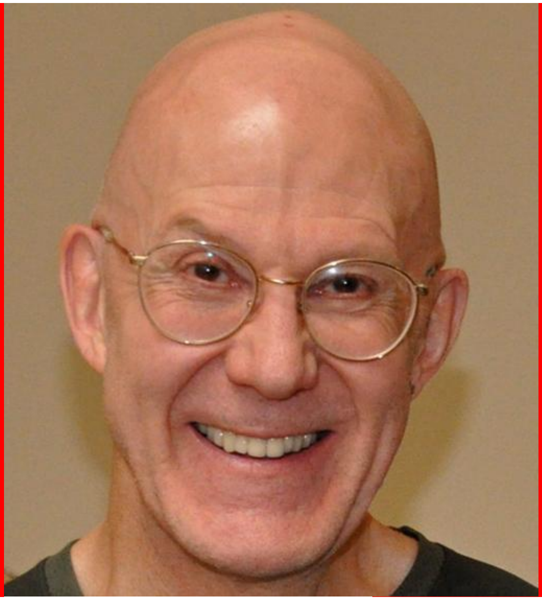


Using the PL/SQL Hierarchical Performance Profiler

Bryn Llewellyn
Product Manager, Database Server Technologies Division, Oracle HQ

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle’s products remain at the sole discretion of Oracle.

Overview

- You can can get information like this:
  - List of subprograms and SQL statements that were executed during the run, ordered by the elapsed time
  - For a particular subprogram, the time spent in itself and the time spent in each of the subprograms it calls
  - For a particular subprogram, the list of subprograms that call it ordered by the total time for those calls
- This information guides you efficiently to the code whose optimization will have the greatest effect

Agenda

- Hierarchical *vs* statement-oriented profiling
- The Hprof Operating model
- What information is delivered?
- Some case studies; looking at the reports
- Use plshprof canned HTML reports or roll your own
- Summary: the *method*

Hierarchical *vs* statement-oriented profiling

- **DBMS_Profiler** watches statements
  - How many times was each statement executed?
    For each, how much time was spent on those executions?
- Doesn’t know about the subprograms within a package…
  - … let alone inner subprograms (arbitrarily deeply nested) within those
- Has no notion of “self time” vs “total time”
  - Both the time for the statement *p()* and the the time for all the statements that *p()* executes show up.
    You have to puzzle it out.

Hierarchical *vs* statement-oriented profiling

- **DBMS_Hprof** watches as control moves into and back from subprograms
  - Records each transition – *i.e.* the explicit call history
  - Notes the time spent between each transition
- No end of interesting reports can be derived from this raw data
  - Allows computing *both* a function's self-time *and* a function's total (a.k.a. subtree) time
- Such reports cannot be derived from bald per-statement times 'cos the overall context is never recorded

Agenda

- Hierarchical *vs* statement-oriented profiling
- **The Hprof Operating model**
- What information is delivered?
- Some case studies; looking at the reports
- Use plshprof canned HTML reports or roll your own
- Summary: the *method*

The Hprof Operating model

- DBA nominates a directory on the database machine’s filesystem and gives the developer’s o/s user read/write access to it
- DBA maps the o/s directory to a directory object and grants the developer’s Oracle user read/write access to it

-
  ```
  begin
    DBMS_Hprof.Start_Profiling('DIR', 'My_Run_1.trc');
    My_Proc();
    DBMS_Hprof.Stop_Profiling();
  end;
  ```
- Format the raw data for human browsing *(plshprof)*
- No installation or configuration.
  No need to “instrument” your code.

