

# Eidetic Systems

**David Devecsery**, Michael Chow, Xianzheng Dou,  
Jason Flinn, Peter Chen  
University of Michigan

# What is an Eidetic System?

*Eidetic* – Having “Perfect memory” or “Total Recall”

*Eidetic System* – A system which can recall and trace through the lineage of any past computation



# Motivation - Heartbleed

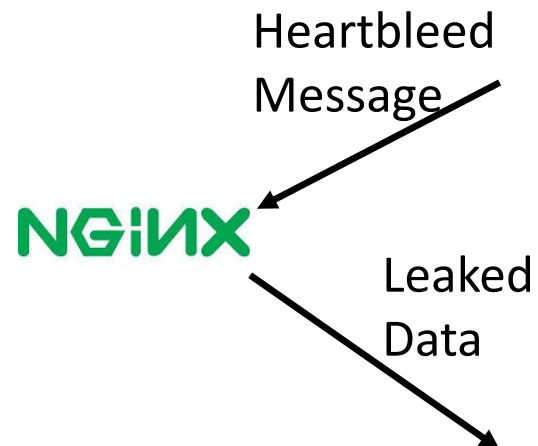


NGINX

- Was Heartbleed exploited?
- What data was leaked?



# Motivation - Heartbleed



- Was Heartbleed exploited? - Yes
- What data was leaked?

# Motivation - Heartbleed



- Was Heartbleed exploited? - Yes
- What data was leaked?

# Motivation – Wrong Reference

## Bad Citation

John Aiken's kit is used to present a gateway. A comprehensive literature of these work is here to present or direct computing with Aiken's kit [1].

**8. Acknowledgments**

We thank the reviewers and our department, Levent Akinci, for their insightful comments. The material is based upon work supported by the National Science Foundation under grant number CNS-09-041445. The Huang visited data and code that we adapted for this work. Our sponsors, funding, contributions, or recommendations expressed in this volume are those of the authors and do not necessarily reflect those of the National Science Foundation.

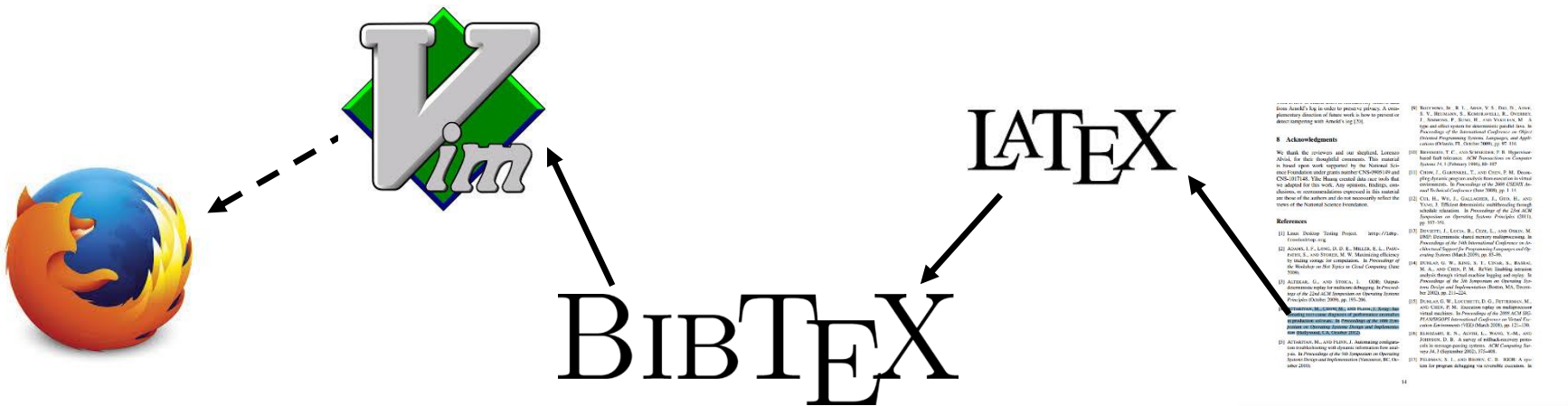
**References**

- [1] Luis Abuelo. "Using Project ...".
- [2] John A. L. Lohr, D. G. W. Miller, ...
- [3] ...
- [4] ...
- [5] ...
- [6] ...
- [7] ...
- [8] ...
- [9] ...
- [10] ...
- [11] ...
- [12] ...
- [13] ...
- [14] ...
- [15] ...
- [16] ...
- [17] ...
- [18] ...
- [19] ...
- [20] ...
- [21] ...
- [22] ...
- [23] ...
- [24] ...
- [25] ...
- [26] ...
- [27] ...
- [28] ...
- [29] ...
- [30] ...
- [31] ...
- [32] ...
- [33] ...
- [34] ...
- [35] ...
- [36] ...
- [37] ...
- [38] ...
- [39] ...
- [40] ...
- [41] ...
- [42] ...
- [43] ...
- [44] ...
- [45] ...
- [46] ...
- [47] ...
- [48] ...
- [49] ...
- [50] ...
- [51] ...
- [52] ...
- [53] ...
- [54] ...
- [55] ...
- [56] ...
- [57] ...
- [58] ...
- [59] ...
- [60] ...
- [61] ...
- [62] ...
- [63] ...
- [64] ...
- [65] ...
- [66] ...
- [67] ...
- [68] ...
- [69] ...
- [70] ...
- [71] ...
- [72] ...
- [73] ...
- [74] ...
- [75] ...
- [76] ...
- [77] ...
- [78] ...
- [79] ...
- [80] ...
- [81] ...
- [82] ...
- [83] ...
- [84] ...
- [85] ...
- [86] ...
- [87] ...
- [88] ...
- [89] ...
- [90] ...
- [91] ...
- [92] ...
- [93] ...
- [94] ...
- [95] ...
- [96] ...
- [97] ...
- [98] ...
- [99] ...
- [100] ...

- How did I get the wrong citation?



# Motivation – Wrong Reference



- How did I get the wrong citation?

# Motivation – Wrong Reference

**X-ray: automating root-cause diagnosis of performance anomalies in production software**

Authors: Mehmet Altarun, Joseph D. McKee, and Ganhui He  
Michael Chow, James H. Kukula, Joseph Chen, James H. Kukula

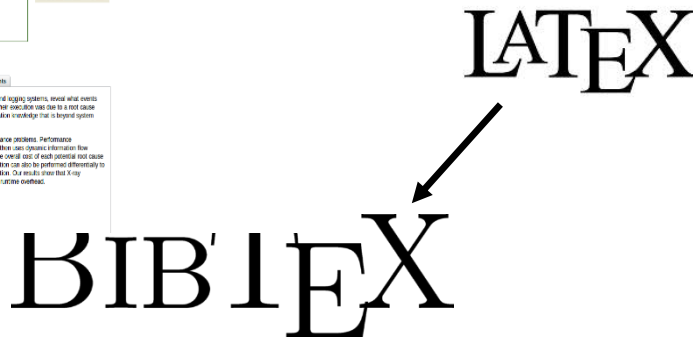
Published in: Proceedings of the 2012 USENIX conference on Operating Systems Design and Implementation (OSDI '12)

Pages: 367-380

ISBN: 978-1-4555-2456-4

DOI: 10.1145/2187191.2187191

This paper introduces performance summarization, a technique for automatically diagnosing the root causes of performance problems. Performance summarization reduces performance anomalies to a small number of root causes. It first attributes performance costs to each basic block. It then uses dynamic information flow tracking to estimate the likelihood of a basic block being the root cause. Finally, it summarizes the overall cost of each potential root cause by summing the per-block cost multiplied by the case-specific likelihood over all basic blocks. Performance summarization can also be performed differentially to explain performance differences between two similar applications. X-ray is a tool that implements performance summarization. Our study shows that X-ray accurately diagnoses 17 performance issues in Apache, lighttpd, Firefox, and PostgreSQL, while adding 2.3% average runtime overhead.

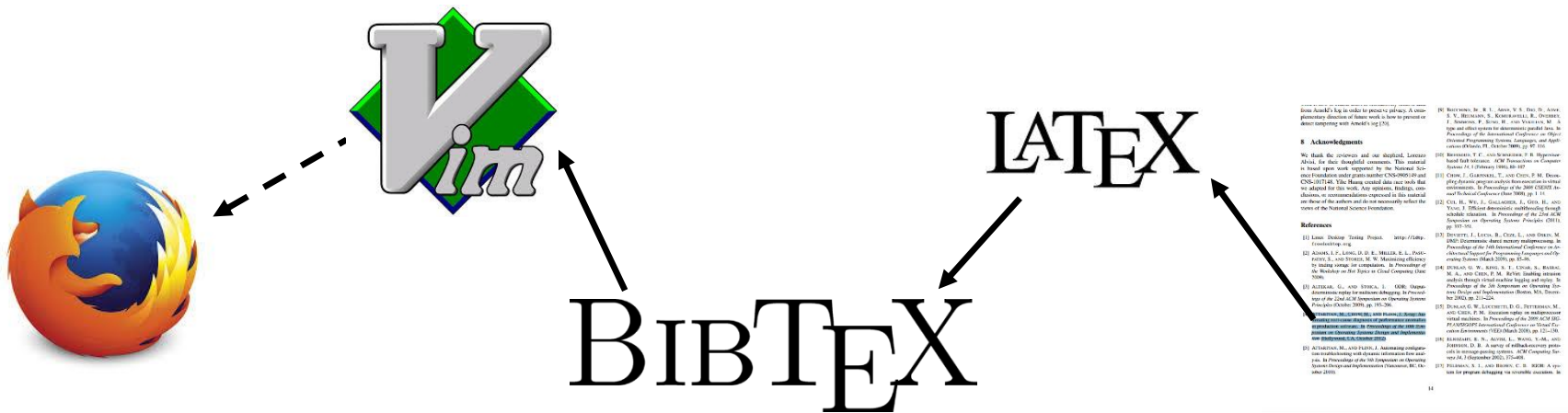


- How did I get the wrong citation?





