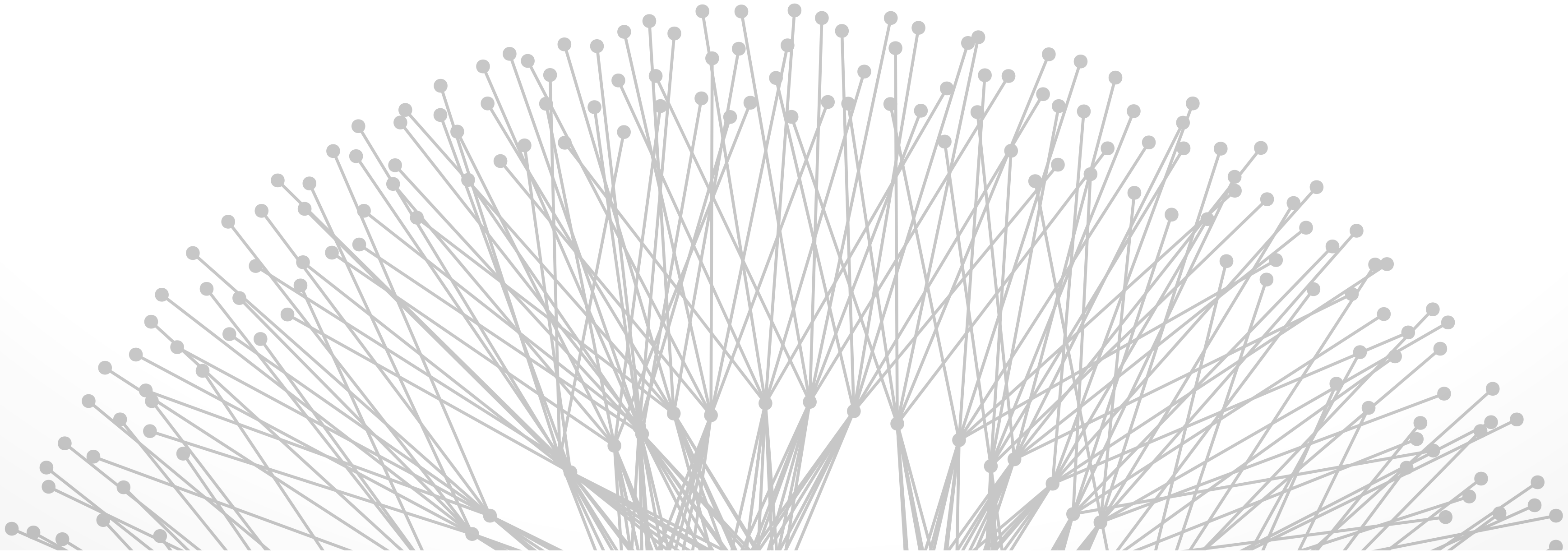
Programmable Switches

Brighten Godfrey
CS 538 April 12 2017



History

Network processors

Active networks (~ 1999)

