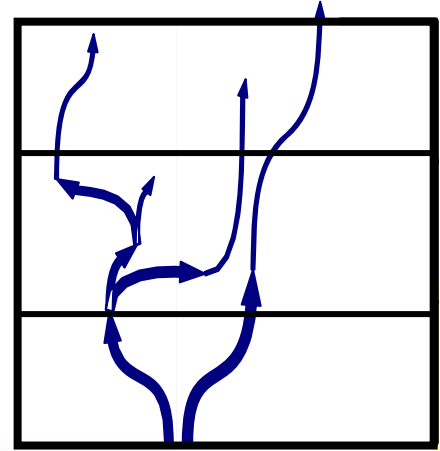


# Streamline

fast and flexible traffic processing



Willem de Bruijn

Herbert Bos

*Vrije Universiteit Amsterdam*



# Goals

- Major:
  - faster, more flexible traffic processing
- Minor:
  - new networking subsystem
  - operate in different space(s)



# 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 our weapons?

to reach these goals despite difficulties

reduce communication

parallelism

specialised hardware

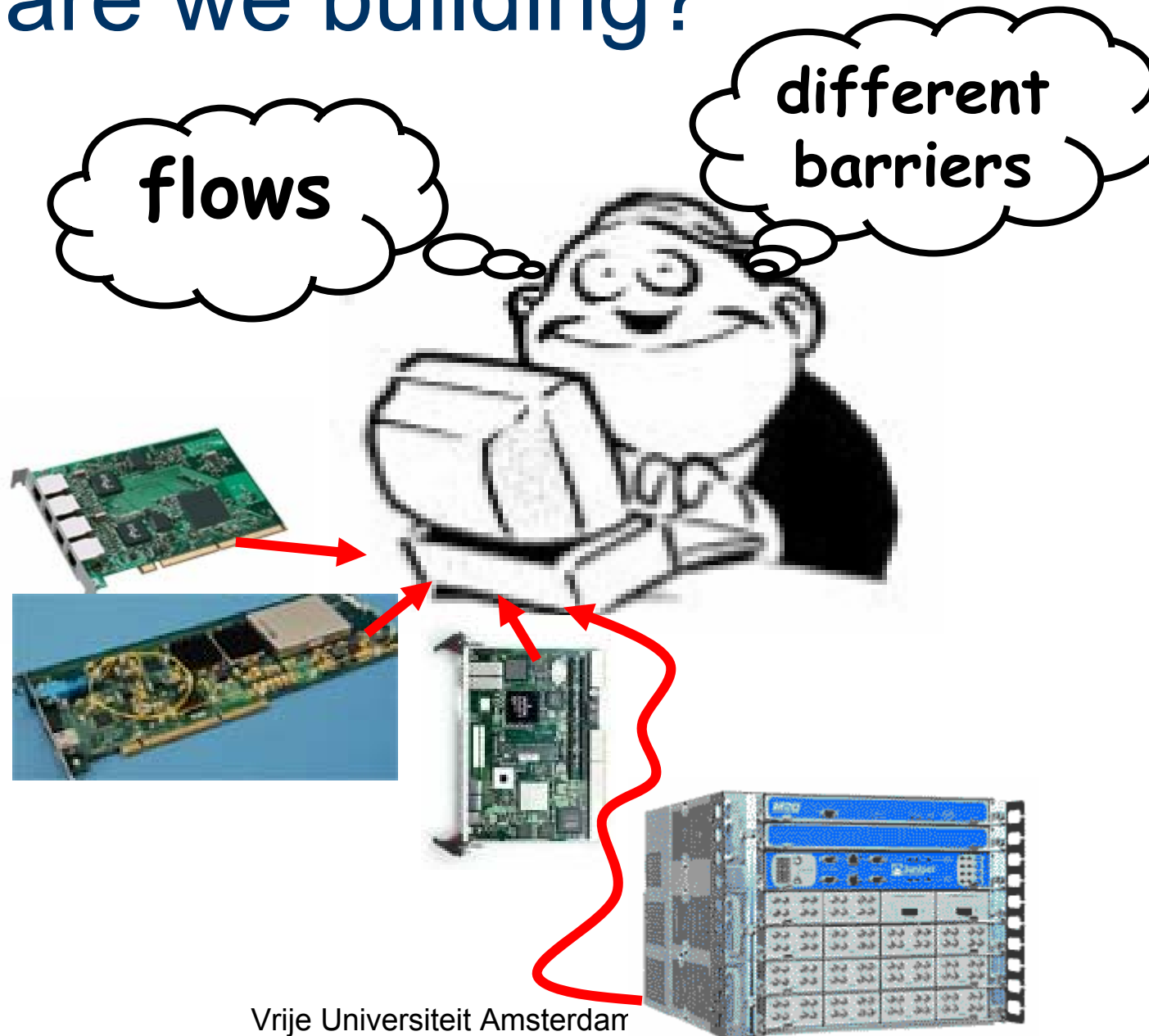
copy avoidance

reduction of miss

fewer context switches



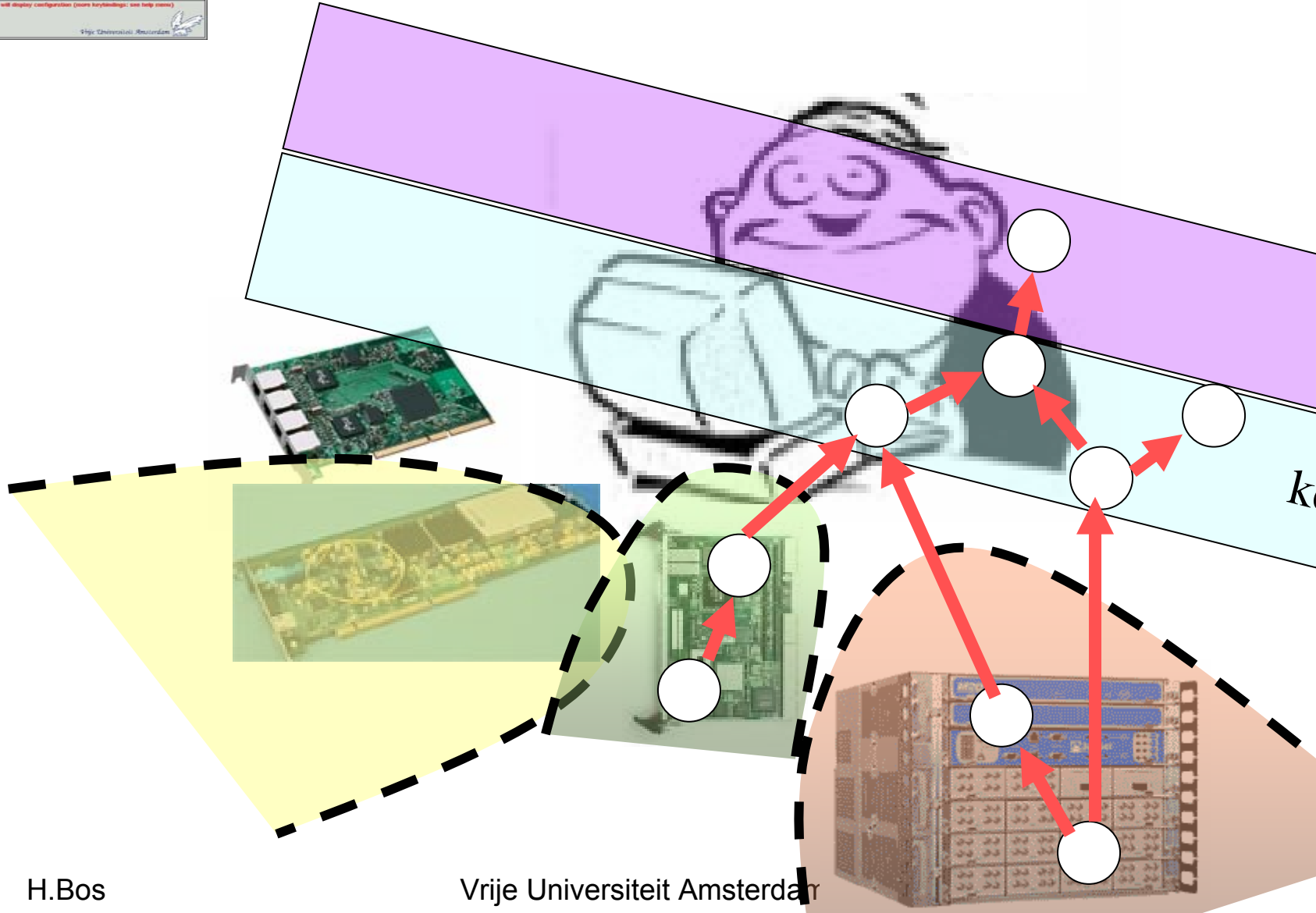
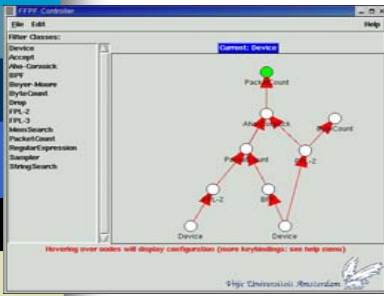
# What are we building?



