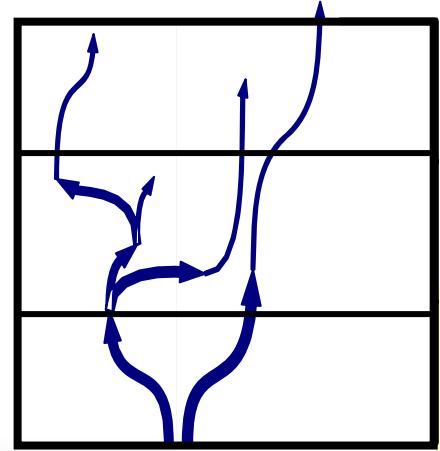


Streamline

[tutorial]



Willem de Bruijn

Herbert Bos

Vrije Universiteit Amsterdam





Goals

Goals

- Major:
 - faster
 - more flexibletraffic processing
- Minor:
 - new networking subsystem
 - use new hardware



Difficulties

- hardware:
 - widening memory gap
 - fast photons, slow silicon
- software/OS:
 - copies
 - context switching
 - memory allocation
- demand:
 - need for expensive processing (like IDS)



Goals (derived)

- cater to
 - individual packets
 - streams
- domain
 - monitoring
 - transmission
- exploit hardware
- retain what is useful

