

Locating Cache Performance Bottlenecks Using Data Profiling

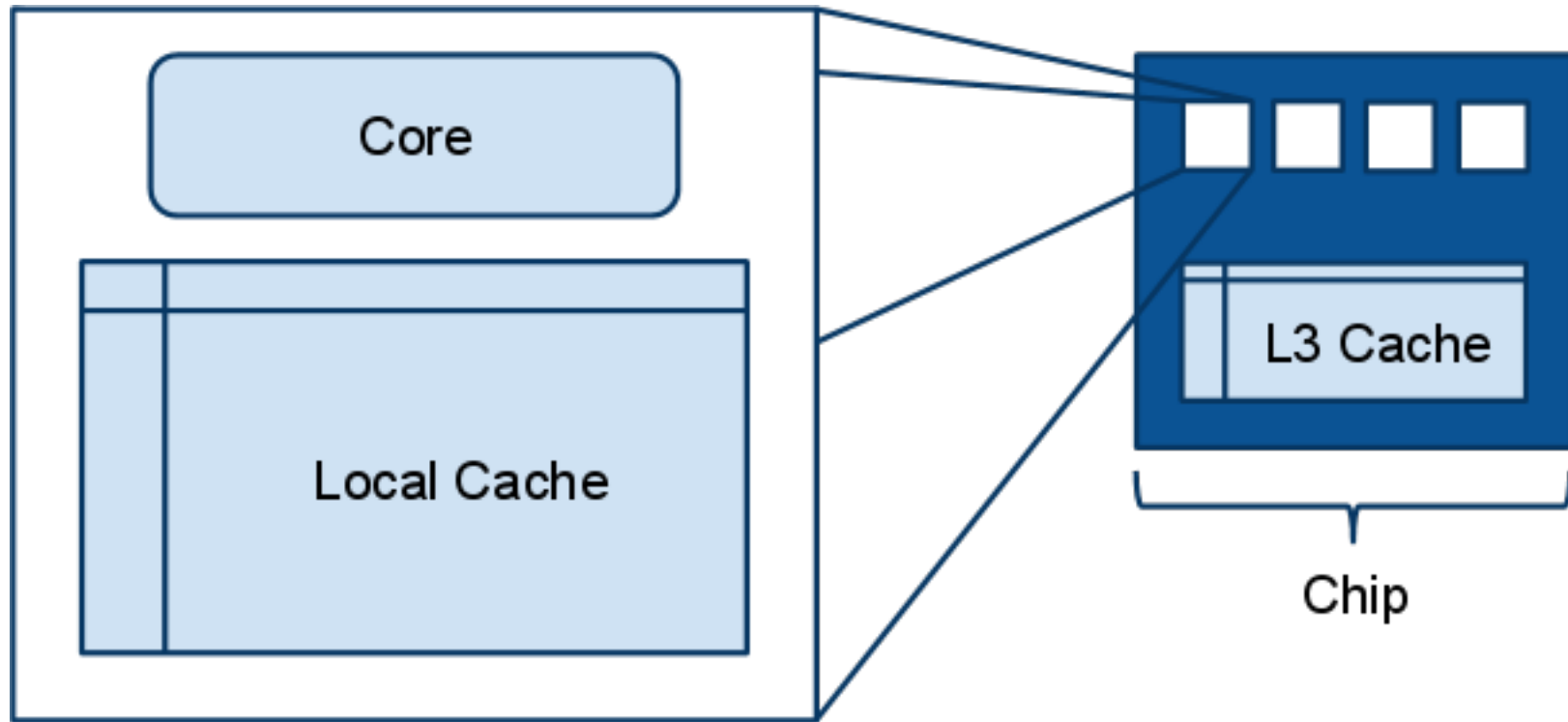
Aleksey Pesterev Nickolai Zeldovich
Robert T. Morris

Massachusetts Institute of Technology

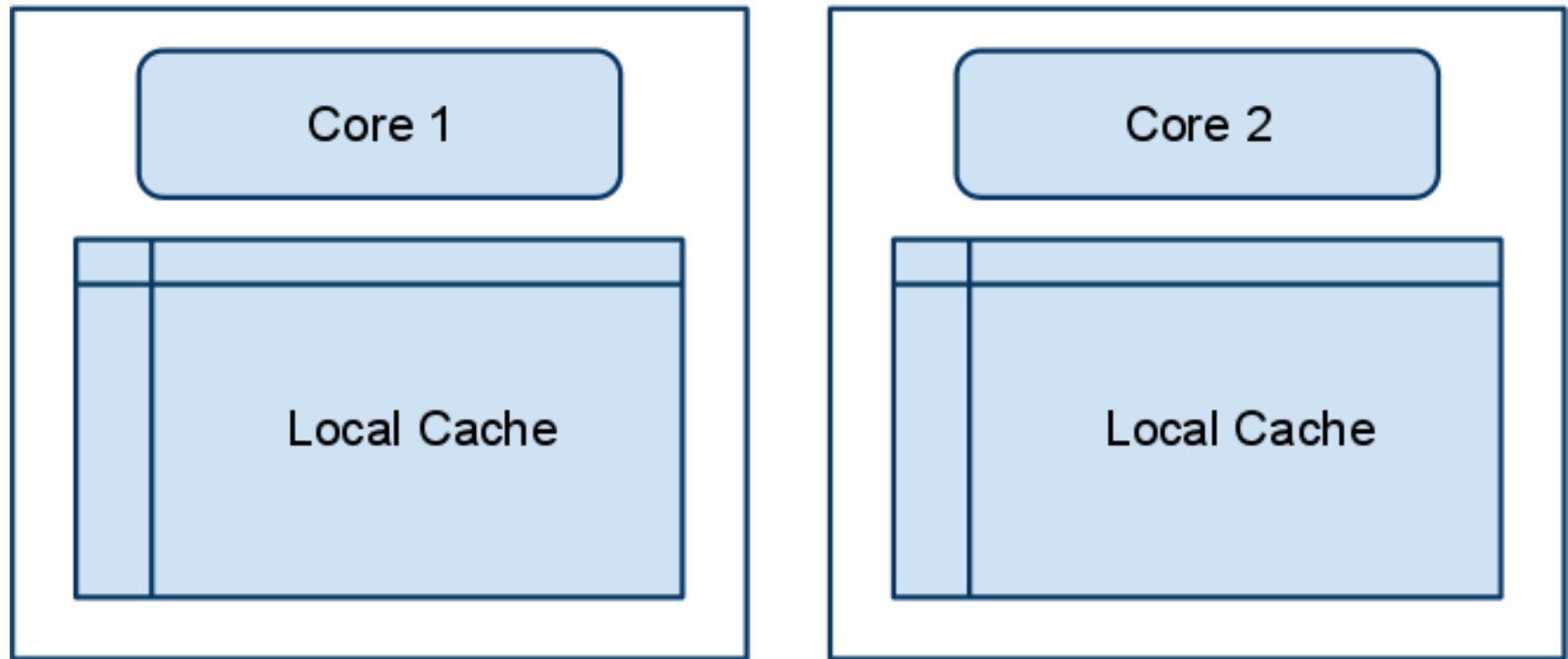
Overview

- Cache misses can degrade performance.
- DProf locates cache misses.
- DProf helps find why cache misses occur.
- Cache Miss Types
 - False Sharing
 - True Sharing
 - Conflict
 - Capacity

