Analyzing Cache Line Contention Using perf c2c

Andres Freund
PostgreSQL Developer & Committer
Email: andres@anarazel.de
Email: andres.freund@microsoft.com

https://anarazel.de/talks/2024-05-29-pgconf-dev-c2c/postgres-perf-c2c.pdf

Microsoft

Motivation

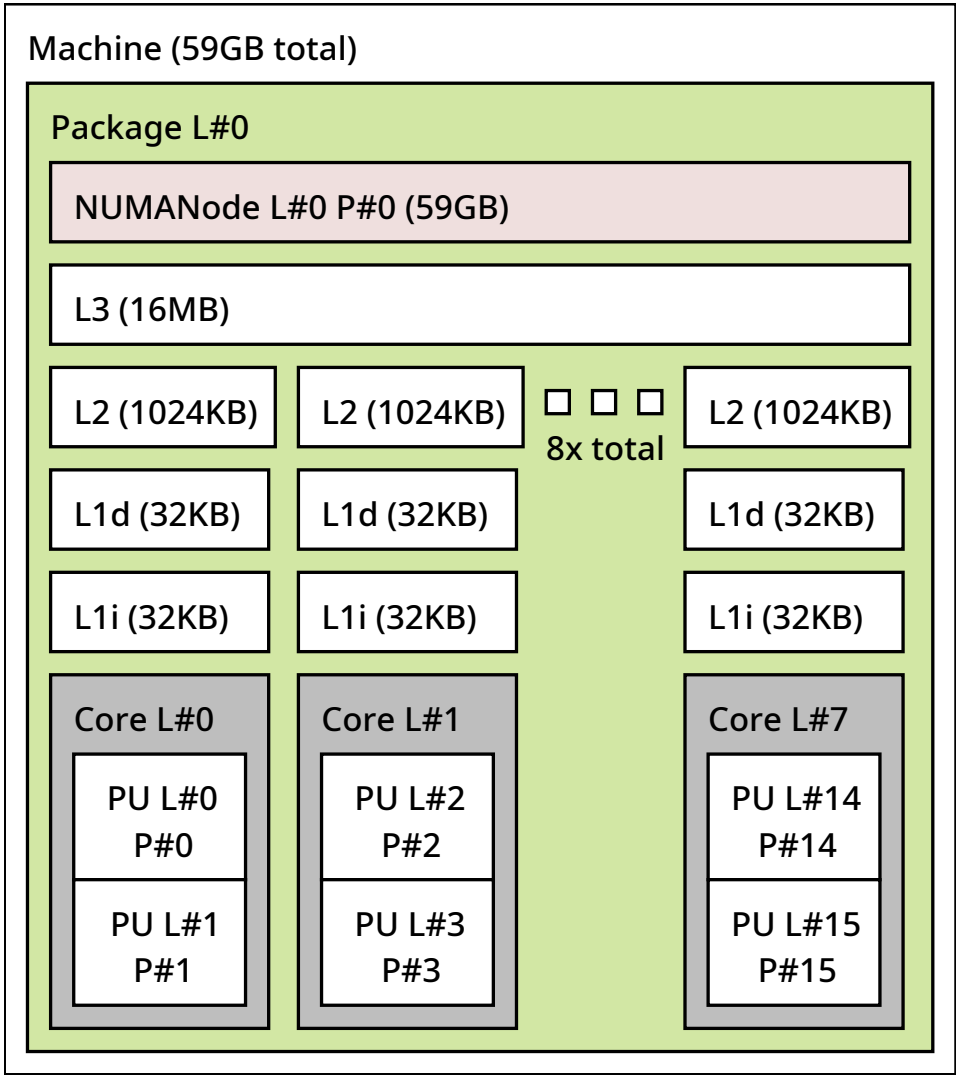
- “Snapshot scalability” work landing in PG 14
- Common question: “How did you know?”
- Help others analyze problems like this
  - False Sharing
  - Contention
- requires some hardware understanding
- perf c2c is poorly documented

Background

- Cache / Memory Hierarchy
- Cache Coherence
- Lots of simplifying (aka lying)

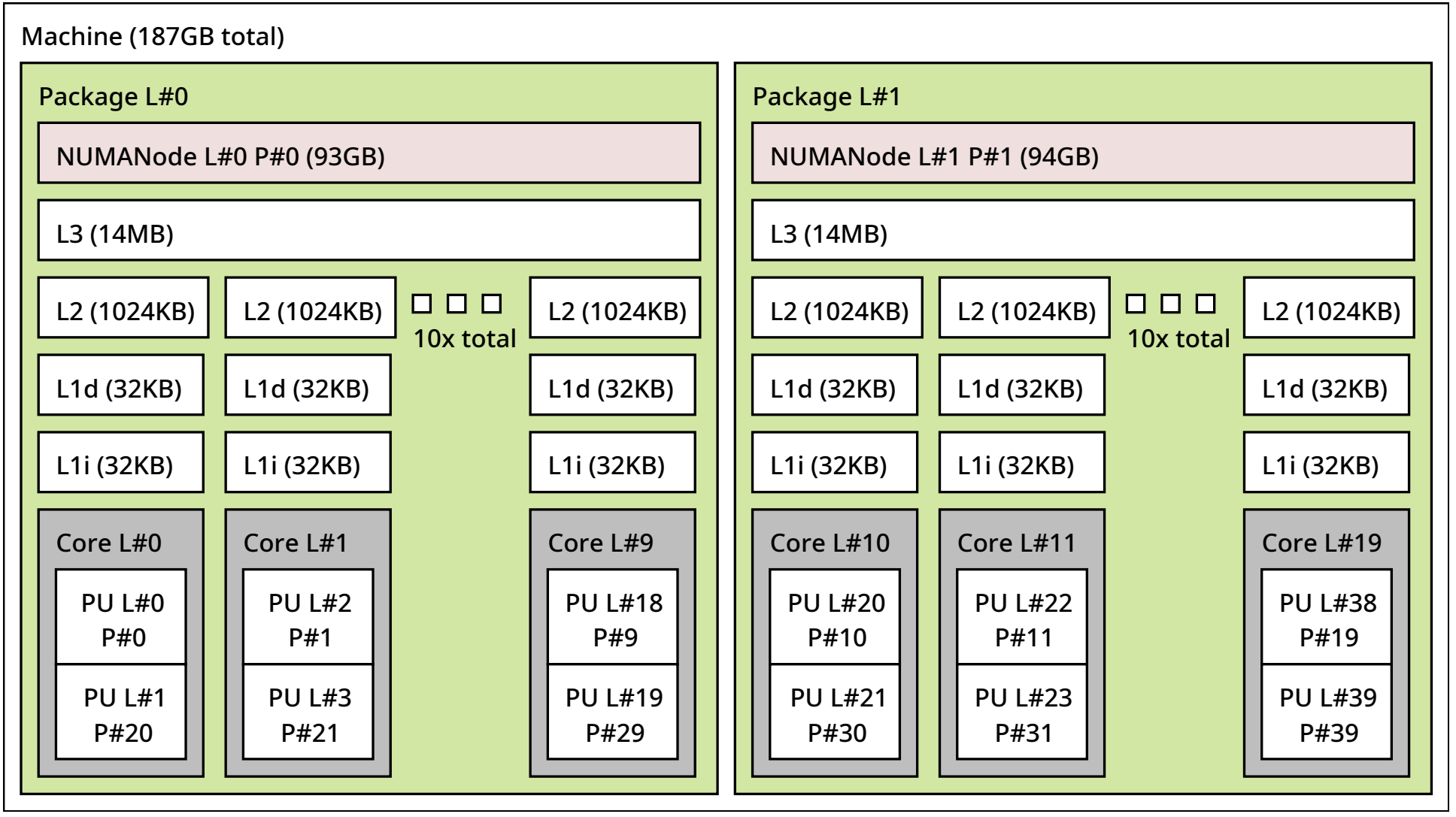
Cache Structure

- Cache Lines: Often 64 bytes
- L1i: Very fast cache for instructions (~1ns, ~32k)
- L1d: Very fast cache for data (~1ns, ~32k)
- L2: Fast cache (~3-5ns, ~1MB)
- L3: Cache, shared across cores (~8 – 20ns, ~16MB)
- Memory (~60-120ns)
- Other details like TLB, store buffer, uop cache, ...