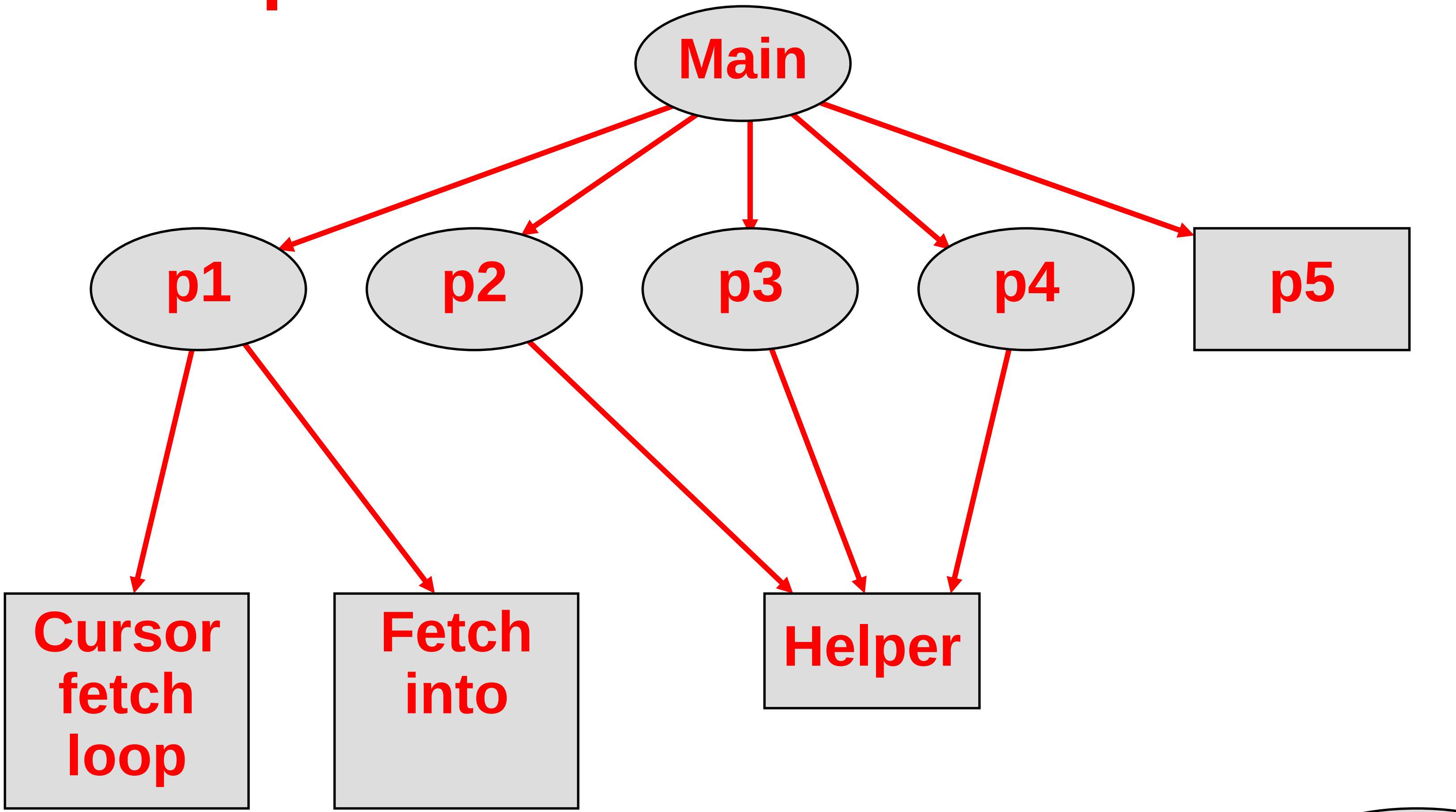
Agenda

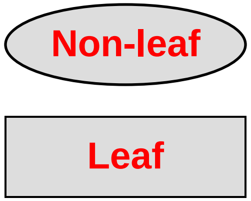
- Hierarchical *vs* statement-oriented profiling
- The Hprof Operating model
- **What information is delivered?**
- Some case studies; looking at the reports
- Use plshprof canned HTML reports or roll your own
- Summary: the *method*

How are subprograms identified?

- Namespace (PL/SQL or SQL)
- Owner
- Unit Name
- Path to subprogram from top of unit
- Source code line number (to distinguish overloads)
- System-generated names
  - __pkg_init
  - __static_sql_exec_line*NNN*
  - __sql_fetch_line*NNN*
  - __dyn_sql_exec_line*NNN*
  - __plsql_vm
  - __anonymous_block

