Cornell University

EME RG Initial Protocol Design Overview (NUTSS)

Paul Francis, Saikat Guha, Cornell

Scott Brim, Melinda Shore, Cisco

Status: feedback requested

- High-level draft design document written
  - draft-irtf-eme-francis-nutss-design-00.txt
  - Francis, Guha, Brim, Shore
- Also a Sigcomm paper
- Goal of draft is to get feedback on basic approach
- Still very little feedback, so hopefully this talk will provide the impetus
- I'm around until 10AM tomorrow

Requirements revisited

- Allow hosts to establish connections that honor access control policies of all stakeholders (middle and end)

- But also:
- Middlebox steering, host mobility, anycast, redirection, multi-homing, multicast, protocol negotiation
- Note absence of QoS requirements

Ramifications of requirements

- User-friendly, long-term stable, aggregatable endpoint names
  - (user-name, FQDN, app-name)
  - In large part, so that name-friendly ACL rules can be created
- Expose middleboxes to ends, and vice versa
  - Allow negotiation among them
- Especially (but not exclusively) NATs and firewalls

The Hannes Caveat

- There is lots of similar/overlapping work out there
  - HIP, NSIS, ICE, IMS, SIP, DIAMETER, RSVP, . . .
- I don't pretend to understand all of this stuff
- I hope NUTSS represents a "clean" design
- EME charter: After we do a clean design, we can revisit existing protocols
  - As well as look at incremental deployment (ICE!)

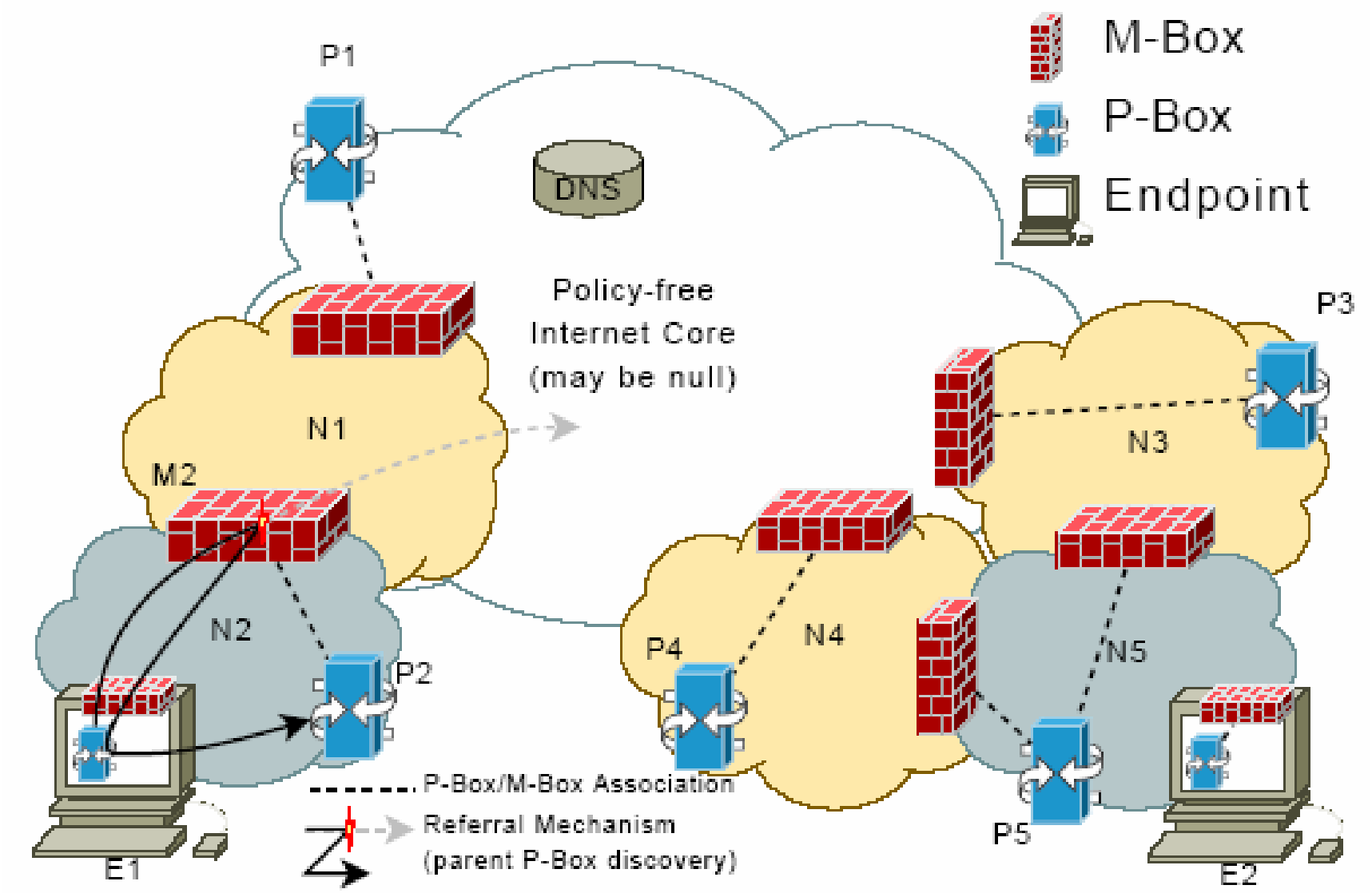
NUTSS High Points

- One signaling protocol with two routing modes:
  - Name-based routing (path decoupled)
  - Address-based routing (path coupled)
- Endpoints named as:
  <user@domain, app_name>
- Policy decision on the name-routed path
  - p-box overlay
- Policy enforcement on the address-routed path
  - m-box

Why bother with name-routing?

- Users/Apps start with a name, not an address
- Resolving name to IP address involves policy
  - (Something DNS lacks)
- May want to deploy p-boxes far from endpoint
  - DoS reasons, among others

Reference Architecture



Ref Arch points to make...

