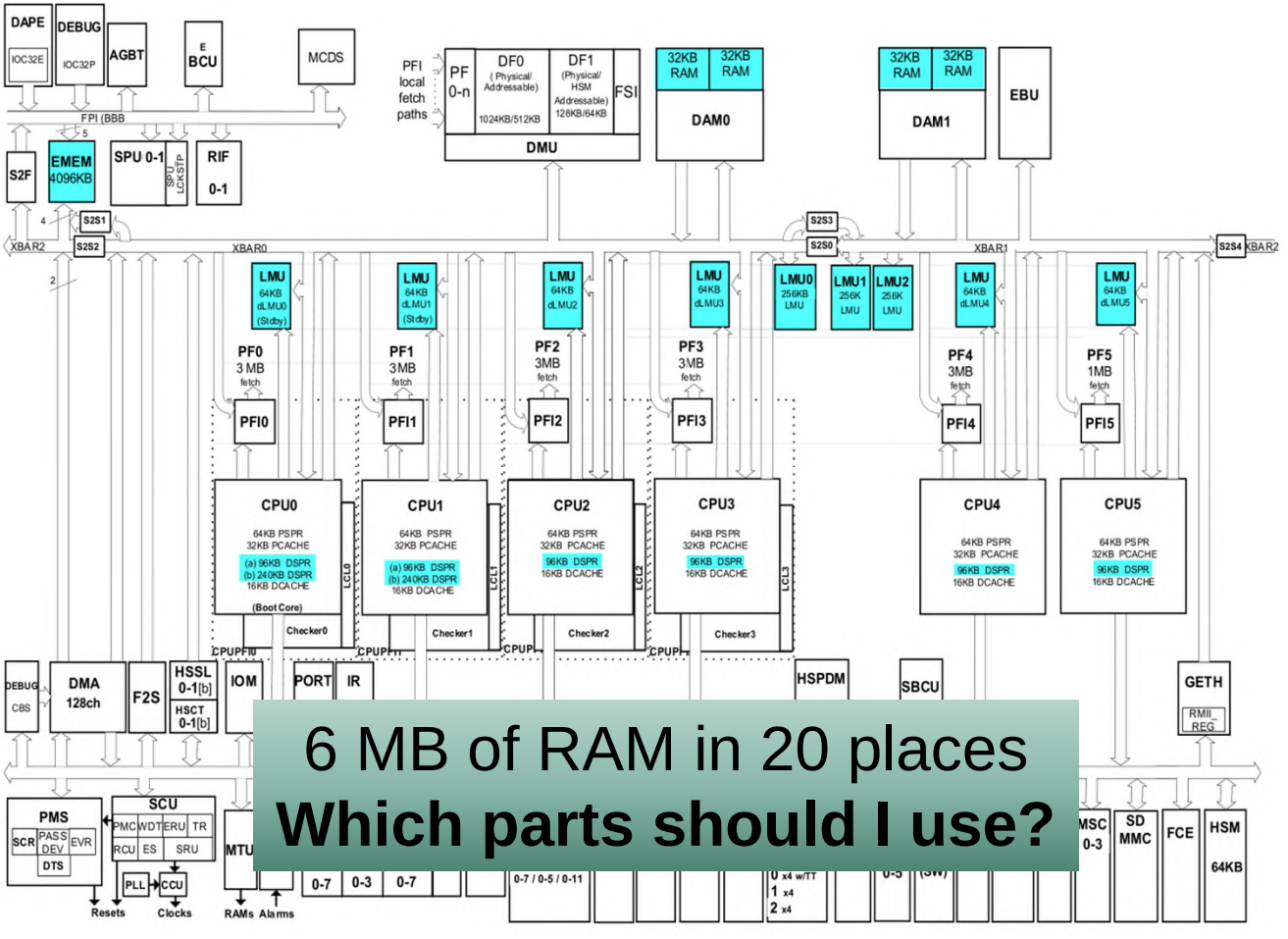
AURIX™ RAM and Performance

April 26 2021

TC397XX Architecture



6 MB of RAM in 20 places

Which parts should I use?

