High-Performance Intrusion Detection with the Open-Source Bro NIDS

Robin Sommer
International Computer Science Institute, &
Lawrence Berkeley National Laboratory

robin@icsi.berkeley.edu
http://www.icir.org

Network Monitoring, Advanced Attack Detection and Prevention
Guest Lecture, RWTH Aachen
December 2010

Outline

1. *Overview of the Bro Network Intrusion Detection System*
   - Philosophy
   - Deployment
   - Architecture and Usage
   - Specific Capability: Dynamic Protocol Detection
2. *High Performance with Concurrent Traffic Analysis*
   - Concurrency Potential
   - Coarse-grained Parallelism: A cluster for load-balancing
   - Fine-grained Parallelism: Designing a multi-threaded NIDS
3. *Future Directions*

System Philosophy

- Bro has been developed at LBNL & ICSI since 1996
  - Designed and originally developed by Vern Paxson, now at ICSI & UC Berkeley
- Bro provides a real-time network analysis framework
  - Primary a network intrusion detection system (NIDS)
  - However it is also used for pure traffic analysis
- Focus is on
  - Application-level semantic analysis (rather than analyzing individual packets)
  - Tracking information over time
- Strong separation of mechanism and policy
  - The core of the system is policy-neutral (no notion of "good" or "bad")
  - User provides local site policy

System Philosophy (2)

- Operators *program* their policy
  - Not really meaningful to talk about what Bro detects "by default"
- Analysis model is *not* signature matching
  - Bro is fundamentally different from, e.g., Snort (though it *can* do signatures as well)
- Analysis model is *not* anomaly detection
  - Though it does support such approaches (and others) in principle
- System thoroughly logs all activity
  - It does not just alert
  - Logs are invaluable for forensics

Target Environments

- Bro is specifically well-suited for scientific environments
  - Extremely useful in networks with liberal ("default allow") policies
  - High-performance on commodity hardware
  - Supports intrusion prevention schemes
  - Open-source with a BSD license
- It does however require some effort to use effectively
  - Pretty complex, script-based system
  - Requires understanding of the network
  - No GUI, just ASCII logs
  - Only partially documented
  - Have been lacking resources to fully polish the system
  - **But**: This is actually just changing; more later
- Development has primarily been driven by *research*
  - However, our focus is operational use; we invest much time into "practical" issues
  - Want to bridge gap between research and operational deployment

Installations

Site	Systems
Berkeley National Laboratory	15,000
Los Alamos	1000s
National Center for Atmospheric Research	6,500
National Center for Supercomputing Applications	5,000
NERSC	5,000
Pittsburgh Supercomputing Center	300
UC Berkeley	100,000
UC Riverside	10,000
The Ohio State University	85,000

... and many more in academia, government, and some in industry.

Lawrence Berkeley National Lab

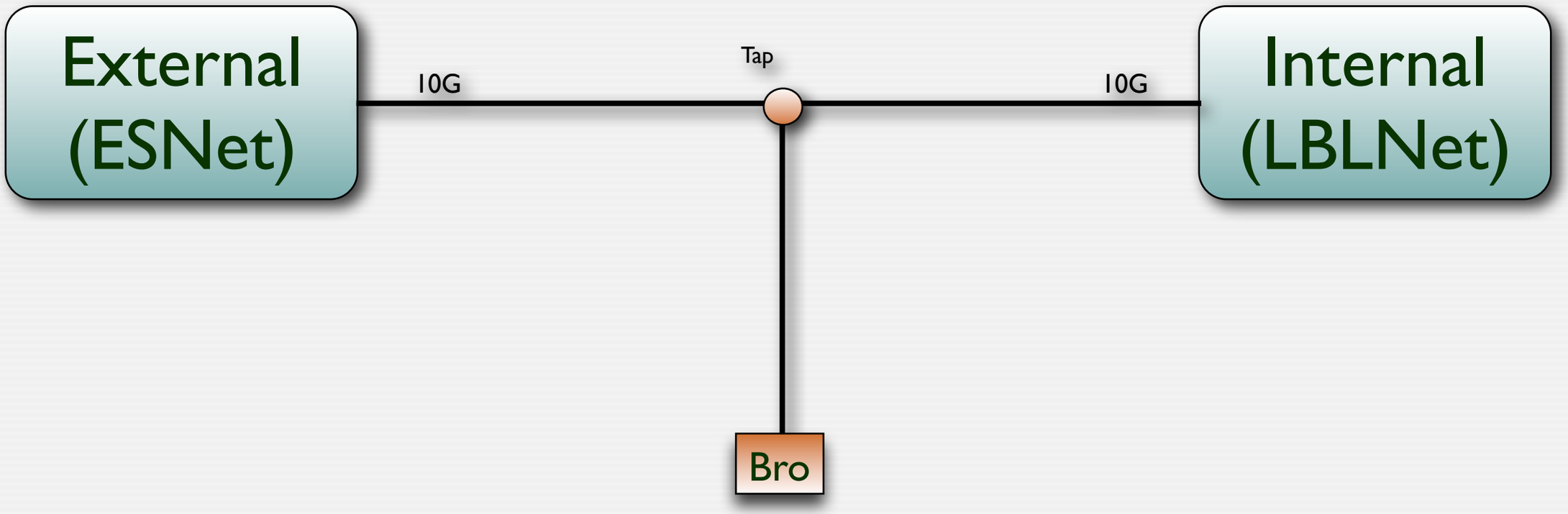
- Main site located on a 200-acre area in the Berkeley hills
- Close proximity to UC Berkeley

- LBNL has been using Bro operationally for >10 years
- It is one of the main components of the lab's network security infrastructure

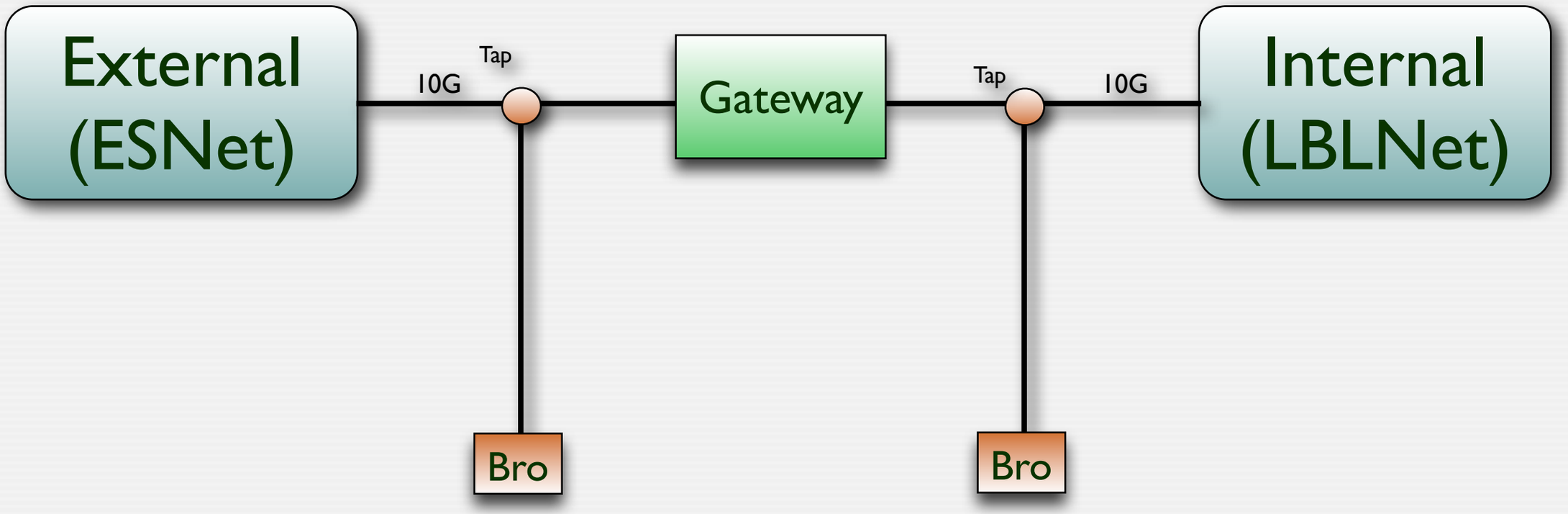
Lawrence Berkeley National Lab (2)

- A Department of Energy National Lab
  - Former head, Stephen Chu, now leading Secretary of Energy
- Open, unclassified research
  - Research is freely shared
  - Nanotechnology, Energy, Physics, Biology, Chemistry, Environmental, Computing
- Diverse user community
  - Scientific facilities used by researchers around the world
  - Many users are transient and not employees
  - Many of the staff have dual appointments with UC Berkeley
- Very liberal, default-allow security policy
  - Characteristic for many research environment, but makes monitoring challenging
  - Government-influenced thread-model

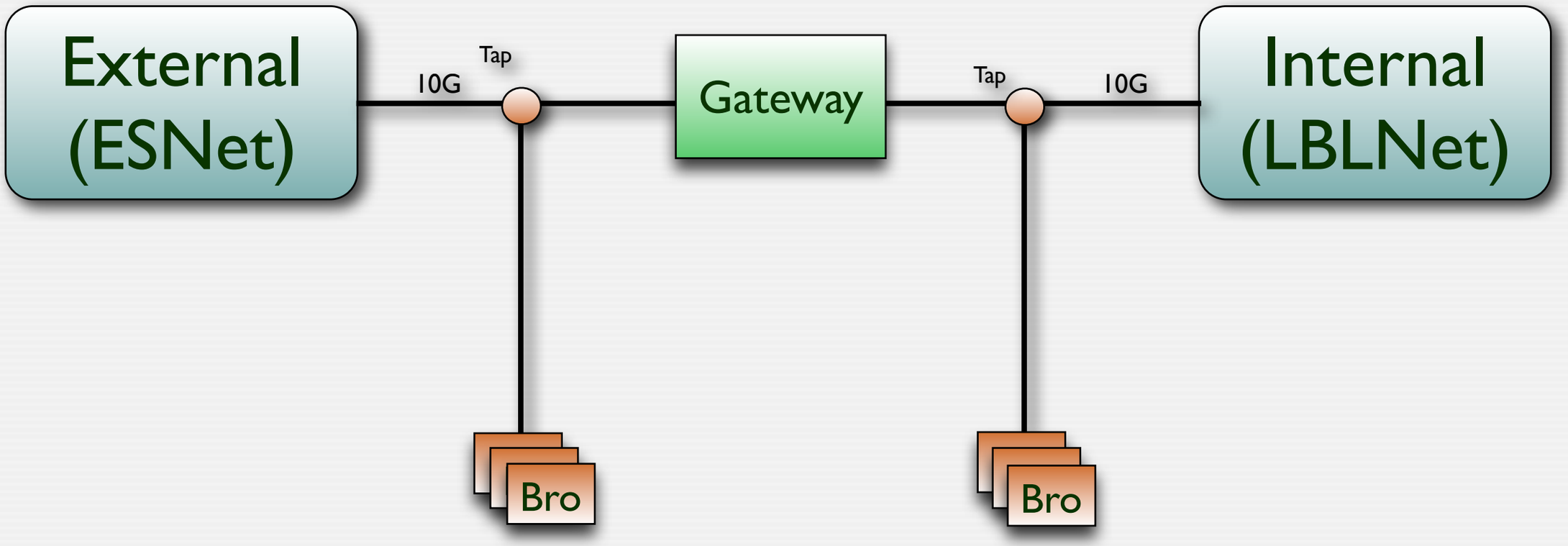
LBNL's Bro Setup



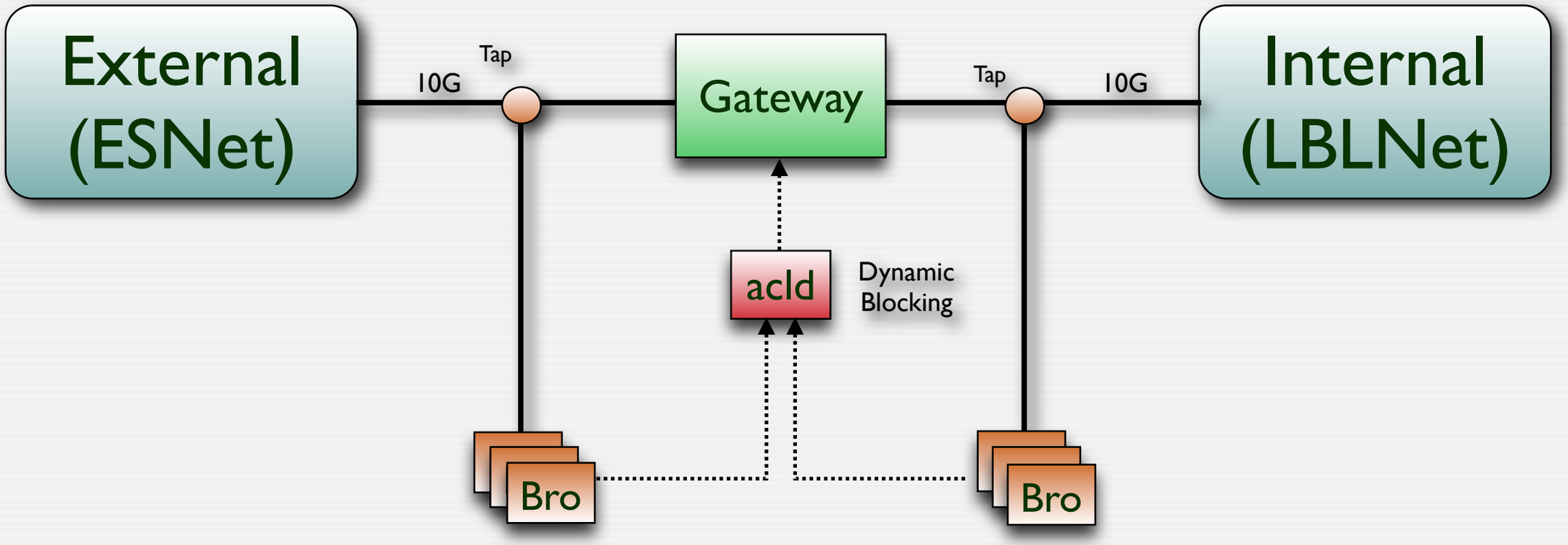
LBNL's Bro Setup



LBNL's Bro Setup



LBNL's Bro Setup



Bro blocks several thousands addresses per day!

Activity Logs: Connections

- One-line summaries for all TCP connections
- Most basic, yet also one of the most useful analyzers

```
> bro -i en0 tcp
```

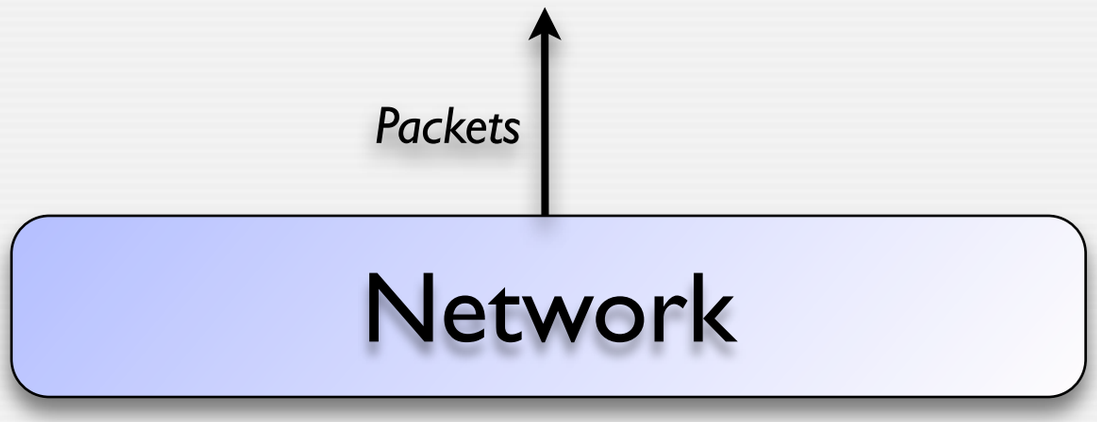
```
Time                 Duration  Source           Destination
1144876596.658302    1.206521  192.150.186.169  62.26.220.2 \
  http  53052    80         tcp    874          1841        SF      X
  Serv  SrcPort  DstPort    Proto  SrcBytes     DstBytes    State   Dir
```

LBNL has connection logs for every connection attempt since June 94!

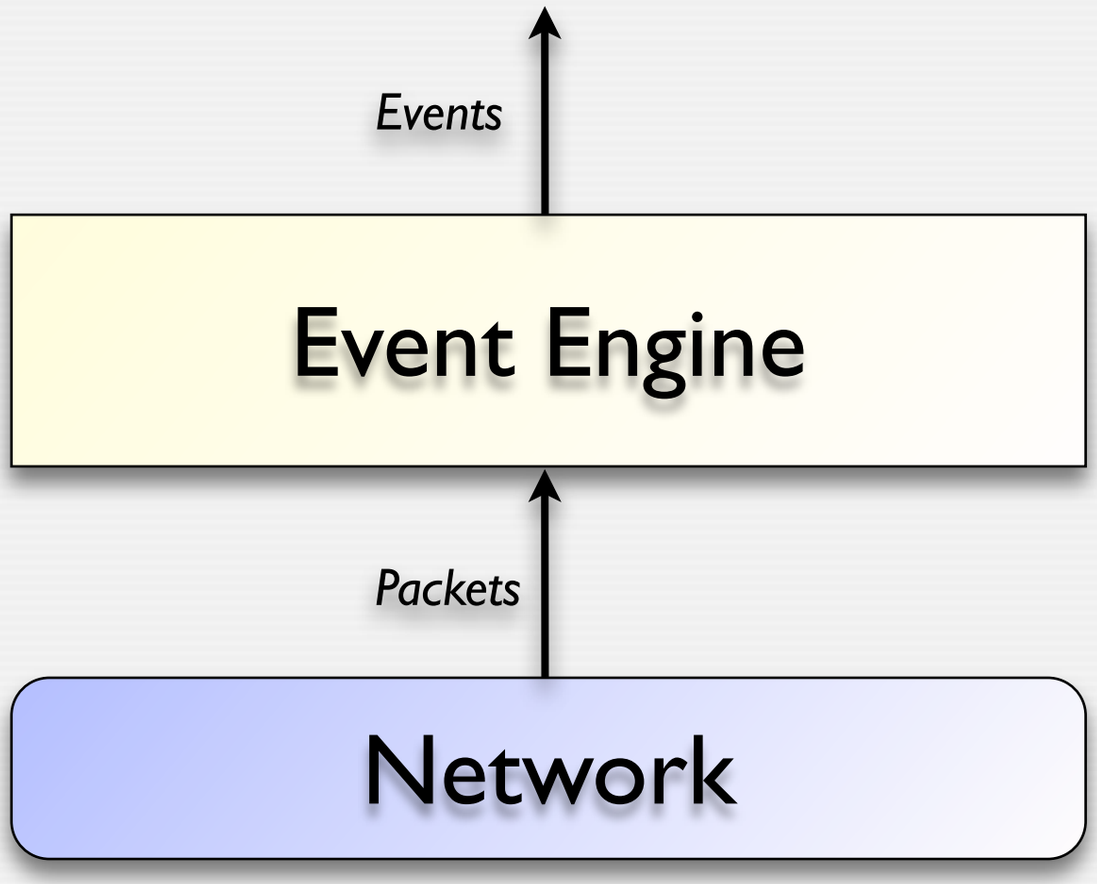
Activity Logs: HTTP Sessions

```
1144876588.30 %2 start 192.150.186.169:53041 > 195.71.11.67:80
1144876588.30 %2 GET /index.html (200 "OK" [57634] www.spiegel.de)
1144876588.30 %2 > HOST:  www.spiegel.de
1144876588.30 %2 > USER-AGENT:  Mozilla/5.0 (Macintosh; PPC Mac OS ...
1144876588.30 %2 > ACCEPT:  text/xml,application/xml,application/xhtml ...
1144876588.30 %2 > ACCEPT-LANGUAGE:  en-us,en;q=0.7,de;q=0.3
[...]
1144876588.77 %2 < SERVER:  Apache/1.3.26 (Unix) mod_fastcgi/2.2.12
1144876588.77 %2 < CACHE-CONTROL:  max-age=120
1144876588.77 %2 < EXPIRES:  Wed, 12 Apr 2006 21:18:28 GMT
[...]
1144876588.77 %2 <= 1500 bytes: "<!-- Vignette StoryServer 5.0 Wed Apr..."
1144876588.78 %2 <= 1500 bytes: "r "http://spiegel.ivwbox.de" r..."
1144876588.78 %2 <= 1500 bytes: "icon.ico" type="image/ico">^M^J ..."
1144876588.94 %2 <= 1500 bytes: "erver 5.0 Mon Mar 27 15:56:55 ..."
[...]
```

