serval-arch.org

Serval: An End-Host Stack for Service-Centric Networking

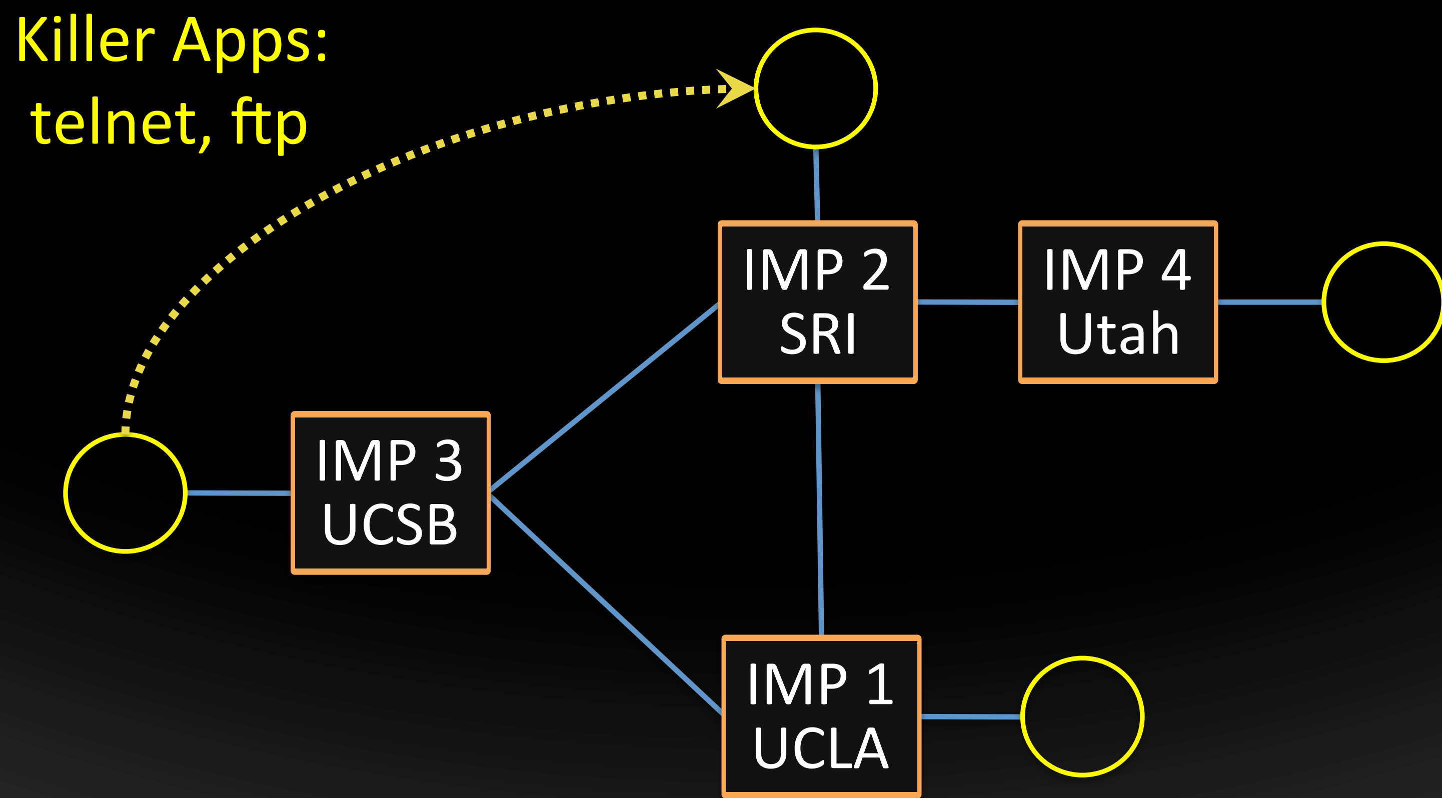
Erik Nordström

David Shue, Prem Gopalan, Rob Kiefer,

Mat Arye, Steven Ko, Jen Rexford, Mike Freedman

Princeton University

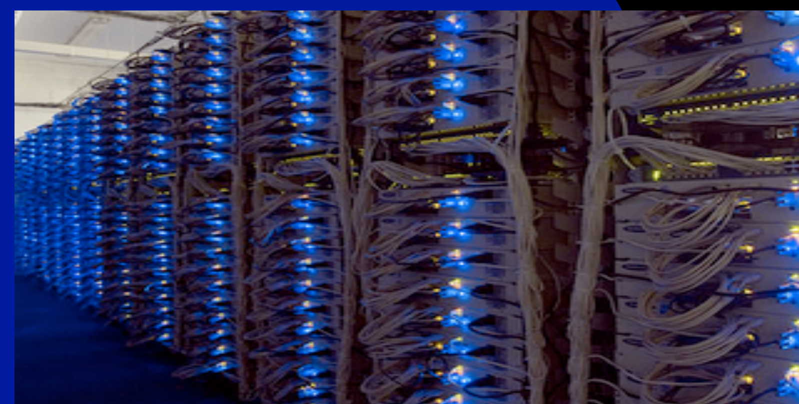
The Internet of the 1970s



Network designed for accessing **hosts**

The Internet of the 2000s

Datacenter



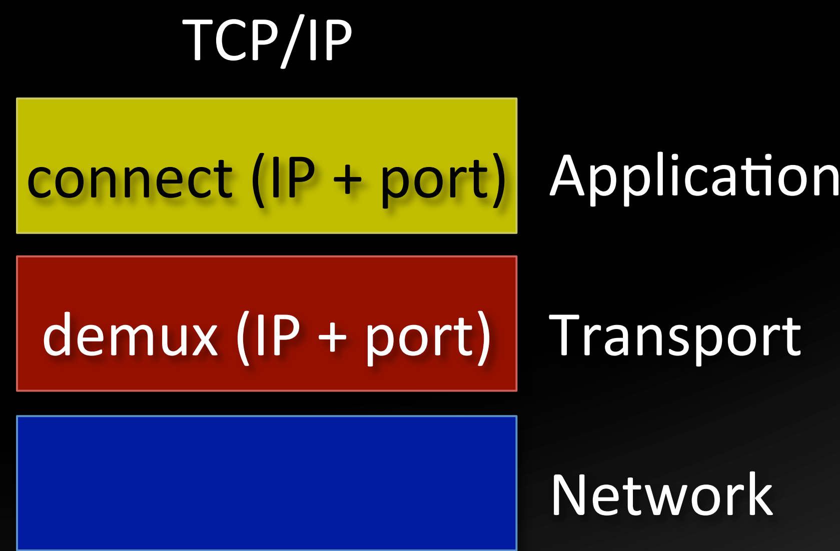
Users agnostic of actual service location and host

What does Service Access Involve?

1. Locate a nearby service datacenter
   – Map ***service*** name to location
2. Connect to service
   – Establish data ***flow*** to instance
   – Load balance between pool of replicas
3. Maintain connectivity to service
   – Migrate between interfaces and networks

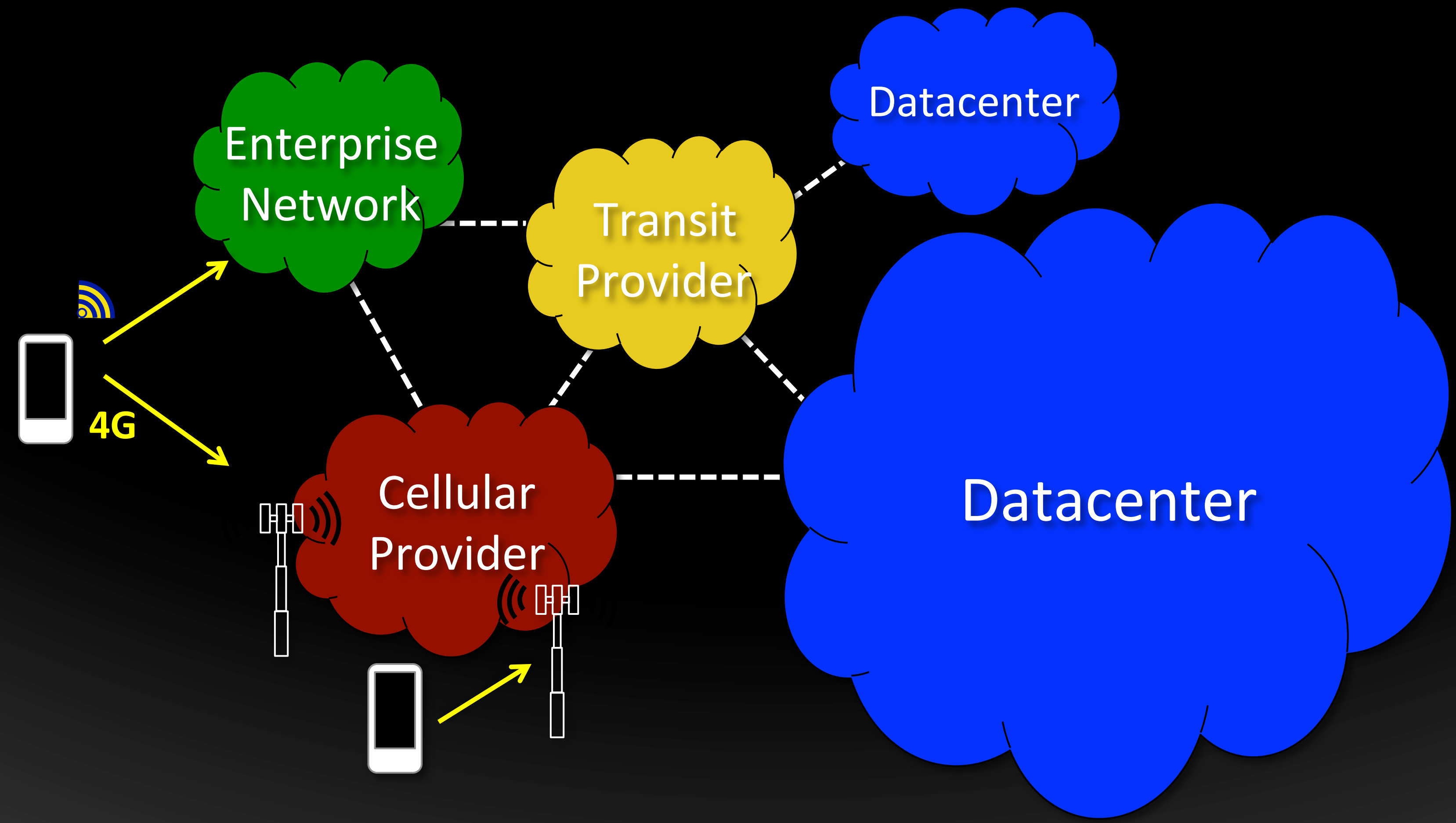
Today’s (Overloaded) Abstractions

- Service is IP + port
  - Exposes location
  - Specifies app. protocol
  - One service per IP

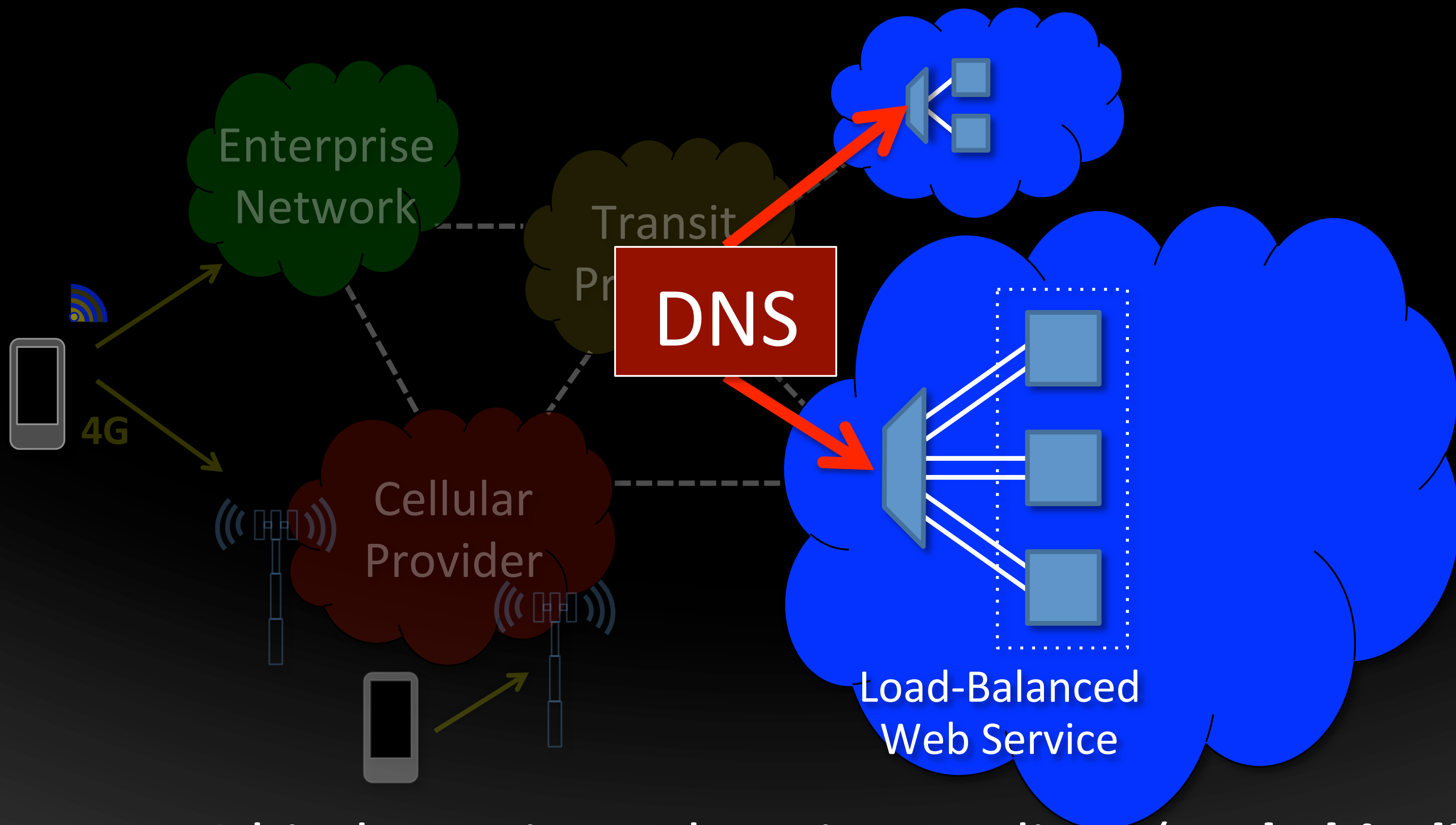


- Flow is “five tuple”
  - Binds flow to interface and location
  - Cannot migrate between interfaces or networks