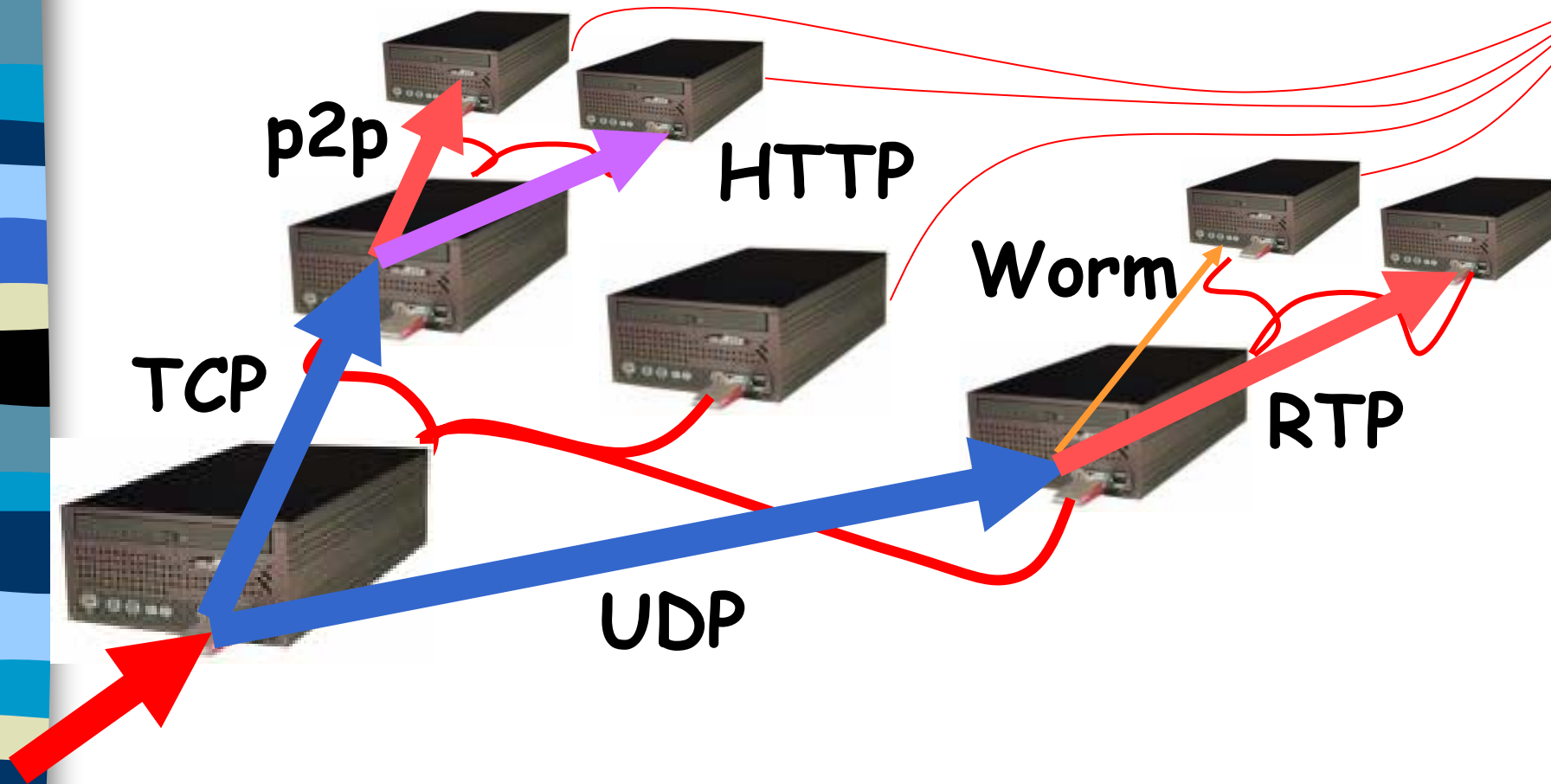
# What are we building?

The screenshot displays the 'FFPP Controller' application window. On the left, a list of 'Filter Classes' is shown, including Device, Accept, Aho-Corasick, BPF, Boyer-Moore, ByteCount, Drop, FPL-2, FPL-3, MemSearch, PacketCount, RegularExpression, Sampler, and StringSearch. The main area shows a hierarchical tree diagram with nodes connected by red arrows. The root node is 'PacketCount' (green circle). Below it is 'Aho-Corasick' (white circle). From 'Aho-Corasick', two arrows point to 'PacketCount' and 'ByteCount' (white circles). The 'PacketCount' node has two arrows pointing to 'FPL-2' and 'BPF' (white circles). The 'FPL-2' node has two arrows pointing to 'Device' (white circles). The 'BPF' node has one arrow pointing to 'Device' (white circle). A blue box at the top of the diagram area says 'Current: Device'. At the bottom, a red text box reads: 'Hovering over nodes will display configuration (more keybindings: see help menu)'. The footer includes the text 'Vrije Universiteit Amsterdam' and a logo of a bird.

# are we building?

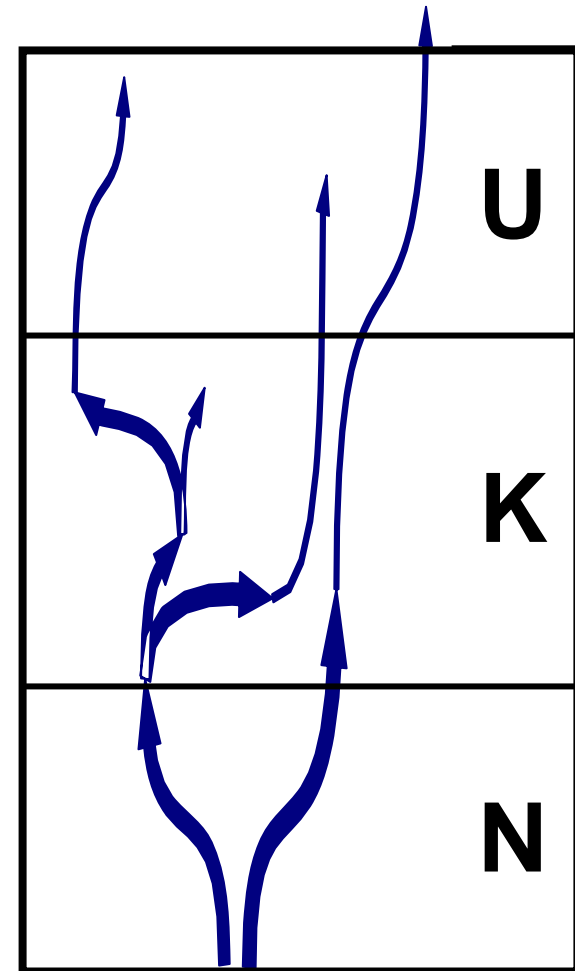


# What are we building?



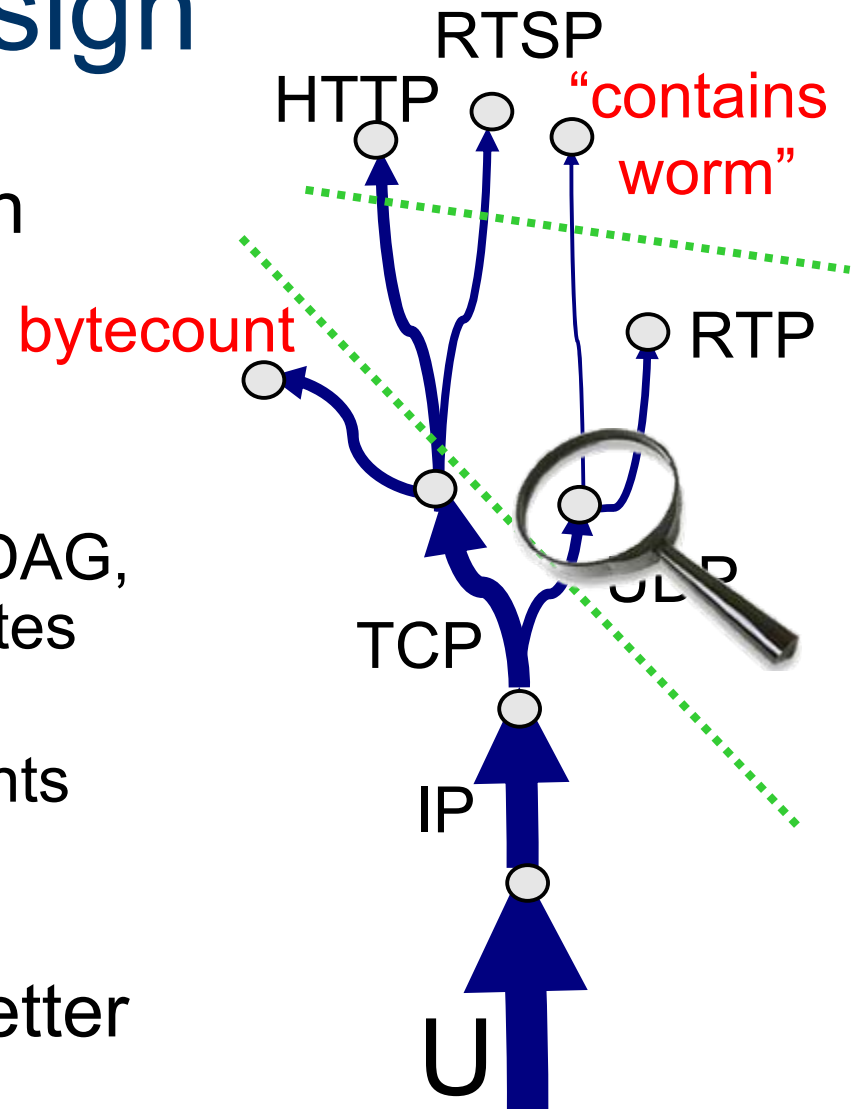
# Streamline design

- 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**'

