WatchdogLite: Hardware-Accelerated Compiler-Based Pointer Checking

Santosh Nagarakatte
Rutgers University

Milo M.K. Martin
Steve Zdancewic
University of Pennsylvania

Project goal:
Make C/C++ safe and secure

Why?

Lack of ***memory safety*** is the root cause of serious **bugs** and **security vulnerabilities**

Security Vulnerabilities due to Lack of Memory Safety

Adobe Acrobat – buffer overflow

CVE-2013-1376- *Severity: 10.0 (High)* January 30, 2014

Oracle MySQL – buffer overflow

CVE-2014-0001 - *Severity: 7.5 (High)* January 31, 2014

Firefox – use-after-free vulnerability

CVE-2014-1486 - *Severity: 10.0 (High)* February 6, 2014

Google Chrome– use-after-free vulnerability

CVE-2013-6649 - *Severity: 7.5 (High)* January 28, 2014

DHS/NIST National Vulnerability Database:

- Last three months: **92 buffer overflow and 23 use-after-free disclosures**
- Last three years: **1135 buffer overflows and 425 use-after-free disclosures**

Project Overview & Progression

Memory safety has two components:

Bounds safety

Use-after-free safety

Project Overview & Progression

Memory safety has two components:

Bounds safety | Use-after-free safety

Hardware

HardBound

[ASPLOS 2008]

- Pointer-based
- Disjoint metadata
- ~10% overhead

Software

SoftBound

[PLDI 2009]

- Pointer-based
- Disjoint metadata
- ~75% overhead

Project Overview & Progression

Memory safety has two components:

Bounds safety | **Use-after-free safety**

Hardware

HardBound
[ASPLOS 2008]
- Pointer-based
- Disjoint metadata
- ~10% overhead

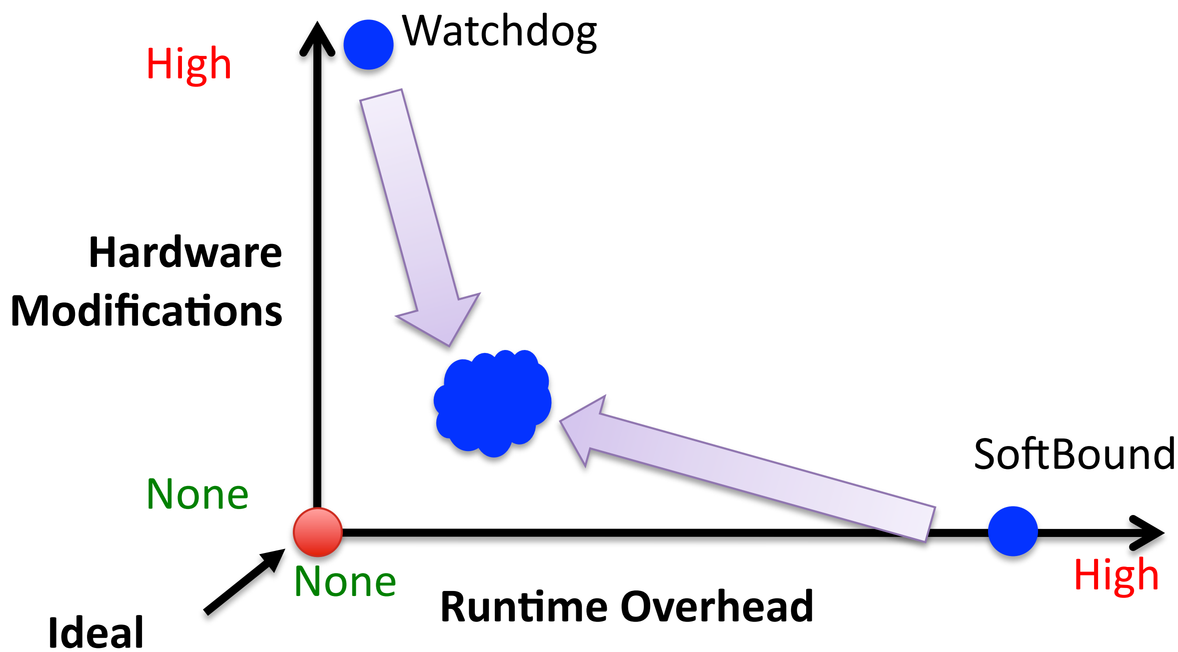
Watchdog
[ISCA 2012]
- Pointer-based, disjoint
- Unique identifier check
- ~15% overhead

Software

SoftBound
[PLDI 2009]
- Pointer-based
- Disjoint metadata
- ~75% overhead

CETS
[ISMM 2010]
- Pointer-based, disjoint
- **Unique identifier check**
- ~50% overhead

WatchdogLite



Pointer-based Checking with disjoint metadata

- Compiler transformation+ four hardware instructions
- Bounds + Use-after-free safety
- 29% overhead
- Similar to Intel MPX for bounds safety (concurrent work)

