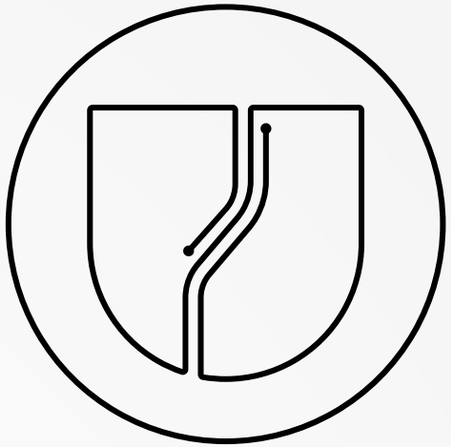
Best Practices

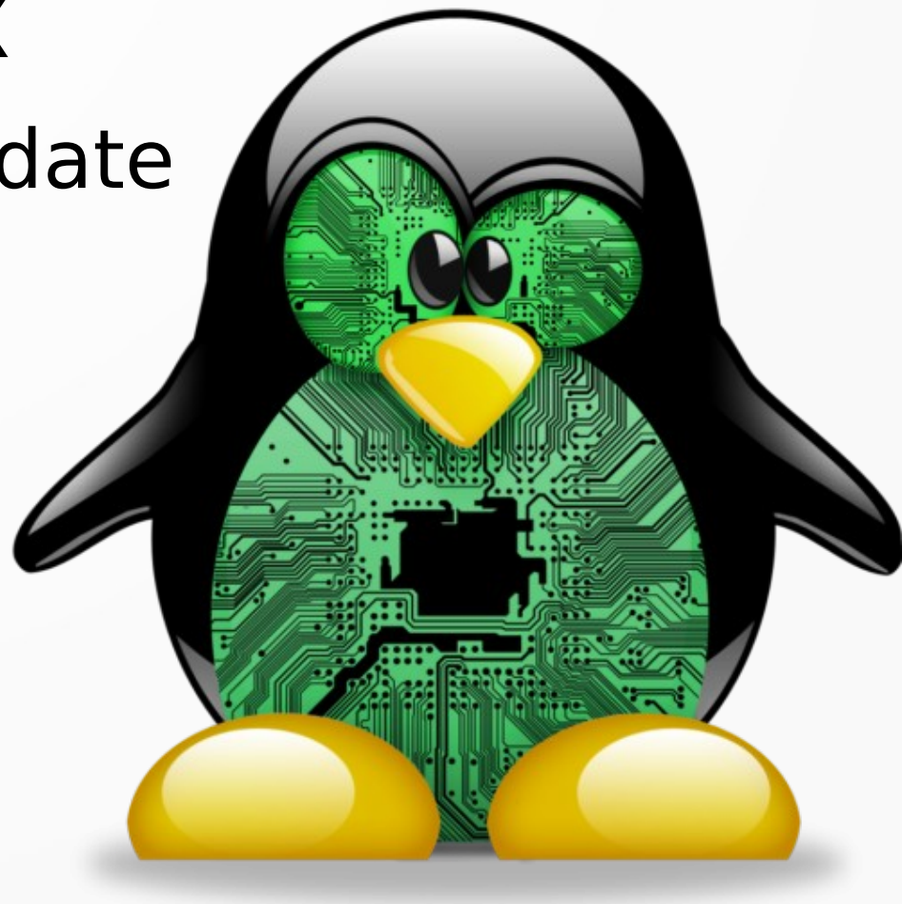


with SWUpdate

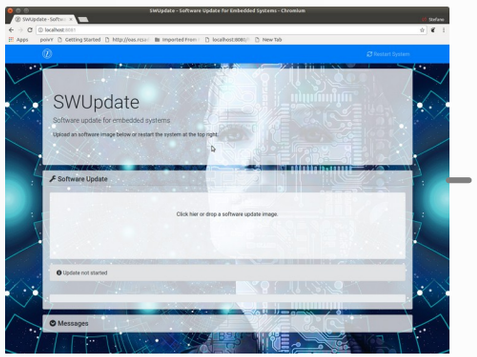
Stefano Babic, DENX Gmbh

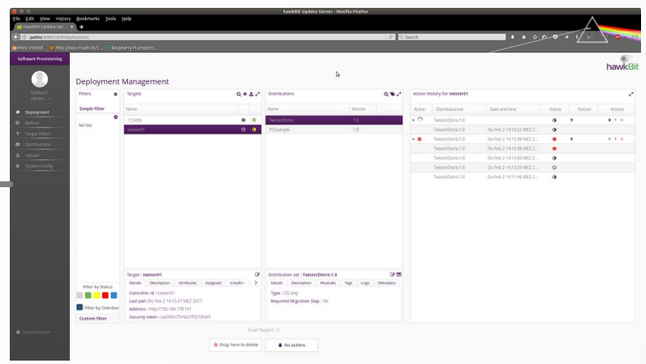
About me

- Freelancer Embedded Developer
- U-Boot Custodian for NXP‘s i.MX
- Author and Maintainer of SWUpdate
- Focus on Linux Embedded

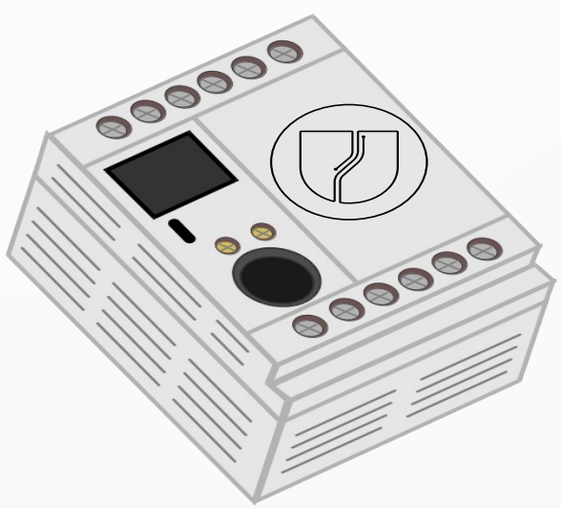


What is SWUpdate ?

