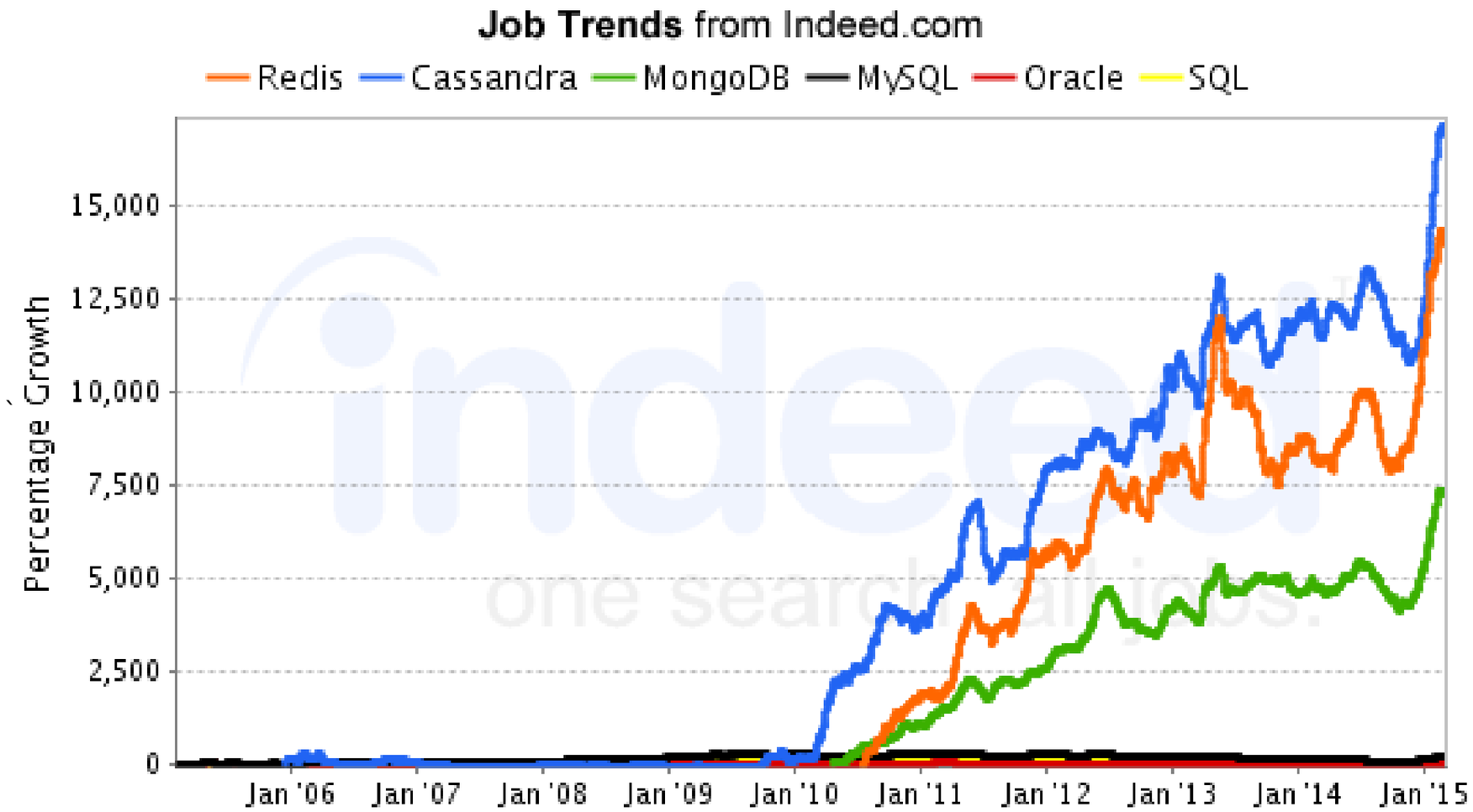
Evolving NoSQL Databases Without Downtime

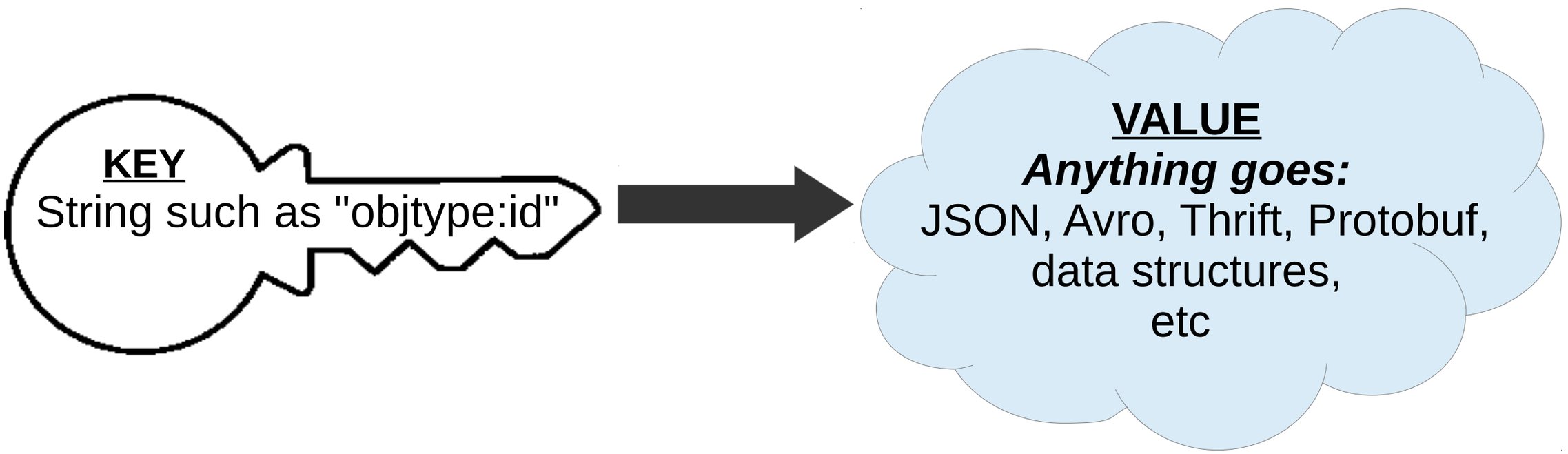
Karla Saur - University of Maryland (now at Intel Labs)
Tudor Dumitraș - University of Maryland
Michael Hicks - University of Maryland

