



# Roadrunner Tutorial

## IBM Software Development Kit for Multicore Acceleration a.k.a. “Cell SDK”

**Cornell Wright**  
**HPC-DO**  
**[cornell@lanl.gov](mailto:cornell@lanl.gov)**

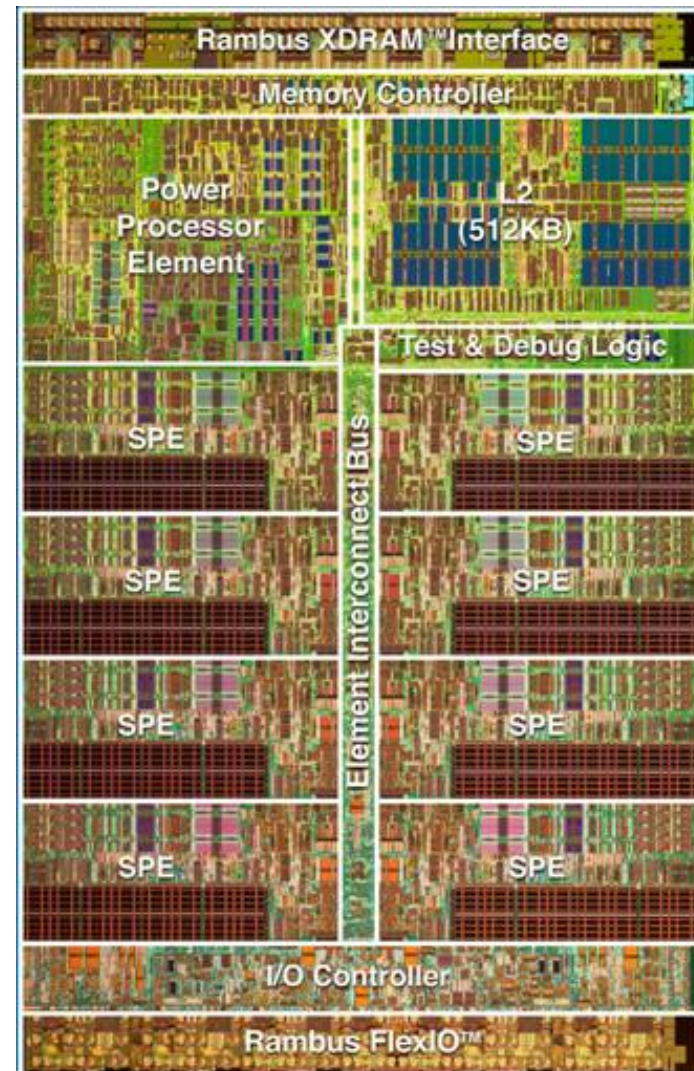
**March 6, 2008**



LA-UR-08-2819

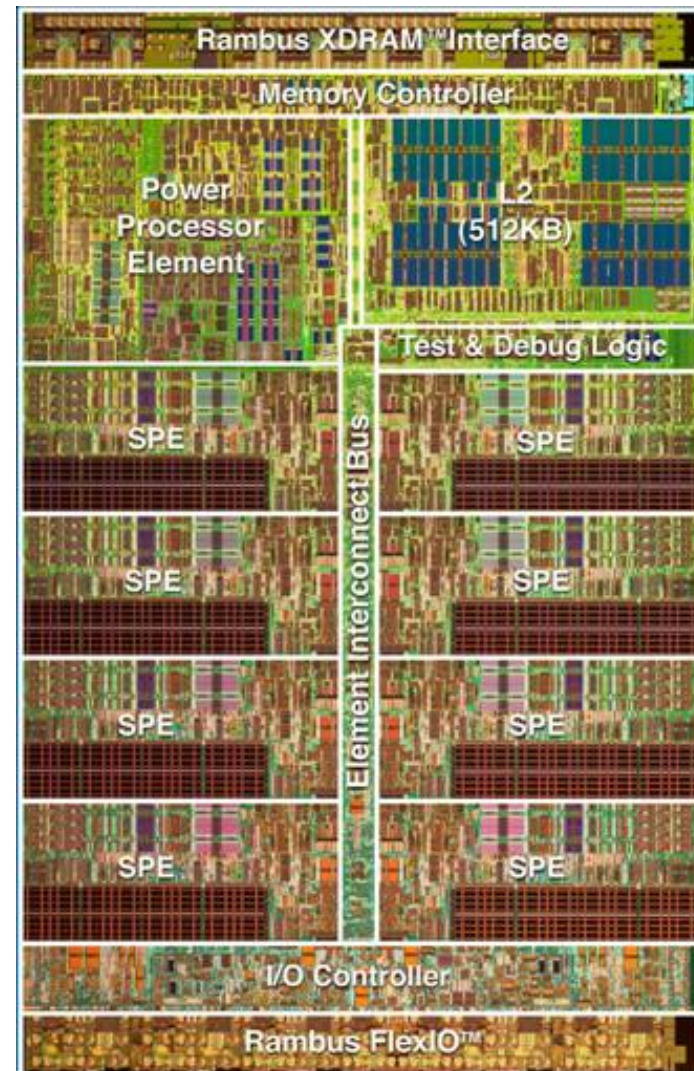
# Contents

- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



# Contents

- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



# References

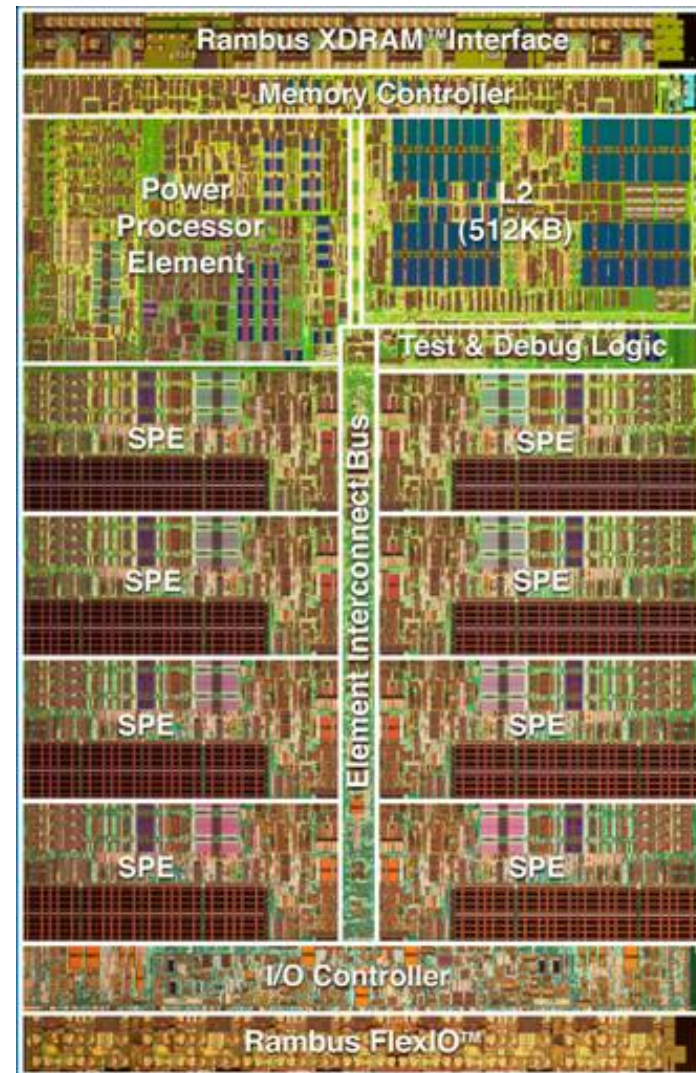
- Roadrunner Web Site
  - <http://lanl.gov/roadrunner/>
  - See “Roadrunner Technical Assessments”
  
- Roadrunner Applications Portal
  - <http://rralgs.lanl.gov/portal>
  - Roadrunner information, Cell and hybrid programming, Tutorial presentations
  - Recently added:
    - ACM Computing Frontiers paper on Roadrunner
    - Pointer to IBM on-line courses on Cell Programming

## References (continued)

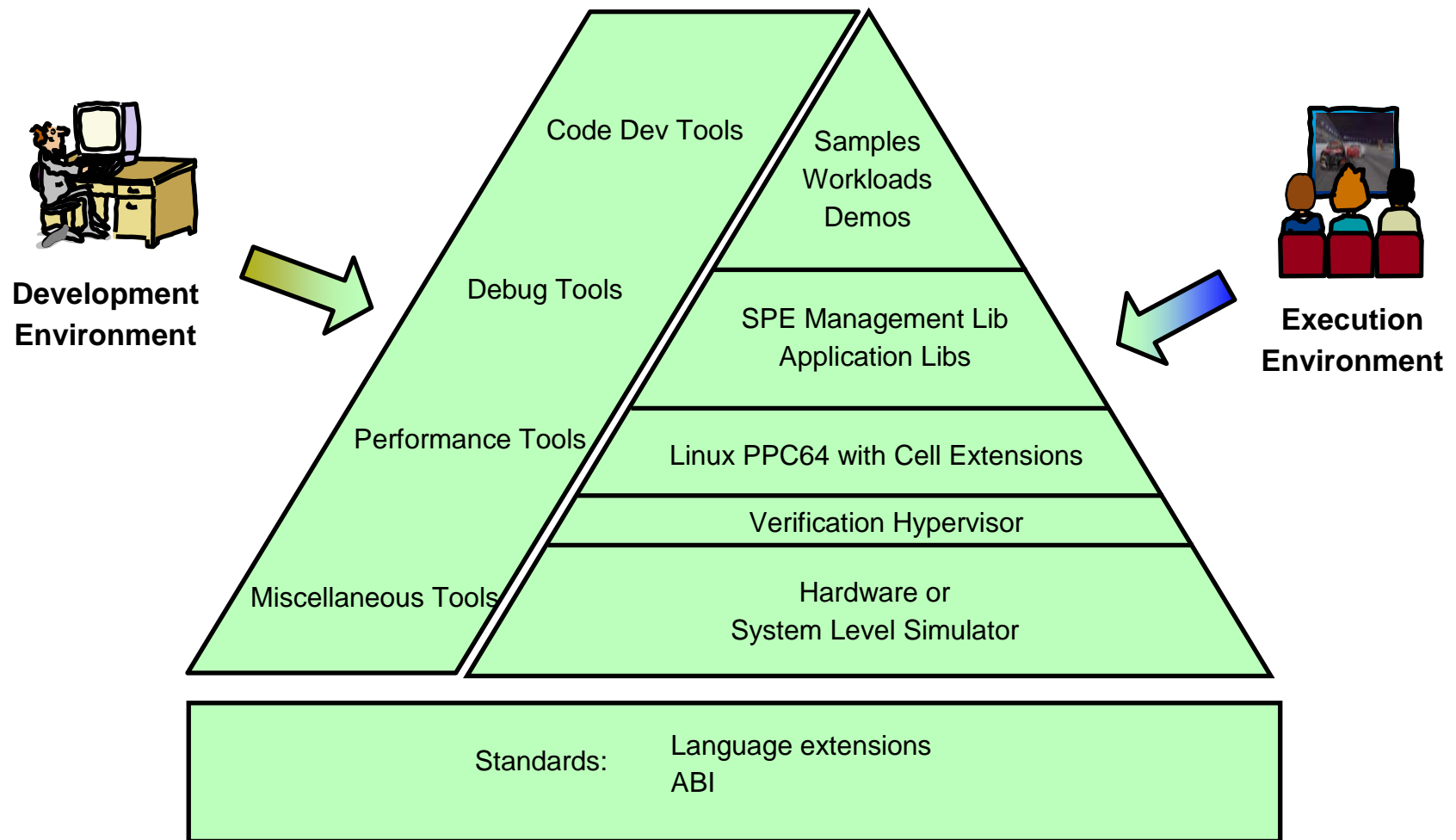
- Cell Resource Center
  - <http://www.ibm.com/developerworks/power/cell/>
  - Select “Docs” tab
  - Introduction to Cell Programming:
    - Cell Broadband Engine Programmer's Guide
    - Cell Broadband Engine Programming Tutorial
    - Cell Broadband Engine Programming Handbook
  - Programming Reference:
    - C/C++ Language Extensions for Cell Broadband Engine Architecture
    - DaCS Library for Hybrid-x86 Programmer's Guide and API Reference
  - Hardware Reference:
    - Cell Broadband Engine Architecture
    - Synergistic Processor Unit Instruction Set Architecture
- See me for a tarball of all Cell Docs
  - Also on AAIS in </home/cornell/celldoc/celldoc-2008mmdd.tgz>

# Contents

- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



# Cell Software Environment



# Cell based systems Software Stack

<b>Applications</b> <i>Partners, ISVs, Universities, Labs, etc</i>
<b>Sector Specific Libraries</b> <i>ISVs, Universities, Labs, Open Source, etc.</i>
<b>Cluster and Scale out Systems</b> <i>Cluster Systems Management, Accelerator Management,, cluster file systems and protocols, etc</i>

Market Segment Specific

Ubiquitous for all Markets

<b>Application Tooling and Environment</b> <i>Programming model/APIs for Cell and Hybrid, Eclipse IDE, Performance Tools</i>
<b>Compilers</b> C, C++, Fortran, etc
<b>Core Libraries</b> e.g. SPE intrinsic, etc
<b>Linux Operating System Features</b> e.g. SPE exploitation, CellBE support
<b>Device Drivers</b>
<b>Firmware</b> e.g. Blades, Development platforms, etc
<b>CellBE Based Hardware</b>

In addition **SDK 3.0** has: MAMBO simulator enhancements and overall simulation speedup; GA Quality Release on RHEL5.1; More comprehensive performance and integration test suite; System test for hybrid systems  
RHEL5.1 Enterprise Linux Distro

← H.264 encoder and decoder, BLAS, ALF samples, Black Scholes sample

← Data communication and synchronization layer (DaCS), Accelerated Library Framework (ALF); Prorotype ALF and DaCS hybrid x86-Cell function, newlib enhancements; Performance tooling & visualization (VPA), Code Analysis Tooling (oProfile, SPU timing, PDT, FDPR-Pro); Prototype Gdbserver support for combined hybrid (Cell/x86) remote debugging; IDE ALF / DaCS Data description templates,wizards,code builders; IDE PDT integration; Prototype IDE Hybrid Performance Analysis

← XLC GA with additional distro hosted platform coverage/OpenMP/Fortran, gcc C++ library support and exception handling; gcc and xlc continued work on auto-vectorization/SIMDization; gcc fortran on PPE& SPE, ADA on PPE

← Libspe2.0 enhancements, Newlib enhancements, tag manager and spetimers on SPEs, Infix vector operations

← RHEL5.1 Distro support, Fedora7: IPMI and Power Executive Support, RAS Enhancements, MSI on QS21, Perfmon2, SPE statistics, SPE Context in Crash Files, Axon DDR2 for Swap

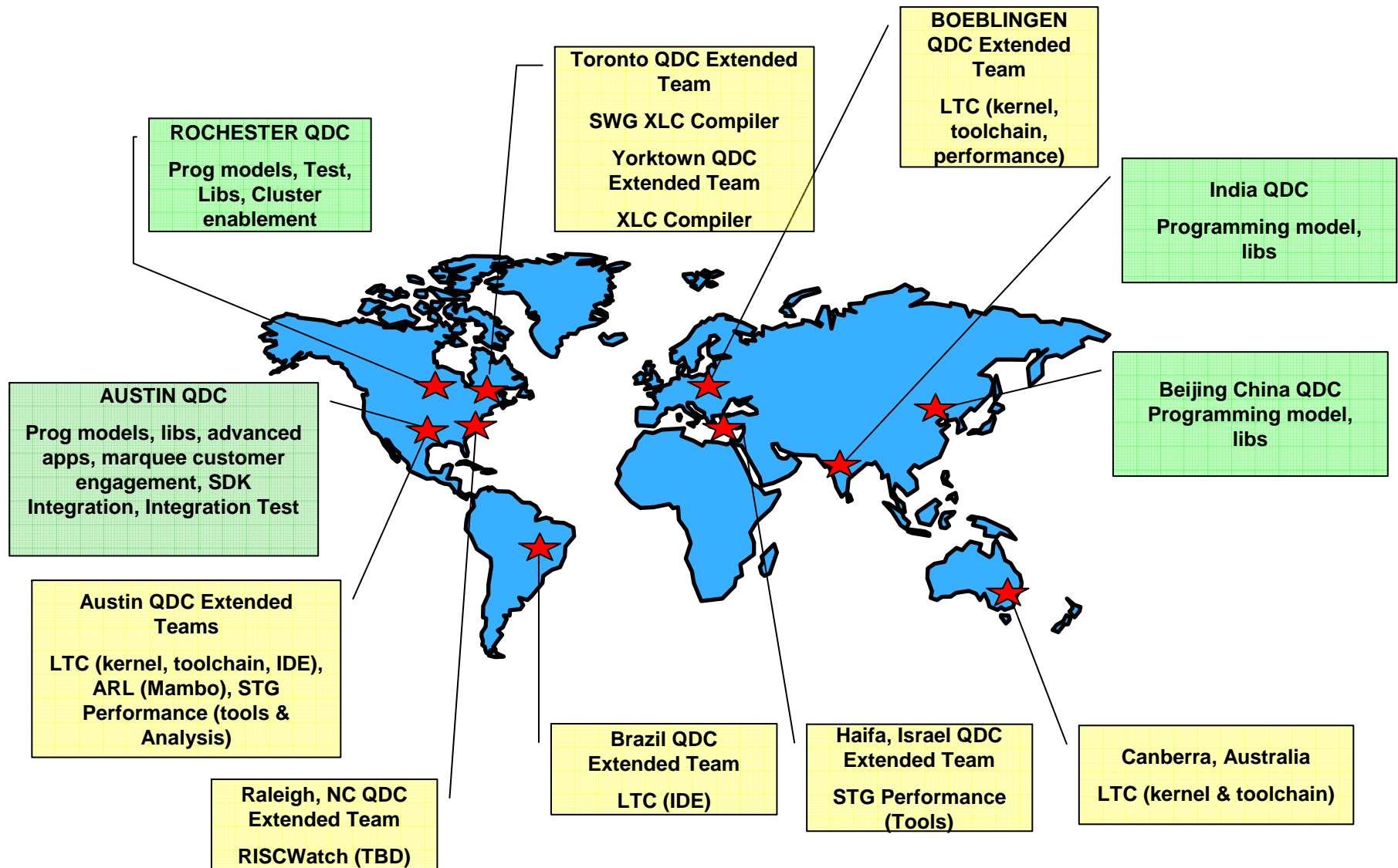
← QS20, QS21 support, F7: Triblade prototype

← QS20, QS21 support, F7: Triblade prototype

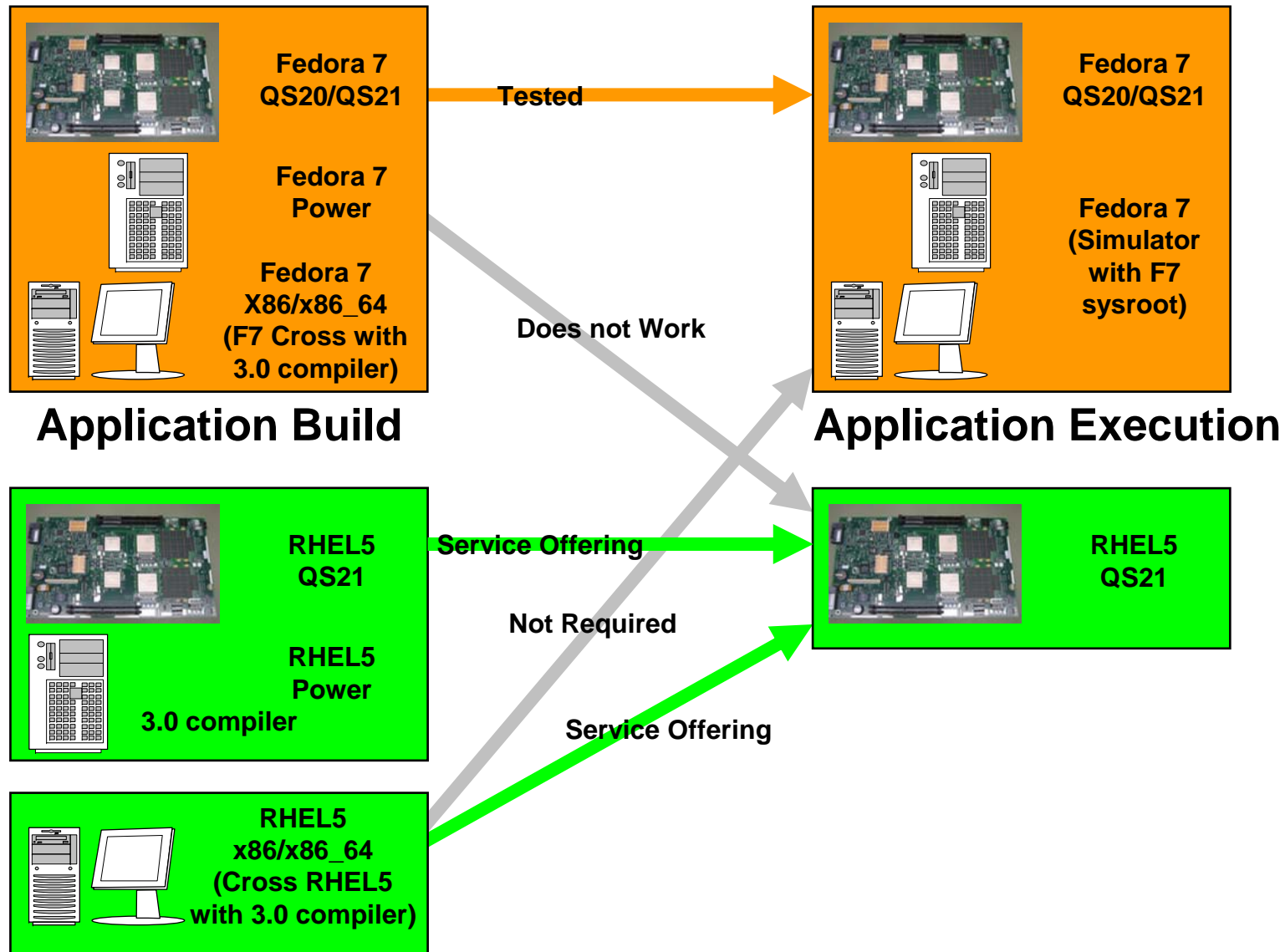
# SDK v3.0 Themes & Enhancements

- **Product Level Tested**
- **Multiple HW Platform Support**
  - QS20 (CB1) – Fedora Only
  - QS21(CB+) – Production Support
- **Linux Support**
  - Fedora 7 (Kernel level 2.6.22)
  - Red Hat Enterprise Level v5.1 (Kernel Level 2.6.18)
  - Toolchain packages: gcc 4.1.1, binutils 2.17+, newlib 1.15+, gdb 6.6+
- **Programmer Productivity – Performance Tools**
  - VPA – Visual Performance Analyzer
  - PDT – Performance Debugging Tool
  - PEP/Lock Analyzer & Trace Analysis Tools
  - CodeAnalyzer
  - Enhanced Oprofile support
  - FDP-PRO for Cell
  - Hybrid Code Analyzer
  - Hybrid System Performance and Tracing Facility
- **Programmer Productivity – Development**
  - Eclipse IDE plug-ins
  - Dual source XLC , Dual Source XLF – Fortran (beta), Single Source XLC (beta)
  - Cell and Hybrid HPC software sample code
  - Enhanced GNU toolchain support
    - GNU Fortran for PPE & SPE
    - GNU ADA (GNAT) for PPE
    - gcc autovectorization and performance enhancements
- **Programmer Productivity - Runtime**
  - Product Level ALF and DaCs for Cell
  - Hybrid DaCS/ALF (Prototype)
  - Productization of combined ppe/spe gdb debugger
  - SPE-side Software Managed Cache (from iRT technology)
- **Market Segment Library Enablement**
  - Highly optimized SIMD and MASS Math Libraries
  - Highly Optimized BLAS
  - Highly optimized libFFT
  - Monte Carlo RNG Library
  - Cell Security Technology (prototype/preview)

# IBM SDK Development Worldwide Teams



# Supported Environments



# SDK 3.0 Packages

	Component	RHEL5u1 Product	RHEL5u1 Devel	RHEL5u1 Extras	Fedora 7 Devel	Fedora 7 Extras
GA SDK	License	IPLA	ILAN	ILAER	ILAN	ILEAR
	Kernel, LibSPE	Included in RHEL			BSC Download	
	GCC Compiler	BSC Download	BSC Download		BSC Download	
	Performance Tools (FDPR-PRO, PDT, PDTR)	Yes	Yes		Yes	
	Development Libraries (ALF, DaCS, BLAS, SIMDMath, MASS)	Yes	Yes		Yes	
	Cell Eclipse IDE	Yes	Yes		Yes	
	Examples, Demos Tutorial, Docs	Yes	Yes		Yes	
	BETA SDK	XL C/C++ SS compilers			Yes	
Development Libraries (FFT, MC, SPU Timer, ALF/DaCS for Hybrid)				Yes		Yes
Performance Tools (SPU Timing)				Yes		Yes
Performance Tools (OProfile)						BSC Download
Mambo Simulator, Isolation, and CPC						Yes
Non-SDK	GA XL C/C++ DS compilers	Separate Product with 60 day Trial				
	GA Fortran DS compiler	Separate Product with Beta and then GM 4Q07			Not Supported	
	Visual Performance Analyzer	alphaWorks Download				

# Installation

- YUM-based with a wrapper script
  - Repositories for BSC and ISO images
  - YUM groups with mandatory, default and optional RPMs
    - Cell Runtime Environment (only installed with - -runtime flag)
    - Cell Development Libraries
    - Cell Development Tools
    - Cell Performance Tools
    - Cell Programming Examples
    - Cell Simulator
  
- GUI install provided by pirut (using `-gui` flag)
  
- Script also supports
  - Verify – lists what is installed
  - Uninstall
  - Update – to a service pack (RHEL5.1 only)
  - Backout – remove previous service pack (RHEL5.1 only)

# Overview of Installation

1. Install Operating System on hardware (diskless QS21 requires remote boot)
2. Uninstall SDK 2.1 or SDK 3.0 early release
3. Install pre-requisites including RHEL5.1 specifics:
  - Install compat-libstdc++
  - Install libspe2 runtimes
  - Create and install ppu-sysroot RPMs for cross-compilation
4. Download installer RPM and required ISO images
  - Physical media and product TAR file contain both and additional instructions (README.1<sup>st</sup>)
5. Install the SDK installer
  - `rpm -ivh cell-install-3.0.0.1.0.noarch.rpm`
6. Start the install
  - `cd /opt/cell`
  - `./cellsdk [--iso <isodir>] [--gui] install`
7. Perform post-install configuration
  - RHEL 5.1 specifics (elfspe on QS21, libspe2 and netpbm development libraries)
  - Complete IDE install into Eclipse and CDT
  - Configure DaCS daemons for DaCS for Hybrid-x86 and ALF for Hybrid-x86
  - Sync up the Simulator sysroot (Fedora 7 only)

