MARS

Multicore Application Runtime System

July 24, 2008

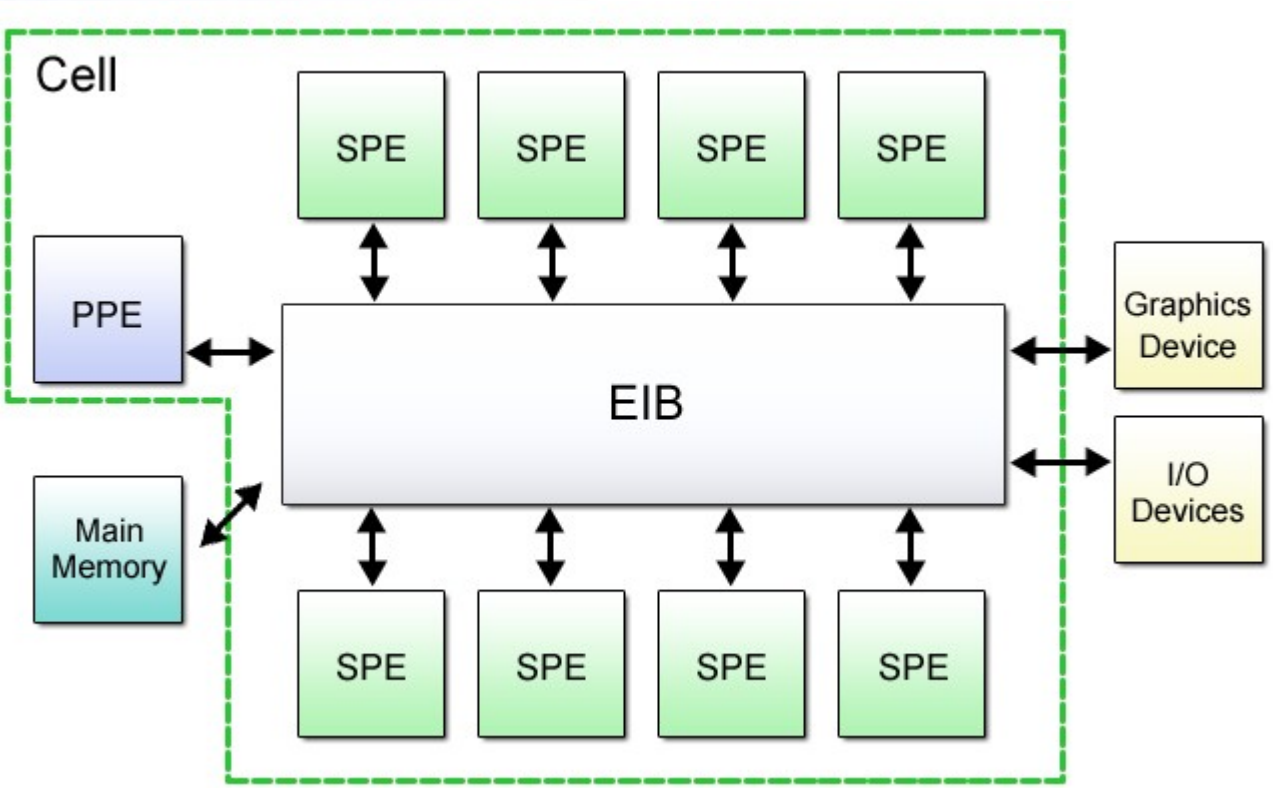
Geoff Levand

<geoffrey.levand@am.sony.com>

Introduction to the Cell Broadband Engine

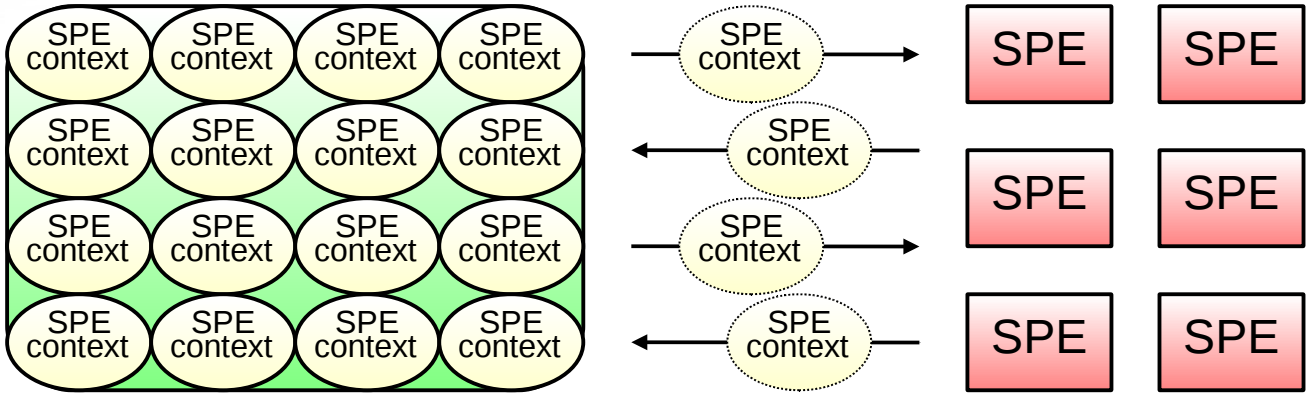
Heterogeneous Multicore Processor

- 1 PPE (PowerPC Processor Element)
  - PPC64 + VMX instructions
- 8 SPE (Synergistic Processor Element)
  - 4way SIMD + 256KiB Local Store



Linux Kernel SPE Context Scheduler

- Physical SPEs virtualized by kernel SPUFS
- Pre-emptive SPE context switching
- High cost to store and reload entire SPE processor state



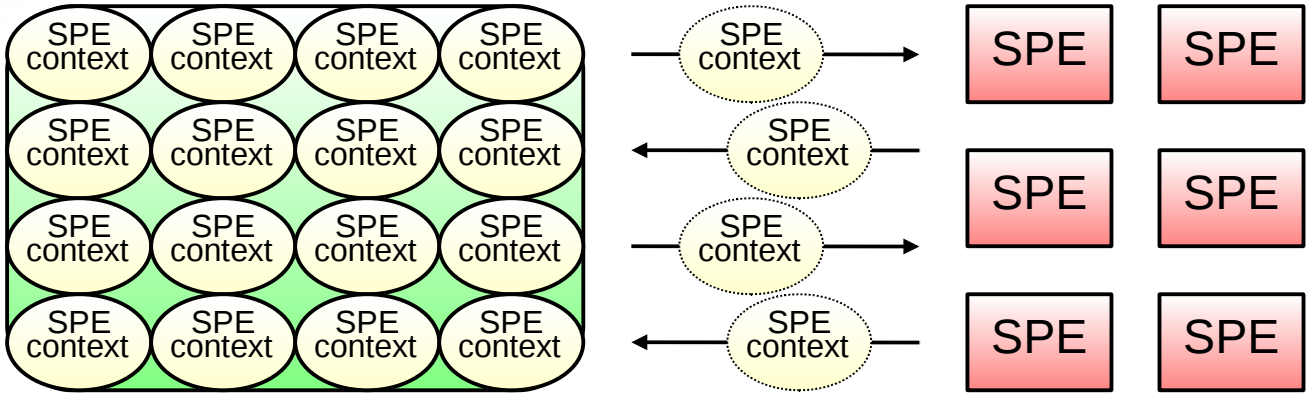
Linux Kernel SPE Context Scheduler

- Best performance when:

  (# of SPE contexts) <= (# of physical SPEs available)

- High context switching overhead when:

  (# of SPE contexts) > (# of physical SPEs available)

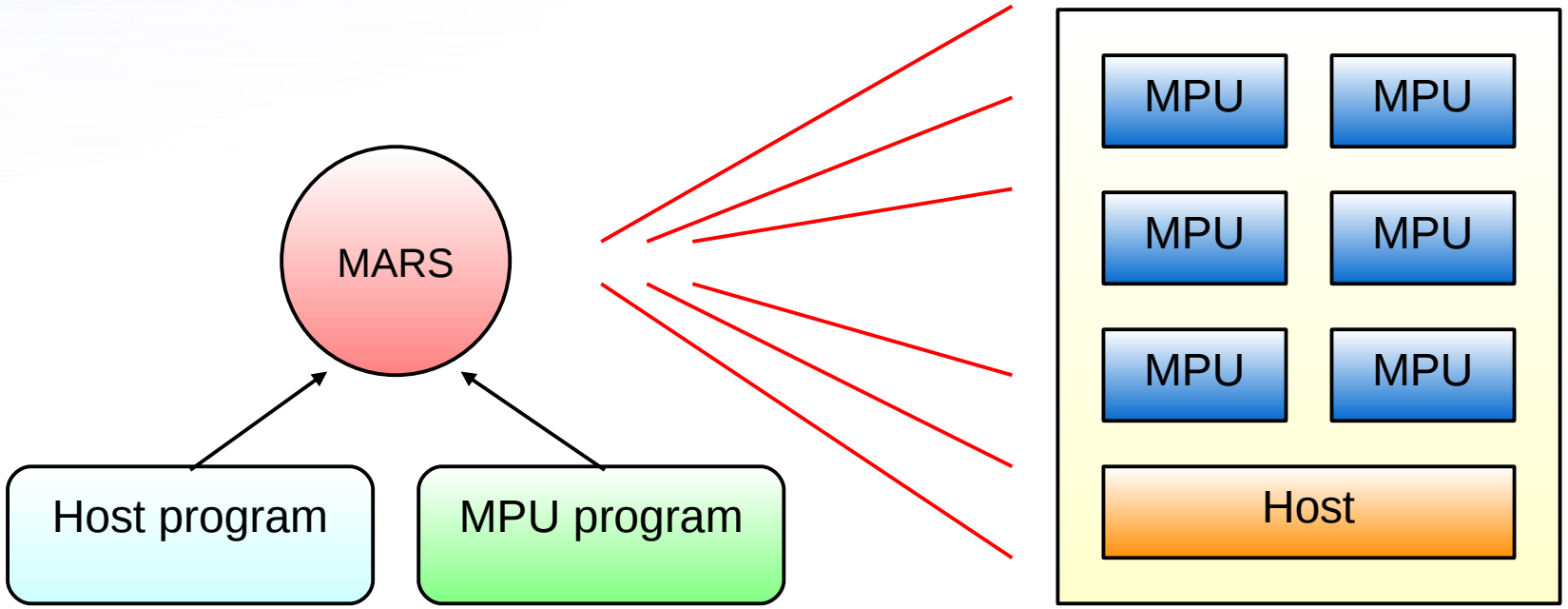


MARS Terminology

- **MARS** – Multicore Application Runtime System
- **Host** – host processor (**PPE**)
- **MPU** – micro-processing unit (**SPE**)
- **Host storage** – shared memory space (**main memory**)
- **MPU storage** – MPU local memory space (**local store**)
- **Workload** – a generic unit of process(es) scheduled for execution on the MPU (**SPE context**)

What is MARS?

