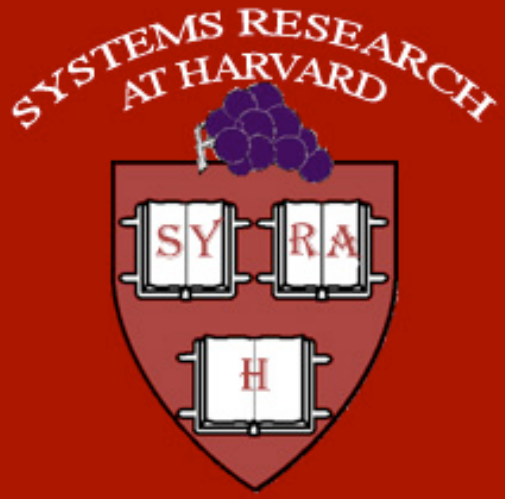
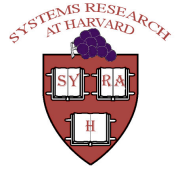
Layering in Provenance Systems



Kiran-Kumar Muniswamy-Reddy,
Uri Braun, David A. Holland, Peter Macko,
Diana Maclean, Daniel Margo, Margo
Seltzer, Robin Smogor

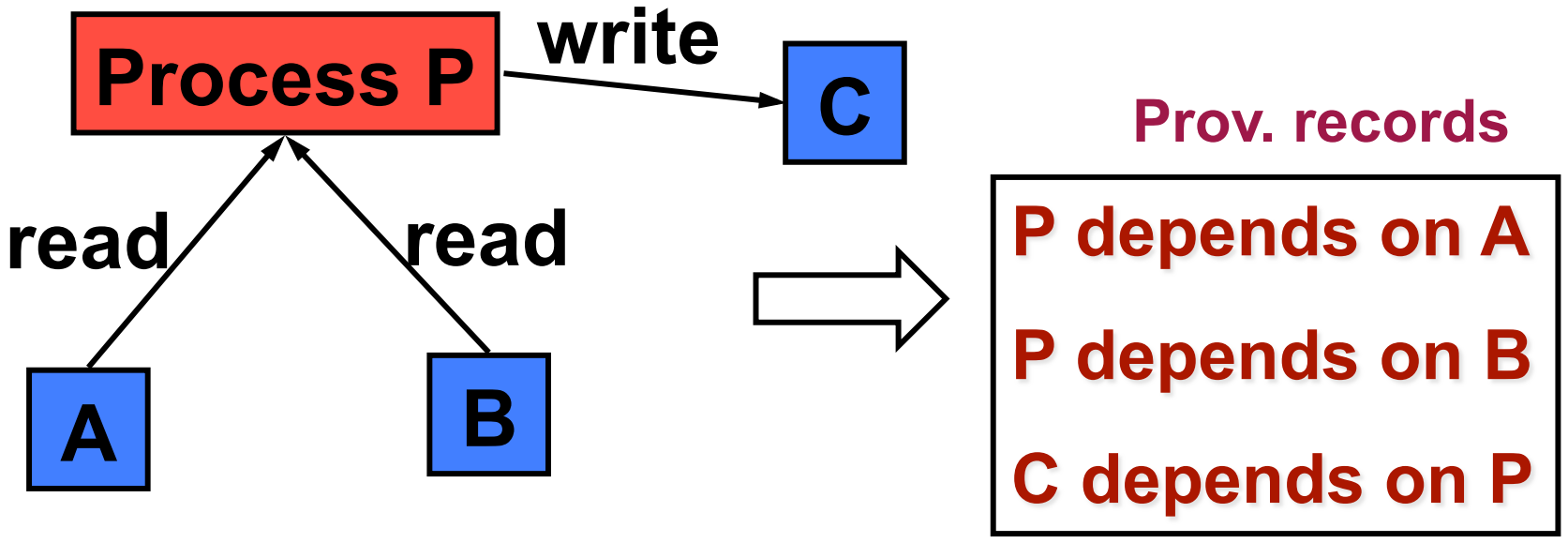
What is Provenance?

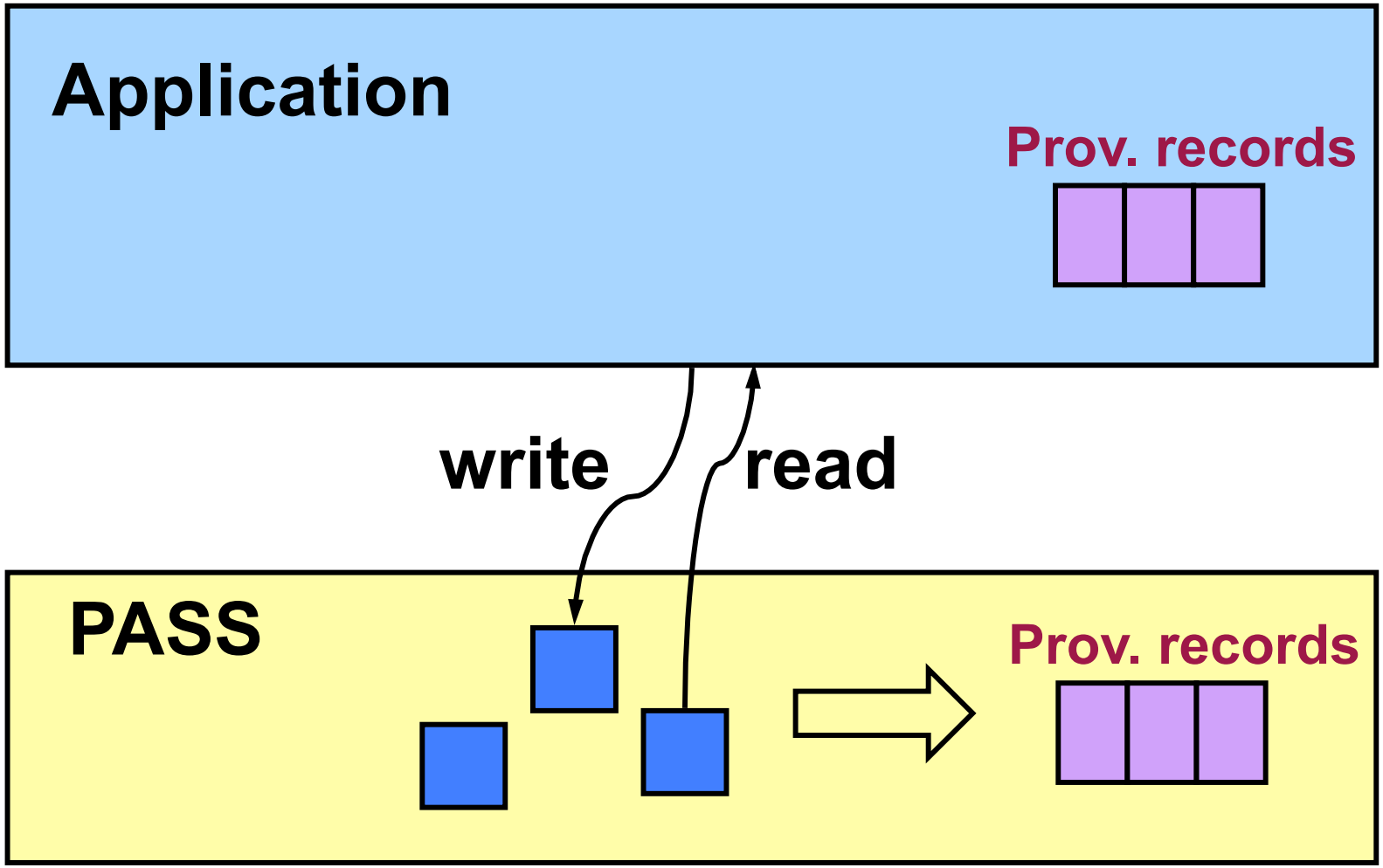
- Meta-data that describes the history of an object
  - What objects does this object depend on?
  - What applications modified/generated this object?
- Useful in various domains
  - Scientific reproducibility
  - Business compliance
  - Security

