Automated Testing Summit 2018

Tim Bird

Fuego Test System Maintainer

Sr. Staff Software Engineer, Sony Electronics

Outline

Introduction

Vision

Problem statements

Discussion Areas

Wrap-up

Our apologies in advance...

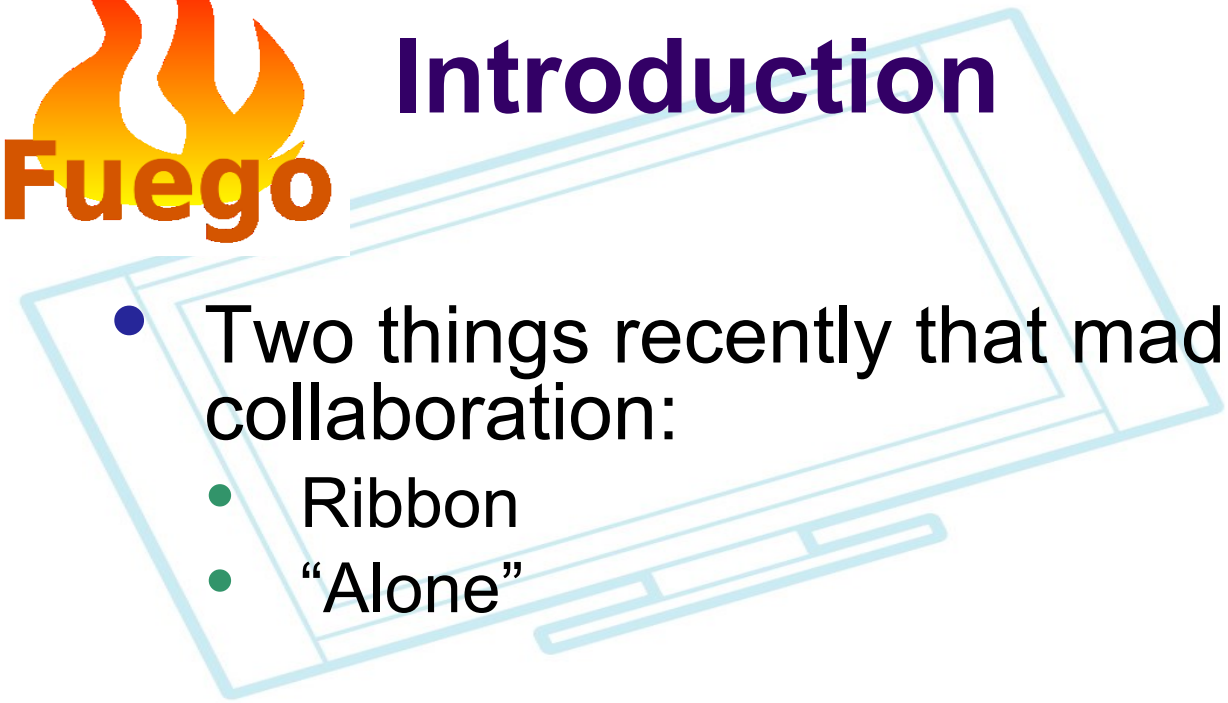
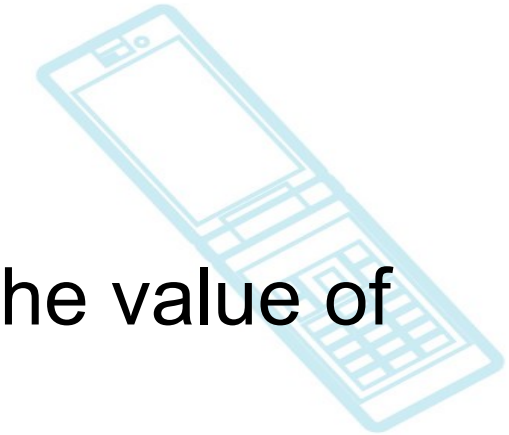
- This will likely be a frustrating day
- Everyone has ideas about testing that they've been thinking about for years
- We can't possibly cover them all in one day
- With 20+ tests and frameworks present, we can't review the details of each one

Goals for today

- Approach a common understanding of the problem space
- Discuss an overarching framework for describing how our different systems work
- Develop common terminology
- Learn how other systems solve problems
- Learn a bit about problems our own systems don’t solve
- Create a path for the future

Introduction

- Two things recently that made me think about the value of collaboration:
  - Ribbon
  - “Alone”

Ribbon

- I got a ribbon from my sister on the present for my birthday

Ribbon thoughts

- Ribbon was inexpensive
  - I almost threw it away
- Ribbon is also a marvel of modern technology
  - dyes of many colors, refined metal, textile, fabrication, distribution
  - Thousands of humans involved in thousands of operations, to bring me a ribbon for less than $1
- Low cost only possible due to high degree of specialization, collaboration and exchange.
- I kept it to remind me of the value of this

“Alone” TV show

- 10 people are placed in wilderness, with clothes and only 10 modern items – completely alone
- Person who can survive the longest wins
- Longest survival time is 87 days

“Alone” lessons

- The same lesson as the ribbon, but from the opposite direction
  - With no specialization, collaboration or exchange of goods or services – a person can’t survive
  - People literally reduced to eating bark
- Very difficult to make your own tools sufficient to survive

Outline

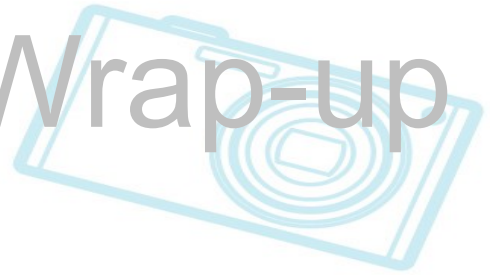
Introduction

Vision

Problem Statements

Discussion Areas

Wrap-up



Vision – super high level

Do for testing what open source has done for coding

- Significant parts of the test process are unshared, ad hoc, private, etc.
  - However, most QA doesn't need to be proprietary
- There are open source frameworks and test programs but more is needed to create an open testing community
- Goal:
  - *Promote the sharing of automated CI components, artifacts, and results, the way code is shared now*
    - Allow components to specialize, and support collaboration between projects

Non-goals for today

- Finish standards for APIs or protocols between systems
  - That's too ambitious, but we can get conversations started today.
- Learn about neat feature of other systems, and start implementing them ourselves
  - That's the wrong approach
  - Instead we should:
    - Identify unique value in our systems, and try to modularize it for re-use by others
    - Identify value in other systems, and start thinking about how to use it in our systems

More concretely...