- P-box and M-box in host
- P-box overlay is “hierarchical” (parent-child, fan-in / fan-out)
  - Forest of these
  - P-box may have multiple parents
  - Reflects hierarchical structure of domains
- P-box need not be in domain
- P-box may be inside or outside of m-box
- P-boxes and M-boxes are associated with each other

P-box (name) routing

- Basically like SIP
- Up – Across – Down path
- P-boxes know how to route towards ancestors and descendants
- UP: through discovery or configuration
- DOWN: through registration along UP path
- ACROSS: P-boxes use DNS

Coupling between dual-signaling modes

- Name-routed and address-routed signaling modes are coupled together
  - Overcomes natural decoupling between IP routing and overlay routing
- P-box → M-box: P-box gives a token to M-box via the endpoint
- M-box → P-box: M-box can refer endpoint to a P-box if token fails

Basic signaling approach

- Ultimately, address-routed signaling is needed to establish a connection
- If an endpoint has an address and a token, it tries address-routed signaling
- If not, it tries name-routed signaling
  - Which obtains the address and token, as well as other negotiated parameters like ports

Signaling messages

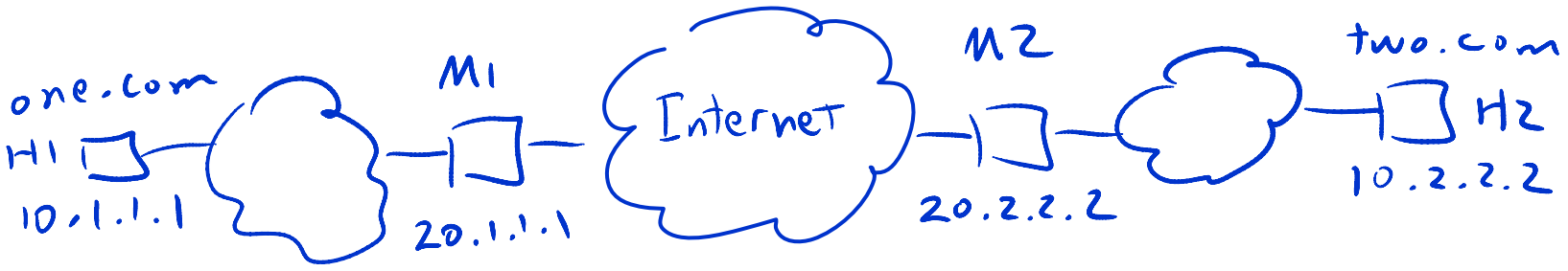
- Address-routed and name-routed use the same message format
  - Both may be loose source routed
- Signaling message contain a series of “devices”, where device = endpoint or m-box
- Signaling messages start with two devices (one for each endpoint), and additional devices, as well as additional info for existing devices, are added as the message traverses m-boxes

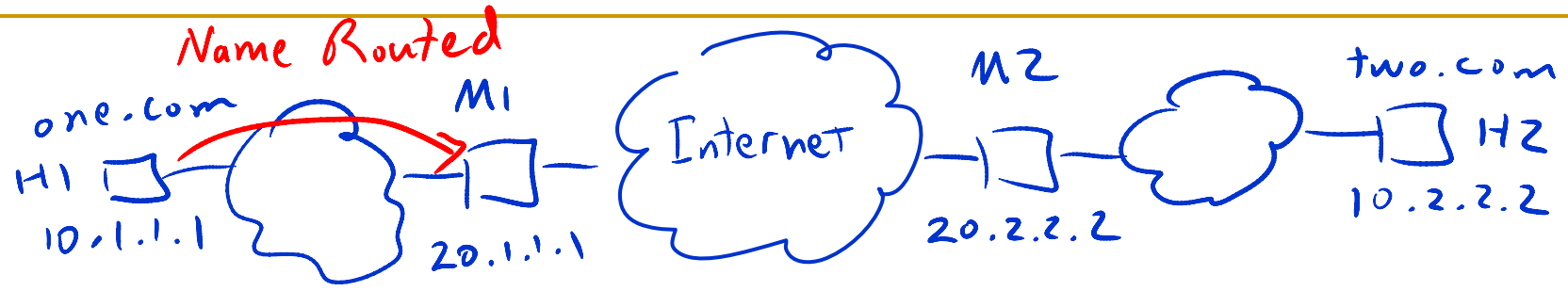
Devices in signaling message

- Device consists of:
  - Names (endpoint and app)
  - Protocol stacks and associated parameters
    - Negotiate which protocol to use
    - Understand what middleboxes are doing
    - Convey protocol parameters (address, port, etc.)
  - Tokens
  - Authentication
    - Can't be modified or deleted by others

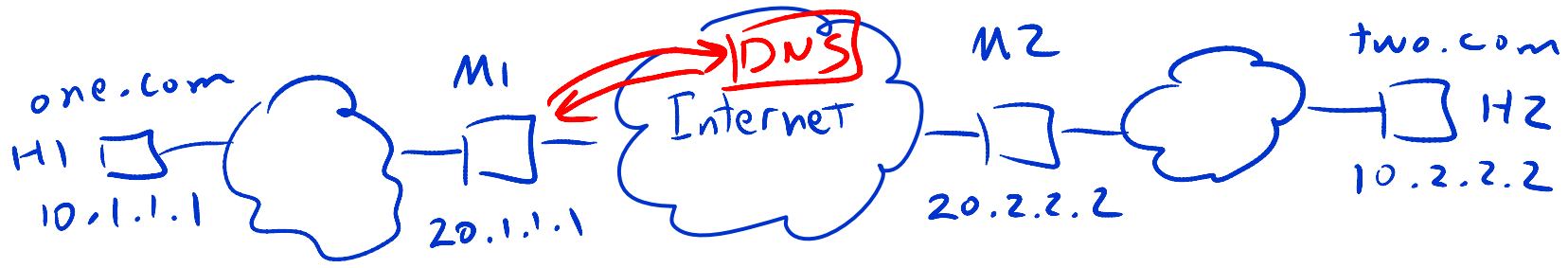
Device authentication

- Devices cannot be modified or deleted except by the device originator
- Devices can be appended (i.e. adding a port number to a protocol stack)
- Devices can be over-ridden (though the message still contains the original information, so it is clear what happened)
- This all intended to make security easier

