



Decoupling Dynamic Information Flow Tracking with a Dedicated Coprocessor

Hari Kannan, Michael Dalton, Christos Kozyrakis

Computer Systems Laboratory
Stanford University



Motivation

- Dynamic analysis help better understand SW behavior
 - Security, Debugging, Full system profiling
- Hardware support for such analyses very useful
 - Provides speed advantage over SW solutions
 - Systems manage **metadata** for analysis in hardware
- Implementation challenges
 - Storage overheads of metadata (Suh'05)
 - Processing of metadata
 - Need fast processing (low overheads)
 - Need cost effective implementation
- **Solution:** Tightly coupled coprocessor for analysis

Case Study – DIFT (Dynamic Information Flow Tracking)



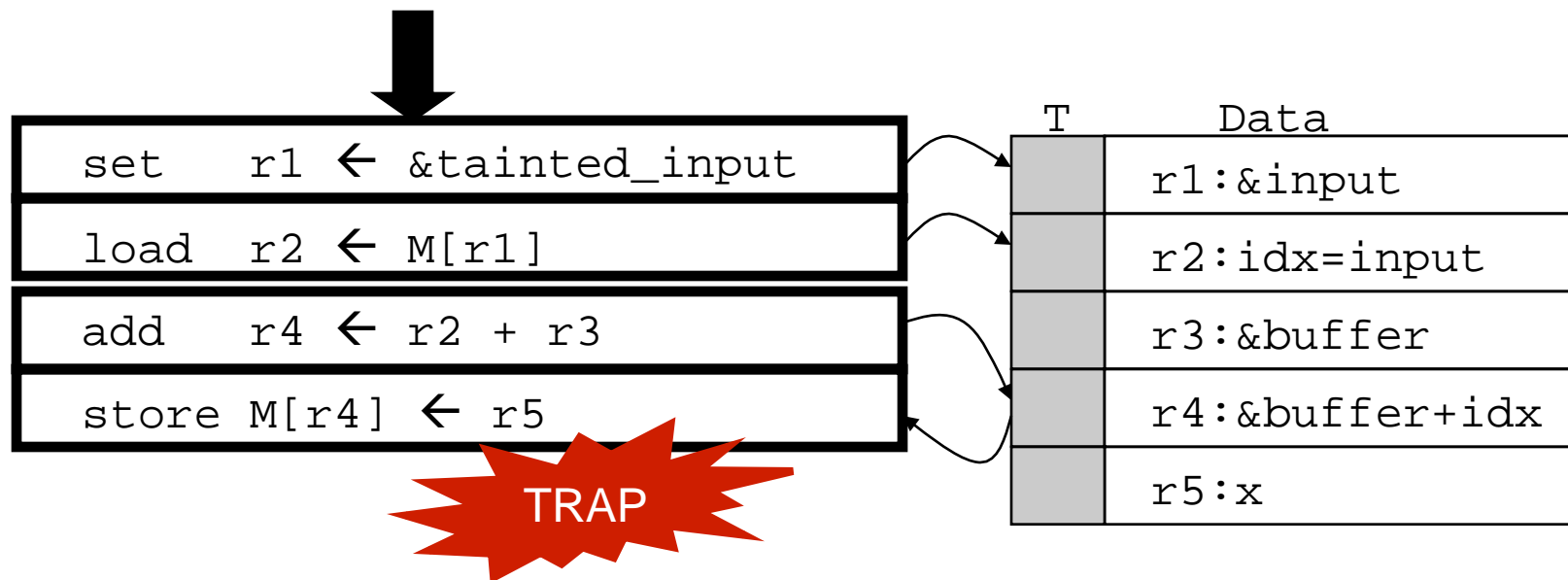
- DIFT taints data from untrusted sources
 - Extra tag bit per word marks if untrusted
- Propagate taint during program execution
 - Operations with tainted data produce tainted results
- Check for suspicious uses of tainted data
 - Tainted code execution
 - Tainted pointer dereference (code & data)
 - Tainted SQL command
- Can detect both low-level & high-level threats

