the gamedesigninitiative at cornell university

Lecture 18

Profiling & Optimization

Sources of Game Performance Issues?

the gamedesigninitiative at cornell university

Avoid Premature Optimization

- Novice developers rely on **ad hoc** optimization
  - Make private data public
  - Force function inlining
  - Decrease code modularity

  } removes function calls

- But this is a **very bad idea**
  - Rarely gives significant performance benefits
  - Non-modular code is very hard to maintain
- Write clean code first; optimize later

the gamedesigninitiative at cornell university

Performance Tuning

- Code follows an 80/20 rule (or even 90/10)
  - 80% of run-time spent in 20% of the code
  - Optimizing other 80% provides little benefit
  - Do nothing until you know what this 20% is
- Be careful in **tuning performance**
  - Never overtune some inputs at expense of others
  - Always focus on the overall algorithm first
  - Think hard before making non-modular changes

the gamedesigninitiative at cornell university

What Can We **Measure**?

Time Performance

- What code takes most time
- What is called most often
- How long I/O takes to finish
- Time to switch threads
- Time threads hold locks
- Time threads wait for locks

Memory Performance

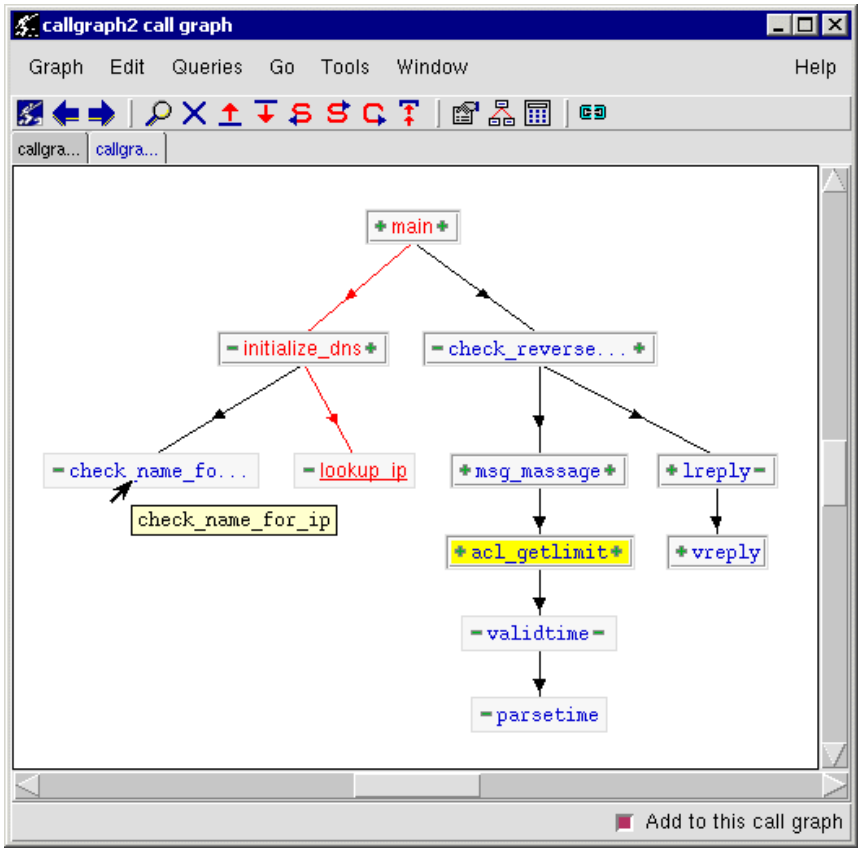
- Number of heap allocations
- Location of allocations
- Timing of allocations
- Location of releases
- Timing of releases
- (Location of memory leaks)

the gamedesigninitiative at cornell university

Analysis Methods

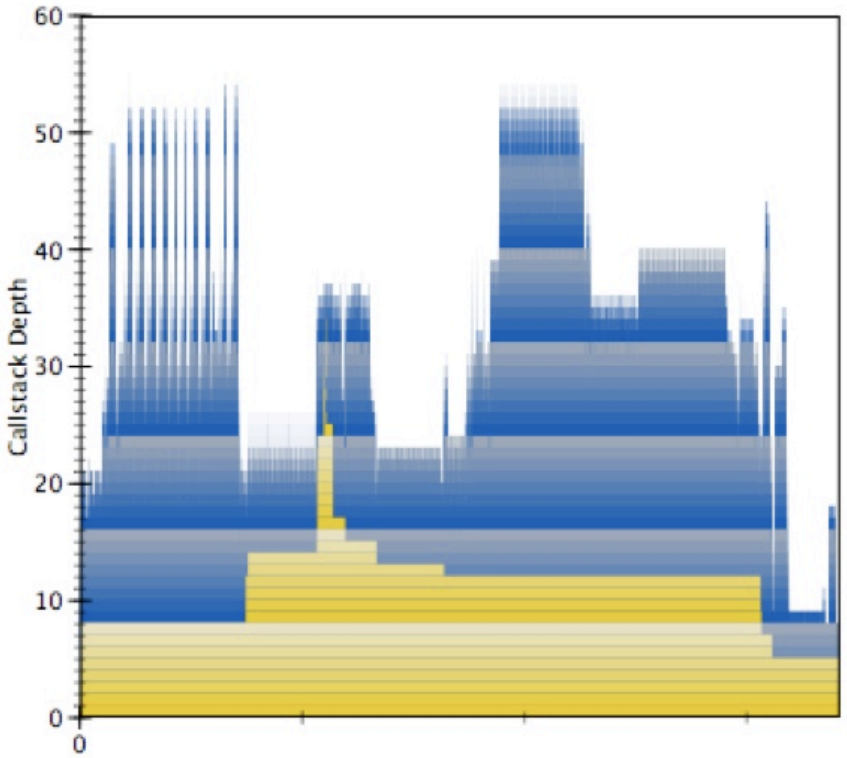
Static Analysis

- Analyze without running
  - Relies on language features
  - Major area of PL research
- **Advantages**
  - Offline; no performance hit
  - Can analyze deep properties
- **Disadvantages**
  - Conservative; misses a lot
  - Cannot capture user input



the gamedesigninitiative at cornell university

Analysis Methods



Profiling

- Analysis runs with program
  - Record behavior of program
  - Helps visualize this record
- **Advantages**
  - More data than static anal.
  - Can capture user input
- **Disadvantages**
  - Hurts performance a lot
  - May *alter* program behavior