Architecture



Architecture

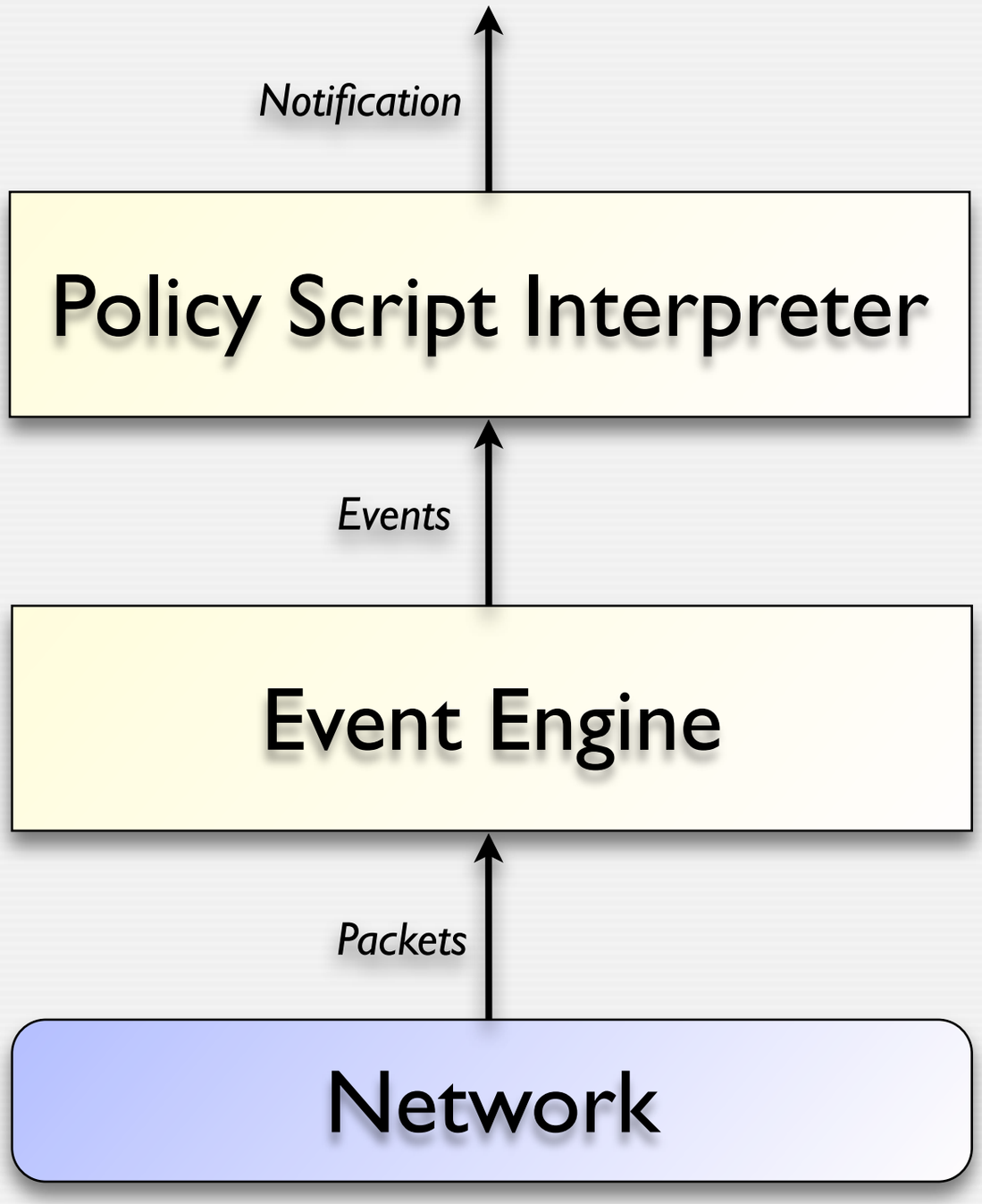


Packet Analysis

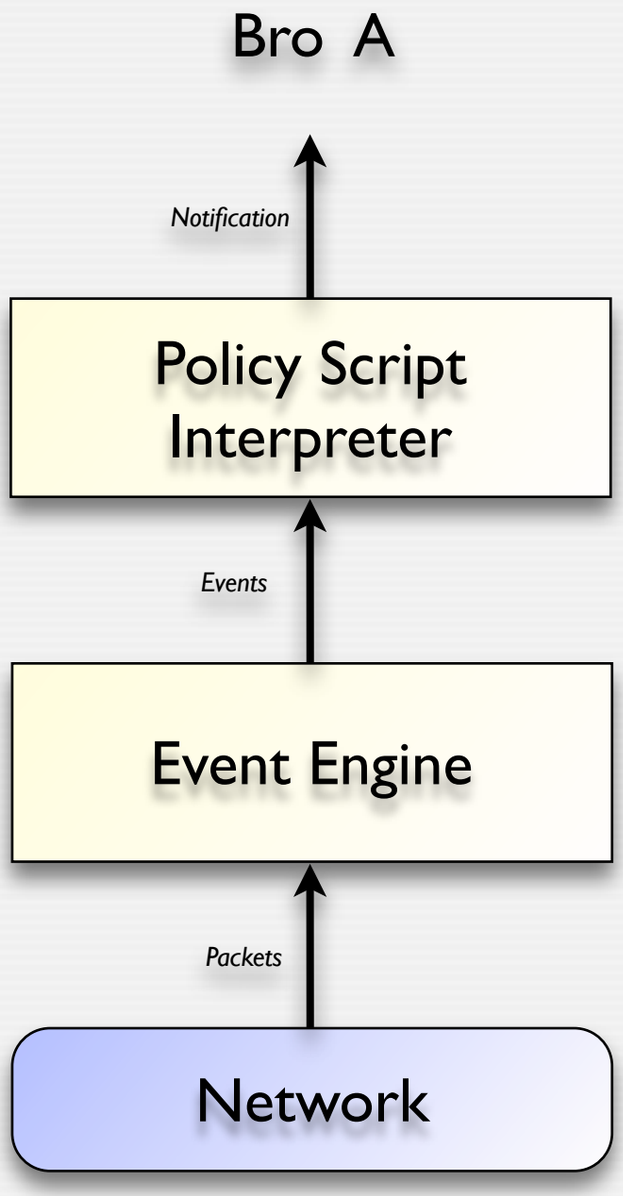
Architecture

Detection Logic

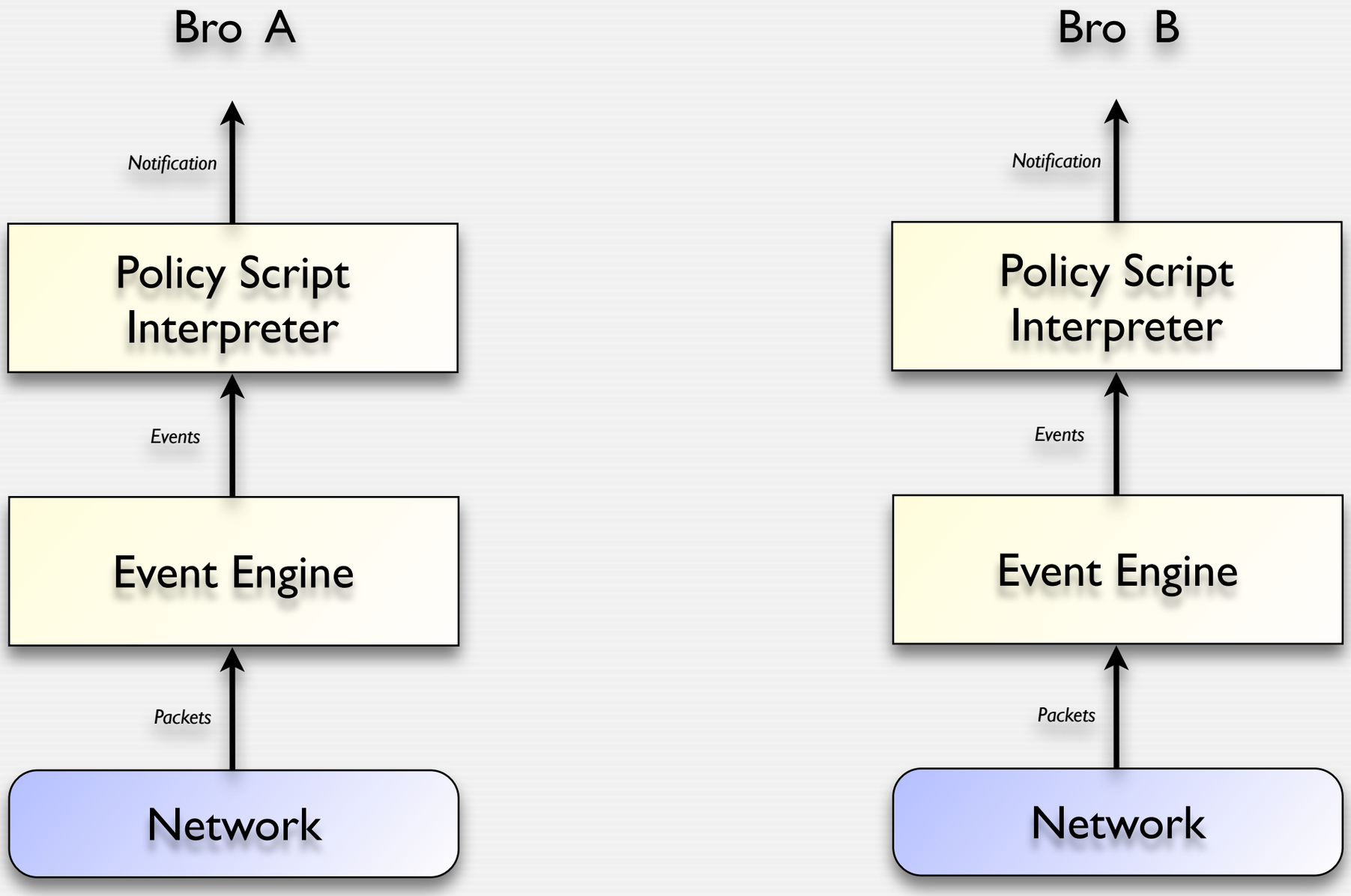
Packet Analysis



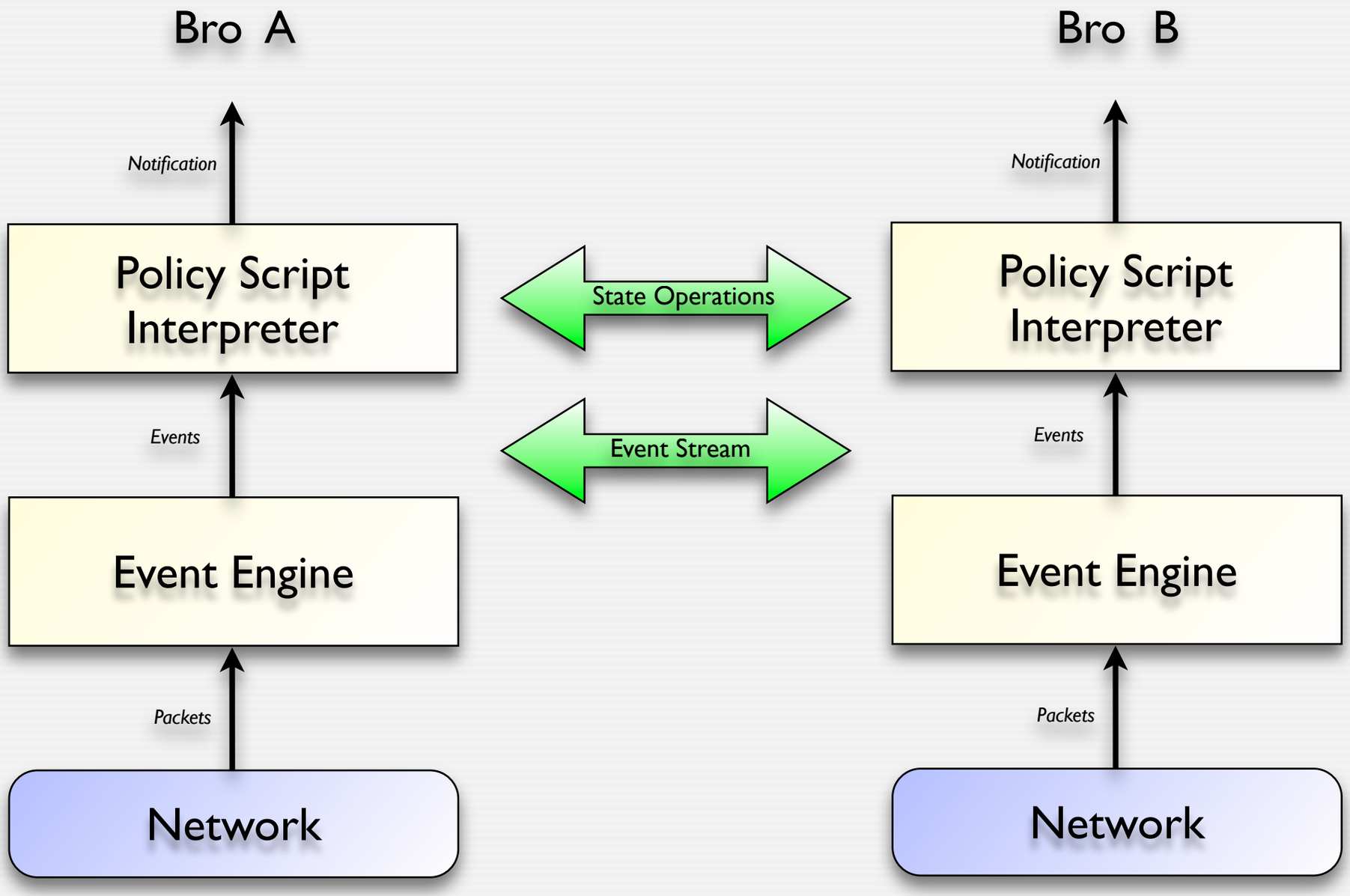
Communication Architecture



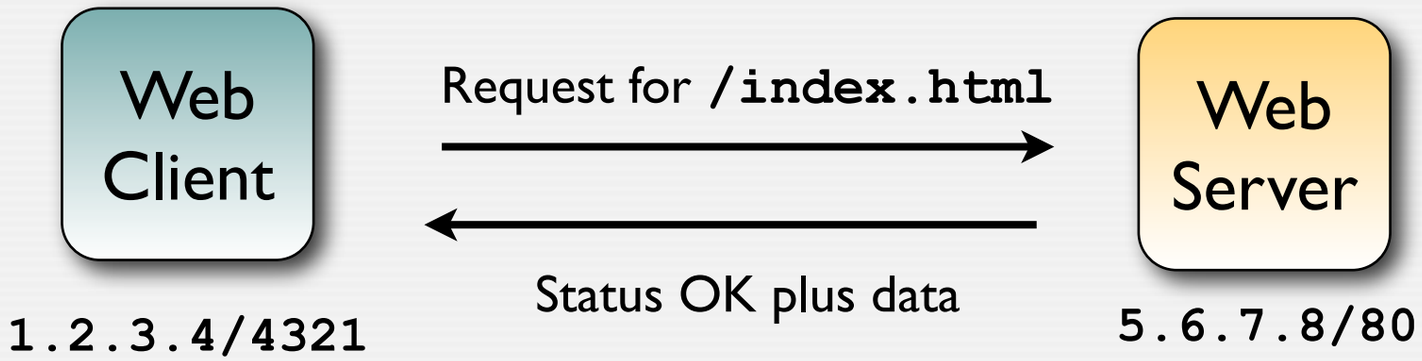
Communication Architecture



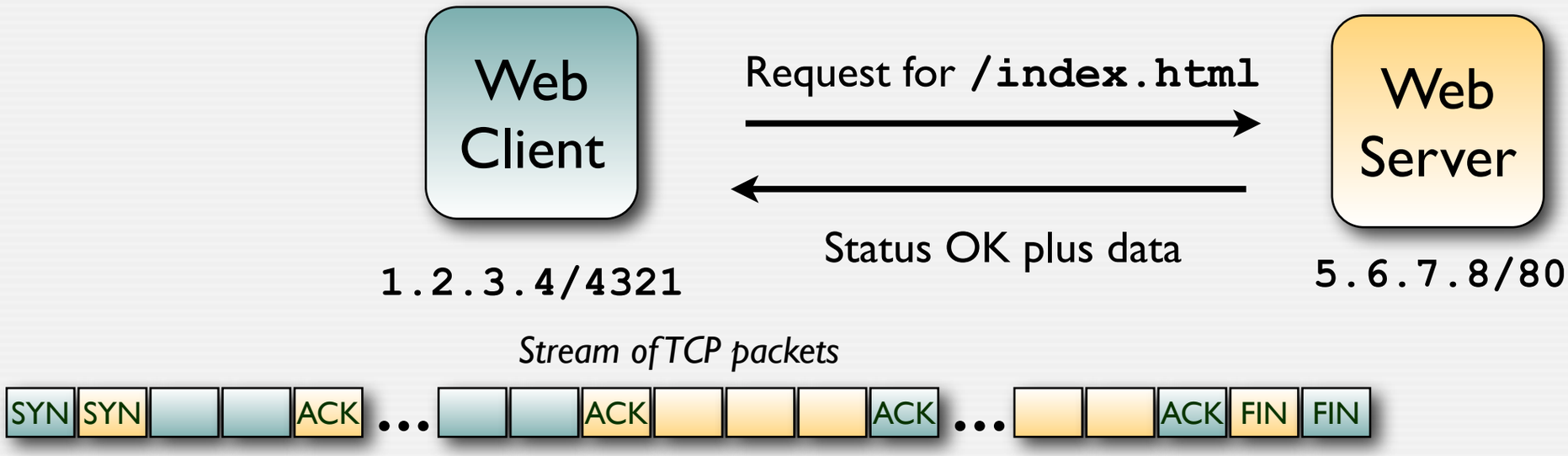
Communication Architecture



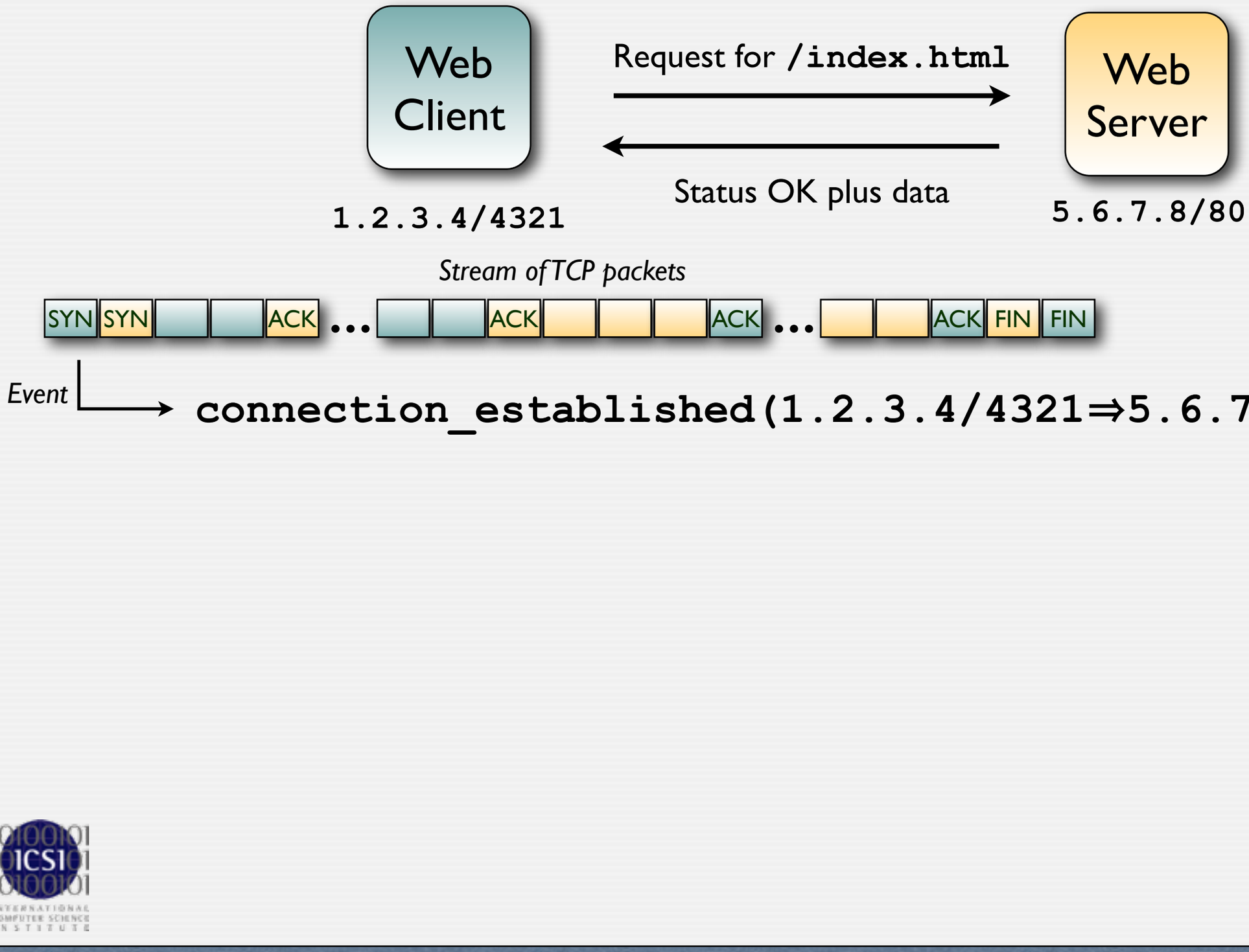
Event Model


Web Client
Request for /index.html
Web Server
Status OK plus data
1.2.3.4/4321
5.6.7.8/80

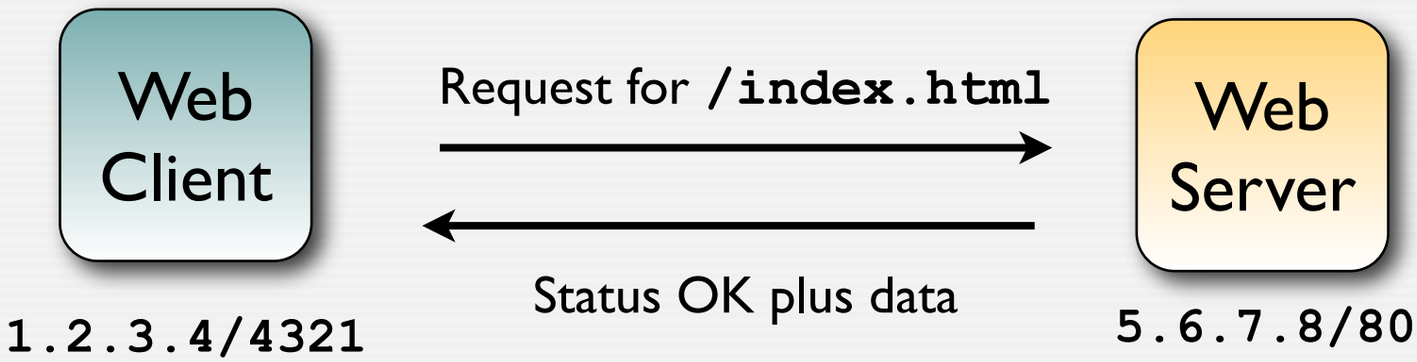
Event Model


Web Client
Request for /index.html
Web Server
Status OK plus data
1.2.3.4/4321
5.6.7.8/80
Stream of TCP packets
SYN
SYN
ACK
ACK
ACK
ACK
FIN
FIN

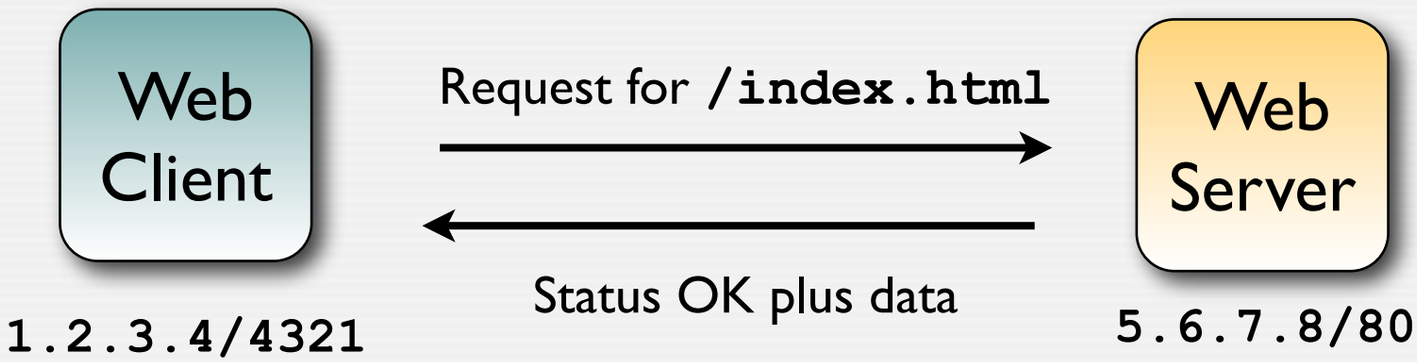
Event Model


Web Client
Request for /index.html
Web Server
Status OK plus data
1.2.3.4/4321
5.6.7.8/80
Stream of TCP packets
SYN
SYN
ACK
ACK
ACK
ACK
FIN
FIN
Event
connection established(1.2.3.4/4321⇒5.6.7.8/80)

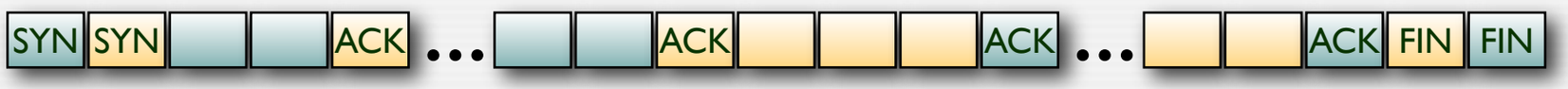
Event Model


Web Client
Request for /index.html
Web Server
Status OK plus data
1.2.3.4/4321
5.6.7.8/80
Stream of TCP packets
SYN
SYN
ACK
ACK
ACK
ACK
FIN
FIN
Event
connection established(1.2.3.4/4321⇒5.6.7.8/80)
TCP stream reassembly for originator
Event
http_request(1.2.3.4/4321⇒5.6.7.8/80, "GET", "/index.html")

Event Model



Stream of TCP packets



Event → `connection_established(1.2.3.4/4321⇒5.6.7.8/80)`

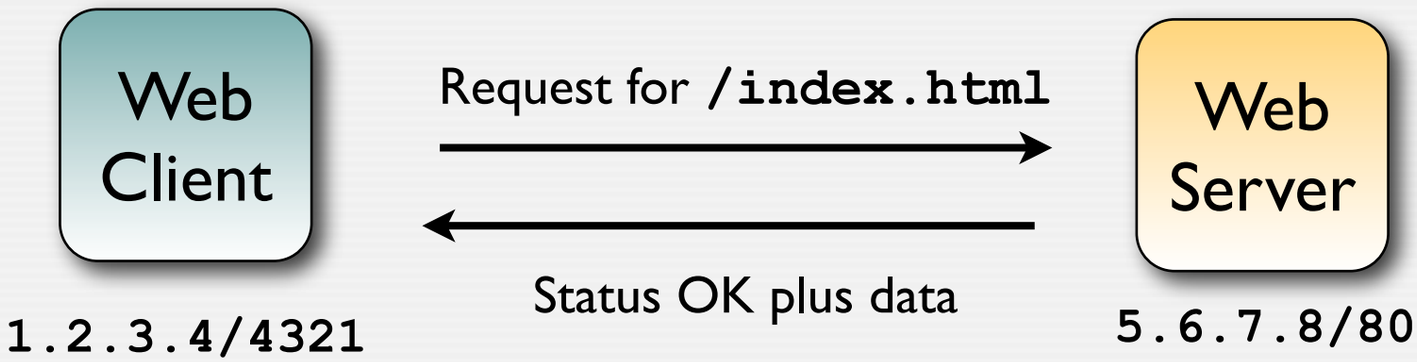
TCP stream reassembly for originator

Event → `http_request(1.2.3.4/4321⇒5.6.7.8/80, "GET", "/index.html")`

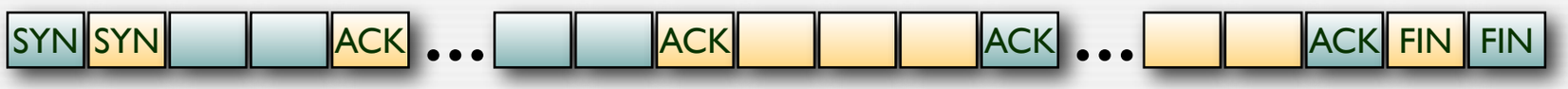
TCP stream reassembly for responder

Event → `http_reply(1.2.3.4/4321⇒5.6.7.8/80, 200, "OK", data)`

Event Model



Stream of TCP packets



Event → `connection established(1.2.3.4/4321⇒5.6.7.8/80)`

TCP stream reassembly for originator

Event → `http_request(1.2.3.4/4321⇒5.6.7.8/80, "GET", "/index.html")`

TCP stream reassembly for responder

Event → `http_reply(1.2.3.4/4321⇒5.6.7.8/80, 200, "OK", data)`

Event → `connection_finished(1.2.3.4/4321, 5.6.7.8/80)`

Event-Engine

- Event-engine is written in C++
- Performs *policy-neutral* analysis
  - Turns low-level activity into high-level events
  - Examples: `connection_established, http_request`
  - Events are annotated with context (e.g., IP addresses, URL)
- Contains *analyzers* for >30 protocols, including
  - ARP, IP, ICMP, TCP, UDP
  - DCE-RPC, DNS, FTP, Finger, Gnutella, HTTP, IRC, Ident, NCP, NFS, NTP, NetBIOS, POP3, Portmapper, RPC, Rsh, Rlogin, SMB, SMTP, SSH, SSL, SunRPC, Telnet
- Analyzers generate ~300 types of events

Policy Scripts

- Scripts process event stream, incorporating ...
  - ... context from past events
  - ... site's local security policy
- Scripts take actions
  - Generating alerts via syslog or mail
  - Executing program as a form of response
  - Recording activity to disk

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c.id.resp_h; # Responder's address
    local service = c.id.resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    alarm fmt("New SSH host found: %s", responder);
    }
```

Expressing Policy

- Custom, domain-specific language
  - Bro ships with 20K+ lines of script code
  - Default scripts detect attacks & log activity extensively
- Language is
  - Procedural
  - Event-based
  - Strongly typed
  - Rich in types
    - Usual script-language types, such as tables and sets
    - Domain-specific types, such as addresses, ports, subnets
  - Supporting state management (expiration, timers, etc.)
  - Supporting communication with other Bro instances

Port-based Analysis

- Bro has lots of application-layer analyzers
- But which protocol does a connection use?
- Traditionally NIDS rely on ports
  - Port 80? Oh, that's HTTP.
- Obviously deficient in two ways
  - There's non-HTTP traffic on port 80 (firewalls tend to open this port...)
  - There's HTTP on ports other than port 80
- Particularly problematic for security monitoring
  - Want to know if somebody avoids the well-known port

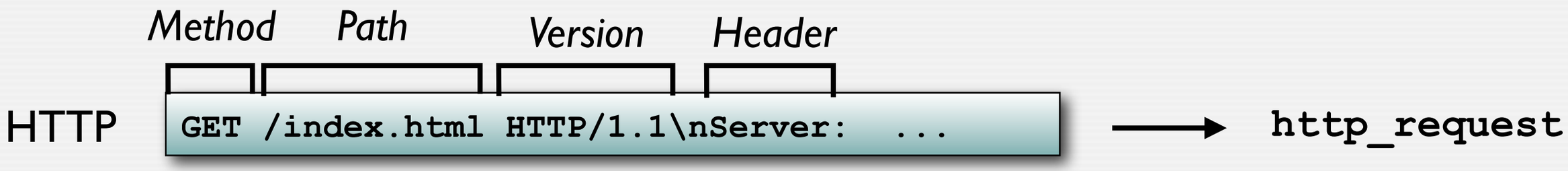
Dynamic Protocol Detection

Dynamic Protocol Detection


