Yocto Project Summit – Lyon
Day 1 : Thursday 31 October 2019

Lieu Ta, Paul Barker, David Reyna, Mark Hatle, John Mason,

Behan Webster, Mirza Krak, Mark Asselstine, Tim Orling

Presenter Slides:

https://wiki.yoctoproject.org/wiki/YP_Summit_Lyon_2019

Agenda – Yocto Project Summit - Day 1

9:00- 9:20	Welcome and Keynote
9:25- 10:10	Creating Friendly Layers
10:15-11:00	Yocto Project and CVEs
11:05-11:15	Morning Break
11:20-12:05	Transitioning from long term stable to CI/CD
12:10-12:55	Binary Package Feeds for Yocto
1:00-1:45	Lunch
1:50- 2:35	Yocto Project state of the Union panel talk
2:40- 3:25	Creating a Yocto/OE-core BSP layer for the Google Coral Dev Board
3:30- 3:45	Afternoon Break
3:45- 4:30	Building Container Images with the Yocto Project
4:35- 5:20	Resulttool

1. Welcome and Keynote

Lieu Ta

2. Creating Friendly Layers

Paul Barker

About Me

- Involved in Yocto Project since 2013
- Work across the whole embedded stack
- Managing Director & Principal Engineer @ Beta Five Ltd
- Contact: paul@betafive.co.uk
- Website: https://www.betafive.co.uk/

About This Talk

- **Introduction**
- **Best Practices**
  - Layers to learn from
  - Methods
  - Examples
- **Parsing details of bblayers.conf and layer.conf files**
- **Suggestions for future work**

There Shall Be No Victims

- I won’t be showing examples of bad practice today

- Sorry to disappoint!

What Is A Friendly Layer?

- **Simply adding the layer doesn't change functionality**
- **Doesn't assume MACHINE, DISTRO, etc**
- **Careful use of bbappends**
- **Avoid clashing with recipe names in existing layers**
- **Place python helpers in a lib directory**
  - Avoid littering the global namespace
- **Well documented**

Why Should You Care?

- **Yocto Project Compatible badge requires this**
- **Makes it easier to integrate with other layers**
  - Less likely to cause conflicts
- **Easier to test and debug builds**
  - Can quickly turn features on and off
- **Can reduce the number of layers you need to create**
  - Check MACHINE instead of having one layer per machine
  - Check features instead of having one layer per feature
- **Actually simplifies development of your layer**

But can’t you just dynamically set BBLAYERS?

- **Not in a multiconfig**
- **Not based on variables in local.conf or some layer**
  - So you may not even know MACHINE, DISTRO, etc
- **Not even very easily in bblayers.conf**
  - Parsing limitations discussed later
- **Dynamically creating bblayers.conf for each build means another script to maintain**

Layers To Learn From

- **meta-virtualization**
- **meta-clang**
- **meta-security**
- **meta-raspberrypi**

Documenting Your Layer

- **You need a README**
- **Also add a ‘docs’ folder at the top level**
  - Sphinx (http://www.sphinx-doc.org) is a good choice
  - Can publish to Read the Docs (https://readthedocs.org)
- **Also clearly identify**
  - Licensing
  - How to contribute
  - Support forums or email addresses

Keep layer.conf simple

- **Settings in layer.conf apply to all recipes**
  - Not just those in your layer
- **Often difficult to override things set in layer.conf**
- **Parsed very early**
  - Details covered later
  - Parsed in BBLAYERS order not BBFILE_PRIORITY order

Adding New Content in Layers

- **New content is typically safe to add**
  - New recipes
  - New classes
  - New machines
  - New distros
- **Watch out for name clashes**
  - Search the layer index first: https://layers.openembeded.org/

Modifying Existing Recipes

- This is where you can cause problems
- Don’t indiscriminately modify variables and tasks
- Use overrides and conditionals
- Check MACHINE, DISTRO, feature variables, etc

_remove: Use with caution

- _remove takes precedence over _append
- _remove cannot be undone easily!
- Avoid it if at all possible

Using Overrides

- **Extend OVERRIDES based on a variable**
- **Use override syntax in variable assignments**
- **Document your new variable**

- **For example, if you support option `a` and option `b`:**

```
OVERRIDES =. "option-${OPTION}“

SRC_URI_append_option-a = "file://a.patch"
SRC_URI_append_option-b = "file://b.patch file://b.conf"
```

Example: Toolchain Override in meta-clang

- **In clang.bbclass:**

```
# choose between 'gcc' 'clang' an empty '' can be used as well
TOOLCHAIN ??= "gcc"

OVERRIDES =. "${@['', 'toolchain-${TOOLCHAIN}:']['${TOOLCHAIN}' != '']}"

CC_toolchain-clang  = "..."
CXX_toolchain-clang = "..."
CPP_toolchain-clang = "..."
CCLD_toolchain-clang = "..."
CLANG_TIDY_EXE_toolchain-clang = "..."
RANLIB_toolchain-clang = "..."
AR_toolchain-clang = "..."
NM_toolchain-clang = "..."
```

Using Features

- **Three classes of feature variables:**
  - DISTRO_FEATURES
  - MACHINE_FEATURES
  - IMAGE_FEATURES
- **Much tidier than messing with overrides**
- **Conditional syntax isn't very pretty though**

Conditional Syntax

- **Python expressions**
  - Can call a function `fn` with the syntax `${@fn()}`
- **Two commonly used condition functions**
  - oe.utils.conditional

```
def conditional(variable, checkvalue, truevalue, falsevalue, d):
    if d.getVar(variable) == checkvalue:
        return truevalue
    else:
        return falsevalue
```

  - bb.utils.contains – is `checkvalues` a subset of `variable`?

```
def contains(variable, checkvalues, truevalue, falsevalue, d)
```

Conditional Inclusion

- You can use Python expressions in include and require statements
- Example:

```
require ${@bb.utils.contains('DISTRO_FEATURES', ...)}
```

- You can have a simple .inc file without conditionals if you have many changes to make based on one condition

Include vs Require Statements

- **`require` errors on missing files**
  - You almost always want this
- **`include` silently ignores missing files**
  - Useful for optional configs
  - Useful when including something from another optional layer

Example: Distro Features in meta-virtualization

- **README**

```
The bbappend files for some recipes (e.g. linux-yocto) in this layer need to
have 'virtualization' in DISTRO_FEATURES to have effect. To enable them, add
in configuration file the following line.

    DISTRO_FEATURES_append = " virtualization"
```

- **linux-yocto_4.19.bbappend**

```
require ${@bb.utils.contains('DISTRO_FEATURES', 'virtualization',
                             '${BPN}_virtualization.inc', '', d)}
```

- **No DISTO_FEATURES conditionals needed in the .inc file**

Example: Conditional inheritance in meta-security

- **linux-%.bbappend**

```
inherit ${@bb.utils.contains('DISTRO_FEATURES', 'modsign',
                             'kernel-modsign', '', d)}
```

- **No DISTRO_FEATURES conditionals needed in kernel-modsign.bbclass**

Adding Sanity Checks

- **Add a handler for bb.event.SanityCheck**
  - Ensures your check only runs once
- **Raise a flag if things look wrong**
  - bb.warn()
  - bb.error()
  - bb.fatal() if you really can't continue
- **Use this if you really must limit supported values of MACHINE, DISTRO, etc**

Example: Sanity Checks in meta-virtualization

- **sanity-meta-virt.bbclass**

```
addhandler virt_bbappend_distrocheck
virt_bbappend_distrocheck[eventmask] = "bb.event.SanityCheck"

python virt_bbappend_distrocheck() {

    skip_check = e.data.getVar('SKIP_META_VIRT_SANITY_CHECK') == "1"

    if 'virtualization' not in e.data.getVar('DISTRO_FEATURES').split()
        and not skip_check:

        bb.warn("...")

}
```

Using Anonymous Python Functions

- **Useful when more complex conditionals are needed**
  - Full support for python if statements, for statements, etc
- **Executed at parse time**
- **Can use d.getVar() to check variables**
- **Can use d.setVar() to modify variables**
- **Syntax:**

```
python() {
    if d.getVar('SOMEVAR').startswith('prefix'):
        d.setVar('SOMEOTHERVAR', '1')
}
```

Using Classes to Modify Recipes

- Define a new class in your layer
- Do not set INHERIT in layer.conf or elsewhere
- Document that your functionality is enabled by adding the new class to INHERIT in local.conf or a distro conf
- Useful if you have similar modifications to make to many recipes

Modifying BBCLASSEXTEND

- **Appending to BBCLASSEXTEND in a bbappend is relatively safe**
- **No need for conditionals here**
- **May be used to add `-native` variant of an existing recipe**
  - Can then be used in the build of another recipe

yocto-check-layer Script

- **Layer compatibility test script**
- **Checks recipe signatures with and without the layer present**
- **Also checks for other common requirements:**
  - Does the layer have a README?
  - Does everything parse correctly?
  - Is LAYERSERIES_COMPAT set?
  - Can we get signatures for `bitbake world`
    - Actual build is not perfomed

In Summary: Think About Downstream Developers

- **How can they extend configuration?**
- **How can they disable things?**
  - Don’t force them to use _remove
- **Don’t assume distro, machine or target image**
  - If support really is limited, add a sanity check

Parsing Details: bblayers.conf

- **Parsed first**
  - Before any layer.conf
  - Before local.conf or other user config files
  - Before base.bbclass
- **BBLAYERS is iterated as soon as bblayers.conf is fully parsed**
  - Can’t depend on variables from any of the above files
- **No access to python lib directories from any layer**
  - Can’t `import oe` or any submodules
  - Can’t use oe.utils.conditional(), use bb.utils.contains() instead

Parsing Details: layer.conf

- **Parsed in sequence of BBLAYERS immediately after bblayers.conf**
- **Still before local.conf, base.bbclass, etc**
- **Still no access to python lib directories from any layer**
  - Including the current layer!

Future Work

- **Make it easier to write friendly layers**
- **Automate checks against the layer index**
  - Catch recipe, machine or class name duplication
- **Nerf layer.conf**
- **Simpler conditionals?**
- **Encourage more layer documentation**
  - Should we standardise here?

Thank You

Any questions?

Follow Up: paul@betafive.co.uk

3. Yocto Project and CVEs

David Reyna

Overview: Security and Yocto Project

- **What this presentation is about**
  - Resources
  - General background about CVEs
  - Process around CVE patching
  - Tools for finding and managing CVEs
  - Work in progress
- **What this presentation is not about**
  - Fixing CVEs
  - Runtime Security checks (e.g. openSCAP)
  - Hacking yoctoproject.org

Overview: Yocto Project Security Management

- **Since the Yocto Project is intended to be flexible and meet the needs of many applications, we leave policy-making decisions around security to our end users.**
  - Our goal instead is to ship each release with metadata that follows best practices in that we do not release recipe versions which are known to have significant security vulnerabilities.
  - Generally this is done by upgrading recipes to newer versions that are no longer vulnerable to these issues.
  - We also solicited and receive direct patches from our community
- **The Yocto Project community is doing a lot of work around CVEs, but that work is not always visible to our members, in terms of tooling, communication, and management**
- **We are looking at ways to better engage the community in tracking, communicating, and fixing CVEs**

Resources

Resources:

- **The Yocto Project Security homepage can be found here:**
  - https://wiki.yoctoproject.org/wiki/Security
- **Public security mailing list (there is also a private one)**
  - yocto [dash] security [at] yoctoproject[dot] org
- **People**
  - Ross Burton (general security)
  - Mark Hatle (general security)
  - Pierre Le Magourou (cve-check)
  - David Reyna (Security Response Tool)
- **Papers from Yocto Project Members**
  - https://events.linuxfoundation.org/wp-content/uploads/2017/12/Keeping-Up-With-The-Joneses-CVEs-David-Reyna-Wind-River-Systems.pdf
  - https://ossna19.sched.com/event/PTaX/open-source-cve-monitoring-and-management-cutting-through-the-vulnerability-storm-akshay-bhat-timesys

Security and Yocto Project: Paper by NCC

- **The was a paper published in 2018 by NCC Group that covered a lot of security topics related to Yocto Project**
  - http://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2018/improving-embedded-linux-security-yocto3.pdf
- **We have recently reviewed that paper, and it seems quite reasonable.**
  - It's correct in that meta-security-isafw is abandoned: that was part of the Intel RefKit effort and that was disbanded some time ago.
  - The paper needs updating in a few places but seems a good overview of the entire field.
- **The cve-check paragraph in this paper is still applicable today as the maintainers did not modify the Yocto user behaviour, except for 2 things:**
  - CVE_CHECK_CVE_WHITELIST is deprecated and has been simplified to **CVE_CHECK_WHITELIST**, in which you only set the CVE numbers that need to be whitelisted.
  - CVSSv3 score has been added in the CVE report.

Some security related links/useful tools:

- **CVE details:**
  - https://www.cvedetails.com/
- **CVE list, Linux kernel 2019**
  - https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2019/Linux-Linux-Kernel.html
- **Meta-security-layer**
  - http://layers.openembedded.org/layerindex/branch/master/layer/meta-security/
- **Making images more secure**
  - https://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#making-images-more-secure
- **Cvechecker**
  - https://github.com/sjvermeu/cvechecker/wiki

General background on CVEs

Background: CVEs

- **CVE (Common Vulnerability Enumerations)**
  - The enumerations of the community tracked security vulnerabilities, separated by the year reported (e.g. CVE-2018-12345)
- **Vendors/Sources**
  - MITRE: Manages the list of CVEs
  - NIST (National Institute of Standards and Technology): manages the National Vulnerability Database (NVD) of CVEs
- **Hardware Vendors, Software Maintainers, Distros**
  - Many vendors track and share CVE's relevant to their product
  - Many CVE aggregators also available (e.g. cvedetails.com)
- **Mailing lists, websites, and forums (public and private)**
  - Preview of coming issues, place to discuss issues

Background: CVEs

- **Volume of CVEs is 1000+ per month and growing**

Quality of CVEs: Issues

- CVEs may only have a brief or incomplete description
- The affected Common Product Enumeration (CPEs) lists in CVE may have gaps, errors, unexpected version deviations, and may even be empty
- CVE content may be misleading, mentioning one package when it actually affects a different package
- CVEs may have few, inaccurate, or missing content links (discussion, reproducers, patches)
- CVE status changes continually as new information is discovered and shared
- Sometimes delays in content updates (dark CVEs)

Quality of CVEs: Issues (2)

- The most recently created CVEs (within the last few months) are particularly prone to the above issues, but unfortunately these are often all that organizations have to work with for their pending releases (i.e. there is often no CPE data to work with)

- Tools (e.g. CVE scanners) generally rely completely on these CPE lists, which is why the above issues are important. **Ideally**, they insure that (a) they are flexible in processing the information, (b) that they can differentiate between strong and weak data, (c) that expectations are set as to what the tool is able conclude and act upon, and that (d) humans are appropriately included in the process.

Quality of CVEs: Examples 1

- CVE-2017-13220:
  - The CPE says “cpe:2.3:o:google:android:-:*:*:*:*:*:*:*” then talks about upstream kernel issues and refers to a kernel SHA.
- CVE-2014-2524:
  - Has a CPE which claims all releases of “readline” 6.3 and below are vulnerable, but the problem only exists in 6.0 onwards.

Quality of CVEs: Examples 2

- CVE-2017-8872:
  - Against “libxml” resulted in a bug and patch, but upstream ignored it. An almost-identical patch was merged recently but no mention of the CVE was made
- CVE-2018-10195:
  - A case study in 'dark CVEs'. Reserved in MITRE, Red Hat have their own notice and a patch. Since it is for software which is long-dead, this patch will never go upstream.

