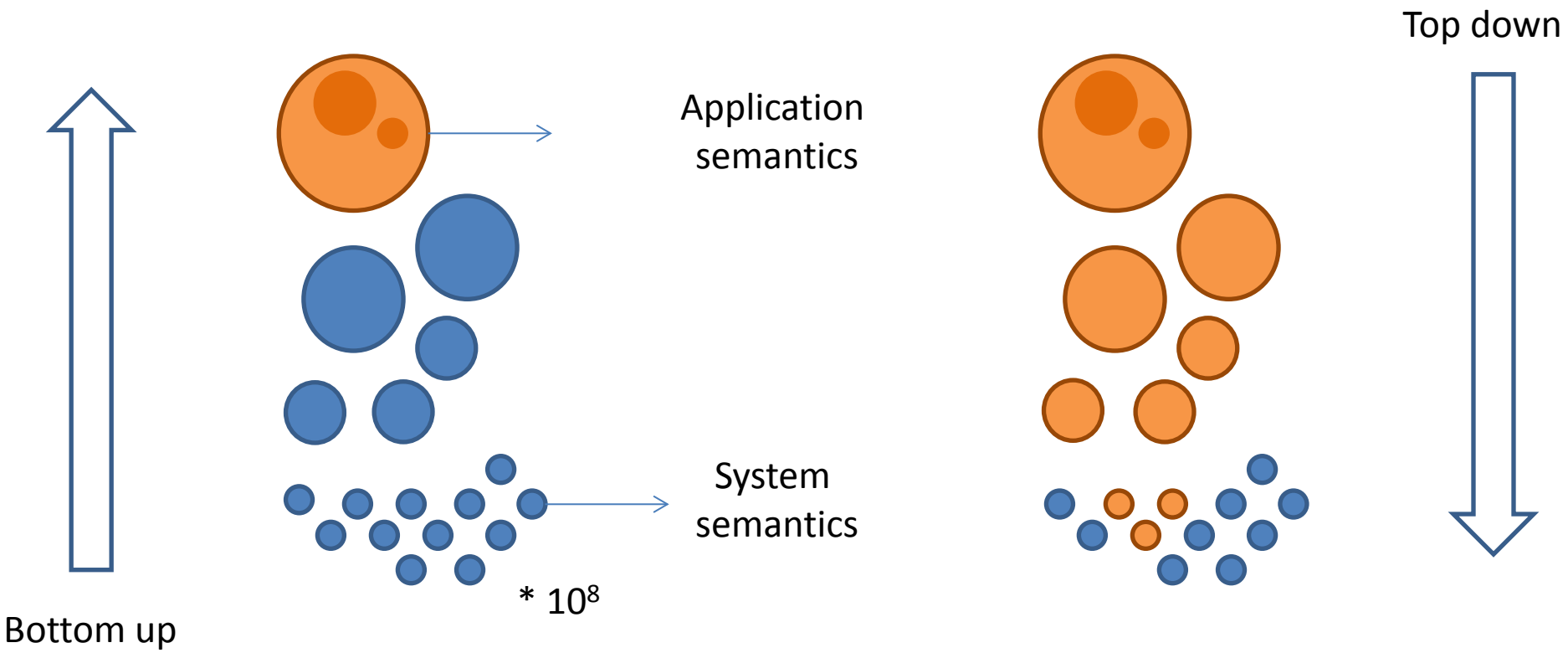
Lucian Carata, Ripduman Sohan, Andrew Rice, Andy Hopper

IPAPI: Designing an Improved Provenance API

Lucian Carata

UNIVERSITY OF CAMBRIDGE

Why do we need provenance APIs?



Why do we need *better* provenance APIs?

- Current approaches: CPL, DPAPI*
  - Centralized philosophy: *"provenance far away from data"*
- Do not really consider:
  - Sub-file granularities
  - Automation
  - Pre-existing provenance data (e.g. logs)

Why do we need *better* provenance APIs?

