Using Modern Assembler
Coding Practices to Reduce CPU Usage

Adam Johanson, Broadcom
Software Developer
Adam.Johanson@broadcom.com

About Me

- Developer on Broadcom's Mainframe storage products
  - 6 years

- Previously:
  - 6+ years as Sysprog/DBA at USAA

  - 8+ years in IMS Development on System Services team
    - Primarily focused on IMS Logger
    - A little bit of User Exit Services

Come Along for the Ride

- Real customer problem
- Tools used
- Why some of the code wasn't performing well
- Three instructive changes made
  - An obvious one
  - One needing a design change
  - One where the problem was more "indirect"
- Learn to shave

First Documentation Sent in

- Application performance monitor

- Initial case contact:
  - “IBM said that some programs don’t execute instructions in a good sequence”

LOAD MODULE	CSECT		CPU % - THIS CSECT		CPU % - TOTAL FOR JOB		
PGM001	PGM001		14.56		14.58		
PGM002	PGM002		3.01		3.01		
PGM003	PGM003		1.67		1.67		

Performance Monitor – PGM001

Code section	Length of section	CPU Time
000000	128	.00
000080	64	.14
0000C0	64	.00
000010	64	.07
000140	64	.46
000180	64	.36
0001C0	64	.06
000200	64	.14
000240	64	.39
000280	64	.22
0002C0	64	11.23
000300	64	.00
000340	64	1.33
000380	64	.02
0003C0	64	.13
000400	304	.00

The Hot Spot

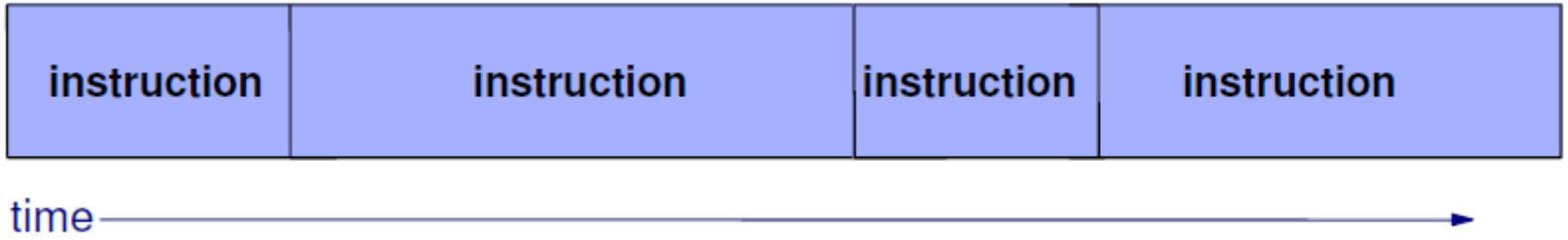
```
****** ********************************************************************** Top of Data ******
000001               LA      R10,HEAD                Start of list
000010               LR      R1,R10                  "
000011 CHECK         DS      0H
000020               CLC     4(8,R1),KEY             Our field?
000100               BE      END
000200               L       R1,0(,R1)               Offset of next field
000300               LTR     R1,R1
000400               BZ      ZEROFND
000500               LA      R1,0(R10,R1)            Add offset to start of list
000510 ZEROFND       DS      0H
000600               LTR     R1,R1                   Yes, it really checks R1 again
000700               BNZ     CHECK
000800 END            DS      0H                      If R1 = 0 here, we didn't find it
****** ********************************************************************** Bottom of Data ***
```

Offset	Field name	Field value
offset of next field	DSN	Data set name
offset of next field	BLKSIZE	Block size of data set
offset of next field	LRECL	LRECL of data set
…	…	…
0	field name	field value

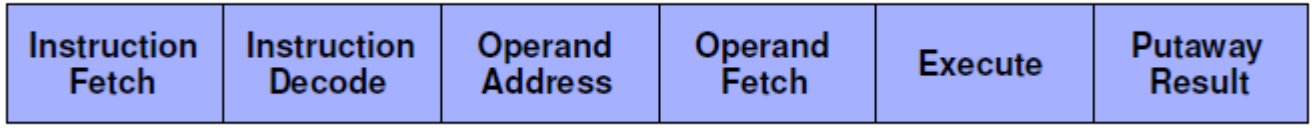
11% of the job's CPU

Why is This a Problem, From the CPU's Point of View?

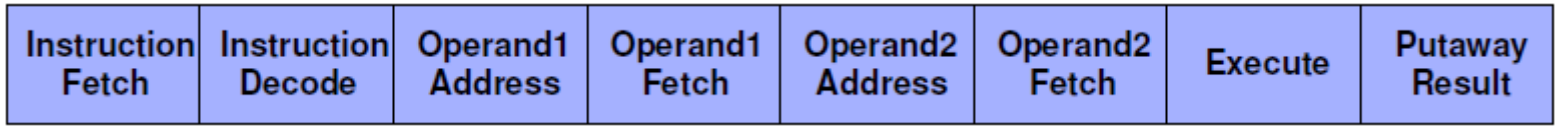
- Instructions appear to be executed "simply serially"



