Service-centric networking with SCAFFOLD

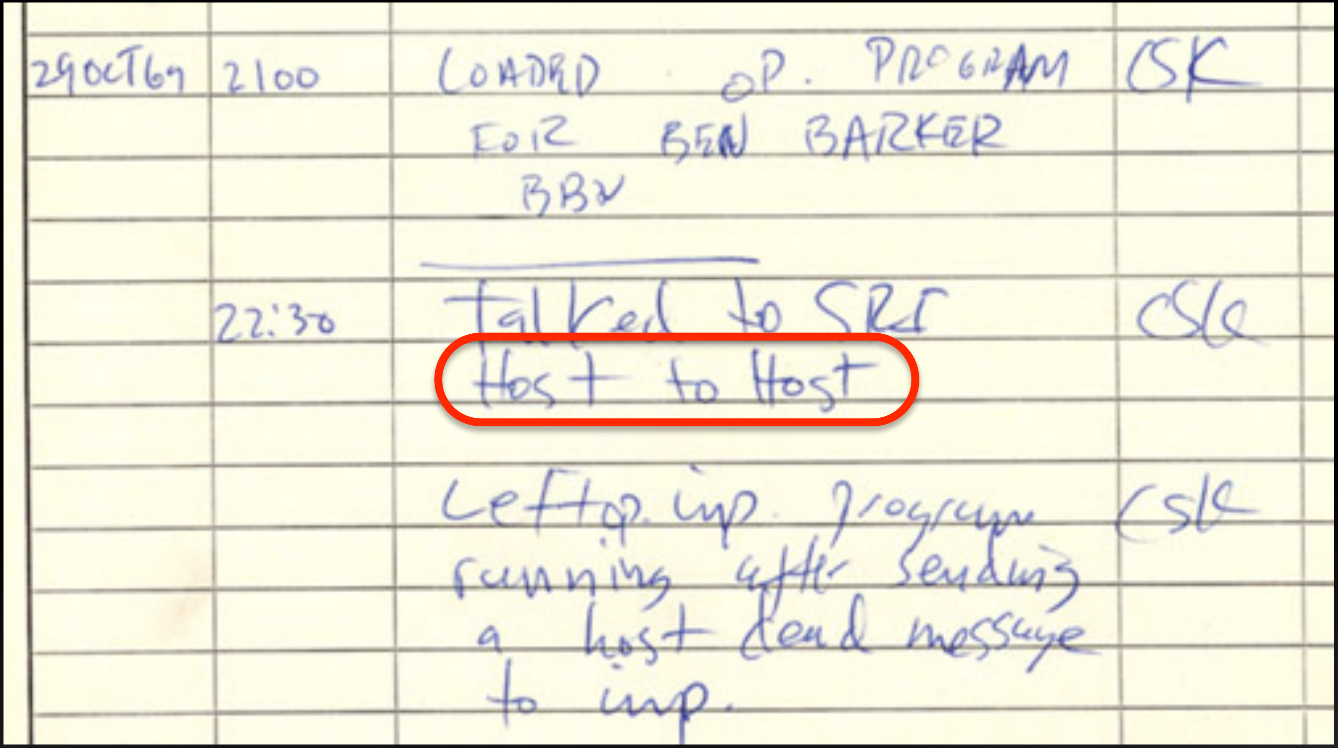
Michael J. Freedman

Princeton University

with Matvey Arye, Prem Gopalan, Steven Ko, Erik Nordstrom, Jen Rexford, and David Shue

From a host-centric architecture

1960s

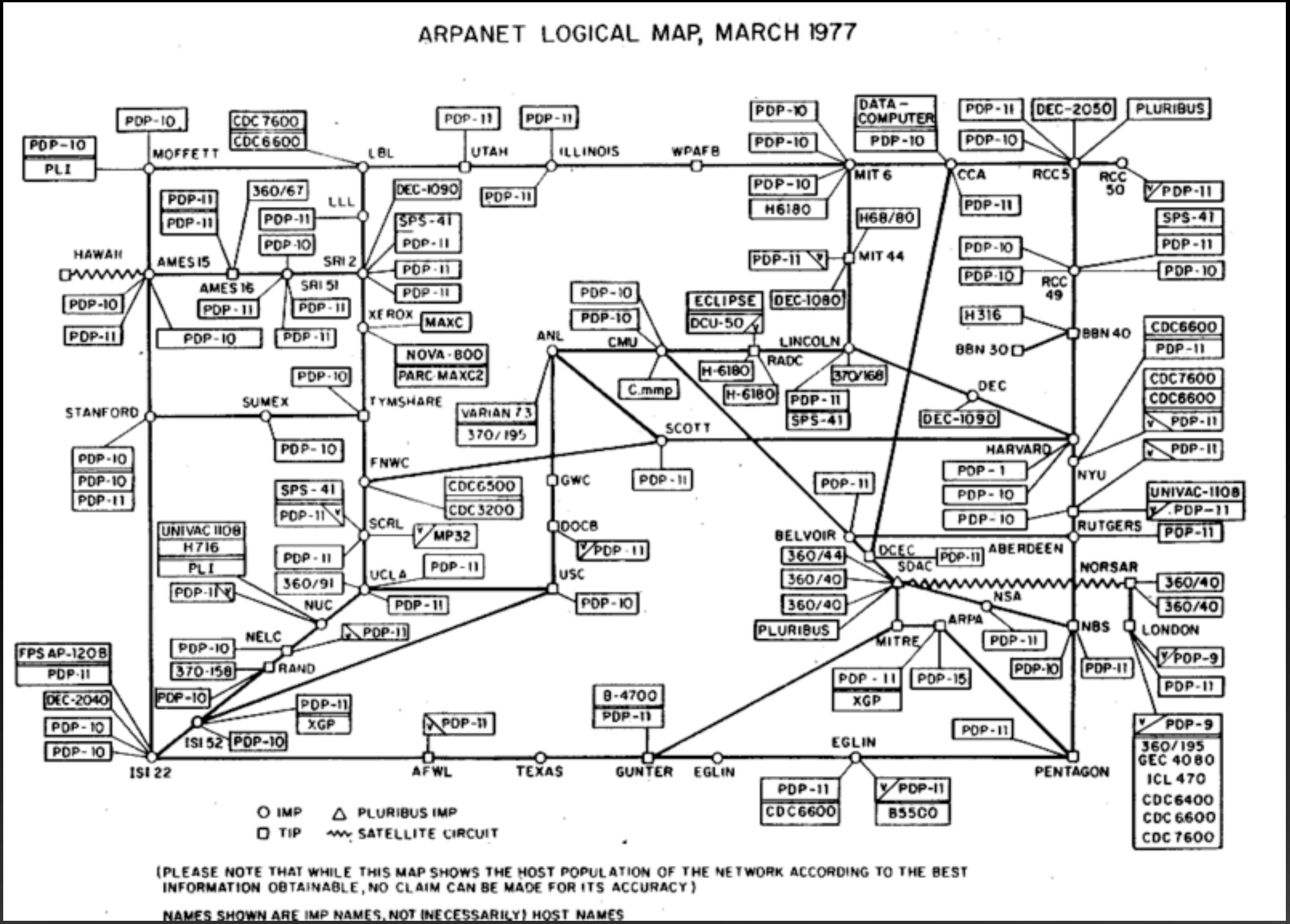


29OCT69	2100	LOADED OP. PROGRAM FOR BEN BARKER BBN	CSK
	22:30	talked to SRI Host to Host	CSK
		Left op. imp program running after sending a host dead message to imp.	CSK

From a host-centric architecture

1960s

1970s



ARPANET LOGICAL MAP, MARCH 1977

(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

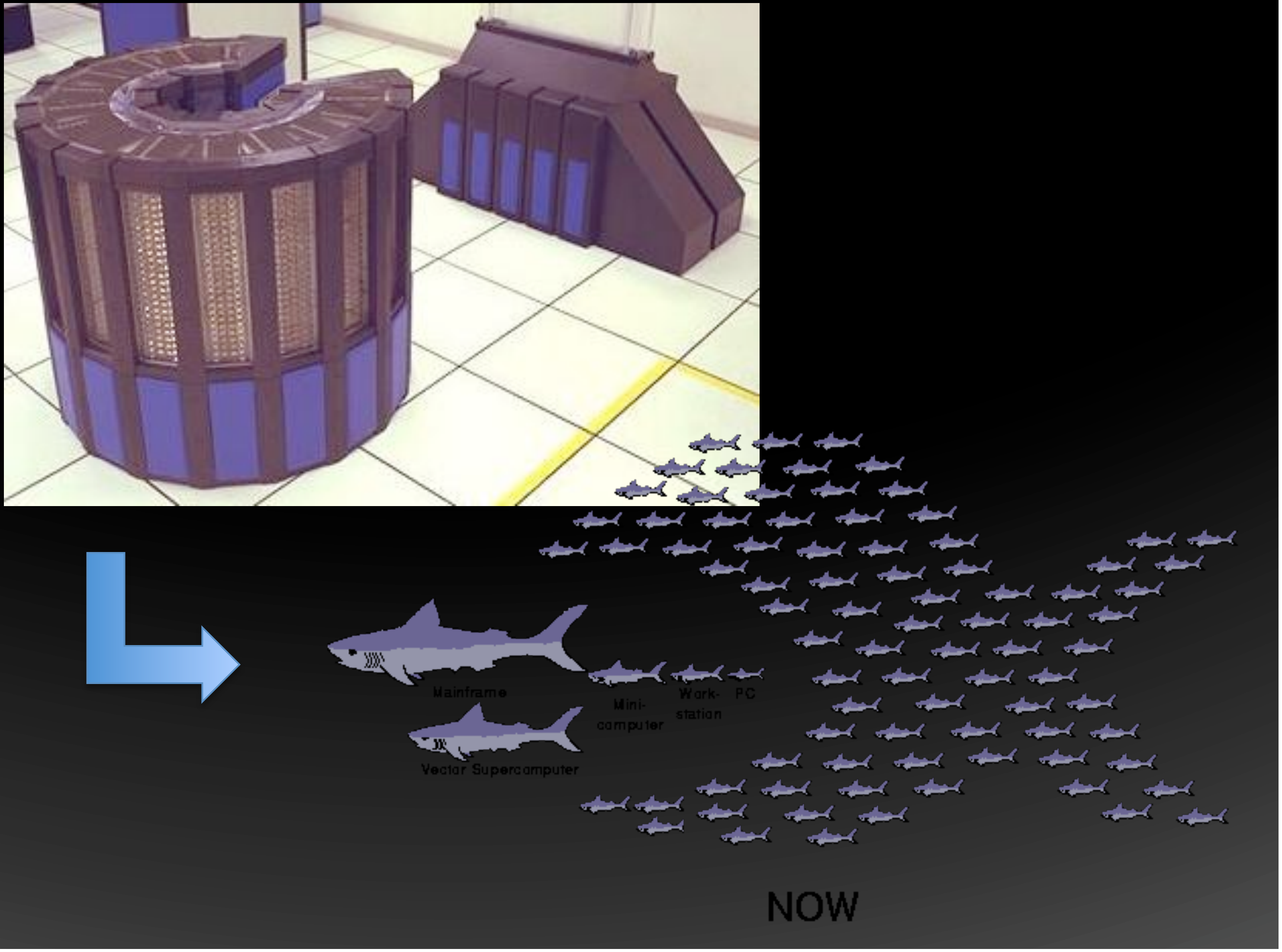
NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

