Fast checkpointing with criu-image-streamer

Nicolas Viennot
Two Sigma

LPC 2020, Aug 24

Important Legal Information

This document is being distributed for informational and educational purposes only and is not an offer to sell or the solicitation of an offer to buy any securities or other instruments. The information contained herein is not intended to provide, and should not be relied upon for, investment advice. The views expressed herein are not necessarily the views of Two Sigma Investments, LP or any of its affiliates (collectively, “Two Sigma”). Such views reflect the assumptions of the author(s) of the document and are subject to change without notice. The document may employ data derived from third-party sources. No representation is made by Two Sigma as to the accuracy of such information and the use of such information in no way implies an endorsement of the source of such information or its validity.

The copyrights and/or trademarks in some of the images, logos or other material used herein may be owned by entities other than Two Sigma. If so, such copyrights and/or trademarks are most likely owned by the entity that created the material and are used purely for identification and comment as fair use under international copyright and/or trademark laws. Use of such image, copyright or trademark does not imply any association with such organization (or endorsement of such organization) by Two Sigma, nor vice versa

Google Preemptible VMs
Up to 5x cheaper
30sec eviction deadline

16 CPUs, 128GB RAM, 2GB/s Network
Goal: **Checkpoint in 30s**

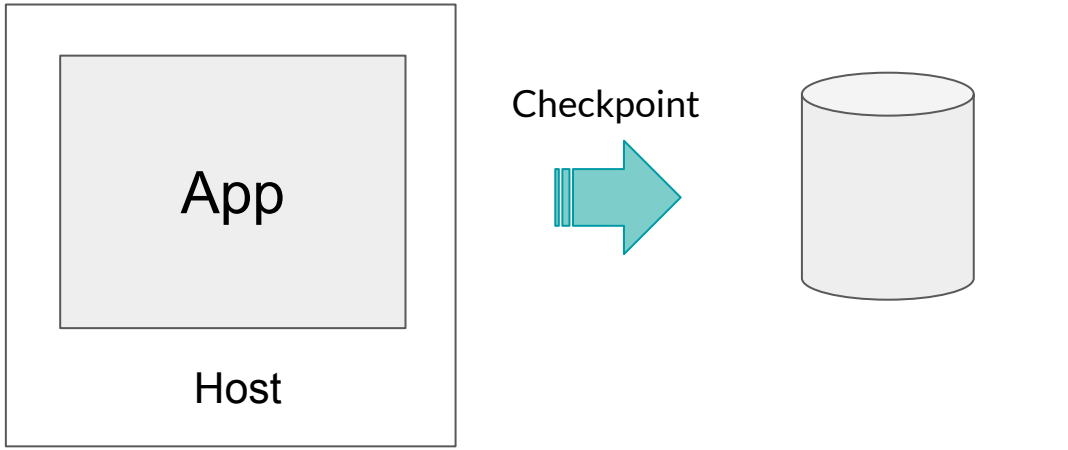