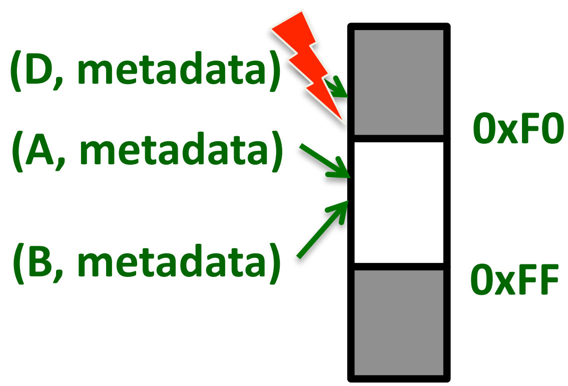
Background on Pointer Checking

Pointer-Based Bounds Checking

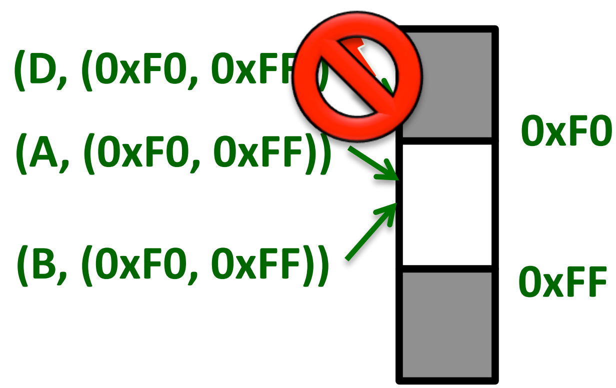
[Ccured, SafeC, SoftBound, CETS, MSCC, Patil & Fischer, …]

- Metadata is maintained with pointers
  - Each pointer has a view of memory it can access
- Challenges
  - What metadata do you maintain?
  - How do you propagate this metadata?