Analysis Methods

Static Analysis

- Analyze without running
  - Relies on language features
  - Major area of PL research
- **Advantages**
  - Offline; no performance hit
  - Can analyze deep properties
- **Disadvantages**
  - Conservative; misses a lot
  - Cannot capture user input

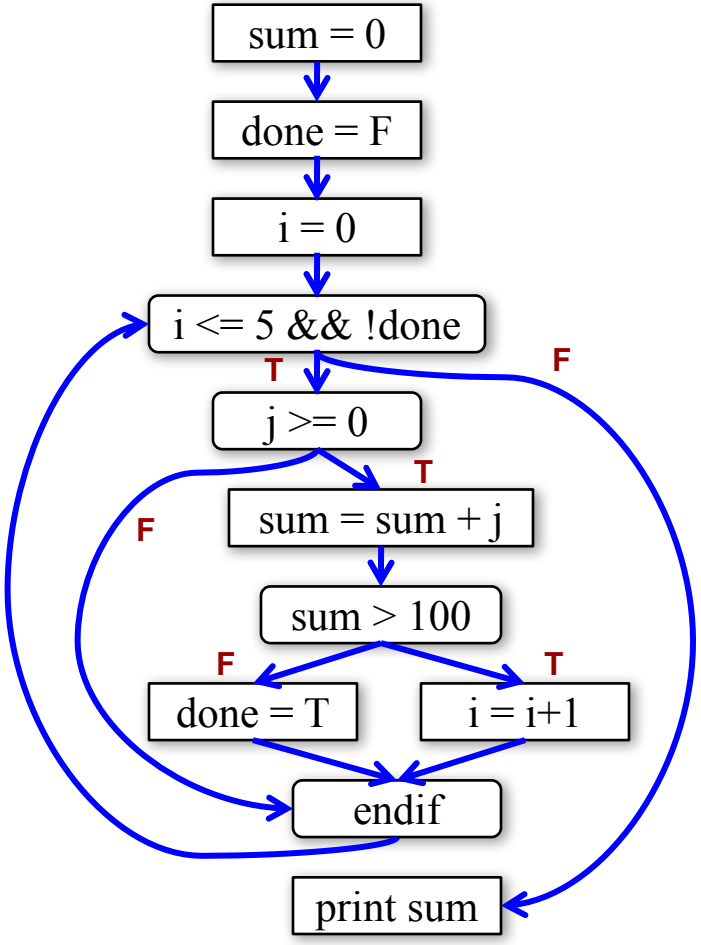
Profiling

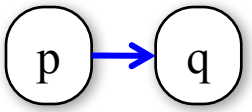
- Analysis runs with program
  - Record behavior of program
  - Helps visualize this record
- **Advantages**
  - More data than static anal.
  - Can capture user input
- **Disadvantages**
  - Hurts performance a lot
  - May *alter* program behavior

the gamedesigninitiative at cornell university

Static Analysis: Control Flow

```
int sum = 0
boolean done = false;
for(int ii; ii<=5 &&!done;) {
   if (j >= 0) {
      sum += j;
      if (sum > 100) {
         done = true;
      } else {
         i = i+1;
      }
}}
print(sum);
```