- I don’t want to add to Fuego:
  - Email-based patch CI triggers
  - SUT deployment abstractions (provisioning)
  - DUT control drivers
  - Centralized results repositories
  - Distributed results visualization
- I want to focus on areas where Fuego is different:
  - Repository of test definitions
  - Sharing of pass criteria and testcase documentation
  - Generalized output parsing system

Outline

Introduction
Vision
Problem Statements
Discussion Areas

Problem statements

- Why are we here?
  - Many aspects of QA are not shared
  - Nobody can do it all themselves
- Tests are viewed as “secret sauce” and are kept proprietary
  - Exactly the same as embedded system software 20 years ago
- Samsung, LG, Sony all produce TV sets
  - Which of these use test software from another vendor?
  - Which of these share their TV functionality tests?

Why are tests not shared?

- No place to share a new test
  - Is that true? What about LTP or kselftest?
  - There *are* open source tests (cyclictest, syzkaller, iozone, lmbench, etc.)
- Often involves lab-specific code
  - e.g. interface to hardware that is unique or rare
- Is often customized to a particular hardware or software configuration on the target
- Test definition is heavily dependent on test framework
  - file format, APIs, architecture

Specialization of tests

- The paradox of generalization and specialization
  - Tests are too specialized to their framework, or their lab, or hardware characteristics, etc.
- Solution is to create more generalized testcases, and allow per-use customizations
  - Ability to customize test (skip lists, customizable expected values, variants)
  - Localized results interpretation (pass criteria)
- Preferably do automatic customization
  - e.g. Benchmark value threshold based on previous results

Factorization

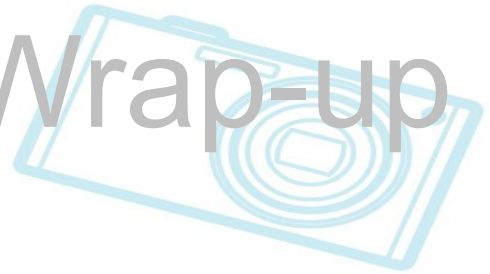
- Different frameworks factor their data and services quite differently.
  - Where operations are performed:
    - 1) central server, 2) on a local host, or 3) on-DUT
  - Party responsible for performing operation:
    - 1) by the test itself, 2) by the framework, 3) by an external service, or 4) by the end user (tester)
  - When are operations performed:
    - 1) during the test, 2) during post-processing, 3) synchronously, 4) asynchronously, etc.
  - Parts of the test definition are in different files, to support per-test, per-board, or per-lab customizations

Fractal nature of testing

- Test features look the same at different levels of abstraction
- Example:
  - Individual testcase has assertions about expectected values
    - actual value different from expected = failure
  - Test suite has aggregation of expected results, with expected results
  - Test plan has aggregation of test results from many test suites, with expected results
- Can do pass criteria, results analysis, reporting at all levels
- But often the features are expressed completely differently at different levels