```
GET /index.html HTTP/1.1\nServer:  ...
```

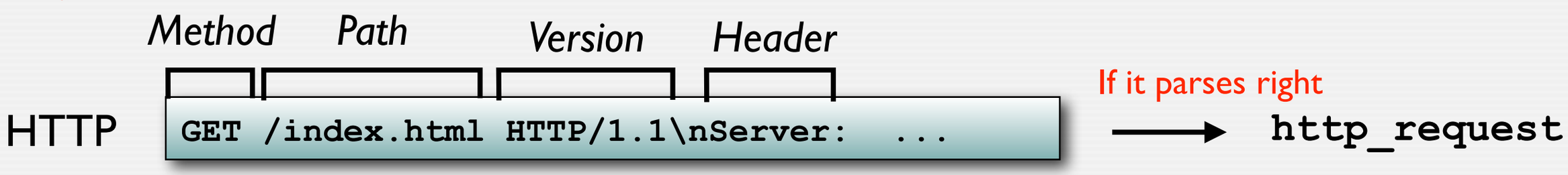
Dynamic Protocol Detection

Port 80?



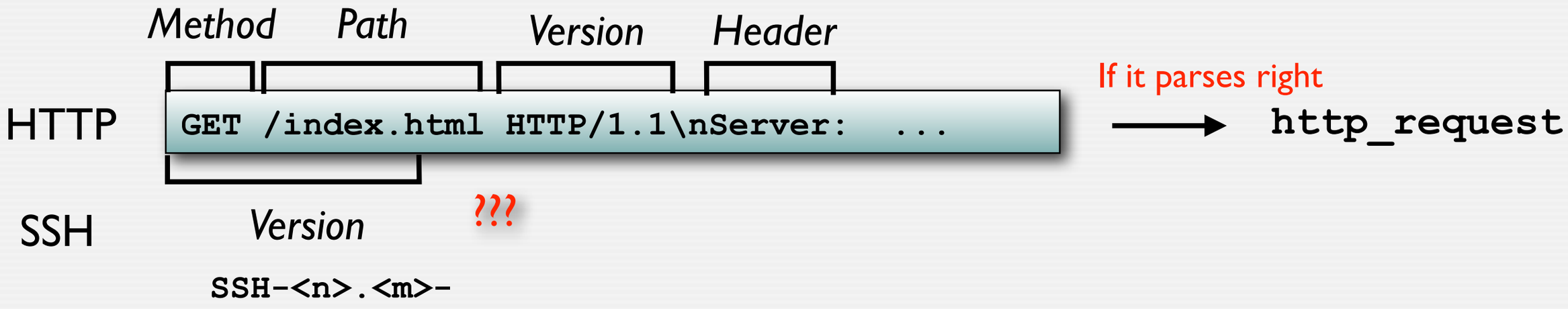
Dynamic Protocol Detection

~~Port 80?~~



Dynamic Protocol Detection

~~Port 80?~~

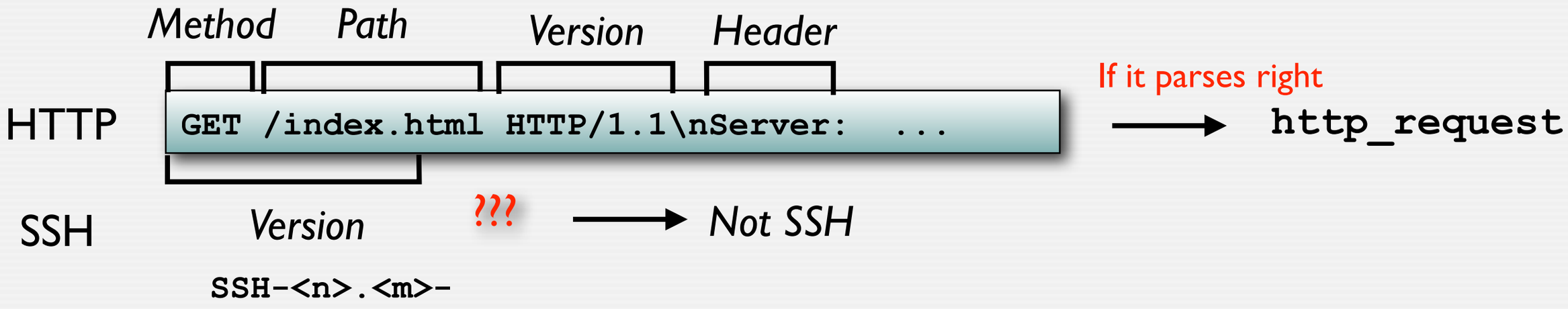


If it parses right

→ `http_request`

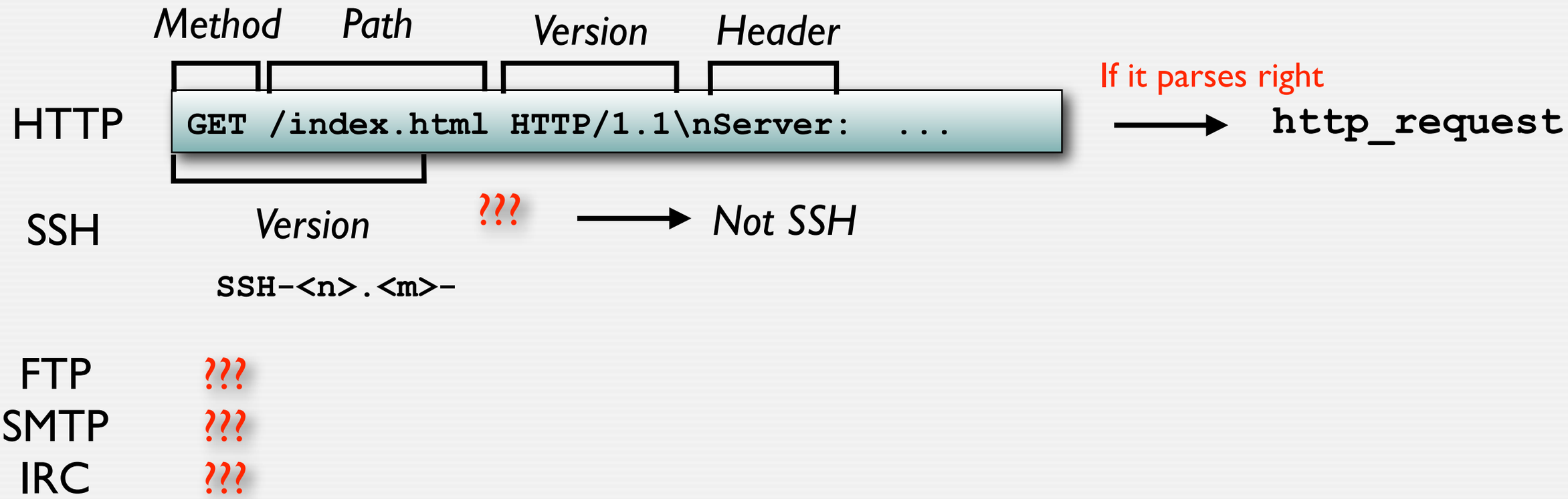
Dynamic Protocol Detection

~~Port 80?~~



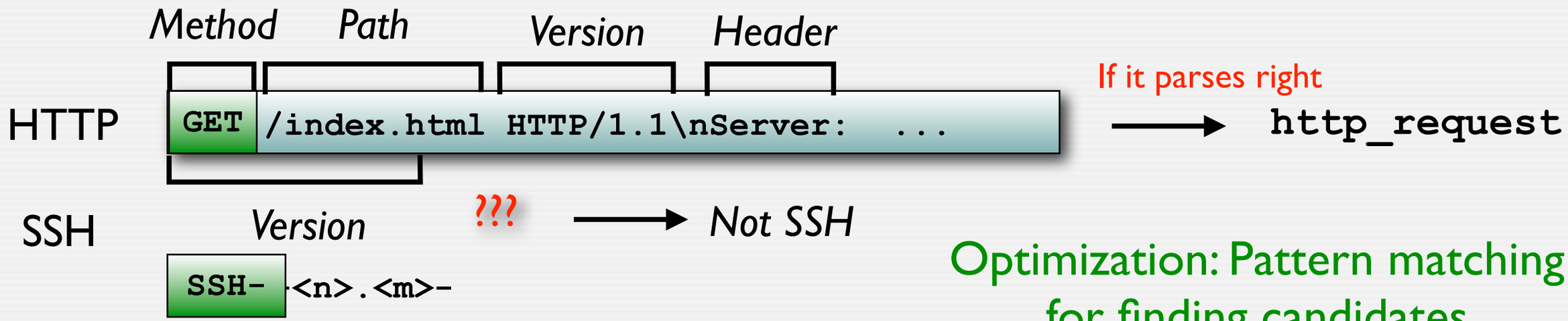
Dynamic Protocol Detection

~~Port 80?~~



Dynamic Protocol Detection

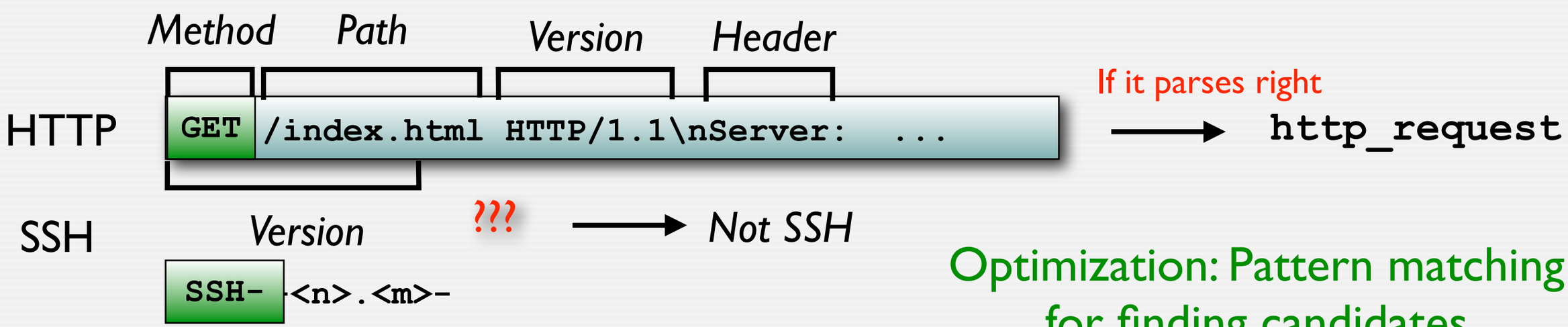
~~Port 80?~~



Optimization: Pattern matching for finding candidates

Dynamic Protocol Detection

~~Port 80?~~

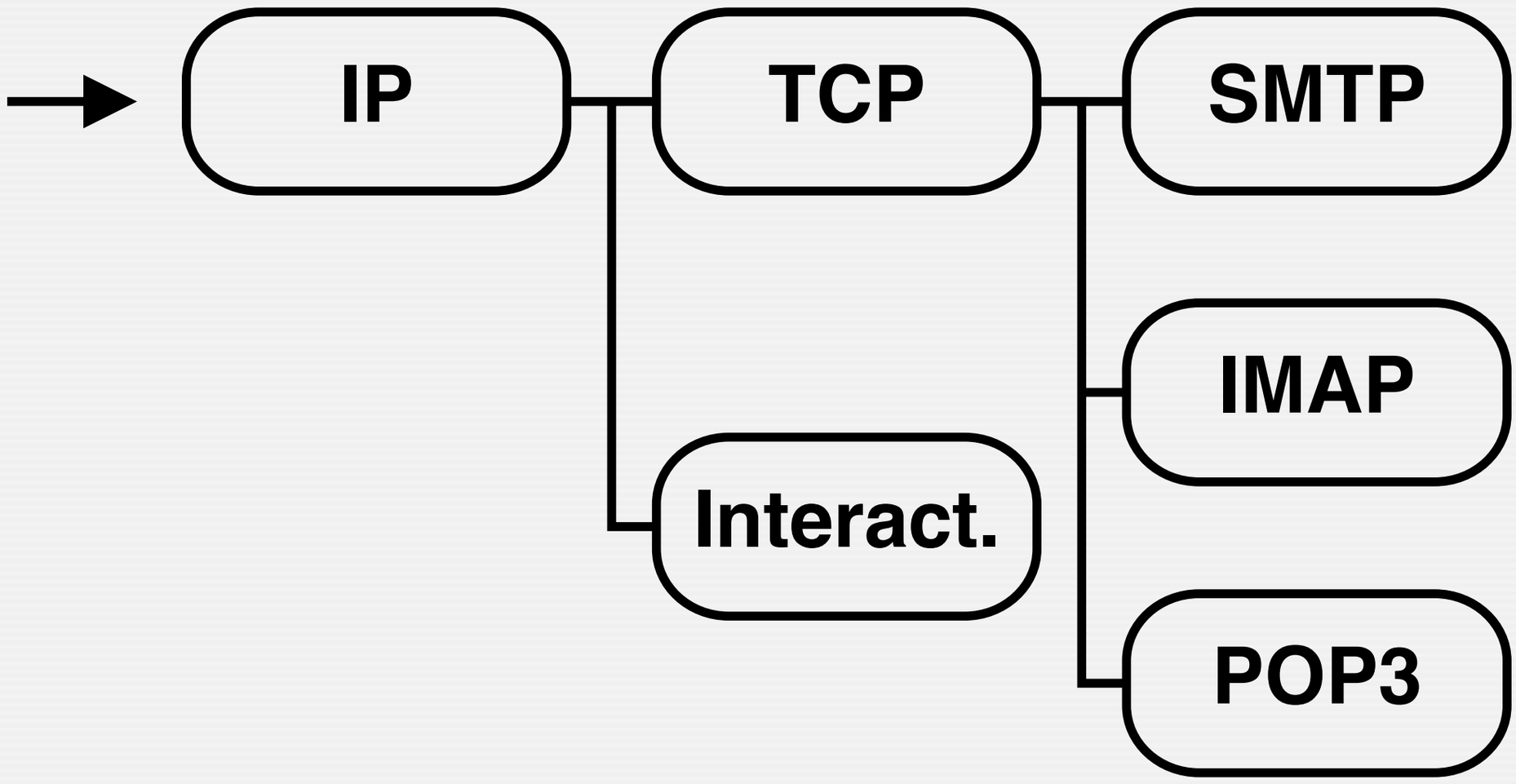


Optimization: Pattern matching for finding candidates

Dynamic Protocol Detection:

1. Identify potential analyzers
2. Verify decision by parsing

Analyzer Trees

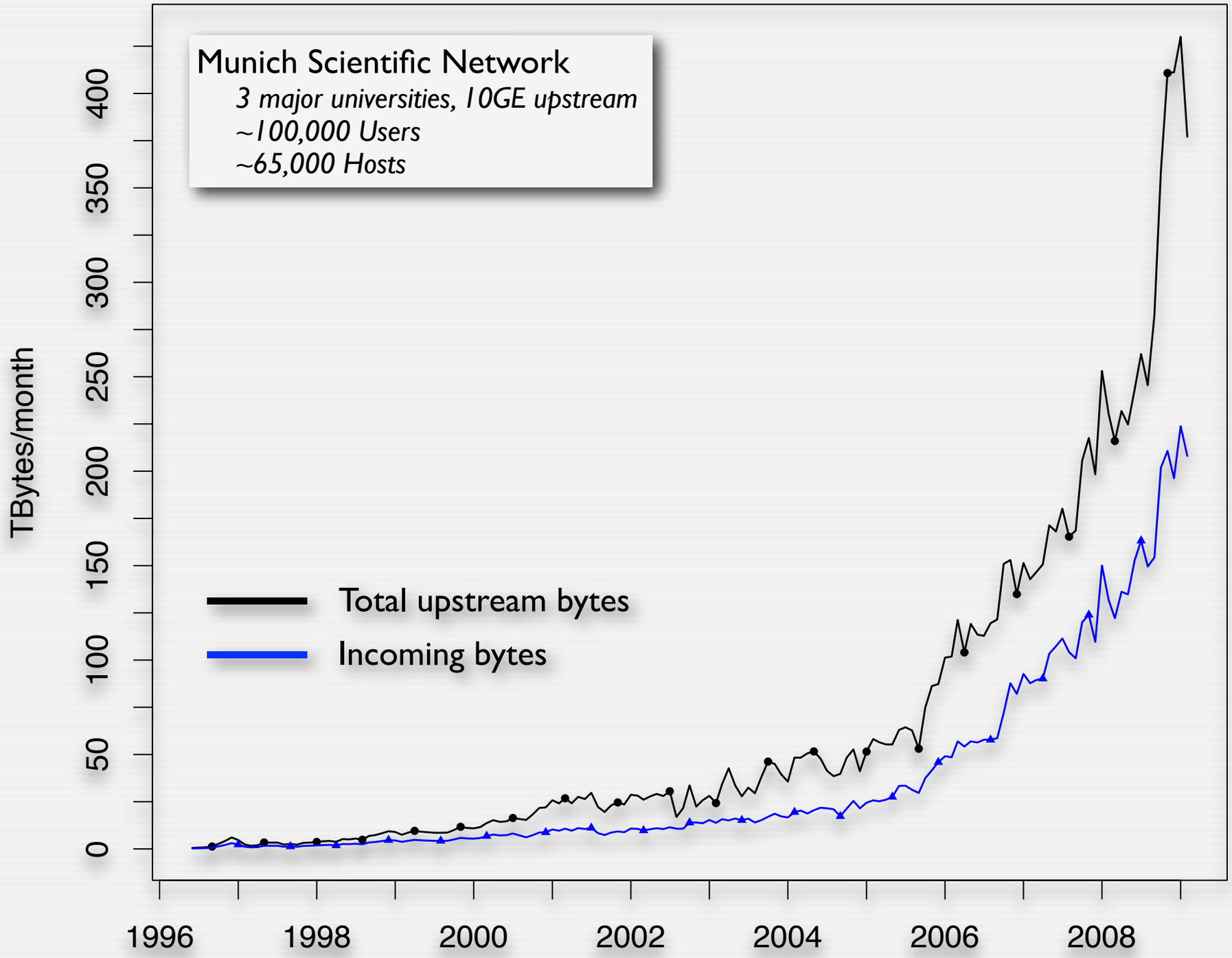


Another Application: FTP Data