Every pointer has metadata



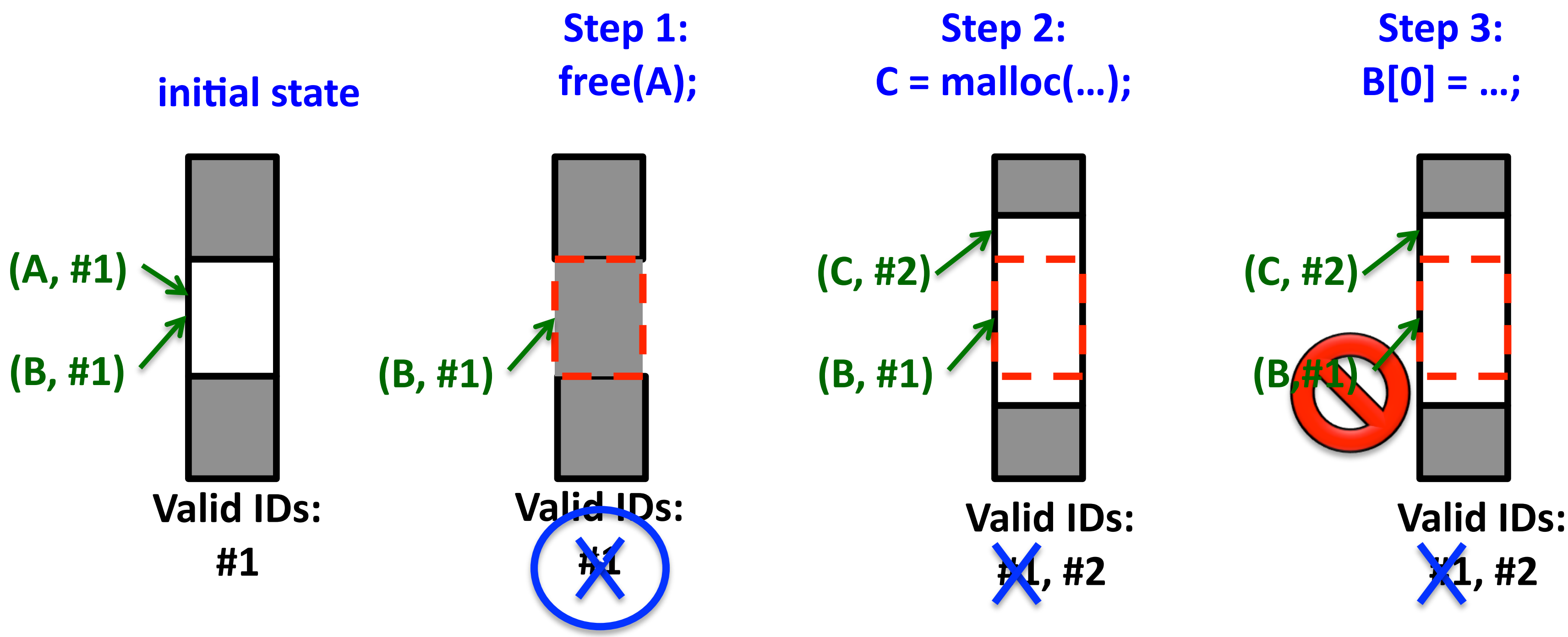
For Bounds Safety



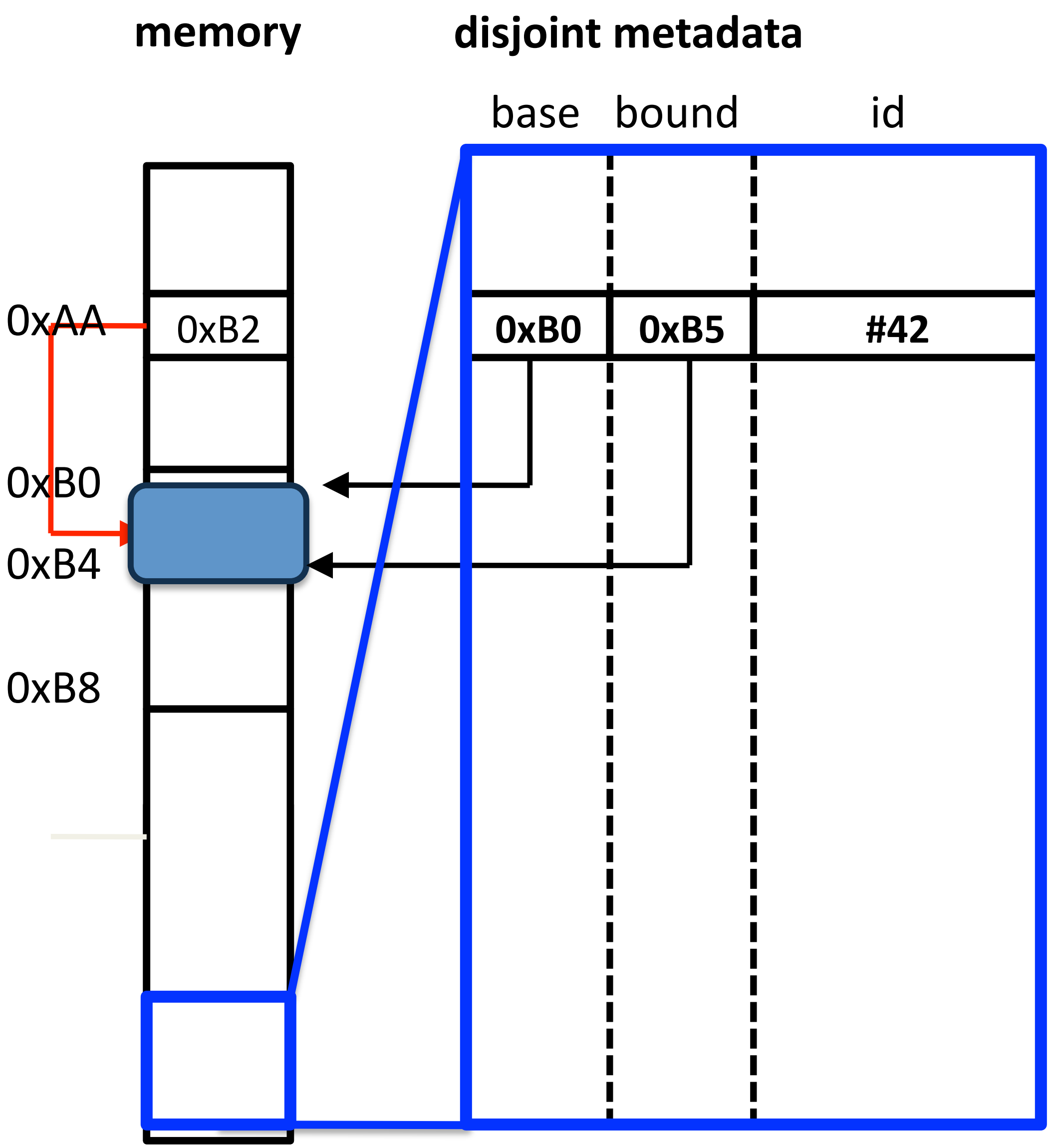
Identifier Checking for Use-After Free Safety

[SafeC, Patil&Fischer, MSCC, CETS, Watchdog, …]

- Allocate ***unique identifier (UID)*** for each allocation
  - Record the set of valid identifiers
  - Track this UID with each pointer
  - Invalidate identifiers on memory deallocation
  - Check for identifier validity on memory accesses



