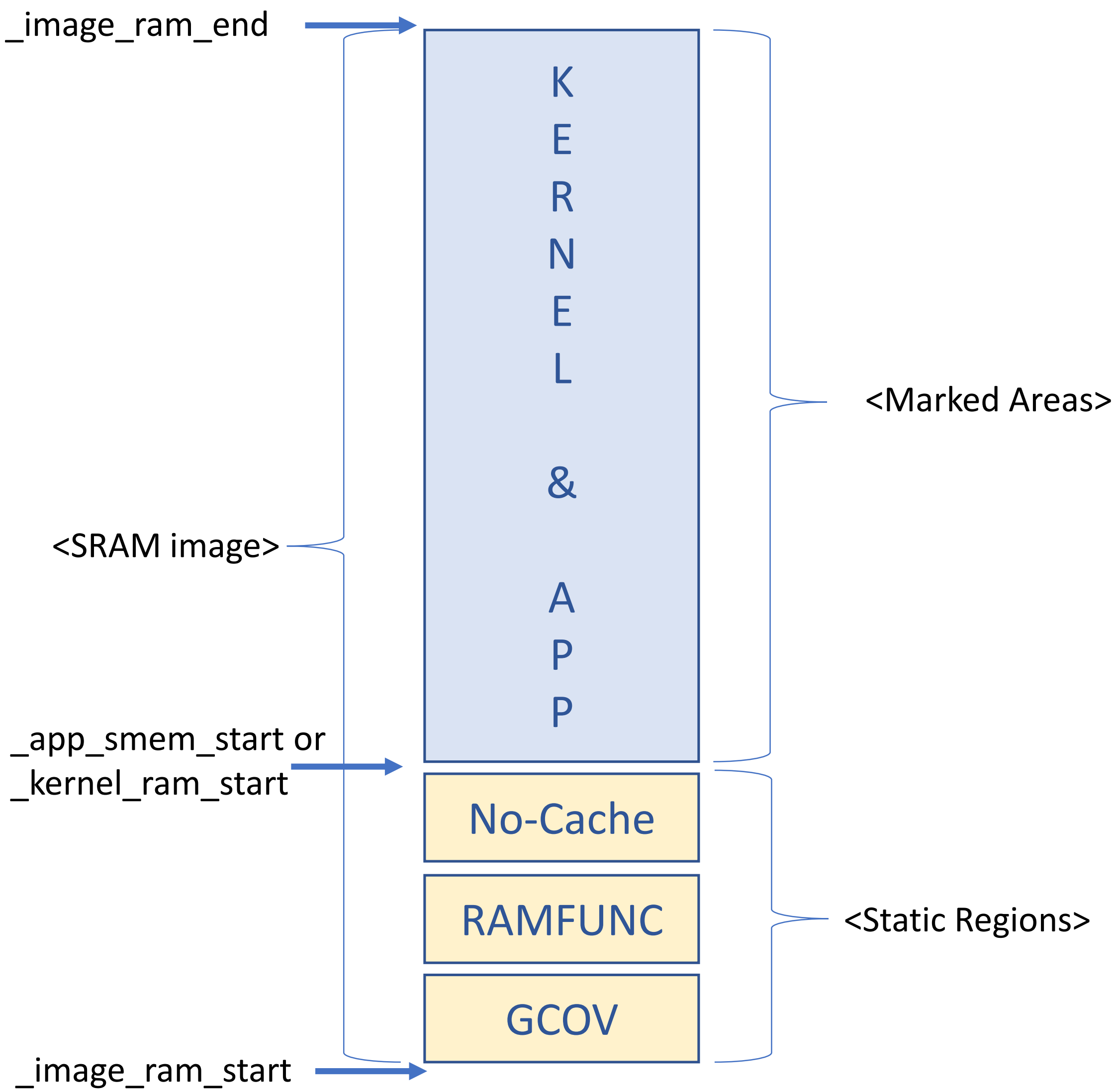
