System z Architecture

Joachim von Buttlar

System z Firmware Development
IBM Deutschland Research & Development GmbH

joachim_von_buttlar@de.ibm.com

Agenda

- Introduction
- Modes of Operation
- Firmware Layers
- Register Sets
- Storage
- Interrupts
- Timing Facilities
- Instructions
- Storage Protection
- Virtual Storage
- Multiprocessing
- Input/Output
- Partitioning and Virtualization
- Parallel Sysplex

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

AIX*	IMS	S/390*	z10 EC
APPN*	InfiniBand*	Sysplex Timer*	z/Architecture*
BladeCenter*	Multiprise*	System/390*	z/OS*
CICS*	OS/2*	System p*	z/VM*
DB2*	Parallel Sysplex*	System x*	z/VSE
DataPower*	Power*	System z*	zEnterpriise
DS8000*	POWER*	System z9*	zSeries*
e business(logo)*	POWER7	System z10*	
ESCON*	Power Architecture*	VSE/ESA	
eServer	PowerVM	WebSphere*	
FICON*	PR/SM	X-Architecture*	
GDPS*	Resource Link*	z9*	
HiperSockets	Redbooks*	z10	
IBM*	REXX	z10 Business Class	
IBM (logo)*	RMF	z10 BC	

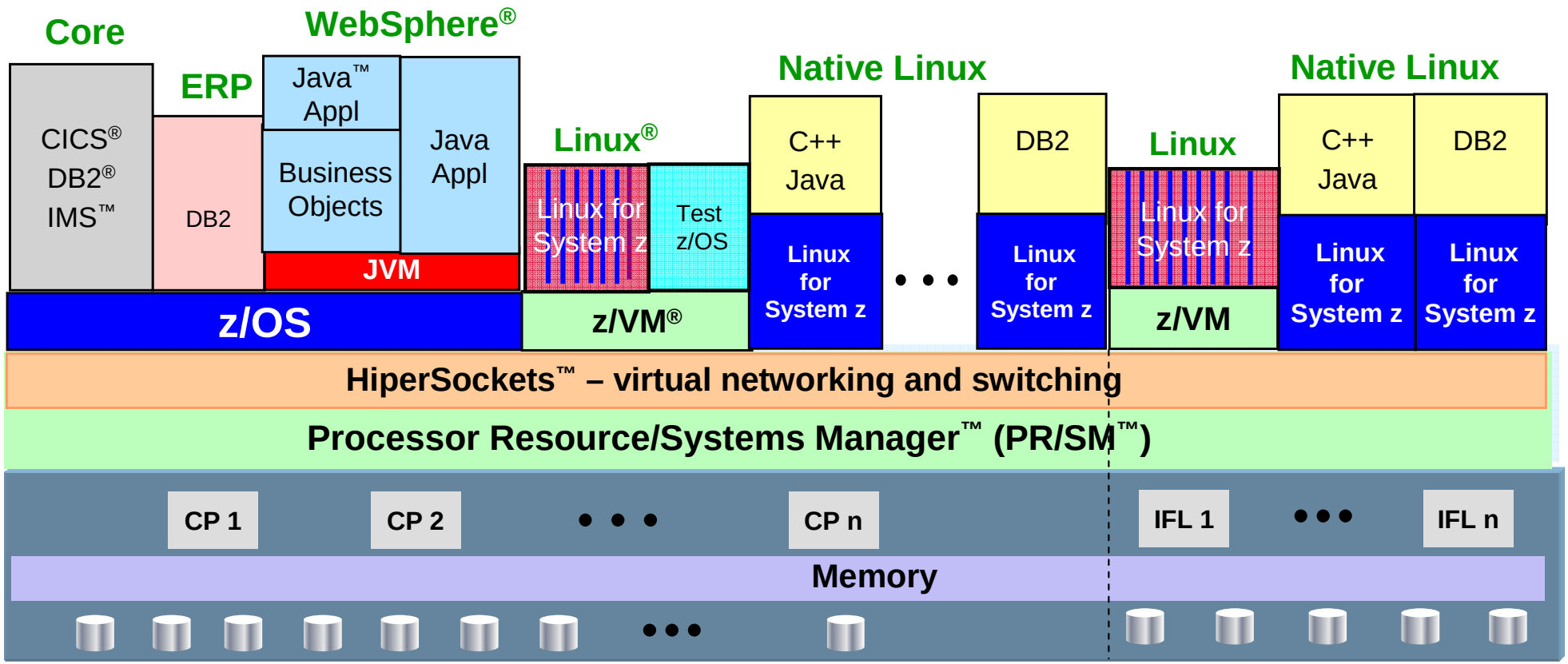
The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Multiple Workloads in the Mainframe – Quality of Service



- **Massive, robust consolidation platform; virtualization is built in, not added on**
- **Up to 60 logical partitions on PR/SM; 100's to 1000's of virtual servers on z/VM**
- **Virtual networking for memory-speed communication, as well as virtual layer 2 and layer 3 networks supported by z/VM**
- **Most sophisticated and complete hypervisor function available**
- **Intelligent and autonomic management of diverse workloads and system resources based on business policies and workload performance objectives**

Multiple Workloads in the Mainframe – Quality of Service

- **Scalability**
  - System structures optimized for data
  - Concurrent capacity optimization
- **High Availability**
  - Parallel Sysplex, ‘Shared everything’
  - GDPS: z/OS, Multiplatform
  - PU sparing, Instruction retry
- **Security**
  - EAL5 (LPAR), EAL3+ (z/OS,…)
  - Crypto hardware
- **Automation**
  - System Automation: z/OS, Multiplatform
  - Intelligent Resource Director, Work Load Manager
- **Virtualization**
  - PR/SM (LPAR)
  - z/VM & Linux
  - HiperSockets

Concurrent (“non-disruptive“) Configuration and Maintenance

- **Important for operation 24 hours a day, 7 days a week (zero downtime)**
  - Concurrent changes to I/O configuration
  - Concurrent upgrade of memory
    - temporary and permanent
  - Concurrent upgrade of processors
    - temporary and permanent
  - Concurrent Book Add
  - Concurrent Book Repair
  - Concurrent installation of firmware fixes and drivers

z196 Overview

Machine Type
- 2817

5 Models
- M15, M32, M49, M66 and M80

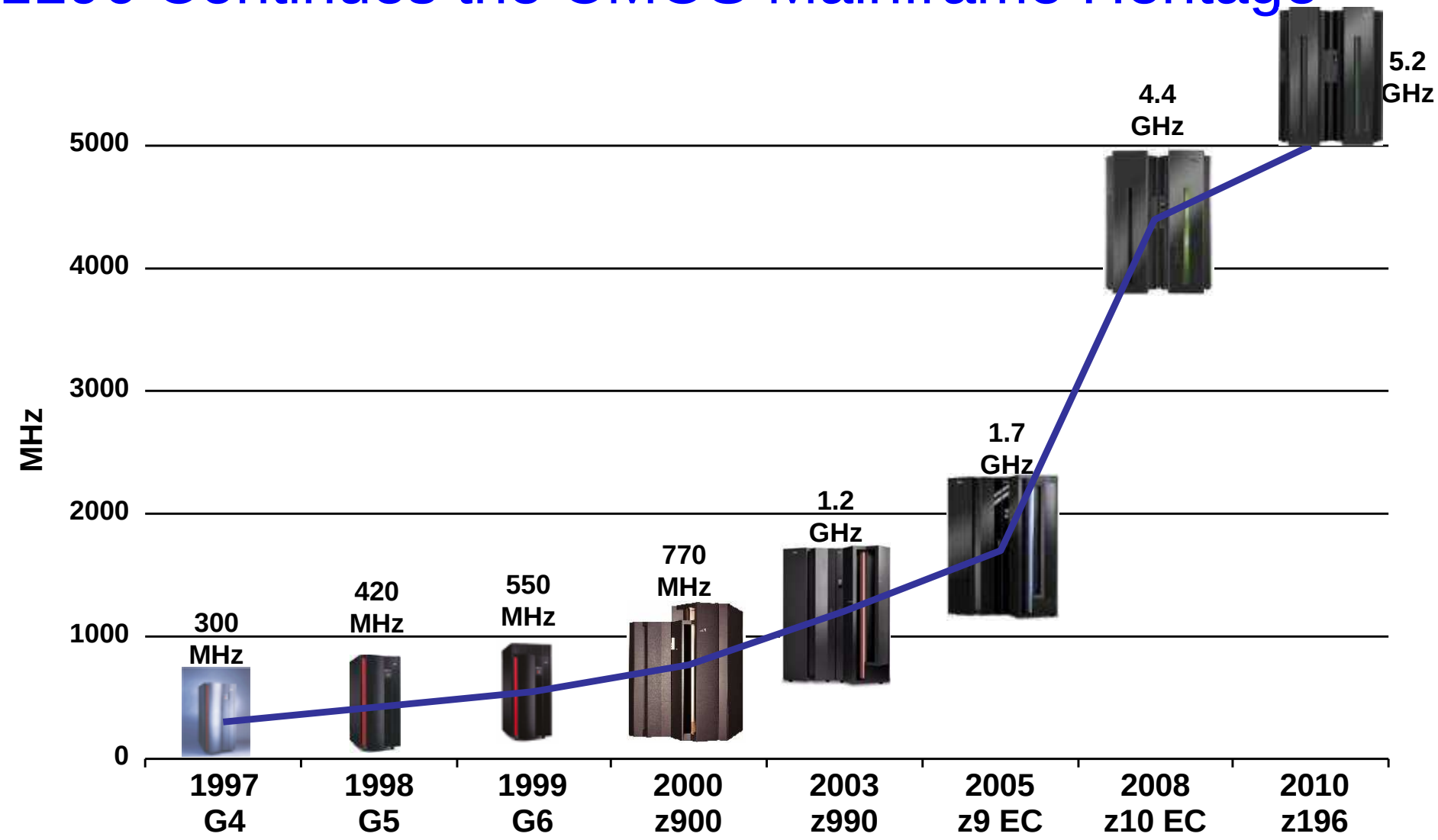
Processor Units (PUs)
- 20 (24 for M80) PU cores per book
- Up to 14 SAPs per system, standard
- 2 spares designated per system
- Dependent on the H/W model - up to 15,32,49,66 or 80 PU cores available for characterization
  - Central Processors (CPs), Integrated Facility for Linux (IFLs), Internal Coupling Facility (ICFs), System z Application Assist Processors (zAAPs), System z Integrated Information Processor (zIIP), optional - additional System Assist Processors (SAPs)
- Sub-capacity available for up to 15 CPs
  - 3 sub-capacity points

Memory
- System Minimum of 32 GB
- Up to 768 GB per book
- Up to 3 TB for System and up to 1 TB per LPAR
  - Fixed HSA, standard
  - 32/64/96/112/128/256 GB increments

I/O
- Up to 48 I/O Interconnects per System @ 6 GBps each
- Up to 4 Logical Channel Subsystems (LCSSs)

STP - optional (No ETR)

z196 Continues the CMOS Mainframe Heritage



- **G4** – 1st full-custom CMOS S/390®
- **G5** – IEEE-standard BFP; branch target prediction
- **G6** – Copper Technology (Cu BEOL)

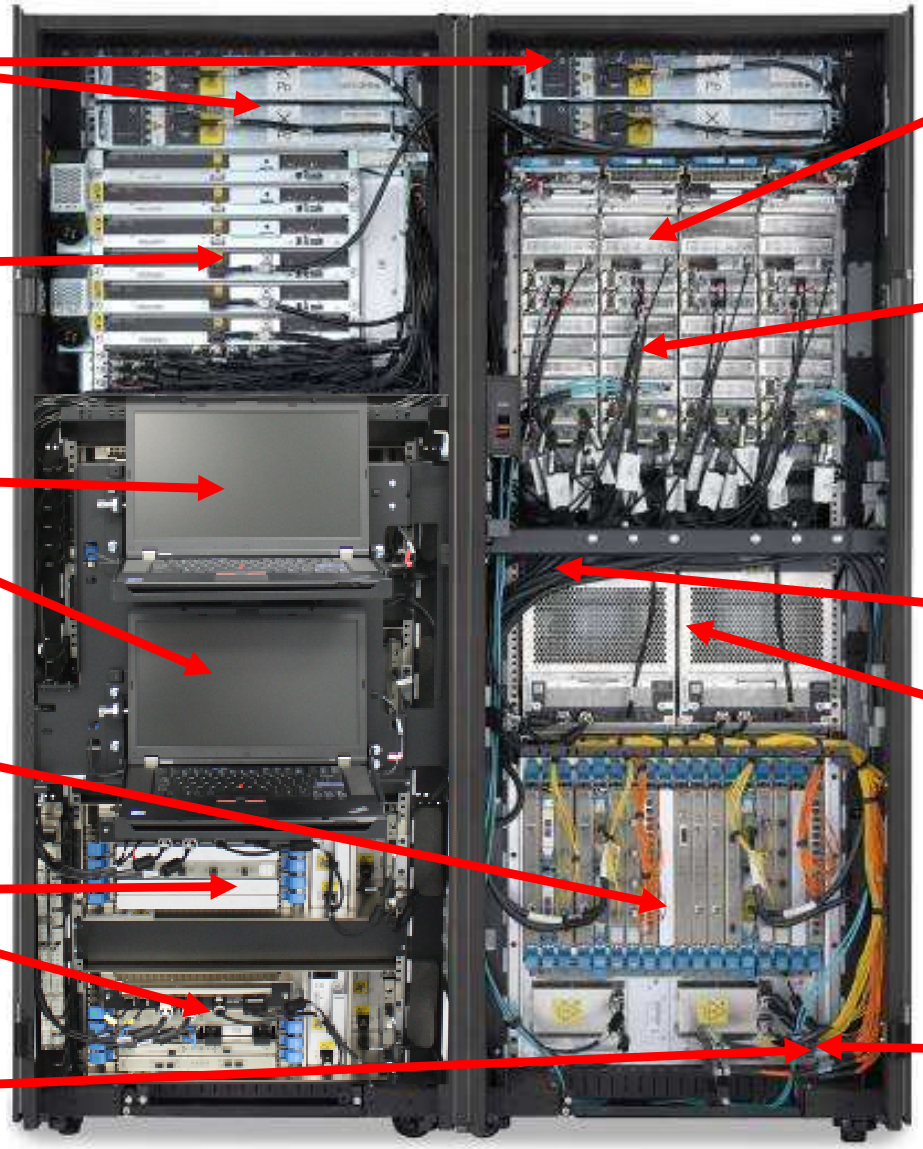
- **z900** – Full 64-bit z/Architecture
- **z990** – Superscalar CISC pipeline
- **z9 EC** – System level scaling

- **z10 EC** – Architectural extensions
- **z196** – Additional Architectural extensions and new cache structure

zBX Overview

- **Machine Type/Model 2458-002**
  - One Model with 5 configurations for IBM Smart Analytics Optimizer
- **Racks – Up to 4 (B, C, D and E)**
  - 42U Enterprise, (36u height reduction option)
  - 4 maximum, 2 chassis/rack
  - 2-4 power line cords/rack
  - Non-acoustic doors as standard
  - Optional Acoustic Doors
  - Optional Rear Door Heat Exchanger (conditioned water required)
- **Chassis – Up to 2 per rack**
  - 9U BladeCenter
  - Redundant Power, cooling and management modules
  - Network Modules
  - I/O Modules
- **Blades (Maximum 112 in 4 racks)**
  - IBM Smart Analytic Optimizer Blades (up to 7 to 56)
    - Can not mix other Blades in the same Chassis
  - Customer supplied POWER7 Blades (up to 112)
  - Customer supplied System x Blades* (up to 112)
  - WebSphere DataPower Appliances* (up to 28)
  - Non-IBM Smart Analytic Optimizer Blades can be mixed in the same chassis
- **Management Firmware**
  - Unified Resource Manager
- **Top of Rack (TOR) Switches - 4**
  - 1000BASE-T intranode management network (INMN)
  - 10 GbE intraensemble data network (IEDN)
- **Network and I/O Modules**
  - 1000BASE-T and 10 GbE modules
  - 8 Gb Fibre Channel (FC) connected to customer supplied disks
  - IBM Smart Analytic Optimizer uses DS5020 disks
    - DS5020s not shared with Customer supplied Blades

z196 – Under the covers (Model M66 or M80)



Internal Batteries (optional)

Power Supplies

2 x Support Elements

I/O cage

I/O drawers

Fiber Quick Connect (FQC) Feature (optional)

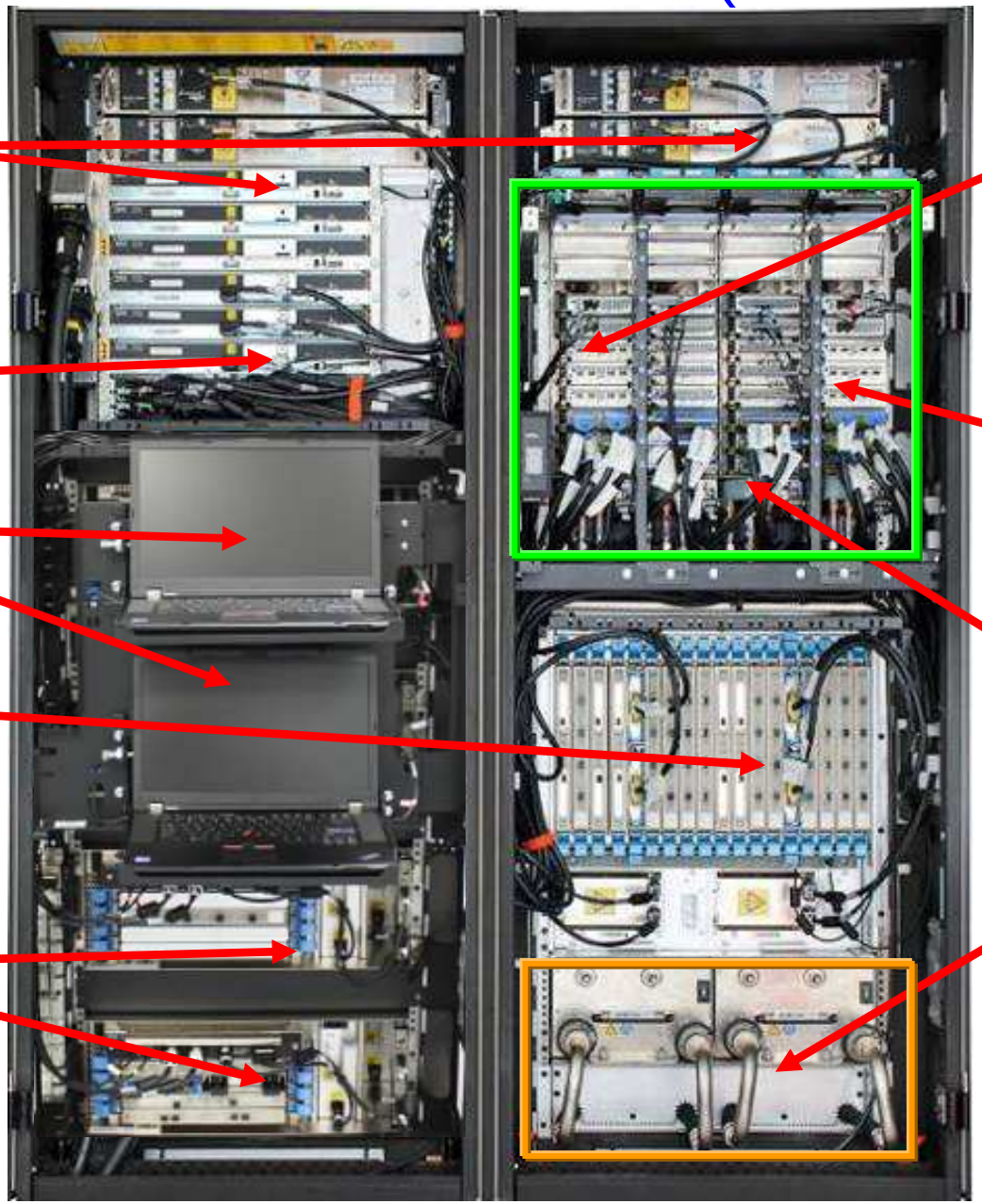
Processor Books, Memory, MBA and HCA cards

Ethernet cables for internal System LAN connecting Flexible Service Processor (FSP) cage controller cards

InfiniBand I/O Interconnects

2 x Cooling Units

z196 Water cooled – Under the covers (Model M66 or M80) front view



Internal Batteries (optional)

Power Supplies

Support Elements

I/O cage

I/O drawers

Ethernet cables for internal System LAN connecting Flexible Service Processor (FSP) cage controller cards

Processor Books, Memory, MBA and HCA cards

InfiniBand I/O Interconnects

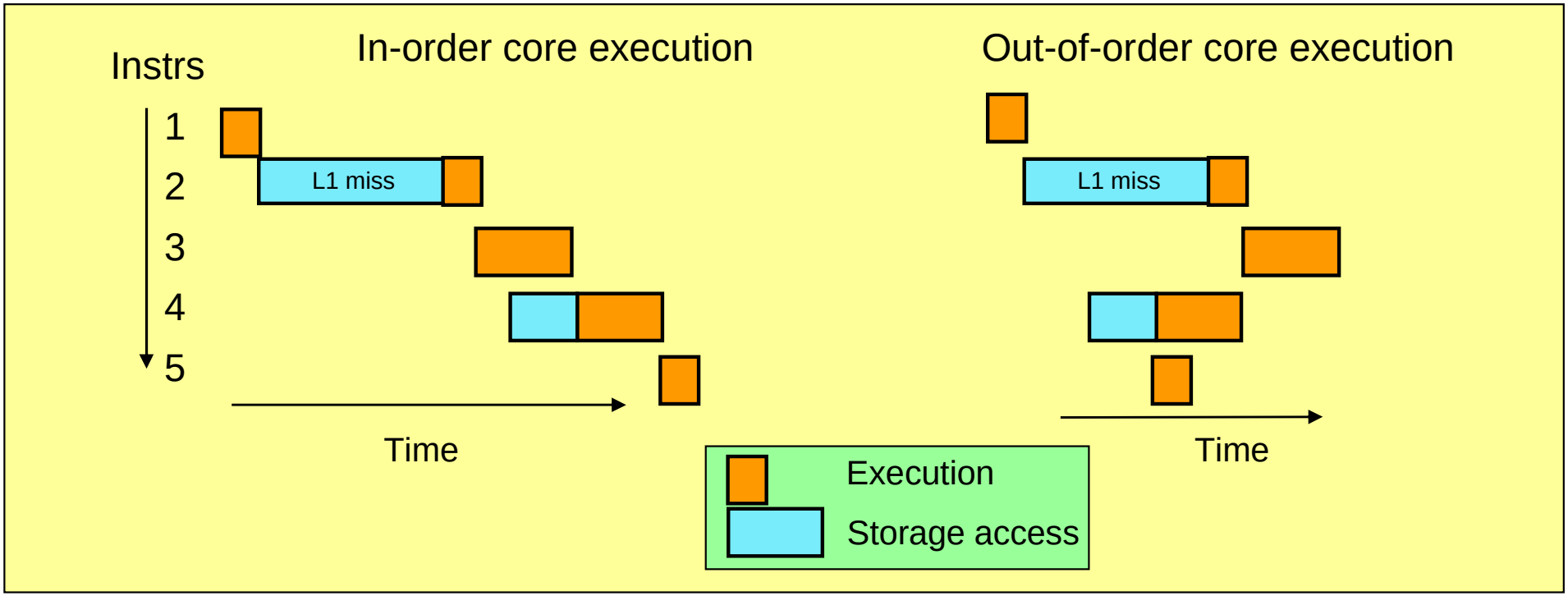
2 x Water Cooling Units

z196 PU core

- **Each core is a superscalar, out of order processor with these characteristics:**
  - Six execution units
    - 2 fixed point (integer), 2 load/store, 1 binary floating point, 1 decimal floating point
  - Up to three instructions decoded per cycle (vs. 2 in z10)
  - 211 complex instructions cracked into multiple internal operations
    - 246 of the most complex z/Architecture instructions are implemented via millicode
  - Up to five instructions/operations executed per cycle (vs. 2 in z10)
  - Execution can occur out of (program) order
    - Memory address generation and memory accesses can occur out of (program) order
    - Special circuitry to make execution and memory accesses appear in order to software
  - Each core has 3 private caches
    - 64KB 1st level cache for instructions, 128KB 1st level cache of data
    - 1.5MB L2 cache containing both instructions and data

z196 Out of Order (OOO) Value

- OOO yields significant performance benefit for compute intensive apps through
  - –Re-ordering instruction execution
    - Later (younger) instructions can execute ahead of an older stalled instruction
  - –Re-ordering storage accesses and parallel storage accesses
- OOO maintains good performance growth for traditional apps

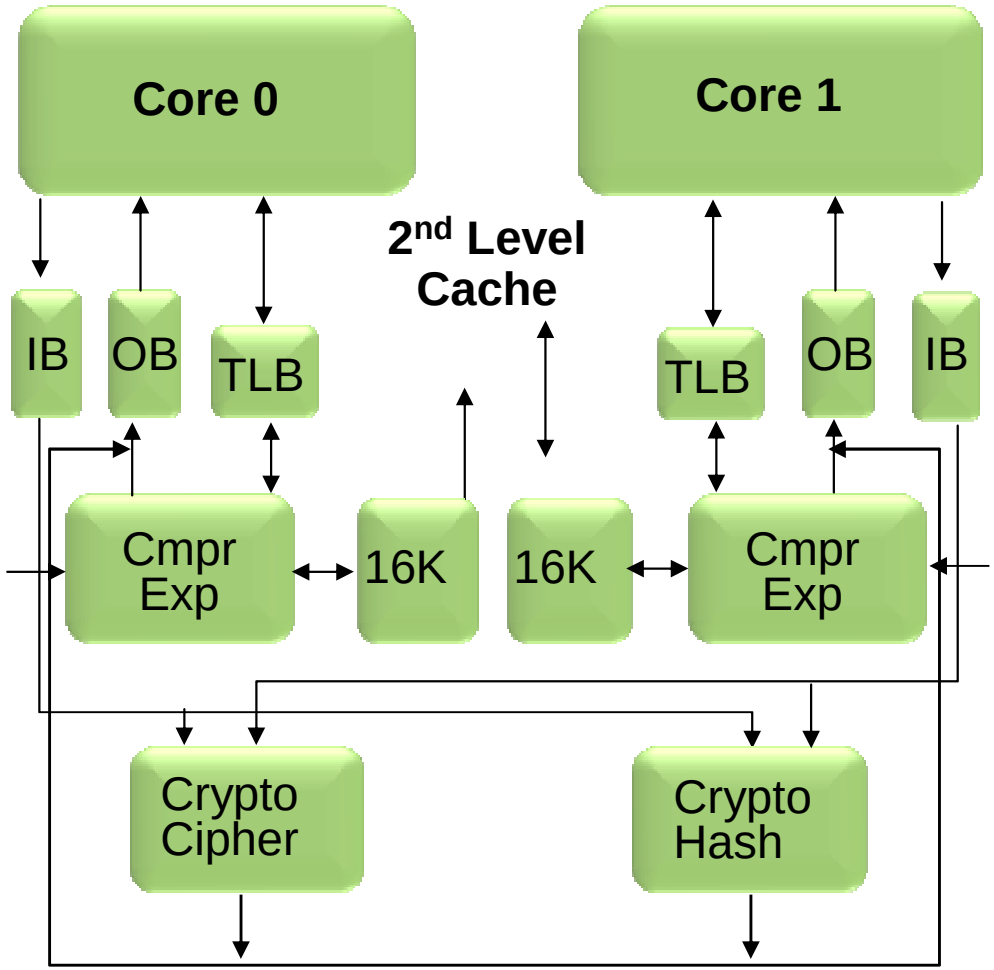


z196 Out of Order Detail

- **Out of order yields significant performance benefit through**
  - Re-ordering instruction execution
    - Instructions stall in a pipeline because they are waiting for results from a previous instruction or the execution resource they require is busy
    - In an in-order core, this stalled instruction stalls all later instructions in the code stream
    - In an out-of-order core, later instructions are allowed to execute ahead of the stalled instruction
  - Re-ordering storage accesses
    - Instructions which access storage can stall because they are waiting on results needed to compute storage address
    - In an in-order core, later instructions are stalled
    - In an out-of-order core, later storage-accessing instructions which can compute their storage address are allowed to execute
  - Hiding storage access latency
    - Many instructions access data from storage
    - Storage accesses can miss the L1 and require 10 to 500 additional cycles to retrieve the storage data
    - In an in-order core, later instructions in the code stream are stalled
    - In an out-of-order core, later instructions which are not dependent on this storage data are allowed to execute

