CS186 Discussion #3

(Joins, Heap Files)

Joins

```
SELECT <columns>
FROM <tables>
WHERE <predicate>
[GROUP BY <column list>
[HAVING <predicate>]];
```

Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
year_released, genre)

```
SELECT *
FROM Artists, Albums
WHERE Artists.artist_id =
Albums.artist_num;
```

Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
       year_released, genre)

```
SELECT *
FROM Artists A, Albums B
WHERE A.artist_id = B.artist_num;
```

```
Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
       year_released, genre)
```

Write a SQL expression for the following query:

The name of all songs with the genre “country” which have spent more than 2 weeks in the top 40.

```
Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
       year_released, genre)
```

Write a SQL expression for the following query:

The name of all songs with the genre “country” which have spent more than 2 weeks in the top 40.

```
SELECT Songs.song_name FROM Albums,
Songs WHERE Songs.album_num =
Albums.album_id AND Albums.genre =
'country' AND Songs.weeks_in_top_40 > 2;
```

```
Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
       year_released, genre)
```

Write a SQL expression for the following query:

The number of albums released by each artist.

```
Songs(song_id, song_name, album_num, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums(album_id, album_name, artist_num,
       year_released, genre)
```

Write a SQL expression for the following query:

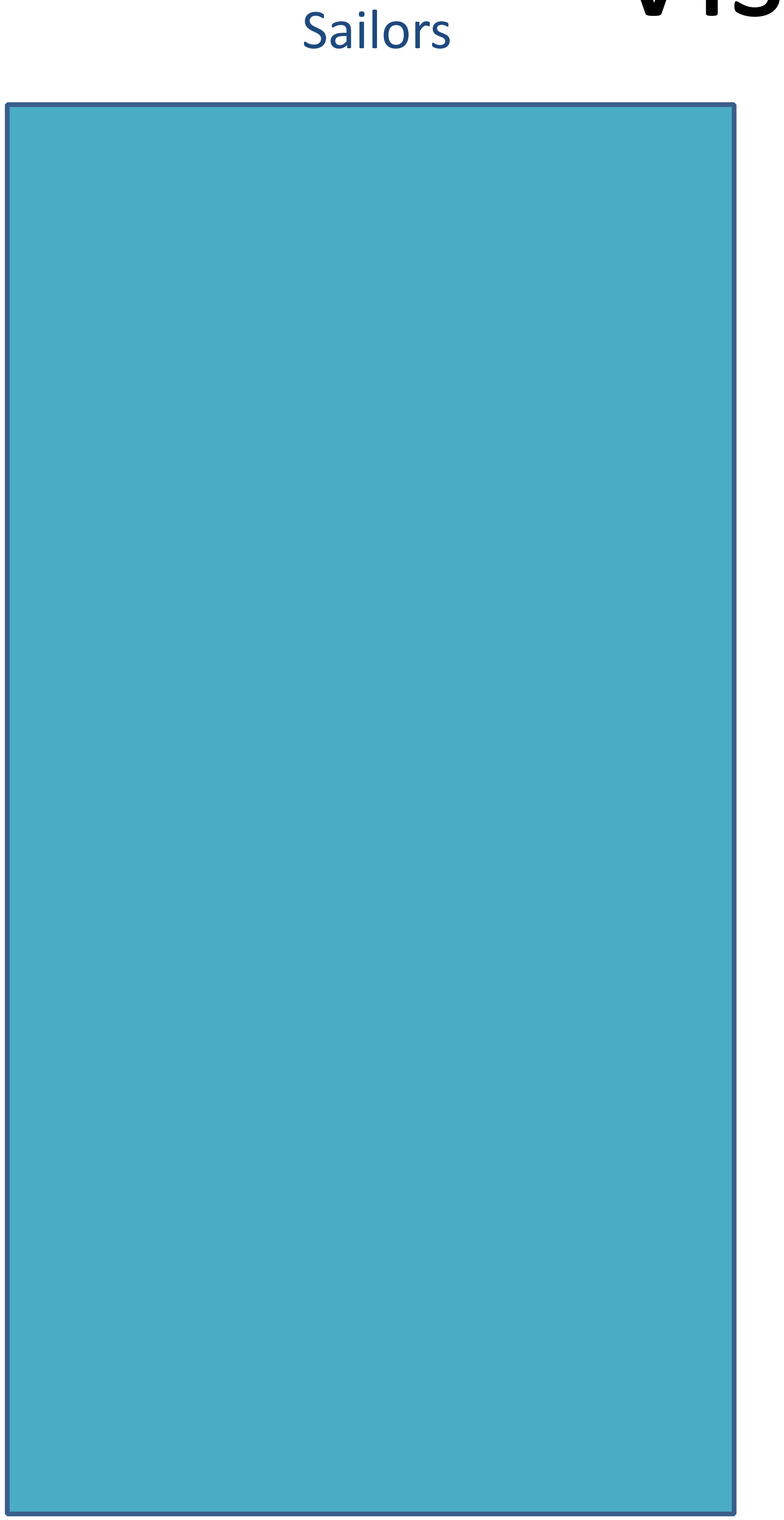
The number of albums released by each artist.

```
SELECT count(*) FROM Artists, Albums
WHERE Artists.artist_id =
Albums.artist_num GROUP BY
Artists.artist_id;
```

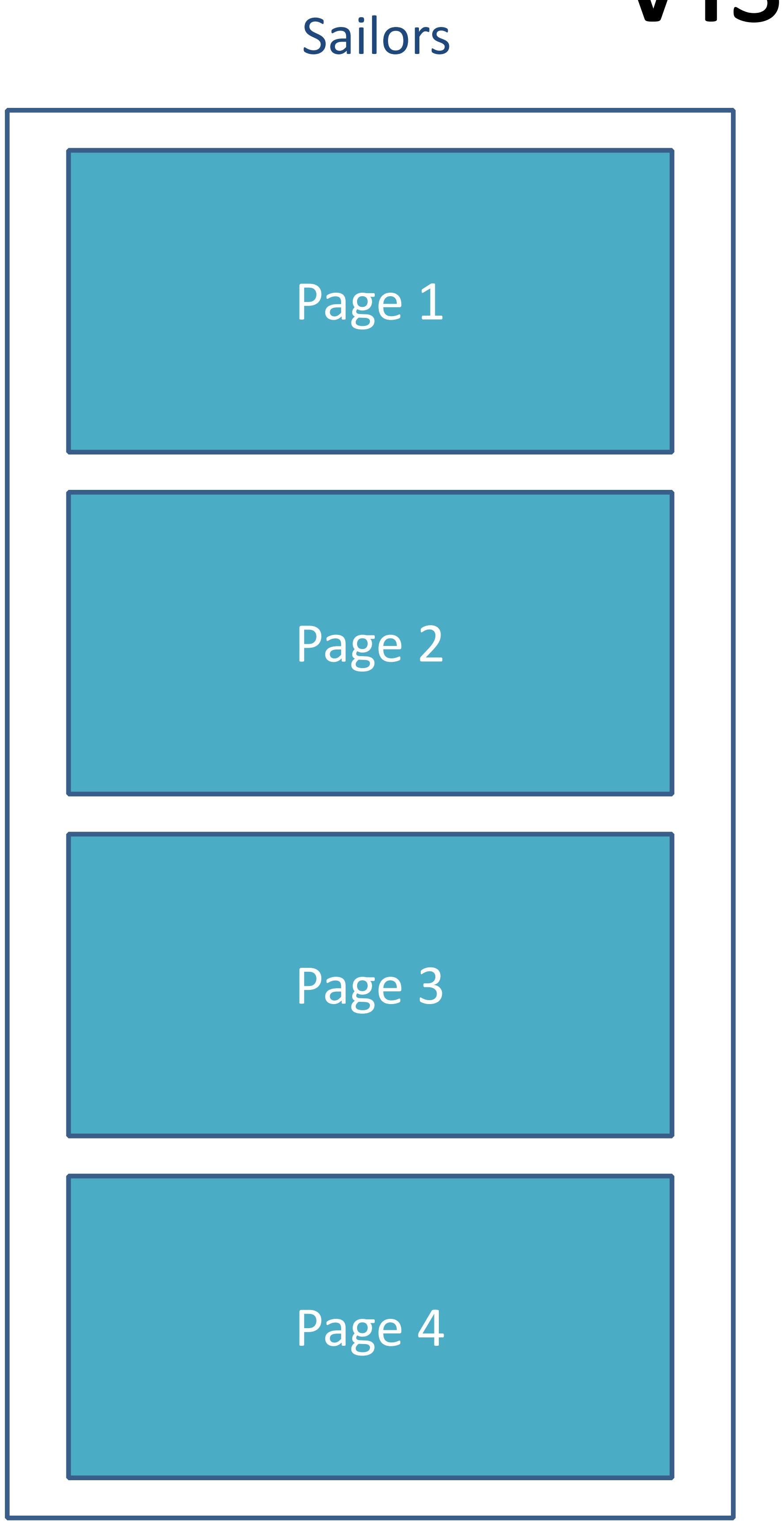
Join Algorithms

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

Visualizations



Visualizations



Sailors

Page 1

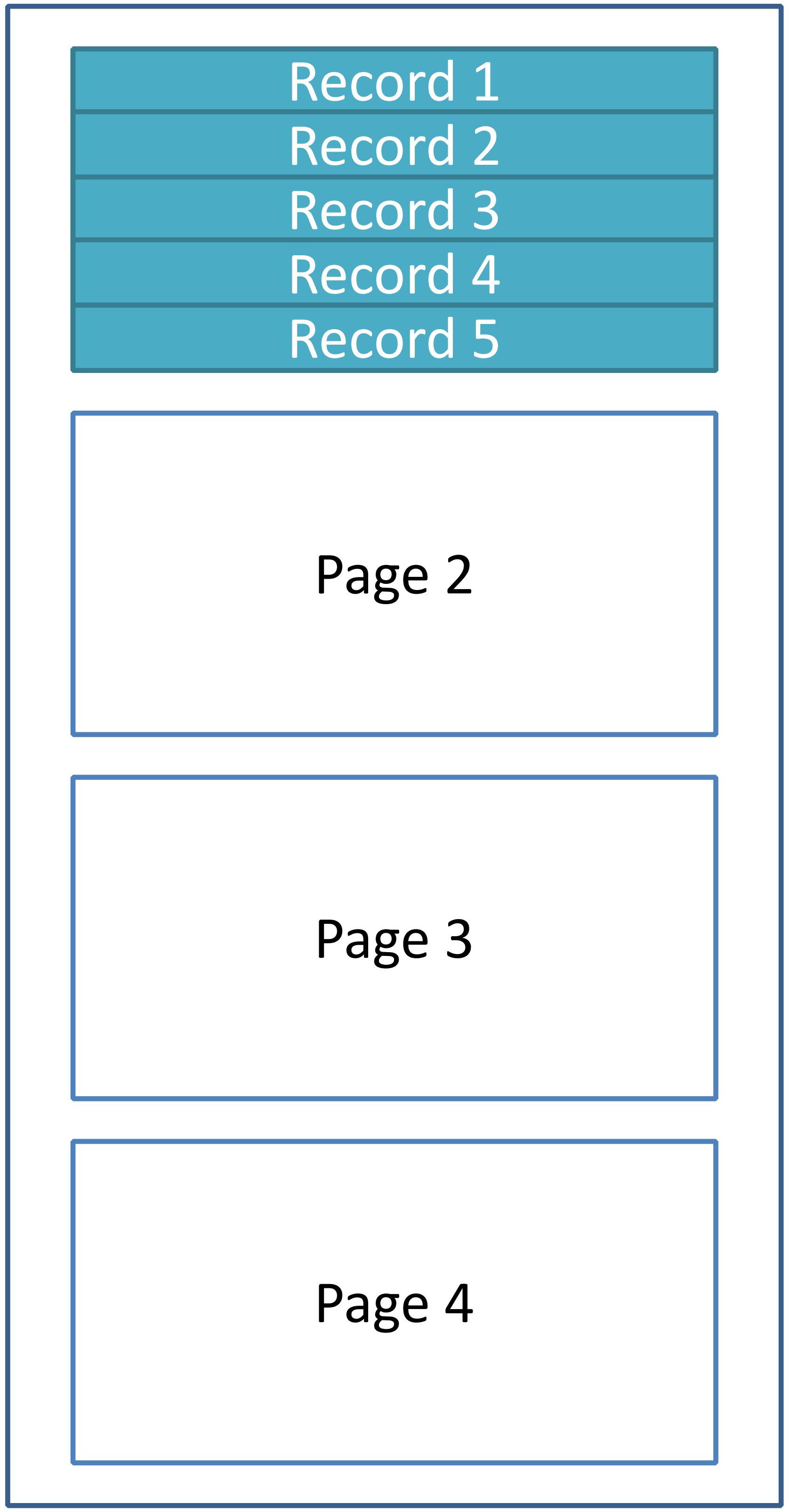
Page 2

Page 3

Page 4

Visualizations

Sailors



Record 1

Record 2

Record 3

Record 4

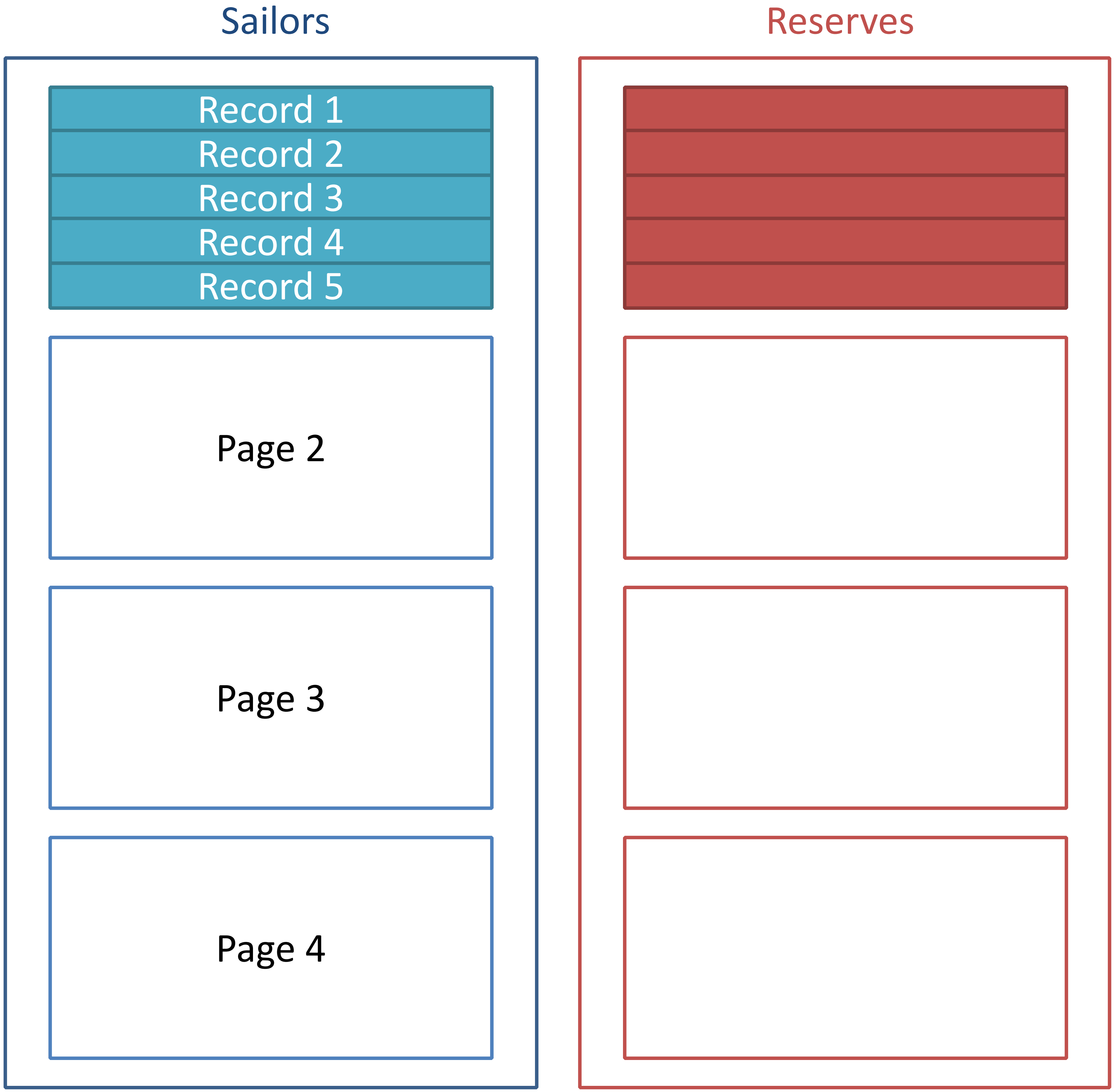
Record 5

Page 2

Page 3

Page 4

Visualizations



Sailors

Record 1

Record 2

Record 3

Record 4

Record 5

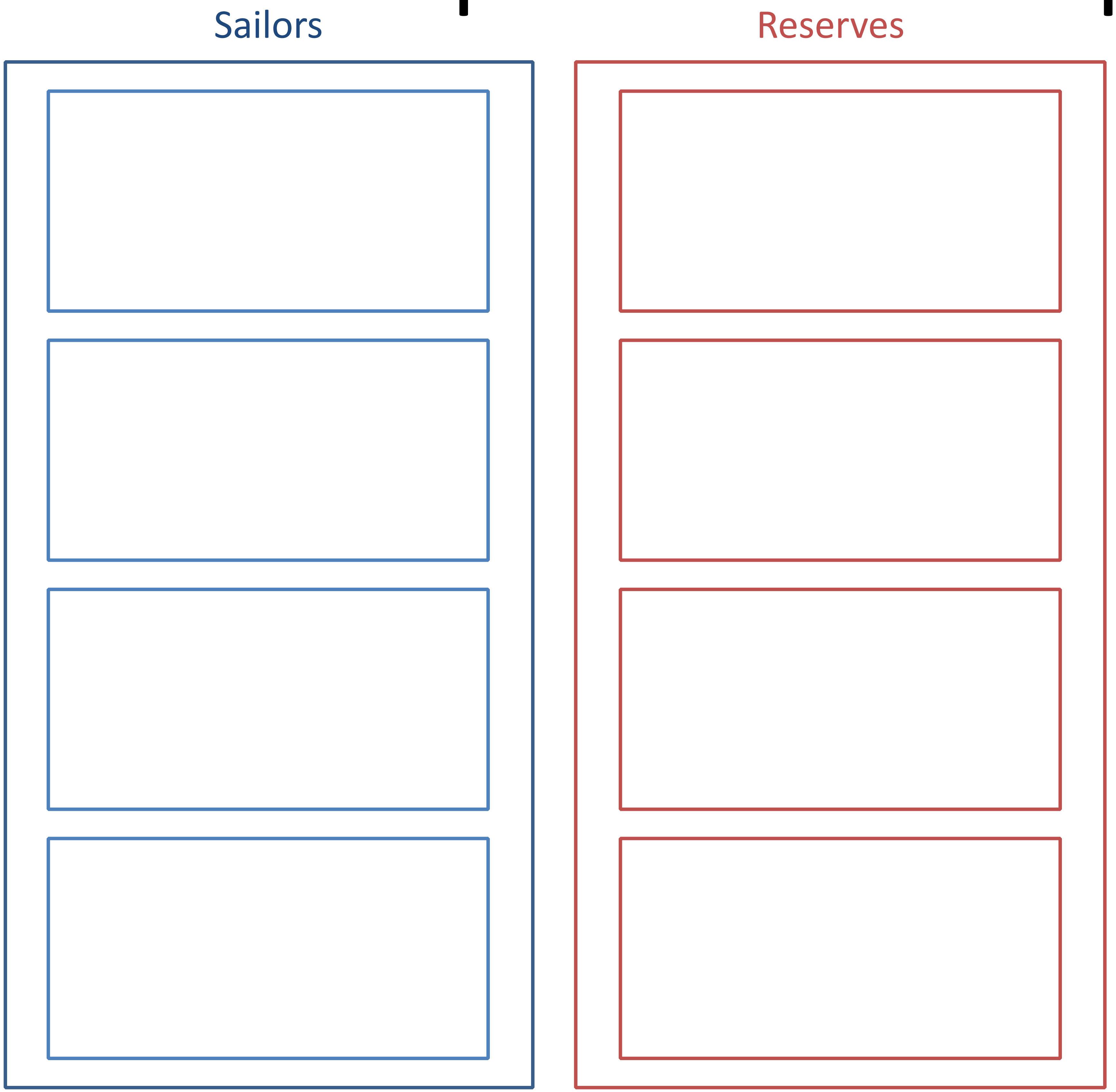
Page 2

Page 3

Page 4

Reserves

Simple Nested Loops Join



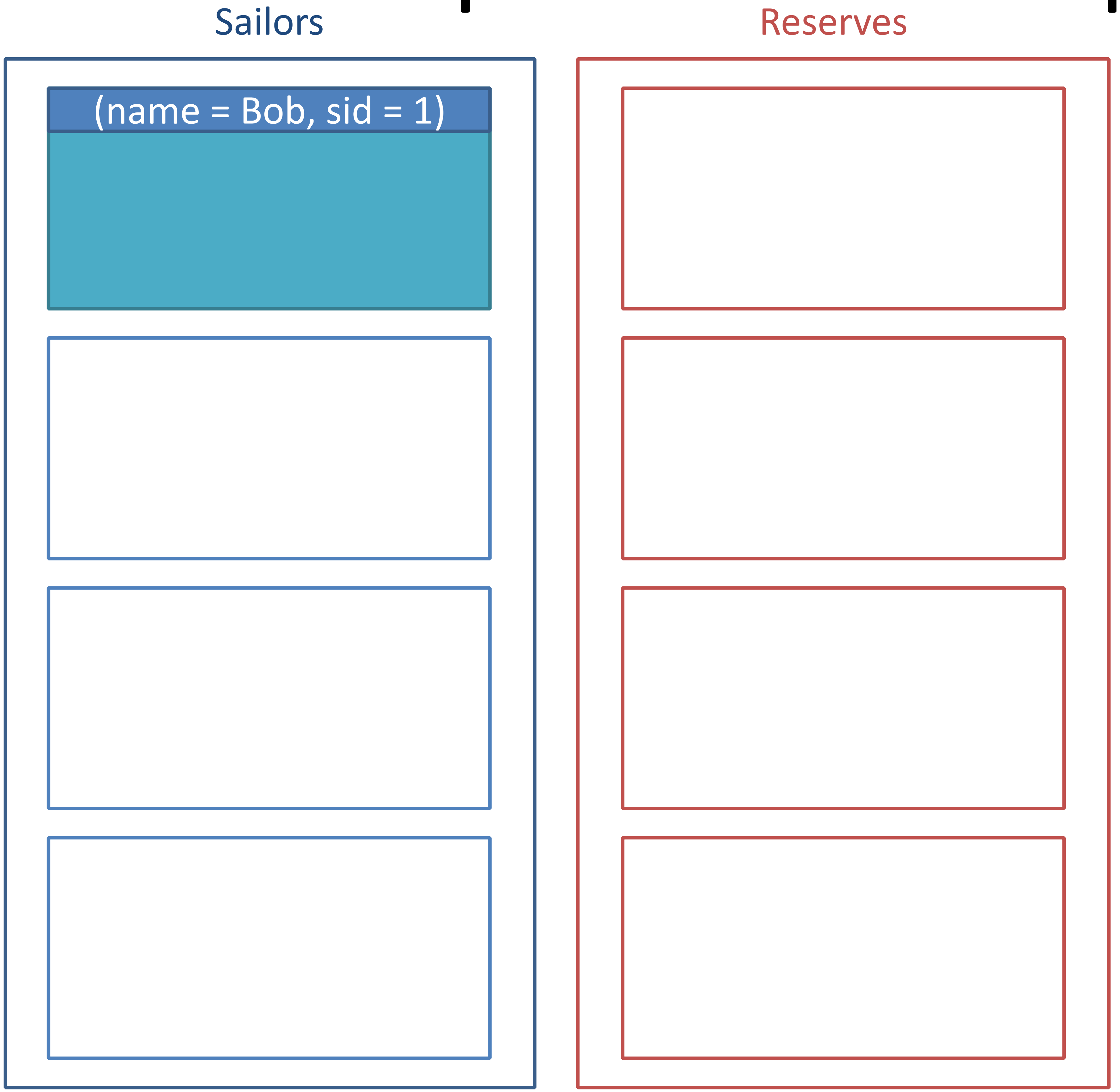
Key idea:

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



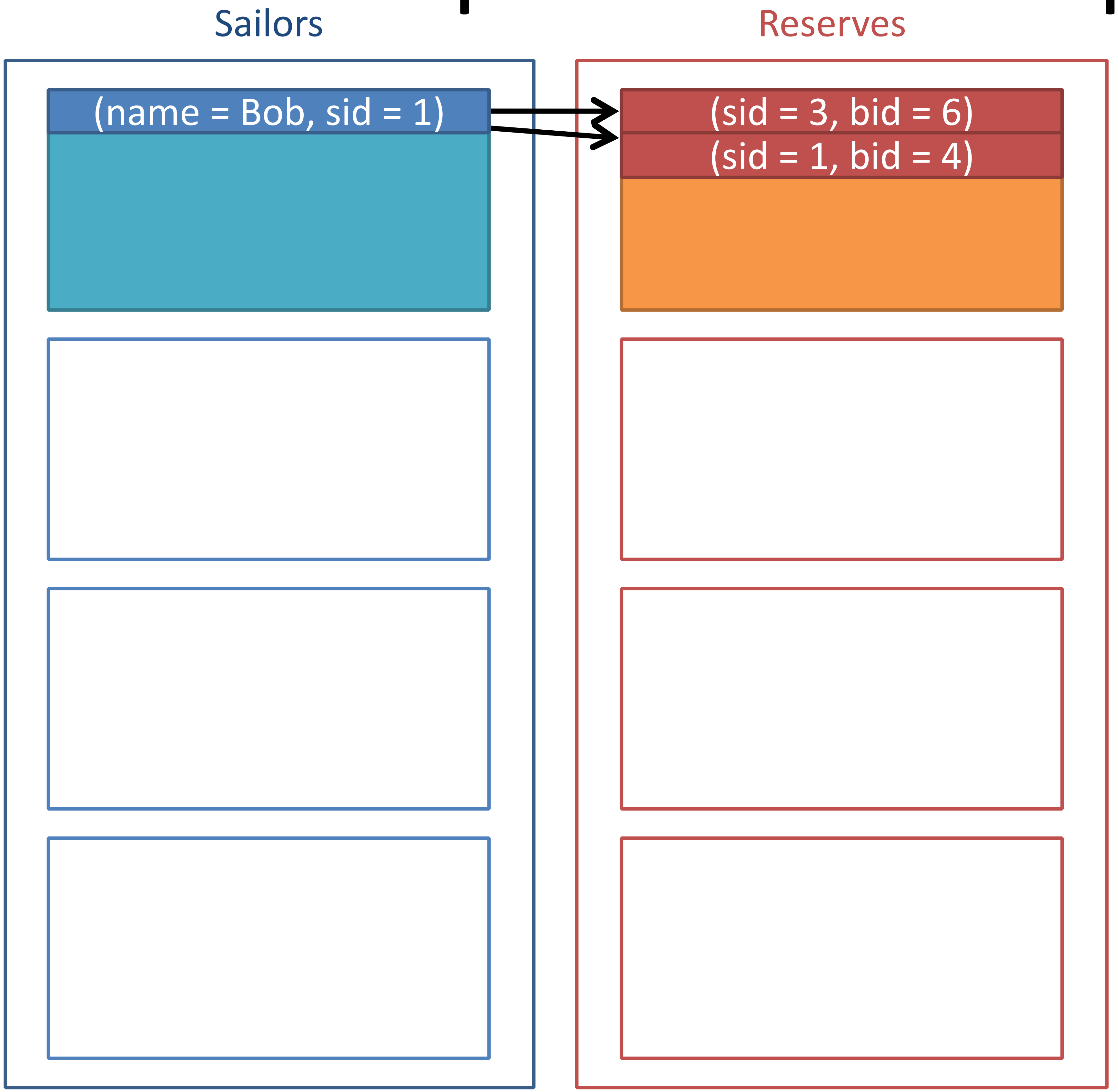
Key idea:

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



Key idea:

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

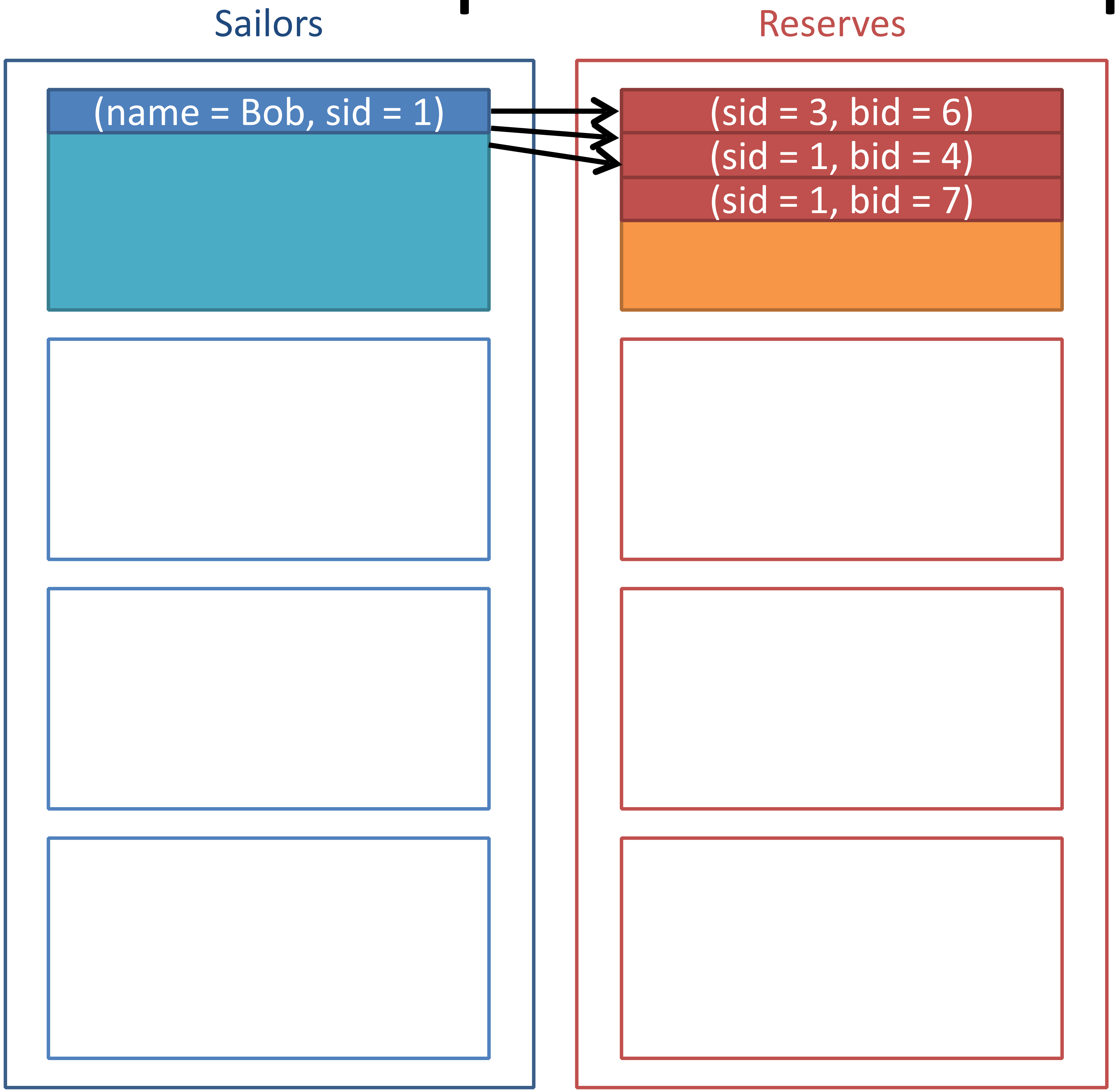
Steps:

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:

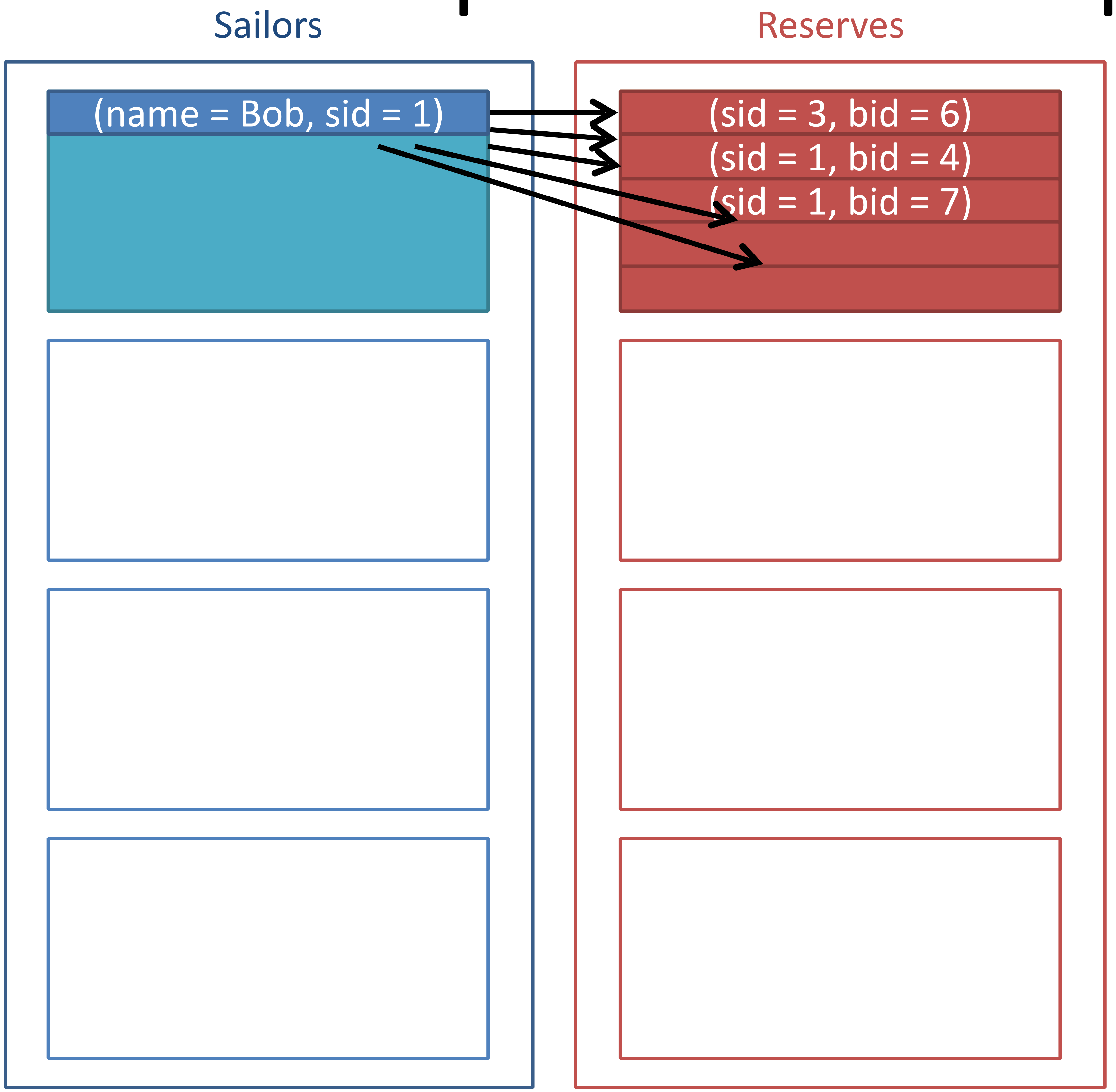
1. Get tuple of S.
2. Iterate through each tuple in R.

Output:

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

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

Simple Nested Loops Join



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:

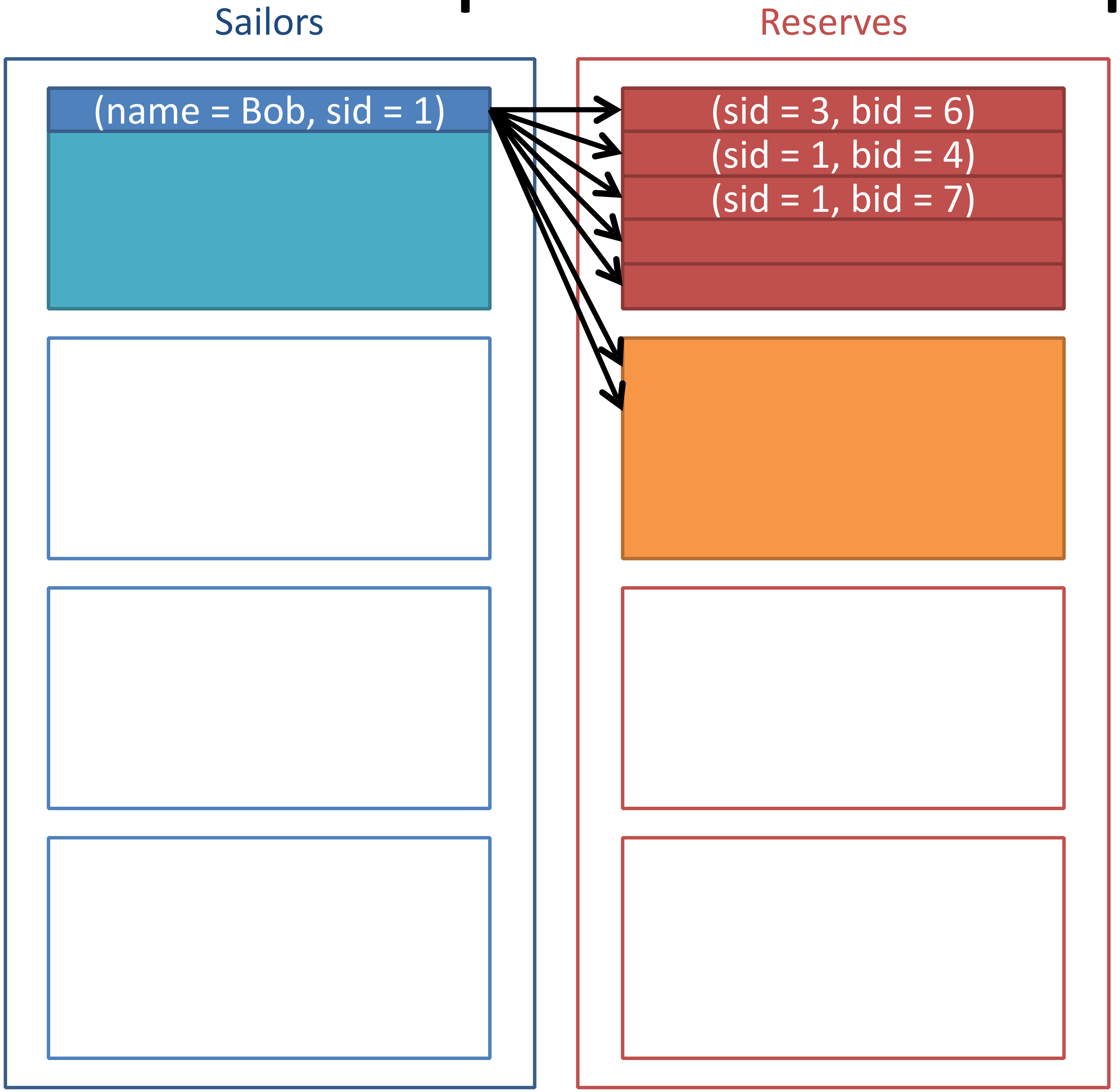
1. Get tuple of S.
2. Iterate through each tuple in R.

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.

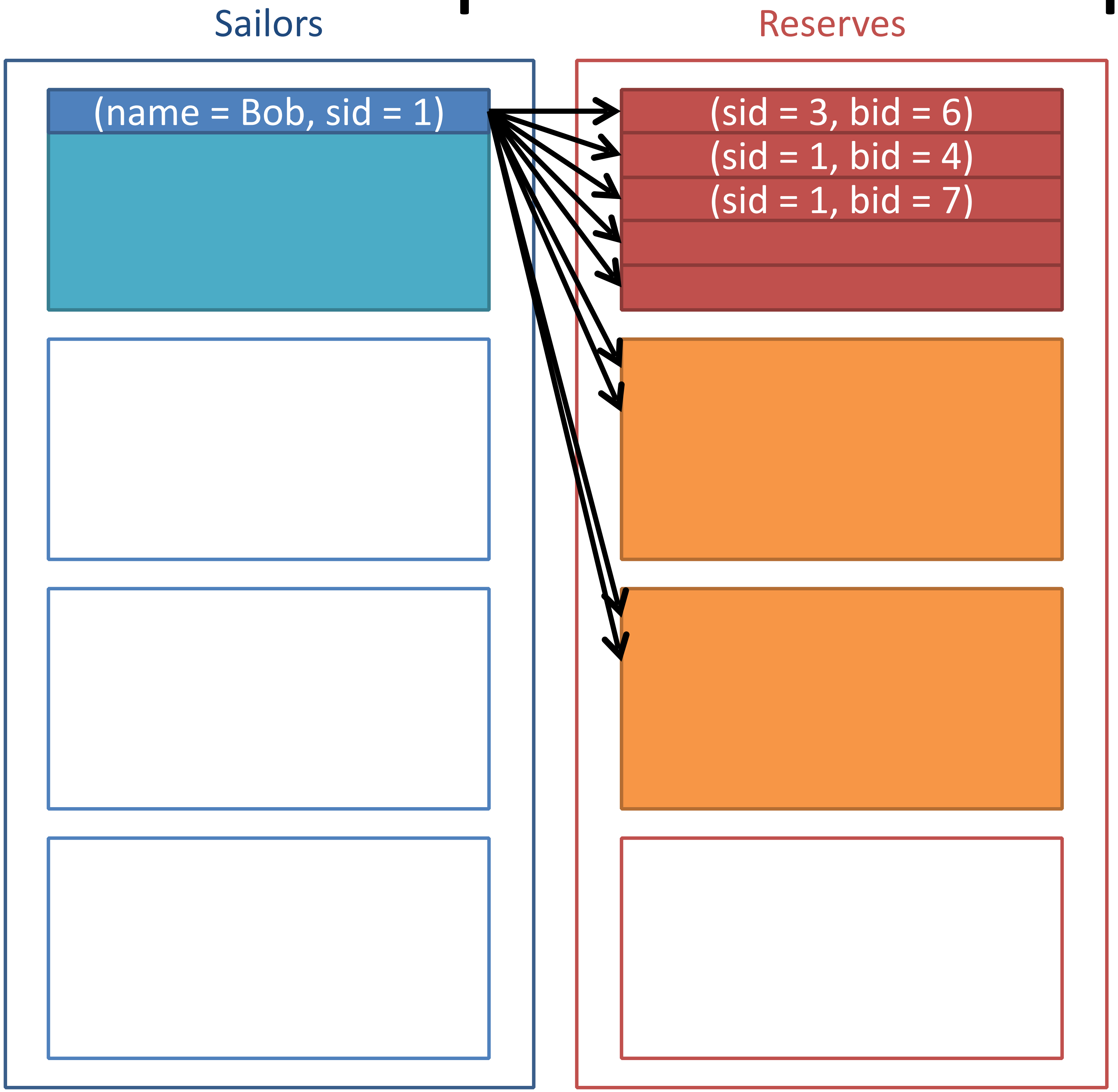
Steps:

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.

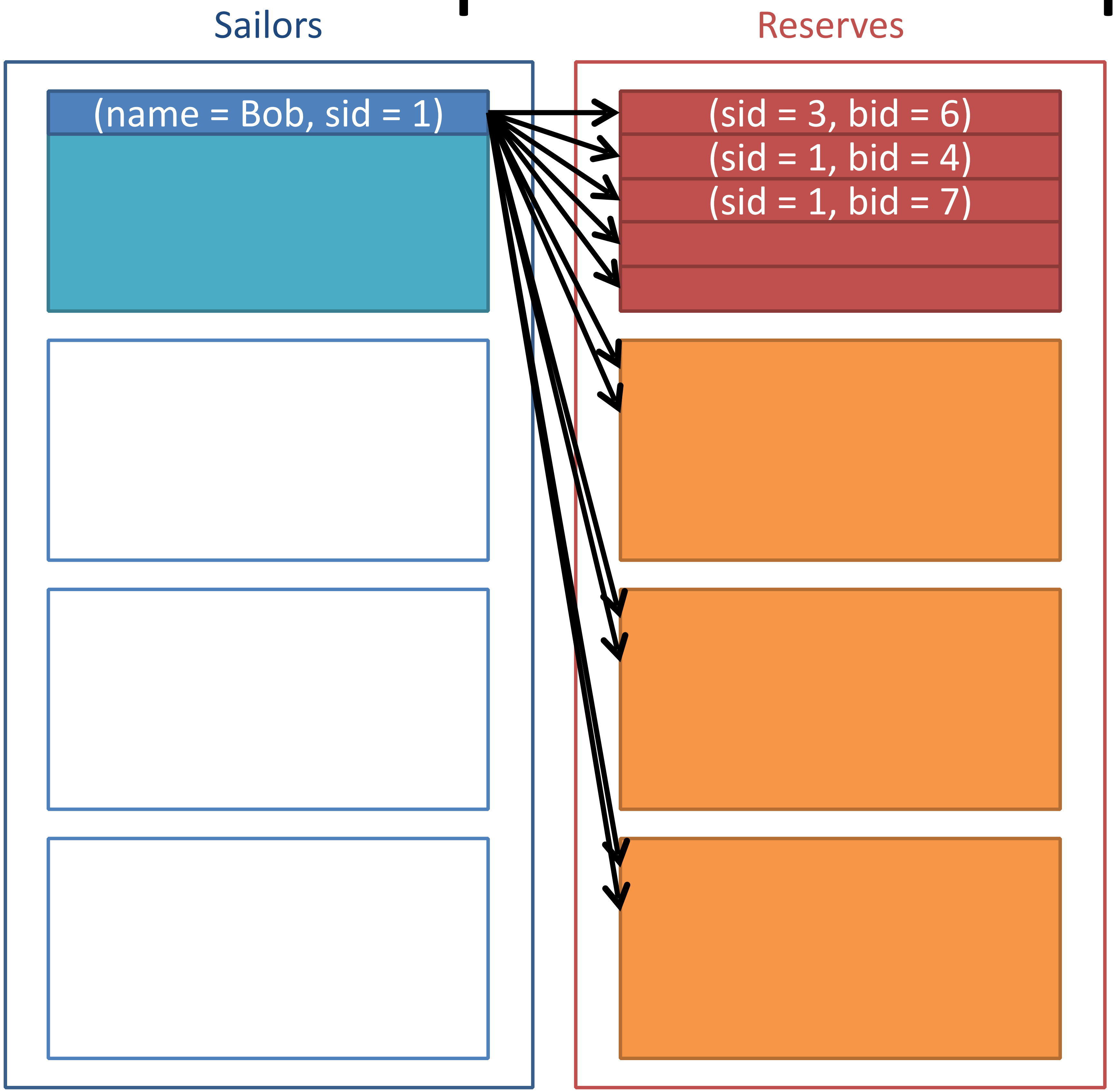
Steps:

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



Key idea:

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

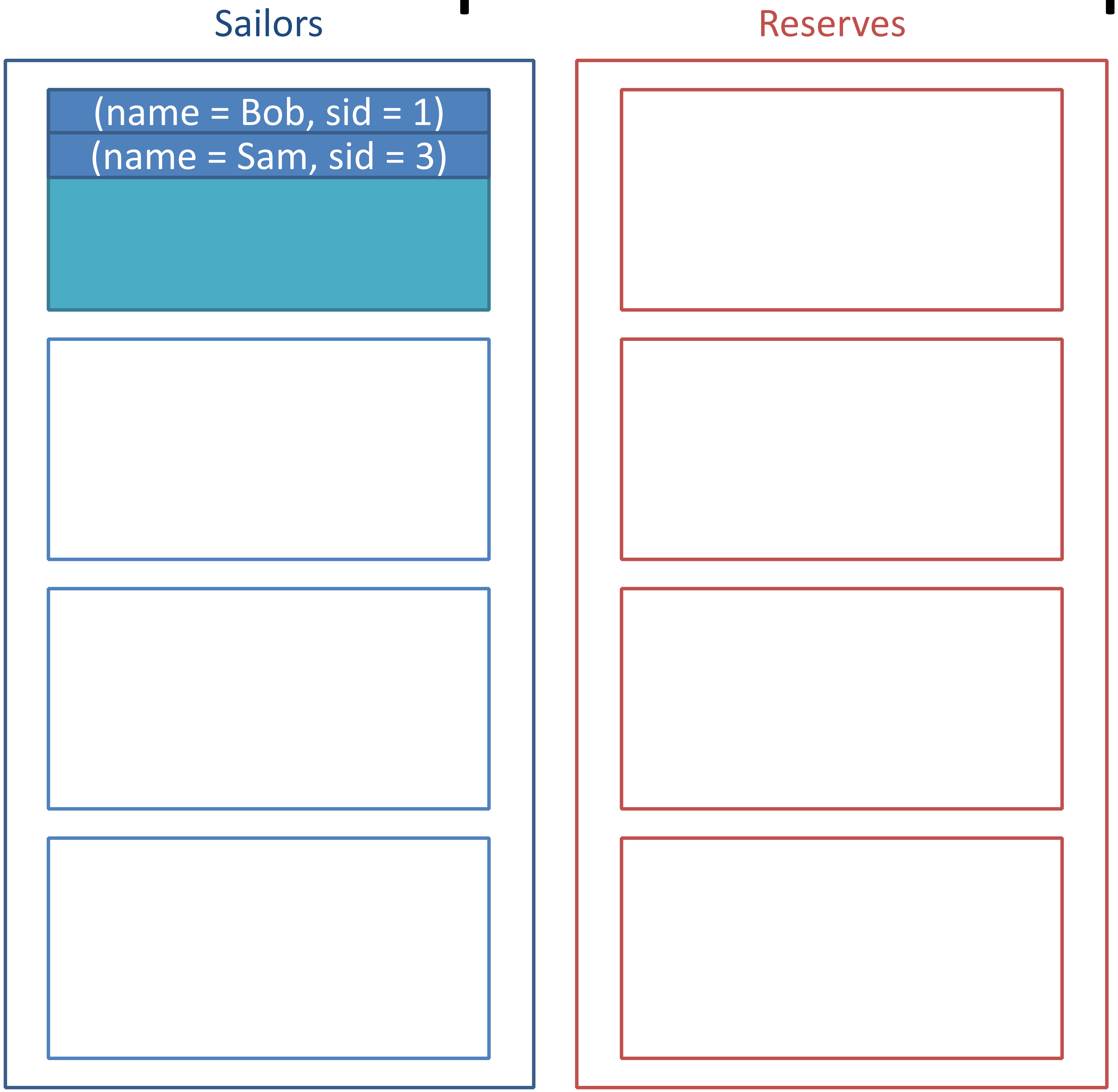
Steps:

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.

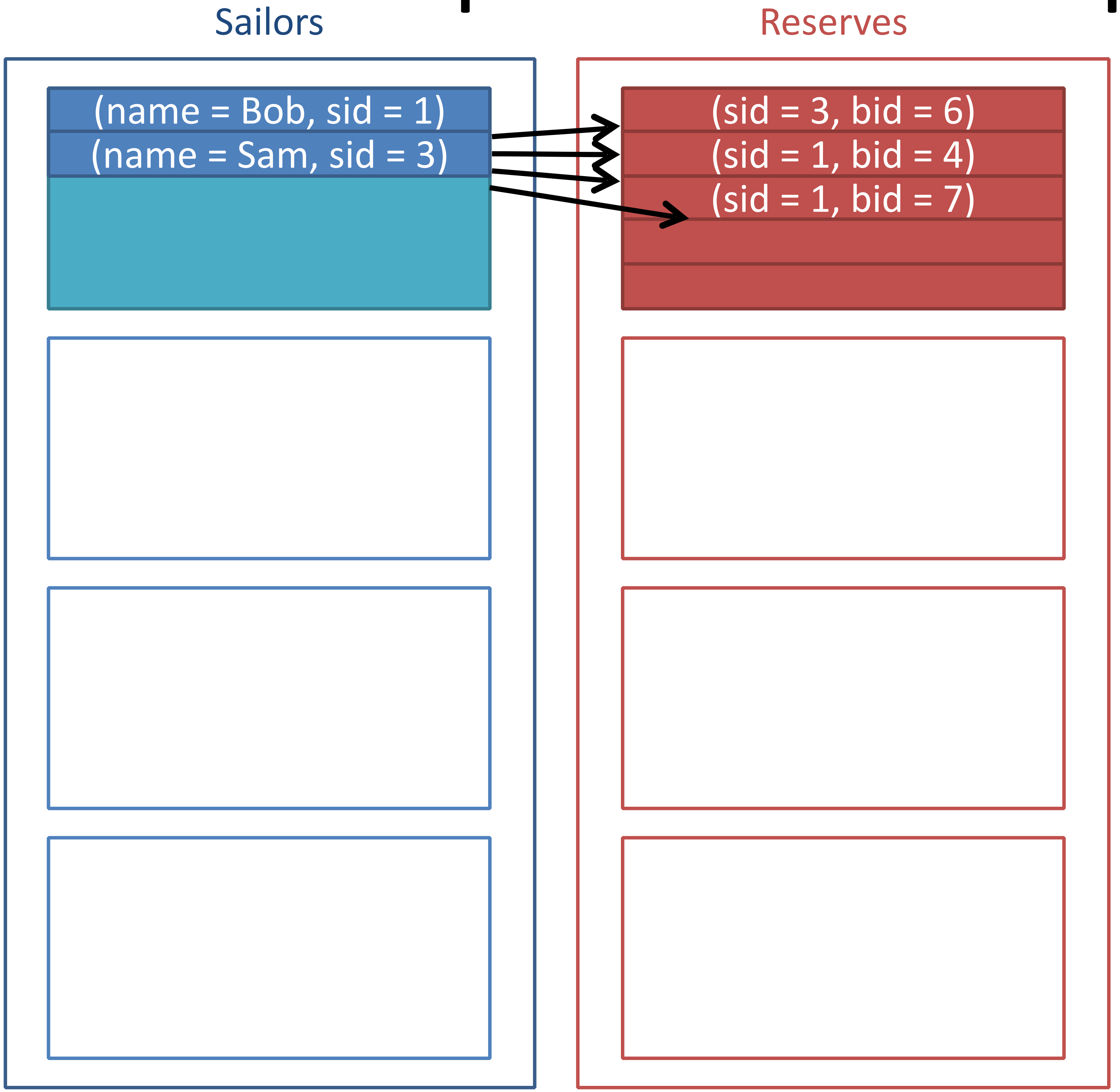
Steps:

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)

(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:

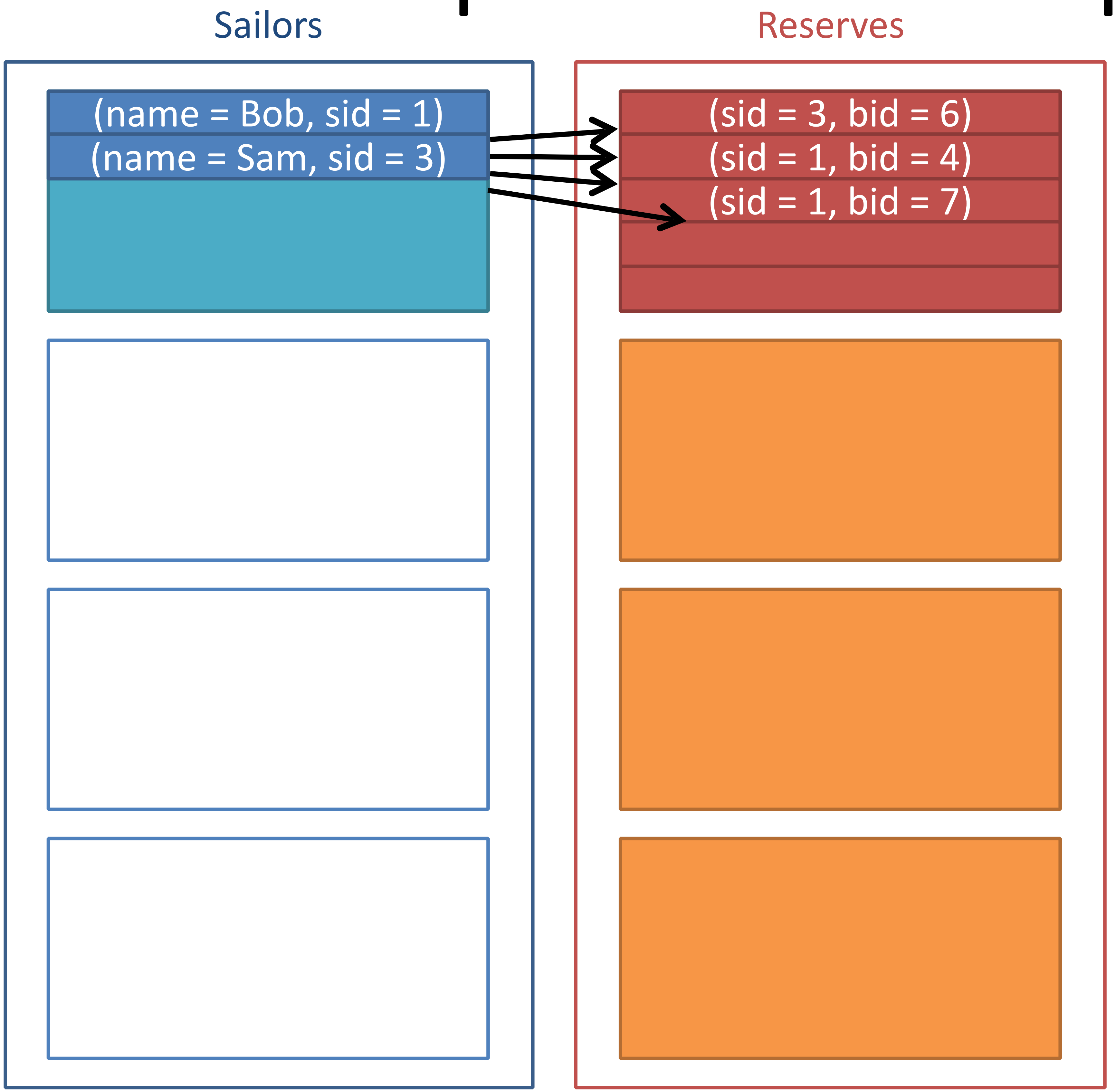
(name = Bob, sid = 1, bid = 4)

(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.