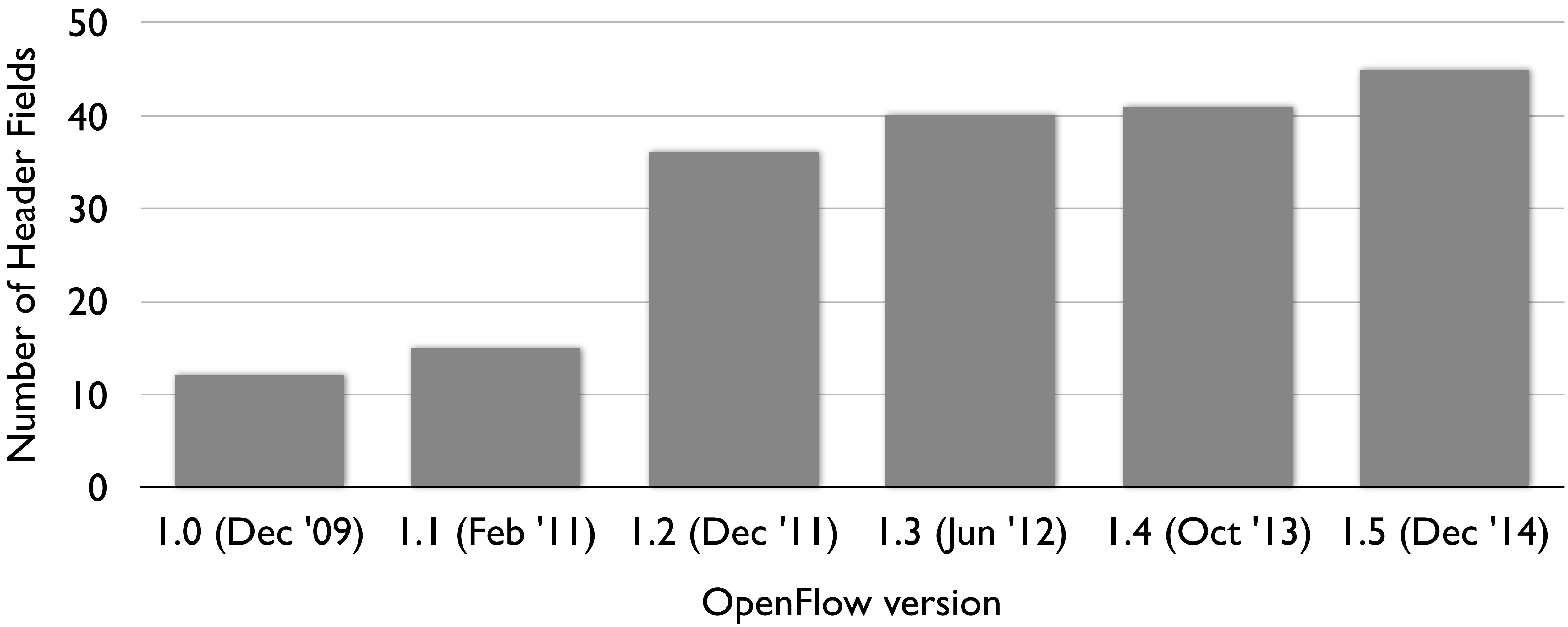
FPGAs (NetFPGA: Lockwood et al, 2007)

Software-Defined Networking (2008)

Drivers of programmable switches

Programmability of the network => SDN

Simplify and future-proof OpenFlow



```
/* OXM Flow match field types for OpenFlow basic class. */
```

Drivers of programmable switches

Programmability of the network => SDN

Simplify and future-proof OpenFlow

New capabilities — *ideas?* [5-min group discussion]

