

Virtualization: Enabler or Problem ?

Dimitri Papadimitriou

ECODE FP7 Project

FIREWeek, Lulea (Sweden)

July 2, 2009

Abstract

- The abstraction paradigm finds its origin in object-oriented-programming and lead to the virtualization paradigm when applied to networking (virtual nodes/networks, links/paths, etc.) e.g. to "overlay" functionalities on top of the network layer. Similarly to object-oriented-programming, the objective is to abstract control actions and data structures to offer desired processing and messaging capability without incurring complexity to external entities. Nevertheless, in distributed environments, factoring out details of individual components requires network-wide level of indirection to abstract reachability/accessibility to these components. For this purpose, an identifier-to-locator mapping-system is required to allow communication between virtual entities and clients-to-virtual serving entities. Additionally, no Internet challenge is identified for which virtualization would constitute a suitable solution: in experimental facilities, it enables resource composition/aggregation as well as re-usability/re-location of dedicated resource shares but has very limited applicability in the Internet. Past experience in overlaying capabilities for e.g. multicast were unsuccessful because of dynamic state processing and host-triggered messaging.
- This talk details the architectural consequences resulting from maintaining and processing resource state information inside the network, and enabling communication with/between virtual entities. It reviews role of experimentation in quality/utility (functional) and rate x state (performance) benchmarking to assess prototypes before positioning virtualization as THE functional enabler of excellence.

Objective

- Architectural consequences resulting from
 - Maintaining and processing resource state information
 - Enabling communication with/between virtual entities
- Role of experimentation for functional (quality/utility) and performance (rate x state) benchmarking
- What is our level of in-depth understanding on virtualization ?

Outline

- Principle and application
- Characterization
- Relationship wrt Overlay and Programmability
- Indirection and Mapping Entries
- Distributed Mapping System
- Communication Principles
- Experimental Challenges

Abstraction

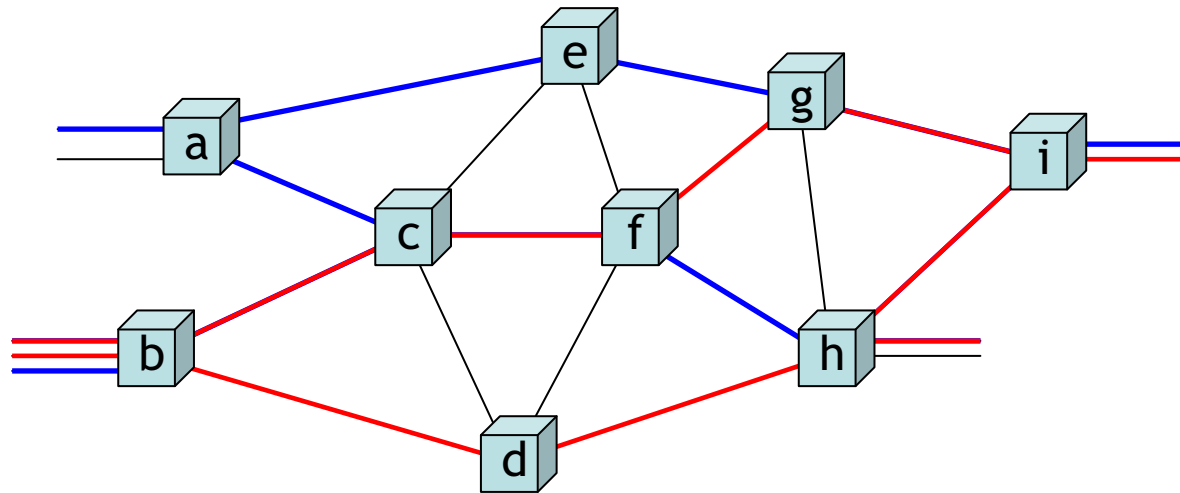
- Principle
 - Origin from object-oriented programming ("hide complexity")
 - Abstracting actions (associated to control/processing) and data structures (associated to message exchanges) to offer desired processing and messaging capability without incurring complexity to entities external to the "system"
 - Factoring out details of the constituting entities of the system itself

Virtualization

- Resources
 - System: CPU, memory, storage capacity
 - Network: forwarding (node), transmission (link)
- Role in experimental facilities:
 - **Resource sharing:** multiple virtual resource instances from single physical/logical resource (-> re-location/re-usability of shares)
 - **Resource aggregation:** grouping of resource components into single virtual resource
 - > Resource either re-used for multiple instances or multiple resource components aggregated into single virtual resource
 - **Resource emulation:** virtual resource type Y from physical/logical (composition) of resource type X_1, \dots, X_n

Virtual Networks

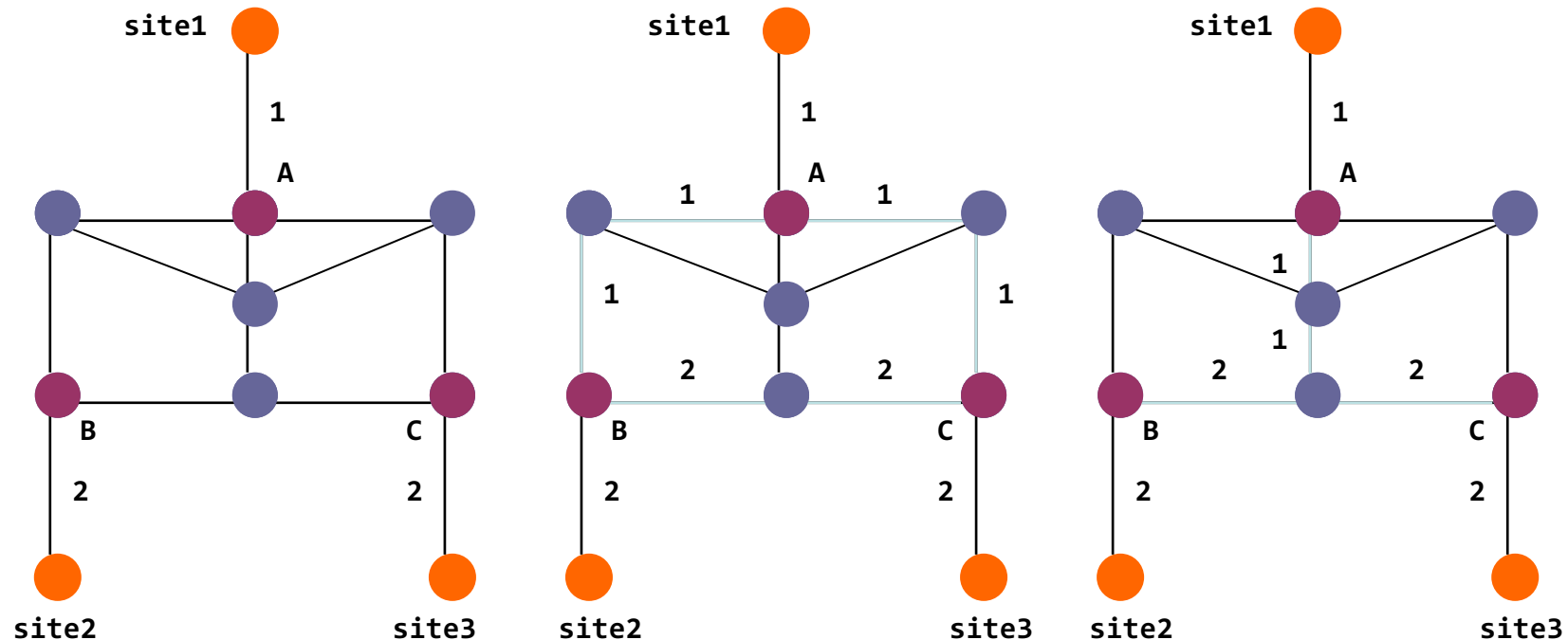
- Virtual (Connectivity) Networks
 - Logical partition of network resources: forwarding (node), transmission (link)
 - Each resource can be sliced so that it can be part of multiple Virtual Networks
 - Each actions in one network do not affect the operation of any other network



