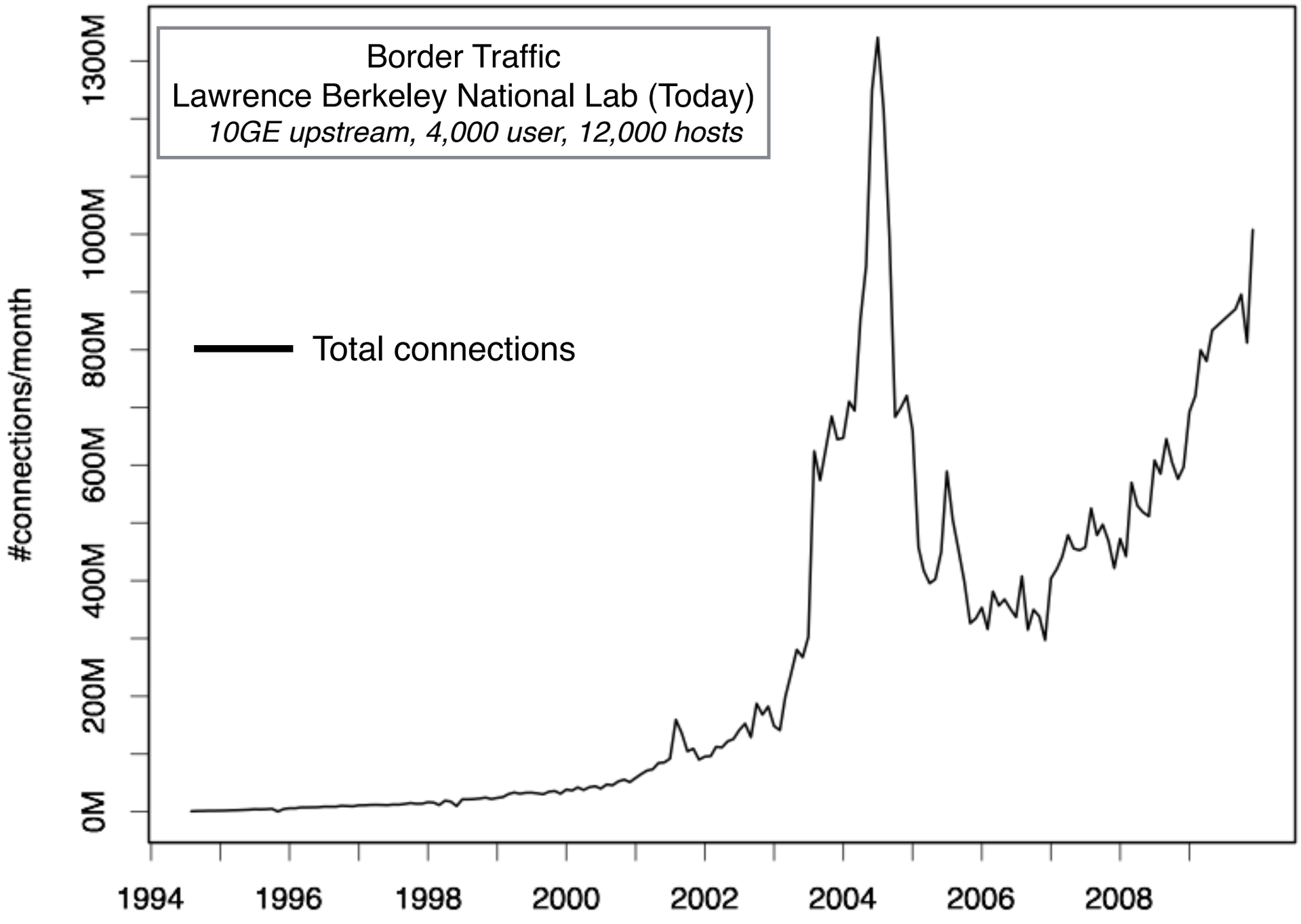
Network Security Today:
Finding Complex Attacks at 100Gb/s

Robin Sommer

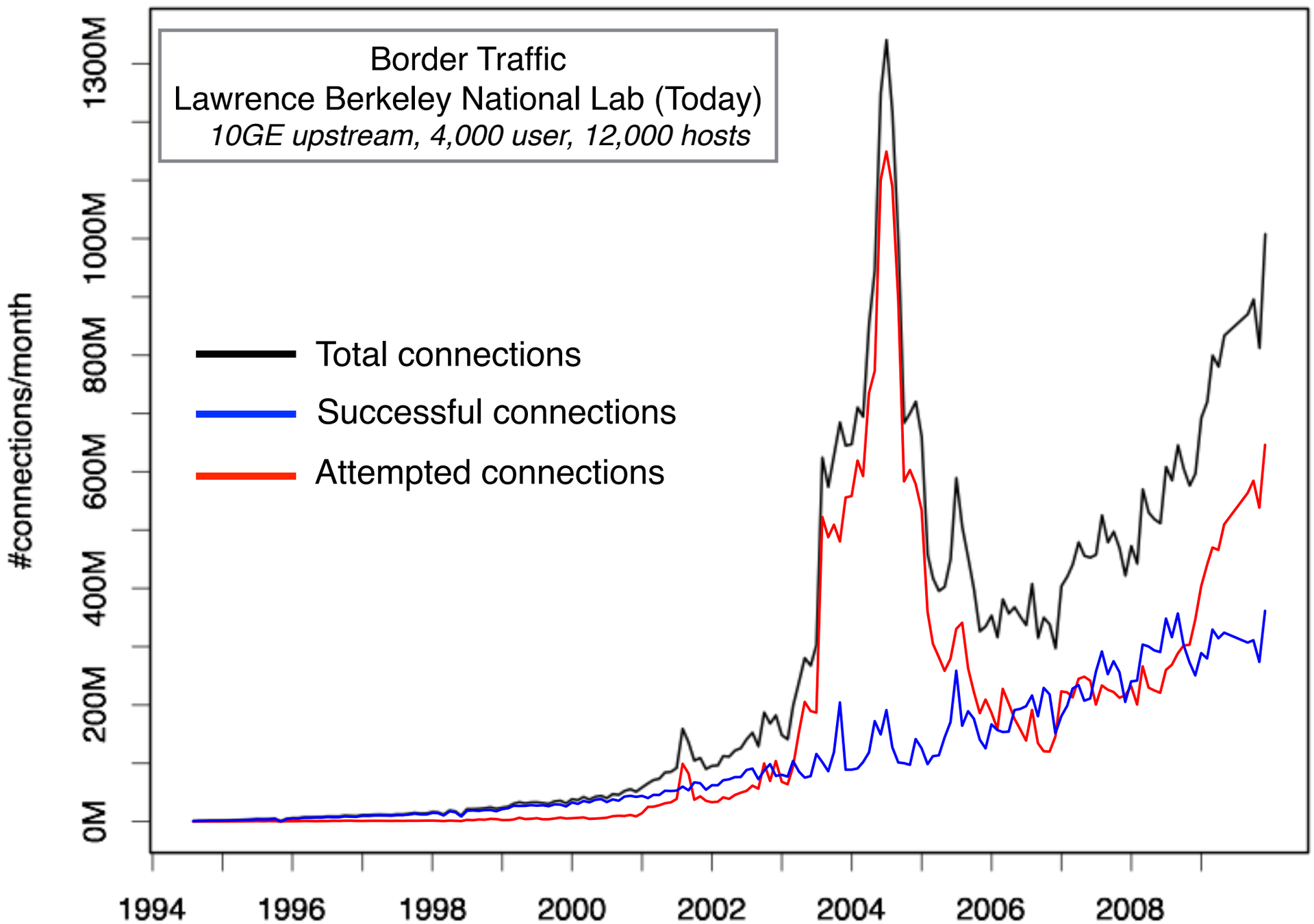
International Computer Science Institute, &
Lawrence Berkeley National Laboratory

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

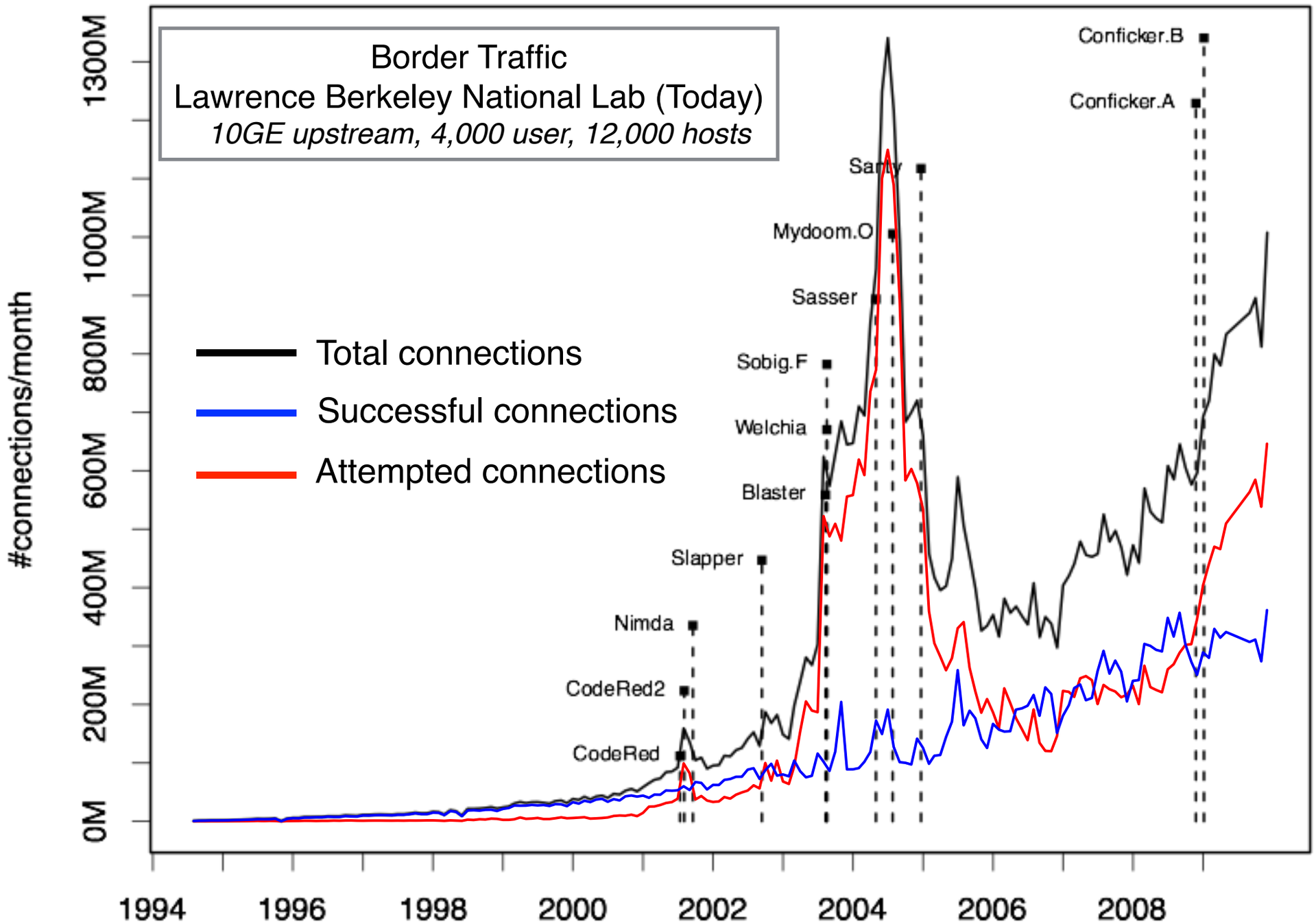
The Old Days …


Border Traffic
Lawrence Berkeley National Lab (Today)
10GE upstream, 4,000 user, 12,000 hosts
Total connections
#connections/month
0M
200M
400M
600M
800M
1000M
1300M
1994
1996
1998
2000
2002
2004
2006
2008

The Old Days …


Border Traffic
Lawrence Berkeley National Lab (Today)
10GE upstream, 4,000 user, 12,000 hosts
Total connections
Successful connections
Attempted connections
#connections/month
0M
200M
400M
600M
800M
1000M
1300M
1994
1996
1998
2000
2002
2004
2006
2008

The Old Days …


Border Traffic
Lawrence Berkeley National Lab (Today)
10GE upstream, 4,000 user, 12,000 hosts
Total connections
Successful connections
Attempted connections
#connections/month
0M
200M
400M
600M
800M
1000M
1300M
1994
1996
1998
2000
2002
2004
2006
2008
Nimda
CodeRed2
CodeRed
Slapper
Welchia
Blaster
Sasser
Mydoom.O
Sobig.F
Conficker.B
Conficker.A

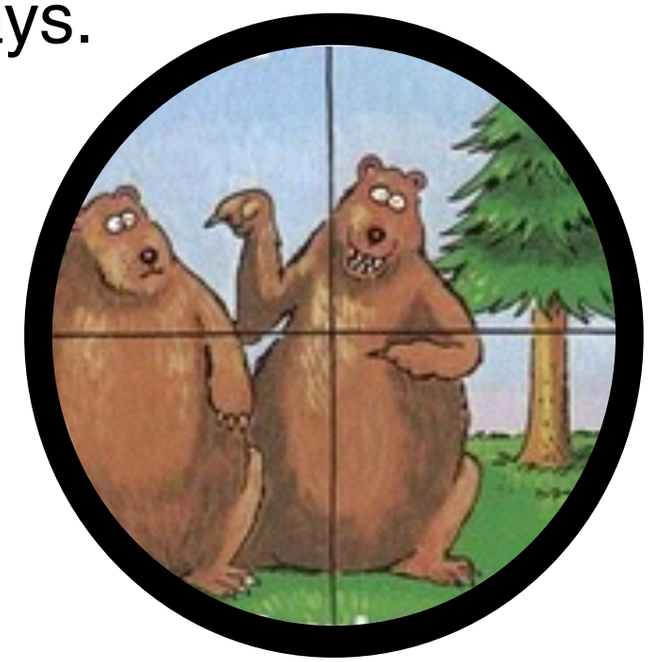
Today's Threats

Today's Threats

Trend 1: Commercialization of attacks

Thriving underground economy ("Crime-as-a-Service").

Bear Race: Attack is good enough if it pays.



Source: Gary Larson

Today's Threats

Trend 1: Commercialization of attacks

Thriving underground economy ("Crime-as-a-Service").
Bear Race: Attack is good enough if it pays.

Trend 2: High-skill / high-resource attacks.

Activist Hacking.
Advanced Persistent Threats / Nation-states.

Source: Wikimedia Commons

Source: Computer Security Articles

Source: EFF

Today's Threats

Trend 1: Commercialization of attacks

- Thriving underground economy ("Crime-as-a-Service").
- Bear Race: Attack is good enough if it pays.

Trend 2: High-skill / high-resource attacks.

- Activist Hacking.
- Advanced Persistent Threats / Nation-states.

Trend 3: Insider Attacks

- Exfiltration
- Sabotage

Defender Challenges

Varying threat models.

No ring rules them all.

Defender Challenges

Varying threat models.

No ring rules them all.

Semantic complexity.

The action is really at the application-layer.

Defender Challenges

Varying threat models.

No ring rules them all.

Semantic complexity.

The action is really at the application-layer.

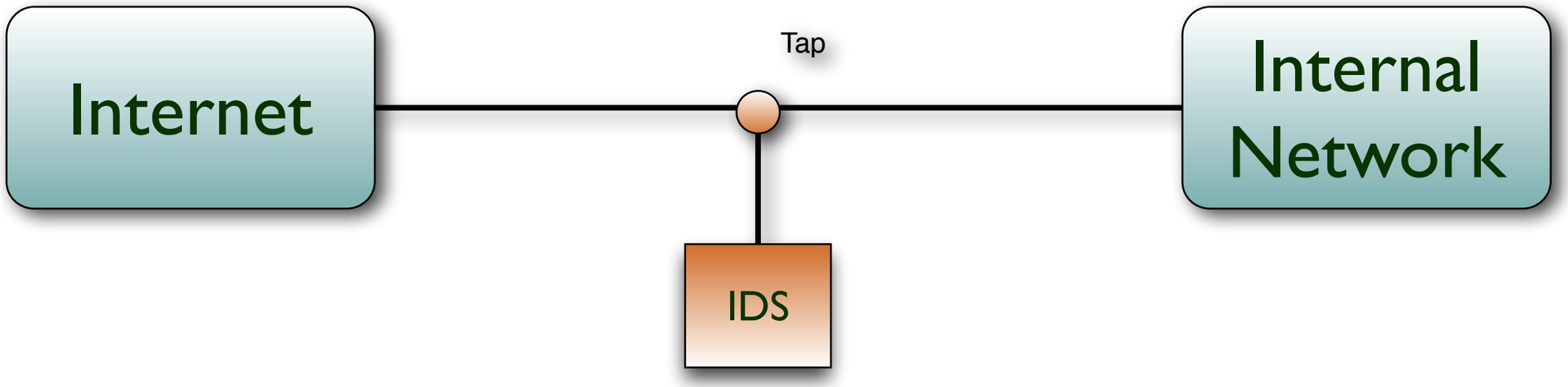
Volume and variability.

Network traffic is an enormous haystack.

Deep Packet Inspection at High Speed

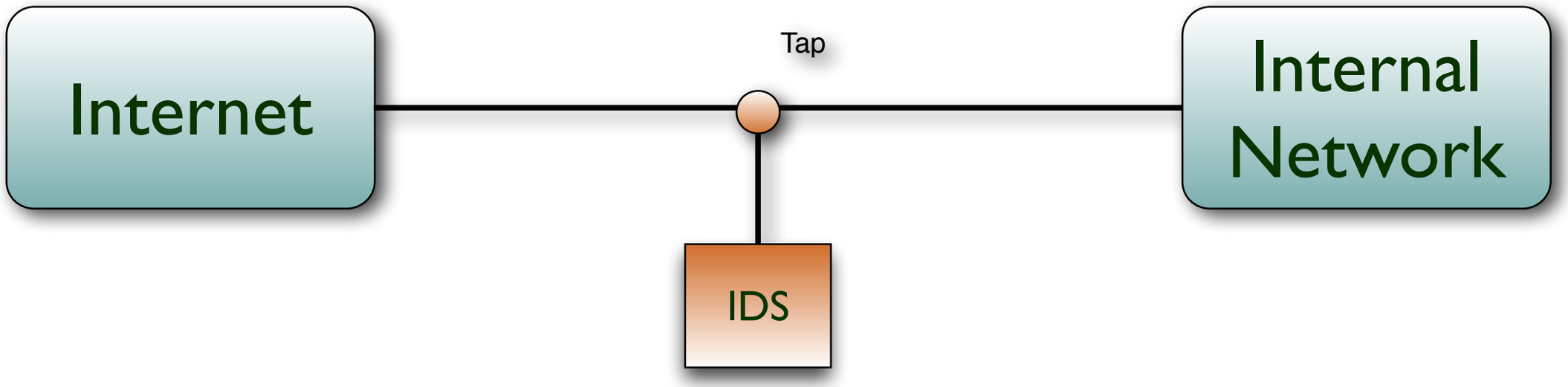
Analyzing Semantics

Analyzing Semantics



Example: Finding downloads of known malware.

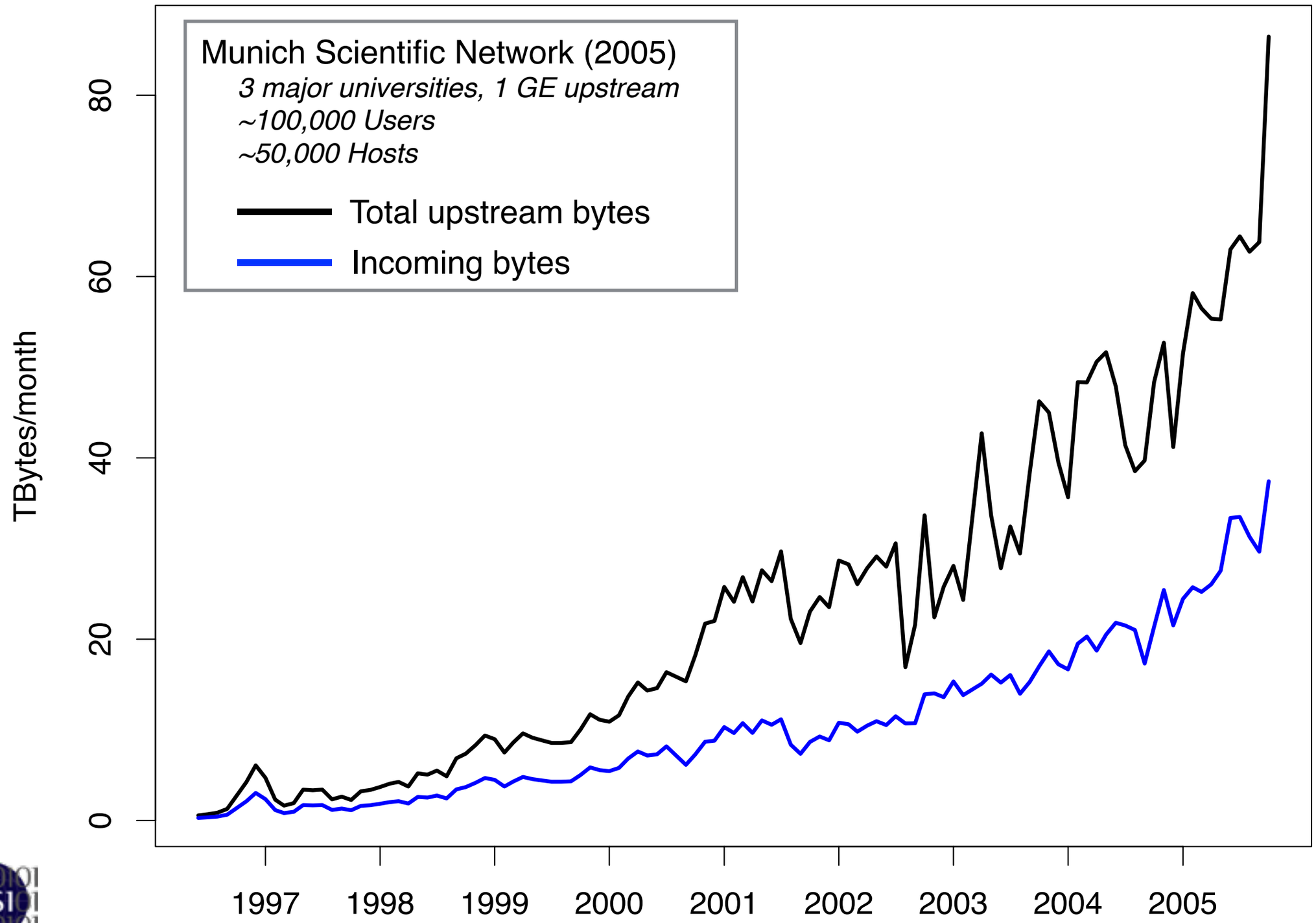
Analyzing Semantics



Example: Finding downloads of known malware.

