XORP Tutorial

Mark Handley

Professor of Networked Systems

Department of Computer Science

UCL

18th December 2005.

Motivation

- Why is XORP the way it is?

- Three perspectives:
  - Network Researcher.
  - Network Operator.
  - Network Equipment Vendor

Motivation: Networking research

Research divorced from reality?

- Gap between research and practice in routing and forwarding.
- Most of the important Internet protocols originated in research, often at universities.
  - It used to be that researchers designed systems, *built implementations*, *tried them out*, and standardized the ones that *survived and proved useful*.
  - What happened?

Motivation: Networking research

Why the divorce?

The commercial Internet:

- Network stability is critical, so experimentation is difficult.
- Cisco and Juniper not motivated to support experimentation.

Network simulators:

- High-quality simulators make a lingua franca for research.

Motivation: Networking research

Simulation is not a substitute for experimentation

- Many questions require real-world traffic and/or routing information.
- Most grad students:
  - Give up, implement their protocol in *ns*
  - Set *ns* parameters based on guesses, existing scripts
  - Write a paper that may or may not bear any relationship to reality
  - Researchers need to be able to run experiments when required!

Motivation: Networking research

The state of the art

- Open APIs facilitate end-system protocol development:
  - WWW, RTP, SIP, RTSP, ...
- Open-source OSes do the same for kernel changes.
  - TCP SACK, IGMPv3, ...
  - Also a history of experimentation in commercial OSes (affiliated labs)
- Overlay networks may help with end-system/network interactions.
  - Field is in its infancy.

Motivation: Networking research

What about protocols that affect the routers?

Option 1:

1. Persuade Cisco to implement your protocol;
2. Persuade ISPs that your protocol won't destabilize their networks;
3. Conduct experiment.

Option 2:

1. Implement routing protocol part in MRTd, GateD, or Quagga;
2. Implement forwarding part in FreeBSD, Linux, Click, Scout, ...
3. Persuade network operators to replace their Ciscos with your PC;
4. Conduct experiment.

Likelihood of success?

Motivation: Networking research

Possible solution

Someone builds a complete open-source router software stack explicit designed for extensibility *and* robustness.

Adventurous network operators deploy this router on their networks; it develops a reputation for stability and configurability.

Result: a fully extensible platform suitable for research *and* deployment.

Motivation

Network Operators

- No two networks are alike.
- Operators need to tailor their networks to their own technical constraints and business drivers.

Motivation: Network Operators

Problem 1: Provider Lock-in

- If you're a large ISP, Cisco may listen to you.
  - You need a feature.
  - They write it, ship you a custom IOS image.
  - You test and debug it.

- If you're not AT&T, MCI, NTT, Sprint, etc:
  - You need a feature.
  - If it's not already shipping, forget it.
  - Hack your way around the problem using the existing feature set.
  - Result: network is more complex and less stable.

Motivation: Network Operators

Problem 2: Unstable experimental code

- Most router stacks are a disaster for deployment of experimental code.
  - Routing code is really hard to debug.
  - If the codebase is less than 5 years old, expect bugs.
  - A bug in experimental code will likely crash your router.

- How can an ISP experiment with next year's services without destabilizing current money-making services?
  - Usually need to use a parallel set of routers.
  - Eg, IPv6, multicast deployments.
  - Expensive.

Motivation: Network Operators

Problem 3: Special Purpose Networks

- Suppose you’re a large investment bank.
  - Your network is not a general-purpose network.
  - Tailored carefully to your specific applications.
  - Eg. multicast in financial services industry.
  - Eg. PGM to support Tibco.
- Router vendors have very limited interest in customizing their software for your applications.
  - Lots of in-house developed or bespoke software in the financial services industry.
  - Routers are the one place they can’t run custom or third-part software.

Motivation

- Network Equipment Vendors
  - There are only two trusted routing stacks:
    - Cisco, Juniper.
  - Many ISPs don’t trust anyone else.

- Time to market for hardware is ~18-24 months.
  - Smart hardware designers can build hardware that is better, or fills a niche.
  - Still can’t break into the ISP market because no-one trusts their software.

Motivation: Network Equipment Vendors

Disrupting the Market

Separate router hardware from core routing software.

- New hardware vendors concentrate on providing best price/performance for hardware.
- Use same core software, plus their own customizations.
  - Widespread usage ensures the core codebase is stable and widely trusted.
- Open software APIs and extensible design allows a market to develop for router applications
  - More like desktop software for Windows.
  - Wide availability of useful software makes hardware platforms more attractive that closed platforms.

Motivation: Network Equipment Vendors

Disrupting the Market

- Open software APIs not sufficient if the software platform is proprietary to a single hardware vendor.
  - Software vendors will fear lock-in to one hardware vendor.
  - Less leverage on investment => less third-party router software developed.

- Need a *vendor-neutral* core router software platform.
  - Open source allows the codebase to gain functionality fastest, and allows bugs to be fixed more easily.

Motivation

- Enabling Internet Evolution

Stalled Evolution in the Internet Core

Stability matters above all else.

- ISPs can’t afford routing problems.
- Won’t run anything experimental on production routers for fear of affecting production traffic.

Building stable routing protocol implementations is really hard.

- Big router vendors don’t have a lot to gain from innovation, because ISPs *can’t afford to run experimental code*.
- Startups can’t help because of the *closed nature of the software market for IP routers*.

Important routing problems remain unaddressed.

Extensible Router Control Plane Software

Extensibility could solve this problem:

- Allow experimental deployment of new routing protocols
- Enable router application market

Extensible forwarding planes exist:

- Network Processors, FPGAs, software (Click, Scout, ...)
- But not control planes: why?

The demands of router software make extensibility hard:

- Stability requirement
- Massive scale distributed computation
- Tight coupling between functions
- Routing protocols themselves not designed for extensibility

Four Challenges

Features

Real-world routers must support a long feature list.

Extensibility

Every aspect of the router should be extensible.

Extensions must be able to co-exist gracefully.

Performance

Scalability in routing table size, low routing latency.

Robustness

Must not crash or misroute packets.

Fast Convergence

Routing protocol implementations have often been *scanner-based*.

- Periodically a scanner runs to accumulate changes, update the forwarding table, notify neighbors, etc.
- Easy to implement.
- Low CPU utilization.
- Poor route convergence properties.

Fast convergence is now a priority.

- Event-driven router implementations are needed to respond to change as quickly as possible.
- Events processed to completion.
- Explicit dependency tracking.
- *Harder to implement, especially in an extensible way.*

XORP

eXtensible Open Router Platform

Open source router software suite, *designed from the outset with extensibility in mind.*

- Main core unicast and multicast routing protocols.
- Event-driven multi-process architecture.
- BSD-style license
- 560,000 lines of C++

XORP Contributions

- Staged design for BGP, RIB.
- Scriptable inter-process communication mechanism.
- Dynamically extensible command-line interface and router management software.
- Extensible policy framework.

First fully extensible, event-driven, open-source routing protocol suite: www.xorp.org.

XORP Status: IGP Standards

RIP and RIPng:

- RFC 2453 (RIP version 2)
- RFC 2082 (RIP-2 MD5 Authentication)
- RFC 2080 (RIPng for IPv6)

OSPFv2:

- RFC 2328 (OSPF Version 2)
- RFC 3101 (The OSPF Not-So-Stubby Area (NSSA) Option)

XORP Status: BGP Standards

- **draft-ietf-idr-bgp4-26** (A Border Gateway Protocol 4 (BGP-4))
- **RFC 3392** (Capabilities Advertisement with BGP-4)
- **draft-ietf-idr-rfc2858bis-03** (Multiprotocol Extensions for BGP-4)
- **RFC 2545** (Multiprotocol Extensions for IPv6 Inter-Domain Routing)
- **RFC 3392** (Capabilities Advertisement with BGP-4)
- **RFC 1997** (BGP Communities Attribute)
- **RFC 2796** (BGP Route Reflection - An Alternative to Full Mesh IBGP)
- **RFC 3065** (Autonomous System Confederations for BGP)
- **RFC 2439** (BGP Route Flap Damping)

XORP Status: Multicast Standards

PIM-SM:

- draft-ietf-pim-sm-v2-new-11 (without SSM).
- draft-ietf-pim-sm-bsr-03

IGMP v1 and v2:

- RFC 2236

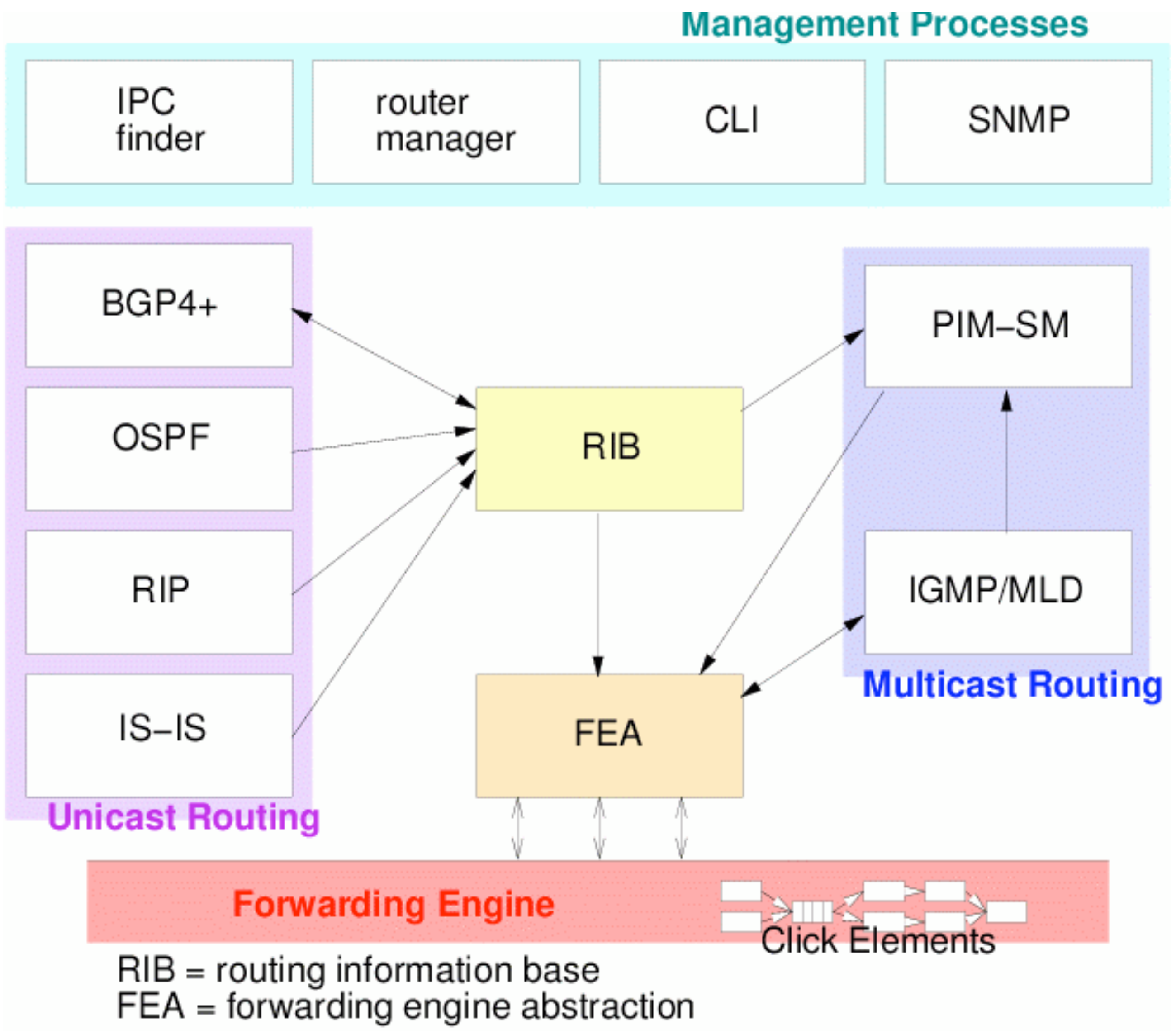
MLD v1:

- RFC 2710

XORP Processes

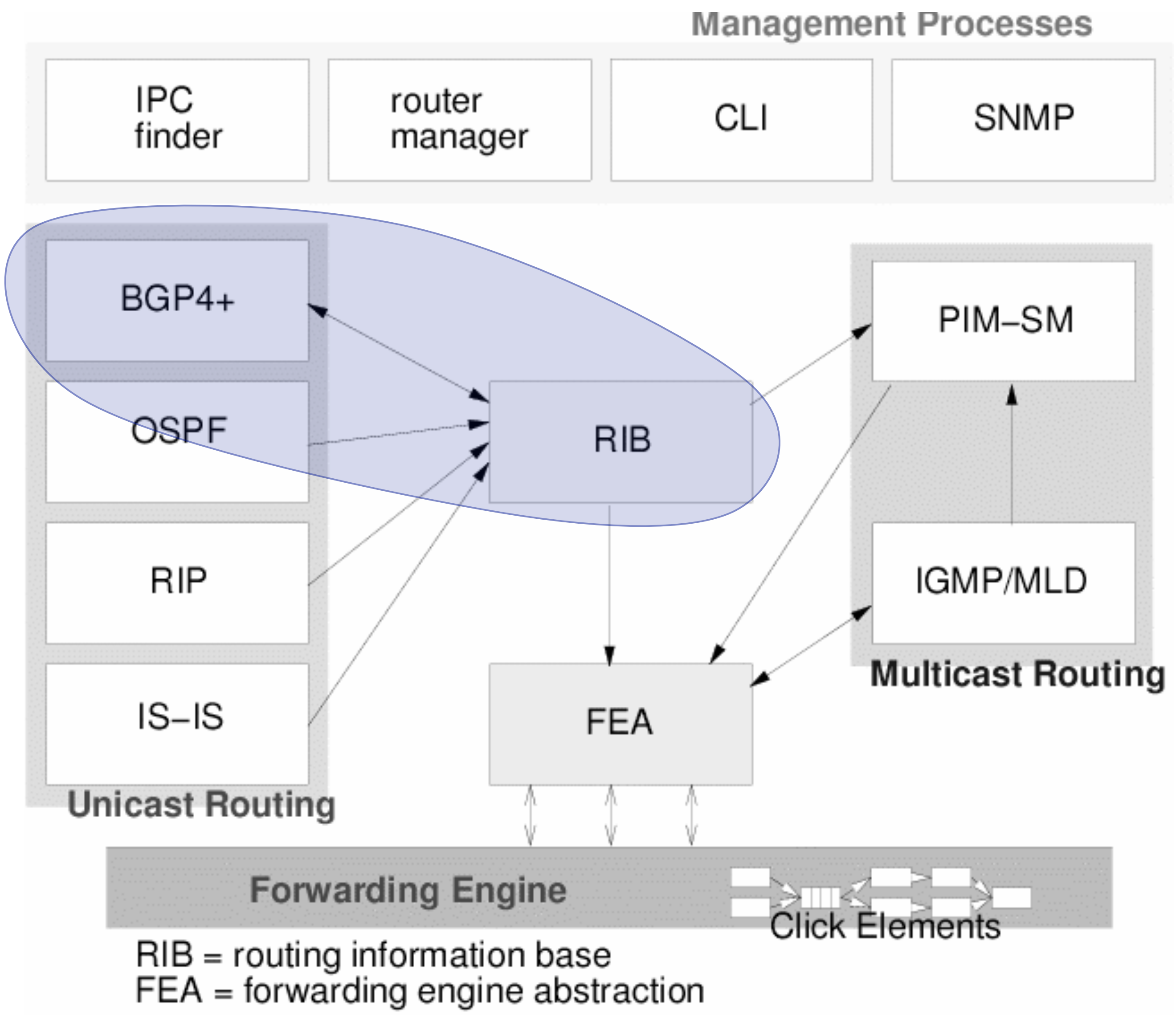
Multi-process architecture, providing isolation boundaries between separate functional elements.