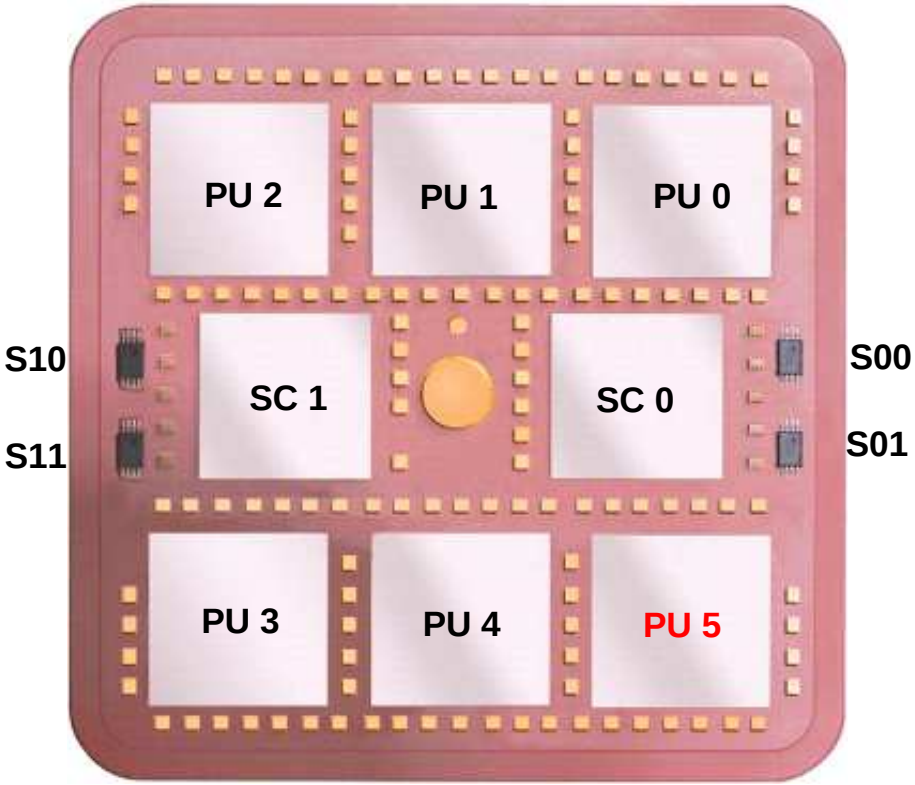
z196 Compression and Cryptography Accelerator

- **Data compression engine**
  - Static dictionary compression and expansion
  - Dictionary size up to 64KB (8K entries)
    - Local 16KB cache per core for dictionary data
- **CP Assist for Cryptographic Function (CPACF)**
  - Enhancements for new NIST standard
  - Complemented prior ECB and CBC symmetric cipher modes with XTS, OFB, CTR, CFB, CMAC and CCM
  - New primitives (128b Galois Field multiply) for GCM
- **Accelerator unit shared by 2 cores**
  - Independent compression engines
  - Shared cryptography engines



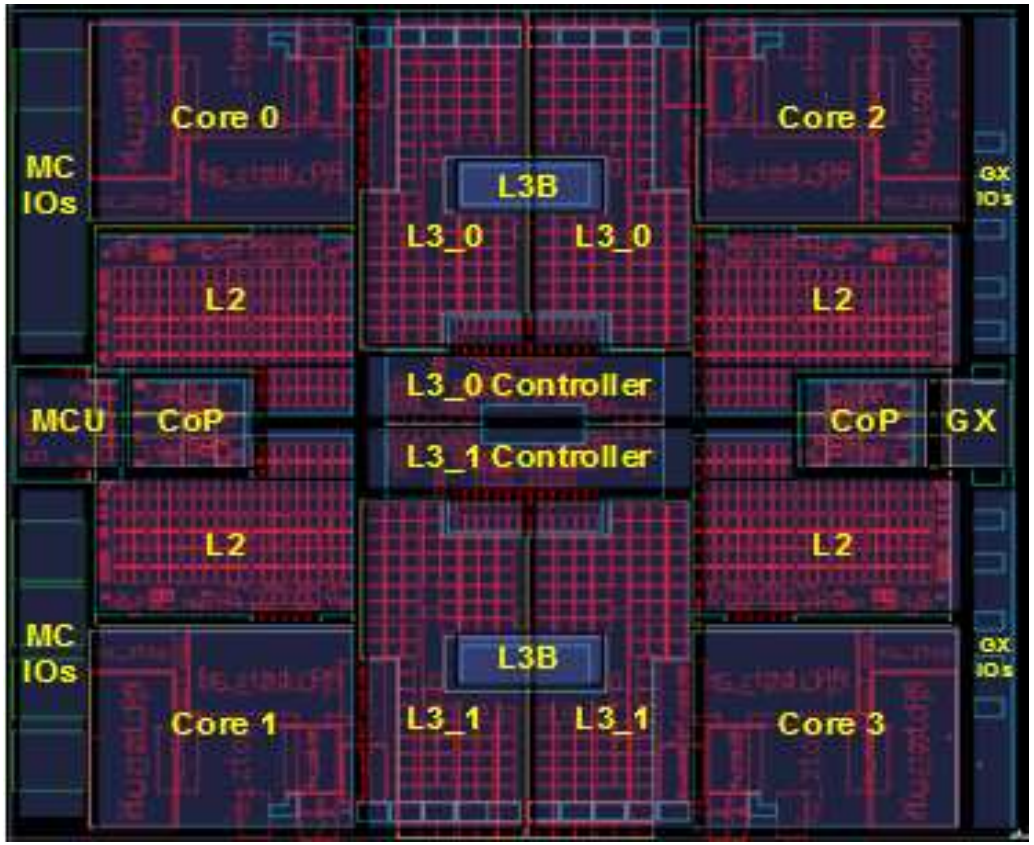
z196 Multi-Chip Module (MCM) Packaging

- **96mm x 96mm MCM**
  - 103 Glass Ceramic layers
  - 8 chip sites
  - 7356 LGA connections
  - 20 and 24 way MCMs
  - Maximum power used by MCM is 1800W



- **CMOS 12s chip Technology**
  - PU, SC, S chips, 45 nm
  - 6 PU chips/MCM – Each up to 4 cores
    - One memory control (MC) per PU chip
    - 23.498 mm x 21.797 mm
    - 1.4 billion transistors/PU chip
    - L1 cache/PU core
      - → 64 KB I-cache
      - → 128 KB D-cache
    - L2 cache/PU core
      - → 1.5 MB
    - L3 cache shared by 4 PUs per chip
      - → 24 MB
    - 5.2 GHz
  - 2 Storage Control (SC) chip
    - 24.427 mm x 19.604 mm
    - 1.5 billion transistors/SC chip
    - L4 Cache 96 MB per SC chip (192 MB/Book)
    - L4 access to/from other MCMs
  - 4 SEEPROM (S) chips
    - 2 x active and 2 x redundant
    - Product data for MCM, chips and other engineering information
  - Clock Functions – distributed across PU and SC chips
    - Master Time-of-Day (TOD) function is on the SC

z196 Quad Core PU Chip Detail

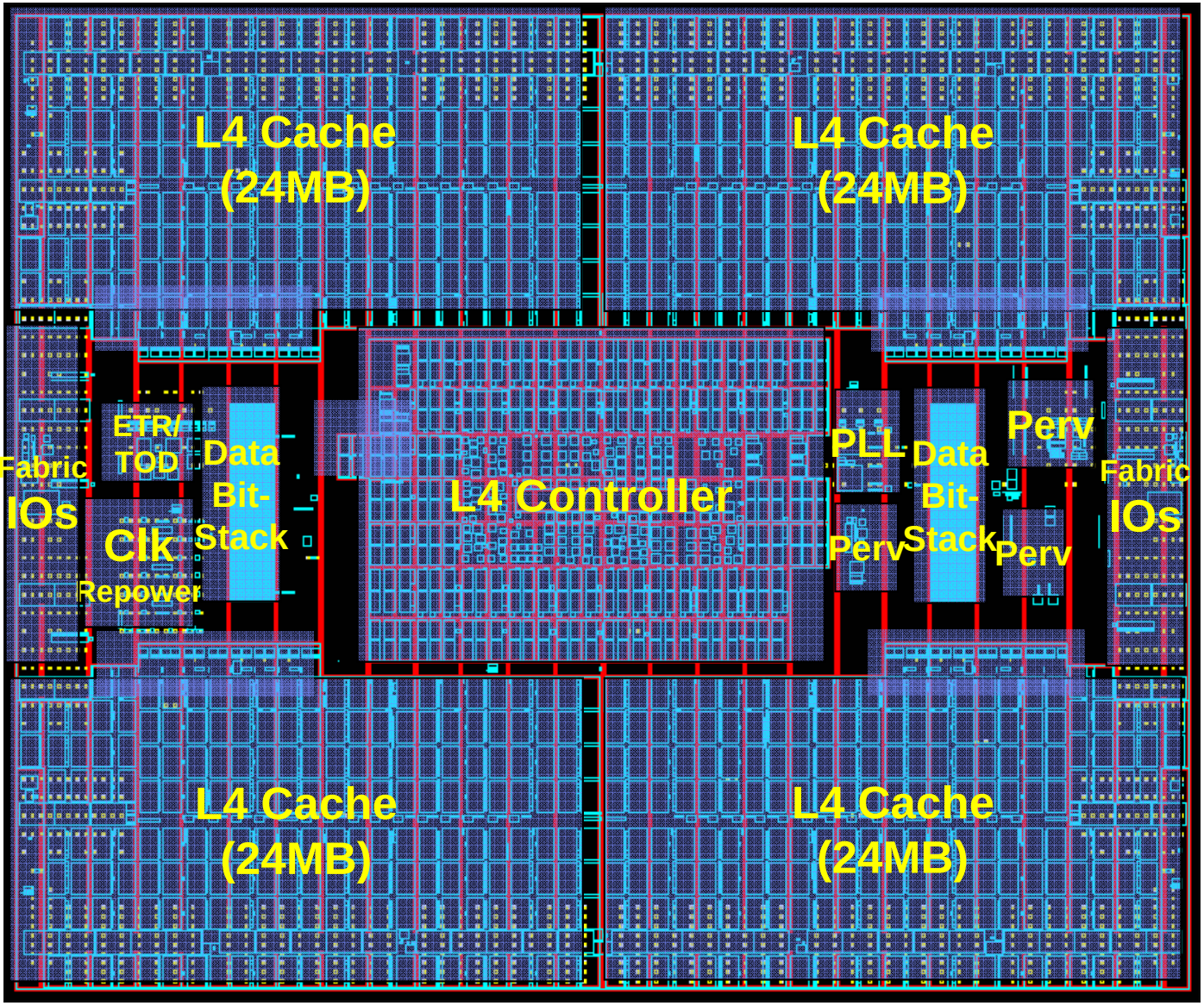


- **12S0 45nm SOI Technology**
  - 13 layers of metal
  - 3.5 km wire
- **1.4 Billion Transistors**
- **Chip Area – 512.3mm²**
  - 23.5mm x 21.8mm
  - 8093 Power C4's
  - 1134 signal C4's

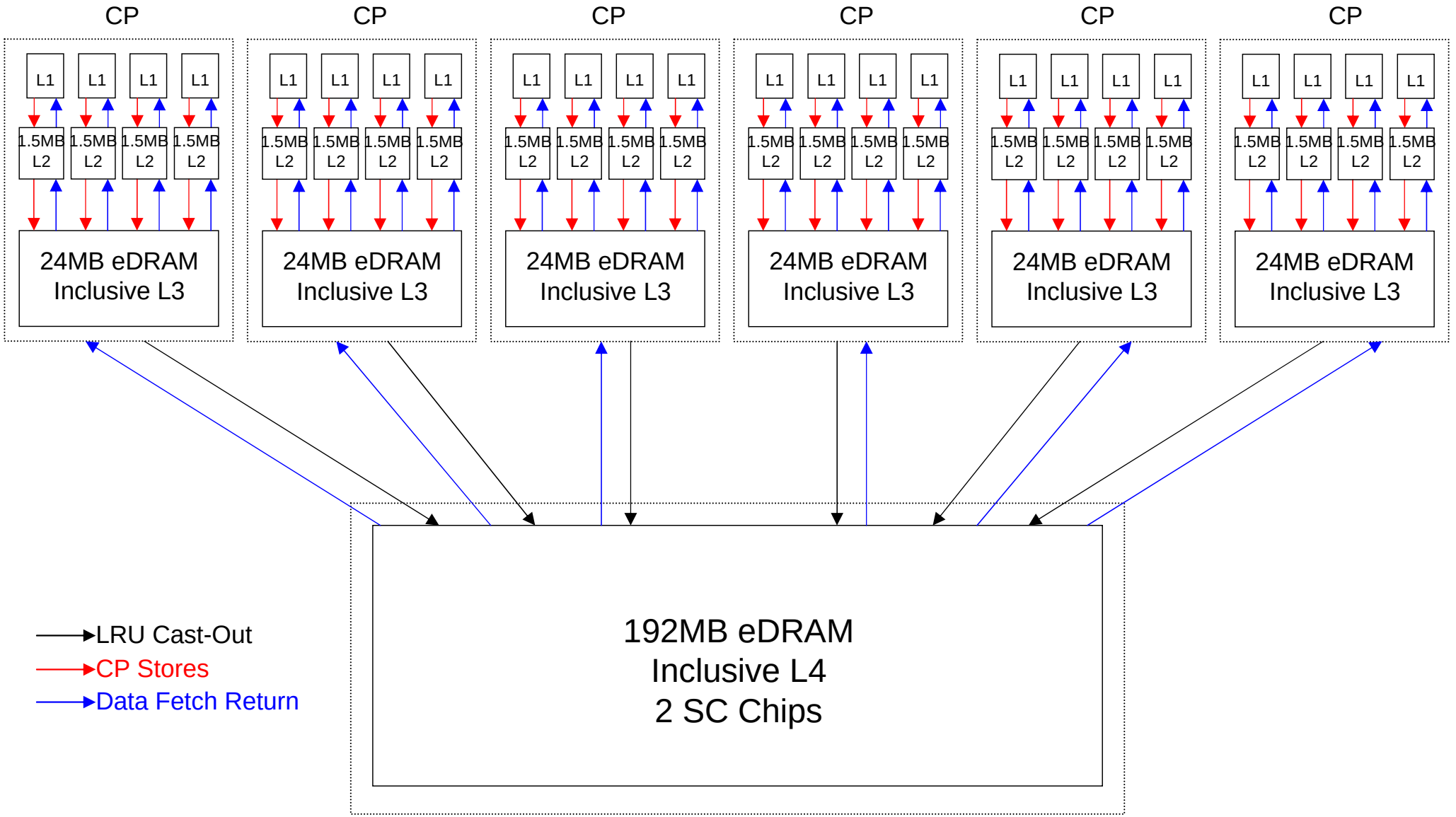
- **Up to four active cores per chip**
  - 5.2 GHz
  - L1 cache/ core
    - 64 KB I-cache
    - 128 KB D-cache
  - 1.5 MB private L2 cache/ core
- **Two Co-processors (COP)**
  - **Crypto & compression accelerators**
  - Includes 16KB cache
  - Shared by two cores
- **24MB eDRAM L3 Cache**
  - Shared by all four cores
- **Interface to SC chip / L4 cache**
  - 41.6 GB/sec to each of 2 SCs
- **I/O Bus Controller (GX)**
  - Interface to Host Channel Adapter (HCA)
- **Memory Controller (MC)**
  - Interface to controller on memory DIMMs
  - Supports RAIM design

z196 SC Chip Detail

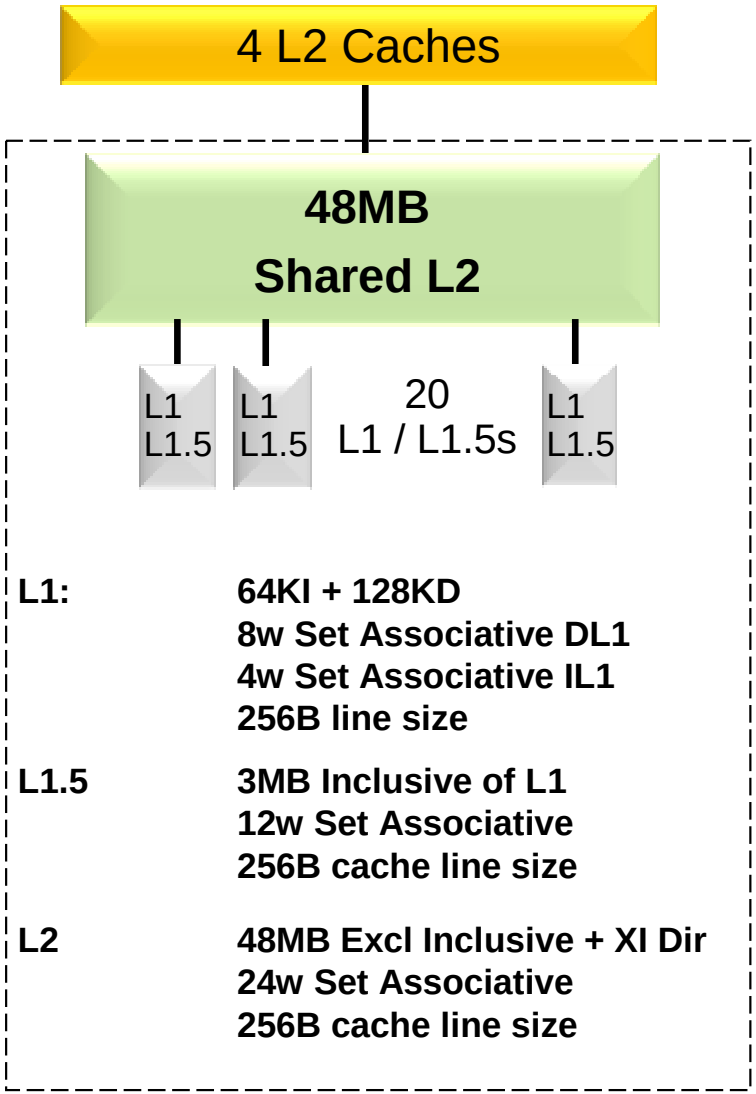
- 12S0 45nm SOI Technology
  - 13 layers of metal
- Chip Area – 478.8mm^2
  - 24.4mm x 19.6mm
  - 7100 Power C4's
  - 1819 signal C4's
- 1.5 Billion Transistors
  - 1 Billion cells for eDRAM
- eDRAM Shared L4 Cache
  - 96 MB per SC chip
  - 192 MB per Book
- 6 CP chip interfaces
- 3 Fabric interfaces
- 2 clock domains
- 5 unique chip voltage supplies



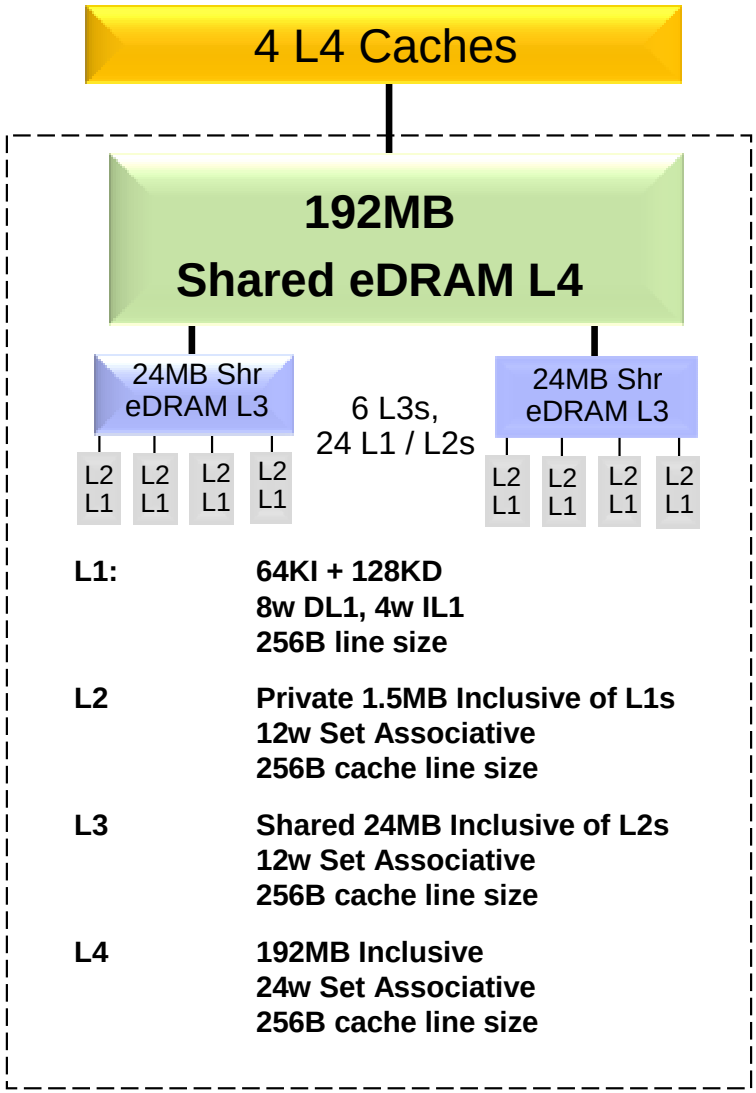
z196 Book Level Cache Hierarchy



System z Cache Topology – z10 EC vs. z196 Comparison

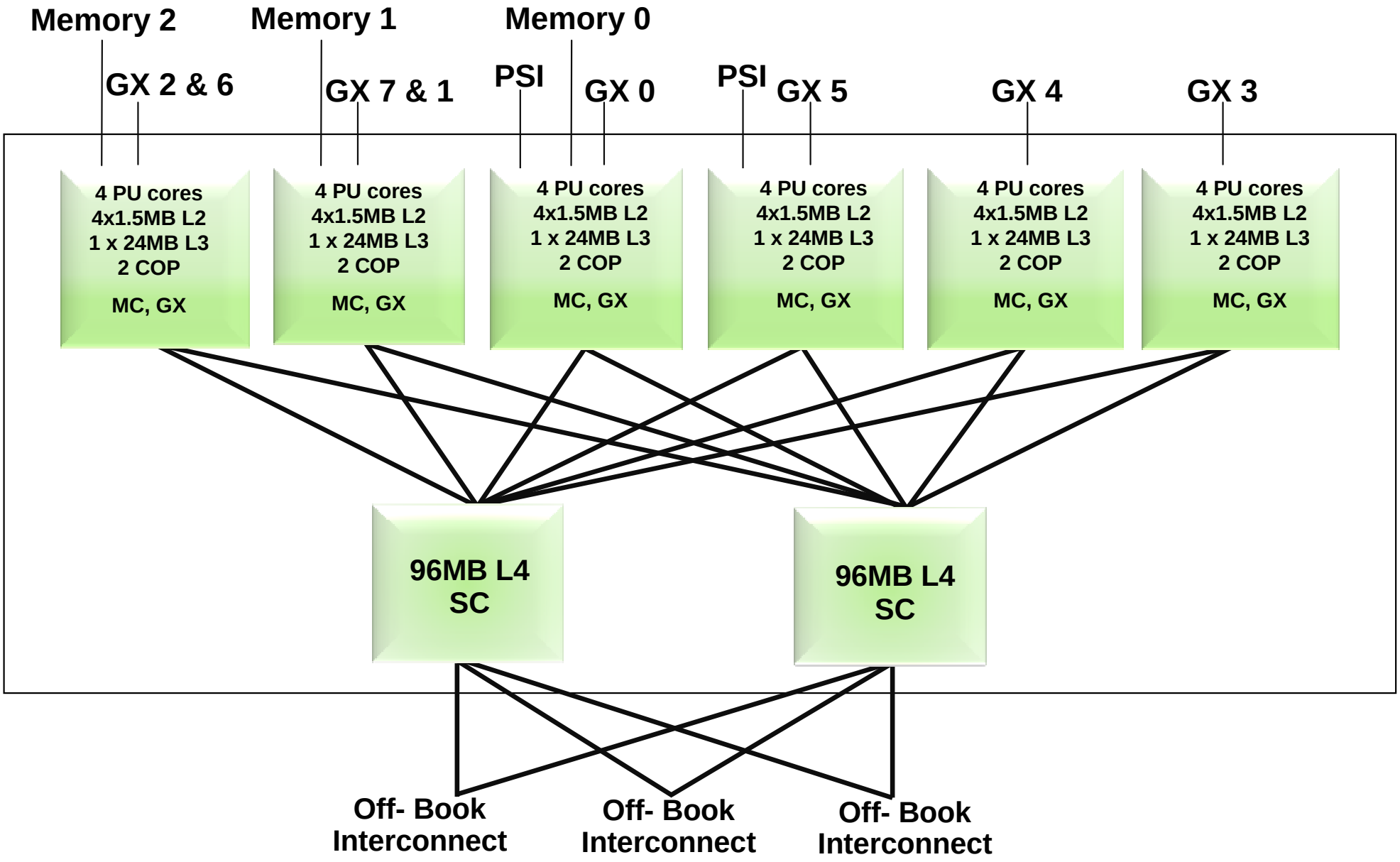


z10 EC



z196

z196 24 PU MCM Structure



z10 EC MCM vs. z196 MCM Comparison

z10 EC MCM

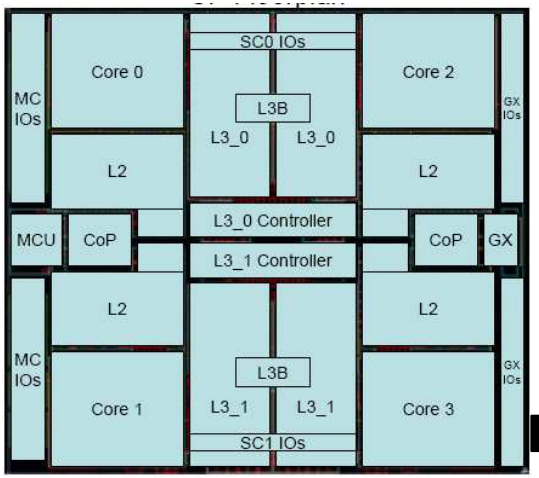
- MCM
  - 96mm x 96mm in size
  - 5 PU chips per MCM
    - Quad core chips with 3 or 4 active cores
    - PU Chip size 21.97 mm x 21.17 mm
    - 4.4 GHz
    - Superscalar, In order execution
    - L1: 64K I /128K D private/core
    - L1.5: 3M I+D private/core
  - 2 SC chips per MCM
    - L2: 2 x 24 M = 48 M L2 per book
    - SC Chip size 21.11 mm x 21.71 mm
  - Power 1800 Watts

z196 MCM

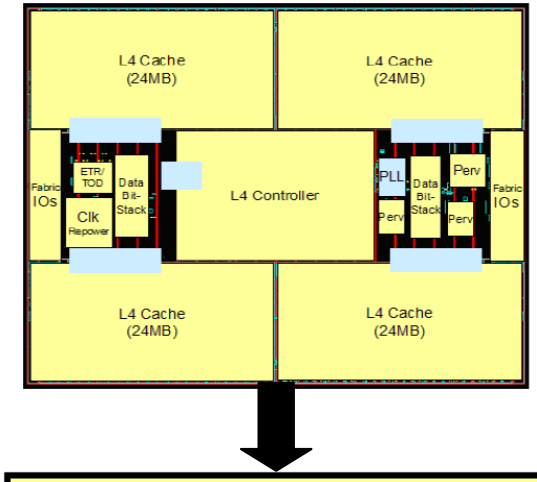
- **MCM**
  - 96mm x 96mm in size
  - 6 PU chips per MCM
    - Quad core chips with 3 or 4 active cores
    - PU Chip size 23.5 mm x 21.8 mm
    - 5.2 GHz
    - Superscalar, OOO execution
    - L1: 64K I / 128K D private/core
    - L2: 1.5M I+D private/core
    - L3: 24MB/chip - shared
  - 2 SC chips per MCM
    - L4: 2 x 96 MB = 192 MB L4 per book
    - SC Chip size 24.4 mm x 219.6 mm
  - Power 1800 Watts