Drivers of programmable switches

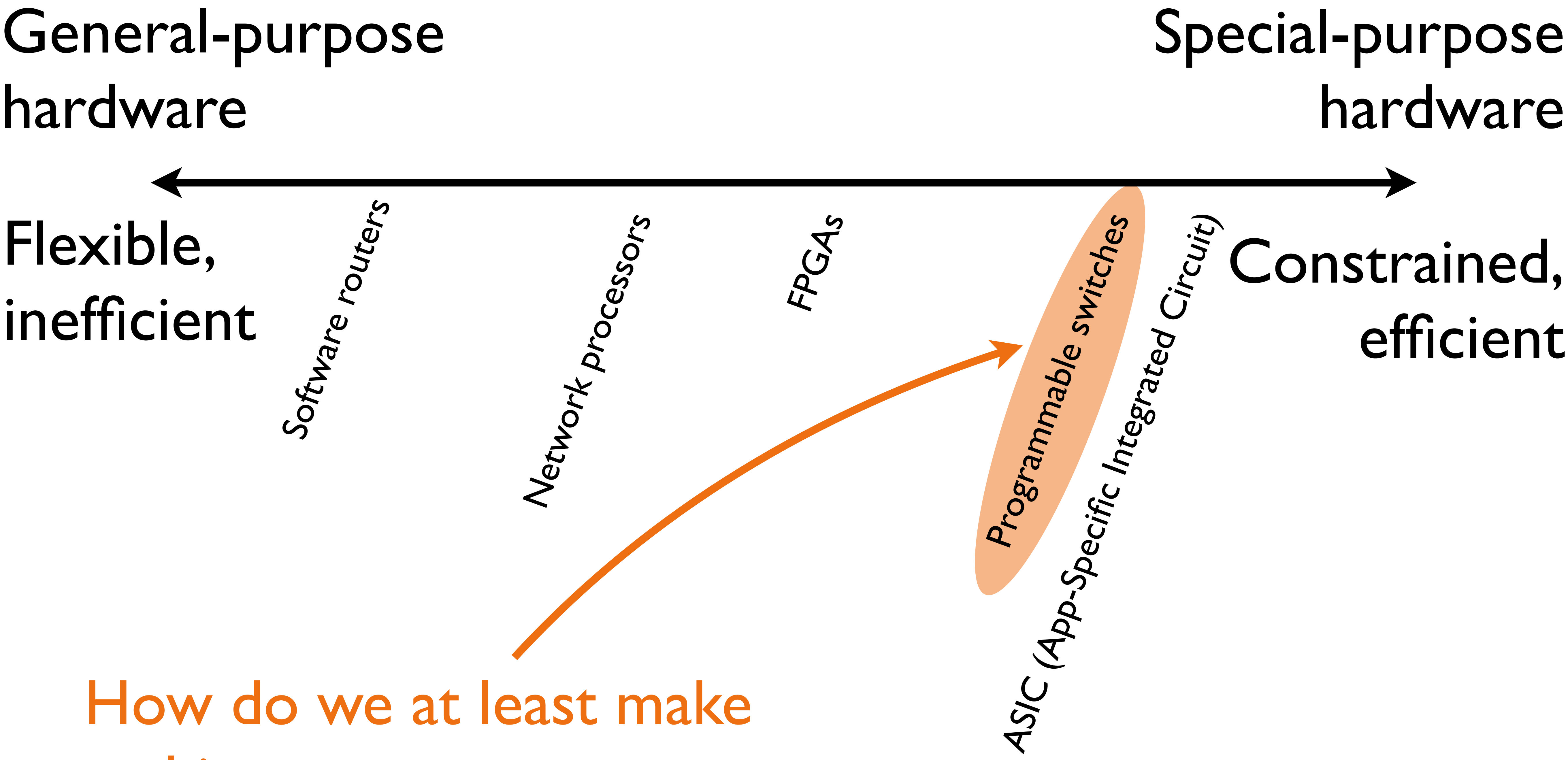
Programmability of the network => SDN

Simplify and future-proof OpenFlow

New capabilities — *ideas?* [5-min group discussion]

- Simplified data planes
- Customizable queueing algorithms
- Fine-grained monitoring
  - e.g. monitor individual flows or microbursts
  - see Barefoot + AT&T + SnapRoute announcement, April 2017

Key tension



General-purpose hardware

Special-purpose hardware

Flexible, inefficient

Constrained, efficient

Software routers

Network processors

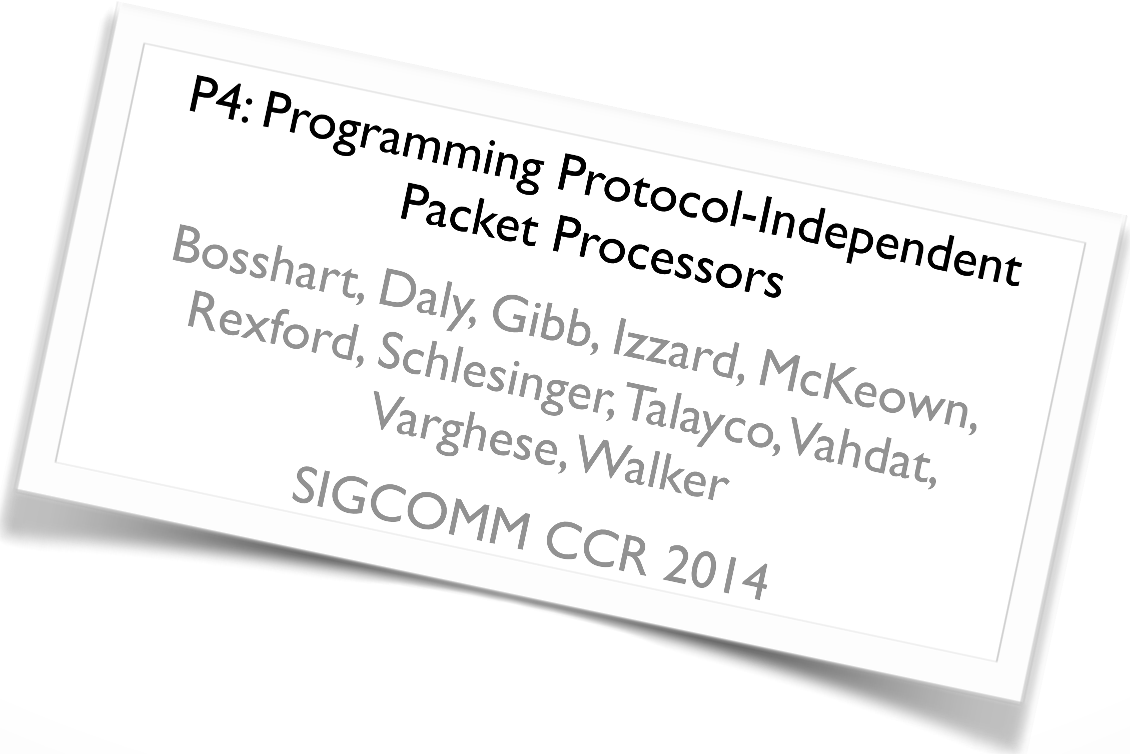
FPGAs

Programmable switches

ASIC (App-Specific Integrated Circuit)

How do we at least make this easy to program, even if it’s not fully flexible?

