Security in Active Networks: Problems, Approaches, Results and Challenges

IWAN '99 Tutorial

June 30th, 1999

Jonathan M. Smith

University of Pennsylvania

http://www.cis.upenn.edu/~jms

Tutorial Outline:

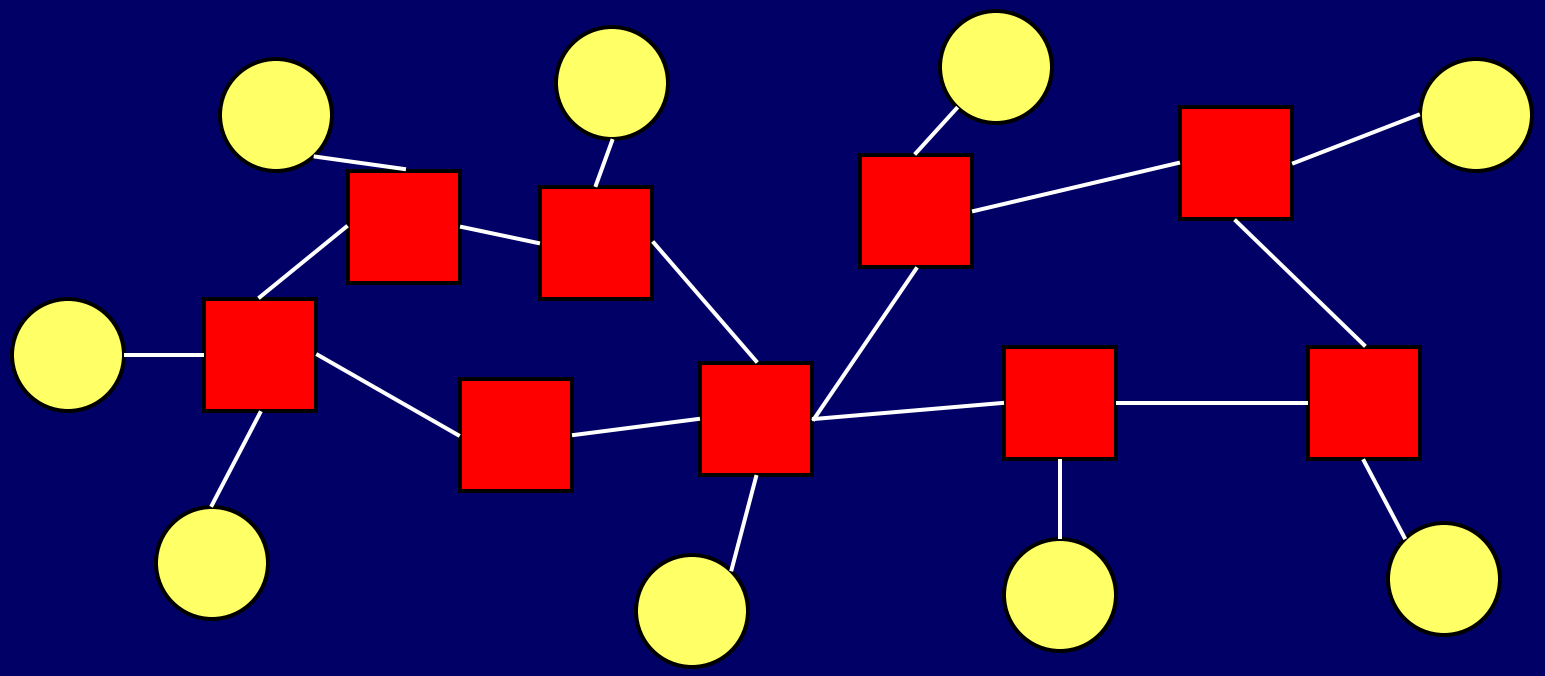
- Introduction to Active Networks
- Security Challenges
- Secure Active Network Environment
- Active Network Encapsulation Protocol
- Case Study: ALIEN Active Loader
- New Approaches
- Summary

1. Introduction

- Store & Forward versus Store, *Compute* & Forward
- Passive versus *Active* Networking
- An Example Application - Active Reliable Multicast (ARM)
- A developing Standard for an Active Network Architecture

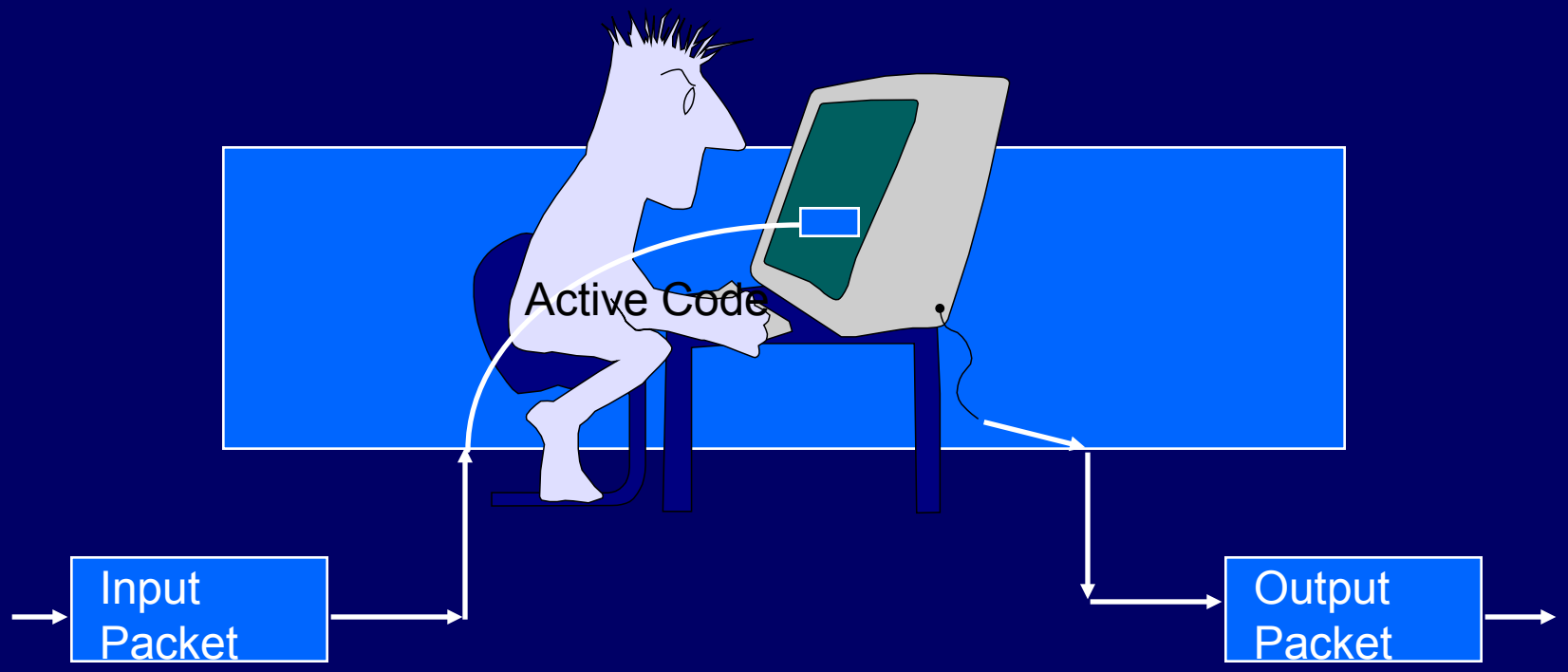
“Passive” Networking

- Smart hosts on the edges
- Passive switches in the center



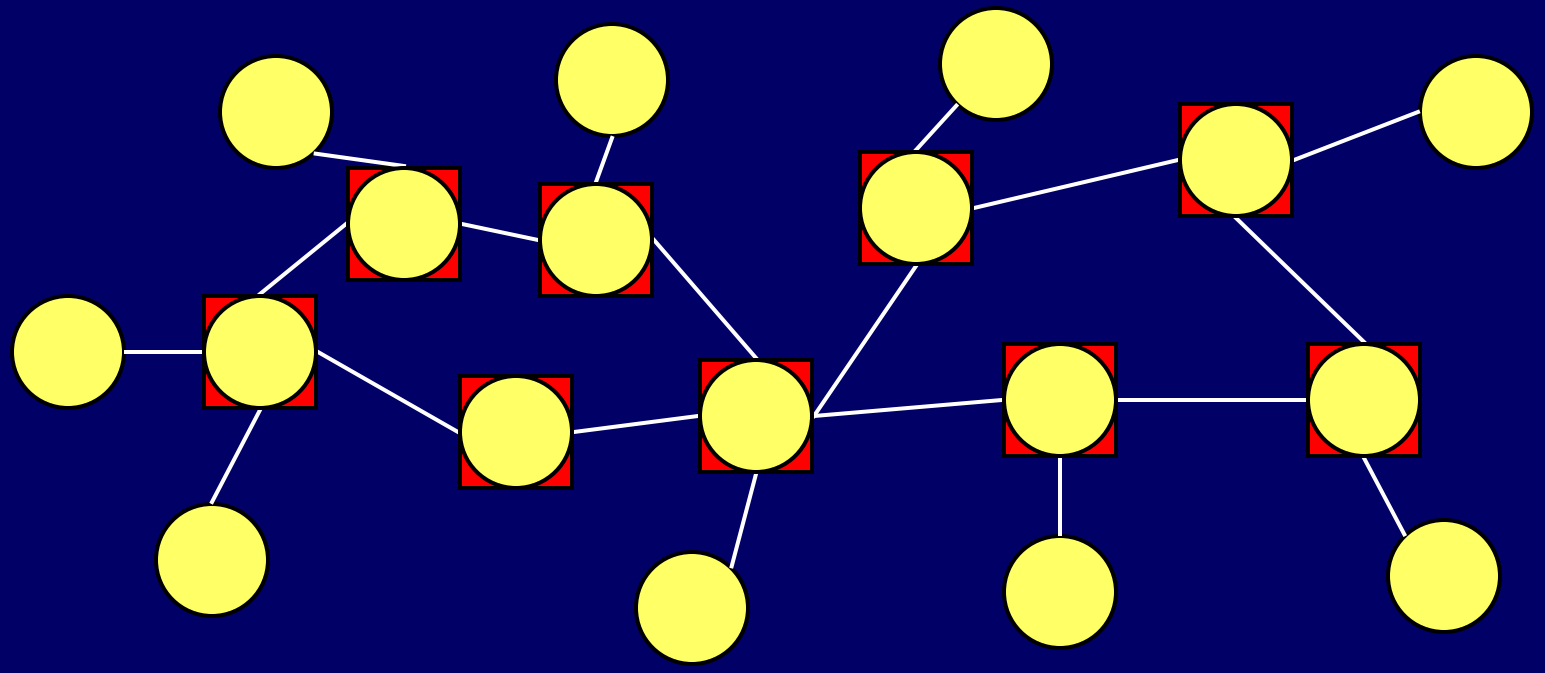
Active Networking Nodes

- Store, COMPUTE and Forward!