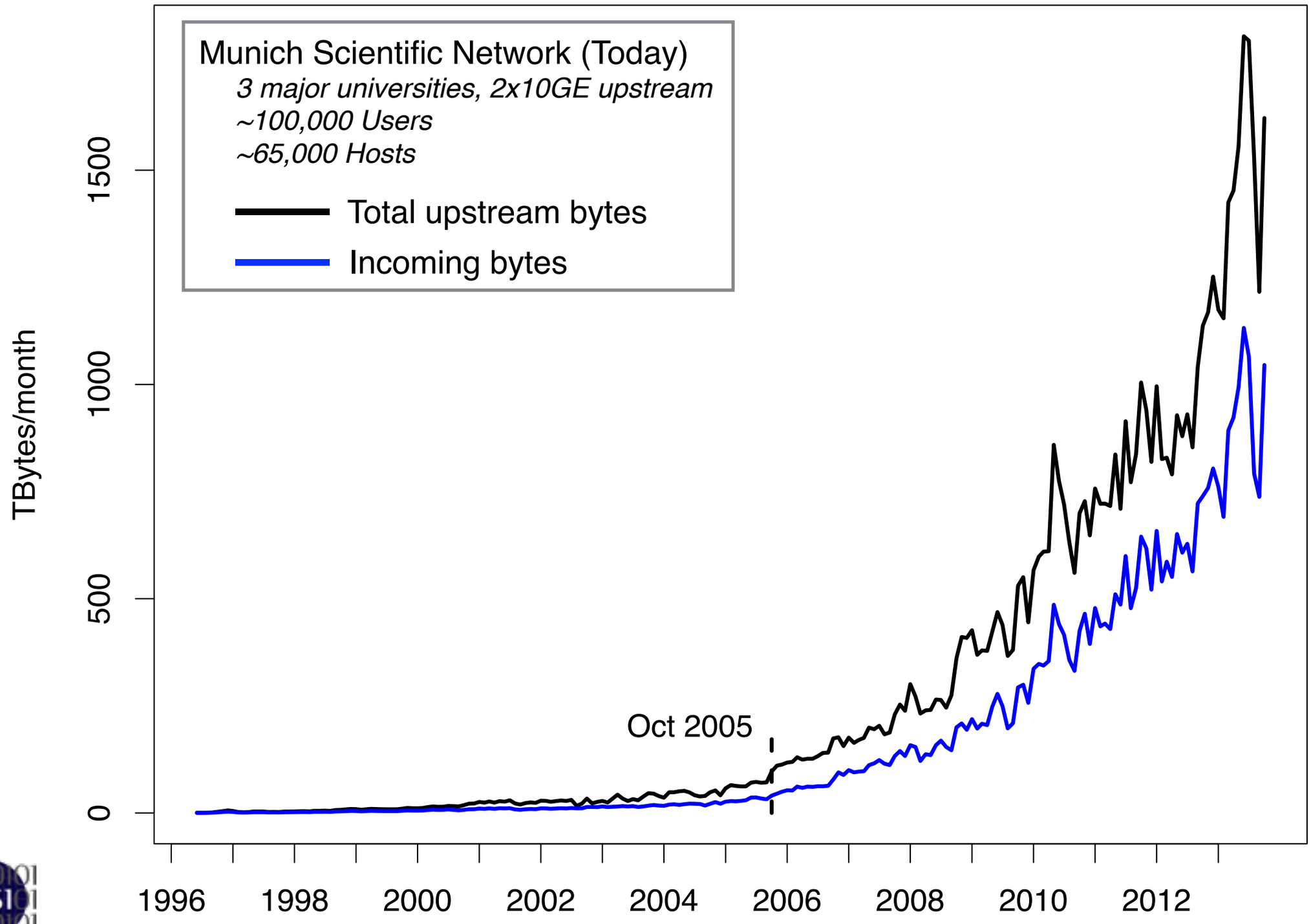
1. Find and parse all Web traffic.
2. Find and extract binaries.
3. Compute hash and compare with database.
4. Report, and potentially kill, if found.

Back in 2005 …



Data: Leibniz-Rechenzentrum, München

Back in 2005 …



Data: Leibniz-Rechenzentrum, München

Traditional Gap: Research vs. Operations

Conceptually simple tasks can be hard in practice.

- Academic research often neglects operational constraints.
- Operations cannot leverage academic results.

We focus on working *with* operations.

- Close collaborations with several large sites.
- Extremely fruitful for both sides.

Research Platform: Bro

Research Platform: Bro

Originally developed by Vern Paxson in 1996.

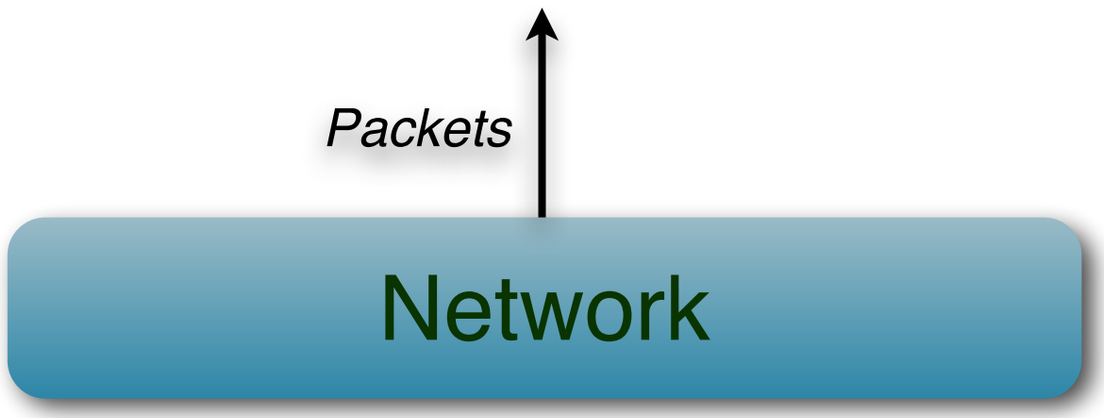
Open-source, BSD-license, maintained at ICSI and NCSA.

In operational use since the beginning.

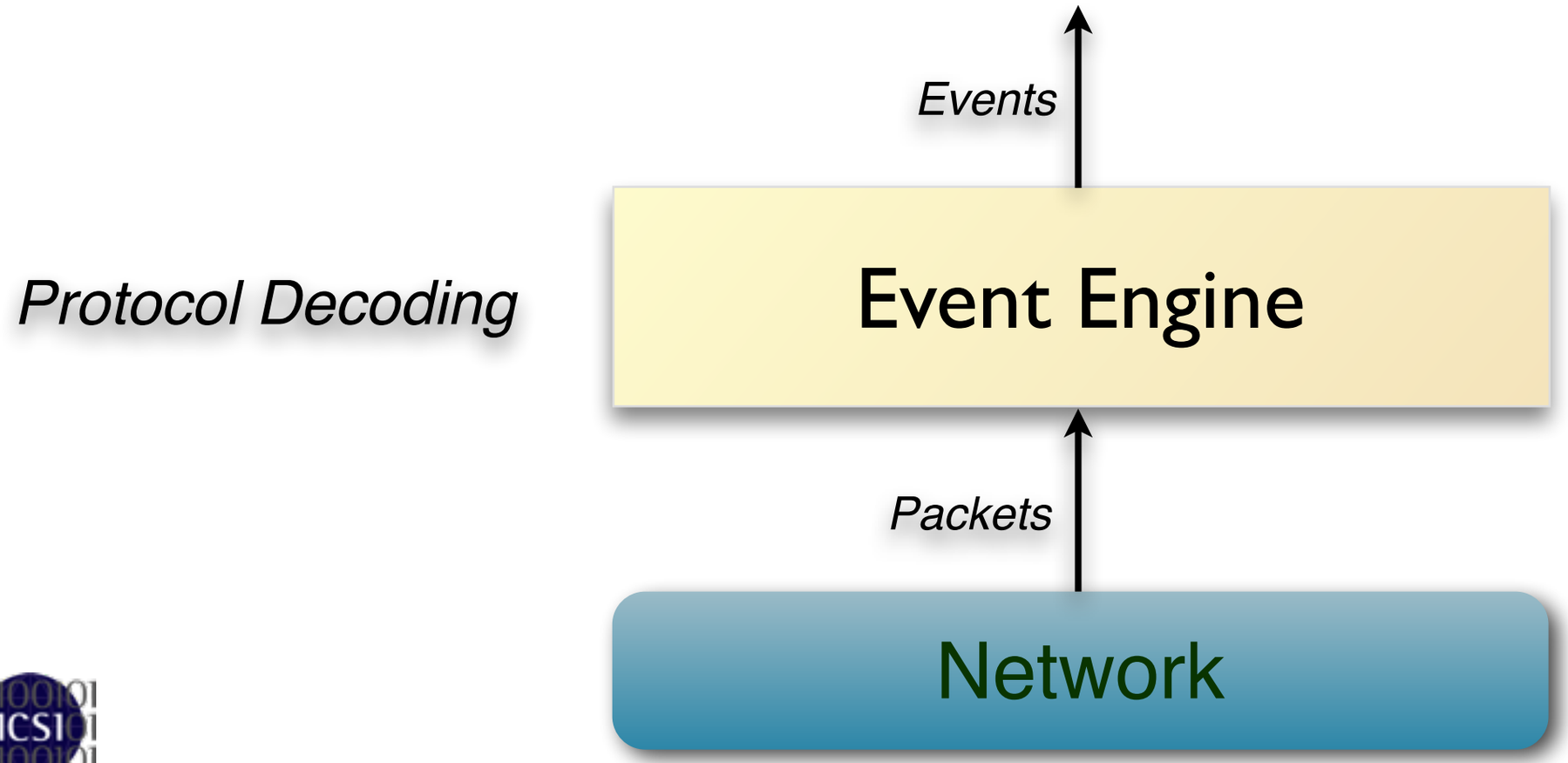
Conceptually very different from other IDS.

http://www.bro.org

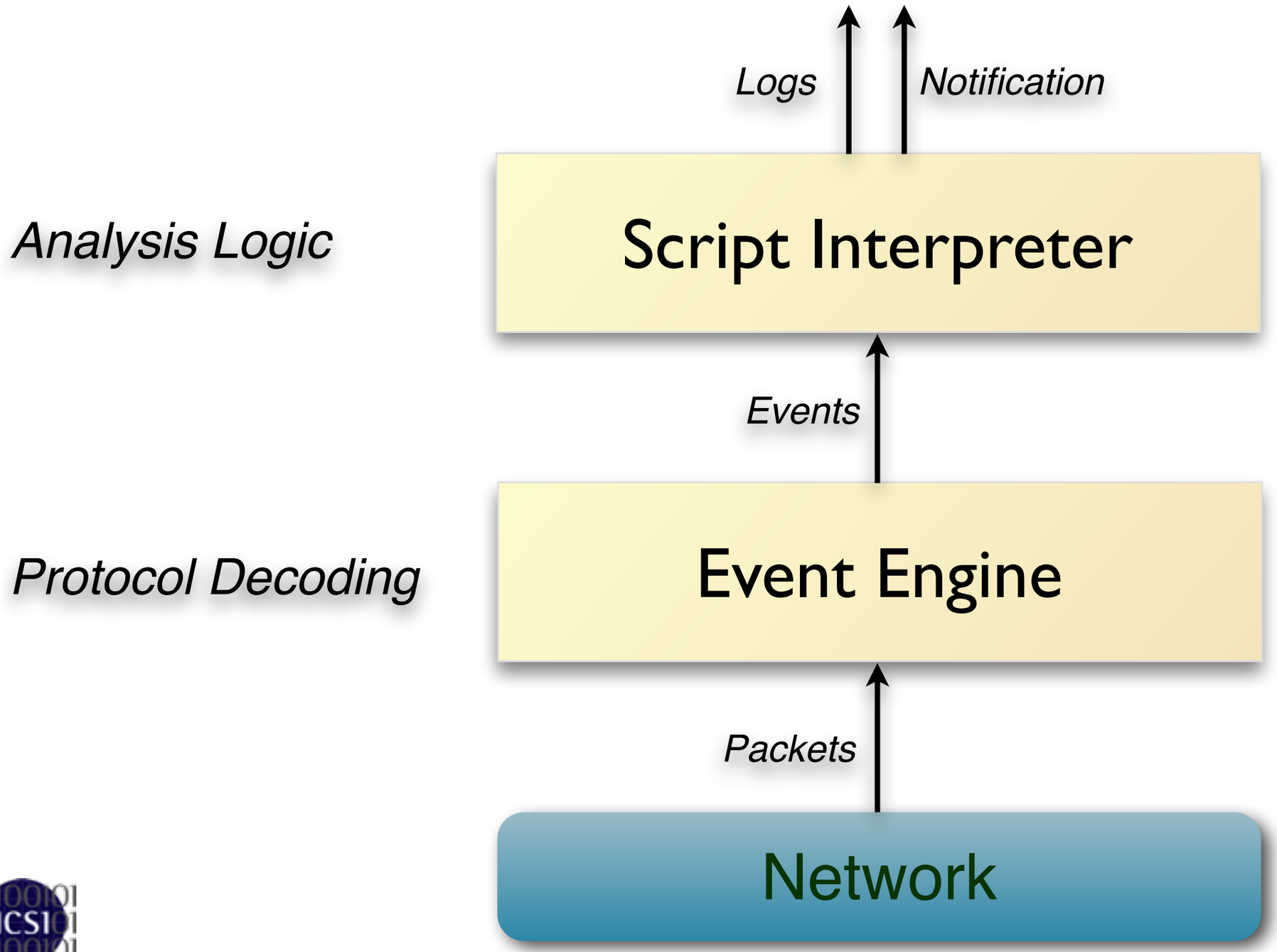
Architecture



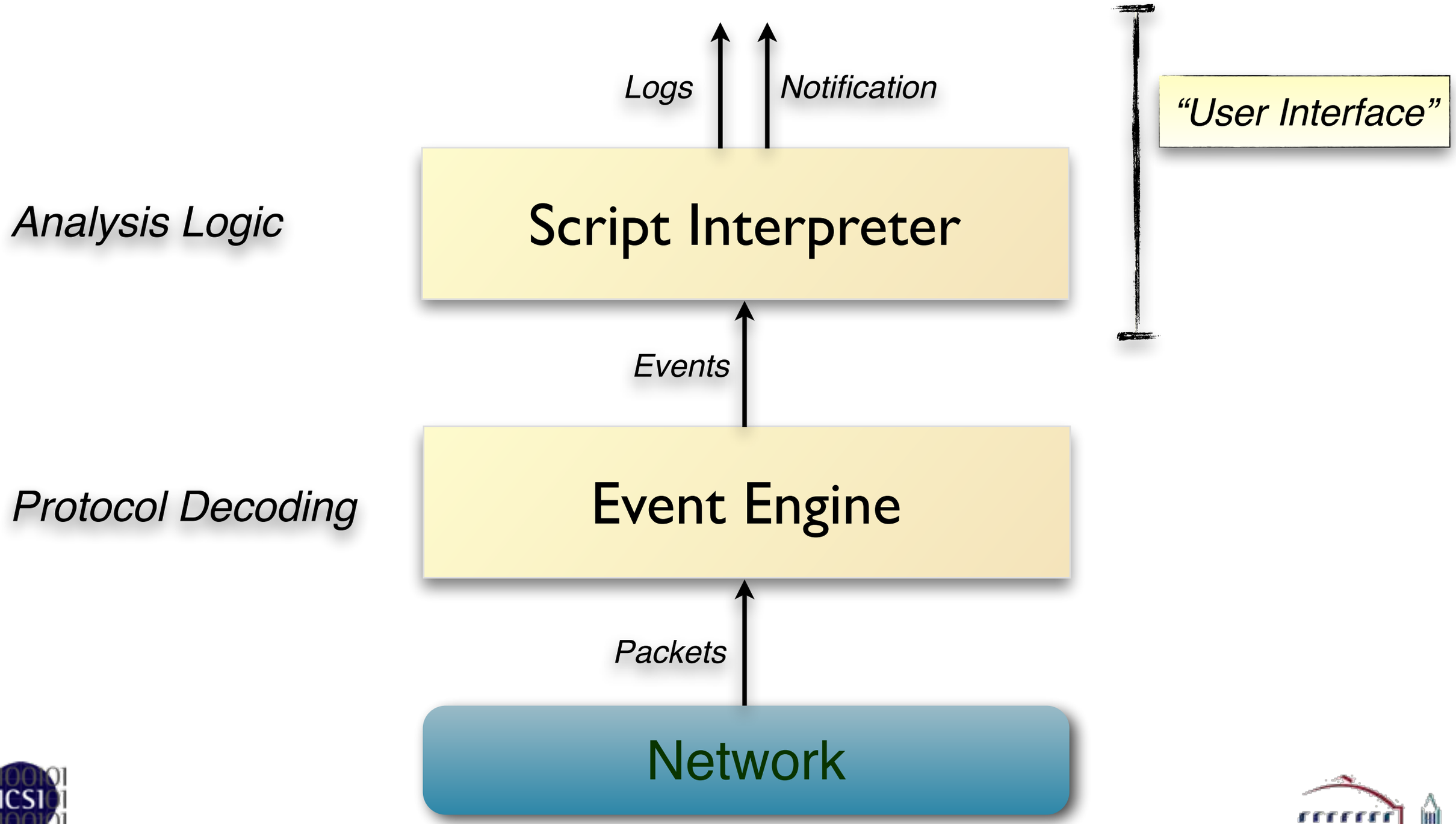
Architecture



Architecture



Architecture



Script Example: Matching URLs

Task: Report all Web requests for a file "`passwd`"

Script Example: Matching URLs

Task: Report all Web requests for a file "`passwd`"

```
event http_request(c: connection,          # Connection.
                   method: string,         # HTTP method.
                   original_URI: string,   # Requested URL.
                   unescaped_URI: string, # Decoded URL.
                   version: string)        # HTTP version.
{
    if ( method == "GET" && unescaped_URI == /.*passwd/ )
        NOTICE(...); # Alarm.
}
```

Script Example: Scan Detector

Task: Count failed connection attempts per source address.

Script Example: Scan Detector

Task: Count failed connection attempts per source address.

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

event connection_rejected(c: connection)
{
    local orig = c$id$orig_h;      # Get originator address.

    local n = ++attempts[orig];    # Increase counter.