- MPU-centric runtime environment for multicore architectures
- Scheduling and management of workloads by MPU
- APIs to manage user programs that run on MPUs

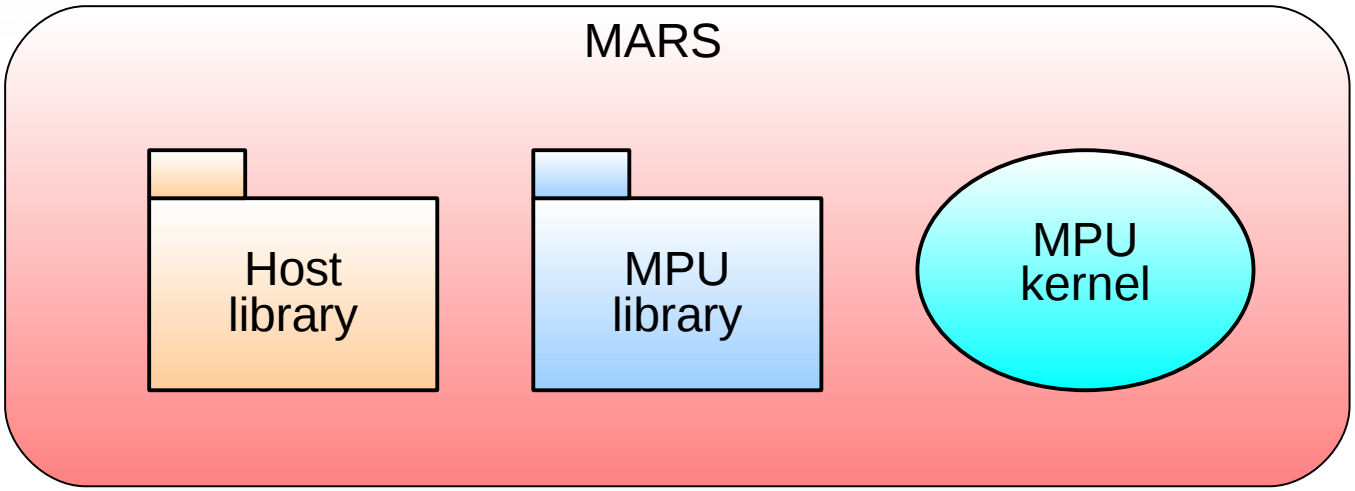


Why use MARS?

- Lightweight context switching
- Performance advantage over libspe when:
  (# **of workloads) > (# of physical MPUs available**)
- Minimizes runtime load of the host processor
- Synchronization objects call the scheduler
- Simplifies maximizing SPE utilization

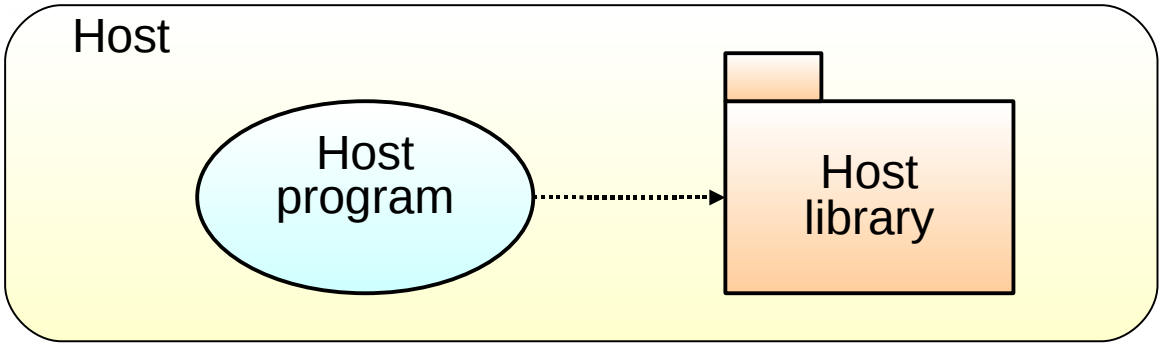
What does MARS provide?

- Host-side programming library
- MPU-side programming library
- MPU-side kernel



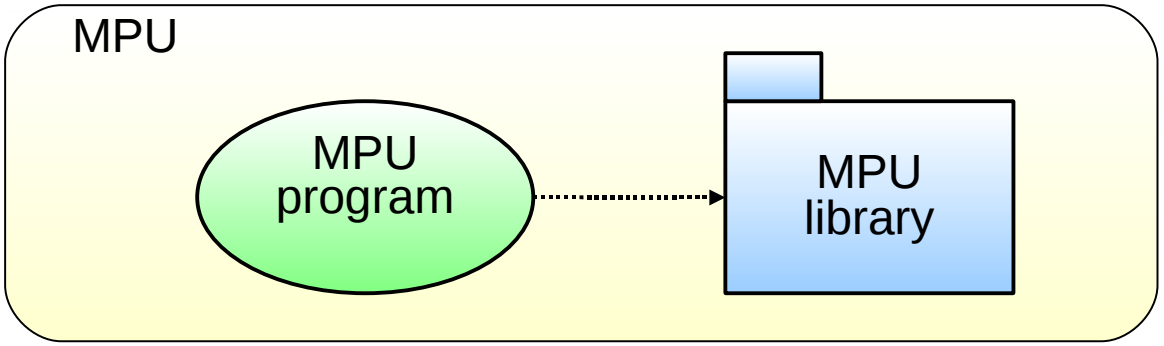
MARS Host Programming Library

- APIs to manage the MARS context
- APIs to initialize/schedule workloads for execution
- APIs to synchronize Host and MPU execution



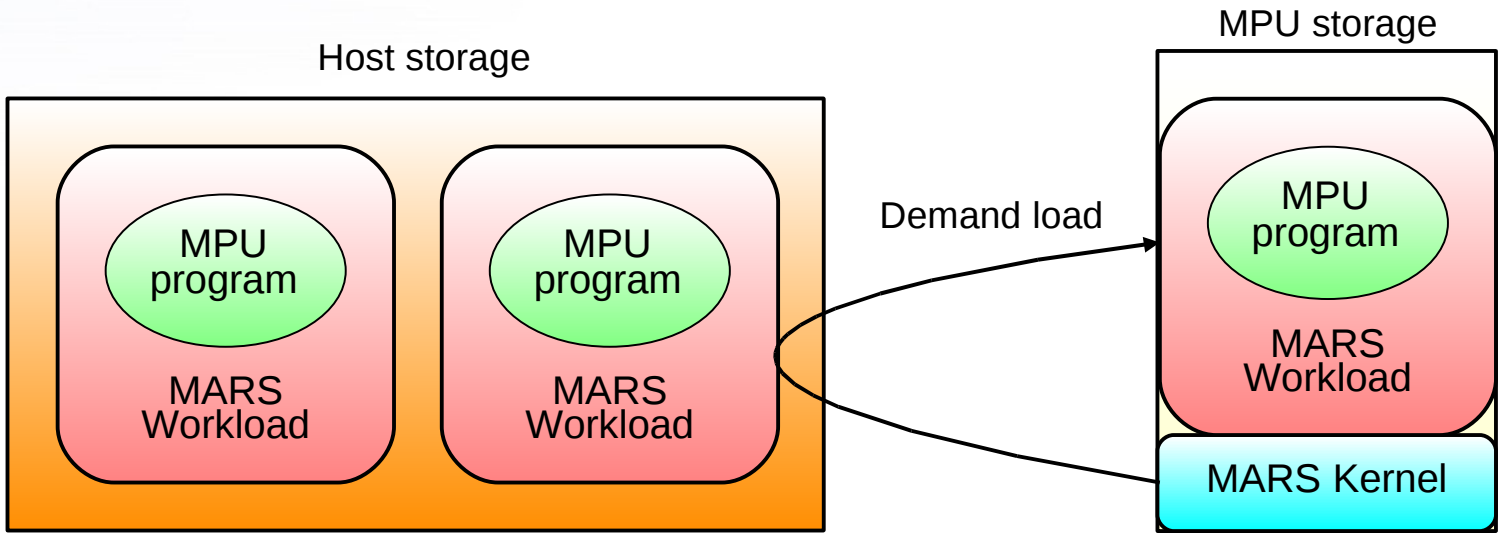
MARS MPU Programming Library

- APIs to manage MPU program state (ex. yield, exit, etc.)
- APIs to schedule initialized workloads
- APIs to synchronize Host and other MPUs



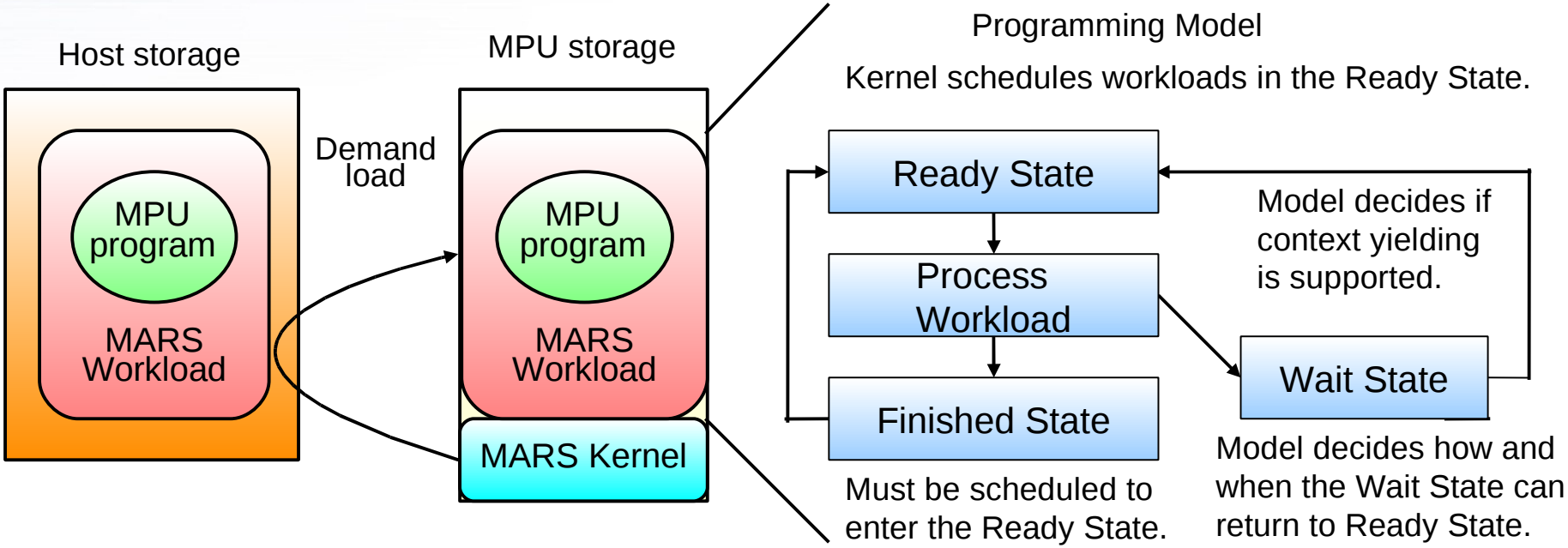
MARS MPU Kernel

- Resident in MPU storage throughout life of MARS context
- Demand loads workloads from main memory to MPU Memory
- Priority-based cooperative scheduling