NoSQL Database Popularity



Updating Key-Value Store Data

Key-value store: type of NoSQL database, maps key → value



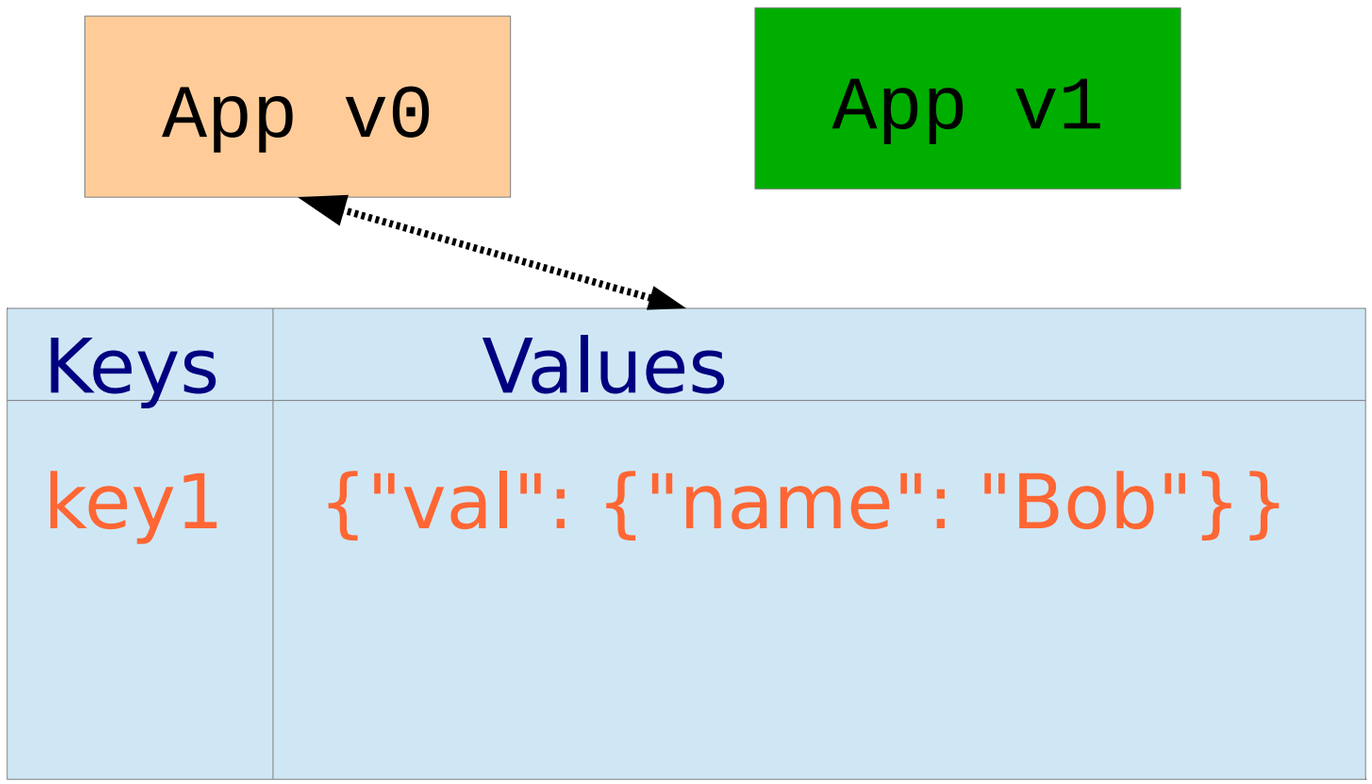
Updating Challenges:

- Schema is *implicitly* defined, no set types/structure
- Large amounts of data with multiple parties accessing the data

Updating Key-Value Store Data

Goals:

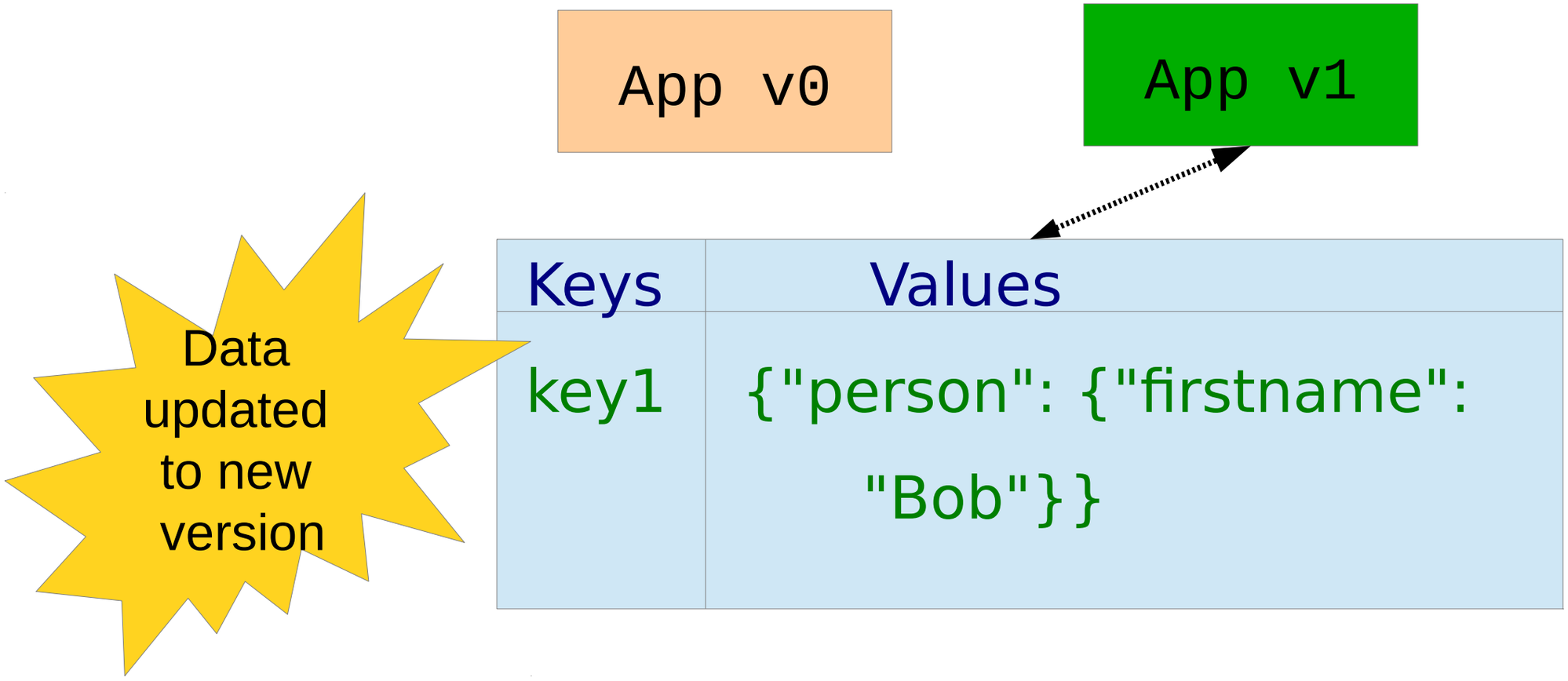
- Update the application's *data* in a key-value store
- On-line, without excessive delays or interruptions