    if ( n == SOME_THRESHOLD )     # Check for threshold.
        NOTICE(...);               # Alarm.
}
```

"Who's Using It?"

Diverse Deployment Base

Universities
Research Labs
Supercomputer Centers
Government Organizations
Fortune 20 Enterprises

Examples

Lawrence Berkeley National Lab
National Center for Supercomputing Applications
National Center for Atmospheric Research
Indiana University

... and many more sites

Fully integrated into *Security Onion*

Popular security-oriented Linux distribution

Recent User Meetings

Bro Workshops 2011/13 at NCSA
Bro Exchange 2012 at NCAR

Attended by about 50-80 operators from from 30-40 organizations

Bro History

1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013

Vern writes 1st line of code

Bro History

1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013

Vern writes 1st line of code

LBNL starts using Bro operationally

v0.2
1st CHANGES entry

v0.4
HTTP analysis
Scan detector
IP fragments
Linux support

v0.6
RegExps
Login analysis

v0.7a48
Consistent CHANGES

v0.7a90
Profiling
State Mgmt

v0.7a175/0.8aX
Signatures
SMTP
IPv6 support
User manual

0.8a37
Communication
Persistence
Namespaces
Log Rotation

v0.8aX/0.9aX
SSL/SMB
STABLE releases
BroLite

v1.0
BinPAC
IRC/RPC analyzers
64-bit support
Sane version numbers

v1.1/v1.2
when Stmt
Resource tuning
Broccoli
DPD

v1.3
Ctor expressions
GeoIP
Conn Compressor

v1.4
DHCP/BitTorrent
HTTP entities
NetFlow
Bro Lite Deprecated

v1.5
BroControl

Bro SDCI

v2.0
New Scripts

v2.1
IPv6
Input Framew.

v2.2
File Analysis
Summary Stat.

Bro Center

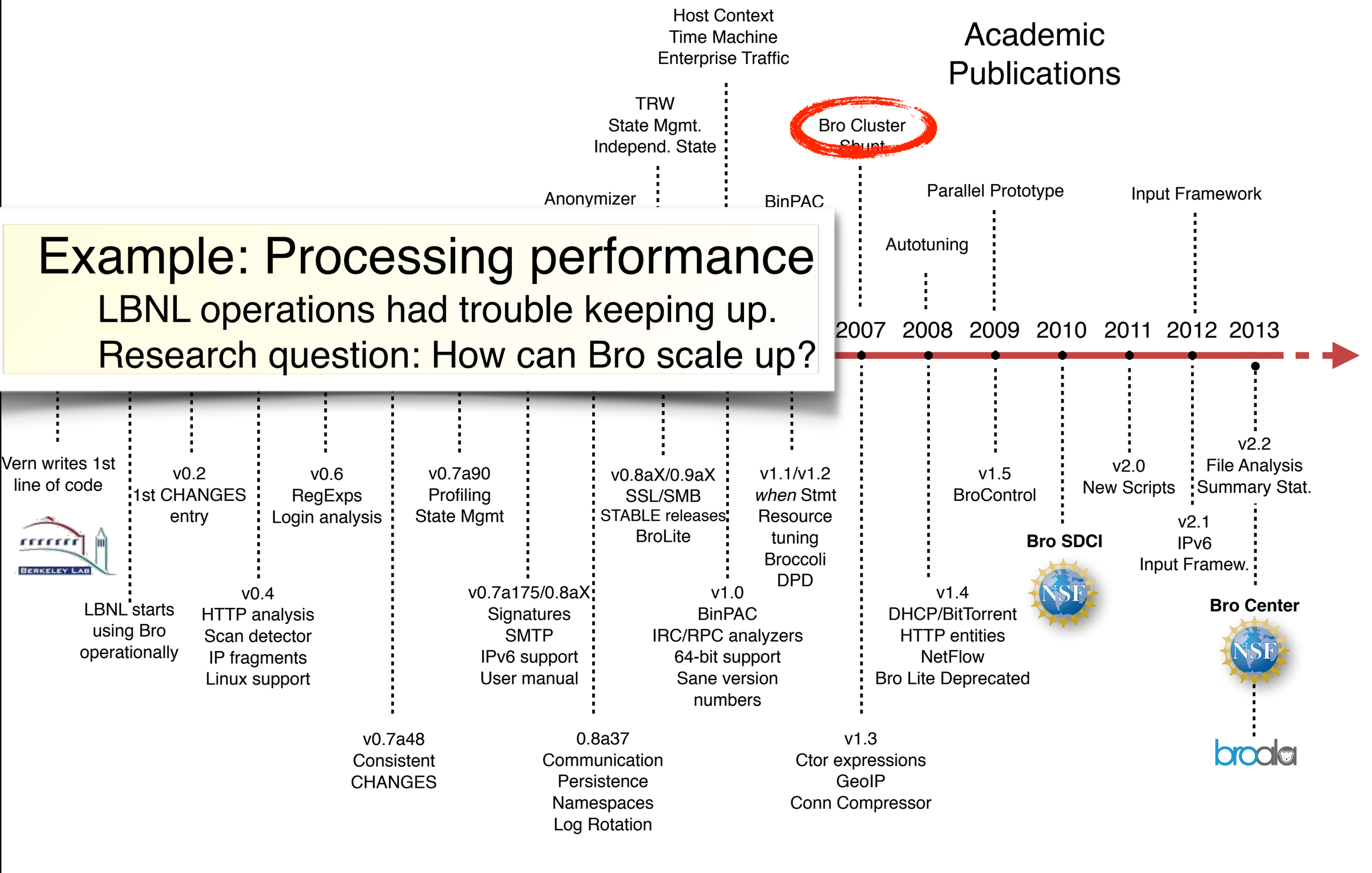
Bro History

Academic Publications

1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013

- 1995 — Vern writes 1st line of code
- 1996 — LBNL starts using Bro operationally
