File System Aging

Featuring slides modified from a talk by

Martín Farach-Colton
Rutgers University

Aging

- Two papers
  - Smith and Seltzer
  - Conway et al.
- How do people feel about the readings?

Aging

- Two papers
  - Smith and Seltzer
  - Conway et al.
- How do people feel about the readings?

Outline

- (Brief) I/O Models overview
- Definitions of Fragmentation
- Aging Problem
- Simulation and measurement
- Discussion

How do we model performance?

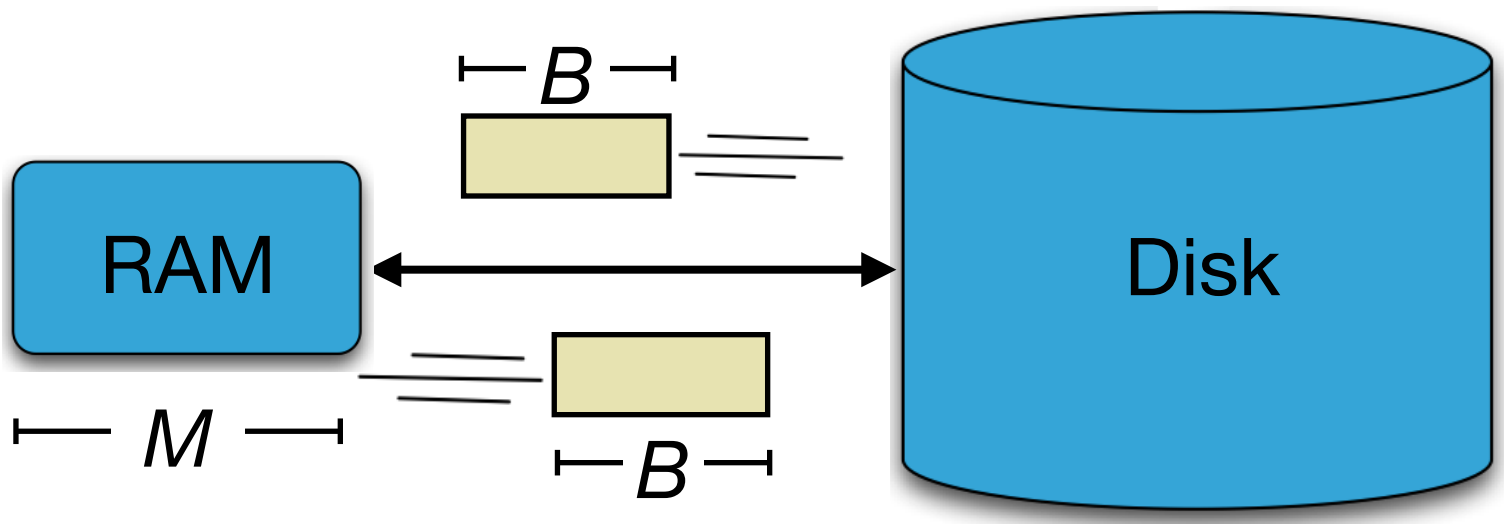
How do we account for disk I/O?

DAM model: How theorists think about external memory algorithms

- Data is transferred in blocks between RAM and disk.
- The number of block transfers dominates the running time.

Goal: Minimize # of I/Os

- Performance bounds are parameterized by block size ***B***, memory size ***M***, data size ***N***.



[Aggarwal+Vitter '88]

Is the DAM Model any good?

Short answer: Yes (2-competitive)

Long answer: No (can’t tune parameters)

Affine Model

Affine model:

- Data is transferred in blocks between RAM and disk.
- If ***k*** blocks are transferred, the cost is

$$1 + \alpha k$$

- On hard disks, 1 is the normalized seek cost and α is the incremental bandwidth cost of subsequent blocks
- On SSDs, it's more complicated but affine still fits better than DAM costs.
  - (And PDAM fits even better…)

Goal: Minimize cost of I/Os

- Performance bounds are parameterized by block size ***B***, memory size ***M***, data size ***N***.

Takeaway: the affine model captures the size of I/Os as well as the speed of the device itself.

Now We Have a Model, What Next?

The goal of our model is to predict performance.

We can verify "things" using a benchmark

- We compare two systems, A and B, by running the same well-specified workload on each system
- We use our model to predict the relative performance of A and B, and either:
  - Validate our hypothesis
  - Revise our model
  - Revise our theory because we learned something new about our system and are better able to present an input to our model

To be useful, we need to run representative benchmarks under representative conditions

Representative State

What is the representative state of a file system?

- How many files?
- What is the organization of the files (directory hierarchy)?
- What is the average size of a file? File size distribution?

Is the state of a file system a path or a point?

- It is a path.
  - Creating files limits/influences the placement decisions for future operations
  - Deleting files creates “holes” in the LBA space
  - Moving (renaming) files alters the relationships between files
- It isn’t enough to look at the contents of a file system in isolation, we need to know where we started and how we got there.

Theory: many file systems will age.

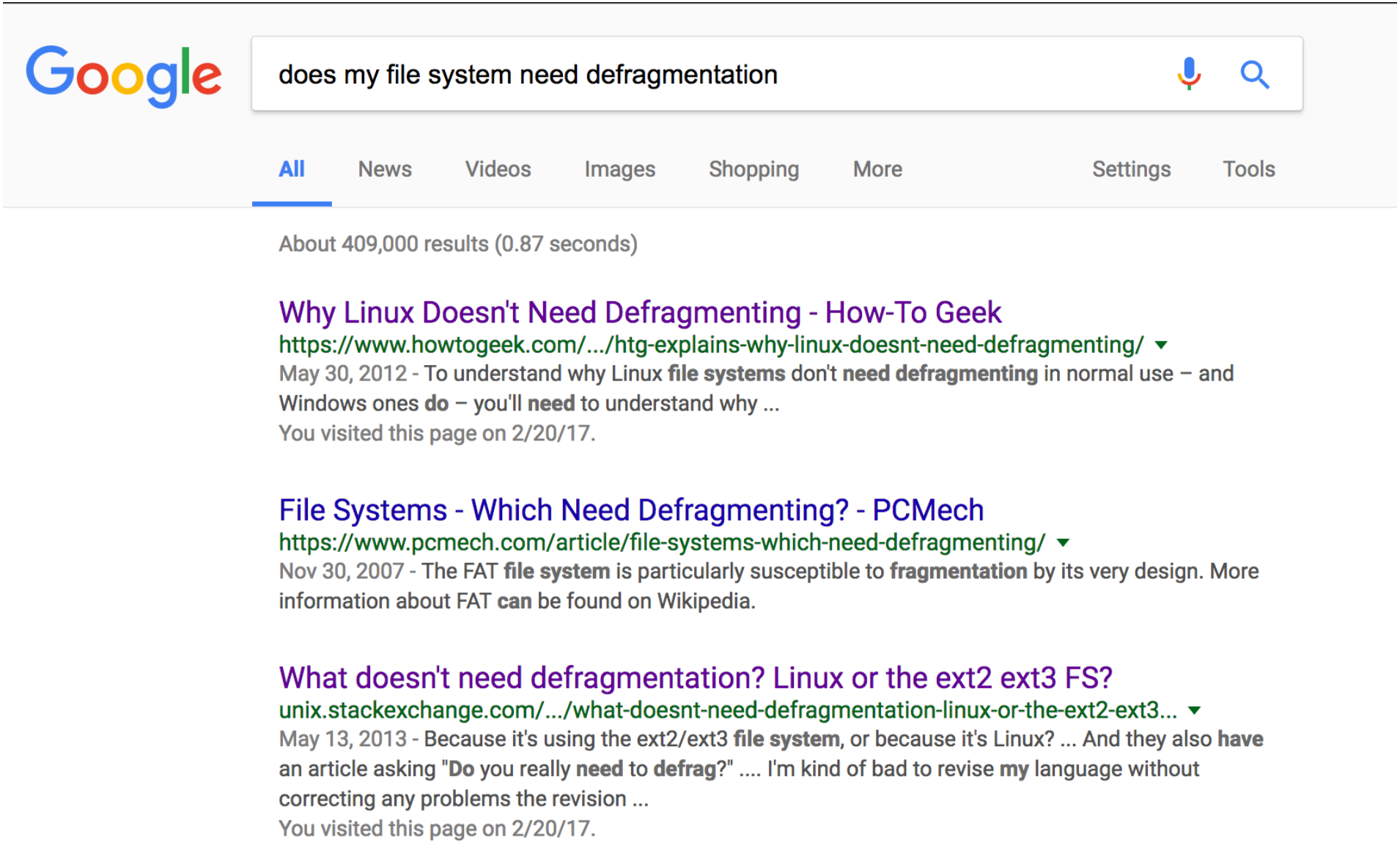
- Aging: the degradation of performance over time.
  - Our models predict this
    - heuristics lead to fragmentation
    - fragmentation leads to increased seeks on important workloads

Two open questions:

- Is the representative state an aged file system?
- If so, how do we create a representatively aged file system?

Does aging happen on modern file systems?

Do file system age?



Do file system age?

I'm Feeling Lucky

Chris Hoffman at howtogeek.com says:

"Linux's ext2, ext3, and ext4 file systems… [are] designed to avoid fragmentation in normal use."

"If you do have problems with fragmentation on Linux, you probably need a larger hard disk."

Do file system age?

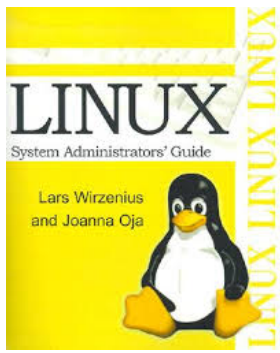
I'm Feeling Lucky

Chris Hoffman at howtogeek.com says:

"Linux's ext2, ext3, and ext4 file systems… [are] designed to avoid fragmentation in normal use."

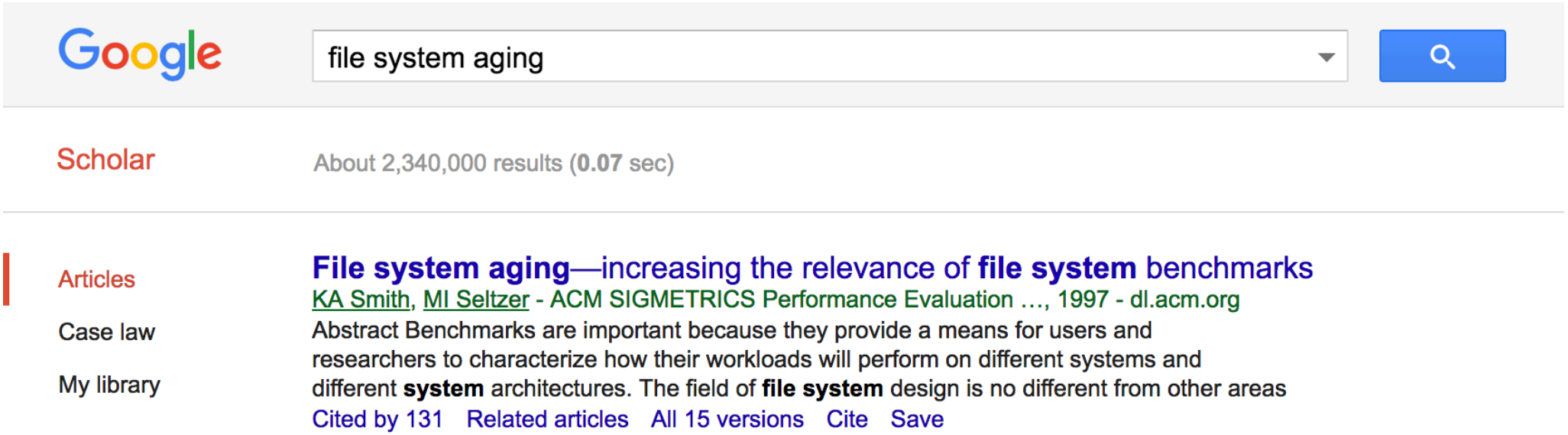
"If you do have problems with fragmentation on Linux, you probably need a larger hard disk."

"Modern Linux filesystems keep fragmentation at a minimum…Therefore it is not necessary to worry about fragmentation in a Linux system."



I guess not. Then was it ever a problem?

Do file system age?



So: as of 1997, file systems aged.

