LA-UR-20-26068

Auto-Mounted SquashFS for Charliecloud Containers

HPC Showcase 2020

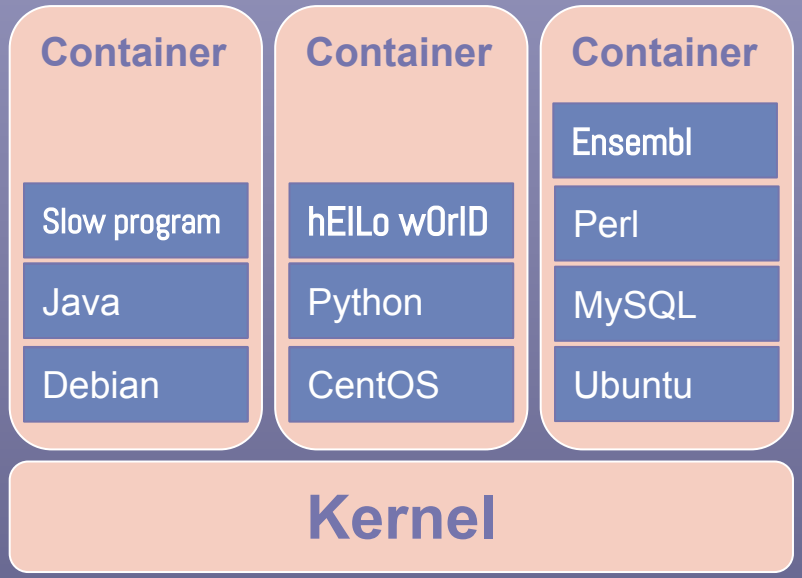
Megan Phinney
Iowa State University
BS Computer Engineering 2022

Anna Chernikov
NC State University
BS Computer Science 2020
University of Arizona
PhD Student

Los Alamos NATIONAL LABORATORY EST. 1943

Containers in HPC

- What are Containers?
  - Contains Application, Software Stack, and OS
  - Can be moved between different machines
- Why Containers in HPC?
  - Hides Complex Dependencies
  - Lightweight
  - Portable
  - Easy Deployment
  - Isolated Environment



What is Charliecloud?

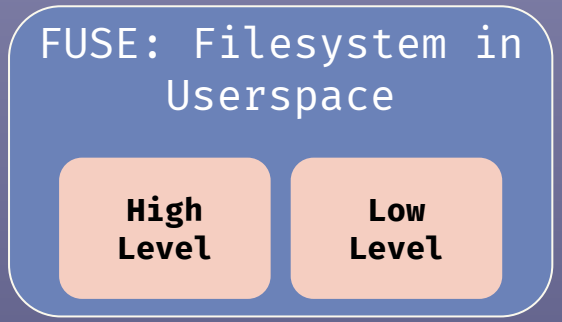
- A Container Runtime developed at LANL specifically for HPC

Why Charliecloud?

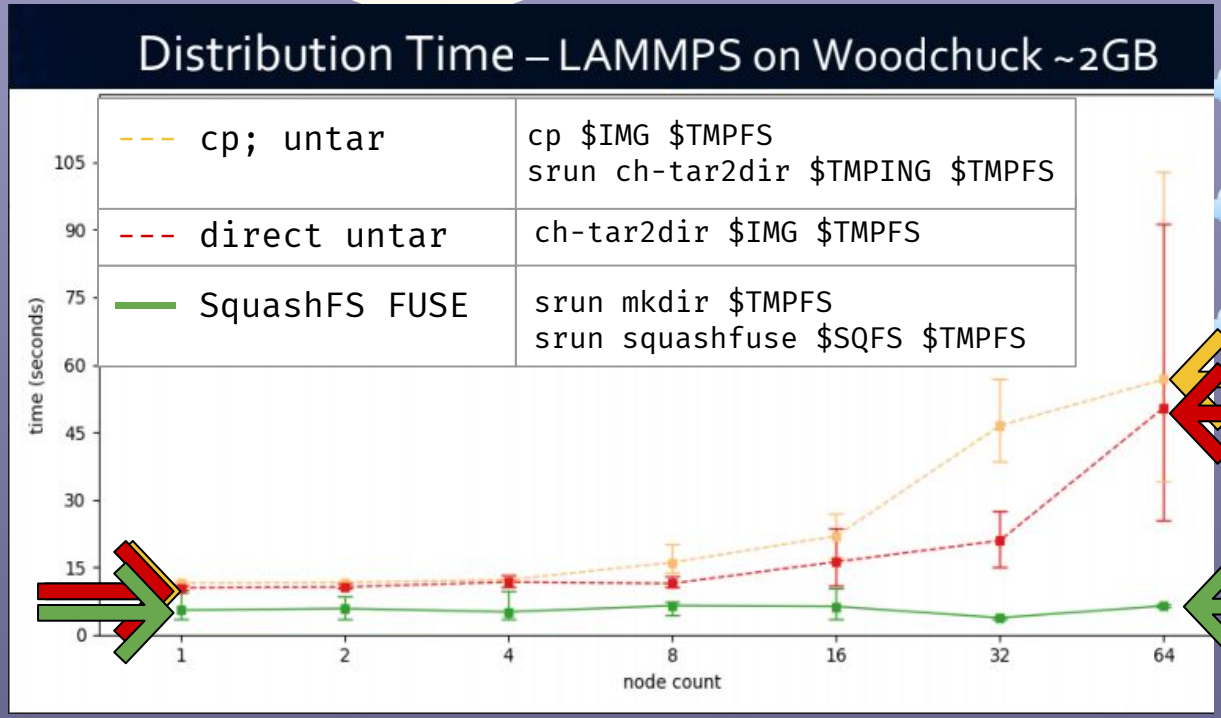
- Light-weight
- Fully Unprivileged
- Better choice for HPC

