GENERIC OPTIMIZATION DATA ANALYZER

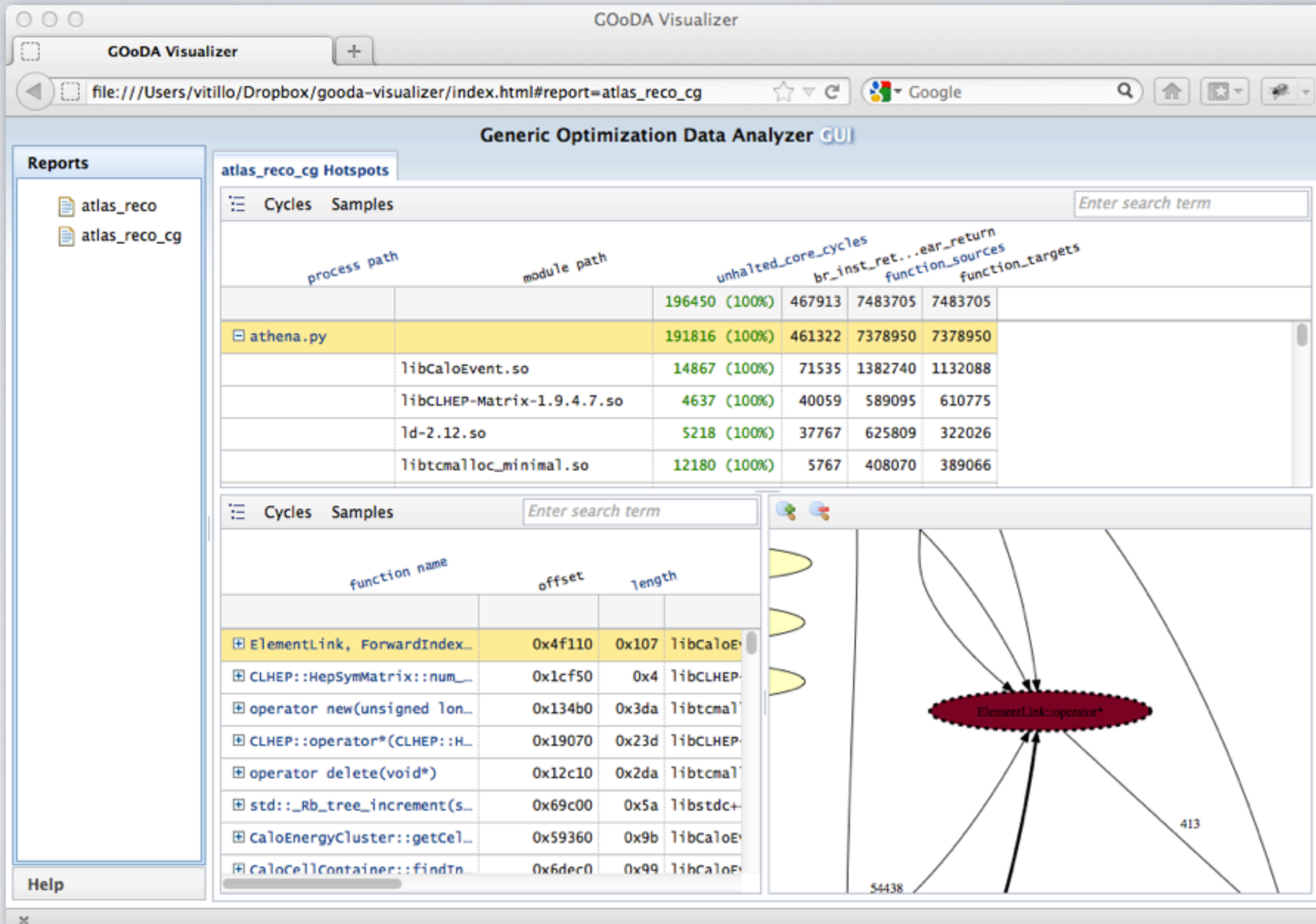
P. Calafiura[1], S. Eranian[2], D. Levinthal[2], S. Kama[3], R. A. Vitillo[1]

CHEP 2012, New York, May 21-25

1. Lawrence Berkeley National Laboratory
2. Google
3. Southern Methodist University

WHAT IS GOODA?