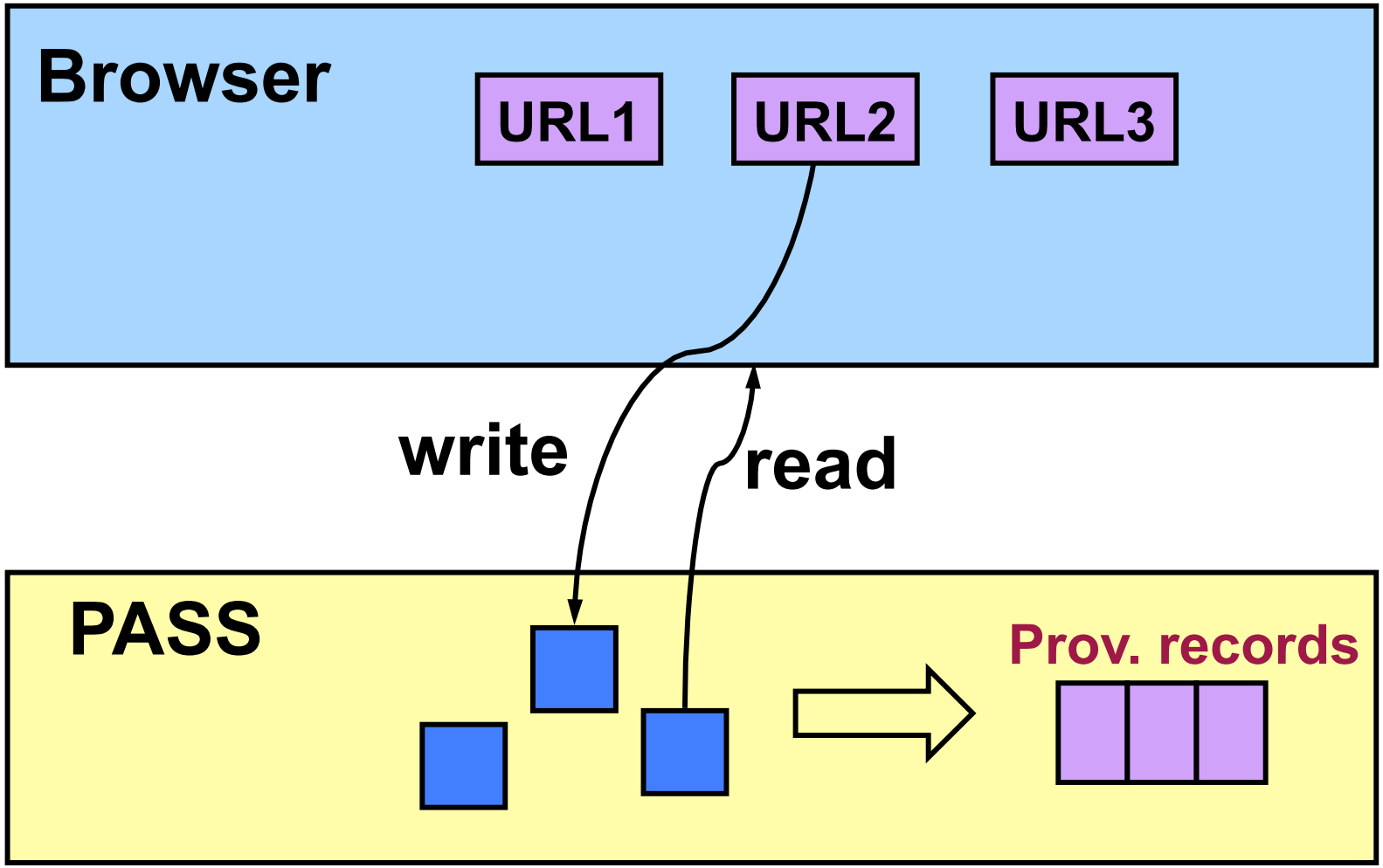


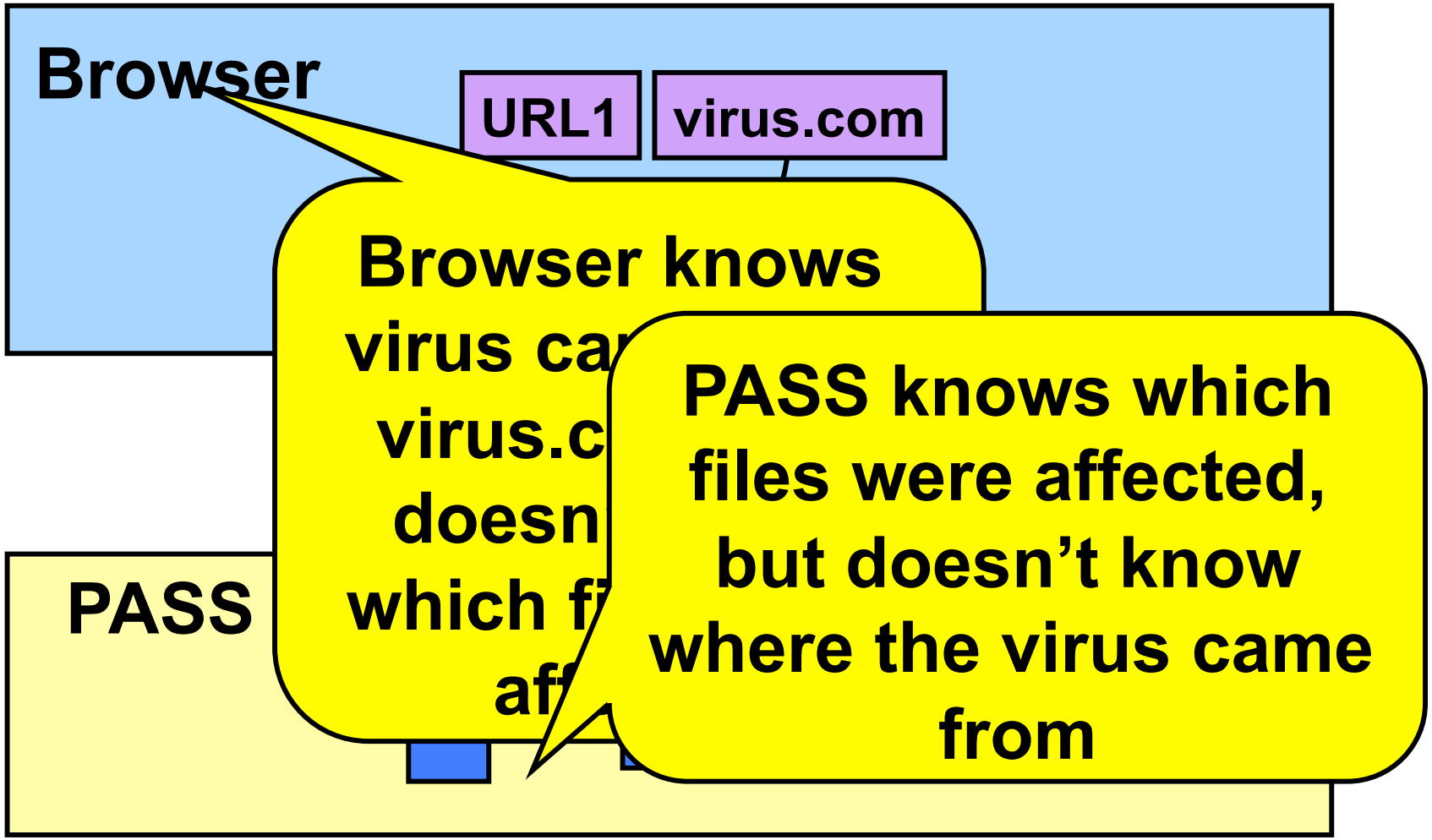
Provenance-Aware Storage System (PASS)

- Observes system calls that applications make and infers relationships between objects



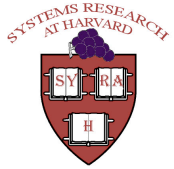

Application
Prov. records
write
read
PASS
Prov. records


Browser
URL1
URL2
URL3
write
read
PASS
Prov. records


Browser
URL1
virus.com
Browser knows
virus ca
virus.c
doesn
which f
af
PASS
PASS knows which files were affected, but doesn't know where the virus came from

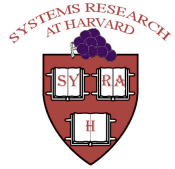
Provenance of each layer is important

- Each layer provides a provenance perspective that is unique and important
- Why not store all provenance in a centralized provenance repository?
  - Requires a mechanism to translate names across layers
  - Every layer must agree on naming convention



