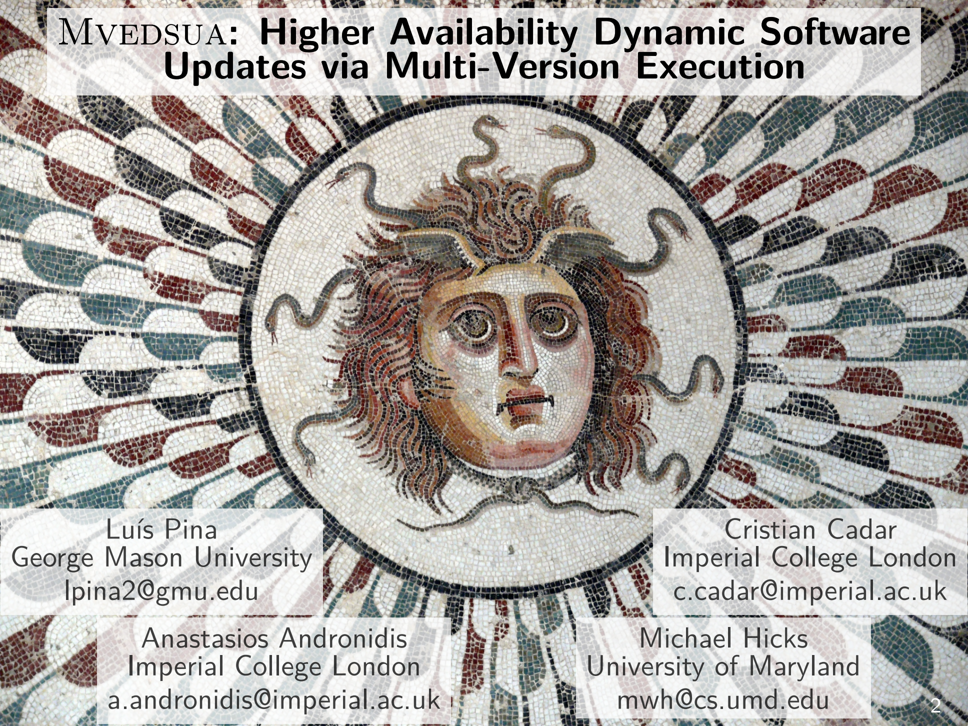
MVEDSUA: Higher Availability Dynamic Software Updates via Multi-Version Execution



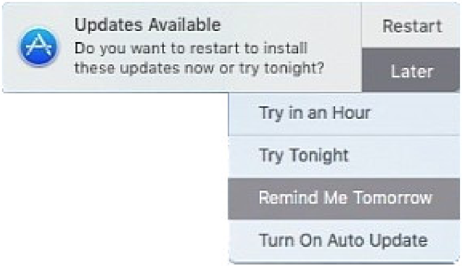
Luís Pina
George Mason University
lpina2@gmu.edu

Cristian Cadar
Imperial College London
c.cadar@imperial.ac.uk

Anastasios Andronidis
Imperial College London
a.andronidis@imperial.ac.uk

Michael Hicks
University of Maryland
mwh@cs.umd.edu

An Update is Available!



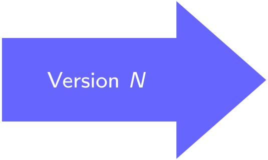
An Update is Available!

Good idea to install

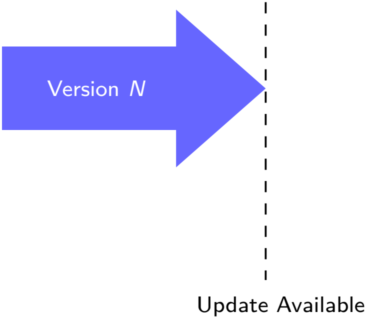
An Update is Available!

