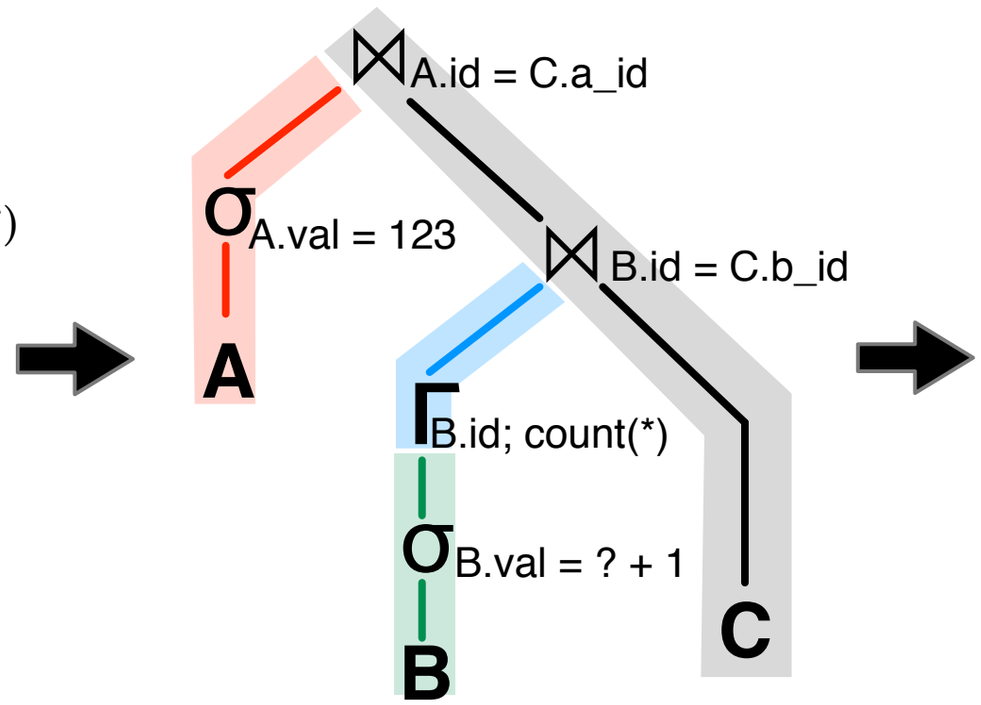
Query Compilation

Prashanth Menon

15-721 Database Systems (S16)

Query Comp 101

```
SELECT *
FROM A, C
   (SELECT B.id, COUNT(*)
    FROM B
    WHERE B.val = ?+1
    GROUP BY B.id)
AS B
WHERE A.val = 123
   AND A.id = C.a_id
   AND B.id = C.b_id
```




```
for t in A:
  if t.val == 123:
    Materialize t into hash-table
```

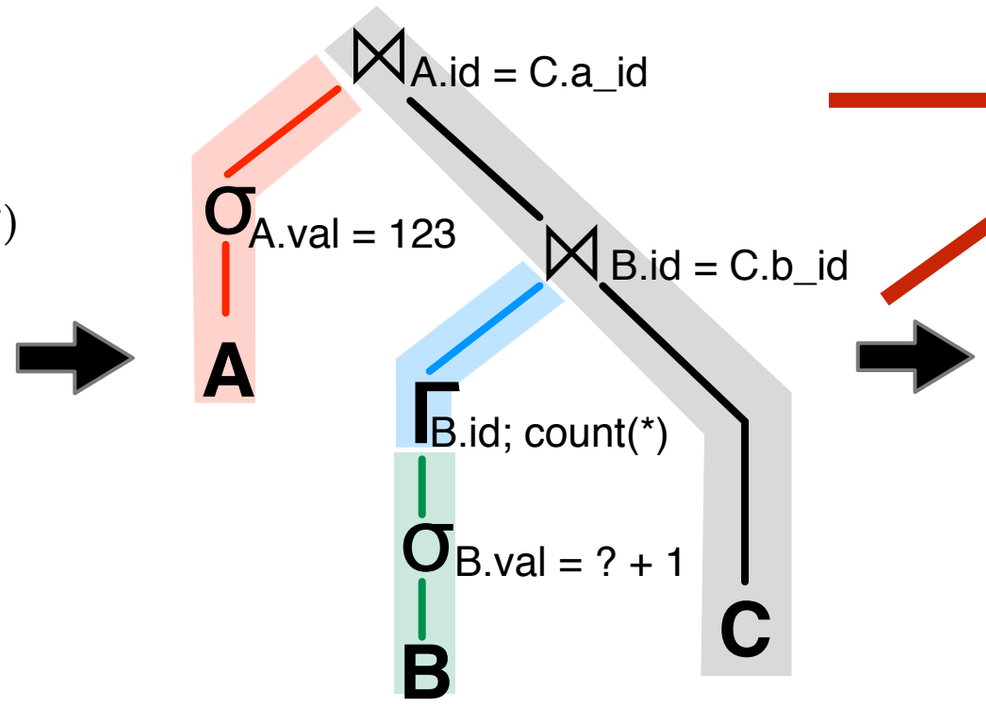
```
for t in B:
  if t.val == <param> + 1:
    Aggregate t in hash-table
```

```
for t in Γ(B.id):
  Materialize t into hash-table
```

```
for t3 in C:
  for t2 in ⋈(t2.id == t3.b_id):
    for t1 in ⋈(t1.id == t3.a_id):
      output(t1 ⋈ t2 ⋈ t3)
```