- 1997 — v0.2 1st CHANGES entry
- 1998 — USENIX Paper; v0.4 HTTP analysis, Scan detector, IP fragments, Linux support
- 1999 — v0.6 RegExps, Login analysis
- 2000 — Stepping Stone Detector; v0.7a48 Consistent CHANGES
- 2001 — v0.7a90 Profiling, State Mgmt
- 2002 — v0.7a175/0.8aX Signatures, SMTP, IPv6 support, User manual
- 2003 — Anonymizer, Active Mapping, Context Signat.; 0.8a37 Communication, Persistence, Namespaces, Log Rotation
- 2004 — TRW, State Mgmt., Independ. State; v0.8aX/0.9aX SSL/SMB, STABLE releases, BroLite
- 2005 — Host Context, Time Machine, Enterprise Traffic; v1.0 BinPAC, IRC/RPC analyzers, 64-bit support, Sane version numbers
- 2006 — BinPAC, DPD, 2nd Path; v1.1/v1.2 *when* Stmt, Resource tuning, Broccoli, DPD
- 2007 — Bro Cluster, Shunt; v1.3 Ctor expressions, GeoIP, Conn Compressor
- 2008 — Autotuning; v1.4 DHCP/BitTorrent, HTTP entities, NetFlow, Bro Lite Deprecated
- 2009 — Parallel Prototype; v1.5 BroControl
- 2010 — **Bro SDCI** (NSF); v2.0 New Scripts
- 2011 — v2.1 IPv6, Input Framew.
- 2012 — Input Framework; v2.2 File Analysis, Summary Stat.
- 2013 — **Bro Center** (NSF); broala

Bro History



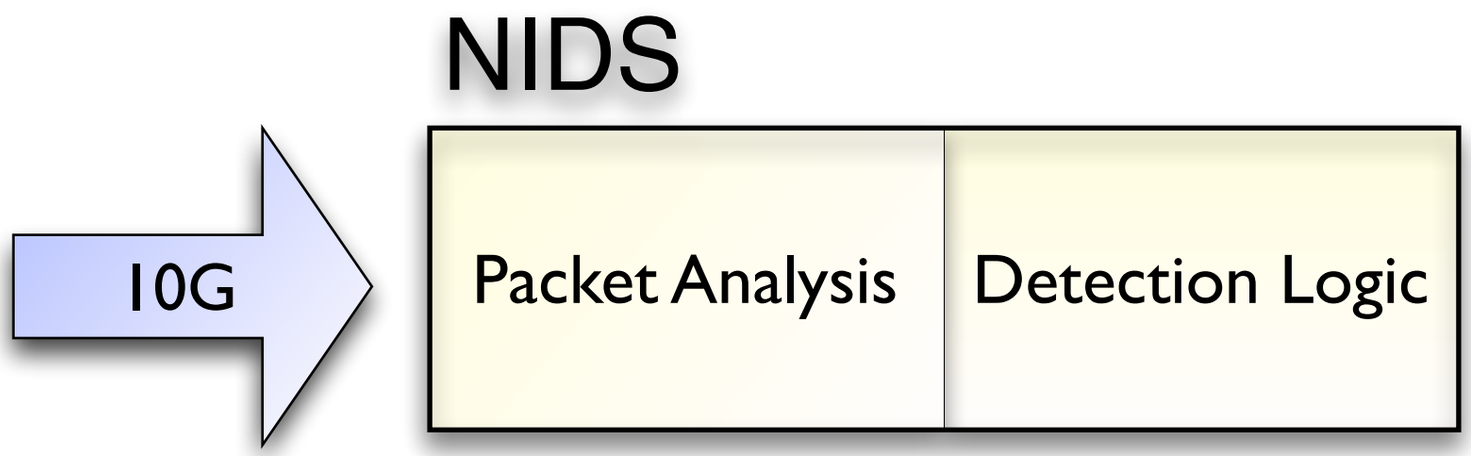
Example: Processing performance

LBNL operations had trouble keeping up.

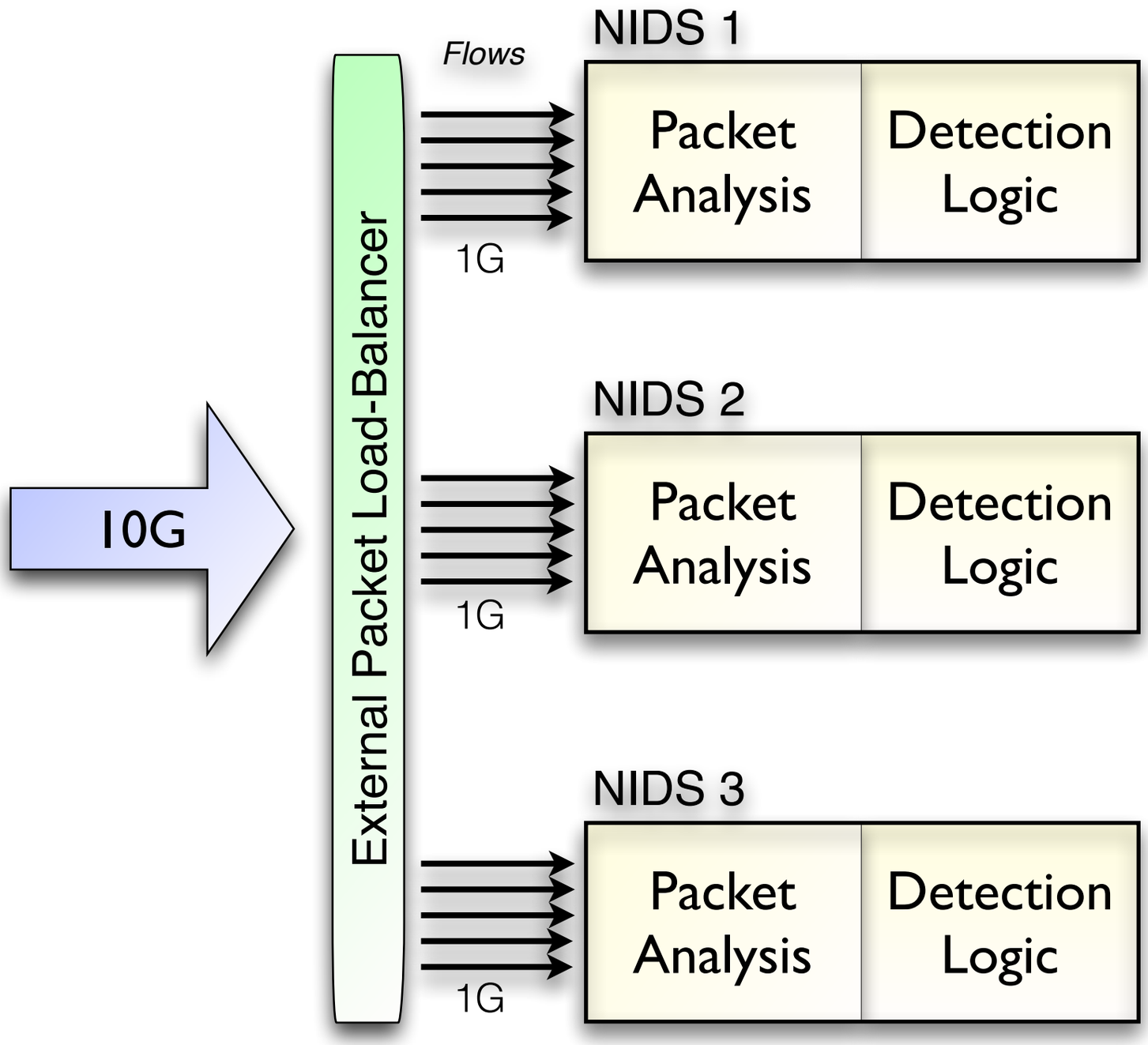
Research question: How can Bro scale up?

Load-balancing Architecture

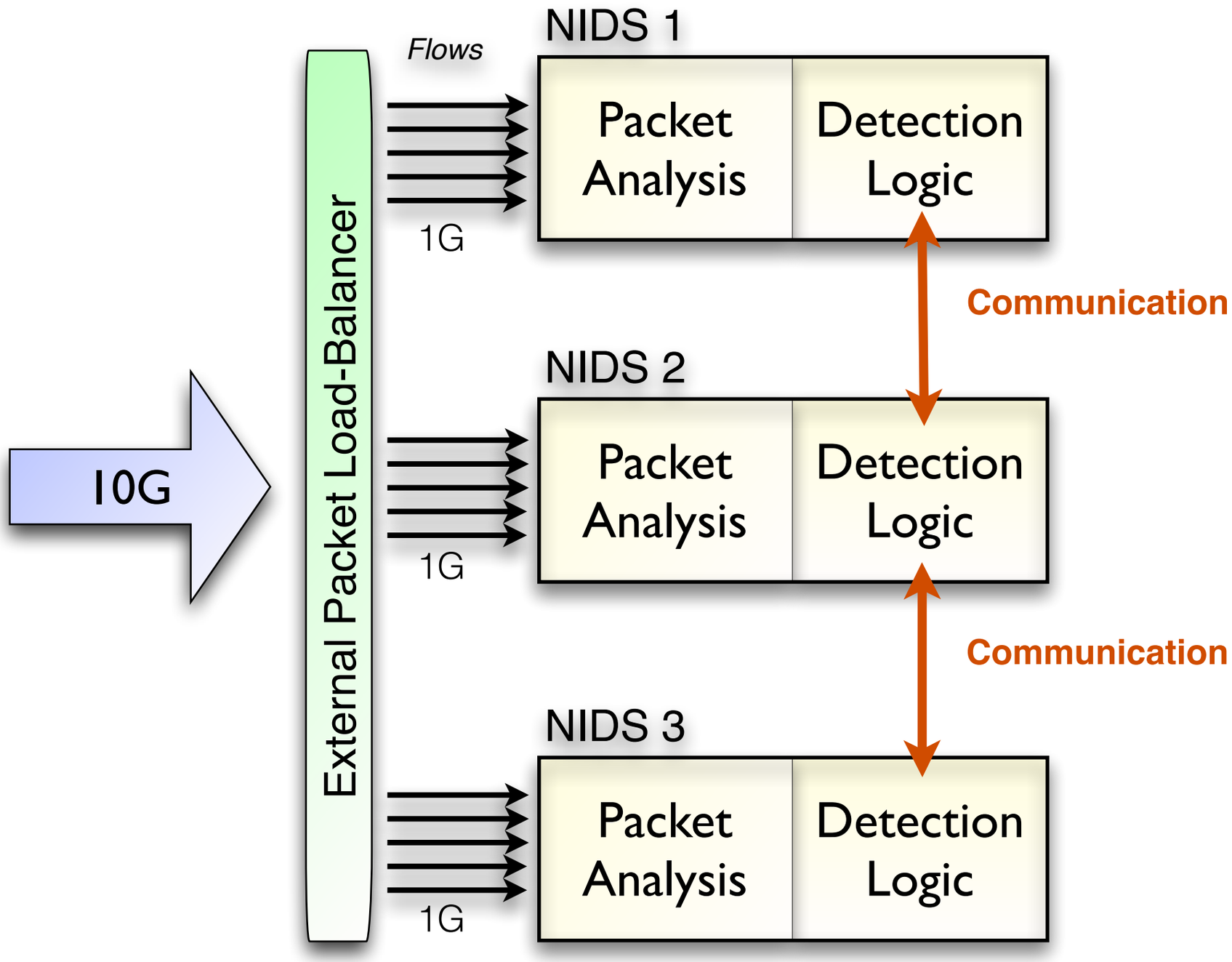
Load-balancing Architecture



Load-balancing Architecture

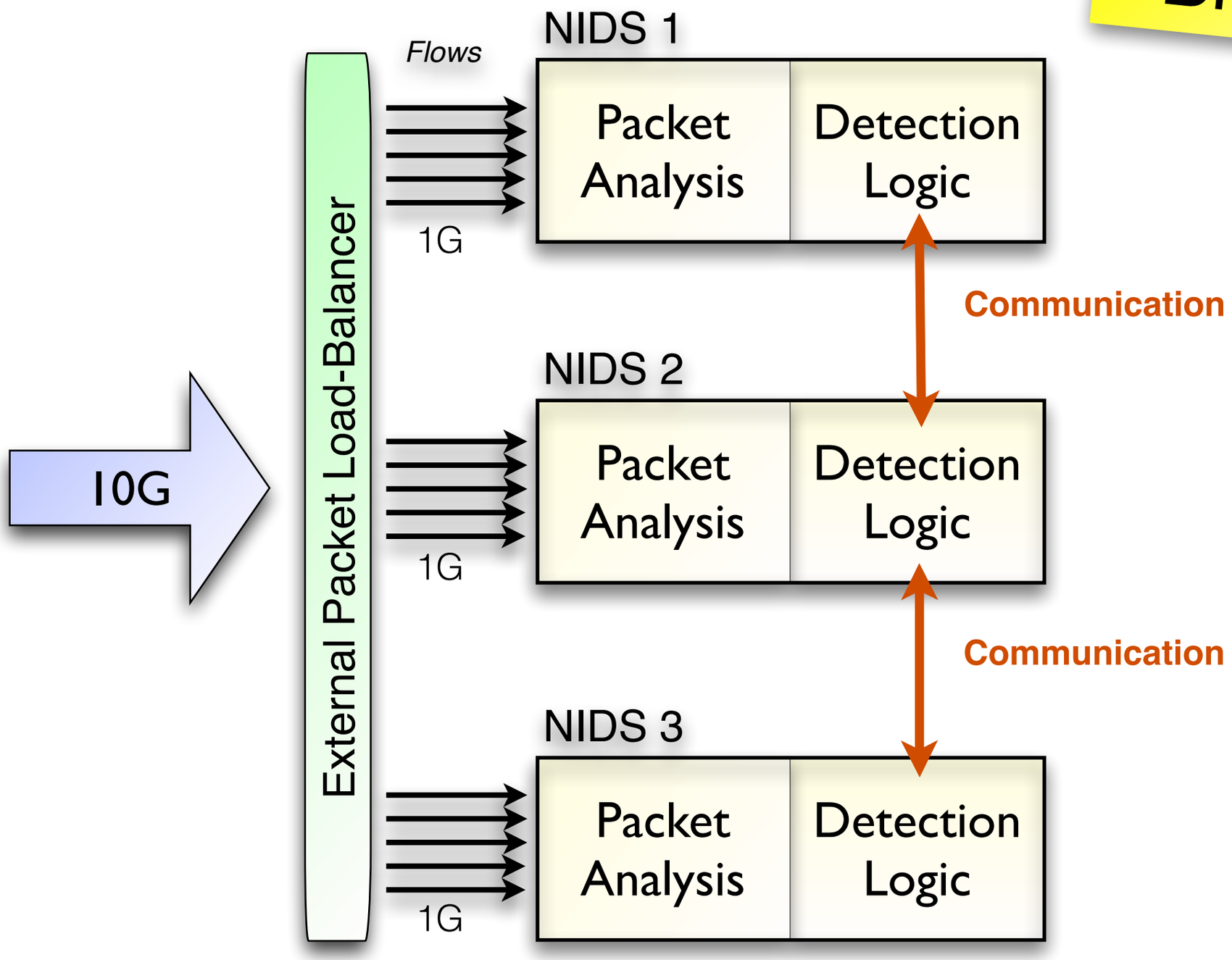


Load-balancing Architecture



Load-balancing Architecture

"Bro Cluster"

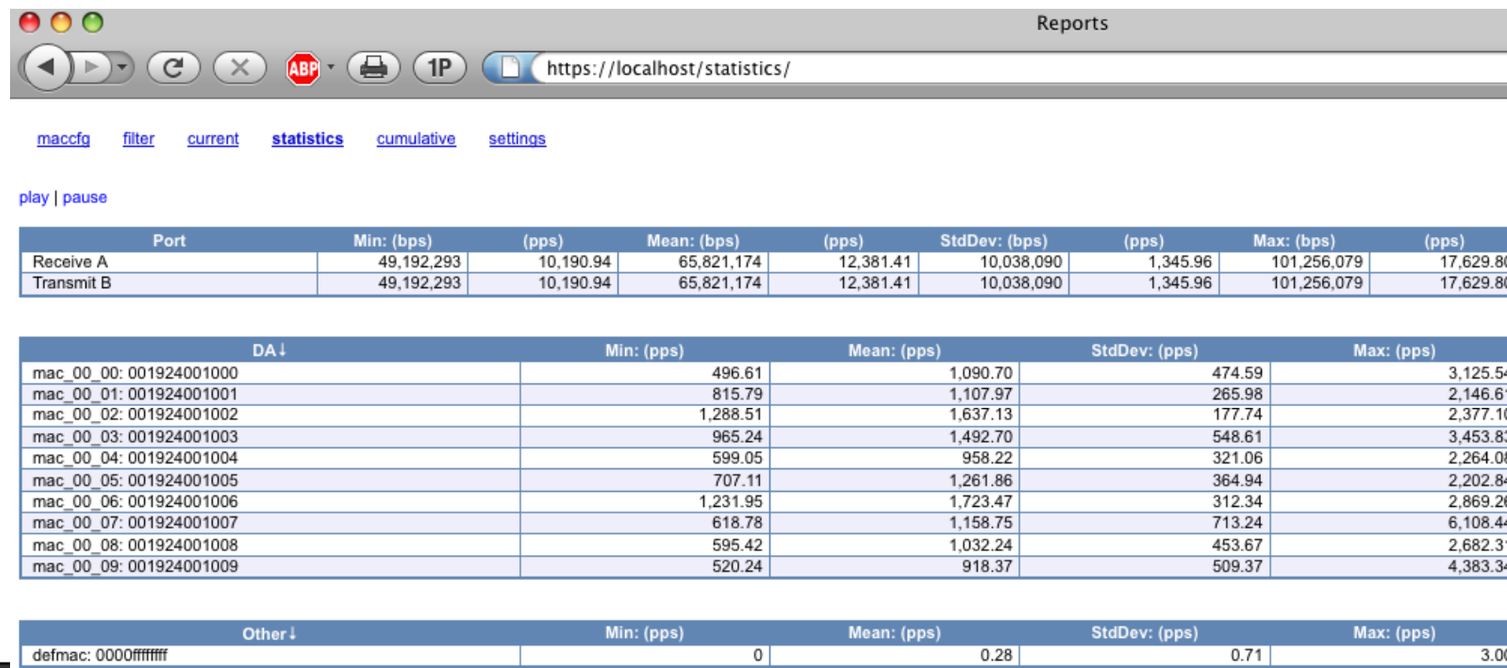


A Production Load-Balancer

A Production Load-Balancer

cFlow: 10GE line-rate, stand-alone load-balancer

10 Gb/s in/out