Example 0





- This is the dynamic call graph of a particular execution of *Main*

Information derived from the raw trace

No.of calls

Some_Subprogram

Subtree time = Self time + Callees time

Information derived – *continued…*

“function” time

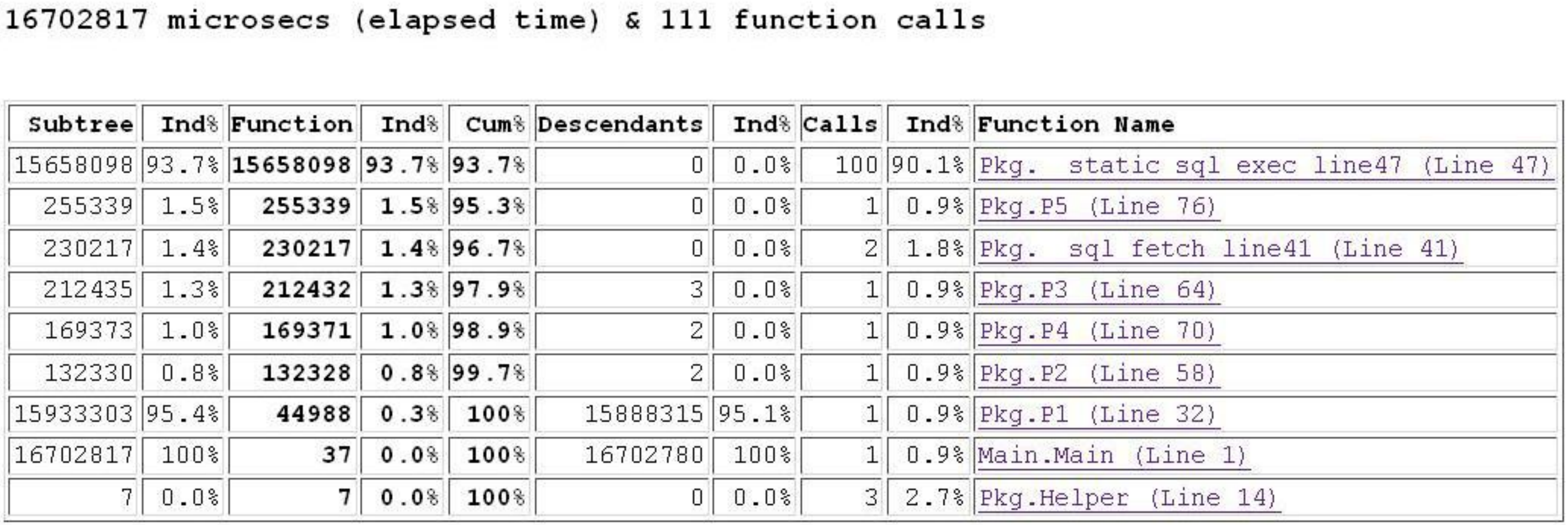
“descendants” time

Subtree time	Self time	Callees time	No.of calls	Name
15.658098	**15.**658098	**0.**000000	**100**	**Static SQL Exec**
0.255339	**0.**255339	**0.**000000	**1**	**P5**
0.230217	**0.**230217	**0.**000000	**2**	**SQL Fetch**
…	…	…	…	…
15.933303	**0.**044988	**15.**888315	**1**	**P1**
16.702817	**0.**000037	**16.**702780	**1**	**Main**
0.000007	**0.**000007	**0.**000000	**3**	**Helper**

sort by descending self time

total
16.702817

Sort by descending self time



16702817 microsecs (elapsed time) & 111 function calls

Subtree	Ind%	Function	Ind%	Cum%	Descendants	Ind%	Calls	Ind%	Function Name
15658098	93.7%	15658098	93.7%	93.7%	0	0.0%	100	90.1%	Pkg. static sql exec line47 (Line 47)
255339	1.5%	255339	1.5%	95.3%	0	0.0%	1	0.9%	Pkg.P5 (Line 76)
230217	1.4%	230217	1.4%	96.7%	0	0.0%	2	1.8%	Pkg. sql fetch line41 (Line 41)
212435	1.3%	212432	1.3%	97.9%	3	0.0%	1	0.9%	Pkg.P3 (Line 64)
169373	1.0%	169371	1.0%	98.9%	2	0.0%	1	0.9%	Pkg.P4 (Line 70)
132330	0.8%	132328	0.8%	99.7%	2	0.0%	1	0.9%	Pkg.P2 (Line 58)
15933303	95.4%	44988	0.3%	100%	15888315	95.1%	1	0.9%	Pkg.P1 (Line 32)
16702817	100%	37	0.0%	100%	16702780	100%	1	0.9%	Main.Main (Line 1)
7	0.0%	7	0.0%	100%	0	0.0%	3	2.7%	Pkg.Helper (Line 14)

The Heisenberg effect

```
 Caption constant varchar2(35) :=  'Elapsed time '||
    $if $$Profiling $then   '(profiling ON)'
    $else                   '(profiling OFF)'
    $end;
  t0 constant integer not null :=
                                DBMS_Utility.Get_Time();
  t integer not null := 0;
begin
  $if $$Profiling $then
    DBMS_Hprof.Start_Profiling('PLSHPROF', 'Run_1.trc');
  $end
  Main();
  $if $$Profiling $then
    DBMS_Hprof.Stop_Profiling();
  $end
  t := DBMS_Utility.Get_Time() - t0;
  DBMS_Output.Put_Line(Caption||Lpad(t, 5));
end;
```

The Heisenberg effect

	seconds
Self-reported* (profiling OFF)	**16.66**
Self-reported (profiling ON)	**16.79**
Hprof-reported	**16.67**

* The self-reported times were done using *DBMS_Utility.Get_Time()*

The Heisenberg effect…

But it’s not always as nice as this!

