Lepton: *a System to Transparently Compress Hundreds of Petabytes of Image Files For a File-Storage Service*

https://github.com/dropbox/lepton

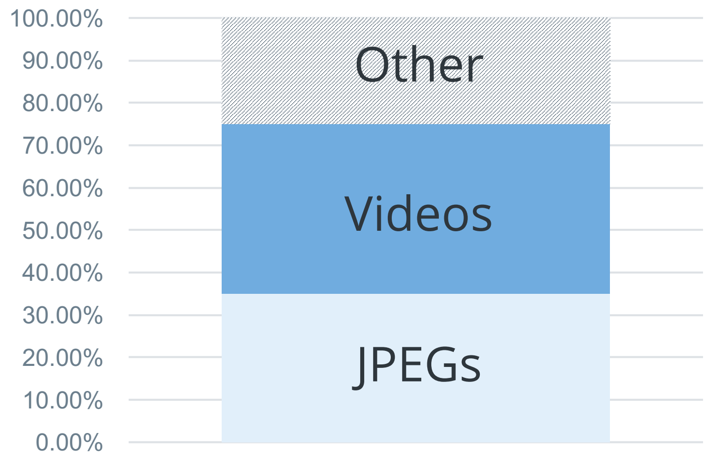
Daniel Reiter Horn, Ken Elkabany, Chris Lesniewski, Keith Winstein

Dropbox

Stanford

Storage Overview at Dropbox

- ¾ Media



- Roughly an Exabyte in storage

- Can we save backend space?

Goals

- High compression
- Byte for byte transparency
- Distributed 4MB chunks
- Fast [Streaming 100 Mbit/s decode]
- Secure
- Trustworthy

Related Work

	Comp ratio	Bit-for-bit	Distributed	Fast	Secure	Trustworthy
packJPG	✅	✅	❌	❌	❌	✓
MozJPG	✓	❌	❌	✓	✓	
JPEGrescan	✓	❌	❌	✅	✓	
zlib, brotli, zstd	❌	✅	✅	✅	✅	✅
Lepton	✅	✅	✅	✅	✅	✅

JPEG File

- Header
- 8x8 blocks of pixels
  - DCT transformed into 64 coefs
    - Lossless
  - Each divided by large quantizer
    - Lossy
  - Serialized using Huffman code
    - Lossless

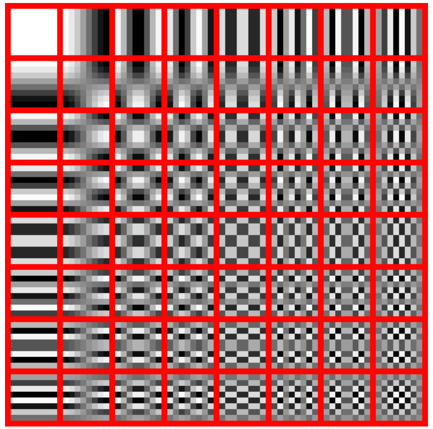


Image credit: wikimedia

Entropy Coding

- Huffman code:
  - Favor frequently seen coefs, 0’s
- Arithmetic Code:
  - Look at values so far
  - Predict next value
  - Good prediction = fewer bits
  - Bad prediction = more bits

Entropy Coding

- Huffman code:
  - Favor frequently seen coefs, 0’s
- Arithmetic Code:
  - Look at values so far to predict next value
  - Good prediction = fewer bits
  - Bad prediction = more bits