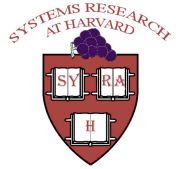
Integrating Provenance

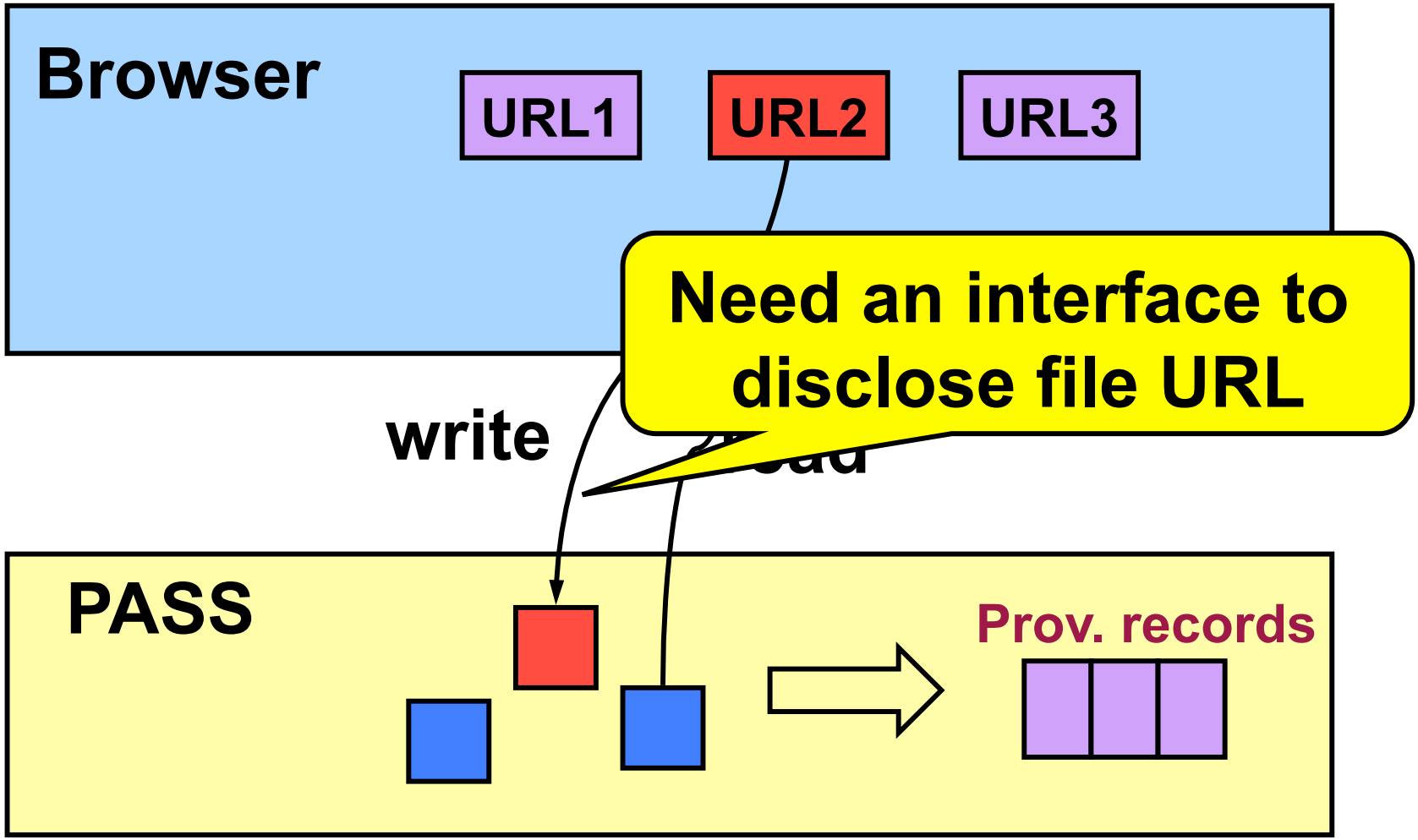
- Provenance systems in different layers should interact directly with one another and integrate provenance by linking objects
- This talk is about the issues, our approach, and our experience in solving it