Disjoint Metadata



Memory layout unchanged

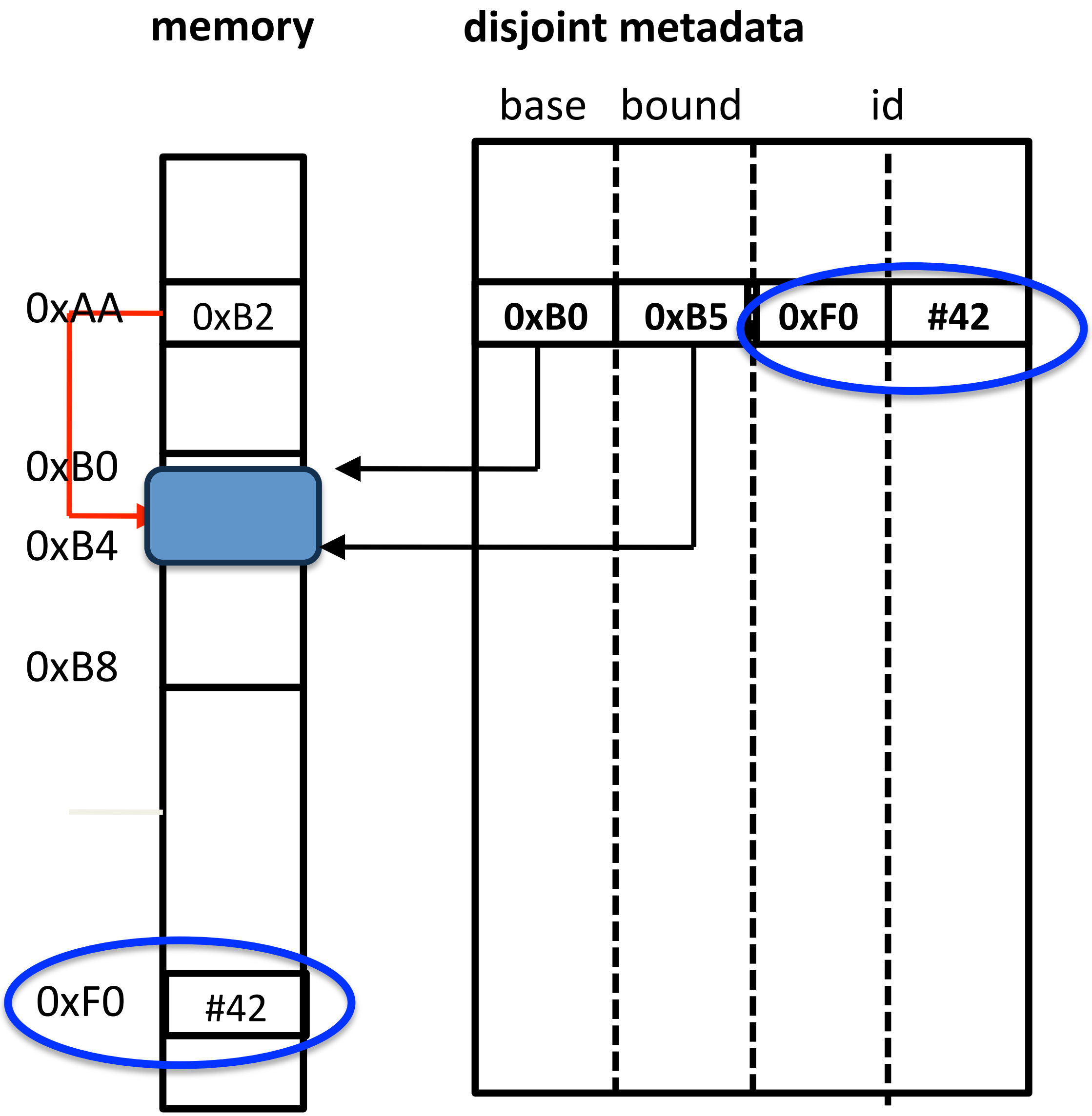
- Protects metadata
- Only pointers in memory have disjoint metadata

Mapped to some part in virtual memory

- Allocated on demand

Lock & Key Checking

[Patil&Fischer, MSCC, CETS, Watchdog, …]



Split UID into "lock" and "key"

Allocation:

