Toward Eidetic Distributed File Systems

Xianzheng Dou, Jason Flinn, Peter M. Chen

UNIVERSITY OF MICHIGAN

Rich file system features

- Modern file systems store more than just data
  - Versioning: retention of past state
  - Provenance-aware: connections between file data

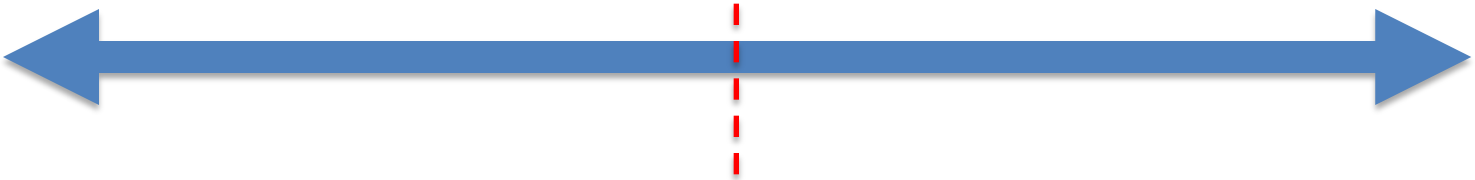
- Problem:
  - High costs for providing these rich features

Versioning FS tradeoffs

- Frequency of versioning

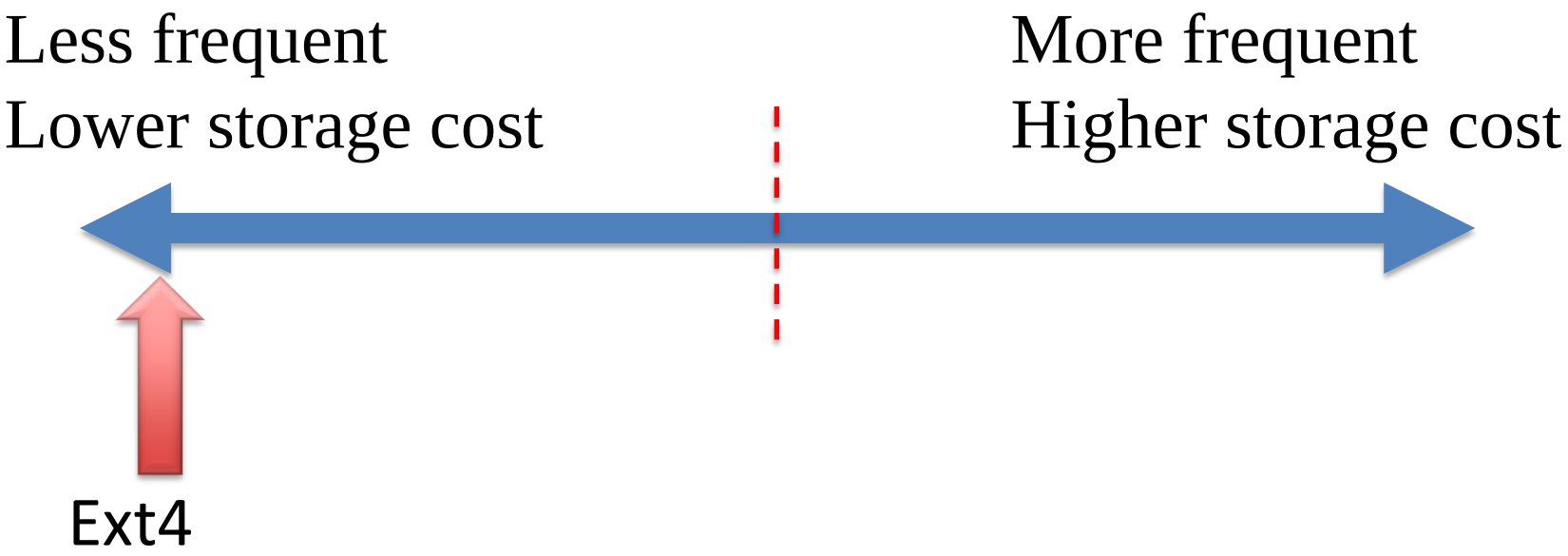
Less frequent
Lower storage cost

More frequent
Higher storage cost



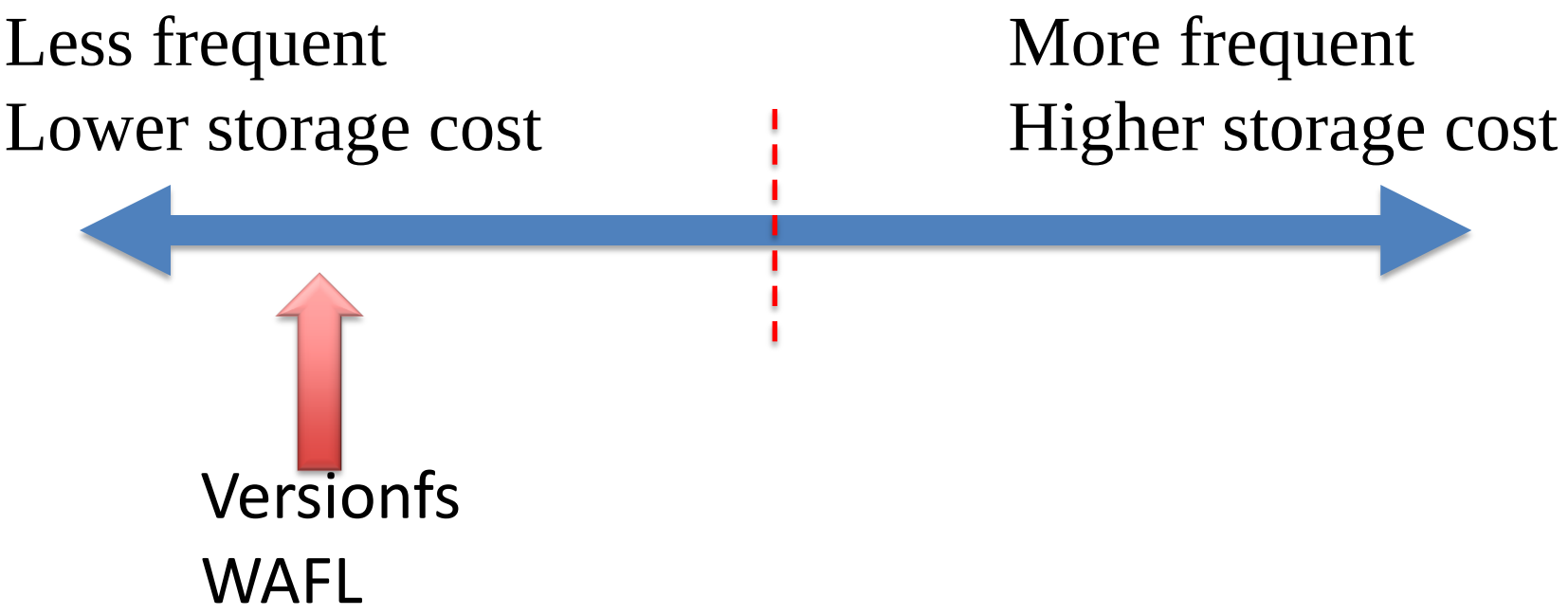
Versioning FS tradeoffs