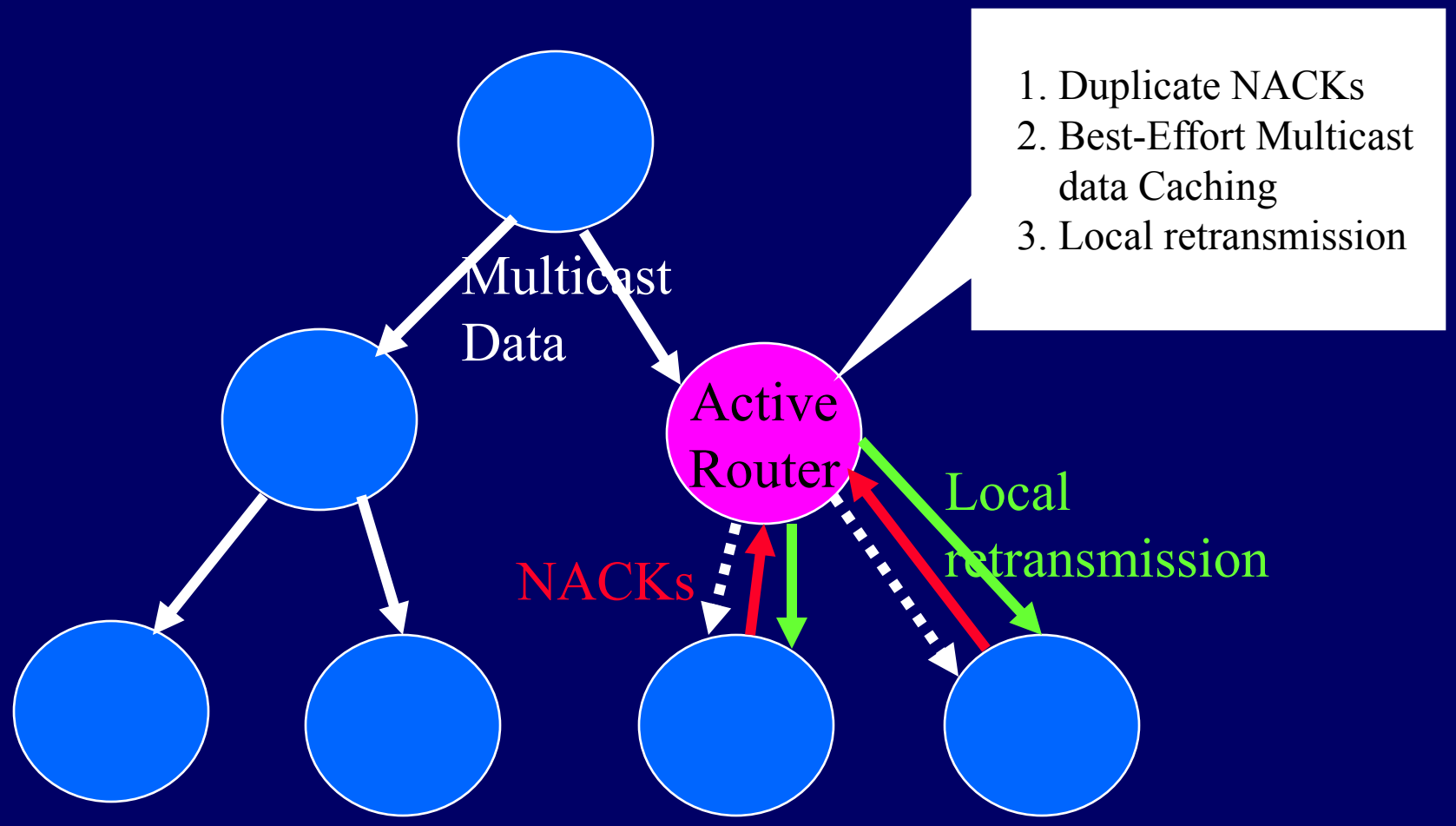


Active Network Model

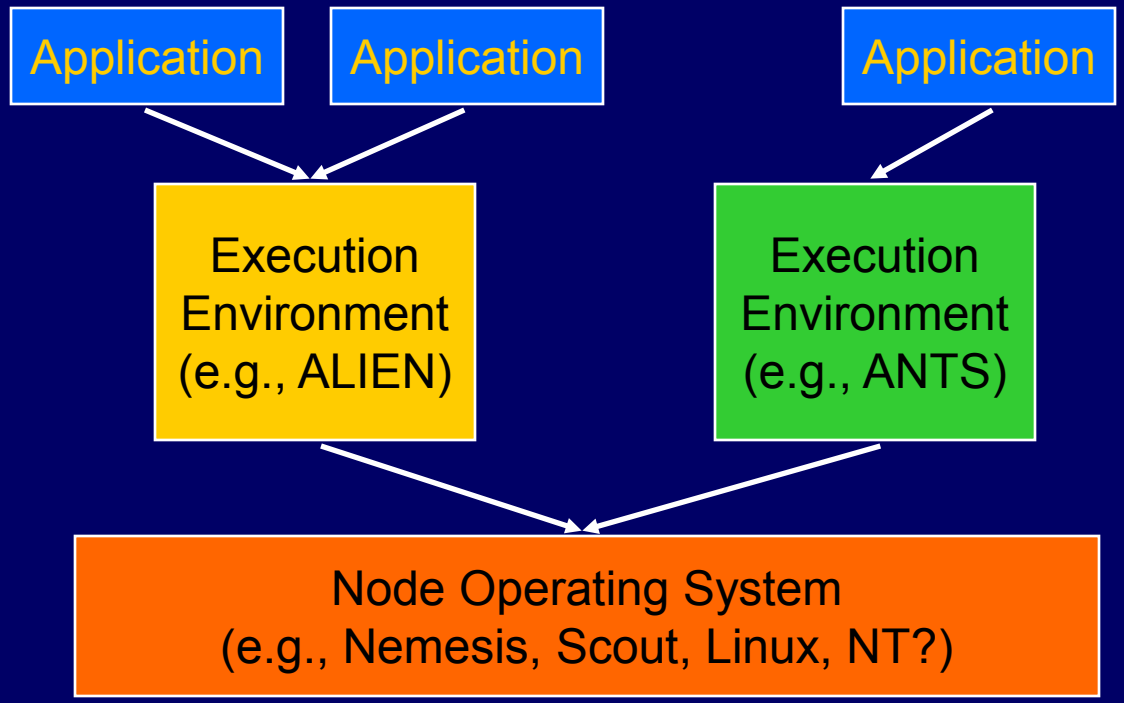
- Packets can change the behavior of the switches "on-the-fly"
  - In-band active packets
  - Out-of-band active extensions



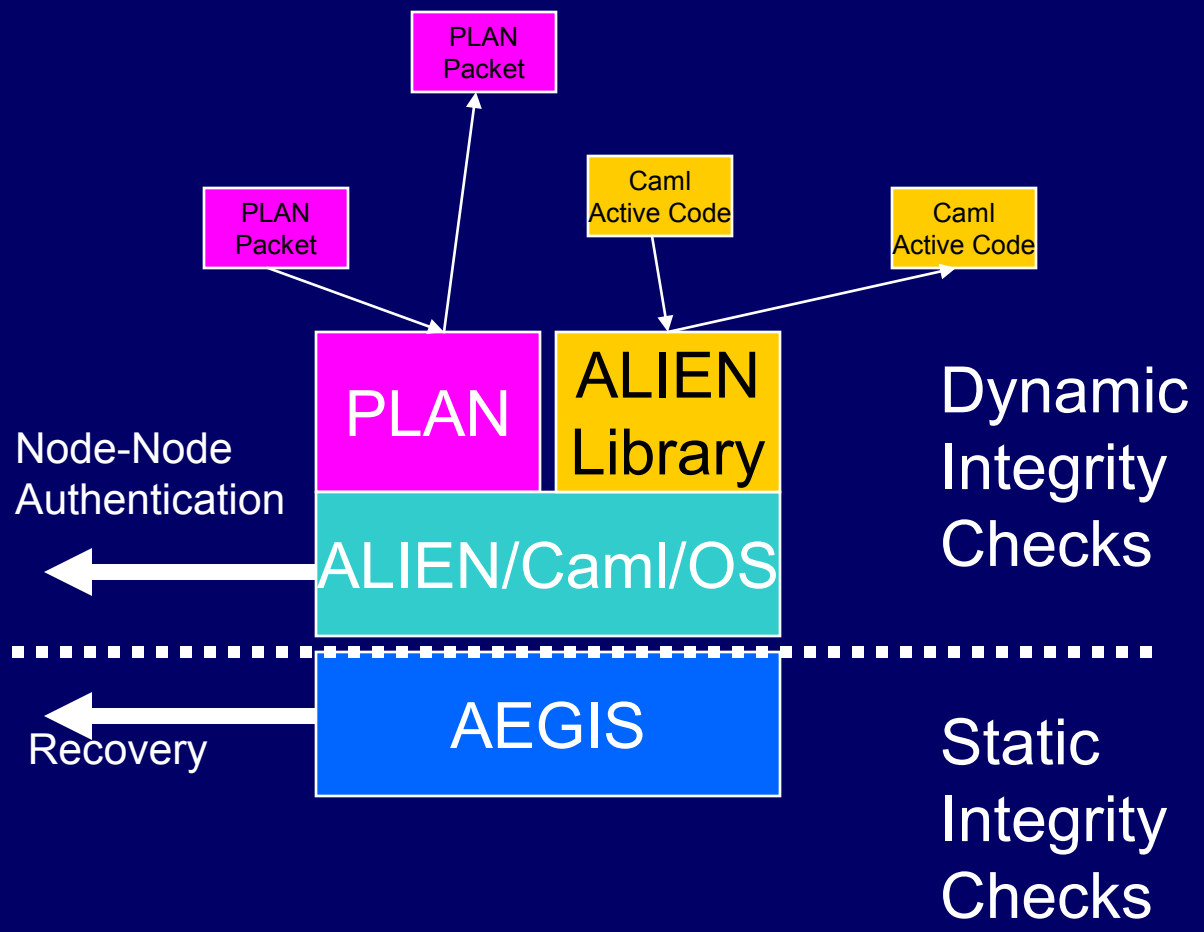
Example Application: ARM



"Active Network Architecture"

