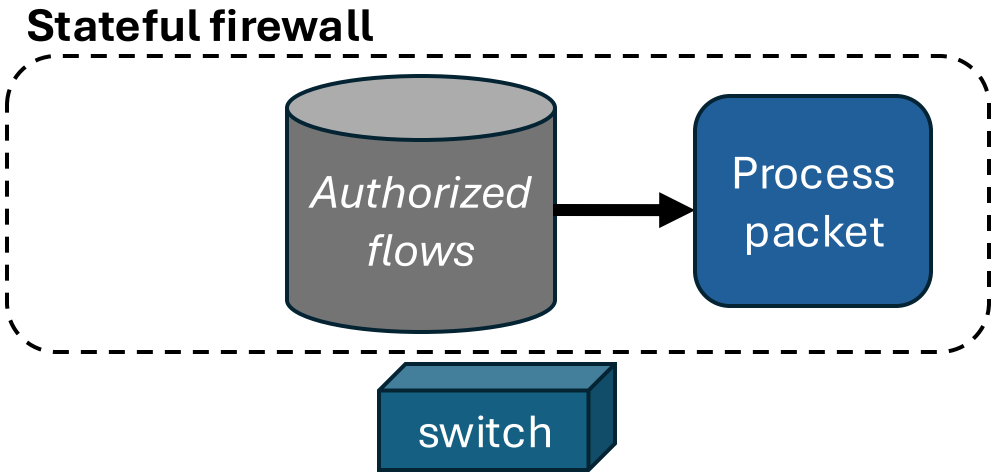
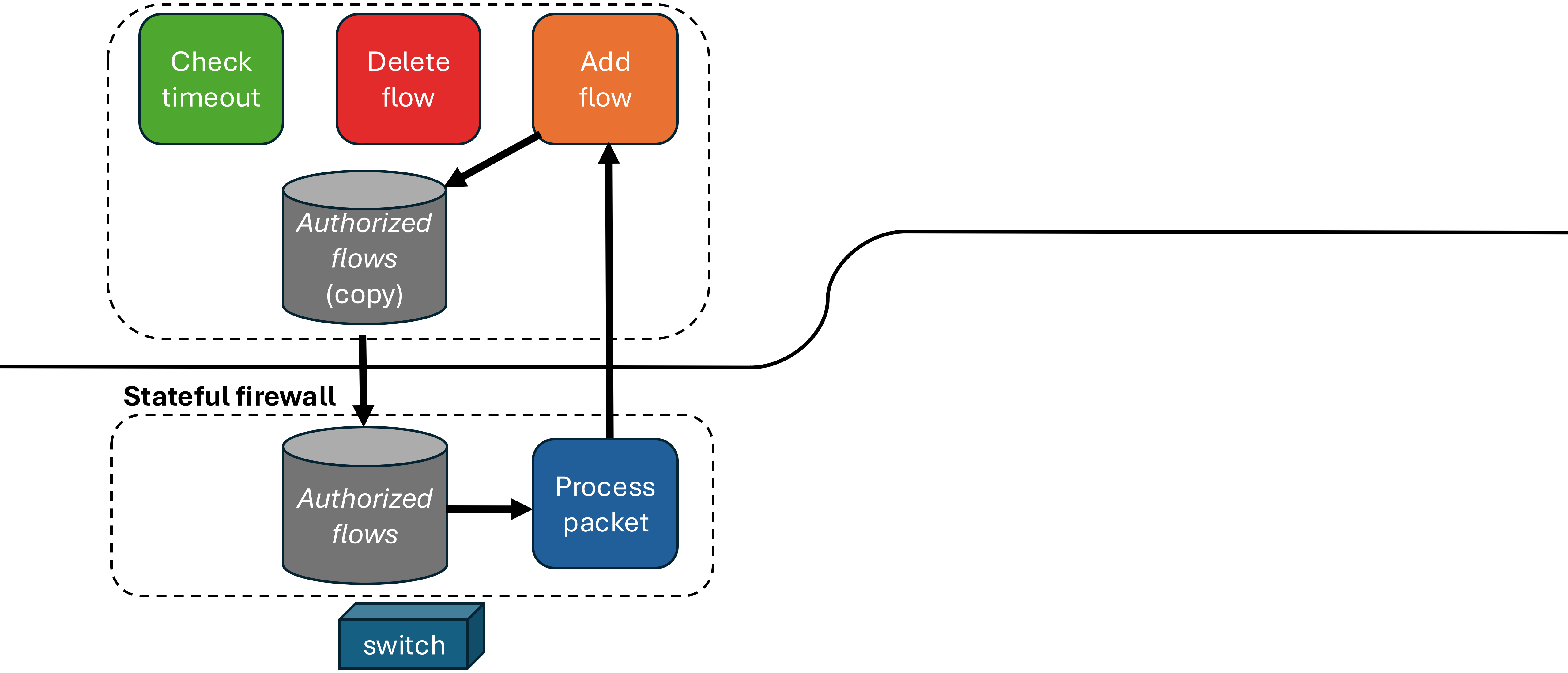
Lucid: A Language for Control in the Data Plane