Lepton: 2 key ideas

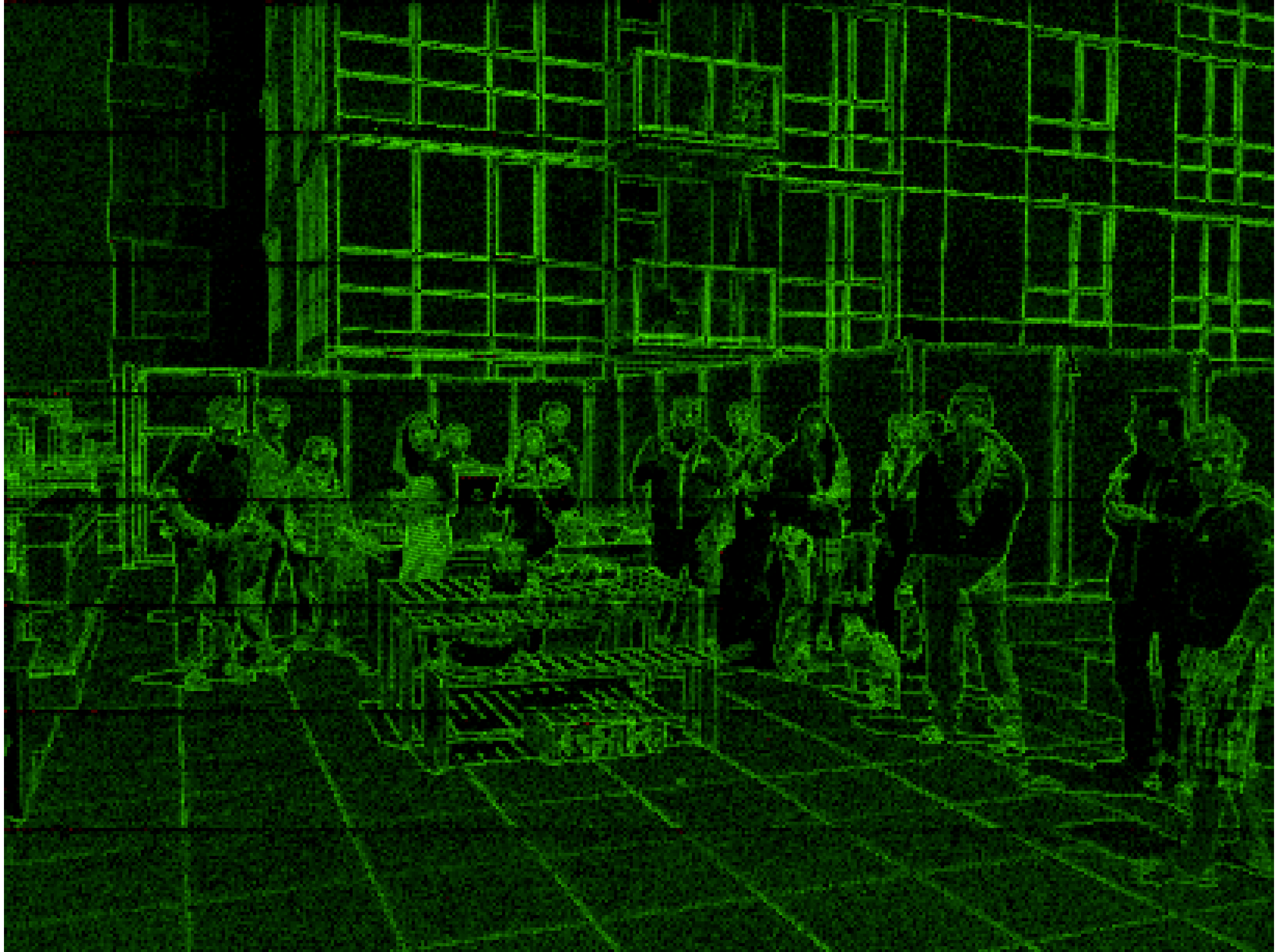
- Streaming arithmetic code with sophisticated predictor
- Make image subsets independent