Web & CLI
Filtering capabilities

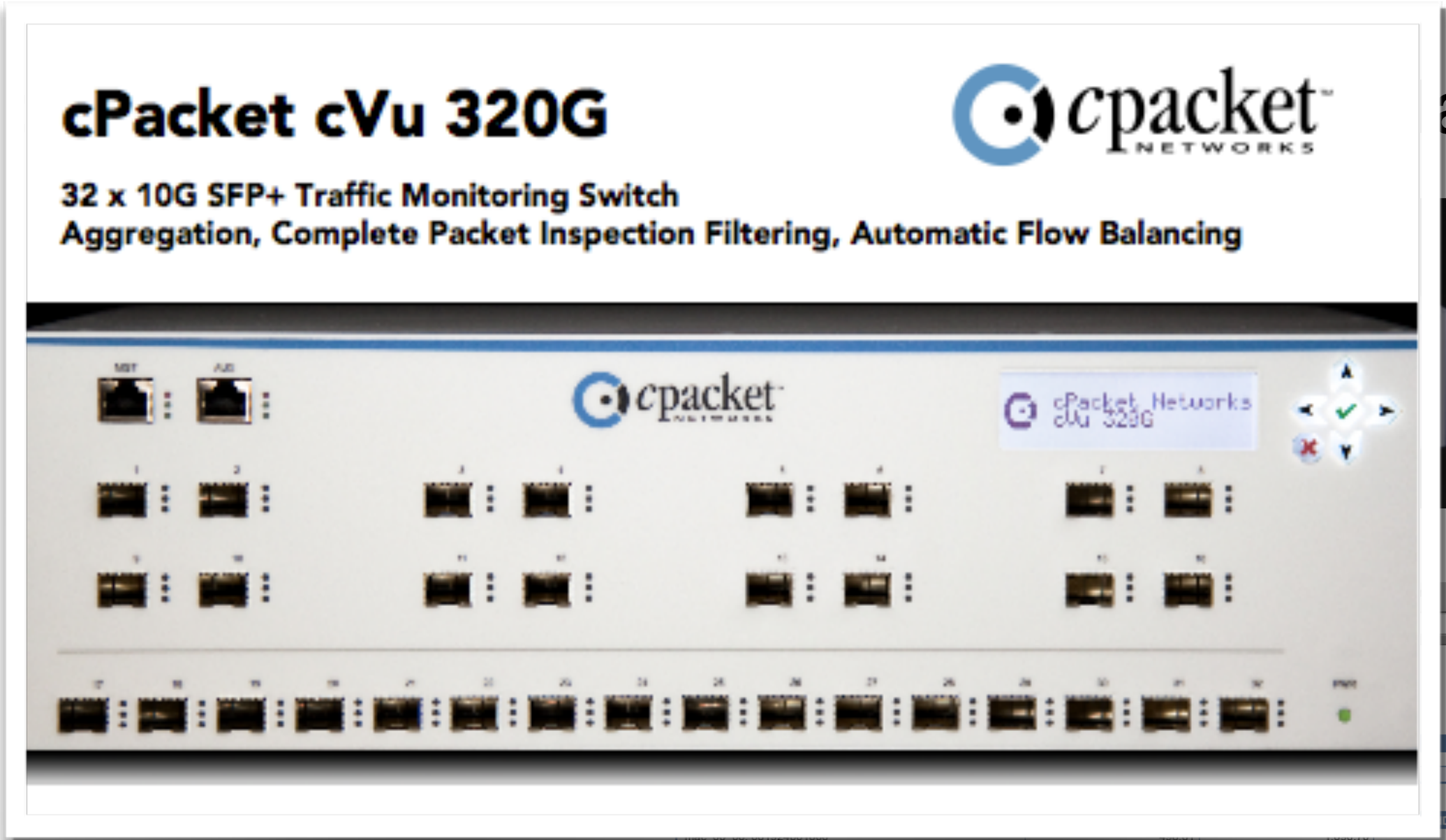


Port	Min: (bps)	(pps)	Mean: (bps)	(pps)	StdDev: (bps)	(pps)	Max: (bps)	(pps)
Receive A	49,192,293	10,190.94	65,821,174	12,381.41	10,038,090	1,345.96	101,256,079	17,629.8
Transmit B	49,192,293	10,190.94	65,821,174	12,381.41	10,038,090	1,345.96	101,256,079	17,629.8

DA↓	Min: (pps)	Mean: (pps)	StdDev: (pps)	Max: (pps)
mac_00_00: 001924001000	496.61	1,090.70	474.59	3,125.5
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	3.0

A Production Load-Balancer



ancer

(pps)	Max: (bps)	(pps)
1,345.96	101,256,079	17,629.8
1,345.96	101,256,079	17,629.8

			Dev: (pps)	Max: (pps)
			474.59	3,125.5
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	3.0

ICSI
INTERNATIONAL COMPUTER SCIENCE INSTITUTE

Net

cpacket NETWORKS

Next Stop: 100 Gb/s

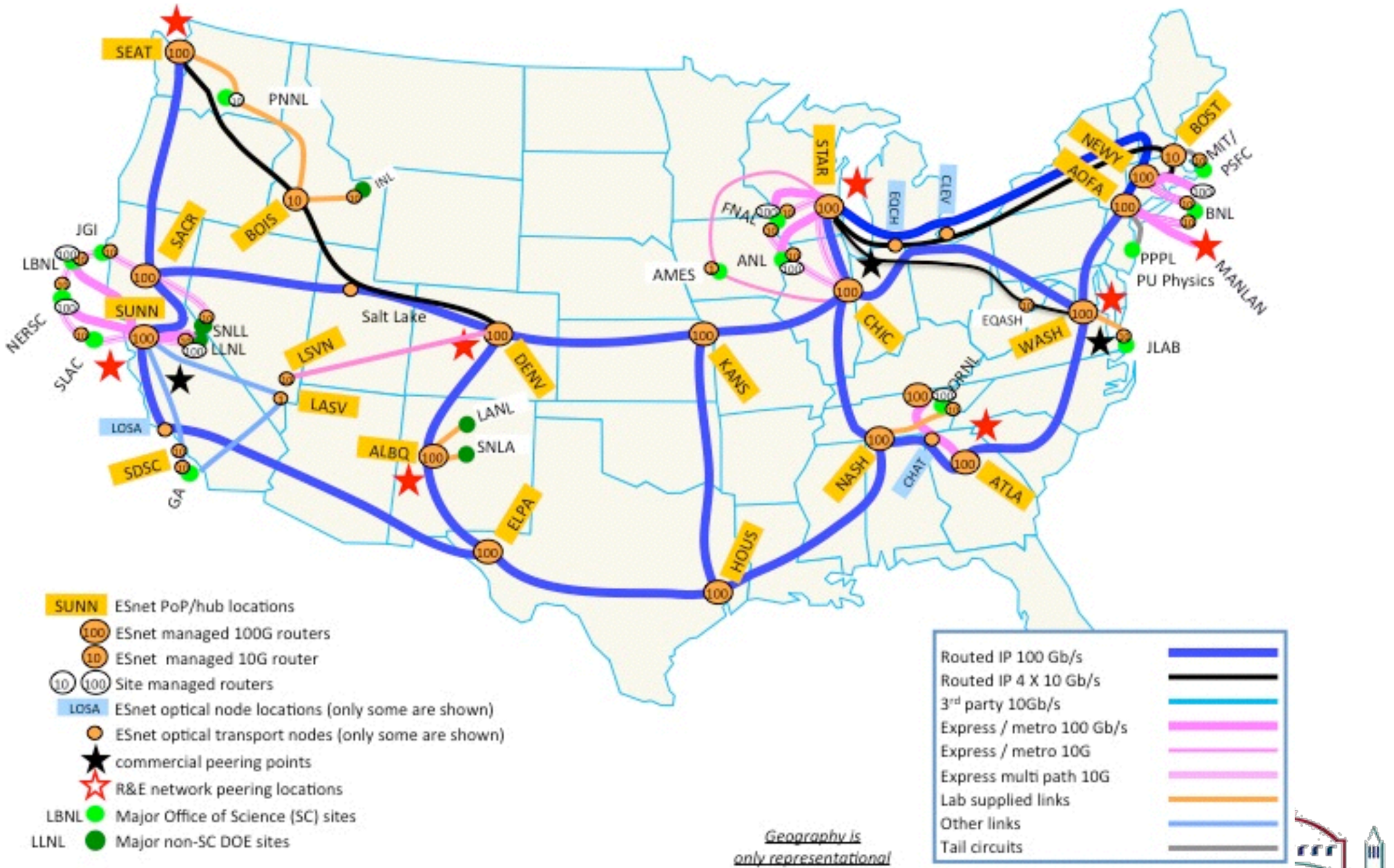
2011

DOE/ESNet
100G Advanced Networking Initiative

Source: ESNet

Next Stop: 100 Gb/s

2014



Source: ESNet

On Deck: 400G Connectivity

Berkeley National Laboratory

Computational Research and Theory Building.

WAN 100G

ESnet Energy Sciences Network

File System Links

2 x 100G

Inter-site Traffic

100G

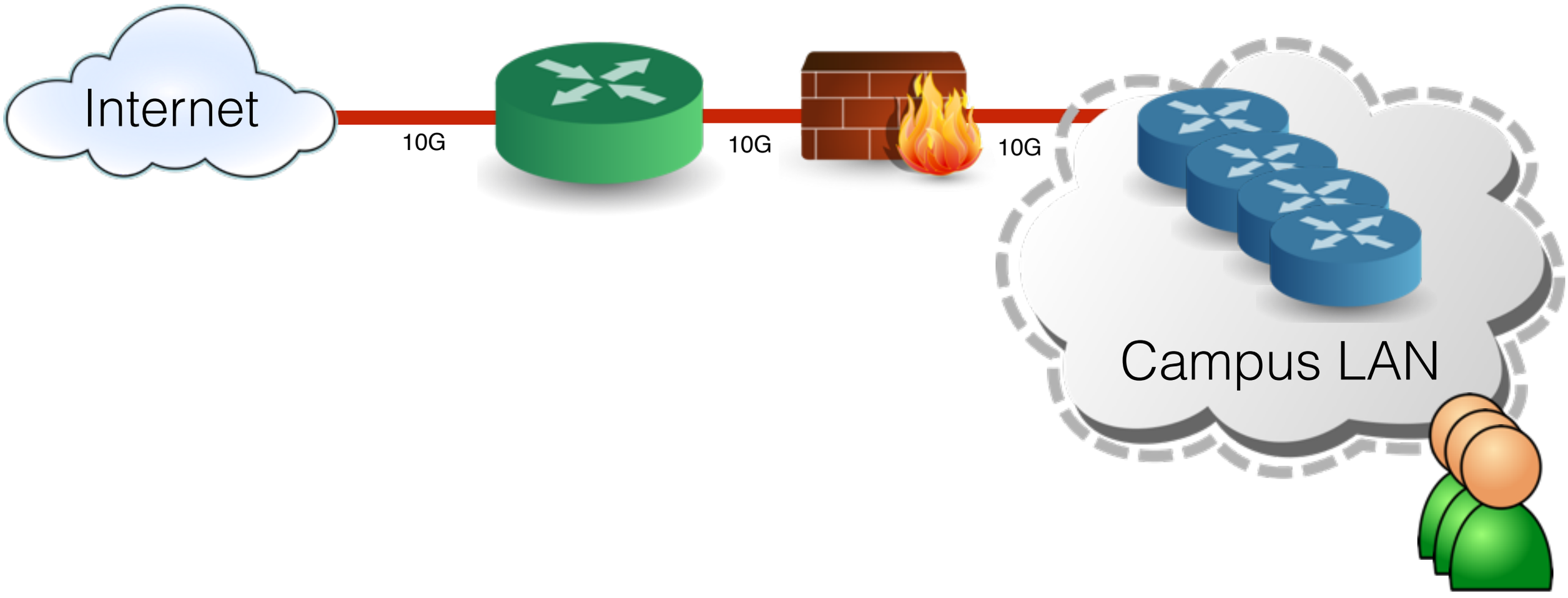
WAN 100G

ESnet Energy Sciences Network

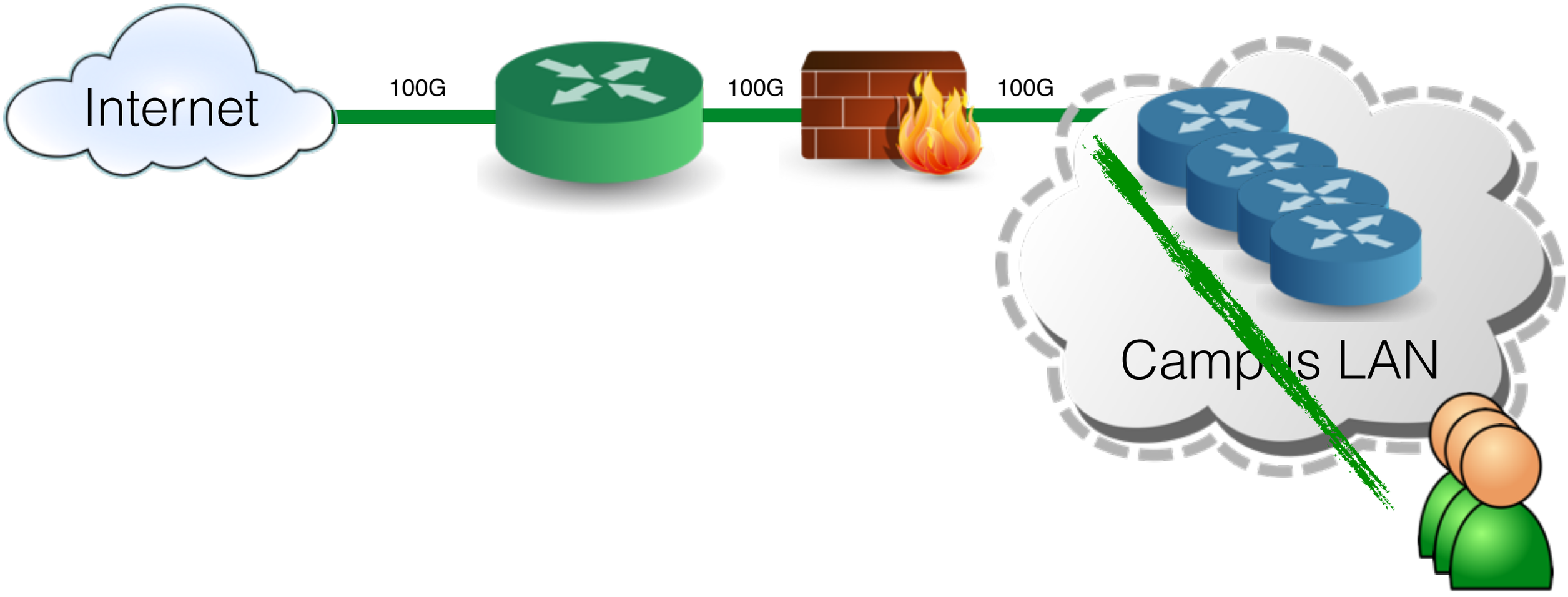
Oakland Scientific Facility.

Sources: ESNet/LBNL/NERSC

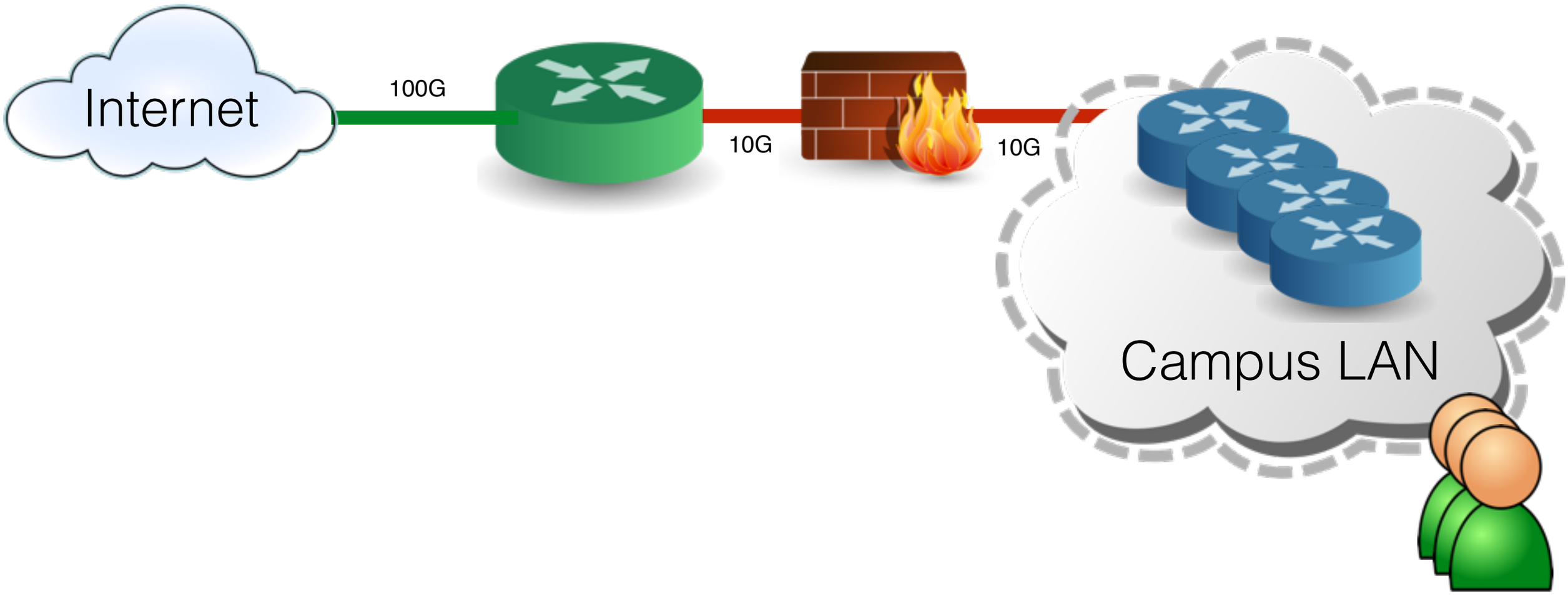
Science DMZ



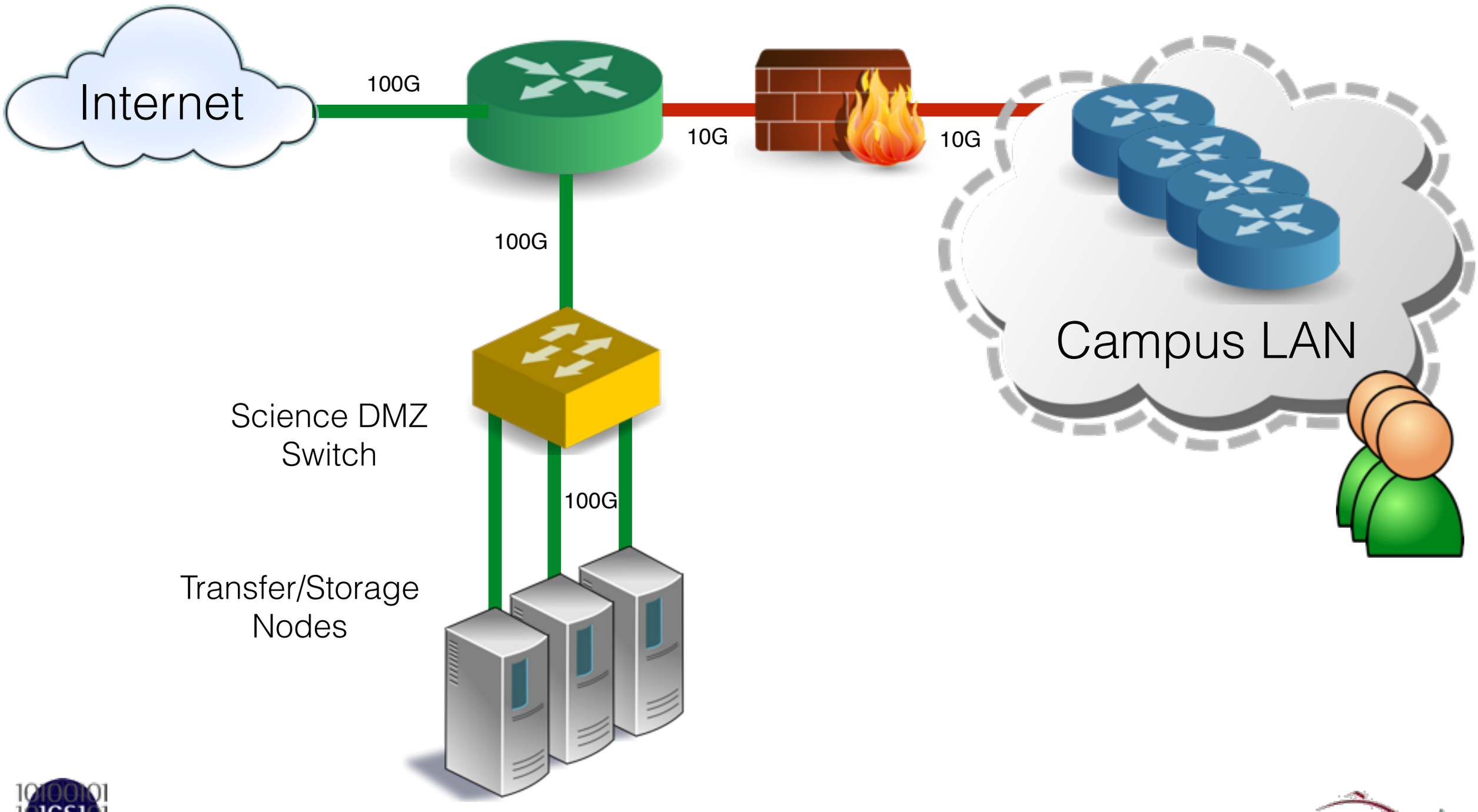
Science DMZ



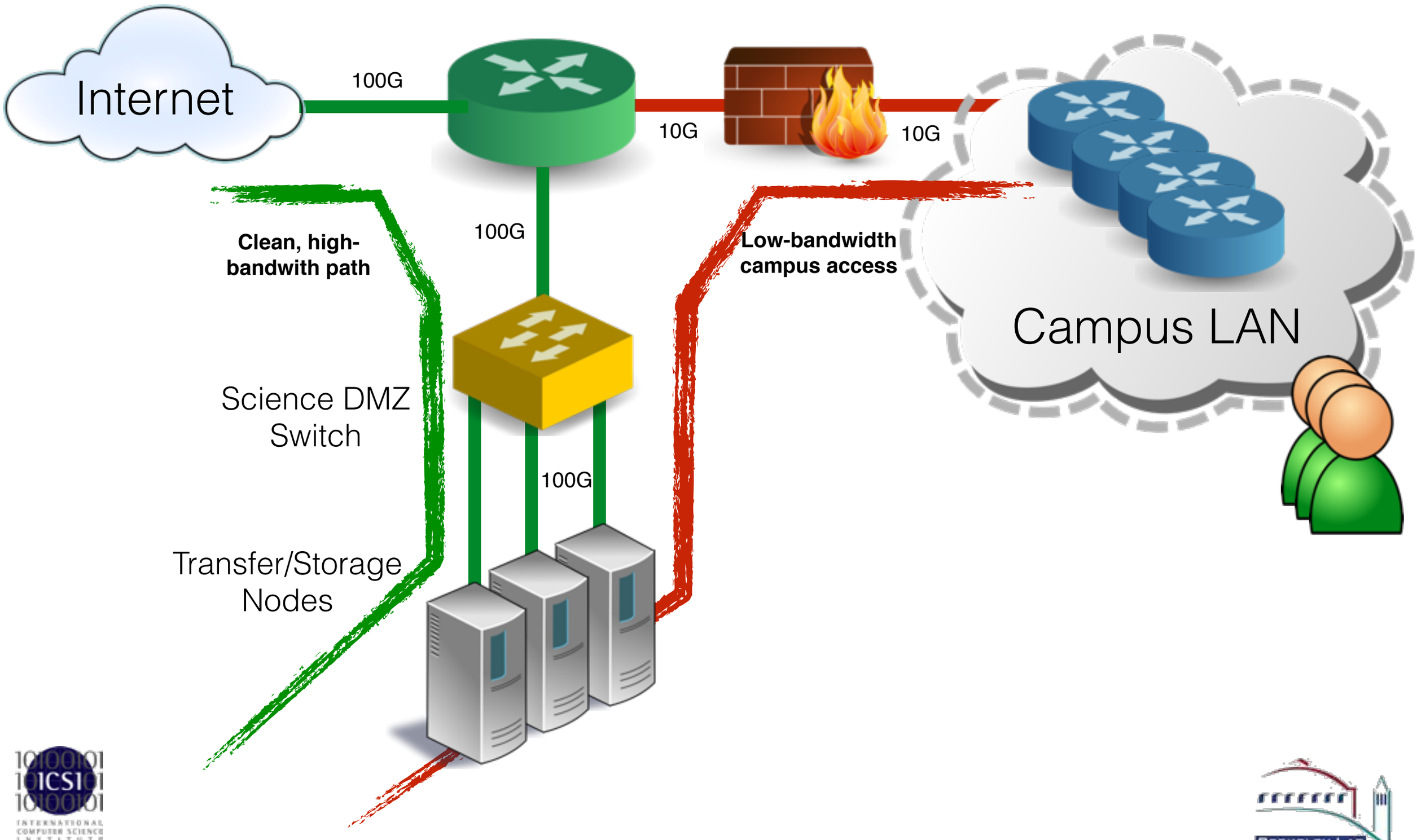
Science DMZ



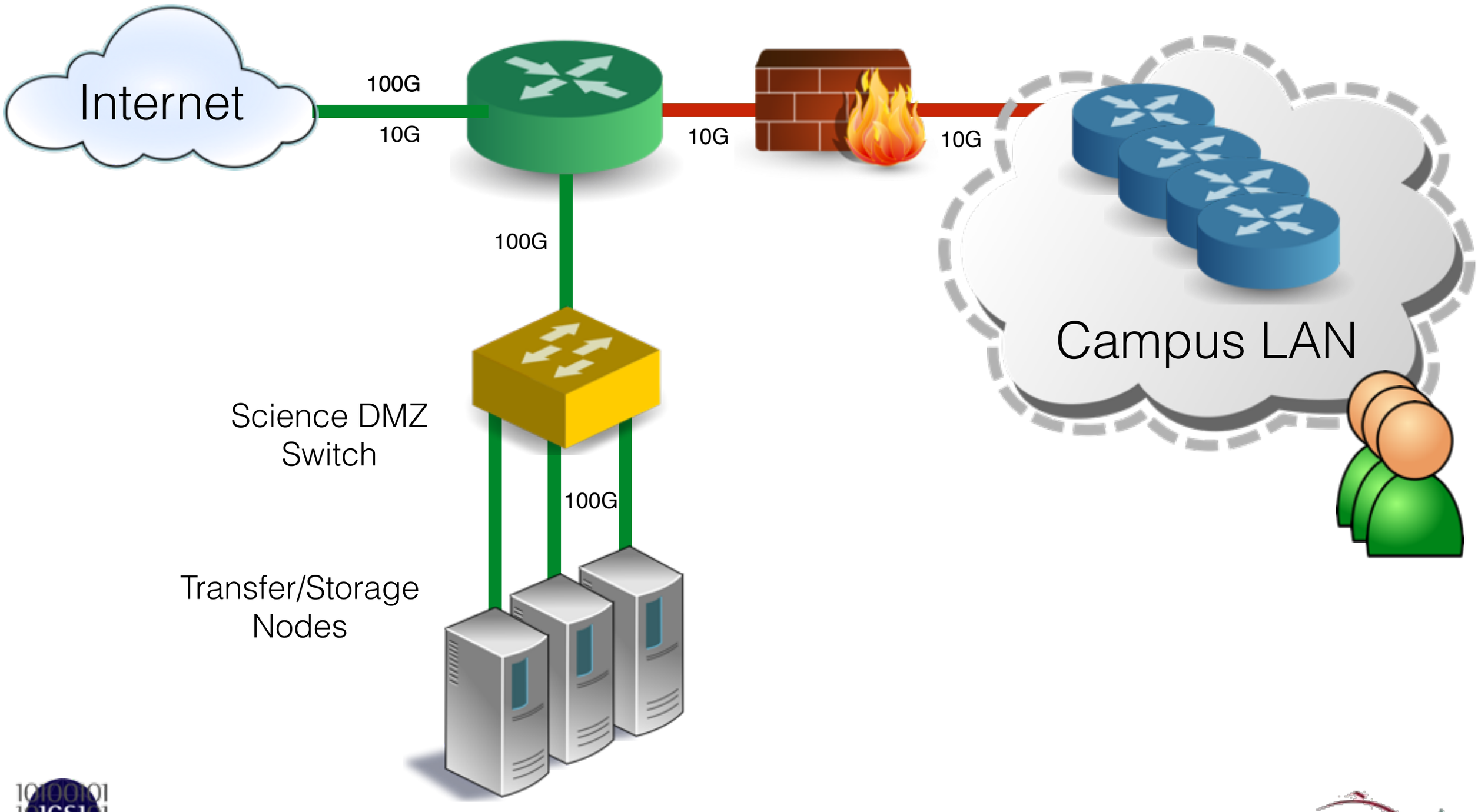
Science DMZ


Internet
100G
10G
10G
100G
Campus LAN
Science DMZ
Switch
100G
Transfer/Storage
Nodes

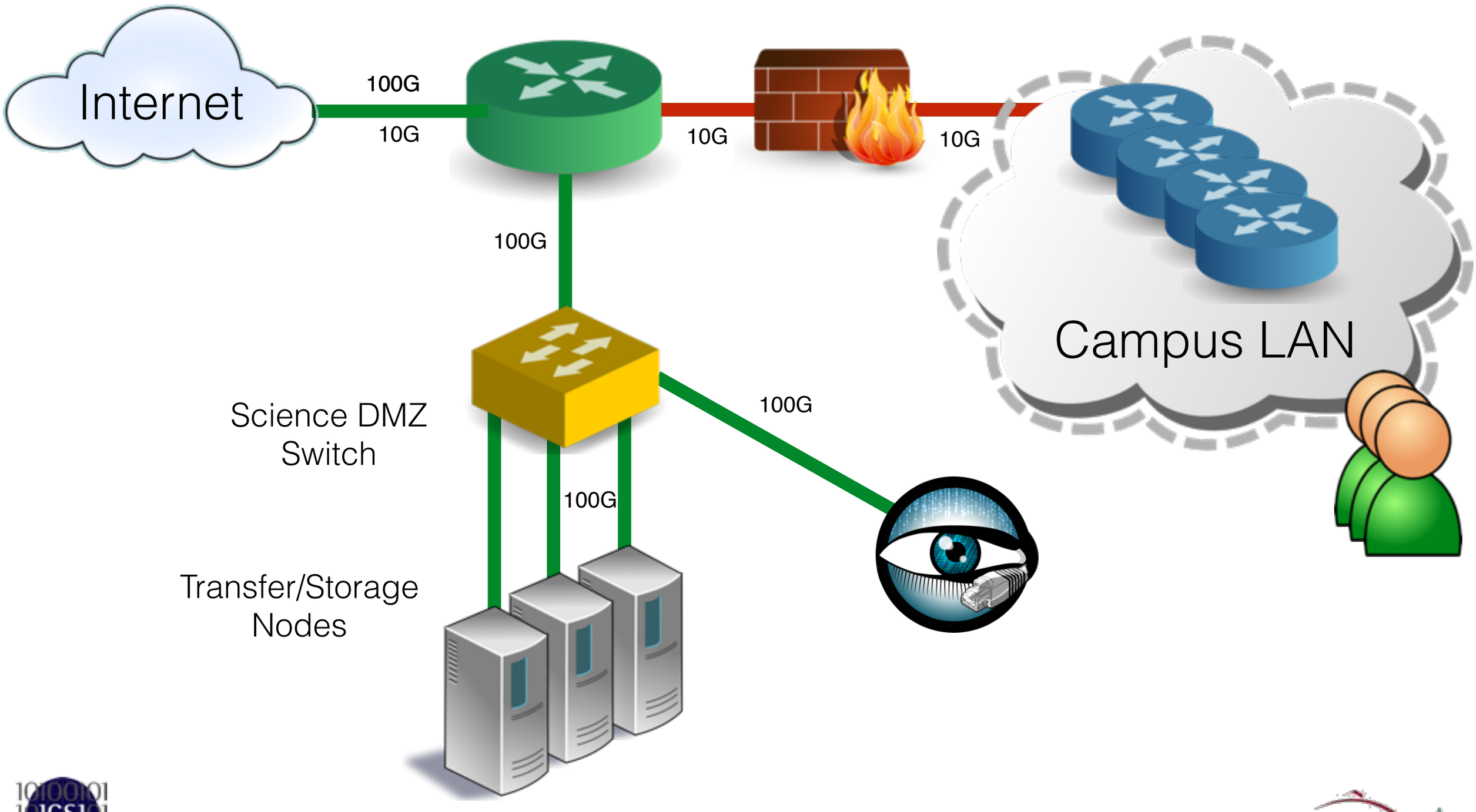
Science DMZ


Internet
100G
10G
10G
Clean, high-bandwith path
100G