# SDK 3.0 Documentation

- Supplied as:
  - PDFs, Man Pages, READMEs, XHTML (for accessibility)
- Contained in:
  - cell-install-3.0.0-1.0.noarch.rpm (Installation Guide in the installer RPM)
  - cell-documentation-3.0-5.noarch.rpm
  - cell-extras-documentation-3.0-5.noarch.rpm
  - alfman-3.0-10.noarch.rpm
  - dacsman-3.0-6.noarch.rpm
  - libspe2man-2.2.0-5.noarch.rpm
  - simdman-3.0-6.noarch.rpm
  - Individual other RPMs e.g. Tutorial, IDE
- Located in:
  - /opt/cell/sdk/docs - subdirectories used for different parts of the SDK
  - developerWorks - <http://www-128.ibm.com/developerworks/power/cell/documents.html>
- Other Documentation:
  - XL C/C++ compilers – see <http://www.ibm.com/software/awdtools/xlcpp/library/>
  - QS20/QS21 hardware – see dW documentation site for links
  - CBEA Architecture docs - see dW documentation site for links

# SDK 3.0 Documentation

<b>Software Development Kit</b>
<a href="#">IBM SDK for Multicore Acceleration Installation Guide</a>
<a href="#">Cell Broadband Engine Programming Handbook</a>
<a href="#">Cell Broadband Engine Programming Tutorial</a>
<a href="#">Cell Broadband Engine Programmer's Guide</a>
<a href="#">Oprofile (SDK Programmer's Guide)</a>
<a href="#">PDT (SDK Programmer's Guide)</a>
<a href="#">Security SDK V3.0 Installation and User's Guide</a>
<b>Programming Tools Documentation</b>
<a href="#">Performance Analysis with the IBM Full-System Simulator</a>
<a href="#">IBM Full-System Simulator User's Guide</a>
<a href="#">XL C/C++ Compiler Information</a> <a href="#">Installation Guide, Getting Started, Compiler Reference</a> <a href="#">Language Reference, Programming Guide</a>
<a href="#">XL Fortran Compiler Information</a> <a href="#">Installation Guide, Getting Started, Compiler Reference</a> <a href="#">Language Reference, Programming Guide</a>
<a href="#">Using the single-source compiler</a>
<a href="#">IBM Visual Performance Analyzer User's Guide</a>

<b>Programming Standards</b>
<a href="#">C/C++ Language Extensions for Cell Broadband Engine Architecture</a>
<a href="#">SPU Application Binary Interface Specification</a>
<a href="#">SIMD Math Library Specification for Cell Broadband Engine Architecture</a>
<a href="#">Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification</a>
<a href="#">SPU Assembly Language Specification</a>
<b>Programming Library Documentation</b>
<a href="#">Data Communication and Synchronization Programmer's Guide and API Reference</a>
<a href="#">Data Communication and Synchronization for Hybrid-x86 Programmer's Guide and API Reference</a>
<a href="#">SPE Runtime Management Library</a>
<a href="#">SPE Runtime Management Library Version 1.2 to 2.2 Migration Guide (revised name)</a>
<a href="#">Accelerated Library Framework Programmer's Guide and API Reference</a>
<a href="#">Accelerated Library Framework for Hybrid-x86 Programmer's Guide and API Reference</a>
<a href="#">Software Development Kit 3.0 SIMD Math Library Specifications</a>
<a href="#">Basic Linear Algebra Subprograms Programmer's Guide and API Reference</a>
<a href="#">Example Library API Reference</a>
<a href="#">Cell BE Monte Carlo Library API Reference Manual</a>
<a href="#">Cell Broadband Engine Security Software Development Kit - Installation and User's Guide</a>
<a href="#">SPU Timer Library</a>
<a href="#">Mathematical Acceleration Subsystem (MASS)</a>

Updated, New

## Related Products

- IBM XL C/C++ for Multicore Acceleration for Linux
  - <http://www-306.ibm.com/software/awdtools/ccompilers/>
  - <http://www-306.ibm.com/software/awdtools/xlcpp/features/multicore/>
- IBM XL Fortran for Multicore Acceleration for Linux on System p
  - <http://www.alphaworks.ibm.com/tech/cellfortran> (GA planned for 11/30)
- Visual Performance Analyzer (VPA) on alphaWorks
  - <http://www.alphaworks.ibm.com/tech/vpa>
- IBM Assembly Visualizer for Cell Broadband Engine on alphaWorks
  - <http://www.alphaworks.ibm.com/tech/asmvis>
- IBM Interactive Ray Trace Demo on alphaWorks
  - <http://www.alphaworks.ibm.com/tech/irt>

# Linux Kernel

- **Patches made to Linux 2.6.22 kernel to provide services required to support the Cell BE hardware facilities**
- **Patches distributed by Barcelona Supercomputer Center**
  - <http://www.bsc.es/projects/deepcomputing/linuxoncell>
- **For the QS20/QS21,**
  - the kernel is installed into the /boot directory
  - yaboot.conf is modified
  - needs reboot to activate this kernel

# Kernel and LibSPE2

- Distro Support
  - RHEL5.1
  - Fedora7
- QS21 Support
  - IPMI and Power Executive Support
  - RAS Enhancements
  - Perfmon2
  - MSI support for QS21 (PCIe interrupt signaling)
  - Axon DDR2 for Swap
- SPE utilization statistics
- SPE preemptive scheduling
- SPE Context in Crash Files
- Enabling for Secure SDK (under CDA only)

# IBM Full-System Simulator

- **Emulates the behavior of a full system that contains a Cell BE processor.**
- **Can start Linux on the simulator and run applications on the simulated operating system.**
- **Supports the loading and running of statically-linked executable programs and standalone tests without an underlying operating system.**
- **Simulation models**
  - **Functional-only simulation:** Models the program-visible effects of instructions without modeling the time it takes to run these instructions.
    - → For code development and debugging.
  - **Performance simulation:** Models internal policies and mechanisms for system components, such as arbiters, queues, and pipelines. Operation latencies are modeled dynamically to account for both processing time and resource constraints.
    - → For system and application performance analysis.
- **Improvements in SDK 3.0:**
  - New Fast execution mode on 64-bit platforms
  - Improved performance models
  - New Graphical User Interface features
  - Improved diagnostic and statistics reporting
  - New device emulation support to allow booting of unmodified Linux kernels
  - Supplied with Fedora 7 Sysroot Image

# GCC and GNU Toolchain

- Base toolchain
  - Based on GCC 4.1.1 extended by PPE and SPE support
  - binutils 2.18, SPE newlib 1.15.0+, GDB 6.6+
- Support additional languages
  - GNU Fortran for PPE and SPE
    - **No** SPE-specific extensions (e.g. intrinsics)
  - GNU Ada for PPE only
    - Will provide Ada bindings for `libspe2`
- Compiler performance enhancements
  - Improved auto-vectorization capabilities
    - Extract parallelism from straight-line code, outer loops
  - Other SPE code generation improvements
    - If-conversion, modulo-scheduling enhancements
- New hardware support
  - Code generation for SPE with enhanced double precision FP

## GCC and GNU Toolchain

- Help simplify Cell/B.E. application development
  - Syntax extension to allow use of operators (+, -, ...) on vectors
  - Additional PPU VMX intrinsics
  - Simplify embedding of SPE binaries into PPE objects
  - SPE static stack-space requirement estimation
  - Extended C99/POSIX run-time library support on SPE
- Integrated PPE address-space access on SPE
  - Syntax extension to provide address-space qualified types
  - Access PPE-side symbols in SPE code
  - Integrated software-managed cache for data access
- Combined PPE/SPE debugger enhancements
  - Extended support for debugging `libspe2` code
  - Improved resolution of multiply-defined symbols

# GNU tool chain

- **Contains the GCC compiler for the PPU and the SPU.**
  - ppu-gcc, ppu-g++, ppu32-gcc, ppu32-g++, spu-gcc, spu-g++
  - For the PPU, GCC replaces the native GCC on PPC platforms and it is a cross-compiler on x86. The GCC for the PPU is preferred and the makefiles are configured to use it when building the libraries and samples.
  - For the SPU, GCC contains a separate SPE cross-compiler that supports the standards defined in the following documents:
    - C/C++ Language Extensions for Cell BE Architecture V2.4
    - SPU Application Binary Interface (ABI) Specification V1.7
    - SPU Instruction Set Architecture V1.2
- **The assembler and linker are common to both the GCC and XL C/C++ compilers.**
  - ppu-ld, ppu-as, spu-ld, spu-as
  - The GCC associated assembler and linker additionally support the SPU Assembly Language Specification V1.5.
- **GDB support is provided for both PPU and SPU debugging**
  - The debugger client can be in the same process or a remote process.
- **GDB also supports combined (PPU and SPU) debugging.**
  - ppu-gdb, ppu-gdbserver, ppu32-gdbserver

# XL C/C++/Fortran Compilers

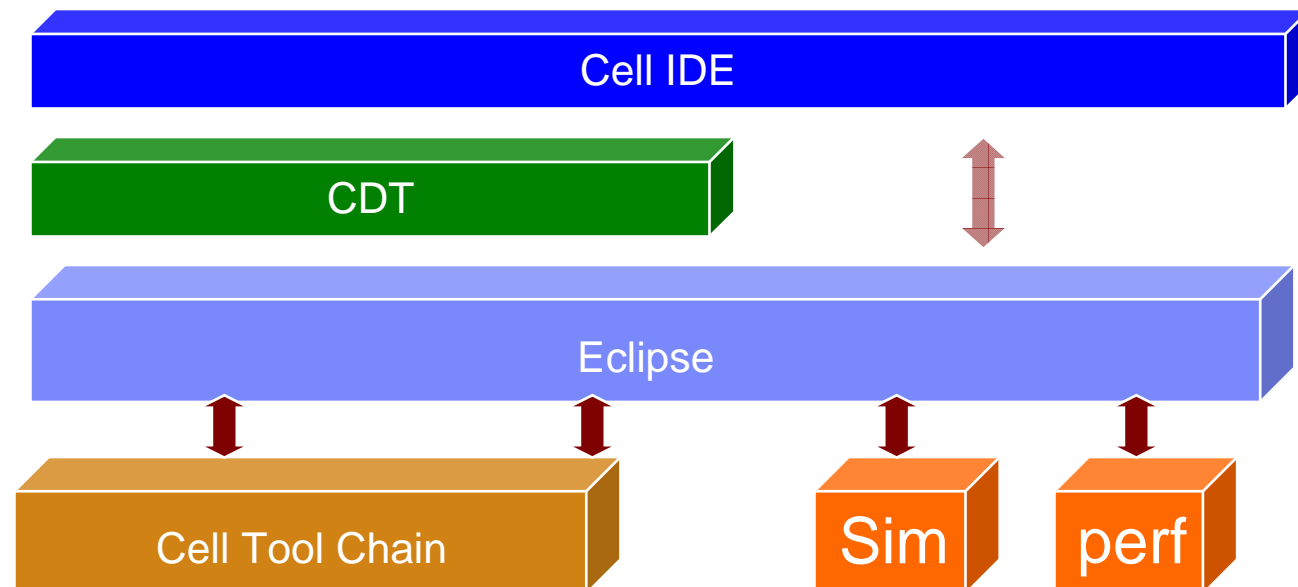
- IBM XL C/C++ for Multicore Acceleration for Linux, V9.0 (dual source compiler)
  - Product quality and support
  - Performance improvements in auto-SIMD
  - Improved diagnostic capabilities for detecting SIMD opportunities (-qreport)
  - Enablement of high optimization levels (O4, O5) on the SPE
  - Automatic generation of code overlays
- IBM XL Fortran for Multicore Acceleration for Linux, V11.1 (dual source compiler)
  - Beta level (with GA targeted for 11/30/07)
  - Optimized Fortran code generation for PPE and SPE
  - Support for Fortran 77, 90 and 95 standards as well as many features from the Fortran 2003 standard
  - Auto-SIMD optimizations
  - Automatic generation of code overlays
- IBM XL C/C++ Alpha Edition for Multicore Acceleration, V0.9 (single source compiler)
  - Beta level
  - Allows programmer to use OpenMP directives to specify parallelism on PPE and SPE
    - Compiler hides complexity of DMA transfers, code partitioning, overlays, etc.. from the programmer
    - See <http://www.research.ibm.com/journal/sj/451/eichenberger.html>

## IBM XL C/C++ compiler

- A cross-compiler hosted on a x86 and PPC platform.
- Requires the GCC Tool chain for cross-assembling and cross-linking applications for both the PPE and SPE.
- Supports the revised 2003 International C++ Standard *ISO/IEC 14882:2003(E), Programming Languages -- C++* and the *ISO/IEC 9899:1999, Programming Languages -- C standard*, also known as C99, and the C89 Standard and K&R style, and language extensions for vector programming
- The XL C/C++ compiler provides the following invocation commands:
  - ppuxlc, ppuxlc++
  - spuxlc, spuxlc++
- The XL C/C++ compiler includes the following base optimization levels:
  - -O0: almost no optimization
  - -O2: strong, low-level optimization that benefits most programs
  - -O3: intense, low-level optimization analysis with basic loop optimization
  - -O4: all of -O3 and detailed loop analysis and good whole-program analysis at link time
  - -O5: all of -O4 and detailed whole-program analysis at link time.
- **Auto-SIMDization is enabled at O3 -qhot or O4 and O5 by default.**

# Eclipse IDE

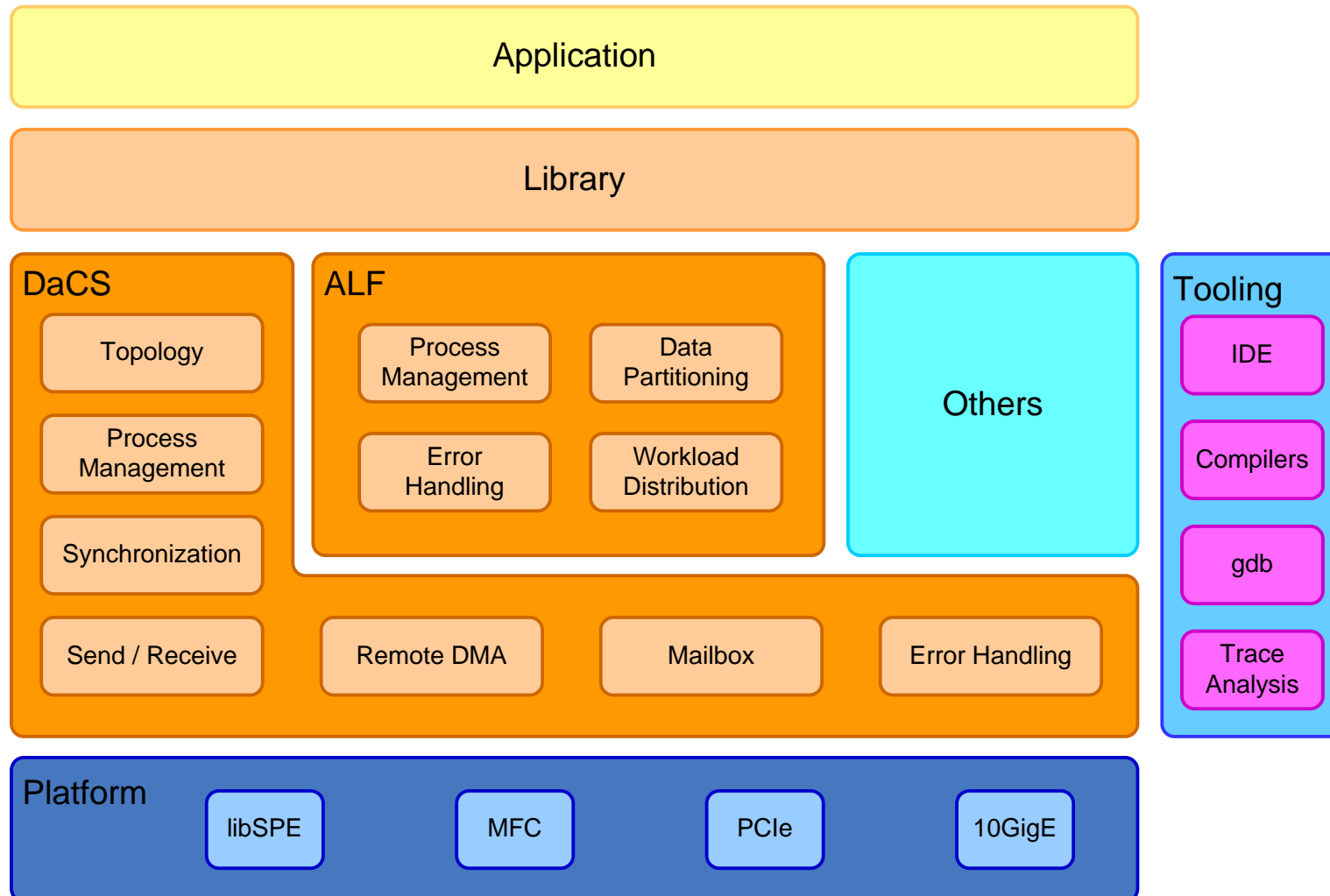
- IBM IDE, which is built upon the Eclipse and C/C++ Development Tools (CDT) platform, integrates Cell GNU tool chain, XLC/GCC compilers, IBM Full-System Simulator for the Cell BE, and other development components in order to provide a comprehensive, user-friendly development platform that simplifies Cell BE development.



## Cell IDE Key Features

- Cell C/C++ PPE/SPE managed make project support
- A C/C++ editor that supports syntax highlighting; a customizable template; and an outline window view for procedures, variables, declarations, and functions that appear in source code.
- Full configurable build properties.
- A rich C/C++ source level PPE and/or SPE cell GDB debugger integrated into eclipse.
- Seamless integration of Cell BE Simulator into Eclipse
- Automatic makefile generator, builder, performance tools, and several other enhancements.
- Support development platforms (x86, x86\_64, Power PC, Cell)
- Support target platforms
  - Local Cell Simulator
  - Remote Cell Simulator
  - Remote Native Cell Blade
- Performance tools Support .
- Automatic embedSPU integration
- ALF programming model support
- SOMA support

# ALF and DaCS: IBM's Software Enablement Strategy for Multi-core Memory-Hierarchy Systems

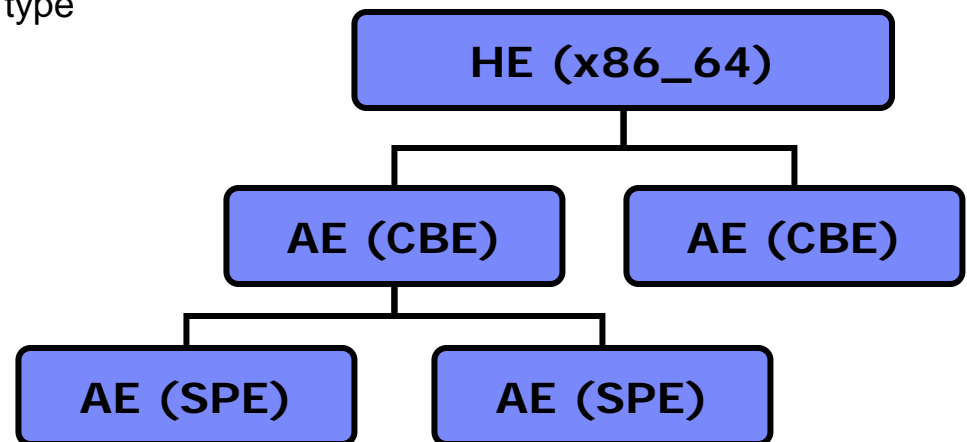


# DaCS – Data Communications and Synchronization

- Focused on data movement primitives
  - DMA like interfaces (put, get)
  - Message interfaces (send/recv)
  - Mailbox
  - Endian Conversion
- Provides Process Management, Accelerator Topology Services
- Based on Remote Memory windows and Data channels architecture
- Common API – Intra-Accelerator (CellBE), Host - Accelerator
- Double and multi-buffering
  - Efficient data transfer to maximize available bandwidth and minimize inherent latency
  - Hide complexity of asynchronous compute/communicate from developer
- Supports ALF, directly used by US National Lab for Host-Accelerator HPC environment
- Thin layer of API support on CELLBE native hardware
- Hybrid DaCS
  - Prototyped on IB Verbs (incomplete)
  - Developed on Sockets (prototype in SDK3)
  - Developed on PCI-e (Tri-blade) SDK3.0 – internal, SDK4.0 GA

# DaCS Components Overview

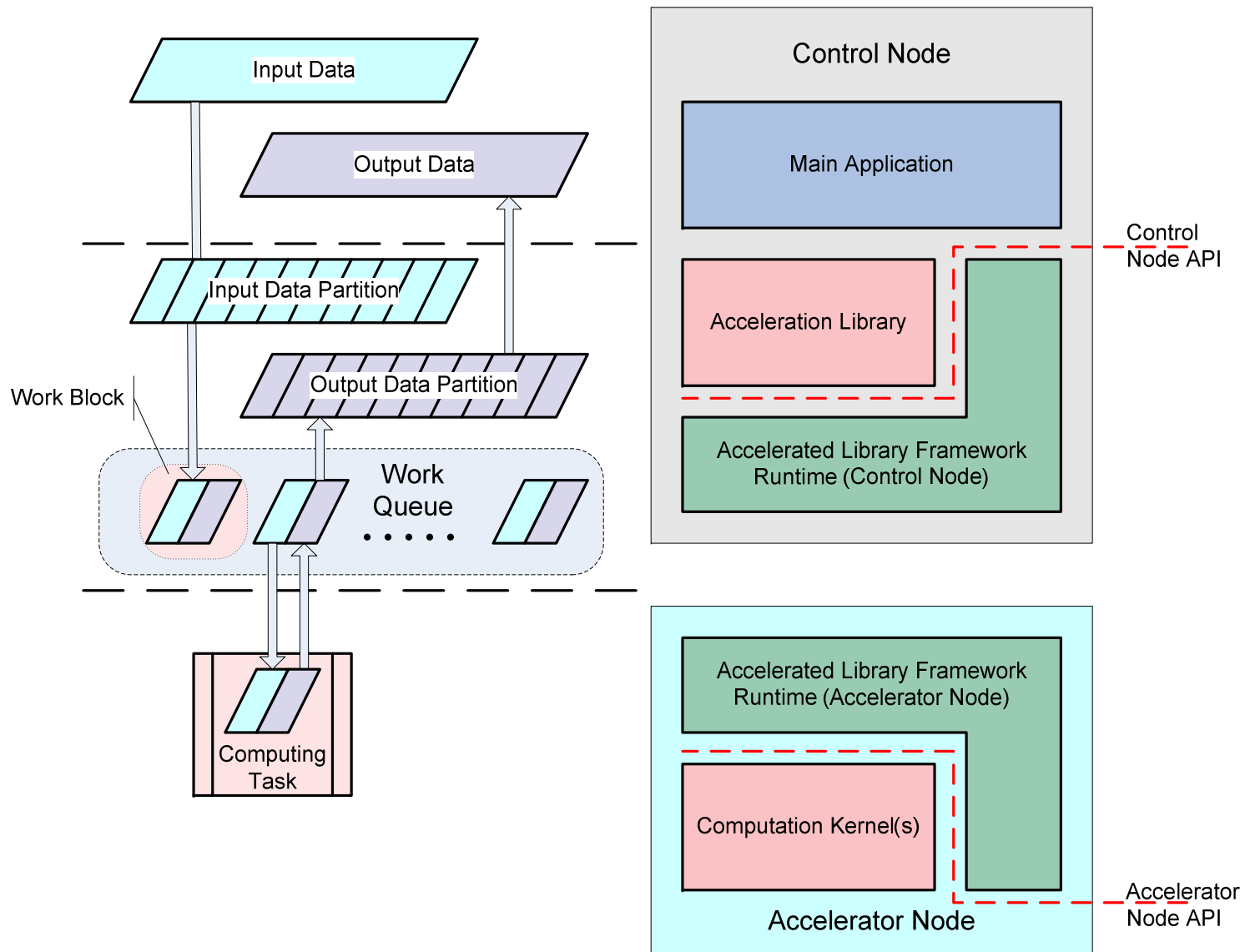
- Process Management
  - Supports remote launching of an accelerator's process from a host process
- Topology Management
  - Identify the number of Accelerators of a certain type
  - Reserve a number of Accelerators of a certain type
- Data Movement Primitives
  - Remote Direct Memory Access (rDMA)
    - put/get
    - put\_list/get\_list
  - Message Passing
    - send/receive
  - Mailbox
    - write to mailbox/read from mailbox
- Synchronization
  - Mutex / Barrier
- Error Handling



# Accelerator Library Framework (ALF) Overview

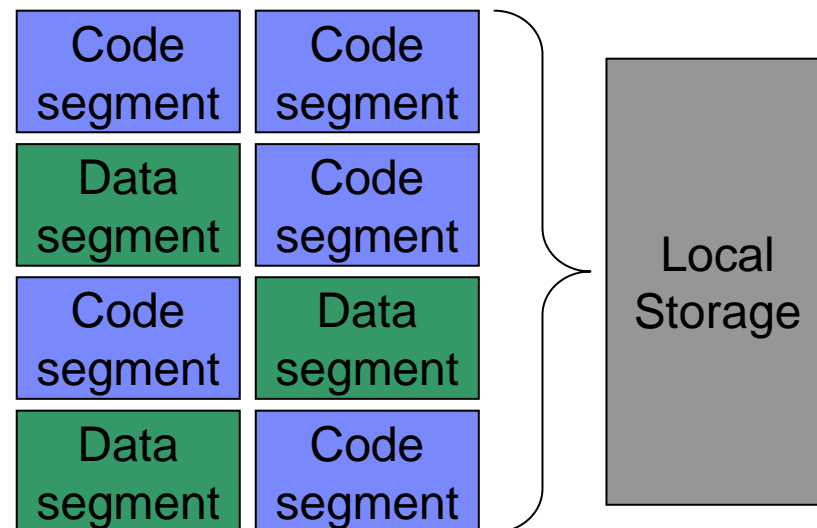
- Aims at workloads that are highly parallelizable
  - e.g. Raycasting, FFT, Monte Carlo, Video Codecs
- Provides a simple user-level programming framework for Cell library developers that can be extended to other hybrid systems.
- Division of Labor approach
  - ALF provides wrappers for computational kernels
  - Frees programmers from writing their own architectural-dependent code including: data transfer, task management, double buffering, data communication
- Manages data partitioning
  - Provides efficient scatter/gather implementations via CBE DMA
  - Extensible to variety of data partitioning patterns
  - Host and accelerator describe the scatter/gather operations
  - Accelerators gather the input data and scatter output data from host's memory
  - Manages input/output buffers to/from SPEs
- Remote error handling
- Utilizes DaCS library for some low-level operations (on Hybrid)

# ALF Data Partitioning



# SPU Overlays

- Overlays must be used if the sum of the lengths of all the code segments of a program, plus the lengths of the data areas required by the program, exceeds the SPU local storage size
- They may be used in other circumstances; for example performance might be improved if the size of a data area can be increased by moving rarely used functions (such as error or exception handlers) to overlays



## SPE Software Managed Cache

- SPE memory accesses are to Local Store Addresses only. Access to main memory requires explicit DMA calls.
  - This represents a new programming model
- Software cache has many benefits in SPE environment
  - Simplifies programming model
    - familiar load/store effective address model can be used
    - Decreases time to port to SPE
  - Take advantage of locality of reference
  - Can be easily optimized to match data access patterns