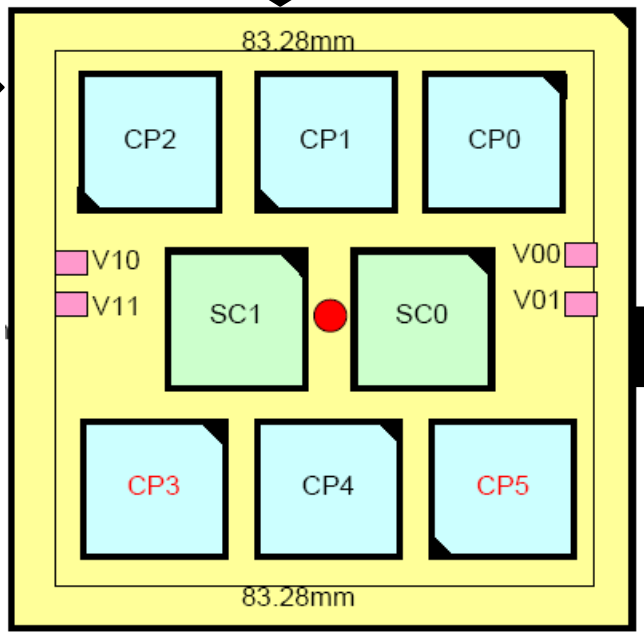
z196 PU chip, SC chip and MCM

z196 Quad Core PU CHIP



96 MB SC CHIP

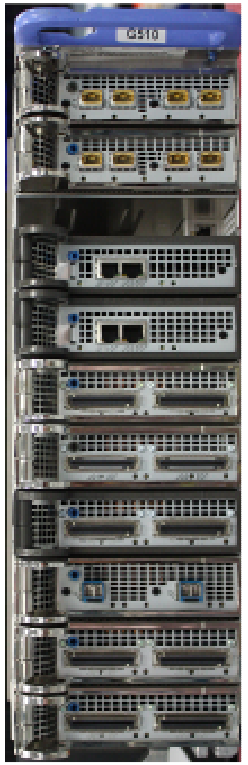




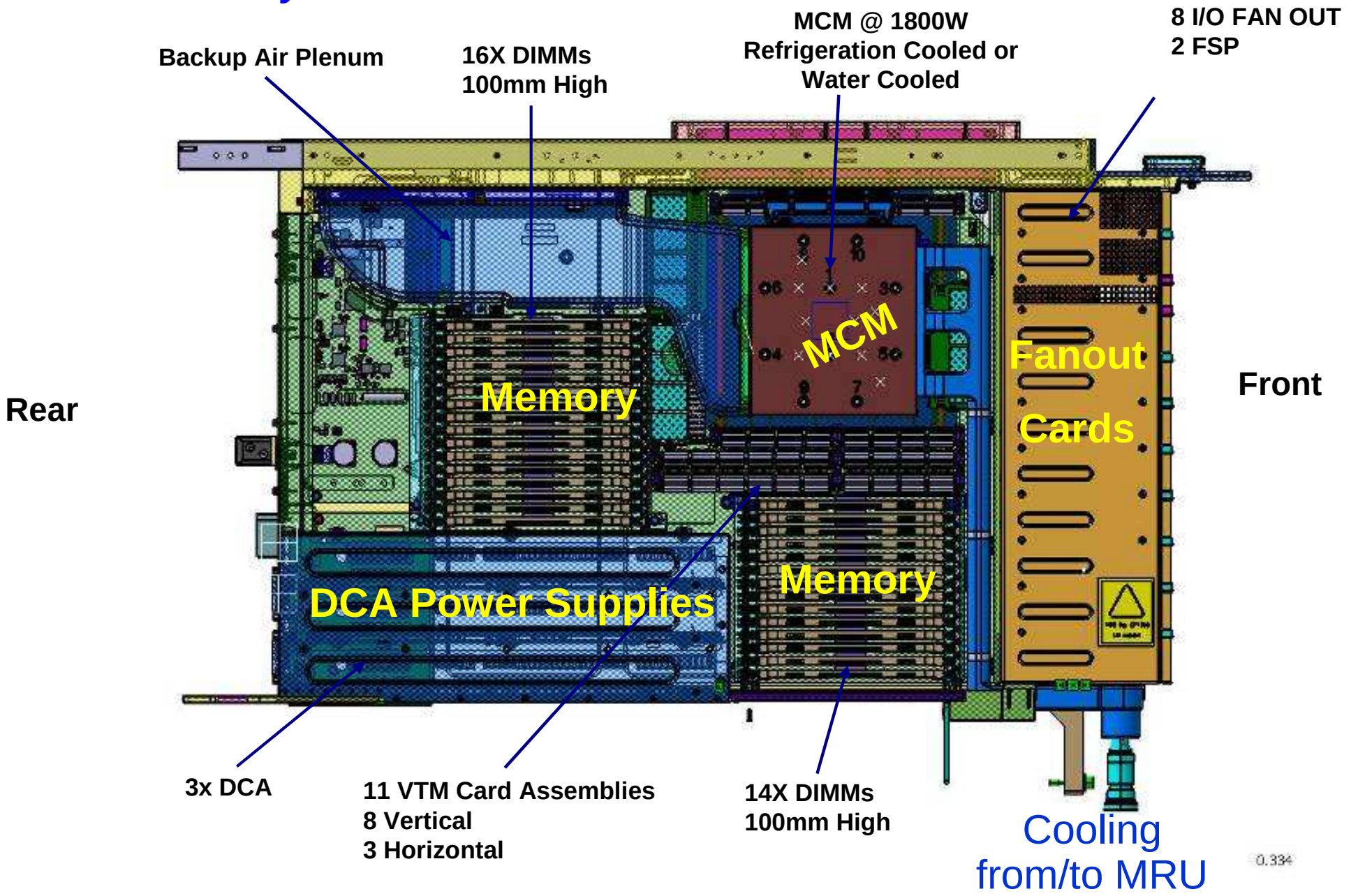
MCM

Front View

Front View Fanouts



z196 Book Layout

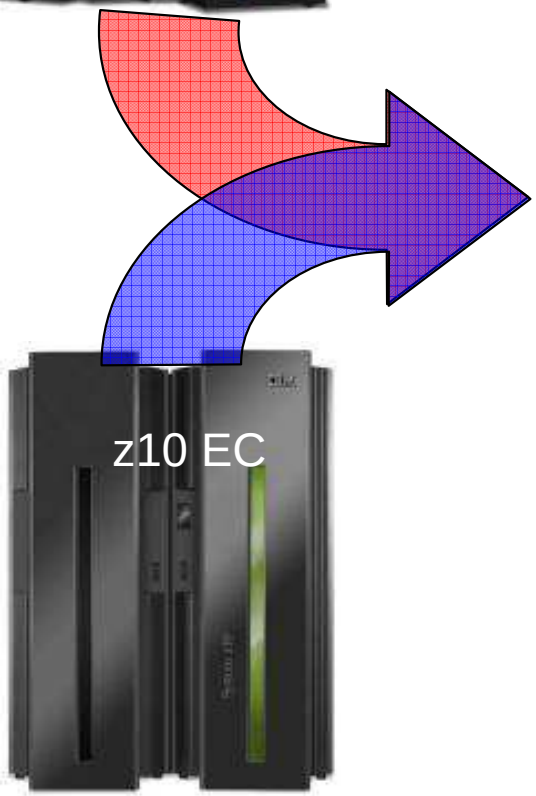


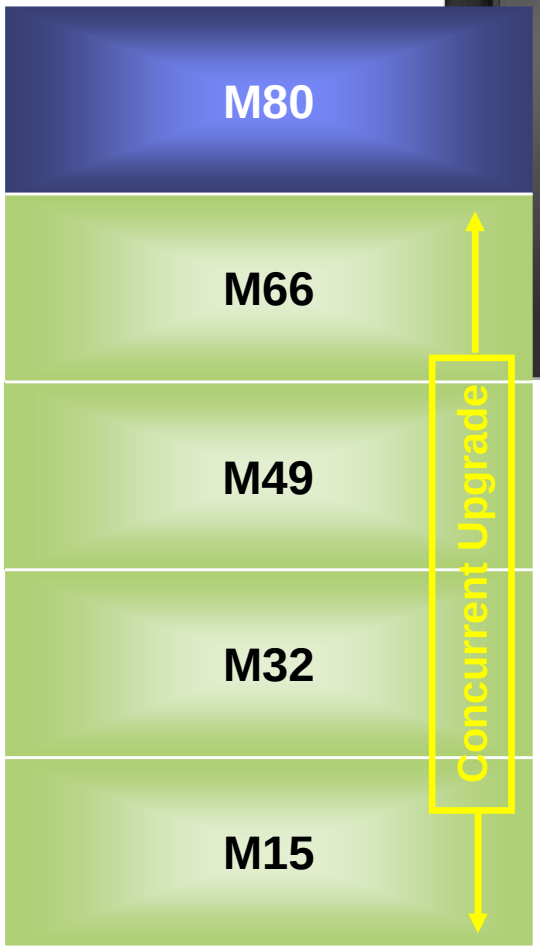
Operating System Support for z196

- **Currency is key to operating system support and exploitation of future servers**
- **The following are the minimum operating systems planned to run on z196:**
  - **z/OS**
    - CPC: z/OS V1.9 [1] for toleration only; exploitation starts with z/OS V1.10 with full exploitation with z/OS V1.12
    - zBX: z/OS V1.10
  - **Linux on System z distributions:**
    - Novell SUSE SLES 10 and SLES 11
    - Red Hat RHEL 5
  - **z/VM**
    - CPC: z/VM V5.4 or higher
    - zBX support: z/VM V6.1
  - **z/VSE V4.1 or higher**
  - **z/TPF V1.1 or higher**
- **Using the general purpose application server blades we have:**
  - **AIX 5.3, 6.1**
  - **Linux on System x[2] (SOD)**

1. z/OS V1.9 support ends on Sept. 30, 2010. Lifecycle Extension for z/OS 1.9 is available Oct. 1, 2010. Note that z/OS 1.8 with the Lifecycle Extension for z/OS 1.8 and z/OS 1.7 with the Lifecycle Extension for z/OS 1.7 are also available with toleration support only.
2. All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represents goals and objectives only.

z196 System Upgrades





- **z196 to higher hardware z196 model**
  - Upgrade of z196 Models M15, M32, M49 and M66 to M80 is disruptive
  - When upgrading to z196 all the Books are replaced
  - Upgrade from Air to Water cooled not available
- **Any z9 EC to any z196**
- **Any z10 EC to any z196**

System z PU Characterization

- **The type of Processor Units (PUs) that can be ordered on System z:**
  - **Central Processors (CPs)**
    - Provides processing capacity for z/Architecture™ and ESA/390 instruction sets
    - Runs z/OS, z/VM, VSE/ESA, z/VSE, TPF/ESA, z/TPF, Linux for System z and Linux under z/VM or Coupling Facility
  - **System z Application Assist Processors (zAAPs)**
    - Under z/OS, zAAPs are used for Java processing by the Java™ Virtual Machine (JVM) as well as XML processing
  - **System z Integrated Information Processors (zIIPs)**
    - First exploited by DB2 Version 8 for z/OS (requires z/OS V1.7)
  - **Integrated Facility for Linux (IFL)**
    - Provides additional processing capacity for Linux workloads
  - **Internal Coupling Facility (ICF)**
    - Provides additional processing capacity for the execution of the Coupling Facility Control Code (CFCC) in a CF LPAR
  - **System Assist Processors (SAPs)**
    - SAPs manage the start and ending of I/O operations for all Logical Partitions and all attached I/O

Modes of Operation

Logical Partitioning

- **A System z machine is always logically partitioned**
  - Up to 60 logical partitions (LPARs) can be defined, each of them hosting an operating system
- **The partitioning is transparent to the software running in an LPAR**
- **An operating system running in an LPAR controls only the resources assigned to it (CPUs, storage, I/O)**

Architecture Modes: ESA/390 vs. z/Architecture

- **System z supports two architecture modes: ESA/390 and z/Architecture**
- **In ESA/390 architecture, the system can run old (31-bit) operating systems and applications**
  - This is like running on an S/390 system such as 9672 G6
- **A system may run any mixture of ESA/390 LPARs and z/Architecture LPARs**
- **"Principles of Operation" describe the function and features of System z at the instruction level**

  http://publibfi.boulder.ibm.com/epubs/pdf/dz9zr008.pdf

z/Architecture Key Characteristics

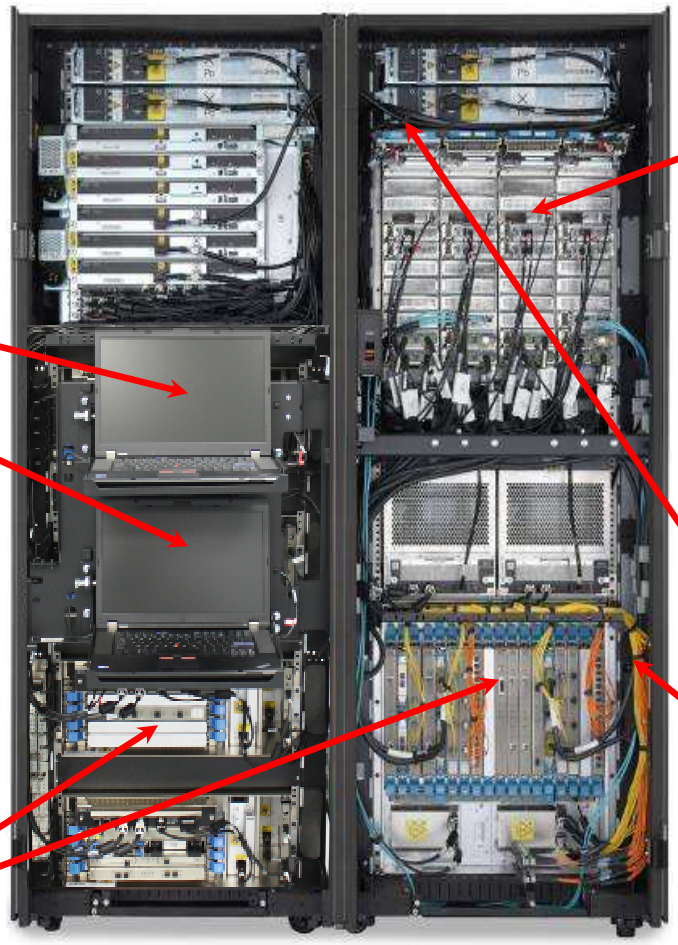
- **64-bit architecture, uses 64-bit storage addresses and 64-bit integer arithmetic and logical instructions**
- **Superset of former ESA/390 architecture that provided 31-bit storage addresses and 32-bit integer arithmetic instructions**
- **Incompatibilities between z/Architecture and ESA/390 usually affect only operating systems, *not* applications**
  - Different formats of the program status word (PSW)
  - Address translation process for virtual storage
  - Layout of the assigned storage locations

Firmware Layers

System z Firmware Components

Support Element
Startup, system control, service
C, C++, Java, Perl

Hardware Management Console
Controls multiple z Systems
C, C++

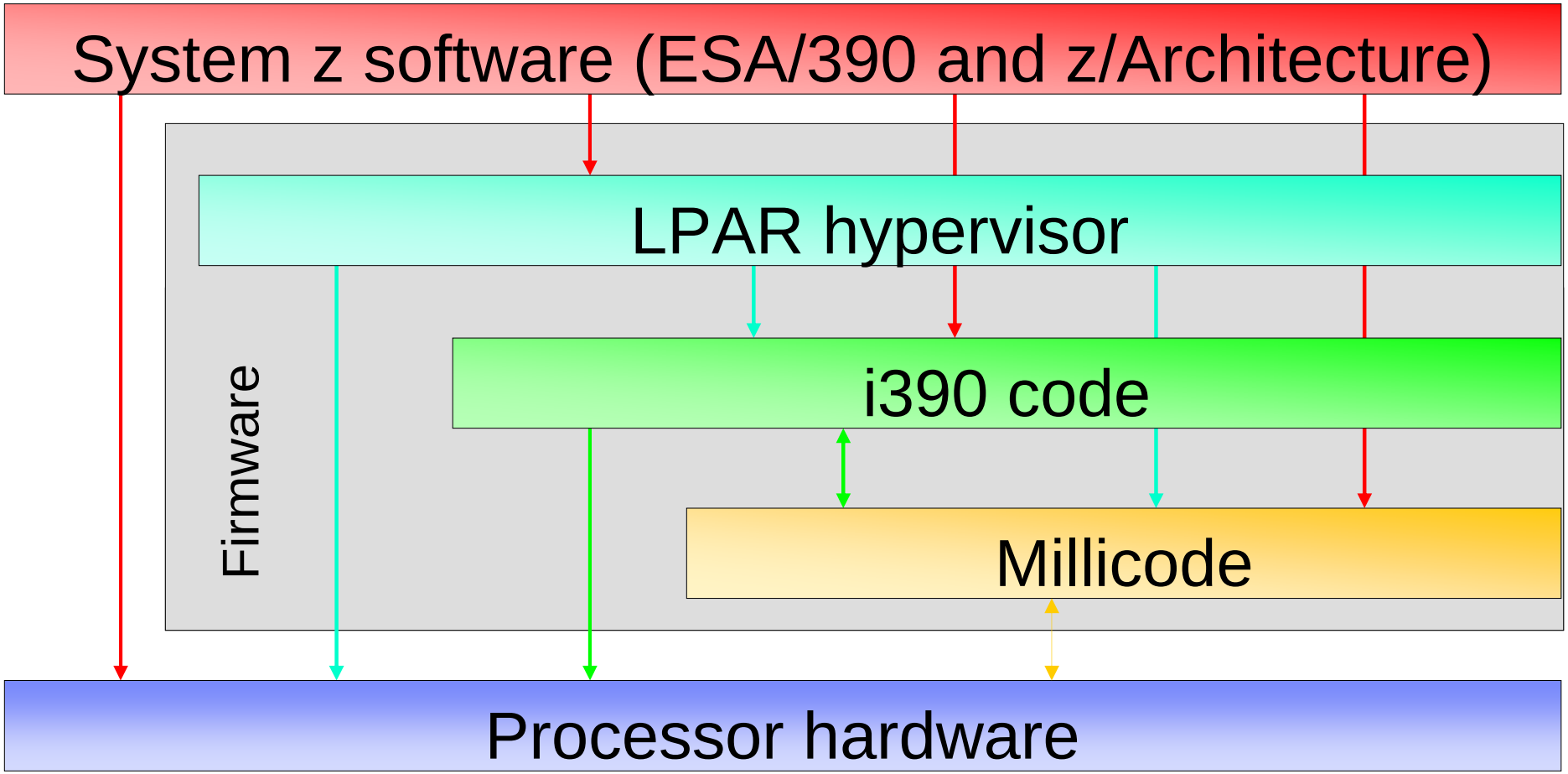


I/O Channels
C, Assembler, horizontal microcode

Central Electronic Complex (CEC)
i390 (C, C++, PL.8), millicode (Assembler), LPAR (PL/X, Assembler):
Complex instructions, I/O subsystem, virtualization, recovery

PSCN / Flexible Support Processor (FSP)
Power control
SE connection
C, C++

System z Software/Firmware Layers



What is Millicode?

- **Low level firmware running on all processors (PUs)**
- **Processors implement only a subset of z/Architecture in hardware (about 75% of the instruction set of ~ 1,000 instructions)**
- **Functions implemented in millicode:**
  - Execution of complex instructions
  - Interrupt handling (program, external, I/O, machine check)
  - Virtualization: interpretive execution
  - i390 specific facilities
  - Special RAS (reliability, availability, serviceability) and debug functions
  - Reset functions
- **Implemented in System z Assembler**

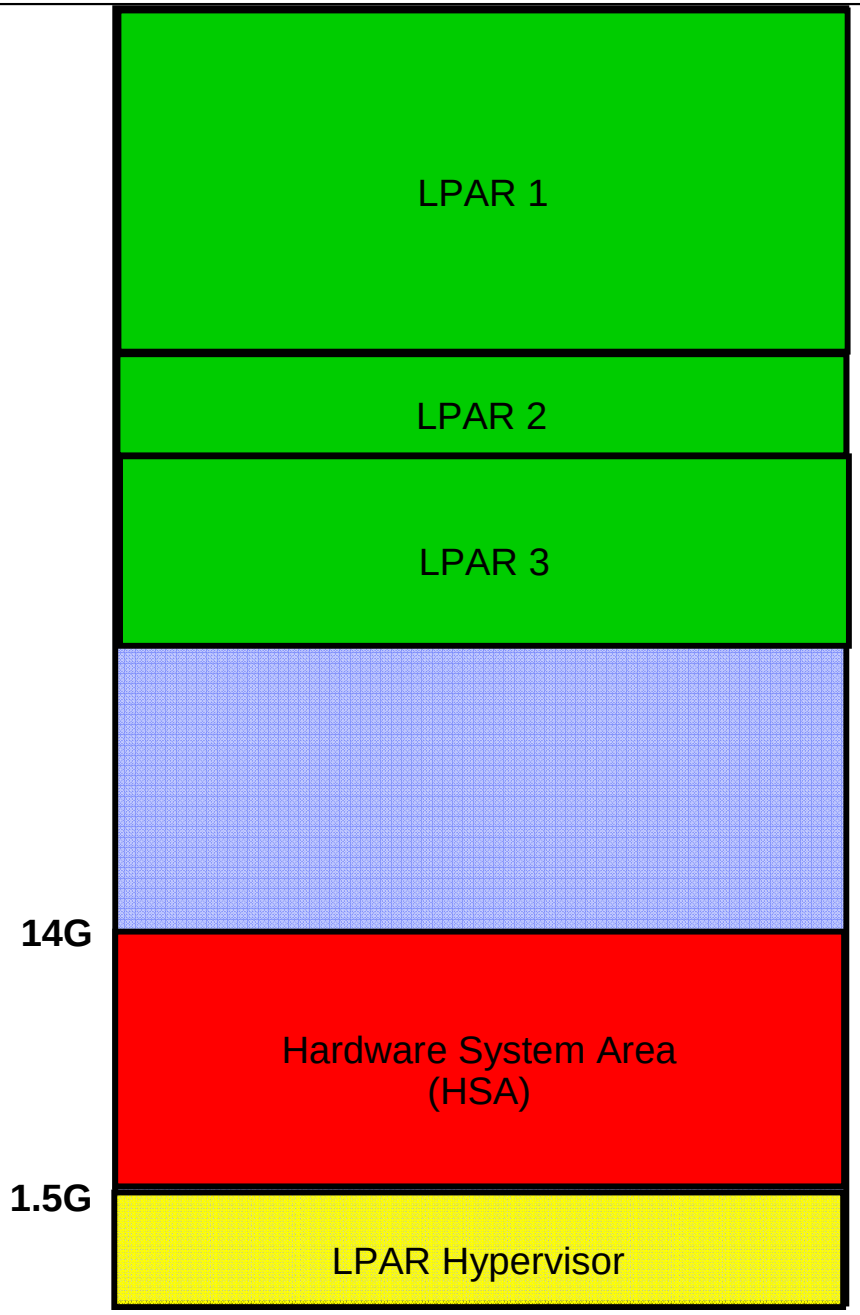
What is i390 Code?

- **High-level firmware running in “internal 390” mode**
- **Performs complex functions such as**
  - System initialization and reset
  - I/O subsystem
  - Concurrent maintenance
  - Communication with the Support Element (SE)
  - A few instructions
- **Durable architecture for System z host firmware**
- **Dates back to 1985, implemented and exploited on all S/390 CMOS systems and System z generations**
- **Consists of System z instructions**
- **“Control spaces” and control space operations (instructions) define i390-architected access to hardware facilities via millicode**
- **Provides a platform for host firmware development with high-level language compilers (C, C++, PL.8) and other tools**

Where does Firmware reside?

- **System z196 has up to 3T storage**
  - Up to 768G per book
- **Hardware System Area (HSA) is reserved for firmware**
- **Fixed size for System z196: 16G**
  - „True“ HSA (12.5G)
  - LPAR hypervisor (1.5G)
  - Storage keys (2G)

System z Storage Layout



Register Sets

Register Sets

- **16 general registers (0 – 15)**
  - Used for address generation/calculation as well as for integer arithmetic (signed and unsigned)
  - Each register has 64 bits, numbered from 0 to 63
  - In ESA/390 architecture, only the low-order 32 bits are accessible (bits 32-63)
- **16 floating-point registers (0 – 15)**
  - Used for binary, decimal, and hexadecimal floating-point operations
  - Each register has 64 bits
  - Extended precision floating-point arithmetic uses register pairs (e. g., 0/2, 1/3, 4/6, 5/7, etc.)
- **Floating-point-control register**
  - Contains IEEE exception masks and flags, defines rounding mode
  - This is a 32-bit register

Register Sets (continued…)

- **16 access registers (0 – 15)**
  - Used to access address spaces
  - Each register has 32 bits
- **Prefix register**
  - Used to define the absolute addresses of the assigned storage locations for a CPU
  - This is a 32-bit register
  - This register is directly accessed only by the operating system

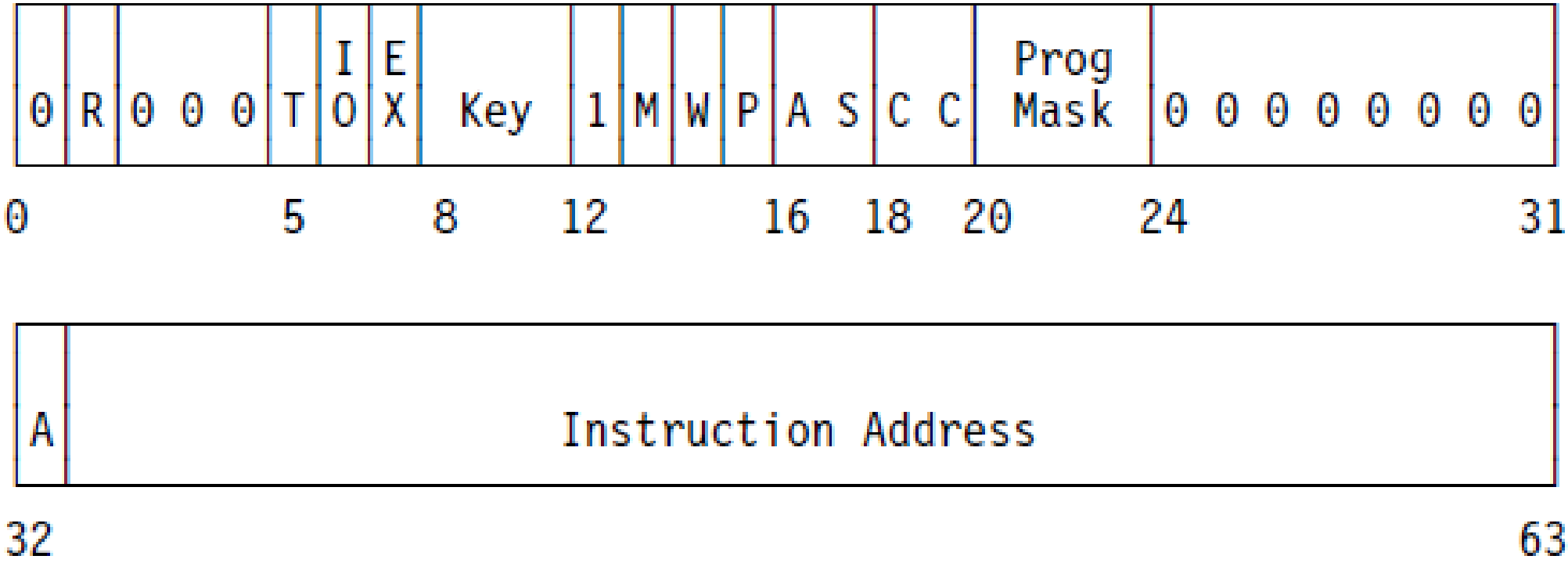
Register Sets (continued…)

- **16 control registers (0 – 15)**
  - Used by the operating system only to control interrupt handling, virtual storage, tracing facilities, etc.
  - Each register has 64 bits
  - In ESA/390 architecture, only the low-order 32 bits are accessible (bits 32-63)

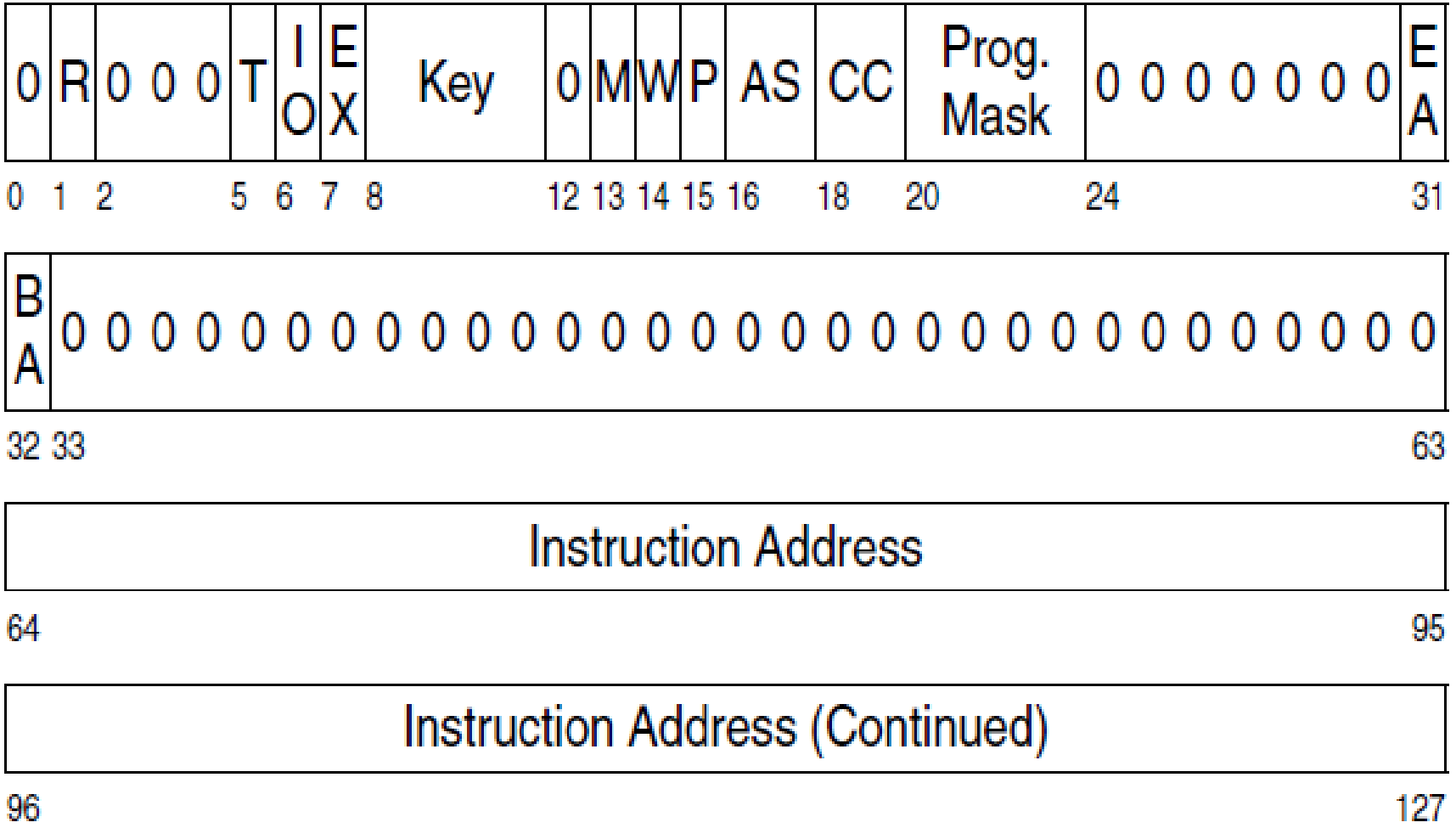
Program Status Word (PSW)