What is a SquashFS File?

- Compressed read-only filesystem
- Like tarball but mountable
- SquashFUSE enables mounting by unprivileged users



Why Squash?



Anaya, Cutshaw, Goff; "Evaluating Container Image Distribution Methods for HPC Using Charliecloud", Supercomputer Institute HPC Showcase 2018

- Faster image distribution times
- SquashFS scales better than tarballs
- Better choice for HPC

Charliecloud Commands

Tarball Workflow	`ch-tar2dir ~/img.tar.gz /var/tmp`	Unpack tarball
	`ch-run /var/tmp/img -- /bin/true`	Run Command
Old SquashFS Workflow	`ch-mount ~/img.sqfs /var/tmp`	Mount SquashFS
	`ch-run /var/tmp/img -- /bin/true`	Run Command
	`ch-umount /var/tmp/img`	Unmount SquashFS
New SquashFS Workflow	`ch-run ~/img.sqfs -- /bin/true`	Mount SquashFS, Run Command, Unmount SquashFS

Typical Tarball Workflow

	Unpack	Mount	Run	Unmount
Tarball Workflow	`ch-tar2dir`		`ch-run`	

Too Slow Distribution time

Takes up too much memory

Typical SquashFS Workflow

	Unpack	Mount	Run	Unmount
Old SquashFS Workflow		`ch-mount`	`ch-run`	`ch-umount`

No automatic clean up mechanism

Doesn't play well with srun (Slurm)