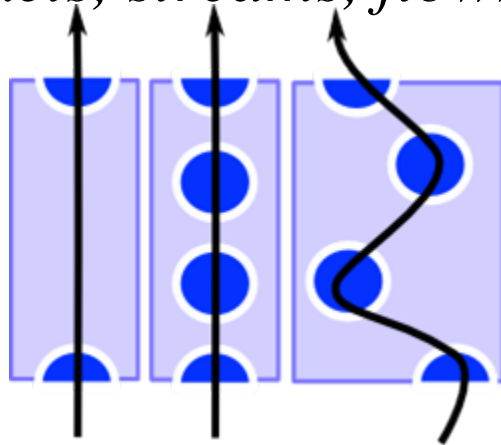


# Streamline design

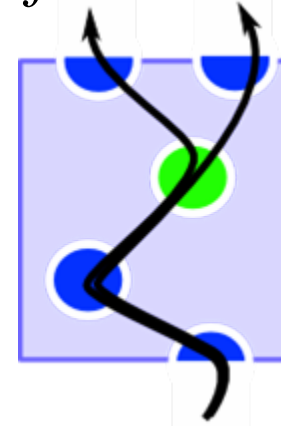
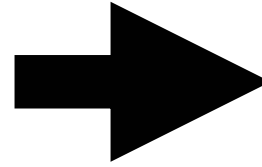
- specify processing in terms of a DAG
- Streamline handles instantiation
  - for each function in DAG, Streamline investigates potential placements
  - collective requirements determine definite placement
- Heuristic: lower is better



*sockets, streams, flows*

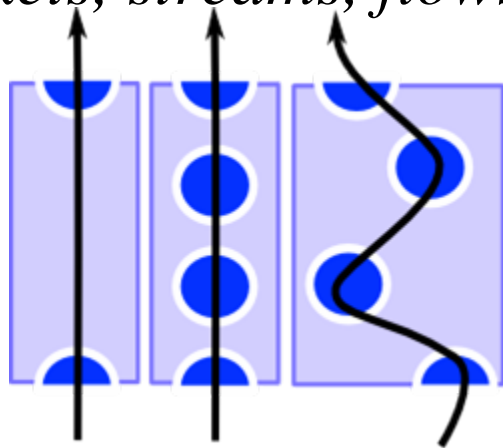


*weave flows into a 'delta'*

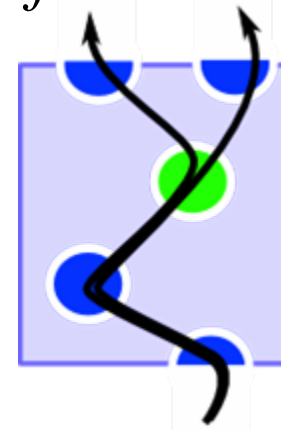
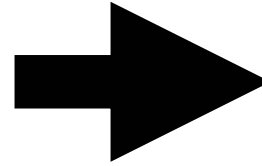