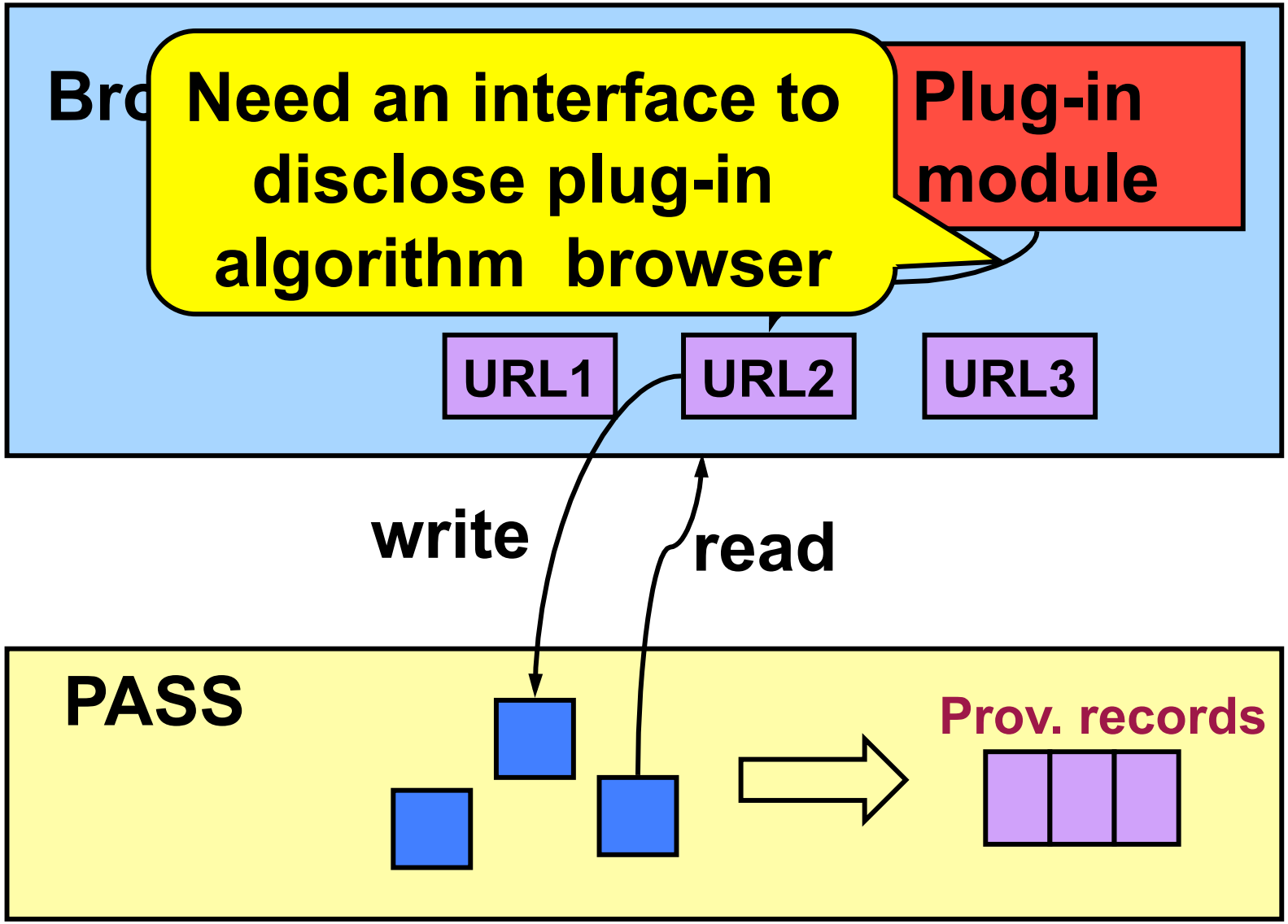


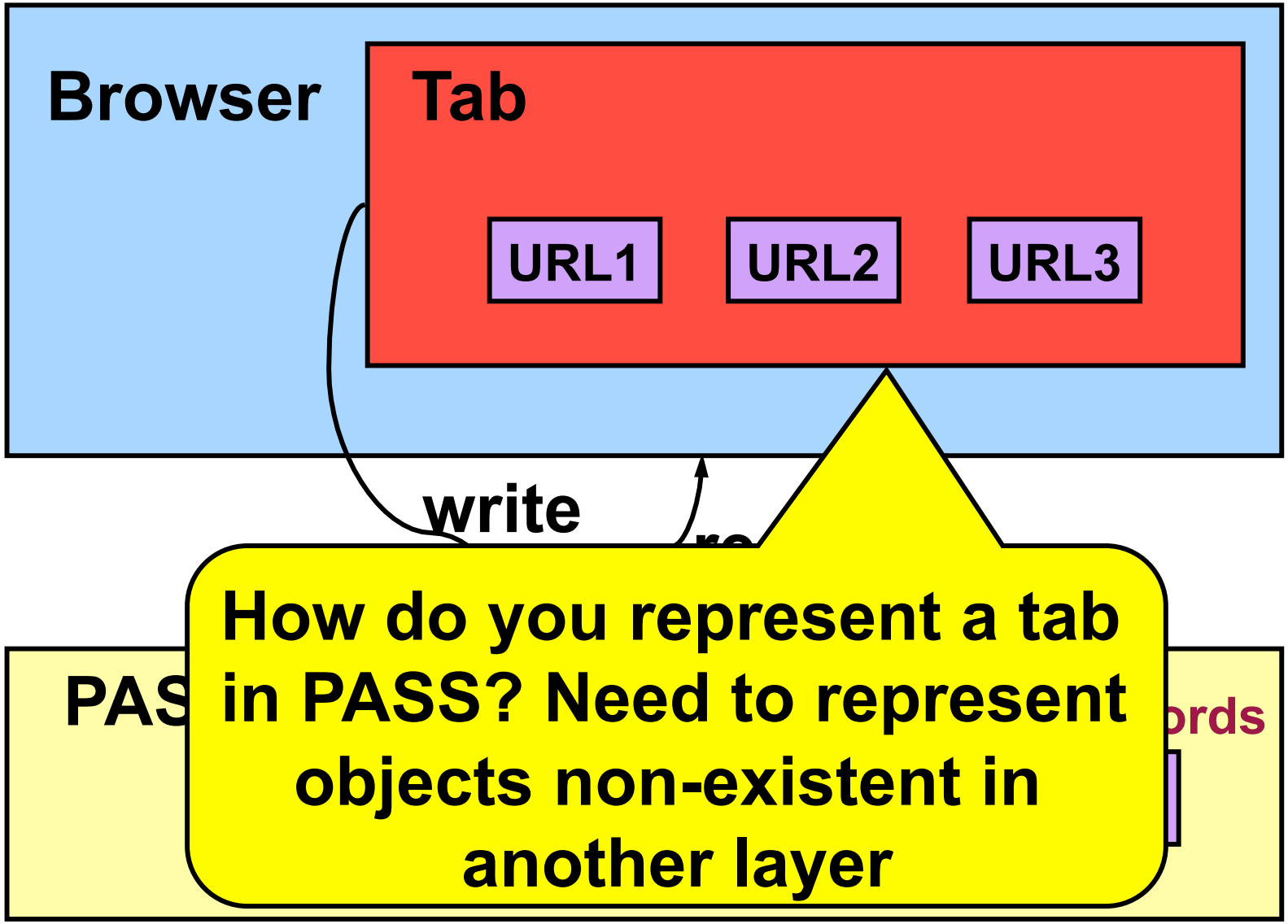
Outline

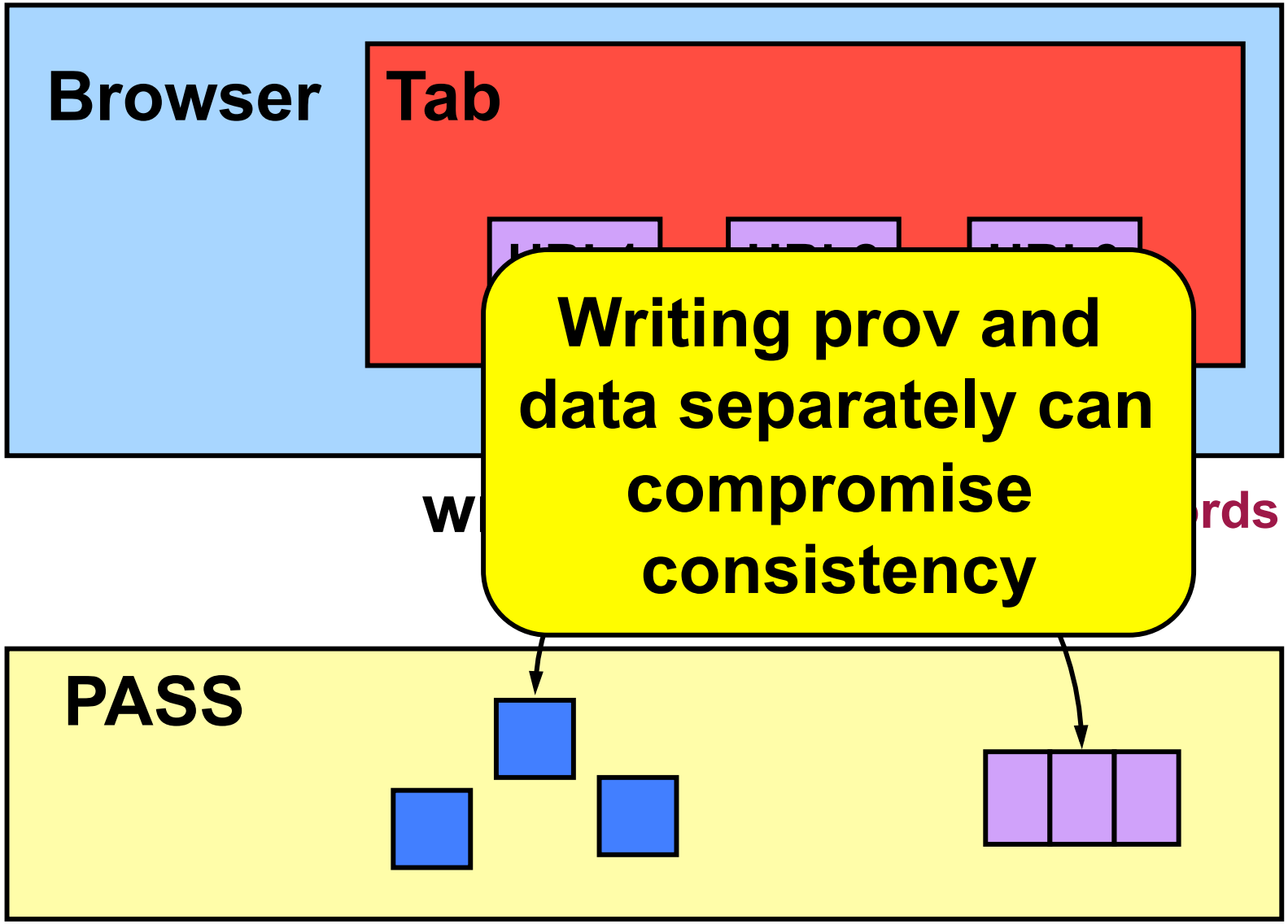
- Introduction
- **Challenges**
- Disclosed Provenance API
- Provenance-Aware Applications
- Lessons Learned
- Conclusions