Virtual Networks: Hose vs Pipe

- Hose model

- Paradigm: set up paths between every pair of VN endpoints such that i) aggregate bandwidth reserved on the links traversed by the paths is minimum ii) share as many links as possible
- Properties: Ease of Specification and Characterization, Flexibility, Multiplexing Gain



Independent Shortest Paths
Total = 8 units

Link sharing
Total = 6 units

Characterization

- **Abstraction:**

Similarly to object-oriented-programming, abstract control actions and data structures to offer desired processing and messaging capability without incurring complexity to external entities (= terminals/hosts)

Implication: in distributed environments, factoring out details of individual components requires network-wide level of **indirection** to abstract reachability/ accessibility to these components

-> indirect access to resource (components)

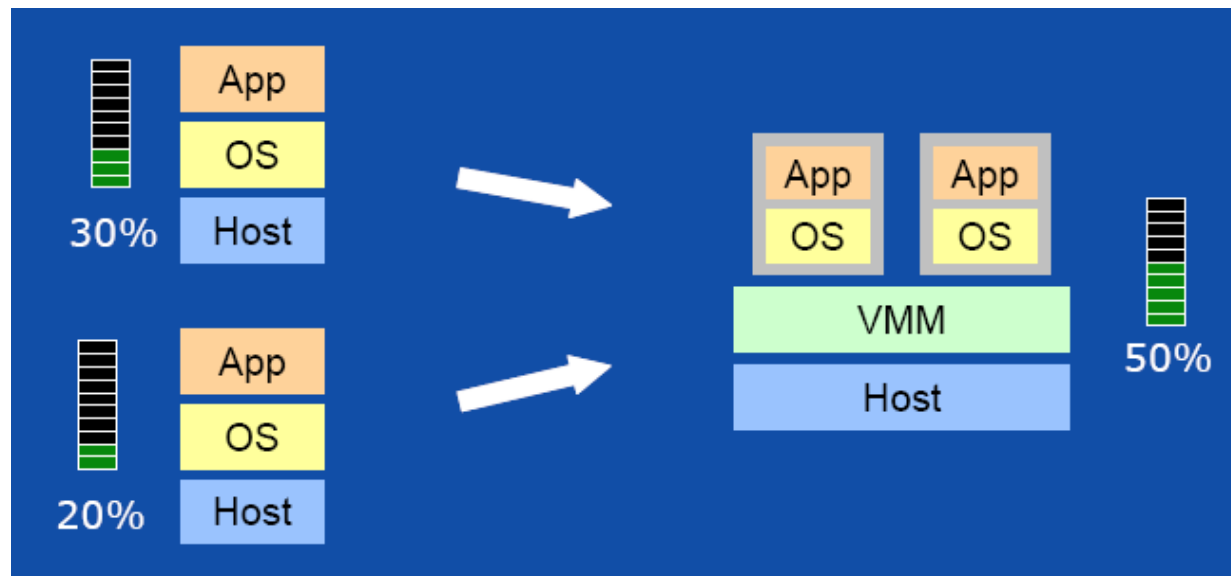
- **Better resource utilization:**

- Sharing (resource re-use/re-allocation)
- Aggregation (resource grouping)
- Emulation (resource re-composition)
- > Stateful (per virtual (sub-)entity)

>< **Isolation** (by loose, strict partitioning)

Second level of indirection

- Resource aggregation (pooling)

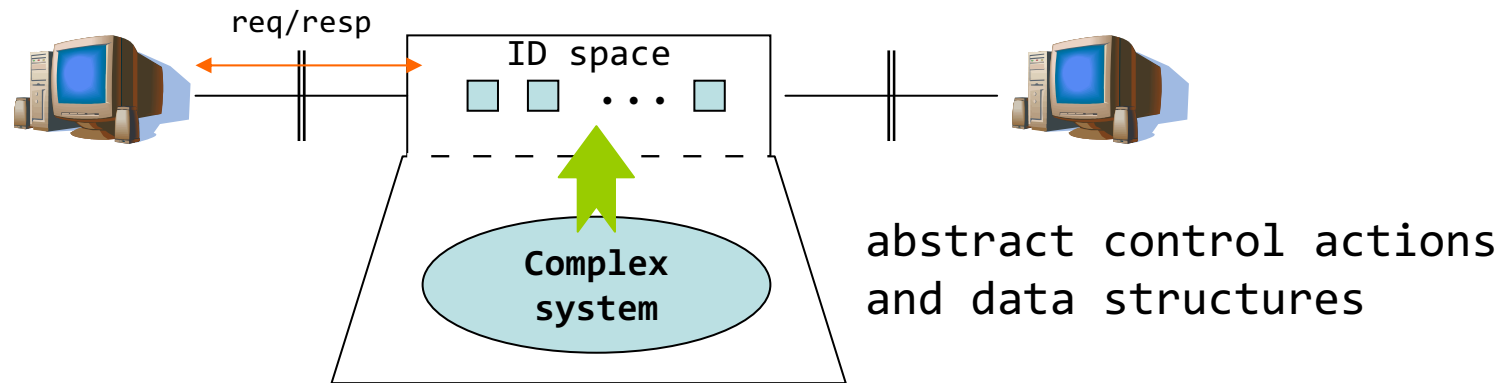


Source: Intel, Keynote for ISPASS 2008

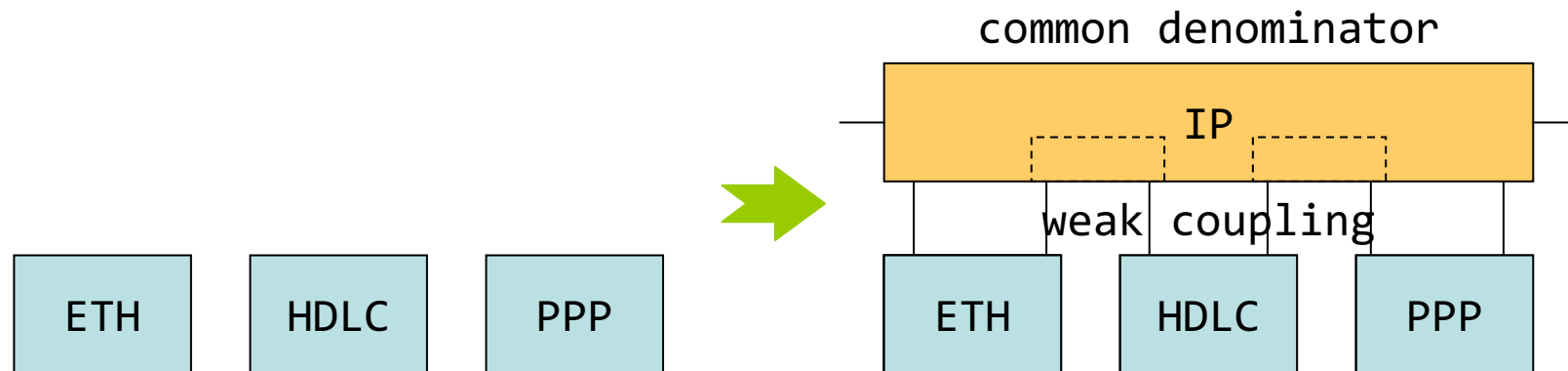
-> Maintain additional level of indirection

Overlay wrt Virtualization

- Virtualization (\leftrightarrow abstraction)