***flow : “a stream of data that matches arbitrary user criteria”***

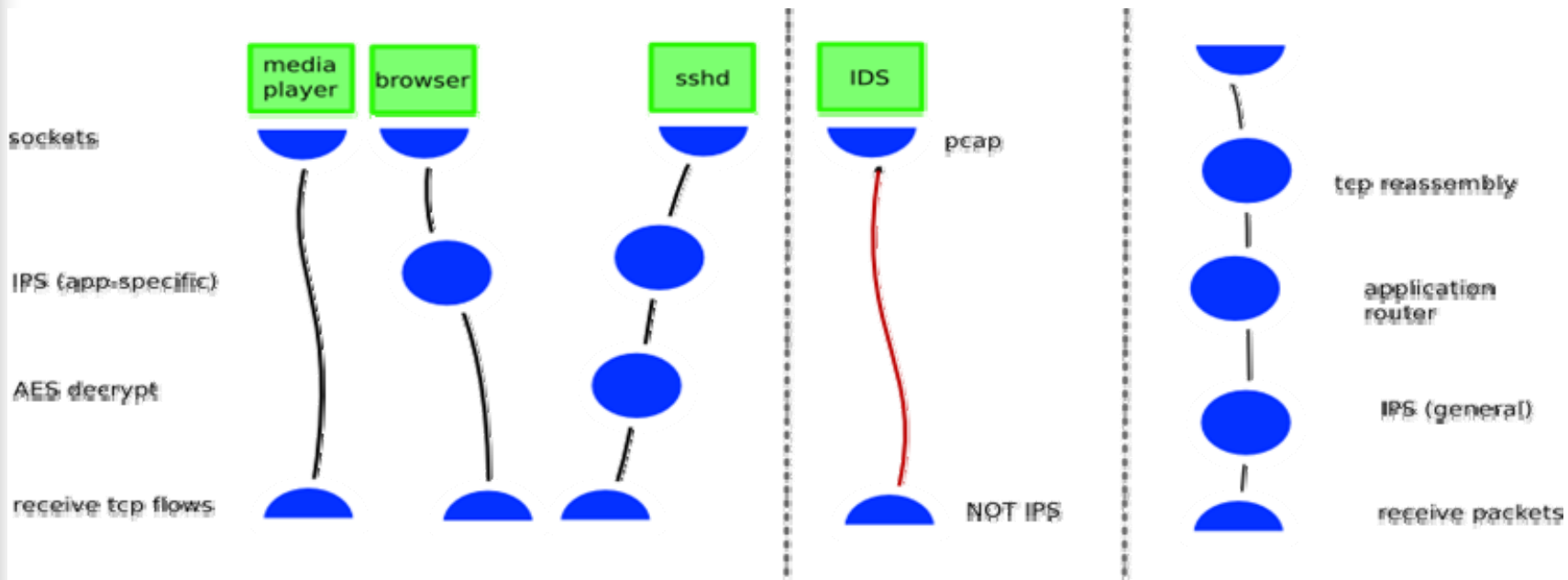
*sockets, streams, flows*



*weave flows into a 'delta'*



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



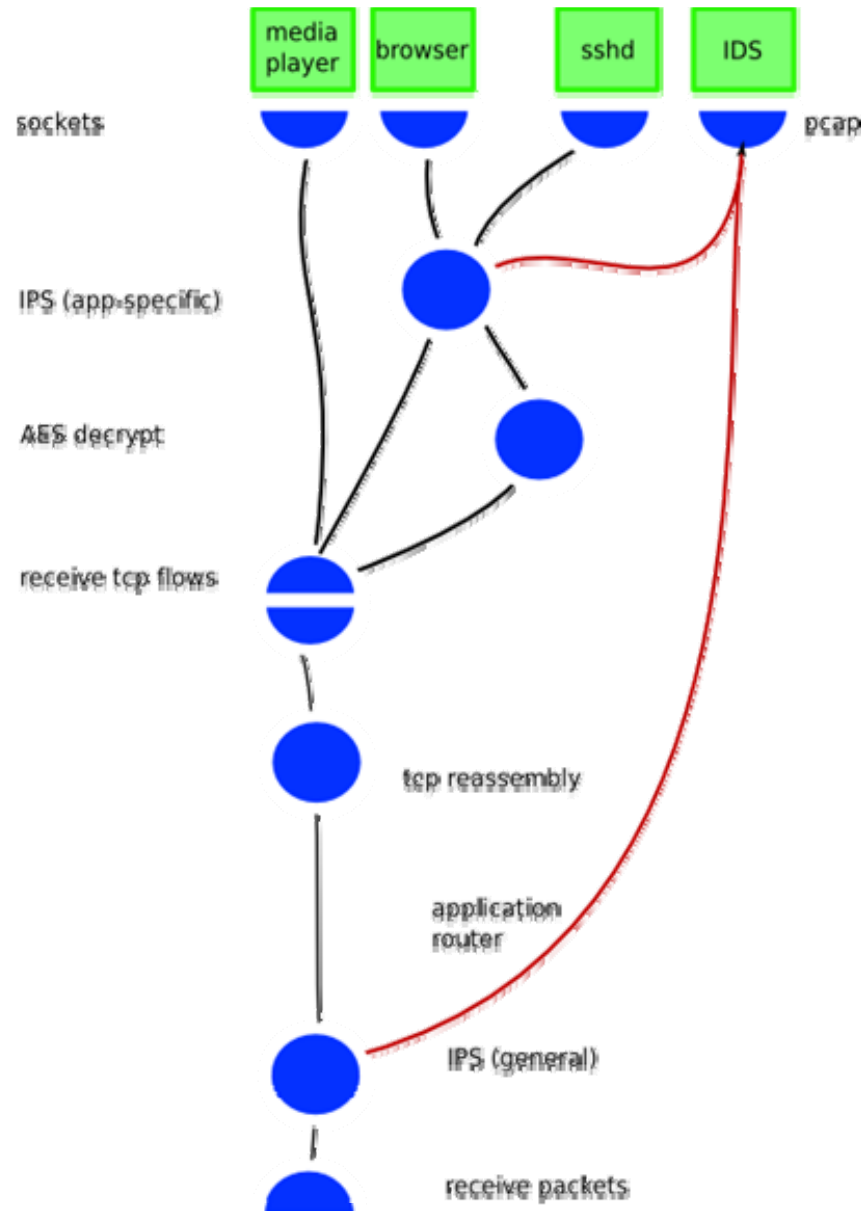
# optimization: merge requests

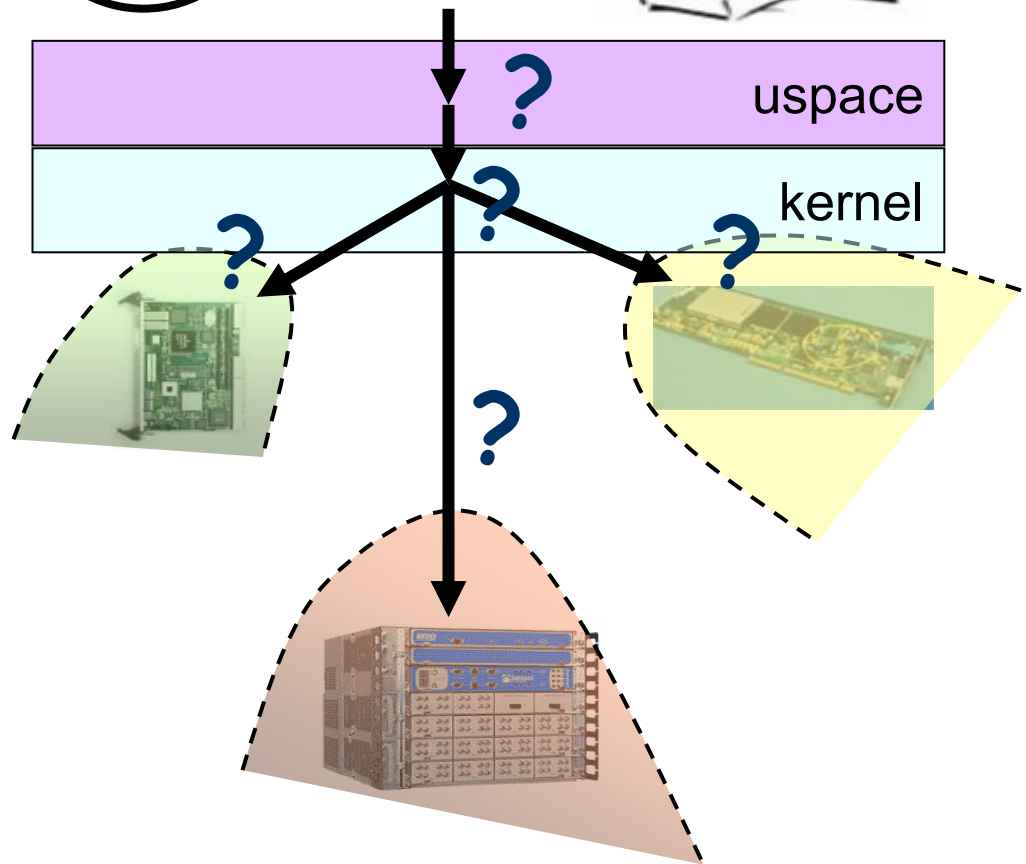
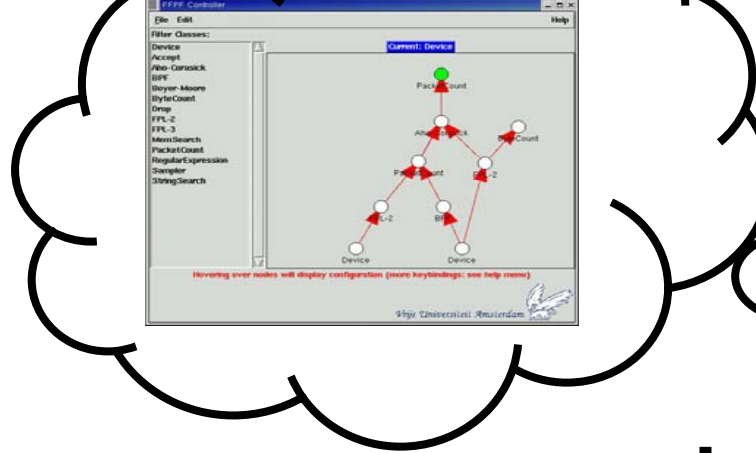
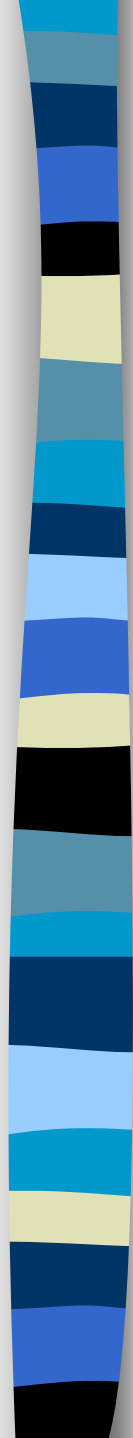
- reuse optimized functions\*
- minimize copy costs
- minimize processing costs

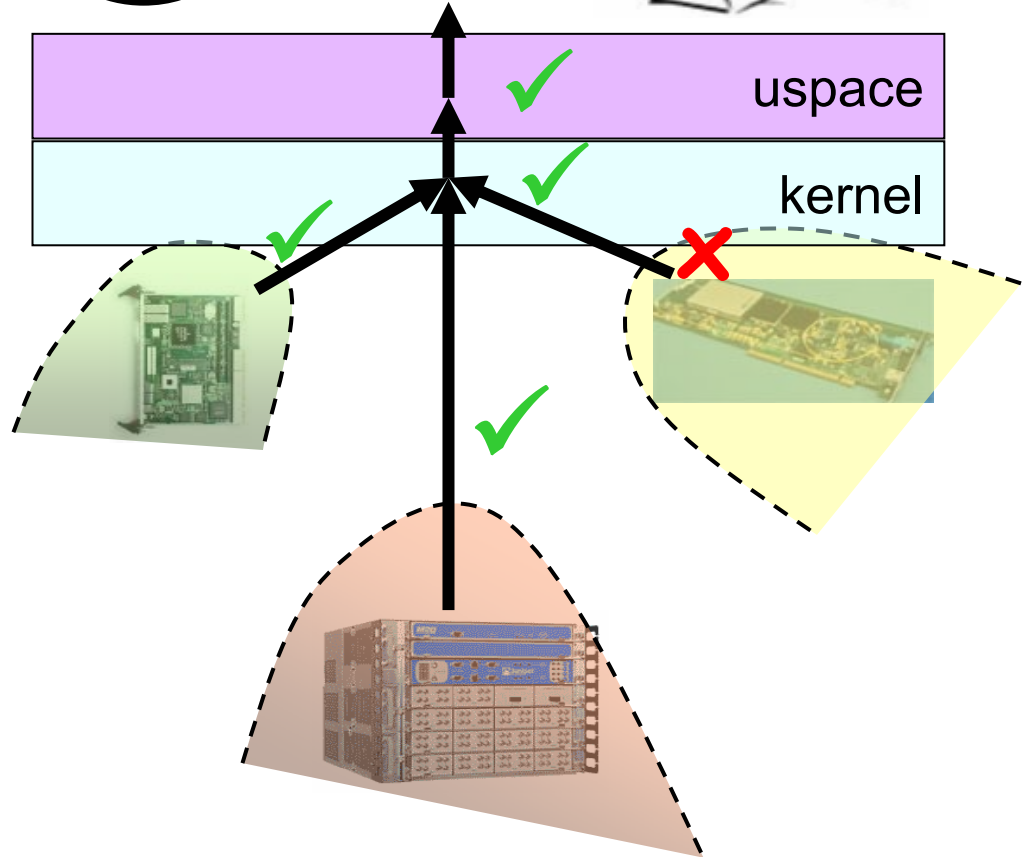
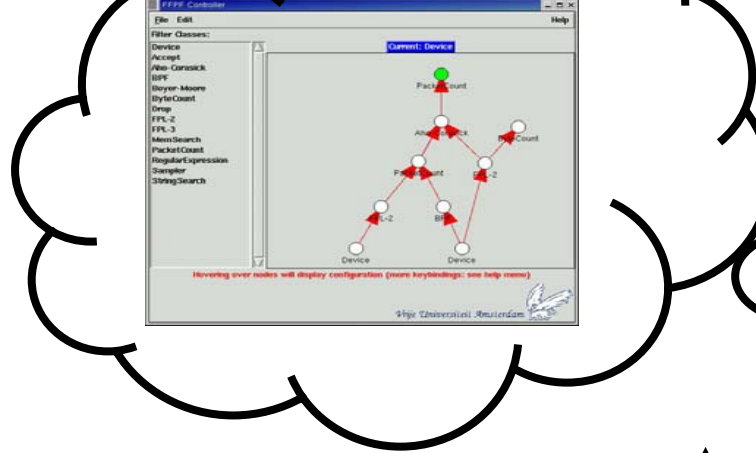
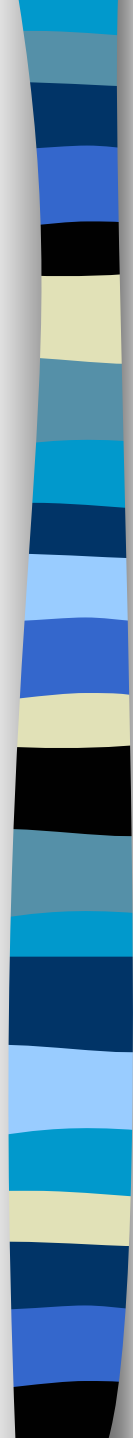
```
(tcpreassemble, port=80) >  
(aes_decrypt, key=123) >  
(ahocorasick, expr="slammer") >  
(socket_rx)
```