Outline

Introduction
Vision
Problem Statements
Discussion Areas
Wrap-up

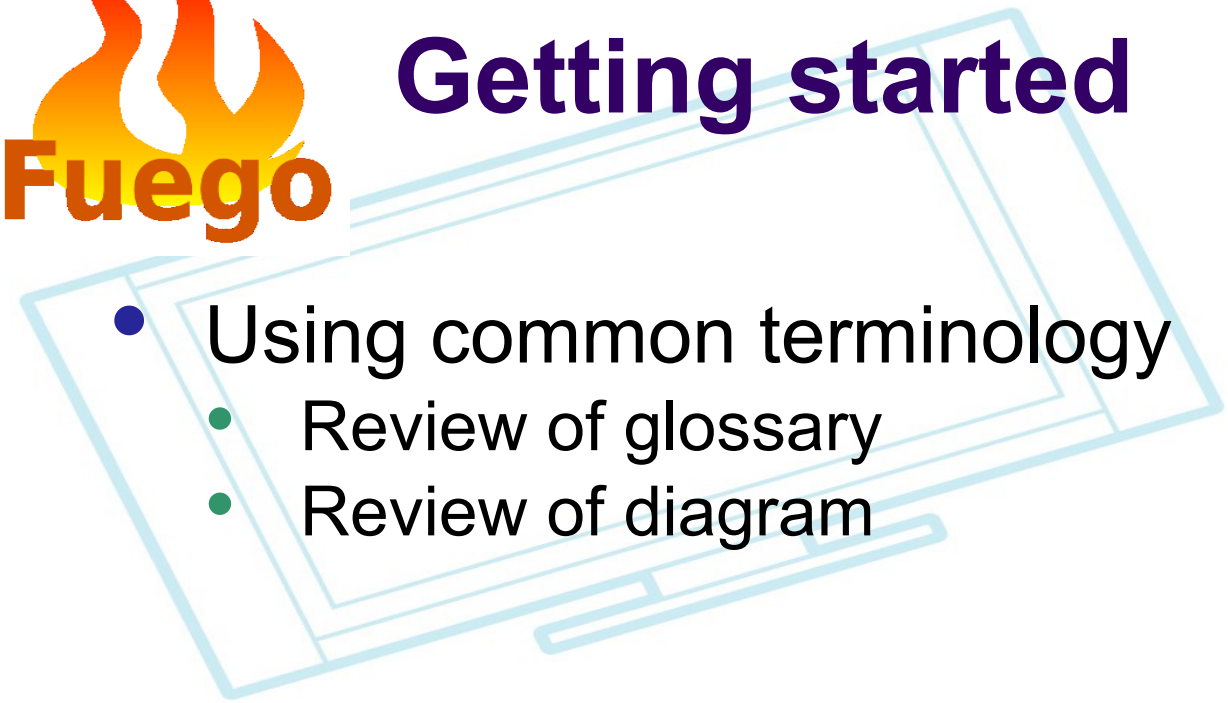


Discussion areas

- Terminology and stack parts
  - Review of glossary
  - Review of diagram
- Different areas of the stack
  - Test Definition, Build Artifacts, Test Execution API (E)
  - Run Artifacts, results format, parsing, Results gathering API (K)
  - Farm standards, DUT control drivers, board definitions
    - APIS F, G (maybe something new?)

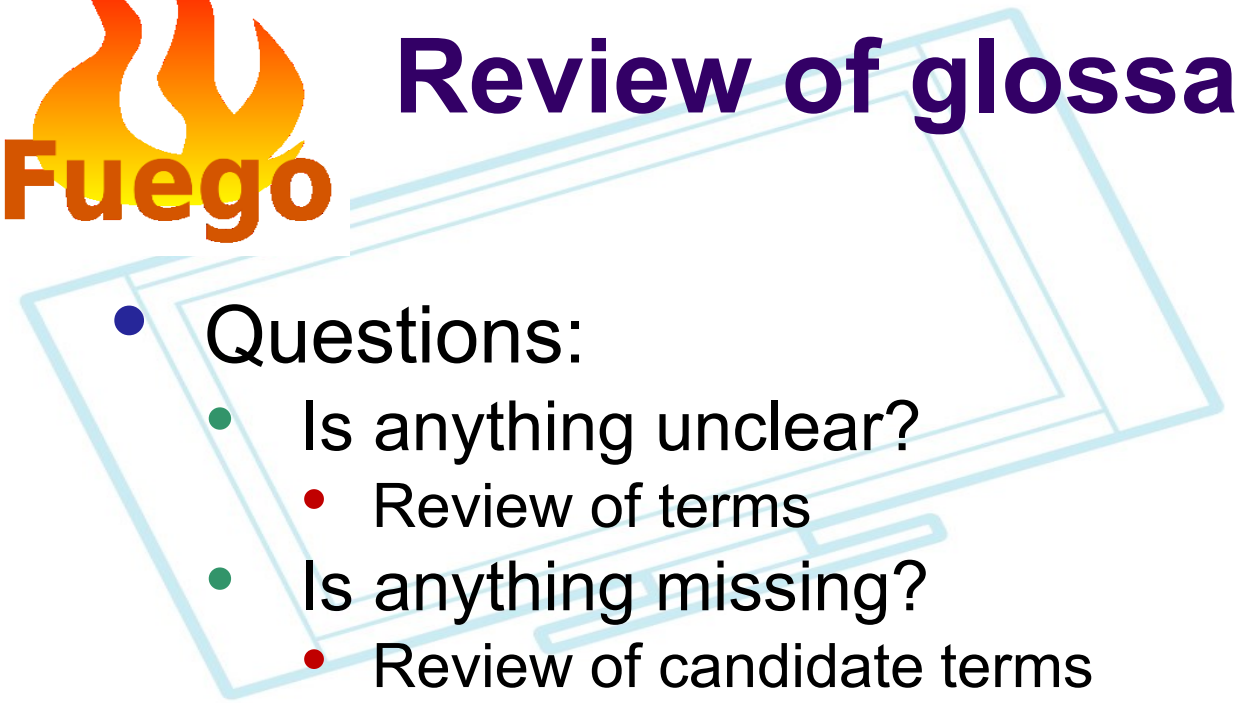
Getting started

- Using common terminology
  - Review of glossary
  - Review of diagram



Review of glossary

- Questions:
  - Is anything unclear?
    - Review of terms
  - Is anything missing?
    - Review of candidate terms