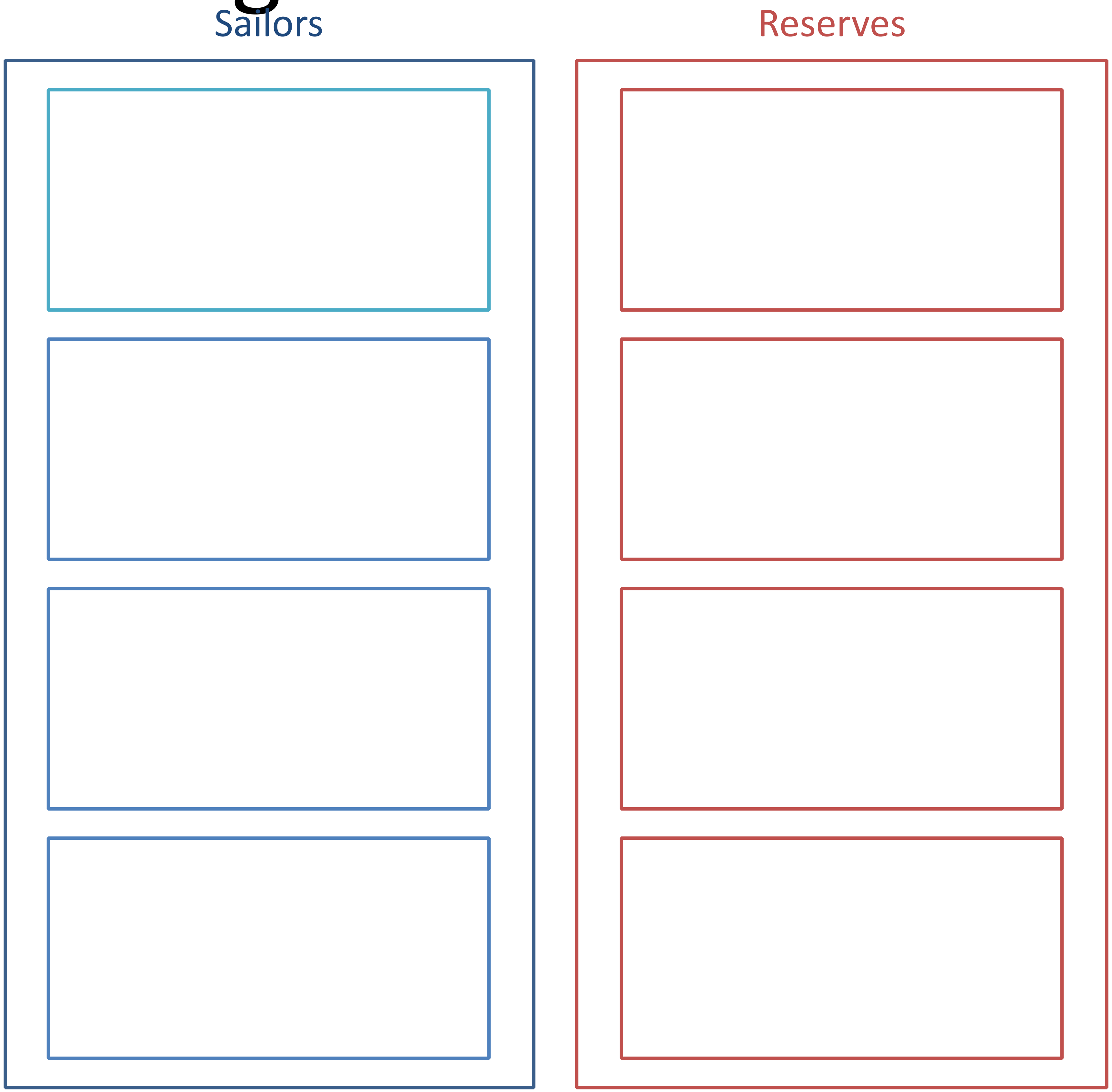
Steps:

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

I/Os:

[S] + |S|*[R]

Page-Oriented Nested Loops Join



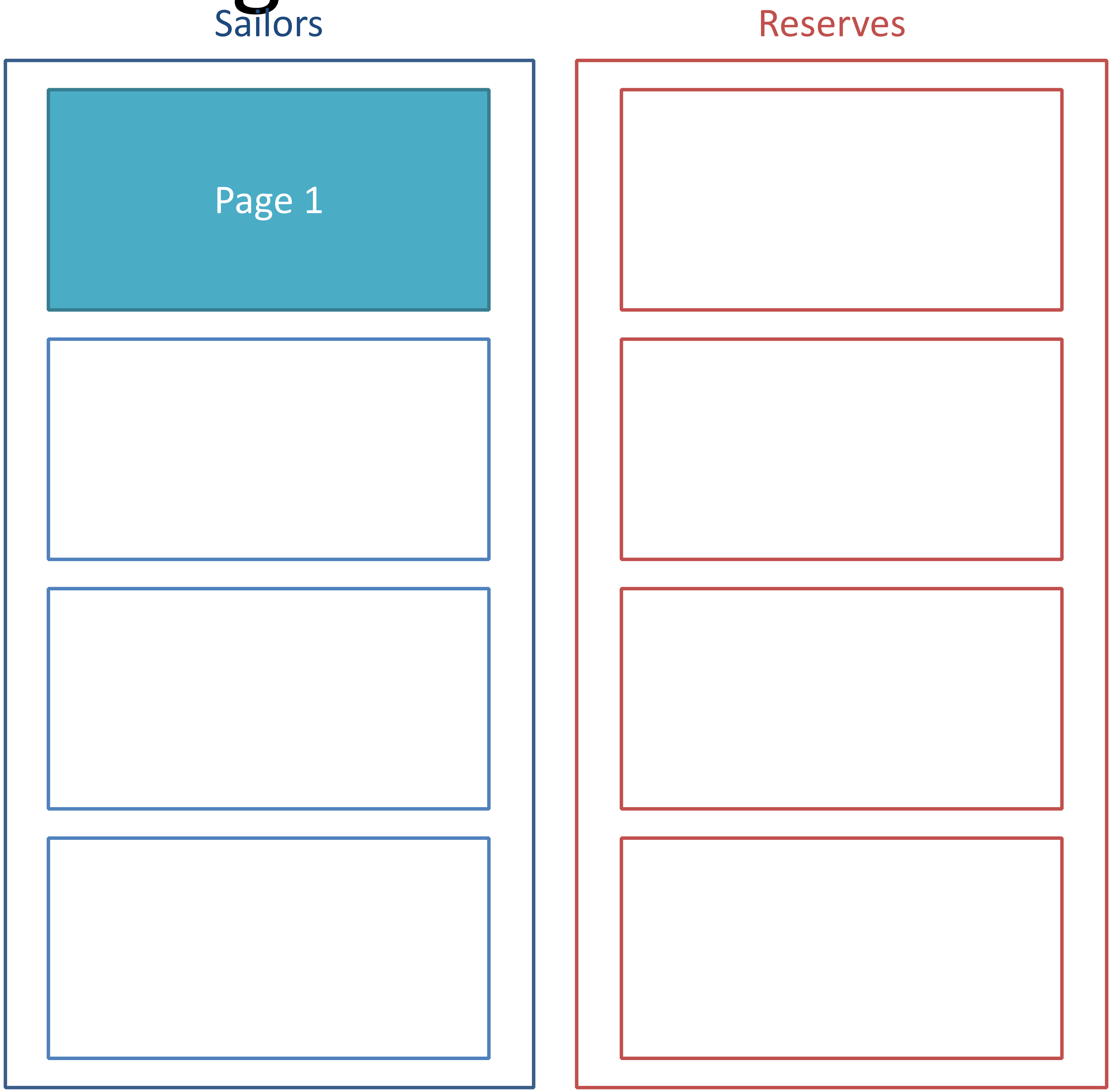
Key idea:

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



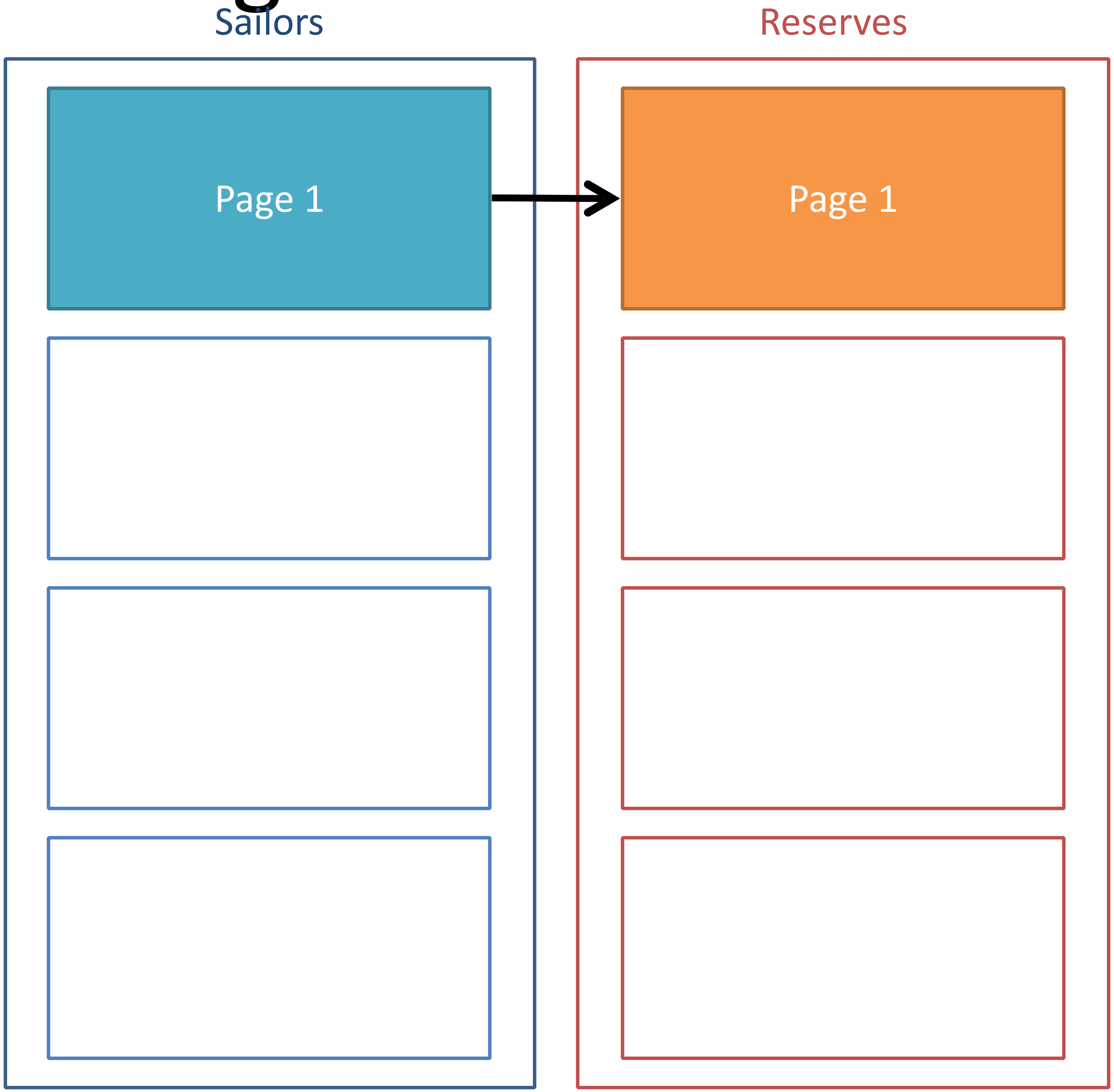
Key idea:

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



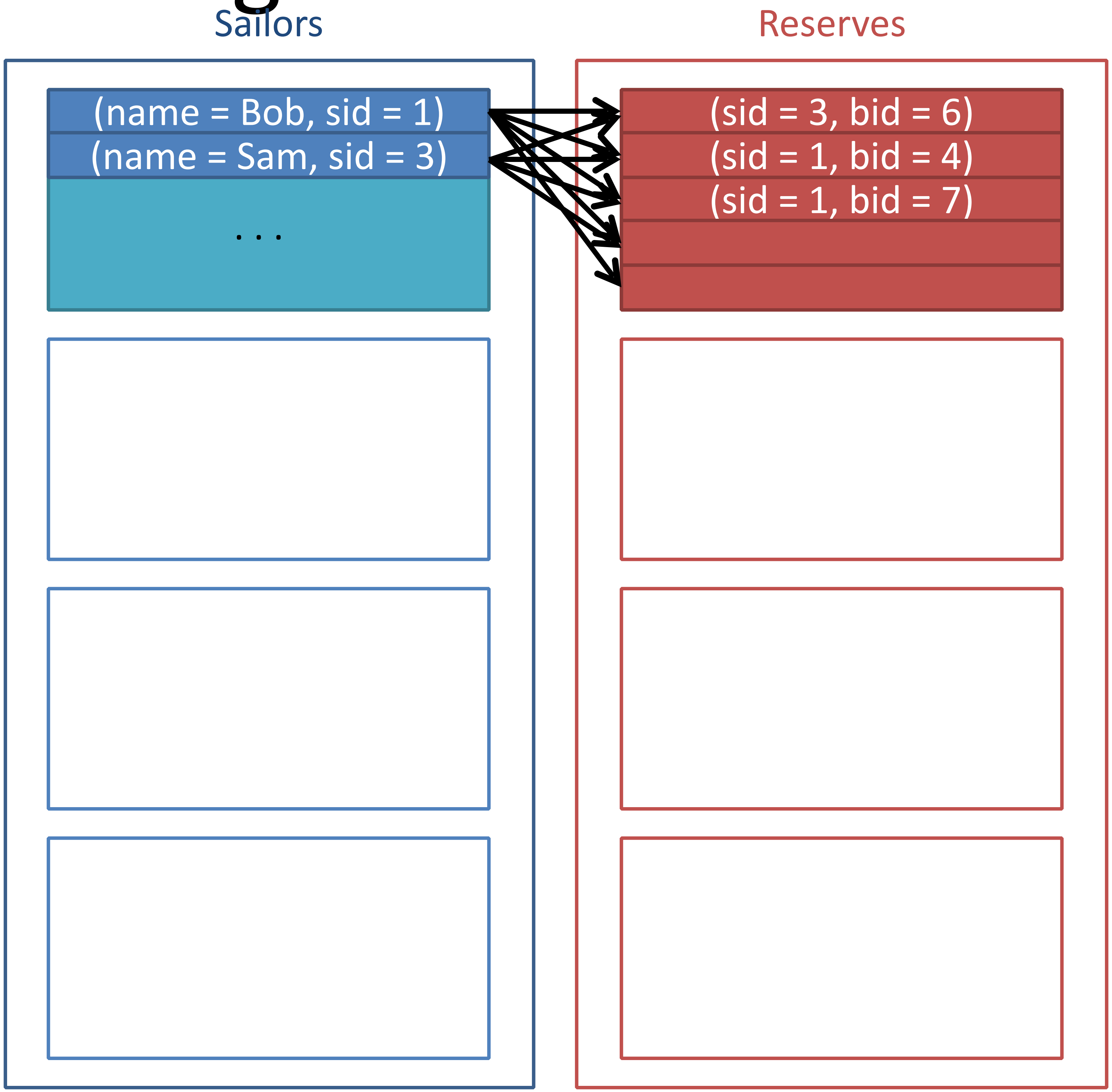
Key idea:

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.

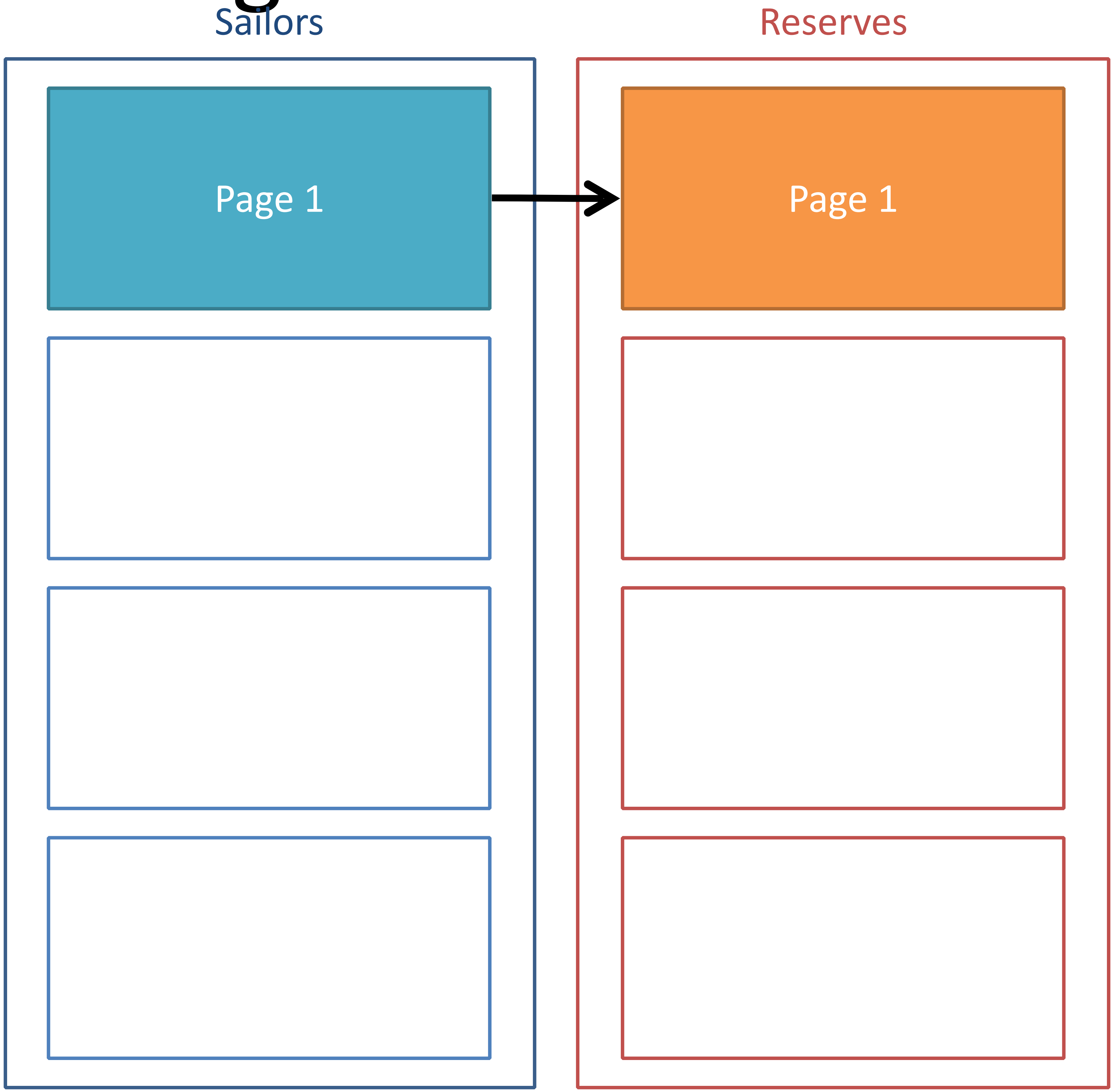
Output:

(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.

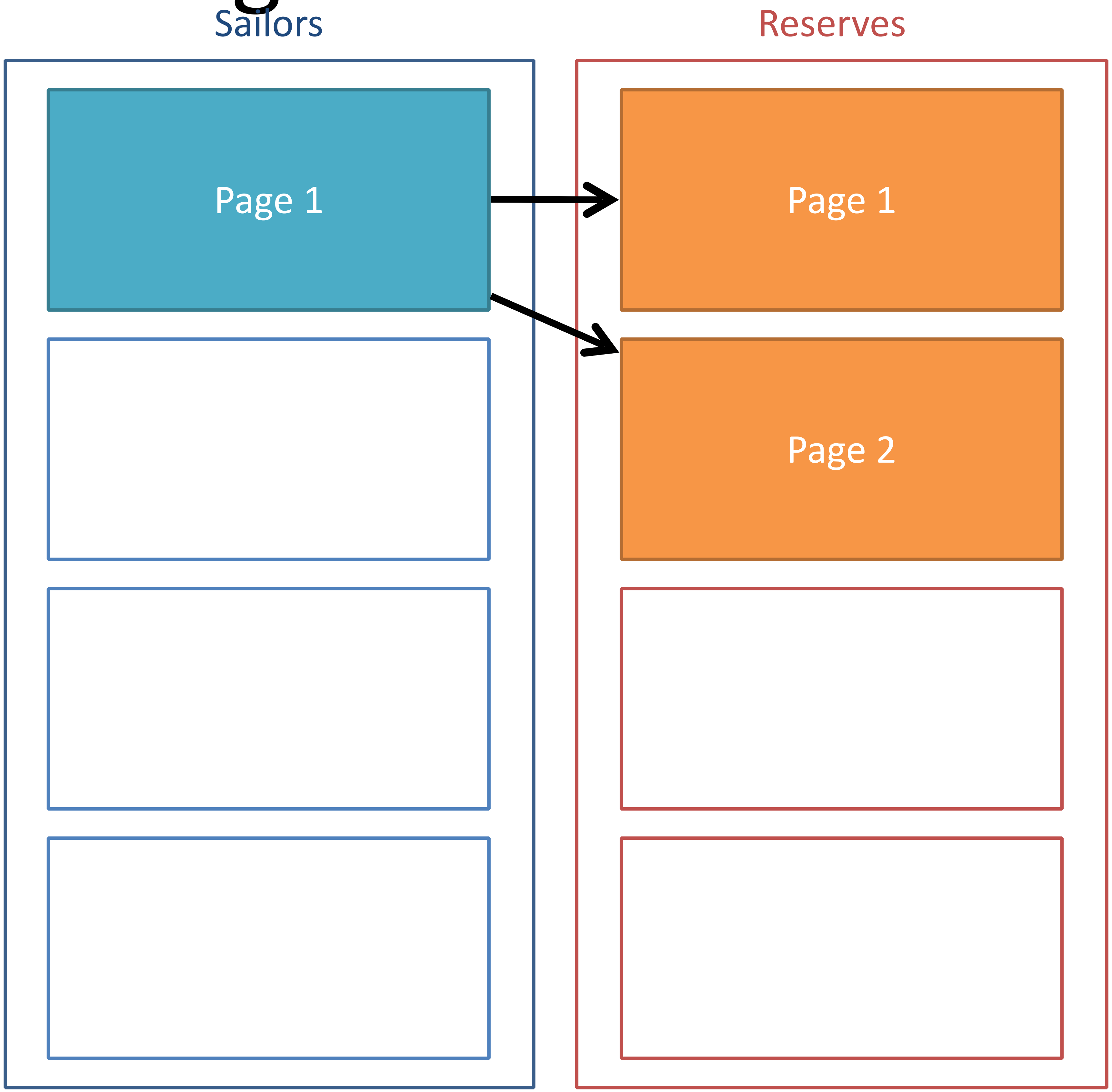
Steps:

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.

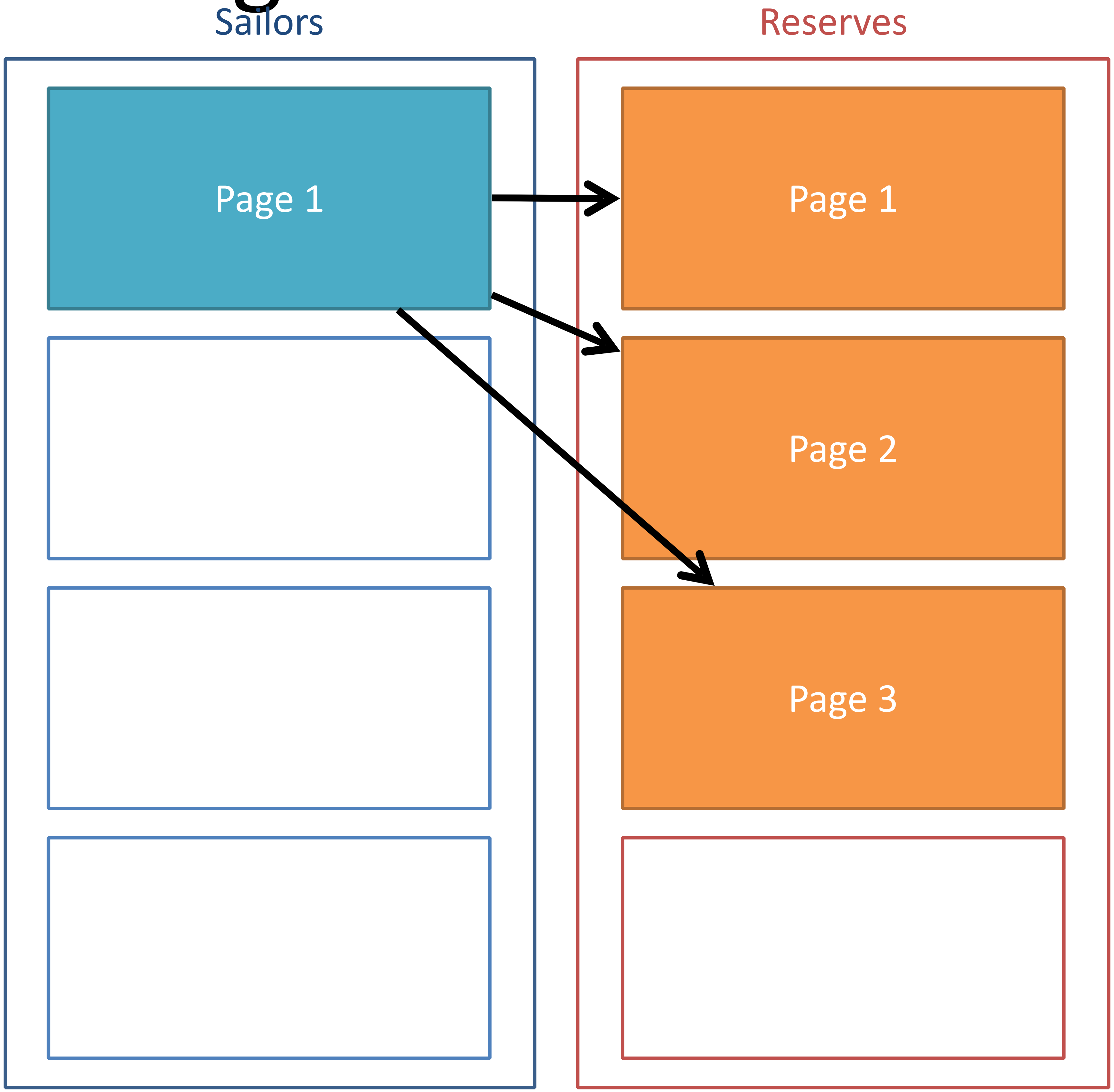
Steps:

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.