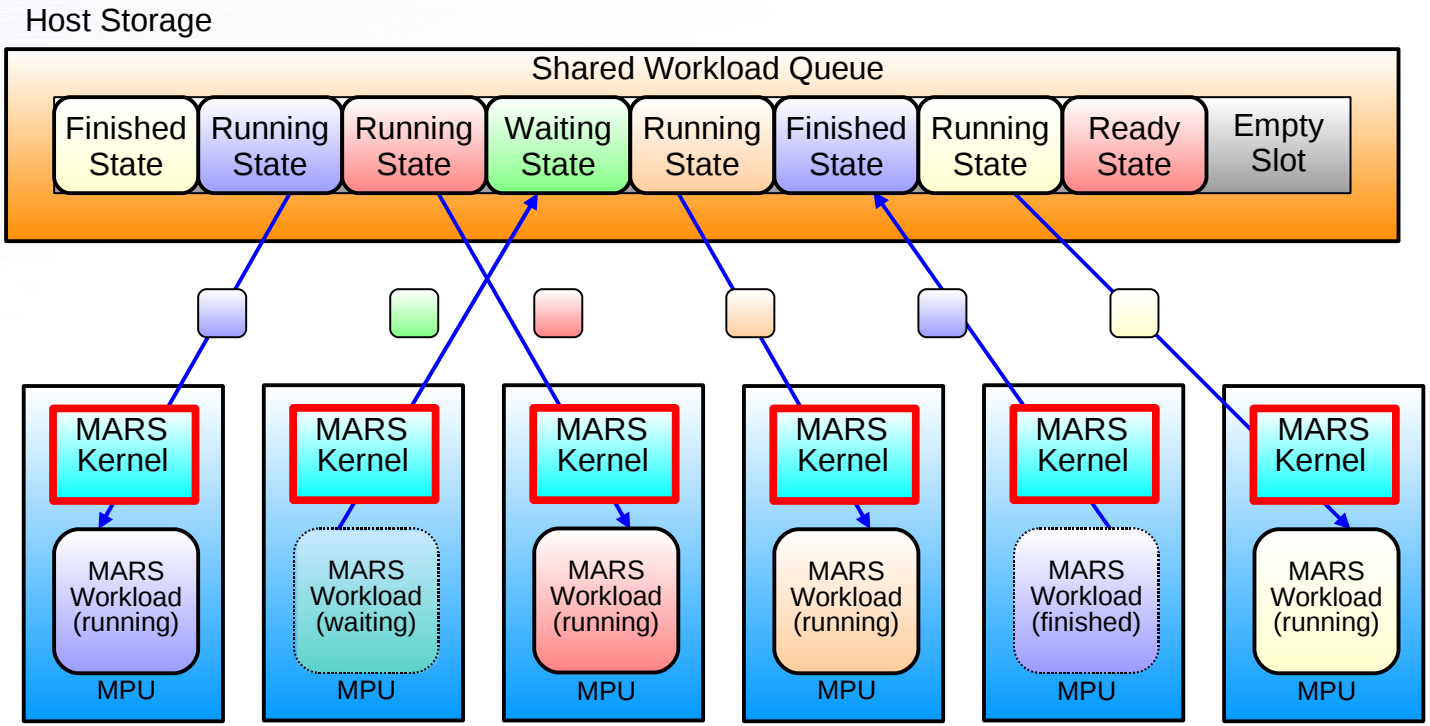
MARS Workload

- Workloads in shared queue scheduled by MARS kernels
- Scheduling based on state, priority, fairness, etc.
- Content of workload determined by programming model
- Task model, Job model, Low level user program, etc.



MARS Workload Queue

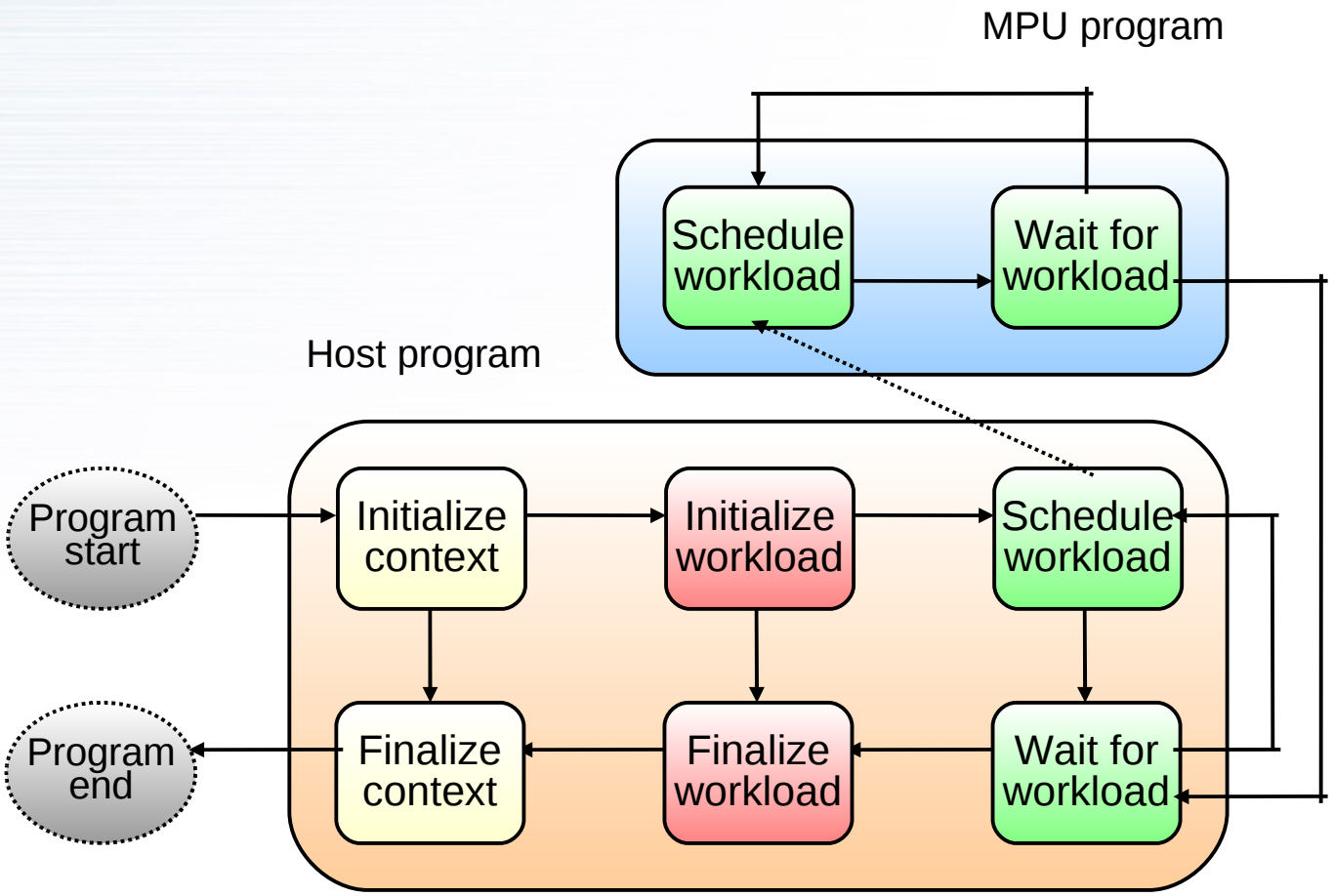
- Workload queue atomically accessed by both Host and MPUs
- Created in main memory and managed through a MARS context
- MARS MPU kernels reserve runnable workloads for execution



MARS General Usage Flow

- Initialize a MARS context
  - Prepares the MARS kernel for each MPU
- Initialize workloads
  - Prepares workload contexts and adds to shared workload queue
- Schedule workloads for execution
  - Workload scheduling can be requested from Host and MPU
- Wait for workload completion
  - Synchronous and asynchronous waiting provided
- Finalize the MARS context
  - Release system resources

MARS General Usage Flow



MARS Host Program

```
#include <libspe2.h>
#include <mars/mars.h>

extern struct spe_program_handle mpu_task_prog;          /* instance of task program ELF */

int main(void)
{
   struct mars_context mars_ctx;                         /* MARS context instance */
   struct mars_task_id task_id;                          /* Task id handle*/
   struct mars_task_params task_params;                  /* Task initialization parameters */
   struct mars_task_args task_args;                      /* Task arguments */
   int task_prio = 0;                                    /* Task scheduling priority */

   mars_initialize(&mars_ctx, NULL);                     /* Initialize MARS context */

   task_params.name = "Hello World";                     /* Task name */
   task_params.elf_image = mpu_task_prog.elf_image;      /* Task program ELF address */
   task_params.context_save_size = 0;                    /* Task context save area */

   mars_task_initialize(&mars_ctx, &task_id, &task_params);  /* Initialize Task context */
   mars_task_schedule(&task_id, &task_args, task_prio);      /* Schedule Task context */
   mars_task_wait(&task_id);                             /* Wait for Task completion */
   mars_task_finalize(&task_id);                         /* Finalize Task context */

   mars_finalize(&mars_ctx);                             /* Finalize MARS context */

   return 0;
}
```