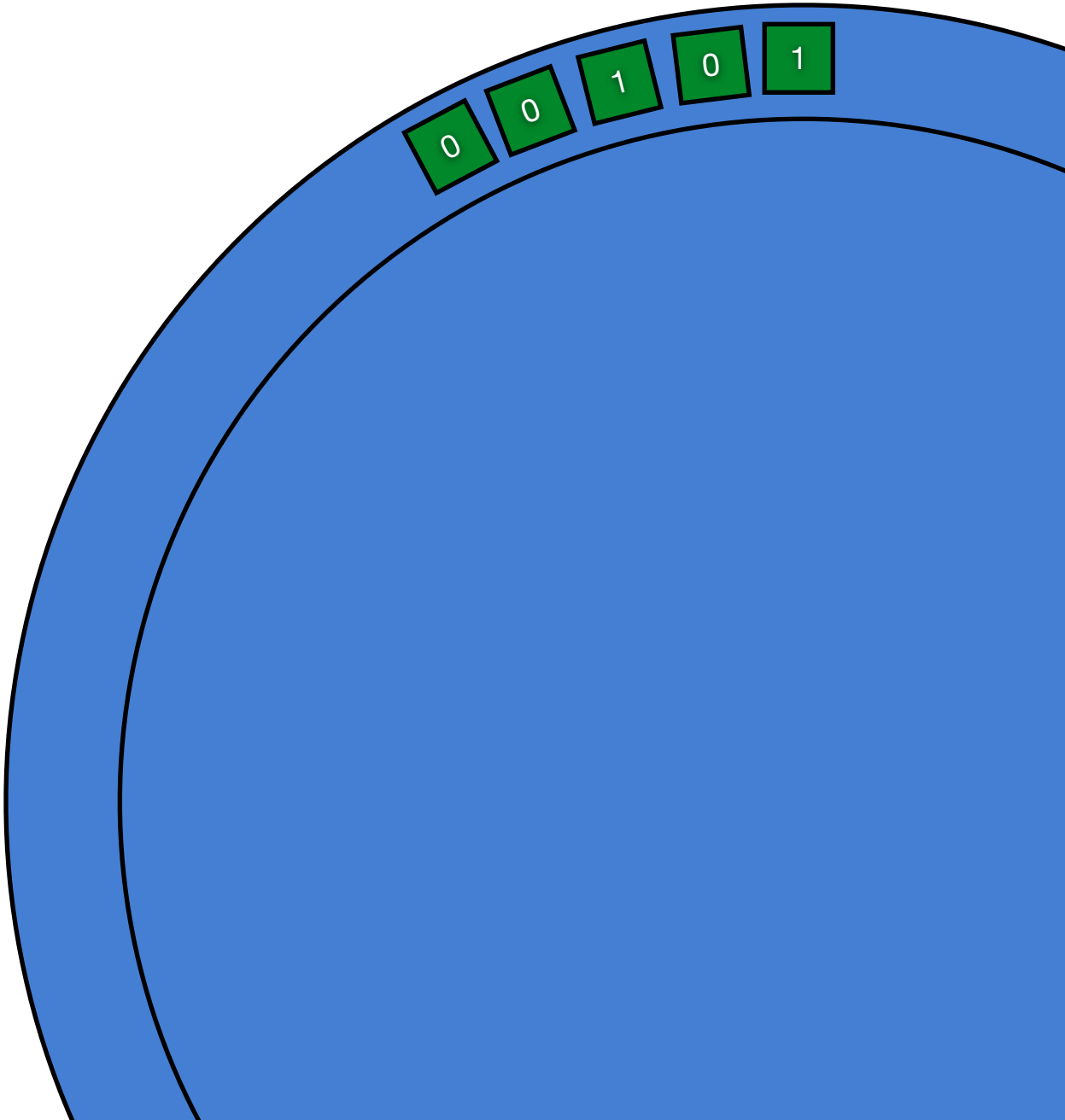
Then file systems got better, and sys admins say they don't age.

What's the actual story?

Theory of Aging over the Ages

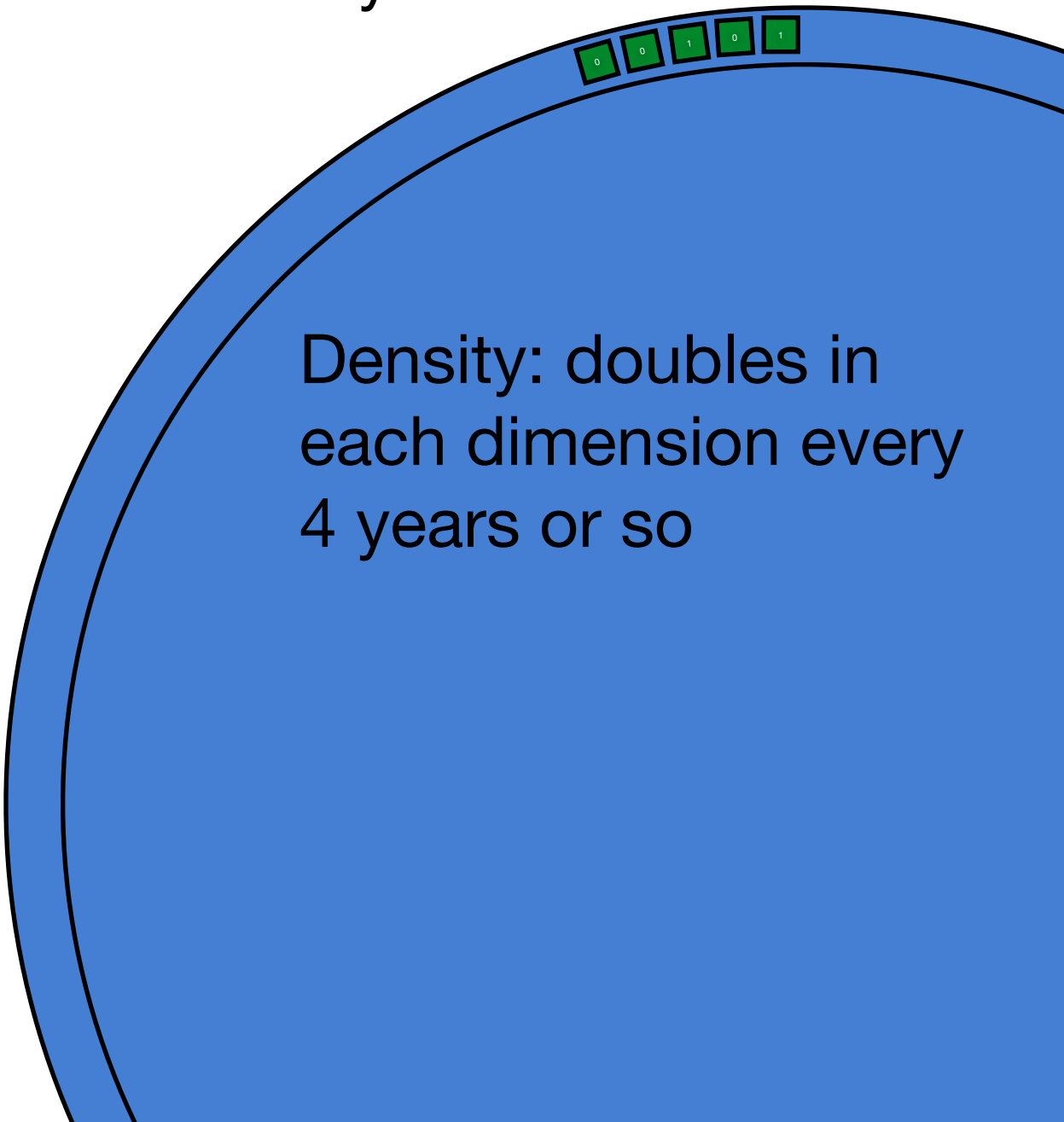
Euclid's view of hard disks

Year: X



Euclid's view of hard disks

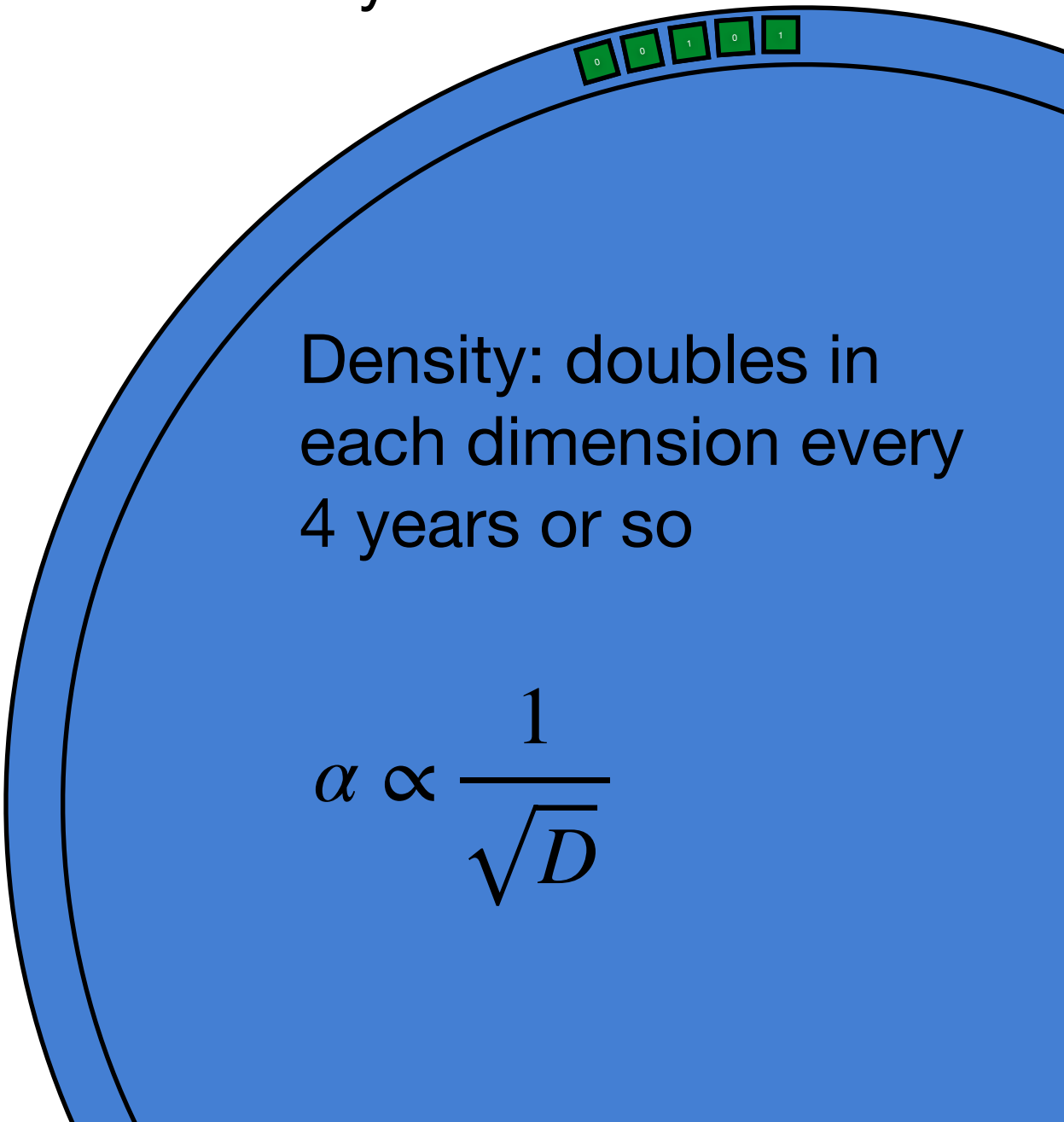
Year: X+~4 years



Density: doubles in each dimension every 4 years or so

Euclid's view of hard disks

Year: X+~4 years



Density: doubles in each dimension every 4 years or so

$$\alpha \propto \frac{1}{\sqrt{D}}$$

Hard disks gradually increase α

Measurements one decade have a sell-by date
… unless you solve the problem algorithmically

