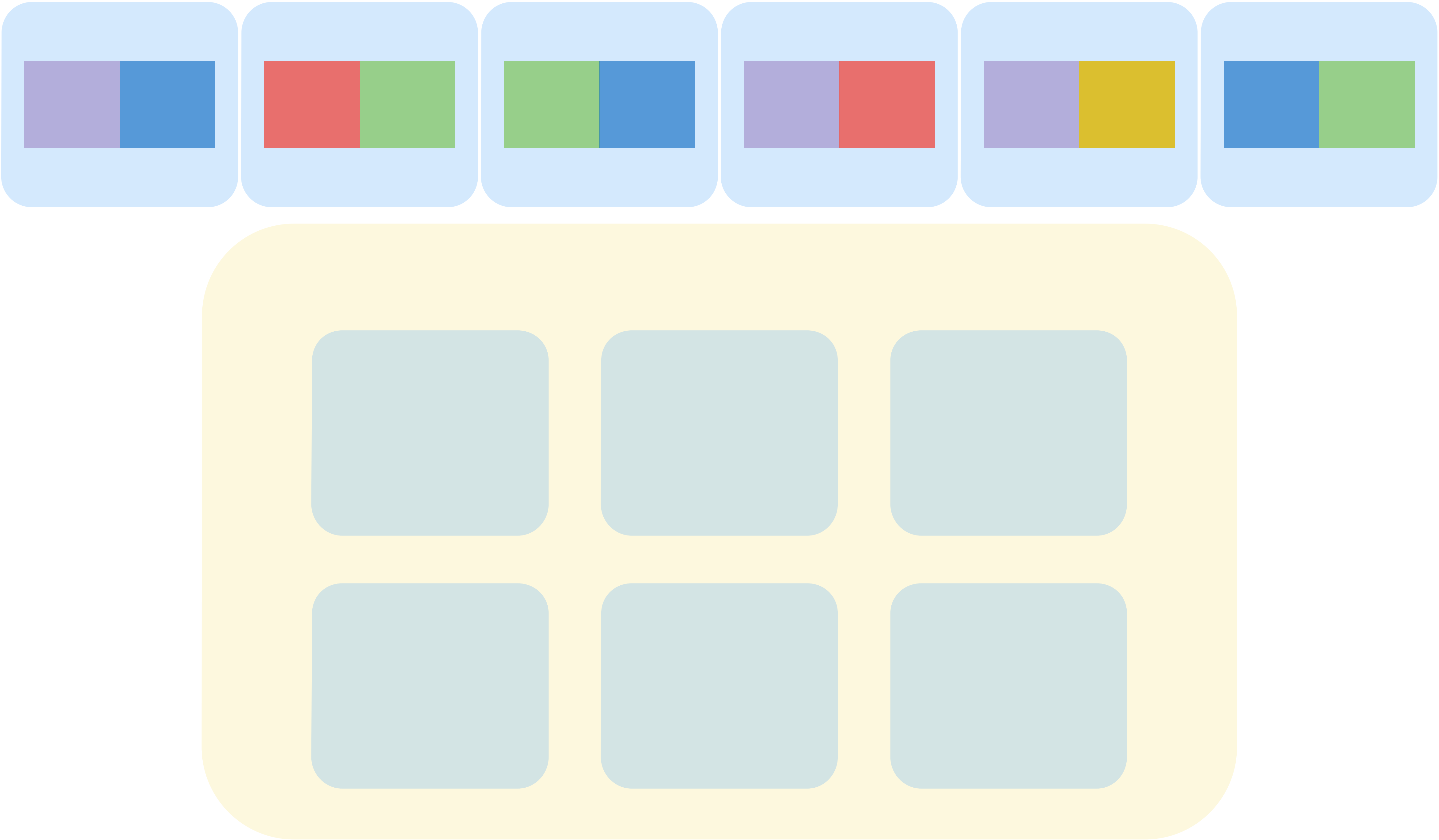
Hash, collect

3([S] + [R])

Partition: 2([S]+[R])

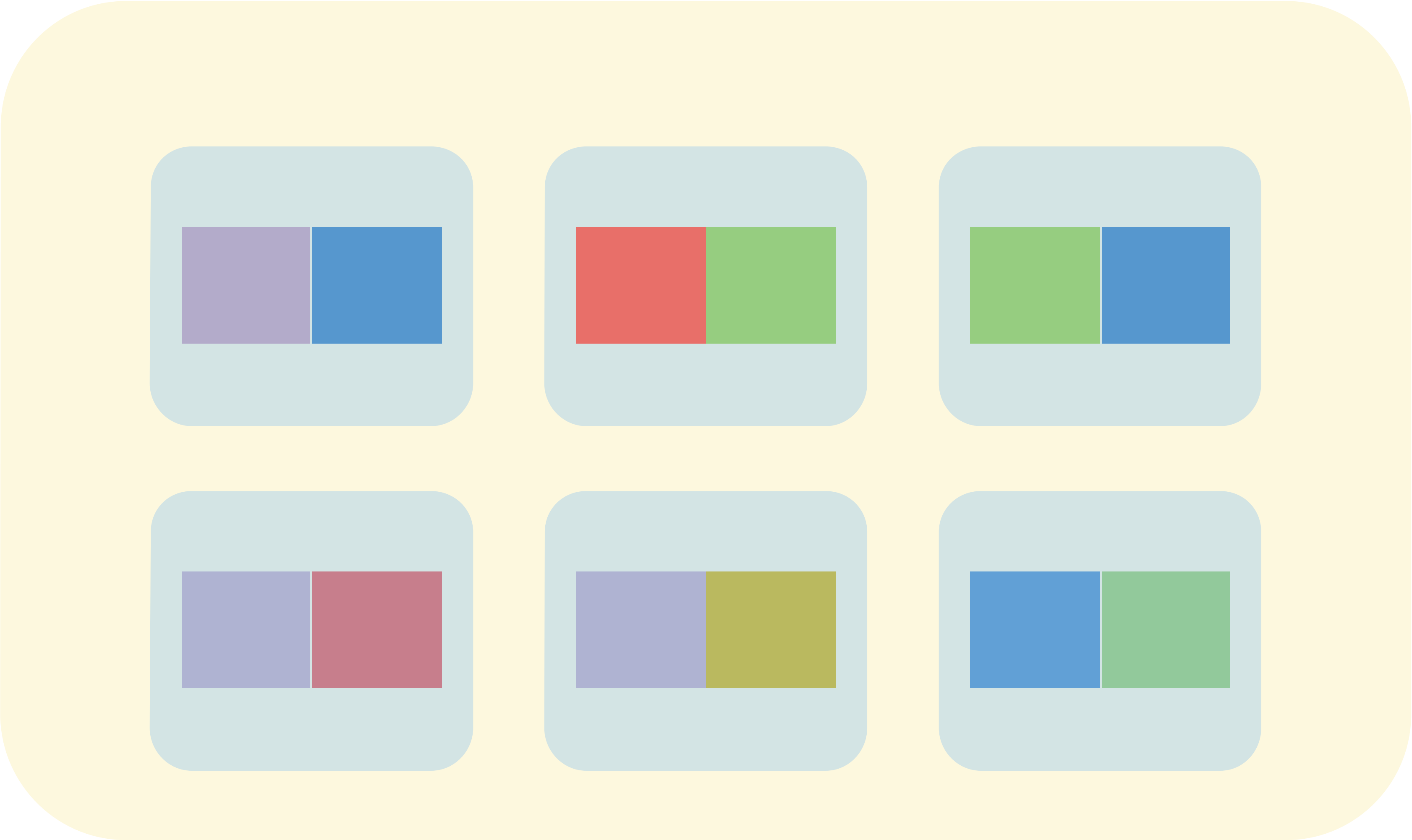
Re-Hash: [S]+[R]

# Hybrid Hashing



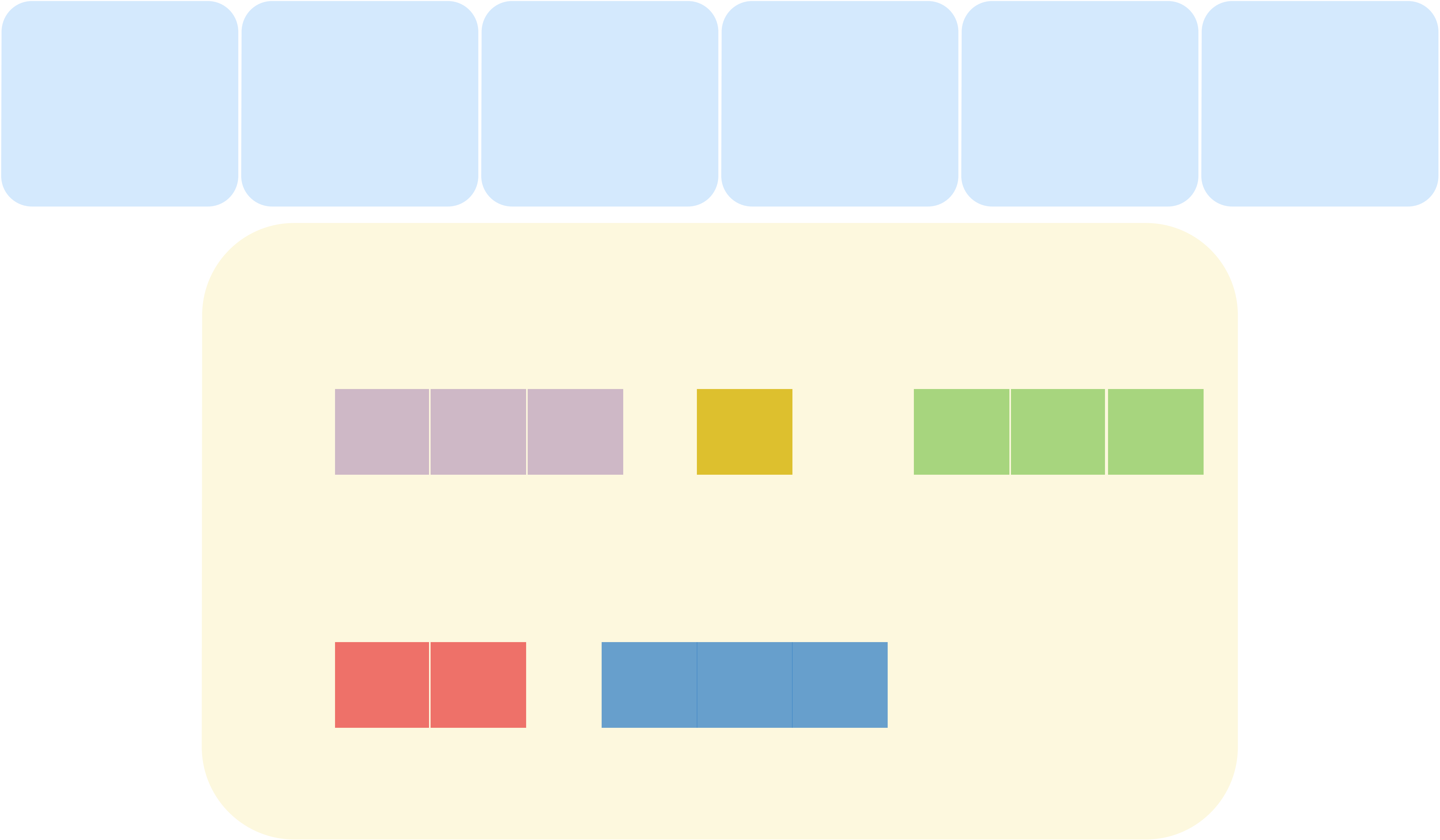
N <= B : Load all data into memory and hash