- **The PSW contains information required for the execution of the current program:**
  - Instruction address
  - Addressing mode
  - Condition code
  - Interrupt masks
  - Indicator problem/supervisor state (user/kernel mode)

Program Status Word – ESA/390 Architecture



Program Status Word – z/Architecture



Program Status Word – Fields

- **R – enable program event recording (PER)**
- **T – enable dynamic address translation (virtual storage)**
- **I – enable I/O interrupts**
- **E – enable external interrupts**
- **Key – define storage protection key**
- **M – enable machine checks**
- **W – wait state**
- **P – problem state (0 = supervisor state)**
- **AS – address space**
  - 00 = primary space mode
  - 01 = access register mode
  - 10 = secondary space mode
  - 11 = home space mode
- **CC – condition code**

Program Status Word – Fields (continued…)

- **Program mask**
  - Bit 20: Enable fixed-point overflow exception
  - Bit 21: Enable decimal overflow exception
  - Bit 22: Enable hex floating-point exponent underflow exception
  - Bit 23: Enable hex floating-point significance exception
- **EA – extended addressing (64-bit addressing mode, BA must also be 1)**
- **BA – basic addressing (31-bit addressing mode, 0 = 24-bit addressing mode)**
- **Instruction address**
  - It is stepped by the length of the current instruction

Problem State (User Mode)

- **In problem state, the program must not execute privileged instructions**
  - No access to control registers
  - No access to timers
  - No access to storage keys
  - Access only to “uncritical” parts of the PSW
  - In general: No access to architecture facilities that are vital to the system as a whole
- **If authorized, a program in problem state may execute certain semiprivileged instructions (e.g. PROGRAM CALL)**

Supervisor State (Kernel Mode)

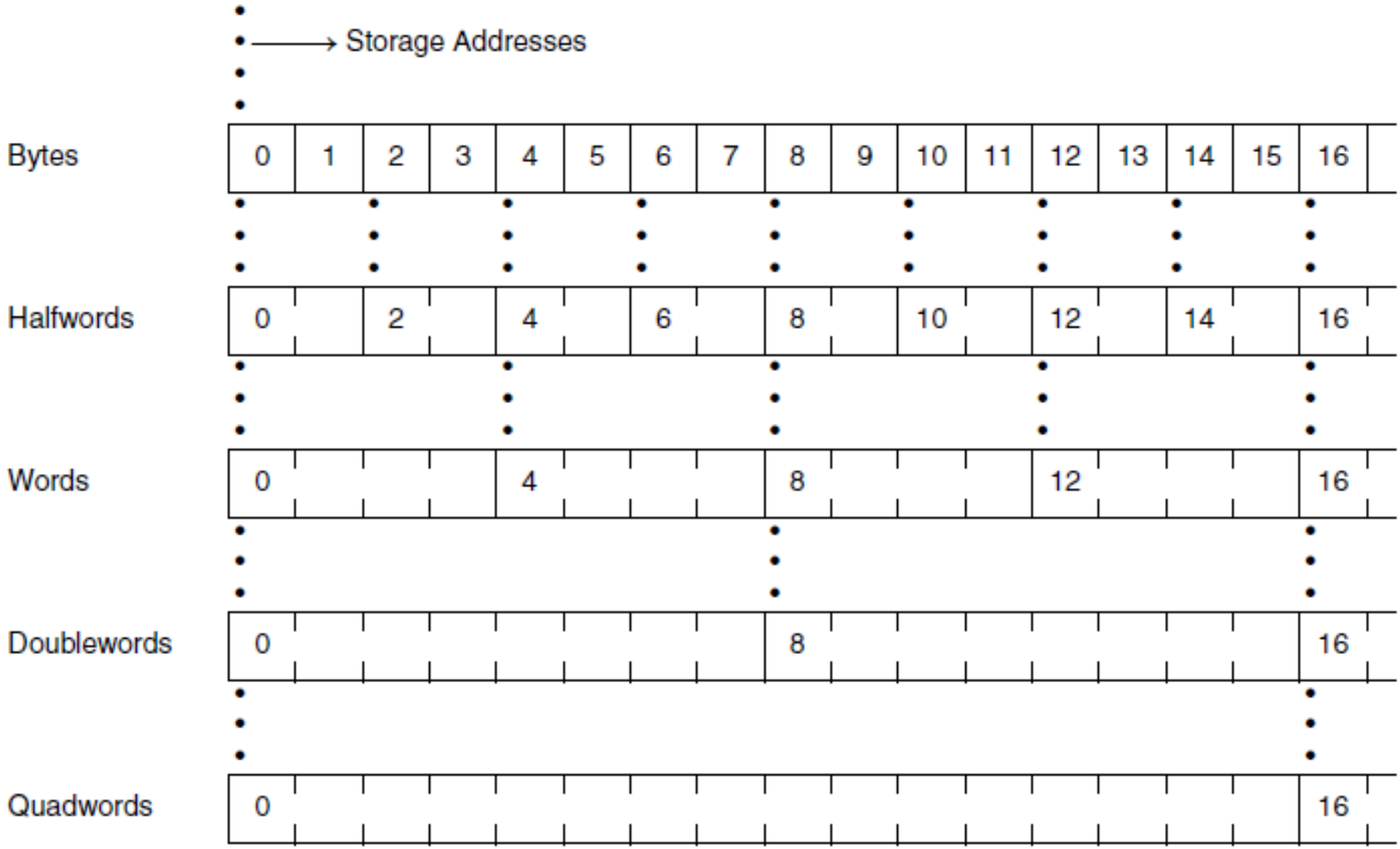
- **In supervisor state, a program may exploit all architecture facilities**
- **Transition from problem to supervisor state**
  - Via an interrupt (by loading a new PSW that indicates supervisor state)
- **Transition from supervisor to problem state**
  - Via the privileged LOAD PSW [EXTENDED] instruction

Storage

Storage

- **System z is a Big-Endian machine (like System p, unlike Intel)**
- **Storage addresses are byte addresses**
- **Types of address:**
  - **Virtual:** Translated by dynamic address translation (DAT) to real addresses
  - **Real:** Translated to absolute addresses using the prefix register
  - **Absolute:** After applying the prefix register
  - **Logical:** The address seen by the program (this can either be a virtual or a real address)

Integral Boundaries



Alignment Requirements

- **Instructions are always aligned on halfword boundaries**
- **Operands of non-privileged instructions normally *do not have to be aligned* on an integral boundary**
  - Exception: Compare and Swap
- **Operands of privileged instructions normally *must be aligned* on an integral boundary**
  - Example: Set CPU Timer

Storage Addressing Modes

- **There are three addressing modes:**

PSW.31 (EA)	PSW.32 (BA)	Addressing mode	Address range
0	0	24-bit	16M
0	1	31-bit	2G
1	0	Invalid	n/a
1	1	64-bit	16E (exa-bytes)

- **The instructions SAM24, SAM31, and SAM64 can be used to switch the addressing mode**

Interrupts

Interrupts

- **There are six classes of interrupts:**
  - Supervisor call
  - Program
  - Machine check
  - External
  - Input/output
  - Restart
- **Each class is associated with a pair of old/new PSWs in the assigned storage locations**

Interrupt Action

- **An interrupt comprises the following steps:**
  - The current PSW is stored in the assigned location named “old PSW”, e.g. “program old PSW” for a program interrupt
  - Additional information, such as an interrupt code, is stored
  - A new PSW is loaded from an assigned location, e.g. the “program new PSW”
- **No registers are saved, this is done by software**

PSWs in the Assigned Storage Locations

Real addresses	Contents
0x120 – 0x012F	Restart old PSW
0x130 – 0x013F	External old PSW
0x140 – 0x014F	Supervisor-call old PSW
0x150 – 0x015F	Program old PSW
0x160 – 0x016F	Machine-check old PSW
0x170 – 0x017F	I/O old PSW
0x1A0 – 0x01AF	Restart new PSW
0x1B0 – 0x01BF	External new PSW
0x1C0 – 0x01CF	Supervisor-call new PSW
0x1D0 – 0x01DF	Program new PSW
0x1E0 – 0x01EF	Machine-check new PSW
0x1F0 – 0x01FF	I/O new PSW

Interrupt Masking

- **I/O interrupts are masked by PSW.6**
  - Additionally, there are subclass mask bits in control register 6:
    - I/O interruption subclass (ISC) 0
    - …
    - I/O interruption subclass (ISC) 7
- **External interrupts are masked by PSW.7**
  - Additionally, there are subclass mask bits in control register 0:
    - CPU timer
    - Clock comparator
    - …
- **Repressible machine checks are masked by PSW.13**
  - Additionally, there are subclass mask bits in control register 14:
    - Channel report
    - Degradation
    - …

Interrupt Masking (continued…)

- **Some program interrupts are masked by PSW.20 – PSW.23:**
  - Fixed-point overflow
  - Decimal overflow
  - Hexadecimal-floating-point exponent underflow
  - Hexadecimal-floating-point significance

- **All other program interrupts cannot be masked**

Interrupt Masking (continued…)

- **There is no masking for**
  - **Supervisor calls (SVCs)**
    - The whole purpose of the SUPERVISOR CALL instruction is to invoke the supervisor via the interrupt mechanism
  - **Restart**
    - SIGNAL PROCESSOR instruction, typically issued by the operating system during startup
    - Manual operation available from the support element (SE) intended to restart the operating system
  - **Exigent machine checks**
    - If PSW.13 is 0, the CPU check stops. An example of such a situation is instruction processing damage.

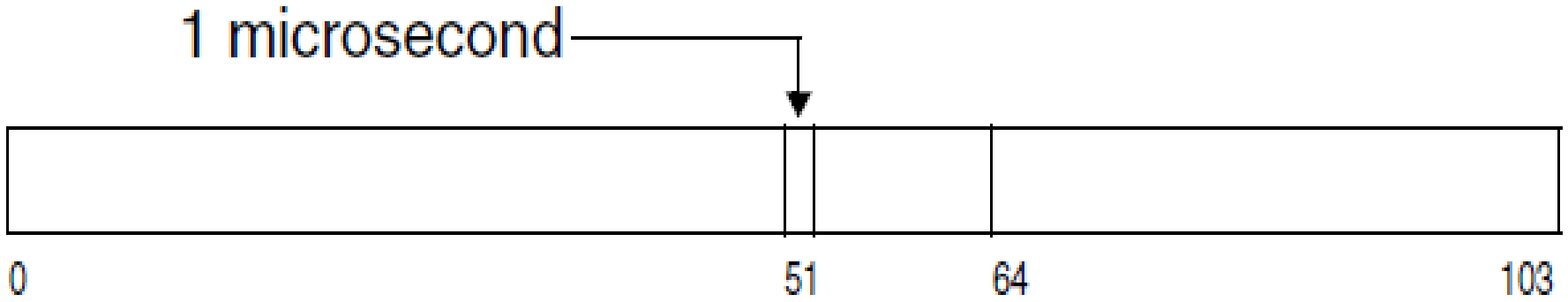
Timing Facilities

Timing Facilities

- **Time-of-day (TOD) clock**
  - one for the system
- **Clock comparator**
  - one per CPU
- **CPU timer**
  - one per CPU

Time-of-Day Clock

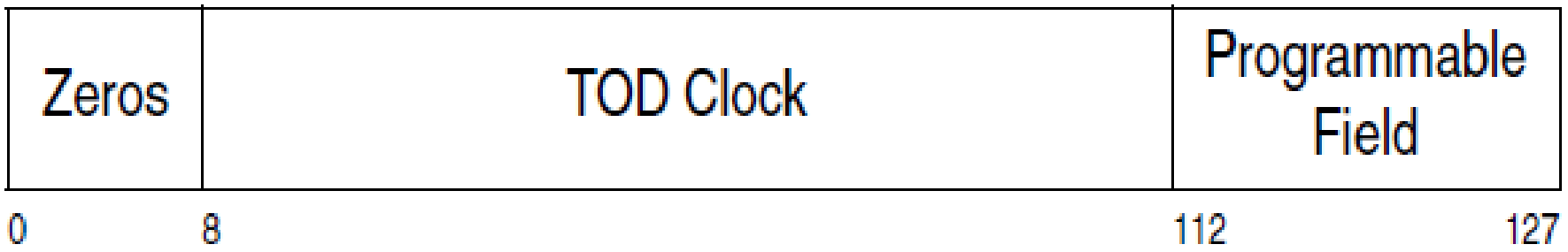
- **Format of the TOD clock:**



- **Value 0 defined as January 1, 1900, 00:00:00 UTC**
- **Overflows on September 17, 2042, 23:53:47 UTC**
- **STORE CLOCK (STCK) instruction returns first 64 bits**

Time-of-Day Clock (continued…)

- **STORE CLOCK EXTENDED (STCKE) returns 128 bits:**



- **First 8 bits (“zeros”) will be used after September 17, 2042 (good until ~ year 38,400 A.D.)**
- **Bit 59 equals 1 microsecond**
- **Bit 111 equals 222 * 10^{-24} seconds**
- **Programmable field used to generate unique value (set by SET CLOCK PROGRAMMABLE FIELD instruction, SCKPF)**

Clock Comparator

- **Same format as TOD clock (bit 51 = 1 microsecond)**
- **Continuously compared to TOD clock**
- **When TOD clock passes the clock comparator value, an external interrupt is generated (code 0x1004)**
- **Good for real-time measurements**
- **Set with SET CLOCK COMPARATOR (SCKC), read with STORE CLOCK COMPARATOR (STCKC)**

CPU Timer

- **Same format as TOD clock (bit 51 = 1 microsecond), but bit 0 is sign bit**
- **Stepped backwards**
- **When the CPU timer is negative, an external interrupt is generated (code 0x1005)**
- **Stopped when CPU stops**
- **Set with SET CPU TIMER (SPT), read with STORE CPU TIMER (STPT), EXTRACT CPU TIME (ECTG)**

Timer Stepping

- **On a real running system, TOD clock and CPU timer are stepped at the same rate.**

- **On a virtual system, the CPU timer is stepped only when the virtual system is dispatched, so it may appear to step slower than the TOD clock.**

  Assume (on z/VM) clock comparator is set to TOD clock + 5 seconds, CPU timer is set to 2 seconds.

  You don't know in advance who expires first.

TOD-Clock Steering

- **Used to correct the TOD clock in a running system**
  - E.g. after retrieving time from a Coordinated Time Server (CTS), i.e., an atomic clock
- **Used to keep TOD clock of multiple systems synchronized**
- **Maintains a TOD-offset register that is added to the TOD clock returned by STORE CLOCK**
- **By adjusting the value in the TOD-offset register, the pace of the TOD clock is steered**

Instructions

Instruction Set

- **S/360 (November 1970) had 143 instructions**
- **System z196 (September 2010) has 966 instructions**
- **Groups:**
  - General instructions
  - Decimal instructions
  - Floating-point instructions
  - Control instructions (privileged)
  - I/O instructions (privileged)
- **For comparison: System p also has over 700 instructions, half of them being vector-related operations**

Instruction Set (continued…)

- **Additionally, some instructions are not described in the Principles of Operation:**
  - Coupling instructions (Parallel Sysplex)
  - Queued-directed I/O, HiperSockets
  - Dynamic I/O configuration (HCD)
  - Service Call (SCLP)
  - Instructions associated with logical partitioning (LPAR) and virtualization (z/VM)
  - New with z10: CPU Measurement Facility
- **Most of these instructions are privileged**

General Instructions (1)

- **Load into/store from general registers**
  - LOAD, INSERT, STORE
  - 8, 16, 32, and 64 bit
- **Binary integer arithmetic**
  - ADD, SUBTRACT, MULTIPLY, DIVIDE
  - 16, 32, and 64 bit
  - Signed (2-complement)
    - From 0 to 2,147,483,647: `0 – 0x7FFFFFFF` (32 bit)
    - From -1 to -2,147,483,648: `0xFFFFFFFF – 0x80000000`
  - Unsigned (LOGICAL)
    - From 0 to 4,294,967,295: `0 – 0xFFFFFFFF` (32 bit)
- **Shift/rotate operations**
  - SHIFT (left and right, signed arithmetic and logical), ROTATE (left)
  - 32 and 64 bit
- **Bitwise logical operations**
  - AND, OR, EXCLUSIVE OR
  - 8, 16, 32, and 64 bit, 1 – 256 bytes
  - Combined ROTATE THEN INSERT / AND / OR / XOR SELECTED BITS

General Instructions (2)

- **Comparisons**
  - Signed arithmetic and unsigned (LOGICAL)
  - 16, 32, and 64 bit
- **Branches (absolute and relative, looping instructions)**
- **Subroutine linkage**
  - BRANCH AND SAVE [AND SET MODE]
- **Bit testing and counting**
  - TEST UNDER MASK
  - FIND LEFTMOST ONE
  - **New with z196:** POPULATION COUNT
- **Storage-to-storage copy and compare**
  - MOVE [LONG [EXTENDED]]
  - COMPARE LOGICAL [LONG [EXTENDED]]
- **Conversion to/from packed decimal format**
  - CONVERT TO BINARY / DECIMAL

General Instructions (3)

- **String processing**
  - TRANSLATE, TRANSLATE AND TEST
  - SEARCH STRING, COMPARE LOGICAL STRING (for null-terminated strings in C)
- **Conversion little / big Endian**
  - LOAD REVERSED, STORE REVERSED
- **Checksum generation**
- **Sorting**
  - COMPARE AND FORM CODEWORD, UPDATE TREE
- **Encryption**
  - CIPHER MESSAGE
- **Atomic updates, locking**
  - COMPARE AND SWAP, PERFORM LOCKED OPERATION
  - New with z196: LOAD AND ADD / AND / OR / XOR

- **Note: Some possibly long-running instructions are interruptible**

Decimal Instructions

- **Use packed decimal format**
  - One decimal digit per 4 bits (hex digit) encoded 0 – 9
  - 1 – 31 decimal digits (always odd number)
  - Rightmost hex digit is sign (A, C, E, F mean plus, B, D mean minus)
  - e. g., 0x123C is decimal +123, 0x456D is decimal -456
  - Used a lot in commercial applications (COBOL, PL/I)
- **Integer arithmetic (+, -, *, /)**
- **Comparison**
- **Data validation**
- **Conversion to printable format (EBCDIC)**
  - EDIT, EDIT AND MARK

Floating-Point Instructions

- **Three precisions**
  - Short (32-bit), ca. 6 – 7 decimal digits
  - Long (64-bit), ca. 16 – 17 decimal digits
  - Extended (128-bit), ca. 33 – 34 decimal digits
- **Binary floating-point format (BFP) characteristics**
  - IEEE 754 standard
  - Number range (long precision): ~ $4.9 \times 10^{-324} \leq M \leq \sim 1.8 \times 10^{308}$
  - Supports infinity and NaN (not-a-number)
- **Decimal floating-point format (DFP) characteristics**
  - Introduced with System z9 GA3 (millicode), **hardware implementation on System z10 and later**
  - IEEE P754 standard
  - Number range (long precision): $1 \times 10^{-398} \leq M \leq (10^{16} - 1) \times 10^{369}$
  - Supports infinity and NaN
  - Exact representation of decimal fractions (e.g. 0.1)
- **Hexadecimal floating-point (HFP) characteristics**
  - System z unique, introduced in S/360
  - Number range (any precision): ~ $5.4 \times 10^{-79} \leq M \leq \sim 7.2 \times 10^{75}$
  - Does not support infinity and NaN

Floating-Point Instructions (continued…)

- One set of floating-point registers used for all formats
- Load into/store from floating-point registers
- Floating-point arithmetic (+, -, *, /, square root)
- Comparison
- Conversion to/from binary integer
- Conversion between BFP/DFP/HFP formats

Control and I/O Instructions

- **All instructions are privileged, i. e., available to the operating system only**
- **Operate on vital system resources**
  - Control registers
  - Program status word
  - Storage keys
  - DAT tables
  - Timers
  - Real storage
  - etc.
- **Even read access to resources such as CPU timer is privileged**

Instruction Execution

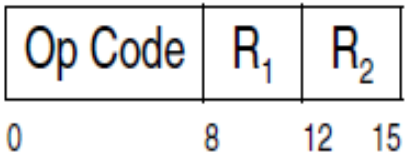
- **After the instruction has been fetched, the instruction address in the PSW is incremented by the instruction length (2, 4, or 6 bytes)**
- **An instruction ends in one of the following ways:**
  - Completion and partial completion
    - This is the normal end
  - Suppression
    - Instruction is not executed, but PSW instruction address has been updated
    - Occurs with most of the program interrupt conditions (program old PSW points after the failing instruction)
  - Nullification
    - Instruction is not executed and PSW instruction address has not been updated
    - Occurs with program interrupt conditions that pertain to DAT handling (i.e. the instruction will be executed after the page has been loaded)
    - Occurs also for some interruptible instructions (MVCL, CLCL) that resume at the point of interrupt
  - Termination
    - Instruction may have been partially executed and PSW instruction address has been updated (very rare situation)

Instruction Formats

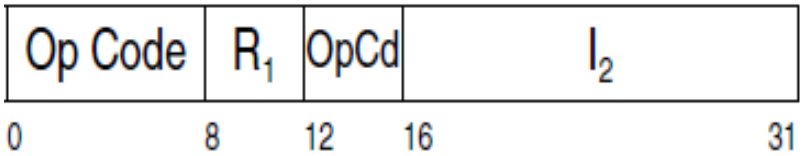
- **An instruction is 2, 4, or 6 bytes in length**
- **It is always halfword-aligned**
- **The first two bits of an instruction determine its length:**
  - 00: 2 bytes
  - 01 and 10: 4 bytes
  - 11: 6 bytes
- **The operation code consists of 8, 12, or 16 bits**
  - An 8-bit operation code always occupies the first byte
  - A 12-bit operation code occupies the first byte and the second half of the second byte
  - A 16-bit operation code occupies the first byte and either the second byte or the last byte of a 6-byte instruction

Examples for 8-, 12-, and 16-bit operation codes

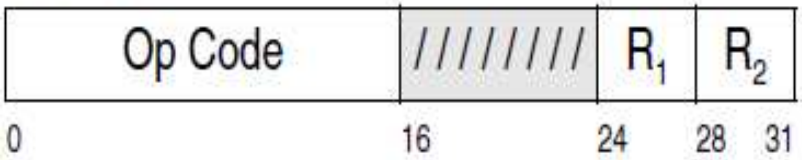
RR Format



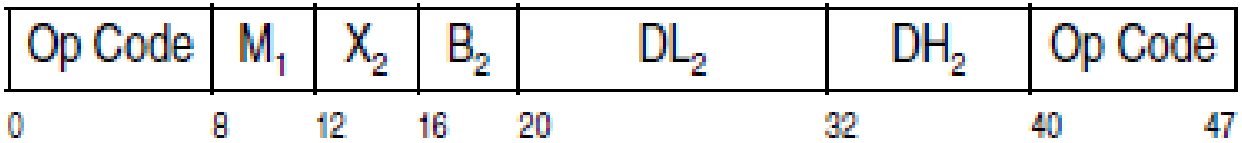
RI Format



RRE Format



RXY Format



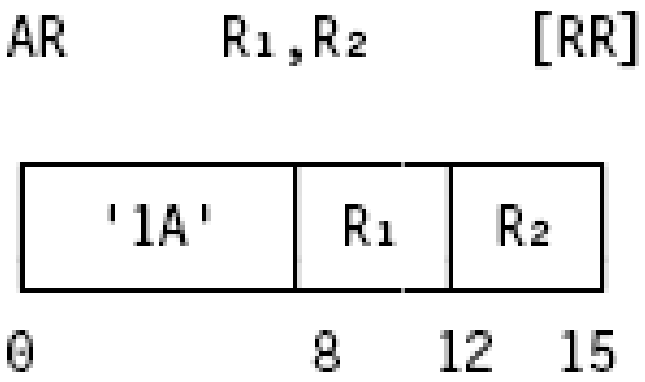
Instruction Execution

- **After the instruction has been fetched, the instruction address in the PSW is incremented by the instruction length (2, 4, or 6 bytes)**
- **An instruction ends in one of the following ways:**
  - Completion and partial completion
    - This is the normal end
  - Suppression
    - Instruction is not executed, but PSW instruction address has been updated
    - Occurs with most of the program interrupt conditions (program old PSW points after the failing instruction)
  - Nullification
    - Instruction is not executed and PSW instruction address has not been updated
    - Occurs with program interrupt conditions that pertain to DAT handling (i.e. the instruction will be executed after the page has been loaded)
    - Occurs also for some interruptible instructions (MVCL, CLCL) that resume at the point of interrupt
  - Termination
    - Instruction may have been partially executed and PSW instruction address has been updated (very rare situation)

Instruction Format RR

- **AR R1,R2**

adds the contents of general register 2 to the contents of general register 1. Both operands are 32-bit and signed. In storage, the instruction appears as 0x1A 12 (operation code, operands):



Instruction Format RRE

- **AGR R1,R2**

  adds the contents of general register 2 to the contents of general register 1. Both operands are 64-bit and signed. In storage, the instruction appears as 0xB908 0012:

```
AGR      R1,R2      [RRE]
```

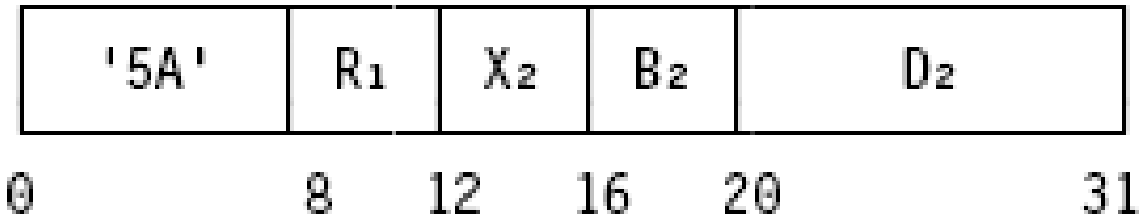
The third byte of the instruction is ignored.