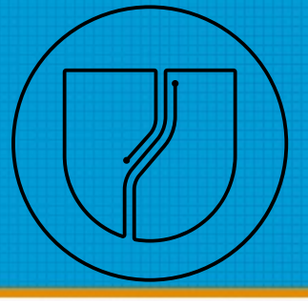




Network



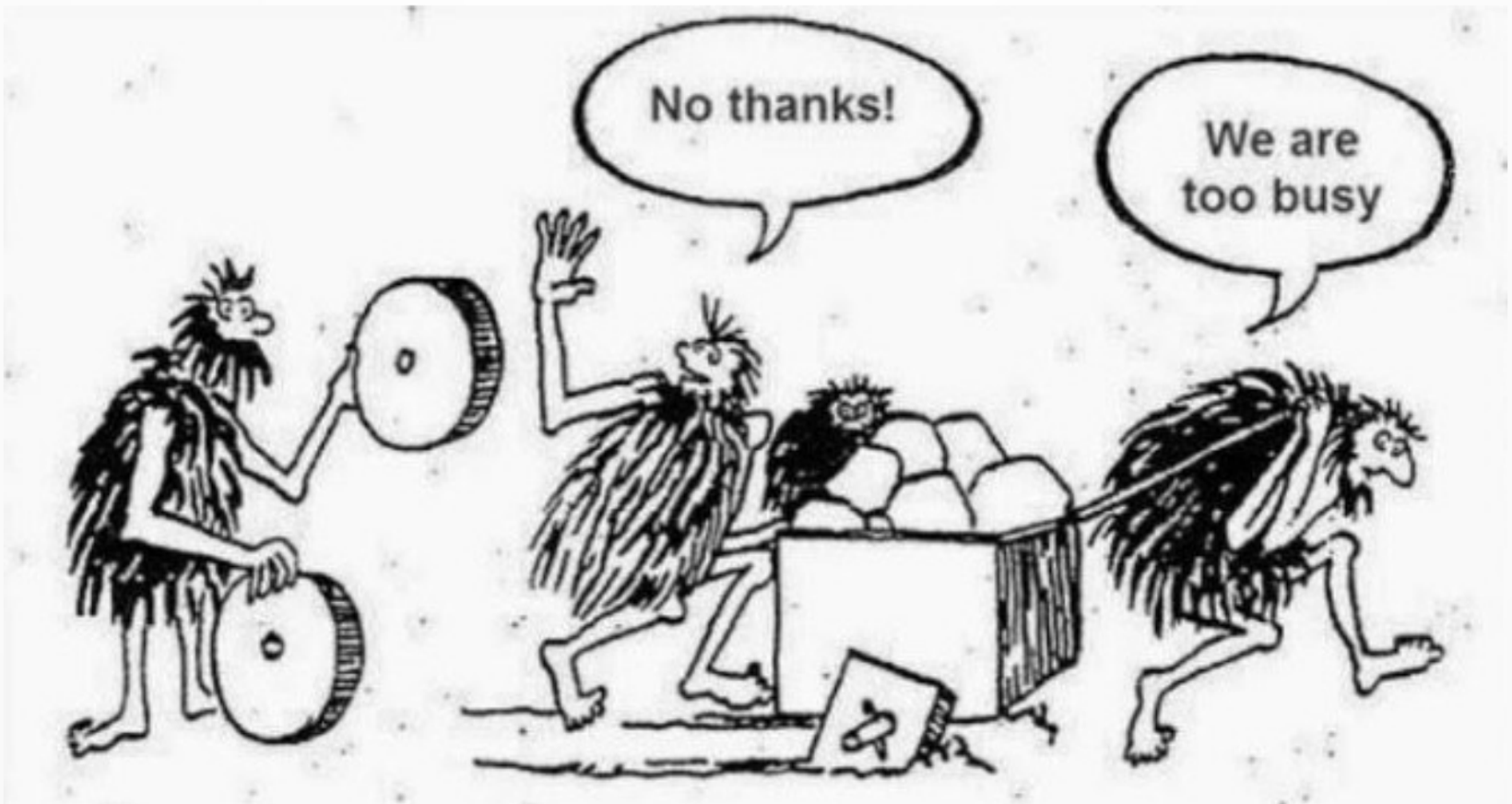
SWUPdate is a framework



- Flexible, but not ready to use
- SWUpdate adapts to your storage layout, not the other way around.
- Some glue logic to start
  - Framework in meta-swupdate, scripts in lib/swupdate/conf.d
- Build and runtime configuration (defconfig + swupdate.cfg)
- Update with one file : SWU (artifacts + sw-description)

Be careful

Best Practices

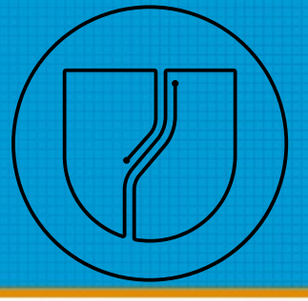


#1. Write a concept

Best Practices

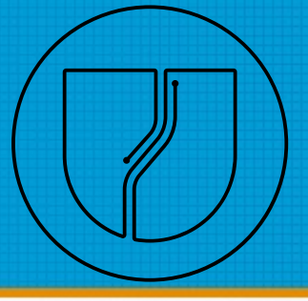

Fleet management
Delta Update
Security
Blabla..
A/B

You have to define:



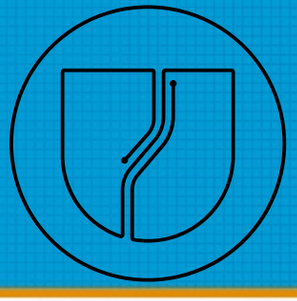
- From which interface you update
  - Local, Webserver, Backend...
- Which HW components you update
    - Storages (eMMC, NAND, ..)
    - Daughterboards
    - FPGAs / µC
- Which components you update
  - Rootfs, application, containers,kernel, bootloader,..