```
xxx.xxx.xxx.xxx/2373 > xxx.xxx.xxx.xxx/5560 start
response (220 Rooted Moron Version 1.00 4 WinSock ready...)
USER ops (logged in)
SYST (215 UNIX Type: L8)
[...]
LIST -al (complete)
TYPE I (ok)
SIZE stargate.atl.s02e18.hdtv.xvid-tvd.avi (unavail)
PORT xxx,xxx,xxx,xxx,xxx,xxx (ok)
STOR stargate.atl.s02e18.hdtv.xvid-tvd.avi, NOOP (ok)
ftp-data video/x-msvideo `RIFF (little-endian) data, AVI'
[...]
response (226 Transfer complete.)
[...]
QUIT (closed)
```

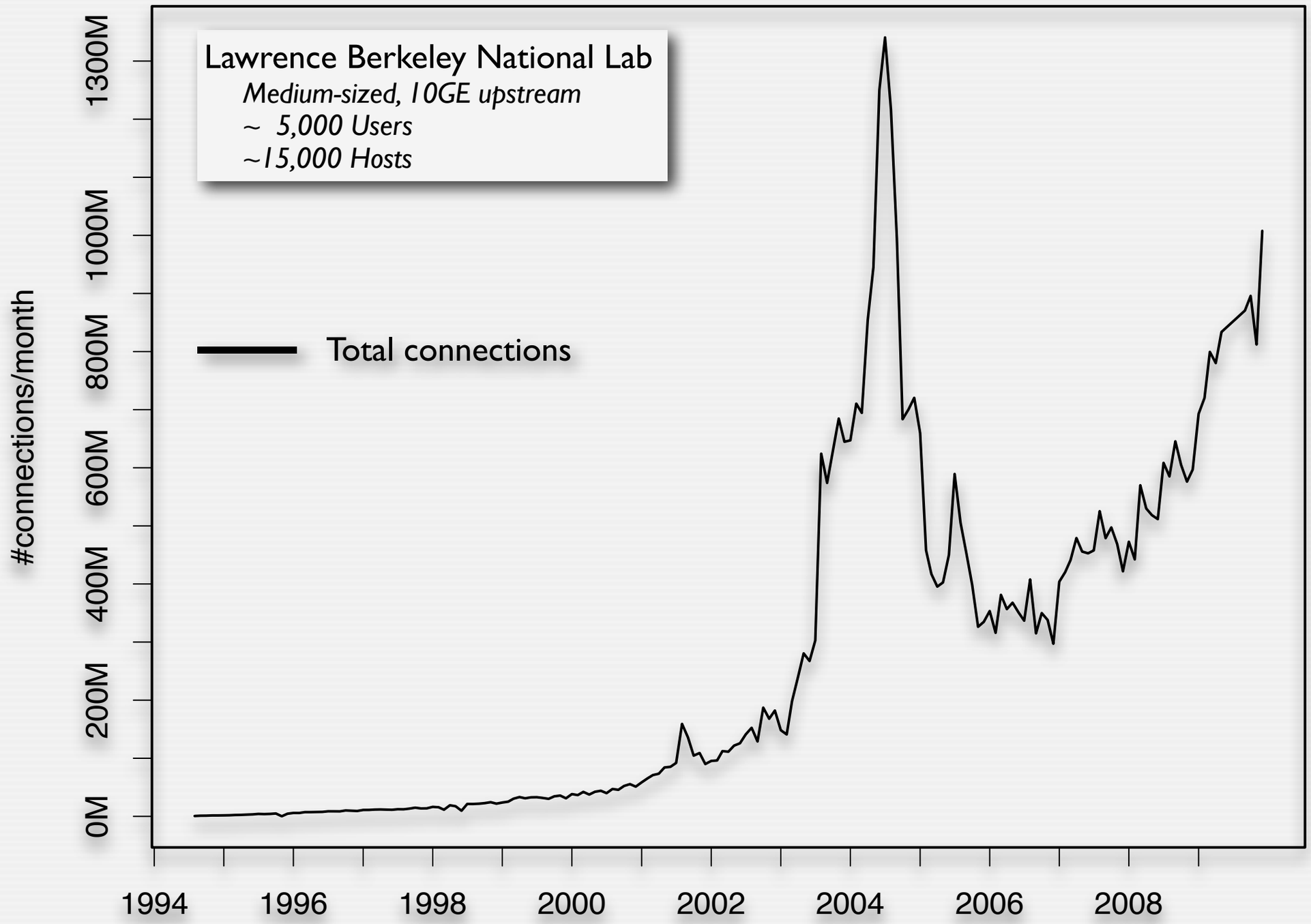
High Performance with Concurrent Traffic Analysis

Development of Internet Traffic



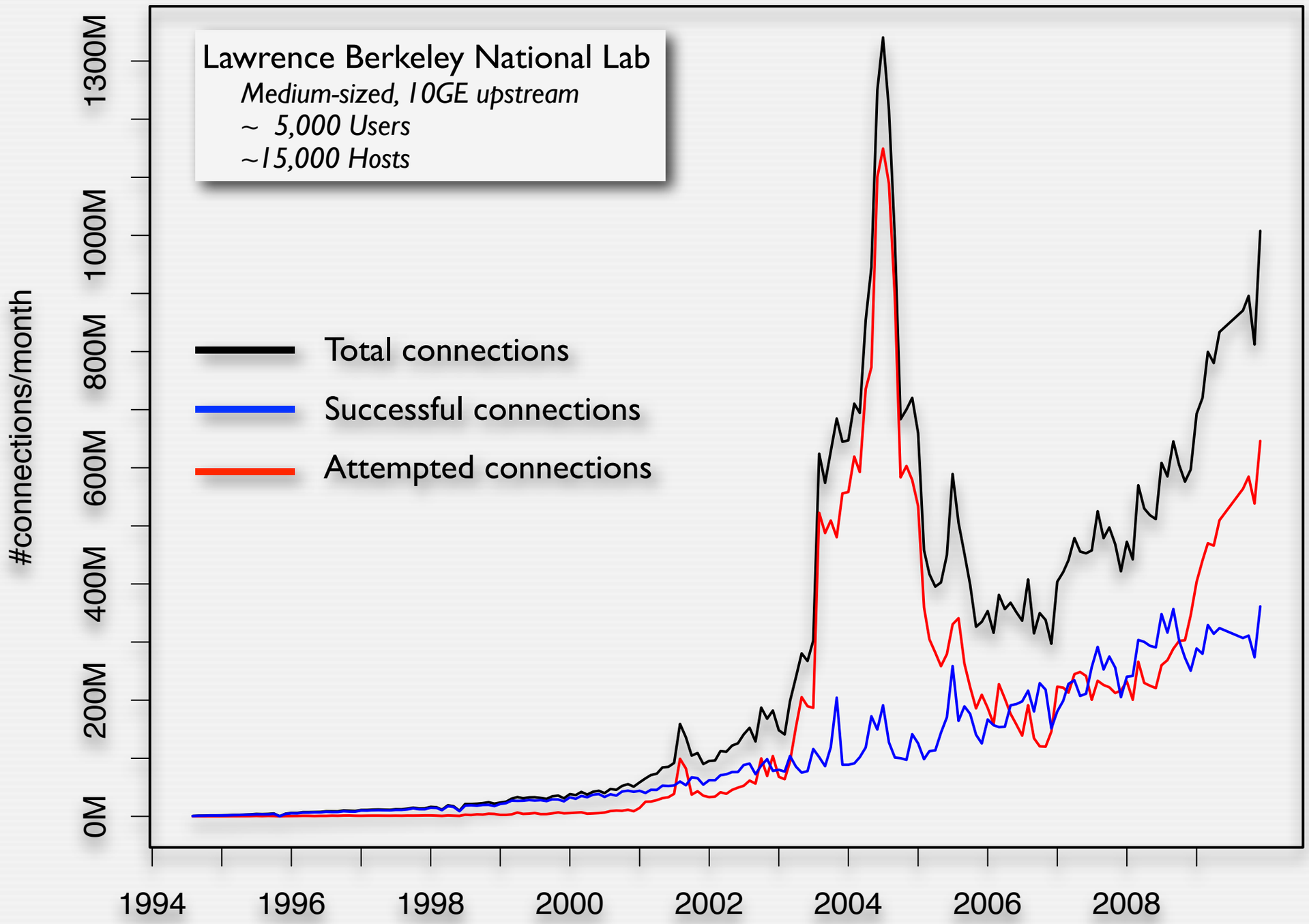
Data: Leibniz-Rechenzentrum, München

Internet Traffic: Connections



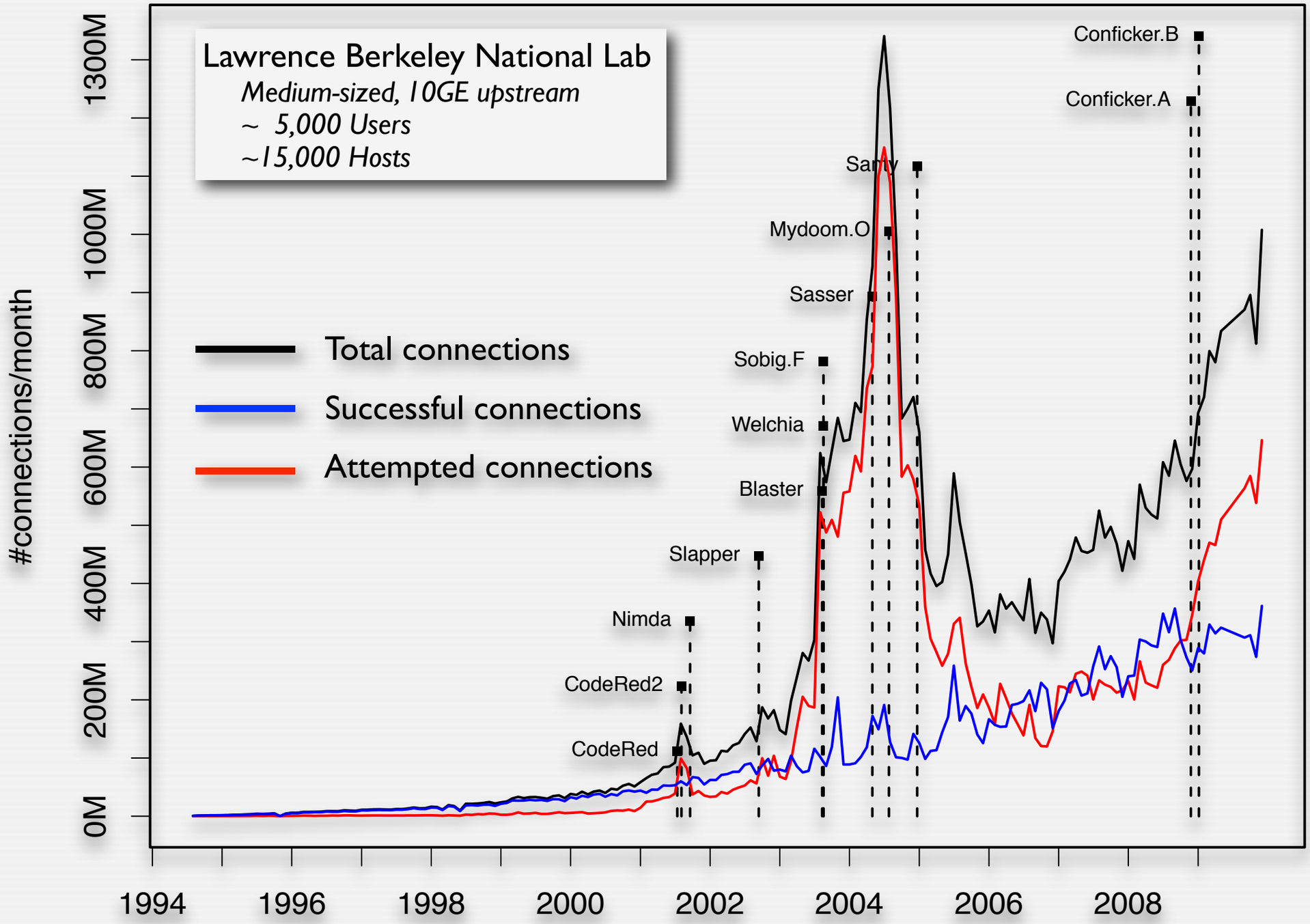
Data: Lawrence Berkeley National Lab

Internet Traffic: Connections



Data: Lawrence Berkeley National Lab

Internet Traffic: Connections



Data: Lawrence Berkeley National Lab

Need for Performance

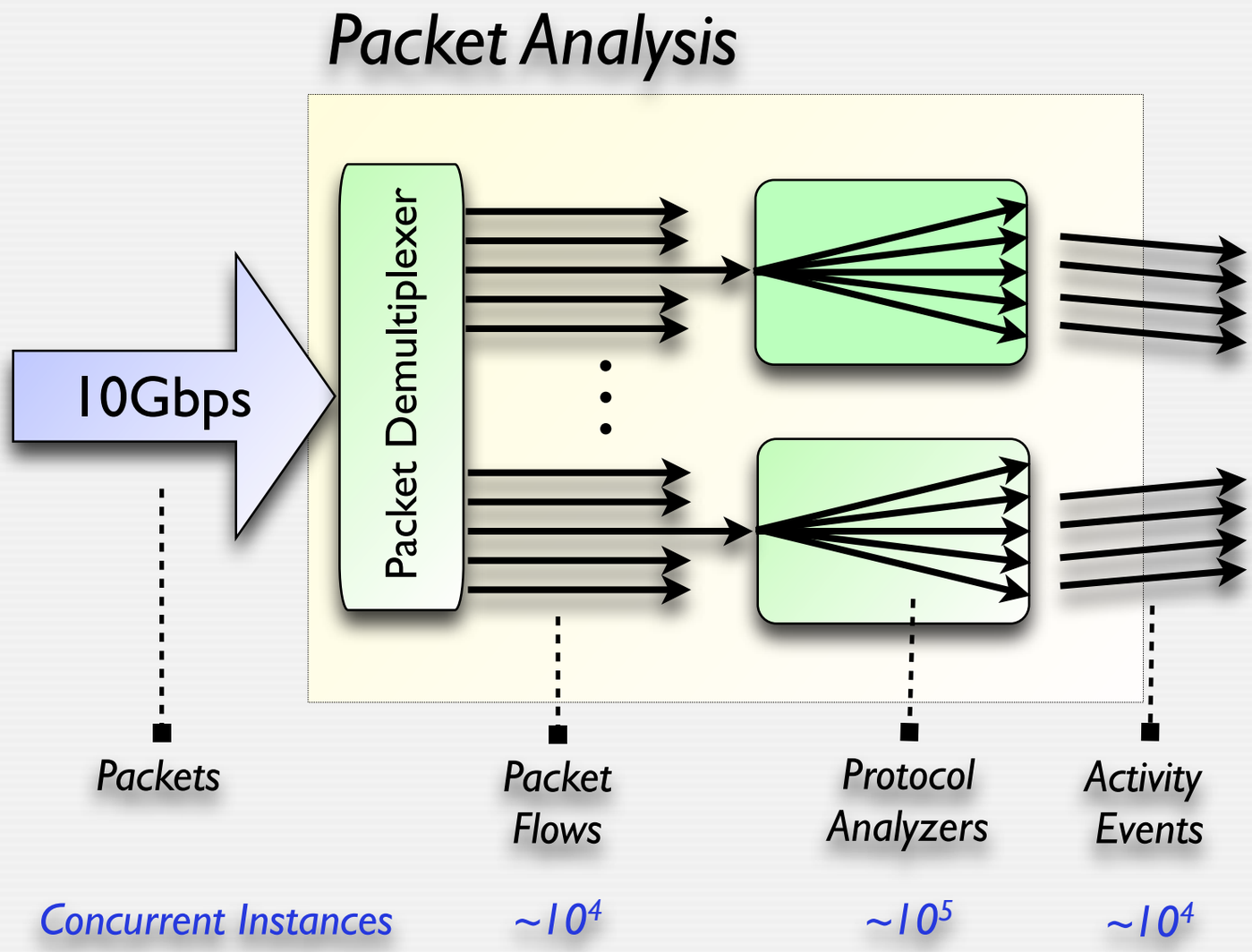
- Keep needing to do *more analysis* on *more data* at *higher speeds*
- NIDS used to run successfully on commodity hardware
  - In particular important for open-source NIDS (e.g., Snort, Bro)
- Not any more!
  - Moore's law doesn't hold for single-core performance anymore
  - Unfortunately, today's NIDS implementations are single-threaded and thus limited
- To overcome, we can
  - Significantly restrict the amount of analysis, *or*
  - Turn to expensive & inflexible custom hardware, *or*
  - *Parallelize processing to leverage commodity multi-core architectures*
- Parallelizing an application is inherently *domain-specific*
  - There's no generic approach to concurrency
  - Need examine carefully where the concurrency potential is that we can exploit

Outline

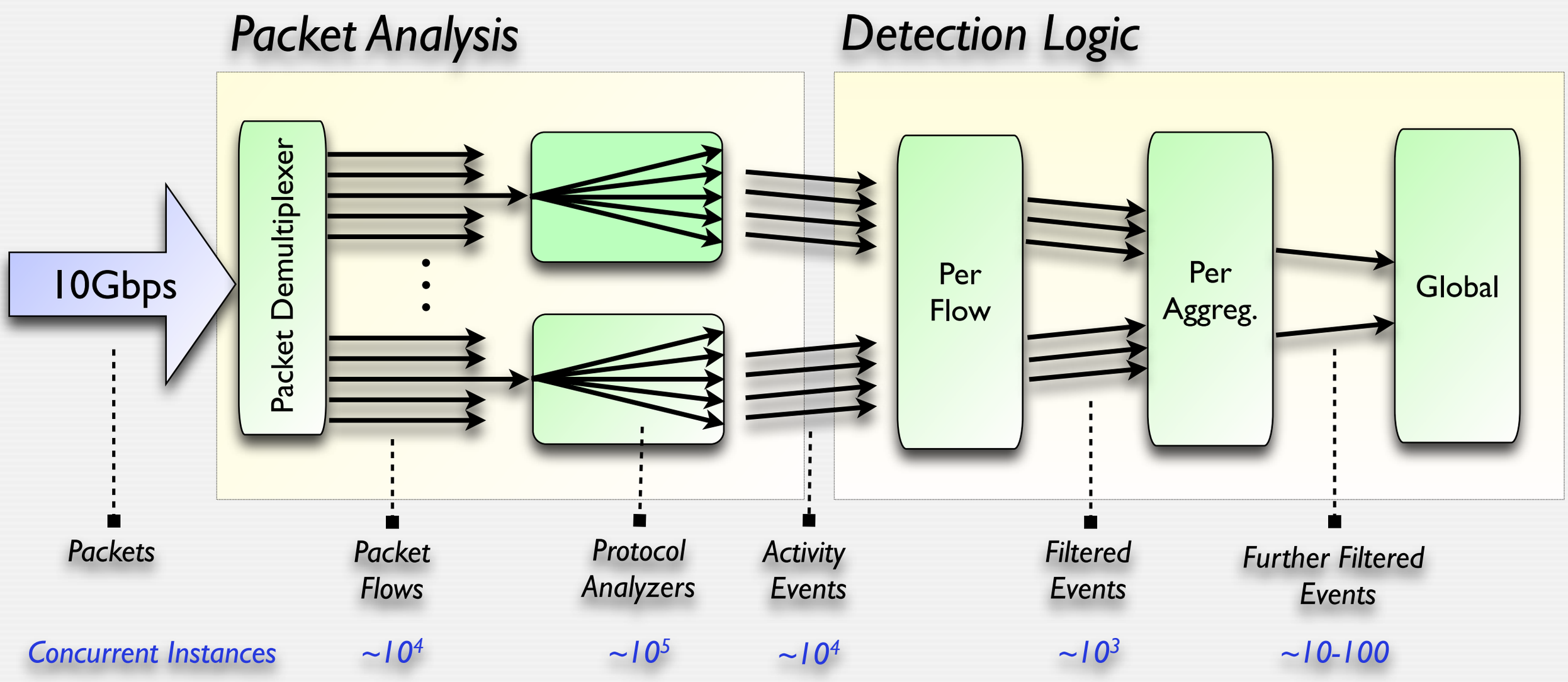
1. *Overview of the Bro Network Intrusion Detection System*
   - Philosophy
   - Deployment
   - Architecture and Usage
   - Specific Capability: Dynamic Protocol Detection
2. *High Performance with Concurrent Traffic Analysis*
   - *Concurrency Potential*
   - *Coarse-grained Parallelism: A cluster for load-balancing*
   - *Fine-grained Parallelism: Designing a multi-threaded NIDS*
3. *Future Directions*

Concurrency Potential

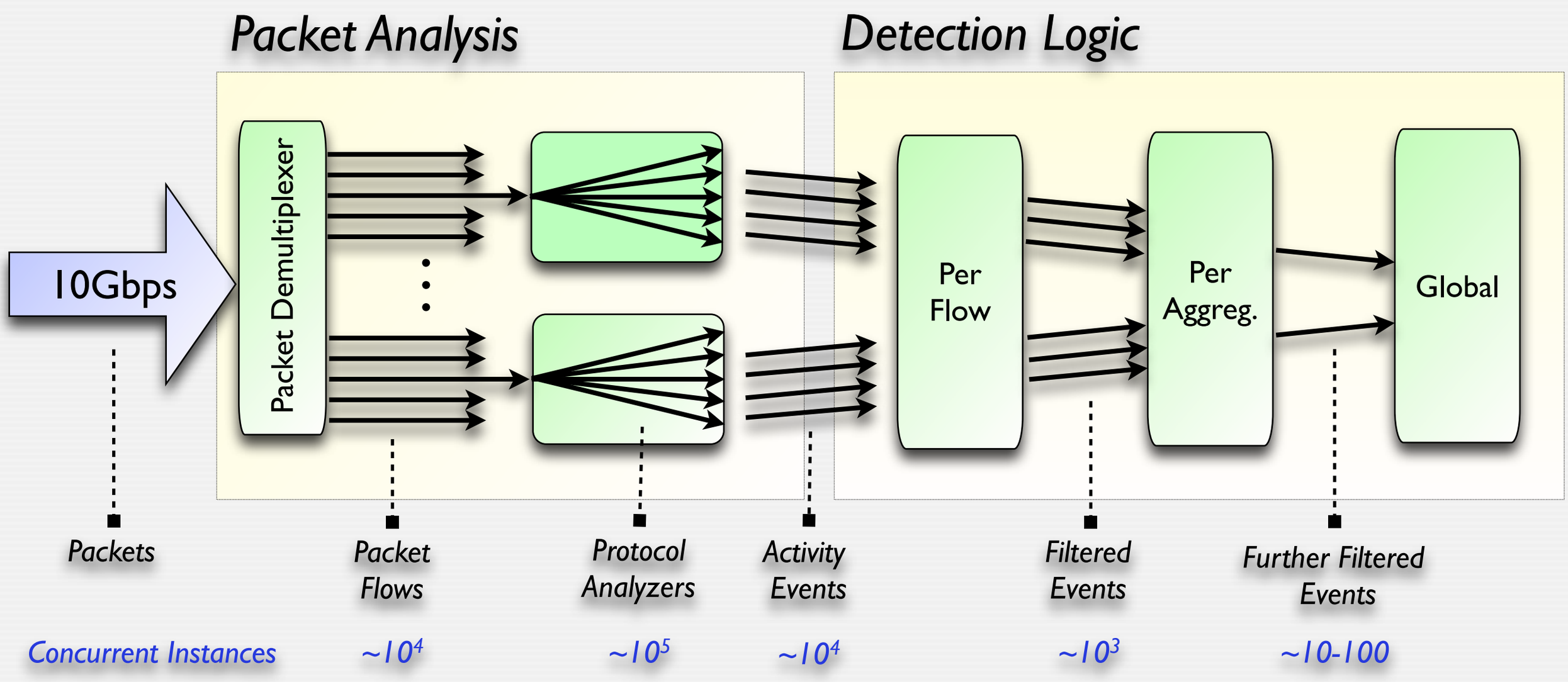
Traffic Analysis Pipeline



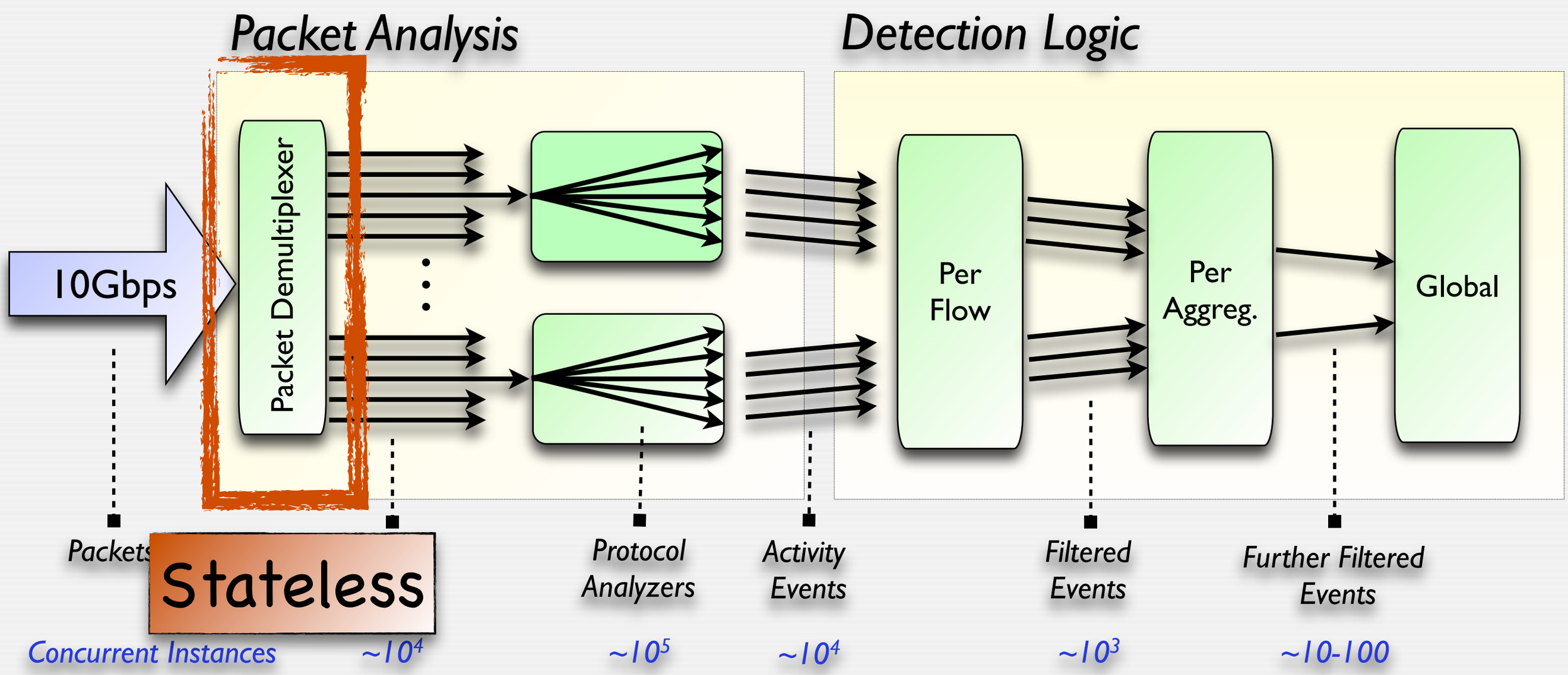
Traffic Analysis Pipeline



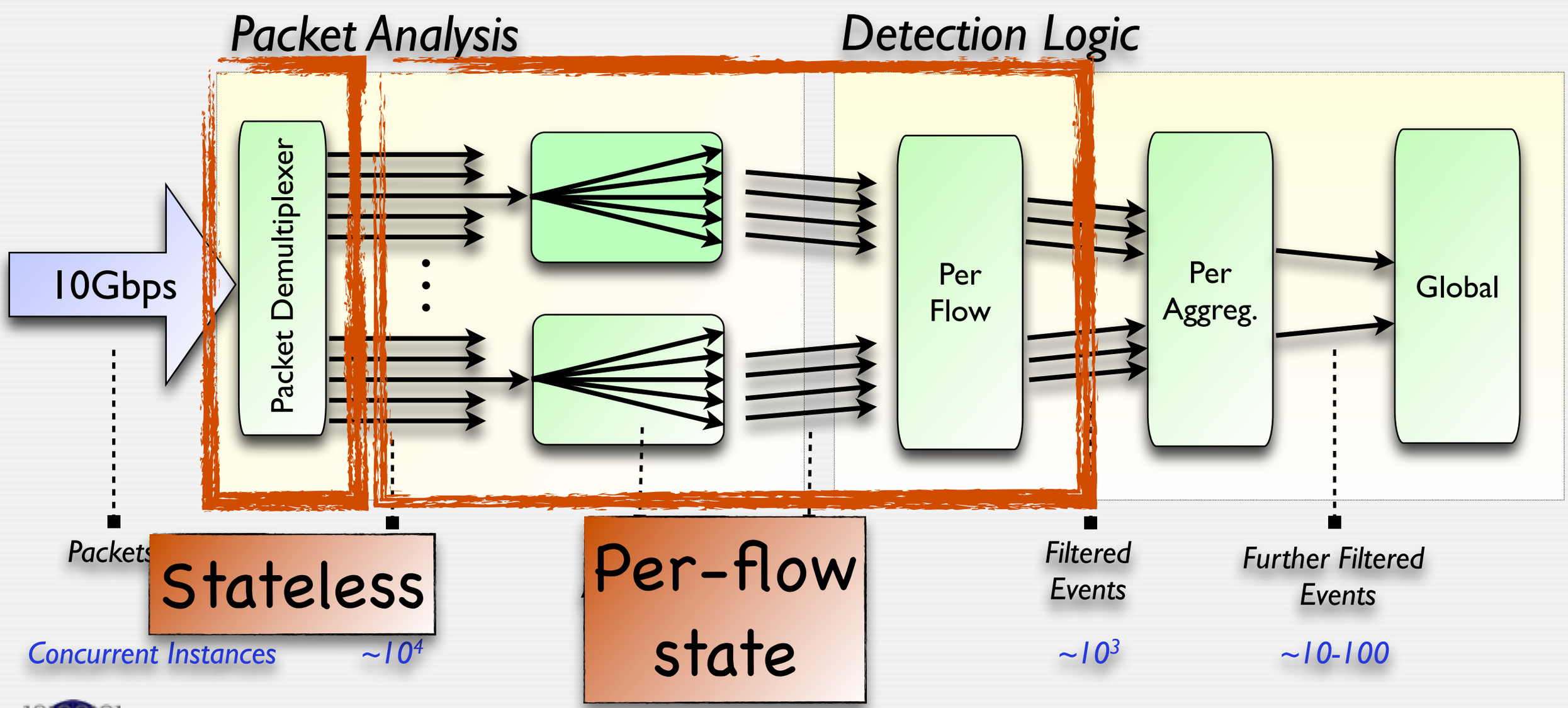
Traffic Analysis Pipeline



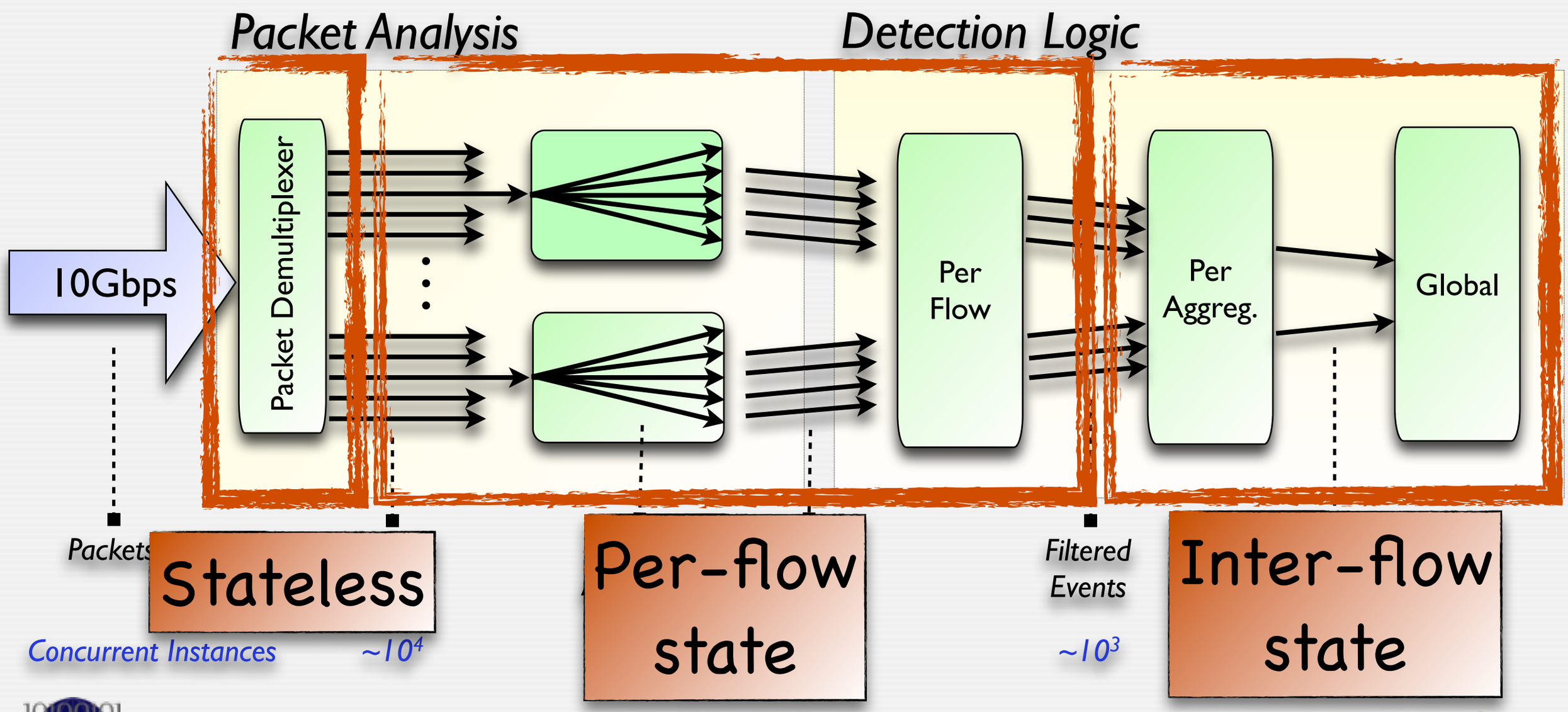
Traffic Analysis Pipeline



Traffic Analysis Pipeline



Traffic Analysis Pipeline

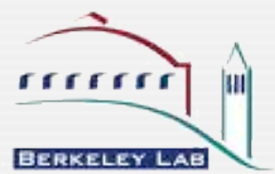


Building a Concurrent NIDS

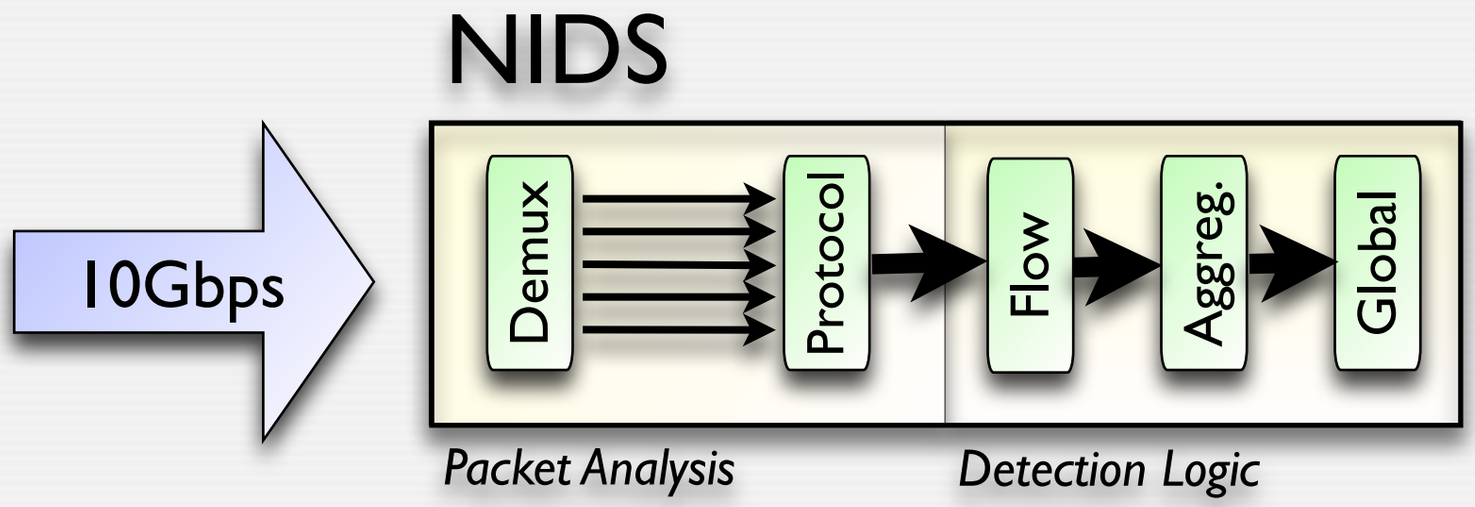
- Can Bro exploit the concurrency of the pipeline?
  - Currently single-threaded.
  - Talking about 160K lines of C++ code, plus another 25K lines of script code
- Two strategies:
  - Coarse-grained parallelism: The Bro Cluster
  - Fine-grained parallelism: Multi-core Bro

Coarse-grained Parallelism

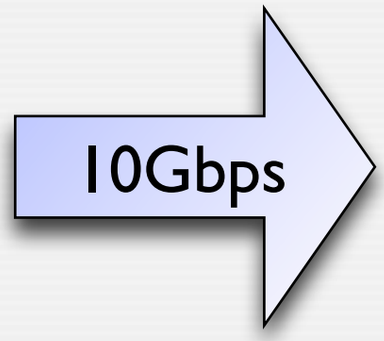
The Bro Cluster



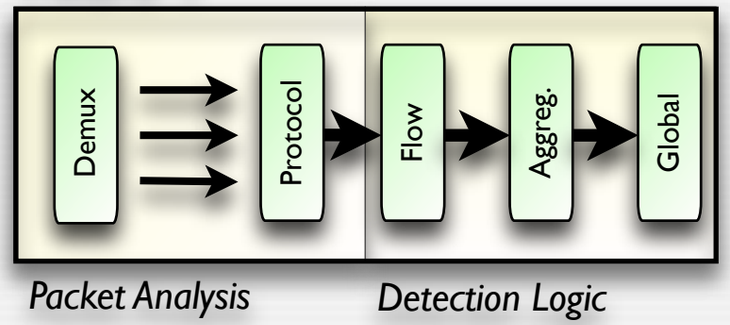
Load-Balancer Approach



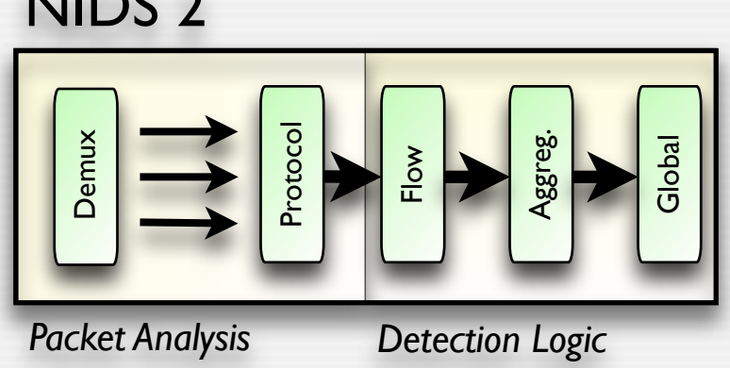
Load-Balancer Approach



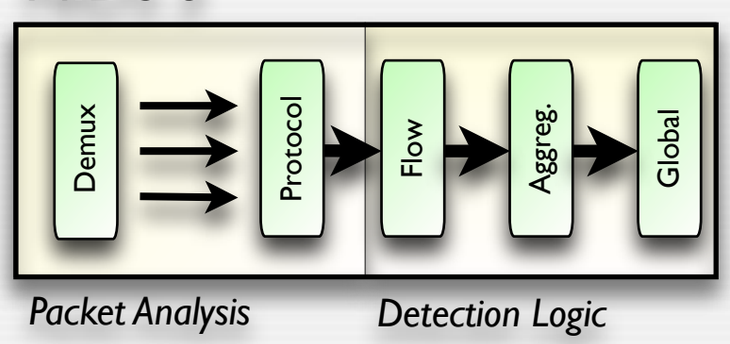
NIDS 1



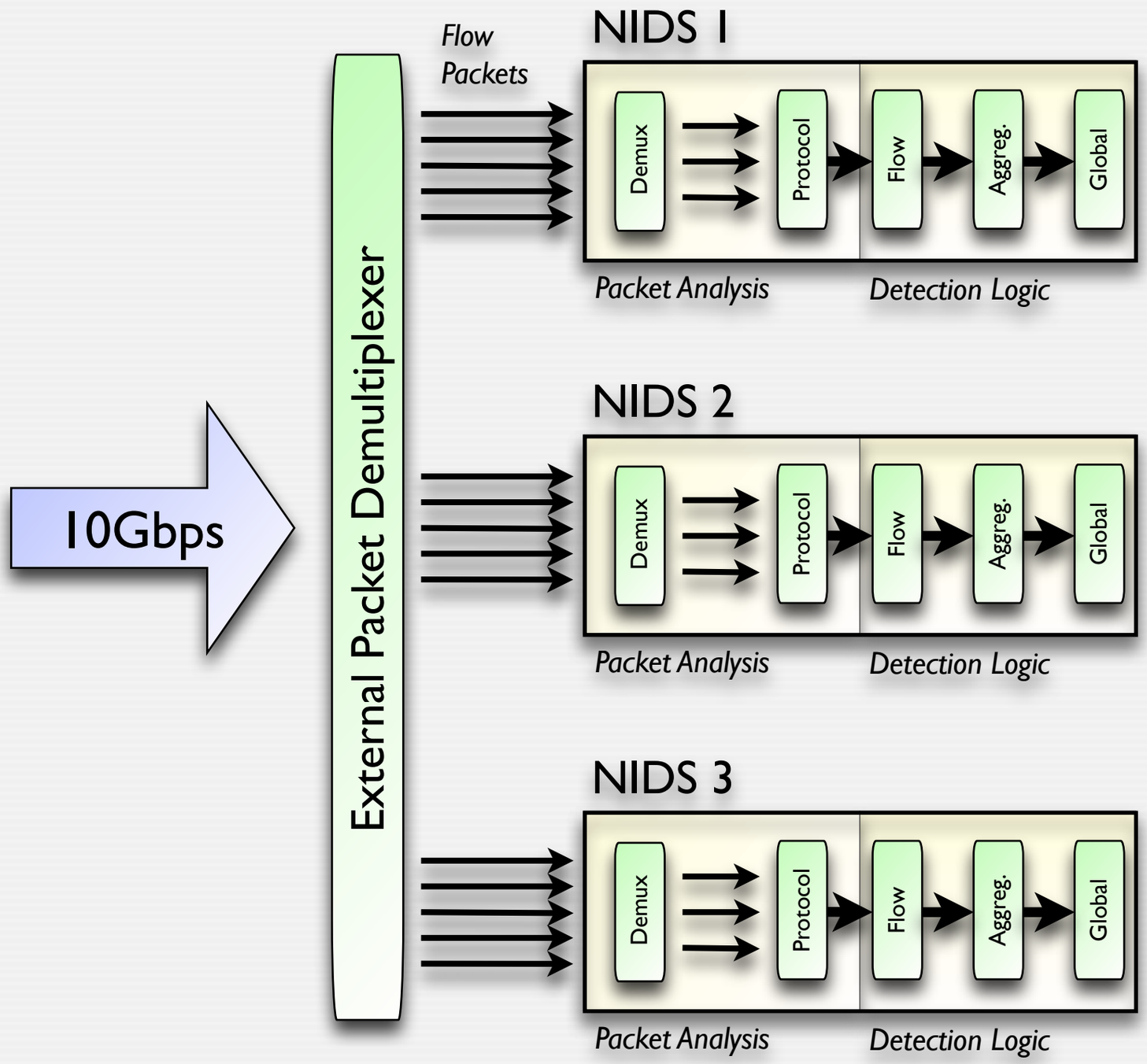
NIDS 2



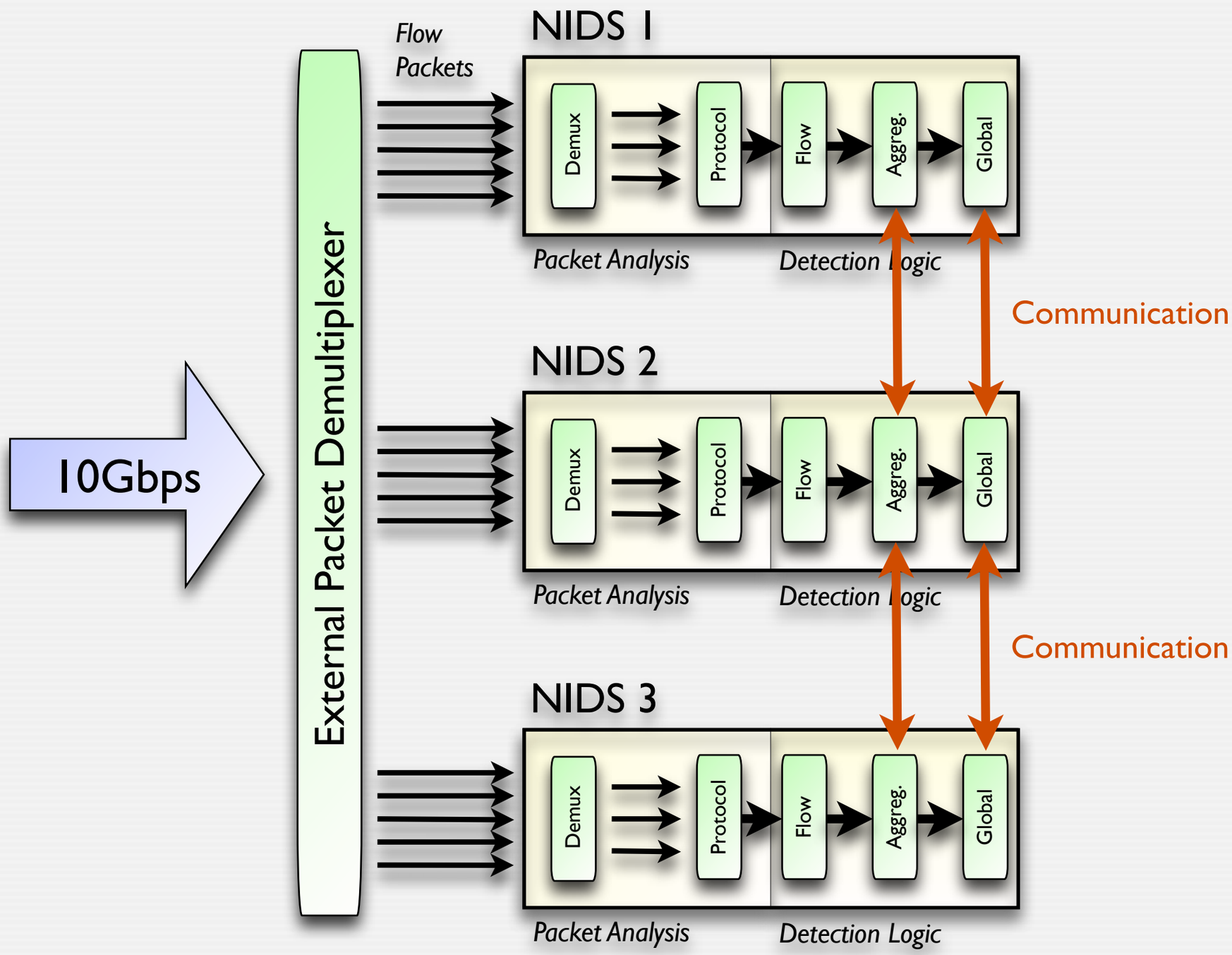
NIDS 3



Load-Balancer Approach



Load-Balancer Approach



The Bro Cluster

There are a number of practical challenges:

- Communication capability required
  Fortunately, Bro has communication primitives built-in

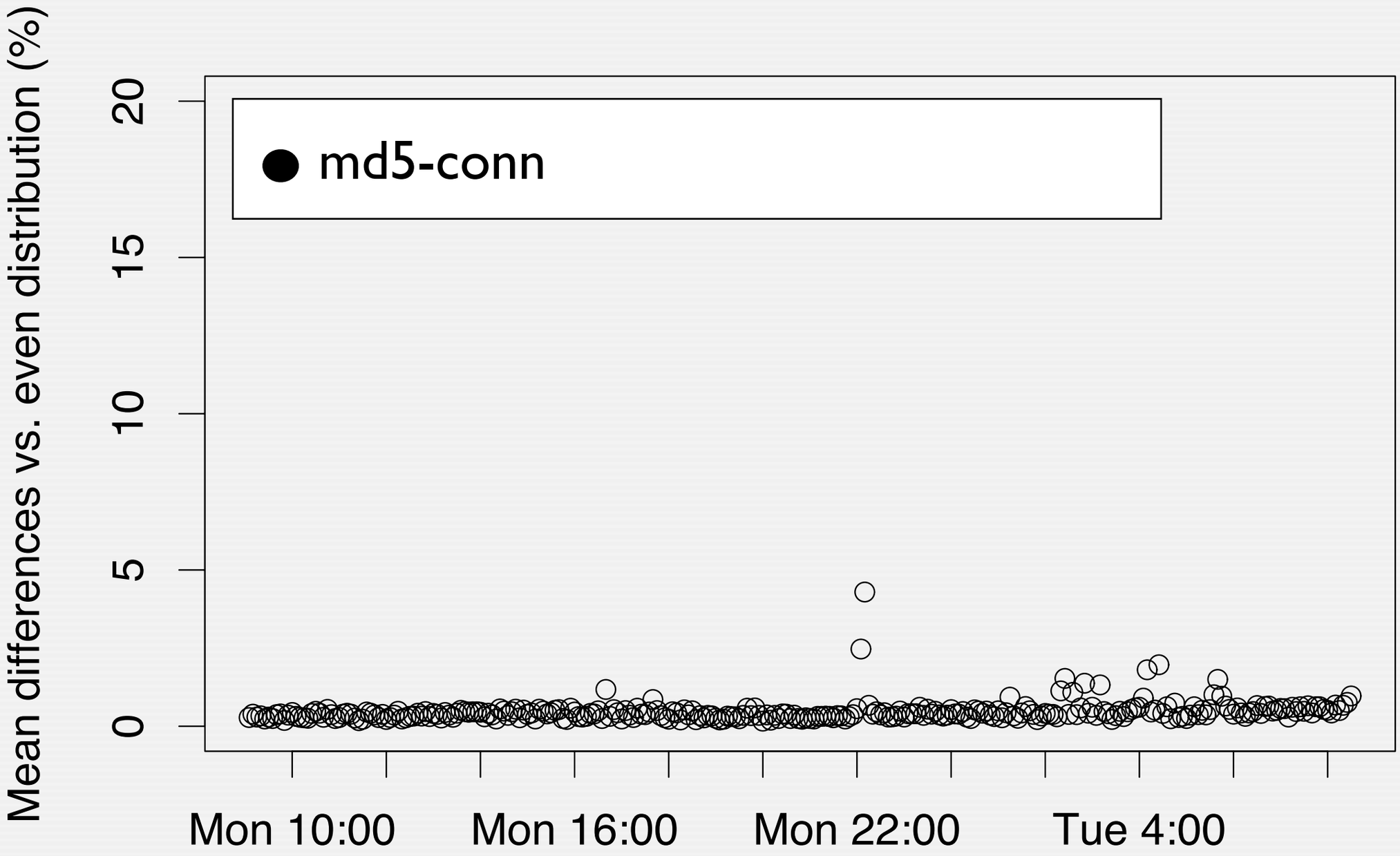
- Management of multi-machine setup is tedious
  Build a management interface transparently hiding the complexity for the operator

- External demultiplexer needs to operate at line-rate
  Worked with a vendor to build an appliance implementing our dispatching scheme

External Packet Dispatcher

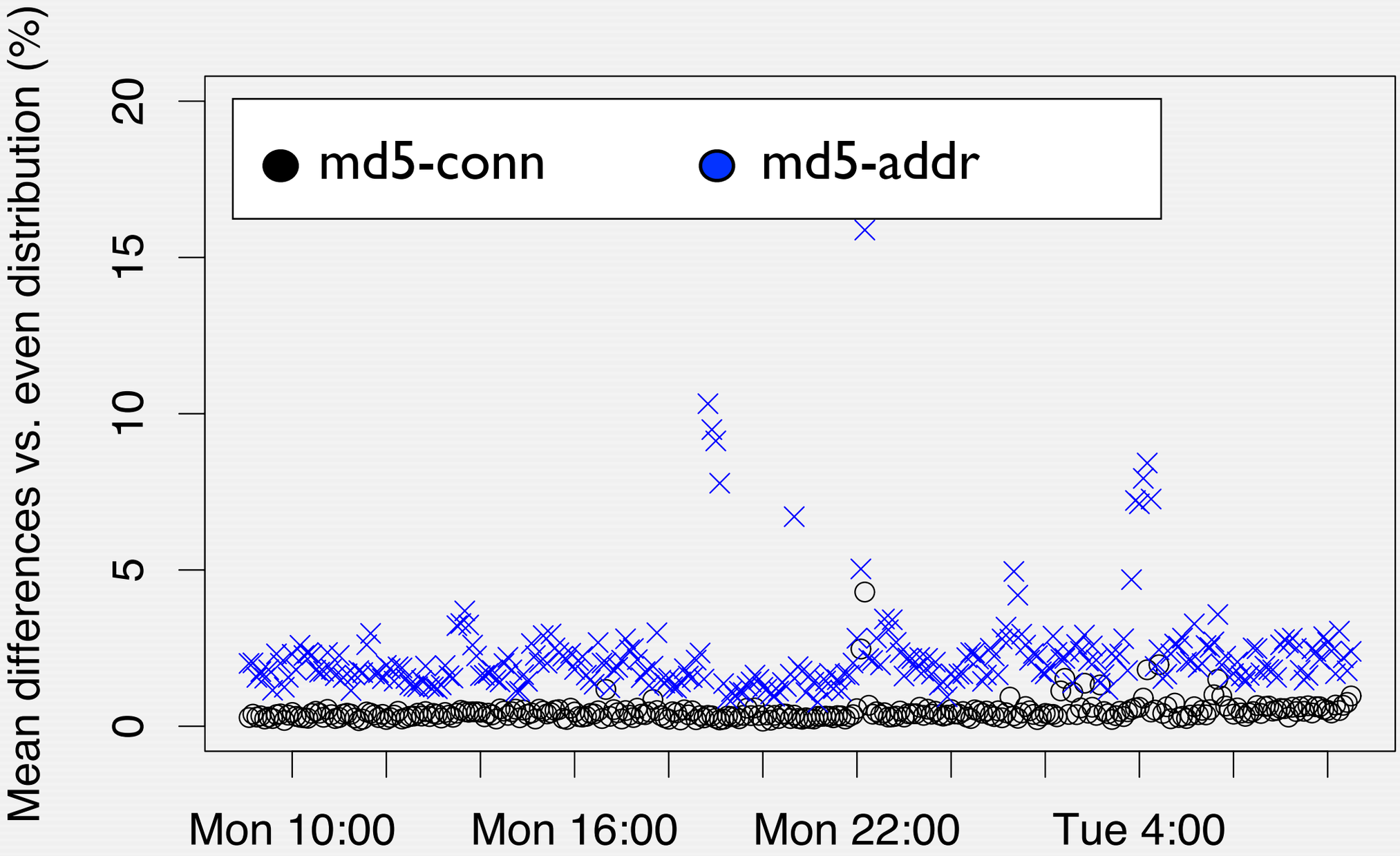
- How to decide where to send a packet?
- We want the dispatcher to
  - Keep flows together
  - Be simple and stateless for implementation in hardware
- Observation: Each packet contains a flow identifier
  - 4-tuple of IP addresses and TCP/UDP port numbers
- Dispatcher can calculate hash over the 4-tuple
