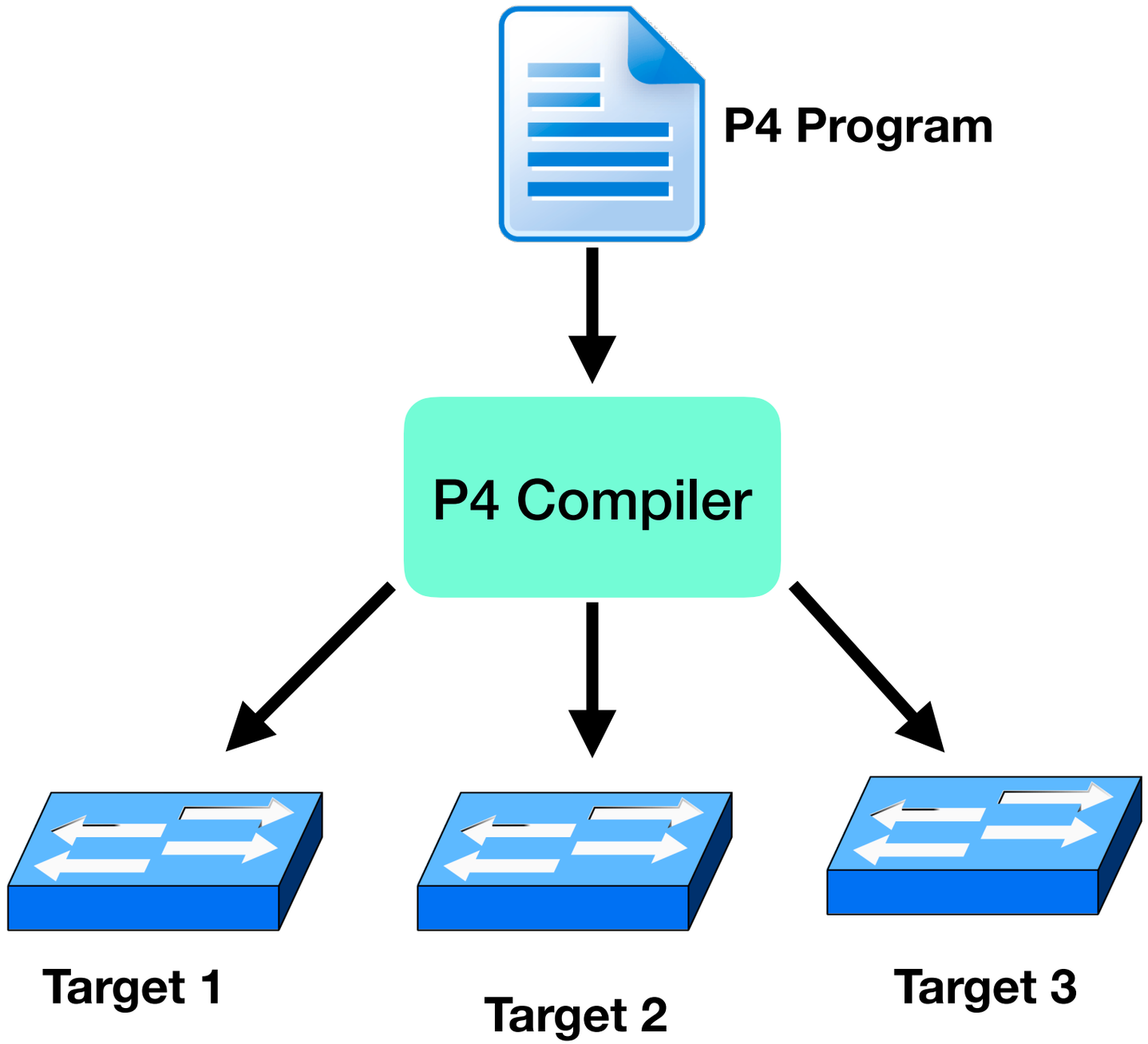
P4All: Enabling Parametrized Description of Data-Plane Algorithms