Updating Key-Value Store Data

Goals:

- Update the application's *data* in a key-value store
- On-line, without excessive delays or interruptions



KVolve Contributions

KVolve: Key-Value store evolve

- *General* approach to updating NoSQL key/values *lazily*
- Provides template framework to assist writing update specs
- Minimal changes to programs, no juggling multi-version code
- Near no overhead normal operation, minimal impact during update

Built as an extension to key-value store Redis

Background: Keys

- Conceptually divide the kinds of objects stored in the database

- Redis advises convention:
  - Keys should have the format *n* : *k*
    n = namespace, *k* = key name

- Example:
  - Key: order:1234
    Val: `{'id' : '222', 'name' : 'foo'}`
  - Key: customer:johndoe111
    Val: `{'email' : 'foo@bar.com'}`

Background: Values

v0

```
{
 "_id": "4BD8AE97C4A580",
 "customerid": 99999,
 "name": "Foo Sushi Inc",
 "since": "12/12/2012",
 "order": {
  "orderid": "UXWE-122012",
  "orderdate": "12/12/2001",
  "orderItems": [
   {"product": "Fortune Cookies",
    "price": 19.99}
  ]
}}
```

JSON Value Example “Schema” Change

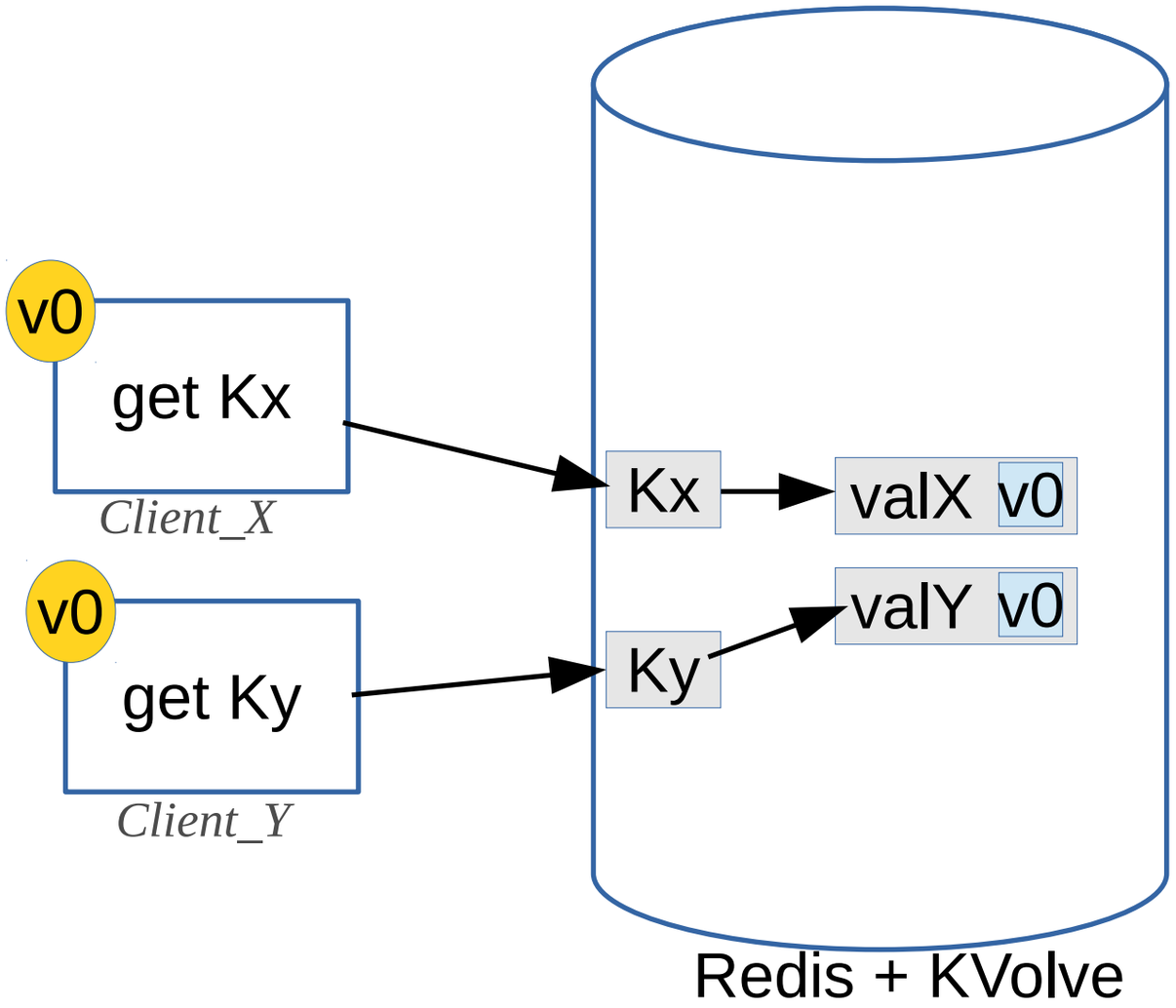
v0

```
{
 "_id": "4BD8AE97C4A580",
 "customerid": 99999,
 "name": "Foo Sushi Inc",
 "since": "12/12/2012",
 "order": {
  "orderid": "UXWE-122012",
  "orderdate": "12/12/2001",
  "orderItems": [
   {"product": "Fortune Cookies",
    "price": 19.99}
  ]
}}
```