Assumption

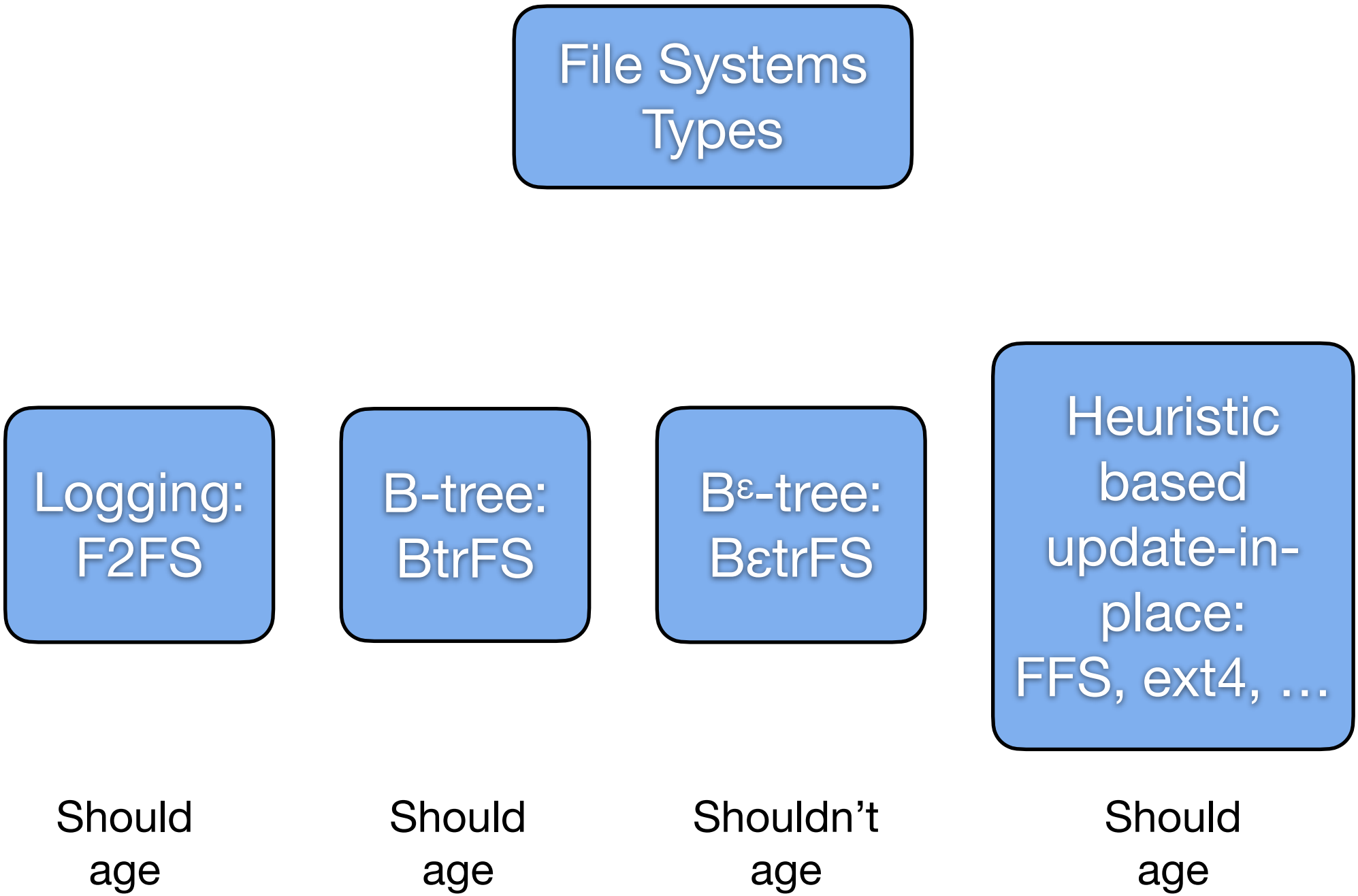
- Random seek is 100x slower than sequential
- 1% of blocks are non-sequential in the file system

Conclusion

- That's enough to limit IO to 50%

So, for people who think that file systems don't age, are you sure that modern file systems keep fragmentation to under 1%?

Which File Systems Age?



Let’s test the hypothesis!
How?

Keith Smith started grad school in ’92

- He decided to take snapshots of a bunch of computers
- Every day
- For years

He and Seltzer found that:

- If you replay the changes implied by the snapshots
- File system performance degrades
- On file systems available in ’97

We are impatient

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

We are impatient

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

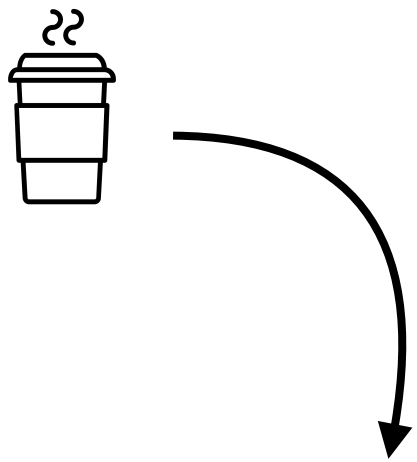
Let's model a very simple case: Developers

```
get coffee
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers



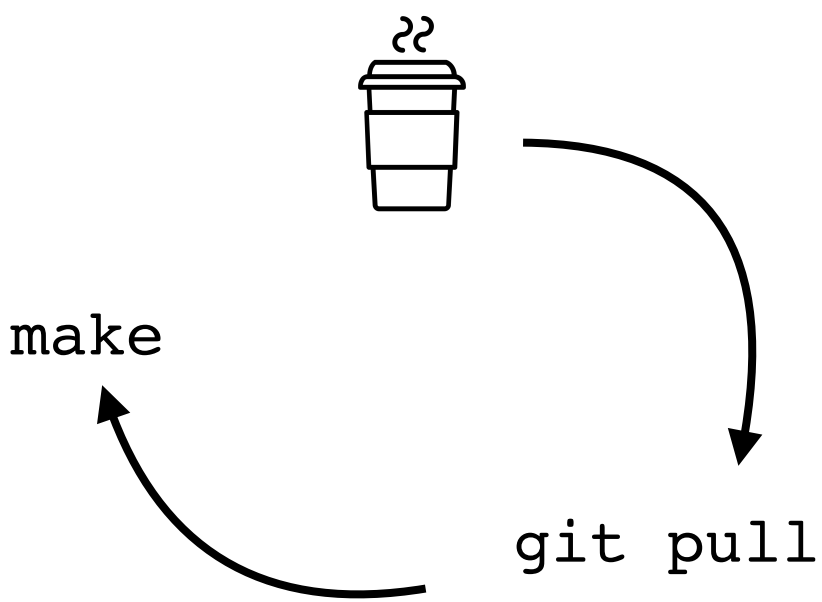
```
get coffee
git pull
```

```
git pull
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

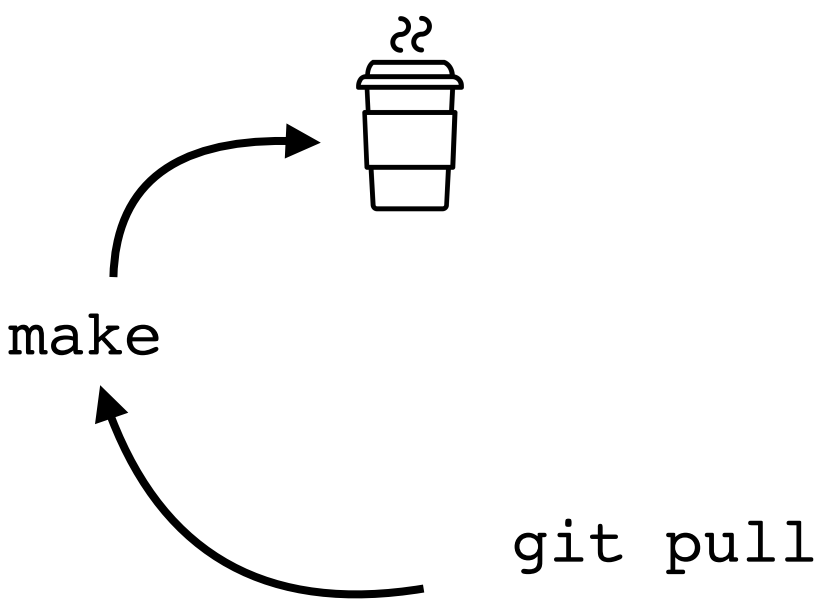



```
get coffee
git pull
make
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

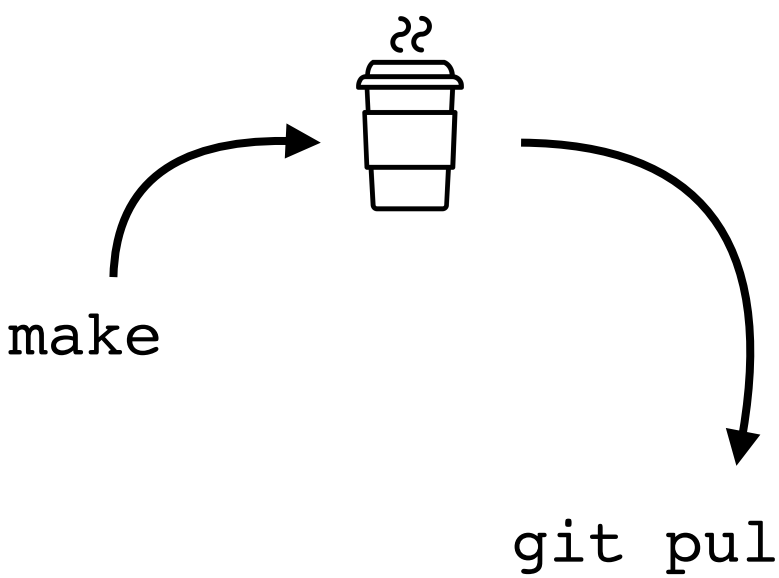



```
get coffee
git pull
make
get coffee
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

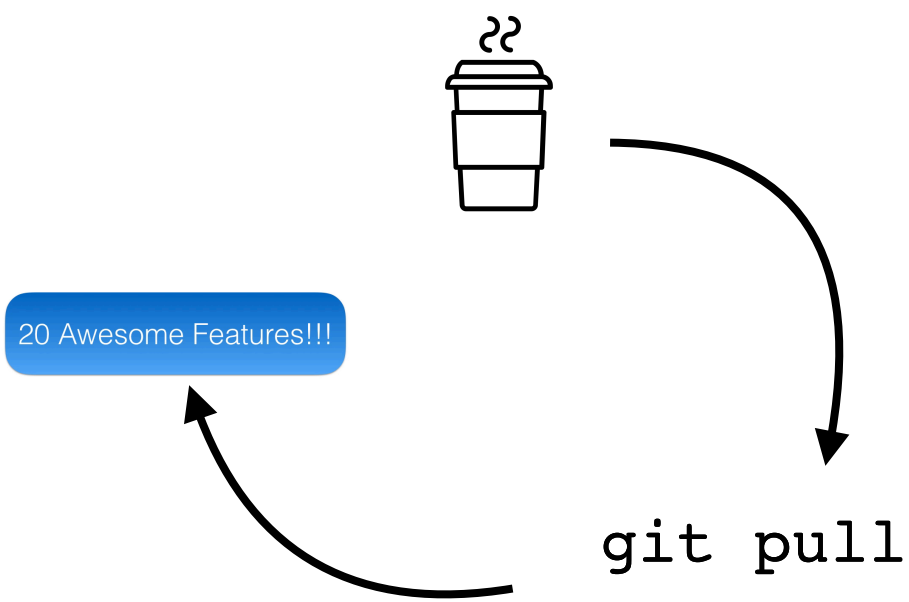



```
get coffee
git pull
make
get coffee
git pull
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

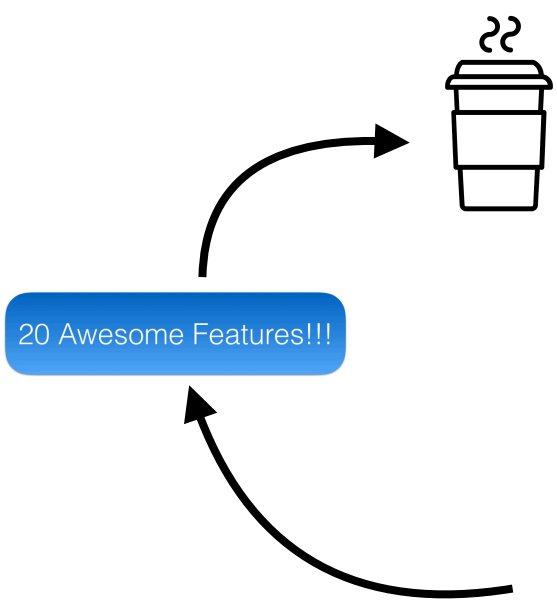



```
get coffee
git pull
make
get coffee
git pull
add awesome features
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

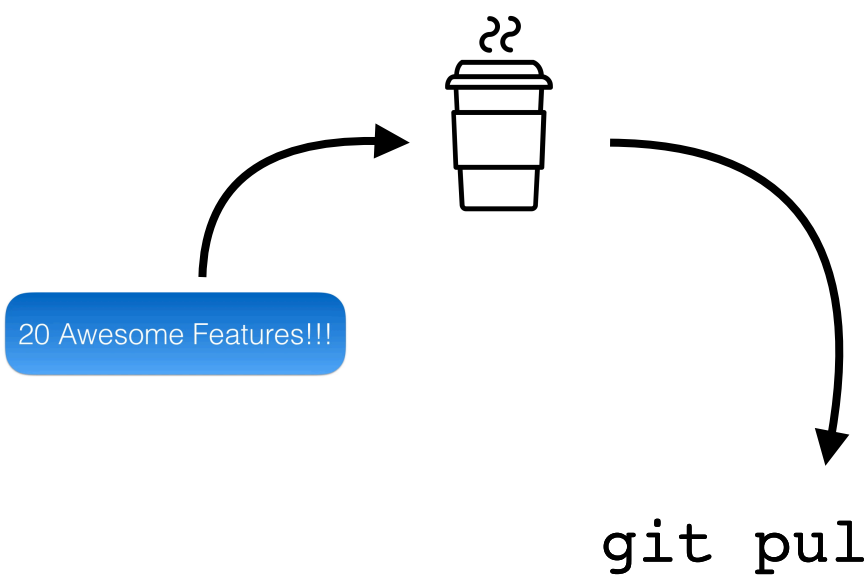



```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

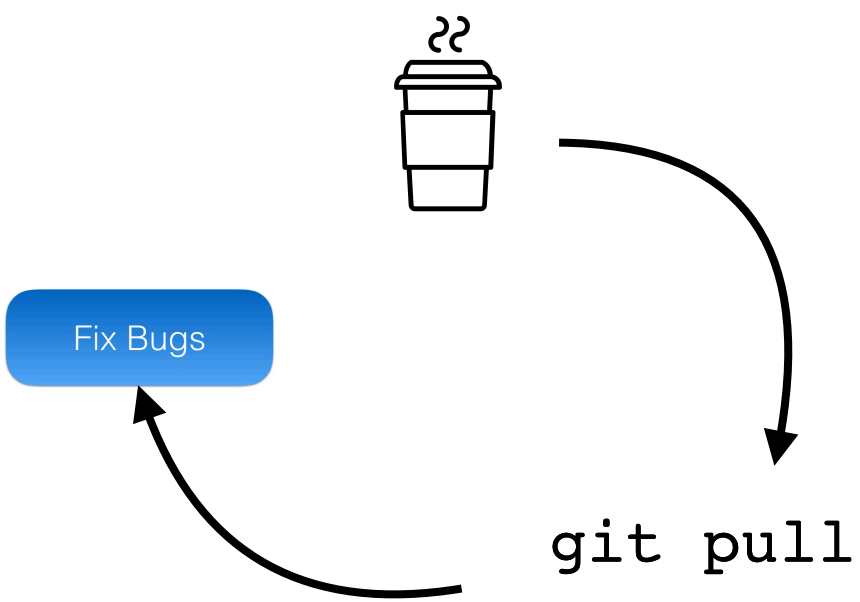