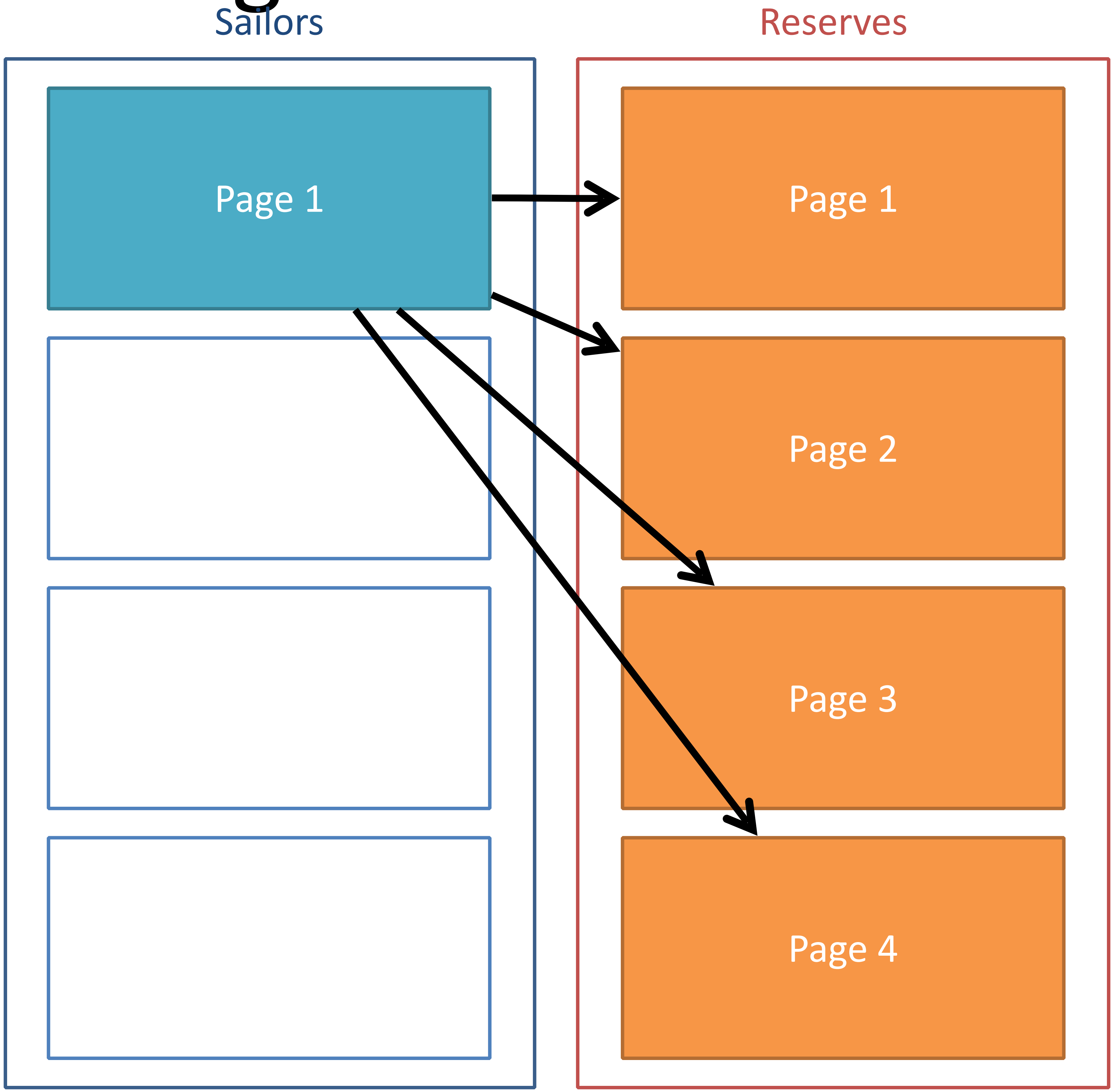
Steps:

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.

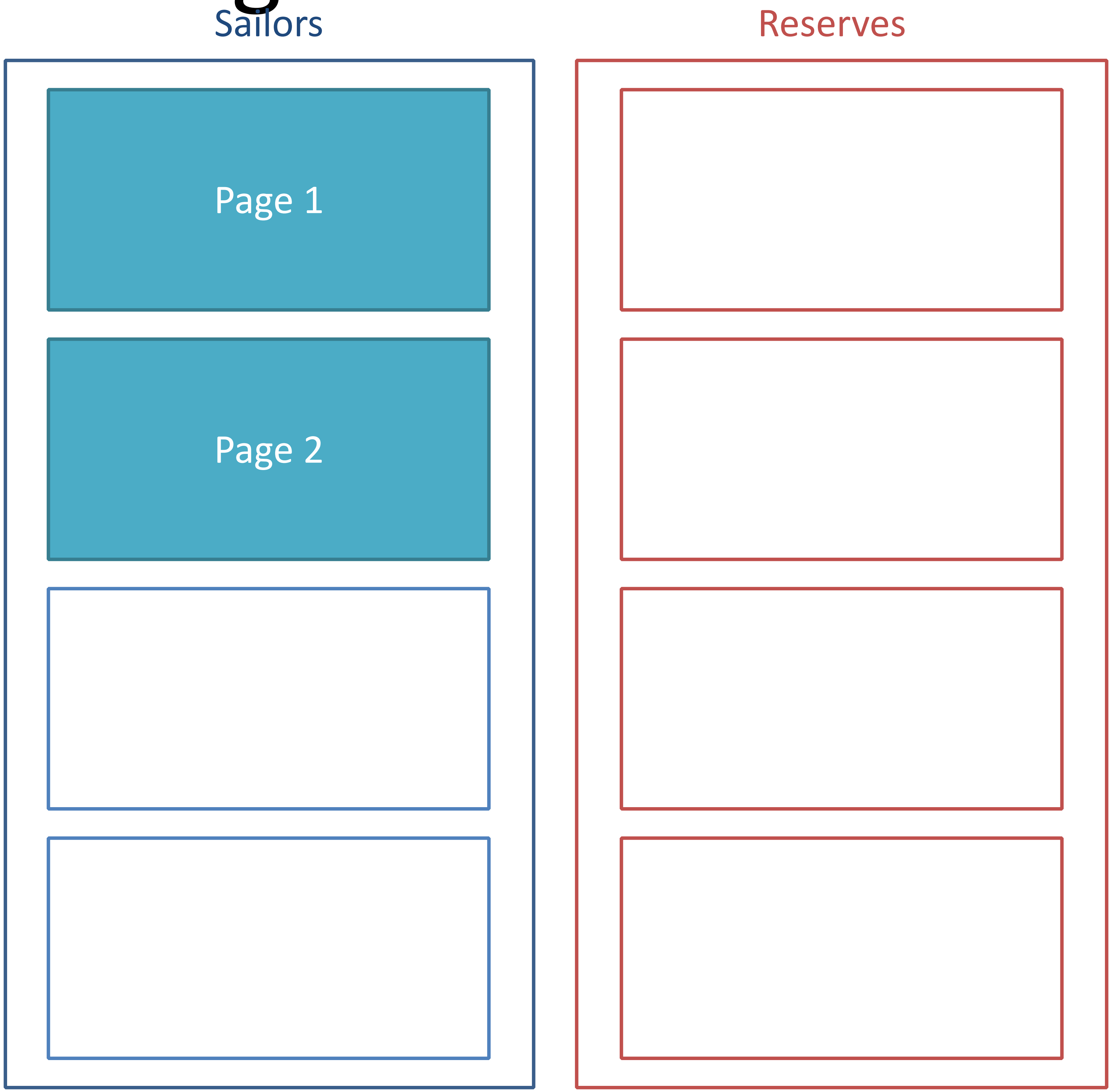
Steps:

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.

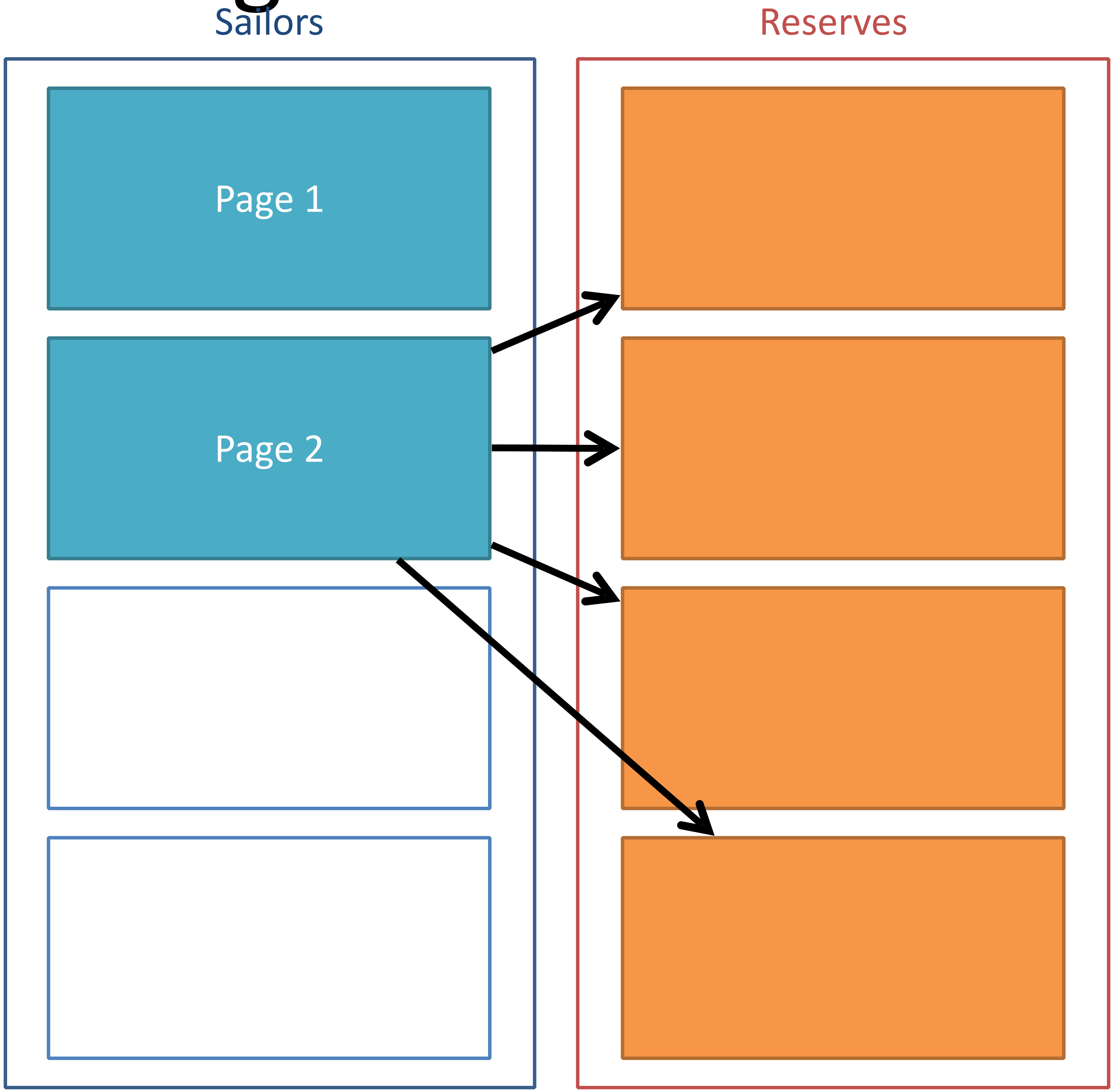
Output:

(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:

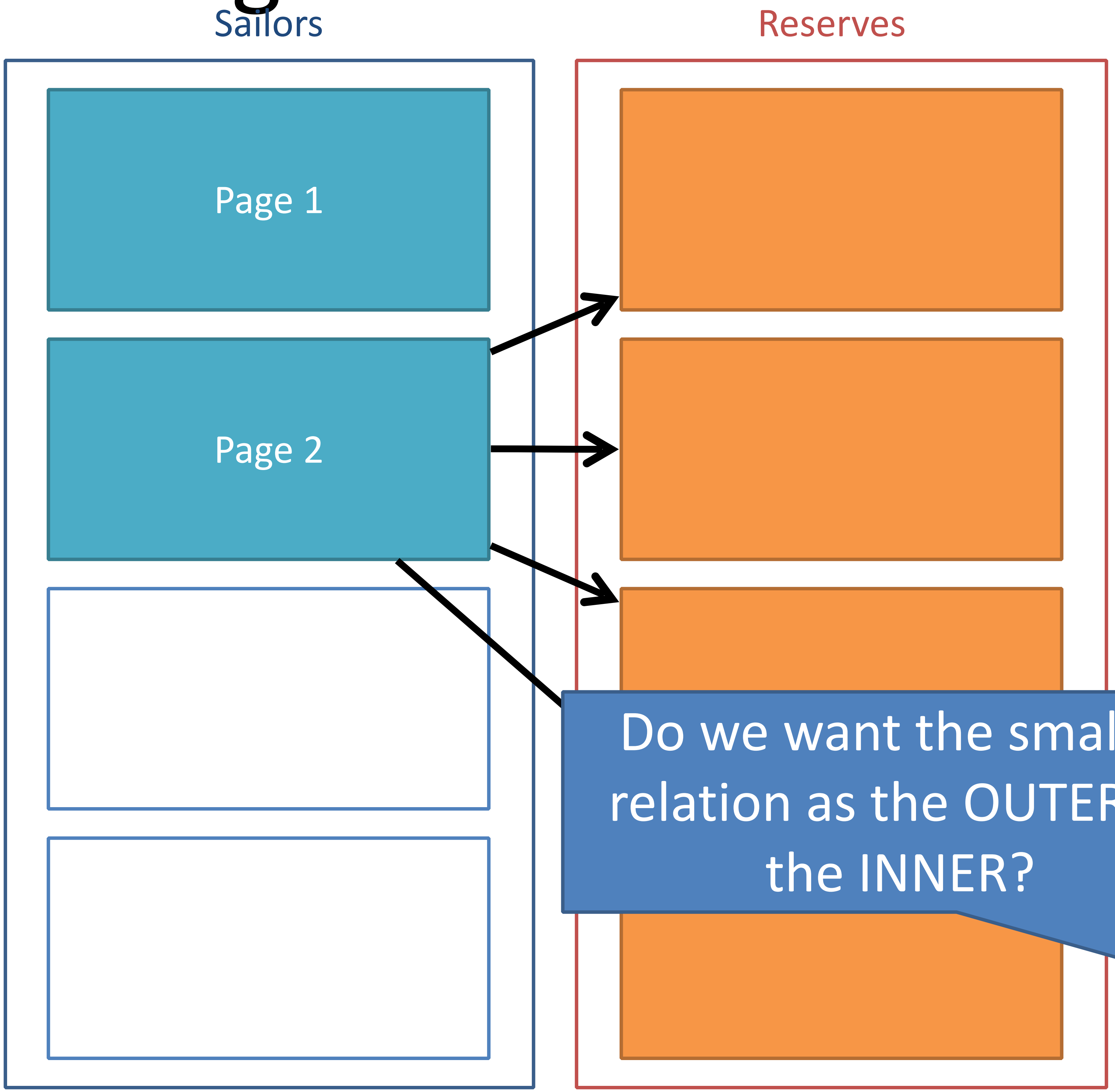
(name = Bob, sid = 1, bid = 4)

(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



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

Key idea:

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

Steps:

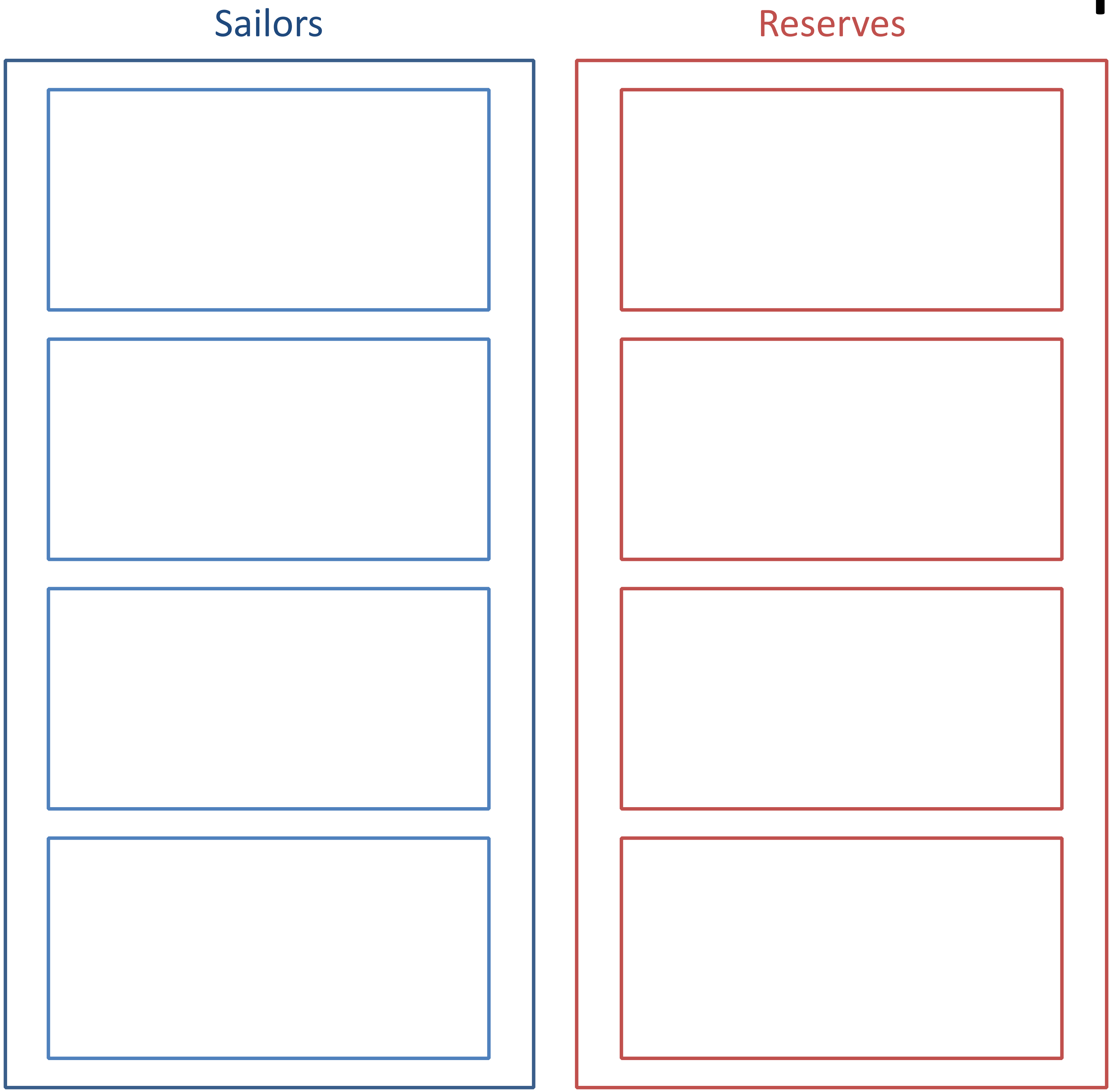
1. Get page of S.
2. Iterate through each page in R.

ompare tuples in ach.

:

[S] + [S]*[R]

Chunk Nested Loops Join



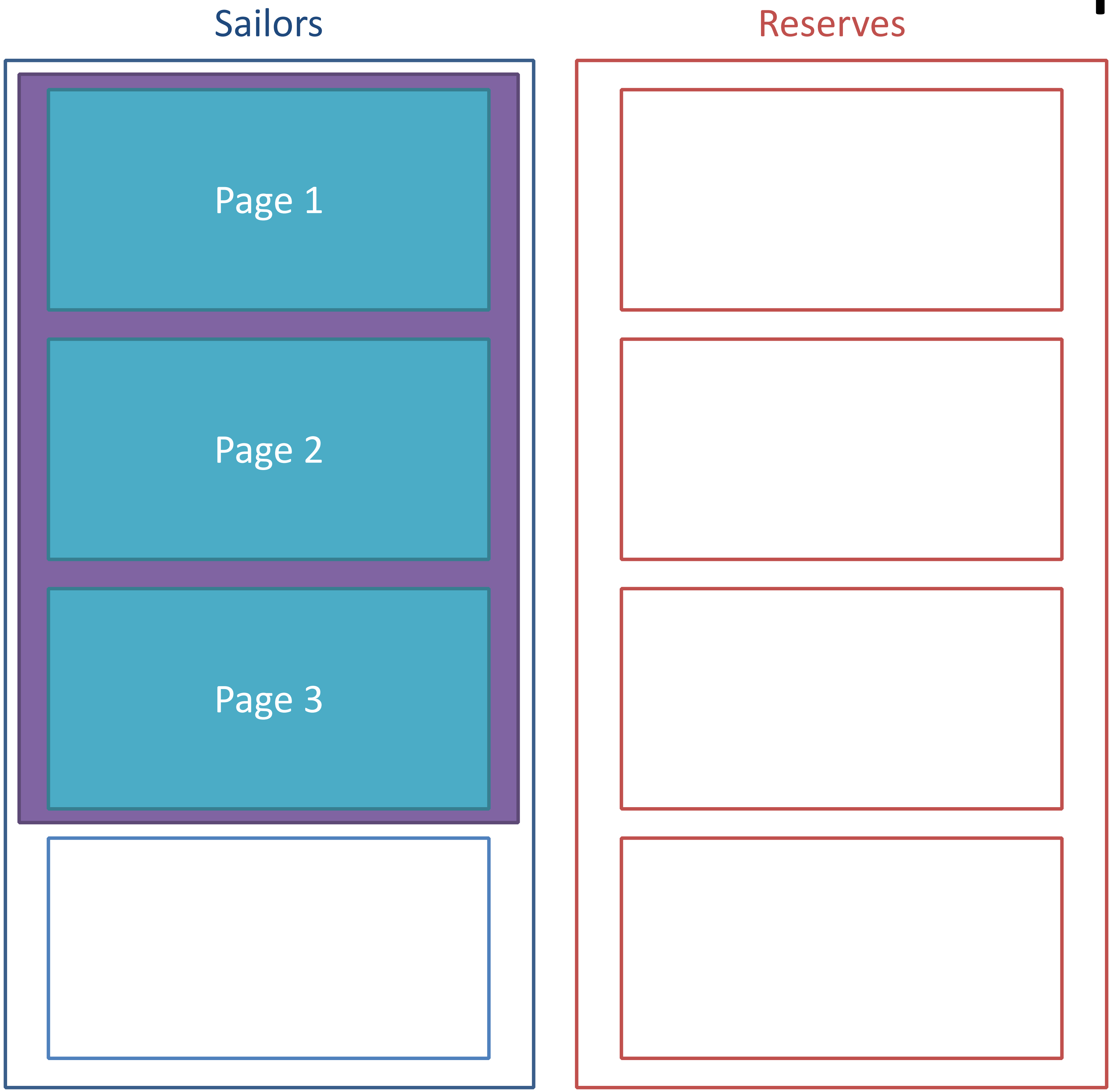
Key idea:

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

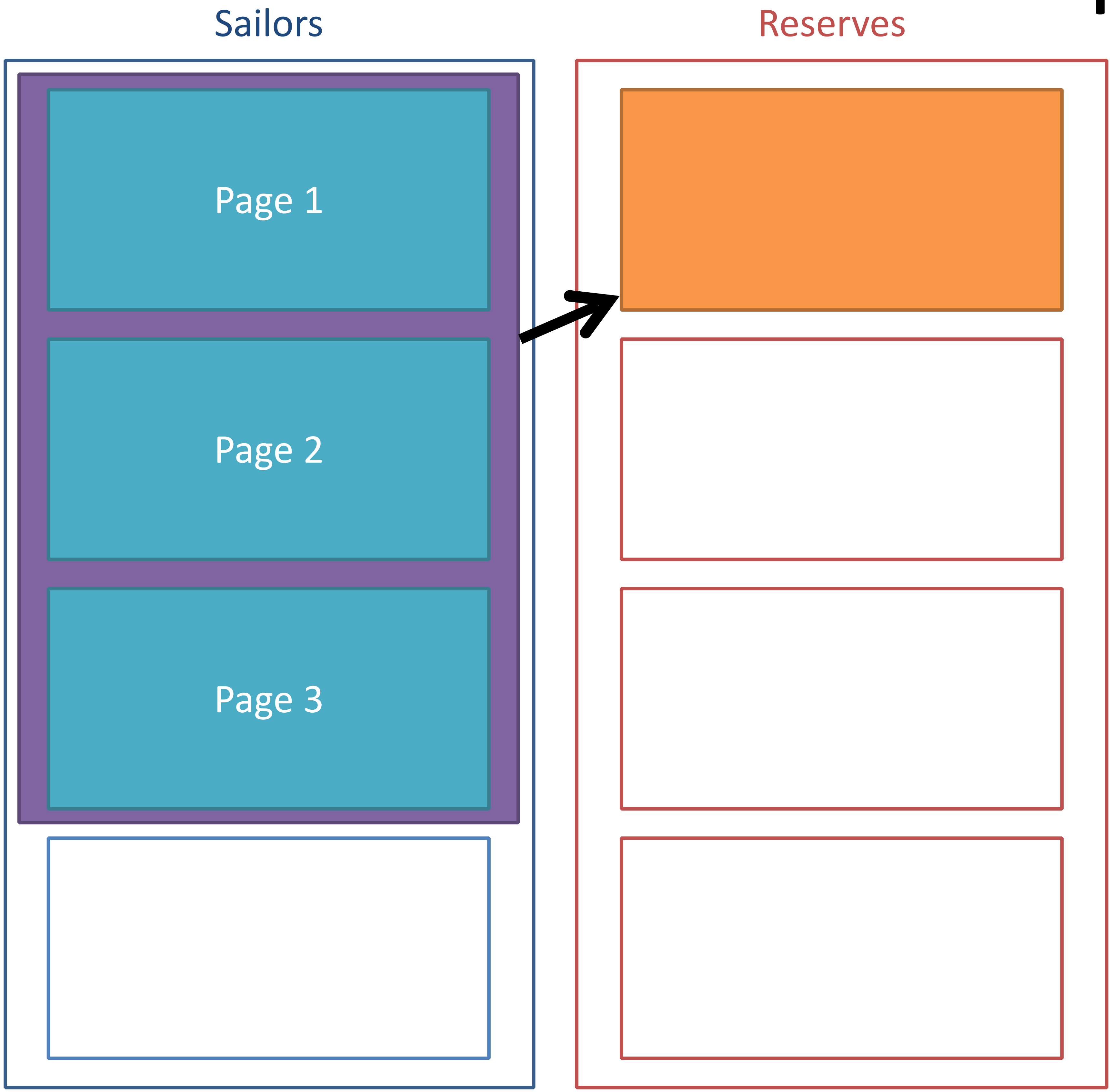


Key idea:
Take **k pages** of S and match with each page of R.

Steps:

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

Chunk Nested Loops Join

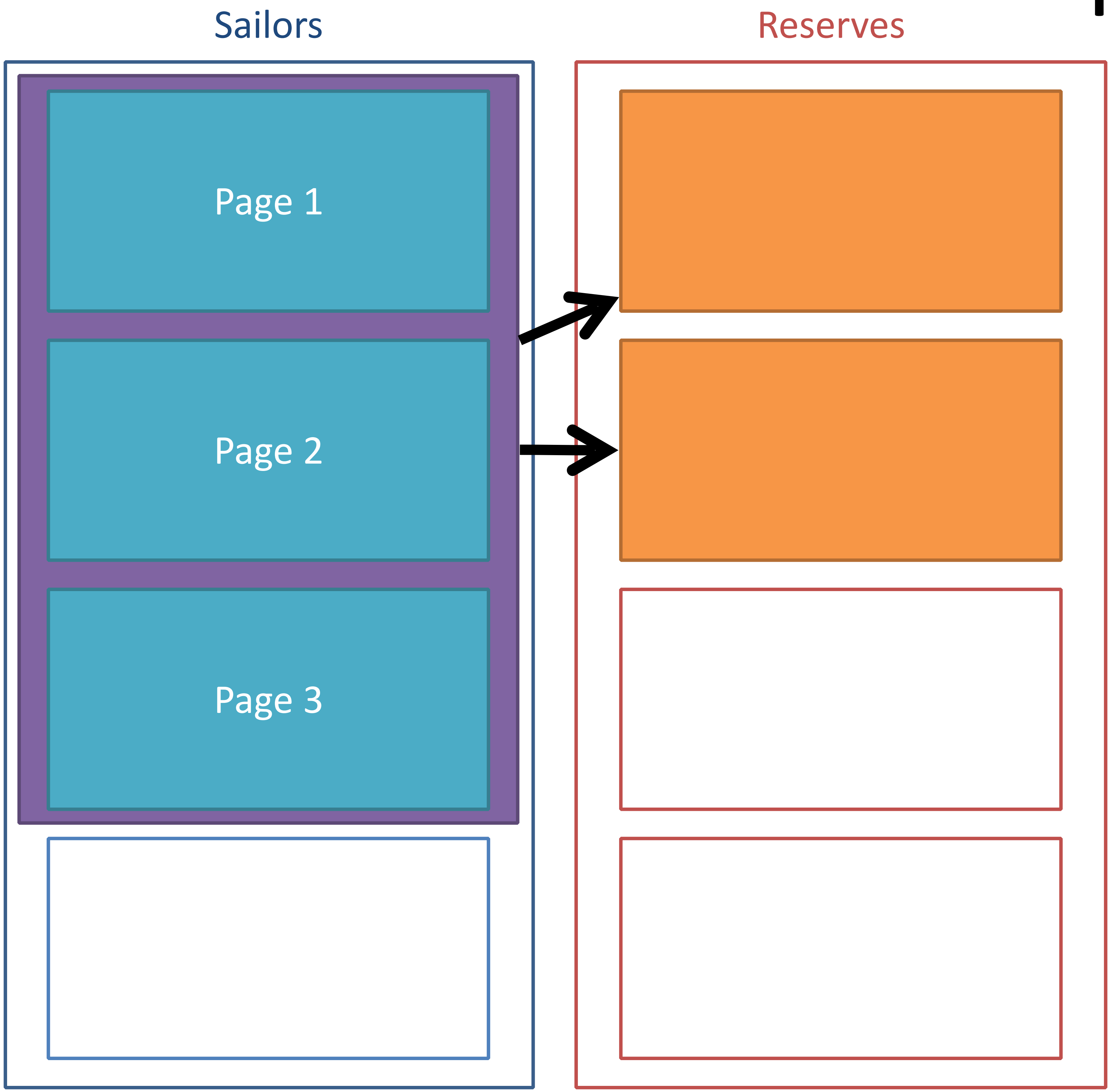


Key idea:
Take **k pages** of S and match with each page of R.