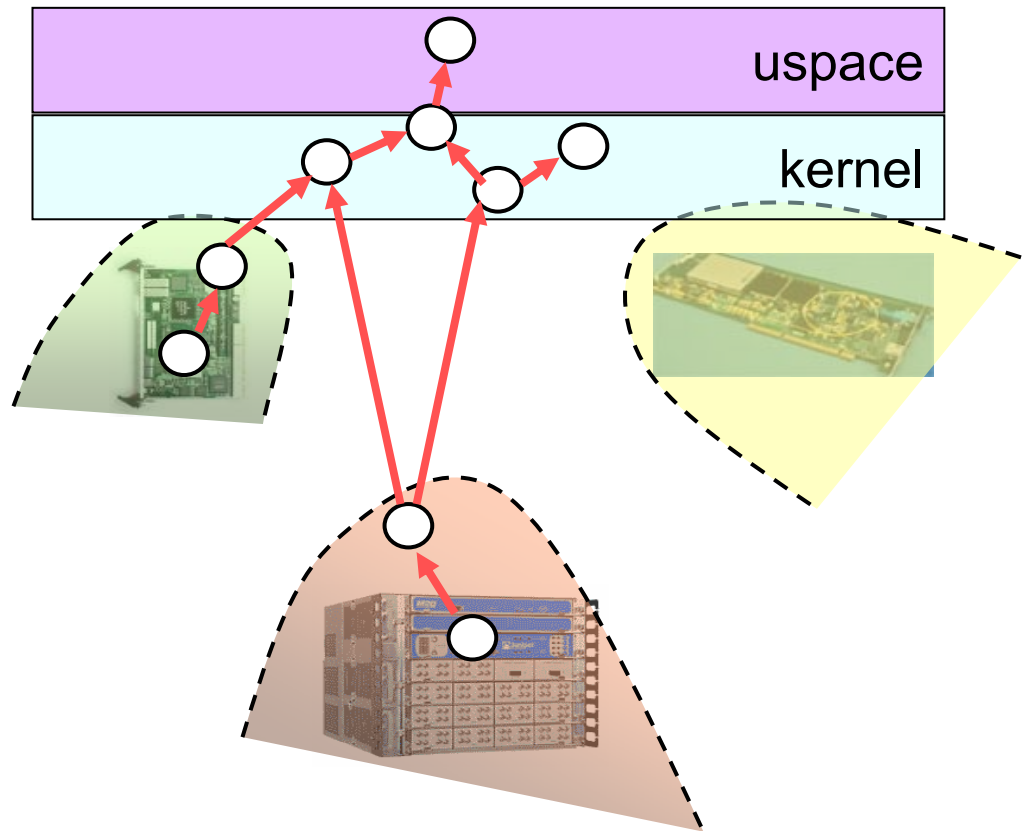
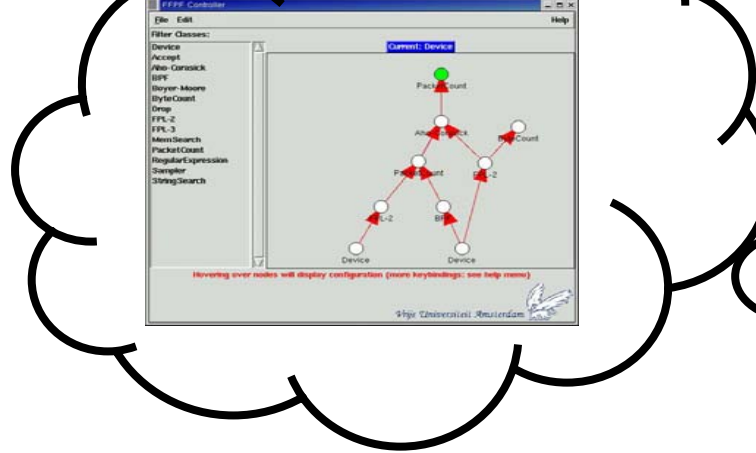
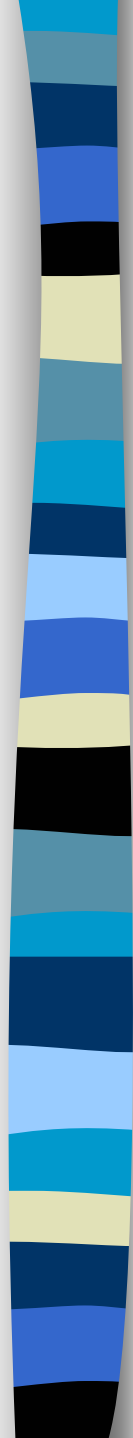
```
(a >!b) | (c & (d > e)) > f
```

\*framework is function agnostic









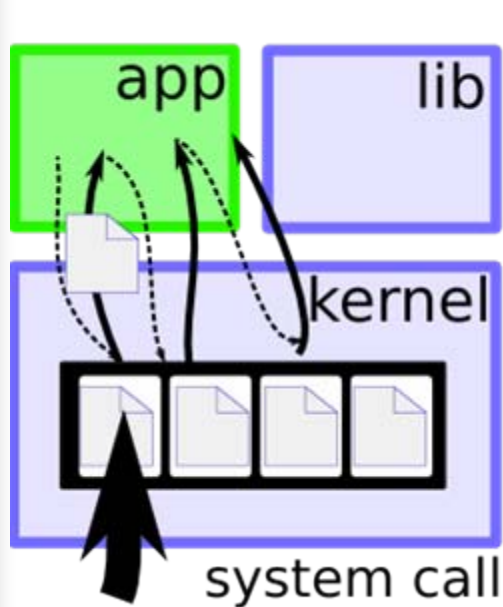


# Signals

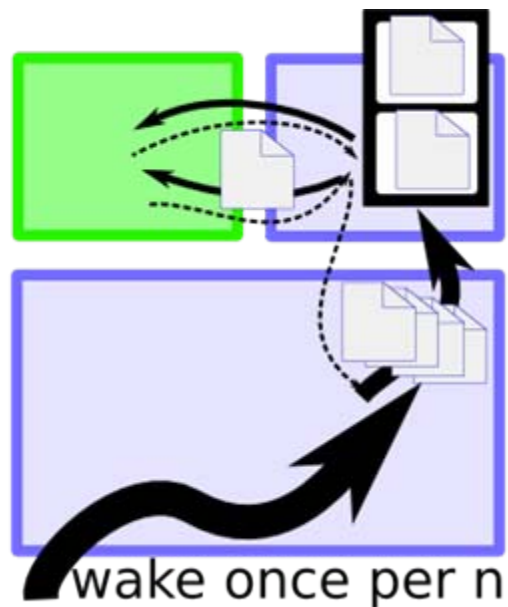
minimize #signals

# application communication

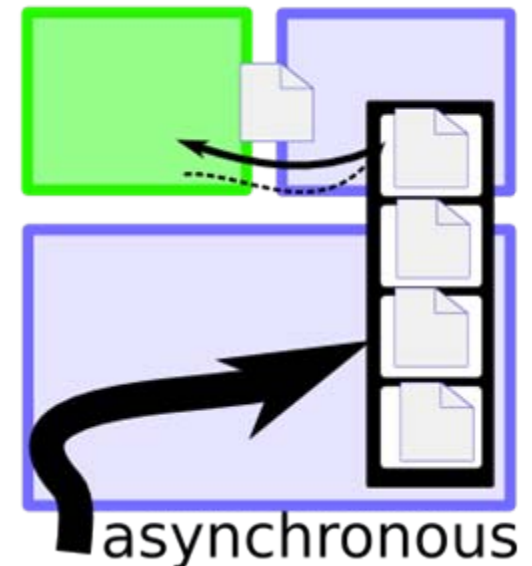
- push processing down
- aggregate signals
- process asynchronously



H.Bos



Vrije Universiteit Amsterdam

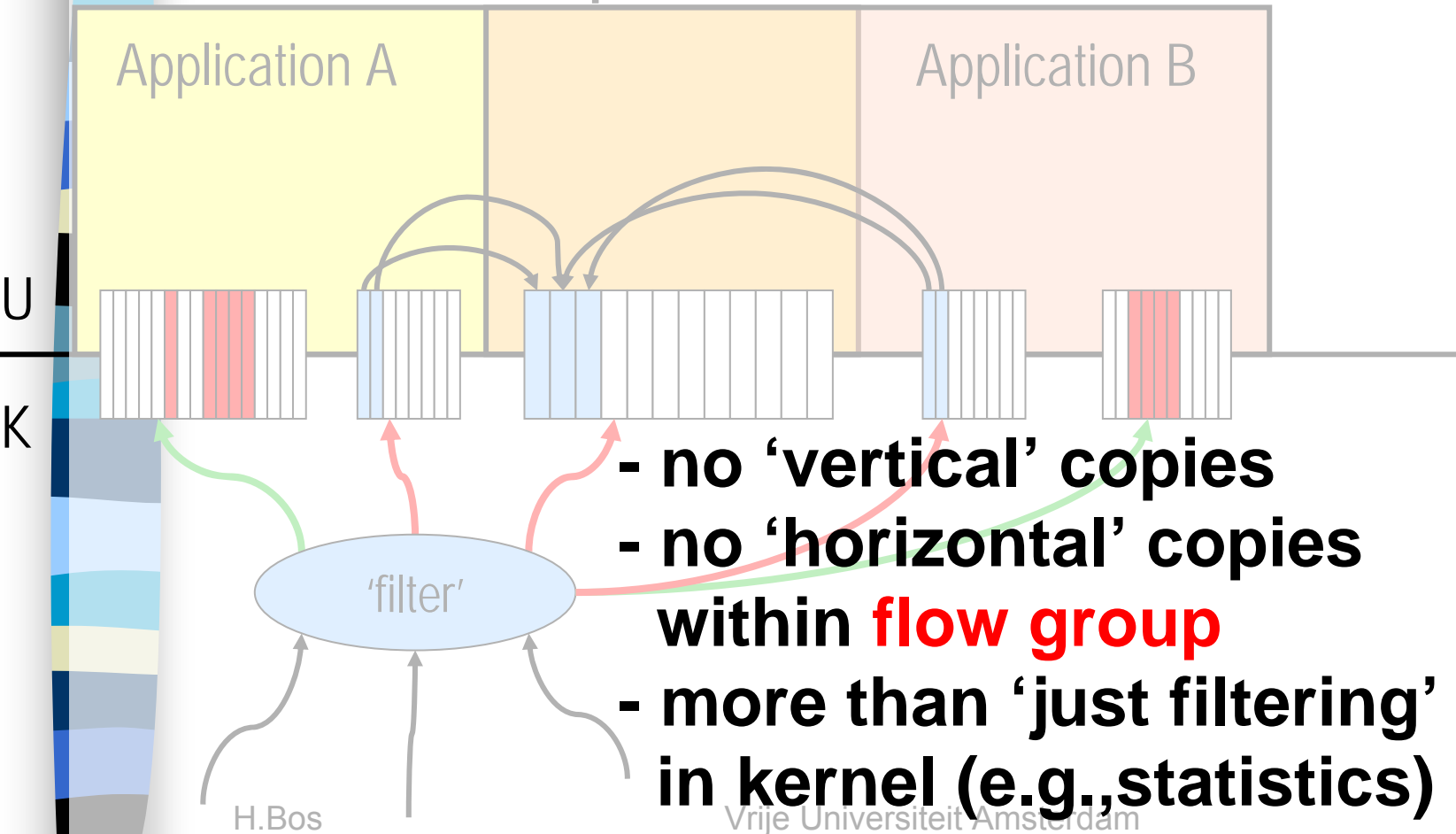




# Buffers

# reduce copying

- Streamline avoids both 'horizontal' and 'vertical' copies



minimize #copies, #cache-misses

# internal transport mechanisms

multiple strategies

zero copy

copy once (immediate, on-demand)