# SIMD Math Library

- **Completed Implementation of JSRE SIMD Math Library by adding:**

- tgammaf4 (PPU/SPU)
- tgamma d2 (SPU)
- lgammaf4 (PPU/SPU)
- erff4 (PPU/SPU)
- erf cf4 (PPU/SPU)
- fpclassify d2 (SPU)
- fpclassify f4 (PPU/SPU)
- nextafter d2 (SPU)
- nextafter f4 (PPU/SPU)
- modff4 (PPU/SPU)
- lldiv i2 (SPU)
- lldiv u2 (SPU)
- iroundf4 (PPU/SPU)
- rintf r (PPU/SPU)
- log1pd2 (SPU)
- log1pf4 (PPU/SPU)
- expm1d2 (SPU)
- expm1f4 (PPU/SPU)
- hypotf4 (PPU/SPU)
- sincos d2 (SPU)
- sincos f4 (PPU/SPU)
- tanhd2 (SPU)
- tanhf4 (PPU/SPU)
- atand2dp
- acosh d2 (SPU)
- acosh f4 (PPU/SPU)
- asinh d2 (SPU)
- asinh f4 (PPU/SPU)
- atanh d2 (SPU)
- atanh f4 (PPU/SPU)
- atan2d2 (SPU)
- atan2f4 (PPU/SPU)

# MASS and MASS/V Library

- Mathematical Acceleration SubSystem
  - High-performance alternative to standard system math libraries
    - i.e. libm, SIMDmath
  - Versions exist for PowerPCs, PPU, SPU
  - Up to 23x faster than libm functions
- PPU MASS
  - 57 scalar functions, 60 vector functions
  - both single and double precision
- SPU MASS
  - SDK 2.1 contains 28 SIMD functions and 28 vector functions (SP only)
  - Expanded SPU MASS to include all single-precision functions in PPU MASS
  - Added 8 new SP functions
    - erf, erfc, expm1, hypot, lgamma, log1p, vpopcnt4, vpopcnt8
  - Improved tuning of existing functions

# BLAS Library

- BLAS on PPU
  - Conforming to standard BLAS interface
  - Easy port of existing applications
  - Selected routines optimized utilizing SPUs
  - Only real single precision and real double precision versions supported. Complex version is not supported.
- Selected Routines (Based on use in Cholesky factorization/LU)
  - BLAS I (scal, copy, axpy, dot, i\_amax)
  - BLAS II (gemv)
  - BLAS III (gemm, syr, trsm)
  - Focus on single precision optimization
- BLAS on SPU
  - Offer SPU Kernel routines to SPU applications
    - Underlying functionality implemented on the SPE
    - Operate on data (input/output) residing in local store
  - Similar to the corresponding PPU routine but not conforming to APIs

# FFT Library (Prototype)

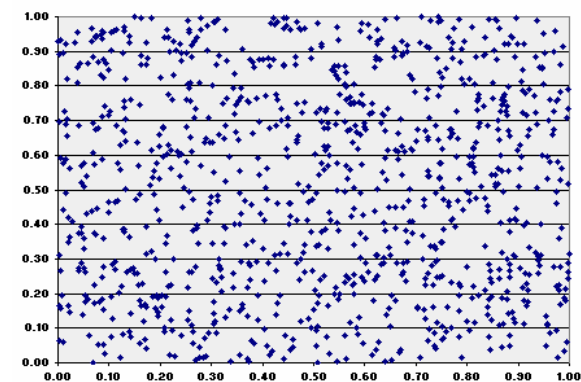
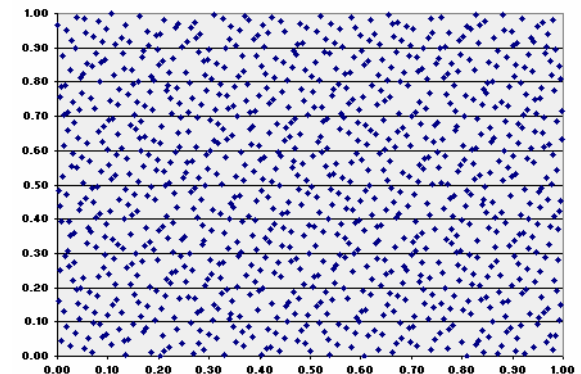
- 1D, 2D Square, 2D rectangular, 3D cube, 3D rectangular
- Integer, **Single Precision**, **Double Precision**
- **Complex to Real**, **Real to Complex**, **Complex to Complex**
- **Row size is Power-of-2, Power-of-Low-Primes**, factorization of row size includes large primes
- **SPU, BE, ppu**
- **In place, out of place**
- **Forward, Inverse**

	<b>C2R power of 2</b>	<b>R2C power of 2</b>	<b>C2C power of 2</b>	<b>C2R low primes</b>	<b>R2C low primes</b>	<b>C2C low primes</b>
<b>1D Single</b>	2 to 8192(SPU) 2 to 8192 (BE) out of place forward	2 to 8192(SPU) 2 to 8192 (BE) out of place forward	2 to 8192 SPU) 2 to 8192 (BE) out of place forward/inverse	2 to 8192(SPU) 2 to 8192 (BE) out of place forward	2 to 8192(SPU) 2 to 8192 (BE) out of place forward	2 to 8192 SPU) 2 to 8192 (BE) out of place forward/inverse
<b>2D Square Single</b>			32 to 2048 (BE) In/out of place forward/inverse			2 to 2048 (BE) out of place forward
<b>2D Rectangular Single</b>						2 to 2048 (BE) out of place forward
<b>2D Rectangular Double</b>			32 to 2048 (BE) In/out of place forward/inverse			

# Monte Carlo Random Number Generator Library (Prototype)

## Types of Random Number Generators

- True -
  - not repeatable
  - hardware support on IBM blades (not Simulator)
  
- Quasi
  - repeatable with same seed
  - attempts to uniformly fill n-dimension space
    - Sobol
  
- Pseudo
  - repeatable with same seed
    - Mersenne Twister
    - Kirkpatrick-Stoll



# Choosing a Random Number Generator

Algorithm	Size	Speed	Randomness
<b>Hardware</b>	Small	Slowest	Random
<b>Kirkpatrick-Stoll</b>	Moderate	Fast	Pseudo
<b>Mersenne Twister</b>	Moderate	Moderate	Pseudo
<b>Sobol</b>	Large	Fastest	Quasi

# SPU Timer Library (Prototype)

- Provides virtual clock and timer services for SPU programs
- Virtual Clock
  - Software managed 64-bit timebase counter
  - Built on top of 32-bit decrementer register
  - Can be used for high resolution time measurements
- Virtual Timers
  - Interval timers built on virtual clock
  - User registered handler is called on requested interval
  - Can be used for statistical profiling
  - Up to 4 timers can be active simultaneously, with different intervals

# Performance Tools – Static Analysis

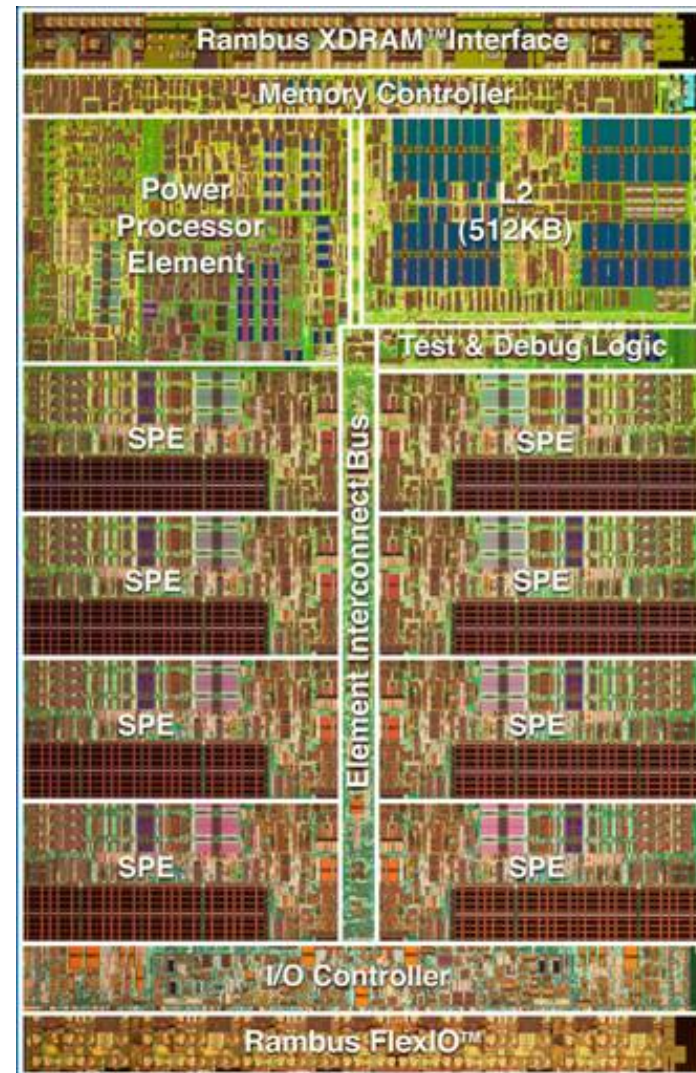
- FDPR-Pro
  - Perform global optimization at the entire executable
- SPU Timing
  - Analysis of SPE instruction scheduling

# Performance Tools – Dynamic Analysis

- Performance Debugging and Tool (PDT)
  - Generalized tracing facility; instrumentation of DaCS and ALF
  - Hybrid system support, e.g. PDT on Opteron, etc.
- PDTR: PDT post-processor
  - Post processes PDT traces
  - Provide analysis and summary reports (Lock analysis, DMA analysis, etc.)
- OProfile (Fedora 7 only)
  - PPU Time and event profiling
  - SPU time profiling
- Hardware Performance Monitoring (Fedora 7 only)
  - Collect performance monitoring events
  - Perfmon2 support and enablement for PAPI, etc.
- Hybrid System Performance Monitoring and Tracing facility
  - Launch, activate and dynamically configure tools on CellBlades and Opteron host blade
  - Synchronize, merge and aggregate traces
- Visual Performance Analyzer (from alphaWorks)

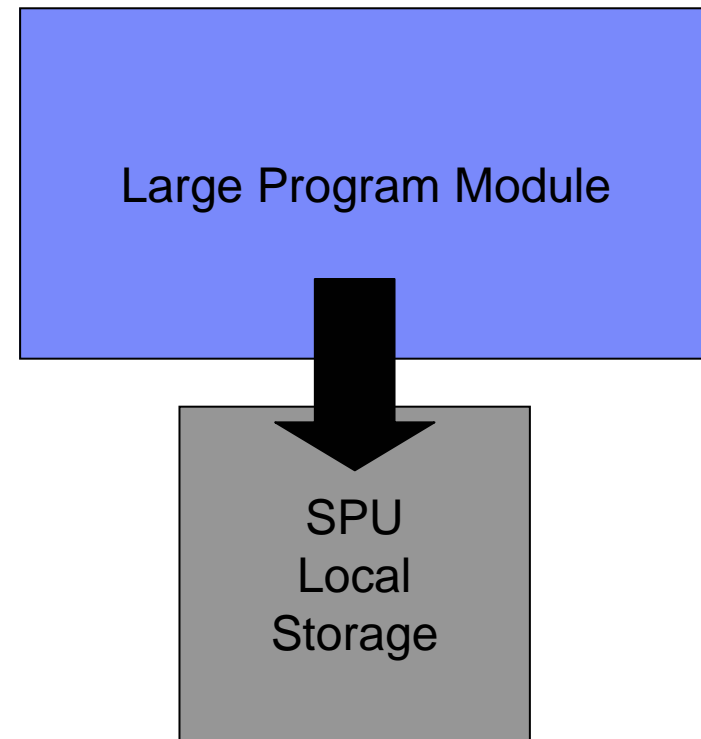
# Contents

- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



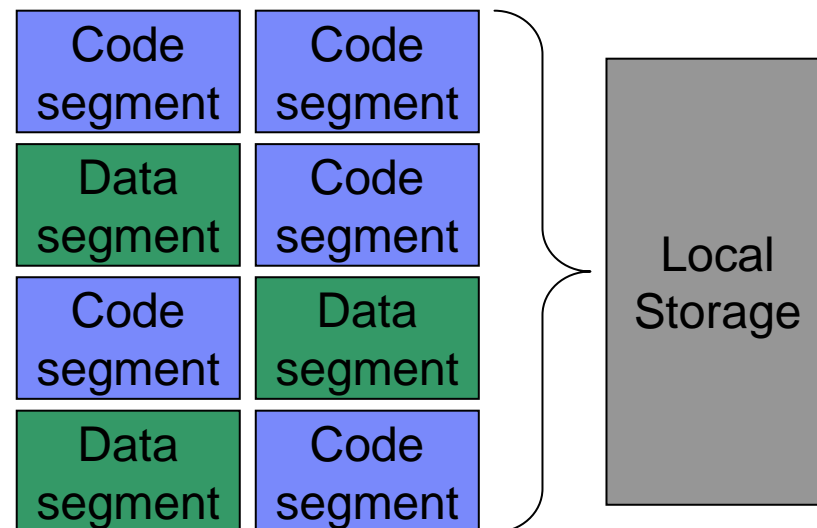
# Overall Problem

- Currently when an SPU program module is executed, all the segments of the module remain in local storage throughout execution
- When local storage is not at a premium, this is the most efficient way to execute a program
- However, when a program approaches or exceeds the limits of the available local storage, one should consider using the SPU overlay facility



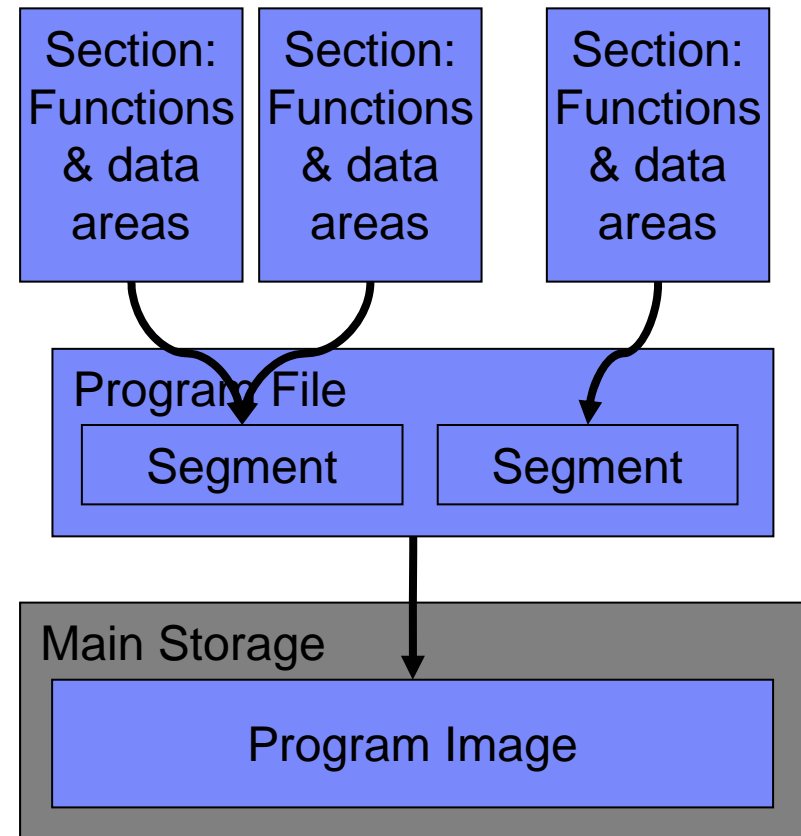
# Why use overlays

- Overlays must be used if the sum of the lengths of all the code segments of a program, plus the lengths of the data areas required by the program, exceeds the SPU local storage size
- They may be used in other circumstances; for example performance might be improved if the size of a data area can be increased by moving rarely used functions (such as error or exception handlers) to overlays



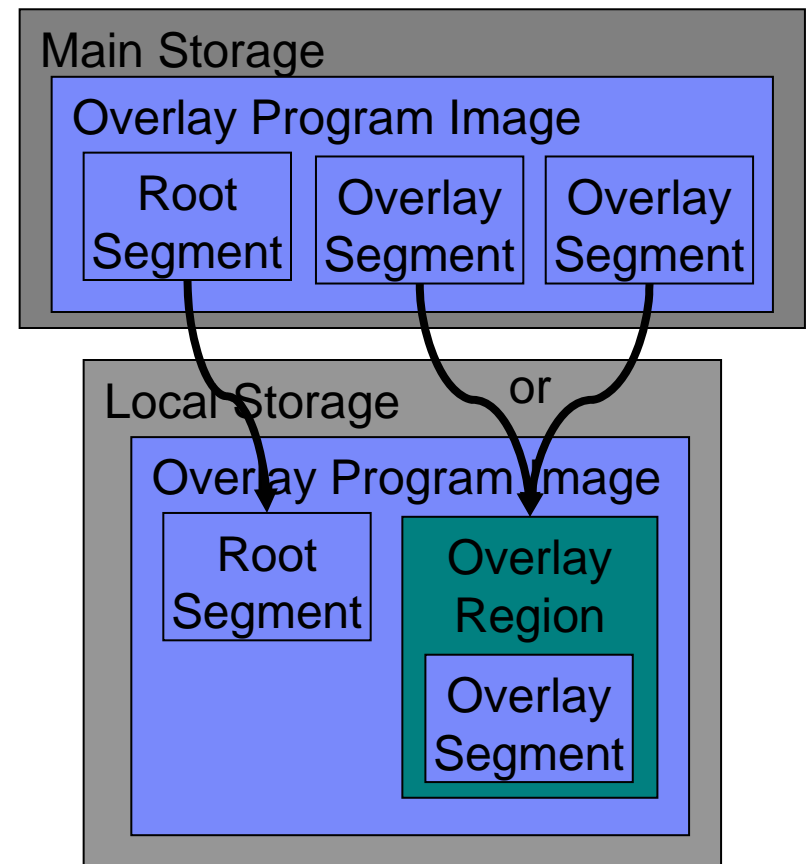
# Program structure definitions

- A segment may contains many program sections such as functions and data areas
- A program file consists of a number of segments
- A program image is a loaded program file
- A segment is the smallest unit which can be loaded as one logical entity during execution



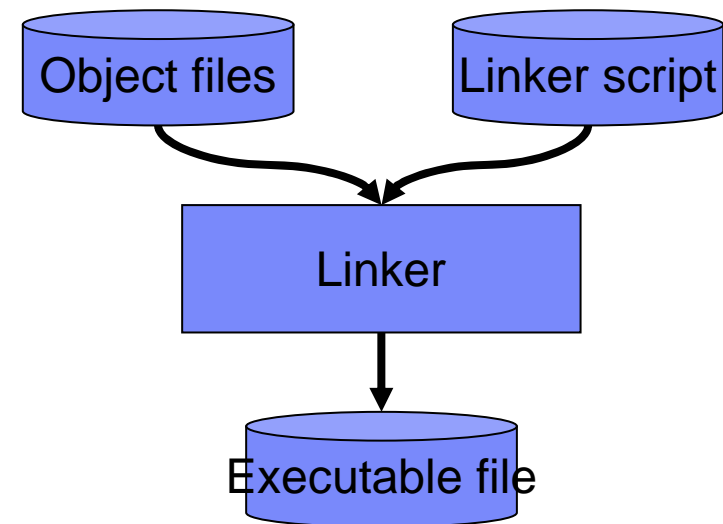
# Overlay definitions

- In an overlay structure the program image is divided into a root segment, which is always in storage, and one or more overlay regions, where overlay segments are loaded when needed
- An overlay segment is a part of a program which is not loaded into SPU local storage before the program executes, but resides in main storage until it is required
- Any given overlay segment is always loaded into the same region
- A region contains one and only one overlay segment at a time



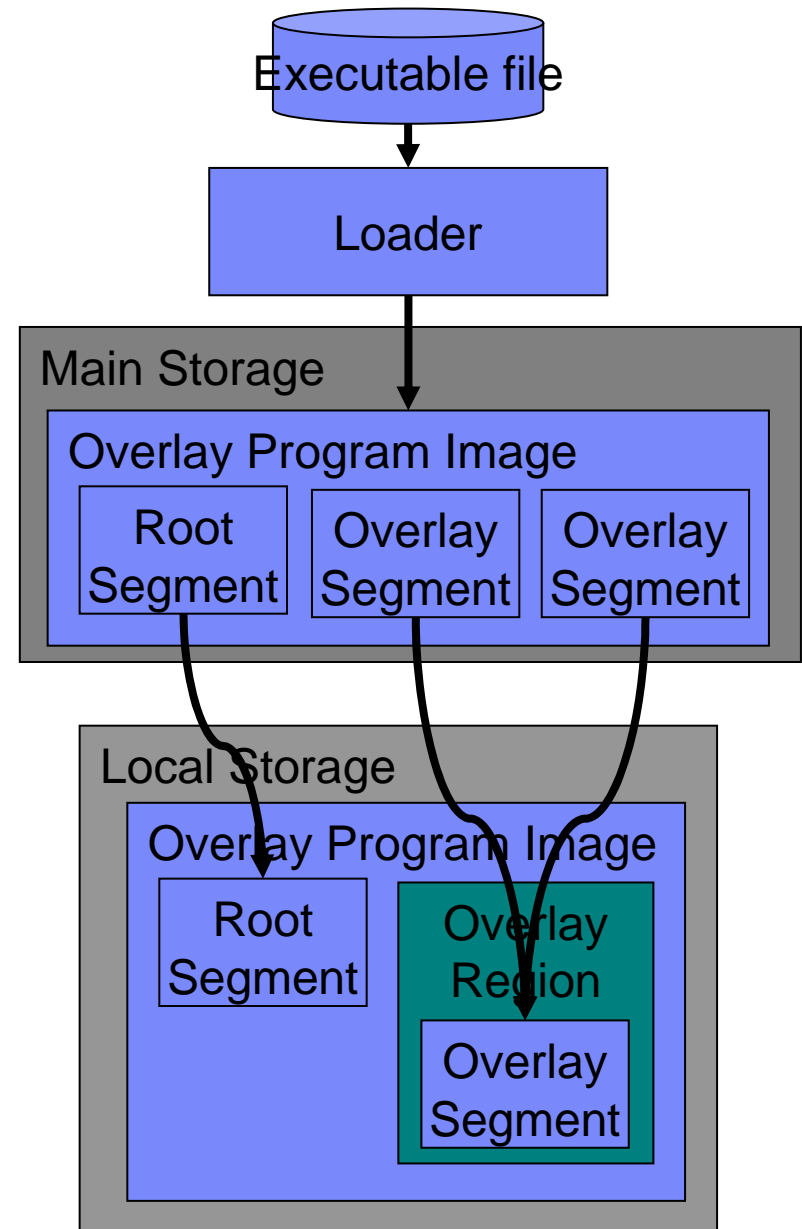
## Build time mechanism

- All that is needed to convert an ordinary program to an overlay program is the addition of a linker script to structure the overlay program
- In the script you choose the segments of the program that can be overlaid
- The system prepares for the loading of the required segments when needed during execution of the program



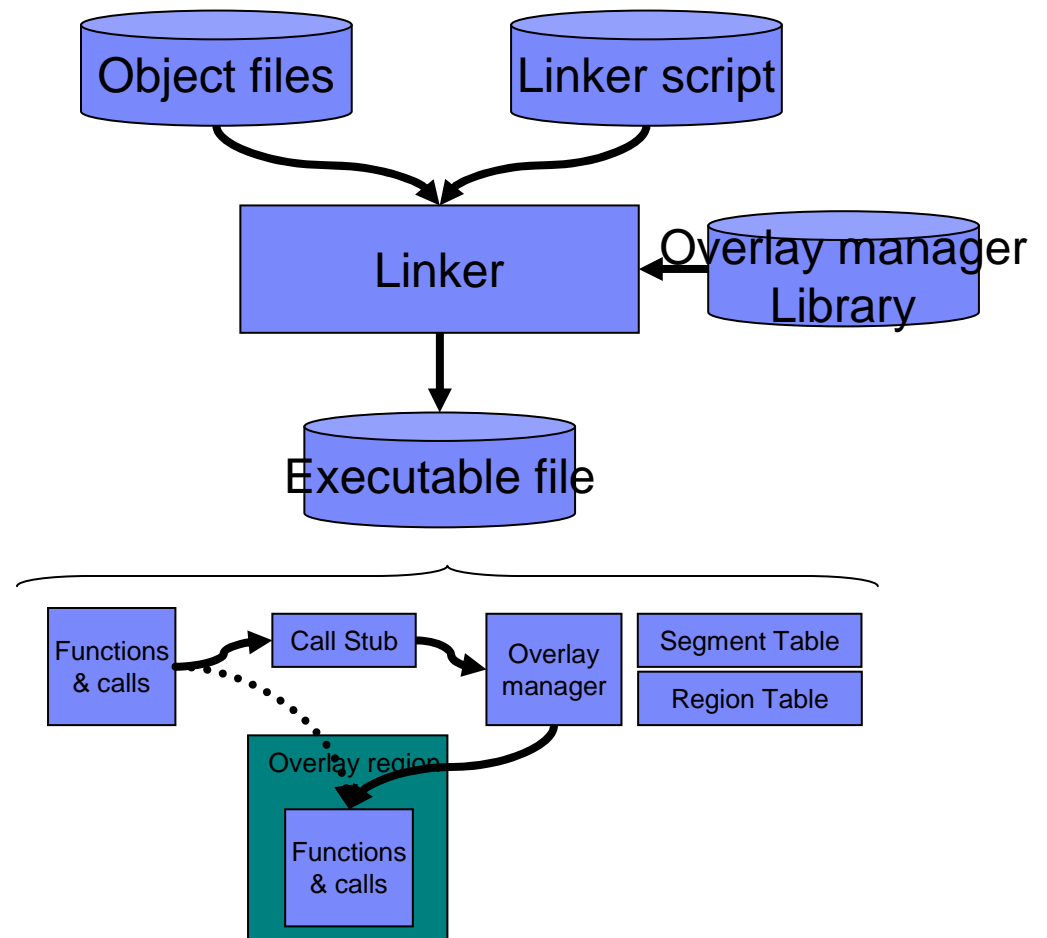
# Runtime mechanism

- During execution when a reference is made from an one segment to another, the system determines whether the code required is already in local storage
- If it is not, the segment is loaded dynamically and may replace or overlay a previous segment already in storage
- The system transfers control to the target segment



