The New Call Graph Based Performance Consultant

Trey Cain

cain@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685
USA

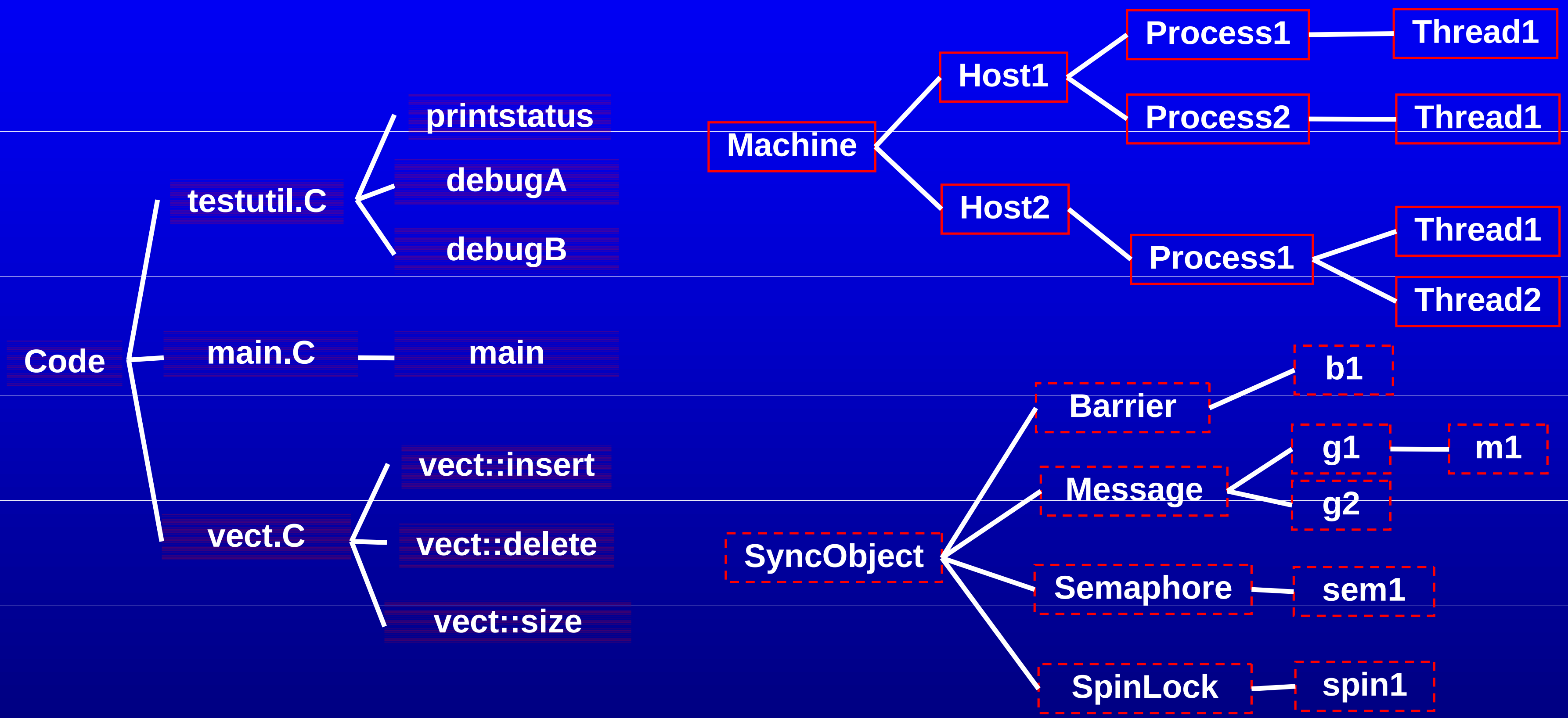
The Performance Consultant (PC)

- Uses two main Paradyn technologies
  - Dynamic instrumentation
  - Automated bottleneck search
- Original version had difficulty searching large applications
- Our solution: direct PC search using application call graph

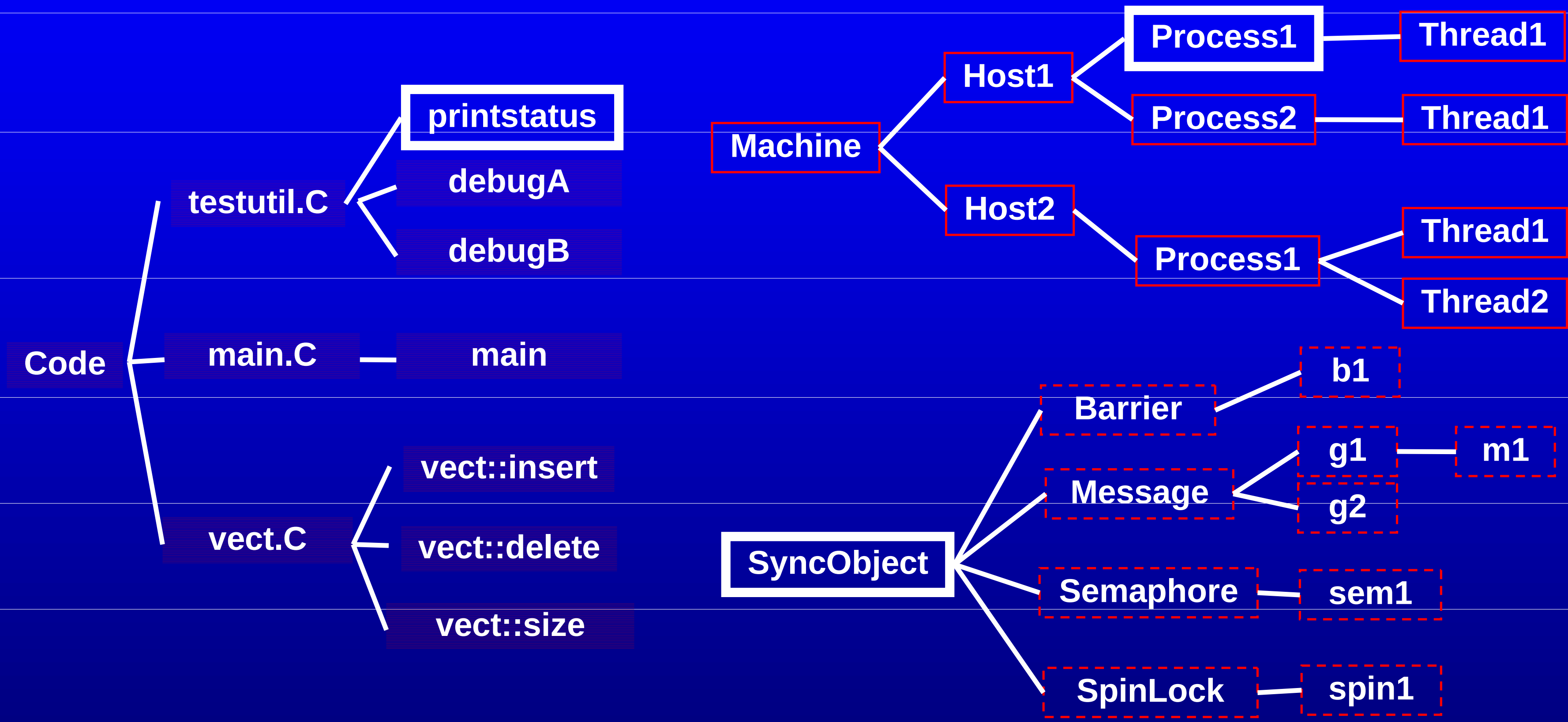
Outline

- Paradyn Basics
- Original Search Strategy
- Call Graph Based Search Strategy
- Dynamic Call Site Instrumentation
- Performance Comparison
- Conclusion