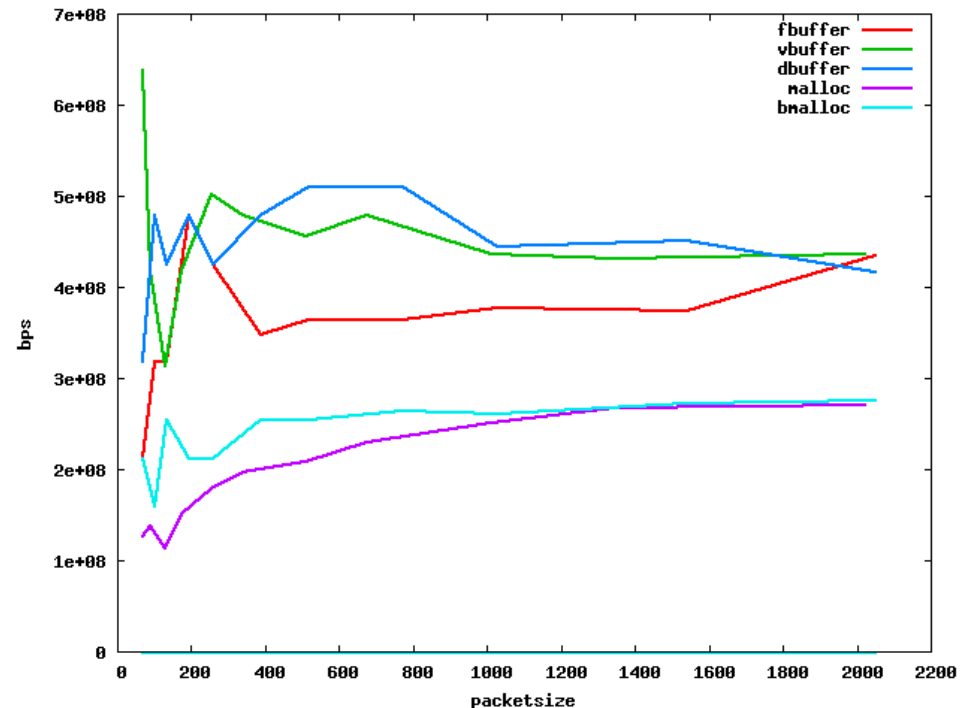
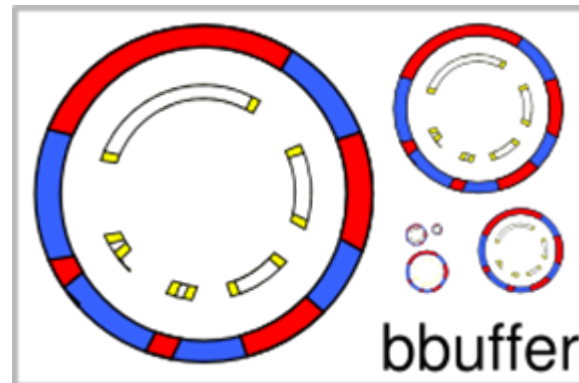
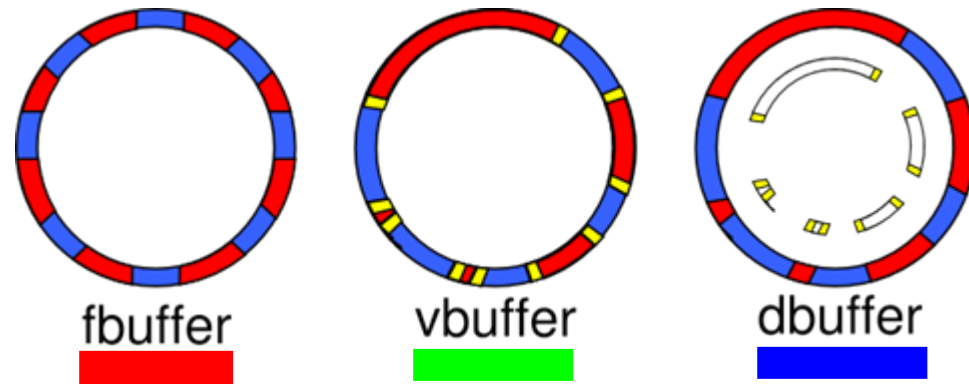
copy-on-write