3 User Commands

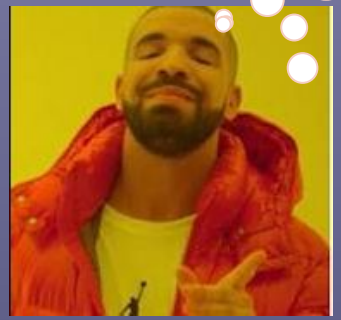
Our New SquashFS Workflow

	Unpack	Mount	Run	Unmount
New SquashFS Workflow			`ch-run`	

Automatically cleans up user mounts

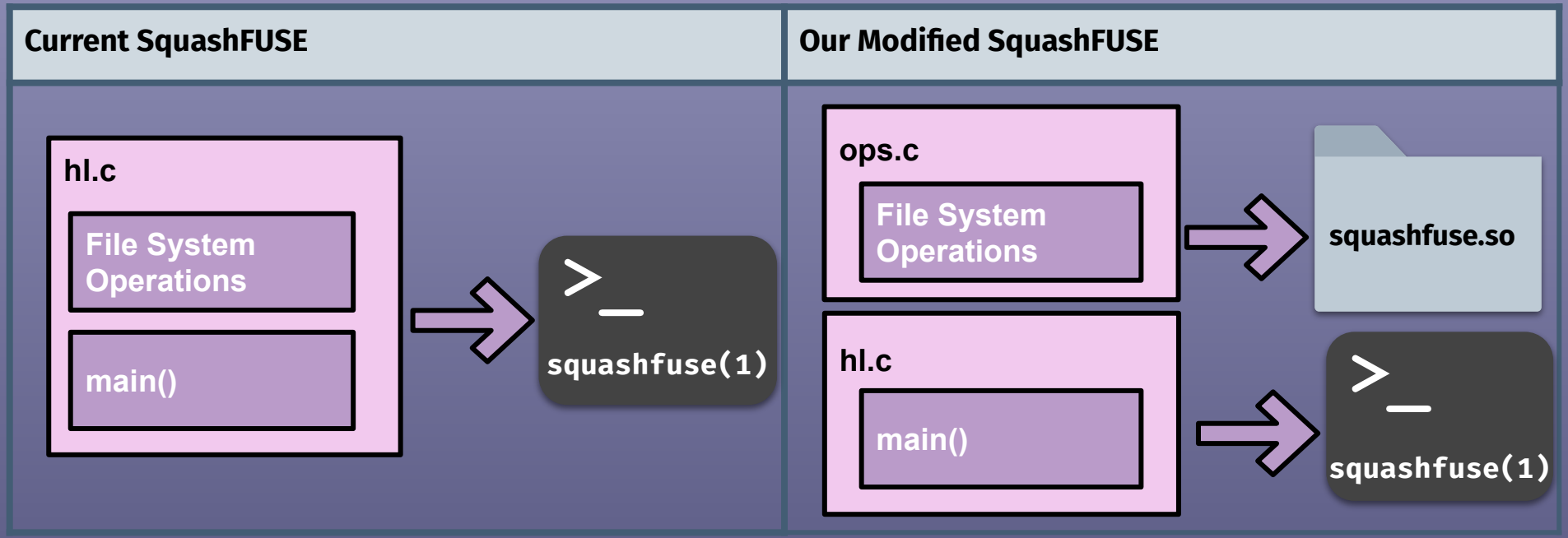
Only 1 User Command

Works with srun



Moved SquashFUSE File System Operations to Shared Library

SquashFUSE file system operations are made accessible to ch-run via our new shared library



Refactored SquashFUSE

6 Makefile.am

```
28  28    libsquashfuse_la_SOURCES = swap.c cache.c table.c dir.c file.c fs.c \
29  29          decompress.c xattr.c hash.c stack.c traverse.c util.c \
30      -       nonstd-pread.c nonstd-stat.c \
    30  +       nonstd-pread.c nonstd-stat.c ops.c \
31  31          squashfs_fs.h common.h nonstd-internal.h nonstd.h swap.h cache.h table.h \
32  32          dir.h file.h decompress.h xattr.h squashfuse.h hash.h stack.h traverse.h \
```

278 hl.c

```
44  - static sqfs_err sqfs_hl_lookup(sqfs **fs, sqfs_inode *inode,
45  -                const char *path) {
46  -        bool found;
47  -
48  -        sqfs_hl *hl = fuse_get_context()->private_data;
```

282 ops.c

```
    17  + static sqfs_err sqfs_hl_lookup(sqfs **fs, sqfs_inode *inode,
    18  +                const char *path) {
    19  +        bool found;
    20  +
    21  +        sqfs_hl *hl = fuse_get_context()->private_data;
```

23 ops.h

```
    11  + #ifndef SQFS_OPS_H
    12  + #define SQFS_OPS_H
    13  + typedef struct fuse_operations fuse_operations;
    14  + typedef struct sqfs_hl sqfs_hl;
```

hpc / squashfuse

forked from vasi/squashfuse

6 Makefile.am

278 hl.c

282 ops.c

23 ops.h

Linked SquashFUSE Libraries to ch-run

All Fuse File System operations in ch-run are referenced from SquashFUSE libraries:

- Mount
- Unmount
- Reads

Updated ch-run source code and documentation

hpc / charliecloud

52 bin/ch-run.c

```
+    if(sqfs_hl_open(argv[arg_next],0))
+        goSquash(sq.parentdir, &argv[arg_next]);
+
```

92 bin/ch_core.c

```
+ int squashmount(struct squash *s)
+ {
```

13 bin/ch_core.h

```
+ struct squash {
+    char *filepath;        // path of sqfs file
+    char *mountdir;        //location where squashfs is mounted
+    pid_t pid;             // process id of the fuse loop
+    struct fuse_chan *ch;  //fuse channel associated with squash fuse session
+    struct fuse *fuse;     //fuse struct associated with squash fuse session
+    char *parentdir;     //location of mountpoint parent directory
+ };

+ extern struct squash *s;
  /** Function prototypes **/

  void containerize(struct container *c);
  void run_user_command(char *argv[], const char *initial_dir);
+ int squashmount(struct squash *s);
+ void kill_fuse_loop();
```