Flexible IPC interface between modules



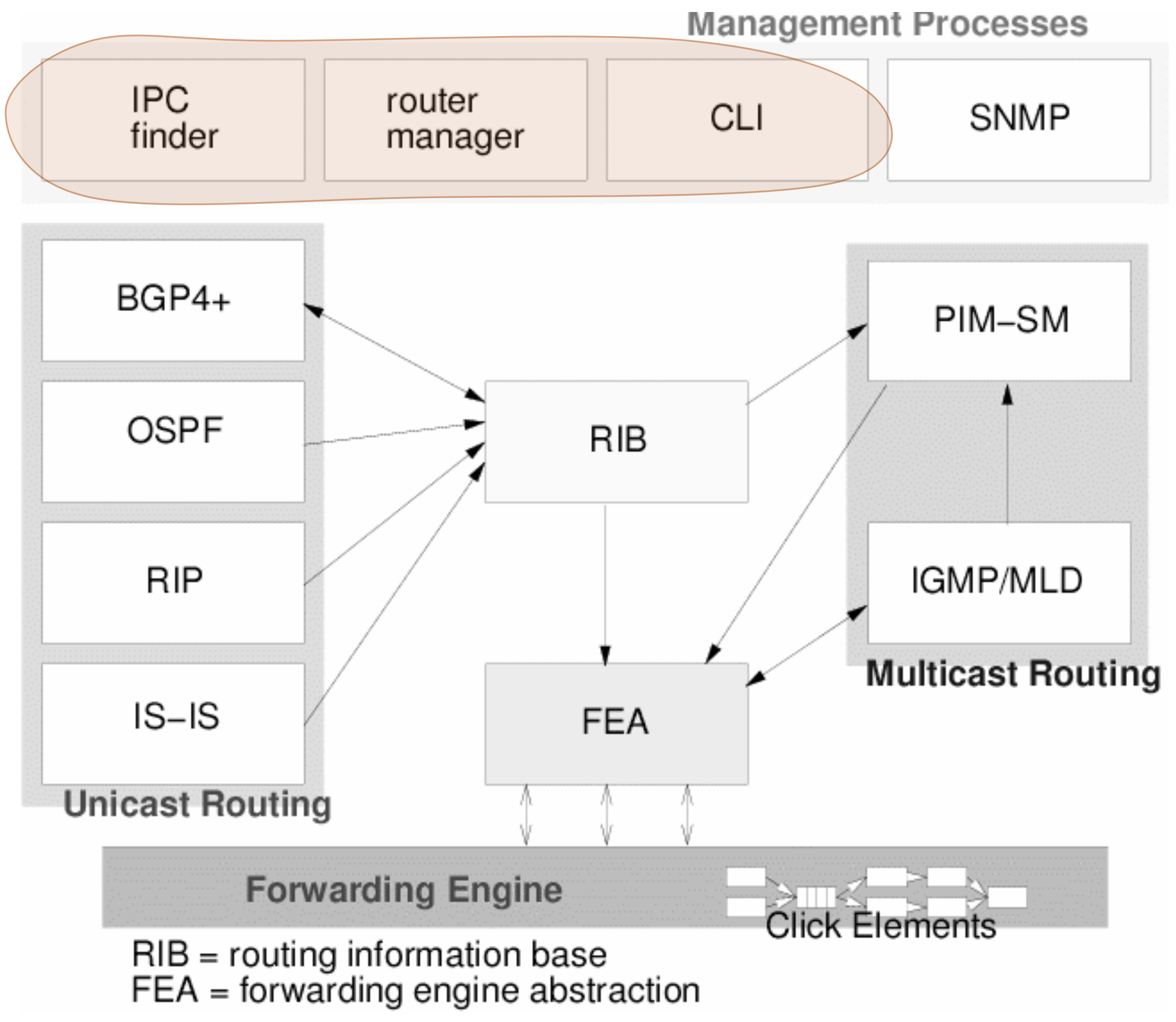
Outline of this talk

1. Routing process design



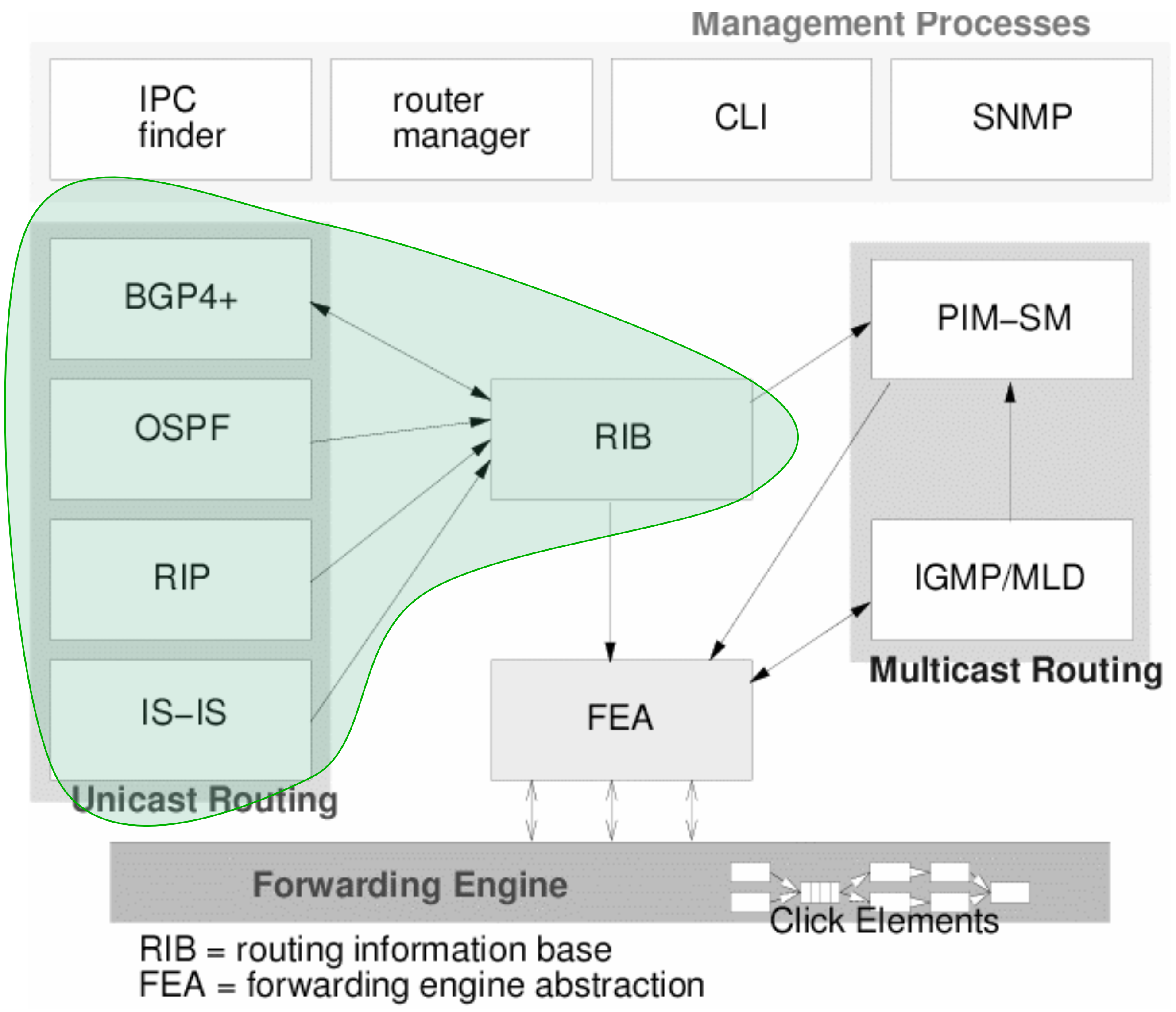
Outline of this talk

1. Routing process design
2. Extensible management framework



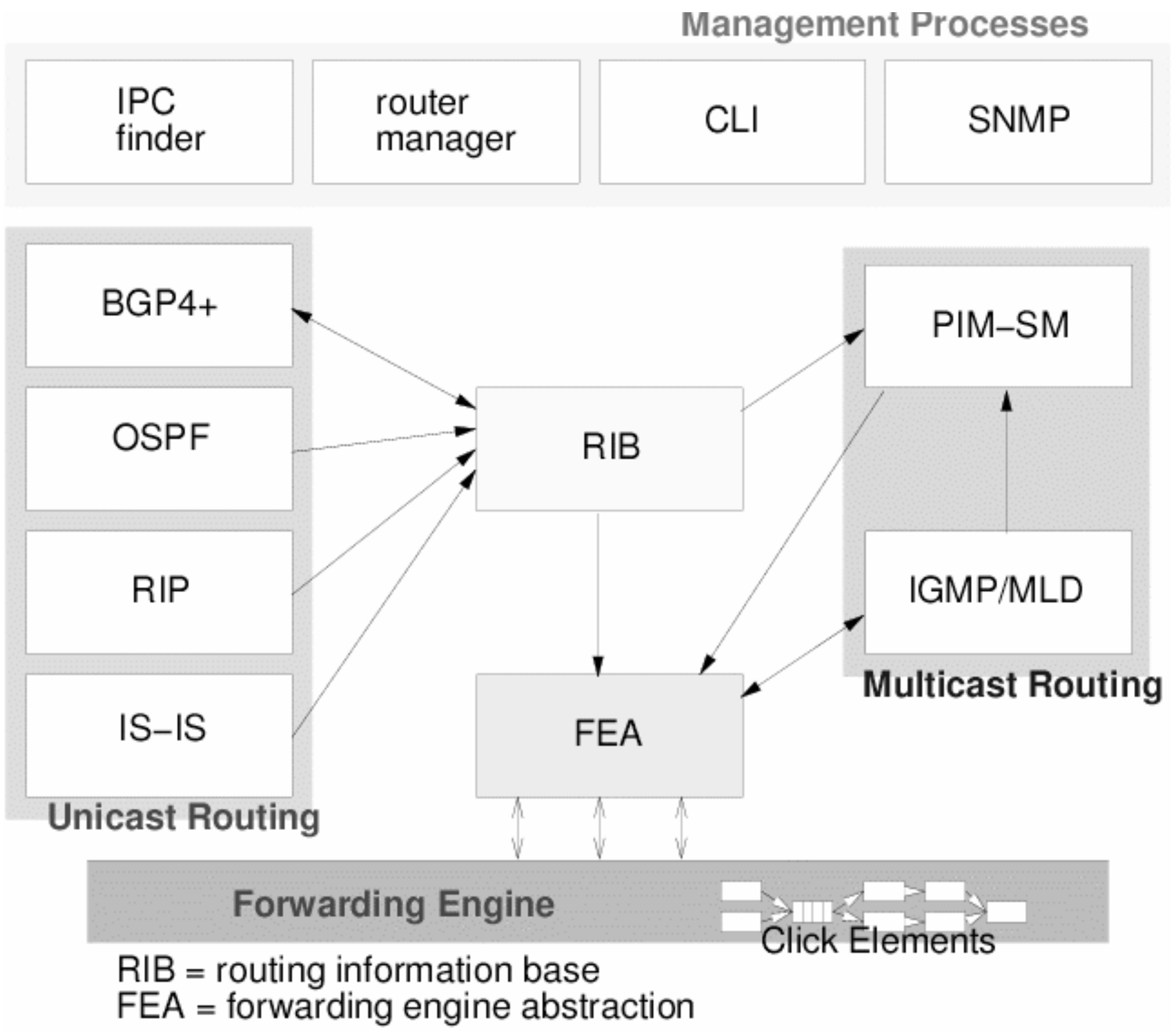
Outline of this talk

1. Routing process design
2. Extensible management framework
3. Extensible policy routing framework



Outline of this talk

1. Routing process design
2. Extensible management framework
3. Extensible policy routing framework
4. Performance results

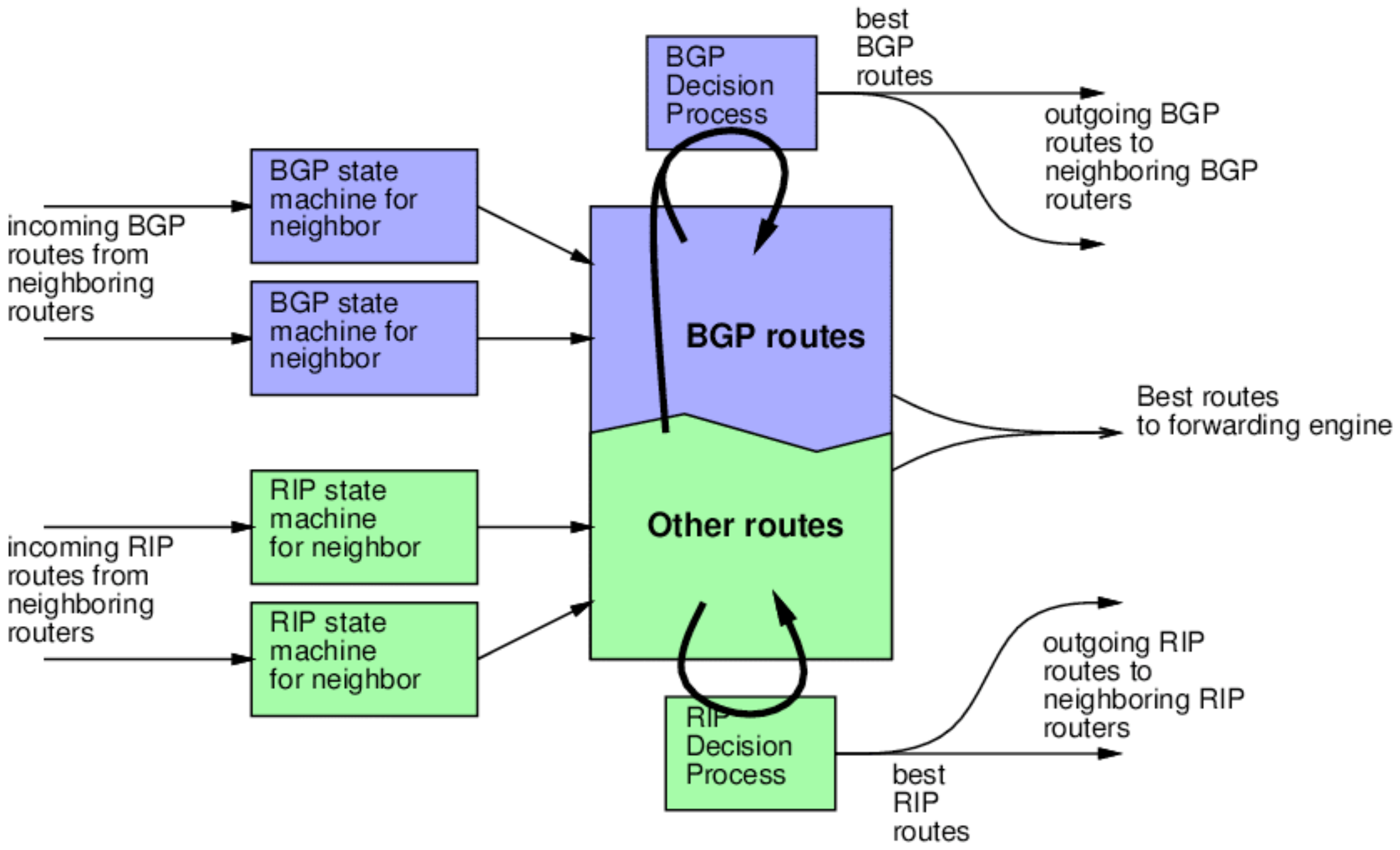


RIB = routing information base
FEA = forwarding engine abstraction

Routing Process Design

- How do you implement routing protocols in such a way that they can easily be extended in the future?

Conventional router implementation

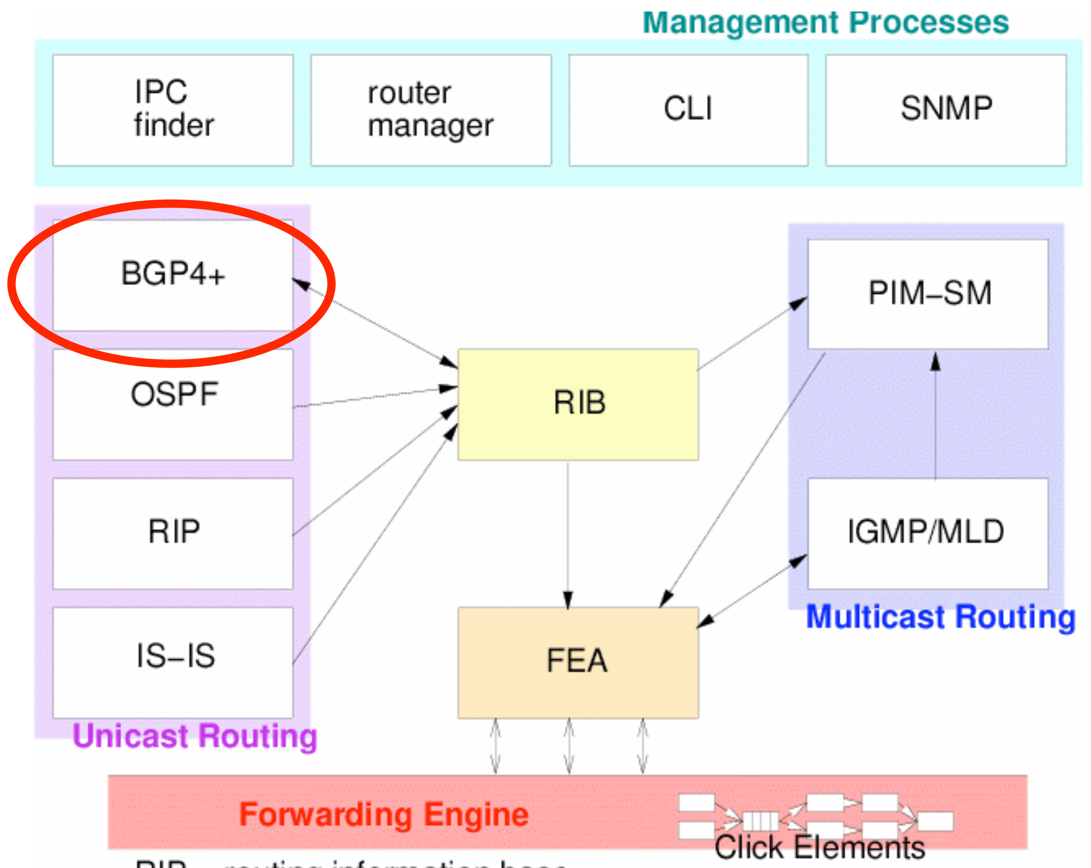

best BGP routes
BGP Decision Process
outgoing BGP routes to neighboring BGP routers
BGP state machine for neighbor
incoming BGP routes from neighboring routers
BGP state machine for neighbor
BGP routes
Best routes to forwarding engine
RIP state machine for neighbor
Other routes
incoming RIP routes from neighboring routers
RIP state machine for neighbor
outgoing RIP routes to neighboring RIP routers
RIP Decision Process
best RIP routes

Implementing for Extensibility

- Tightly coupled architectures perform well, but are extremely hard to change without understanding how all the features interact.

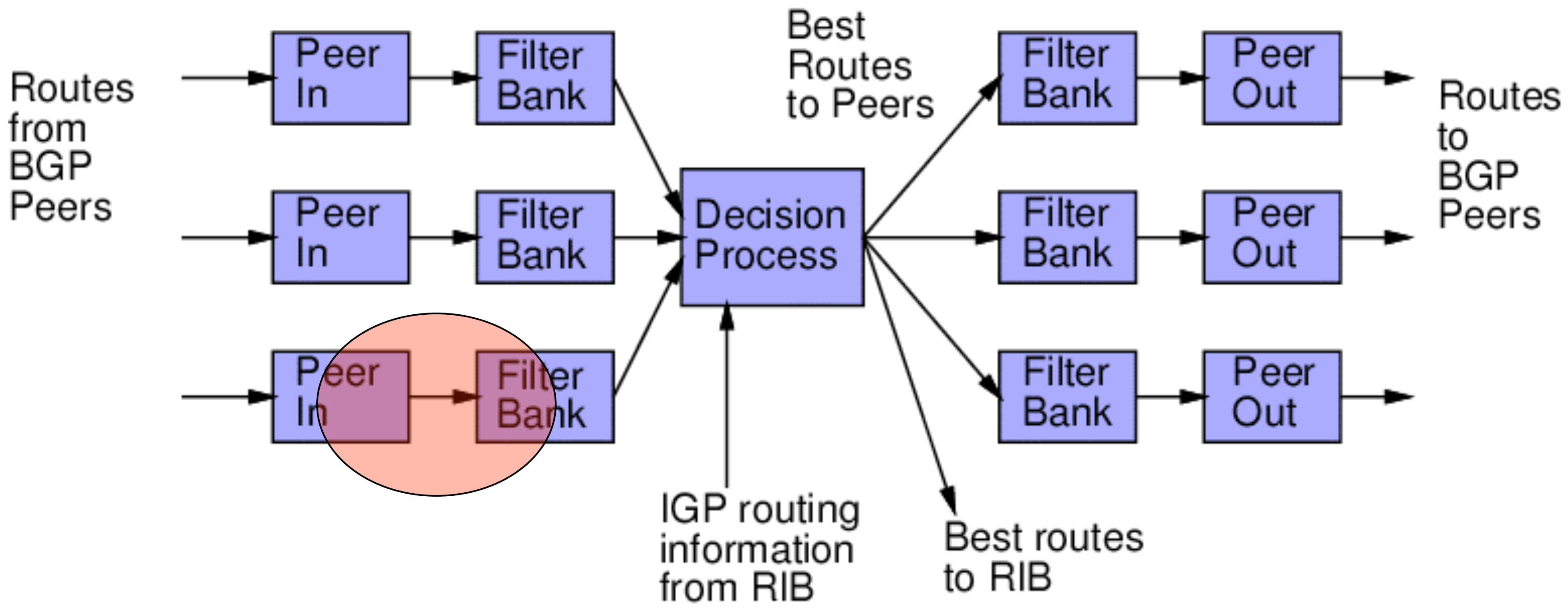
- Need an architecture that permits future extension, while minimizing the need to understand all the other possible extensions that might have been added.
  - We chose a *data-flow architecture*.
  - Routing tables are composed of dynamic processes through which routes flow.
  - Each stage implements a common simple interface.

BGP

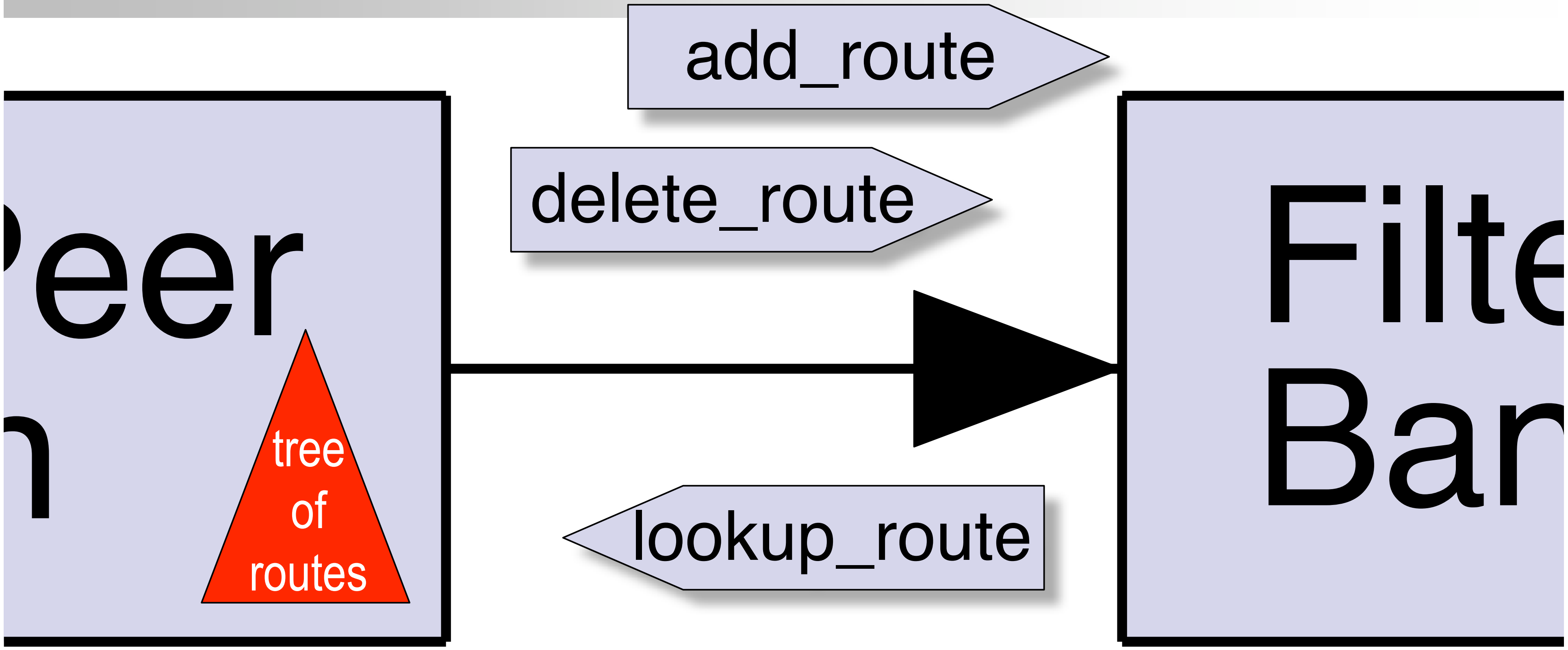


RIB = routing information base
FEA = forwarding engine abstraction

BGP Staged Architecture



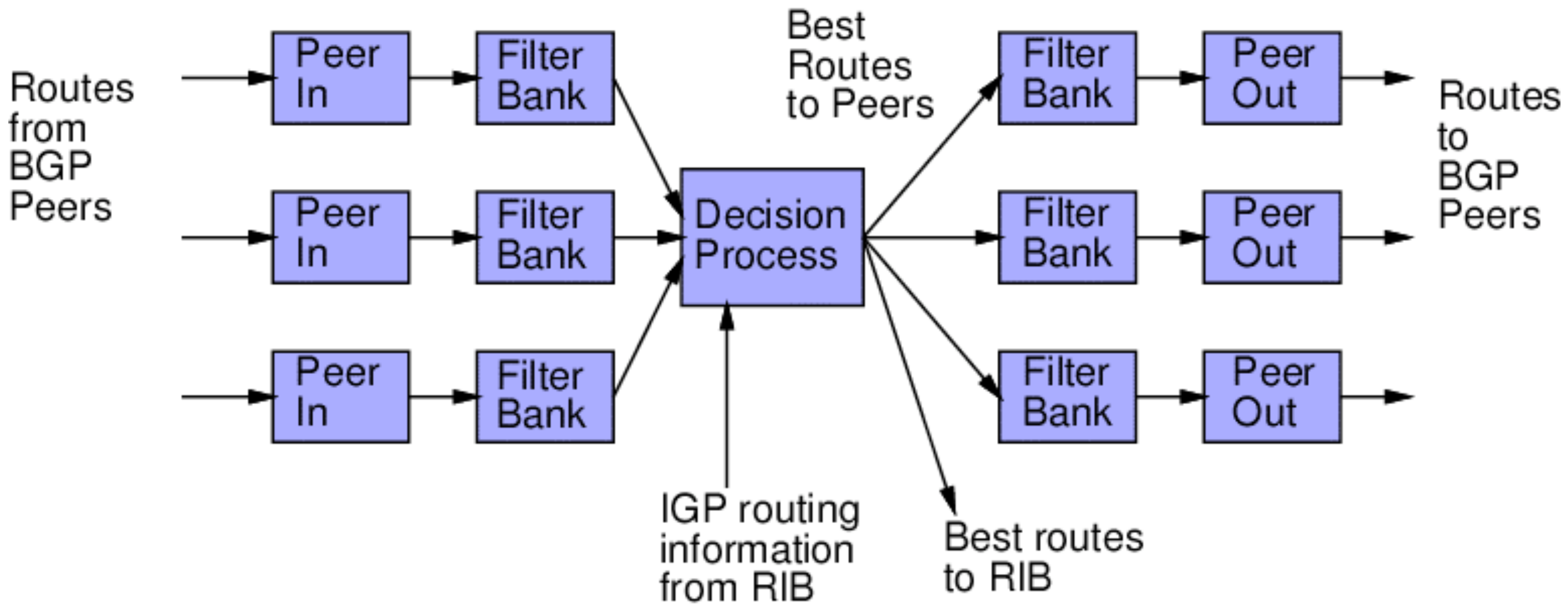
Messages



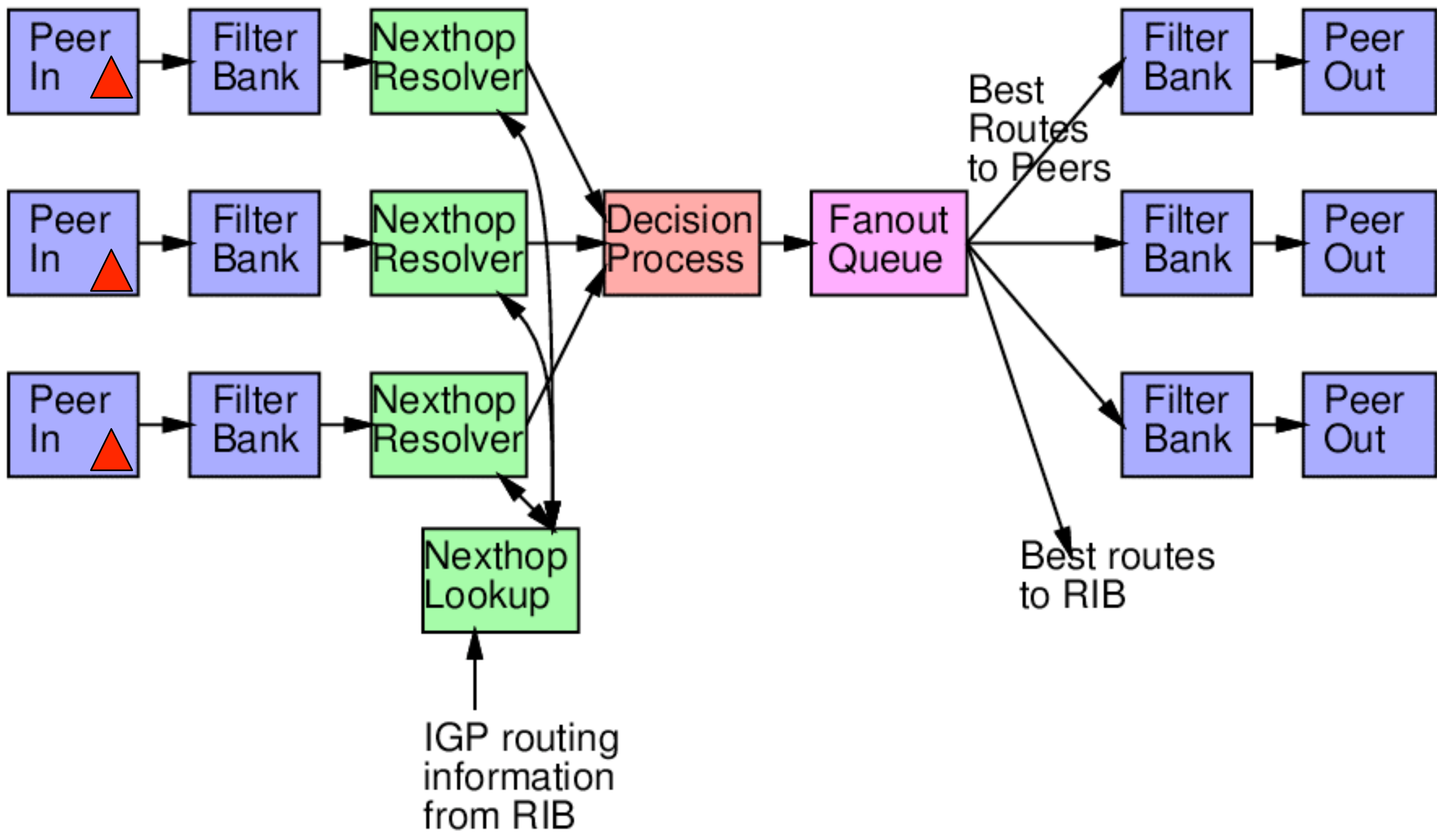
Unmodified routes stored at ingress

Changes in downstream modules (filters, nexthop state, etc) handled by PeerIn pushing the routes again.