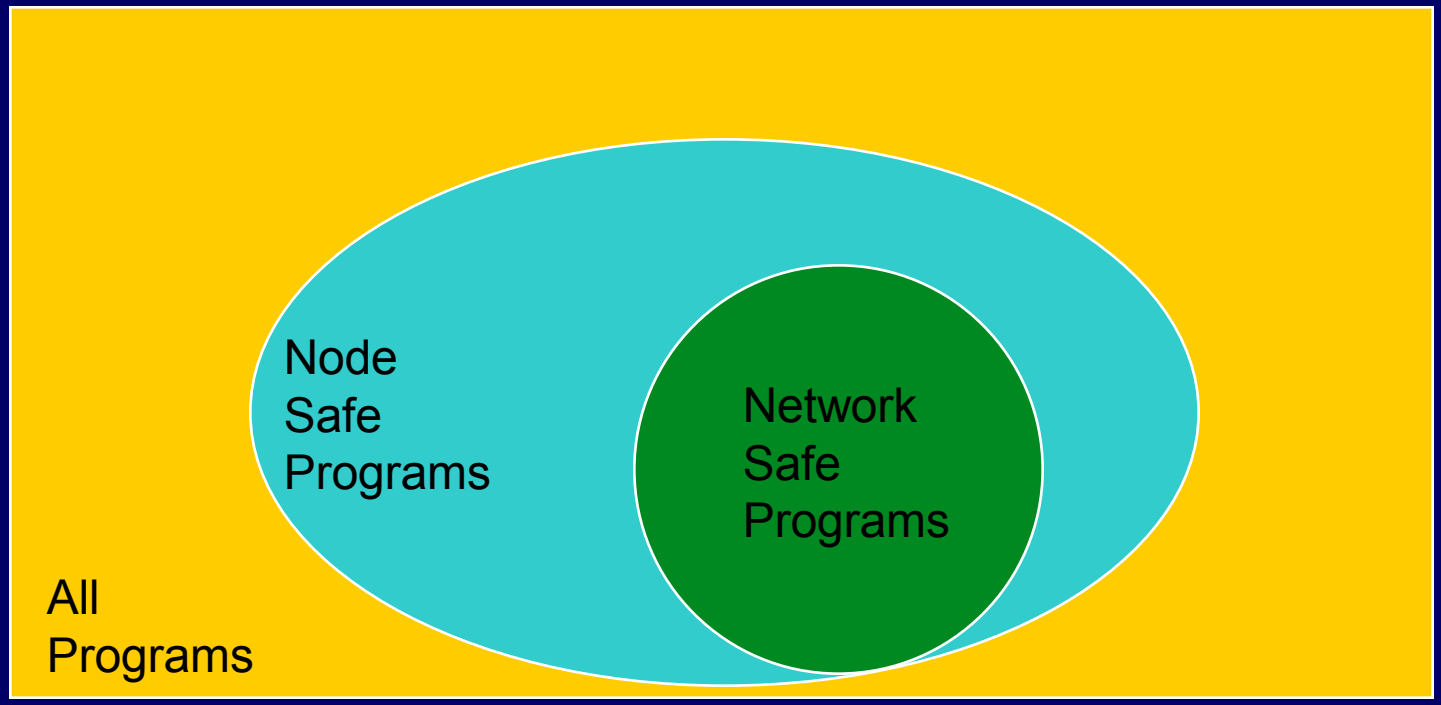


Example: SwitchWare Architecture



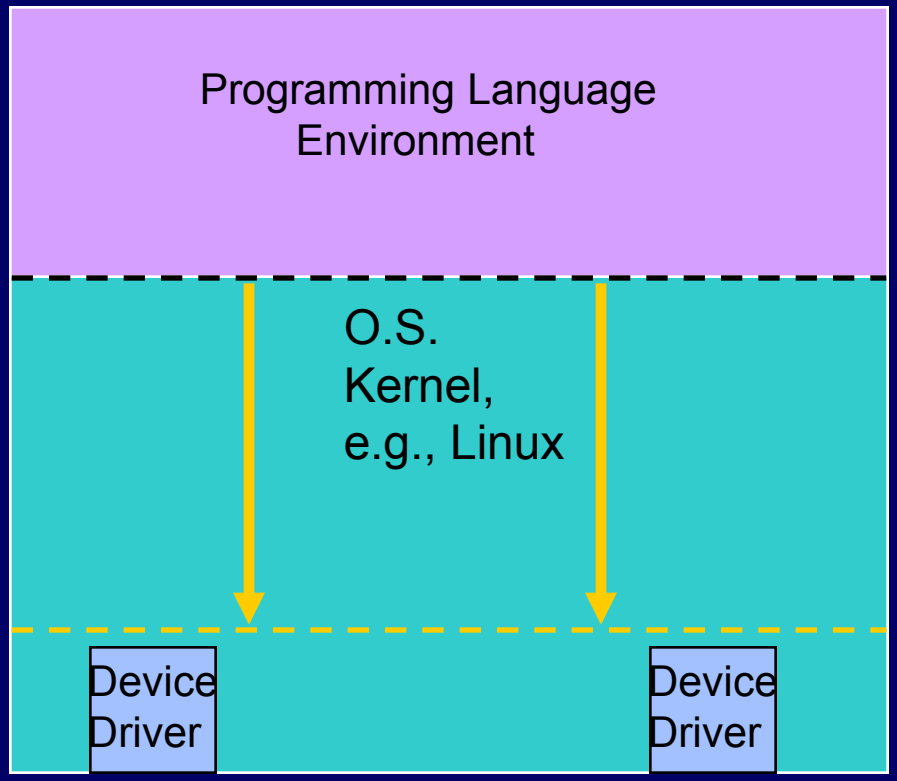
Restricting Programs

- Node safe versus network safe



How do we control programs?

- Safety & Security: P.L., O.S. or hybrid?



Challenges: Safety & Security

- Safety: Accidents; Security: Malice
- Specification of goal (@30,000 feet!):
  - *Right* Information to
  - *Right* Place at
  - *Right* Time
- Insecurity: Deviation from goal
  - e.g., information to *wrong* place

Right information/Right place

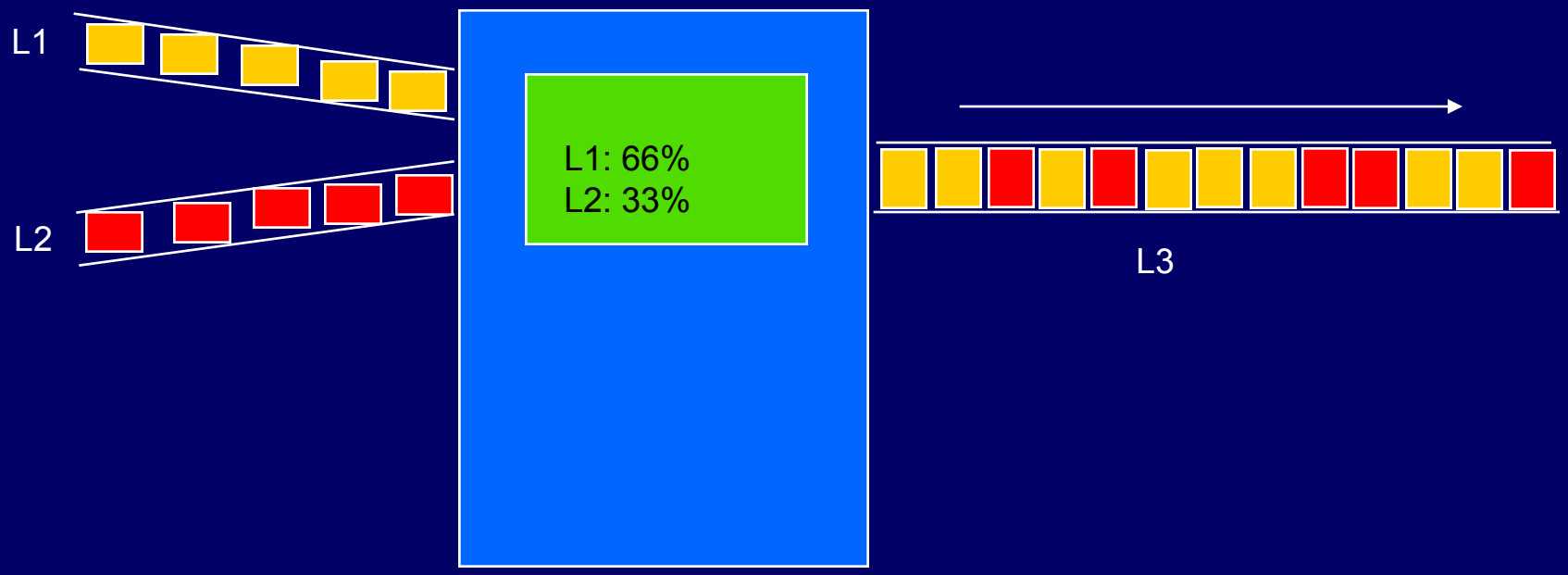
- Requires identifying information units
- Requires identifying places
  - e.g., locations, personnel, etc.
- Requires *security association*
  - e.g., per-place *password* encrypts info.
  - deny information to other places
  - cryptographic protocols: good progress

Right Time (the tricky one)

- Late information may be useless
- Basis of *denial of service* attacks
- Requires identifying *real* times
- Languages have no time semantics
  - gettimeofday() in C/Unix world
  - Is ML better? (Dannenberg's *Arctic?)*