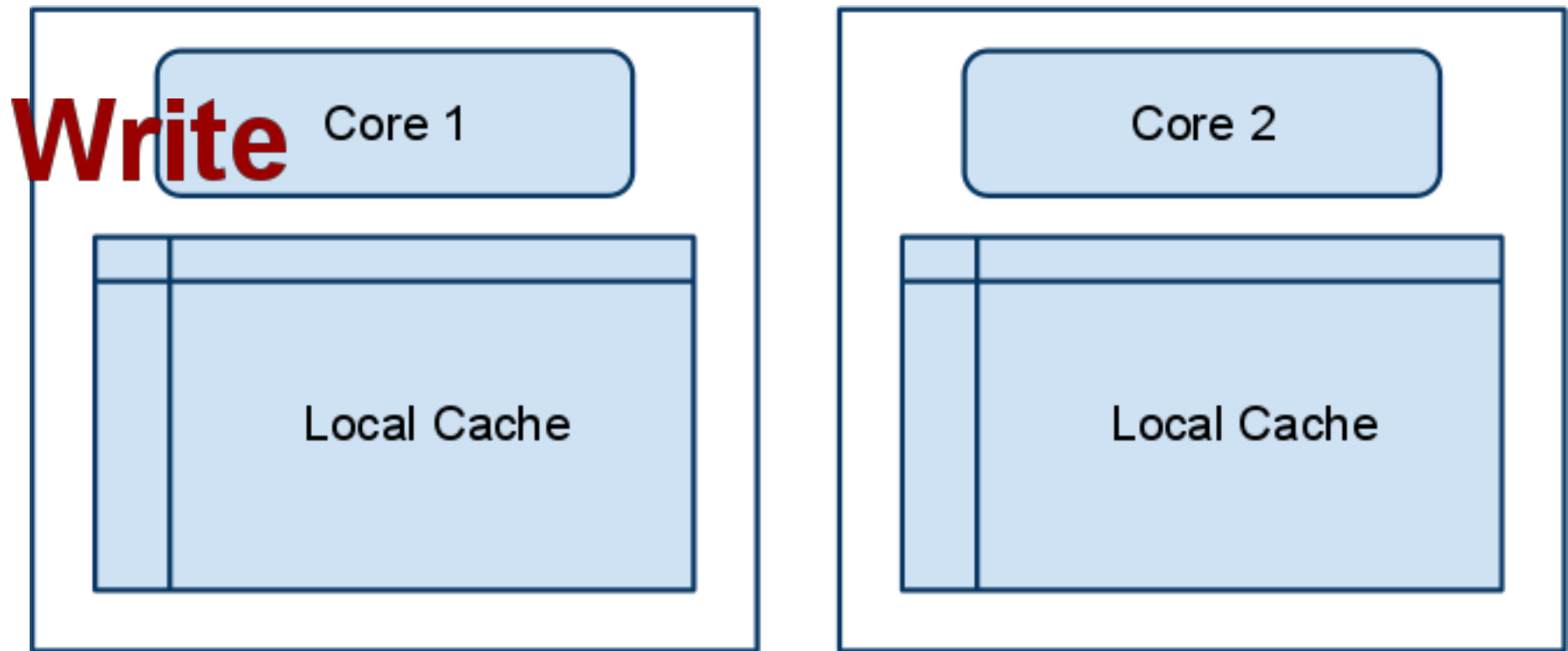
Multicore Chip



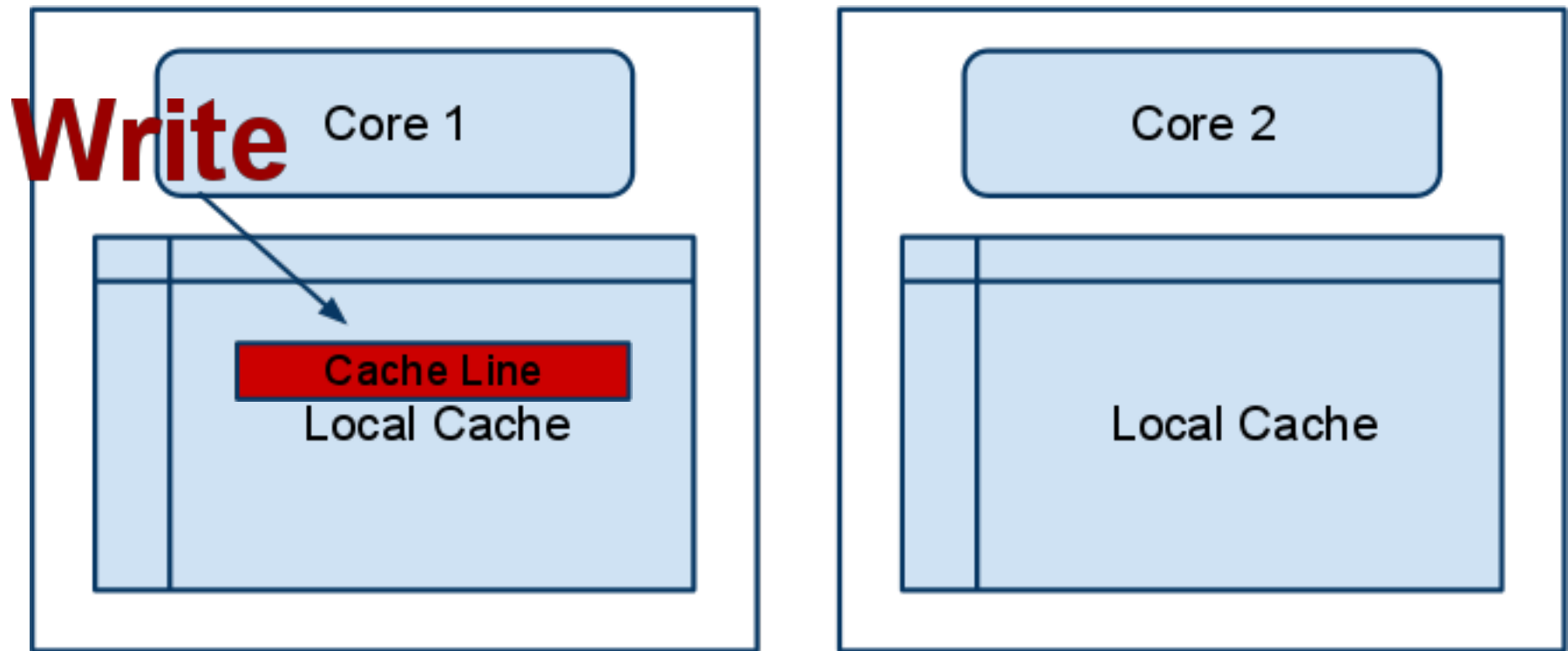
True Sharing



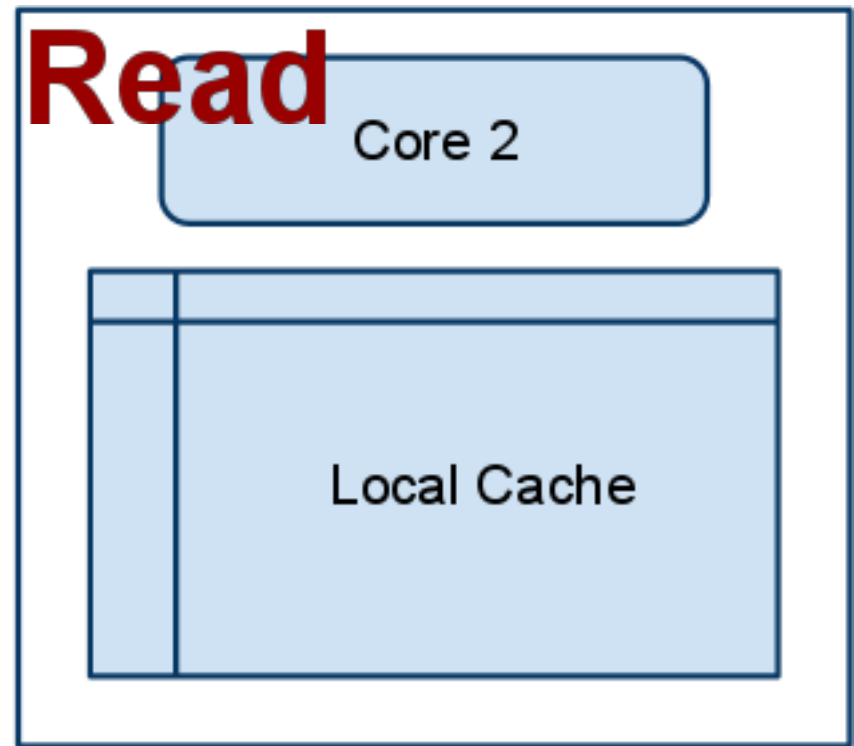
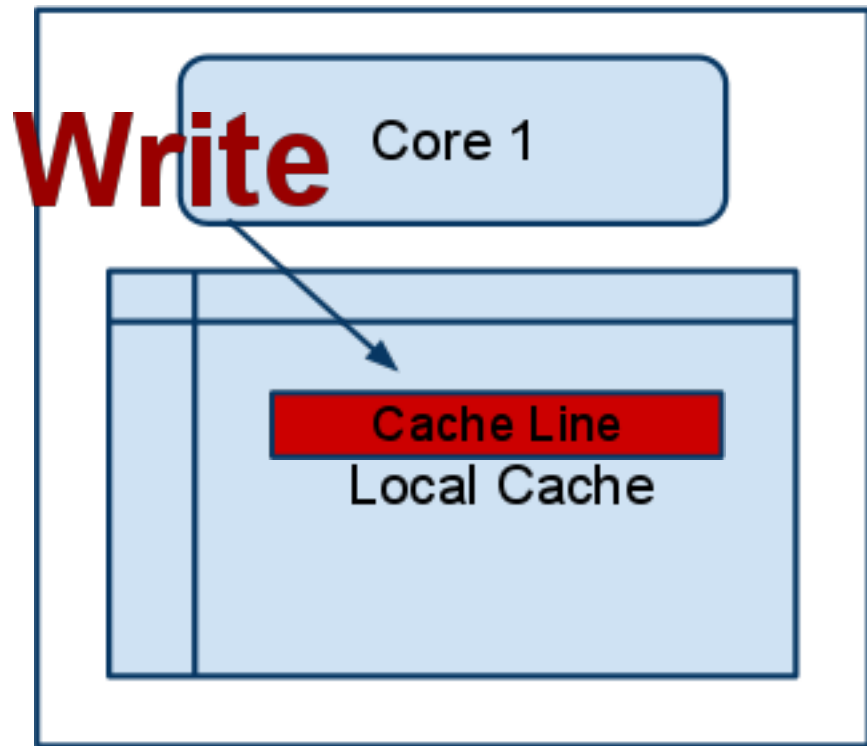
True Sharing



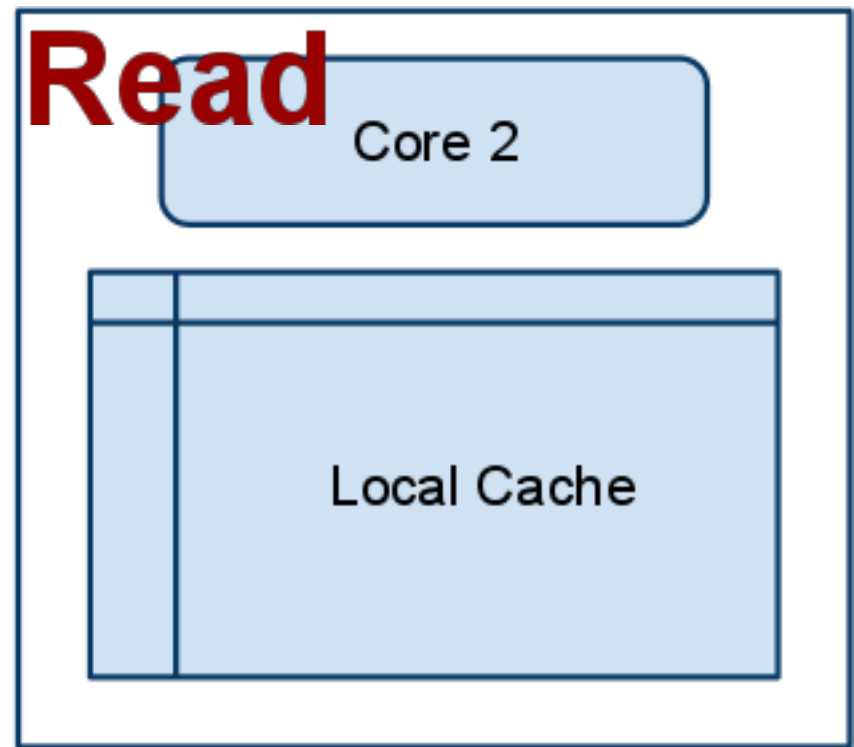
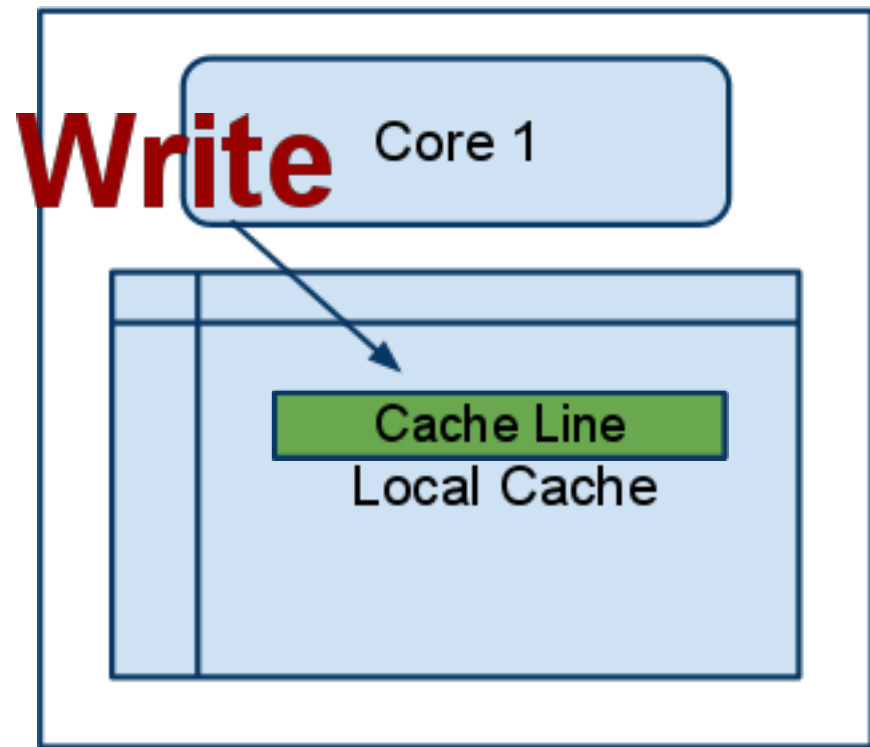
True Sharing



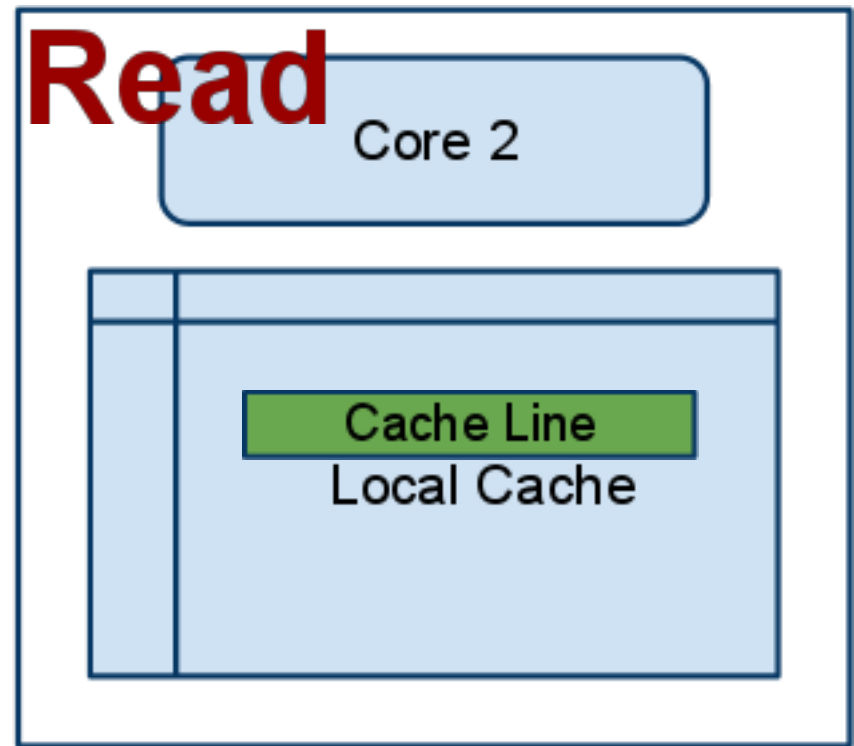
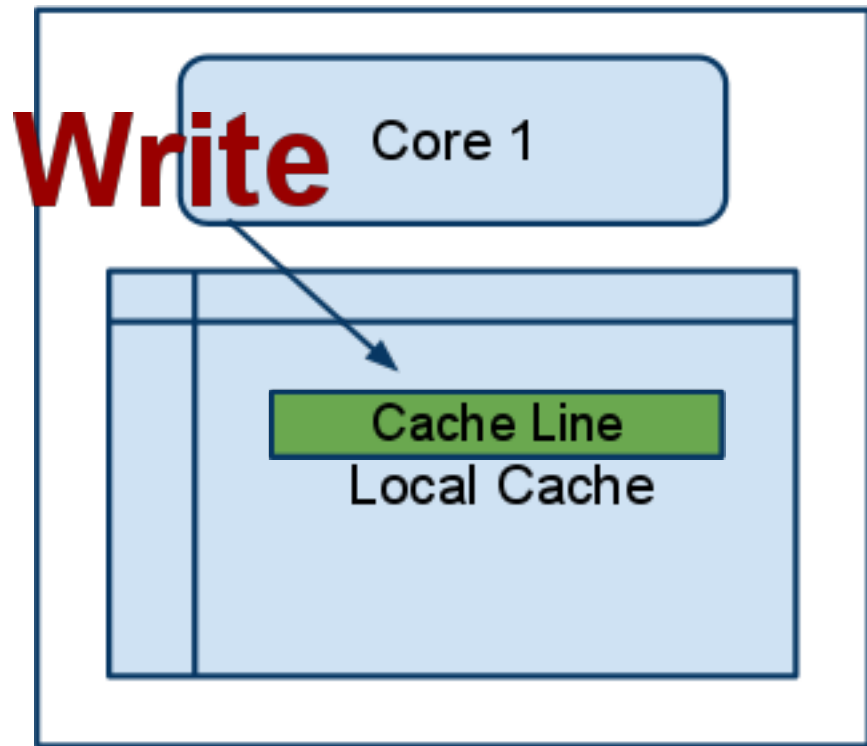
True Sharing