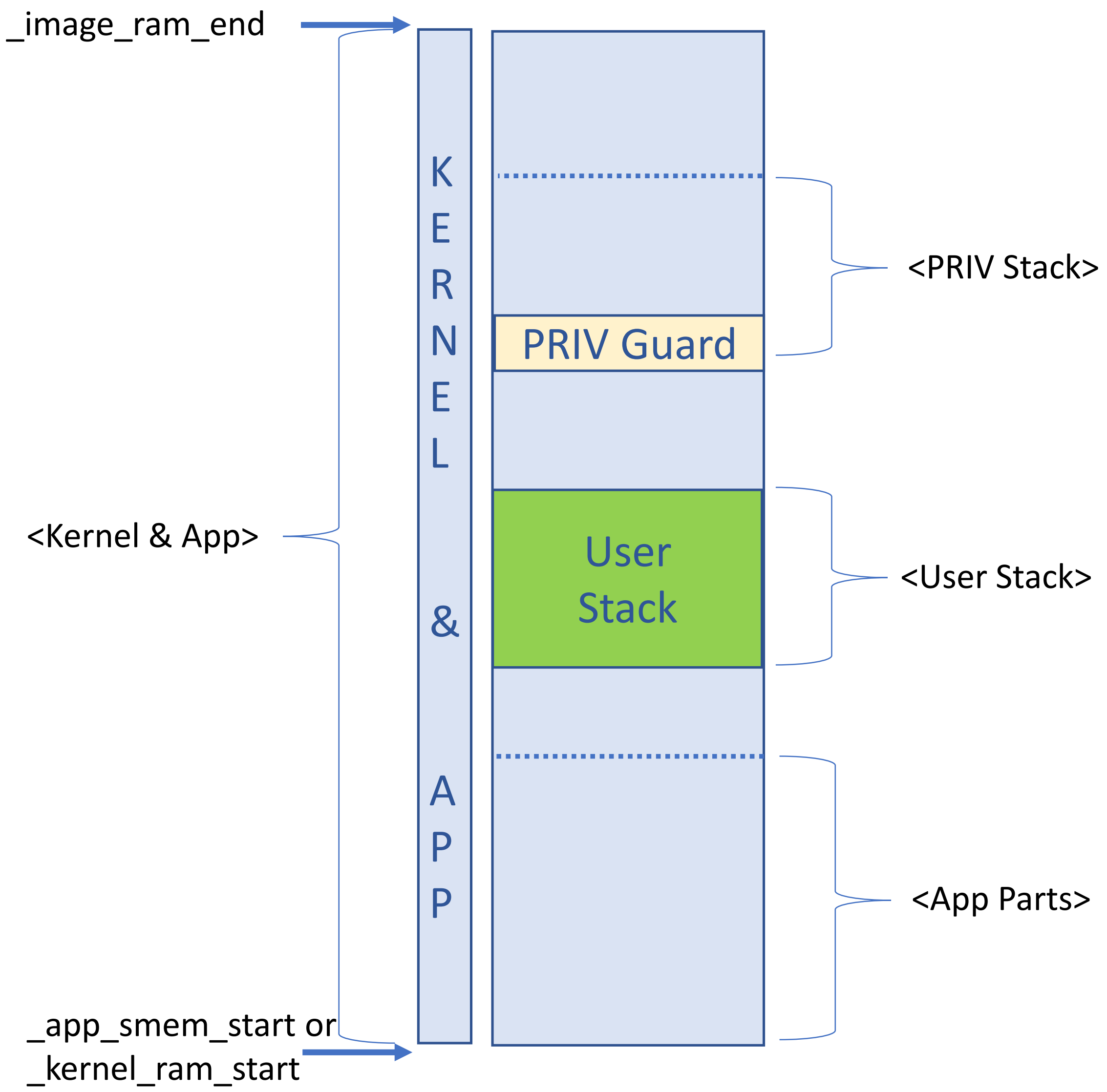
SRAM Partitioning