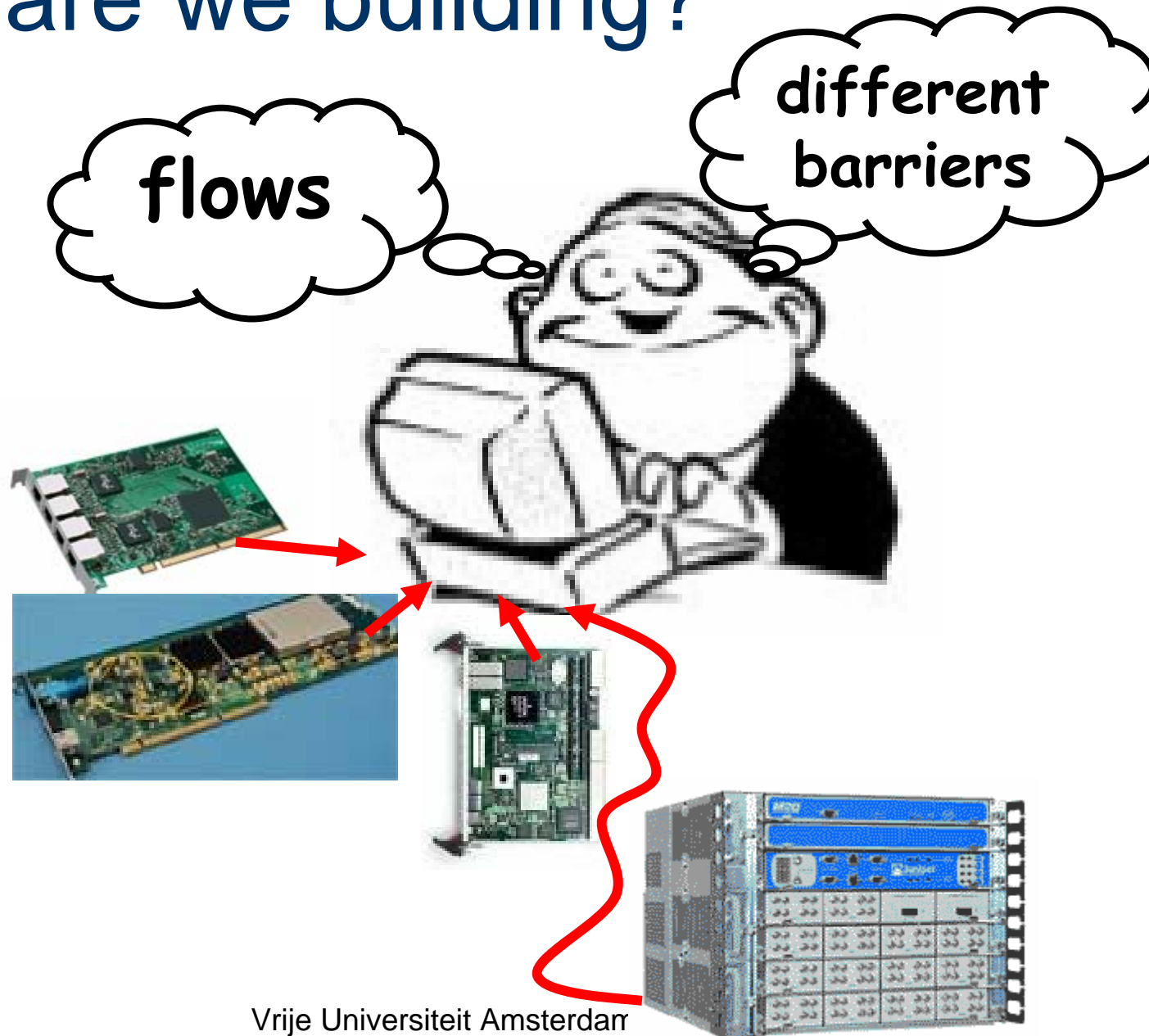
NAPI

BPF

Files

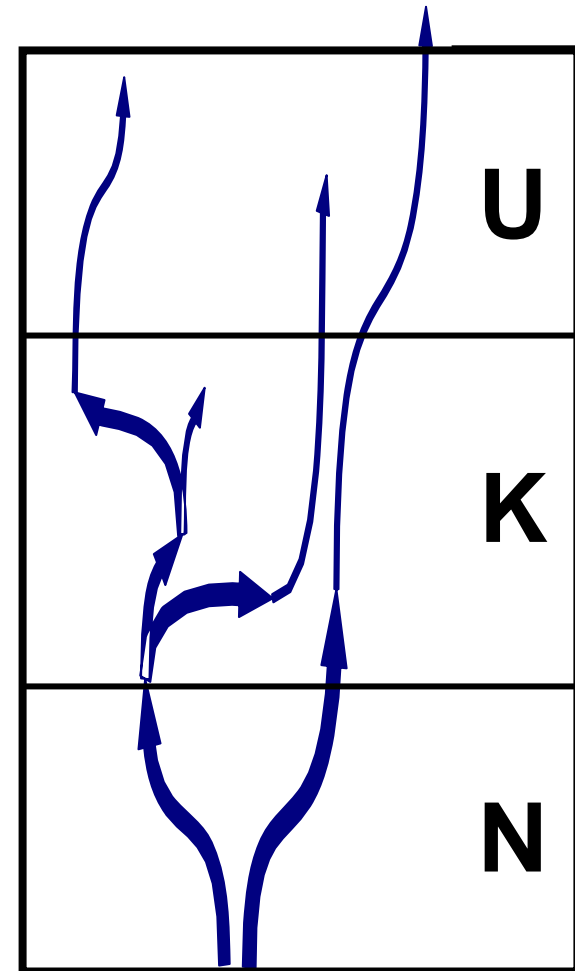


What are we building?