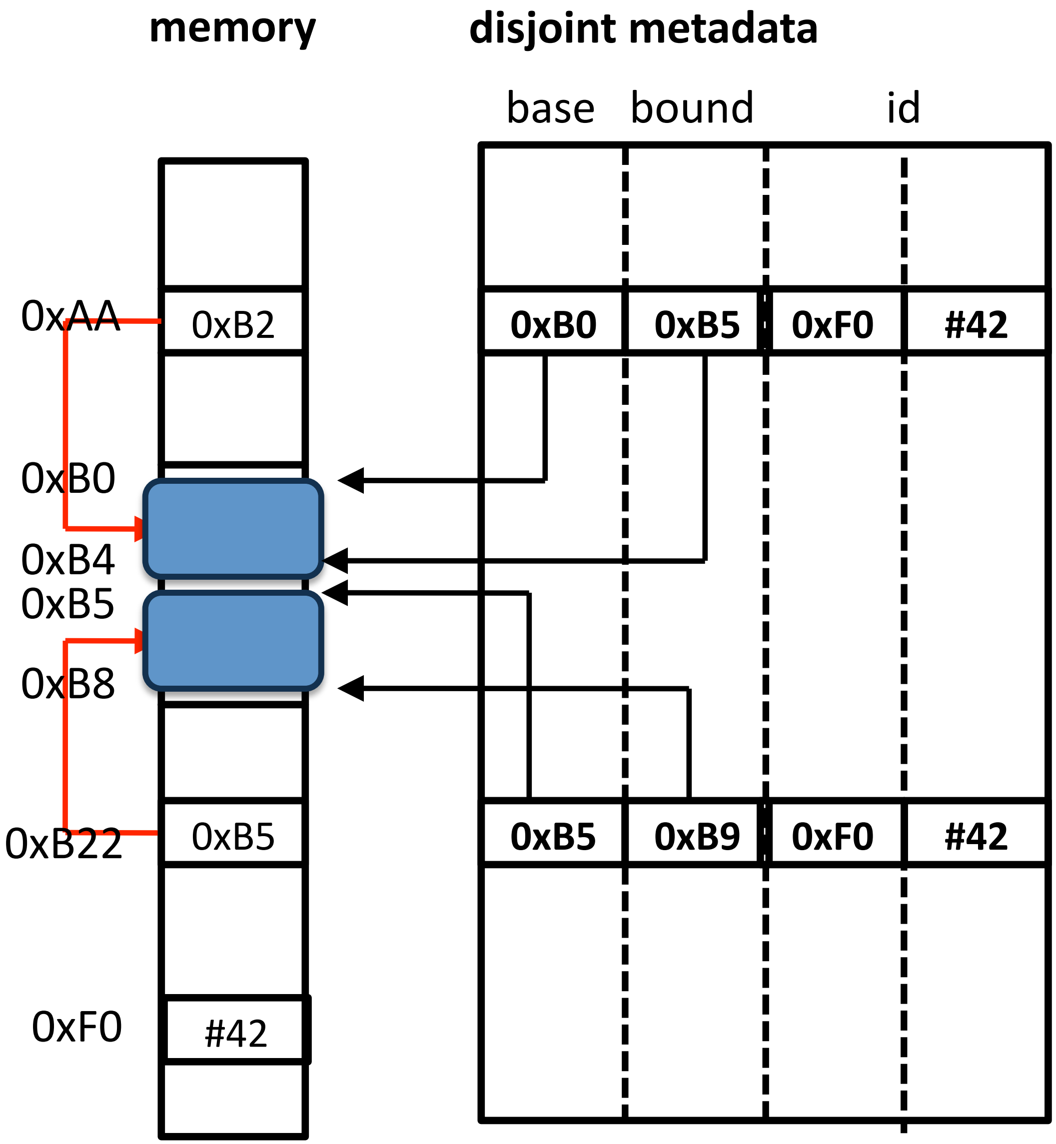
memory[lock] = key

Invariant:

memory[lock] == key

Lock & Key Checking

[Patil&Fischer, MSCC, CETS, Watchdog, …]



Split UID into "lock" and "key"

Allocation:

memory[lock] = key

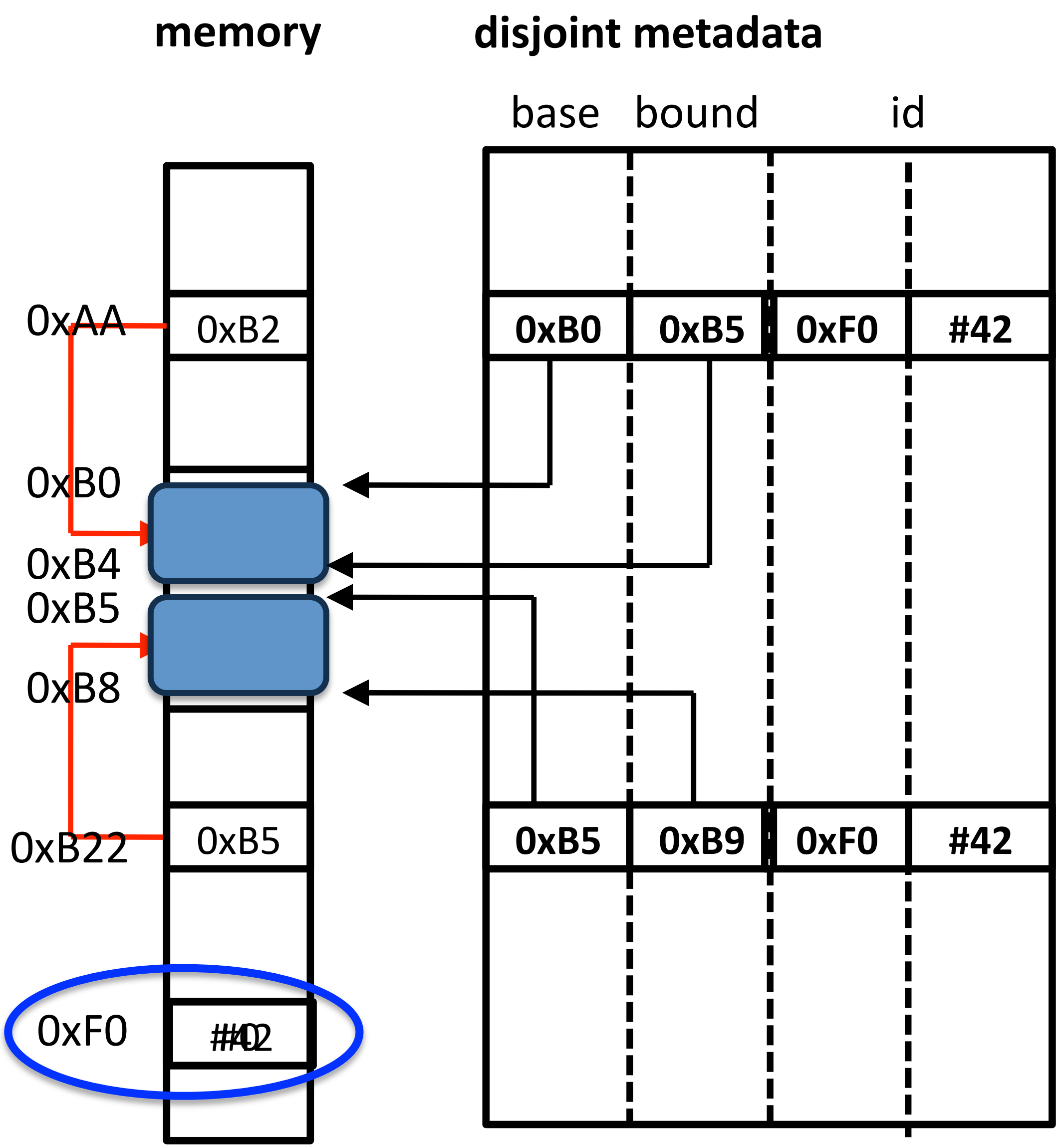
Invariant:

memory[lock] == key

Pointer copies ➔ copy metadata

Lock & Key Checking

[Patil&Fischer, MSCC, CETS, Watchdog, …]



Split UID into "lock" and "key"

Allocation:

memory[lock] = key

Invariant:

memory[lock] == key

Pointer copies ➔ copy metadata

Deallocation:

memory[lock] = 0

Check is "load" + "compare"

Hardware vs Software Implementation

Task	**Watchdog** [ISCA 2012]	**SoftBoundCETS** [PLDI 2009, ISMM 2010]
Pointer detection	Conservative	Accurate with compiler

Hardware vs Software Implementation

Task	Watchdog [ISCA 2012]	SoftBoundCETS [PLDI 2009, ISMM 2010]
Pointer detection	Conservative	Accurate with compiler
Op Insertion	Micro-op injection	Compiler inserted instructions

Hardware vs Software Implementation

Task	Watchdog [ISCA 2012]	SoftBoundCETS [PLDI 2009, ISMM 2010]
Pointer detection	Conservative	Accurate with compiler
Op Insertion	Micro-op injection	Compiler inserted instructions
Metadata Propagation	Copy elimination using register renaming	Standard dataflow analysis

Hardware vs Software Implementation

Compiler can do these tasks efficiently

Task	Watchdog [ISCA 2012]	So... [PLDI 2009, ISMM 2010]
Pointer detection	Conservative	Accurate with compiler
Op Ins...	...jection	Compiler inserted instructions
Metad... Propagation	...ation using register renaming	Standard dataflow analysis
Checks	+ fast checks (implicit) - no check optimization	- Instruction overhead + Check optimization
Metadata Loads/Stores	+ Fast lookups	- Instruction overhead

Hardware can accelerate checks & metadata accesses

What is WatchdogLite?

Hardware acceleration with new instructions for compiler based pointer checking

Instructions added to the ISA

- Bounds check & use-after-free check instructions
- Metadata load/store instructions

Pack four words of metadata into a single wide register

- Single wide load/store → eliminates port pressure
- Avoid implicit registers for the new instructions
- Reduces spills/restores due to register pressure

Spatial (Bound) Check Instruction

```
int p;
 …
~~if( q < q_base ||
    q + sizeof(int) >= q_bound){
   abort();~~
}
p = *q;
```

Schk.size imm(r1), ymm0

Supports all addressing modes

Size of the access encoded

Operates only on registers

Executes as one micro-op

Latency is not critical

5 instructions for the spatial check

Temporal (Use-After-Free) Check Instruction

int p;

…

~~if(q_key!= *q_lock){~~

~~abort();~~

~~}~~

Tchk ymm0

p = *q;

Performs a memory access

3 instructions for the temporal check

Executes as two micro-ops

Latency is not critical

Metadata Load/Store Instructions

int *p, **q;

...

~~p_metadata = table_lookup(q);~~ Metaload %ymm0, imm(%rax)

p = *q;

..