Query Comp 101

```
SELECT *
FROM A, C
   (SELECT B.id, COUNT(*)
    FROM B
    WHERE B.val = ?+1
    GROUP BY B.id)
AS B
WHERE A.val = 123
   AND A.id = C.a_id
   AND B.id = C.b_id
```



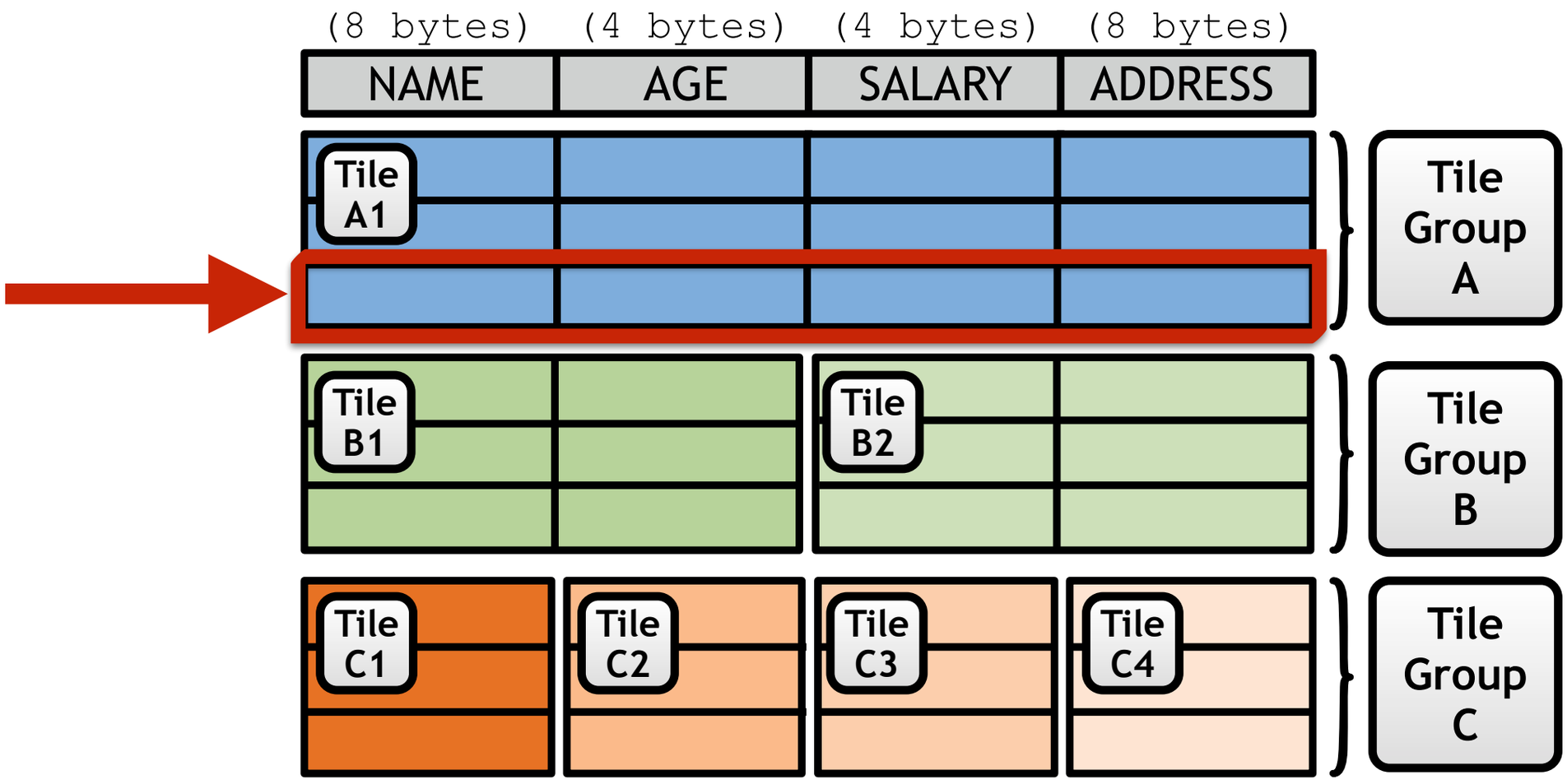

```
for t in A:
  if t.val == 123:
    Materialize t into hash-table
```

```
for t in B:
  if t.val == <param> + 1:
    Aggregate t in hash-table
```