- Each instruction has multiple activities to do
  - `A  R1,D2(X2,B2)`

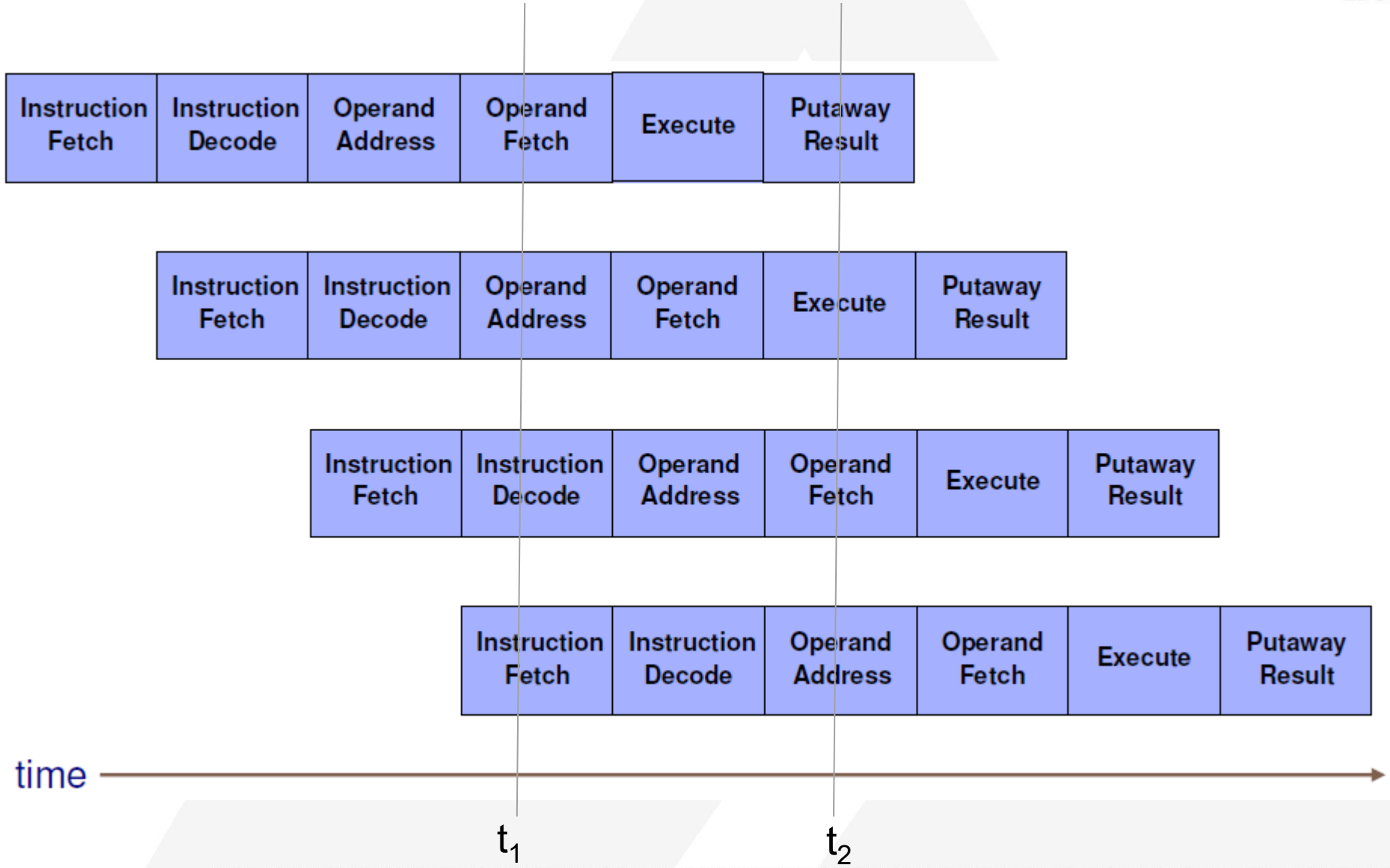


  - `CLC  D1(L,B1),D2(B2)`



* Graphics used from Bob Rogers' presentations in "Sources" section

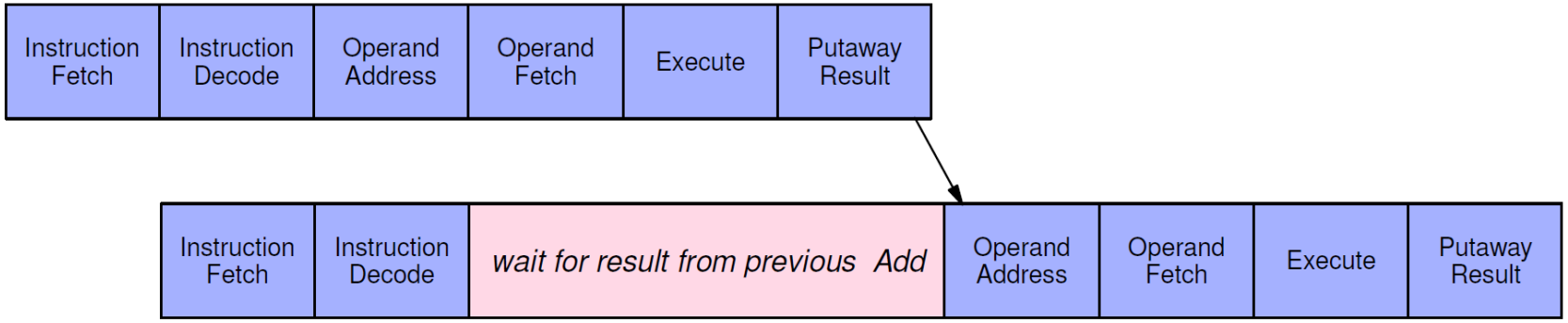
Instructions are Executed as Part of a Pipeline



What Does This Have to do With Our Hot Spot?

- Address Generation Interlock (AGI)
  - Waiting on results of a previous instruction to compute an operand address

```
A       R1,field
L       R2,0(,R1)
```




```
******  ********************************************************************** Top of Data ******
000001              LA      R10,HEAD                Start of list
000010              LR      R1,R10                  "
000011  CHECK       DS      0H
000020              CLC     4(8,R1),KEY             Our field?
000100              BE      END
000200              L       R1,0(,R1)               Offset of next field
000300              LTR     R1,R1
000400              BZ      ZEROFND
000500              LA      R1,0(R10,R1)            Add offset to start of list
000510  ZEROFND     DS      0H
000600              LTR     R1,R1                   Yes, it really checks R1 again
000700              BNZ     CHECK
000800  END         DS      0H                      If R1 = 0 here, we didn't find it
******  ********************************************************************** Bottom of Data ***
```

Back to Our Hot Spot

```
****** ********************************************************** Top of Data ******
000001             LA      R10,HEAD                Start of list
000010             LR      R1,R10                  "
000011 CHECK       DS      0H
000020             CLC     4(8,R1),KEY             Our field?
000100             BE      END
000200             L       R1,0(,R1)               Offset of next field
000300             LTR     R1,R1
000400             BZ      ZEROFND
000500             LA      R1,0(R10,R1)            Add offset to start of list
000510 ZEROFND     DS      0H
000600             LTR     R1,R1                   Yes, it really checks R1 again
000700             BNZ     CHECK
000800 END          DS      0H                      If R1 = 0 here, we didn't find it
****** ********************************************************** Bottom of Data ***
```

Things I tried:

- Replaced L / LTR sequence with ICM
- Changed to use relative branches
- Changed the

  LTR R1,R1

  BZ .... → CLIJE R1,0,label

Design Change For Holding This Info

As opposed to:

Offset	Field name	Field value
offset of next field	DSN	Data set name
offset of next field	BLKSIZE	Block size of data set
offset of next field	LRECL	LRECL of data set
...	...	...
0	field name	field value

A hash table is constructed based on field name

Hash Table

- For some performance changes, a design change is needed

  EBCDIC values for the field names are “added” up as halfwords

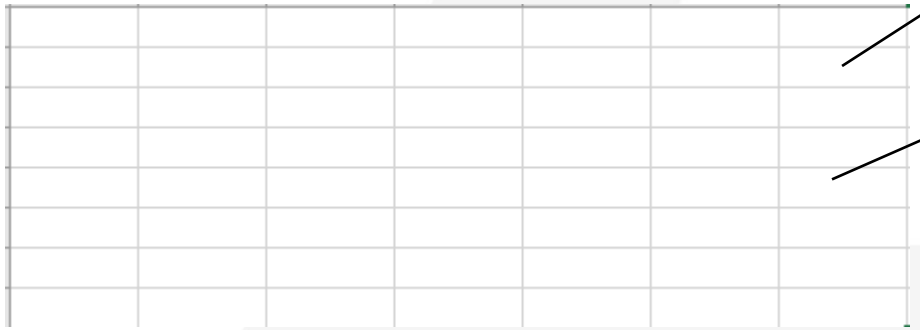
  E.g. c'BLKSIZE' = x'C2D3D2E2C9E9C540'

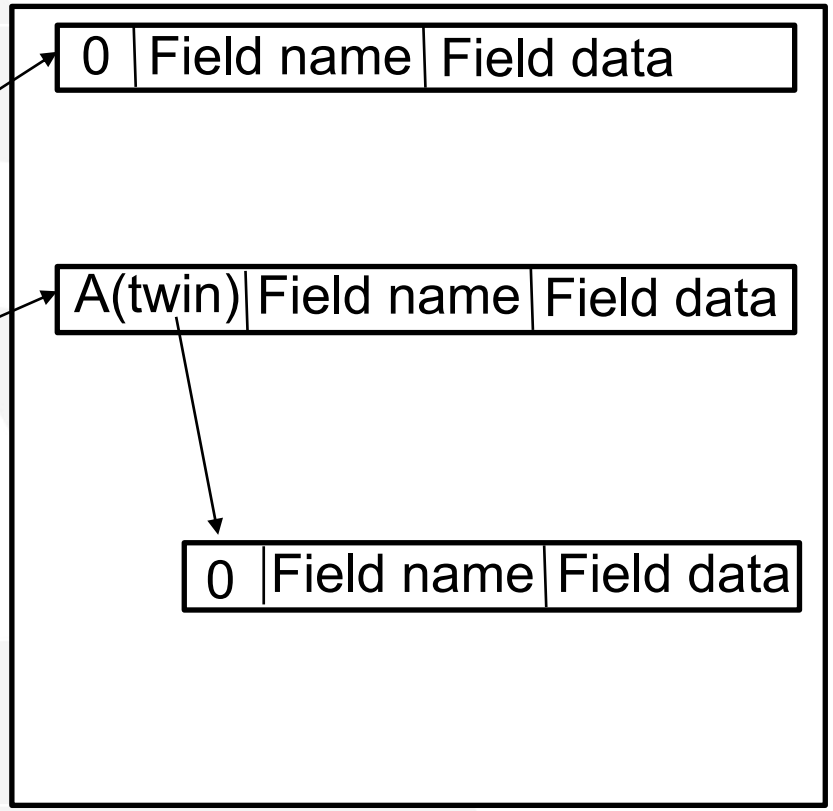
  C2D3 + D2E2 + C9E9 + C540 = x'324DE'