Judith Hershko & Yonah Thienpont
Future Internet

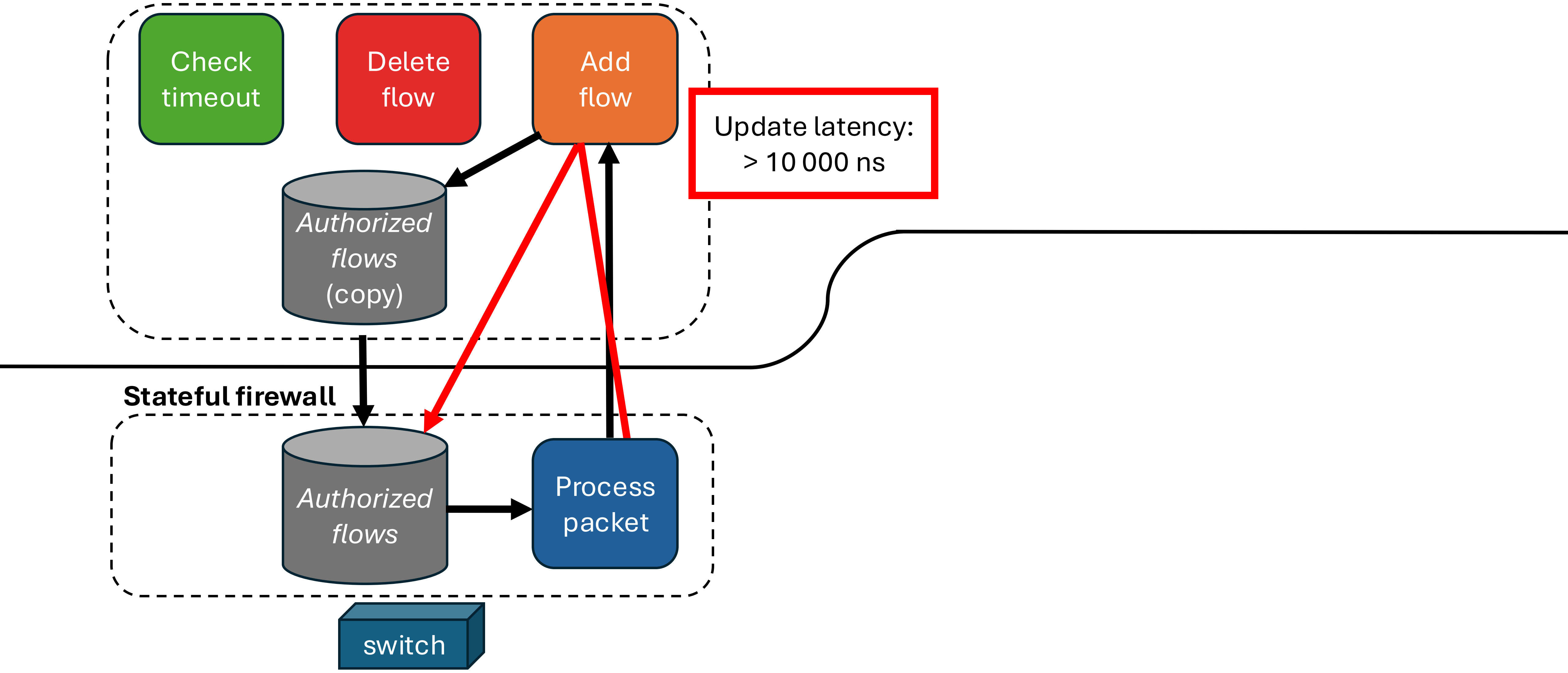
Control in the data plane



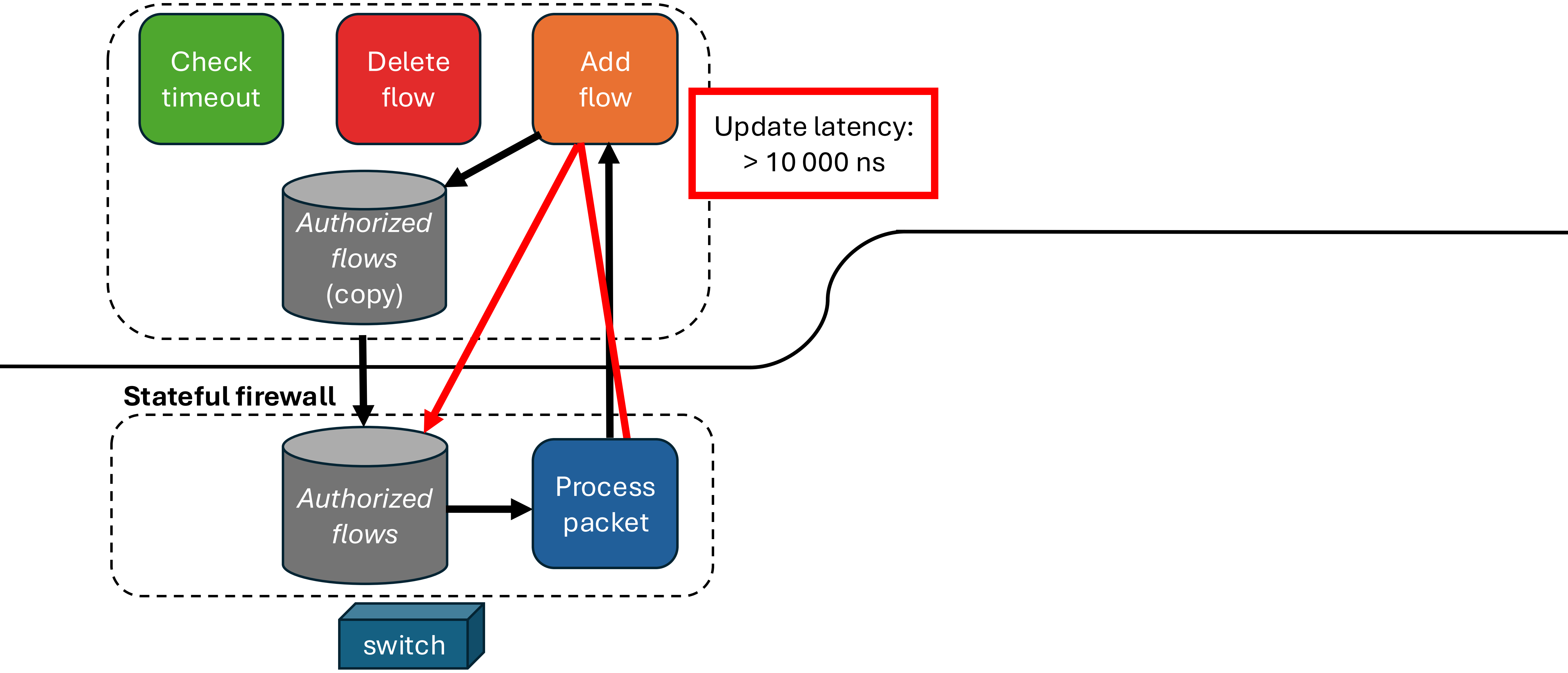