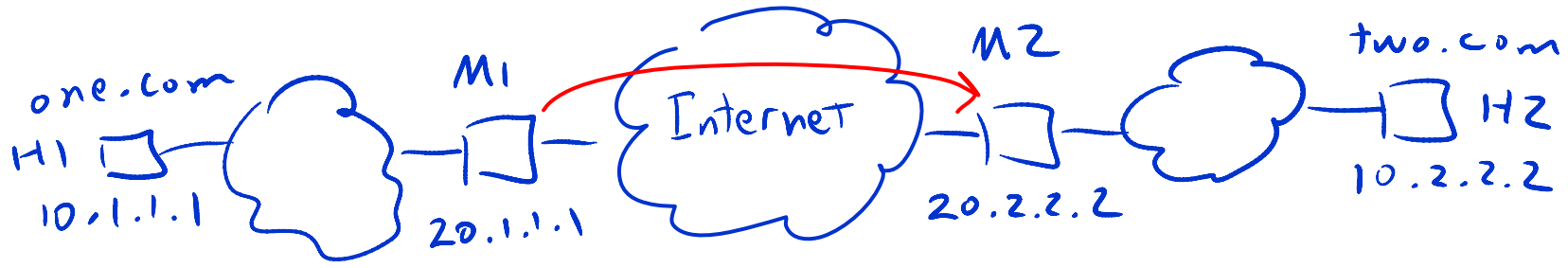

one.com
H1
10.1.1.1
M1
20.1.1.1
Internet
M2
20.2.2.2
two.com
H2
10.2.2.2



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]



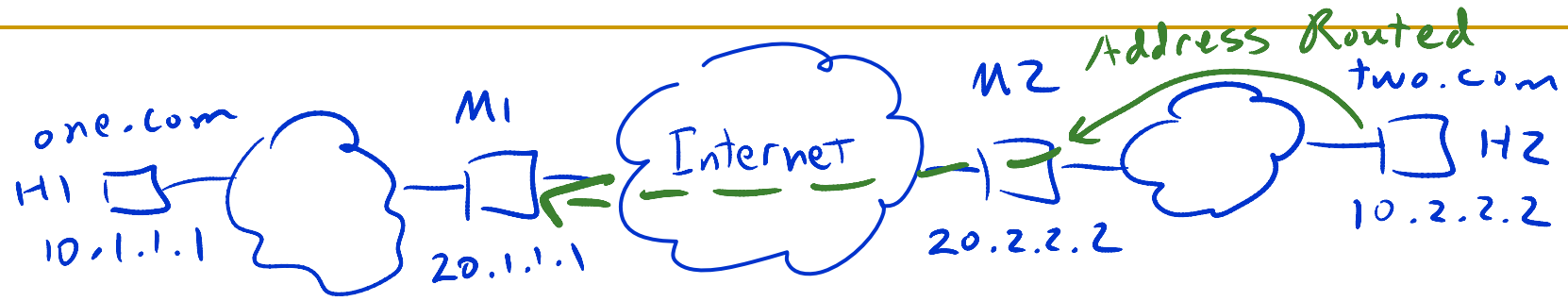
H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M1[Stack(IP-TCP-IP,
IP=20.1.1.1, TCP=1111),
Token("flow enabled")]

H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M1[Stack(IP-TCP-IP,
IP=20.1.1.1, TCP=1111),
Token("flow enabled")]

M2[Stack(IP-TCP-IP,
IP=20.2.2.2, TCP=2222),
Token("flow enabled")]

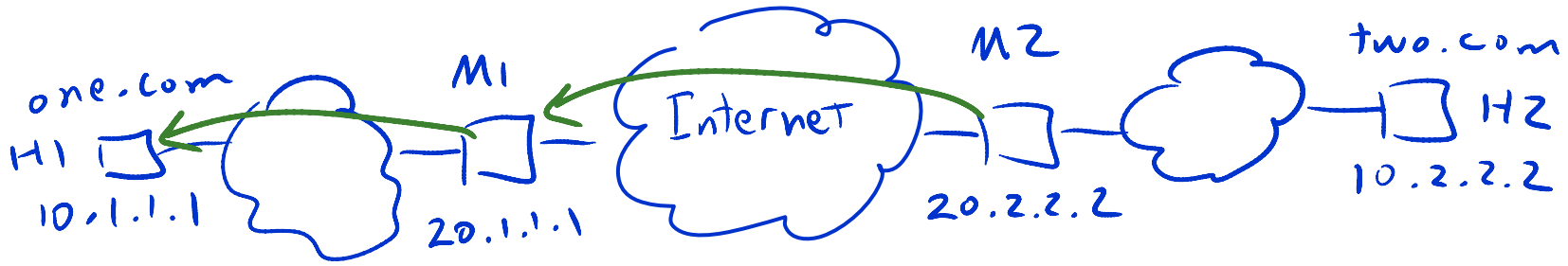


H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M1[Stack(IP-TCP-IP,
IP=20.1.1.1, TCP=1111),
Token("flow enabled")]

M2[Stack(IP-TCP-IP,
IP=20.2.2.2, TCP=2222),
Token("flow enabled")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