  Result divided by a number of "slots" in the hash table

  Big, prime, number

  This "slot" points to area with data





Performance Problem #2 – SIIS Violations

- Mentioned in a case update

"Storing Into Instruction Stream"

One of the worst things you can do, performance-wise

Even worse on modern processors

Involves changing "stuff" that's "close" to code

Can be obvious and explicit:

```
            CLI     VARIABLE,C'1'
            JE      DONE
            MVI     VARIABLE,C'1'
*     Do stuff *
            J       DONE
VARIABLE    DS      C
DONE        DS    0H
```

```
            BC      0,ONE_TIME
            OI      *-3,X'F0'
*     Do one-time logic stuff
ONE_TIME    DS    0H
```

SIIS Violations

SIIS violations can also happen "on your behalf" by macros:

```
                                    4          OPEN  ((R2),OUTPUT)
000000                              6+         CNOP  0,4                    Align list to word
000000 4D10 C008            00008   7+         BAS   1,*+8                  Load reg 1 with list address
000004 00000000                     8+         DC    A(0)                   Opt byte & DCB or ACB addr
000008 BE27 1001            00001   9+         STCM  R2,B'0111',0+1(1)       Store DCB or ACB addr     @L3C
00000C 928F 1000      00000         10+        MVI   0(1),143               Set option byte
000010 0A13                         11+        SVC   19                     Issue OPEN SVC
```

SIIS Violations

Sneaky with writable variables in program

```
RETURN   DS    0H                  Return to caller
         L     R13,4(,R13)         Restore A(caller's save area)
         L     R14,12(,R13)        Restore R14
         XR    R15,R15             RC = 0
         LM    R0,R12,20(R13)      Restore R0-R12
         BR    R14                 Adios
*
FIELDA   DS    CL8
FIELDB   DS    CL8
```

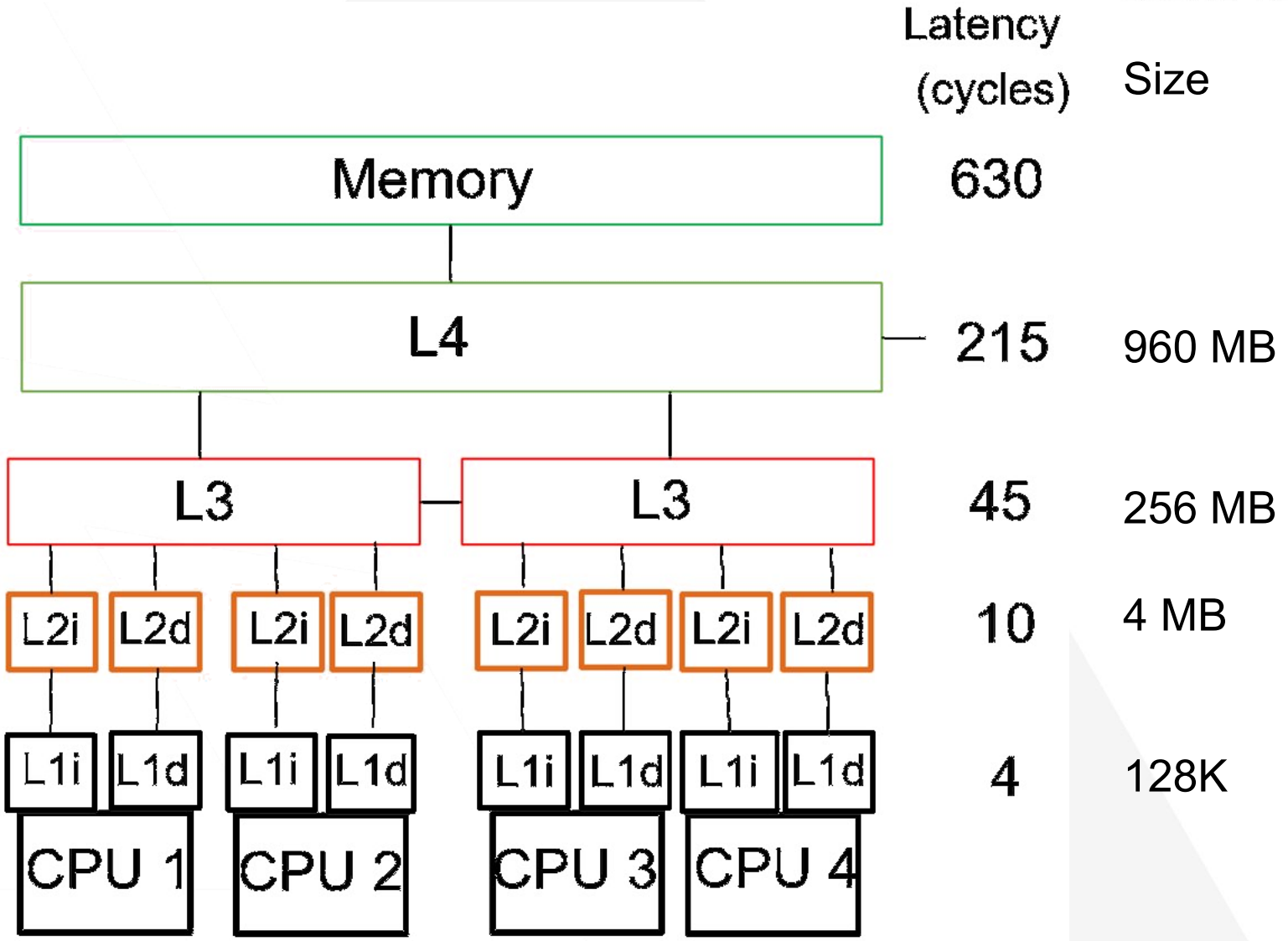
SIIS Violations – Why They Bad

Processor cache hierarchy – z15

Instructions/data must be L1

Cache lines are 256 bytes

SIIS violations invalidate cache lines



SIIS Violations – Identifying Them

- CPU Measurement Facility
  - Non-disruptive monitoring at the hardware level
  - Populates "counters" to store data about the goings-on in the hardware

- Hardware Instrumentation Service (HIS) component
  - Software-side component of CPU MF that gets the data from the hardware
  - Writes data to Unix files and/or SMF records
  - 3 main outputs:
    - PSW sampling
    - Counters data
    - Mapping data

SIIS Violations – Identifying Them

CPU MF Counters

Raw processor data

```
COUNTER SET= BASIC
COUNTER IDENTIFIERS:
   0: CYCLE COUNT
   1: INSTRUCTION COUNT
   2: L1 I-CACHE DIRECTORY-WRITE COUNT
   3: L1 I-CACHE PENALTY CYCLE COUNT
   4: L1 D-CACHE DIRECTORY-WRITE COUNT
   5: L1 D-CACHE PENALTY CYCLE COUNT

START TIME: 2021/04/15 12:09:26  START TOD: D991057ADA26A988
END TIME:   2021/04/15 12:19:26  END TOD:   D99107B70ECAC692
COUNTER VALUES (HEXADECIMAL) FOR CPU 0000 (CPU SPEED = 5208 CYCLES/MIC, TYPE=CP):
   0-   3 00000063EB2BE0E0 0000001A076B999C 00000000D274EE61 0000001C4E8259AD
   4-   7 00000000A4875AD6 00000039E157C6E5 ---------------- ----------------

START TIME: 2021/04/15 12:09:26  START TOD: D991057ADA26A988
END TIME:   2021/04/15 12:19:26  END TOD:   D99107B70ECAC692
COUNTER VALUES (HEXADECIMAL) FOR CPU 0002 (CPU SPEED = 5208 CYCLES/MIC, TYPE=zIIP):
   0-   3 000000018D35D071 00000000502F73DB 00000000014B8969 0000000066E28222
   4-   7 0000000002705A97 000000017EB9739E ---------------- ----------------
```

SIIS Violations – Identifying Them

- CPU MF Extended Counters

  100+ of them

Counter 136 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from the Level-2 Instruction cache.