# Hybrid Hashing



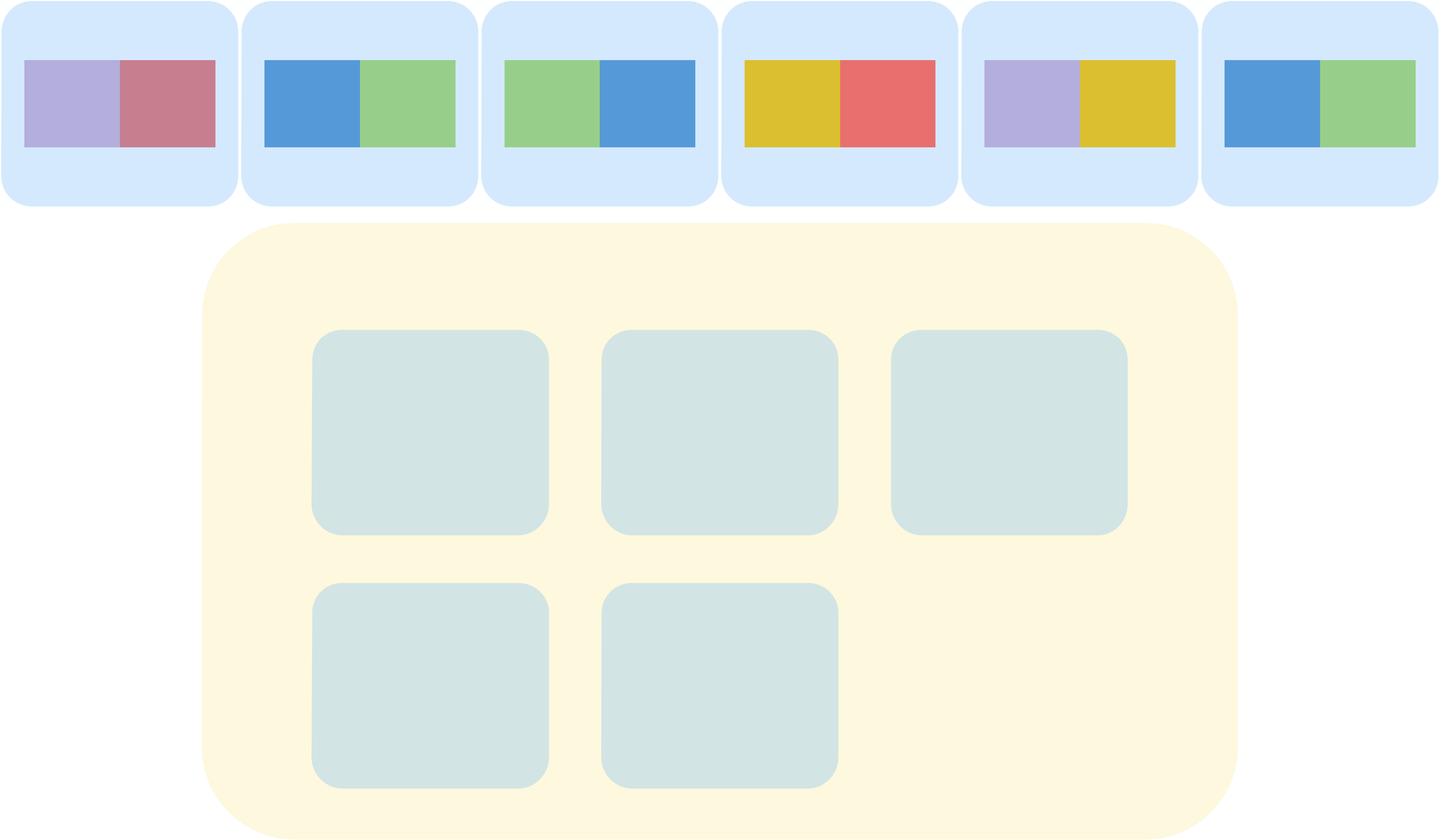
N <= B : Load all data into memory and hash

# Hybrid Hashing



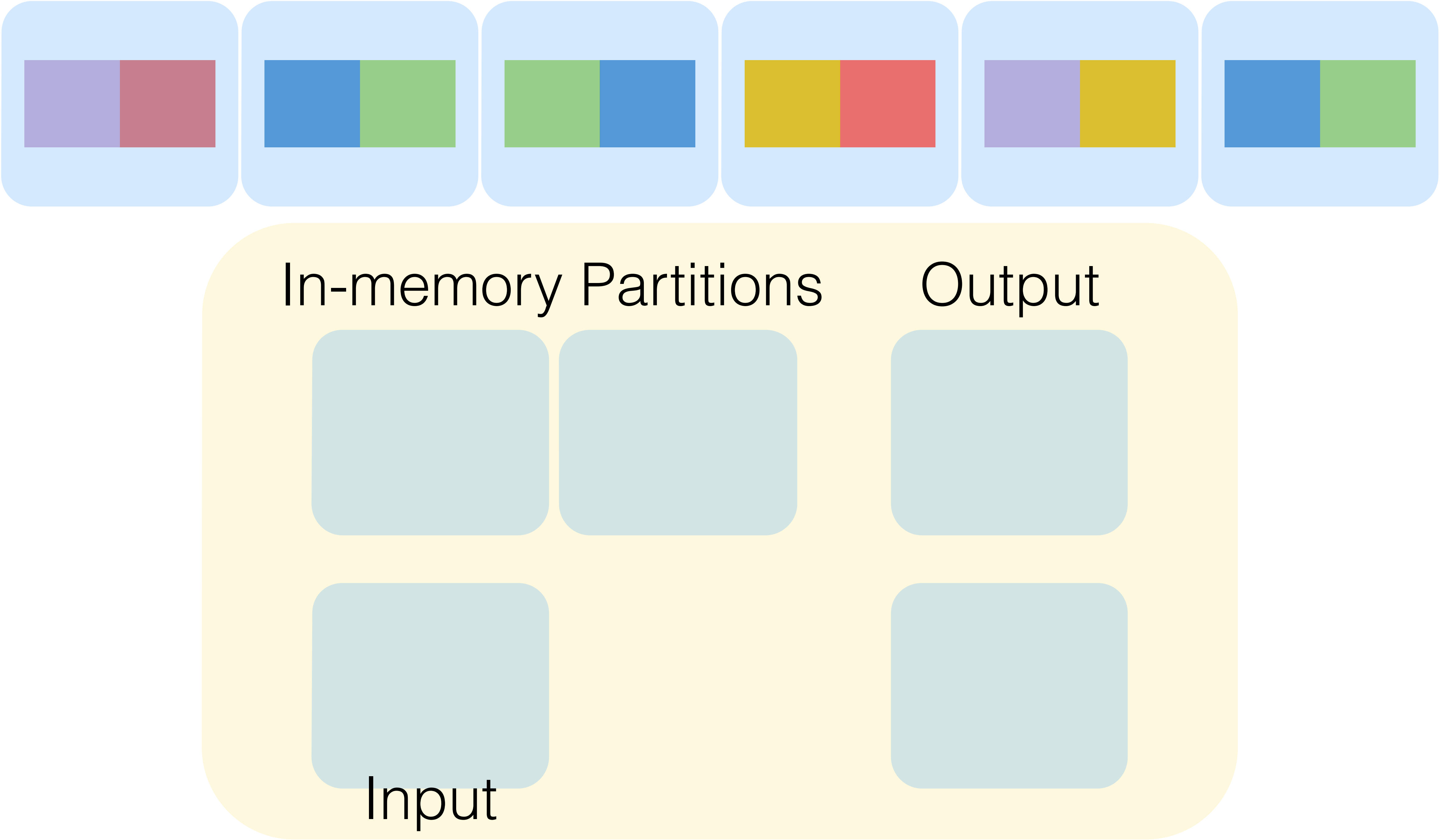
N <= B : Load all data into memory and hash

