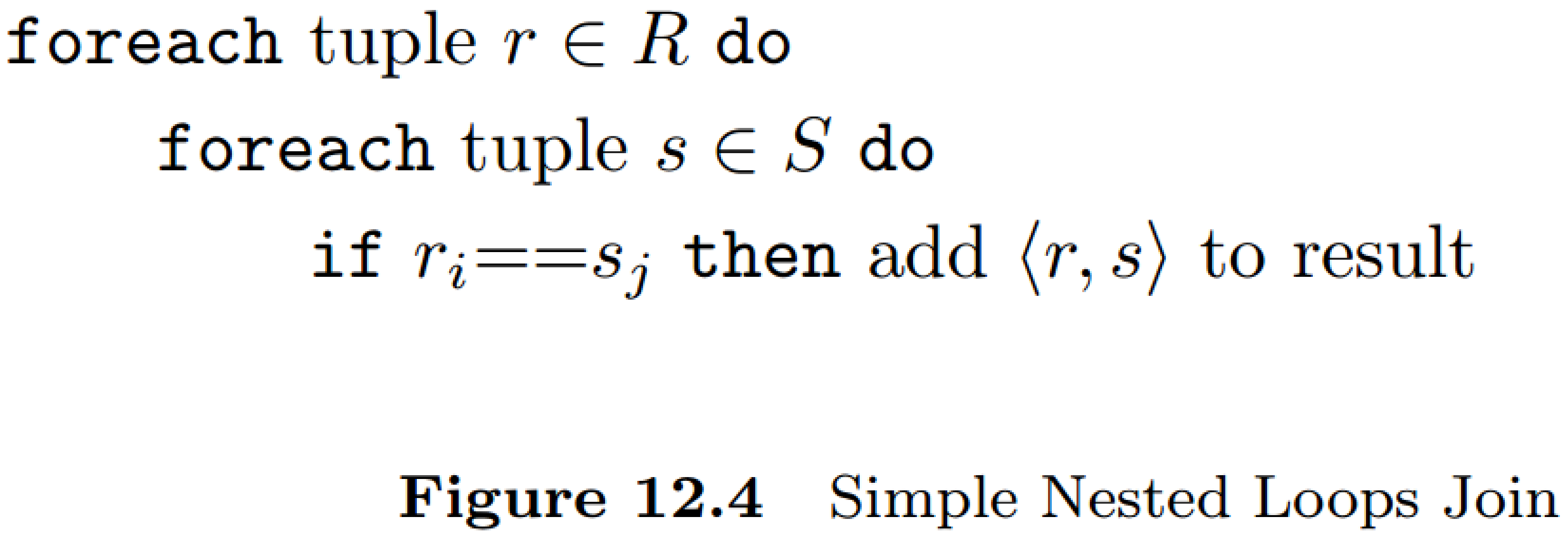
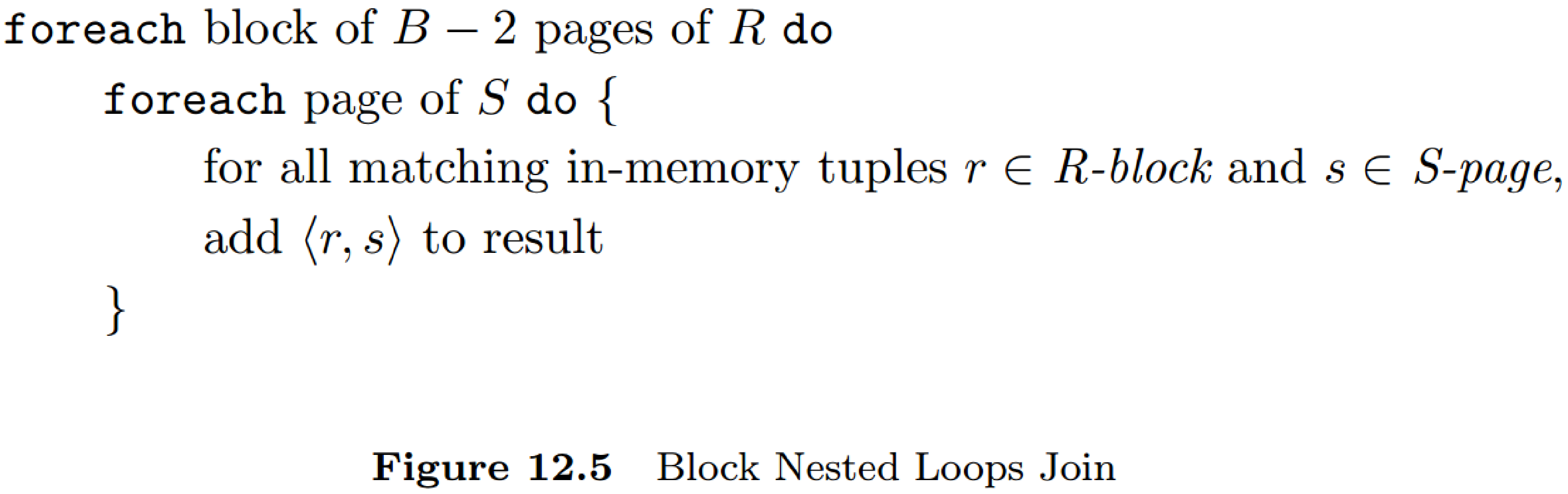
Simple Nested Loops Join



```
foreach tuple $r \in R$ do
    foreach tuple $s \in S$ do
        if $r_i == s_j$ then add $\langle r, s \rangle$ to result
```

