Equality Joins

```
SELECT  *
FROM    Users U, GroupMembers GM
WHERE   U.uid = GM.uid
```

	uid	uname	experience	age
$p1_u$	123	Dora	2	13
	132	Bug	8	60
$p2_u$	267	Sakura	7	15
	111	Cyphon	8	35

	uid	gid	stars
$p1_g$	123	G1	2
	132	G1	5
$p2_g$	132	G2	3
	132	G3	1
$p3_g$	123	G2	4
	111	G4	2

Join Cardinality Estimation

- | Users $\bowtie$ GroupMembers | = ? cardinality of a join
  - Join attribute is primary key for Users, foreign key in GroupMember
  - Each GroupMember tuple matches exactly with one Users tuple
  - Result: |GroupMembers|
- | Users X GroupMembers | = ? cross product
  - Result: |Users| * |GroupMembers|
  - Cross product is always the product of individual relation sizes
- For other joins more difficult to estimate

Cardinality Estimation

- | Users $\bowtie \sigma_{(stars > 3)}$(GroupMembers) | = ?
  - Result: | $\sigma_{(stars > 3)}$(GroupMembers) |
  - Assuming 1-5 stars, uniform distribution for stars $\frac{2}{5}$
  - Red($\sigma_{(stars > 3)}$(GroupMembers)) = $\boxed{0.4}$ reduction factor
  - Result: $\boxed{0.4}$ * |GroupMembers|
- | $\sigma_{(experience > 5)}$ (Users) $\bowtie$ (GroupMembers) | = ?
  - Assume 1-10 experience levels, uniform distribution for experience
  - Red($\sigma_{(experience > 5)}$(Users)) = $\boxed{1/2}$ reduction factor 5/10
  - Result: $\boxed{½}$ * |GroupMembers|

Simple Nested Loop Join

- For each tuple in the *outer* relation Users U we scan the entire *inner* relation GroupMembers GM.

```
foreach tuple u in U do
 foreach tuple g in GM do
    if u.uid == g.uid  then add <u, g> to result
```

$p1_u$ $p2_u$

uid	uname	experience	age
123	Dora	2	13
132	Bug	8	60
267	Sakura	7	15
111	Cyphon	8	35

$p1_g$ $p2_g$ $p3_g$

uid	gid	stars
123	G1	2
132	G1	5
132	G2	3
132	G3	1
123	G2	4
111	G4	2

Simple Nested Loop Join

- For each tuple in the *outer* relation Users U we scan the entire *inner* relation GroupMembers GM.

```
foreach tuple u in U do
 foreach tuple g in GM do
    if u.uid == g.uid  then add <u, g> to result
```

- Cost: UserPages + |Users| * GroupMemberPages = 500 + 40,000*1000 !