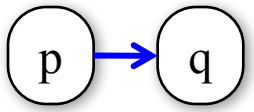




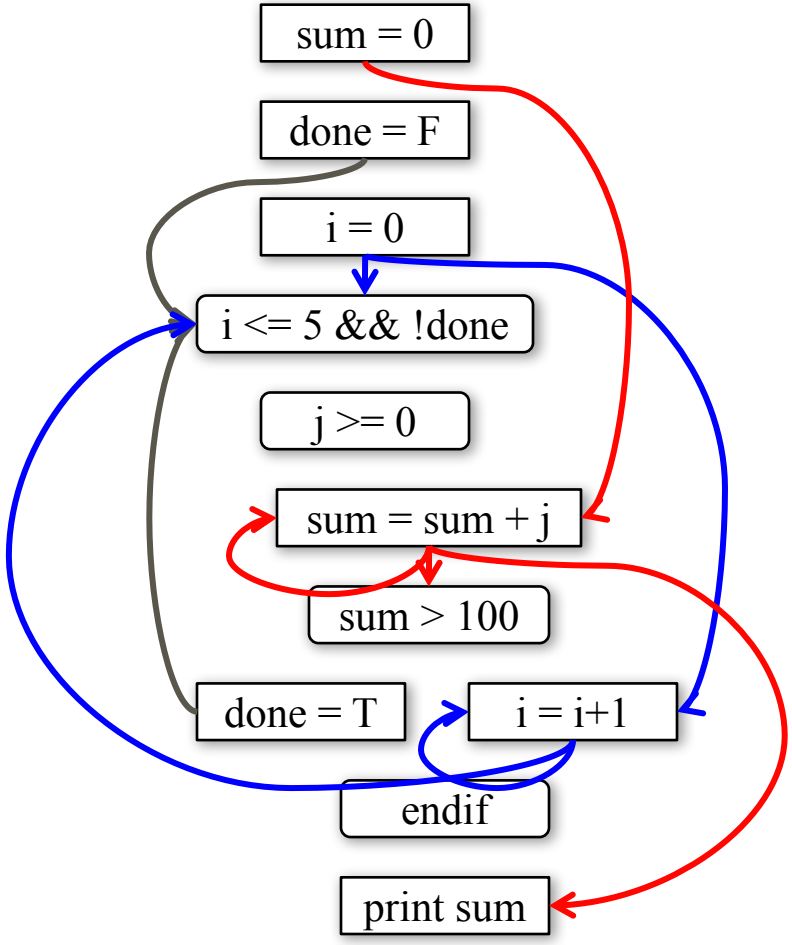
q may be executed immediately after p

Static Analysis: Flow Dependence

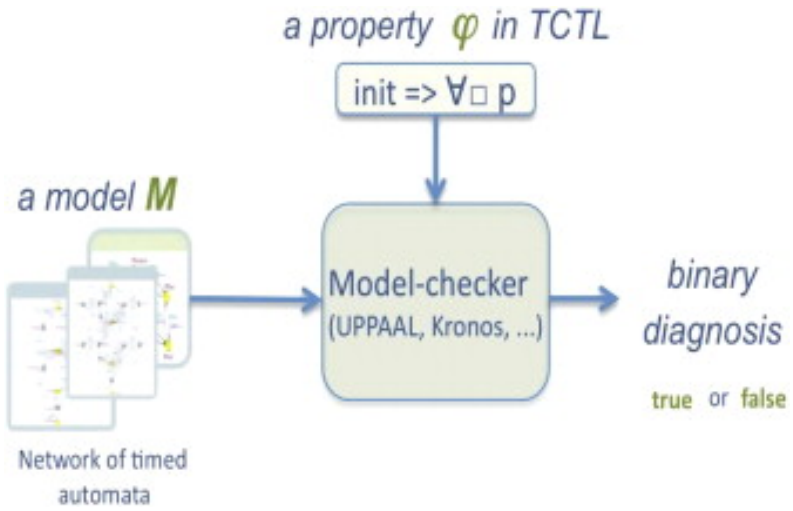
```
int sum = 0
boolean done = false;
for(int ii; ii<=5 &&!done;) {
   if (j >= 0) {
      sum += j;
      if (sum > 100) {
         done = true;
      } else {
         i = i+1;
      }
}}
print(sum);
```

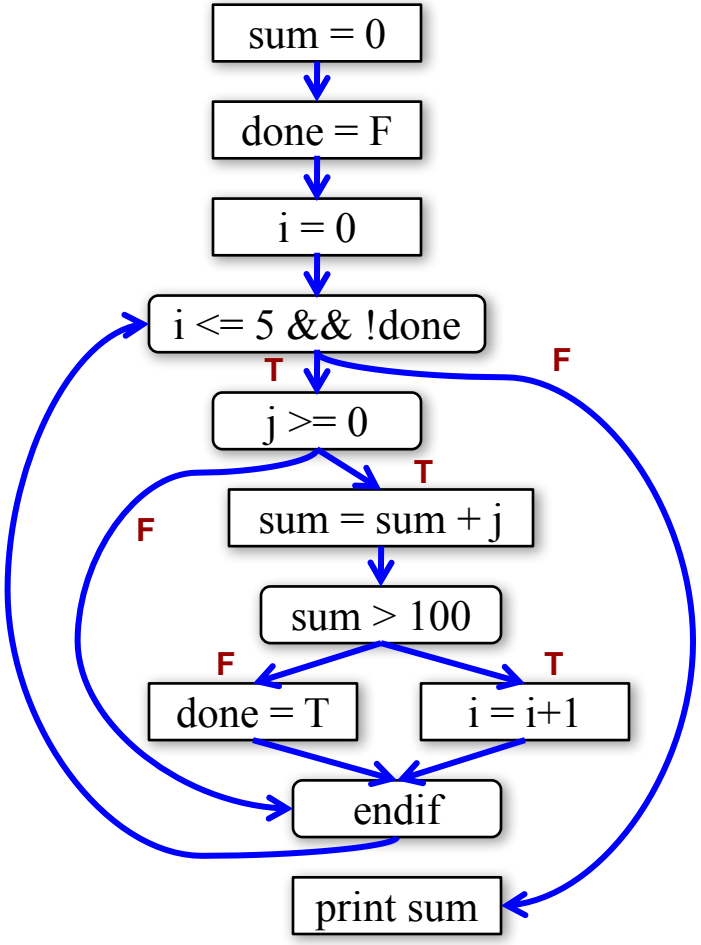


Value assigned at p
is read at command q



Model Checking



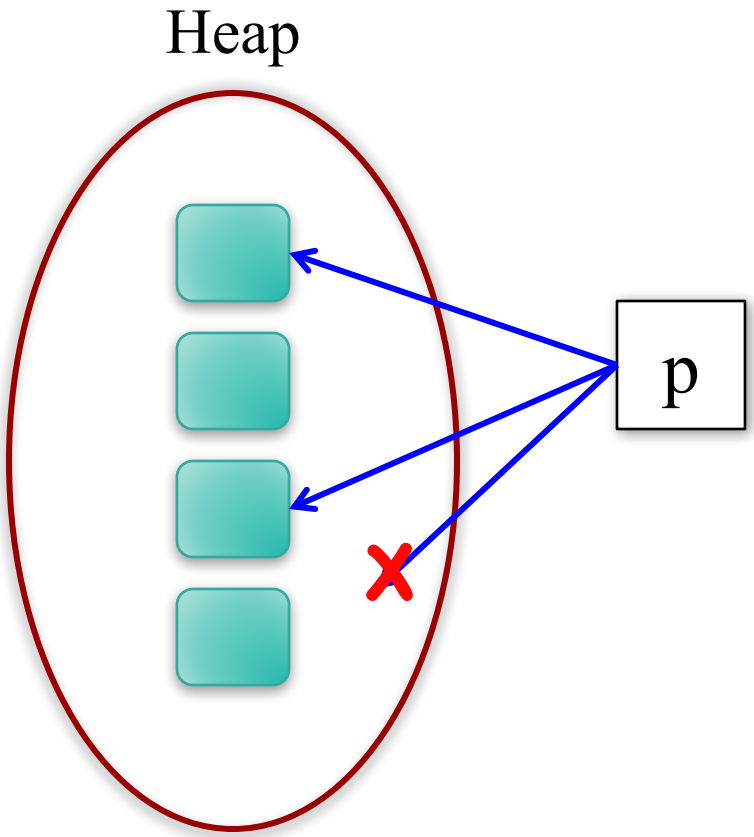


- Given a graph, logical formula φ
  - φ expresses properties of graph
  - Checker determines if is true
- Often applied to software
  - Program as control-flow graph
  - φ indicates acceptable paths