copy-on-read

multiple data types

discrete chunks

continuous streams



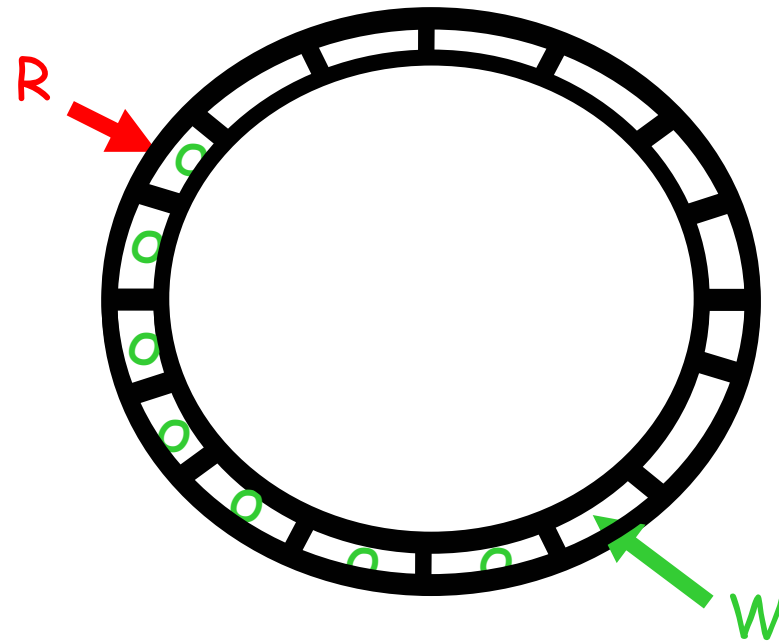
# Buffers

## ■ PacketBuf

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

## ■ IndexBuf

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



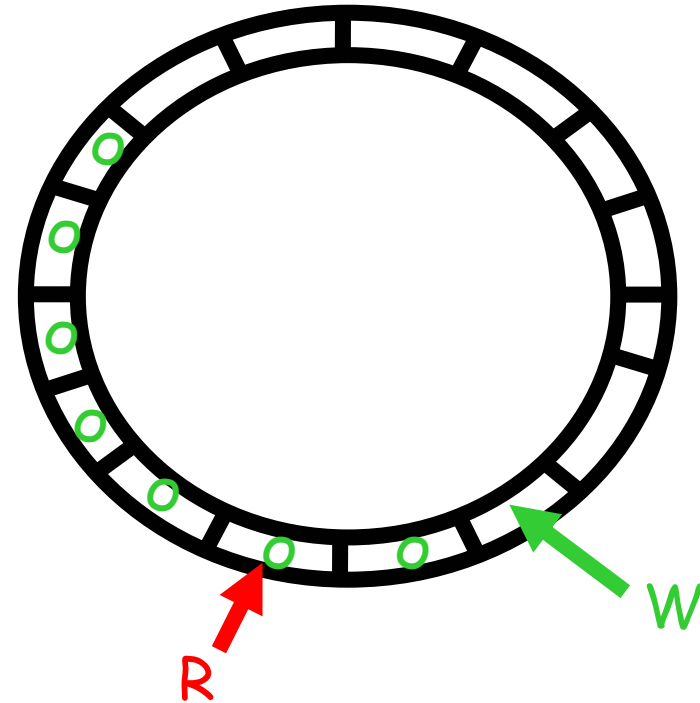
# Buffers

## ■ PacketBuf

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

## ■ IndexBuf

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



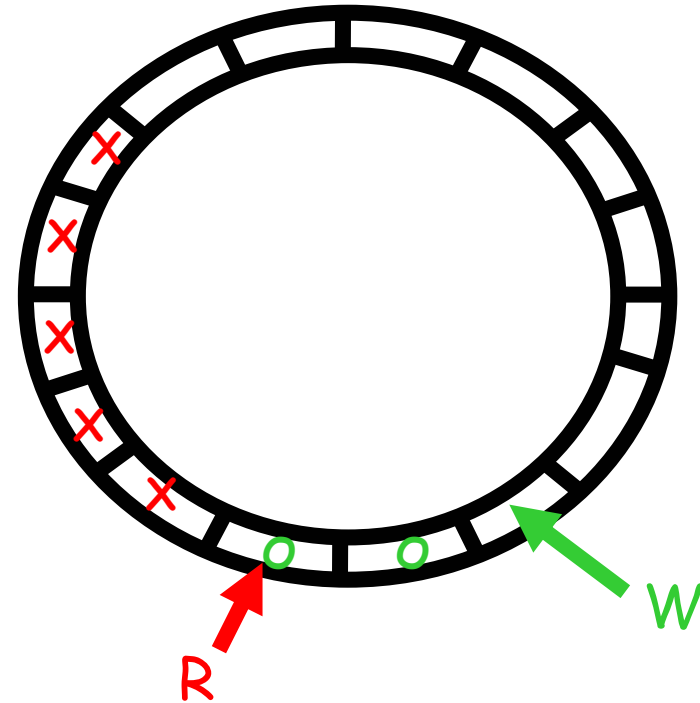
# Buffers

## ■ 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

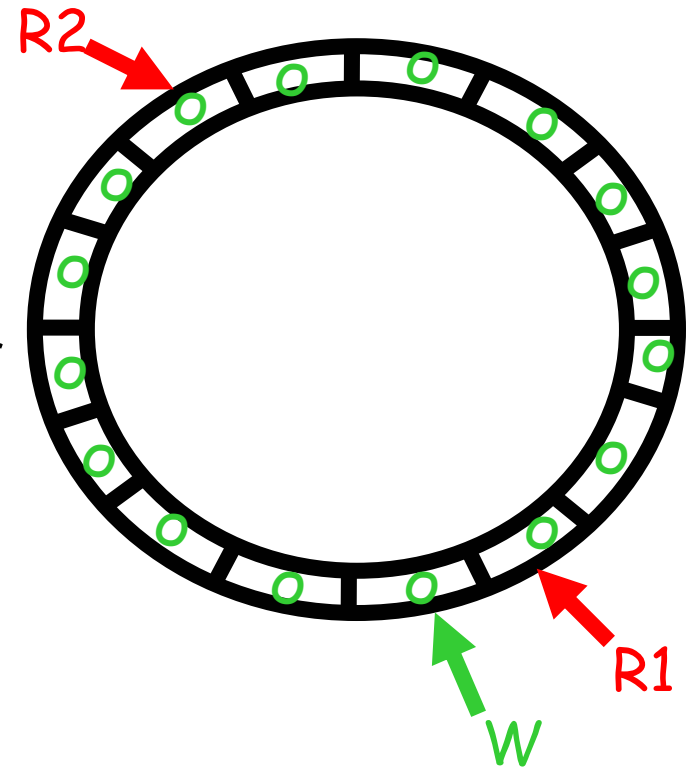
→ what to do if writer catches up with slowest reader?

## ■ 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





# hardware

minimize #copies, #cache-misses

# integrate hardware assistance

Network Processors: "NIC with logic"

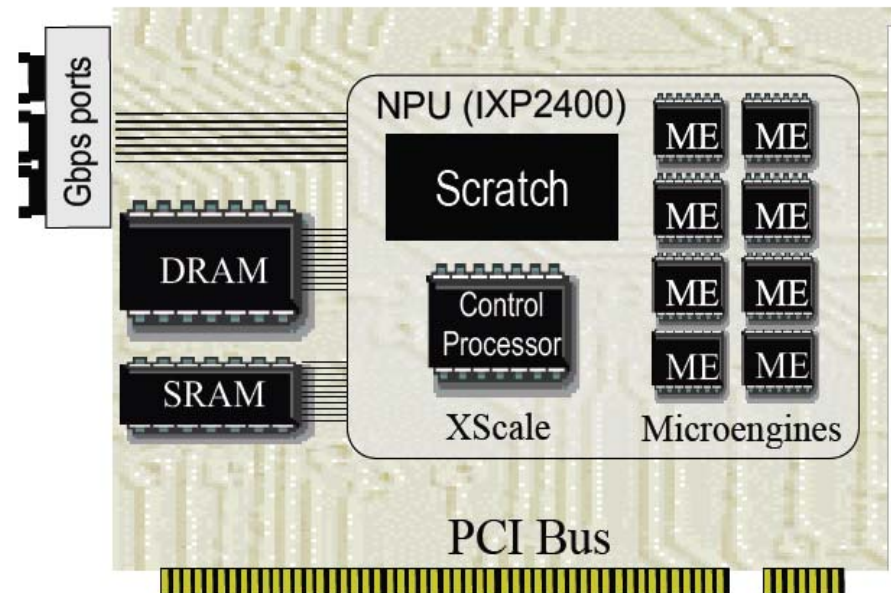
bottom of the processing hierarchy

eliminates mem & bus bottlenecks

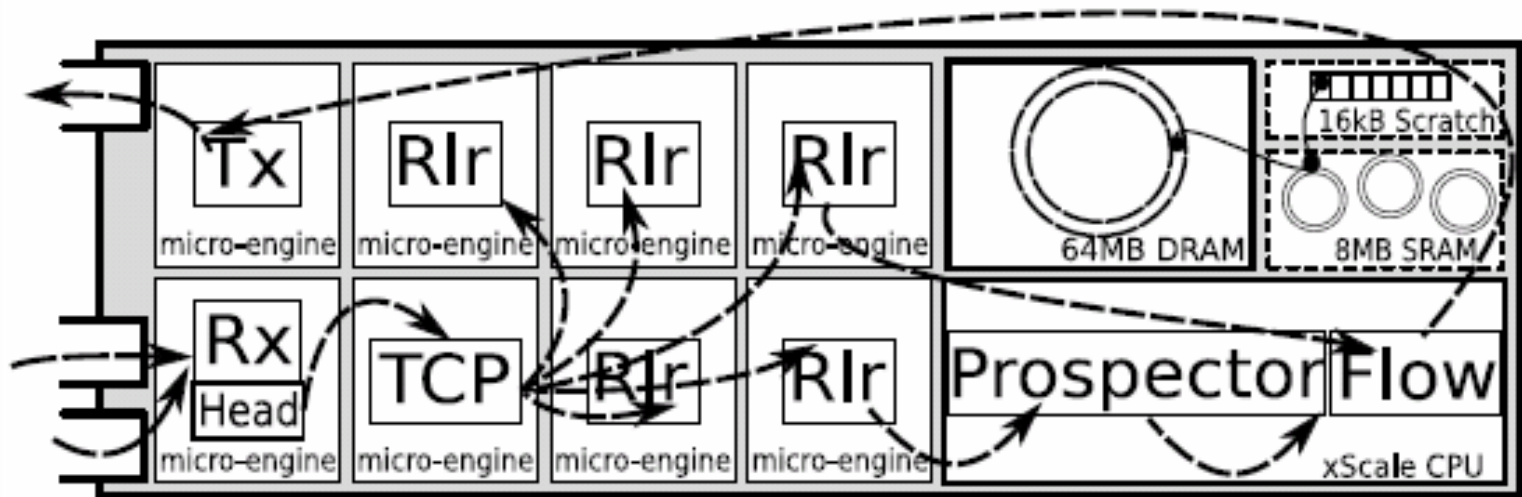
- FPL: automatically generated code
- Ruler: anonymization

Future work:

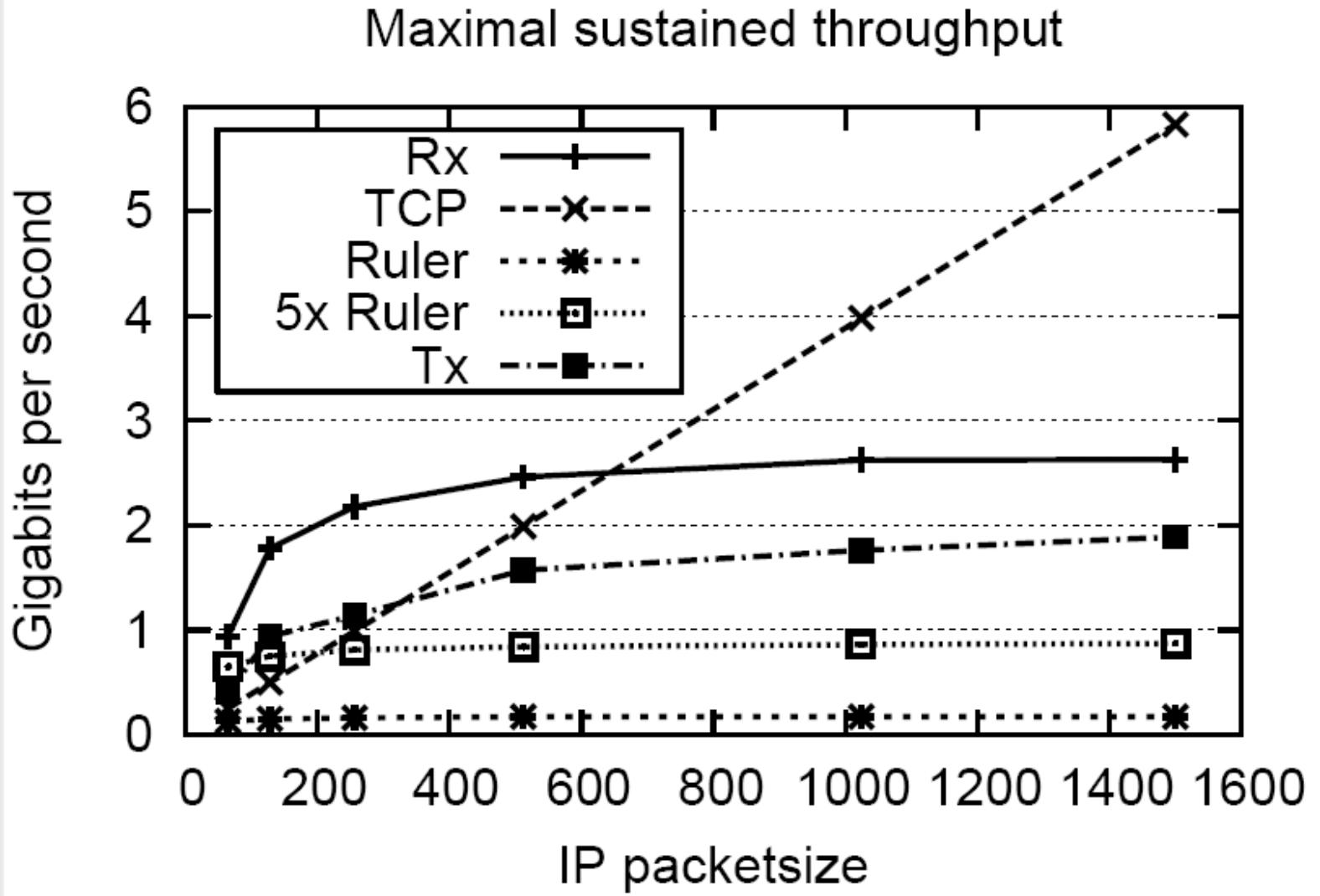
FPGA, ToE, GPU, ...