16 CPUs, 128GB RAM, 2GB/s Network

Goal: **Checkpoint in 30s**

128/30 = 4.2GB/s


SHOEI

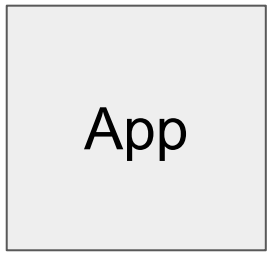
Problem: Fast checkpointing



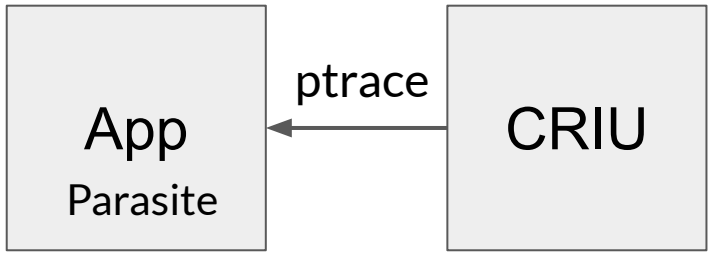
Running Application

Remote Storage
AWS S3, GCS

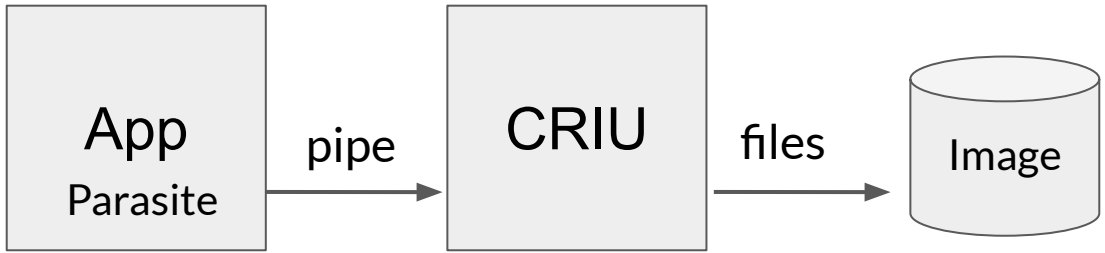
Checkpointing: the old way



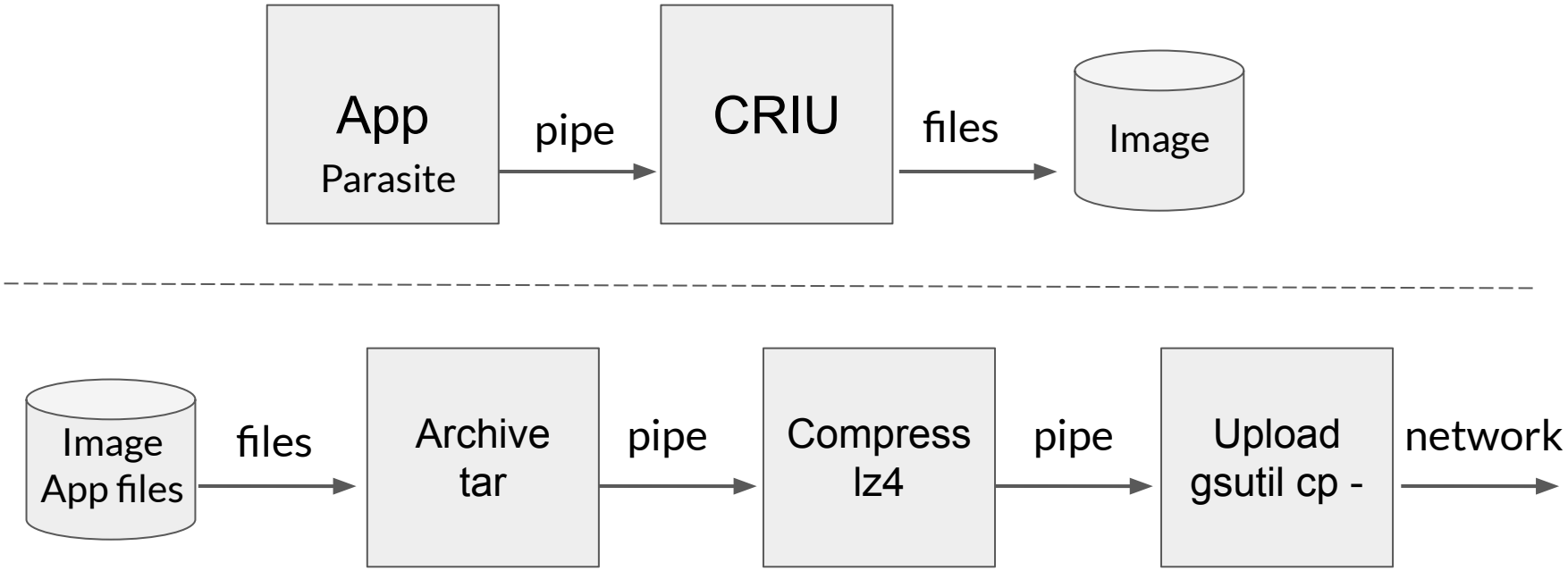
Checkpointing: the old way



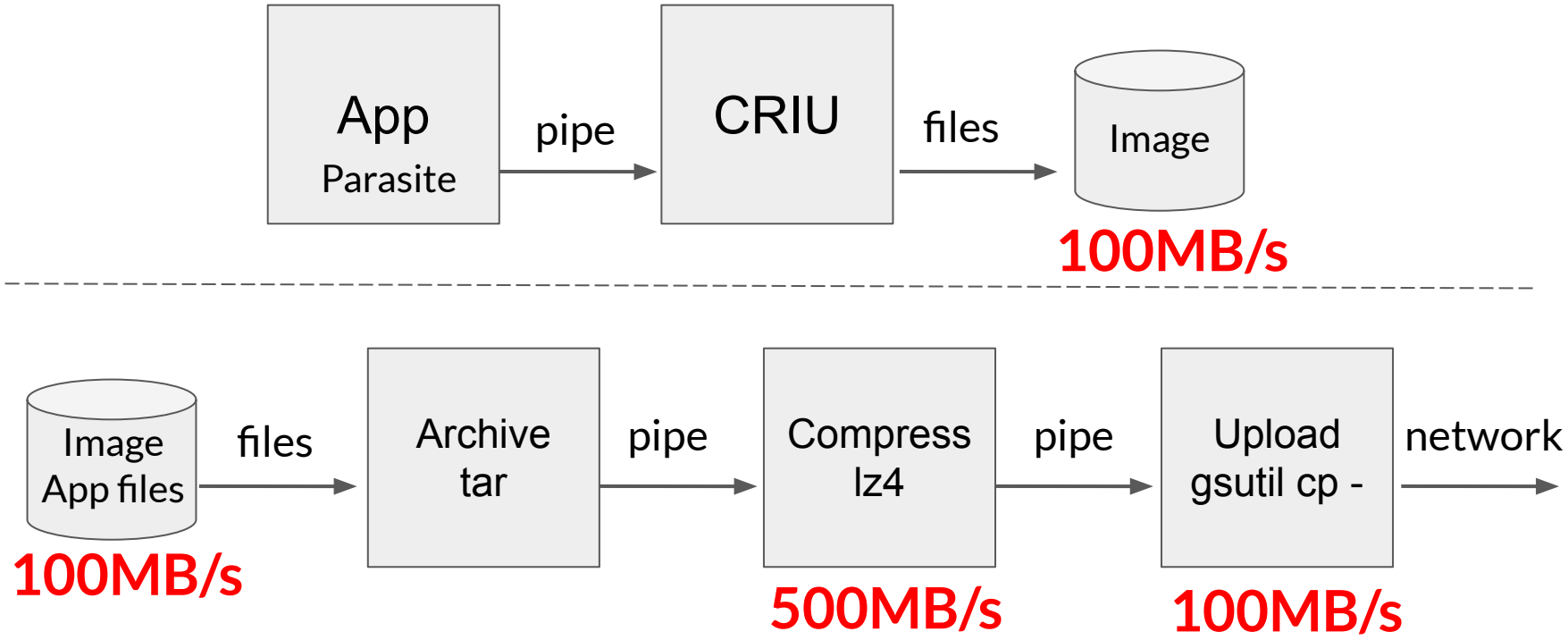
Checkpointing: the old way



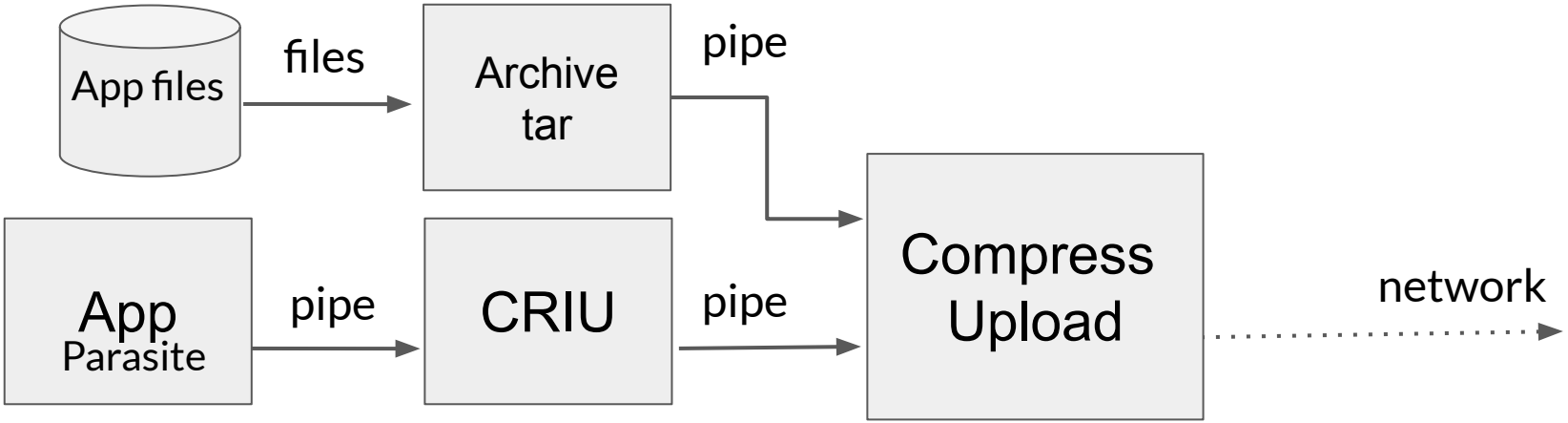
Checkpointing: the old way