- We need flexibility/generality to drive adoption
  - less infrastructure requirements (e.g db services)
  - more clear guarantees (completeness, durability, overheads); explicit tradeoffs

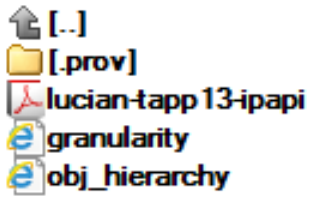
Overview: IPAPI

- A C++ **library** that you link into your application, some **tools** to manage provenance.

- Not dependent on any existing system services

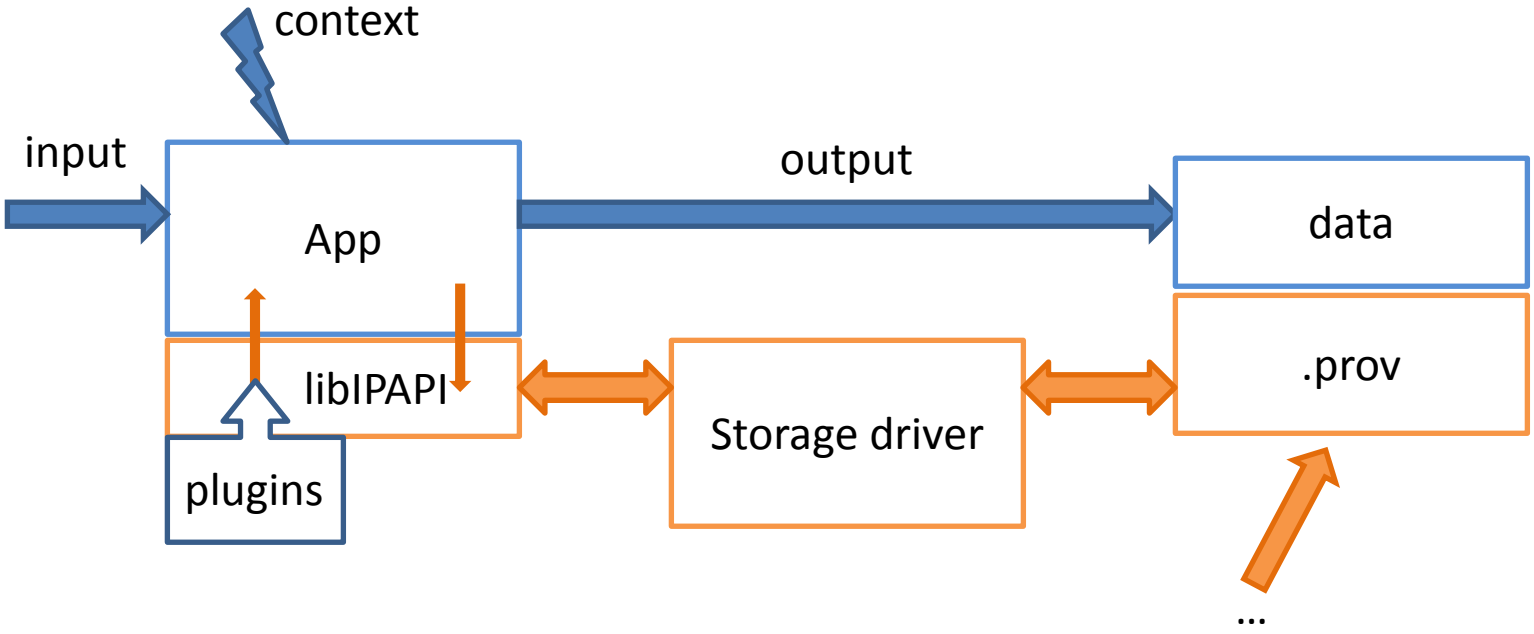
Overview: IPAPI

- **Decentralized** (distributed provenance repositories)
  - Provenance is closer to data (and can move together with it)