BGP Staged Architecture

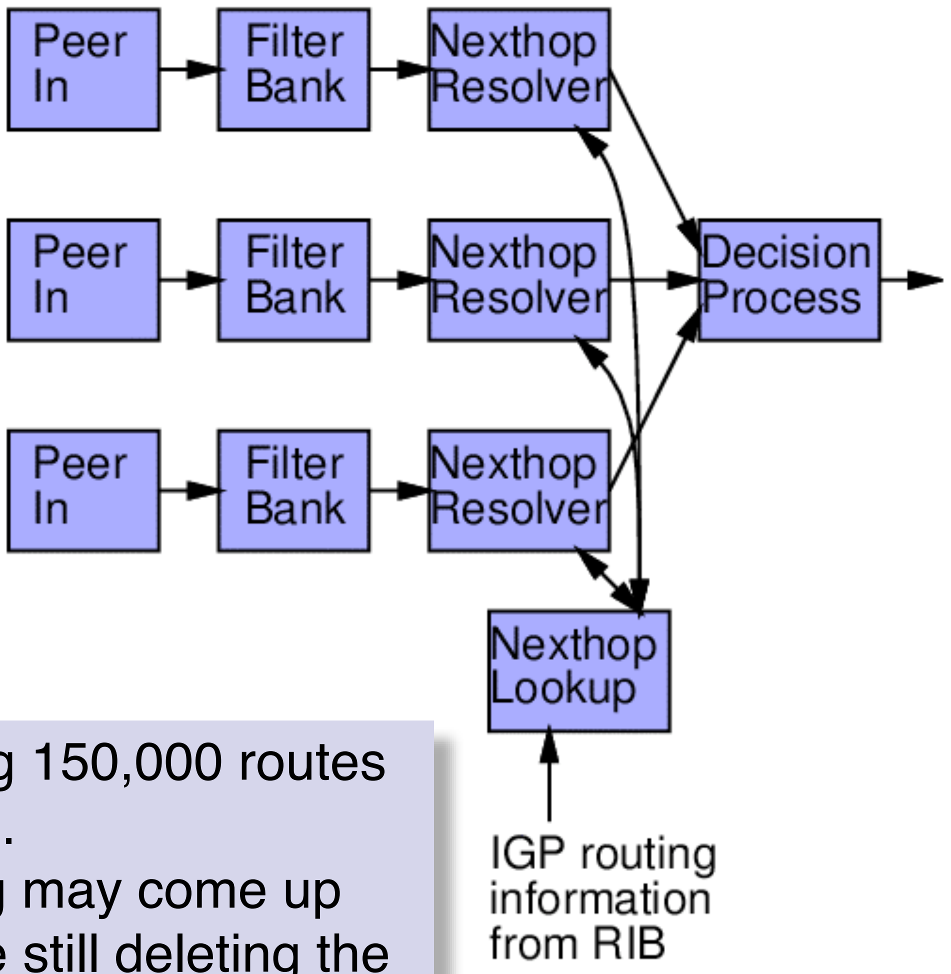


Decomposing BGP Decision



Dynamic Stages

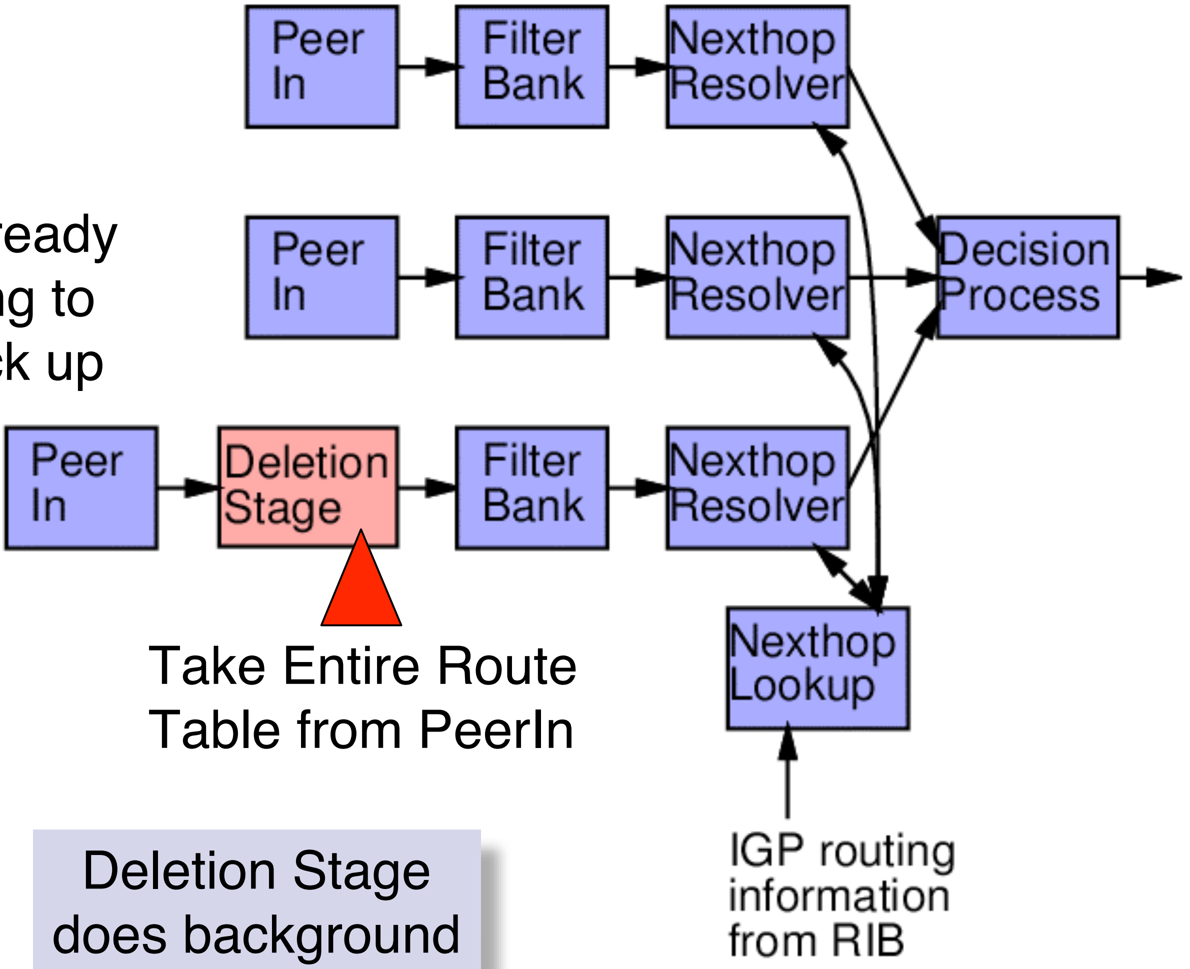
Peering
Went
Down!



Problem 1: deleting 150,000 routes takes a long time.
Problem 2: peering may come up again while we're still deleting the routes

Dynamic Stages

PeerIn is ready for peering to come back up



Take Entire Route Table from PeerIn

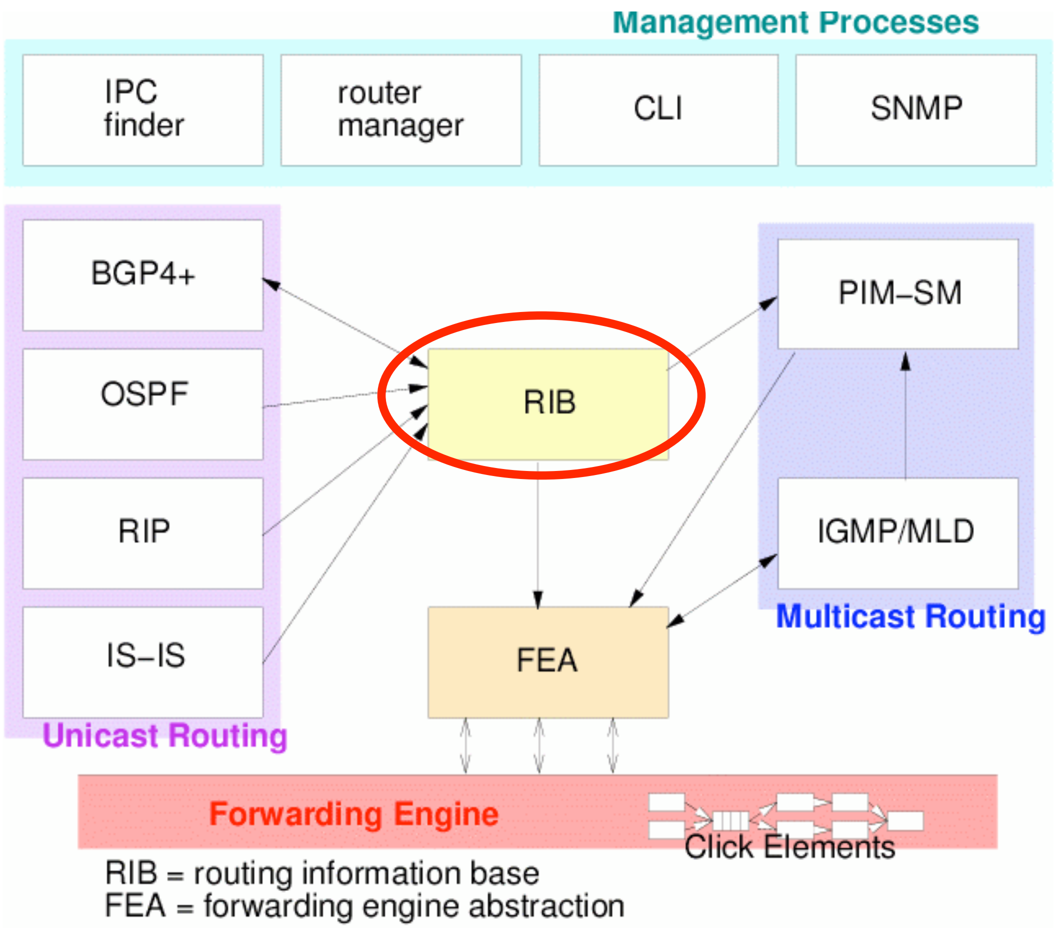
Deletion Stage does background deletion

More features, more stages…

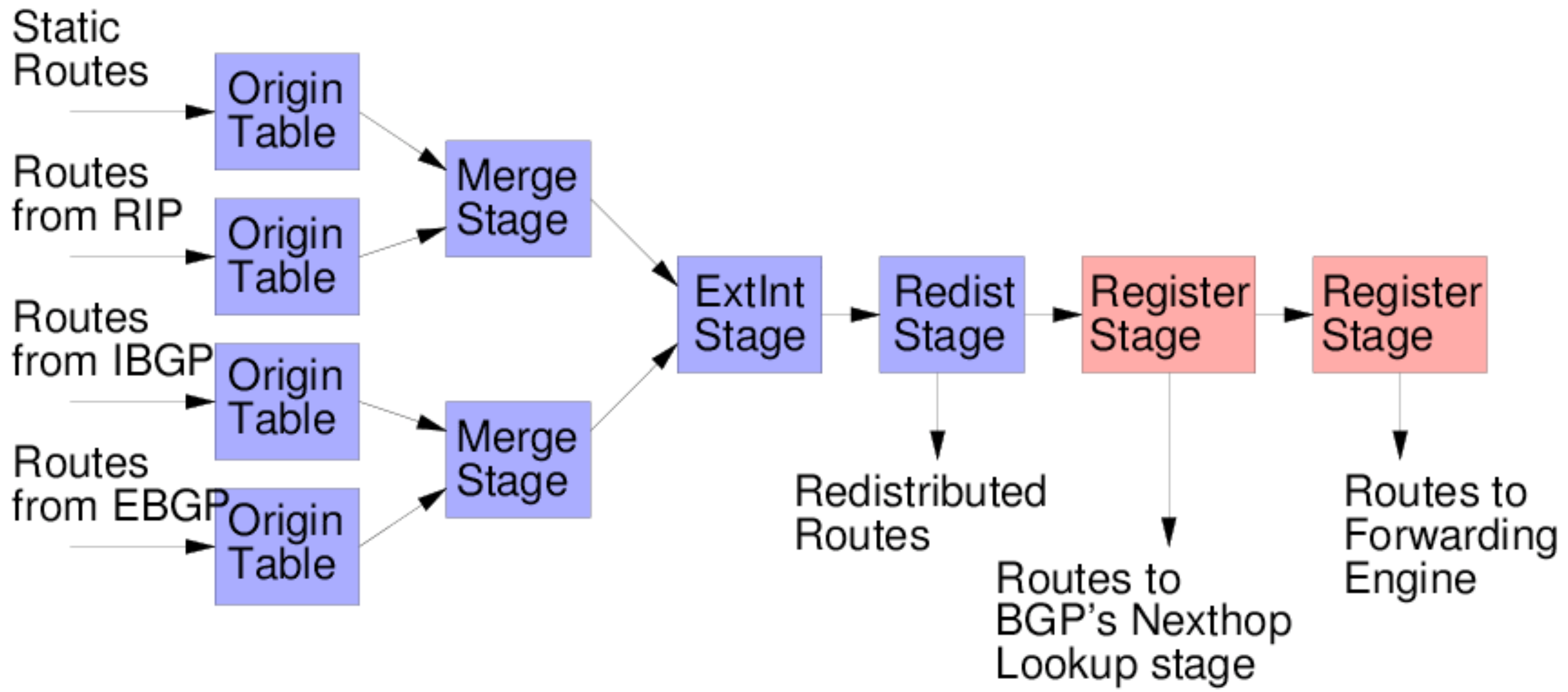
- Aggregation stage
  - Implements route aggregation.
- Policy filter stages.
  - Flexible high-performance programmable filters
- Route dump stage.
  - Dynamic stage that handles dumping existing routes to a peer that comes up.

RIB

Routing Information Base

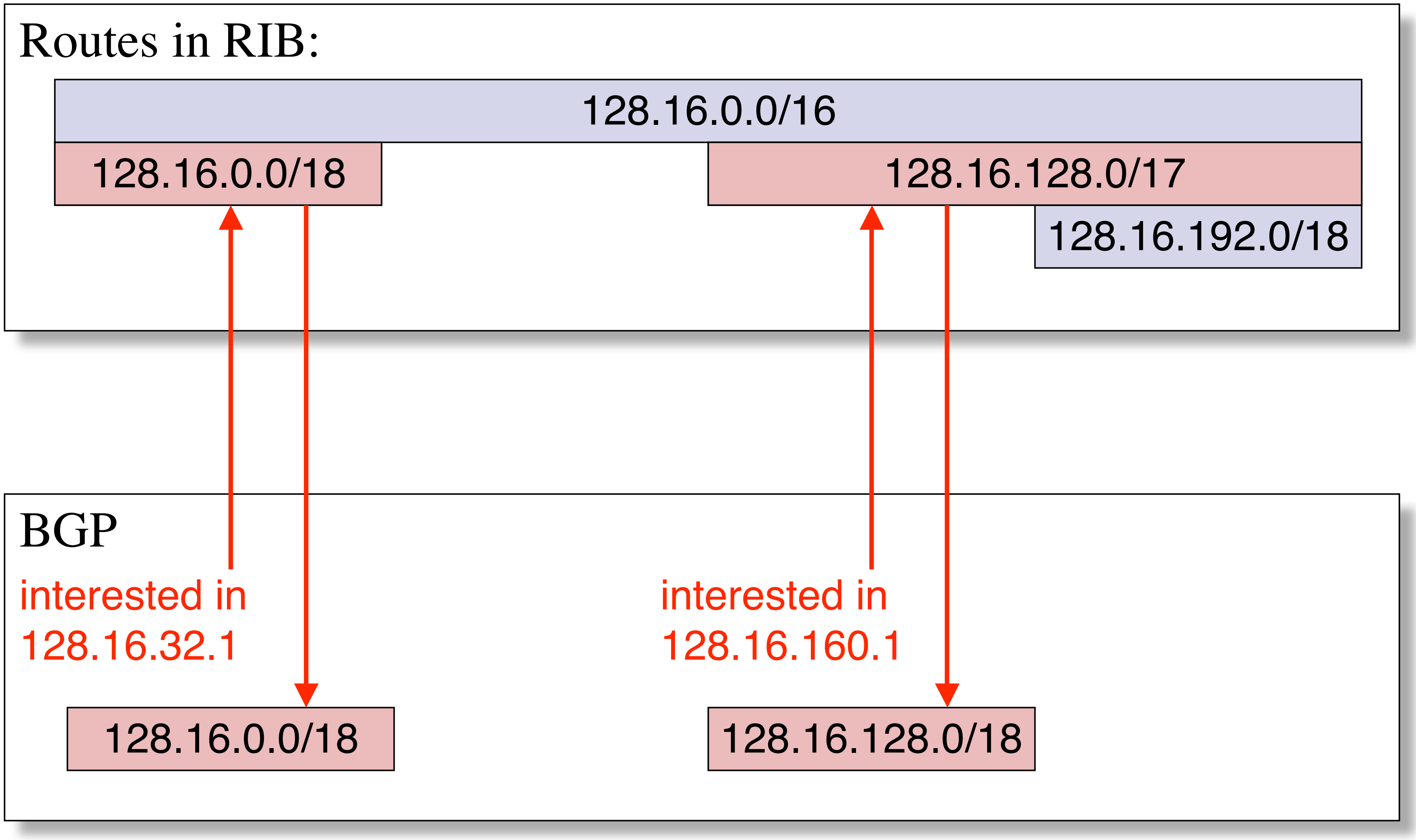


RIB Structure



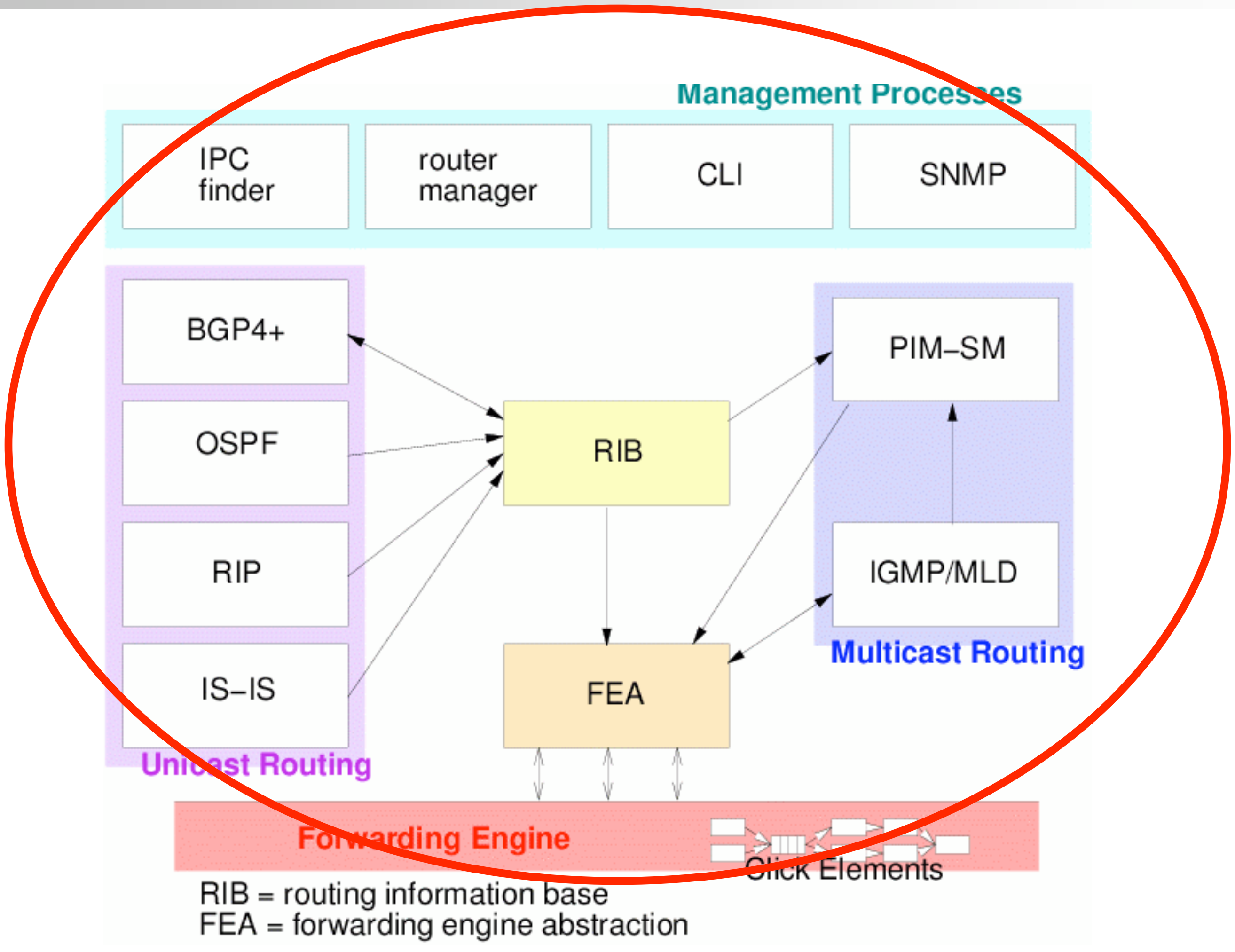
Routing protocols can register interest in tracking changes to specific routes.