DIFT Example: Memory Corruption



Vulnerable C Code

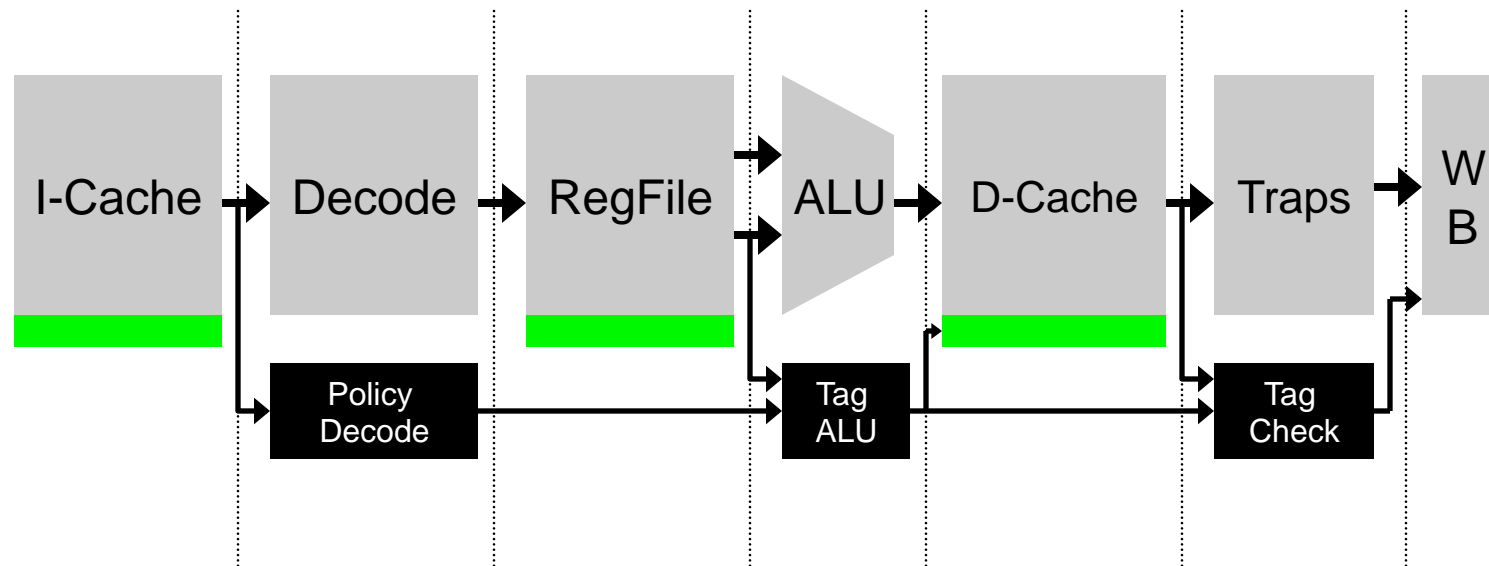
```
int idx = tainted_input;  
buffer[idx] = x; // memory corruption
```



- Tainted pointer dereference → security trap



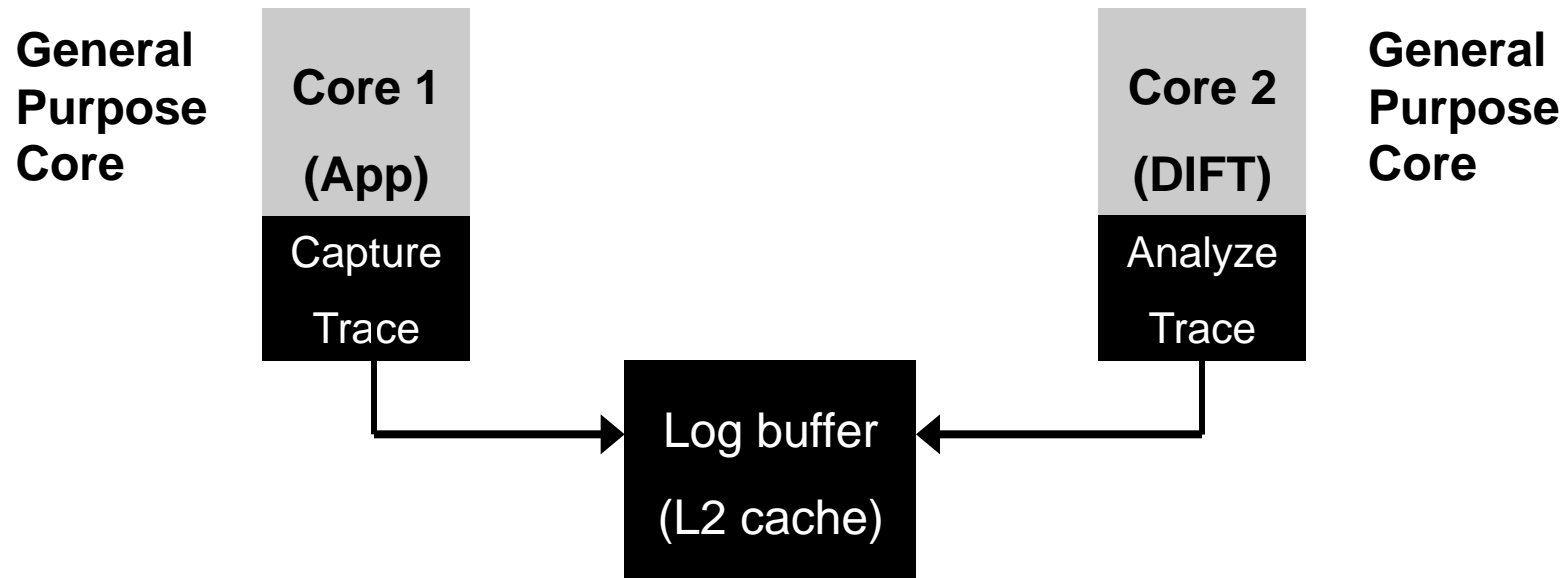
HW Option 1: In-core DIFT



- Integrated DIFT hardware [Dalton'07, Suh'04, Chen'05]
 - 👉 No performance, minor power, and minor area overhead
 - 👉 Invasive changes to processor
 - 👉 High design and validation costs
- 👉 Synchronizes metadata and data per instruction