- **Low overhead** open source Performance Monitoring Unit (PMU) event data analysis package
  - A CPU profiler
- Developed in collaboration between Google and LBNL
- Logically composed of four main components:
  - A kernel subsystem that provides an interface to the PMU
  - An event data collection tool
  - An analyzer creates call graphs, control flow graphs and spreadsheets for a variety of granularities (process, module, function, source etc.)
  - A web based GUI displays the data



GOoDA Visualizer
file:///Users/vitillo/Dropbox/gooda-visualizer/index.html#report=atlas_reco_cg
Google
Generic Optimization Data Analyzer GUI
Reports
atlas_reco
atlas_reco_cg
Help
atlas_reco_cg Hotspots
Cycles
Samples
Enter search term
process path
module path
unhalted_core_cycles
br_inst_ret...ear_return
function_sources
function_targets
196450 (100%) 467913 7483705 7483705
athena.py 191816 (100%) 461322 7378950 7378950
libCaloEvent.so 14867 (100%) 71535 1382740 1132088
libCLHEP-Matrix-1.9.4.7.so 4637 (100%) 40059 589095 610775
ld-2.12.so 5218 (100%) 37767 625809 322026
libtcmalloc_minimal.so 12180 (100%) 5767 408070 389066
function name
offset
length
ElementLink, ForwardIndex... 0x4f110 0x107
CLHEP::HepSymMatrix::num_... 0x1cf50 0x4
operator new(unsigned lon... 0x134b0 0x3da
CLHEP::operator*(CLHEP::H... 0x19070 0x23d
operator delete(void*) 0x12c10 0x2da
std::_Rb_tree_increment(s... 0x69c00 0x5a
CaloEnergyCluster::getCel... 0x59360 0x9b
413
54438

MOTIVATION

- What we were looking for:
  - Low overhead profiling
  - Call counts statistics
  - Microarchitectural insights
  - User friendly GUI
  - Open Source

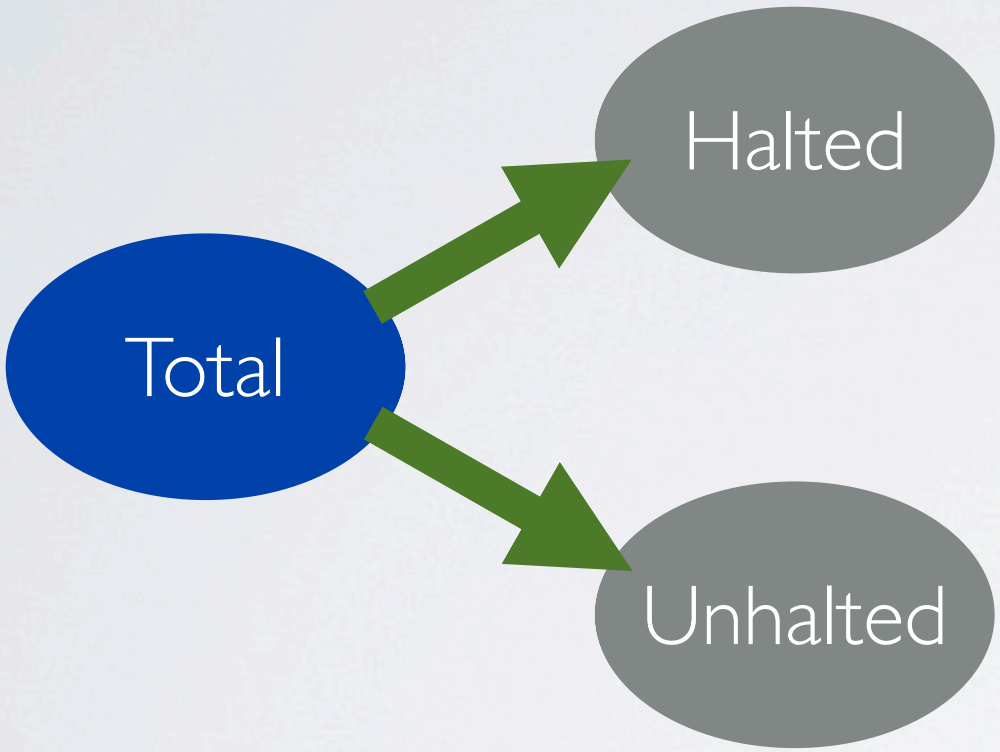
CODE OPTIMIZATION

- Code optimization is minimizing CPU cycles
  - nothing else matters
- Decisions of what code to work on must be based on reasonably accurate estimates of what can be gained... **in cycles!**
- Cycles can be grouped into architecture independent groups
  - forms an hierarchical tree