Squash FUSE auto-mount option for `ch-run`

Description

By de
auton
squas
will b

The S

1.
2. I
3. T
4.
5. V
6. T
7. A
8. A

Using Squash FileSystems

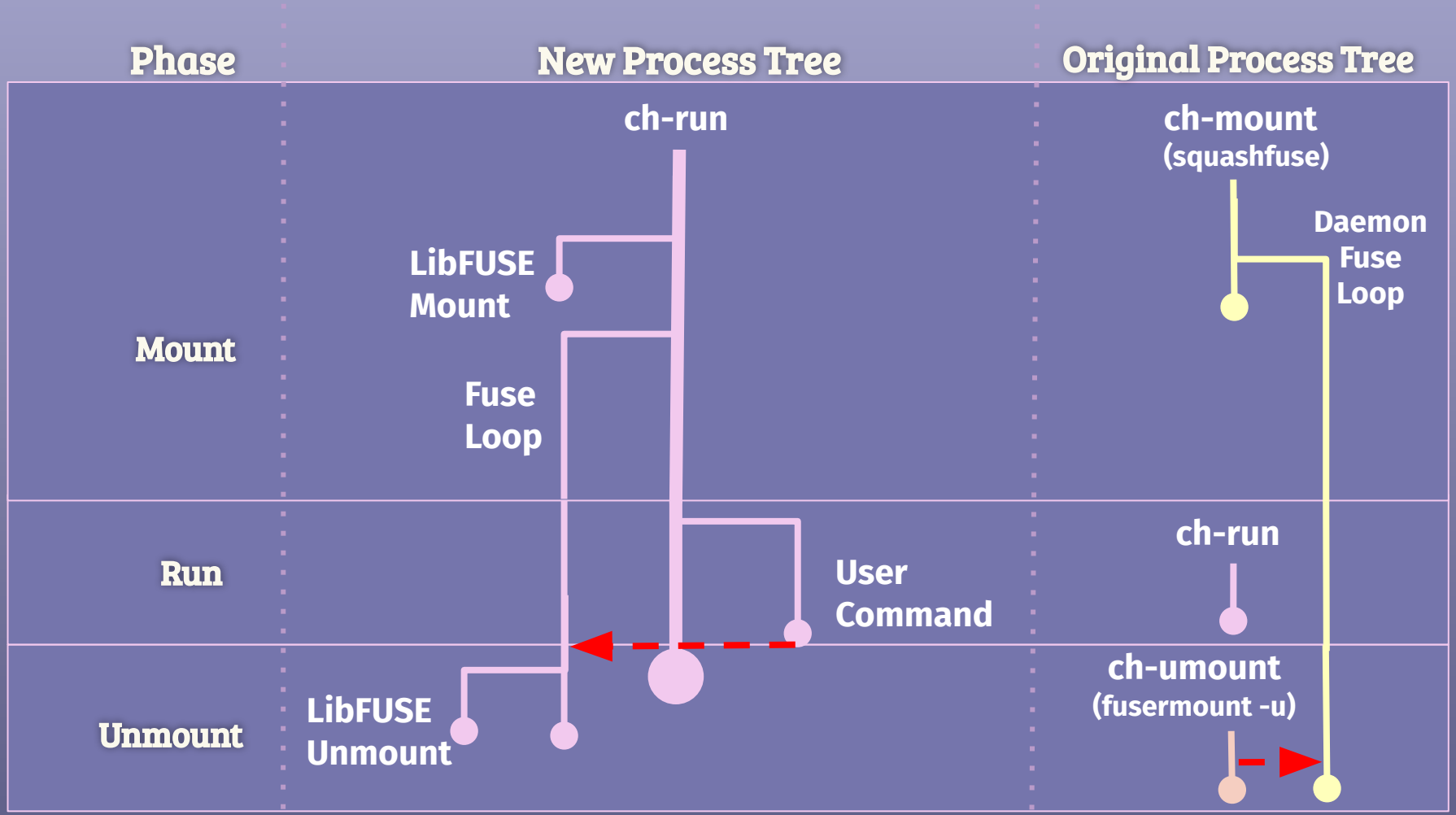
ch-run will handle Squash Filesystems passed in as the `IMAGE`. They will be automatically mounted prior to execution, and unmounted as part of the cleanup. the `--squashmnt` option allows you to specify the parent directory at which the squash filesystem will be mounted.

Example 1: Create and Run a SquashFilesystem image:

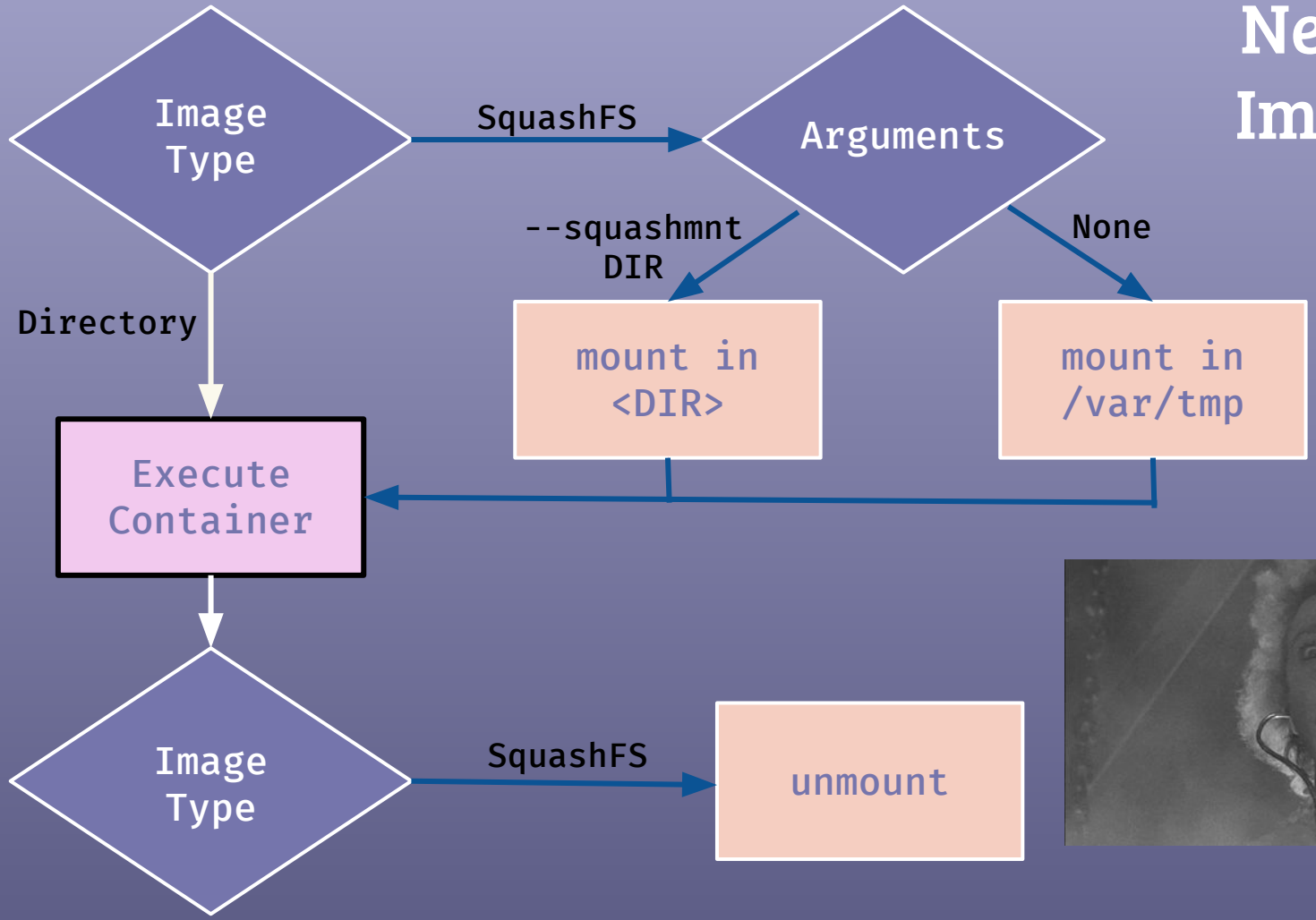
```
$ ch-build -t hello $HOME/chorkshop/hello
$ ch-builder2squash hello $HOME/images/
$ ch-run $HOME/images/hello.sqfs -- ./hello.py
```