Service Access Today

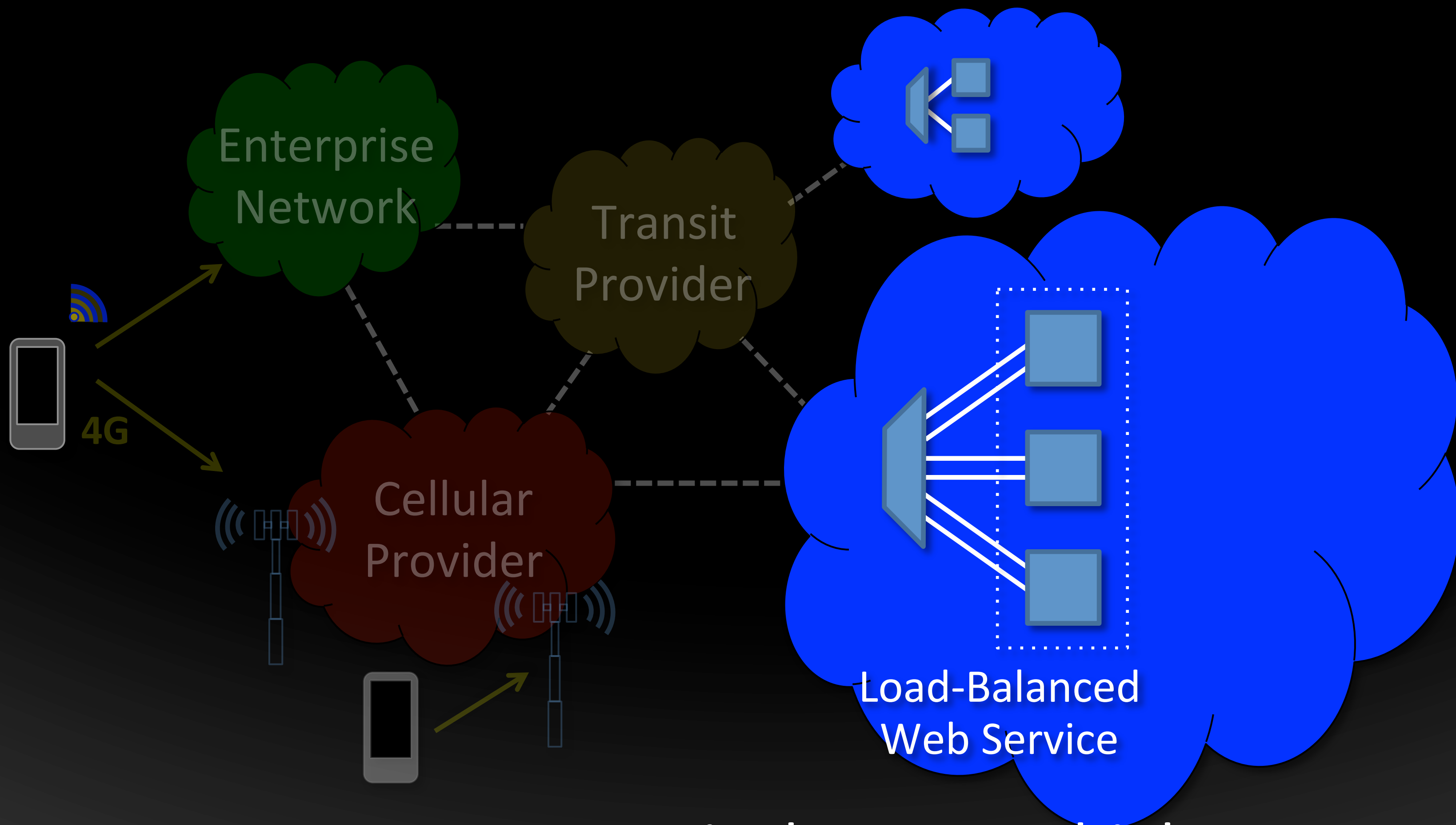


Finding a Service Location



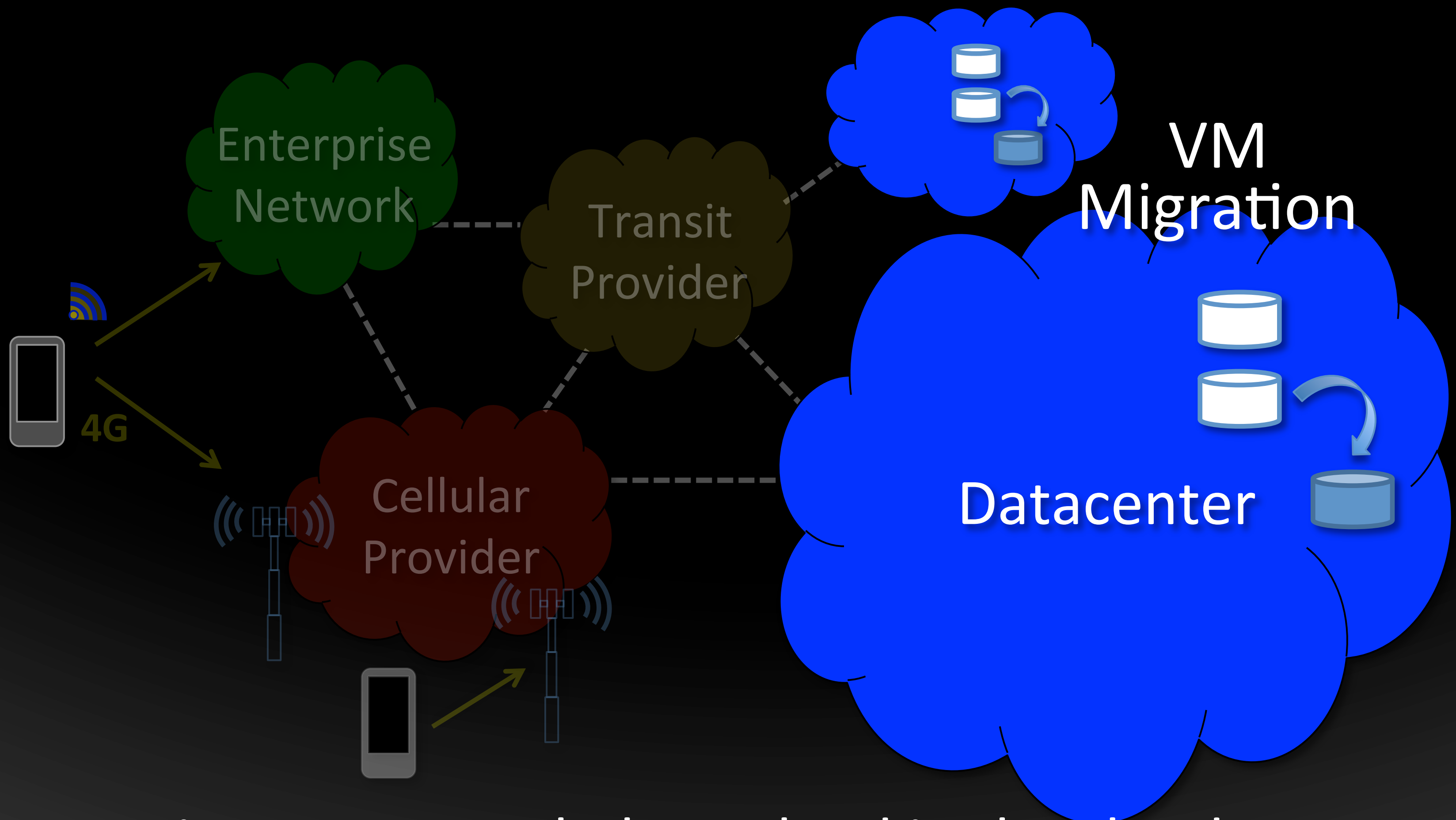
- DNS binds service to location at client (**early binding**)
  - Caching and ignoring TTL exacerbates the problem
  - Slow failover when instance or load balancer fail

Connecting to Service



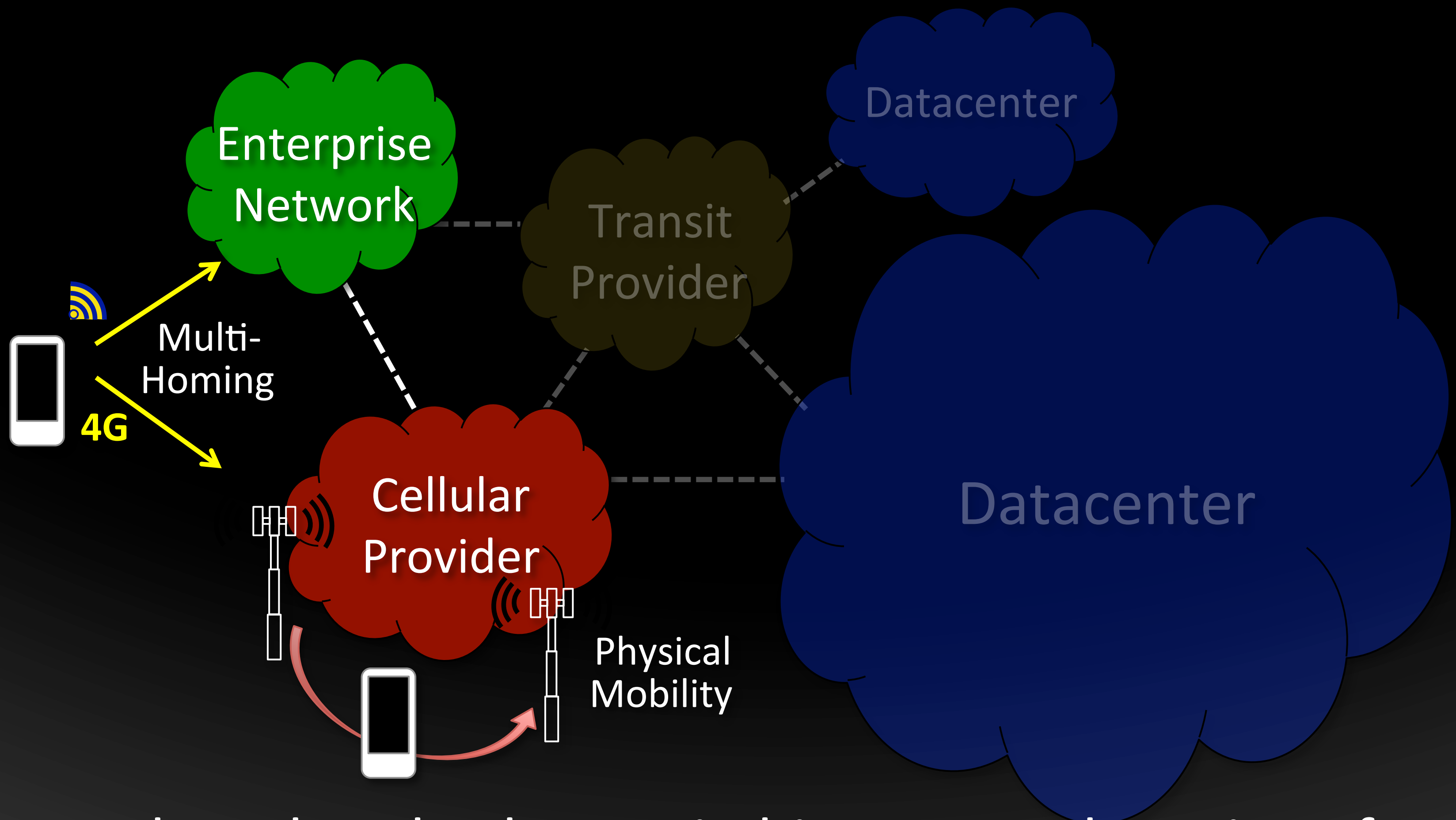
- Datacenter LB maps single IP to multiple servers
  - Must do this for every packet on path -> fate sharing
  - Increases complexity and cost

Maintaining Connectivity to Service



- Migrate VMs to balance load in the cloud
  - Requires flat addressing or tunneling within datacenter

Maintaining Connectivity to Service



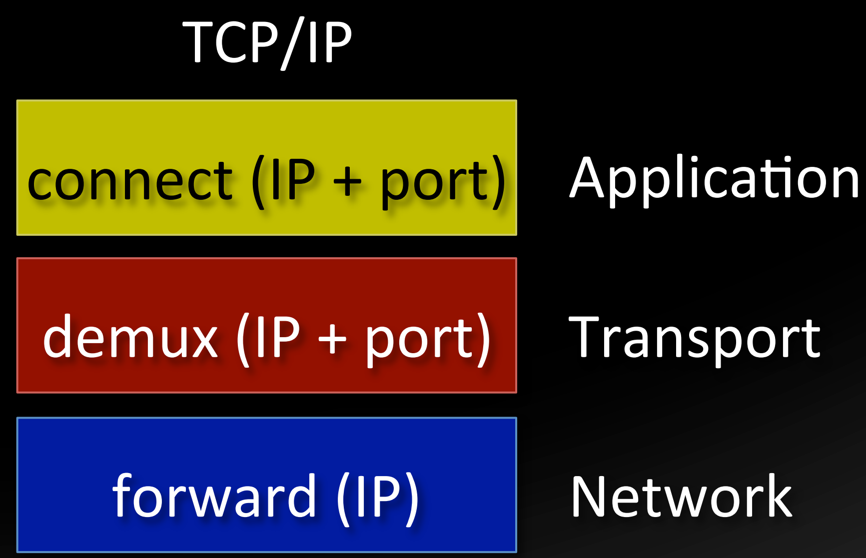
- Flows break when switching networks or interfaces

Contributions

- Naming abstractions
  - Services, flows
  - Clean role separation in the network stack
- Software architecture for services (Serval)
  - Service-level control/data plane split
  - Service-level events