Instruction Format RX

- **A R1,4(R2,R3)**

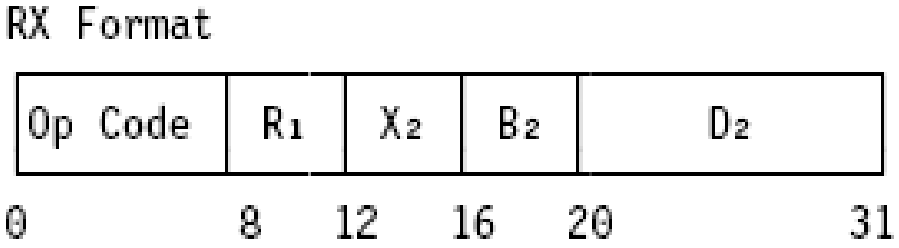
adds the contents of general register 2 to the contents of general register 3, then adds the value 4. The sum is the address of a fullword in storage whose contents is added to the contents of general register 1. In storage, the instruction appears as 0x5A 1 2 3 004:

```
A       R1,D2(X2,B2)       [RX]
```



Instruction Format RX (continued…)

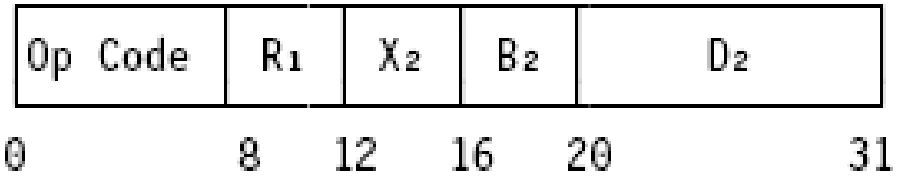
- **B_2 is called base register. If it is 0, the value 0 is used, not the contents of general register 0.**
- **D_2 is called displacement. It has 12 bits and thus ranges from 0 to 4095.**
- **X_2 is called index register. If it is 0, the value 0 is used, not the contents of general register 0.**
- **The actual length of the resulting address (24, 31, or 64 bits) is controlled by the addressing mode in the PSW.**



Instruction Format RX (continued…)

- **Nearly all instructions that access storage use the $D_2(B_2)$ operand.**
- **Many of them allow for the specification of an index register in the form $D_2(X_2,B_2)$. It is typically used to access array elements in a loop.**
- **If neither base nor index register is used, the displacement alone forms the address. This is used to access the assigned storage locations.**

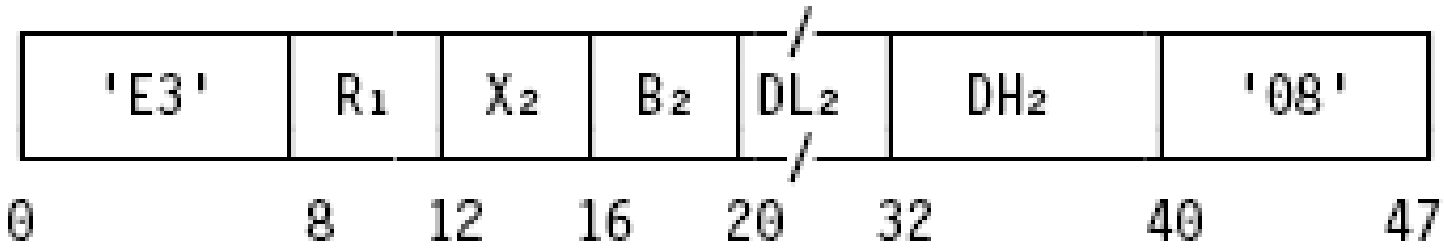
RX Format



Instruction Format RXY (long displacement)

- **The 12-bit displacement was considered insufficient, so many newer instructions use a 20-bit *signed* displacement:**

```
AG        R1,D2(X2,B2)        [RXY]
```



DH_2 is the high-order part of the displacement, so

AG R1,-4(R2,R3)

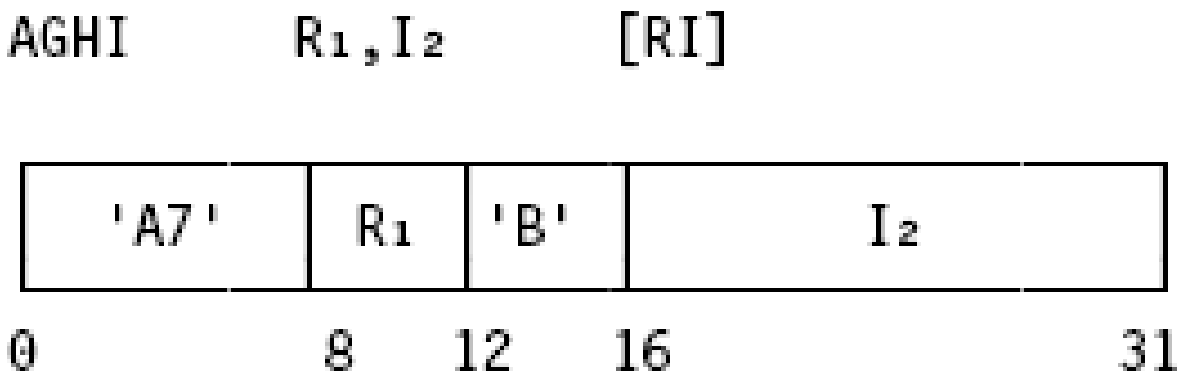
is assembled as 0xE3 1 2 3 FFC FF 08

(-4 = 0xFF FFC)

Instruction Format RI

- **AGHI R1,-3**

  adds -3 to general register 1 (64-bit):



The instruction is assembled as 0xA7 1 B FFFD

All 64 variations of ADD (z196)

Name	Mnemonic	Characteristics																			Op-code	Page
ADD (32)	A	RX-a	C			A			IF											B_2	5A	7-25
ADD (32)	AR	RR	C						IF												1A	7-24
ADD (32)	ARK	RRF-a	C	DO					IF												B9F8	7-25
ADD (32)	AY	RXY-a	C	LD		A			IF											B_2	E35A	7-25
ADD (64)	AG	RXY-a	C	N		A			IF											B_2	E308	7-25
ADD (64)	AGR	RRE	C	N					IF												B908	7-25
ADD (64)	AGRK	RRF-a	C	DO					IF												B9E8	7-25
ADD (64←32)	AGF	RXY-a	C	N		A			IF											B_2	E318	7-25
ADD (64←32)	AGFR	RRE	C	N					IF												B918	7-25
ADD (extended BFP)	AXBR	RRE	C				SP	Db	Xi		Xo	Xu	Xx								B34A	19-11
ADD (extended DFP)	AXTR	RRF-a	C	TF			SP	Dt	Xi		Xo	Xu	Xx								B3DA	20-19
ADD (extended DFP)	AXTRA	RRF-a	C	F			SP	Dt	Xi		Xo	Xu	Xx	Xq							B3DA	20-19
ADD (long BFP)	ADB	RXE	C			A		Db	Xi		Xo	Xu	Xx							B_2	ED1A	19-11
ADD (long BFP)	ADBR	RRE	C					Db	Xi		Xo	Xu	Xx								B31A	19-11
ADD (long DFP)	ADTR	RRF-a	C	TF				Dt	Xi		Xo	Xu	Xx								B3D2	20-19
ADD (long DFP)	ADTRA	RRF-a	C	F				Dt	Xi		Xo	Xu	Xx	Xq							B3D2	20-19
ADD (short BFP)	AEB	RXE	C			A		Db	Xi		Xo	Xu	Xx							B_2	ED0A	19-11
ADD (short BFP)	AEBR	RRE	C					Db	Xi		Xo	Xu	Xx								B30A	19-11
ADD DECIMAL	AP	SS-b	C			A		Dd	DF							ST	B_1	B_2			FA	8-6
ADD HALFWORD	AH	RX-a	C			A			IF											B_2	4A	7-27
ADD HALFWORD	AHY	RXY-a	C	LD		A			IF											B_2	E37A	7-27
ADD HALFWORD IMMEDIATE (32)	AHI	RI-a	C						IF												A7A	7-27
ADD HALFWORD IMMEDIATE (64)	AGHI	RI-a	C	N					IF												A7B	7-27
ADD HIGH (32)	AHHHR	RRF-a	C	HW					IF												B9C8	7-27
ADD HIGH (32)	AHHLR	RRF-a	C	HW					IF												B9D8	7-27
ADD IMMEDIATE (32)	AFI	RIL-a	C	EI					IF												C29	7-25
ADD IMMEDIATE (32←16)	AHIK	RIE-d	C	DO					IF												ECD8	7-25
ADD IMMEDIATE (32←8)	ASI	SIY	C	GE		A			IF							ST	B_1				EB6A	7-25
ADD IMMEDIATE (64←16)	AGHIK	RIE-d	C	DO					IF												ECD9	7-25
ADD IMMEDIATE (64←32)	AGFI	RIL-a	C	EI					IF												C28	7-25
ADD IMMEDIATE (64←8)	AGSI	SIY	C	GE		A			IF							ST	B_1				EB7A	7-25
ADD IMMEDIATE HIGH (32)	AIH	RIL-a	C	HW					IF												CC8	7-25
ADD LOGICAL (32)	AL	RX-a	C			A														B_2	5E	7-28
ADD LOGICAL (32)	ALR	RR	C																		1E	7-28
ADD LOGICAL (32)	ALRK	RRF-a	C	DO																	B9FA	7-28
ADD LOGICAL (32)	ALY	RXY-a	C	LD		A														B_2	E35E	7-28
ADD LOGICAL (64)	ALG	RXY-a	C	N		A														B_2	E30A	7-28
ADD LOGICAL (64)	ALGR	RRE	C	N																	B90A	7-28
ADD LOGICAL (64)	ALGRK	RRF-a	C	DO																	B9EA	7-28
ADD LOGICAL (64←32)	ALGF	RXY-a	C	N		A														B_2	E31A	7-28
ADD LOGICAL (64←32)	ALGFR	RRE	C	N																	B91A	7-28
ADD LOGICAL HIGH (32)	ALHHHR	RRF-a	C	HW																	B9CA	7-29
ADD LOGICAL HIGH (32)	ALHHLR	RRF-a	C	HW																	B9DA	7-29
ADD LOGICAL IMMEDIATE (32)	ALFI	RIL-a	C	EI																	C2B	7-28
ADD LOGICAL IMMEDIATE (64←32)	ALGFI	RIL-a	C	EI																	C2A	7-28
ADD LOGICAL WITH CARRY (32)	ALC	RXY-a	C	N3		A														B_2	E398	7-29
ADD LOGICAL WITH CARRY (32)	ALCR	RRE	C	N3																	B998	7-29
ADD LOGICAL WITH CARRY (64)	ALCG	RXY-a	C	N		A														B_2	E388	7-29
ADD LOGICAL WITH CARRY (64)	ALCGR	RRE	C	N																	B988	7-29
ADD LOGICAL WITH SIGNED IMMEDIATE (32←16)	ALHSIK	RIE-d	C	DO																	ECDA	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (32←8)	ALSI	SIY	C	GE		A										ST	B_1				EB6E	7-30

Name	Mnemonic	Characteristics																			Op-code	Page
ADD LOGICAL WITH SIGNED IMMEDIATE (64←16)	ALGHSIK	RIE-d	C	DO																	ECDB	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (64←8)	ALGSI	SIY	C	GE		A										ST	B_1				EB7E	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIH	RIL-a	C	HW																	CCA	7-31
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIHN	RIL-a		HW																	CCB	7-31
ADD NORMALIZED (extended HFP)	AXR	RR	C				SP	Da	EU	EO	LS										36	18-8
ADD NORMALIZED (long HFP)	AD	RX-a	C			A		Da	EU	EO	LS							B_2			6A	18-8
ADD NORMALIZED (long HFP)	ADR	RR	C					Da	EU	EO	LS										2A	18-8
ADD NORMALIZED (short HFP)	AE	RX-a	C			A		Da	EU	EO	LS							B_2			7A	18-8
ADD NORMALIZED (short HFP)	AER	RR	C					Da	EU	EO	LS										3A	18-8
ADD UNNORMALIZED (long HFP)	AW	RX-a	C			A		Da		EO	LS							B_2			6E	18-9
ADD UNNORMALIZED (long HFP)	AWR	RR	C					Da		EO	LS										2E	18-9
ADD UNNORMALIZED (short HFP)	AU	RX-a	C			A		Da		EO	LS							B_2			7E	18-9
ADD UNNORMALIZED (short HFP)	AUR	RR	C					Da		EO	LS										3E	18-9

The Condition Code in the PSW

- **The condition code (cc) in the PSW is set by many, but not all instructions**
- **If an instruction does not set the cc, it remains unchanged**
- **Examples of instructions that set the cc:**
  - Add
  - Subtract
  - Compare
- **Examples of instructions that do not set the cc:**
  - Load
  - Store
  - Multiply
  - Divide
- **There are conditional branching instructions that act depending on the condition code**

The Condition Code in the PSW (continued...)

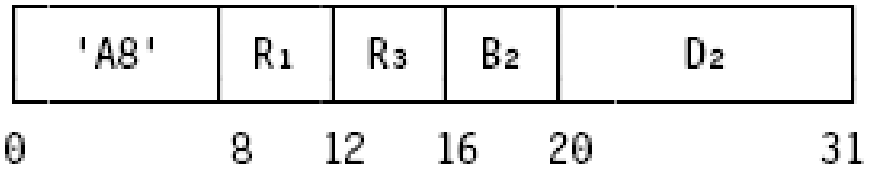
- **The meaning of the cc is individual for an instruction but there are common rules**
  - cc = 0 means “equal operands” for comparisons, “zero result” for add and subtract operations
  - cc = 1 means “first operand is low” for comparisons, “negative result” for add and subtract operations
  - cc = 2 means “first operand is high” for comparisons, “positive result” for add and subtract operations
  - cc = 3 is used for various purposes, e.g. to indicate overflow
- **A subsequent branch instruction is used to act on that condition code**
- **New with z196: Conditional LOAD / STORE instructions perform the requested operation or not, depending on the condition code, thus avoiding branches**

Long-Running Instructions

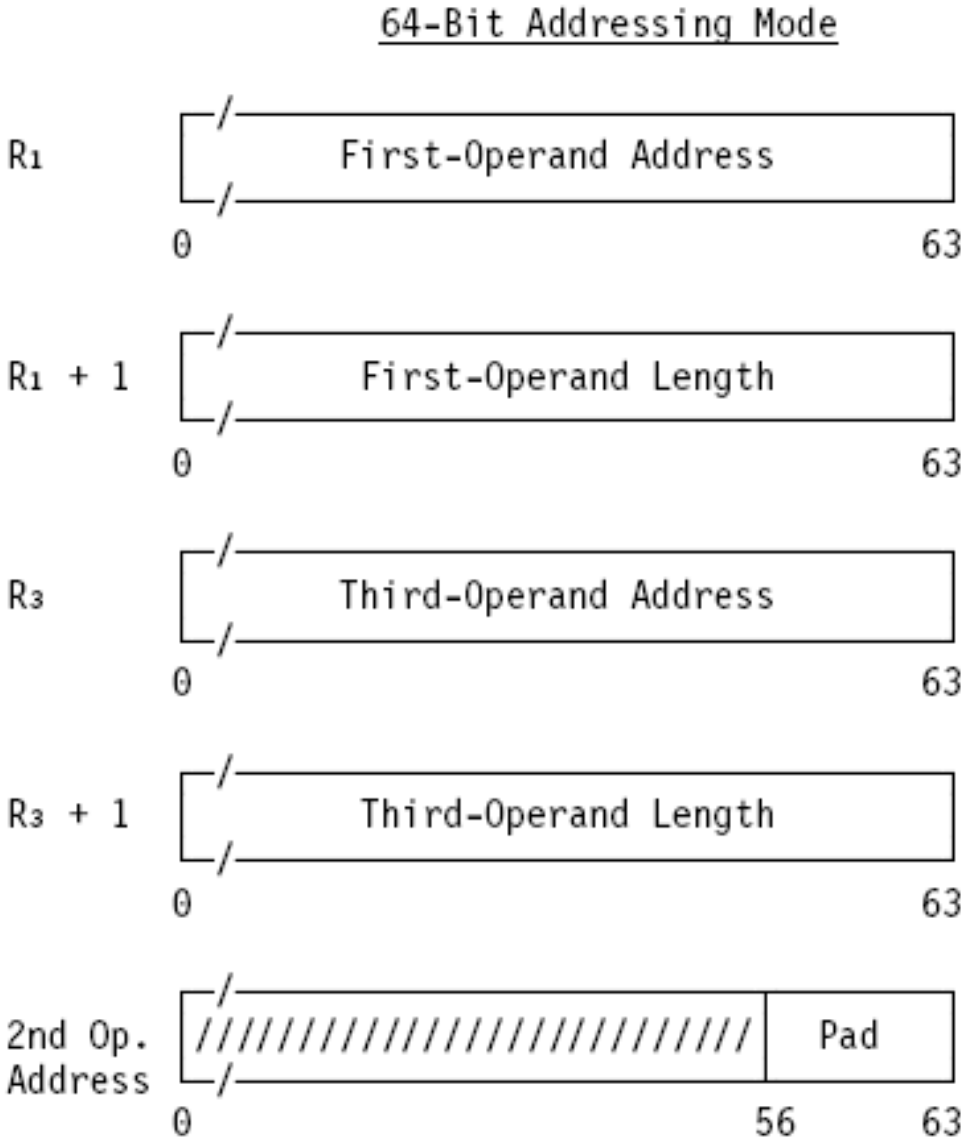
- **Some instructions may process large amounts of data**
  - Excessive time required to complete execution
  - Instruction accesses many pages of storage
    - Not all pages are necessarily available at the same time in a virtual storage environment
- **Some of these instructions are interruptible**
- **Other instructions indicate partial completion with cc 3**

Long-Running Instruction: MOVE LONG EXTENDED

```
MVCLE    R1,R3,D2(B2)      [RS]
```



- **On completion**
  - The addresses are incremented
  - The lengths are decremented
  - If completion was partial, cc 3 is set



Partial Completion

- **MOVE LONG EXTENDED (and a few others) may perform *partial completion:***
  - The registers that describe their operands are updated
  - cc 3 is set to indicate partial completion
  - Software has to branch back to the instruction on cc 3:

```
LOOP     MVCLE    R2,R4,0
         BRC      B'0001',LOOP
```

  - Interrupts may occur after each execution
  - Software has the option to do something else before resuming the instruction
  - This approach is used for all new long-running instructions

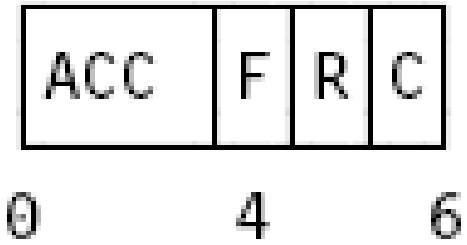
Storage Protection

Storage Protection Mechanisms

- **Key-controlled protection**
- **DAT protection (formerly called page protection)**

Storage Keys

- **A storage key is associated with each 4K-byte block of real storage. The storage key has the following format:**



- **ACC = access-control bits**
- **F = fetch-protection bit**
- **R = reference bit**
- **C = change bit**

Storage Key Protection Rules

Conditions		Is Access to Storage Permitted?	
Fetch-Protection Bit of Storage Key	Key Relation	Fetch	Store
0	Match	Yes	Yes
0	Mismatch	Yes	No
1	Match	Yes	Yes
1	Mismatch	No	No

Explanation:

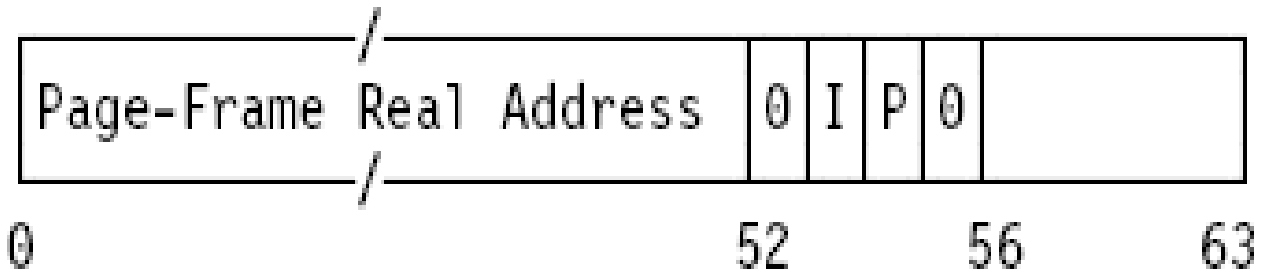
Match — The four access-control bits of the storage key are equal to the access key, or the access key is zero.

Yes — Access is permitted.

No — Access is not permitted. On fetching, the information is not made available to the program; on storing, the contents of the storage location are not changed.

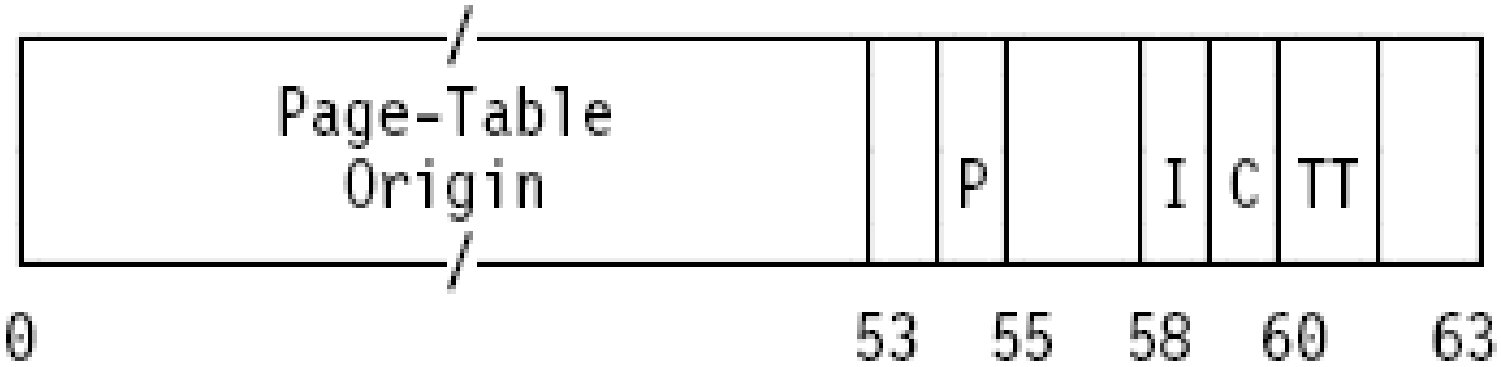
DAT Protection

- **Prevents a page (4K) from being altered, does not provide fetch protection**
- **Used, for instance, to implement the POSIX fork() function**
- **Enabled by setting the P bit in a page-table entry (PTE):**



DAT Protection (continued…)

- **May be applied to an entire segment (1M) by setting the P bit in the segment-table entry:**



- **New with z10:**

  May be applied to an entire region (2G, 4T, 8P) by setting the P bit in the region-table entry

Virtual Storage

Virtual Storage

- **Virtual storage is created by multi-level lookup tables in storage that describe the virtual-to-real address translation. This process is called dynamic address translation (DAT).**
- **The base pointers to the top-level tables are kept in control registers (CR1, CR7, and CR13) or are described by access registers.**
- **Multiple address spaces may be accessed at the same time.**

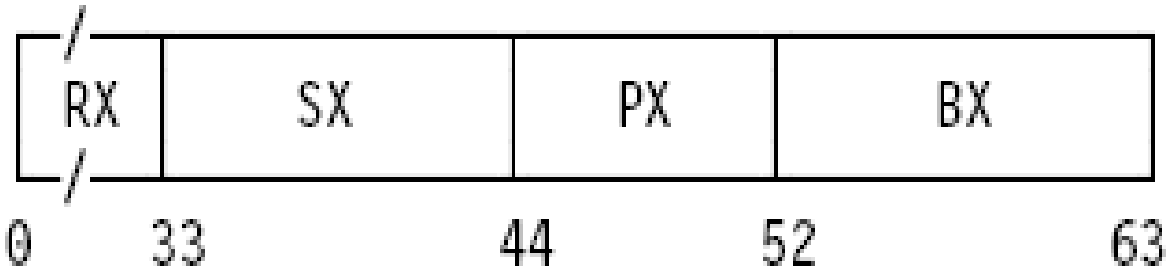
Virtual Storage – Address Space Control

Several PSW bits control which address space is used:

PSW Bit			DAT	Mode	Handling of Addresses	
5	16	17			Instruction Addresses	Logical Addresses
0	0	0	Off	Real mode	Real	Real
0	0	1	Off	Real mode	Real	Real
0	1	0	Off	Real mode	Real	Real
0	1	1	Off	Real mode	Real	Real
1	0	0	On	Primary-space mode	Primary virtual	Primary virtual
1	0	1	On	Access-register mode	Primary virtual	AR-specified virtual
1	1	0	On	Secondary-space mode	Primary virtual	Secondary virtual
1	1	1	On	Home-space mode	Home virtual	Home virtual

Virtual Storage (z/Architecture)

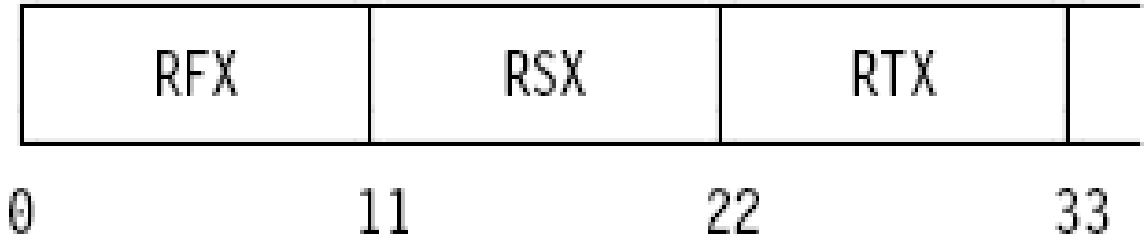
- **In z/Architecture, an address has up to 64 bits:**



- **An address space in virtual storage is up to 16E in size, consisting of**
  - 2G regions
  - 1M segments
  - 4K pages

Virtual Storage (z/Architecture) continued…

- **The region index consists of 33 bits:**



- **It is subdivided in 3 groups of 11 bits each: Region-first, region-second, and region-third index.**

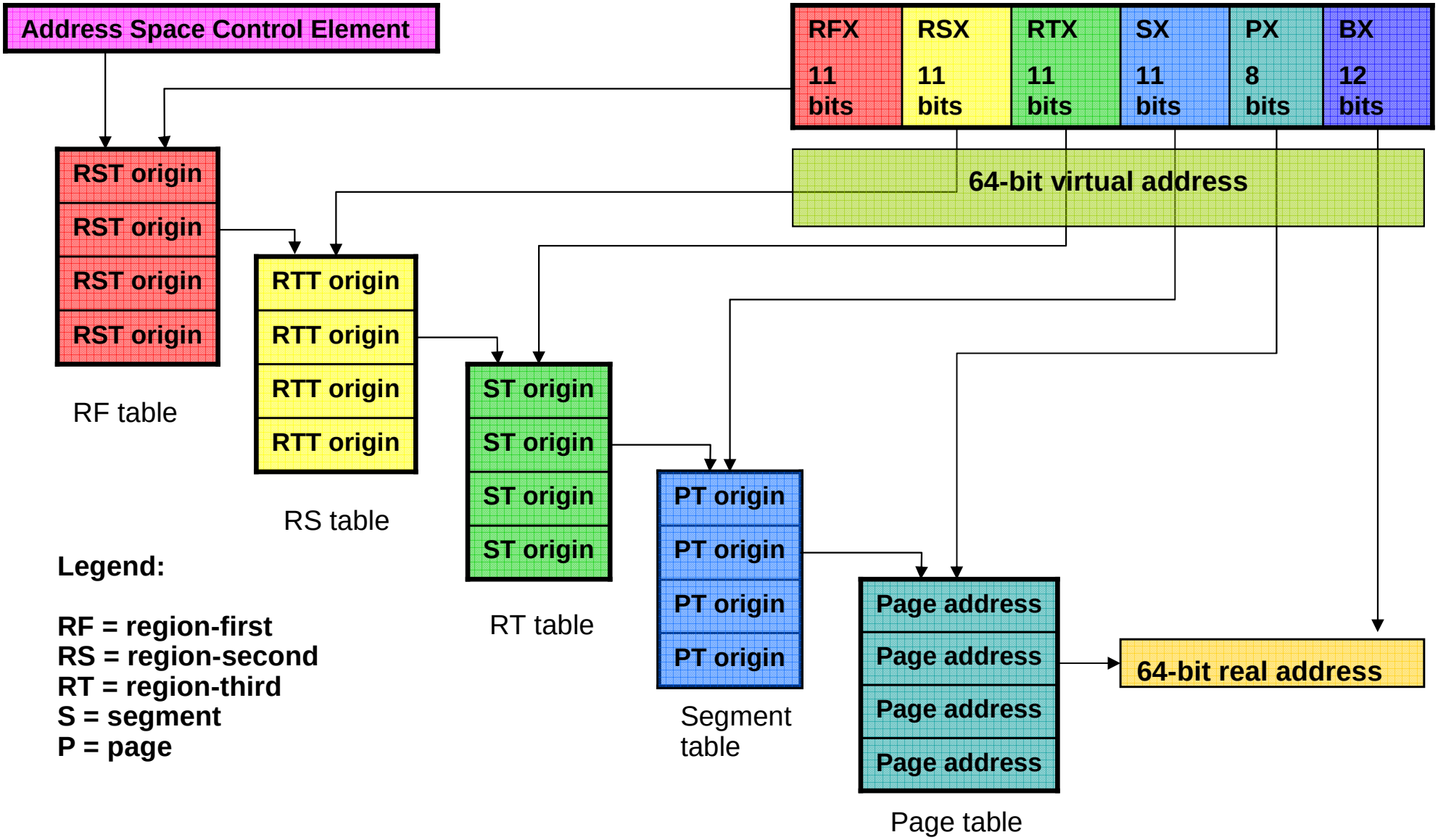
Virtual Storage (z/Architecture) continued…

- **Since a virtual address consists of six parts (region-first, region-second, region-third, segment, page, base), a five-level table lookup would be required:**
  - Costly in terms of performance
  - Currently unnecessary, because even a huge 4T address space can be handled with a region-third index (region-first and region-second will always be zero)
- **Therefore, z/Architecture DAT allows to specify at which level translation is to start (region-first, region-second, region-third, or segment)**