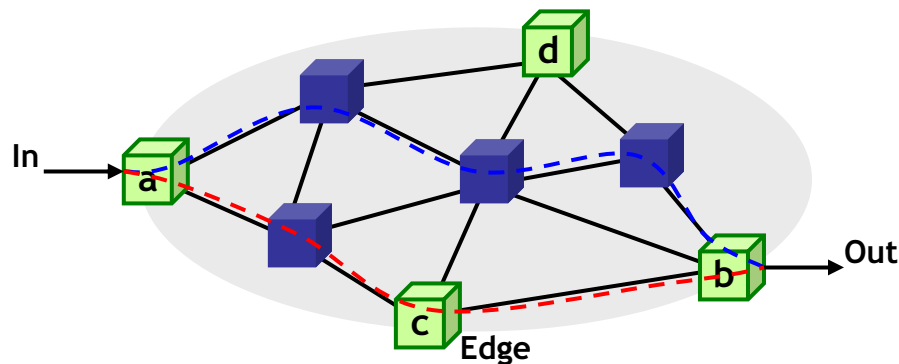
- Overlay (\leftrightarrow layer)



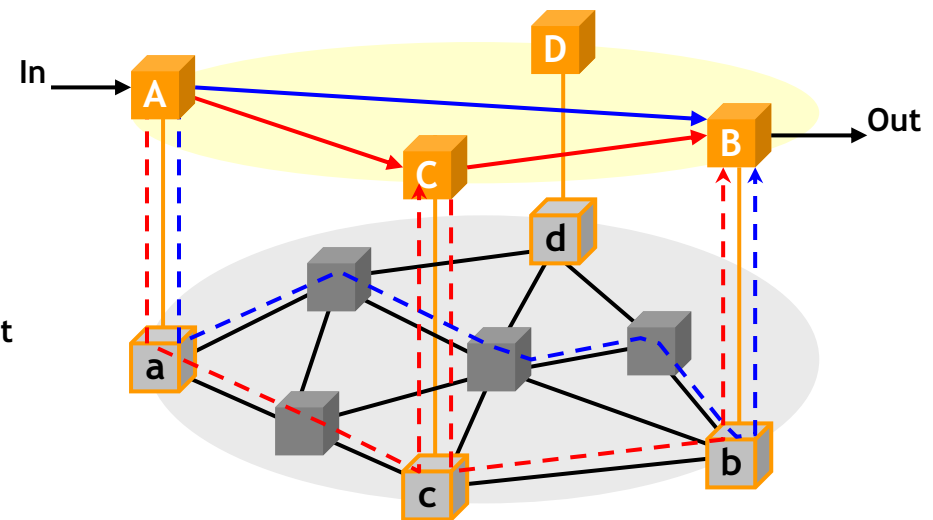
Network routing vs Overlay routing

1. Tackle technological limits (scalability, resiliency, stability, etc. but **also sub-optimal user performance**) and operational limits (policing) of existing **network infrastructure routing**

1. Revisit network “routing functions”



2. Infrastructure-based overlay



2. Or build (infrastructure-based) overlay on top of existing IP layer = additional level of indirection with benefits (such as customization and independence) but also side effects
 1. Change properties in one or more areas of underlying network (e.g. edges)
 2. Horizontal and vertical cross-layer conflicting interactions impacting overall network performance (amplified by selfish routing)
 3. Scalability, stability/convergence, security, etc.

Some Observations on Overlay Routing

- Performing dynamic routing at both overlay and native IP layer leads to conflicting cross-layer interactions because of
 - Functional overlap (unintended interactions/interferences)
 - Vertical: mismatch/conflict in (re-)routing objectives
 - Horizontal: contention for limited physical resources (race conditions & load oscillations)
- Complex cross-layer interactions amplified by
 - Selfish routing where individual user/overlay controls routing of infinitesimal amount of traffic to optimize its own performance without considering system-wide criteria
 - Lack of information about other layer(s) \Rightarrow uninformed optimizations leading to loose-loose situations

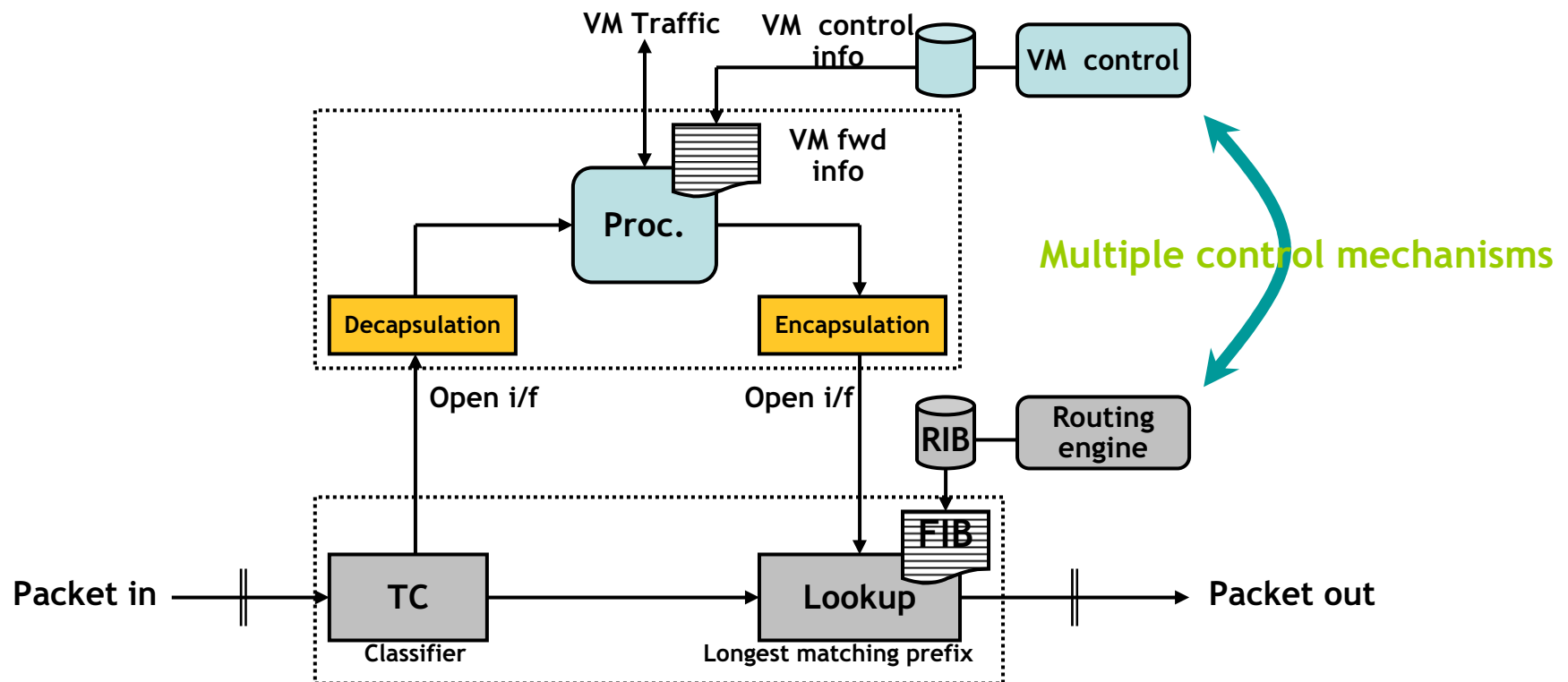
\Rightarrow Need to overcome degradation of overall network performance

General Problem of Indirection

“Any problem in computer science can be solved with another layer of indirection.” – *David Wheeler*

“But that usually will create another problem.”