  - `backend := hash(`*`tuple`*`) mod N`
- But how smooth a distribution does that yield?

Simulation of Packet Dispatcher



1 day of UC Berkeley campus TCP traffic (231M connections), n = 10

Simulation of Packet Dispatcher

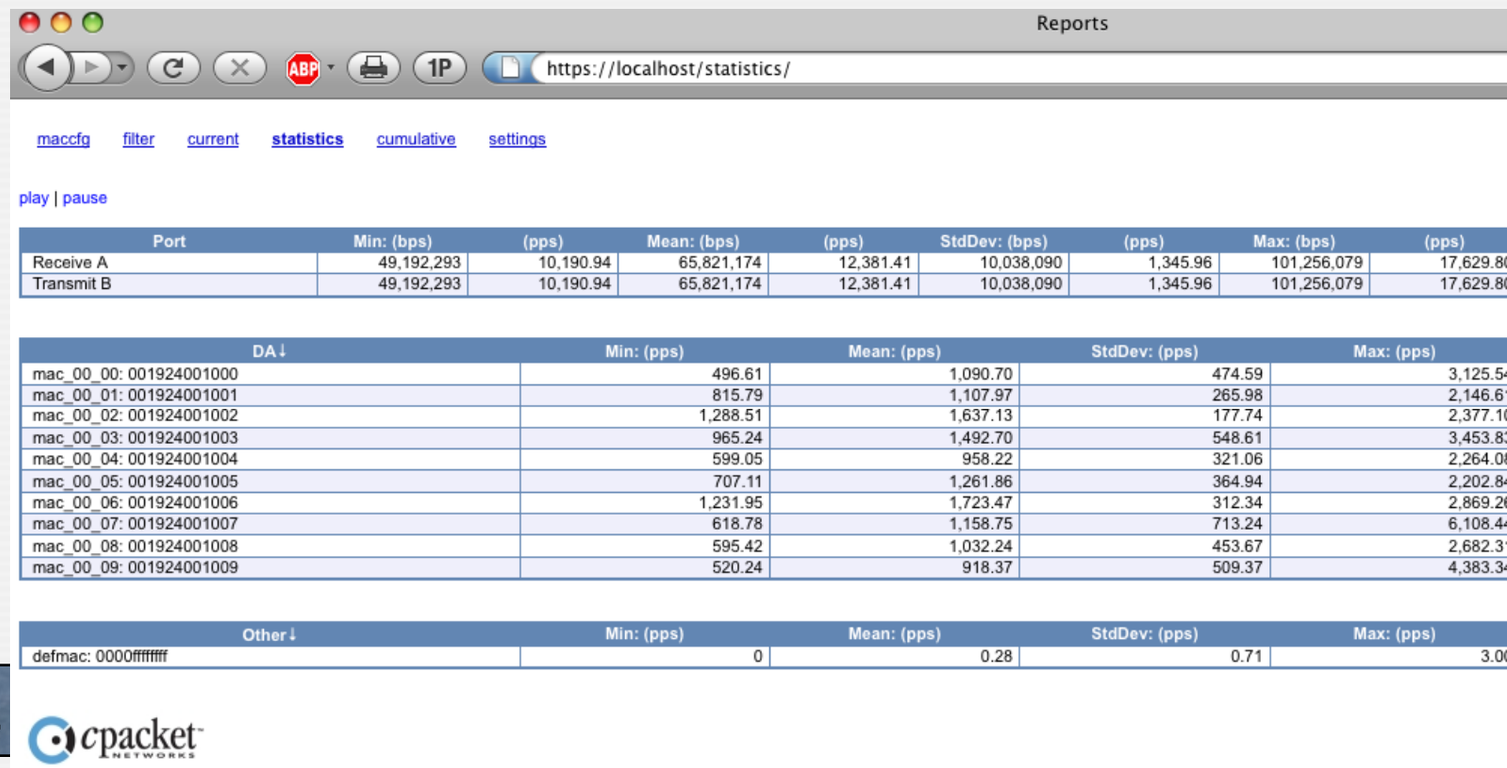


1 day of UC Berkeley campus TCP traffic (231M connections), n = 10

cFlow: A Production Load-Balancer

- LBNL worked with cPacket Networks
- cFlow: 10GE line-rate, stand-alone load-balancer

- 10GE in/out
- Web & CLI
- Filtering capabilities
- Available from cPacket



Port	Min: (bps)	(pps)	Mean: (bps)	(pps)	StdDev: (bps)	(pps)	Max: (bps)	(pps)
Receive A	49,192,293	10,190.94	65,821,174	12,381.41	10,038,090	1,345.96	101,256,079	17,629.80
Transmit B	49,192,293	10,190.94	65,821,174	12,381.41	10,038,090	1,345.96	101,256,079	17,629.80

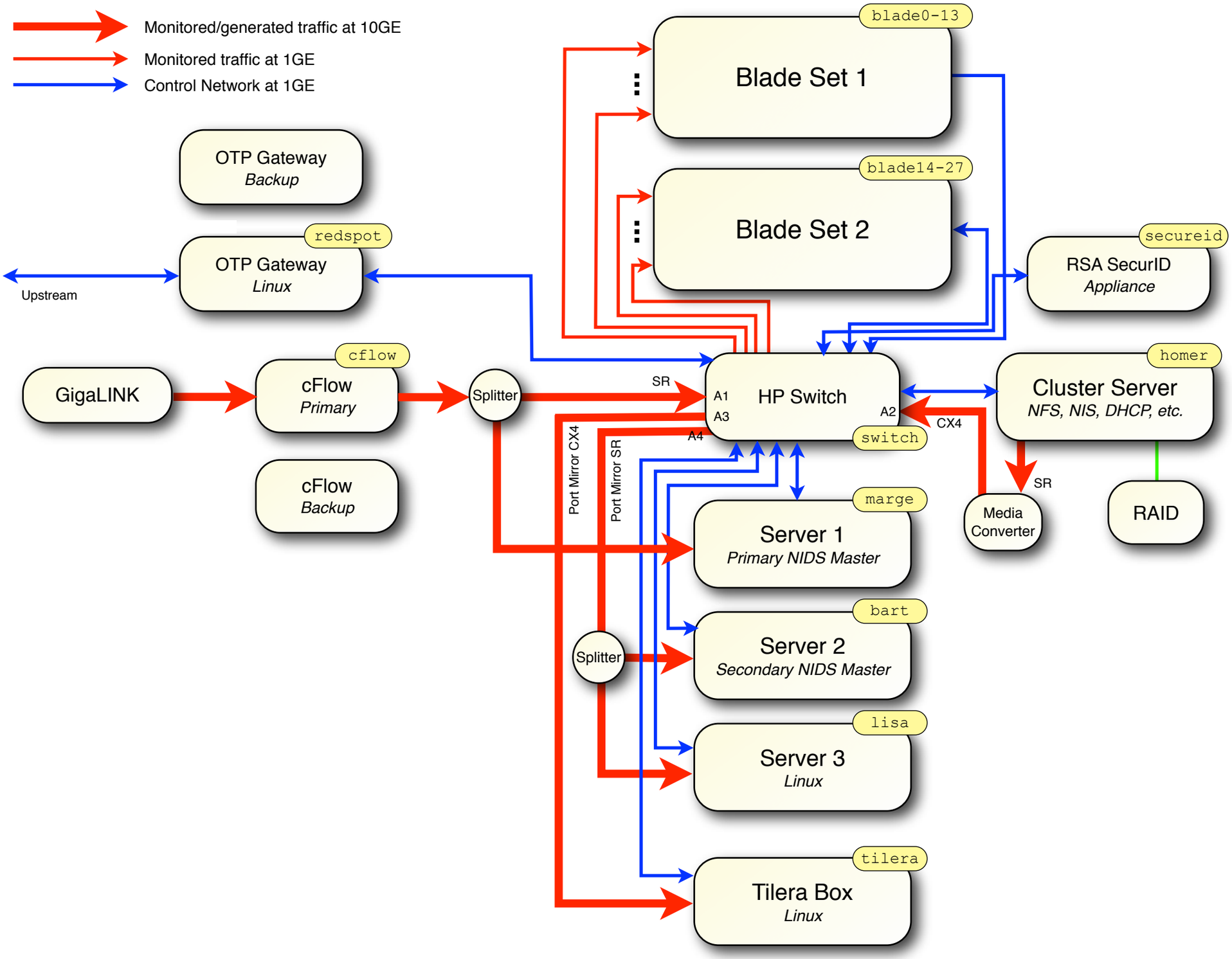
DA↓	Min: (pps)	Mean: (pps)	StdDev: (pps)	Max: (pps)
mac_00_00: 001924001000	496.61	1,090.70	474.59	3,125.54
mac_00_01: 001924001001	815.79	1,107.97	265.98	2,146.6
mac_00_02: 001924001002	1,288.51	1,637.13	177.74	2,377.1
mac_00_03: 001924001003	965.24	1,492.70	548.61	3,453.8
mac_00_04: 001924001004	599.05	958.22	321.06	2,264.0
mac_00_05: 001924001005	707.11	1,261.86	364.94	2,202.8
mac_00_06: 001924001006	1,231.95	1,723.47	312.34	2,869.2
mac_00_07: 001924001007	618.78	1,158.75	713.24	6,108.4
mac_00_08: 001924001008	595.42	1,032.24	453.67	2,682.3
mac_00_09: 001924001009	520.24	918.37	509.37	4,383.3

Other↓	Min: (pps)	Mean: (pps)	StdDev: (pps)	Max: (pps)
defmac: 0000ffffffff	0	0.28	0.71	3.00

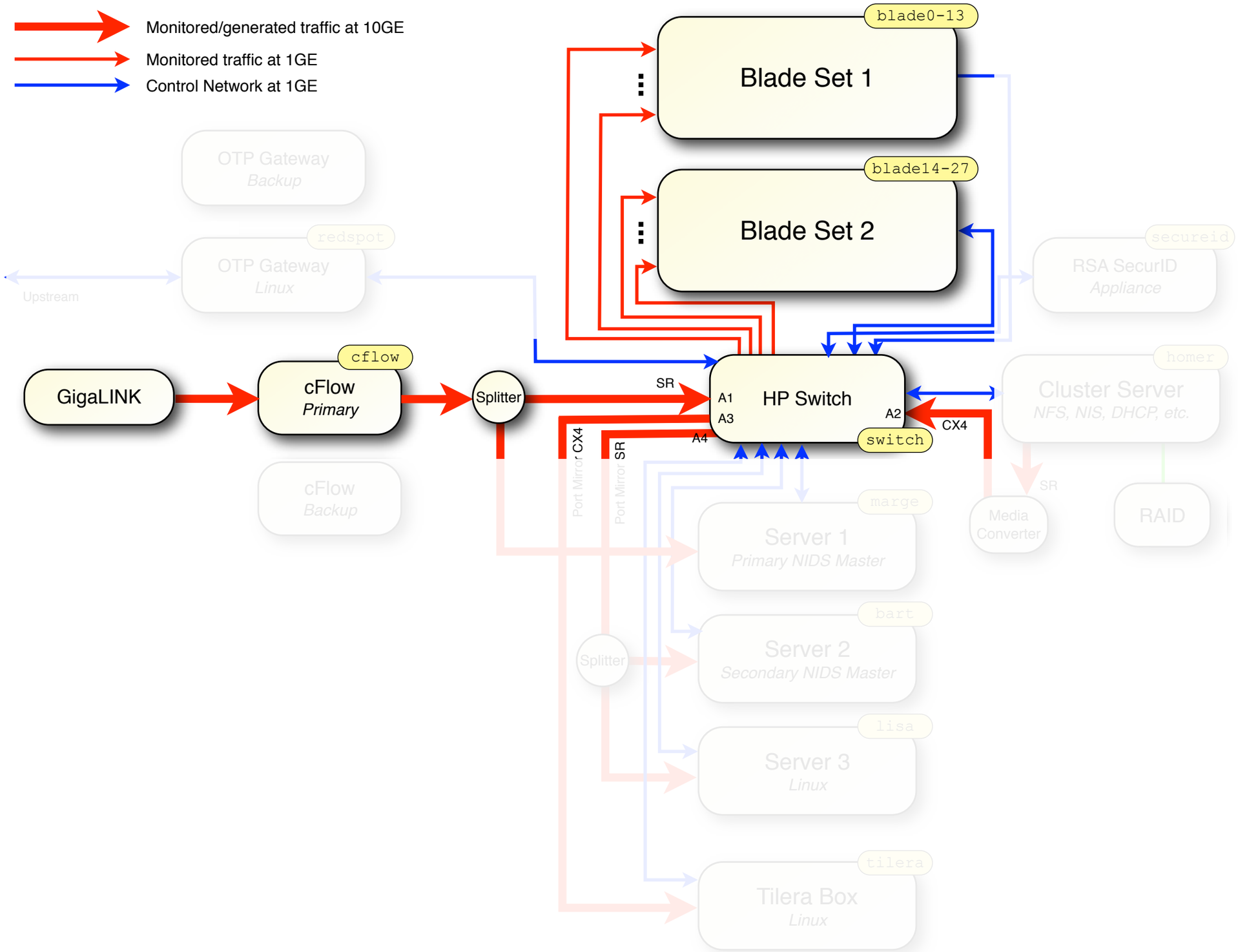
Cluster Installations

- Lawrence Berkeley National Lab
  - Medium-sized network, 15,000 hosts, 10Gbps upstream, 500-1000Mb/sec.
  - Two operational clusters at LBNL w/ 15 backends, replacing Labs' monitoring
  - We are operating a research Bro cluster with 12 individual PCs.
- UC Berkeley Campus
  - Large-scale network, 100,000 hosts, 2x10Gbps upstream, 2-3Gb/sec.
  - Just started to operate a new research cluster, with 30 individual machines.
  - Funded by NSF's *Computing Research Infrastructure* program.
- Further cluster installations in operation or planing.

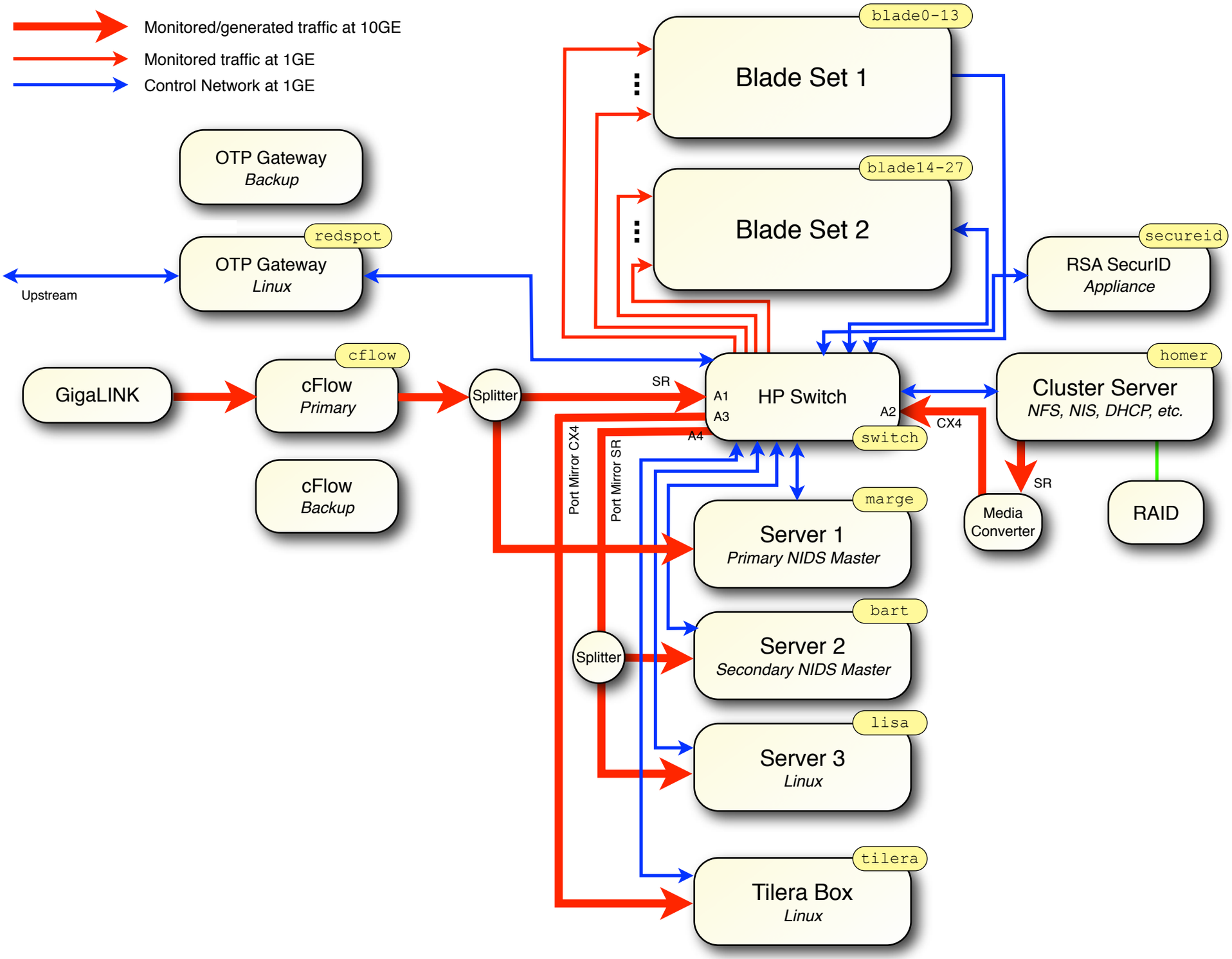
UC Berkeley Cluster



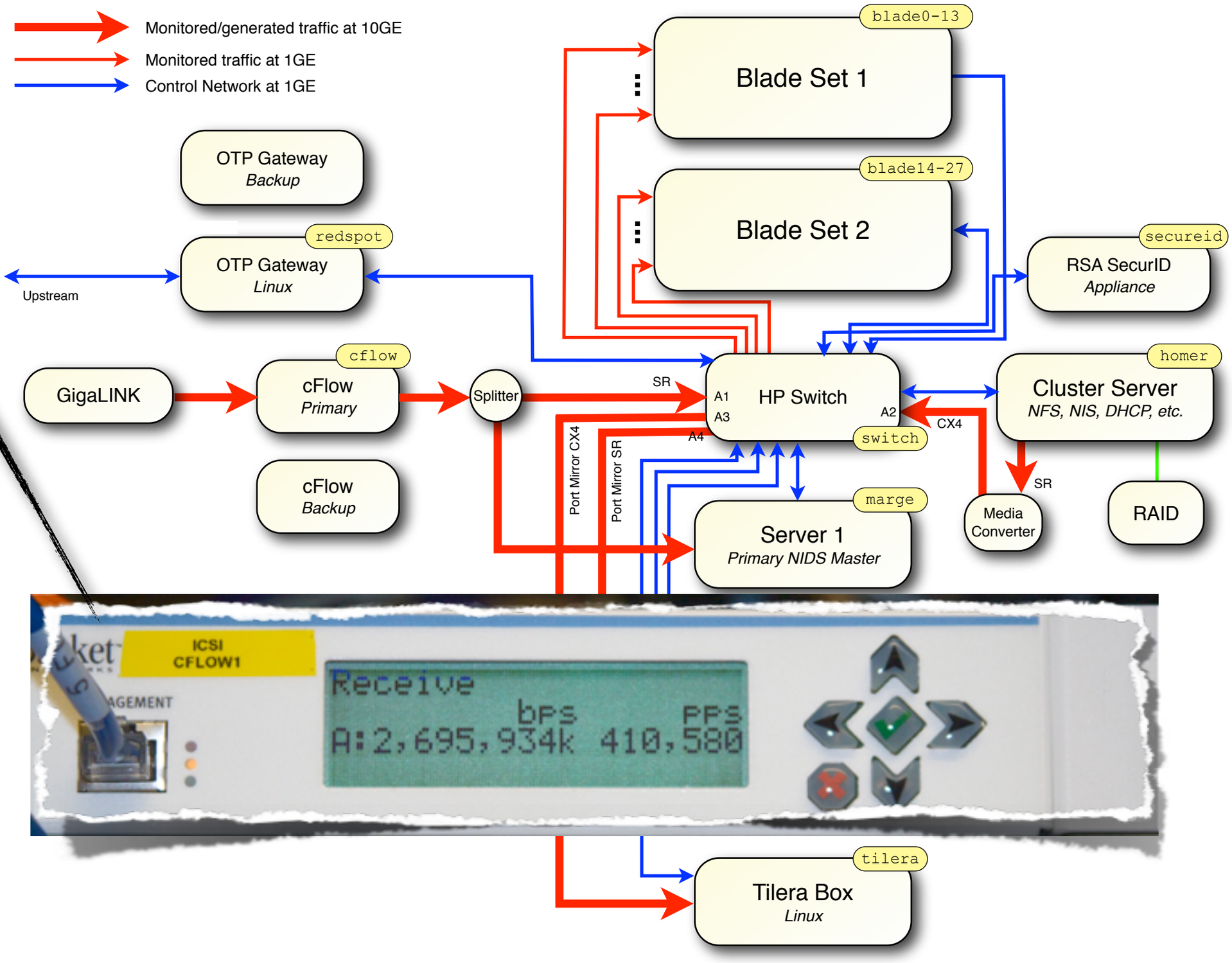
UC Berkeley Cluster



UC Berkeley Cluster



UC Berkeley Cluster

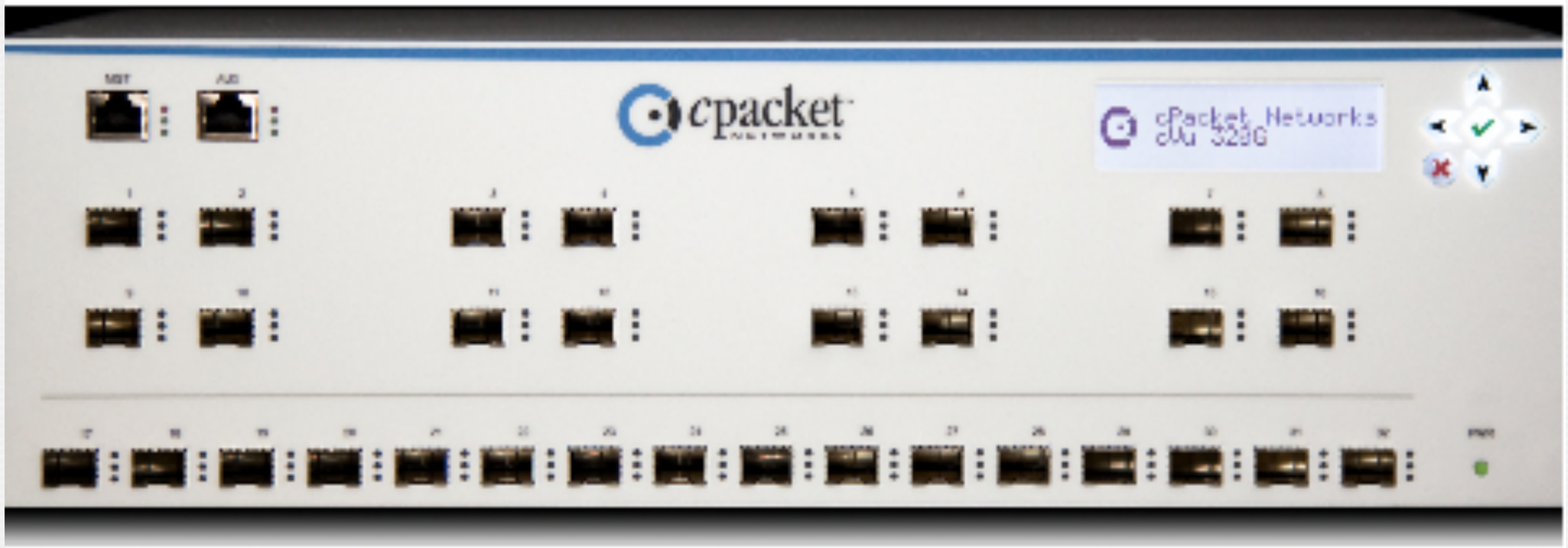


Extensions

cPacket cVu 320G

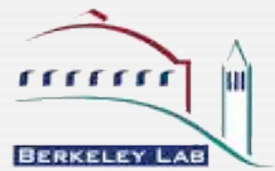
cpacket NETWORKS

32 x 10G SFP+ Traffic Monitoring Switch
Aggregation, Complete Packet Inspection Filtering, Automatic Flow Balancing



100GE(!) version is in planing ...

Fine-grained Parallelism
Building a Multi-Threaded NIDS



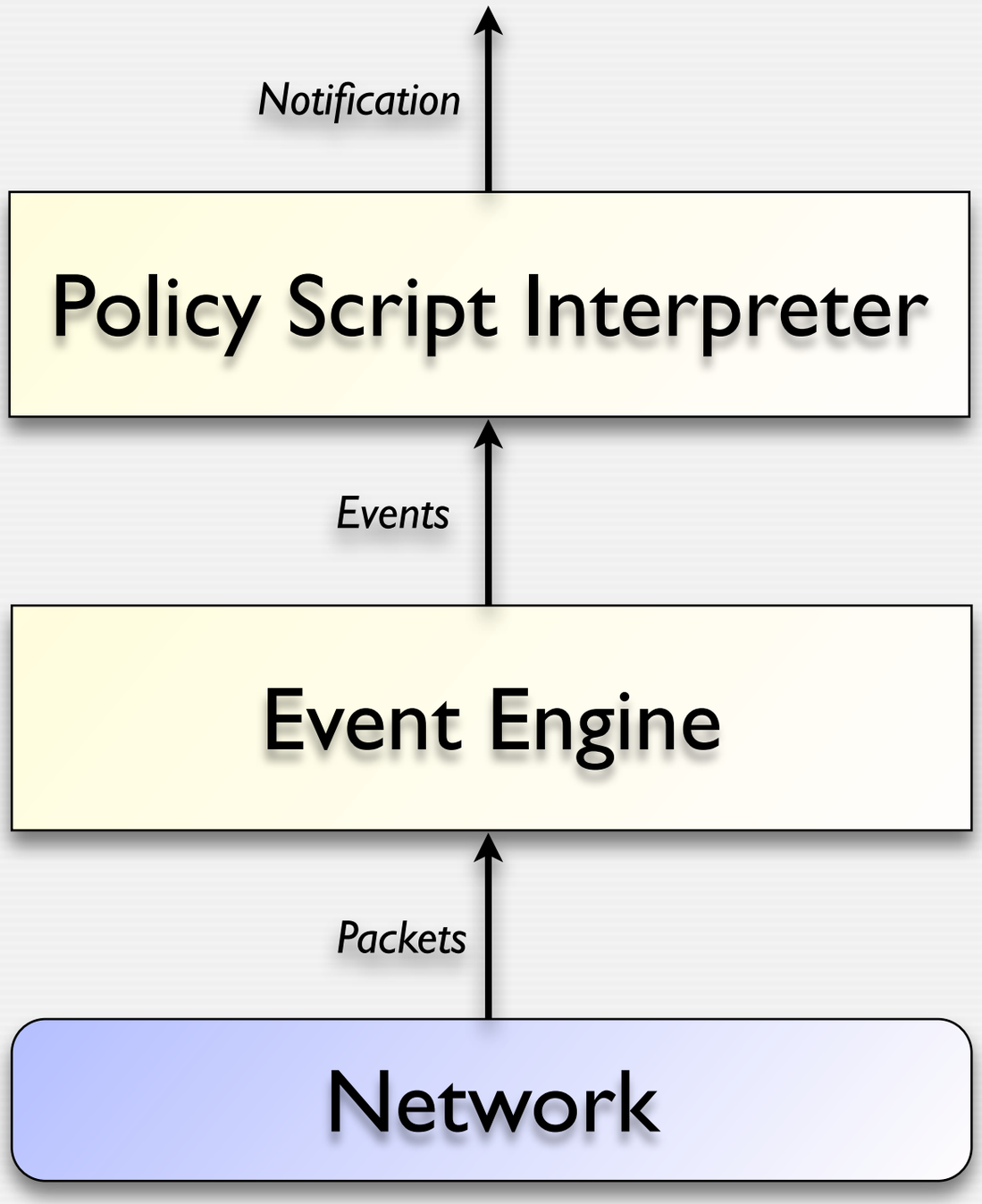
"Real" Multi-Core NIDS

- Cluster has short-comings:
  - Chances are that today's backends have multiple cores, which will be wasted
  - State is unnecessarily duplicated across all backends
  - Communication introduces race-conditions
  - Setup requires quite a bit of effort (and money)
- What we *really* want is a multi-threaded NIDS
  - ... and we want it to scale well with increasing numbers of cores
- Still don't want to write a new NIDS from scratch
  - Turn the traditional Bro into a multi-threaded application
- Main objective is doing that *transparently:*
  - Do not expose parallel processing to the operator
  - But parallelize internally "under the hood"

Bro's Architecture

Detection Logic

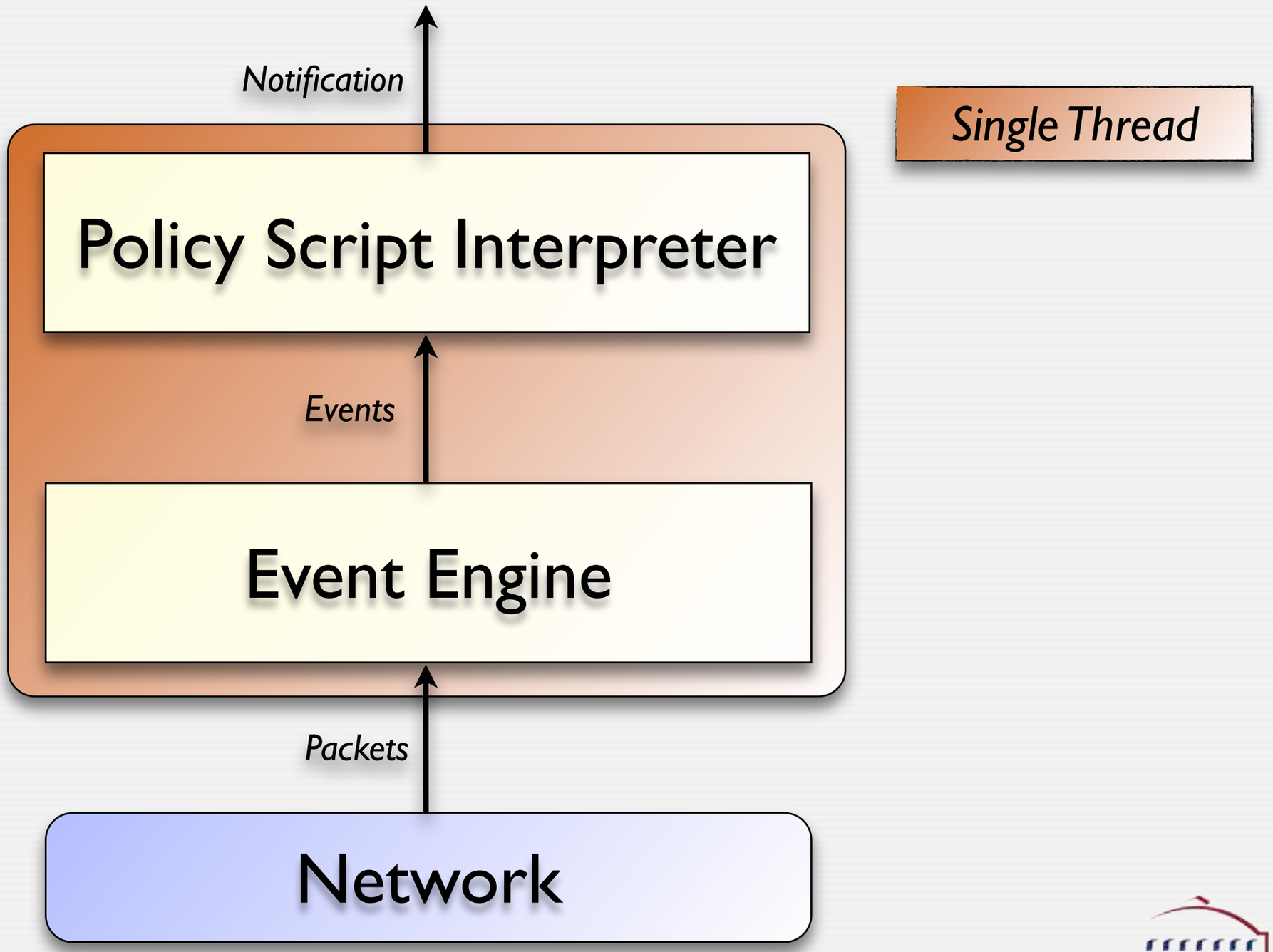
Packet Analysis



Bro's Architecture

Detection Logic

Packet Analysis



Single Thread

Bro's Architecture

Detection Logic

Notification

Policy Script Interpreter

Script Threads

Events

Event Engine

Event Engine Threads

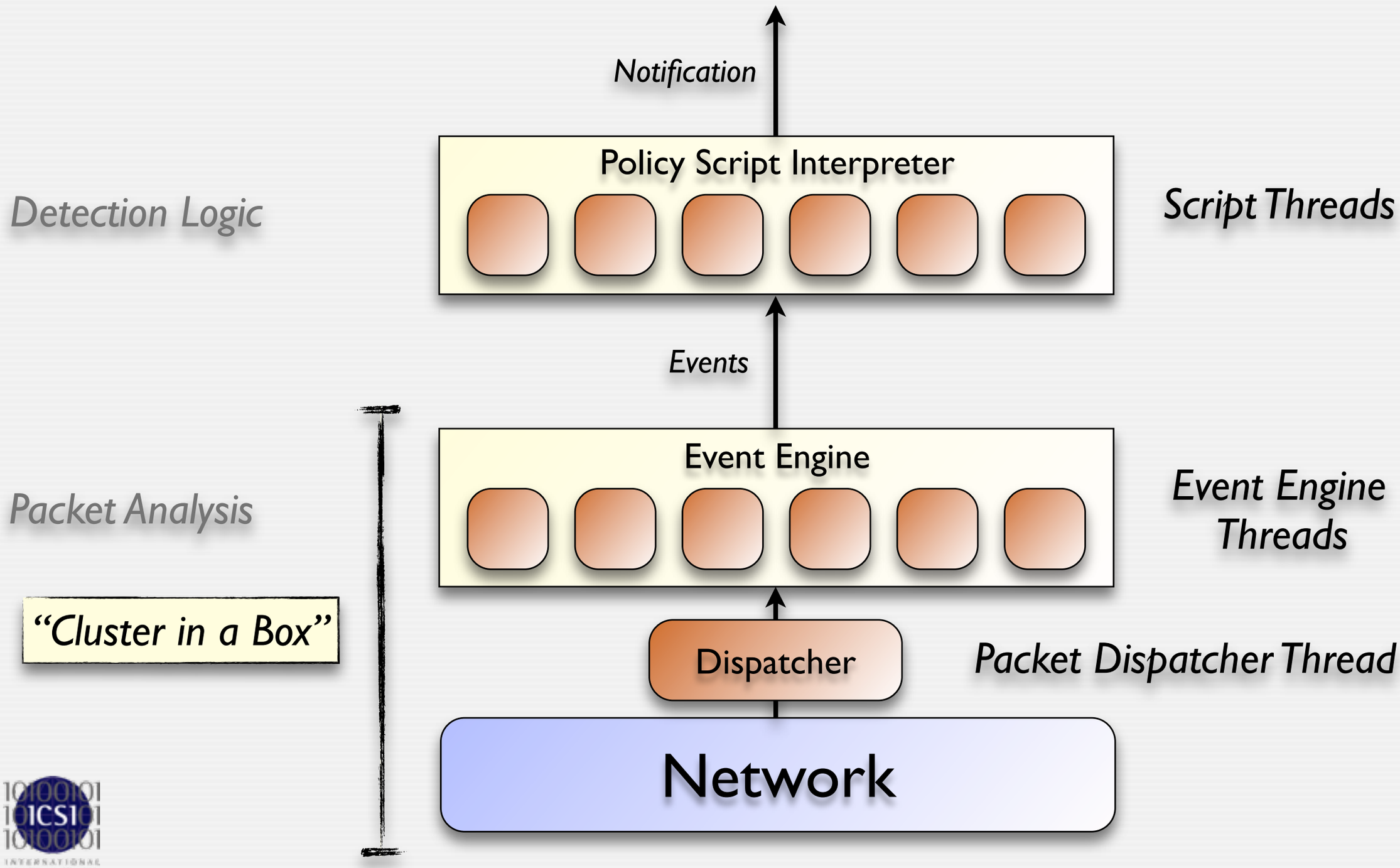
Packet Analysis

Dispatcher

Packet Dispatcher Thread

Network

Bro's Architecture



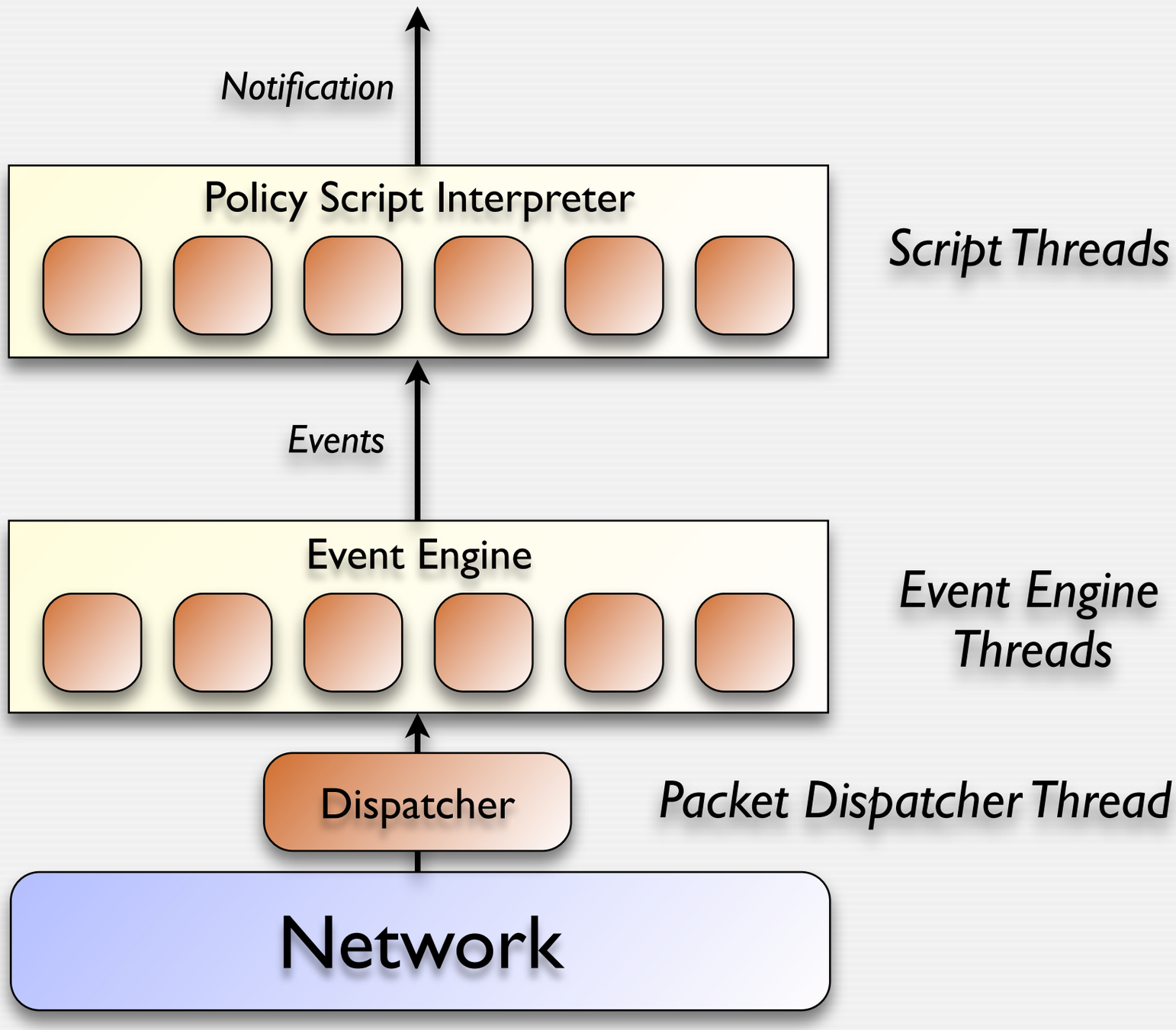
Bro's Architecture

How to parallelize a scripting language?

Detection Logic

Packet Analysis

"Cluster in a Box"



Script Threads

Event Engine Threads

Packet Dispatcher Thread

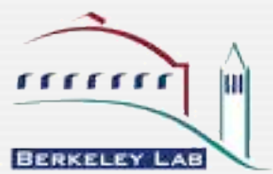
Script Example: Matching URLs

Task: Report all Web requests for files called "`passwd`".