Registering Interest in Routes



Libxorp

Common Code for all of XORP



RIB = routing information base
FEA = forwarding engine abstraction

Libxorp

- Libxorp contains basic data structures that can be used by XORP processes.
  - Main eventloop.
  - Timer classes.
  - Selectors.
  - Route Trees.
  - Refptrs.
  - Address classes.
  - Debugging code.
  - Logging code.

Libxorp and C++

- Libxorp gathers together a large number of useful classes for use by any new XORP process.
  - Result is that programming is “higher level” than it would be in C.
- Use of C++ templates encourages efficient code reuse.
  - Extensive use of C++ Standard Template Library.
  - C++ strings avoid security problems.
  - C++ maps give O(log(n)) lookup for many data-structures.
  - New routing-specific templates in libxorp, such as a route trie for longest-prefix match.

Libxorp and C++ Templates

Example: *Dual IPv4/IPv6 support in BGP.*

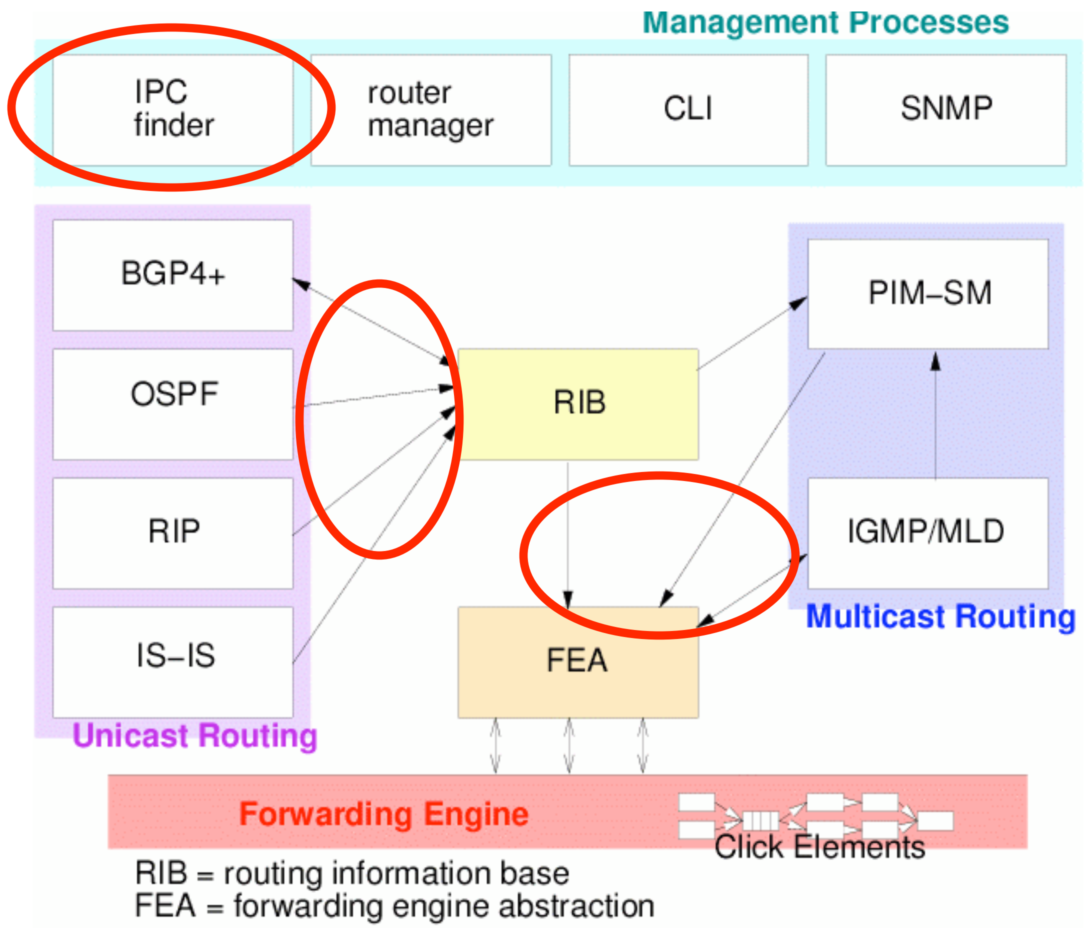
- Libxorp contains IPv4 and IPv6 address classes (and derived classes such as IPv4Net).
- All of the BGP core is templatized by address class.
- One code tree for both so they stay in sync.
- Compiler generates specialized code for IPv4 and IPv6, so it's efficient and safe.
- Only message encoding and decoding needs different code for IPv4 and IPv6 branches.

Libxorp Detail

- One detail: *safe route iterators*
- *Problem:*
  - Background task like a deletion stage needs to keep track of where it was in walking the route tree.
  - New events can cause routes to be deleted.
  - It’s very hard to program and debug such code - too much potential for race conditions.
- Solution:
  - Safe route iterators, combined with reference counted data structures, ensure that the iterator will never be left invalid.

XRLs

Interprocess communication



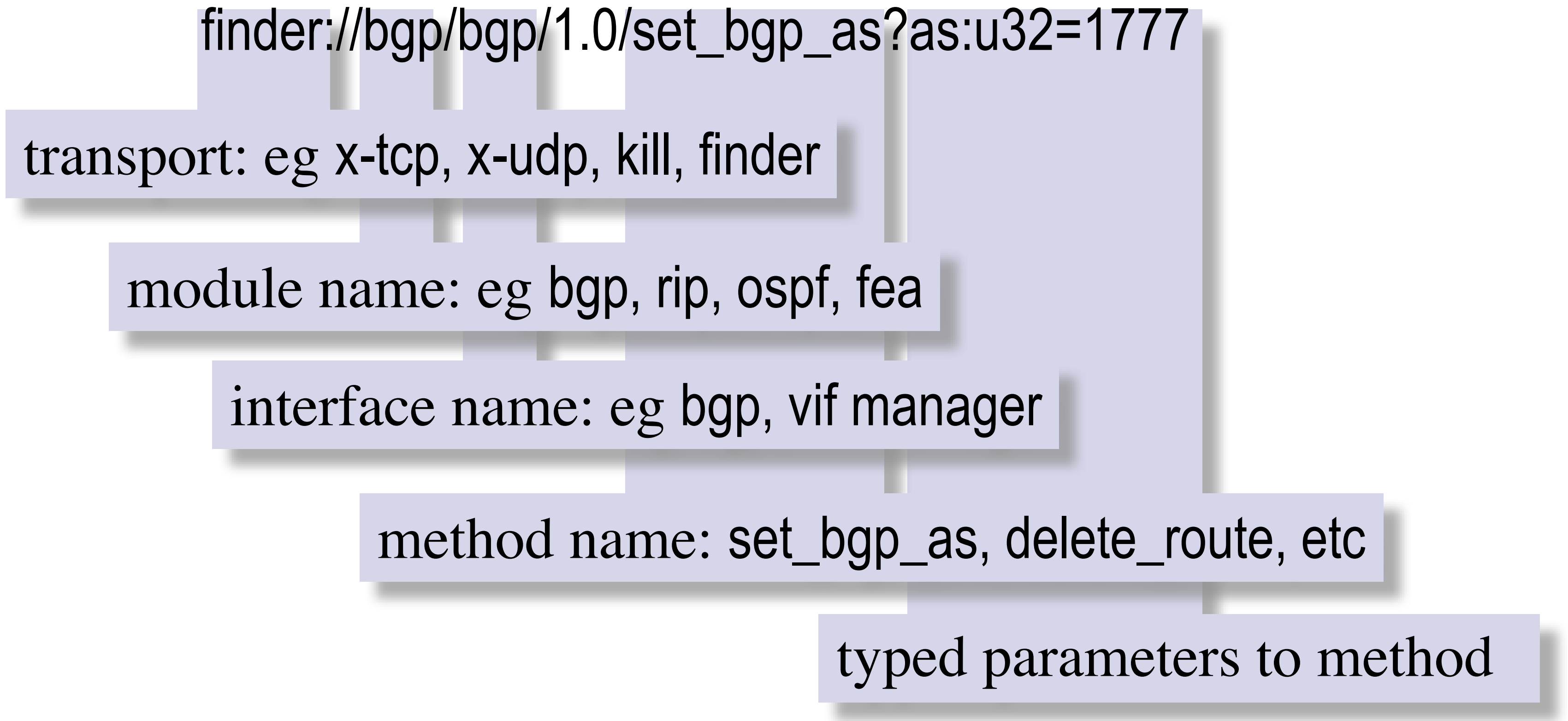
IPC Framework

- Want to enable integration of future protocols from third party vendors, without having to change existing core XORP code.
- Want to be able to build distributed routers
  - More than one control engine.
  - Robustness, performance.
- Want to aid testing and debugging.
- Every API should be a hook for extensions.
- Minimize a-priori knowledge of who performs which function.
  - Allow refactoring.
  - Allow tuning of function-to-process binding under different deployment scenarios.

Inter-process Communication

XORP Resource Locators (XRLs):

- URL-like unified structure for inter-process communication:
- Example:


finder://bgp/bgp/1.0/set_bgp_as?as:u32=1777
transport: eg x-tcp, x-udp, kill, finder
module name: eg bgp, rip, ospf, fea
interface name: eg bgp, vif manager
method name: set_bgp_as, delete_route, etc
typed parameters to method

Inter-process Communication

XORP Resource Locators (XRLs):

- URL-like unified structure for inter-process communication:
- Example:

  ```
  finder://bgp/bgp/1.0/set_bgp_as?as:u32=1777
  ```

- Finder resolves to a concrete method instance, instantiates transport, and performs access control.

  ```
  xtcp://192.1.2.3:8765/bgp/1.0/set_bgp_as?as:u32=1777
  ```

Inter-process Communication

- XRLs support extensibility by allowing "non-native" mechanisms to be accessed by unmodified XORP processes.
  - Add new XRL protocol families: eg kill, SNMP
- ASCII canonical representation means XRL can be scripted from python, perl, bash, etc.
  - XORP test harnesses built this way.
- ASCII representation enables design of an extensible router management framework via configuration template files.
- Efficient binary representation normally used internally.
  - Stub compiler eases programmer's job.

Calling an XRL (1)

Step 1: Each process registers its XRL interfaces and methods with the finder.

- Generic names used.
- Random key added to method names.

Step 2: When a process wants to call an XRL, it uses the generic name of an interface/method.

- XRL library in process requests resolution of XRL by finder.
- Finder checks if this process is allowed to access this method.
- Finder resolves the method to the current specific instance name, including the random key.
- Finder chooses the appropriate transport protocol depending on instance location registered capabilities of target.

Calling an XRL (2)

Step 3: Process sends the request to the target.

- Target checks random key.
- Processes request.
- Sends response.

Step 4: Process's IPC library caches resolved XRL.

- Future calls go direct, bypassing the finder.

- Transport reliability is provided by XRL library.
- If a call fails, the cache is cleared, the application is notified, and processes should follow the XORP Error Handling conventions.

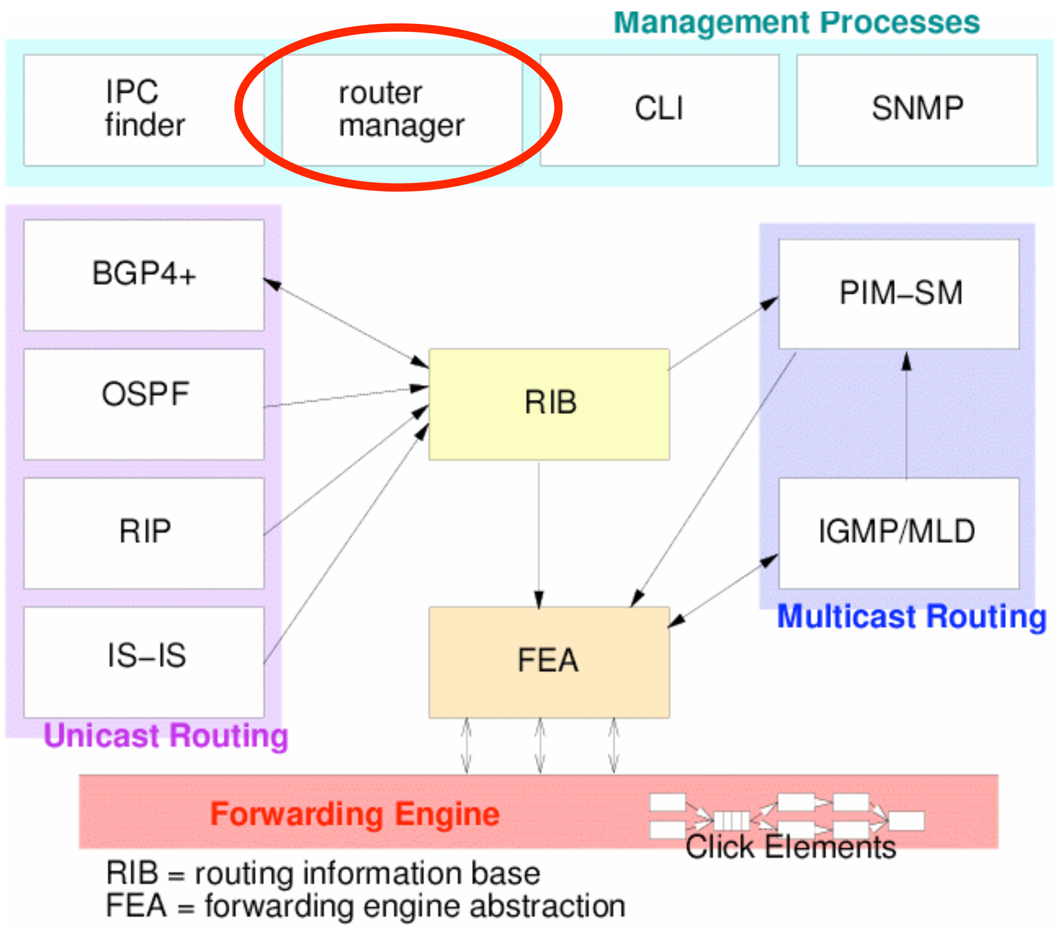
XRL Security and Process Sandboxing

- Random key in method name ensures a process cannot call a method on another process unless the finder has authorized it.
- Finder is central location for configuring security for experimental (untrusted processes).
  - XRL sandbox capability under development.
  - Experimenting with using XEN virtualization to run untrusted code.
  - Fine-grain per-domain ACLs to control precisely what XRLs may be called and what parameters may be supplied to them.
  - Sending/receiving from net also via XRLs.

Process Birth and Death Events

- A process can register with the finder to discover when other processes start or terminate.
- The finder continuously monitors liveness of all registered processes.
  - Keepalivcs every few seconds.
  - Keepalive failure indicates process failure (either death or lockup).
- Processes that have registered interest are notified of failure.
  - The action to take depends on what failed.
  - Rtrmgr should kill and restarted failed processes.
  - Other processes cleanup orphaned state, or restart themselves, as appropriate.

rtrmgr
Router Manager Process



Extensible Router Manager Framework

- How do you implement a single unified router management framework and command line interface, when you don’t know what protocols are going to be managed?

XORP Router Manager Process

- *The XORP router manager is dynamically extensible using declarative ASCII template files linking configuration state to the XRLs needed to instantiate that configuration.*

Router Manager template files

Map Juniper-style configuration state to XRLs

Configuration File

```
protocols ospf {
  router-id: 128.16.64.1
  area 128.16.0.1 {
    interface xl0 {
      hello-interval: 30
} } }
```

Template File

```
protocols.ospf {
  area @: ipv4 {
    interface @: txt {
      hello-interval: u32 {
        %set: xrl "ospfd/set_interface_param ? area_id:ipv4=$(area.@)
                  & interface:txt=$(interface.@)
                  & ospf_if_hello_interval:u32=$(@)";
} } } }
```

Template Files

Template files provide a *configuration template:*

- What can be configured?
- Which process to start to provide the functionality?
  - What the process depends on.
  - BGP depends on RIB depends on FEA.
  - This determines startup ordering.
- Configuration constraints:
  - Which attributes are mandatory?
  - What ranges of values are permitted?
  - What syntax is permitted for each value?
- How to configure each attribute.
  - Which XRL to call, or process to run.

Template Files

- Each XORP process has its own template file.
  - The entire router template tree is formed at runtime from the union of the templates for each available process.
- Rtrmgr needs *no inbuilt knowledge* about the processes being configured.
  - Add a new option to BGP: just add it to the template file and rtrmgr can configure it.
  - Add a new routing process binary to the system: add an additional template file and rtrmgr can configure it.
- Currently, templates are read at rtrmgr startup time.
  - Plan is to allow templates to be re-read at runtime, to allow *on-the-fly upgrading* of running processes.

xorpsh
(xorp command line interface)

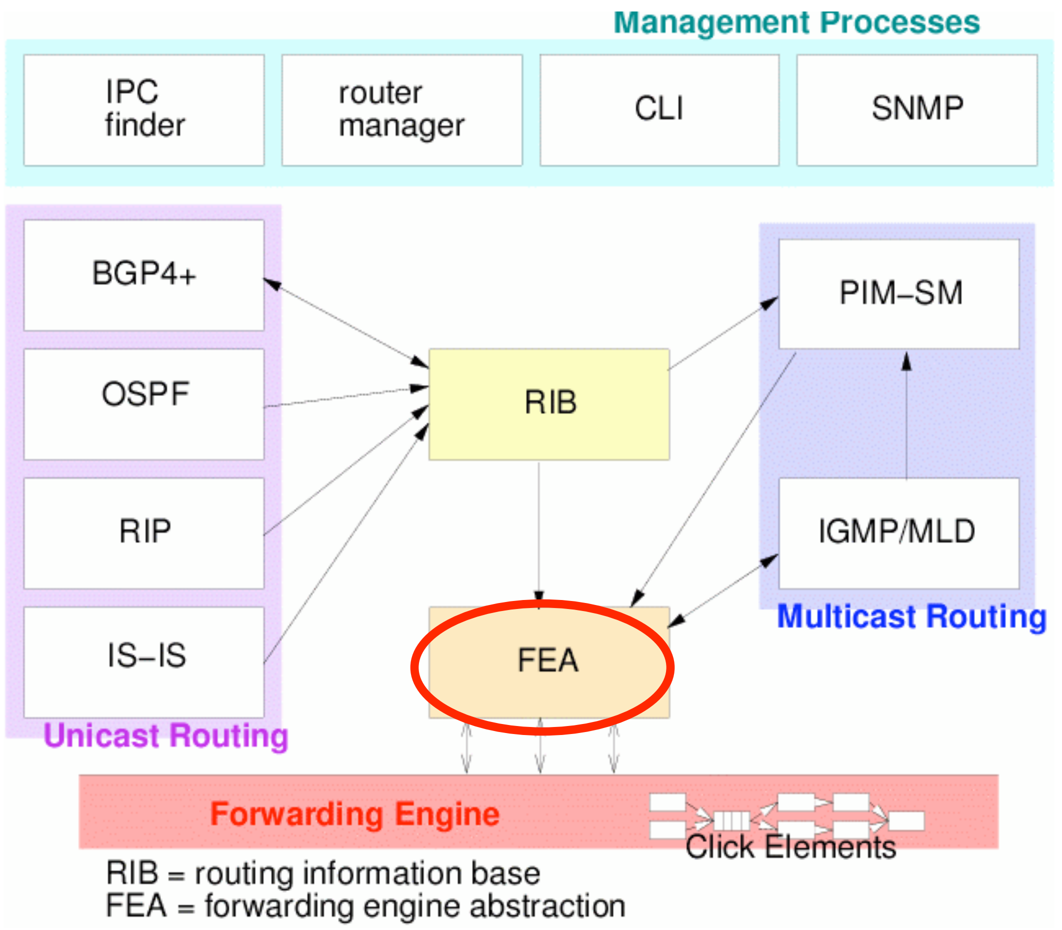
- Multiple human operators can be interacting with a router at the same time.
  - Some can be privileged, some not.
- An instance of *xorpsh* is run for each login.
  - Authenticates with rtrmgr.
  - Downloads the current router config.
    - Receives dynamic updates as the config changes.
  - Provides the CLI to the user, allowing him to configure all the functionality from the template files.
    - Full command line completion.
    - No changes made until “commit”.

xorpsh

(xorp command line interface)

- To commit changes, xorpsh sends config changes to the rtrmgr.
  - rtrmgr must run as root to start processes.
  - xorpsh does not run as root.
  - rtrmgr enforces any restrictions for that user.
- To perform operational mode commands, xorpsh reads a second set of template files.
  - Eg "**`show route table bgp`**"
  - Xorpsh runs the relevant monitoring tool, which communicates directly with the target process.
  - Minimizes the amount of code that must run as root, and avoids loading the rtrmgr with monitoring tasks.

FEA
Forwarding Engine Abstraction

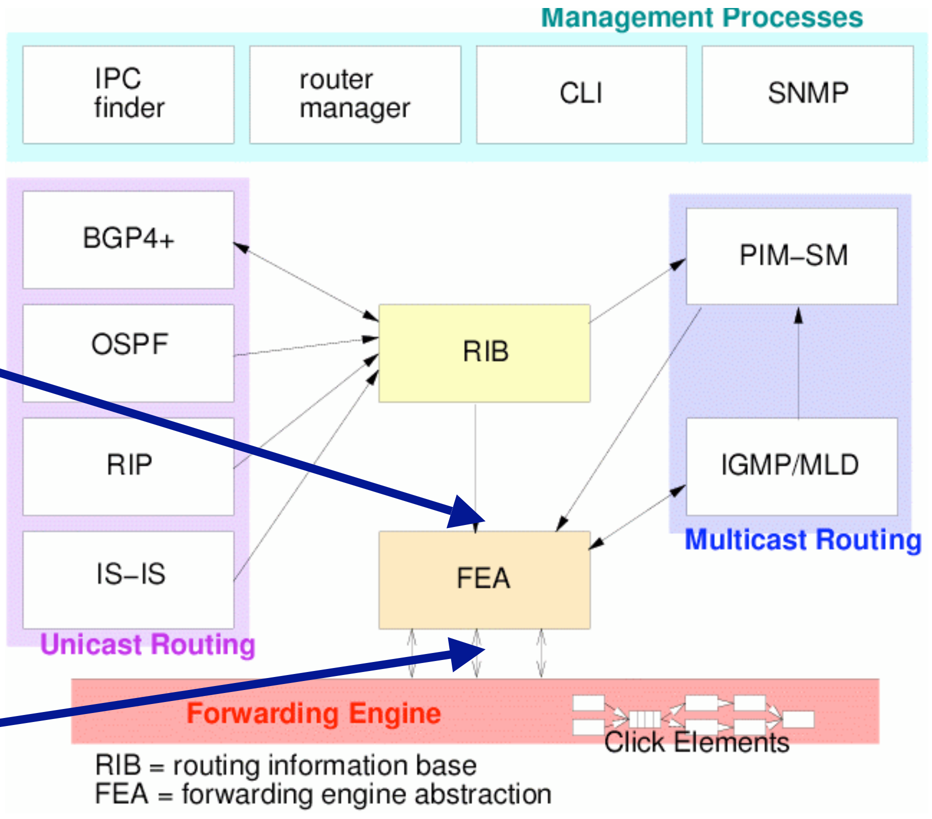


FEA
Forwarding Engine Abstraction

- Main purpose of FEA is to provide a stable API to the forwarding engine.