– *rest of the quote*

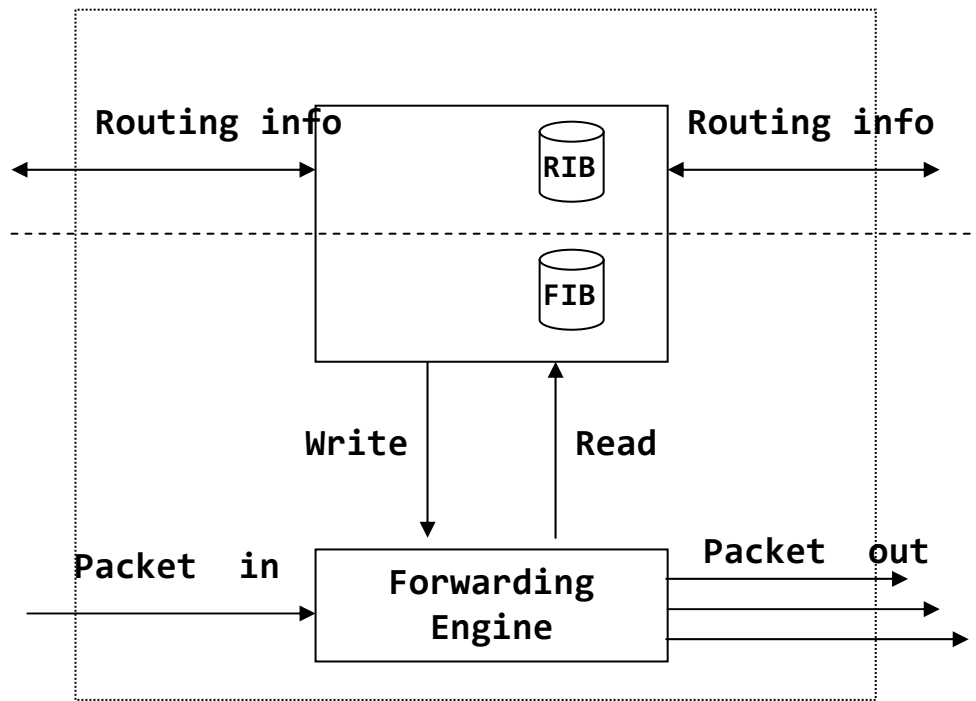


Programmability wrt Virtualization

- Virtualization
 - Multiple logical routers on a single hardware (platform)
 - Resource isolation in CPU, FIBs, and link capacity (bandwidth)
- Programmability (part of)
 - General-purpose CPUs for control and processing
 - *NPA*s (and *FPGAs*) for fast forwarding
 - Extensible and open routing engine

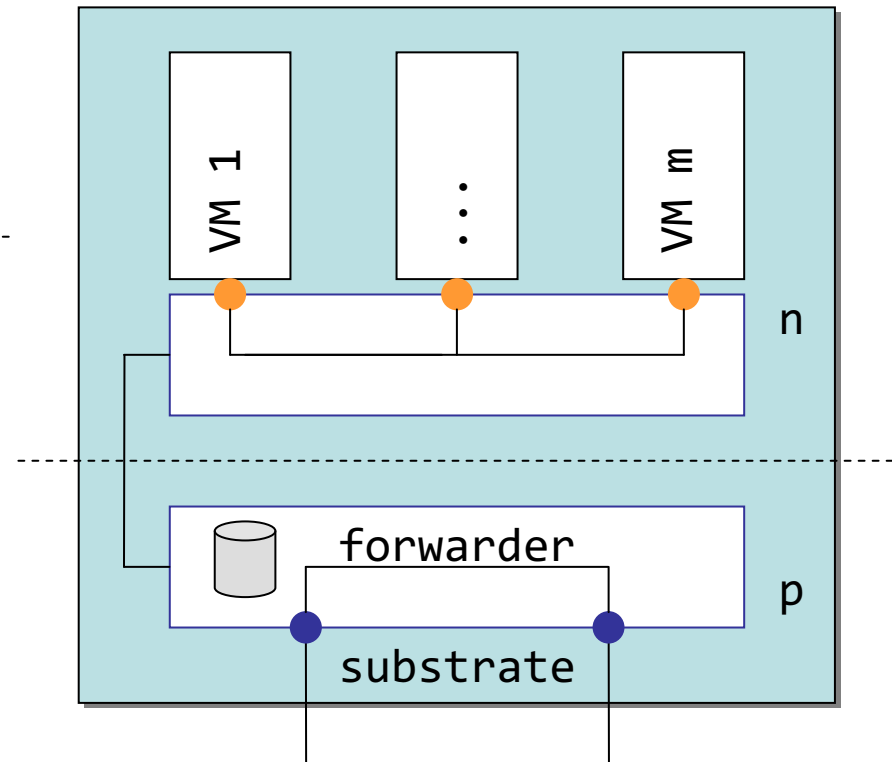
Router and Virtual Router

Functional view of "hard" router



Forwarding tables (usually)
distributed on line cards

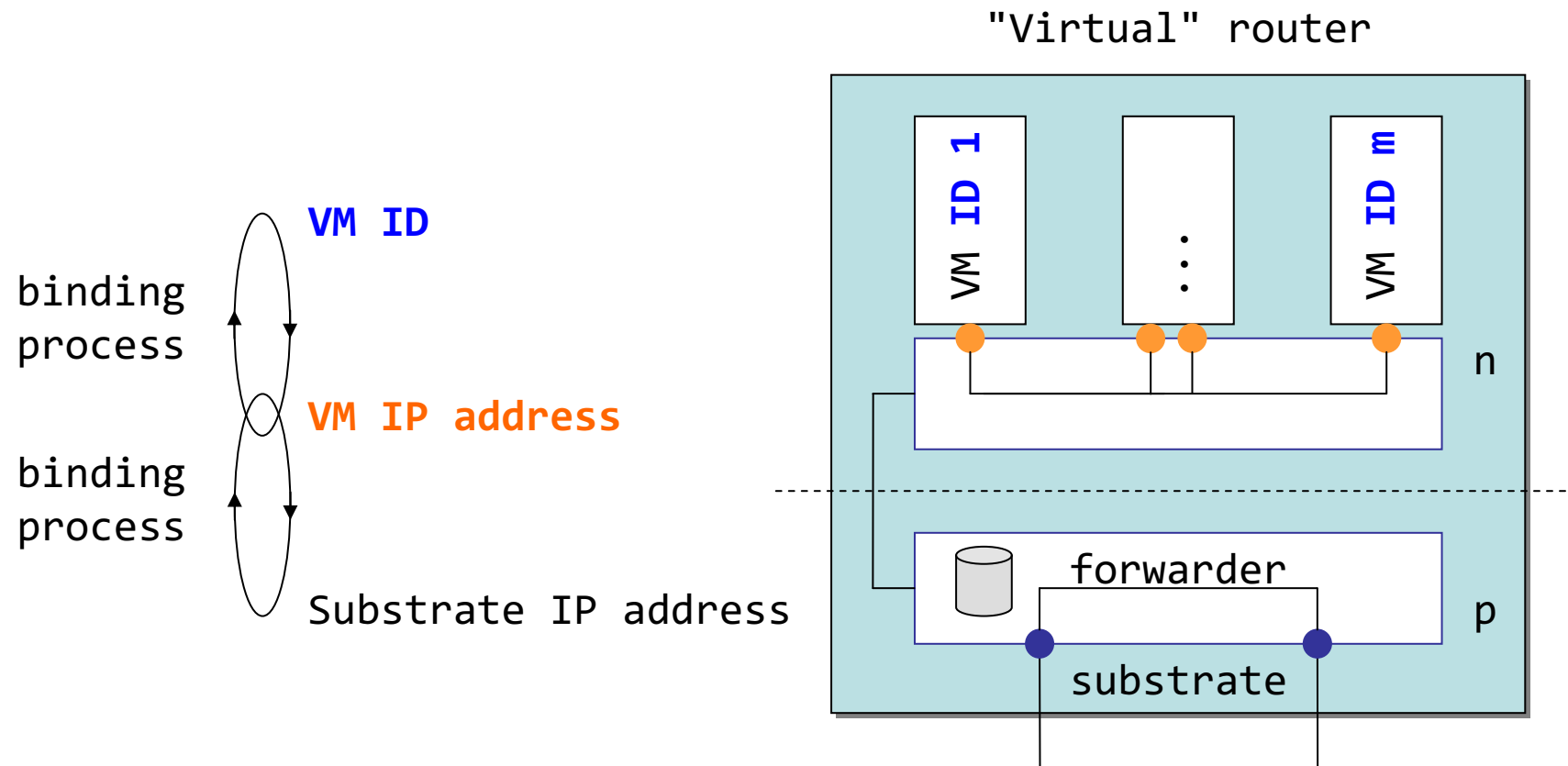
"Virtual" router



VM = virtual machines

Virtual Router

- Local inter-connection: binding process



VM = virtual machines

Programmability wrt Virtualization

- **Specialized platforms (hard-routers)**
 - Root-cause: monolithic design of certain components (performance) and optimized low level code that does not allow for customization (option stripping at design phase)
 - Distributed packet processing on LCs (distributed FIBs with "virtual output queues") interconnected by a cross-bar switched backplane
 - Con's:
 - admitted complexity
 - little flexibility in terms of (re-)programmability (no APIs)
 - no possibility of custom-addition of features (to prevent performance degradations)
- **Generic platforms (soft-routers)**
 - Objectives (programmability): performance at lower cost vs flexibility, and evolutivity at reasonable performance
 - How to build platforms that can process packets at wirespeed (problem of "switching" in shared bus commodity hardware is still not resolved at least since so far)
 - Trend: specialization of commodity platforms in similar direction than 3rd generation routers