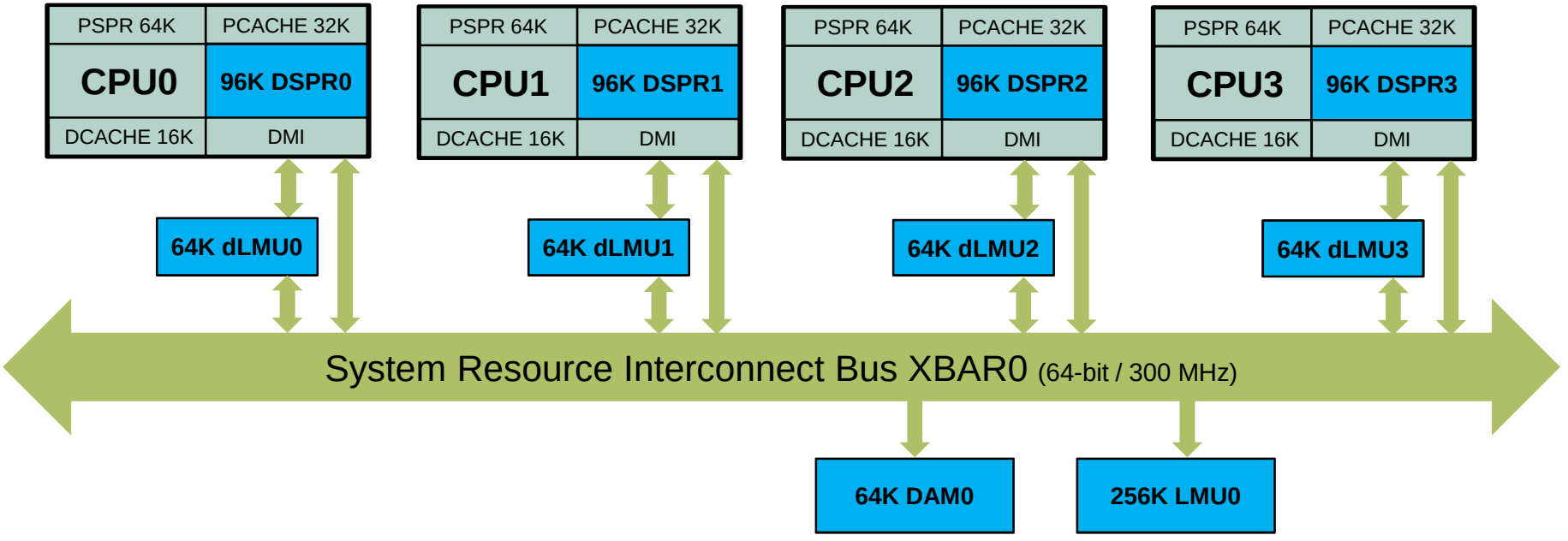
Agenda

1. AURIX™ Architecture
2. TriCore™ Architecture
3. Data Cache and Cache Coherency
4. Maximizing Performance

Agenda

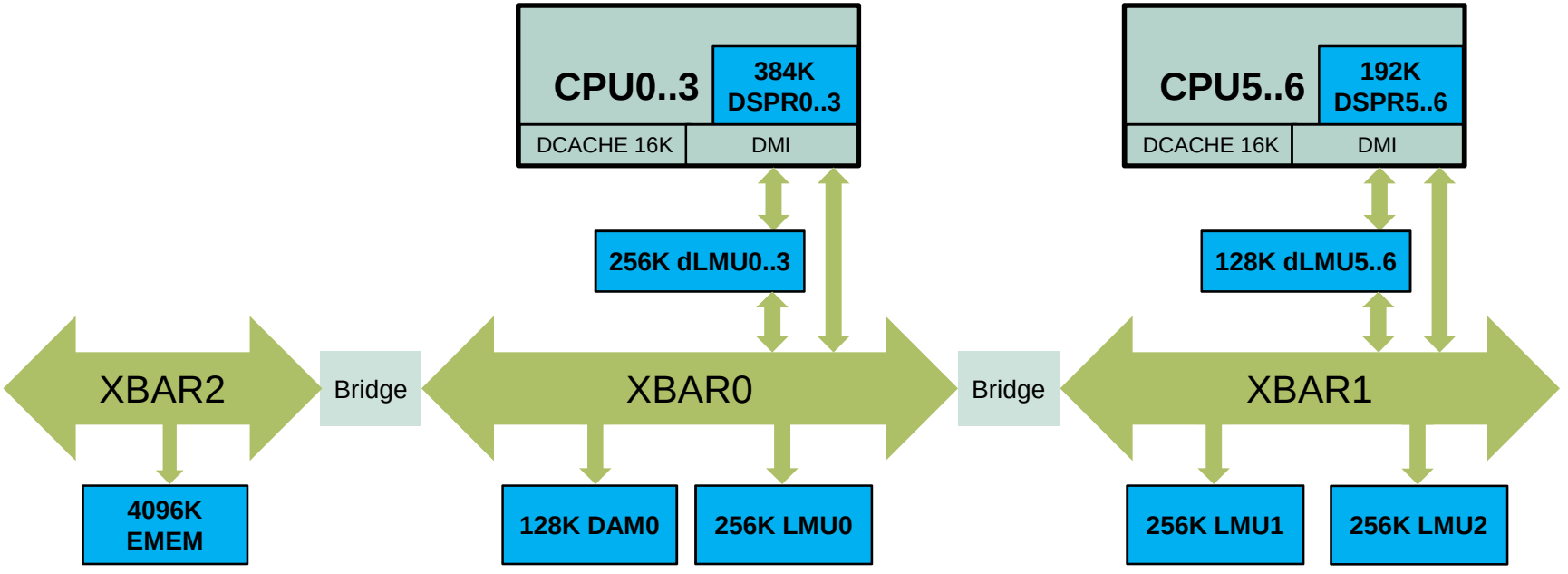
1. AURIX™ Architecture
2. TriCore™ Architecture
3. Data Cache and Cache Coherency
4. Maximizing Performance

AURIX™ Architecture
TC397XX XBAR0 Cluster RAM-centric view



- Each CPU has local **Data Scratchpad Memory** (**DSPR**) – 0 wait states (CPU clocks)
- Each CPU has a **Distributed Local Memory Unit** (**dLMU**) – 0 cycle read, 2 write
- Accessing memories over the SRI bus is slower

AURIX™ Architecture
TC397XX RAM-centric view



For best performance, use memories with the shortest path from the CPU!

AURIX™ Architecture
TC397XX Memory Map - RAM

Address	Memory
0x10000000-0x10017FFF	CPU5 DSPR
0x30000000-0x30017FFF	CPU4 DSPR
0x40000000-0x40017FFF	CPU3 DSPR
0x50000000-0x50017FFF	CPU2 DSPR
0x60000000-0x60017FFF	CPU1 DSPR
0x70000000-0x70017FFF	CPU0 DSPR
0x90000000-0x9000FFFF	CPU0 dLMU
0x90010000-0x9001FFFF	CPU1 dLMU
0x90020000-0x9002FFFF	CPU2 dLMU
0x90030000-0x9003FFFF	CPU3 dLMU

Address	Memory
0x90040000-0x9007FFFF	LMU0
0x90080000-0x900BFFFF	LMU1
0x900C0000-0x900FFFFF	LMU2
0x90100000-0x9010FFFF	CPU4 dLMU
0x90110000-0x9011FFFF	CPU5 dLMU
0x90400000-0x9041FFFF	DAM
0x98E00000-0x993FFFFF	EMEM

Note: dLMU and LMU are contiguous (easy!)