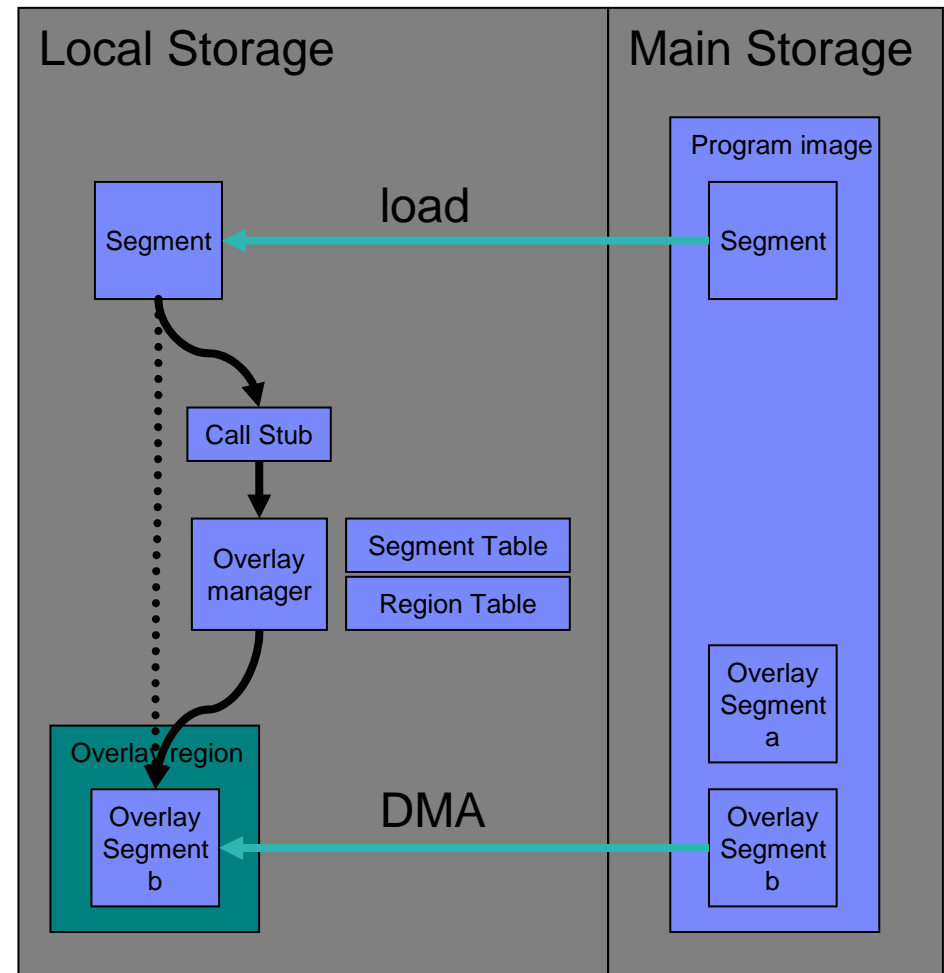
## How overlays work

- The linker generates call stubs and associated tables for overlay management
- The linker prepares the required segments so that they may be loaded when needed during execution of the program, and also adds supporting code from the overlay manager library
- Function calls in overlay segments are replaced by branches to call stubs to call the overlay manager



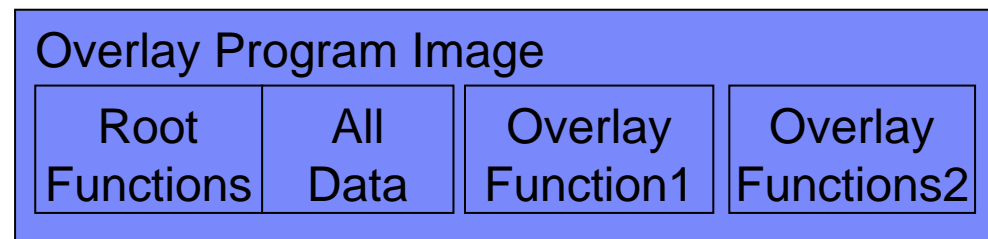
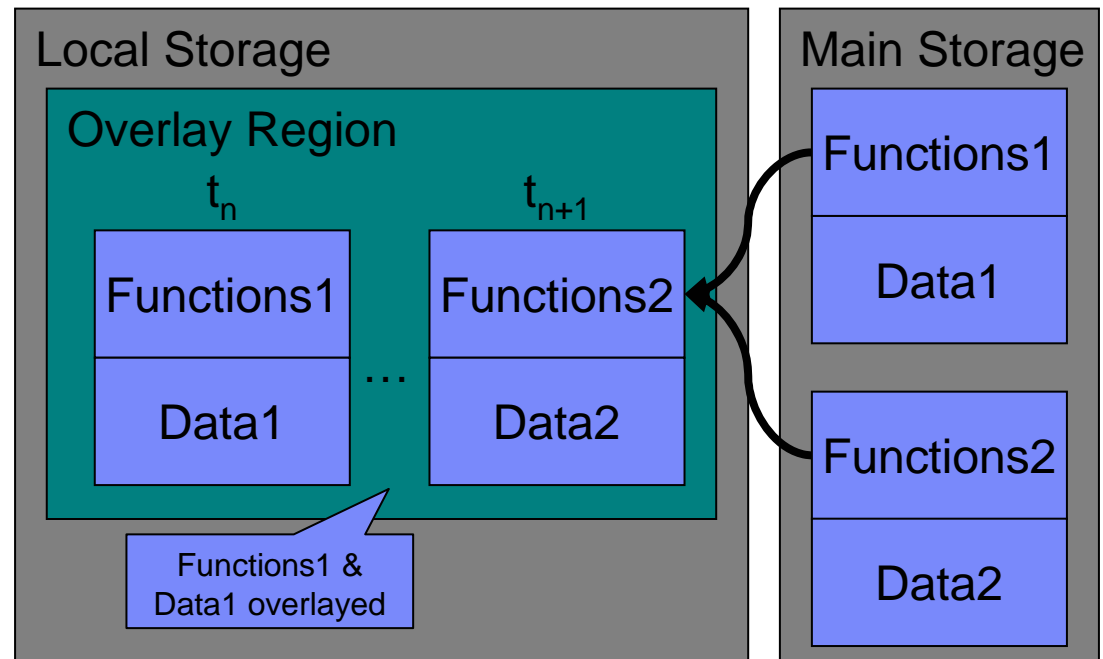
## How overlays work - 2

- At execution time when a call is made from an executing segment to another segment the system determines from the overlay tables whether the requested segment is already in local storage
- If it is not, this segment is loaded dynamically (this is carried out by a DMA), and overlays other segment which had been loaded previously
- Then the overlay manager branches to the target function



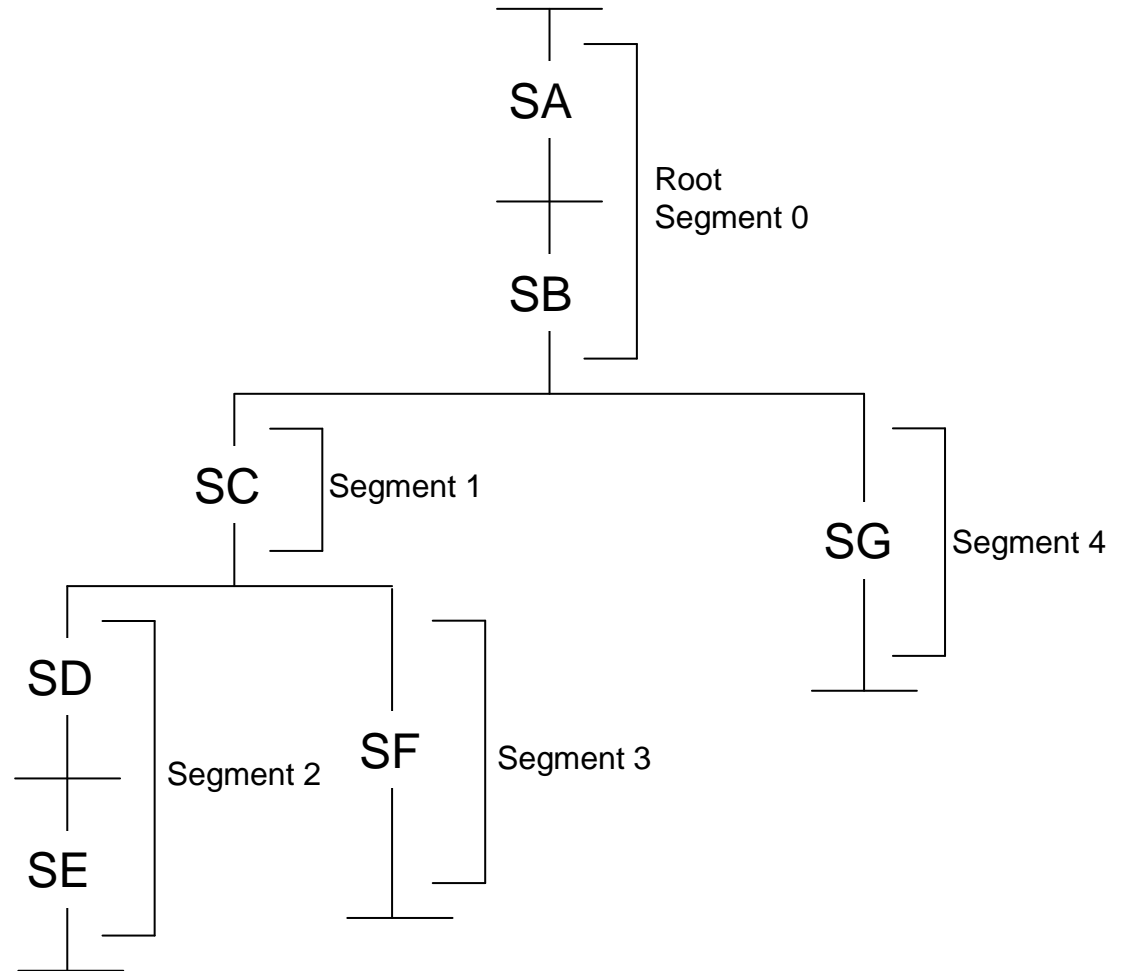
# Restrictions on the use of overlays

- When using overlays you must consider the scope of data very carefully
- It is a widespread practice to group together program sections and the data sections used by them
- If these are located in an overlay region the data can only be used transiently - overlay sections are not 'swapped out' (written back to main storage) but are replaced entirely by other overlays
- Ideally all data sections are kept in the root segment which is never overlaid
- If the data size is too large for this then sections for transient data may be included in overlay regions, but the implications of this must be carefully considered



# Overlay Tree Structure example

- Suppose that a program contains seven functions which are labeled SA through SG, and that the total length of these exceeds the amount of local storage available
- The relationship between the functions and their segments can be shown with a tree structure
- This graphically shows how segments can use local storage at different times
- It does not imply the order of execution (although the root segment is always the first to receive control)

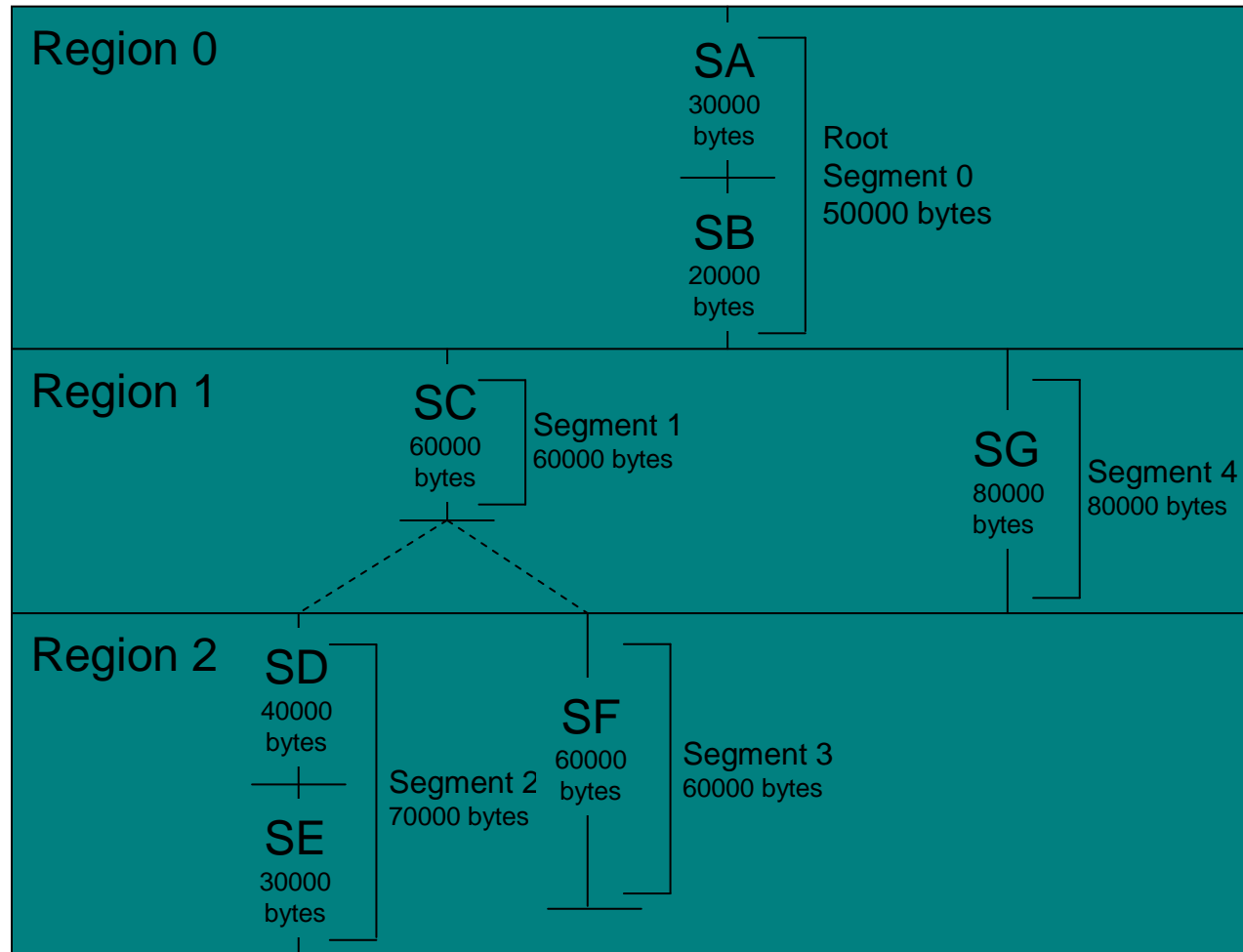


## Length of an overlay program

- If the program did not use overlays it would require 320,000 bytes of local storage; the sum of all sections
- With overlays, however, the storage needed for the program is the sum of all overlay regions, where the size of each region is the size of its largest segment
- In this structure the maximum is formed by segments 0, 4, and 2; these being the largest segments in regions 0, 1, and 2
- The sum of the regions is then 200,000 bytes

Section	Length (in bytes)
SA	30,000
SB	20,000
SC	60,000
SD	40,000
SE	30,000
SF	60,000
SG	80,000

# Overlay Tree Structure example with lengths



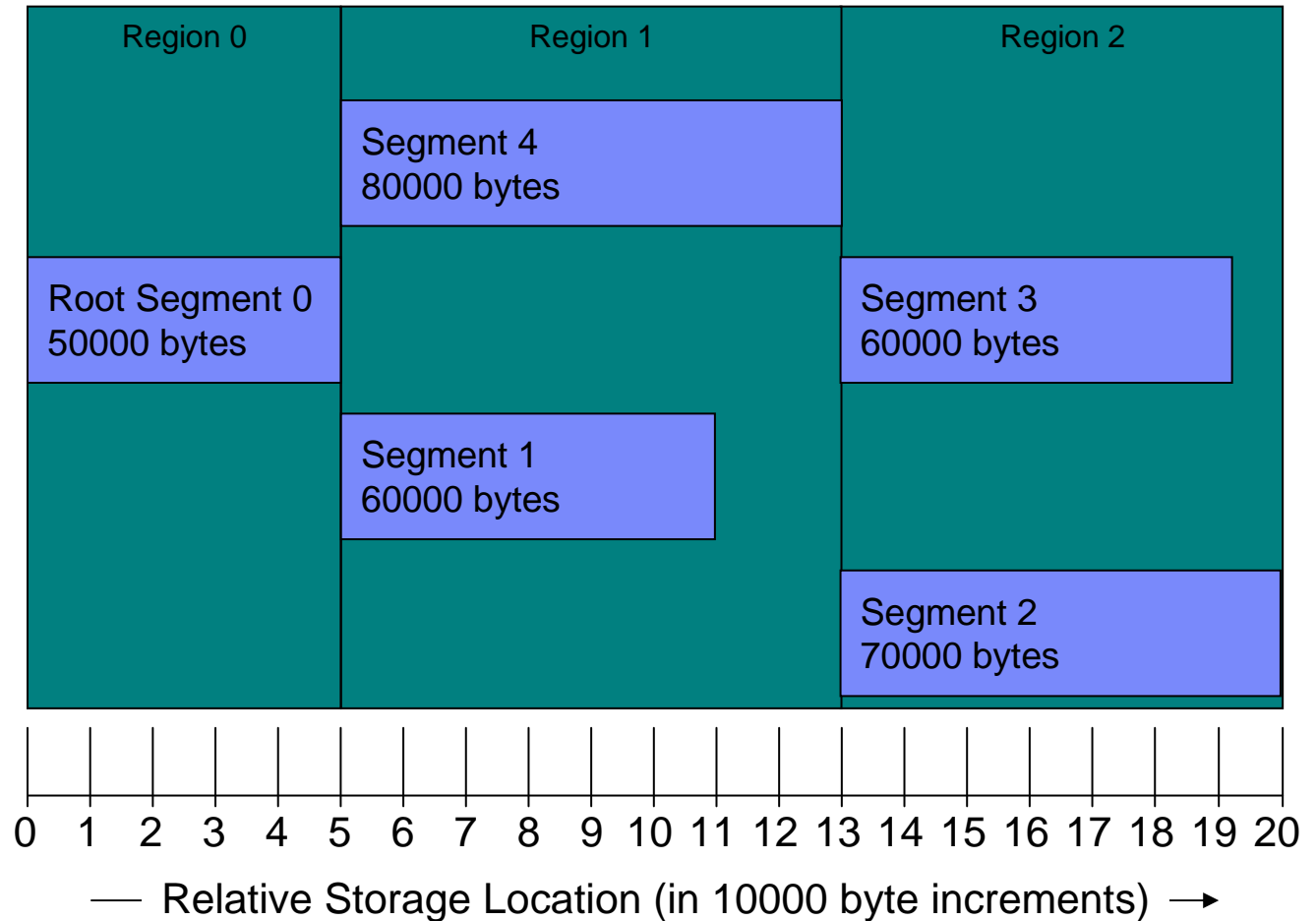
## Segment origin

- The linker typically assigns the origin of the root segment (the origin of the program) to address 0x80
- The relative origin of each segment is determined by the length of all previously defined regions
- For example, the origin of segments 2 and 3 is equal to the root origin plus 80,000 (the length of region 1 and segment 4) plus 50,000 (the length of the root segment), or 0x80 plus 130,000

Section	Origin (add 0x80)
0	0
1	50,000
2	130,000
3	130,000
4	50,000

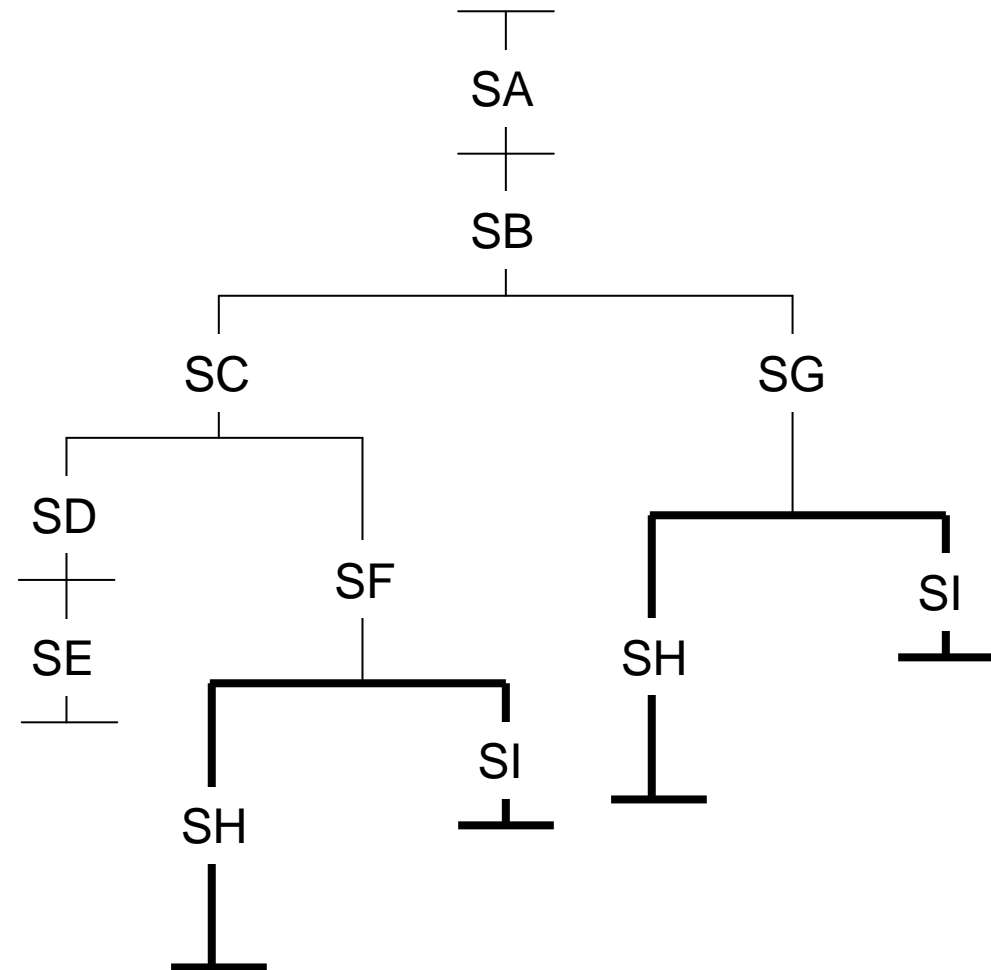
## Segment origin - 2

- The segment origin is also called the load point, because it is the relative location where the segment is loaded
- The vertical bars indicate segment origin; two segments with the same origin use the same storage area
- The longest path is that for segments 0, 4, and 2



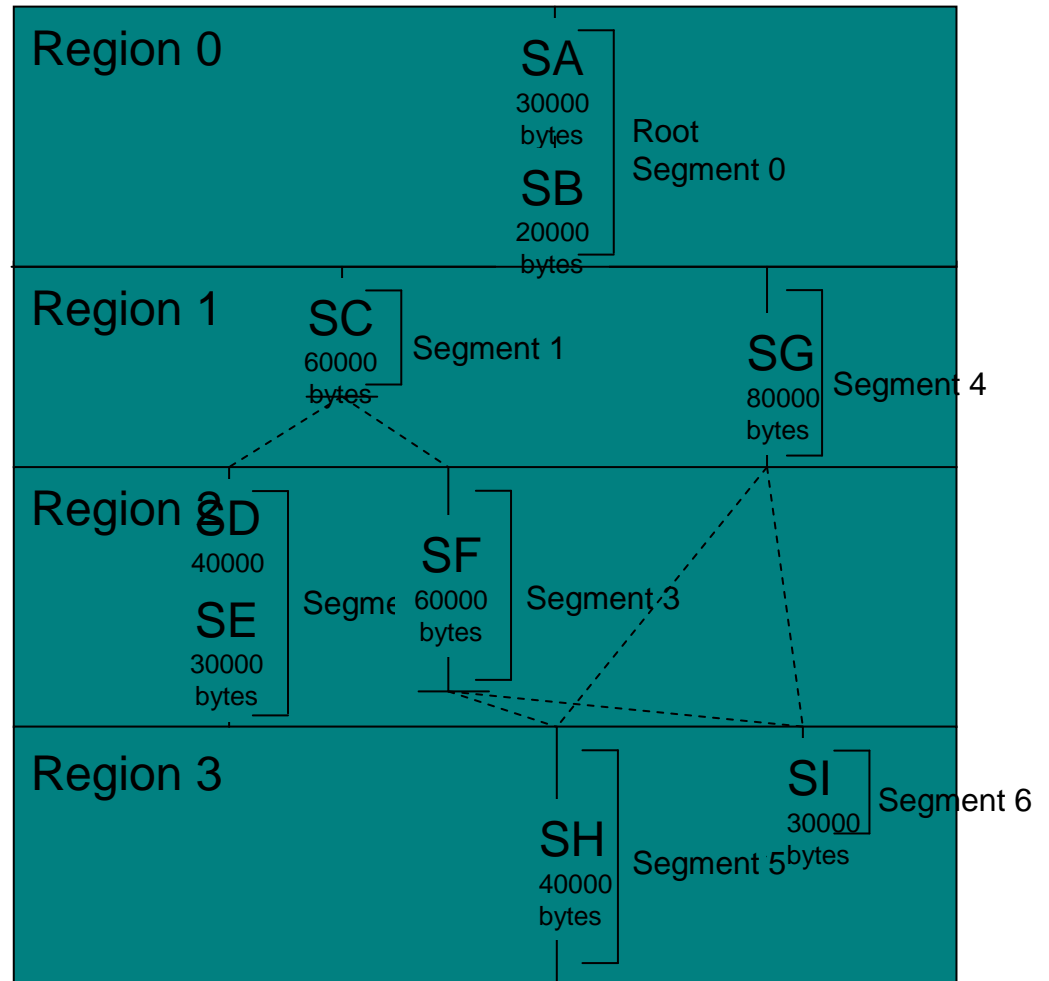
# Overlay Graph Structure example

- Adding new sections in the sample program: SH and SI
- The two new sections are each used by two other sections in different segments
- Placing SH and SI in the root segment makes the root segment larger than necessary, because SH and SI can overlay each other



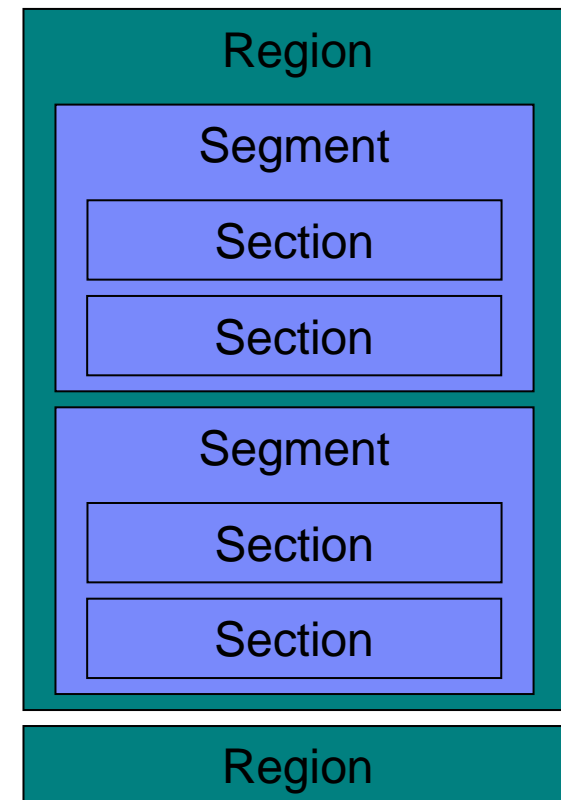
# Overlay Graph Structure example with lengths

- The relative origin of region 3 is determined by the length of the preceding regions (200,000 bytes)
- Region 3, therefore, begins at the origin plus 200,000 bytes
- The local storage required for the program is determined by adding the lengths of the longest segment in each region
- If SH is 40,000 bytes and SI is 30,000 bytes the storage required is 240,000 bytes plus the storage required by the overlay manager, its call stubs and its overlay tables



# Specification of an SPU overlay program

- Once you have designed an overlay structure, the program must be arranged into that structure
- You must indicate to the linker the relative positions of the regions, the segments, and the sections in each segment, by using OVERLAY statements
- Positioning is accomplished as follows:
  - **Regions** Are defined by each OVERLAY statement. Each OVERLAY statement begins a new region
  - **Segments** Are defined within an OVERLAY statement. Each segment statement within an overlay statement defines a new segment. In addition, it provides a means to equate each load point with a unique symbolic name
  - **Sections** Are positioned in the segment specified by the segment statement with which they are associated



## Linker input script sequence

- The input sequence of control statements and sections should reflect the sequence of the segments in the overlay structure, region by region, from top to bottom and from left to right
- The origin of every region is specified with an OVERLAY statement
- Each OVERLAY statement defines a load point at the end of the previous region
- That load point is logically assigned a relative address at the quadword boundary that follows the last byte of the largest segment in the preceding region
- Subsequent segments defined in the same region have their origin at the same load point