Counter 157 – A directory write to the Level-1 Data cache directory where the returned cache line was sourced from Off-Drawer Level-4 cache.

Counter 164 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On-Chip Level-3 cache with intervention.

Counter 172 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from Off-Drawer memory.

SIIS Violations – Identifying Them

- CPU MF Extended Counters – Word of Warning

Model-dependent

z14:

Counter 164 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On-Chip Level-3 cache with intervention.

z16 and z17:

Counter 164 – A directory write to the Level-1 Data or Level-1 instruction cache directory where the returned cache line was sourced from an On-Drawer Level-2 cache after using chip level horizontal persistence, Chip-HP hit.

SIIS Violations – Identifying Them

CPU MF Extended Counters

```
COUNTER SET= EXTENDED
COUNTER IDENTIFIERS:
 MODEL DEPENDENT INFORMATION NOT AVAILABLE

START TIME: 2021/04/15 12:09:26  START TOD: D991057ADA26A988
END TIME:   2021/04/15 12:19:26  END TOD:   D99107B70ECAC692
COUNTER VALUES (HEXADECIMAL) FOR CPU 0000 (CPU SPEED = 5208 CYCLES/MIC, TYPE=C
 128- 131 00000000081EF965 0000000010B3427F 0000000395CE9580 000000000083E45E
 132- 135 0000000000000000 00000000636C5FF4 00000000032A79CF 000000012C3E1638
 136- 139 000000009FD6FD6B 00000000104F2504 00000000030AB1F4 0000000545DDC68E
 140- 143 0000000000D87E01 0000000000000000 0000000035286C201 00000033AA90EE37
 144- 147 000000001E4D2722 0000000000294500 000000000FD50F46 000000000059133A
 148- 151 0000000000156FD33 00000000005ED9AA 00000000014BDA3A 00000000029DCD7B
 152- 155 000000000553713B 0000000000035B706 0000000000008AAC9 00000000014CE747
 156- 159 0000000004AC8787 00000000000EC743 00000000013CDEB4 0000000003A2198D
 160- 163 00000000BF9546AE 0000000005DEFD524 0000000026C41F9F 00000000000A66D4
 164- 167 0000000006E6DB7E 0000000000067D7F1 0000000000B26BC1 000000000001ECC
 168- 171 0000000000008E0A1 0000000000CE5734 0000000000001E272 0000000000000387
 172- 175 0000000000000074E8 0000000000000016B 0000000002D65361 000000000000DAB4
 176- 179 0000000000000000 00000000ED581453 0000000044E97646 00000000068DC617
 180- 183 0000000FAA40ABD2 0000000089D81471 0000000C9DE4C90F 00000000D6A189B7
 184- 187 0000000065762BF91 0000000006FBABD23 0000000024A23A64A 0000000DED178C4A
 188- 191 00000009D7E5BC04 00000000989C9FAA 0000000002BC07A0A 00000003A17D9721
 192- 195 0000000845550C50 0000000001E68C121 00000001FBC7743A 000000000000105E
 196- 199 00000004FE5135CF 00000003CB4788EF 0000000011041F63F 0000000000000000
 200- 203 0000000000000000 0000000000074F88 00000062BCD5AF3F 0000000000000000
 204- 207 0000001A06EB5B5D 0000000000000000 0000000056EBAADF 000000000B844C16
 208- 211 0000000009C96FE9E 000000008957418C 0000000005FC13BD 00000000619C7C07
 212- 215 00000002CAAE2E93 0000000017314FCA1 0000000197CBA57FE 00000016A559434B
 216- 219 0000000000000000 00000000000020A7 0000000000023489C 0000000122B3C496
 220- 223 00000000CF3EAC36 0000000000000000 0000000435516975 00000000000BE820
 224- 227 0000000000000000 0000000000000000 0000000011A9B61 000000001E938956
 228- 231 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 232- 235 0000000000000000 00000000FD664B6F 0000000009437BC7 00000004E28BFA48
 236- 239 0000000025EEE199 0000000834BE6EAD 00000001DF30CCDA 00000007B98CC17C
 240- 243 00000000586CB806 00000000A68AABE9 000000641F5EE01E 0000000000000000
 244- 247 0000000000001005 0000000000000000 0000000000000000 0000000000003D82
 248- 251 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 252- 255 0000000000ECC08 0000000000000000 0000000000000000 0000000000000000
```

SIIS Violations – Identifying Them

"SIIS Percentage"

Processor	SIIS Indicator %	Description
zEC12 / zBC12	E130 / B4 * 100%	D Writes sourced with L2 intervention / D Writes
z13 / z13s	E163 / B2 * 100%	I Writes sourced with L3 intervention / I Writes
z14 / ZR1	E164 / B2 * 100%	I Writes sourced with L3 intervention / I Writes
z15	E164 / B2 * 100%	I Writes sourced with L3 intervention / I Writes

Counter 164 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On-Chip Level-3 cache with intervention.

```
COUNTER SET= BASIC
COUNTER IDENTIFIERS:
   0: CYCLE COUNT
   1: INSTRUCTION COUNT
   2: L1 I-CACHE DIRECTORY-WRITE COUNT
```

SIIS Violations – Identifying Them

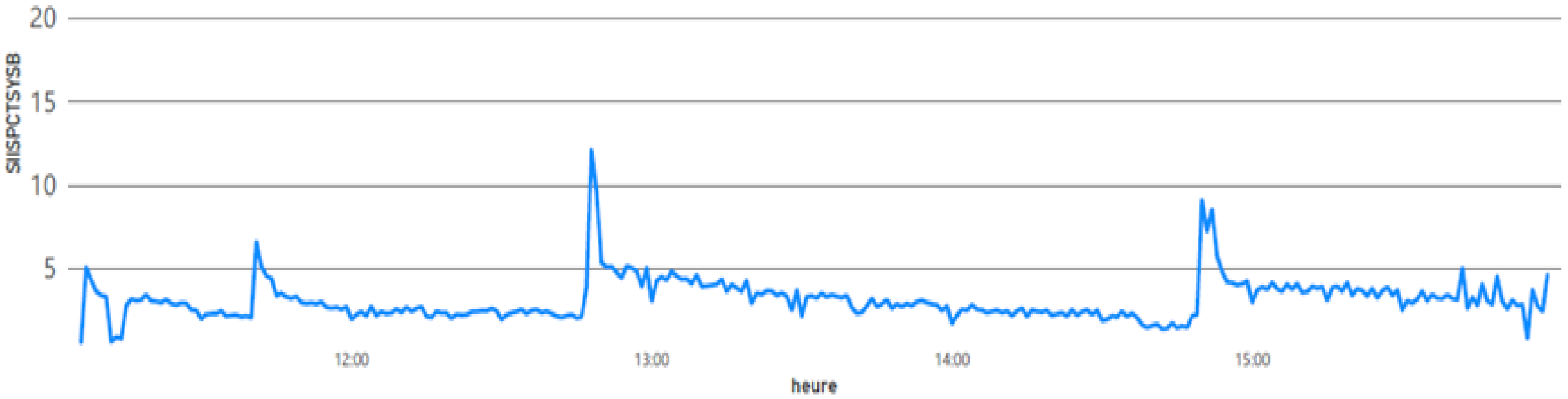
"SIIS Percentage“ (including z16 and z17)

Processor	SIIS Indicator %	Description
zEC12 / zBC12	E130 / B4 * 100%	D Writes sourced with L2 intervention / D Writes
z13 / z13s	E163 / B2 * 100%	I Writes sourced with L3 intervention / I Writes
z14 / ZR1	E164 / B2 * 100%	I Writes sourced with L3 intervention / I Writes
z15	E164 / B2 * 100%	I Writes sourced with L3 intervention / I Writes
z16	E170 / B2 * 100%	I Writes sourced with L2 intervention / I Writes
z17	E170 / B2 * 100%	I Writes sourced with L2 intervention / I Writes