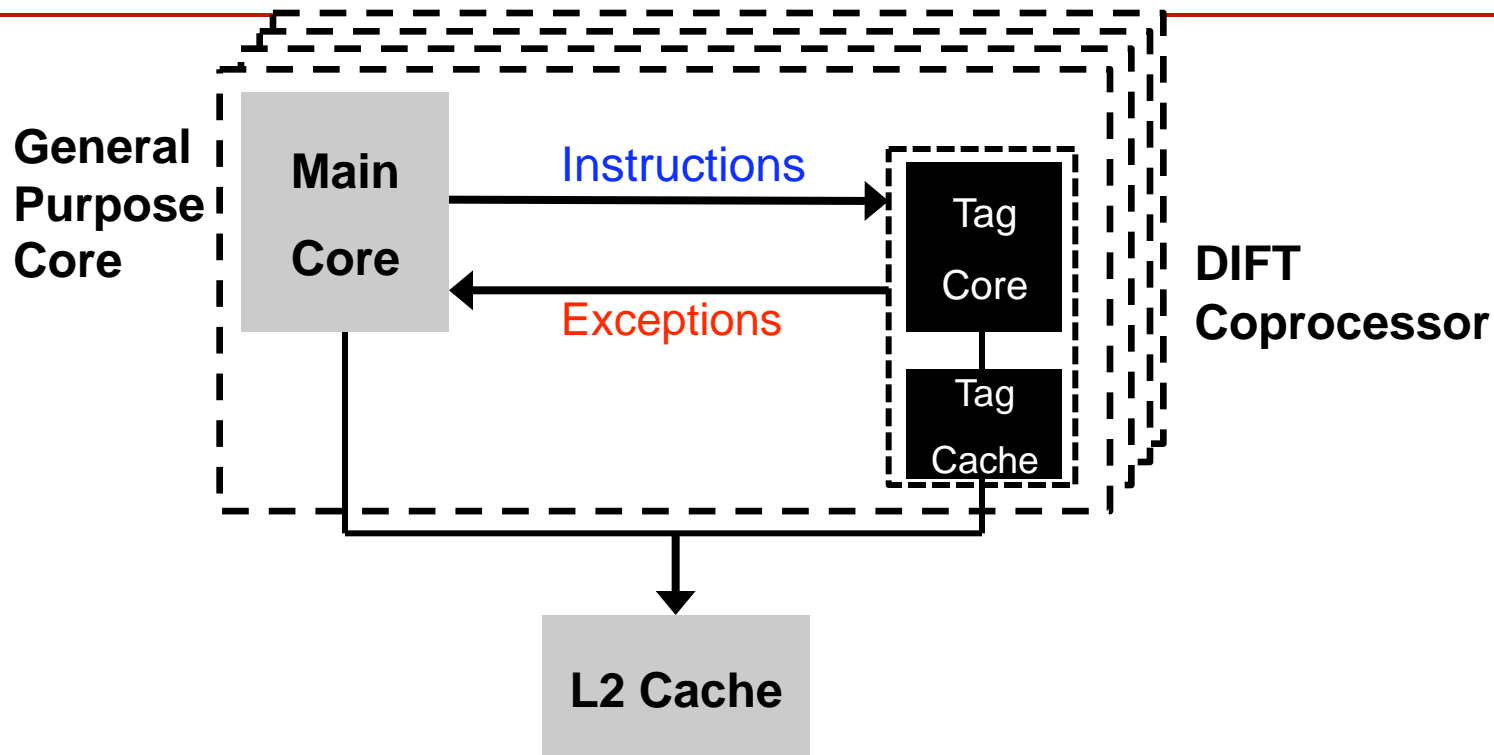


HW Option 2: Offloading DIFT



- **SW DIFT** on modified multi-core chip (e.g., CMU's LBA)
 - 👉 Flexible support for various analyses
 - 👉 Large area & power overhead (2nd core, trace compress)
 - 👉 Large performance overhead (DBT, memory traffic)
 - 👉 Significant changes to processor & memory hierarchy

Our Proposal: DIFT Coprocessor



- Off-core DIFT coprocessor (similar to watchdog processors)
 - 👉 Small performance, power, and area overhead
 - 👉 Minor changes to processor
 - 👉 Reuse across processor designs



Outline

- Motivation & Overview
- Software Interface of the coprocessor
- Architecture of the coprocessor
- Performance & Security Evaluation
- Conclusion



Coprocessor Setup

- A pair of policy registers
 - Accessible via coprocessor instructions
 - Could also be memory-mapped

- Policy granularity: operation type
 - Select input operands to be checked (if tainted)
 - Select input operands that propagate taint to output
 - Select the propagation mode (and, or, xor)

- ISA instructions decomposed to ≥ 1 operations
 - Types: ALU, logical, branch, memory, compare, FP, ...
 - Makes policies independent of ISA packaging
 - Same HW policies for both RISC & CISC ISAs

What happens without Proc/Coproc Synchronization?