Quality of CVEs: Some Good News

- **Pierre has actually worked directly with NVD in fixed some CVEs. It is a small team, but he has found that they have been very responsive and timely for his requested fixes**

- **Here is the mail address of the nvd team:**
  nvd@nist.gov

- **They would be happy to update their database (as time and resources allow) if you find a problem in it. They ask you to provide publicly available information though, to be able to verify your claims.**

CVE Tools 1: CVE System Analysis

- **Can be very valuable in targeting product specific review activities**
- **Tells you of known vulnerabilities, but not what you are NOT vulnerable to**
- **Scans almost exclusively in the category of 'needs investigation‘**
- **Depends on known data (CPEs)**
- **Can be very expensive**
- ***Example: Nessus***

CVE Tools 2: CVE Build/Source Analysis

- Can be more precise than system analysis
- Possible for something to trigger a vulnerable warning for components never used
- You still need to determine what you are not vulnerable to, understand the items that were reported, etc.
- **Depends on known data (CPEs)**
- *Examples: Black Duck, Yocto Project 'cve-report', Dependency Tracker*

CVE Tools 3: Catch CVEs Early and Often

- Actively scan the incoming and updated CVE records, and compare against your product(s) source
- Proactively prevent vulnerability injection, use expertise to interpret CVE content, merge with other vulnerability resources (e.g. private lists)
- **Depends on engineering time and expertise**
- *Examples: Security Response Tool (SRTool)*

Process

The Security Homepage process statement

- **Since the Yocto Project is intended to be flexible and meet the needs of many applications, we leave policy-making decisions around security to our end users.**
- **Our goal instead is to ship each release with metadata that follows best practices in that we do not release recipe versions which are known to have significant security vulnerabilities.**
- **Generally this is done by upgrading recipes to newer versions that are no longer vulnerable to these issues.**

The Security Homepage process statement

- **Upgrading recipes to the newer versions in the maintenance branches is not always easy, this is why we provide a patch for the existing version instead if we detect a vulnerability in a package. The patches are added to the recipes, see example below:**

poky/recipes-connectivity/bind/bind_9.9.5.bb

```
SRC_URI = "ftp://ftp.isc.org/isc/bind9/${PV}/${BPN}-${PV}.tar.gz \
           file://conf.patch \
           ...
           file://bind9_9_5-CVE-2014-8500.patch \
```

Yocto Security Team

- **The purpose of creating a security team in the Yocto project is to discuss, sync and organize security related activities.**

- **The team's main responsibilities among others are:**
  - Scanning of security forums/mailing list(s) to detect security vulnerabilities reported by community
  - Responsible for fixing CVEs in the Yocto releases & maintained branches
  - Evaluation of tools for security tests
  - Responsible for security related info in the Yocto documentations
  - Hardening of Yocto release

Branches maintained with security fixes

- **See the “Stable branches maintenance” link for detailed info regarding the policies and maintenance of Stable branch.**
  - https://wiki.yoctoproject.org/wiki/Stable_branch_maintenance
- **Policy: all CVE (security) patches should be back-ported if at all possible. If a CVE fix is only appropriate to a stable branch the patch submission should detail why this is the case.**
- **The older versions in grey are no longer actively maintained with security patches, but well-tested patches may still be accepted for them**

Policy for updating package versions for the stable branches

- The Yocto project purposely limits updating of packages on oe-stable releases to items that address security problems (e.g. CVEs).
- For packages like QEMU we avoid updating between from one "dot.dot" to another related "dot.dot" version since it has been seen in the past that even with "simple" updates, things can go wrong and a lot more testing is required to verify compatibility.
- Better to only add CVE patches to fix specific point problems.

A practical discussion between Mark and Ross

- **Patch contribute is not yet a fully formal process. Contributors send a patch as per usual. If it fixes a CVE we hope that they backport it to the stable branches too.**
- **Specifically patches have to go to master first, and then be backported. If it's not applicable to master, then they can go directly to the affected layer.**
- **Many CVE fixes for stable branches come from OSVs who are sharing the fixes they've integrated (WR, Mentor, MV, etc) but it's a fact that these OSVs are not contributing all of the fixes they have for various reasons.**

A practical discussion between Mark and Ross - 2

- **This lack should be changing soon: the Yocto Project Technical Steering Committee is looking at "the security question" and plans to get something formalized in the future, probably involving SRTool and a team of people across companies.**
- **The Yocto Project will do 'minor' upgrades for security fixes, but only if they are sure that the API/ABI is consistent before and after the -minor- upgrade. There have been numerous instances, such as boost, where this is not true. In those cases, individual fixes are applied.**
- **I think this 'hope' is the biggest issue that a 'more' formal process needs to resolve. If you don't know it's an applicable CVE process then they don't know it needs to be backported. That's where things need to start. Knowing if a particular version is likely affected by a problem, and being able to backport the fix if a newer version is not affected. (Knowing WHY it's not affected will significantly help with this effort.)**

In Practice: If you want to know if your project is vulnerable

- Check Bugzilla if there is a defect with the CVE number
- Check the commit logs if there is a patch with the CVE number
- Run tools like cve-check to check your build manifest
- Soon: use SRTool to have the above data already correlated
- Watch for the regular CVE patch emails from the Security Team (automated dispatcher)

In Practice: If you find a security vulnerability

- **If you find a security flaw; a crash, an information leakage, or anything that can have a security impact if exploited in any Open Source packages used by the Yocto Project, please report this to the Yocto Security Team.**
- **If you prefer to contact the upstream project directly, please send a copy to the security team at Yocto as well.**
- **If you believe this is sensitive information, please report the vulnerability in a secure way, i.e. encrypt the email and send it to the private list.**

Meta-Security Layer

Meta-Security

- **The meta-security layer "provides security tools, hardening tools for Linux kernels"**
  - http://git.yoctoproject.org/cgit/cgit.cgi/meta-security/tree/README
- **Notes from the layer maintainer (Armin)**
  - All packages in this layer get their build and runtime tests executed on a regular basis
  - Popular packages include apparmor, smack, clamav, openSCAP, and even the older bastille
- **See the NCC paper for more reviews and explanation of the meta-security layer content**

Security Build Flags

Security build flags

- **There is a class that defines the that your build can use to inject static and runtime checks:**
  - https://git.openembedded.org/openembedded-core/tree/meta/conf/distro/include/security_flags.inc

```
…
# Inject pie flags into compiler flags if not configured with gcc itself
# especially useful with external toolchains
SECURITY_PIE_CFLAGS ?= "${@" if '${GCCPIE}' else '-pie -fPIE'}"
SECURITY_NOPIE_CFLAGS ?= "-no-pie -fno-PIE"
SECURITY_STACK_PROTECTOR ?= "-fstack-protector-strong"
SECURITY_CFLAGS ?= "${SECURITY_STACK_PROTECTOR} ${SECURITY_PIE_CFLAGS} …
SECURITY_NO_PIE_CFLAGS ?= "${SECURITY_STACK_PROTECTOR} ${lcl_maybe_fortify} …
SECURITY_LDFLAGS ?= "${SECURITY_STACK_PROTECTOR} -Wl,-z,relro,-z,now"
SECURITY_X_LDFLAGS ?= "${SECURITY_STACK_PROTECTOR} -Wl,-z,relro"

# powerpc does not get on with pie for reasons not looked into as yet
GCCPIE_powerpc = ""
GLIBCPIE_powerpc = "“
…
```

Tools

CVE-CHECK

- **A recipe called cve-update-db populates a sqlite database from NVD json feeds. The cve-check class reads the database for each recipe to check for CVEs. To use it, you just have to add this to your "local.conf":**

```
inherit += "cve-check"
```

- **In the Yocto source code for CVE patches, it generates a text report that says if the CVE is patched or unpatched in the image we built.**
- **You can whitelist some CVEs in a recipe with the CVE_CHECK_WHITELIST variable**
- **Pierre reports that cve-check is working well and that he uses it every day.**

CVE-CHECK

- **Here is some example results:**

```
WARNING: wpa-supplicant-2.6-r0 do_cve_check: Found unpatched CVE (CVE-2017-13077 ↩
CVE-2017-13078 CVE-2017-13079 CVE-2017-13080 CVE-2017-13081 CVE-2017-13082 CVE-2017-13084 ↩
CVE-2017-13086 CVE-2017-13087 CVE-2017-13088), for more  information check ↩
${BUILDDIR}/tmp/work/arm1176jzfshf-vfp-poky-linux-gnueabi/wpa-supplicant/2.6-r0/cve/cve.log
```

- **Snippet from CVE log:**

```
PACKAGE NAME: wpa-supplicant
PACKAGE VERSION: 2.6
CVE: CVE-2017-13077
CVE STATUS: Unpatched
CVE SUMMARY: Wi-Fi Protected Access (WPA and WPA2) allows reinstallation of the Pairwise ↩
Transient Key (PTK) Temporal Key (TK) during the four-way handshake, allowing an attacker ↩
within radio range to replay, decrypt, or spoof frames.
CVSS v2 BASE SCORE: 5.4
VECTOR: ADJACENT_NETWORK
MORE INFORMATION: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-13077
```

NOTE: CVSS V3 data added recently

(examples from NCC presentation)

CVE-CHECK

- **Suppressing a false-positive CVE (list):**

```
# CVE-2017-13084 does not affect wpa-supplicant 2.6 because the affected PeerKey
# implementation is not fully functional, and therefore poses no practical risk.
#
# See the "PeerKey / TDLS PeerKey" section of:
# https://w1.fi/security/2017-1/wpa-packet-number-reuse-with-replayed-messages.txt
#
CVE_CHECK_WHITELIST = "{\
    'CVE-2017-13084', \
}"
```

- **By the way, there is a self test recipe for cve-check in the “meta-security” layer**

(examples from NCC presentation)

Cve-check

- **cve-check-tool replaced by cve-update-db (JSON feeds)**
- • Master (and now Zeus) branch only!
  - https://git.yoctoproject.org/cgit/cgit.cgi/poky/log/meta/recipes-core/meta/cve-update-db-native.bb
  - https://git.yoctoproject.org/cgit/cgit.cgi/poky/log/meta/classes/cve-check.bbclass
- **CVE result improvements**
  - cve-check-tool (string compare) vs. cve-update-db (>=, <= etc.)

Recipe	Rev	Previously missed
wpa-supplicant	2.6	3
python	3.5	5
sumo	2.30	5

(data from Timesys presentation)

Yocto Project CPE to Recipe Mapping

- **CVE_PRODUCT: recipe name to NVD name mapping**
  - curl_7.65.3.bb: CVE_PRODUCT = "curl libcurl“
  - openssl_1.1.1c.bb: CVE_PRODUCT = "openssl:openssl“
  - python-urllib3.inc: CVE_PRODUCT = "urllib3"
- **CVE_VERSION: recipe version to NVD version mapping**
  - krb5_1.17.bb: CVE_VERSION = "5-${PV}"
- **Tracks patched CVEs**
  - CVE ID in patch header (preferred)
  - CVE ID in file name

(examples from Timesys presentation)

Security Response Tool (SRTool)

- ***While there is heighten awareness about device vulnerabilities, what is often missing is awareness about the process of managing the security response process itself***
- **Wind River is sharing to open source a tool to help manager the organization's security response management:**
  - Better ways to handle 1000+ CVEs per month
  - Better ways to connect CVE's to defects to product
  - Better ways to allow easy access to the full vulnerability status, generate reports, clean exports to public CVE DB
  - Better ways to use automation to keep all the data sources automatically up to date
- **Community Page:**
  - https://wiki.yoctoproject.org/wiki/Contribute_to_SRTool
- **ELCE Presentation:**
  - https://sched.co/HOLr

Work in progress

Yocto Project CPE to Recipe Mapping

- Always working to better map CVEs and CPEs to Yocto Project Recipes
- Make it easier for people find the project's Security information
- Publish documentation on these tools, especially the cve-check
- Leverage the SRTool database and its automated update features to help drive the CVE input for tools like the cve-check
- Extend the SRTool to merge data from tools like cve-check into its database, and also add GUI tools and reports around those tools
- Extend the SRTool to automatically scan the YP/OE repositories for patches that are tags as CVE fixes, and allow correlation with the CVE and cve-check data

4. Transitioning from long term stable to CI/CD

Mark Hatle

Terms

- **Long Term Stable – a release that has some defined period of maintenance, and a defined bug fix strategy**
- **Continuous Integration – Act of integrating upstream source code and local changes on a continuous basis.**
- **Continuous Development – Developing against the latest Continuous Integration OS**
- **Continuous Delivery – Delivery production code to customers on a continuous basis**
- **DevOps – Encompasses Continuous Integration, Continuous Development, Continuous Delivery and necessary organizational changes to support this model.**
- **Technical Debt – The work that you need to maintain, as it is not in the community**

Stable Release Strategy

Traditional Approach – Long Term Stable

a.k.a. Periodic Uplift

Long Term Stable
Traditional OSS device development model

- **Why?**
  - This is what people have been doing for years
  - People are used to managing the risks, challenges, and maintenance
  - Well supported by community and commercial interests
- **Starts by choosing a "stable" version of the Open Source software with a plan to remain there for a period of time (part or all of expected product life cycle)**
- **Your development then is on top of the stable, and expects minimal changes to the stable base over time.**

Long Term Stable

Long Term Stable

Yocto Project Dev
(6 Months)

Yocto Project Stable
(12 Months)

By the time development starts, you are 1-7+ months out of date with for software features, but APIs are established.

Product Developments

Product Maintenance

By the time you deploy, you are on your own for support… (Can be mitigated w/ OSV support)

Long Term Stable

- **Advantages:**
  - Lowers perceived risks*****
  - You know exactly what you will get now and into the future
  - Minimal to no API changes over life of product
- **Disadvantages:**
  - *****May actually increase long term support risks (Bugs, CVEs, etc)
  - Use commercial OSVs to mitigate this risk
  - Can’t rely on Open Source communities to help with maintenance
  - Software may be obsolete by the time you use it
  - Functional capabilities are locked down
  - No or minimal new features

Continuous Integration & Development

Advanced Approach – CI with Long Term Stable

Continuous Integration/Development

- **Regular integration/rebase of open source components**
- **Why?**
  - Better understanding of new features
  - Ability to influence community direction (and features)
  - Time to market
  - Ability to transition to “stable” model
- **Starts by using the in-development version of the Yocto Project, then follows the stable branch when available.**
- **Your development is based on in development work, and expects minimal changes to the stable base over time.**

Long Term Stable

Yocto Project Dev
(6 Months)

Yocto Project Stable
(12 Months)

Product Developments

Product Maintenance

Long Term Stable

Yocto Project Dev
(6 Months)

Yocto Project Stable
(12 Months)

**Staying current with development means you are up-to-date,
but you need to keep rebasing to stay current… churn can cause rework**

Product Developments

Product Maintenance

By the time you deploy, you know the quality level of the components and you can piggy back on the Yocto Project stable support path longer. Can benefit from OSV support.

Moving from LTS to CI
Define quality objectives (OS)

- **How do you measure what the quality level is?**
  - Automation is key!
- **Testing (OS)**
  - Frameworks(s)
  - Community Tests
  - Your Own Tests
- **What is acceptable quality?**
  - Don’t have to test everything, but you need to test your use cases.