```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
git pull
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

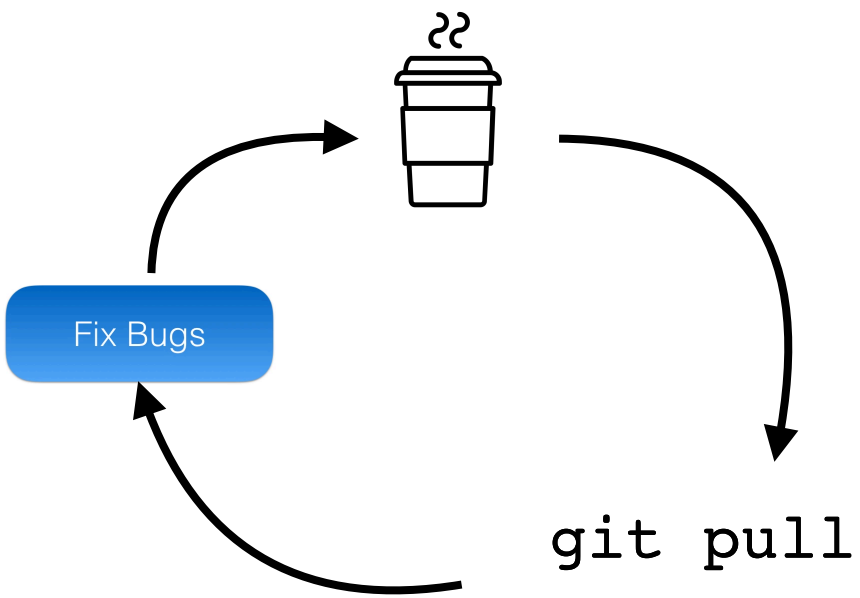



```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
git pull
fix bugs
. . .
```

We'd like a history of file systems changes

- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers

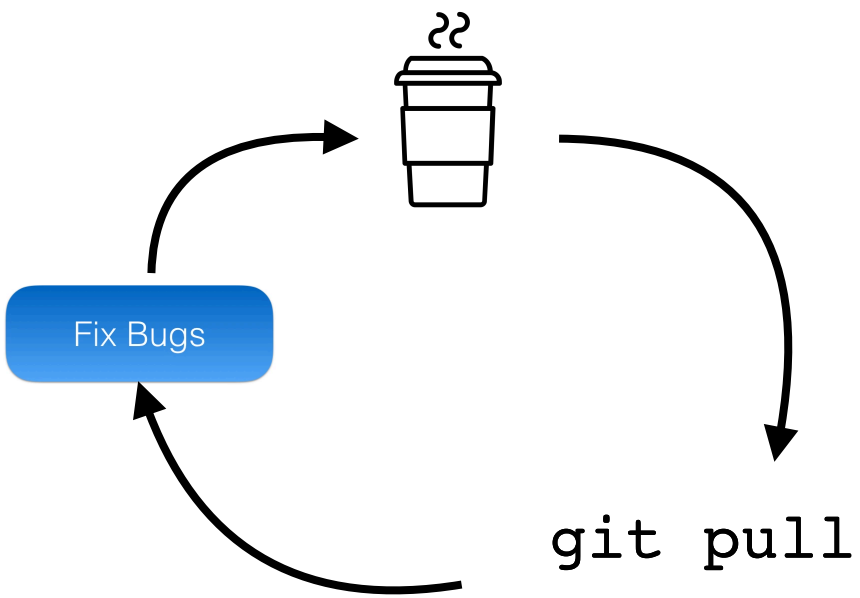



```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
git pull
fix bugs
. . .
```

We'd like a history of file systems changes

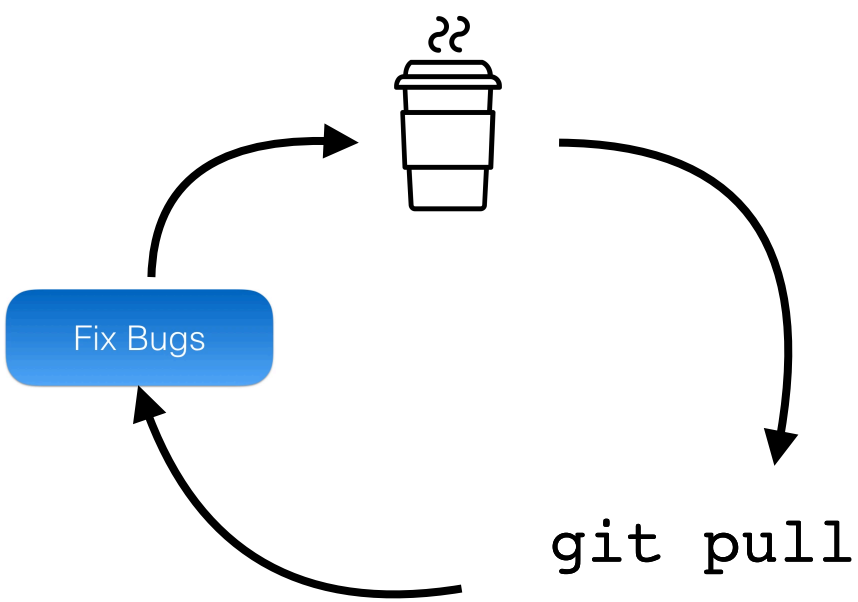
- That we can replay on any system
- We don't have to wait for years
- Years of history should be readily available

Let's model a very simple case: Developers




```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
git pull
fix bugs
. . .
```

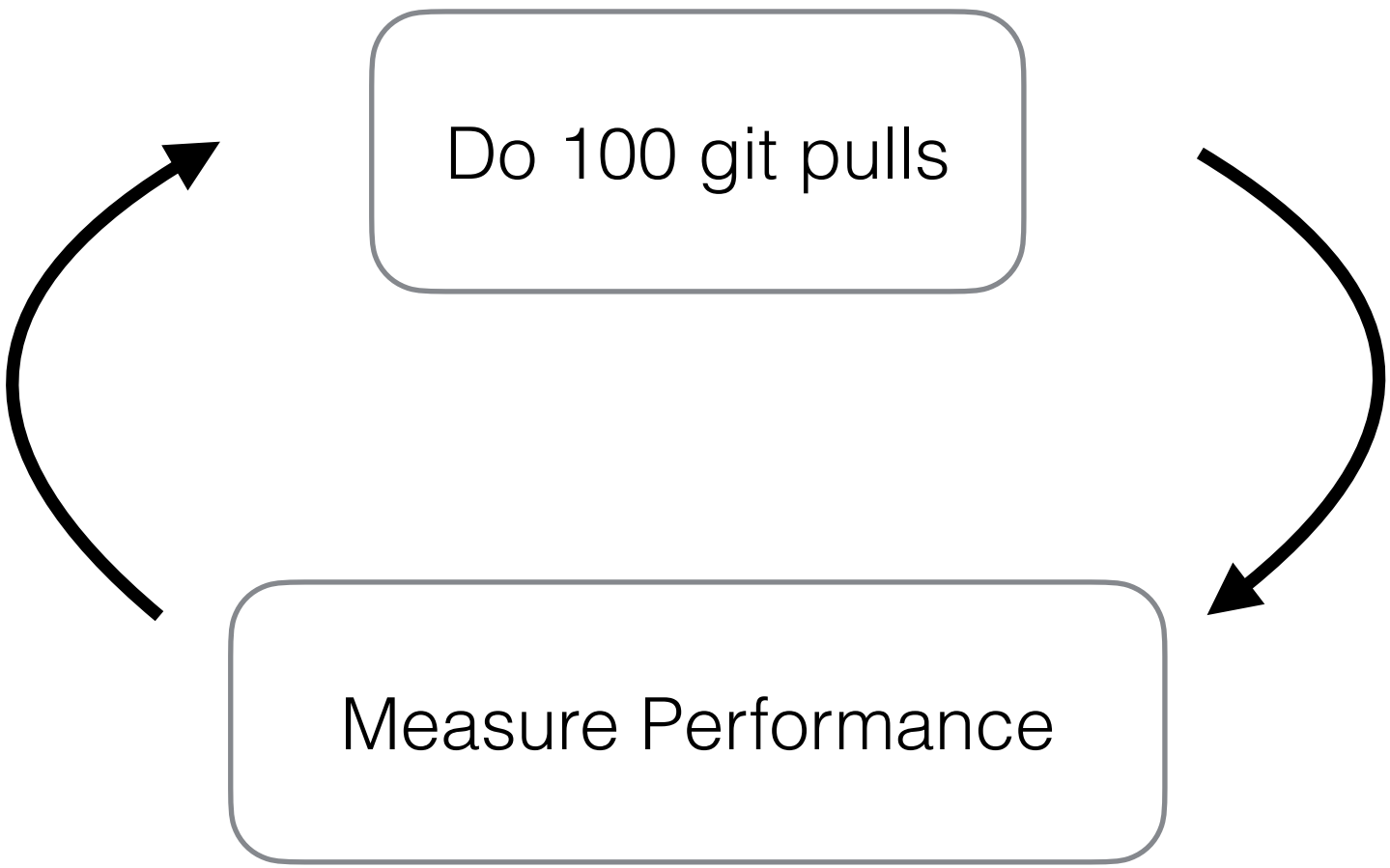
We can simulate a developer by replaying Git histories




```
get coffee
git pull
make
get coffee
git pull
add awesome features
get coffee
git pull
fix bugs
. . .
```

Simulating a Developer

Use the Linux kernel repo from github.com



How do we measure fragmentation?

```
time grep -r random_string /path/to/fs
```

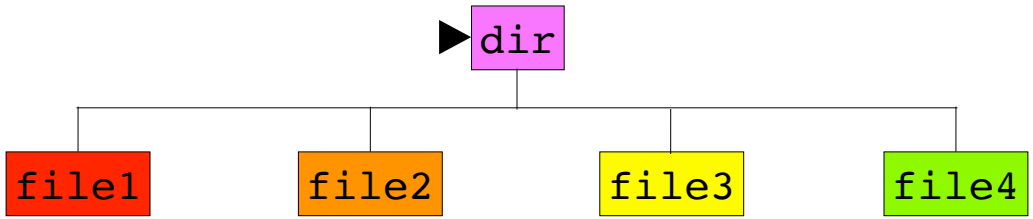
Like timing a preorder traversal of tree…

Should measure fragmentation

- Why?