From a host-centric architecture

1960s

1970s

1990s



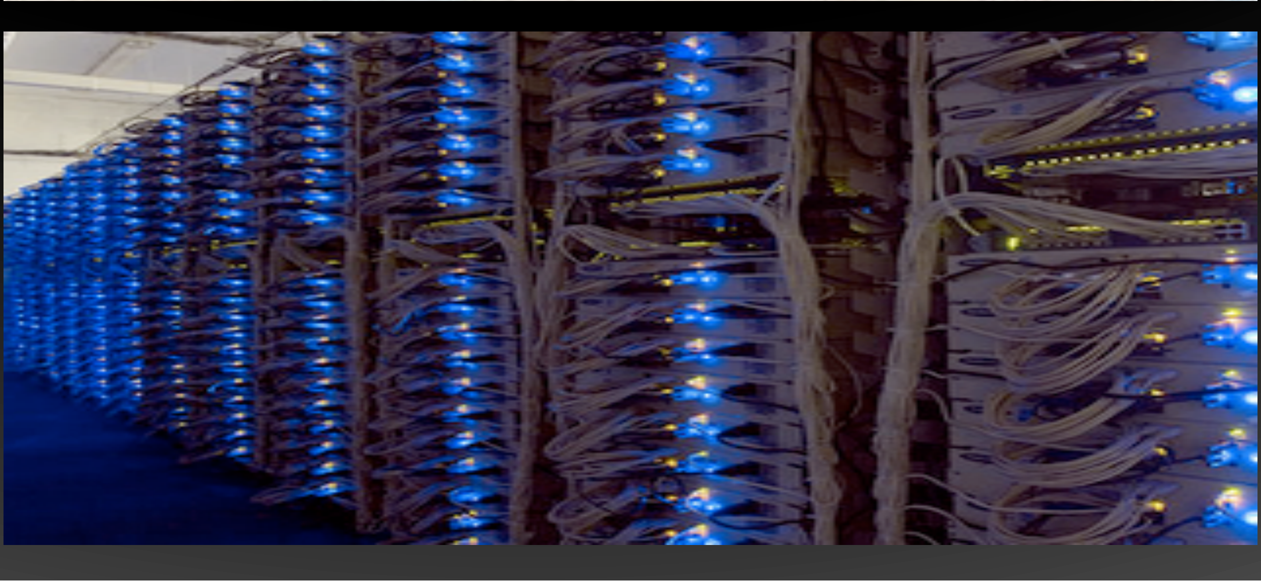
To a service-centric architecture

1960s

1970s

1990s

2000s



To a service-centric architecture

- Users want services, agnostic of actual host/location

- Service operators need: replica selection / load balancing, replica registration, liveness monitoring, failover, migration, …

Hacks to fake service-centrism today

Layer 4/7:	DNS with small TTLs
	HTTP redirects
	Layer-7 switching
Layer 3:	IP addresses and IP anycast
	Inter/intra routing updates
Layer 2:	VIP/DIP load balancers
	VRRP, ARP spoofing

+ Home-brewed registration, configuration, monitoring, …

To a service-centric architecture

- Users want services, agnostic of actual host/location

- Service operators need: replica selection / load balancing, replica registration, liveness monitoring, failover, migration, …
- Service-level anycast as basic network primitive

Two high-level questions

- Moderate vision: Can network support aid self-configuration for replicated services?

- Big vision: Should "service-centric networking" become the new thin waist of Internet?

Naming as a "thin waist"

- Host-centric design: Traditionally one IP per NIC
  - Load balancing, failover, and mobility complicates
  - Now: virtual IPs, virtual MACs, …

- Content-centric architecture: Unique ID per data object
  - DONA (Berkeley), CCN (PARC), …

- SCAFFOLD: Unique ID per group of processes
  - Each member must individually provide full group functionality
  - Group can vary in size, distributed over LAN or WAN

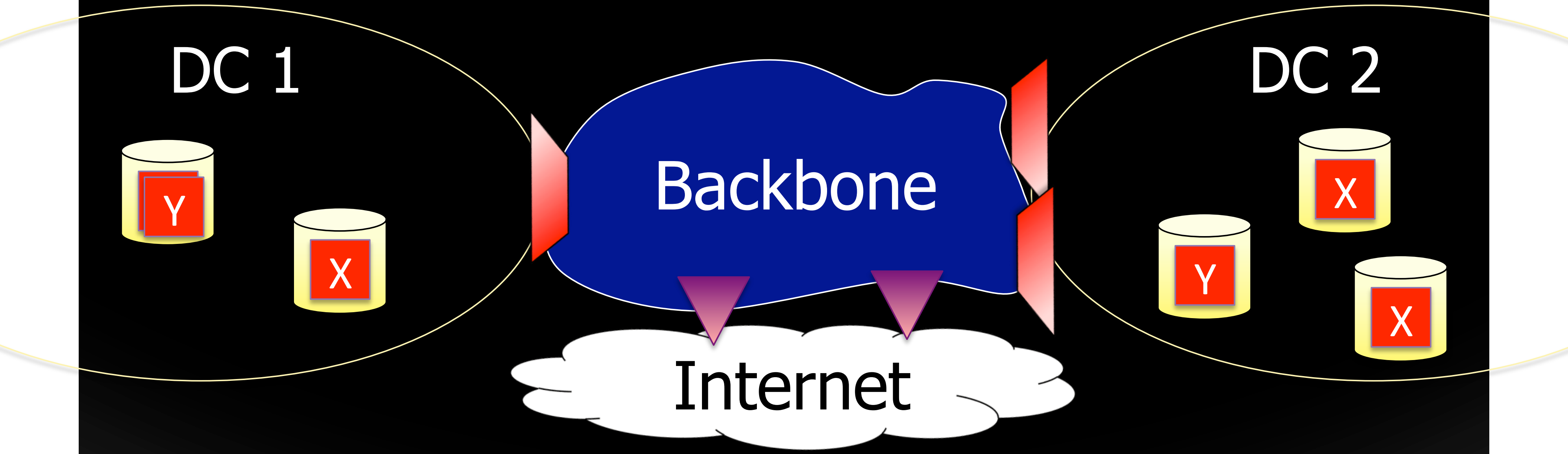
Object granularity can vary by service

		Fixed Bit-length	
SCAFFOLD ObjectID	=	K-bit Admin Prefix	Machine-readable ObjectID
YouTube	=	Google	YouTube Service
IZ – "Somewhere over the rainbow"	=	Google	IZ – "Somewhere" video
facebook **Memcache Partition**	=	Facebook	Partition 243
	=	Comcast	Mike's Laptop

SCAFFOLD as …

- Clean slate design
- Multi-datacenter architecture for single administrative domain
  - Deployed over legacy networks
  - Few / no modifications to applications

Target: Single administrative domain



- Datacenter management more unified, simple, centralized
- Host OS net-imaged and can be fork-lift upgraded
- Already struggling to provide scalability and service-centrism
- Cloud computing lessen importance of fixed, physical hosts

Goals for Service-Centrism

- Handling replicated services
  - Control over replica selection among groups
  - Control of network resources shared between groups
  - Handling dynamics among group membership and deployments
- Handling churn
  - Flexibility: From sessions, to hosts, to datacenters
  - Robustness: Largely hide from applications
  - Scalability: Local changes shouldn’t need to update global info
  - Scalability: Churn shouldn’t require per-client state in network
  - Efficiency: Wide-area migration shouldn’t require tunneling

Clean-Slate Design

Principles of SCAFFOLD

1. Service-level naming exposed to network
2. Anycast with flow affinity as basic primitive
3. Migration and failover through address remapping
   - Addresses bound to physical locations (aggregatable)
   - Flows identified by each endpoint, not pairwise
   - Control through in-band signalling; stateless forwarders
4. Minimize visibility of churn for scalability
   - Different addr's for different scopes (successive refinement)
5. Tighter host-network integration
   - Allowing hosts / service instances to dynamically update network

Principles of SCAFFOLD

1. Service-level naming exposed to network

2. Anycast with flow affinity as basic primitive

Principles of SCAFFOLD

1. Service-level naming exposed to network

2. Anycast with flow affinity as basic primitive

SCAFFOLD address

Admin Prefix	Object Name	SS Label	Host Label	SocketID
ObjectID		FlowID		

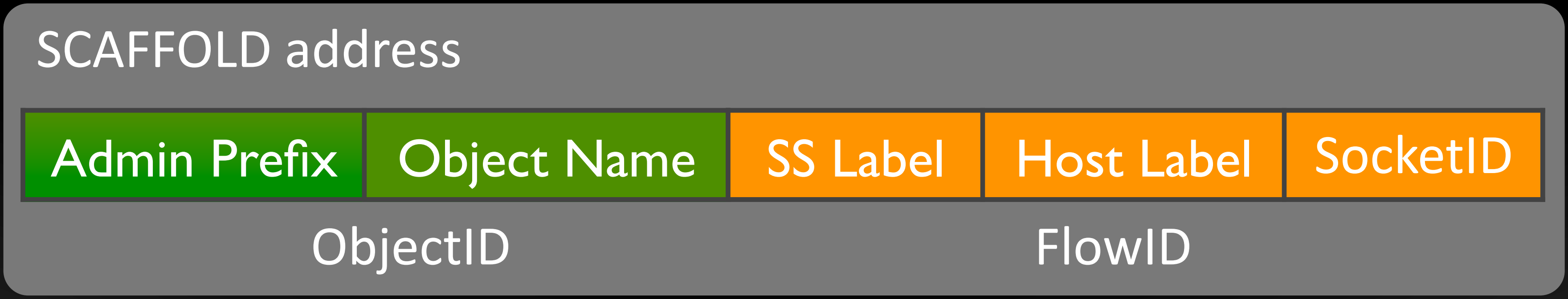
(i) Resolve ObjectID to an instance FlowLabel