P4 Introduction



P4: Programming Protocol-Independent Packet Processors

Bosshart, Daly, Gibb, Izzard, McKeown, Rexford, Schlesinger, Talayco, Vahdat, Varghese, Walker

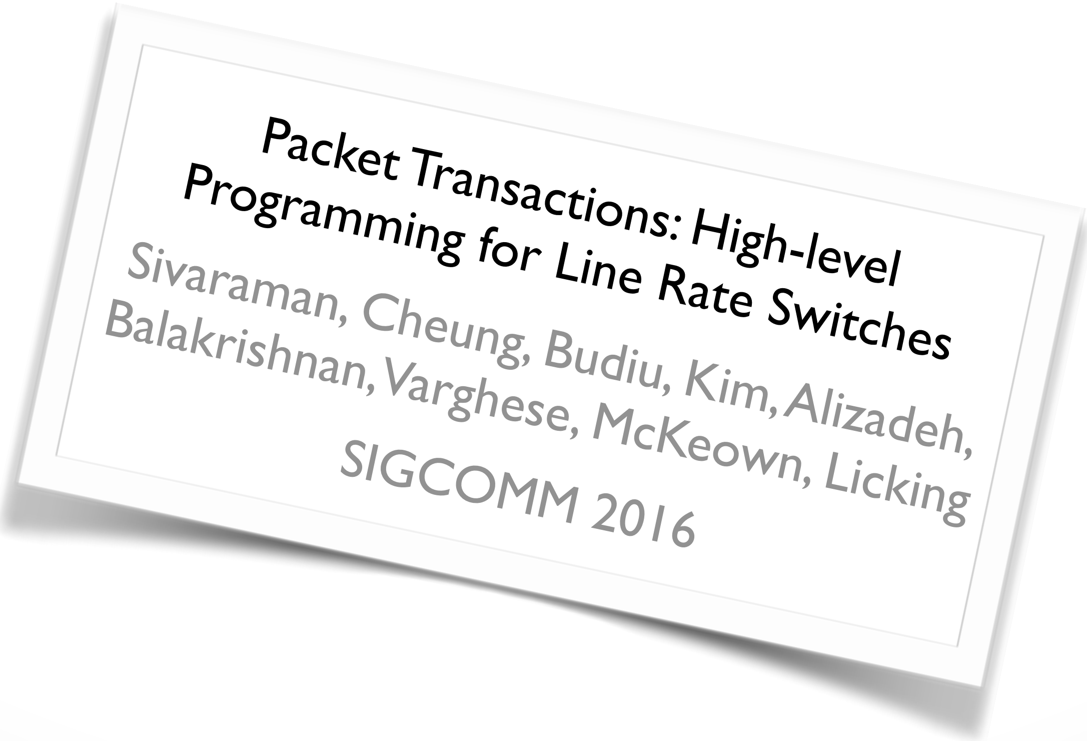
SIGCOMM CCR 2014

P4 discussion

It’s pretty low level; what does it do for you?

- Compiles parser
- Compiles imperative control-flow spec to table dependency graph
    - Compiler looks for opportunities for parallelism
- Unified hardware-independent standard
    - Intermediate table dependency graph mapped to actual hardware by target-specific back-end
    - Software switch, hardware switch with TCAM, various constraints on table size or number of tables, …

Packet Transactions



Packet Transactions: High-level Programming for Line Rate Switches

Sivaraman, Cheung, Budiu, Kim, Alizadeh, Balakrishnan, Varghese, McKeown, Licking