Information derived – *continued…*

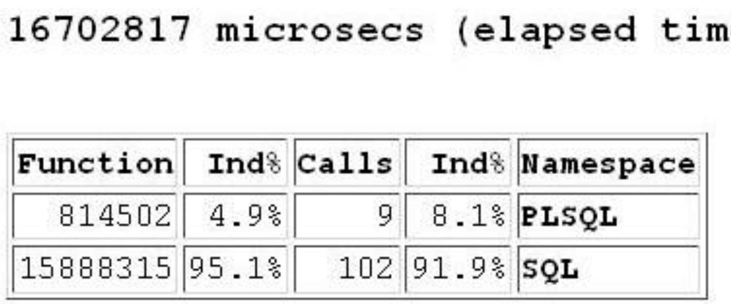
- Order by
  - Self time
  - Subtree time
  - No.of calls
  - Callees time
  - Alphabetically by name

Information derived – *continued…*

- Rollup by PL/SQL vs SQL (a.k.a. “namespace”)
  - Order by self time
  - Order by no.of calls
  - Order by namespace

- Rollup by PL/SQL Unit (a.k.a. “module”)
  - Order by self time
  - Order by no.of calls
  - Order by name

Sort by elapsed time in namespace



```
16702817 microsecs (elapsed time) & 111 function calls
```

Function	Ind%	Calls	Ind%	Namespace
814502	4.9%	9	8.1%	**PLSQL**
15888315	95.1%	102	91.9%	**SQL**

- Say no more!

  This one isn’t a PL/SQL performance exercise.