Control in the data plane



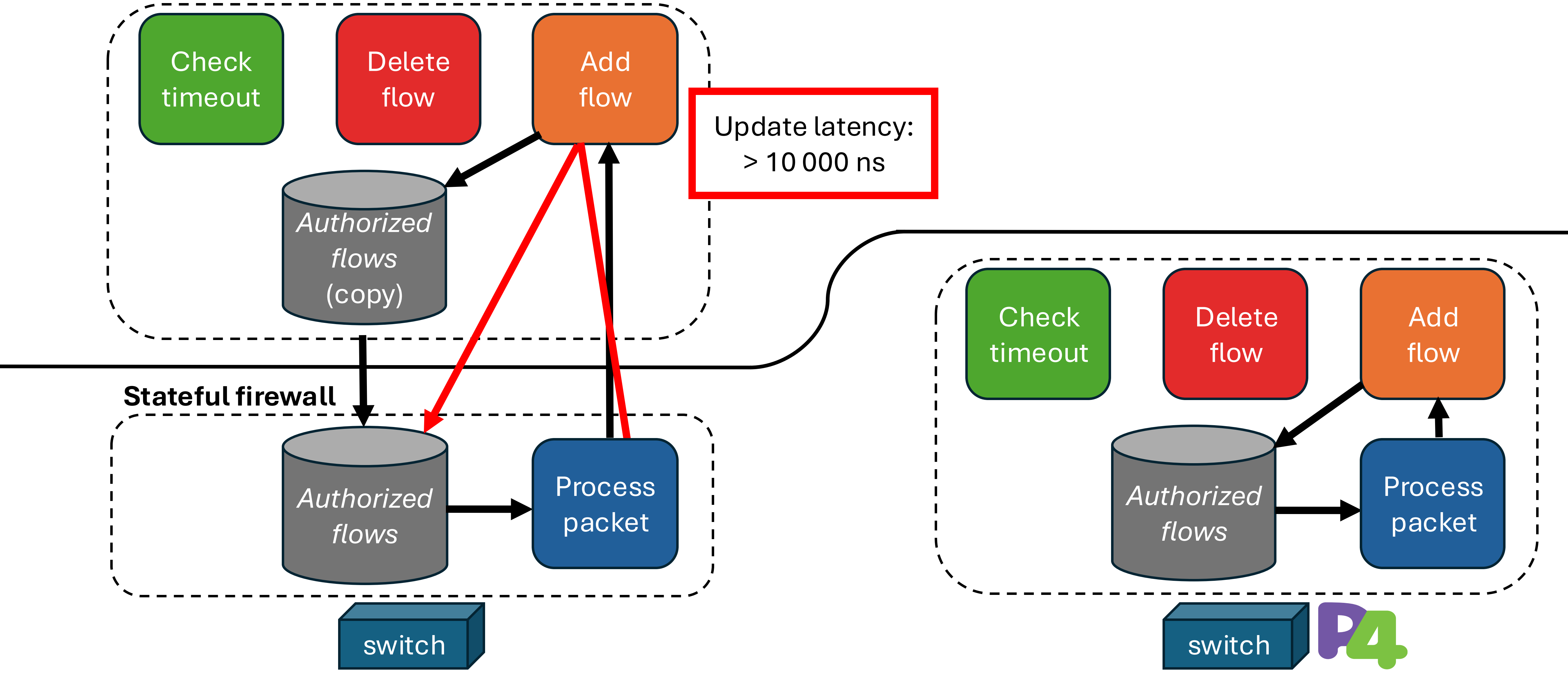
Control in the data plane



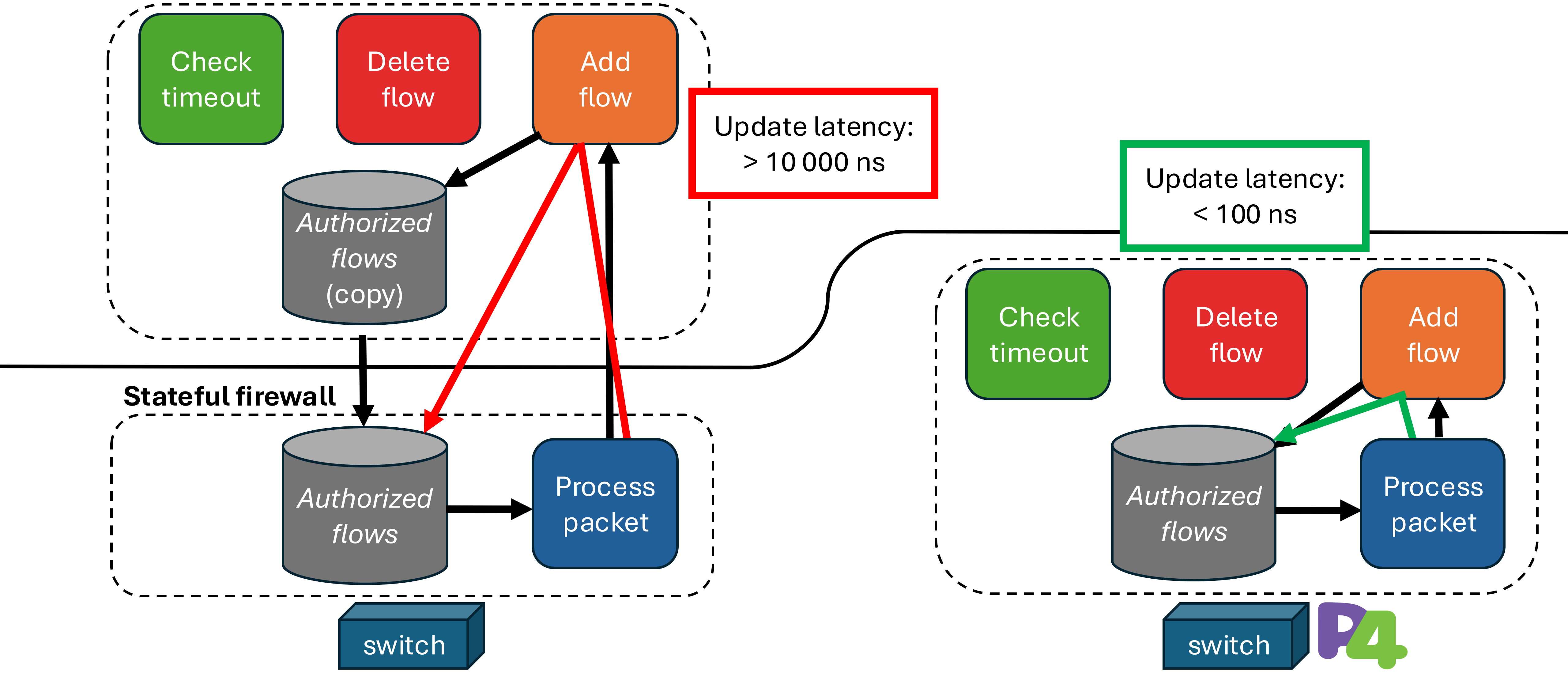
Control in the data plane