the gamedesigninitiative at cornell university

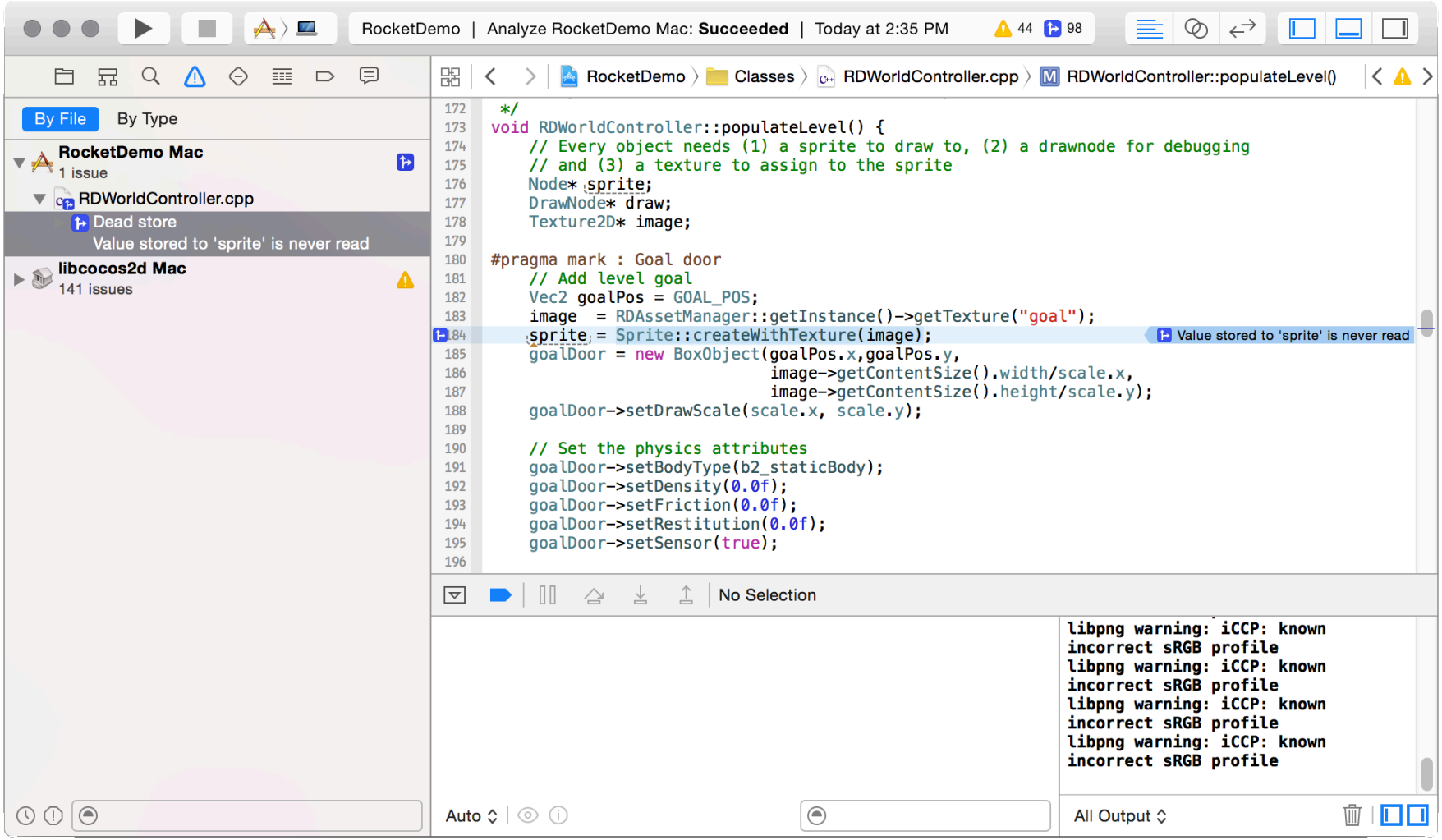
Static Analysis: Applications

- **Pointer analysis**
  - Look at pointer variables
  - Determine possible values for variable at each place
  - Can find memory leaks
- **Deadlock detection**
  - Locks are flow dependency
  - Determine possible owners of lock at each position
- **Dead code analysis**