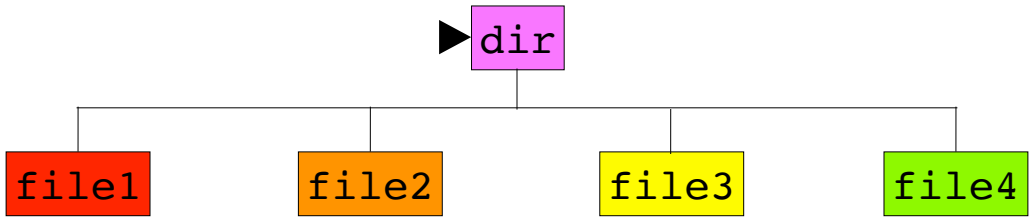
Measuring Aging

```
time grep -r random_string /path/to/filesystem
```



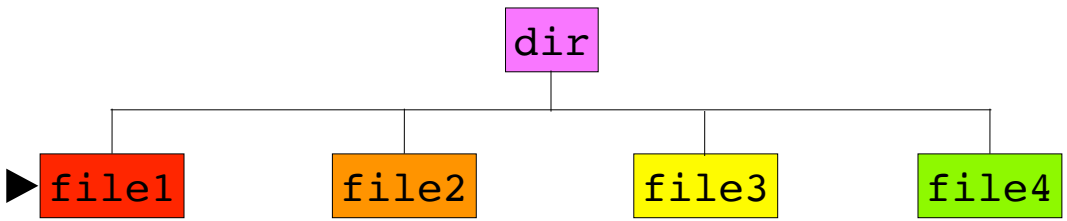
Measuring Aging

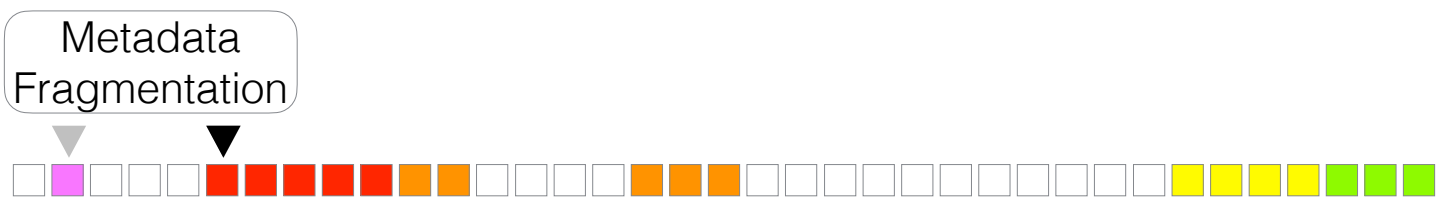
```
time grep -r random_string /path/to/filesystem
```



Measuring Aging

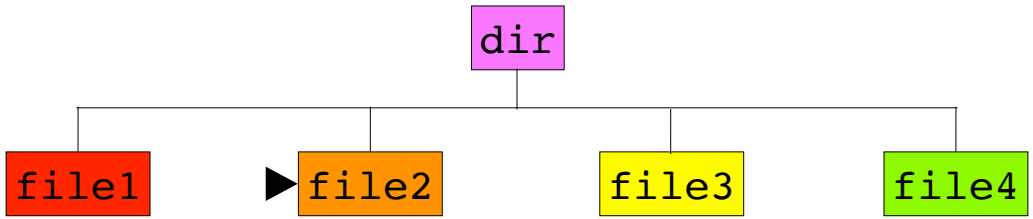
```
time grep -r random_string /path/to/filesystem
```

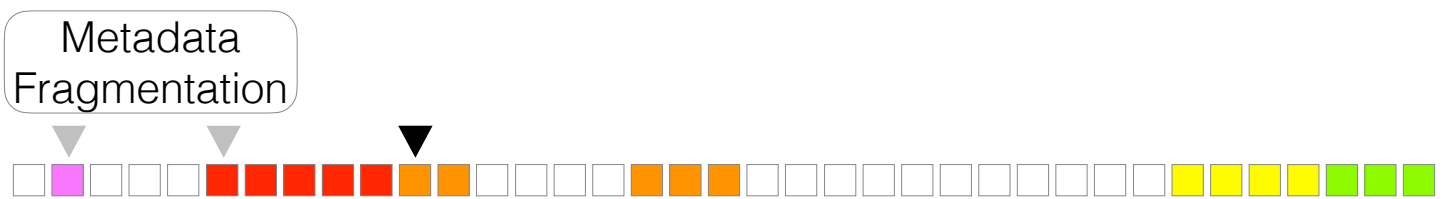




Measuring Aging

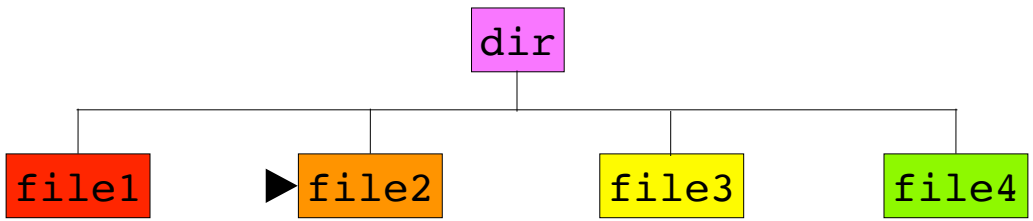
```
time grep -r random_string /path/to/filesystem
```

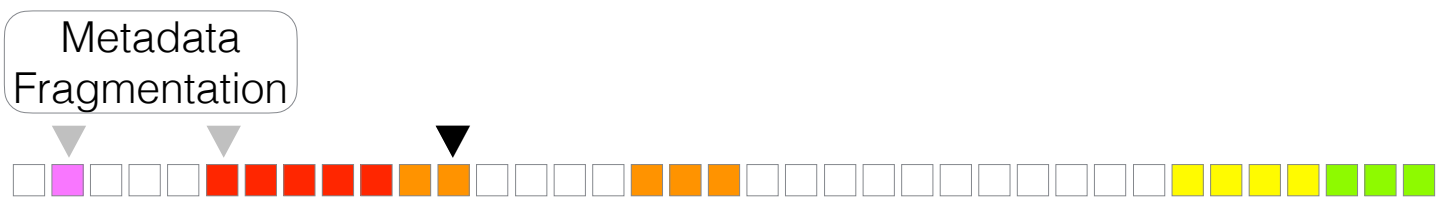




Measuring Aging

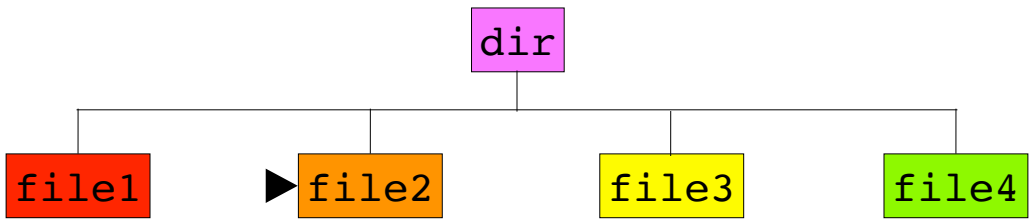
```
time grep -r random_string /path/to/filesystem
```

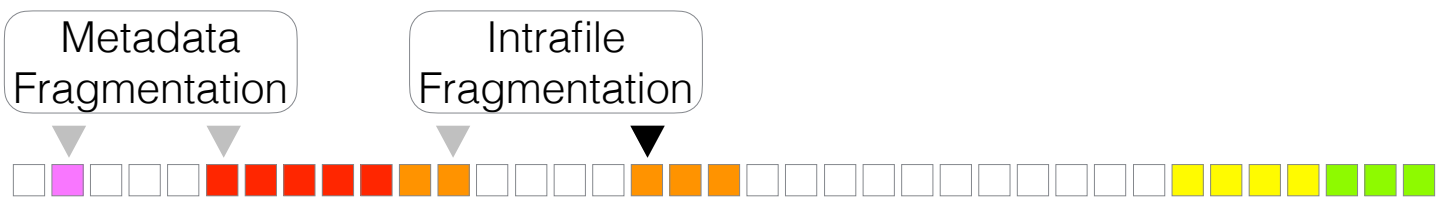




Measuring Aging

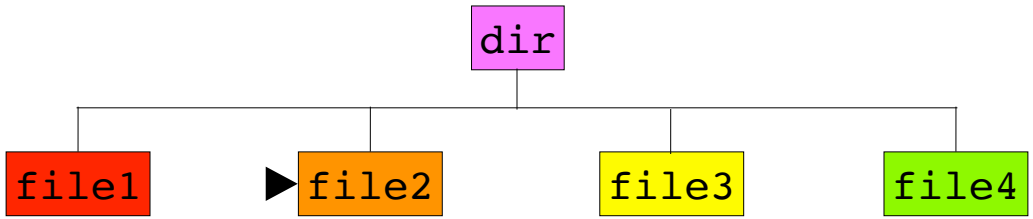
```
time grep -r random_string /path/to/filesystem
```

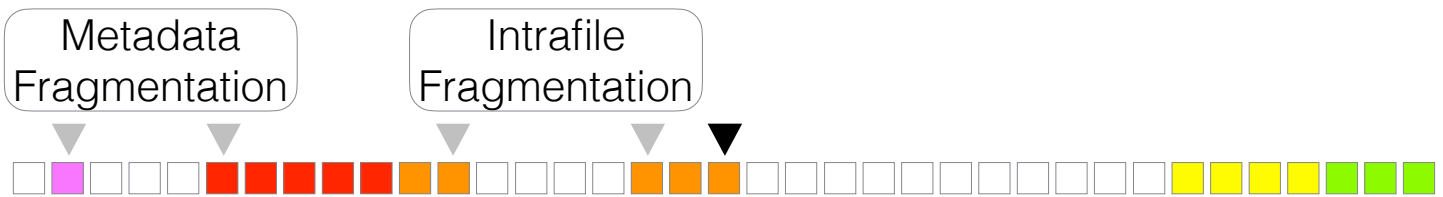




Measuring Aging

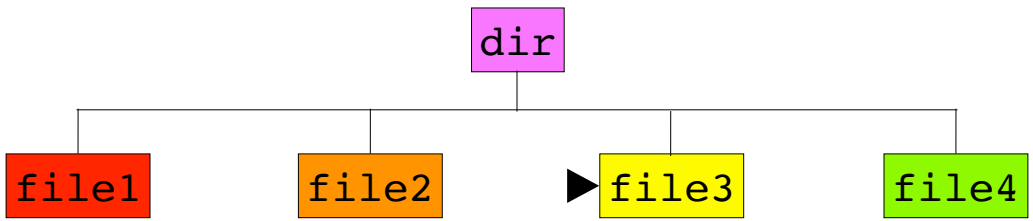
```
time grep -r random_string /path/to/filesystem
```

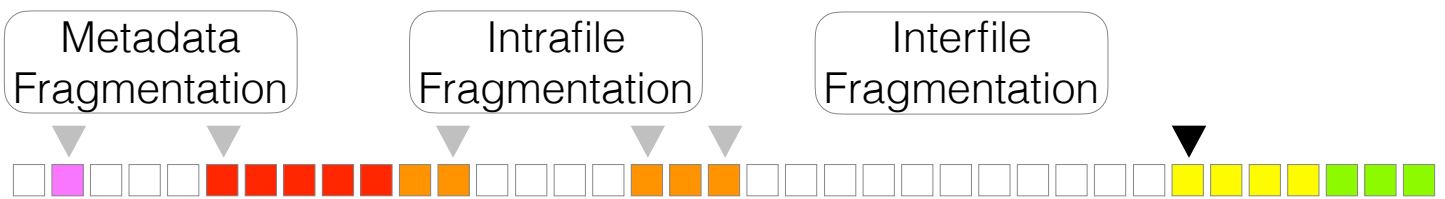




Measuring Aging

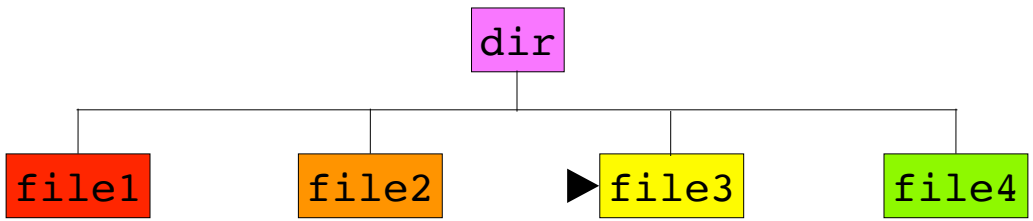
```
time grep -r random_string /path/to/filesystem
```

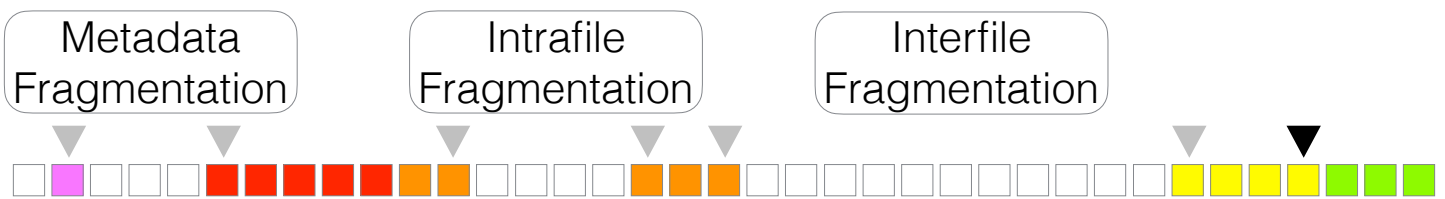




Measuring Aging

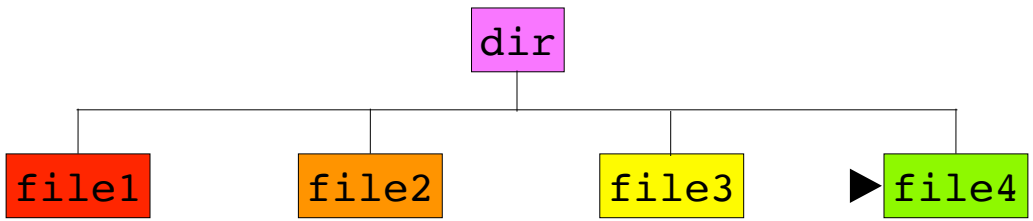
```
time grep -r random_string /path/to/filesystem
```