v1

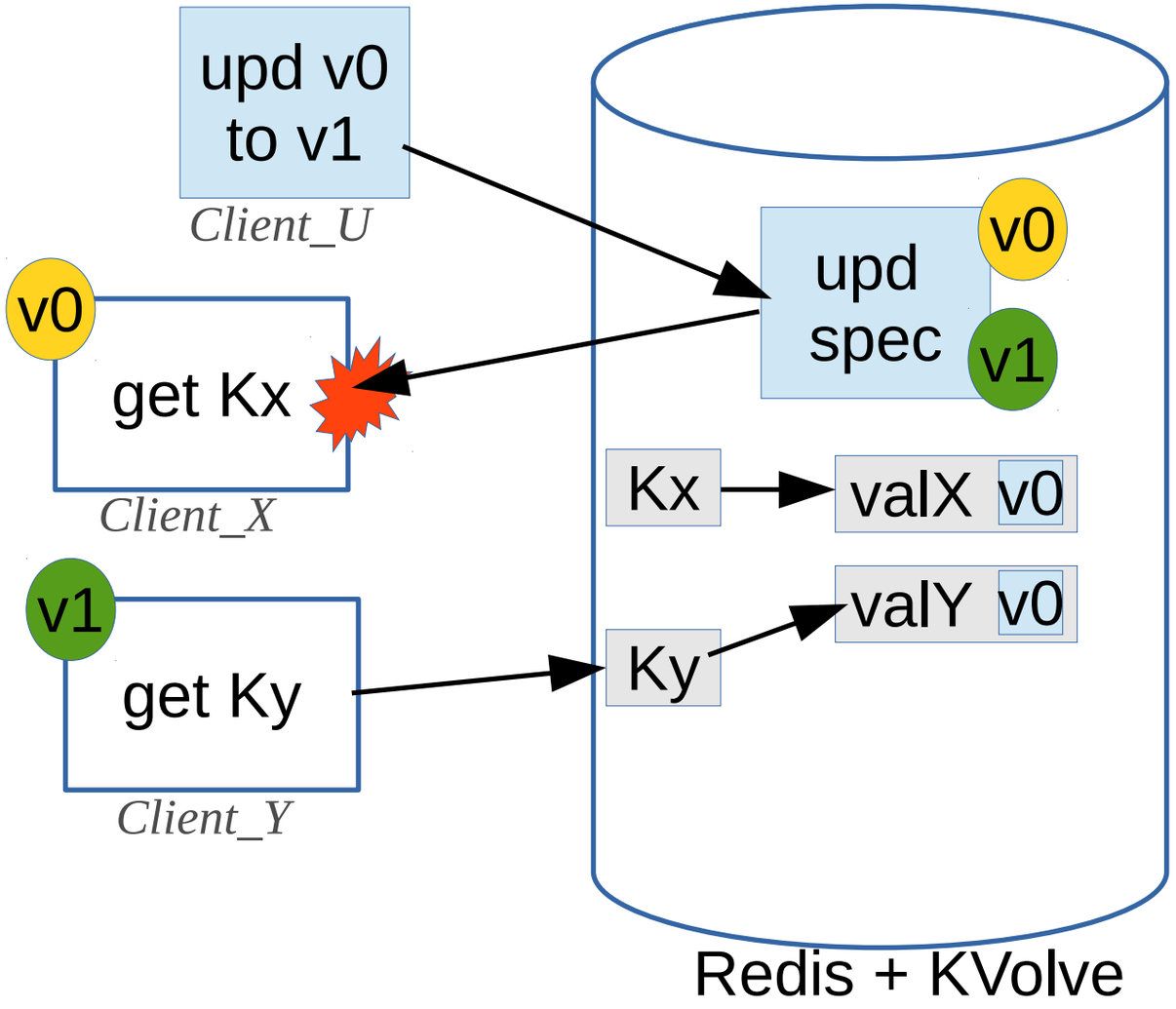
```
{
 "_id": "4BD8AE97C4A580",
 "customerid": 99999,
 "name": "Foo Sushi Inc",
 "since": "12/12/2012",
 "order": {
  "orderid": "UXWE-122012",
  "orderdate": "12/12/2001",
  "orderItems": [
   {"product": "Fortune Cookies",
    "fullPrice" : 19.99,
    "discountedPrice": 16.99}
  ]
}}
```