- Frequency of versioning



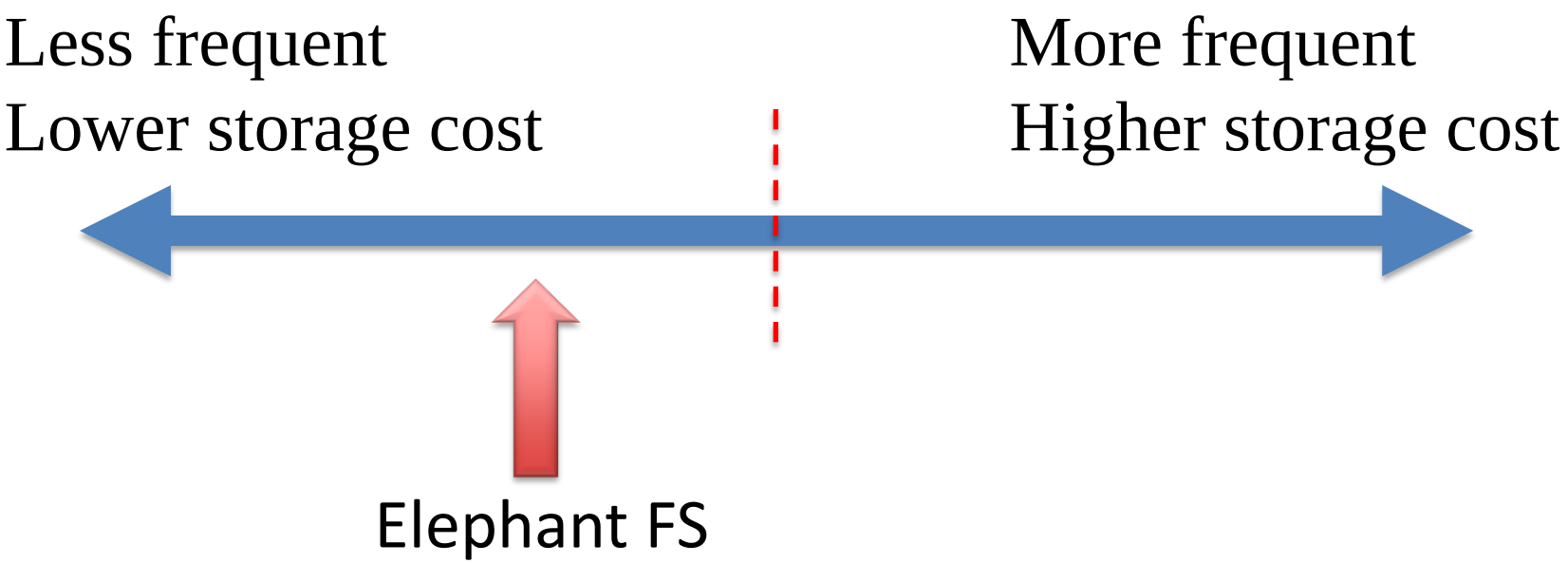
Versioning FS tradeoffs

- Frequency of versioning



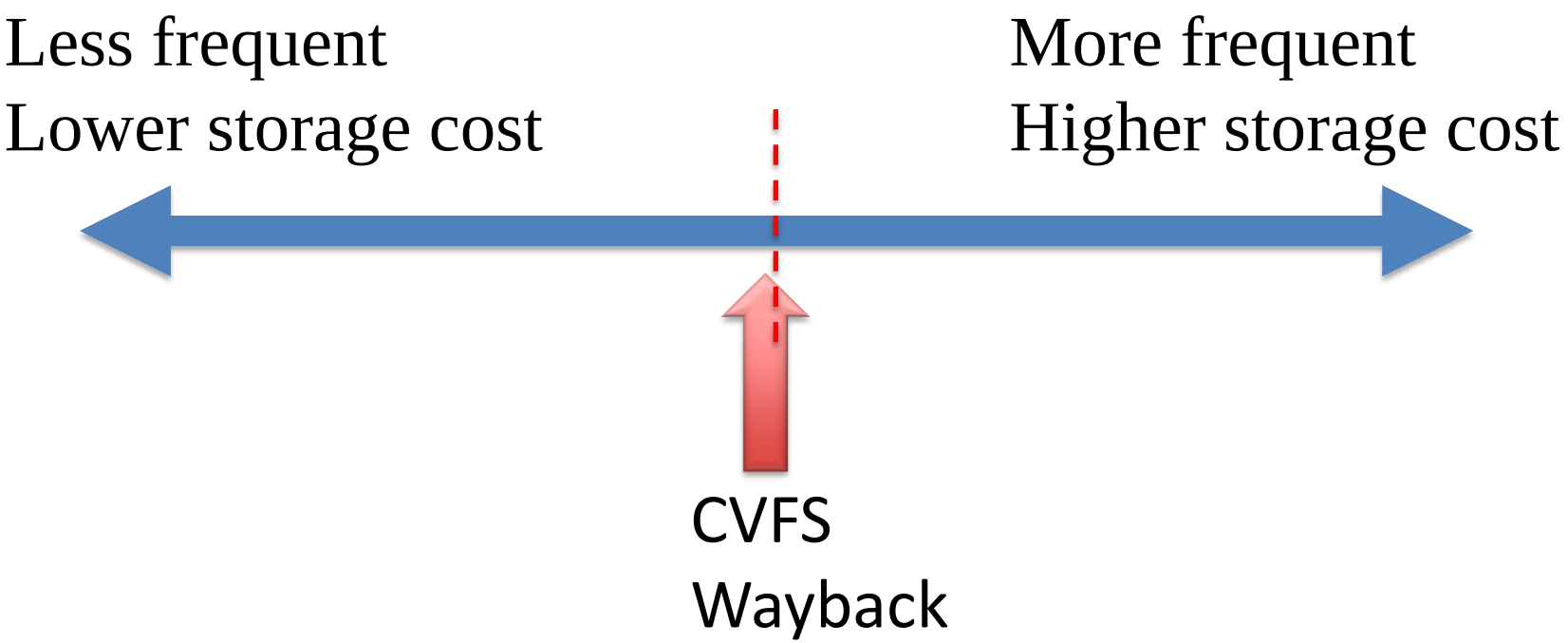
Versioning FS tradeoffs

- Frequency of versioning



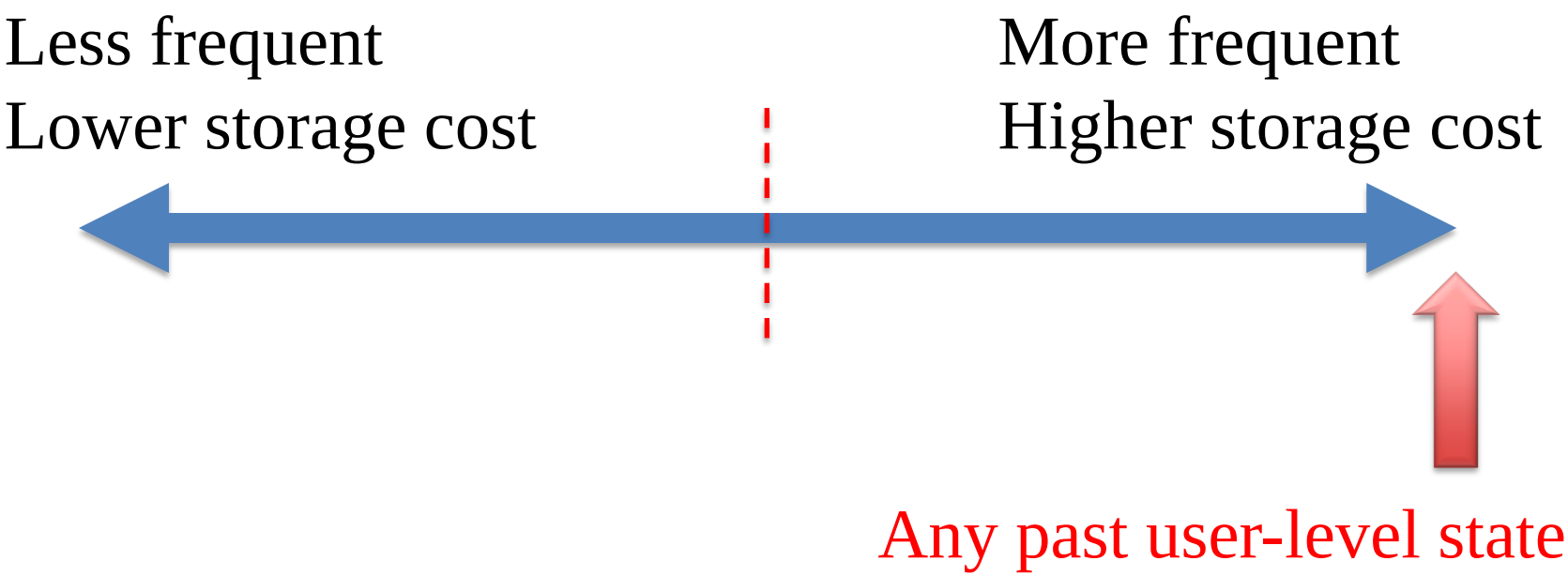
Versioning FS tradeoffs

- Frequency of versioning



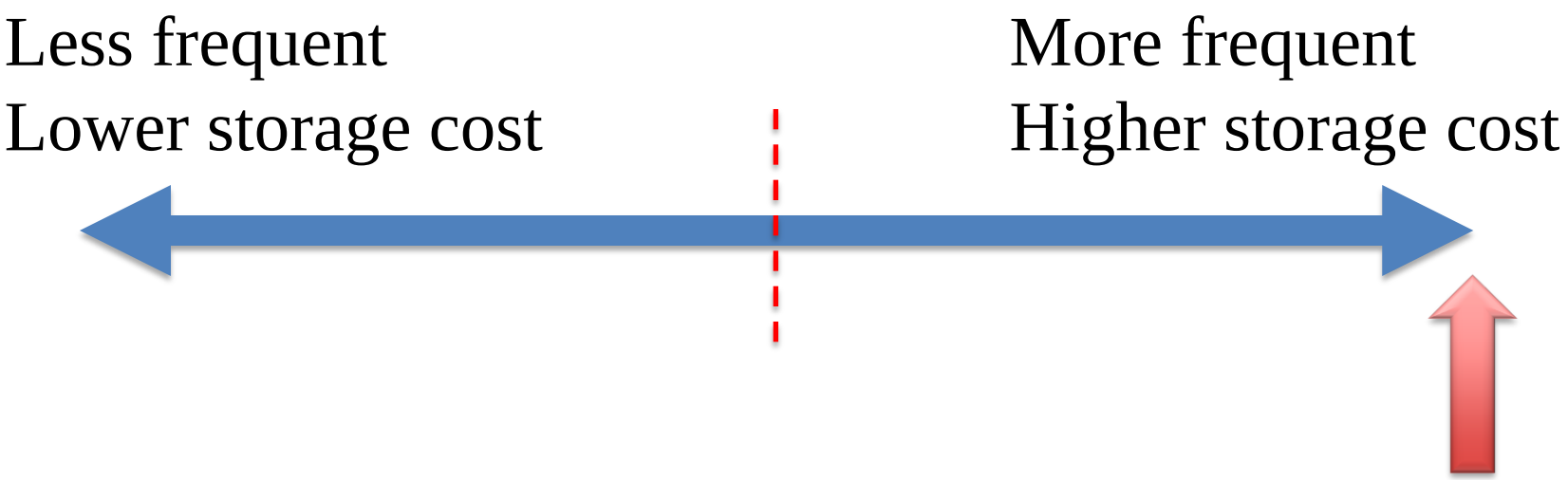
Versioning FS tradeoffs

- Frequency of versioning



Versioning FS tradeoffs

- Frequency of versioning



Any past user-level state?

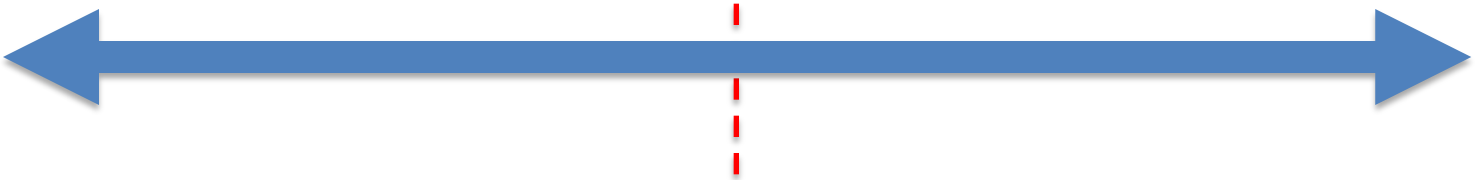
Any past file system state and any transient state