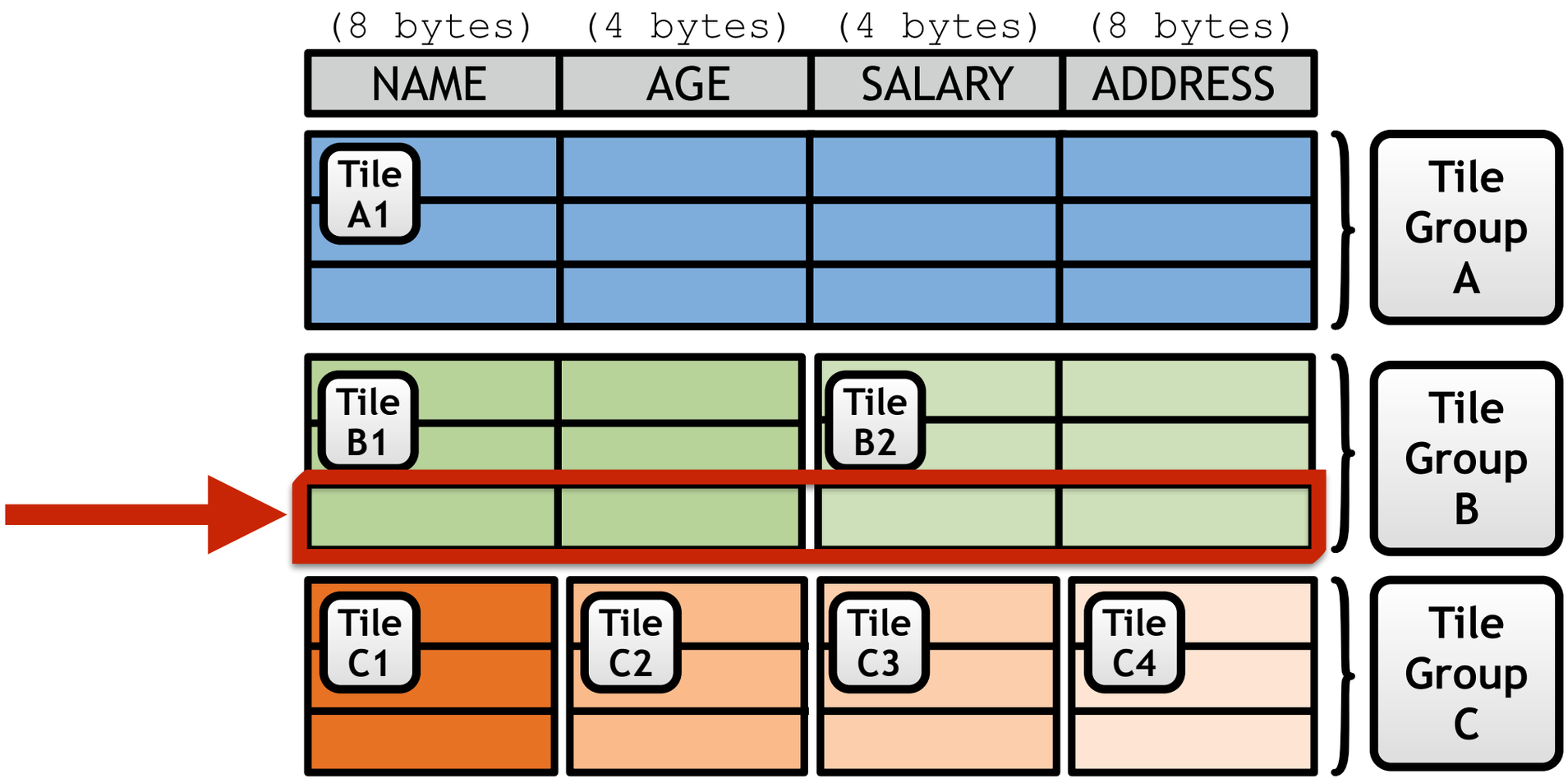
```
for t in Γ(B.id):
  Materialize t into hash-table
```

```
for t3 in C:
  for t2 in ⋈(t2.id == t3.b_id):
    for t1 in ⋈(t1.id == t3.a_id):
      output(t1 ⋈ t2 ⋈ t3)
```