~~table_lookup(q) = p_metadata~~ Metastore imm(%rax), %ymm0

*q = p

14 instructions for the metadata load

16 instructions for the metadata store

Performs a wide load/store

Executes as two micro-ops

– address computation

-- wide load/store uop

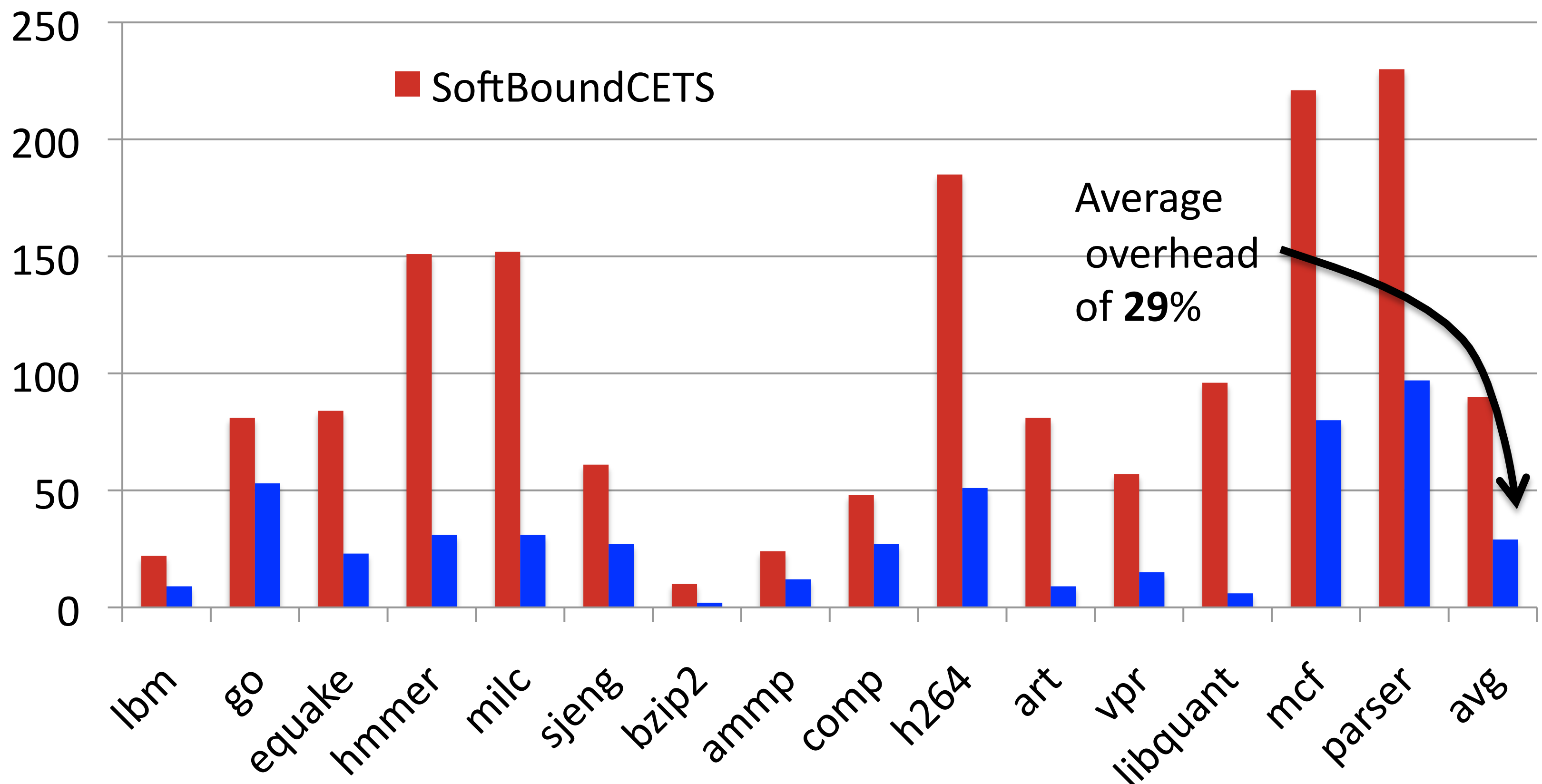
Shadow space for the metadata

See Paper For ….

- Compiler transformation to use wide metadata
- Metadata organization
- Check elimination effectiveness
- Effectiveness in detecting errors
- Narrow mode instructions
- Comparison of related work