# Hybrid Hashing



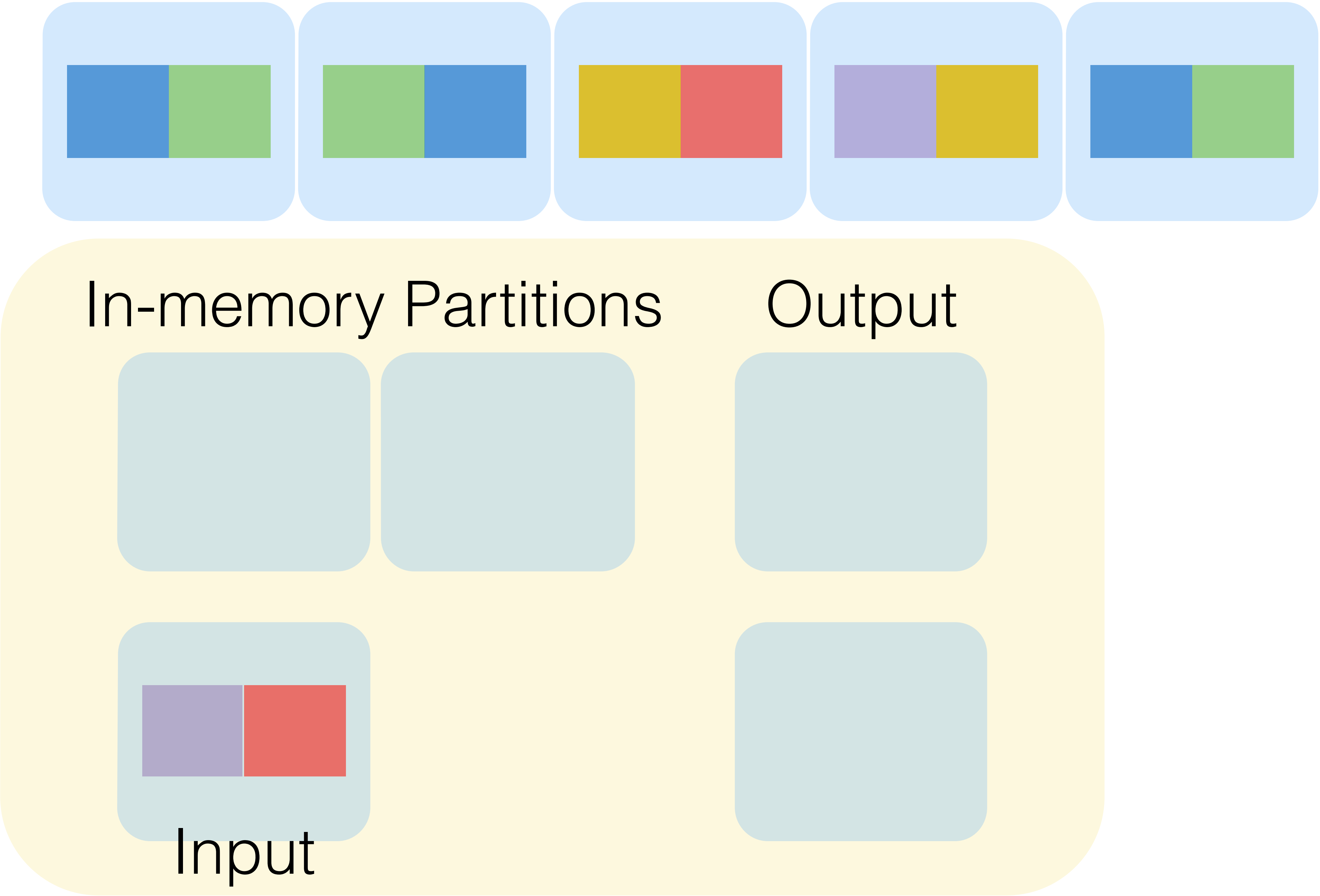
N >= B : Must perform at least 2 passes

# Hybrid Hashing



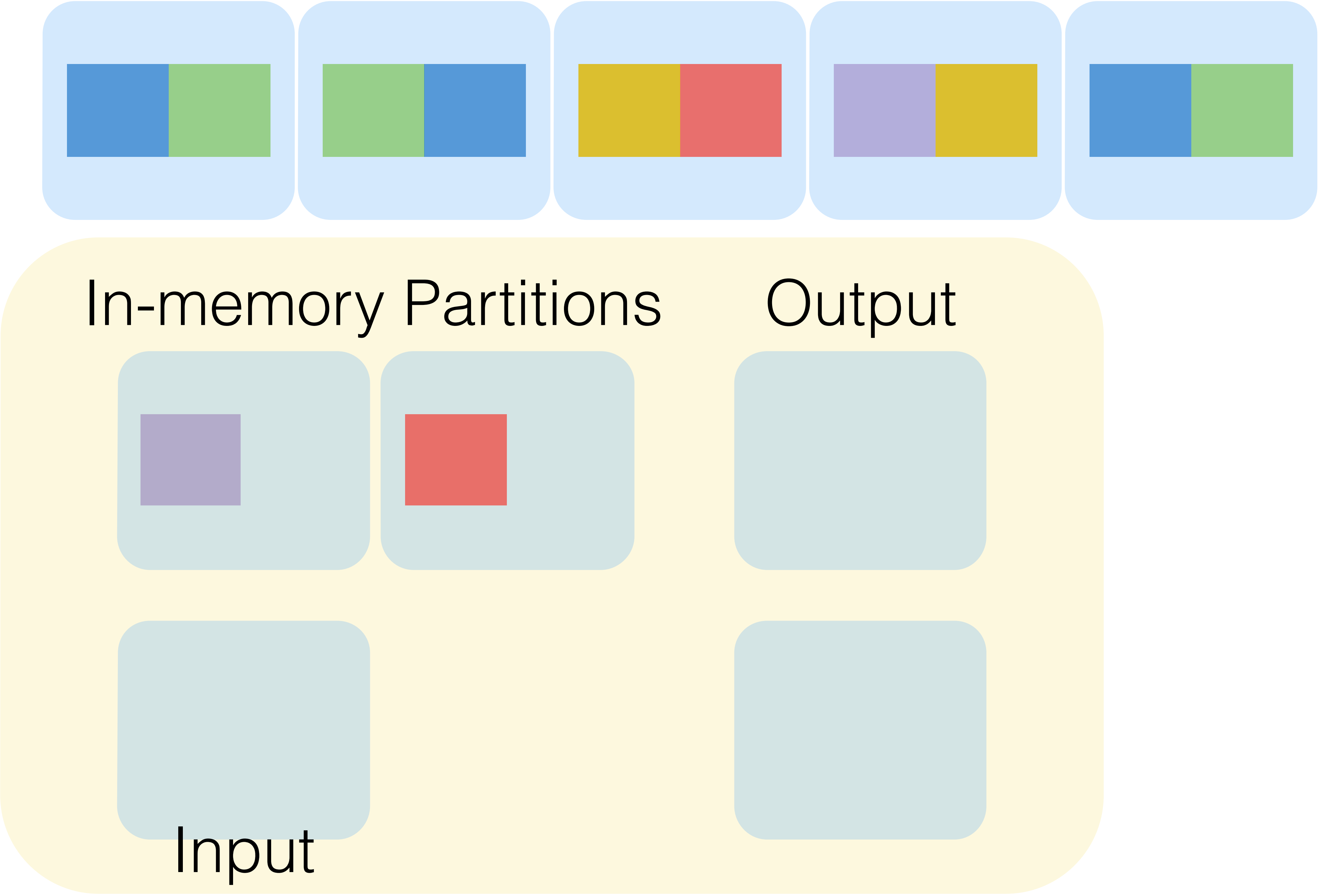
Solution: Keep some partitions in memory

# Hybrid Hashing



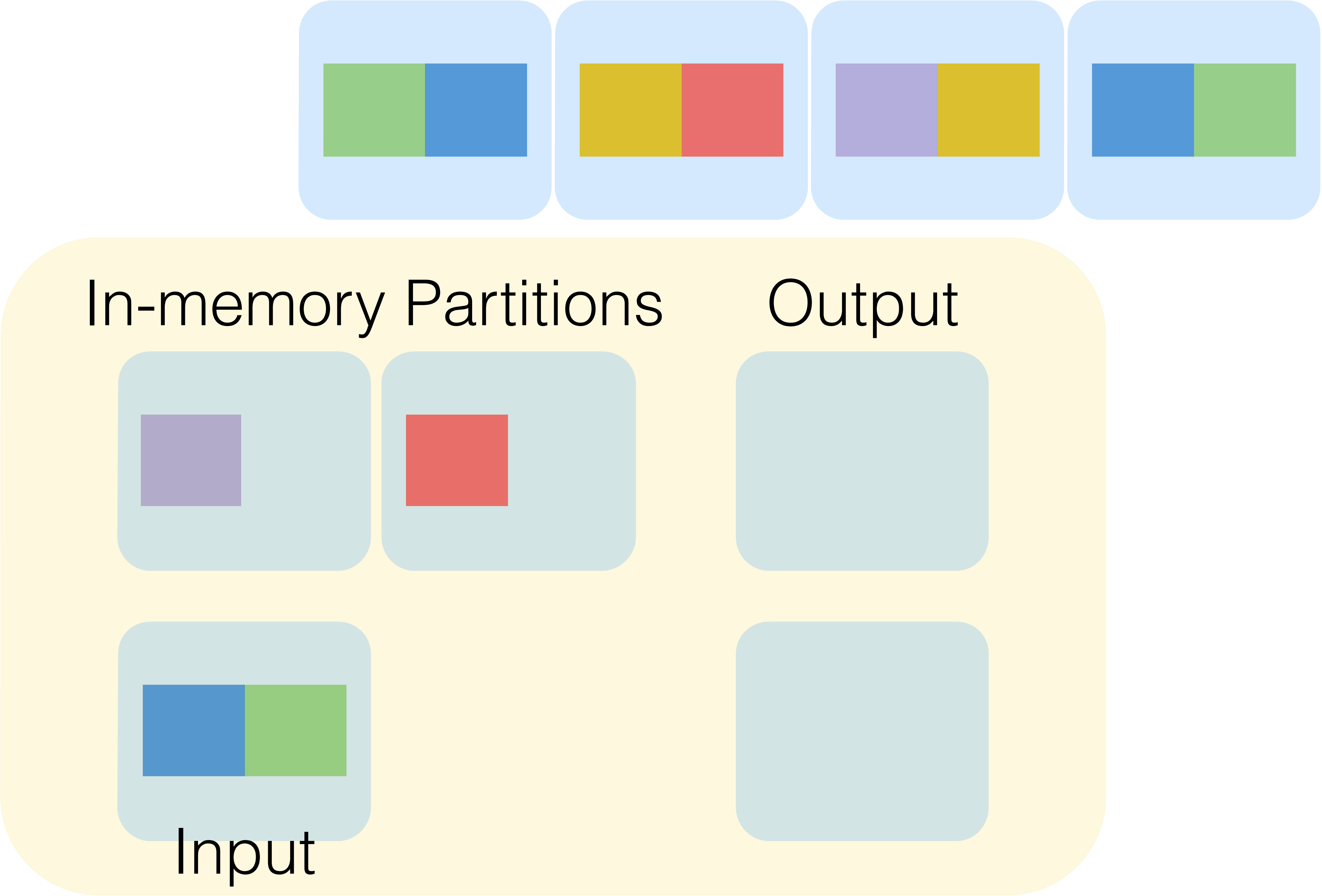
Solution: Keep some partitions in memory