KVolve Architecture



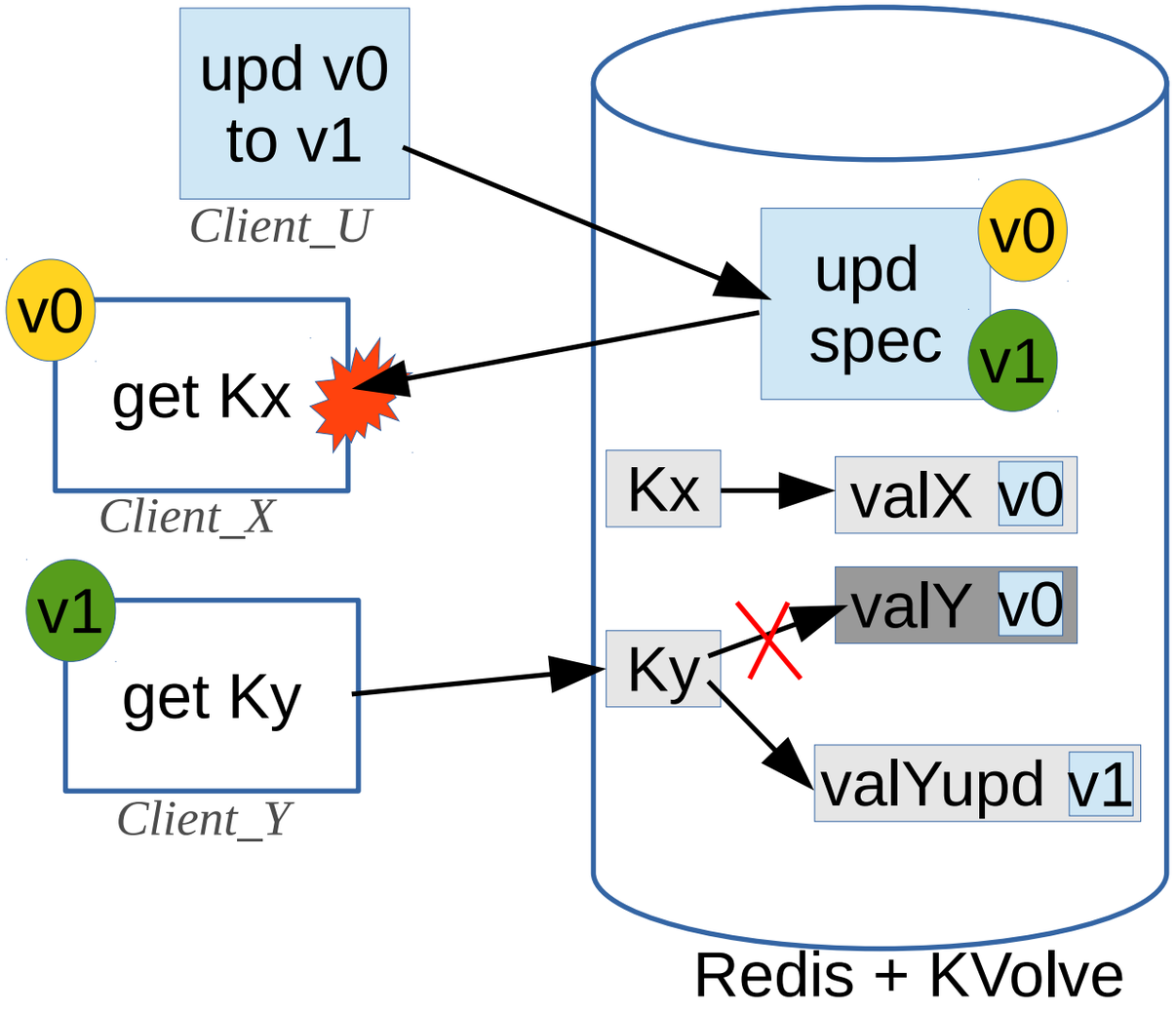
- Apps know their version number
- DB entries are individually tagged with the version number
- KVolve uses these tags to track updating

KVolve Architecture



- The update spec defines functions that take a *v0* value and produce a *v1* value
- Connected clients are notified of the update

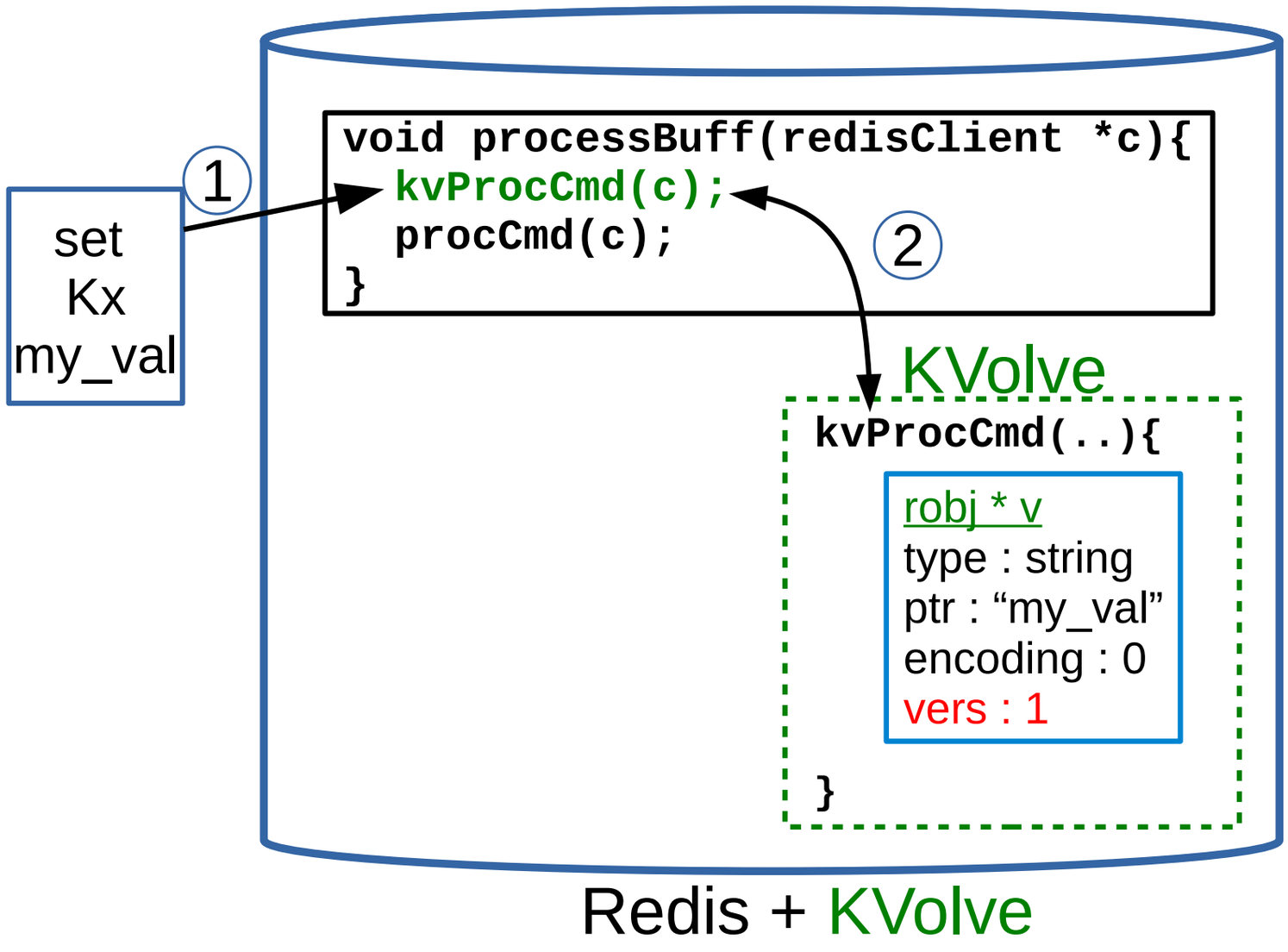
KVolve Architecture



- As clients access the database entries, they are updated, so that the data is updated on-demand (lazily)
- We support limited upgrades to keys as well