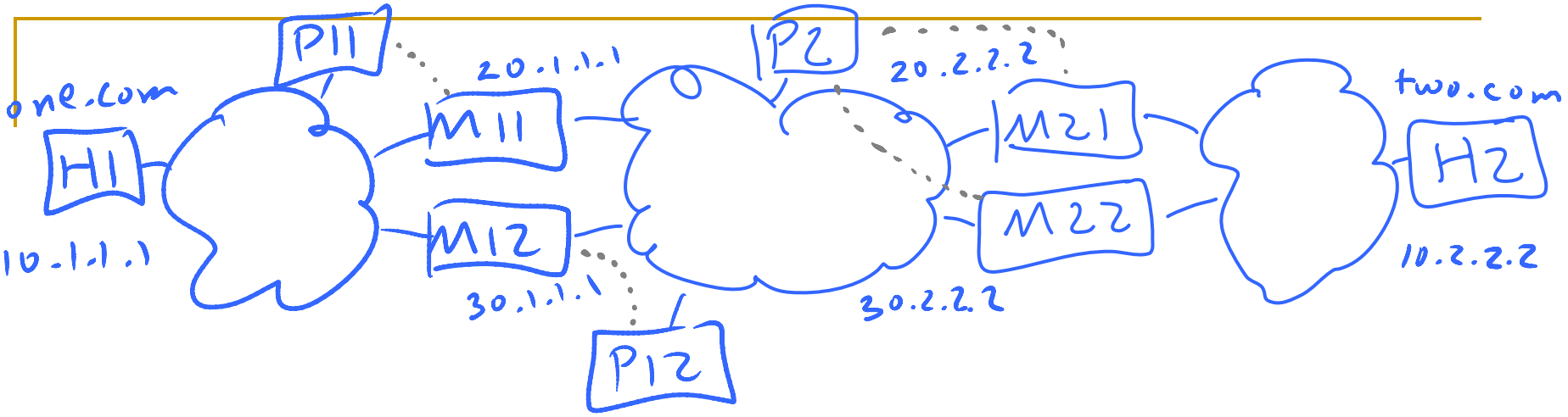


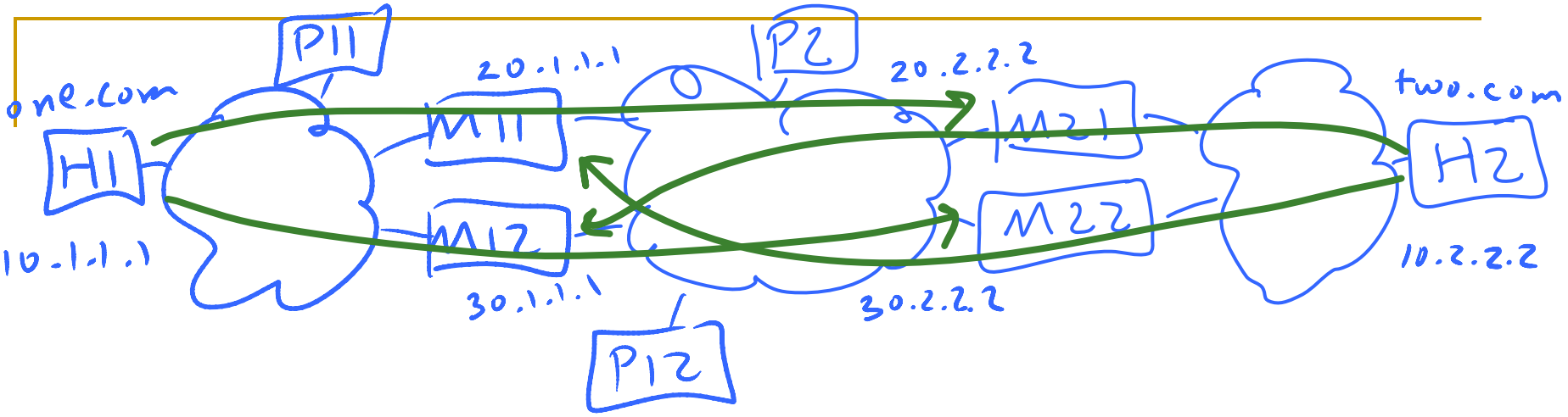
H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M1[Stack(IP-TCP-IP,
IP=20.1.1.1, TCP=1111),
Token("flow enabled")]

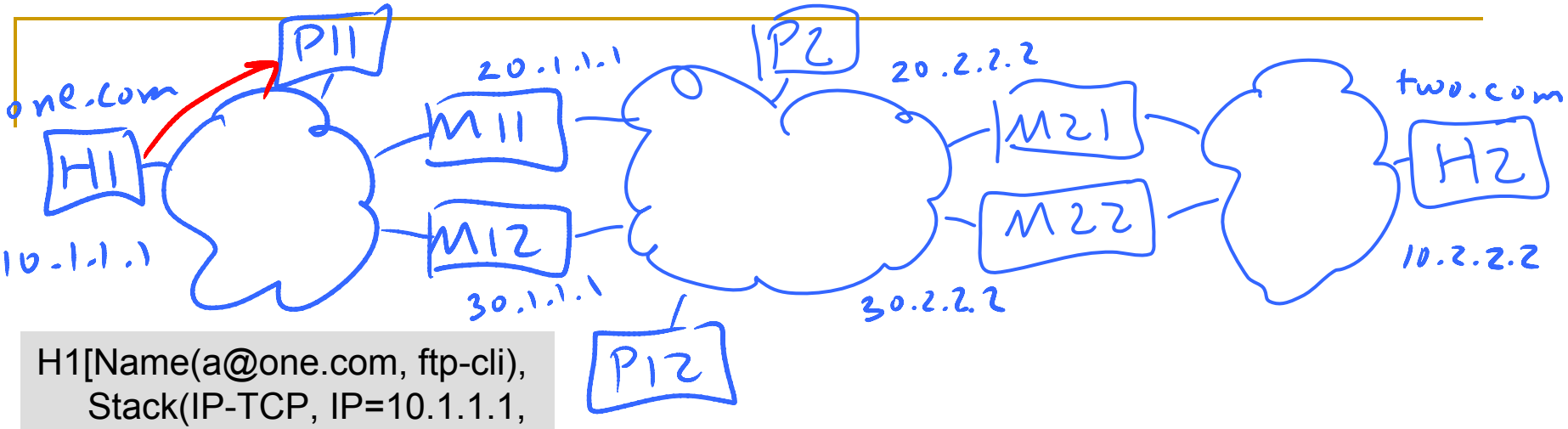
M2[Stack(IP-TCP-IP,
IP=20.2.2.2, TCP=2222),
Token("flow enabled")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

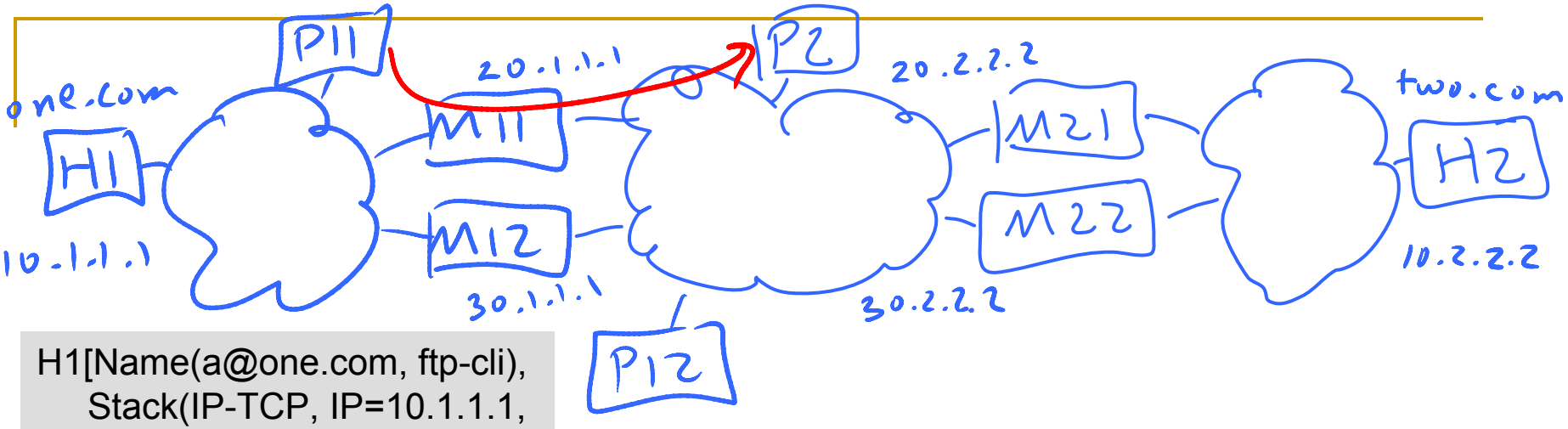

one.com
H1
10.1.1.1
P11
M11
M12
20.1.1.1
30.1.1.1
P12
P2
20.2.2.2
30.2.2.2
M21
M22
two.com
H2
10.2.2.2