# Hybrid Hashing



Solution: Keep some partitions in memory

# Hybrid Hashing



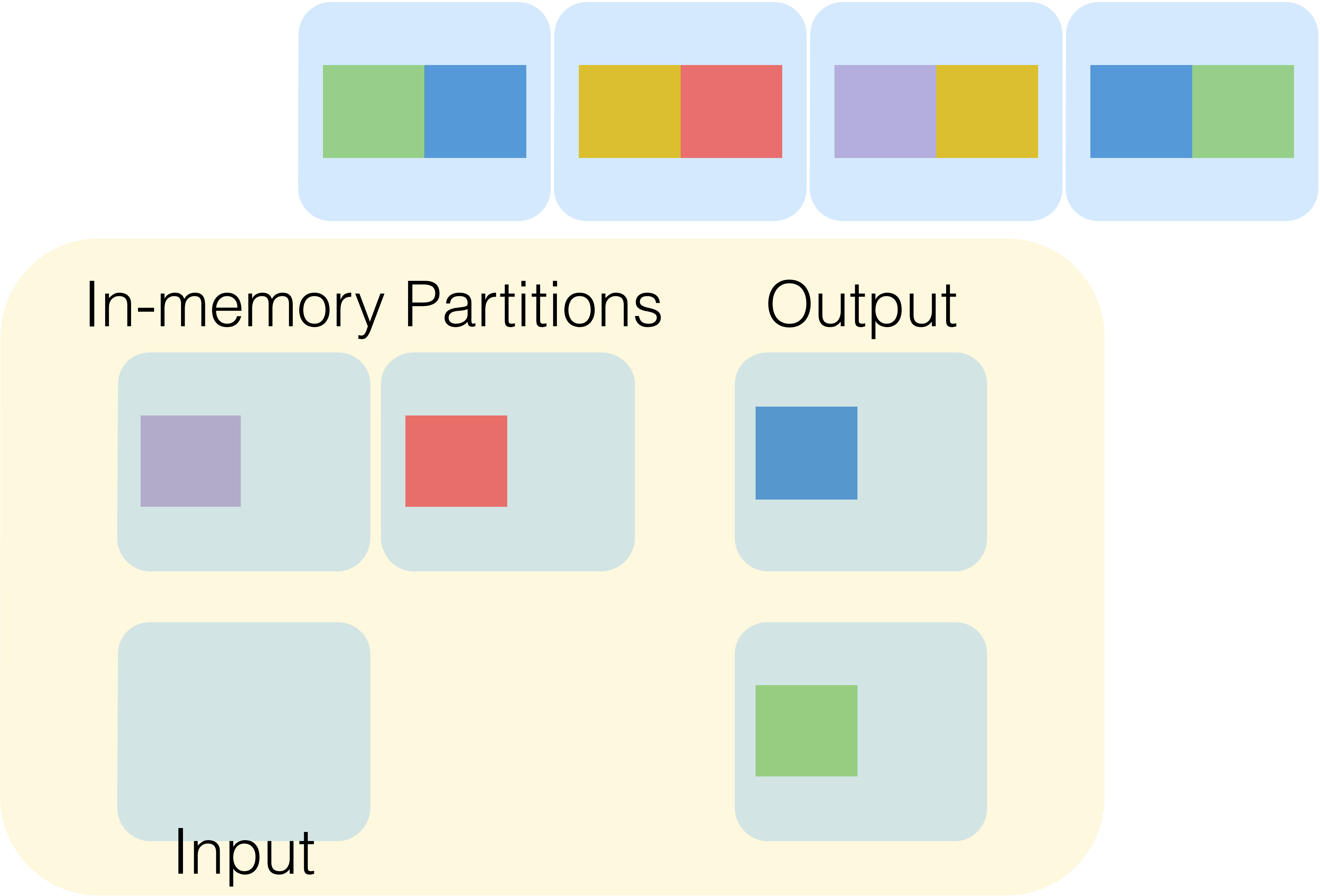
In-memory Partitions

Output

Input

Solution: Keep some partitions in memory

# Hybrid Hashing



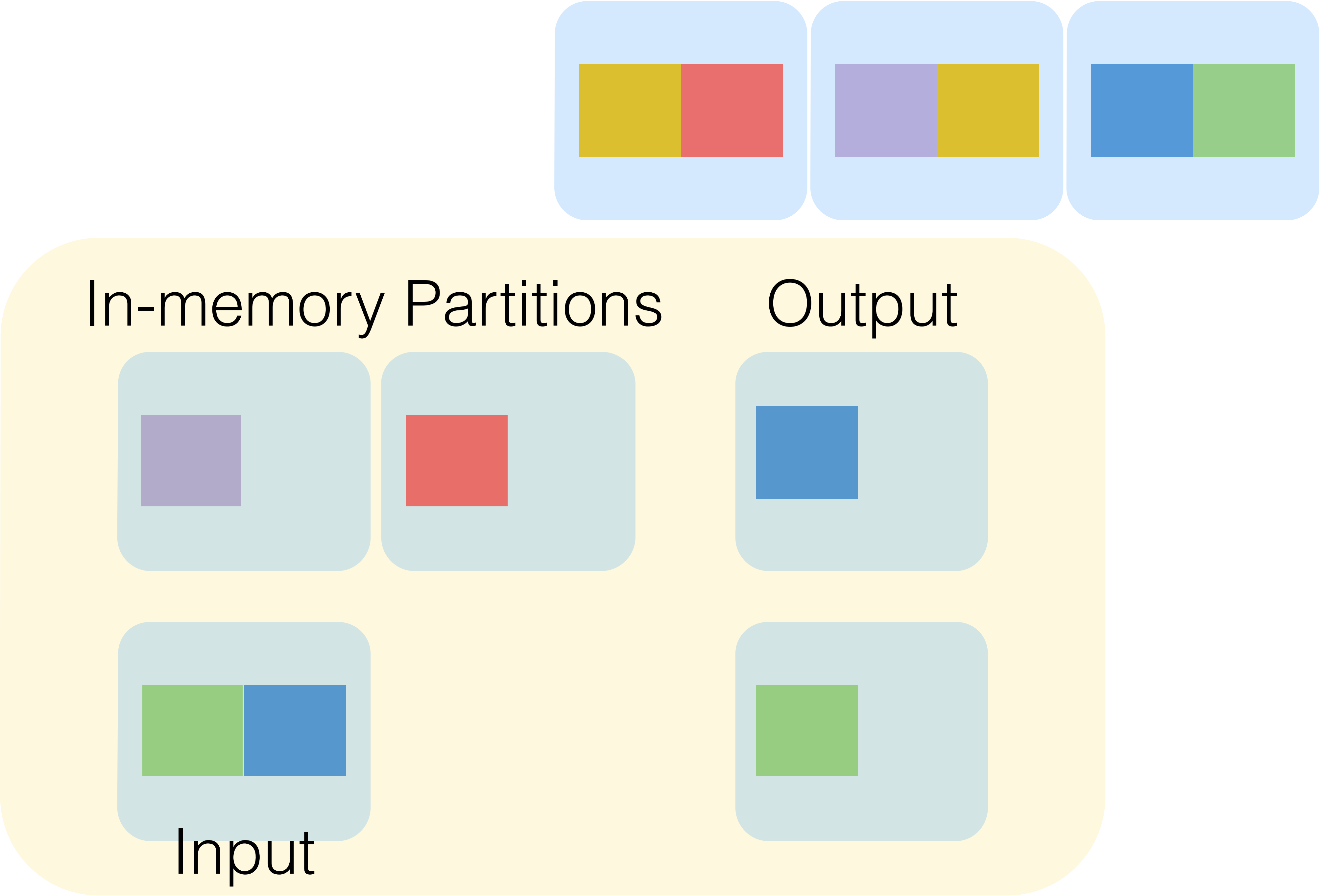
In-memory Partitions

Output

Input

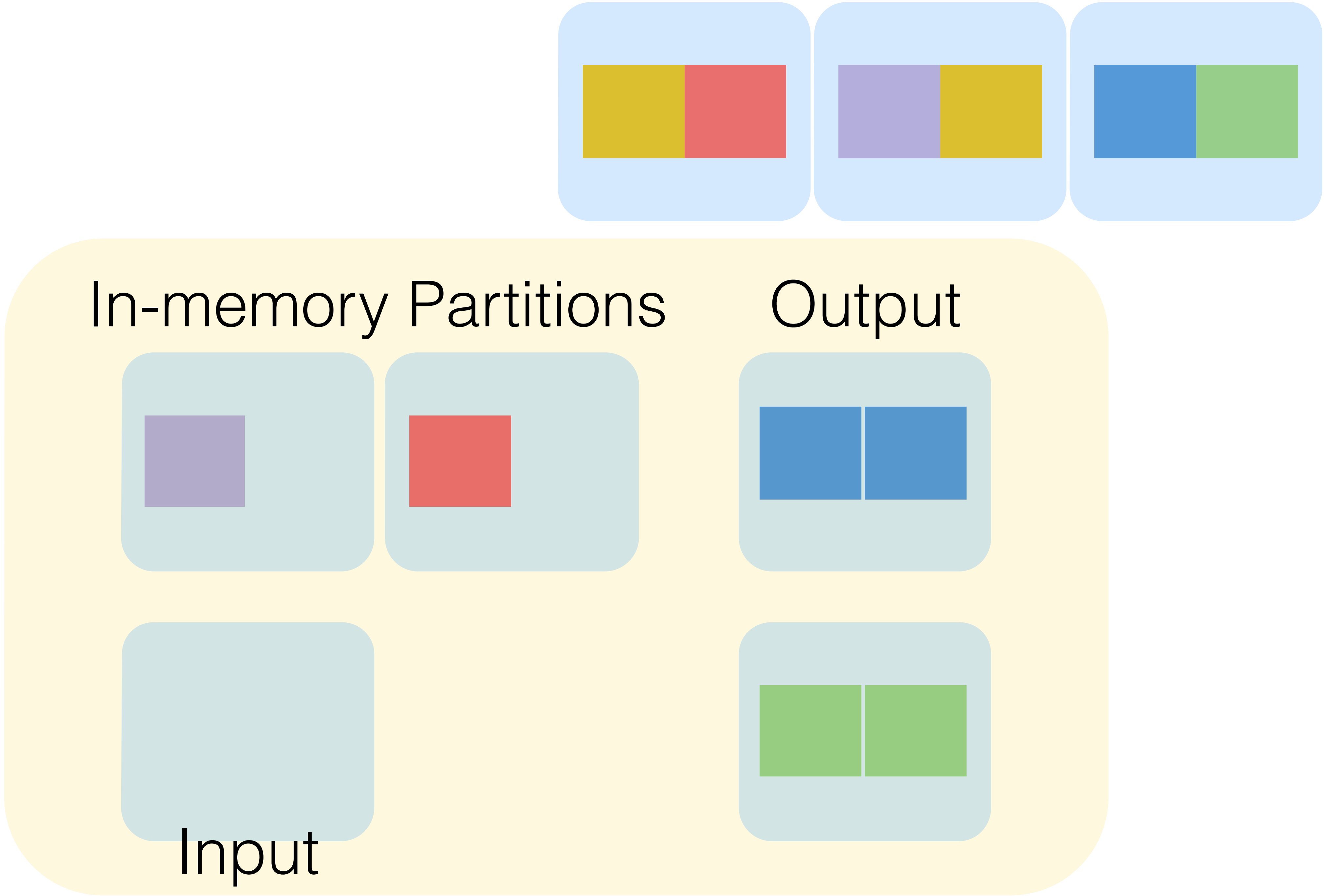
Solution: Keep some partitions in memory

# Hybrid Hashing



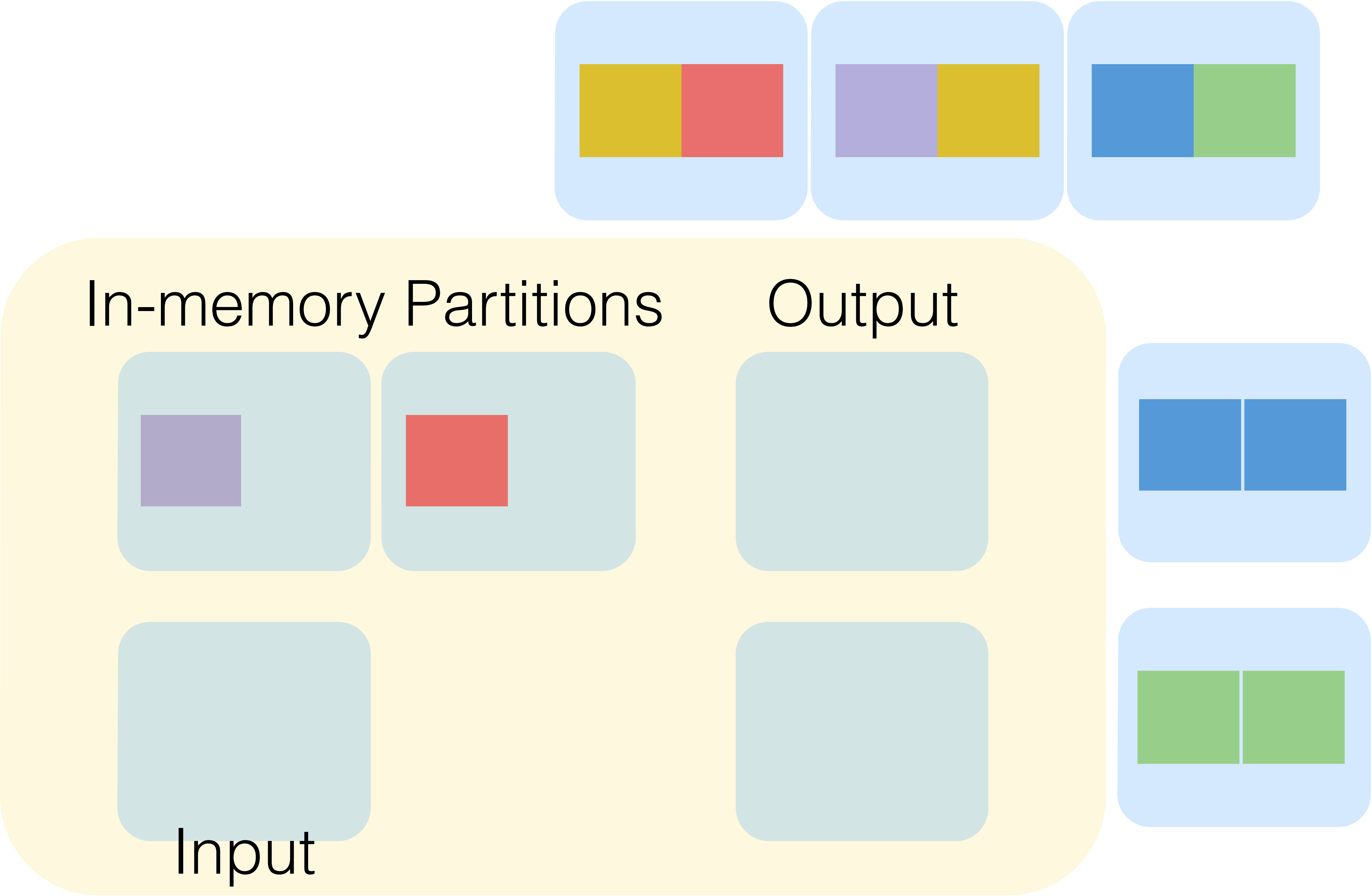
Solution: Keep some partitions in memory