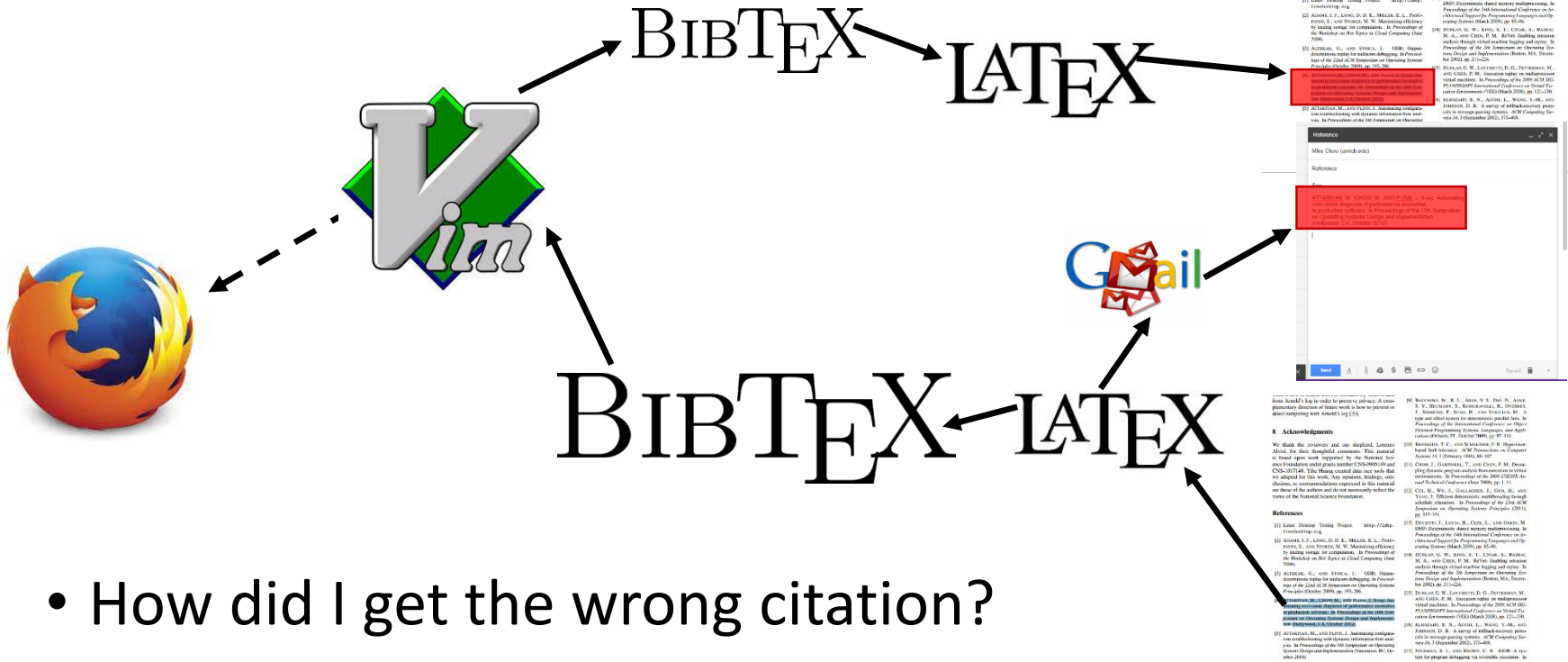
# Motivation – Wrong Reference



- How did I get the wrong citation?
- What else did this affect?



# Motivation



- How did I get the wrong citation?
- What else did this affect?



# Arnold

- First practical eidetic computer system
  - Efficiently records & recalls all user-space computation
    - Process register/memory state
    - Inter-process communication
  - Handles lineage queries
    - What data was affected?
    - What states and outputs were affected?
- Targeted towards desktop/workstation use
- Reasonable overheads
  - Record 4 years of data on \$150 commodity HD
  - Under 8% performance overhead on most benchmarks



# Overview

- Introduction
- Motivation
- How Arnold remembers all state
- How Arnold supports lineage queries
- Conclusion

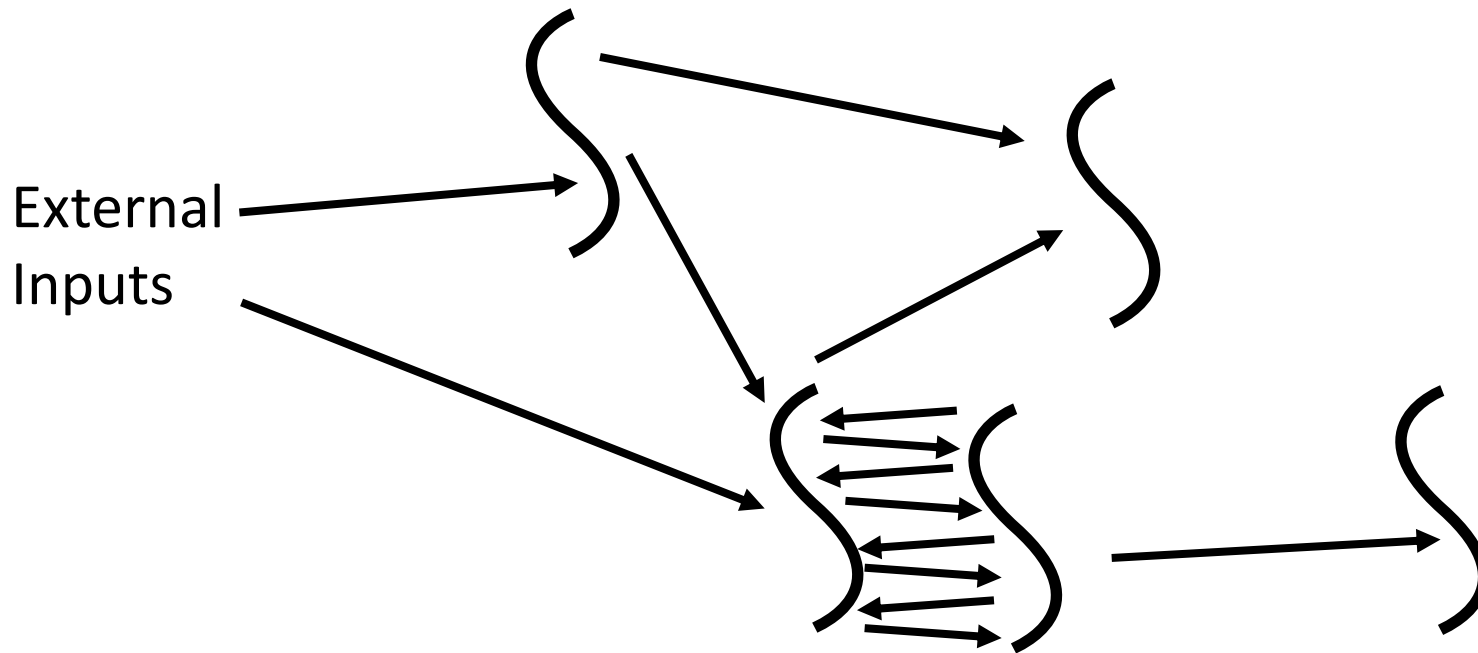


# Remembering State

- Requirements:
  - Store years of state on a single disk
    - Memory/register space within a process
    - Inter process communication
    - File state
  - Recall any state in reasonable time
- Solution:
  - Deterministic record & replay
    - “Process group” based replay
    - “Process graph” to track inter-process lineage
  - Log compression