Vulnerable C Code

```
int idx = tainted_input;  
buffer[idx] = x; // memory corruption
```



```
set   r1 ← &tainted_input  
load  r2 ← M[r1]  
add   r4 ← r2 + r3  
store M[r4]  
...  
exec (sys call)
```

T	Data
	r1:&input
	r2:idx=input
	r3:&buffer
	r4:&buffer+idx
	r5:x

**SYSTEM
COMPROMISE**

- Attacker executes system call → system compromise



System Calls as Sync points

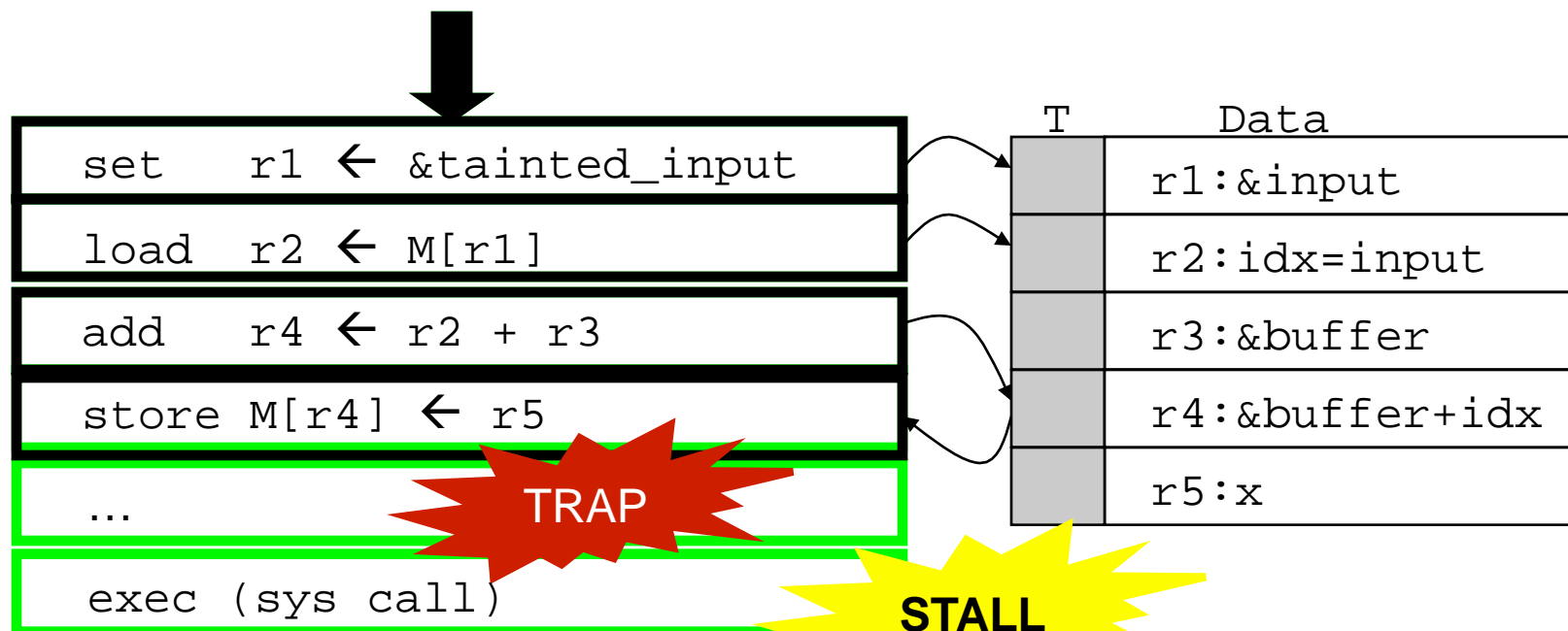
- **Key Idea:** Main core and coproc sync at system calls
- **Security:**
 - This prevents attacker from executing system calls
 - Application's corrupted address space can be discarded
 - Does not weaken the DIFT model
 - DIFT detects attack only at time of **exploit**, not corruption
- **Performance:**
 - Synchronization overhead typically tens of cycles
 - Function of decoupling queue size
 - Lost in the noise of system call overheads (hundreds of cycles)