Glossary

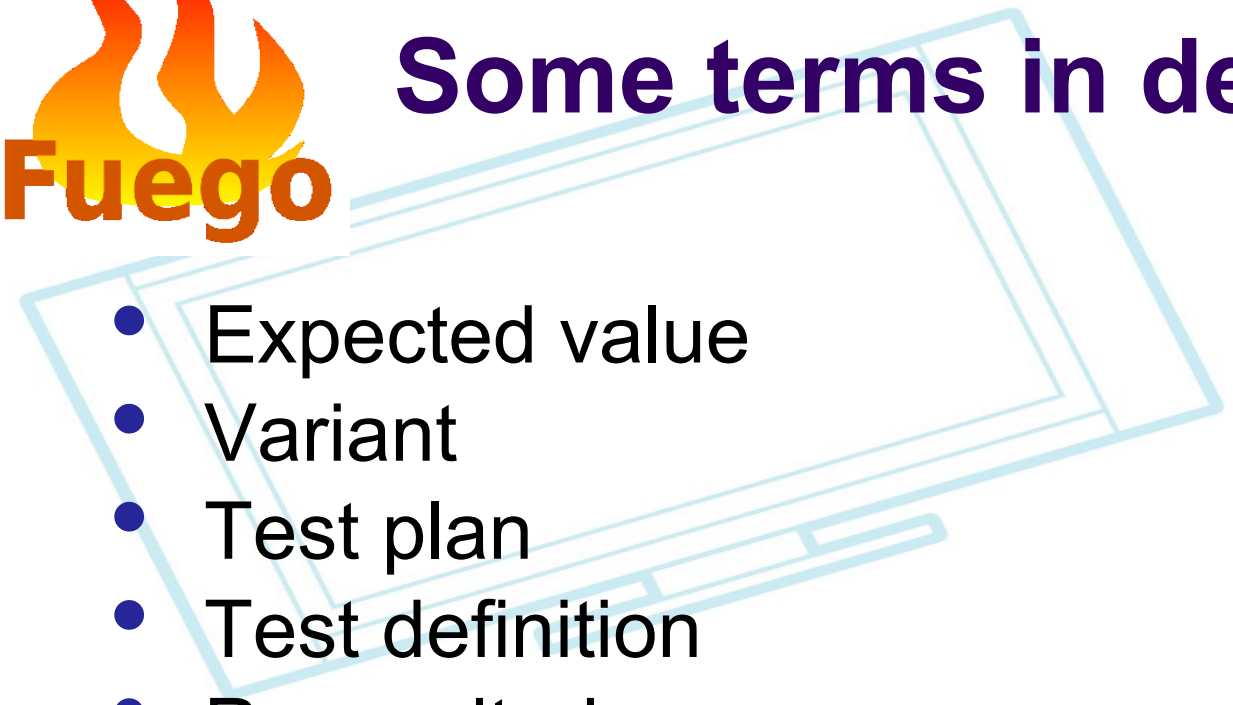
- Bisection
- Boot
- Build artifact
- Build manager
- Dependency
- Deploy
- Device under Test (DUT)
- DUT controller
- DUT scheduler
- Lab
- Log
- Log Parsing
- Monitor
- Notification
- Pass criteria Provision (verb)
- Report generation
- Request (noun)
- Result
- Results query
- Run (noun)
- Run artifact
- Serial console
- Software under test (SUT)
- Test agent
- Test definition
- Test program
- Test scheduler
- Test software
- Transport (noun)
- Trigger (noun)
- Variant
- Visualization

Candidate terms

- **Actual Value** - the value that was seen for an operation performed by a test
- **Expected value** - the value that was expected for an operation performed by a test
- **Feature** - an attribute of a DUT or SUT or test environment that can be used to match tests. (used by labgrid)
- **Device type** - The name of a set of DUTs that have identical or similar features, such that any one of them can be used to run a test (used by LAVA)
  - ** Tim's comment: Some examples would be good. Is there a term for the set of boards that have a particular type? (e.g. something that refers to the pool of boards, rather than the characteristics of the set? Maybe DUT pool?)
- **PDU** - Power Distribution Unit - a piece of hardware used to control power to one or more DUTs (used by LAVA)
- **Interactive DUT access** - the ability to take a board out of automated testing service, for use in interactive testing or debugging sessions (or for some other reason.
  - Alternates: "DUT-offlining"? "DUT reservation"?)
- **DUT Supervisor** - provides connection to the DUT and abstraction for DUT management actions (used by SLAV)
- **Test Profile** - same thing as Test Definition. (used by Phoronix Test Suite)

Some terms in detail

- Expected value
- Variant
- Test plan
- Test definition
- Pass criteria
- Dependency



Expected Value

- Value that is expected result for an operation.
- Many tests have this hardcoded
- However, it's nice if this is customizable
- Some tests allow taking a snapshot, and using that as a baseline
  - This makes it possible to customize the expected value, possibly in an automated way.
- Example:
  - test script that checks for a hardcoded list of services that are supposed to be running after boot vs. test script that checks for a user-provided list of services

Expected value (cont.)

- If a test has configurable expected value, then it is more general, and can be customized by the user for different test scenarios

Variant

- Is something about the environment or command line that can be controlled at test run time
- Example:
  - Dhrystones number of loops
    - Is a command line option that controls test duration
    - If not set correctly, Dhrystone fails on some boards
    - if not default, must be specified per board
- Many command-line options for tests fall in this category
  - They exist to customize the test for particular scenarios

Variant (cont.)

- Variant are hard to configure without domain-specific knowledge
- Would be good to share the most common ones (ie the most useful command line combinations)
- Is a way to customize a generic test
- Need to be able to customize by board, or by file system, or by network
  - Variants can't only be defined per-test
    - Example: cyclictest arguments should be customized for your RT requirements