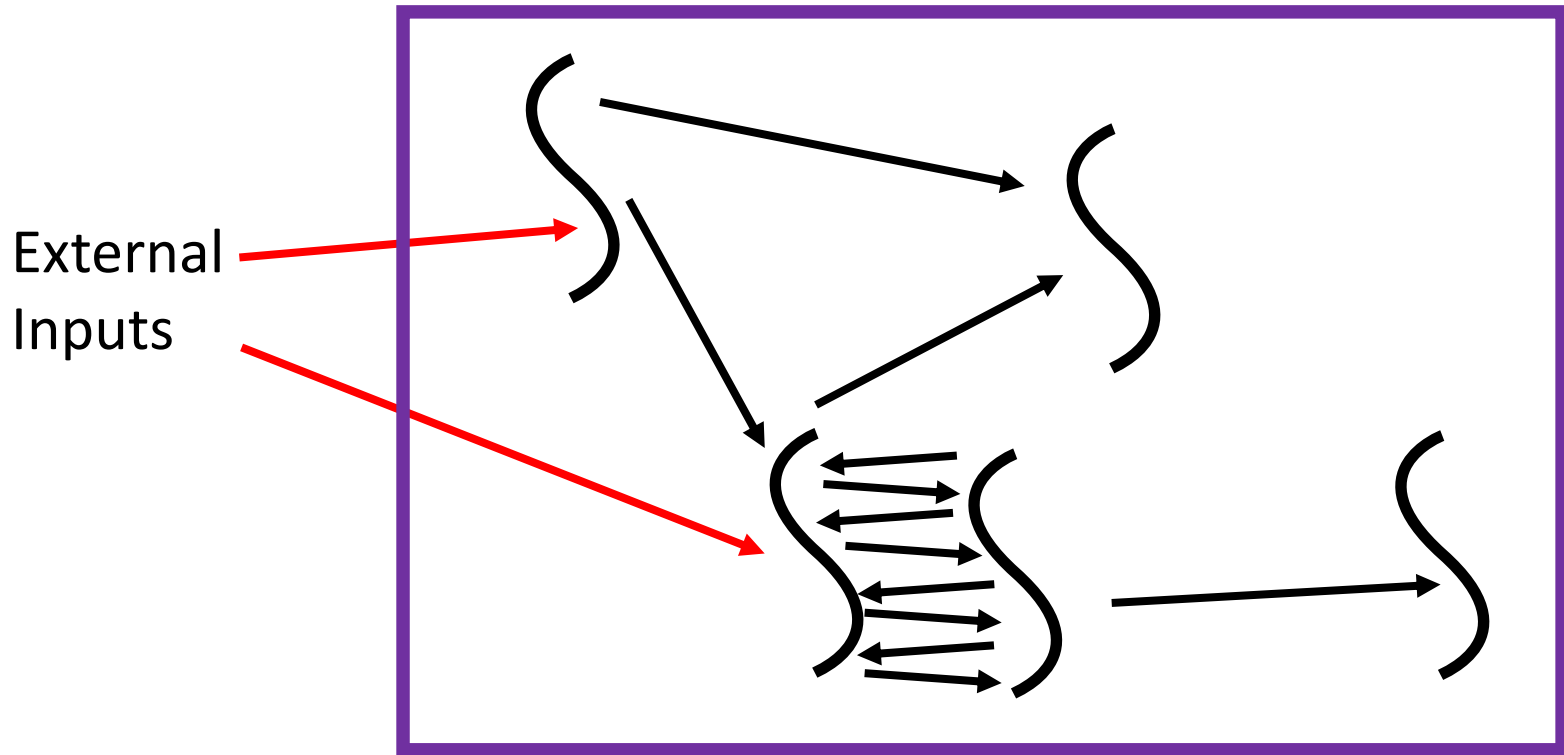


# Recording Granularity



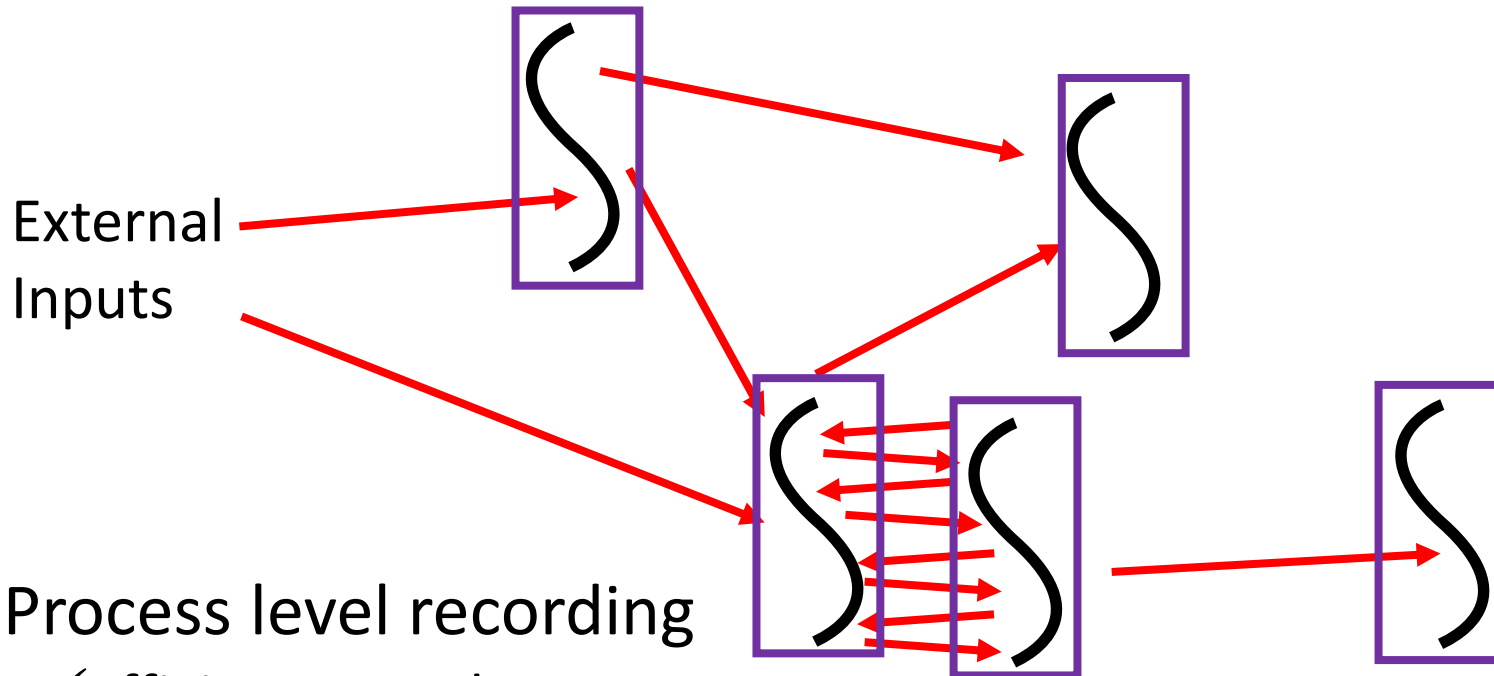
- What granularity is best to record our system?

# Recording Granularity



- Whole system recording
  - ✓ Low space overhead
  - × Costly to replay

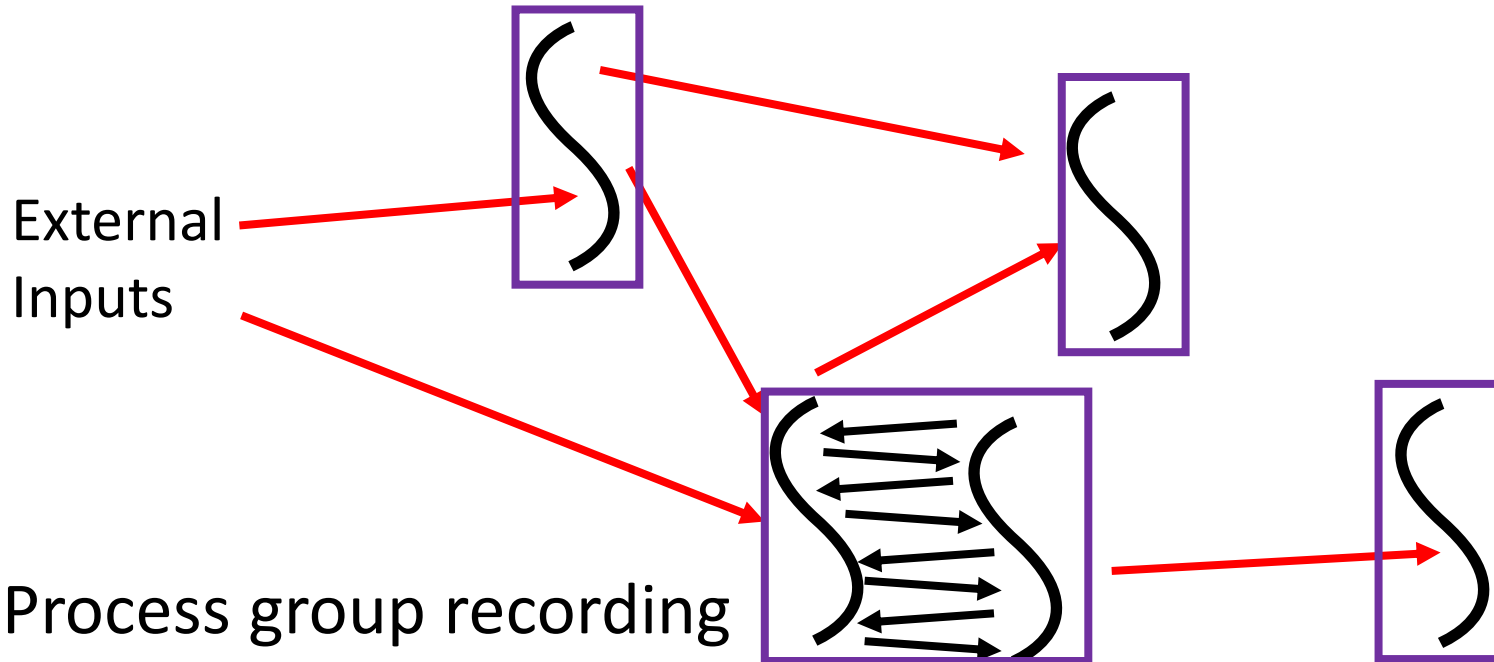
# Recording Granularity



- Process level recording
  - ✓ Efficient to replay
  - × Uses extra disk space
  - × No Inter-process tracking