Good idea to install, **especially** if you keep sensitive data

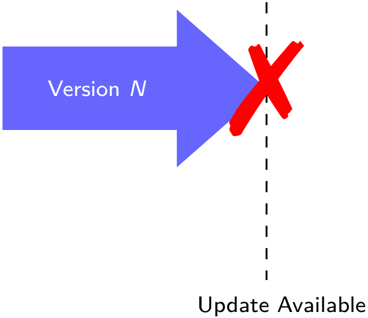
Software Updates are Disruptive



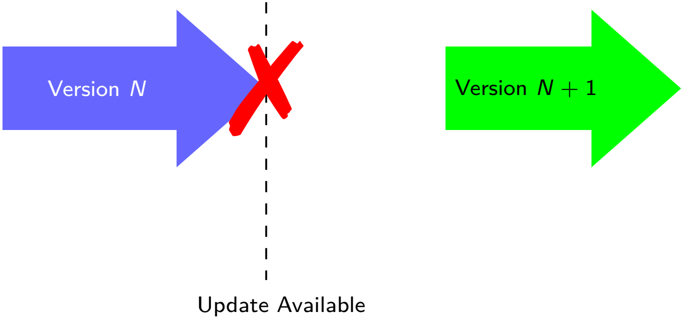
Software Updates are Disruptive



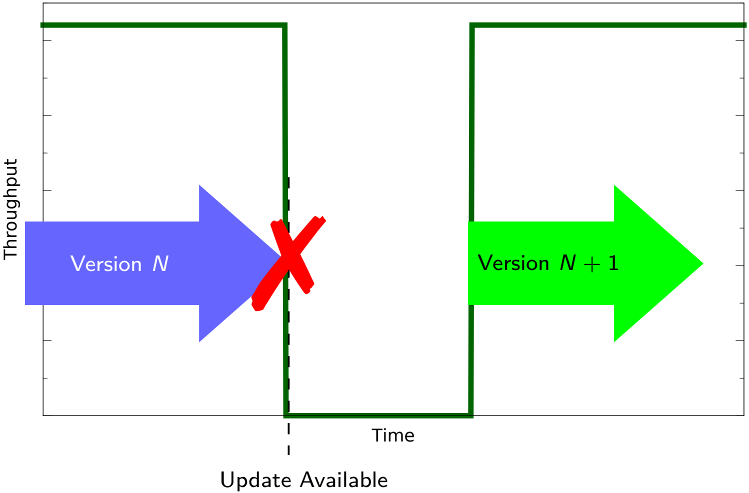
Software Updates are Disruptive



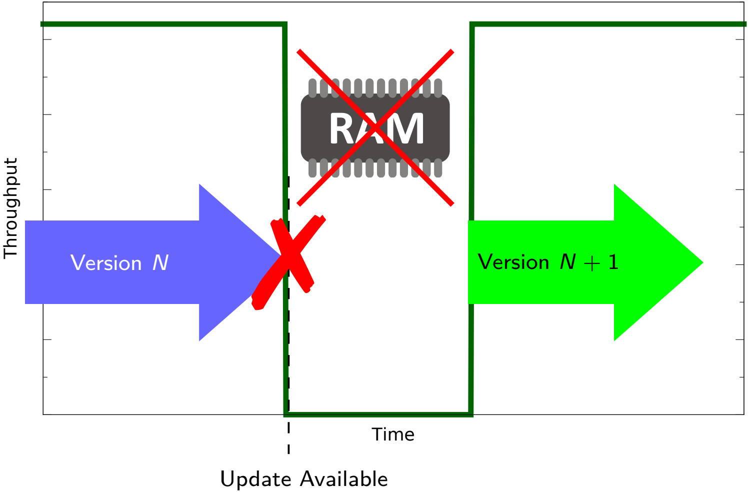
Software Updates are Disruptive



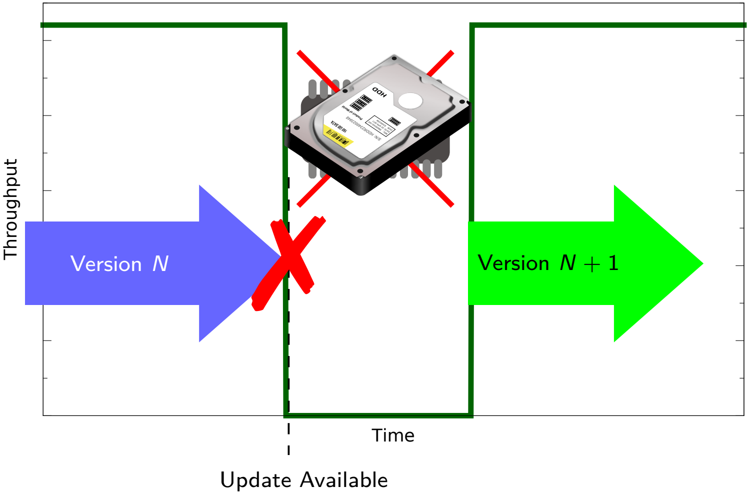
Software Updates are Disruptive



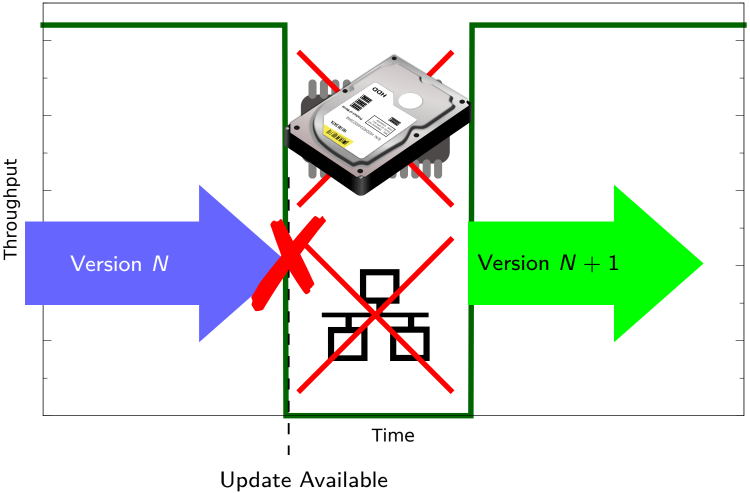
Software Updates are Disruptive



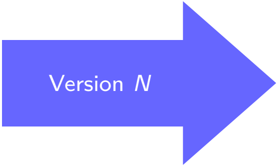
Software Updates are Disruptive



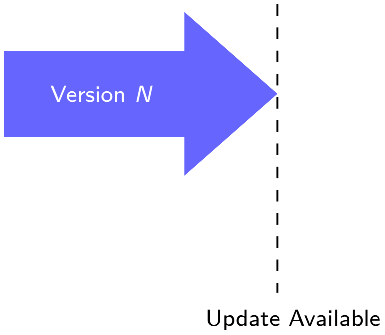
Software Updates are Disruptive



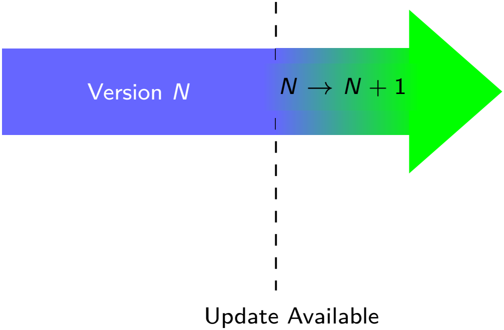
Dynamic Software Updating (DSU)



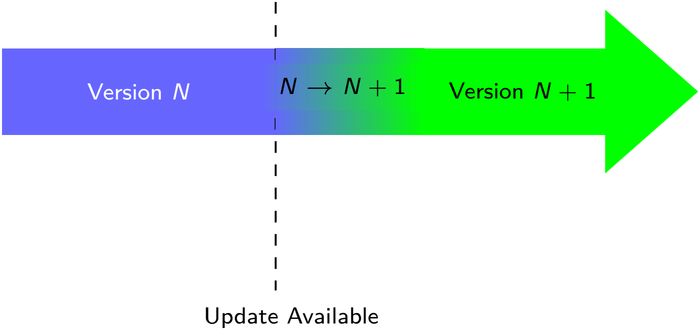
Dynamic Software Updating (DSU)



Dynamic Software Updating (DSU)



Dynamic Software Updating (DSU)



Update Errors

Errors:

- New version
- State transformation
- Code to start the new version
- Code to stop the old version

Update Errors

Errors:

- New version
- State transformation
- Code to start the new version
- Code to stop the old version

Results:

- Crash/hang
- Corrupted state
- Loss of state

Mvedsua: Higher Availability Dynamic Software Updates via Multi-Version Execution

Mvedsua: Higher Availability Dynamic Software Updates via Multi-Version Execution

- Dynamic Software Updates
  - No loss of state during updates
- Multi-Version Execution
  - Execute old and new versions in parallel
  - Tolerate/detect update errors
- Match old and new states
  - Through developer-specified rules
- Good performance
  - Low steady-state overhead
  - Mask update pause

Example DSU

Changes

In-memory key-val store with simple format:

put(key,val)

An update adds types:

put(type ,key,val)

With types `string` or `int`

Example DSU

Updates as any other program feature

- How to transform the state?

(key,val)=("string",key,val)

- When to update?
  - E.g., when not processing any request
- How to restart the program from where it stopped?

Example DSU

Updates as any other program feature

- How to transform the state?

(key,val)=("string",key,val)

- When to update?
  - E.g., when not processing any request
- How to restart the program from where it stopped?