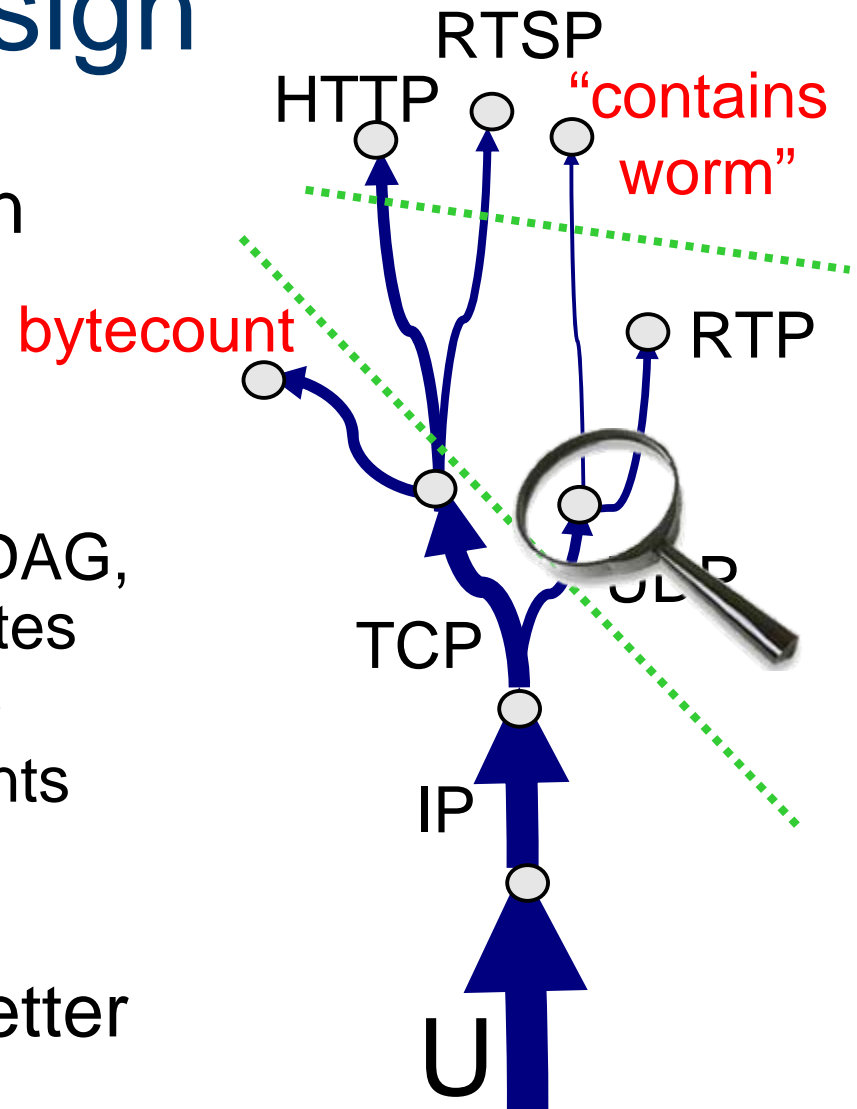
Streamline design

- single framework for pkt processing that
 - uses **all levels** in the processing hierarchy
 - is **language neutral**
 - offers advanced **processing in NIC**
 - supports **stateless** and **stateful** filters
 - is backward compatible with **pcap** (while also supporting much more **powerful packet languages**)
 - helps users to build complex monitoring applications by '**clicking components together**'



Streamline design

- specify processing in terms of a DAG
- Streamline handles instantiation
 - for each function in DAG, Streamline investigates potential placements
 - collective requirements determine definite placement
- Heuristic: lower is better