(ii) Route on instance FlowLabel to the destination

(iii) Subsequent flow packets use same FlowLabel

Principles of SCAFFOLD

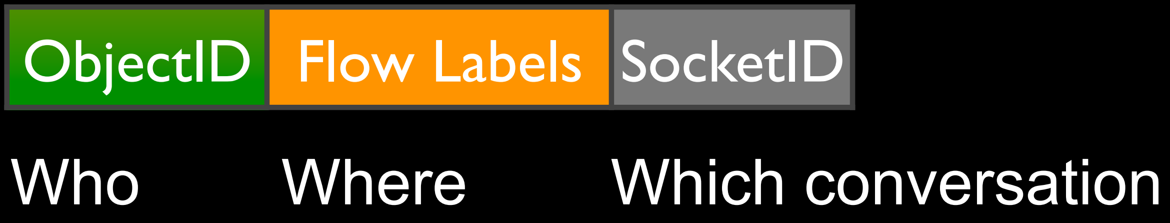
1. Service-level naming exposed to network
2. Anycast with flow affinity as basic primitive



SCAFFOLD address

Admin Prefix	Object Name	SS Label	Host Label	SocketID
ObjectID		FlowID		

Decoupled flow identifiers

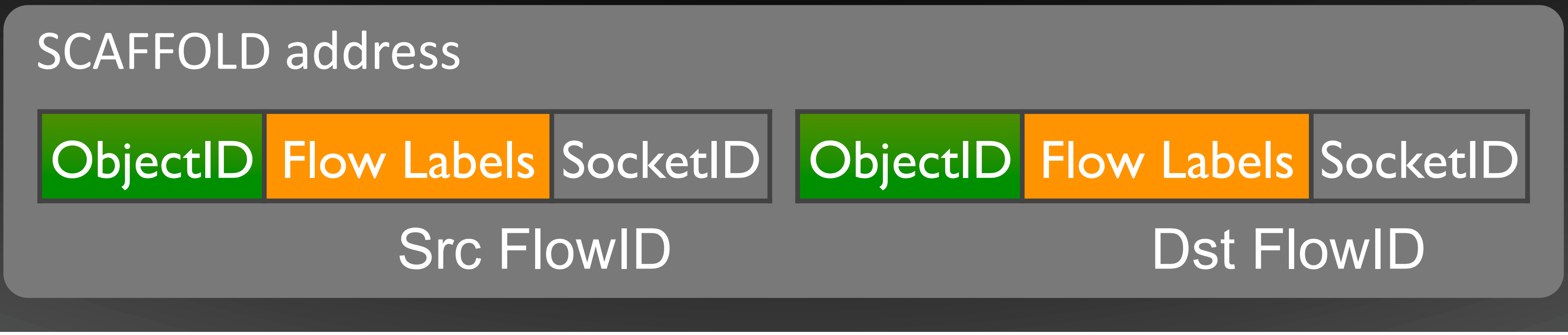


ObjectID Flow Labels SocketID

Who Where Which conversation

3. Migration and failover through address remapping

4. Minimize visibility of churn for scalability



SCAFFOLD address

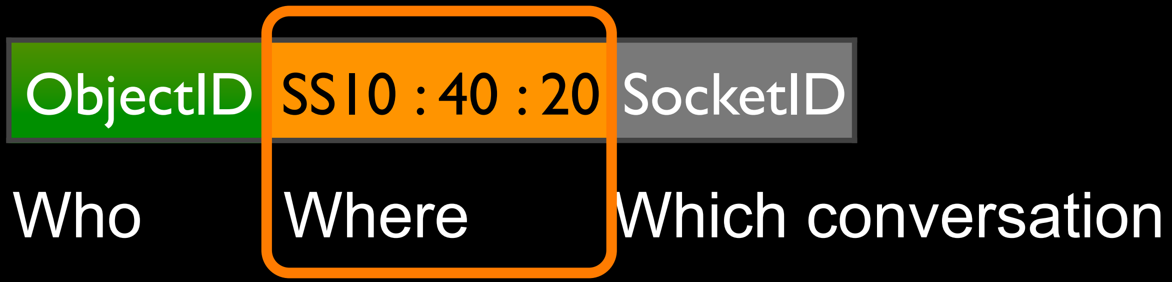
ObjectID Flow Labels SocketID

Src FlowID

ObjectID Flow Labels SocketID

Dst FlowID

Manage migration / failover through *in-band address remapping*



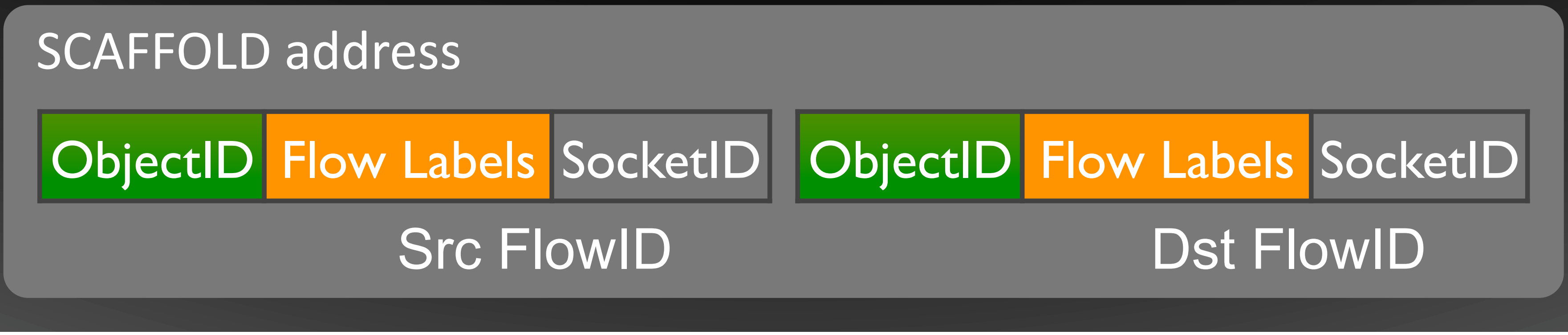
ObjectID | SS10 : 40 : 20 | SocketID

Who | Where | Which conversation

(i) Local end-point changes location, assigned new address

(ii) Existing connections signal new address to remote end-points

(iii) Remote network stack updated, application unaware



SCAFFOLD address

ObjectID | Flow Labels | SocketID — Src FlowID

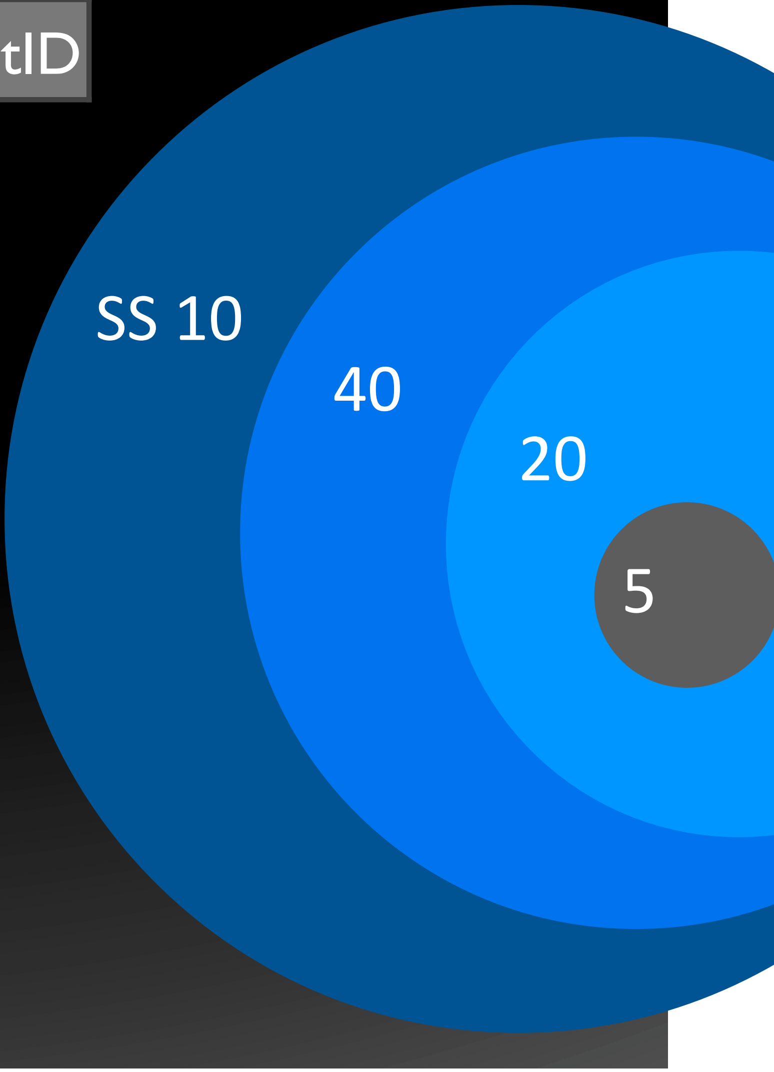
ObjectID | Flow Labels | SocketID — Dst FlowID