AURIX™ Architecture
RAM Access Times (CPU cycles)

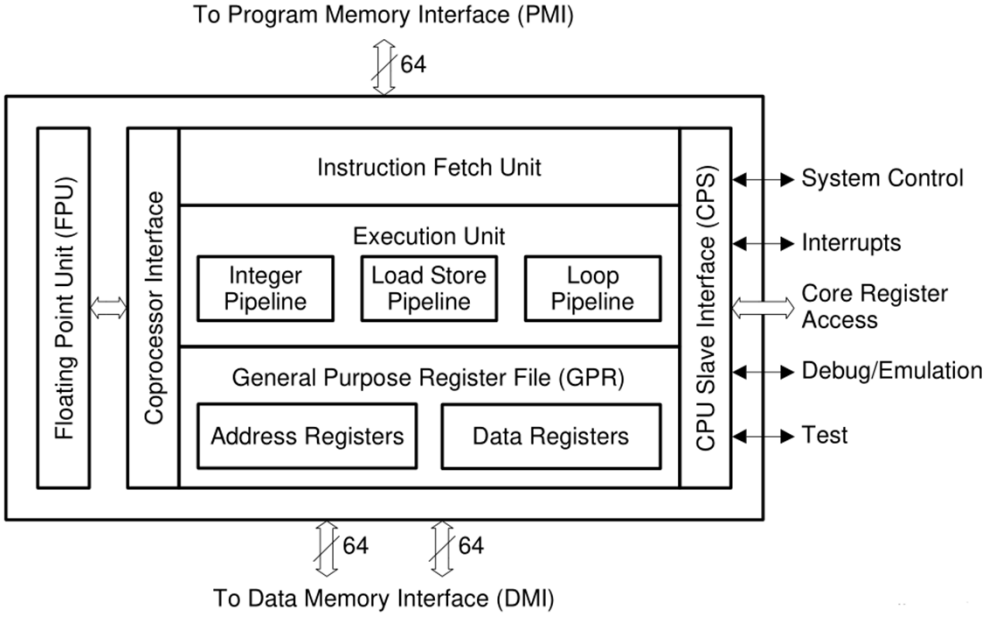
	Local CPU	Local SRI	Remote SRI
Read DSPR	0	7	10
Write DSPR	0	5	5
Read dLMU	0	7	10
Write dLMU	2	5	5
Read PSPR	5	5	5
Write PSPR	5	5	5
Read LMU	-	7	10
Write LMU	-	5	5
Read EMEM	-	-	14,15
Write EMEM	-	-	9
Read DAM	-	10	13
Write DAM	-	7	7

Local SRI = same XBAR (see slide 6)

Agenda

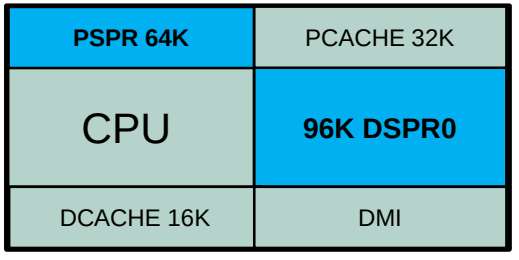
1. AURIX™ Architecture
2. TriCore™ Architecture
3. Data Cache and Cache Coherency
4. Maximizing Performance