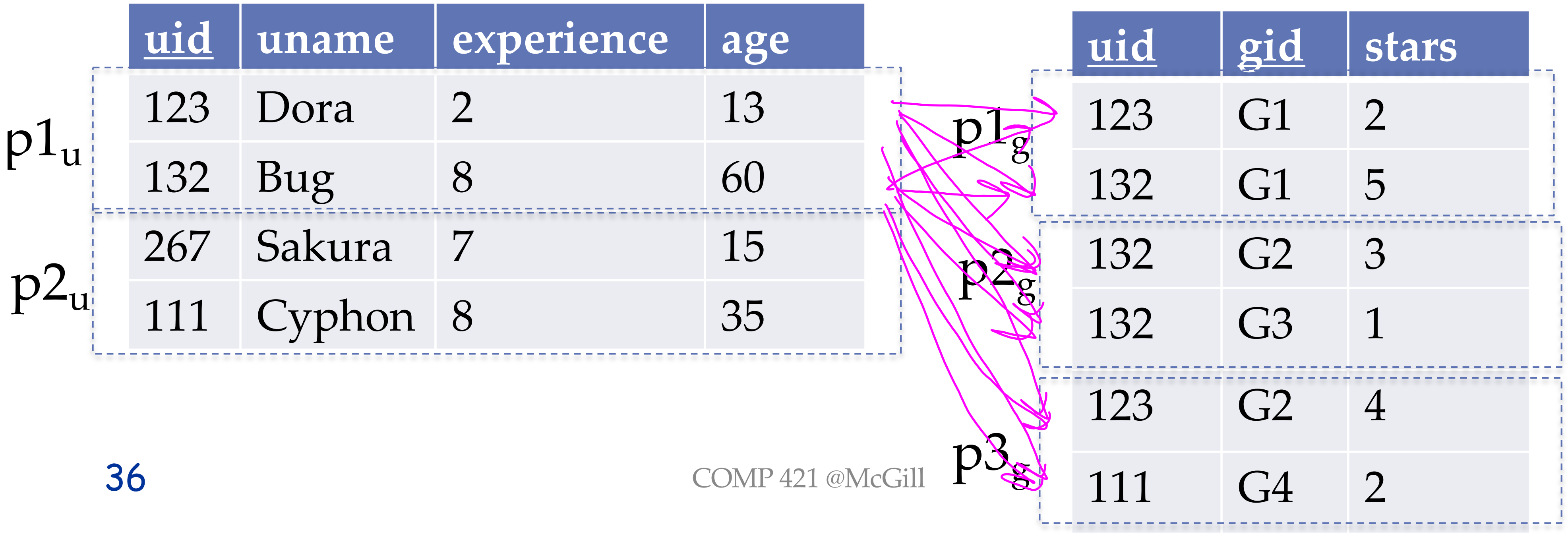
- NOT GOOD
- We need page-oriented algorithm!

tuple oriented algorithm

Page Nested Loop Join

- For each *page* p_u of Users U, get each *page* p_g of GroupMembers GM
  - write out matching pairs <u, g>, where u is in p_u and g is in p_g.

```
For each page $p_u$ of Users U
  for each page $p_g$ of GroupMembers GM
    for each tuple u in $p_u$ do
      for each tuple g in $p_g$ do
        if u.uid == g.uid  then add <u, g> to result
```



	uid	uname	experience	age
$p1_u$	123	Dora	2	13
	132	Bug	8	60
$p2_u$	267	Sakura	7	15
	111	Cyphon	8	35

	uid	gid	stars
$p1_g$	123	G1	2
	132	G1	5
$p2_g$	132	G2	3
	132	G3	1
$p3_g$	123	G2	4
	111	G4	2

Page Nested Loop Join

- For each *page* p_u of Users U, get each *page* p_g of GroupMembers GM
  - write out matching pairs <u, g>, where u is in p_u and g is in p_g.

```
For each page $p_u$ of Users U
  for each page $p_g$ of GroupMembers GM
    for each tuple u in $p_u$ do
       for each tuple g in $p_g$ do
         if u.uid == g.uid  then add <u, g> to result
```

- Cost: UserPages + UserPages*GroupPages = 500 + 500*1000 = 500,500

Still not great

Block Nested Loop Join

- For each *block of pages* bp_u of Users U, get each *page* p_g of GroupMembers GM
  - write out matching pairs <u, g>, where u is in bp_u and g is in p_g.
- *block of pages* bp_u and one page of GM must fit in main memory
  - For each block of pages bp_u
    - 1 Load block into main memory
    - 2 Get first page from GM
      - Do all the matching between users in bp_u and group members in first page
    - 3 Get second page from GM (into the same frame the first one was in before)
      - Do all the the matching between users in bp_u and group members in second page
    - …
    - n Get last page from GM (into again that frame reserved for GM)
      - ...
- Cost: UserPages + UserPages / $|bp_u|$ * GroupMemberPages

Block Nested Loop

	uid	uname	experience	age
$p1_u$	123	Dora	2	13
	132	Bug	8	60
$p2_u$	267	Sakura	7	15
	111	Cyphon	8	35

	uid	gid	stars
$p1_g$	123	G1	2
	132	G1	5
$p2_g$	132	G2	3
	132	G3	1
$p3_g$	123	G2	4
	111	G4	2

Block Nested Loop Join

- Examples depending on available main memory:
- 51 Buffer Frames:
  - user pages 10 blocks gln relation
  - 500 + 500/50 * 1000 = 500 + 10,000
- 501 Buffer Frames
  - 500 + 500/500 * 1000 = 500 + 1000
  - Special case: outer relation fits into main memory!!