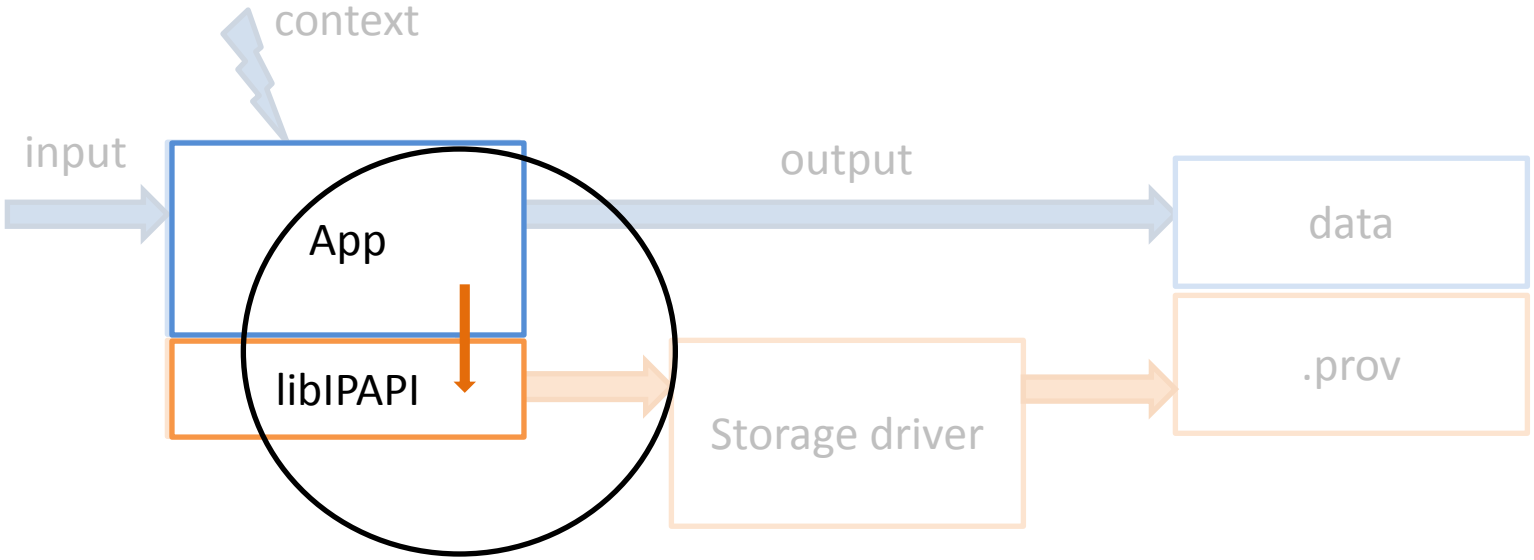
- Developed with automation in mind:
  - "how can other applications use the provenance disclosed by my application?"

IPAPI Structure



- Link with `-lipapi` and include `ipapi.h` in one of your source files.
- You are now tracking (basic) process provenance
- Almost no overhead (minimal increase in startup time)
- Need more? Actually call the API functions.