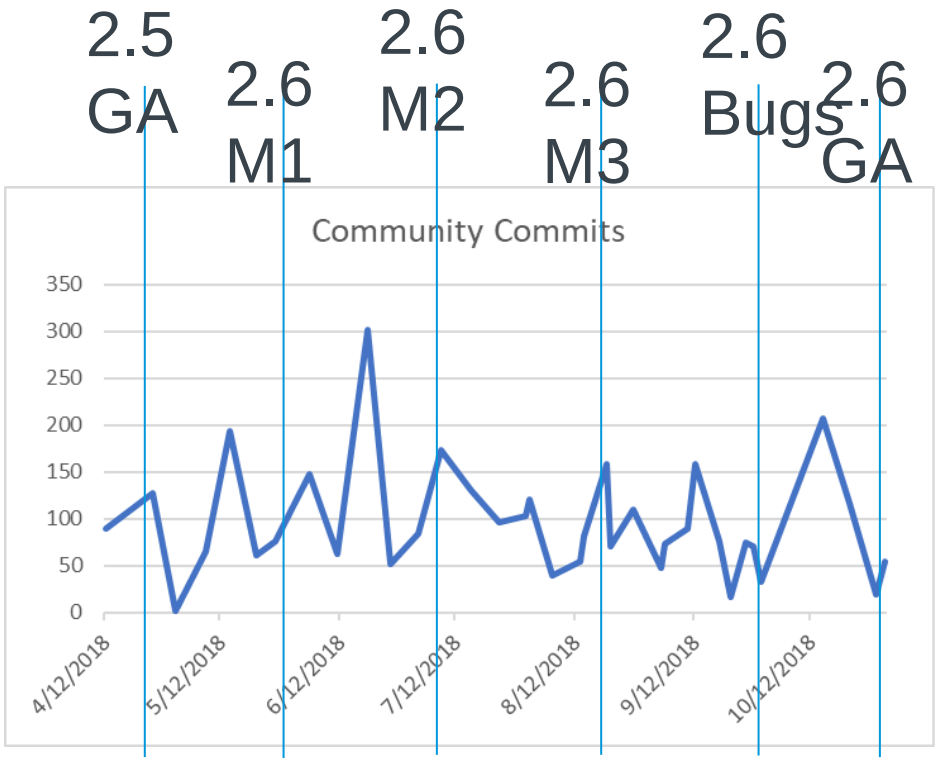
Moving from LTS to CI
Define synchronization strategy (OS)

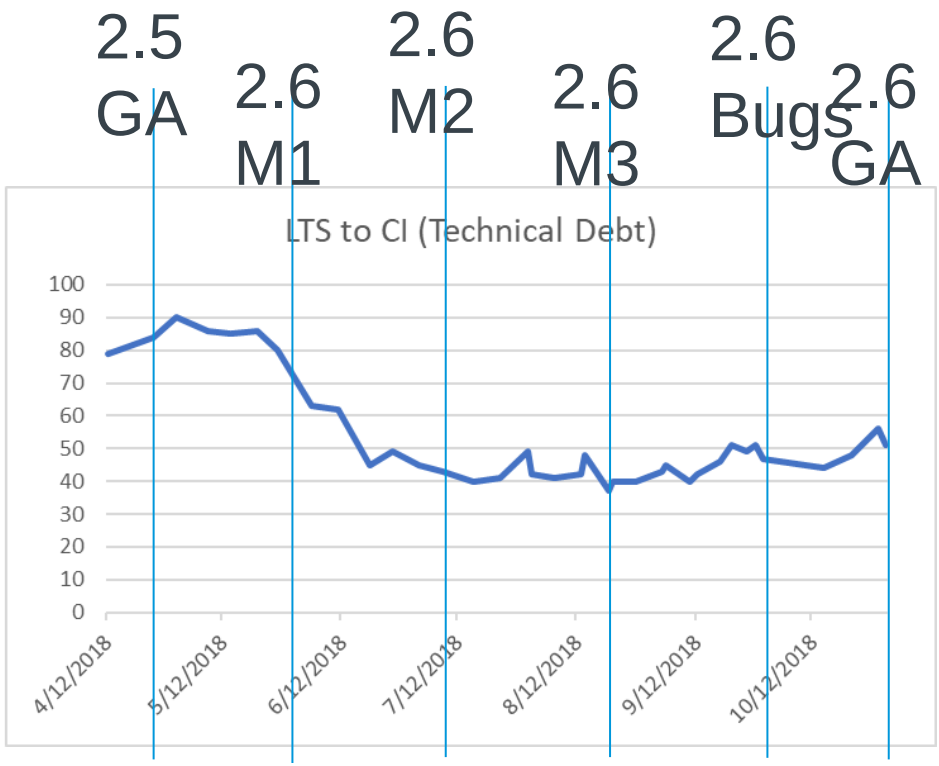
- **Merge or rebase?**
  - Merge hides technical debt
  - Rebase brings technical debt ‘to the top’, at the expense of non-FF
- **How often to resync?**
  - Daily – every 2 weeks
- **What happens when there is a conflict?**
  - Who fixes the problem?
- **Identify “changes”, and communication to users is key**
- **If/when to push upstream (lower technical debt)**

Moving from LTS to CI
Integration Strategy (Product)

- **Decide when to integrate into product development**
  - Quality Criteria?
  - Time?
- **How to integrate development work**
  - Rebase? Merge? “next” development? Etc…
- **How to deal with periods of transition?**
  - What happens when API/ABI changes?
- **Testing**
  - How to catch when something unexpected changed, but didn’t trigger build-time error
  - May need more diligent functional testing, then otherwise necessary

Rebase Work





Based on http://github.com/WindRiver-OpenSourceLabs – bitbake, oe-core, meta-yocto, meta-openembedded

Continuous Integration & Development

- **Advantages:**
  - Ability to influence quality and features of OSS components
  - Faster time to market/More up-to-date features
  - Longer Open Source support window
  - ***Lowers perceived maintenance risks, once on stable
  - Stable API/components after release
- **Disadvantages:**
  - Need to coordinate development and release schedule with YP release
  - Requires additional testing for risk management
  - *Once on stable, same long term support risks (Bugs, CVEs, etc)
  - Use commercial OSVs to mitigate this risk
  - Can’t rely on Open Source communities to help with maintenance, long term
  - Functional capabilities are locked down
  - No or minimal new features

Continuous Integration & Continuous Delivery

DevOps Approach

Continuous Integration & Continuous Delivery

- **Regular integration/rebase of open source components**
- **Why?**
  - Better understanding of new features
  - Ability to influence community direction (and features)
  - Time to market
  - Continuous ability to incorporate new features
  - Easier to resolve defects
  - Keeps technical debt under control
- **Using the in-development version of the Yocto Project**
- **Your development is based on in development work, expect to adjust over time to new features**

Continuous Integration & Continuous Delivery

Yocto Project Dev
(6 Months)

**Staying current with development means you are up-to-date,
but you need to keep rebasing to stay current… churn can cause rework**

Product Developments

With a CI/CD approach to released product, life span of a product can be longer, as new features are easier to introduce.

Moving from CI to DevOps
Define quality objectives (OS & Product)

- **How do you measure what the quality level is?**
  - Automation is key!
- **Testing (OS)**
  - Frameworks(s)
  - Community Tests
  - Your Own Tests
- **What is acceptable quality?**
  - Don’t have to test everything, but you need to test your use cases.

Same as CI approach!

Moving from CI to DevOps
Define synchronization strategy (OS & Product)

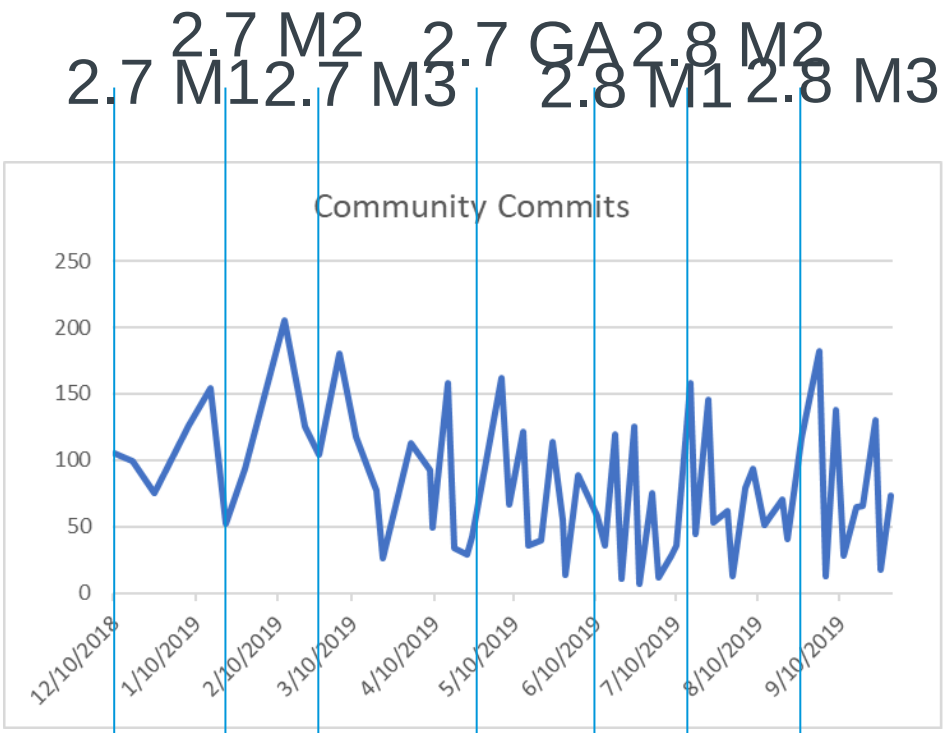
- **Merge or rebase?**
  - Merge hides technical debt
  - Rebase brings technical debt ‘to the top’, at the expense of non-FF
- **How often to resync?**
  - Daily – every 2 weeks
- **What happens when there is a conflict?**
  - Who fixes the problem?
- **Identify “changes”, and communication to users is key**
- **If/when to push upstream (lower technical debt)**

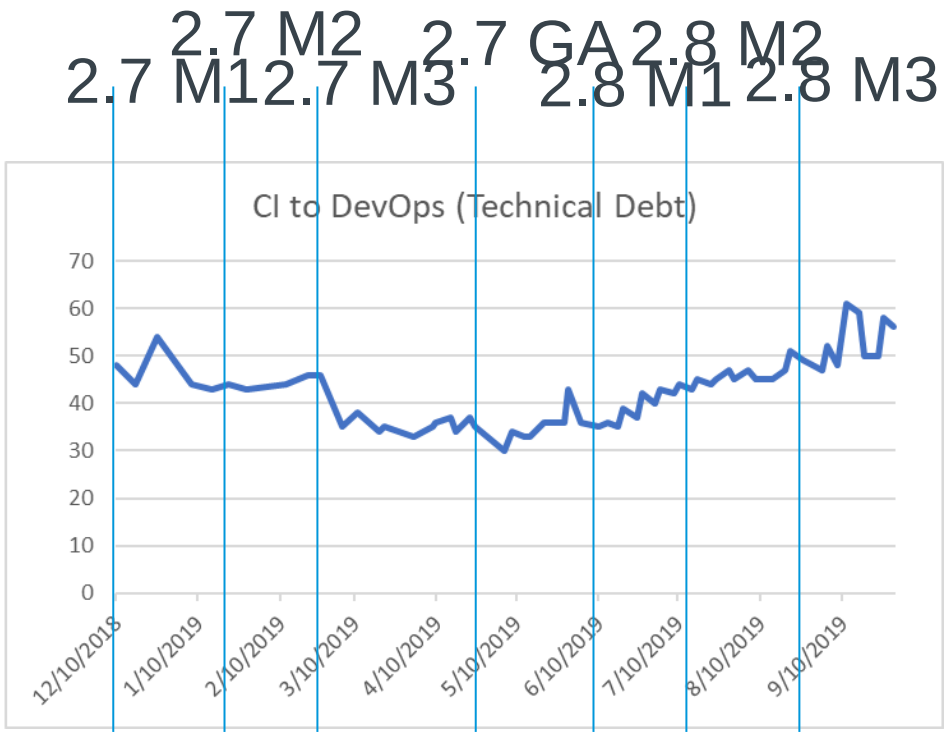
Same as CI approach!

Moving from CI to DevOps Integration Strategy (Product)

- **Decide when to integrate**
  - Keep OS and Product in lock step!
- **Define Release Criteria**
  - What do we do if we can't meet the criteria?
  - "Skip" a release – focus on the next version
- **How long is a release supported for?**
  - Short support windows are key
  - The longer an individual release is maintained, the higher the work required
  - Think weeks, not years!
  - Can't overlook the need for some maintenance activities
  - *Always* backport

Moving from CI to DevOps





Based on http://github.com/WindRiver-OpenSourceLabs – bitbake, oe-core, meta-yocto, meta-openembedded

Continuous Integration & Continuous Delivery

- **Advantages:**
  - Ability to influence quality and features of OSS components
  - Faster time to market/More up-to-date features
  - Always active maintenance window
  - New features and upgrades available
- **Disadvantages:**
  - YP master Quality varies at different stages of development
  - Need to be willing to ‘wait’, or contribute to improving the OSS quality
  - Requires additional testing for risk management
  - Need to prepare for when a required feature is made obsolete

Recap

LTS to DevOps
A journey, doesn't happen overnight!

	LTS	CI + LTS	DevOps
Ability to Influence	None	Some	A Lot
Time to Market	Slow	Faster	Fastest
Rate of Change	Small	Variable (Dev) Small (Maint)	Variable
Feature Changes	None	Yes (Dev) None (Maint)	Yes
Maint Require	Grows over time	Grows over time	Predictable
Requires Automation	No	Yes	Yes

Questions?