TriCore™ Architecture
PMI and DMI



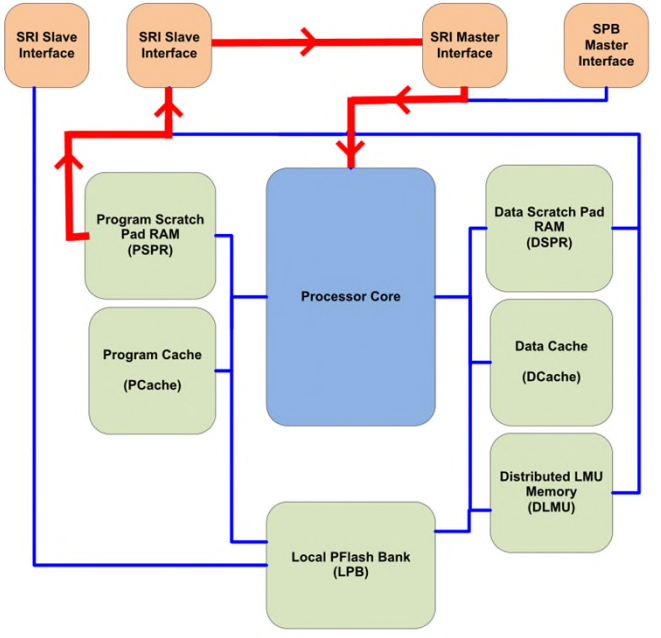
- TriCore™ is a **Harvard architecture**
- **Instructions** pass through the ***Program Memory Interface***
- **Data** passes through the ***Data Memory Interface***

TriCore™ Architecture
PSPR and DSPR



- Each CPU has 64K of **Program Scratchpad RAM (PSPR)**
  - 0-wait state for **instructions**
  - Faster than fetching from PFLASH
  - Best used for trap table, interrupts, OS swap, commonly used code
- Each CPU has 96K of **Data Scratchpad RAM (DSPR)**
  - 0 wait state access for **data**

TriCore™ Architecture
PMI, DMI, SRI

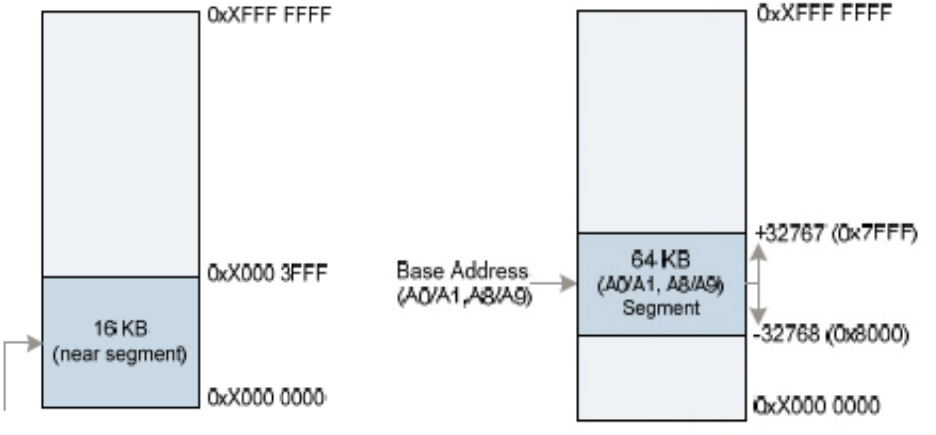


- It is possible to use PSPR as RAM, **but…**
  - PSPR is connected to the PMI, not the DMI
  - The DMI reads PSPR via the SRI bus, which is slower

- Tricore CPUs support 3 **addressing modes**.
  - Default: **far** data (anywhere)
  - Absolute / **Near** data: First 16kB in each segment.
  - **Small** / Literal data: Base + Offset using A0/A1 & A8/A9 registers (64kB).
  - Initialized during startup code / cinit.

Table 17 Addressing Modes

Addressing Mode	Syntax	Effective Address	Instruction Format
Absolute	Constant	{offset 18[17:14], 14'b00000000000000, offset 18[13:0]}	ABS
Base + Short Offset	A[n]offset	A[n]+sign_ext(offset10)	BO
Base + Long Offset	A[n]offset	A[n]+sign_ext(offset16)	BOL



TriCore™ Architecture
Data Addressing – efficiency

How many instructions does this C code generate?

```
83
84      xi = 2;
85      yi = xi;
86
```

Default __far addressing:

5 instructions

```
84      xi = 2;
P:80000938 F7000091        movh.a   a15,#0x7000
P:8000093C 203CF259        st.w     [a15]0xBC,d2
85      yi = xi;
P:80000940 203CFF19        ld.w     d15,[a15]0xBC
P:80000944 F7000091        movh.a   a15,#0x7000
P:80000948 2038FF59        st.w     [a15]0xB8,d15
```