Set security level

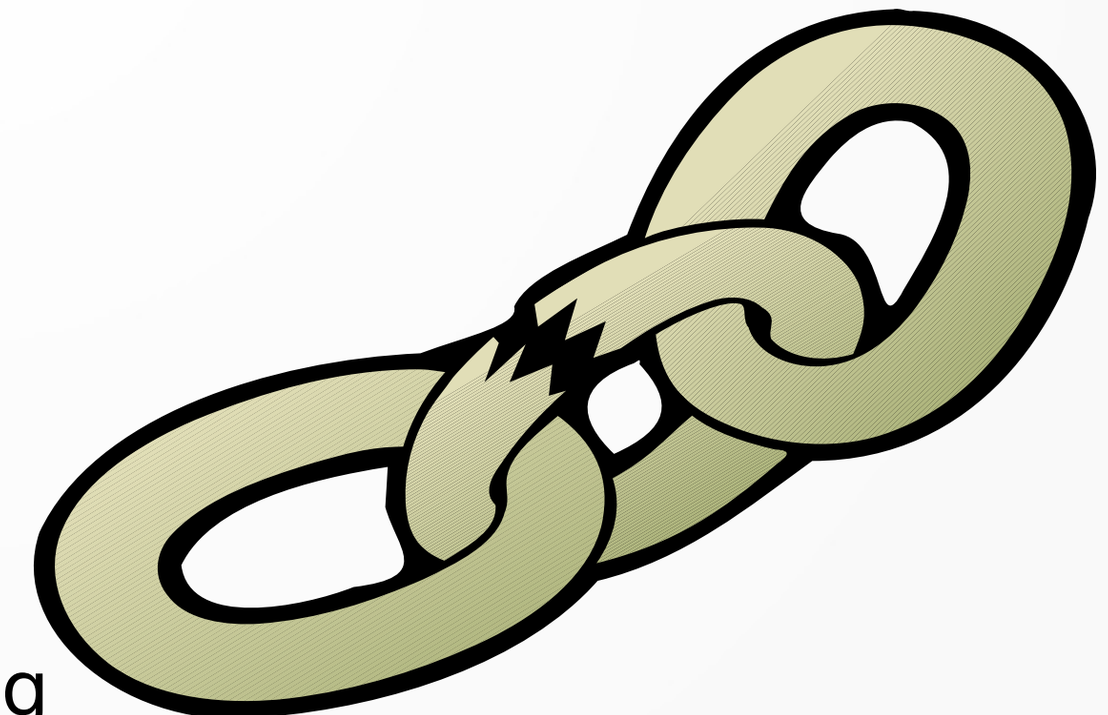


- Signed Image → CONFIG_SIGNED_IMAGES=y
  - RSA PKCS#1.5 → CONFIG_SIGALG_RAWRSA=y
  - RSA PSS → CONFIG_SIGALG_RSAPSS=y
  - Certificate based, PKI (CONFIG_SIGALG_CMS=y)
    - Elapse of certificate can be ignored
    - Signer can be ignored
- Runtime (privilege separation):
  - Set userid / groupid in swupdate.cfg
- IP Property → Encryption on