Hierarchy of my laptop, generated by lstopo

Microsoft



Hierarchy of my workstation, generated by lstopo

Microsoft

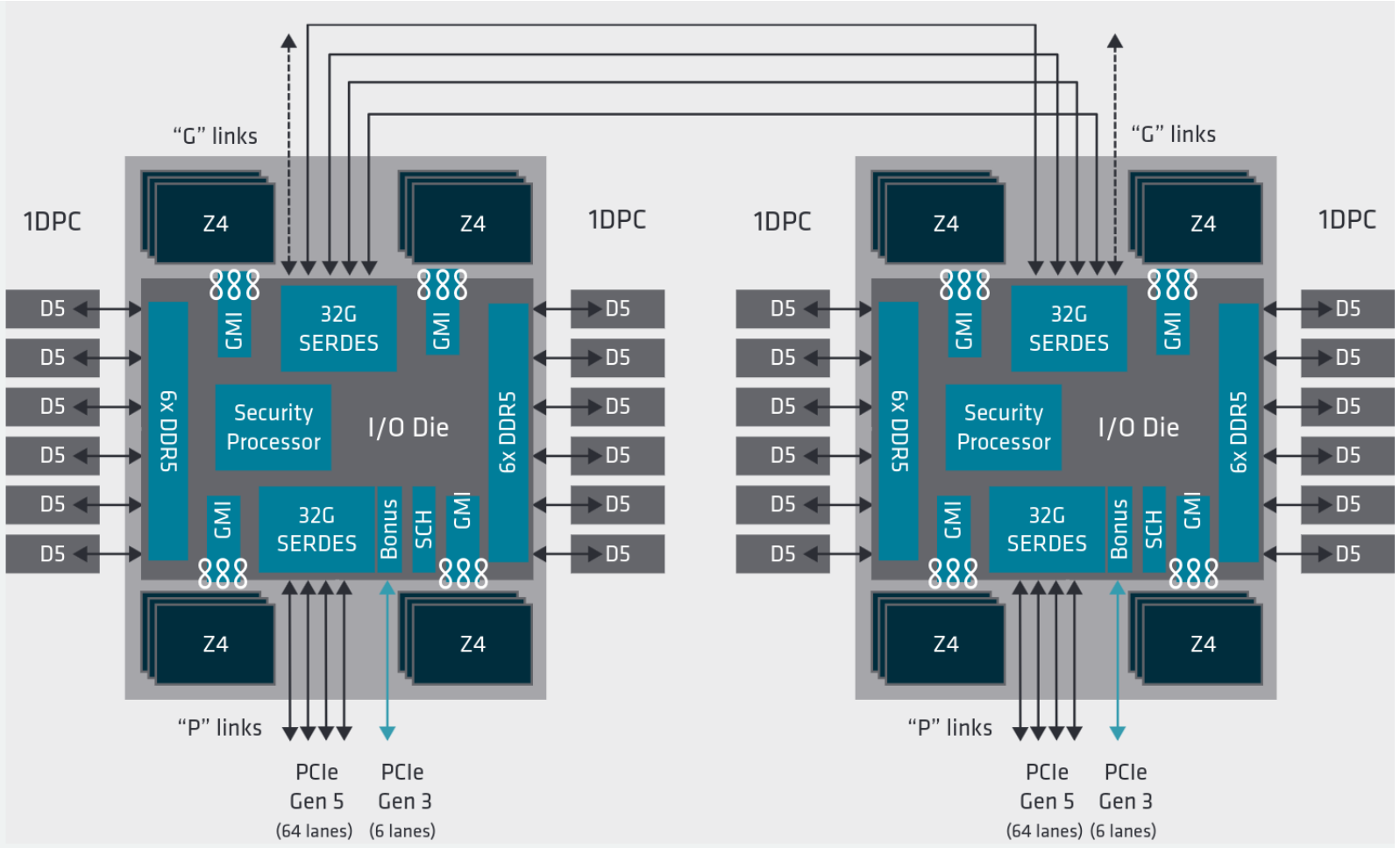


Figure 8: EPYC 9004 Series processors in a 2-socket configurations

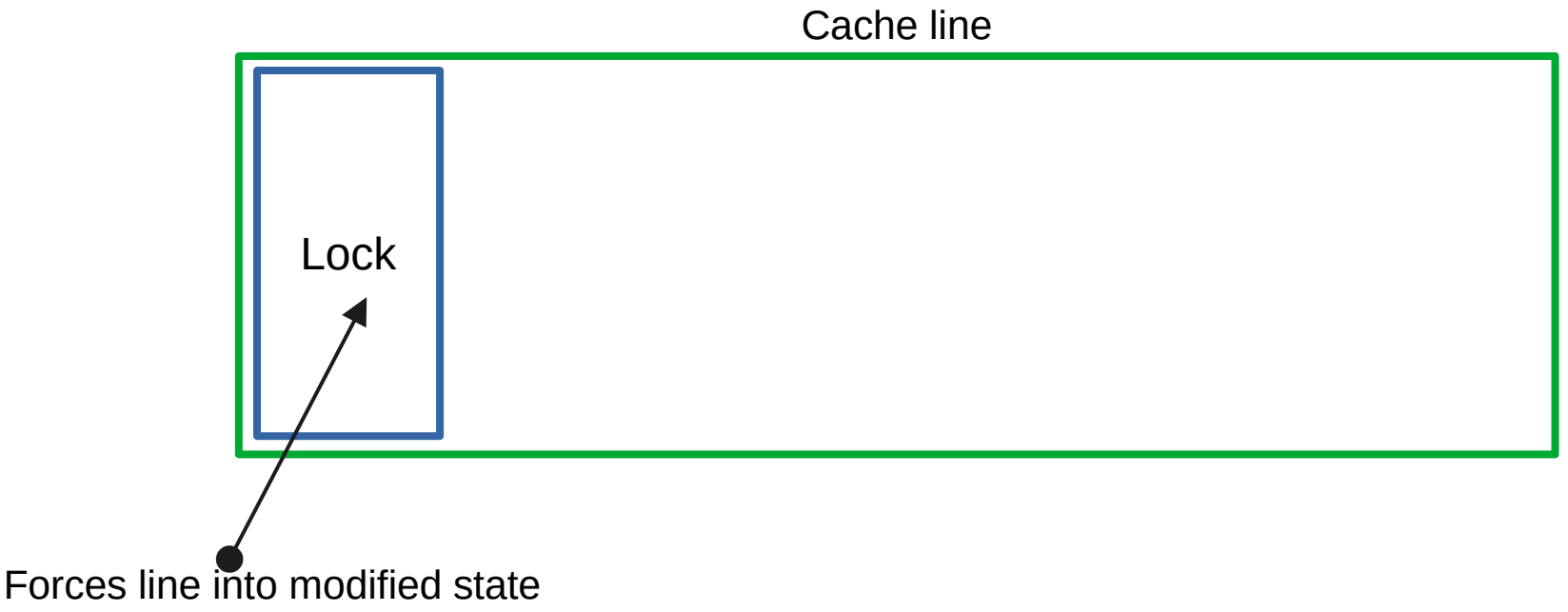
https://www.amd.com/system/files/documents/4th-gen-epyc-processor-architecture-white-paper.pdf

Microsoft

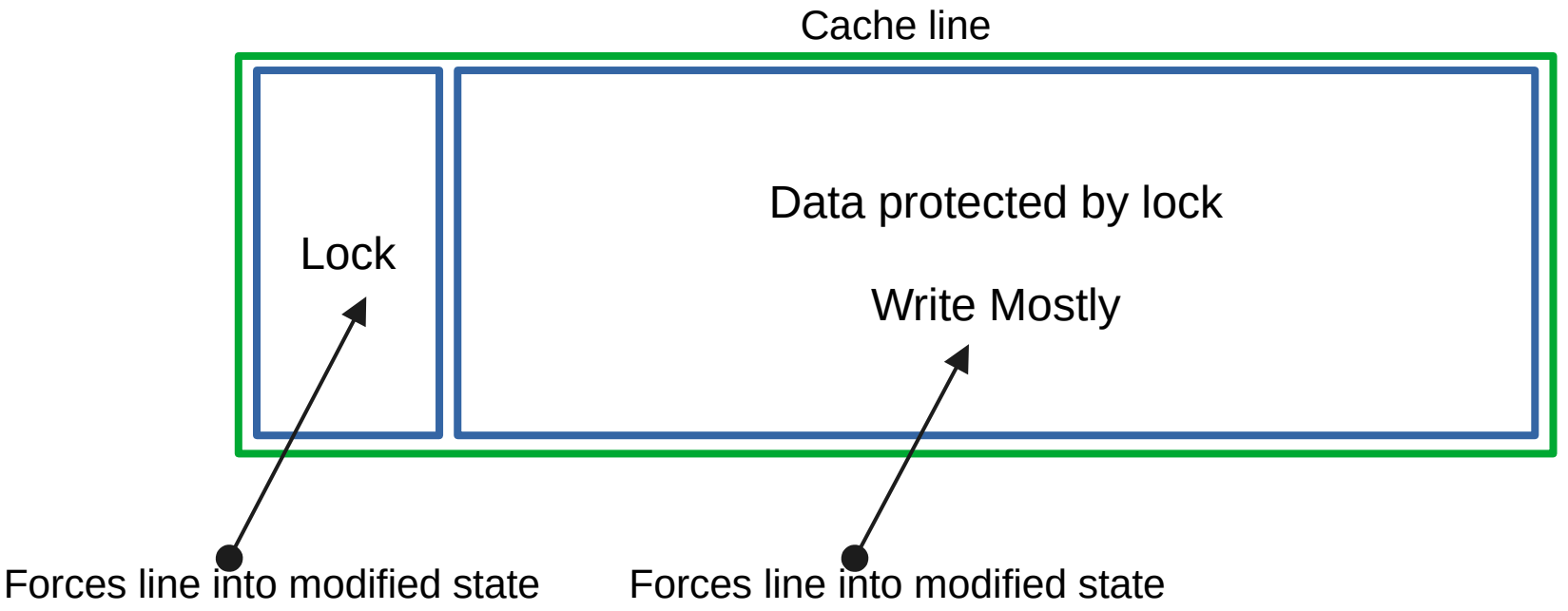
Cache Coherence

- Cache Lines: Commonly 64 bytes, some 128 bytes
- Basic Protocol is called MESI, lots of variations
  - **M**odified: line is modified, on one core
  - **E**xclusive: line isn’t modified, one core, may be modified
  - **S**hared: line isn’t modified, on multiple cores, may not be modified
  - **I**nvalid: line is unused
- More expensive to access cache dirty cache lines on other cores
- More expensive to access memory on other nodes