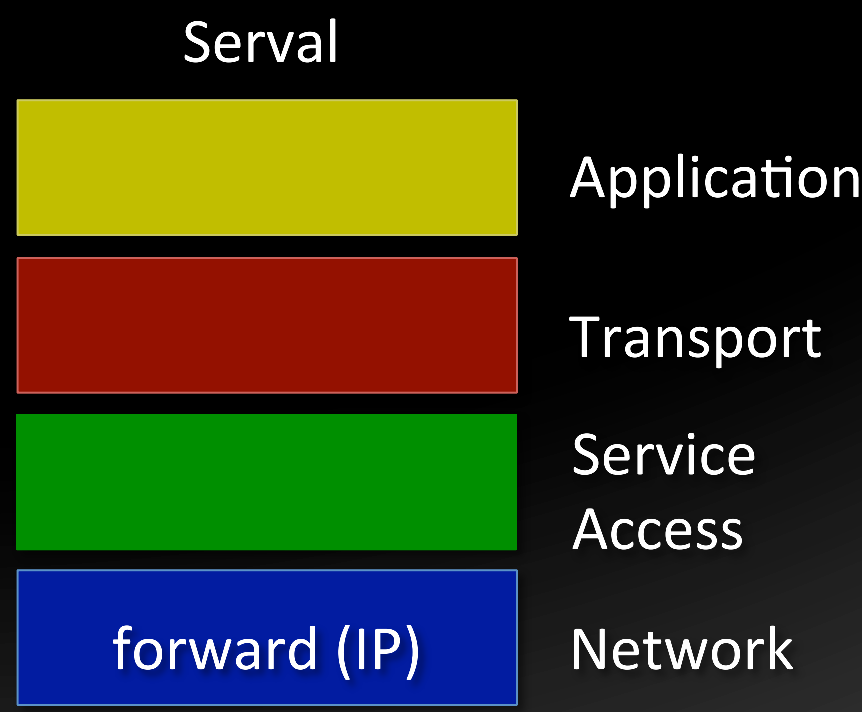
Naming Abstractions

Today's (Overloaded) Abstractions



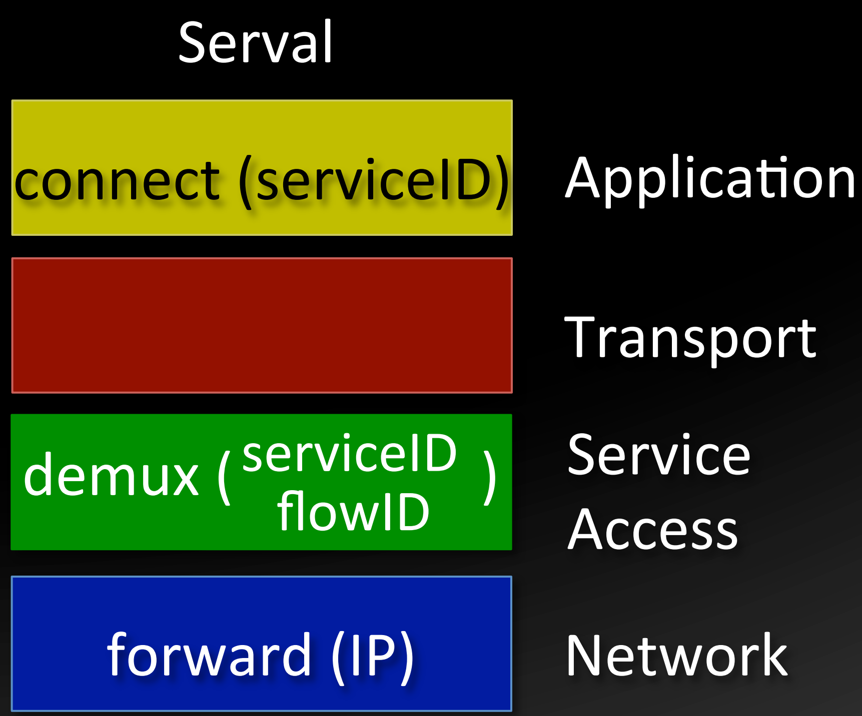
Serval Abstractions

- Serval cleans the slate
  - (But not completely)
- Network layer unmodified!
- Service Access Layer (SAL)
  - Connects to services
  - Maintains connectivity



Serval Abstractions

- Service = ServiceID
  - **Group** of processes with identical functionality
- Flow = FlowID
  - Invariant demux key
  - Host-local, ephemeral
- Location = IP address
  - Location, interface
  - Can change dynamically



A Clean Role Separation in the Stack

- What you access (**serviceID**), over which flows (**flowIDs**), and at which service instance (**IP address**)

	TCP/IP	Serval
Application	connect (IP + port)	connect (serviceID)
Transport	demux (IP + port)	
Service Access		demux (serviceID flowID)
Network	forward (IP)	forward (IP)

Service Names (ServiceIDs)

Provider prefix	Provider-specific	Self-certifying

- ServiceIDs allocated in blocks
  - Prefix ensures global uniqueness
  - Prefix-based aggregation and LPM
- A ServiceID late binds to service instance
  - ServiceID in *first* packet of connection
  - Service-level routing and forwarding

A Service-Aware Network Stack

connect(sock, **serviceID**)

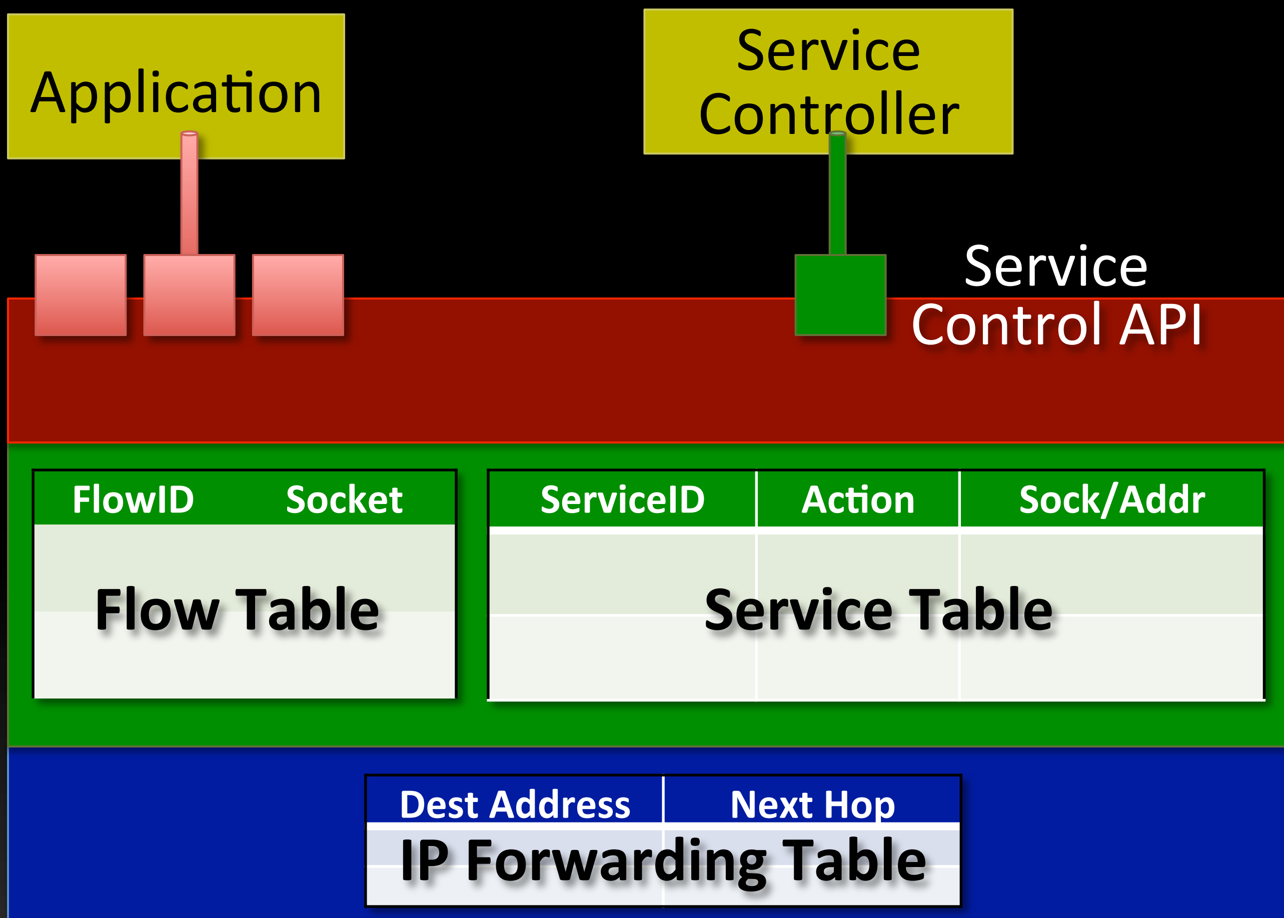
bind(sock, **serviceID**)
listen(sock)

Network stack must *resolve* service to instance for *client*

Network stack must *advertise* service for *server*

Software Architecture

Serval End-host Architecture



Data Plane: The Service Table

ServiceID	Action	Rule State
Prefix A	FORWARD	Send to addr A1
Prefix B	FORWARD	Send to [A2, A3, A4]
Prefix C	DEMUX	Send to listening sock *s*
Prefix D	DELAY	Queue and notify service controller
Prefix E	DROP	
default	FORWARD	Send to A5

Data Plane: The Service Table

ServiceID	Action	Rule State
Prefix A	FORWARD	Send to addr A1
Prefix B	FORWARD	Send to [A2, A3, A4]
Prefix C	DEMUX	Send to listening sock *s*
Prefix D	DELAY	Queue and notify service controller
Prefix E	DROP	
default	FORWARD	Send to A5

Data Plane: The Service Table

ServiceID	Action	Rule State
Prefix A	FORWARD	Send to addr A1
Prefix B	FORWARD	Send to [A2, A3, A4]
Prefix C	DEMUX	Send to listening sock *s*
Prefix D	DELAY	Queue and notify service controller
Prefix E	DROP	
default	FORWARD	Send to A5

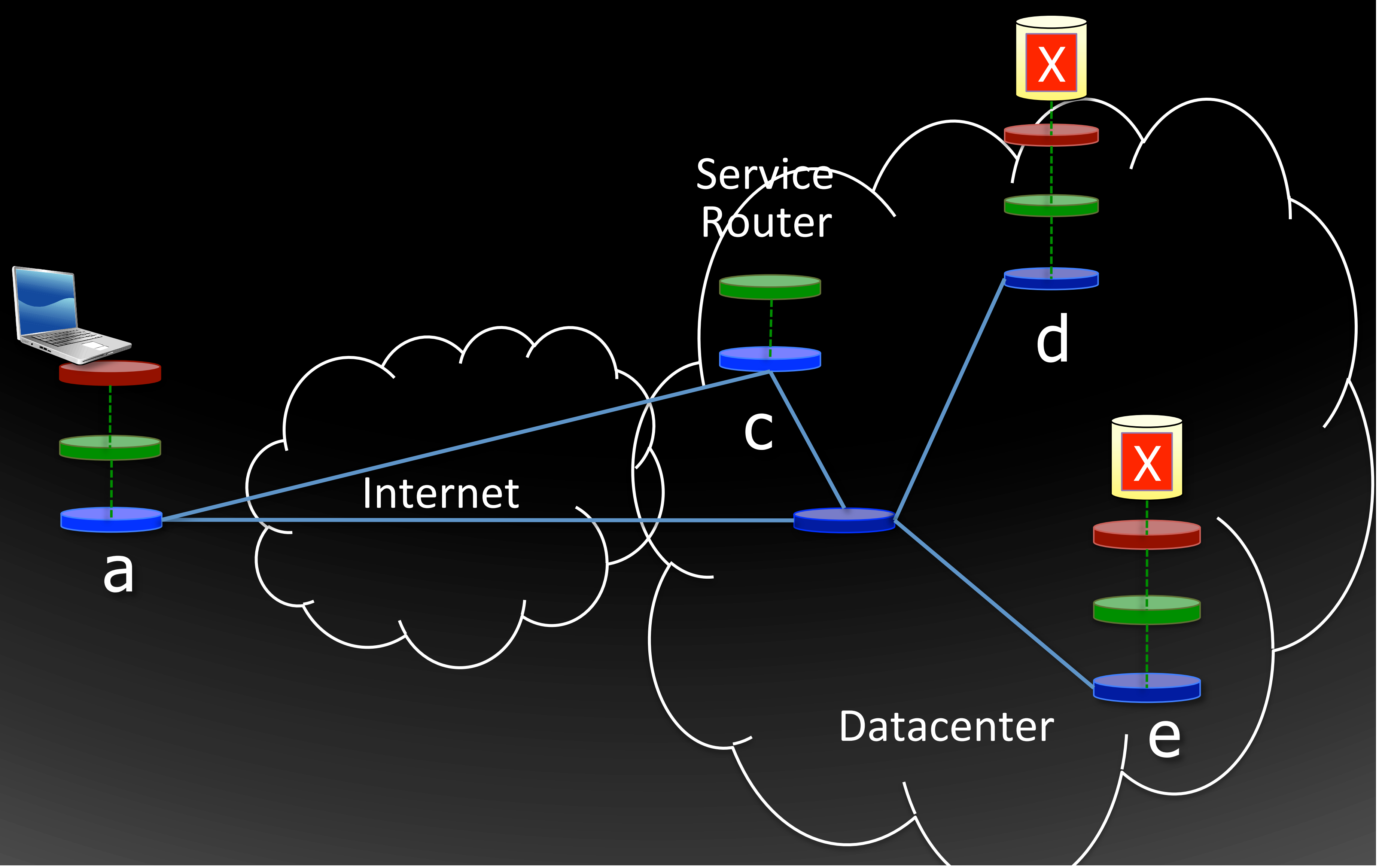
Data Plane: The Service Table

ServiceID	Action	Rule State
Prefix A	FORWARD	Send to addr A1
Prefix B	FORWARD	Send to [A2, A3, A4]
Prefix C	DEMUX	Send to listening sock *s*
Prefix D	DELAY	Queue and notify service controller
Prefix E	DROP	
default	FORWARD	Send to A5