Example 2: Create and Run a Squash Filesystem image but with preferred mount directory:

```
$ ch-build -t hello $HOME/chorkshop/hello
$ ch-builder2squash hello $HOME/images/
$ ch-run --squashmnt=/tmp/mytmp/ $HOME/images/hello.sqfs -- ./hello.py
```


Phase
New Process Tree
Original Process Tree
Mount
Run
Unmount
ch-run
LibFUSE
Mount
Fuse
Loop
User
Command
LibFUSE
Unmount
ch-mount
(squashfuse)
Daemon
Fuse
Loop
ch-run
ch-umount
(fusermount -u)

New ch-run Image Logic



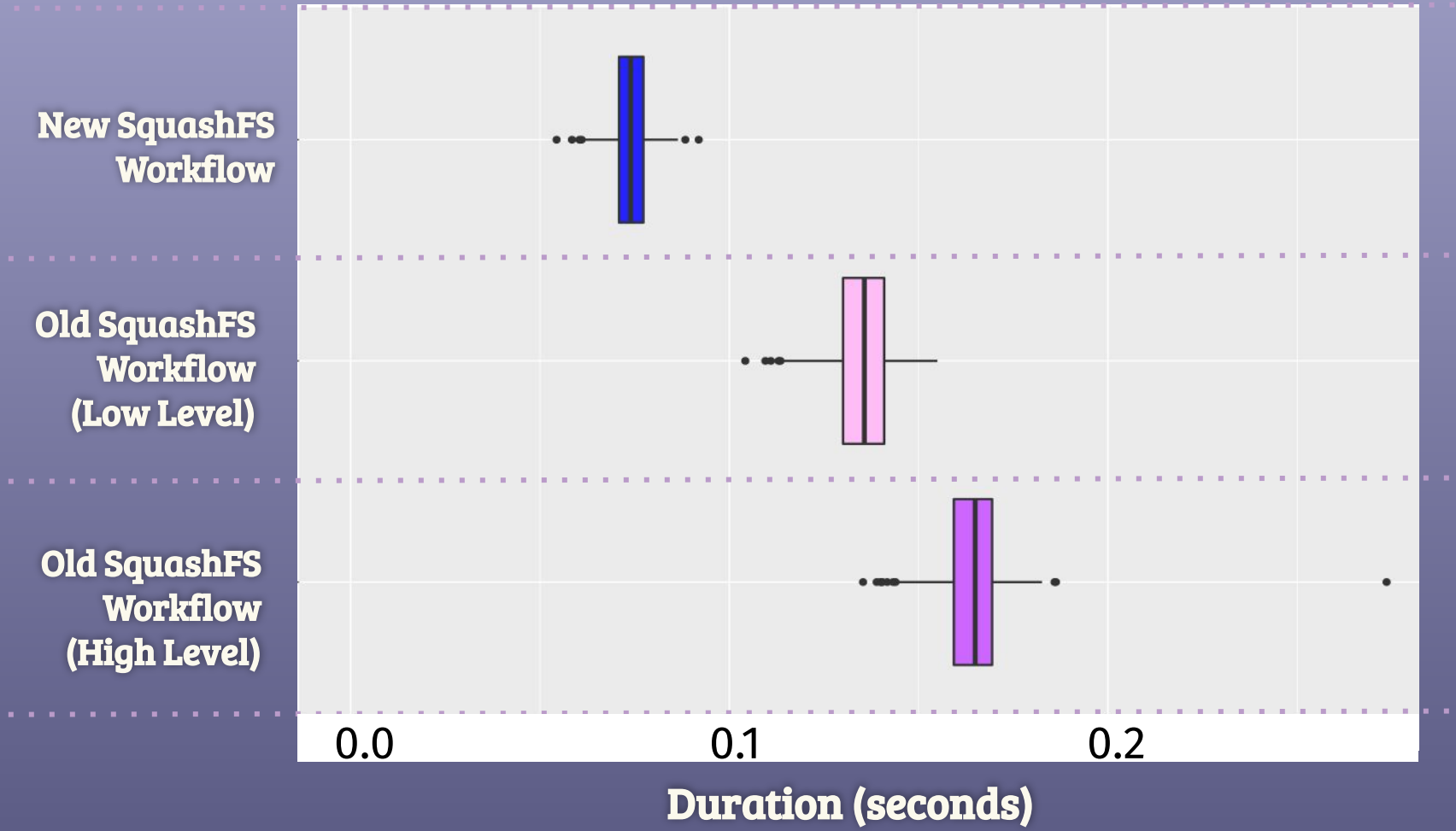
How does the startup and teardown time of the new workflow compare to the old workflow?