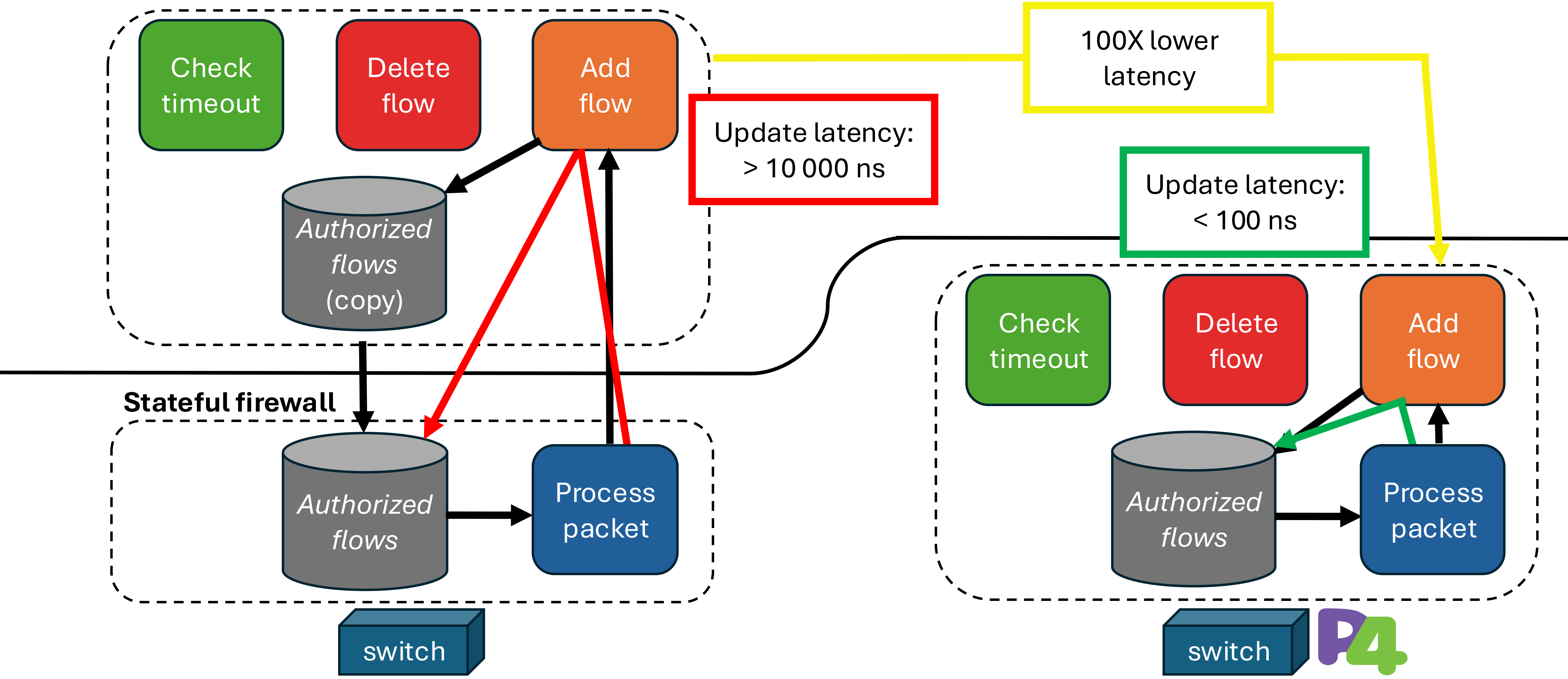
Control in the data plane



Control in the data plane



Control in the data plane



Control in the data plane

- Low latency for performance-critical tasks
- Scalability under high traffic loads
- Reduced communication overhead
- Enhanced reliability
- Enabling new applications