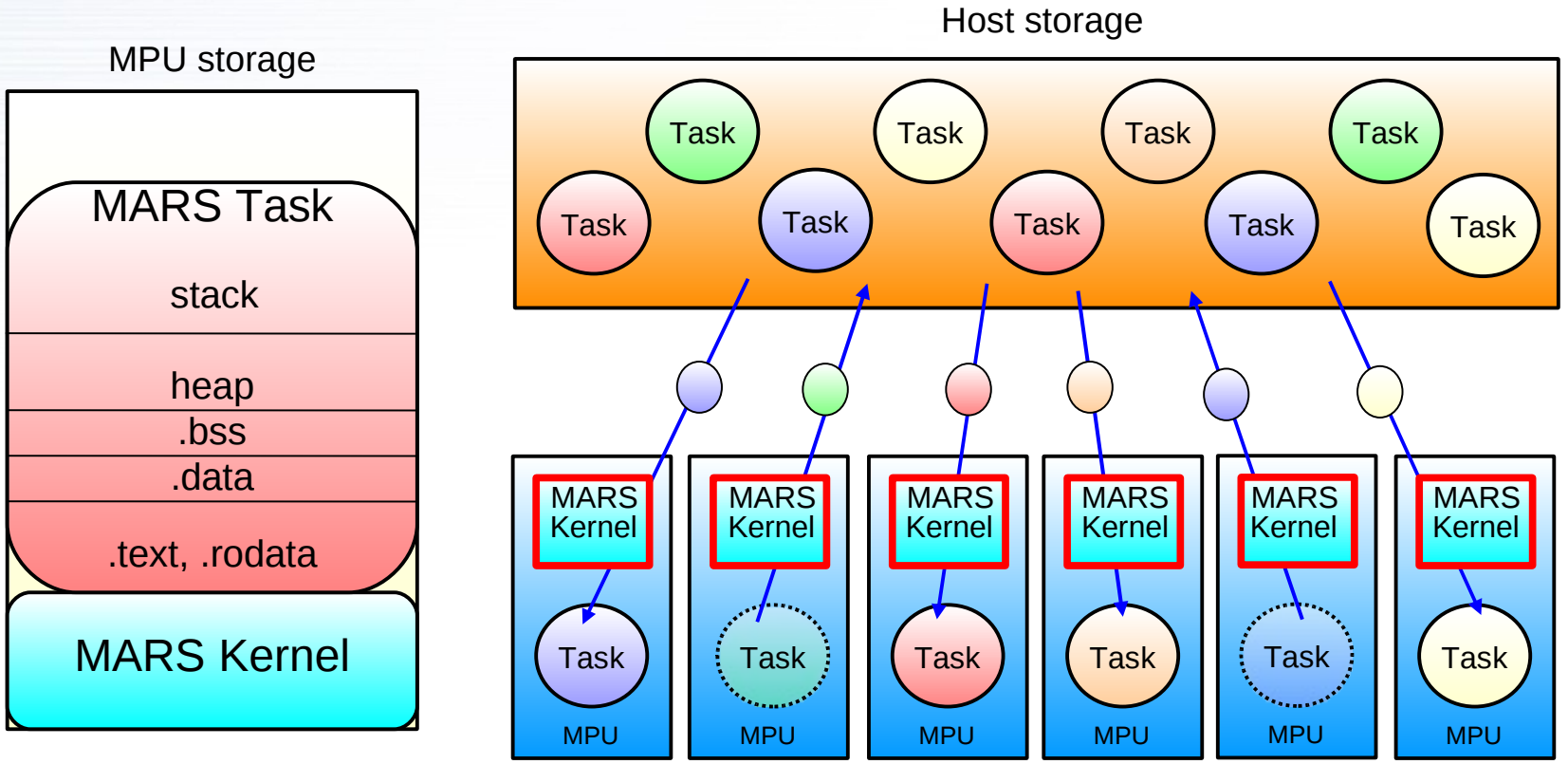
MARS Task Program

- “Hello World!” from a MARS Task

```
#include <stdio.h>
#include <mars/mars.h>

int mars_task_main(const struct mars_task_args *task_args)
{
        (void)task_args;                /* passed in by user when scheduling task */

        printf(“Hello World!\n");       /* do some useful stuff instead! */

        return 0;
}
```

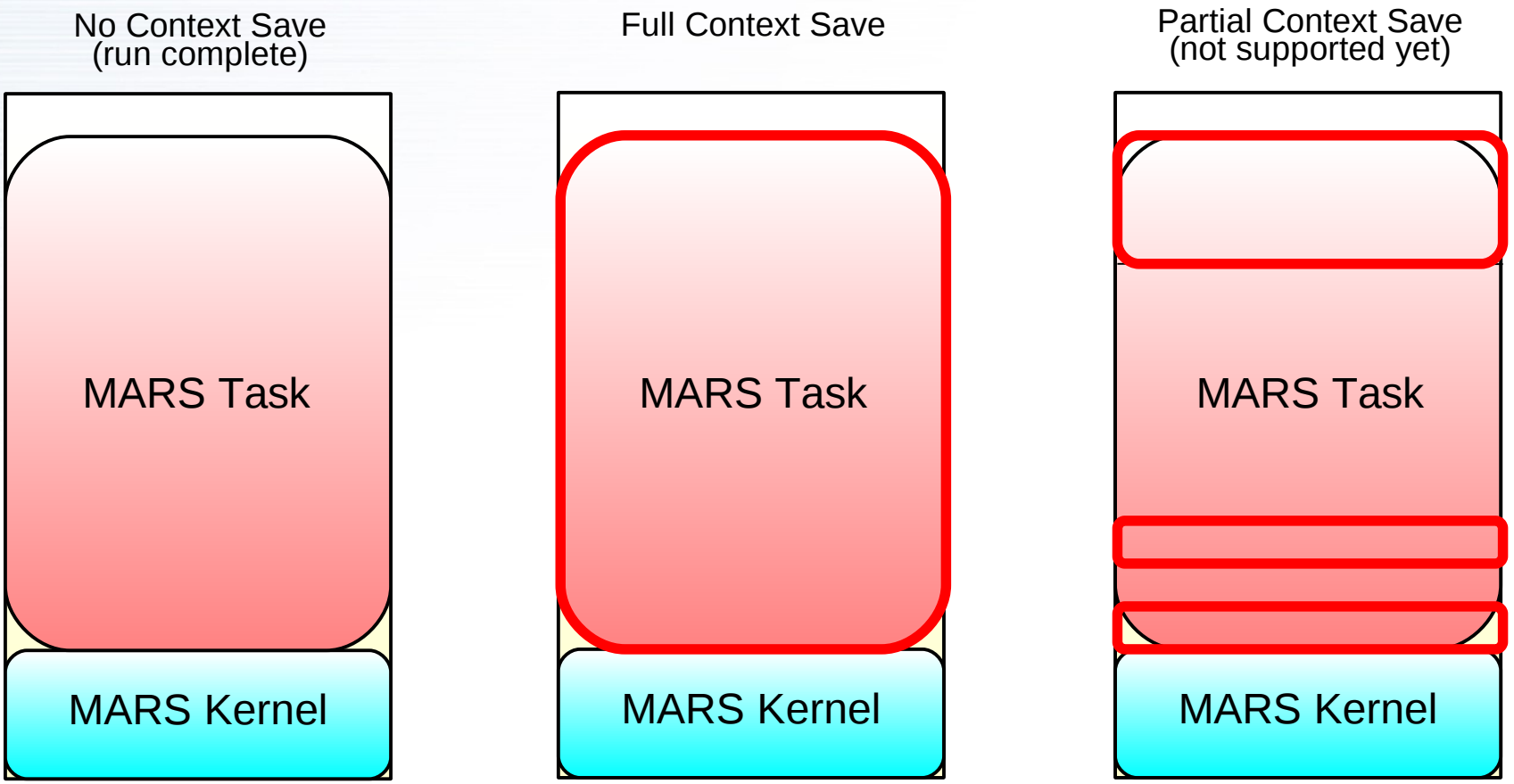
MARS Task Model

- Break up large processing into multiple smaller MARS Tasks
- Multi-task multiple unrelated tasks

