EOS testing

Overview:
Who tests what (and how) around EOS / CERNBox

Q: areas that need attention
Q: areas that are over-tested(?)

in-tree

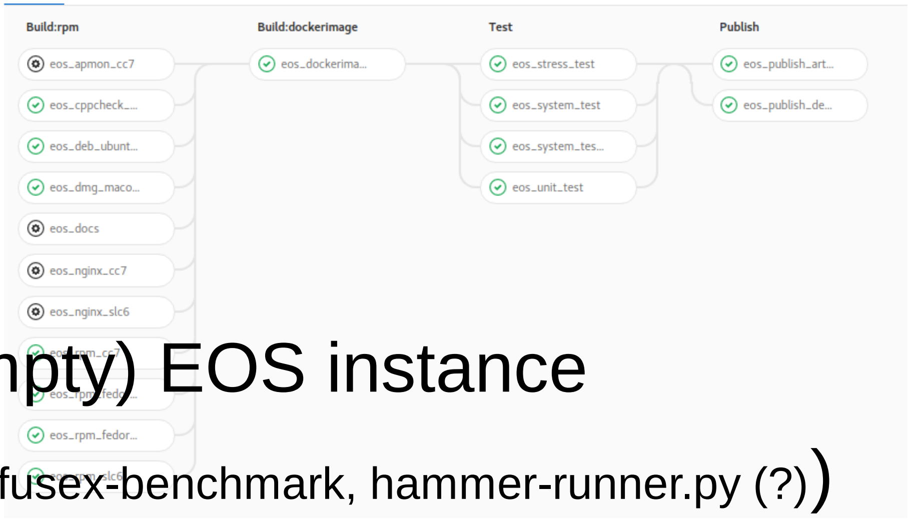
- In-tree means no “expected to fail” tests - these sit on a branch that succeeds them
- “eos-instance-test” link:
  - ~450 functional tests (and some local “stress” tests)
  - Runs as ‘root’ on MGM
  - Shell-based driver script
  - Several compiled executables
  - In main ‘eos’ GIT repo
- Google test unittests link
  - C++-level unit tests
  - (recent addition?)
- “eos-fusex-certify” link
  - FUSEx-specific functional test
    - Also “git clone” and compilation
    - Pulls in “eosclient/microtests” and “eosclient/functional” (see next slide)
  - Shell-based driver script
- “EosFuseTests” link
  - FUSE functional test
  - Compiled executable

Out-of-tree

- “eosclient-tests” link
  - Dan’s microtests (originally meant to highlight performance issues)
    - 3 driver scripts: CI, standalone, timing results →Grafana
  - Functional tests based on user tickets
    - No driver script
    - “interface”: $1=writeable directory ; exit 0 = OK
    - Some “known to fail” (hardlinks)
- “Massimo’s batch test”
- Dan's “fsping” link
  - Latency for writes to become visible between 2 clients
  - Manual: No good/bad/fail classifier?
- Rainer: ex-AFS corener cases
  - Single-byte writes, data propagation, sendmail locking, binaries..
- EOS functional tests (XSLS) link
  - Cover xrdcp, lcg-cp (SRM, gridftp)
  - Timeouts =~ max. latency bound
  - Perma-”to-be-rewritten” status