# Overlay Tree Structure linker script

- In this sample overlay tree program, two load points are assigned to the origins of the two OVERLAY statements and their regions. Segments 1 and 4 are at the first load point; segments 2 and 3 are at the second load point

- The linker flags used are:

```
- LDFLAGS = -Wl,-T,linker.script
```

- The following sequence of linker script statements results in the structure in

```
.text : { *( EXCLUDE_FILE(./sc.o ./sd.o ./se.o ./sf.o ./sg.o ./sh.o ./si.o) .text .stub  
  .text.* .gnu.linkonce.t.*) crt1.o*(.gnu.warning) }
```

```
OVERLAY {
```

```
  - .segment1 {./sc.o(.text)}
```

```
  - .segment4 {./sg.o(.text)}
```

```
}
```

```
OVERLAY {
```

```
  - .segment2 {./sd.o(.text) ./se.o(.text)}
```

```
  - .segment3 {./sf.o(.text)}
```

```
}
```

- Note: By implication sections SA and SB are associated with the root segment since they are NOT specified in the OVERLAY statements

# Overlay Graph Structure linker script

- In the sample overlay graph program, one more load point is assigned to the origin of the last OVERLAY statement and its region
- Segments 5 and 6 are at the third load point
- The following linker script statements add to the sequence for the overlay tree program creating the structure:

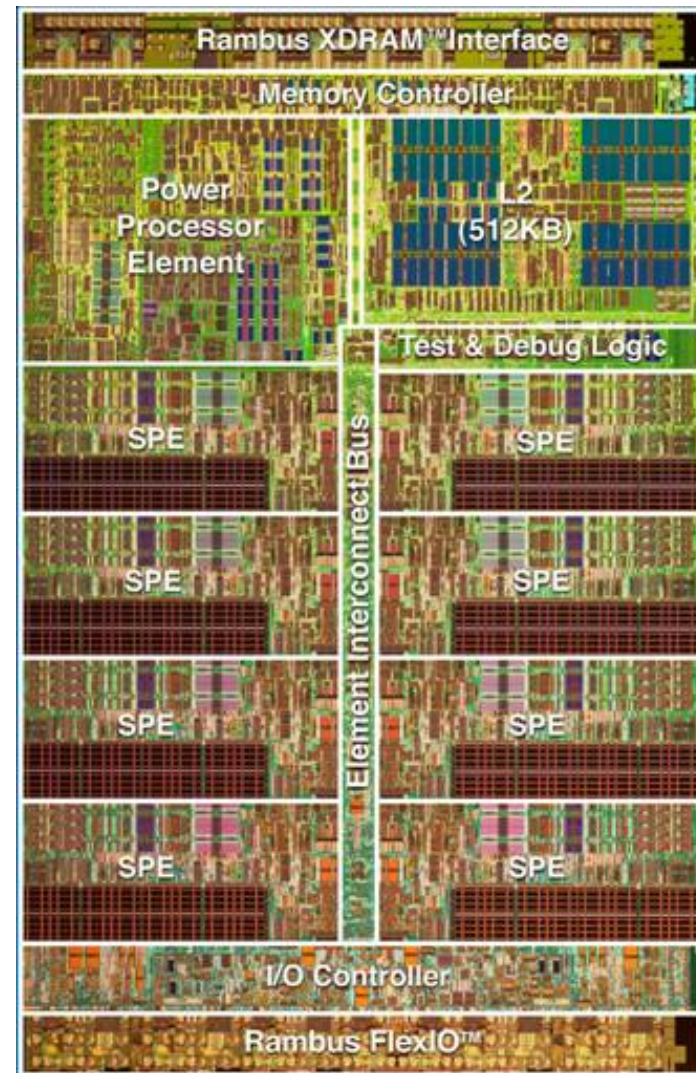
```
- ...  
- OVERLAY {  
  - .segment5 {./si.o(.text)}  
  - .segment6 {./sh.o(.text)}  
- }
```

# Summary

- SPU Overlay facility allows the user to fit large programs into the SPU local storage
- A linker script is used to control the specification of overlay regions, segments and sections
- No source editing or re-compilation is required
- The linker incorporates an overlay manager, call stubs, segment table, and region table into the resulting executable
- During execution all intersegment calls are dispatched through call stubs to the overlay manager
- The overlay manager controls the loading of segments into regions and the invocation of the target function and its return
- Developers must understand what and how data is located, accessed and updated

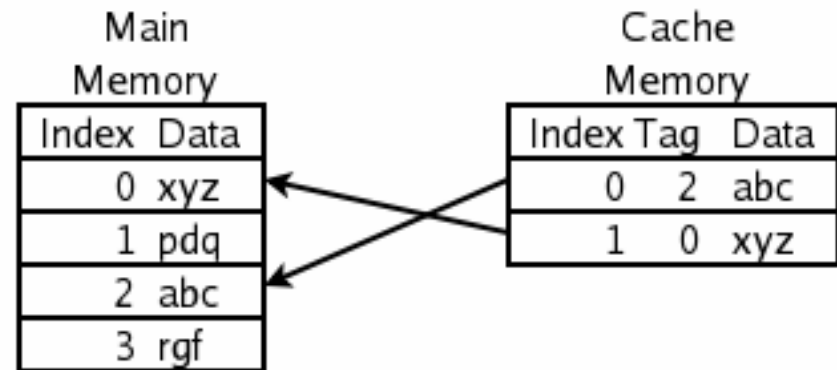
# Contents

- References
- Cell SDK
- SPE Code Overlays
- **SPE Software Cache**
- SPU Timing Tool
- DaCS and ALF
- Q & A



# Basic Cache Concept

- **Cache is a temporary storage area where frequently accessed data can be stored for rapid access.**
- **Cache components**
  - Has a pool of entries.
  - Each entry contains a datum which is a copy of the datum in some main memory.
  - Each entry also has a tag, which specifies the identity of the datum in the memory of which the entry is a copy.
- **Cache hit/miss**
  - When you want to access a datum presumably in the memory, you first check the cache. If an entry can be found with a tag matching that of the desired datum, the datum in the entry is used instead. This situation is known as a cache hit. Otherwise, it is a miss.



# Cache to System Memory Mapping

- Cache is usually organized as a block of cache lines where each cache line contains a number of bytes
- Direct Mapping: Map each memory chunk to a cache line
  - For example, if you have 4K cache lines and 64MB of main memory, then you would have  $64\text{MB}/4\text{KB} = 16\text{KB}$  chunks
  - Each cache line would be shared by 16K memory addresses (64MB divided by 4KB).
- Fully Associative Mapping – Map any memory location to a cache line
- N-Way Set Associative Mapping – Map any memory location to a set of cache lines
  - Breakup the cache into sets, typically 2, 4, 8 etc. → Each set contains "N" cache lines, e.g., 4.
  - Then, each memory address is assigned a set, and can be cached in any one of those 4 locations within the set that it is assigned to. In other words, within each set the cache is associative, and thus the name.

# Motivation

- SPE memory accesses are to Local Store Addresses only
  - This represents a new programming model
- Software cache has many benefits in SPE environment
  - Simplifies programming model
    - familiar load/store effective address model can be used
    - Decreases time to port to SPE
  - Take advantage of locality of reference
  - Can be easily optimized to match data access patterns
  - Provides ability to apply sophisticated replacement algorithms such as those developed for virtual-memory paging

# Cache Implementation – Defining a Cache

- Implemented as macros and inline functions a header file
  - `SDK/usr/include/spu/cache-api.h`
- User simply `#defines` cache attributes and includes the header
  - Name
  - Associativity (current support for direct mapped or 4-way set associative)
  - Line size (16B – 16KB)
  - ReadOnly or ReadWrite
  - Type of data object to cache (power of 2 size)
  - Number of sets
- User can define multiple caches by re-defining attributes and re-including the header
- Data mapped to EA space is accessed “directly” (through cache)

# Cache Attributes

Symbol	Description	Possible Values	Default Value
CACHED_TYPE	Specifies the type of data being cached.	any valid data type	none (required)
CACHE_NAME	Specifies the unique string associated with the cache.	any	none (required)
CACHELINE_LOG2SIZE	Specifies the log base 2 of cache line size in bytes.	4-12	7 (128 bytes)
CACHE_LOG2NSETS	Specifies the log base 2 of the number of sets in the cache.	0-12	6 (64 sets)
CACHE_LOG2NWAY	Specifies the associativity of the cache.	0 or 2	2 (4-way)
CACHE_TYPE	Specifies the type of cache. 0 specifies that the cache is a read only cache, 1 specifies that the cache is read/write.	0 or 1	1 (read/write)
CACHE_SET_TAGID(set)	Specifies the tag ID for the given set.	any (return <= 31)	set & 0x1F
CACHE_READ_X4	When the cached type is an integral type that fits in a 32-bit word, this define enables the <code>cache_rd_x4()</code> service which caches and returns four data items at a time in a <code>vec_uint4</code> .	-	not defined
CACHE_STATS	When this is defined, the cache code maintains metrics on cache activity. These can be displayed by calling the <code>cache_pr_stats()</code> service from within the program using the cache.	-	not defined

## Example: Defining a Cache

```
#define CACHE_NAME          my_cache
#define CACHED_TYPE        my_type_t
#define CACHE_LOG2NWAY     2      /* 4-way */
#define CACHE_LOG2NSETS   3      /* 8 sets */
#define CACHE_TYPE         1      /* RW */
#define CACHELINE_LOG2SIZE 9      /* 512b */
#define CACHE_STATS        /* enable stats */
#include <cache-api.h>
```

# Cache Implementation - Interfaces

- `cache_rd(name, eaddr)`
  - Read value @ `eaddr`, cache it, and return value
- `cache_wr(name, eaddr, val)`
  - Write value to `eaddr` into cache (no writethrough)
- `cache_touch(name, eaddr)`
  - Read value @ `eaddr` into cache, return `Isa` pointer (non-blocking)
- `cache_wait(name, isa)`
  - Wait for touched data @ `isa` to arrive in the cache
- `cache_rw(name, eaddr)`
  - Read (with intent to write) value @ `eaddr`, cache it, and return `Isa` pointer

# Cache Implementation - Interfaces

- `cache_lock(name, lsa)`
  - Lock data @ lsa in the cache
- `cache_unlock(name, lsa)`
  - Unlock data @ lsa from the cache
- `cache_flush(name)`
  - Force writeback of all dirty lines
- `cache_rd_x4(name, eaddr4)`
  - Special purpose interface to cache and return 4 uints in a vector
- `cache_pr_stats(name)`
  - Display cache stats

# Cache Implementation - Algorithms

- Replacement policy
  - User-defined by providing `__cache_replace_idx()` service
  - Default is round-robin (FIFO)
- Set mapping
  - User-defined by providing `__cache_set_num()` service
  - Ability to tune algorithm to reduce set collisions
- Tag ID
  - User-defined by providing `CACHE_SET_TAGID(set)` service
  - Default is `(set & 0x1F)`

## Cache Implementation - Algorithms

- **Write policy is “write-back”**
- **Asynchronous cache touch service allows for latency hiding**
- **Well-known optimizations such as prefetching and write buffering can be easily added to further reduce memory stalls**
- **Ability to select topology and algorithms is a big advantage of software vs. hardware cache**

# Programming Notes

- Only `CACHED_TYPE` and `CACHE_NAME` are required to be defined by the programmer
- It is recommended to use a previously typedef-ed type for `CACHED_TYPE` to avoid any potential operator precedence issues that might arise
- Must include the SPE cache header files in order to call the cache interfaces
- Multiple caches on local memory may be defined by re-including the cache header files.
- `CACHE_NAME` can be any string that would be suitable for a C function name, and is the same string which must be used to reference the cache using the supported interfaces.

# Cache Example 1

## Using the cache to swap two values (safe interfaces):

```
a = cache_rd(my_items, eaddr_a);
```

The *cache\_rd* service reads from the cache, specified by *my\_items*, the data from the specified 32-bit effective address *eaddr\_a*.

```
b = cache_rd(my_items, eaddr_b);
```

The *cache\_rd* service reads from the cache, specified by *my\_items*, the data from the specified 32-bit effective address *eaddr\_b*.

```
cache_wr(my_items, eaddr_b, a);
```

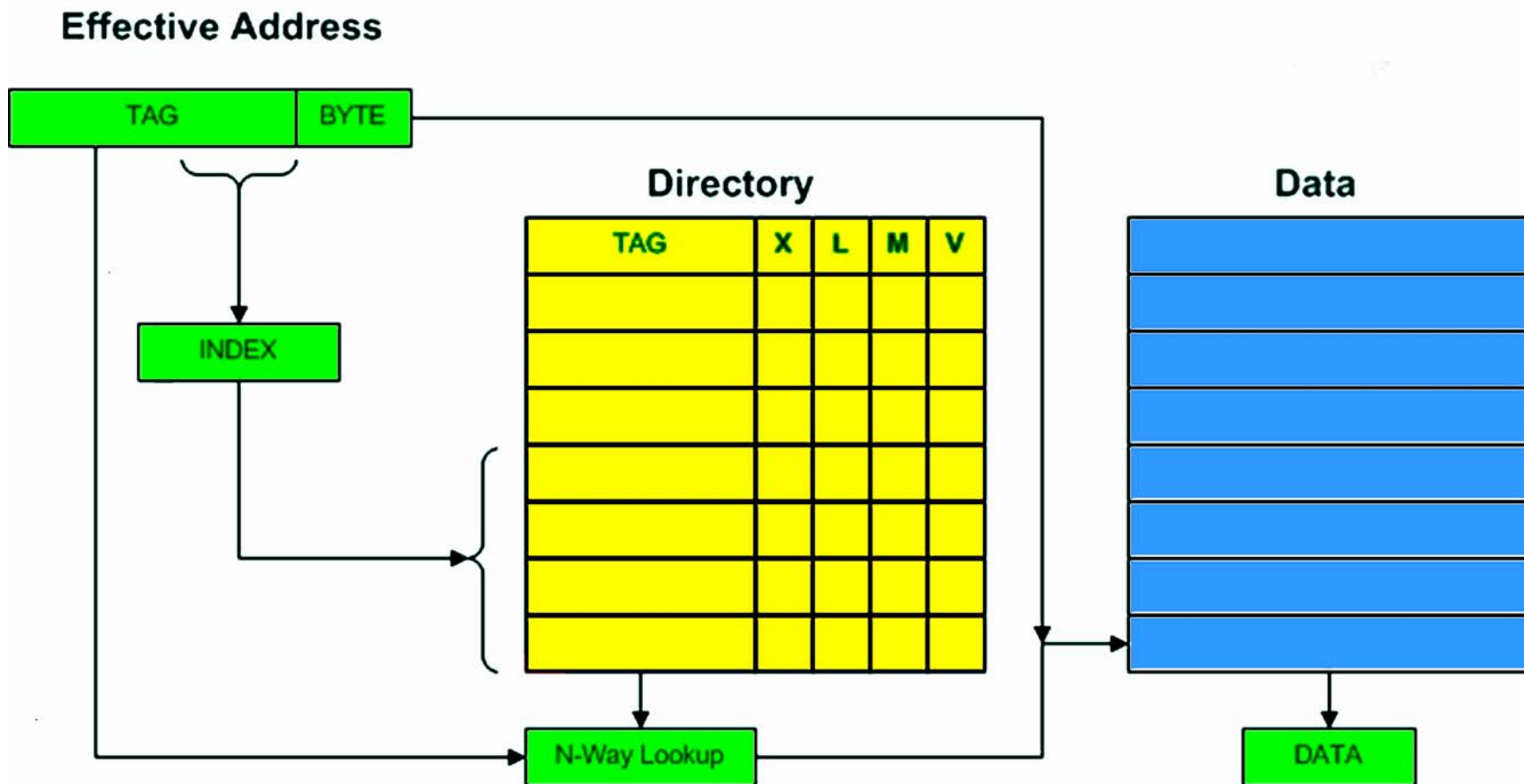
The *cache\_wr* service writes the value *a* to the cache specified by *my\_items*, to the specified 32-bit effective address *eaddr\_b*.

```
cache_wr(my_items, eaddr_a, b);
```

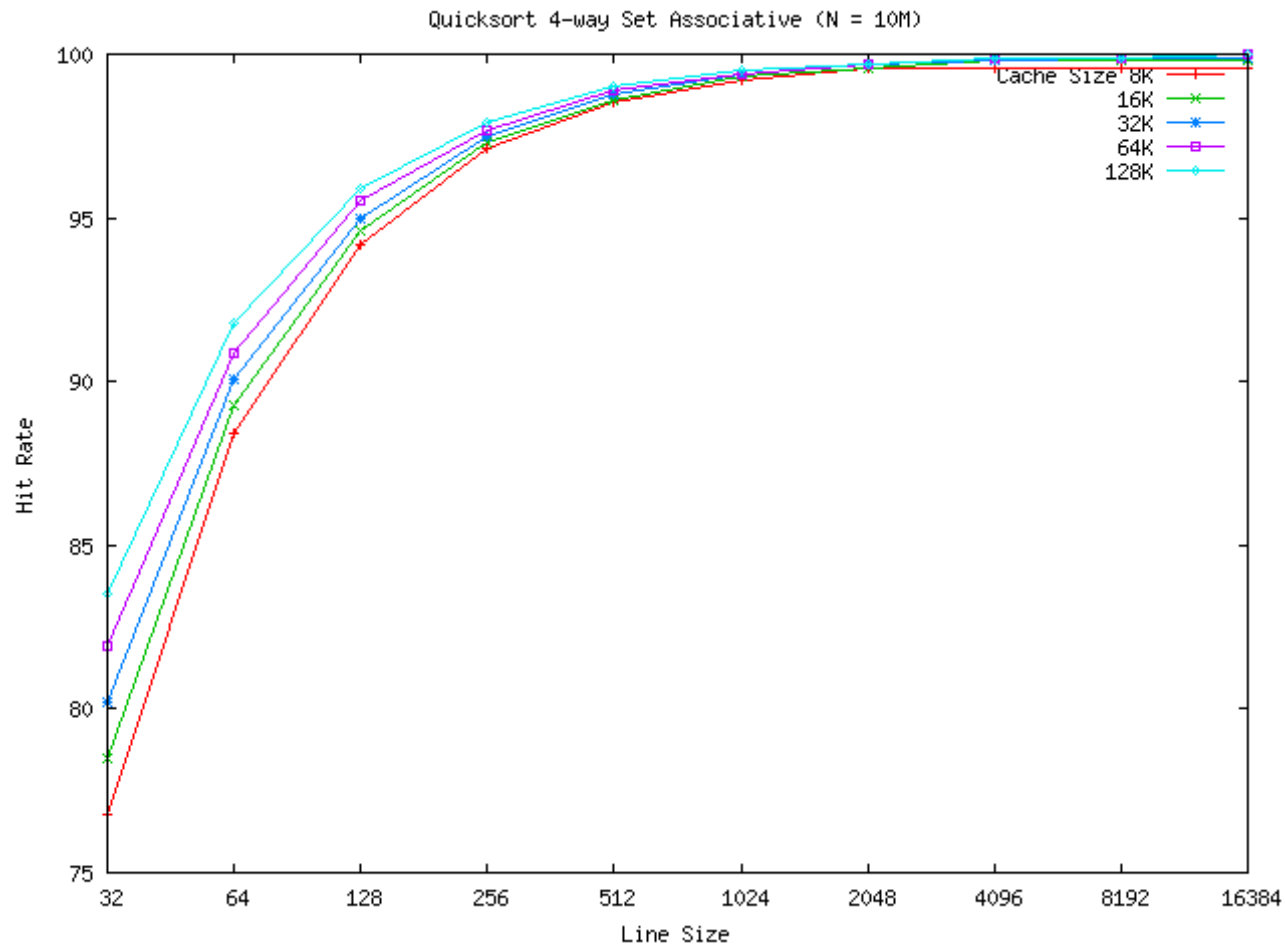
The *cache\_wr* service writes the value *b* to the cache specified by *my\_items*, to the specified 32-bit effective address *eaddr\_a*.

# Cache Implementation – Topology

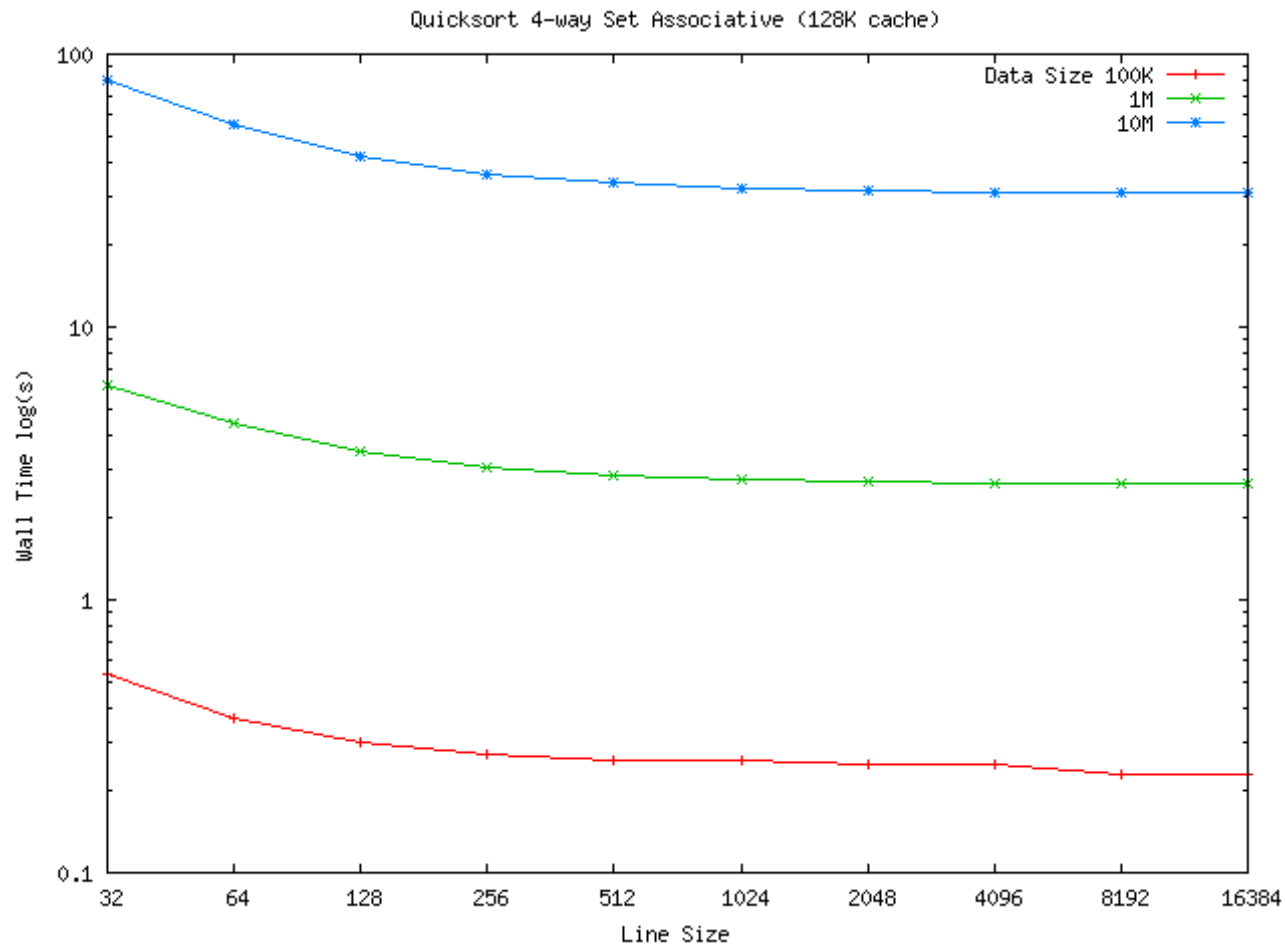
X (unused) L = locked M = modified V = valid



# Analysis: Quicksort – hit rate vs. line size



# Analysis: Quicksort – run time vs. line size



# Quicksort: Cache Stats

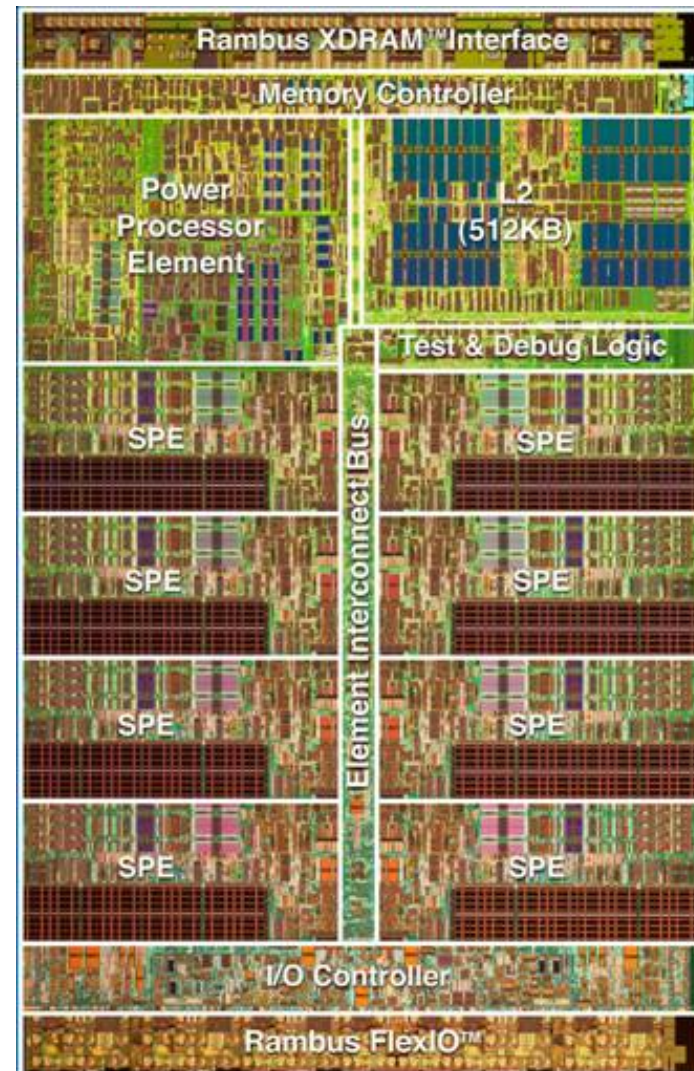
name=qsort type=READ\_WRITE nway=4 nsets=16 linesz=128 totsz=8192

SET	READ			WRITE			REPL	WRBK
	HIT	MISS	HITPCT	HIT	MISS	HITPCT		
0	6640754	386910	94.5%	176390	15625	91.9%	402535	386913
1	6633135	387337	94.5%	176437	15625	91.9%	402962	387340
2	6656198	387905	94.5%	176259	15625	91.9%	403530	387909
3	6634743	387710	94.5%	176222	15625	91.9%	403335	387713
4	6653756	388817	94.5%	176321	15625	91.9%	404442	388821
5	6649381	388691	94.5%	176231	15625	91.9%	404316	388695
6	6649961	388223	94.5%	176211	15625	91.9%	403848	388227
7	6641903	388528	94.5%	176551	15625	91.9%	404153	388532
TOT	106416161	6208013	94.5%	2821700	250000	91.9%	6458013	6208074

Wall Time: 9.009s

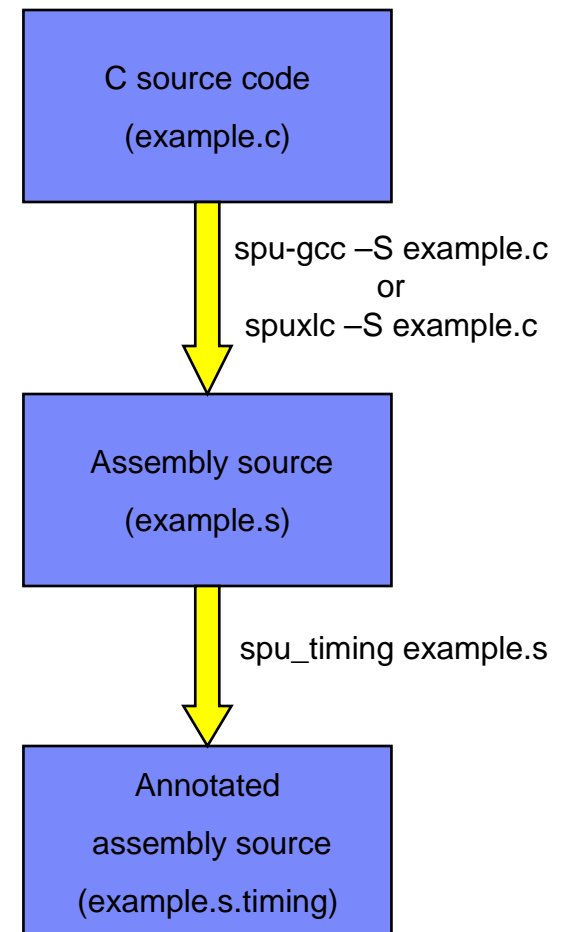
# Contents

- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



# What is the SPU timing tool

- Annotates an assembly source file with static analysis of instruction timing assuming linear (branchless) execution.
  - Simplified pipeline model
  - Does not account for:
    - instruction fetch stalls
    - local store contention
    - branching
  - Supports Cell SDK 2.0 SPU models



# Invocation Syntax

`spu_timing [options ...] [input_file]`

- options:
  - `--help` displays a verbose help screen.
  - `-march=<cpu>` specifies the target architecture. `<cpu>` is either *cell* or *cell\_edp*
  - `-o <file>` specifies the output file. Default is `<input_file>.timing` or stdout if no input file is specified.
  - `-running-count` include column of running cycles counts for start of each instruction.
- `<input_file>` specifies the assembly input file. If not specified, *spu\_timing* sources its input from stdin.