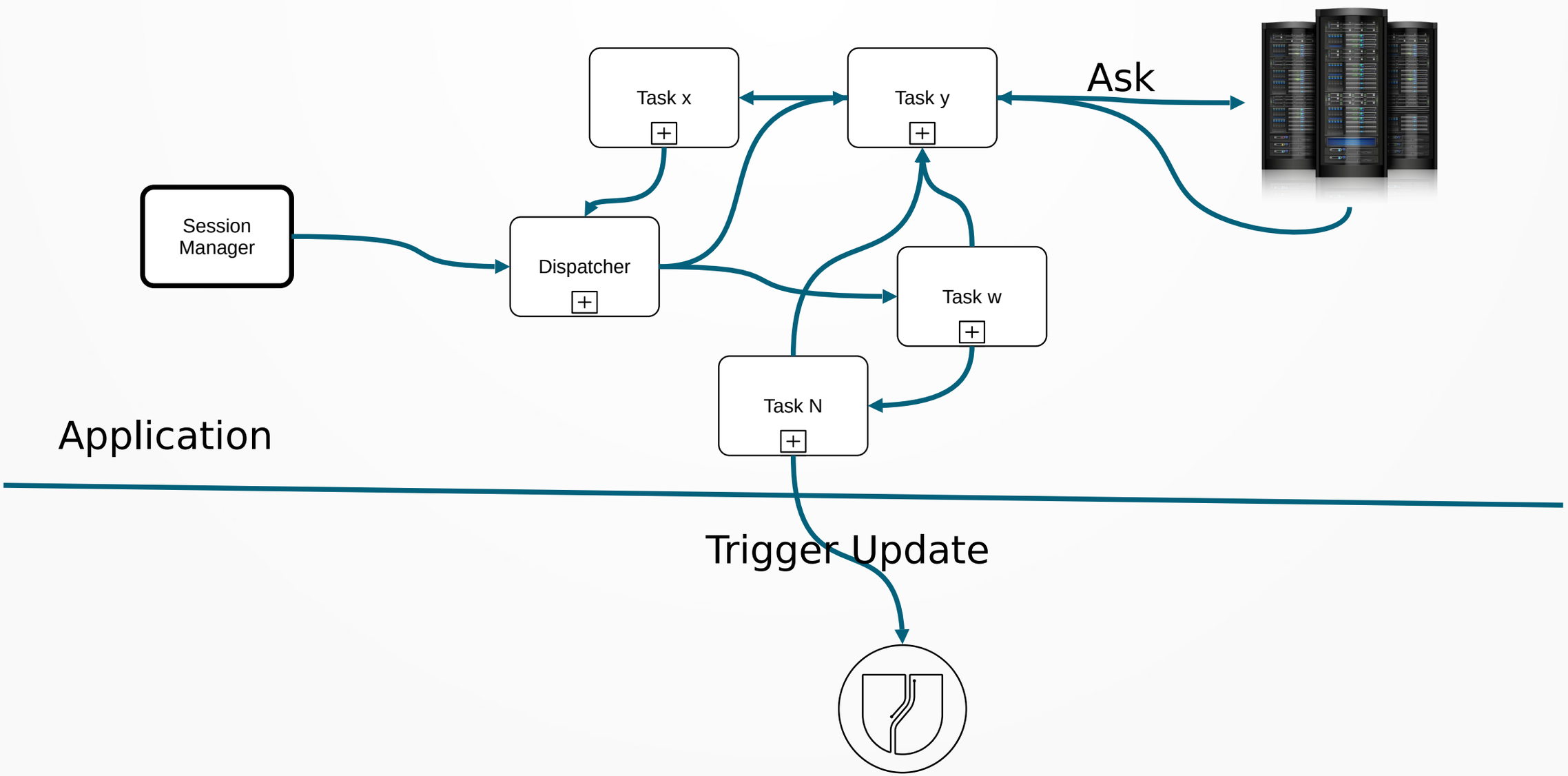
#2. Examine risks



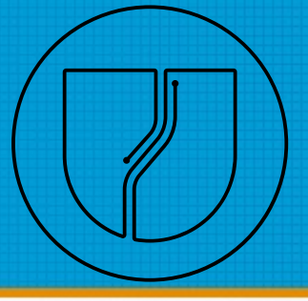
- Find single point of failure
  - Bootloader
  - Not duplicated resources
- Evaluate risks
  - Update vs risk
- No streaming for them !
  - installed-directly = false
- Enable SWUpdate versioning
  - Install-if-different, install-if-greater



#3. Reduce depend



Hard dependency issues

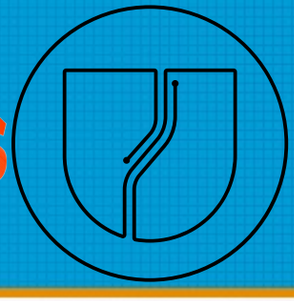


- Application frequently updated
  - Regression bugs
- Complex logic does not work
  - Even if SWUpdate runs, no update possible
- Add a Plan-B
  - Stand alone update via SWUpdate
  - No dependency with application

#4. Verify build config

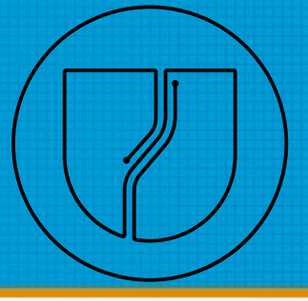
- SWUpdate use Kbuild like kernel
- Just enable what you need
  - NO raw NAND, no UBIVOL handler
- Verification algorythm (RSA, CMS)
- CONFIG_HW_COMPATIBILITY=y
  - You do not know if hw will be changed
  - It allows to check if SW is for a target