Streaming Arithmetic Code

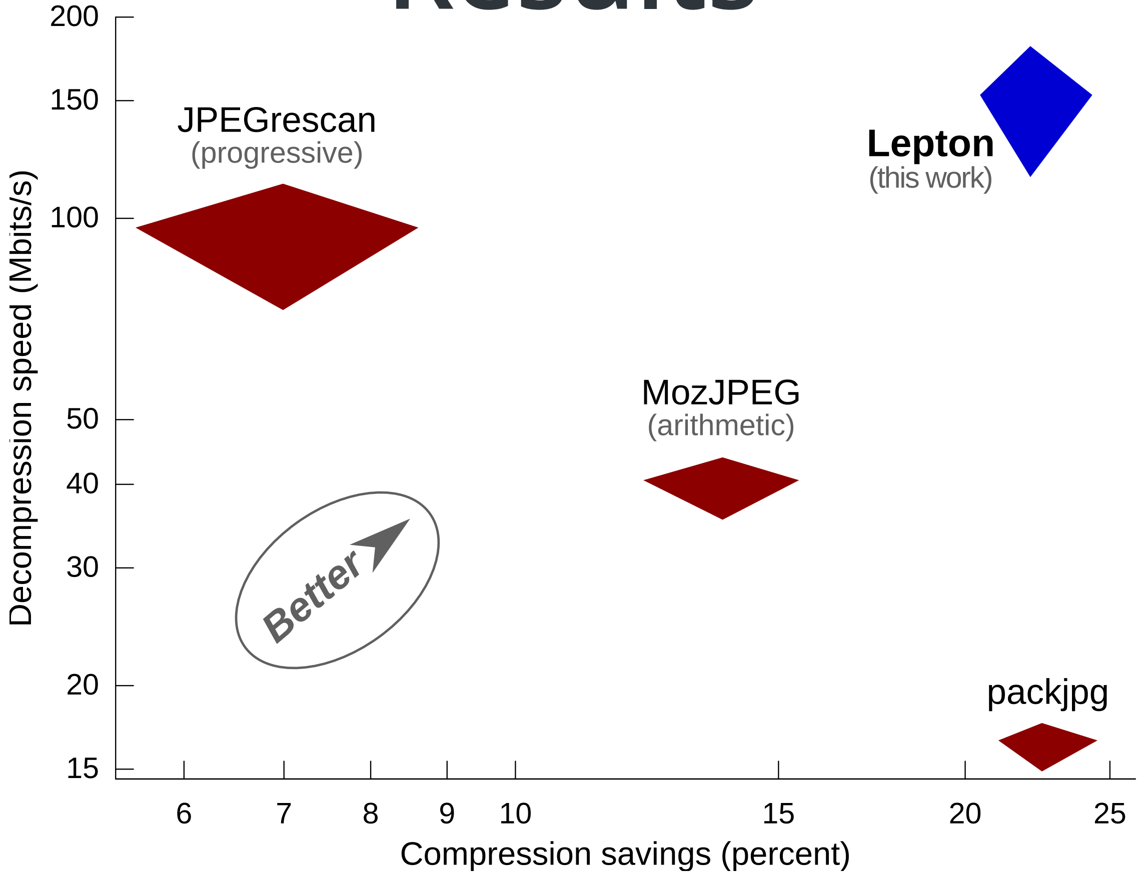
- Lepton Predictor
  - Massive 2.2MB probability model
  - Pulls out correlation across files
  - Pixel space prediction of DC value
  - Predictions for horizontal and vertical patterns

Making image subsets independent

- Probability model reset per thread
- Huffman encoder state serialized in Lepton header per thread
  - Allows 8-way parallel decode
  - Helps to attain 100mbit



Results


200
150
100
50
40
30
20
15
Decompression speed (Mbits/s)
6
7
8
9
10
15
20
25
Compression savings (percent)
JPEGrescan
(progressive)
Lepton
(this work)
MozJPEG
(arithmetic)
packjpg
Better