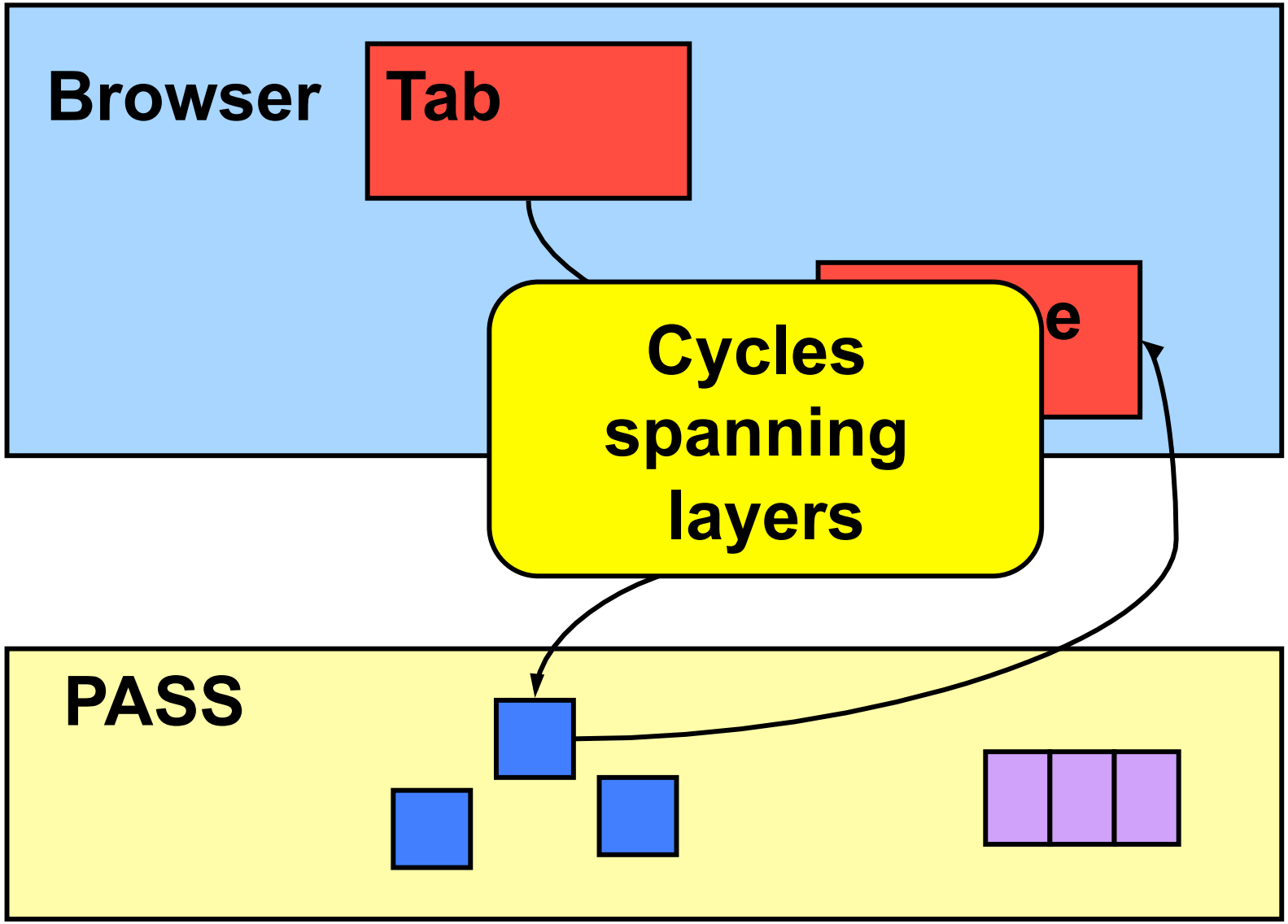



Browser
URL1
URL2
URL3
Need an interface to disclose file URL
write
PASS
Prov. records


Bro
Need an interface to disclose plug-in algorithm browser
Plug-in module
URL1
URL2
URL3
write
read
PASS
Prov. records

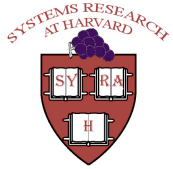

Browser
Tab
URL1
URL2
URL3
write
How do you represent a tab in PASS? Need to represent objects non-existent in another layer


Browser
Tab
Writing prov and data separately can compromise consistency
PASS


Browser
Tab
Cycles spanning layers
PASS

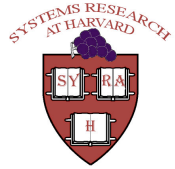
Naming

- How do we reconcile objects having different names in different layers?
  - An layer might treat a set of objects as one object



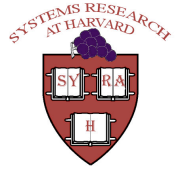
Challenges (summary)

- Interfacing between layers
- Represent objects in another layer
- Consistency
- Cycles
- Naming



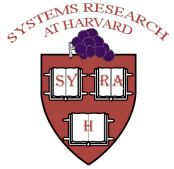
Outline

- Introduction
- Challenges
- **Disclosed Provenance API**
- Provenance-Aware Applications
- Lessons Learned
- Conclusions



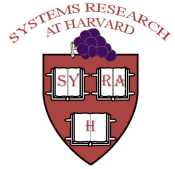
DPAPI: The Disclosed Provenance API

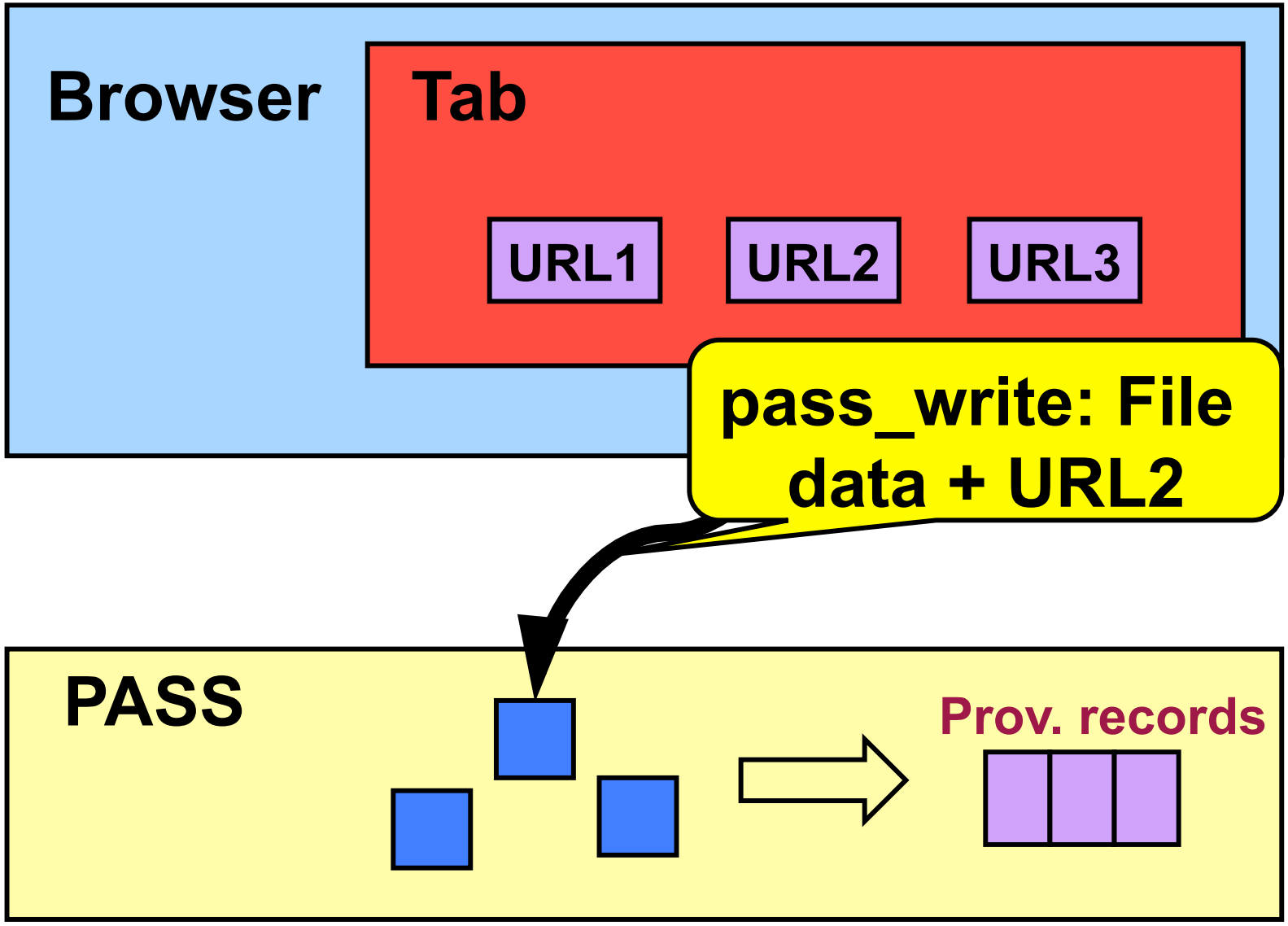
- Provide an API for higher layers to disclose provenance to lower levels
  - Six calls
- Used as the universal internal API between components in the PASS architecture
- Has evolved through three generations
- Exported to applications as a library



DPAPI Functions: Consistency

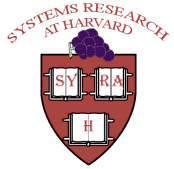
- Pass_read: Returns data with a reference to its provenance
  - Reference = object ID + version
- Pass_write: Writes data with provenance