Minimize visibility of churn through *successive refinement*

ObjectID | SS10 : 40 : 20 | SocketID

Where

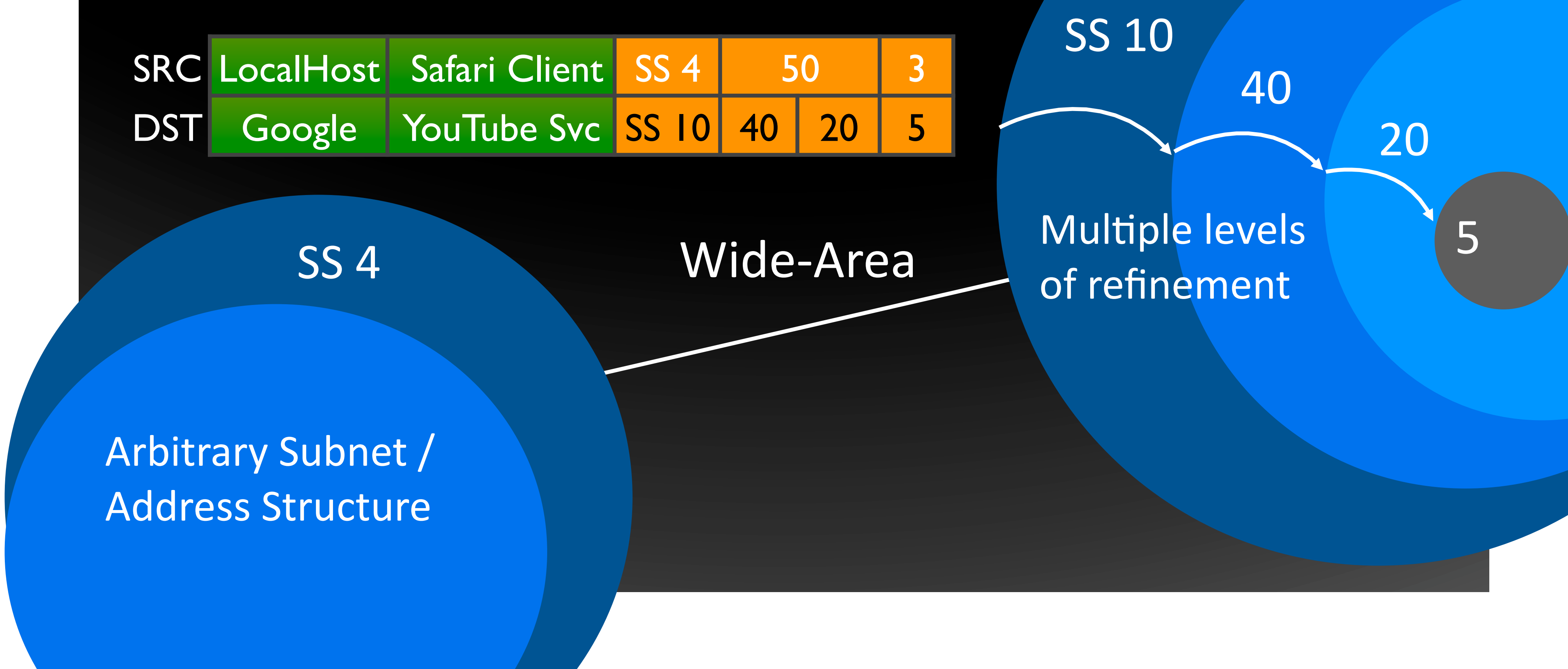
Wide-Area



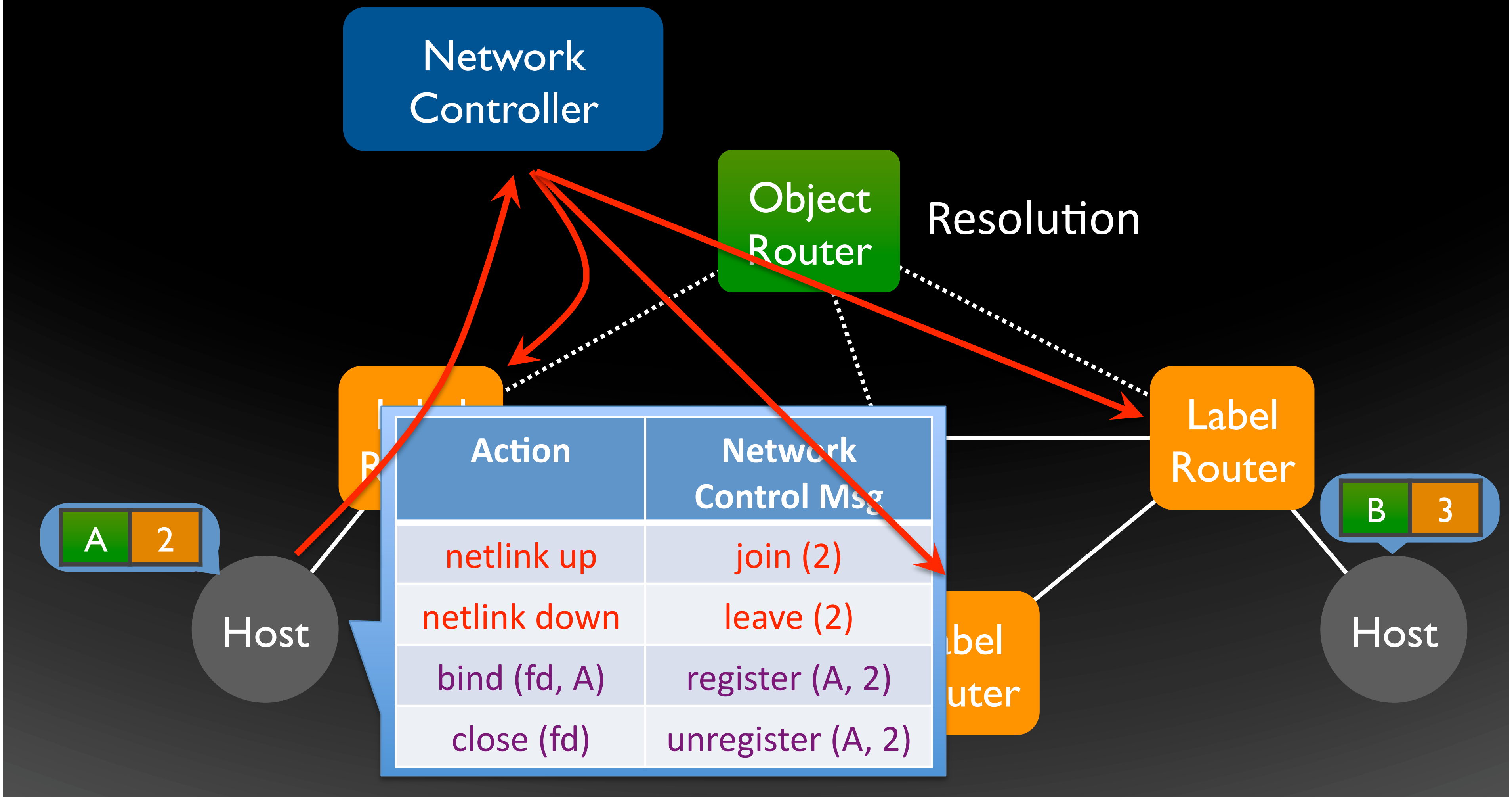
Minimize visibility of churn through *successive refinement*

- Scalability: – Local churn only updates local state
  – Addresses remain hierarchical
- Info hiding: Topology not globally exposed

SRC	LocalHost	Safari Client	SS 4	50		3
DST	Google	YouTube Svc	SS 10	40	20	5


SS 10
40
20
5
Multiple levels of refinement
SS 4
Wide-Area
Arbitrary Subnet / Address Structure

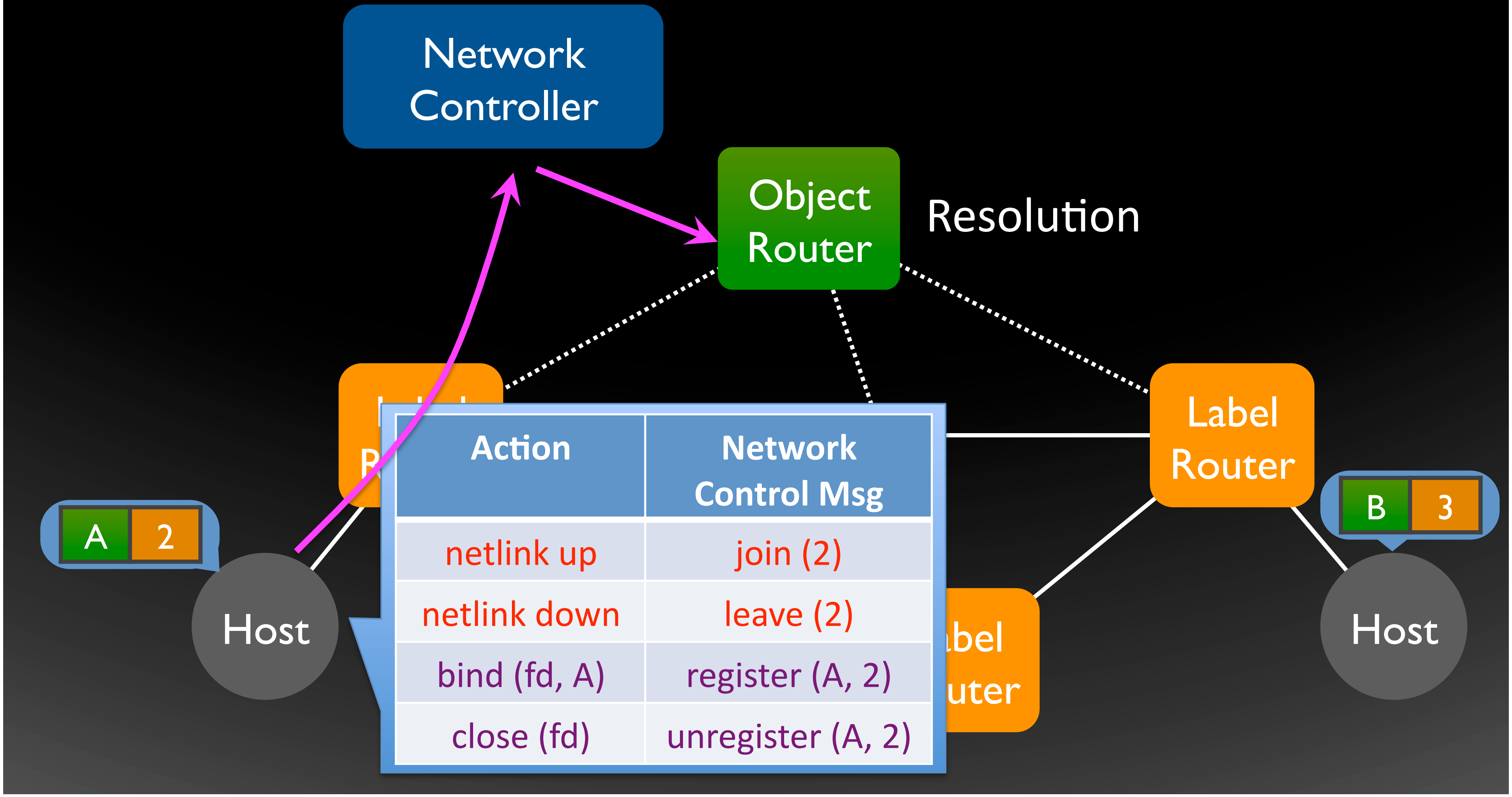
Integrated service-host-network management