Small / __a0:

3 instructions

```
84      xi = 2;
P:80000940 08040259        st.w     [a0]-0x7FFC,d2
85      yi = xi;
P:80000944 08040F19        ld.w     d15,[a0]-0x7FFC
P:80000948 08000F59        st.w     [a0]-0x8000,d15
```

__near:

3 instructions

```
84      xi = 2;
P:80000940 000472A5        st.w     0x70000004,d2   ; xi,d2
85      yi = xi;
P:80000944 00047F85        ld.w     d15,0x70000004  ; d15,xi
P:80000948 00007FA5        st.w     0x70000000,d15  ; yi,d15
```

Tasking compiler:
__far
__a0, __a1, __a8, __a9
__near

GNU:

.bss
.sdata
.sbss
.sdata2
.zdata
.zrodata
.zbss

Faster Code!
Less PFLASH space!

- CPU writes outside of DSPR are handled with **Store Buffers**
  - The CPU keeps executing while the Store Buffer handles the write
  - This is why write errors cause a Data **Asynchronous** Error Trap, while read errors cause a Data **Synchronous** Error Trap
- Each CPU has **six** store buffers to improve performance
- You can read back from the same location or use the DSYNC instruction to wait for the store buffers to complete

Agenda

1. AURIX Architecture
2. TriCore Architecture
3. Data Cache and Cache Coherency
4. Maximizing Performance

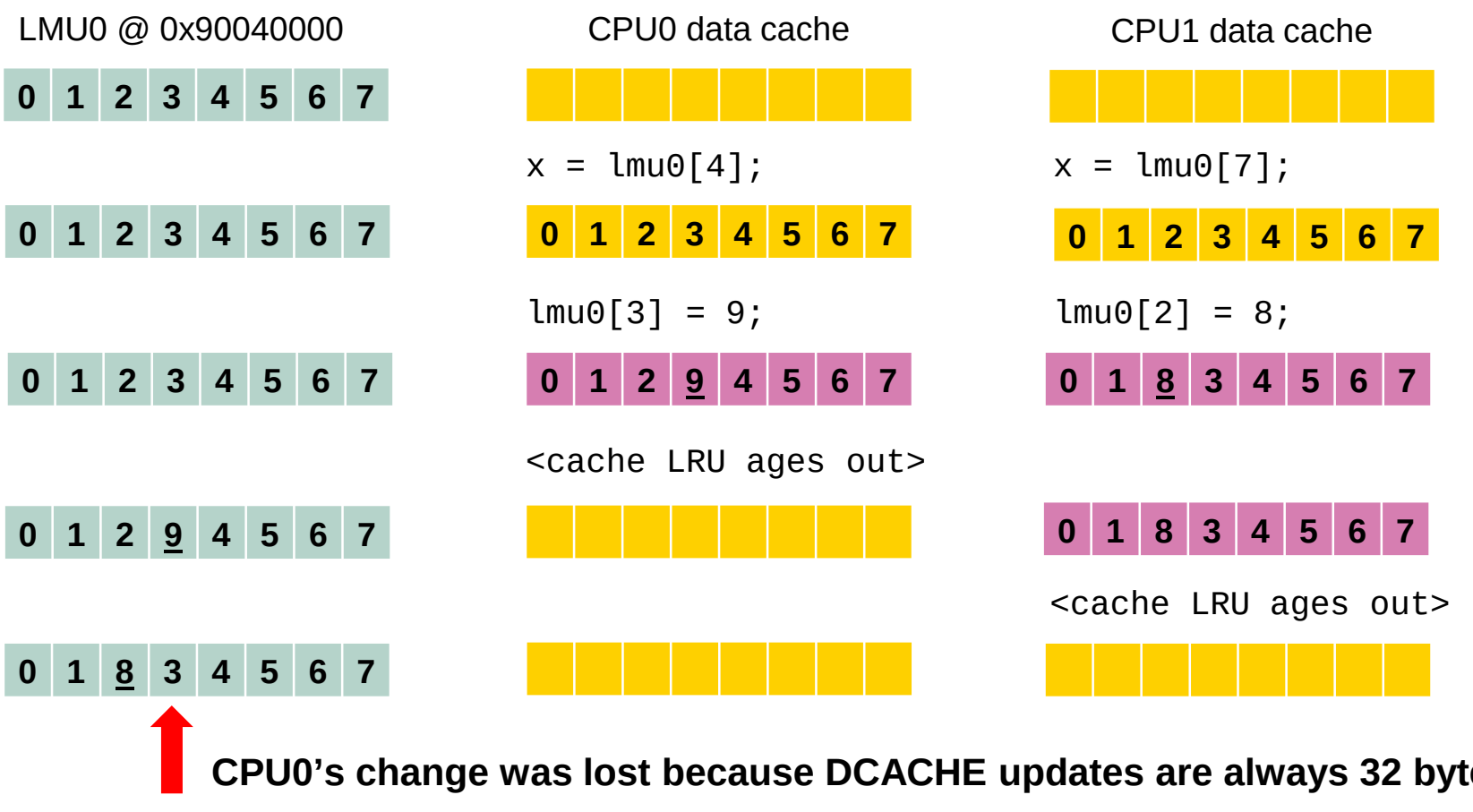
Data Cache and Cache Coherency

- Each CPU has 16K of **Data Cache**
- Data cache consists of 512 entries, **32 bytes** each
- Data cache is a 2-way set associative cache with a Least Recently Used replacement algorithm
- Data Cache is used for all RAM outside of local DSPR but can also help with caching constant data from PFLASH
- TriCore™ emphasizes consistent realtime performance and **does not have automatic cache coherency**

Data Cache and Cache Coherency
Where and When

- The **DCON0** register specifies which segments are cacheable
  - By default, DCON0 specifies segment 8 (PFLASH) and segment 9 (LMU)