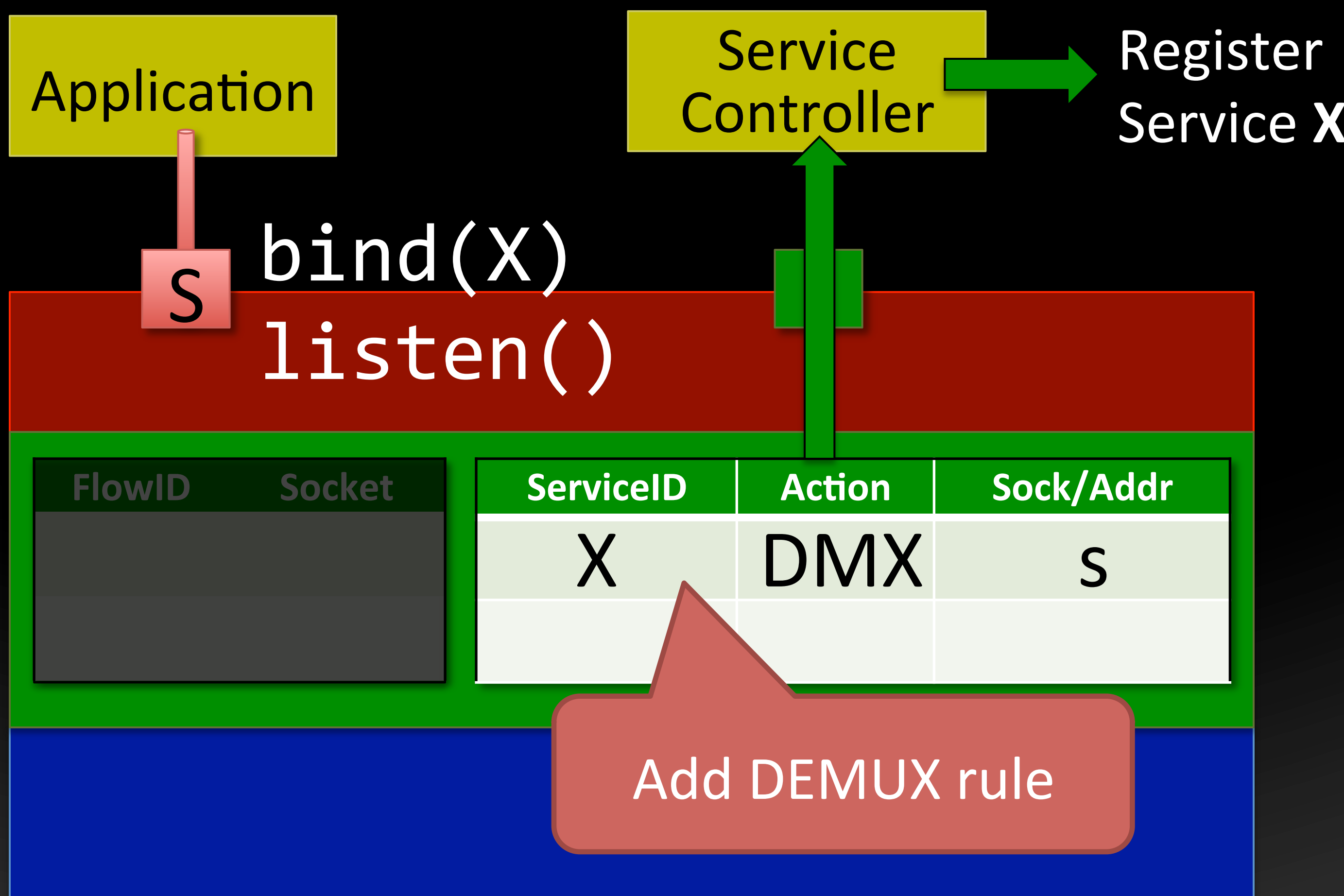
Data Plane: The Service Table

ServiceID	Action	Rule State
Prefix A	FORWARD	Send to addr A1
Prefix B	FORWARD	Send to [A2, A3, A4]
Prefix C	DEMUX	Send to listening sock *s*
Prefix D	DELAY	Queue and notify service controller
Prefix E	DROP	
default	FORWARD	Send to A5