```
event http_request(c: connection, method: string, path: string)
{
    if ( method == "GET" && path == /.*passwd/ )
        NOTICE(SensitiveURL, c, path); # Alarm.
}
```

(Syntax simplified.)

Example: ***http_request(1.2.3.4/4321⇒5.6.7.8/80, "GET", "/index.html")***



Script Example 2: Flow-based

Task: *Report all successful HTTP requests for files called "`passwd`".*

```
global potentially_sensitive: table[connection] of string;

event http_request(c: connection, method: string, path: string)
{
    if ( method == "GET" && path == /.*passwd/ )
        potentially_sensitive[c] = path; # Add to table.
}

event http_reply(c: connection, response: int, reason: string) )
{
    if ( response == OK && c in potentially_sensitive )
        NOTICE(SensitiveURL, c, potentially_sensitive[c]);
}
```

(Syntax simplified.)

Script Example 3: Aggregated

Task: *Count failed connection attempts per source address*.

```
global attempts: table[addr] of int &default=0;

event connection_rejected(c: connection)
{
    local source = c.orig_h;         # Get source address.
    local n = ++attempts[source];    # Increase counter.
    if ( n == SOME_THRESHOLD )       # Check for threshold.
        NOTICE(Scanner, source);     # If so, report.
}
```

(Syntax simplified.)

"Scheduling Scopes"

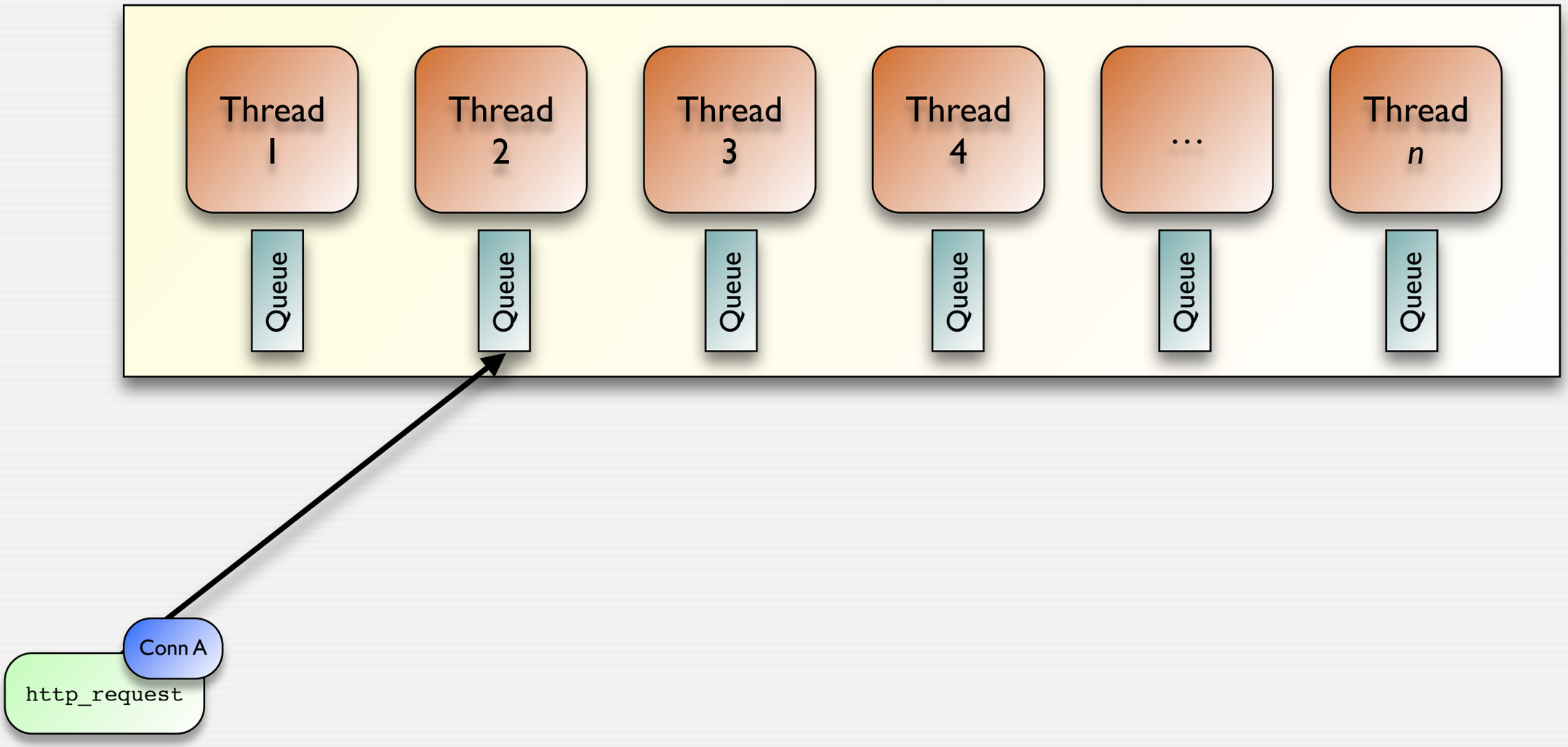
- Accessing a piece of state from only *one* thread buys us:
  - Lock-free memory accesses
  - Preservation of temporal order of event execution
- We add the concept of *scopes* to Bro's script language:
  - For each variable, one specifies the semantic granularity of accesses
    (e.g., *connection, originator, responder, host pair*)
  - All accesses with the same underlying unit will come from the same thread.
  - Internally, we keep thread-local versions of each variable
  - For each event handler, Bro derives a scope based on which variables it accesses
  - When it is scheduled, the scope & current unit determine which thread it goes to