Steps:

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

Chunk Nested Loops Join



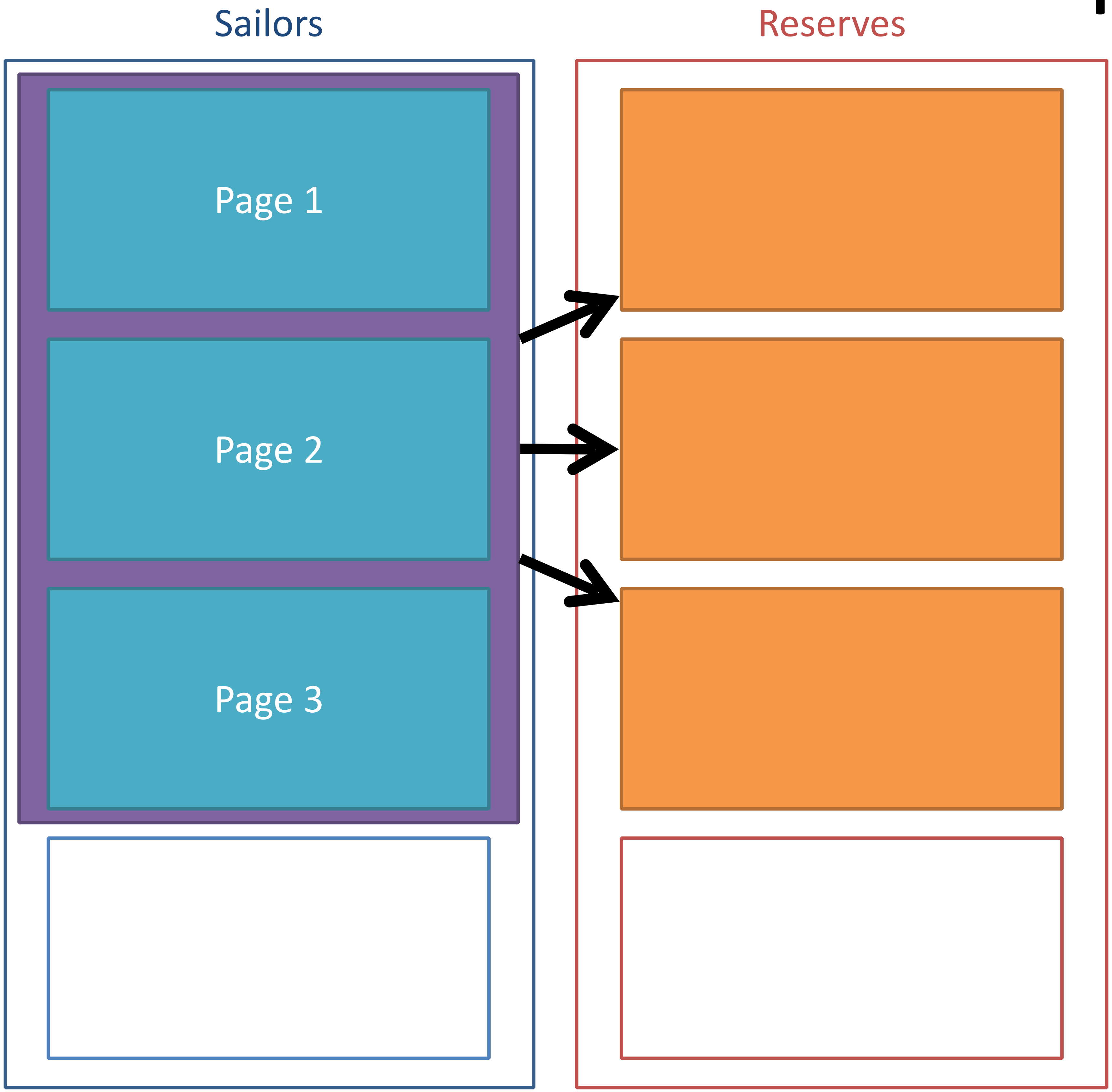
Key idea:

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



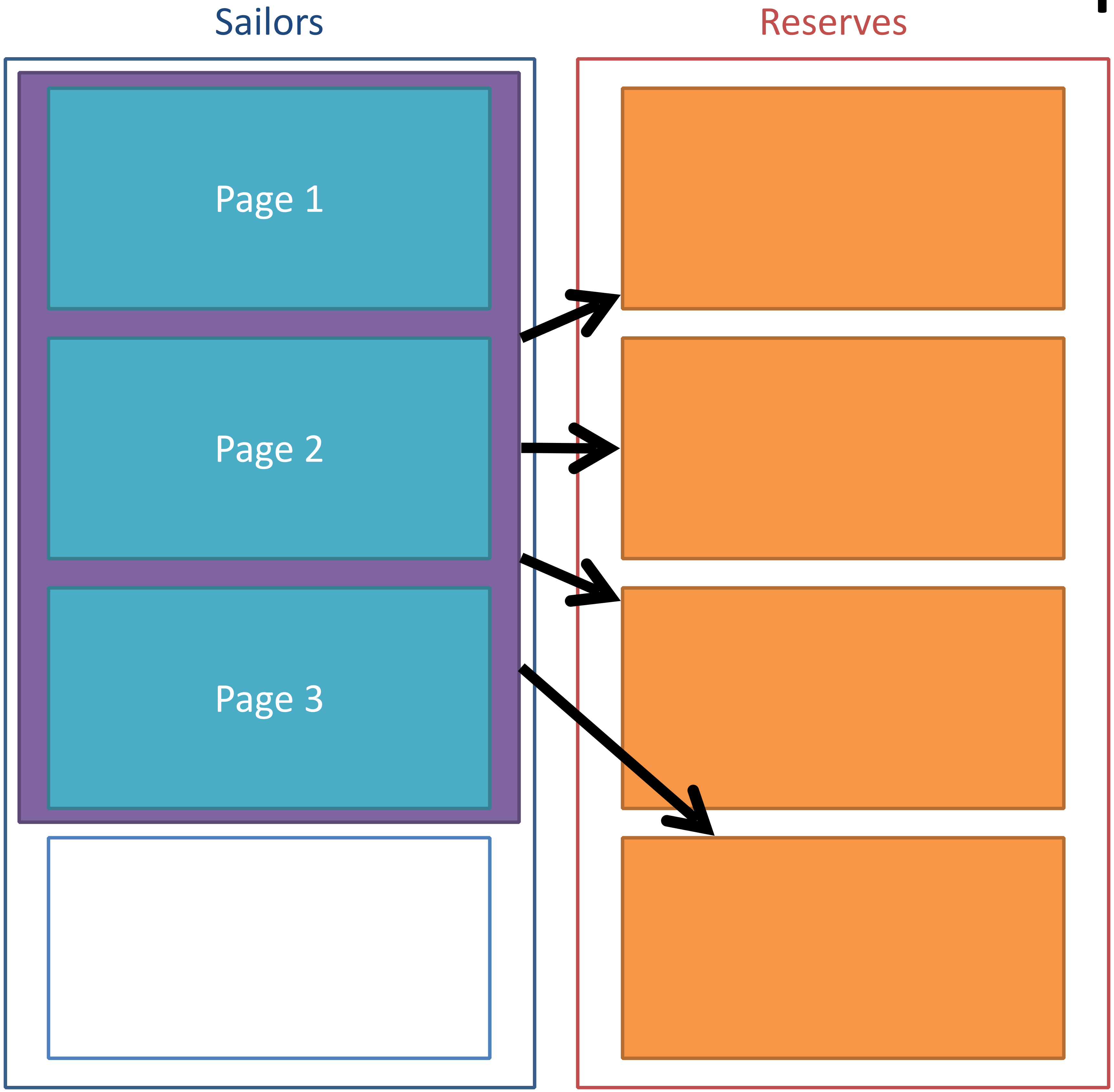
Key idea:

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



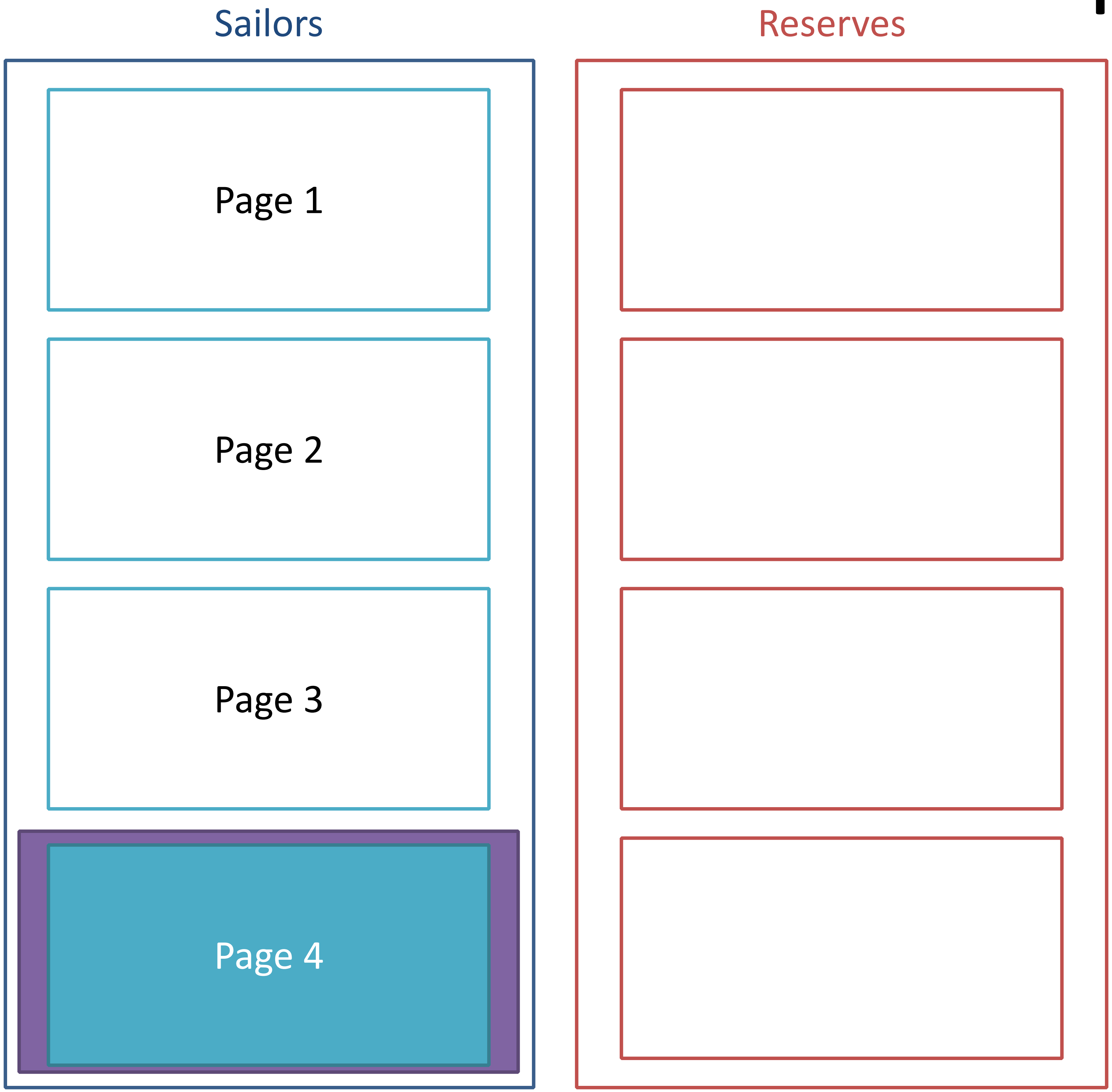
Key idea:

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:

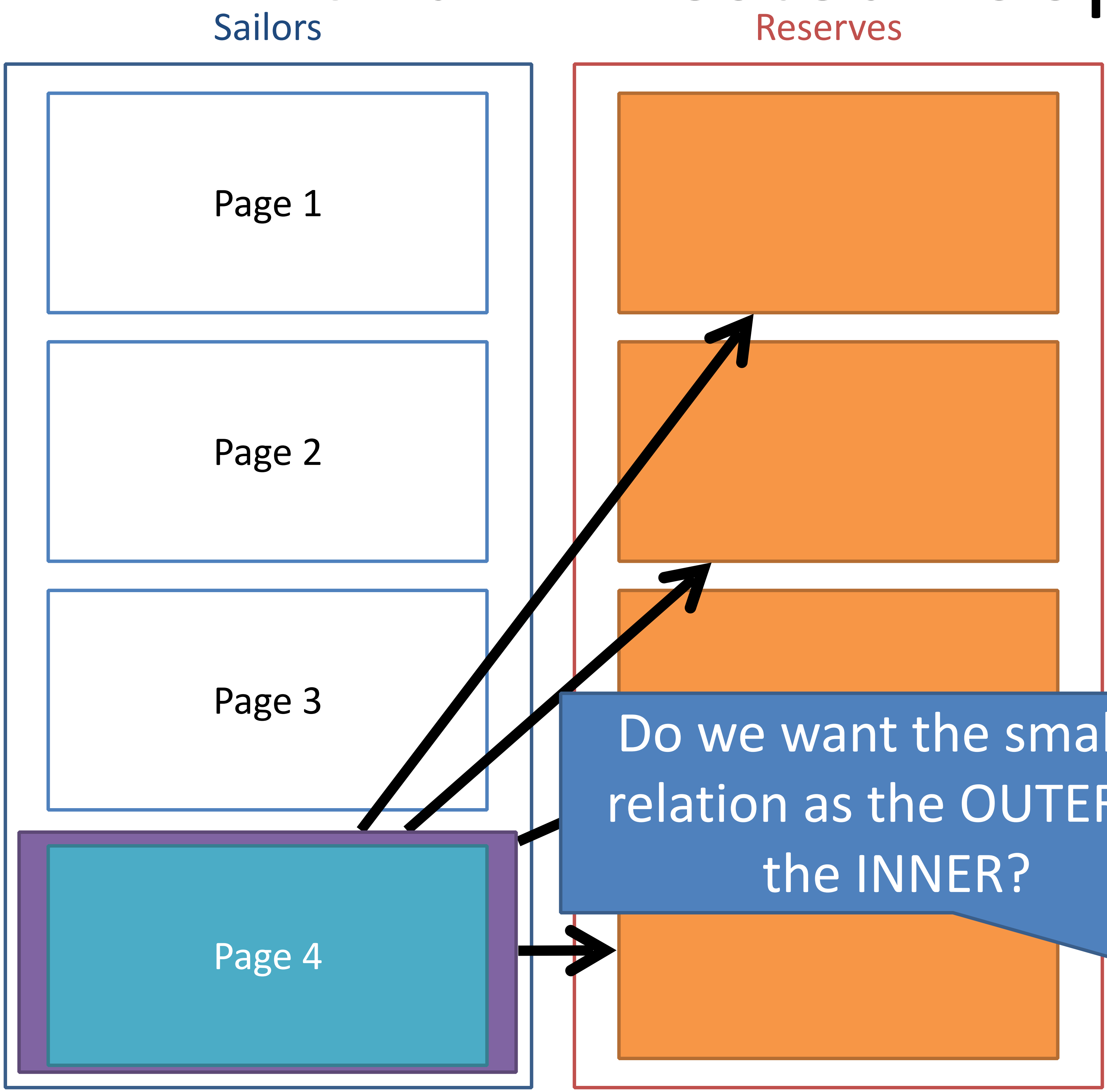
Take **k pages** of S and match with each page of R.

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:

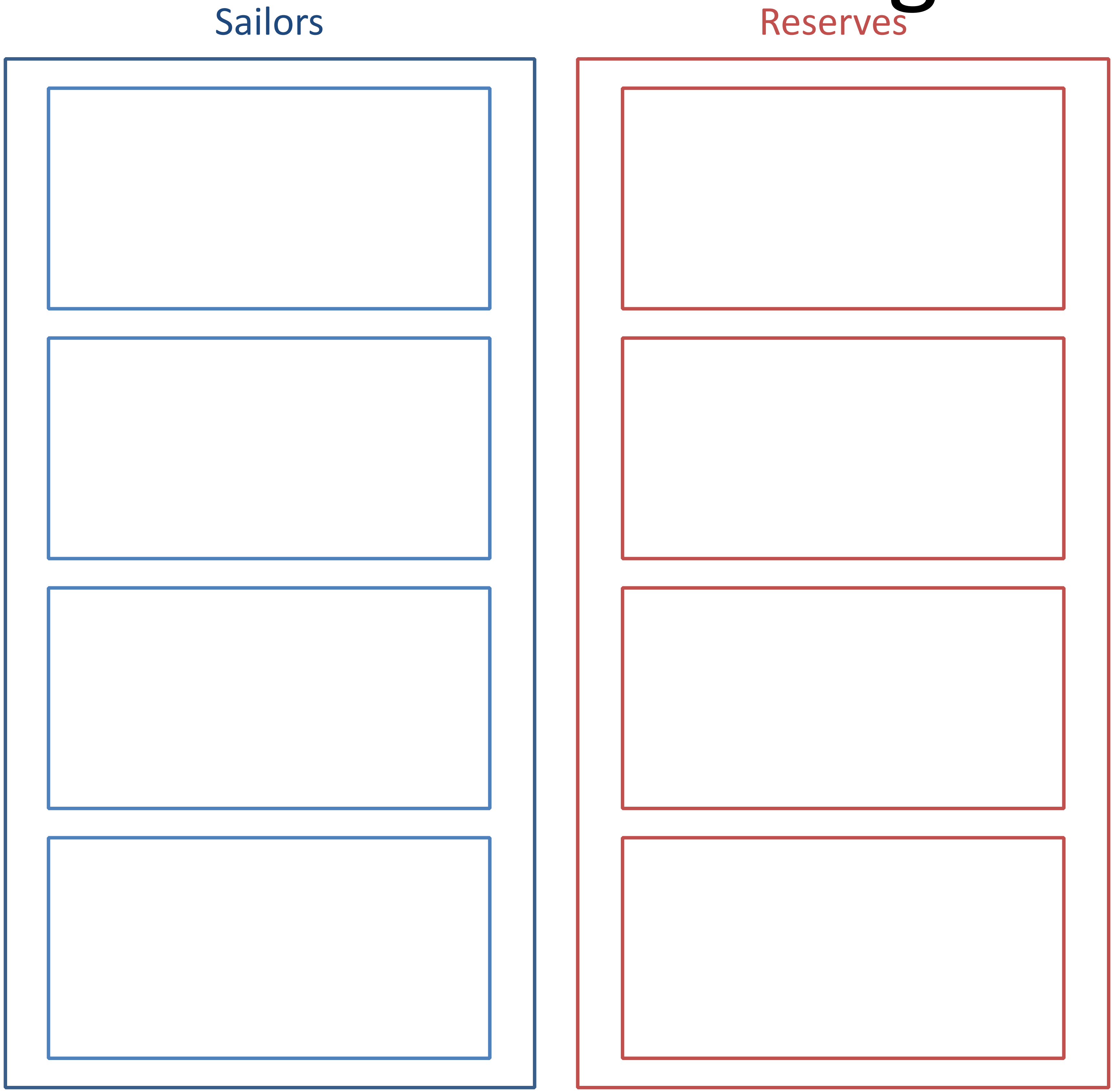
1. Get **k** pages of S.
2. Iterate through each page in R.

Compare tuples in each.

s:

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

Sort-Merge Join



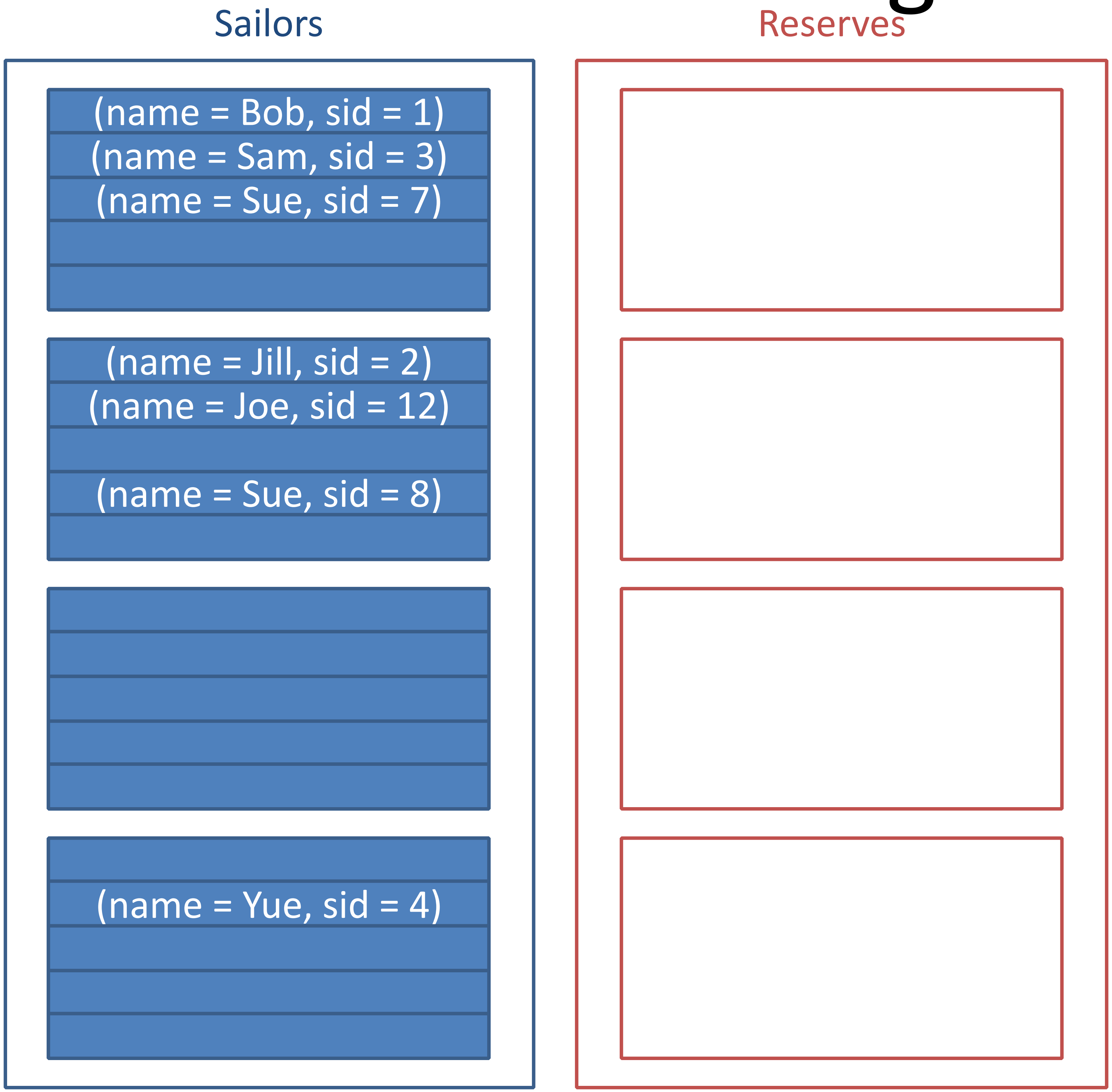
Key idea:

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

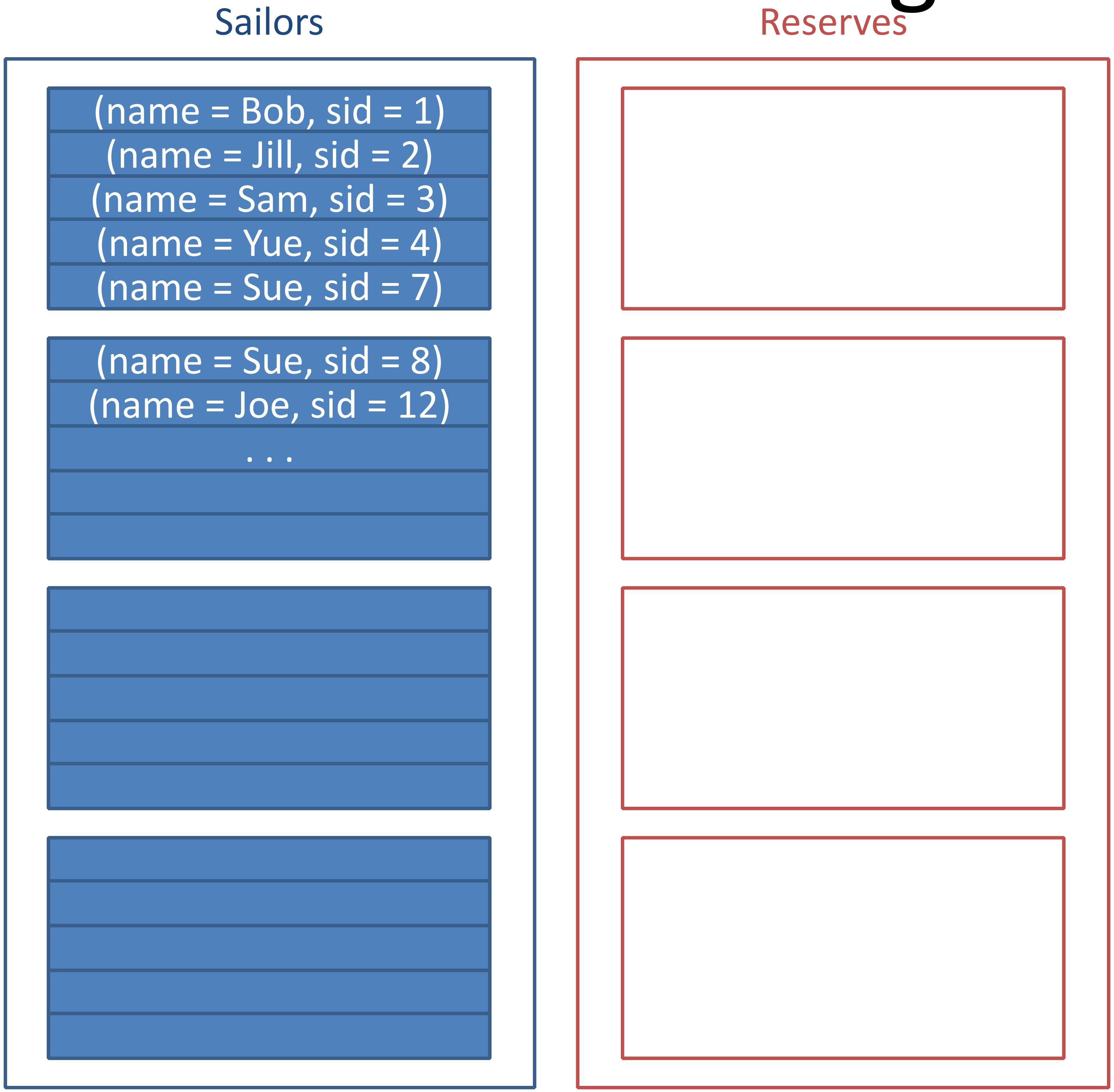
Key idea:

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

Steps:

1. Sort S and R.
2. “Zip” or merge.

Sort-Merge Join



(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)
. . .

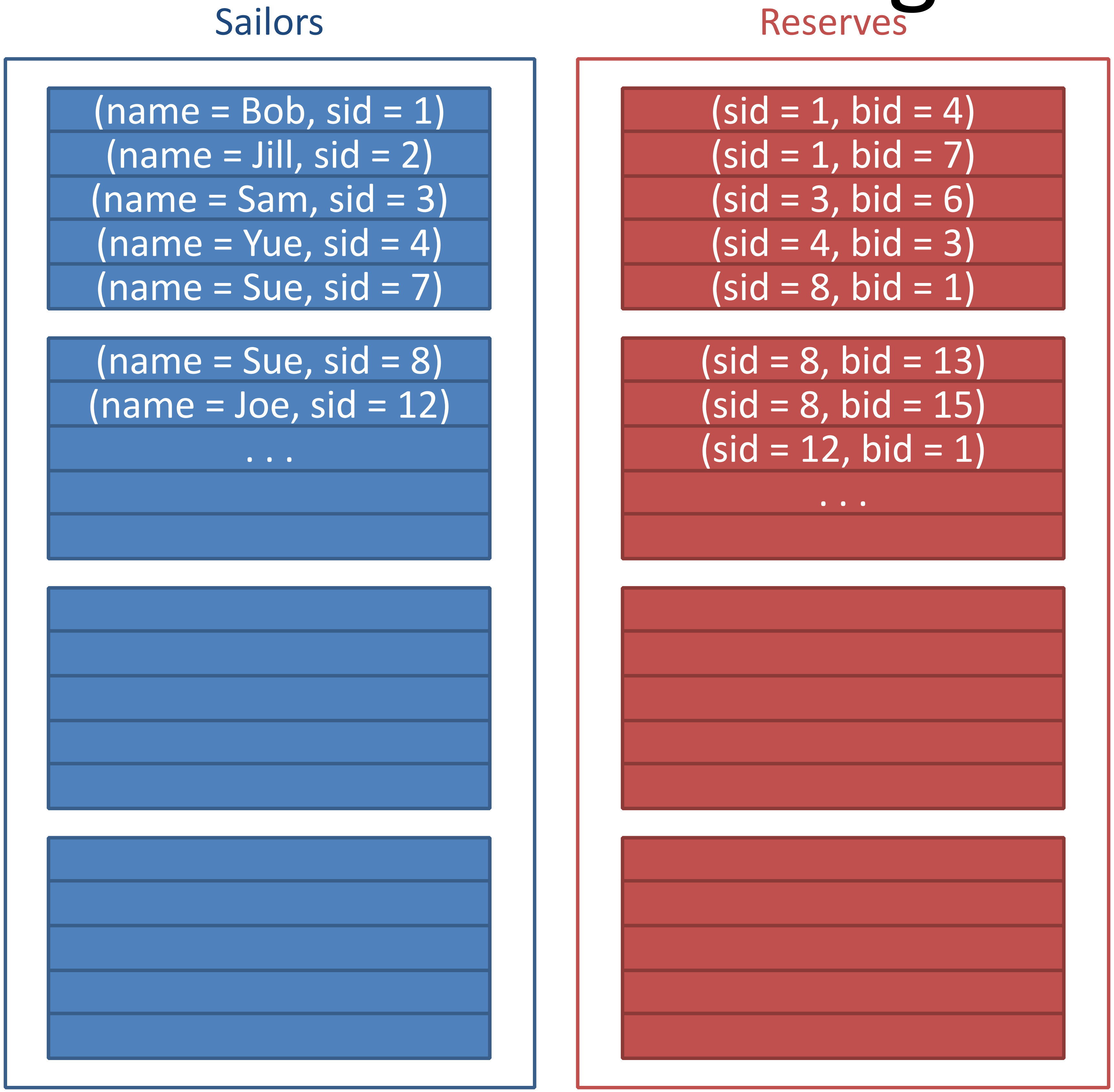
Key idea:

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)
. . .