Paradyn Basics: Resource Hierarchies



Paradyn Basics: Resource Hierarchies



Example Focus: {/Code/testutil.C/printstatus, /Machine/host1/process1, /SyncObject }

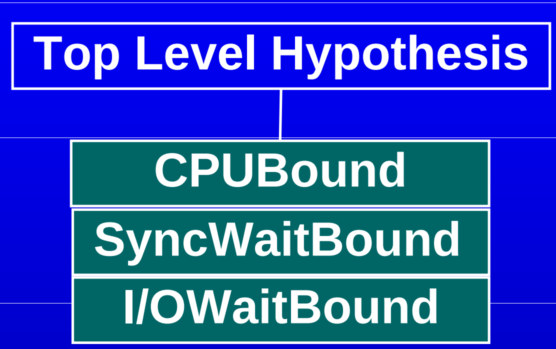
Paradyn Basics: Performance Metrics

- Metrics are measurable performance characteristics such as CPU time, function calls, I/O bytes transferred
- Performance data collected for metric/focus pair
- Example metric/focus pairs:
  - cpu:{/Code/mod1/func1 }
  - msgs:{/Code/mod1/func1, /Machine/host1/proc4/thread2, /SyncObject/Message/1/0}

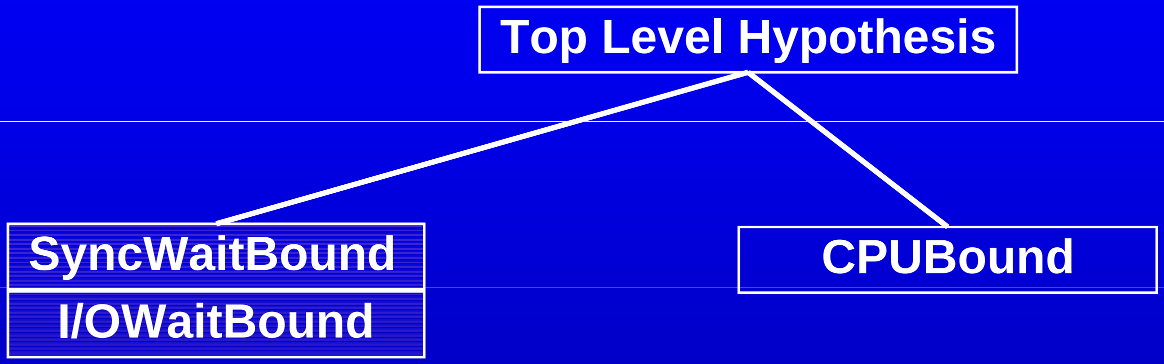
Performance Consultant Basics

- Why is the application running slowly?
  - Test bottleneck hypotheses
    - CPU Bound?
    - I/O Wait Bound?
    - Synchronization Wait Bound?
    - Memory Bound?
  - Performance metric associated with each hypothesis
- Which part of the application is slow?
  - Isolates bottleneck to part of resource hierarchy