# Hybrid Hashing



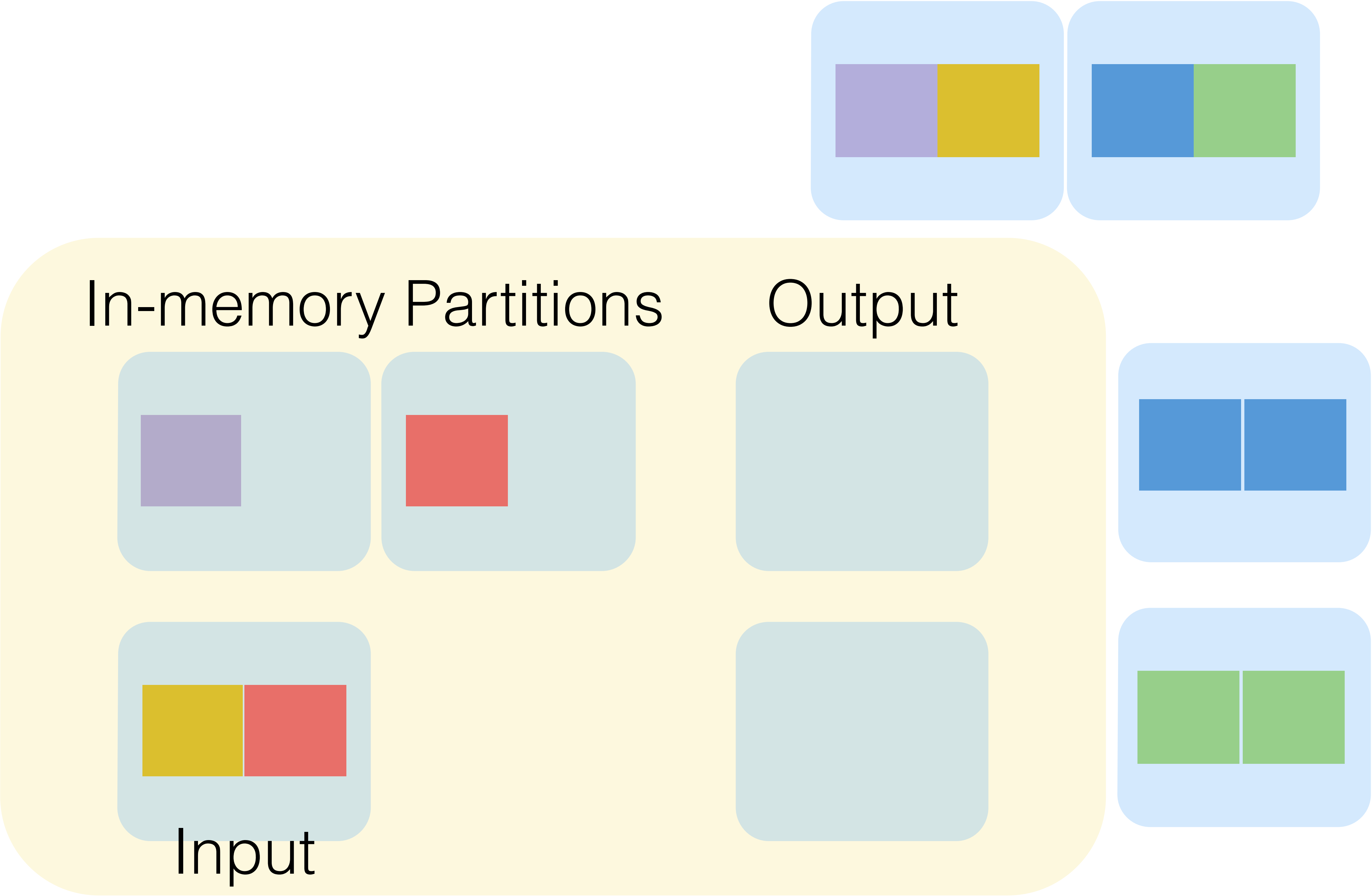
Solution: Keep some partitions in memory

# Hybrid Hashing



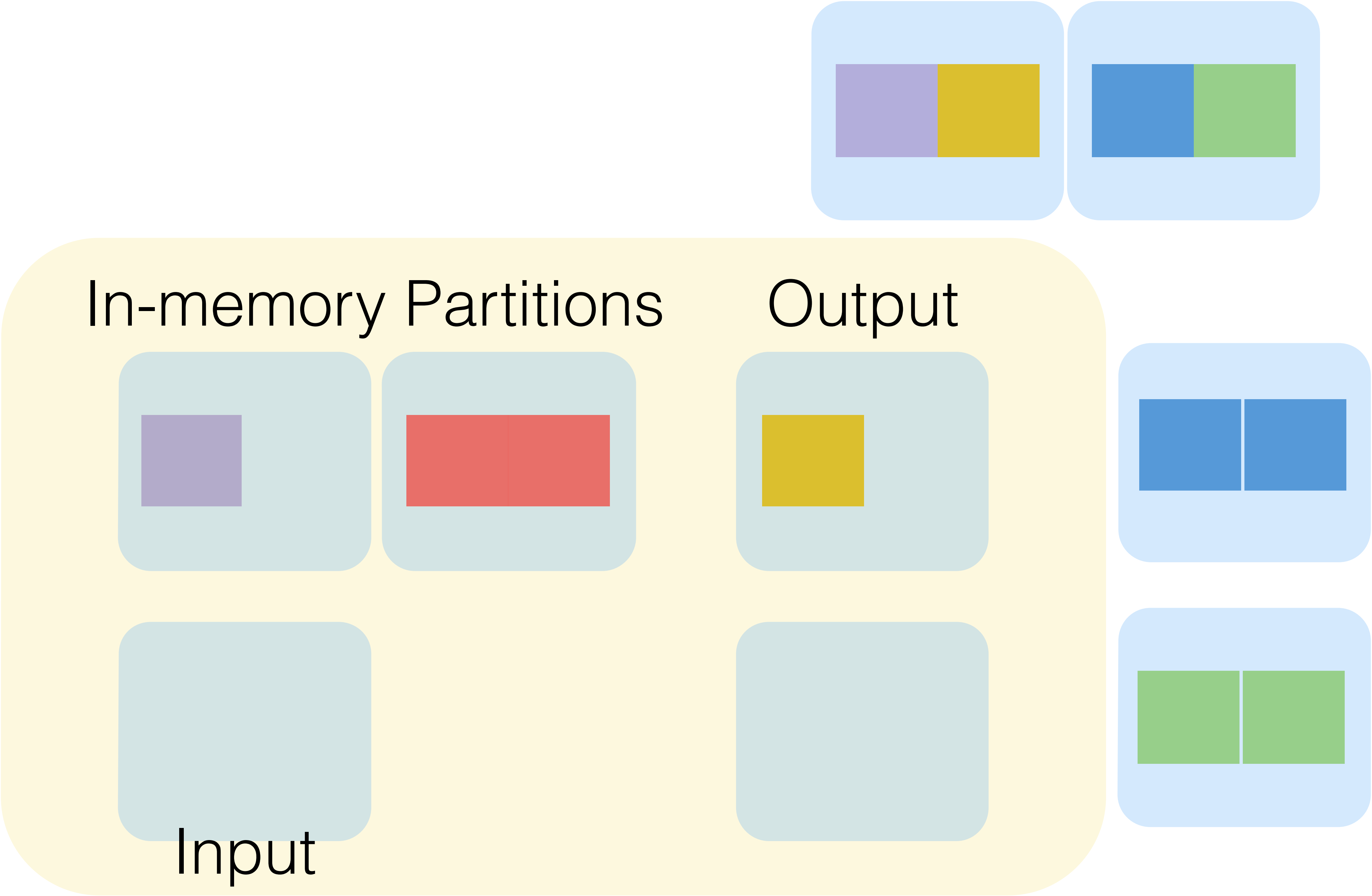
Solution: Keep some partitions in memory

# Hybrid Hashing



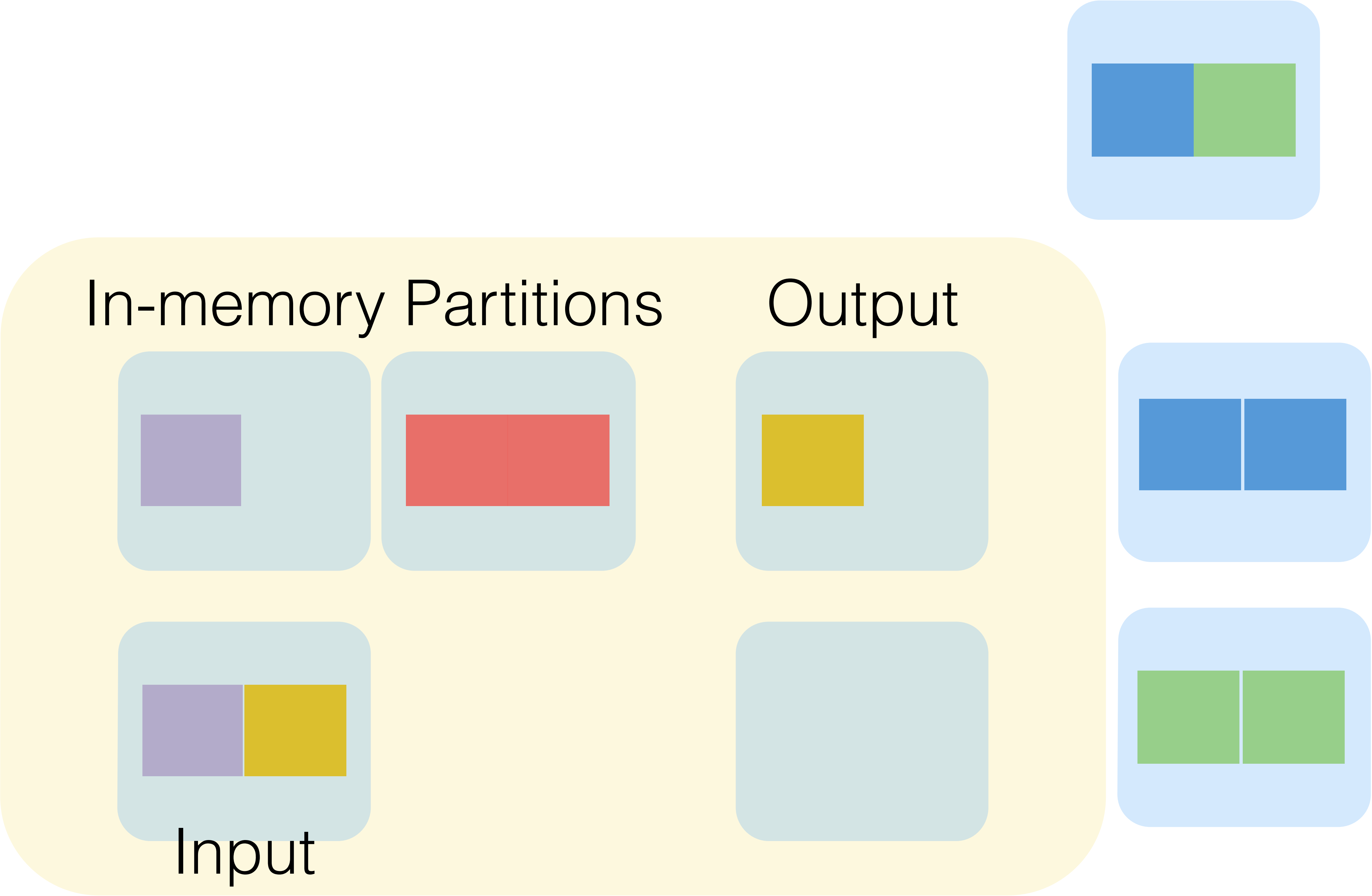
Solution: Keep some partitions in memory