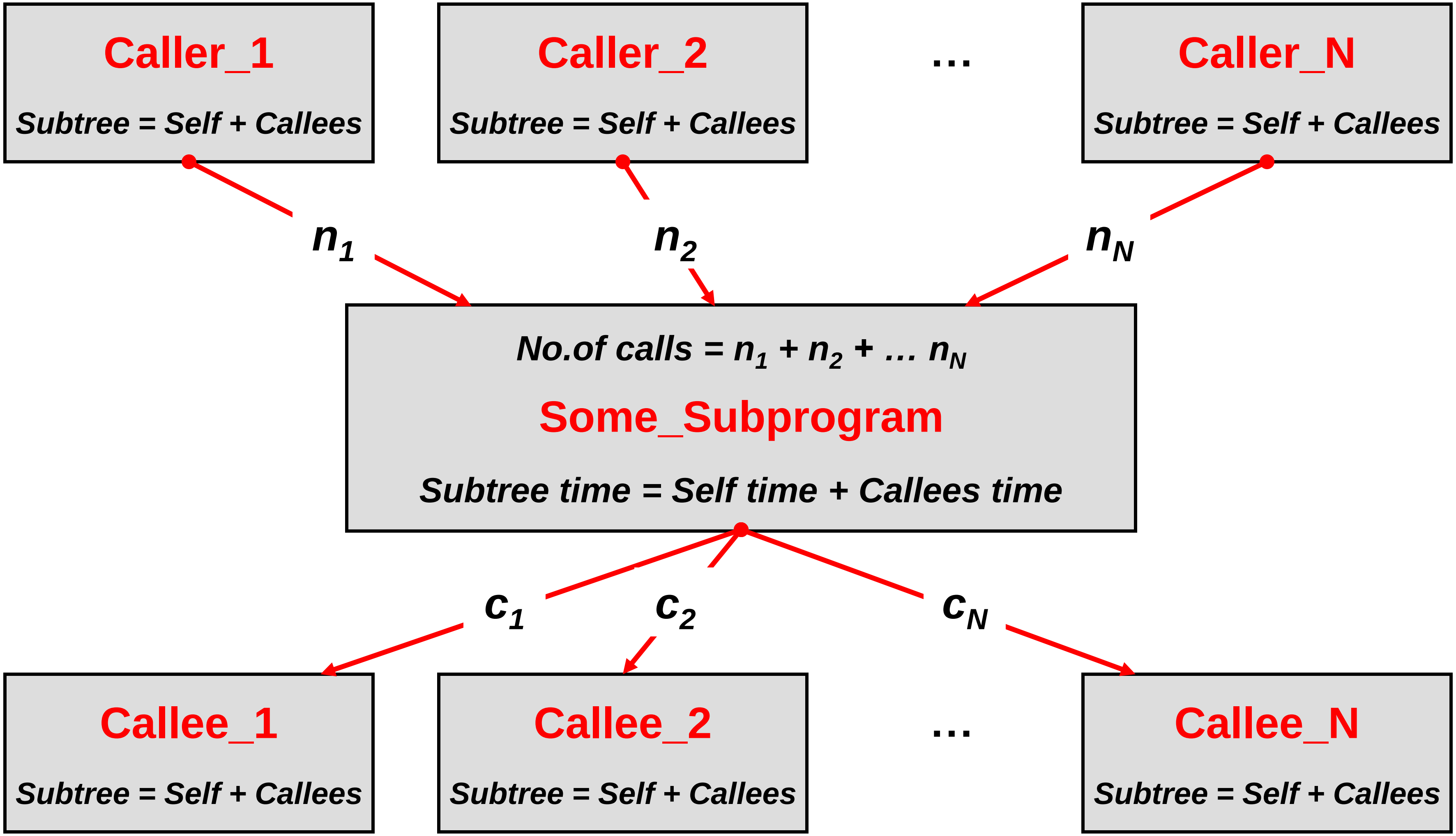
Information derived – *continued…*

No.of calls

Some_Subprogram

Subtree time = Self time + Callees time

Information derived – *continued…*



Caller_1

Subtree = Self + Callees

Caller_2

Subtree = Self + Callees

…

Caller_N

Subtree = Self + Callees

n_1 n_2 n_N

No.of calls = $n_1 + n_2 + \dots n_N$

Some_Subprogram

Subtree time = Self time + Callees time

c_1 c_2 c_N

Callee_1

Subtree = Self + Callees

Callee_2

Subtree = Self + Callees

…

Callee_N

Subtree = Self + Callees

Information derived – *continued…*

- For each caller, we see:
  - How many times it calls *Some_Subprogram*
  - That portion of *Some_Subprogram*’s time consumption for which that caller is responsible
  - The sum of these, over the callers, is equal to the figures noted for *Some_Subprogram* itself
- Each caller might not call *Some_Subprogram* in each call to it
- For each child, we see:
  - How many times it was called by *Some_Subprogram*
  - Its time consumption when called from *Some_Subprogram*
- Each child may be called by other subprograms

Information derived – *continued…*

Subtree time	Self time	Callees time	No.of calls	Name
42.448067	1.577936	40.870131	421	Some_Subprogram

35.240135	1.215322	34.024813	323	Caller_1
6.917576	0.337475	6.580101	91	Caller_2
…	…	…	…	…
0.290356	0.025139	0.265217	7	Caller_N

28.908427	28.908427	0.000000	8712	Callee_1
7.266276	7.266276	0.000000	6495	Callee_2
…	…	…	…	…
4.695428	4.695428	0.000000	1435	Callee_N