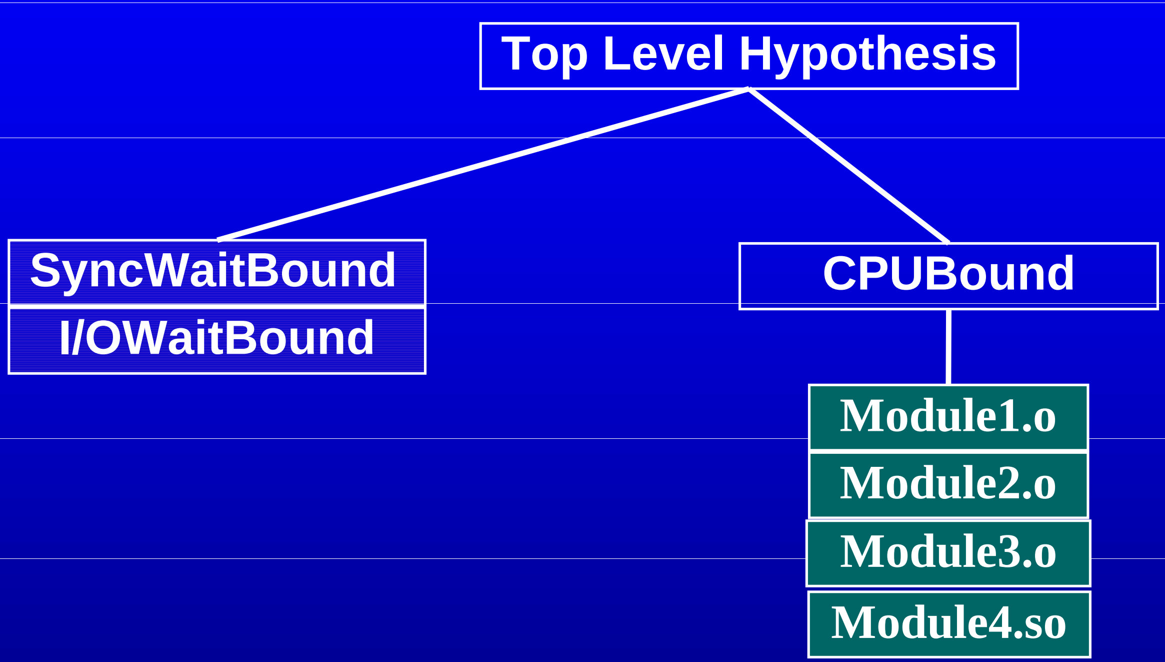
Original PC Example



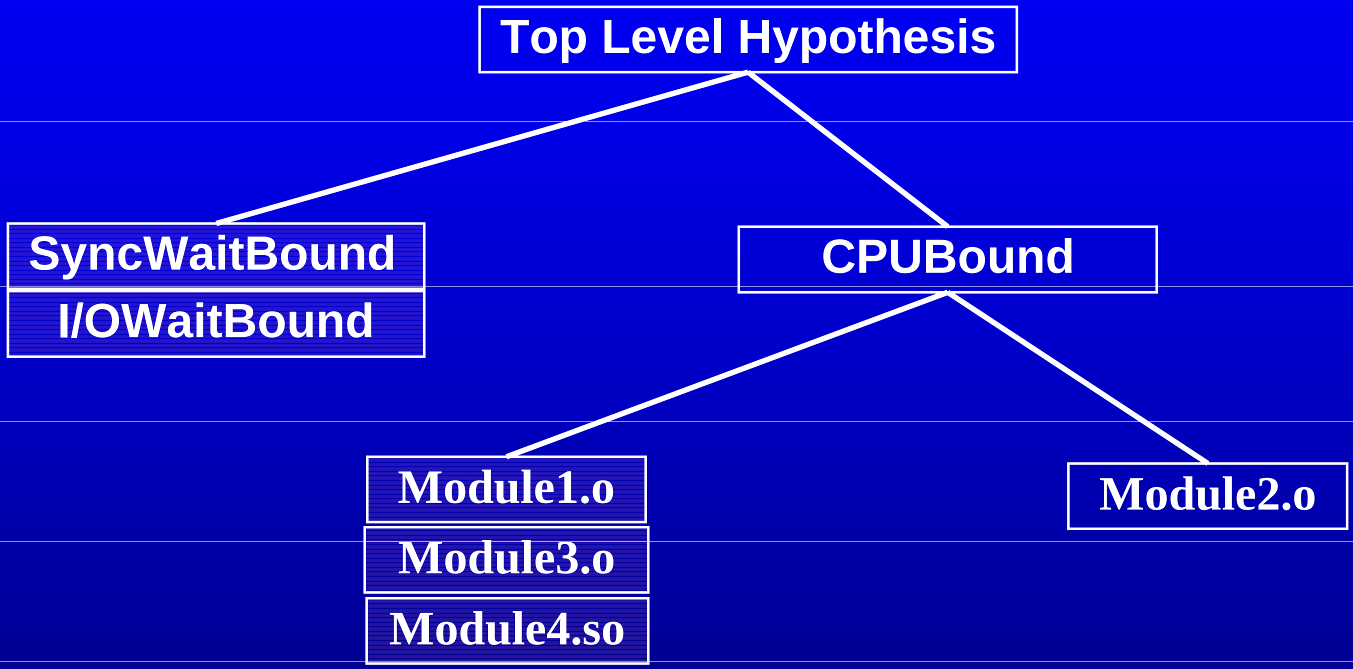
Original PC Example



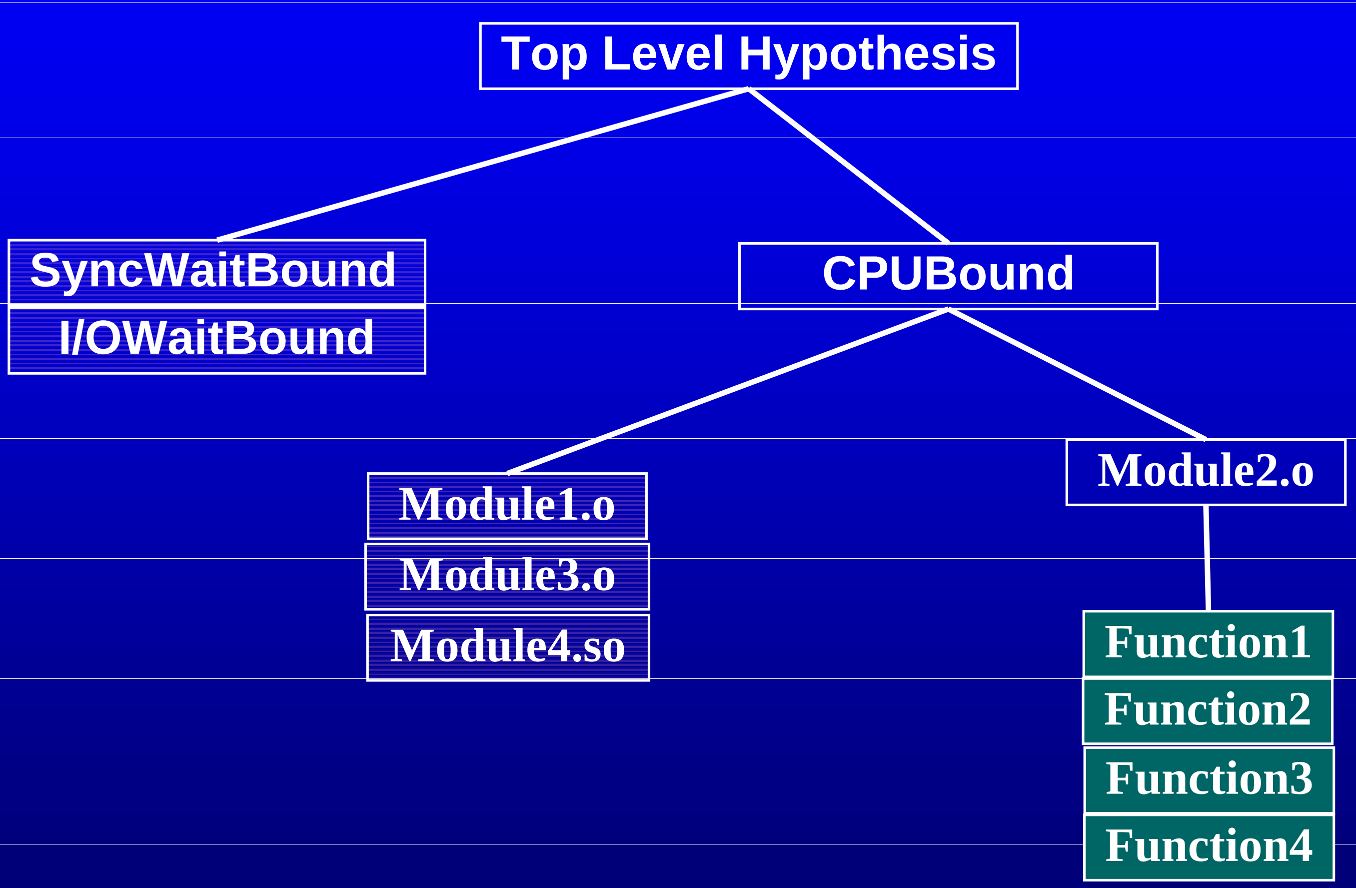
Original PC Example



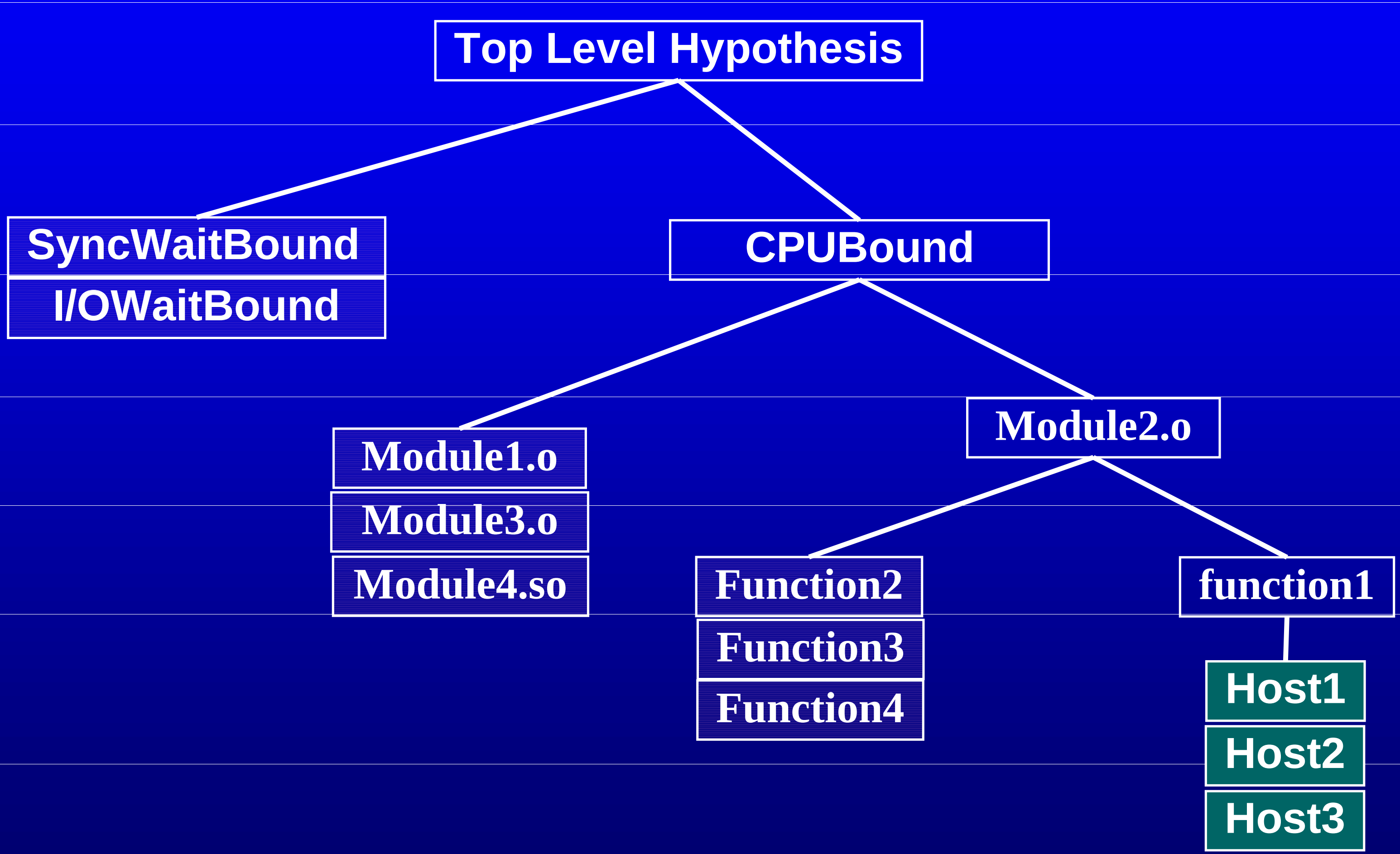
Original PC Example



Original PC Example



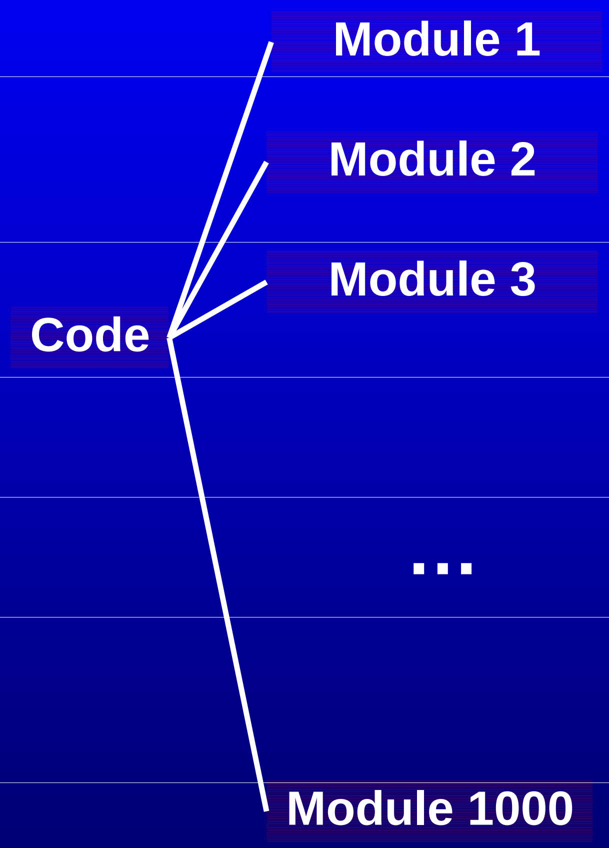
Original PC Example



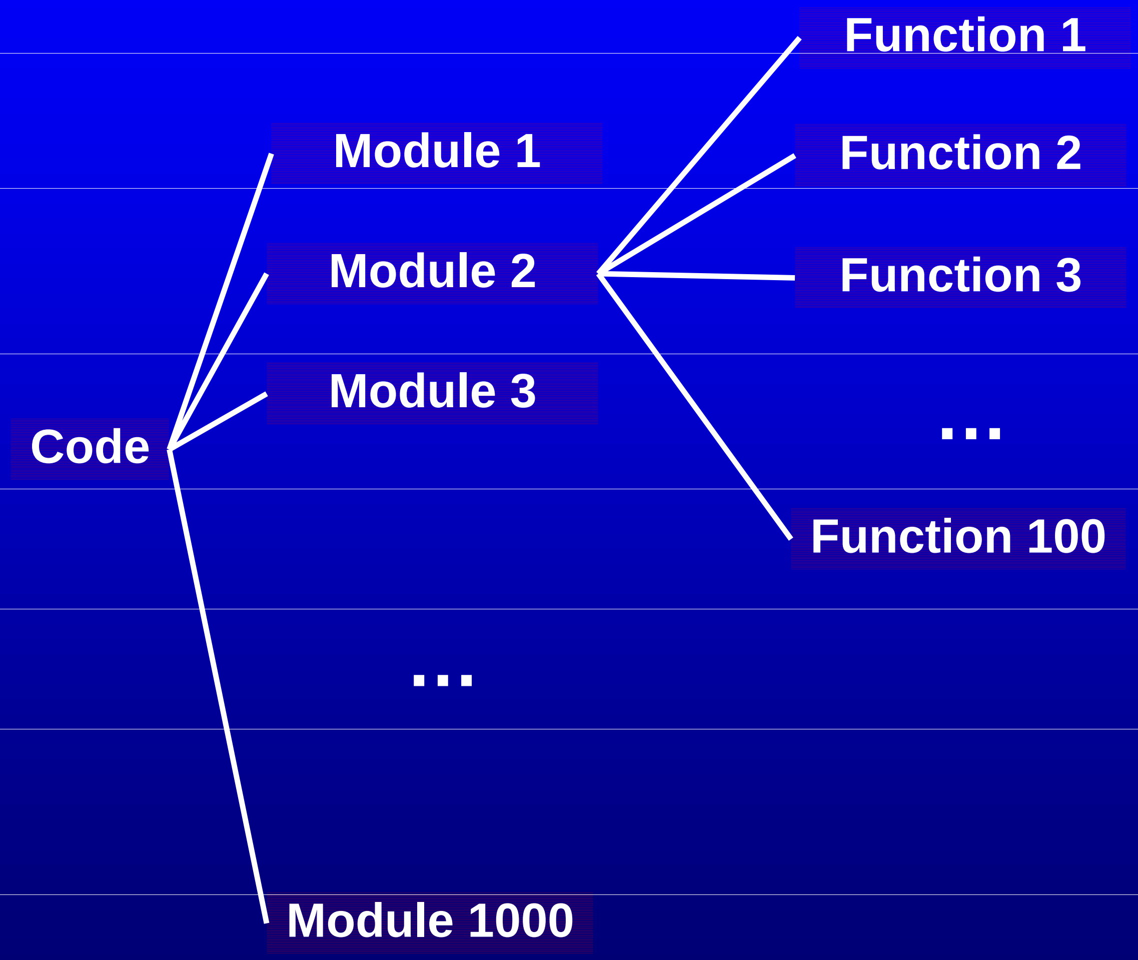
Original Performance Consultant

- Problem: Traversing the code hierarchy does not scale
  - Search space too large: too many modules, too many functions
  - Module instrumentation is not cheaper than instrumenting all of module's functions
  - Exclusive metrics are costly
- We would like to avoid excessive instrumentation

Too many modules and functions