Problem: Control programs are not trivial

Low-level primitives

- Focussed on processing streams of packet headers

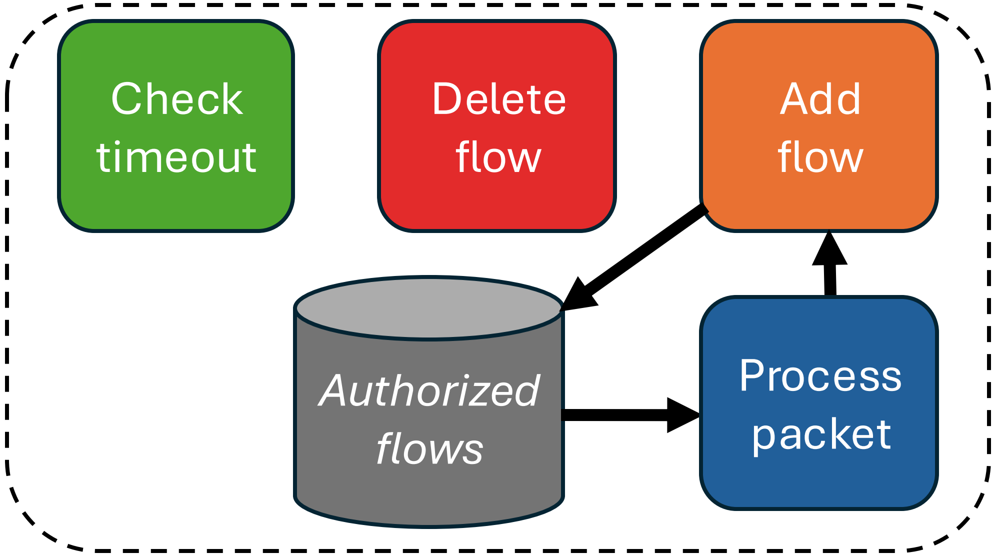
Problem

Low-level primitives

- Focussed on processing streams of packet headers

Stateful firewall

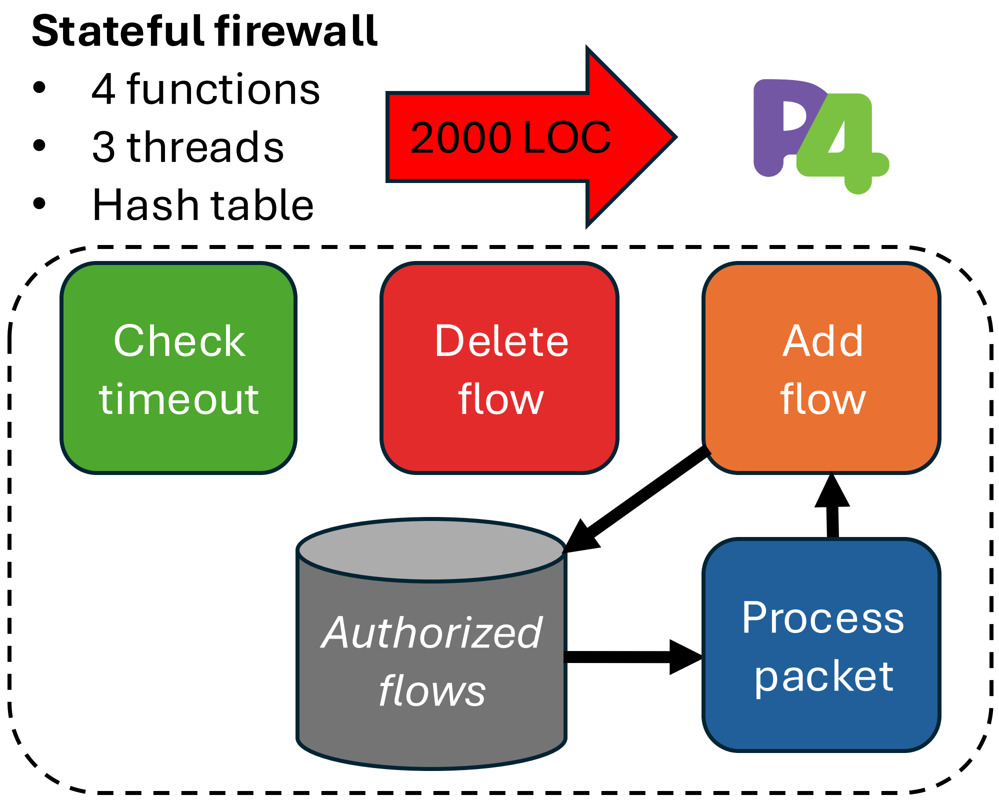
- 4 functions
- 3 threads
- Hash table



Problem

Low-level primitives

- Focussed on processing streams of packet headers
- Stateful firewall example takes 2000 LOC in P4
  - Not modular, hard to maintain, hard to understand



Problem

Low-level primitives

- Focussed on processing streams of packet headers
- Stateful firewall example takes 2000 LOC in P4
  - Not modular, hard to maintain, hard to understand

Low-level constraints

- Hardware constraints not enforced by language
- Operations on persistent state common in control programs
- P4 compiler does *not* handle hardware constraint violations

Lucid abstractions

High-level general purpose primitives

- Events
- Handlers
- Arrays

→ Write ~10x fewer lines of code than in P4

Simple syntax rules

- Ordered arrays
- Memory operation restrictions

→ Build compiling prototypes without hardware expertise

Lucid abstractions

High-level general purpose primitives

- Events
- Handlers
- Arrays

→ *Write ~10x fewer lines of code than in P4*

Simple syntax rules

- Ordered arrays
- Memory operation restrictions

→ *Build compiling prototypes without hardware expertise*