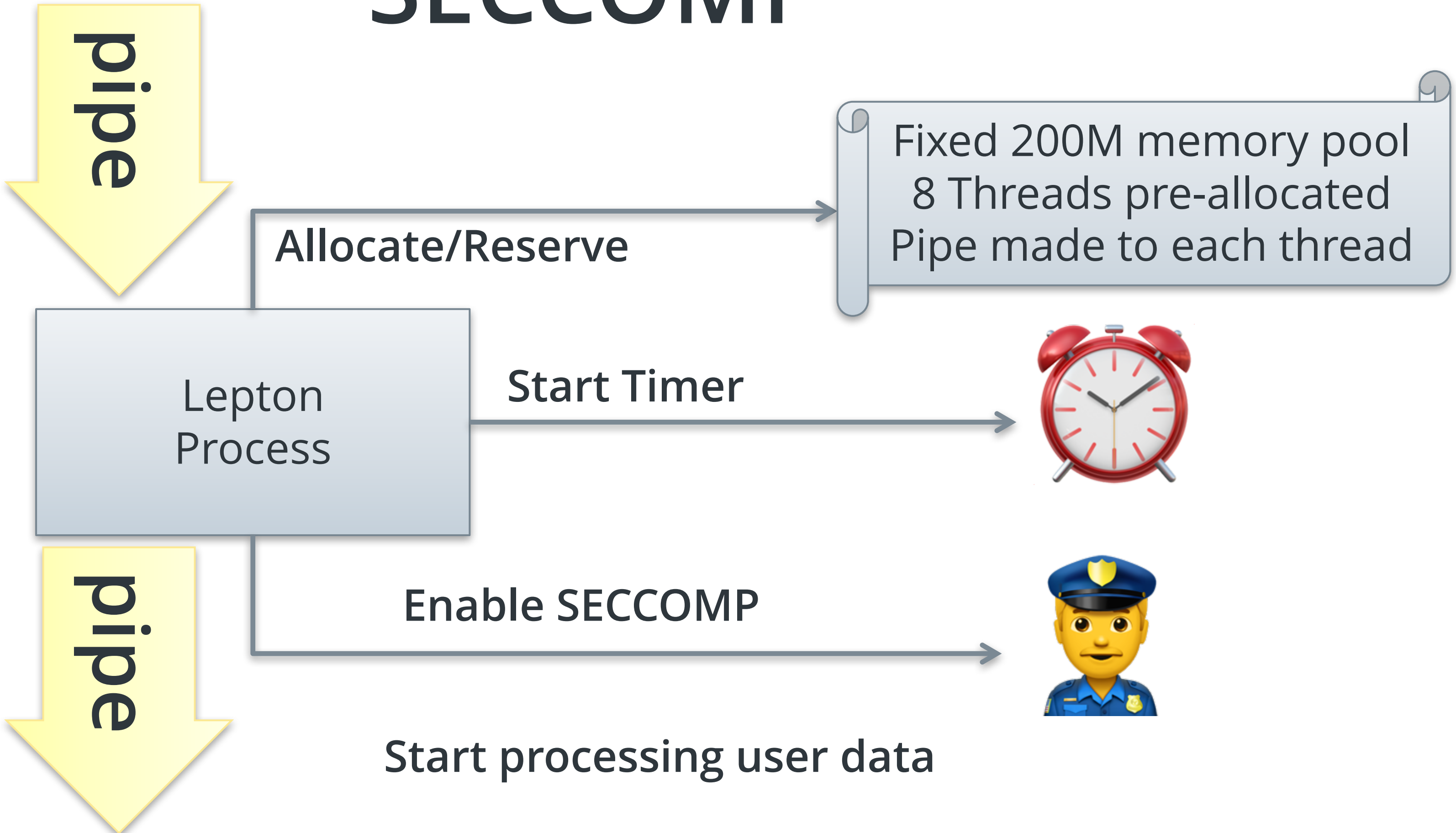
Security Challenges

- Concern about malicious crafted JPEG
  - Triggering *Buffer overruns*
  - Avoiding safety checks elided by *Undefined behavior*
  - Exploiting *Use-after-free* errors

Solution: SECCOMP

- Restrict Syscalls
  - read/write/sigreturn/exit
- Severely limits scope of any attack
  - Attacker could only write stdout or read stdin
  - Separate process per encode or decode
- Awkward Ergonomics
  - No dynamic allocation, no mutex, no thread create

SECCOMP



pipe
Allocate/Reserve
Fixed 200M memory pool
8 Threads pre-allocated
Pipe made to each thread
Lepton
Process
Start Timer
Enable SECCOMP
pipe
Start processing user data

Trustworthiness Requirements

- Bit-for-bit: Dropbox as a filesystem
  - Sha256 must match on reconstruction of original
- Determinism
  - Decodes need to work every single time
- Resistant to operator/developer error
  - We are our own worst enemy

Bit for bit roundtrip

- Lepton Compress, Encrypt File
- MD5 result
- Decrypt and Lepton Decompress
  - Decrypt in a separate process address space
  - Make sure sha256 matches client-computed
  - If not, repeat, but with zlib algorithm
- Upload; make sure Md5 matches

Determinism

Why we need it

- Compression uses all prior data read so far to predict next bit
- A single bit can change prediction
  - Nondeterministic prediction source would render Lepton file unreadable

Determinism

- “Qualify” every Lepton binary
  - Build each binary with *icc*, *gcc*
  - Turn on *gcc* address sanitizer
  - Run over 4 billion images in single+multithread
  - make sure *icc* matches *gcc* in both cases
- Upon Qualification
  - Mark *icc* binary as qualified, allow it to be pushed

Determinism

- Fuzz the code
  - Ran Coverity
  - 3rd party ran a checker
- Added array bounds checks
  - 10% performance degradation
  - Worth it.