Kitsune: Efficient, General-purpose Dynamic Software Updating for C

Christopher M. Hayden   Edward K. Smith   Michail Denchev
Michael Hicks   Jeffrey S. Foster
University of Maryland, College Park, USA
{hayden,tedks,mdenchev,mwh,jfoster}@cs.umd.edu

KITSUNE

Rolling Upgrades

Version N Version N Version N

Rolling Upgrades

Rolling Upgrades

Version $N+1$

Version N

Version N

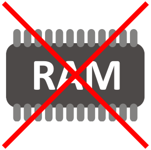
Rolling Upgrades

Version $N+1$ | Version $N+1$ | Version $N+1$

Rolling Upgrades

Loss of State

Restarting in Version $N + 1$



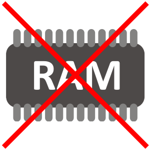
Version N

Version N

Rolling Upgrades

Loss of State

Restarting in Version $N+1$



Version N

Version N

Scaling Memcache at Facebook

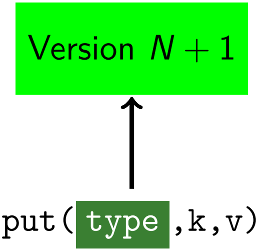
Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, Venkateshwaran Venkataramani

{rajeshn,hans}@fb.com, {sgrimm, marc}@facebook.com, {herman, hcli, rm, mpal, dpeek, ps, dstaff, ttung, veeve}@fb.com

Facebook Inc.