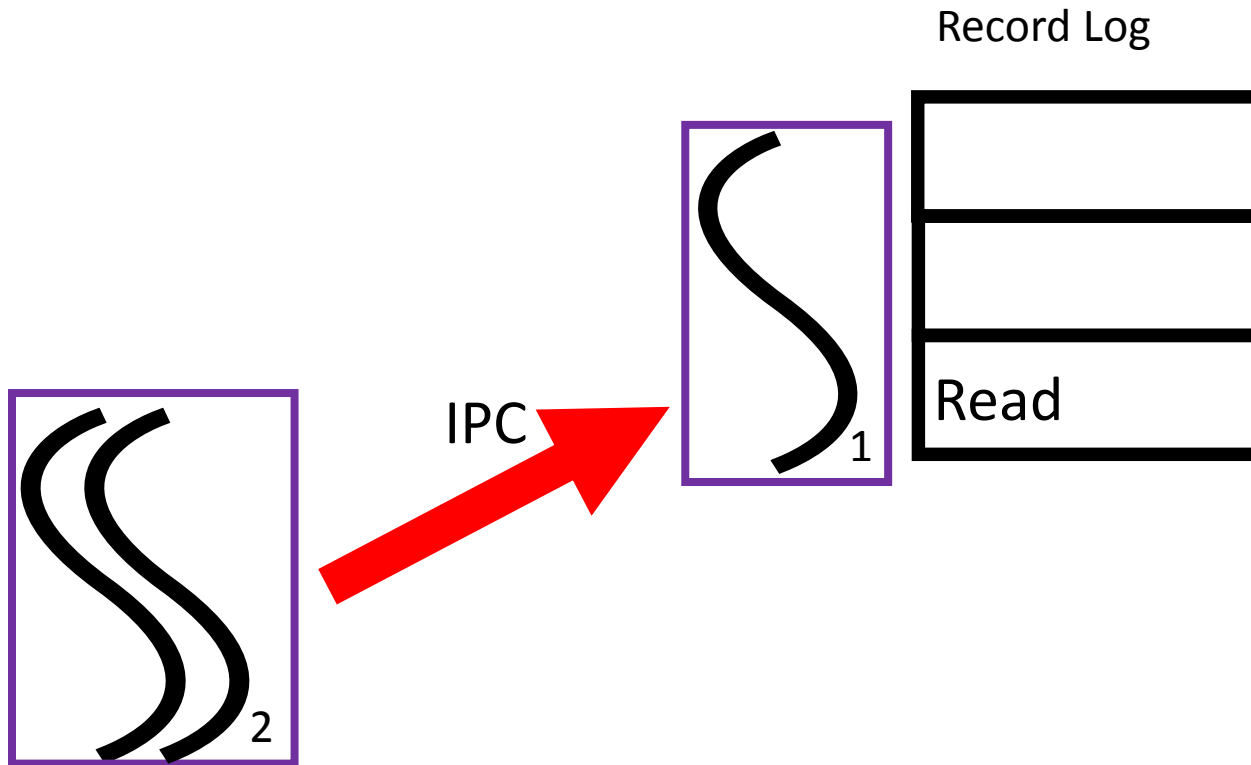


# Recording Granularity

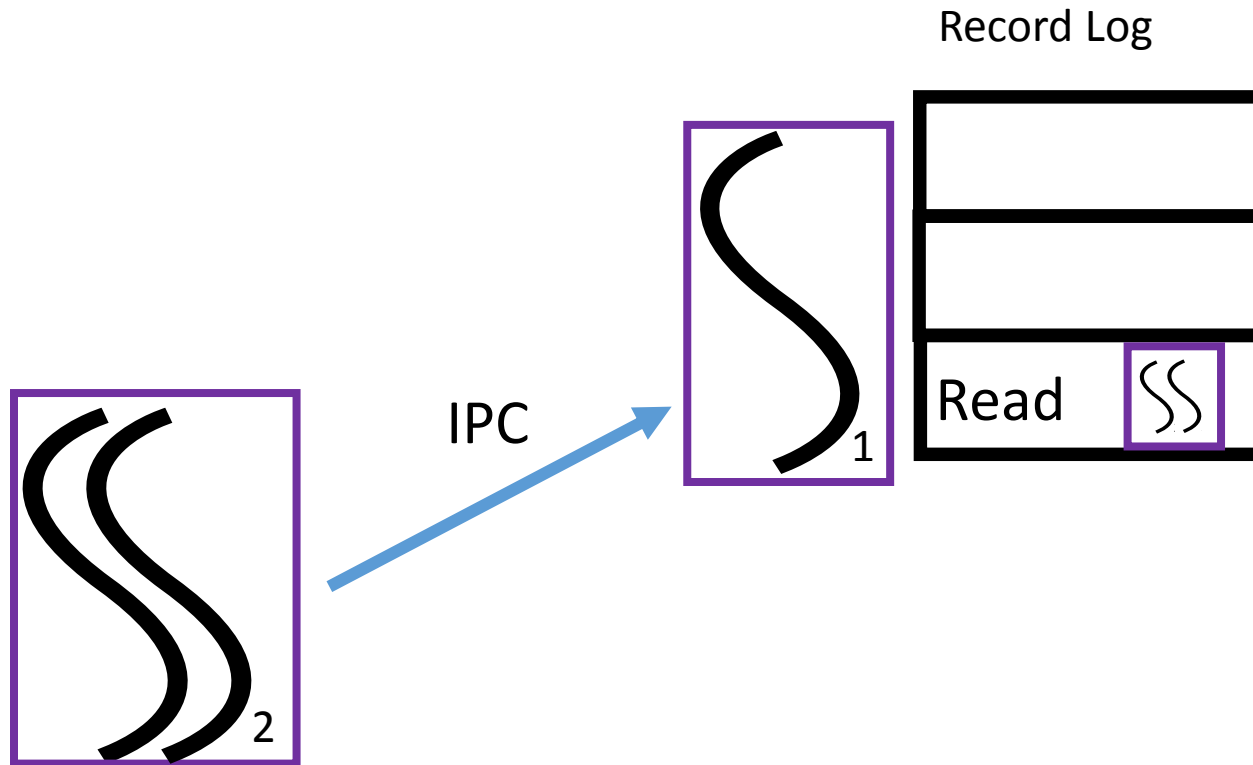


- Process group recording
  - ✓ Efficient to replay
  - ✓ Reasonable disk space
  - × No Inter-process tracking

