

Advanced Computer Systems Architecture

Chip-Multiprocessors: Applications and Architectures

CSE 561M

Prof. Patrick Crowley

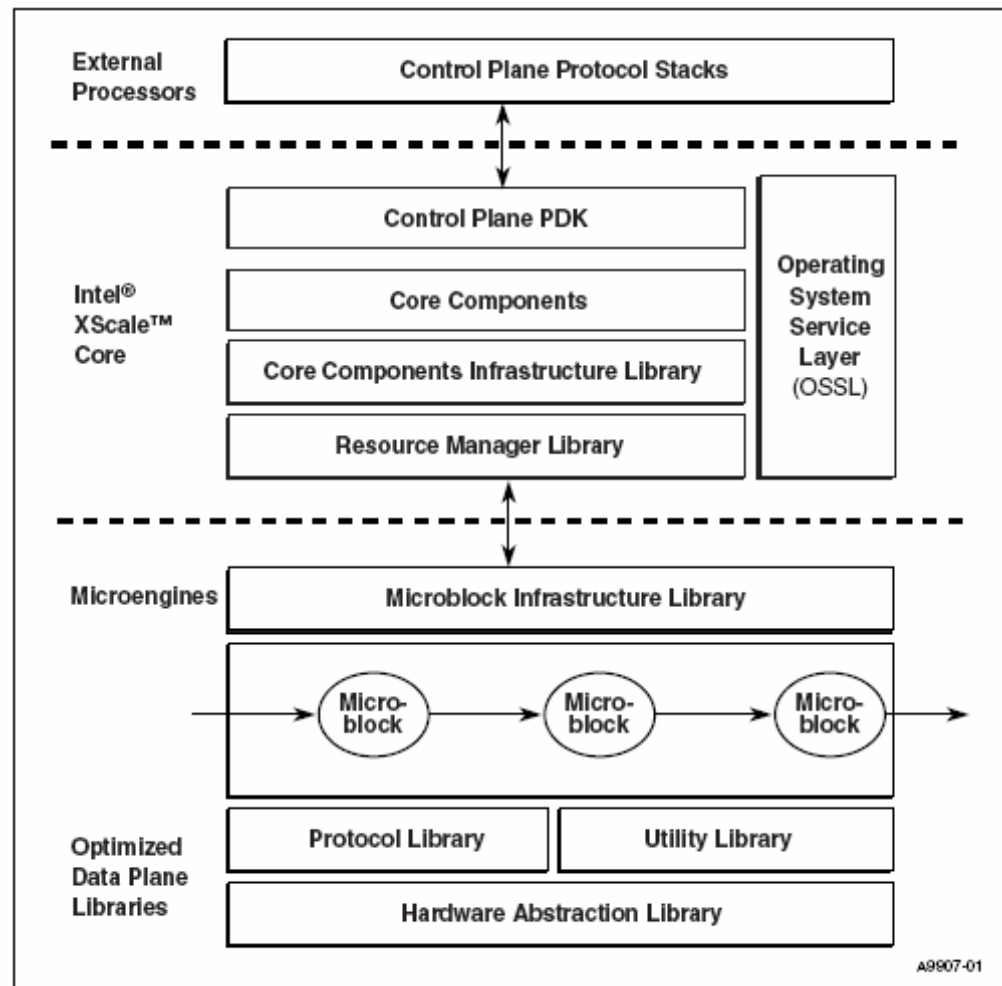
Plan for Today

- Questions
- Today's discussion

Objective

- Structure of IXA Applications
- Building Microblocks and Dispatch Loops
- Simple Receive, Process and Transmit shell

Intel IXA Portability Framework



See: IXA Portability Framework Developer's and Reference Manuals

Microblocks 1

- Structured code that runs on microengines
- Meant to be interchangeable, composable blocks of functionality
- In full blown IXA environment, microblocks should have a control counterpart (i.e., core) on the XScale

Microblocks 2

- By convention, each implementation has its own unique source file (to enable reuse)
- Application-specific **dispatch loops** on each ME include multiple microblocks, and coordinate flow between them
- Are initialized and invoked with macro calls

Dispatch Loop

- An infinite loop that *manages flow between microblocks on a single microengine*
- Dispatch loop source files are the root for most `.list` files
- Dispatch loop state (`dl_*`) is used to pass information between microblocks
 - this is Intel's programming convention
- More details can be found here
 - Ch. 5 of the *Framework Developers Manual* (PDF file)
 - Ch. 2 of the *Framework Reference Manual* (PDF file)

Dispatch Loop State

- Common variable names used to pass *buffer handles* between microblocks
- Main dispatch loop variables
 - dl_buf_handle: handle for “current” data
 - dl_next_block: ID of the microblock that should receive this data
- Dispatch loop variables
 - Accessed in assembly via macros, e.g., dl_buf_get_data()
 - Lots of extended information stored here, but is a bit arcane