Programmability wrt Virtualization

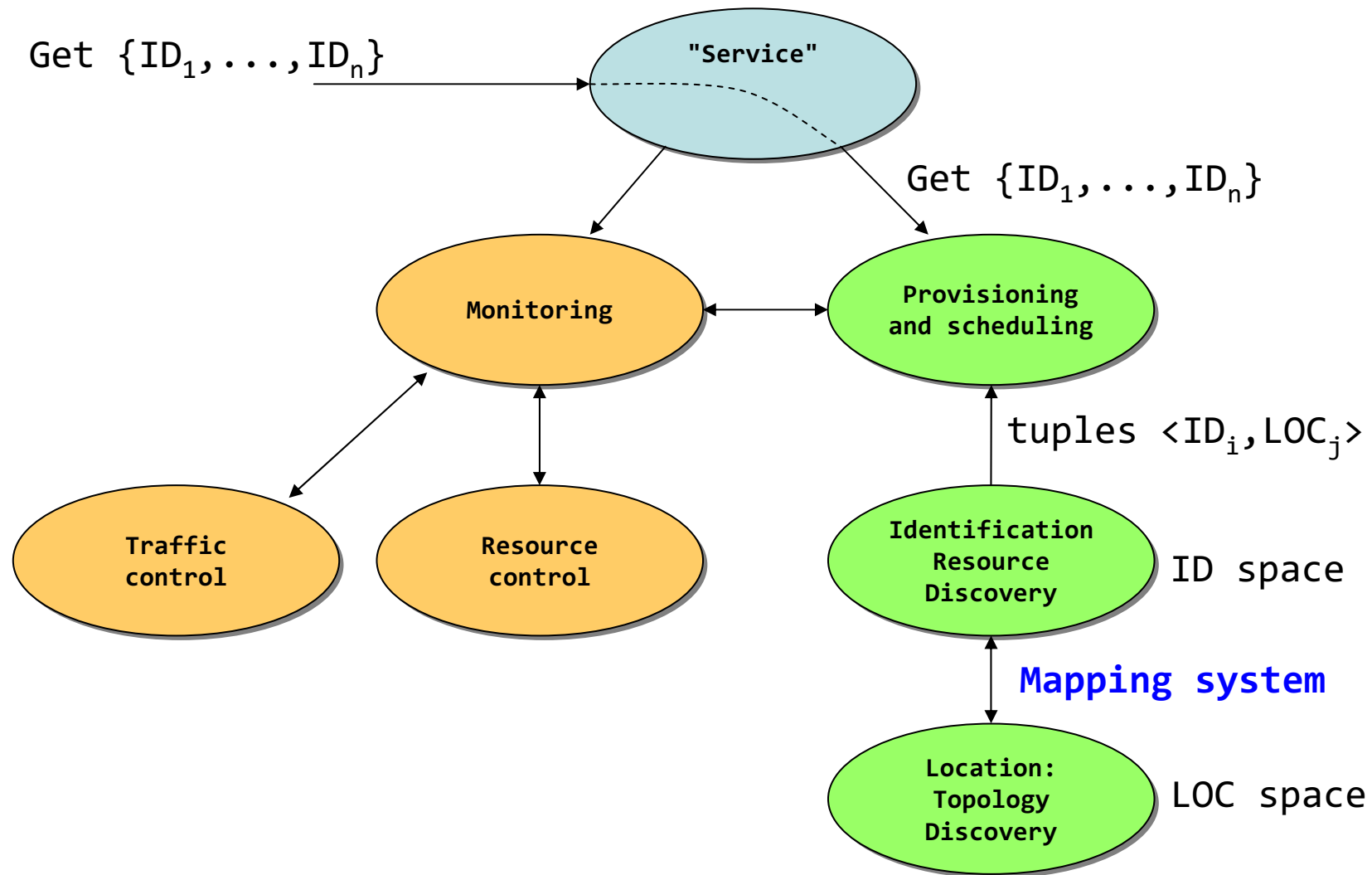
Open Questions

- Trade-off between Complexity (resulting from feature addition and dimensions), flexibility (does one size fits all), and performance (both in terms of energy but also CPU/memory consumption) ?
- Cyclic debate between "specializing" open commodity platforms and "generalizing" close dedicated platforms (first generation routers were also driven by computing system I/O design and soft-processing)

Note: if the evolution is

- Step 1) incorporate new features in software over commodity hardware (-> specialized platform) and then
- Step 2) specialization to non-commodity hardware
Then "soft-platforms" of today will not be programmable tomorrow !