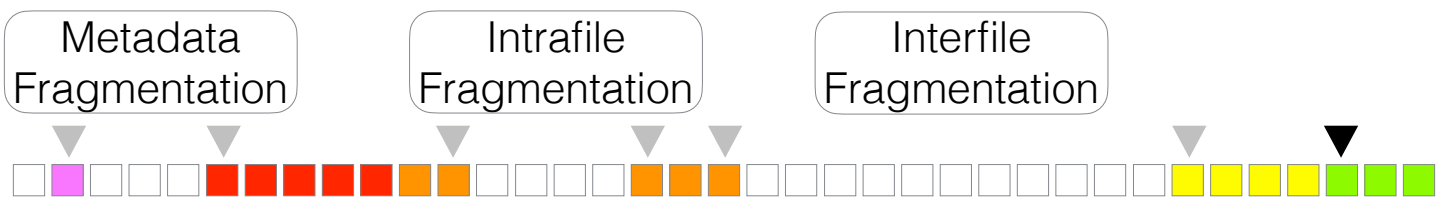




Measuring Aging

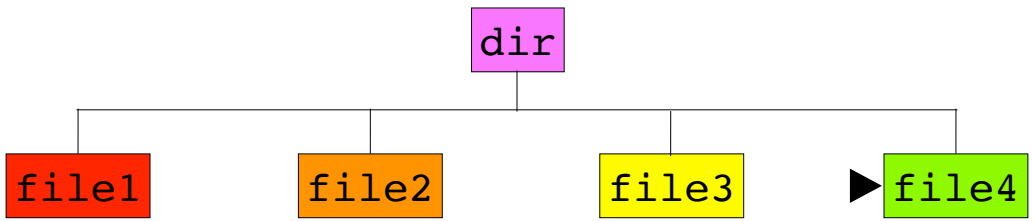
```
time grep -r random_string /path/to/filesystem
```

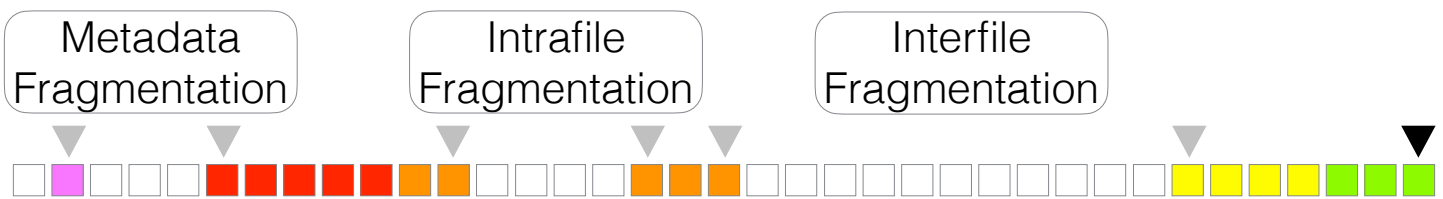




Measuring Aging

```
time grep -r random_string /path/to/filesystem
```

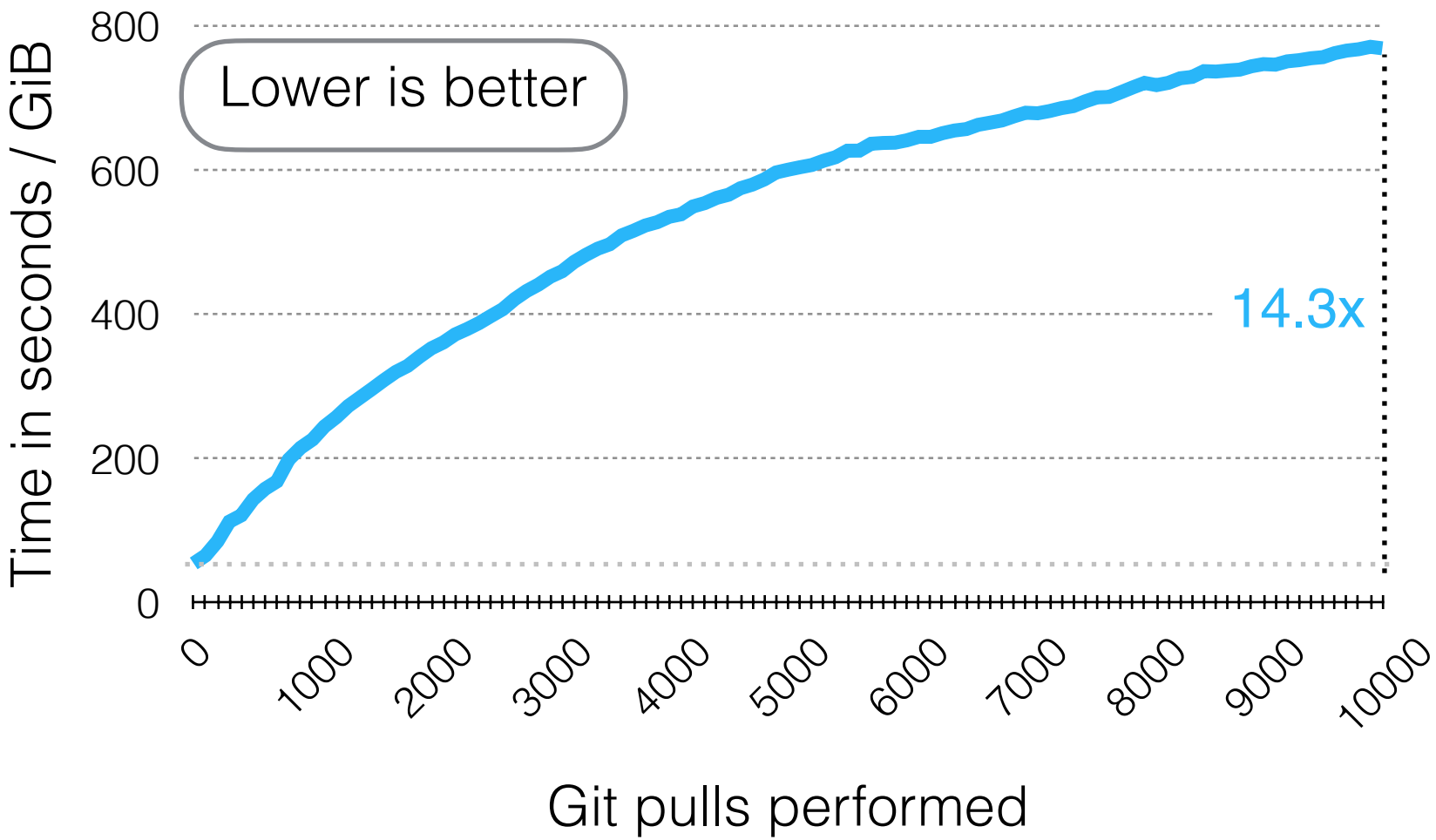




Then normalize per gigabyte read

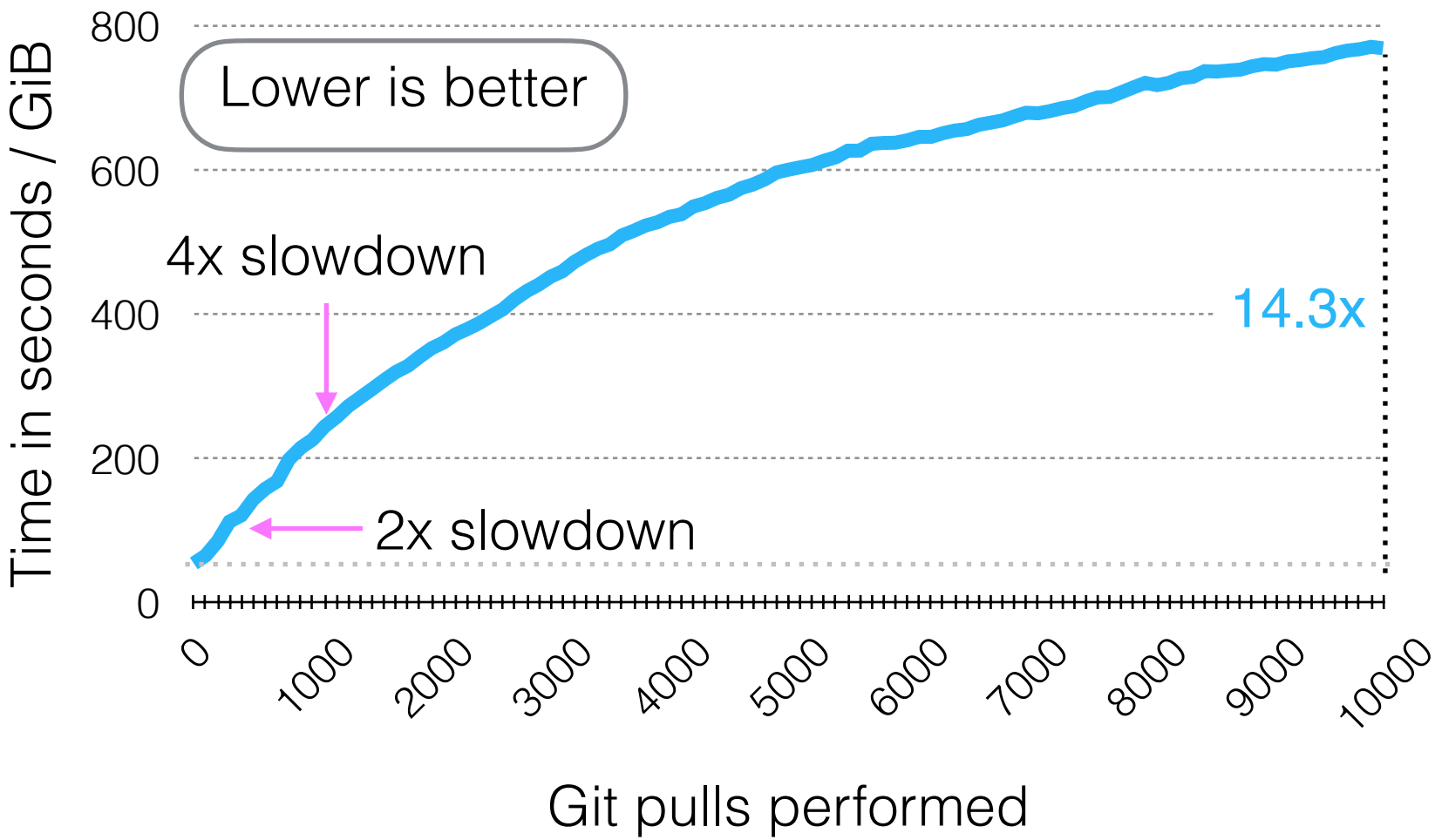
Do modern file systems really age?

Git workload on ext4 on HDD



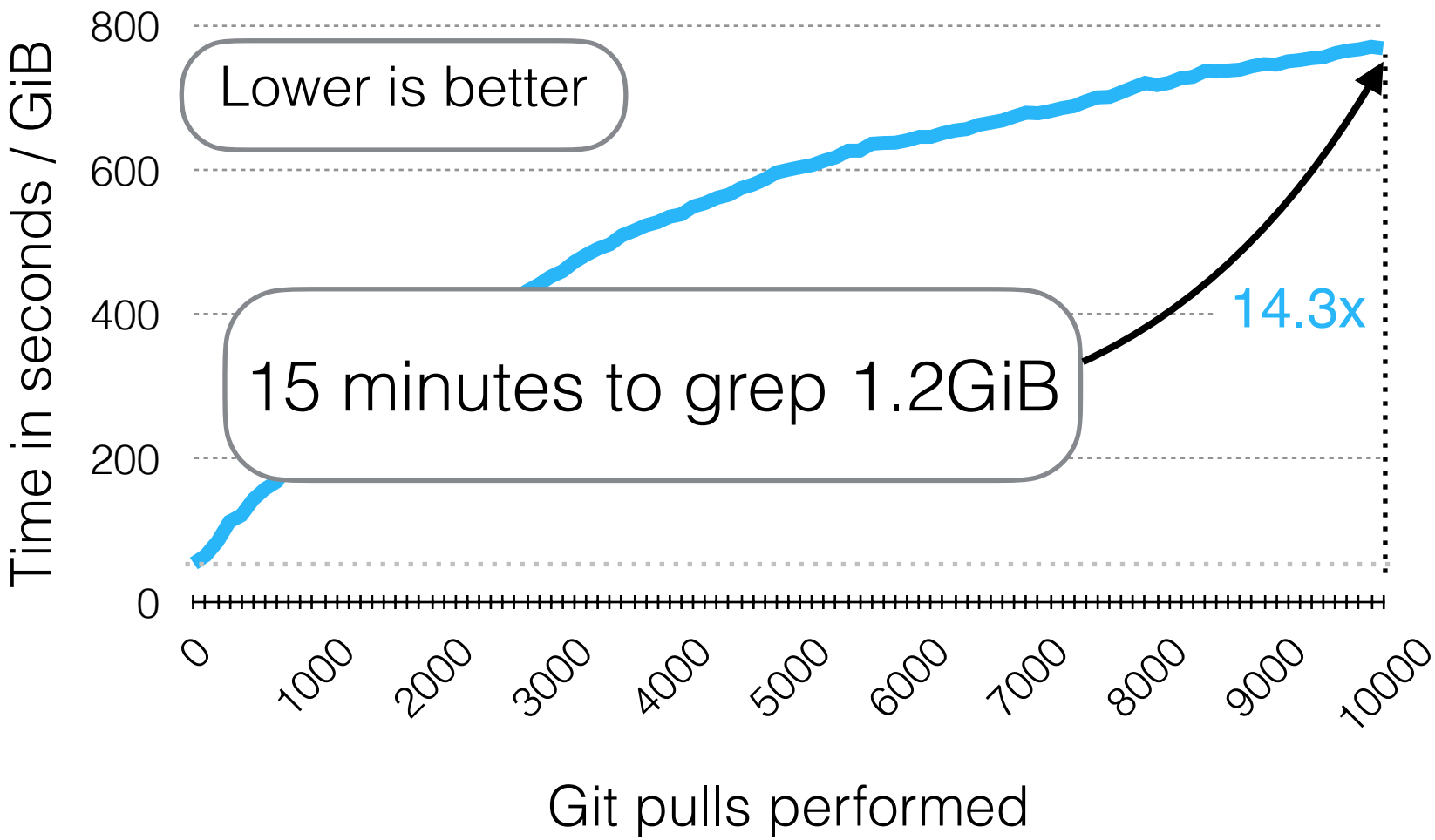
Our Setup: Cold Cache, 3.4 GHz Quad Core, 4GiB RAM,
20 GiB HDD partition - SATA 7200 RPM