5. Binary Package Feeds for Yocto

John Mason

Download the presentation

https://wiki.yoctoproject.org/wiki/File:BinaryPackageFeed.pptx

6. Yocto Project state of the Union panel talk

Moderator: Behan Webster

7. Creating a Yocto/OE-core BSP layer for the Google Coral Dev Board

Mirza Krak

About me

- **Mirza Krak**
  - 8 years in embedded Linux
  - Board Support Package
  - Yocto Project
  - Open source
- **mender.io**
  - OTA updates for embedded Linux devices
  - Apache 2.0
  - End-to-end solution
  - ~40 device integrations using Yocto Project

Session overview

- **Share the journey of creating a BSP layer for the Coral Dev Board**
  - approach can probably be applied to other boards

Coral Dev Board - Hardware

SOC	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Integrated GC7000 Lite Graphics
ML accelerator	Google Edge TPU coprocessor
RAM	1 GB LPDDR4
Flash memory	8 GB eMMC
Wireless	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2

https://coral.withgoogle.com/products/dev-board

Coral Dev Board - Software

- **Mendel Linux (release Chef)**
  - Mendel Linux is a lightweight derivative of Debian Linux
  - Debian apt repositories
  - Additions for Coral Dev Board peripherals
- **Pre-built images**
- **Convenient for prototyping**

Coral Dev Board - Software

- **Contacted support**
- **Custom build system to generate images**
  - debootstrap wrapper
- **https://coral.googlesource.com**
  - source code of all “extra” components and build system

Coral Dev Board - Software

- **Forked BSP components**
  - uboot-imx
  - linux-imx
  - imx-firmware
  - wayland-imx
  - ….
- **Only a few had actually been changed**
  - uboot-imx and linux-imx

Coral Dev Board - Software

- **Conclusion: Based on NXP BSP**
  - could probably reuse much of what is in meta-freescale
- **Obvious that they are using Yocto as reference**
  - *“Import IMX8MM bl31/tee from Yocto”*

meta-coral

- **Depend on meta-freescale**
  - to get NXP BSP components
- **Machine**
  - conf/machine/coral-dev.conf
  - based on meta-freescale/conf/machine/imx8mqevk.conf
  - updated dtb names, U-Boot defconfig etc..

meta-coral

- **Started with U-Boot recipe**
  - u-boot-coral_2017.03.bb
  - based on u-boot-imx_2017.03.bb from meta-freescale (thud)
  - had to make a small patch to imx-mkimage (hardcoded dtb name)
  - core-boot-script.bb (boot.txt from Mendel Linux)

meta-coral

- **Linux kernel recipe**
  - linux-coral_4.9.51.bb
  - based on linux-imx_4.9.123.bb from meta-freescale
  - imported defconfig from linux-imx-debian (Mendel Linux)

meta-coral

- **Custom WKS file to create disk image**
  - meta-coral/wic/coral-bootpart.wks.in
  - image suitable to write to SD card (or eMMC)
  - meta-freescale/wic/imx-imx-boot-bootpart.wks.in

```
part u-boot --source rawcopy --sourceparams="file=imx-boot" --ondisk mmcblk --no-table --align
${IMX_BOOT_SEEK}
part /boot --source bootimg-partition --ondisk mmcblk --fstype=ext4 --label boot --active --align 4096 --size 16
part / --source rootfs --ondisk mmcblk --fstype=ext4 --label root --align 4096

bootloader --ptable msdos
```

meta-coral - bootable

```
meta-coral/
├── conf
│   ├── layer.conf
│   └── machine
│       └── coral-dev.conf
├── recipes-bsp
│   ├── coral-boot-script
│   │   ├── cora-boot-script.bb
│   │   └── files
│   │       └── boot.txt
│   ├── imx-mkimage
│   │   ├── files
│   │   │   └── 0001-add-BOARD-argument.patch
│   │   └── imx-boot_0.2.bbappend
│   └── u-boot
│       ├── u-boot-coral
│       │   ├── 0001-tools-allow-to-override-python.patch
│       │   └── 0002-ext4-cache-extent-blocks-during-file-reads.patch
│       └── u-boot-coral_2017.03.bb
├── recipes-kernel
│   └── linux
│       ├── linux-coral
│       │   └── defconfig
│       └── linux-coral_4.9.51.bb
└── wic
    └── coral-bootpart.wks.in
```

meta-coral - Edge TPU

- **ASIC designed by Google**
  - high performance ML inferencing for TensorFlow Lite models
  - PCIe and I2C/GPIO to interface with the iMX8M SoC

meta-coral - Edge TPU

- **Binary blobs (libedgetpu)**
  - x86_64, armhf, arm64 (aarch64)
  - depends on clang (meta-clang)
  - libedgetpu_1.0.bb
- **Python API (edgetpu)**
  - python3-edgetpu.bb
- **Python Vision API (edgetpuvision)**
  - python3-edgetpuvision.bb

meta-coral - Works

- Boot
  - from SD card
  - eMMC should work
- Ethernet
- HDMI
- WiFi
- Edge TPU
- Cooling fan

meta-coral - Future work

- **USB Gadget**
  - not tested yet
- **Bluetooth**
  - not tested yet
  - https://coral.googlesource.com/bluez-imx/
- **Edge TPU still needs some work and testing**
  - recipes for examples/demos
  - edgetpu-demo.bb
    - issues with gstreamer1.0-plugins-base-imx
    - requires gobject-introspection but disabled (error if enabled)

meta-coral - Future work

- **New release of Mendel Linux (day)**
  - Not released
  - Based on MM_04.04.05_1902_L4.14.x
  - would like to include this in zeus branch
- **QA**
  - automated builds
- **Audio configuration**
  - port configuration from Mendel Linux

Demo

8. Building Container Images with the Yocto Project

Mark Asselstine

1. Why Build Containers?

To Push to a Container Registry

- Secure/Insecure public or private Docker registries
- Secure public or private registries like Harbor
- Local container registry

Usecase – To make software available in the form of a container to a deployed system

- Docker pull
- Kubernetes pod (kind: Deployment…containers: name)

To Include in a Rootfs

- **Alternative to rpm or ipk**
  - To organize software (dependencies) and configuration, example Apache
  - To allow for simplified uprev, example replace ‘factory’ container with container pulled from a container registry
  - To isolate Application and Platform SW
- **To encapsulate 3rd party SW and required libraries**
- **To compartmentalize builds**
- **As a transition to microservices**

2. Dockerfile - Simple

Hello World! Application

Code

```
#include <stdio.h>
int main(int argc, char **argv)
{
        printf("\nHello World!\n");
        return 0;
}
```

Compile

```
gcc -o hello -static -Os -fno-asynchronous-unwind-tables hello.c
```

Hello World! Dockerfile

Code

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

Explanation

1. Don’t use a layer 1 (essentially a nop)
2. Copy the ‘hello’ executable to the root of the container
3. Default parameters to ENTRYPOINT

Hello World! Build and Prepare

Directory Contents

```
%> ls -l
total 20
-rw-r--r-- 1 root root   42 Oct 18 18:10 Dockerfile
-rwxr-xr-x 1 root root 9576 Oct 18 18:16 hello
-rw-r--r-- 1 root root  178 Oct 18 18:12 hello.c
```

Build the Container Image

```
docker image build -t hello-world:yp .
```

Hello World! Inspect and Run

Inspect

```
%> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         yp                  f3f730d7ee41        31 minutes ago      863kB
```

Run

```
docker run --rm hello-world:yp

Hello World!
```

The ‘git’ Container

Code

```
FROM alpine
LABEL maintainer Bill Wang ozbillwang@gmail.com
RUN apk --update add git less openssh && \
  rm -rf /var/lib/apt/lists/* && \
  rm /var/cache/apk/*
VOLUME /git
WORKDIR /git
ENTRYPOINT ["git"]
CMD ["--help"]
```

Reference: **https://hub.docker.com/r/alpine/git/dockerfile/**

Build the ‘git’ Container

```
%> time docker image build -t git .
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM alpine
…
Step 7/7 : CMD ["--help"]
 ---> Running in e1750d6b05eb
Removing intermediate container e1750d6b05eb
 ---> 128c824f6e91
Successfully built 128c824f6e91
Successfully tagged git:latest
real    0m17.356s
user    0m0.083s
sys     0m0.192s

%> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
git                 latest              128c824f6e91        3 minutes ago       29.2MB
alpine              latest              961769676411        8 weeks ago         5.58MB
```

Hello World! Multi Stage

Code

- ```
  FROM alpine:latest AS builder
  RUN apk --update add gcc libc-dev
  COPY hello.c .
  RUN gcc -o hello -static -Os -fno-asynchronous-unwind-tables hello.c

  FROM scratch
  COPY --from=builder hello /
  CMD ["/hello"]
  ```

Inspect

- %> docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	a891eee89a48	3 hours ago	93.2kB
  ```

What have we learned about Dockerfiles

- Pros
  - Can produce small images
  - Fast builds
  - Dockerfile are easy to read, generally map to cmdline ops
- Cons
  - Can produce large images
  - No easy way to cross compile
  - No easy way to know about licensing

3. About Container Images

Open Container Initiative(OCI) Container Format

The OCI specification defines a format for encoding a container as a **Filesystem Bundle**

All the information required to load and run a container

Must Have:

- config.json – must be at the root, must be called config.json, contains configuration data
- Root filesystem – must be at the root, referenced by root.path in config.json

Reference: https://github.com/opencontainers/runtime-spec/blob/master/bundle.md

Docker Image Format

Depends on image format version but at its minimum **Must** have:

- **VERSION file**
- **Image JSON**
- **layer.tar**
  - A root filesystem image
    or
  - Filesystem changeset

Reference: https://github.com/moby/moby/blob/master/image/spec/v1.2.md

4. Using Yocto to Create a Container RootFS

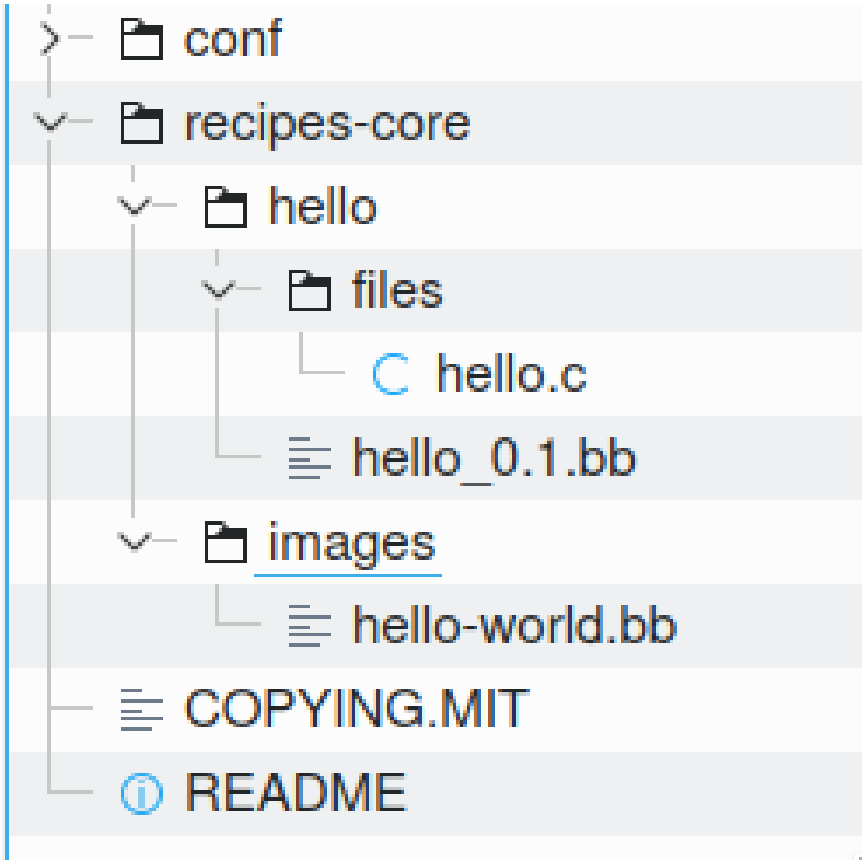
image-container.bbclass

- Found in poky/meta/classes
- Enforces use of linux-dummy for PREFERRED_PROVIDER_virtual/kernel
- Adds the ‘container’ IMAGE_FSTYPES
- Disables the installation of ROOTFS_BOOTSTRAP_INSTALL (ie. run-postinsts)
- Inherited automatically with the inclusion of ‘container’ in IMAGE_FSTYPES

Configure a New Build and ‘local’ Layer

- . ~/git/poky/oe-init-build-env container-build
- cd container-build
- echo "IMAGE_FSTYPES='container'" >> conf/local.conf
- Echo " PREFERRED_PROVIDER_virtual/kernel = 'linux-dummy'" >> \
  conf/local.conf
- bitbake-layers create-layer local
- bitbake-layers add-layer local

Populate ‘local’ layer



hello_0.1.bb

Code

```
SUMMARY = "A Hello World example"
DESCRIPTION = "A simple Hello World example."
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://hello.c"
S = "${WORKDIR}"

do_compile() {
    ${CC} -o ${B}/hello -static ${LDFLAGS} hello.c
}
do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${B}/hello ${D}${bindir}
}
```

Hello-world.bb

Code

- SUMMARY = "A Hello World container image rootfs"
  DESCRIPTION = "A Hello World container image rootfs."
  LICENSE = "MIT"
  LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

  IMAGE_INSTALL = "hello"
  IMAGE_FEATURES = " "

  inherit image

Build, Import, Inspect and Run

- **bitbake hello-world**
- **docker import **
  tmp/deploy/images/qemux86-64/hello-world-qemux86-64.tar.bz2 hello-world
- **docker run --rm hello-world /usr/bin/hello**

  Hello World!
- **%> docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	75506629d2fc	37 minutes ago	3.73MB

Summary

Pros

- Yes it can be done
- Full Yocto Project benefits (license.manifest)
- Cross compile friendly

Cons

- Slow (mostly due to –native, 1200+ tasks,23 minutes)
- Large
- Requires Docker install or other metadata handler

5. Size

Shrinking Size

- **glibc contributes to most of the size**
  -rwxr-xr-x 1 mark mark 671K Oct 21 21:03 hello
  -rw-r--r-- 1 mark mark 2.9M Oct 25 14:33 locale-archive

- **Use musl instead**
  %> TCLIBC=musl bitbake hello-world

- **%> docker images**

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	62200ca82e06	6 seconds ago	17.7kB

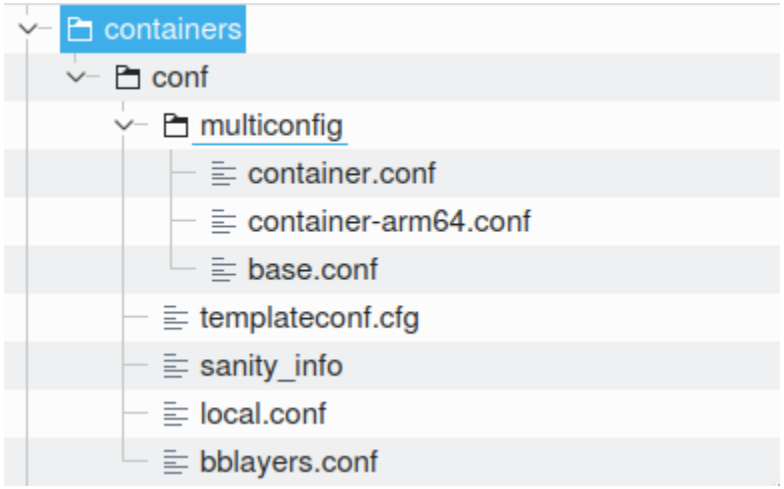
6. Cross Compile

Multiconfig

- Use ‘multiconfig’ to build several ARCH or MACHINE in the same TOPDIR
- Shares many, but not all, -native packages
- Provides a way to build and include container images in a rootfs in one ‘build’

Multiconfig Setup

- **Add BBMULTICONFIG to local.conf**
  ex.: BBMULTICONFIG = "base container container-arm64“
- **Setup ‘multiconfig’ directory**



Multiconfig Setup

base.conf

- TMPDIR = "${TOPDIR}/base“

container.conf

- TCLIBC = "musl“
  TMPDIR = "${TOPDIR}/container“
  IMAGE_FSTYPES='container’
  PREFERRED_PROVIDER_virtual/kernel = "linux-dummy“

Container-arm64.conf

- TCLIBC = "musl“
  MACHINE = "qemuarm64“
  TMPDIR = "${TOPDIR}/container-arm64“
  IMAGE_FSTYPES='container’
  PREFERRED_PROVIDER_virtual/kernel = "linux-dummy"

Multiconfig Build

- **Build**

  **%> bitbake mc:container:hello-world **
  mc:container-arm64:hello-world

- **Result**

  %> ls container/deploy/images/qemux86-64/hello-world-qemux86-64.tar.bz2
  container/deploy/images/qemux86-64/hello-world-qemux86-64.tar.bz2

  %> ls container-arm64/deploy/images/qemuarm64/hello-world-qemuarm64.tar.bz2
  container-arm64/deploy/images/qemuarm64/hello-world-qemuarm64.tar.bz2

7. Saving Time

Builder Container

Idea: create a cloud friendly container and reuse –native build artifacts in the process

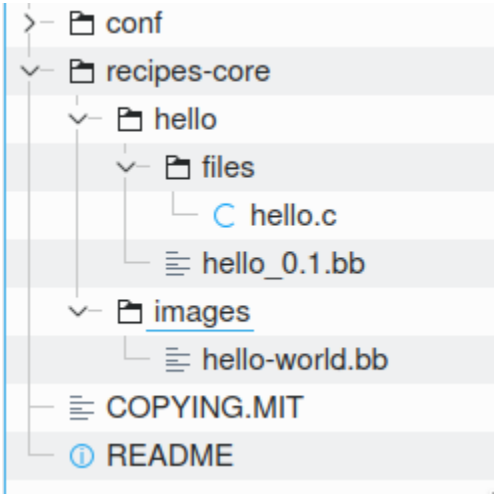
1. **Complete a simple build – hello-world**
2. **Put the whole build into a container**
3. **Use the container to build more containers**

2. Put the Whole Build into a Container