Stateful firewall example ≈ 200 LOC:
https://github.com/PrincetonUniversity/lucid/blob/main/examples/tofino_apps/src/simple_cuckoo_firewall.dpt

Lucid Language Cheat Sheet

Events

```
// declare a kind of event
event foo(type1 id1, type2 id2, ...);

// create an event value
event x = foo(arg1, arg2, ..., argN);

// pack arguments directly into packet
packet event foo(type1 id1, ...);

// send event to port n
generate_port(n, x);

// send event to switches in group g
generate_port(g, x);

// queue event for recirculation
generate(x);
```

Parser Declarations

```
// example parser implementation
parser main(bitstring pkt) {
  int<48> d = read(pkt);
  int<48> s = read(pkt);
  int<16> t = read(pkt);
  match t with
    | LUCID_ETHERTY -> {do_lucid_parsing(
   pkt);}
    | _ -> {
      skip(32, pkt);
      int<16> csum = hash<16>(checksum, d,
    s, t);
      generate(eth_with_csum(csum, d, s, t
   , Payload.parse(pkt))); }
}
```

Parser Actions

```
// read from bitstring
read(pkt);

// pop n bits off bitstring
skip(n, pkt);

// computes the n-bit checksum
hash<n>(checksum, a1, a2, ..., an);
```

Parser Steps

```
// generate event
generate(foo(a1, a2, ..., an));

// call to previously declared parser
my_parser(a1, a2, ..., an);

// drop the packet
drop;

// branch to parse blocks
match proto with
  | 0x800 -> {parse_ipv4(pkt);}
  | _ -> {...}
```

Handlers

```
// define handler for corresponding event
handle foo(type1 id1, type2 id2, ...) {
  ...
}
```