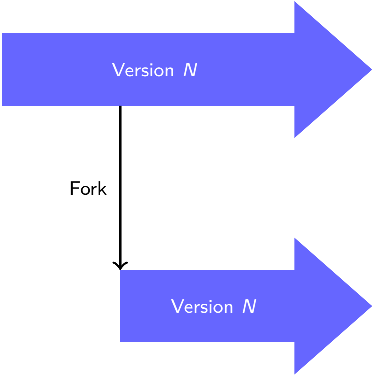
Rolling Upgrades

Incompatible Protocol

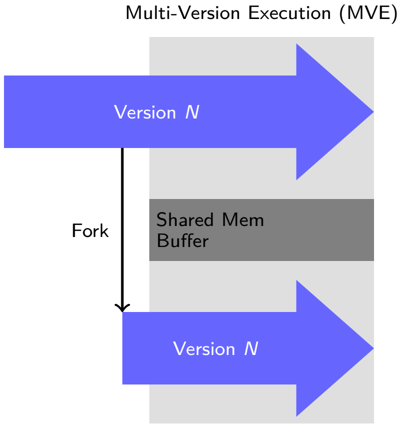


Reliable Updates with MVEDSUA

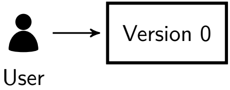
Reliable Updates with MVEDSUA



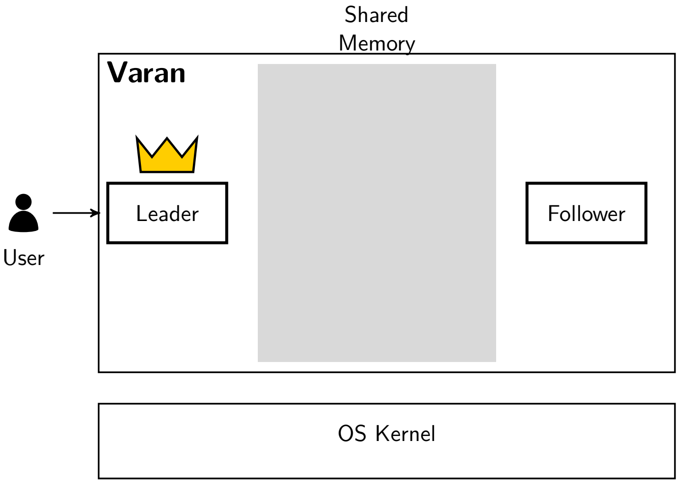
Reliable Updates with MVEDSUA



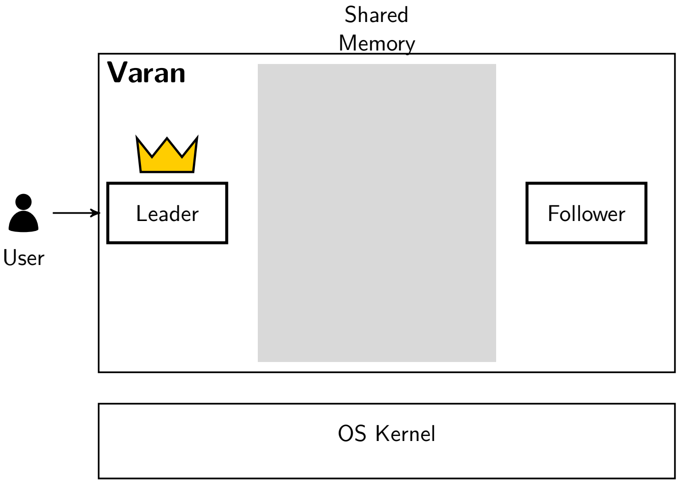
Multi-Version Execution



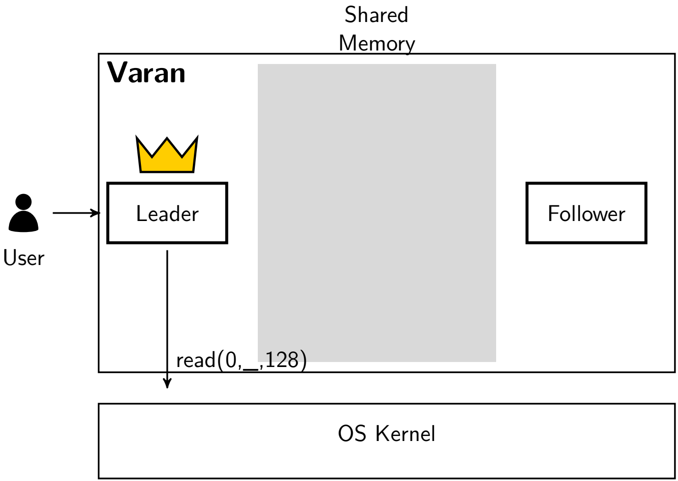
Multi-Version Execution



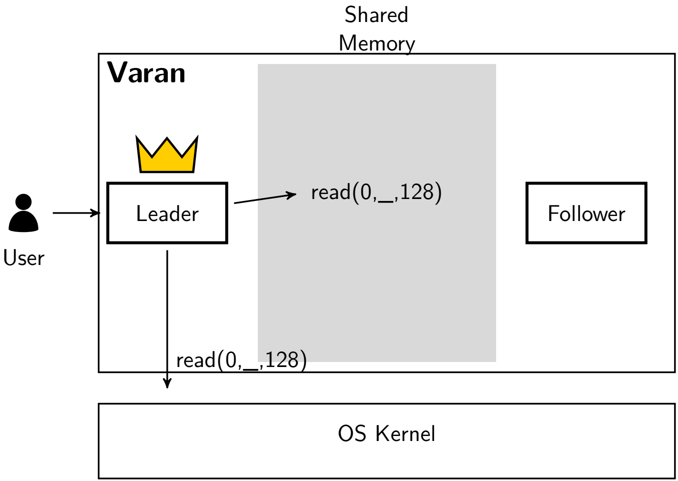
Multi-Version Execution



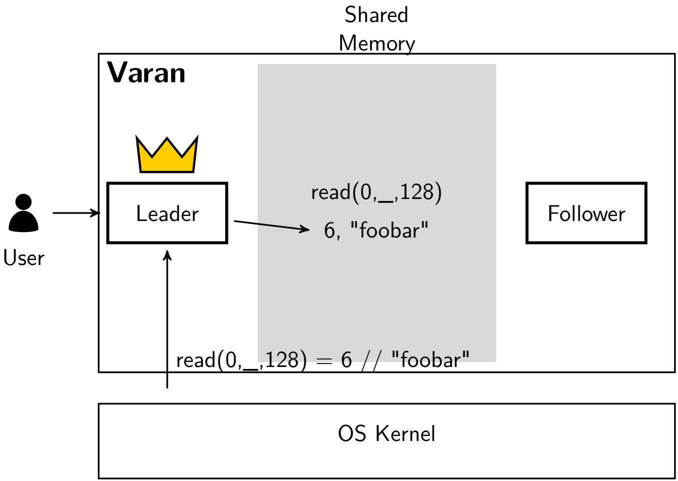
Multi-Version Execution



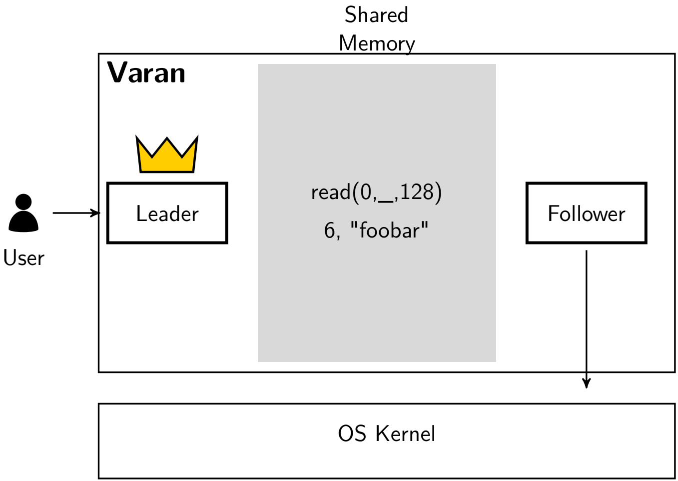
Multi-Version Execution



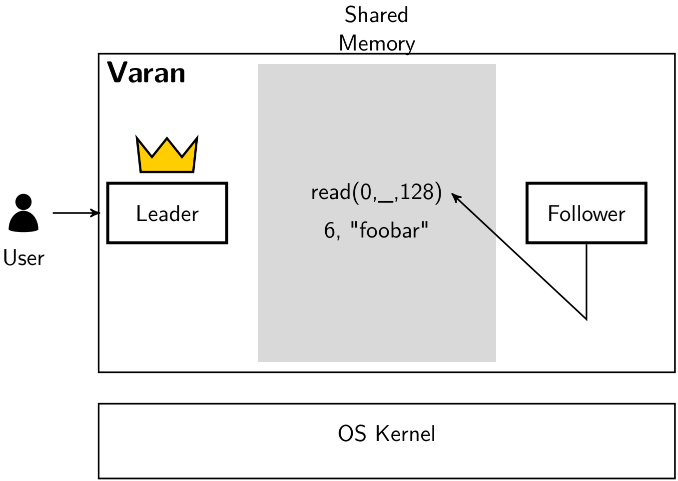
Multi-Version Execution



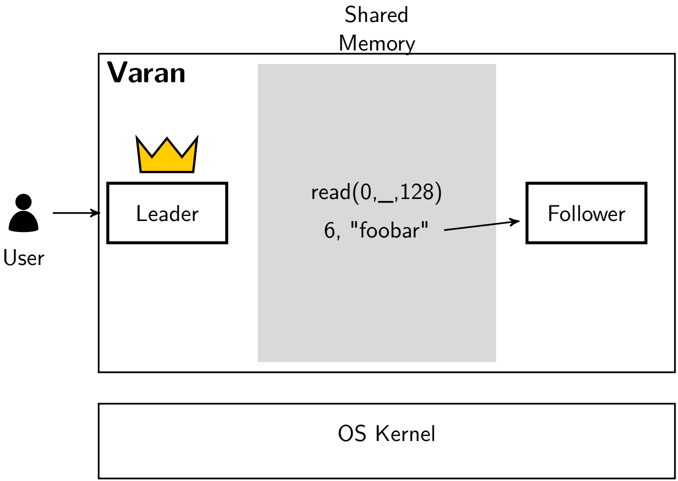
Multi-Version Execution