"Reservation - Monitoring" Chain

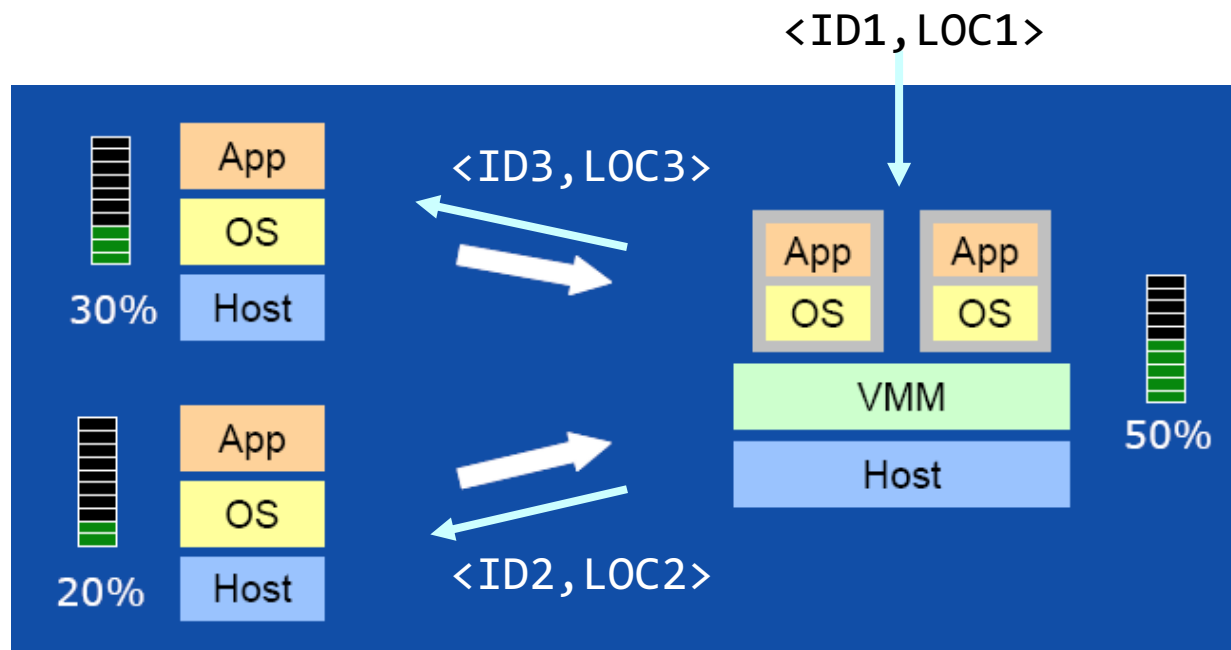


Mapping entries

- Mapping entries: $\{\langle ID_1, LOC_1 \rangle, \dots, \langle ID_i, LOC_j \rangle, \dots, \langle ID_m, LOC_n \rangle\}$
- Dynamics of mapping entries driven by resource sharing (sharing) x resource aggregation (pooling)
 - ADD
 - ADD ID_n at given LOC: $\{\langle ID_1, LOC \rangle, \dots, \langle ID_i, LOC \rangle, \dots, \langle ID_m, LOC \rangle\}$
 - ADD LOC_n for given ID: $\{\langle ID, LOC_1 \rangle, \dots, \langle ID, LOC_j \rangle, \dots, \langle ID, LOC_m \rangle\}$
 - REMOVE
 - Remove ID_n at given LOC: $\{\langle ID_1, LOC \rangle, \dots, \langle ID_i, LOC \rangle, \dots, \langle ID_n, LOC \rangle\}$
 - Remove LOC_n for given ID: $\{\langle ID, LOC_1 \rangle, \dots, \langle ID, LOC_j \rangle, \dots, \langle ID, LOC_m \rangle\}$
 - MOVE
 - $ID_m \rightarrow ID_n$ at given LOC
 - $T(i)$: $\{\langle ID_1, LOC \rangle, \dots, \langle ID_i, LOC \rangle, \dots, \langle ID_m, LOC \rangle\}$
 - $T(i+1)$: $\{\langle ID_1, LOC \rangle, \dots, \langle ID_i, LOC \rangle, \dots, \langle ID_n, LOC \rangle\}$
 - $LOC_m \rightarrow LOC_n$ for given ID
 - $T(i)$: $\{\langle ID, LOC_1 \rangle, \dots, \langle ID, LOC_j \rangle, \dots, \langle ID, LOC_m \rangle\}$
 - $T(i+1)$: $\{\langle ID, LOC_1 \rangle, \dots, \langle ID, LOC_j \rangle, \dots, \langle ID, LOC_n \rangle\}$

Second level of indirection

- Resource aggregation (pooling)
-> Maintain additional indirection



Source: Intel, Keynote for ISPASS 2008

Mapping entries

- Set of tuples $\langle ID_i, LOC_j \rangle$: $|ID| = M$, $|LOC| = n$, $\sim O(n.M)$
number of states in system

- Effect of sharing:

$$|ID| = M \Rightarrow |\text{shared ID}| = \sum_j k_j |M_j|$$

where k_j is the share rate (per location j) for resource M_j

- Effect of pooling:

$$|ID| = M \Rightarrow |\text{pooled ID}| = \sum_1 \sum_i \mathbf{C}(m, i)$$

(1 subsets of $|ID|$, each with m elements) assuming simple combinations across all locations

\Rightarrow Flexibility decreases scalability

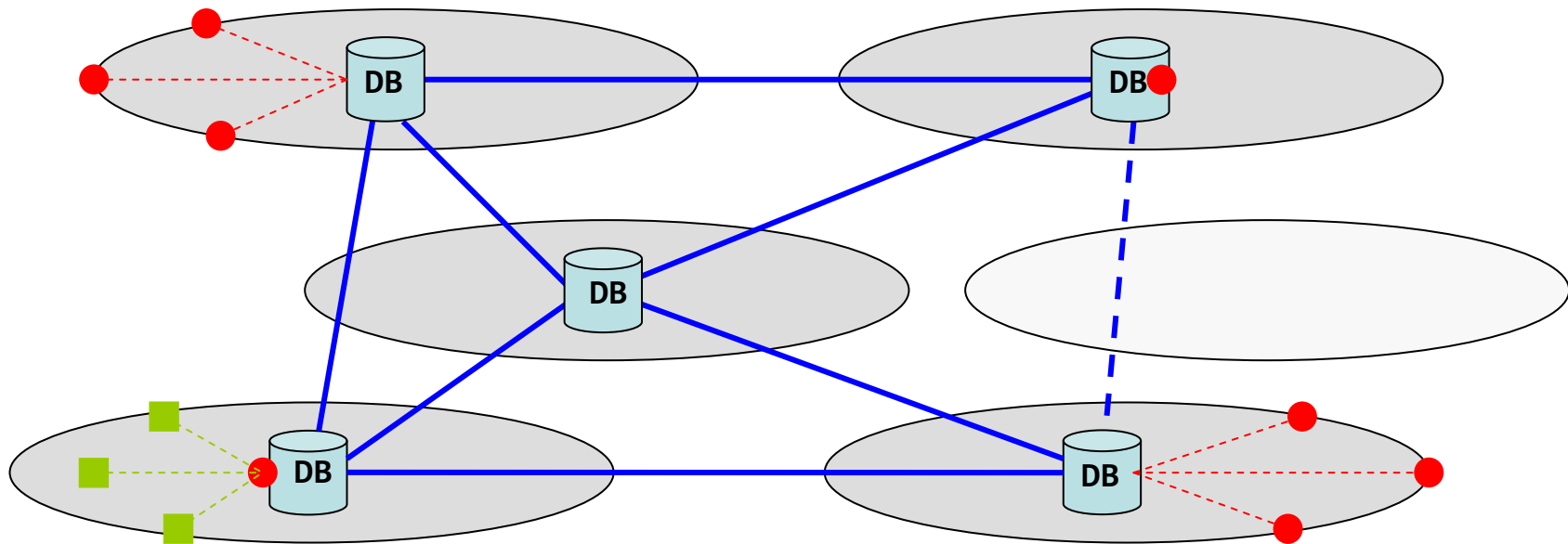
$$|\text{Shared ID} \times \text{Pooled ID}| \sim n.[k.M] + \sum_i \mathbf{C}(m, i)$$




Indirection

- Identifier-to-locator mapping-system required to allow provisioning & scheduling but also monitoring of **distributed** (virtual) resource components
 - Remember binding between <VM ID, VM Locator>
- Host/terminal-to-virtual serving entities communicate by means of "resource IDs"

Distributed Mapping System

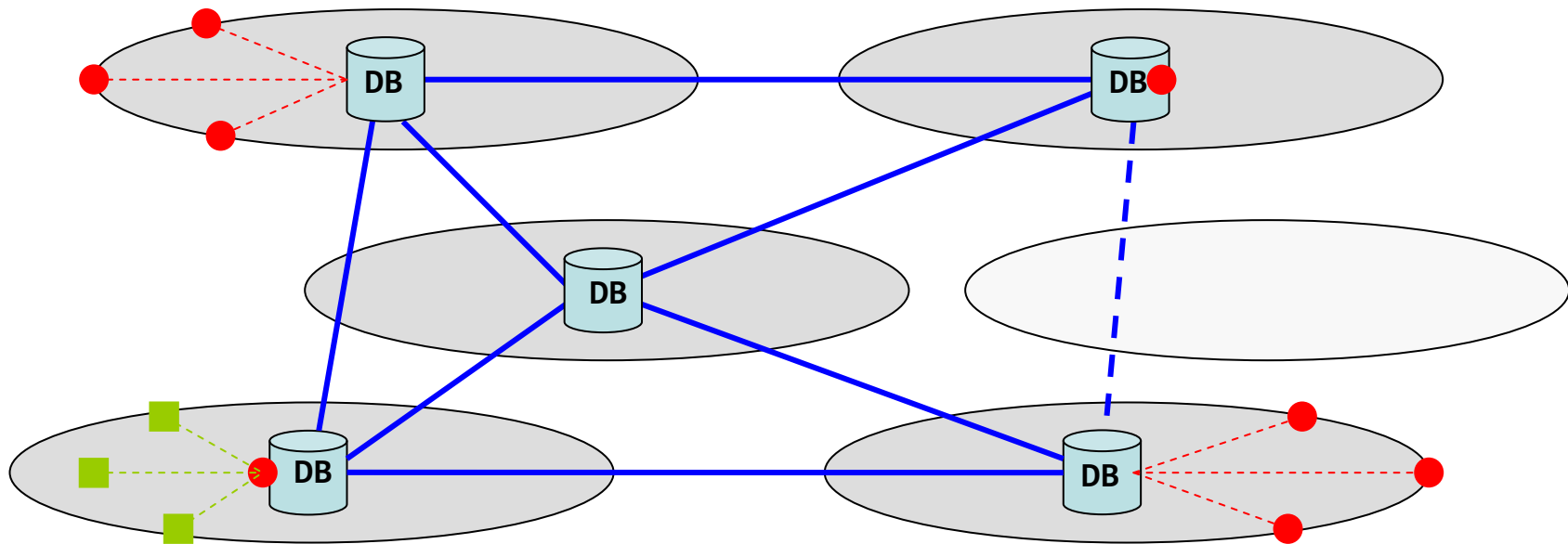
- At min. each admin unit with its own <ID,LOC> database