Maximum Address Space Sizes

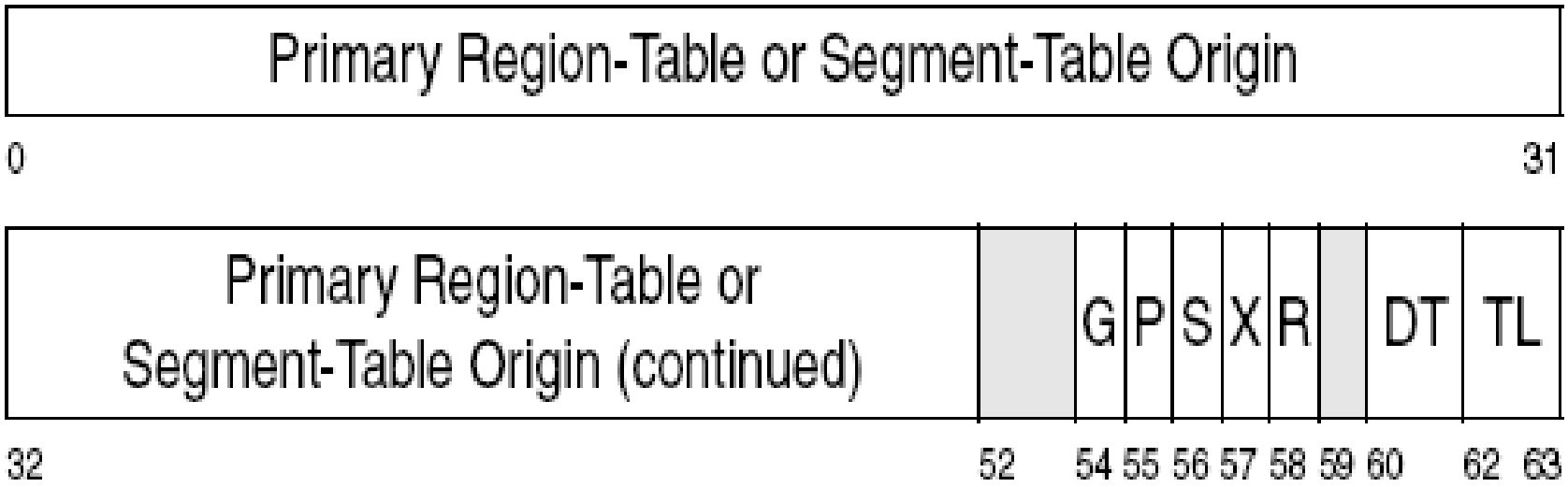
DAT table used	Maximum address space size
Segment table	2 Gigabytes (2^{31})
Region-third table	4 Terabytes (2^{42})
Region-second table	8 Petabytes (2^{53})
Region-first table	16 Exabytes (2^{64})

z/Architecture Dynamic Address Translation



z/Architecture Address Space Control Element

- **An address space control element (ASCE) describes an address space, e.g.:**

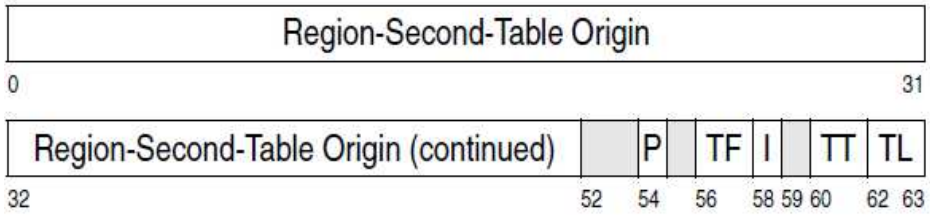


- **It is contained in CR1, CR7, CR13, or described by an access register**

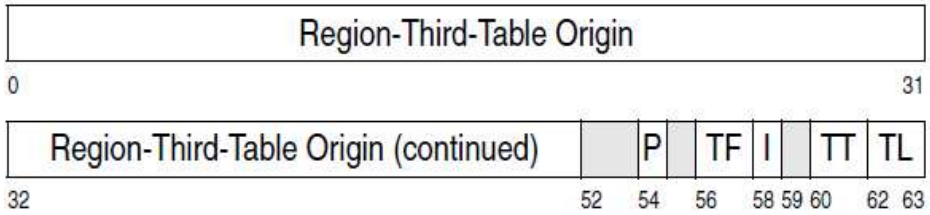
z/Architecture DAT Table Entries

- **The region-table entries all have the same format:**

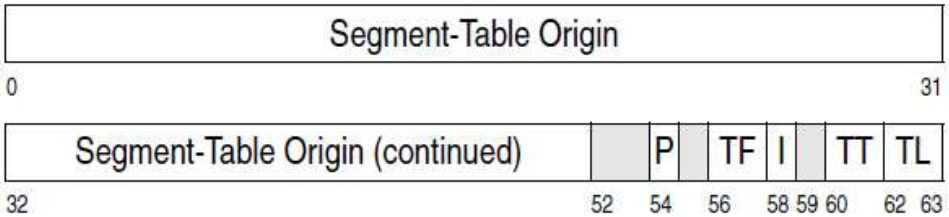
Region-First-Table Entry (TT=11)



Region-Second-Table Entry (TT=10)

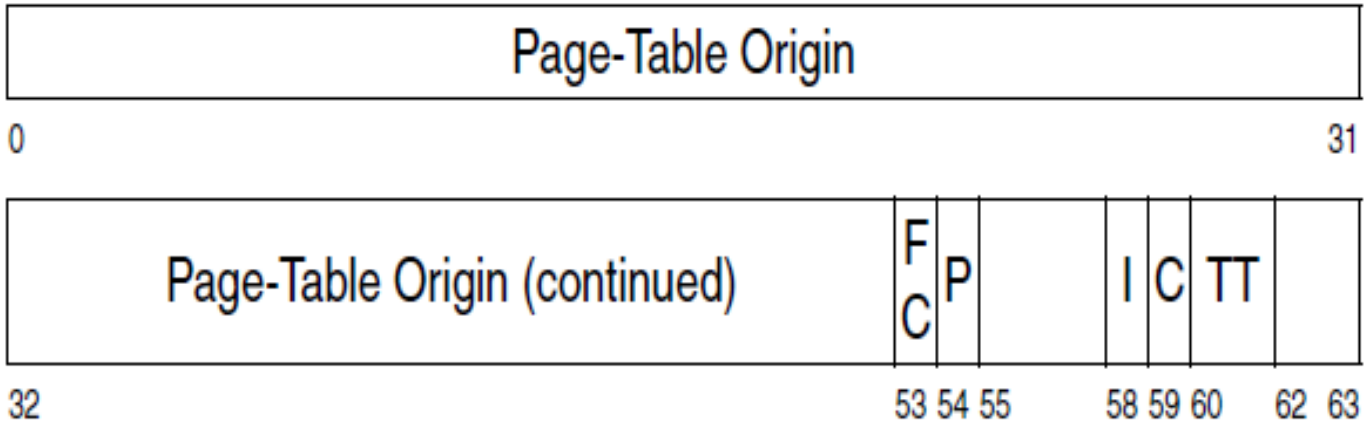


Region-Third-Table Entry (TT=01)

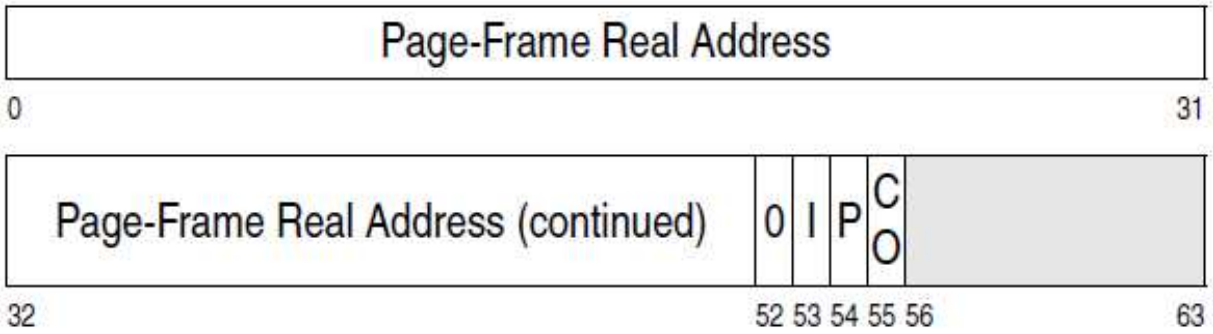


z/Architecture DAT Table Entries (continued…)

- **The segment-table entries look as follows:**



- **And finally here are the page-table entries:**



Translation-Lookaside Buffer

- **To speed up the translation process, translations are cached in a TLB**
- **Each CPU has its own TLB**
- **The TLB is filled in automatically by hardware as the program executes**
- **When DAT tables are changed, TLB entries must be purged**

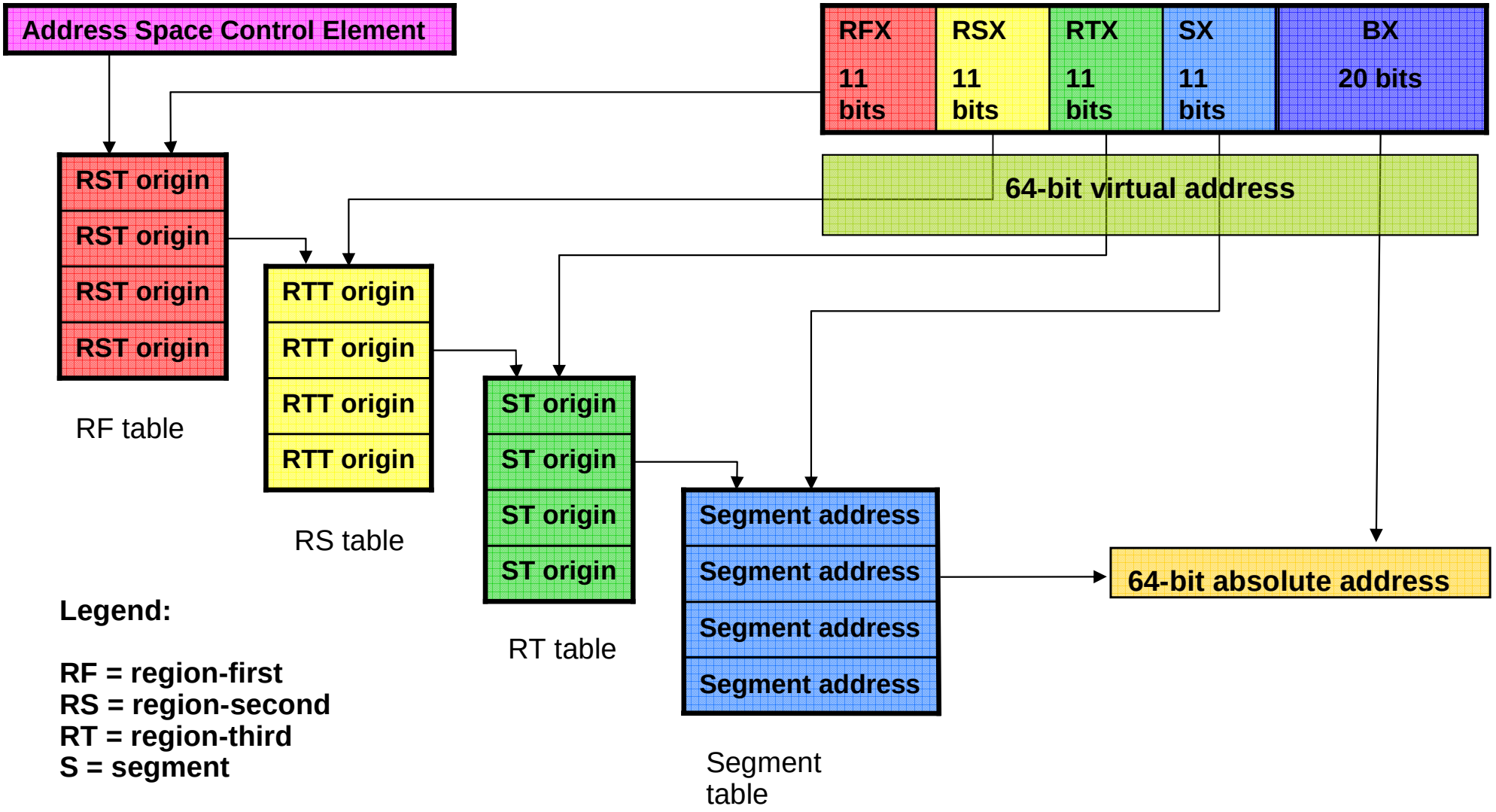
Translation-Lookaside Buffer (continued…)

- **The TLB may be purged completely by the instruction PURGE TLB (PTLB)**
  - This clears the TLB of the CPU issuing PTLB
- **Selected entries may be purged by**
  - INVALIDATE PAGE TABLE ENTY (IPTE)
  - INVALIDATE DAT TABLE ENTRY (IDTE)
  - COMPARE AND SWAP AND PURGE (CSP, CSPG)
  - The entry is updated in the DAT tables as well as in the TLBs of all CPUs in the configuration

System z10 DAT Enhancements

- **With the growing size of address spaces, page tables become huge**
  - Remember: One page table entry covering 4K bytes requires 8 bytes. E. g., to map 4G, you need page tables occupying 8M of storage
- **With z10, a segment table entry may either point to**
  - a page table (as before) **or**
  - directly to a 1M area in absolute storage, thus eliminating the need for 256 pages table entries (256 * 8 = 2K)
- **This feature is sometimes referred to as “large page.”**

Enhanced DAT: No Page Table Required



The “Invalid” Bit

- **Address translation is prevented when the I (invalid) bit is set in a region-, segment- or page table entry**
- **Access to such an address results in a page-, segment-, or region-translation exception (program interrupt)**
- **The instruction or, in case of an interruptible instruction, the unit of operation is nullified, i.e., the program old PSW points *to* the instruction, not after it**

Multiprocessing

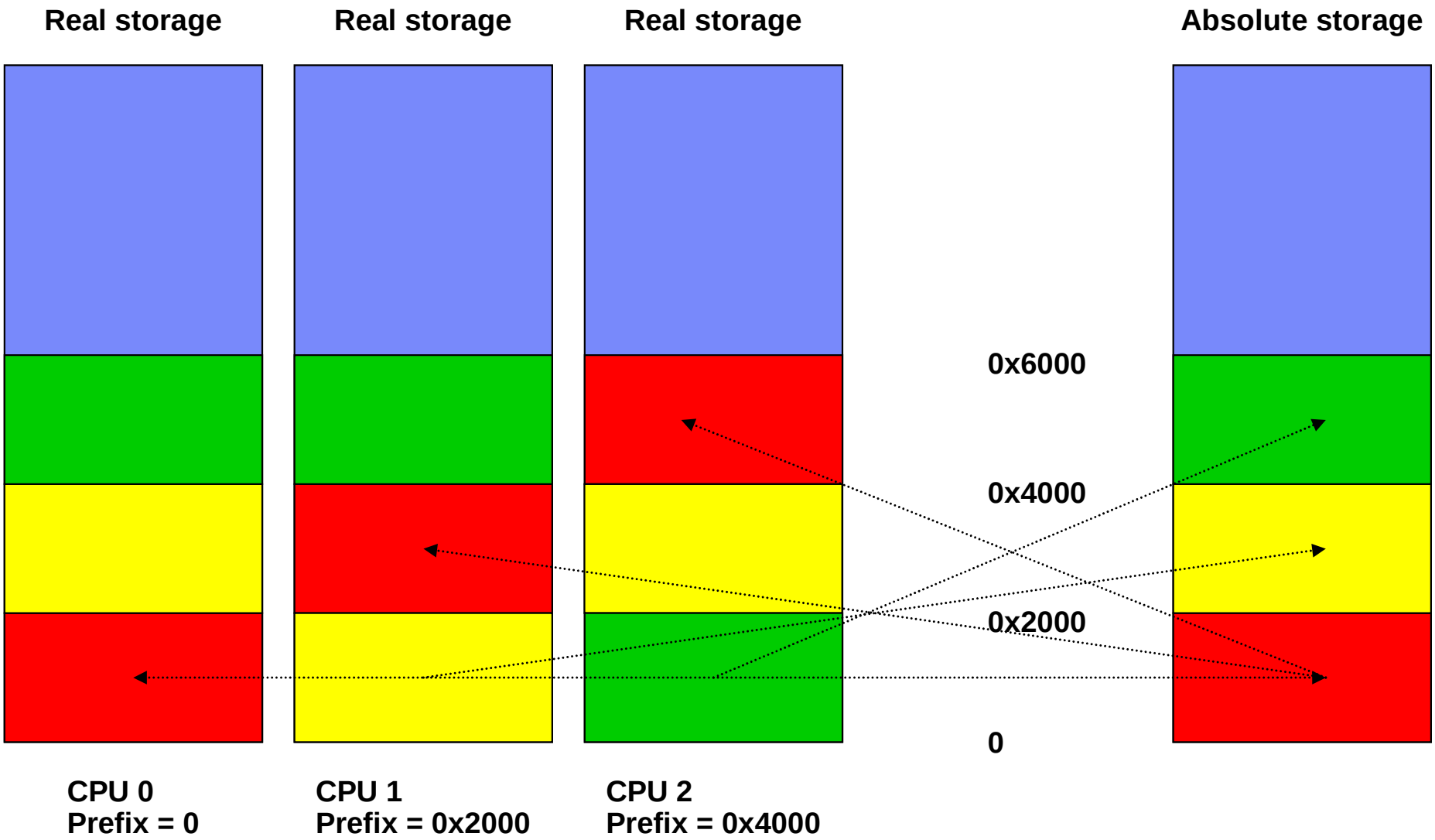
Multiprocessing

- **Largest z196 has 80 CPUs (model M80)**
- **Largest z10 EC has 64 CPUs (model E64)**
- **Shared main storage across all books**
- **Prefix area unique for each CPU**
- **Shared data must be updated interlocked by special instructions:**
  - TEST AND SET (antique)
  - COMPARE AND SWAP [AND STORE]
  - PERFORM LOCKED OPERATION
  - New with z196: LOAD AND ADD [LOGICAL] / AND / OR / EXCLUSIVE OR
- **CPUs communicate via SIGNAL PROCESSOR instruction (SIGP) and external interrupts**

Assigned Storage Locations and Prefixing

- **Other names: fixed storage locations, low core, prefix area**
- **Assigned storage locations are used to exchange information between system and software, e. g. to handle interrupts**
- **They are located in the address range 0 – 0x1FFF (real)**
- **Each CPU has to manage its own information, so for each CPU this address range must be mapped to a different place in absolute storage**
- **The prefix register of each CPU specifies this absolute address**
- **The operating system has to ensure that each CPU has a different prefix register**

Mapping of Real to Absolute Storage (Prefixing)



Example: Prefix Register = 0x4000

- **If the prefix register contains 0x4000, then**
  - Real addresses 0 – 0x1FFF translate to 0x4000 – 0x5FFF absolute
  - Real addresses 0x4000 – 0x5FFF translate to 0 – 0x1FFF absolute
  - In all other cases, real and absolute addresses are identical
- **Essentially, the ranges 0 – 0x1FFF and the range of the prefix register are swapped**
- **Each CPU has a separate area addressable as 0 – 0x1FFF**
- **If the prefix register is 0, prefixing has no effect**
- **The prefix register must specify an address < 2G**
- **In ESA/390 architecture, the prefix register contains an address on a 4K boundary and applies only to 4K (0 – 0xFFF)**
- **For a detailed mapping, check the Principles of Operation, “Assigned Storage Locations” in chapter 3**

Compare and Swap

- **Purpose**
  - Safe update of data in storage shared between several routines running on different processors
  - May also be needed on uniprocessor when routines run in different threads/tasks
- **Concept**
  - Fetch the original value from storage
  - Create a new value, based on the original value
  - Compare the private copy of the original value to the current value in storage – if it has not changed since the original fetch, replace it with the new value
  - However, if the value in storage has changed in between, retry the process
  - So, in **most** cases, Compare and Swap is done in a loop

Compare and Swap (continued…)

Example 1: Increment a counter

```
         L       R0,COUNT          fetch original value
LOOP     LR      R1,R0             create copy
         AHI     R1,1              increment copy
         CS      R0,R1,COUNT       if R0 = COUNT then
*                                      save R1 in COUNT
         BRNZ    LOOP              else
*                                      load COUNT into R0
         ...
COUNT    DS      F                 full word
```

Compare and Swap (continued…)

Example 2: Obtain a lock (non-preferred solution)

```
        LHI     R1,-1           create lock (0xFFFFFFFF)
LOOP    LHI     R0,0            create expected lock (0)
        CS      R0,R1,LOCK      if LOCK = 0, store R1 into LOCK
        BRNZ    LOOP            didn't get the lock, try again
*
DONE    ...                     proceed
        ...
LOCK    DS      F               full word
```

- Disadvantage: CS locks the cache line exclusively upfront
- When the lock is held by another CPU, the cache line will bounce back and forth between the CPU owning the lock and the CPU requesting the lock
- Solution: If it is likely that CS will not succeed (cc 1), do a trial fetch first

Compare and Swap (continued…)

Example 2: Obtain a lock (preferred solution)

```
        LHI     R1,-1           create lock (0xFFFFFFFF)
        LHI     R0,0            create expected lock (0)
LOOP    CS      R0,R1,LOCK      if LOCK = 0, store R1 into LOCK
        BRZ     DONE            we got the lock
*                               else, LOCK is loaded into R0
TEST    LT      R0,LOCK         try again (simple fetch)
        BRNZ    TEST            still locked
        BRU     LOOP            no longer locked, try CS again
*
DONE    ...                     proceed
        ...
LOCK    DS      F               full word
```

- Note: it may be desirable to place lock and data in different cache lines

Variations of Compare and Swap

Mnemonic	Operand length	Notes
CS	Word (32 bits)	
CSY	Word	Long displacement
CSG	Double word (64 bits)	Single register
CDS	Double word	Lower halves of even/odd register pair
CDSY	Double word	Long displacement
CDSG	Quad word (128 bits)	Even/odd register pair

Note: Operand must be aligned according to its length (word, double word, quad word boundary)

More Atomic Instructions

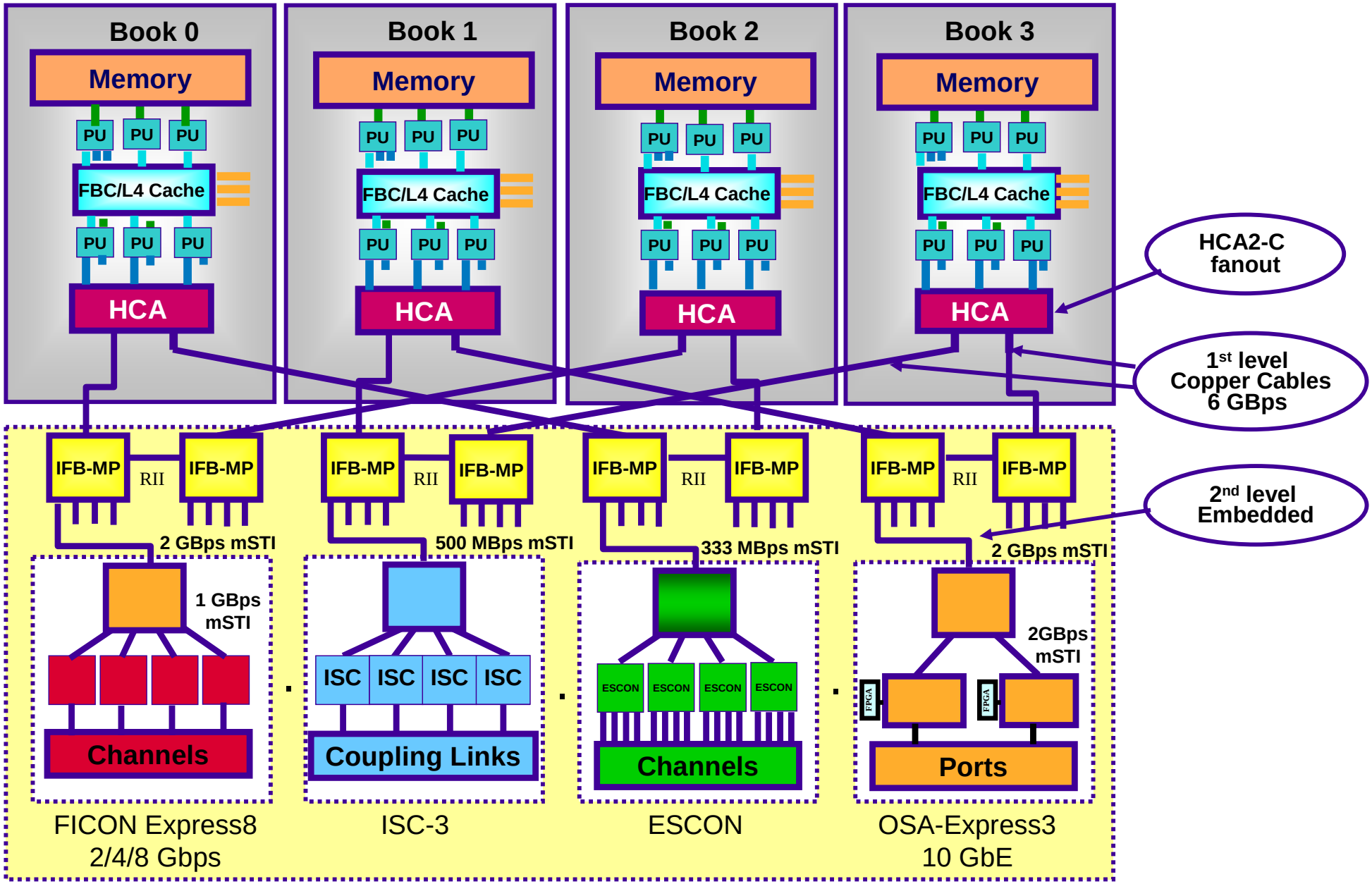
- **COMPARE AND SWAP AND STORE (CSST)**
  - Does a separate store after COMPARE AND SWAP function within one instruction
  - For details, read the Principles of Operation, chapter 7
- **PERFORM LOCKED OPERATION (PLO)**
  - For complex locking protocols (up to 8 operands)
  - Locks only against other PLO locks
  - For details, read the Principles of Operation, chapter 7
- **New with z196:**
  LOAD AND ADD [LOGICAL], LOAD AND AND / OR / EXCLUSIVE OR
  - Performs atomic update on a word or doubleword in a single instruction, without the use of a Compare and Swap loop
  - Similar to GCC __sync_fetch_and_xxx() built-in functions
  - For details, read the Principles of Operation, chapter 7

Input/Output

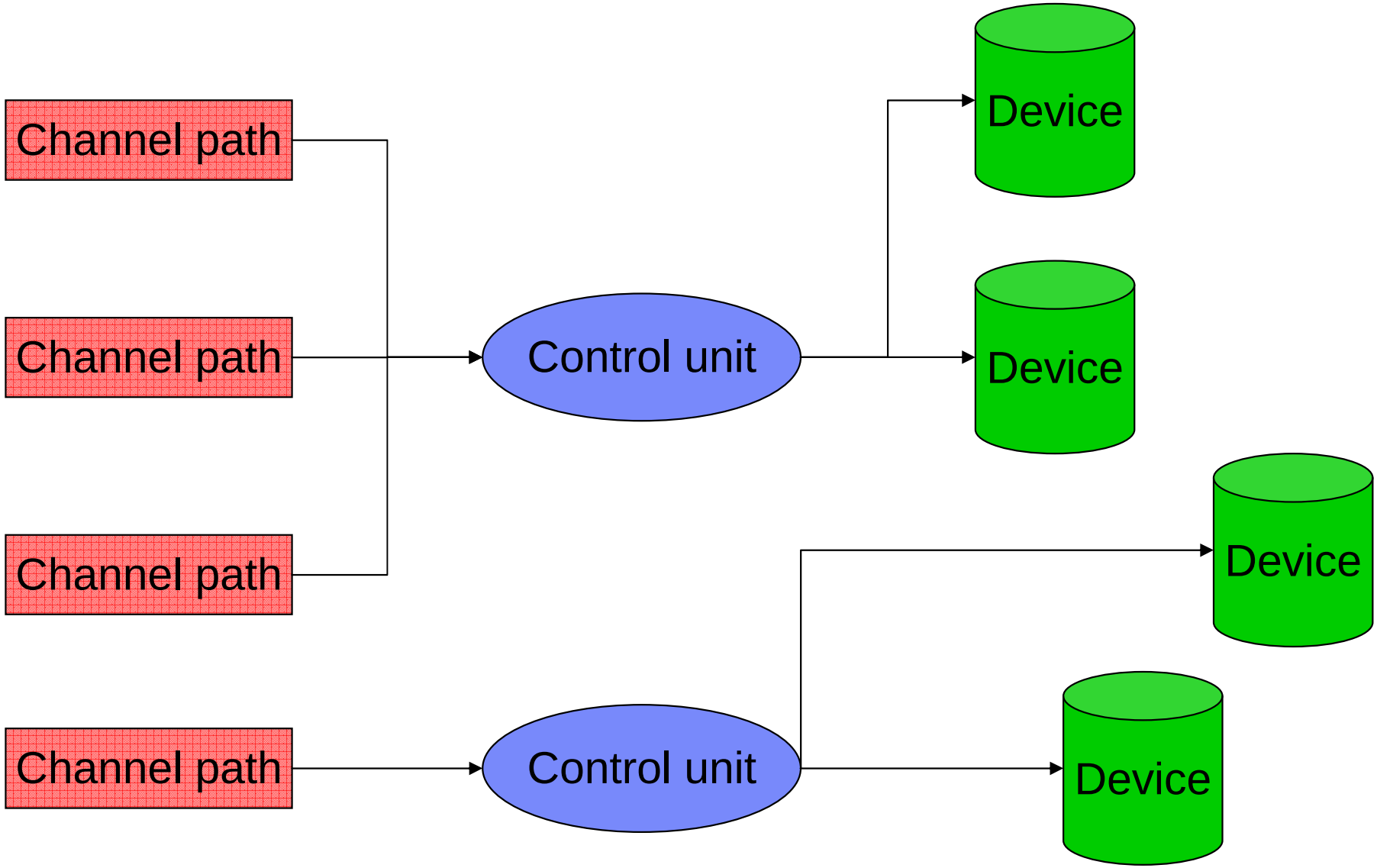
The I/O Subsystem (“Channel Subsystem”)

- **Directs the flow of information between I/O devices and main storage**
- **Relieves the CPU of the task of communicating directly with I/O devices**
- **Allows CPU processing to proceed concurrently with I/O operations**
- **In general, runs on the SAPs**
- **System z does not use memory-mapped I/O**

z196 I/O Infrastructure



Input/Output



Input/Output

- **One control unit may serve multiple devices**
- **Up to 8 channel paths may be attached to one control unit**
  - Better performance
  - Improved reliability, availability, serviceability

Channel Paths, Control Units, and Devices

- **A channel path is a separate processor that controls data transfer between main storage and devices**
  - Data to be read from/written to an external medium
  - Control information
- **A control unit understands commands in detail (e.g. disk head positioning)**
- **A device is driven by a control unit**
- **A device is assigned a device number (0 – 0xFFFF) by the system administrator**
  - used by the operating system when displaying information or processing commands
- **A device is assigned a “subchannel” number (0 – 0xFFFF) by the system when it is configured**
  - used in I/O instructions

I/O Configuration

- **The initial I/O configuration is defined in an I/O configuration data set (IOCDS)**
- **The I/O configuration may be changed under software control by adding or deleting any components**
  - Channel paths
  - Control units
  - Devices
- **The software front-end to change the IOCDS dynamically is known as HCD (Hardware Configuration Definition)**
- **There is also a graphical front-end HCM (Hardware Configuration Management)**

Maximum I/O Configuration (z10)

- **Addressability:**
  - Up to 1,024 channel paths
  - Up to 261,120 devices

- **Pluggable:**
  - Up to 84 I/O cards in up to 3 I/O cages
  - Each card has 1, 2, 4, or 15 (+ 1 spare) channel paths
  - This leads to a total of up to 1,260 channel paths

Flow of an I/O Operation

CPU	SAP	Channel
Software issues SSCH		
Passes subchannel to SAP		
Returns cc 0 to software (operating system)	Selects channel path (1 out of up to 8)	
	Passes subchannel to channel	
		Executes channel program
		Signals completion to SAP
	Processes channel's response	
	Offers I/O interrupt to all CPUs	
One CPU in the configuration takes the I/O interrupt		

The Channel Program

- **The START SUBCHANNEL instruction specifies**
  - The device where the I/O operation is to be performed
  - The command sequence to be performed (the channel program)
- **The channel program consists of one or more channel command words (CCWs)**
- **The channel program resides in absolute storage**
  - The channel does not use dynamic address translation
  - The channel does not use a prefix register

The Channel Program

- **Each CCW specifies**
  - A command code
    - Read
    - Write
    - Control
    - Sense
  - A data address in absolute storage
  - A count field
  - Several flag bits
    - E.g. command chaining

How are virtual addresses managed?