Counter 170 – A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from the requestor's Level-2 cache with intervention.

```
COUNTER SET= BASIC
COUNTER IDENTIFIERS:
   0: CYCLE COUNT
   1: INSTRUCTION COUNT
   2: L1 I-CACHE DIRECTORY-WRITE COUNT
```

SIIS Violations – SIIS Percentage



SIIS Description	SIIS Indicator %	Action
Noise – it will never be 0%	< 2%	None
Minimal SIIS impact	2% < 5%	Low Priority but potential MSU savings
Noteworthy SIIS impact	5% < 10%	Medium Priority – Investigate and Remediate
Considerable SIIS impact	>= 10%	Top Priority – Investigate and Remediate

SIIS Violations – Identifying Them

Hopefully it's a hot spot

Performance Monitor for PGM002:

Code section	Length of section	CPU Time
000000	64	.03
000040	64	.00
000080	64	3.0
...	...	...
0008C0	64	1.12
...	...	...
Program totals		5.75

SIIS Violations From Hot Spot – Start of Program

SHARE
EDUCATE • NETWORK • INFLUENCE

```
********************************************************** Top of Data ****
PGM002    CSECT ,
FASTENT   B    FASTOK                  Allow "fast path"
SLOWENT   B    SLOWONLY                Alternate entry
          B    ABEND                   Error
*
FAST_PATH DC    X'FF'                <--- CAN FAST PATH BE USED? 0=YES 1=NO
*
SAVR0R15  DS    0F                      Local copy of caller's regs
SAVR0     DS    F
...
SAVR15    DS    F
*
FASTOK    MVI  FAST_PATH,YES            Allow fast path
          B    STARTUP
SLOWONLY  MVI  FAST_PATH,NO             Don't allow fast path
          B    STARTUP
ABEND     ABEND 1503,REASON=4,DUMP      Invalid entry
*
STARTUP   DS   0H
          STM  R14,R12,12(R13)          Save caller's regs
          STM  R0,R15,SAVR0R15          Save them for later reference
...
```

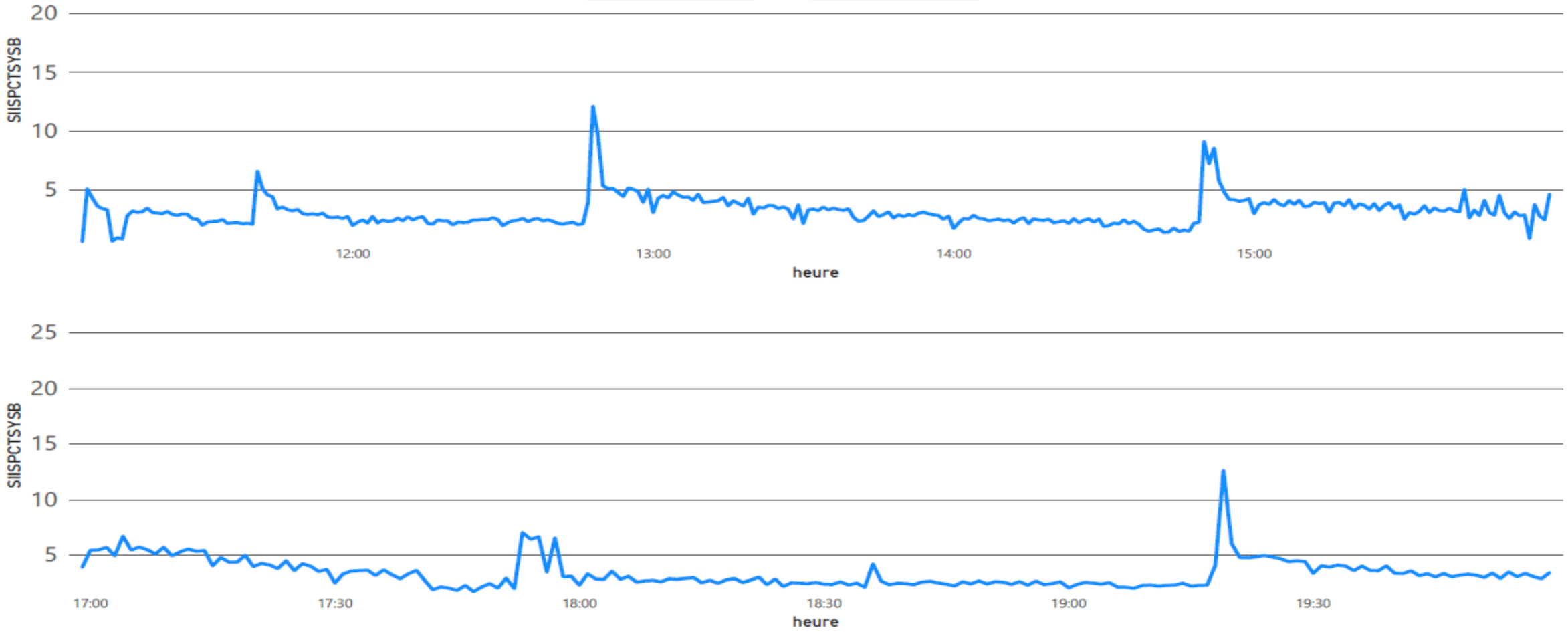
SIIS Violations From Hot Spot

Farther down in program:

```
PGM002    CSECT ,
*  ... A bit of code
*
          BASSM R14,R15                  Call target module
*
*  ... More code
          BSM   0,R14                    Return to caller
RRCODES   DS    0F
RET_CODE  DS    F
RSN_CODE  DS    F
*
SUBRTN    DS    0H
```

SIIS Violations – Identifying Them

- Fixed the violations – Before/after report from customer:



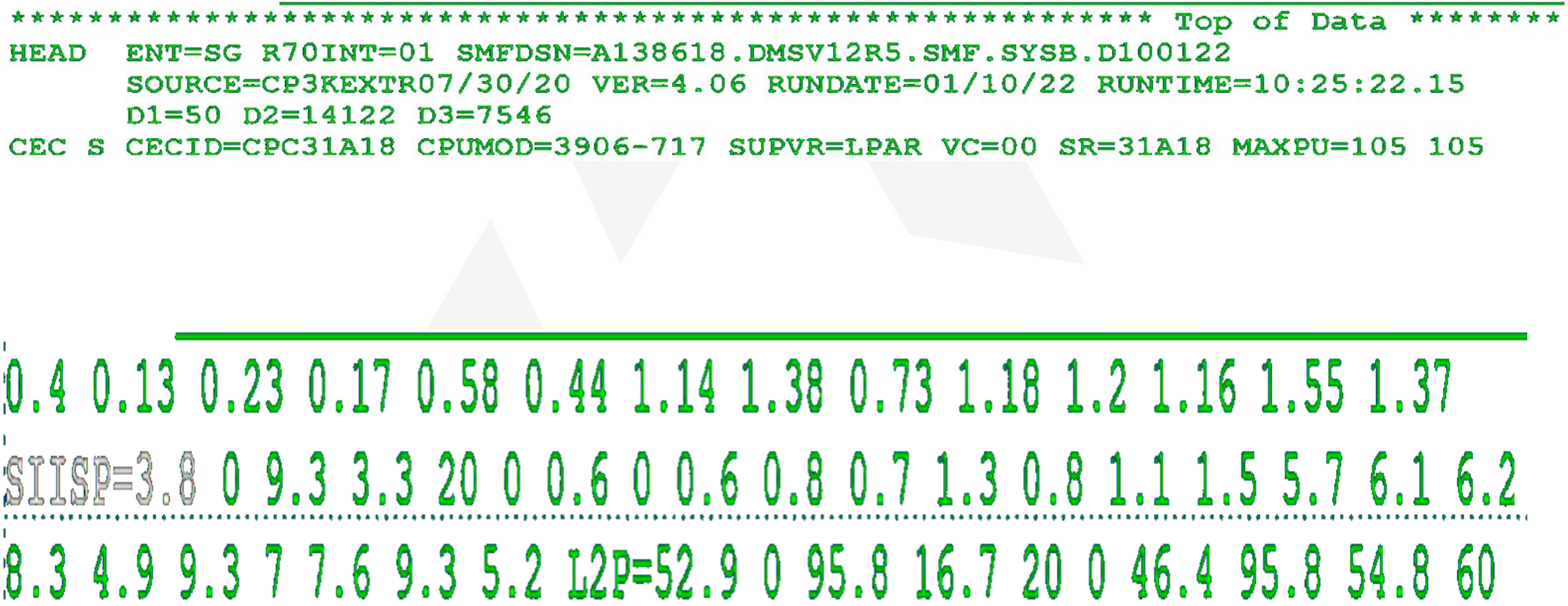
SIIS Violations – Identifying Them

- Narrowing in on a time frame
- CP3KEXTR Report
  - "z/OS Data Extraction Program“