Action	Network Control Msg
netlink up	join (2)
netlink down	leave (2)
bind (fd, A)	register (A, 2)
close (fd)	unregister (A, 2)

Integrated service-host-network management

Self-configuration + adaptive to churn



Action	Network Control Msg
netlink up	join (2)
netlink down	leave (2)
bind (fd, A)	register (A, 2)
close (fd)	unregister (A, 2)

Using SCAFFOLD:

Network-level protocols

and network support

Application’s network API

Today (IP / BSD sockets)

```
fd = open();
```

Datagram:

```
sendto (IP:port, data)
```

Stream:

```
connect (fd, IP:port)
send (fd, data);
```

SCAFFOLD

```
fd = open();
```

Unbound datagram:

```
sendto (objectID, data)
```

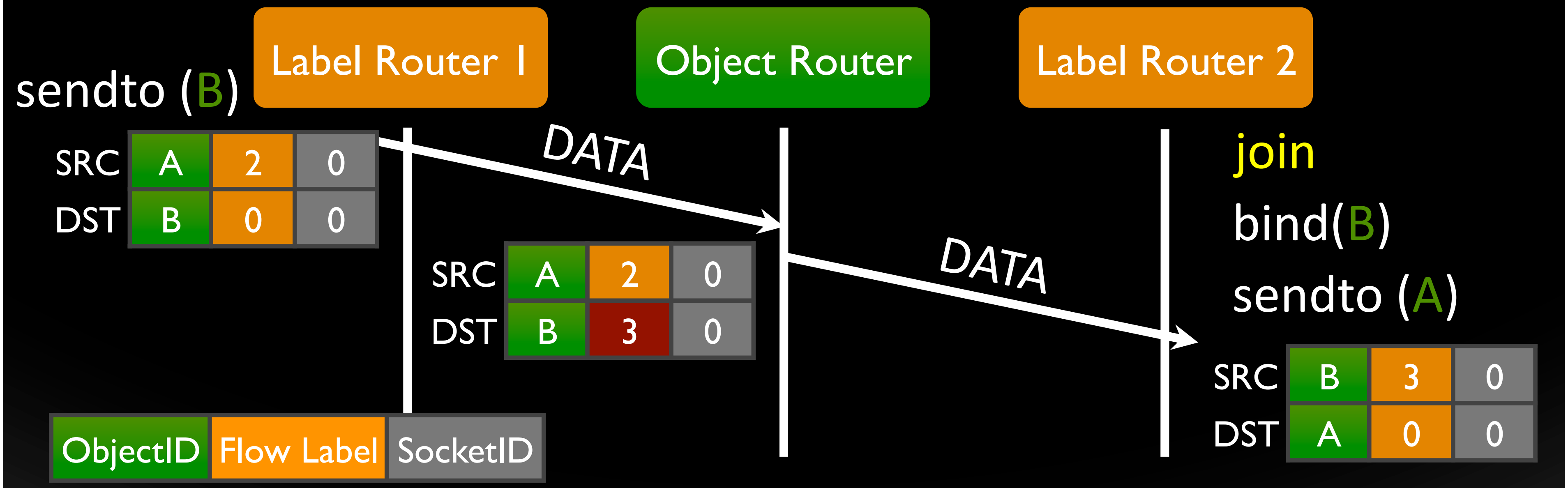
Bound datagram:

```
connect (fd, objectID)
send (fd, data);
```

IP: Application sees network, network doesn’t see app

SCAFFOLD: Network sees app, app doesn’t see network

Unbound Flows



sendto (B)

SRC	A	2	0
DST	B	0	0

SRC	A	2	0
DST	B	3	0

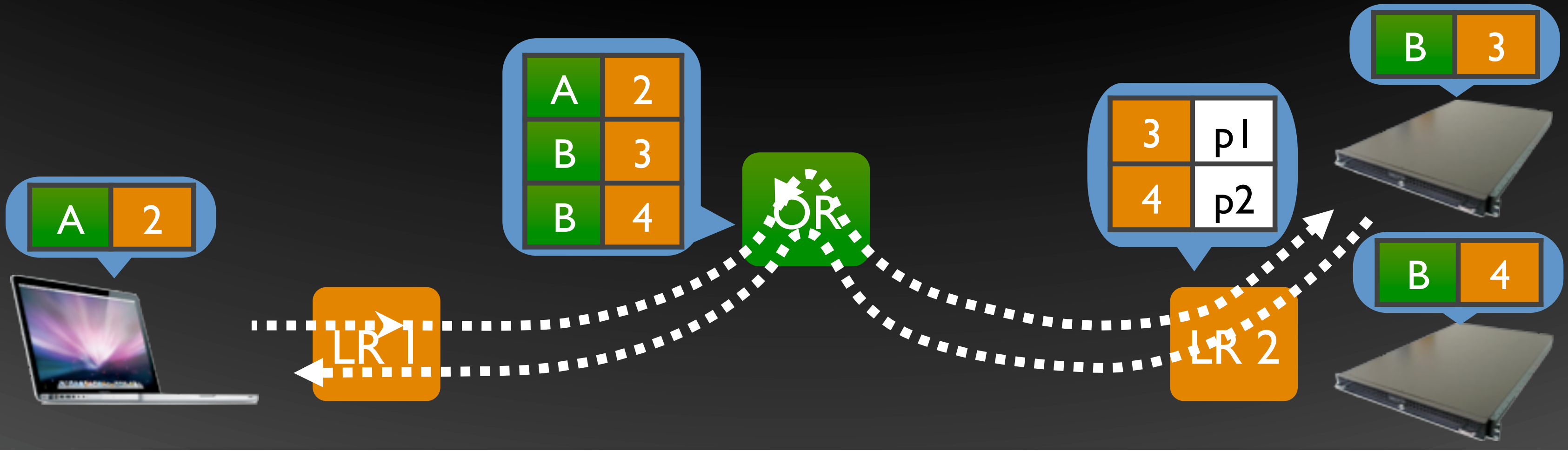
join

bind(B)

sendto (A)

SRC	B	3	0
DST	A	0	0

ObjectID	Flow Label	SocketID



A	2

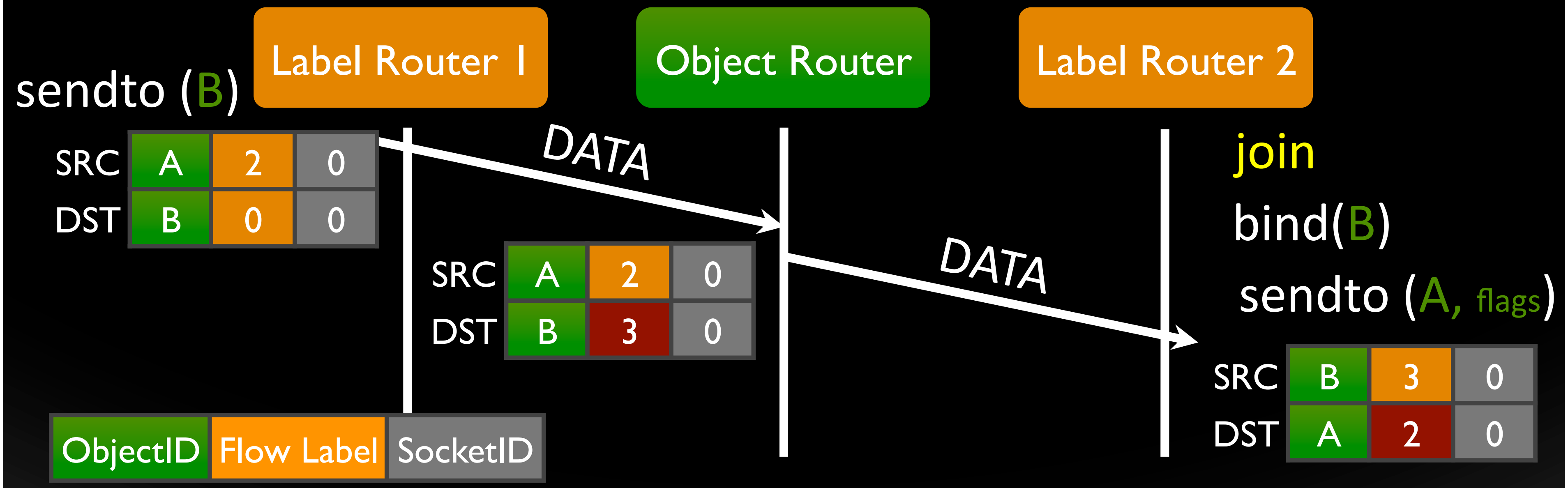
A	2
B	3
B	4

3	p1
4	p2

B	3

B	4

Half-Bound Flows



sendto (B)