Git workload on ext4 on HDD



Our Setup: Cold Cache, 3.4 GHz Quad Core, 4GiB RAM,
20 GiB HDD partition - SATA 7200 RPM

Git workload on ext4 on HDD



Our Setup: Cold Cache, 3.4 GHz Quad Core, 4GiB RAM,
20 GiB HDD partition - SATA 7200 RPM

Ruling out alternative explanations

Is it a change in the file system?

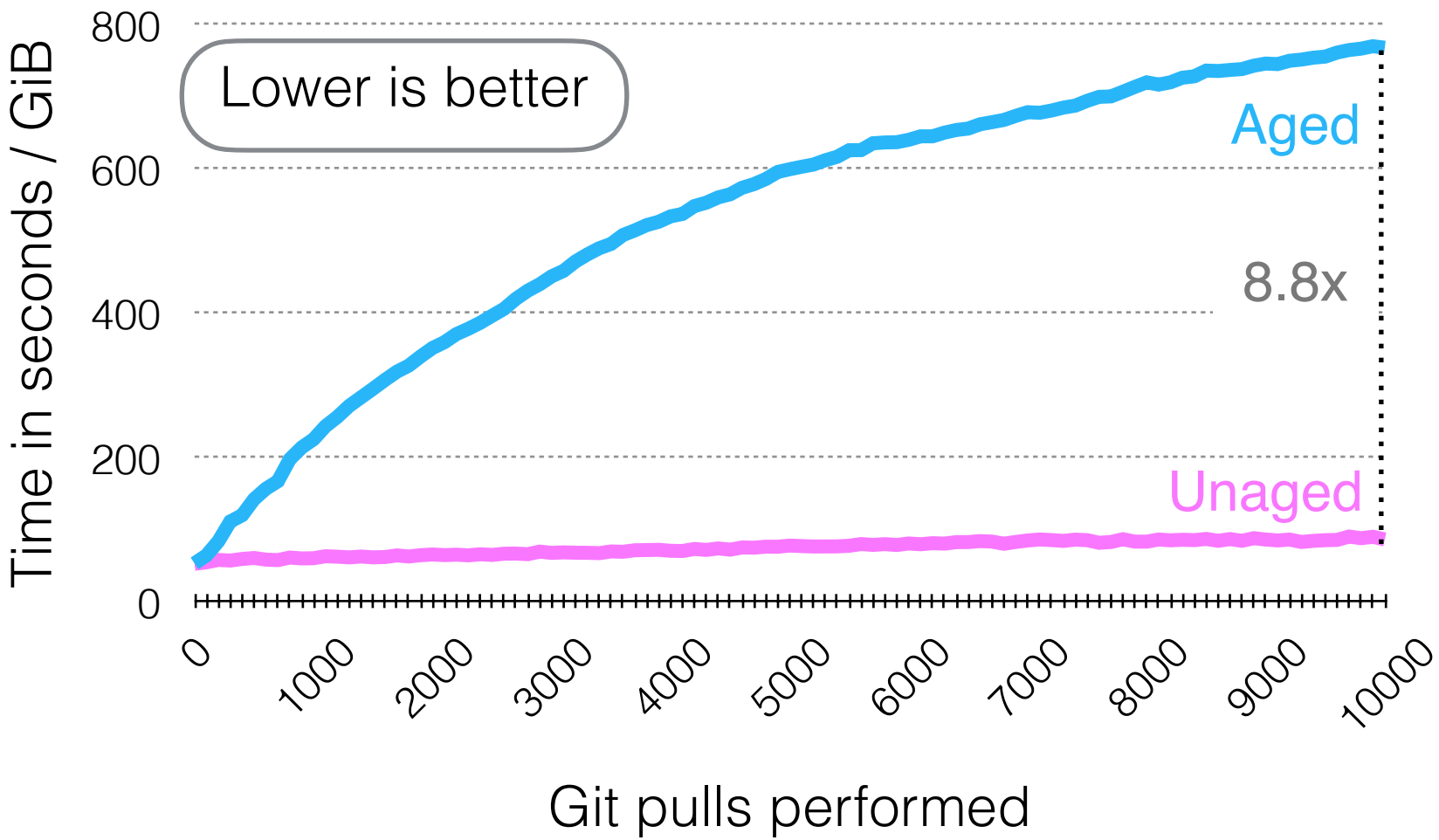
Smaller files, shallower tree, …

File System Rejuvenation

Idea: copy same logical state to new partition

- After each 100 pulls
- Compare grep cost

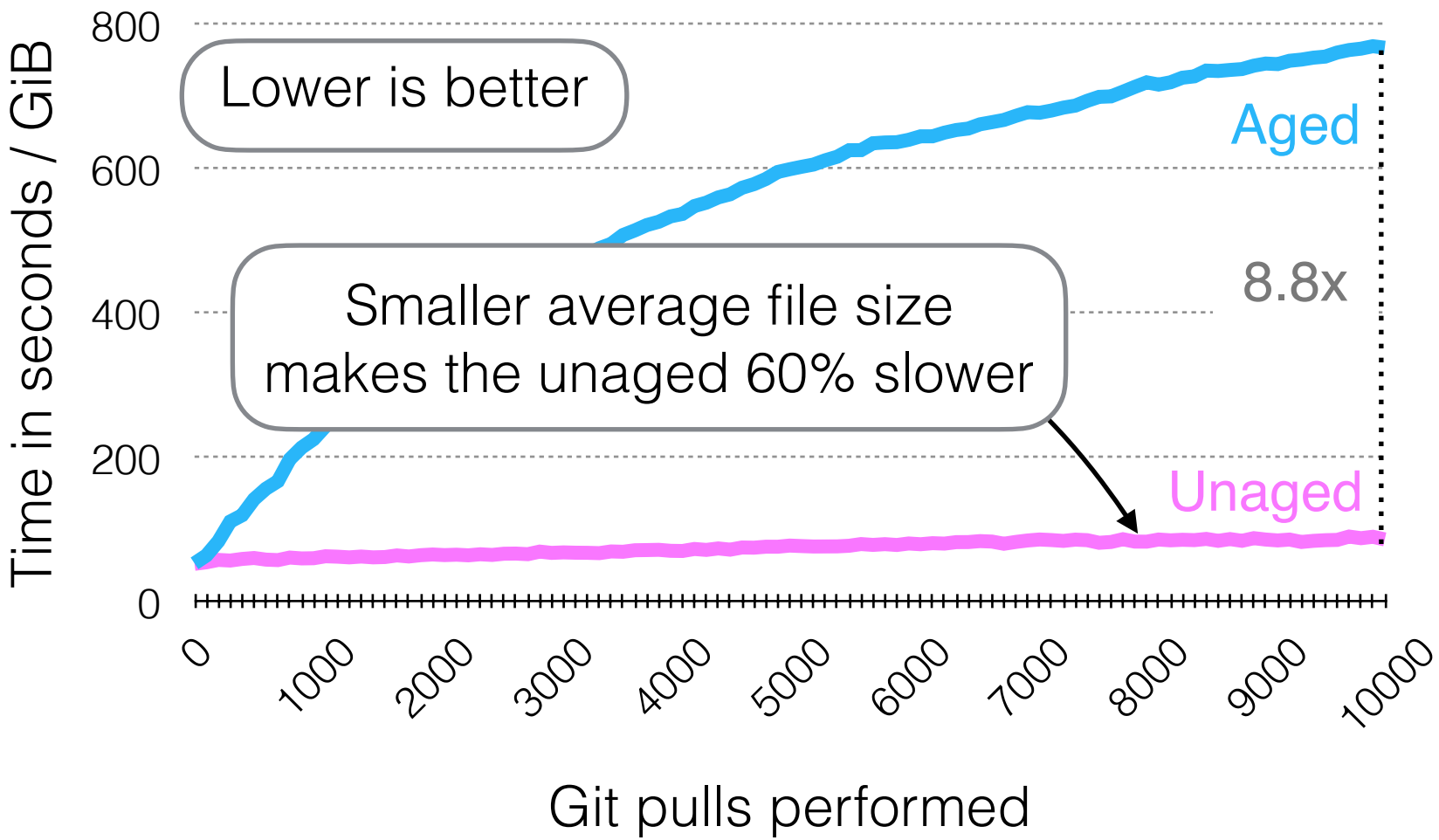
Aging ext4 with Git on HDD



Maybe it's full disks?