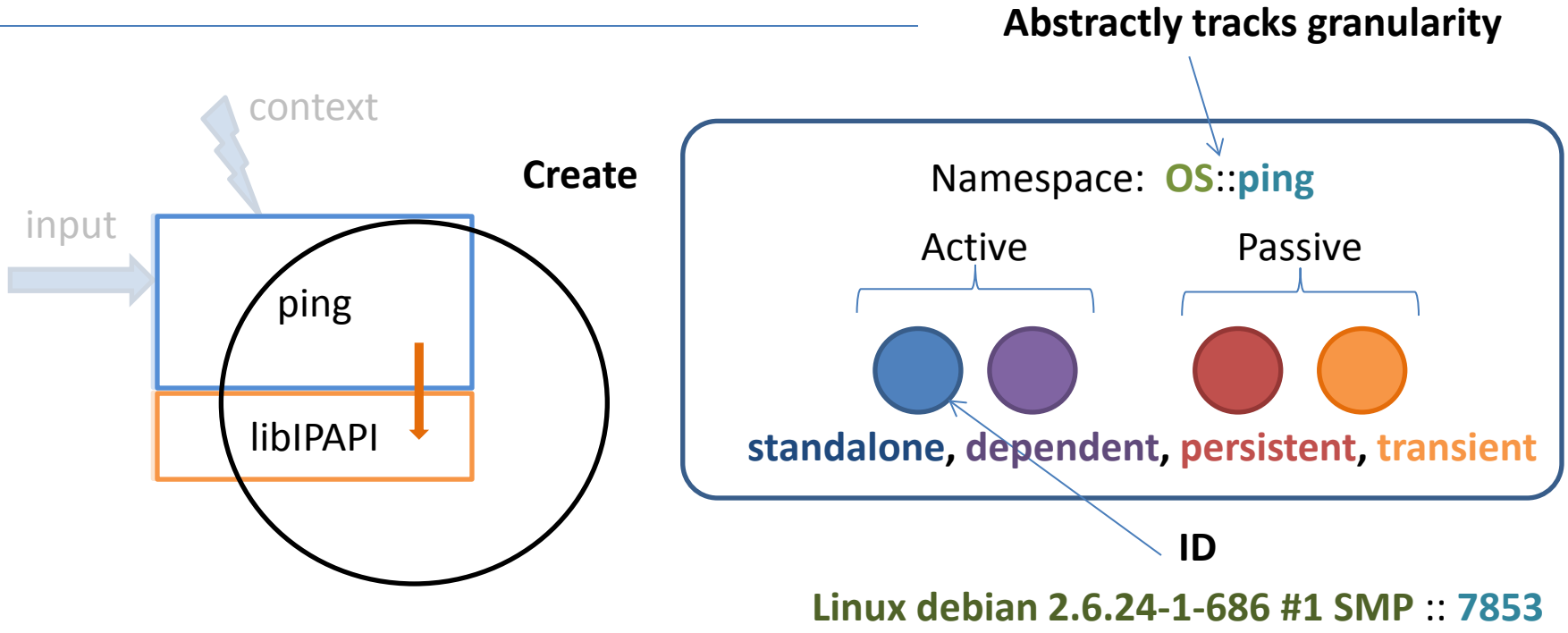
IPAPI Structure



Application calling IPAPI

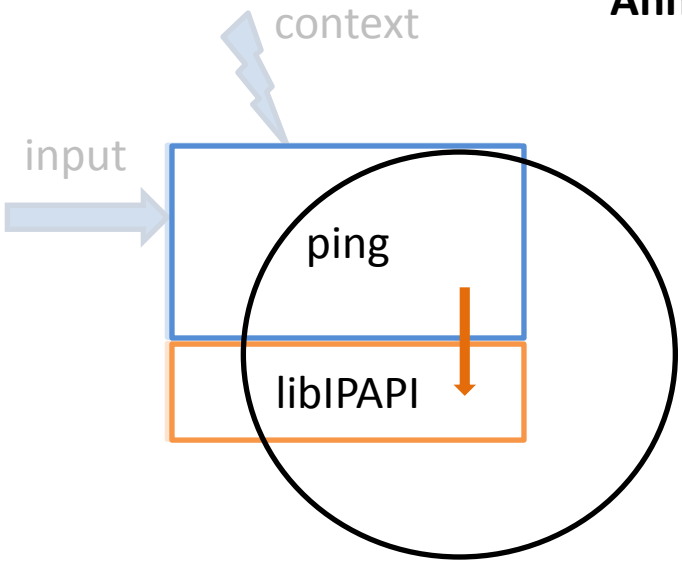
Focusing in on the interface offered by the API