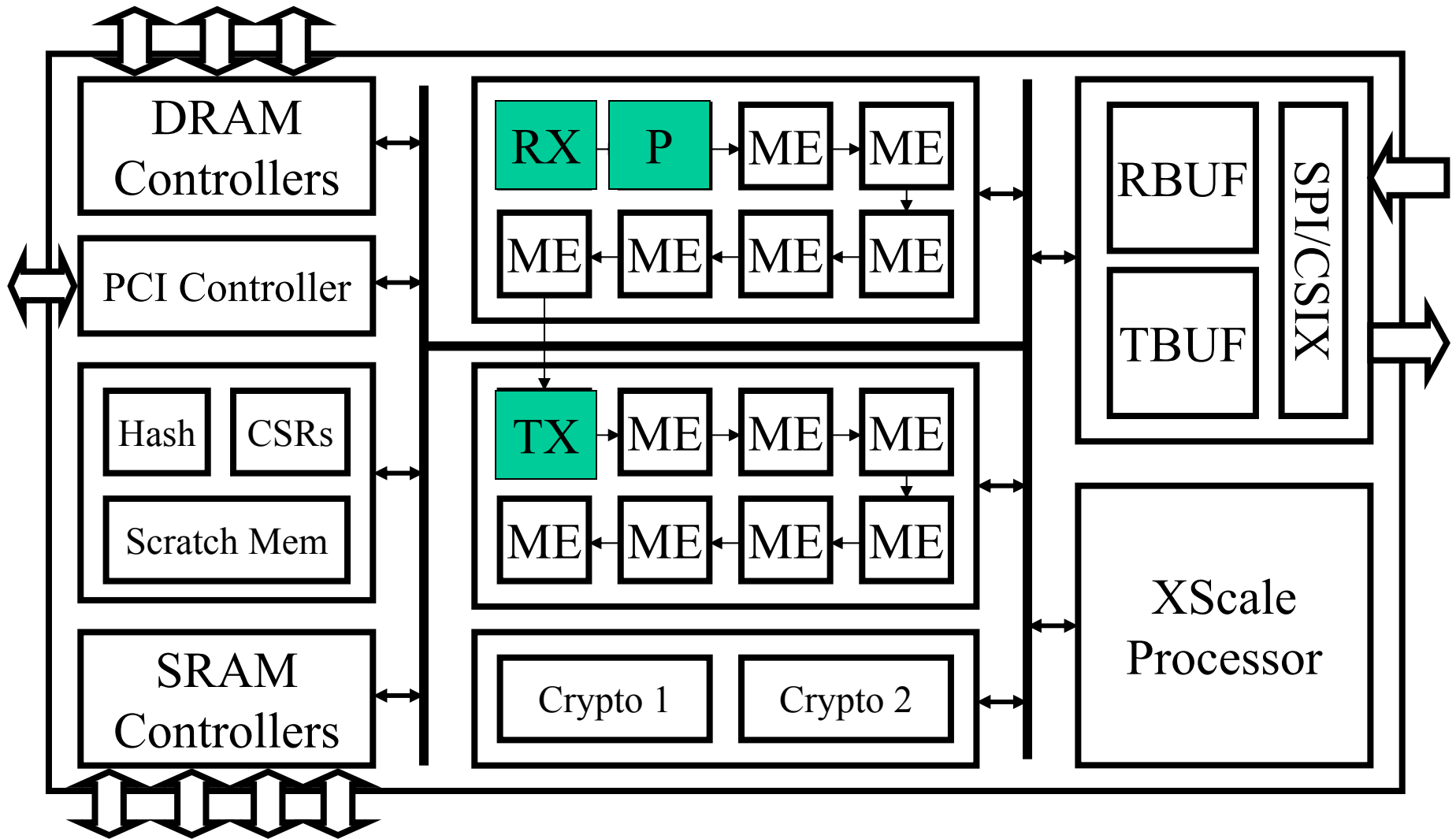
Microblock Example

- Consider the “process” microblock
- Microblock interface includes:
 - Standard dispatch loop variables: `dl_buf_handle`, `dl_next_block`, etc.
 - Static label for output (for microblock chaining)
 - `PROCESSING_NEXT_BLOCK`
 - Macros
 - `process_init` (initialization)
 - `process` (implementation)
- This microblock can be used within a dispatch loop