Experiment Procedure

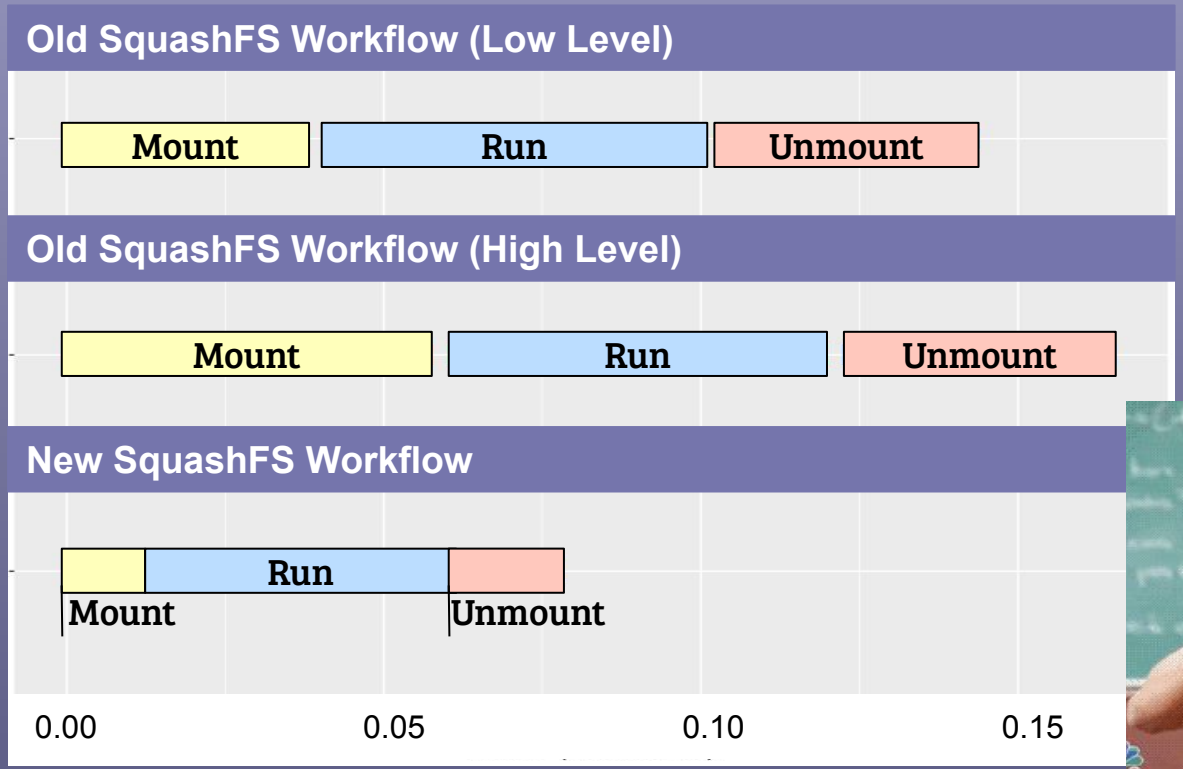
- Test Program: /bin/true
- Test Groups
  - Old Workflow (Low Level Fuse API)
  - Old Workflow (High Level Fuse API)
  - New Workflow (High Level Fuse API)
- Surround each step of the workflow with date
  Calculate total durations
- Repeat ×1000

```
date '+%s.%N'
$CHMOUNTLL $SQFS /var/tmp
date '+%s.%N'
date '+%s.%N'
$CHRUN /var/tmp/$NAME -- $PROG
date '+%s.%N'
date '+%s.%N'
$CHUMOUNT /var/tmp/$NAME
(date '+%s.%N'
```

Total Workflow Time



Median Timeline for SquashFS Workflows



New SquashFS Workflow:

- Reduces Mount and Unmount Time
- Container Execution Time is constant

Possible Causes for Shorter Mount Times

Mounting with Old SquashFS Workflow

Mounting with New SquashFS Workflow

Shell

C Program

SquashFUSE

Daemon

Child Process

How does running an image multiple times scale?