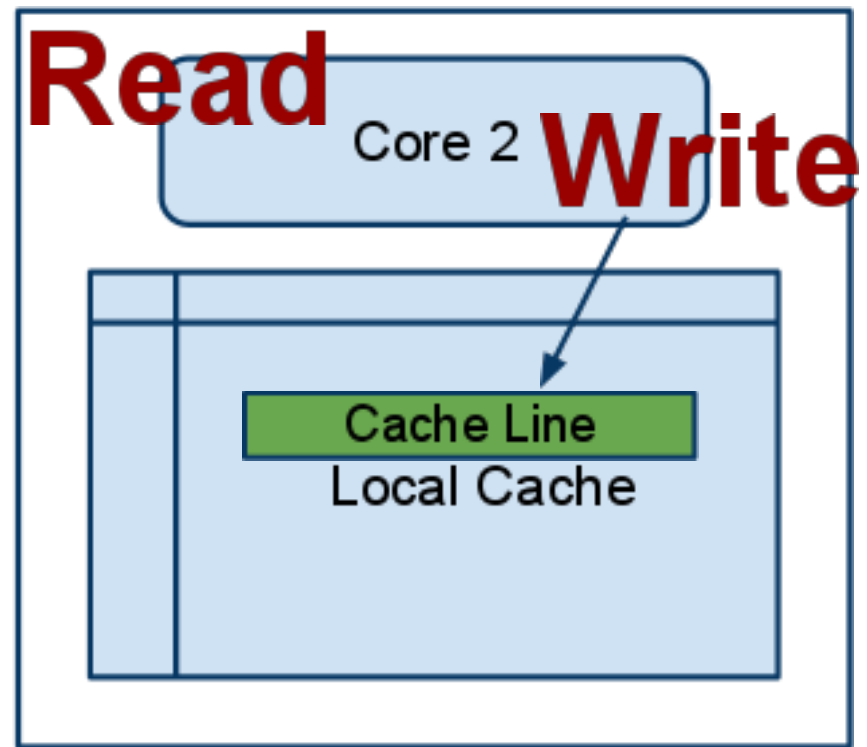
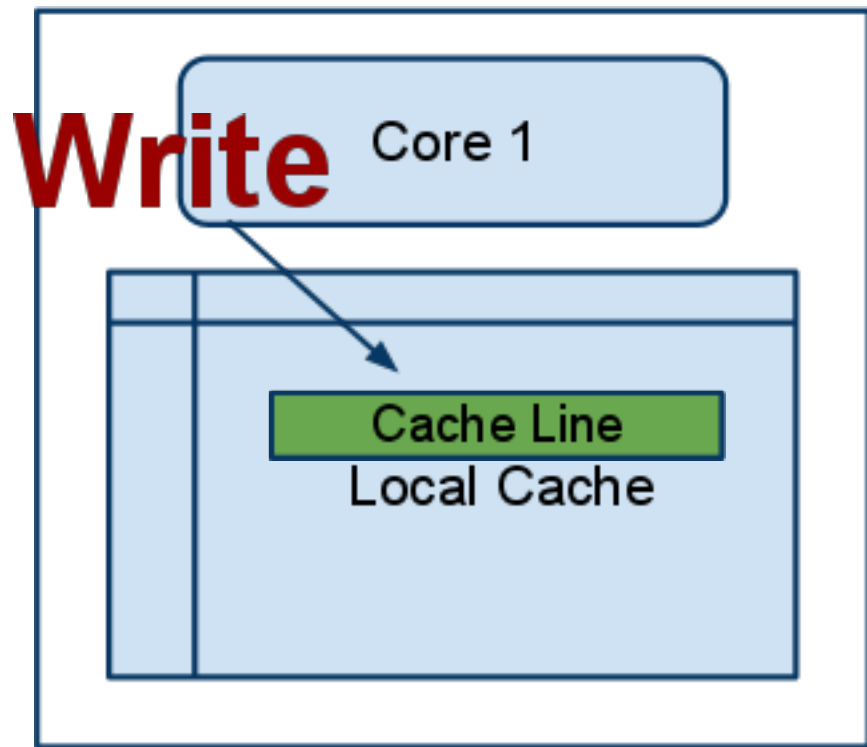
True Sharing



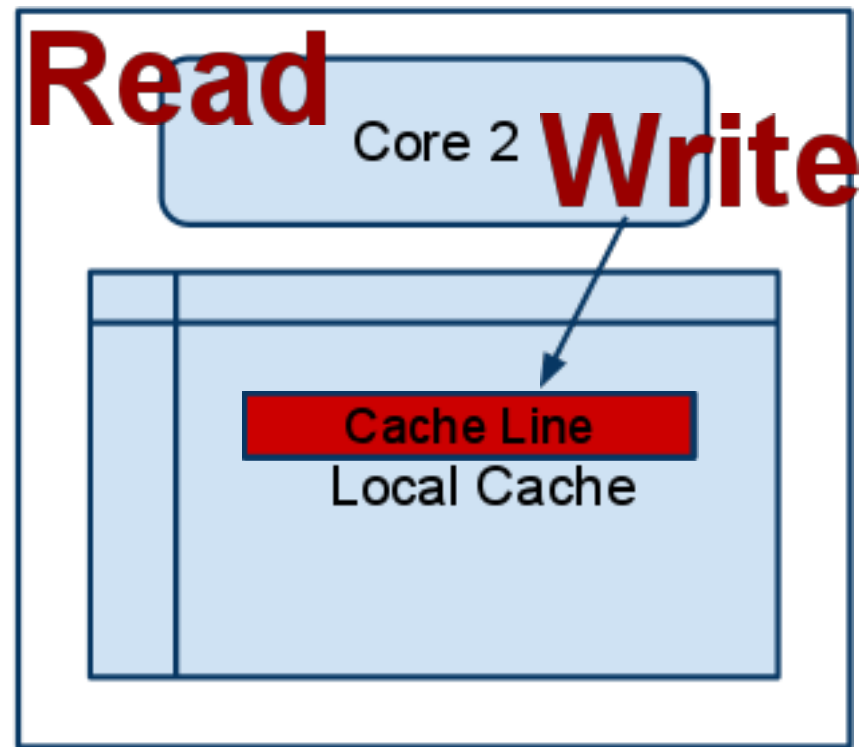
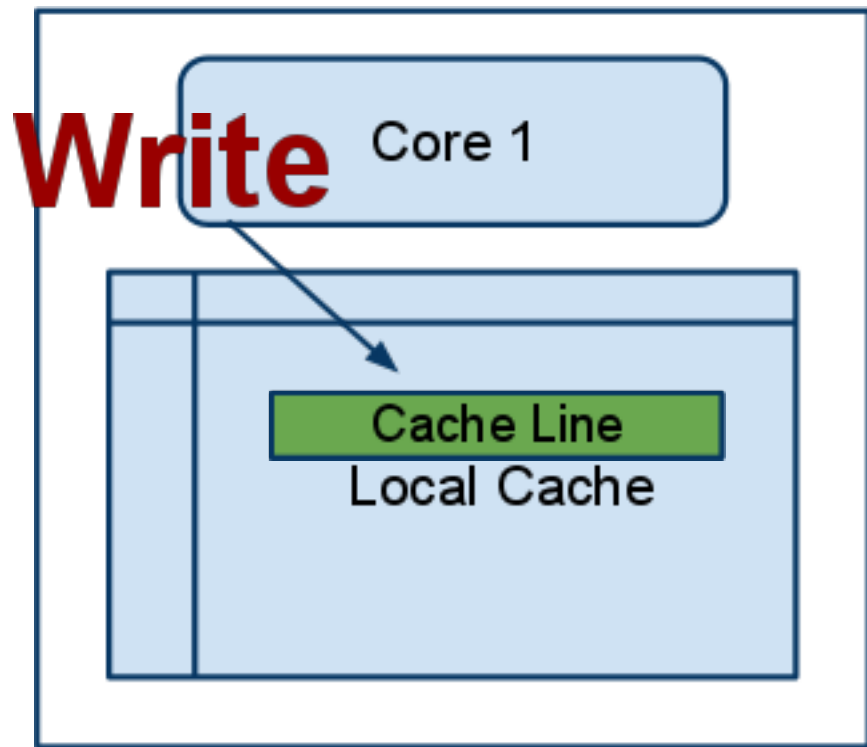
True Sharing