Provenance FS tradeoffs

- Details of connection information

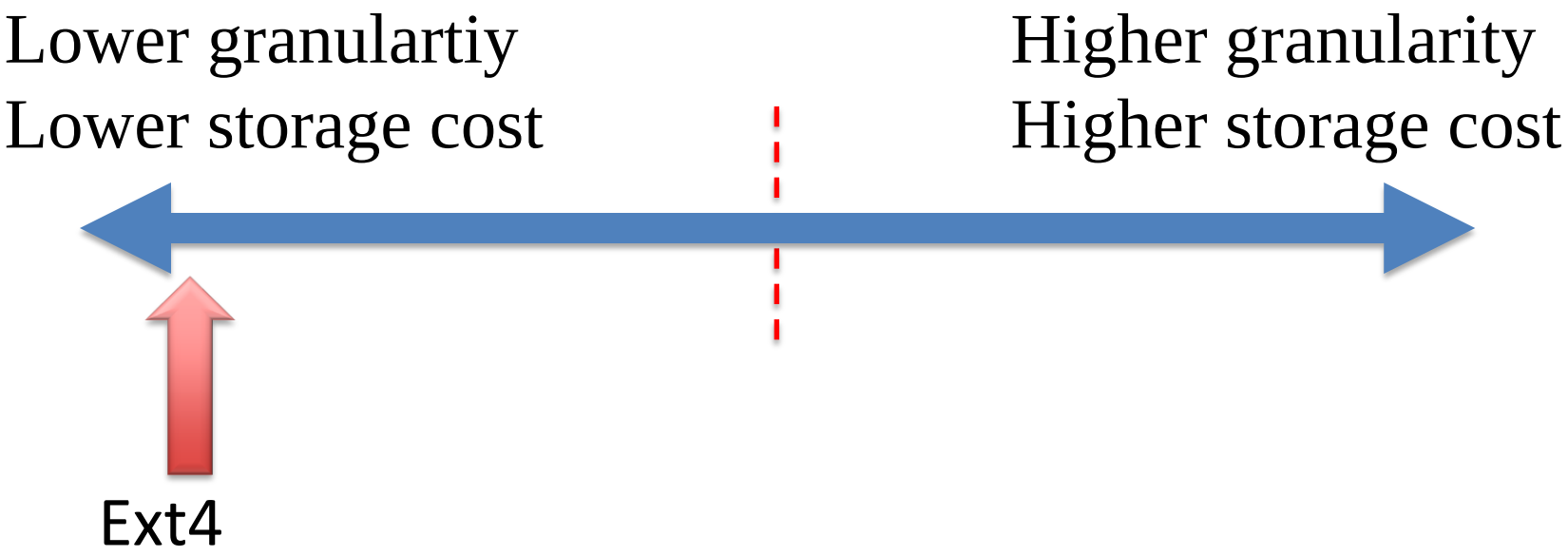
Lower granulartiy
Lower storage cost

Higher granularity
Higher storage cost



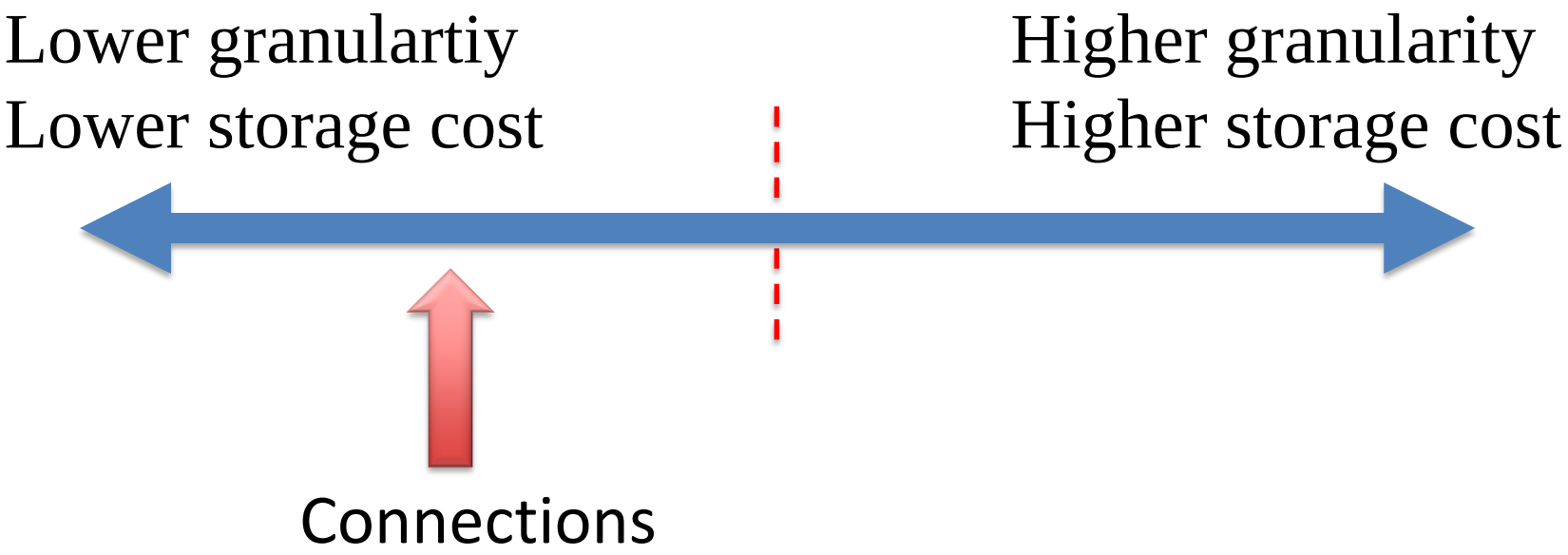
Provenance FS tradeoffs

- Details of connection information



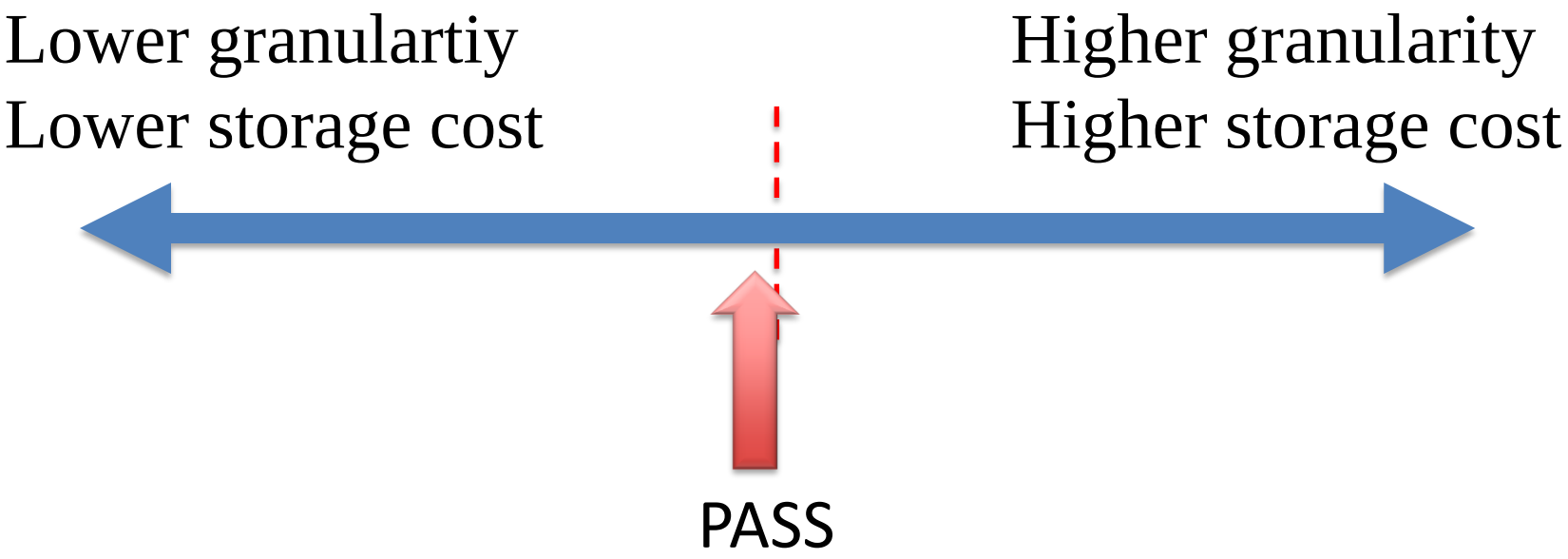
Provenance FS tradeoffs

- Details of connection information



Provenance FS tradeoffs

- Details of connection information

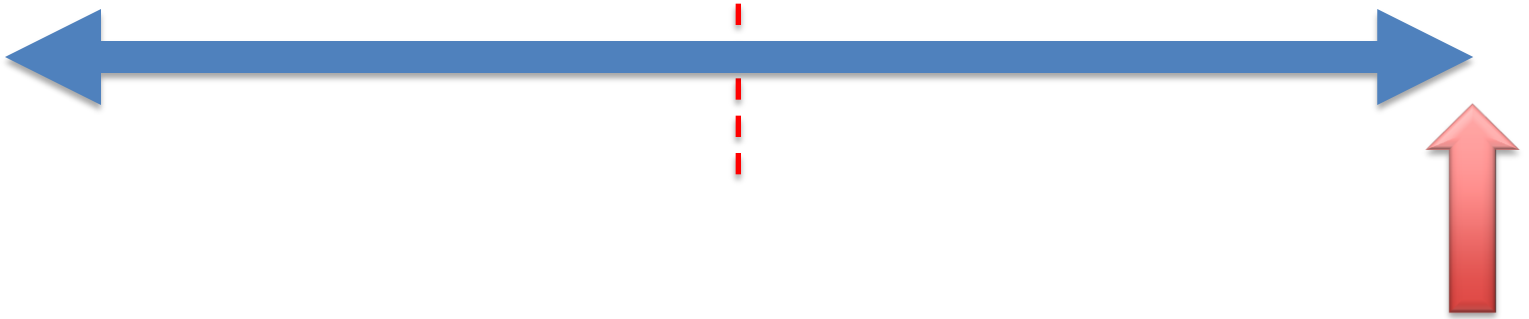


Provenance FS tradeoffs

- Details of connection information

Lower granulartiy
Lower storage cost

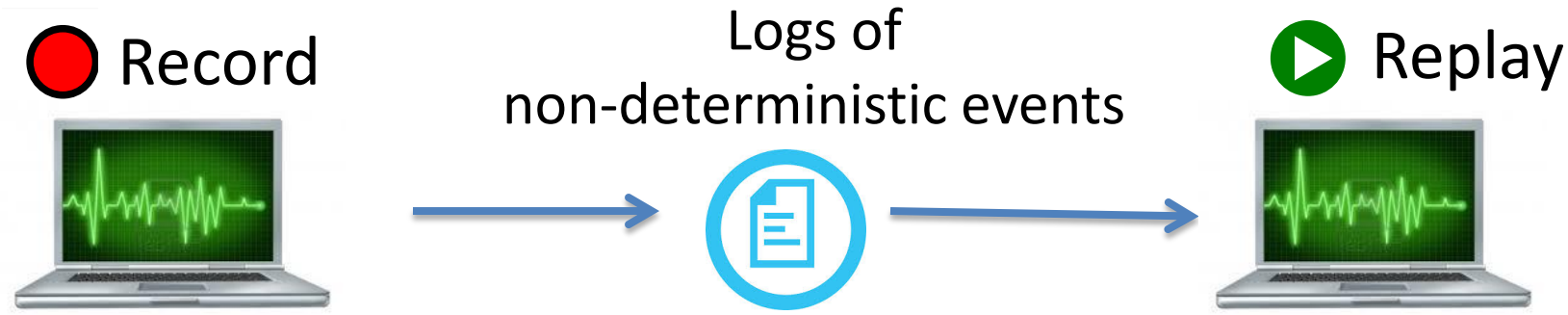
Higher granularity
Higher storage cost



Complete byte-level provenance?

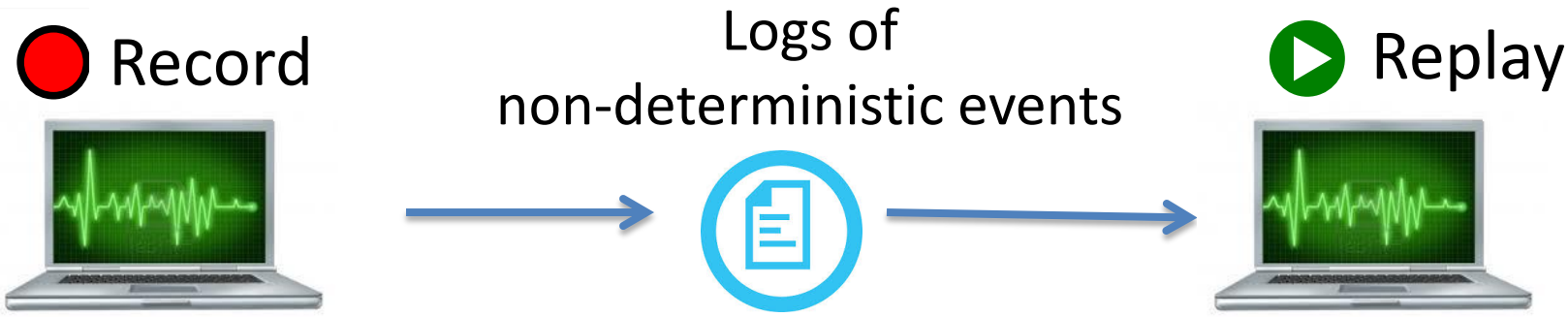
Background: eidetic systems[OSDI'14]

- Recall any past user-level state
  - By pervasive deterministic record and replay



Background: eidetic systems[OSDI'14]

- Recall any past user-level state
  - By pervasive deterministic record and replay



- Provenance at the byte granularity
  - Intra-process lineage: dynamic information tracking
  - Inter-process lineage: data flow dependency graph