Low-bandwidth campus access
Campus LAN
Science DMZ Switch
100G
Transfer/Storage Nodes

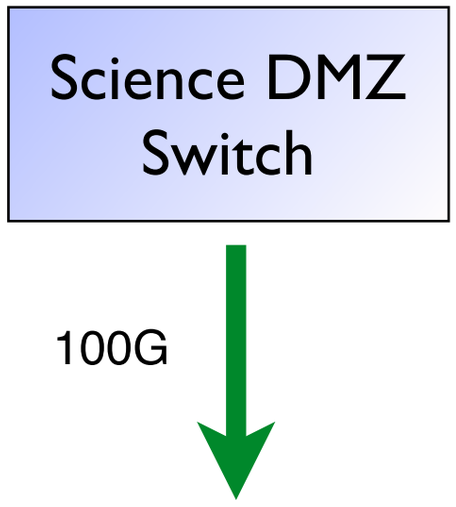
Science DMZ


Internet
100G
10G
10G
10G
100G
Campus LAN
Science DMZ
Switch
100G
Transfer/Storage
Nodes

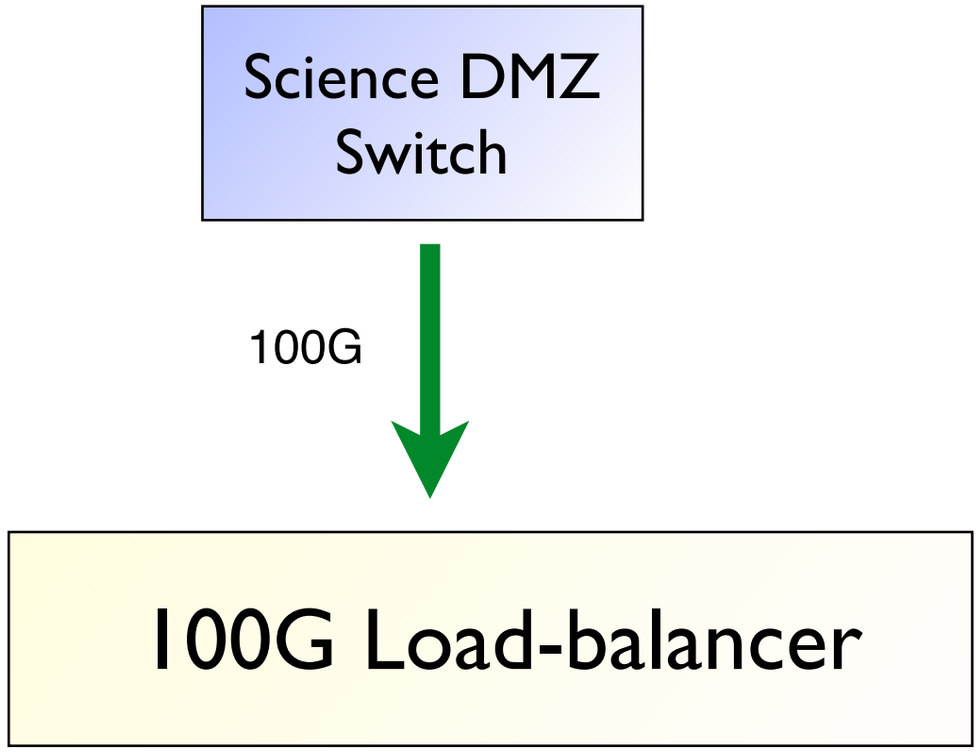
Science DMZ


Internet
100G
10G
10G
10G
100G
Campus LAN
Science DMZ
Switch
100G
100G
Transfer/Storage
Nodes

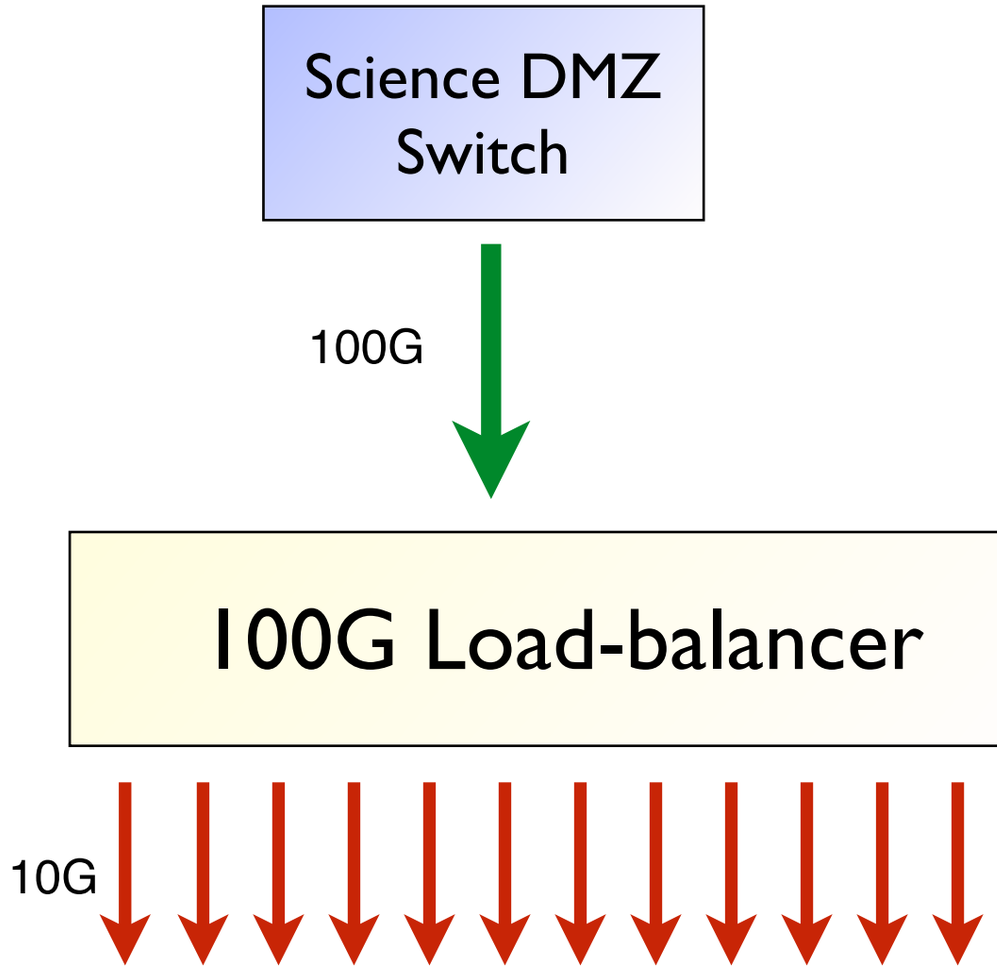
100G Bro Cluster



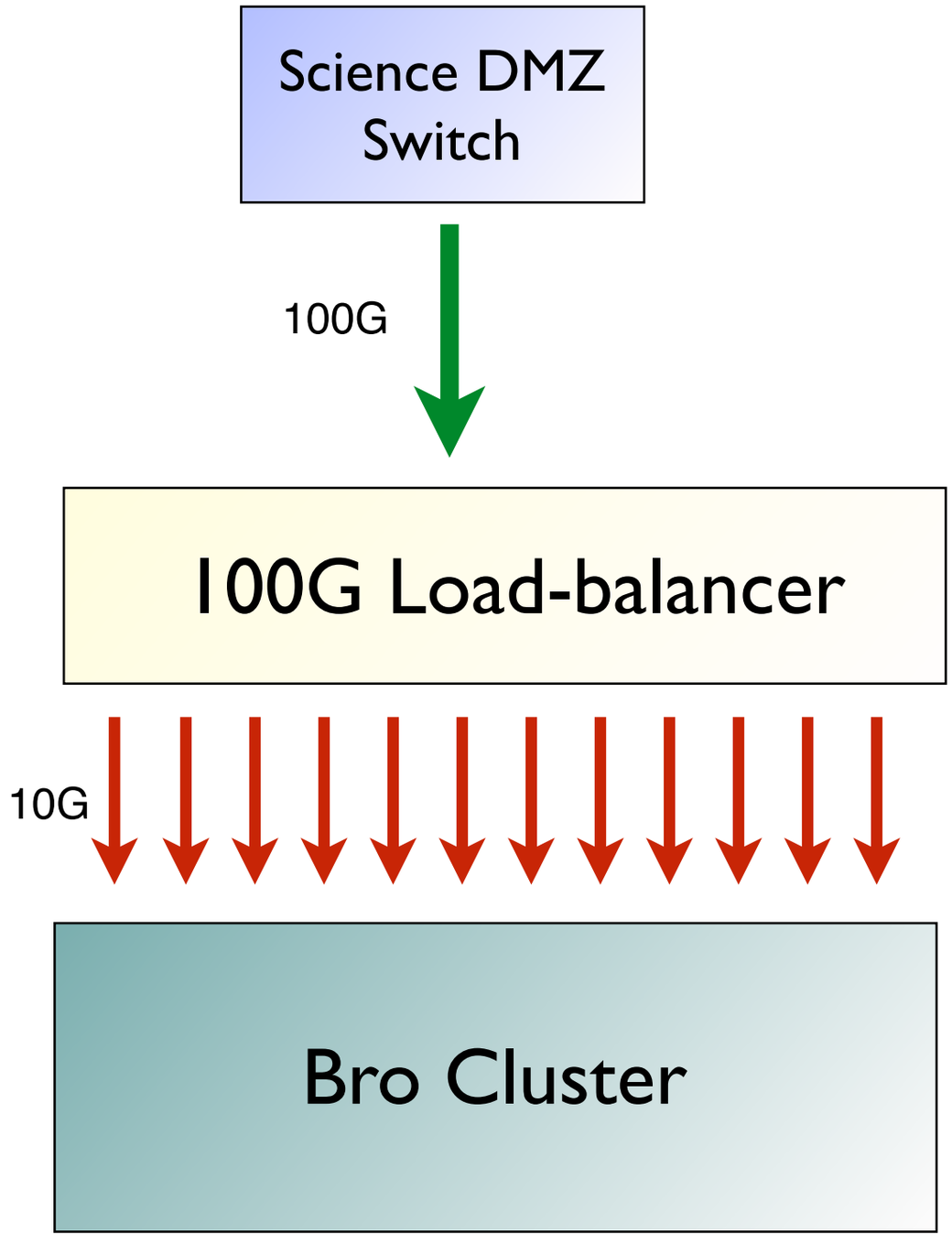
100G Bro Cluster



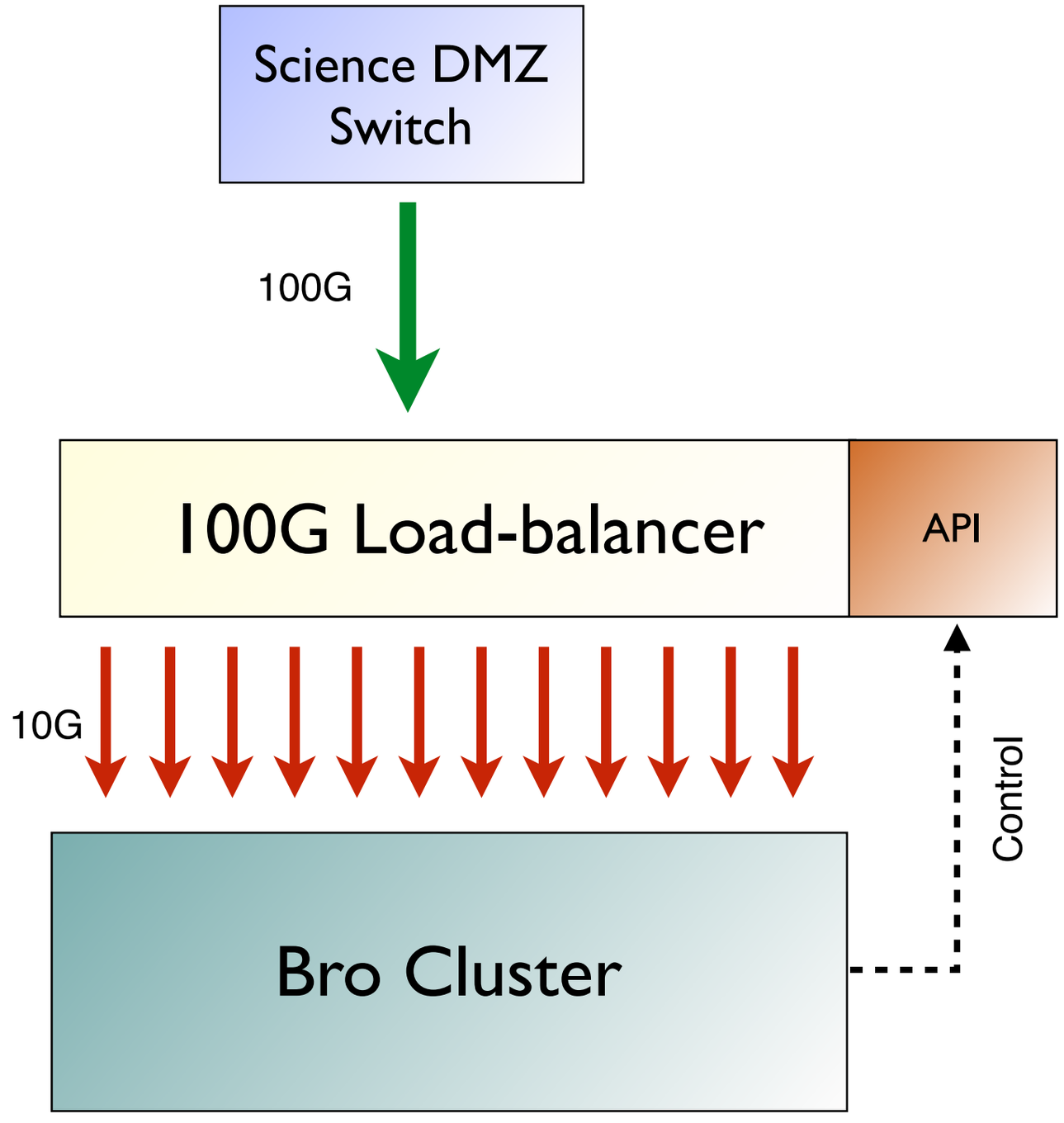
100G Bro Cluster



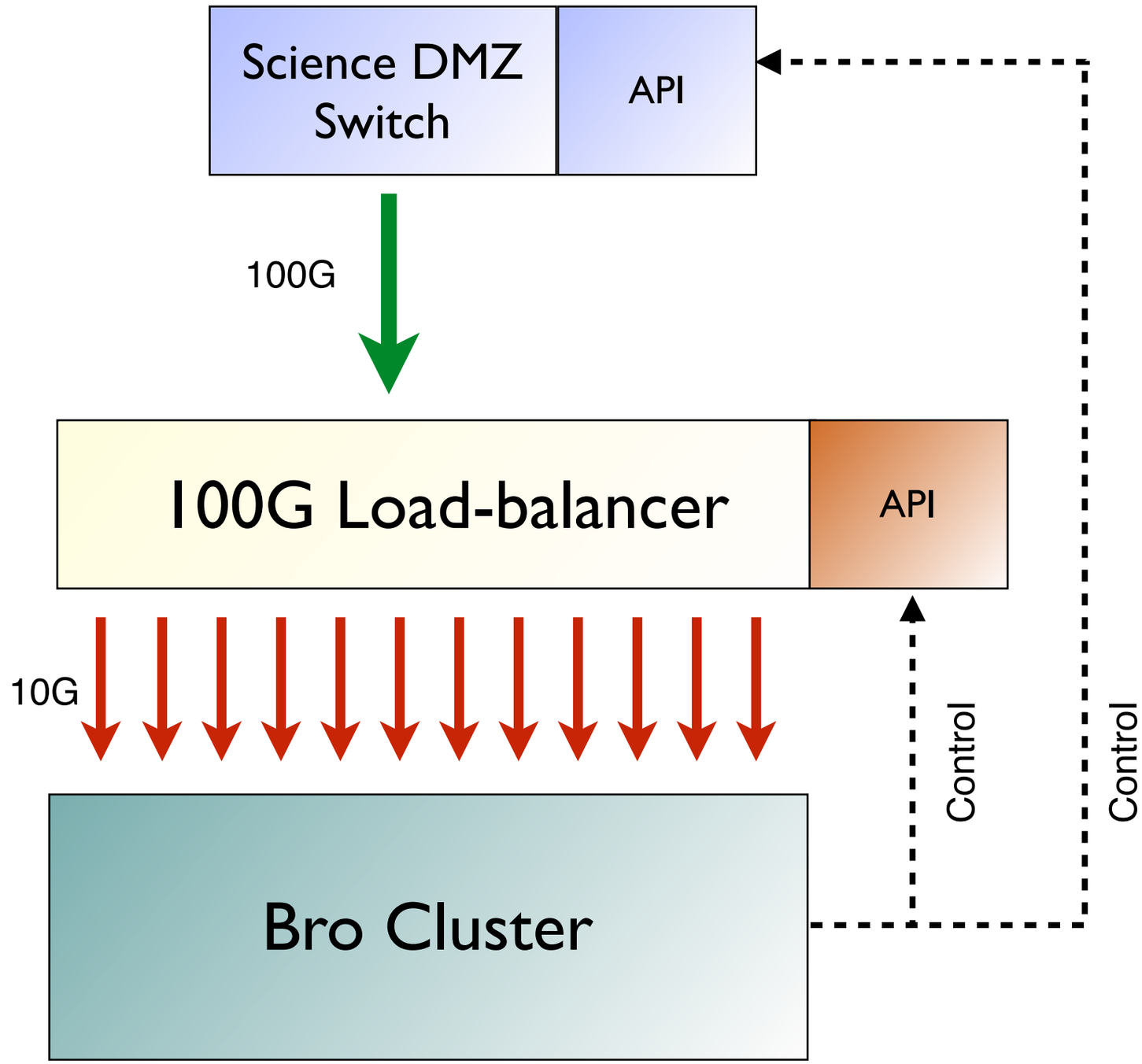
100G Bro Cluster



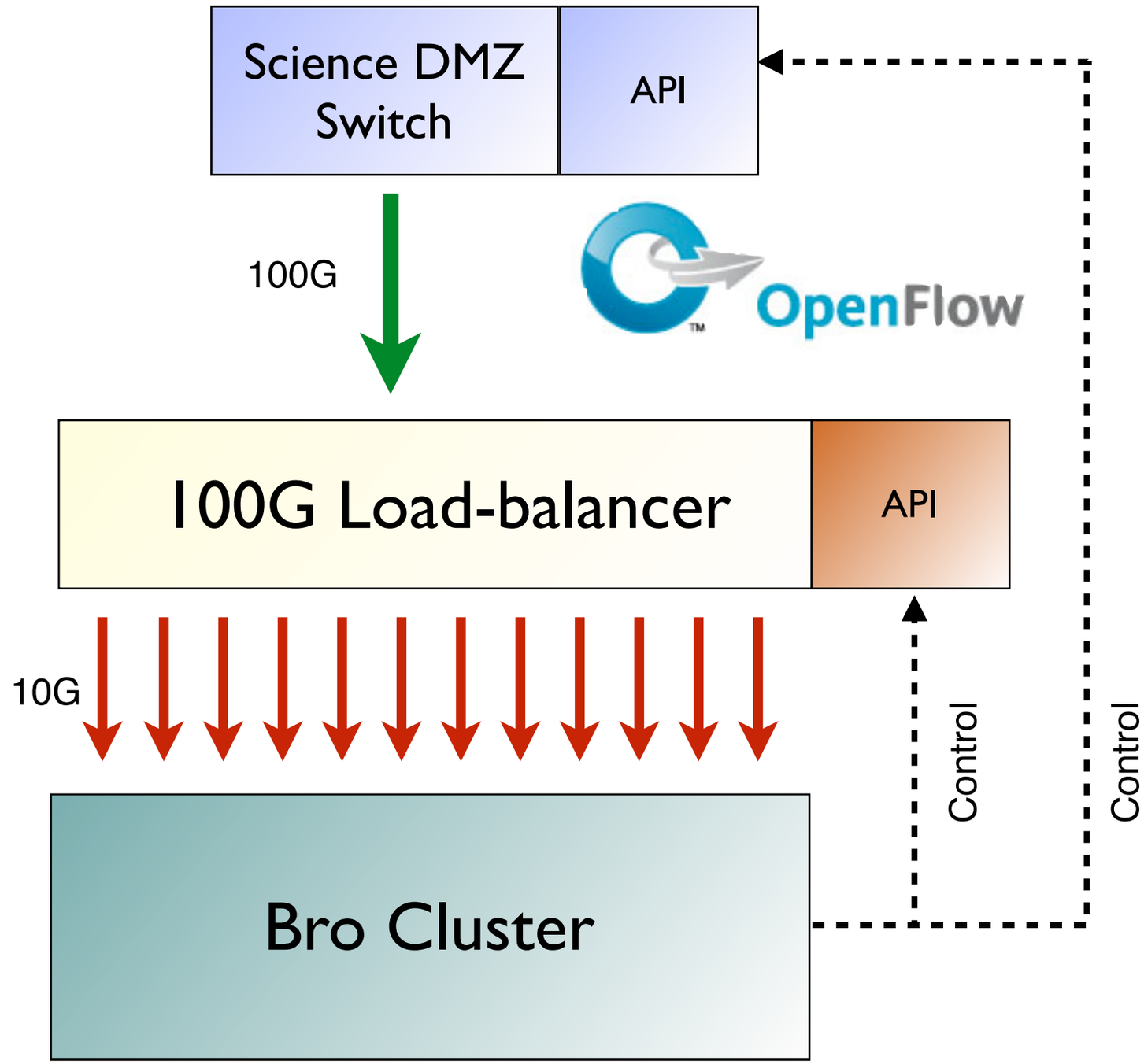
100G Bro Cluster



100G Bro Cluster



100G Bro Cluster



Parallelizing DPI on Multi-core Systems

Going Multi-Core …

Bro is single-threaded

- Cluster backends have muitple cores, mostly idle.
- Work-around: “Cluster in a box”

We really want multi-threading, though.

- Needs to scale well with increasing numbers of cores.
- Needs to be transparent to the operator.

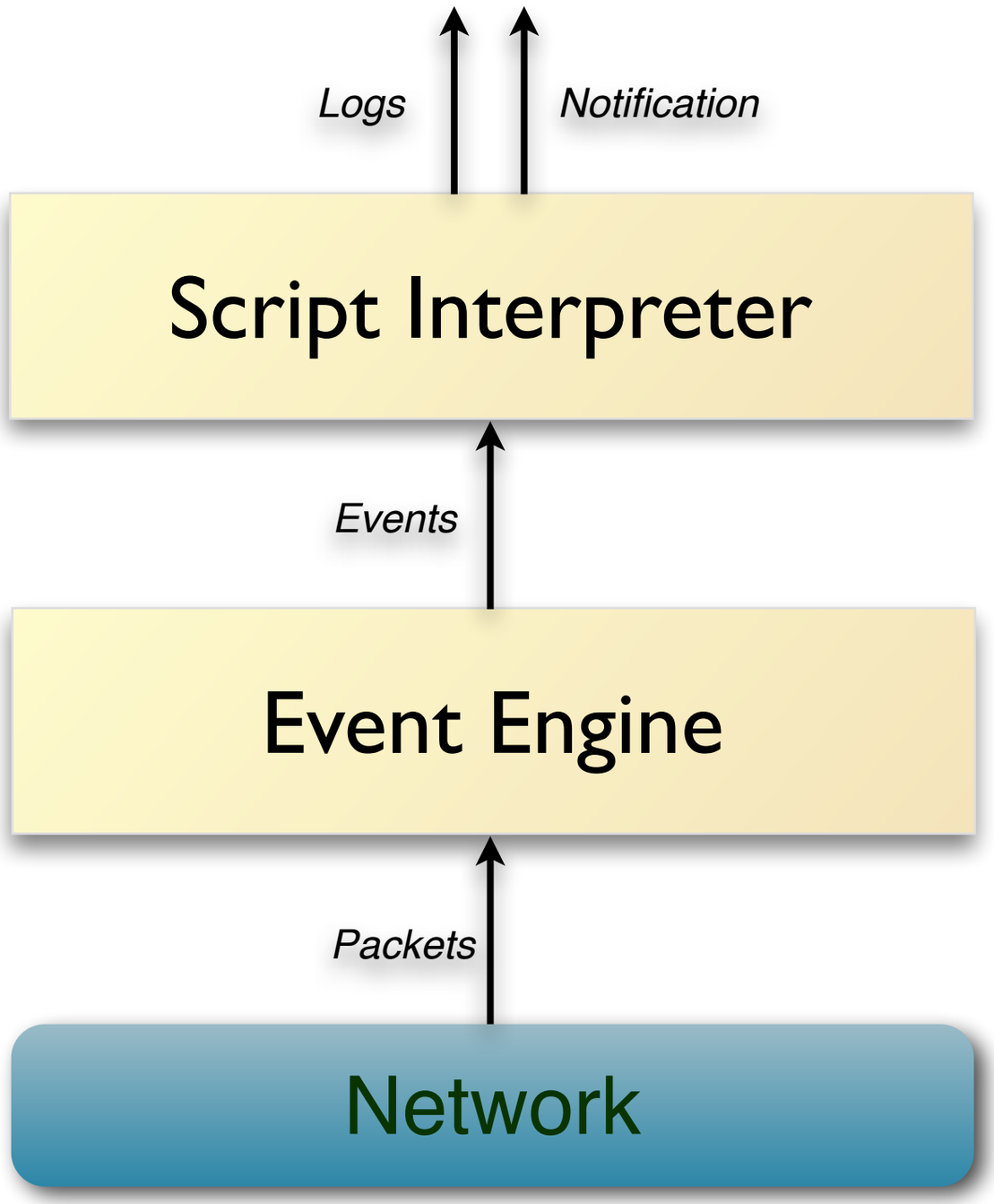
For some IDS, that’s not so hard.

- For others, it is ...

Concurrent Analysis

Analysis Logic

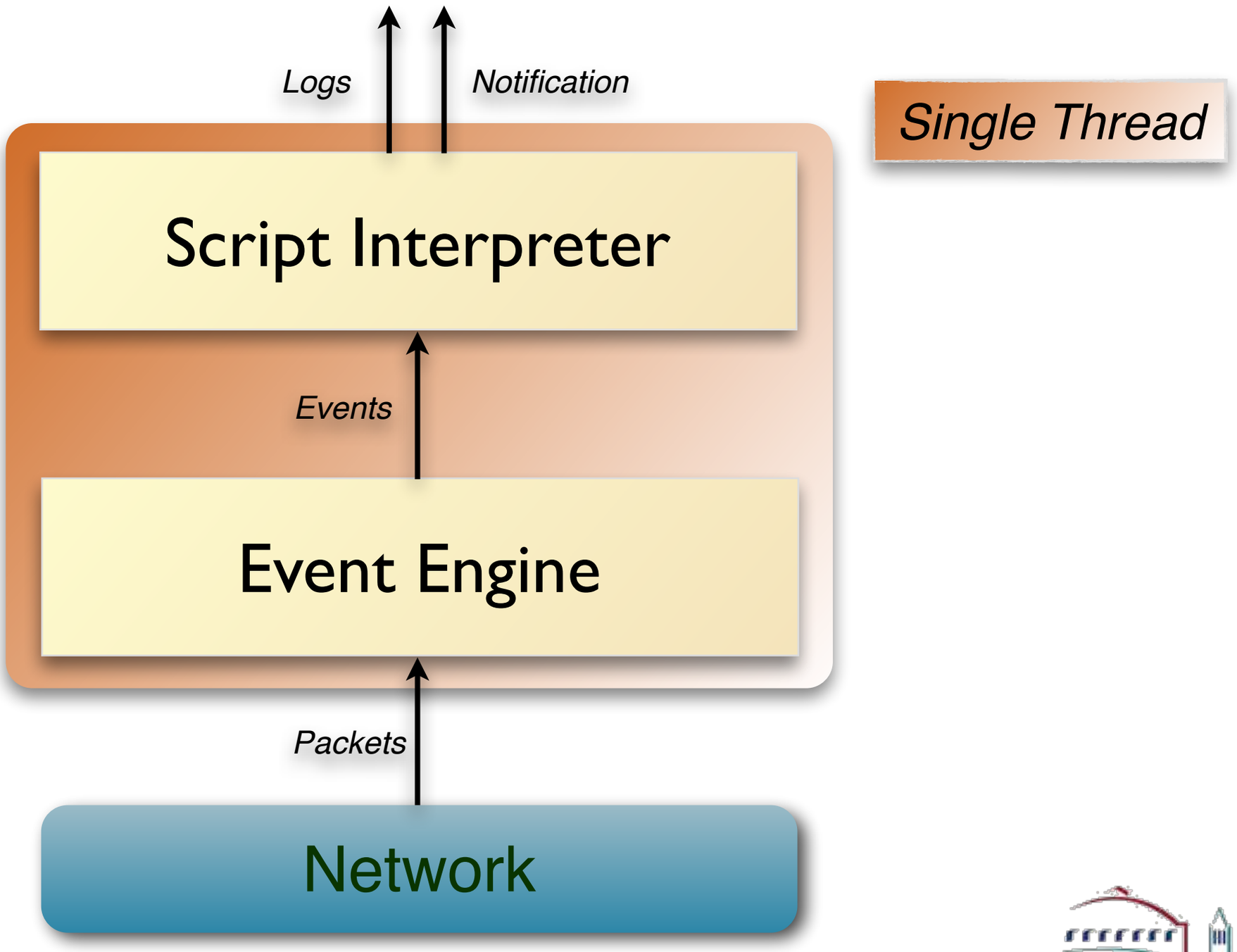
Protocol Decoding



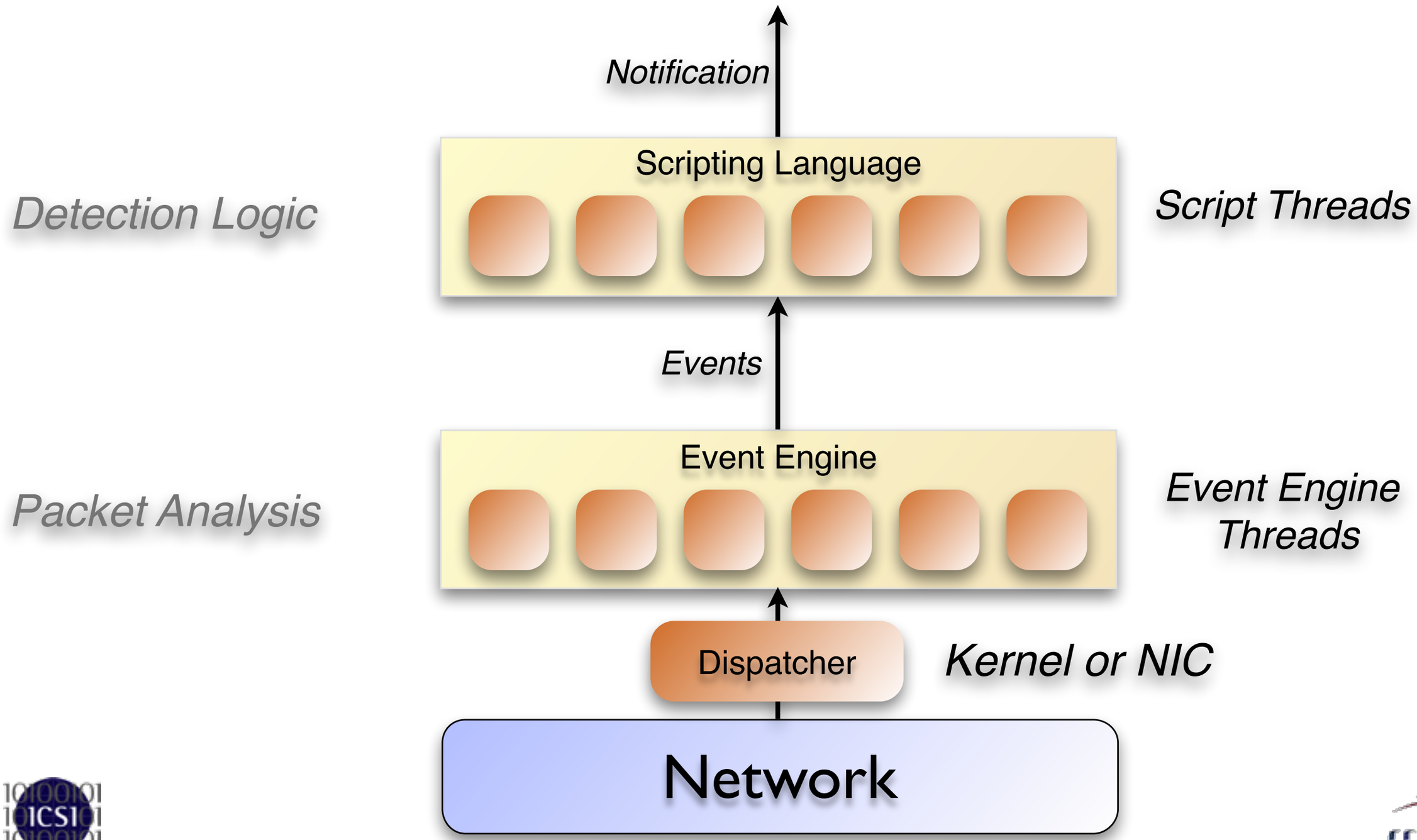
Concurrent Analysis

Analysis Logic

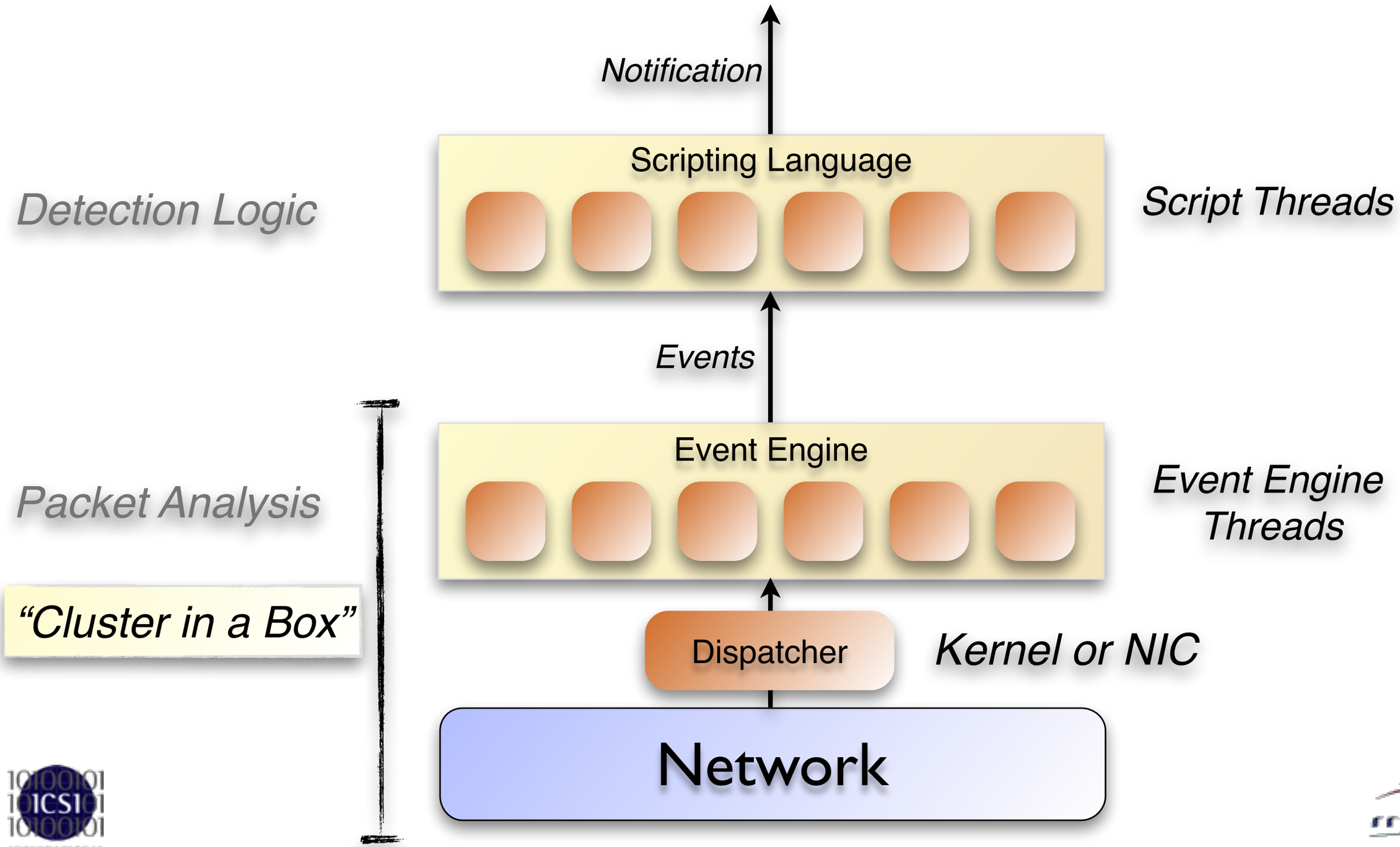
Protocol Decoding



Concurrent Analysis



Concurrent Analysis



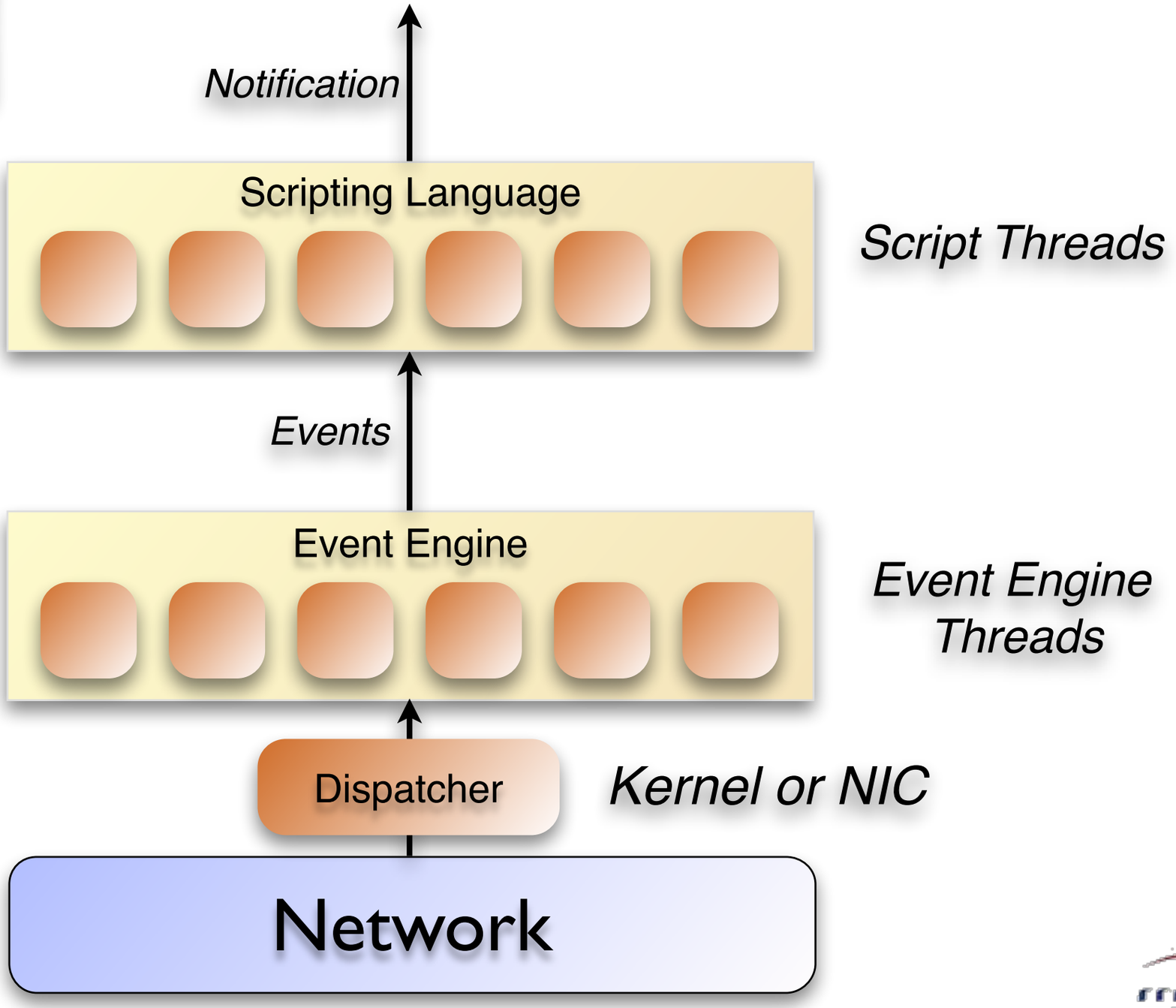
Concurrent Analysis

How to parallelize a scripting language?

Detection Logic

Packet Analysis

"Cluster in a Box"



Script Threads

Event Engine Threads

Kernel or NIC

How to Parallelize Event Handlers?

Simple: State-less Analysis

How to Parallelize Event Handlers?

Simple: State-less Analysis