Mary Hogan*, Shir Landau Feibish*, Mina Tahmasbi Arashloo+, Rob Harrison++, Jennifer Rexford*, David Walker*

*Princeton University, +Cornell University, ++United States Military Academy

P4 is target independent



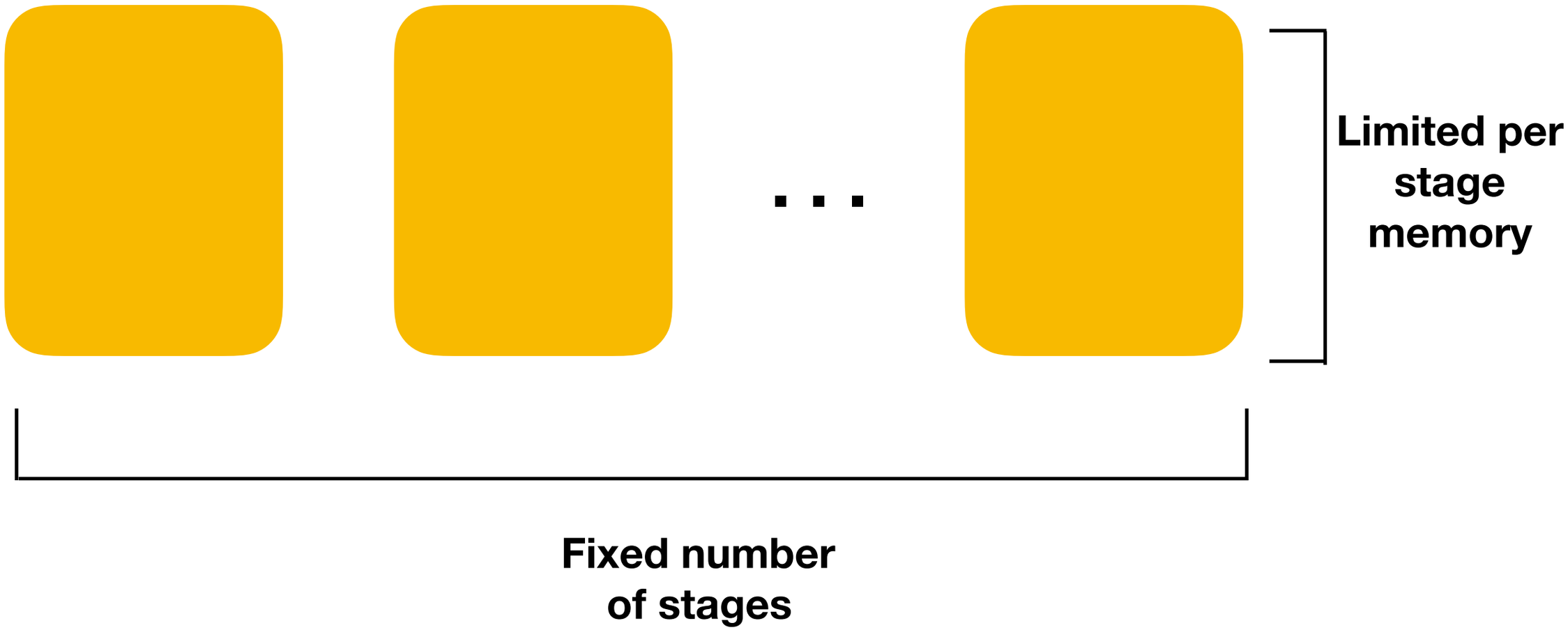
P4 is target independent

Data structures (per flow counters, streaming algorithms, etc.) are valid for a range of sizes