Arrays

```
// array with n entries of sz-bit ints
global Array.t<<sz>> arr = Array.create(n)

// return the value at index
Array.get(arr, idx);

// store the value at index
Array.set(arr, idx, v);

// apply memop setop to v using getarg
Array.getm(arr, idx, getop, getarg);

// apply memop setop to v using setarg
Array.setm(arr, idx, setop, setarg);

// return getop(v, getarg) and replace
    with setop(v, setarg)
Array.update(arr, idx, getop, getarg,
    setop, setarg);

// update for three-argument memop
Array.update_complex(arr, idx, memop, arg1
    , arg2, default)
```

Tables

```
// create a table
global Table.t<<key_ty, data_ty, arg_ty,
    ret_ty>> t = Table.create(sz, actions,
    default_action, default_data);

// table lookup
ret_ty result = Table.lookup(t, key, arg);

// install table entry; vendor dependent
Table.install(t, key, acn, data);
```

Memops

```
// two arguments: two forms

// single return
memop foo(int mem, int local) {
    return <e>;
}

// single if with single return per branch
memop foo(int mem, int local) {
    if (<e>) then { return <e>; } else {
    return <e>; }
}


// four arguments, structured like
memop foo(int mem1, int mem2, int local1,
    int local2) {
  bool b1 = <boolexp>; // May be omitted
  bool b2 = <boolexp>; // May be omitted

  // Omitted entirely, or just else branch
    if (<cond>) { cell1 = <ret_exp> } else
  { if (<cond>) { cell1 = <ret_exp> }

  // Omitted entirely, or just else branch
    if (<cond>) { cell2 = <ret_exp> } else
  { if (<cond>) { cell2 = <ret_exp> }

  // May be omitted. No else permitted
  if (<cond>) { return <local_exp> }
}

// three arguments; no mem2
memop foo(int mem, int loc1, int loc2)
```

Lucid Language Cheat Sheet

Events

```
// declare a kind of event
event foo(type1 id1, type2 id2, ...);

// create an event value
event x = foo(arg1, arg2, ..., argN);

// pack arguments directly into packet
packet event foo(type1 id1, ...);

// send event to port n
generate_port(n, x);

// send event to switches in group g
generate_port(g, x);

// queue event for recirculation
generate(x);
```

Handlers

```
// define handler for corresponding event
handle foo(type1 id1, type2 id2, ...) {
  ...
}
```

```
type eth_t = {
    int<48> dmac;
    int<48> smac;
    int<16> etype;
}
type ip_t = {
    int src;
    int dst;
    int<16> len;
}

event eth_ip(eth_t eth, ip_t ip);

event prepare_report(eth_t eth, ip_t ip);

event report(int src, int dst, int<16> len) {skip;}

handle eth_ip(eth_t eth, ip_t ip) {
    // 1. forward an event representing the packet out of port 1.
    generate_port(OUT_PORT, eth_ip(eth, ip));
    // 2. prepare a report to send to the monitoring server.
    generate(prepare_report(eth, ip));
}

handle prepare_report(eth_t eth, ip_t ip) {
    printf("sending report about packet {src=%d; dst=%d; len=%d} to monitor on port %d", ip#src,
ip#dst, ip#len, SERVER_PORT);
    // make an report event and send it to the monitoring server.
    event r = report(ip#src, ip#dst, ip#len);
    generate_port(SERVER_PORT, r);
}
```

Implementation

(Demo)

Try it yourself

- Project Repository (us): https://github.com/ythienpont/lucid
    - Cheatsheet
    - Demo code
- Lucid Repository: https://github.com/PrincetonUniversity/lucid
    - Lucid interpreter
    - Open source compiler, optimized for Intel Tofino
    - Example programs, including programs for Tofino

Summary

- Writing **control programs** in existing data-plane languages (e.g., P4) is **complex** and **error-prone**
- **Lucid** introduces **event-driven** programming with **high-level abstractions** to interleave control and packet processing seamlessly
- Guarantees programs fit hardware constraints via "**correct-by-construction**" checks
- **Simplifies programming**, enabling fast prototyping even for newcomers

References

- https://www.cs.princeton.edu/~dpw/papers/lucid-SIGCOMM-2021.pdf
- https://dl.acm.org/doi/10.1145/3452296.3472903
- https://github.com/PrincetonUniversity/lucid