#5. Don't abuse shell scripts



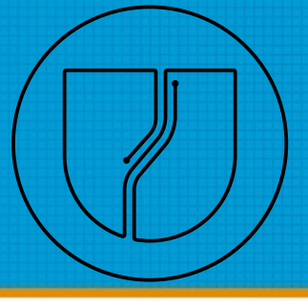
```
software =
{
    version = "0.1.0";
    my-fancy-board = {
        hardware-compatibility: [ "1.0" ];

        scripts: (
            {
                filename = "do_any_mess.sh";
                type = "shellscript";
            }
        );
    };
}
```

Why is it evil ?



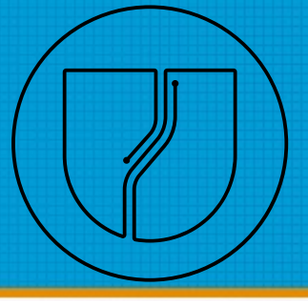
- Calling a shell script means:
  - Fork() is called
  - System() or exec() are called
  - Resources (memory, etc.)
- */bin/*sh often under attacks
- External scripts maybe with malicious code
  - Maybe dependencies with old version ?
- Shellscripts run in the context of installer (root)
- An updater should be self-containing (as much as possible)

Use Lua instead



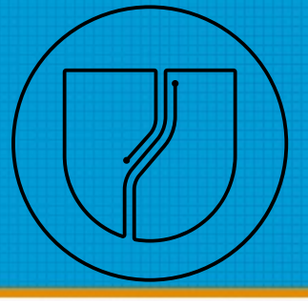
- Interpreter embedded in SWUpdate
- No fork required
- Better control
- SWUpdate provides a lua library
  - require(„swupdate“)
- Again: do not abuse as wrapper to call again the shell !

Typical misunderstanding



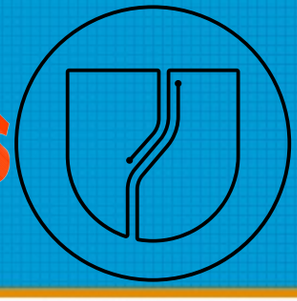
```
software = {
    version = "0.1.0";
    my-fancy-board = {
        hardware-compatibility: [ "1.0" ];
            images: (
            {
                ..............
                installed-directly = true;
            }
        );

        scripts: (
            {
                filename = "preinstall.sh";
                type = "shellscript";
            }
        );
    };
}
```

Do this !



```
software =
{
    version = "0.1.0";
    embedded-script = "
        require(‚swupdate‘)
        function preinstall(image)
            <do preinstall stuff>
        end
";
    my-fancy-board = {
        hardware-compatibility: [ "1.0" ];
            images: (
            {
                ..............
                installed-directly = true;
                hook = „preinstall“;
            }
        );}
```

#6 Activate streaming

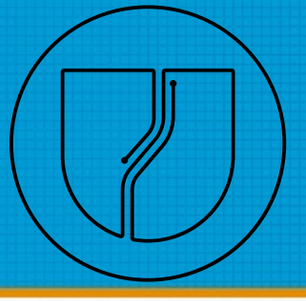


- installed-directly = true;
- Always safe with A/B concept !
- No temporary space needed !
- Working even if filesystem has issues !
- Data remains in SWUpdate’s heap
  - More security
  - No time window to change data

#7 enable checks

- Always enable sha256
  - Mandatory for signed images
- Set always „type“ attribute
- Do not abuse of install order

#8 Check bootloader env

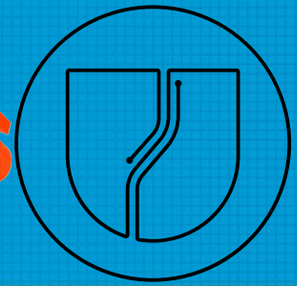


- GRUB : no redundancy available !
- EFIBootguard: redundancy out of the box
- Cboot: redundancy out of the box
- U-Boot:
  - CONFIG_SYS_REDUNDAND_ENVIRONMENT=y
  - Adjust fw_env.config for redundancy
  - Provide u-boot-initial-env
  - SWUpdate uses libubootenv only