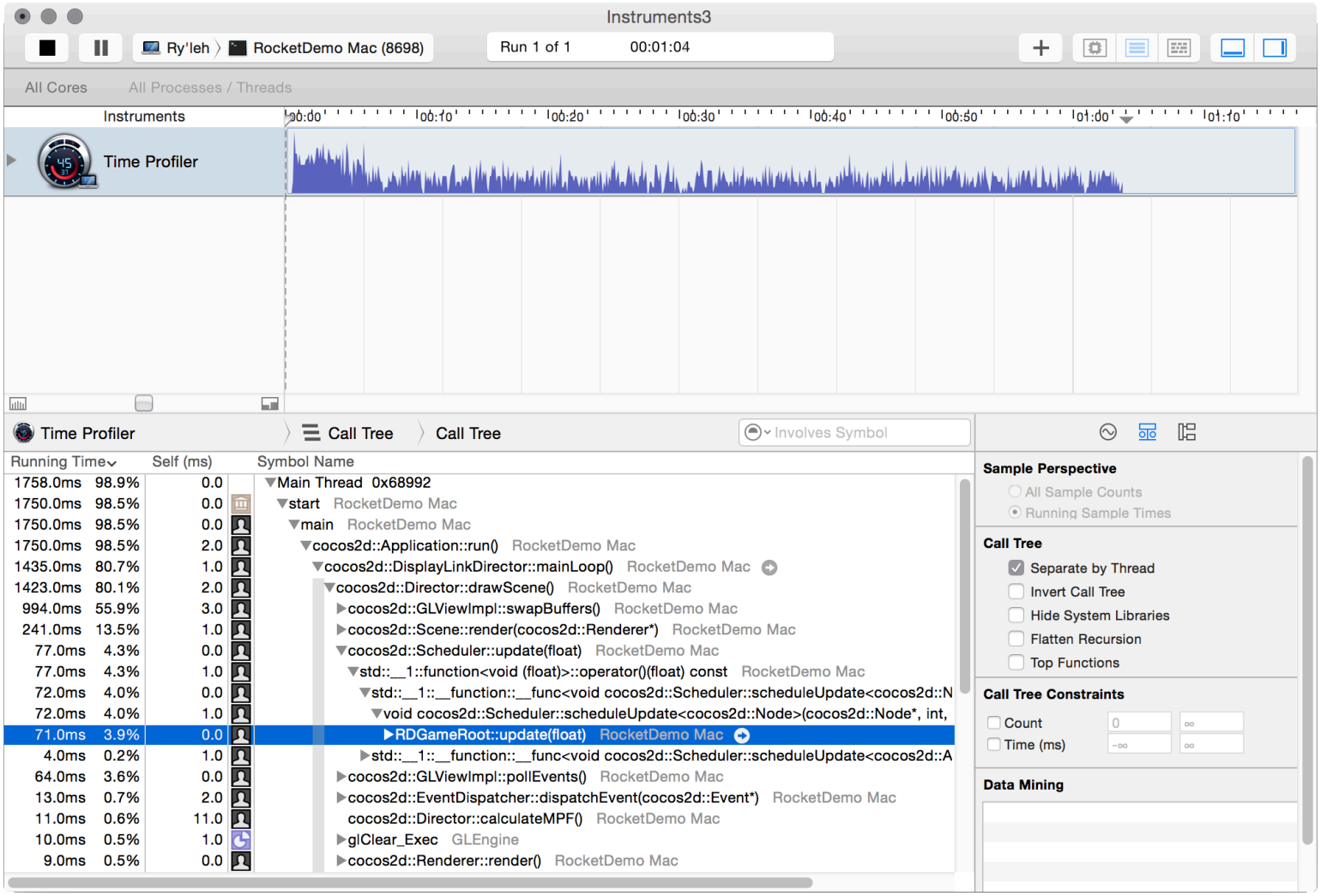


the gamedesigninitiative at cornell university

Example: Analyze in X-Code



Time Profiling



Time Profiling: Methods

Software

- Code added to program
  - Captures start of function
  - Captures end of function
  - Subtract to get time spent
  - Calculate percentage at end
- **Not completely accurate**
  - Changes actual program
  - Also, how get the time?

Hardware

- Measurements in hardware
  - Feature attached to CPU
  - Does not change how the program is run
- Simulate w/ hypervisors
  - Virtual machine for Oss
  - VM includes profiling measurement features
  - **Example**: Xen Hypervisor

the gamedesigninitiative at cornell university

Time Profiling: Methods

Time-Sampling

- Count at periodic intervals
  - Wakes up from sleep
  - Looks at parent function
  - Adds that to the count
- Relatively lower overhead
  - Doesn't count everything
  - Performance hit acceptable
- May miss small functions

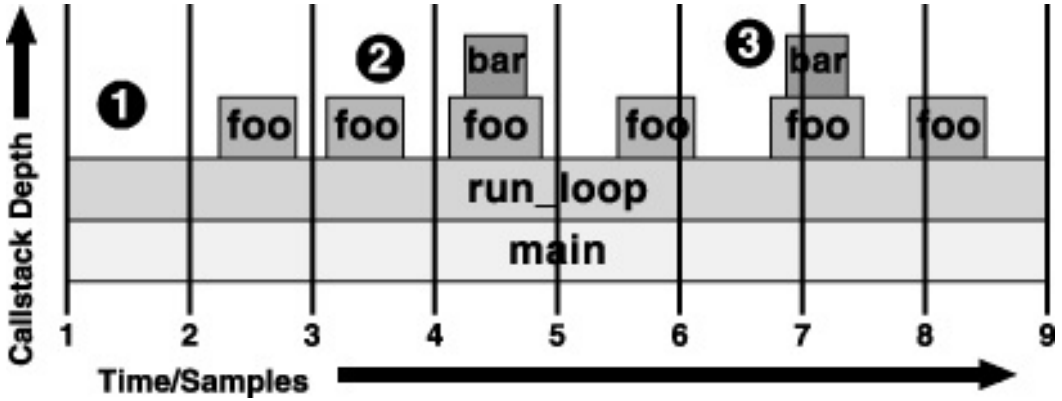
Instrumentation

- Count pre-specified places
  - Specific function calls
  - Hardware interrupts
- Different from sampling
  - Still not getting everything
  - But **exact view** of slice
- Used for targeted searches