Buffers

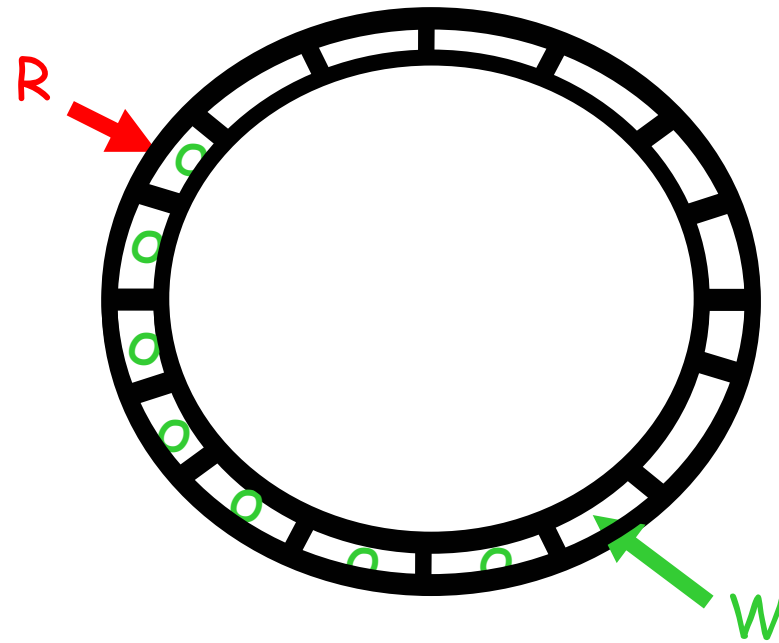
Buffers

■ PacketBuf

- circular buffer with N slots
- e.g., large enough to hold packet

■ IndexBuf

- circular buffer with N slots
- pointers to packets in PacketBuf



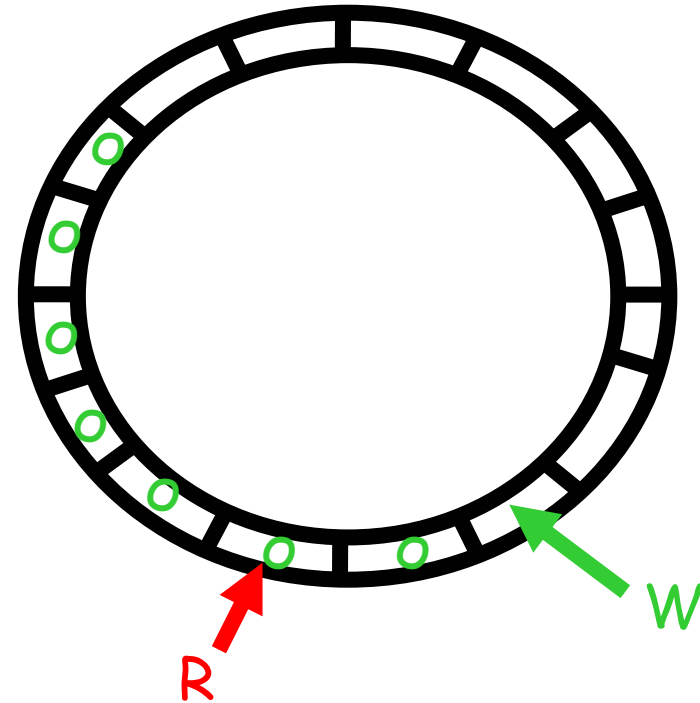
Buffers

■ PacketBuf

- circular buffer with N slots
- e.g., large enough to hold packet

■ IndexBuf

- circular buffer with N slots
- pointers to packets in PacketBuf



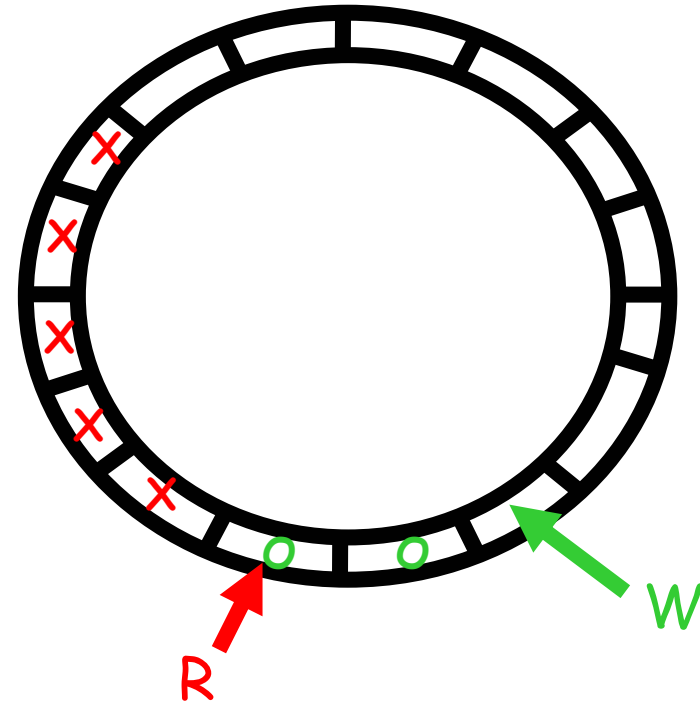
Buffers

■ PacketBuf

- circular buffer with N slots
- e.g., large enough to hold packet

■ IndexBuf

- circular buffer with N slots
- pointers to packets in PacketBuf



Buffer management

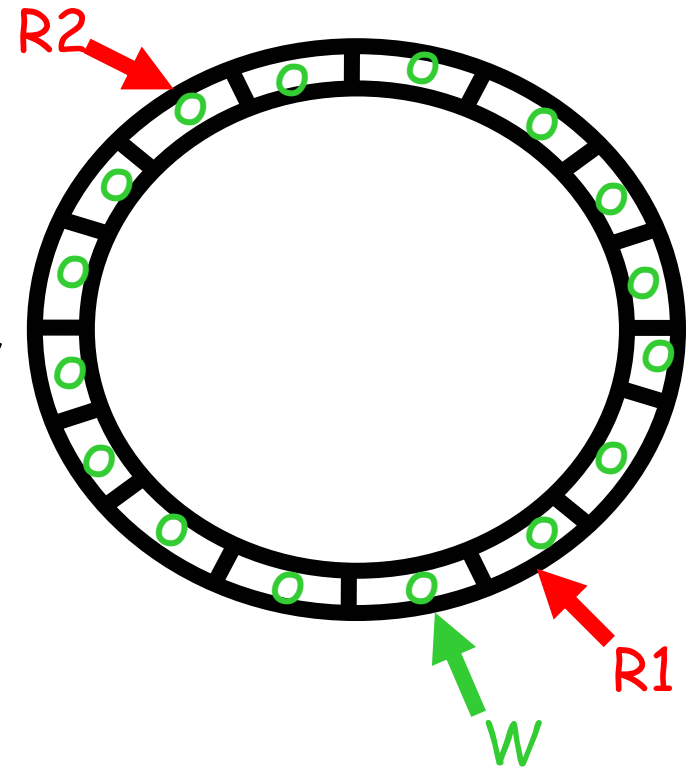
→ what to do if writer catches up with slowest reader?