```
FROM debian:stretch-slim
COPY . /mark/bld/container
RUN apt update && apt install -y locales gawk wget git-core cpio \
    diffstat unzip texinfo gcc-multilib python2.7 python2.7-minimal \
    python-minimal build-essential chrpath socat libsdl1.2-dev xterm && \
    rm -rf /mark/bld/container/cbuilder/local && \
    rm -rf /mark/bld/container/cbuilder/tmp/deploy/images/qemux86-64 && \
    sed -i -e 's/# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/' /etc/locale.gen && \
    locale-gen
ENV HOME=/mark
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8
WORKDIR /mark/bld/container/cbuilder
VOLUME /mark/bld/container/cbuilder/local
   /mark/bld/container/cbuilder/tmp/deploy/images/qemux86-64
   /mark/bld/container/downloads
ENTRYPOINT ["/mark/bld/container/entrypoint.sh"]
```

3. Use the Container to Build More Containers

- **Create a layer - similar to local layer created for Hello World!**



- **Create an ‘out’ directory**

- **sudo docker run --rm -it **
  **-v $PWD/local:/mark/bld/container/cbuilder/local **
  **-v $PWD/out:/mark/bld/container/cbuilder/tmp/deploy/images/qemux86-64 **
  yp-cbuilder bitbake <image-name>'

8. DevOps Example

Use Container Build in Test and Deploy

- **Use Docker bindings with python scripts to run the container and push it to a registry**

Using Docker Python Bindings

```
import os
import pexpect
import docker
from subprocess import call

PROMPT = "[0-9]+#"
SF_PROMPT = "bash-4.4#"
CLIENT_PROMPT = "root@.*:/#"
IMAGE_FILE = "tmp/deploy/images/genericx86-64/simple-firewall-genericx86-64.tar.bz2"
FNULL = open(os.devnull, 'w')