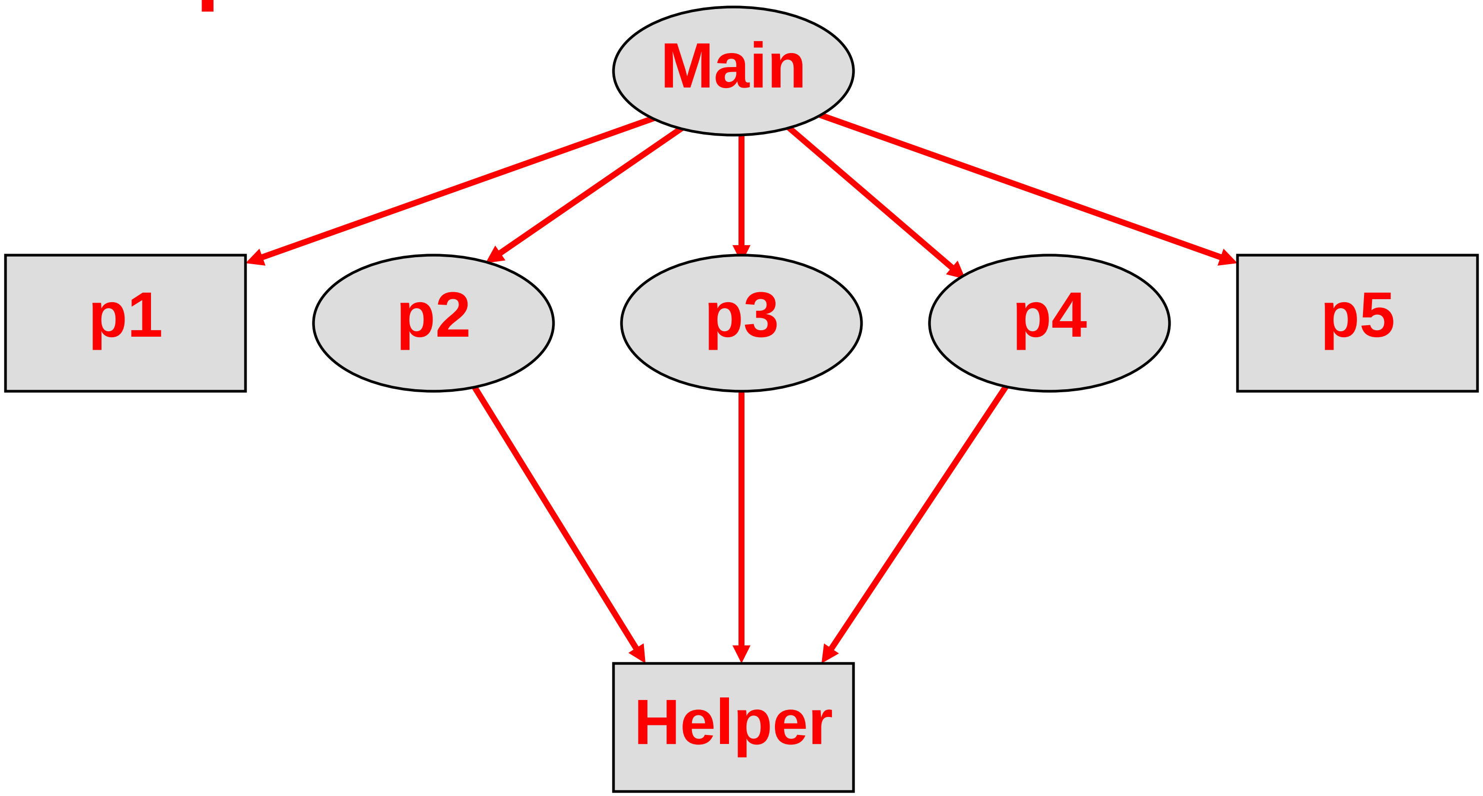
Live exploration of available report

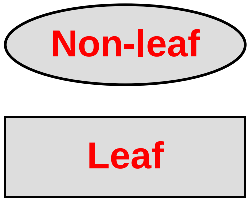
- Order by
  - Self time
  - Subtree time
  - No.of calls
  - Alphabetically by name
- Navigating up to a caller and down to a callee
  - Start with Callee_3 (has the biggest self time)
  - Navigate to *Some_Subprogram*
  - Look at all of *Some_Subprogram*’s callers and callees
- ⌘ – 0

Agenda

- Hierarchical *vs* statement-oriented profiling
- The Hprof Operating model
- What information is delivered?
- **Some case studies; looking at the reports**
- Use plshprof canned HTML reports or roll your own
- Summary: the *method*