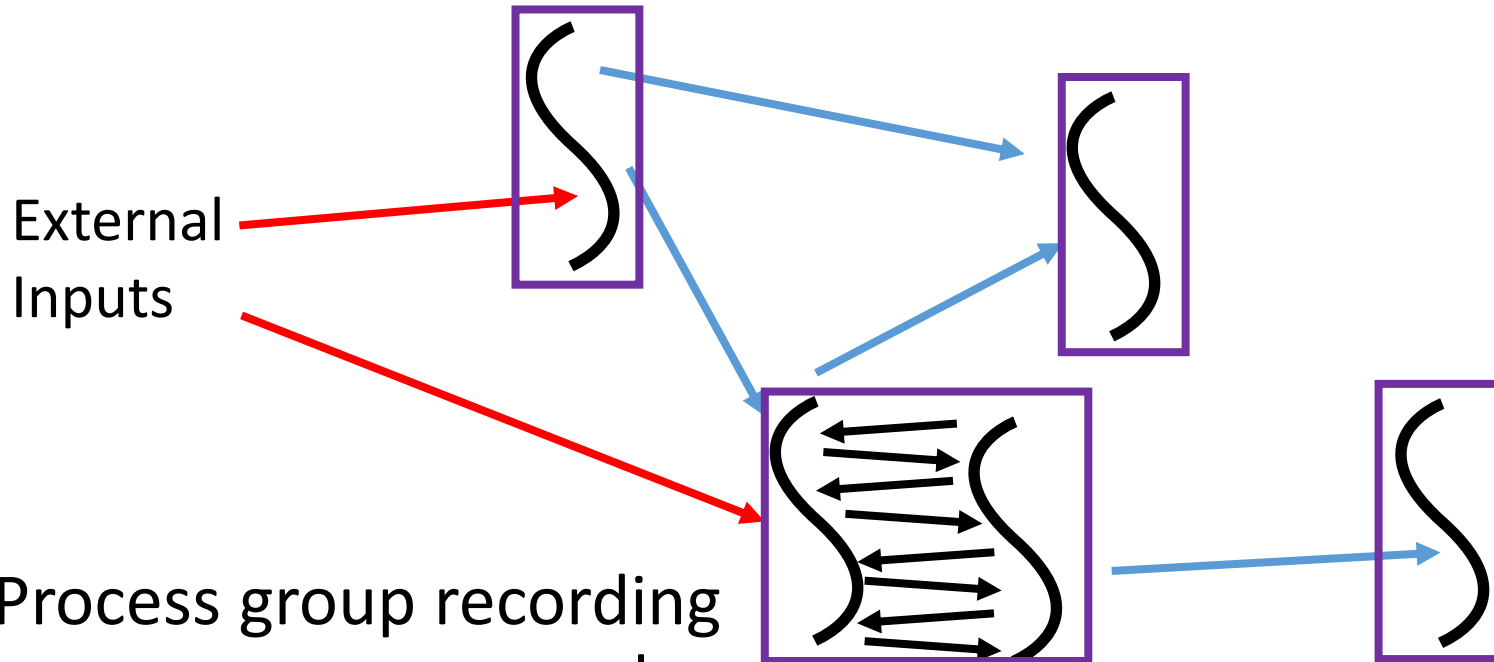
# Implementation – Process Graph



# Implementation – Process Graph



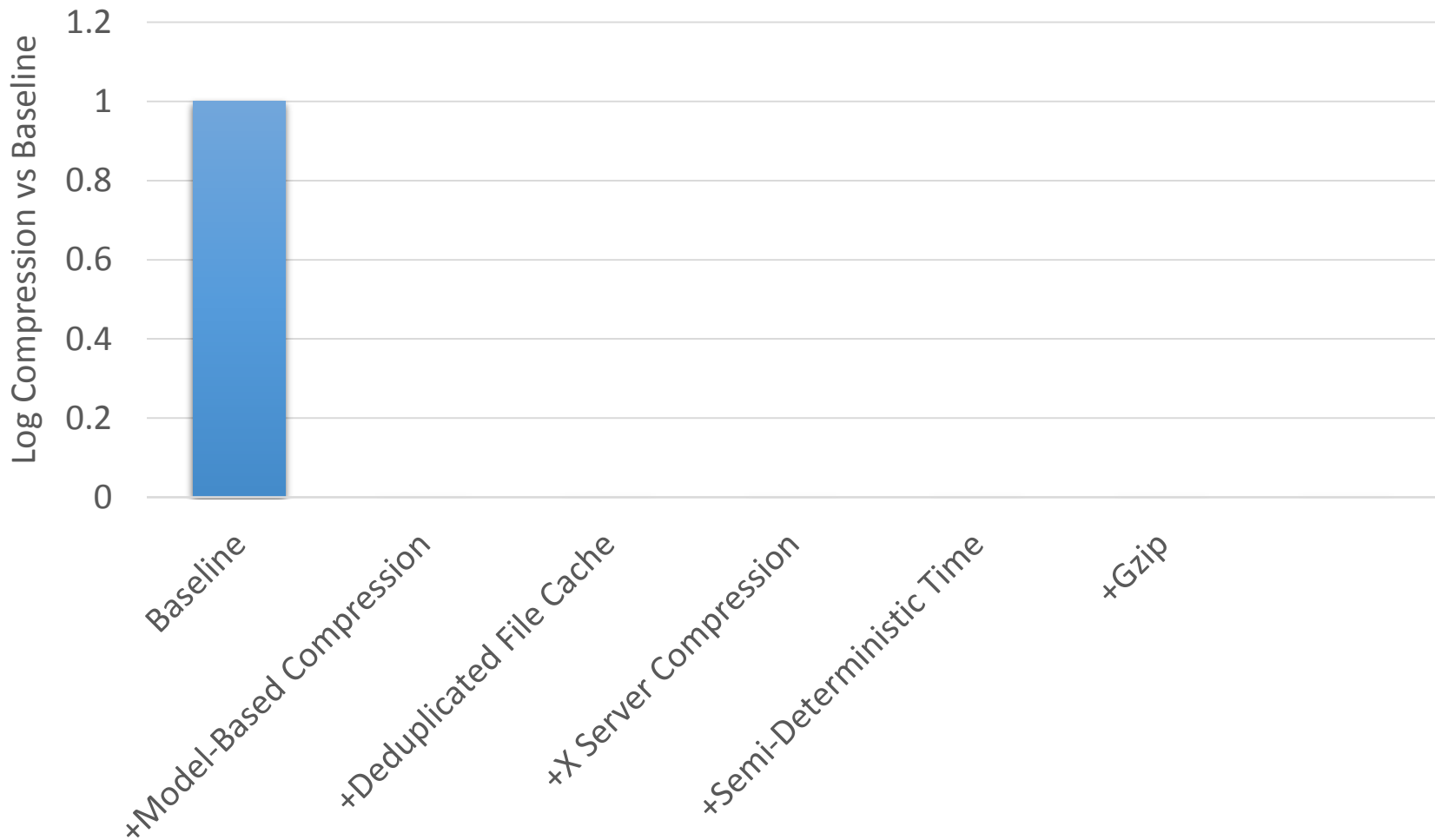
# Recording



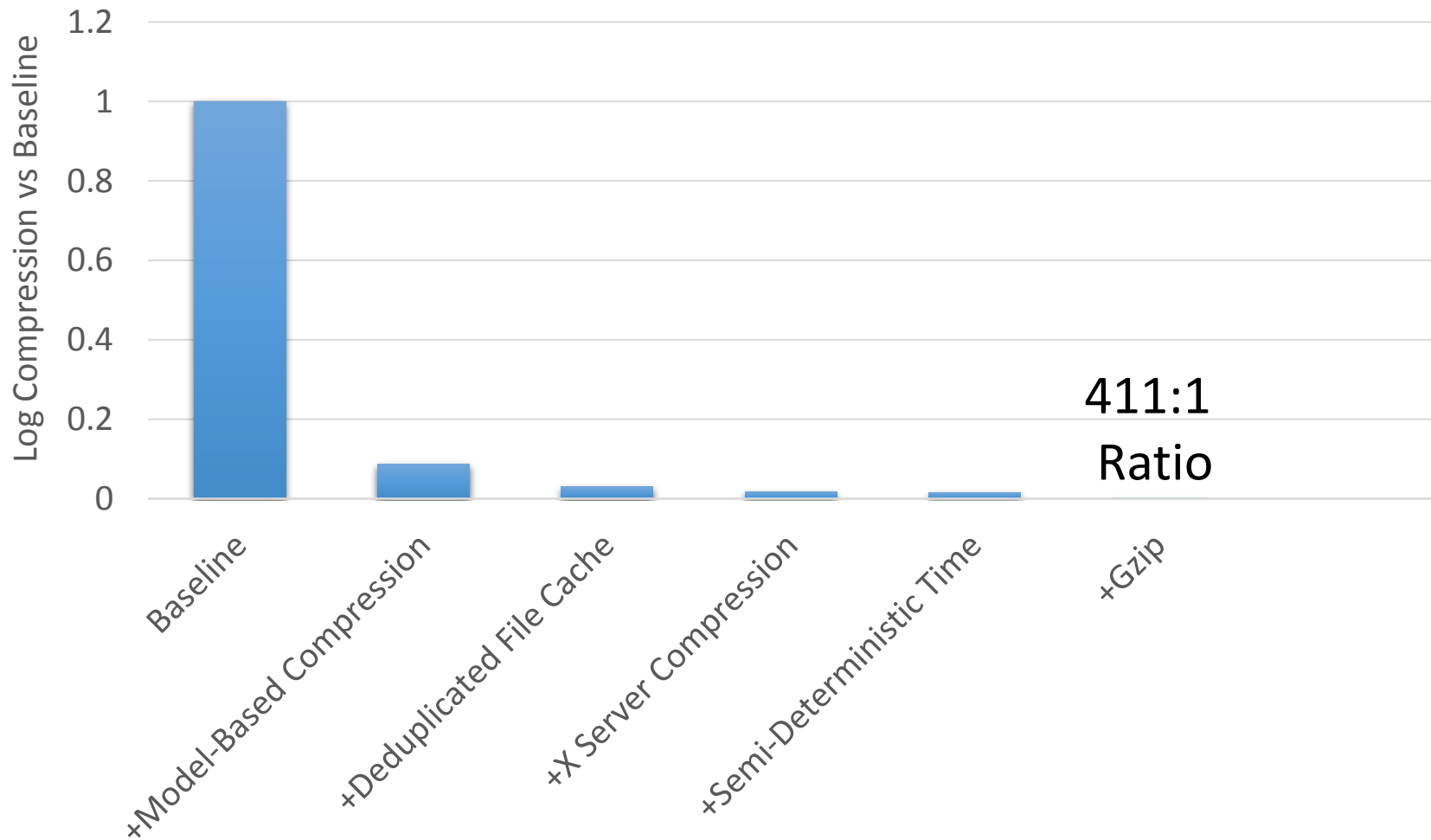
- Process group recording + process graph
  - ✓ Efficient to replay
  - ✓ Reasonable disk space
  - ✓ Inter-process tracking



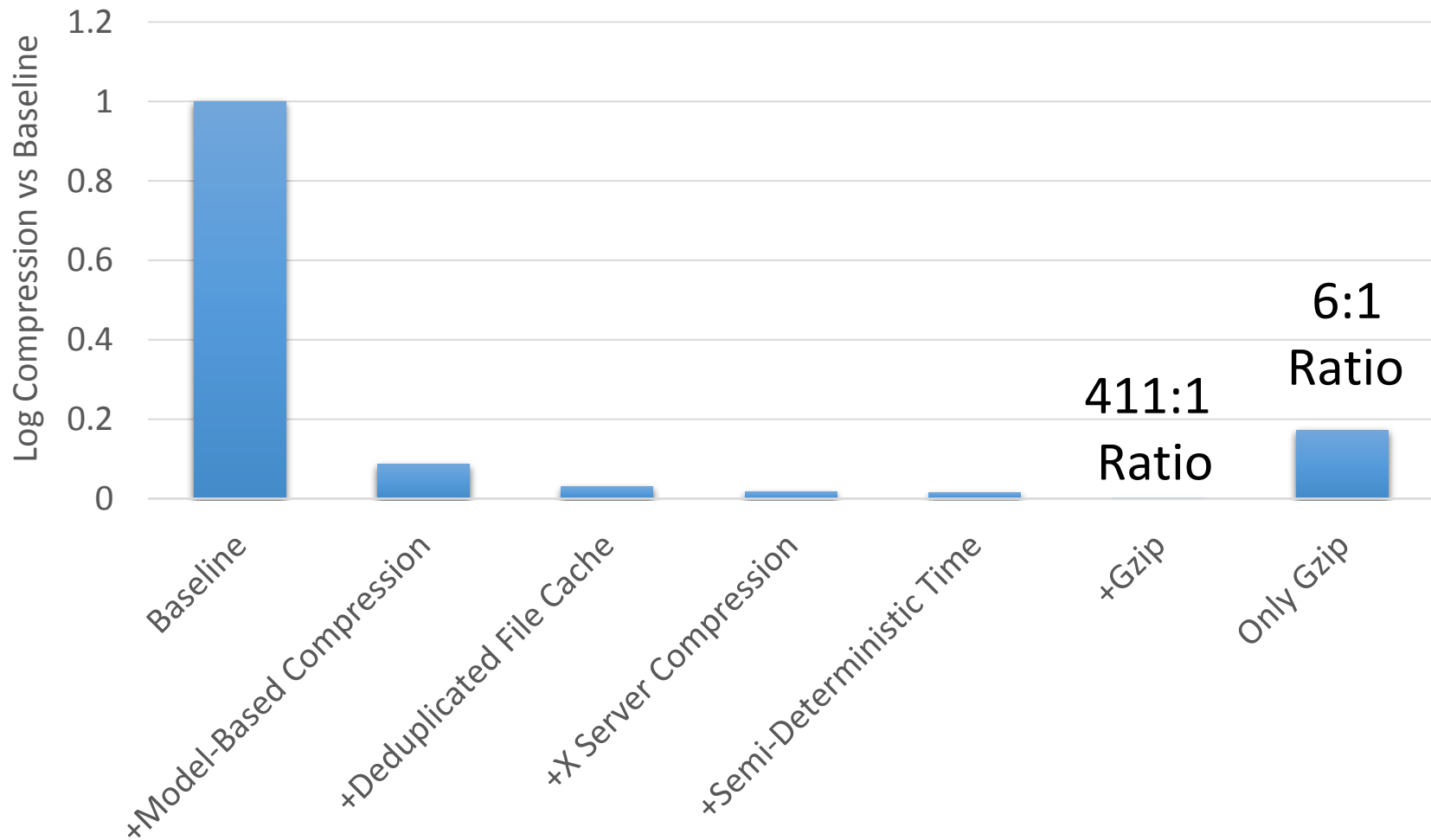
# Space Optimizations



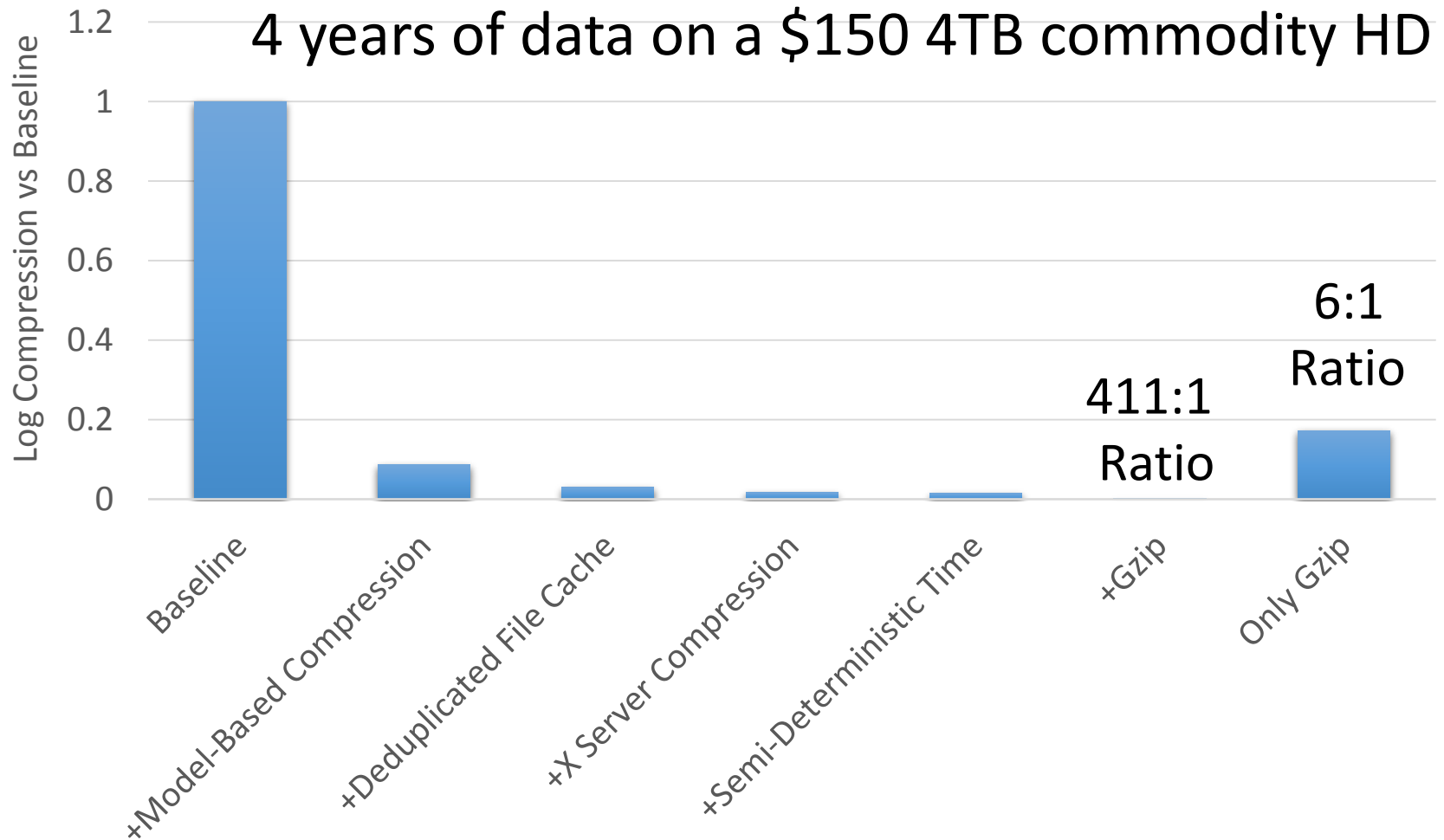
# Space Optimizations



# Space Optimizations



# Space Optimizations





# Model-Based Compression

- Formulate a model of a typical execution

- Only record deviations from that model

```
ret_val = sys_read (fd, buffer, count);
```