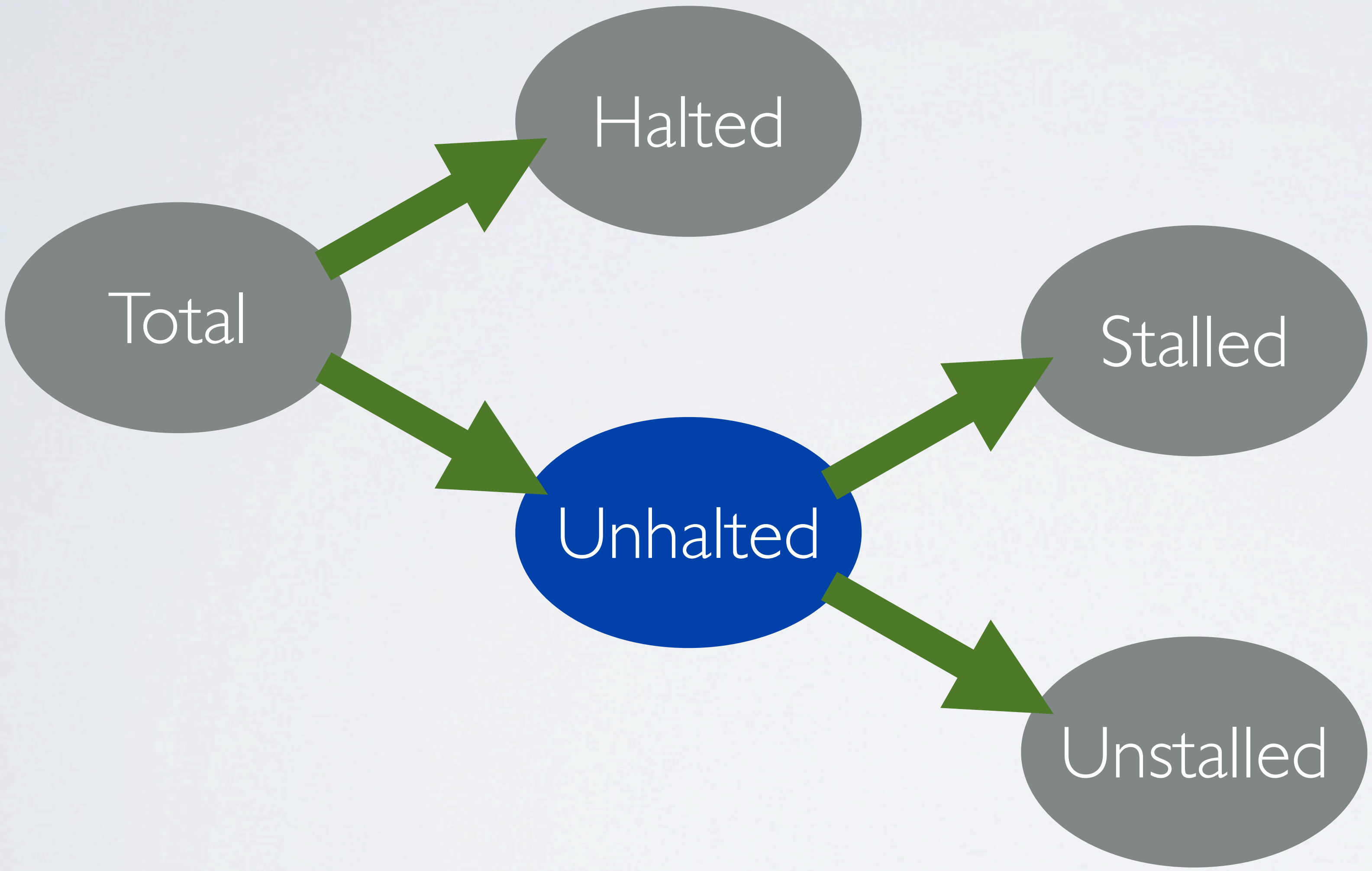
CYCLE ACCOUNTING

Total

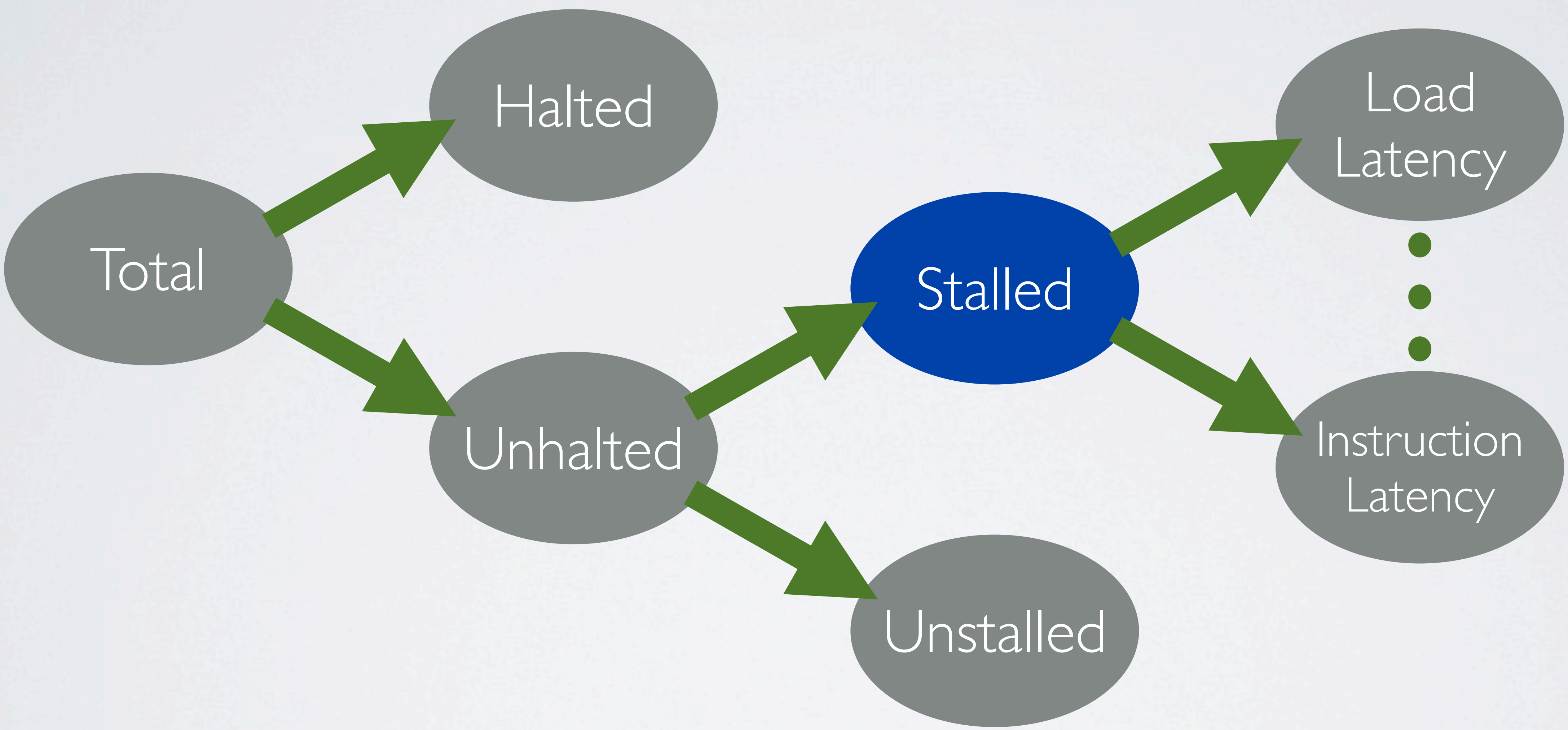
CYCLE ACCOUNTING



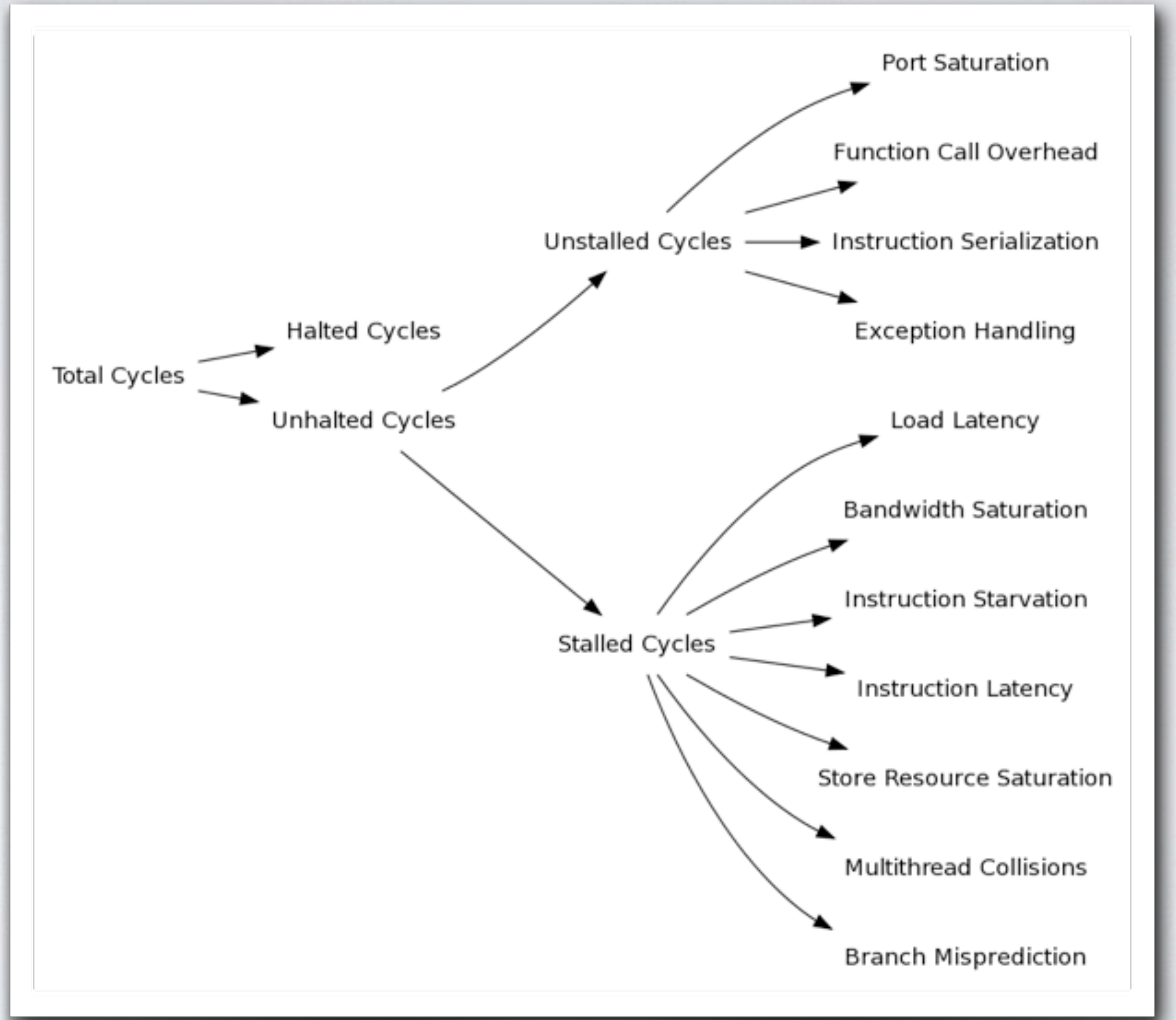
CYCLE ACCOUNTING



CYCLE ACCOUNTING



CYCLE ACCOUNTING



HARDWARE EVENT COLLECTION

- Modern CPU's include a Performance Monitoring Unit (PMU)
- Provides the ability to count the occurrence of micro-architectural events, e.g.:
  - Executed instructions
  - Cache misses
- Events expose inner workings of the processor as it executes code
  - hundreds of events per architecture
  - **caveat**: events do not map consistently between different architectures

HARDWARE EVENT COLLECTION

- PMU interrupt mode: profile where events occur vs assembly and source
  - Initialize counters to the sampling period
  - An interrupt is triggered when counter is zero
  - Capture IP, PID, TID, LBR, CPU and other data on interrupt
- How do we convert event samples to cycles?