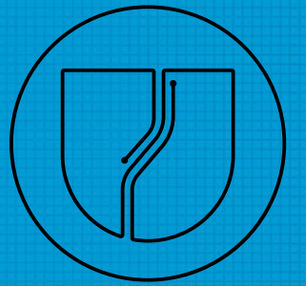
#9 Plan a rescue

- swupdate-image in meta-swupdate
  - Some work as rescue-image inside fitImage will be merged soon.
- Increase reliability
- Avoid to return the device
- Useful in factory for first deployment
- Put in a different storage as production SW

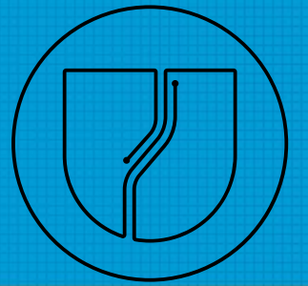
#10 Do not trust vendor



- Vendors are doing **their** job !
  - They provide an example
  - They want to sell SOC / SOM
  - They are not developing the final product
- Security / Reliability is not first goal !
- Integrators shouldn’t copy from vendor !
  - Take it as it is (an example)



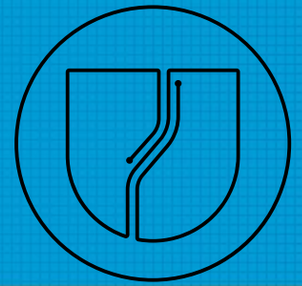
```
software =
{
    version = "0.1.0";
    my-fancy-board = {
        hardware-compatibility: [ "1.0" ];

        files: (
            {
                filename = "myfancyrootfs.tar.gz";
                type = "archive";
                compressed = true;
                device = "/dev/update";
                filesystem = "ext4";
                path = "/";
            }
        );

        scripts: (
            {
                filename = "update.sh";
                type = "shellscript";
            }
        );
    };
}
```



```
#!/bin/sh
if [ $# -lt 1 ]; then
	exit 0;
fi

function get_current_root_device
{
	for i in `cat /proc/cmdline`; do
		if [ ${i:0:5} = "root=" ]; then
			CURRENT_ROOT="${i:5}"
		fi
	done
}

function get_update_part
{
	CURRENT_PART="${CURRENT_ROOT: -1}"
	if [ $CURRENT_PART = "1" ]; then
		UPDATE_PART="2";
	else
		UPDATE_PART="1";
	fi
}

function get_update_device
{
	UPDATE_ROOT=${CURRENT_ROOT%?}${UPDATE_PART}
}
function format_update_device
{
	umount $UPDATE_ROOT
	mkfs.ext4 $UPDATE_ROOT -F -L rootfs${UPDATE_PART}
}
```

Not always valid (DM, encrypted rootfs)

Use external tool instead builtin



```
if [ $1 == "preinst" ]; then
        # get the current root device
        get_current_root_device

        # get the device to be updated
        get_update_part
        get_update_device

        # format the device to be updated
        format_update_device

        # create a symlink for the update process
        ln -sf $UPDATE_ROOT /dev/update
fi

if [ $1 == "postinst" ]; then
        get_current_root_device

        if [ ! -d "/sys/kernel/debug/gpmi-nand" ]; then
                # Adjust u-boot-fw-utils for eMMC on the installed rootfs
                mount -t ext4 /dev/update /tmp/datadst
                rm /tmp/datadst/sbin/fw_printenv-nand
                mv /tmp/datadst/sbin/fw_printenv-mmc /tmp/datadst/sbin/fw_printenv
                sed -i "/mtd/ s/^#*/#/" /tmp/datadst/etc/fw_env.config
                CURRENT_BLK_DEV=${CURRENT_ROOT%p?}
                sed -i "s/#*\/dev\/mmcblk./${CURRENT_BLK_DEV//\/\\/}/" /tmp/datadst/etc/fw_env.config
                umount /dev/update
        fi

        get_update_part

        fw_setenv mmcbootpart $UPDATE_PART
        fw_setenv mmcrootpart $UPDATE_PART
fi
```

T = swupdate.get_root()

Dependency on ext. tools

Change in running SW

NOT ATOMIC !!!!