Too many modules and functions



PC Timing Metrics

- Performance Consultant based on the idea that coarse grained instrumentation is cheaper than fine grained…
- But instrumenting a module has the same cost as instrumenting each function in the module individually.

Exclusive vs. Inclusive Metrics

```
foo() {

    bar();

     car();

}
```

Exclusive Timer

Exclusive vs. Inclusive Metrics

```
startTimer(t) ---->  foo() {

stopTimer(t)  ---->
                        bar();
startTimer(t) ---->

stopTimer(t)  ---->
                        car();
startTimer(t) ---->

stopTimer(t)  ---->
                     }
```

Exclusive Timer

Exclusive vs. Inclusive Metrics

```
startTimer(t) →  foo() {
stopTimer(t)  →
startTimer(t) →      bar();
stopTimer(t)  →
startTimer(t) →      car();
stopTimer(t)  →  }
```

Exclusive Timer

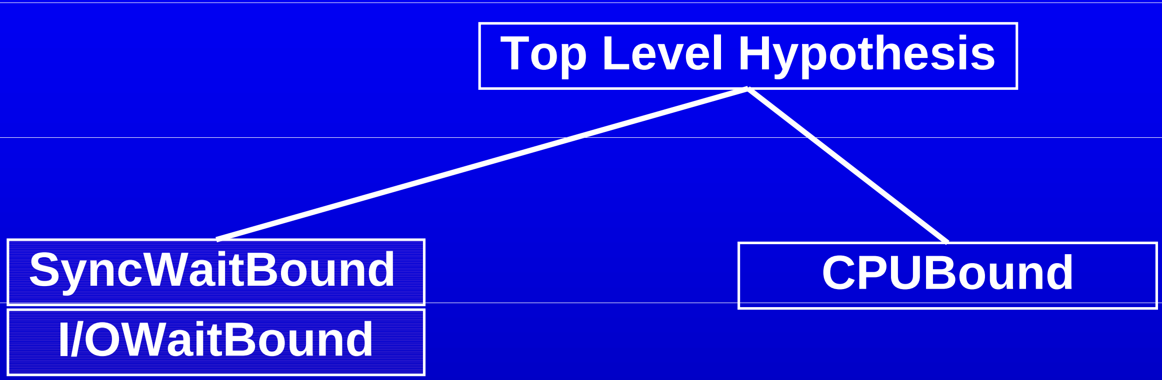
```
startTimer(t) →  foo() {
                     bar();
                     car();
stopTimer(t)  →  }
```

Inclusive Timer

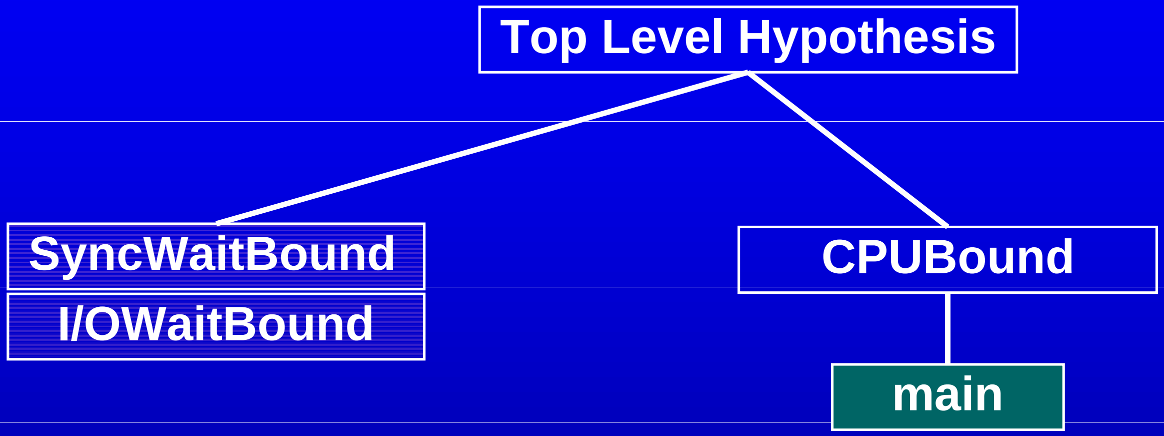
Call Graph Based Performance Consultant

- Based on application's call graph
- Code hierarchy search starts at function `main`, search continues to `main`'s children
- Advantages: Lots!
  - It's Scalable: Natural hierarchical refinement from course grained search to fine grained search
  - Uses less costly inclusive metrics
  - Functions which are not part of call graph will never be instrumented