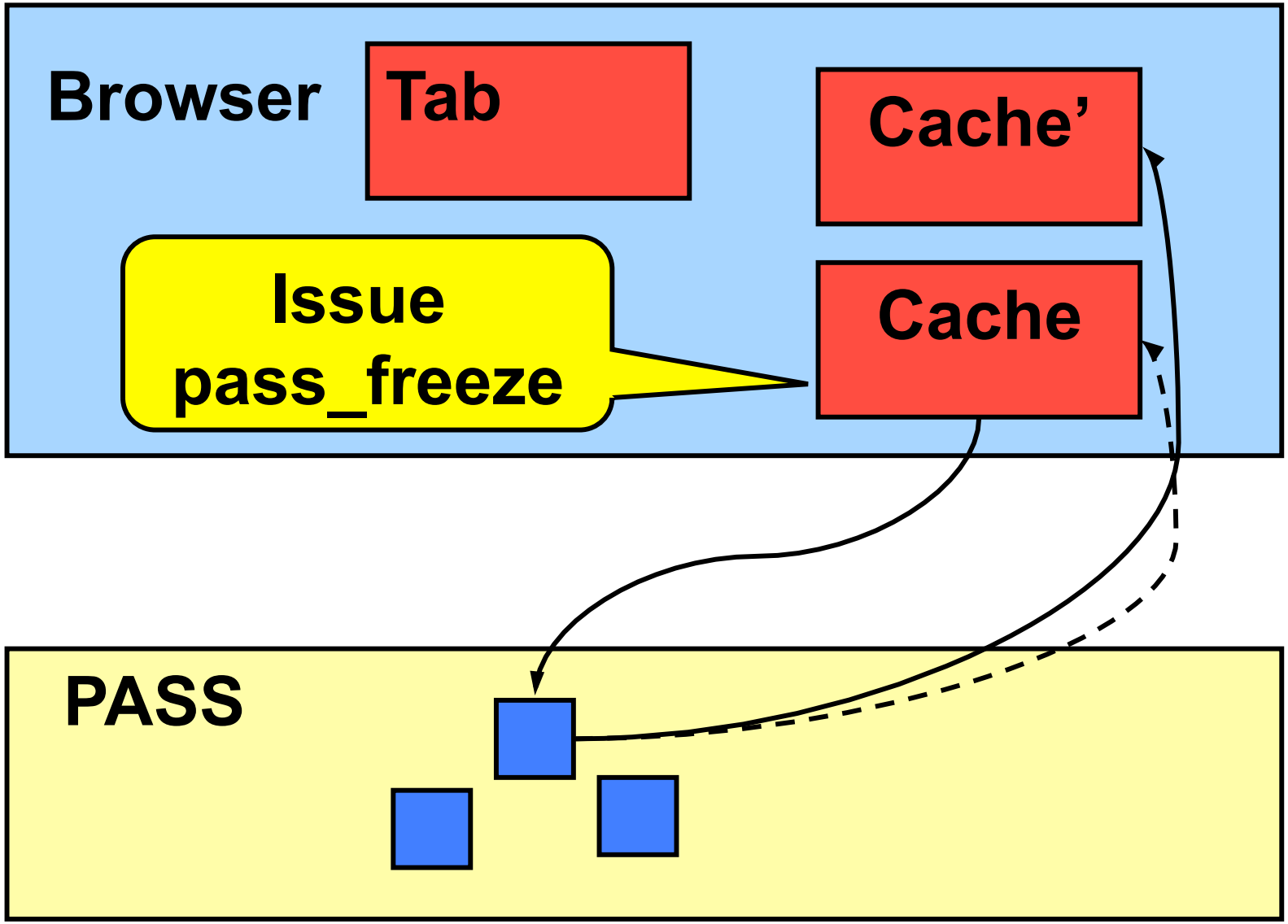



Browser
Tab
URL1
URL2
URL3
pass_write: File data + URL2
PASS
Prov. records

DPAPI Functions: Cycle Breaking

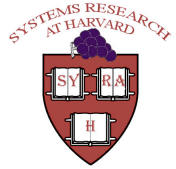
- Pass_freeze: creates a new version of object

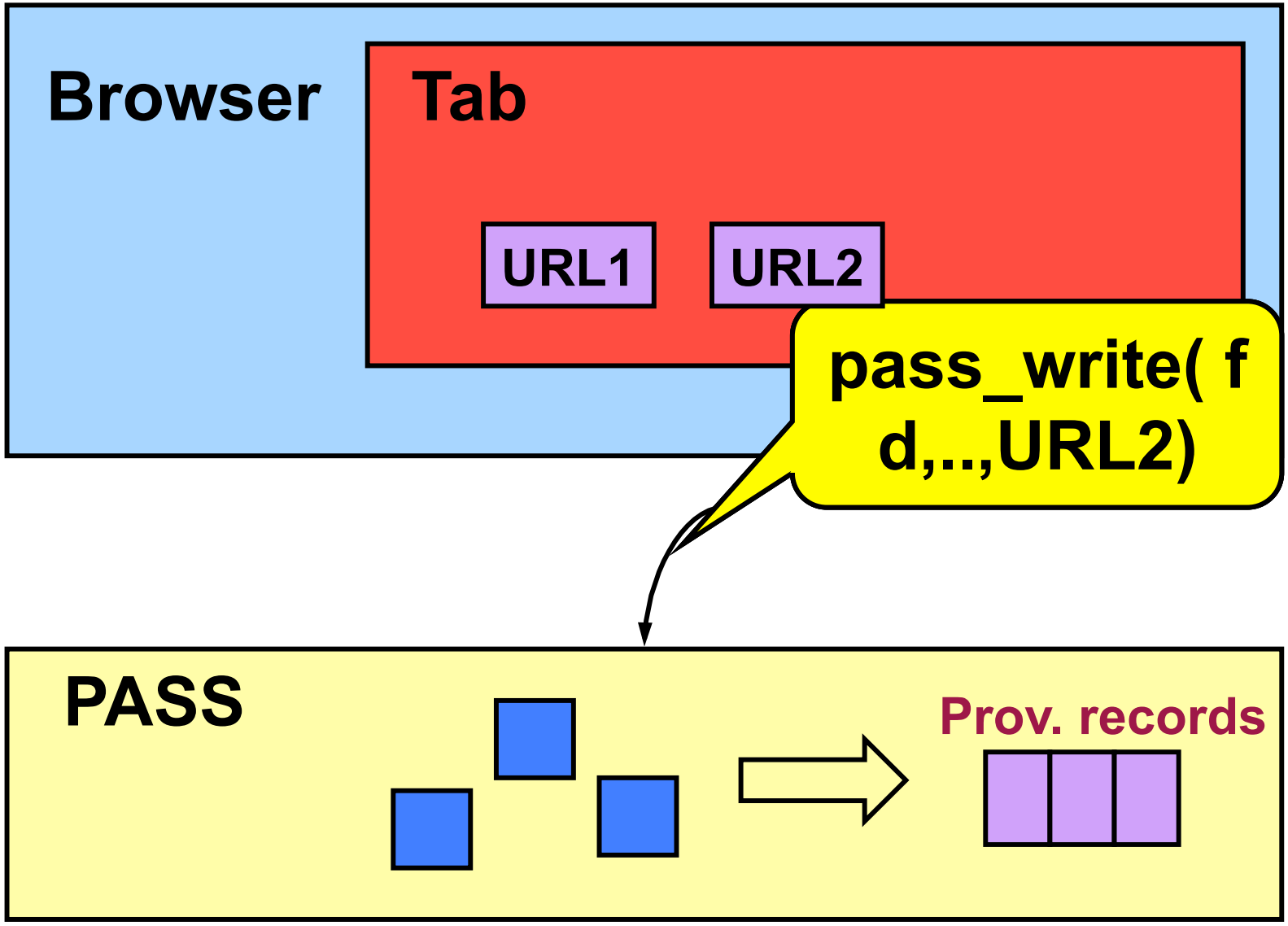



Browser
Tab
Cache'
Issue
pass_freeze
Cache
PASS

DPAPI Functions: Abstract Objects

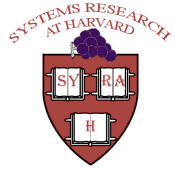
- Pass_mkobj: Create an object to represent something at a different abstraction layer
- Creates a logical object and returns a file handle
- Similar to a pipe: no name, no persistent data, can only store provenance
- Ex: represent browser tab, process, etc.