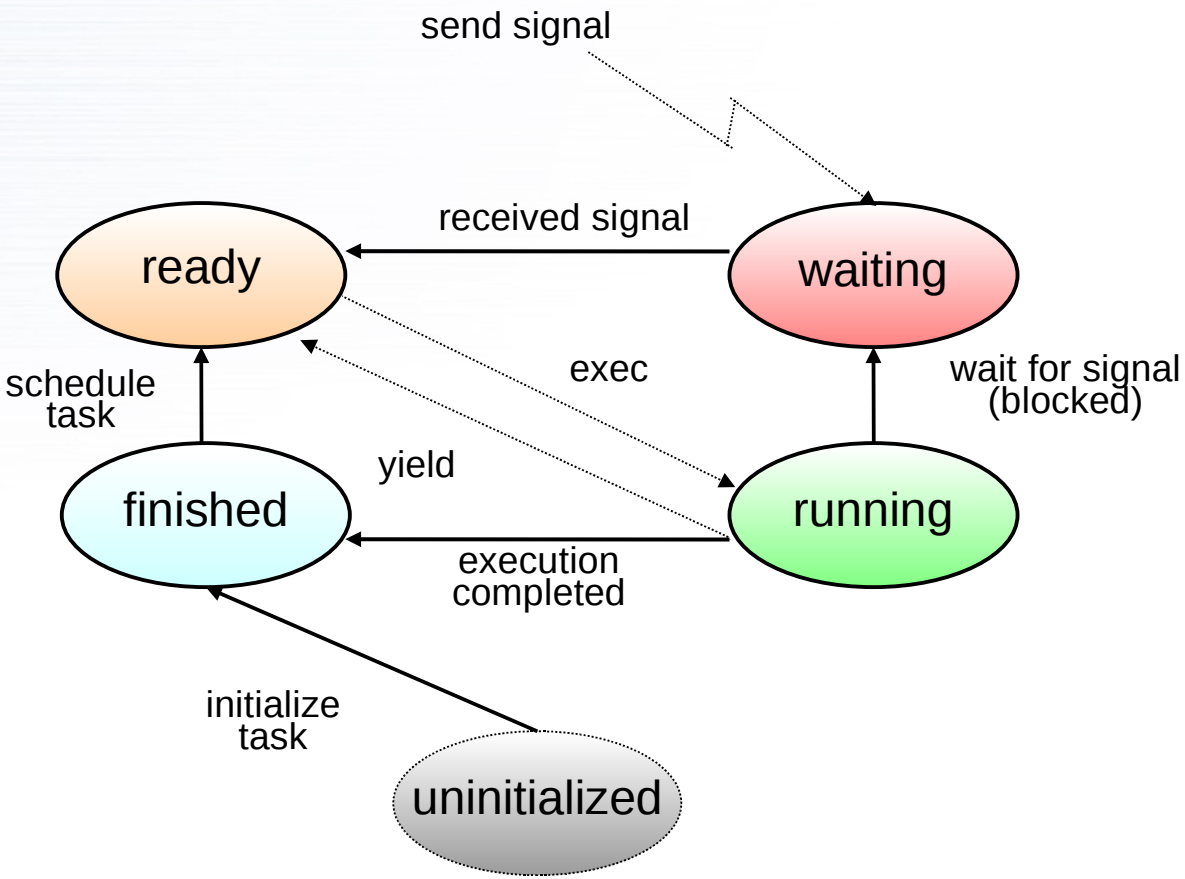


MARS Task Context Switch

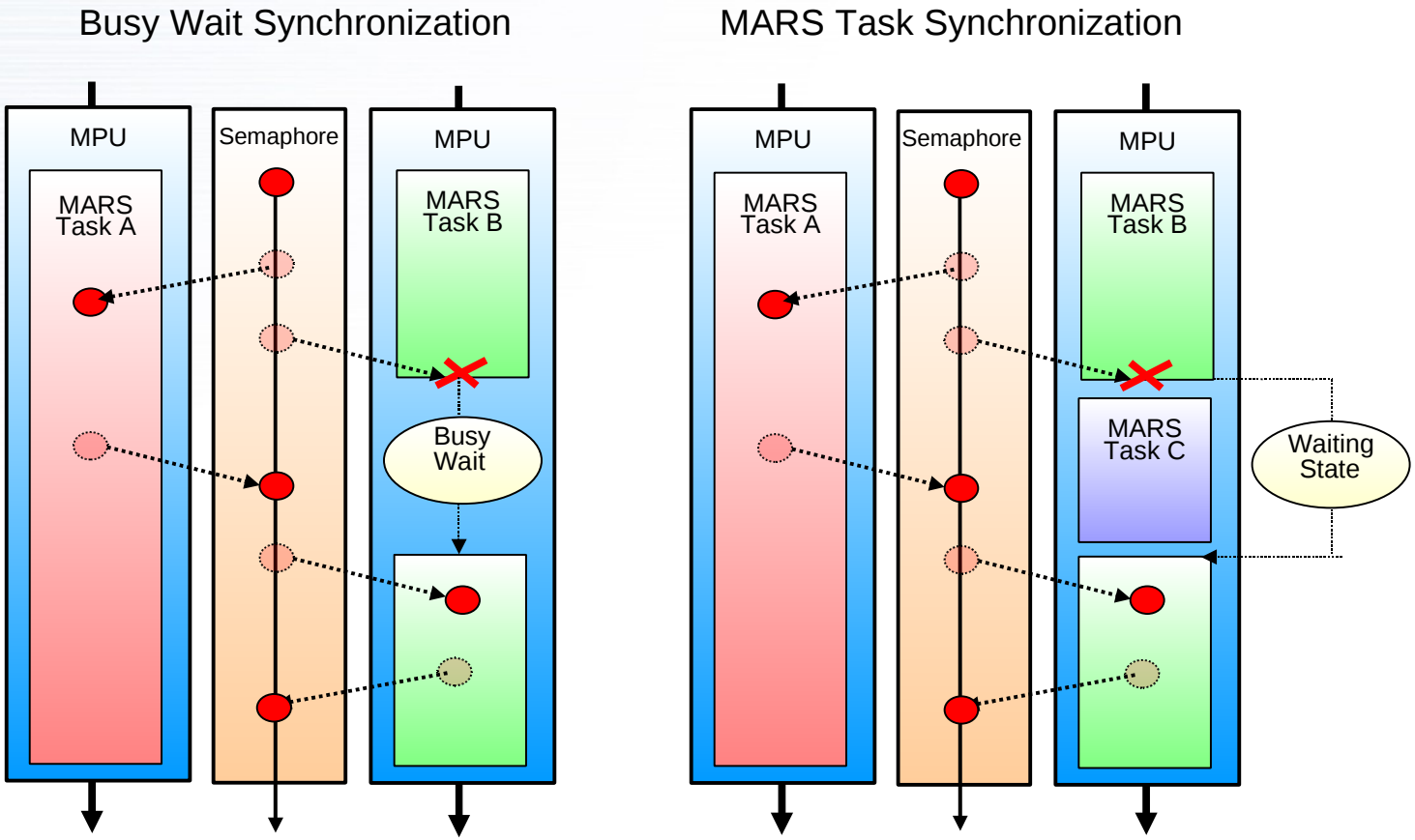
Contexts DMA'ed from/to main memory



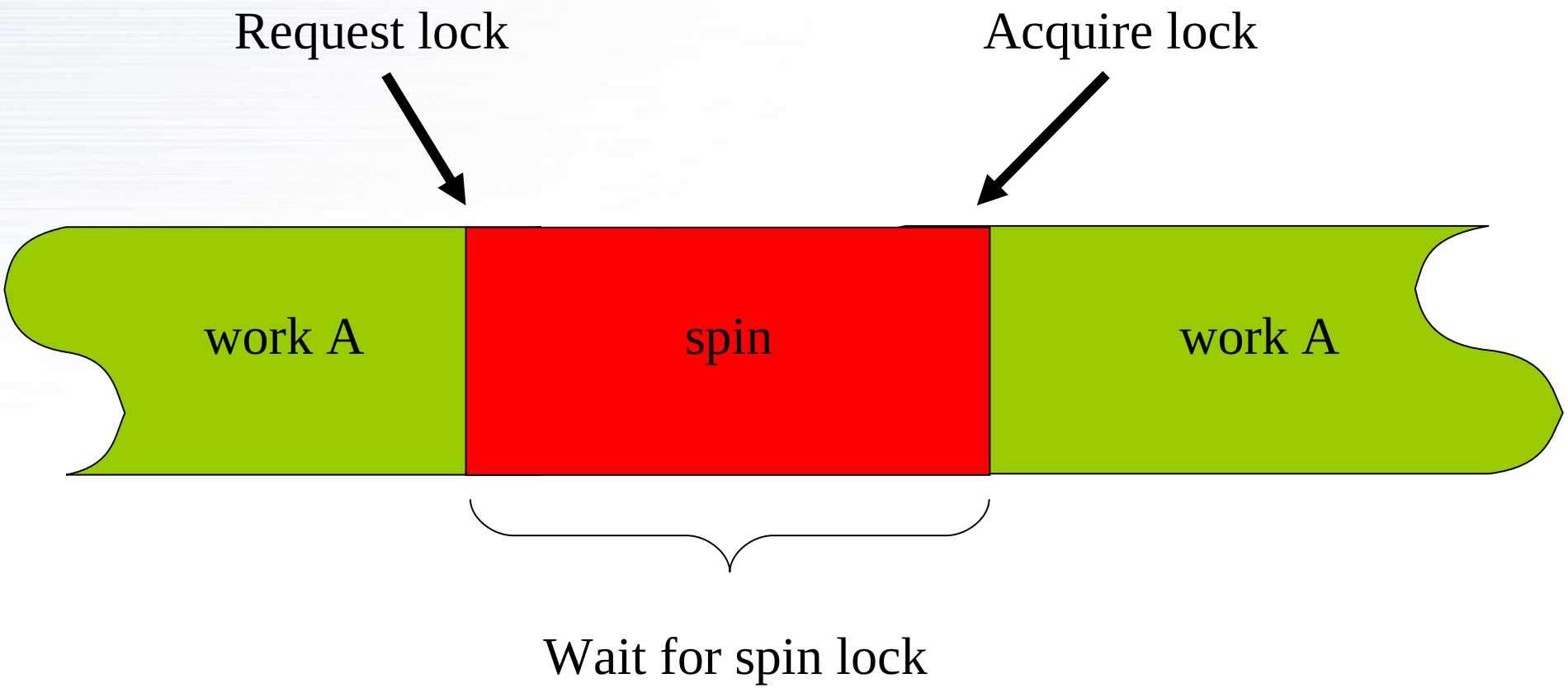
MARS Task State Diagram



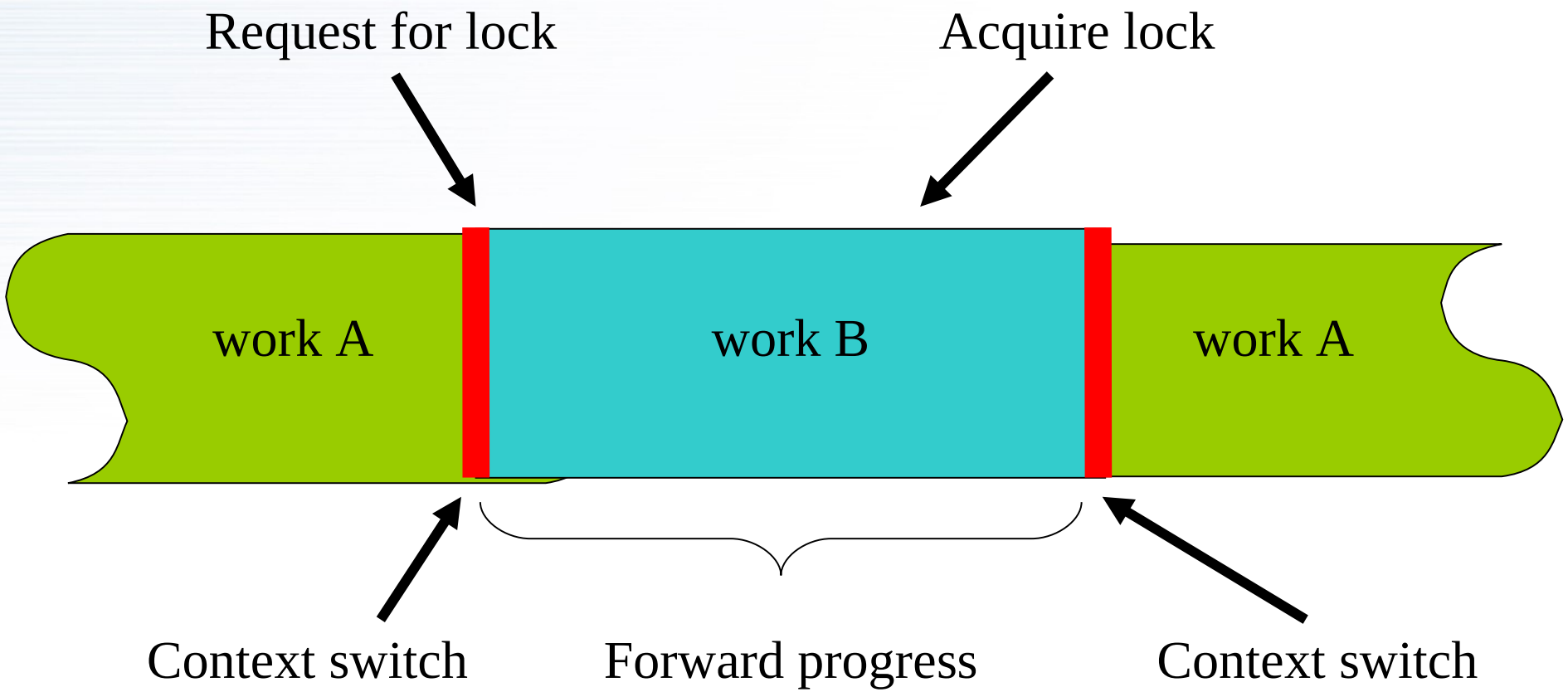
MARS Task Synchronization



Busy Wait Synchronization

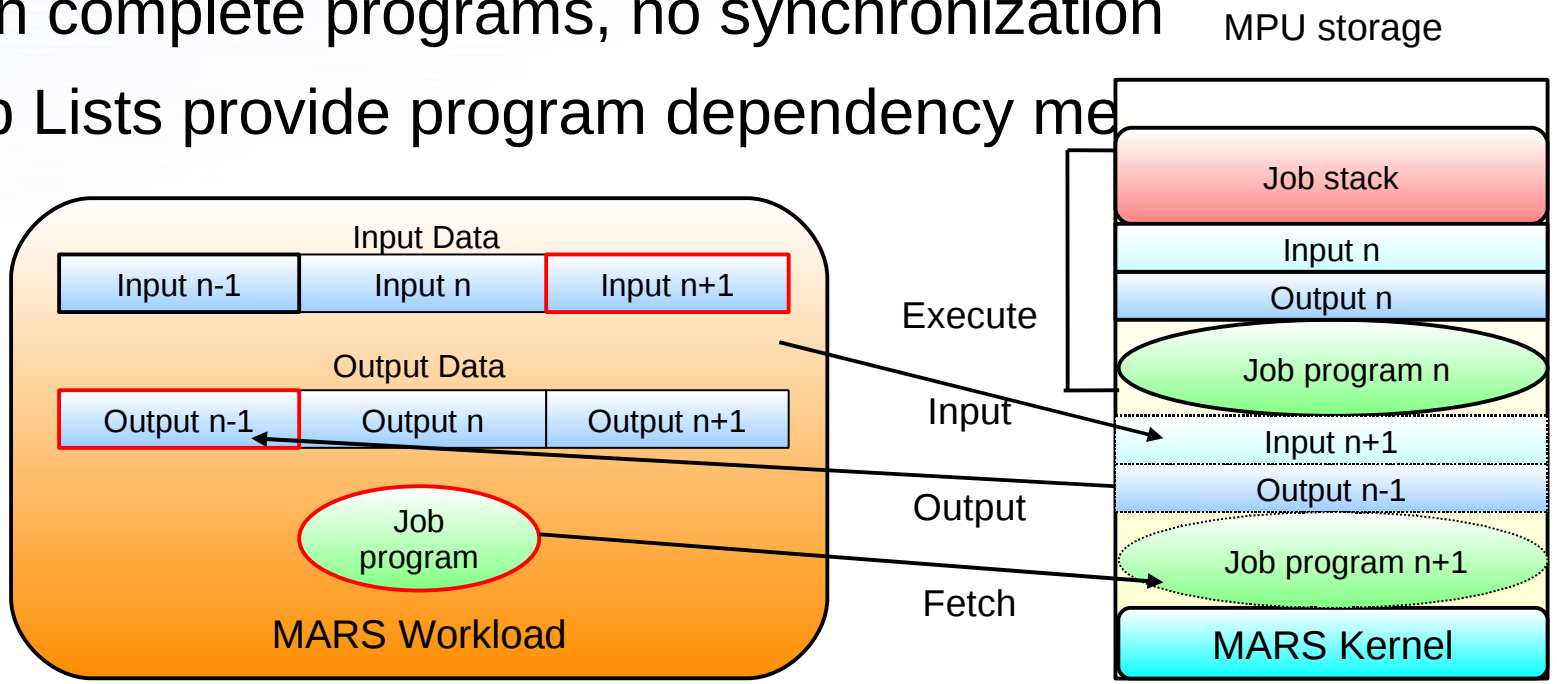


Mars Task Synchronization



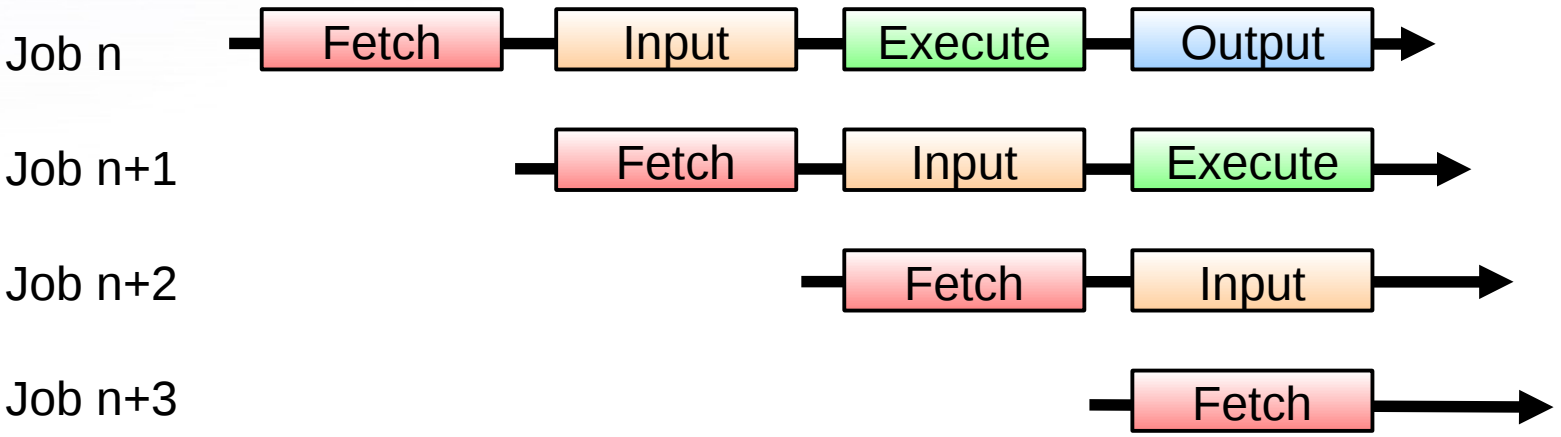
MARS Job Model
(proposed)

- Large number of programs with execution times comparable to data transfer times
- Fixed MPU input and output buffer size
- Pipelined transfer of program and data
- Run complete programs, no synchronization
- Job Lists provide program dependency me