Index Nested Loops Join

- For each tuple in the *outer* relation Users U we find the matching tuples in GroupMembers GM through an index
  - Condition: GM must have an index on the join attribute

```
foreach tuple u in U do
 find all matching tuples g in GM through index
    then add all <u, g> to result
```

	uid	uname	experience	age
$p1_u$	123	Dora	2	13
	132	Bug	8	60
$p2_u$	267	Sakura	7	15
	111	Cyphon	8	35

	uid	gid	stars
$p1_g$	123	G1	2
	132	G1	5
$p2_g$	132	G2	3
	132	G3	1
$p3_g$	123	G2	4
	111	G4	2

Use index to find matching tuples

Index Nested Loops Join

```
foreach tuple u in U do
 find all matching tuples g in GM through index
   then add all <u, g> to result
```

- Index MUST be on the inner relation (in this case GM).
- Cost: OuterPages + CARD(OuterRelation) * cost of finding matching tuples in inner relation
- In example of previous page:
  - Index on uid on GM is clustered:
    - 500 + 40.000 * (1 leaf page +1 data pages)
  - Index on uid on GM is not clustered:
    - 500 + 40.000 * (1 leaf page + 2.5 data pages) (on average 2.5 tuples in GM per user)

Index Nested Loops Join

- Switch inner and outer if index is on uid of Users
- Note: uid is primary key in User
  - Only one tuple matches!

```
foreach tuple g in GM do
 find the one matching tuple u in U through index
   then add <g, u> to result
```

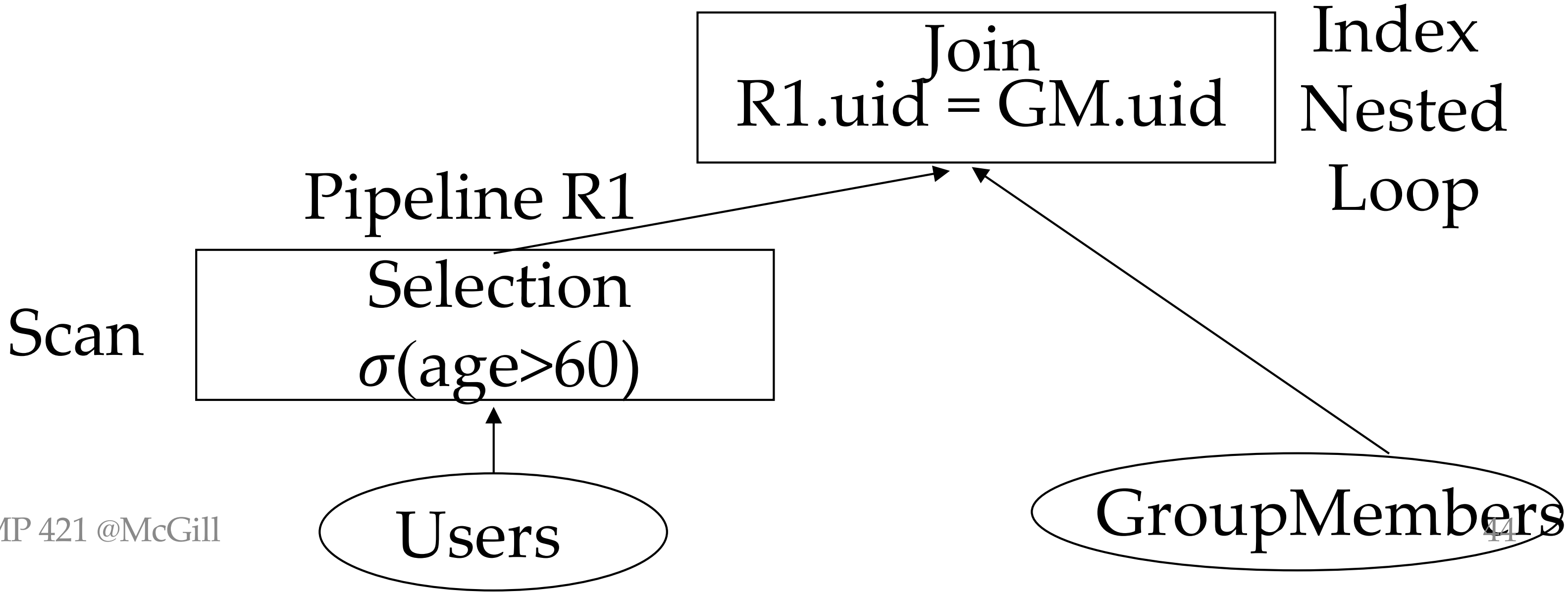
- Cost: 1000 + 100.000 * (1 leaf page + 1 data page)