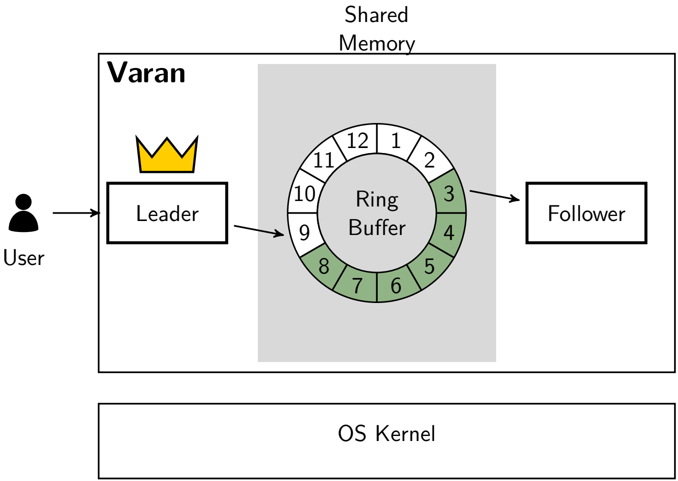
Multi-Version Execution



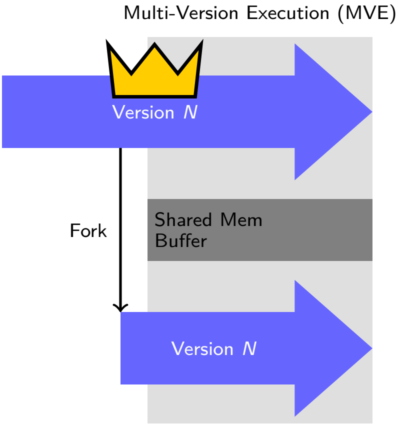
Multi-Version Execution



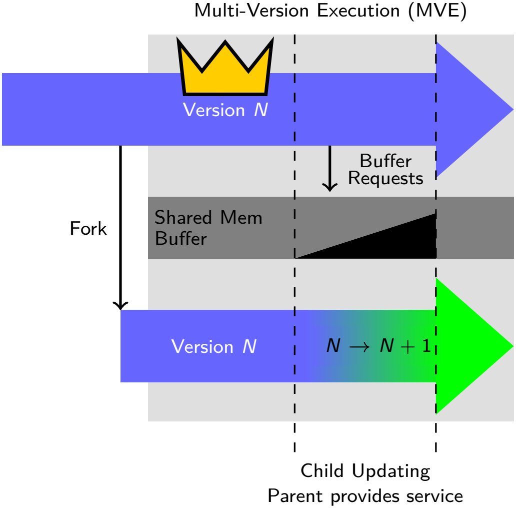
Multi-Version Execution



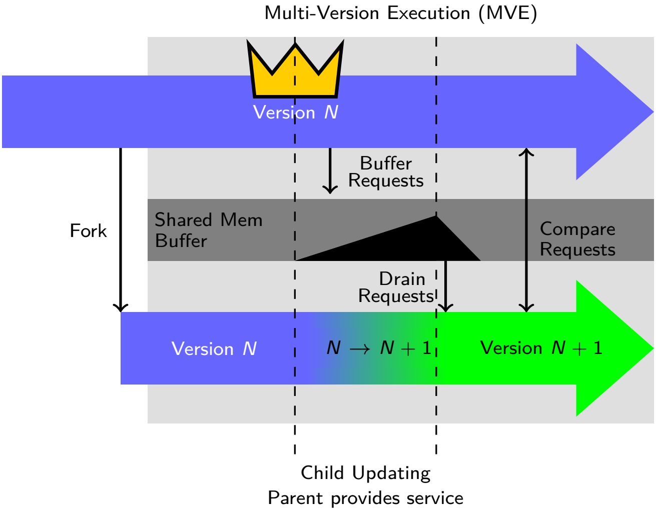
Reliable Updates with Mvedsua



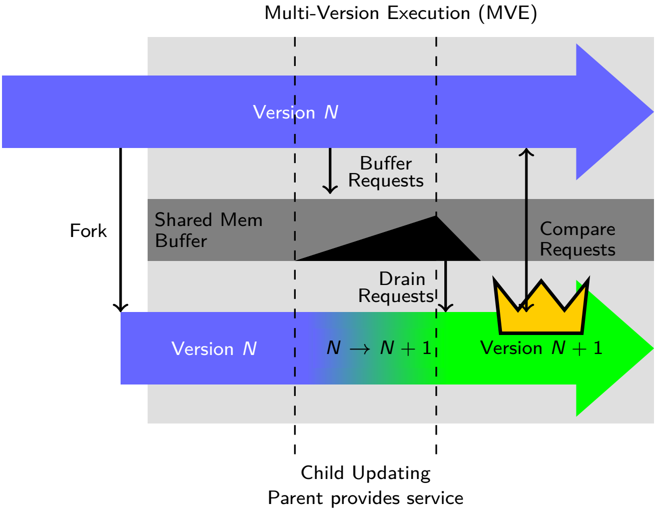
Reliable Updates with MVEDSUA



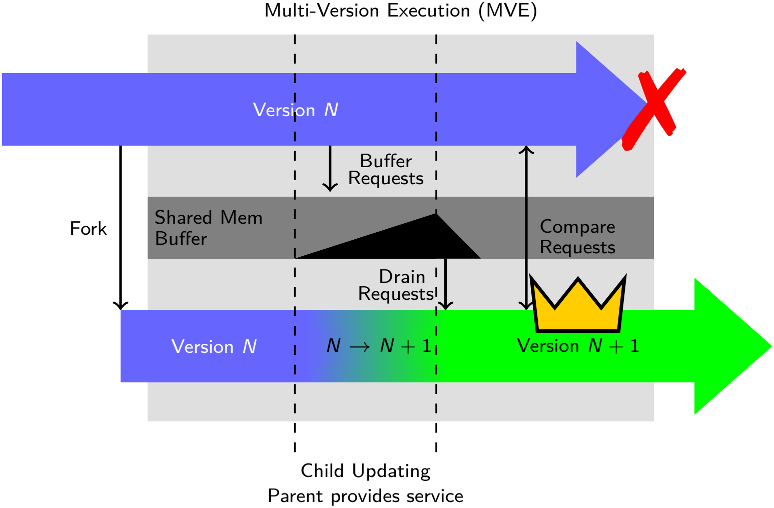
Reliable Updates with MVEDSUA



Reliable Updates with MVEDSUA



Reliable Updates with MVEDSUA



What about
non-backwards-compatible
features?

(Or backwards-compatible features implemented differently?)

Mapping semantics

Example update

In-memory key-val store with simple wire protocol:

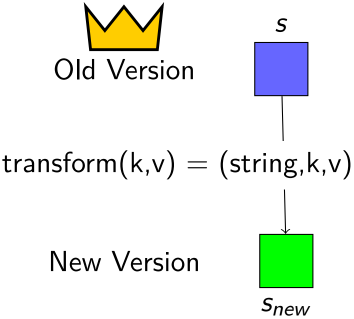
`read(_,`**"put(key,val)"**`,_)`

An update adds types:

`read(_,`**"put($\boxed{\text{type}}$,key,val)"**`,_)`

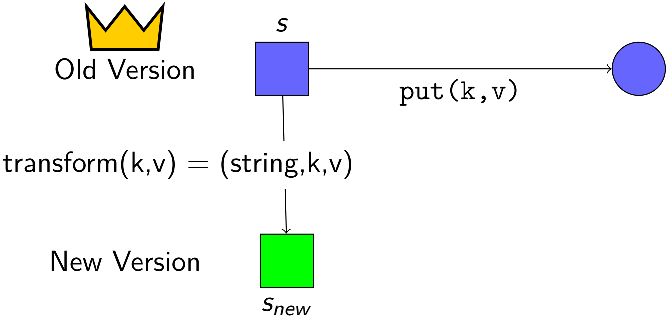
With types `string` or `int`

Mapping semantics
MVE



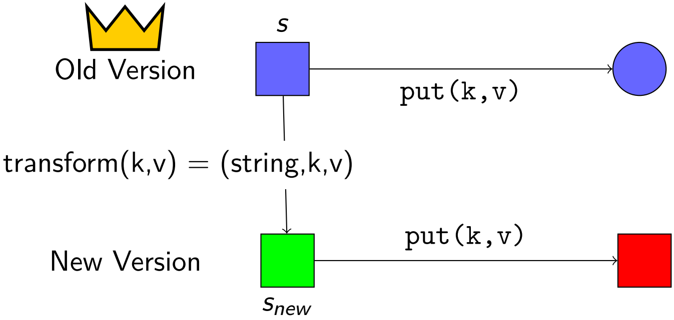
Mapping semantics

MVE



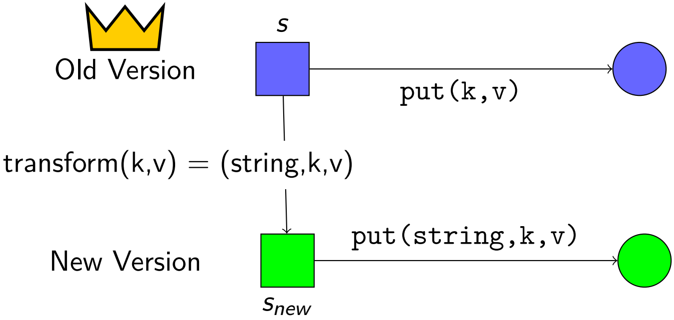
Mapping semantics

MVE

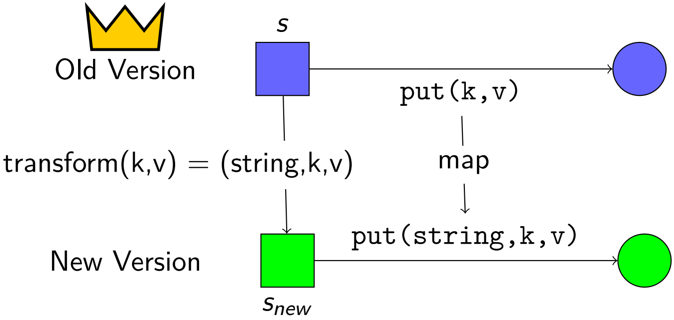


Mapping semantics

MVE



Mapping semantics
MVE



Mapping semantics

Command mappings

1. $\texttt{map}_{\texttt{old} \to \texttt{new}}$`[put(k,v)] = put(string,k,v)`
2. $\texttt{map}_{\texttt{old} \to \texttt{new}}$`[put(string,k,v)] = not supported`
3. $\texttt{map}_{\texttt{new} \to \texttt{old}}$`[put(string,k,v)] = put(k,v)`
4. $\texttt{map}_{\texttt{new} \to \texttt{old}}$`[put(k,v)] = not supported`

Implementing MVEDSUA

Kitsune: Efficient, General-purpose Dynamic Software Updating for C

Christopher M. Hayden Edward K. Smith Michail Denchev
Michael Hicks Jeffrey S. Foster
University of Maryland, College Park, USA
{hayden,tedks,mdenchev,mwh,jfoster}@cs.umd.edu

KITSUNE

VARAN the Unbelievable

An Efficient N-version Execution Framework

Petr Hosek Cristian Cadar
Department of Computing
Imperial College London
{p.hosek, c.cadar}@imperial.ac.uk

Evaluating MVEDSUA

1. Number of rules needed
2. Steady-state overhead
3. Update overhead
   - Performance in MVE
   - Update pause
4. Errors detected/tolerated

Mapping sematics

Number of Rules

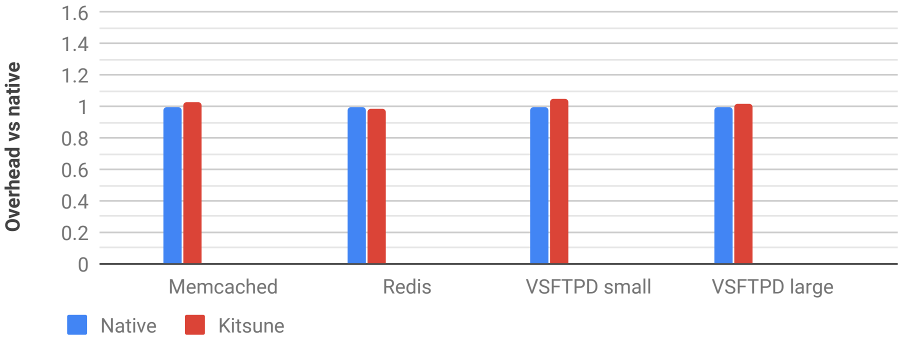
VSFTPD

Versions	# rules	Versions	# rules
1.1.0 → 1.1.1	—	2.0.0 → 2.0.1	—
1.1.1 → 1.1.2	2	2.0.1 → 2.0.2	1
1.1.2 → 1.1.3	—	2.0.2 → 2.0.3	1
1.1.3 → 1.2.0	2	2.0.3 → 2.0.4	1
1.2.0 → 1.2.1	—	2.0.4 → 2.0.5	1
1.2.1 → 1.2.2	—	2.0.5 → 2.0.6	—
1.2.2 → 2.0.0	3	Average	0.85

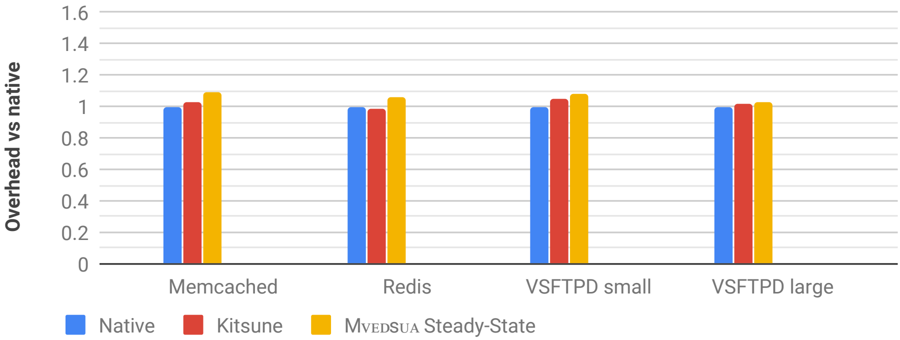
Redis: 2.0.0 → 2.0.1 (1 rule), 2.0.1 → 2.0.2, 2.0.1 → 2.0.3