- Idea: Partial determinism

- Encourage the program to conform to the model



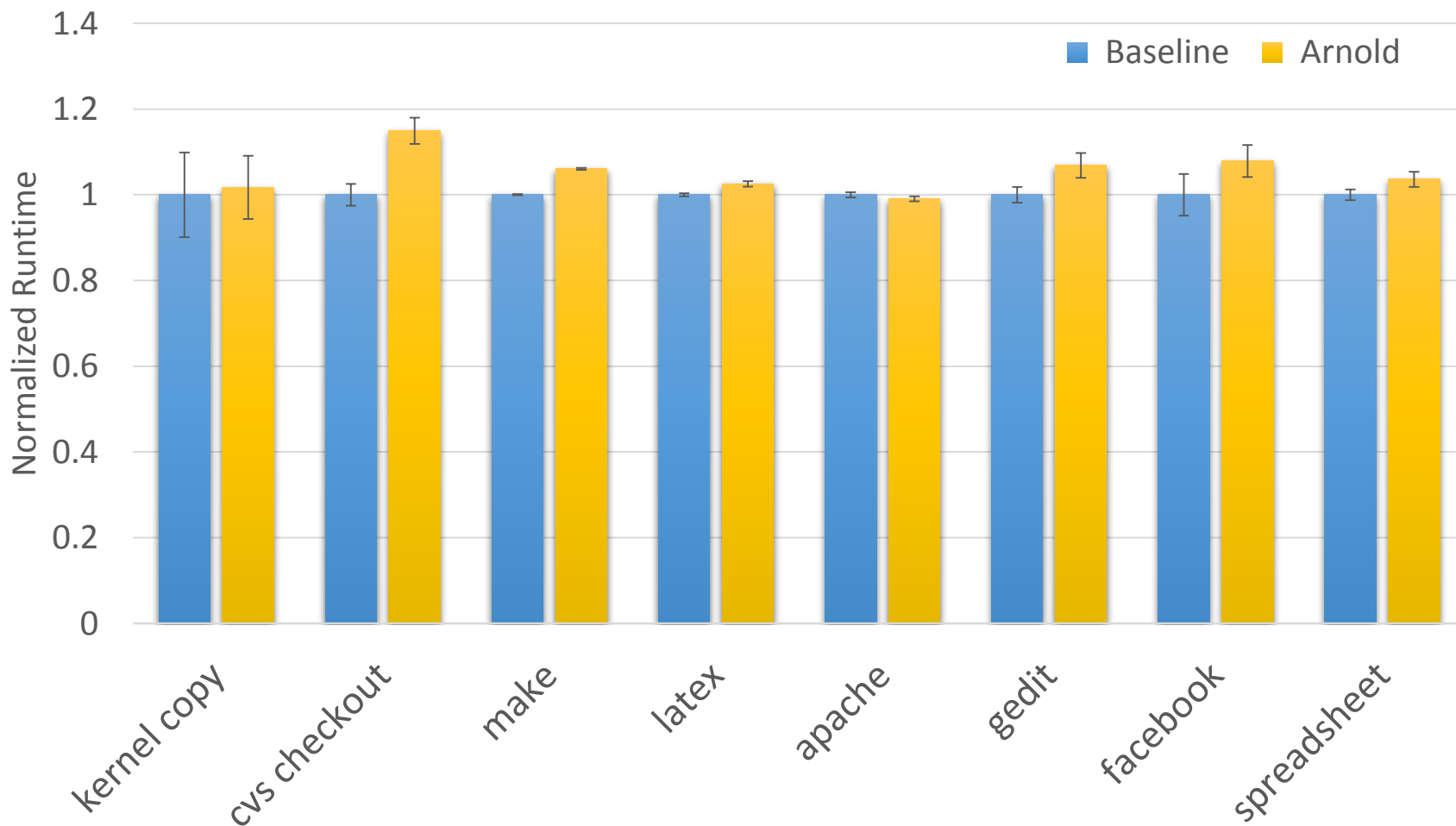
# Semi-Deterministic Time

- Frequent time queries are non-deterministic
- Use partially deterministic clock at record time
  - Real time clock & deterministic clock
  - Bound deviation

```
if (deterministic_clock – real_time_clock < threshold) {  
    adjust deterministic_clock  
    record deviation  
}  
return deterministic_clock
```



# Performance Evaluation



# Overview

- Introduction
- Motivation
- How Arnold remembers all state
- How Arnold supports lineage queries
- Conclusion



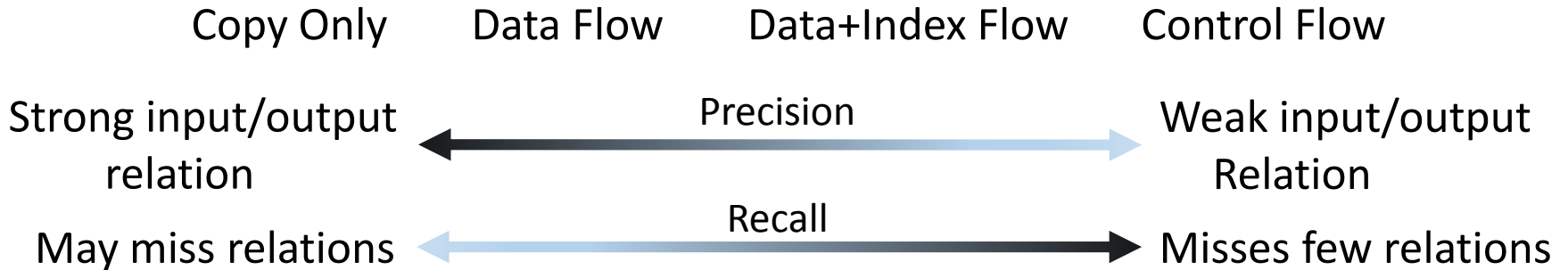
# Querying Lineage

- Two types of queries:
  - Reverse: Where did this data come from?
  - Forward: What did this data affect?
- How does Arnold support these queries?
  - User specifies initial state
  - Trace the lineage of the computation
    - Intra-process tracking
    - Inter-process tracking

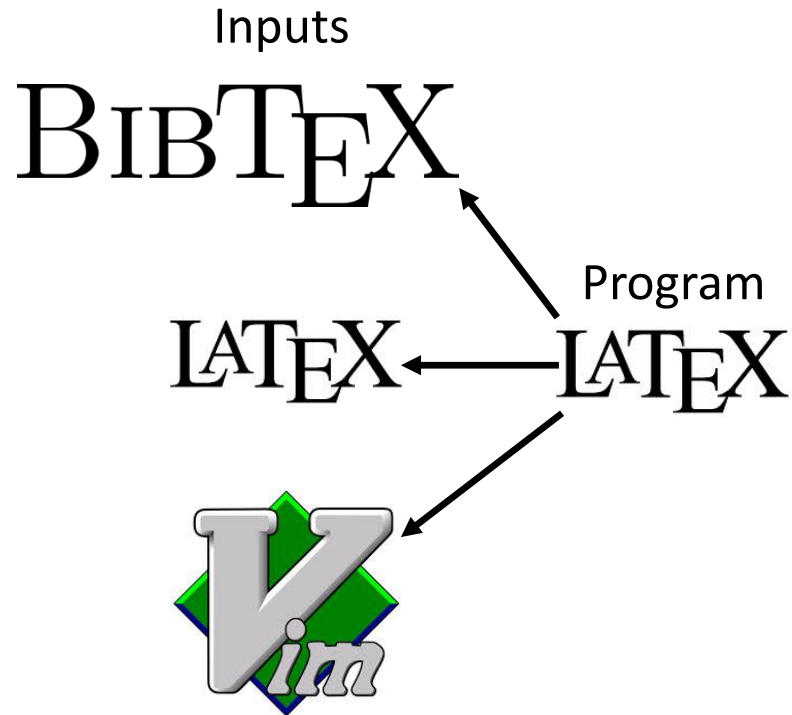


# Intra-Process Lineage

- Use taint tracking for intra-process causality
  - Run retroactively, on recorded execution
  - Parallelizable
- Arnold supports several notions of causality:



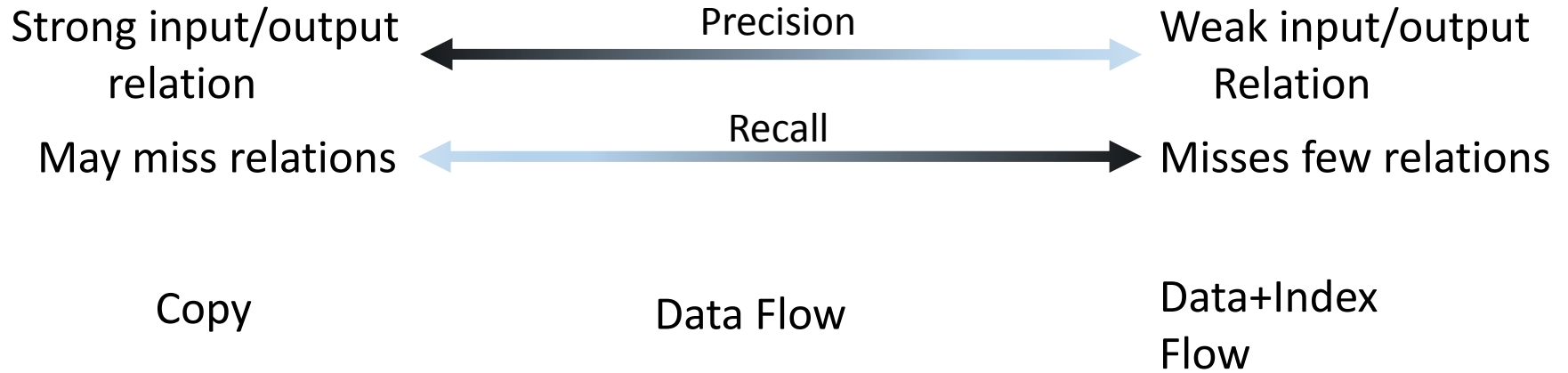
# Intra-Process Lineage



Which linkage tool should Arnold use?