Pass criteria

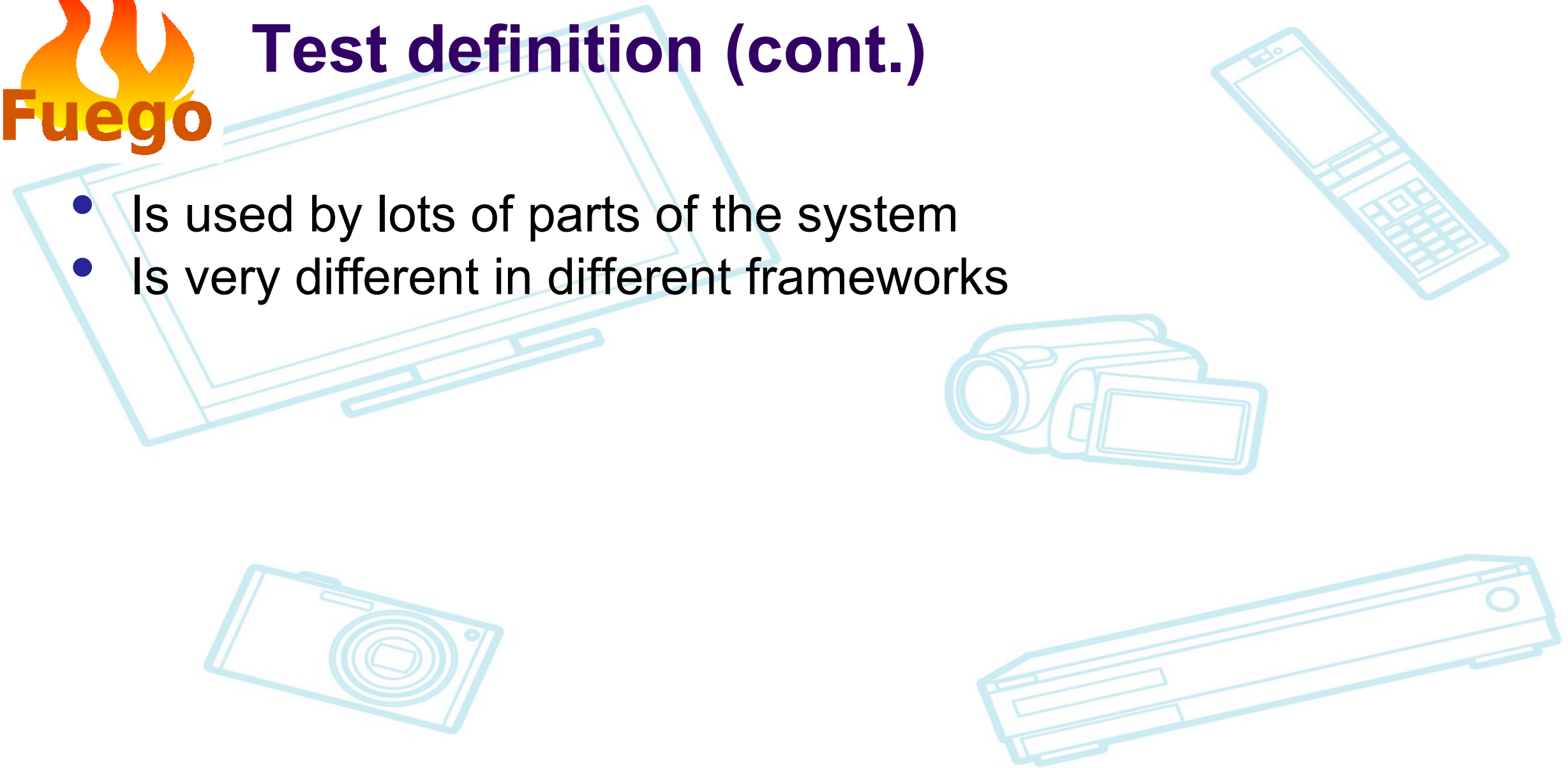
- Describes the requirements (pass counts, fail counts, fail-ok-lists, benchmark value thresholds) that determine the final test result
  - Used for automated test interpretation
  - This determines the ultimate ‘red or green’ result
- Must also be able to customize per board, or per filesystem, or per- some other attribute
- Example: LTP
  - raspberry pi has 28 failures
  - beagleone black has 67 failures, 2 hangs, and 1 kernel panic
  - List of expected failures, or results that are ignored for now

Test definition

- All the data and instructions associated with a test
  - source code, repositories, build instructions
  - dependencies
  - license, author, version, and other meta-data
  - expected execution time (for timeouts)
  - actual instructions to run on DUT
  - monitors and snapshots
  - results parser
  - pass criteria
  - visualization configuration (tables vs. graphs)

Test definition (cont.)

- Is used by lots of parts of the system
- Is very different in different frameworks



Dependency

- A pre-requisite that must be filled in order for a test to run
  - Lots of different kinds:
    - compatible OSes/Distros
    - required file, program, package, library
    - required feature
    - required permissions (eg root)
    - required memory, kconfig, processors
- Action may be to exclude test, cause installation, or change status (sudo)

Review of diagram

- Questions:
  - Anything unclear?
  - Anything else needed?
    - Does anyone's system do something completely outside the diagram?
      - e.g. where is 0day's maillist scanner (used as a CI trigger)?
  - Are the divisions in the diagram workable
    - people have lots of ways they factor this stuff – (where they put functionality, etc.)
      - Despite differences, is the diagram useful to communicate with each other?