Memcached: 1.2.2 → 1.2.3, 1.2.3 → 1.2.4

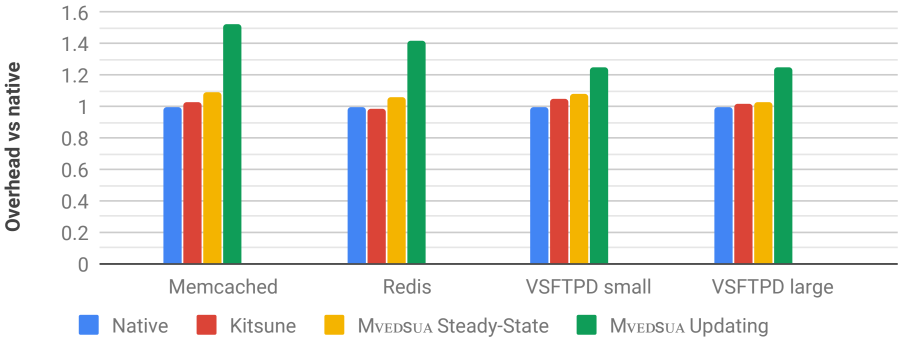
Steady State Overhead



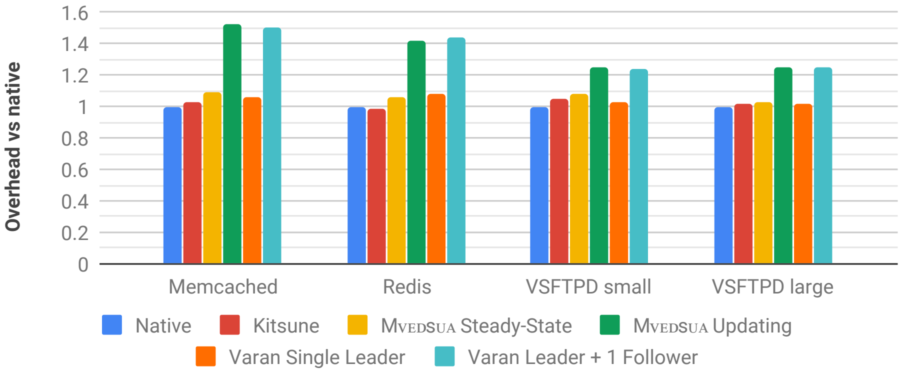
Steady State Overhead



Steady State Overhead

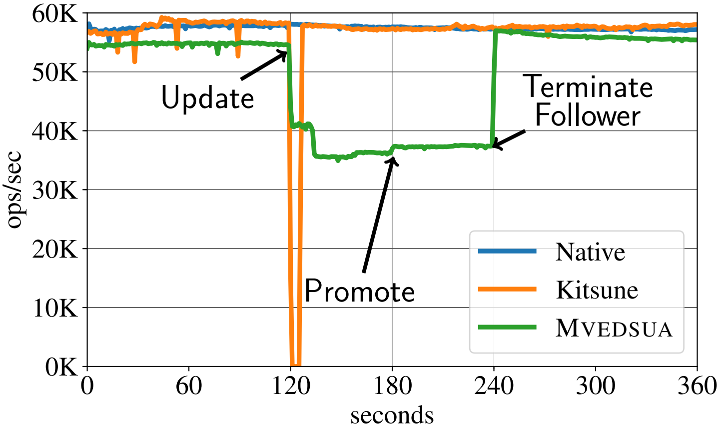


Steady State Overhead



Update Overhead (worst-case)

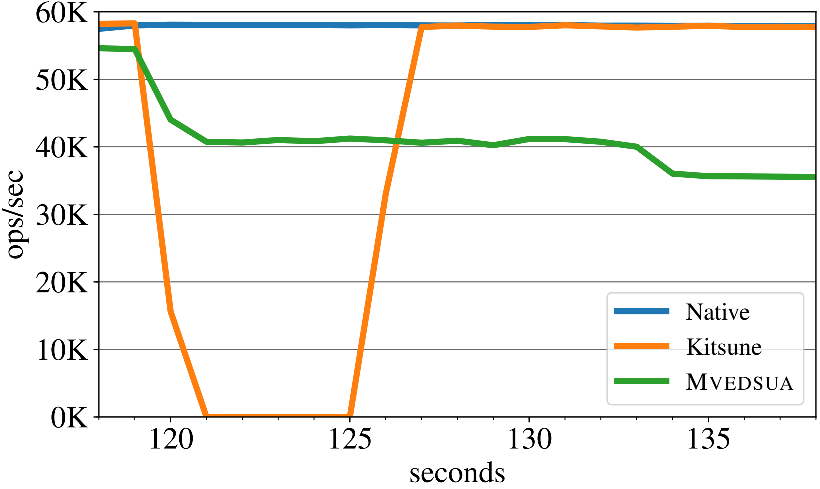
Redis with large state



- 1M entries, 250MB resident process space
- Kitsune takes 5s to transform state
- Promote + terminate could happen early at $120s + (5s \times 2)$

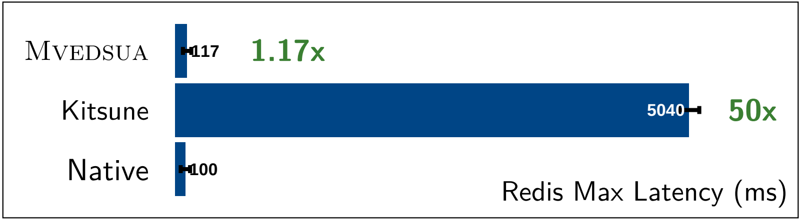
Update Overhead

Redis with large state — Zoom around update



Update Overhead

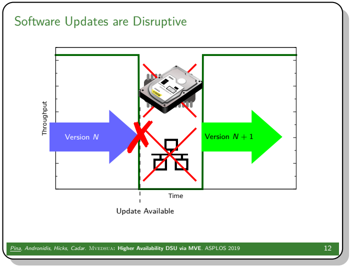
Redis with large state — Max latency



MVEDSUA tolerates errors

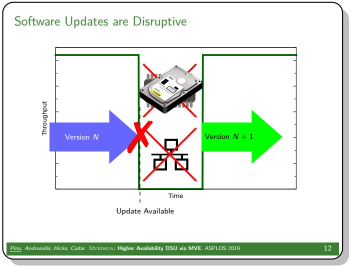
- New code Redis revision `7fb16bac` crashes on `HMGET`
  - New version introduces error on existing `HMGET` command
- State transform Use-after-free for Memcached update
  - Transformation logic frees memory used by libevent
  - No problem for small number of threads/key-value pairs
  - Crashes in production
- Timing Memcached and libevent after update
  - libevent keeps round-robin list of active FDs
  - Update reorders that list (resets all FDs)
  - Divergence when order does not match pre-update

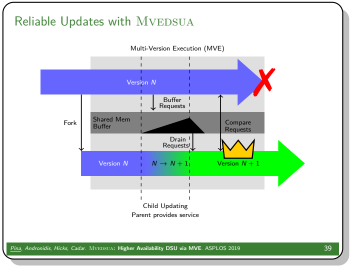
Conclusion



Software Updates are Disruptive
Throughput
Version N
Version N + 1
Time
Update Available
Pina, Andronidis, Hicks, Cadar. MVEDSUA: Higher Availability DSU via MVE. ASPLOS 2019
12