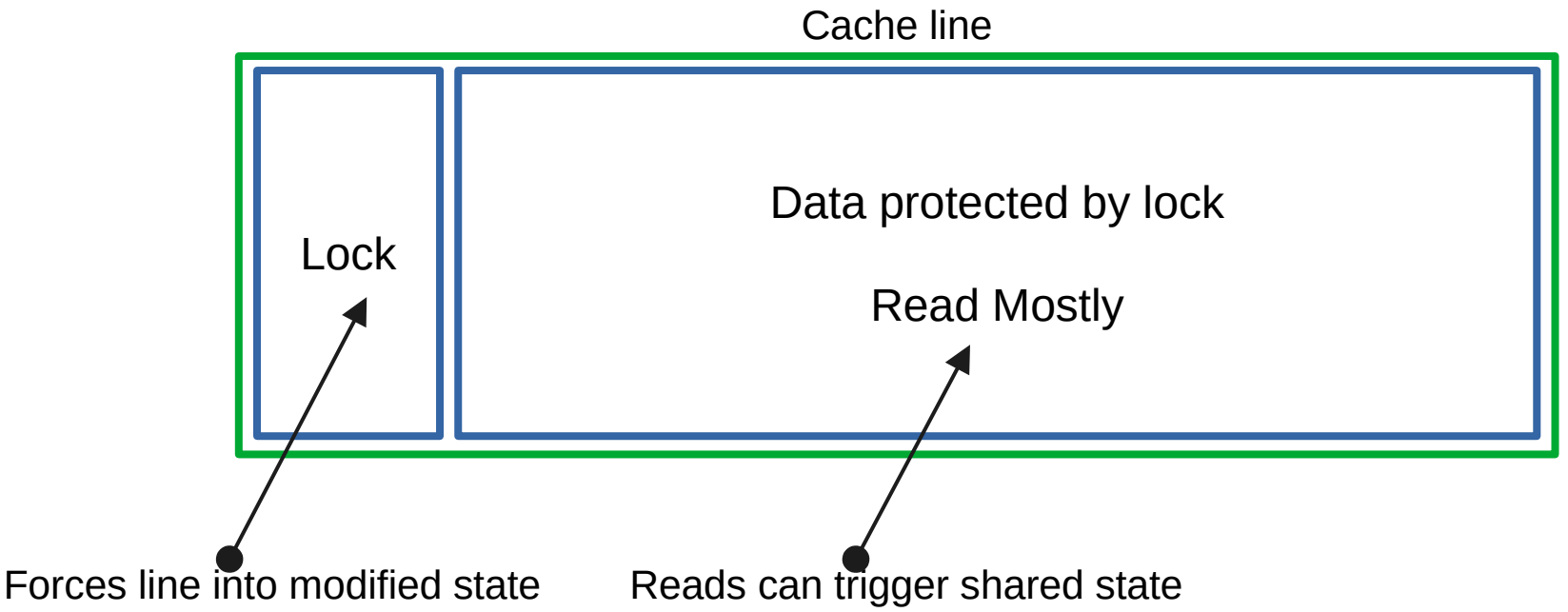
Real Sharing



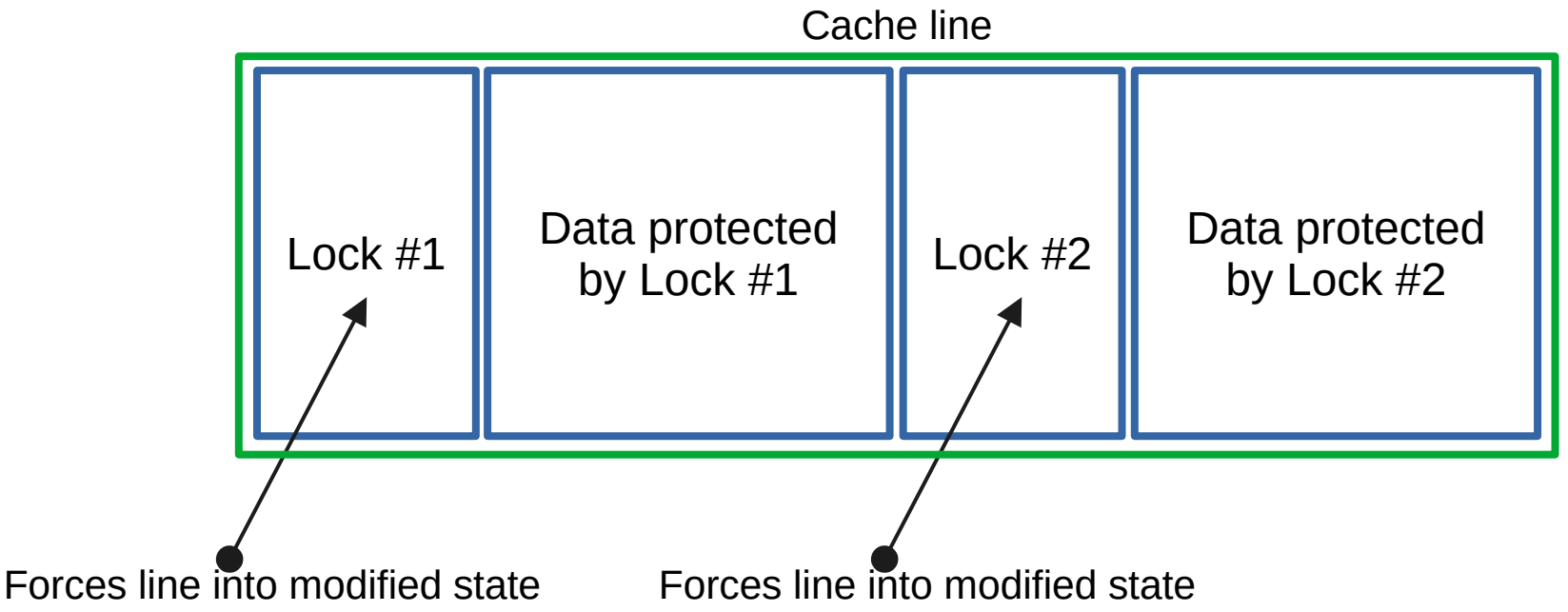
Real Sharing



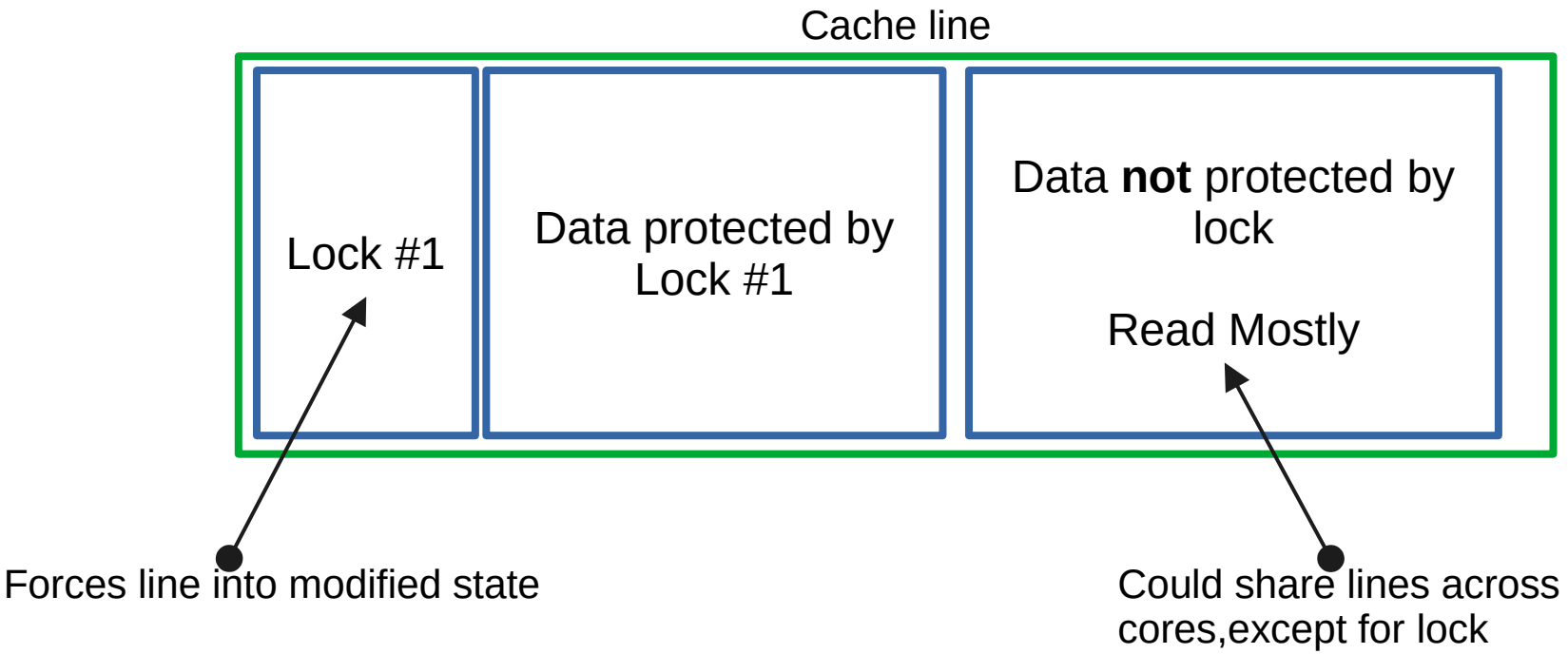
Real Sharing – “conflicting state”