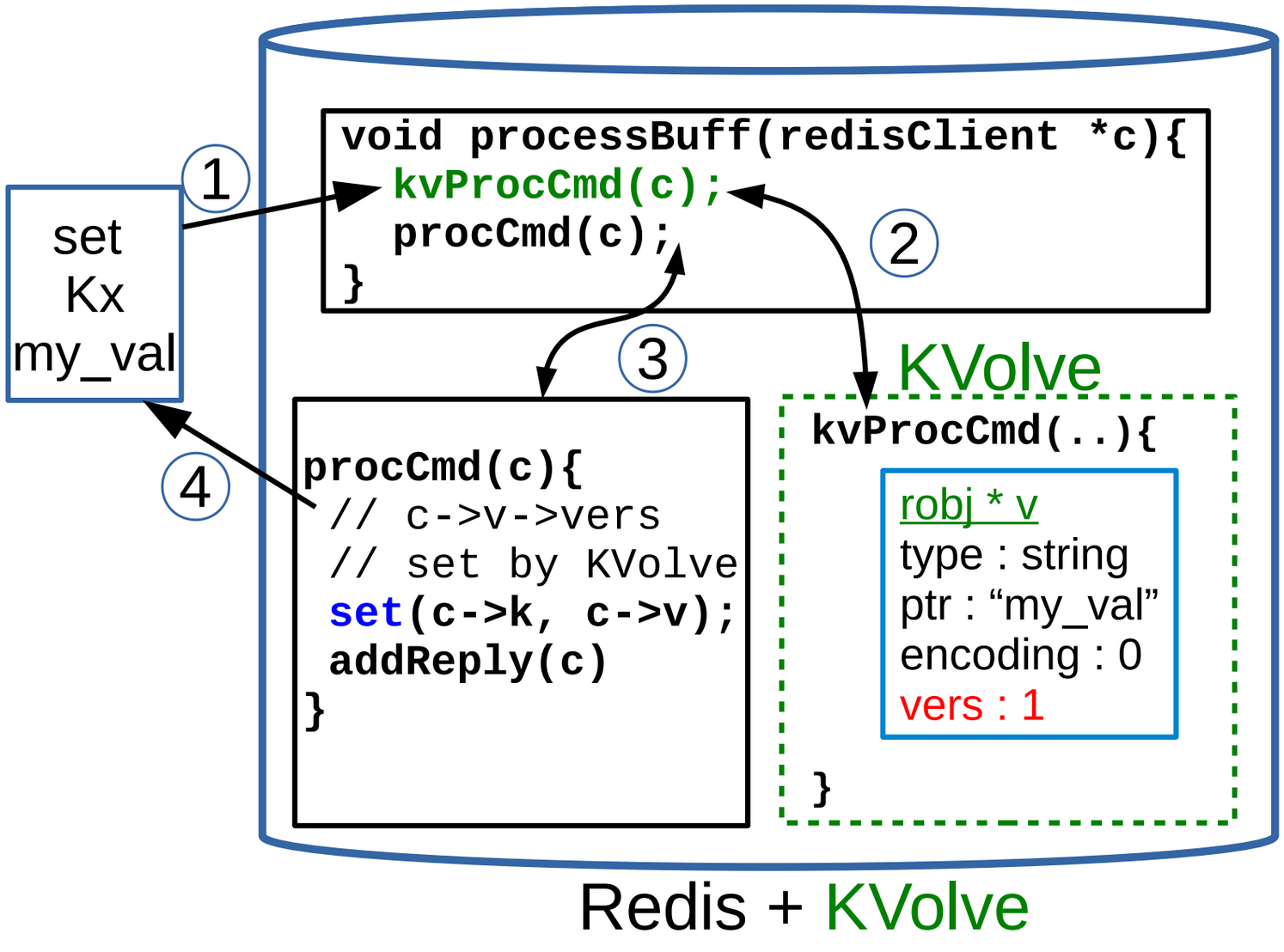
Control Flow for Redis and KVolve

- Intercepts Redis commands, prior to normal processing
- Adds a version field to Redis value structure