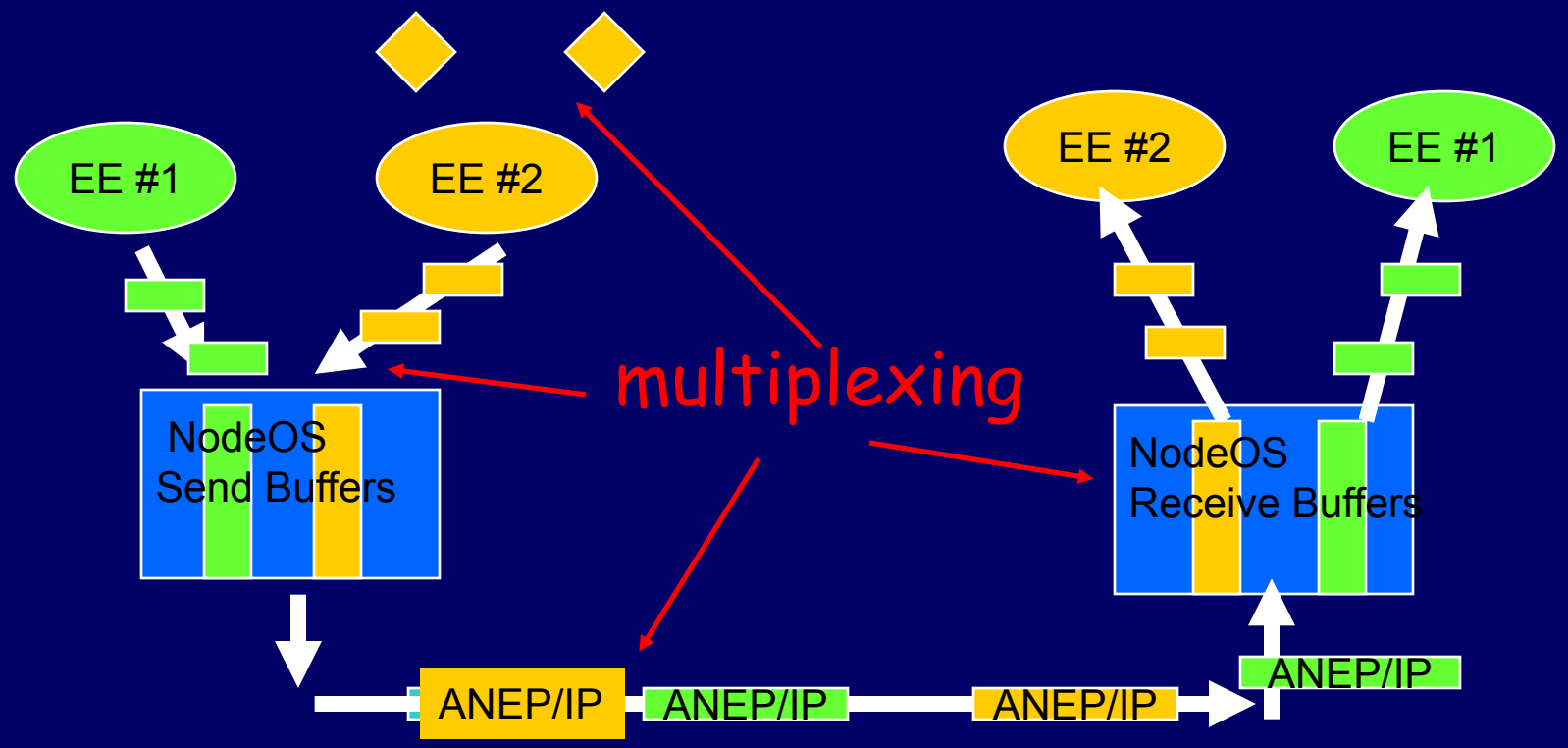
Need to control multiplexing

- E.g., assign L3 bandwidth 66%/33%



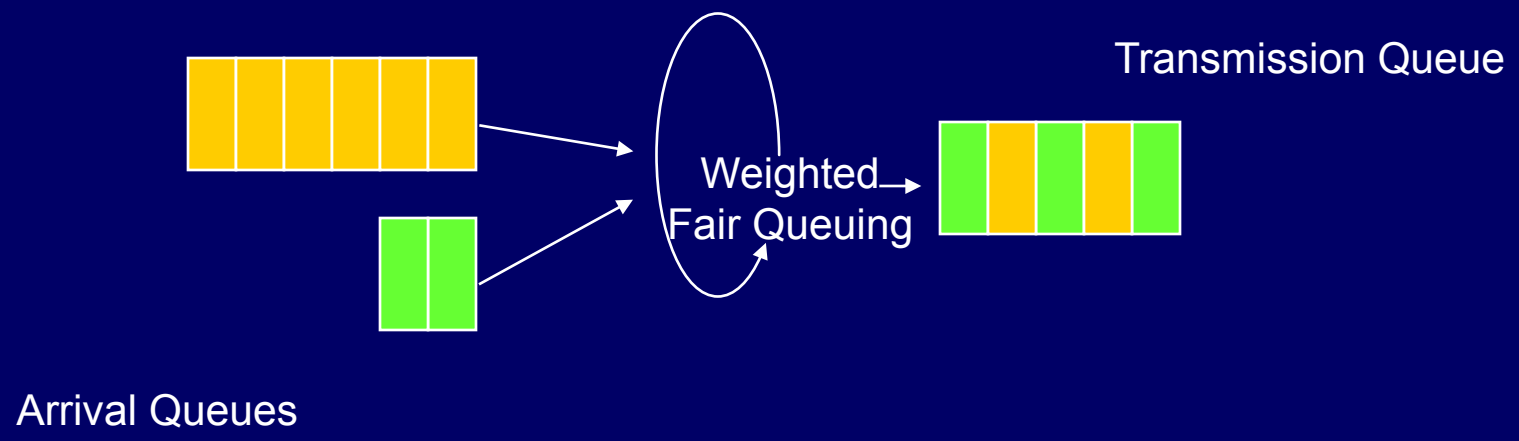
Resource Management, End-to-End

- Resource Management Challenges



Piecewise A.N. Node Solution: Loadable "Queue Management"

- Discriminates between "flows"
- Separate queue for each current flow
- Queues are serviced WFQ
- Control via RSVP, QoS Broker, etc.

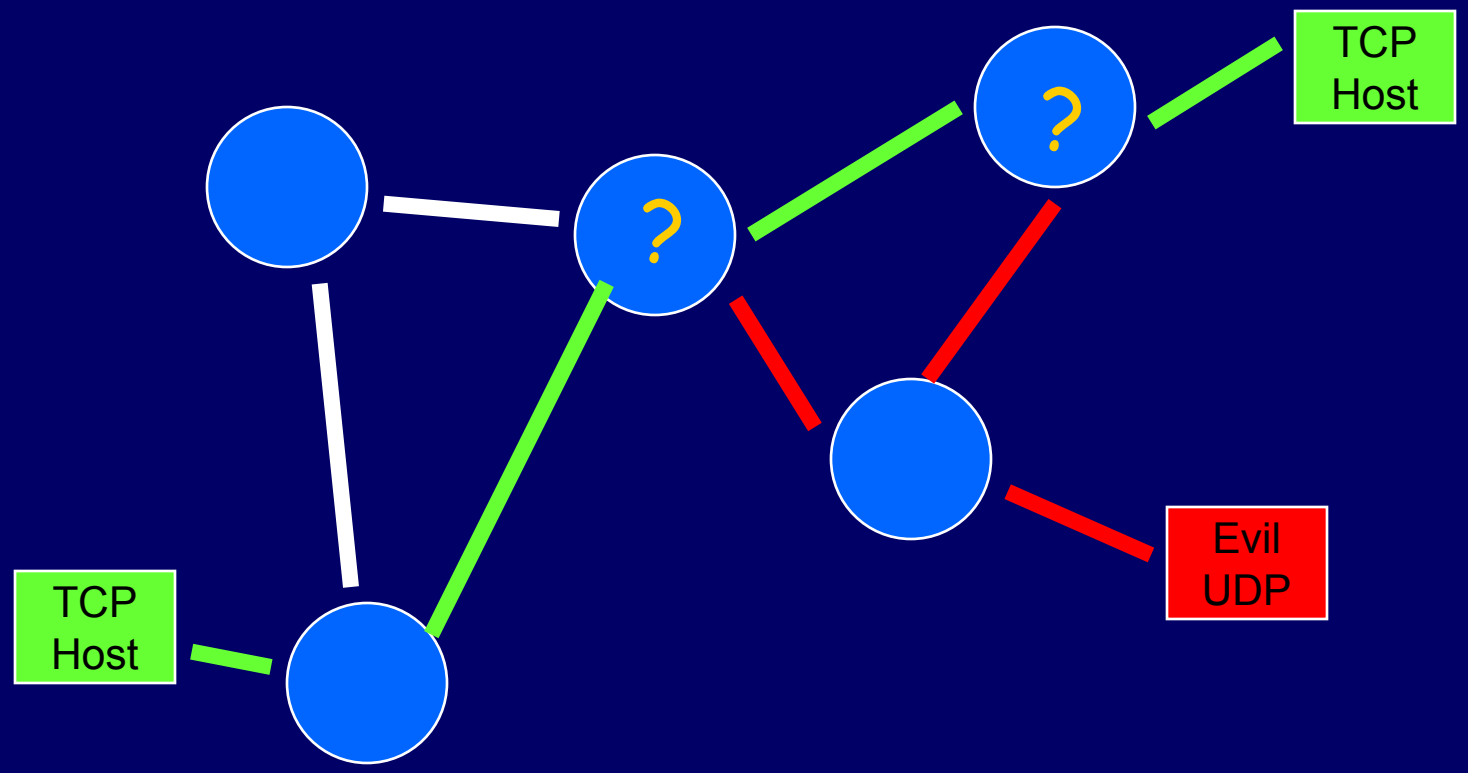


QoS & Security: Denial of Service

- Easy to protect server hosts
  - Resource domains, interrupt masking, firewall shielding on host *itself*
- But service is unprotected between client and server site
- This problem *must* be solved with network-embedded functionality

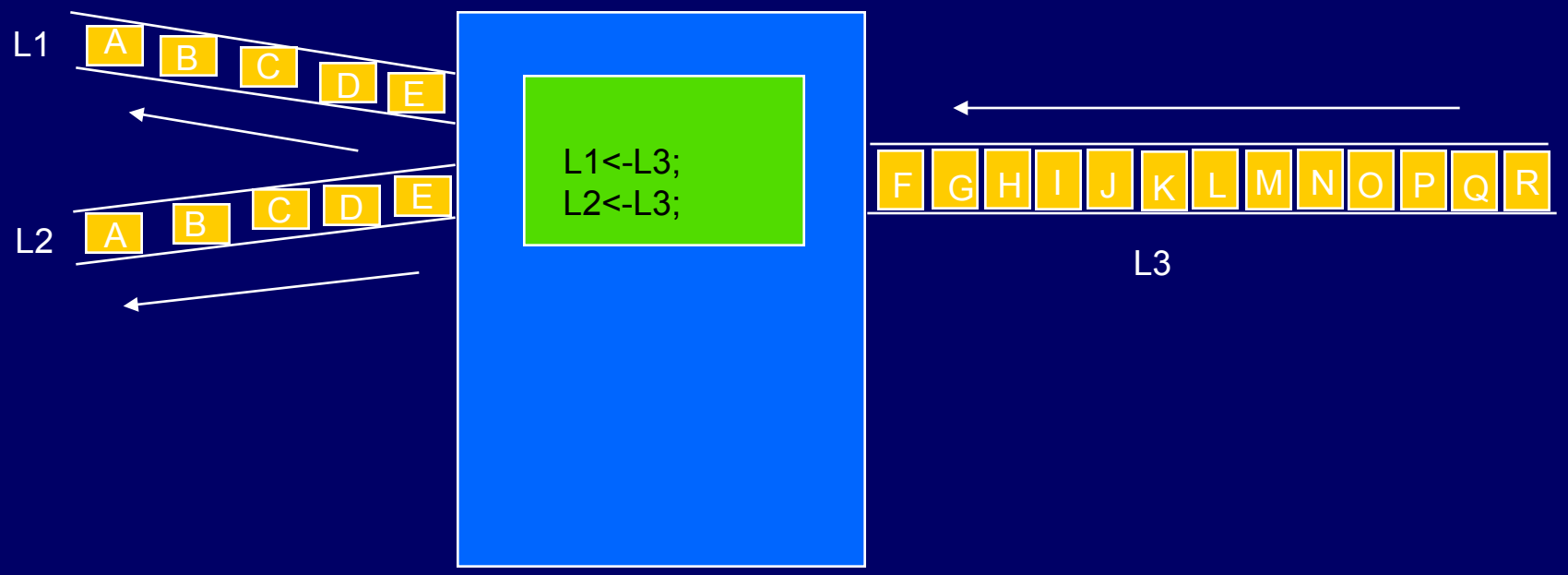
Denial of Service attack

- Cross traffic in an Internet



Unsolved "gotchas": Local versus Global control

- Program copies L3 (in) to L1, L2 (out)



- Is this "Multicast" Program "safe"?

Can Active Packets trust the EE?