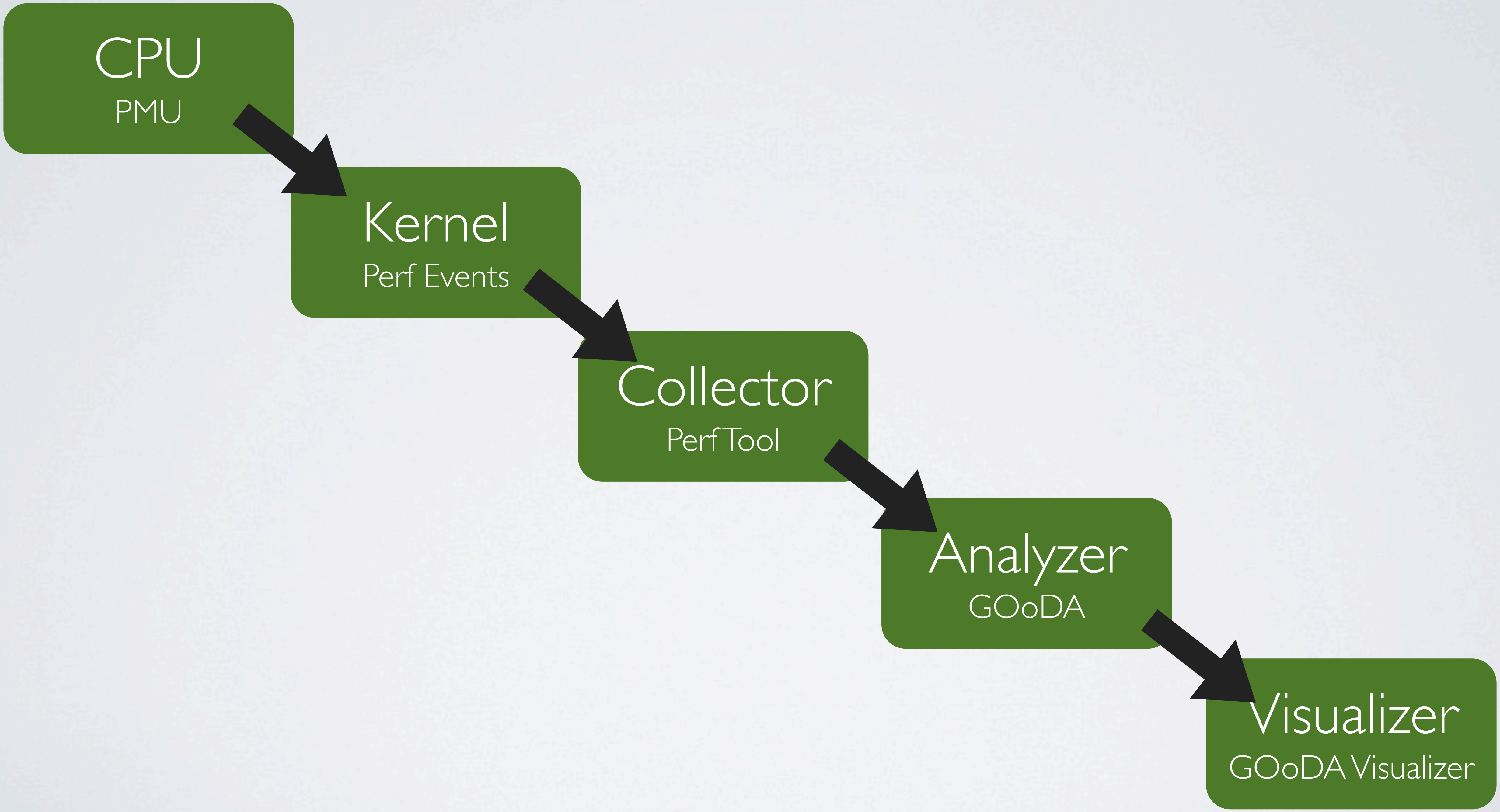
CYCLE DECOMPOSITION

- Stalled/unstalled cycles are decomposed as a sum of count(event) * cost(event)
  - the cost is the penalty paid in cycles for a specific event
- Example: Load Latency:
  - Use exclusive hit events
  - Includes load accesses to caches and memory, load DTLB costs and blocked store forwarding... **lots of events!**
  - Latency depends on specific configuration that needs to be determined with micro benchmarks

CYCLE DECOMPOSITION

- Load Latency on Westmere
  - 6 * mem_load_retired:l2_hit +
  - 52 * mem_load_retired:l3_unshared_hit +
  - 85 * (mem_load_retired:other_core_l2_hit_hitm - mem_uncore_retired:local_hitm) +
  - 95 * mem_uncore_retired:local_hitm +
  - 250 * mem_uncore_retired:local_dram_and_remote_cache_hit +
  - 450 * mem_uncore_retired:remote_dram +
  - 250 * mem_uncore_retired:other_llc_miss +
  - 7 * (dtlb_load_misses:stlb_hit + dtlb_load_misses:walk_completed) + dtlb_load_misses:walk_cycles +
  - 8 * load_block_overlap_store
- Tools needs to know methodology so users don't!

HOW GOODA WORKS



PERF EVENTS

- Performance monitoring interface introduced in the kernel in 2009
- Unified interface to access hardware performance counters, kernel software counters and tracepoints
- System call interface that exposes an high level abstraction known as event
- Events are manipulated via file descriptor obtained through the perf_event_open system call
- Samples are saved into a kernel buffer which is made visible to tools via the mmap system call

PERF TOOL

- User space tool which allows counting and sampling of events
- Many events can be sampled at the same time
- Used by the GOoDA collection scripts to collect samples into a data file

ANALYZER

- Reads and parses a perf data file
- Implements the cycle accounting methodology
  - depends on the underlying architecture!
- Generates spreadsheets for:
  - hot processes and functions
  - source and assembly for the N hottest functions
- Generates SVG's of the Call Graph and the Control Flow Graph

VISUALIZER

- HTML5, CSS3 & Javascript based GUI
- Reads, parses and displays the spreadsheets generated by the Analyzer
- Can be deployed on a webserver or on a client machine
- A modern browser is the only dependency

IN ACTION: HOT PROCESSES

process path	module path	unhalted_core_cycles	uop
		473185 (100%)	266508
athena.py		463031 (100%)	246143
vmlinux		9006 (100%)	19529 (
gnome-settings-		328 (100%)	156
irqbalance		253 (100%)	142
khugepaged		164 (100%)	142
perf		134 (100%)	85
flush-253:0			14
ksoftirqd/3		45 (100%)	