- SRAM split into two parts
  - Areas for static regions (RAMFUNC, etc.)
    - Special permissions, depending on usage
  - Areas marked for dynamic MPU re-programming
    - I.e. for Thread stacks, application partitions, guards, etc.
    - **PRIV-RW/nPRIV-NA** permissions
- This configuration shall not change after boot
- The static regions shall have no gaps in between
  - For optimizing the number of used MPU regions
  - Can be enforced in the linker
- We already have this in the tree
  - **1** MPU index for each Static memory region (maximum 4 in total)
  - Same for all MPU architectures, as the partitioning is proper



APP/Kernel Memory Partitioning – current state

- During context-switch – program dynamic regions:
  - App memory partitions
  - **Supervisor threads**: Guard region
  - **User threads**: PRIV Guard region and User thread stack region
- This configuration shall not change until the next context-switch
  - Except for APP partition removal, if applicable
- We do **not** re-program during system calls
- We already have this in the tree.
  - Additional MPU indices required:
    - ARMv7-M – **2 + <APP_PARTS>***
    - ARMv8-M – **4 + <2x APP_PARTS>****
    - NXP – **3 + <APP_PARTS>*****



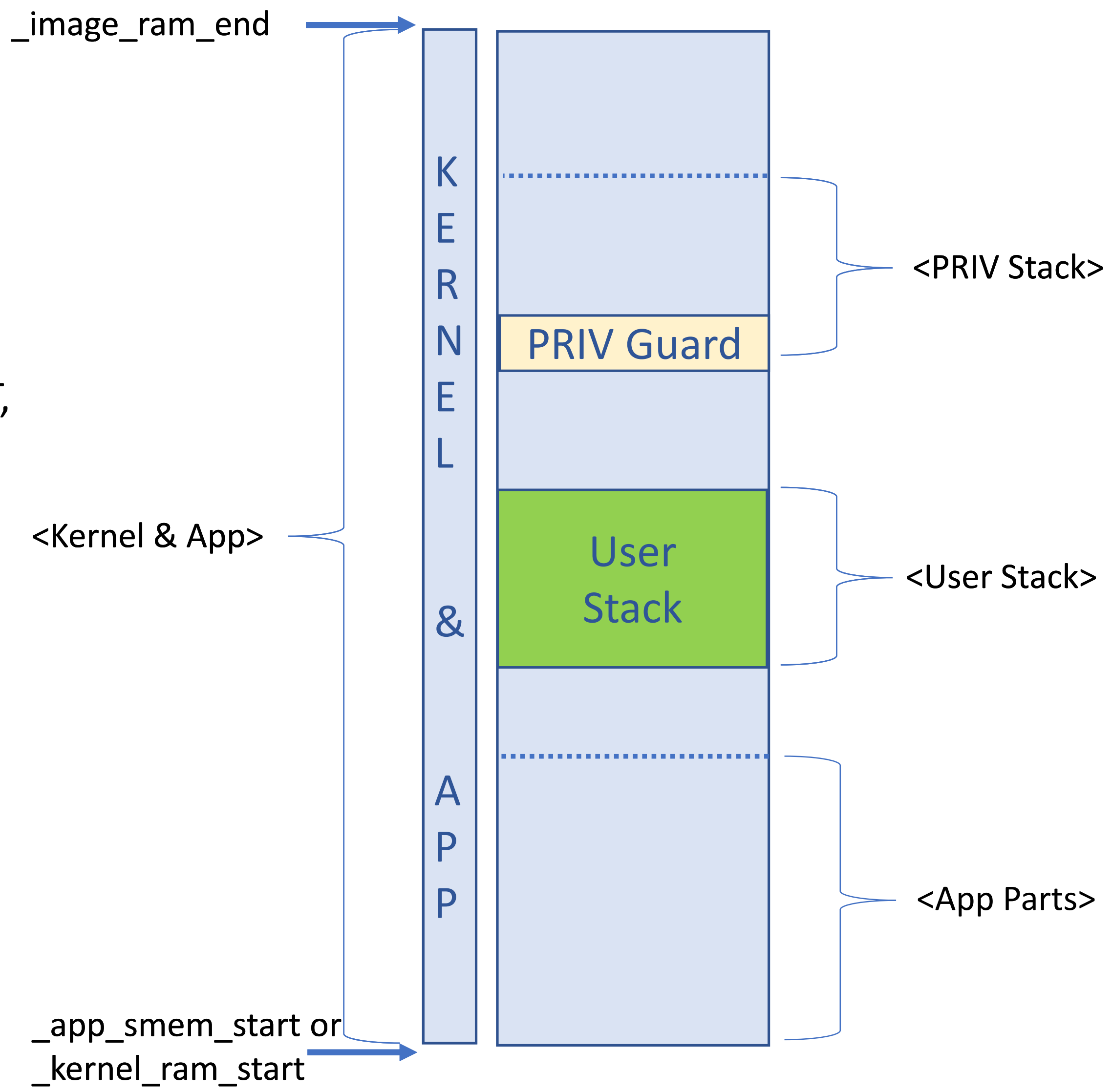
* higher-index precedence

** no-overlap

*** OR-policy

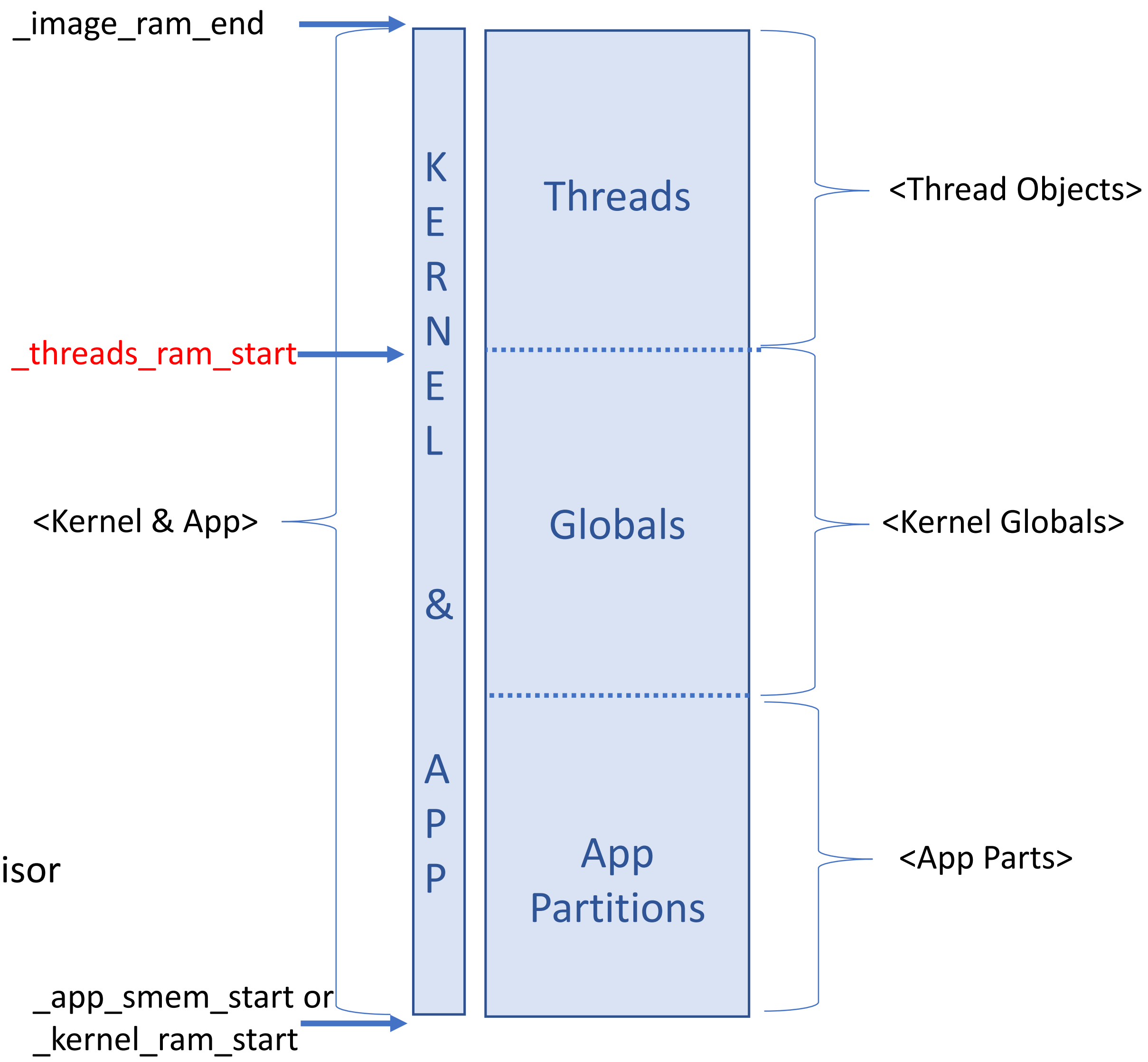
Challenges of the current solution

- Guards are, simply, wasted SRAM
- Guards need to be quite large when we build with CONFIG_FLOAT, to accommodate the whole exception stack frame
- The amount of wasted SRAM is proportional to the number of threads
- Efficient solution requires
  - use of as few MPU regions as possible
  - Use of as little wasted SRAM as possible