- "Reflections on Trusting Trust"
  - Example of self-replicating compiler virus
  - Lesson: You are *trusting* infrastructure!
- A.N. concern so far: trust of code
  - Can the code trust the A.N.?
- Goal in an A.N.:
  - Either operate in untrusted environments
  - Or establish web of trust

Strategies for paranactive nets

- Carry *all* code with you in a capsule
  - how do you load your code?
- Telescope out trust relationships with cryptography and identities
  - need to think about *ad-hoc* relations
- Pre-establish trust relationships and verify at node

Result: E.E. in known state, *but…*

- Still trust some hardware
- Also trust repository for recovery
- Need *basis*, like diplomatic pouch containing a one-time pad
- *Applications* aware AEGIS executed?
- Can *applications* know that system integrity has been *preserved?*

Some (maybe crazy) ideas:

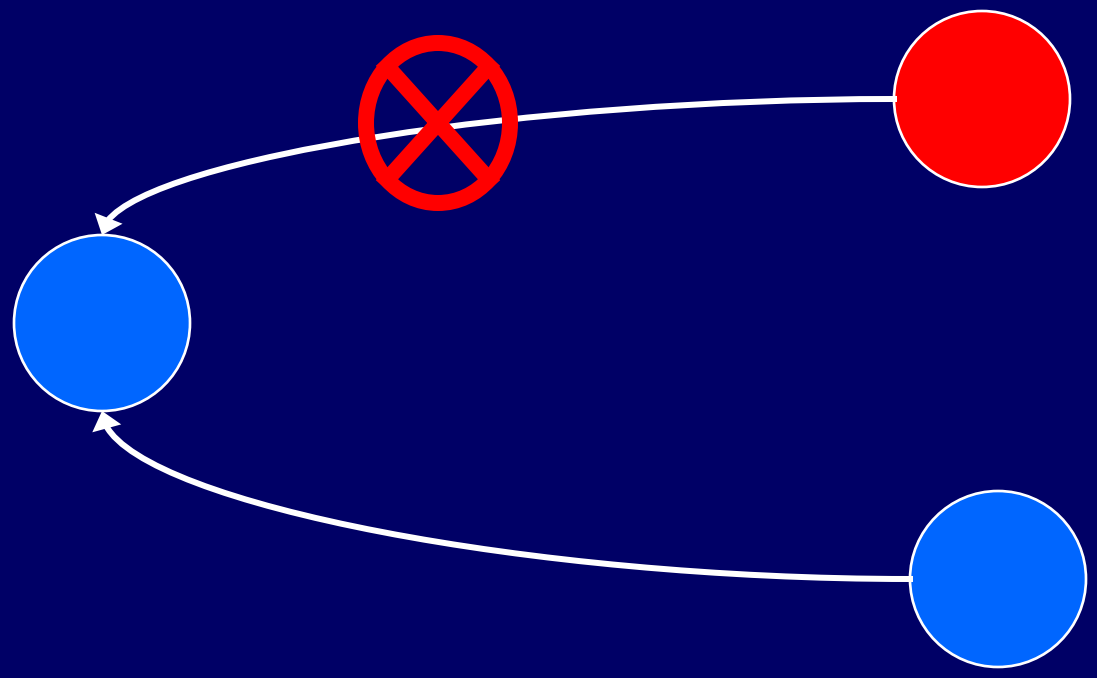
- Allow paranactive applications to invoke AEGIS with themselves as target…
  - Awful performance, poor multiplexing :-)
- Paranactive applications "disarm" *gradually* (gradually expose more code and credentials as environment is checked)
- Automated Trust Management (need new acronym - "third rail" of nets!)

3. The Secure Active Network Environment (SANE)

- Demonstrates active packet programming
- Mobile code authentication with cryptography
- Guarantees no corrupted component
- Allows recovery of failed components
- Enables trust relationships between nodes
- http://www.cis.upenn.edu/~waa
- http://www.cis.upenn.edu/~angelos

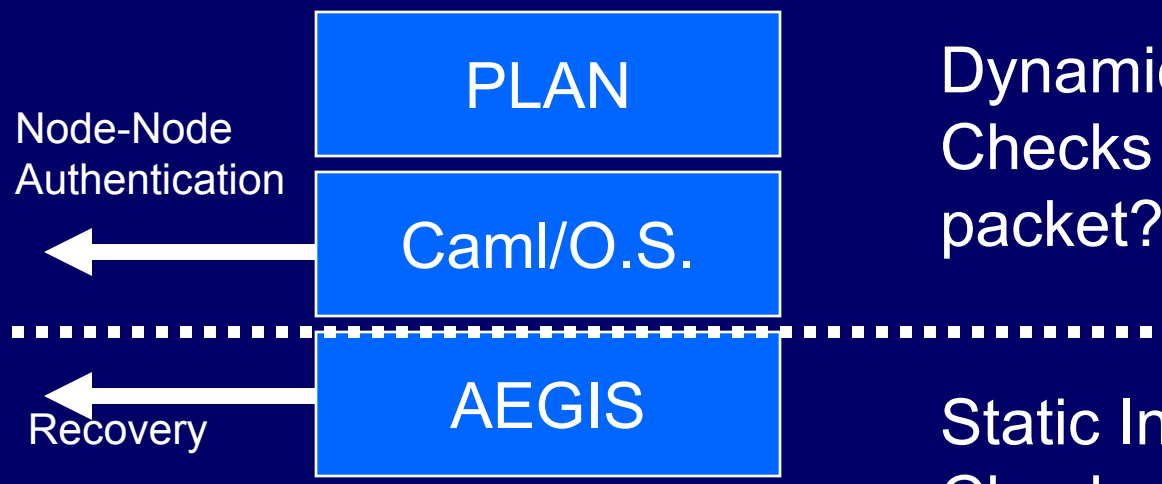
SANE Security Model

- Only process packets from trusted hosts



SANE Architecture

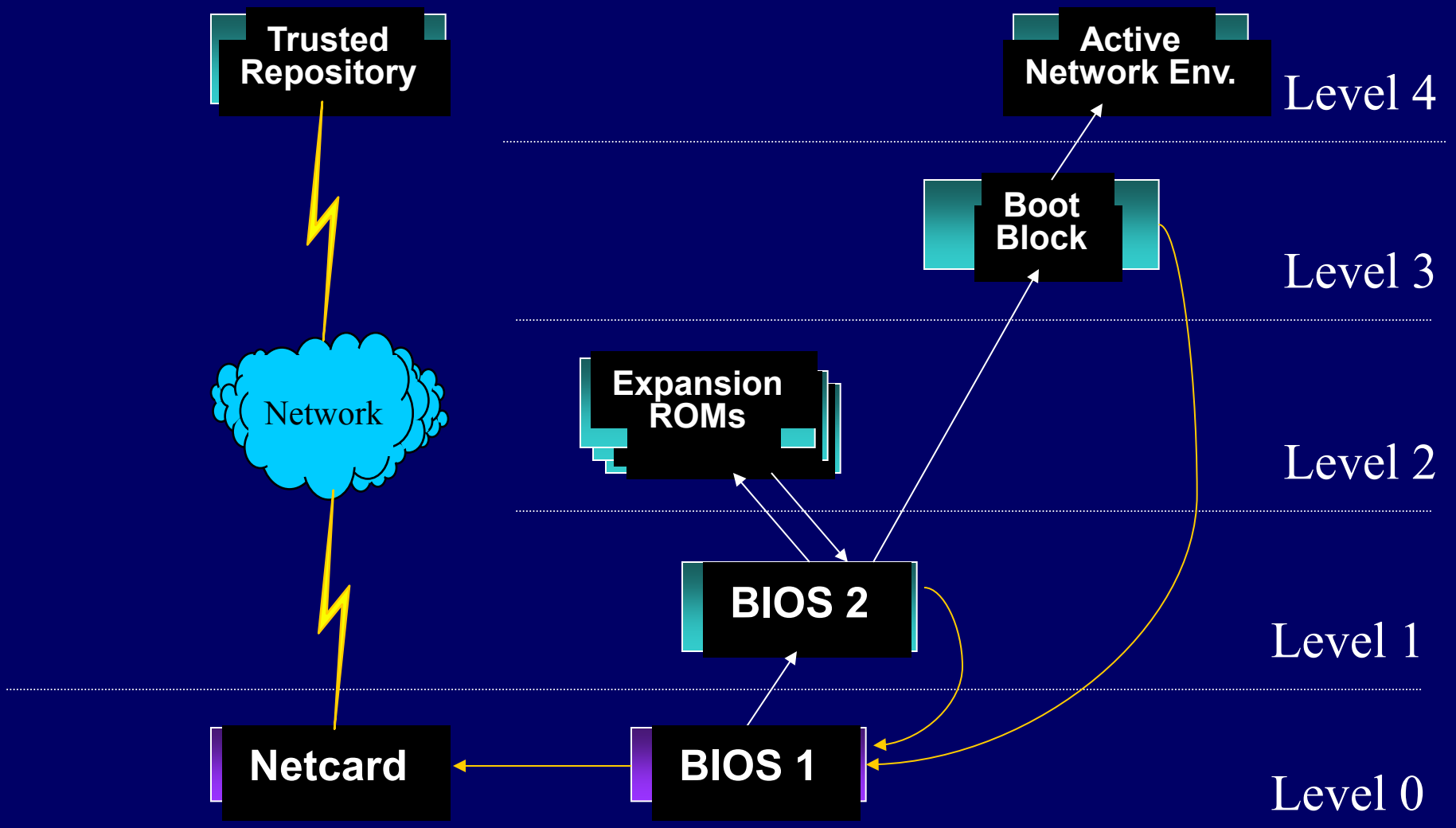
- "Trust, but Verify"



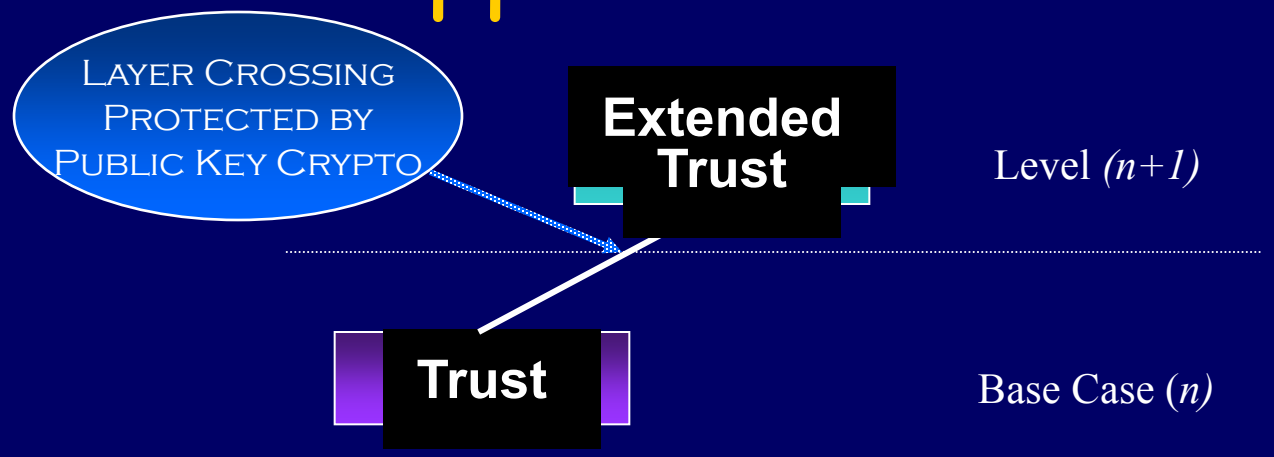
Dynamic Integrity Checks (Maybe per-packet?)