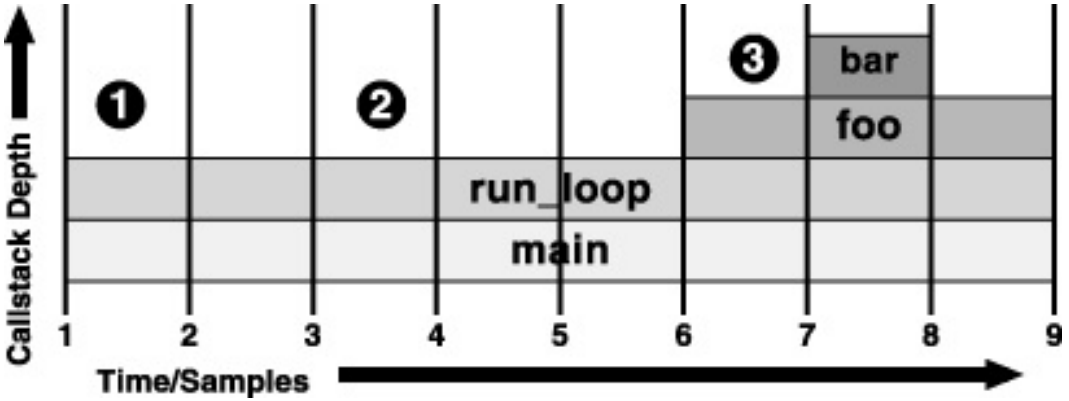
the gamedesigninitiative at cornell university

Issues with Periodic Sampling

Real

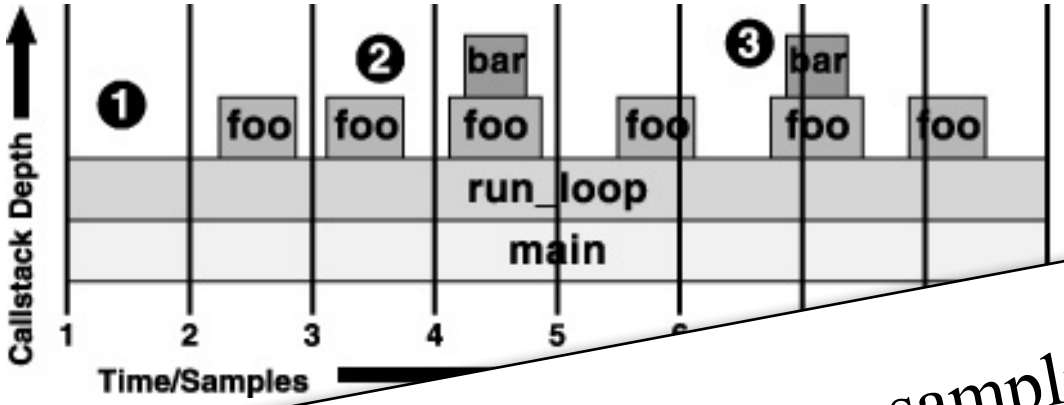


Sampled

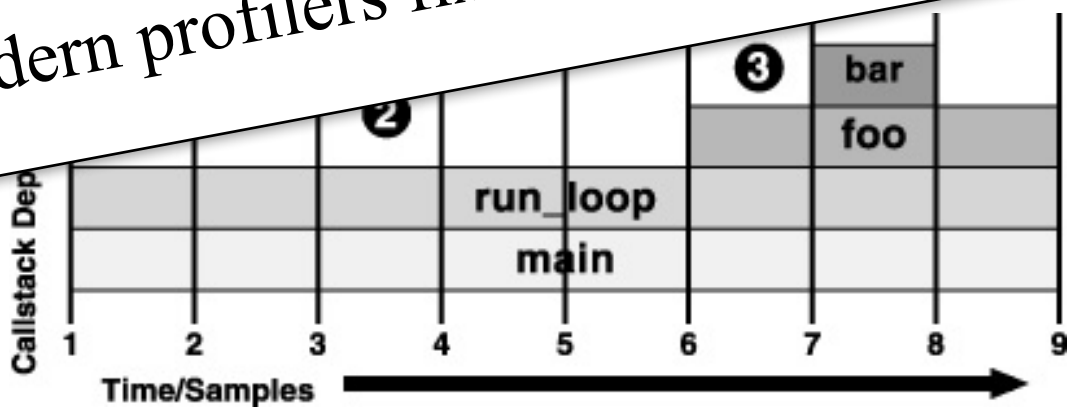


Issues with Periodic Sampling

Real



Sampled



Modern profilers fix with random sampling

the gamedesigninitiative at cornell university

What Can We **Measure**?

Time Performance

- What code takes most time
- What is called most often
- How long I/O takes to finish
- Time to switch threads
- Time threads hold locks
- Time threads wait for locks

Memory Performance

- Number of heap allocations
- Location of allocations
- Timing of allocations
- Location of releases
- Timing of releases
- (Location of memory leaks)

What Can We Instrument?

Time Performance

- What code takes most time
- What is called most often
- How long I/O takes to finish
- Time to switch threads
- Time threads hold locks
- Time threads wait for locks