Same XRL interface on
All forwarding engines

Different OS calls.
Different kernel functionality.
Different hardware capabilities.
Multiple forwarding engines



FEA Functionality:
Interfaces

- Discover and configure network interfaces.
  - Physical vs virtual.
- Provides a way for processes to register interest in interface state changes and config changes.
  - Eg. interface goes down, OSPF needs to know immediately to trigger an LSA update.
- Soon: provide a standard way to create virtual interfaces on a physical interface.
  - Eg: VLANs, ATM VCs.

FEA Functionality:
Routes

Unicast Routing:

- Sends routes to the forwarding engine.
- Reporting of errors.

Multicast Routing:

- Sets/removes multicast forwarding state.
- Relays notifications:
  - IGMP join/leave messages.
  - PIM messages.
  - Notifications of packet received on OIF (needed for PIM asserts and related data-drived functionality)

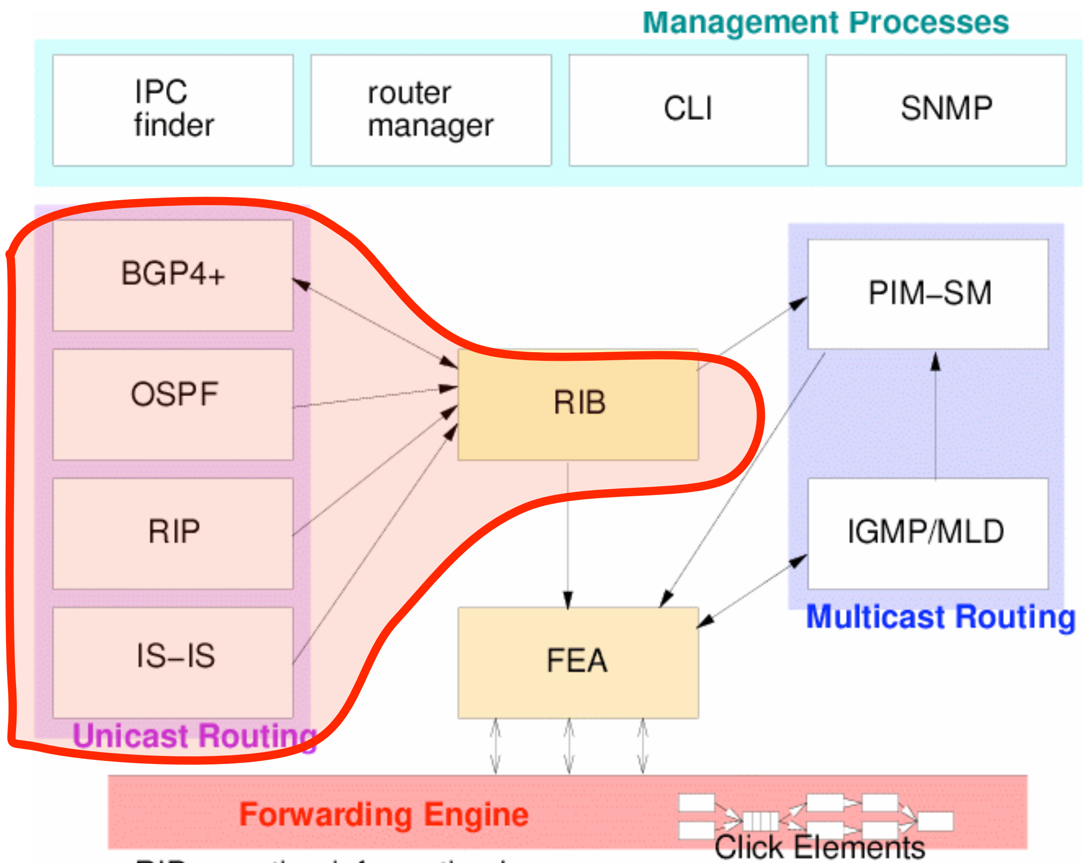
FEA Functionality:
Routing Traffic Relay

Different systems have different conventions for sending raw packets, etc.

XORP relays routing messages through the FEA so that routing processes can be FE-agnostic.

- Relaying has *security advantages*.
  - Routing protocols don’t run as root.
  - XRL sandboxing will limit what a bad process can send and receive.
- Relaying enables *distributed routers*.
  - Routing process does not care what box it runs on.
  - May be able to migrate a routing process, or fail over to a standby route processor.
- Relaying enables *process restart*.
  - Socket can be kept open.
  - On-the-fly software upgrade?

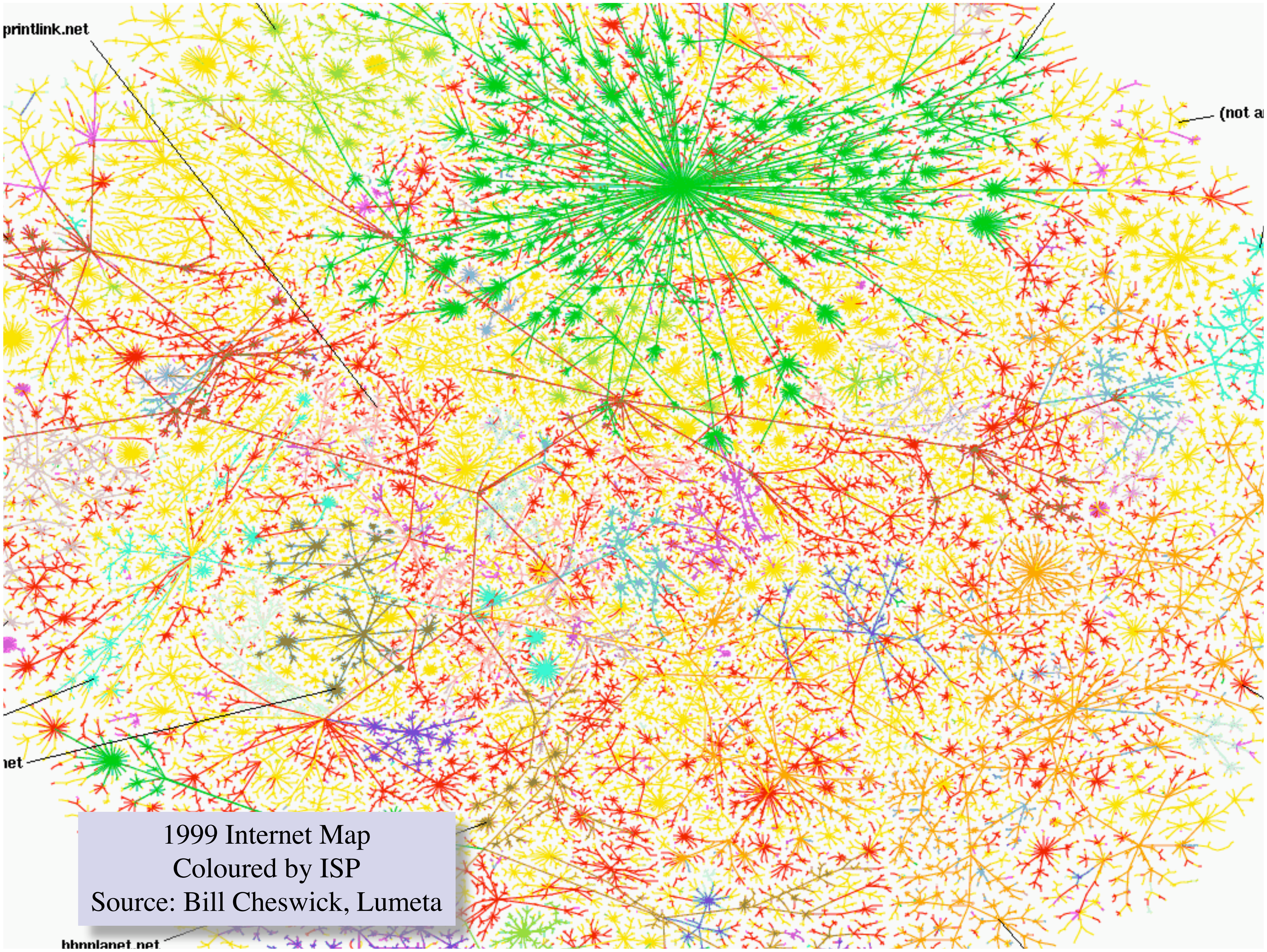
Routing Policy



RIB = routing information base
FEA = forwarding engine abstraction

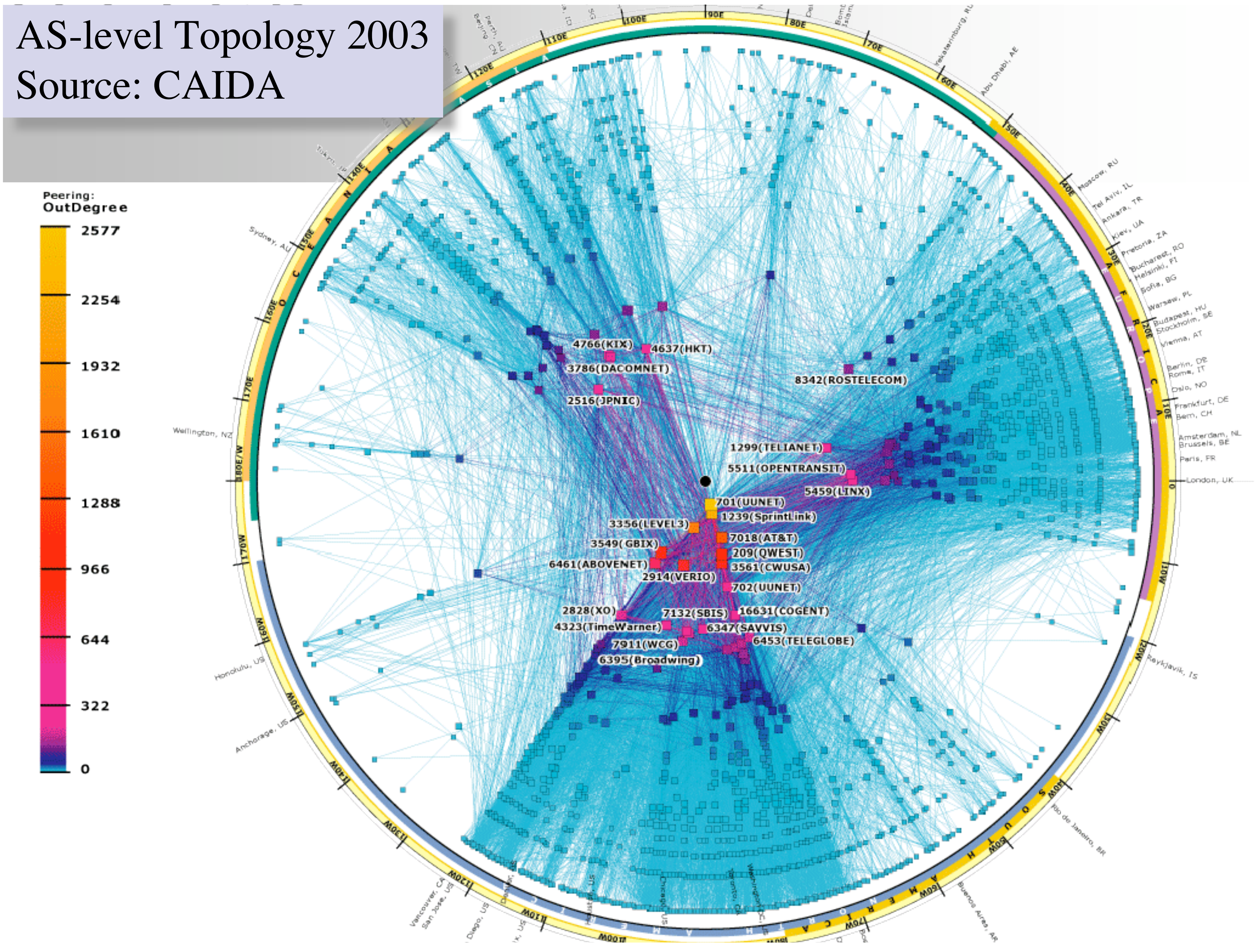
Routing Policy

- How do you implement a routing policy framework in an extensible unified manner, when you don’t know what future routing protocols will look like?