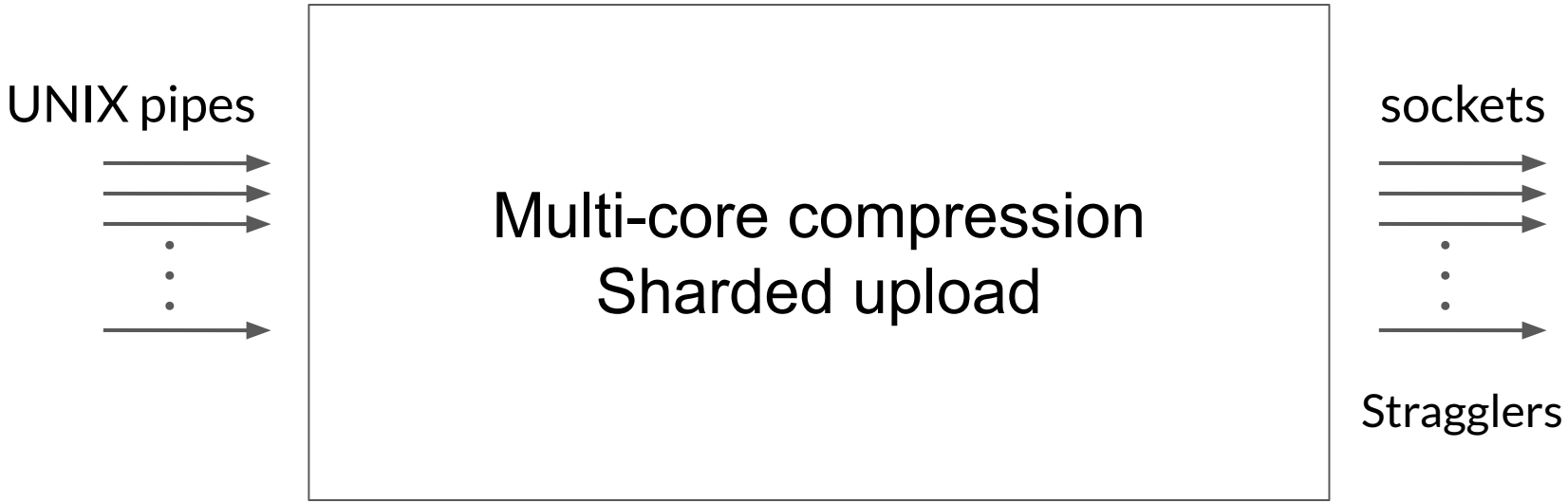
Checkpointing: the old way



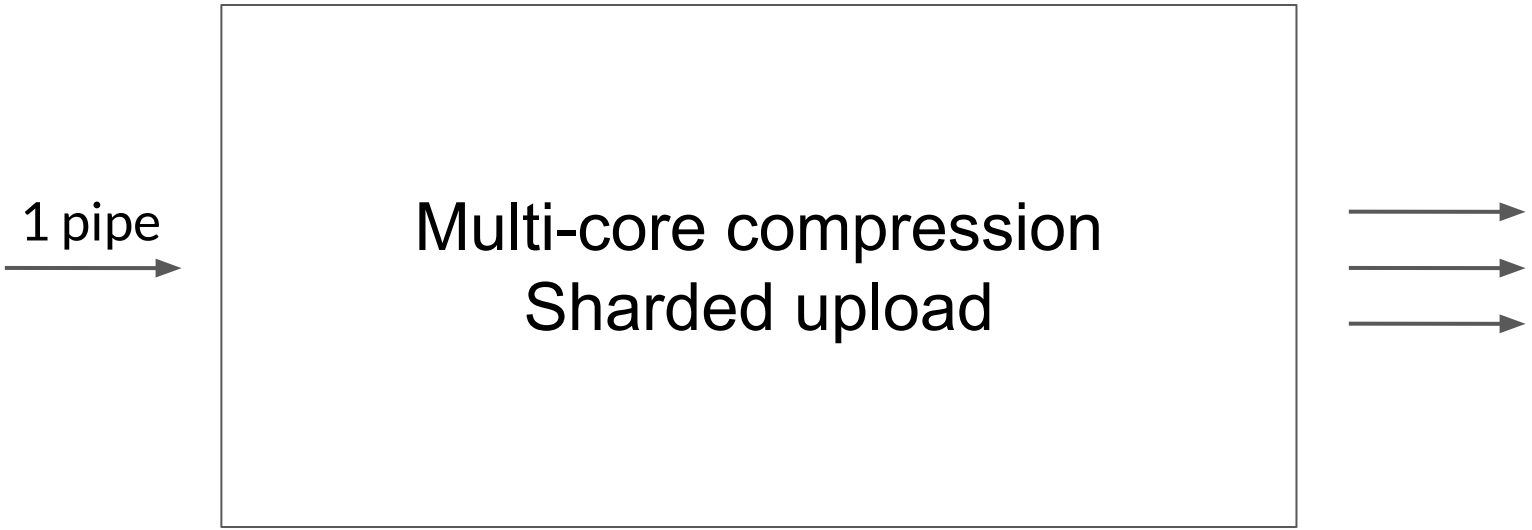
Streaming Checkpointing



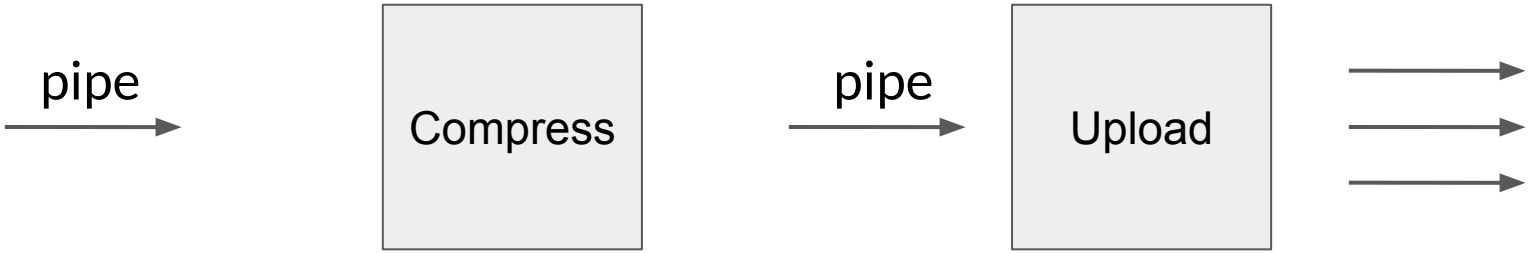
Problem: Compress and upload stream



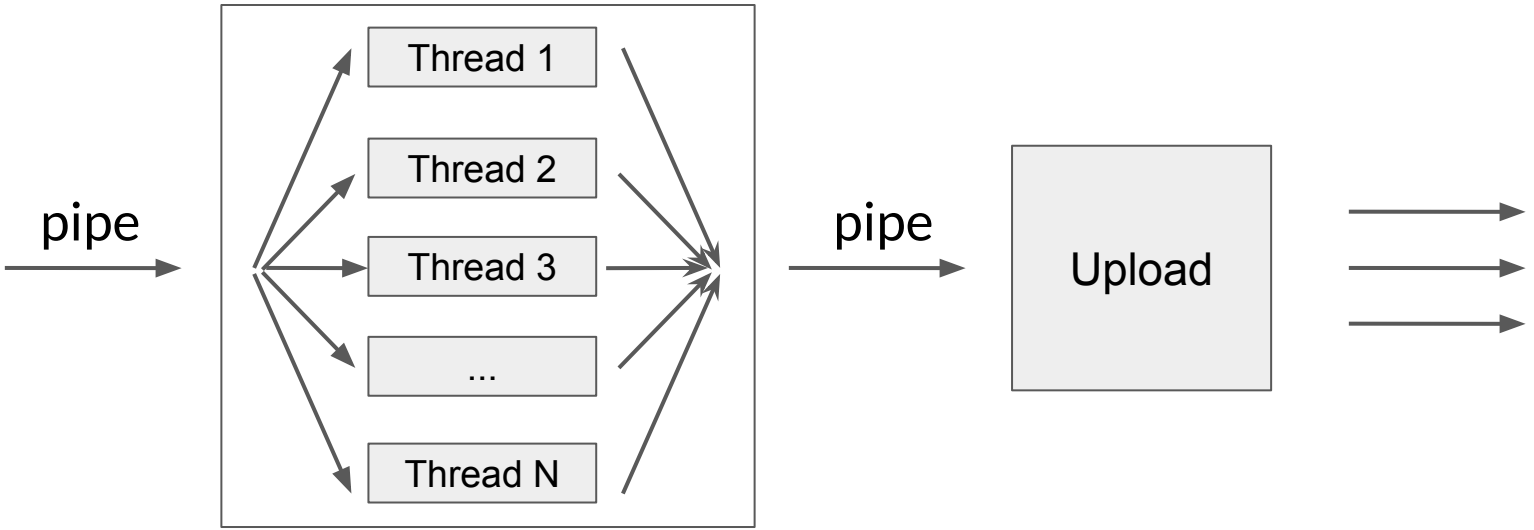
Problem: Compress and upload stream



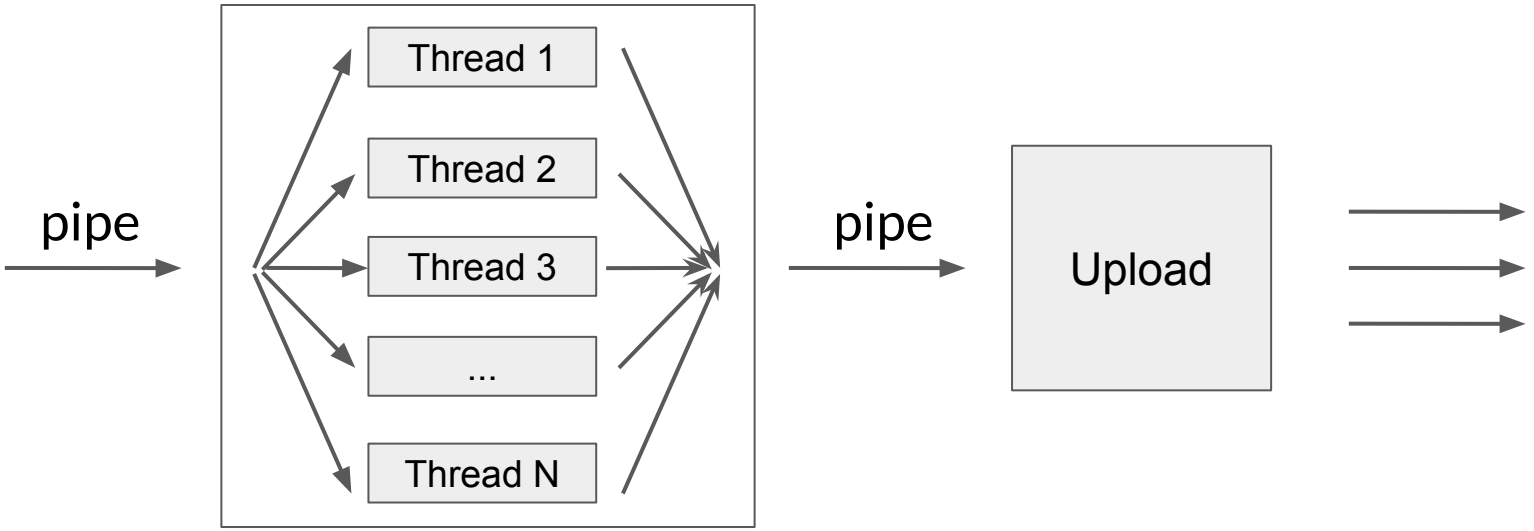
Problem: Compress and upload stream



Problem: Compress and upload stream

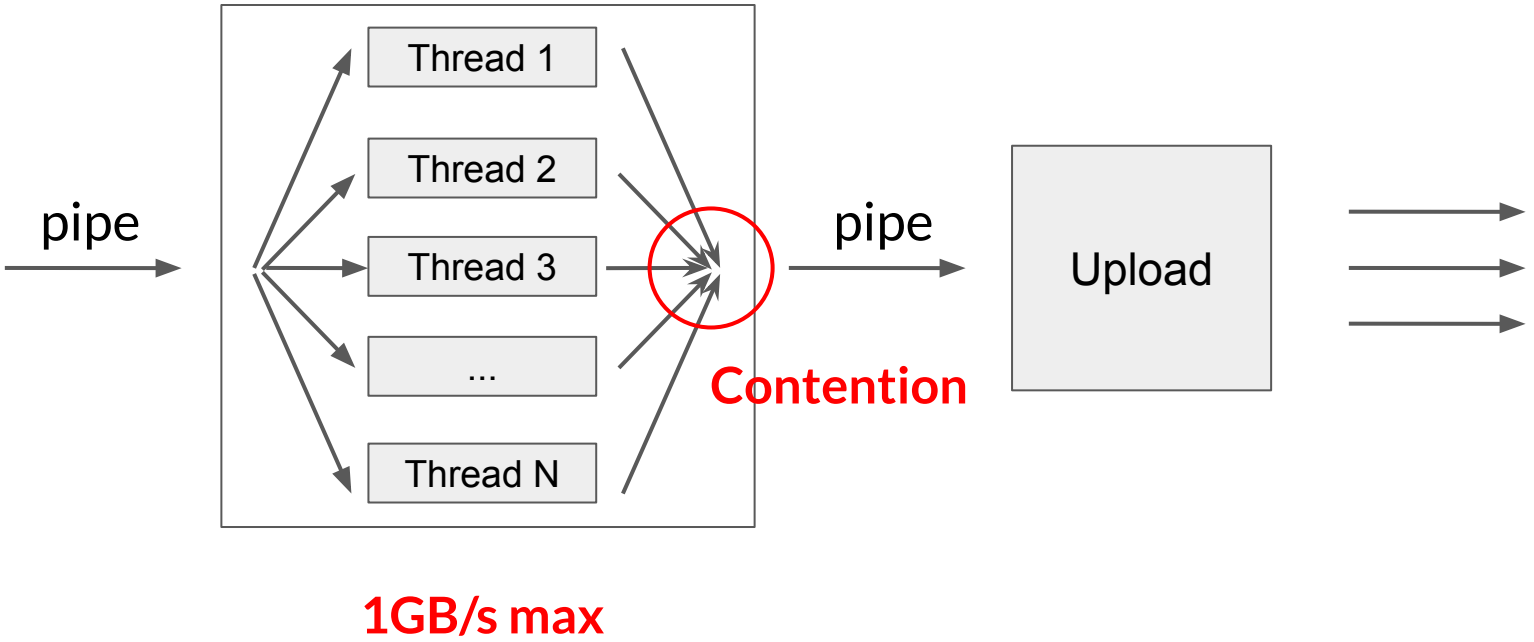


Problem: Compress and upload stream

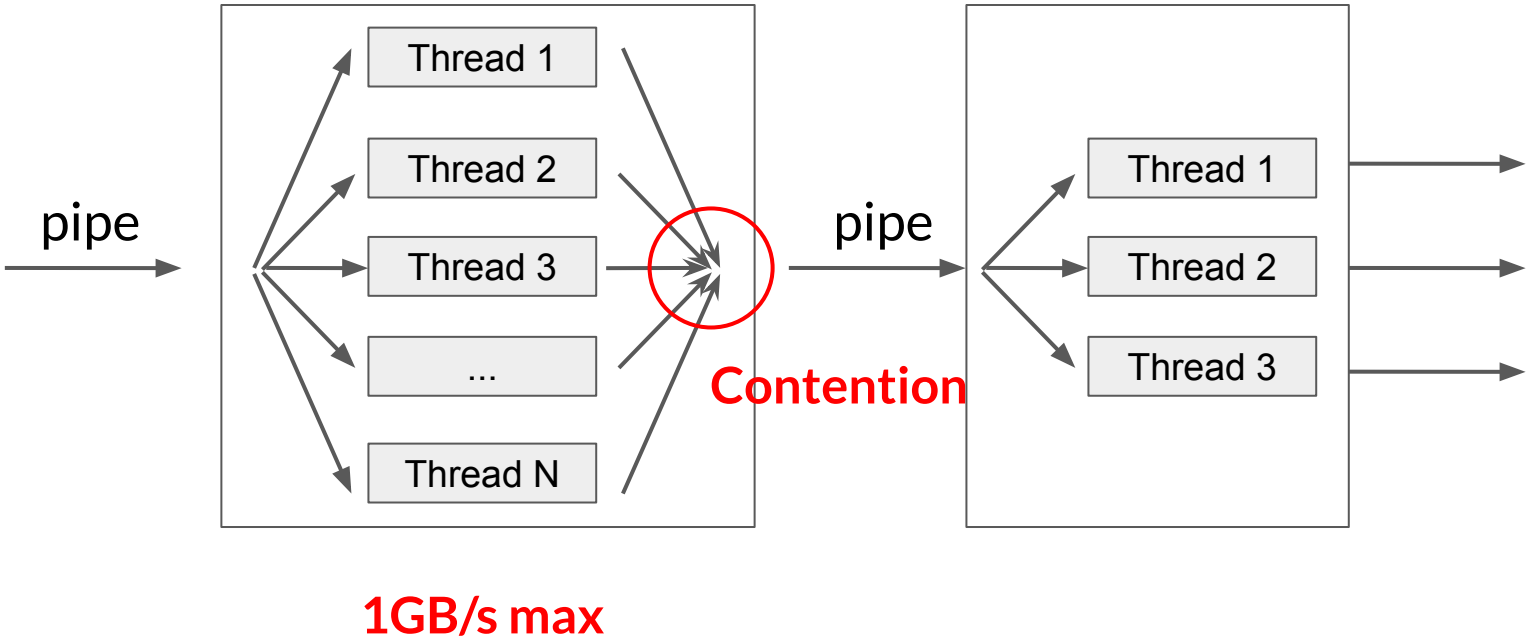


1GB/s max