A clean-sheet design of FS

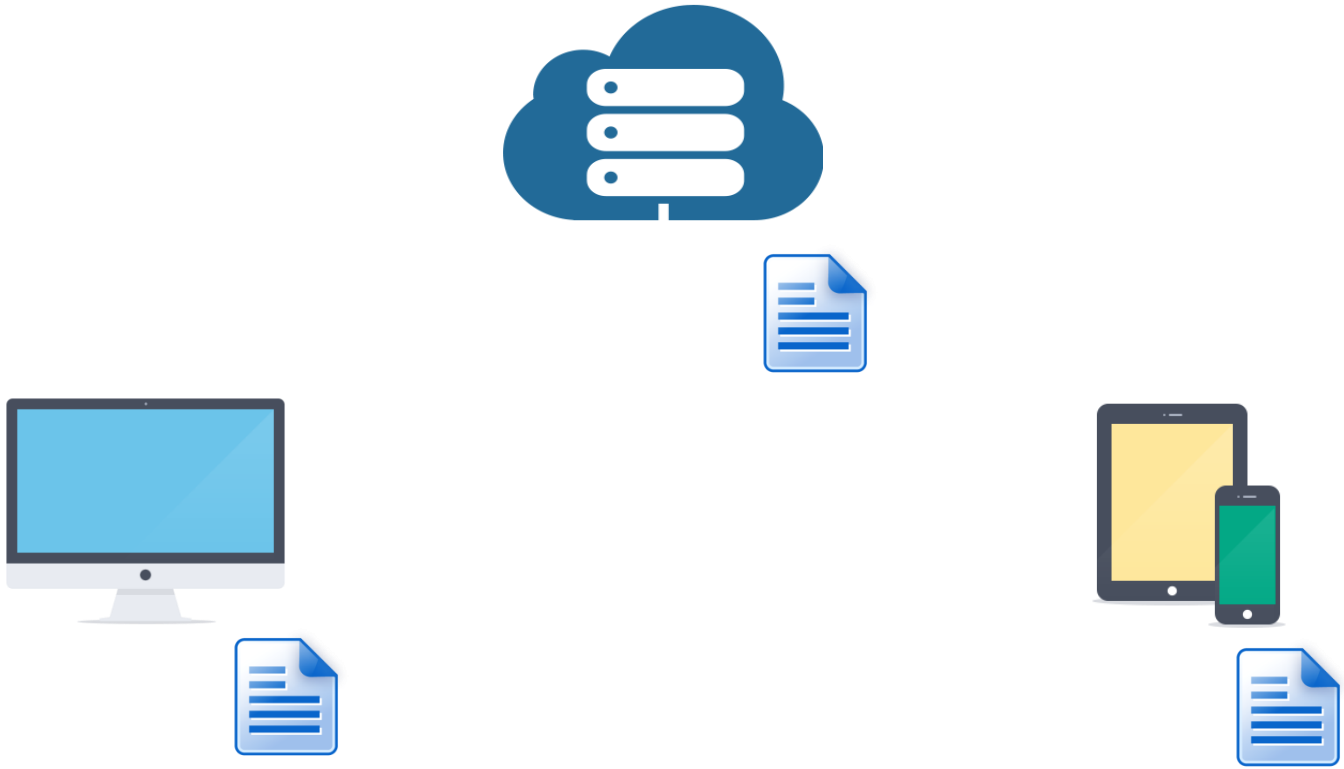
- Eidetic systems prototype
  - Graft eidetic support onto an existing FS
  - Consider only local storage
- An eidetic distributed file system
  - A small number of personal devices + cloud servers
- New design choices
  - Fundamental unit of persistent storage
  - File transfer

Traditional distributed FS

Traditional distributed FS

Traditional distributed FS

Traditional distributed FS



Eidetic distributed file systems

Eidetic distributed file systems

Fundamental unit

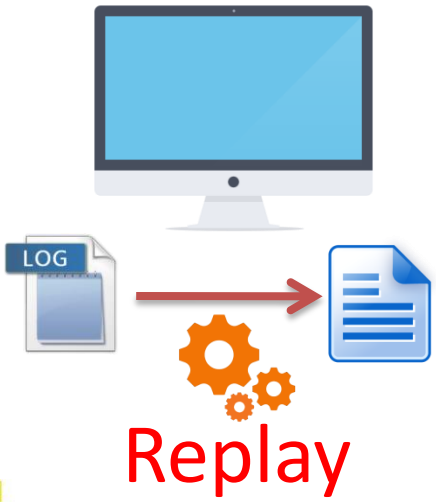
- What is the fundamental unit of persistent storage?

Fundamental unit

- What is the fundamental unit of persistent storage?

Fundamental unit

- What is the fundamental unit of persistent storage?

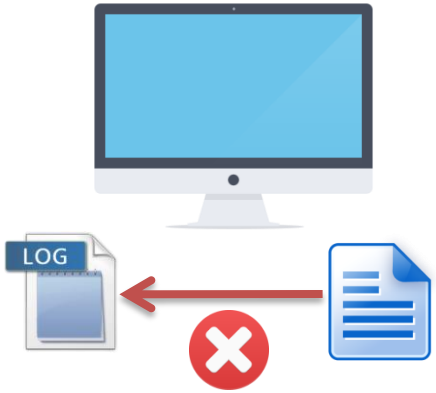


Fundamental unit

- What is the fundamental unit of persistent storage?

Fundamental unit

- What is the fundamental unit of persistent storage?



Fundamental unit

- What is the fundamental unit of persistent storage?

Fundamental unit: ***Logs of non-determinism***

Files are only considered as caches

File persistency

- When is a file considered persistent on the server?

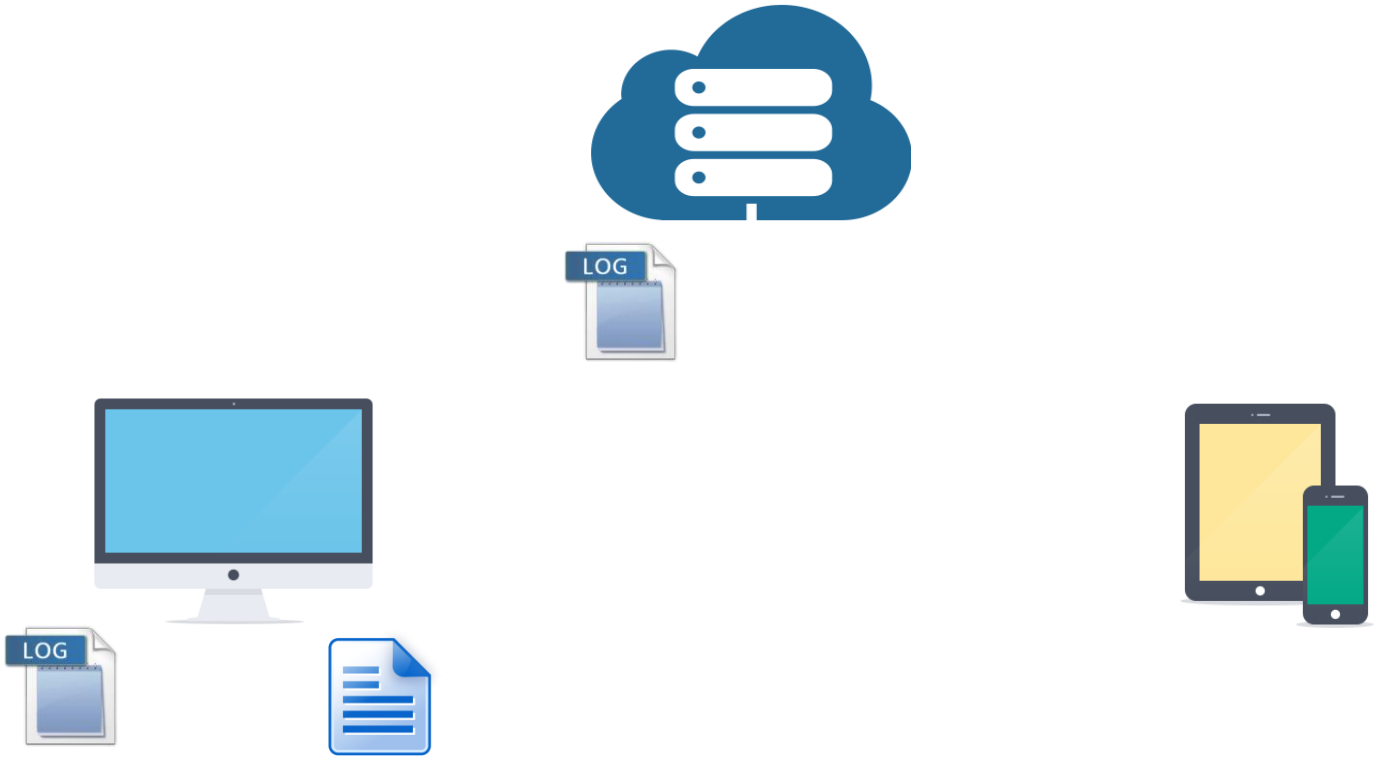
File persistency

- When is a file considered persistent on the server?

As long as logs generating the data are persistent

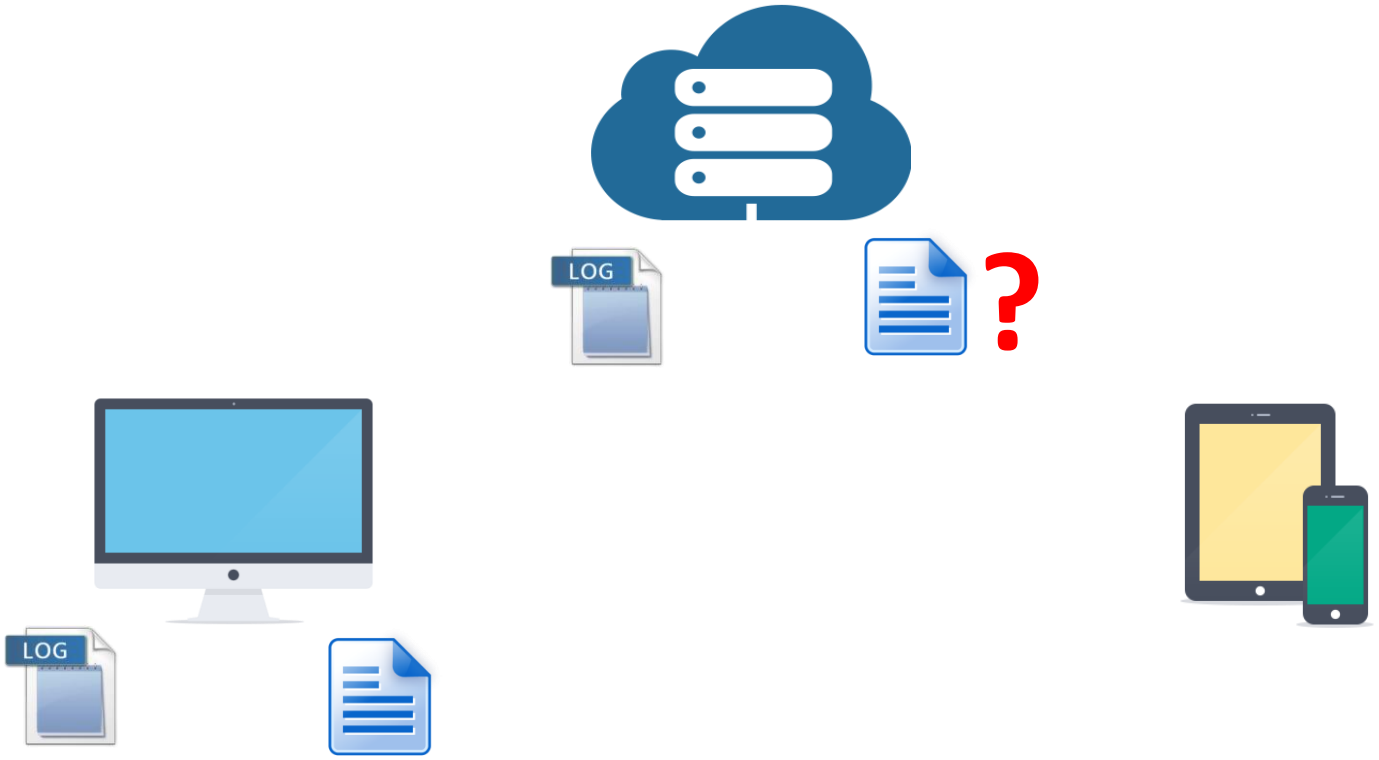
File persistency

- When is a file considered persistent on the server?



Updating server cache

- Should the server cache the file version?