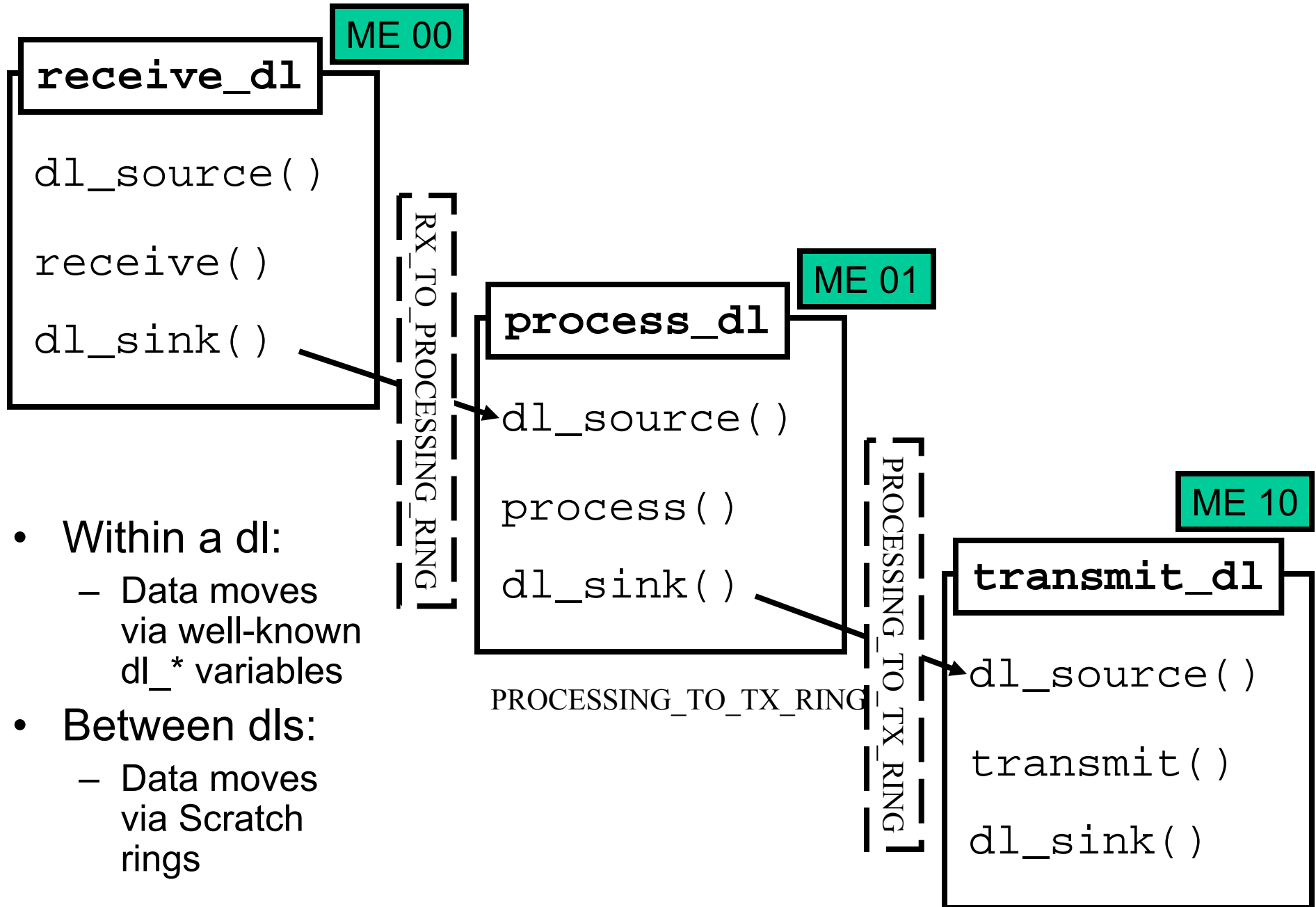
Anatomy of a Microblock and Its Dispatch Loop

- Example project (simple_ixa_app)

Sample Project Setup



Microblocks and Dispatch Loops



- Within a dl:
 - Data moves via well-known `dl_*` variables
- Between dls:
 - Data moves via Scratch rings

Organizing Your Project Files

Template:

- [Directory] Application/Project
 - [Directory] dispatch_loop
 - [Files] dispatch loop implementations
 - [File] **dispatch_loop.{uc|h}** (dispatch loop library code)
 - [File] **dl_system.h** (app-specific constants, microblock topology)
 - [File] **dl_source.{uc|h}** (app-specific source and sink init)
 - [File] **system_init.uc** (app-specific system initialization)
 - [File] Project .dwp file
 - [Files] Microblock implementations

Example:

- [CSE561_simple_ixa_app]
 - [dispatch_loop]
 - process_dl.uc
 - receive_dl.uc
 - transmit_dl.uc
 - **dispatch_loop.{uc|h}**
 - **dl_system.h**
 - **dl_source.{uc|h}**
 - **system_init.uc**
 - CSE561_simple_ixa_app.dwp
 - scratch_rings.uc
 - process.uc
 - receive.uc
 - transmit.uc

Simple IXA Example

- Example Project (simple_ixa_app)
- Questions
 - How is the system initialized?
 - How is control/data passed between microblocks on the *same* ME?
 - How is data passed between microblocks on *different* MEs?

Assignment

- Tuesday
 - Skim Ch. 4, Read Ch. 5
 - Commentary: Choose one of the following
 - How would you improve on the scratchpad ring put performance? Describe your design, implement it and report results.
 - Implement an analogous set of macros for SRAM rings. Describe the differences and performance and share your implementation in your commentary.
 - How does this compare to a dynamically-linked ring implementation on your desktop machine? Describe your experience in your commentary.
- Thursday
 - J&K: Read Ch. 6