Browser
Tab
URL1
URL2
pass_write(f
d,..,URL2)
PASS
Prov. records

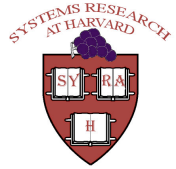
DPAPI Functions: Manipulating Abstract Objects (1)

- Process and Tab have conflicting needs
  - Tab: need to persist provenance
  - Process: capture provenance and cache it in memory till process actually generates data.
    - Avoid generating provenance for read-only workloads
- Pass_mkobj objects: by default provenance is cached in memory
- Pass_sync: Flush an object's provenance to disk



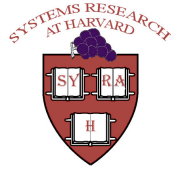
DPAPI Functions: Manipulating Abstract Objects (2)

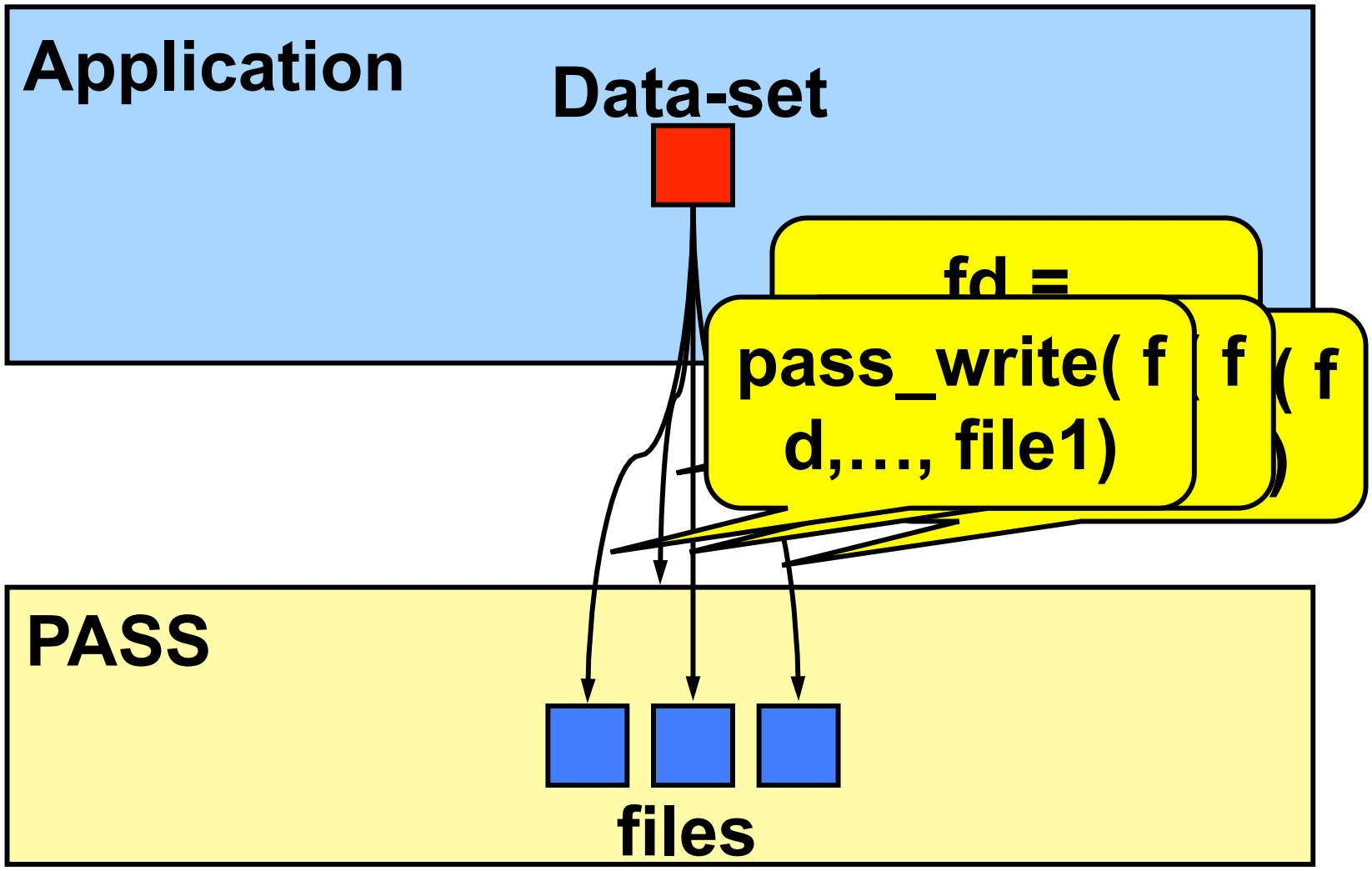
- Pass_reviveobj: Takes the object id and revives it
- Initially designed pass_mkobj objects to be one-time-use i.e., could never be accessed after a close
- Changed our minds after experience with browser tabs
  - Ex: Revive an object representing a tab



DPAPI Functions: Manipulating Abstract Objects (3)

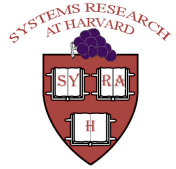
- Relate objects at one level to objects at another level
- Create an object using pass_mkobj and create dependencies between objects using pass_write
  - Ex: data-set object to its files




Application
Data-set
fd =
pass_write(f
d,…, file1)
f
(f
PASS
files

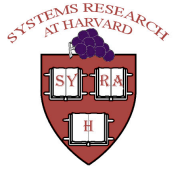
Outline

- Introduction
- Challenges
- Disclosed Provenance API
- **Provenance-Aware Applications**
- Lessons Learned
- Conclusions



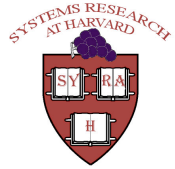
Provenance-Aware Applications

- Provenance-Aware: Applications augmented to disclose provenance to PASS
- We augmented the following applications
  - Links (text-based browser)
  - Kepler (Provenance workflow engine)
  - Python (run-time wrapper)



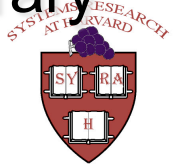
Provenance Aware-Kepler

- Provenance: operators used to generate data
- By default, stores provenance in file/ database
- Added extensions to store provenance using DPAPI



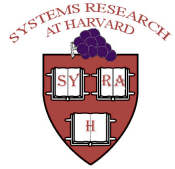
Use Case: Kepler

- Kepler tracks the operators that were used internally for producing an output
- Scenario: Library upgrade corrupts some of the operators
- Without Integration:
  - Kepler knows which files were affected by corrupt operator
  - PASS knows which files were affected by library upgrade
- With Integration:
  - Can identify files that were affected by both the library upgrade and corrupt operator



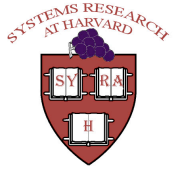
Provenance Aware Python

- Provenance: internal functions/algorithms invoked in computing results
- A set of wrappers that track provenance in Python applications
- A set of Python bindings for DPAPI
- Applications similar to Kepler



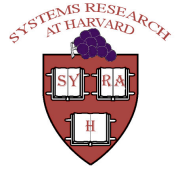
Outline

- Introduction
- Challenges in layering
- Background
- Disclosed Provenance API
- Provenance-Aware Applications
- **Lessons Learned**
- Conclusions



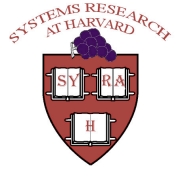
Lessons Learned

- Application architecture dictates how difficult this is
  - Firefox's modular architecture makes it difficult to have provenance and data flow together through the browser
- APIs are never done
  - DPAPI continues to evolve
  - Added two new calls early in 2009



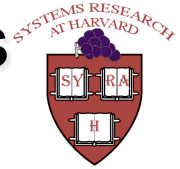
Lessons Learned (2)

- Differentiating applications from substrates:
  - We initially thought that our Python wrappers made *Python* provenance-aware
  - Instead they enabled provenance-aware *Python applications*
  - Making *Python* provenance-aware requires changes to the interpreter -- similar to those to make an operating system provenance-aware



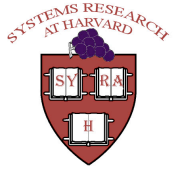
Lessons Learned (3)

- Guidelines for making applications or systems provenance-aware:
  - Identify *what* provenance you want to collect
    - Replace read calls with pass_read calls
    - Replace write calls with pass_write calls
  - To capture semantic provenance
    - Create objects as necessary using pass_mkobj
    - Accumulate provenance records for those objects
    - Use pass_write to relate objects
  - If necessary, export DPAPI to *higher* layers



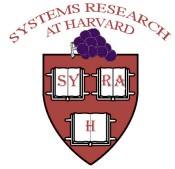
Outline

- Introduction
- Challenges in layering
- Disclosed Provenance API
- Provenance-Aware Applications
- Lessons Learned
- **Conclusions**



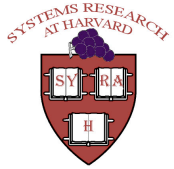
In the paper..

- Re-designed PASS System Architecture
- NFS protocol extensions to support DPAPI
- PQL – query language
- Evaluation
  - Results for: Linux compile, Postmark, Blast, user activity, Kepler workload
  - Overheads were reasonable (max 23%)



Conclusions

- Provenance is useful at all layers of the system:
  - Capture semantics of applications
  - Capture system dependencies
- Integrating provenance across layers is important!
- We provide a framework for solving this

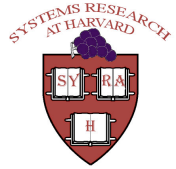


Questions?