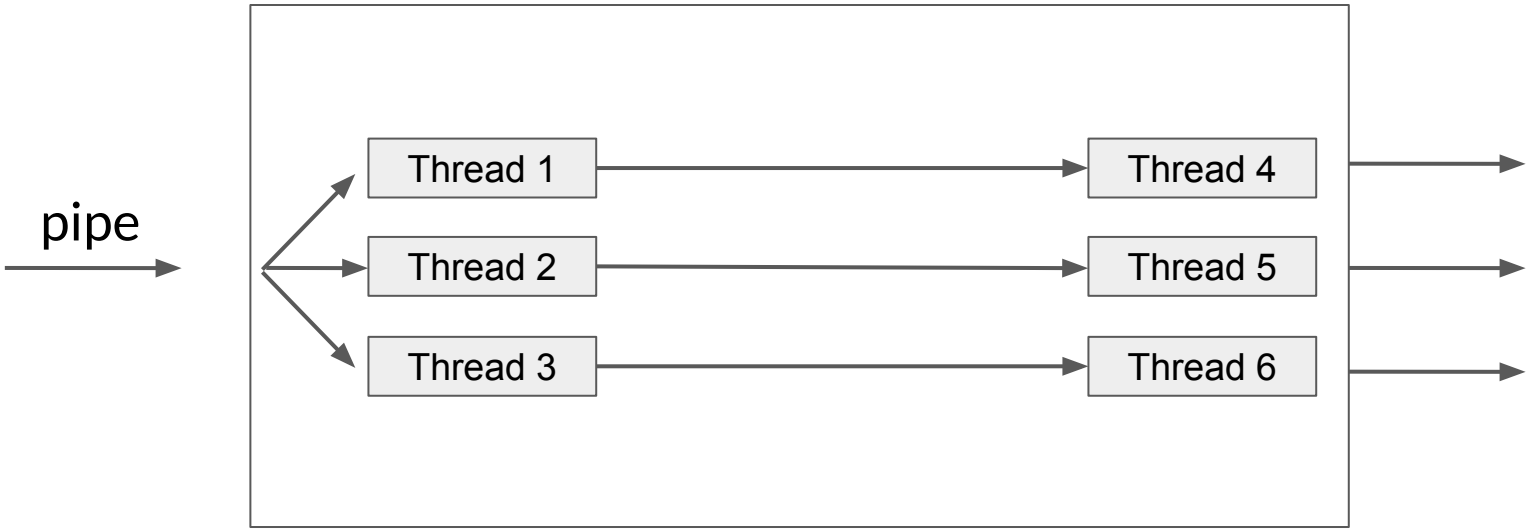
Problem: Compress and upload stream



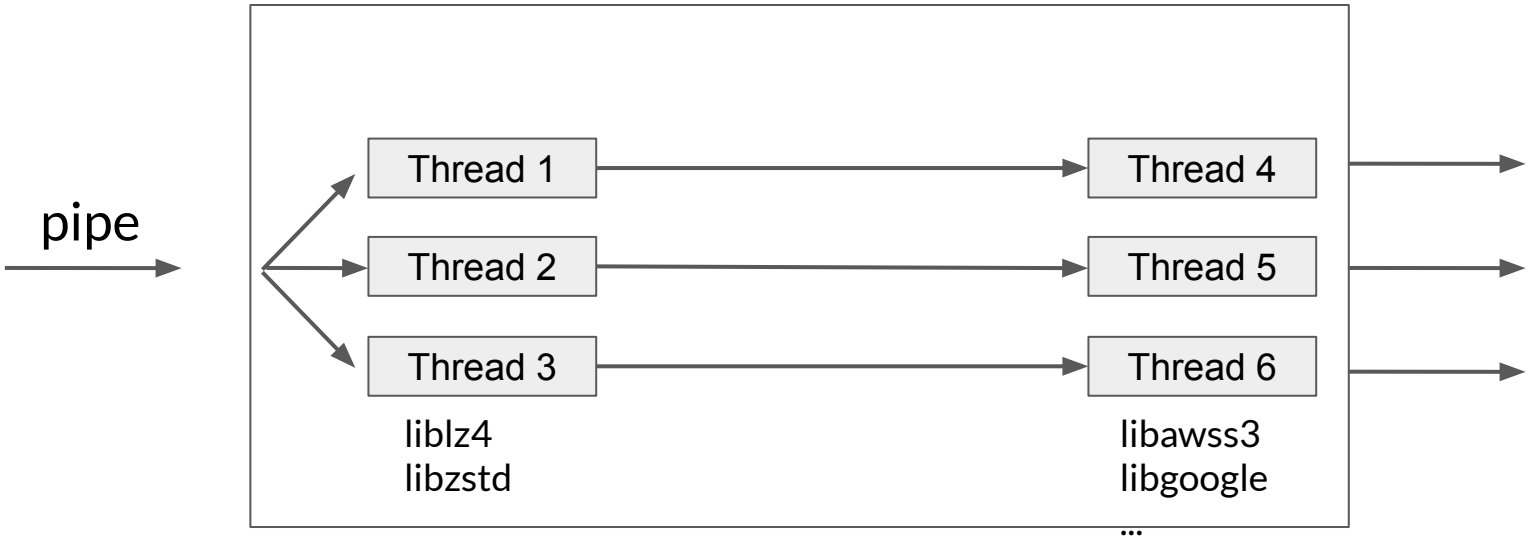
Problem: Compress and upload stream



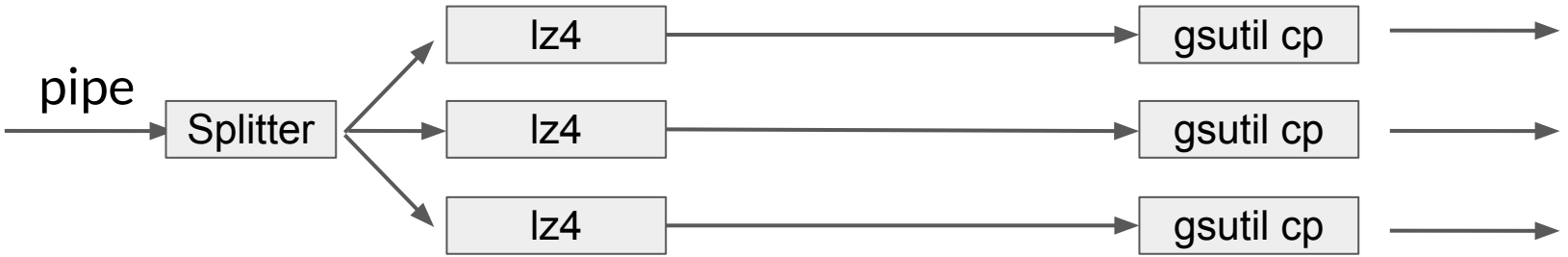
Problem: Compress and upload stream



Problem: Compress and upload stream



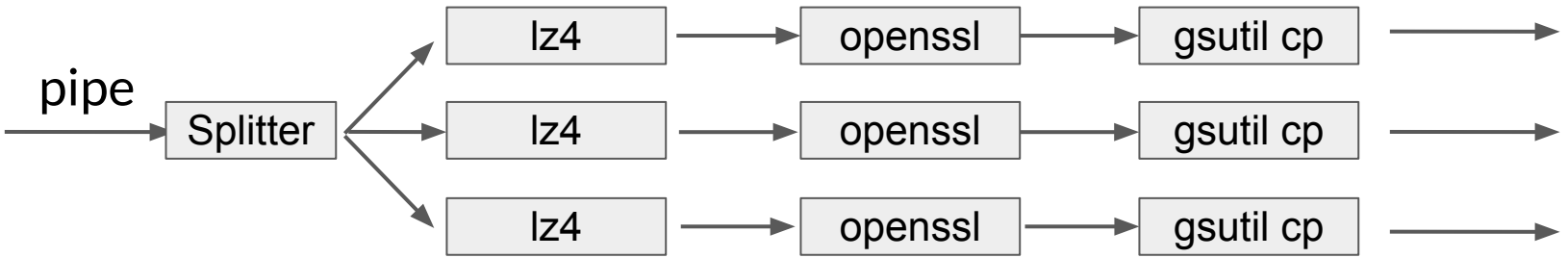
Problem: Compress and upload stream



Shell command

lz4 - - | gsutil cp - gs://bucket/file

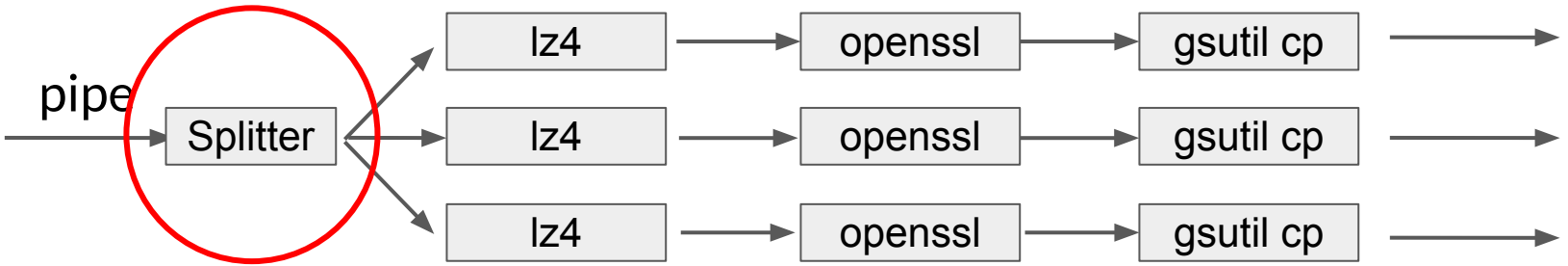
Problem: Compress and upload stream



Shell command

lz4 - - | openssl ... | gsutil cp - gs://bucket/file