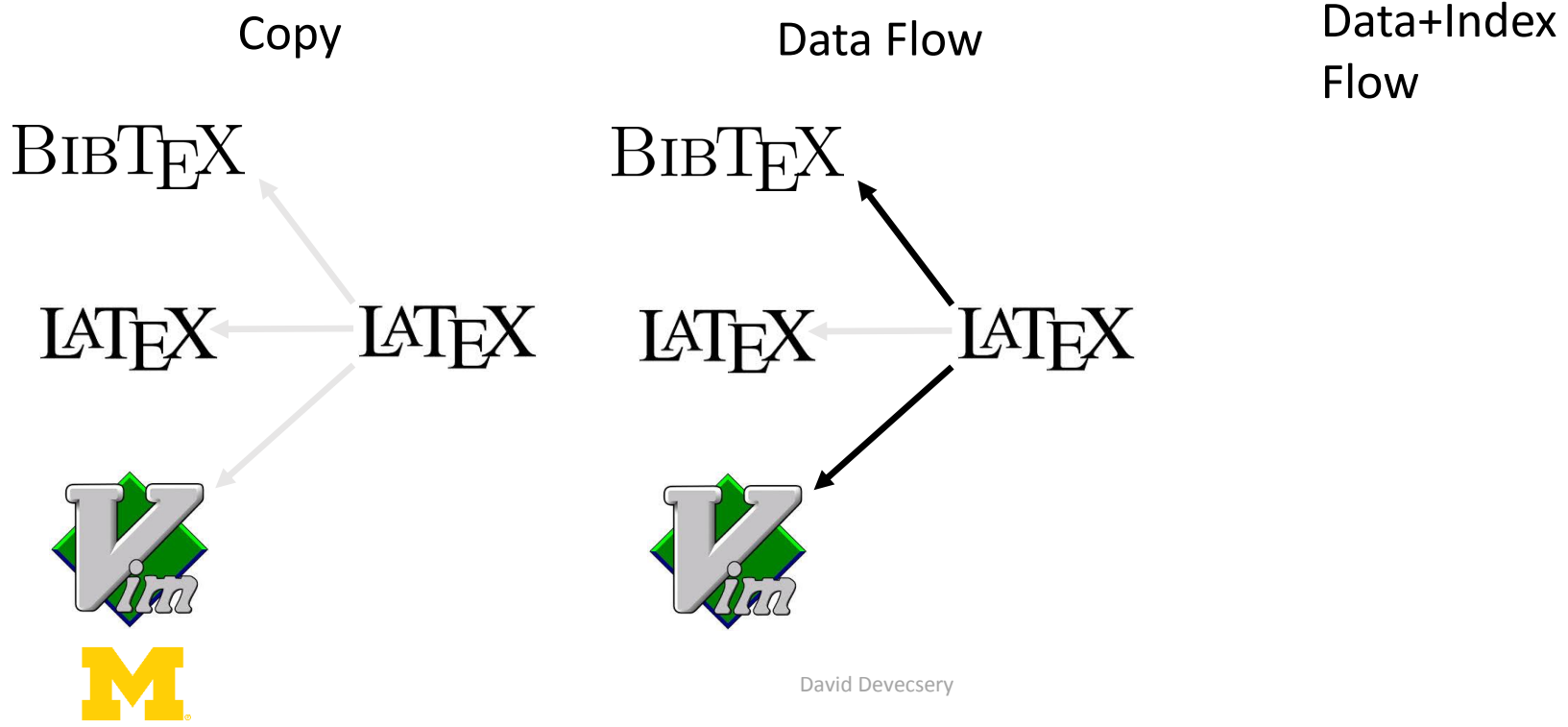
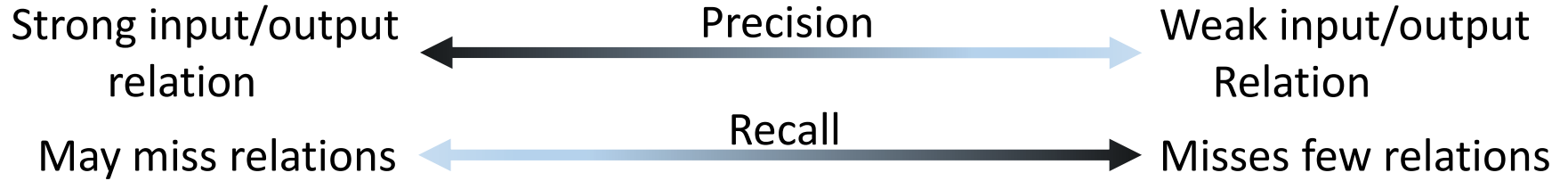
# Intra-Process Lineage



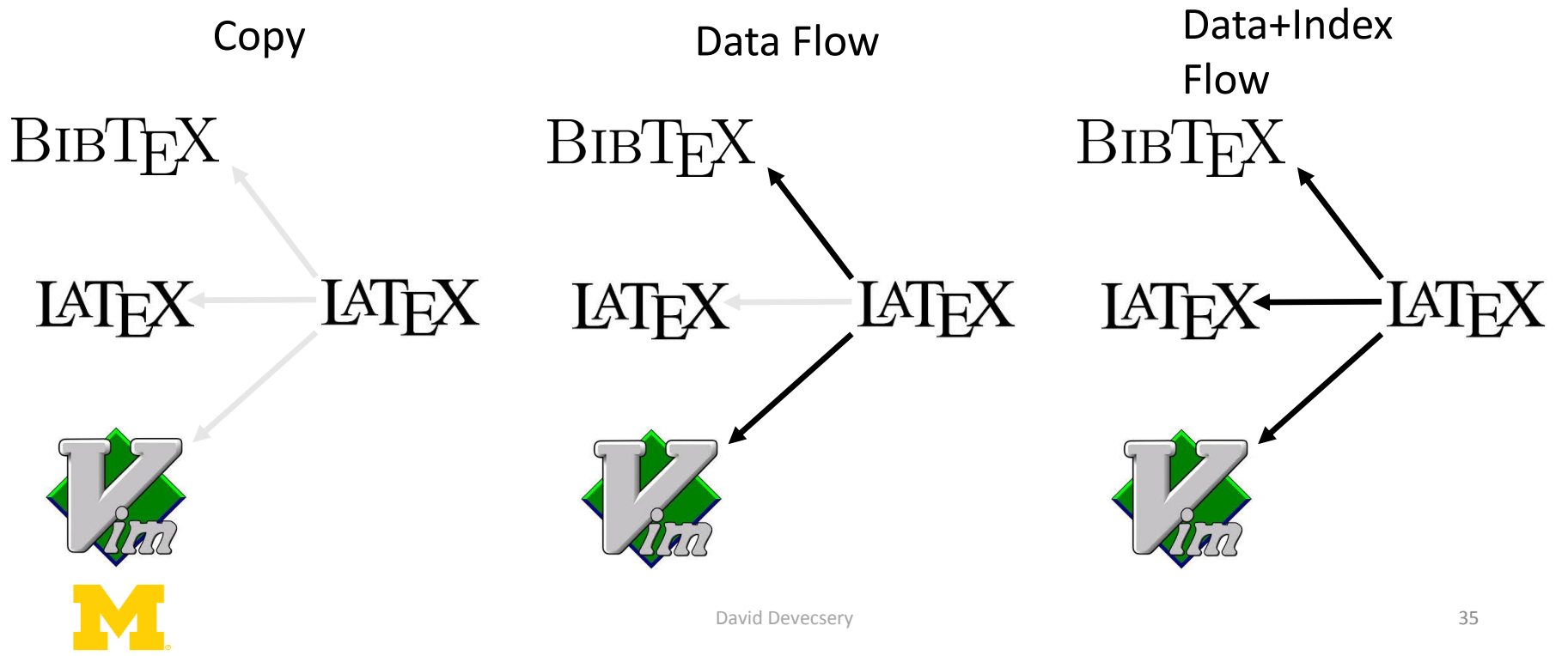
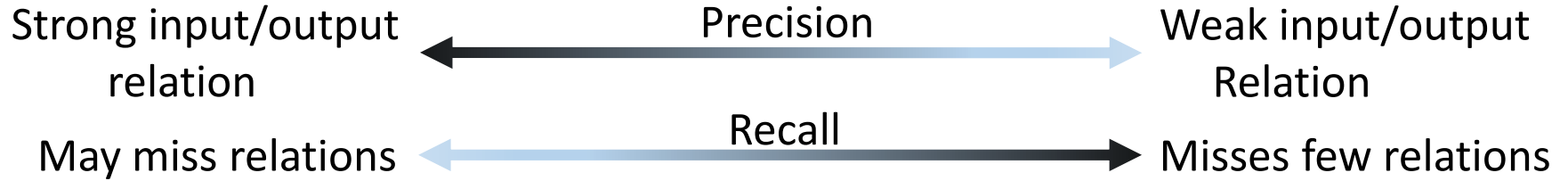