Routing is asymmetric as shown here

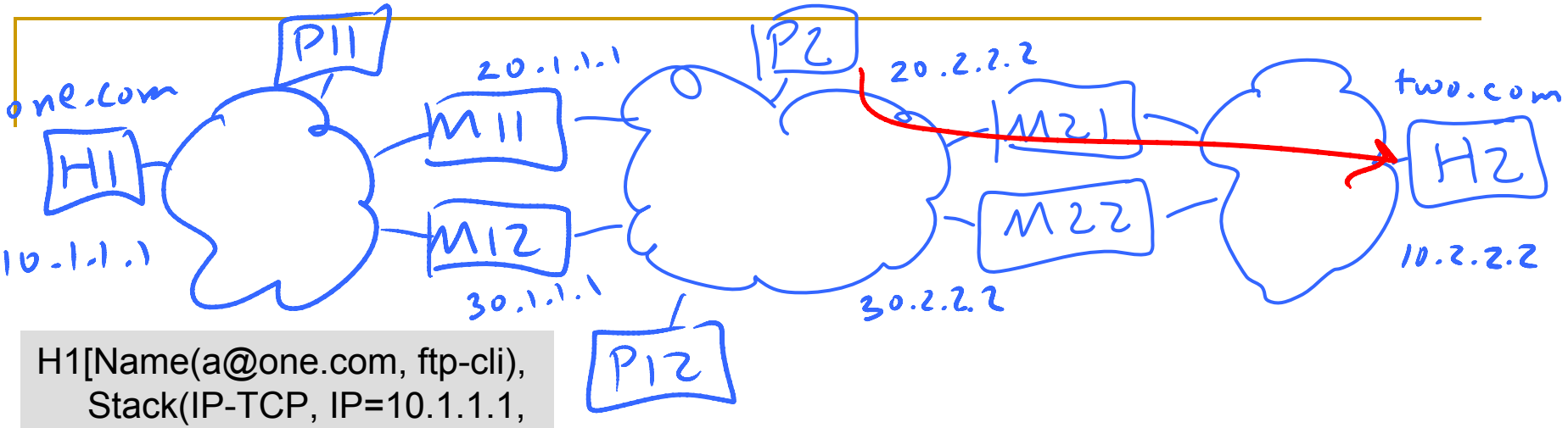


H1[Name(a@one.com, ftp-cli), Stack(IP-TCP, IP=10.1.1.1, TCP=1234)]
H2[Name(b@two.com, ftp-srv)]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

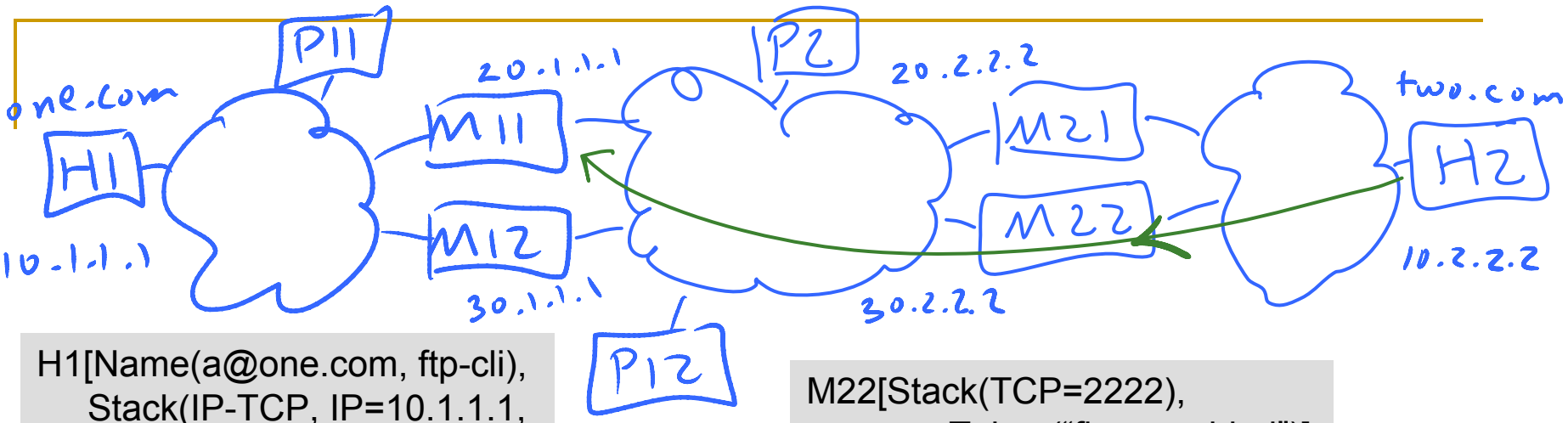
M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]



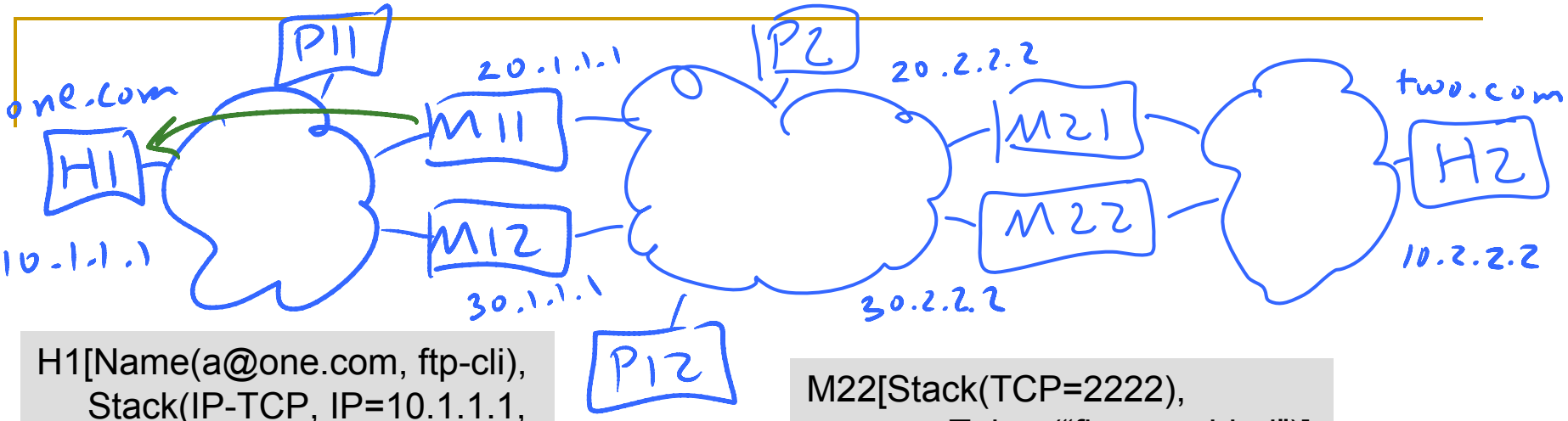
H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