- Requires communication (of <ID,LOC> tuples \equiv states)
 - Between databases 
 - Between DB and "provisioning & scheduling (PS) points" 
 - Between PS and distributed "monitoring points" 

Distributed Mapping System

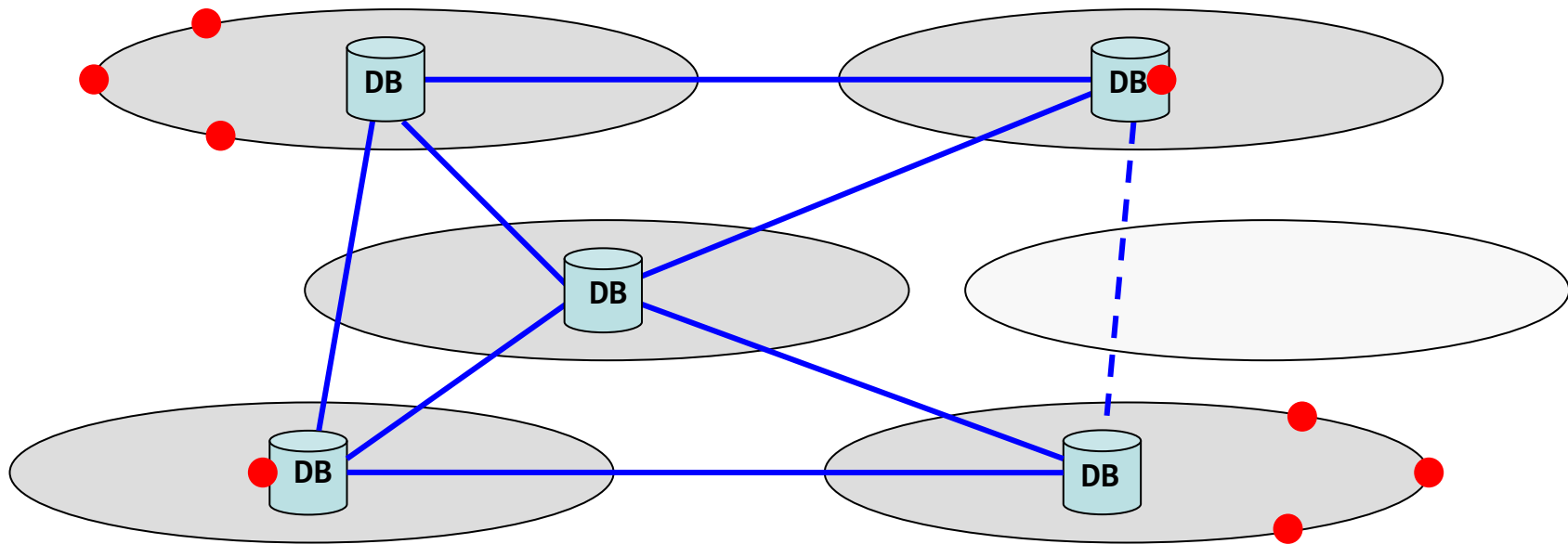
- At min. each admin unit with its own <ID,LOC> database



- Communication mode:
 - Push (dissemination) usually between DB
 - Pull (request/ response) to DB
 - Hybrid push-pull

Distributed Mapping System

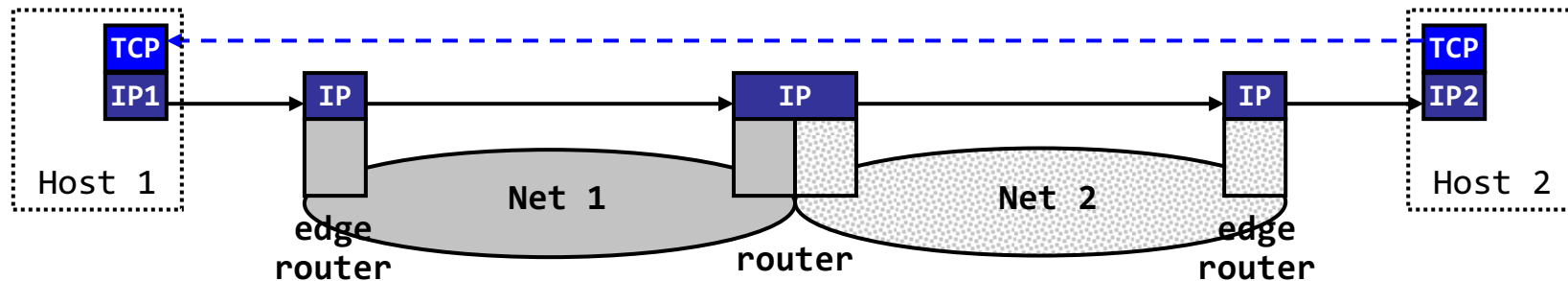
- At min. each admin unit with its own <ID,LOC> database



- Mapping system dynamics driven by tradeoff between (update) rate x (number of) state x latency (push vs pull)

Communication Principles

- Internet model
 - End-to-end loose coupling (vertical): **user-stateless network**
 - Internetwork layer for inter-connection of autonomous networks with loose coupling (**horizontal**): **network state independence**



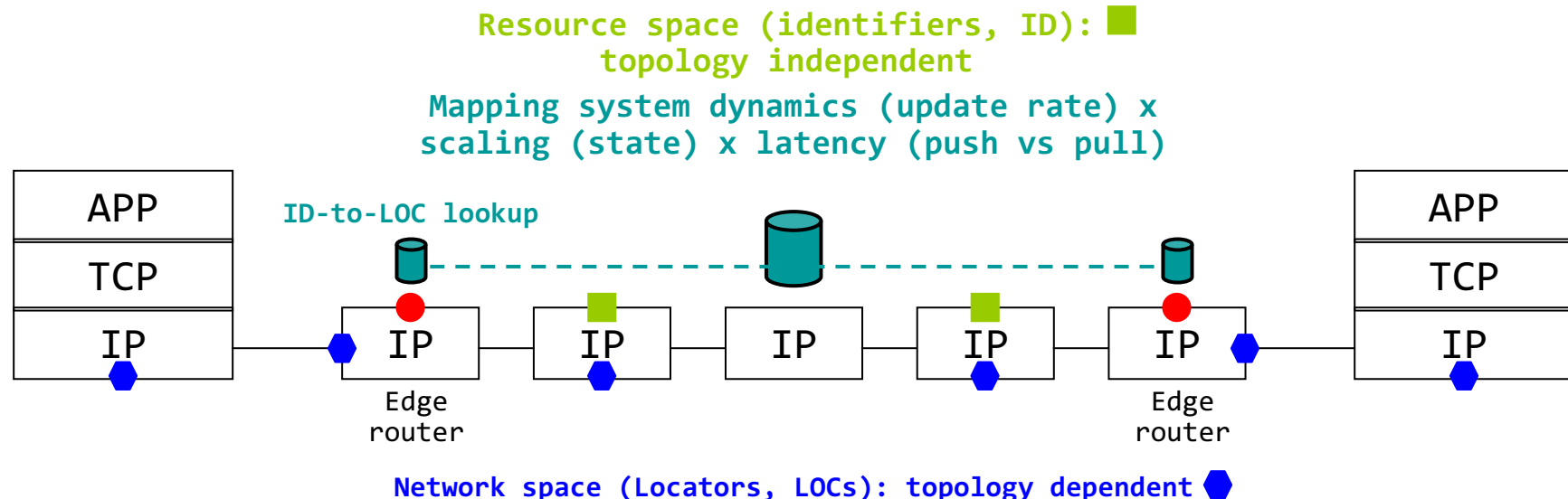
- > Host1 knows source IP (via DHCP), and IP dest (IP2) about the network itself (and will receive feedback from TCP)