	uid	gid	stars
$p1_g$	123	G1	2
	132	G1	5
$p2_g$	132	G2	3
	132	G3	1
$p3_g$	123	G2	4
	111	G4	2

	uid	uname	experience	age
$p1_u$	123	Dora	2	13
	132	Bug	8	60
$p2_u$	267	Sakura	7	15
	111	Cyphon	8	35

Block Nested Loop vs. Index

- Best Case for Block Nested Loop (if outer relation fits in main memory)
  - OuterPages + InnerPages
- Index Nested Loop:
  - OuterPages + Card(Outer) * matching tuples Inner
- Index Nested Loop wins if:
  - InnerPages > Card(Outer) * matching tuples Inner
  - E.g., if Outer is the result of a selection that only selected very few tuples
    - $\sigma_{(age > 60)}$ (Users) ⋈ (GroupMembers)



Sort-Merge Join

- Sort U and GM on the join column, then scan them to do a "merge" (on join col.), and output result tuples.
  - In loop:
    - Assume the scan cursors currently points to U tuple u and GM tuple g. Advance scan cursor of U until u.uid >= g.uid and then advance scan cursor of GM so that g.uid >= u.uid. Do this until u.uid = g.uid.
    - At this point, all U tuples with same value in uid (*current U group*) and all GM tuples with same value in uid (*current GM group*) *match*; output <u, g> for all pairs of such tuples.
  - Then resume scanning U and GM.
- U is scanned once; each GM group is scanned once per matching U tuple. (Multiple scans of an GM group are likely to find needed pages in buffer.)

Example of Sort-Merge Join

$p1_u$

$p2_u$

uid	uname	experience	age
111	Cyphon	8	35
123	Dora	2	13
132	Bug	8	60
267	Sakura	7	15

$p1_g$

$p2_g$

$p3_g$

uid	gid	stars
111	G4	2
123	G1	2
123	G2	4
132	G1	5
132	G2	3
132	G3	1

Cost of Sort-Merge Join

- Relations are already sorted: (on the disk)
  - UserPages + GroupMemberPages = 500 + 1000
- Relations need to be sorted – simple way:
  - Sort relations and write sorted relations to temporary stable storage
  - Read in sorted relations and merge
  - Costs: assuming 100 buffer pages
    - both Users and GroupMembers can be sorted in 2 passes (Pass 0 and 1): 4* UserPages + 4 GroupPages
    - Final merge: 500 + 1000 = (UserPages + GroupPages)
    - Total: 5 * UserPages + 5 GroupPages

Cost of Sort-Merge Join

- Relations need to be sorted – use pipelining to combine last sort pass and join
  - Sorting performs Pass 0 reads and writes each of the relations: 2 * UserPages + 2 GroupPages
  - Pass 1 reads data, sorts and then performs merge in pipeline fashion (ignore details): UserPages + GroupPages
  - Total: 3 * UserPages + 3 * GroupPages = 4,500