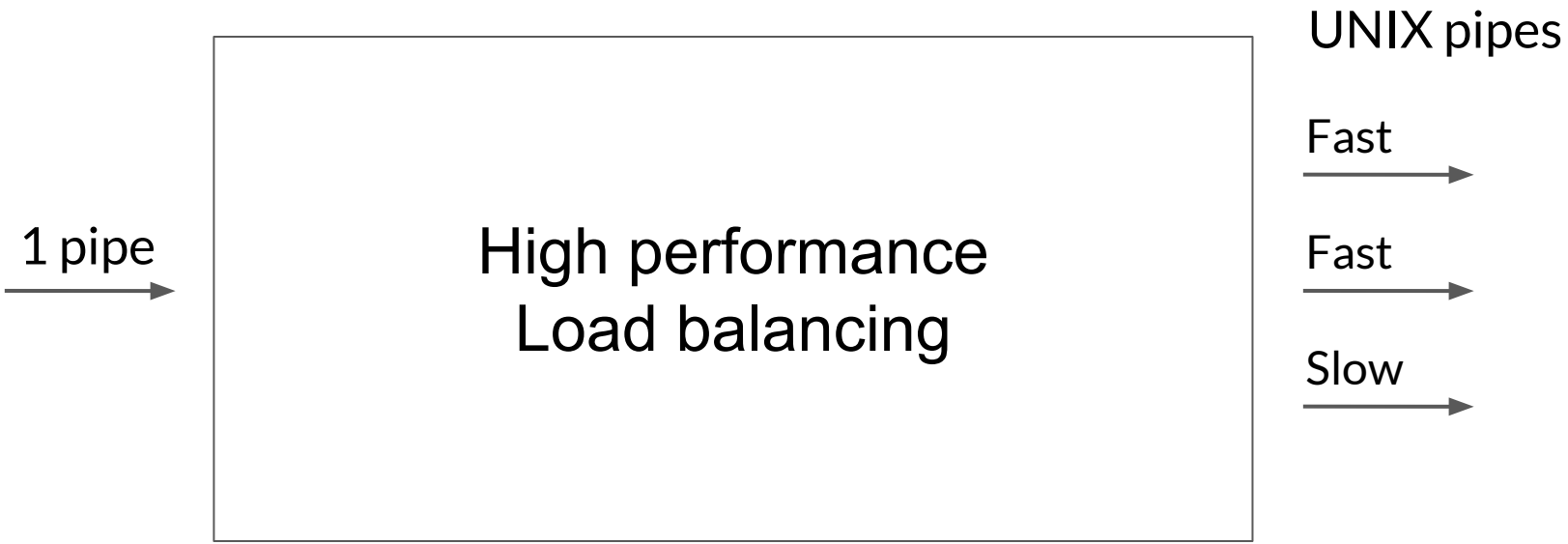
Problem: Compress and upload stream



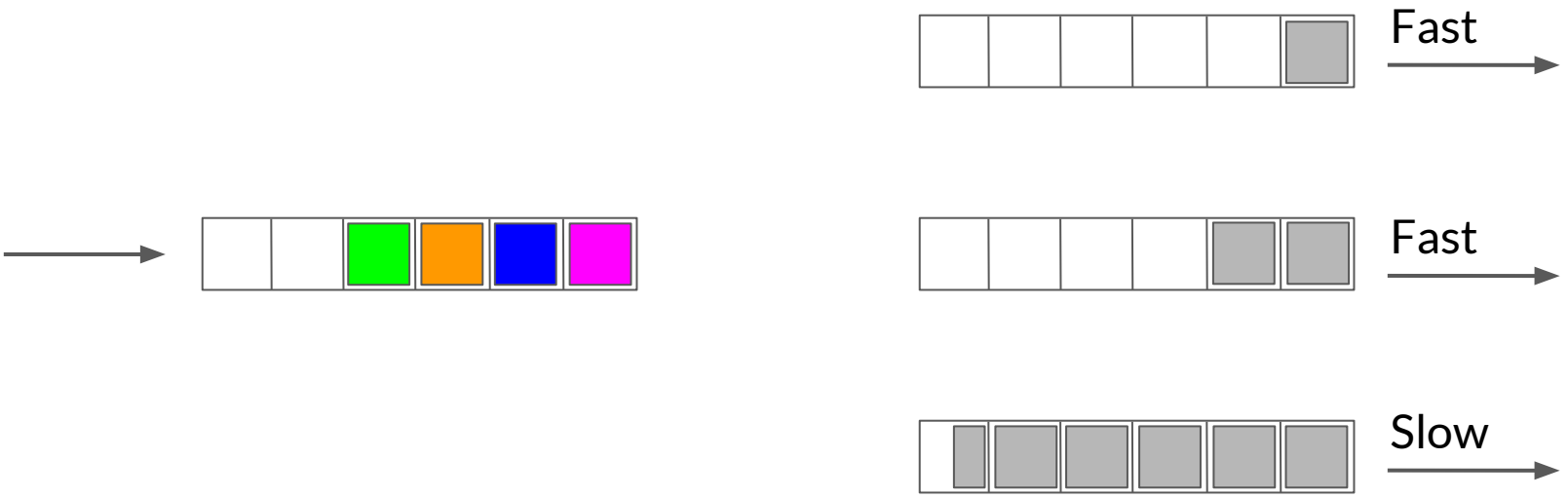
Shell command

lz4 - - | openssl ... | gsutil cp - gs://bucket/file

Problem: Split pipe and load-balance outputs

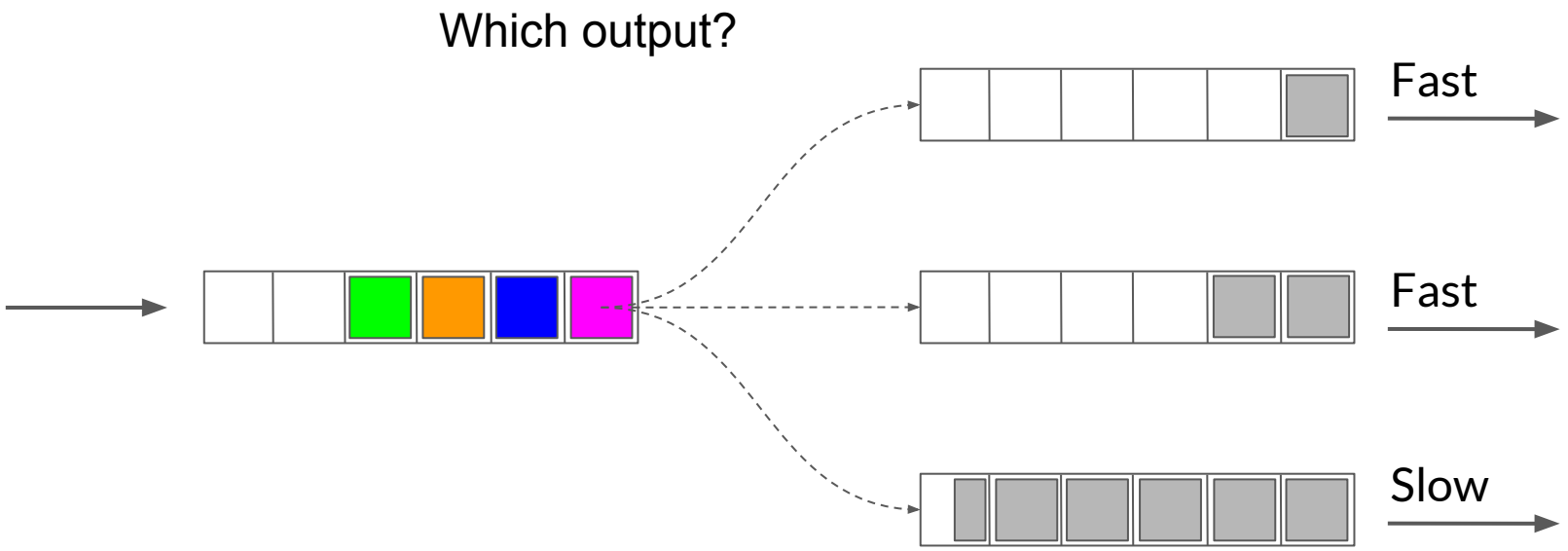


Split pipe and load-balance outputs



Kernel pipe buffers

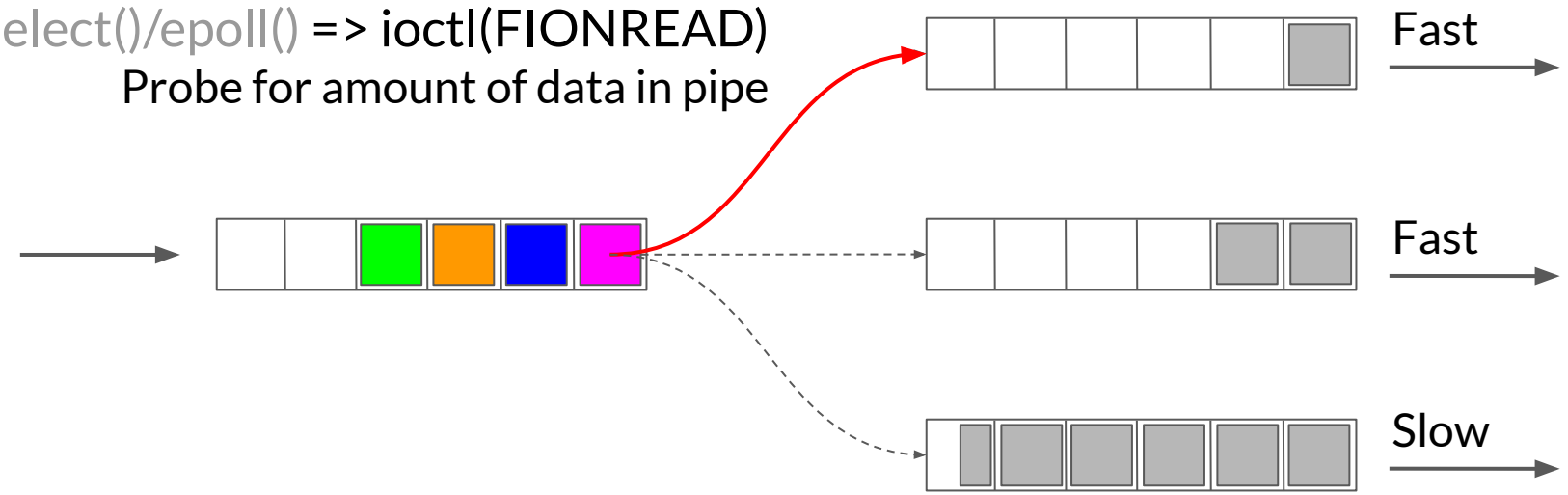
Split pipe and load-balance outputs



Split pipe and load-balance outputs

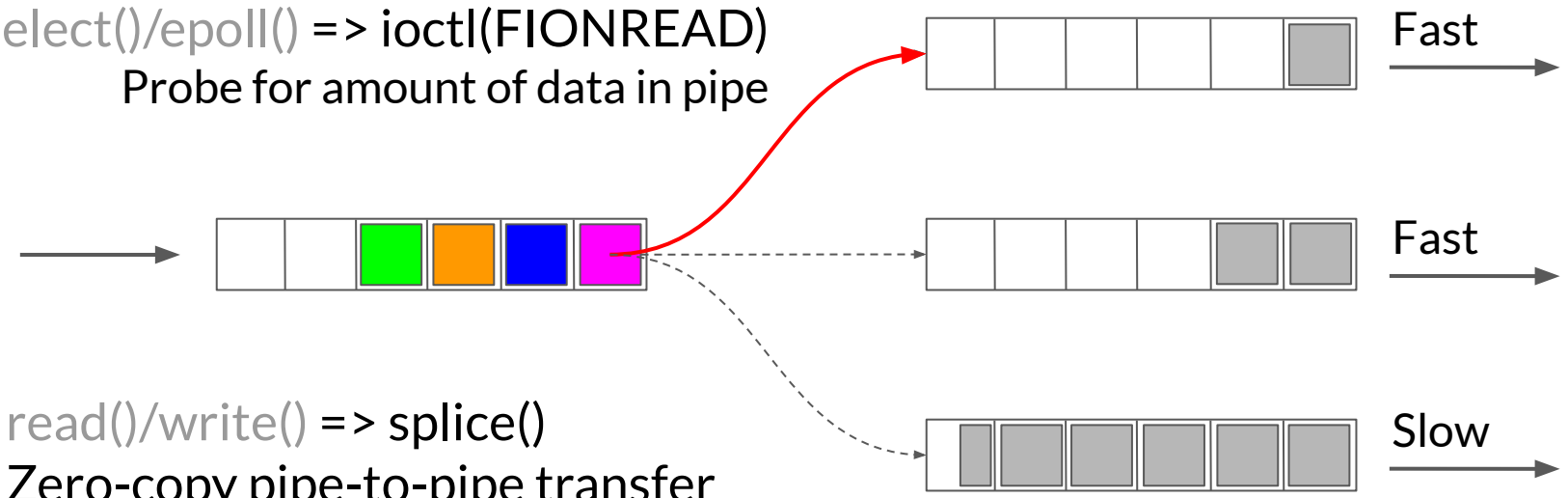
select()/epoll() => ioctl(FIONREAD)