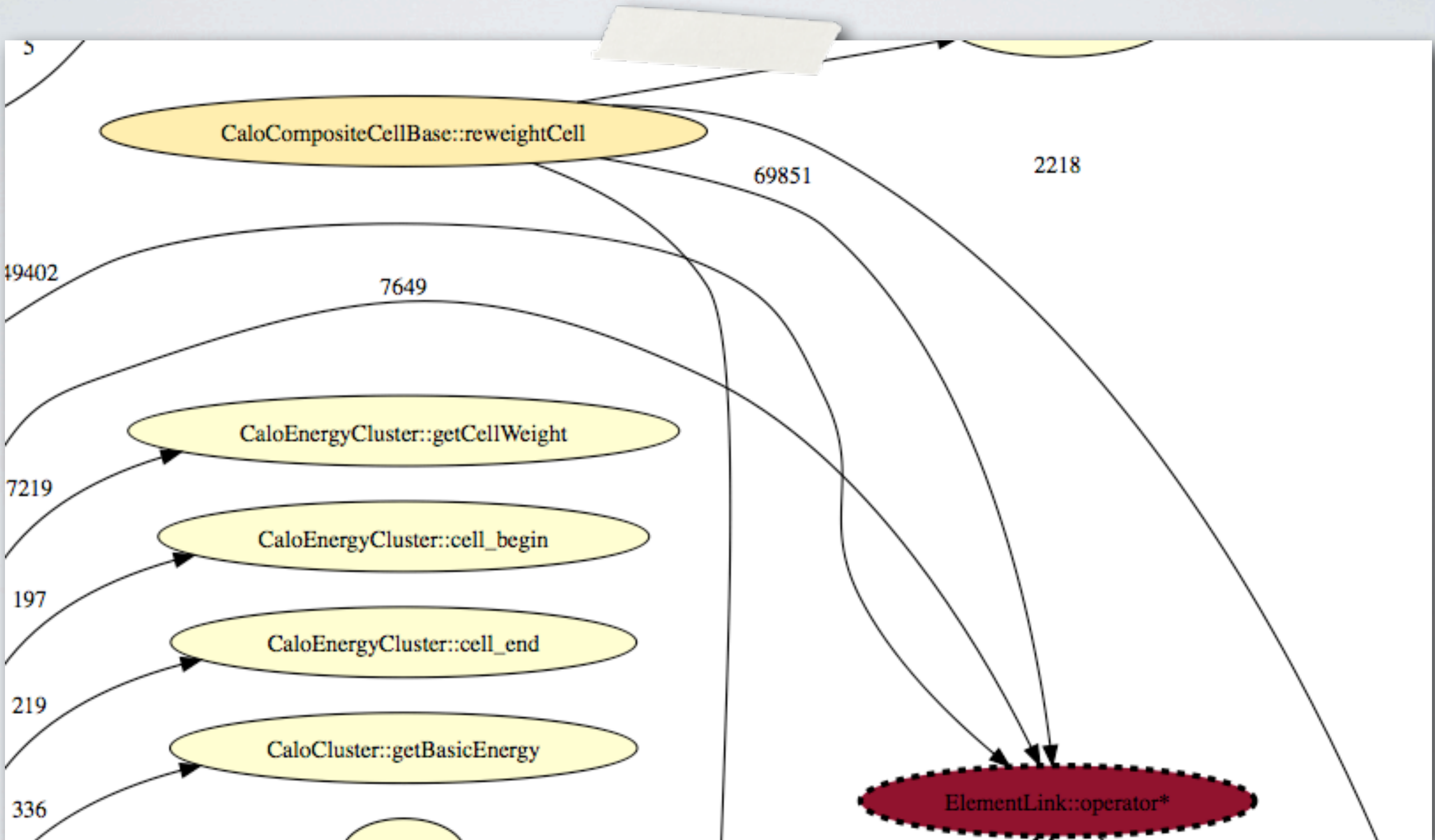
Processes ordered by hotness

IN ACTION: HOT MODULES

process path	module path	unhalted_core_cycles		uop
		473185	(100%)	266508
⊟ athena.py		463031	(100%)	246143
	libCaloEvent.so	28434	(100%)	15320
	libtcmalloc_minimal.so	28897	(100%)	14966
	libm-2.12.so	41526	(100%)	22066
	libBFieldStand.so	28792	(100%)	15122
	libstdc++.so.6.0.10	22440	(100%)	14030
	libCLHEP-Matrix-1.9.4.7.so	12644	(100%)	5017
	ld-2.12.so	11451	(100%)	4478
	libTrkAlgebraUtils.so	13464	(100%)	5456