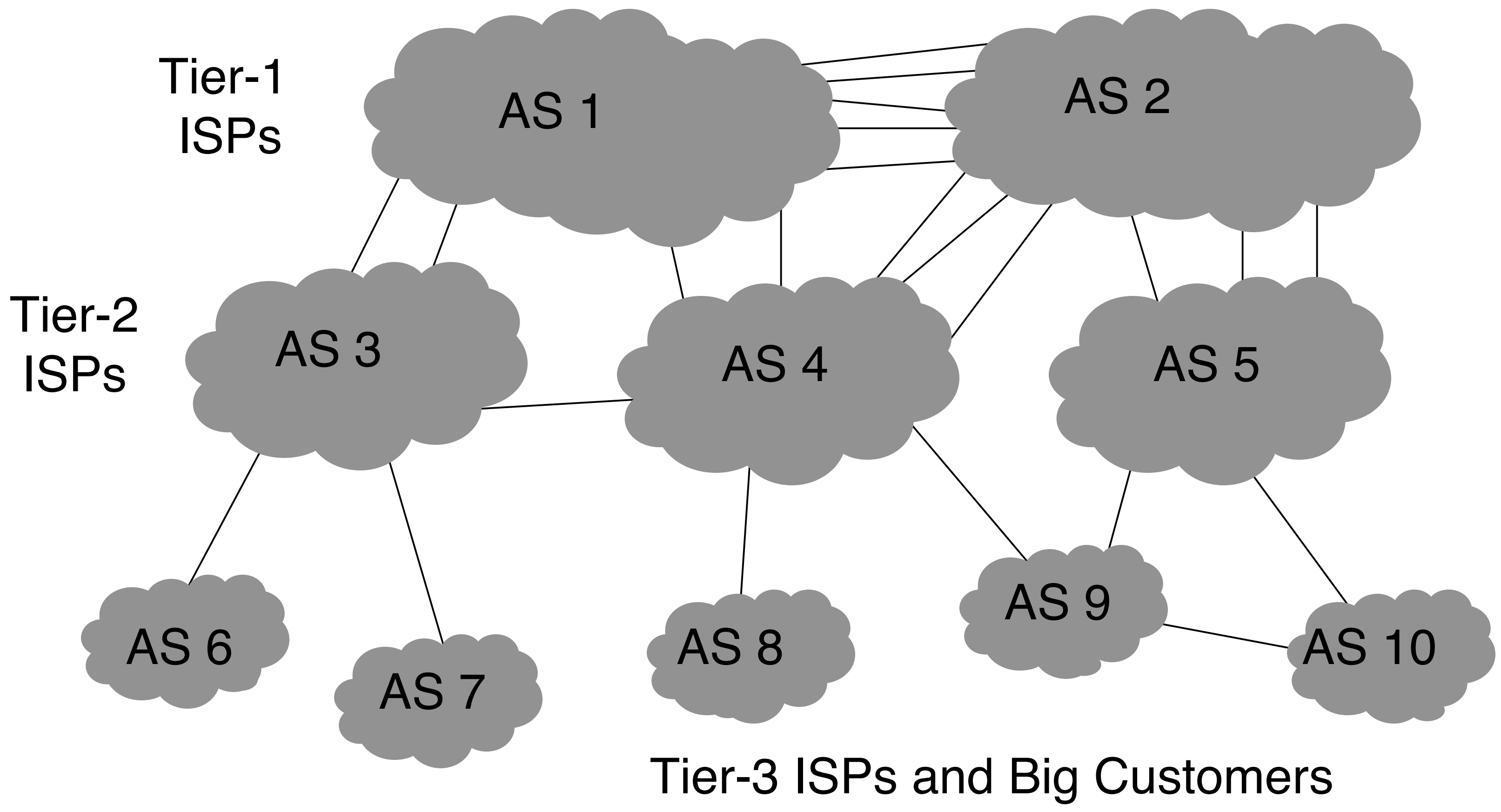


1999 Internet Map
Coloured by ISP
Source: Bill Cheswick, Lumeta

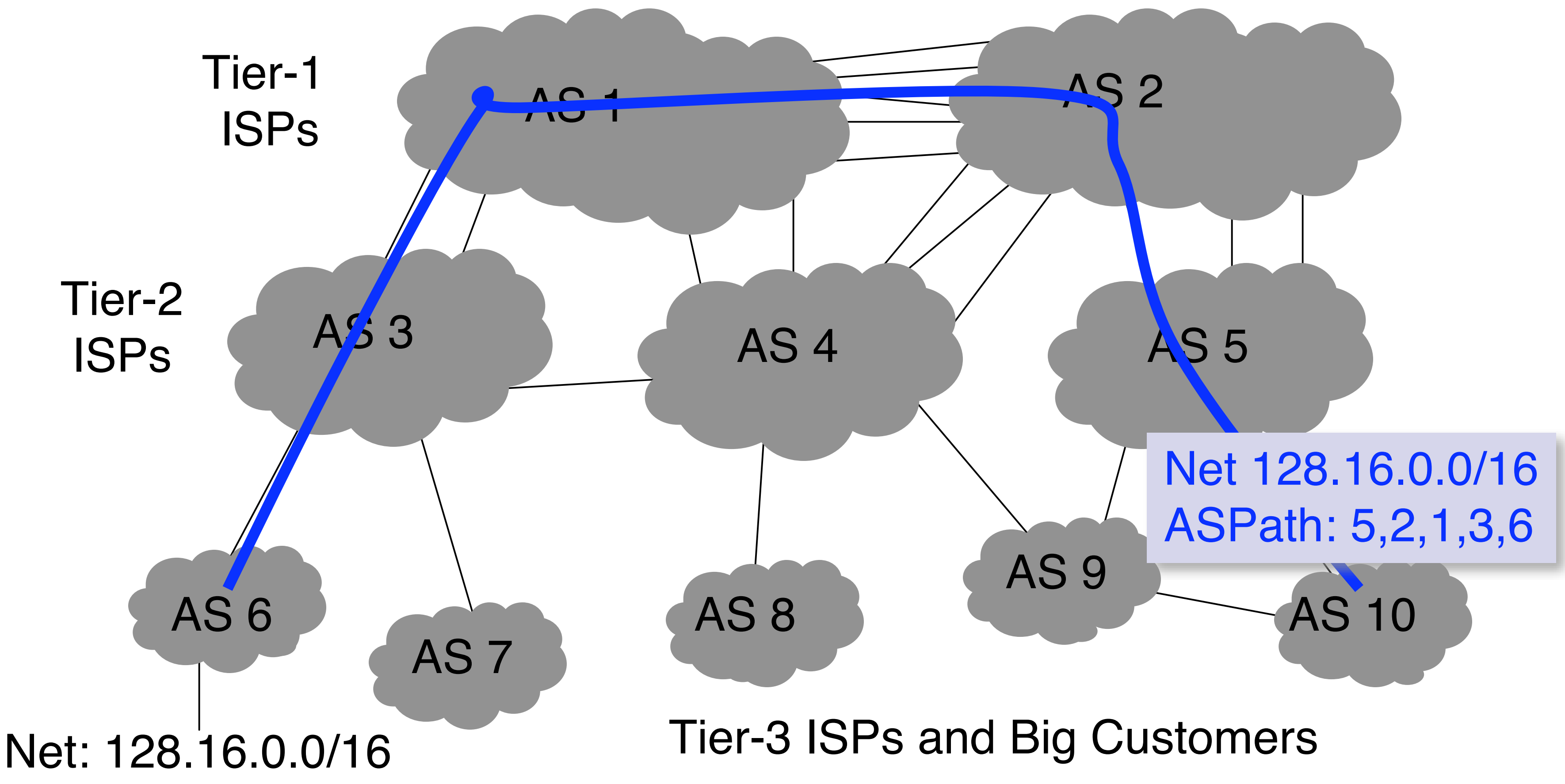
AS-level Topology 2003
Source: CAIDA



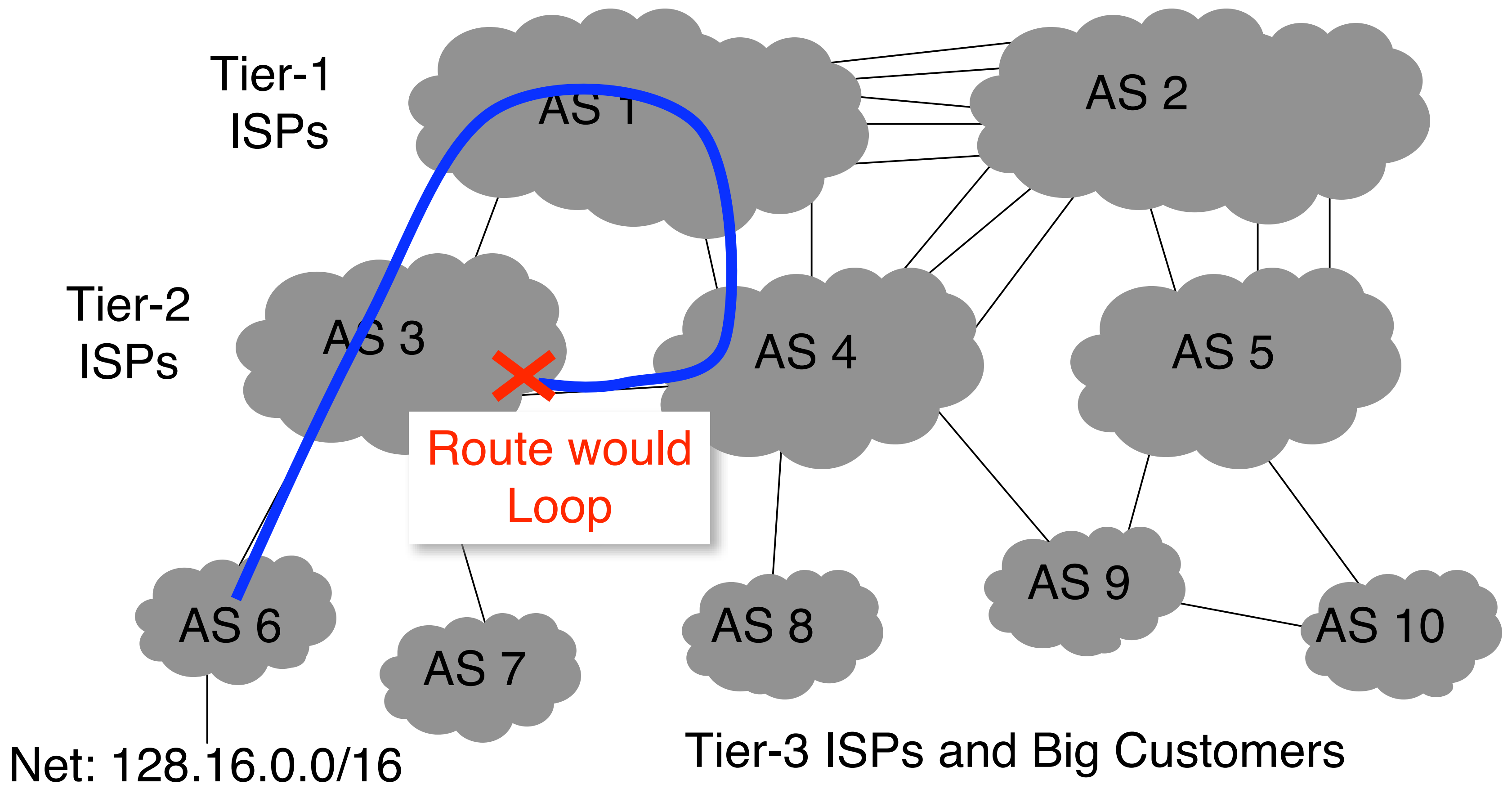
Inter-domain Routing



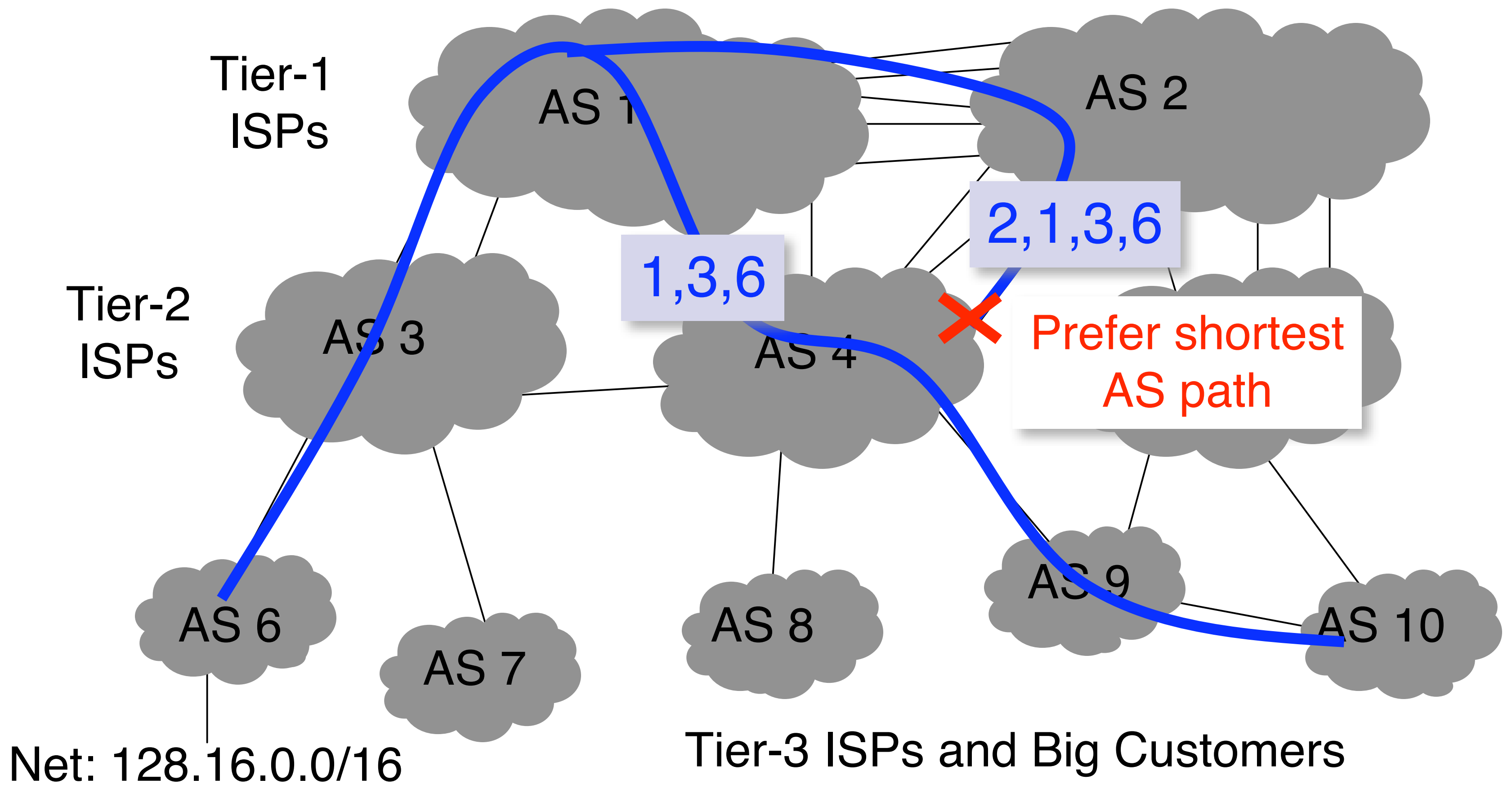
Inter-domain Routing



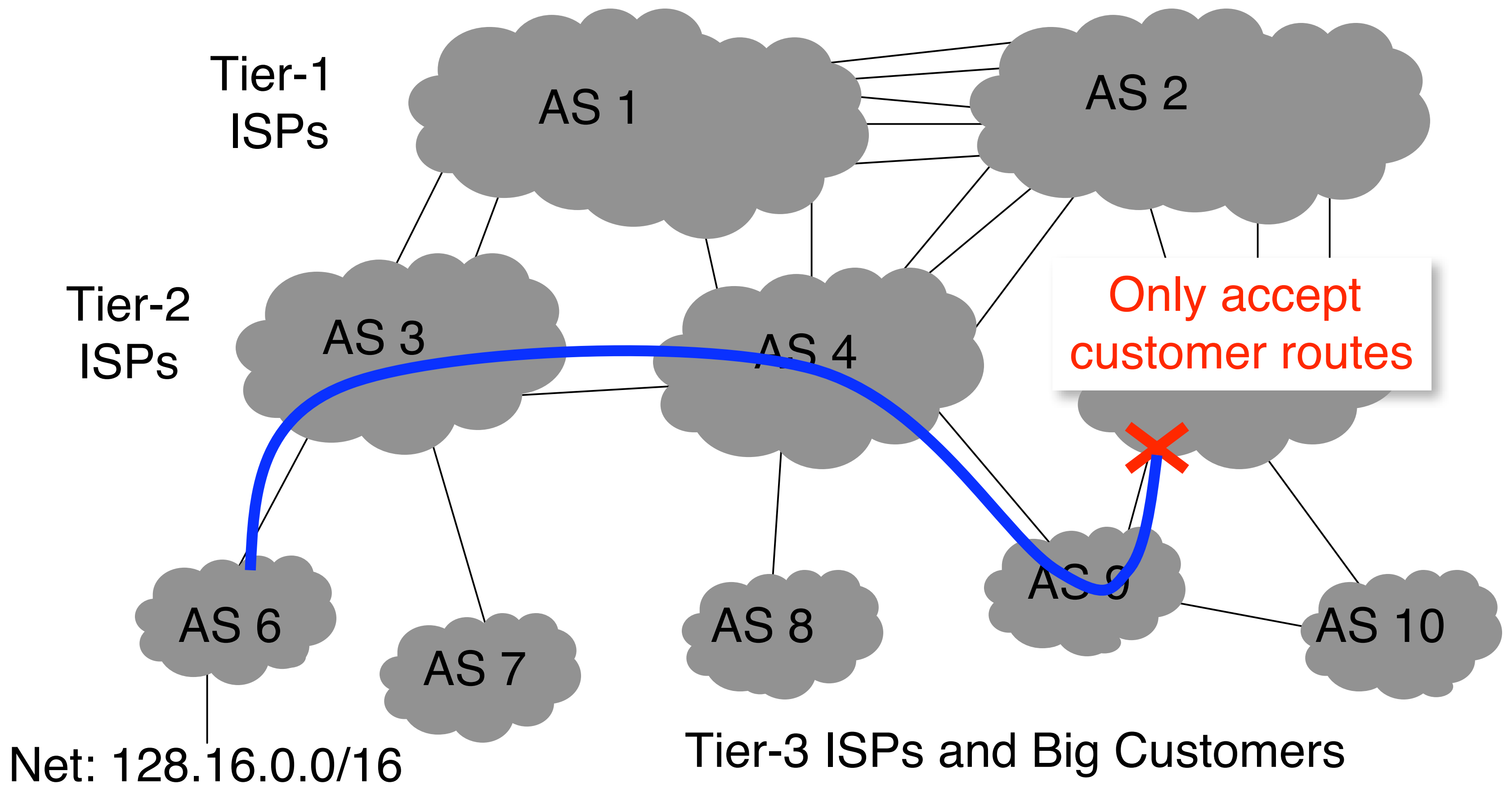
Inter-domain Routing



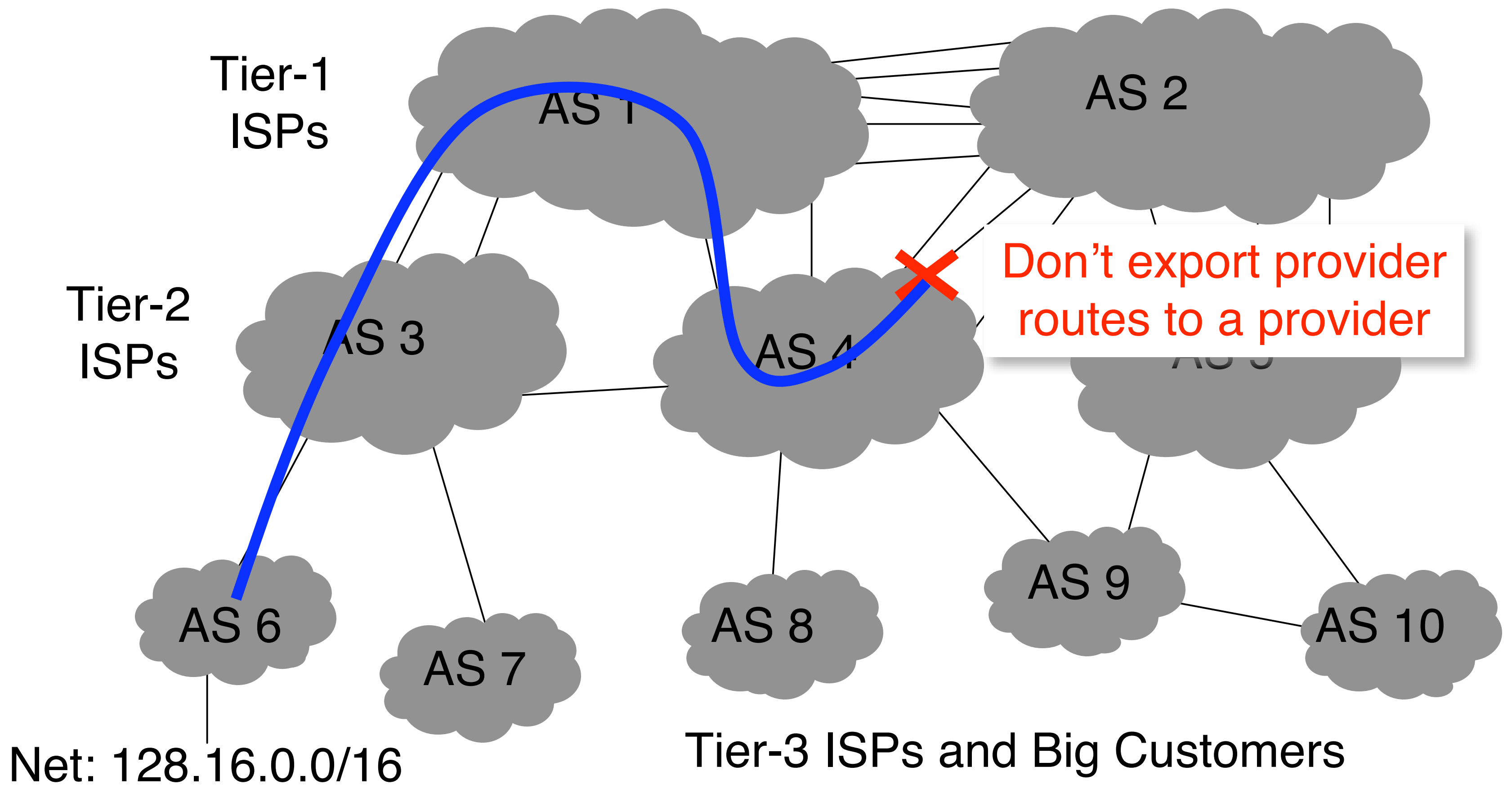
Inter-domain Routing



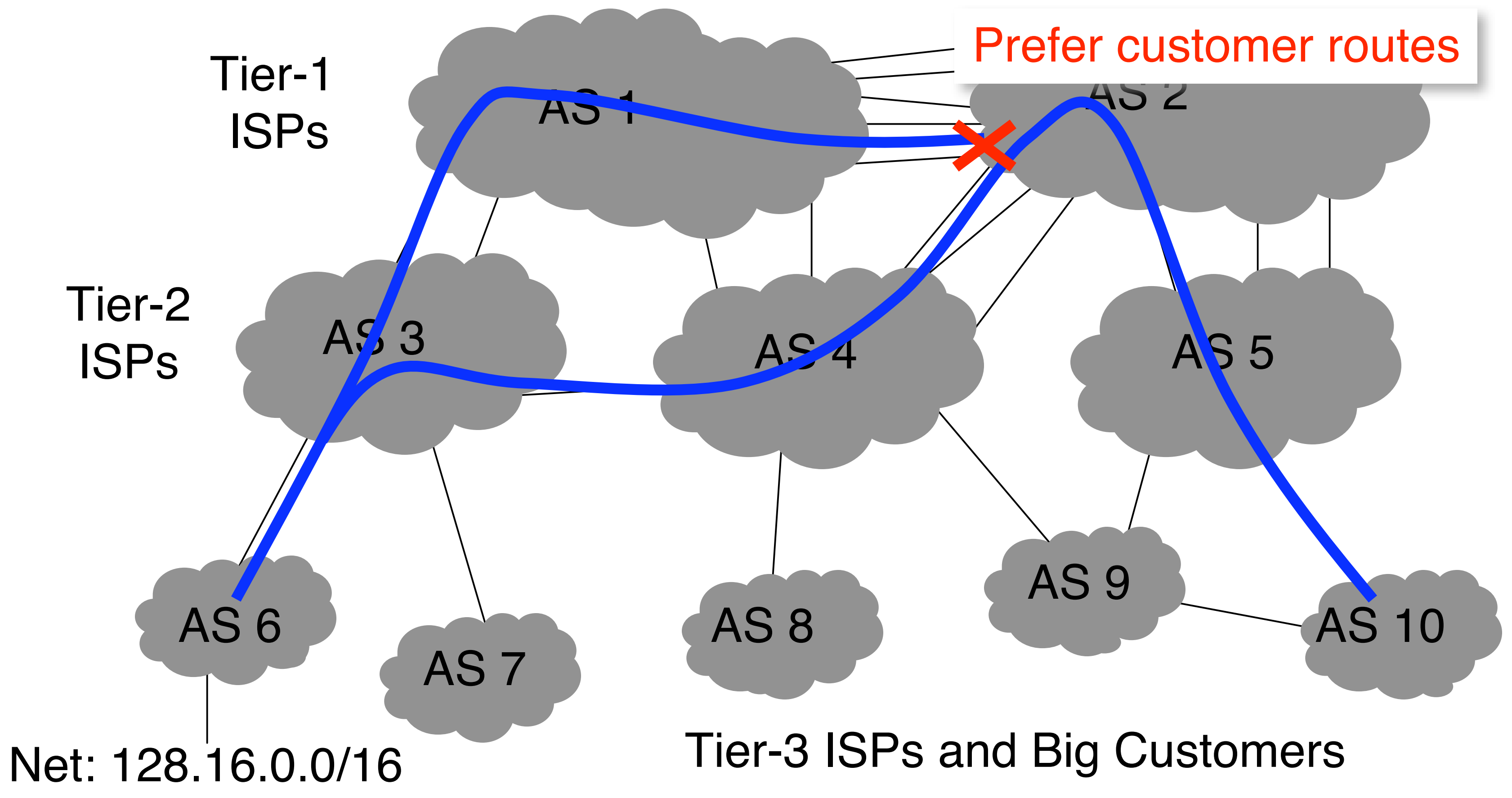
Inter-domain Routing Policy



Inter-domain Routing Policy



Inter-domain Routing Policy



Examples of Policy Filters

Import filters:

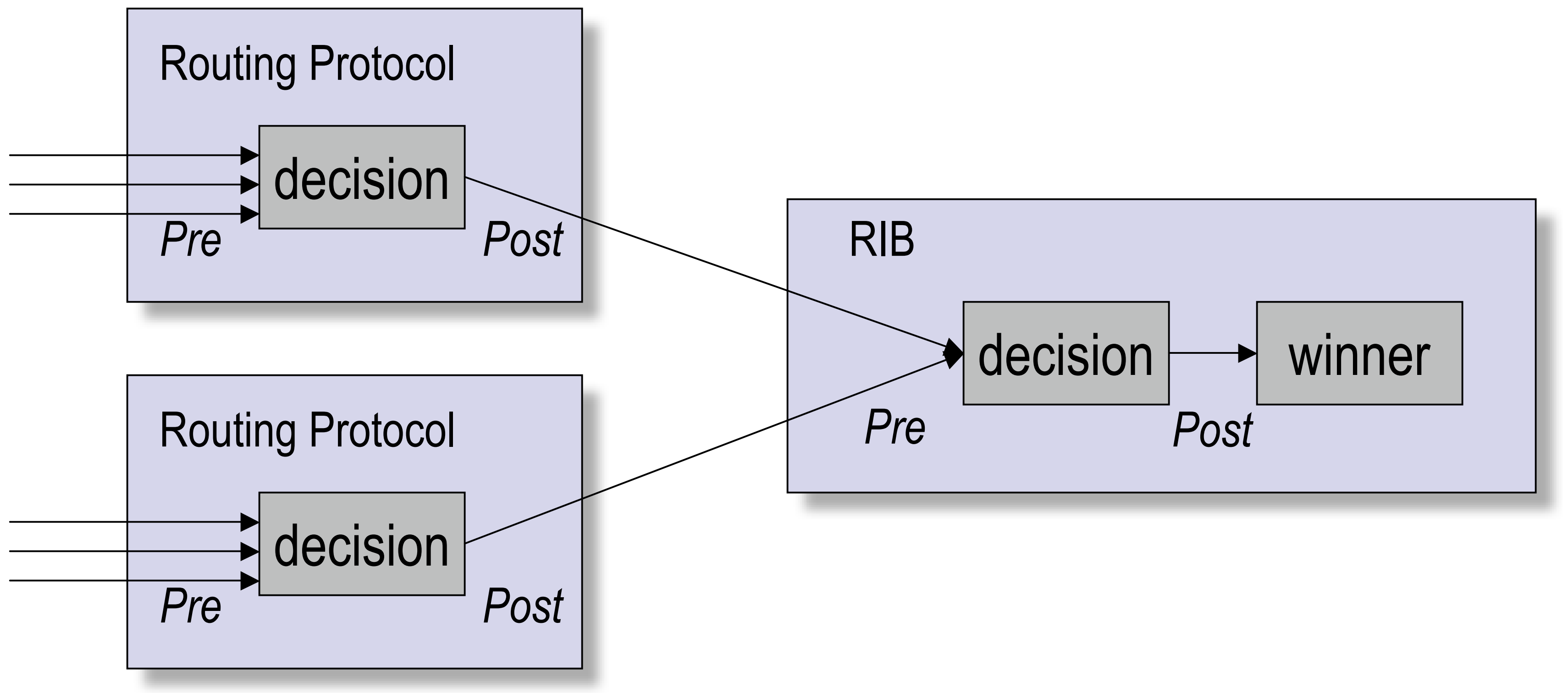
- Drop incoming BGP routes whose AS Path contains AS 1234.
- Set a LOCALPREF of 3 on incoming BGP routes that have a nexthop of 128.16.64.1

Export filters:

- Don't export routes with BGP community *xyz* to BGP peer 128.16.64.1
- Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.

Where to apply filters?

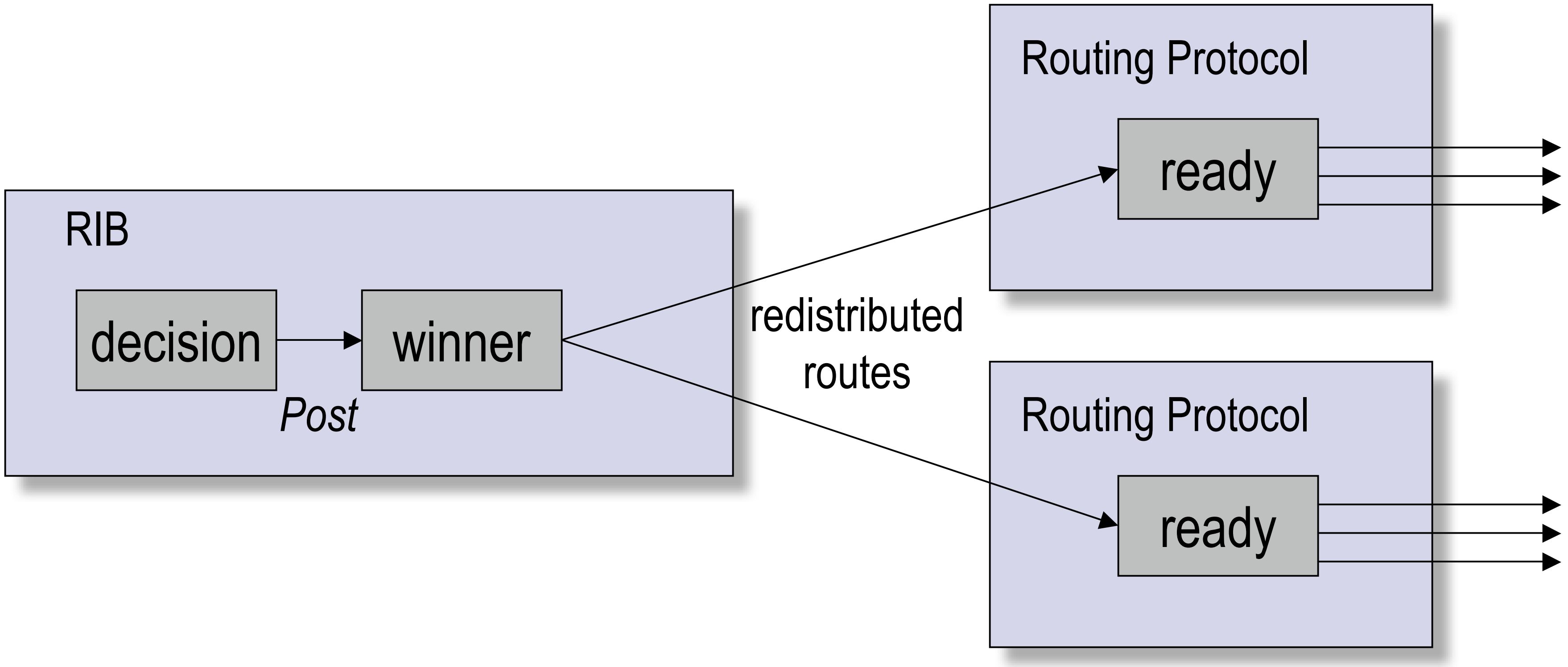
Flow of incoming routes:



Originated ⟶ Accepted ⟶ Winner

Where to apply filters?