System Call Synchronization

Vulnerable C Code

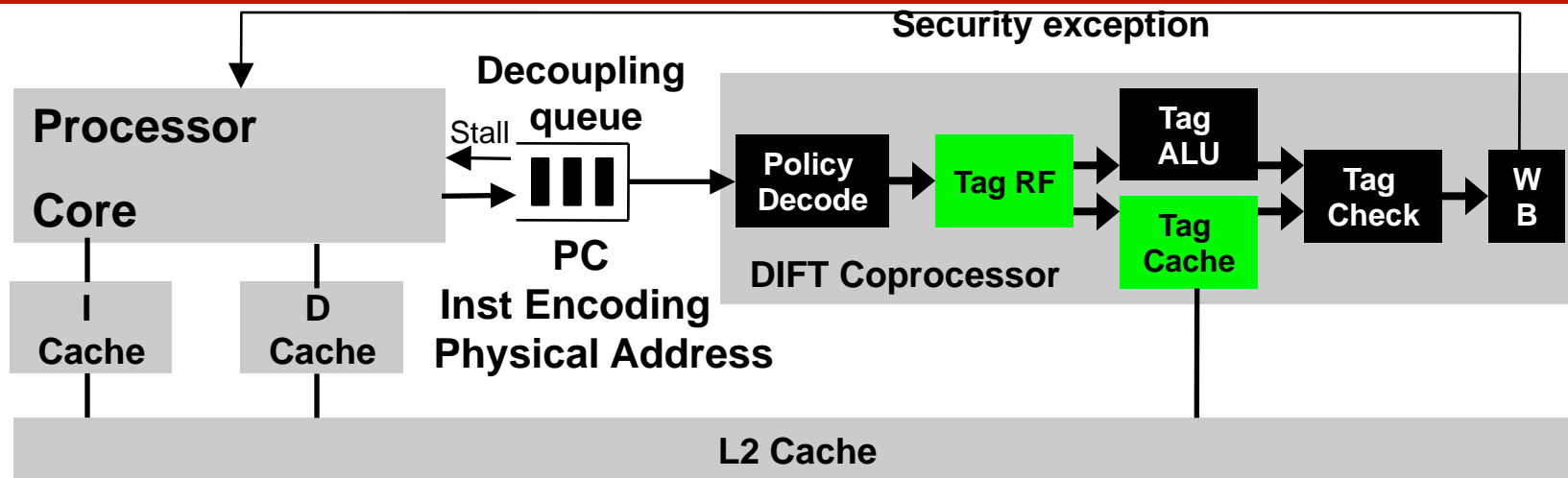
```
int idx = tainted_input;  
buffer[idx] = x; // memory corruption
```



- Tainted pointer dereference → security exception



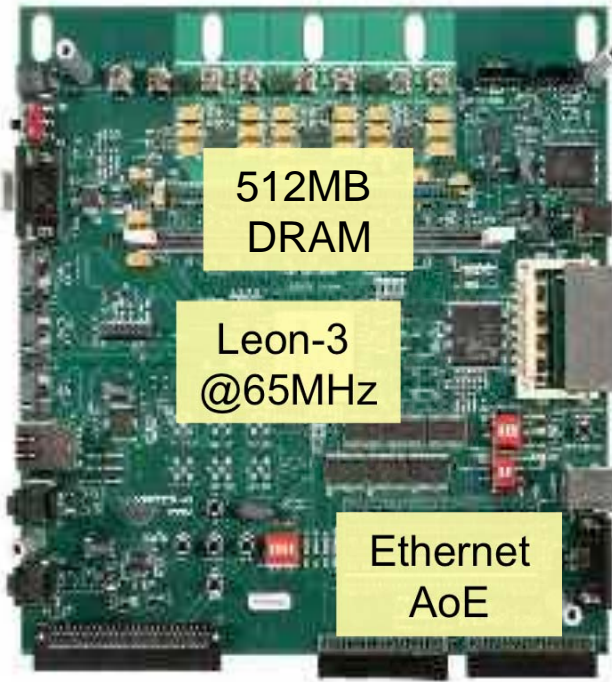
Coprocessor Design



- **DIFT functionality in a coprocessor**
 - 4 tag bits of metadata per word of data
- **Coprocessor Interface (via decoupling queue)**
 - Pass committed instruction information
 - Instruction encoding could be at micro-op granularity (in x86)
 - Physical address obviates need for MMU in coprocessor