Communication Principles

Principles

Separate address topology independent resource identifier (= ID space) and topology dependent locator (= address space)

Resolution (for location) via distributed database incl. information necessary to translate topology independent identifiers to topology dependent addresses (locators)



Performance Dependencies

- **Responsiveness:** how system "responds" to
 - Demand/arrival rate (and variation) wrt load
 - Intentional changes
 - ↔ Mapping system dynamics driven by trade-offs between rate x state x latency
- **Churn:** effect of intentional system changes for "in-use" mappings
 - How mapping system adapts to unintentional changes in reachability (if at all)
 - Dependence on something not on the data path (→ uncontrolled weaknesses)
- Resilience of mapping system itself
- "Complexity/robustness/fragility" spiral [DOYLE02]

Which Edge ?

Which entities external (customer edge or host/terminal) ?

- Network locators terminate at customer premises
- TCP connection at host/terminal

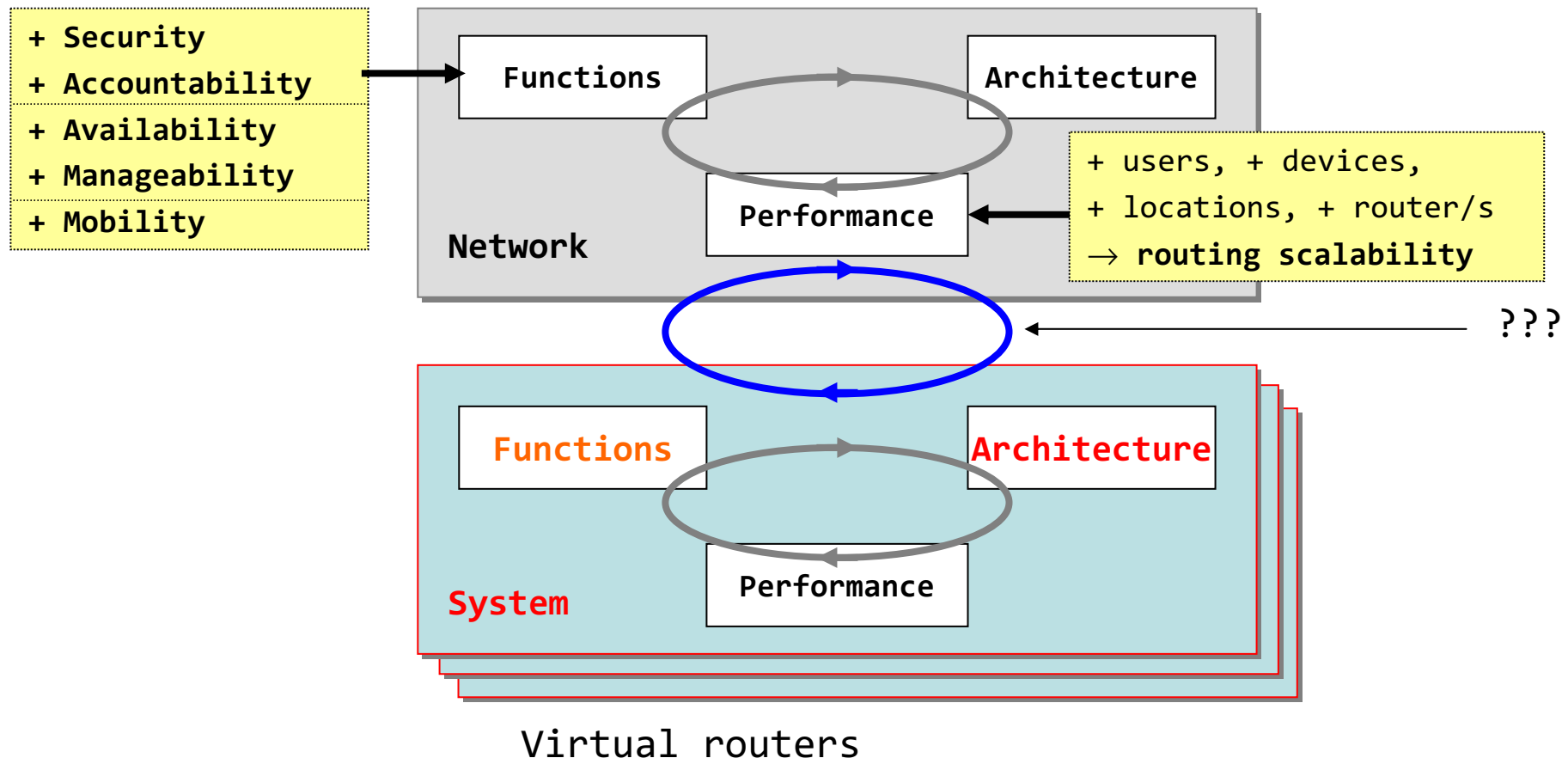
Which service points (Network edge or Customer edge) ?

- Performance metrics and criteria ?
- Boils down to ask: where is **location** of the Application-to-Network boundary (today in the terminal) ?

-> Moving this boundary poses technological challenges (as already explained) but also questions evolvability and security challenges

Role of Virtualization wrt Internet Challenges

- Internet challenges \equiv networking challenges



So...

- What does virtualization (is intended to) solve in practice in terms of networking/Internet challenges ?
- Question: is "flexibility" (better resource utilization) leading to possible answer for (some of) Internet network challenges ?

... or is the concept an "enabler" to address specific needs identified in the context of the Internet ?

In that case, the "design" becomes even more complex (objective functions) -> will require to address genericity and evolvability

Research Challenges

- Elaborate "problem statement" vs "needs"
 - Why are the intended "functionality" not resolvable with current Internet/where are the limitations ?
- Identify architectural and design goals (both at system AND network level)
 - Note: if the approach is inductive documenting set of representative use cases can be considered
- Identify architectural impact resulting from "virtualization" (at both system and network level)
 - Note: Detailed network modeling is still missing

Performance Challenges

Virtual "system" (e.g. virtual router)

- Optimization of the "routing systems" being executed through VMs while maintaining overall system performance
- Model (resource) requests in relation to VM configuration (parametric)
- Identify system behavior to predict load variation (-> dynamically adapt VM's capacity e.g. CPU, memory)
- Dynamic on-line provisioning, control, and performance monitoring of VMs in a distributed environment
- **Mathematical performance modeling & analysis** as well as numerical simulations **complementary** to experimentation (by means of emulation and prototypes)

Trade-offs

- Trade-off between efficiency, complexity and dynamics plays fundamental role in design
 - Statistics collection algorithms (monitoring)
 - Dynamics may affect confidence in statistical information
 - Confidence in statistical information affects efficiency: less accurate statistical information leads to e.g. oscillations effects (action/reaction chains)
- How to design "adequate" performance monitoring tools ?