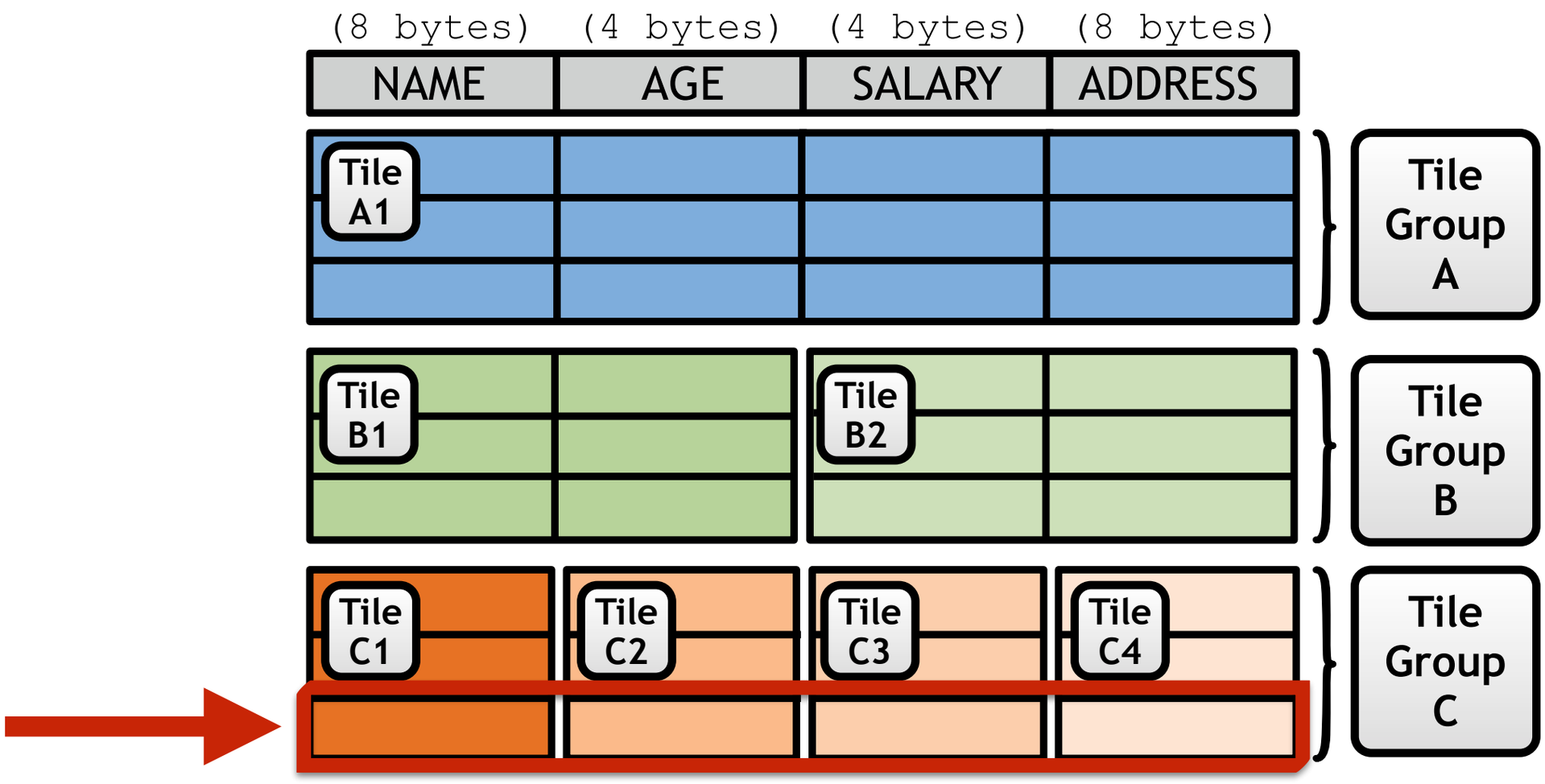
Hybrid Layout



Hybrid Layout



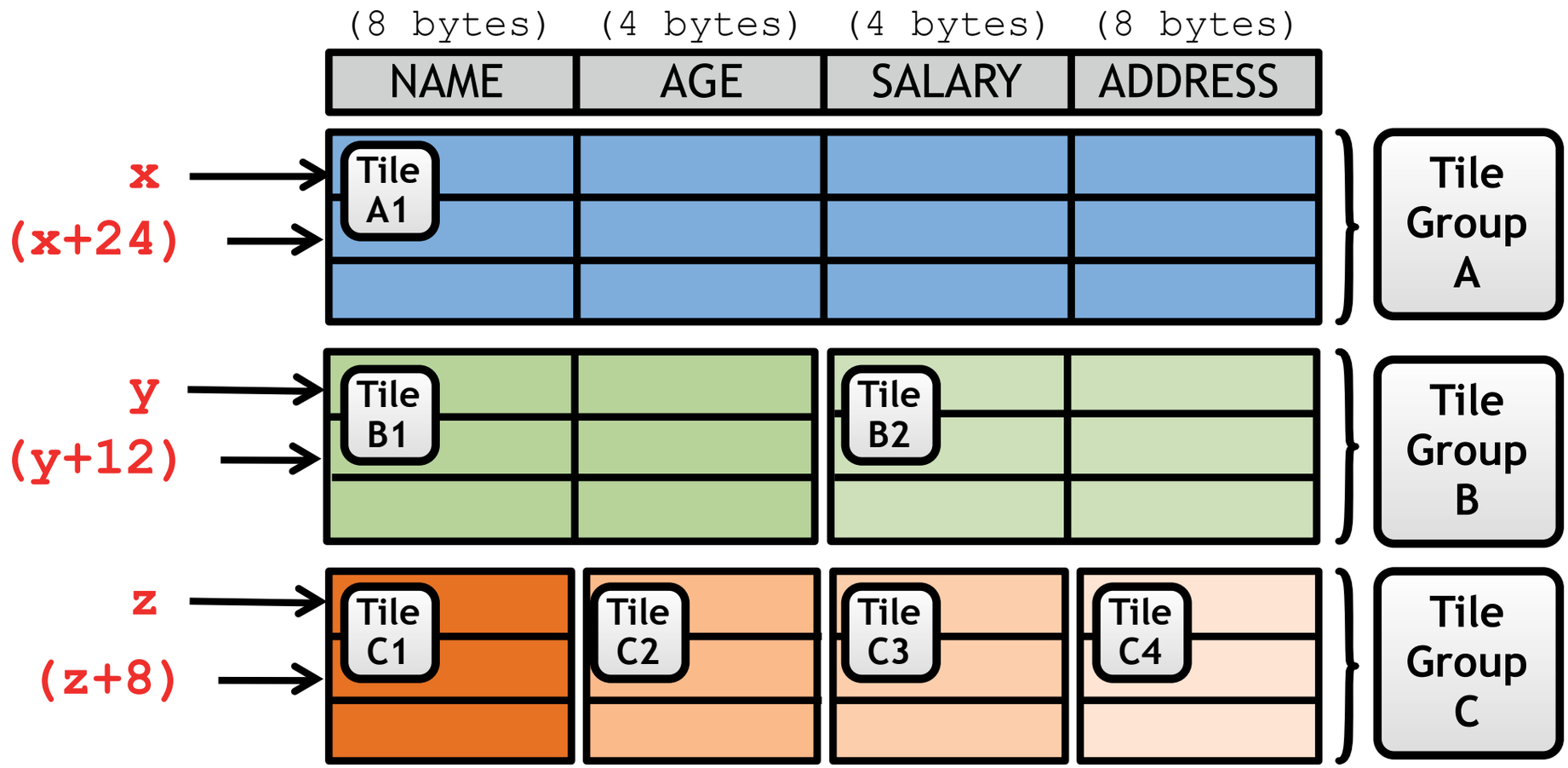
Hybrid Layout



Codegen for Hybrid

- View entire table as logically columnar
  - Name[tid], Age[tid], Salary[tid], Address[tid]
- Define every attribute with an **offset** and **stride**
  - Strided accesses over memory to mimic columnar storage
- Produce **layout-agnostic** code

Strided Access