MARS Job Model - 4 Stage Pipeline

- Fetch: Loads Job executable into MPU storage
- Input: Loads pre-specified amount of data to MPU storage
- Execute: Runs Job executable
- Output: Stores pre-specified amount of data to Host storage



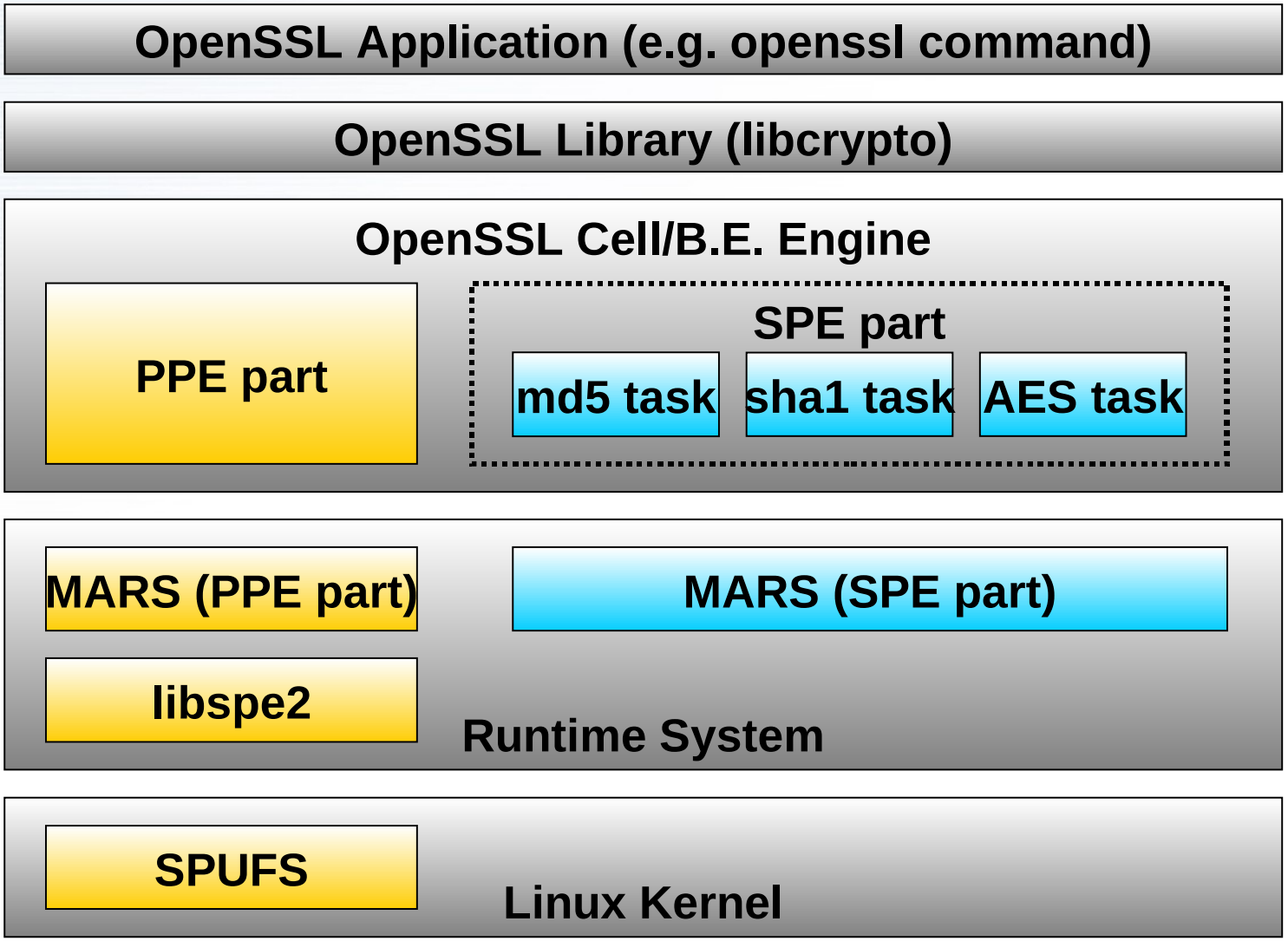
Current Status of MARS

- Mailing List discussions: cbe-oss-dev@ozlabs.org
- IRC discussions: #cell at irc.freenode.org
- Prototype source, samples:
  - ftp://ftp.infradead.org/pub/Sony-PS3/mars
- Git source code repositories:
  - http://git.infradead.org/ps3
- Support for task workload model
  - APIs for task management
  - APIs for task synchronization
    - event flag, barrier, queue, semaphore, signal