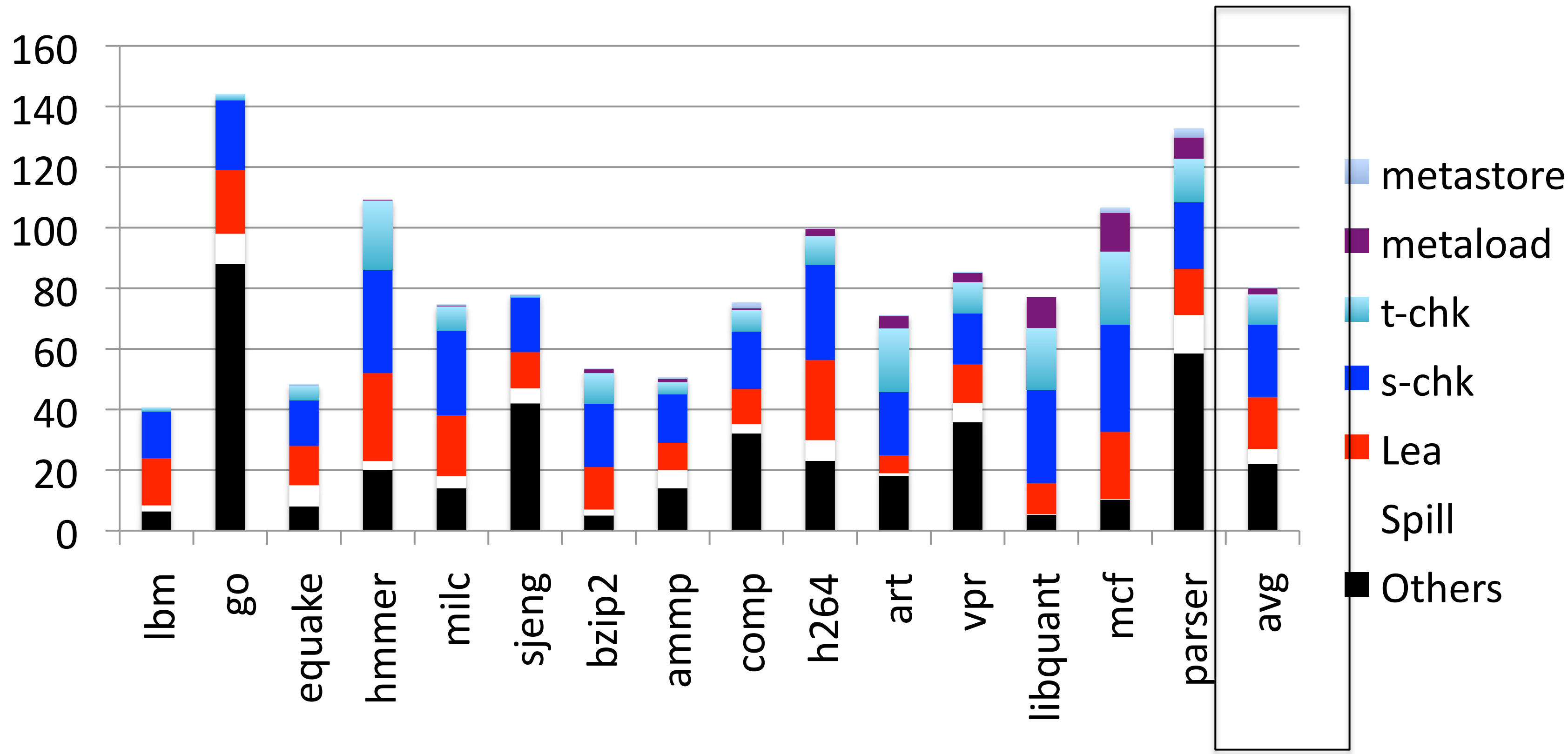
Evaluation

Evaluation – Performance Overheads



- Timing simulations of wide-issue out-of-order x86 core
- Average performance overhead: **29%**
  - Reduces average from 90% with SoftBoundCETS

Remaining Instruction Overhead



- Average instruction overhead reduces to 81% (from 180% with SoftBoundCETS)
- Spatial checks → better check optimizations can help
- Lea instructions → change code generator

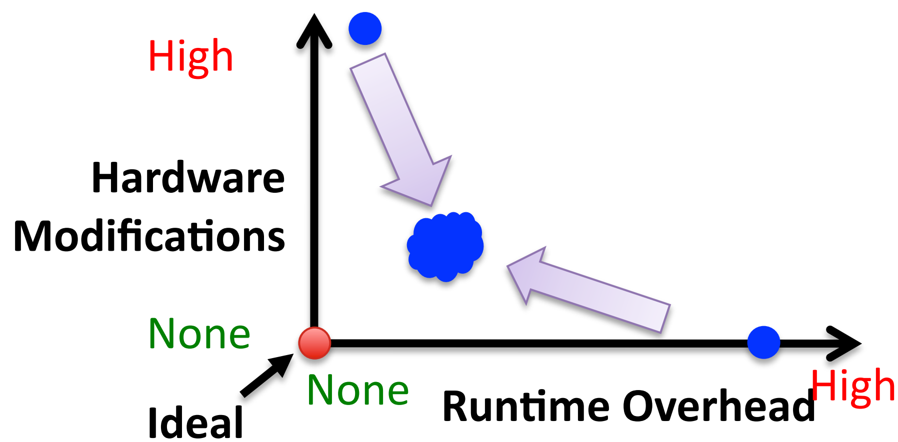
Intel MPX (Concurrent Work)

- In July 2013, Intel MPX announced ISA specification
  - Similar hardware/software approach
    - Pointer-based checking: base and bounds metadata
    - Disjoint metadata in shadow space
    - Adds new instructions for bounds checking
  - Differences
    - Adds new bounds registers vs reusing existing AVX registers
    - Changes calling conventions to avoid shadow stack
    - Backward compatibility features
      - Interoperability with un-instrumented and instrumented code
      - Validates metadata by redundantly encoding pointer in metadata
      - Calling un-instrumented code clears bounds registers
    - Does not perform use-after-free checking

Conclusion

- Safety against buffer overflows & use-after-free errors
  - Pointer based checking
  - Bounds and identifier metadata
  - Disjoint metadata
- WatchdogLite
  - Four new instructions for compiler-based pointer checking
  - Four new instructions
  - Packs the metadata in wide registers

Leveraging the compiler enables WatchdogLite to use simpler hardware for comprehensive memory safety



Thank You

Try SoftBoundCETS for LLVM-3.4

http://github.com/santoshn/softboundcets-34/