IPAPI Structure

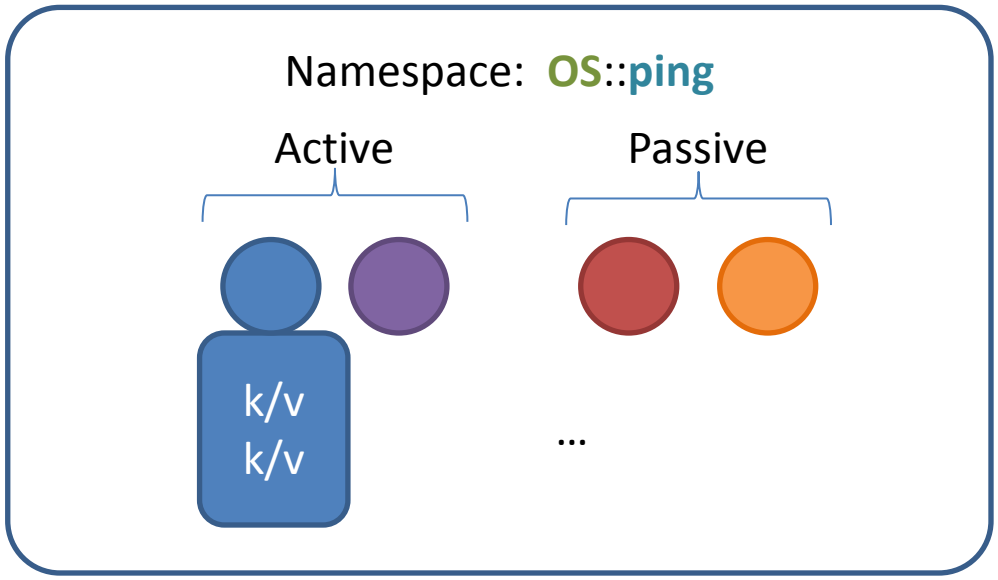


- An entity model that maps to system objects (processes, files, pipes, sockets)