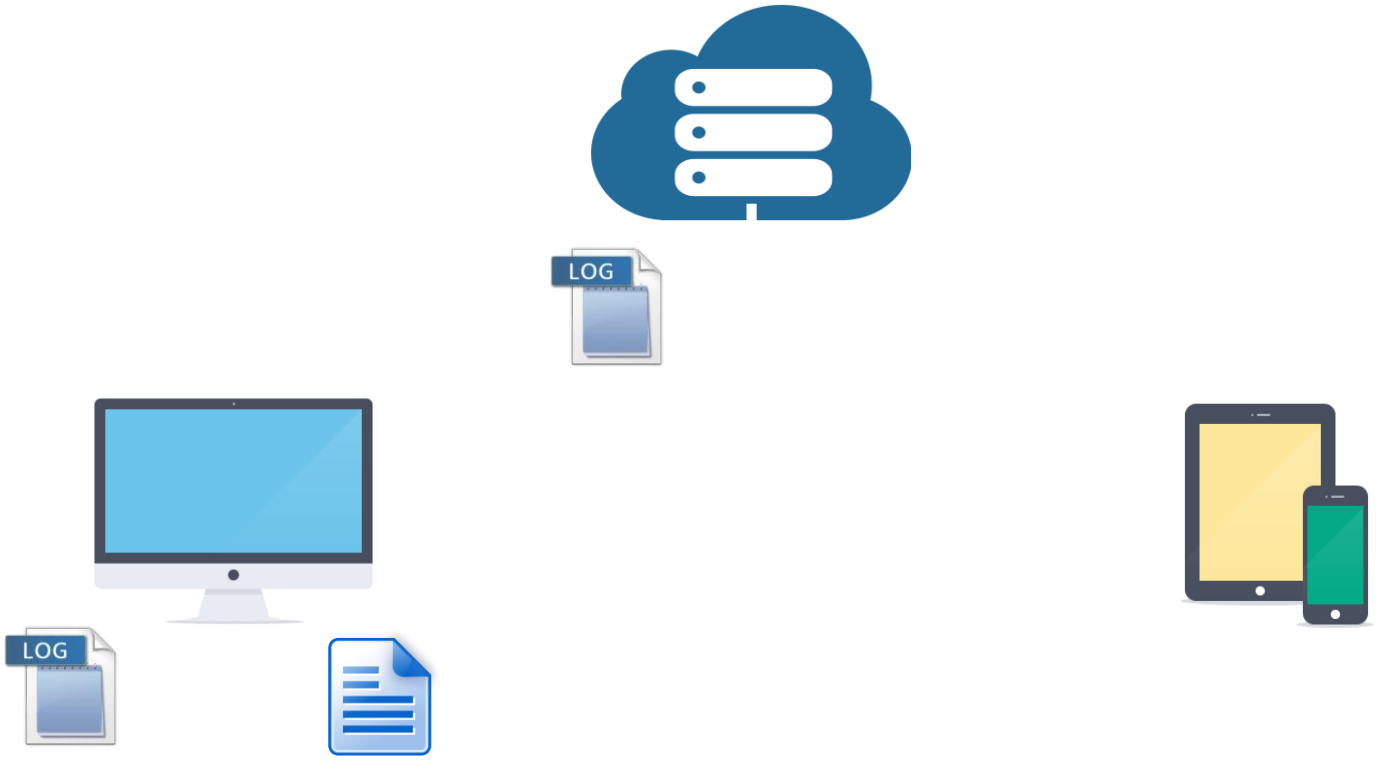
Updating server cache

- Should the server cache the file version?

Probability of future access
Costs for regeneration

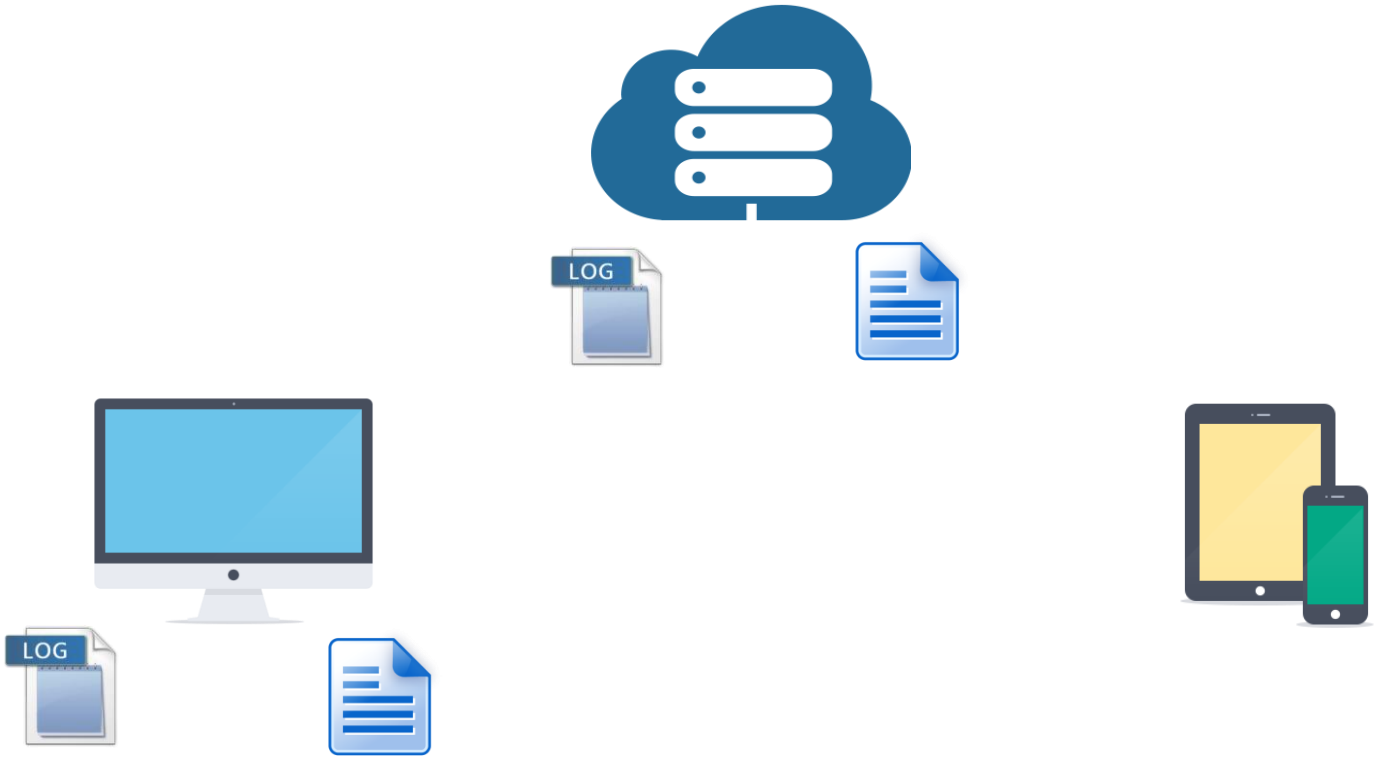
File transfer methods

- How are files transferred to the server?



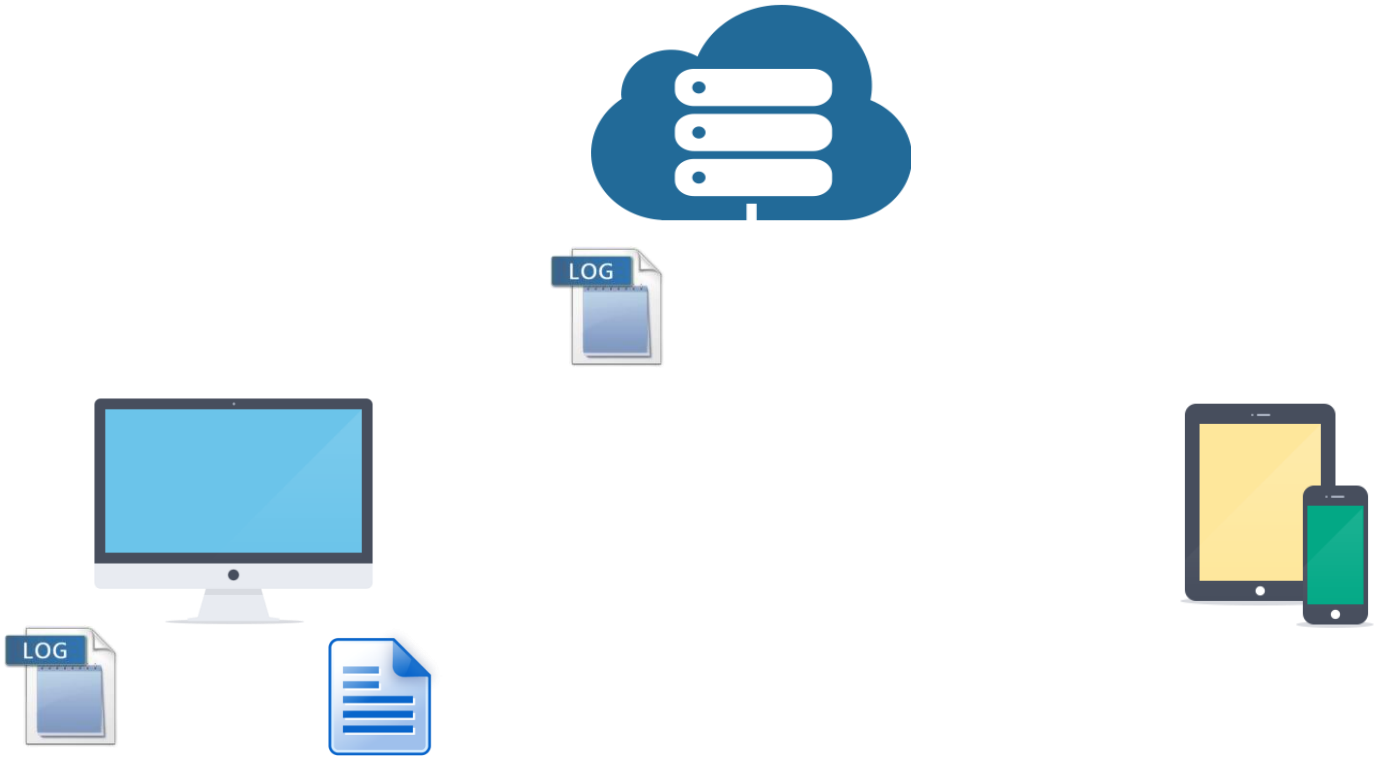
File transfer methods

- How are files transferred to the server?