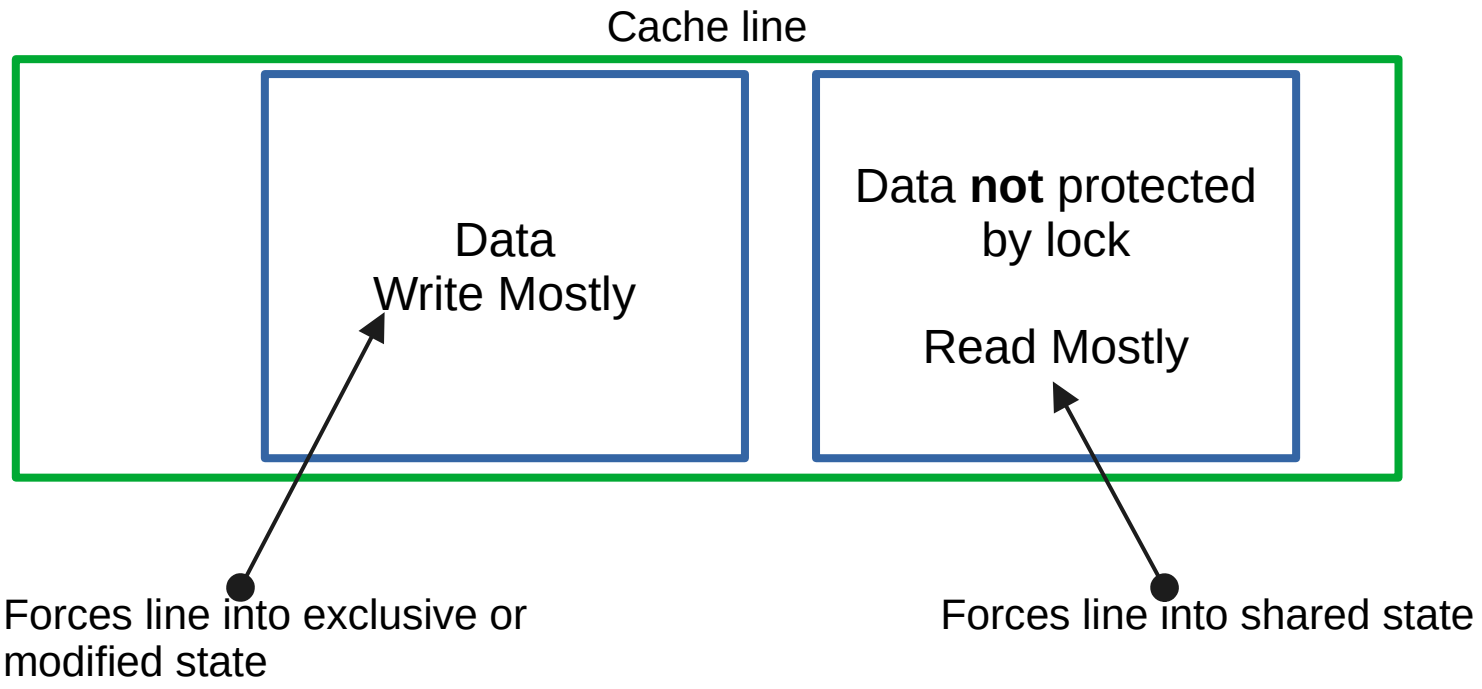
Sharing => “Unnecessary” Contention



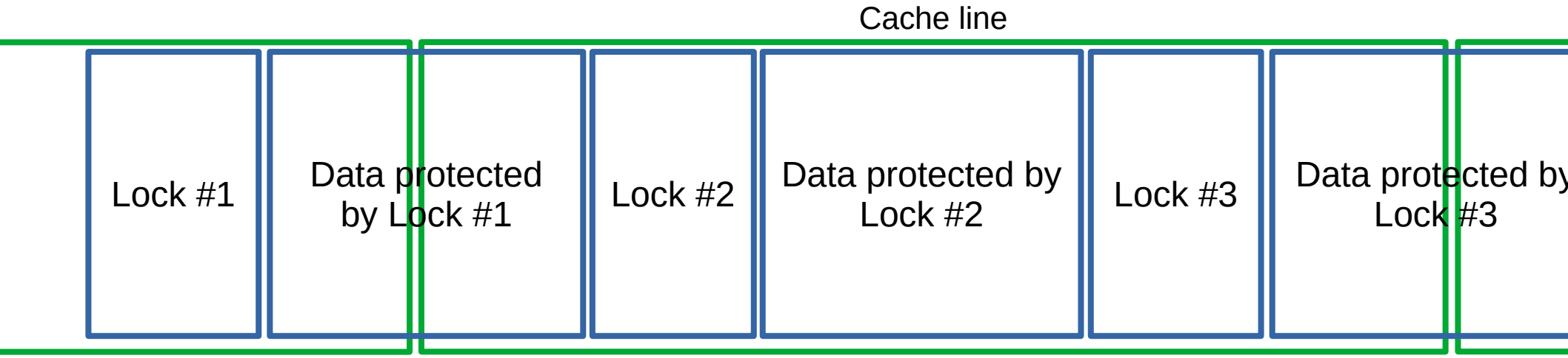
False Sharing: “Mixed Use”



False Sharing: “Mixed Use”



False Sharing: “Alignment”



When is this a problem?

- High concurrency
- Really Bad: Multiple sockets
- Bad: Multiple chiplets in one socket
- Kinda bad: Several cores in one chiplet

What can we do?

- Split data across more cache lines
  - Wasting memory may lead to performance regressions at low concurrency
- Separate write-heavy and read-heavy data
- Redesign data structures more fundamentally
- Redesign locking