# Hybrid Hashing



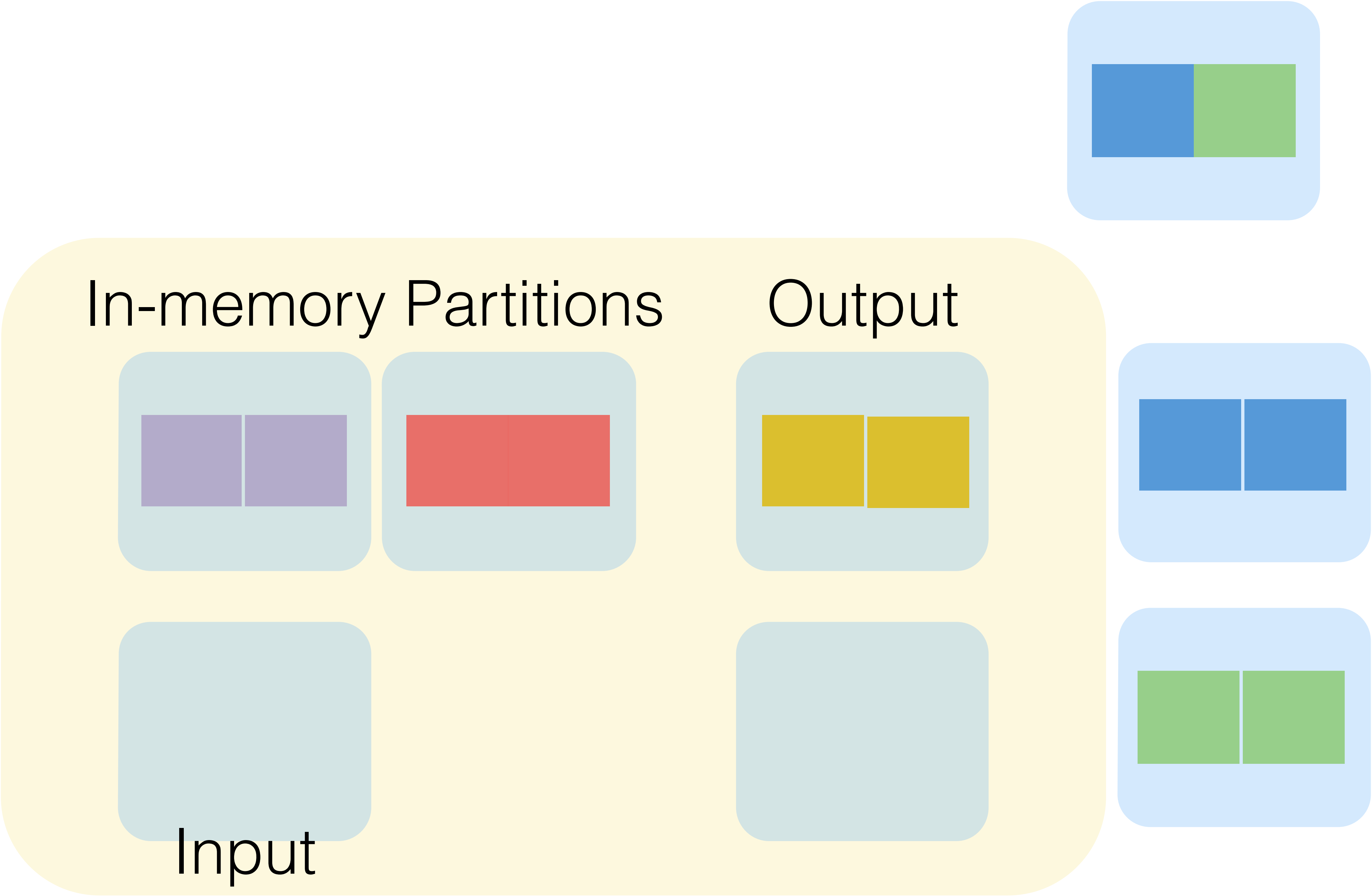
Solution: Keep some partitions in memory

# Hybrid Hashing



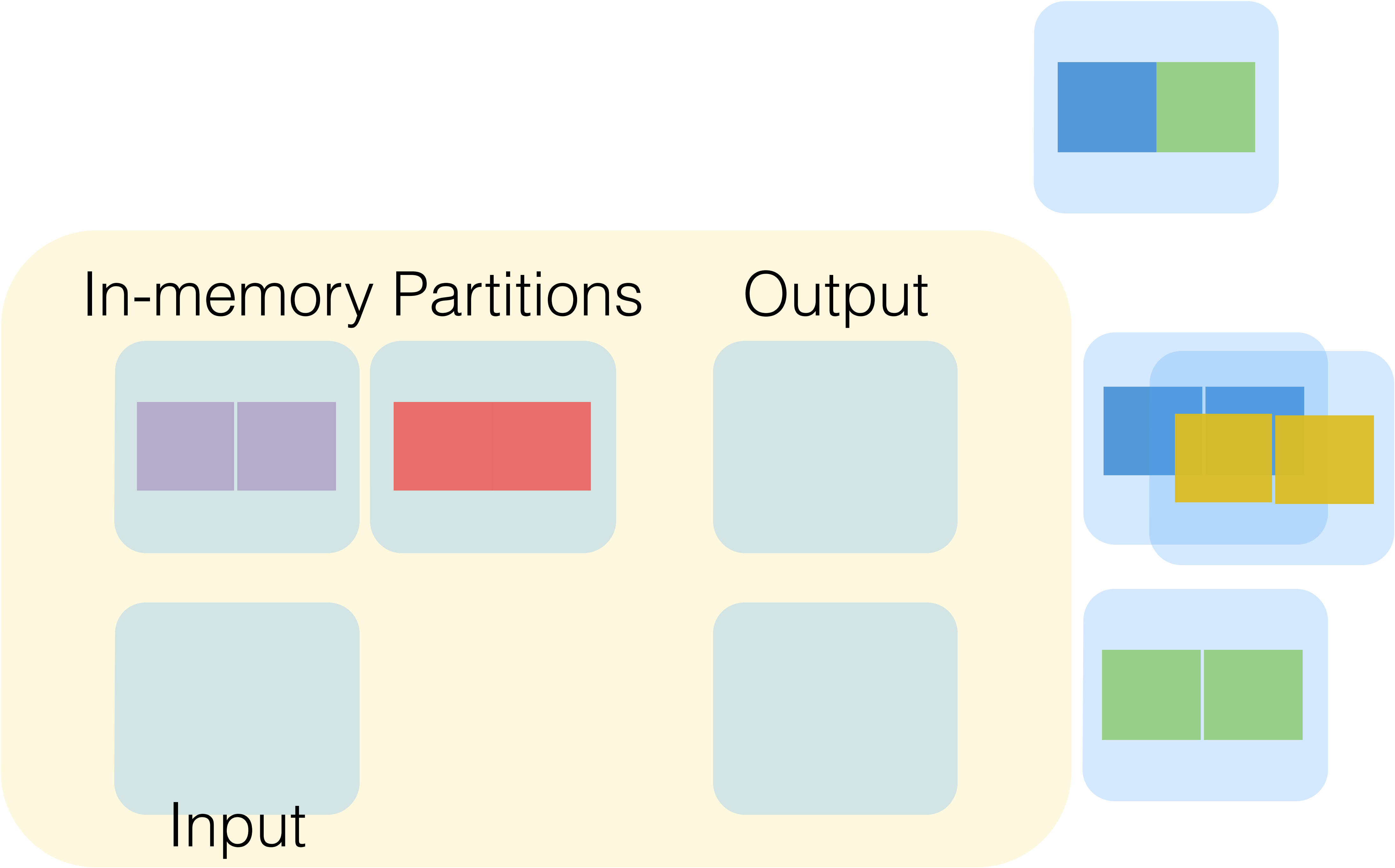
Solution: Keep some partitions in memory

# Hybrid Hashing



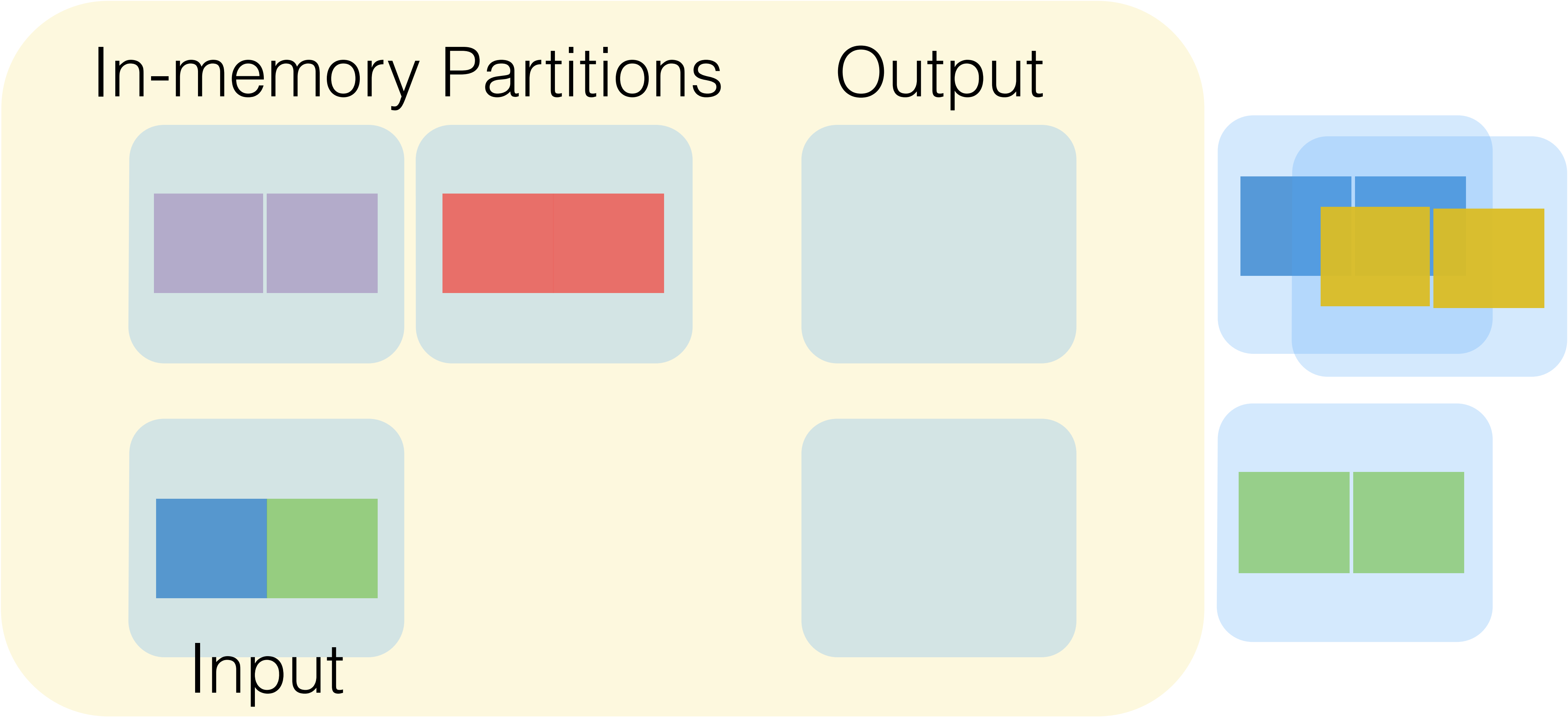
Solution: Keep some partitions in memory