■ slow reader preference

- drop new packets
(traditional way of dealing with this)
- overall speed set by slowest reader

■ fast reader preference

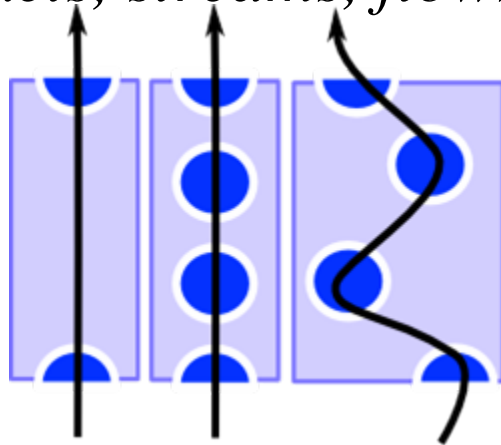
- overwrite existing packets
- application responsible for keeping up
 - check whether packets have been overwritten
 - different drop rates for different apps



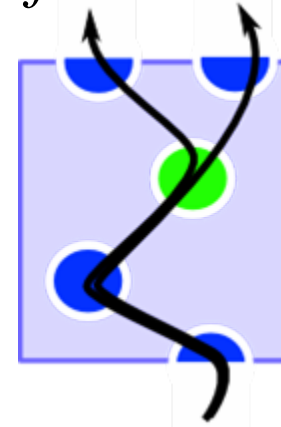
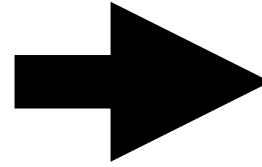