Modules ordered by hotness

IN ACTION: CALLGRAPH



No instrumentation required

IN ACTION: HOT FUNCTIONS

function name	offset	length	module	process	unhalted_core_cycles		uop
					473185	(100%)	266508
⊞ operator new(unsigned lon…	0x134b0	0x3da	libtcmalloc_minimal.so	athena.py	12927	(100%)	5442
⊞ master.0.gbmagz_	0xfb80	0x4a0b	libBFieldStand.so	athena.py	13882	(100%)	5995
⊞ operator delete(void*)	0x12c10	0x2da	libtcmalloc_minimal.so	athena.py	7619	(100%)	3741
⊞ std::_Rb_tree_increment(s…	0x69c00	0x5a	libstdc++.so.6.0.10	athena.py	8633	(100%)	5697
⊞ get_bsfield_	0xed60	0xe16	libBFieldStand.so	athena.py	11407	(100%)	7809
⊞ Trk::STEP_Propagator::pro…	0x2b230	0x18e2	libTrkExSTEP_Propagator.so	athena.py	6337	(100%)	2792
⊞ Trk::RungeKuttaPropagator…	0x250e0	0x1051	libTrkExRungeKuttaPropagato…	athena.py	7589	(100%)	4478
⊞ ma27od_	0x22000	0x26ee	libTrkAlgebraUtils.so	athena.py	6397	(100%)	2083
⊞ Trk::FitMatrices::solveEq…	0x108a0	0x49a	libTrkiPatFitterUtils.so	athena.py	4935	(100%)	1701
⊞ deflate_slow	0x6850	0x976	libz.so.1.2.3	athena.py	5189	(100%)	2395

Dive into assembly and source code…

IN ACTION: SOURCE

line number	source	unhalted_core_cycles		uops_	
		6337	(100%)	2792	(44
1050	`numSf++;`	45	(100%)	14	(31
1051	`} else {`				
1052	`// save the nearest distance to surface`				
1053	`m_currentDist.push_back( std::pair<int,std::pair<double,double> >(-1…`	641	(100%)	184	(28
1054	`}`				
1055	`}`				
1056					
1057	`if (distanceToTarget == maxPath || numSf == 0 ) {`				
1058	`//std::cout << "propagateWithJacobian: initial distance estimate faile…`				
1059	`if( m_currentDist.capacity() > m_maxCurrentDist ) m_currentDist.reserv…`				

Pinpoint hot source lines

IN ACTION: ASSEMBLY

address	princ_l#	disassembly		unhalted_core_cycles		uops_	
				6397	(100%)	2083	(32
0x23db4	1643	mov	%esi,0xe67e(%rip)	15	(100%)		
0x23dba	1645	jl	23d48				
0x23dbc	1645	⊟ Basic Block 262 <0x23dc0>					
0x23dbc	1645	nopl	0x0(%rax)				
0x23dc0	1645	⊟ Basic Block 263 <0x23dc0><0x23e04>		5099	(100%)	1616	(31
0x23dc0	1646	mov	0xe64e(%rip),%ecx	45	(100%)		
0x23dc6	1646	mov	%ecx,%eax	119	(100%)	57	(47
0x23dc8	1647	movslq	%ecx,%rdx	567	(100%)	113	(19
0x23dcb	1646	sub	%edi,%eax				
0x23dcd	1647	sub	$0x1,%rdx	30	(100%)	14	(46
0x23dd1	1645	cmp	%ecx,%esi	15	(100%)	14	(93

Pinpoint hot basic blocks

CYCLE ACCOUNTING TREE

:any
inst_ret...ear_return
load_latency
instruction_starvation
bandwidth_saturated
branch_misprediction
store_resources_saturated
instruction_latency
exceptio

258131	(54%)	65263	(13%)	6963	(1%)	13628	(2%)	56481	(11%)	29016	(6%)	7232	(1%)
8782	(67%)	1342	(10%)	30	(0%)	328	(2%)	1998	(15%)	45	(0%)		
4488	(32%)	507	(3%)	15	(0%)	104	(0%)			3772	(27%)	268	(1%)
4399	(57%)	731	(9%)			75	(0%)	746	(9%)			30	(0%)
7798	(90%)	1088	(12%)	179	(2%)	820	(9%)			30	(0%)		
164	(1%)	15	(0%)							6695	(58%)		
1238	(19%)	910	(14%)			358	(5%)	2043	(32%)	537	(8%)	15	(0%)

Branches can be expanded and explored

CONCLUSION

- Low overhead profiler
- Implements a novel cycle accounting methodology
- Visualization of reports require only a browser
- Open Source Tool (contributions welcome!)

RESOURCES

GOoDA

http://code.google.com/p/gooda/

GOoDA Visualizer

http://code.google.com/p/gooda-visualizer/