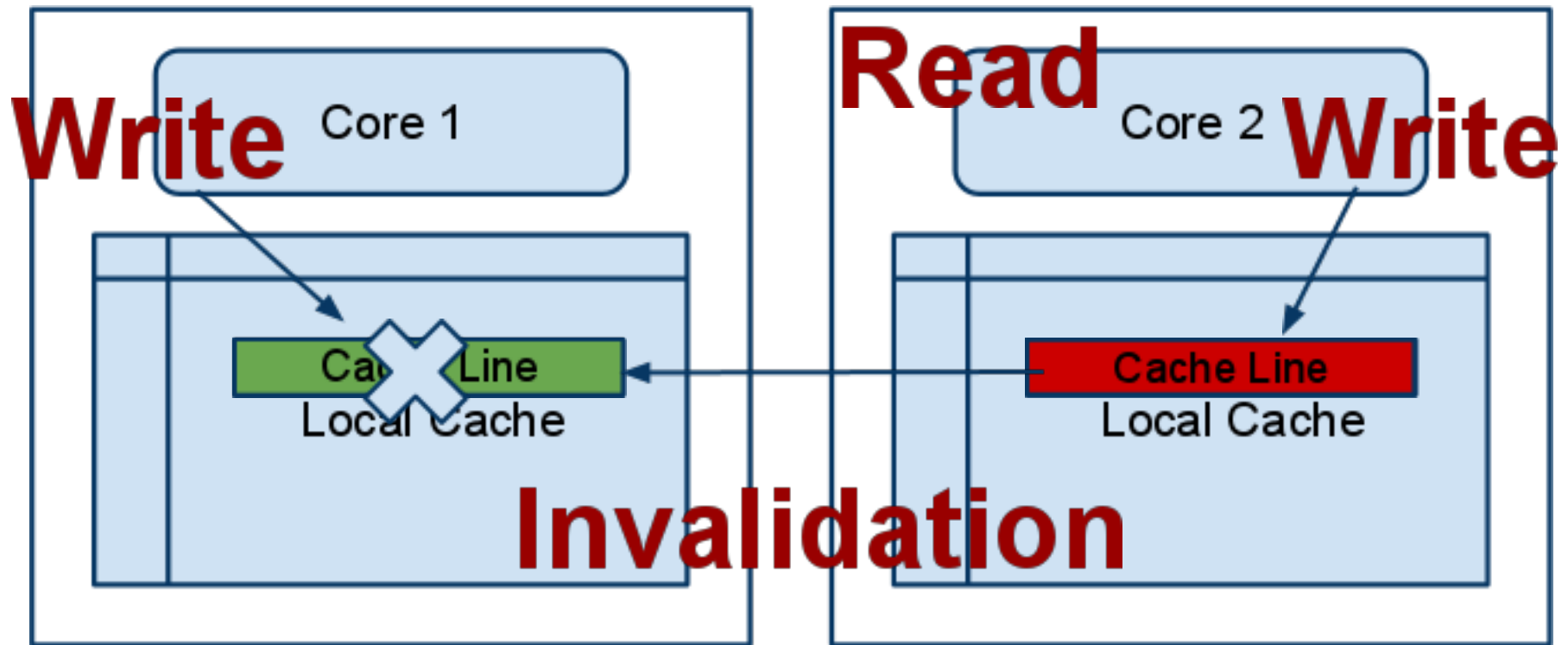
True Sharing



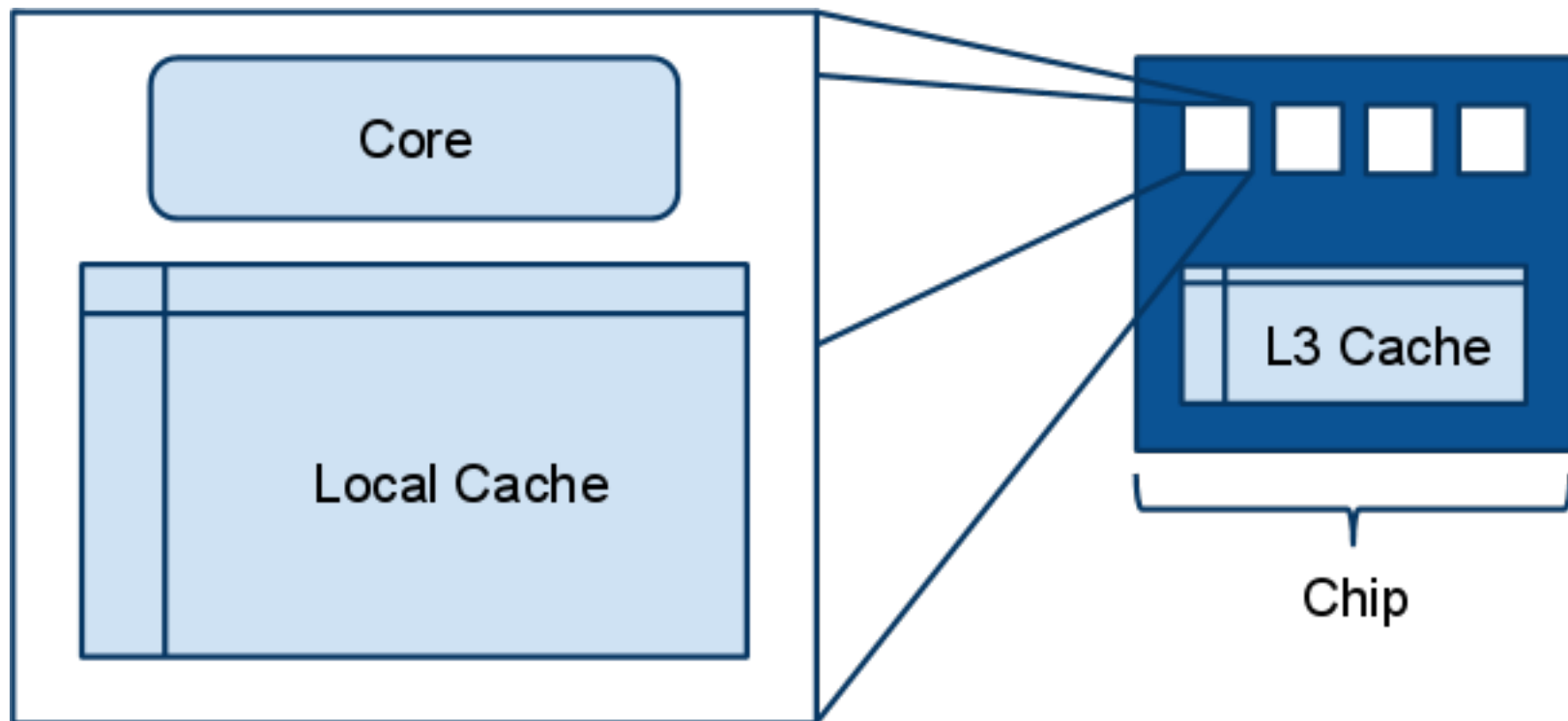
True Sharing



True Sharing



The Cost of Cache Accesses (AMD Opteron)



- Local cache accesses: 3-14 cycles
- Remote cache accesses: 50-280 cycles
- DRAM: 250-330 cycles

Understanding True Sharing Misses is Hard

- Data movement between cores is implicit.
 - Thus can't guess where misses will occur by looking at code.
- Accesses to distinct parts of a data structure happen in different functions.
 - Thus can't detect the full extent of true sharing by looking at any one function.

Contributions: DProf

- Data Profile: Aggregate cache misses by data type to bring misses to programmers attention.
- Access Trace: Show how a program accesses a data type over time.

DProf Data Profile for Linux

Type Name	% of L3 Misses	% Bounce	Count	Size
slab	32%	90%	40,000	2.5MB
udp_sock	23%	90%	16	11KB
size-1024	14%	90%	140,000	20MB
net_device	12%	90%	1	5KB
skbuff	12%	90%	160,000	34MB
ixgbe_tx_ring	1.7%	0%	16	16.KB
socket_alloc	1.7%	90%	20	2.3KB
Qdisc	0.8%	90%	16	3KB
array_cache	0.4%	90%	16	3KB

DProf Data Profile for Linux

Miss Profile

Type Name	% of L3 Misses	% Bounce	Count	Size
slab	32%	90%	40,000	2.5MB
udp_sock	23%	90%	16	11KB
size-1024	14%	90%	140,000	20MB
net_device	12%	90%	1	5KB
skbuff	12%	90%	160,000	34MB
ixgbe_tx_ring	1.7%	0%	16	16.KB
socket_alloc	1.7%	90%	20	2.3KB
Qdisc	0.8%	90%	16	3KB
array_cache	0.4%	90%	16	3KB

DProf Data Profile for Linux

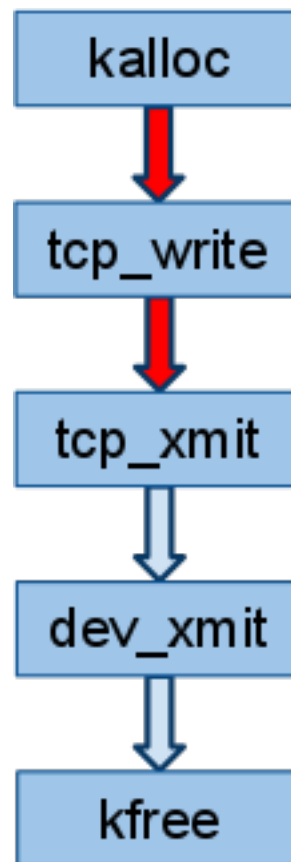
Miss Profile

Working Set

Type Name	% of L3 Misses	% Bounce	Count	Size
slab	32%	90%	40,000	2.5MB
udp_sock	23%	90%	16	11KB
size-1024	14%	90%	140,000	20MB
net_device	12%	90%	1	5KB
skbuff	12%	90%	160,000	34MB
ixgbe_tx_ring	1.7%	0%	16	16.KB
socket_alloc	1.7%	90%	20	2.3KB
Qdisc	0.8%	90%	16	3KB
array_cache	0.4%	90%	16	3KB

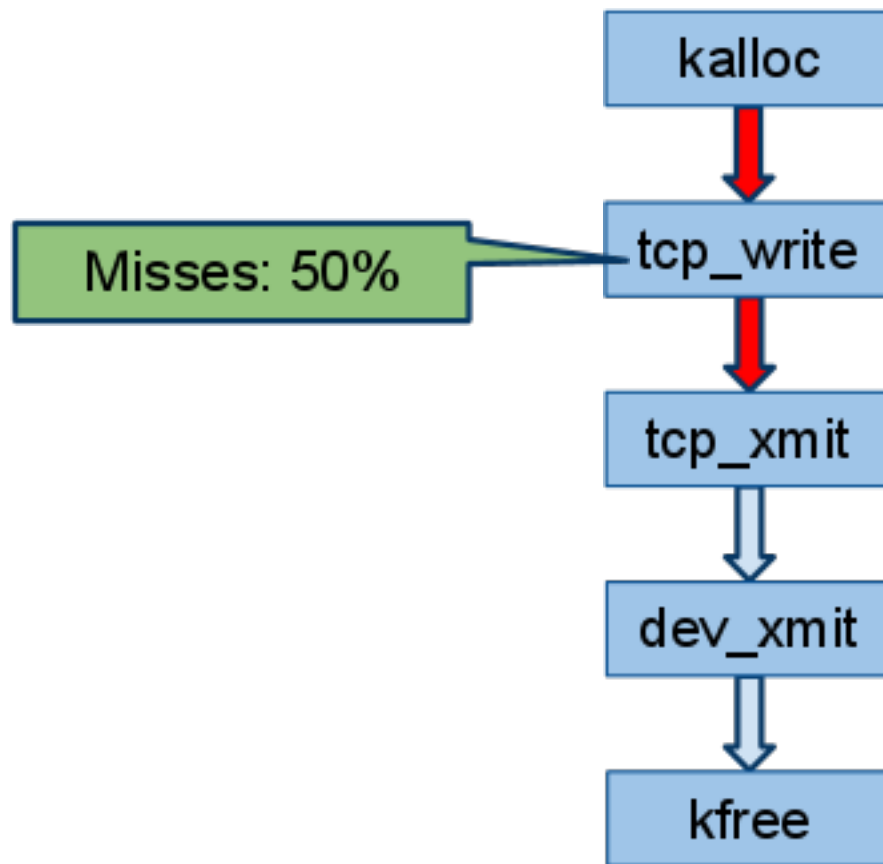
Access Trace for *tcp_sock*

- Sequence of functions that access one data type.
- Aggregated over many instances that share the same trace.



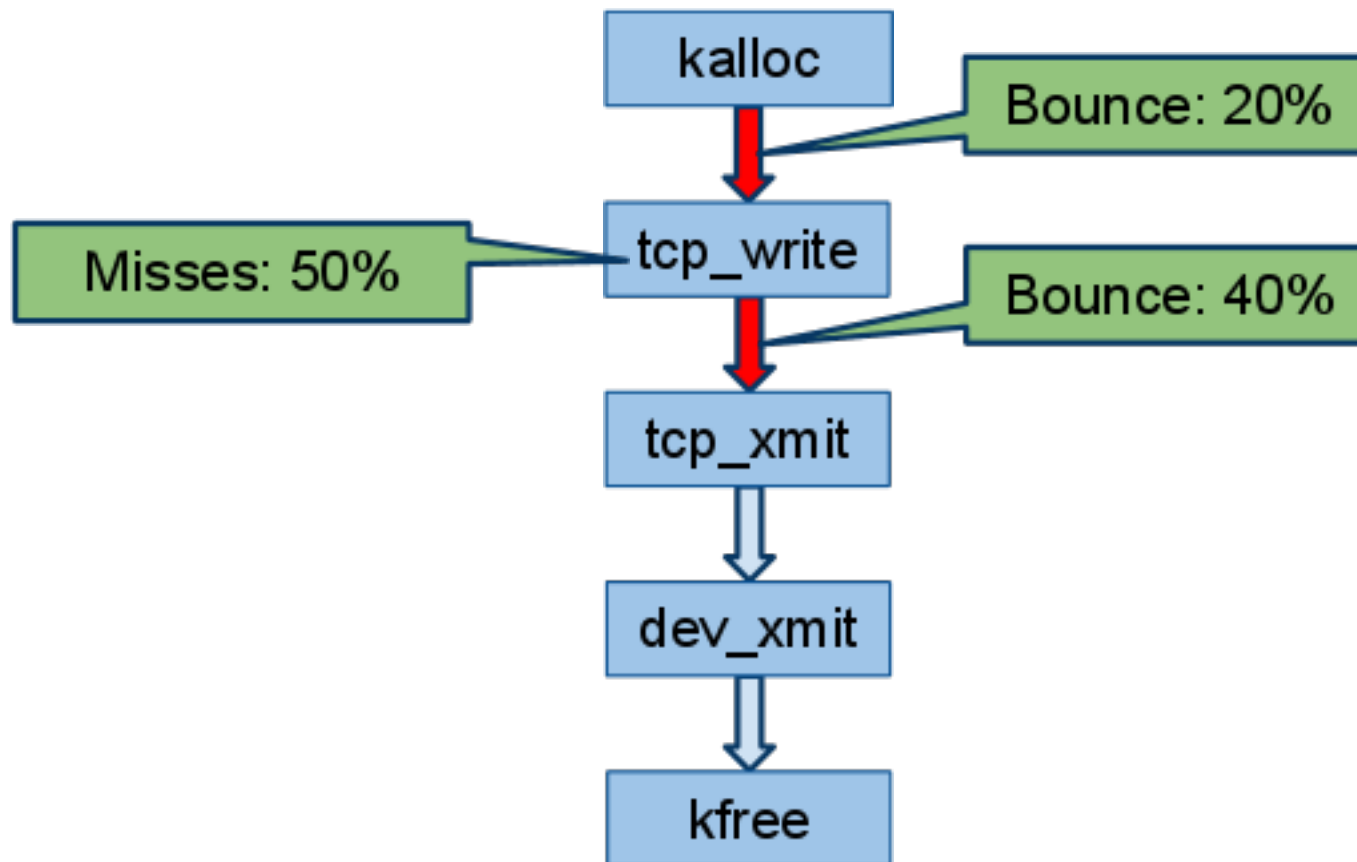
Access Trace for *tcp_sock*

- Sequence of functions that access one data type.
- Aggregated over many instances that share the same trace.