```
event http_request(c: connection,           # Connection.
                   method: string,          # HTTP method.
                   original_URI: string,    # Requested URL.
                   unescaped_URI: string, # Decoded URL.
                   version: string)         # HTTP version.
{
    if ( method == "GET" && unescaped_URI == /.*passwd/ )
        NOTICE(...); # Alarm.
}
```

How to Parallelize Event Handlers? (2)

Challenging: Analysis that keeps global state.

How to Parallelize Event Handlers? (2)

Challenging: Analysis that keeps global state.

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

event connection_rejected(c: connection)
{
    local orig = c$id$orig_h;      # Get originator address.

    local n = ++attempts[orig];    # Increase counter.

    if ( n == SOME_THRESHOLD )     # Check for threshold.
        NOTICE(...);               # Alarm.
}
```

Parallelizing Event Execution

`attempts[addr] of count`

addr	count
131.234.142.33	12
134.96.7.179	32
141.142.192.147	71
192.150.187.12	8
128.3.41.105	555
131.159.15.49	1

```
connection_rejected(c):

  s = c.originator

  ++attempts[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

addr	count
131.234.142.33	12
134.96.7.179	32
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	1

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

addr	count
131.234.142.33	12
134.96.7.179	**32**
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	**1**

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

addr	count
131.234.142.33	12
134.96.7.179	**32**
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	**1**

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator
  LOCK(attempts)
  ++attempts[s]
  UNLOCK(attempts)
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator
  LOCK(attempts)
  ++attempts[s]
  UNLOCK(attempts)
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator
  LOCK(attempts)
  ++attempts[s]
  UNLOCK(attempts)
```

Parallelizing Event Execution

`attempts[addr] of count`

addr	count
131.234.142.33	12
134.96.7.179	**32**
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	**1**

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

	addr	count
`attempts_1`	131.234.142.33	12
	134.96.7.179	**32**
`attempts_2`	141.142.192.147	71
	192.150.187.12	**8**
`attempts_3`	128.3.41.105	555
	131.159.15.49	**1**

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

	addr	count
`attempts_1`	131.234.142.33	12
	134.96.7.179	**32**
`attempts_2`	141.142.192.147	71
	192.150.187.12	**8**
`attempts_3`	128.3.41.105	555
	131.159.15.49	**1**

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts_1[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts_2[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts_3[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

	addr	count
`attempts_1`	131.234.142.33	12
	134.96.7.179	**32**
`attempts_2`	141.142.192.147	71
	192.150.187.12	**8**
`attempts_3`	128.3.41.105	555
	131.159.15.49	**1**

`hash: addr -> {1, 2 ,3}`

hash(addr)
1
1
2
2
3
3

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts_1[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts_2[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts_3[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

`hash: addr -> {1, 2 ,3}`

	addr	count
`attempts_1`	131.234.142.33	12
	134.96.7.179	**32**
`attempts_2`	141.142.192.147	71
	192.150.187.12	**8**
`attempts_3`	128.3.41.105	555
	131.159.15.49	**1**

hash(addr)
1
1
2
2
3
3

Thread 1

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread 2

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread 3

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts_(hash(s))[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

	addr	count
`attempts_1`	131.234.142.33	12
	134.96.7.179	**32**
`attempts_2`	141.142.192.147	71
	192.150.187.12	**8**
`attempts_3`	128.3.41.105	555
	131.159.15.49	**1**

`hash: addr -> {1, 2 ,3}`

hash(addr)
1
1
2
2
3
3

Thread `hash(s)`

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread `hash(s)`

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread `hash(s)`

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts_(hash(s))[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

`hash: addr -> {1, 2 ,3}`

Thread 1's **`attempts`**

Thread 2's **`attempts`**

Thread 3's **`attempts`**

addr	count
131.234.142.33	12
134.96.7.179	**32**
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	**1**

hash(addr)
1
1
2
2
3
3

Thread `hash(s)`

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread `hash(s)`

```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts_(hash(s))[s]
```

Thread `hash(s)`

```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts_(hash(s))[s]
```

Parallelizing Event Execution

`attempts[addr] of count`

`hash: addr -> {1, 2 ,3}`

Thread 1's **`attempts`**

Thread 2's **`attempts`**

Thread 3's **`attempts`**

addr	count
131.234.142.33	12
134.96.7.179	**32**
141.142.192.147	71
192.150.187.12	**8**
128.3.41.105	555
131.159.15.49	**1**

hash(addr)
1
1
2
2
3
3

Thread `hash(s)`

```
connection_rejected(c):
        # 134.96.7.179
  s = c.originator

  ++attempts[s]
```

Thread `hash(s)`

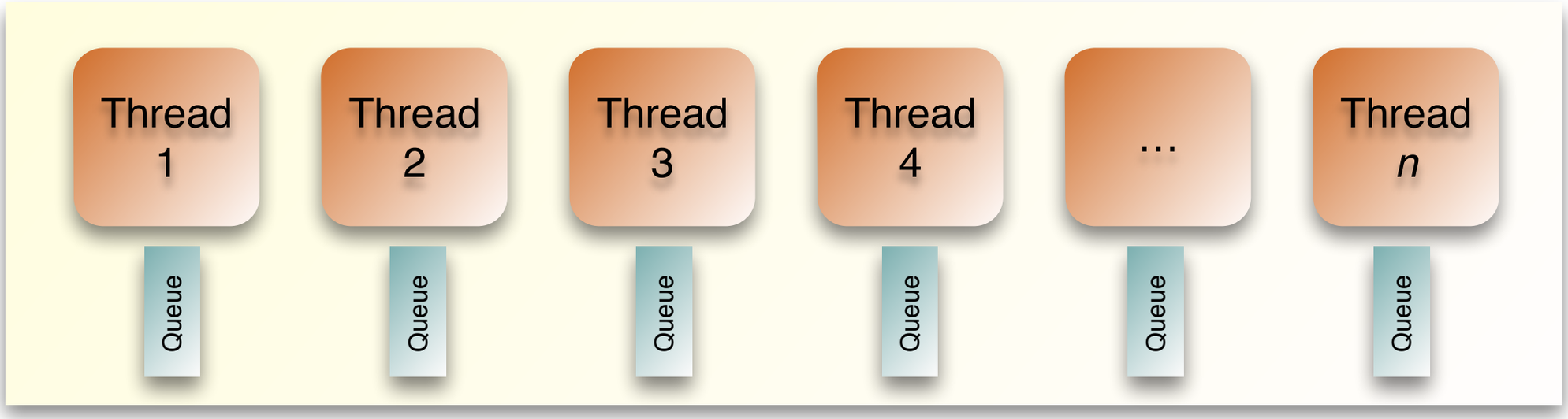
```
connection_rejected(c):
        # 192.150.187.12
  s = c.originator

  ++attempts[s]
```

Thread `hash(s)`

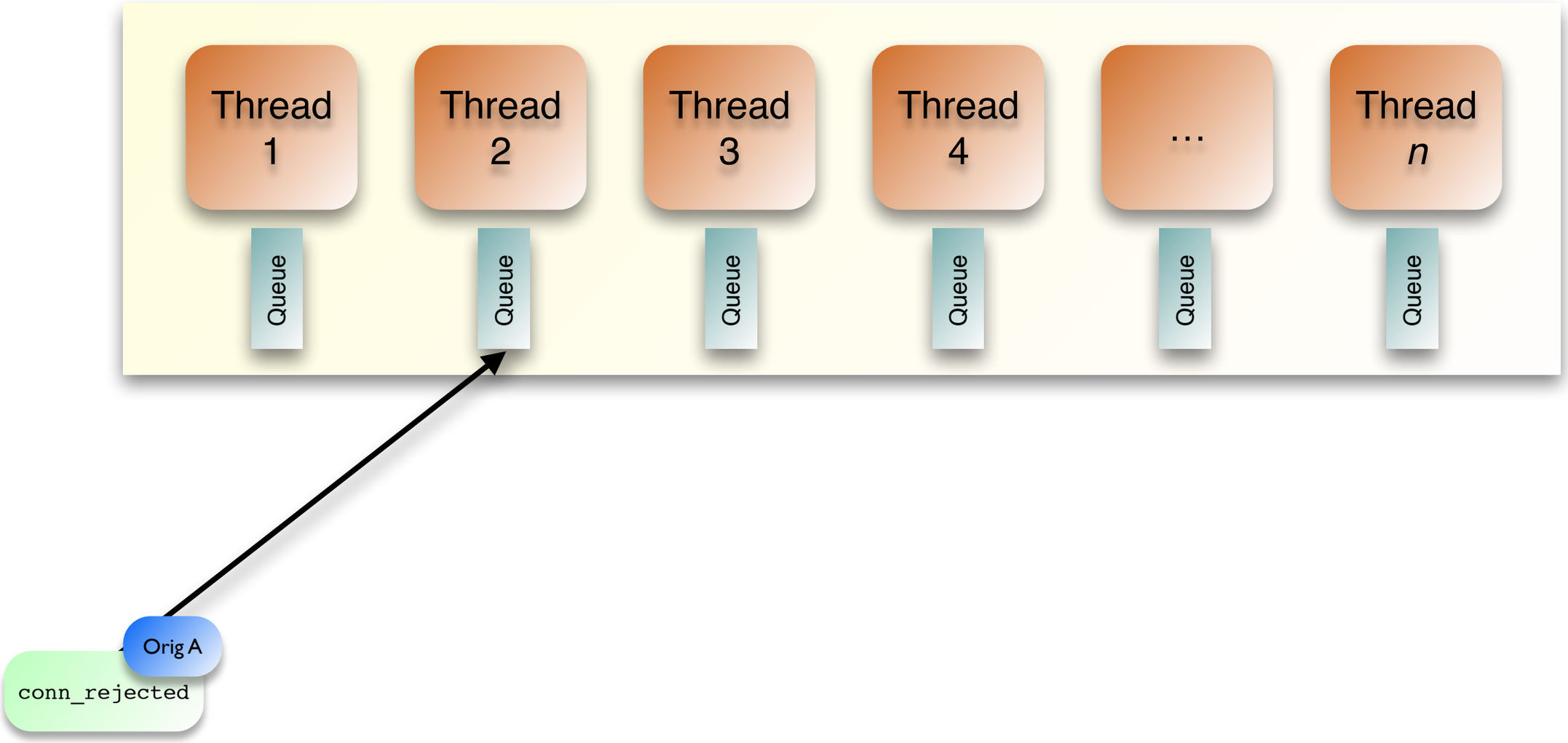
```
connection_rejected(c):
        # 131.159.15.49
  s = c.originator

  ++attempts[s]
```