## Sample - C source (example.c)

```
#include <spu_intrinsics.h>

// Compute  $y = \alpha * x + y$ , where alpha is a scalar and x
// and y are 4*n element vectors.

void saxpy(int n, float alpha, vec_float4 x[], vec_float4 y[])
{
    int i;
    vec_float4 a;

    a = spu_splats(alpha);

    for (i=0; i<n; i++) {
        y[i] = spu_madd(a, x[i], y[i]);
    }
}
```

# Sample - assembly source (example.s)

```
.file      "example.c"
.text
    .align  3
    .global saxpy
    .type   saxpy, @function
saxpy:
    ila     $2,66051
    shlqbyi $7,$3,0
    cgti    $3,$3,0
    shufb   $8,$4,$4,$2
    nop     $127
    biz     $3,$1r
    ori     $4,$7,0
    hbra    .L8,.L4
    il      $7,0
    lnop
.L4:
    ai      $4,$4,-1
    lqx     $2,$7,$5
    lqx     $3,$7,$6
    fma     $2,$8,$2,$3
    stqx    $2,$7,$6
    ai      $7,$7,16
.L8:
    brnz    $4,.L4
    bi      $1r
```

# Sample – annotated source (example.s.timing)

```
000000 0D 01
000000 1D 0123
000001 0d 12
000002 1d -2345
000003 0D 3
000003 1D 3456
000004 0D 45
000004 1D 456789
000005 0D 56
000005 1D 5

000006 0d 67
000007 1d -789012
000008 1 890123
000014 0 -----456789
000020 1 -----012345

000022 1 2345
000023 1 3456
```

```
.file "example.c"
.text
.align 3
.global saxpy
.type saxpy, @function
saxpy:
ila $2,66051
shlqbyi $7,$3,0
cgti $3,$3,0
shufb $8,$4,$4,$2
nop $127
biz $3,$1r
ori $4,$7,0
hbra .L8,.L4
il $7,0
lnop

.L4:
ai $4,$4,-1
lqx $2,$7,$5
lqx $3,$7,$6
fma $2,$8,$2,$3
stqx $2,$7,$6

.L8:
brnz $4,.L4
bi $1r
```

# Sample – annotated source (example.s.timing)

**running-count** – cycle count for which each instruction starts. Useful for determining the cycles in a loop. For example, our loop is 17 cycles (23-6).

```
000000 0D 01
000000 1D 0123
000001 0d 12
000002 1d -2345
000003 0D 3
000003 1D 3456
000004 0D 45
000004 1D 456789
000005 0D 56
000005 1D 5

000006 0d 67
000007 1d -789012
000008 1 890123
000014 0 -----456789
000020 1 -----012345

000022 1 2345
000023 1 3456
```

```
.file "example.c"
.text
.align 3
.global saxpy
.type saxpy, @function
saxpy:
    ila    $2,66051
    shlqbyi $7,$3,0
    cgti   $3,$3,0
    shufb  $8,$4,$4,$2
    nop    $127
    biz    $3,$1r
    ori    $4,$7,0
    hbra   .L8,.L4
    il     $7,0
    lnop

.L4:
    ai     $4,$4,-1
    lqx    $2,$7,$5
    lqx    $3,$7,$6
    fma    $2,$8,$2,$3
    stqx   $2,$7,$6

.L8:
    brnz   $4,.L4
    bi     $1r
```

# Sample – annotated source (example.s.timing)

**Execution pipeline** – the pipeline in which the instruction is issued. Either 0 (even pipeline) or 1 (odd pipeline).

```
000000 00 01
000000 10 0123
000001 00 12
000002 10 -2345
000003 00 3
000003 10 3456
000004 00 45
000004 10 456789
000005 00 56
000005 10 5

000006 00 67
000007 10 -789012
000008 1 890123
000014 0 -----456789
000020 1 -----012345

000022 1 2345
000023 1 3456
```

```
.file "example.c"
.text
.align 3
.global saxpy
.type saxpy, @function
saxpy:
    ila    $2,66051
    shlqbyi $7,$3,0
    cgti   $3,$3,0
    shufb  $8,$4,$4,$2
    nop    $127
    biz    $3,$1r
    ori    $4,$7,0
    hbra   .L8,.L4
    il     $7,0
    lnop

.L4:
    ai     $4,$4,-1
    lqx   $2,$7,$5
    lqx   $3,$7,$6
    fma   $2,$8,$2,$3
    stqx  $2,$7,$6

.L8:
    brnz  $4,.L4
    bi    $1r
```

# Sample – annotated source (example.s.timing)

**dual-issue status** – blank indicates single issue.

```

.file      "example.c"
.text
.align    3
.global   saxpy
.type     saxpy, @function
saxpy:
    ila      $2,66051
    shlobvi  $7,$3,0
    cgti     $3,$3,0
    shufb    $8,$4,$4,$2
    nop      $127
    biz      $3,$1r
    ori      $4,$7,0
    hbra     .L8,.L4
    il       $7,0
    lnop
.L4:
    ai       $4,$4,-1
    lqx      $2,$7,$5
    lqx      $3,$7,$6
    brnz     $4,.L4
    bi       $1r
    
```

```

000000 D 01
000000 D 0123
000001 d 12
000002 d -2345
000003 D 3
000003 D 3456
000004 D 45
000004 D 456789
000005 D 56
000005 D 5
000006 d 67
000007 d -789012
000008 .
000014 .
000020 .
000022 .
000023 .
    
```

**d** – indicates that for the pair of instructions, dual-issue is possible but will not occur due to a dependency stall.

**D** – indicates the pair of instructions will be dual-issued.

# Sample – annotated source (example.s.timing)

```

000000 0D 01
000000 1D 0123
000001 0d 12
000002 1d -2345
000003 0D 3
000003 1D 3456
000004 0D 45
000004 1D 456789
000005 0D 56
000005 1D 5

000006 0d 67
000007 1d -789012
000008 1 890123
000014 0 -----456789
000020 1 -----012345

000022 1 2345
000023 1 3456
    
```

**Instruction clock cycle occupancy** – A digit (0-9) is displayed for every clock cycle the instruction executes. Operand dependency stalls are flagged by a dash (“-”) for every clock cycle the instruction is expect to stall.

Steeply sloping cascading numbers signify good scheduling.

Shallow sloping (horizontal) numbers signify poor scheduling.

```

...ple.c"
...
...y, @function
...6051
...3,0
...3,0
...4,$4,$2
...lr
...7,0
...L4
...4,-1
...7,$5
...7,$6
...Lma $2,$8,$2,$3
...stqx $2,$7,$6
.L8:
...brnz $4,.L4
...bi $lr
    
```

# Sample – annotated source (example.s.timing)

**Inner Loop** – contains lots of dependency stalls.

The load of **y** stalls 1 cycle for address increment. The **fma** stalls 5 cycles waiting for the load to complete. The store of the resulting **y** stalls 5 cycles waiting for the **fma** to complete.

Dependency stalls could be eliminated by unrolling the loop. Loop unrolling could also result in moderate dual issue because the instruction mix is 1/3 pipe 0 and 2/3 pipe 1.

```
000000 0D 01
000000 1D 0123
000001 0d 12
000002 1d -2345
000003 0D 3
000003 1D 3456
000004 0D 45
000004 1D 456789
000005 0D 56
000005 1D 5
```

```

                                .L4:
000006 0d      67                                ai      $4,$4,-1
000007 1d     -789012                            lqx     $2,$7,$5
000008 1      890123                            lqx     $3,$7,$6
000014 0      -----456789                            fma     $2,$8,$2,$3
000020 1      -----012345                            stqx    $2,$7,$6
                                .L8:
000022 1      2345                                brnz    $4,.L4
000023 1      3456                                bi      $1r
```

```
"example.c"
3
saxpy
saxpy, @function

$2,66051
$7,$3,0
$3,$3,0
$8,$4,$4,$2
$127
$3,$1r
$4,$7,0
.L8,.L4
il      $7,0
lnop
```

## Useful Technique – profile markers

- For complex source code, insert profile checkpoint markers to locate specific code sections.
  - `#include <profile.h>`
  - place `prof_cp#()` function in desired locations.
  - use unique `#` for improved identification.
  - `prof_cp#` results in “`and $#, $#, $#`” instructions, where `#` is 0 – 31.
  - When using `spuxlc`, the profile checkpoints are coded as `mc_funcs`. Therefore, to locate them, search for a “.word 0x#####”, where ##### corresponds to the encode and instruction.

## Useful Technique – profile markers

```
#include <spu_intrinsics.h>
#include <profile.h>

// Compute y = alpha * x + y, where
// alpha is a scalar and x and y are
// 4*n element vectors.

void saxpy(int n, float alpha,
vec_float4 x[], vec_float4 y[])
{
    int i;
    vec_float4 a;

    a = spu_splats(alpha);

    prof_cp1();
    for (i=0; i<n; i++) {
        y[i] = spu_madd(a, x[i], y[i]);
    }
    prof_cp2();
}
```

saxpy:

```
.align    3
.global   saxpy
.type     saxpy, @function

ila      $2,66051
shlqbyi  $7,$3,0
and      $1,$1,$1; lnop
cgti     $3,$3,0
shufb    $8,$4,$4,$2
nop      $127
biz      $3,$1r
ori      $4,$7,0
hbra     .L8,.L4
il       $7,0
lnop

.L4:
ai       $4,$4,-1
lqx      $2,$7,$5
lqx      $3,$7,$6
fma      $2,$8,$2,$3
stqx     $2,$7,$6
ai       $7,$7,16

.L8:
brnz     $4,.L4
and      $2,$2,$2; lnop
bi       $1r
```

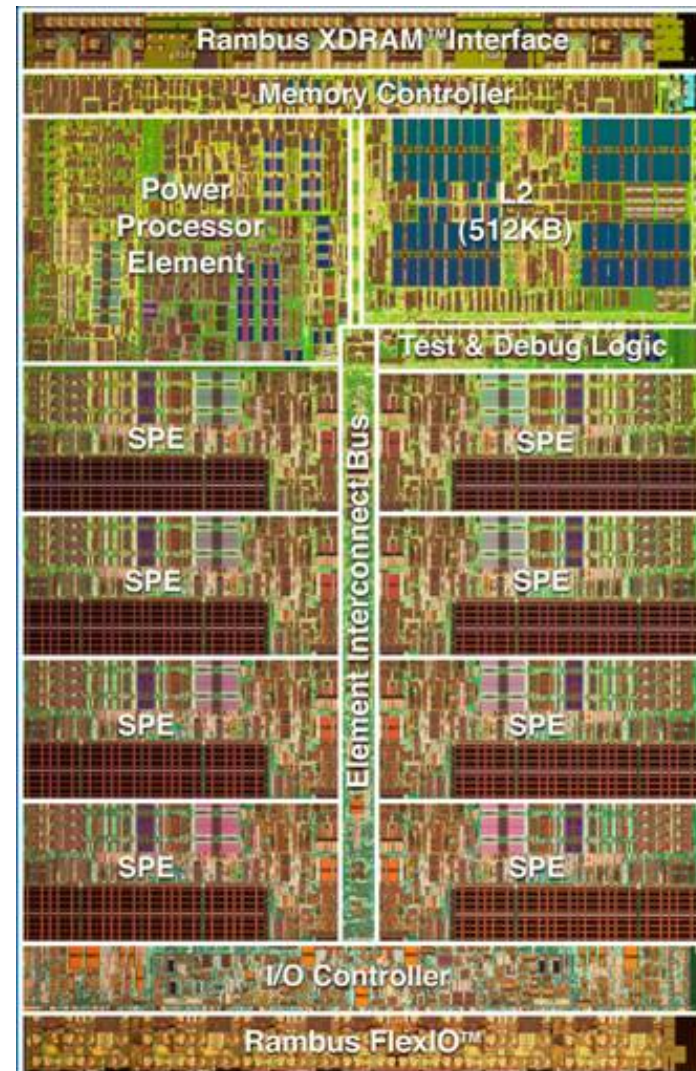
## Functional Limitations

- Does not support multiple assembly instructions per line.
- Does not support generalized expressions. An expression will terminate the assembly parser.
- Does not support symbols and symbol substitution. Can terminate the assembly parser.
- Does not support completely the .repeat assembler directive.

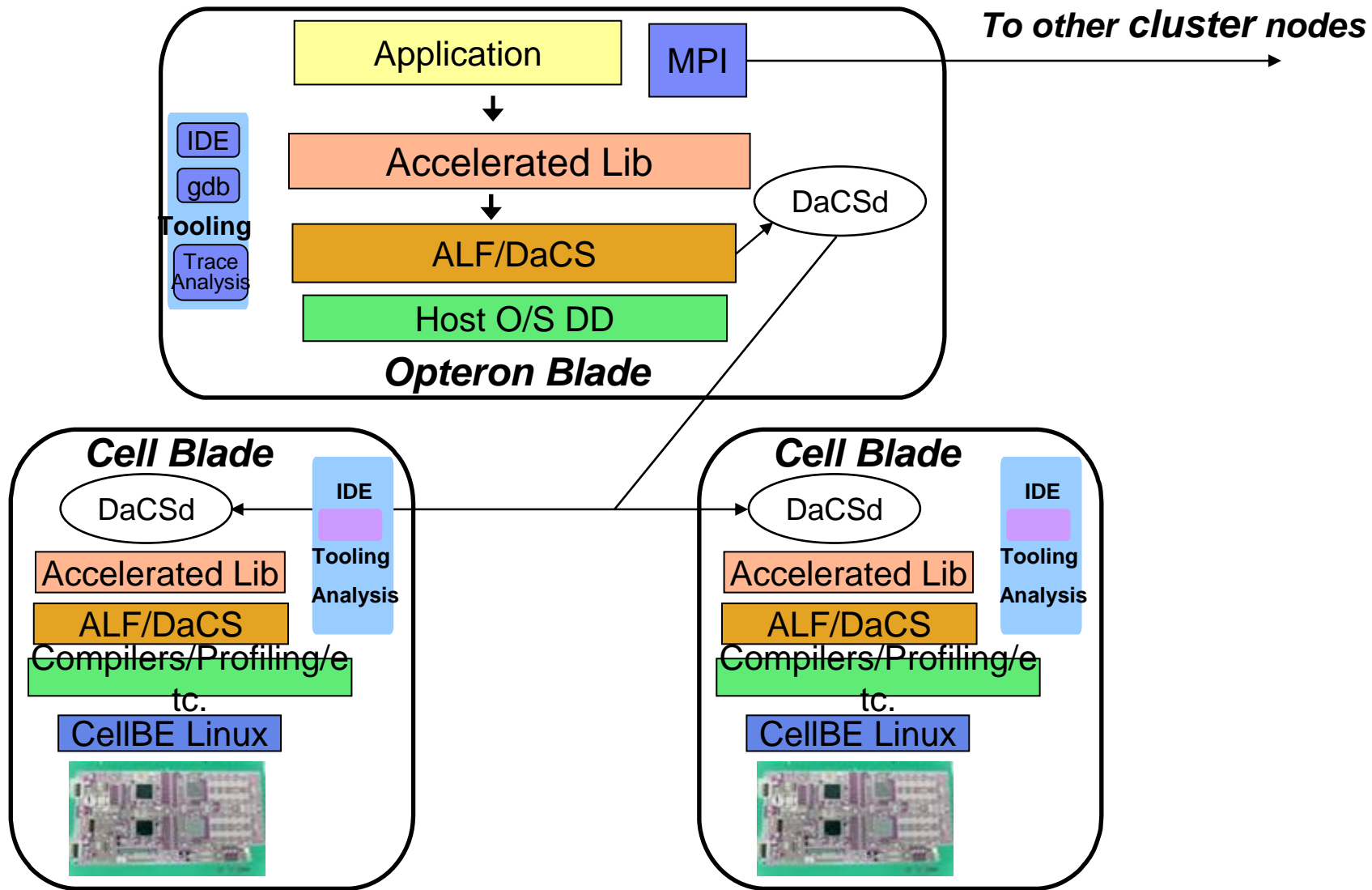
Works OK with compiled assembly, but often doesn't work for hand written assembly.

# Contents

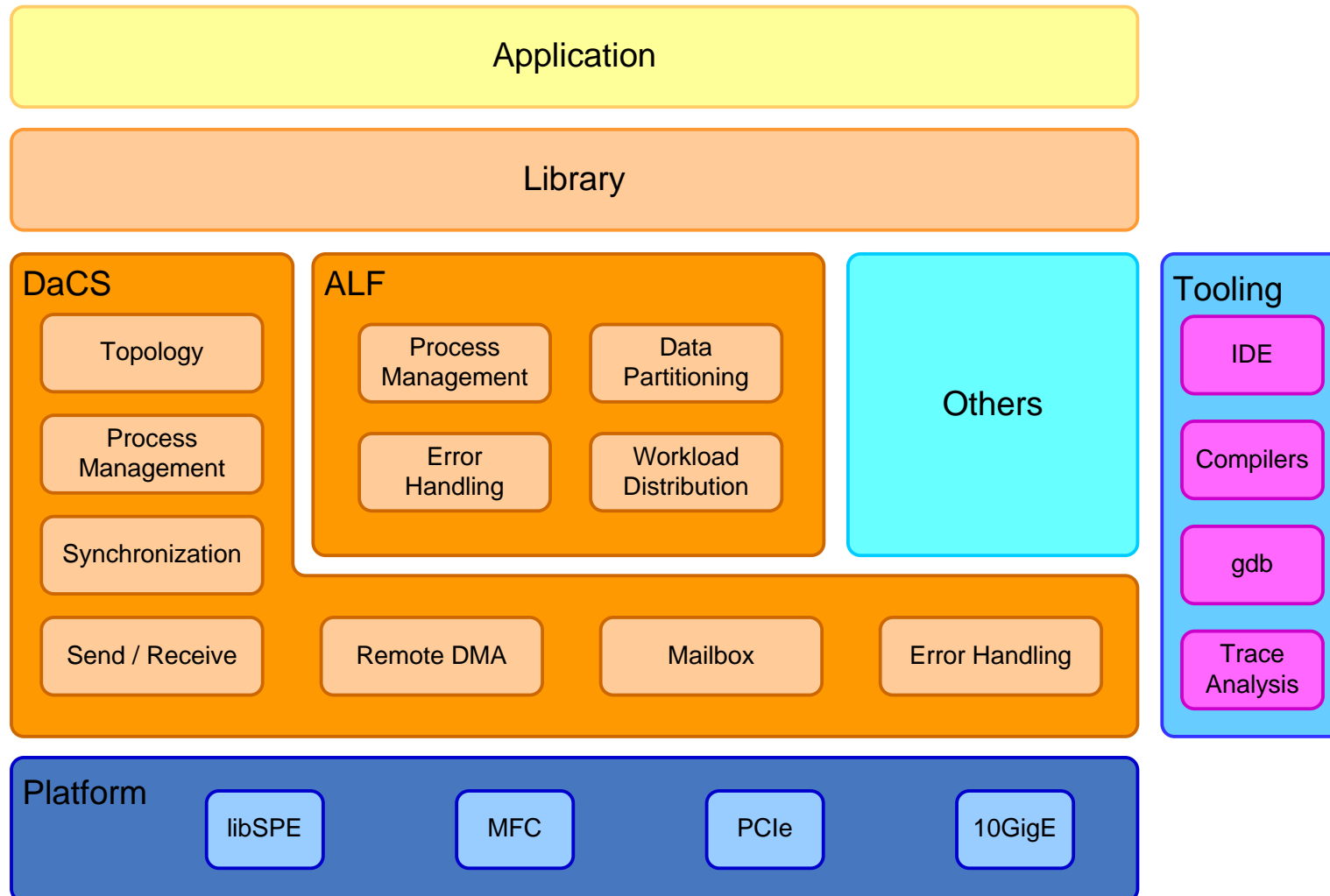
- References
- Cell SDK
- SPE Code Overlays
- SPE Software Cache
- SPU Timing Tool
- DaCS and ALF
- Q & A



# Hybrid Node System Software Stack



# ALF and DaCS: IBM's Software Enablement Strategy for Multi-core Memory-Hierarchy Systems

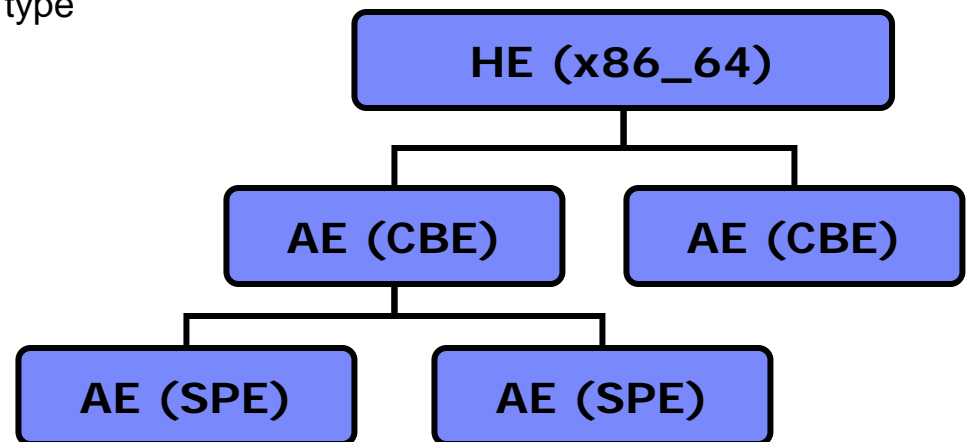


# DaCS – Data Communications and Synchronization

- Focused on data movement primitives
  - DMA like interfaces (put, get)
  - Message interfaces (send/recv)
  - Mailbox
  - Endian Conversion
- Provides Process Management, Accelerator Topology Services
- Based on Remote Memory windows and Data channels architecture
- Common API – Intra-Accelerator (CellBE), Host - Accelerator
- Double and multi-buffering
  - Efficient data transfer to maximize available bandwidth and minimize inherent latency
  - Hide complexity of asynchronous compute/communicate from developer
- Supports ALF, directly used by US National Lab for Host-Accelerator HPC environment
- Thin layer of API support on CELLBE native hardware
- Hybrid DaCS
  - Prototyped on IB Verbs (incomplete)
  - Developed on Sockets (prototype in SDK3)
  - Developed on PCI-e (Tri-blade) SDK3.0 – internal, SDK4.0 GA

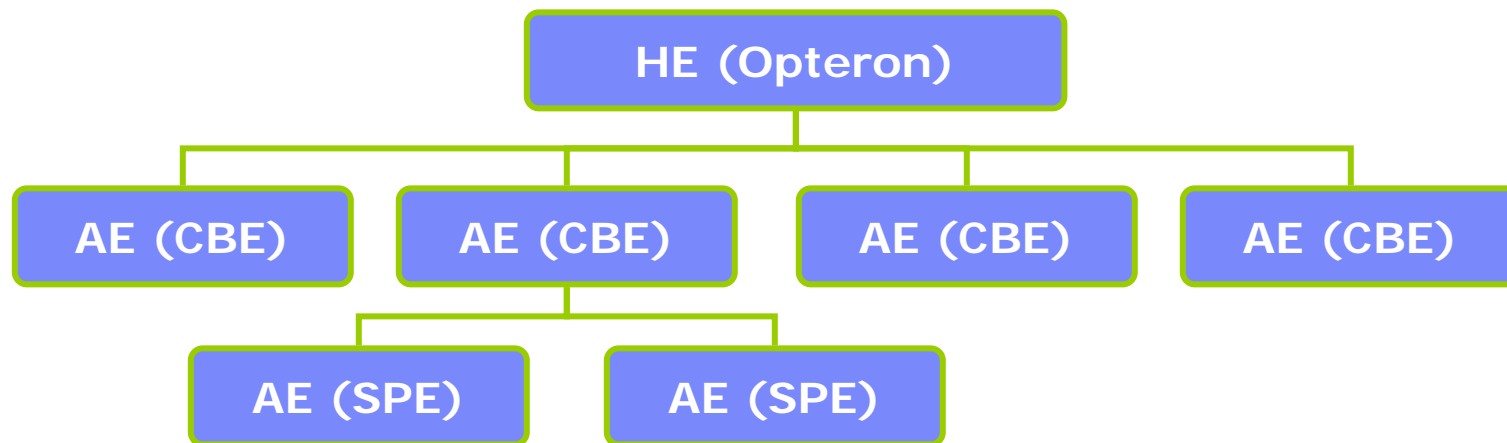
# DaCS Components Overview

- Process Management
  - Supports remote launching of an accelerator's process from a host process
- Topology Management
  - Identify the number of Accelerators of a certain type
  - Reserve a number of Accelerators of a certain type
- Data Movement Primitives
  - Remote Direct Memory Access (rDMA)
    - put/get
    - put\_list/get\_list
  - Message Passing
    - send/receive
  - Mailbox
    - write to mailbox/read from mailbox
- Synchronization
  - Mutex / Barrier
- Error Handling



## DaCS Component – Topology Management

- Identify the topology of Accelerator Elements for a specified Host Element
- Reserve a specific Accelerator Element
- Reserve a number of Accelerators of a certain type
- Release Accelerator Elements
- Heartbeat Accelerators for Availability