# Hybrid Hashing



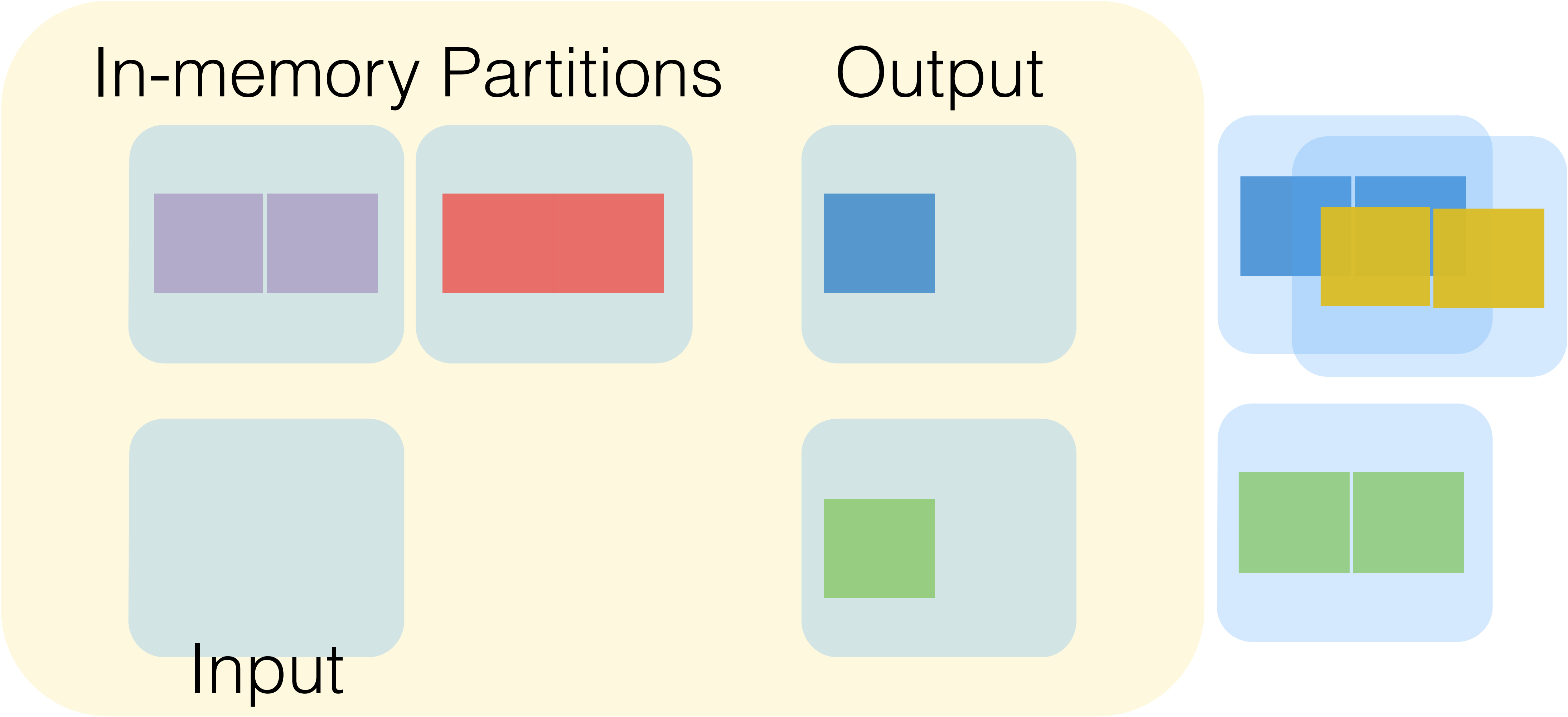
Solution: Keep some partitions in memory

# Hybrid Hashing



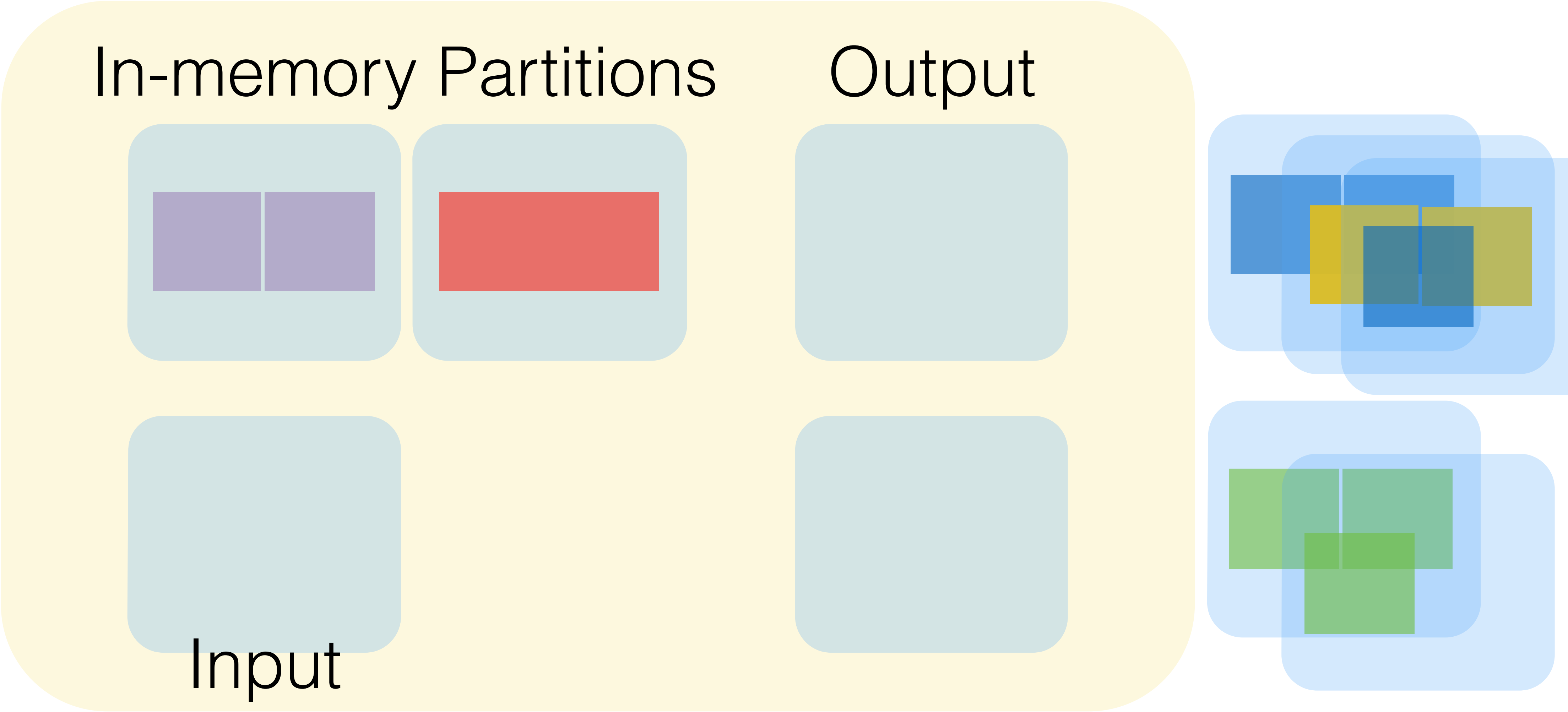
Solution: Keep some partitions in memory

# Hybrid Hashing



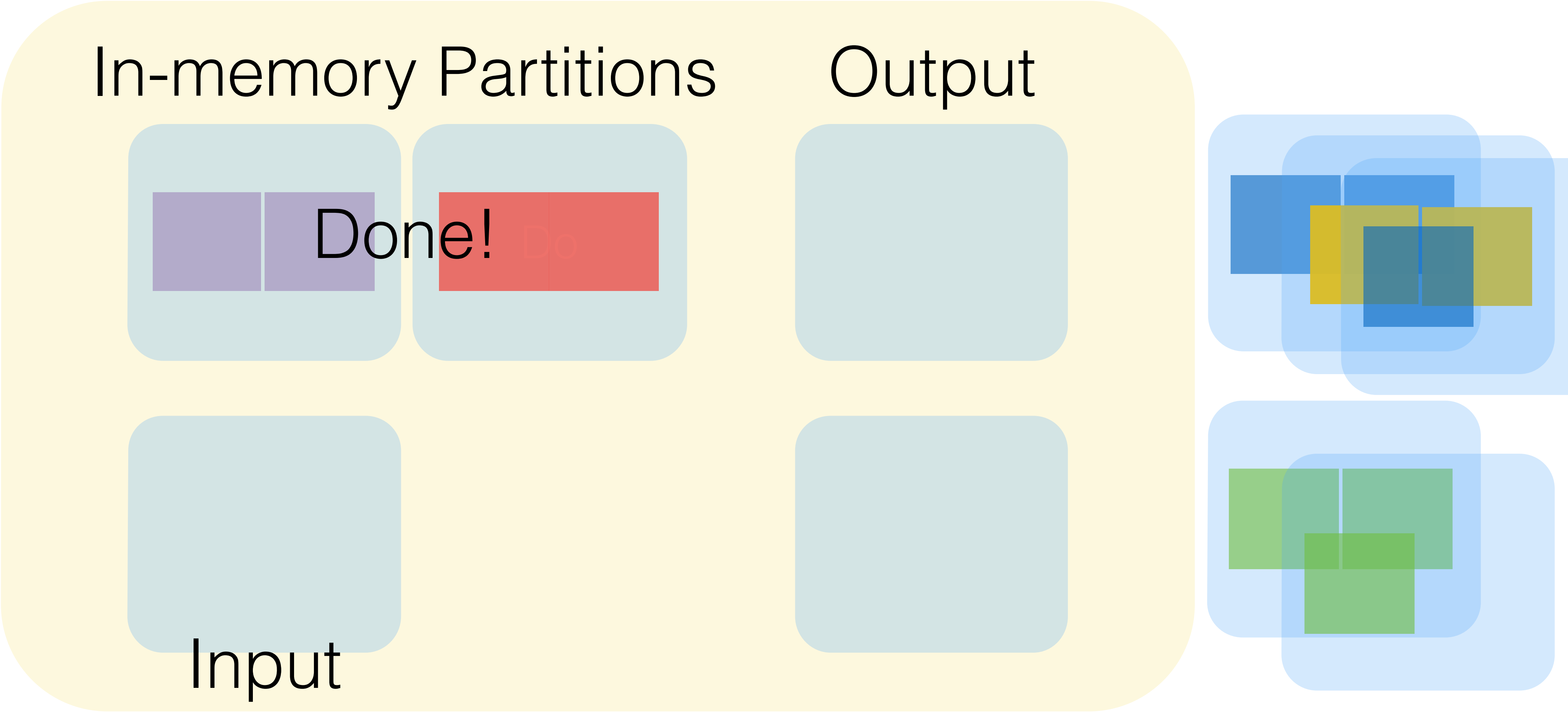
Solution: Keep some partitions in memory