Figure 12.4 Simple Nested Loops Join

Block Nested Loops Join



```
foreach block of $B - 2$ pages of $R$ do
    foreach page of $S$ do {
        for all matching in-memory tuples $r \in R\text{-}block$ and $s \in S\text{-}page$,
        add $\langle r, s \rangle$ to result
    }
```

Figure 12.5 Block Nested Loops Join

Buffer Usage in Block Nested Loops Join

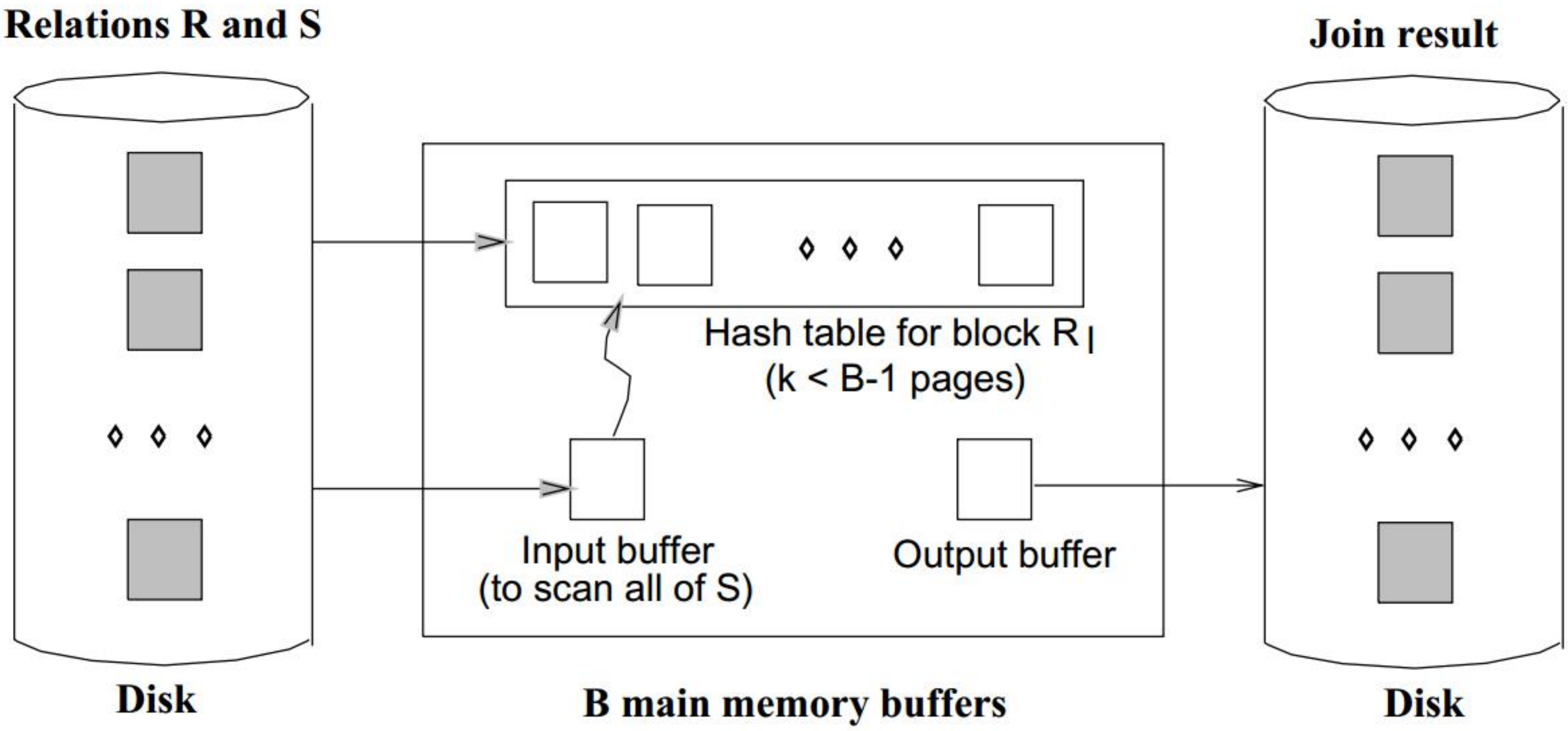
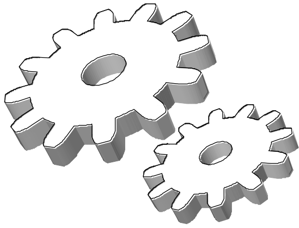
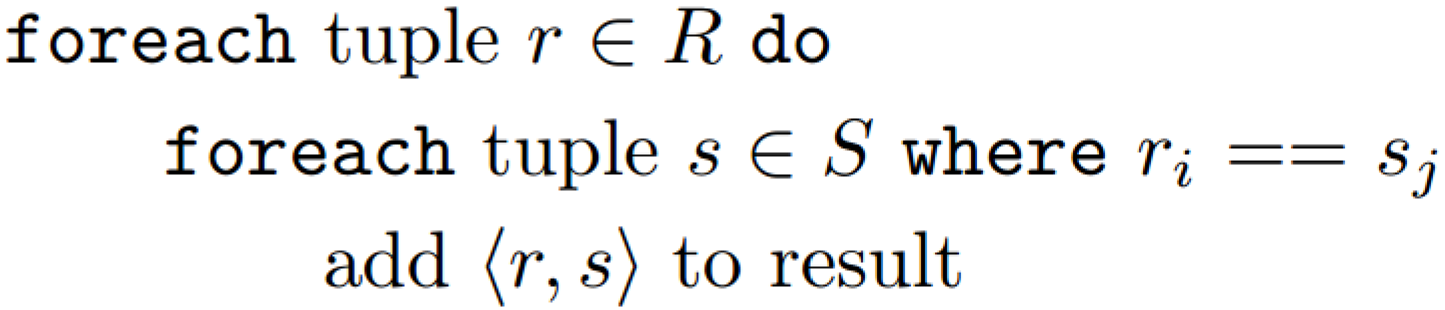