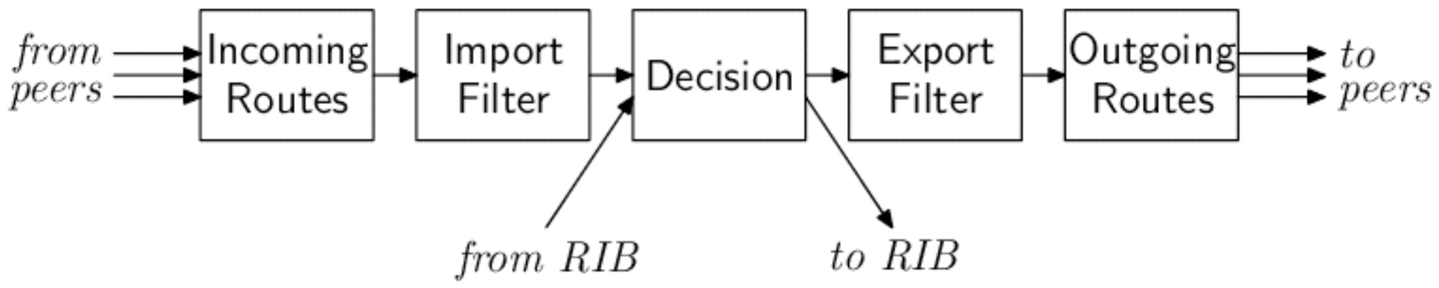
Flow of outgoing routes:



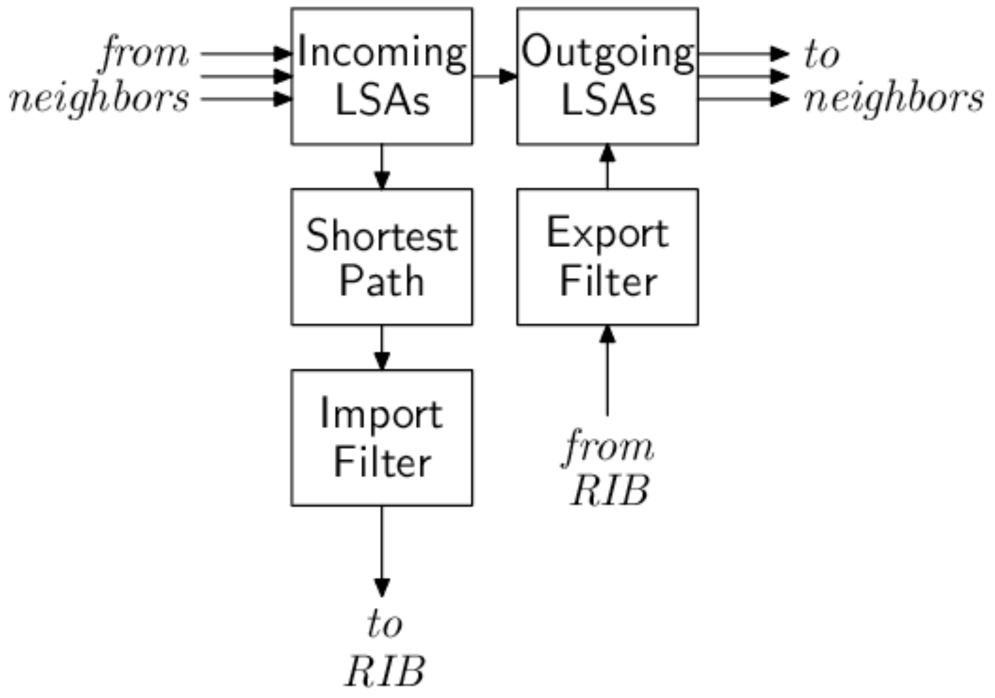
$$\text{winner} \longrightarrow \text{ready}$$

Where to apply filters?

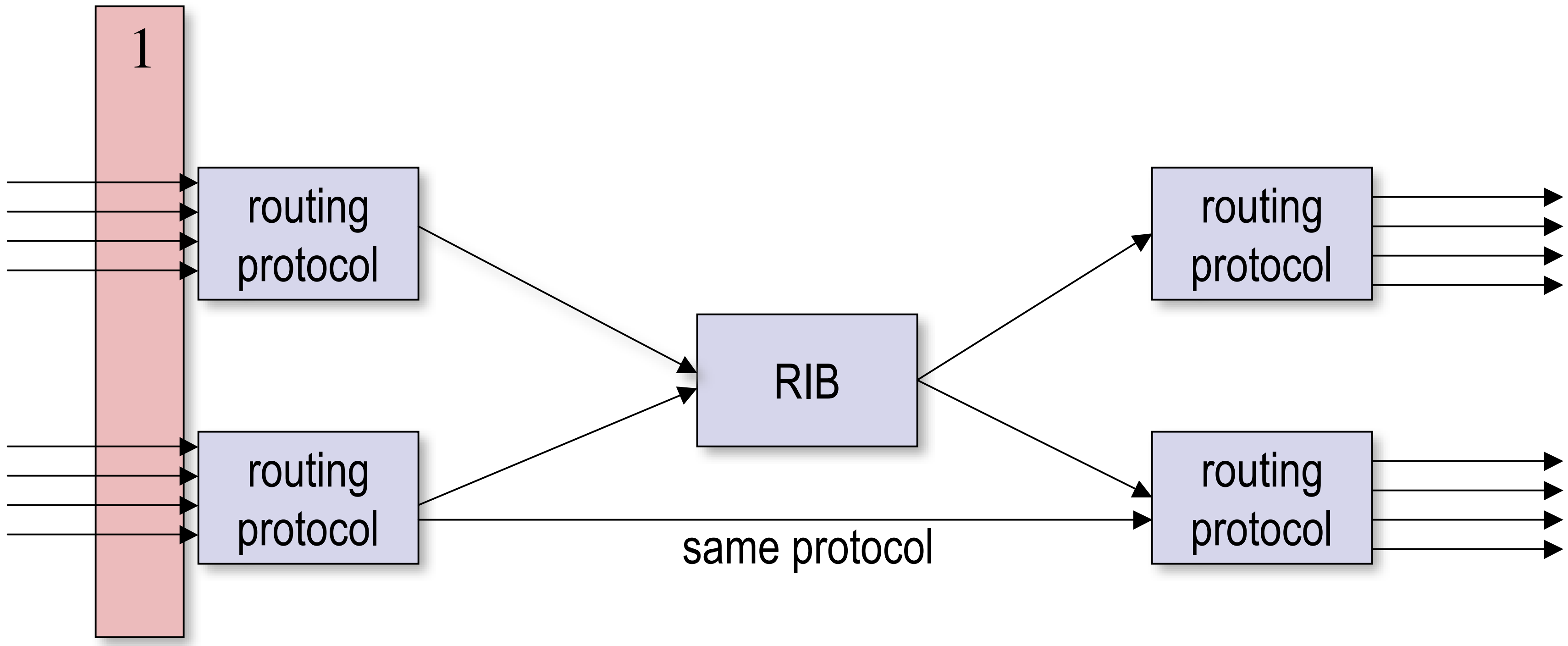
Vector:
BGP, RIP



Link State:
OSPF, IS-IS



Filter Banks

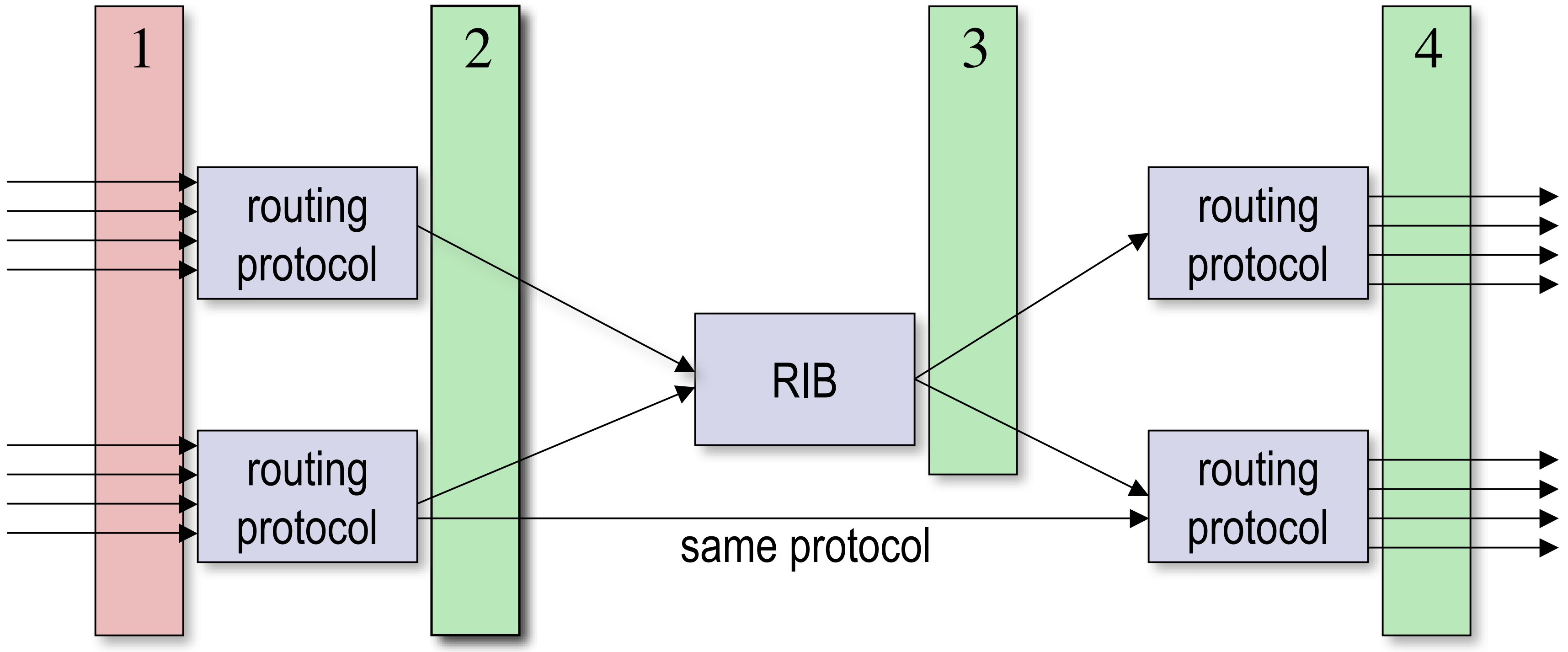


import:
match,
action

Set a LOCALPREF of 3 on incoming BGP routes that have a nexthop of 128.16.64.1

Filter Banks

Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.



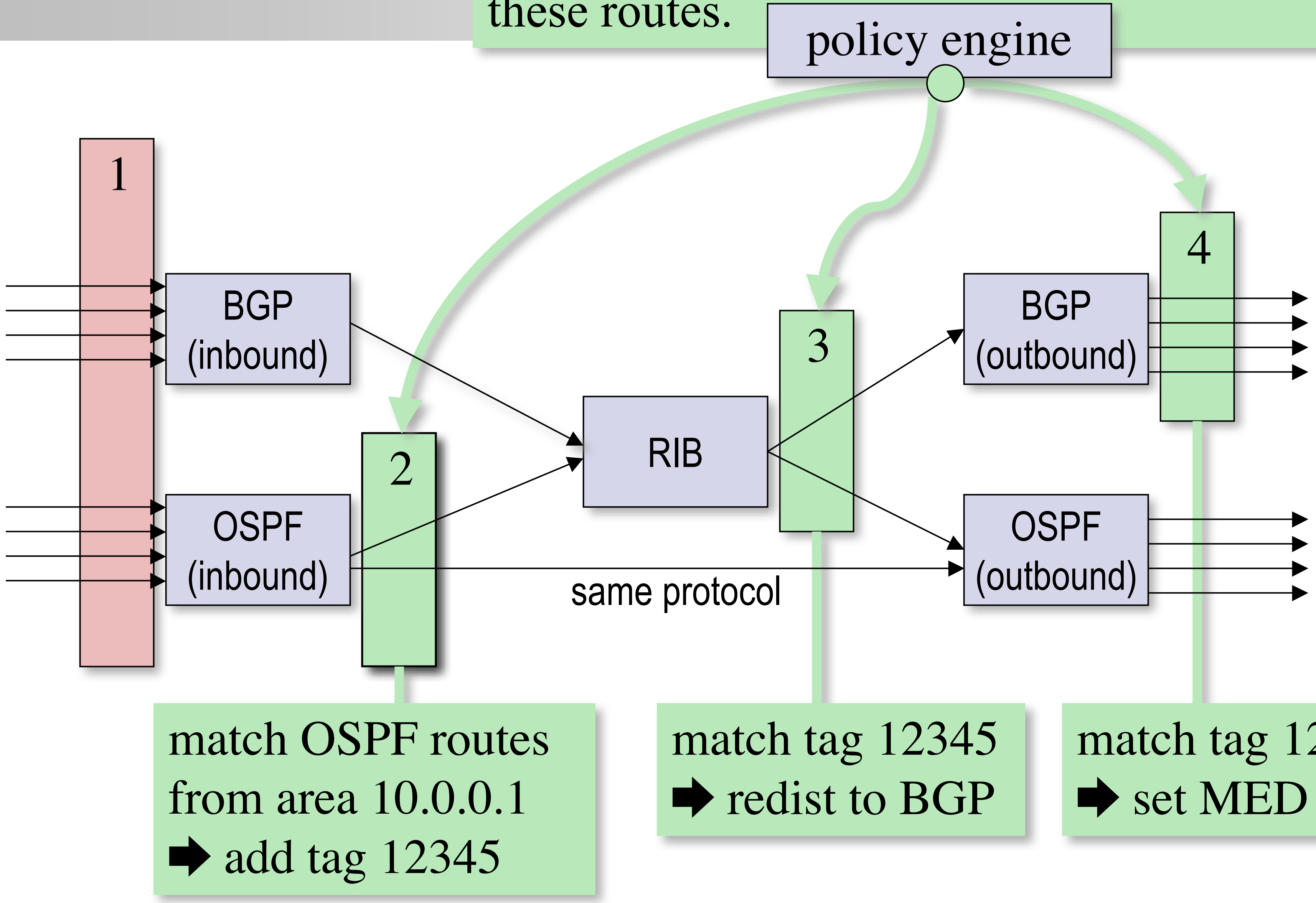
export:
source match

export:
redistribute selected

export:
dest match, action

Filter Banks

Redistribute OSPF routes from OSPF area 10.0.0.1 to BGP and set a BGP MED of 1 on these routes.



match OSPF routes from area 10.0.0.1
➡ add tag 12345

match tag 12345
➡ redist to BGP

match tag 12345
➡ set MED = 1

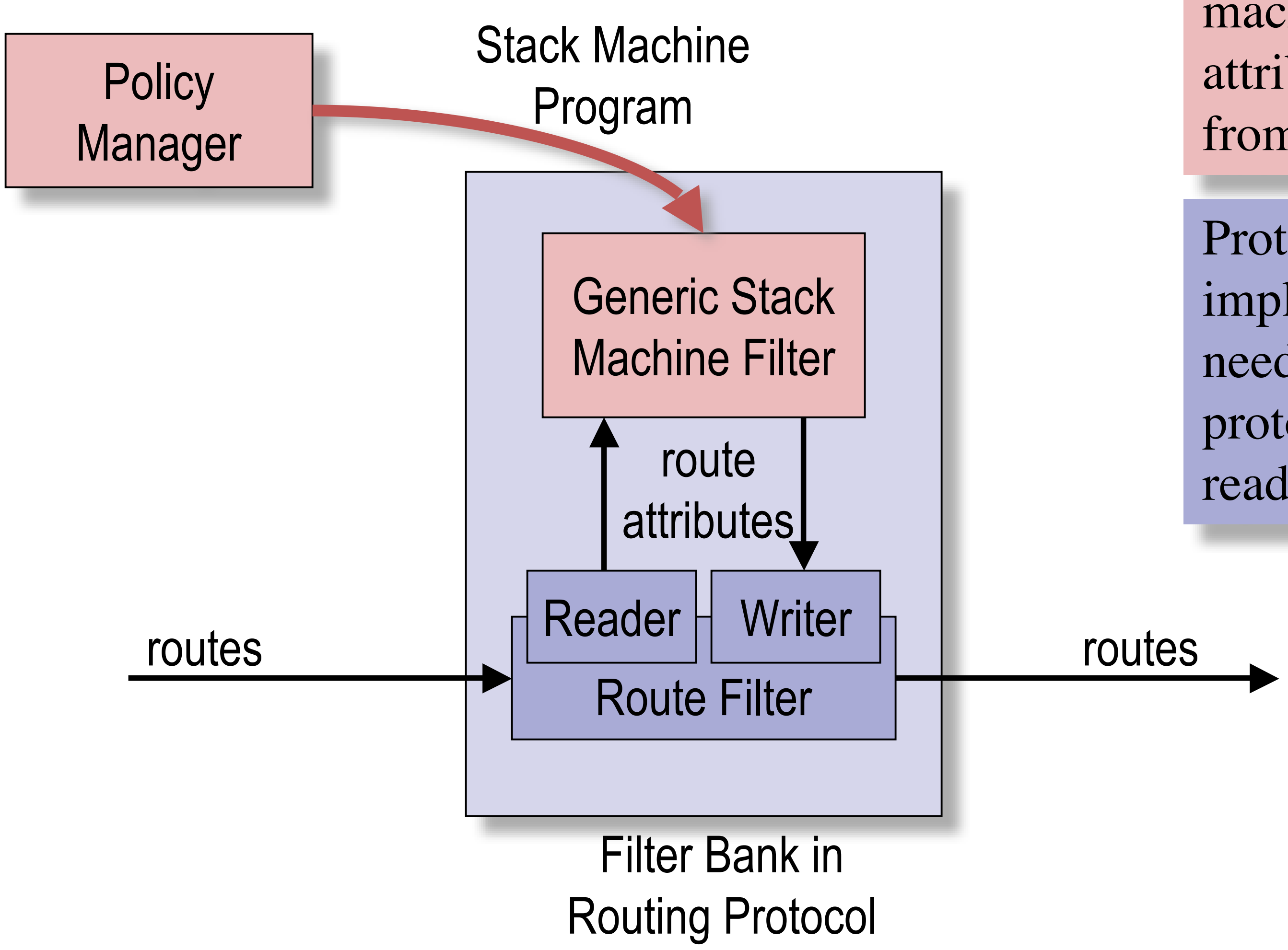
Policy Manager Engine

- Takes a complete routing policy for the router:
  1. Parses it into parts (1), (2), (3), and (4) for each protocol.
  2. Checks the route attribute types against a dynamically loaded set of route attributes for each protocol.

```
bgp     aspath          str     rw
bgp     origin          u32     r
bgp     med             u32     rw
rip     network4        ipv4net r
rip     nexthop4        ipv4    rw
rip     metric          u32     rw
```

  3. Writes a simple stack machine program for each filter, and configures the filter banks in each protocol.

Policy Filter Bank



All filters use the same stack machine. Stack machine requests attributes by name from routing filter

Protocol implementer only needs to write the protocol-specific reader and writer

Stack Machine

Policy Statement

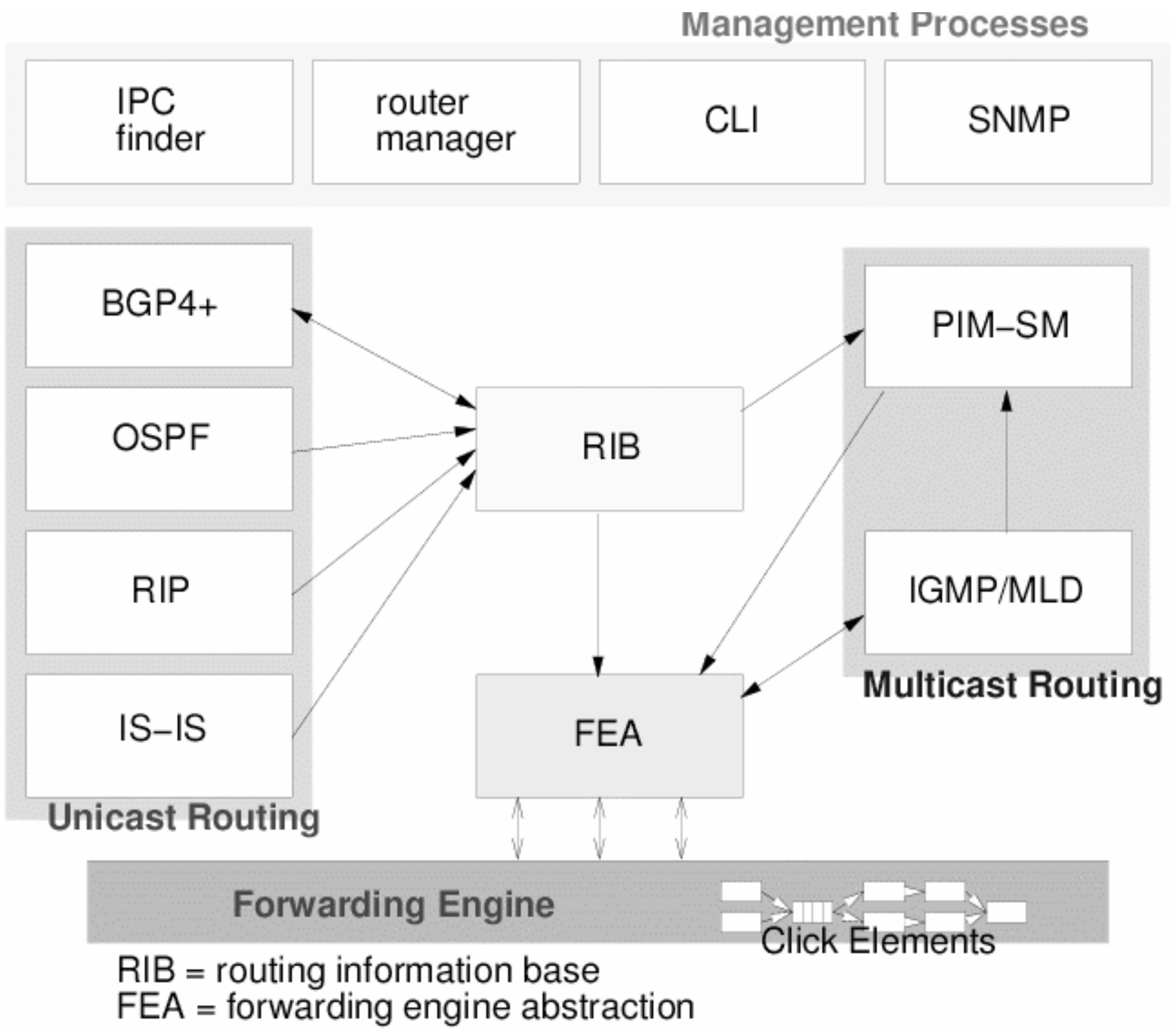
```
from {
    metric > 4
}
then {
    metric = metric * 2
    accept
}
```