## DaCS Components – Process Management

- Remote launching and termination of an accelerator's process from a host process
- DaCS provides a method to start accelerator by sending executables and associated libraries
- DaCS utilizes a DaCS Daemon to facilitate process launching, error detection, and orphan process cleanup
- One DaCS Daemon (dacsd) per reserved accelerator

# DaCS APIs

## ▪ Init / Term

- dacs\_runtime\_init
- dacs\_runtime\_exit

## ▪ Reservation Service

- dacs\_get\_num\_avail\_children
- dacs\_reserve\_children
- dacs\_release\_de\_list

## ▪ Process Management

- dacs\_de\_start
- dacs\_num\_processes\_supported
- dacs\_num\_processes\_running
- dacs\_de\_wait
- dacs\_de\_test

## ▪ Group Functions

- dacs\_group\_init
- dacs\_group\_add\_member
- dacs\_group\_close
- dacs\_group\_destroy
- dacs\_group\_accept
- dacs\_group\_leave
- dacs\_barrier\_wait

## ▪ Data Communication

- dacs\_remote\_mem\_create
- dacs\_remote\_mem\_share
- dacs\_remote\_mem\_accept
- dacs\_remote\_mem\_release
- dacs\_remote\_mem\_destroy
- dacs\_remote\_mem\_query
- dacs\_put
- dacs\_get
- dacs\_put\_list
- dacs\_get\_list
- dacs\_send
- dacs\_recv
- dacs\_mailbox\_write
- dacs\_mailbox\_read
- dacs\_mailbox\_test
- dacs\_wid\_reserve
- dacs\_wid\_release
- dacs\_test
- dacs\_wait

## ▪ Locking Primitives

- dacs\_mutex\_init
- dacs\_mutex\_share
- dacs\_mutex\_accept
- dacs\_mutex\_lock
- dacs\_mutex\_try\_lock
- dacs\_mutex\_unlock
- dacs\_mutex\_release
- dacs\_mutex\_destroy

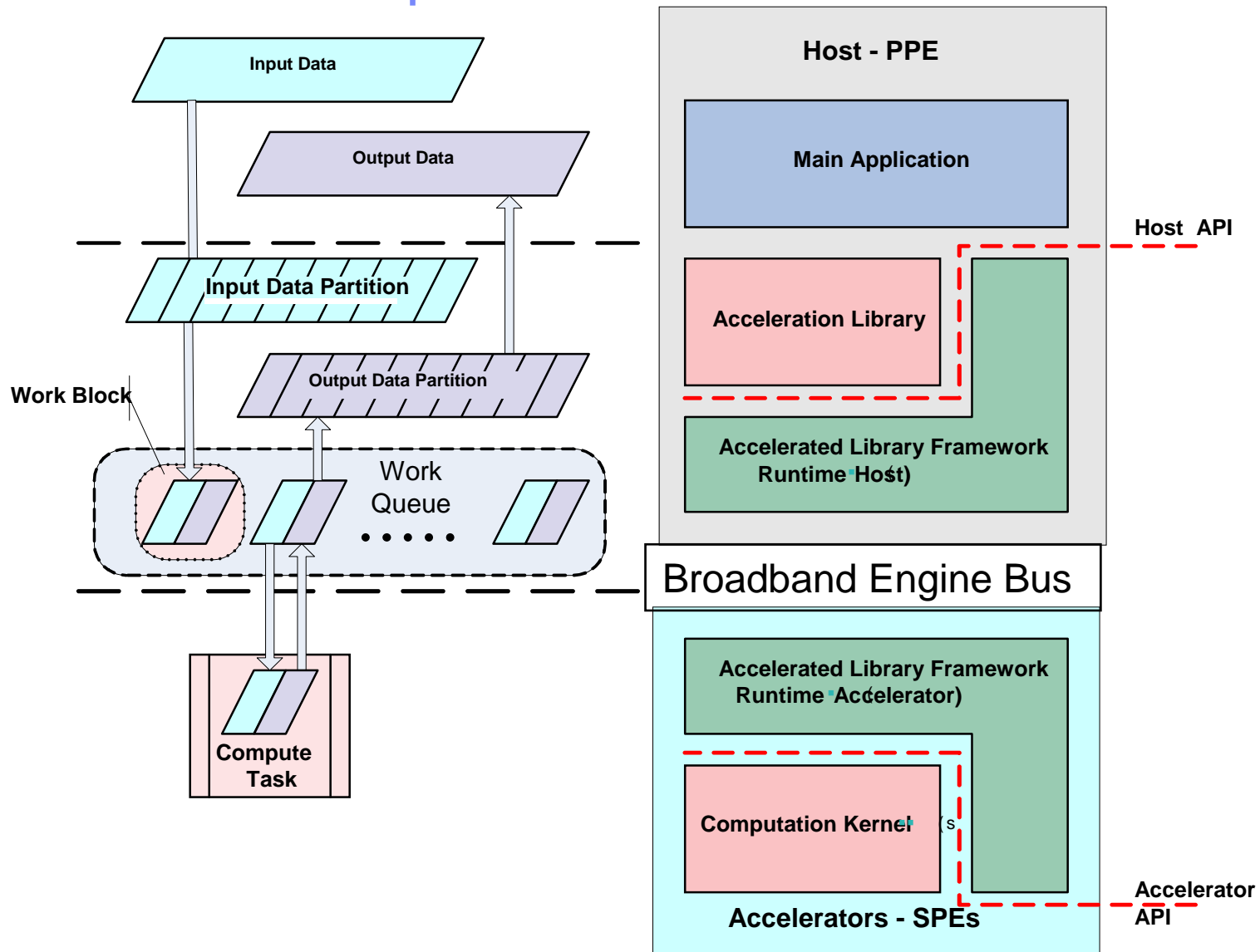
## ▪ Error Handling

- dacs\_errhandler\_reg
- dacs\_strerror
- dacs\_error\_num
- dacs\_error\_code
- dacs\_error\_str
- dacs\_error\_de
- dacs\_error\_pid

# ALF

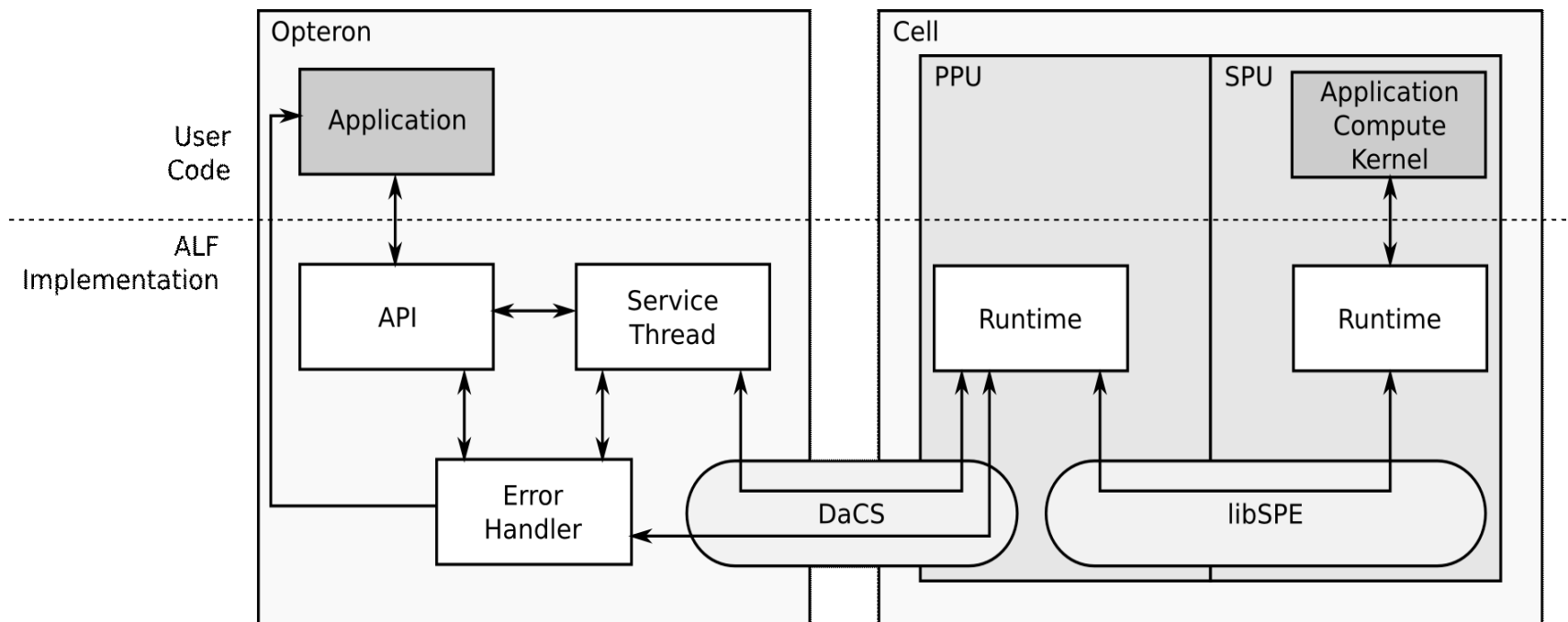
- Division of labor approach
  - ALF provides wrappers for computational kernels; synthesize kernels with data partitioning
- Initialization and cleanup of a group of accelerators
  - Groups are dynamic
  - Mutex locks provide synchronization mechanism for data and processing
- Remote error handling
- Data partitioning and list creation
  - Efficient scatter/gather implementations
    - Stateless embarrassingly parallel processing, strided partitioning, butterfly communications, etc.
  - Extensible to variety of data partitioning patterns
- SPMD – SDK2.1 , MPMD – SDK3.0
- Target prototypes: FFT, TRE, Sweep3D, Black Scholes, Linear Algebra

# ALF – Core Concepts

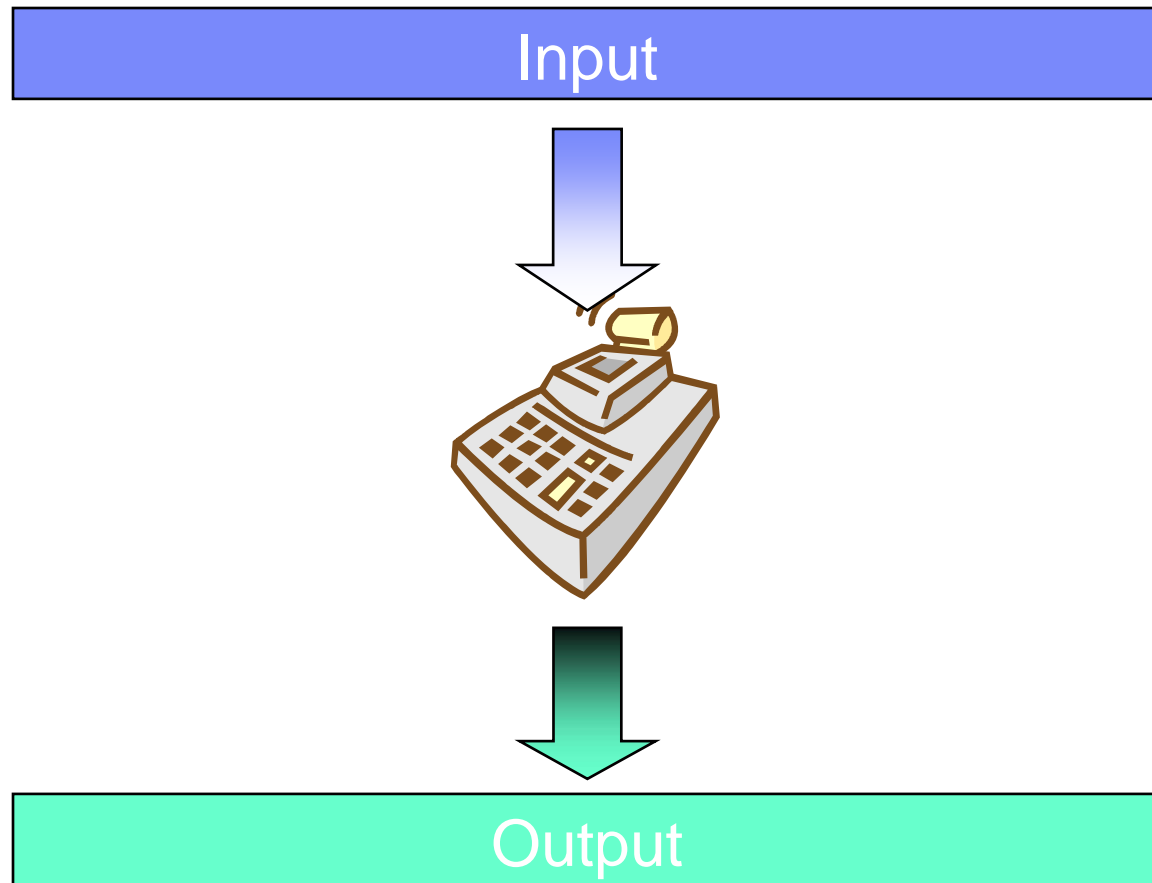


# ALF / DaCS Hybrid Implementation

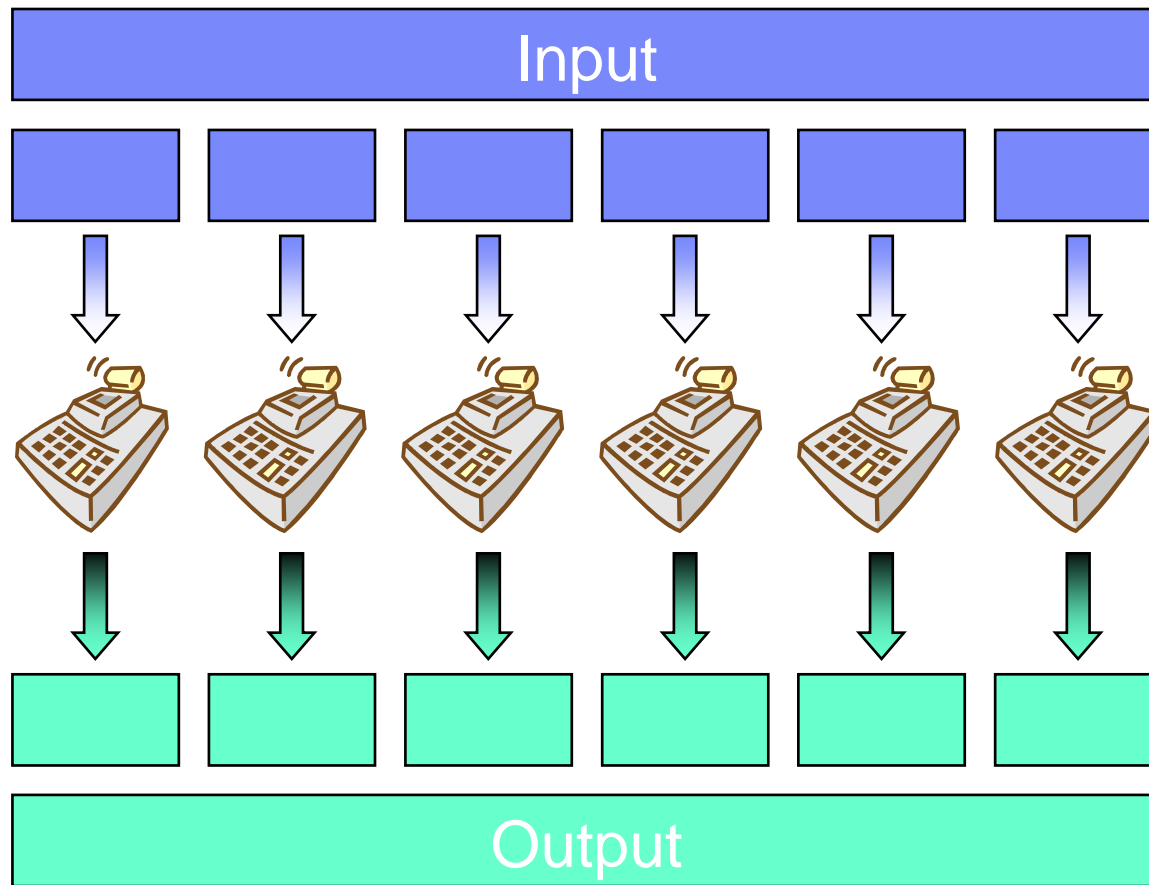
- **Model 1:** At an API level, the Opteron is acting as host and SPEs are accelerators. The PPE is a facilitator only. Programmers do not interact with the PPEs directly.
- **Model 2:** At an API level, the Opteron is acting as hosts and CellBE processors are accelerators. The PPE runs as ALF Accelerator and ALF host to the SPE accelerators. Opteron programmers do not interact with the SPEs directly.
- **ALF is being implemented using DaCS as the basic data mover and process management layer.**



# Overview - ALF on Data Parallel Problem

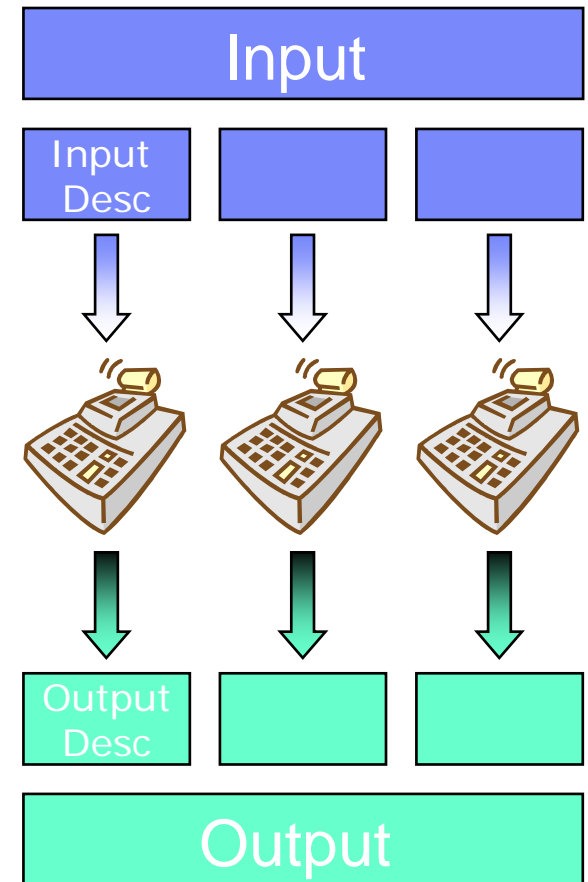
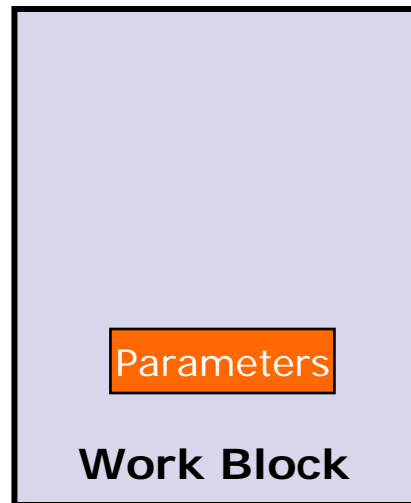


# ALF on Data Parallel Problem



# ALF - Workblock

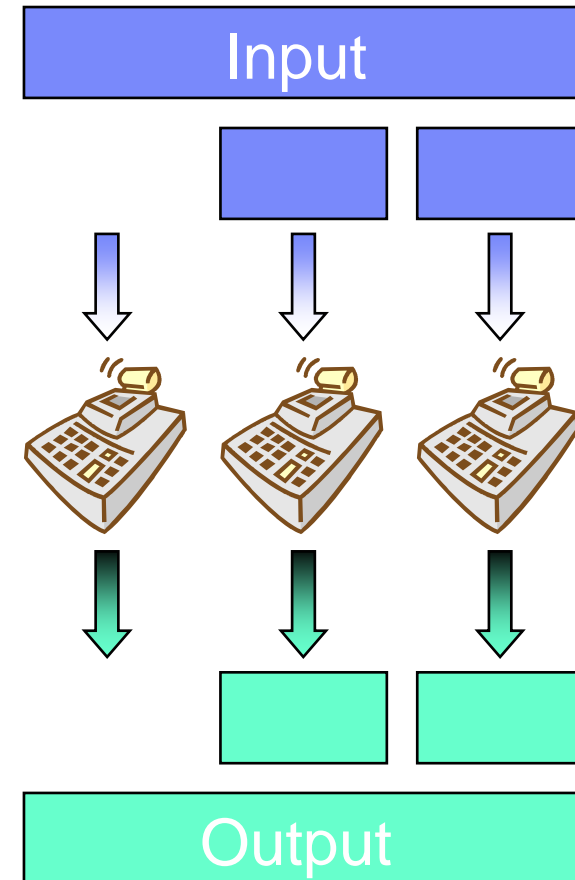
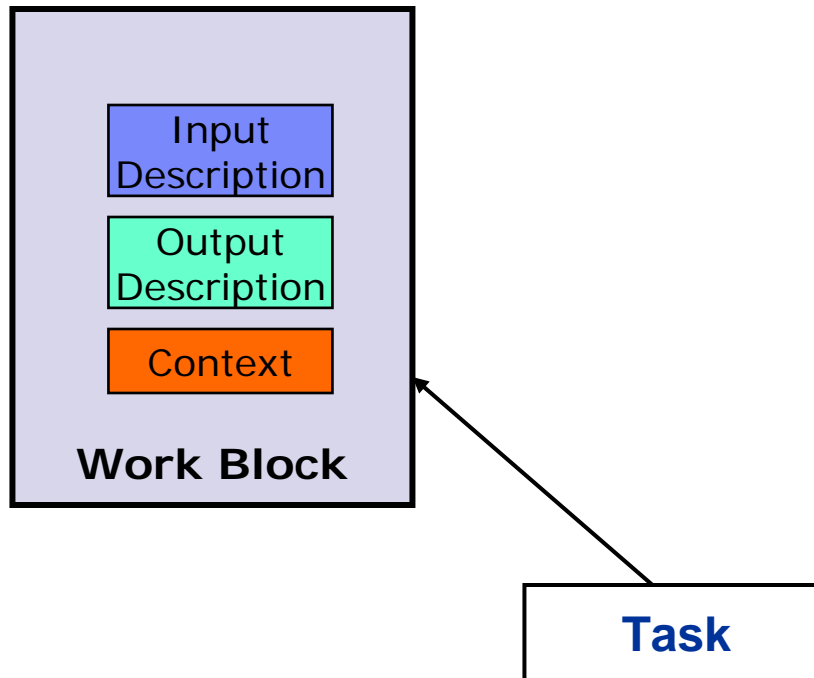
- **Workblock is the basic data unit of ALF**
- **Workblock = Partition Information ( Input Data, Output Data ) + Parameters**



- **Input Data and Output Data for a work load can be divided into many work blocks**

# ALF – Compute Task

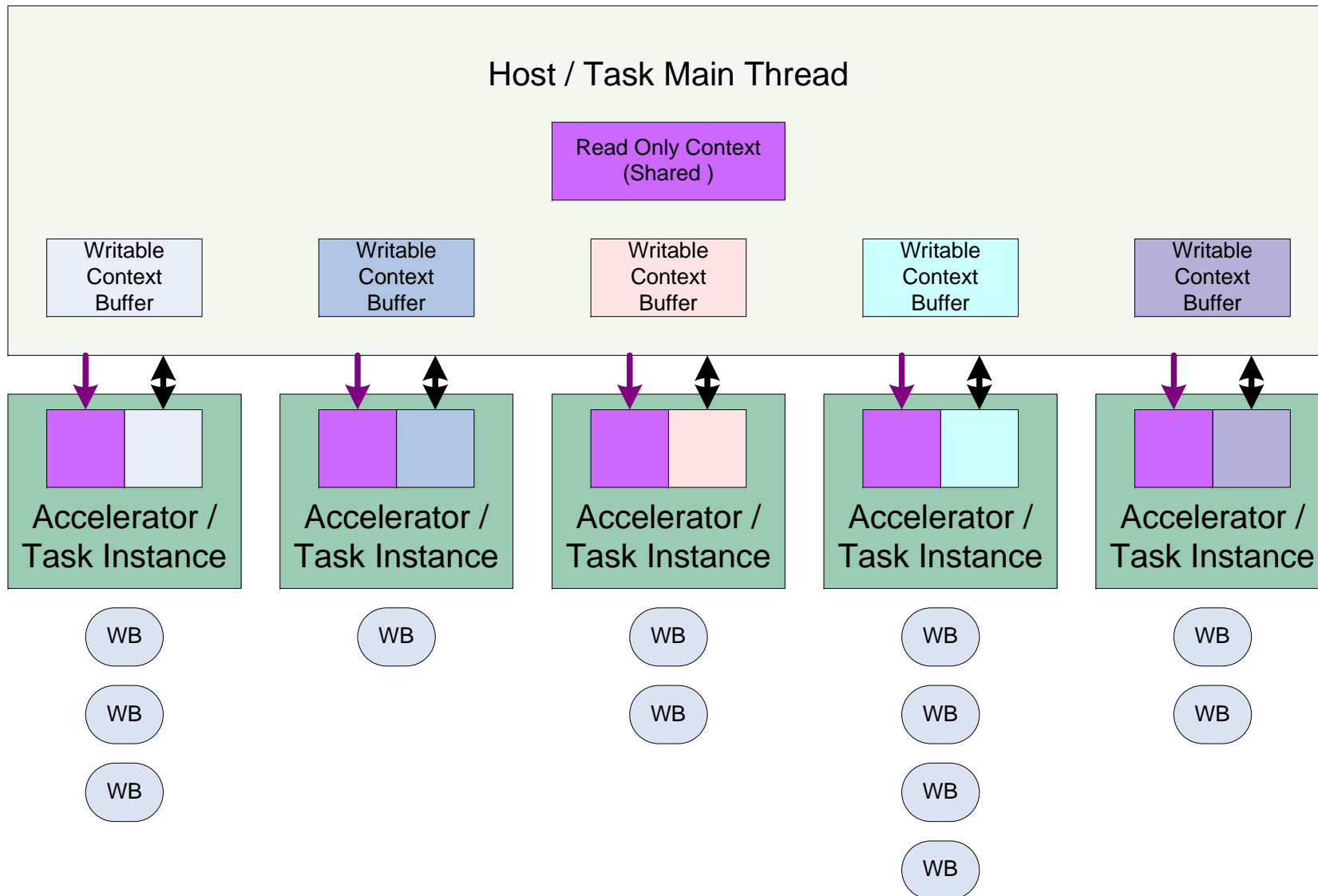
- **Compute Task processes a Work Block**
- **A task takes in the input data, context, parameters and produces the output data**