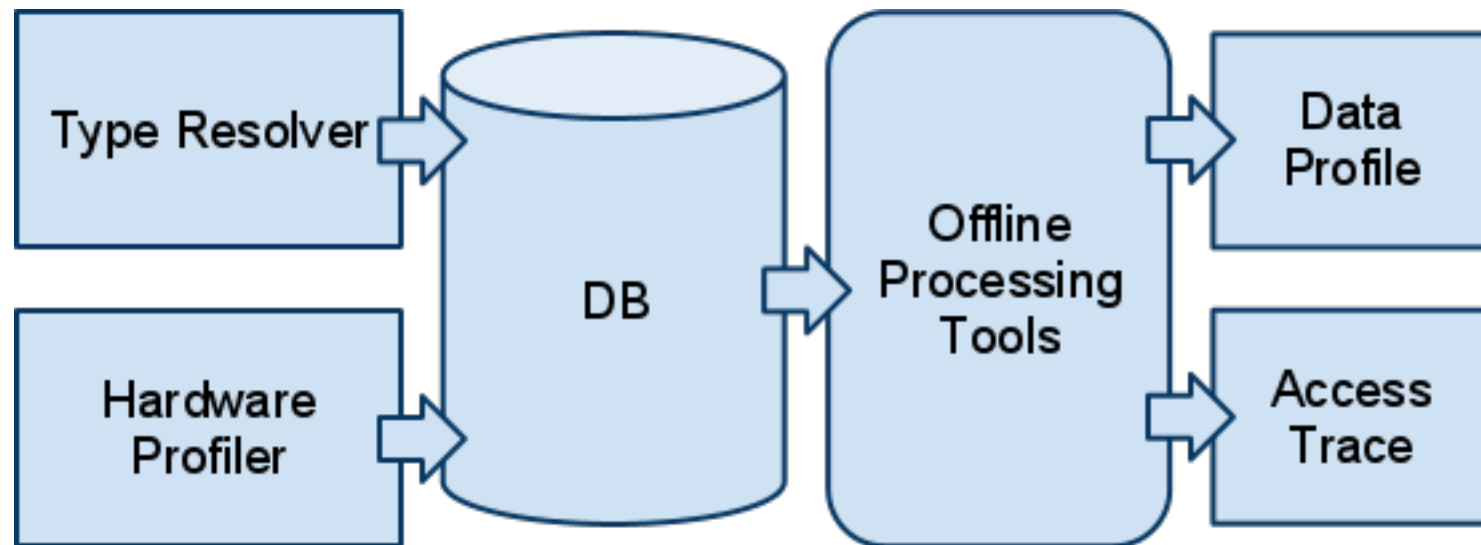
Access Trace for *tcp_sock*

- Sequence of functions that access one data type.
- Aggregated over many instances that share the same trace.

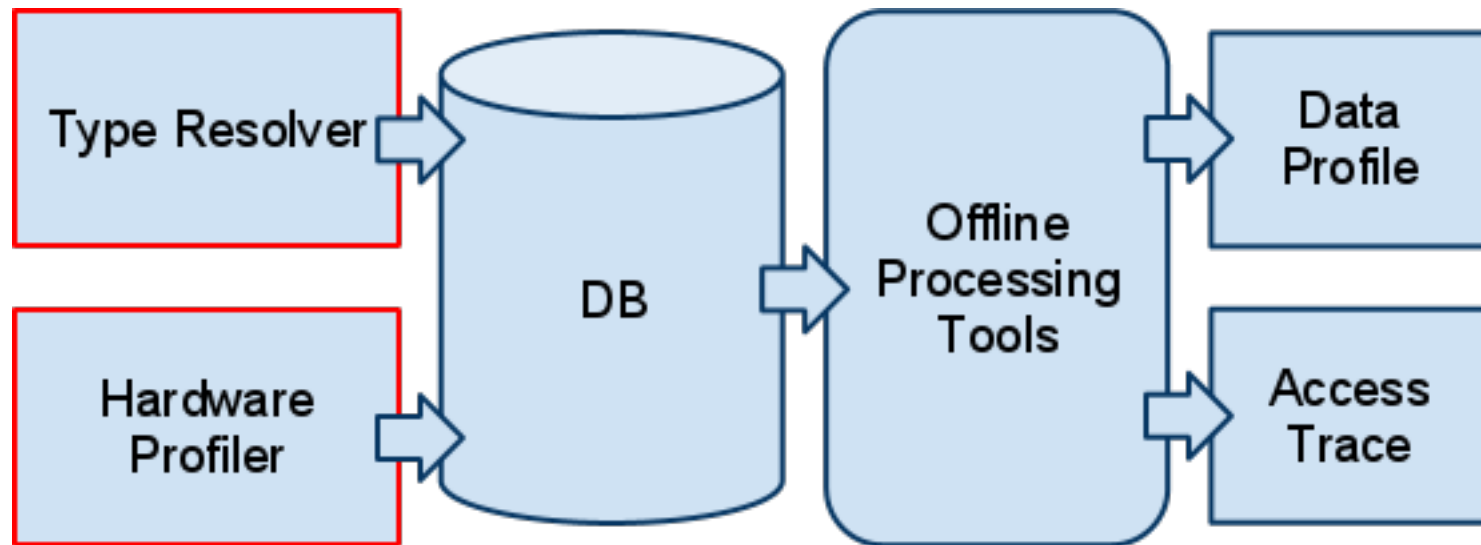


DProf's Overall Design

- Statistical profiler.
- Runs along side the program to be profiled.
- Our prototype profiles the Linux kernel.



Hardware Profiler



Ideal Hardware Support

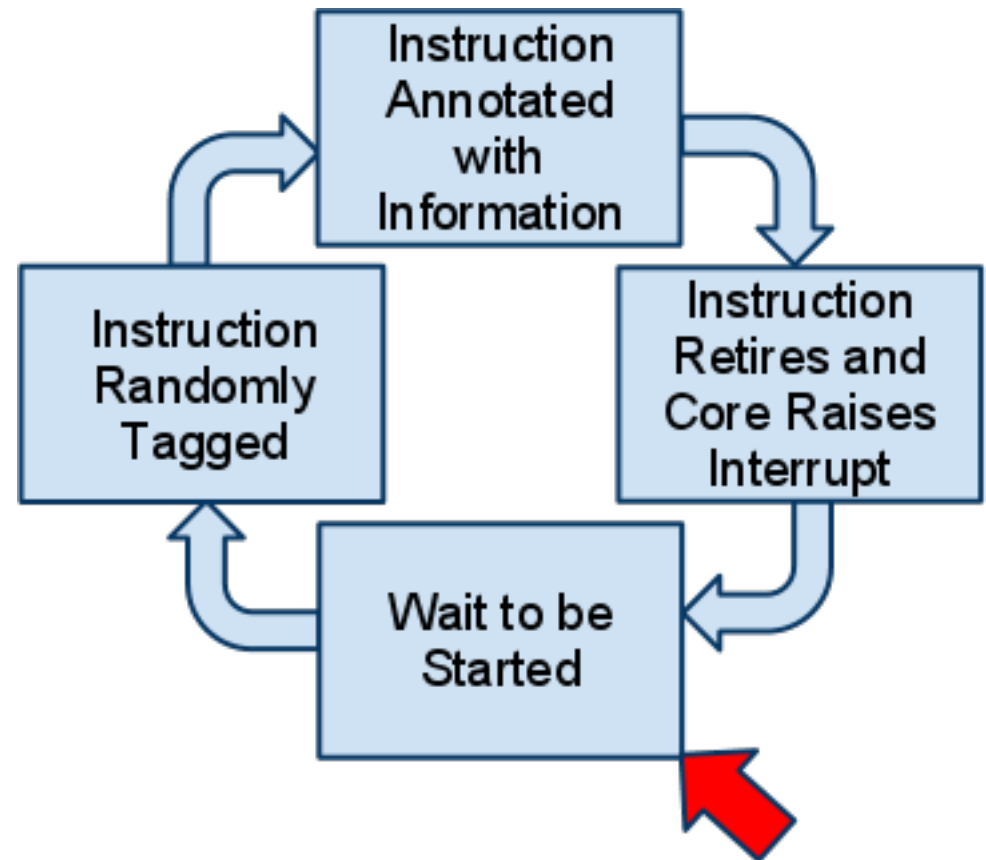
- The sequence of program counters that access a particular object.
 - Information about the access.

What the Hardware/OS Provides

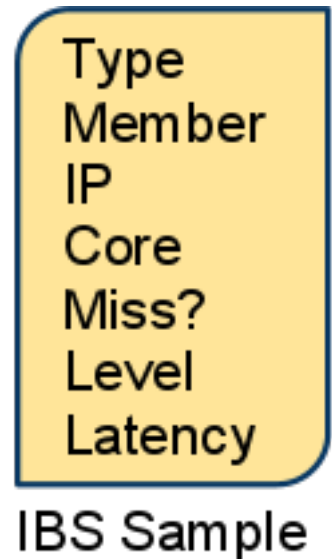
- Hardware: AMD Instruction Based Sampling (IBS)
 - Information about a randomly chosen accesses.
 - Accesses lack order.
- Hardware: DR Debug Registers
 - Gives sequence of instruction pointers that access a range of bytes.
- OS: Linux memory allocator

AMD Instruction Based Sampling

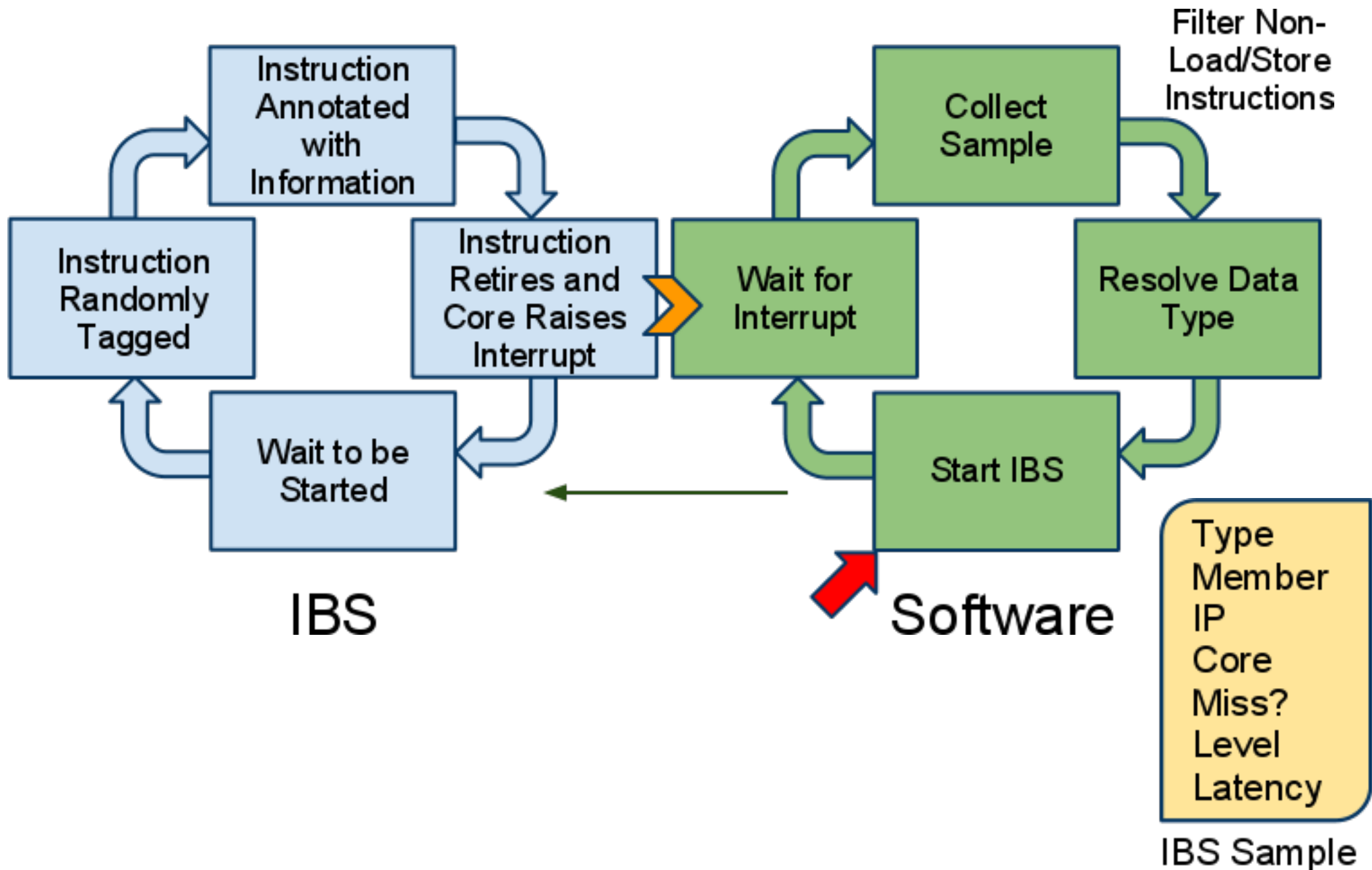
- Track one randomly chosen instruction.
- Information collected:
 - Instruction address
 - Load/Store
 - Data address
 - Cache miss
 - Hit in which cache level
 - Cache miss latency