Basic abstraction: flow

sockets, streams, flows

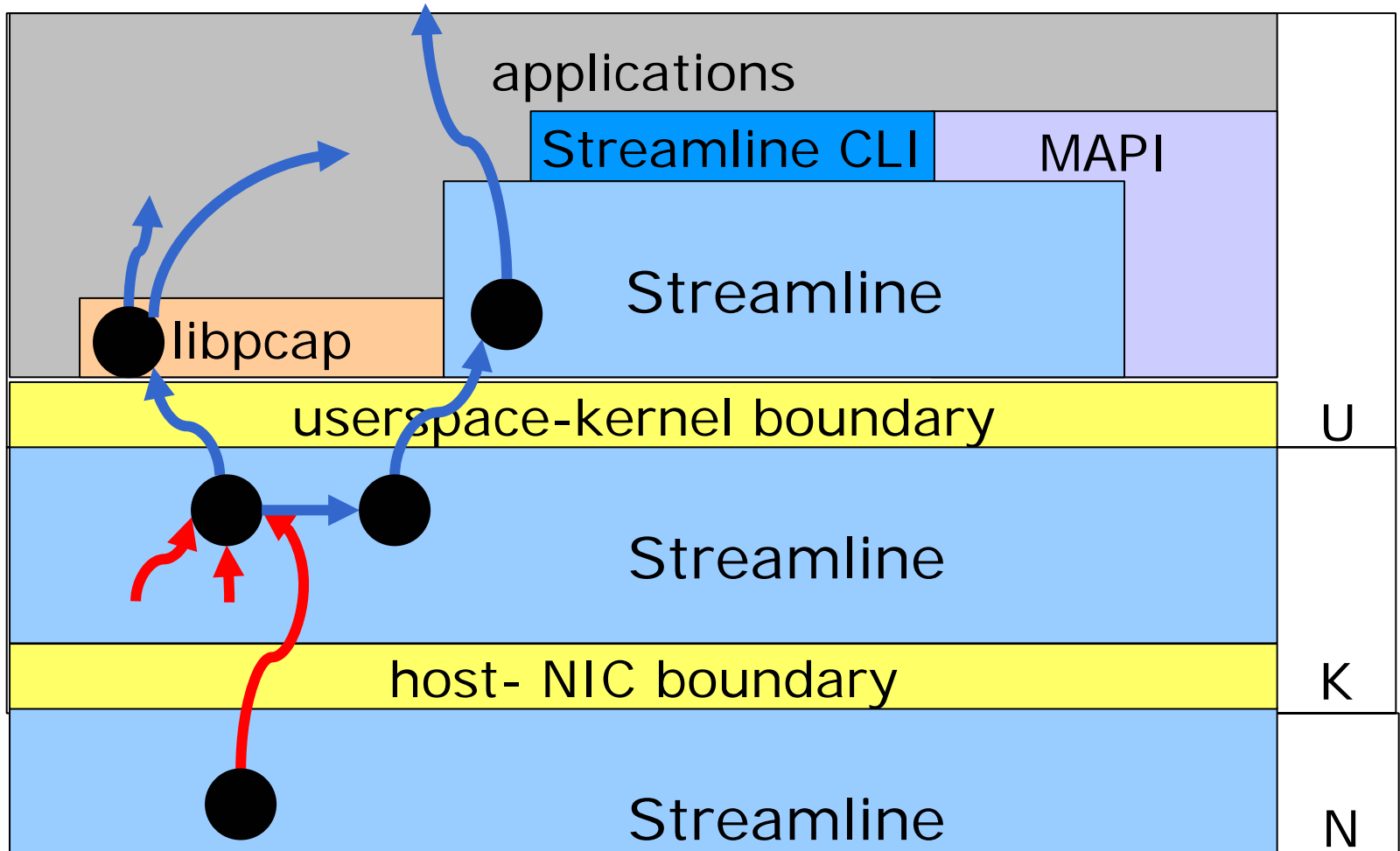


weave flows into a 'delta'



flow : “a stream of data that matches arbitrary user criteria”

Software structure





How to use it?

- we build complex applications out of simple components
- different types of use
 - administrators
 - CLI/GUI: click applications together
 - program filters/functions in high-level languages
 - programmers
 - write code against the API
 - write new functions to be used by Streamline



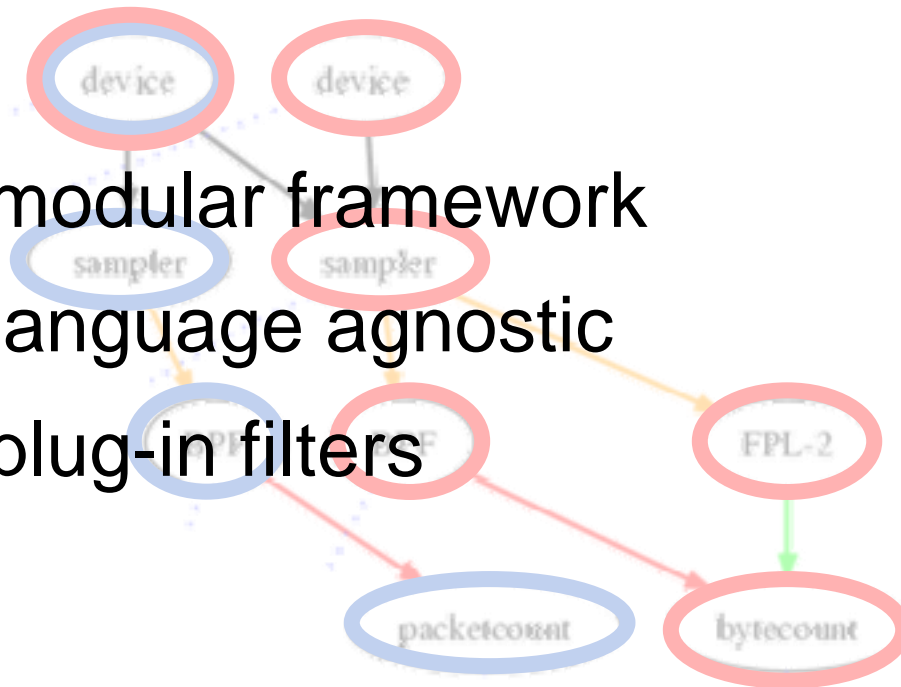
Part I: the administrator

Extensible

```
(device,expr=eth0) > (sampler, expr=2) > (BPF,expr="..") > (packetcount)
```

```
[(device,expr=eth0) | (device,expr=eth1)] > (sampler,expr=2) > [(FPL-2,expr="..") | (BPF,expr="..")] > (bytecount)
```