Example 1





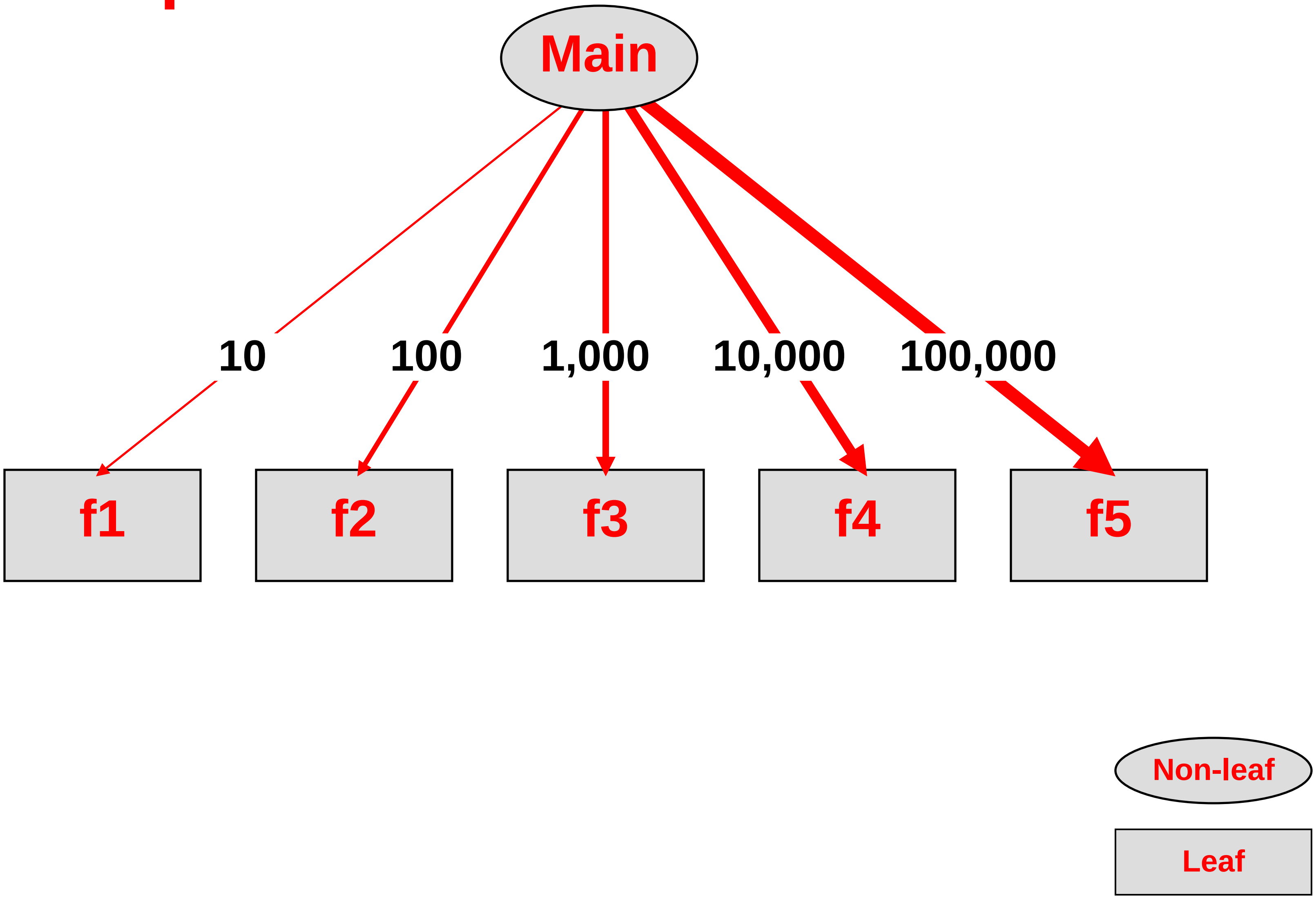
Live exploration of available reports

- Order by
  - Self time

- There’s an obvious culprit!
- Fix it
- Do another Hprof run
- Look at the new report
- Look at the *difference* report

- ⌘ – 1 ⌘ – 2 ⌘ – 3

Example 2



Live exploration of available reports

- Order by
  - Self time

- ⌘ – 4 ⌘ – 5 ⌘ – 6

Live exploration of available reports

- Order by
  - Self time

- Both *f5* and *Main* have a very big self time
- Together, these dominate
- But *Main* does no “real work”
- And, looking at *f5*, it’s very lightweight
- But Main calls *f5* 100,000 times!
- All the time is going on the mechanics of calling
- The fix is to inline *f5* into *Main*

- ⌘ – 4 ⌘ – 5 ⌘ – 6

The Heisenberg effect – Example 2

	seconds
Self-reported* (profiling OFF)	0.06
Self-reported (profiling ON)	2.46
Hprof-reported	0.50

No.of calls = 111,112

Self-reported* (profiling OFF)	0.02
Self-reported (profiling ON)	0.33
Hprof-reported	0.06