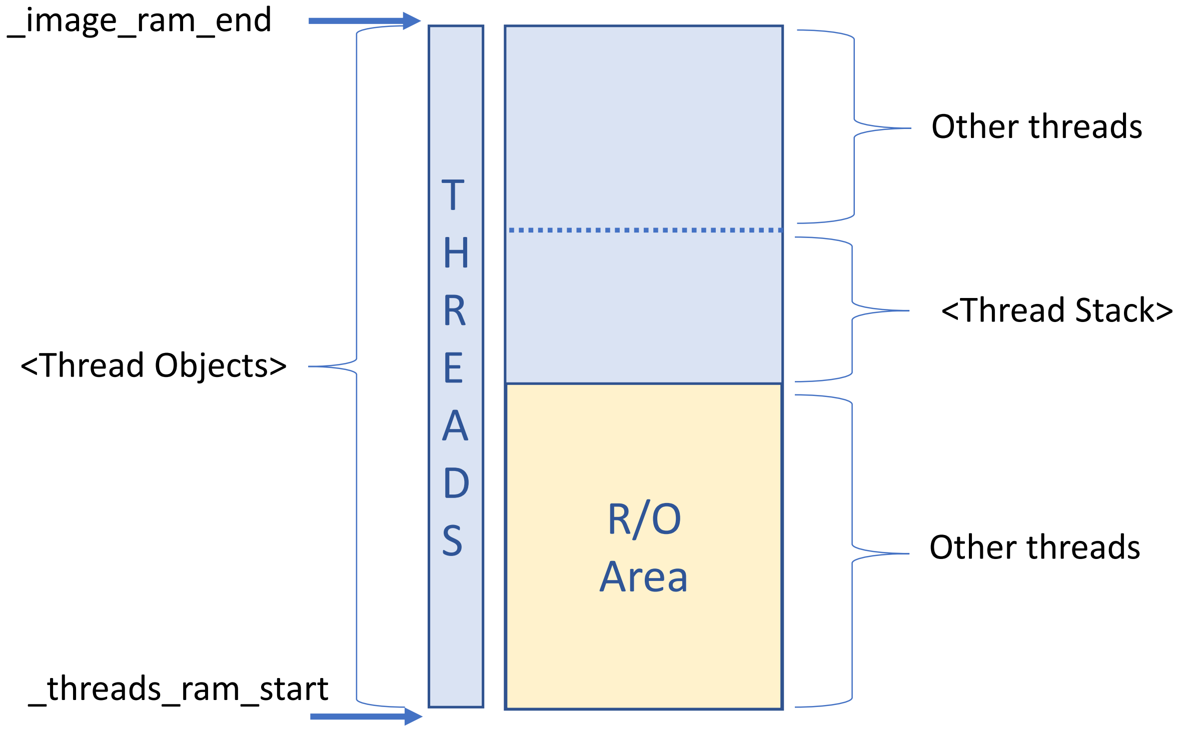
Proposal

- Idea:
  - Supervisor threads shall also be sand-boxed into own areas
- Implementation
  - Thread stack objects allocated in separate section
  - Some MPU-programming can occur in system calls
- Assumptions/Prerequisites
  - Stacks are fully descending