Prototype



■ Hardware

- Paired with simple SPARC V8 core (Leon-3)
- Mapped to FPGA board

■ Software

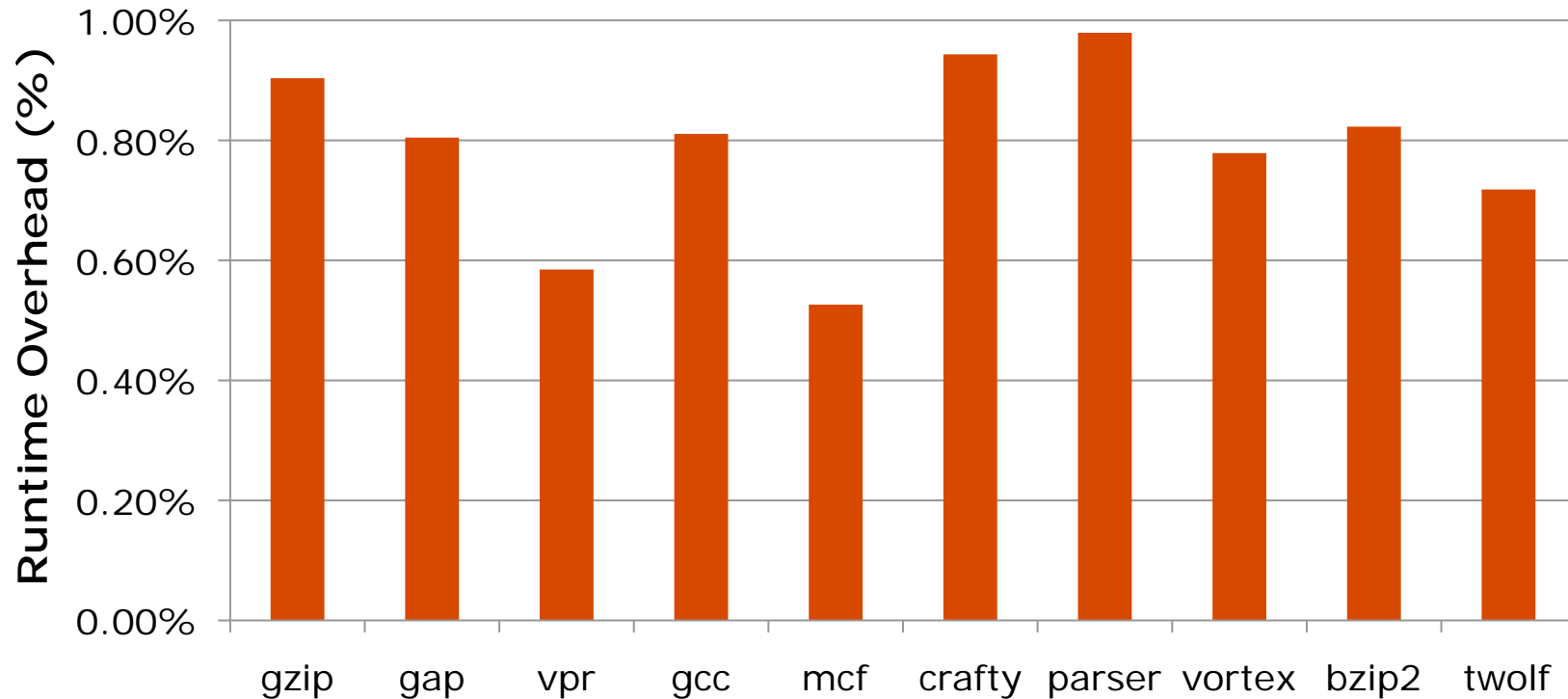
- Fully-featured Linux 2.6

■ Design statistics

- Clock frequency: same as original
- Logic: +7.5% overhead
 - ... of simple in-order core with no speculation