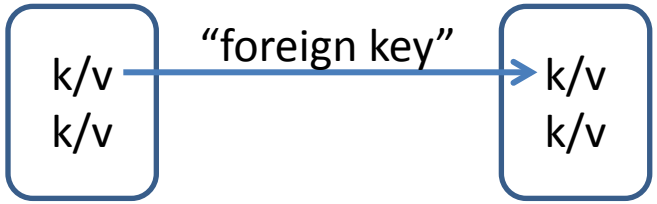
IPAPI Structure



Annotate



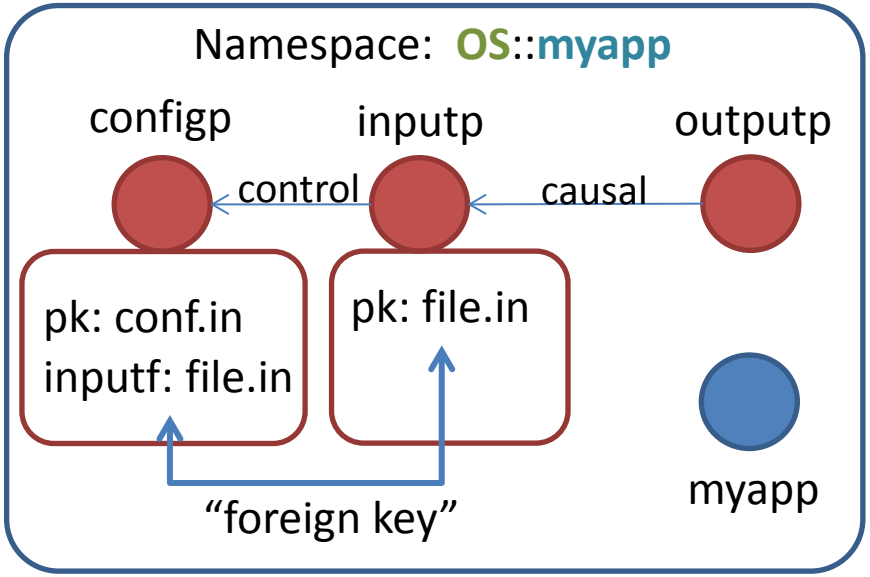
Relate



Example

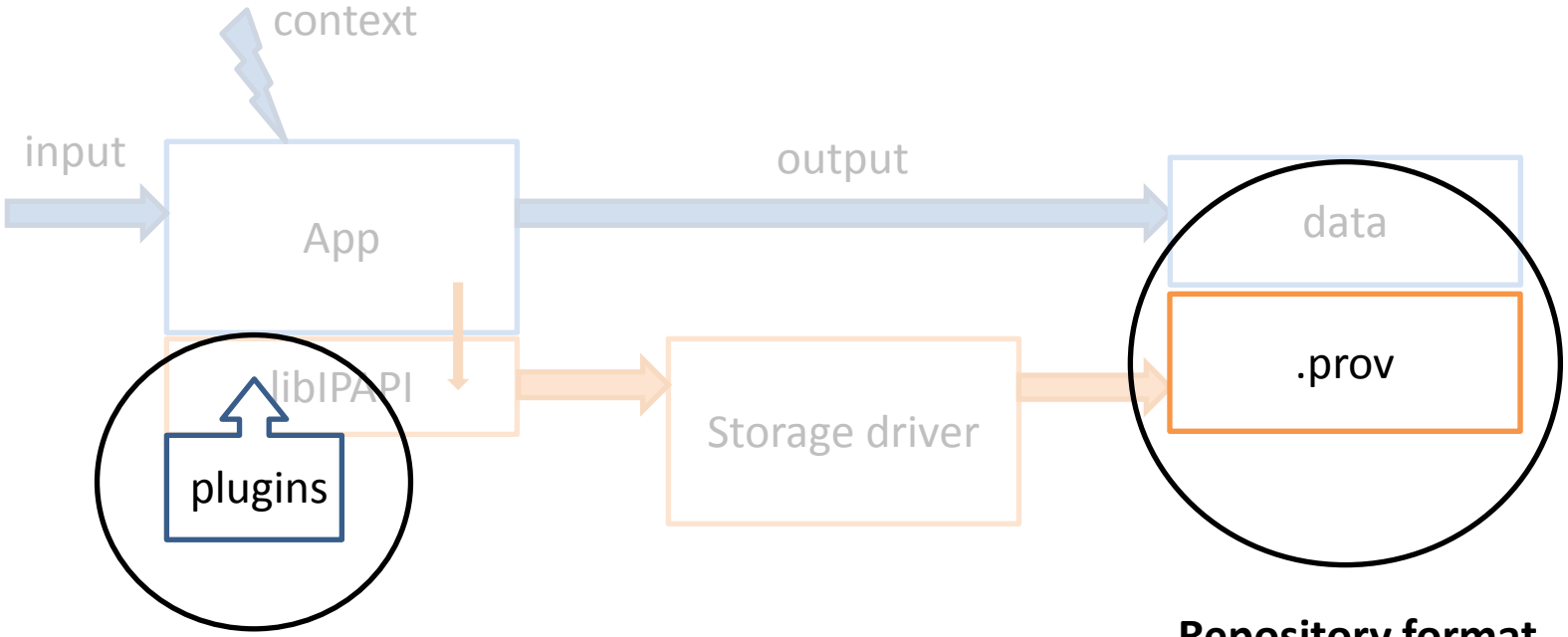
"This program gets the name of its input from a configuration file"

```
ifstream cfgin("config.in");
cfgin>>fname;
ifstream in(fname);
//… process in and write to out
ofstream out("out.dat");

Pobject<ifstream> configp, inputp;
Pobject<ofstream> outputp;

configp.addkv("inputf","file.in");
inputp.pk.key_relation(configp,"inputf");
outputp.obj_relation(CAUSAL, inputp);
```



- Applications are able to create their own provenance-aware objects. They need to respect the `is_prov_aware` trait