  - The existing proposal to unify the areas of user and privilege stacks is implemented
  - Threads only need access to own, stack, kernel globals (supervisor threads only) and application memory



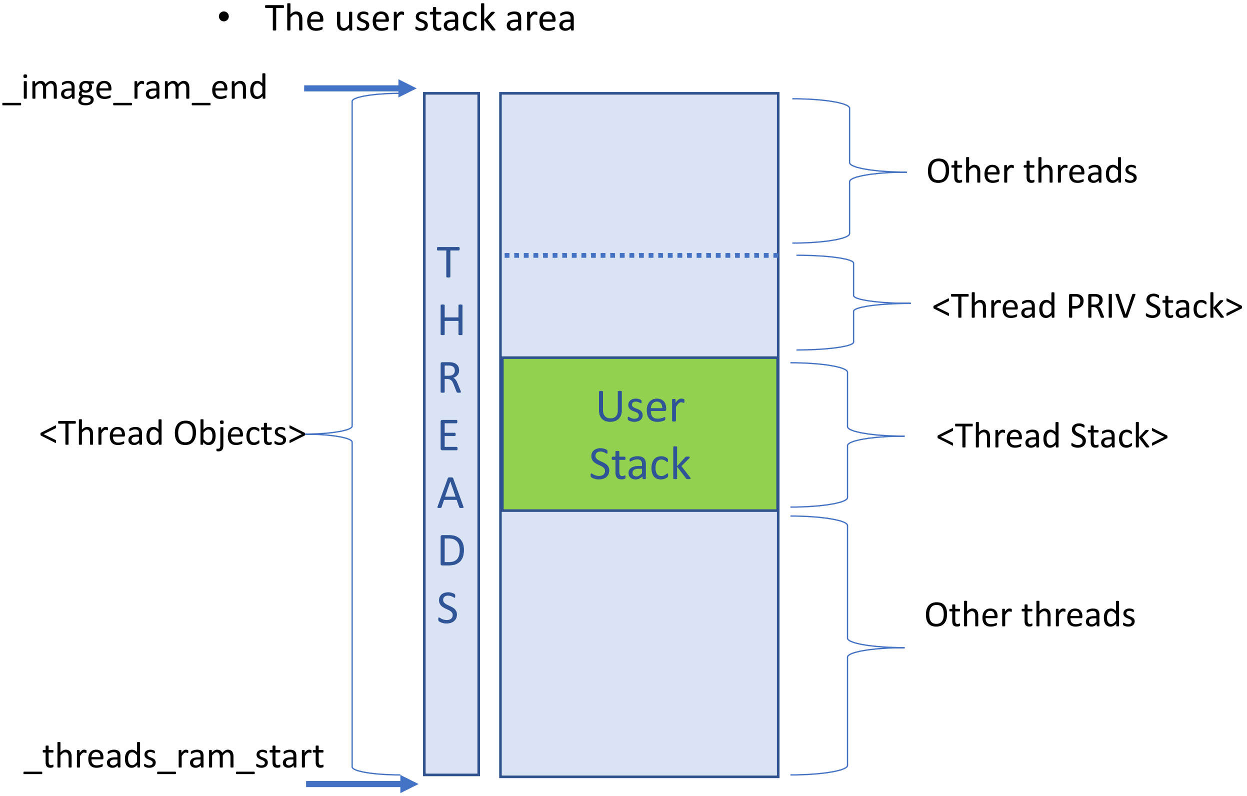
Proposal implementation details (1/2)