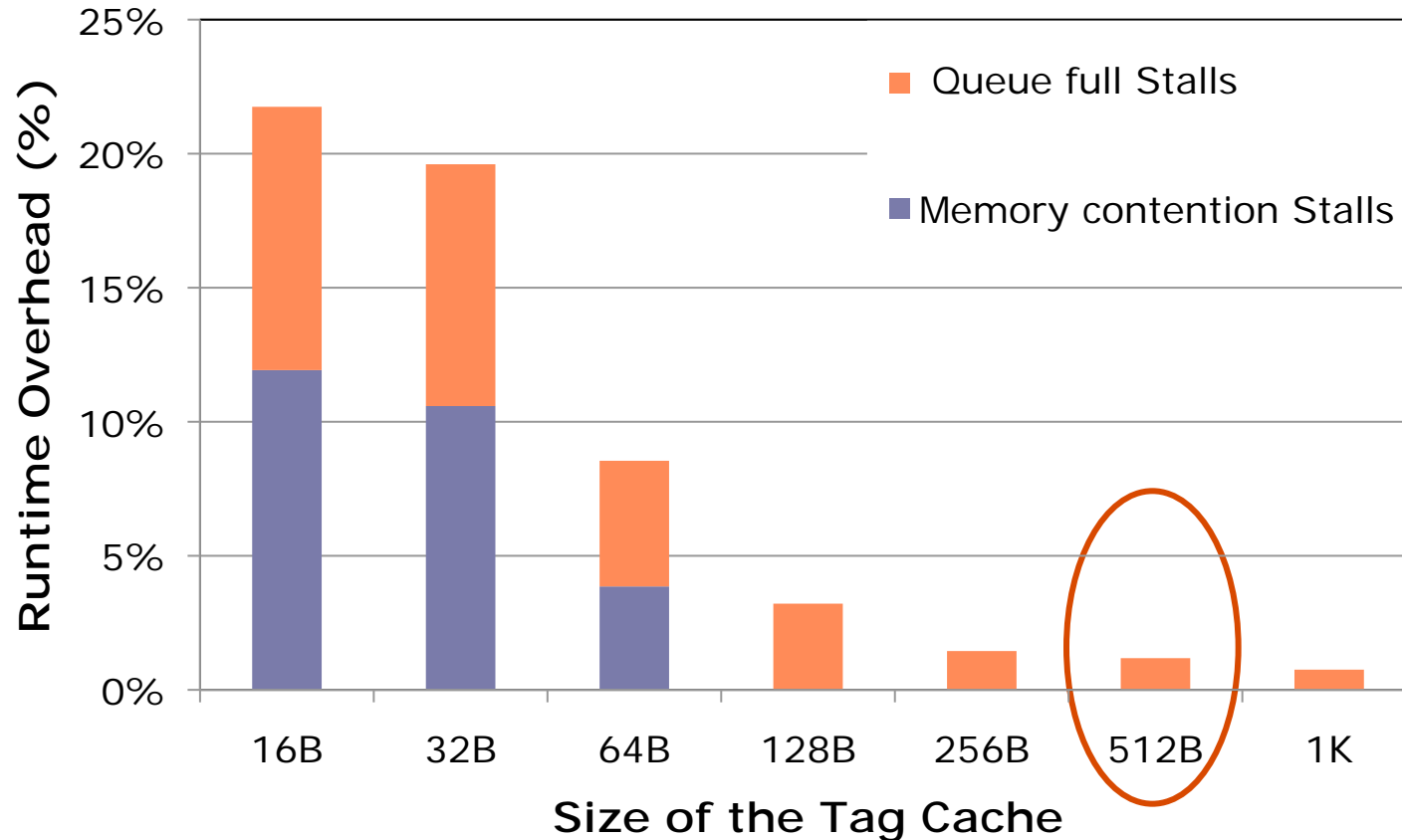
System Performance Overheads



- Runtime overhead < 1% over SPEC benchmarks
 - 512 byte tag cache
 - 6-entry decoupling queue