SRC	A	2	0
DST	B	0	0

SRC	A	2	0
DST	B	3	0

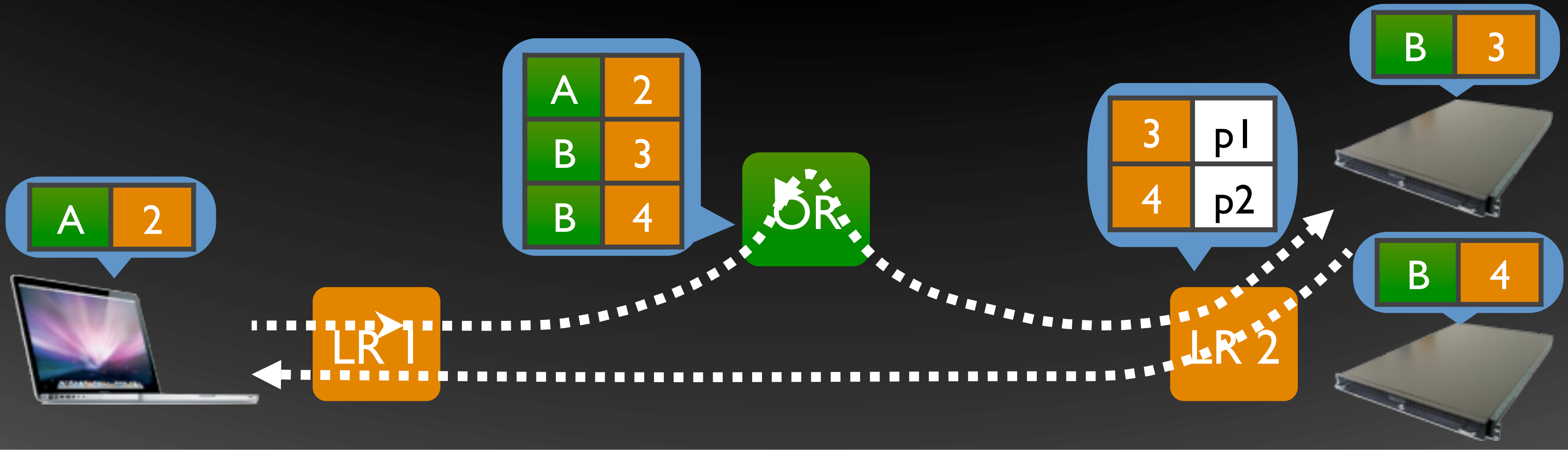
ObjectID	Flow Label	SocketID

join

bind(B)

sendto (A, flags)

SRC	B	3	0
DST	A	2	0



A	2

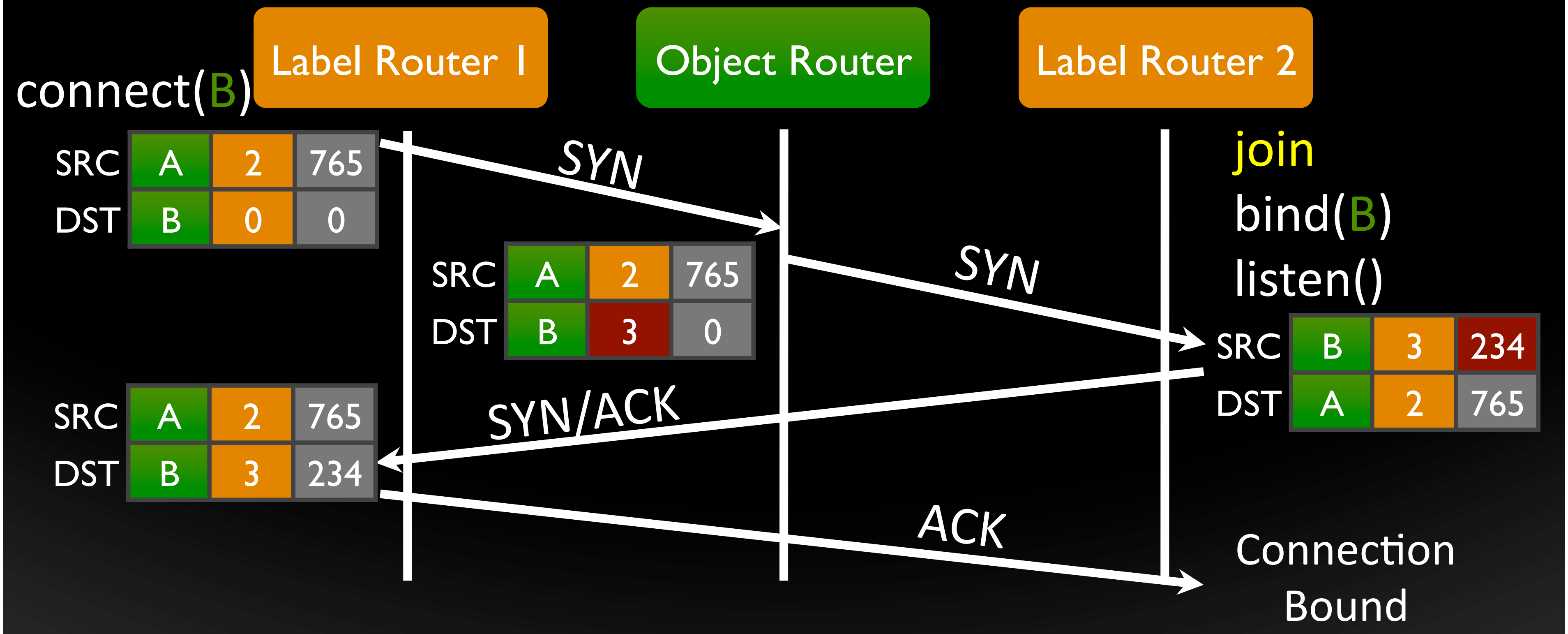
A	2
B	3
B	4

3	p1
4	p2

B	3

B	4

Bound Flows



Label Router 1

Object Router

Label Router 2

connect(B)

SRC	A	2	765
DST	B	0	0

SYN

SRC	A	2	765
DST	B	3	0

SYN

join

bind(B)

listen()