- No-charge tool that IBM supplies

- Summarizes SMF data
  - Produces EDF (Enterprise Data File)
  - Updated regularly

CP3KEXTR EDF Report



SIIS Violation – Narrowing in on Time Frame

- Apart from the counters data, there's PSW sampling data
  - Hardware Instrumentation Services (HIS) component of CPU MF

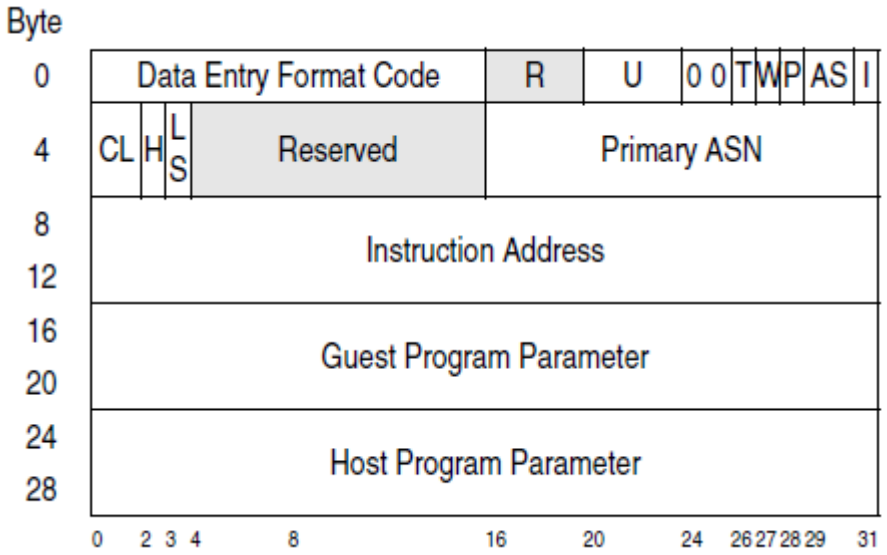
- Samples collected based at a specified frequency
  - # of samples to take in a 1-minute interval
  - Hardware-based sampling

- Output written to Unix files

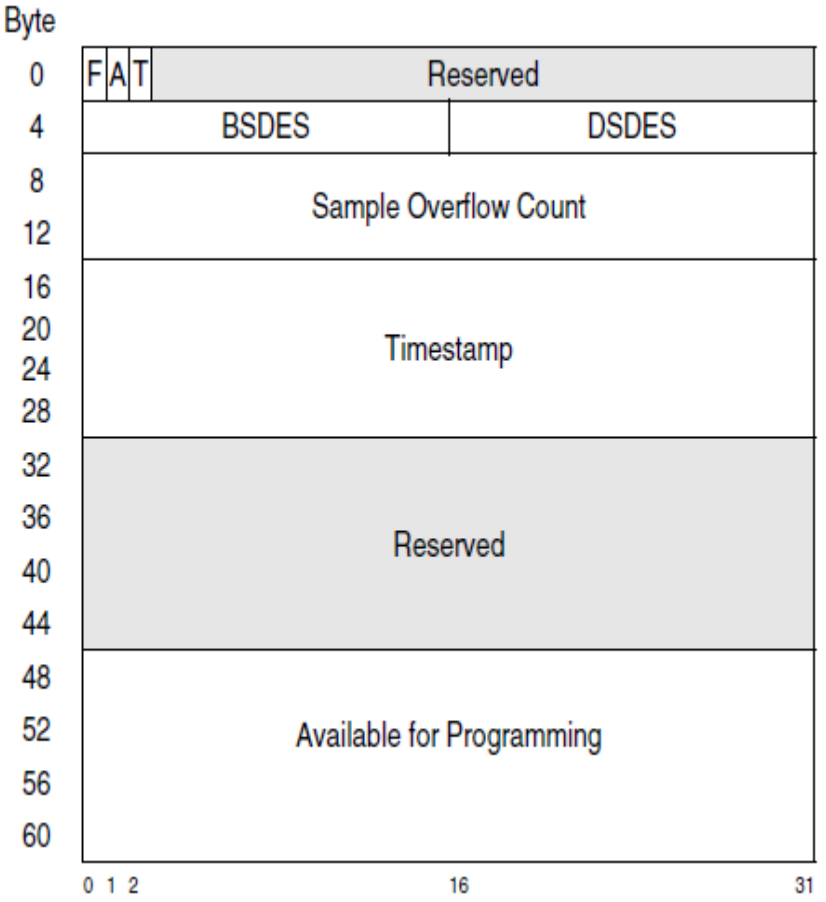
CPU Measurement Facility - HIS Sampling data

Sampling record
Sampling record
...
Sampling record
Sampling record
Trailer record
Sampling record
...
Sampling record
Trailer record