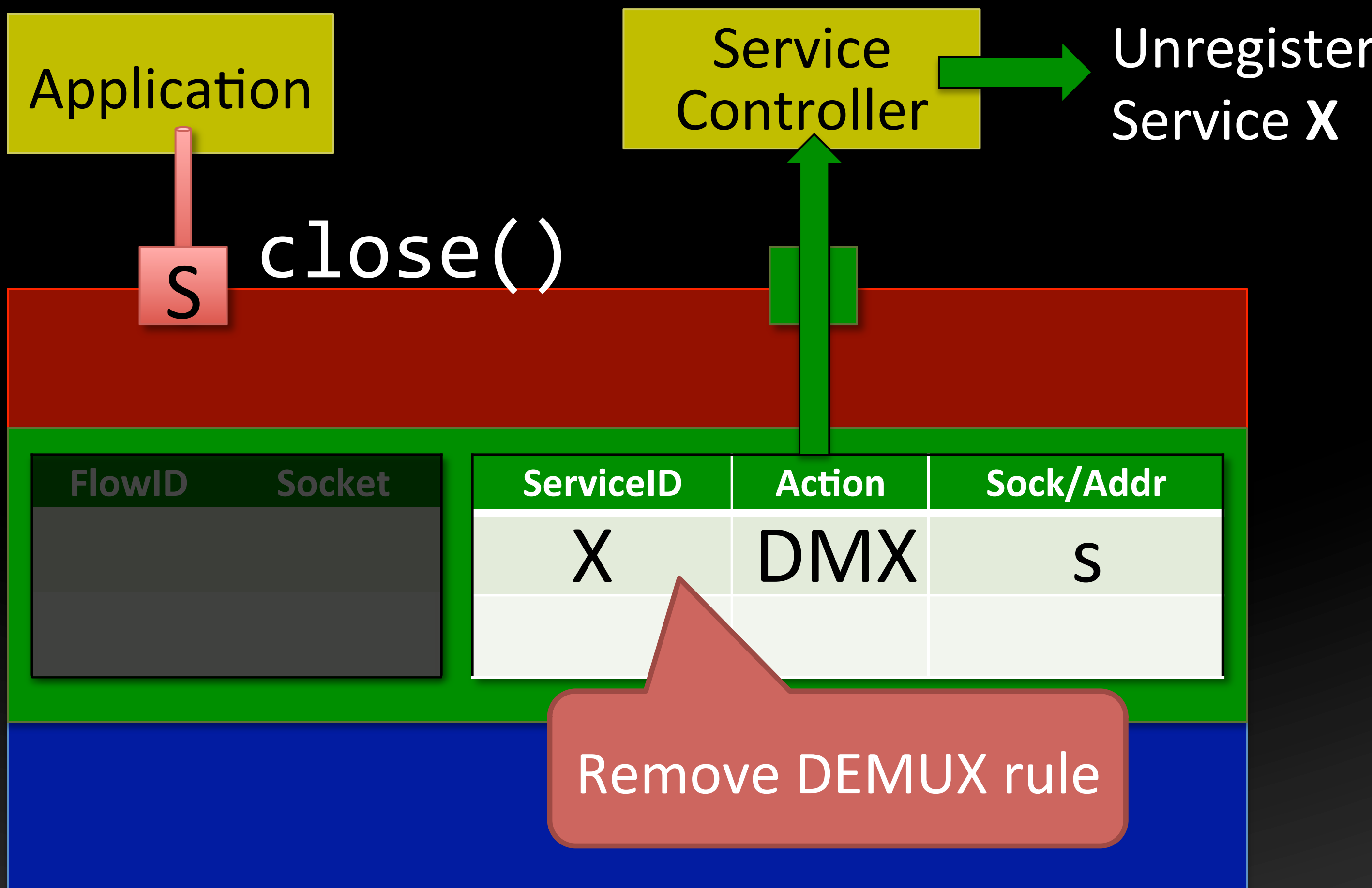
Service Access with Serval



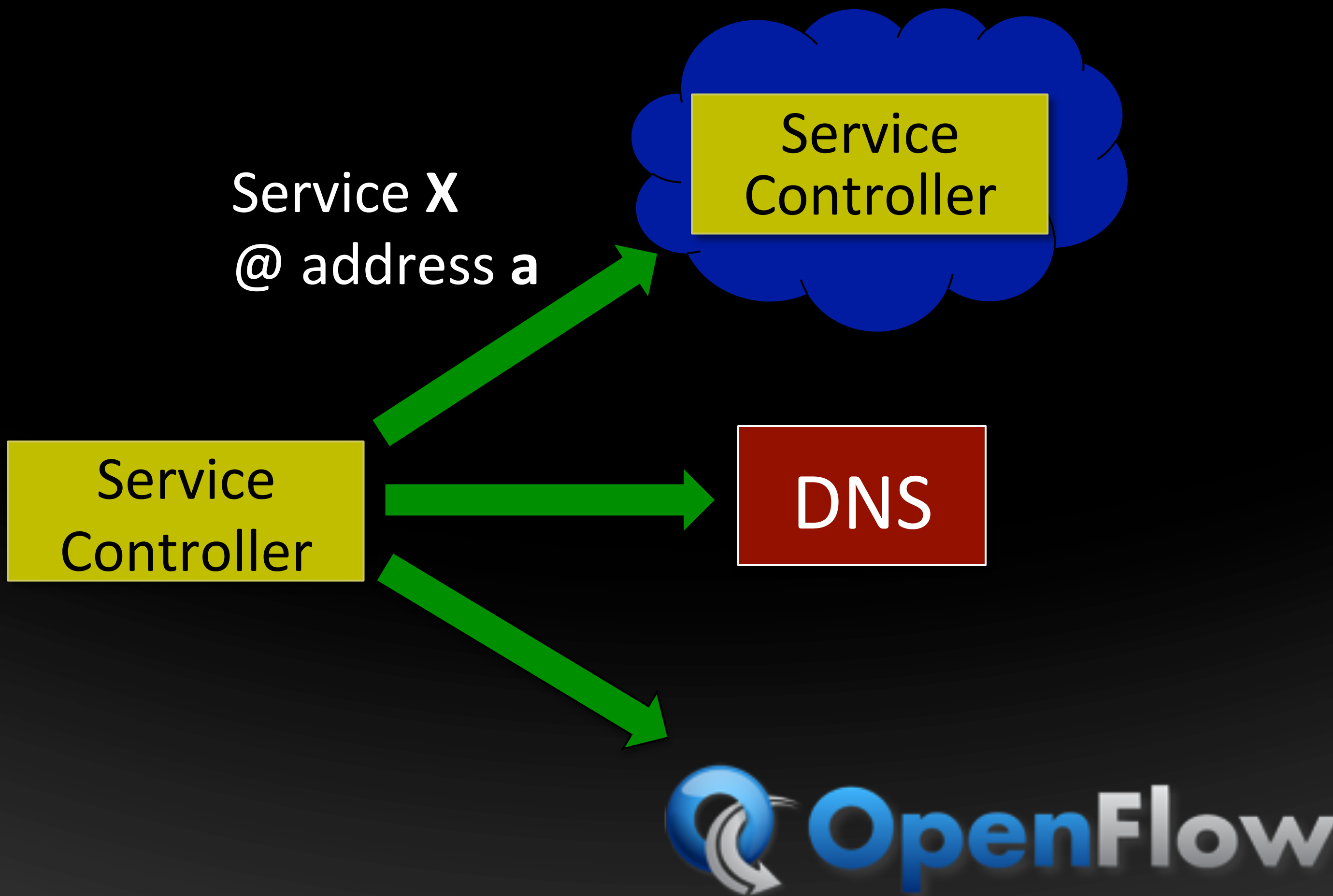
Adding a Service Instance



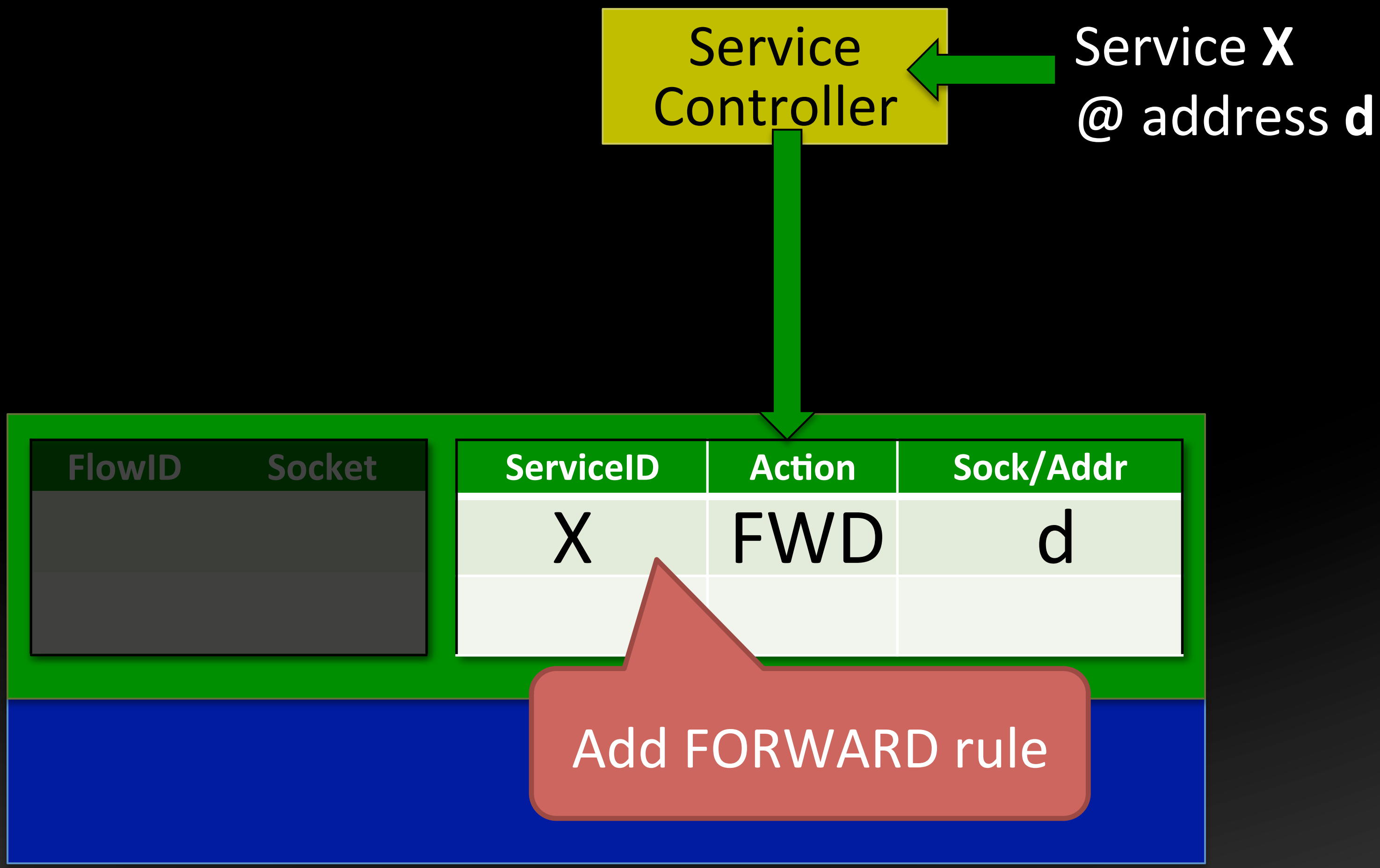
Removing a Service Instance



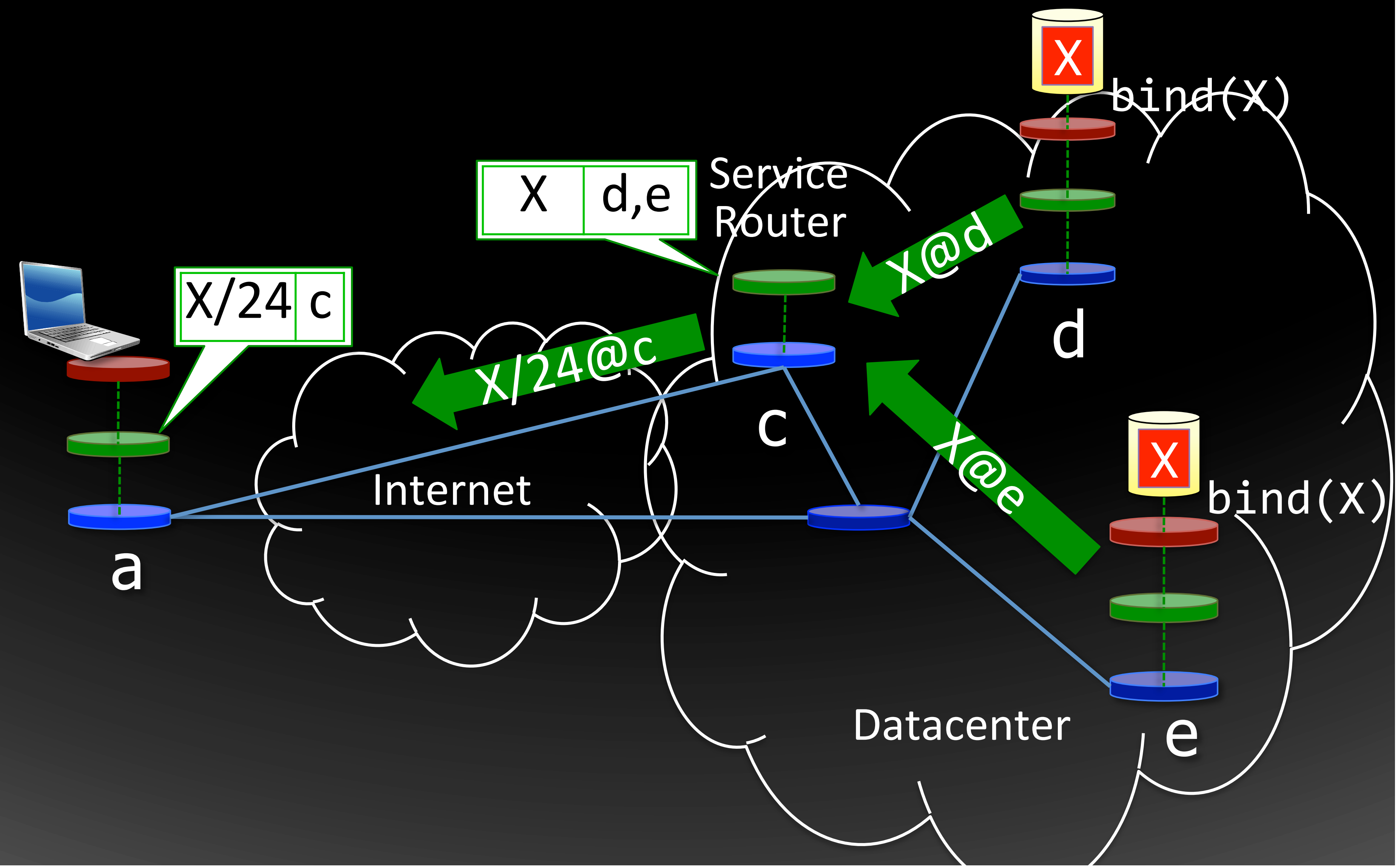
Control Plane: The Service Controller



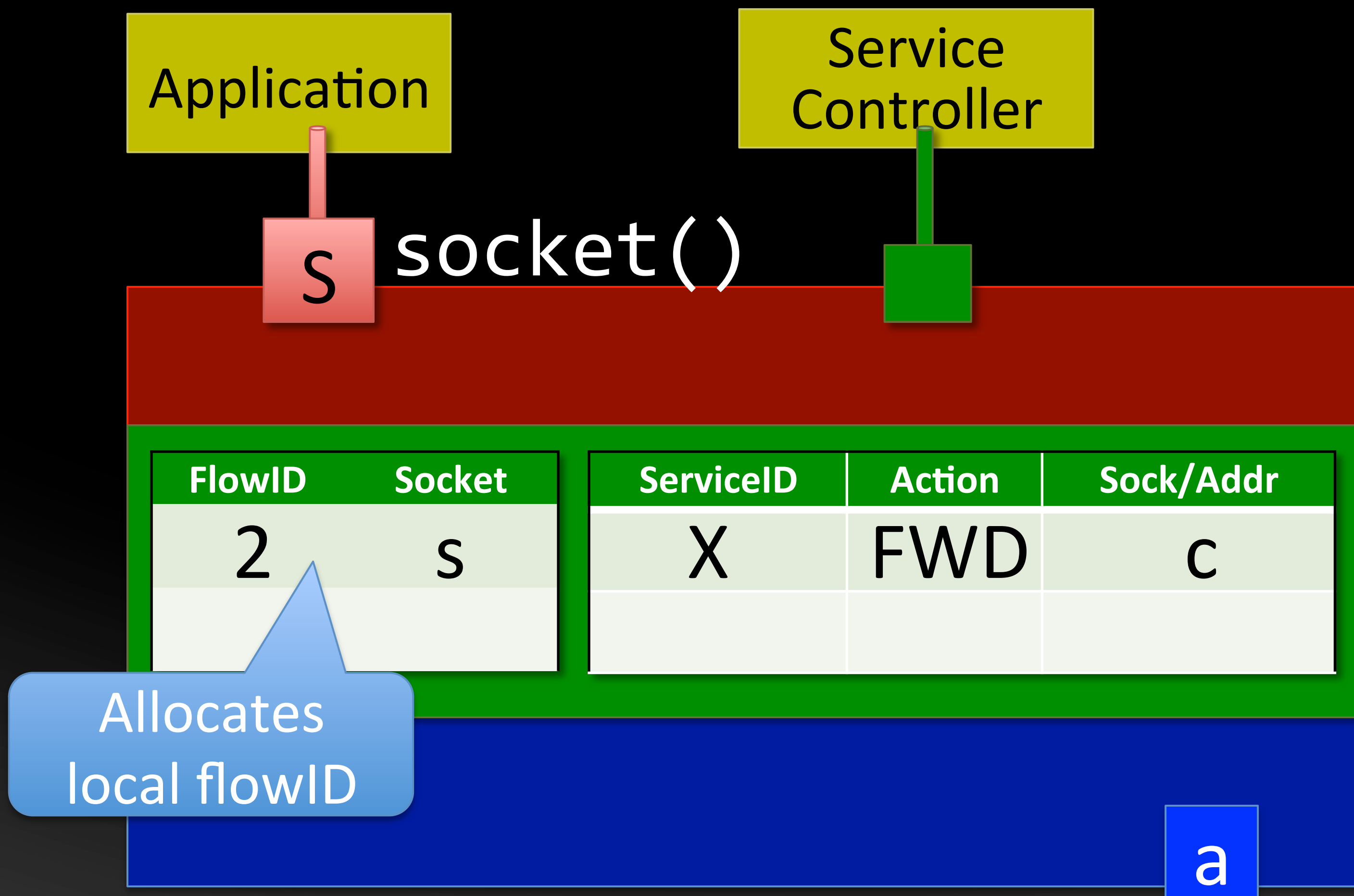
Control Plane: The Service Controller



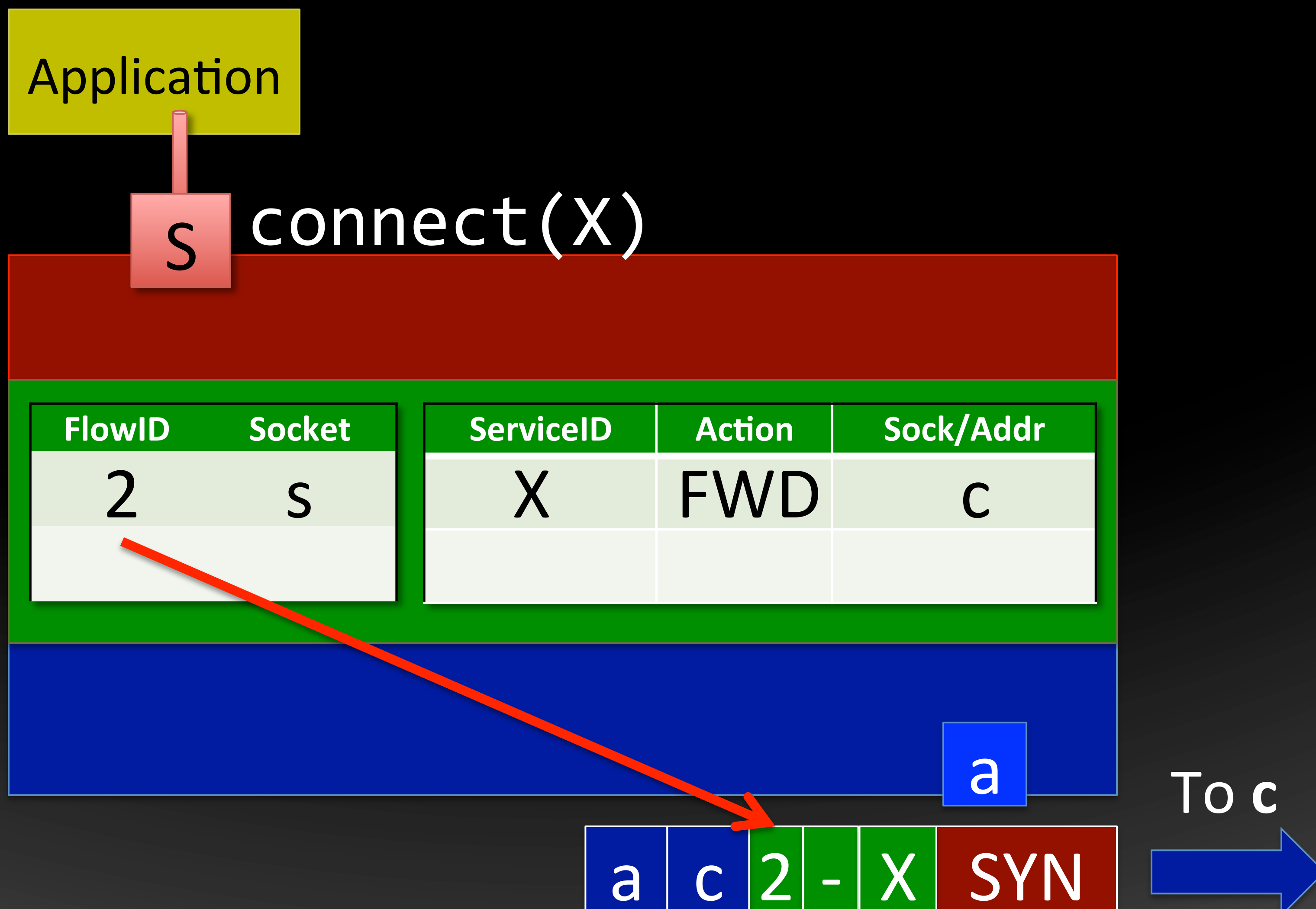
Service Access with Serval



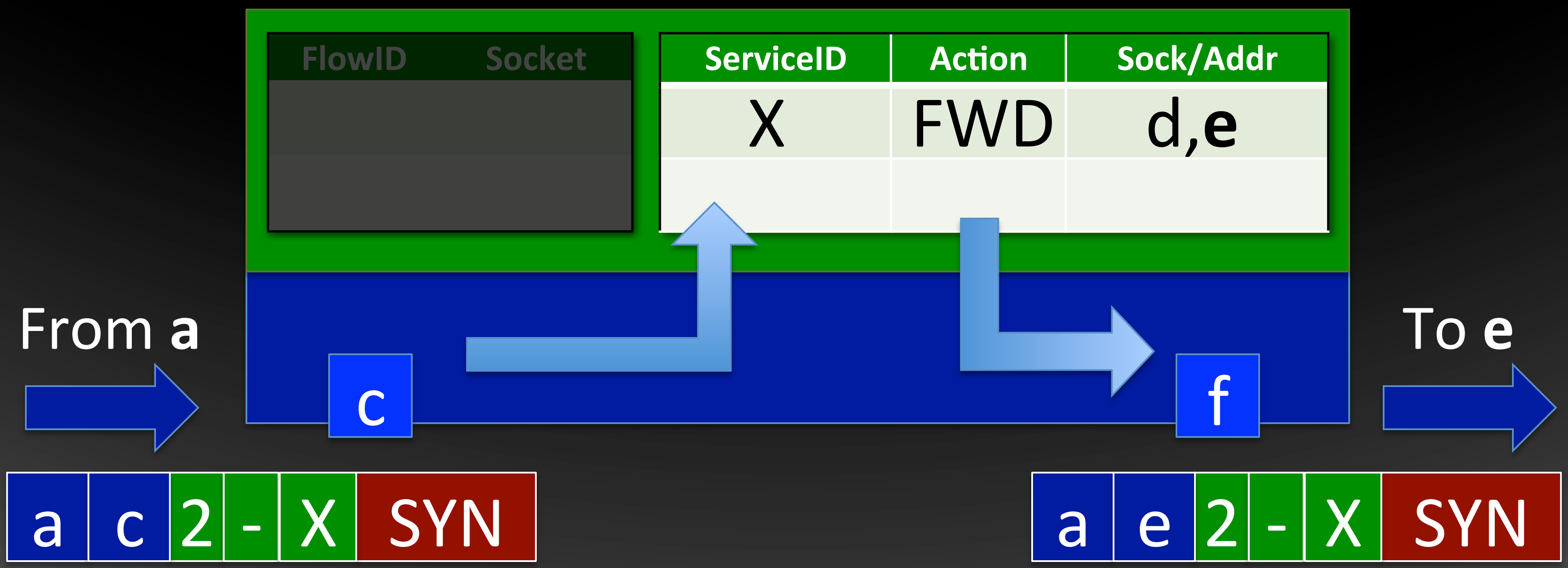
Connecting to Service X



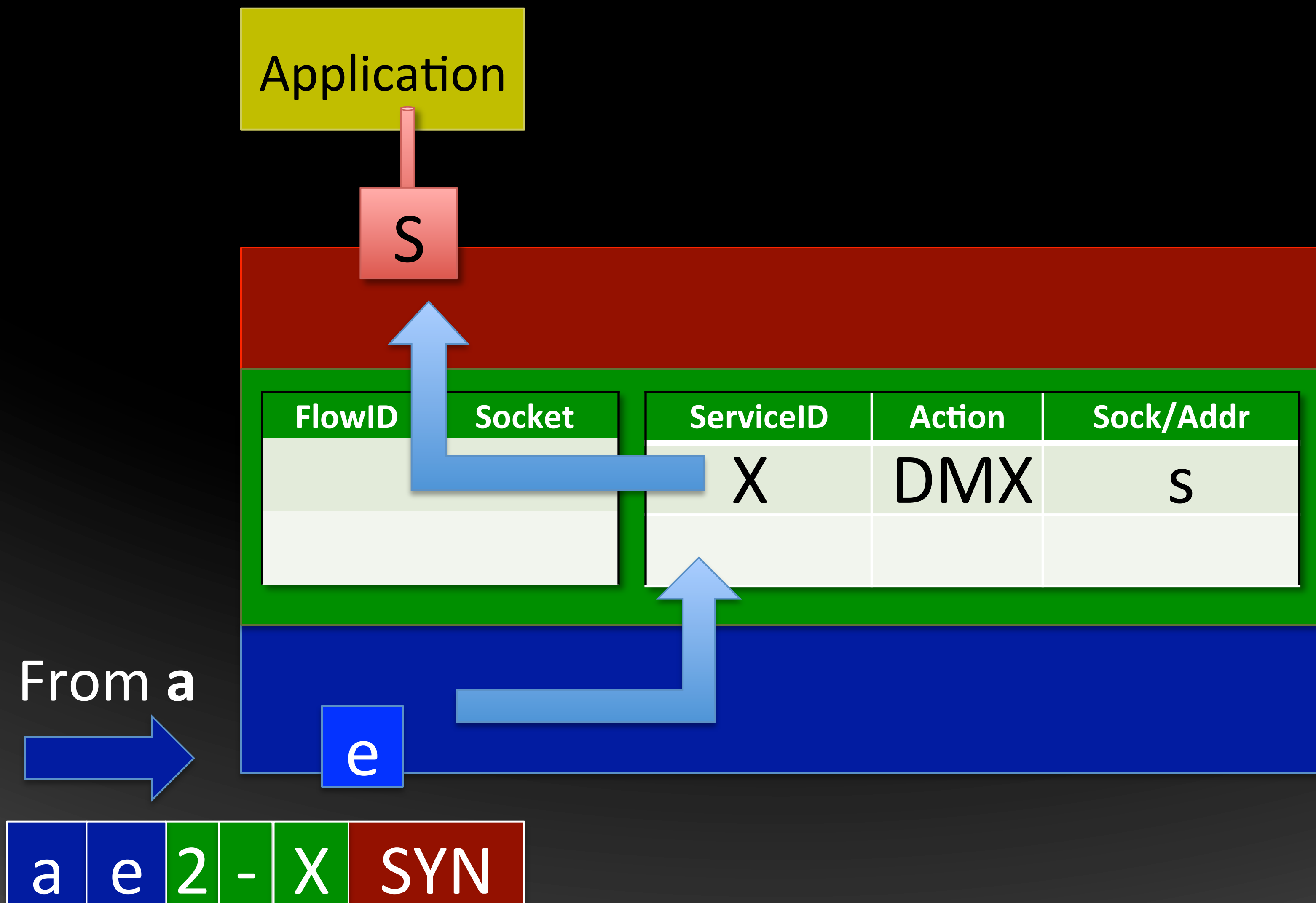
Connecting to Service X



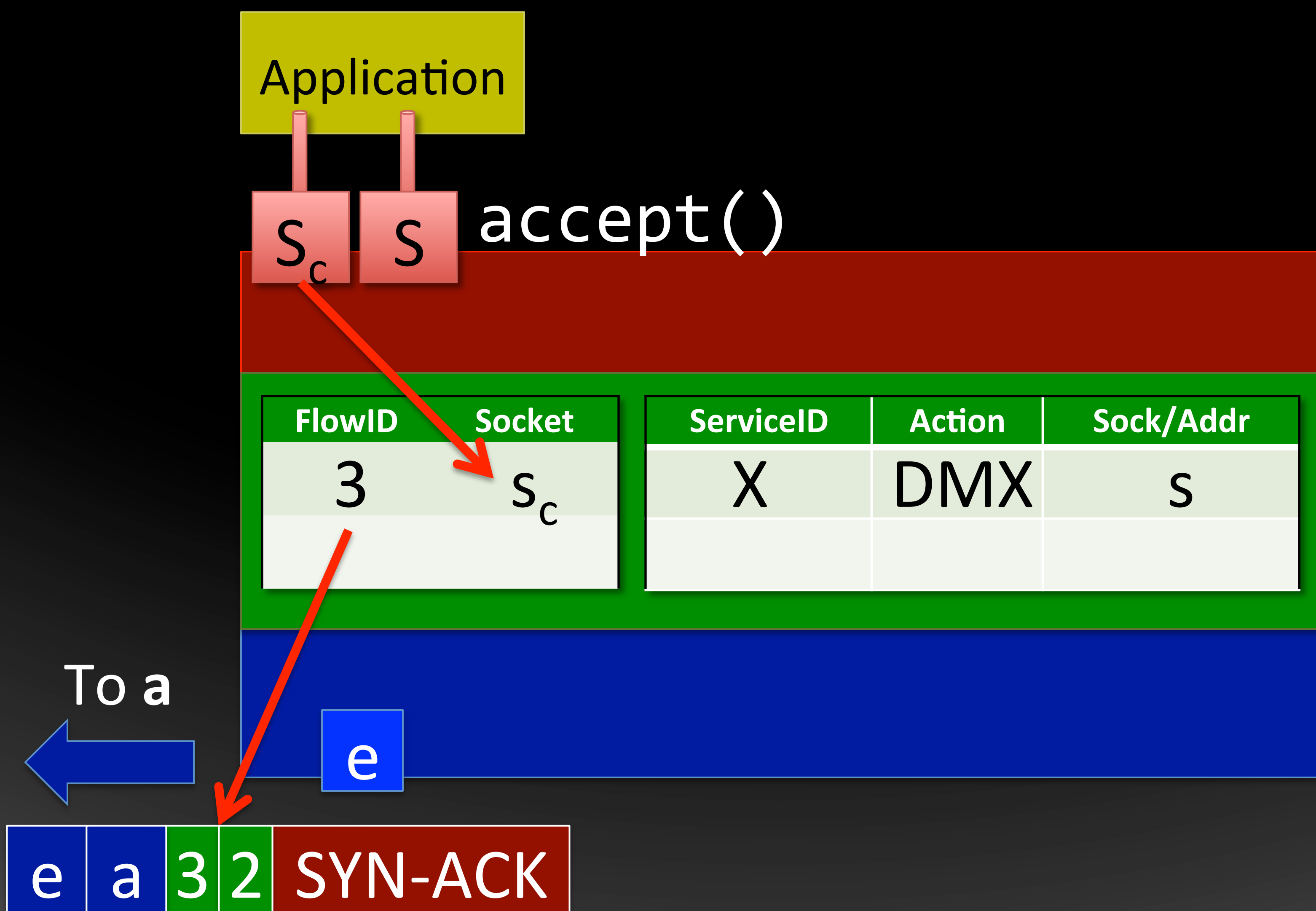