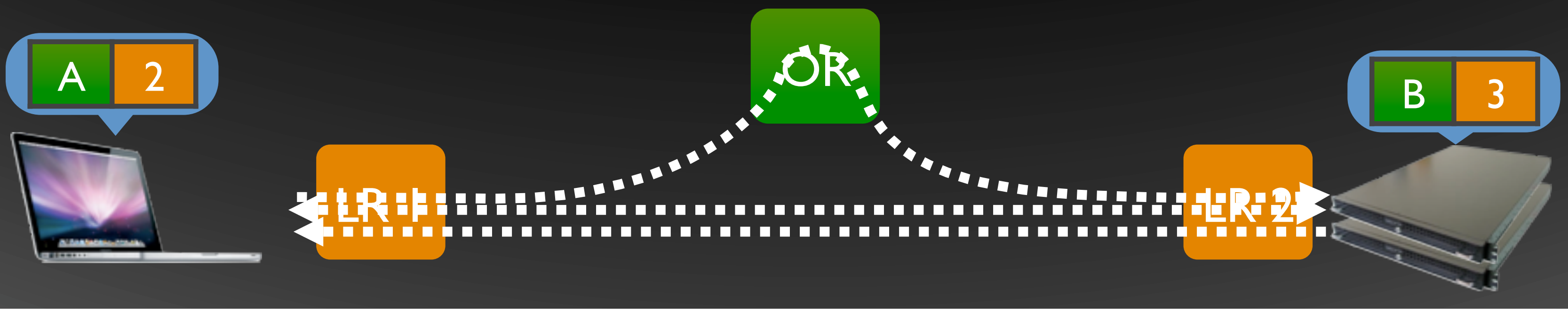
SRC	B	3	234
DST	A	2	765

SYN/ACK

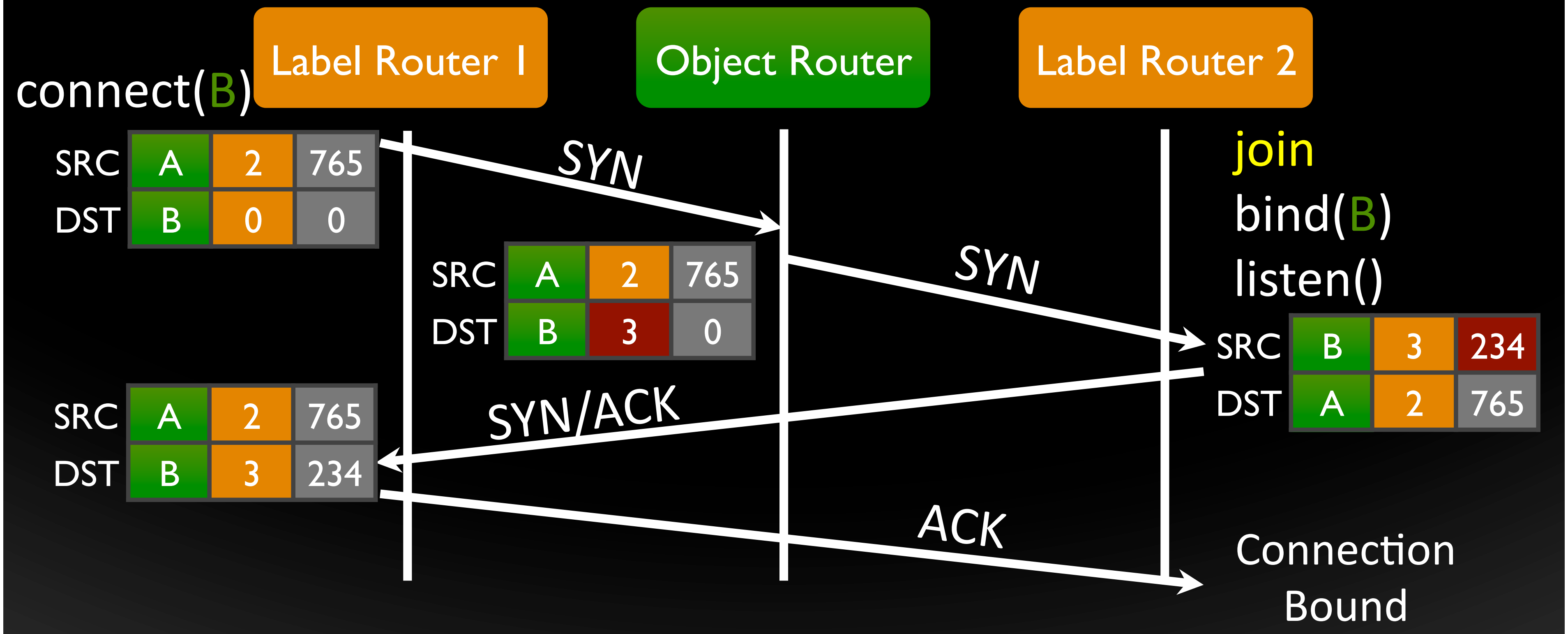
SRC	A	2	765
DST	B	3	234

ACK

Connection Bound



Bound Flows



SRC	A	2	765
DST	B	0	0

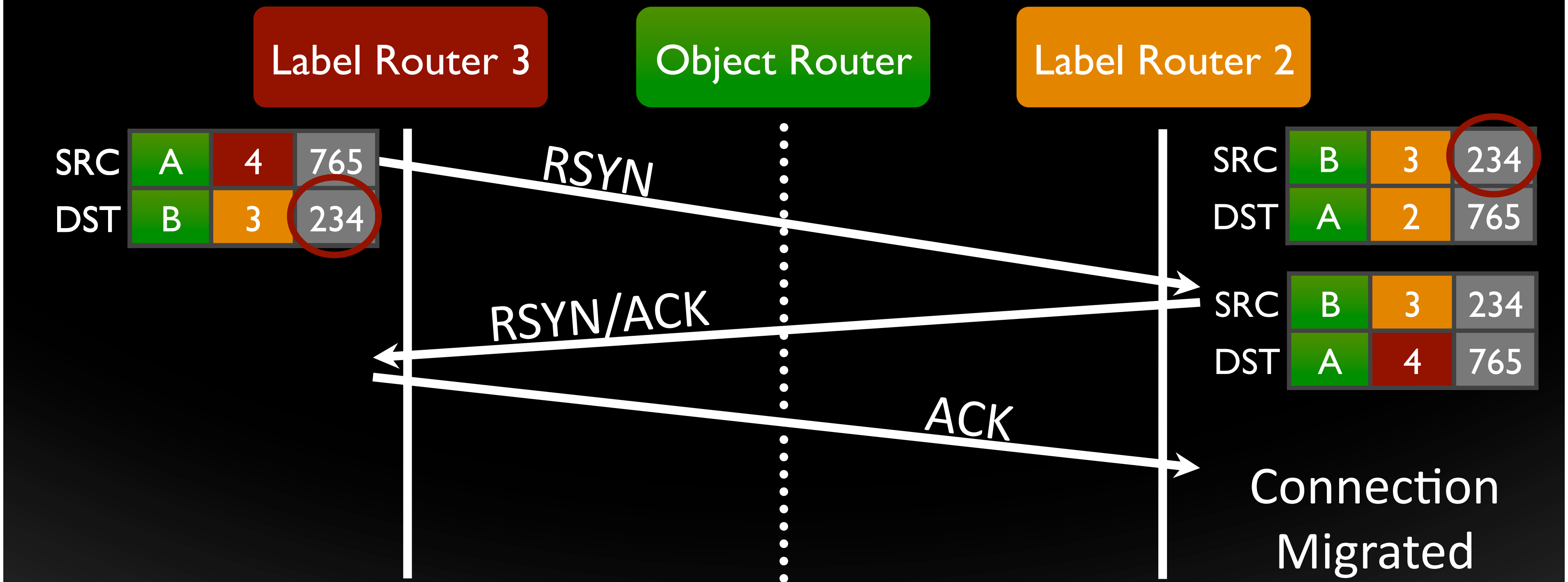
SRC	A	2	765
DST	B	3	0

SRC	B	3	234
DST	A	2	765

SRC	A	2	765
DST	B	3	234

- Applications bind on object-level names
- Network forwards on resolved addresses