Static Integrity Checks (Done Once)

AEGIS Architecture

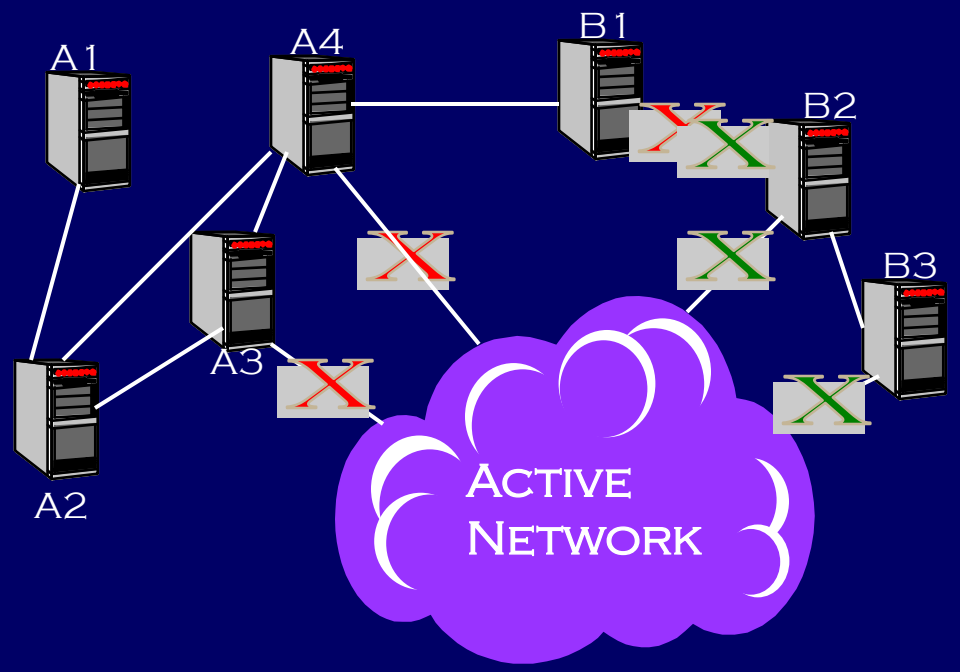


Approach



- Integrity and Trust Must be "*Grounded*" at the Lowest Possible Point.
- Chaining Layered Integrity Checks (CLIC) Extends Trust Beyond the Base Case.

Mutually Suspicious Nodes



- Nodes Authenticate their Neighbors
- Establish Trust Relations with Peers (PolicyMaker?)
- Use Trust Relations to Solve Existing Problems (eg. Routing)
- Optimize Authentication

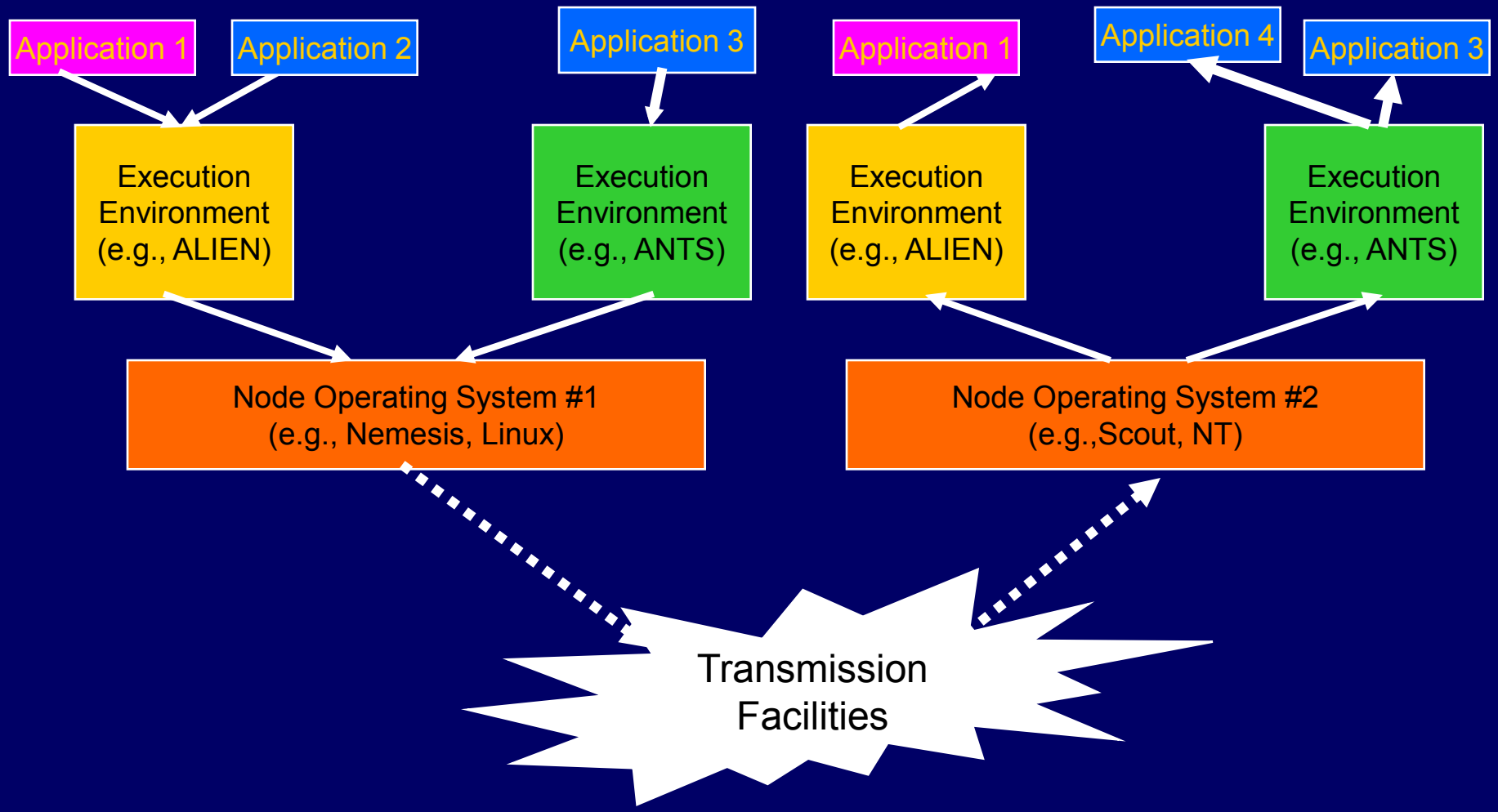
Node to Node Authentication

- Once at Boot Time, Periodically Thereafter (Crypto "heartbeat")
- Modified Station-to-Station Protocol (Well Known and Understood)
- Key Can be Used to Authenticate on a Hop-by-Hop Basis, Encrypt Sensitive Information
- Make Traffic Analysis <u>Hard</u>

4. Active Network Encapsulation Protocol (ANEP)

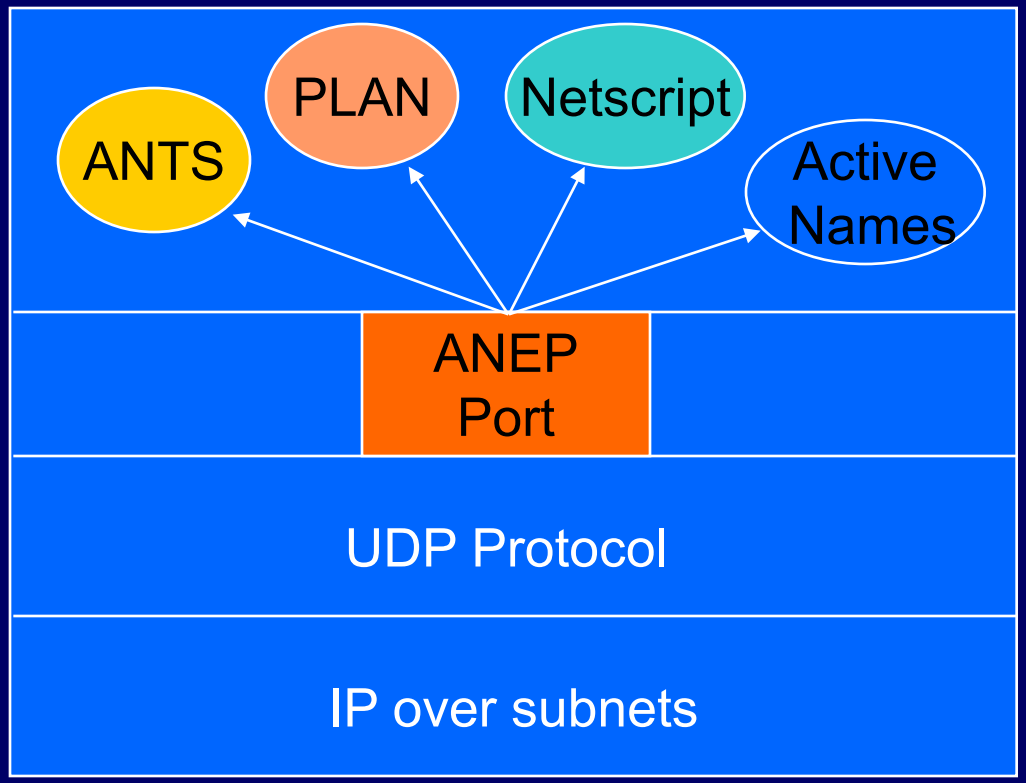
- Why ANEP?
- ANEP details
- Security features of ANEP

Internode Interoperation



ANEP demultiplexes to EEs

- Well-known UDP/IP Port for ANEP



ANEP Header Formaat

- Format of ANEP Header:

Terminology, FYI:

- *Packet*: ANEP Header + Payload
- *Active Node*: Network Element that can evaluate active packets
- *TLV*: Type/Length/Value triple
- *Basic Header*: First two words (8 octets) of the ANEP Header

ANEP Details: Fields

- *Version*: now 1; change w/ANEP header; discard if unknown value
- *Flags*: for V1, only MSB used
  - MSB=0, try to forward w/default
  - MSB=1, discard if TypeID not recognized
- *ANEP Header Length*: in 32 bit words
  - includes options; 2 if no options

Details: More fields…

- *TypeID*: evaluation environment for message; 16 bits; values by ANANA
  - ANANA is currently Bob Braden
  - Unrecognized value? Check *Flags* MSB
- *ANEP Packet Length*: Length of entire packet in *octets* (including payloads)
- *Options* length (variable) computed from Packet and Header length difference

Options

- Zero or more Type/Length/Value (TLV) constructs
- Follow the basic header. Format:

Option Fields

- *Option Type*: 14 bits, used to interpret *Option Payload*.
- Values assigned by ANANA; private when MSB of *FLG* is set.
- Unrecognized value? LSB of *FLG* 0, continue; 1 discard packet. Should log.
- *Option Length*: 16 bits; TLV length in 32 bit words; >= 1.

Option Type Values

- Reserved:
  - 1 - Source ID
  - 2 - Destination ID
  - 3 - Integrity Checksum
  - 4 - Non-Negotiated Authentication
- Format for Source, Destination, N-N:

Scheme Identifier
...
Option Payload

Source Identifier

- Uniquely identifies sender
- *Scheme Identifier* is 32 bits; identifies addressing scheme to interpret the variable size *Option Payload*
- Reserved:
  - 1 - IPv4 Address (32 bits)
  - 2 - IPv6 Address (128 bits)
  - 3 - 802.3 Address (48 bits) (last two octets 0)

Destination Identifier

- Uniquely identifies destination in the active network
- Same payload option format as Source Identifier

Integrity Checksum

- Detect some packet integrity losses
- 16 bit 1's-complement of 1's-complement sum of the ANEP packet from the ANEP Version field
- Payload zero for computing checksum
- Option length field is 2.