- LMU RAM can be accessed through the data cache (segment 9), or bypass the data cache (segment B)
- DSPR memories by default are not cacheable
  - If you’re not using one or more CPUs, consider making those DSPR segments cacheable

Data Cache and Cache Coherency
Problem example



CPU0's change was lost because DCACHE updates are always 32 bytes!

Data Cache and Cache Coherency
3 Ways to Stay out of Trouble

- **Method 1: Use *cachea* instruction**
  - For **writing**, use *cachea.wi* to force updates to write through
  - For **reading**, use *cachea.i* to invalidate and fetch new data

- **Method 2: Use non-cached addressing**
  - Each CPU can **use** the data cache to address LMU RAM in segment 9
  - Each CPU can **bypass** the data cache for LMU RAM with segment B
  - Remember that data cache writes are always 32 bytes wide, so do not place variables updated by more than one CPU in the same 32 byte-aligned area

- **Method 3: Restrict data cache to PFLASH only**
  - Data cache will only apply to PFLASH if your application changes CPUx_PMA0 to 0x100 (segment 8 only)
  - Worst performance option, but easy to implement

Agenda

1. AURIX Architecture
2. TriCore Architecture
3. Data Cache and Cache Coherency
4. Maximizing Performance

Maximizing Performance

- Keep data closest to the CPU that needs it
  - Prioritize RAM use by access times: local DSPR is best, dLMU second, etc.
- Use small/near/a0/a1/a8/a9 whenever possible
  - Less instructions => faster execution + less PFLASH
- Use PSPR to reduce PFLASH stalls
  - Put Trap Vector table in PSPR (most OSes use SYSCALL a lot)
  - OS task swap code, Interrupt Vector table + code, frequently used code
- Avoid data cache coherency problems
  - Unpredictable and subtle
  - Use one of the methods outlined in this presentation and **be consistent**!

Part of your life. Part of tomorrow.