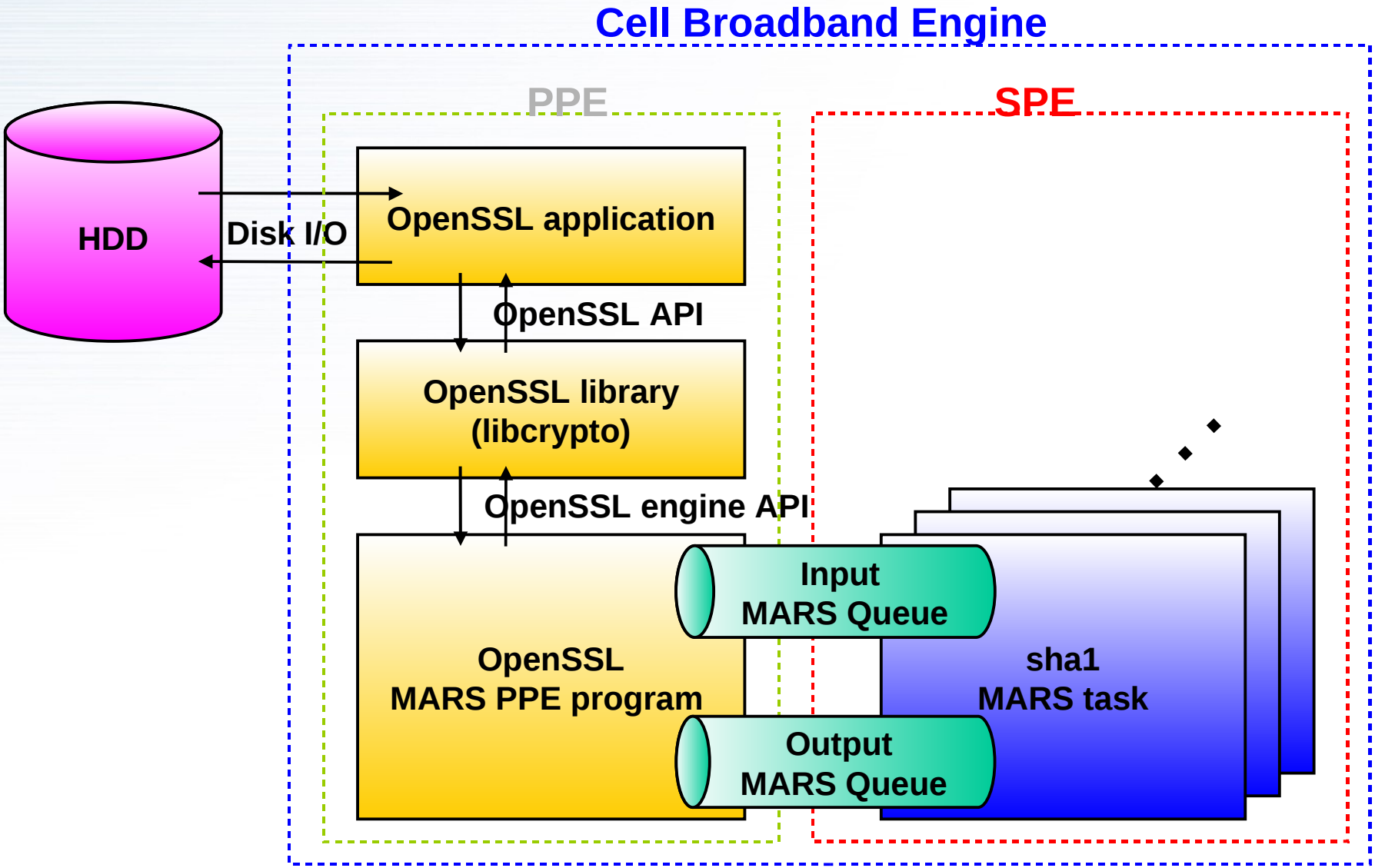
Future Plan for MARS

- Add support for other workload programming models
- MARS Task partial context save/restore
- Performance optimizations and feature improvements
- Test suite
- gdb debugger support