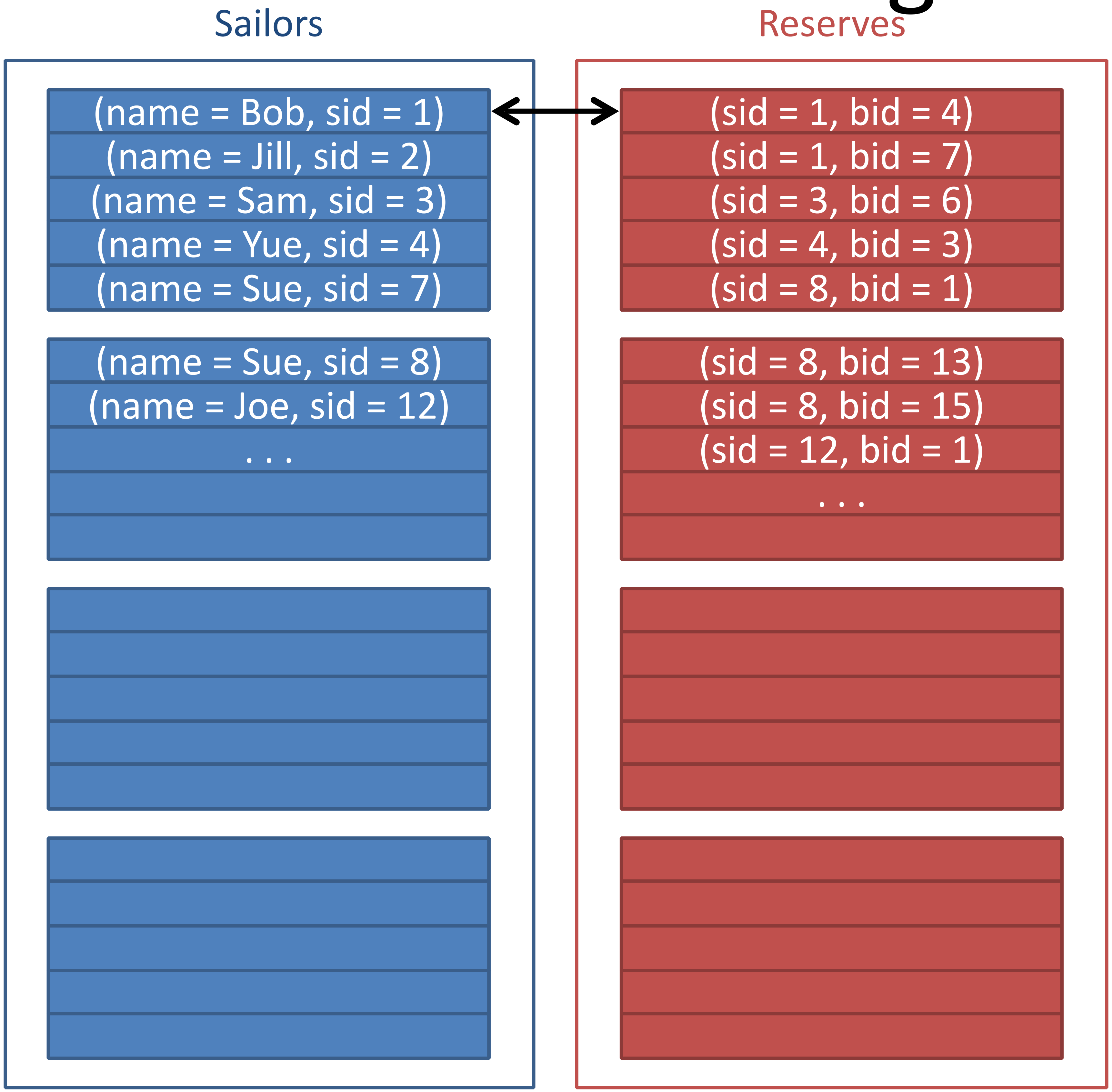
Key idea:

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!

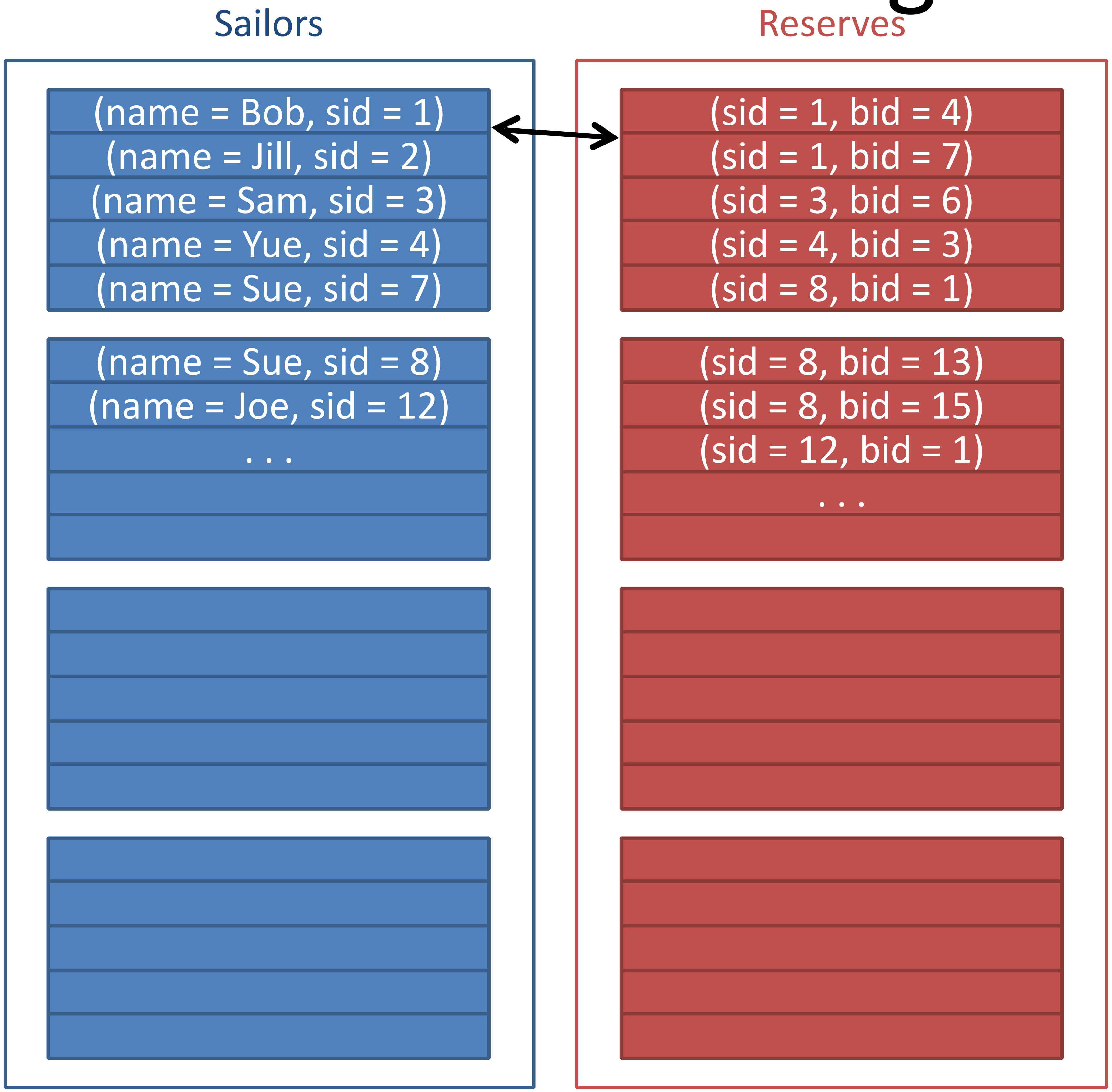
Steps:

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!

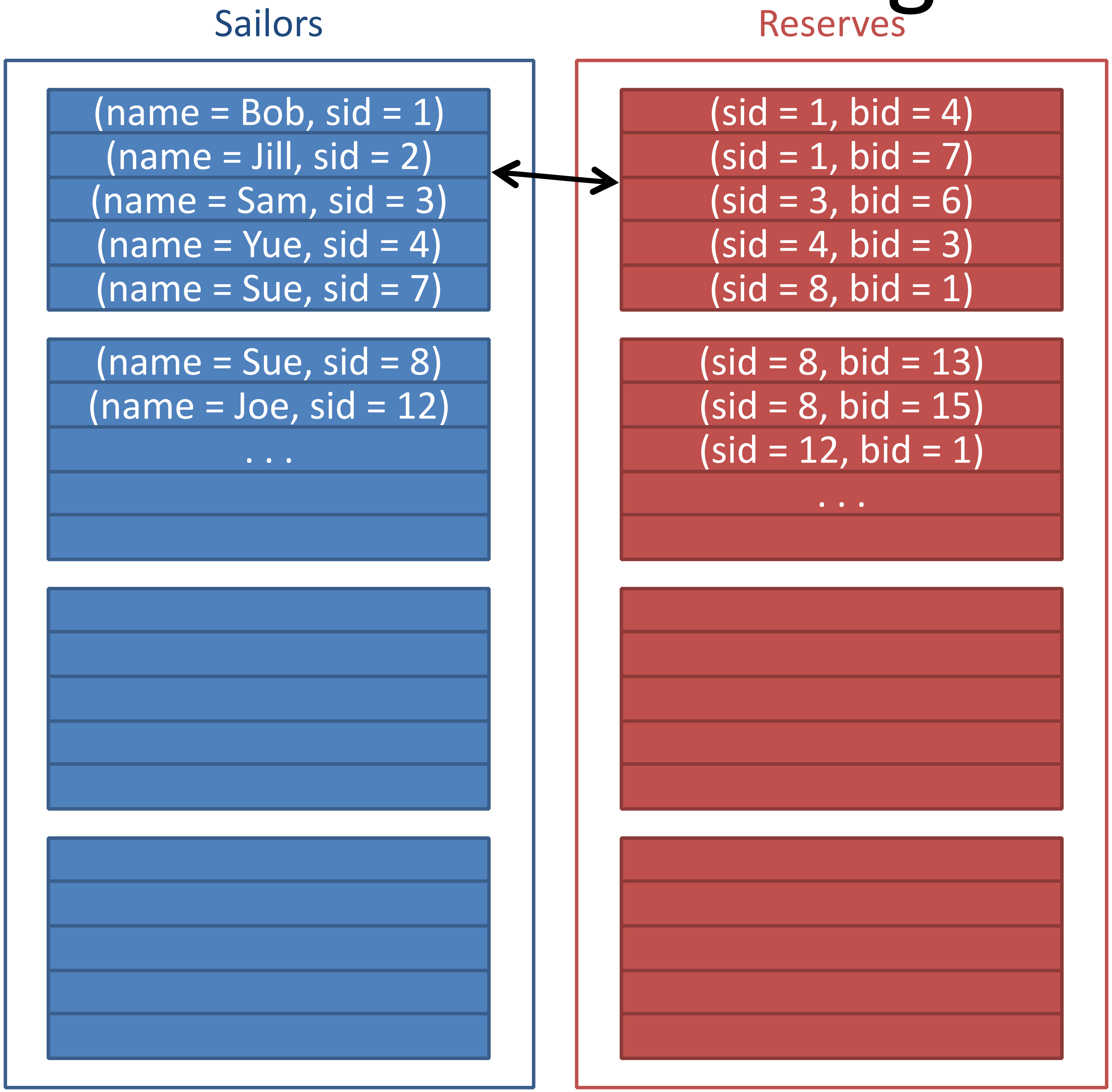
Steps:

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!

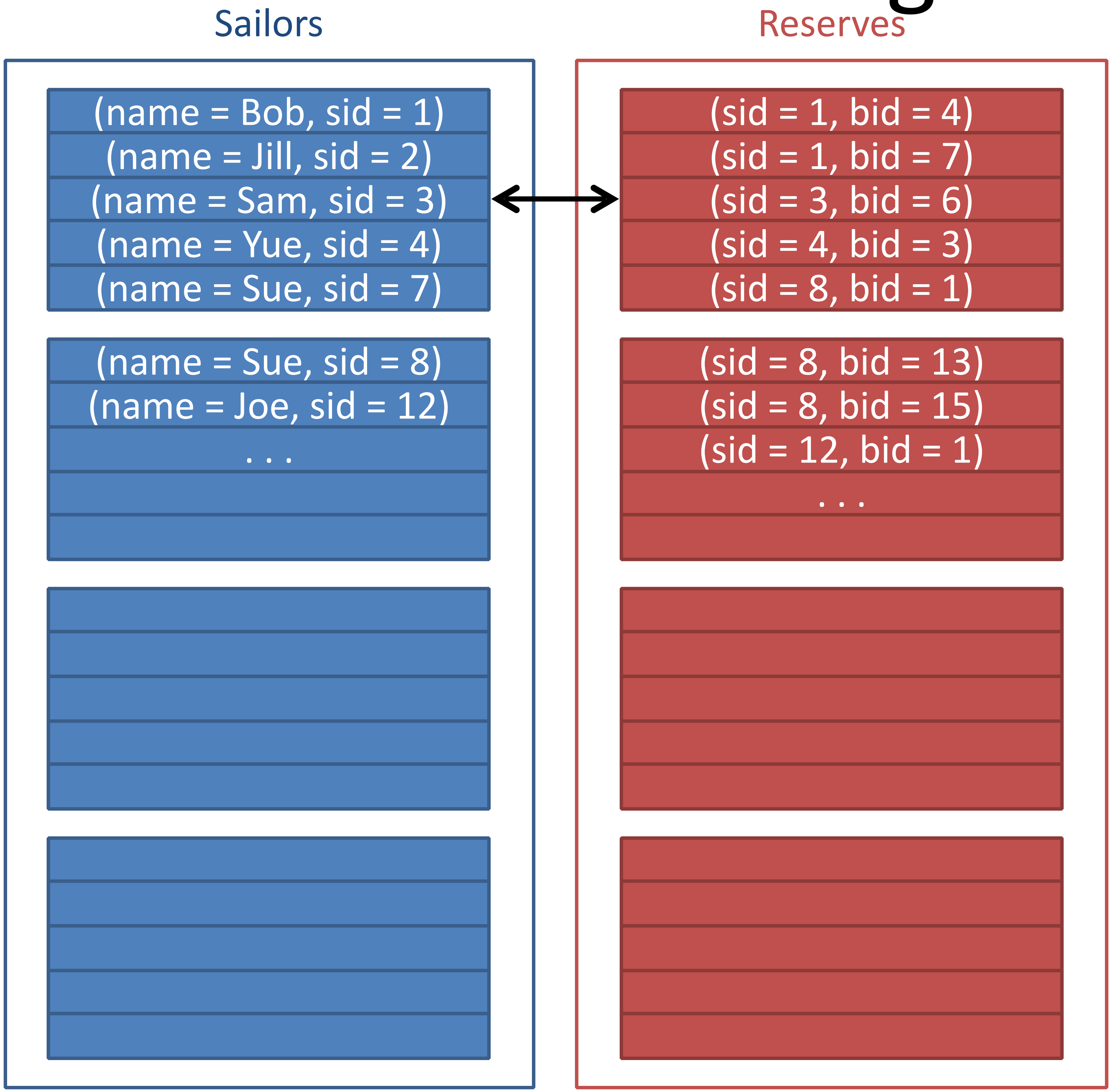
Steps:

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!

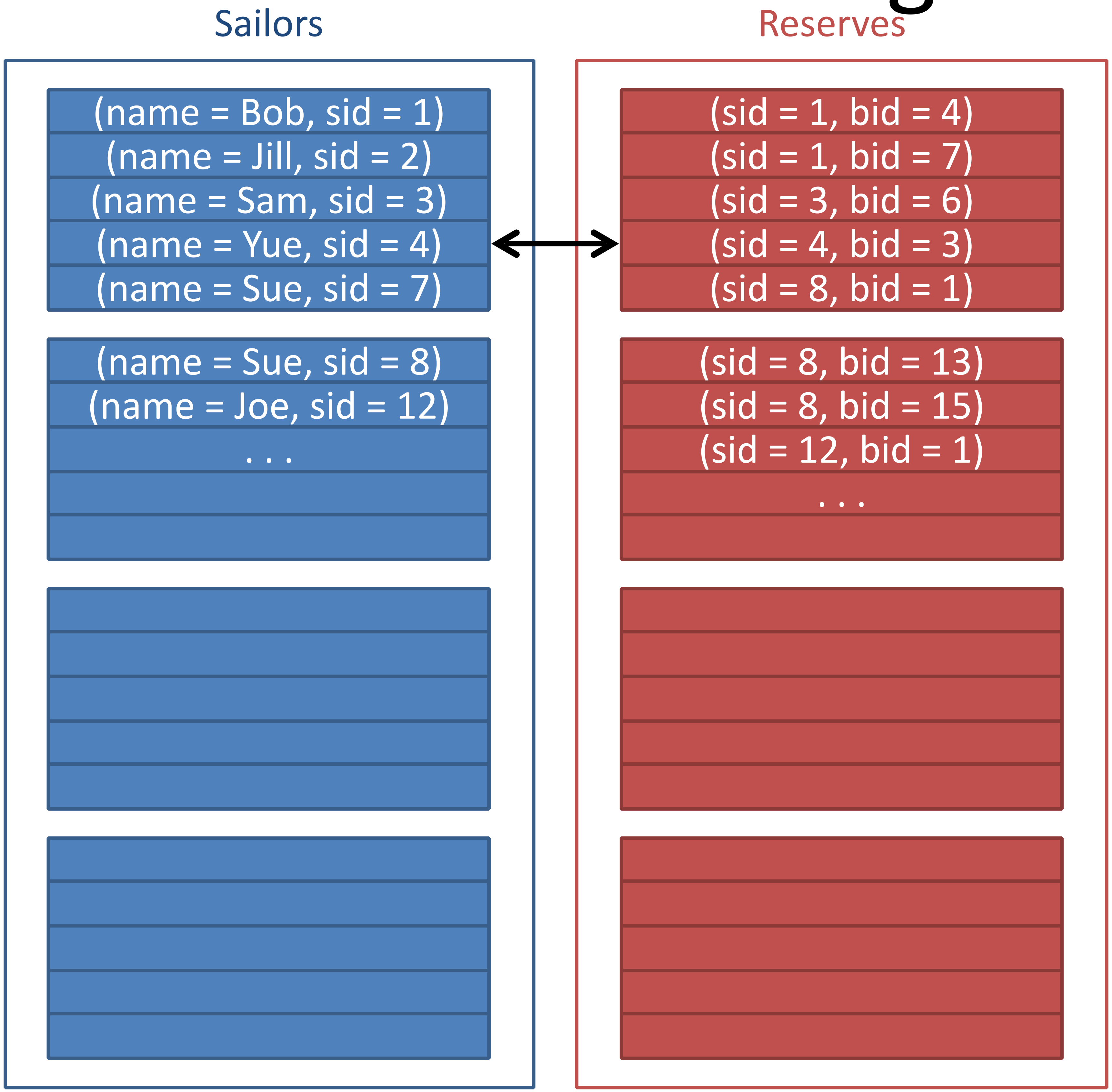
Steps:

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



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!

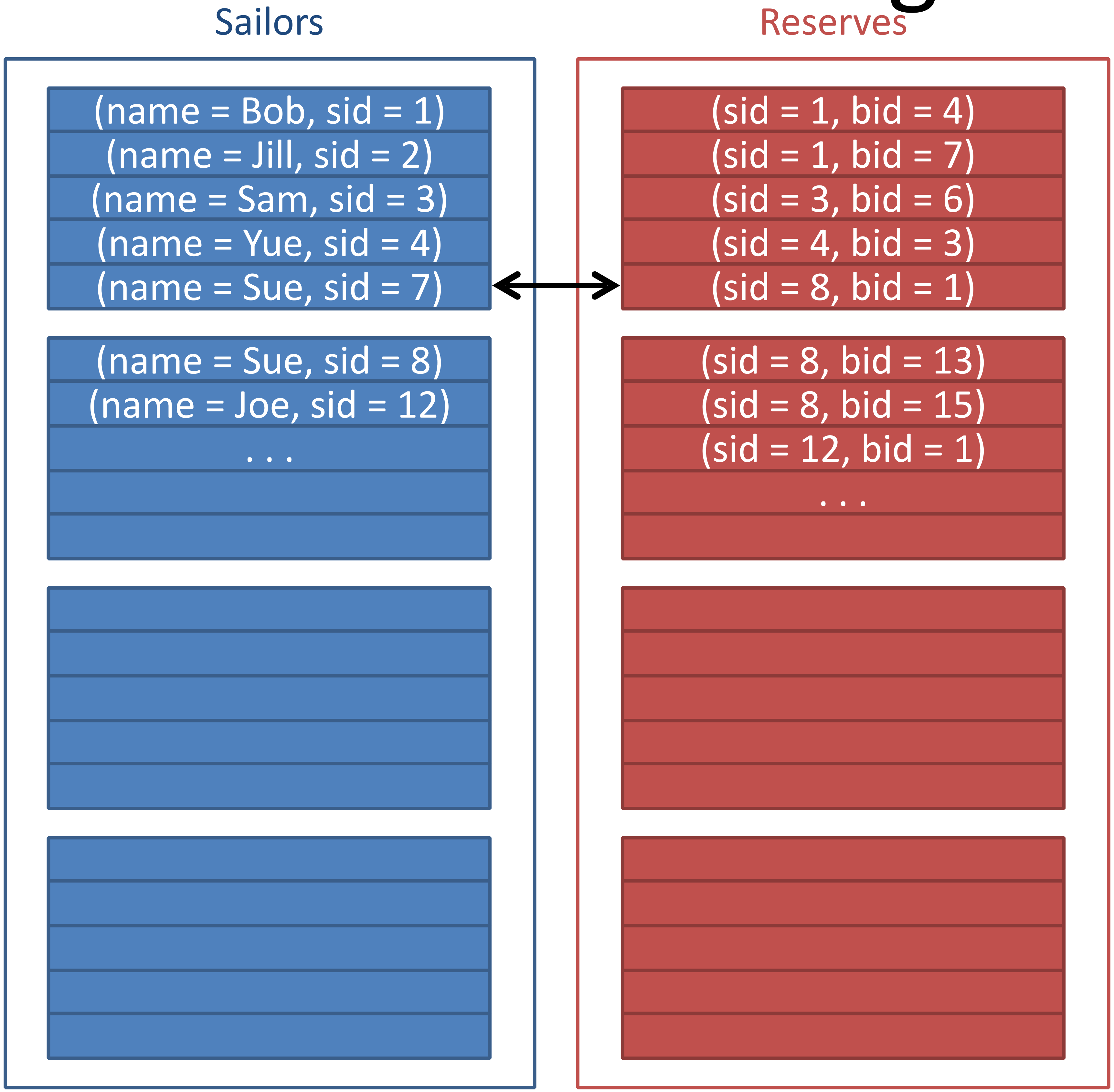
Steps:

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

(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!

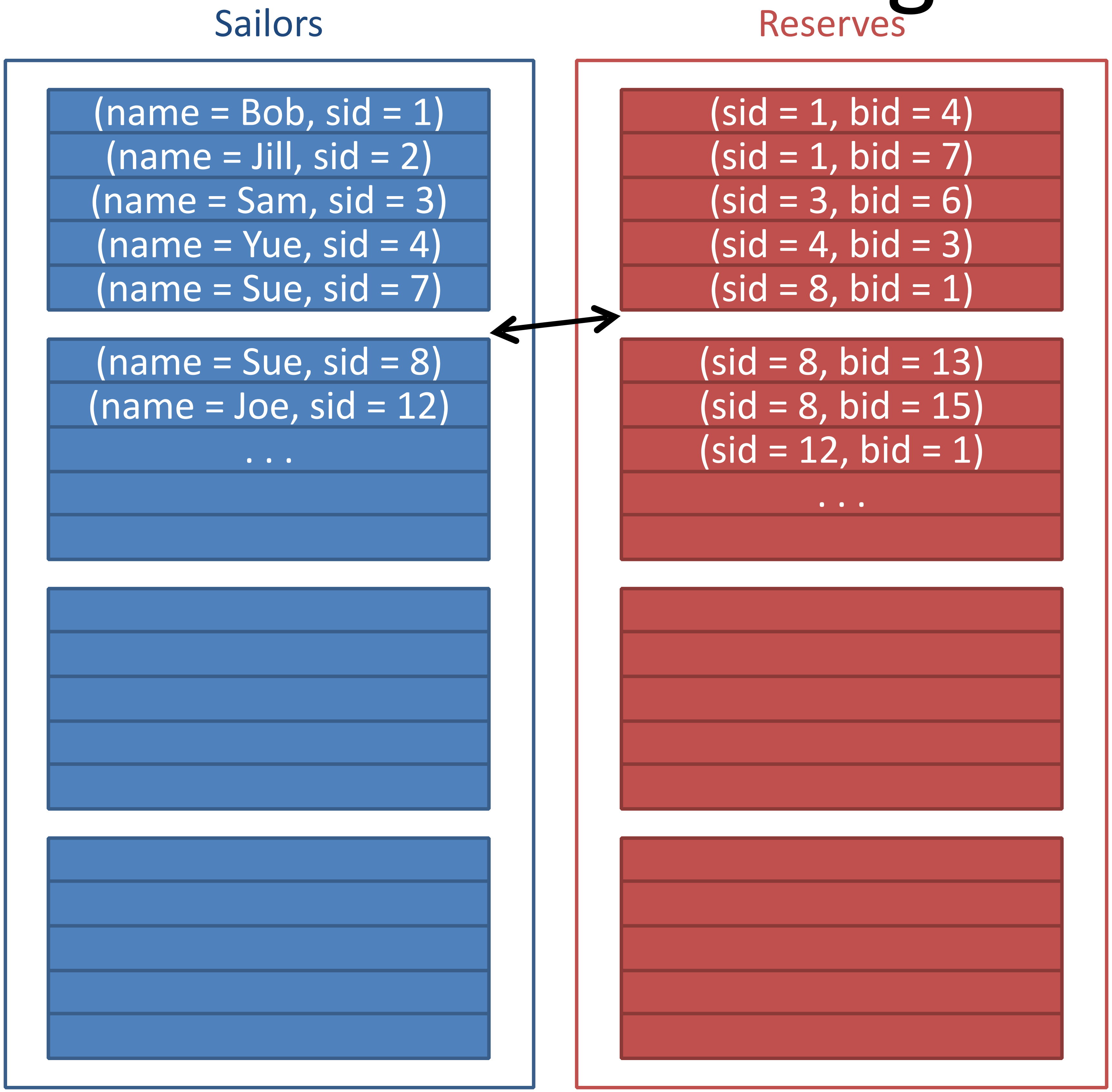
Steps:

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

(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!