# Hybrid Hashing



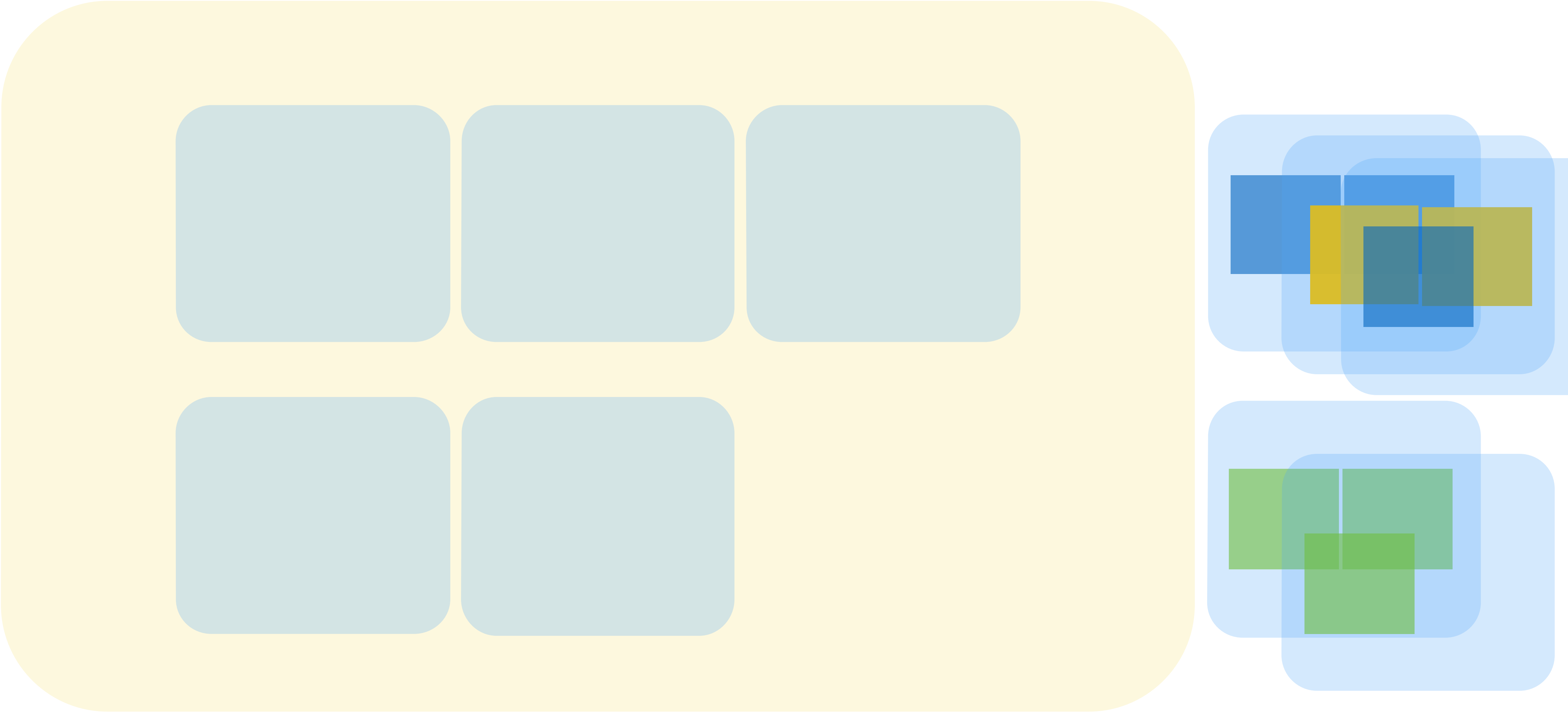
Solution: Keep some partitions in memory

# Hybrid Hashing



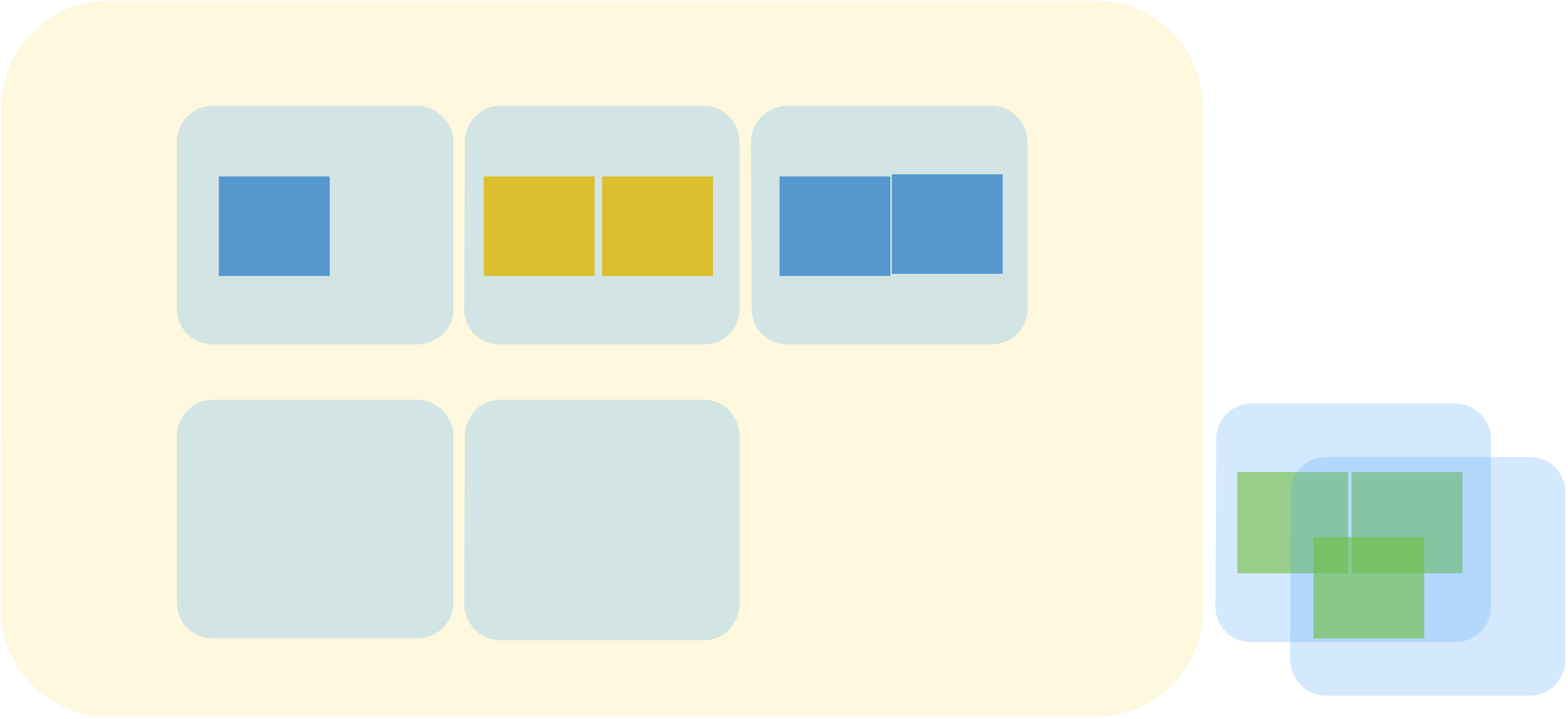
Save cost of reading those partitions from memory.

# Hybrid Hashing



Read other partitions in memory.

# Hybrid Hashing



# Join Cheatsheet

**Notation: [S] == "# pages in S" ;**
**|S| == "# tuples in S"**

- Chunk nested loop join
  - Take **k pages** of S and match with each page of R.
  - Total Cost: [S] + ([S] / k)*[R]
- Sort merge join
  - Sort S and R **on join column**, then merge them!
  - Total Cost: ~5([S] + [R])
- Hash join
  - Partition S and R using same hash fn, then collect same partitions
  - Total Cost: ~3([S] + [R])
    - Assuming len(partition) ≤ B pages

# When is a chunk-nested loops join the best?

# When is a chunk-nested loops join the best?

- Not using an equality predicate
- Join is just a cross product

# When is a sort-merge join the best?

# When is a sort-merge join the best?

- Skewed input data
  - Could prompt recursive hashing
- Want sorted output or input already sorted

# When is a hash-join the best?

# When is a hash-join the best?

- One table is large, and the other small (can keep perform hybrid-hashing)

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page and [S] holds 50 tuples per page.

How many disk reads are needed to perform a Simple Nested Loops join?

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page and [S] holds 50 tuples per page.

How many disk reads are needed to perform a Simple Nested Loops join?

Using S as the outer relation yields the lowest I/O count.

(#tuples in S)*[R]*[S] + [S] = 50*100*50 + 50 = 250050 I/O's

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page
and [S] holds 50 tuples per page.

How many disk reads are needed to perform Chunk Nested Loops Join?

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page
and [S] holds 50 tuples per page.

How many disk reads are needed to perform Chunk Nested Loops Join?

(# of pages in smaller relation) + ((# of pages in smaller relation) / (# of pages in memory - 2 for I/O)) * (# of pages in larger relation)

= 50 + (50/10) * (100) = 550

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page and [S] holds 50 tuples per page.

How about a Sort Merge Join? (Assume the join column is unique in both tables and that both tables are sorted in 2 passes)

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page and [S] holds 50 tuples per page.

How about a Sort Merge Join? (Assume the join column is unique in both tables and that both tables are sorted in 2 passes)

5[R] + 5[S] = 750 I/O’s

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page and [S] holds 50 tuples per page.

How about a Hash Join? (Assume no recursive partitioning)

We have 12 pages of memory, and we want to join two tables [R] and [S] where [R] is 100 pages and [S] is 50 pages. [R] holds 100 tuples per page
and [S] holds 50 tuples per page.

How about a Hash Join? (Assume no recursive partitioning)

Partitioning Phase: 2([R]+[S])

Matching Phase: [R]+[S]

Total = 3([R] + [S]) = 3* 150 = 450 I/O’s

# Midterm Practice

S with 11,000 pages, 2000 records per page
R with 500 pages, 500 records per page
Memory: 100 pages
How many I/O's are required to perform a simple, "unoptimized" sort-merge join?

# Midterm Practice

S with 11,000 pages, 2000 records per page
R with 500 pages, 500 records per page
Memory: 100 pages
How many I/O's are required to perform a simple, "unoptimized" sort-merge join?

cost to sort S + cost to sort R + cost to merge

# Midterm Practice

S with 11,000 pages, 2000 records per page
R with 500 pages, 500 records per page
Memory: 100 pages
How many I/O's are required to perform a simple, "unoptimized" sort-merge join?

cost to sort S + cost to sort R + cost to merge
2*3*11,000 + 2*2*500 + 11000 + 500 = 79500 I/Os