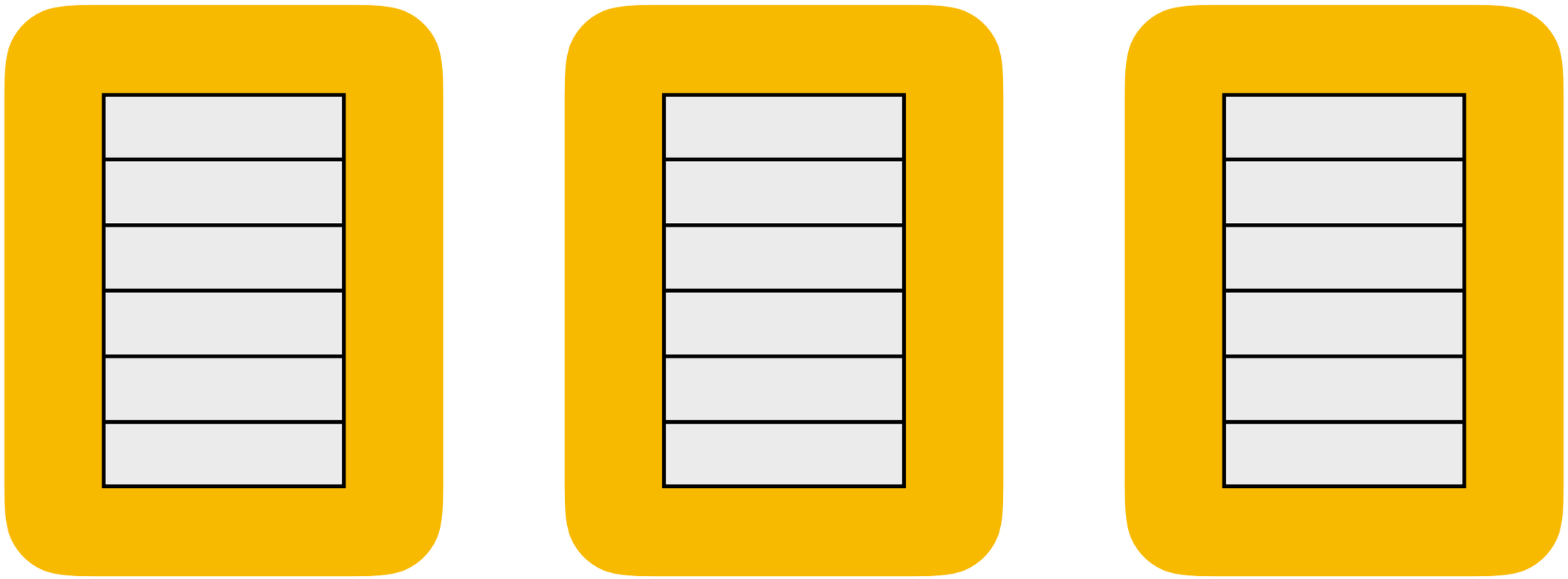
P4 data structures must be adapted to fit specific targets

P4 requires explicit definition of size - amount of memory, logical tables, etc.