Figure 12.6 Buffer Usage in Block Nested Loops Join



Index Nested Loops Join

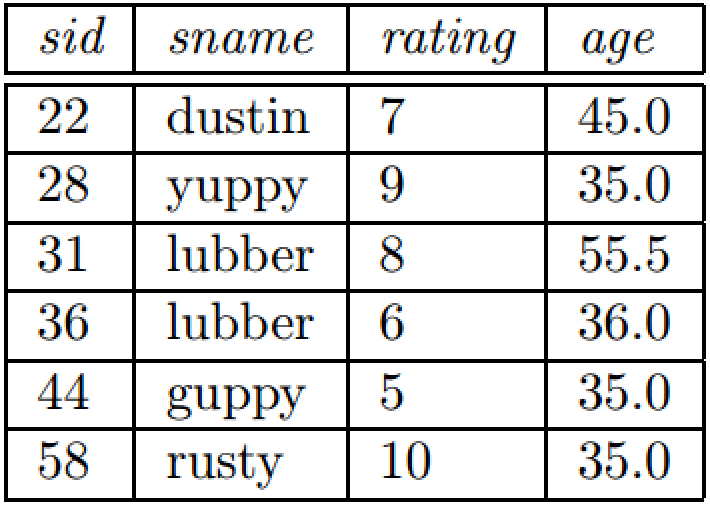


```
foreach tuple $r \in R$ do
    foreach tuple $s \in S$ where $r_i == s_j$
        add $\langle r, s \rangle$ to result
```

Figure 12.7 Index Nested Loops Join

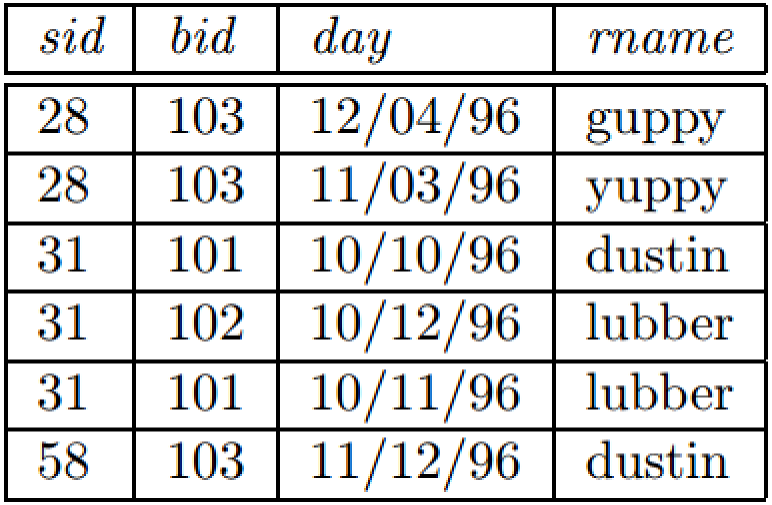
```
proc $smjoin(R, S, `R_i = S'_j)$

if $R$ not sorted on attribute $i$, sort it;
if $S$ not sorted on attribute $j$, sort it;

$Tr$ = first tuple in $R$;                              // ranges over $R$
$Ts$ = first tuple in $S$;                              // ranges over $S$
$Gs$ = first tuple in $S$;                              // start of current $S$-partition

while $Tr \neq eof$ and $Gs \neq eof$ do {

    while $Tr_i < Gs_j$ do
        $Tr$ = next tuple in $R$ after $Tr$;            // continue scan of $R$

    while $Tr_i > Gs_j$ do
        $Gs$ = next tuple in $S$ after $Gs$             // continue scan of $S$

    $Ts = Gs$;                                          // Needed in case $Tr_i \neq Gs_j$
    while $Tr_i == Gs_j$ do {                           // process current $R$ partition
        $Ts = Gs$;                                      // reset $S$ partition scan
        while $Ts_j == Tr_i$ do {                       // process current $R$ tuple
            add $\langle Tr, Ts \rangle$ to result;     // output joined tuples
            $Ts$ = next tuple in $S$ after $Ts$;}       // advance $S$ partition scan
        $Tr$ = next tuple in $R$ after $Tr$;            // advance scan of $R$
        }                                               // done with current $R$ partition

    $Gs = Ts$;                                          // initialize search for next $S$ partition
    }
```

Figure 12.8 Sort-Merge Join



sid	*sname*	*rating*	*age*
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
36	lubber	6	36.0
44	guppy	5	35.0
58	rusty	10	35.0

Figure 12.9 An Instance of Sailors



sid	*bid*	*day*	*rname*
28	103	12/04/96	guppy
28	103	11/03/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Figure 12.10 An Instance of Reserves