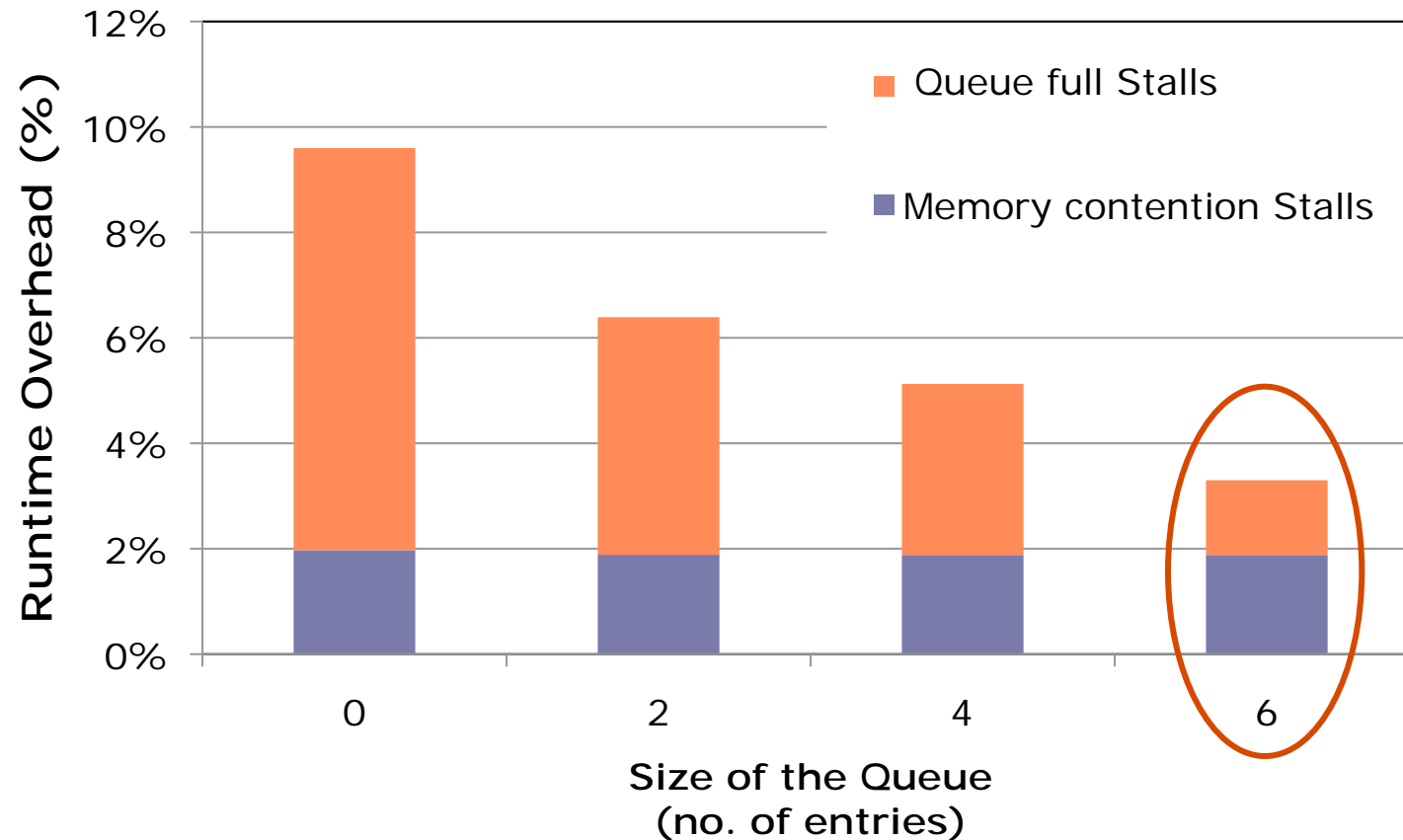
Scaling the tag cache



- Worst case micro-benchmark
 - 512-byte tag cache provides good performance



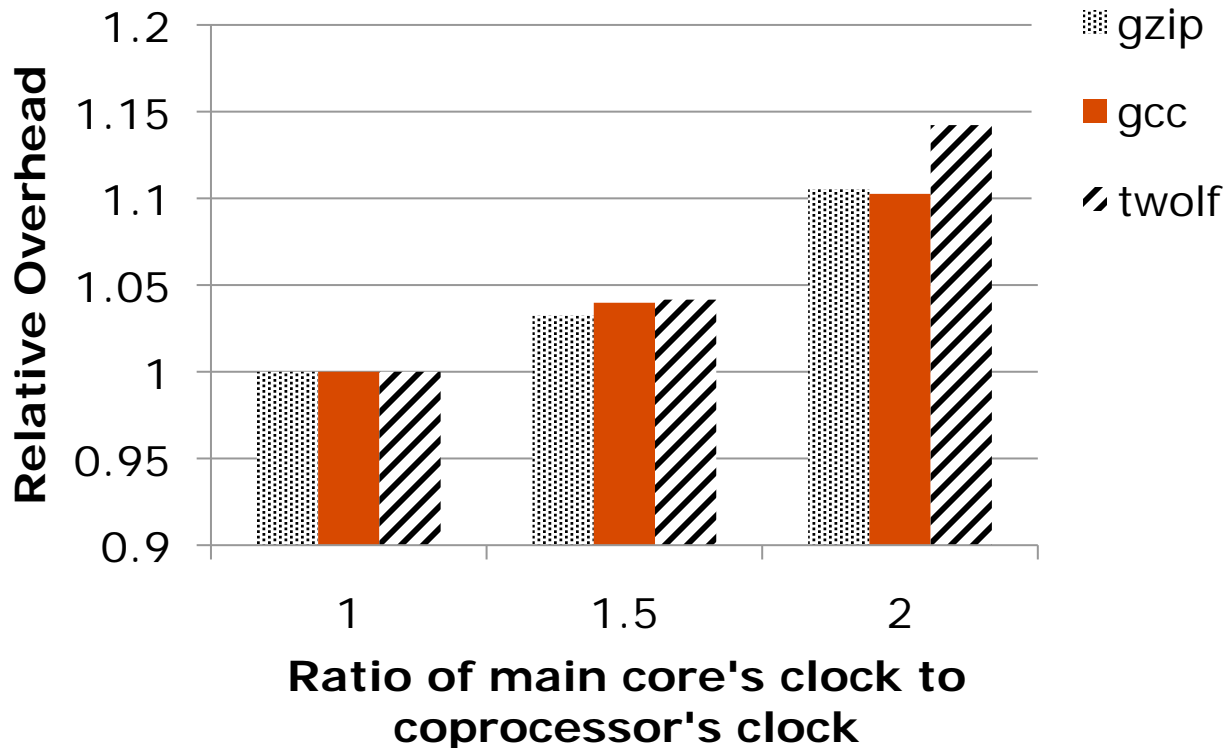
Scaling the decoupling queue



- Worst case micro-benchmark
 - 6 entry queue reduces performance overhead



Coprocessors for complex cores



- Modest overheads with higher IPC cores
 - Because main core rarely achieves peak IPC (=1)
 - Coprocessor performs very simple operations
- Implies coprocessor can be paired with complex cores



Security Policies Overview

		P Bit	T Bit	B Bit	S Bit
Buffer Overflow Policy	Identify all pointers, and track data taint. Check for illegal tainted ptr use.	Y	Y		
Offset-based attacks (control ptr)	Track data taint, and bounds check to validate.			Y	
Format String Policy	Check tainted args to print commands.		Y		Y
SQL/XSS	Check tainted commands.		Y		Y
Red zone Policy	Sandbox heap data.				Y
Sandboxing Policy	Protect the security handler.				Y



Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Unmodified SPARC binaries from real-world programs
 - Basic/net utilities, servers, web apps, search engine



Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Protection against low-level memory corruptions
 - Both in userspace and kernelspace



Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Protection against semantic vulnerabilities



Security Experiments

Program	Lang.	Attack	Detected Vulnerability
tar	C	Directory Traversal	Open tainted dir
gzip	C	Directory Traversal	Open tainted dir
Wu-FTPD	C	Format String	Tainted '%n' in vfprintf string
SUS	C	Format String	Tainted '%n' in syslog
quotactl syscall	C	User/kernel pointer dereference	Tainted pointer to kernelspace
sendmail	C	Buffer (BSS) Overflow	Tainted code ptr
polymorph	C	Buffer Overflow	Tainted code ptr
htdig	C++	Cross-site Scripting	Tainted <script> tag
Scry	PHP	Cross-site Scripting	Tainted <script> tag

- Protection is independent of programming language
 - Propagation & checks at the level of basic ops



Conclusions

- Hardware dynamic analyses aid program understanding
 - Decoupling analyses from main core essential for practicality
- Proposed a tightly coupled coprocessor for DIFT
 - Does not compromise security model
 - Has low performance and area overheads
- Full-system FPGA prototype
 - Reliably catches exploits in user & kernel-space