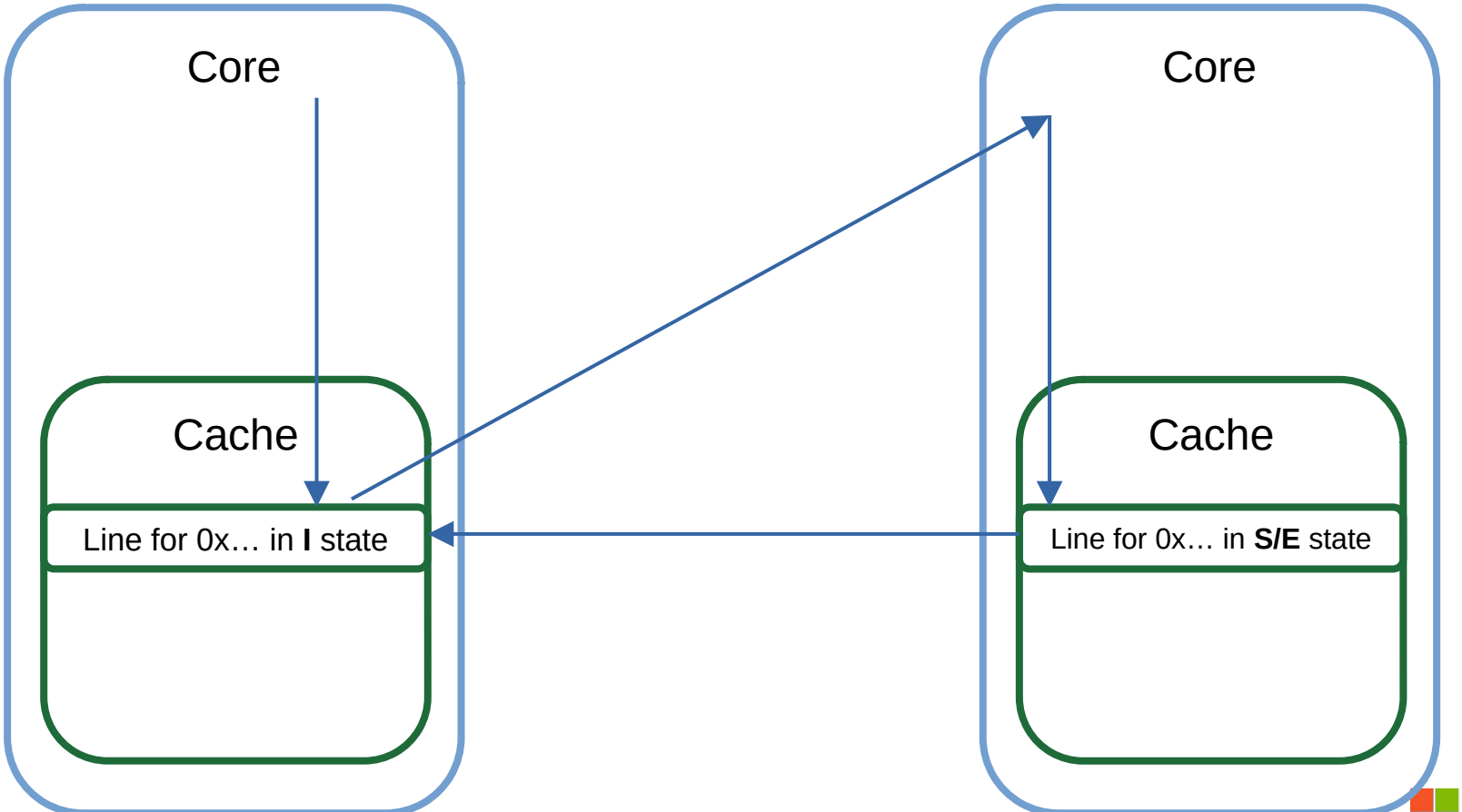
perf c2c

- “cache 2 cache”
- detects accesses to dirty cache lines owned by another core
  - including call-site and offset!
  - “HitM” – Hit Modified Line
- Available on
  - Intel CPUs – most mature
  - AMD Zen4 (possibly earlier) – less precise, only system wide
  - ARM – haven’t tried
- Better results on bigger machines, multiple sockets are helpful

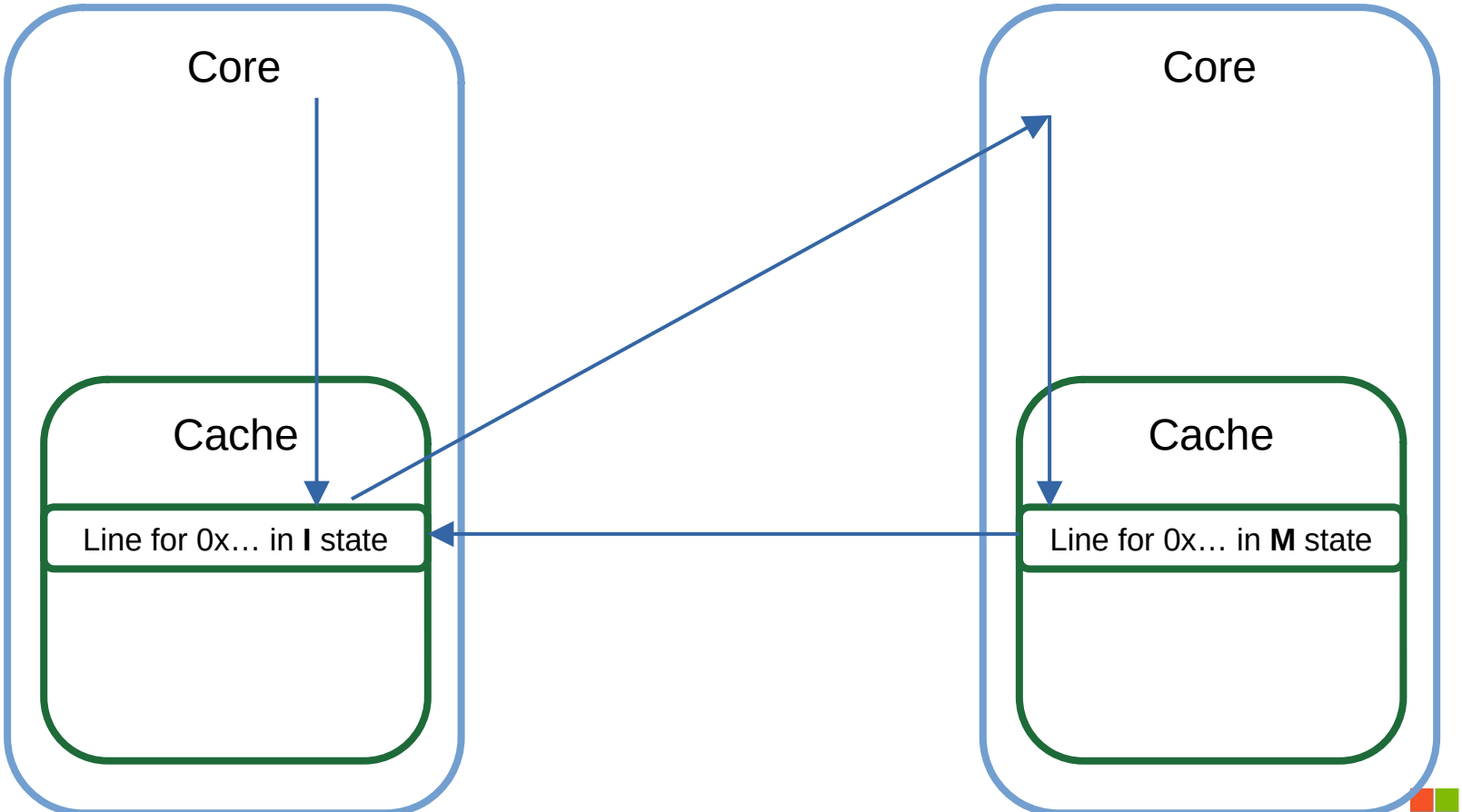
perf c2c - tips

- compile with `-fno-omit-frame-pointer`
  - allows using –call-graph fp
  - smaller profile data, lower overhead
- `--call-graph fp` sometimes imprecise, `--call-graph dwarf` or `lbr` can be more precise
- Increasing frequency with `-F10000`, particularly on AMD

Hit



HitM



Test Workload

Table "public.accounts"

Column	Type	Collation	Nullable	Default
aid	integer			
bid	integer			

Indexes:
    "accounts_aid_idx" btree (aid)
    "accounts_bid_idx" btree (bid)

Table "public.transactions"

Column	Type	Collation	Nullable	Default
aid	integer			
diff	integer			
tid	integer		not null	nextval('transactions_tid_seq'::regclass)

Indexes:
    "transactions_pkey" PRIMARY KEY, btree (tid)
    "transactions_aid_idx" btree (aid)

Microsoft

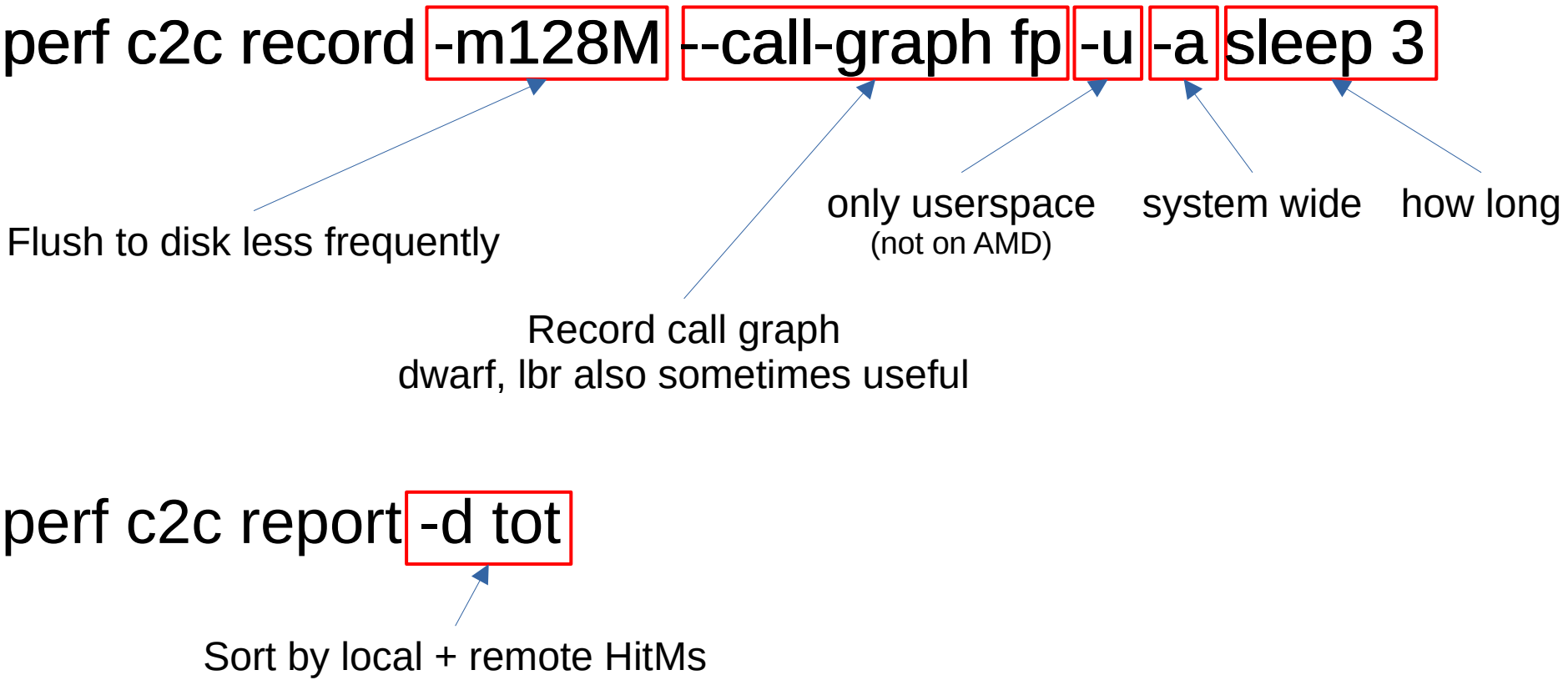
Test Workload

```
postgres[286004][1]=# EXPLAIN (COSTS off) SELECT sum(diff)
FROM transactions t
    JOIN accounts a ON t.aid = a.aid
WHERE tid > 100000 and tid < 101000;
```