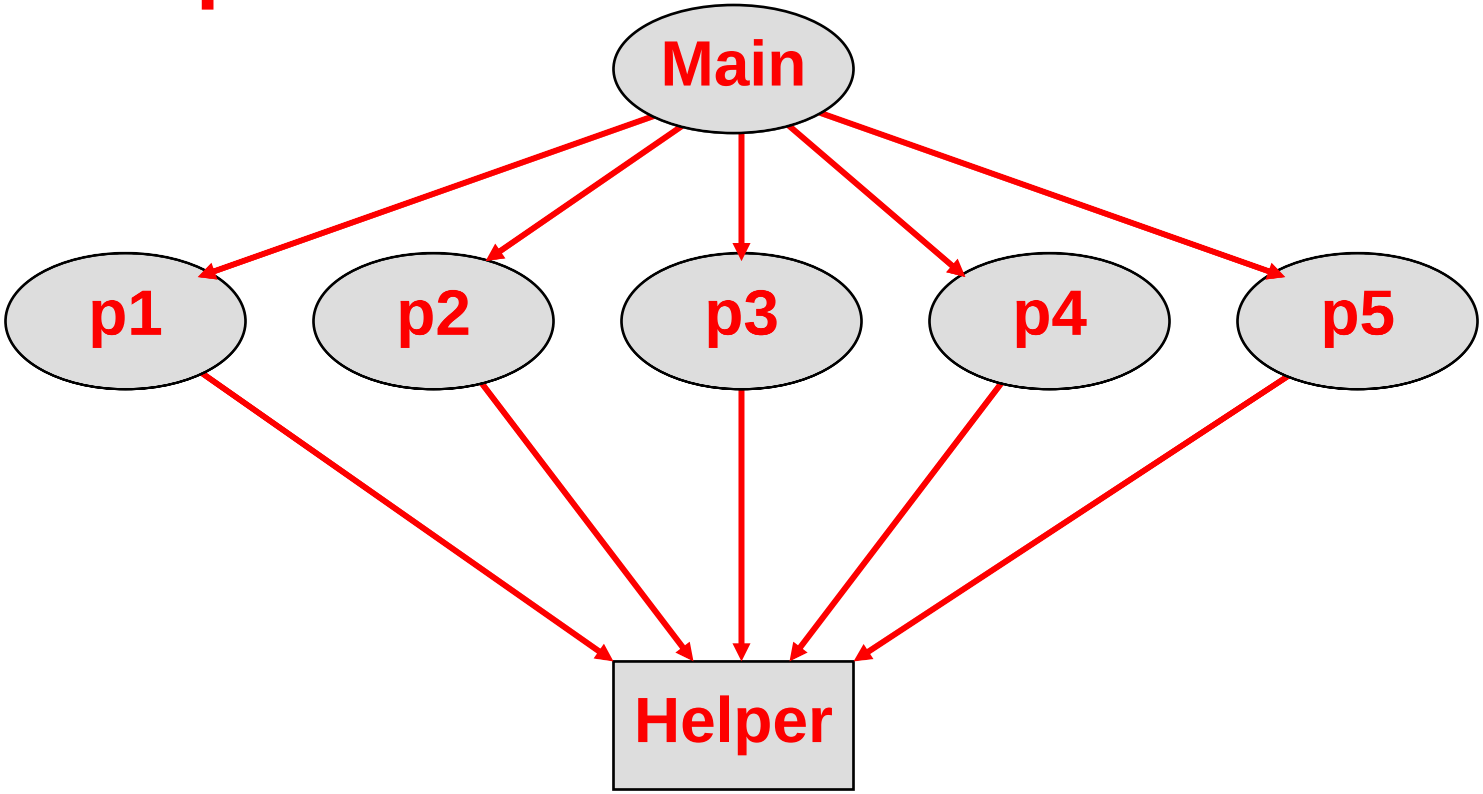
No.of calls =   11,112

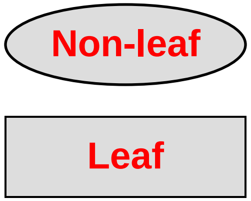
The Heisenberg effect – Example 0

	seconds
Self-reported* (profiling OFF)	16.66
Self-reported (profiling ON)	16.79
Hprof-reported	16.67

No.of calls = 112

Example 3





Live exploration of available reports

- Order by
  - Self time

- There’s something fishy with *Helper*

- ⌘ – 7 ⌘ – 8 ⌘ – 9

Live exploration of available reports

- Order by
  - Self time

- There’s something fishy with *Helper*
- Its self time when called from *p3* is hugely bigger than when called from elsewhere
- Ah…

  p3 called it with an actual requesting self-tracing!

- ⌘ – 7 ⌘ – 8 ⌘ – 9

Agenda

- Hierarchical *vs* statement-oriented profiling
- The Hprof Operating model
- What information is delivered?
- Some case studies; looking at the reports
- **Use plshprof canned HTML reports or roll your own**
- Summary: the *method*

Use plshprof canned HTML reports or roll your own

- Run the script rdbms/admin/dbmshptab.sql

-
  ```
  Unique_Run_ID := DBMS_Hprof.Analyze(
    Location          The_Directory,
    Filename          The_Filename,
    Summary_Mode      Full_Analysis,
    ...               ,
    Run_Comment       The_Run_Comment);
  ```
- Tables are populated with data sufficent to let you write reports with the same information content as the supplied ones
- You could use APEX

Agenda

- Hierarchical *vs* statement-oriented profiling
- The Hprof Operating model
- What information is delivered?
- Some case studies; looking at the reports
- Use plshprof canned HTML reports or roll your own
- **Summary: the *method***

Summary: the *method*

```
if
    the SQL time dominates the PL/SQL time
then
    Stop obsessing about your PL/SQL performance
    and fix the SQL;
elsif
    …
```

Summary: the *method*

```
elsif
    one PL/SQL subprogram, p1, has a dominant self time
then
    Fix the implementation of p1;
elsif
    …
```

- If you can’t spot the problem in p1 just by reading the code and thinking about it (e.g. binary search using index-by-varchar2 table vs pre-9.2 linear scan)…
- Then this is where you might want statement-level profiling

Summary: the *method*

```
elsif

     (one PL/SQL subprogram that ought to be quick, p2,
     has a very big self time)
     and
     (p2's caller has a surprisingly big self time)

then

     Check how many times the caller calls p2;
     if
       p2 is called a huge number of times
     then
       Inline p2 into its caller;
     end if;

elsif

     …
```

Summary: the *method*

```
elsif
      (one PL/SQL subprogram that ought to be quick, p3,
      has a very big self time)
      and
      (p3 is called by many callers)
       and
      (p3's self time depends hugely on who calls it)
then
      Check for the explanation;
      if
        p3 is called in self-tracing mode from just one caller
      then
        Rewrite the call so's not to ask for self-tracing;
      end if;
elsif …
```

Summary: the *method*

```
else
    Sort the report by subtree time;

    (Mentally) prune away the quick subtrees;

    Focus attention on the slowest subtree and understand its purpose;

    Understand the design and consider alternative designs that implement the same purpose;

    Tell your manager that this one is going to be hard;
end if;
```

Finally…

For more information…

- The PL/SQL hierarchical performance profiler
  is documented
  in the

  Oracle Database
  Advanced Application Developer’s Guide

Q&A