name[rid] = **name.offset** + (rid * **name.stride**)

Vary per tile-group

Vary per tile-group

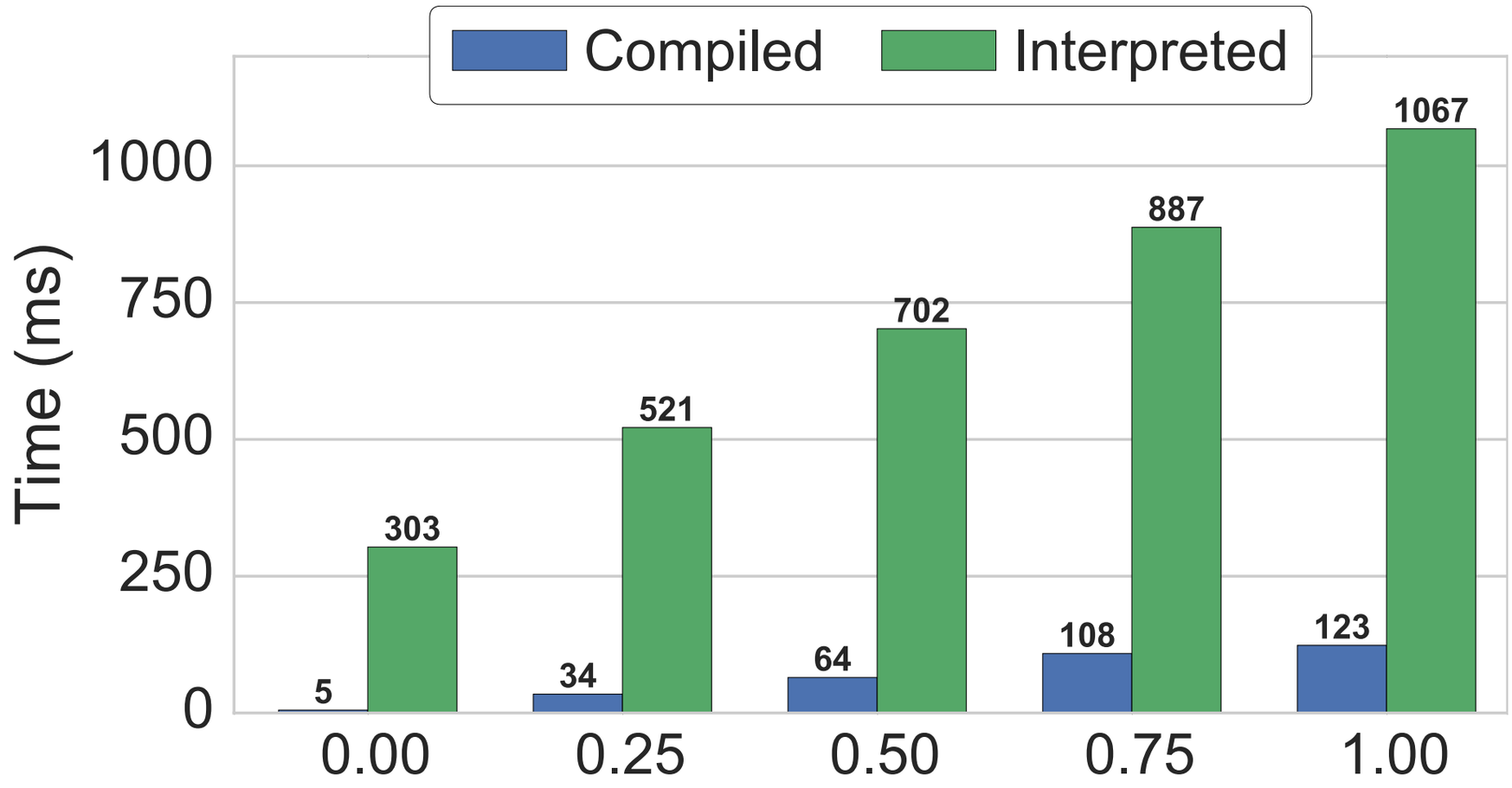
Sample

```
struct Layout { char * offset, int stride }
Layout name, age, salary, address

for layoutId in A.getLayouts():
  A.getLayoutDetail(layoutID, &name, &age
                    &salary, &address)
 for rid in layout.size():
   t_name = name.offset + (rid * name.stride)
   t_age  = age.offset + (rid * age.stride)
 . . .
```