# Example application: IPS on the network card



# Results



# more information

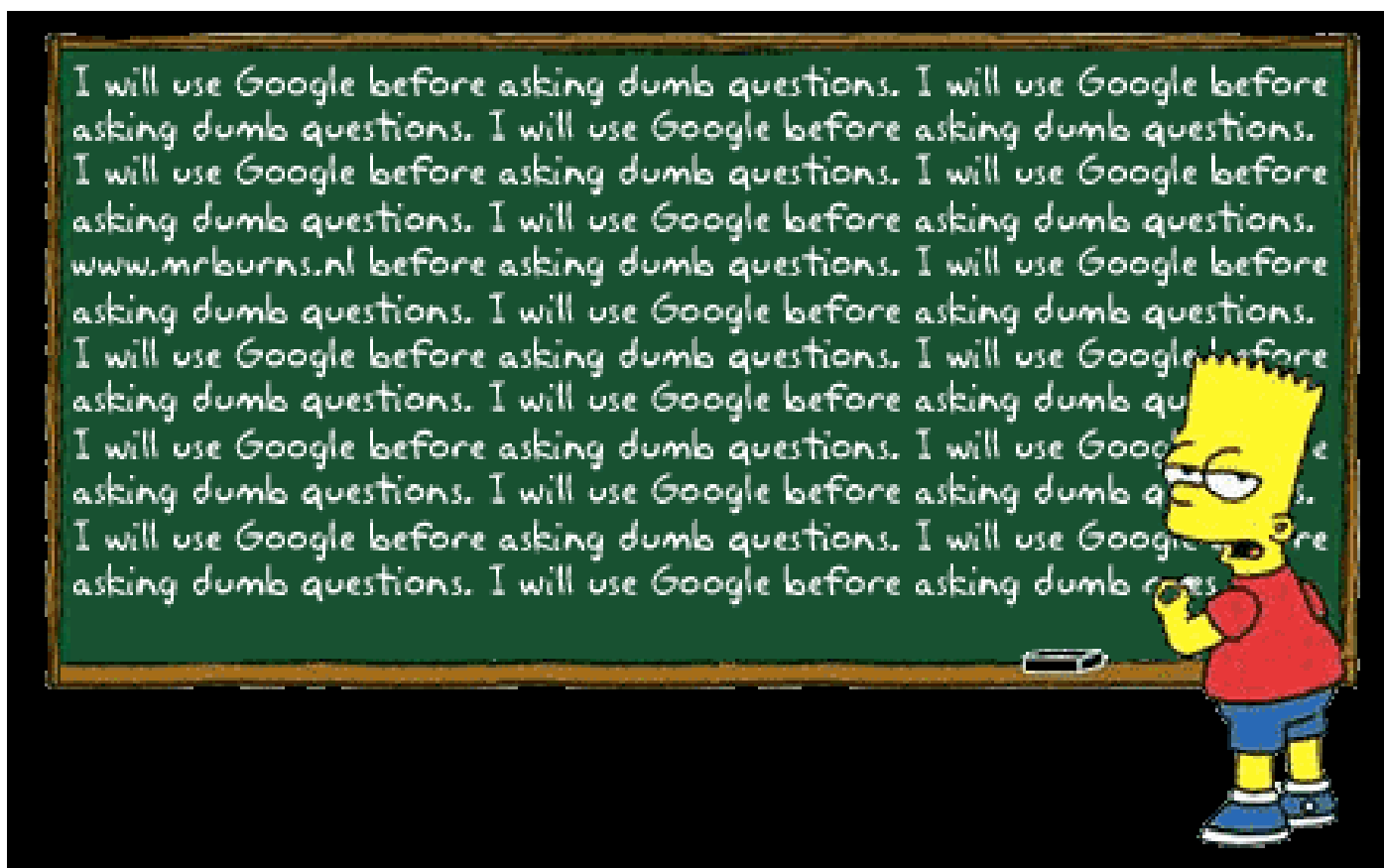
<http://ffpf.sourceforge.net/>



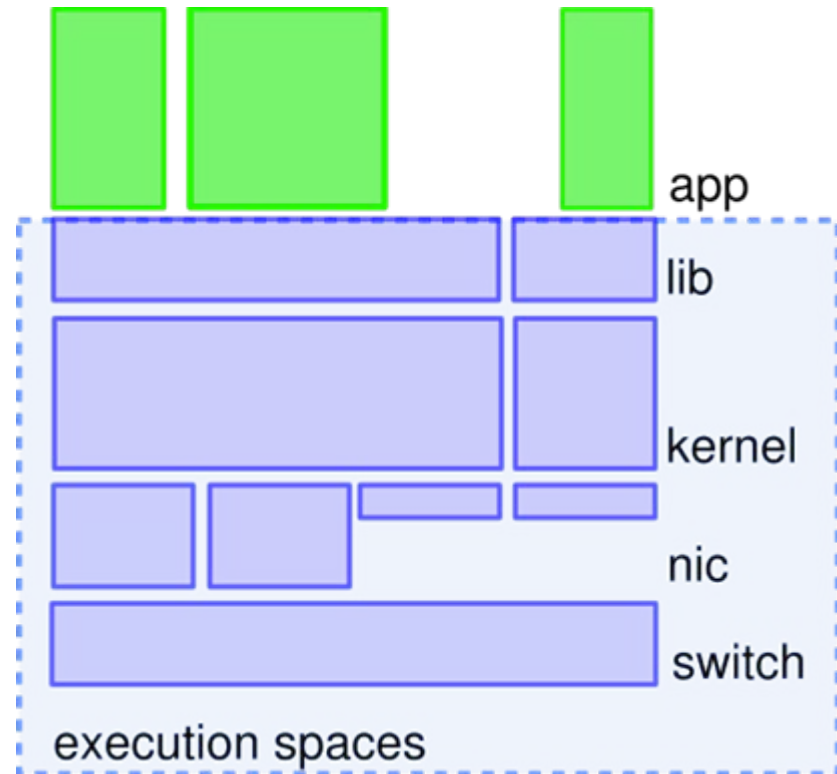
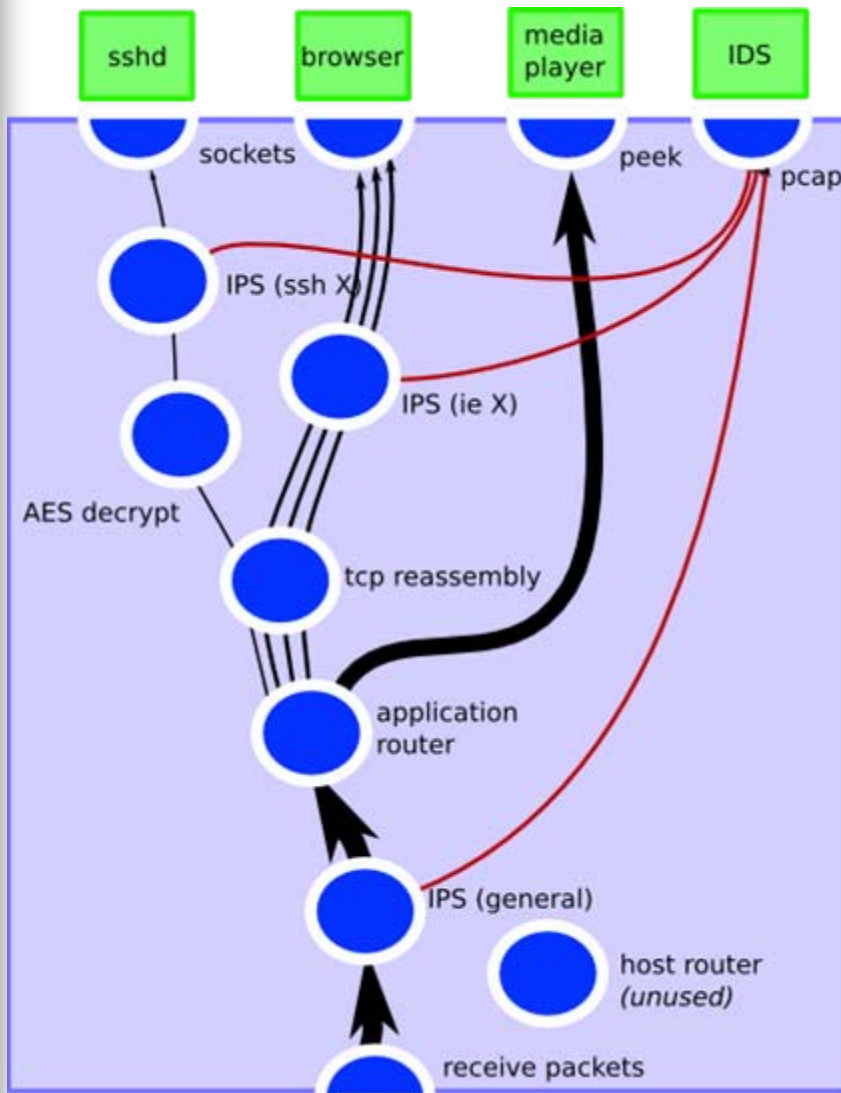
The screenshot shows a Konqueror web browser window titled "Fairly Fast Packet Filter - overview - Konqueror". The address bar contains the URL "http://ffpf.sourceforge.net/general/overview.php". The page content is as follows:

<h1>FFPF</h1>	<h2>&gt;&gt; overview</h2>	
<b>general</b> <ul style="list-style-type: none"><li>overview</li><li>news</li><li>downloads</li><li>webcvs</li><li>project summary</li><li>people</li></ul>	<b>Introduction</b> <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. FFPF is designed for high speed by pushing computation to the hardware.</p>	<b>News</b> <p><b>FFPF 1.3 developer reference documentation online</b> <i>wdebruij - 2004-07-18 10:19</i></p> <p>[Read]</p>
<b>science</b> <ul style="list-style-type: none"><li>publications</li></ul>		

# Questions?



# optimization: just-in-time placement



# optimization: just-in-time placement