# pexpect connections
sf_conn = None
client_conn = None

# Test results
TOTAL_TEST = 3
num_skipped = TOTAL_TEST
num_passed = 0
num_failed = 0

# Connect to docker client
docker_client = docker.DockerClient(base_url='unix://var/run/docker.sock')
```

Using Docker Python Bindings

- **Import an image tarball**
  ret = call(["docker", "import", IMAGE_FILE, "simple-firewall"], stdout=FNULL)
- **Push a container to a registry**
  image = docker_client.images.get("simple-firewall")
  image.tag("localhost:5000/simple-firewall")
  docker_client.images.push("localhost:5000/simple-firewall")
- **Details see https://github.com/masselstine/simple-firewall**

9. Future Work / Ideas

Container Registry Fetcher

Idea: To include existing containers in a rootfs image

- **Add the ability to write recipes which reference existing containers on a registry**
- **Register included containers with a container runtime to have them start automatically**

Write container metadata

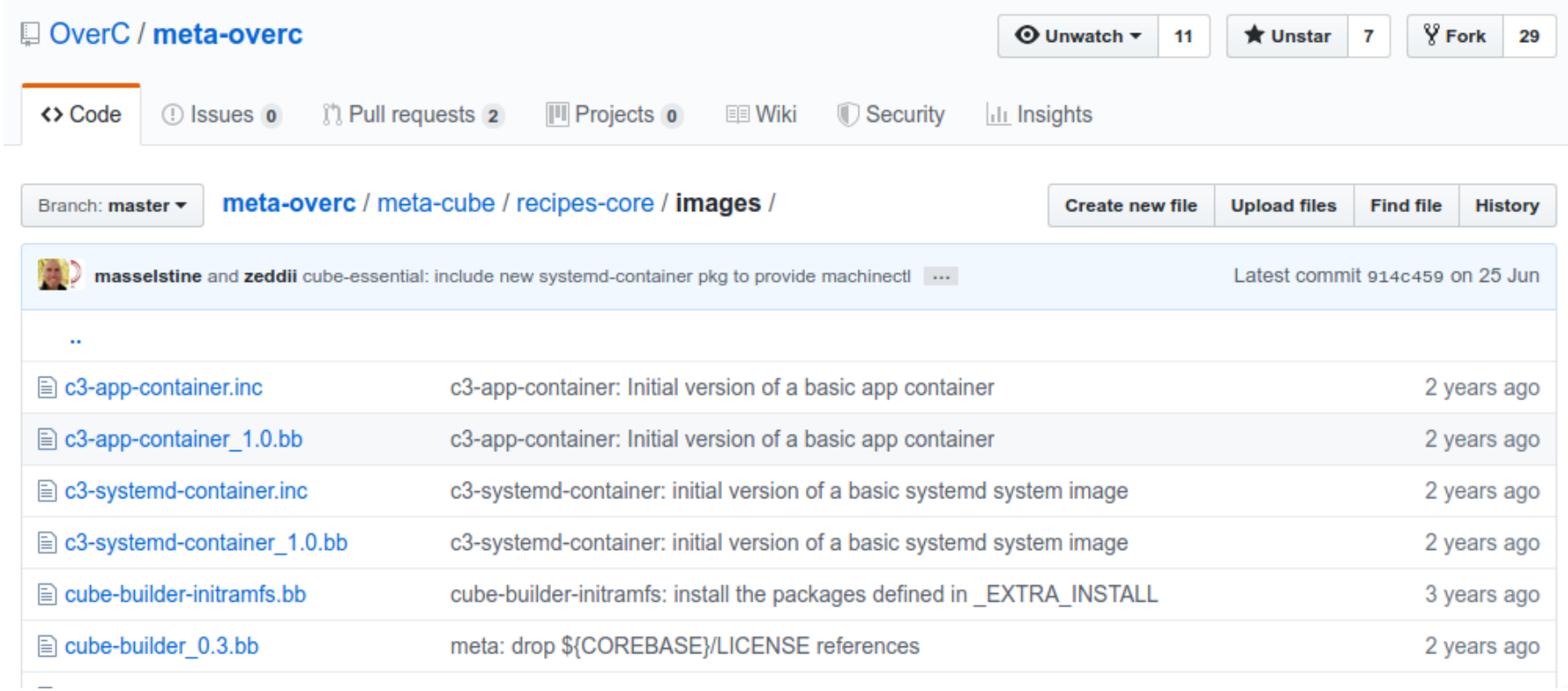
Idea: go beyond creating just the container images

- **Write OCI compatible config.json**
- **Write Docker VERSION and Image JSON**
- **Provide the ability to push to a container registry from the build**

meta-overc

- OverC is a containerized OS framework
- Extensible via the addition of containers
- Supports runC container runtime natively
- Supports OCI container format natively
- Container image agnostic, works with OCI and Docker image formats and repositories