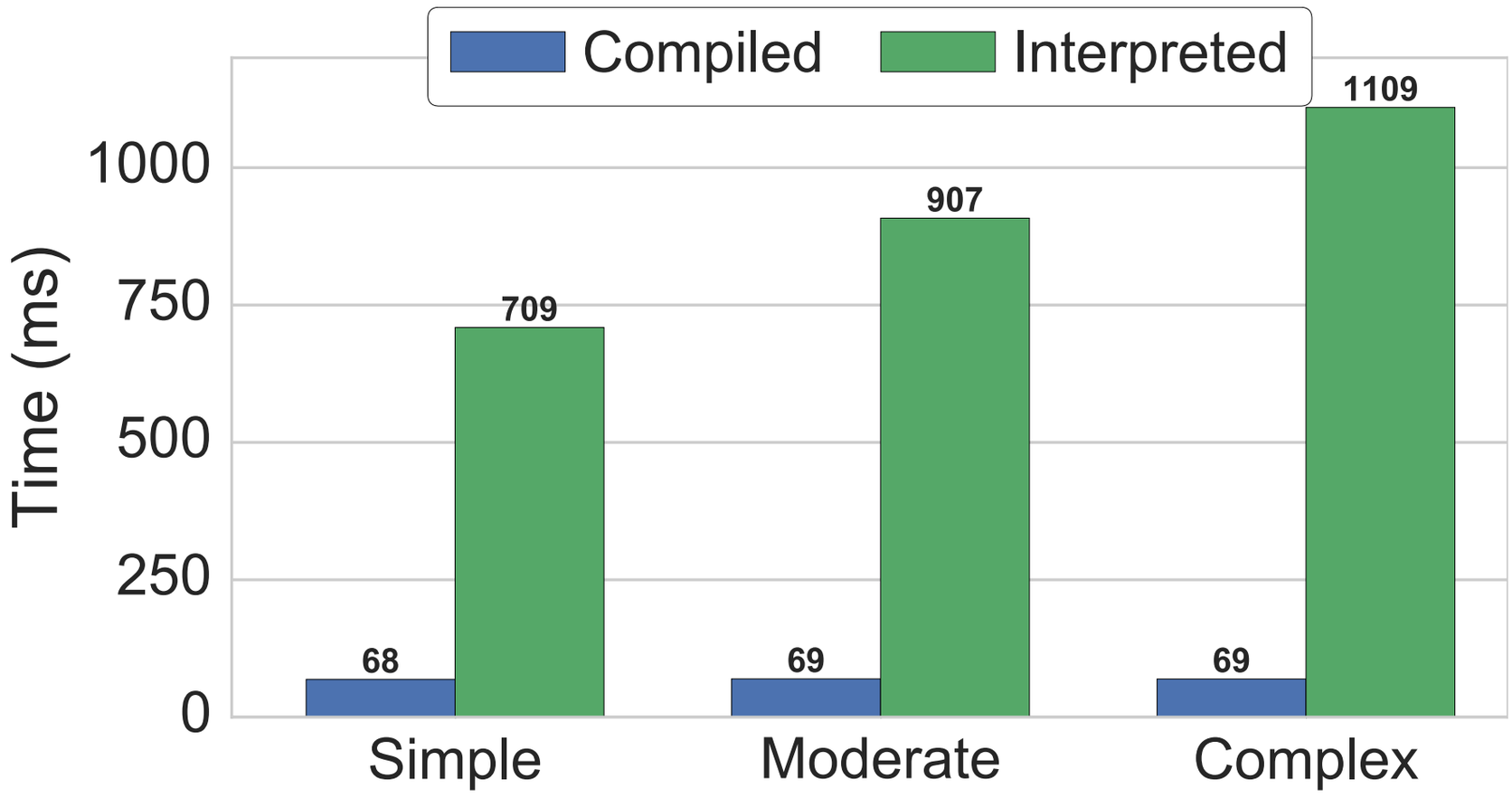
Scans

- Table scan selecting all columns
- Single predicate with **varying selectivity**