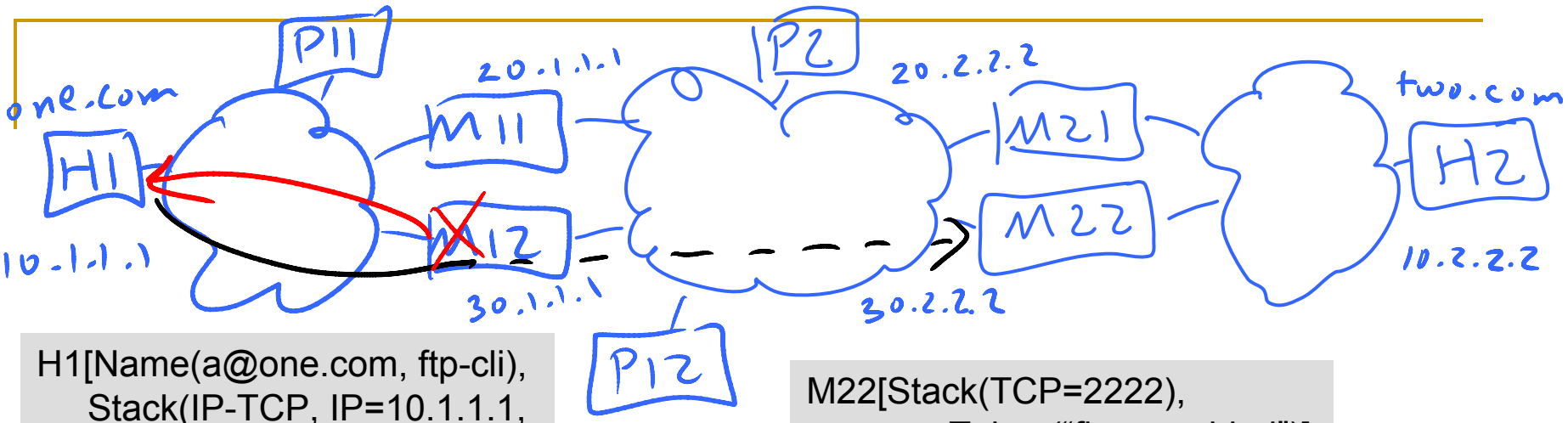
M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

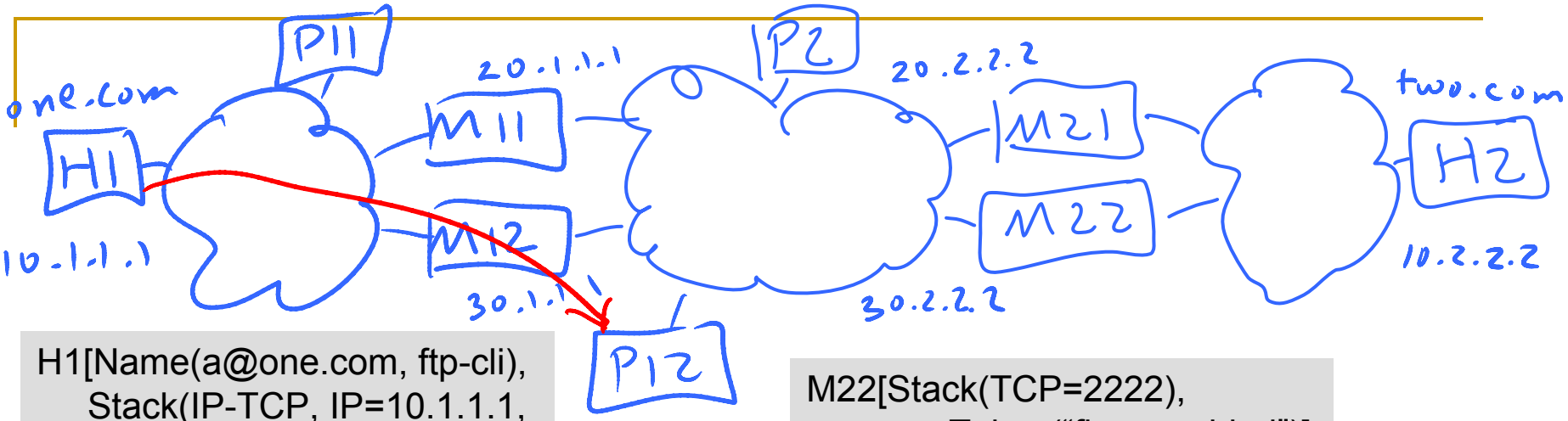
M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

Referral to P12!