Programmer/Operator Error

- Safety Net
  - When the system is changed, Safety net activated
- All uploads saved to a S3 bucket
  - Encoded with Zlib, then encrypted
- Bucket expires files after 30 days

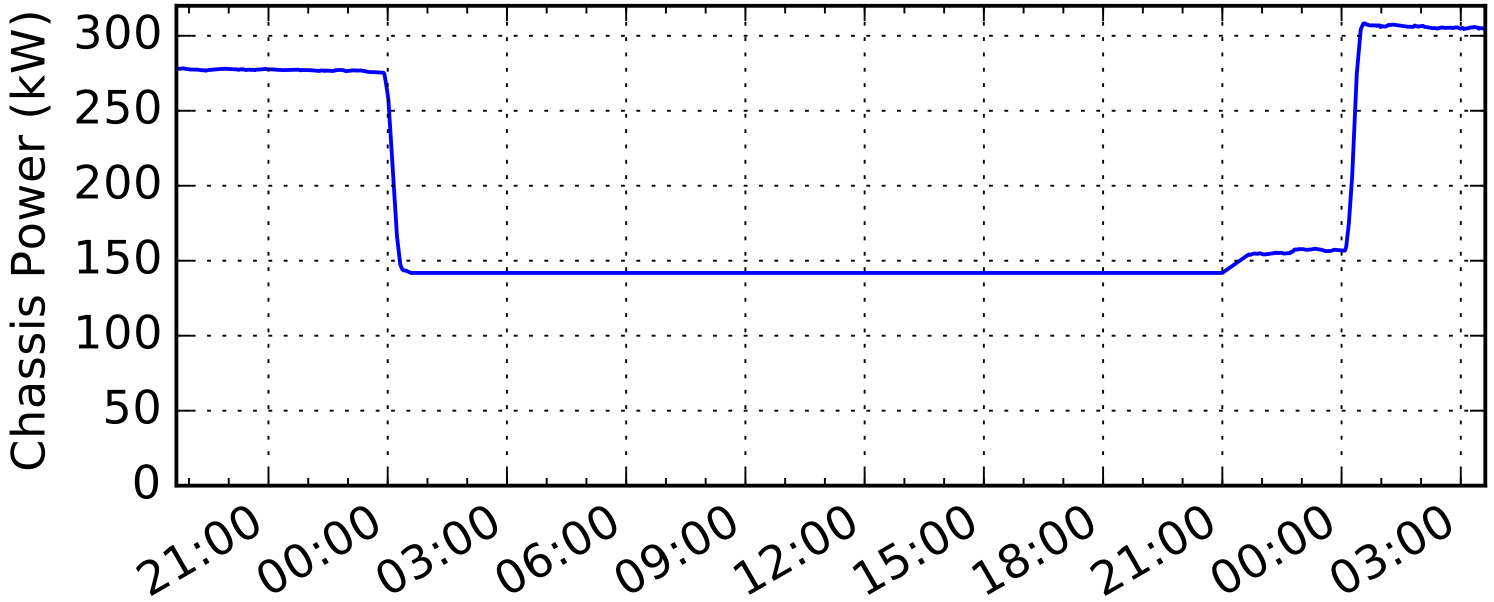
Supported (Strange) JPEGs

- Unexpected 0 runs near file end
  - May be from full SD cards, disk errors, power failure
  - Zero runs in the middle only ~0.003% files
- Garbage at the end
  - Ex: my H/D has files with TV-ready previews at end
- Arbitrary bits filling partial-bytes