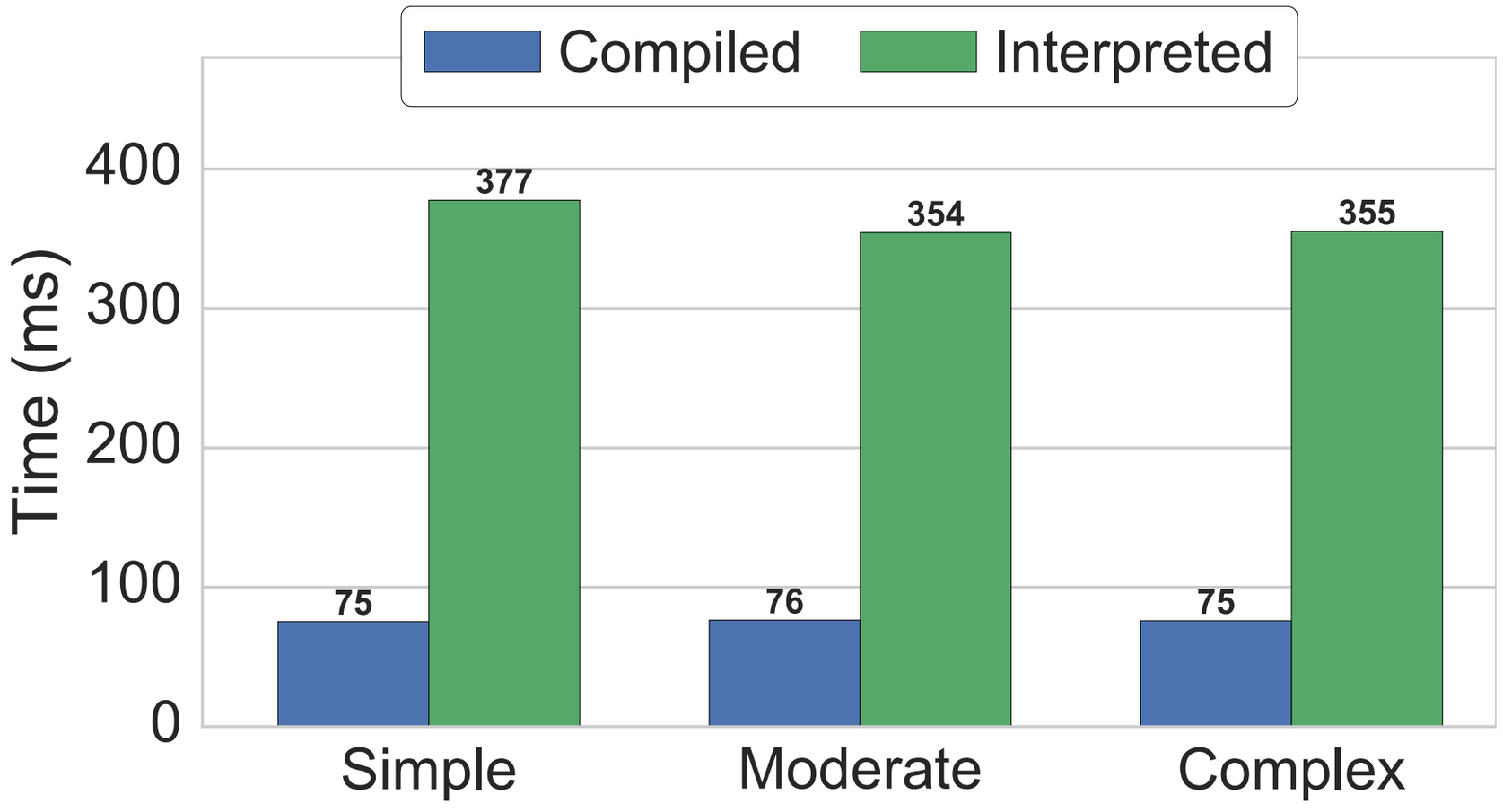
Scans

- Table scan selecting all columns
- Vary # predicates from one to three, keep selectivity constant