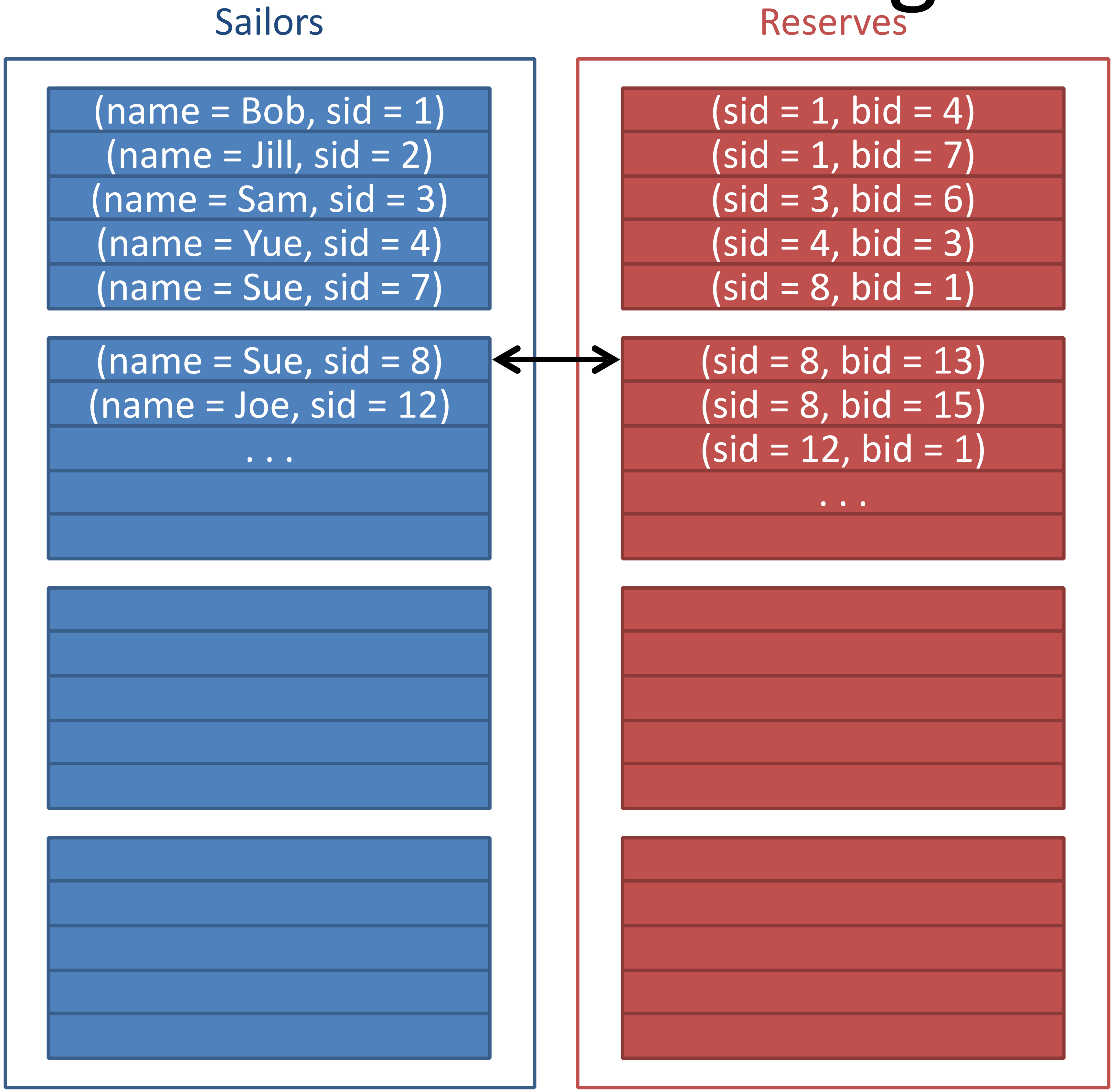
Steps:

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

(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!

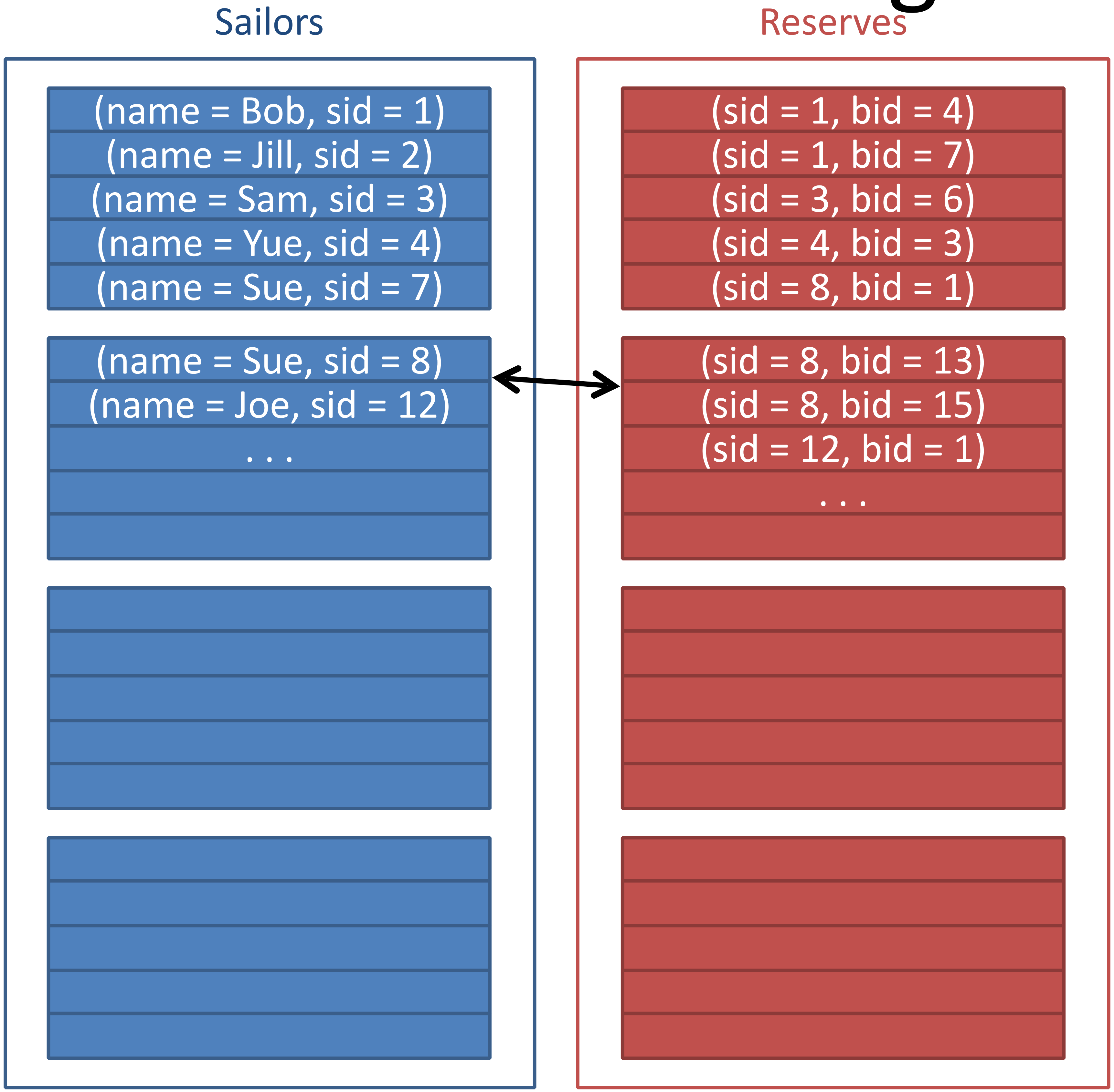
Steps:

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

(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!

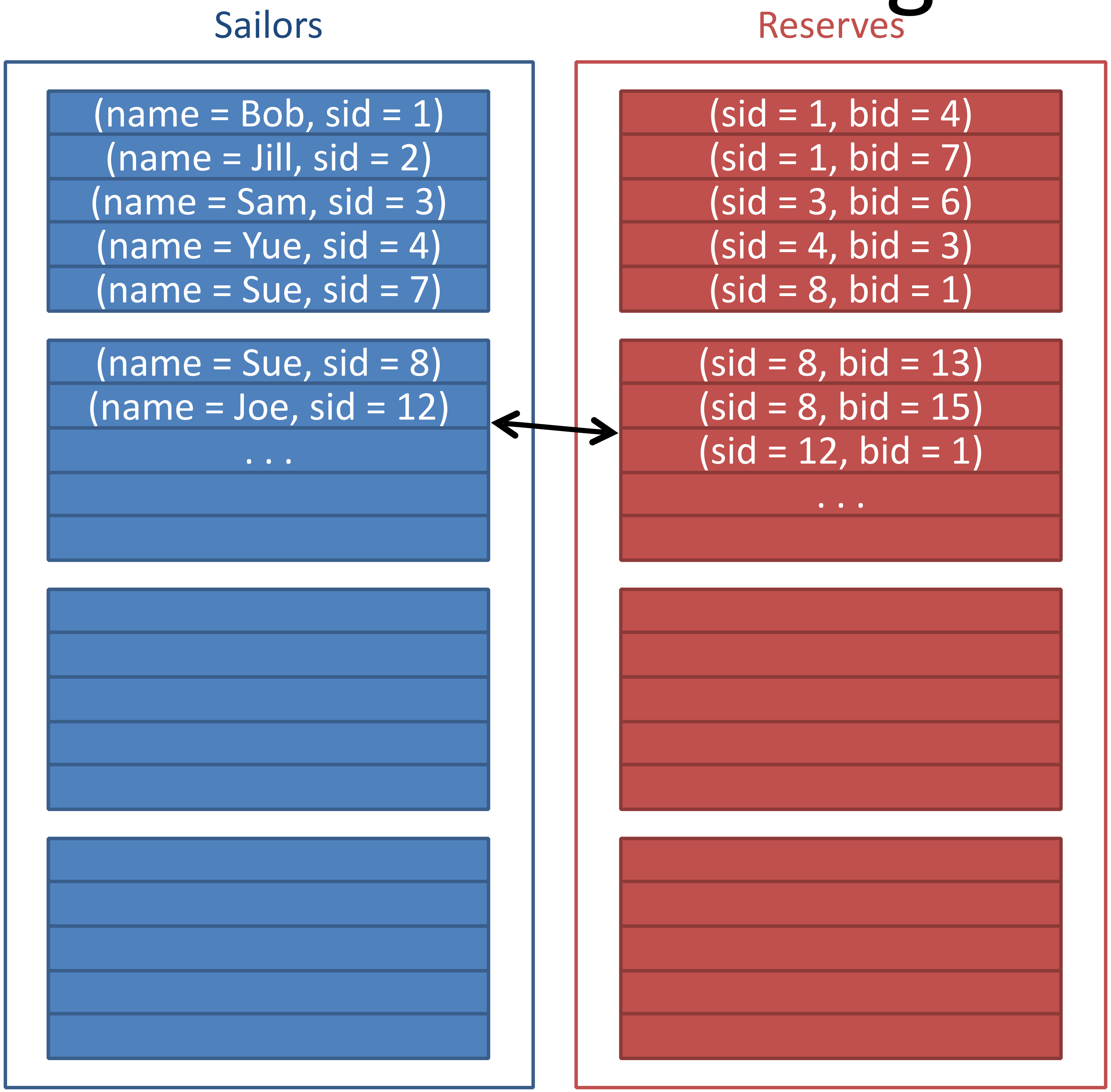
Steps:

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

(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:

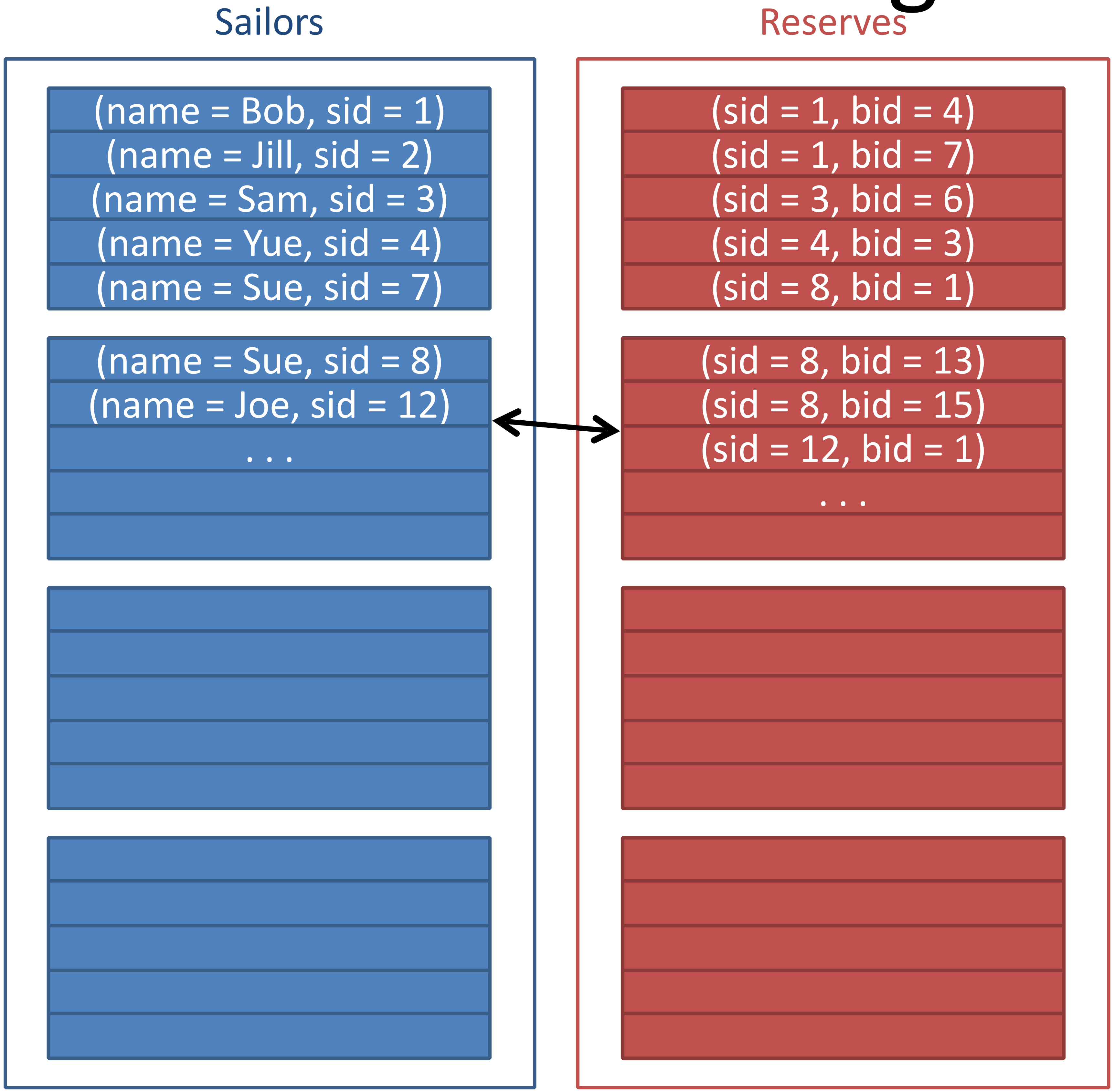
1. Sort S and R.
2. “Zip” or merge.

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.

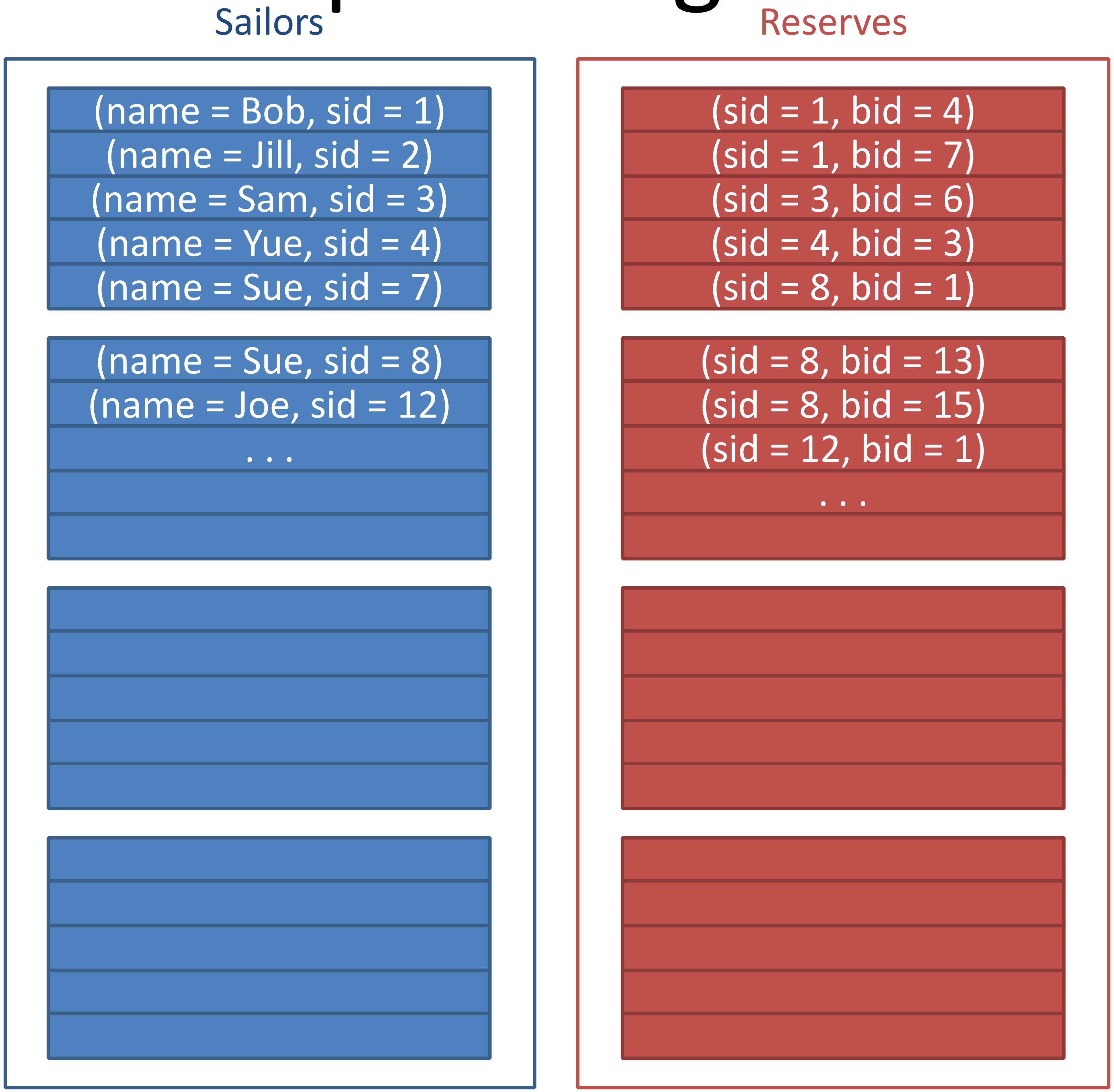
I/Os:

~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)
. . .

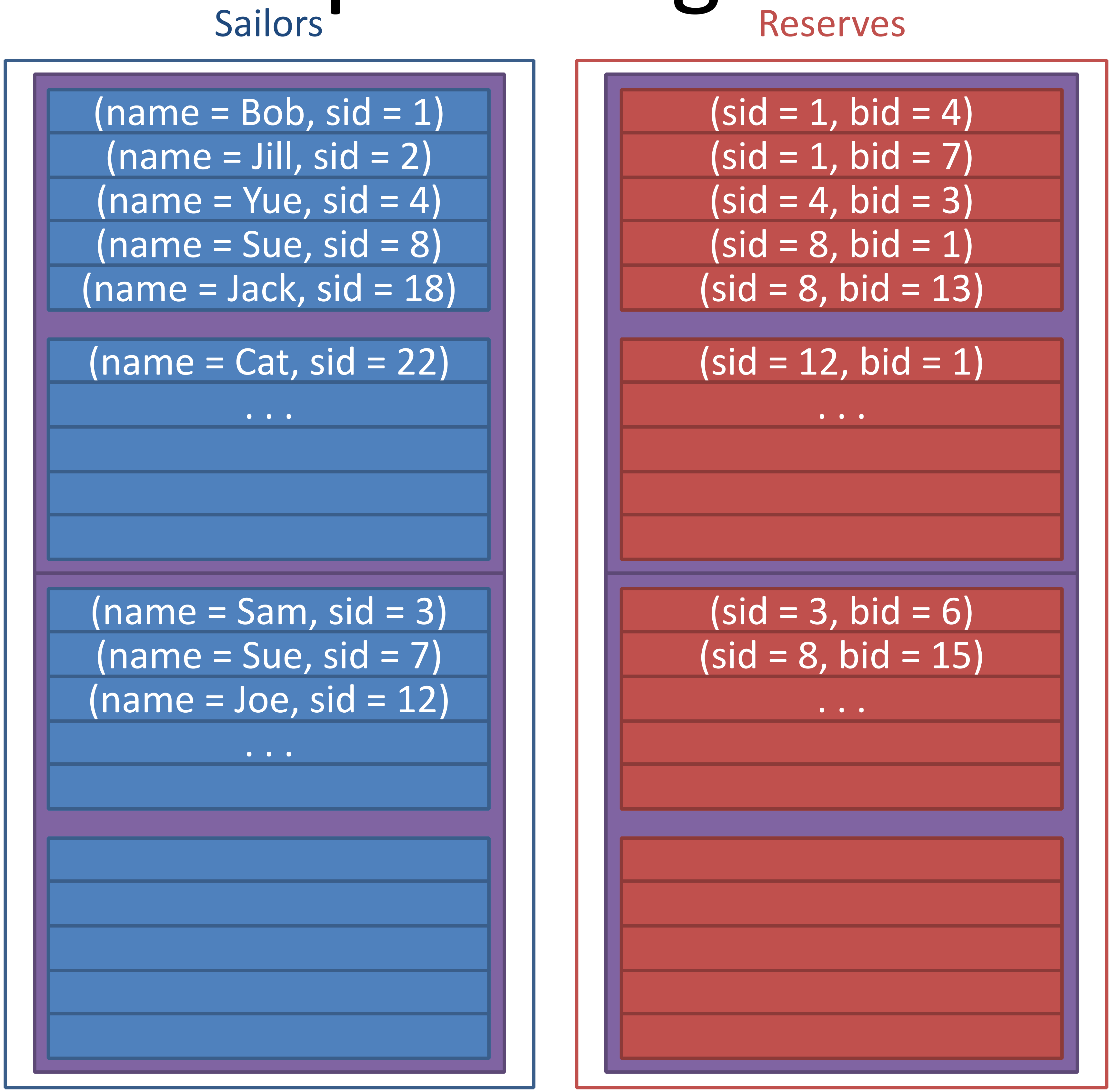
Key idea:

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



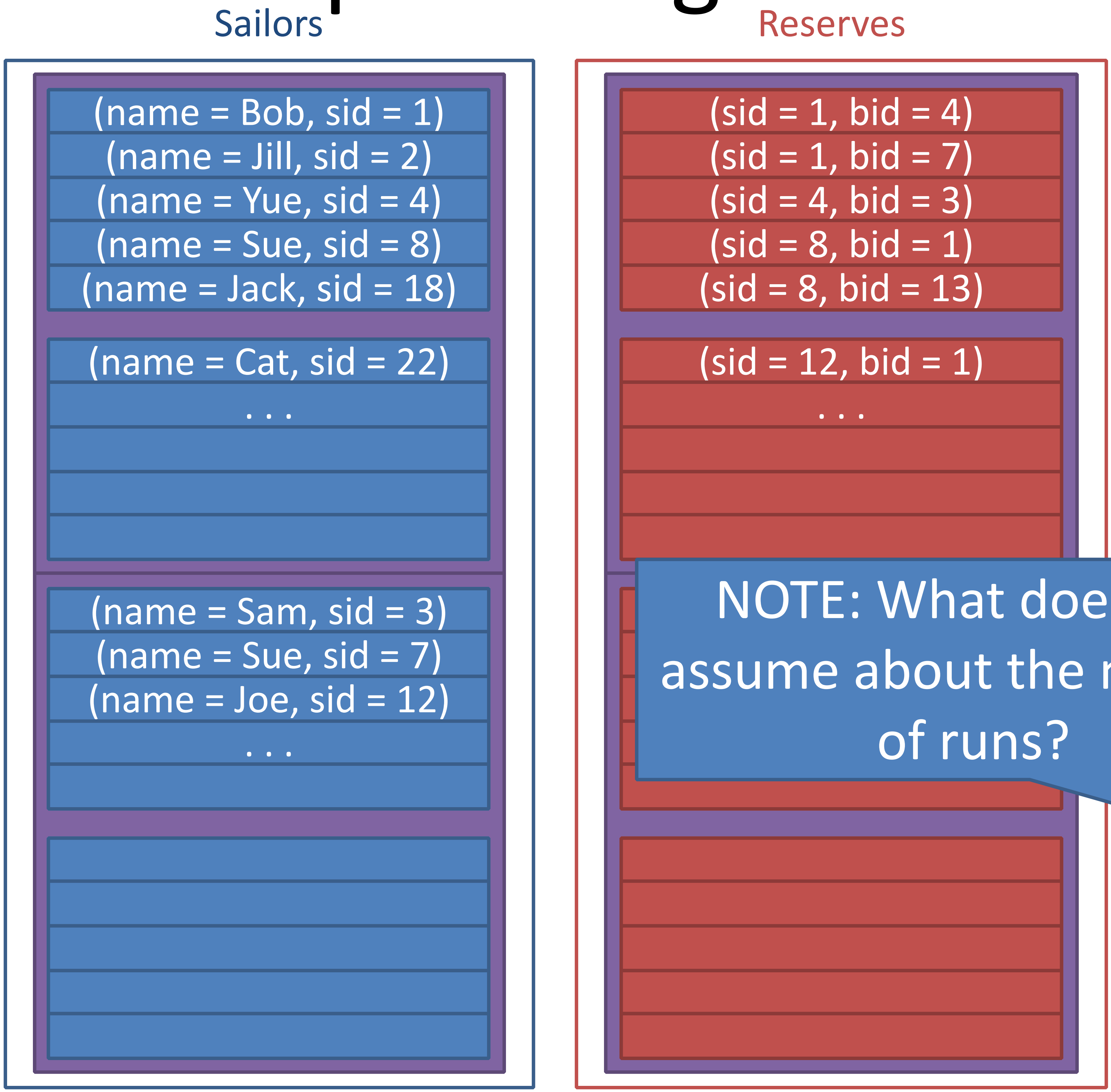
Key idea:

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	Reserves
(name = Bob, sid = 1)	(sid = 1, bid = 4)
(name = Jill, sid = 2)	(sid = 1, bid = 7)
(name = Yue, sid = 4)	(sid = 4, bid = 3)
(name = Sue, sid = 8)	(sid = 8, bid = 1)
(name = Jack, sid = 18)	(sid = 8, bid = 13)
(name = Cat, sid = 22)	(sid = 12, bid = 1)
. . .	. . .
(name = Sam, sid = 3)	
(name = Sue, sid = 7)	
(name = Joe, sid = 12)	
. . .	

Key idea:

Internal Sort on both. Perform merge on all runs!

Steps:

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

ge all runs.

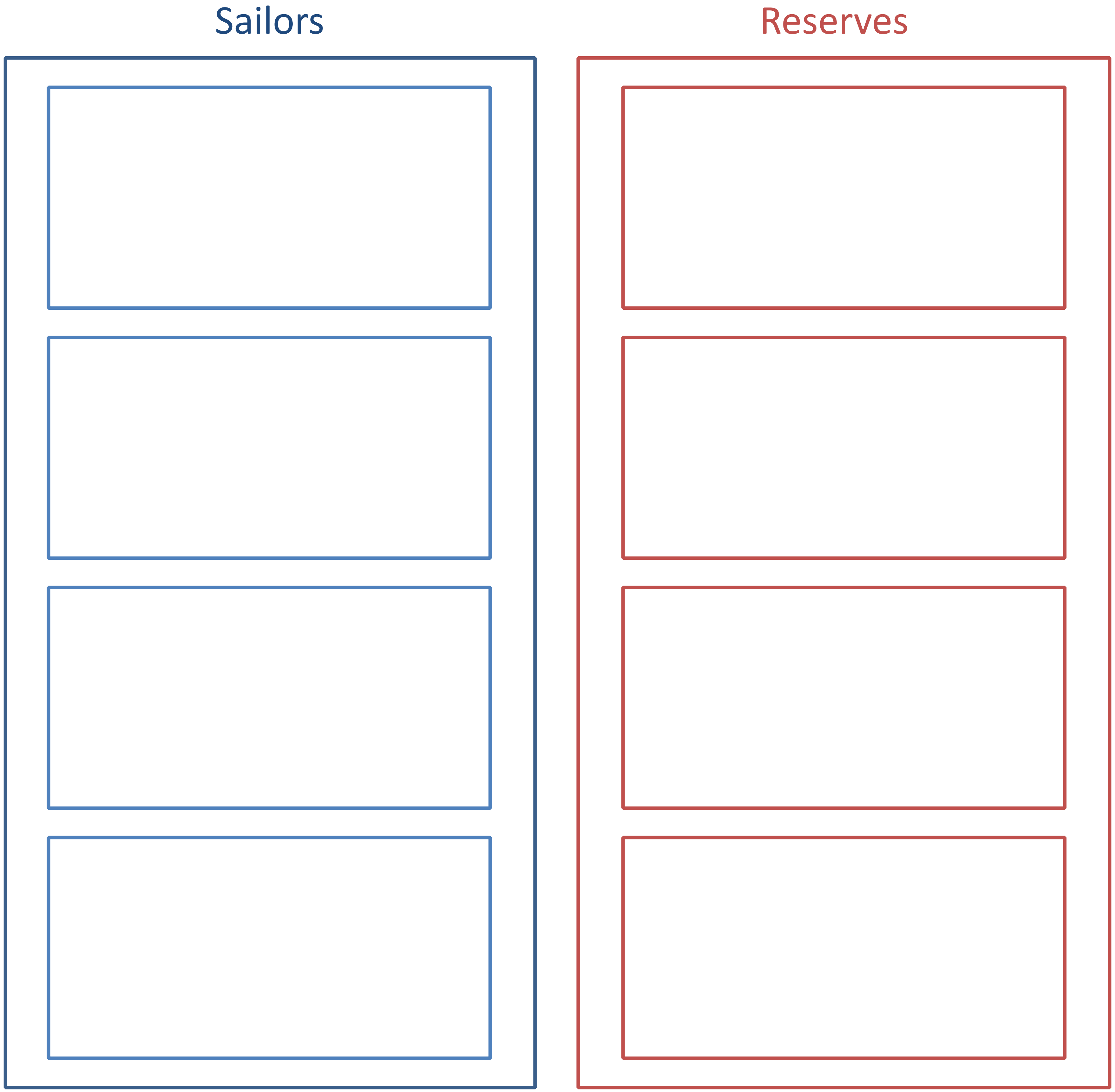
NOTE: What does this assume about the number of runs?