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

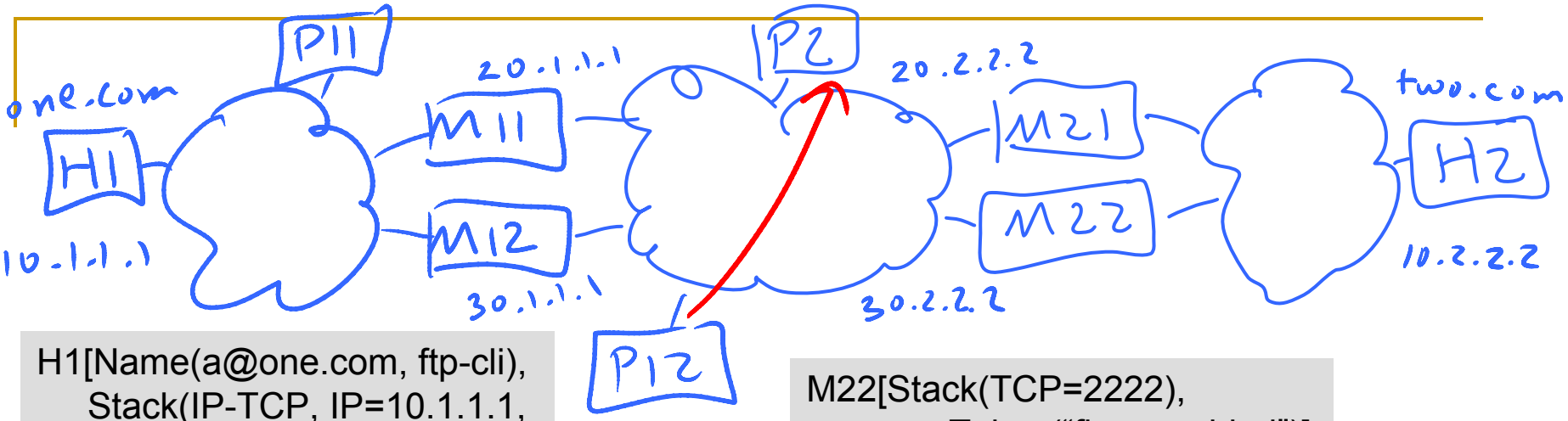
M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

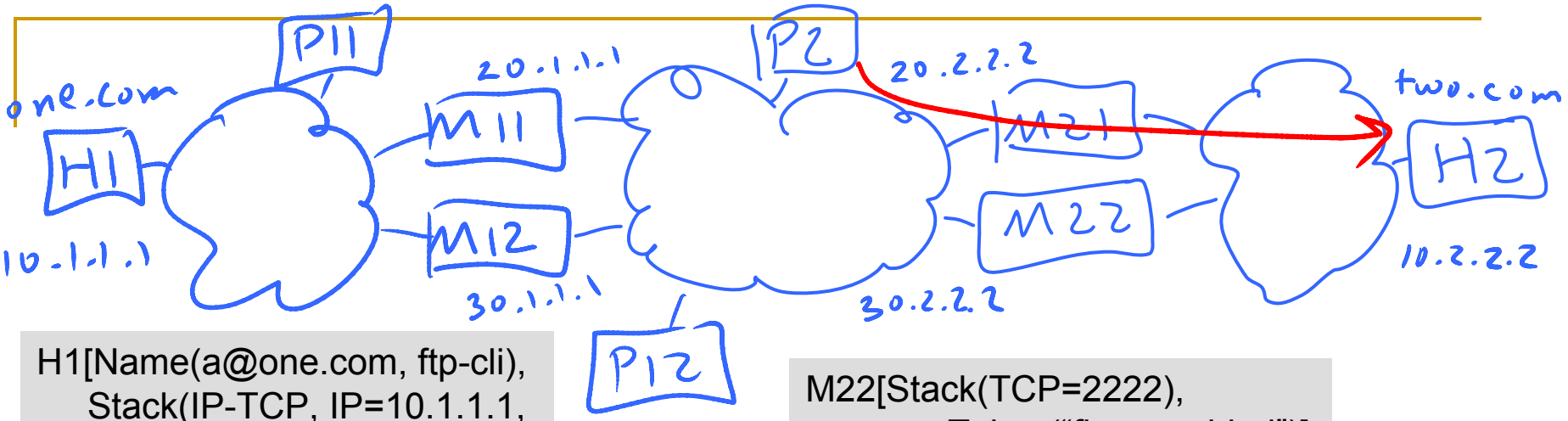
M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

M12[Stack(IP-TCP-IP,
IP=30.1.1.1),
Token("flow allowed")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

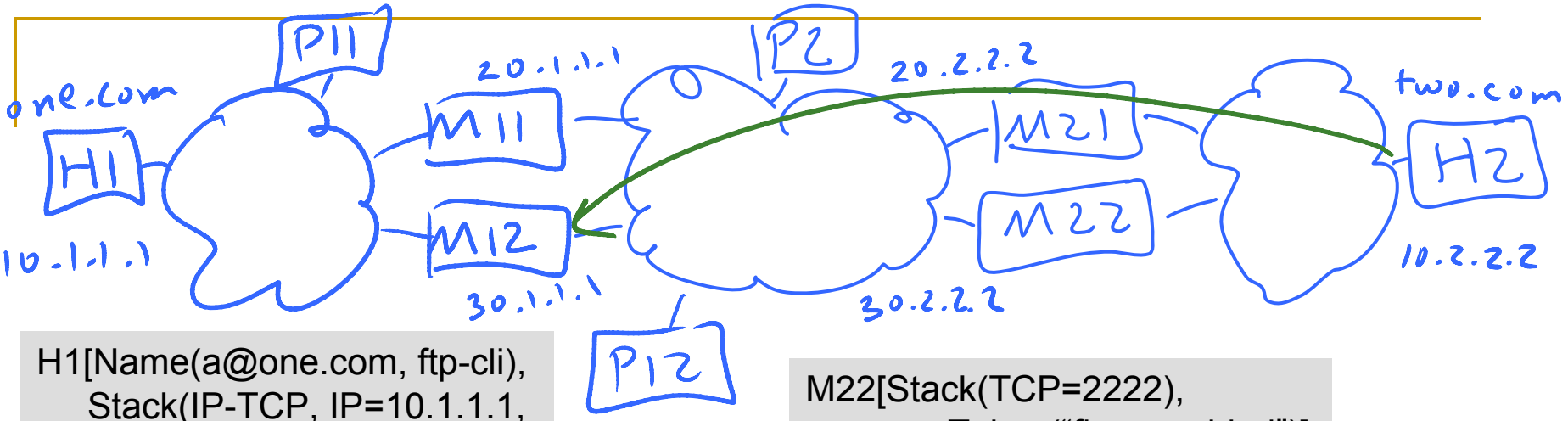
M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

M12[Stack(IP-TCP-IP,
IP=30.1.1.1),
Token("flow allowed")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