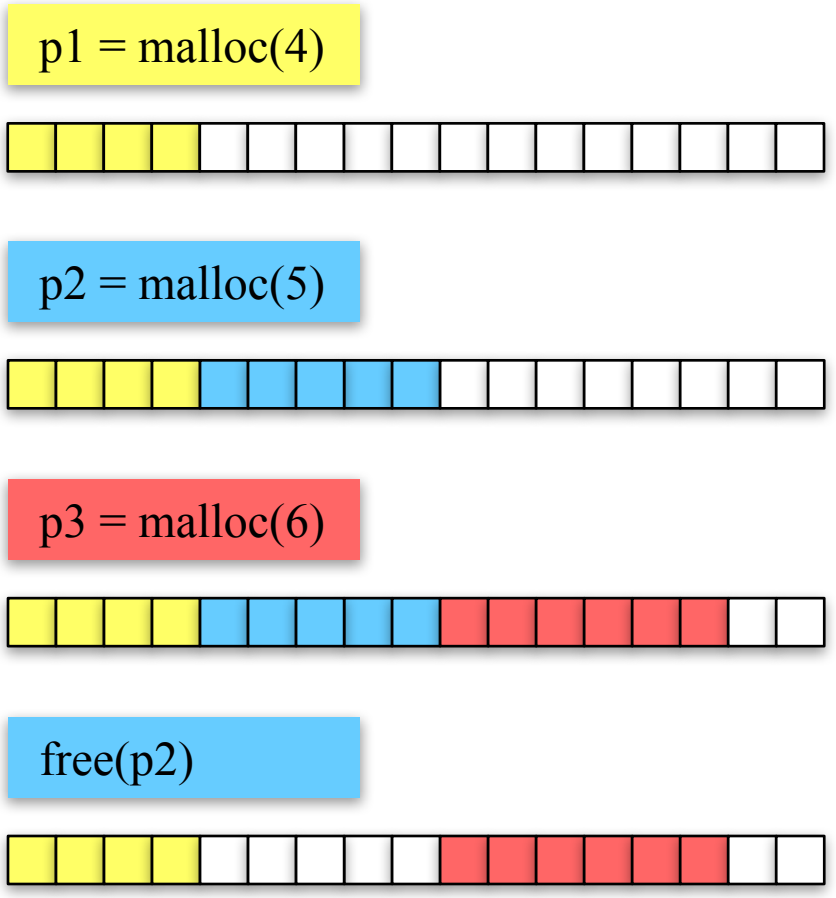
Memory Performance

- Number of heap allocations
- Location of allocations
- Timing of allocations
- Location of releases
- Timing of releases
- (Location of memory leaks)

the gamedesigninitiative at cornell university

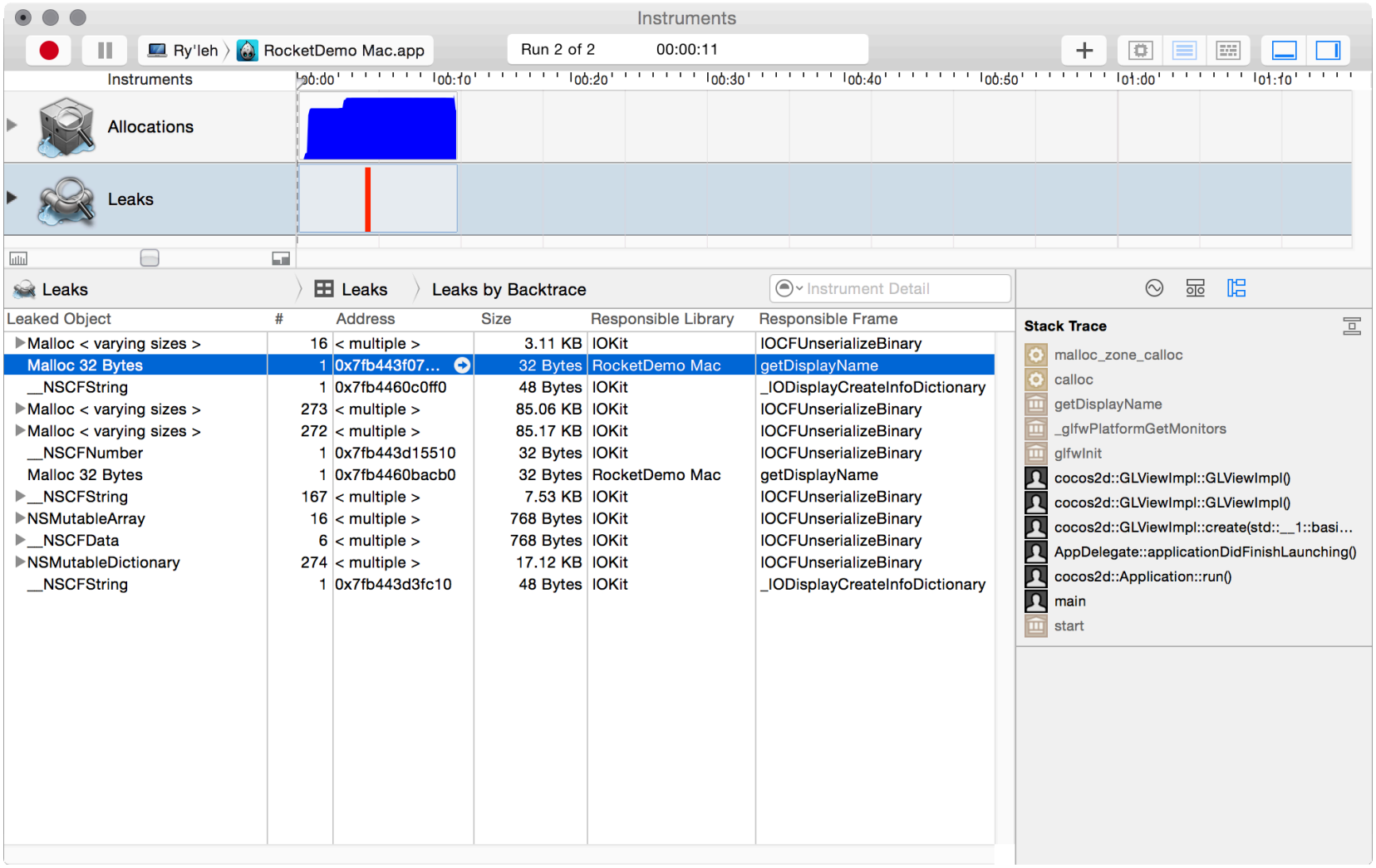
Instrumentation: Memory

- Memory handled by malloc
  - Basic C allocation method
  - C++ `new` uses malloc
  - Allocates raw bytes
- malloc can be **instrumented**
  - Count number of mallocs
  - Track malloc addresses
  - Look for frees later on
- Finds memory leaks!