~3([S] + [R])

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

Merging: [S]+[R]

Hash-Join



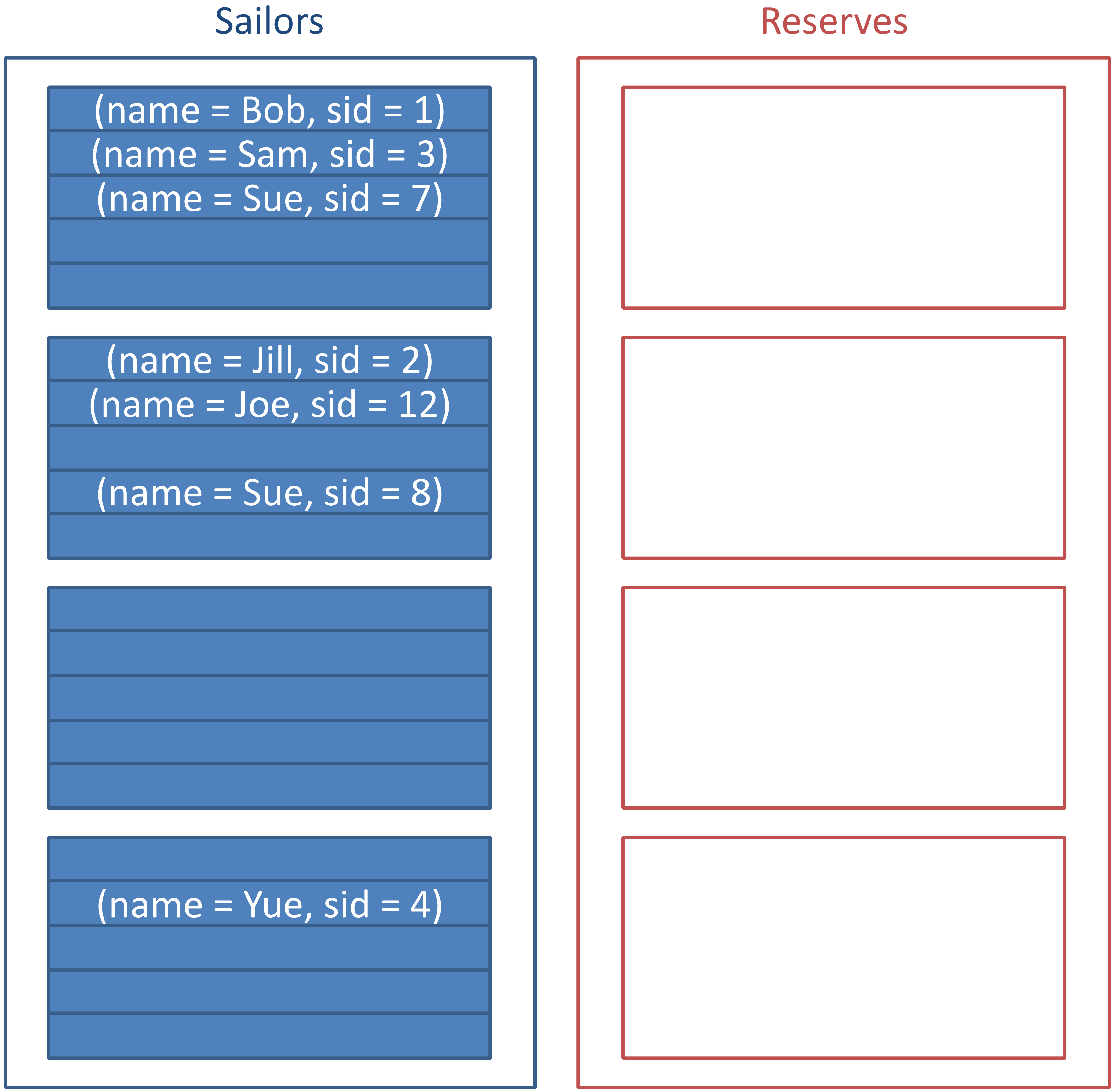
Key idea:

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:

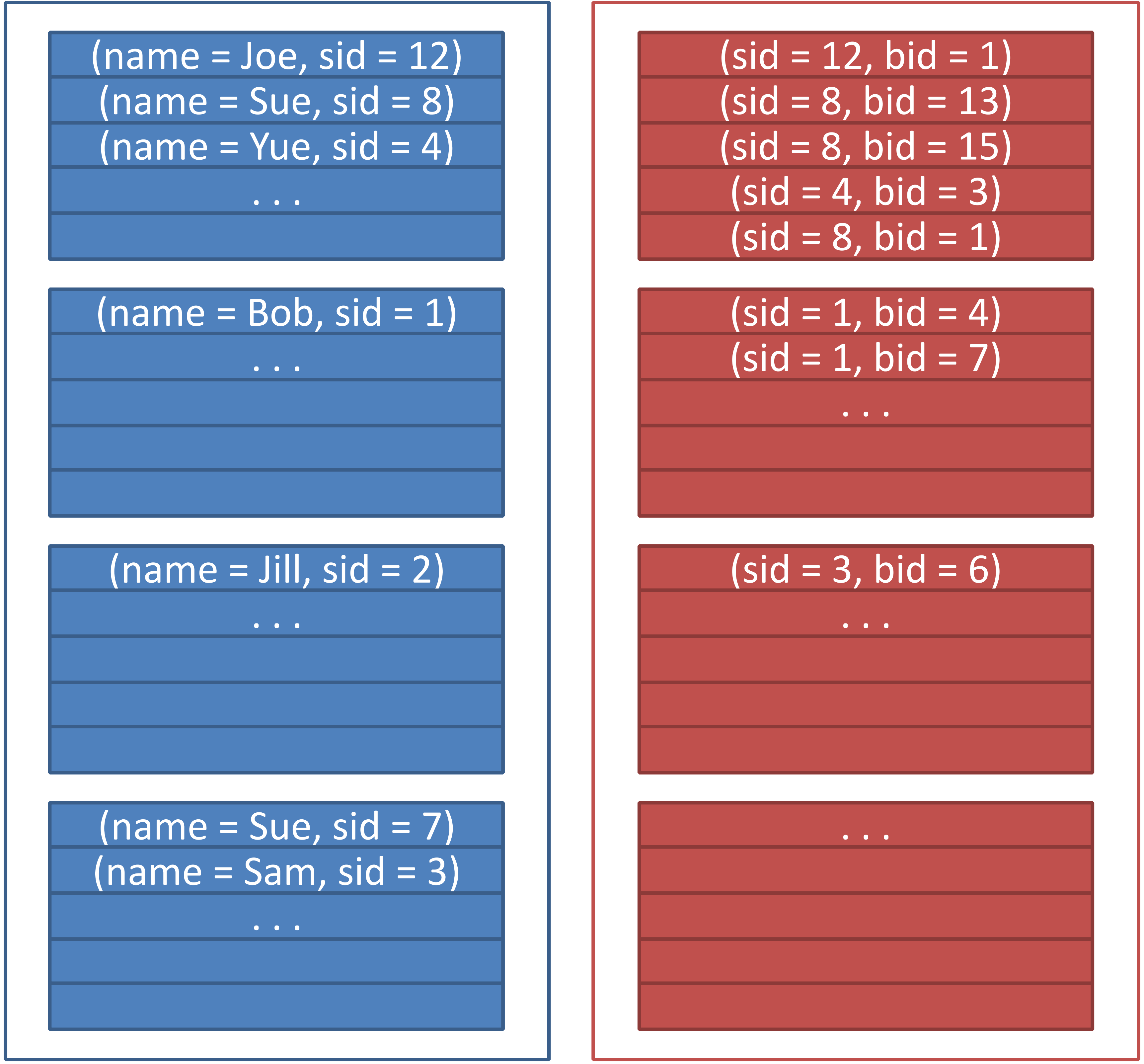
Partition S and R using same hash fn, then collect same partitions

Steps:

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

Hash function: **sid mod 4**

lash-Join



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

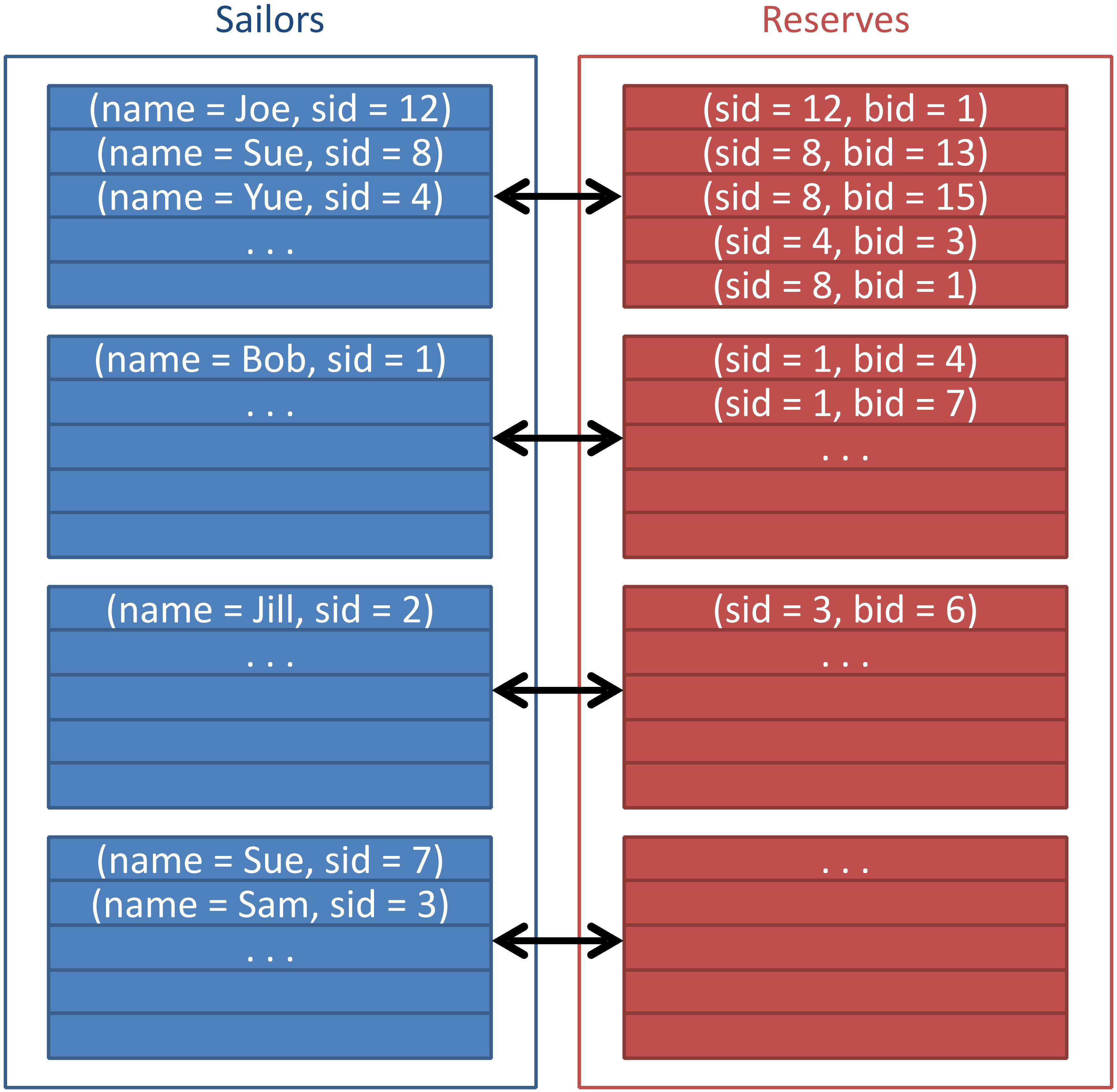
Key idea:

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 = 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:

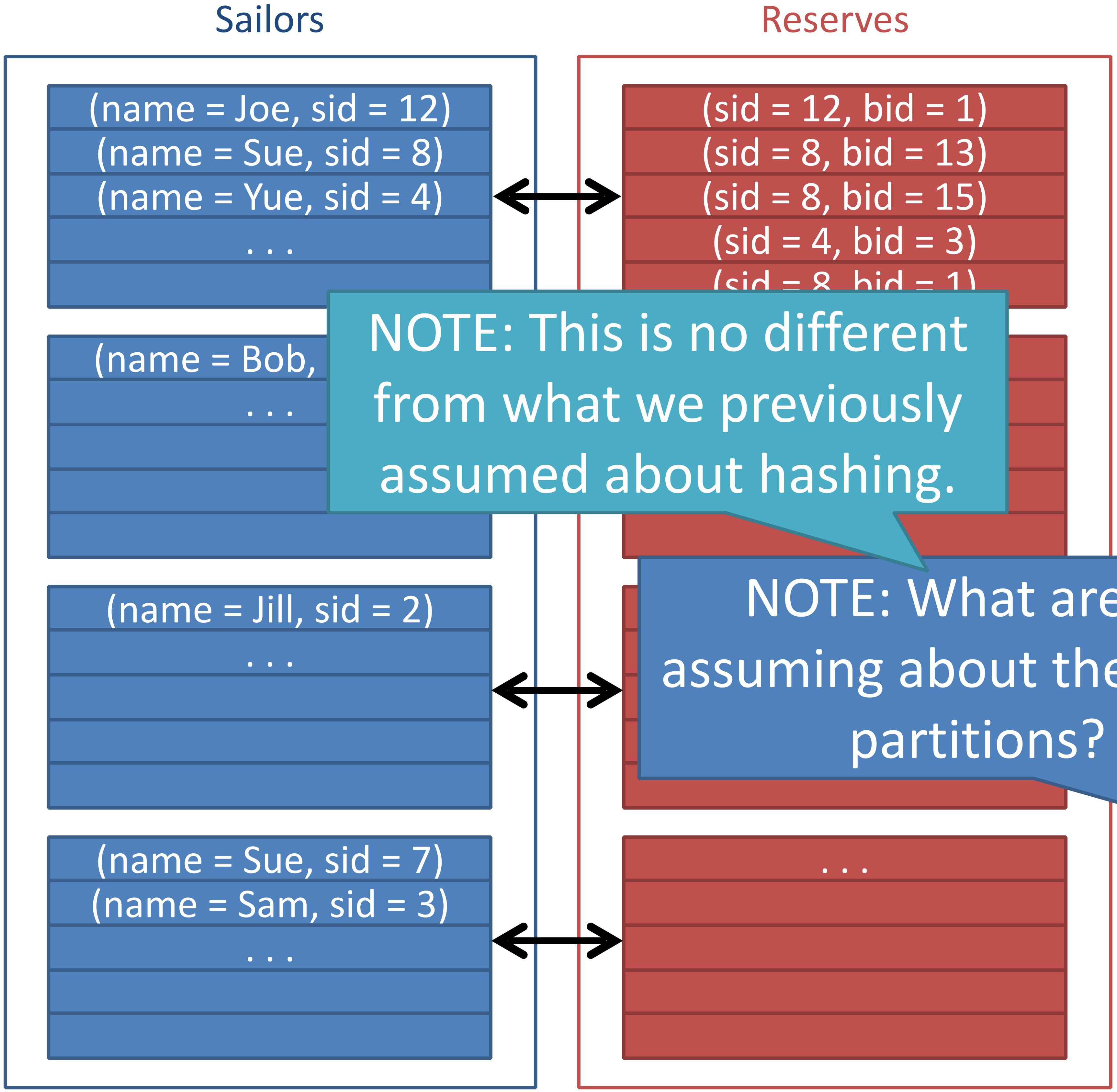
Partition S and R using same hash fn, then collect same partitions

Steps:

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

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

Hash-Join



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]

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
- Small memory size
- Want sorted output/have sorted input

When is a hash-join the best?

When is a hash-join the best?

- One partition large, the other small (can keep in memory)

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.

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.

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.

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.

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

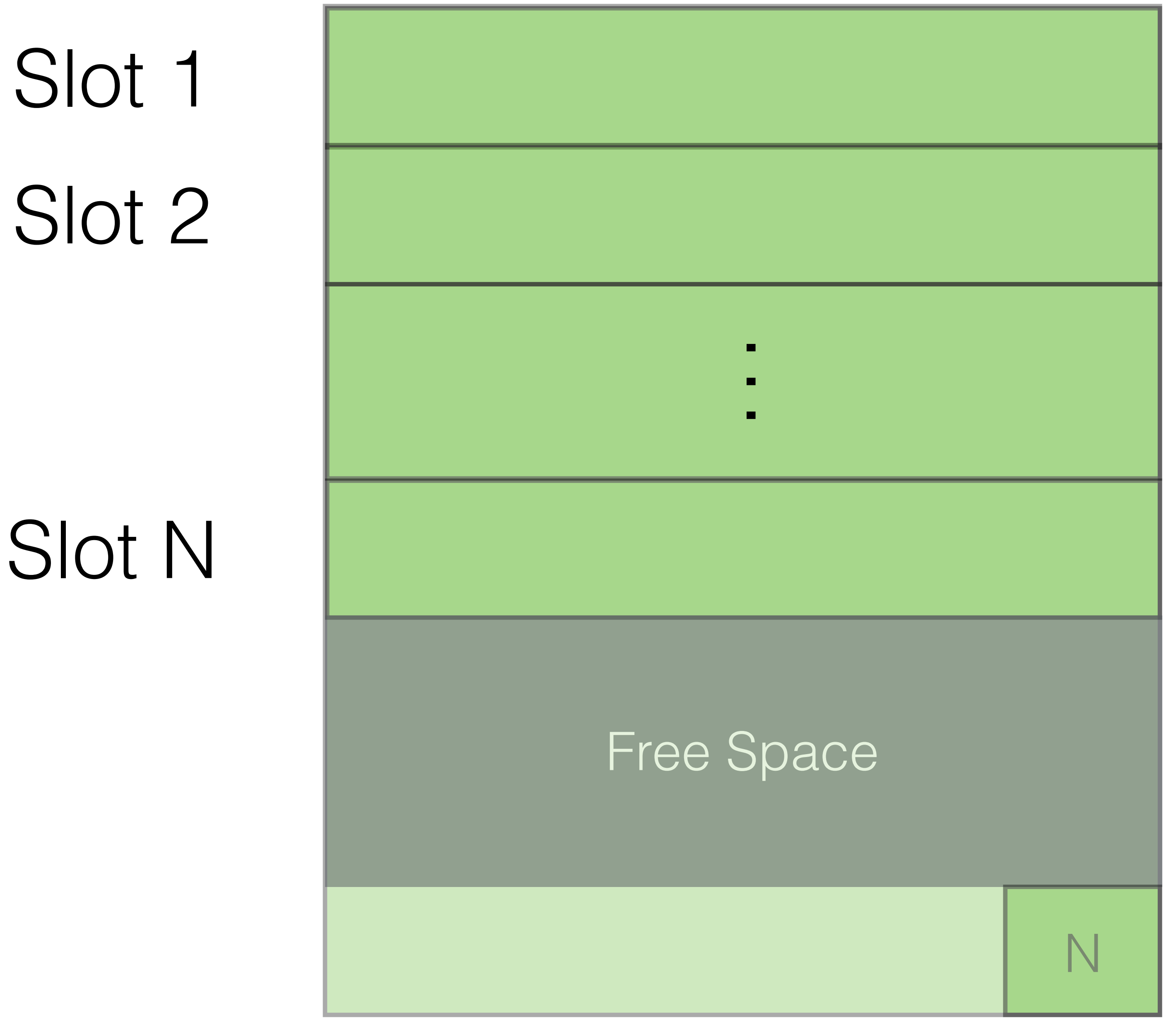
(# of pages in both relations) * (1 read before hashing + 1 read after hashing)

=(100+50) * 2
= 300

Heap Files

(Page Formats)

Fixed-Length Records

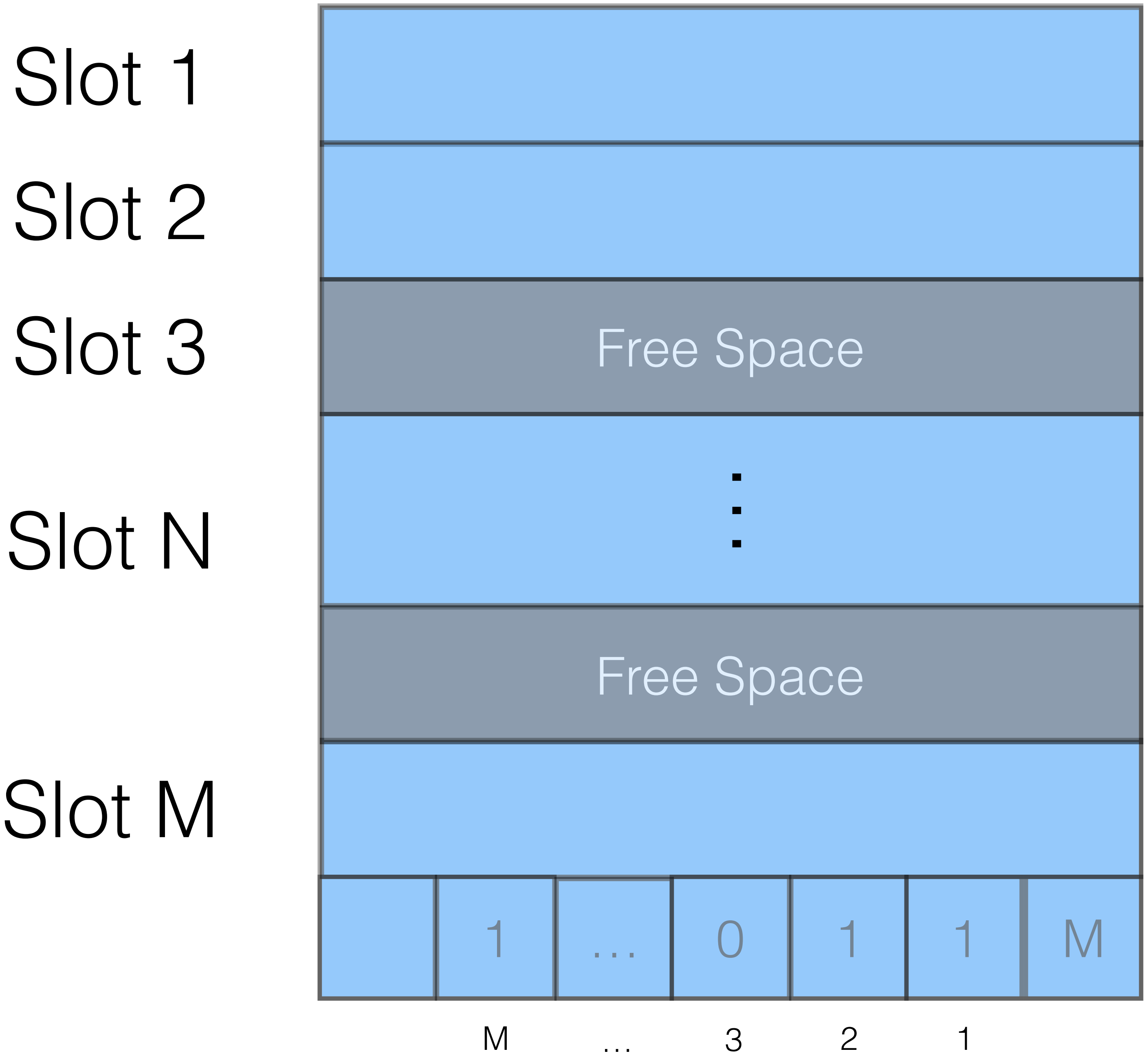


Record id:
<Page id, slot #>

Number of Records

Packed

Fixed-Length Records

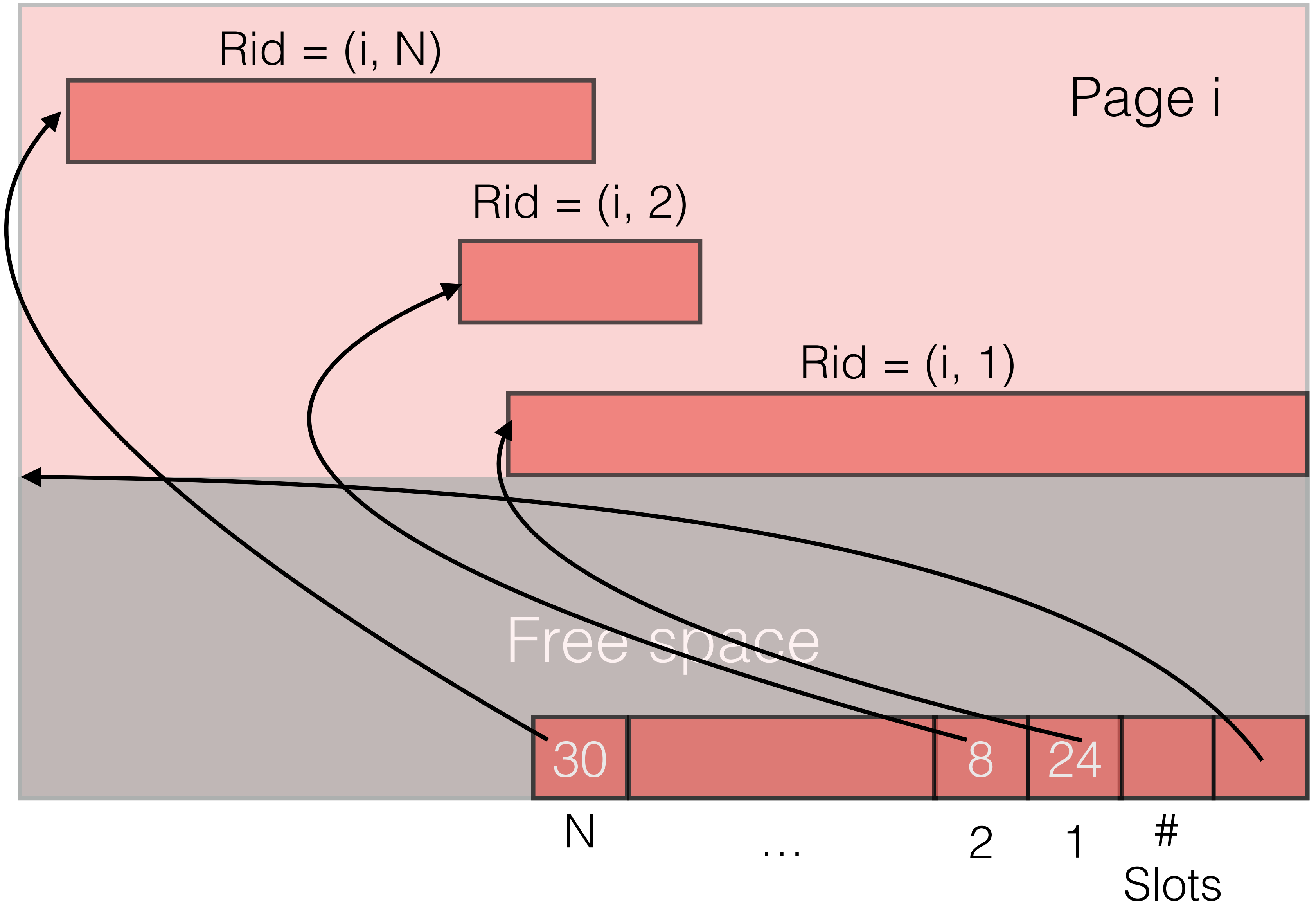


Record id:
<Page id, slot #>

Number of Slots

Unpacked (Bitmap)

Variable Length Records



Pointer to start of free space

Record id:
<Page id, slot #>

Slotted Page

What are the advantages and disadvantages of using slotted pages or bitmaps over just tightly packing records together?

What are the advantages and disadvantages of using slotted pages or bitmaps over just tightly packing records together?

- Allow movement of records without changing record ID
- Slotted pages support variable-length records

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

What’s the size of the largest record you can insert?

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

What’s the size of the largest record you can insert?

Need 4 bytes for the entry, so (80 - 4) = 76 bytes

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

At most, how many 1-byte large records can you insert?

You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

At most, how many 1-byte large records can you insert?

- Amount of space taken up by x 1-byte records
= (1 byte for record + 4 for directory entry)
= (5 bytes / record)

- Free space / (amount of space per record)
= 80 / 5 = 16 records