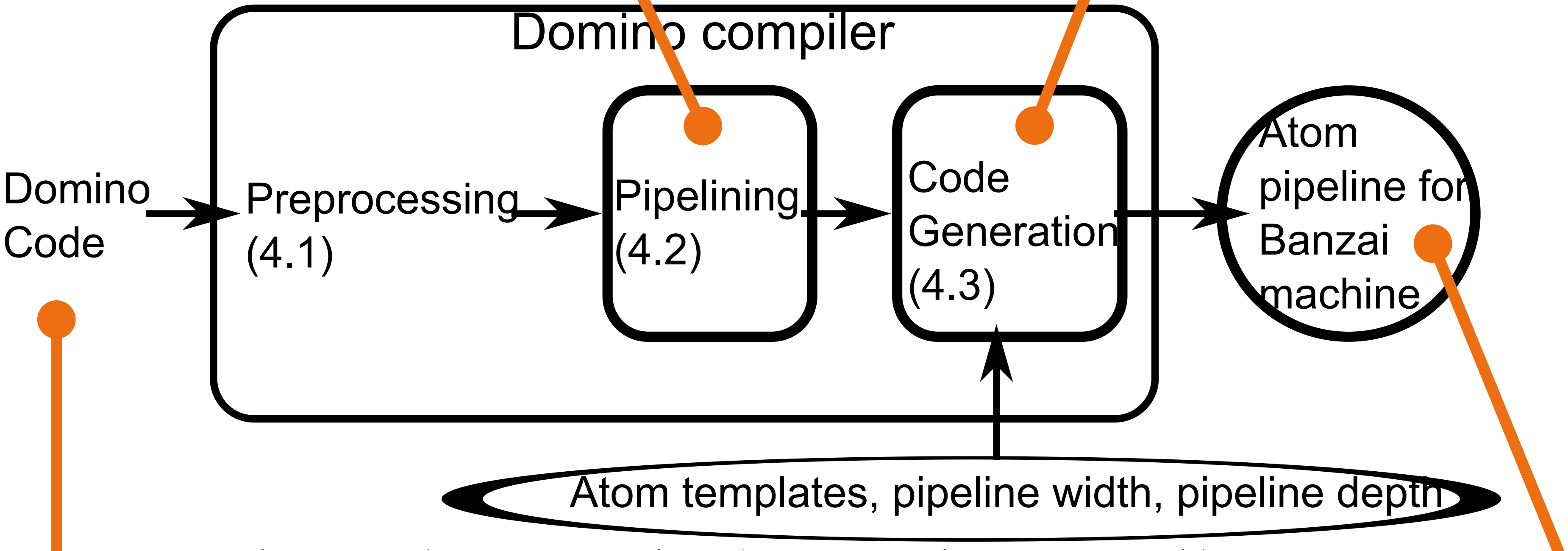
SIGCOMM 2016

Packet Transactions

What does Domino do for you?

- Stateful operations, atomic for each packet
  - but local to the processing element
- Higher-level language (C-like; no need to specify tables)
- Automagically compiles using program synthesis

Domino key features



Dependency graph of ‘codelets’

Use SKETCH to automatically find hardware’s ‘atoms’ that can implement each codelet

High level language

Key is to minimize delay (area not as big a deal)

[Figure from Sivaraman et al.]

Ex.: Fair Queueing prioritization

[https://github.com/packet-transactions/domino-examples]

```
#include "hashes.h"

#define NUM_FLOWS 8000
#define TIME_MIN 1

struct Packet {
  int sport;
  int dport;
  int id;
  int start;
  int length;
  int virtual_time;
};

int last_finish [NUM_FLOWS] = {TIME_MIN};

void stfq(struct Packet pkt) {
  pkt.id  = hash2(pkt.sport,
                  pkt.dport)
            % NUM_FLOWS;
```

Ex.: Fair Queueing prioritization

[https://github.com/packet-transactions/domino-examples]

```
  int dport;
  int id;
  int start;
  int length;
  int virtual_time;
};

int last_finish [NUM_FLOWS] = {TIME_MIN};

void stfq(struct Packet pkt) {
  pkt.id  = hash2(pkt.sport,
                  pkt.dport)
            % NUM_FLOWS;

  if ((last_finish[pkt.id] > TIME_MIN) && (pkt.virtual_time <
last_finish[pkt.id])) {
    pkt.start = last_finish[pkt.id];
    last_finish[pkt.id] += pkt.length;
  } else {
    pkt.start = pkt.virtual_time;
    last_finish[pkt.id] = pkt.virtual_time + pkt.length;
  }
}
```

Discussion

What code will be placed within a pipeline? What will be placed across multiple pipelines?

Domino models the computation "but not how packets are matched (e.g., direct or ternary)" – what do those mean?

How did Domino navigate the tradeoff between efficiency and ease of programmability?

Announcements

Assignment 2

- Release date delayed till next class

Monday

- Content Distribution & Overlay Networks