Load Balancing in Service Router



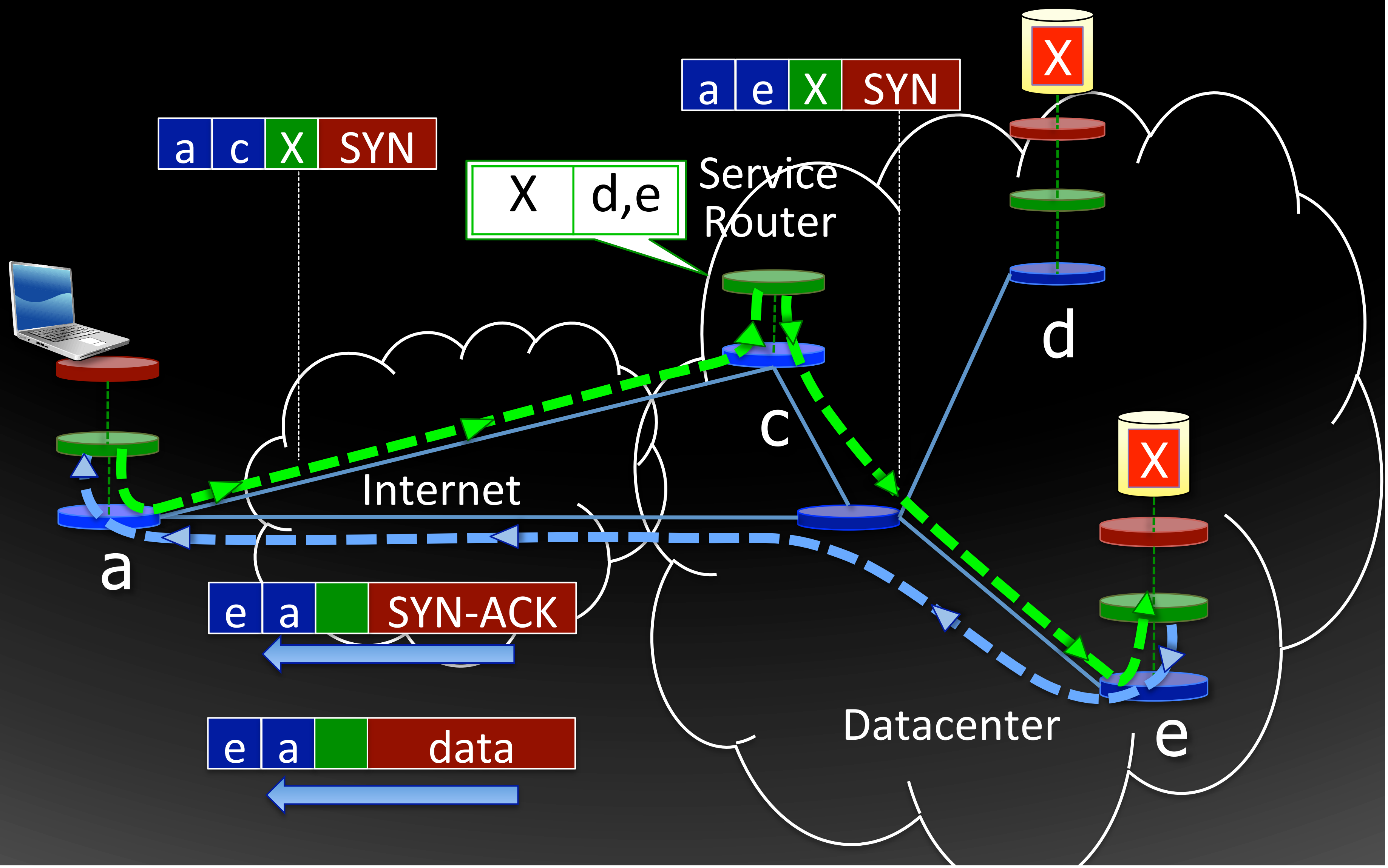
Service Instance Providing Service X



Service Instance Providing Service X

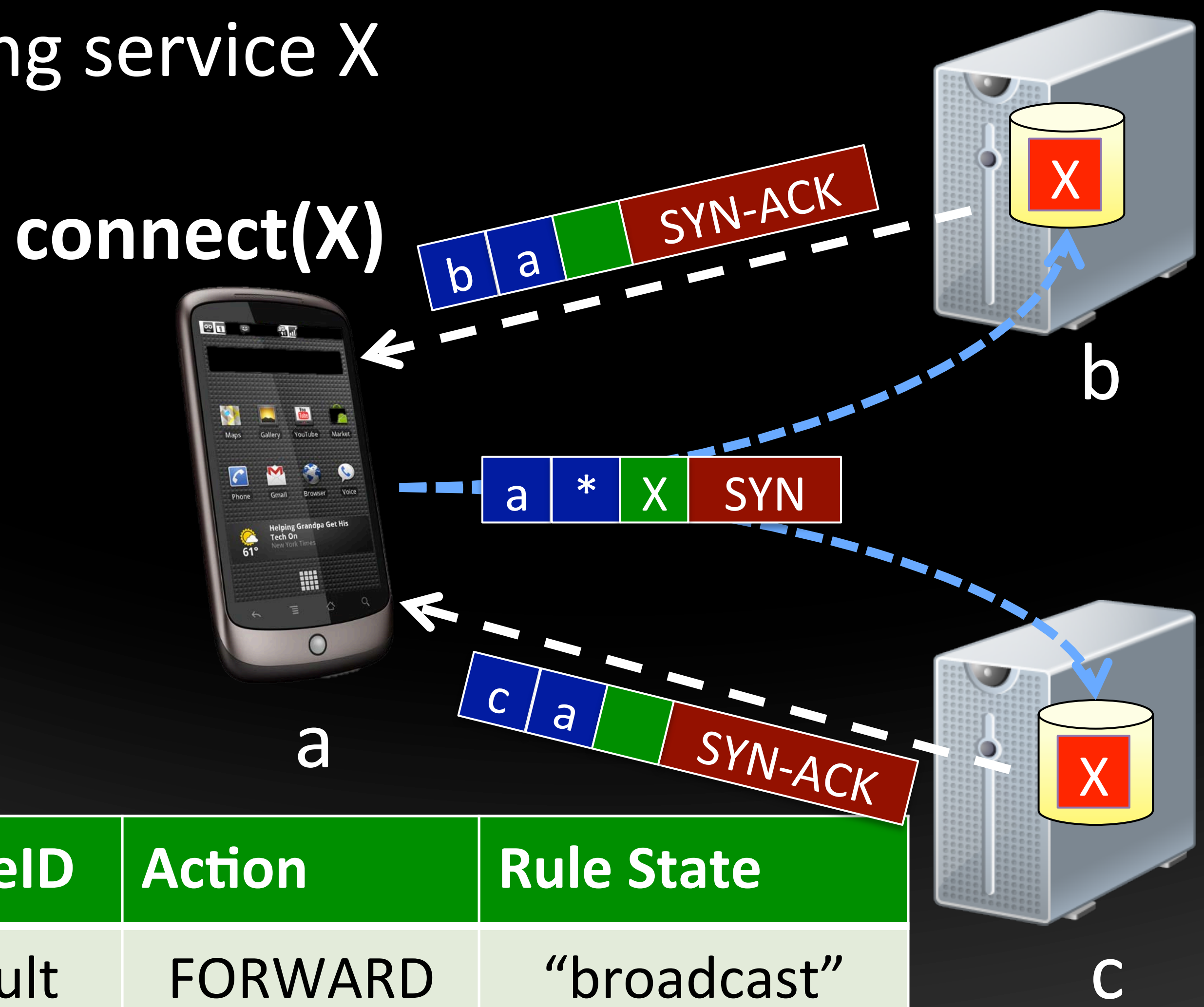


Service Access with Serval



Ad hoc Service Discovery

Accessing service X

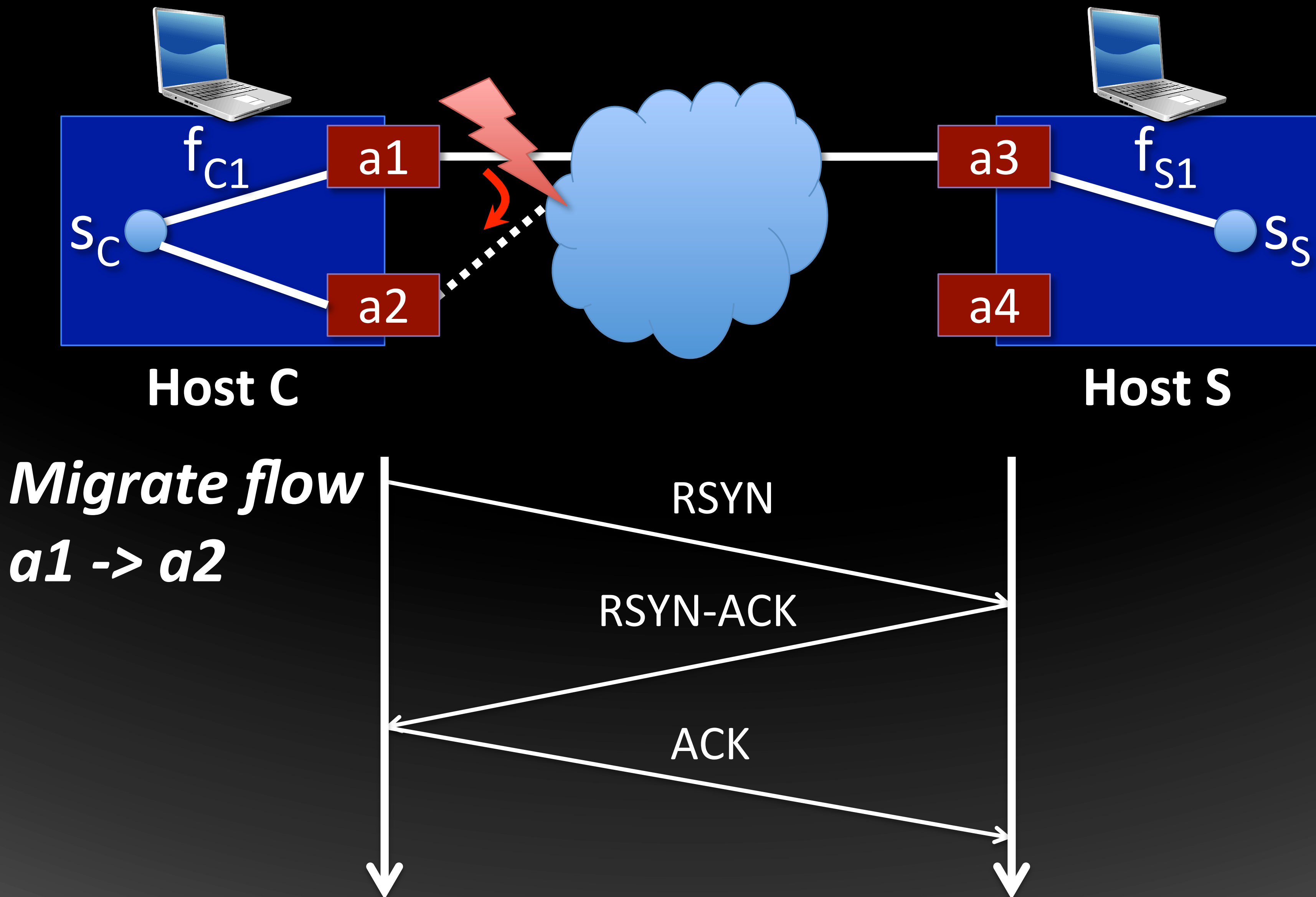


ServiceID	Action	Rule State
default	FORWARD	“broadcast”

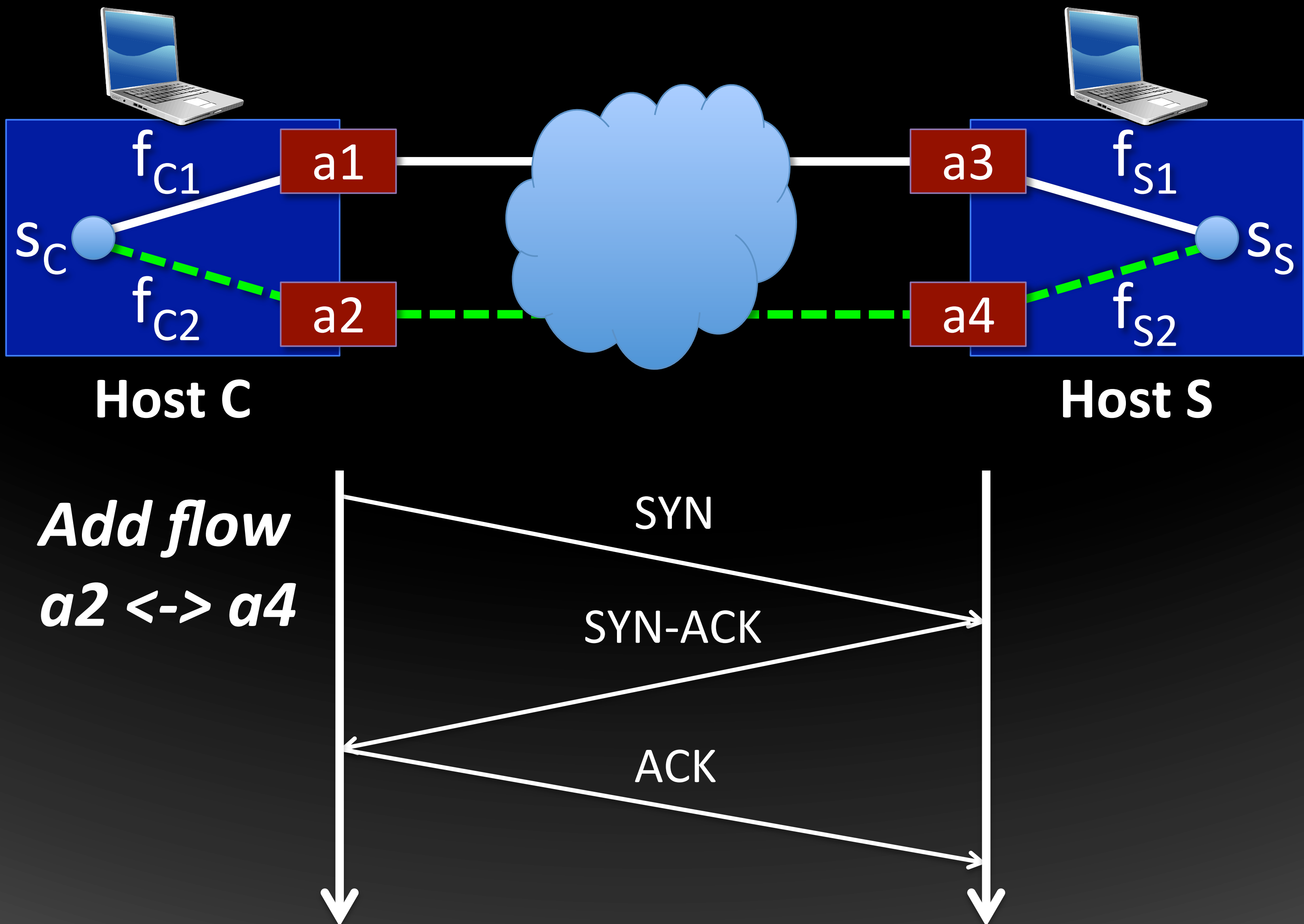
What does Service Access Involve?

1. Locating a nearby service datacenter
   – Map ***service*** name to location
2. Connecting to service
   – Establish data ***flow*** to instance