```
global potentially_sensitive: table[connection] of string
   &scope=connection;

global attempts: table[addr] of int &default=0
   &scope=originator;
```

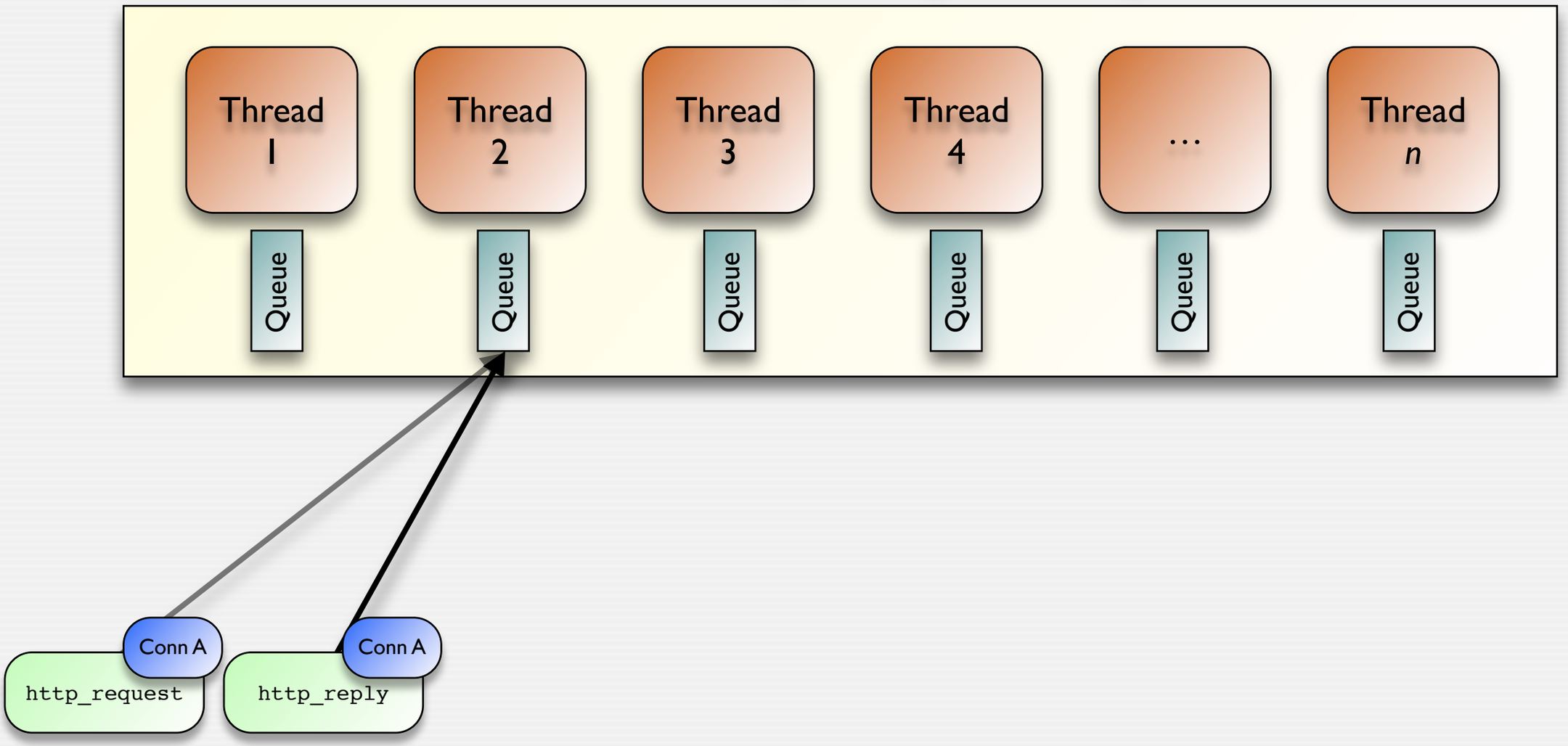
Parallel Event Scheduling

Threaded Policy Script Interpreter



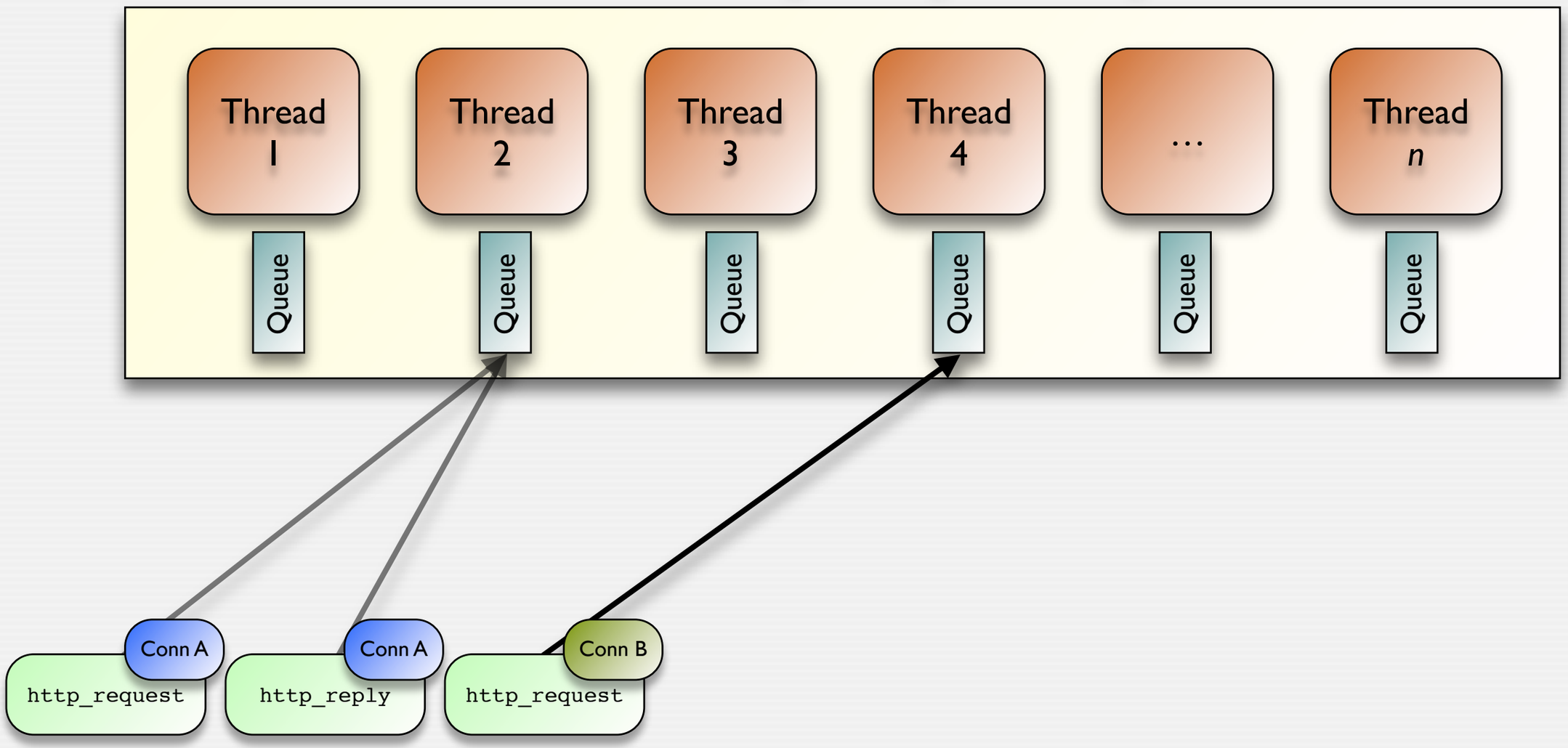
Parallel Event Scheduling

Threaded Policy Script Interpreter



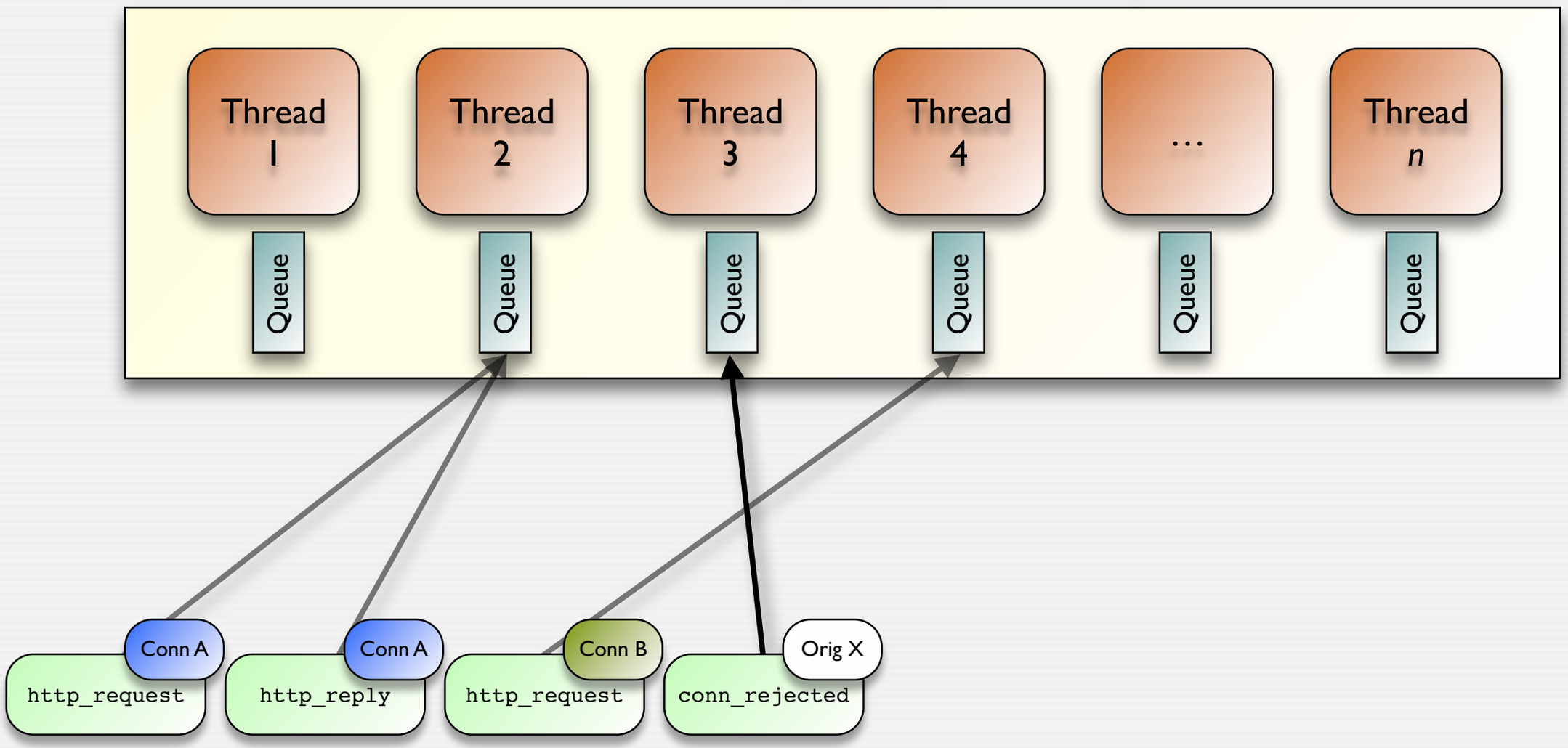
Parallel Event Scheduling

Threaded Policy Script Interpreter



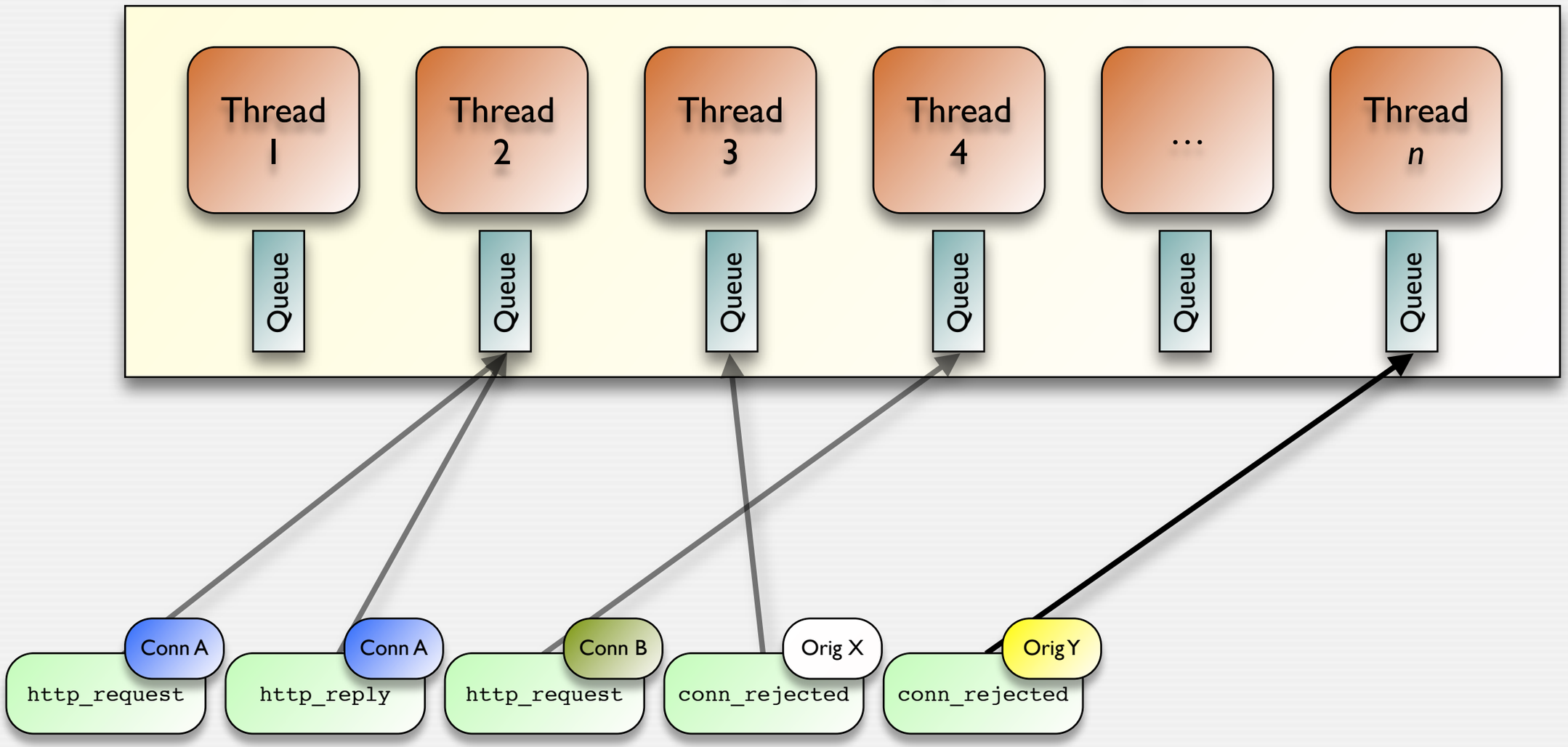
Parallel Event Scheduling

Threaded Policy Script Interpreter



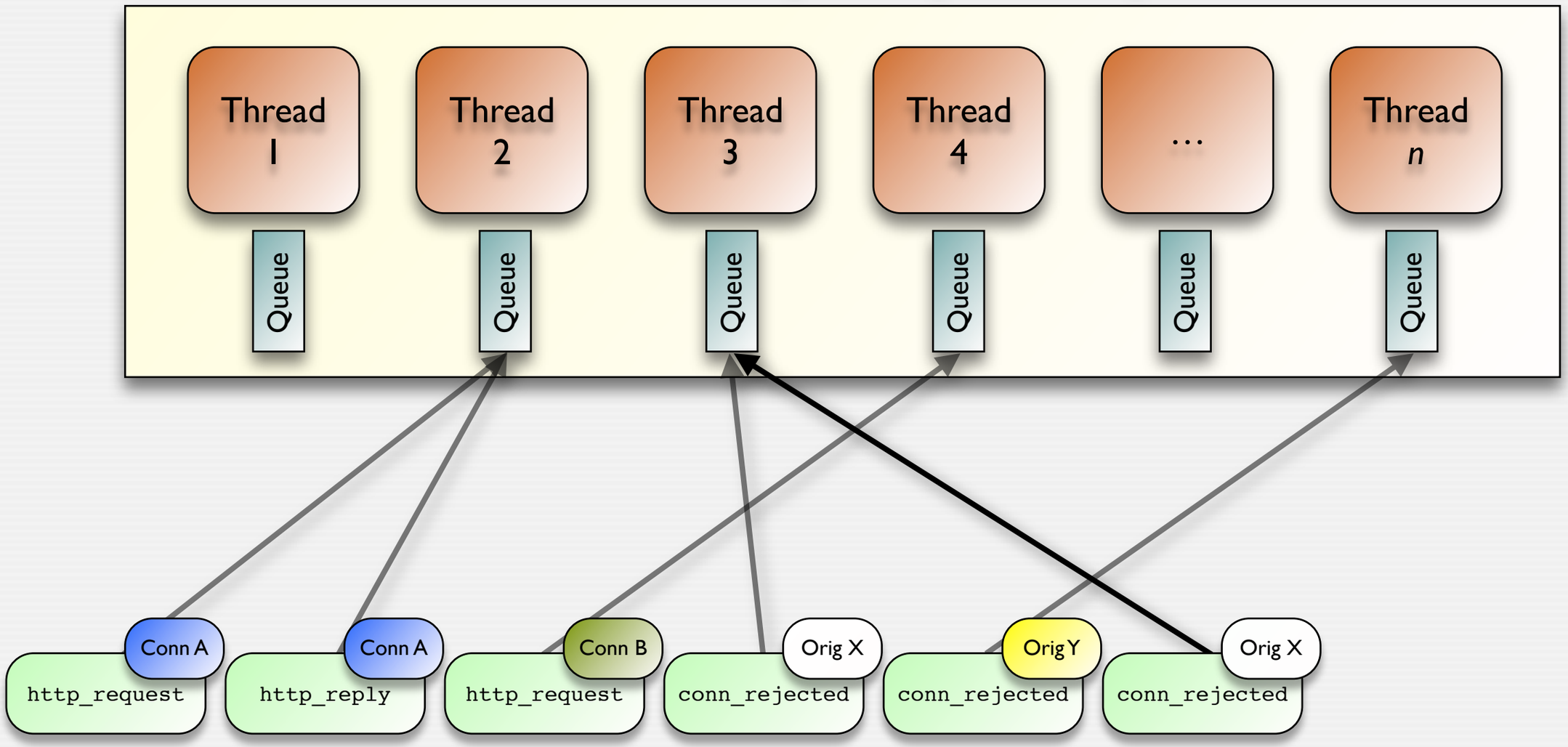
Parallel Event Scheduling

Threaded Policy Script Interpreter



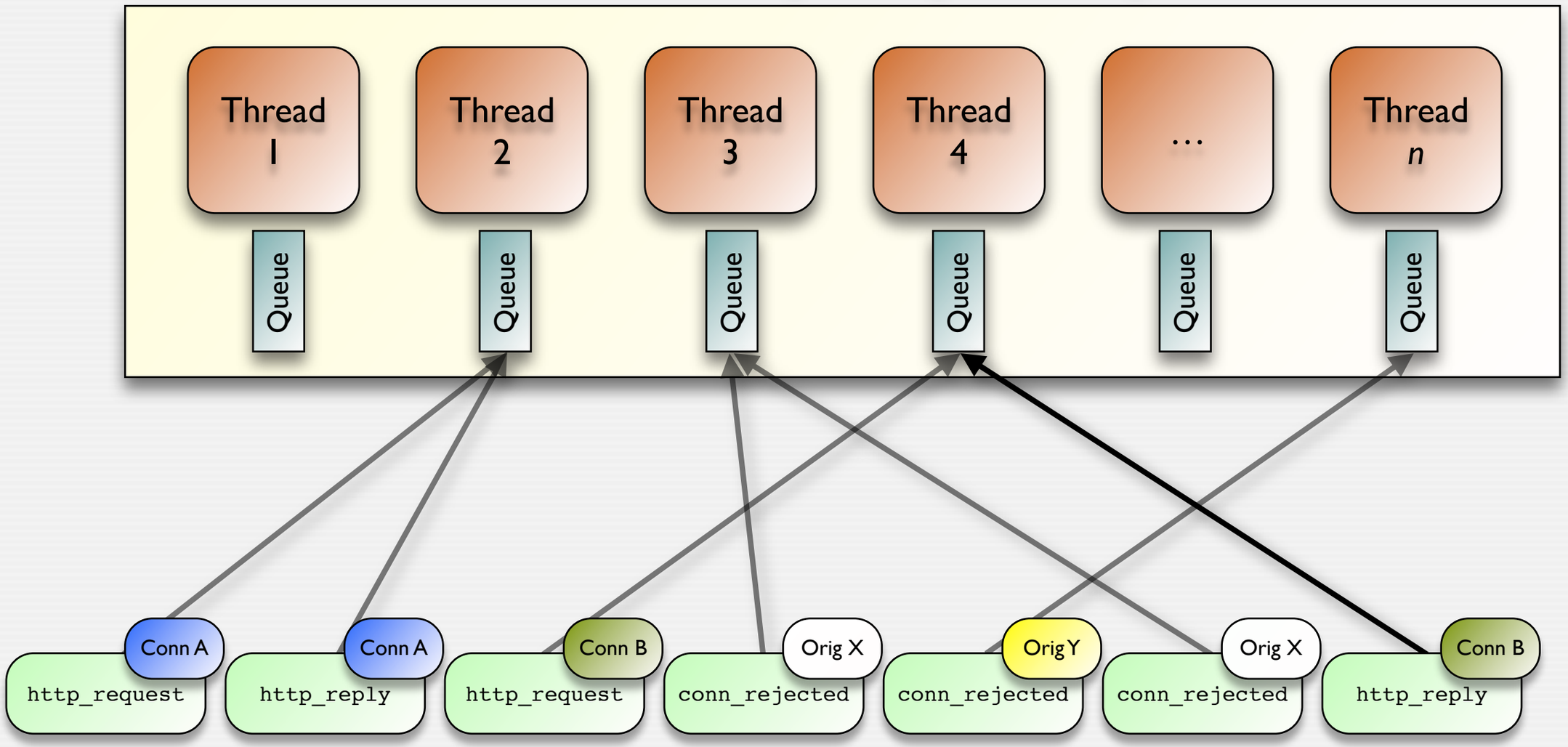
Parallel Event Scheduling

Threaded Policy Script Interpreter



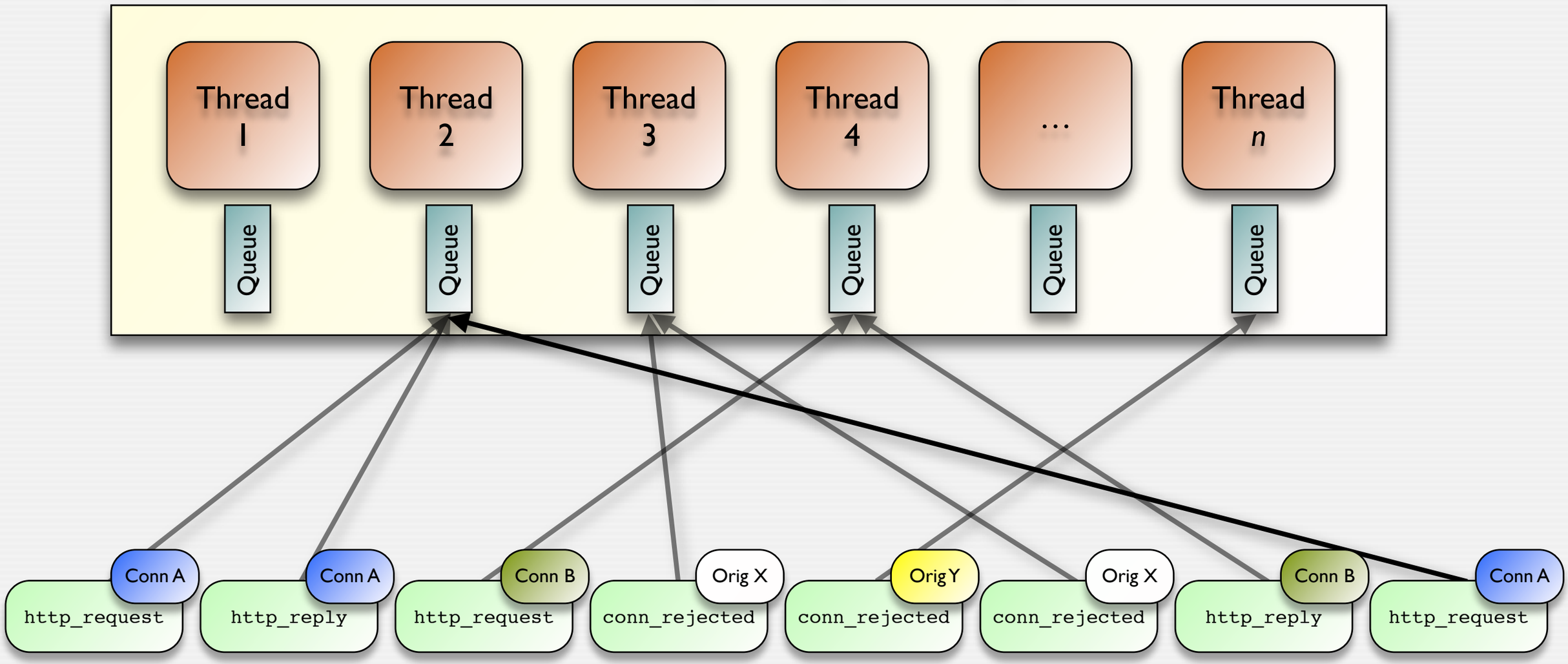
Parallel Event Scheduling

Threaded Policy Script Interpreter



Parallel Event Scheduling

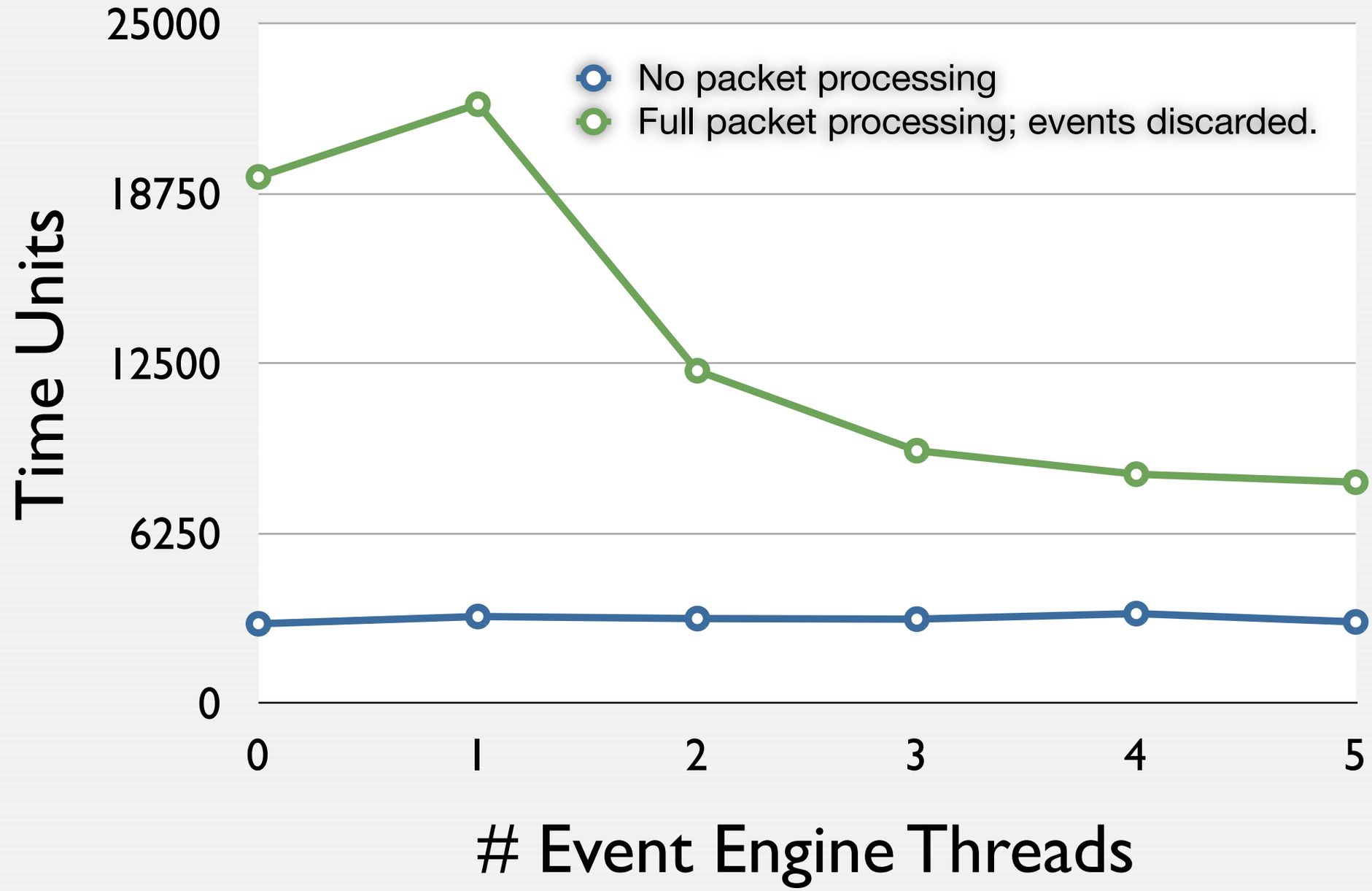
Threaded Policy Script Interpreter



Implementation of Multi-Core Bro

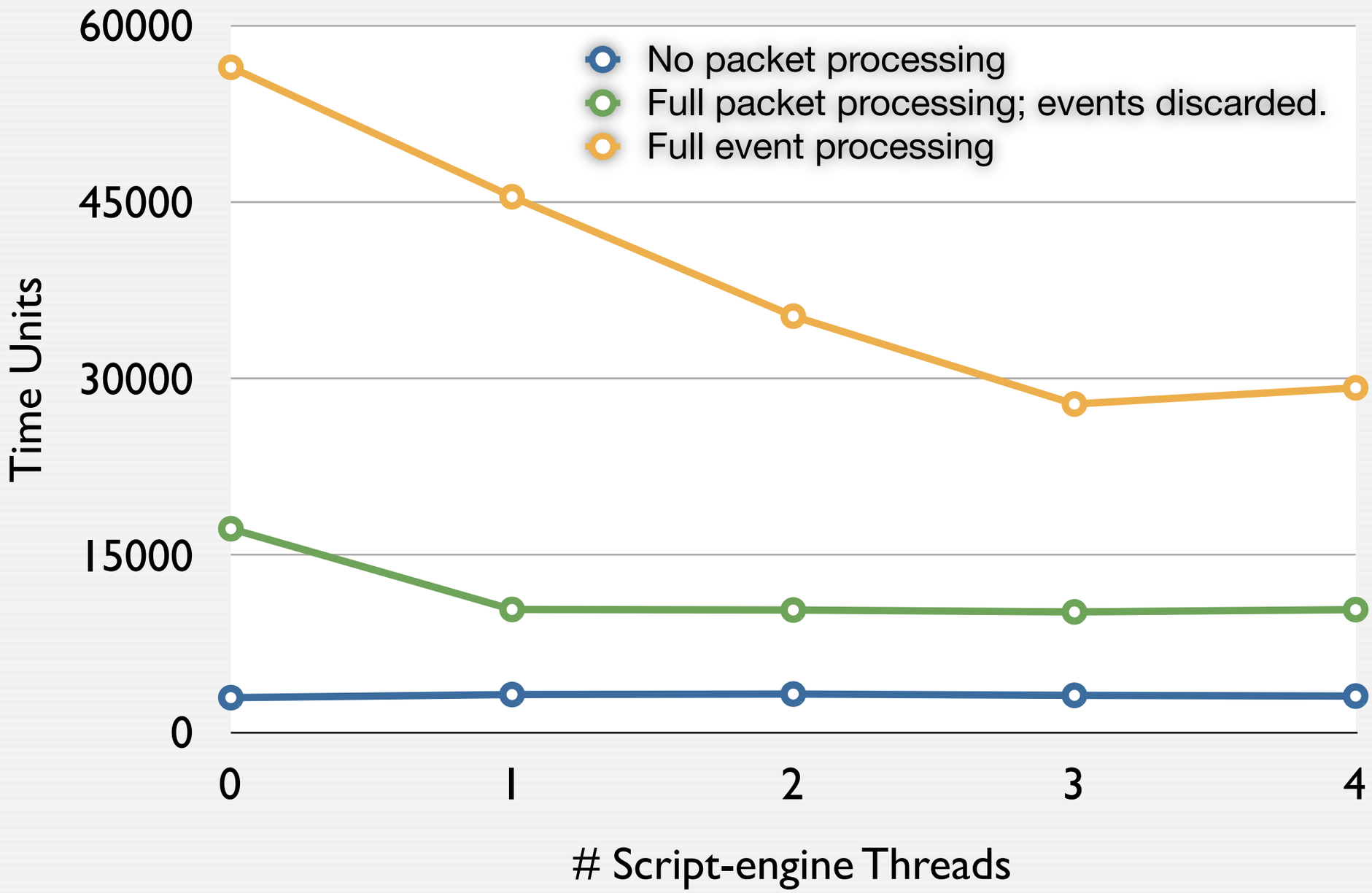
- We have a prototype that we are now profiling further
  - Parallelization based on Intel's Threading Building Blocks
  - We do not use TBB's task concept; just a portable thread abstraction
- Spent a lot of time in making Bro's code thread-safe
  - Extensive use of globals and statics ...
  - Race conditions, e.g., in memory management
  - Added lots of debugging code
  - Not pretty ...
- Assigned scopes to the most important script globals
  - Profiling showed which global variables are accessed the most (>100)
  - Surprisingly many are covered with a small set of scopes
  - Some minor script adaptations to observe scoping rules
  - "Real" globals are fully locked

Event Engine Performance So Far ...



Dual quad-core Xeon system with 10min/11GB of LBL traffic
1 script-engine thread

Script-Engine Performance So Far ...



Dual quad-core Xeon system with 10min/11GB of LBL traffic
2 event-engine threads

Future Directions

Outline

1. *Overview of the Bro Network Intrusion Detection System*
   - Philosophy
   - Deployment
   - Architecture and Usage
   - Specific Capability: Dynamic Protocol Detection
2. *High Performance with Concurrent Traffic Analysis*
   - *Concurrency Potential*
   - *Coarse-grained Parallelism: A cluster for load-balancing*
   - *Fine-grained Parallelism: Designing a multi-threaded NIDS*
3. *Future Directions*

Understanding the Bottlenecks

- Concurrent execution is just the start
  - "Just" parallelizing execution does not necessarily yield the expected speed-up
- Extensive profiling and optimization
  - CPU usage
  - Impact of the memory hierarchy
  - Explore different scheduling strategies
  - Use well-defined input traffic to understand the effects.
  - Offline vs. online
- Evaluation on different hardware platforms
  - Commodity systems with dual quad-core Xeons (54xx vs. 55xx)
  - 64-core Tilera platform
  - Simics simulator to explore a wide variety of options
- Parallelizing the packet dispatcher
  - Several vendors provides platforms that directly schedule packets to threads
  - Tilera can do that as well

Automating the Scoping

- Scopes are assigned manually in our prototype
  - Not ideal, as it's not completely transparent to the user
- Bro should be able to infer scopes automatcially
  - Static and dynamic analysis of access patterns

- Scoping rules require minor script modifications
- Likewise, Bro could rewrite code internally
  - For example, auto-split event handlers

Building an Abstract Machine

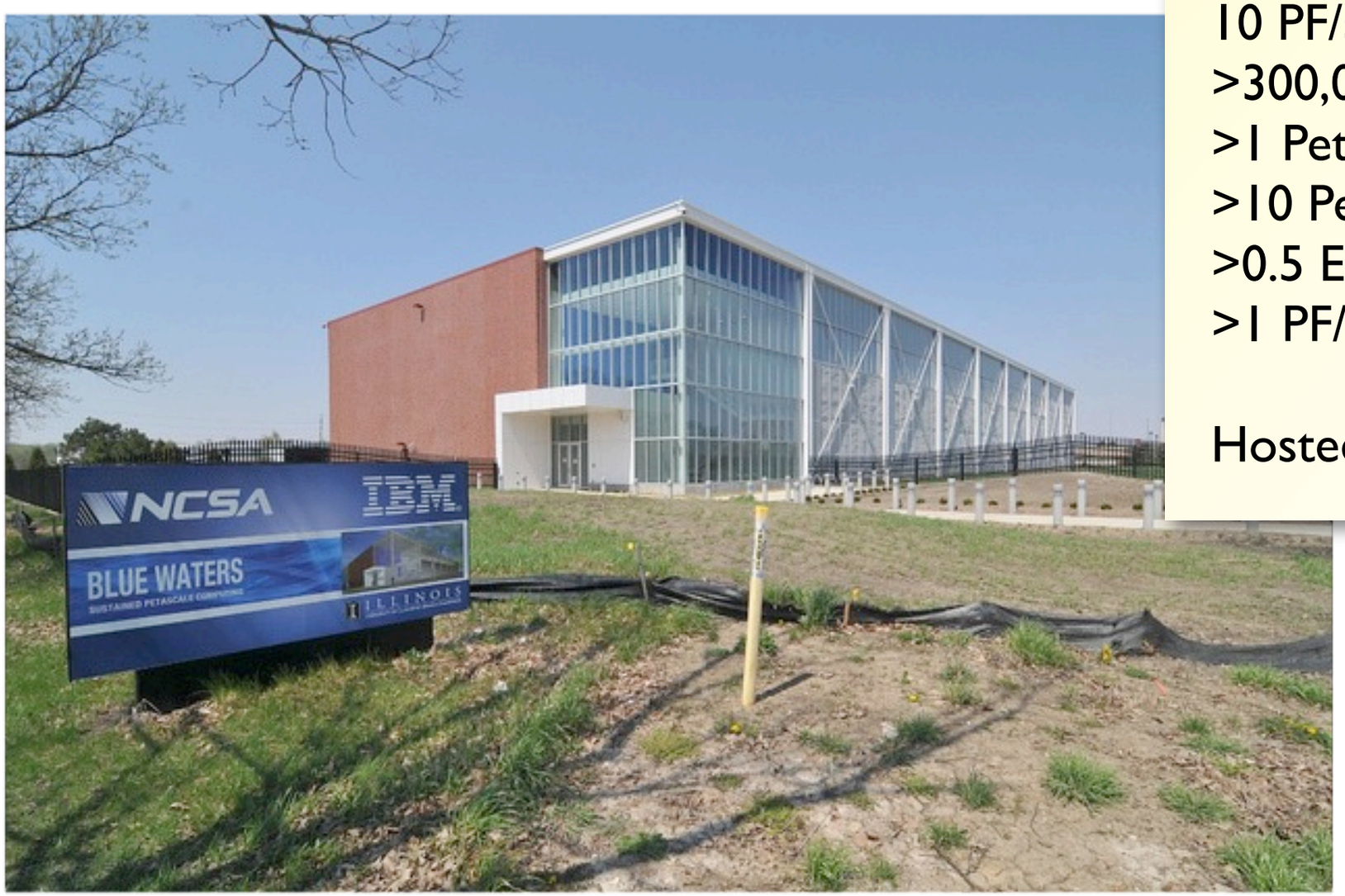
- Working on an abstract machine for traffic analysis
  - Instruction set with domain-specific support for typical operations
  - Compiler to turn it into highly efficient native code
- Developing a concurrency abstraction
  - Large set of very light-weight threads, plus execution context
  - Will work well for the scoping model
  - Eventually, Bro scripts will be compiled into this execution model
  - Avoids having to deal with all the low-level concurrency in C++
- *HILTI: A High-level Intermediary Language for Traffic Analysis*
  - Leveraging LLVM as the backend for code generation
  - Prototype exists

Software Development Support

- Bro has been lacking resources for "polishing"
  - Primarily funded by research grants
  - Maintenance, support & documentation
- Now, NSF has awarded ICSI funding for *development*
  - Three full-time engineers/programmers
  - Collaboration with *National Center for Supercomputing Applications (NCSA)*
- Allows to address areas where Bro needs work
  - User experience
  - Performance

Target: Blue Waters

One of the most powerful supercomputers,
being built at the *National Center for Supercomputing Applications (NCSA)*



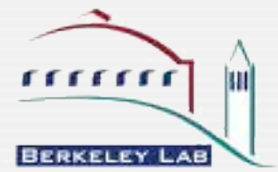
10 PF/s peak performance
>300,000 cores
>1 PetaByte memory
>10 Petabyte disk storage
>0.5 Exabyte archival storage
>1 PF/s sustained on applications

Hosted in 88,000-square-foot facility

Conclusion

The Bro NIDS

- Powerful, open-source system
  - Fundamentally more powerful than the "classic" NIDS
  - Bridges the gap between academia and operations
- Performance is one of the main challenges
  - Load-balancing approach is working today
  - Multi-threading on the radar, but challenging
- The "Bro World" is moving quickly right now
  - NSF support for software development, outside of research grants
- Open-source, and help is always welcome!
  - Student projects are available in many areas ...

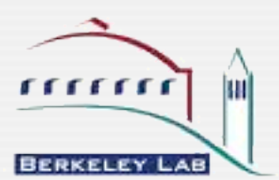


Thanks for your attention.

Robin Sommer

International Computer Science Institute, &
Lawrence Berkeley National Laboratory

robin@icsi.berkeley.edu
http://www.icir.org



Work Plan

Milestones	Bro 1.6				Bro 1.7				Bro 2.0				
	Year 1				**Year 2**				**Year 3**				**Results**
Quarter	1	2	3	4	5	6	7	8	9	10	11	12	
Documentation and Support													
Documentation	P D S	P D S	P D S	P D S	P D S	P D S	S	S	S	S	S	S	Comprehensive manuals for Bro 1.6+; procedures for keeping them current.
Packaging & Installation	D	D											Linux RPMs; FreeBSD port; live CD; modernized build infrastructure.
Configuration Tools	D S	D S	D S	D S	S	S	S	S					Configuration “Wizards”.
Detection Capabilities	D	D	D	D	D	D	D	D	D	D	D	D	New policy scripts detecting recent attack vectors.
Community Portal	S	S	S	S	S	S	S	S	S	S	S	S	Public, web-based portal for exchange of experiences and functionality.
Community Events	P D S	P D S	P D S	P D S	P D S	P D S	P D S	P D S	P D S	P D S	P D S	P D S	Bro workshops and user meetings.
User Support	S	S	S	S	S	S	S	S	S	S	S	S	Dedicated contact person for Bro deployments.
Reliability and Sustainability													
Bugs and other Inconveniences	S	S	D S	D S	D	D							Issue tracker tickets addressed; code analysis of core components.
Quality Assurance			D	D	D	D							Comprehensive test-suite; automated cross-platform builds/tests via NMI.
Modernizing Code Base					D	D							Consistent code base; unified analysis output; removal of obsolete code.
Performance													
Multi-threaded Implementation						P D	P D	P D	P D	D	D	D	Multi-threaded Bro implementation able to exploit modern many-core platforms.
Script Compiler							P D	P D	D	D	D		Compiler translating Bro scripts into native code for the target platform.
Data Analysis													
Interactive Analysis Tool									P D	P D	P D	D	Highly interactive graphical analysis tool for incident analysis and response.
Database Interface									P D	P D	D	D	Transparent database interface for Bro’s scripting language.

Roles: *Principal Investigators* | *Developers* | *User Support*

Going Back in Time with the *Time Machine*

The Utility of Time Travel

- Bro's policy-neutral logs are often the most useful output
  - Typically we do not know in advance how the next attacks looks like
  - But when an incident occurred, we need to understand *exactly* what happened

    "How did the attacker get in? What damage did he do? Did the guy access other hosts as well? How can we detect similar activity in the future?"

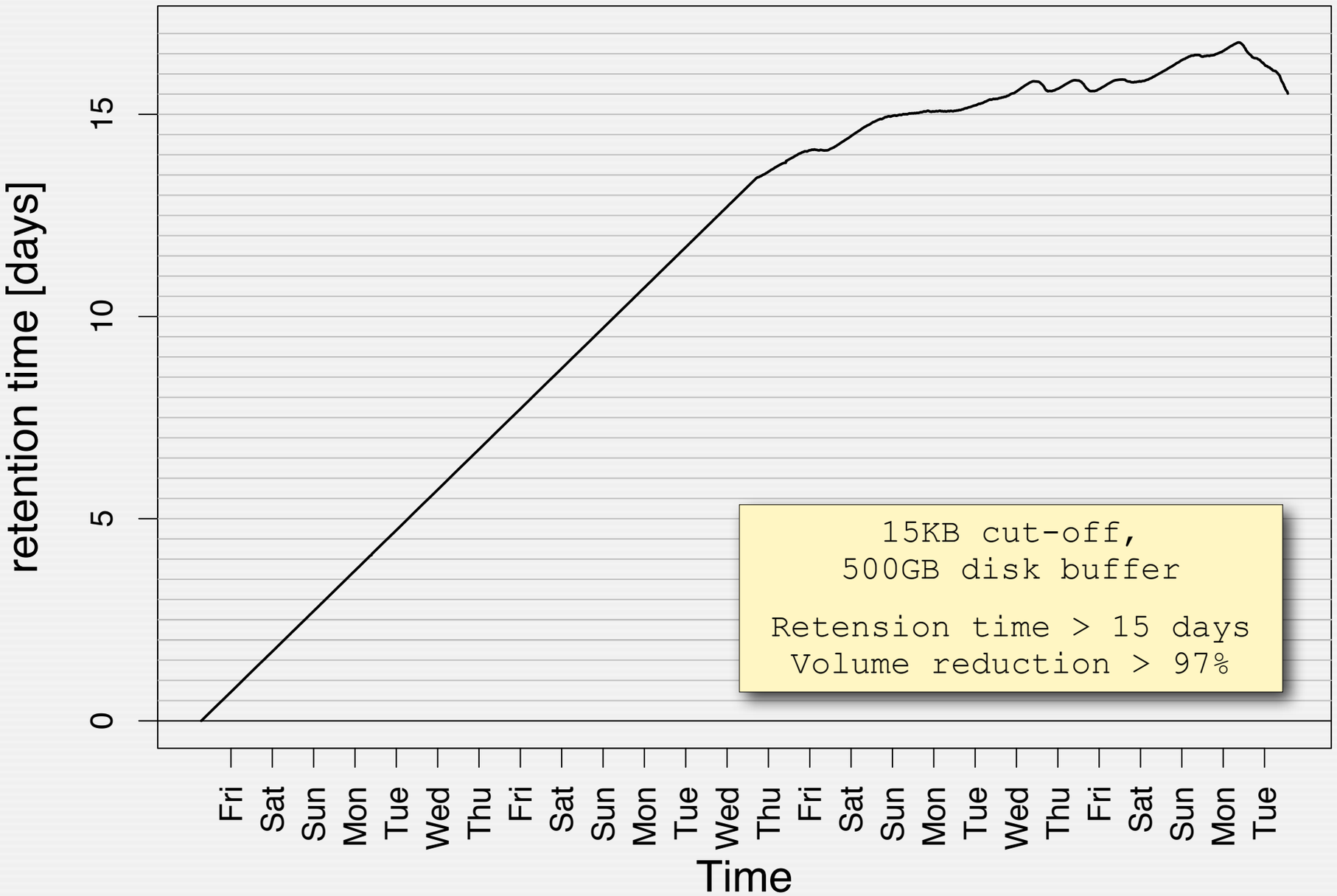
- This is when you need all information you can find

- The most comprehensive resource are *packet traces*
  - Wouldn't it be cool if you had a packet trace of that incident?

The Time Machine

- The *Time Machine*, a bulk-recorder for network traffic
  - Efficient packet recorder for high-volume network streams
  - Taps into a network link and records packets in their entirety
  - Provides efficient query interface to retrieve past traffic
- Storing *everything* is obviously not feasible
- TM uses heuristics for volume reduction
  - *Cut-off*: For each connection, TM stores *only the first few KB*
  - *Expiration*: Once space is exhausted, TM expires oldest packets automatically
- Simple yet very effective scheme
  - Leverages network traffic's "heavy-tails"
  - Even in large networks we can go back in time for several days
  - Proven to be extremely valuable for network forensics in operational use at LBL

Example: Retension Time at LBNL



10Gbps upstream link, 10,000 hosts, 100-200Mbps average, 1-2TB/day

Query Interface

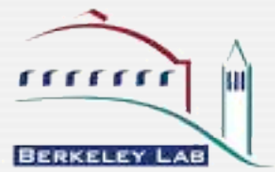
- Interactive console interface