Stack Machine Program

```
PUSH u32 4
LOAD metric
>
ON_FALSE_EXIT
PUSH u32 2
LOAD metric
*
STORE metric
ACCEPT
```

Outline of this talk

1. Routing process design
2. Extensible management framework
3. Extensible policy routing framework
4. Performance results



RIB = routing information base
FEA = forwarding engine abstraction

Summary: Design Contributions

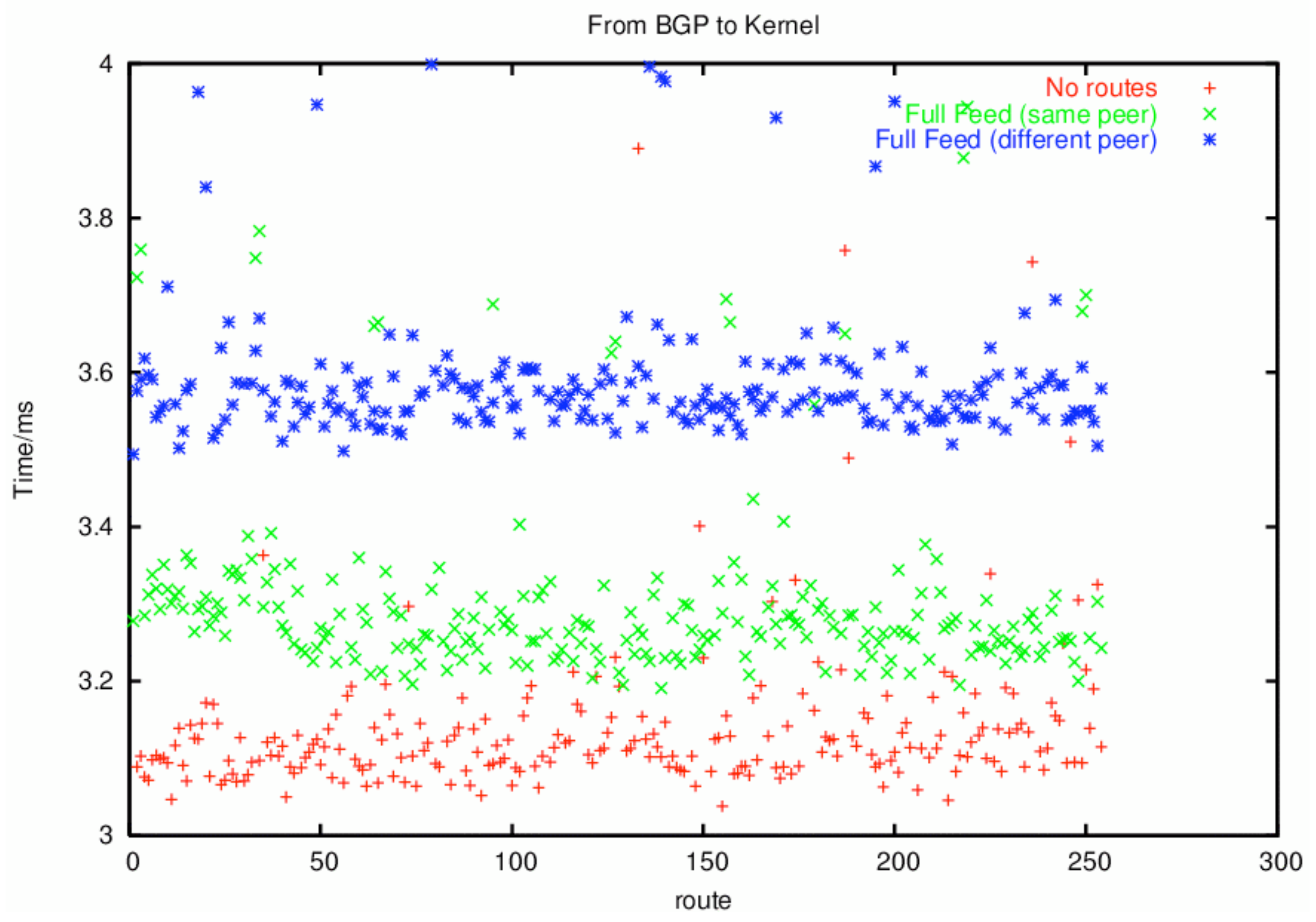
- Staged design for BGP, RIB.
- Scriptable XRL inter-process communication mechanism.
- Dynamically extensible command-line interface and router management software.
- Re-usable data structures such as safe iterators.
- FEA that isolates routing processes from all details of the box the process is being run on.
- Extensible policy framework.

Evaluation

- Was performance compromised for extensibility?

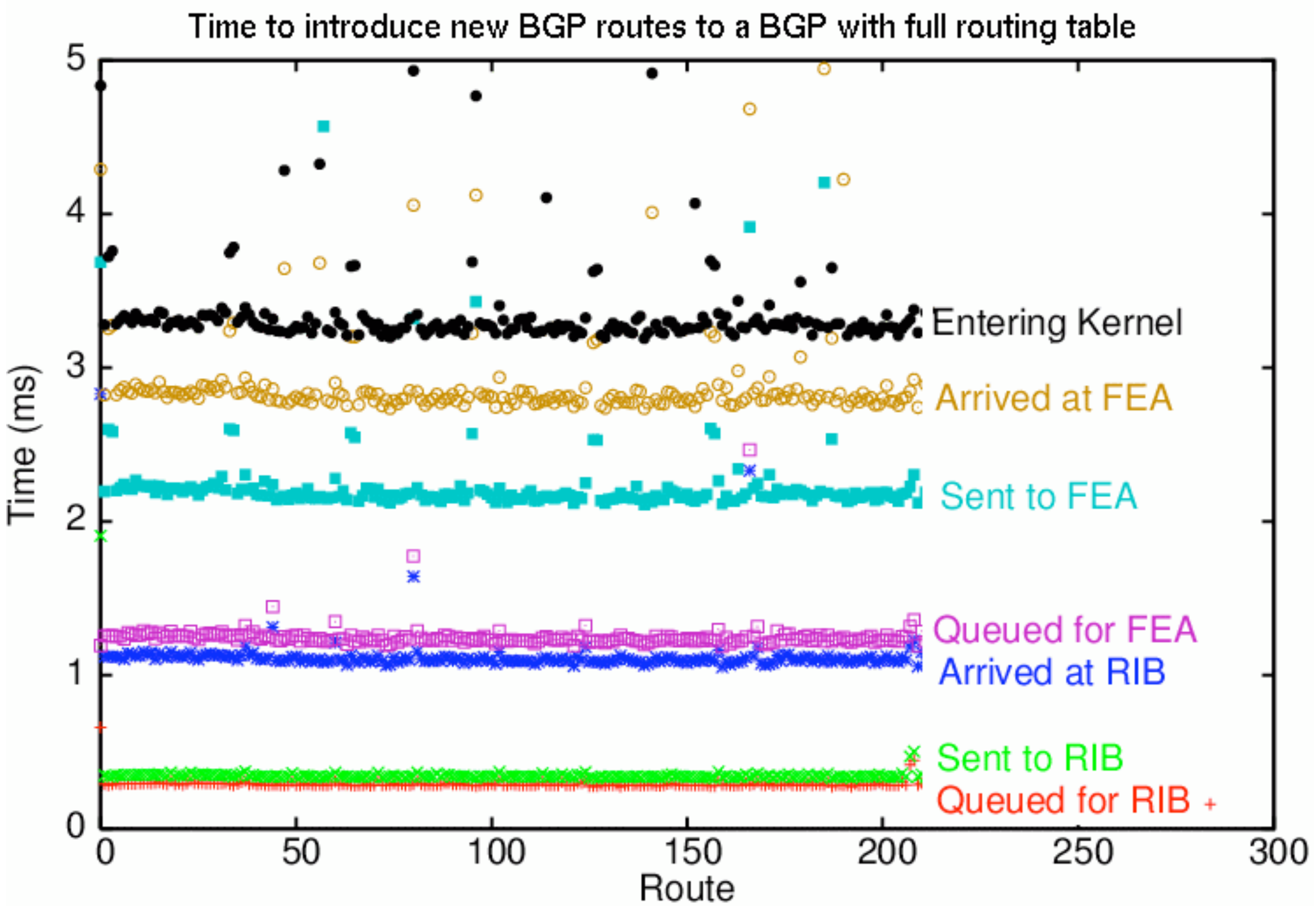
Performance:

Time from received by BGP to installed in kernel



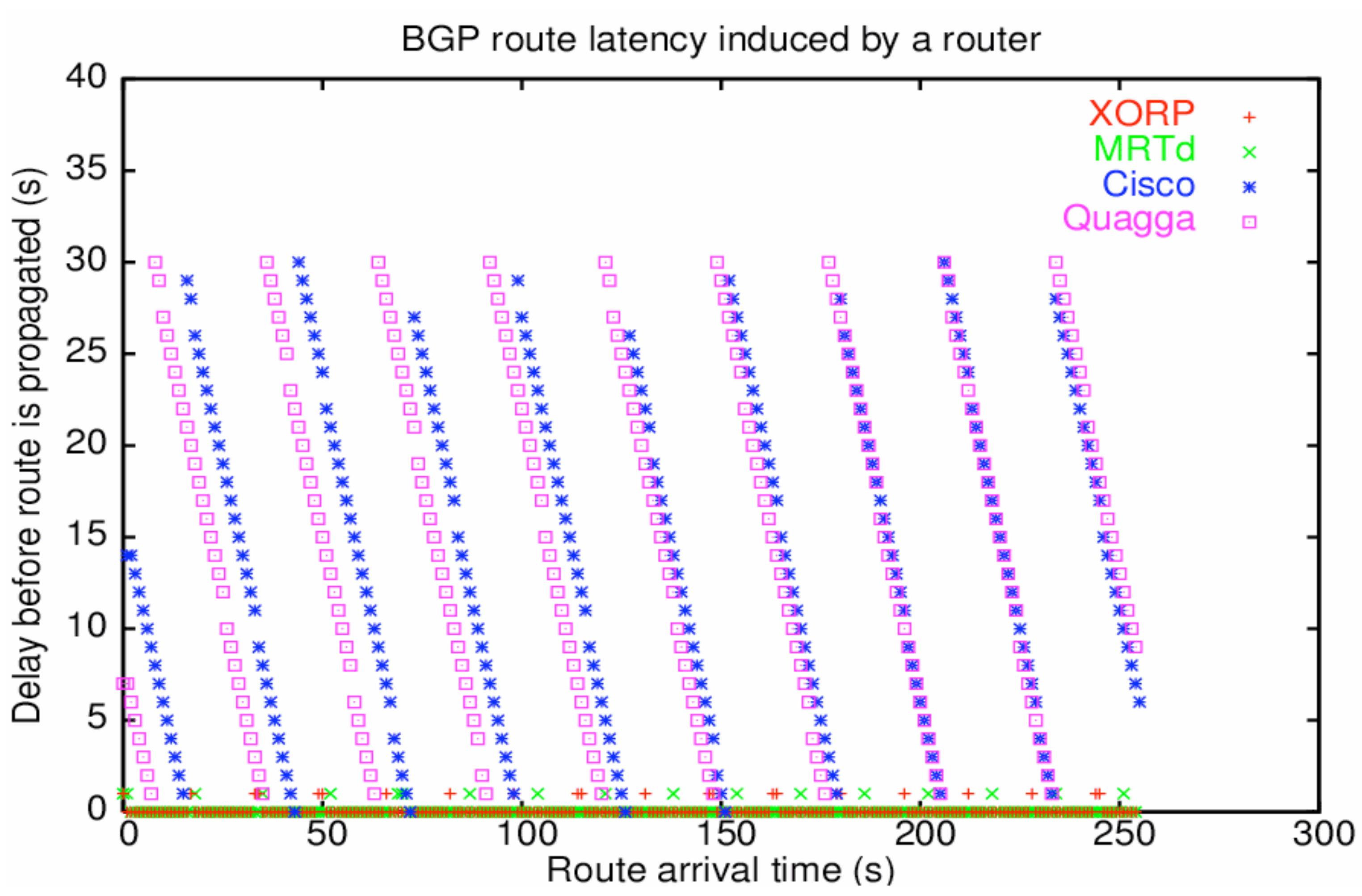
Performance:

Where is the time spent?



Performance:

Time from received by BGP until new route chosen and sent to BGP peer



Current Status

Functionality:

Stable: BGP, OSPFv2, RIPv2, RIPng, PIM-SM, IGMPv2, MLDv1, RIB, XRLs, router manager, xorp command shell, policy framework.

In progress: OSPFv3, IS-IS.

Next: IGMPv3, MLDv2, Bidir-PIM, Security framework.

Supported Platforms:

Stable: FreeBSD, OpenBSD, NetBSD, MacOS, Linux.

In progress: Windows 2003 Server.

Summary

- Designing for extensibility is difficult
  - Needs to be a priority from day one.
  - Anecdotal evidence shows XORP to be easy to extend (once you’ve learned the basics)

- Performance always at odds with modularity and extensibility.
  - For routing software, scalability matters most.
  - Modern CPUs change the landscape, and make better solutions possible.

- Only time will tell if XORP is adopted widely enough to change the way Internet protocols are *developed*, *tested*, and *deployed*.

The Future

- Some more far out plans…
  - Security Ideas
  - On-the-fly software upgrades
  - Hot standby processes
  - Network simulation framework

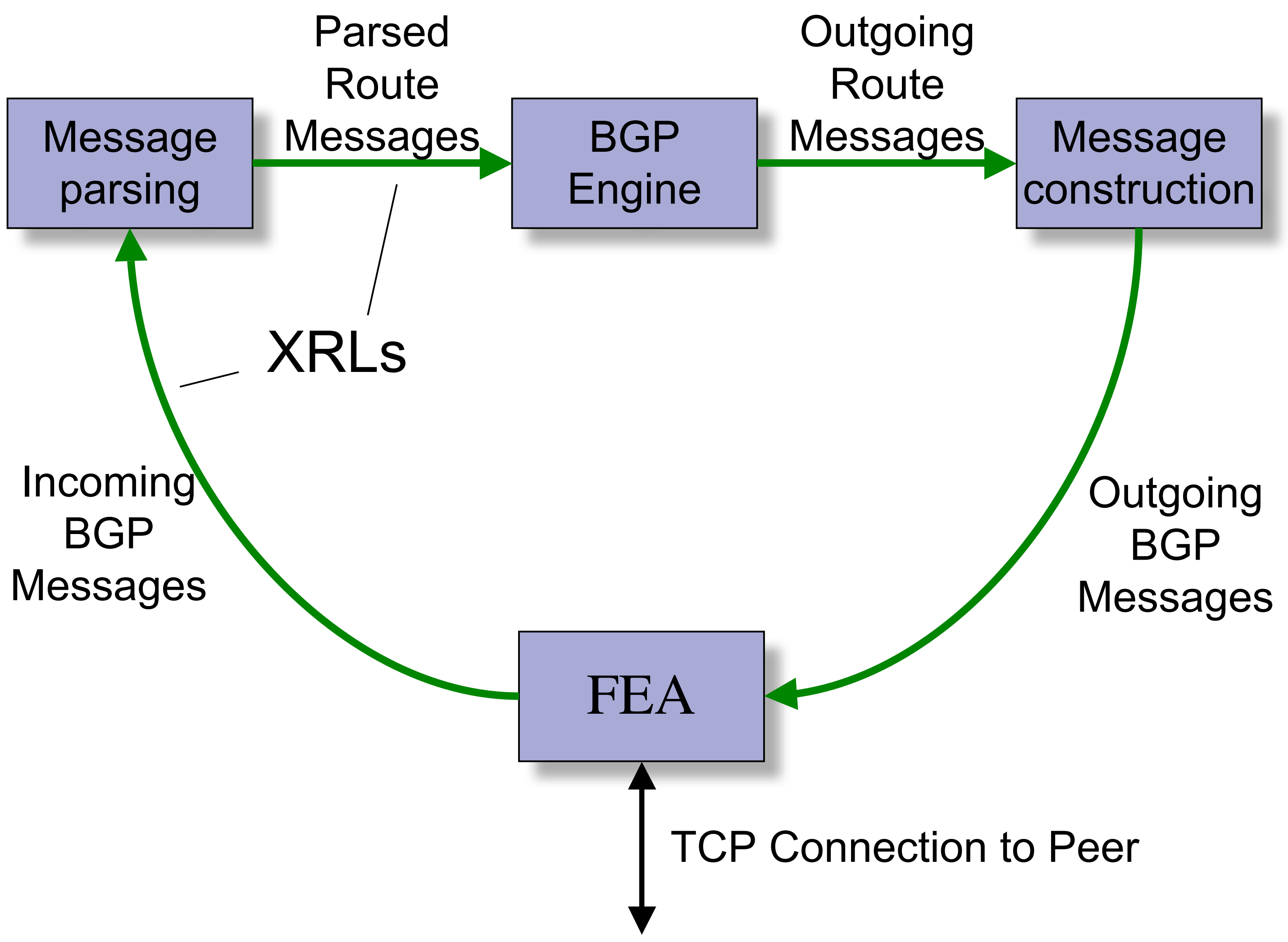
Security

- Use virtualization to isolate processes.
  - Can only interact with the rest of the world through XRLs.
- Use XRL firewalling to restrict XRLs and their parameters needed for the task at hand.
  - Can use a processes template file to specify its permissions.
- Result:
  - An experimental process can do very little harm if it malfunctions or gets compromized.
  - Small performance cost due to XRL firewall enforcement and virtualization.

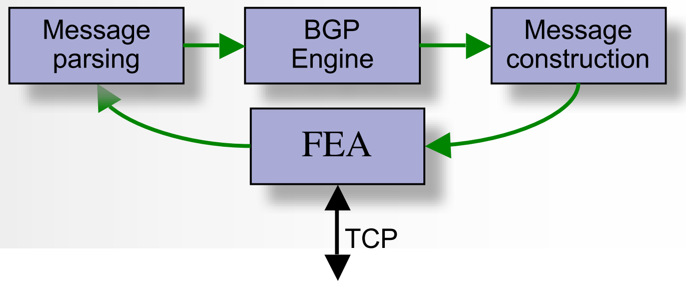
Router Worms

- If I can compromise one BGP, I can compromise them all and shut down your network.
  - Don’t need to break out of the sandbox to do damage.
- Can we prevent router worms?

Router Worm Prevention

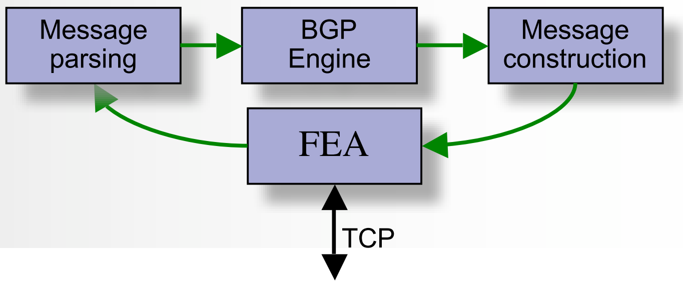


Router Worm Prevention



- Most exploitable security problems are in code handling input data.
- BGP Process separation:
  - Separate packet parsing into a separate process.
  - Parser has no access to data structures and cannot do anything other than send routes via XRLs to BGP Engine.
  - BGP Engine cannot craft BGP messages directly, but relies on separate message construction process.

Router Worm Prevention



Advantages of BGP Process separation:

- Very difficult to compromise *BGP Engine* via parser.
- Even more difficult to compromise *Message Construction* process
  - Would need to do this to craft bad BGP messages.
- Can use safe language for parser (speed less critical)
- Can upgrade parser to fix vulnerability without even dropping the peering.

On-the-fly upgrades

- High uptime is becoming very important.
  - Don’t want to cause routing instability while performing scheduled upgrades of routers.
- BGP is separate from TCP socket code, which is held in the FEA.
  - Can we restart BGP with upgraded software without dropping peerings?

On-the-fly upgrades

At convenient (low churn) moment, can *pause BGP*.

1. Allow internal queues to drain.
2. Only generate keepalives.
3. Save peerings state and RibIn routes.
4. Restore config from rtrmgr and policies from policy manager.
5. Restart BGP process with upgraded software.
6. Reload RibIn routes and peerings state.
7. Unpause BGP.

Constraints:

- The decision algorithm must not change.
- The filter behaviour must not change.
- Policy language must be compatible.

Hot standby process

Can run *two different versions* of a routing process in parallel.

- Hook XRLs so they both receive the same config and incoming routing messages.
- Only one set of routing messages actually sent to neighbours.
- Can switch between primary and secondary process as needed.

Constraints:

- Difficult for hard-state protocols such as BGP, as the two versions may have sent inconsistent messages.
- Good for soft update processes (OSPF, RIP, PIM-SM)

Network Simulation Framework

- Debugging routing code is difficult, time-consuming, and expensive.
- XORP isolates routing code from the OS via the FEA.
  - Already have a Dummy FEA for testing.
- Can craft a Sim-FEA that communicates with other Sim-FEA instances to mimic specific network topologies and configurations.
  - Either on a single machine or a large local cluster.
- Uses might include protocol testing, debugging, and research into new protocols or changes to existing protocols.
  - Close the loop between research and the real world.

Thanks

Main XORP developers:

- Adam Barr, Fred Bauer, Andrea Bittau, Javier Cardona, Atanu Ghosh, Adam Greenhalgh, Tim Griffin, Mark Handley, Orion Hodson, Eddie Kohler, Luigi Rizzo, Bruce M. Simpson, Pavlin Radoslavov, Marko Zec