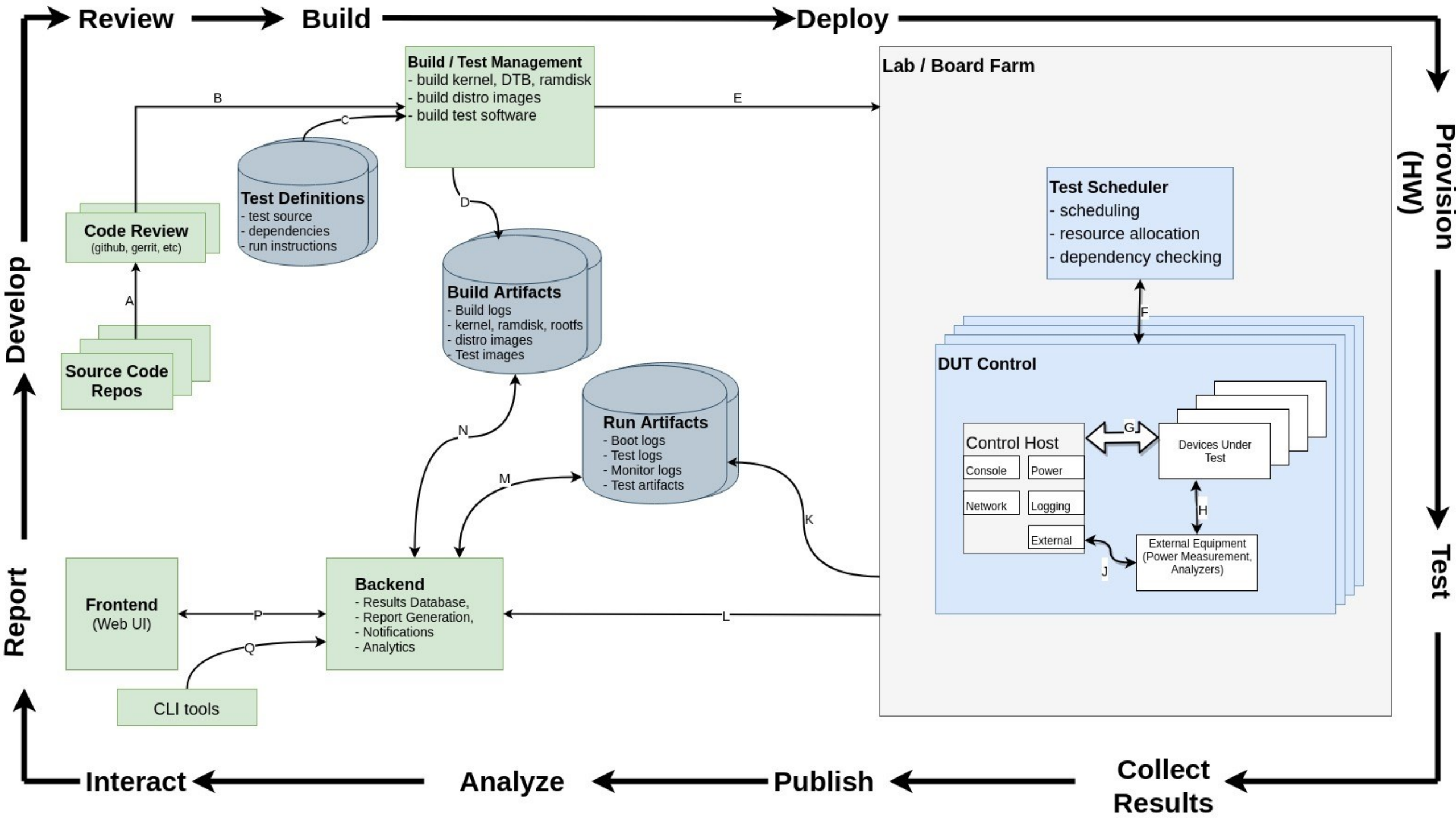

Review
Build
Deploy
Provision (HW)
Test
Collect Results
Publish
Analyze
Interact
Report
Develop
Build / Test Management
- build kernel, DTB, ramdisk
- build distro images
- build test software
Test Definitions
- test source
- dependencies
- run instructions
Code Review
(github, gerrit, etc)
Source Code Repos
Build Artifacts
- Build logs
- kernel, ramdisk, rootfs
- distro images
- Test images
Run Artifacts
- Boot logs
- Test logs
- Monitor logs
- Test artifacts
Lab / Board Farm
Test Scheduler
- scheduling
- resource allocation
- dependency checking
DUT Control
Control Host
Console
Power
Network
Logging
External
Devices Under Test
External Equipment (Power Measurement, Analyzers)
Backend
- Results Database,
- Report Generation,
- Notifications
- Analytics
Frontend
(Web UI)
CLI tools
A
B
C
D
E
F
G
H
J
K
L
M
N
P
Q

Diagram key

- Boxes = processes or services
- Cylinders = repositories (persistent storage)
- Lines = APIs

- Lots of systems have implicit APIs or hardcoded values
  - e.g. save a raw file to local filesystem

Diagram elements – APIS 1

- APIS
  - A = source repository access API
  - B = CI trigger API
  - C = test definition (access) API
  - D = build artifact repository API
  - E = test execution API
  - F = board access API (DUT controller API?)
  - G = DUT control
  - H = hardware API
  - J = test equipment API

Diagram elements – APIS 2

- APIS
  - K = results retrieval and storage API
  - L = backend notification API
  - M = run artifact repository access API
  - P = results query API
  - Q = results query API (command line)

Diagram elements – processes or servers

- Test Manager
- Test Scheduler
- Test Runner (not shown)
- DUT controller
- DUT supervisor (not shown)
- Results data server
- Framework web UI

Diagram elements – repositories

- Test Definition repository (TD)
- Build Artifact repository (BA)
- Run Artifact repository (RA)

How to share

- How to use each other's code?
  - harmonize object definitions
    - test definition, run request definition
  - support APIs
  - modularize pieces
    - e.g. You don't want to download and install all of Fuego just to get the parser code.
- How to use each other's data
  - build artifacts, run artifacts
    - bundle definitions
    - standardized field names
  - shared servers

Specific Discussion Areas

- Before Lunch:
  - Test Definition (TD)
  - Build Artifacts (BA)
  - Test Execution API (E)

Test Definitions

- Storage format(s)
- Repository Access API
- Elements
- Issues:
  - What fields do people have? Why?
  - Could we somehow interoperate?
    - Allow one system to run tests from another?
    - Do the execution models prohibit this?
      - Can this be fixed?

Notes

- opentest stuff
  - sw assets/ build description
  - name, type,
  - kernel reference URL: http://..
    - could be a reference to yocto
    - (called build execution engines)
- Testcase definition
  - test execution engine (lava, batf, fuego, etc.)
  - TEE logic: script: path to test script
  - test params: (variant)
  - hardware requirements (dependency)

Dependencies (TD element)

- Kinds
  - memory, packages, root, hardware, kernel config, files, features, permissions)
- *Expression and management*
- *Actions*
  - *exclude test, install item, change status*
- Side note: Phoronix seems to have come up with a system to express package dependencies that spans even multiple OSes (Linux, BSD, Windows) - that's impressive.
  - Any way to leverage without adopting all of Phoronix?

Build artifacts

- Storage format
- Repository Access API
- Elements
- Issues:
  - What meta-data is stored?
  - Can artifacts be shared?
  - What are the bundle formats? (PTS?, Fuego?, Lava?, 0day?)

Test Execution API (E)

- Elements
- API method
- Endpoints
- Issues:
  - Synchronous or Asynchronous?
  - What fields and why?
  - Is there a ‘run request’ object? Is it persistent?

Lunch: 12:30 – 2:00

Located...?

(CELP Brainstorming session)

- For those interested
- Held during lunch (1:00-2:00)
  - Grab lunch from buffet, and come back to room for discussion
- Discuss current status of Embedded Linux
  - Any projects or features that need LF funding?