```
# An example query. Results are stored in a file.
query to_file "trace.pcap" index ip "1.2.3.4"

# Dynamic class. All traffic of IP 5.6.7.8 is
# assigned to class alarm
set_dyn_class 5.6.7.8 alarm
```

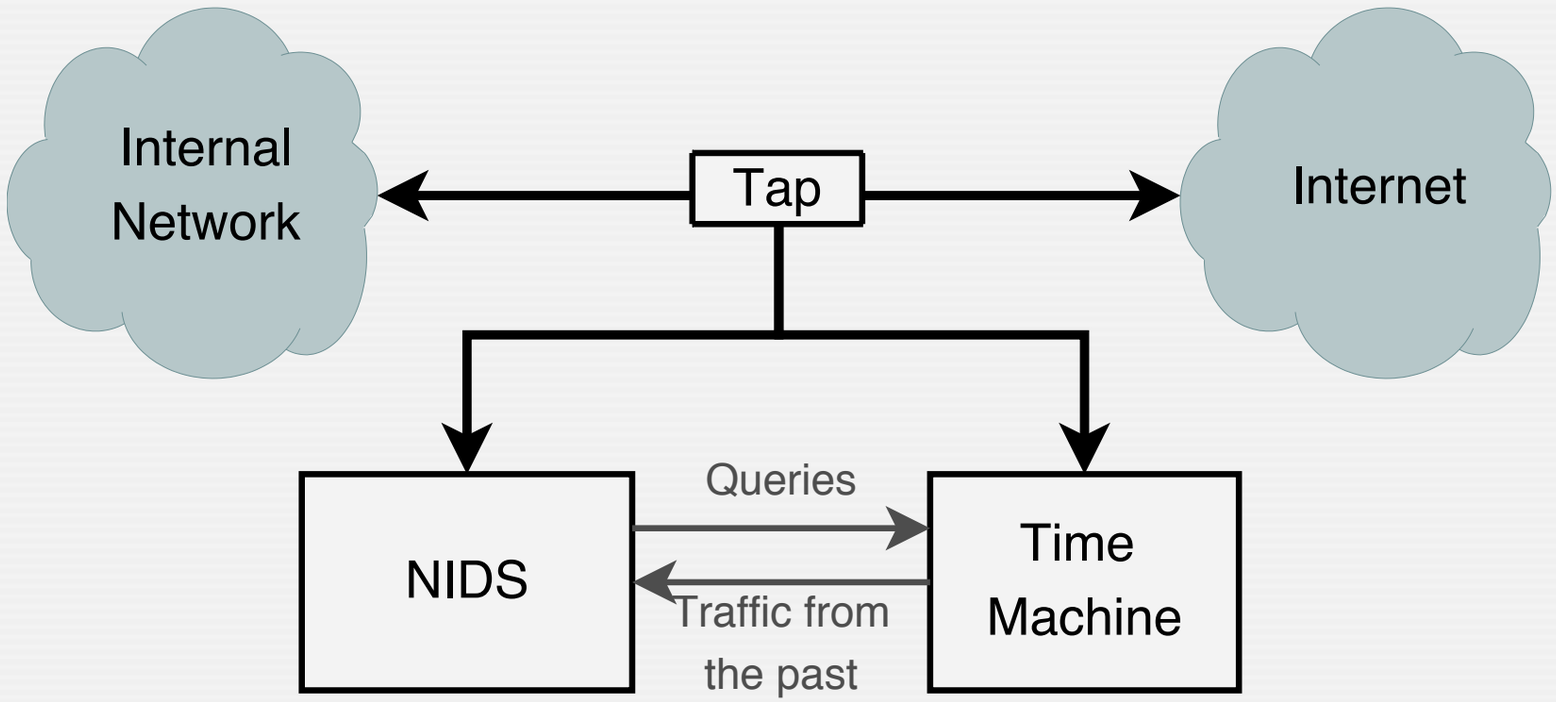
- Command-line client for common queries

```
tm-query --ip 1.2.3.4 localhost host.pcap --time 12h
```



Interfacing the TM with Bro

- The Time Machine can provide a NIDS with historic traffic



- Applications
  - Bro controls the TM (change cut-offs, switch storage class)
  - Bro permanently stores attack traffic
  - Bro analyzes traffic *retrospectively*

Augmenting Bro Alerts with Traffic

01/25/08 12:23:03 HTTP_SensitiveURI [redacted]:55899/tcp = [redacted]
[redacted]:80/tcp = [redacted]

[redacted]/55899 > [redacted]/http %worker-8-1708639: GET /index.php?content=/etc/passwd (200 "OK" [1469] www[redacted])

⊟ Tcpdump of connection's packets (file size: 2725 bytes)

```
12:23:03.048404 IP [redacted]5899 > [redacted].80: S 1104981131:1104981131(0)
12:23:03.048635 IP [redacted].80 > [redacted]899: S 1655847285:1655847285(0) ack 11
12:23:03.236799 IP [redacted]5899 > [redacted].80: . ack 1 win 5840
12:23:03.237600 IP [redacted]5899 > [redacted].80: P 1:110(109) ack 1 win 5840
12:23:03.237841 IP [redacted].80 > [redacted]899: . ack 110 win 5840
12:23:03.239162 IP [redacted].80 > [redacted]899: . 1:1461(1460) ack 110 win 5840
12:23:03.239165 IP [redacted].80 > [redacted]899: P 1461:1699(238) ack 110 win 5840
12:23:03.239167 IP [redacted].80 > [redacted]899: F 1699:1699(0) ack 110 win 5840
12:23:03.426493 IP [redacted]5899 > [redacted].80: . ack 1461 win 8760
12:23:03.426495 IP [redacted]5899 > [redacted].80: . ack 1699 win 8760
12:23:03.426497 IP [redacted]5899 > [redacted].80: F 110:110(0) ack 1700 win 8760
12:23:03.426990 IP [redacted].80 > [redacted]899: . ack 111 win 5840
```

⊞ Strings in connection's packets (file size: 2725 bytes)

⊞ Tcpdump of host's traffic (file size: 14414 bytes)

⊞ Strings in host's traffic (file size: 14414 bytes)

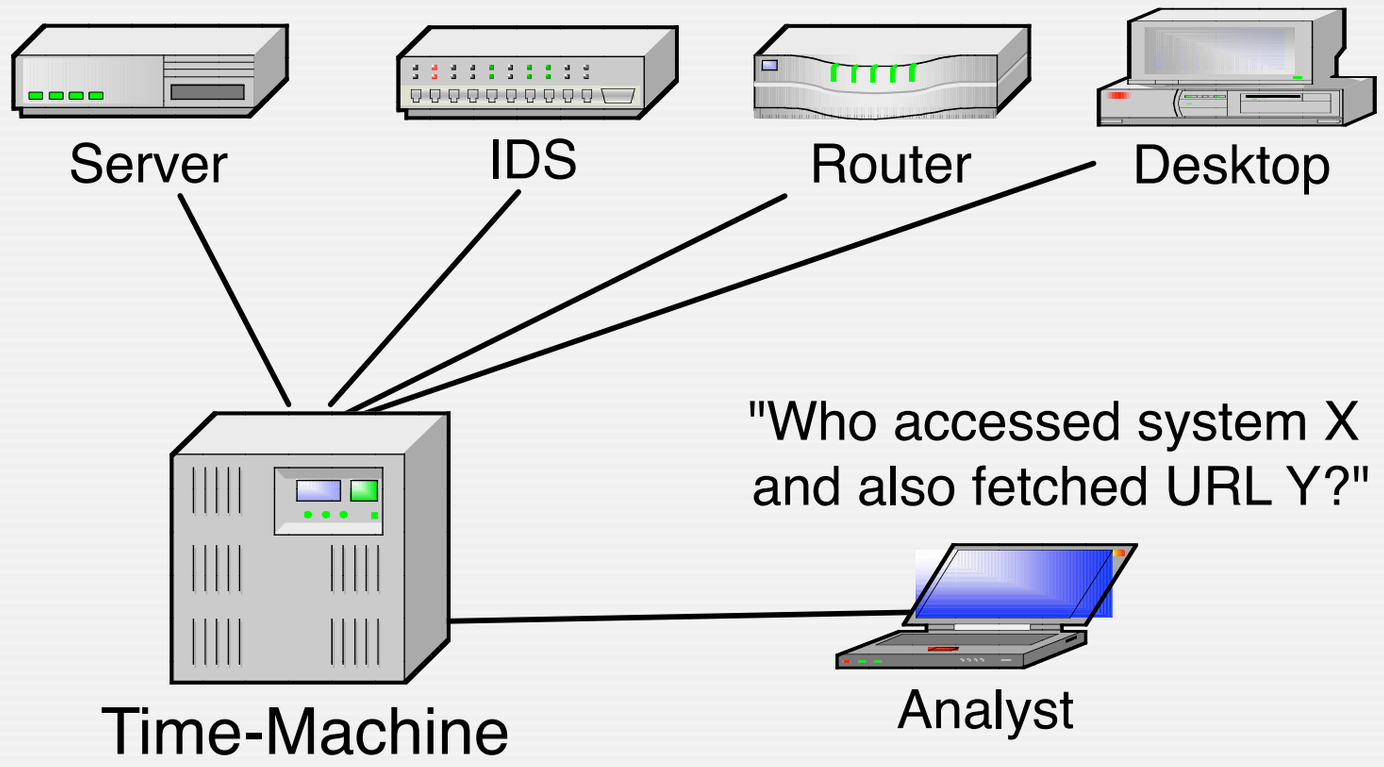
⊞ Reassembled originator contents (file size: 109 bytes)

⊟ Reassembled responder contents (file size: 1698 bytes)

```
HTTP/1.1 200 OK
Date: Fri, 25 Jan 2008 20:23:03 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.8a PHP/5.1.6 mod_jk/1.2.19
X-Powered-By: PHP/5.1.6
Content-Length: 1469
Connection: close
```

Current Work: "Time Machine NG"

- We are now building a generalized Time Machine
  - Incorporates arbitrary network activity rather than just packets
  - Allows live queries for future activity



One goal: Facilitate cross-site information sharing

Multi-Core Bro Data Flow