Probing Phase of Hash Join

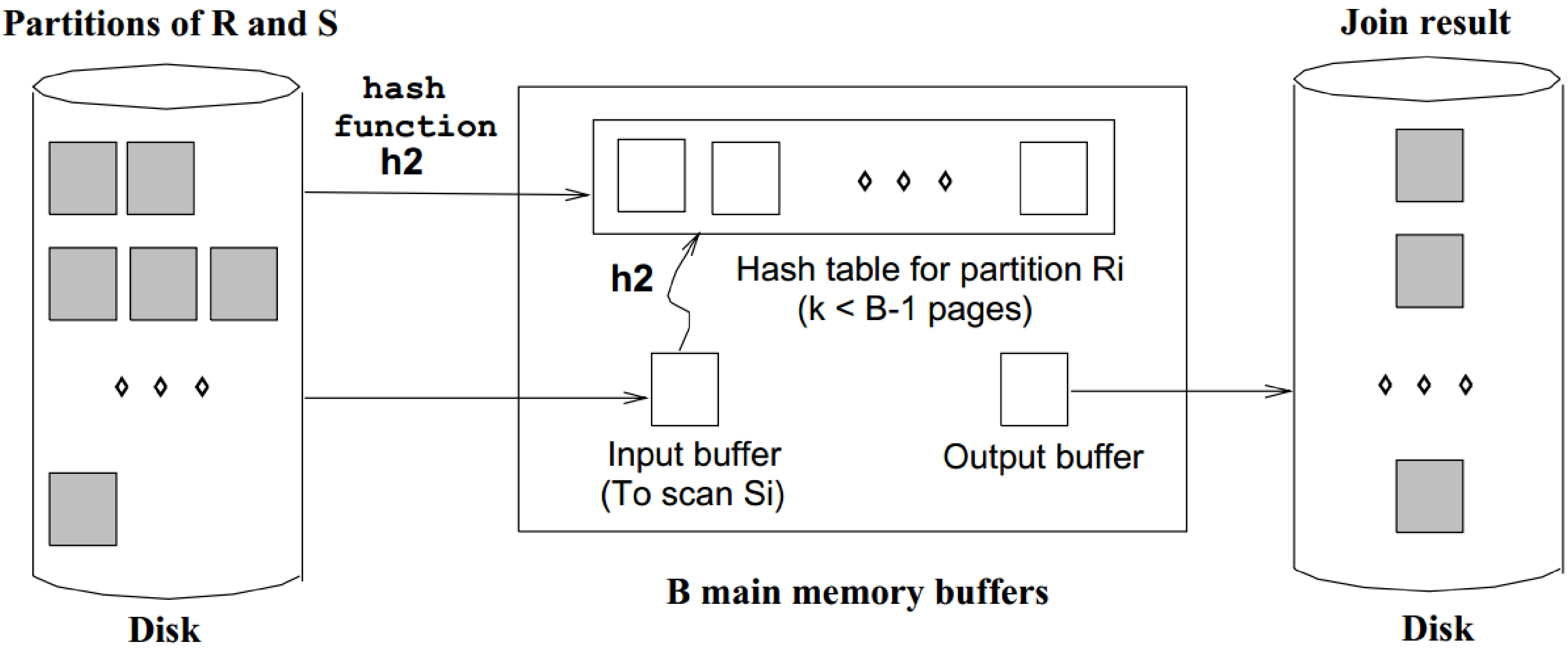
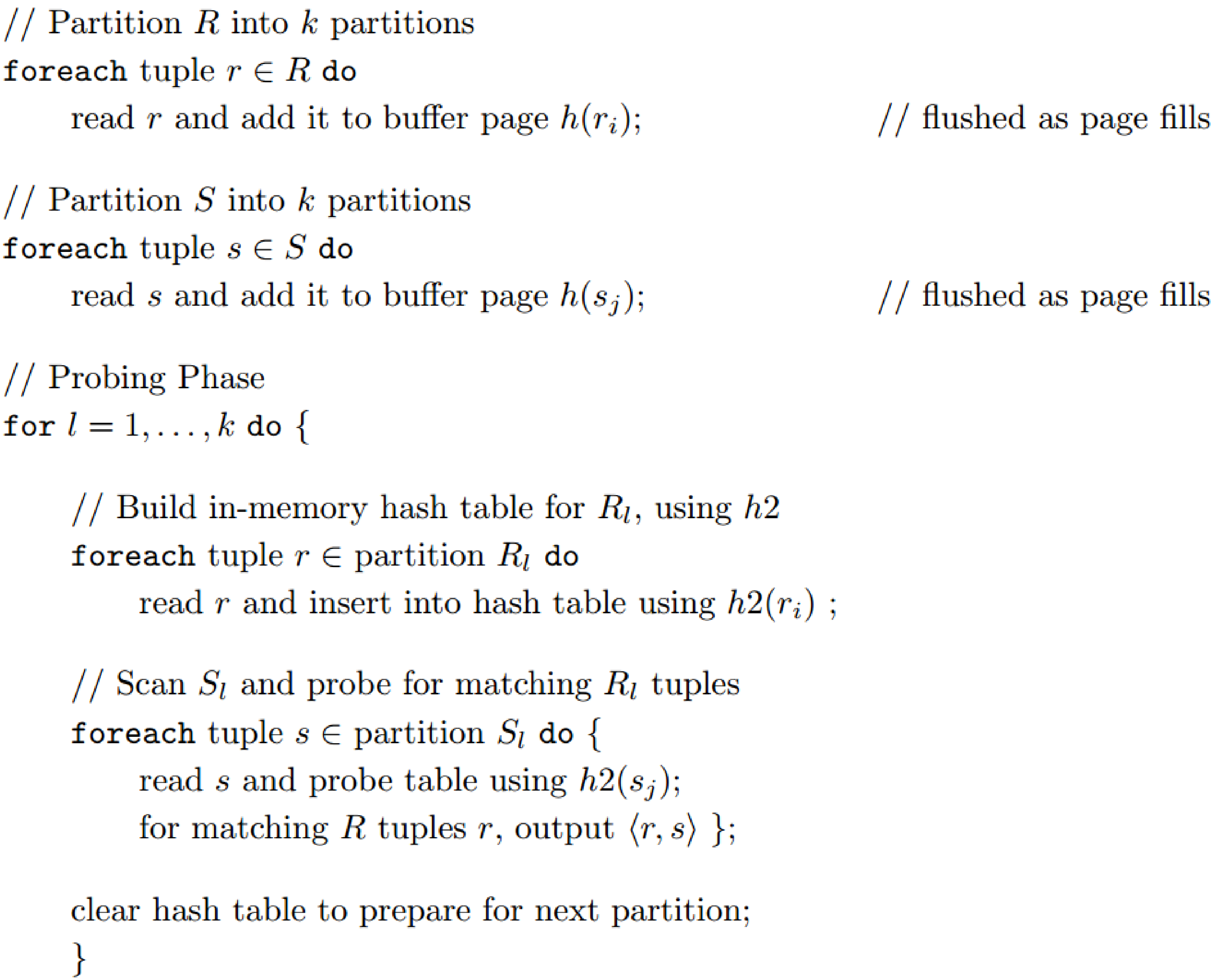


Figure 12.11 Probing Phase of Hash Join



```
// Partition $R$ into $k$ partitions
foreach tuple $r \in R$ do
    read $r$ and add it to buffer page $h(r_i)$;            // flushed as page fills

// Partition $S$ into $k$ partitions
foreach tuple $s \in S$ do
    read $s$ and add it to buffer page $h(s_j)$;            // flushed as page fills

// Probing Phase
for $l = 1, \ldots, k$ do {

    // Build in-memory hash table for $R_l$, using $h2$
    foreach tuple $r \in$ partition $R_l$ do
        read $r$ and insert into hash table using $h2(r_i)$ ;

    // Scan $S_l$ and probe for matching $R_l$ tuples
    foreach tuple $s \in$ partition $S_l$ do {
        read $s$ and probe table using $h2(s_j)$;
        for matching $R$ tuples $r$, output $\langle r, s \rangle$ };

    clear hash table to prepare for next partition;
    }
```

Figure 12.12 Hash Join