Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Multiple Personalities

Thoughts on a Virtualized RIOT

Background

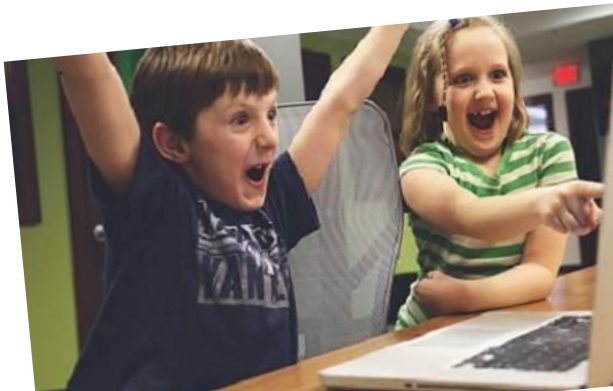
- Last year we told you why RIOT (sometimes) sucks
- This year we try to be a bit more productive (promise!)

- What do we mean by virtualisation?
  - Virtualization is an overloaded term nowadays
    - Storage, network, execution environments, virtual reality, …
    - We are talking about **system virtualization**
    - Container/partitions on top of a µKernel
- Alternative title: Why RIOT OS should become a *real* µKernel OS?

Motivation

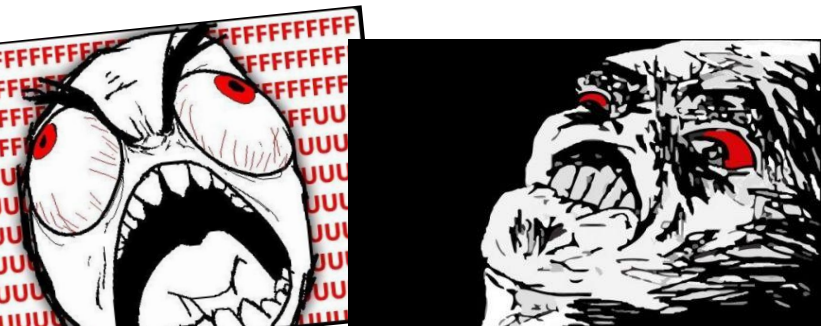
- How do we build a secure Internet-of-Things?
  - chinese manufacturers of IoT devices have an answer
  - notable exceptions (IKEA?)
- Billions of “smart” devices in the internet soon
  - unpatched, open, vulnerable, and easy to hack
  - a paradise for botnets, worms, and things we don’t even think about
- Is RIOT part of the problem or the solution?

- We need safe and secure systems!
- We need means to build these systems!
- We have to divide application concerns!
- How do we do it?



Pre-Requirements, or why is RIOT not a µKernel?

- Minimal basic OS-API ✓
- µKernel runs in kernel mode, everything else in user mode
- Memory-protection for "applications", drivers, stacks in user mode
- Not a Software issue, hardware is needed: MMU or MPU
  - So far our targets have not supported either
  - This is changing! MPUs are coming!



What is a MPU?

- Memory Protection Unit (MPU), ARM feature
  - No memory translation, everything uses physical addresses
  - Protects up to 8 memory regions (windows) with n subregions
  - Overlapping protecting windows
  - Access permissions -> violation / mismatch calls MemManage fault handler
- Memory access -> MPU checks if allowed
- MPU windows needed for
  - Kernel itself (static)
  - Partitions code and data (changes every partition switch)
  - Specific windows for tasks (usually stack, changes every task switch)
- Switch by config of MPU registers in kernel mode
  - Needs time
  - Should be static (as possible)
- Specify this config is the primary nightmare...