- Provided as a Yocto Project layer
- Assembled by a custom installer (ie. No wic)

Use cube-builder to Seed a Build Container



Questions

9. Resulttool or: How I Learned to Stop Worrying and Love testresults

Tim Orling

Download the slides from here:

- **https://wiki.yoctoproject.org/wiki/File:Yocto_Project_Summit_2019_resulttool.pptx**

Bonus Slides

Hash Equivalency/Runqueue

Joshua Watt

Outline

1. What is the Runqueue?
2. Traditional Runqueue Execution
3. What is the purpose of Hash Equivalence?
4. Runqueue Execution with Hash Equivalence Server
5. Signature Generation with Hash Equivalence Server
6. Live Demo
7. The Role of Reproducible Builds
8. Alternate Output Hash Methods

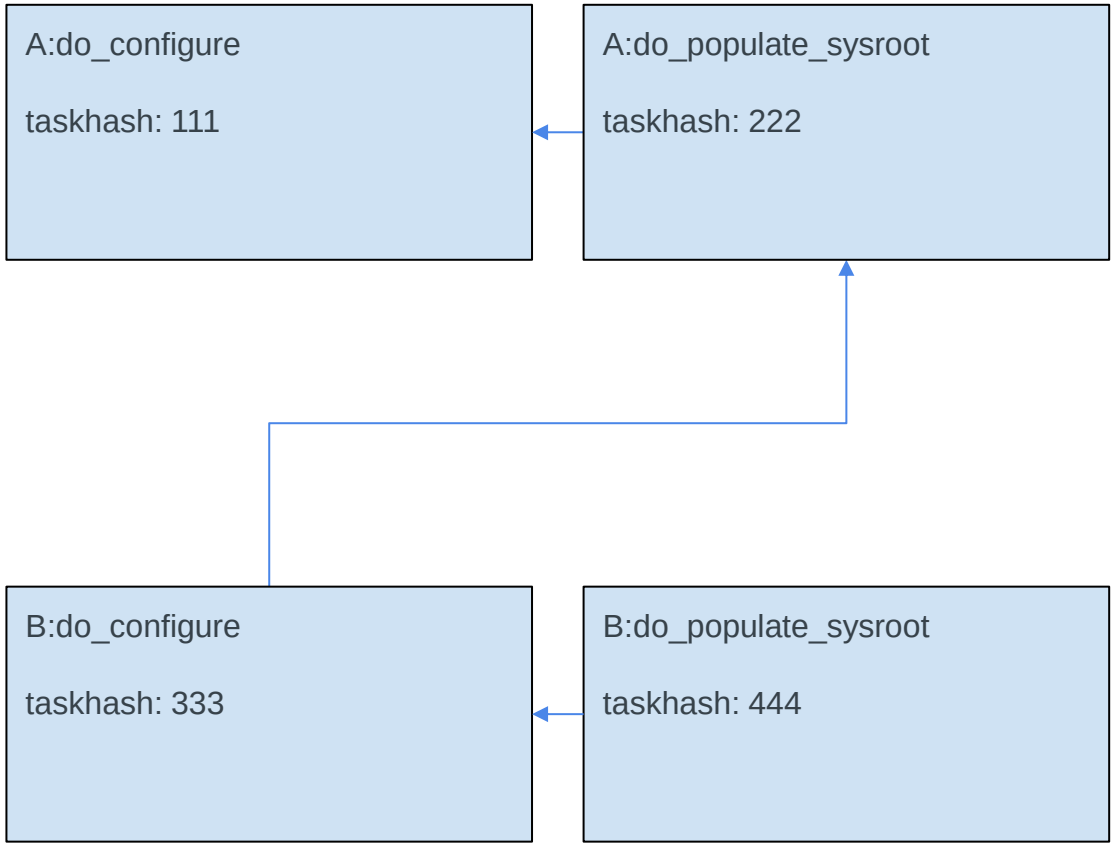
What is the Runqueue?

What is the Runqueue?

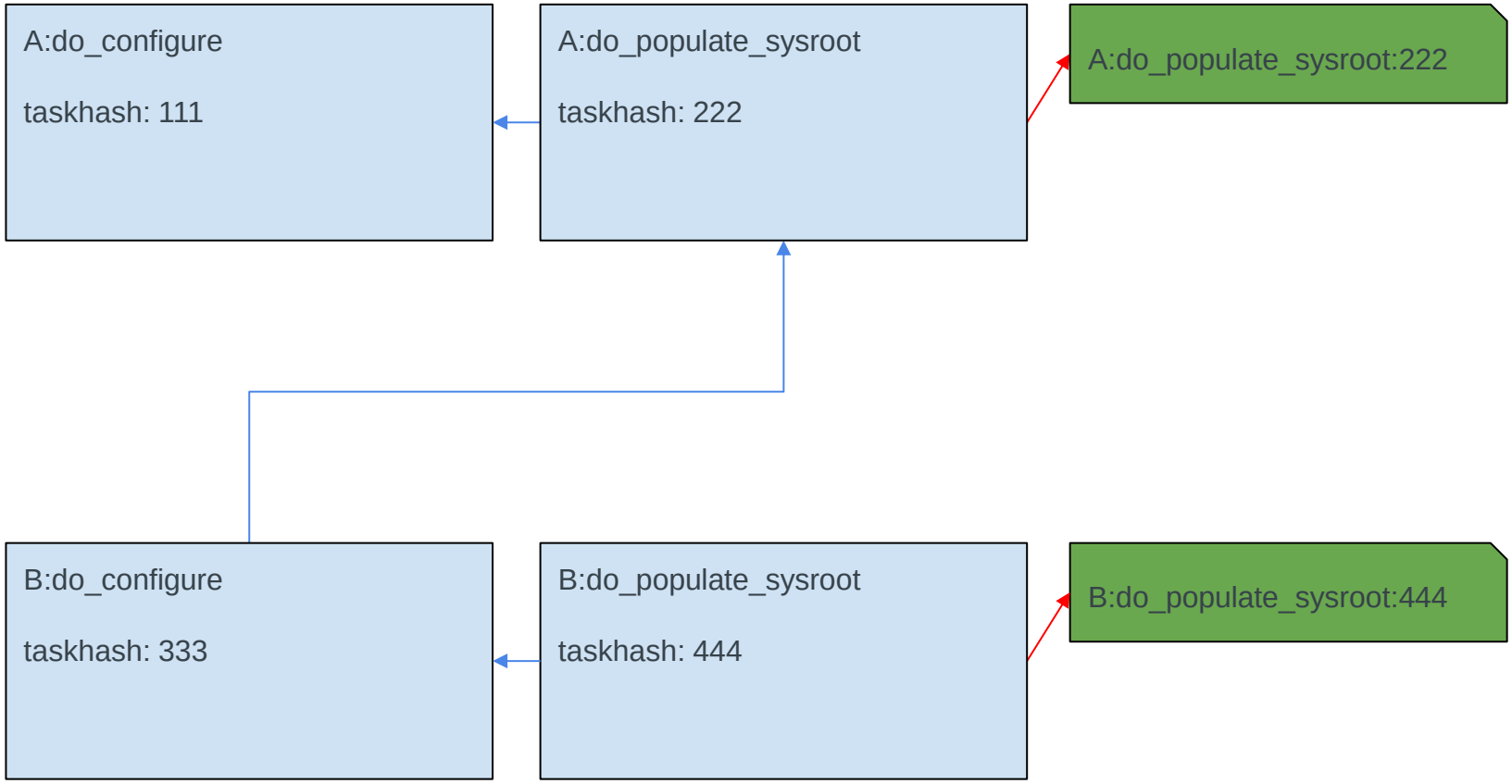
- **The “queue” (tree) of tasks that bitbake will execute for a given build**
  - Records task dependencies
  - Record task state (completed, ready to run, not ready)
  - As tasks are executed bitbake marks them as complete

Traditional Runqueue Execution

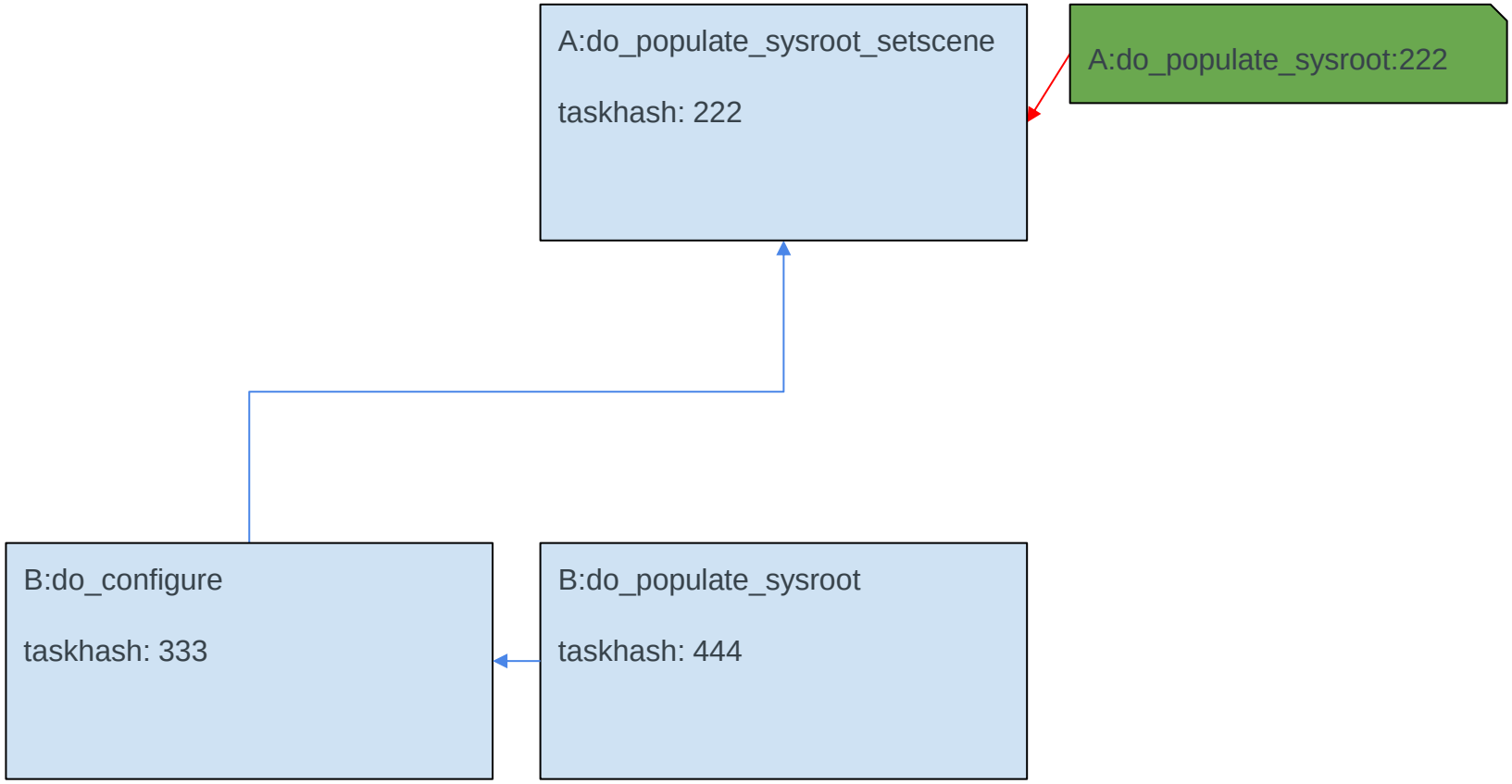
Traditional Runqueue Execution



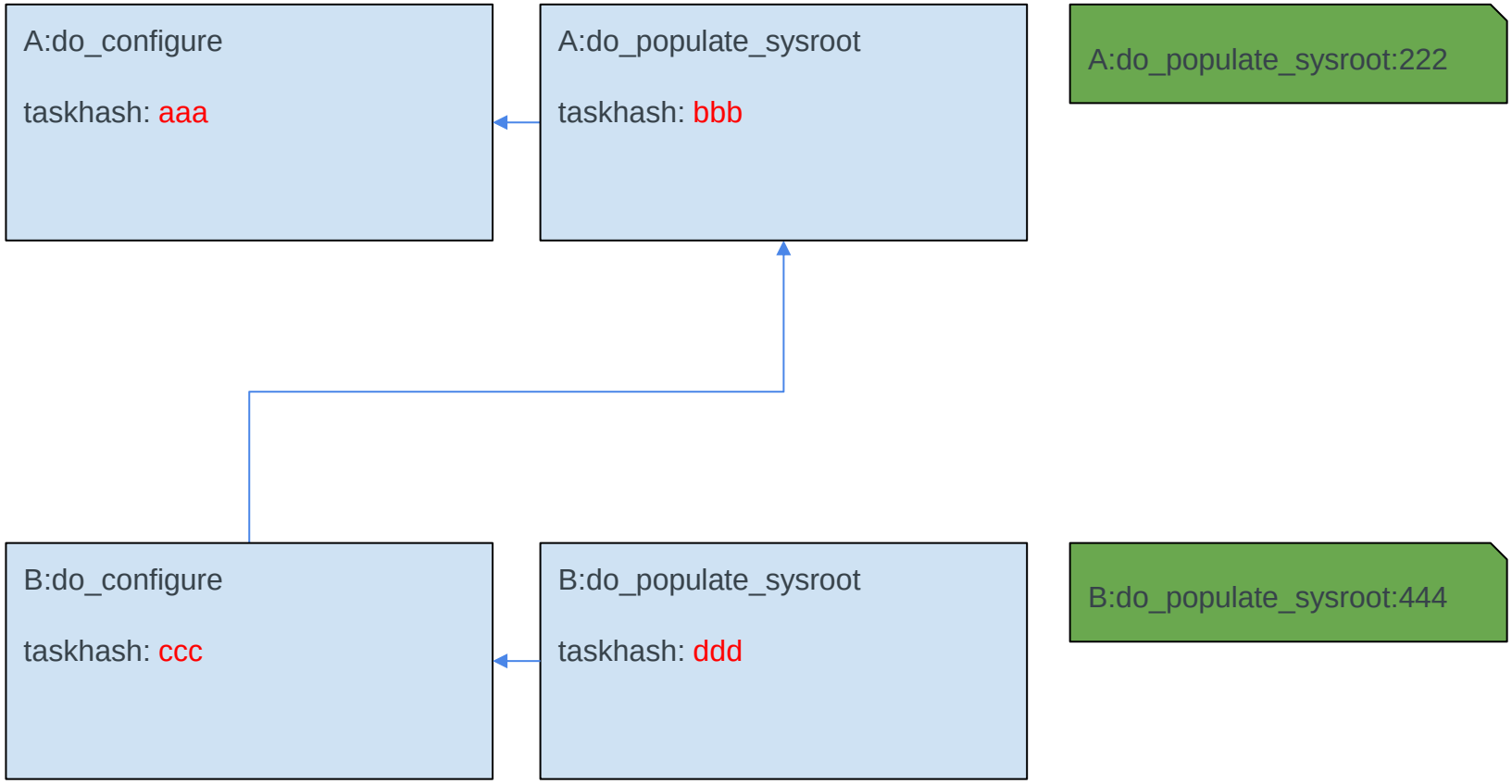
Traditional Runqueue Execution



Traditional Runqueue Execution



Traditional Runqueue Execution

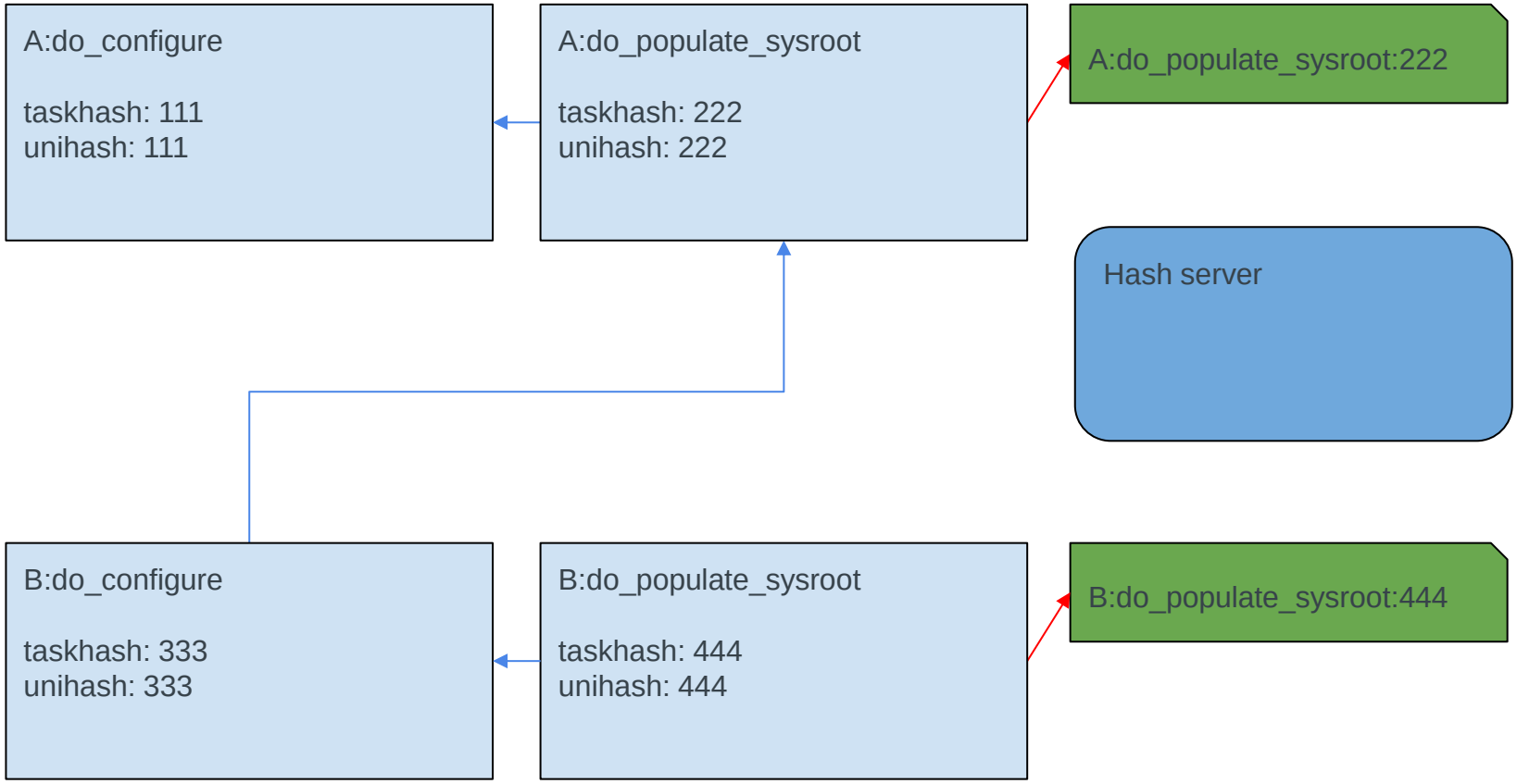


What is the purpose of Hash Equivalence?

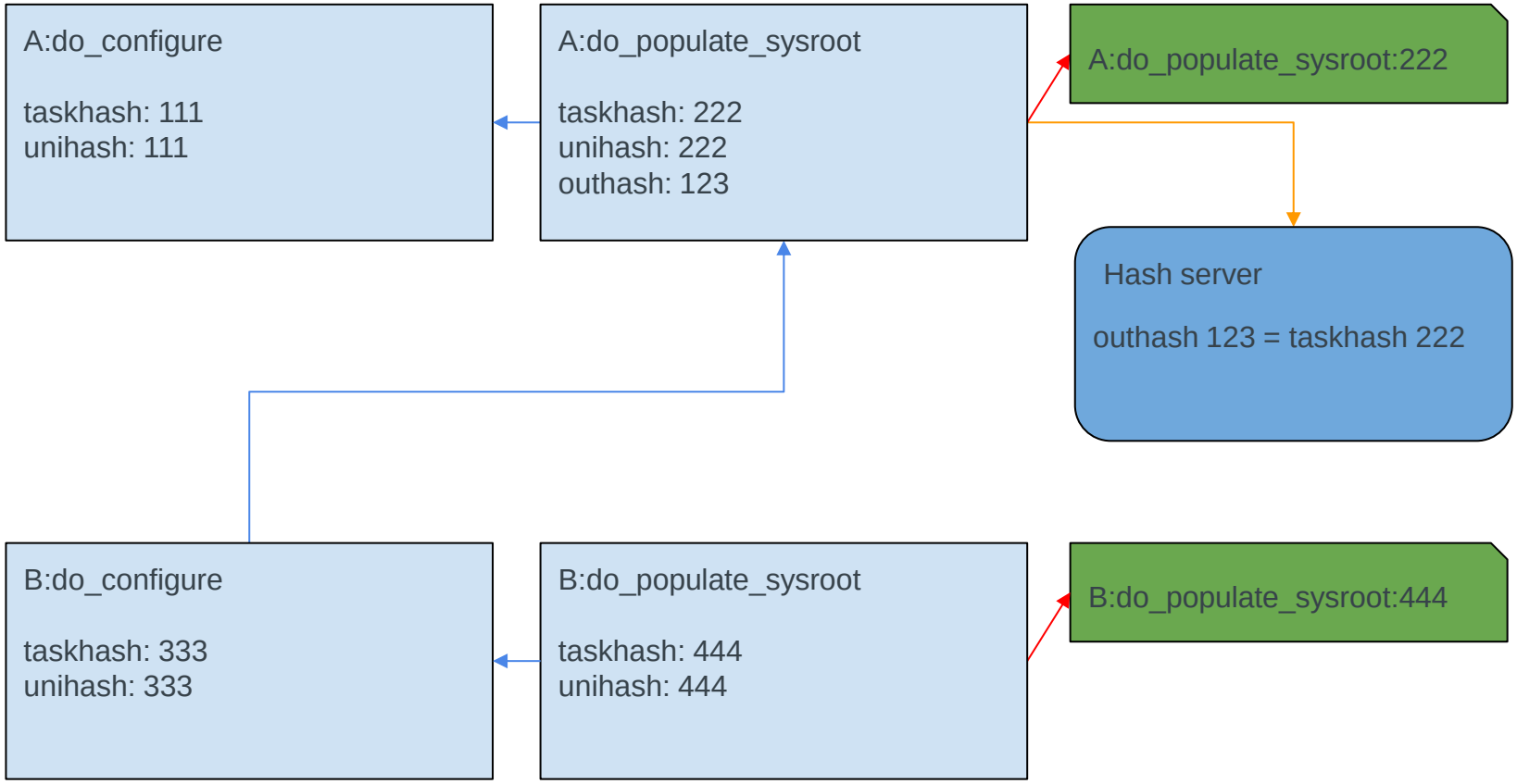
- Improve the reuse of sstate
- Reduce unnecessary rebuilds of recipes
- Reduce build times