Sampling record



Trailer record

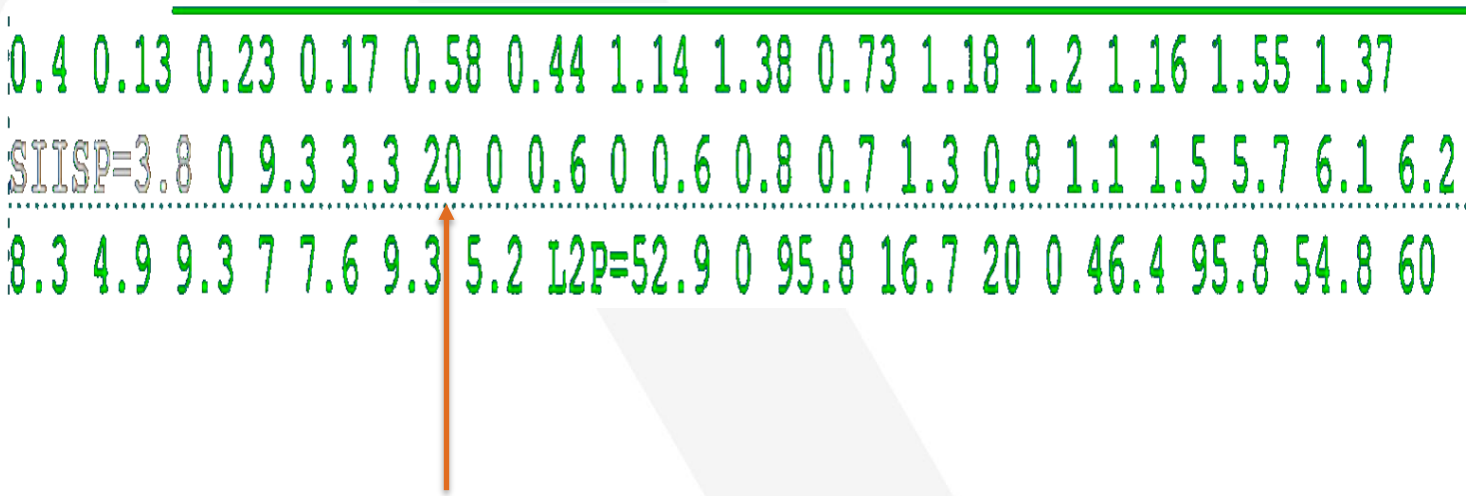


HIS Sampling Data – Time Period of Interest

- Making Sense of the Data

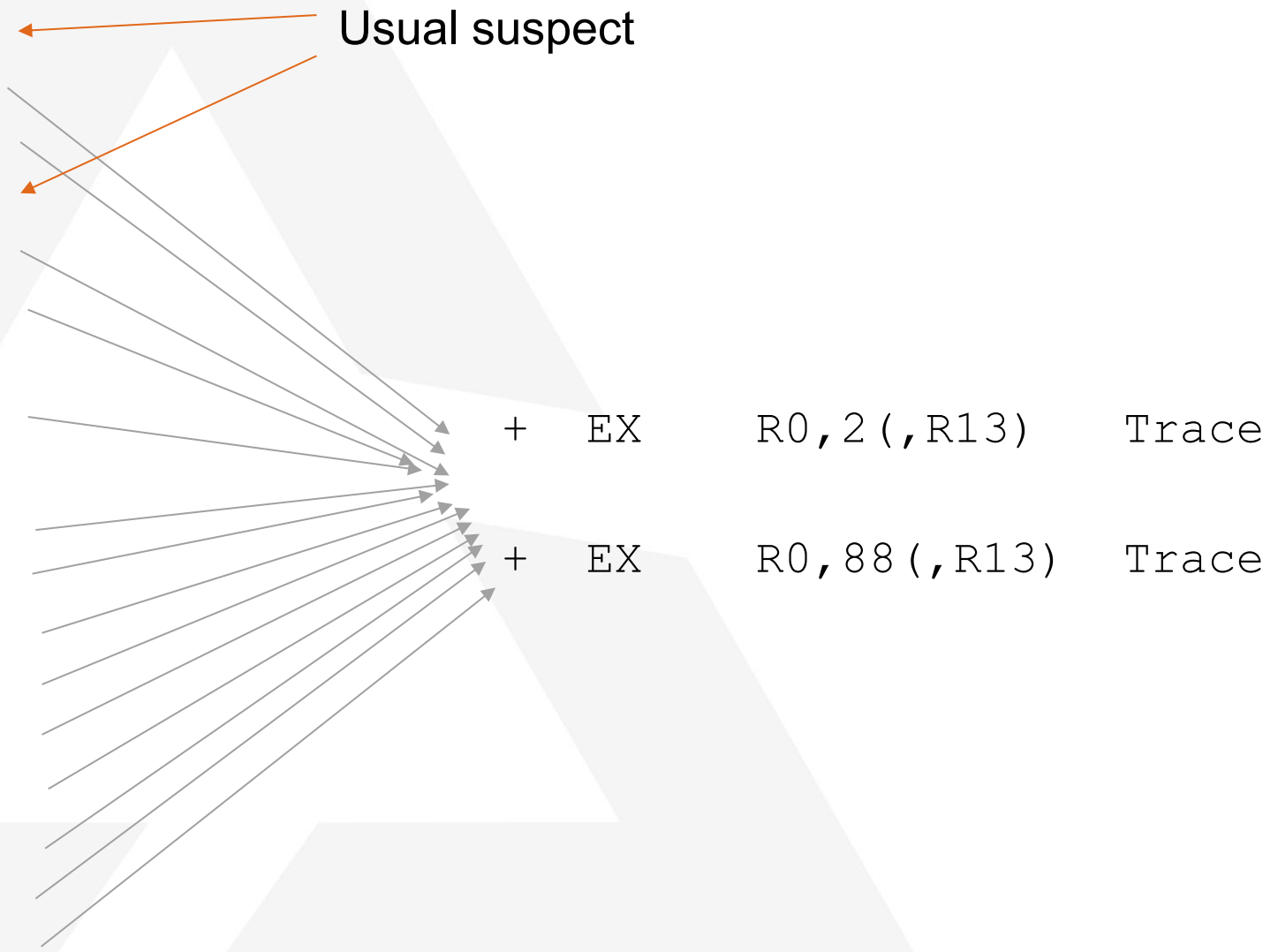
Samples	Module	CSECT	Offset
29165	PGM001	PGM001	000002C0
12485	PGM500	PGM500	00000502
10978	PGM500	PGM500	000003BA
10362	PGM001	PGM001	000002D4
9247	PGM501	PGM501	000002A0
8367	PGM500	PGM500	000006C4
5416	PGM002	PGM002	000000B4
5044	PGM502	PGM502	00000610
4391	PGM001	PGM001	00000354
4081	PGM500	PGM500	00000326
3644	PGM503	PGM503	00000290
3556	PGM500	PGM500	000004FE
3404	PGM503	PGM503	0000026E
3109	PGM500	PGM500	00000624
3093	PGM500	PGM500	0000057A
3054	PGM504	PGM504	000002DE
2698	PGM505	PGM505	00000168
2168	PGM506	PGM506	000002CE

Home-grown program



HIS Sampling Data – Time Period of Interest

Samples	Module	CSECT	Offset
29165	PGM001	PGM001	000002C0
12485	PGM500	PGM500	00000502
10978	PGM500	PGM500	000003BA
10362	PGM001	PGM001	000002D4
9247	PGM501	PGM501	000002A0
8367	PGM500	PGM500	000006C4
5416	PGM002	PGM002	000000B4
5044	PGM502	PGM502	00000610
4391	PGM001	PGM001	00000354
4081	PGM500	PGM500	00000326
3644	PGM503	PGM503	00000290
3556	PGM500	PGM500	000004FE
3404	PGM503	PGM503	0000026E
3109	PGM500	PGM500	00000624
3093	PGM500	PGM500	0000057A
3054	PGM504	PGM504	000002DE
2698	PGM505	PGM505	00000168
2168	PGM506	PGM506	000002CE