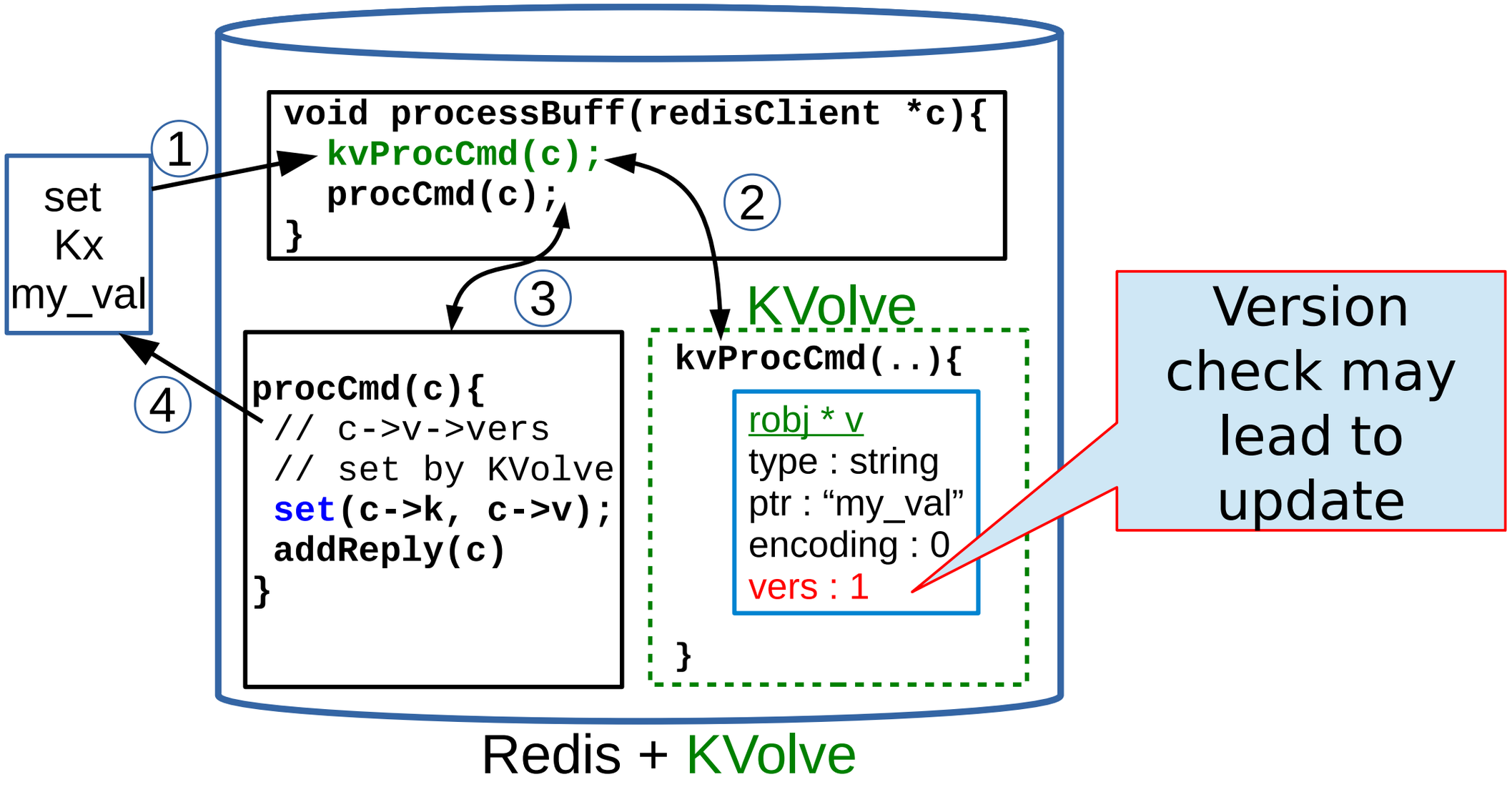
Control Flow for Redis and KVolve

- Version field automatically added to the Redis database
- Control flow proceeds as usual after version check



Control Flow for Redis and KVolve

- Version field automatically added to the Redis database
- Control flow proceeds as usual after version check



KVolve Update Specifications

Program's *code* can be any language, *update spec* in C

- Function to describe the update information:

```
void kvolve_upd_spec(char *from_ns, char *to_ns,
  int from_vers, int to_vers, int n_funs, ...);
```

- Function to transform the values:

```
typedef void (*kvolve_upd_fun)(char **key,
  void **value, size_t *val_len);
```

KVolve Update Specifications: ***Values***

Version 0

```
"orderItems": [
 {"product": "Fortune Cookies",
  "price": 19.99}
]
```

Version 1

```
"orderItems": [
 {"product": "Fortune Cookies",
  "fullPrice" : 19.99,
  "discountedPrice": 16.99}
]
```

Update Spec:

```
kvolve_upd_spec("order",
  "order", 0, 1, 1,
  fun_upd_price);
```

Update Function:

```
void fun_upd_price(...){
  json_t *ele;
  ...
  json_object_set(ele,
    "fullPrice", "price");
  ...
}
```

KVolve Update Specifications: ***Keys***

Version 1	Version 2
`amico:followers:alice`	`amico:followers:default:alice`
`amico:followers:bob`	`amico:followers:default:bob`
`amico:followers:charlie`	`amico:followers:default:charlie`
`amico:followers:eve`	`amico:followers:default:eve`

Update Spec:

```
kvolve_upd_spec(
 "amico:followers",
 "amico:followers:default",
 1, 2, 0);
```

Update Function:

(none)

KVolve: Steady-state overhead

- We benchmarked KVolve for various configurations (with and without existing updates installed) across several types of data structures (strings, lists, etc)

- During normal (non-update) situations:
  - Overhead is in the noise for single instruction (-0.77% to 2.49%)
  - Overhead is minimal for pipelined instructions (sending instructions as a batch yielded -0.52% to 5.74% overhead)