Limited Switch Resources



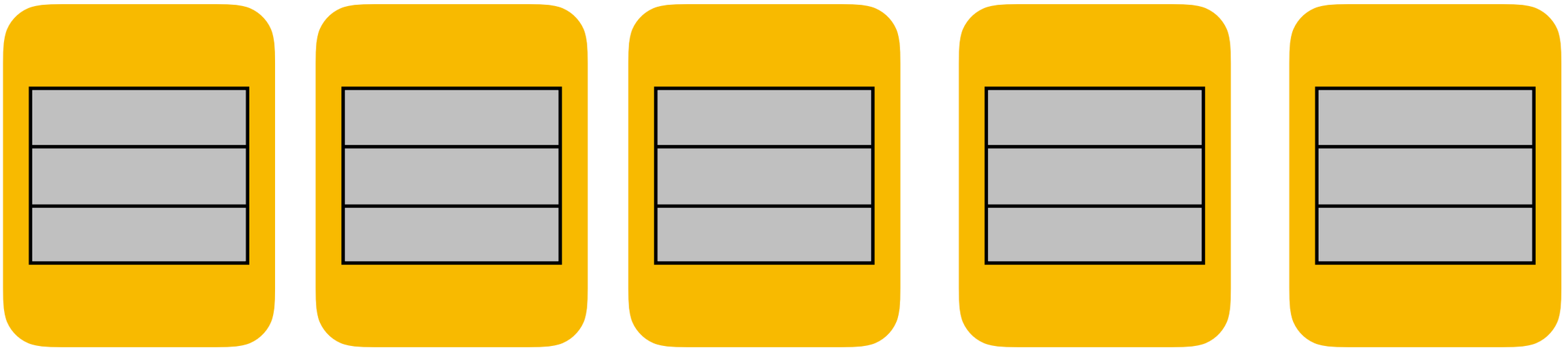
P4 is target dependent



3 stages

3x6 matrix

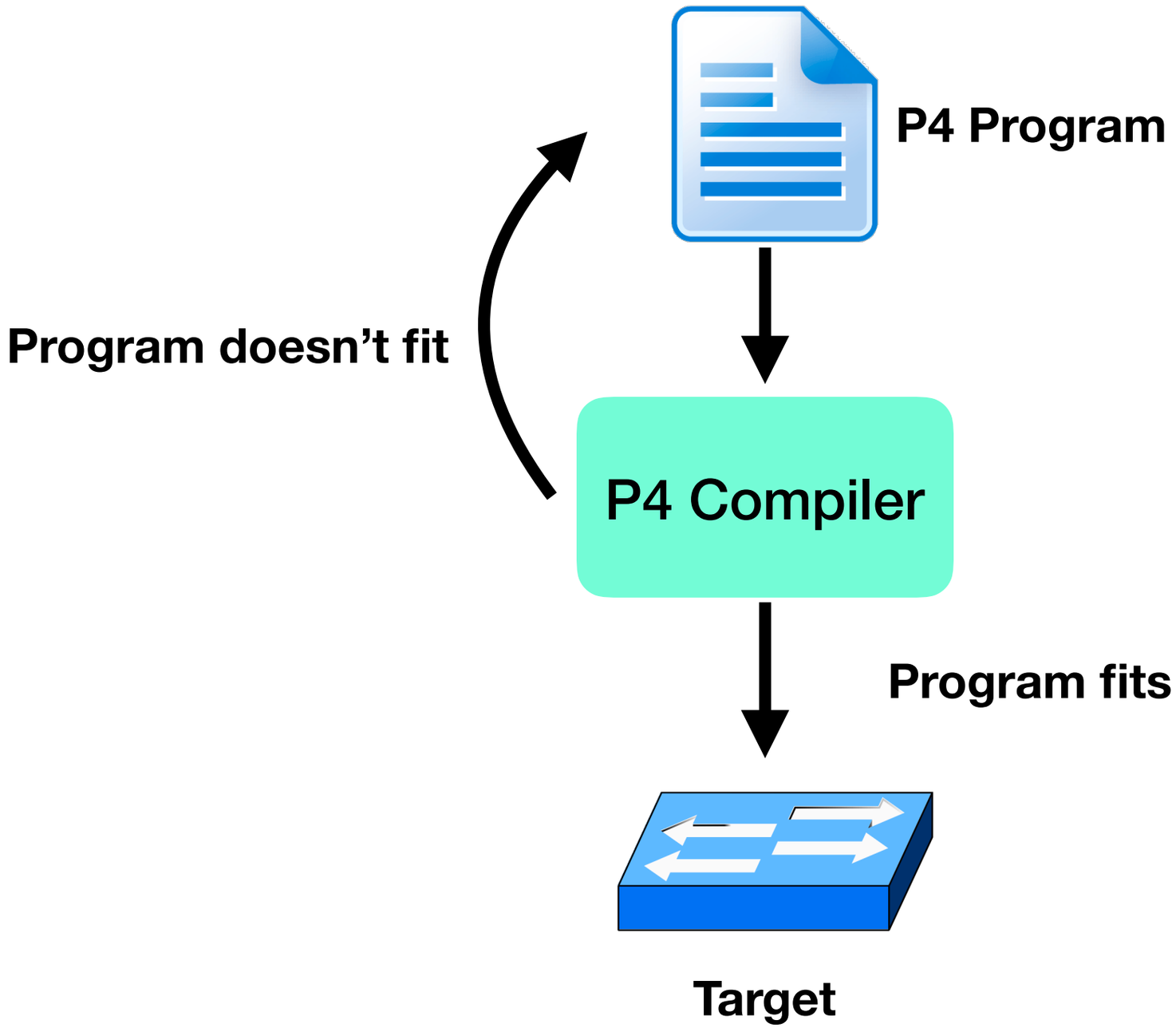
P4 is target dependent



5 stages

5x3 matrix

Circular Development



P4All is target independent

P4All is a parameterized extension to P4

Minimal constructs added to P4 to allow for general data structures