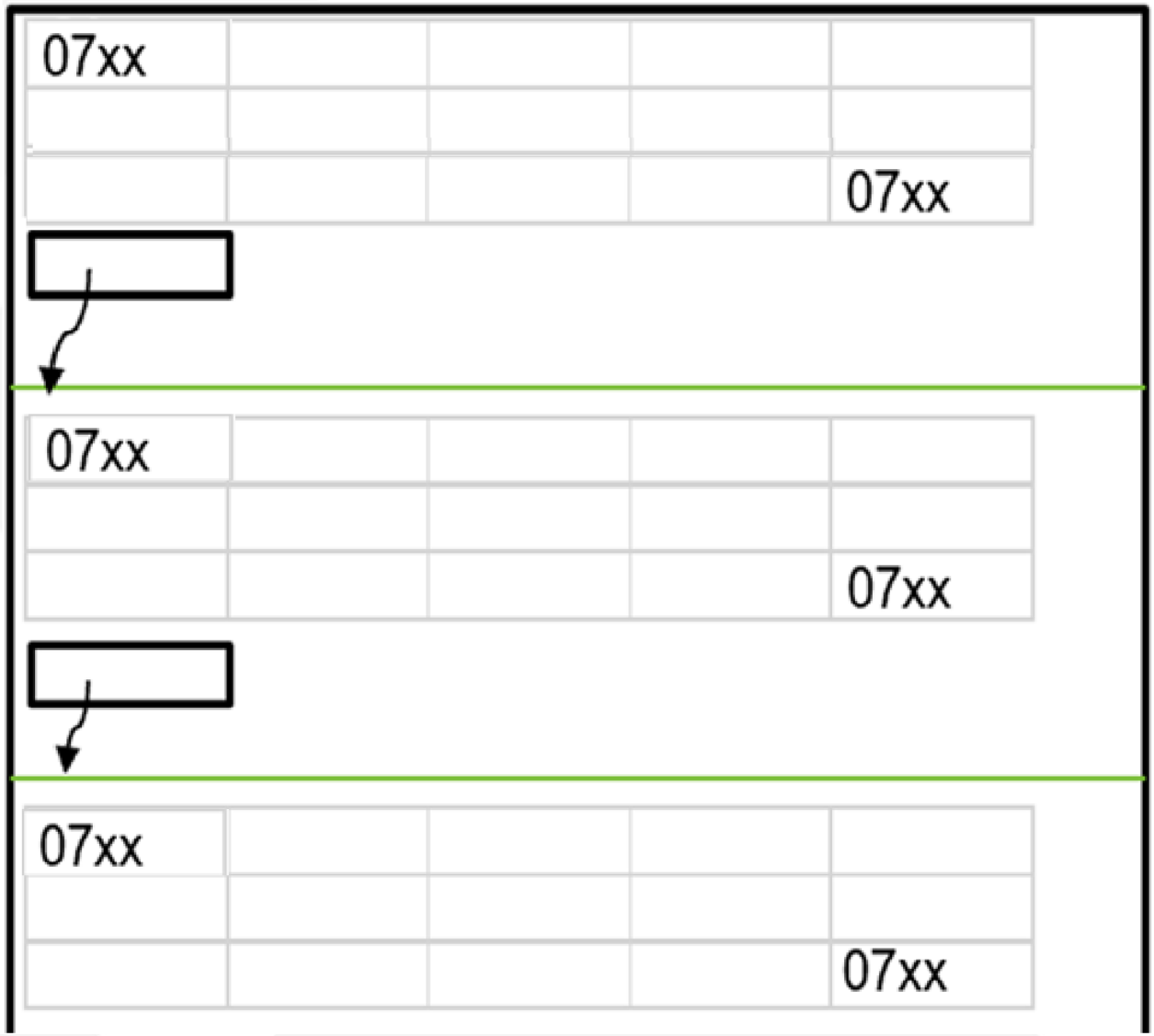


The Targets of the EXs

Dynamic Storage Area

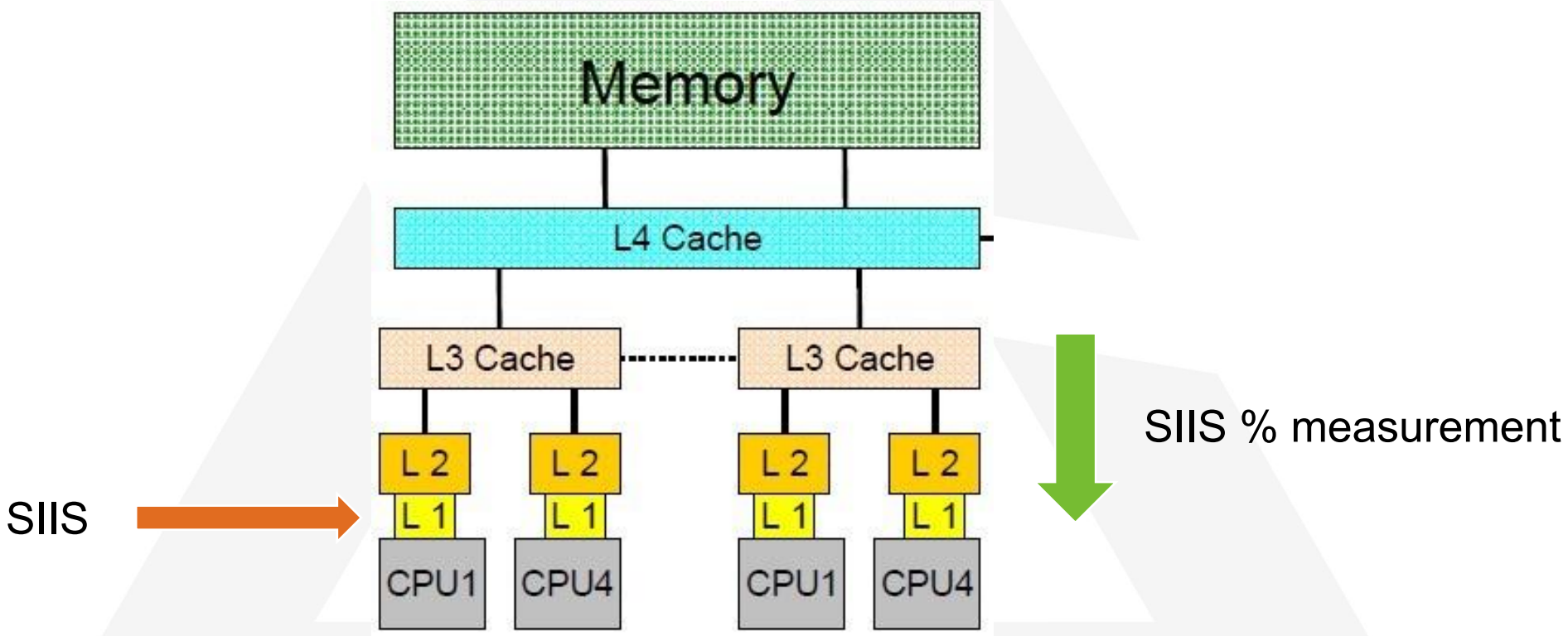
"Trap points" for tracing have either:

- 0Axx
- 07xx



Placing Instructions in Data Areas – SIIS?

SIIS is usually thought of as the other way around



Instruction / Data Mixing - Fix

```
          CLI    EX_TARGET,X'07'
          JE     NO_TRACE
          EX     R0,EX_TARGET
NO_TRACE  DS     0H
```

- New release was in development

- Q: Issue with pre-fetching / branch prediction?
  - A: No

- So how much did this help???

Code Change Results

Job Run	CPU - Prior version	CPU - New version		% Decrease	
1	25.39	7.75		69	
2	26.46	8.26		69	
3	24.46	8.91		64	
4	24.14	7.95		67	
5	24.58	8.21		67	
6	22.85	8.46		63	
7	26.6	7.7		71	

- Results not typical
  - Code was:
    - Seriously violating rules
    - Extremely pervasive

General Considerations

Obviously, don't mix instructions and data

Sneaky: R/W variables at end of program

```
000278 00000384                      734+PDBO0045 DC    A(PDB0005)              <=== PARAMETER DESC. BLOCK
                                     736          PGMLTORG ,          MAKE LITERAL POOL AT LEAST 1 CACHE LINE
                     0027C           737+PGMLT1_0051       EQU *                 Beginning of literal pool
000280                               738+         LTORG
000280 F1F1F1F1F1F0F0F4              739                   =C'11111004'
000288 0004                          740                   =H'4'
00028A 0000                          741                   =H'0'
                     0028C           742+PGMLT2_0051       EQU *                 End of literal pool
                     00001 00010     743+PGMLT_EQU  EQU 1,PGMLT2_0051-PGMLT1_0051       Literal pool length
00028C                               744+         DS    XL(256-16)              Pad area for cache line
00037C 0000                          746 PG004RC  DC    H'0'
```

General Considerations

SHARE
EDUCATE • NETWORK • INFLUENCE

- Target of EX instructions
  - Instruction fetch / branch
  - “Close” is good

- Minimize # of cache lines
  - Split out infrequently used code paths

Sources

- David Hutton. “IBM Z / LinuxONE System Processor Optimization Primer”

https://www.ibm.com/support/pages/ibm-z-linuxone-system-processor-optimization-primer

- Identifying "Store Into Instruction Stream" (SIIS) Inefficiency by Using CPU MF Counters

https://www.ibm.com/support/pages/identifying-%E2%80%9Cstore-instruction-stream%E2%80%9D-siis-inefficiency-using-cpu-mf-counters

- Bob Rogers. “How Do You Do What You Do When You’re a z196 CPU”. SHARE in Anaheim, 2011. (Technical content & graphics)

https://watsonwalker.com/publications/presentations/

- CPU Measurement Facility:

Load-Program-Parameter and the CPU MF & CPU MF Extended Counters

Join a Mainframe Technical Exchange

Apr. 21-23, 2026	June 23-25, 2026	Oct. 20-22, 2026
European MTE Prague, Czech Republic	Virtual MTE Held Virtually	North American MTE Plano, TX

- Network with peers and Mainframe technical experts
- Participate in technical how-to sessions and hands-on workshops
- No registration fee!