Run artifacts

- Storage format
- Repository Access API
- Elements
- Issues:
  - What fields and why?
  - Can results artifacts be shared?
  - What are the bundle formats? (kernelci? LAVA?)
  - What logs, monitor results
  - unified results format

Run artifact creation

- a) results parsing (RA, API 'K')
- b) unified results format (RA)
  - tguids (testcase globally unique identifier)
    - naming, using the same name space for the same test (e.g. LTP)
    - e.g. (test suite, test set, testcase, measure)
  - common meta-data names, types, units (duration, start time, trigger types, etc.)
  - common results names
  - common results format (json, xml, etc.) (or interchangeability between formats)

specific standards (cont.)

- c) common results names (RA, backend)
  - try to align on common meanings for results values?
  - What are different ones? XFAIL

Results analysis

- f) pass criteria (test runner?, RA, backend?)
  - comparison of what people are doing now, and why?
  - when applied?
  - where does it live?
    - see next slide
  - fields, how expressed, how used and edited
  - relationship to visualization

Pass criteria

- Describes the requirements (pass counts, fail counts, fail-ok-lists, benchmark value thresholds) that determine the final test result
  - Used for automated test interpretation
  - This determines the ultimate ‘red or green’ result
- Must also be able to customize per board, or per filesystem, or per- some other attribute
- Example: LTP
  - raspberry pi has 28 failures
  - beagleone black has 67 failures, 2 hangs, and 1 kernel panic
  - List of expected failures, or results that are ignored for now

visualization

- h) results colors (frontend)
- i) chart configuration
  - How does user customize visualization? Is it persistent?

Board farm standards

- Required operations for board management (API G)
- Integrating lab/DUT management with the test system (API F)
- DUT controller drivers
  - What drivers are needed: power, network, USB, button, relays, serial, bus control, logging?
  - Can the driver interfaces be standardized? (what language?)
  - This is an API (not displayed) on the Control Host, to the boxes inside it in the diagram.
    - How to share? (what repo? Who manages?)
- Board definitions? Lab definitions?
  - what fields and why? (format?)

Board farm standards (cont.)

- What API style for API F?
  - cli?, network?, USB? (I've seen all these)
  - discoverability
- Hardware standards for DUT management
  - Best practices for DUT makers
    - don't require a button press to boot
    - support update mechanism aside from manually rewriting the SDcard
    - buttons needed for automation should have pins
      - etc.
    - hardware interfaces that are nice to have on board (and are physically accessible)

Board farm standards (cont2.)

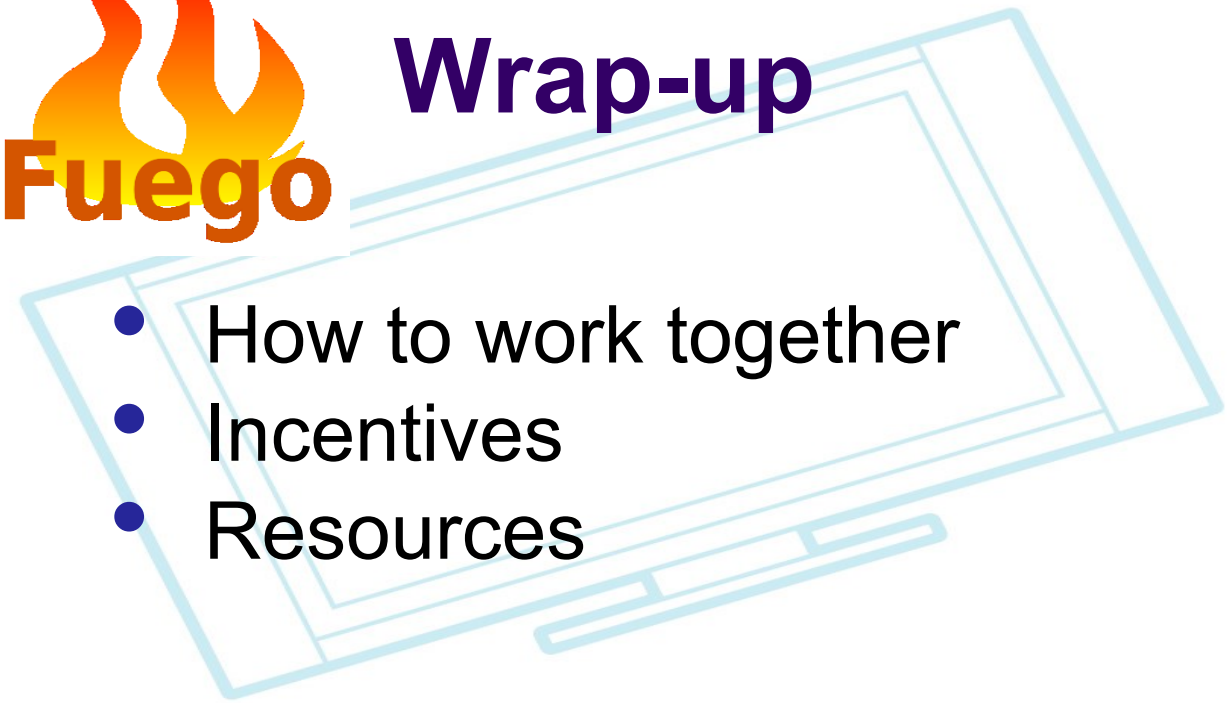
- Required operations for test equipment? (API J)
  - example: monitor power during run
  - synchronous or asynchronous?