P4All compiler determines parameter values for individual targets

P4All vs P4

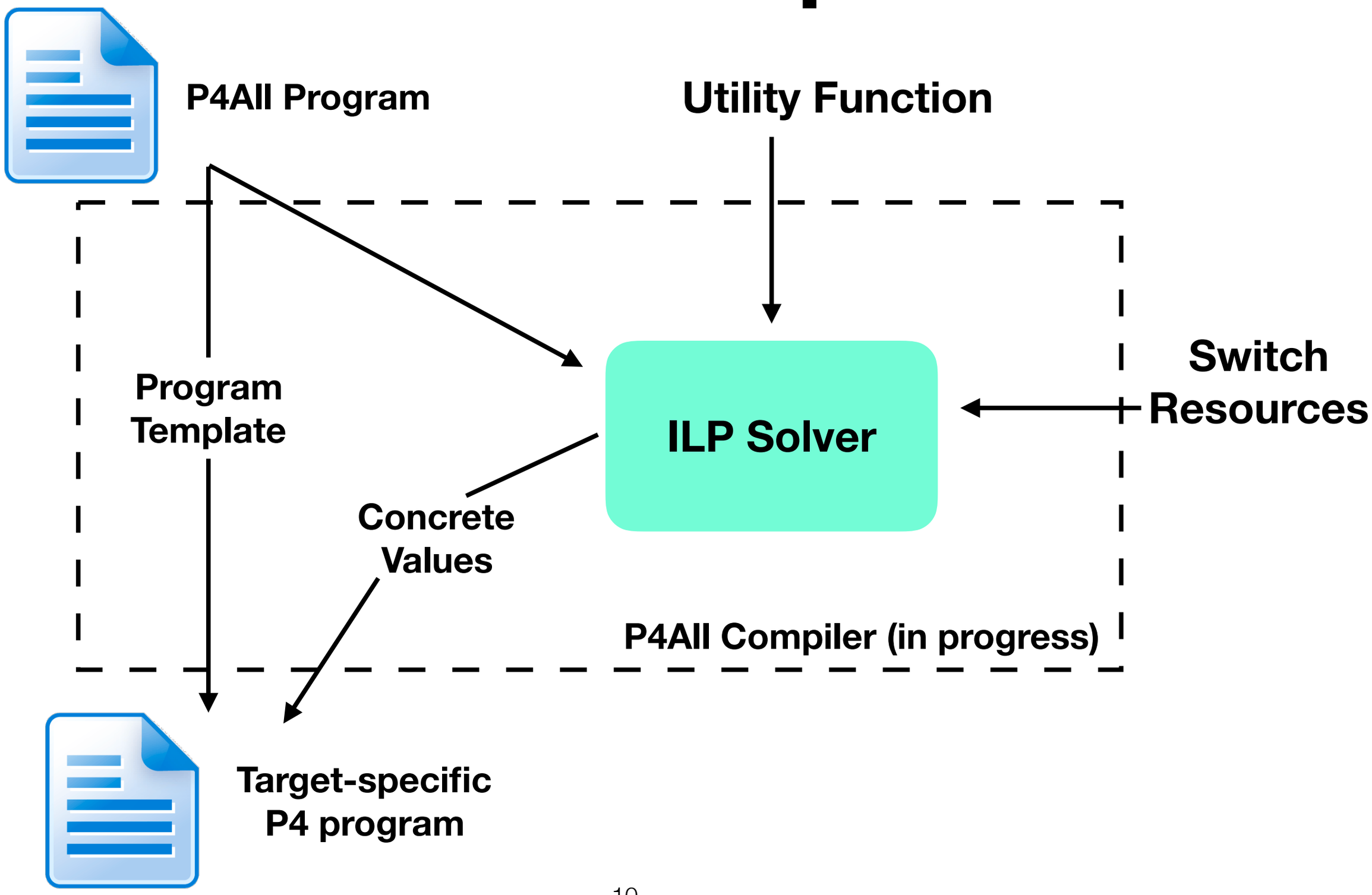
```
symbolic int num_columns;
symbolic int num_rows;

register<bit<32>>(num_columns) counter;
counter[num_rows] counters;
.
.
.
for (int i = 0; i < num_rows; i++) {
    write_to_row(i);
}
```