Runqueue Execution with Hash Equivalence Server

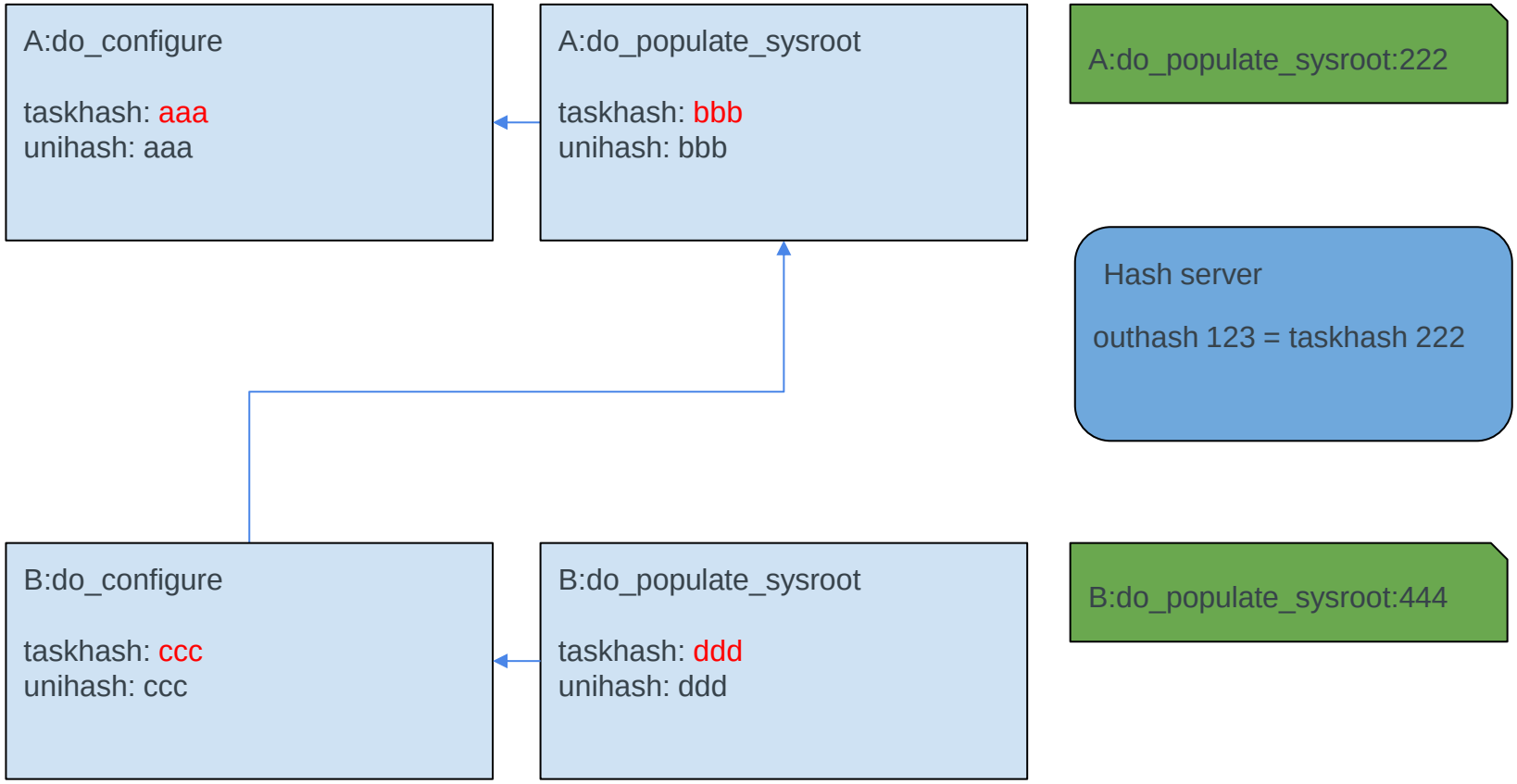
Runqueue Execution with Hash Equivalence Server



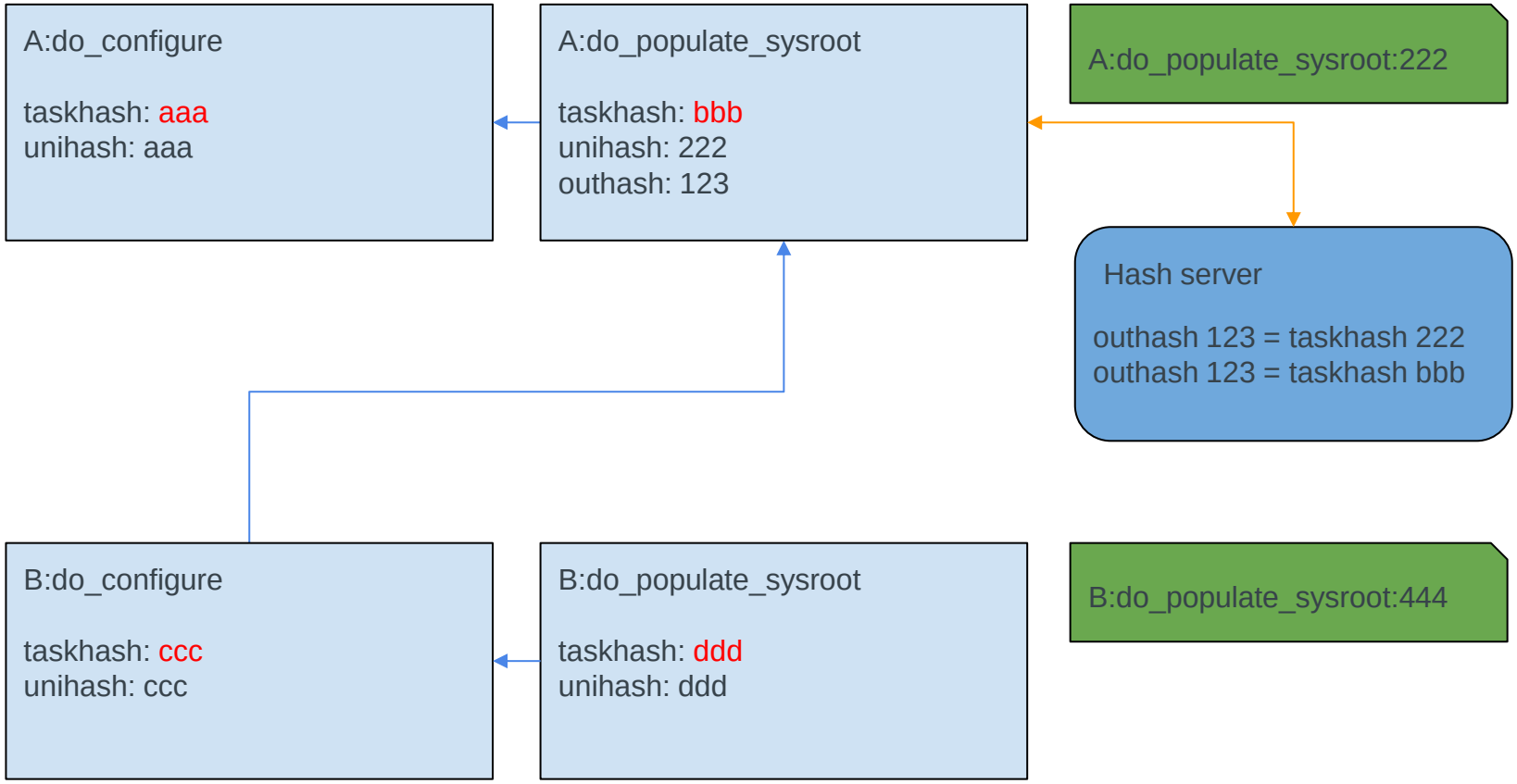
Runqueue Execution with Hash Equivalence Server



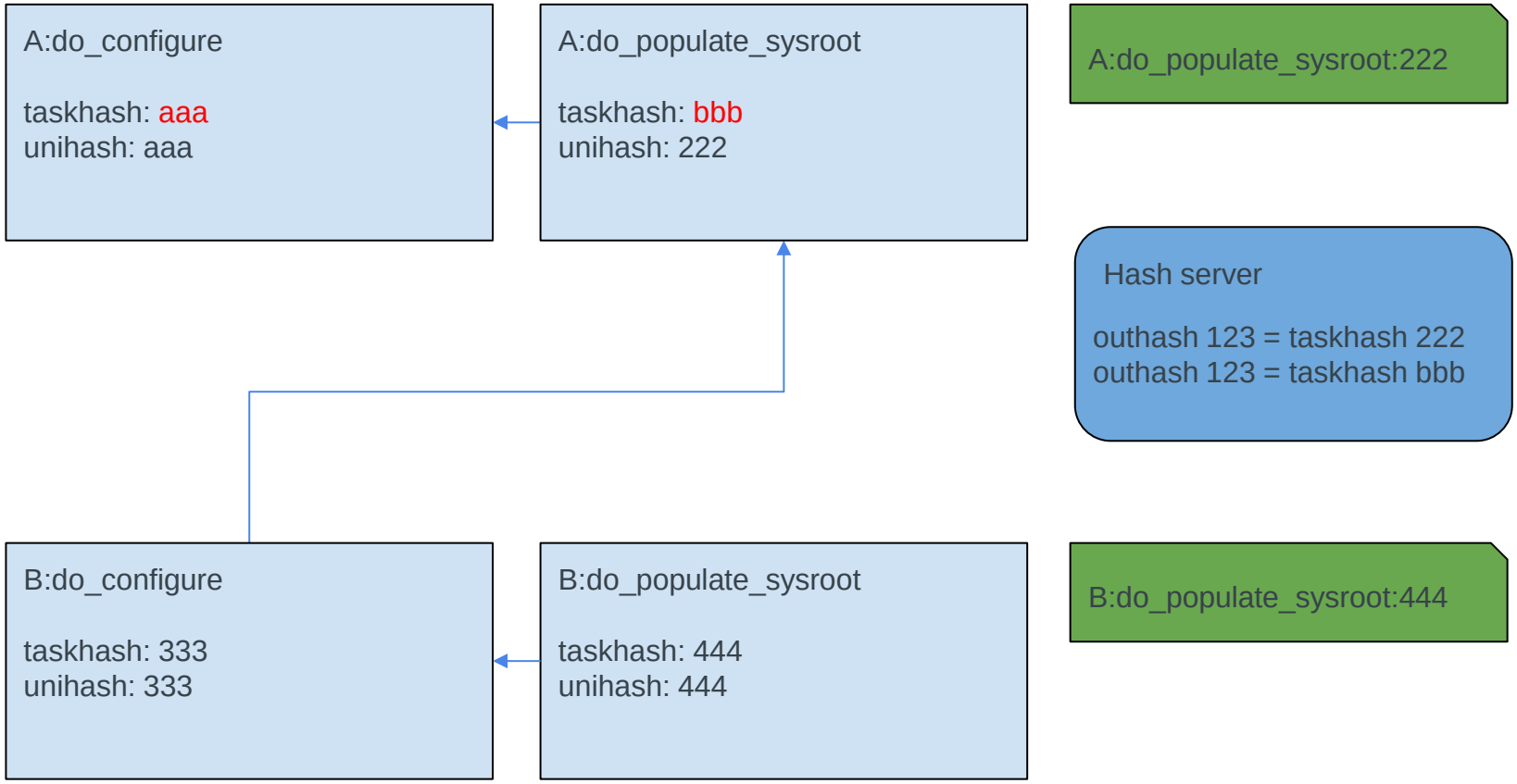
Runqueue Execution with Hash Equivalence Server



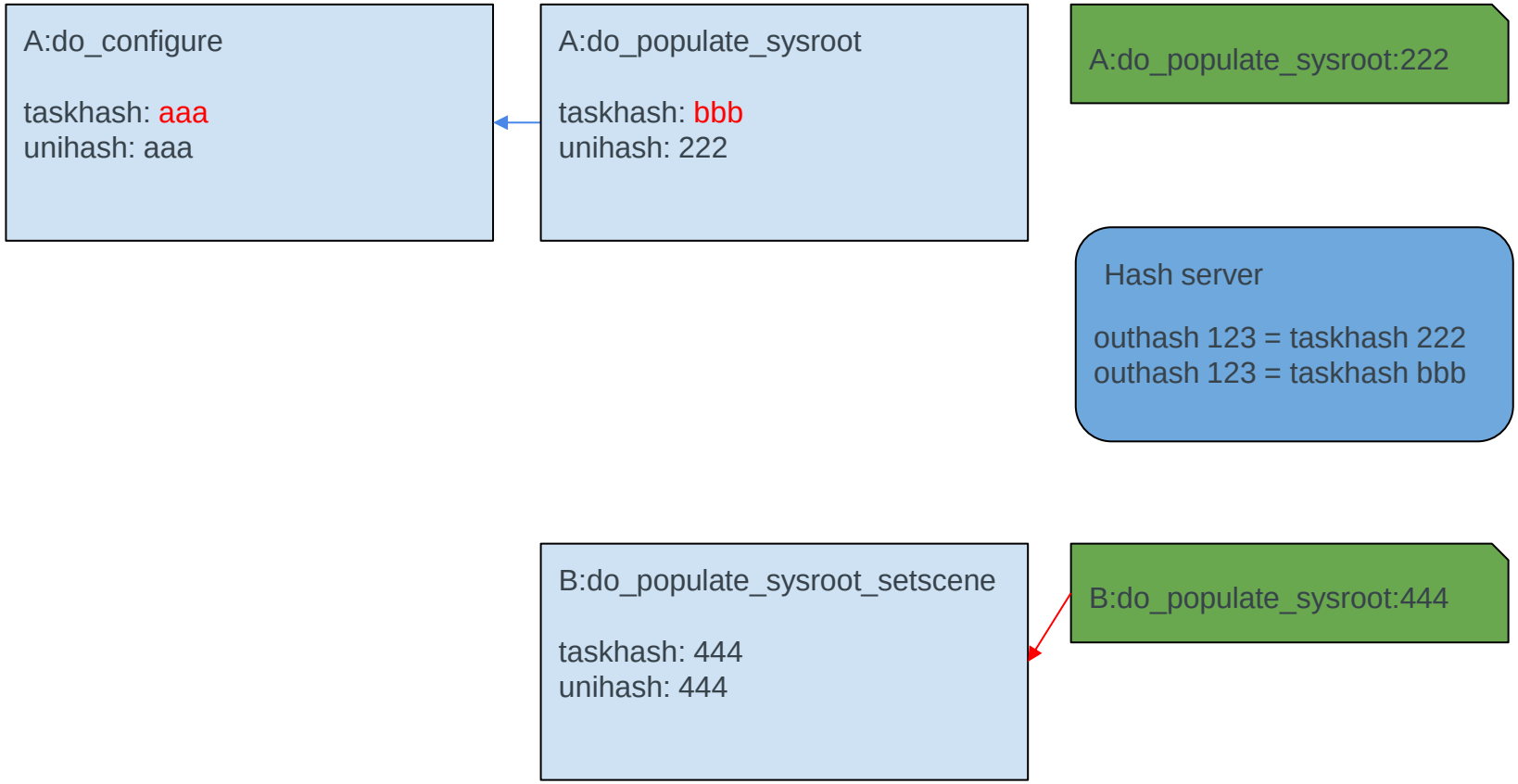
Runqueue Execution with Hash Equivalence Server



Runqueue Execution with Hash Equivalence Server

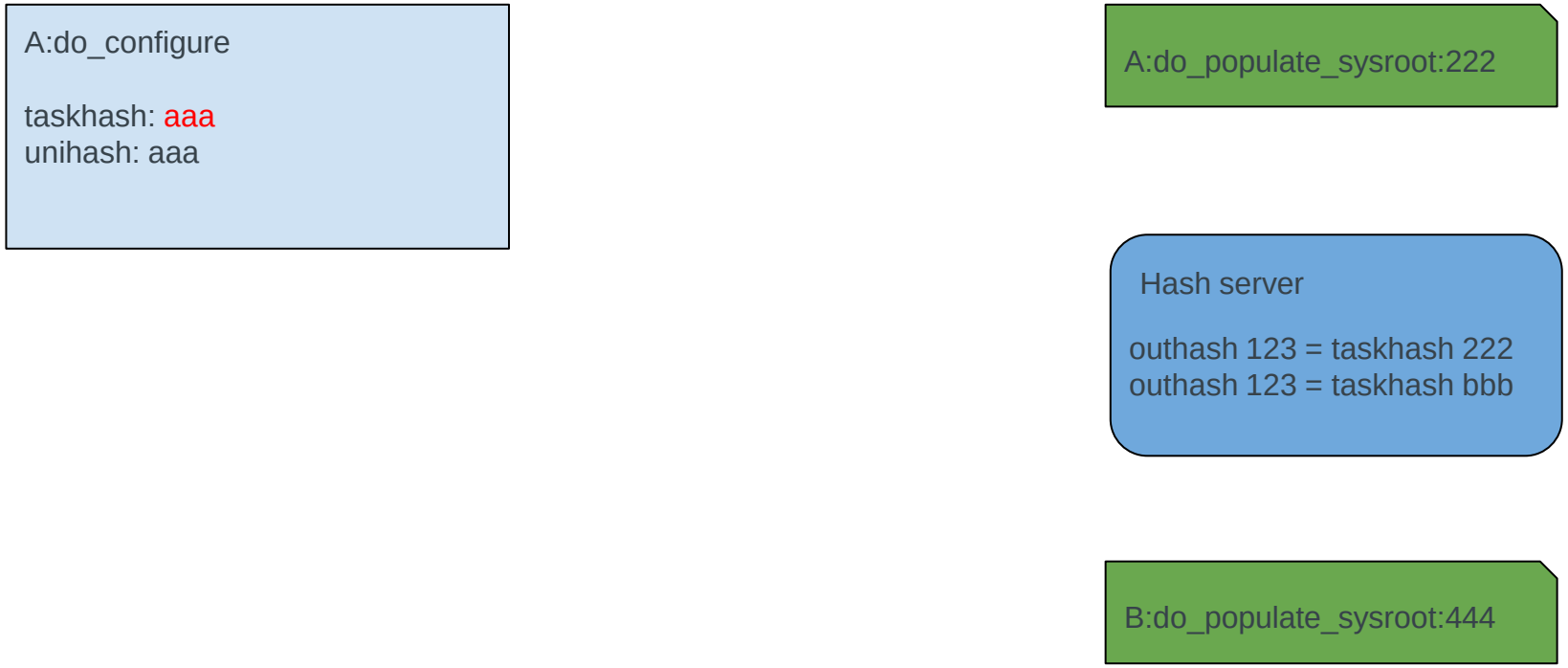


Runqueue Execution with Hash Equivalence Server

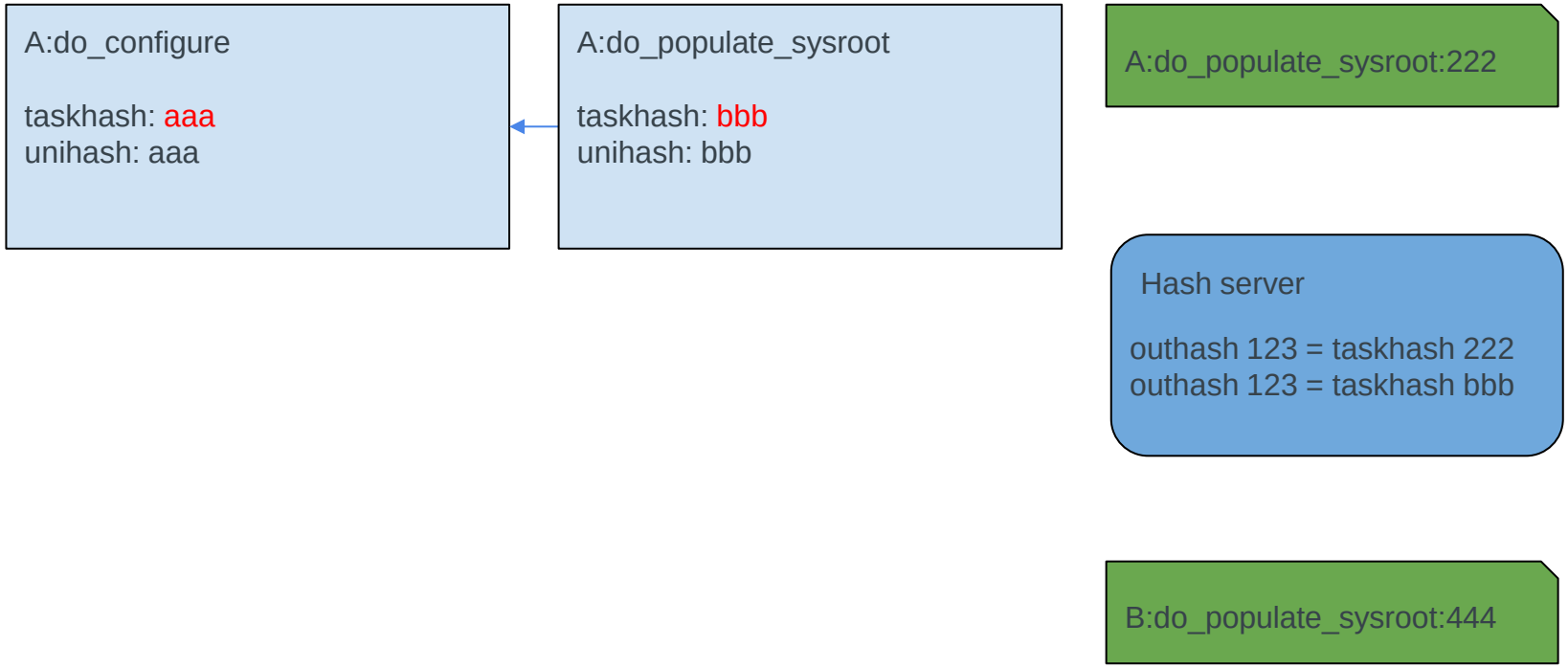


Signature Generation with Hash Equivalence Server

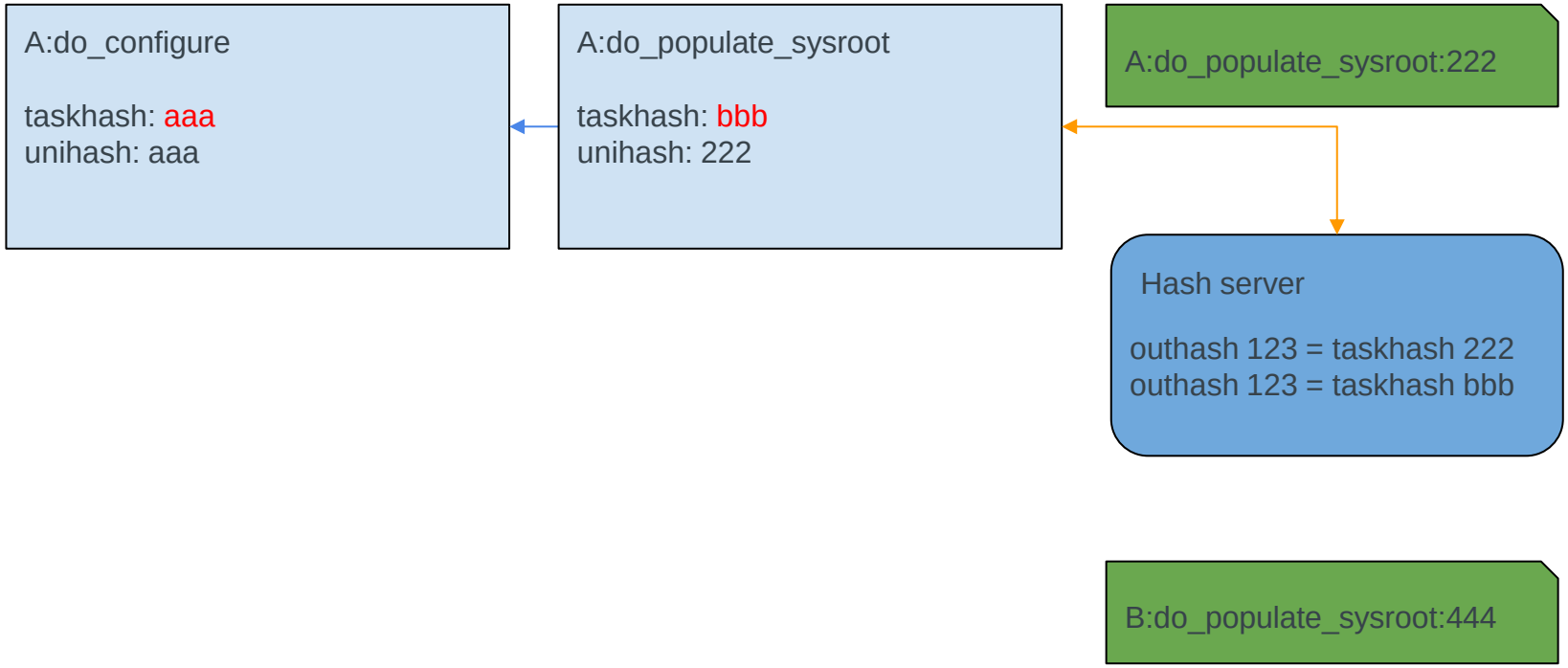
Signature Generation with Hash Equivalence Server



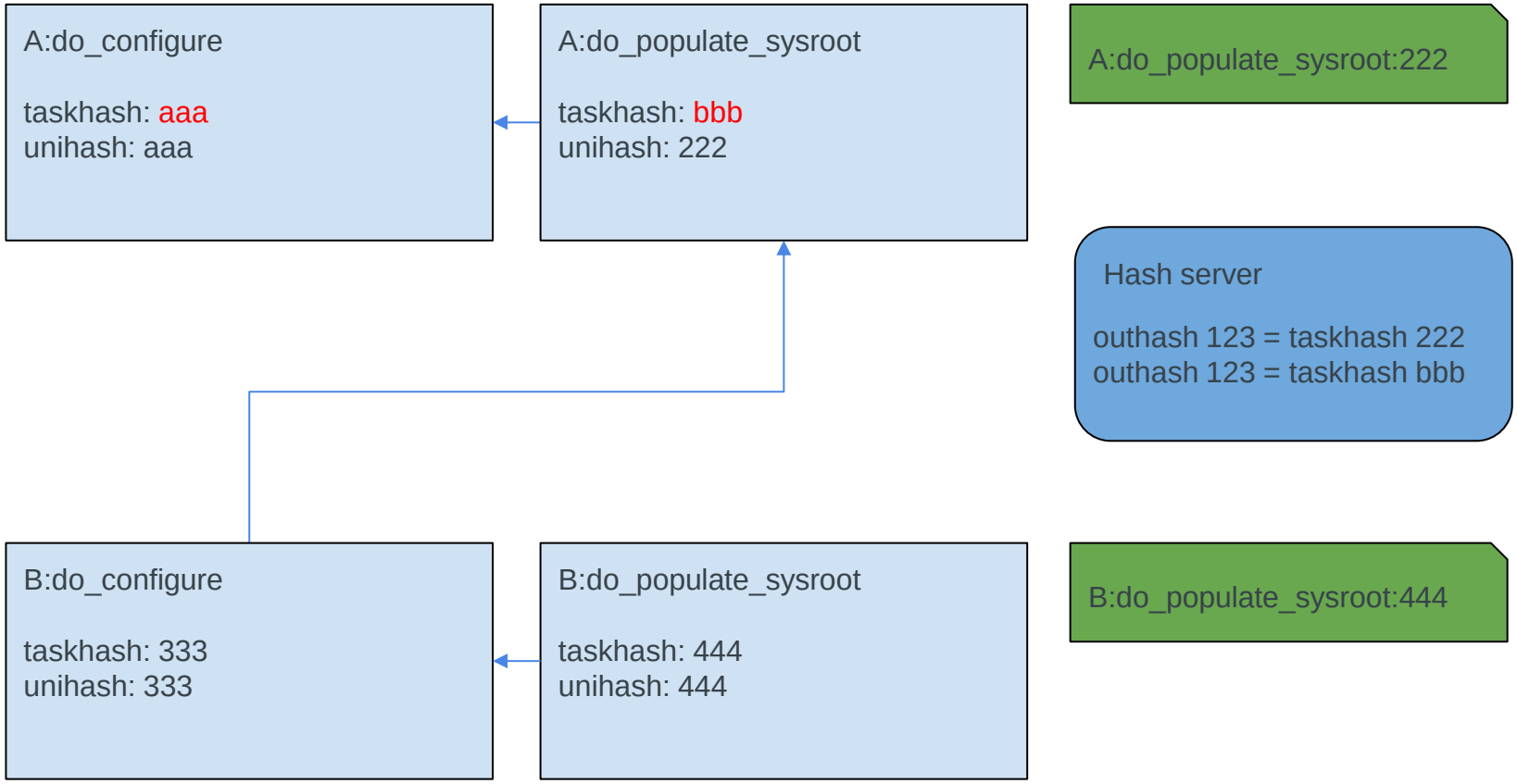
Signature Generation with Hash Equivalence Server



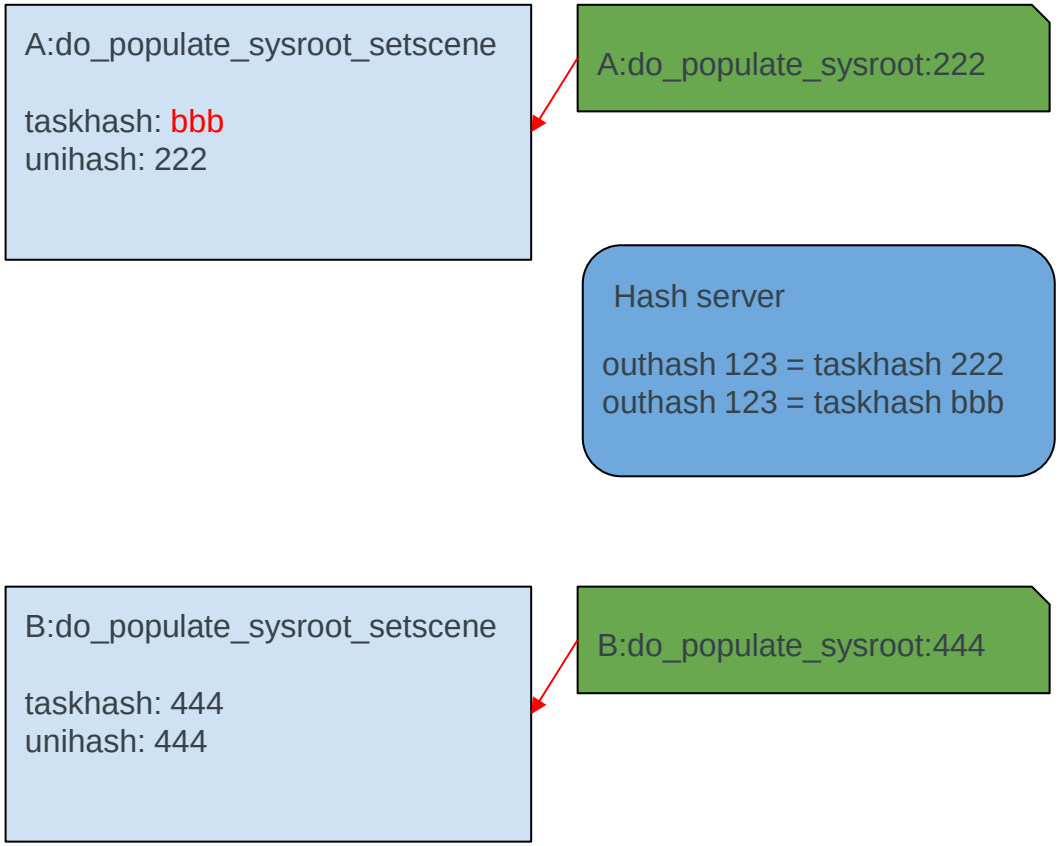
Signature Generation with Hash Equivalence Server



Signature Generation with Hash Equivalence Server



Signature Generation with Hash Equivalence Server



Live Demo & Exercise

The Role of Reproducible Builds

- **Hash equivalence and reproducible builds go together**
  - Better reproducibility means better hash equivalence

Alternative Output Hash Methods

- **The output hashing method can be replaced**
- **Opportunity to implement context-sensitive hashes**
  - ELF Library Symbol hashing
  - Scripting language specific hashing
  - Locking hashes

Questions and Answers

Thank you for your participation!