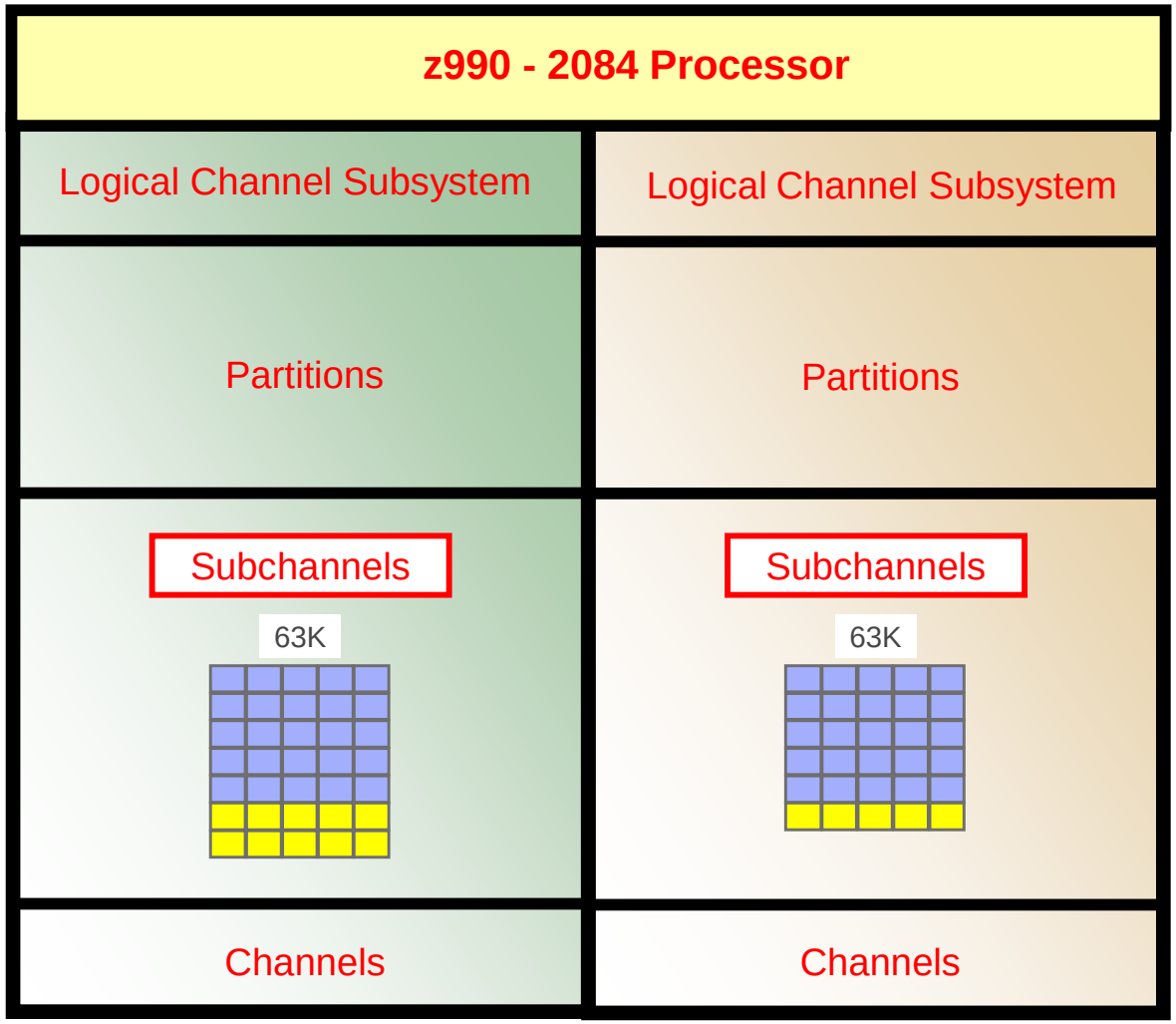
- **The operating system pins all pages that contain data buffers for the duration of the I/O operation**

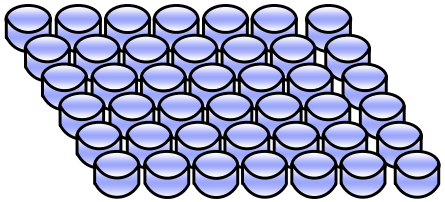
- **The operating system translates all virtual buffer addresses to absolute ones**
  - Indirect data address lists are used in a channel program to describe virtual buffers scattered in absolute storage

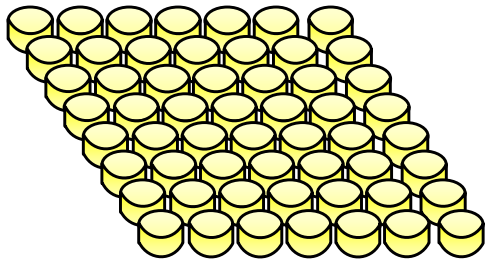
Multiple channel subsystems (MCSS)

- **1 to 4 logical channel subsystems (LCSS)**
  - Up to 256 channel paths per logical partition (LPAR)
  - Up to 63-64K devices (subchannels) per LPAR
  - An LCSS may be shared by multiple LPARs
- **MCSS allows much larger configurations (e.g. to consolidate multiple older systems on a new one)**

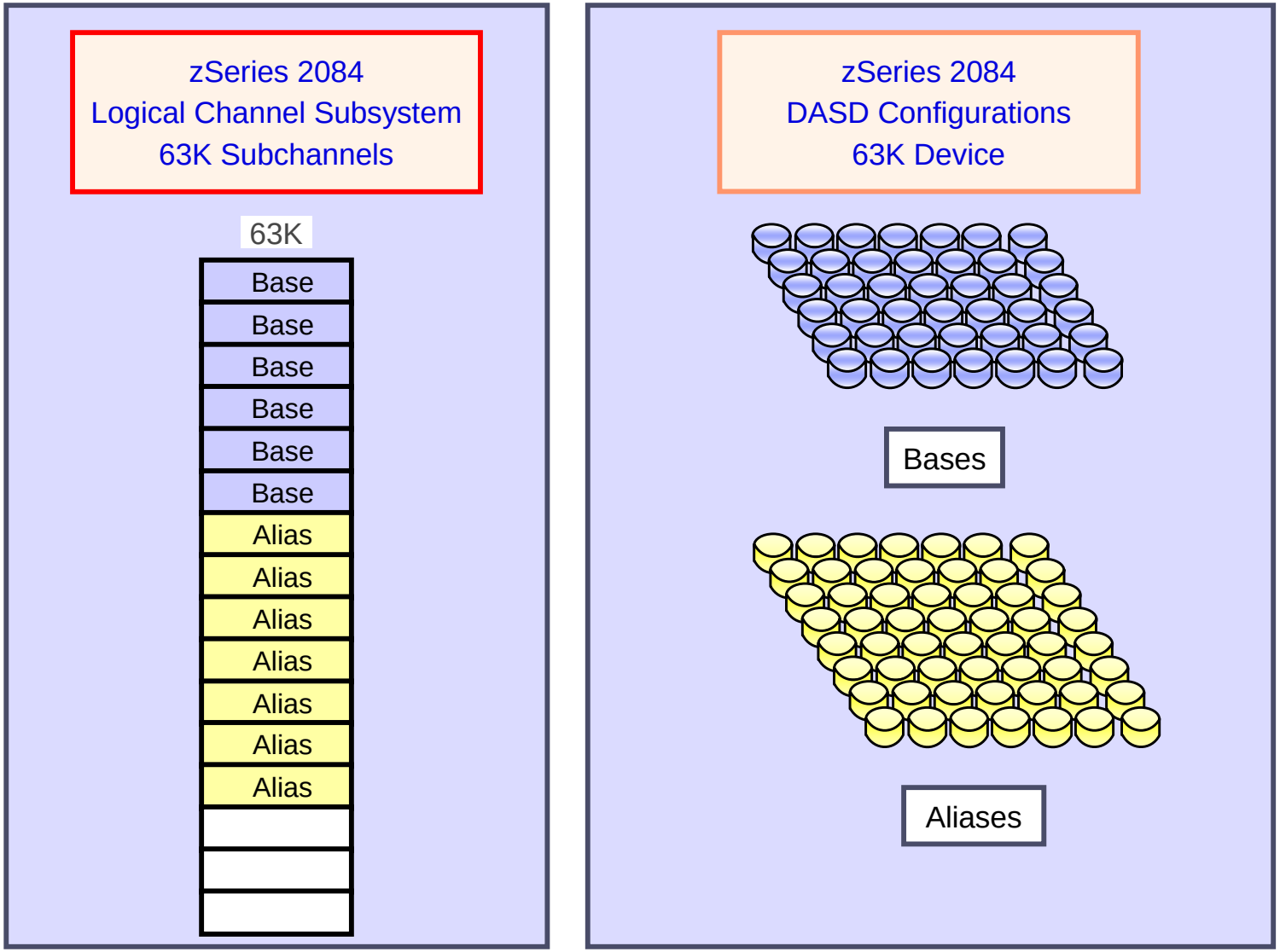
MCSS – I/O Configuration Support







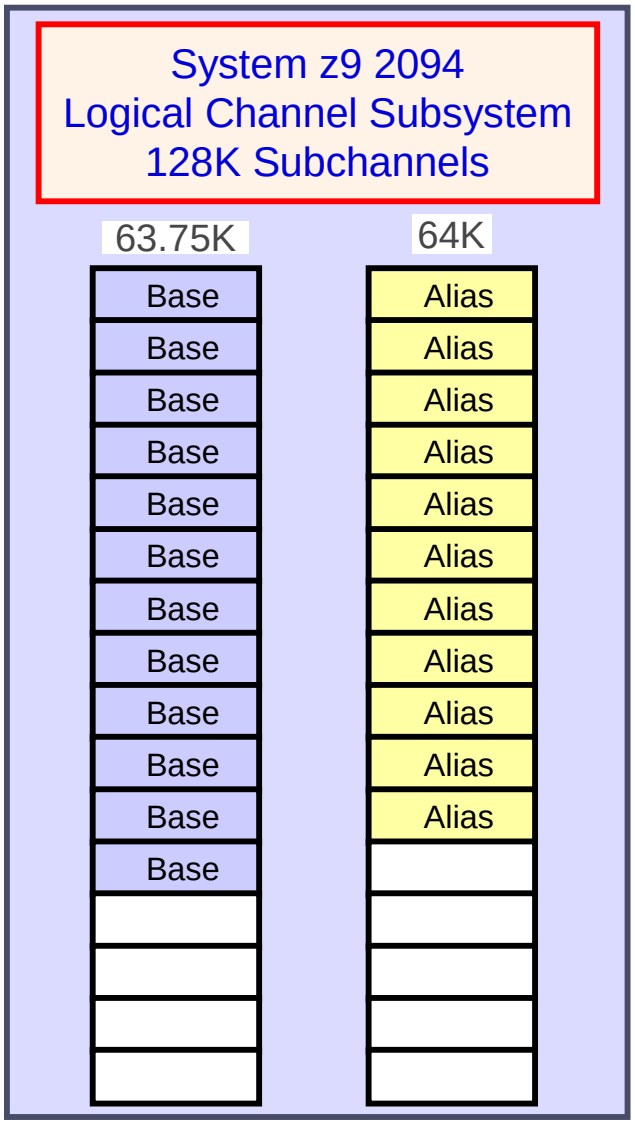
MCSS – I/O Configuration Support

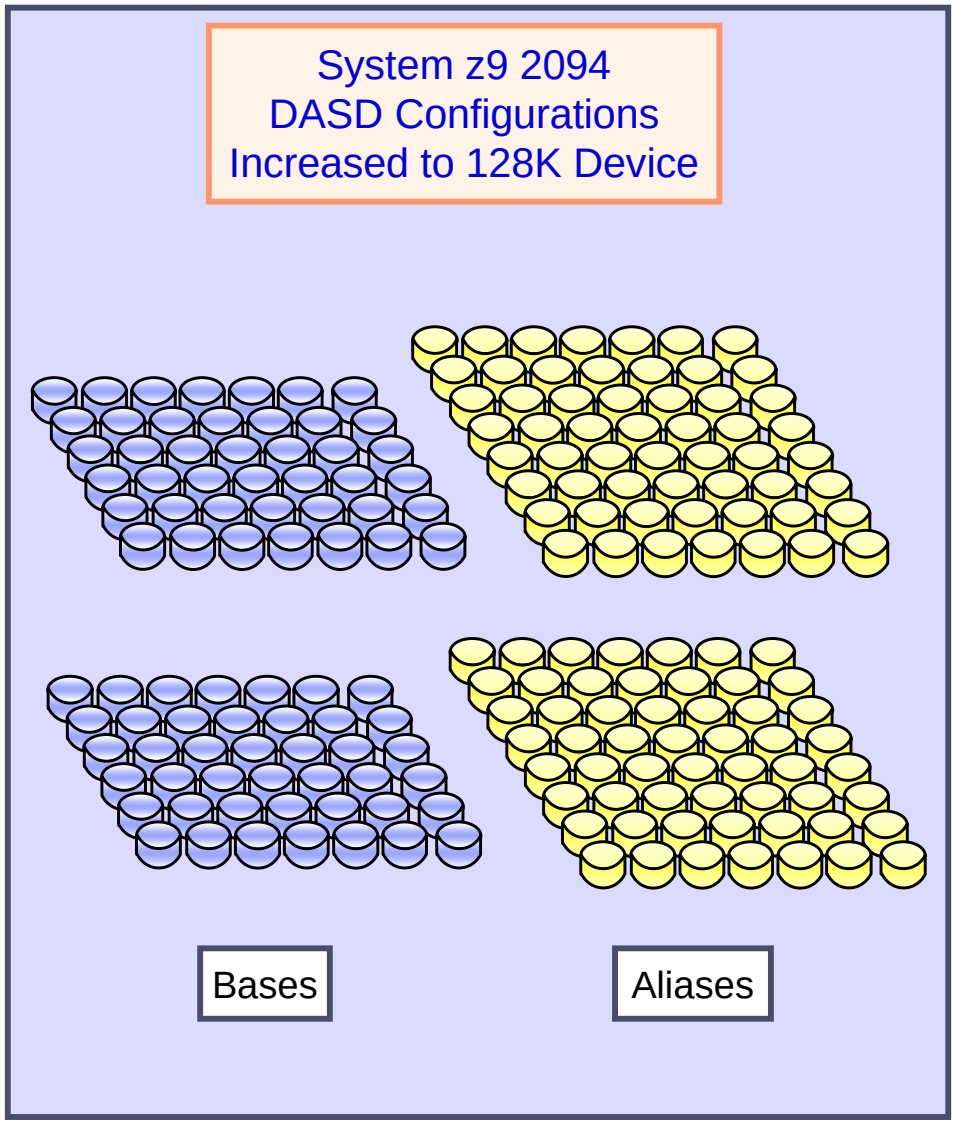


Multiple subchannel sets (MSS)

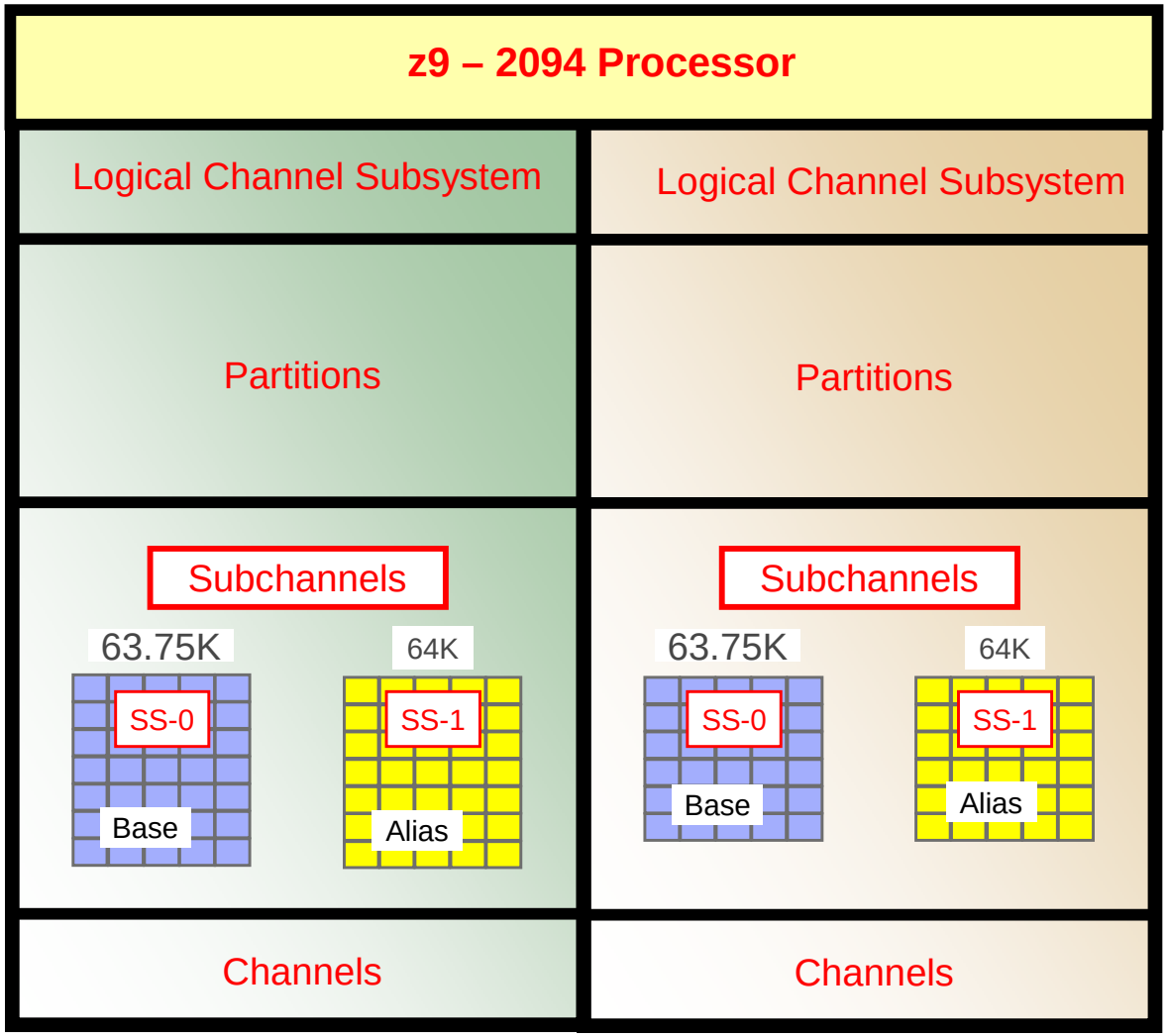
- **The problem: Originally, System z had only 64,512 (=63K) subchannels (1,024 out of 65,536 were reserved for internal use)**
- **This became a problem for large installations because of PAV (parallel access volumes):**
  - With PAV, a single disk drive often consumes four subchannels (base address plus three aliases)
- **The solution: multiple subchannel sets**
  - 65,280 subchannels in set 0 (256 reserved for internal use)
  - 65,535 subchannels in set 1
- **z/OS 1.7 exploits second subchannel set to access *alias addresses* of parallel access volumes (PAV)**

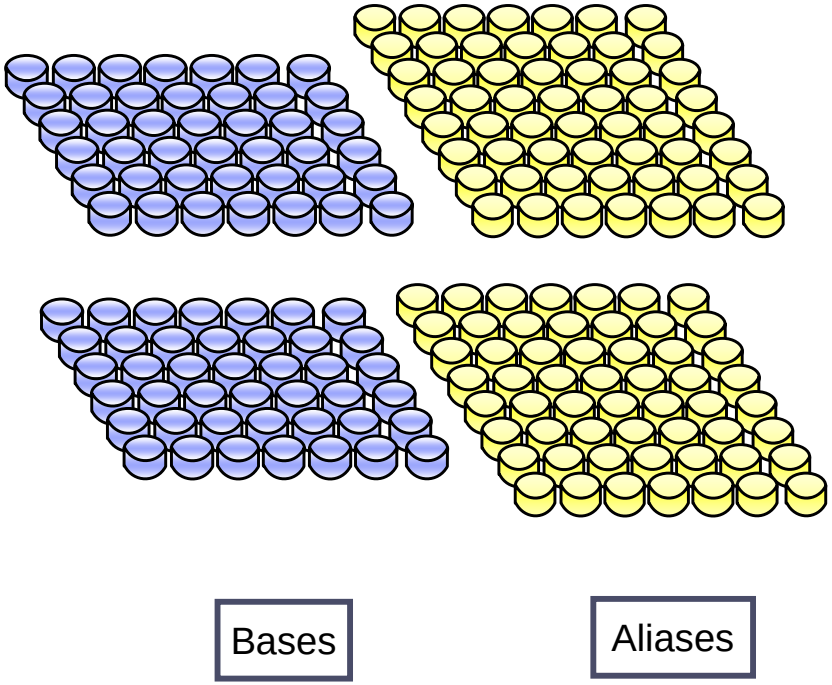
I/O Configuration Growth - Multiple Subchannel Sets (MSS)





Multiple Subchannel Sets per LCSS





Partitioning and Virtualization

Partitioning and Virtualization

- **Purpose**
  - Run more than one operating system on a system
- **Method**
  - Make each operating system *think* it owns a whole machine
- **Comes in two flavors**
  - LPAR (logical partitioning)
  - z/VM (virtual machine)
- **Both flavors can be combined**
  - z/VM runs under LPAR (two-level virtualization)

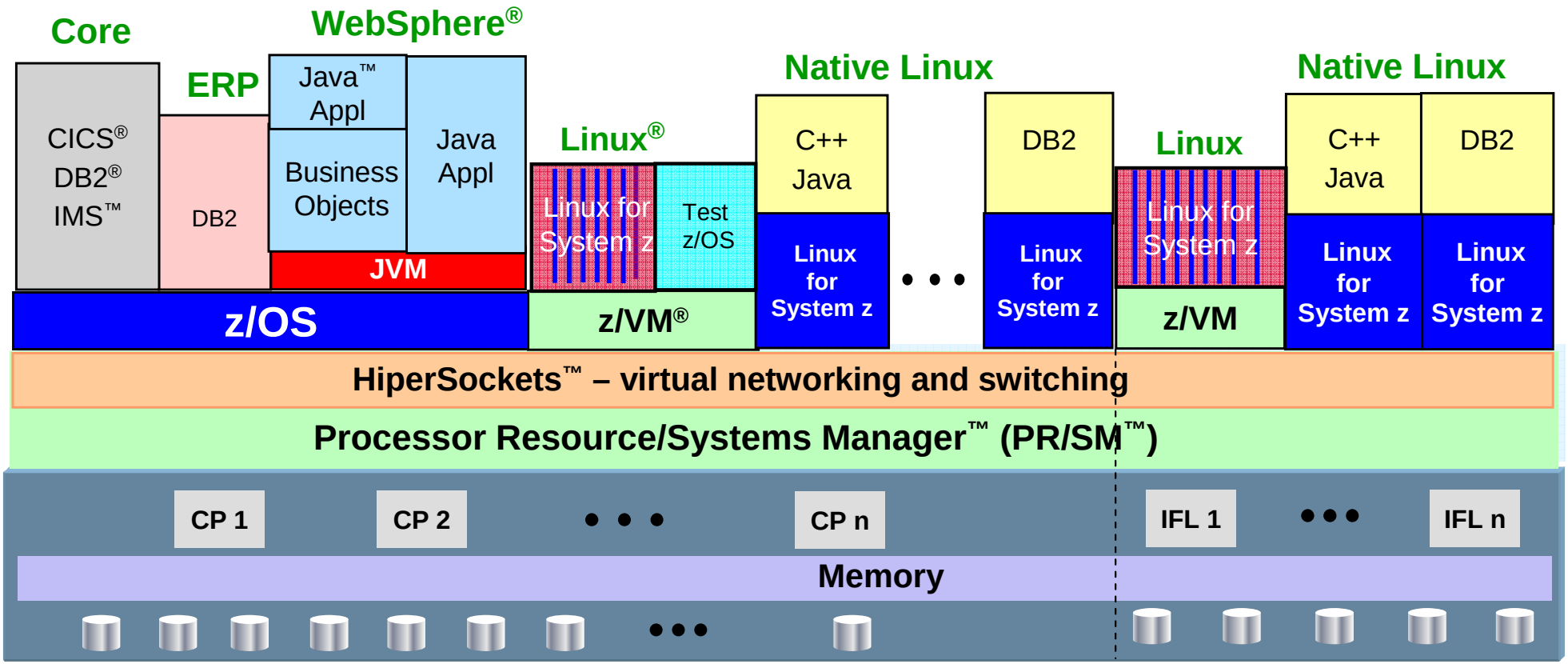
LPAR

- **A control program (LPAR hypervisor) manages logical partitions:**
  - Each partition owns a defined amount of physical storage
  - Strictly no storage shared across partitions
  - No virtual storage / paging done by LPAR hypervisor
  - Zone relocation lets each partition start at address 0
  - CPUs may be dedicated to a partition or may be shared by multiple partitions
  - I/O channels may be dedicated to a partition or may be shared by multiple partitions (Multiple image facility, MIF)
  - Each LPAR has its own architecture mode (ESA/390 or z/Architecture)
- **LPAR hypervisor is shipped with System z (considered part of firmware)**
- **Since z990 (2003), the LPAR hypervisor is always loaded (no Basic Mode anymore)**
- **Separation of logical partitions is considered as good as having each partition on a separate system (Evaluation Assurance Level 5)**
- **LPAR is also referred to as PR/SM (Processor Resource/System Manager)**

z/VM

- **A control program (CP) creates a virtual machine for each user**
  - Each virtual machine has an own address space starting at 0
  - This is virtual storage managed by CP (subject to paging)
  - Each virtual machine has its own architecture mode (ESA/390 or z/Architecture)
  - CP simulates one or multiple (virtual) CPUs per virtual machine
  - CP simulates an I/O configuration for each virtual machine
    - Dedicated devices, e. g., console
    - Shared devices, e. g. minidisks (disks that are partitioned for many virtual machines), printers (spooling devices)
  - CP simulates communication paths between virtual machines
    - channel-to-channel adapter
    - Inter-user communication vehicle (IUCV)
    - Virtual LANs

System z – The Ultimate Virtualization Resource



- Massive, robust consolidation platform; virtualization is built in, not added on
- Up to 60 logical partitions on PR/SM; 100's to 1000's of virtual servers on z/VM
- Virtual networking for memory-speed communication, as well as virtual layer 2 and layer 3 networks supported by z/VM
- Most sophisticated and complete hypervisor function available
- Intelligent and autonomic management of diverse workloads and system resources based on business policies and workload performance objectives

Interpretive Execution (SIE)

- **Both LPAR and z/VM use the instruction START INTERPRETIVE EXECUTION (SIE) to run a logical partition or virtual machine, respectively**
- **The program issuing SIE is called host**
- **The program running under SIE is called guest**
- **The operand of the SIE instruction is a state description, it describes the guest**
- **One level of nesting (SIE under SIE) is supported, used when z/VM runs in a logical partition**

History of SIE

- **The SIE instruction was introduced with the 370/XA architecture (early 1980s)**
- **Invented based on the experiences with VM/370 on S/370 systems**
- **Not documented in the Principles of Operation**
- **Partially documented in SA22-7095 (public, 1985)**
- **Updated for z/Architecture**

SIE State Description

- **Guest PSW**
- **Guest CPU timer, clock comparator**
- **Guest epoch difference (for guest TOD clock)**
- **Guest control registers**
- **Guest general registers 14 and 15**
- **Guest prefix register**
- **... and various other fields**

Host Program Responsibilities

- **On SIE entry**
  - Load guest general registers 0 – 13
  - Load guest floating-point registers, floating-point-control register
  - Load guest access registers
- **On SIE exit**
  - Handle interception

Programmable SIE Exit Conditions

- **Interception controls**
  - Certain instructions
  - SVCs by SVC number
  - LCTL for any set of control registers
- **PSW enabled for I/O interrupts (PSW.6 = 1)**
- **PSW enabled for external interrupts (PSW.7 = 1)**
- **Stop request (triggered from another CPU)**

SIE Exit Conditions

- **An interception**
  - The state descriptor is updated, an interception code is stored and the host program resumes after the SIE instruction.