P4All

```
register<bit<32>>(256) reg1;
register<bit<32>>(256) reg2;
register<bit<32>>(256) reg3;
.
.
.
write_to_row_1();
write_to_row_2();
write_to_row_3();
```

P4

P4All Compiler



Status + Future Work

Data Structures
Count-Min Sketch
Key Value Store
Bloom Filter
Per-Flow Counters

Switch Resources
Pipeline stages
Per stage memory
Concurrent memory accesses
Metadata

P4All: Enabling Parametrized Description of Data-Plane Algorithms

Mary Hogan*, Shir Landau Feibish*, Mina Tahmasbi Arashloo+, Rob Harrison++, Jennifer Rexford*, David Walker*

*Princeton University, +Cornell University, ++United States Military Academy

Backup slides

CMS in P4All

```
symbolic int num_rows;
symbolic int num_columns;

. . .

control MyIngress(inout headers hdr,
                  inout custom_metadata_t meta,
                  inout standard_metadata_t standard_metadata) {

        register<bit<32>>(num_columns) counter;
        counter[num_rows] counters;

        bit<32> tmp_count;

        action count()[int i] {
                /* compute hash index */
                hash(meta.index, HashAlgorithm.crc16, HASH_BASE,
                      {hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, hdr.ipv4.protocol}, HASH_MAX);

                /* increment counter  - read, increment, write*/
                counters[i].read(tmp_count, meta.index);
                meta.current_count = tmp_count + 1;
                counters[i].write(meta.index, meta.current_count);
        }

        action set_min(){
                meta.count_min = meta.current_count;
        }

        apply {
                for (int i = 0; i < num_rows; i++) {
                        count()[i];
                        if (meta.current_count < meta.count_min) {
                                set_min();
                        }
                }
        }
}
```

ILP Constraints

ILP Constraints
Number of stages
Per stage memory
Concurrent memory accesses
Action dependencies
Amount of metadata

CMS in P4

```
control MyIngress(inout headers hdr,
                  inout custom_metadata_t meta,
                  inout standard_metadata_t standard_metadata) {

        register<bit<32>>(2048) counter0;
        register<bit<32>>(2048) counter1;

        action count0() {
                /* compute hash index */
                hash(meta.index, HashAlgorithm.crc16, HASH_BASE,
                      {hdr.ipv4.srcAddr, hdr.ipv4.dstAddr, hdr.ipv4.protocol}, HASH_MAX);

                /* increment counter  - read, increment, write*/
                counter0.read(tmp_count, meta.index);
                meta.current_count = tmp_count + 1;
                counter0.write(meta.index, meta.current_count);
        }

        action count1() {
                . . .
        }

        action set_min(){
                meta.count_min = meta.current_count;
        }

        apply {
                count_0();
                if (meta.current_count < meta.count_min) {
                        set_min_0();
                }
                count_1();
                if (meta.current_count < meta.count_min) {
                        set_min_1();
                }
        }
}
```