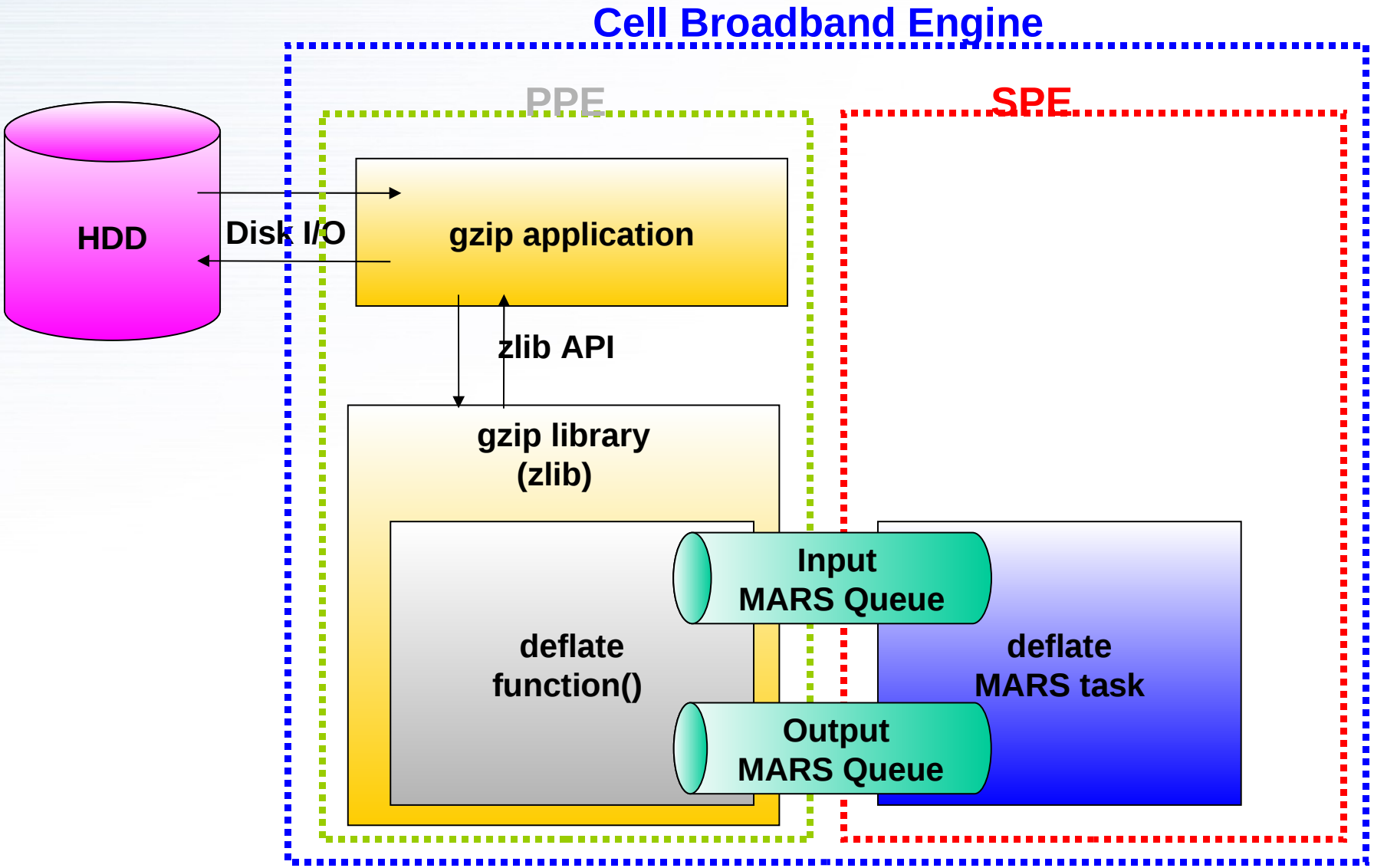
OpenSSL for MARS Software Stack

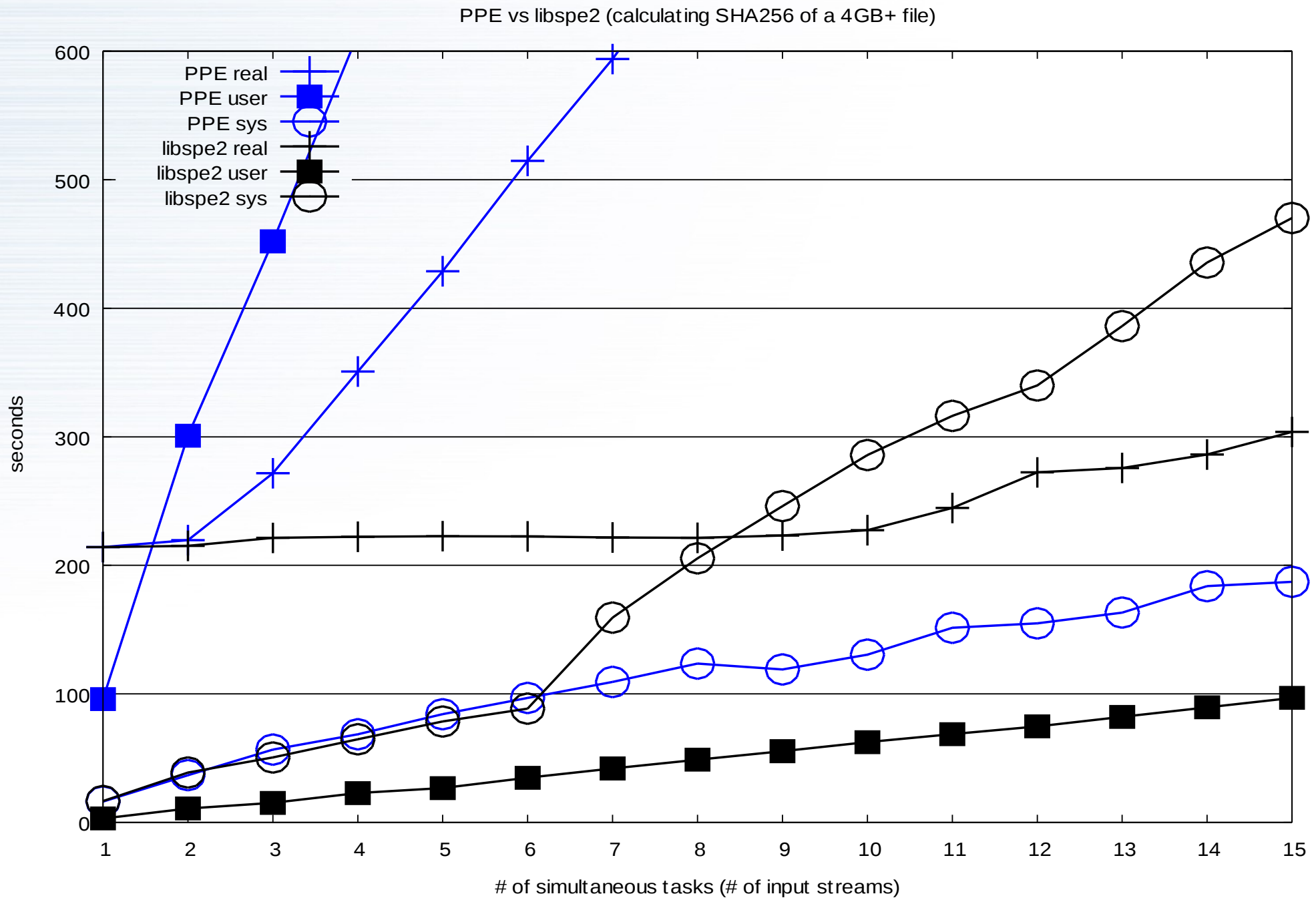


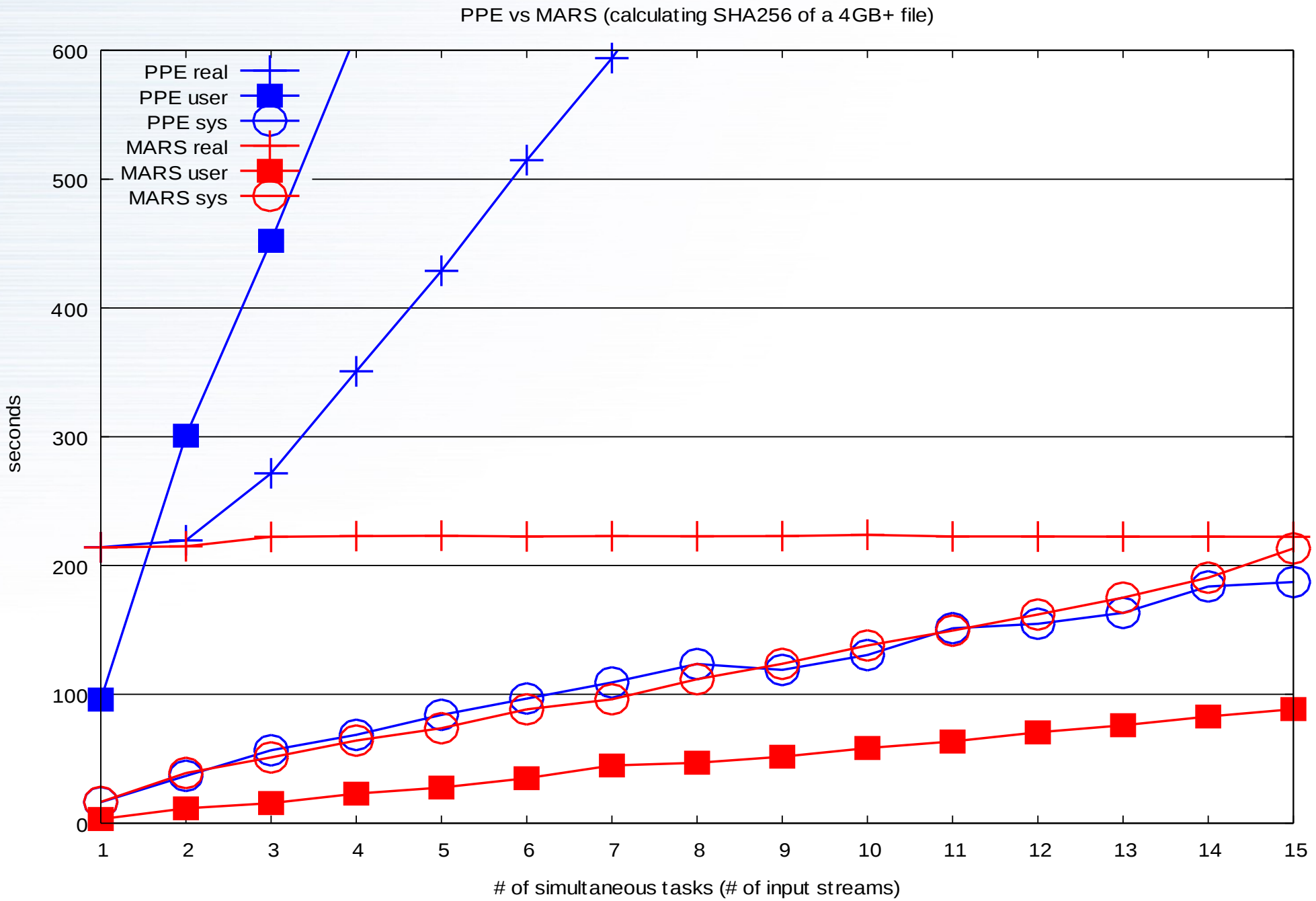
OpenSSL for MARS Overview

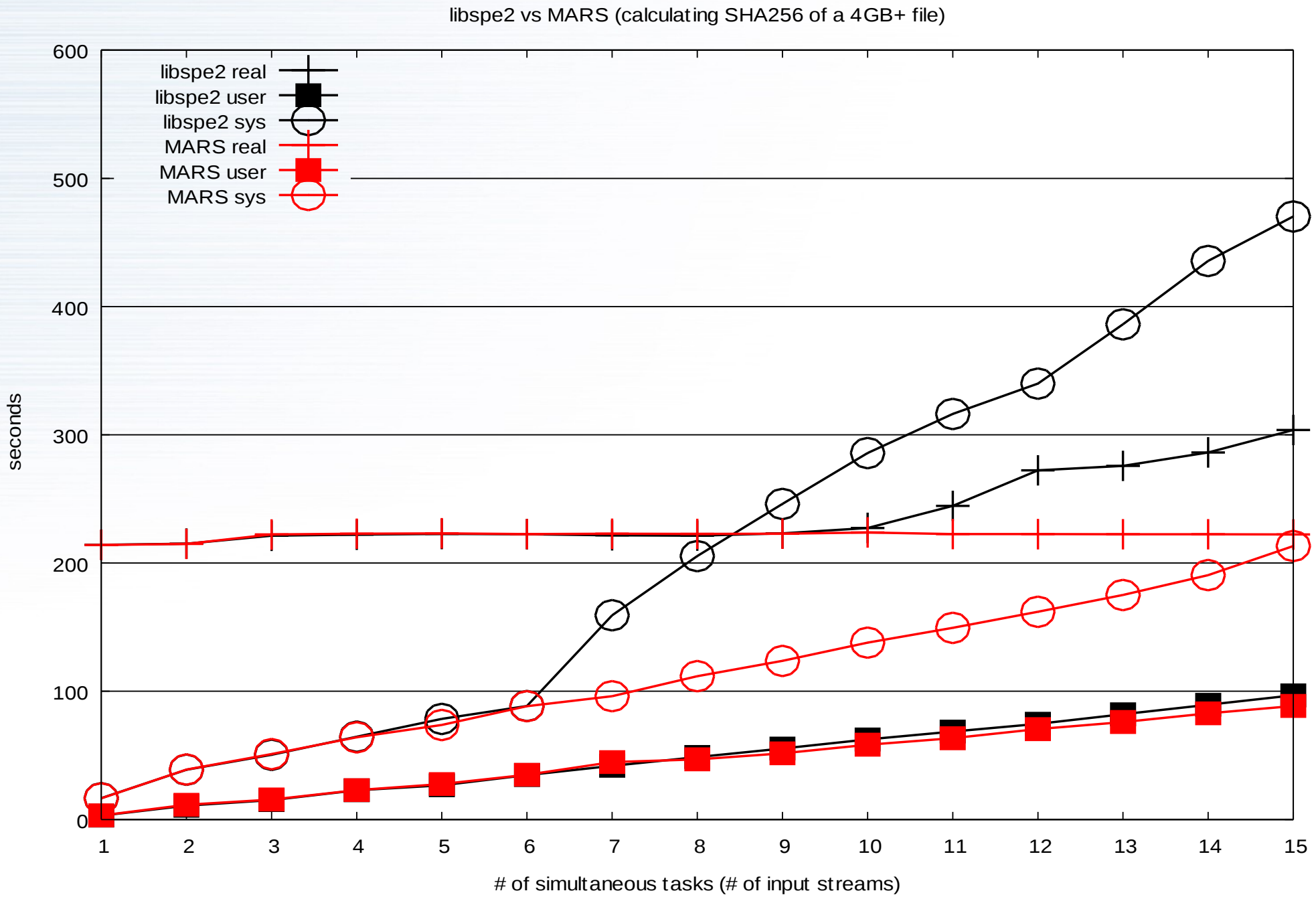


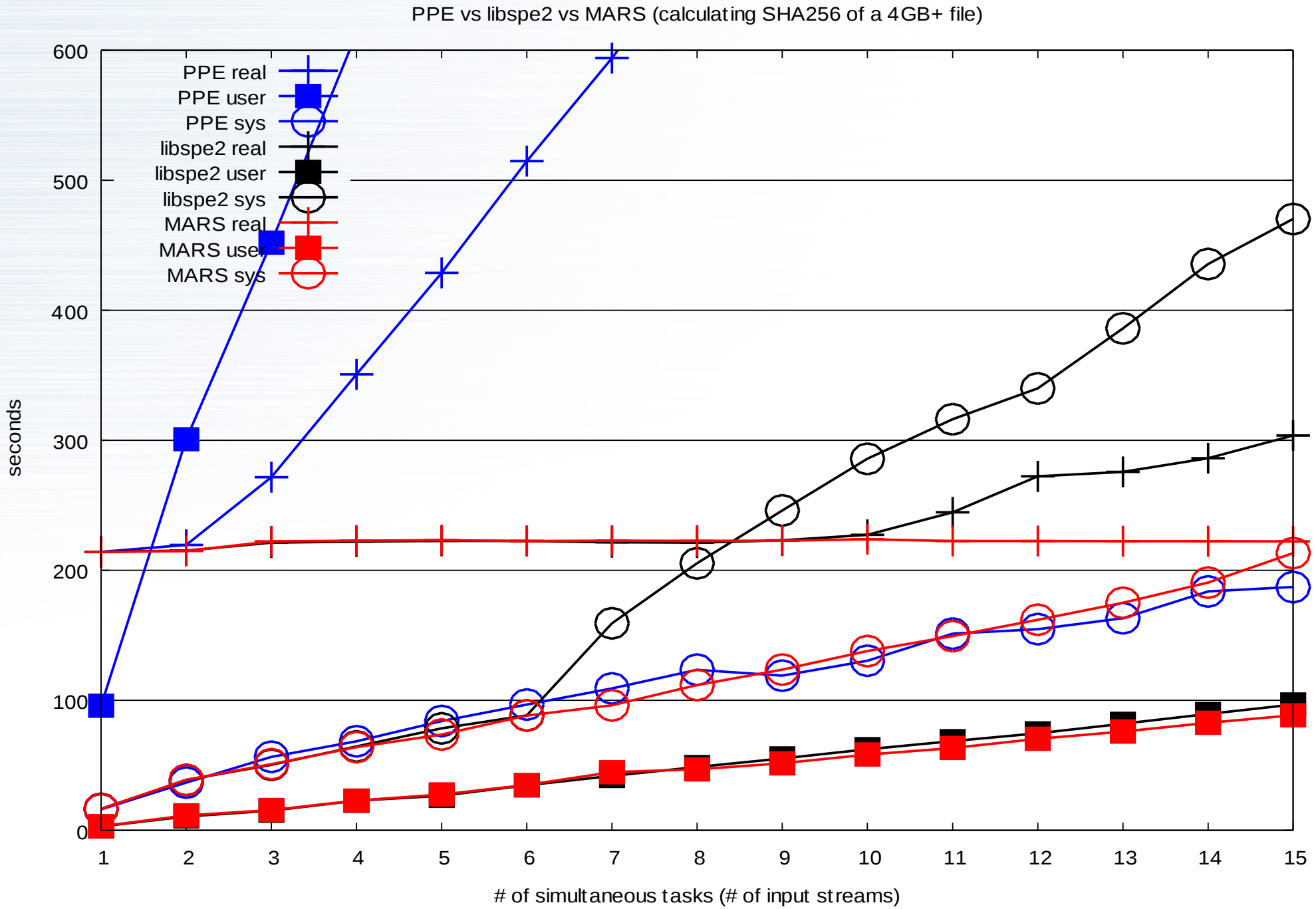
zlib for MARS Overview











Thank You

Geoff Levand
<geoffrey.levand@am.sony.com>