Test frameworks

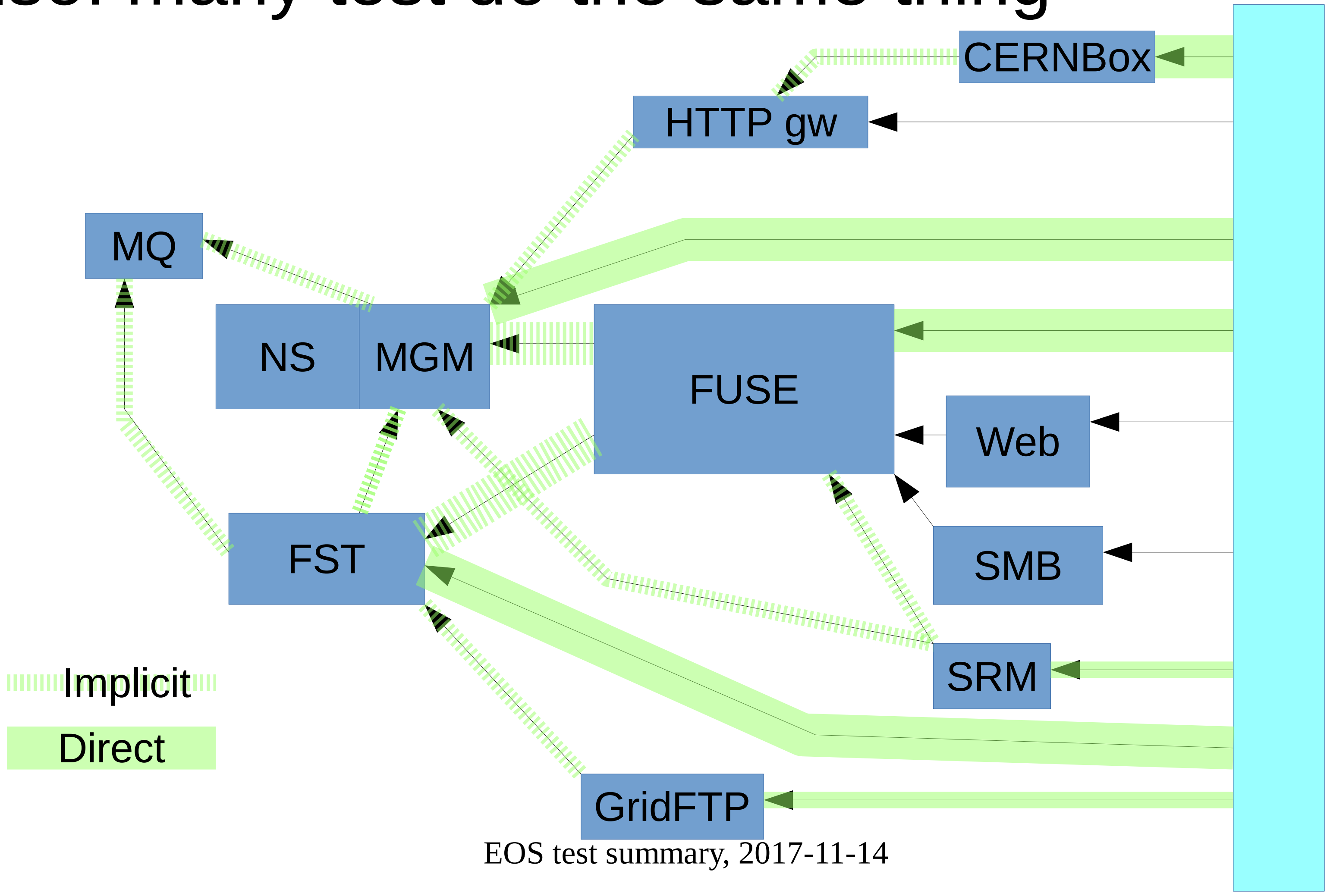
- Gitlab-ci link
  - Runs on every commit
  - Test builds containerized (empty) EOS instance
  - Test definitions: link (microtests, fusex-benchmark, hammer-runner.py (?))



- Smashbox Kibana
  - Matrix of simple sync client tests (OS / syncclient_version / endpoint)
  - Could add direct filesystem-level endpoints: FUSE, SMB (link to FS access)

Coverage

- Impressive – but not exhaustive
- Also: many test do the same thing



Missing testing?

- Many tests are simple: functional, sequential, single user “expect-to-work” cases
- Stress tests: aim to show “good behaviour” for simple load, not discovering limits
- Error conditions? Packet loss, hanging connections, client or server “goes away”
  - Desired behaviour usually undefined, but not “crash”.
- “Concurrency”?
  - Time-order dependencies (who gets to see what+when)
    - Locking (between clients, between servers)
  - (Graceful) behaviour under overload
- Confidentiality
  - ACLs respected?
  - (inadvertent) false sharing – client cache?
- Security
  - Way too many errors result in a crash (also in “underlying” Xrootd)
  - Some protocols are exposed only to “friendly” traffic so far (MQ?)
  - Resilience to active attacks? “how would **you** kill EOS?”

Maximize returns from tests?

- Many people are eagerly waiting for FUSEx

  = free testing manpower
- Project should defined their “preferred” testcases
  - Simple / minimal effort for tester:
    - some odd (reproducible!) error →script→ testcase
  - Usable by the project. eg.
    - “exit != 0” is bad
    - First argument will be a writeable location
  - Actually being tested against (minimal effort to add to CI)
  - Some tests will be “known to fail” for months – how to mark as such?