Probe for amount of data in pipe



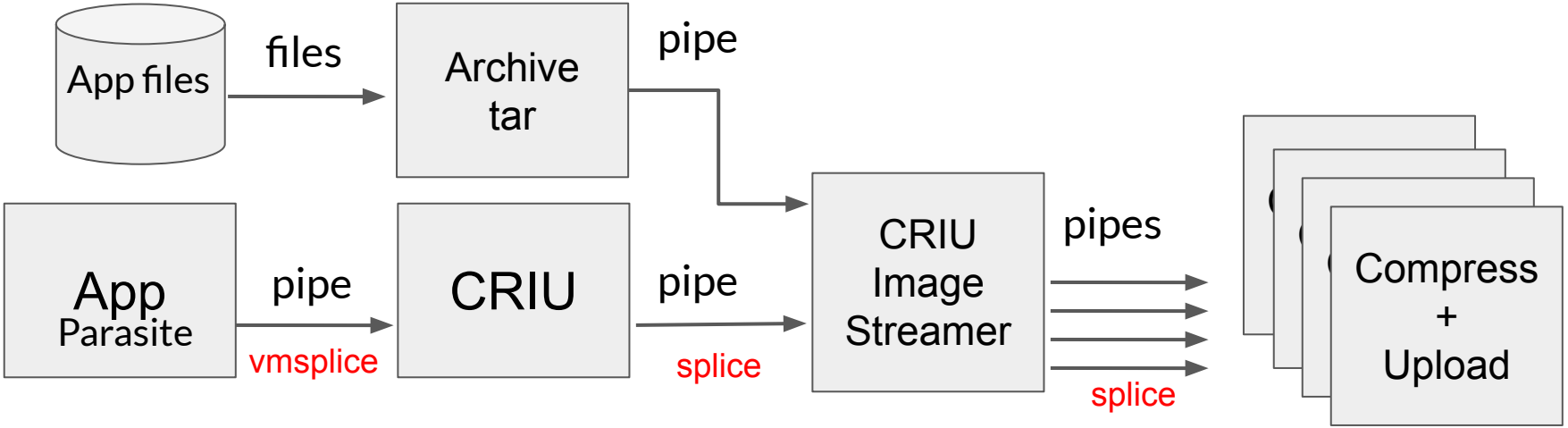
Split pipe and load-balance outputs

select()/epoll() => ioctl(FIONREAD)
Probe for amount of data in pipe

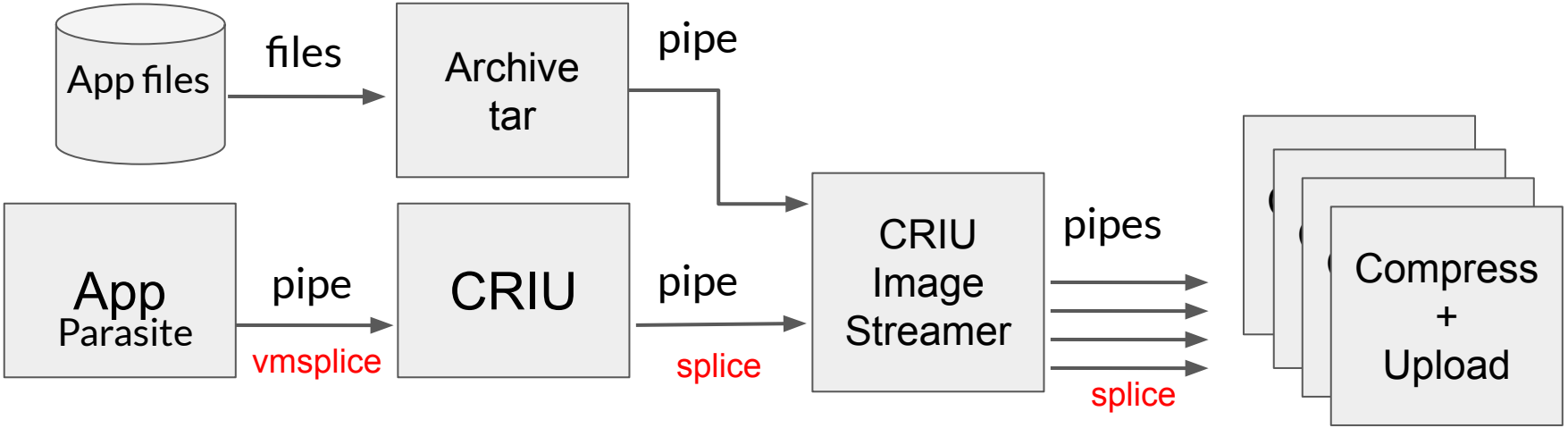


read()/write() => splice()
Zero-copy pipe-to-pipe transfer

Streaming Checkpointing



Streaming Checkpointing



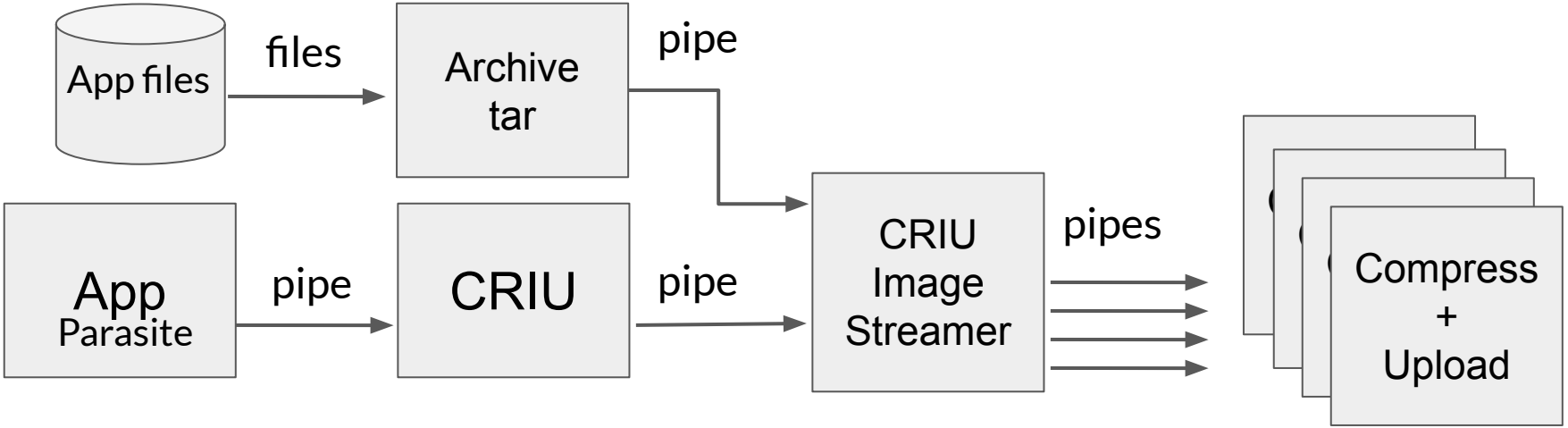
cat > /dev/null

15 GB/s

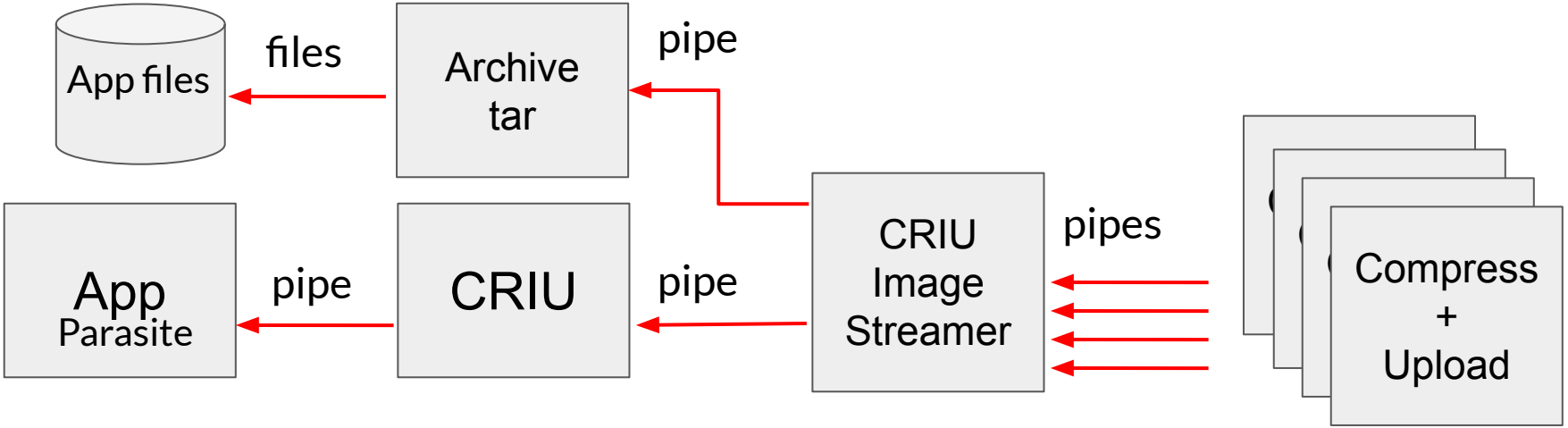
Performance Trade-off: CPU vs Network

Upload Protocol	HTTPS	HTTPS	HTTPS
Compression Algo	**None**	**lz4**	**zstd**
Compression Ratio (typical Java app)	1x	3x	5x
Throughput per CPU (before compression)	550 MB/s	350 MB/s	220 MB/s
Time to checkpoint (16 CPUs, 128GB RAM, 2GB/s NIC)	64s	24s	37s

Streaming Checkpoint



Streaming Restore



Problem: CRIU reads image out-of-order

- Writes `inventory.img` **at the end** of the stream
- Reads `inventory.img` **at the beginning** of the stream

Problem: CRIU reads image out-of-order

- Writes `inventory.img` **at the end** of the stream
- Reads `inventory.img` **at the beginning** of the stream

Solution

- Buffer the entire image in memory
- Let CRIU access image in arbitrary order
- 2x memory problem solved by deallocating after transferring data to CRIU

Checkpoint example, it’s Unix

```
exec 10> >(lz4 - - | gsutil cp - gs://bucket/img-1.lz4)
exec 11> >(lz4 - - | gsutil cp - gs://bucket/img-2.lz4)

exec 20< <(tar -C / -cpf - /scratch/app)

criu-image-streamer \
    --shard-fds 10,11 --ext-file-fds fs.tar:20 \
    --images-dir /tmp capture &
criu dump --images-dir /tmp --tree $APP_PID --stream
```

Live Migration

```
criu-image-streamer capture | ssh remote criu-image-streamer serve
```

Acknowledgements

- Two Sigma
  - Vitaly Davidovich
  - Peter Burka
- CRIU team
  - Andrei Vagin, Google
  - Radostin Stoyanov, Cambridge
  - Mike Rapoport, IBM

Future Work

- Speed-up restore: zero-copy
- Support pre-dump
- Perform dedup
- Adaptive compression
- Add kernel pipe statistics

Conclusion

- We can checkpoint at 15GB/s
- Unix philosophy with performance
- `ioctl(FIONREAD)` and `splice` for zero-copy load-balancing

https://github.com/checkpoint-restore/criu-image-streamer

Nicolas.Viennot@twosigma.com