Parallel Event Scheduling

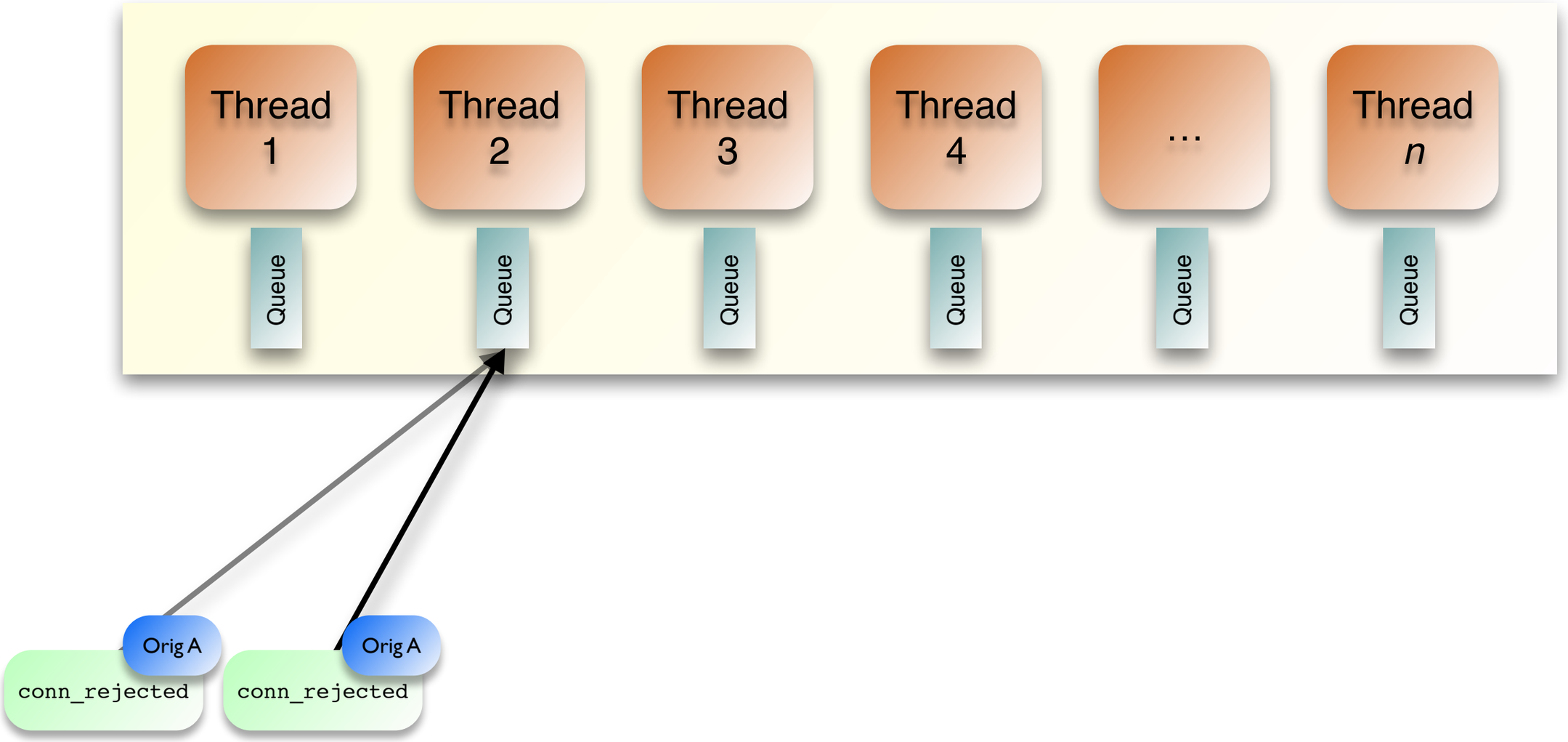
Parallel Event Scheduling



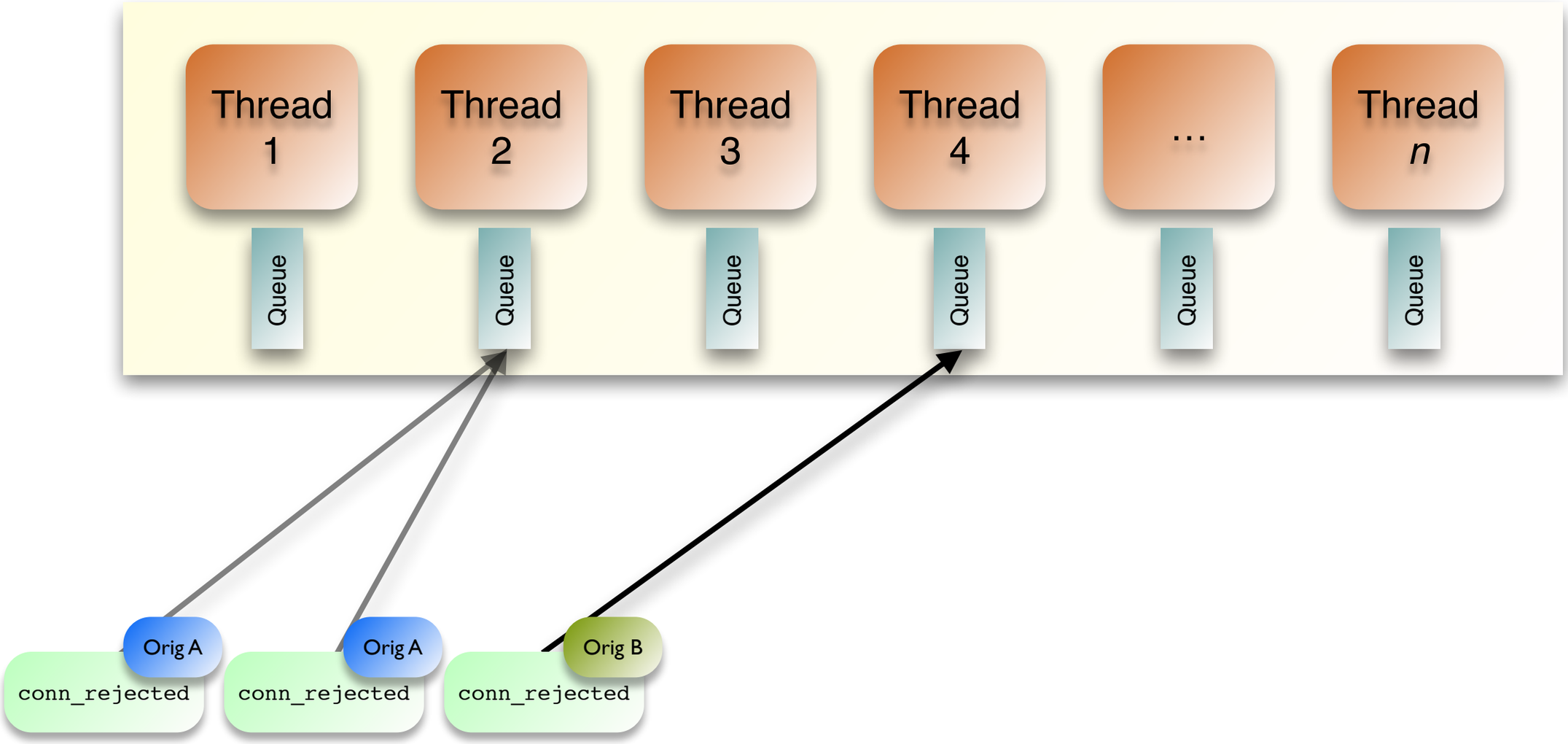
Parallel Event Scheduling



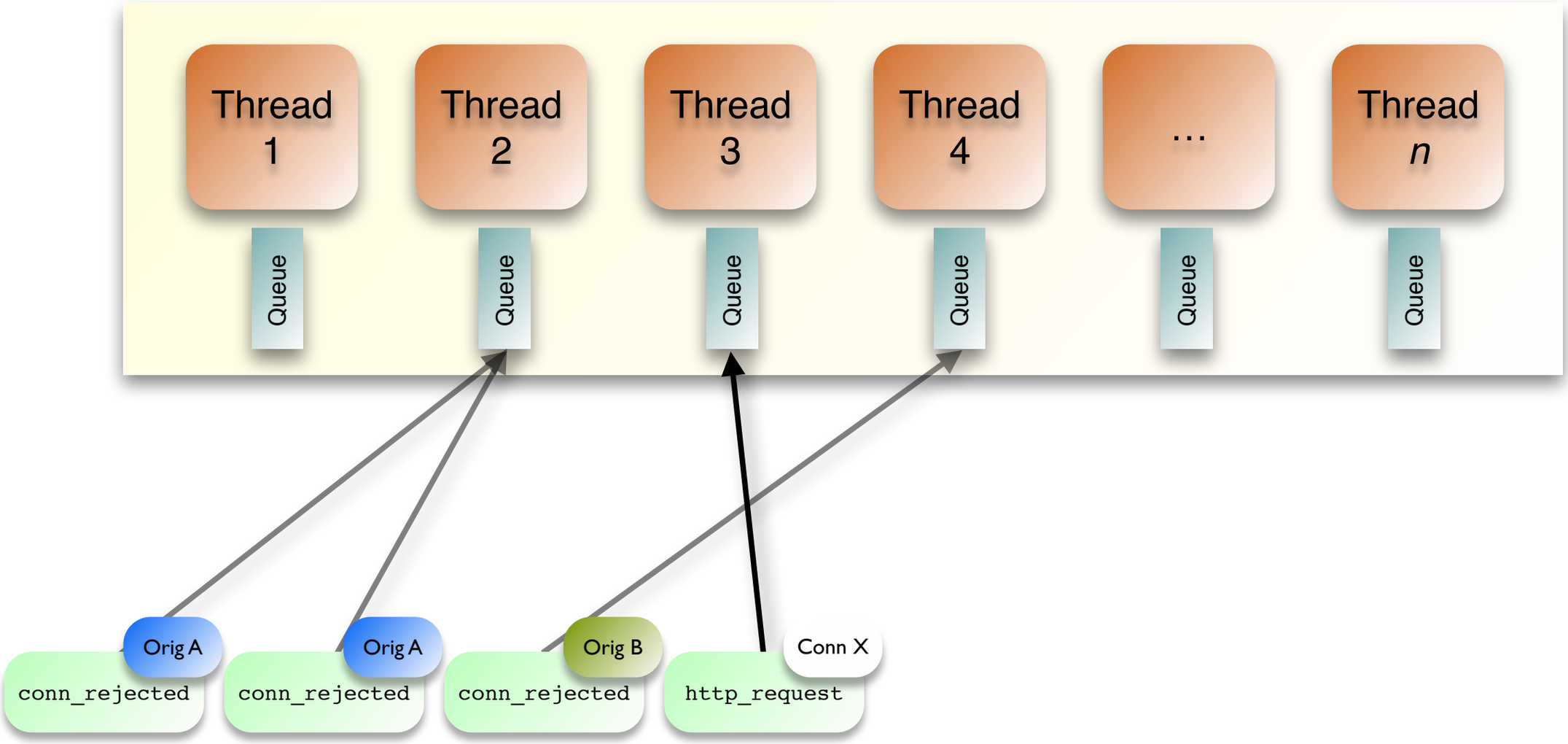
Parallel Event Scheduling



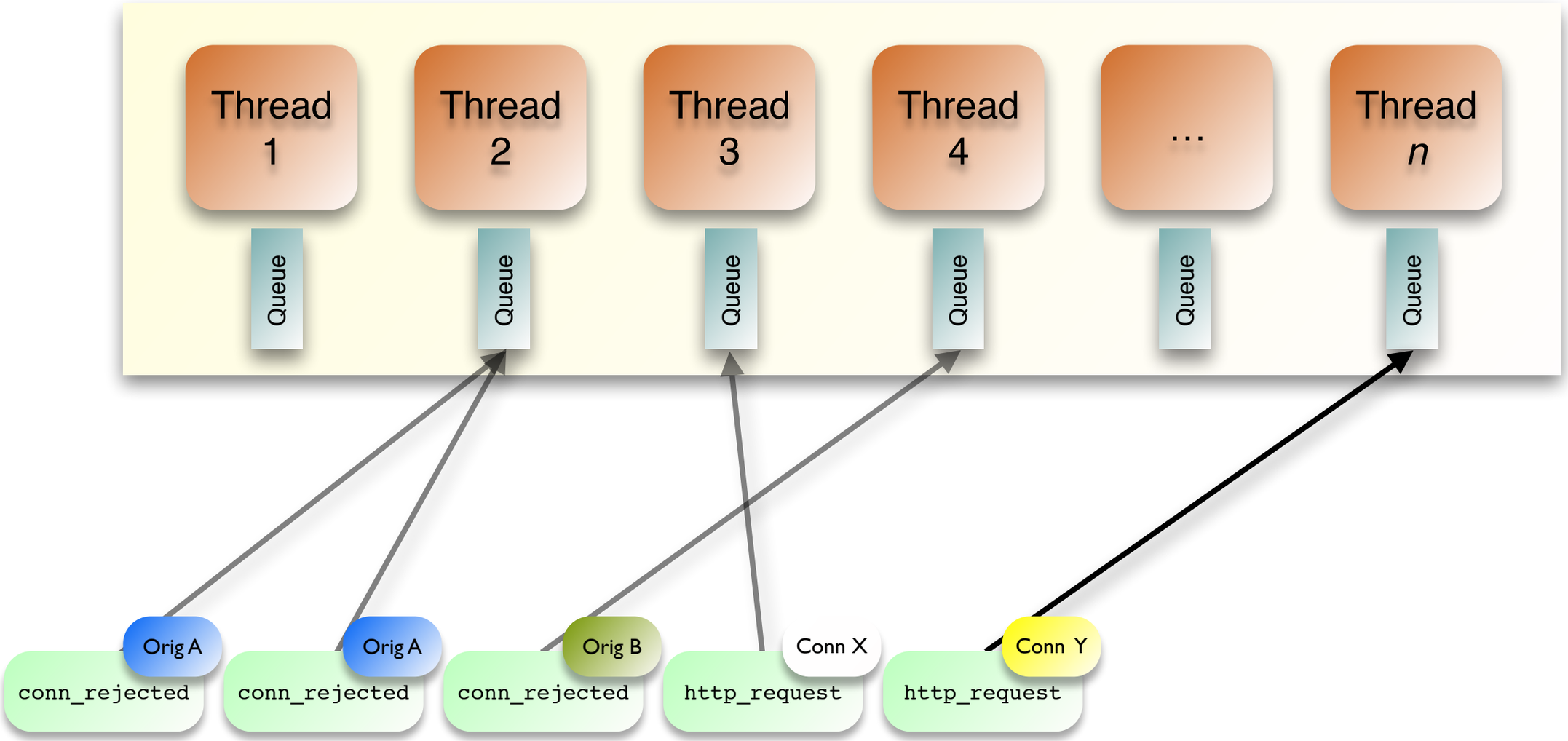
Parallel Event Scheduling



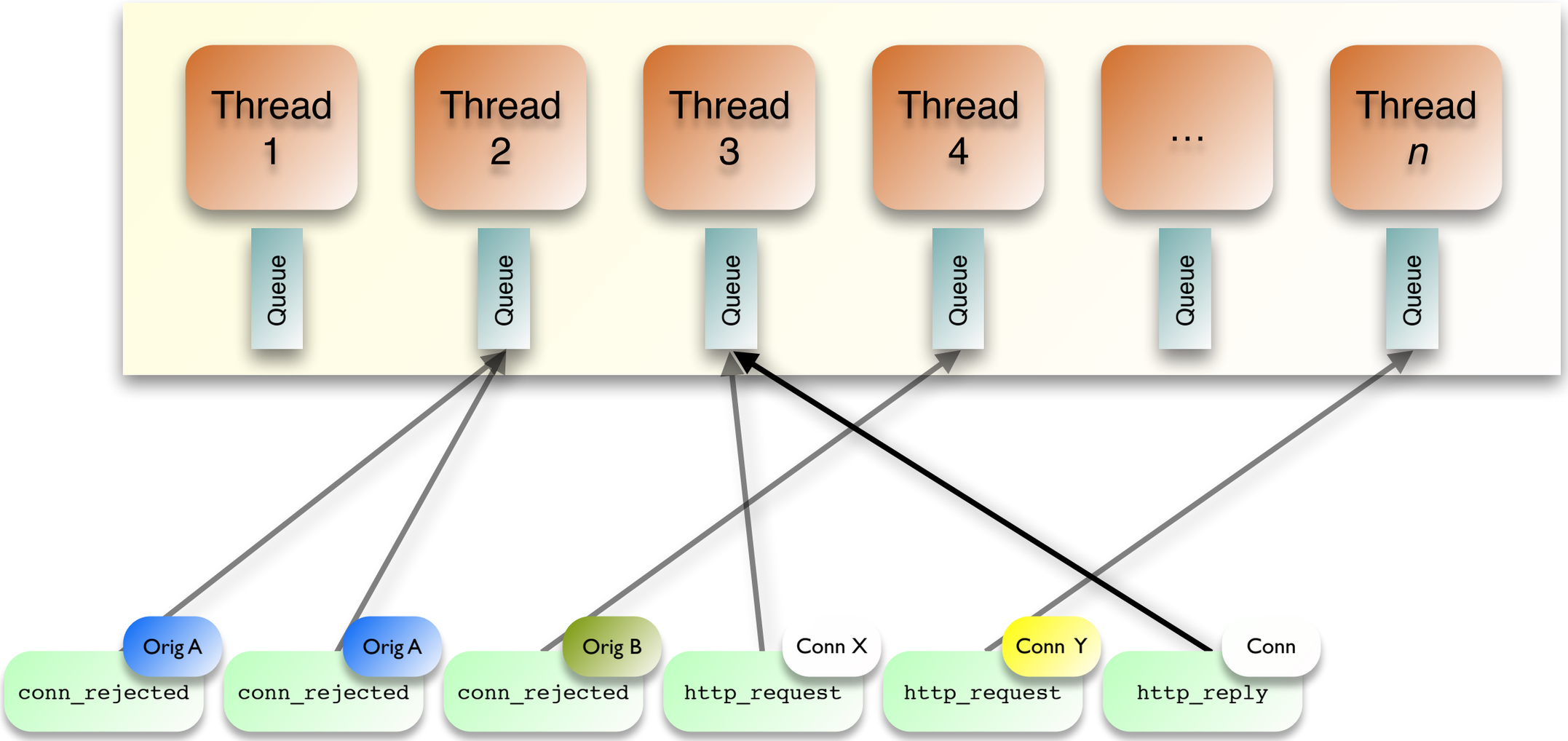
Parallel Event Scheduling



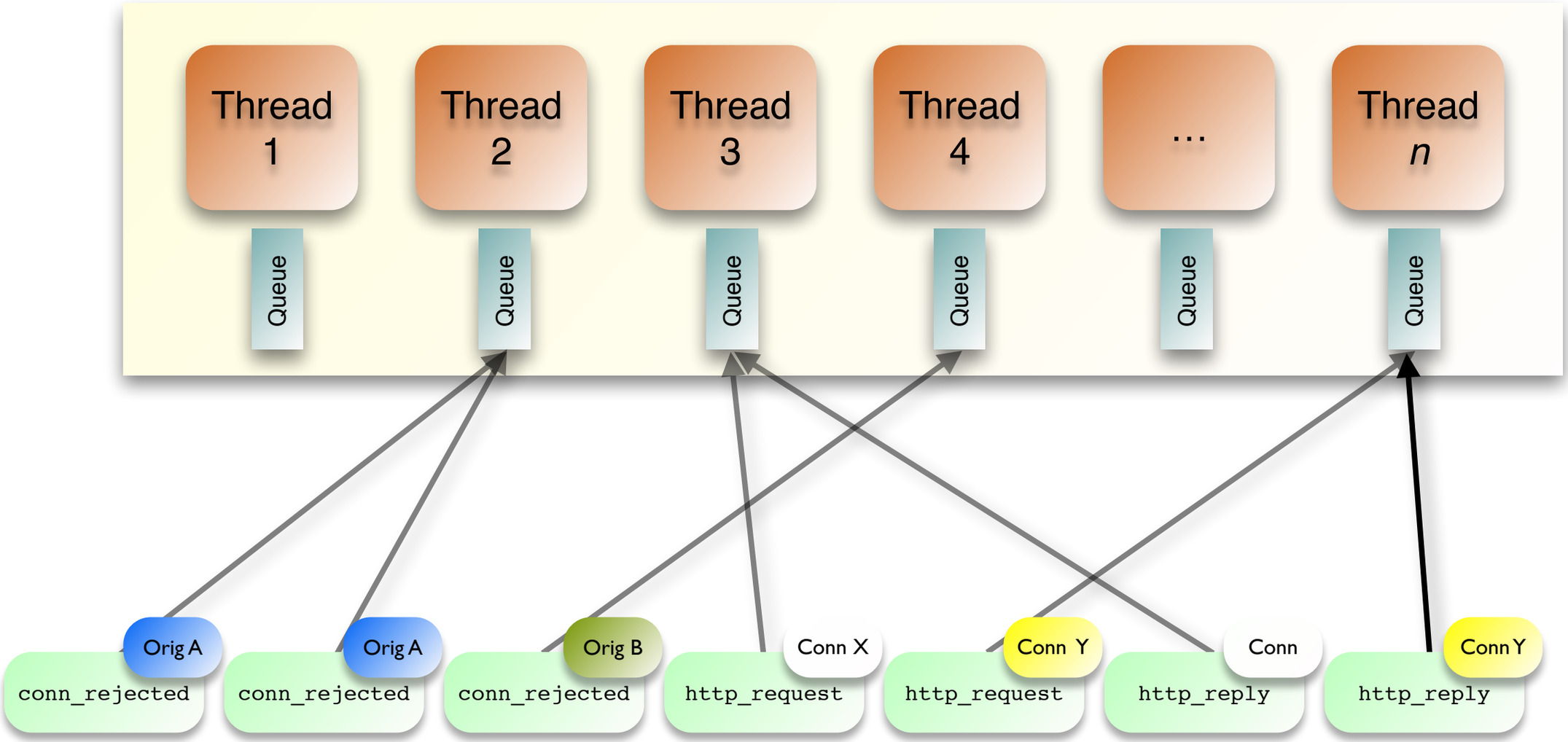
Parallel Event Scheduling



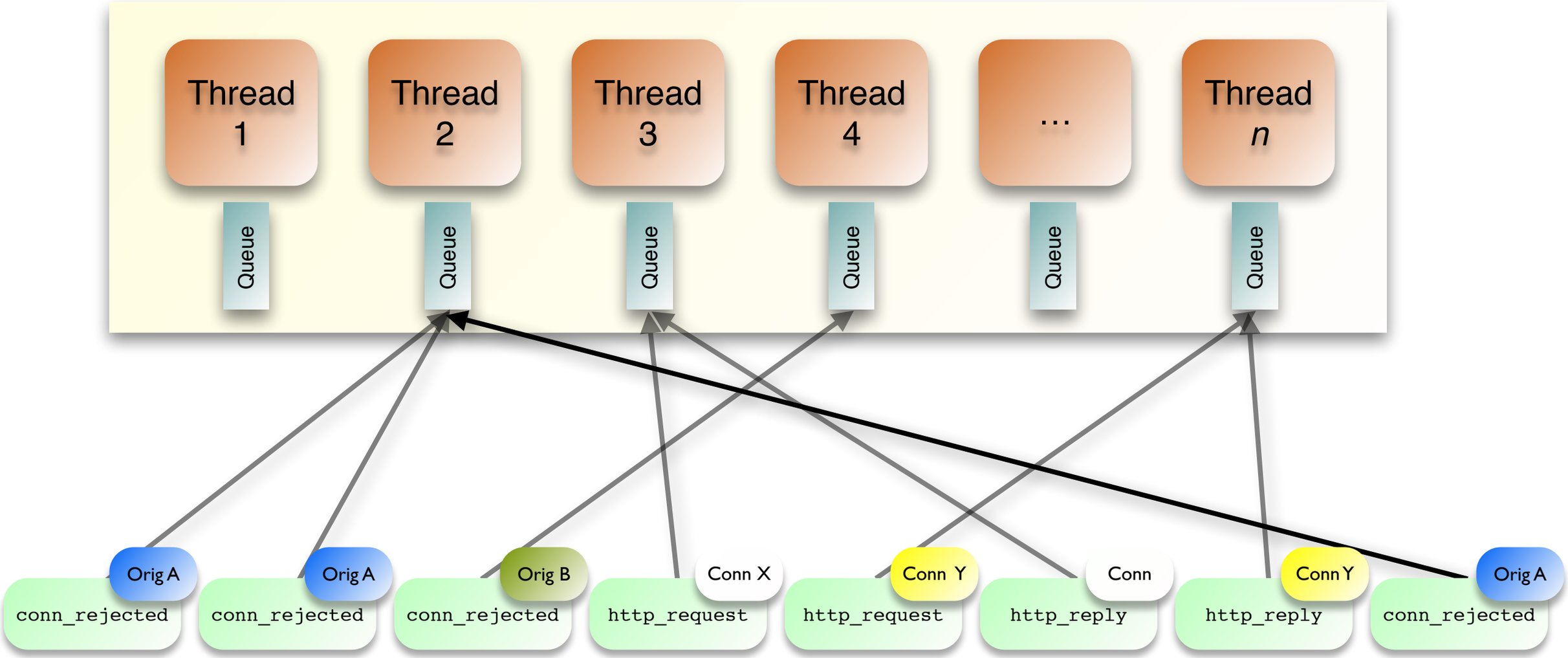
Parallel Event Scheduling



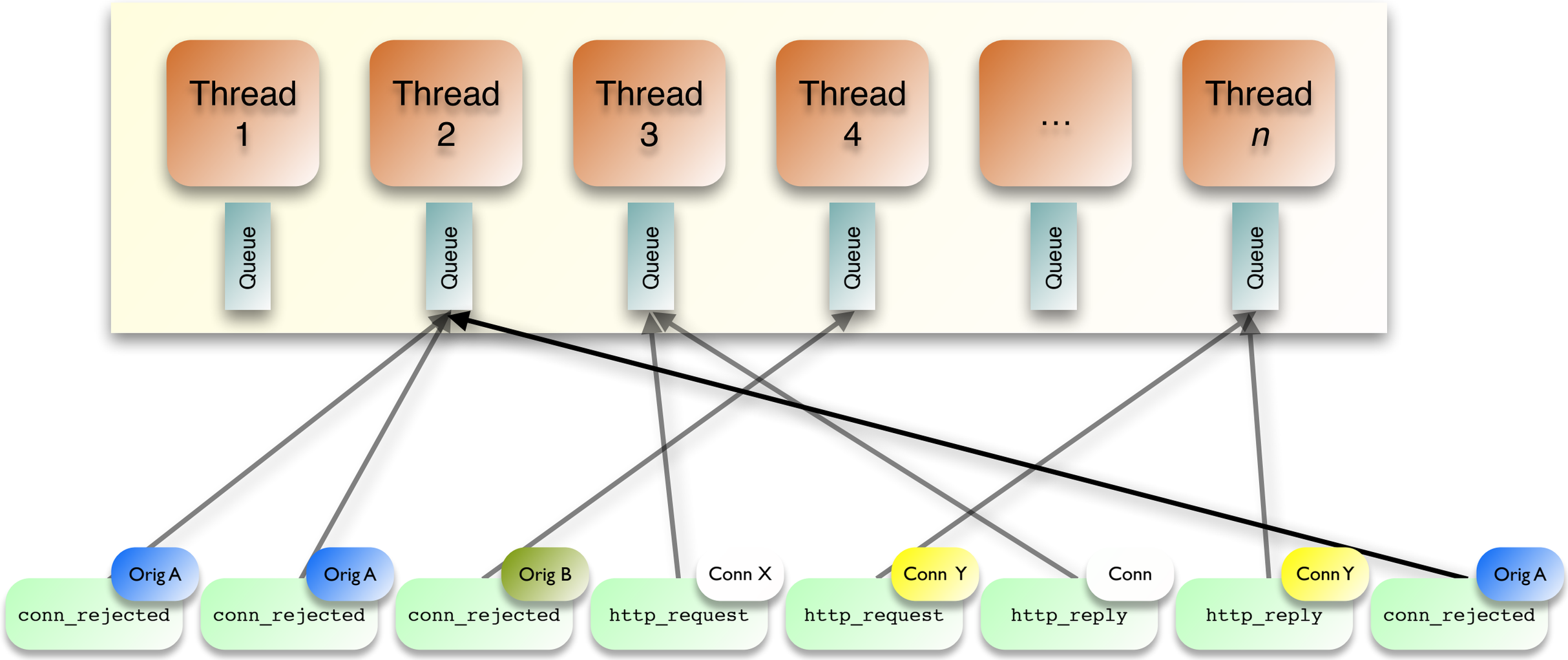
Parallel Event Scheduling



Parallel Event Scheduling



Parallel Event Scheduling



Challenge: Implementing this …

New Platform: Abstract Machine

A **H**igh-Level **I**ntermediary **L**anguage for **T**raffic **I**nspection

New Platform: Abstract Machine

A **H**igh-Level **I**ntermediary **L**anguage for **T**raffic **I**nspection

Domain-specific Data Types	State Management	Concurrent Analysis	Real-time Performance	Robust/Secure Execution	High-level Standard Components
First-class networking types built-in	Containers with state management support	Domain-specific concurrency model	Scalability through parallelization	Well-defined, contained execution environment	Platform for building high-level, reusable functionality on
	Timers can drive execution	Support for incremental processing	Compilation to native code	Static type-system, and robust error handling	
			Extensive optimization potential		

New Platform: Abstract Machine

A **H**igh-Level **I**ntermediary **L**anguage for **T**raffic **I**nspection

Domain-specific Data Types	State Management	Concurrent Analysis	Real-time Performance	Robust/Secure Execution	High-level Standard Components
First-class networking types built-in	Containers with state management support	Domain-specific concurrency model	Scalability through parallelization	Well-defined, contained execution environment	Platform for building high-level, reusable functionality on
	Timers can drive execution	Support for incremental processing	Compilation to native code	Static type-system, and robust error handling	
			Extensive optimization potential		

Summary

Conclusions

Threats have changed.

- Detection requires deep, flexible, semantic analysis.

Working to push the limits.

- Leverage capabilities of modern network hardware.
- Exploit parallelism inherent in network traffic analysis.

Bro is an ideal platform for such work.

- Operationally deployed across the country.
- Bridges traditional gap between academia and operations.

Thanks for you attention!

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

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