Using IBS to Get Access Samples

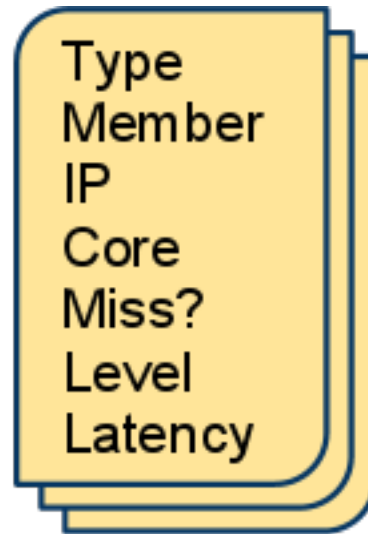


Using IBS to Get Access Samples



IBS Output

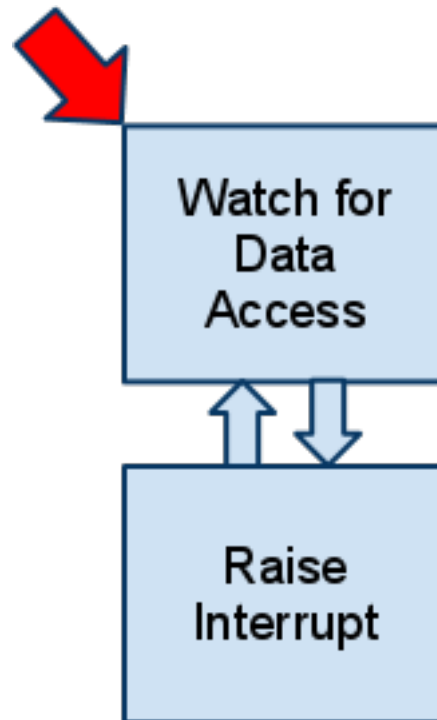
- Random load/store instruction pointers.
- No sequence information.



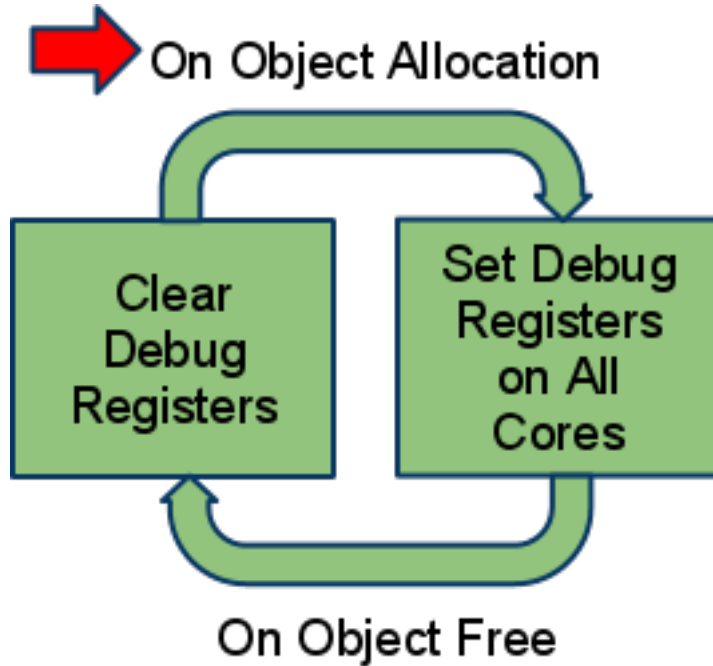
IBS Samples

DR Debug Registers

- Available on AMD and Intel processors.
- Set range of bytes.
- Interrupt on each access to bytes.