- MPU re-programming for **supervisor** threads during **context-switch**
  - Application partitions (as usual and if applicable)
  - The area below the thread's stack – until the start of the threads' linker section – as a "big" read-only guard
- 2 MPU regions for NXP, ARMv8-M, 1 MPU region for ARMv7-M

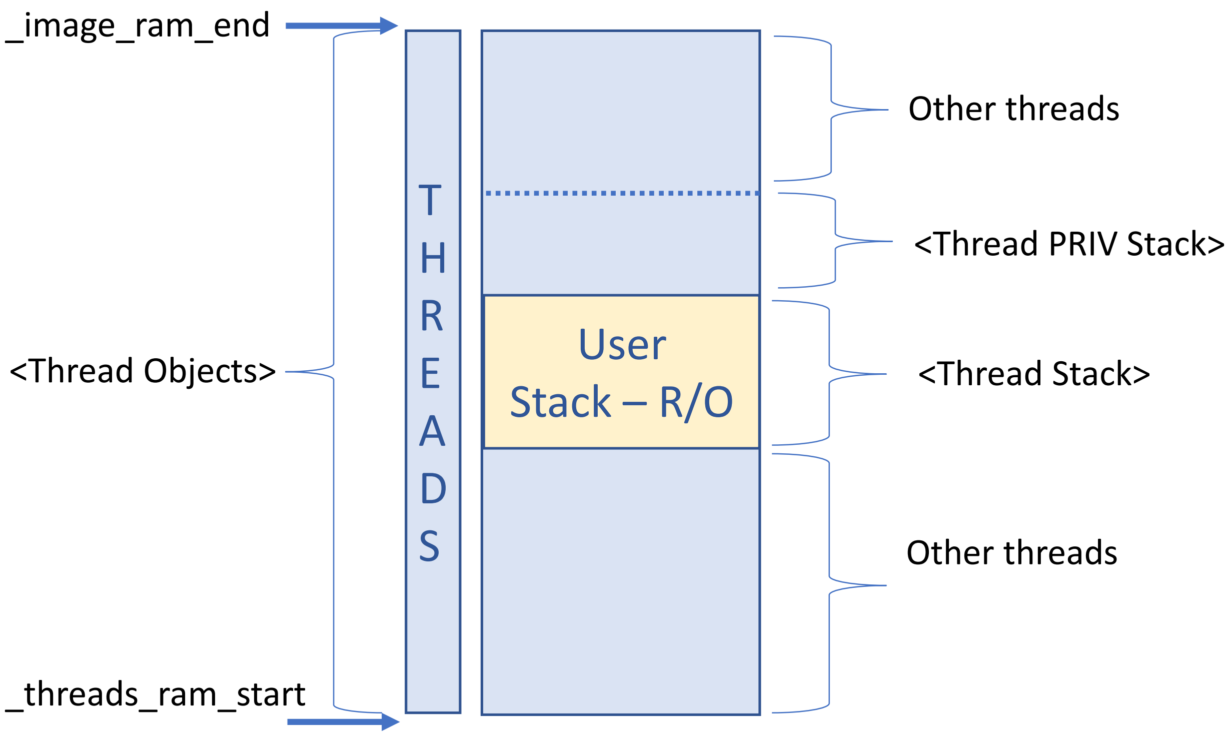


Proposal implementation details (2/2)

- MPU re-programming for **user** threads during **context-switch**
  - Application partitions (as usual and if applicable)
  - The user stack area



- MPU re-programming for **user** threads during **system-calls**
  - Program the whole user thread stack as read-only



2 MPU regions for NXP, ARMv8-M, 1 MPU region for ARMv7-M