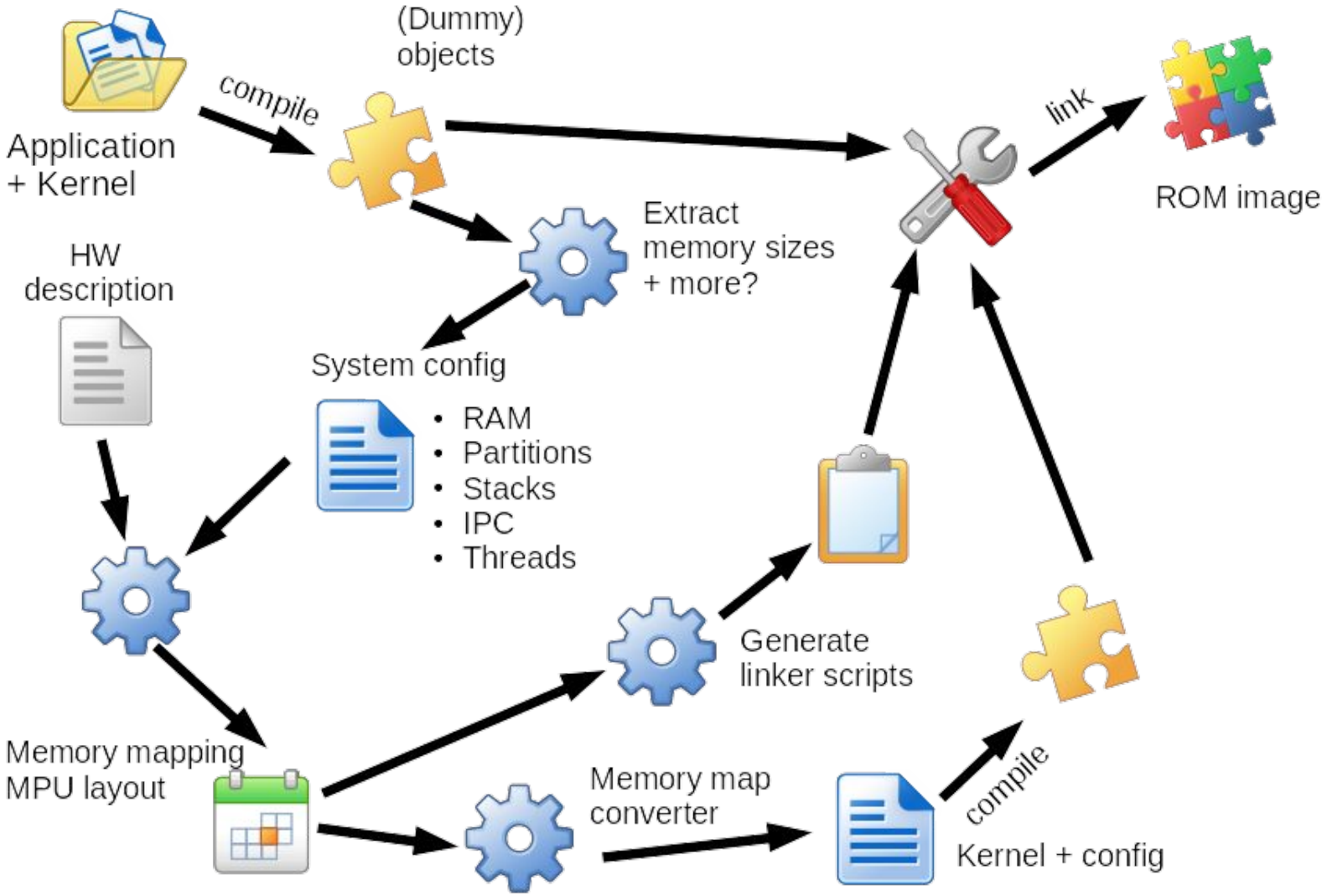
How to slice the cake ^m^m RIOT OS?

- Assumptions:
  - MPU hardware support (MMU would be “degraded” to MPU functionality)
  - Relativ static system (kernel / app resources are known at build time)
- Iterative approach / implementation should be possible
- Where to slice? (easy...)
  - Stacks (stack overflow protection, already started in RIOT OS)
  - Kernel and one application domain
  - Multiple application container / partitions
  - µKernel -> driver, stacks, modules in own partitions
- But how?

One possible approach (“borrowed”)

- Approach from AUTOBEST (r&d project by A. Züpke, R. Kaiser et.al. with easycore GmbH, Erlangen)
  - Minimal configuration, use existing information in the code
    - RAM, partitions, stacks, IPC, threads, ...
  - Generate what is needed (MPU / system config, linker scripts)
- Open questions / possible problems:
  - How good is the separation of modules, functionality lived in RIOT?
  - Only upper OS API used, or are there cross reference calls in the kernel itself?
  - How about the drivers? Or the stacks?

One possible approach (“borrowed”)



But why all the hassle?

- Helps the application / kernel development
  - Isolation of errors
  - More control of the resources
- Groundwork for more research
  - Resource optimization
  - (Partial) updates
  - Security (how about MILS?)
- Be able to call RIOT a µKernel on a conference without blushing?
- Because RIOT should be part of the solution and not the problem!

This is not the end

What does it mean for RIOT?

- Will there be a riot after this talk?
- It means changes and effort and tears
  - Like going from “Bolzplatz” to the amateur league
  - Code has to be refactored
  - Existing structure requisitioned
  - Probably there might be (even) more API-CHANGES!!!
- But is it doable? We think yes!
- But is it it worth? We say YES!!!!11!!