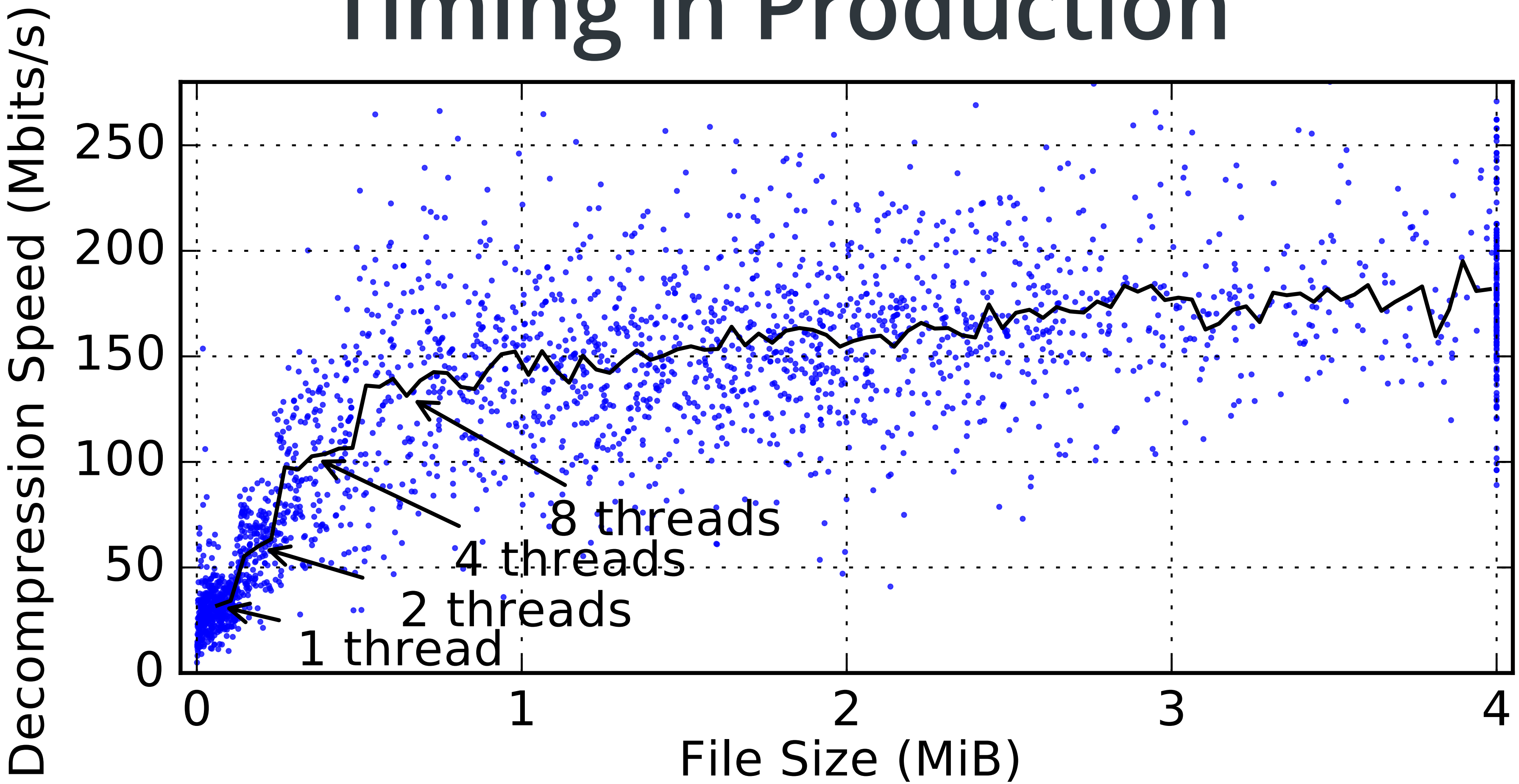
Deployment

- Lepton has encoded 150 billion files
  - 203 PiB of JPEG files
  - Saving 46 PiB
  - So far…
    - Backfilling at > 6000 images per second

Power Usage at 6,000 Encodes



Timing in Production



War Stories

War Story: Safety Net

Situation

- Safety net required 2x traffic
- Failover requires extra capacity
- First routine failover test post-Lepton
  - Traffic shifted from Virginia to Texas
  - Texas S3 proxies overwhelmed by safety net traffic