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate
   ->  Nested Loop
         ->  Index Scan using transactions_pkey on transactions t
               Index Cond: ((tid > 100000) AND (tid < 101000))
         ->  Index Only Scan using accounts_aid_idx on accounts a
               Index Cond: (aid = t.aid)
```

https://anarazel.de/talks/2024-05-29-pgconf-dev-c2c/setup.sql
https://anarazel.de/talks/2024-05-29-pgconf-dev-c2c/q.sql

Microsoft

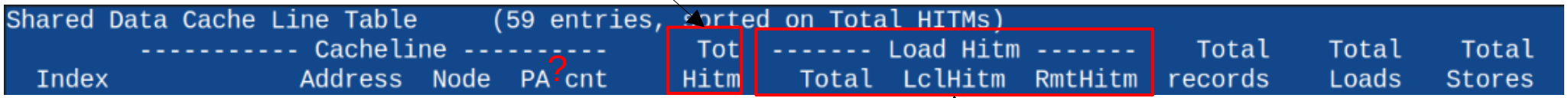

perf c2c record -m128M --call-graph fp -u -a sleep 3
Flush to disk less frequently
Record call graph
dwarf, lbr also sometimes useful
only userspace
(not on AMD)
system wide
how long
perf c2c report -d tot
Sort by local + remote HitMs

Shared Data Cache Line Table (59 entries, sorted on Total HITMs)

	Index	Cacheline Address	Node	PA cnt	Tot Hitm	Load Hitm Total	Load Hitm LclHitm	Load Hitm RmtHitm	Total records	Total Loads	Total Stores	Stores L1Hit	Stores L1Miss	Stores N/A	Core Load Hit FB	Core Load Hit L1	Core Load Hit L2	LLC Load Hit LclHit	LLC Load Hit LclHitm	RMT Load Hit RmtHit	RMT Load Hit RmtHitm	Load Dram Lcl	Load Dram Rmt
+	0	0x7ffd9212ea80	0	49020	48.47%	17265	13134	4131	74914	66750	8164	8164	0	0	13114	28318	0	3859	13134	0	4131	3	4191
+	1	0x7ffd80008300	0	1	20.30%	7231	4403	2828	20799	17181	3618	3617	1	0	302	5165	1	1375	4403	0	2828	8	3099
+	2	0x7ffd9212eb00	0	41	3.67%	1306	1297	9	3244	3244	0	0	0	0	100	5	0	892	1297	0	9	174	767
+	3	0x7ffd92132a40	0	3950	1.91%	680	369	311	5446	4798	648	648	0	0	2932	807	0	35	369	0	311	0	344
+	4	0x7ffd9212fe00	0	3961	1.89%	673	329	344	5550	4836	714	714	0	0	2956	807	0	21	329	0	344	0	379
+	5	0x7ffd92131940	0	3889	1.85%	659	343	316	5423	4658	765	765	0	0	2845	764	0	34	343	0	316	0	356
+	6	0x7ffd921323c0	0	4020	1.73%	616	326	290	5604	4822	782	782	0	0	3024	767	2	85	326	0	290	3	325
+	7	0x7ffd92132ec0	0	3751	1.70%	607	311	296	5205	4472	733	733	0	0	2785	728	0	26	311	0	296	1	325
+	8	0x7ffd9212ec80	0	1887	0.74%	262	190	72	2528	2208	320	320	0	0	1378	340	0	10	190	0	72	1	217
+	9	0x7ffd80007580	0	1	0.28%	100	51	49	419	324	95	95	0	0	1	136	0	34	51	0	49	0	53
+	10	0x7ffd80006c80	0	1	0.28%	99	45	54	402	316	86	86	0	0	0	128	0	35	45	0	54	0	54
+	11	0x7ffd80008680	0	1	0.27%	97	66	31	446	347	99	99	0	0	1	144	0	37	66	0	31	0	68
+	12	0x7ffd80007b80	0	1	0.25%	88	36	52	386	275	111	111	0	0	0	99	0	34	36	0	52	0	54
+	13	0x7ffd80009e80	0	1	0.22%	78	39	39	327	242	85	85	0	0	2	88	0	33	39	0	39	0	41
+	14	0x7ffd80007680	0	1	0.21%	76	47	29	355	266	89	89	0	0	0	119	0	37	47	0	29	0	34
+	15	0x7ffd80006680	0	1	0.21%	74	30	44	342	248	94	94	0	0	4	101	0	24	30	0	44	0	45
+	16	0x7ffd80006e80	0	1	0.20%	73	37	36	366	280	86	86	0	0	1	127	0	37	37	0	36	0	42
+	17	0x7ffd80007380	0	1	0.20%	72	41	31	313	237	76	76	0	0	0	103	0	28	41	0	31	0	34
+	18	0x7ffd80009480	0	1	0.20%	72	42	30	322	244	78	78	0	0	0	112	0	29	42	0	30	0	31
+	19	0x7ffd80006780	0	1	0.20%	71	33	38	353	256	97	97	0	0	1	119	0	25	33	0	38	0	40
+	20	0x7ffd80009300	0	1	0.19%	68	34	34	283	211	72	72	0	0	0	72	0	30	34	0	34	0	41
+	21	0x7ffd80006880	0	1	0.18%	64	33	31	299	214	85	85	0	0	0	96	0	21	33	0	31	0	33
+	22	0x7ffd80008980	0	1	0.17%	60	34	26	323	235	88	88	0	0	1	97	0	34	34	0	26	0	43
+	23	0x7ffd9212ee80	0	181	0.17%	60	49	11	491	221	70	70	0	0	40	69	0	2	49	0	11	0	50
+	24	0x7ffd80009380	0	1	0.16%	58	26	32	302	216	86	86	0	0	0	99	0	24	26	0	32	0	35
+	25	0x7ffd80007300	0	1	0.15%	54	24	30	236	178	58	58	0	0	1	76	0	17	24	0	30	0	30
+	26	0x7ffd80007b00	0	1	0.15%	54	24	30	221	167	46	46	0	0	1	73	0	18	24	0	30	0	31
+	27	0x7ffd8000a300	0	1	0.15%	53	26	27	246	181	65	65	0	0	0	80	0	21	26	0	27	0	27
+	28	0x7ffd80007c80	0	1	0.14%	51	26	25	223	170	53	53	0	0	1	68	0	20	26	0	25	0	30
+	29	0x7ffd80007600	0	1	0.14%	50	21	29	246	178	68	68	0	0	0	76	0	17	21	0	29	0	35
+	30	0x7ffd80006b80	0	1	0.14%	49	19	30	235	183	52	52	0	0	3	76	0	20	19	0	30	0	35
+	31	0x7ffd80008f80	0	1	0.14%	49	36	13	263	194	69	69	0	0	1	83	0	22	36	0	13	0	39
+	32	0x7ffd80007880	0	1	0.13%	48	21	27	221	159	62	62	0	0	1	61	0	19	21	0	27	0	30
+	33	0x7ffd80008e80	0	1	0.13%	48	27	21	222	182	40	40	0	0	3	78	0	26	27	0	21	0	27
+	34	0x7ffd80006b00	0	1	0.13%	47	24	23	294	222	72	72	0	0	0	121	0	28	24	0	23	0	26
+	35	0x7ffd80007700	0	1	0.13%	47	22	25	195	146	49	49	0	0	0	56	0	14	22	0	25	0	29
+	36	0x7ffd8000a380	0	1	0.13%	46	17	29	210	157	53	53	0	0	0	66	0	15	17	0	29	0	30
+	37	0x7ffd80006c00	0	1	0.13%	45	18	27	209	151	58	58	0	0	0	65	0	14	18	0	27	0	27
+	38	0x7ffd80007f80	0	1	0.13%	45	21	24	191	138	53	53	0	0	0	52	0	16	21	0	24	0	25
+	39	0x7ffd80008180	0	1	0.13%	45	33	12	260	186	74	74	0	0	1	82	0	22	33	0	12	0	36
+	40	0x7ffd80008080	0	1	0.12%	44	28	16	230	171	59	59	0	0	0	76	0	20	28	0	16	0	31
+	41	0x7ffd80009b80	0	1	0.12%	42	16	26	196	152	44	44	0	0	1	64	0	18	16	0	26	0	27
+	42	0x7ffd80006580	0	1	0.12%	41	19	22	197	158	39	39	0	0	0	78	0	16	19	0	22	0	23
+	43	0x7ffd80008500	0	1	0.12%	41	33	8	194	134	60	59	1	0	0	49	0	15	33	0	8	0	29
+	44	0x7ffd8000a080	0	1	0.11%	40	24	16	172	130	42	42	0	0	0	53	0	19	24	0	16	0	18
+	45	0x7ffd80008280	0	1	0.11%	39	32	7	213	160	53	53	0	0	0	67	0	19	32	0	7	0	35
+	46	0x7ffd80009080	0	1	0.11%	39	19	20	139	111	28	28	0	0	0	39	0	12	19	0	20	0	21
+	47	0x7ffd80009200	0	1	0.11%	39	17	22	183	133	50	50	0	0	0	55	0	16	17	0	22	0	23
+	48	0x7ffd80009500	0	1	0.11%	39	19	20	195	152	43	43	0	0	1	66	0	23	19	0	20	0	23
+	49	0x7ffd80009d80	0	1	0.11%	39	21	18	188	140	48	48	0	0	0	66	0	17	21	0	18	0	18
+	50	0x7ffd80008200	0	1	0.11%	38	25	13	203	146	57	57	0	0	0	58	0	17	25	0	13	0	33
