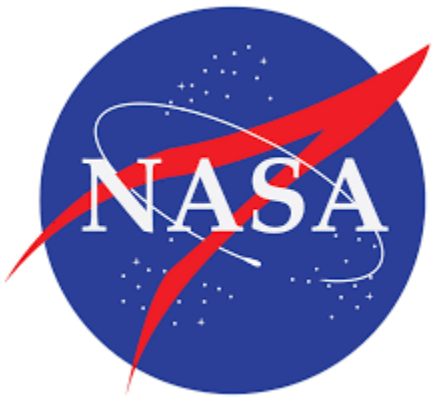
Overlay: To fit any oversize program to the execution memory

Did you ever think how wonderful it can be if your device will not be limit your program size? A device that can contain any oversize program.
We have MMU and OS that can do paging, but what if we have a small embedded device, an IoT controller that does not have room for HW MMU or a fat OS kernel.
Well, there is a solution, a software solution that does not involve any use of HW component.

In the early days of embedded computing, there was a technique to load code in Real-Time when it was needed for execution. This technique was named **Overlay**, and it was threaded with the toolchain (compiler, liker, and more), providing an accessible application interface for the SW developers.



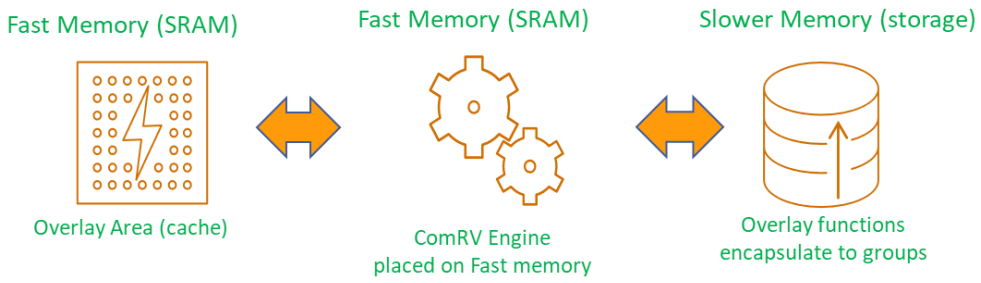
NASA used this technique in their early shuttles flight control system. Flight computers had very limited memory, so they used SW overlays – only a small amount of code necessary for a particular phase of the flight (e.g., ascent, on-orbit, or entry activates) is loaded in computer memory at any one time. At quiescent points in the mission, the memory contents are "swapped out" for program applications that are needed for the next phase of the mission.
Read more at Statistical Software Engineering (1996) - Chapter: Case Study: NASA Space Shuttle Flight Control Software

This technique slowly disappeared because memories became bigger, cheaper, and MMU emerged, replacing the need for paging software solutions.

Today, IoT devices (Internet of things) are very strict with memory size and power, alongside requirements for simple HW implementation which does not contain MMU or a high-end operation system to manage it (Linux/windows)

This technique gives the flexibility in reducing physical memory, and therefore reduces energy and space. It saves memory in magnitude proportions over any solution in the ISA domain (E.g., on RISCV: eABI, code-density, bitmanip, etc.…)

So how does it works?



A run-time module operates on the fast memory, deciding which function to load from a storage device to a fast memory heap.
Code is dynamically loaded to “cache” and executed according to the program flow.
The run-time module manages the cache and invoking the overlay functions.
The software engineer experience is very smooth; they need to add an *_attribute_* to the function designated to be an overlay function, a function to load and execute only when called, and that is it.

For example:

```
void __attribute__ ((overlaycall)) bar(void);

void foo(void)
{
        bar();
}
```

In contrast to MMU, Software Overlay is not using direct mapping; it means we can almost fit and program size to a very small heap/cache area. For example, 16Kbyte of cache can fit Megabytes of code

FOSSi Foundation and Overlay:
Western Digital took the Overlay task on itself; to standardize and develop an Overlay application for RISC-V to provide a solution for code size limitations in the RISC-V targets.

RISC-V organization noticed that Overlay could be a good solution for any target and advised open-source for everyone.
At that time, a task group was created in the RISC-V org names overlay-tg; the goal is to provide a full application suite for Overlay, from the toolchain to a driving SW engine.
Alongside, the Task-group created an official overlay standard that fits any target. It means a unified specification for Enginee and Toolchain applications that can be used for any target.
Lately, this standard was ported to FOSSi Foundation. The first step on porting all the Overlay suite to FOSSi Foundation.

Searching for Overlay applications on the web will quickly show no results. Especially not in open source and not for RISC-V. Supporting this feature on RISC-V and other targets will significantly improve the likelihood of adoption Overlay in smaller IoT.