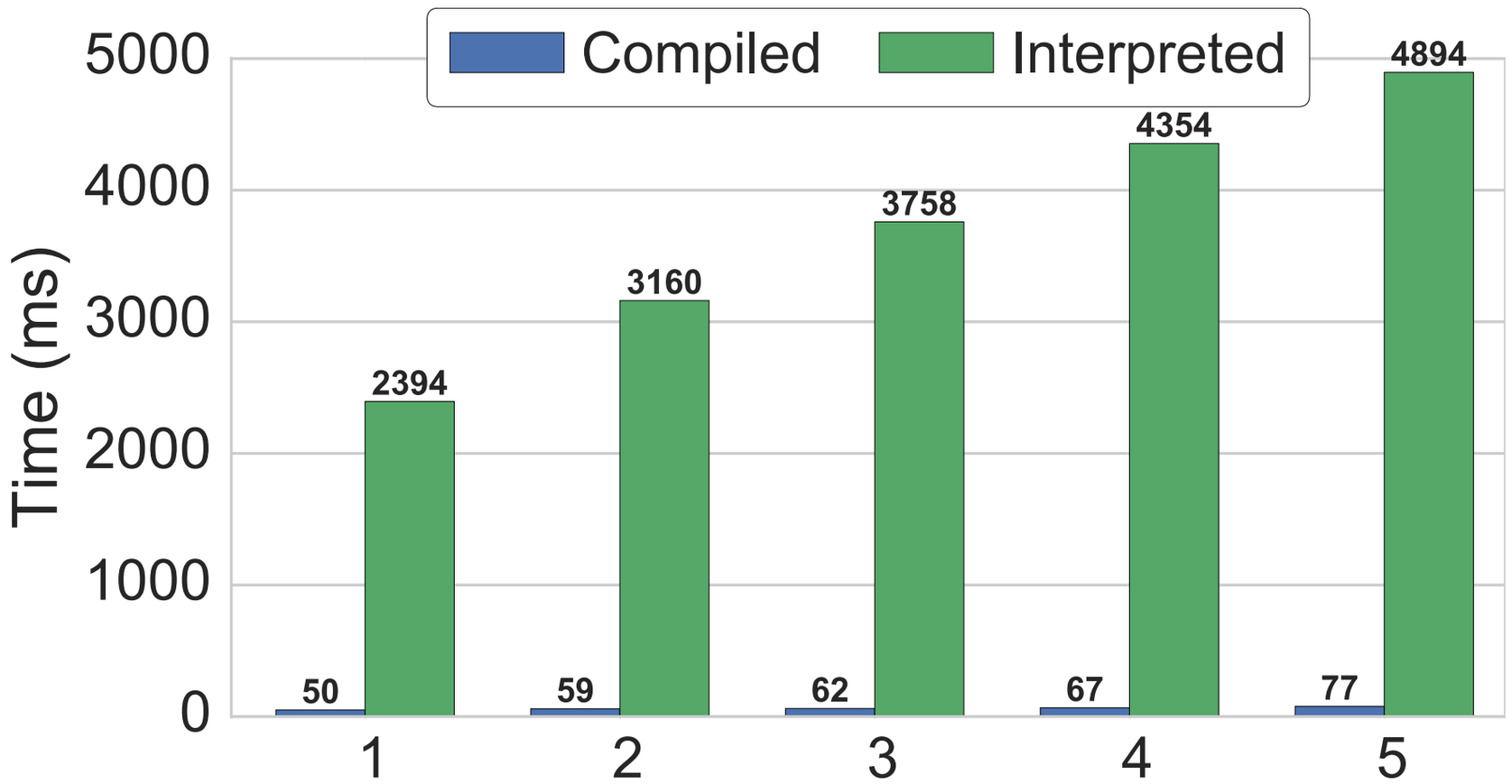
Joins

- Two 8-column tables, A, B
- Vary # join predicates from one to three, constant selectivity



Aggregates

- Vary # aggregates from one to five
- Aggregates are SUM over different column



Where benefits come from

- Inlining operator logic
  - No need to materialize tuples
  - No interpretation overhead
    - Evaluation of expression trees and predicates are compiled away
- Projections become no-ops
- No intermediate vectors are produced

Completed

- Support majority of SQL types
- Selections
    - With arbitrarily complex predicates
- Projections
- Scans
- Hash-based aggregations
    - All aggregates except FIRST
- Hash-joins (only INNER)
- Order-by sorting
- Run-time switch to enable/disable compilation
- Postgres integration
- The main components to a completely layout-agnostic JITing query engine

TBD

- Case expressions
- ANTI and SEMI joins
- Subquery
- Subquery expressions
  - Partly through
- Exploiting SIMD

Goals

- 75% (Completed)
  - Support scans over hybrid layouts
- 100% (Completed)
  - Support joins, aggregations, projections and sorting
- 125% (Not complete)
  - Support a majority of TPC-H queries