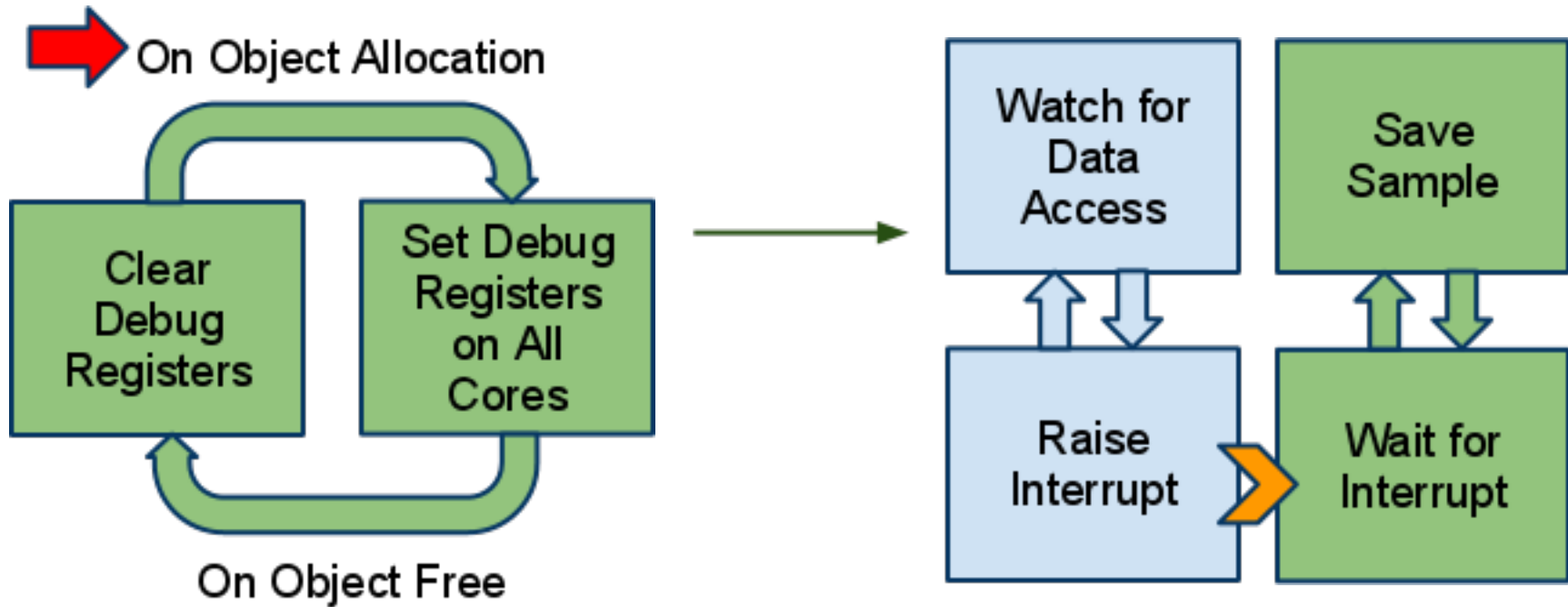


Using DR Debug Registers To Get Traces



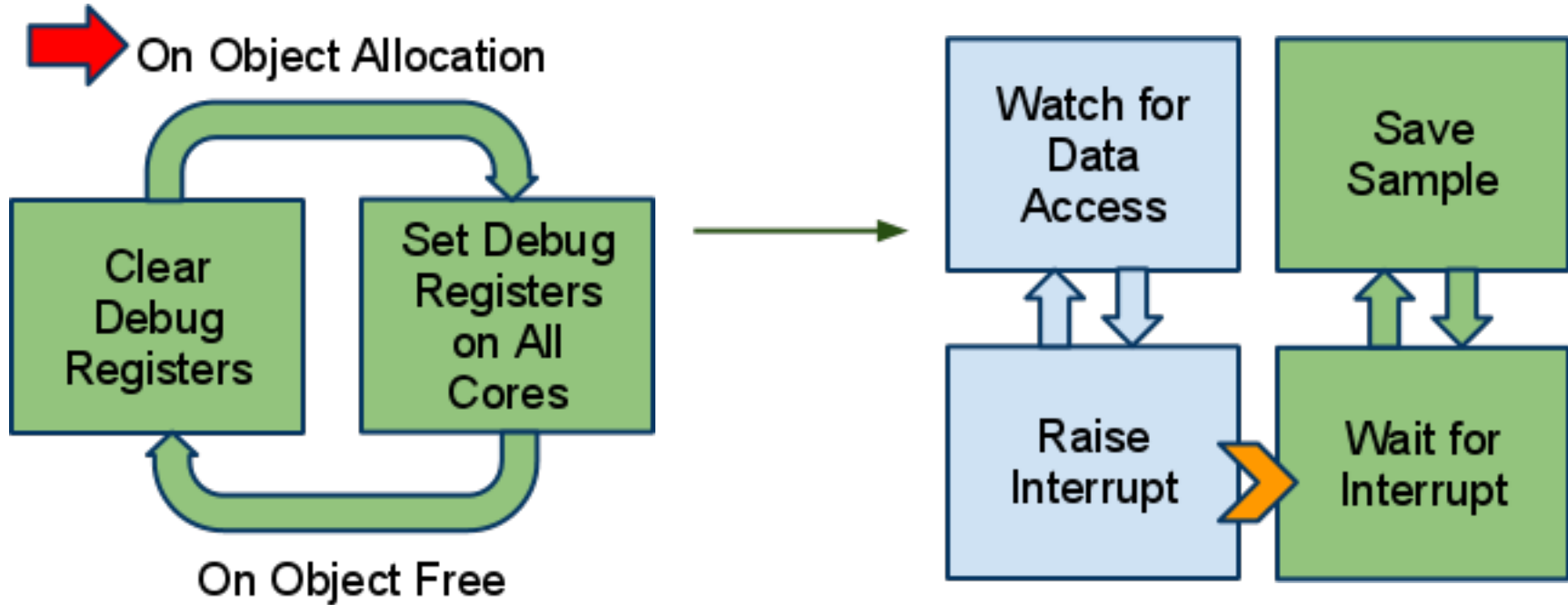
In Memory Allocator

Using DR Debug Registers To Get Traces



In Memory Allocator

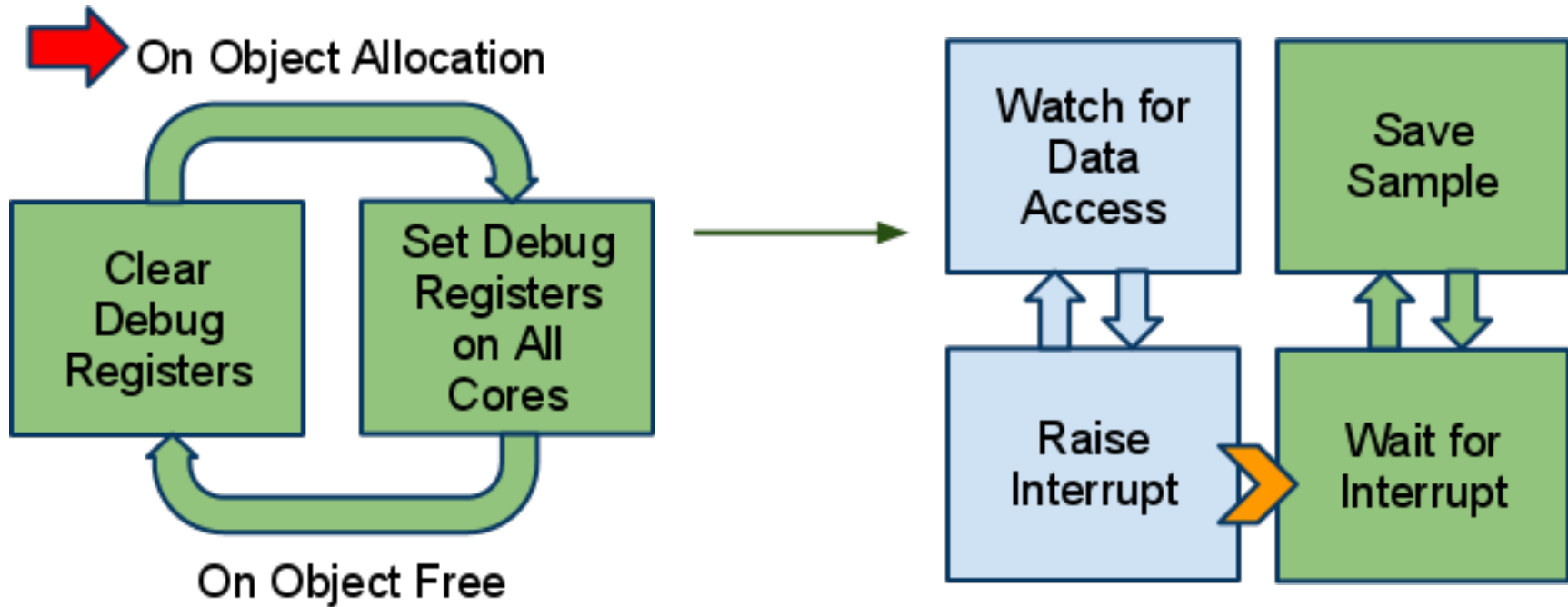
Using DR Debug Registers To Get Traces



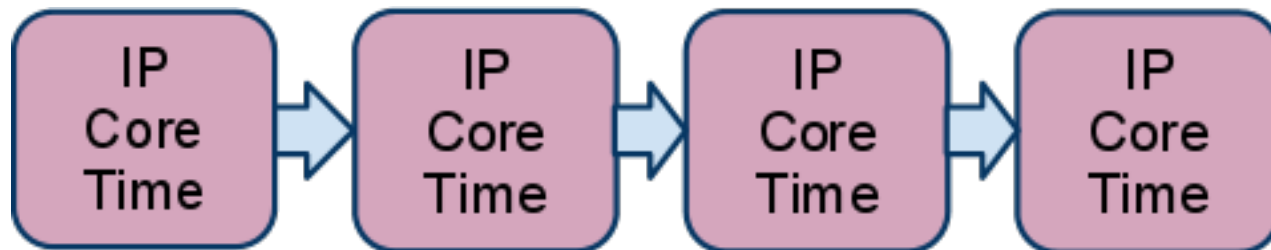
In Memory Allocator



Using DR Debug Registers To Get Traces

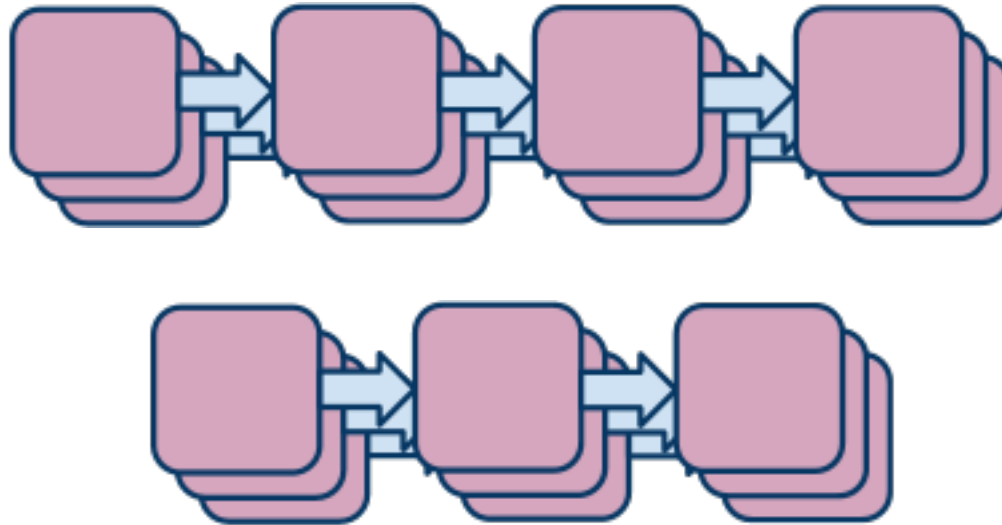


In Memory Allocator



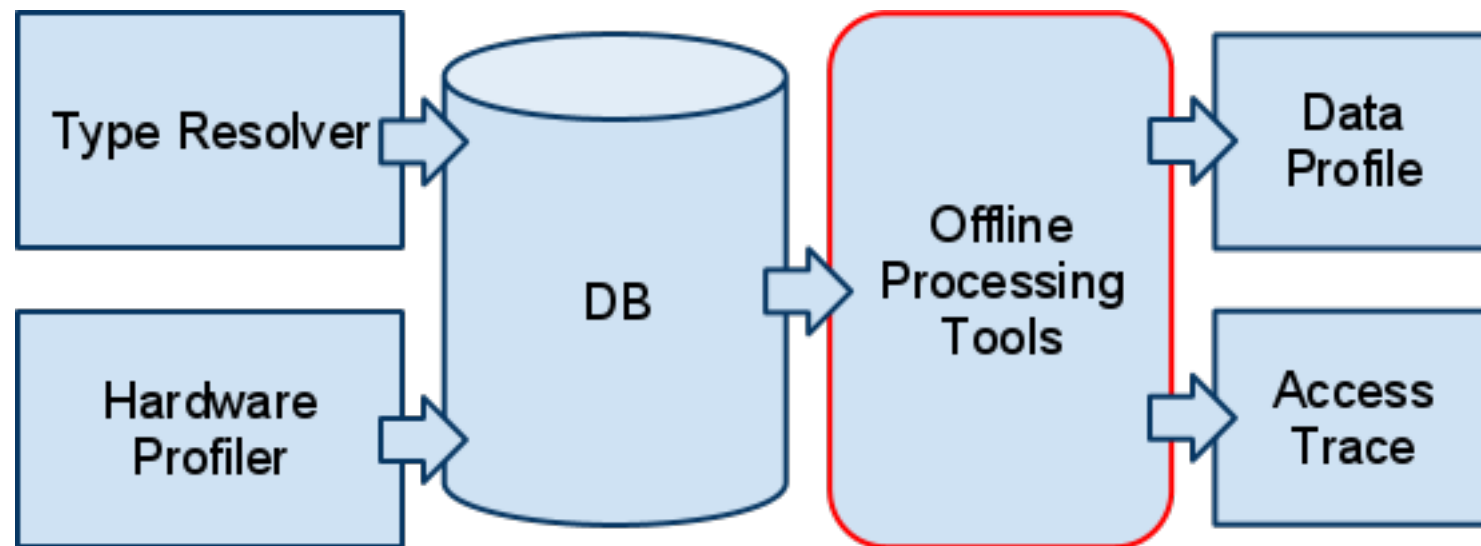
DR Debug Register Output

- Collects multiple traces.
- Traces have no information about the accesses.



DR Traces

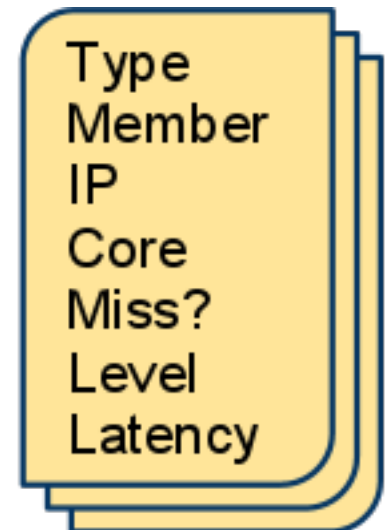
Generating Data Profile and Access Trace



Generating The Data Profile

- Aggregate IBS samples by type.

Type Name	% of L3 Misses	% Bounce
slab	32%	90%
udp_sock	23%	90%
size-1024	14%	90%
net_device	12%	90%
skbuff	12%	90%
ixgbe_tx_ring	1.7%	0%



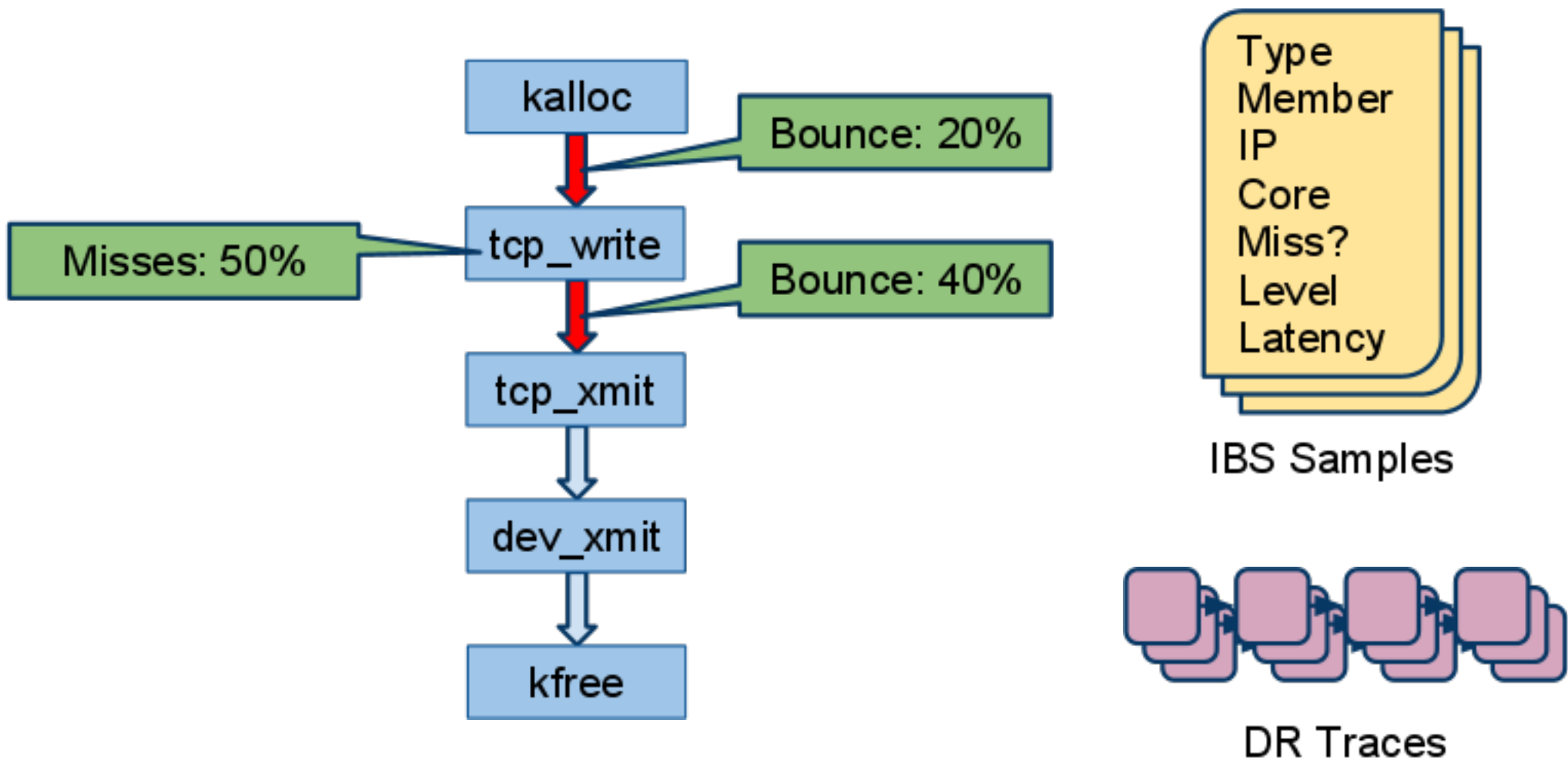
IBS Samples



DR Traces

Generating an Access Trace

- Join DR traces with IBS samples.