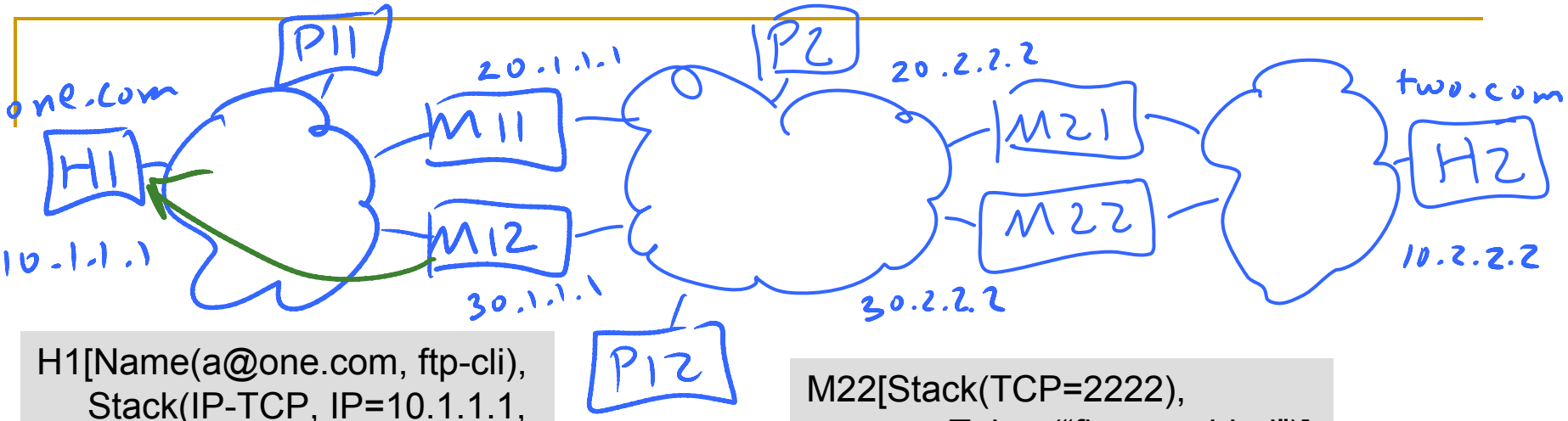
H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

M12[Stack(IP-TCP-IP,
IP=30.1.1.1),
Token("flow allowed")]

M21[Stack(TCP=3333),
Token("flow enabled")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

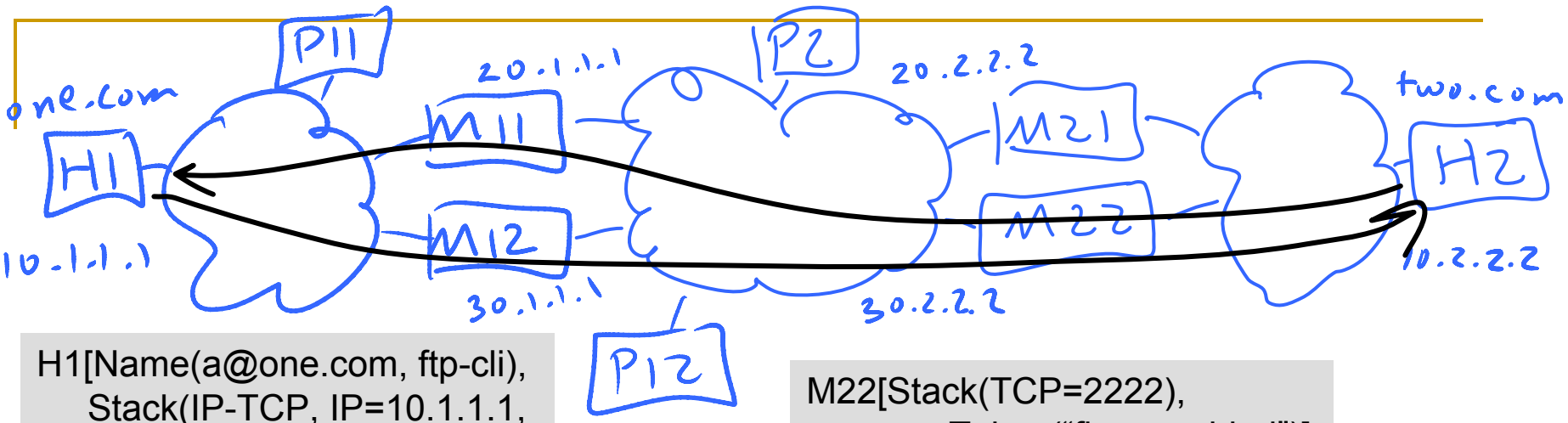
M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

M12[Stack(IP-TCP-IP,
IP=30.1.1.1),
Token("flow allowed")]

M21[Stack(TCP=3333),
Token("flow enabled")]

M12[Stack(TCP=4444),
Token("flow enabled")]



H1[Name(a@one.com, ftp-cli),
Stack(IP-TCP, IP=10.1.1.1,
TCP=1234)]
H2[Name(b@two.com, ftp-srv)]

M11[Stack(IP-TCP-IP,
IP=20.1.1.1),
Token("flow allowed")]

M21[Stack(IP-TCP-IP,
IP=20.2.2.2),
Token("flow allowed")]
M22[Stack(IP-TCP-IP,
IP=30.2.2.2),
Token("flow allowed")]

H2[Stack(IP-TCP,
IP=10.2.2.2, TCP=5678)]

M22[Stack(TCP=2222),
Token("flow enabled")]

M11[Stack(TCP=1111),
Token("flow enabled")]

M12[Stack(IP-TCP-IP,
IP=30.1.1.1),
Token("flow allowed")]

M21[Stack(TCP=3333),
Token("flow enabled")]

M12[Stack(TCP=4444),
Token("flow enabled")]

Data flows work, paths asymmetric

Next steps

- Agreement (or disagreement) that NUTSS I-D represents a good high-level approach
  - I like this direction, though
- Discussion of whether and how much to integrate with HIP (later in the agenda)
- More involvement...

Lots of interesting problems

- Secure, dynamic (non-DHCP) discovery of P-boxes, especially for mobile hosts
  - Exploit m-box/p-box shared secret, but tricky
- Support for anycast (parallel/forking versus serial)
- Non-DNS for “across” part of p-box signaling path (i.e. P2P-SIP)
- Handling private information in message “devices”
  - I.e. private address, which only the NAT needs to know
- Use of existing protocols and incremental deployment
- Etc. etc. etc.

Software

- http://nutss.net/
- Implemented name-based communication library
  - Reliable sockets for now (i.e. TCP)
  - libnutss v0.1
- SIP for offpath signaling
  - SIP registrar running at Cornell
- STUN-like onpath signaling
  - Couple of STUN-like servers on Planetlab
- v0.2 coming soon
  - Datagram sockets
  - Full ICE/STUN implementation
  - NUTSS-like unified signaling over SIP+ICE/STUN transport