Old SquashFS Workflow

```
ch-mount ~/img.sqfs /var/tmp

for i in $N

do

    ch-run /var/tmp/img -- /bin/true

done

ch-umount /var/tmp/img
```

Mount Once

Unmount Once

New SquashFS Workflow

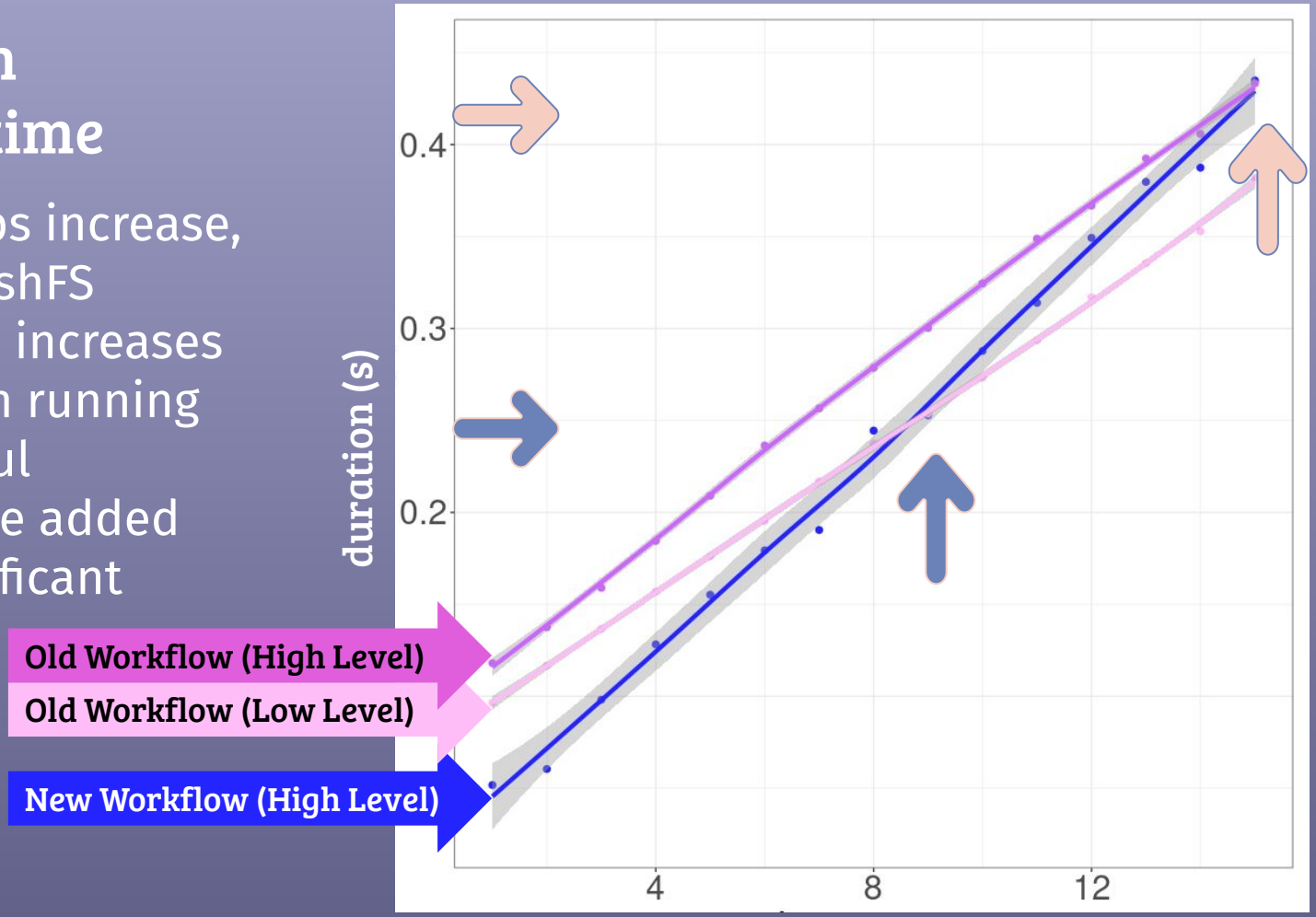
```
for i in $N
do
    ch-run ~/img.sqfs -- /bin/true
done
```