Shared hosted services

- Results aggregation (RA, backend)
  - candidates: kernelci, LKFT?
- Build services - (BA, build/test management)
  - candidates: kernelci, kerneltests
- Test repositories (TD, BA)
  - candidates: phoronix, Fuego, LAVA?, YP?
- Visualization (backend, frontend)
  - candidates: kernelci, squad

Wrap-up

- How to work together
- Incentives
- Resources



How to collaborate

- Going from monolithic systems to modular, interworking systems?
  - How to do it given the wide disparity in systems?
  - How do the different systems integrate, communicate requirements, etc.
- Systems have different languages
- Systems have different division of labor (!!)
- Systems have different execution models
  - e.g. Fuego = test-runner based; PTS & LTP = DUT-based

Setting standards

- Who will do it?
- Where can we standardize?
- Who benefits?
  - Finding or enumerating incentives to avoid fragmentation

Process going forward

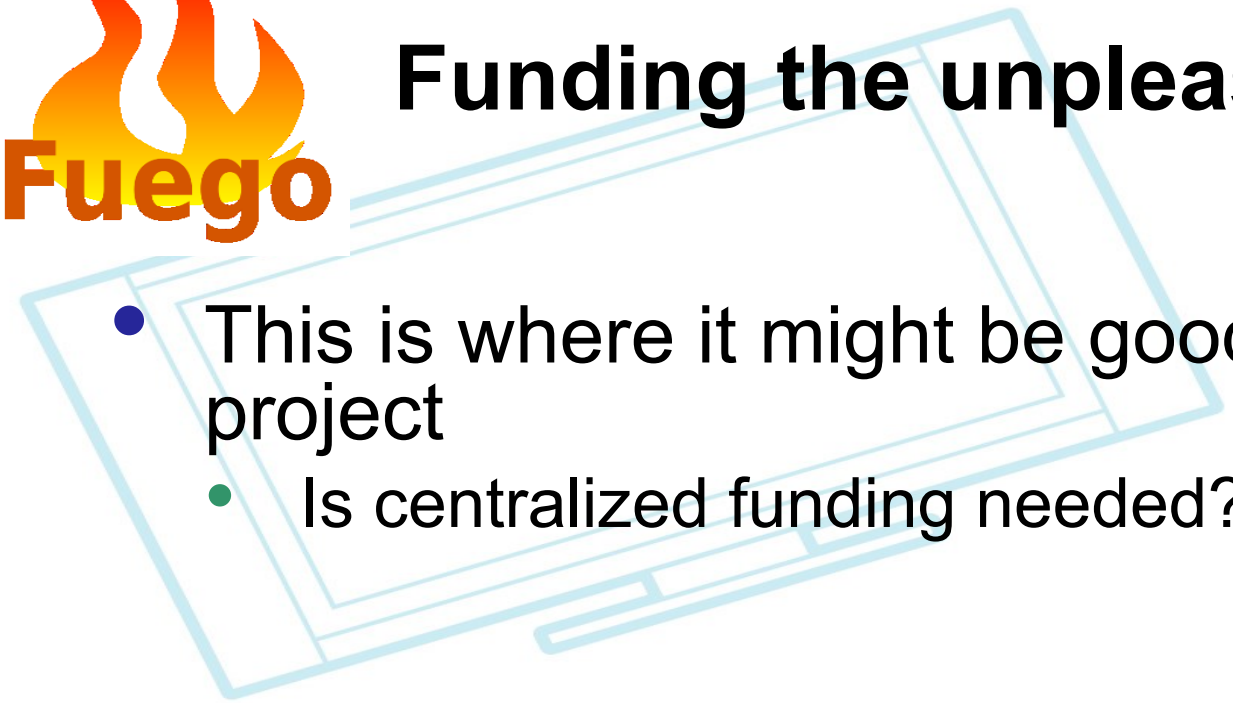
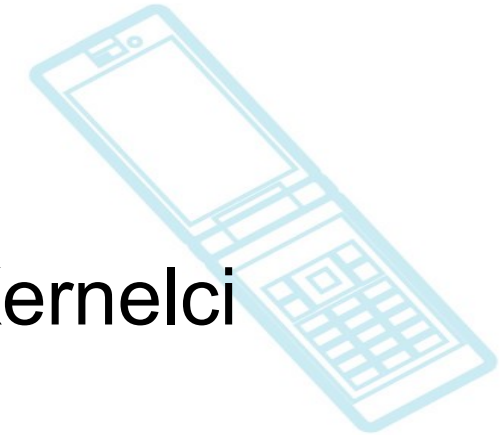
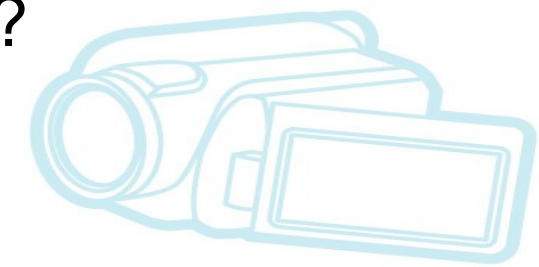
- Next event?
- New mailing list?
- Is anyone willing to take work assignments?
  - ie write standards documents, organize meetings, implement shims, perform compatibility tests, etc.

Incentives

- Nobody wants to commoditize their own layer
- People still need to perform their own testing
  - Which means they need all parts of their current monolithic CI framework, while they modularize parts for re-use by other systems
- It's hard to maintain software you're not using
  - e.g. DUT control driver for hardware not in your lab, or tests that you don't use

Funding the unpleasant work

- This is where it might be good to mention the Kernelci project
  - Is centralized funding needed? good?


Fuego

Ideas

- what tests need to be supported?
  - boot-time
  - run-time
  - package-based (package unit tests)
  - driver (hardware specific?)
    - requiring specialized hardware external to board (e.g. canbus simulator, hdmi frame-grabber)
  - multinode
    - how to allocate/schedule multiple pieces of equipment for a test (e.g. 2 or more nodes for a network test)

Ideas (cont2)

- results reporting
  - centralized server and API to it (kernelCI json?)
- how to define standards
  - de-facto only? (dominant project? (cough, LAVA))
  - documents?
- What to do with survey results?
  - still need to add additional clarification responses