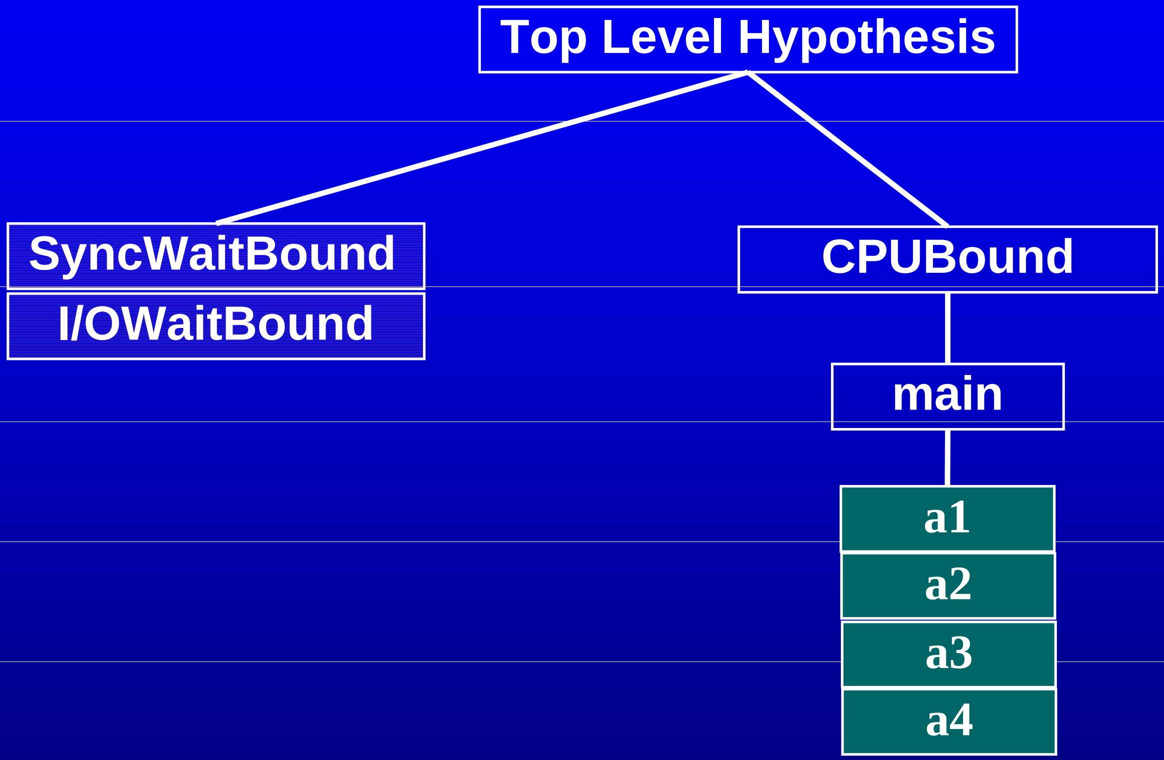
Call Graph Based PC Example



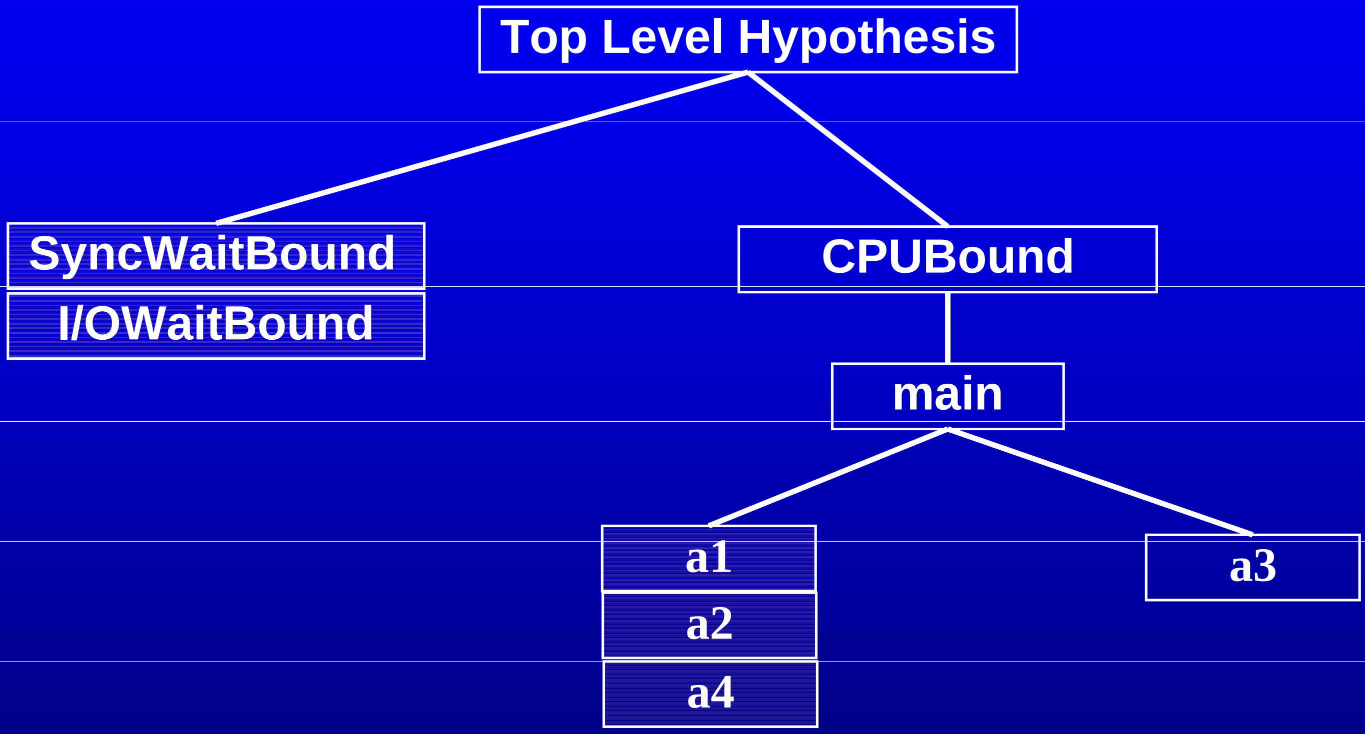
Call Graph Based PC Example



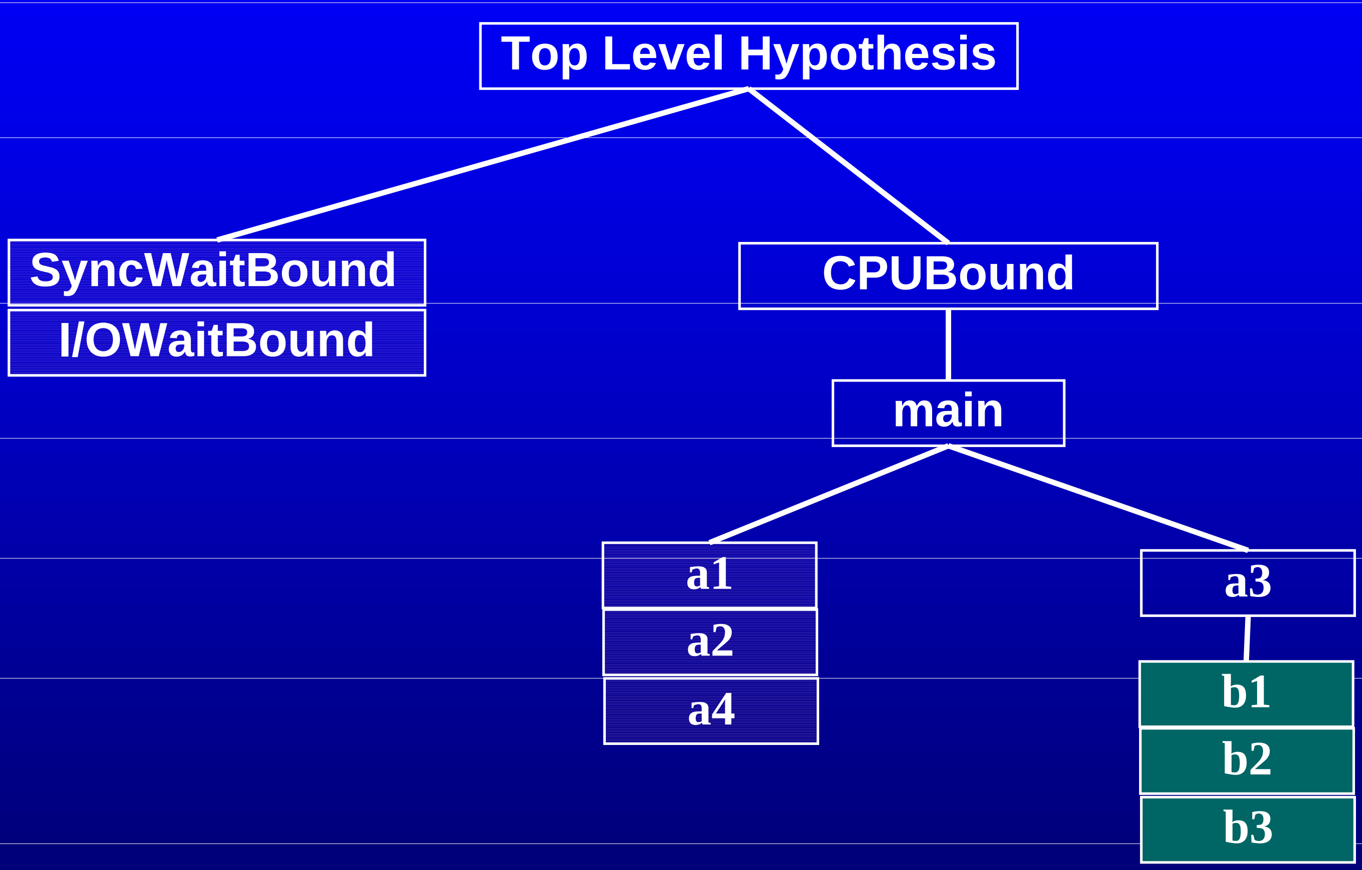
Call Graph Based PC Example



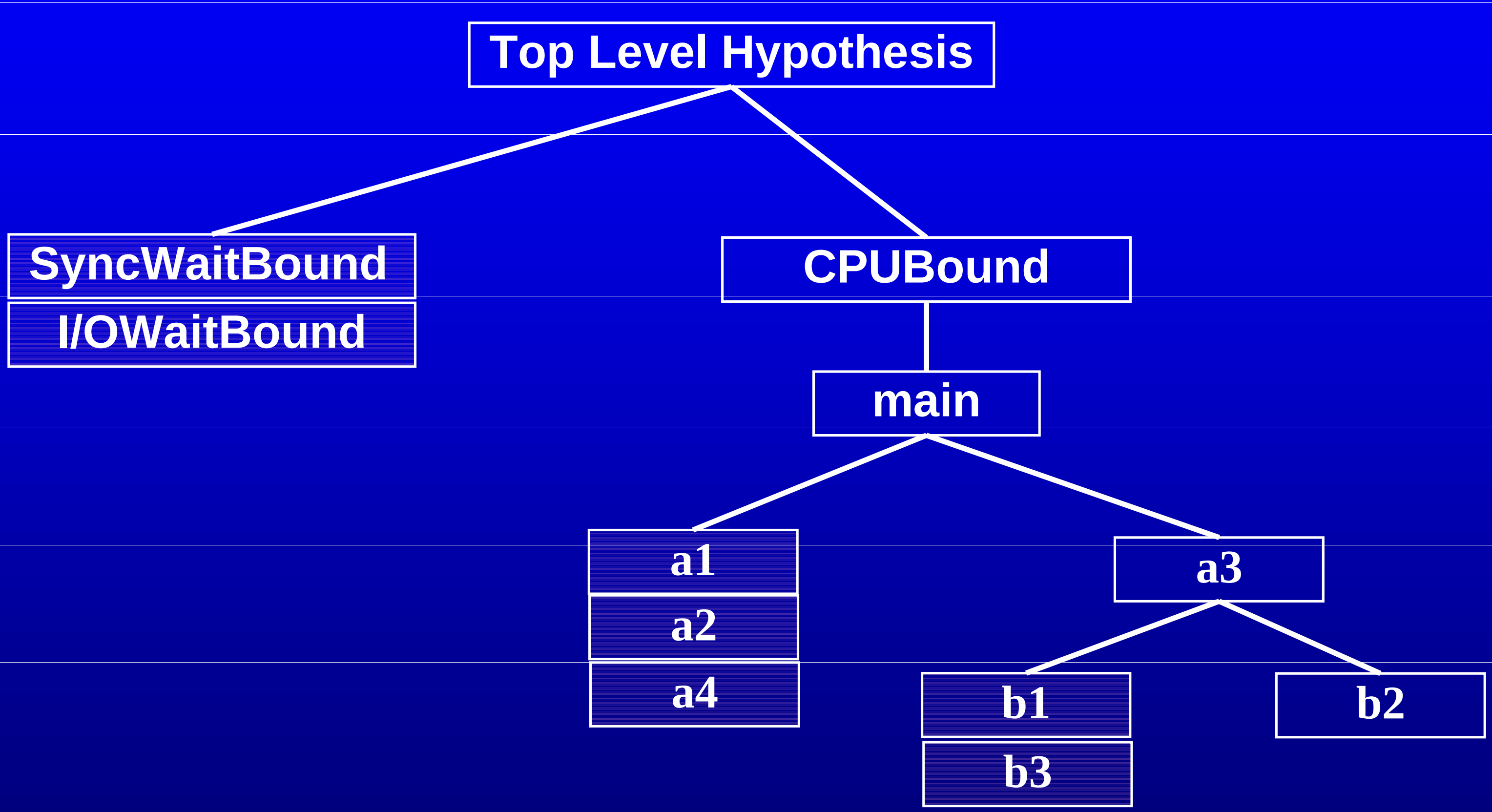
Call Graph Based PC Example



Call Graph Based PC Example



Call Graph Based PC Example



Call Graph Construction

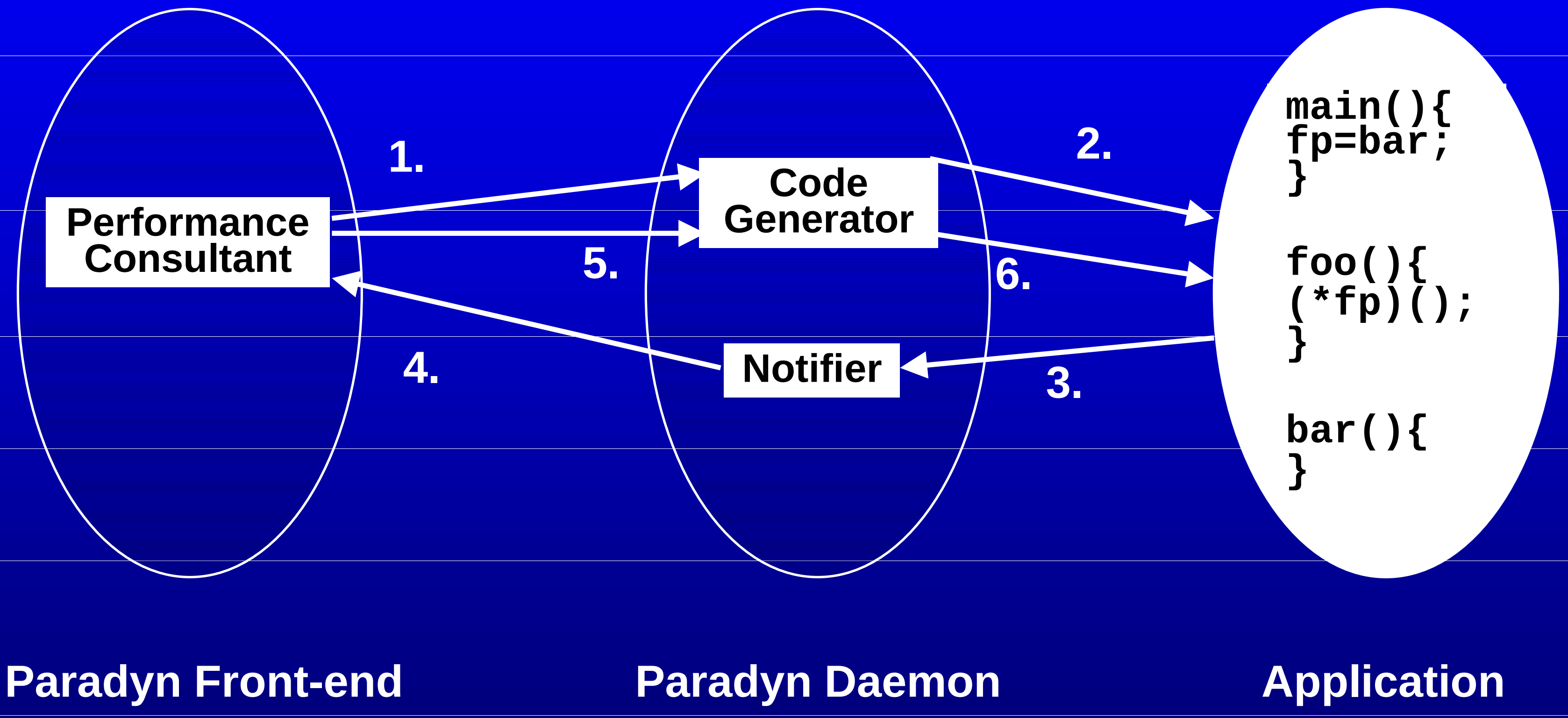
- Problem: targets of calls using function pointers and virtual functions are not statically determinable.
- Unknown callees in static call graph may cause blind spots in new PC search
- We resolve dynamic callee addresses at run time
- Strategy:
  - Build static call graph at program start