Contact:

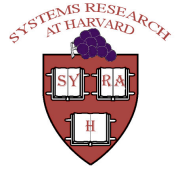
pass@eecs.harvard.edu

www.eecs.harvard.edu/~pass



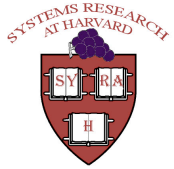
DPAPI (detail)

```
int dpapi_freeze(int fd);
int dpapi_mkobj(int reference_fd);
int dpapi_revive_obj(int reference_fd,        __pnode_t
    pnode, version_t version);
ssize_t paread(int fd, void *data, size_t datalen, __pnode_t
    *pnode_ret, version_t *version_ret);
ssize_t pawrite(int fd, const void *data,
   size_t datalen, const struct dpapi_addition *records,
    unsigned numrecords);
int dpapi_sync(int fd);
```

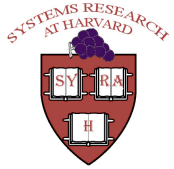
Provenance Aware links

- Text based browser
  - Chose it due to its simplicity
- Captures
  - URL of downloaded file
  - Sequence of webpages visited before download
  - Webpage a user was viewing on download



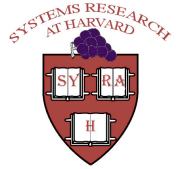
Provenance Aware links

- Group provenance by session
  - Create a PASS object using pass_mkobj
  - For every visited site, record a VISITED_URL and record using pass_write
- On download, write 3 records using pass_write
  - dependency between file and session
  - dependency between file and url
  - dependency between file and current_url



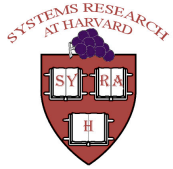
Provenance-Aware Python App

- A set of wrappers to track provenance in Python applications
  - Wrap objects, modules, basic types, and output files
- Create Python bindings for DPAPI



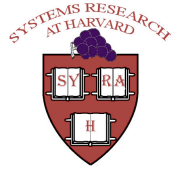
Provenance-Aware Python App

- Wrapper creates a pass object for every wrapped object
- Intercepts method invocations
  - Create records that connect method invocations to inputs and outputs
- Record these records using pass_write



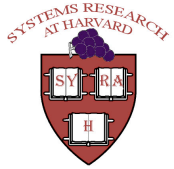
Provenance-Aware Kepler

- Kepler is a scientific workflow engine
- Records provenance in a text file/database
- Added the option of recording provenance using DPAPI



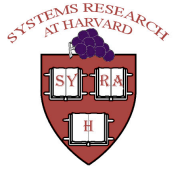
Provenance-Aware Kepler

- Create a pass object for every workflow operator using pass_mkobj
- Record provenance whenever an operator produces a result
  - We issue pass_write on such instances
- For file operations, we had to modify its source and sink operations



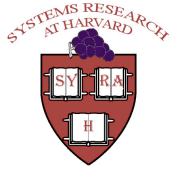
Provenance Systems

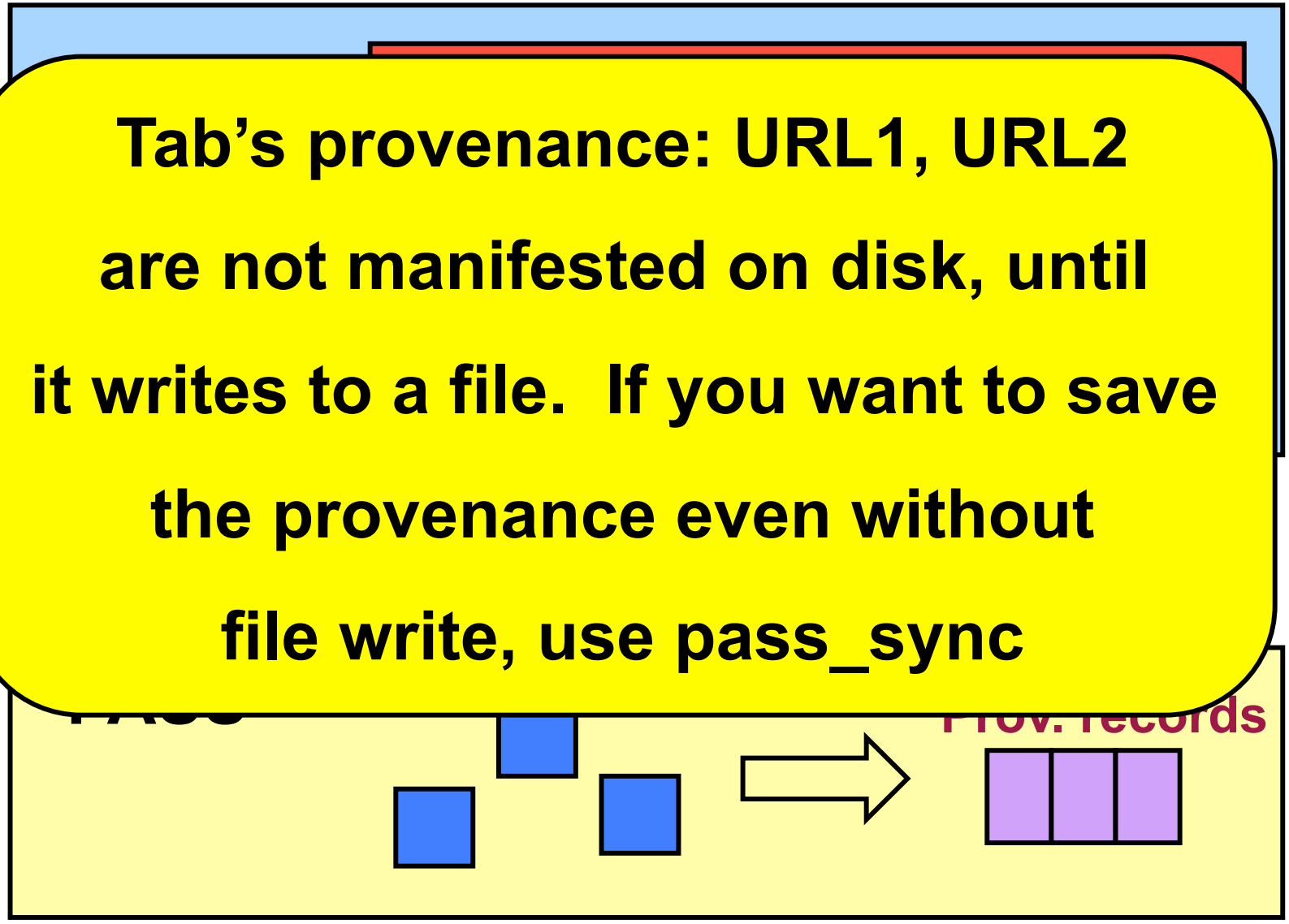
- Operate at different layers
  - System-call level: files
  - Database systems: tuples
  - Workflow engines: objects
  - Applications:
    - Variable (from an interpreter)
    - Links (from a browser)



Naming

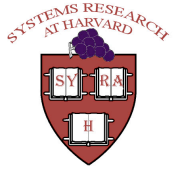
- How do we reconcile objects having different names in different layers?
  - Browser can process data internally referencing the object by its URL
  - PASS references the object using its object-ID/name



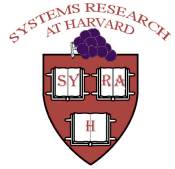

Tab's provenance: URL1, URL2
are not manifested on disk, until
it writes to a file. If you want to save
the provenance even without
file write, use pass_sync
Prov. records

DPAPI Functions: Manipulating Abstract Objects (3)

- Relate objects at one level to objects at another level
  - Create an object using mkobj and create dependencies between objects using pass_write
  - Ex: URL and file name



```
int url_fd = pass_mkobj();
pass_write(url_fd, NULL, 0, “URL=URL1”);
int file_fd = open(“URL_FILE”);
/*… create a record ‘rec’ that says that url_fd is a
   descendant of file_fd */
/* now write the record */
pass_write(file_fd, NULL, 0, url_fd);
/* the record links file_fd and url_fd, so users can
   query at whatever level is most convinient*/
```

```
/* create an object corresponding to the dataset */
int ds_fd = pass_mkobj();
pass_write(ds_fd, NULL, 0, “NAME=DS-NAME”);
for (i = 0; i < n; ++i) {
    int file_fd = open(File i in dataset);
    /*… create a record ‘rec’ that says that ds_fd is a
    ancestor of file_fd. the record links file_fd and
    ds_fd, so users can query at whatever level is
    most convinient */
    /* now write the record */
    pass_write(file_fd, NULL, 0, rec);
}
/* continue accumulating provenance for ds_fd.. */
```