Some Design Decisions



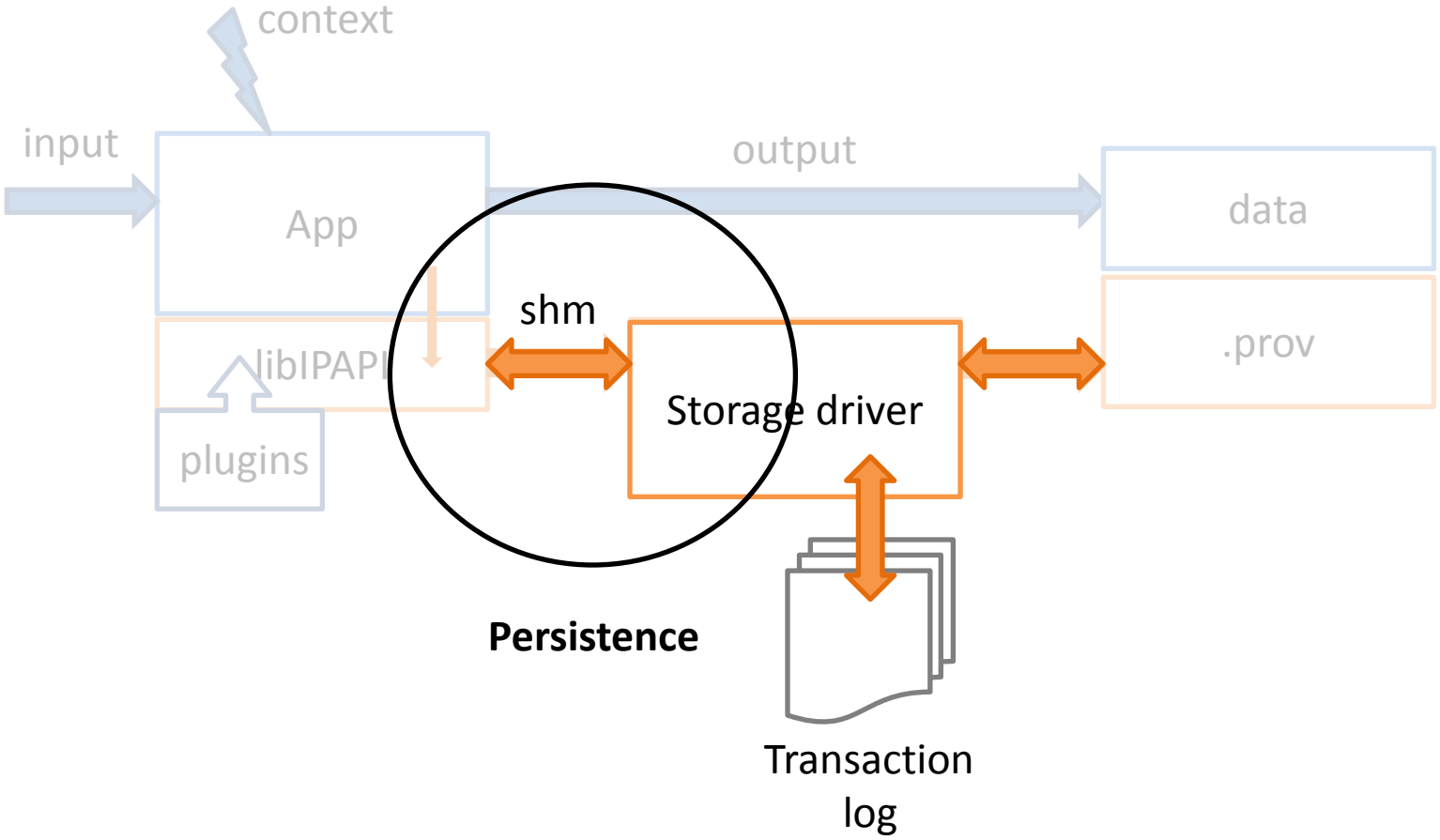
Extensibility

Repository format

+ flat object structure, identified by hashes

+ multiple hierarchical views on top (namespaces)

Some Design Decisions - Persistence



Conclusions

- IPAPI is developed in order to experiment with alternative provenance systems: ones which are **flexible**, **extensible** and work in **distributed environments**
- Clear understanding of granularity boundaries
- “You only pay for what you get” overheads
- Provenance-awareness at object level

Thank you!

lucian.carata@cl.cam.ac.uk