Mount N Times

Unmount N Times

Multiple ch-run Execution Runtime

- As ch-run steps increase, the New SquashFS workflow time increases
- However when running any meaningful application the added time is insignificant



Conclusions

Our New SquashFS Workflow:

- More user-friendly
- No additional performance cost
- Plays well with srun
- Auto-cleans SquashFS mounts

Future Work

- Merge branch into Charliecloud
- Add low-level FUSE API
- Test the new workflow with larger SquashFS images
- Submit pull-request for SquashFUSE

Our Team

Megan Phinney

Iowa State University
Computer Engineering
mphinney@iastate.edu

Anna Chernikov

NC State , University of Arizona
Computer Science
chernikov@email.arizona.edu

Jordan Ogas

Creator of R. Peezee

Alfred Torrez

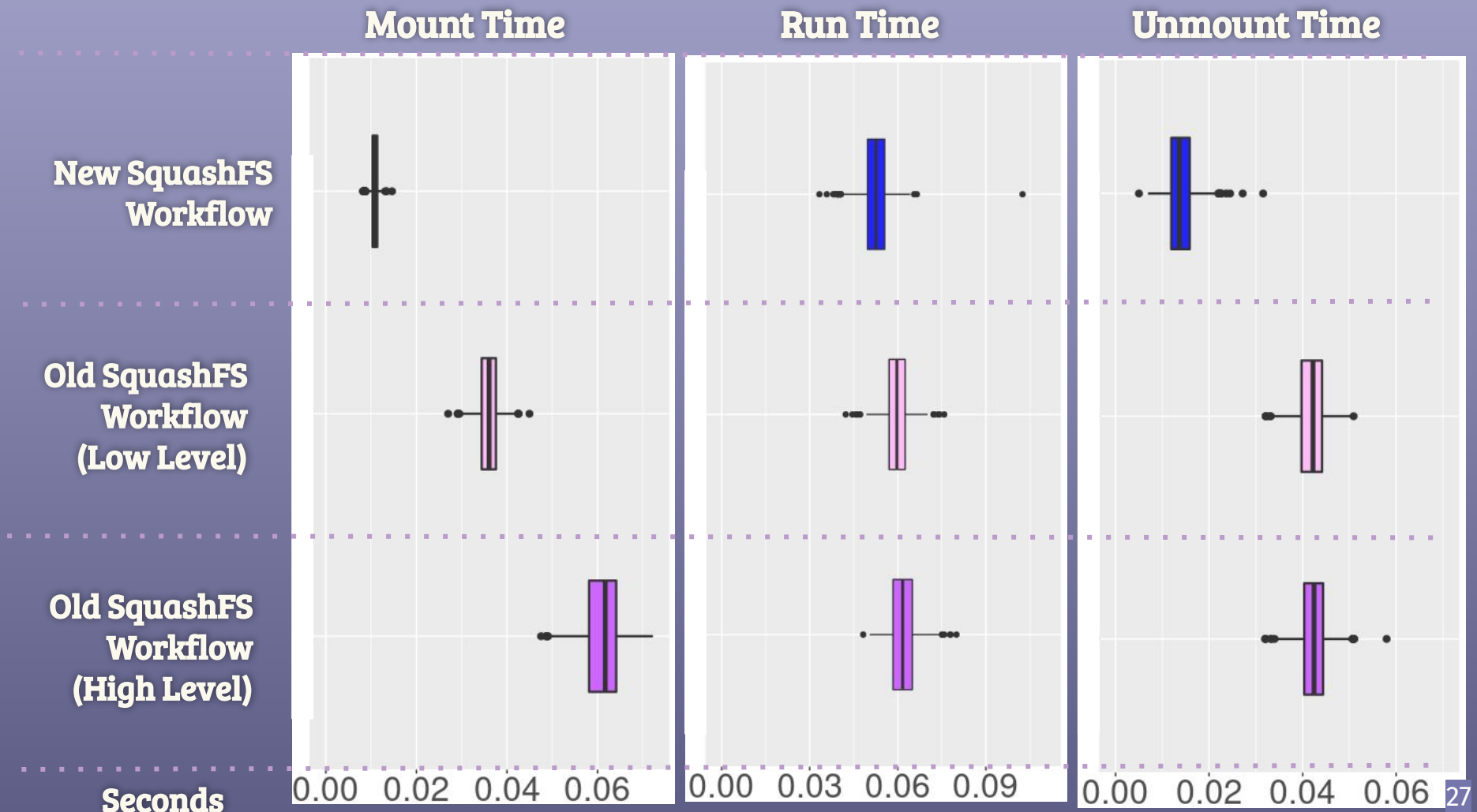
Grill Dad

Shane Goff

Brofessional

Reid Priedhorsky

King of Charliecloud


Mount Time
Run Time
Unmount Time
New SquashFS Workflow
Old SquashFS Workflow (Low Level)
Old SquashFS Workflow (High Level)
Seconds
0.00
0.02
0.04
0.06
0.00
0.03
0.06
0.09
0.00
0.02
0.04
0.06