## ALF - Task Context

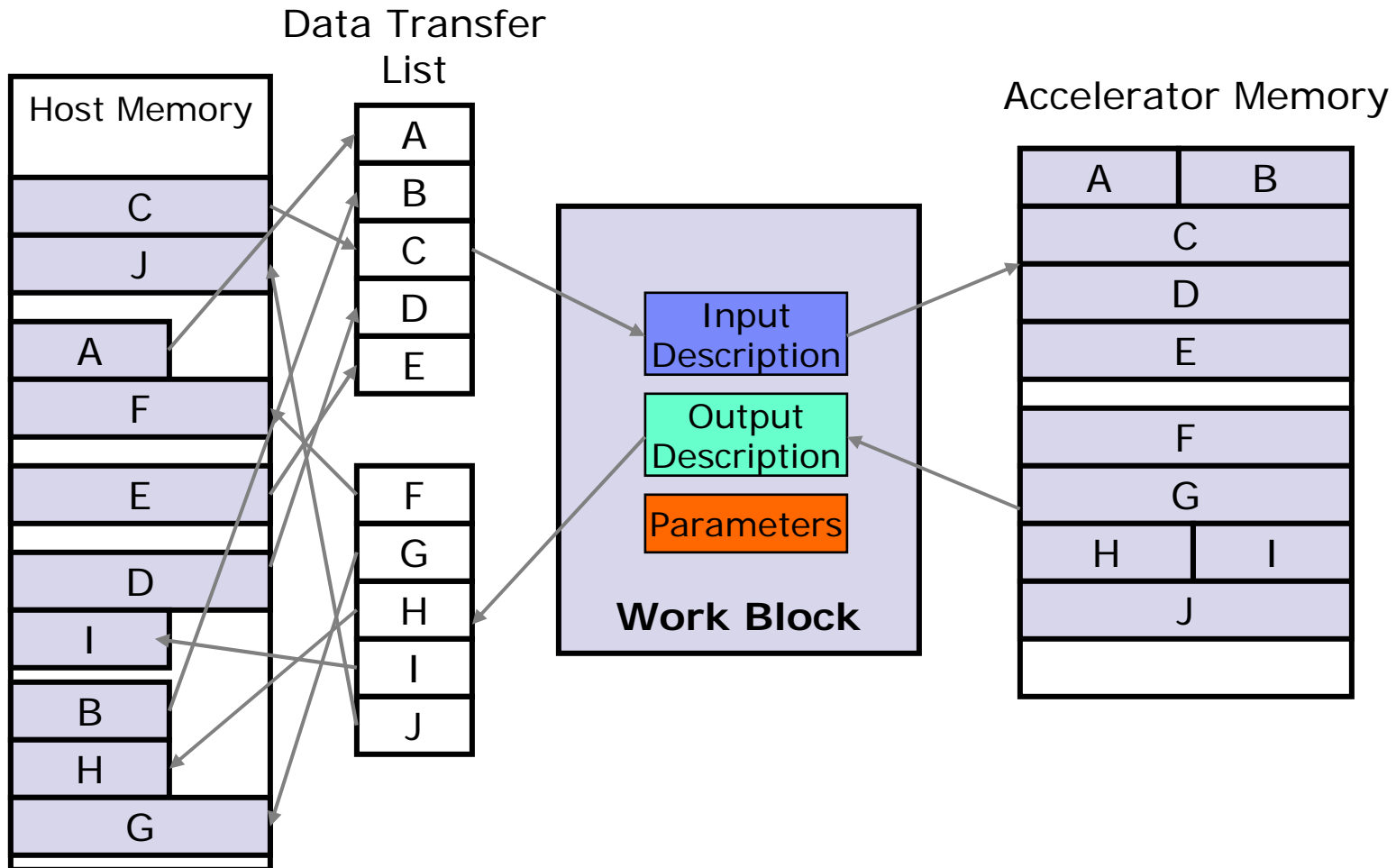
- **Provides persistent data buffer across work blocks**
- **Can be used for all-reduce operations such as min, max, sum, average, etc.**
- **Can have both read-only section and writable section**
  - Writable section can be returned to host memory once the task is finished
  - ALF runtime does not provide data coherency support if there is conflict in writing the section back to host memory
  - Programmers should create unique task context for each instance of a compute kernel on an accelerator

# ALF – Task Context



# ALF – Data Transfer List

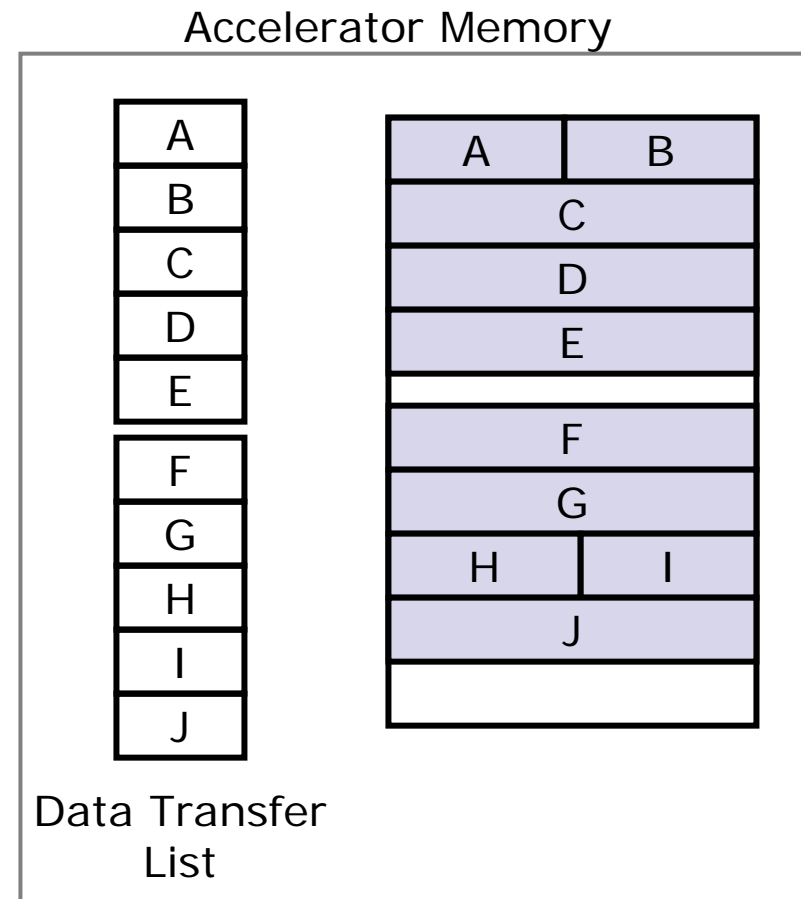
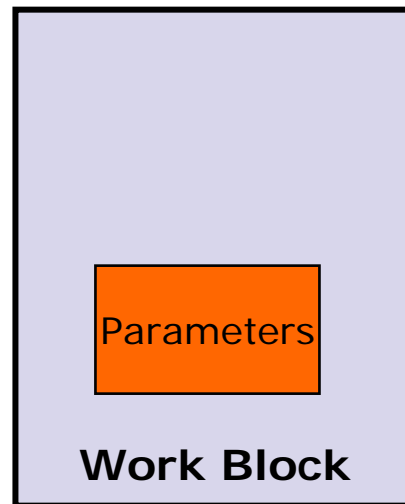
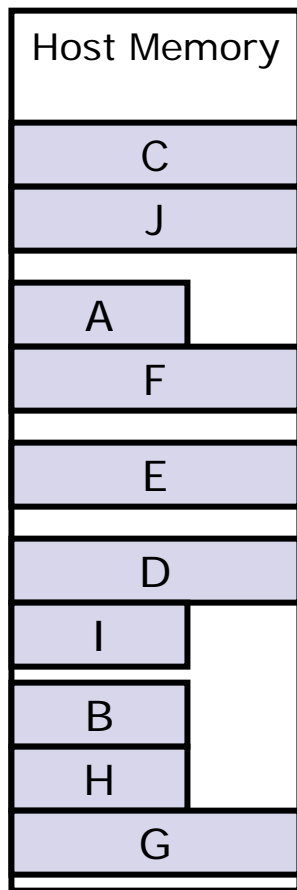
- Work Block input and output data descriptions are stored as Data Transfer List
- Input data transfer list is used to gather data from host memory
- Output data transfer list is used to scatter data to host memory



# ALF – Accelerator side data partitioning

## Two approaches to generate data transfer list

- Generate on the host side
  - ✓ Straight forward and easier to program
  - ✗ Host might not be able to support all accelerators
- Generate on the accelerator side
  - ✗ Indirect and harder to program (parameters will be passed in)
  - ✓ Many accelerators are stronger than a single control node



## ALF - Queues

- **Two different queues are important to programmers**
  - **Work block queue for each task**
    - Pending work blocks will be issued to the work queue
    - Task instance on each accelerator node will fetch from this queue
  - **Task queue for each ALF runtime**
    - Multiple Tasks executed at one time
      - except where programmer specifies dependencies
    - Future tasks can be issued. They will be placed on the task queue awaiting execution.
- **ALF runtime manages both queues**

## ALF - Buffer management on accelerators

- **ALF manages buffer allocation on accelerators' local memory**
- **ALF runtime provides pointers to 5 different buffers to a computational kernel**
  - Task context buffer (RO and RW sections)
  - Work block parameters
  - Input Buffer
  - Input/Output Buffer
  - Output Buffer
- **ALF implements a best effort double buffering scheme**
  - ALF determines if there is enough local memory for double buffering
- **Double buffer scenarios supported by ALF**
  - 4 buffers: [In0, Out0; In1, Out1]
  - 3 buffers: [In0, Out0; In1] : [Out1; In2, Out2]

# Synchronization Support

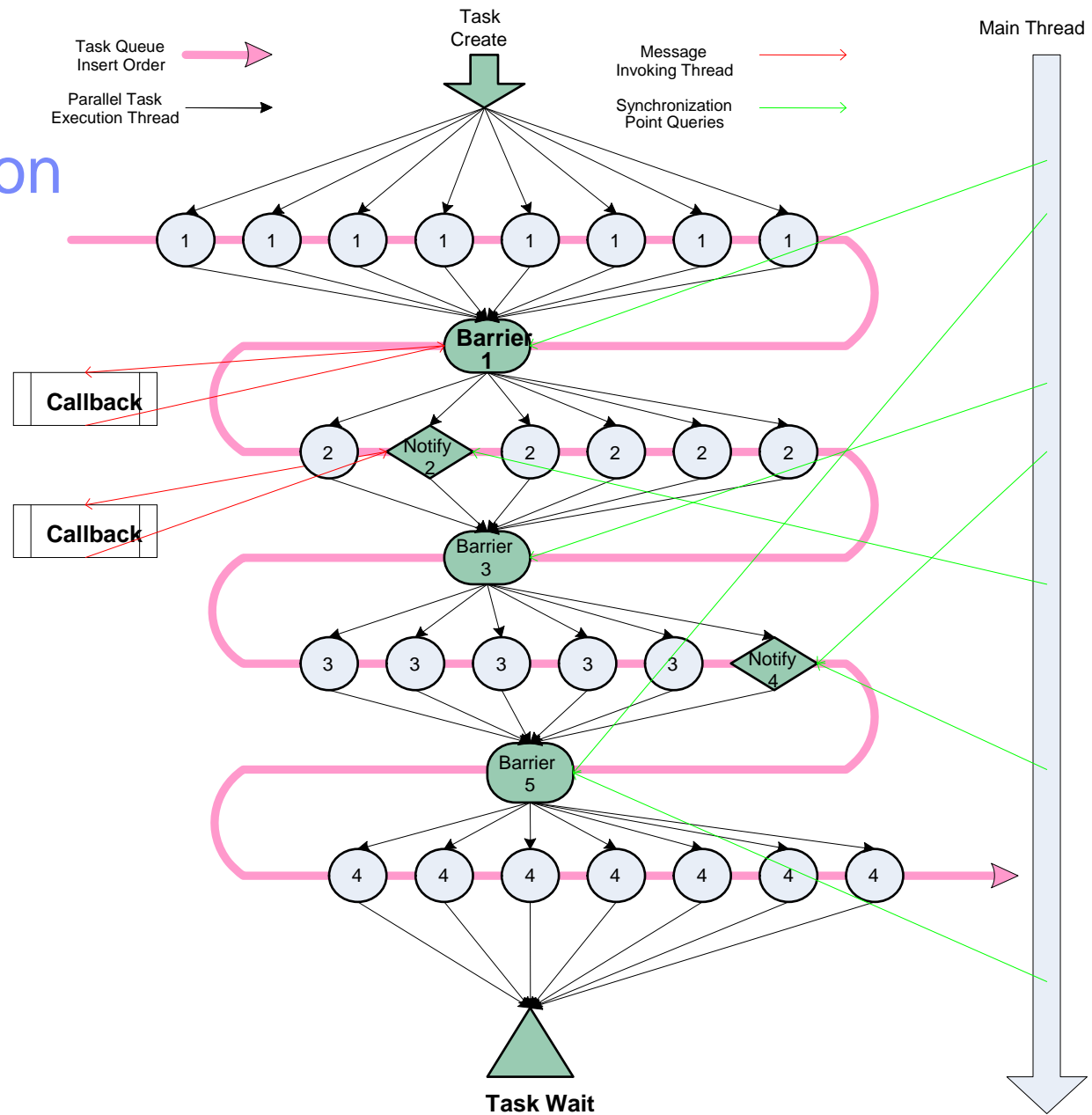
## ■ Barrier

- All work blocks enqueued before the barrier are guaranteed to be finished before any additional work block added after the barrier can be processed on any of the accelerators
- Programmers can register a callback function once the barrier is encountered. The main PPE application thread cannot proceed until the callback function returns.

## ■ Notification

- Allows programmers query for a specific work block completion
- Allows programmers to register a callback function once this particular work block has been finished
- Does not provide any ordering

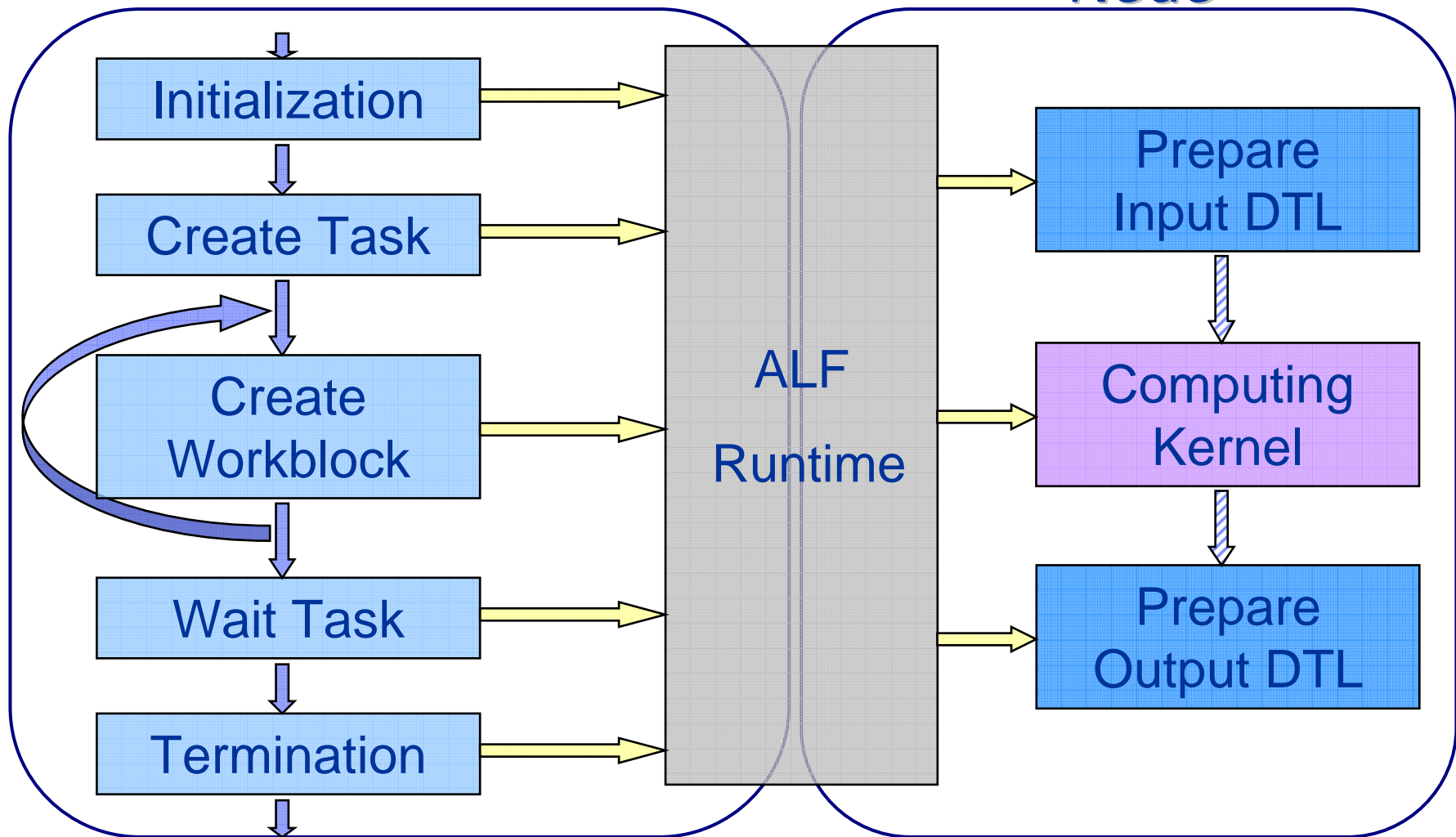
# ALF - Synchronization constructs



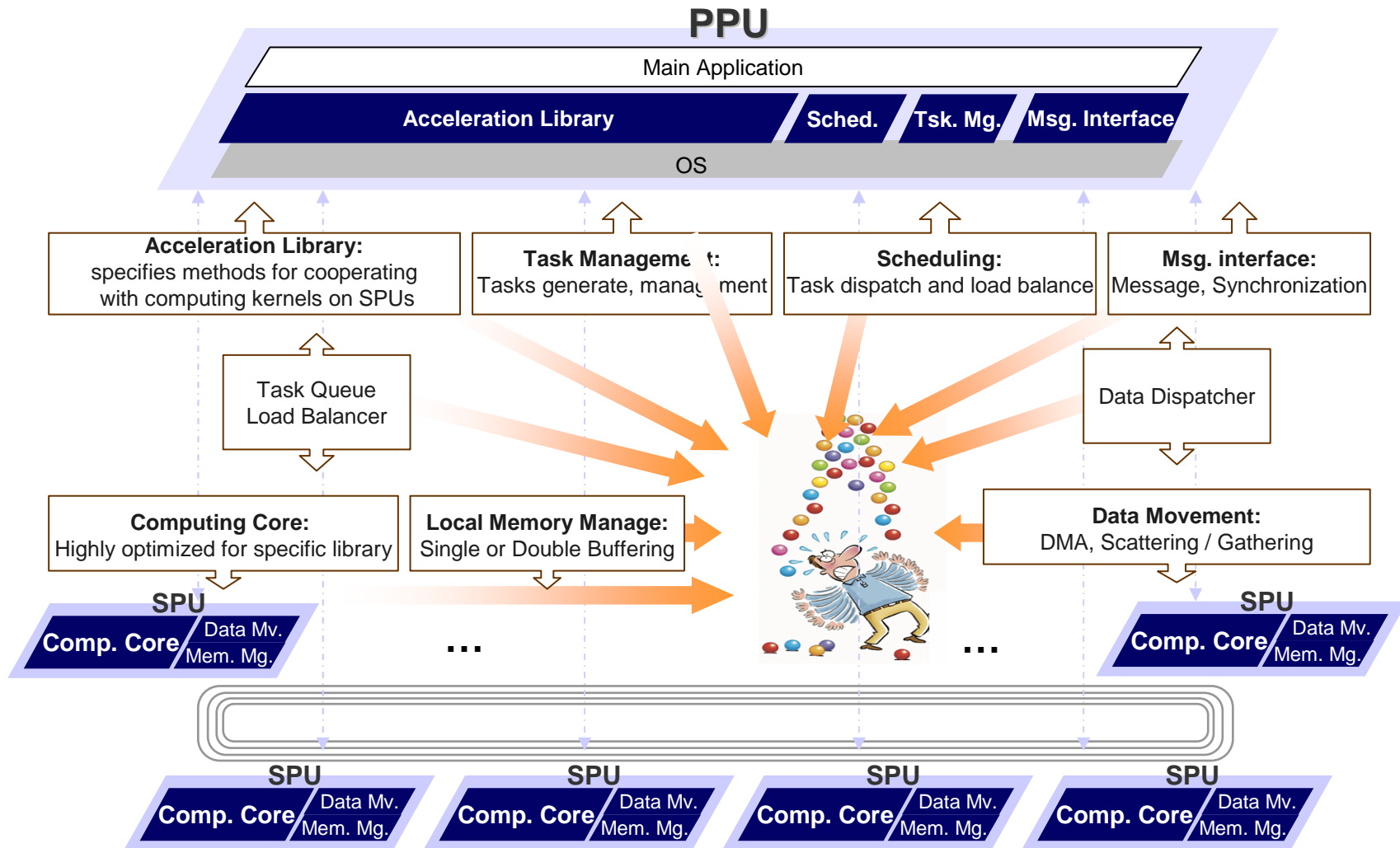
# Basic Structure of an ALF Application

*Control Node*

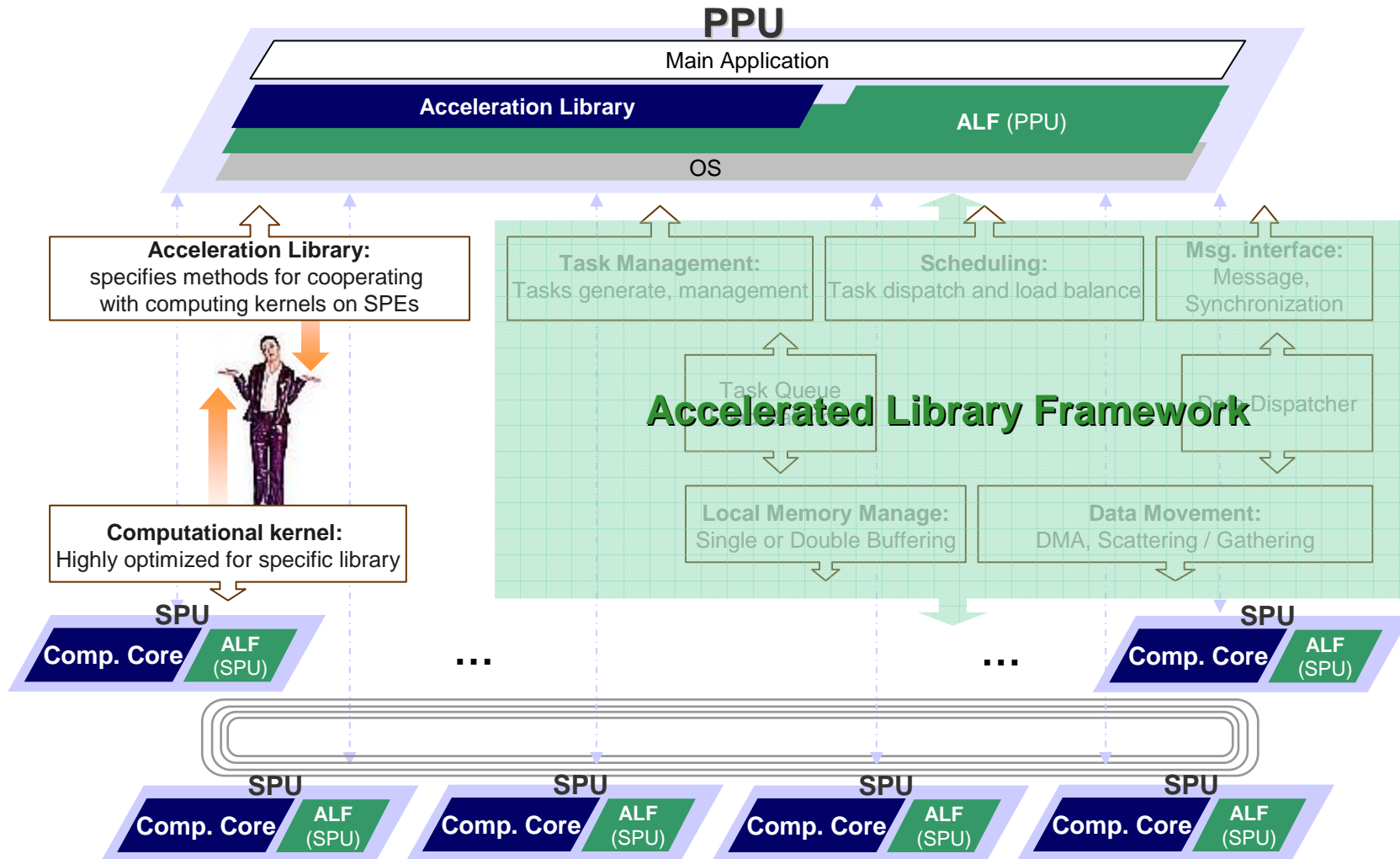
*Accelerator Node*



# Previous Development Environment



# Development Environment with ALF



## DaCS and ALF bindings

Package	C / C++ Binding	Fortran Binding
Cell DaCS	Yes	Yes
Hybrid DaCS	Yes	Yes
Cell ALF	Yes	No
Hybrid AFL	Yes	No

Thank You



Questions . . .

# Special Notices -- Trademarks

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Many of the features described in this document are operating system dependent and may not be available on Linux. For more information, please check: [http://www.ibm.com/systems/p/software/whitepapers/linux\\_overview.html](http://www.ibm.com/systems/p/software/whitepapers/linux_overview.html)

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised January 19, 2006

## Special Notices (Cont.) -- Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries: alphaWorks, BladeCenter, Blue Gene, ClusterProven, developerWorks, e business(logo), e(logo)business, e(logo)server, IBM, IBM(logo), ibm.com, IBM Business Partner (logo), IntelliStation, MediaStreamer, Micro Channel, NUMA-Q, PartnerWorld, PowerPC, PowerPC(logo), pSeries, TotalStorage, xSeries; Advanced Micro-Partitioning, eServer, Micro-Partitioning, NUMACenter, On Demand Business logo, OpenPower, POWER, Power Architecture, Power Everywhere, Power Family, Power PC, PowerPC Architecture, POWER5, POWER5+, POWER6, POWER6+, Redbooks, System p, System p5, System Storage, VideoCharger, Virtualization Engine.

A full list of U.S. trademarks owned by IBM may be found at: <http://www.ibm.com/legal/copytrade.shtml>.

Cell Broadband Engine and Cell Broadband Engine Architecture are trademarks of Sony Computer Entertainment, Inc. in the United States, other countries, or both.

Rambus is a registered trademark of Rambus, Inc.

XDR and FlexIO are trademarks of Rambus, Inc.

UNIX is a registered trademark in the United States, other countries or both.

Linux is a trademark of Linus Torvalds in the United States, other countries or both.

Fedora is a trademark of Redhat, Inc.

Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries or both.

Intel, Intel Xeon, Itanium and Pentium are trademarks or registered trademarks of Intel Corporation in the United States and/or other countries.

AMD Opteron is a trademark of Advanced Micro Devices, Inc.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

TPC-C and TPC-H are trademarks of the Transaction Performance Processing Council (TPPC).

SPECint, SPECfp, SPECjbb, SPECweb, SPECjAppServer, SPEC OMP, SPECviewerperf, SPECapc, SPECchpc, SPECjvm, SPECmail, SPECimap and SPECsfs are trademarks of the Standard Performance Evaluation Corp (SPEC).

AltiVec is a trademark of Freescale Semiconductor, Inc.

PCI-X and PCI Express are registered trademarks of PCI SIG.

InfiniBand™ is a trademark the InfiniBand® Trade Association

Other company, product and service names may be trademarks or service marks of others.

Revised July 23, 2006

# Special Notices - Copyrights

(c) Copyright International Business Machines Corporation 2005.  
All Rights Reserved. Printed in the United States September 2005.

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

Other company, product and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division  
1580 Route 52, Bldg. 504  
Hopewell Junction, NY 12533-6351

The IBM home page is <http://www.ibm.com>  
The IBM Microelectronics Division home page is  
<http://www.chips.ibm.com>