- ✓ modular framework
- ✓ language agnostic
- ✓ plug-in filters





CLI language

- flowgraph language

- boolean constructs

- **a & b**

- **a | b**

- **!a**

- stream:

- **a -> b**

- **a -80-> b**

- grouping:

- **a -> (b | c)**

Example expressions

- trivial:

(generator,expression="1:0") > (accept,export,name=mybuffer)

(pcapin,expression=infectedhttp.pcap) > (packetcount) >
(sampler,expression=50) > (packetcount)

- cleaning streams:

(pcapin,expression="infectedhttp.pcap") > (tcp) >
(tcpreassemble) > (ips_prospector) >
(outfiles,expression=cleantcp)

- classification:

(pcapin,expression="infectedhttp.pcap") > (sourceport)
-80-> (tcp) > (tcpreassemble) > (inspect)

- transcoding:

(ipv4_receive,expression="t 127.0.0.1 5050") > (rot13) >
(ipv4_transmit,expression="t 127.0.0.1 5051")

Example 1: GUI for creating flowgraphs

The screenshot shows a window titled "FFPPF Controller" with a menu bar containing "File", "Edit", and "Help". Below the menu bar is a section labeled "Filter Classes:" containing a list of filter types: Device, Accept, Aho-Corasick, BPF, Boyer-Moore, ByteCount, Drop, FPL-2, FPL-3, MemSearch, PacketCount, RegularExpression, Sampler, and StringSearch. A blue button labeled "Current: Device" is positioned above the main workspace. The workspace contains a flowgraph with two nodes: a white circle labeled "BPF" and a white circle labeled "Device". A red arrow points from the "Device" node to the "BPF" node. At the bottom of the workspace, a red text message reads: "Hovering over nodes will display configuration (more keybindings: see help menu)". In the bottom right corner, there is a logo for "Vrije Universiteit Amsterdam" featuring a stylized eagle.

Example 1: GUI for creating flowgraphs

The screenshot shows the 'FFPF Controller' application window. On the left, a list of 'Filter Classes' includes: Device, Accept, Aho-Corasick, BPF, Boyer-Moore, ByteCount, Drop, FPL-2, FPL-3, MemSearch, PacketCount, RegularExpression, Sampler, and StringSearch. A blue box above the main canvas indicates 'Current: Device'. The main canvas displays a flowgraph with four nodes: 'Device' at the bottom left, 'Device' at the bottom right, 'FPL-2' in the middle left, and 'PacketCount' at the top center. Red arrows point from the bottom 'Device' nodes to 'FPL-2' and 'BPF', and from 'FPL-2' and 'BPF' to 'PacketCount'. A red text box at the bottom of the window reads: 'Hovering over nodes will display configuration (more keybindings: see help menu)'. The bottom right corner features the logo of Vrije Universiteit Amsterdam.

Example 1: GUI for creating flowgraphs

The screenshot shows the 'FFPF Controller' application window. On the left, a list of 'Filter Classes' is displayed, including Device, Accept, Aho-Corasick, BPF, Boyer-Moore, ByteCount, Drop, FPL-2, FPL-3, MemSearch, PacketCount, RegularExpression, Sampler, and StringSearch. A blue box above the main canvas indicates 'Current: Device'. The main canvas contains a flowgraph with nodes labeled 'Device', 'FPL-2', 'PacketCount', 'BPF', 'FPL-2', and 'ByteCount', connected by red arrows. A red text box at the bottom states: 'Hovering over nodes will display configuration (more keybindings: see help menu)'. The Vrije Universiteit Amsterdam logo is in the bottom right corner.