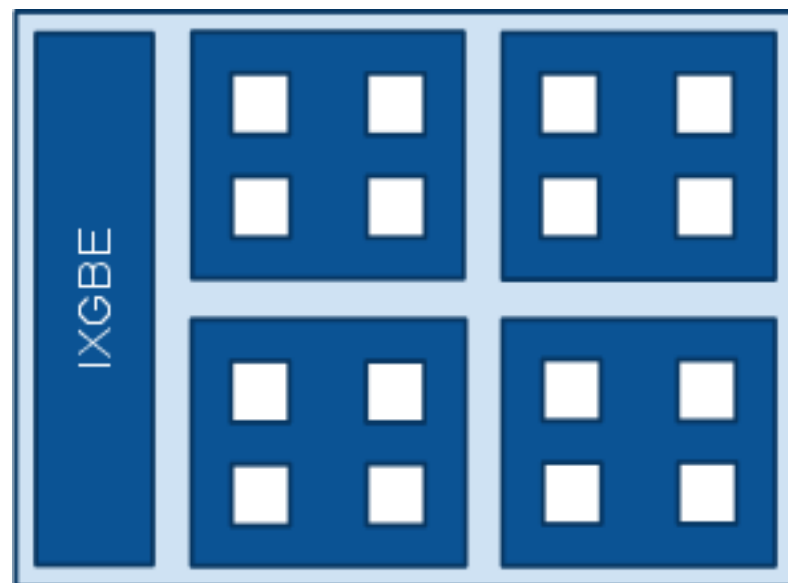


Case Study of True Sharing in The Linux Kernel

- Run Memcached to generate load for the Linux kernel.
 - In memory key-value store.
 - Can be used to create a distributed key-value store.
 - Multiple memcached servers.
 - Keys deterministically distributed among servers.
- Kernel suffers from cache misses due to true sharing.

Experiment Setup

- Test machine: 16 core AMD Opteron machine.
 - 4 chips, 4 cores per chip
 - Intel 10GB Ethernet card
- 16 memcached instances each pinned to one core.
 - Avoids memcached internal locking problems.
- NIC routes received packets to core that will process the packet.
 - 16 RX and 16 TX queues
- Clients query one non-existent key over UDP.



Memcached Performance

- Over 80% of the time spent in the kernel.

	1 Core	16 Cores
Requests per Second per Core	140,000	70,000

Locks

Lock Name	Overhead
epoll	0.14%
wait queue	0.12%
Qdisc	0.25%
SLAB cache	0.01%

Lock usage statistics.

OProfile Report for Linux Kernel

% CLK	% L2	Function Name
5.1	9.7	ixgbe_clean_rx_irq
3.2	7.5	__alloc_skb
3.2	6.6	kmem_cache_free
4.4	6.3	kfree
2.6	3.8	ixgbe_xmit_frame
1.3	3.7	gart_unmap_page
2.8	3.6	ixgbe_clean_tx_irq
1.2	3.5	skb_dma_map
1.5	2.4	dev_kfree_skb_irq
1.0	2.4	ixgbe_unmap
1.0	2.4	sock_def_write_space
1.5	2.3	lock_sock_nested
2.1	2.1	ip_finish_output
1.8	2.1	dev_queue_xmit
1.2	2.1	__skb_recv_datagram

- Nothing suspicious.
 - Workload is processing many packets per second.

Cache Misses

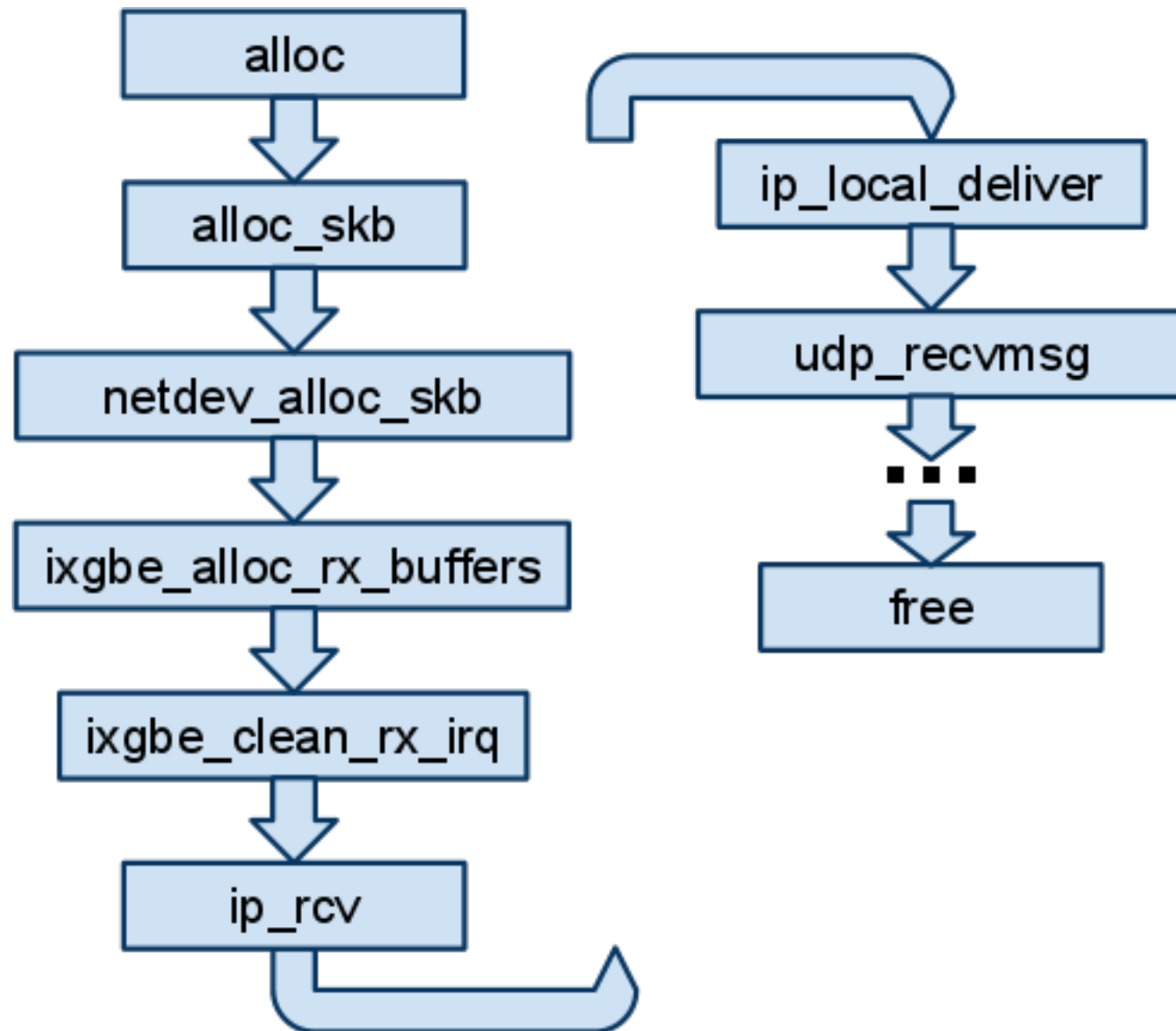
	1 Core	16 Cores
Requests per Second per Core	140,000	70,000
L2 Misses per Request	30	80
L3 Misses per Request	20	70

DProf Data Profile for the Linux Kernel Running Memcached

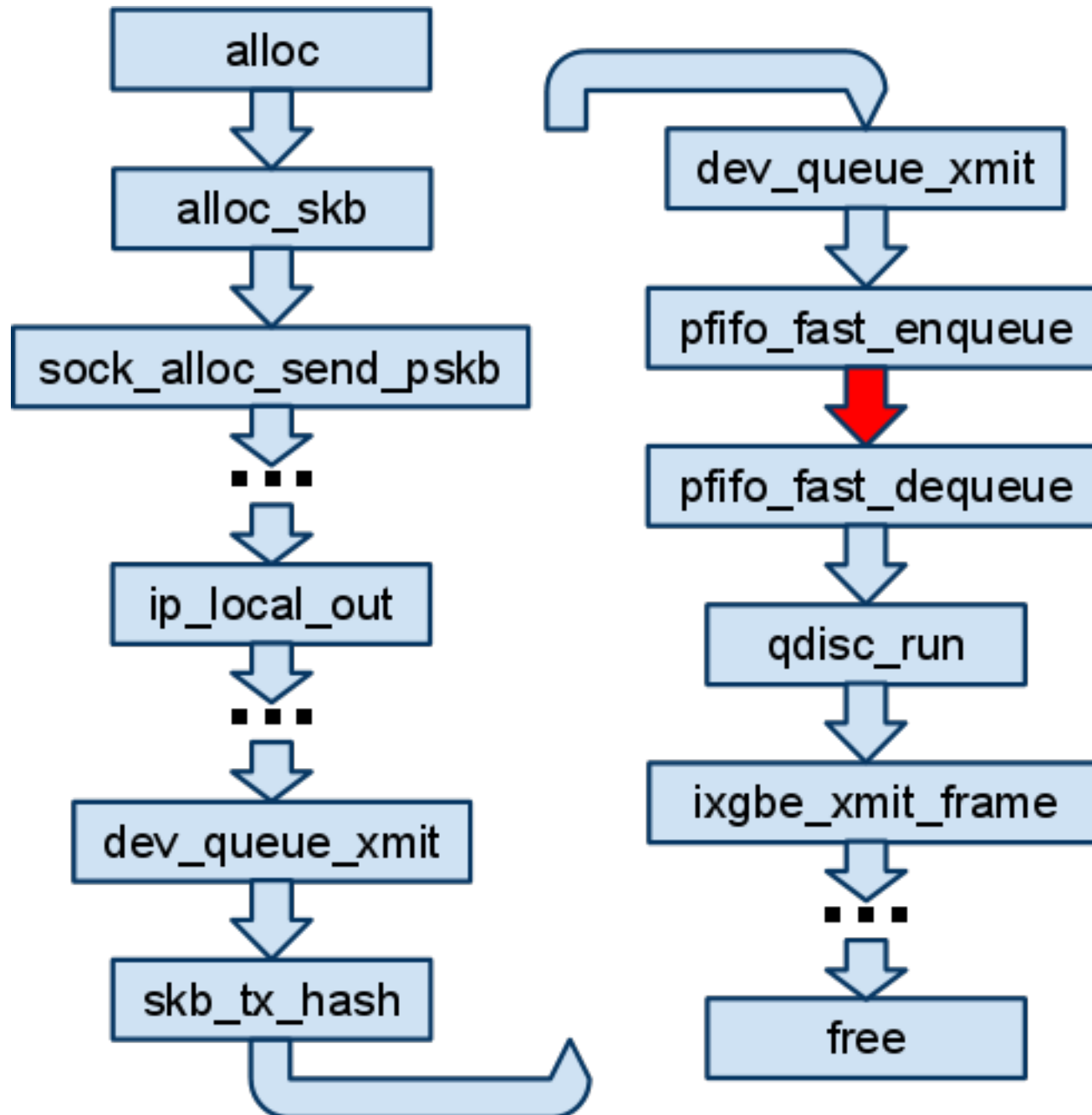
Data Profile View

Type Name	% of L3 Misses	% Bounce
slab	32%	90%
udp_sock	23%	90%
size-1024	14%	90%
net_device	12%	90%
skbuff	12%	90%
ixgbe_tx_ring	1.7%	0%
socket_alloc	1.7%	90%
Qdisc	0.8%	90%
array_cache	0.4%	90%

Access Trace for *skbuff* on RX Path



Access Trace for *skbuff* on TX Path



Memcached Performance

- Close to 60% improvement.
- Improvement comes from reduce in cache misses.
 - $14.3 \text{ us} - 9.1 \text{ us} = 5.2 \text{ us}$
 - $5.2 \text{ us} / (0.5 \text{ ns/cycle}) = 10,400 \text{ cycles}$
 - $10,400 \text{ cycles} / 40 \text{ misses} = 260 \text{ cycles/miss}$

Configuration	Hashed TX Queue	Local TX Queue
Requests per Second per Core	70,000	110,000
L2 Misses per Request	80	40
L3 Misses per Request	70	30

Overhead

- Since sampling, overhead can be adjusted.
- IBS Sampling:
 - Overhead of 6%.
 - Sampled for 120 second.
- Trace sampling (of an *skbuff* data type):
 - Overhead of 0.8%.
 - Sampled for 95 seconds.

Related Work

- Execution Profilers
 - OProfile, DCPI, Gprof, Quartz, VTune, PTU
- Cache Simulators
 - Valgrind's Cachegrind, CProf, MemSpy
- Hardware Support for Profiling
 - ProfileMe
 - FlashPoint

Conclusion

- Data cache misses affect performance.
- True sharing misses are hard to understand.
- Proposed a data profiler: DProf.
 - Data Profile
 - Access Trace

Questions?