the gamedesigninitiative at cornell university

Instrumentation: Memory



Profiling Tools

- **iOS/X-Code**
  - Instruments (wide variety of special tools)
  - GNU gprof (Profile ⌘I) for sampled time
- **Android (Java)**
  - Dalvik Debug Monitor Server (DDMS) for traces
  - TraceView helps visualize the results of DDMS
- **Android (C++)**
  - Android NDK Profiler (3rd party)
  - GNU gprof visualizes the results of `gmon.out`

the gamedesigninitiative at cornell university

Android NDK Profiling

```
// Non-profiled code
monstartup("your_lib.so");
// Profiled code
moncleanup();
// Non-profiled code
```

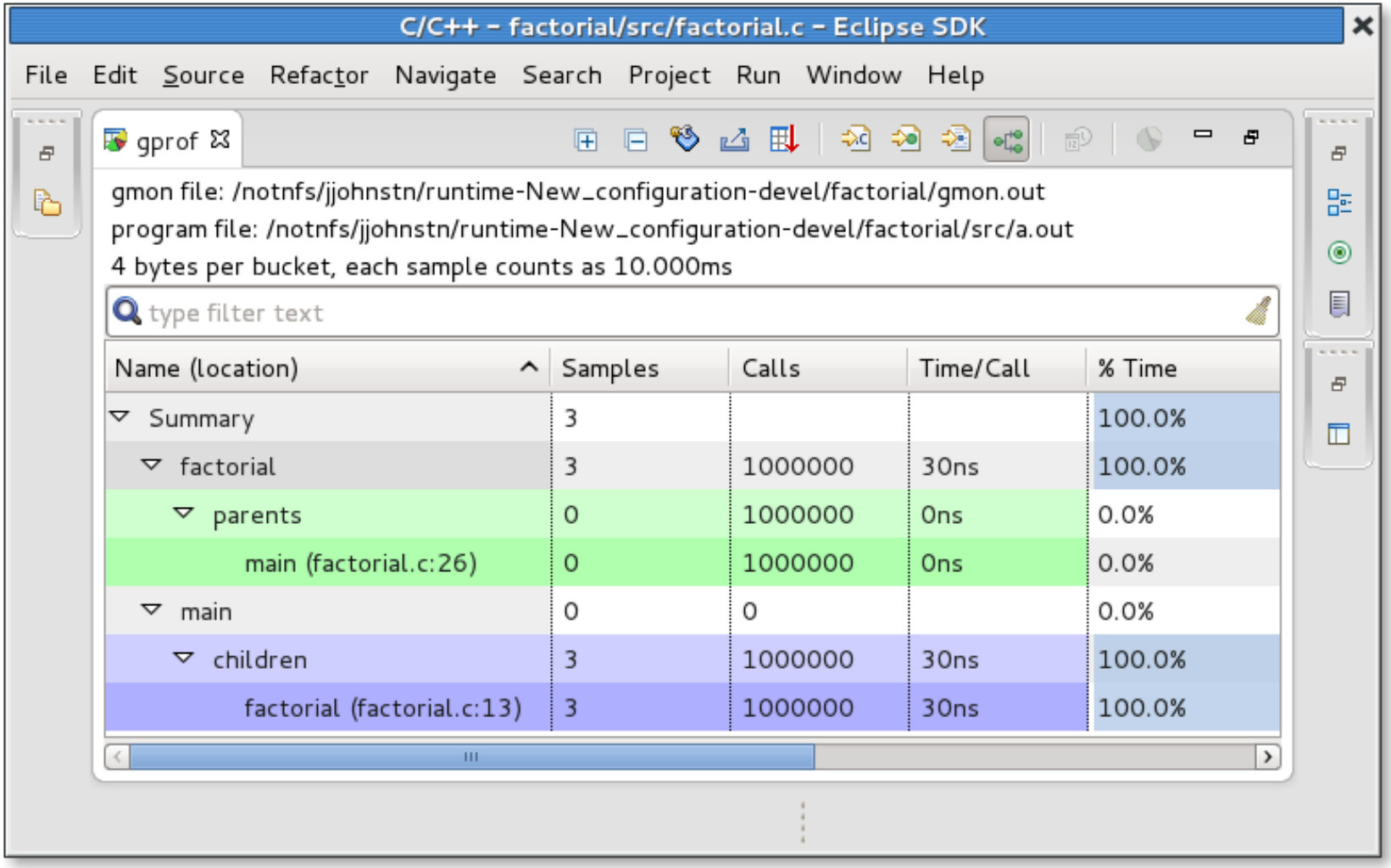
captures everything

Android App

gmon.out

gprof

Android Profiling



the gamedesigninitiative at cornell university

Poor Man's Sampling

Call Graph

- Create a hashtable
  - Keys = pairs (*a* calls *b*)
  - Values = time (time spent)
- Place code around call
  - Code inside outer func. *a*
  - Code before & after call *b*
  - Records start and end time
  - Put difference in hashtable

Timing

- Use the processor's timer
  - Track time used by program
  - System dependent function
  - **Java**: `System.nanoTime()`
- Do not use "wall clock"
  - Timer for the whole system
  - Includes other programs
  - **Java** version: `System.currentTimeMillis()`

the gamedesigninitiative at cornell university

Poor Man's Sampling

Call Graph

- Create a hashtable
  - Keys = pairs (*a* calls *b*)
  - Values = time (time spent)
- Place co
  - Code
  - Code
  - Records start and end time
  - Put difference in hashtable

call *b*

Timing

- Use the processor's timer
  - Track time
  - by program
  - function
  - Time()
  - "wall clock"
  - Timer for the whole system
  - Includes other programs
  - **Java** version: `System.currentTimeMillis()`

Useful in networked setting

Summary

- Premature optimization is bad
  - Make code unmanageable for little gain
  - Best to identify the bottlenecks first
- Static analysis is useful in some cases
  - Finding memory leaks and other issues
  - Deadlock and resource analysis
- Profiling can find runtime performance issues
  - But changes the program and incurs overhead
  - Sampling and instrumentation reduce overhead

the gamedesigninitiative at cornell university