FFPF Controller

File Edit Help

Filter Classes:

- Device
- Accept
- Aho-Corasick
- BPF
- Boyer-Moore
- ByteCount
- Drop
- FPL-2
- FPL-3
- MemSearch
- PacketCount
- RegularExpression
- Sampler
- StringSearch

Current: Device

ByteCount

PacketCount

FPL-2

BPF

FPL-2

Device

Device

Hovering over nodes will display configuration (more keybindings: see help menu)

Vrije Universiteit Amsterdam

Example 1: GUI for creating flowgraphs

FFPF Controller

File Edit Help

Filter Classes:

- Device
- Accept
- Aho-Corasick
- BPF
- Boyer-Moore
- ByteCount
- Drop
- FPL-2
- FPL-3
- MemSearch
- PacketCount
- RegularExpression
- Sampler
- StringSearch

Current: Device

PacketCount

Aho-Corasick

ByteCount

PacketCount

FPL-2

FPL-2

BPF

Device

Device

Hovering over nodes will display configuration (more keybindings: see help menu)

Vrije Universiteit Amsterdam



mapping on hardware

Streamline is responsible for mapping the flowgraph on the underlying hardware

Copying

Three flavours of packet processing

- **Regular**
 - copy only when needed
 - may be slow depending on access pattern
- **Zero copy**
 - *never* copy
 - may be slow depending on access pattern
- **Copy once**
 - copy always



Part II: the savvy administrator

Example 2: FPL-3

- new pkt processing language: FPL-2

- for IXP, kernel and userspace
- simple, efficient and flexible
- simple example: filter all webtraffic

```
IF ( (PKT.IP_PROTO == PROTO_TCP)
      && (PKT.TCP_PORT == 80)) THEN RETURN 1;
```

- more complex example: count pkts in all TCP flows

```
IF (PKT.IP_PROTO == PROTO_TCP) THEN
  R[0] = Hash[ 14, 12, 1024];
  M[ R[0] ]++;
FI
```

FPL-3

- all common arithmetic and bitwise operations
- all common logical ops
- all common integer types
 - for packet
 - for buffer (← useful for keeping state!)
- statements
 - Hash
 - External functions
 - to call hardware implementations
 - to call fast C implementations
 - If ... then ... else
 - For ... break; ... Rof
 - Return

Example application: dynamic ports

```
1. // R[0] stores no. of dynports found (initially 0)
2. IF (PKT.IP_PROTO==PROTO_TCP) THEN
3.   IF (PKT.TCP_DPORT==554) THEN
4.     M[R[0]]=EXTERN("GetDynTCPDPortFromRTSP",0,0);
5.     R[0]++;

6.   ELSE // compare pkt's dst port to all ports in array – if match, return pkt
7.     FOR (R[1]=0; R[1] < R[0]; R[1]++)
8.       IF (PKT.TCP_DPORT == M[ R[1] ]) THEN
9.         RETURN TRUE;
10.      FI
11.    ROF
11.  FI
12. FI
12. RETURN FALSE;
```



Other example: Ruler

- regular expression matching and rewriting language
 - ➔ see the Lobster workshop tomorrow



Part III: the programmer

trivial example

```
1. int main (int argc, char** argv){
2.     int fd, bd, len;
3.     char *pkt;
4.
5.     // open FFPF and start processing
6.     streamline_init();
7.     fd=streamline_request_insert("(generator,expression=\"1:0\") > (packetcount, export, name=mybuf)");
8.
9.     // access the data
10.    bd = open_stream("mybuf", O_RDONLY | OF_INDIRECT); // extended, but posix compatible API
11.    while (1) {
12.        sleep(1);
13.        len = read_stream(bd, &pkt, 1500); // direct data access
14.        printf("read a packet of %d bytes\n", len);
15.    }
16.    close_stream(bd);
17.
18.    // close FFPF
19.    streamline_request_remove(fd);
20.    streamline_exit();
21.    return 0;
22. }
```



Concluding remarks

- achieved: fast flexible packet processing
- minimising copying/context switching
- cater to different users
- exploit advanced hardware
- new languages

more information

<http://ffpf.sourceforge.net/>



The screenshot shows a Konqueror browser window titled "Fairly Fast Packet Filter - overview - Konqueror". The address bar contains the URL "http://ffpf.sourceforge.net/general/overview.php". The page content is as follows:

<h1>FFPF</h1>	<h2>>> overview</h2>	
general <ul style="list-style-type: none">overviewnewsdownloadswebcvsproject summarypeople	Introduction <p>The fairly fast packet filter (FFPF) is an approach to network packet processing that adds many new features to existing filtering solutions like BPF. FFPPF is designed for high speed by pushing computation to the hardware.</p>	News <p>FFPF 1.3 developer reference documentation online <i>wdebruij - 2004-07-18 10:19</i></p> <p>[Read]</p>
science <ul style="list-style-type: none">publications		