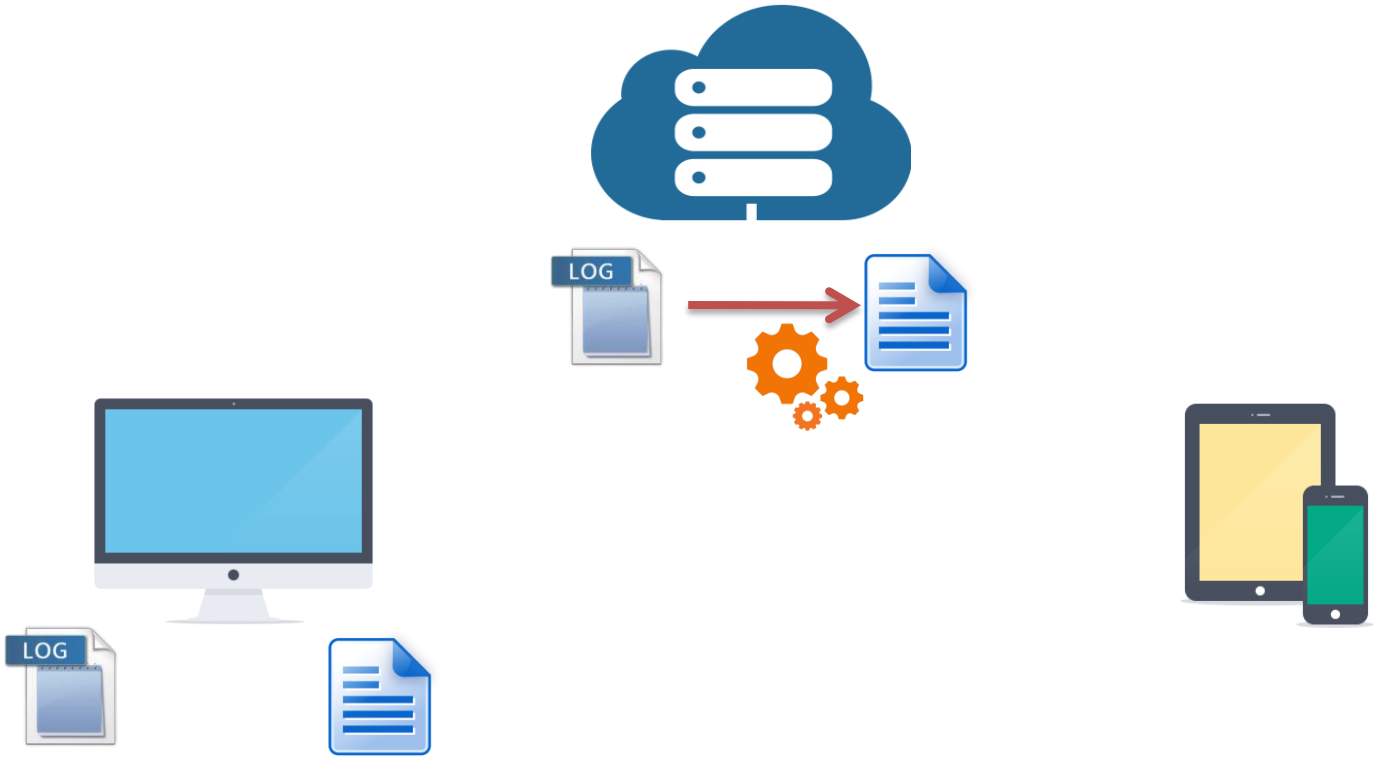
File transfer methods

- How are files transferred to the server?



File transfer methods

- How are files transferred to the server?



File transfer methods

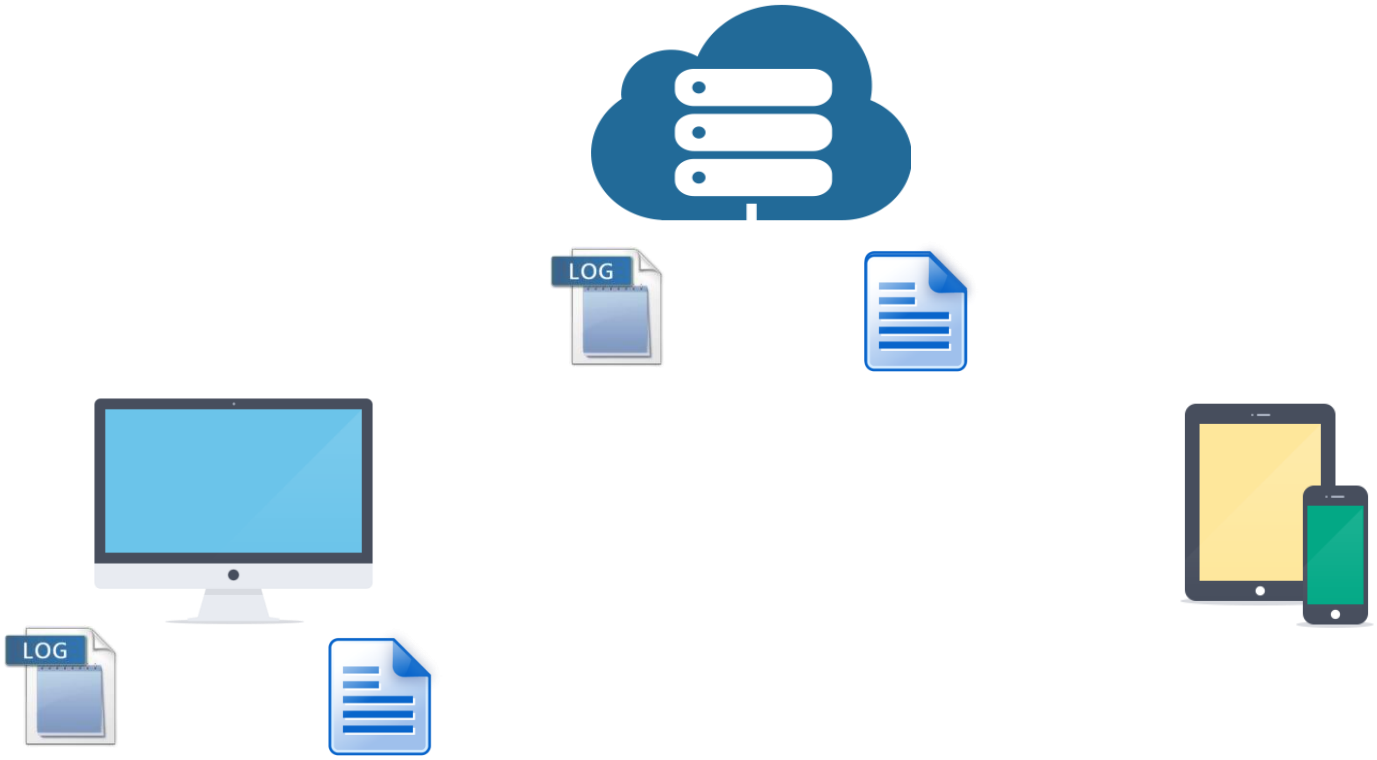
- How are files transferred to the server?

Compare computation costs with communication costs

- by value (file data)
- or by replay

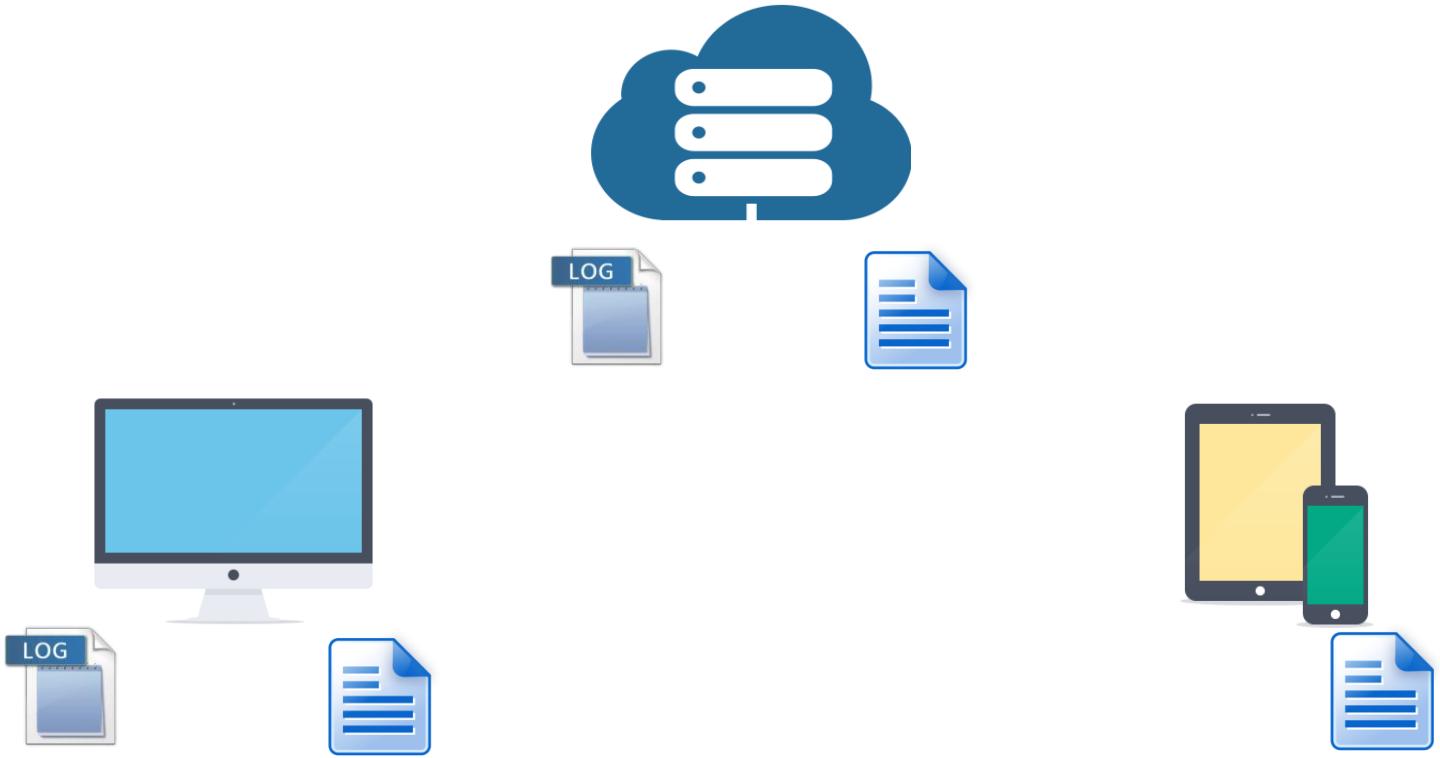
Read path

- How should a client read a particular version?



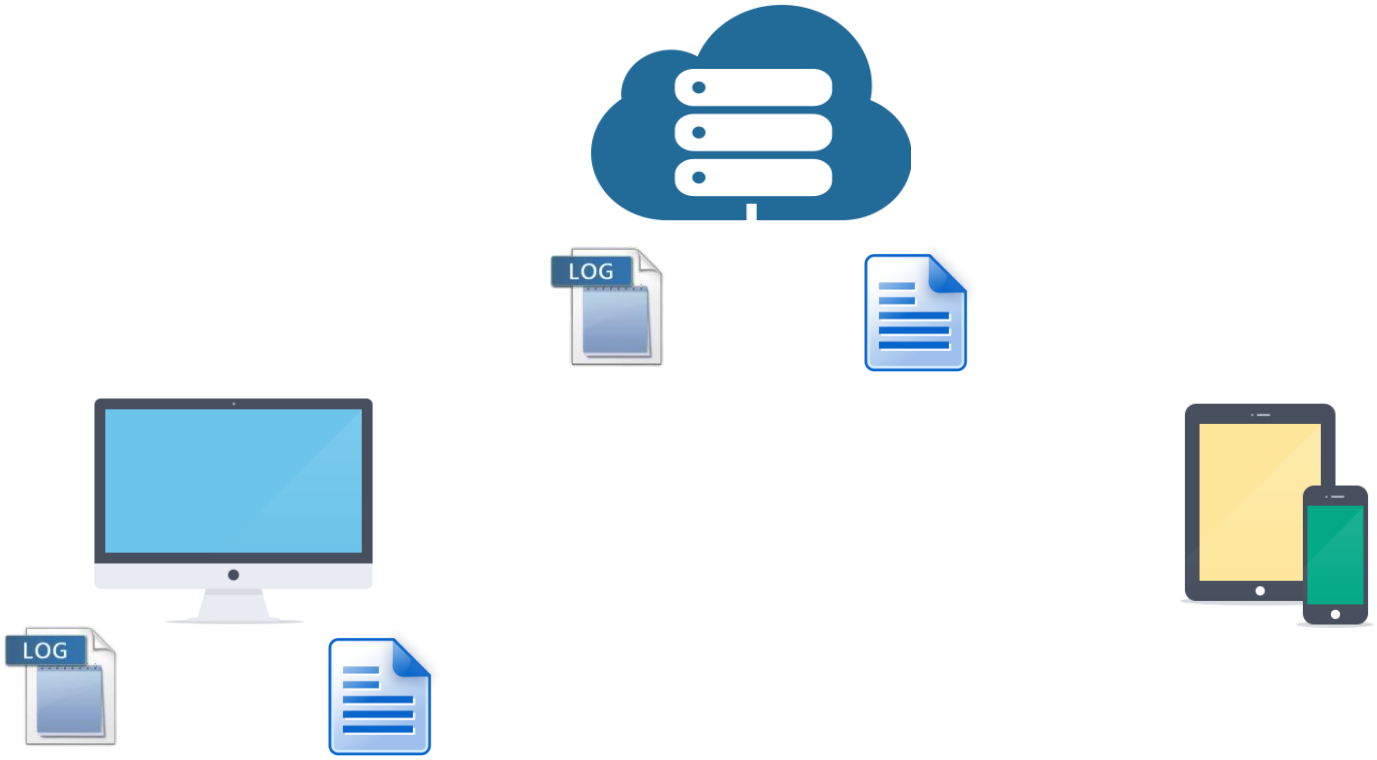
Read path

- How should a client read a particular version?