War Story: Ancient Code Push

Situation

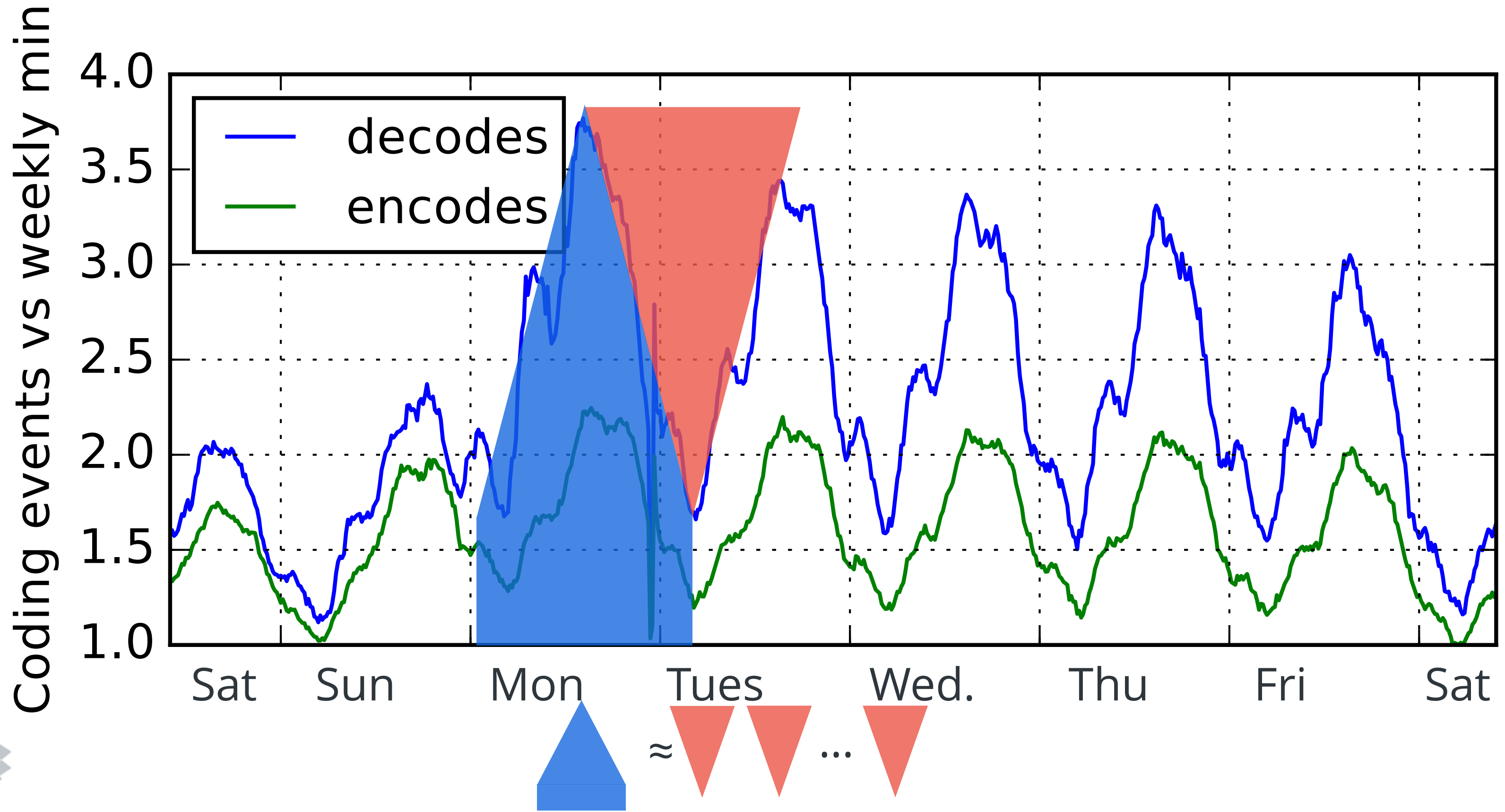
- Example Operator Error
- Bad default in deployment form
  - Field specifying git hash to deploy
  - If left blank: oldest qualified version deployed
- Features deprecated since original

Ancient Code Push

Detection

- Lowered availability on “Canary”
- Alarm: Lepton rejected decode of stored blocks
  - Due to deprecated features

Weekly and Diurnal Patterns



Ancient Code Push

Resolution: No durability impact

- Lepton disabled after 2 hours
  - Scanned billions of images uploaded since incident
  - Fixed all 17 images that had deprecated features
  - No data loss
- Recovery time = Small multiple of incident time

Conclusions

- Determinism is important
- Undefined behavior is undesirable
- Configuration management
- Safety in the face of human operators and developers

Acknowledgements

- Thanks to Jongmin Baek, Sujay Jayakar, Mario Brito, Preslav Le, David Mah, James Cowling, Nipunn Koorapati, Lars Zornes, Rajat Goel, Bashar Al-Rawi, Tim Douglas, Oleg Guba, Ross Delinger, Dimitry Kotlyarov, Nahi Ojeil, Bean Anderson, Devdatta Akhawe, Ziga Mahkovec, David Mann, Jeff Arnold, Jessica McKellar, Akhil Gupta, Aditya Agarwal, Arash Ferdowsi, and Drew Houston
- Matthias Stirner, Gopal Lakhani, Archie Russell, Loren Merritt for their inspiring JPEG compression work

- Questions?