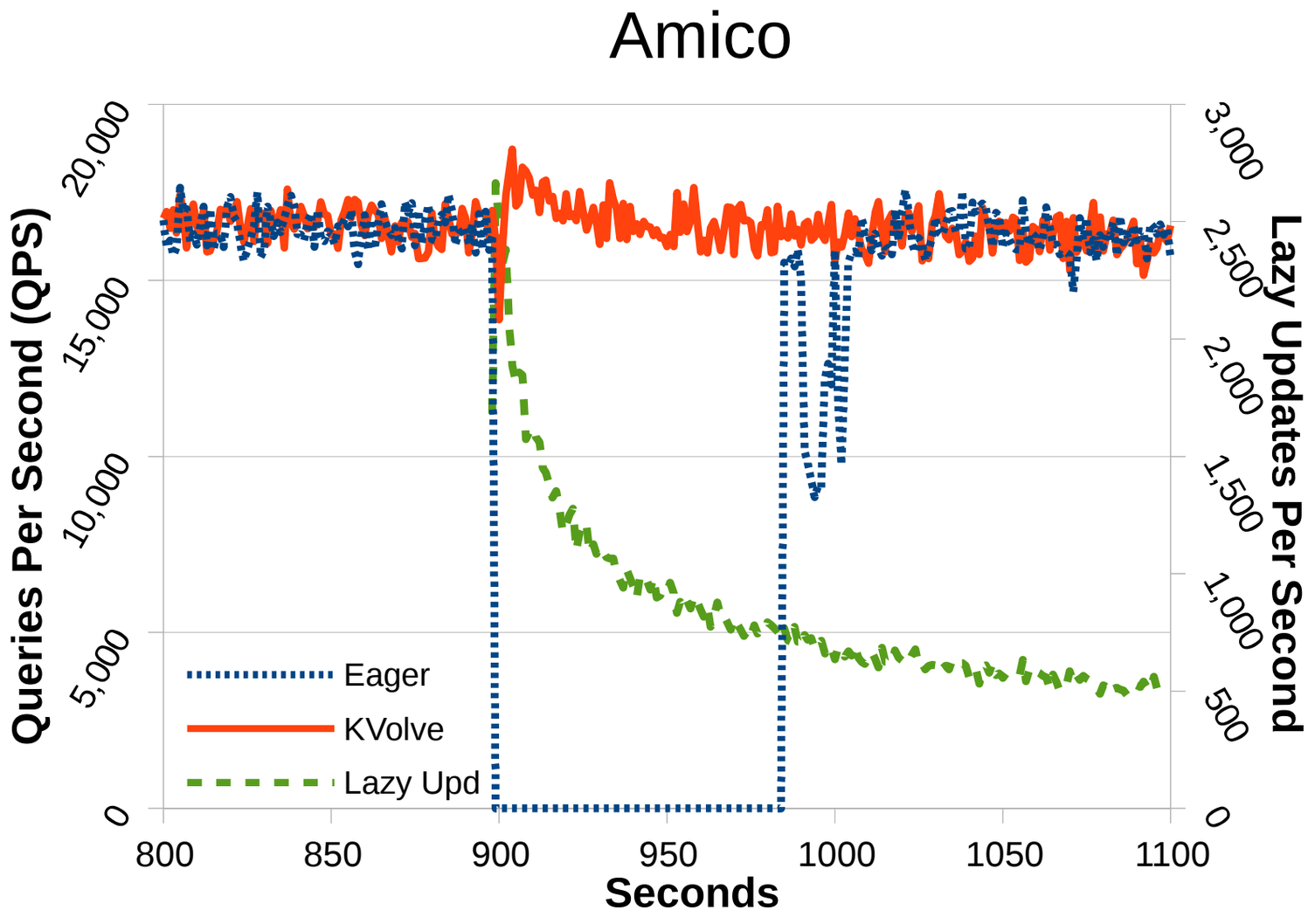
KVolve: Application Benchmarks

From GitHub: examples of schema changes with Redis

- Amico models relationships for social networks: “A follows B”
  - 200 lines of Ruby code, 10 versions 2012-2013
  - Added a prefix to all keys
- Redisfs uses Redis as the backend to the FUSE file system
  - 2.2K lines of C code, 8 versions 2011
  - Added prefix to some keys, added compression to all data

KVolve: Application Benchmarks

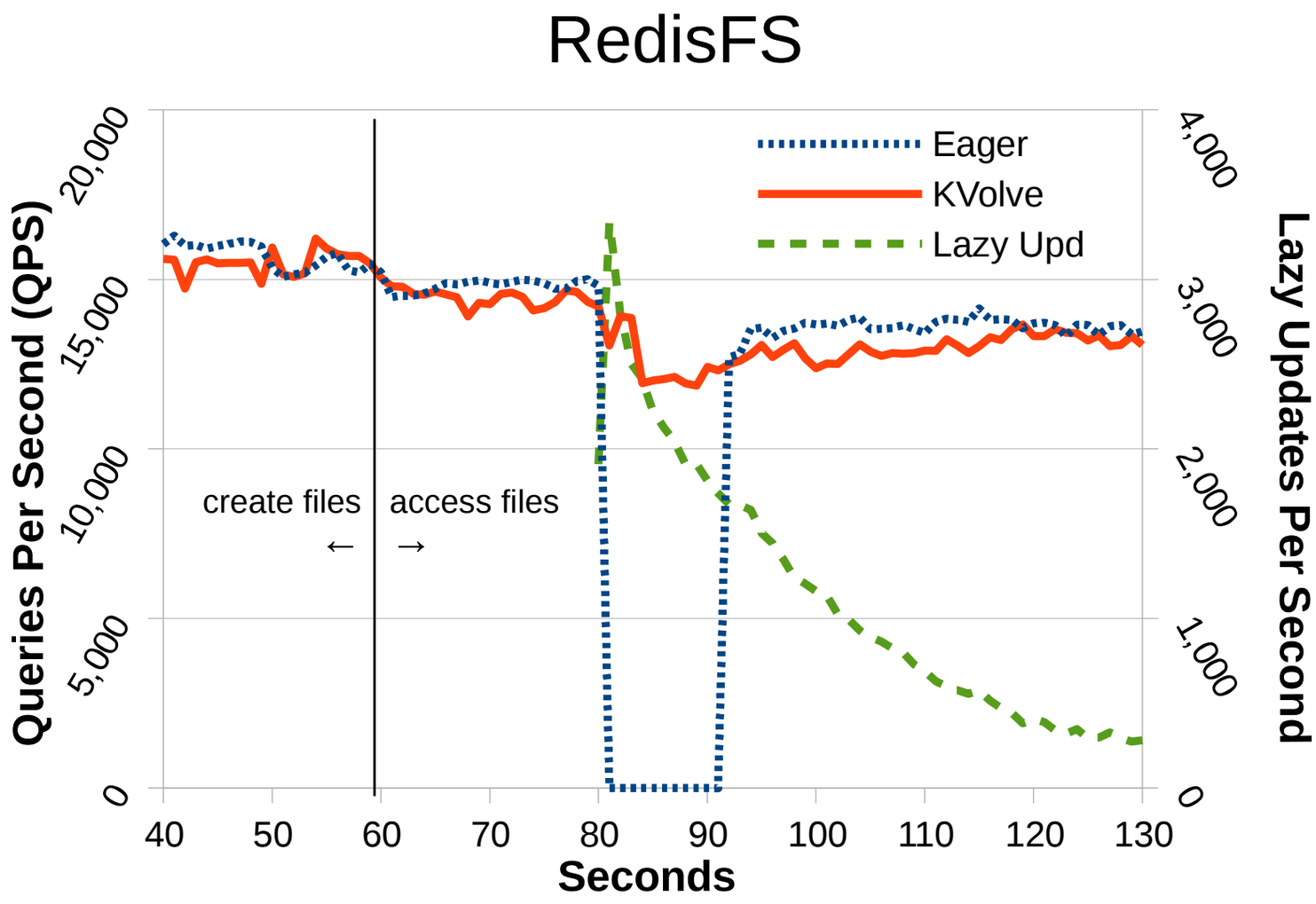
Green line (right y-axis) shows lazy transformation



- Renamed ~792K keys from LiveJournal data set
- Offline migration took ~87s

KVolve: Application Benchmarks

Green line (right y-axis) shows lazy transformation



- Updated ~123K total keys for files generated by PostMark
- Combined w/Kitsune *[Hayden et al. TOPLAS 2014]* for no downtime

Future Work

- Distributed data updates:

  Using locking/consensus mechanisms to expand beyond centralized Redis instance

- Automatic generation of transformation functions in NoSQL updates:

  A DSL or other tool to help update-writers

Thanks!

Code:

https://github.com/plum-umd/kvolve