# Intra-Process Lineage



# Intra-Process Lineage



# Intra-Process Lineage

Strong input/output relation

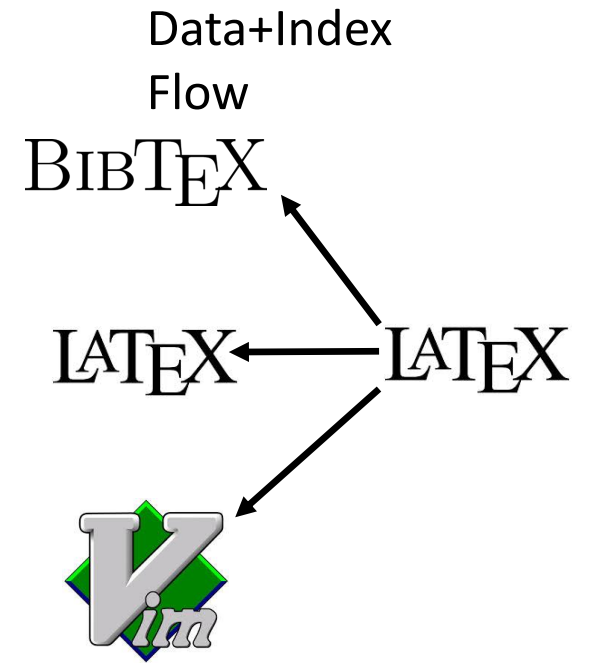
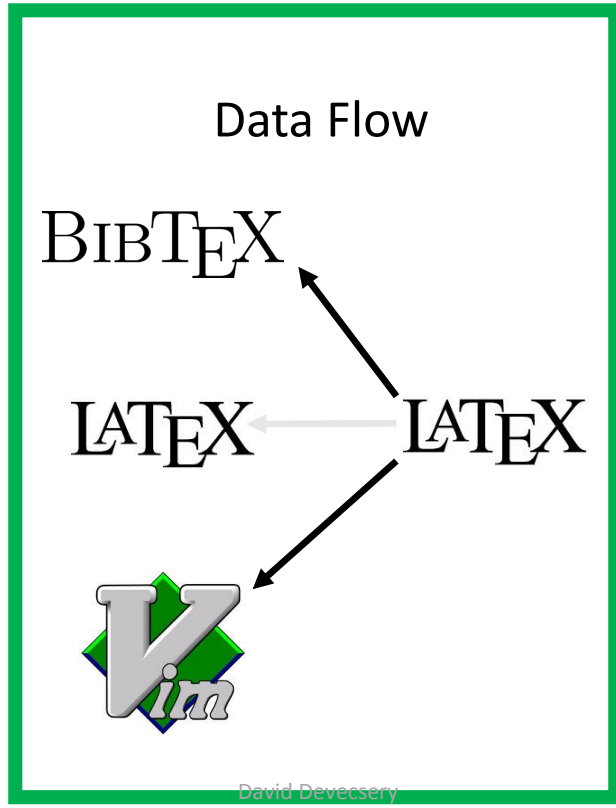
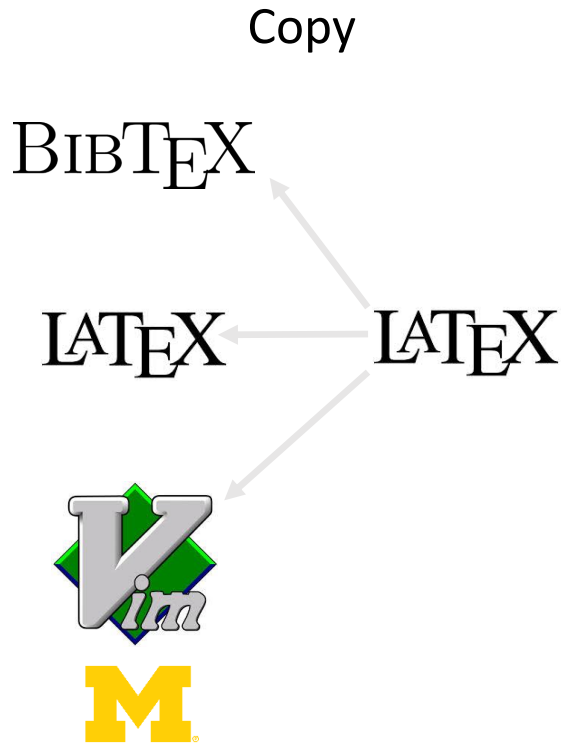
Precision

Weak input/output Relation

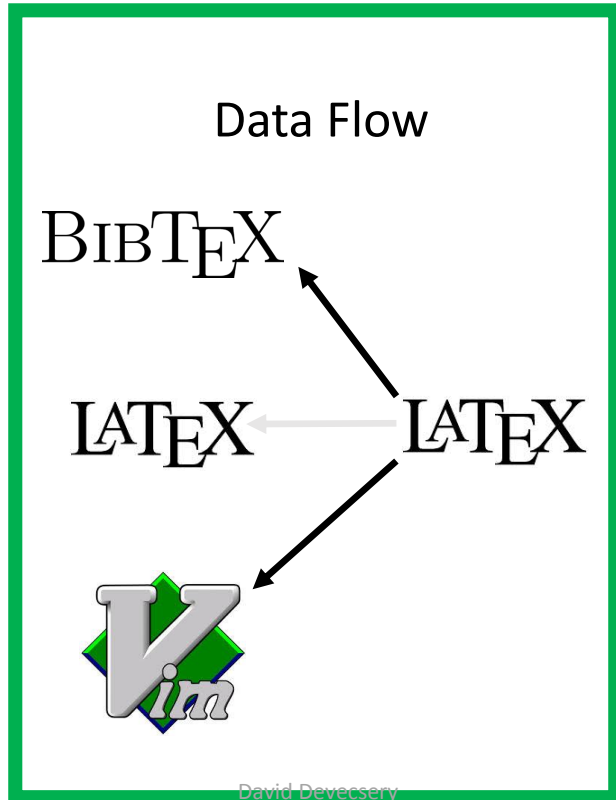
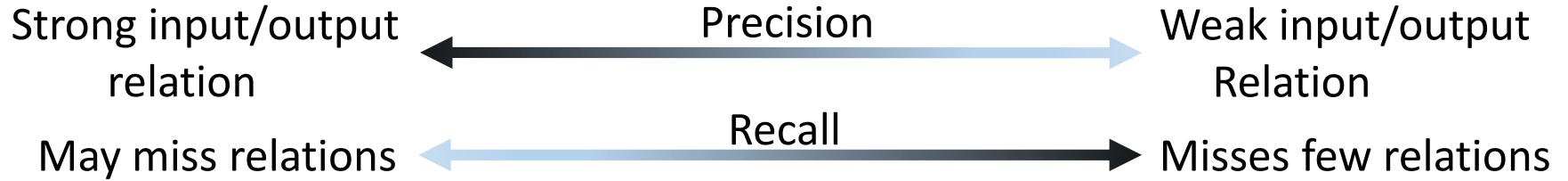
May miss relations

Recall

Misses few relations



# Intra-Process Lineage



Arnold selects the most precise tool with at least one result

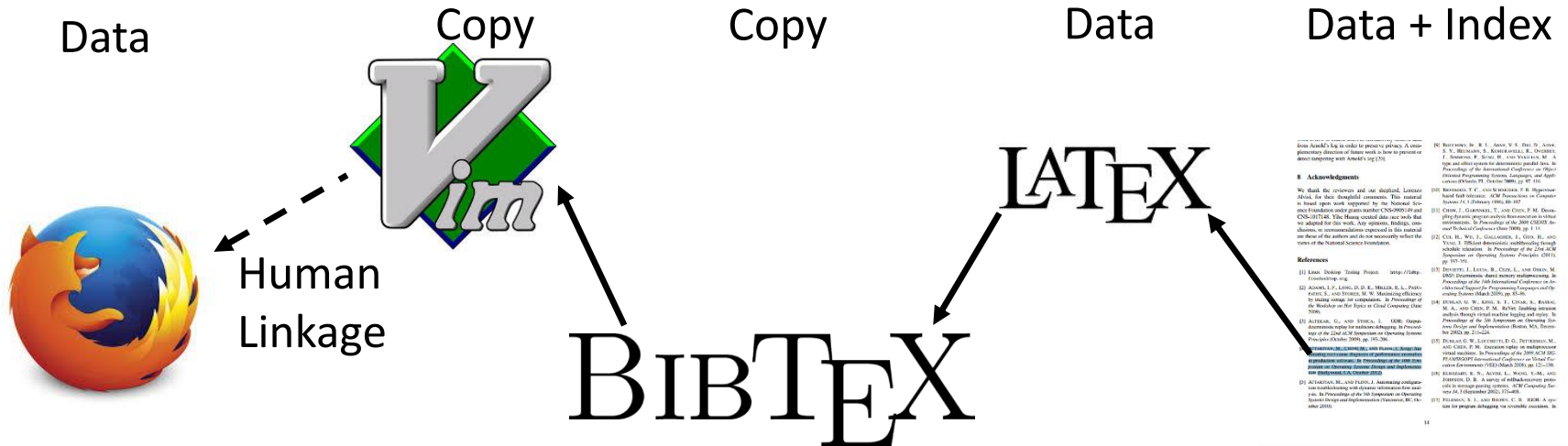


# Inter-Process Lineage

- Two notions of inter-process linkage
- Process graph
  - Tracks lineage through inter-process communication
  - Precise
  - Captures group to group communication
- Human linkage
  - Handles relations between user inputs and outputs
  - Infers linkages based on data content and time
  - Imprecise – may have false negatives and false positives
  - Can capture linkages the process graph can miss



# Evaluation – Wrong Reference



- Few false positives (font files, latex sty files, libc.so, libXt.so)
- No false negatives

Record Time	Replay Time	Replay + Pin Time	Query Time
96.1s	2.2s	70.0s	209.5s

# Evaluation – Heartbleed

Data + Index

Data + Index

Data + Index



- No false positives or negatives

Record Time	Replay Time	Replay + Pin Time	Query Time
230.3s	0.4s	139.5s	235.1s



# Conclusion

- Eidetic Systems are powerful tools
  - Complete vision into past computation
  - Answer powerful queries about state's lineage
- Arnold – First practical Eidetic System
  - Low runtime overhead
  - 4 years of computation on a commodity HD
  - Supports powerful lineage queries
- Code is released
  - <https://github.com/endplay/omniplay>



# Questions?



# Backup Slides



# Cloud Storage

- Future work
- Two approaches:
  - Statically served content
  - Distributed replay system



# Related Works

- Execution Mining (Tralfamadore)
- DeJaVu
- RAIL



# User Study Log-Sizes

Users	Days	Groups Per Day	Storage Utilization (MB)			
			RAW File Cache	Logs	Filemap	Total
A	25	995	475	267	36	779
B	24	475	1095	936	339	2064
C	21	26122	869	350	690	1910
D	16	3339	1675	838	838	2594