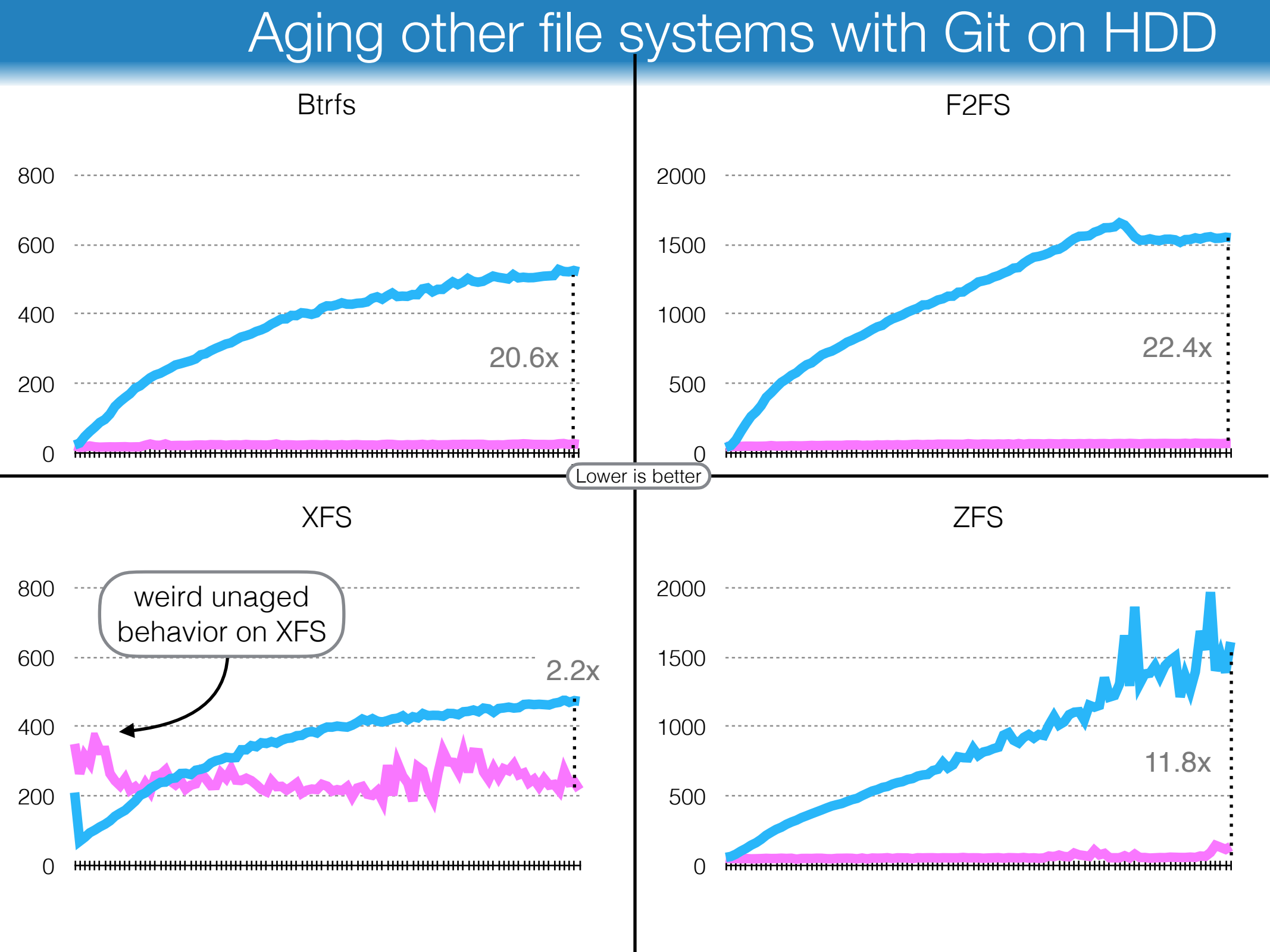
Nope: 20GiB partition, 1.2 GiB data

Aging ext4 with Git on HDD



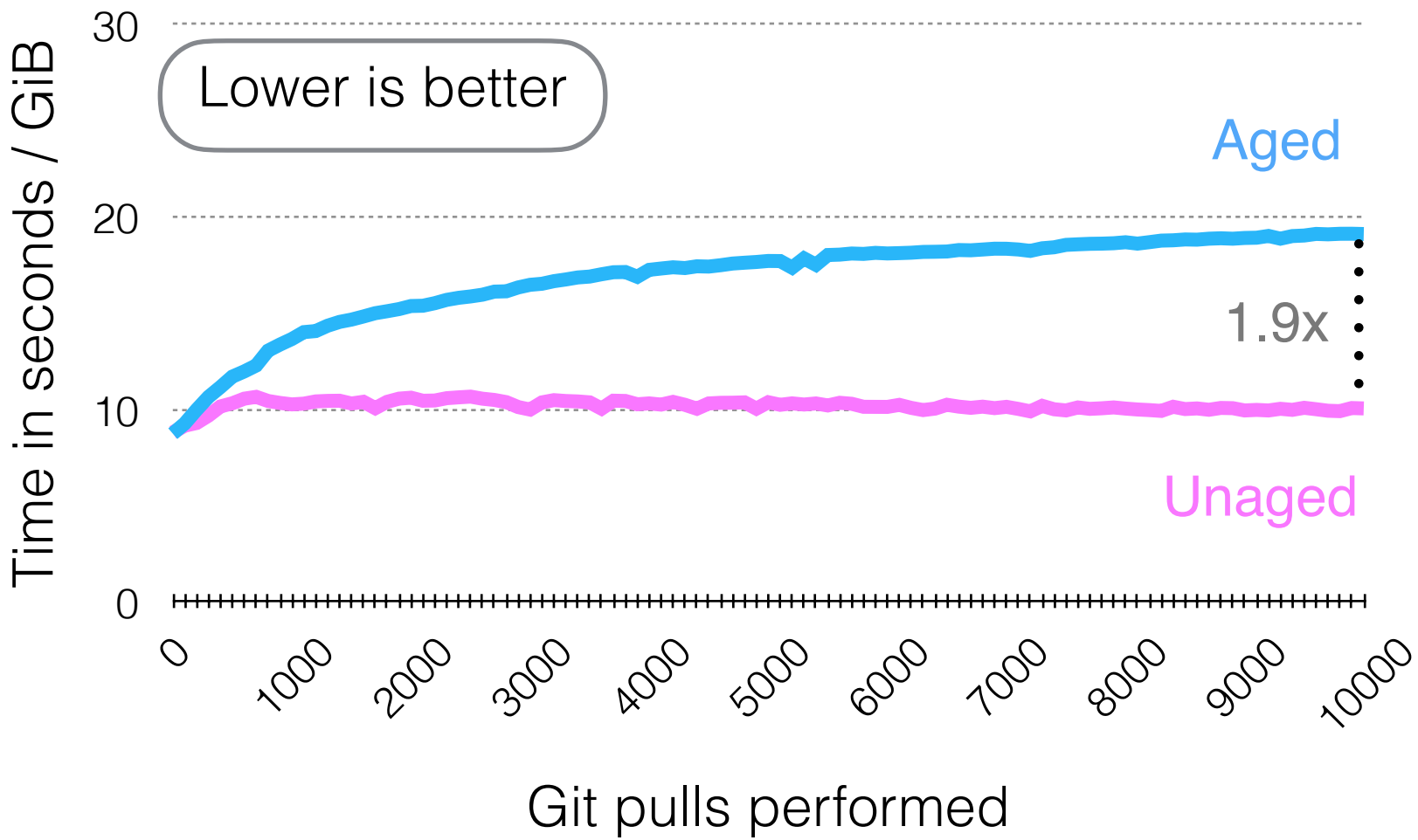
Is it just ext4?

Aging other file systems with Git on HDD



Will SSDs save us?

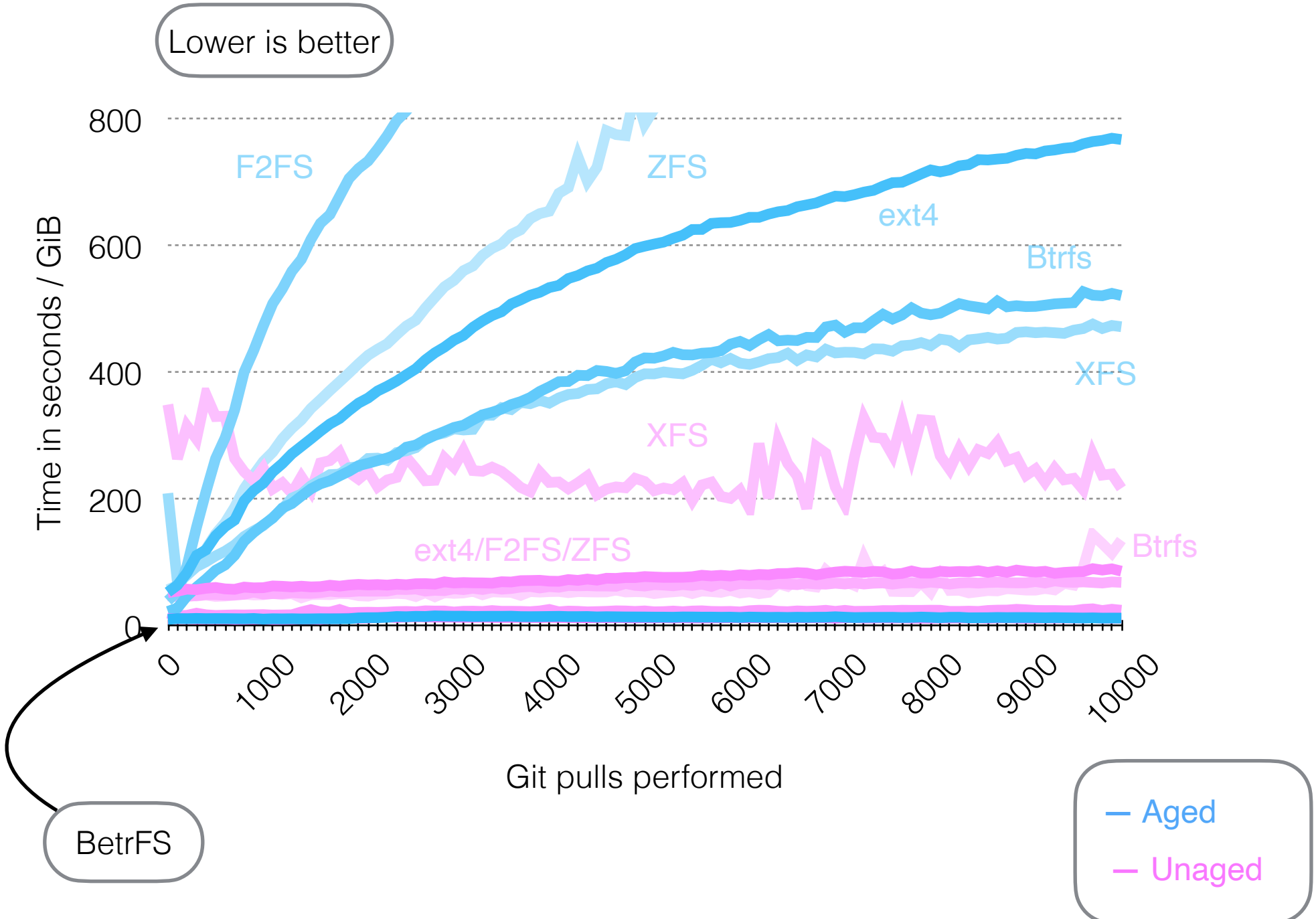
Git Workload on XFS on SSD



Other file systems give similar results (~2x slowdown)

And now for BεtrFS

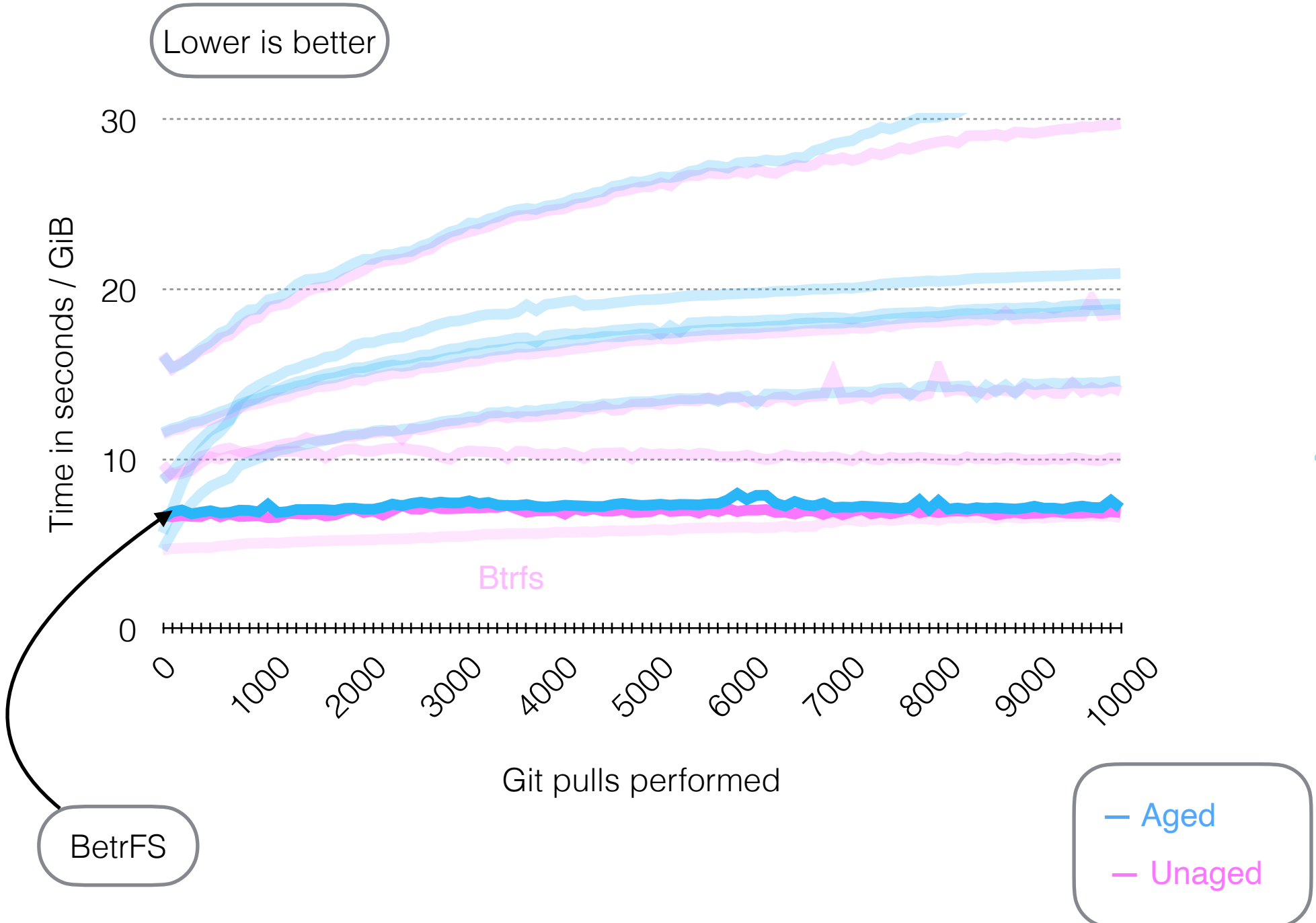
Git on BεtrFS on HDD



Git on BεtrFS on HDD

And SSDs?

Git on BεtrFS on SSD



File Systems Fated for Senescence? Nonsense, Says Science!

Alex Conway[🂡] Ainesh Bakshi[🂡] Yizheng Jiao[♢]

Yang Zhan[♢] Michael A. Bender[♠] William Jannen[♠]

Rob Johnson[♠] Bradley C. Kuszmaul[♡] Donald E. Porter[♢]

Jun Yuan[♣] Martin Farach-Colton[🂡]

[🂡]Rutgers University, [♢]The University of North Carolina at Chapel Hill, [♠]Stony Brook University, [♡]Oracle Corporation and Massachusetts Institute of Technology, [♣]Farmingdale State College of SUNY