  - Fill in dynamic call graph on demand.

Dynamic Call Sites

- Characterized by keeping the address of a callee in a register or memory location
- New type of instrumentation necessary to determine callee
- Examples:

Instruction Set	Call Instruction
MIPS	`jalr $t9`
X86	`call [%edi]`

Call Site Instrumentation: Chain of Events



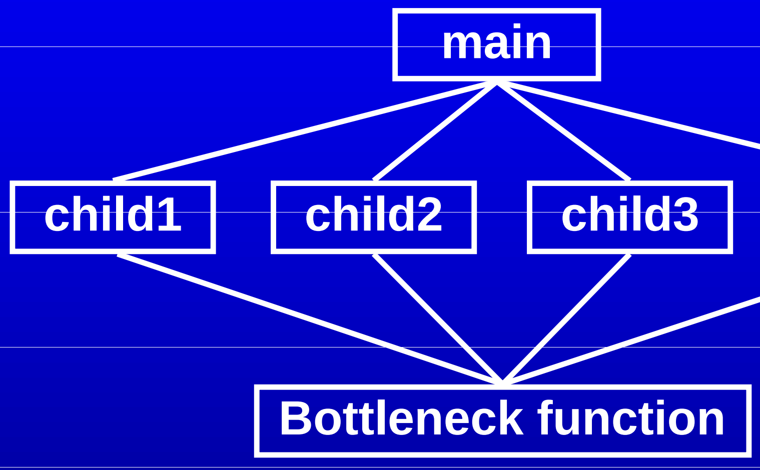
Performance Results

	Bottlenecks found in complete search		Instrumentation Mini-tramps Used		Required Search Time (seconds)	
Application	Original	Call Graph	Original	Call Graph	Original	Call Graph
Draco	3	5	14,317	228	1,006	322
go	2	4	12,570	284	755	278
Fpppp	3	3	474	96	141	186
ssTwod (MPI)	9	9	43,230	11,496	461	316
OM3 (MPI)	13	16	184,382	60,670	2,515	957

Conclusion

- Call graph based search strategy perturbs application less than old search
- New search also faster than old search
- New version of PC available in Paradyn 3.0
- Room for future work…
  - Exclusive bottleneck verification
  - Finding a way to avoid potential blind spots.

Potential Blind Spot for New PC



A rare scenario: we haven't seen it happen yet.

Retroactive Instrumentation

- Problem: Find CPU Time for a function if we are executing in one of its children.
- When do we start the timer for the entry to function?
- Need mechanism to trigger instrumentation code.
- Retroactive instrumentation walks stack, triggering outstanding timers