   – Load balance between pool of replicas
3. Maintaining connectivity to service
   – Migrate between interfaces and networks

Migration of Flows



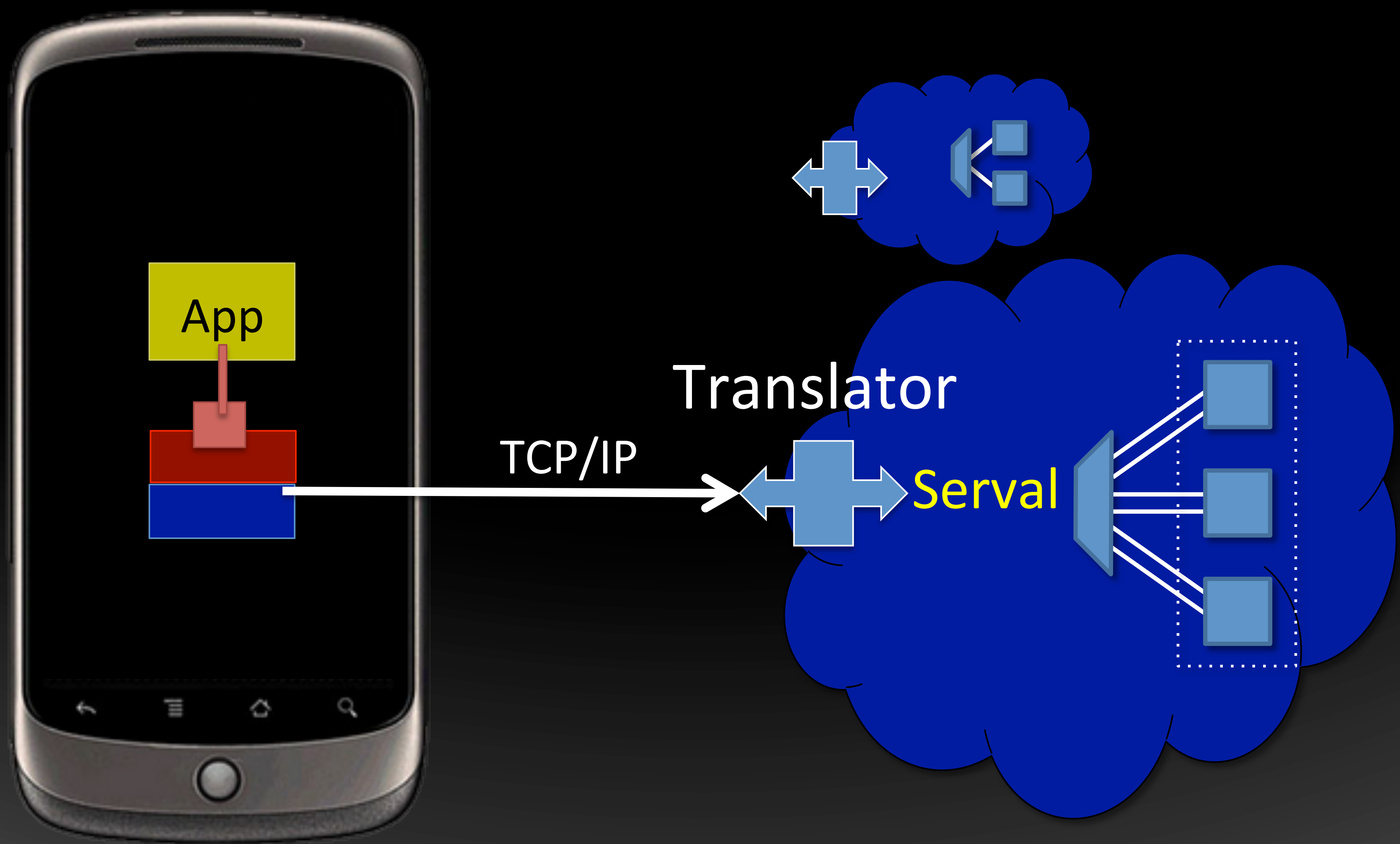
Multipath with Multiple Subflows



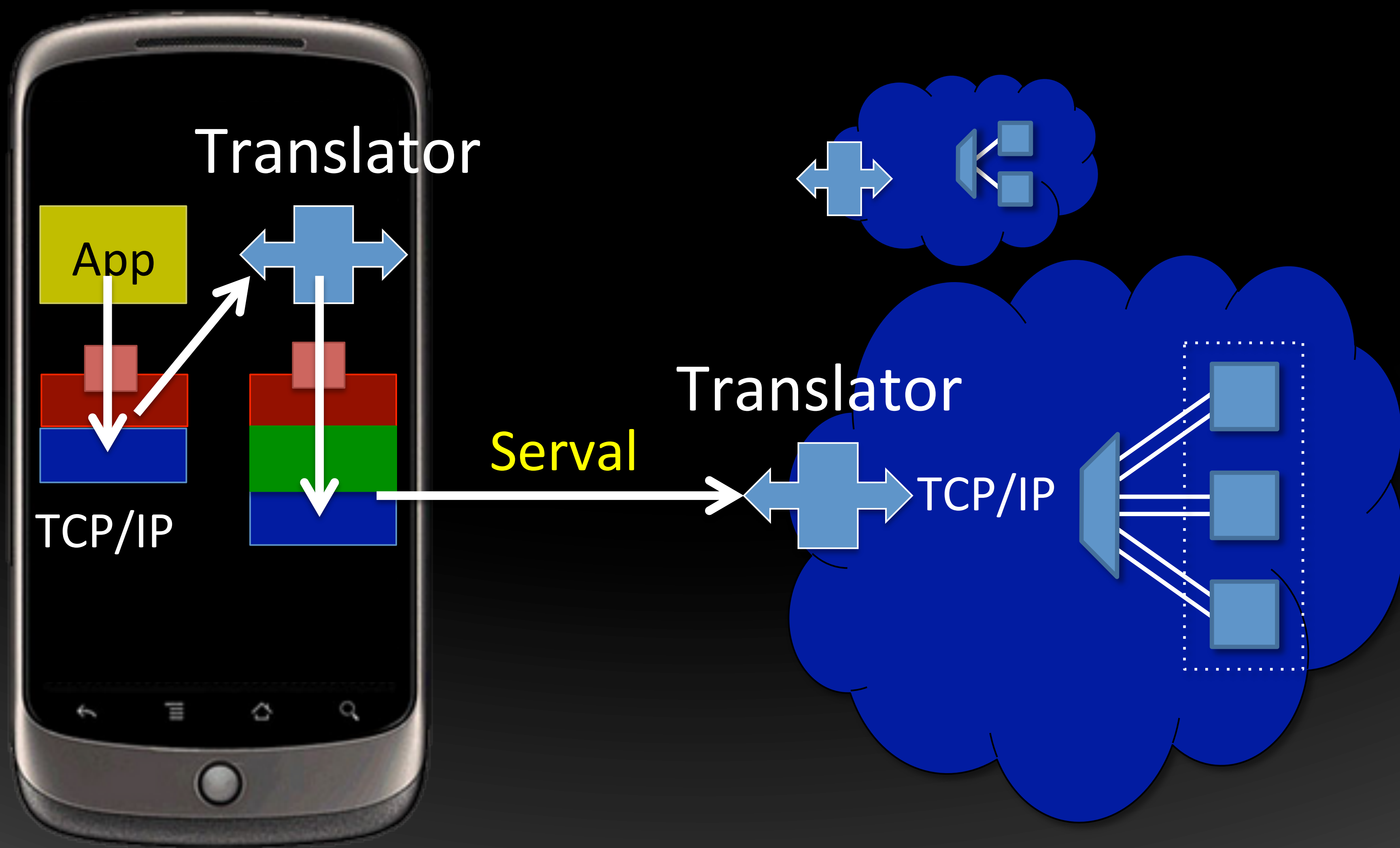
Prototype

- End-host network stack (28,000 LOC)
  - Linux kernel module
  - BSD sockets with AF_SERVAL protocol family
  - AF_INET sockets can be accessed simultaneously
- Legacy middleboxes / NATs handled via encap.
- Translator for incremental deployment
  - Unmodified apps and end-hosts
  - Serval apps with unmodified services