    Example: SSCH instruction

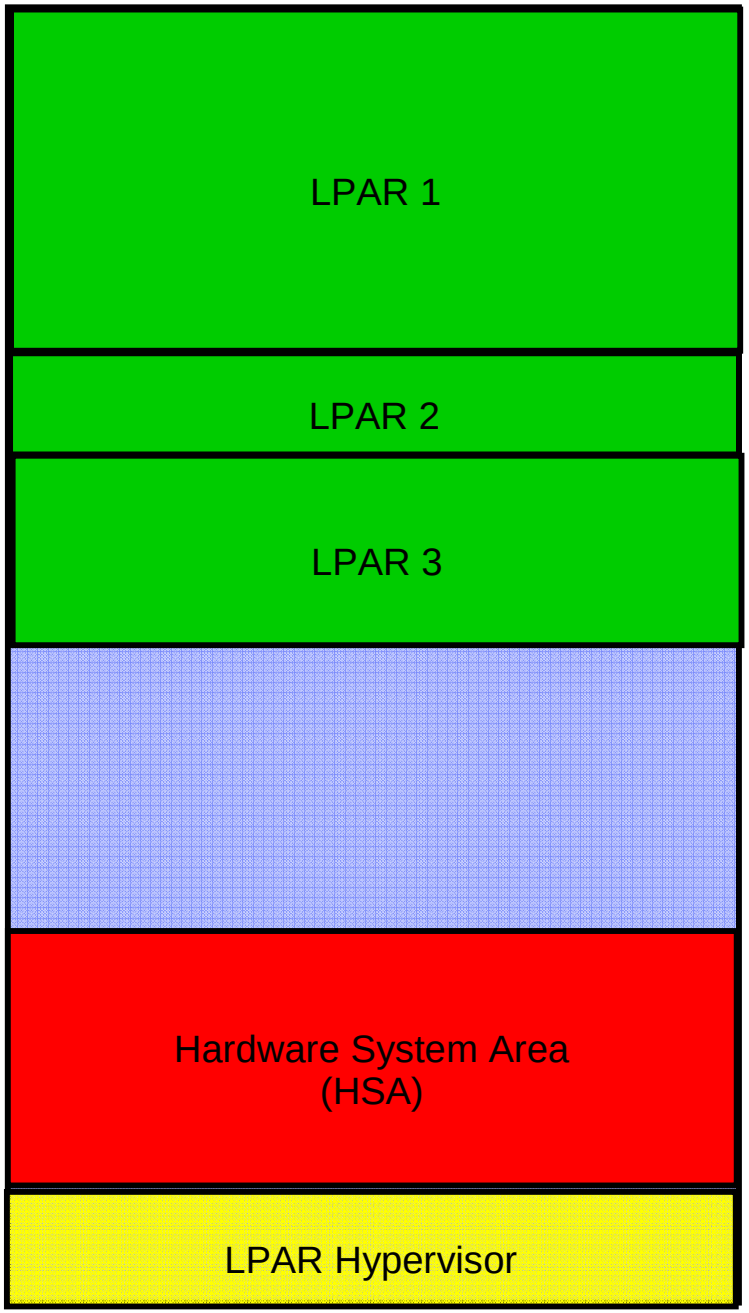
- **A host interrupt**
  - e.g. an external or I/O interrupt, or a translation exception. In this case, the unit of operation is nullified, the old PSW points to the SIE instruction. No interception code is stored.

Storage of a Logical Partition

- **The storage assigned to a logical partition is called a zone**
  - The zone origin is the host absolute address where the zone starts (zone address 0)
  - The zone limit is the host absolute address where the zone ends
  - Origin / limit pairs are associated with a zone number (LPAR number)

Storage Layout in LPAR Mode

- **Absolute storage of an LPAR must be contiguous**
- **Activating and deactivating LPARs may lead to fragmentation of storage**
- **How can we assign all unused storage to a single LPAR?**



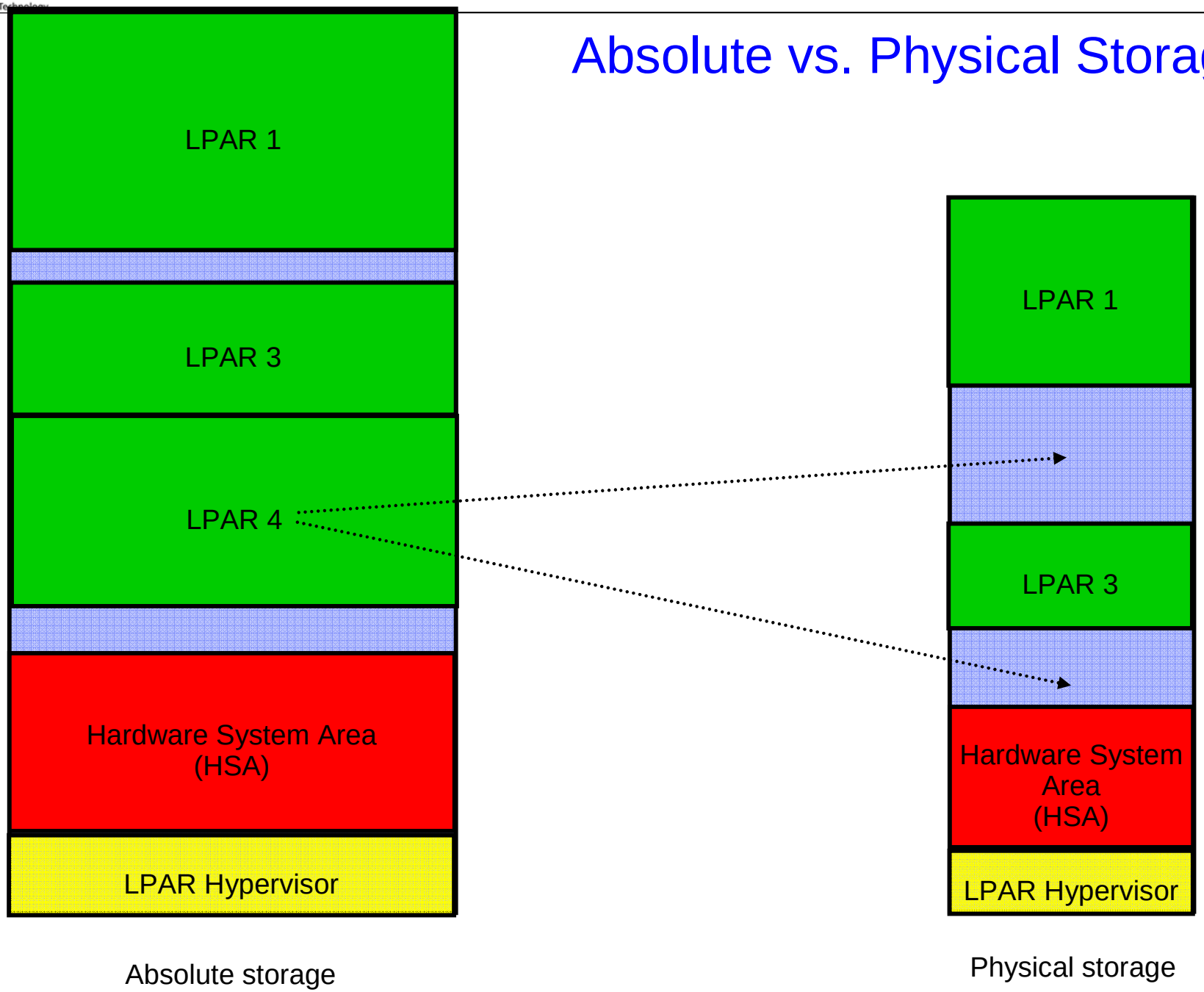
Storage Layout in LPAR Mode (continued…)

- **Another mapping of storage is introduced:**

  Absolute storage is mapped to physical storage
  - Absolute storage size is higher than the maximum physical storage size (e.g. 2x)
  - Mapping allows LPAR portions scattered in physical storage to appear contiguous in absolute storage

- **With pre-planning, it is possible to extend the storage size of an LPAR on the fly**

Absolute vs. Physical Storage



Storage of a Virtual Machine under z/VM

- **The primary address space created by z/VM describes the virtual machine's absolute storage (host CR 1).**
  - This storage is subject to paging by z/VM
- **Note: Since the virtual machine may use its own virtual storage, two DAT translations are required:**
  - Guest-2 virtual to guest-2 absolute
  - Guest-1 virtual (= guest-2 absolute) to guest-1 absolute
- **This is all done in hardware (no shadow tables required)**

The Complete Address Translation Process (1)

- **Scenario**
  - Application running in an operating system
  - Operating system (guest-2) running under z/VM
  - z/VM (guest-1) running in an LPAR
  - LPAR managed by the LPAR hypervisor (host)

The Complete Address Translation Process (2)

- **Application uses virtual addresses (guest-2 virtual)**
- **Application address is translated**
  - To guest-2 real using the operating system's DAT tables, then
  - To guest-2 absolute using the operating system's prefix register

The Complete Address Translation Process (3)

- **The guest-2 absolute address is taken to be guest-1 virtual address by z/VM. It is translated**
  - To guest-1 real using z/VM's DAT tables, then
  - To guest-1 absolute using z/VM's prefix register
- **The guest-1 absolute address (absolute address within the LPAR) is now translated**
  - To host absolute by adding the LPAR's zone origin (It is also checked against LPAR's zone limit)

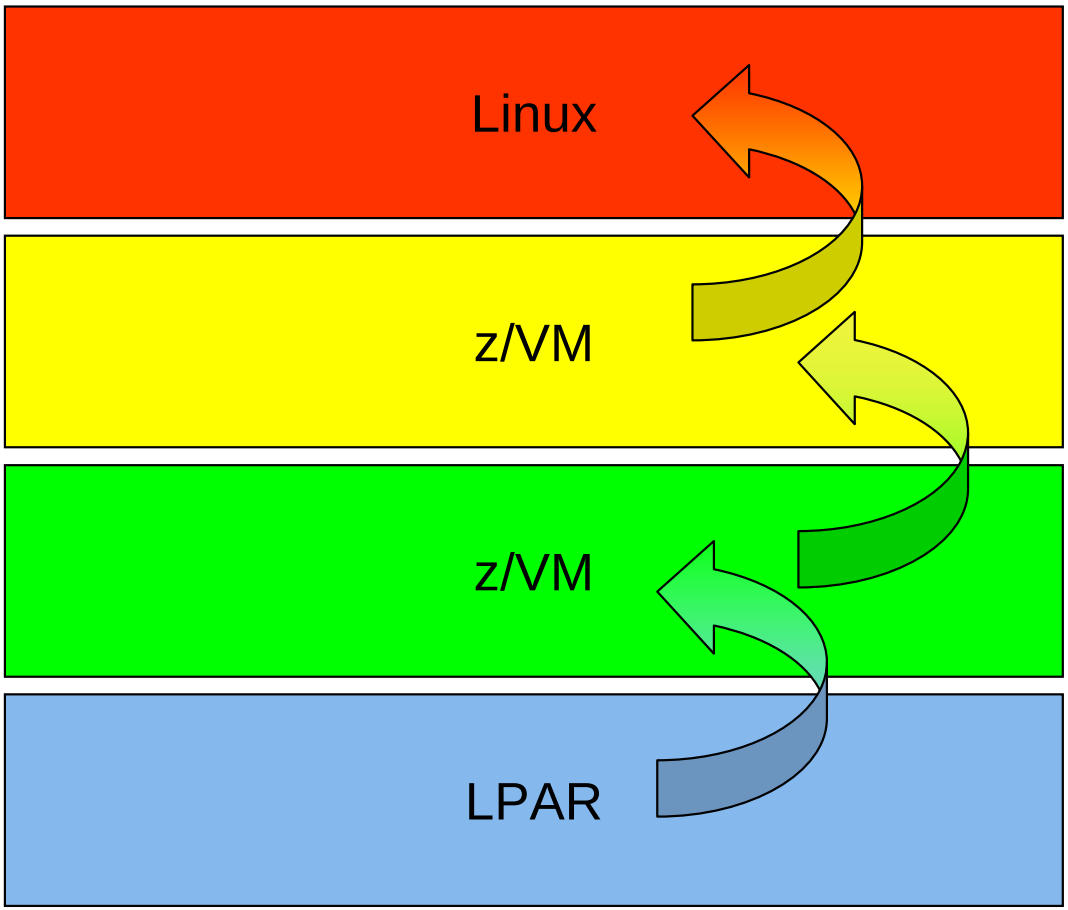
The Complete Address Translation Process (4)

- **Finally, the host absolute address is translated**
  - To a physical address using the configuration array

z/VM under z/VM under z/VM ...

- **z/VM supports interpretation of SIE to allow z/VM under z/VM**
- **Theoretically, an arbitrary nesting level is possible**
- **Typical scenario is testing of a new z/VM version on an old z/VM version**

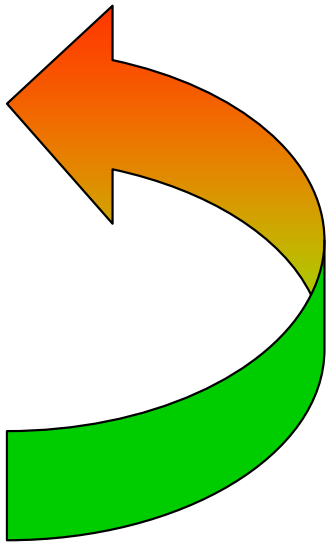
Nested Virtualization



Guest-2→ Guest-3

Guest-1→ Guest-2

Host → Guest-1

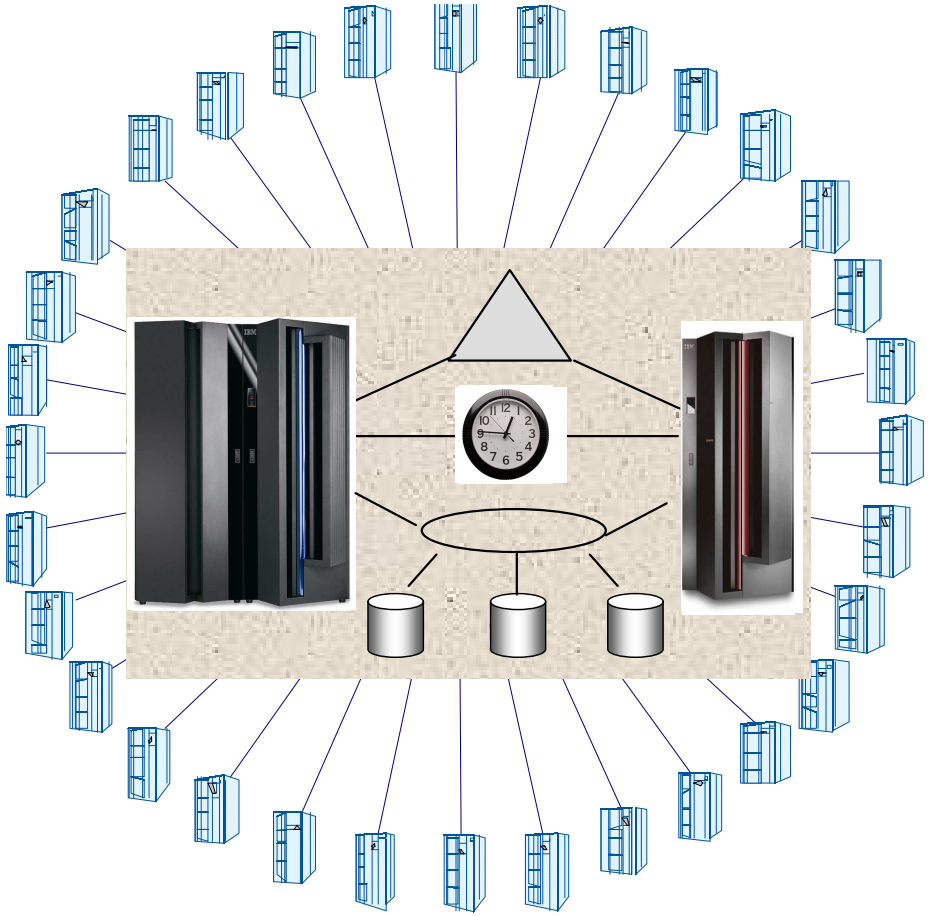


Guest-1→ Guest-3

Parallel Sysplex

Parallel Sysplex

- **Several separate systems are connected to a coupling facility (CF) via high-speed fiber links (coupling channels)**
- **This configuration forms a “Parallel Sysplex” with a star topology**
- **The coupling facility is the hub of this star**
- **In a Parallel Sysplex, each system has access to the complete disk pool (shared-data model)**
- **Parallel Sysplex is supported by z/OS**
- **Timing is synchronized with a Sysplex Timer or (new) with a Server Time Protocol (STP)**



2 – 32 systems

Rationale for Parallel Sysplex

- **Processing capacity beyond single SMP**
- **Non-disruptive addition of processing capacity without changes to customer applications**
- **Improved application availability, reduction of planned outages**
- **Incremental growth – up to 32 systems (5 – 6 typical)**
- **Operability / Manageability (single system image)**
- **Work load management provides load balancing across systems**

Coupling Facility Characteristics

- **Runs in an LPAR**
  - In a separate system or side-by-side with z/OS LPARs
- **Uses InterSystem Channels (ISCs) and integrated cluster buses (ICBs) to communicate with systems**
- **Does not use ESCON / FICON I/O**
- **Uses console integration of support element (SE) to communicate with system operator**
- **Runs Coupling Facility Control Code (CFCC), a control program that talks to the systems via ISCs and ICBs**
- **CFCC is firmware that is shipped as part of the system**
- **CFCC can be run in a virtual machine on z/VM**

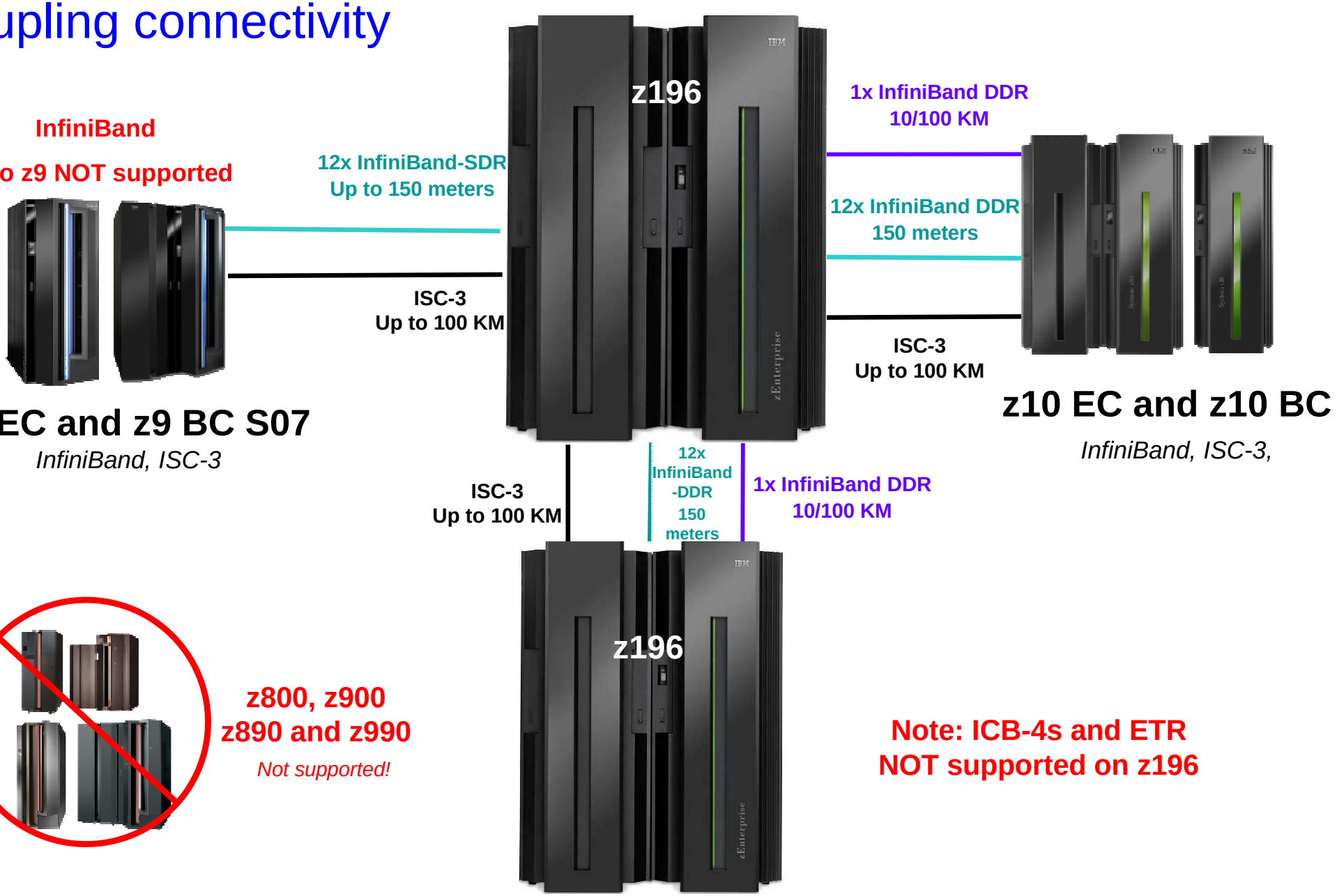
Coupling Facility Characteristics (continued…)

- **The CF architecture provides three behavioral models to enable efficient clustering models**
  - **Lock model:** enables a specialized lock manager (e.g. database lock manager) to be extended into a multi-system environment
  - **Cache model:** provides the functions needed for multi-system shared-data cache coherency management
  - **Queue/List model:** provides a rich set of queuing constructs in support of workload distribution, message pathing, and sharing of state information
- **Multiple CFs can be connected for availability, performance, and capacity reasons**

Coupling Channels

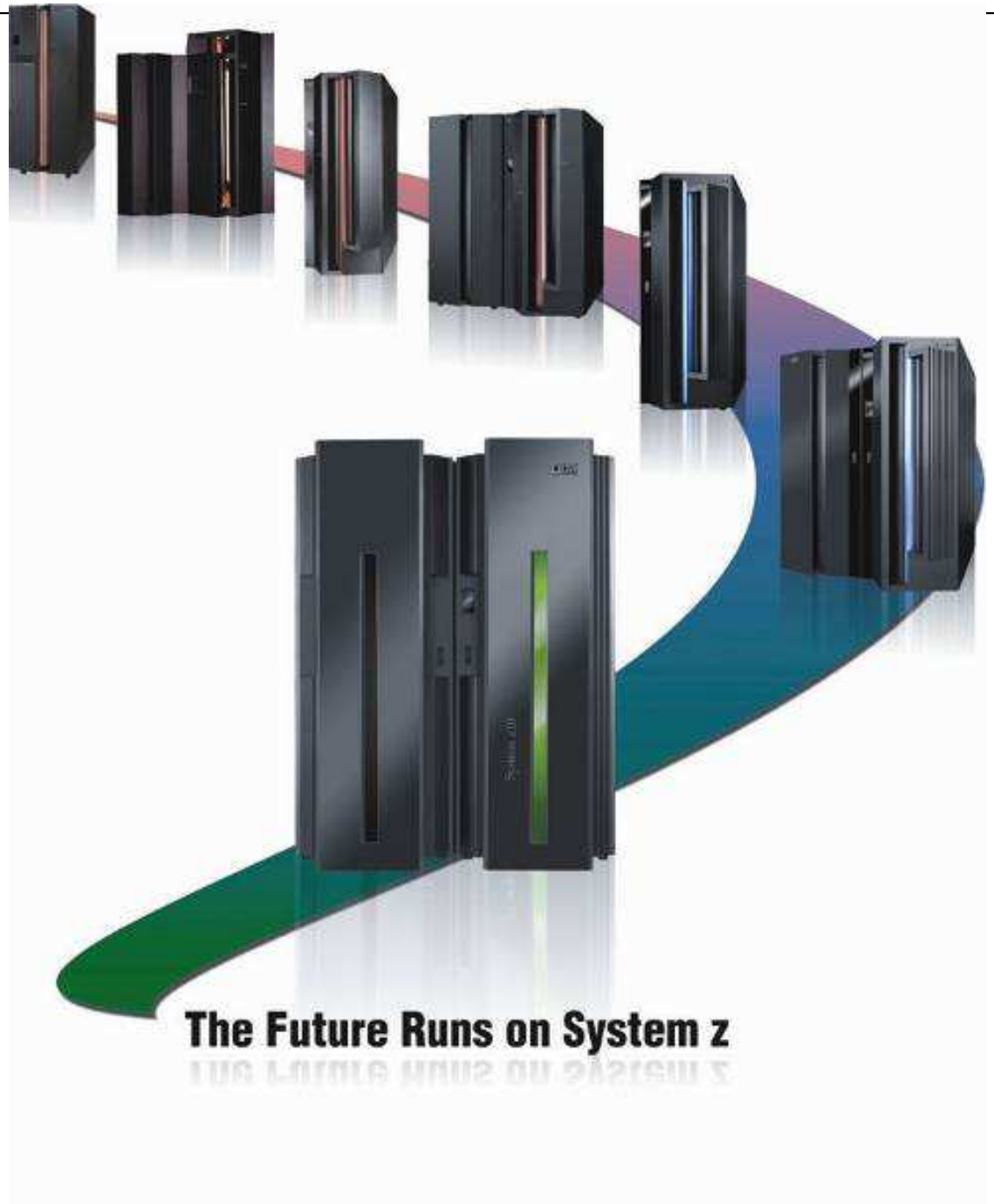
- **InterSystem Channels (ISCs)**
  - With repeaters, can span distances of up to 100 km
  - Good for distributed environments (e.g. Geographically Dispersed Parallel Sysplex, GDPS)
  - Without repeaters, 10 km distance and 2 Gigabits/sec
- **Integrated Cluster Buses (ICBs)**
  - Short distance: about 10 m
  - High performance
    - ICB-3: 1 Gigabyte/sec
    - ICB-4: 2 Gigabytes/sec
- **Parallel Sysplex over InfiniBand (PSIFB)**
  - Distances of about 150 m
  - High performance
    - HCA1-O (z9): 3 Gigabytes/sec
    - HCA2-O (z10 and z196): 6 Gigabytes/sec
- **Internal Channels (ICs)**
  - LPAR-to-LPAR communication within a system
  - No external hardware used

z196 Parallel Sysplex coexistence of Servers/CFs and coupling connectivity



CF Duplexing

- **Running the CF and the z/OS on the same system (LPAR), a system failure affecting both images is not recoverable.**
- **To solve this problem, CF Duplexing is introduced**
  - It allows pairs of CFs to automatically synchronize two copies of CF data structures, one in each CF.
- **Each CF is placed in a different physical server.**



Appendix

References (1)

- **z/Architecture Principles of Operation, SA22-7832**
- **z/Architecture Reference Summary, SA22-7871**
- **ESA/390 Principles of Operation, SA22-7201**
- **System/370 Extended Architecture Interpretive Execution, SA22-7095**
- **Online (PDF) available here:**

  http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss

  Select country. Then click “Search for publications” and enter the publication number.

References (2)

- **IBM zEnterprise System Technical Introduction**

  http://www.redbooks.ibm.com/redpieces/abstracts/sg247832.html?Open

- **IBM zEnterprise System Technical Guide**

  http://www.redbooks.ibm.com/redpieces/abstracts/sg247833.html?Open

- **Technical Leader Library (TLLB) for IBM zEnterprise 196 (z196)**

- **Getting Started with InfiniBand on System z10 and System z9, SG24-7539**

  http://www.redbooks.ibm.com/abstracts/sg247539.html

References (3)

- IBM z/Journal February/March 2008: Basics of z/VM Virtualization, by Bill Bitner, Brian Wade

  http://www.zjournal.com/index.cfm?section=article&aid=946

- IBM Systems Journal, Vol. 30, No. 1, 1991: ESA/390 interpretive-execution architecture, foundation for VM/ESA, by D. L. Osisek, K. M. Jackson, P. H. Gum

  http://domino.research.ibm.com/tchjr/journalindex.nsf/a3807c5b4823c53f85256561006324be/cae3cf60a5eee39585256bfa00685c52?OpenDocument

- IBM Journal of Research and Development, Vol. 46, Nos. 4/5, 2002: Development and attributes of z/Architecture, K. E. Plambeck, W. Eckert, R. R. Rogers, and C. F. Webb

  http://www.research.ibm.com/journal/rd46-45.html

- IBM Journal of Research and Development, Vol. 53, No. 1, 2009: IBM System z10

  http://www.research.ibm.com/journal/rd53-1.html

References (4)

- **IBM Journal of Research and Development, Vol. 51, Nos. 1/2, 2007: IBM System z9**

  http://www.research.ibm.com/journal/rd51-12.html

- **IBM Journal of Research and Development, Vol. 48, Nos. 3/4, 2004: IBM eServer z990**

  http://www.research.ibm.com/journal/rd48-34.html

- **S/390 cluster technology: Parallel Sysplex, J. M. Nick, B. B. Moore, J.-Y. Chung, N. S. Bowen, IBM Systems Journal, Vol. 36, No. 2, 1997**

  http://domino.research.ibm.com/tchjr/journalindex.nsf/495f80c9d0f539778525681e00724804/05ffabe33879ed1485256bfa00685ddf?OpenDocument