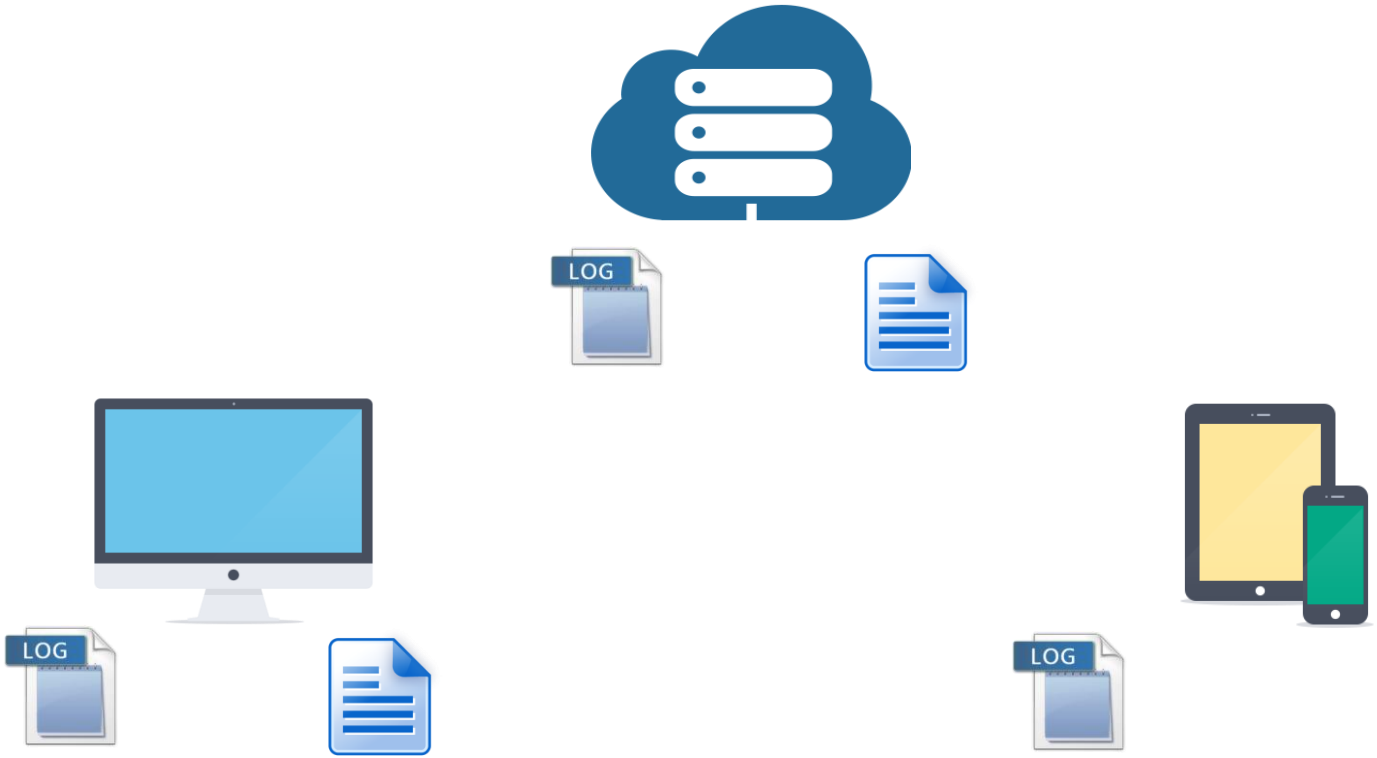
Available transfer methods

- How should a client read a particular version?



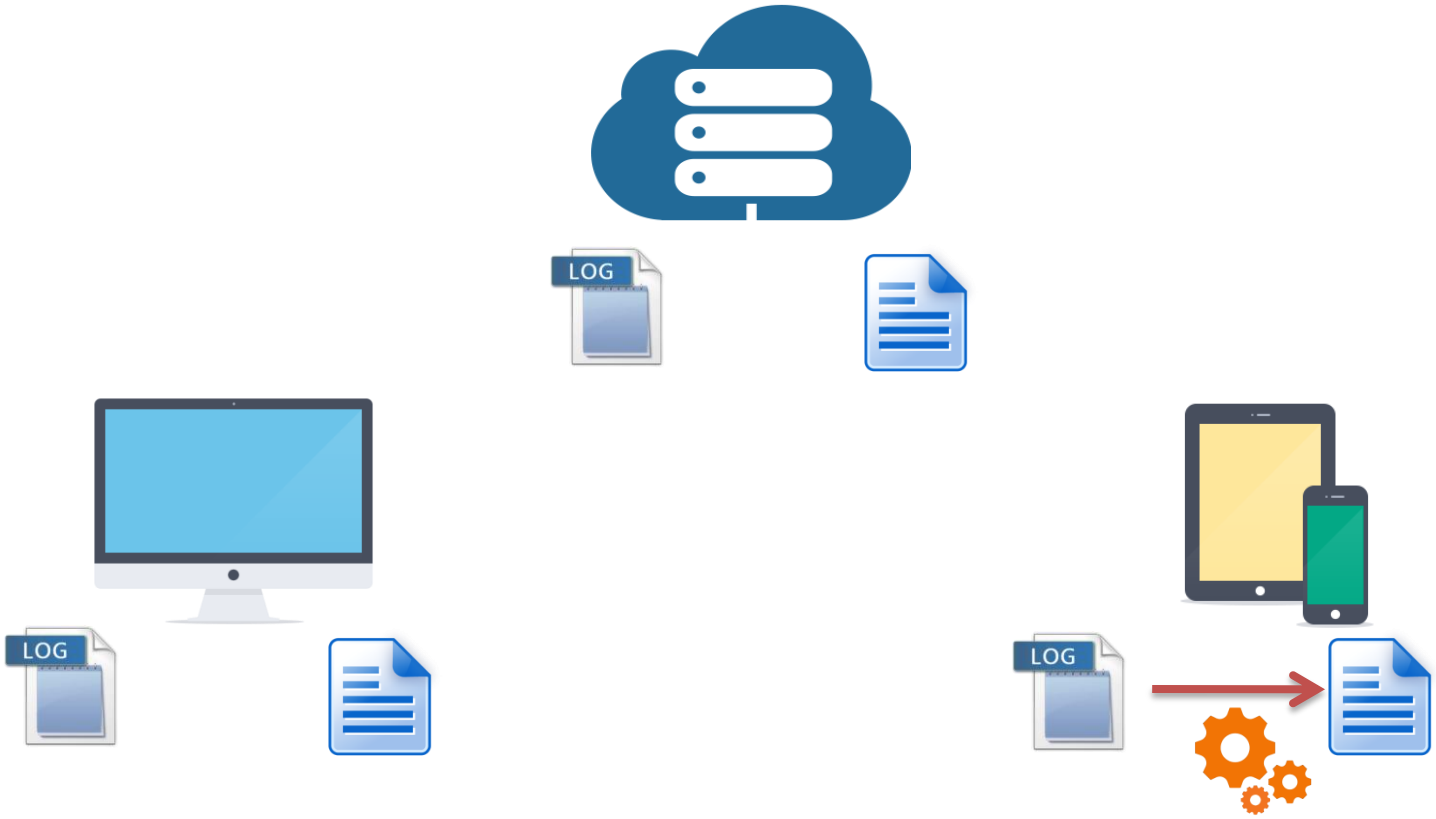
Available transfer methods

- How should a client read a particular version?



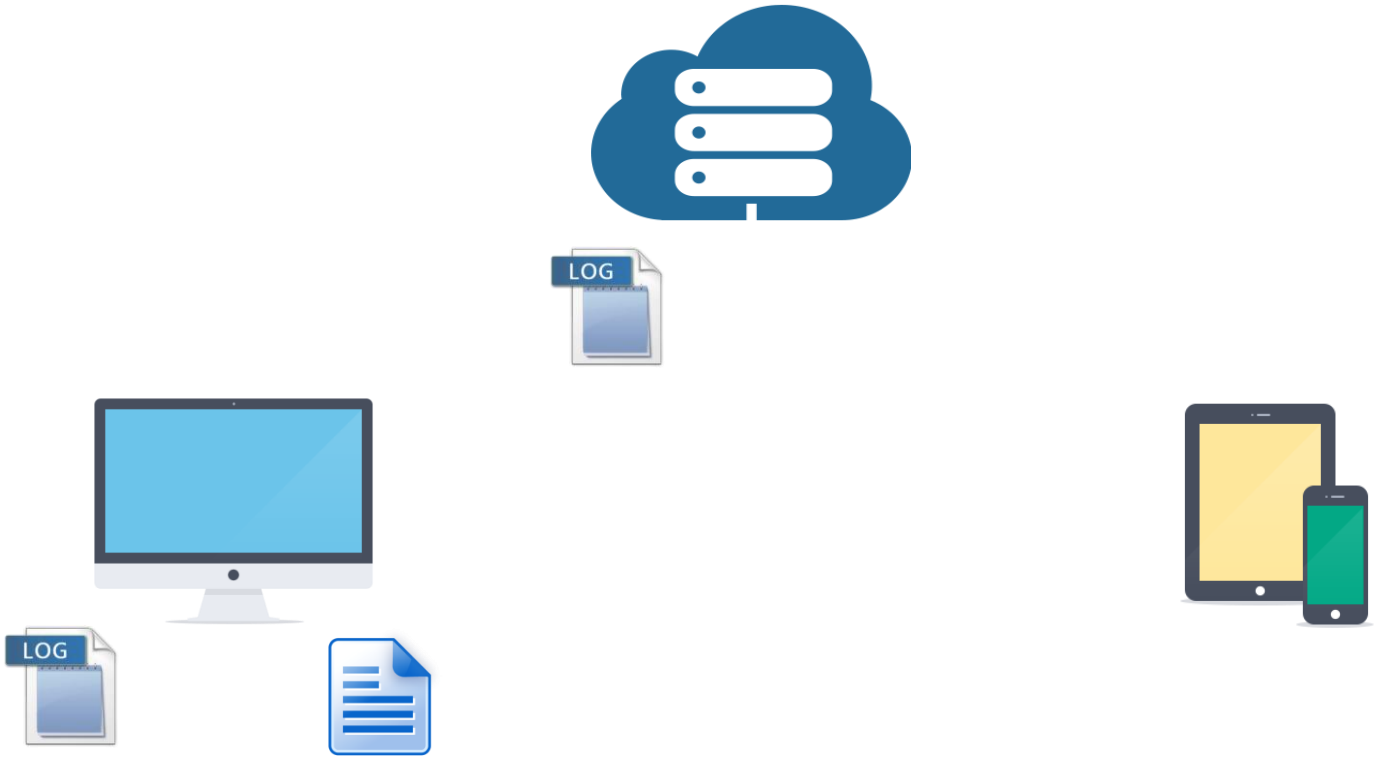
Available transfer methods

- How should a client read a particular version?