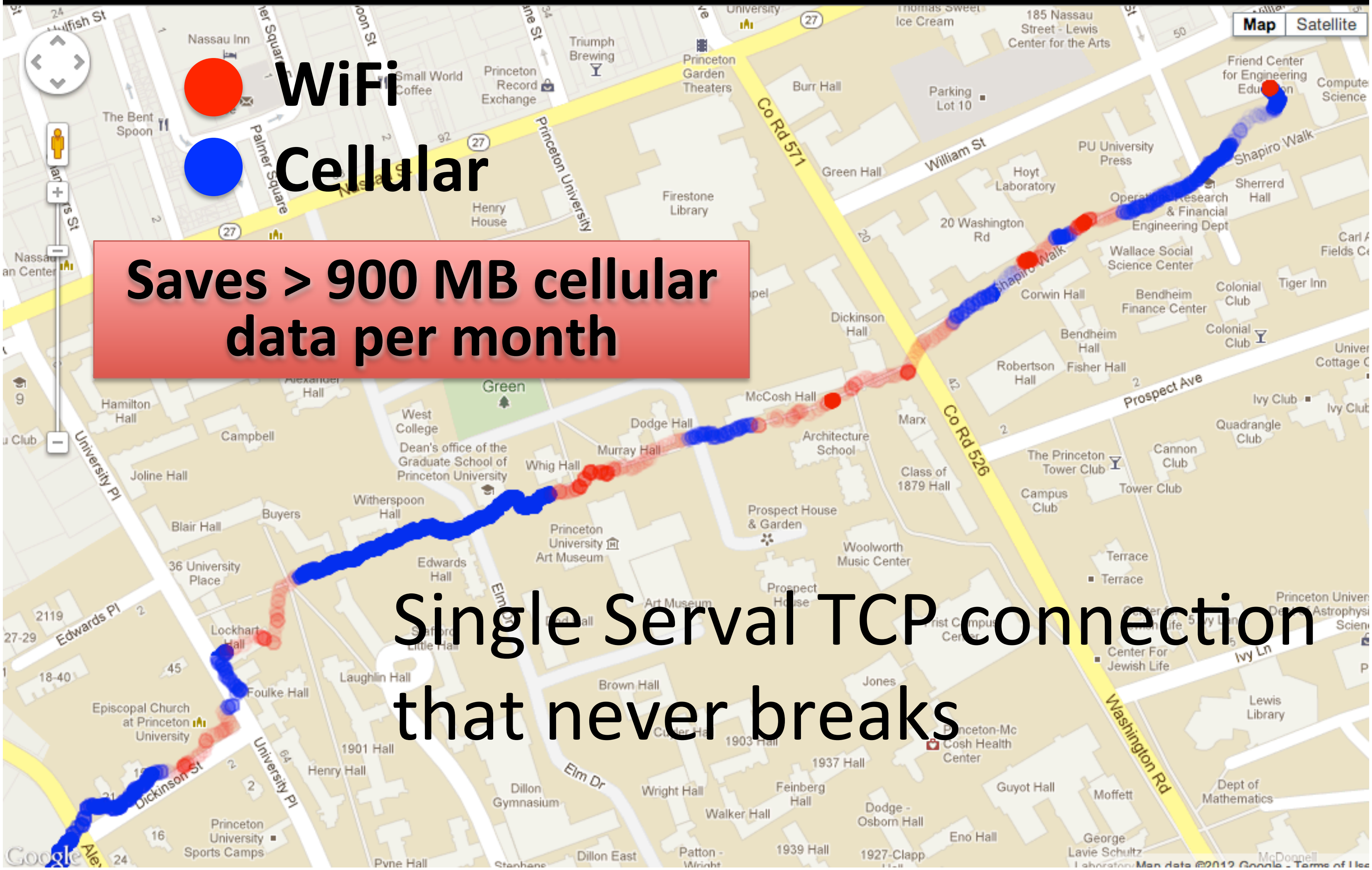
Incremental Deployment



Incremental Deployment



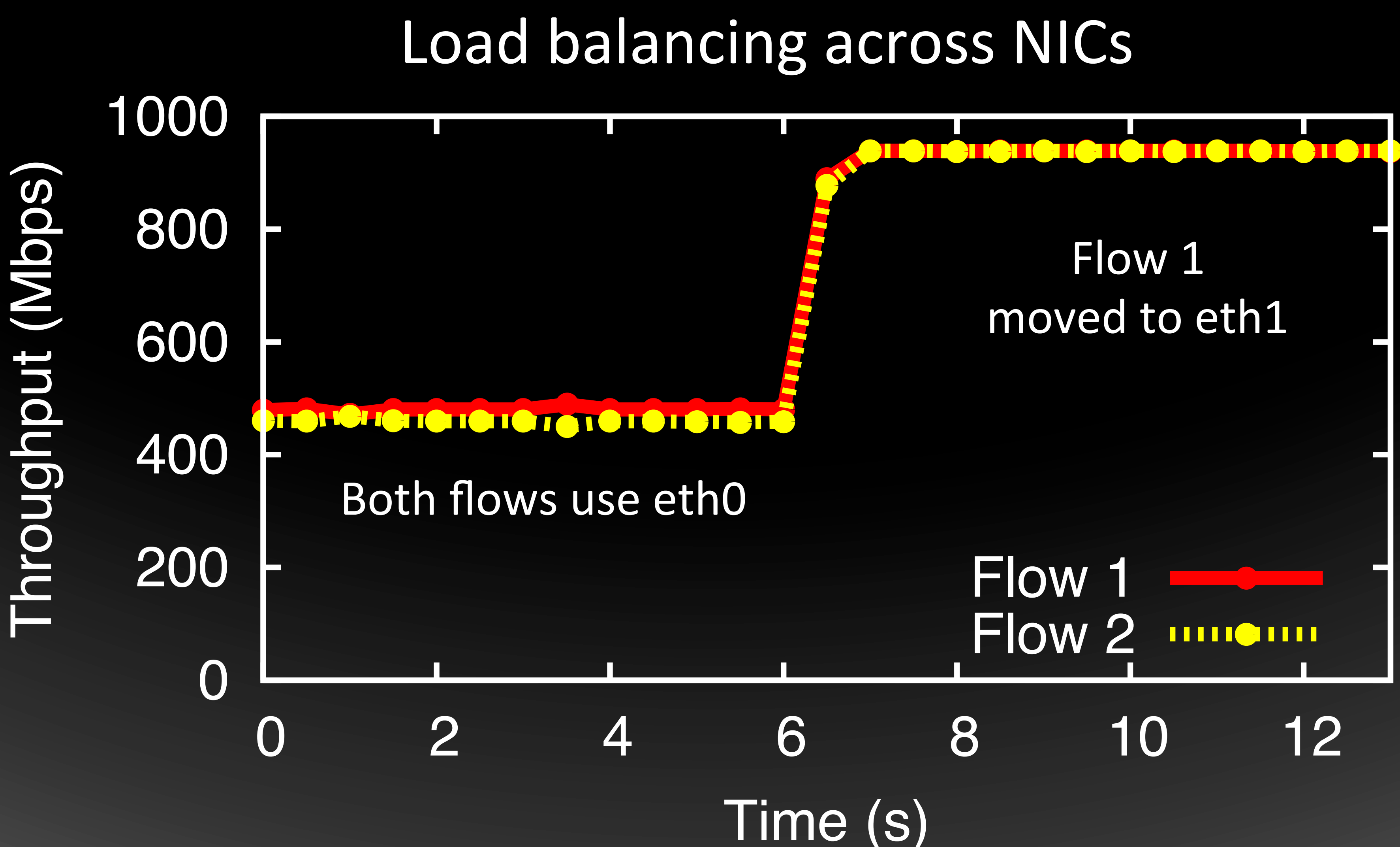
Use of Migration on Clients



Saves > 900 MB cellular data per month

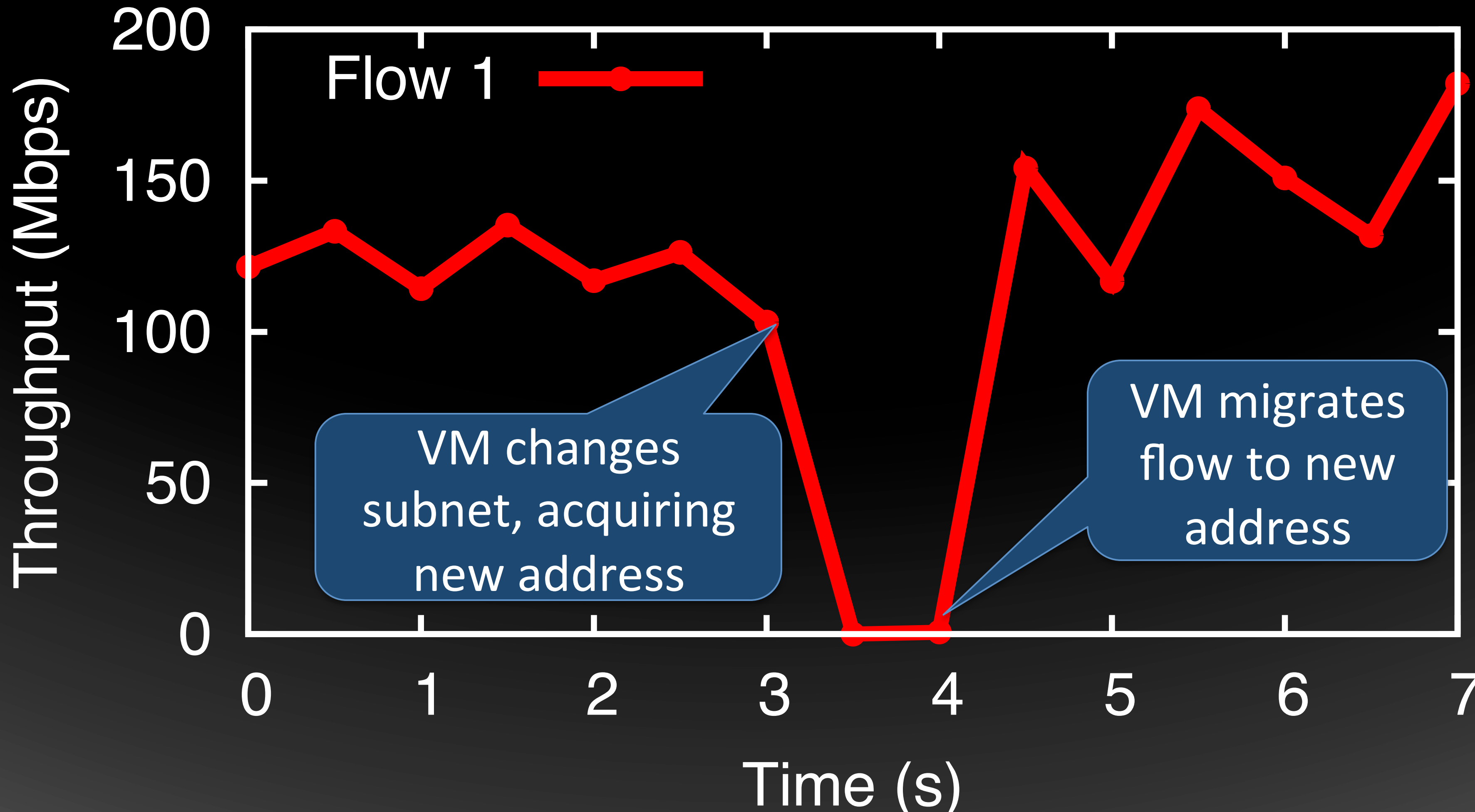
Single Serval TCP connection that never breaks

Uses of Migration on Servers



Uses of Migration on Servers

Migrating VMs across subnets



Competitive Performance

TCP Throughput

	Mean (Mbit/s)
TCP/IP	934.5
Serval	933.8
Translator	932.1

Service Table Throughput

	Mbit/s	Kpkt/s
IP forwarding	987	388.4
Serval	872	142.8

Applications are Easy to Port

Application	Codebase	Changes
Iperf	5,934	240
TFTP	3,452	90
wget	87,164	207
Elinks browser	115,224	234
Firefox browser	**4,615,324**	**70**
Mongoose webserver	8,831	425
Memcached server	8,329	159
Memcached client	12,503	184

SDN to the Edges!

- SDN about network-wide visibility and control
  - Today's "SDN" (OpenFlow) primarily focuses on layer-2 / layer-3 abstractions
- Serval extends SDN model to the network edge
  - New programming abstractions for services, flows, hosts, and interfaces
  - Service-level control/data plane split
- Joint service and network control

Summary of Contributions

- New naming abstractions
  - Clean role separation in the stack
  - Makes it easier to build and manage services
- Software architecture for services
  - Flexible service resolution and discovery
  - Maintains robust connectivity
  - Joint service and network management

serval-arch.org

@servalnetwork

Papers, demos, source code (GPL) online

Related Work

- Locator/identifier separation
  - HIP, i3, HAIR, DOA, LISP, LNA
- Data-oriented networking
  - DONA, CCNx
- Support for mobility and migration
  - TCP Migrate, Mobile IP, ROAM
- Multipath and multi-homing
  - MPTCP, SCTP, Shim6