+	51	0x7ffd80008880	0	1	0.11%	38	30	8	217	160	57	57	0	0	0	74	0	23	30	0	8	0	25
+	52	0x7ffd80009580	0	1	0.11%	38	21	17	193	143	50	50	0	0	0	70	0	12	21	0	17	0	23
+	53	0x7ffd9212ecc0	0	150	0.11%	38	29	9	209	162	47	47	0	0	32	56	0	1	29	0	9	0	35
+	54	0x7ffd80006d00	0	1	0.10%	37	19	18	178	129	49	49	0	0	1	54	0	16	19	0	18	0	21
+	55	0x7ffd80009600	0	1	0.10%	37	18	19	139	108	31	31	0	0	1	38	0	10	18	0	19	0	22
+	56	0x7ffd9212eec0	0	149	0.10%	37	29	8	214	170	44	44	0	0	39	52	0	5	29	0	8	0	37
+	57	0x7ffd80007180	0	1	0.10%	36	18	18	181	133	48	48	0	0	0	67	0	12	18	0	18	0	18
+	58	0x7ffd80007c00	0	1	0.10%	36	22	14	148	110	38	38	0	0	0	41	0	14	22	0	14	0	19

perf c2c: List of Cache Lines

Percentage of overall HitM



Store found line where?

Used Modified Cache line on same/different socket

Loads on Current core

Loads on same node

Loads on remote node

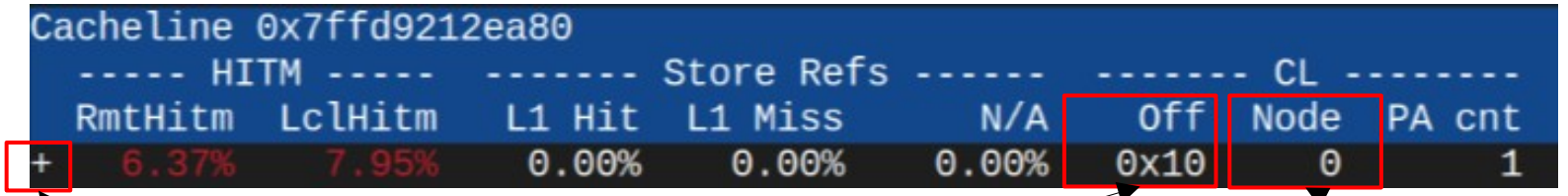
Microsoft

Shared Data Cache Line Table (59 entries, sorted on Total HITMs)

	Cacheline			Tot	Load Hitm		
Index	Address	Node	PA cnt	Hitm	Total	LclHitm	RmtHitm
+ 0	0x7ffd9212ea80	0	49020	48.47%	17265	13134	4131
+ 1	0x7ffd80008300	0	1	20.30%	7231	4403	2828
+ 2	0x7ffd9212eb00	0	41	3.67%	1306	1297	9
+ 3	0x7ffd92132a40	0	3950	1.91%	680	369	311
+ 4	0x7ffd9212fe00	0	3961	1.89%	673	329	344
+ 5	0x7ffd92131940	0	3889	1.85%	659	343	316
+ 6	0x7ffd921323c0	0	4020	1.73%	616	326	290
+ 7	0x7ffd92132ec0	0	3751	1.70%	607	311	296
+ 8	0x7ffd9212ec80	0	1887	0.74%	262	190	72
+ 9	0x7ffd80007580	0	1	0.28%	100	51	49
+ 10	0x7ffd80006c80	0	1	0.28%	99	45	54
+ 11	0x7ffd80008680	0	1	0.27%	97	66	31
+ 12	0x7ffd80007b80	0	1	0.25%	88	36	52
+ 13	0x7ffd80009e80	0	1	0.22%	78	39	39
+ 14	0x7ffd80007680	0	1	0.21%	76	47	29
+ 15	0x7ffd80006680	0	1	0.21%	74	30	44
+ 16	0x7ffd80006e80	0	1	0.20%	73	37	36
+ 17	0x7ffd80007380	0	1	0.20%	72	41	31
+ 18	0x7ffd80009480	0	1	0.20%	72	42	30

------- Stores --------			----- Core Load Hit -----			- LLC Load Hit --		- RMT Load Hit --		--- Load Dram ----	
L1Hit	L1Miss	N/A	FB	L1	L2	LclHit	LclHitm	RmtHit	RmtHitm	Lcl	Rmt
8164	0	0	13114	28318	0	3859	13134	0	4131	3	4191
3617	1	0	302	5165	1	1375	4403	0	2828	8	3099
0	0	0	100	5	0	892	1297	0	9	174	767
648	0	0	2932	807	0	35	369	0	311	0	344
714	0	0	2956	807	0	21	329	0	344	0	379
765	0	0	2845	764	0	34	343	0	316	0	356
782	0	0	3024	767	2	85	326	0	290	3	325
733	0	0	2785	728	0	26	311	0	296	1	325
320	0	0	1378	340	0	10	190	0	72	1	217
95	0	0	1	136	0	34	51	0	49	0	53
86	0	0	0	128	0	35	45	0	54	0	54
99	0	0	1	144	0	37	66	0	31	0	68
111	0	0	0	99	0	34	36	0	52	0	54
85	0	0	2	88	0	33	39	0	39	0	41
89	0	0	0	119	0	37	47	0	29	0	34
94	0	0	4	101	0	24	30	0	44	0	45
86	0	0	1	127	0	37	37	0	36	0	42
76	0	0	0	103	0	28	41	0	31	0	34
78	0	0	0	112	0	29	42	0	30	0	31
97	0	0	1	119	0	25	33	0	38	0	40
72	0	0	0	72	0	30	34	0	34	0	41
85	0	0	0	96	0	21	33	0	31	0	33
88	0	0	1	97	0	34	34	0	26	0	43
70	0	0	40	69	0	2	49	0	11	0	50
86	0	0	0	99	0	24	26	0	32	0	35
58	0	0	1	76	0	17	24	0	30	0	30
46	0	0	1	73	0	18	24	0	30	0	31
65	0	0	0	80	0	21	26	0	27	0	27

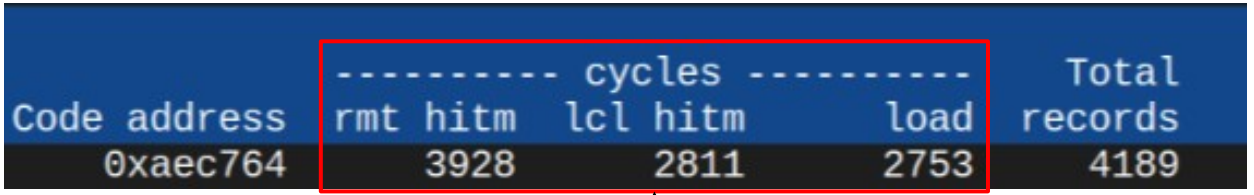
press [d]etail to view one cacheline



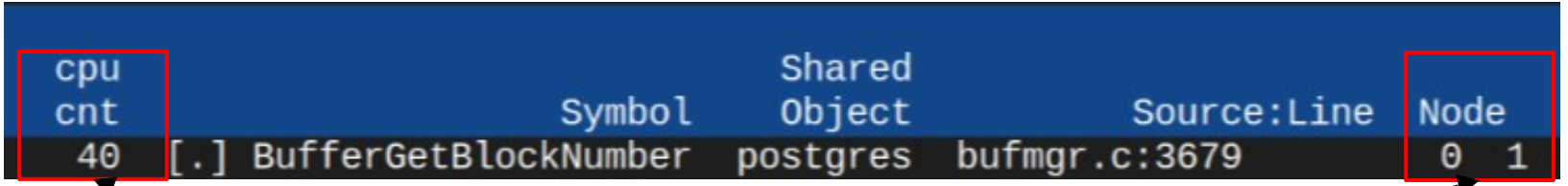
Press + to view callers

Offset in cacheline

Where is memory located



Mean Access Time



Accessed by how many cpus?

Accessed on which nodes?

Microsoft

2nd Cache Line

Cacheline 0x7ffd80008300

	----- HITM -----		------- Store Refs ------			------- CL --------				---------- cycles ----------		Total
	RmtHitm	LclHitm	L1 Hit	L1 Miss	N/A	Off	Node	PA cnt	Code address	rmt hitm	lcl hitm	load	records
+	51.39%	49.70%	0.00%	0.00%	0.00%	0x4	0	1	0xb20903	523	310	365	5862
+	28.96%	29.74%	51.32%	0.00%	0.00%	0x4	0	1	0xb20918	3122	2837	829	8564
+	19.65%	20.54%	48.68%	0.00%	0.00%	0x4	0	1	0xb2123d	3983	3546	767	6542
+	0.00%	0.02%	0.00%	0.00%	0.00%	0x4	0	1	0xb2091c	0	26920	0	1

same offset for all

cpu cnt		Symbol	Shared Object	Source:Line	Node
40	[.]	LWLockAcquire	postgres	generic.h:48	0 1
40	[.]	LWLockAcquire	postgres	arch-x86.h:177	0 1
40	[.]	LWLockRelease	postgres	generic-gcc.h:212	0 1
1	[.]	LWLockAcquire	postgres	arch-x86.h:177	0

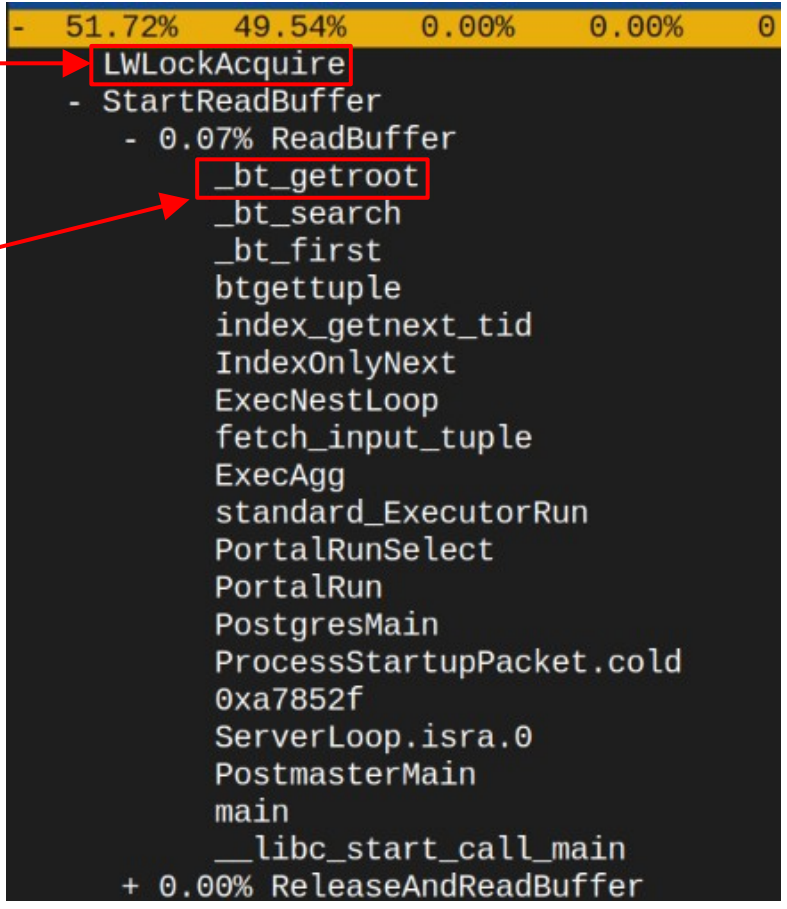
on all cores

Microsoft

2nd Cache Line

Shared Lock

Could be cached!



Microsoft

1st Cache Line

Cacheline 0x7ffd9212ea80

	----- HITM -----		------- Store Refs ------			------- CL -		cpu			
	RmtHitm	LclHitm	L1 Hit	L1 Miss	N/A	Off	Node	cnt	Symbol	Node	
+	6.37%	7.95%	0.00%	0.00%	0.00%	0x10	0	40	[.] BufferGetBlockNumber	0	1
+	3.95%	3.62%	0.00%	0.00%	0.00%	0x10	0	40	[.] ReleaseAndReadBuffer	0	1
+	8.98%	8.76%	0.00%	0.00%	0.00%	0x14	0	40	[.] StartReadBuffer	0	1
+	6.97%	7.87%	0.00%	0.00%	0.00%	0x14	0	40	[.] PinBuffer	0	1
+	3.41%	3.60%	0.00%	0.00%	0.00%	0x14	0	40	[.] UnpinBufferNoOwner	0	1
+	0.97%	1.05%	0.00%	0.00%	0.00%	0x14	0	40	[.] ReleaseAndReadBuffer	0	1
+	21.88%	19.49%	19.17%	0.00%	0.00%	0x18	0	40	[.] PinBuffer	0	1
+	17.67%	17.95%	23.24%	0.00%	0.00%	0x18	0	40	[.] UnpinBufferNoOwner	0	1
+	2.32%	2.22%	0.00%	0.00%	0.00%	0x18	0	40	[.] UnpinBufferNoOwner	0	1
+	2.01%	2.06%	0.00%	0.00%	0.00%	0x18	0	40	[.] PinBuffer	0	1
+	0.02%	0.00%	0.00%	0.00%	0.00%	0x18	0	2	[.] PinBuffer	0	
+	16.12%	16.39%	33.17%	0.00%	0.00%	0x28	0	40	[.] LWLockAcquire	0	1
+	5.37%	5.70%	24.41%	0.00%	0.00%	0x28	0	40	[.] LWLockRelease	0	1
+	3.95%	3.33%	0.00%	0.00%	0.00%	0x28	0	40	[.] LWLockAcquire	0	1
+	0.00%	0.01%	0.00%	0.00%	0.00%	0x28	0	1	[.] LWLockAcquire	0	
+	0.00%	0.01%	0.01%	0.00%	0.00%	0x28	0	2	[.] LWLockRelease	0	

Microsoft

1st:

```
BlockNumber BufferGetBlockNumber(Buffer buffer)
…
    return bufHdr->tag.blockNum;
```

2nd:

```
Buffer
ReleaseAndReadBuffer(Buffer buffer,
                     Relation relation,
                     BlockNumber blockNum)
...
          if (bufHdr->tag.blockNum == blockNum && ...
              return buffer;
```

3rd:

```
static inline Buffer
BufferDescriptorGetBuffer(const BufferDesc *bdesc)
{
    return (Buffer) (bdesc->buf_id + 1);
}
```

```
struct BufferDesc {
        BufferTag                       tag;                    /*     0    20 */
        int                             buf_id;                 /*    20     4 */
        pg_atomic_uint32                state;                  /*    24     4 */
        int                             wait_backend_pgprocno; /*    28     4 */
        int                             freeNext;               /*    32     4 */
        LWLock                          content_lock;           /*    36    16 */

        /* size: 52, cachelines: 1, members: 6 */
        /* last cacheline: 52 bytes */
};

union BufferDescPadded {
        BufferDesc                      bufferdesc;             /*     0    52 */
        char                            pad[64];                /*     0    64 */
};
```

Quick and Dirty Fix

```
typedef struct BufferDesc
{
    BufferTag tag;              /* ID of page contained in buffer */
    int       buf_id;           /* buffer's index number (from 0) */

    int       wait_backend_pgprocno;  /* backend of pin-count waiter */
    int       freeNext;     /* link in freelist chain */

    /* state of the tag, containing flags, refcount and usagecount */
    pg_atomic_uint32 state pg_attribute_aligned(64);

    /* to lock access to buffer contents */
    LWLock    content_lock pg_attribute_aligned(64);
} BufferDesc;
```

Microsoft

Before / After

	Clients	TPS Before	TPS After	Pct
Laptop	1	1660	1635	0.98x
Laptop	16	7538	10782	1.43x
Workstation	50	2285	3015	1.32x

decreased cache usage?

nice gain

nice, but less so

scales really badly compared to laptop
cacheline contention

Microsoft

Caveats

- Somewhat artificial workload – but it reproduces even on small machines
- Originally issue found with pgbench -S
  - but reproduces only on big machines
  - on medium size machines pipelining suffices
  - large: ~1/3 improvement

Analyzing Cache Line Contention Using perf c2c

Andres Freund
PostgreSQL Developer & Committer
Email: andres@anarazel.de
Email: andres.freund@microsoft.com

https://anarazel.de/talks/2024-05-29-pgconf-dev-c2c/postgres-perf-c2c.pdf

Microsoft