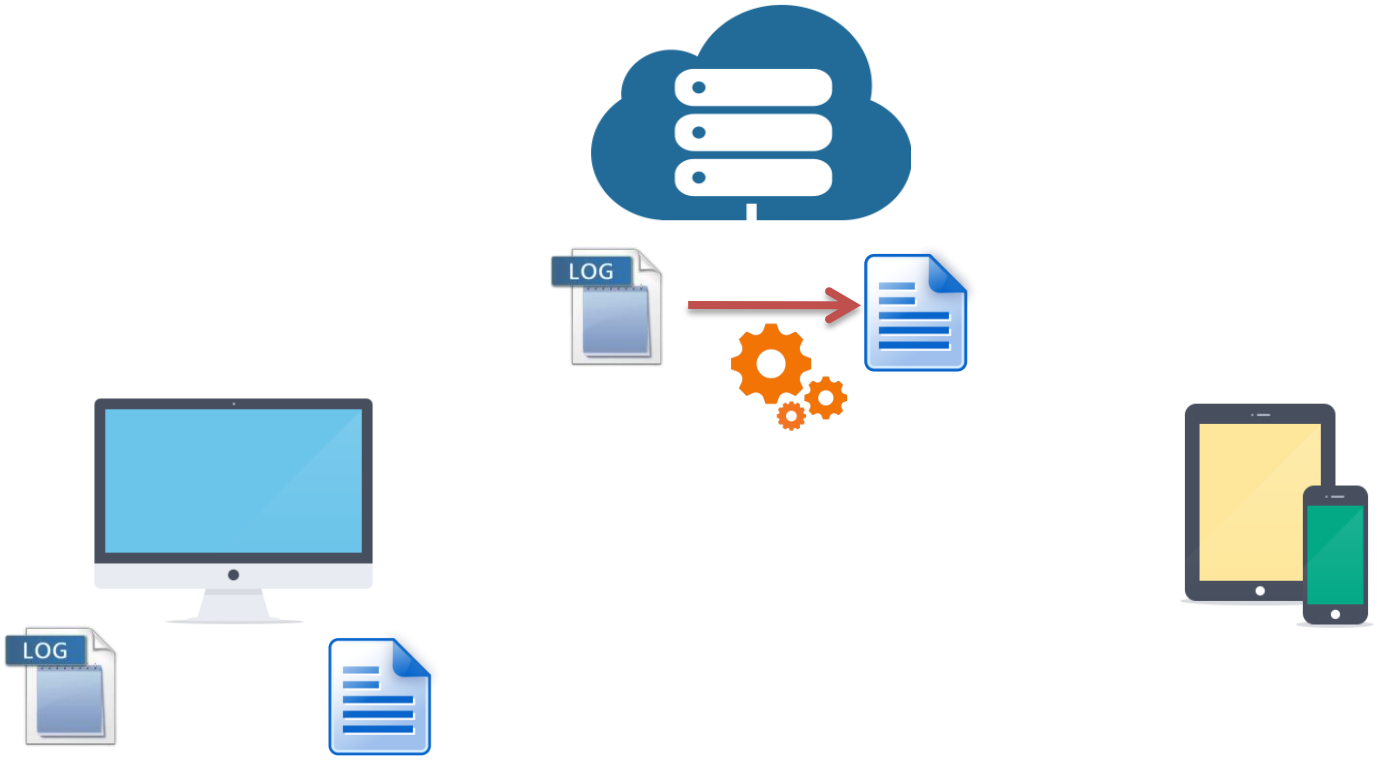
Available transfer methods

- How should a client read a particular version?



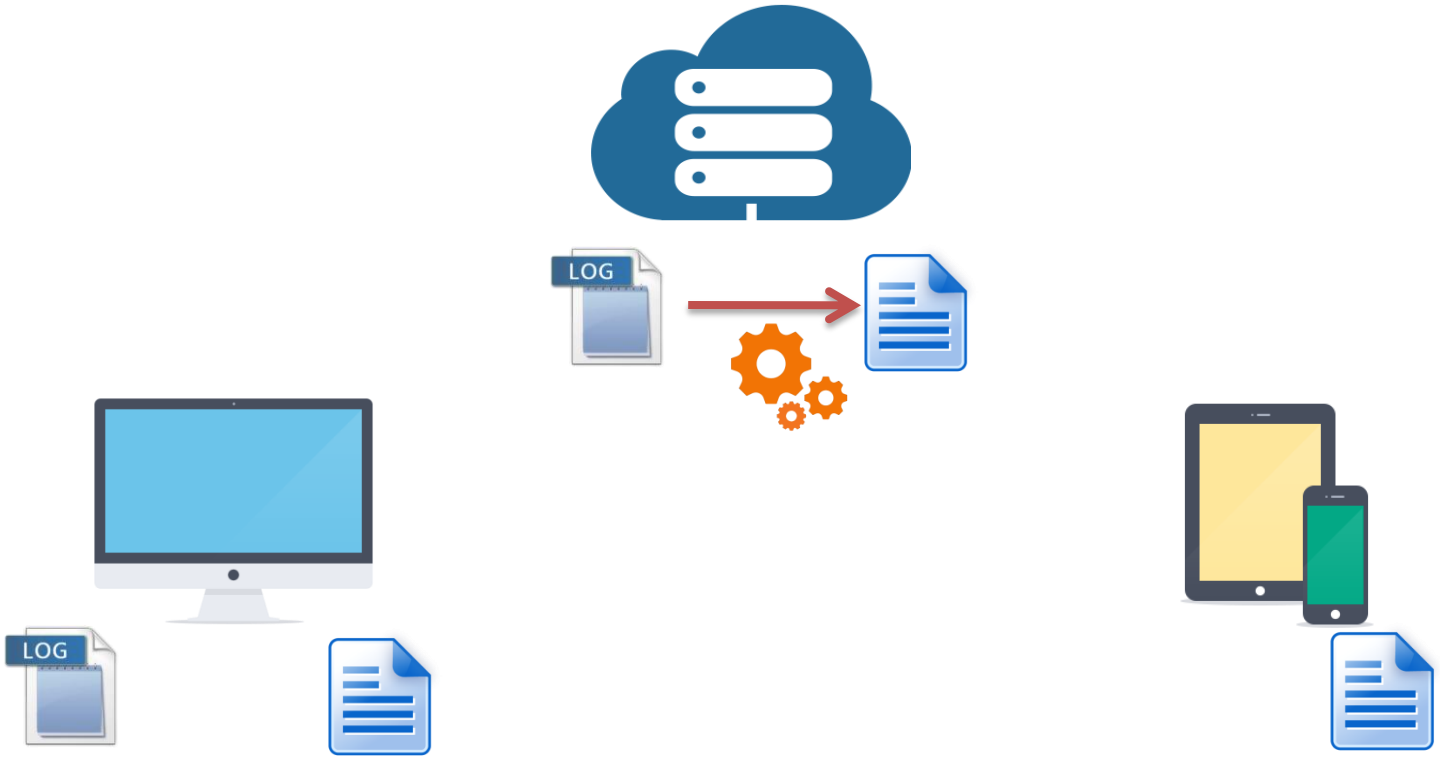
Available transfer methods

- How should a client read a particular version?



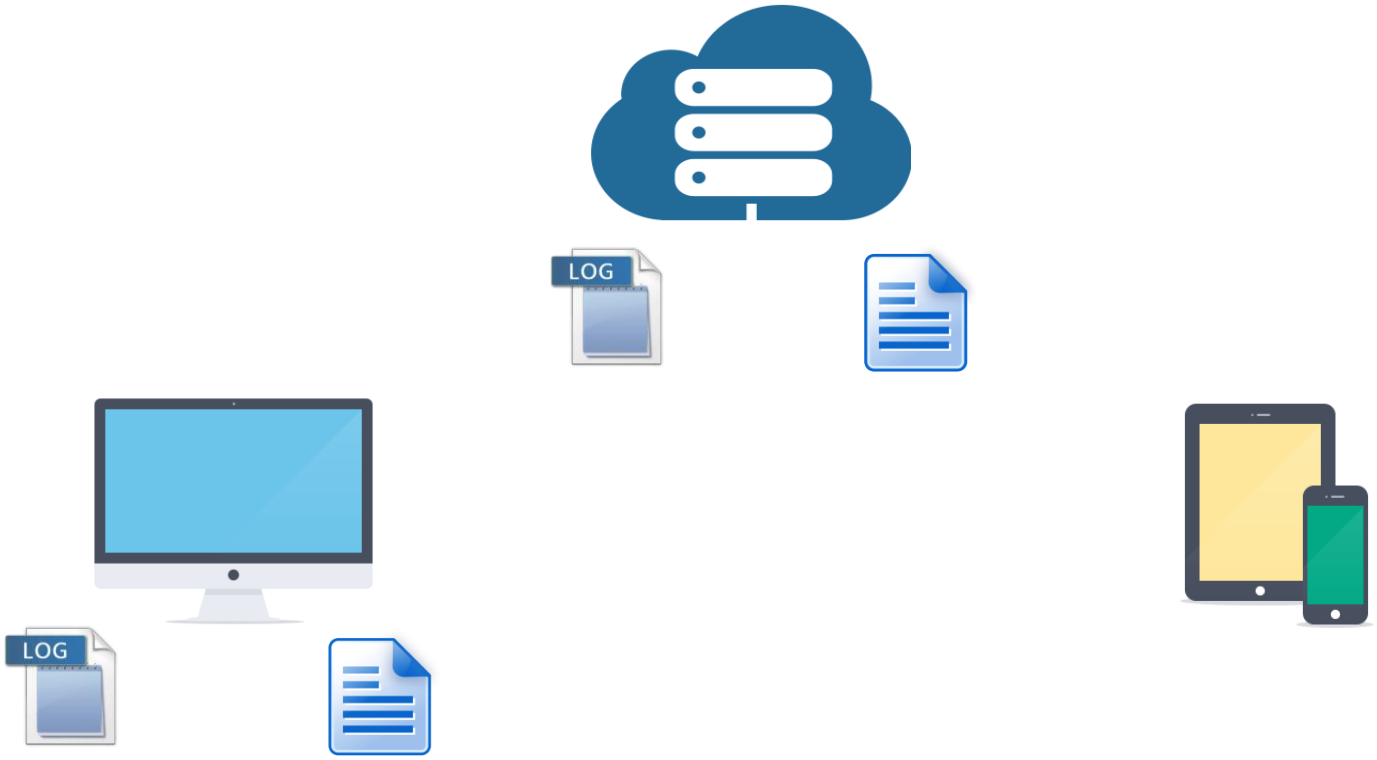
Available transfer methods

- How should a client read a particular version?



Available transfer methods

- How should a client read a particular version?



Available transfer methods

- How should a client read a particular version?

By value
By replay on the client
By replay on the server
From peers

Choosing the right method

- How should a client read a particular version?

- Server has the most complete knowledge
- Metrics
  - User waiting time
  - Monetary cost
  - Client energy consumption

Feasibility

- Eidetic system overheads
  - Record 4 years of workstation data on a 4TB hard disk
  - Under 8% performance overhead on most benchmarks
- Applications (log size vs. diff size)
  - Logs are smaller
    - image/audio editing, latex, make, slides editing
  - Diffs are smaller: text editing
- File sharing
  - Most files are not shared

Conclusions

- A new point in the design space of
  - Versioning file systems
  - Provenance-aware file systems
- Hypothesis
  - More effective in versioning and provenance
  - Achieving reasonable overheads
- Under implementation

Thank you!