Non-Negotiated Authentication

- Provides 1-way authentication
- No prior negotiation assumed
- Option payload: 32 bit authentication scheme, followed by scheme's data.
- Option length field >2.
- Reserved:
  - 1 SPKI self-signed certificate
  - 2 X.509 self-signed certificate

5. Case Study: ALIEN Active Loader

- Programming Language Approach
- Protection with “namespace sandbox”
- Extend to network with crypto
- Performance implications
- Not the whole story

The Design Space

- Usability *vs.* Flexibility *vs.* Security *vs.* Performance
- A General-Purpose Language gets the first two for free; other two are <u>hard</u>!
- Domain-specific Languages may achieve different tradeoffs

The ALIEN Approach

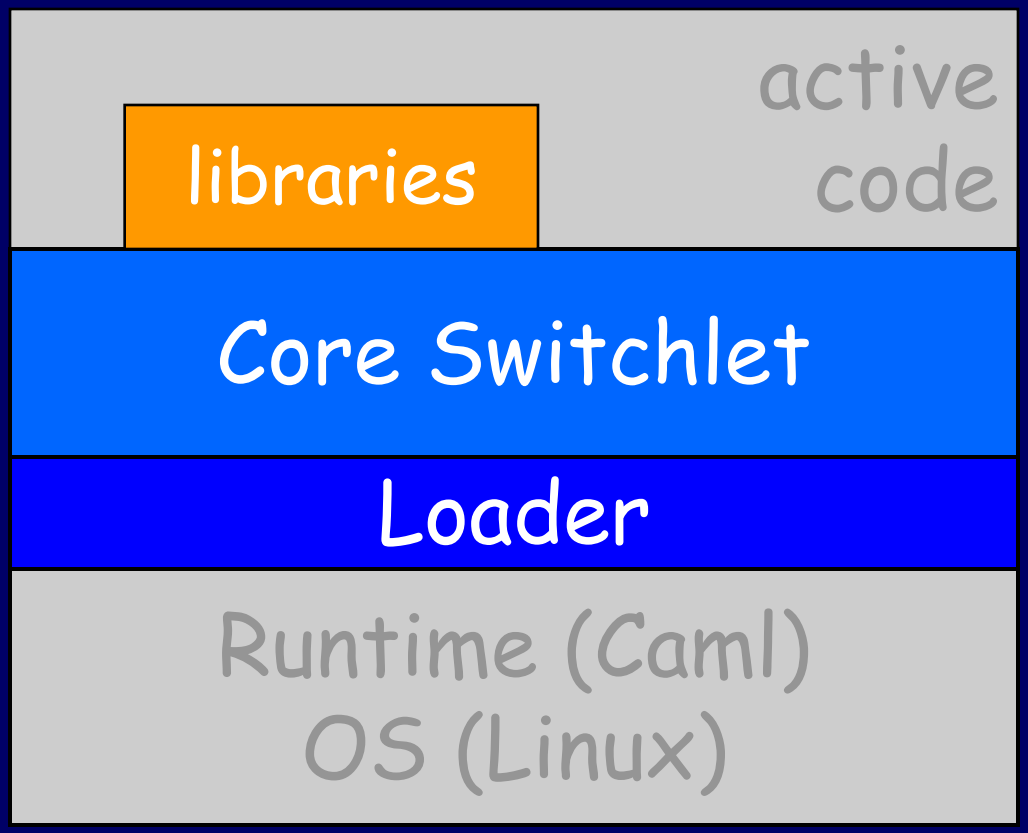
- Achieved by *restricting* a general computing model
- Realized in ALIEN, an active loader for Caml
  - General computing model
  - Interface to OS
  - Interface to active code
- Only privileged portions of the system can directly access shared resources

The ALIEN Active Loader

- D. Scott Alexander
- CAML runtime
- CAML capsules restricted via module thinning
- Digitally-signed certificates for remote accesses to resources
- Will use for detailed case study

ALIEN in an Active Element

- Three layer architecture



Implementation of Active Code

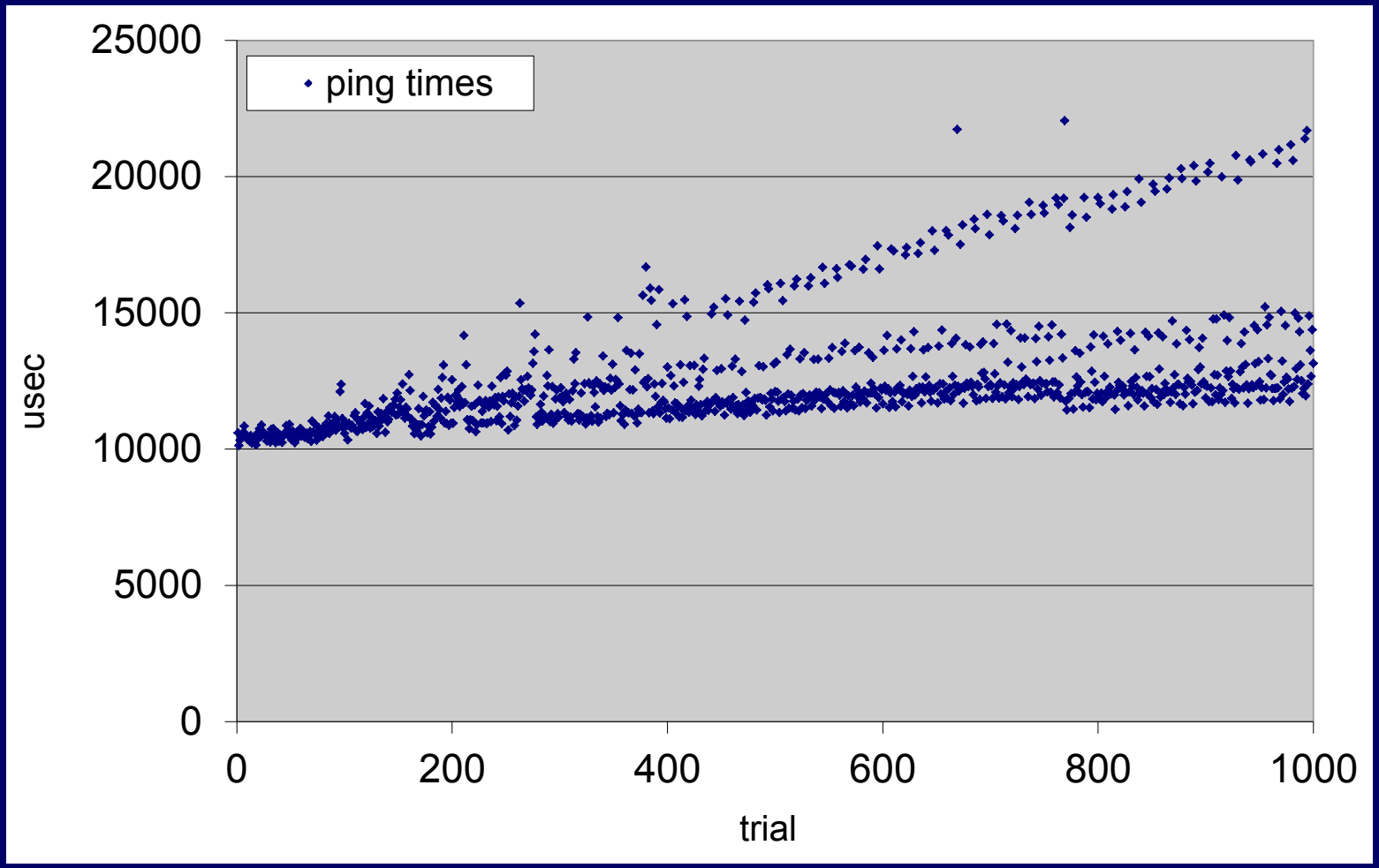
- Active Extensions
  - Loaded from disk or network (TFTP)
  - We use queues for communication
  - Could use upcalls...
    - Security?
  - ...or blocking downcalls
- Active Packets
  - ANEP encapsulated (over UDP or link layer)
  - Can use SANE for security
  - Linker/ procedure call for communications

Active Packets in ALIEN

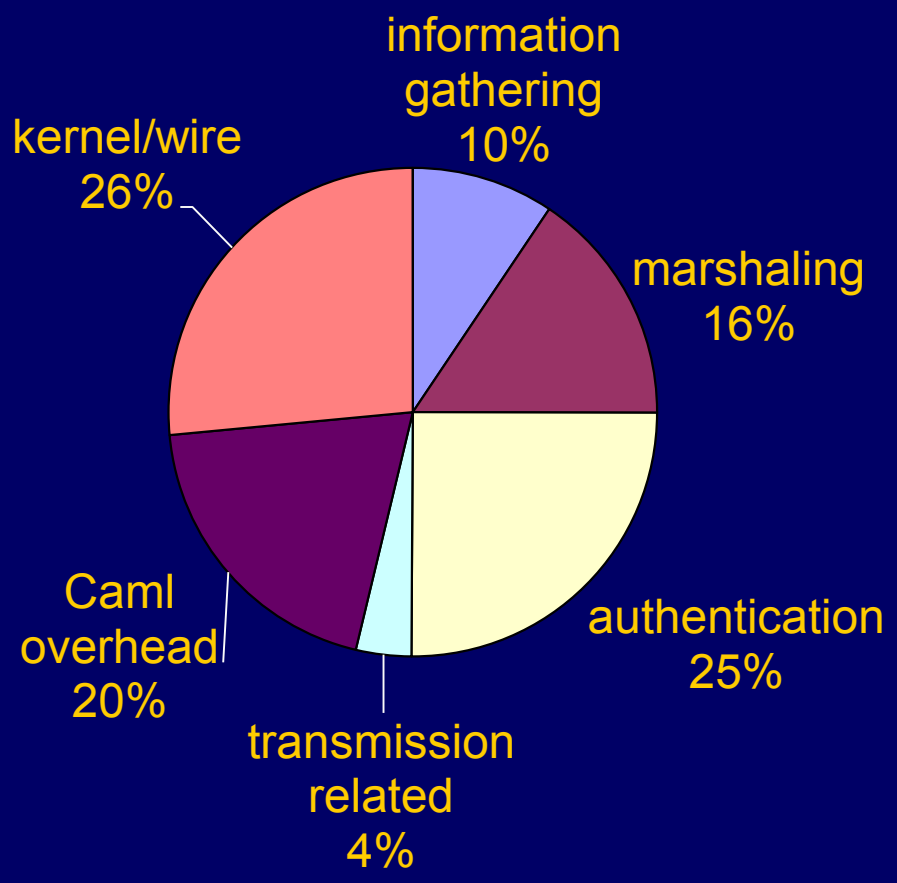
- If ANEP header indicates ALIEN
  - SANE processing as part of ANEP
  - Code portion is loaded
  - *func* is called with code, data, and func name as arguments

link layer header	ANEP header/ SANE auth	code portion	data portion	func name

saneping Performance



Overall Breakdown of Costs



Major Costs

- Kernel/Wire (26%, 3078 μs)
  - Kernel time + transmission time
  - To avoid
    - Reduce size of packet
    - Reduce or avoid kernel boundary crossing cost
- Authentication (25%, 2910 μs)
  - Mostly cost of performing SHA-1 (4 times)