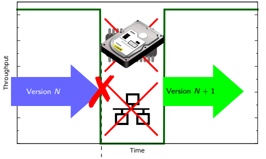
Conclusion


Software Updates are Disruptive
Throughput
Version N
Version $N+1$
Time
Update Available
Pina, Andronidis, Hicks, Cadar. MVEDSUA: Higher Availability DSU via MVE. ASPLOS 2019
12

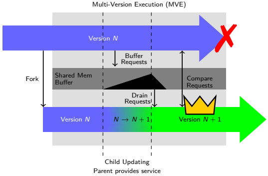

Reliable Updates with MVEDSUA
Multi-Version Execution (MVE)
Version N
Buffer Requests
Fork
Shared Mem Buffer
Compare Requests
Drain Requests
Version N
$N \rightarrow N+1$
Version $N+1$
Child Updating
Parent provides service
Pina, Andronidis, Hicks, Cadar. MVEDSUA: Higher Availability DSU via MVE. ASPLOS 2019
39

Conclusion

Software Updates are Disruptive



Reliable Updates with MVEDSUA



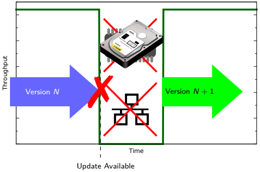
Mapping semantics

Command mappings

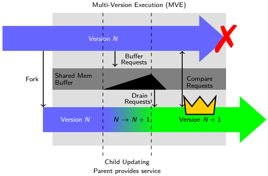
1. `map`$_{\texttt{old}\to\texttt{new}}$`[put(k,v)] = put(string,k,v)`
2. `map`$_{\texttt{old}\to\texttt{new}}$`[put(string,k,v)] = not supported`
3. `map`$_{\texttt{new}\to\texttt{old}}$`[put(string,k,v)] = put(k,v)`
4. `map`$_{\texttt{new}\to\texttt{old}}$`[put(k,v)] = not supported`

Conclusion

Software Updates are Disruptive



Reliable Updates with MVEDSUA

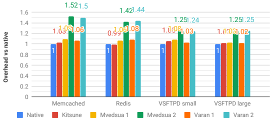


Mapping semantics

Command mappings

1. map$_{\text{old} \rightarrow \text{new}}$[put(k,v)] = put(string,k,v)
2. map$_{\text{old} \rightarrow \text{new}}$[put(string,k,v)] = not supported
3. map$_{\text{new} \rightarrow \text{old}}$[put(string,k,v)] = put(k,v)
4. map$_{\text{new} \rightarrow \text{old}}$[put(k,v)] = not supported

Steady State Overhead



Looking for students!

Conclusion

- Software updates are disruptive at best, catastrophic at worst
  - DSU helps, but updates can still fail
- MVEDSUA performs DSU reliabily using MVE
  - Great single-version performance
  - Eliminates update pauses
  - Tolerates update errors
- Non-backwards-compatible features require new-to-old and old-to-new command mappings
  - Not always possible to write
- I'm looking for students!

Extra Slides

Update by dumping to disk and restarting

- 10GB Redis takes 28s vs 22s with Kitsune (21ms opt)
- 1M entries (2GB) takes 5s vs 3.5s with Kitsune (22ms opt)

Kitsune: Efficient, General-purpose Dynamic Software Updating for C

Christopher M. Hayden, University of Maryland, College Park
Karla Saur, University of Maryland, College Park
Edward K. Smith, University of Maryland, College Park
Michael Hicks, University of Maryland, College Park
Jeffrey S. Foster, University of Maryland, College Park

- 1GB H2 takes 13s vs 3s with Rubah

Rubah: Efficient, General-purpose Dynamic Software Updating for Java

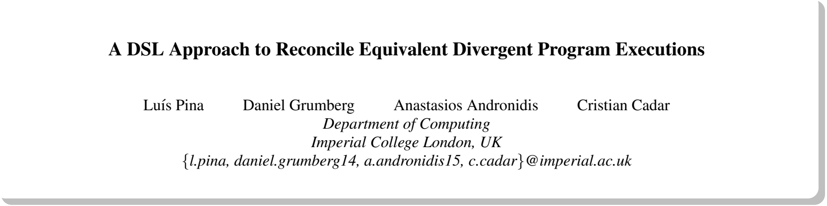
Luís Pina*
INESC-ID / Instituto Superior Técnico
Lisbon, Portugal

Michael Hicks
University of Maryland
College Park, USA

Mapping semantics

Sample rule

```
read(fd,s,n) {{ return strstr("put",s) == 0; }}
  =>
    read(fd,ss,n)
```



A DSL Approach to Reconcile Equivalent Divergent Program Executions

Luís Pina Daniel Grumberg Anastasios Andronidis Cristian Cadar

Department of Computing

Imperial College London, UK

{*l.pina, daniel.grumberg14, a.andronidis15, c.cadar*}*@imperial.ac.uk*

Mapping semantics

Sample rule

```
read(fd,s,n) {{ return strstr("put",s) == 0; }}
  =>
    read(fd,ss,n)
      {{
         sscanf(s,"put(%s,%s)",&k,&v);

      }}
```

A DSL Approach to Reconcile Equivalent Divergent Program Executions

Luís Pina Daniel Grumberg Anastasios Andronidis Cristian Cadar
Department of Computing
Imperial College London, UK
{*l.pina, daniel.grumberg14, a.andronidis15, c.cadar*}*@imperial.ac.uk*

Mapping semantics

Sample rule

```
read(fd,s,n) {{ return strstr("put",s) == 0; }}
  =>
    read(fd,ss,n)
      {{
         sscanf(s,"put(%s,%s)",&k,&v);
         sprintf(ss,"put(string,%s,%s)",k,v);
      }}

```

A DSL Approach to Reconcile Equivalent Divergent Program Executions

Luís Pina   Daniel Grumberg   Anastasios Andronidis   Cristian Cadar

Department of Computing

Imperial College London, UK

{l.pina, daniel.grumberg14, a.andronidis15, c.cadar}@imperial.ac.uk

Mapping semantics

Sample rule

```
read(fd,s,n) {{ return strstr("put",s) == 0; }}
  =>
    read(fd,ss,n)
      {{
         sscanf(s,"put(%s,%s)",&k,&v);
         sprintf(ss,"put(string,%s,%s)",k,v);
      }}
      {{ ret += 6; }}
```

A DSL Approach to Reconcile Equivalent Divergent Program Executions

Luís Pina Daniel Grumberg Anastasios Andronidis Cristian Cadar
Department of Computing
Imperial College London, UK
{*l.pina, daniel.grumberg14, a.andronidis15, c.cadar*}*@imperial.ac.uk*

Mapping semantics

MVE