Cryptography is Expensive

- Implemented in C because too slow in Caml
- Times to hash 4MB of data

	bytecode	native
Caml Int32	86.45s	61.99s
Caml int	36.03s	2.48s
C		0.33s

The take-home lesson:

- Must reduce per-packet crypto costs:
  - Active extension amortizes costs
  - ANTS caching amortizes costs
  - Smaller packets (Dense CISC, a la BBN)
- Or, find another way to avoid crypto in the common case…

6. New Approaches

- PLAN
- RCANE
- Code generation and distribution tools
- STRONGMAN

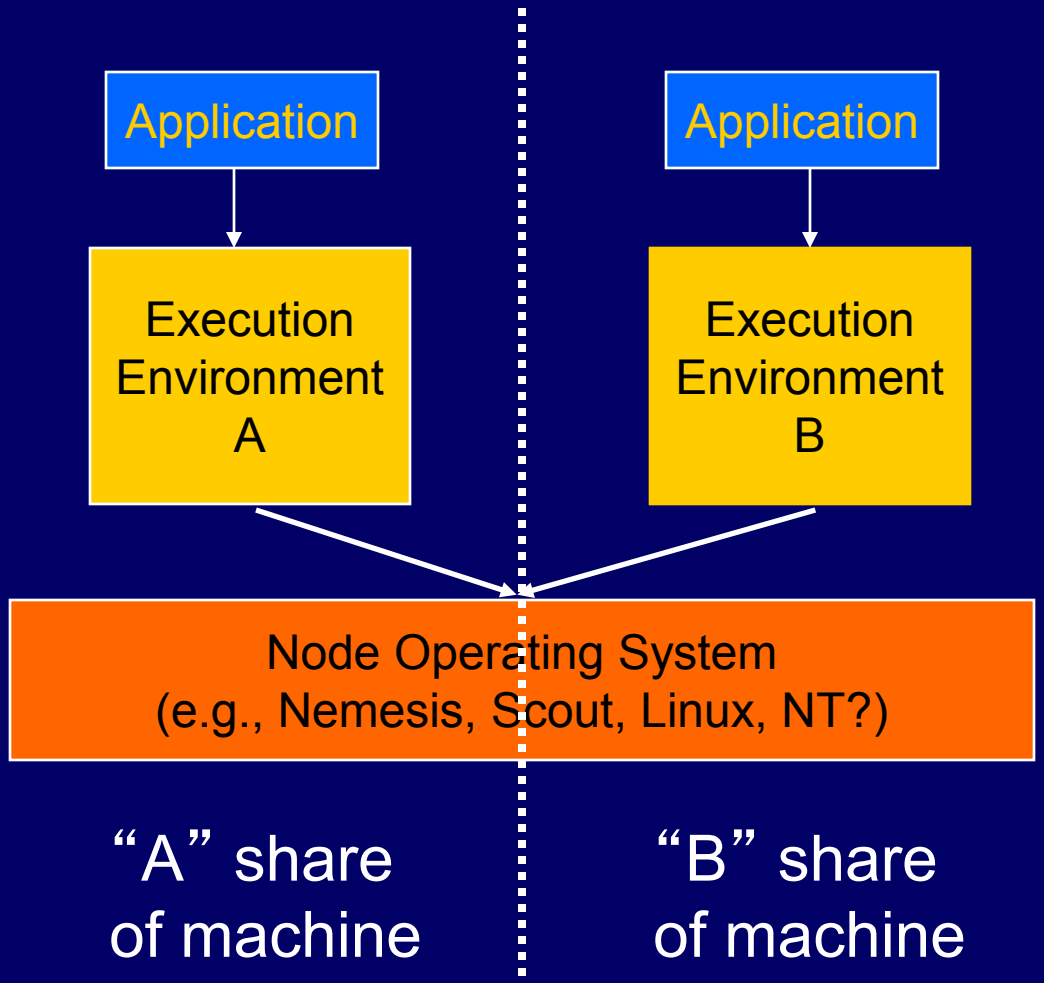
Packet Language for Active Networks (PLAN)

- Hicks, Kakkar, Moore, Gunter, Nettles
- Capsule-based approach
- CAML runtime
- Highly-restricted domain specific language (a safe "glue" language, like the UNIX shell), extensible via ALIEN
- Active extensions do restricted things

Resource Controlled Active Network Element (RCANE)

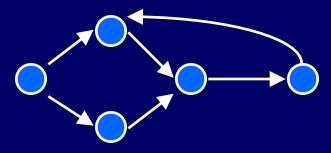
- Manage CPU, Memory and Bandwidth
  - Challenge: Modern PL heaps (GC)
  - Challenge: Interrupts
  - Challenge: CPU/Mem/BW tradeoffs
- Approach
  - Experimental RCANE with Cambridge (UK) using Nemesis O.S. for NodeOS and SwitchWare E.E.; NSF-funded at Penn; see IWAN talk by Paul Menage of Cambridge

RCANE Vertical Architecture:



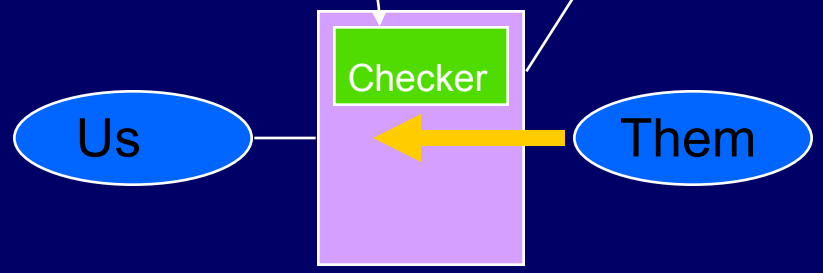
Model->Modules->Actions

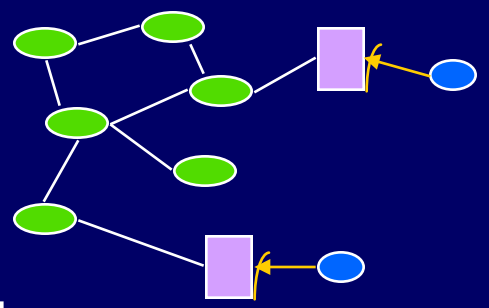
- Syntax, Semantics, Node vs. Network
- Example: Securing a Network



1. System Model

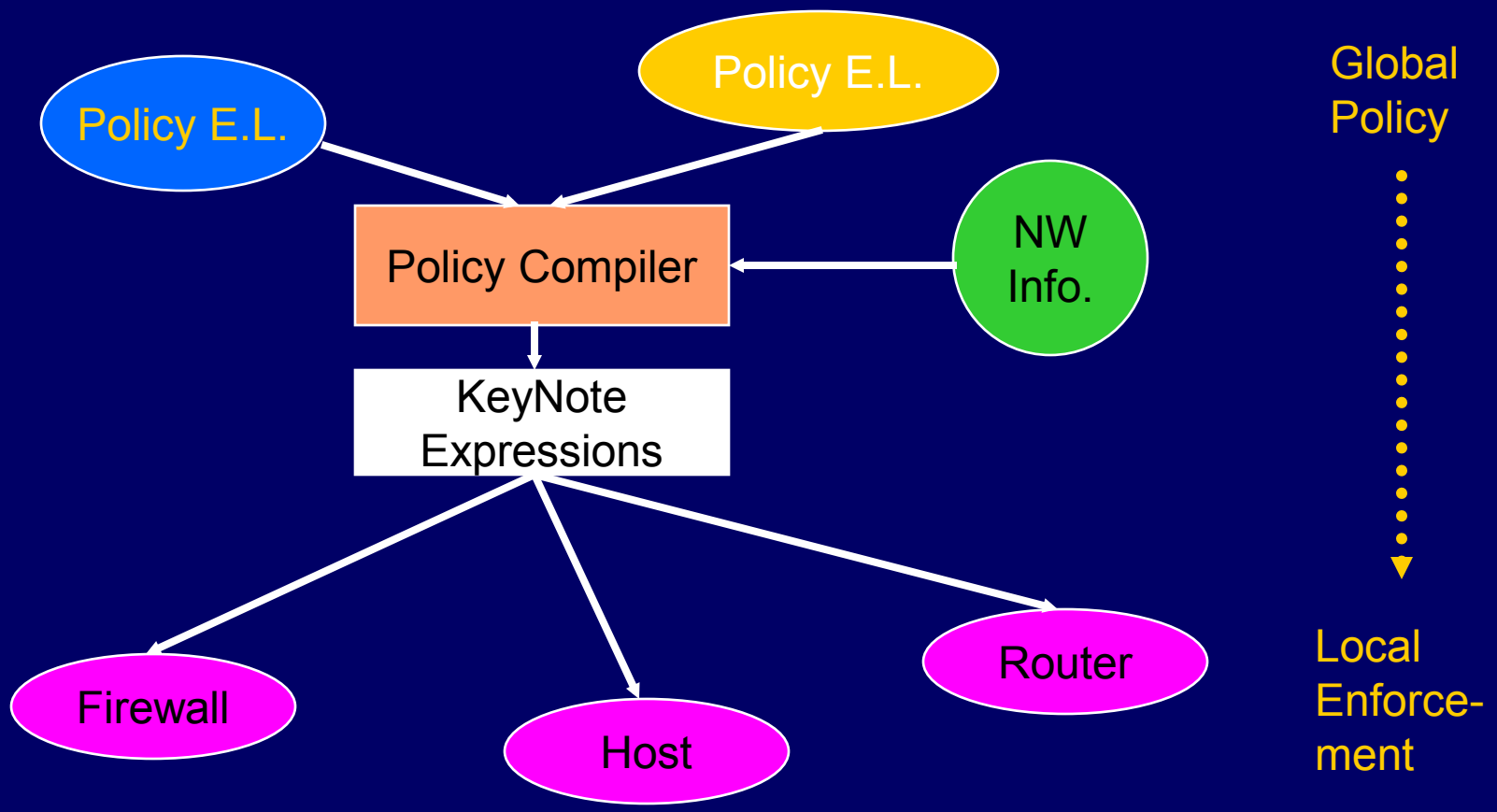
2. Modules loaded into nodes





3. Resulting in a robust Network

STRONGMAN Architecture



STRONGMAN

- Penn / AT&T Research
- Logical "meta-KeyNote"
- High-level *policy* compiles to KeyNote
- Policy-based configuration of *groups* of security endpoints (firewalls, hosts, routers, …)
- Multiple *policy expression languages* compile to common KeyNote policy model

Describing Actions in KeyNote

- <Attribute,Value> **Action Environment**

  $filename "/home/stan/foo"
  $owner "stan"
  $hostname "lake.sp.co.us"

- Attribute semantics application-specific
- An Action always associated w/Requestor

KeyNote Example

- Authorizer: *stan's public key*
- Licensees: *wendy's public key*
- Conditions: $file_owner == "stan"
  && $filename ~= "/home/stan/[^/]*"
  -> "true";
- Signature: stan's signature

7. Summary

- A.N. Security *designed in!*
- Wide variety of problems solved or being attacked
- Many are reprises of old problems
- But new tools and approaches are being developed
  - many have broader applicability