Supporting Mobility and Migration



Label Router 3

Object Router

Label Router 2

SRC	A	4	765
DST	B	3	234

RSYN

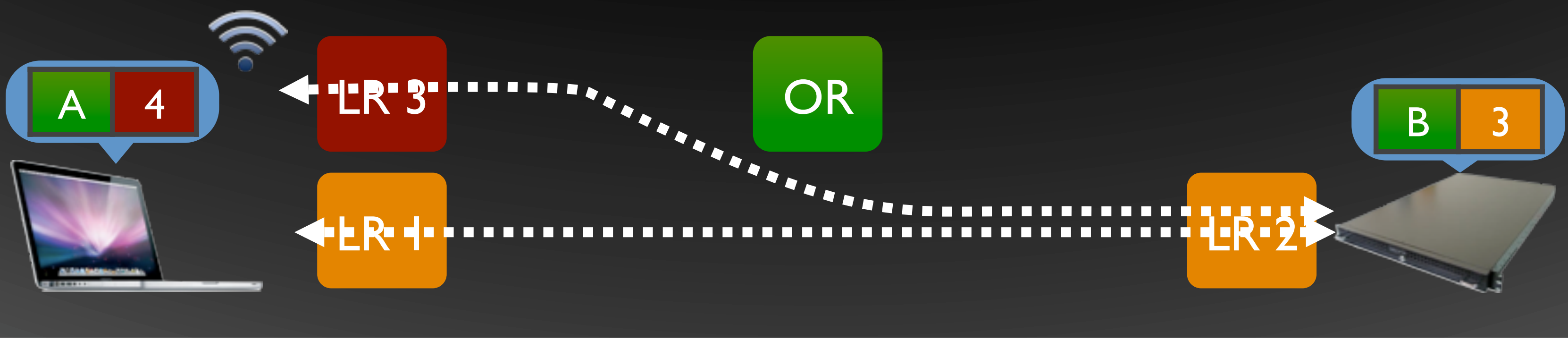
SRC	B	3	234
DST	A	2	765

RSYN/ACK

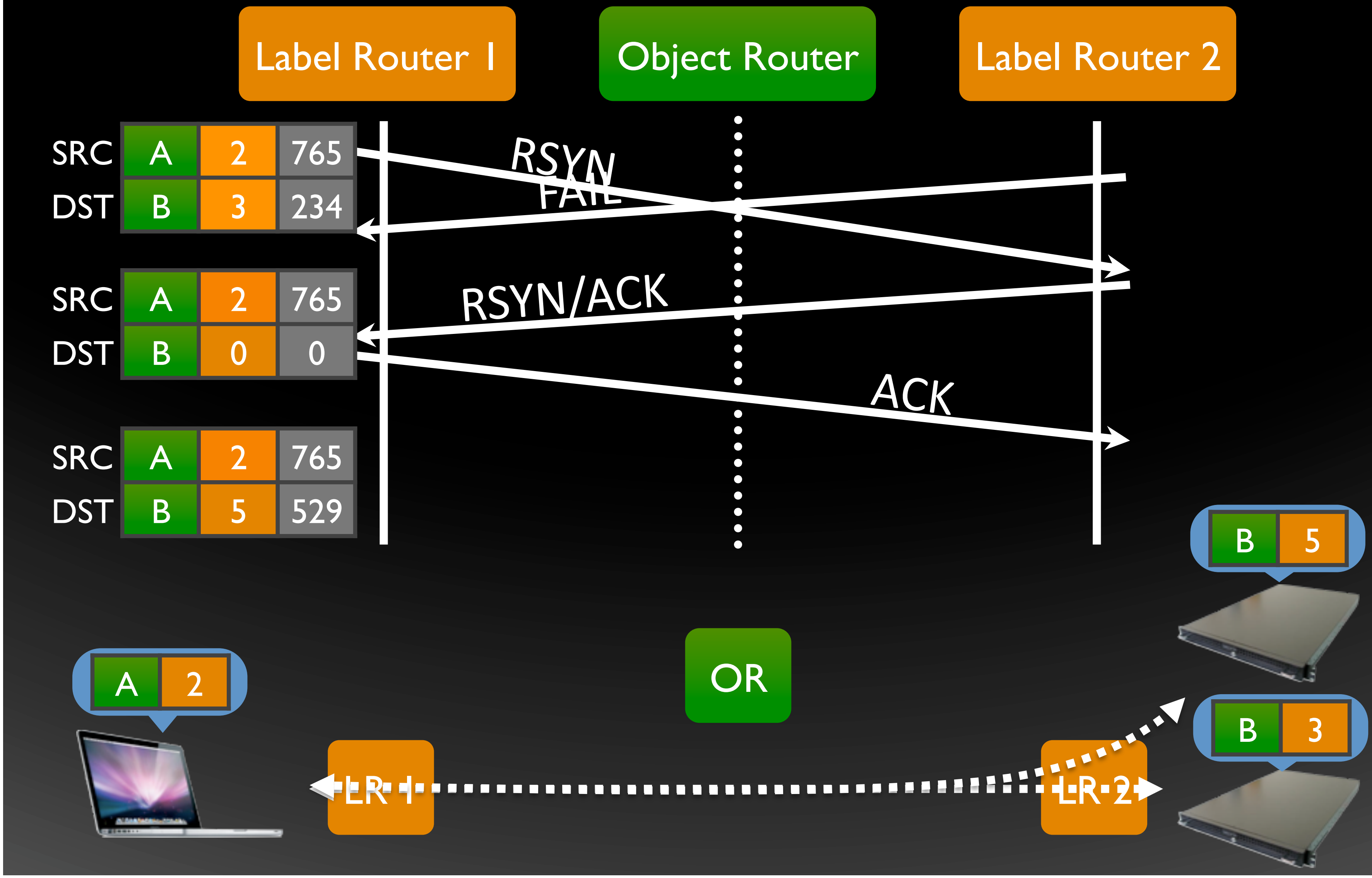
SRC	B	3	234
DST	A	4	765

ACK

Connection Migrated



Supporting Failover and Load Shedding

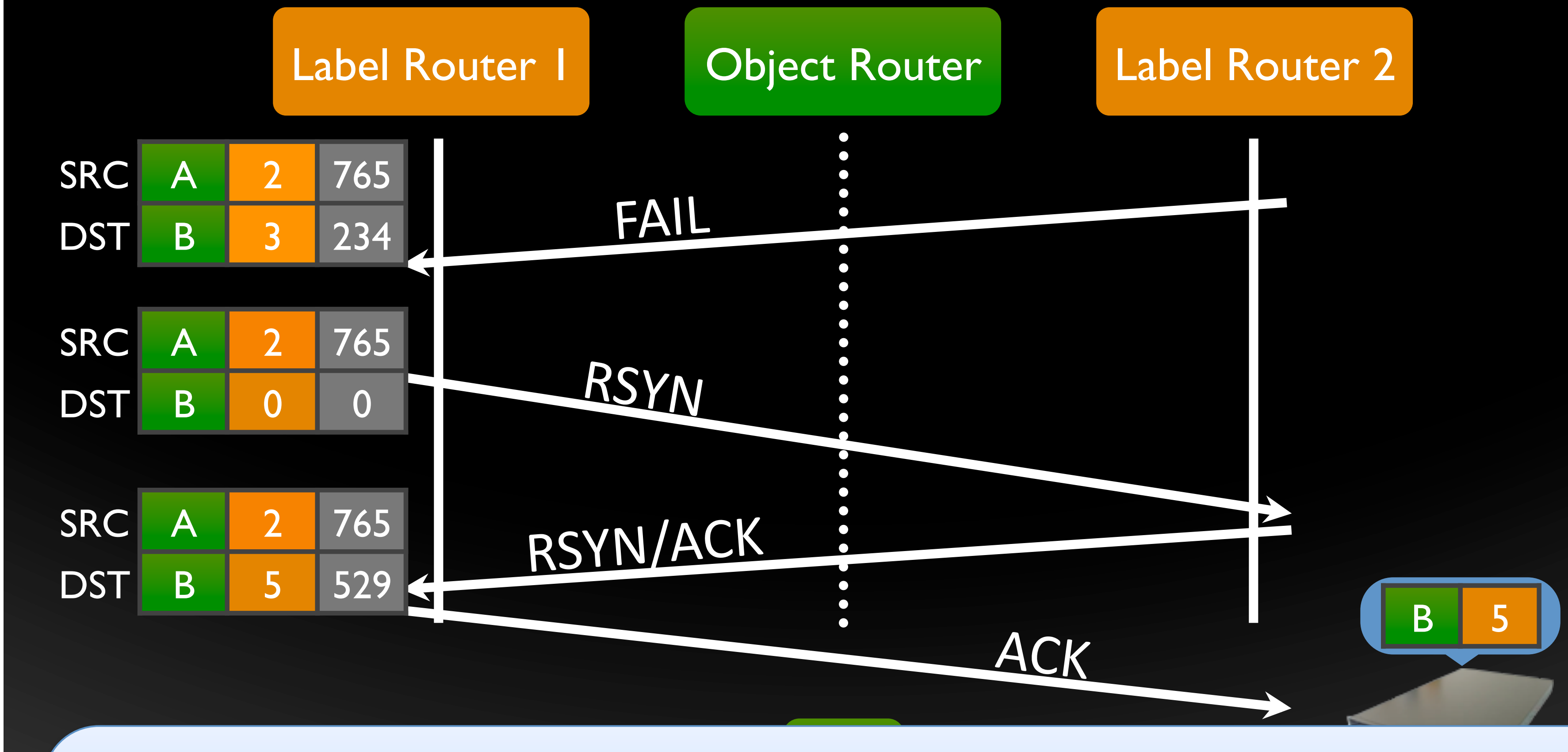


SRC	A	2	765
DST	B	3	234

SRC	A	2	765
DST	B	0	0

SRC	A	2	765
DST	B	5	529

Supporting Failover and Load Shedding



SRC	A	2	765
DST	B	3	234

SRC	A	2	765
DST	B	0	0

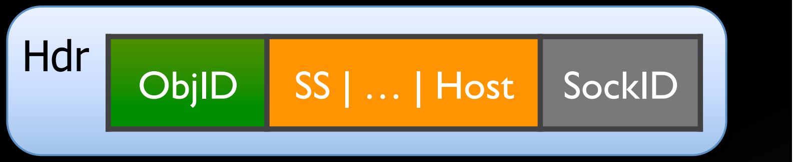
SRC	A	2	765
DST	B	5	529

- Decoupled id’s enable in-band migration and recovery
- Flow affinity without per-flow state in the network

Extent of changes

✓ Change socket layer + stack

✓ Change the packet format



✓ Change in-network support

Network Controller

Object Router

Label Router

Yet:

- ✓ Can run on top of legacy networks (IP and Ethernet)
- ✓ Few/easy/no changes to applications

Backwards Compatibility

Hide physical location from app

Today (IP / BSD sockets)

```
fd = open();
```

Datagram:

```
sendto (IP:port, data)
```

Stream:

```
connect (fd, IP:port)
send (fd, data);
```

SCAFFOLD

```
fd = open();
```

Unbound datagram:

```
sendto (objectID, data)
```

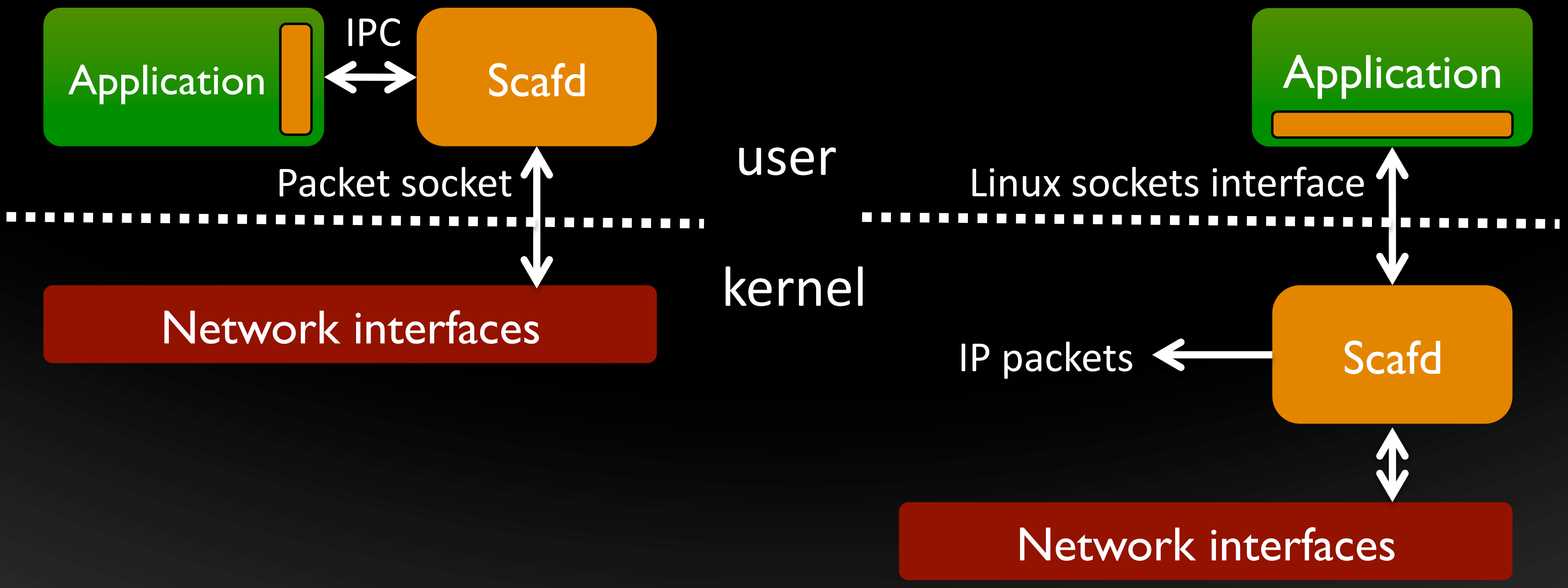
Bound datagram:

```
connect (fd, objectID)
send (fd, data);
```

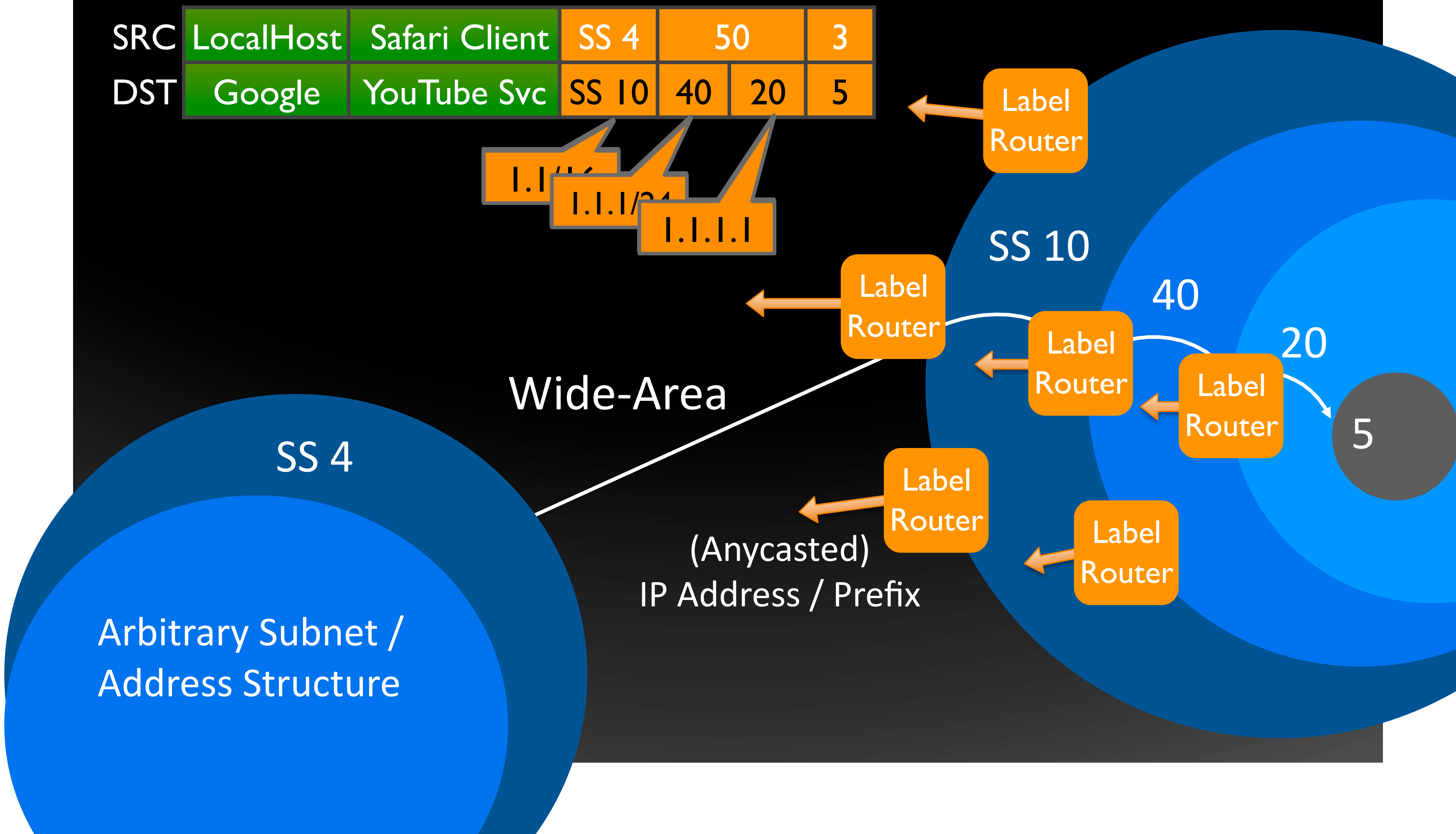
Current applications

- iperf, TFTP, PowerDNS

SCAFFOLD network stack

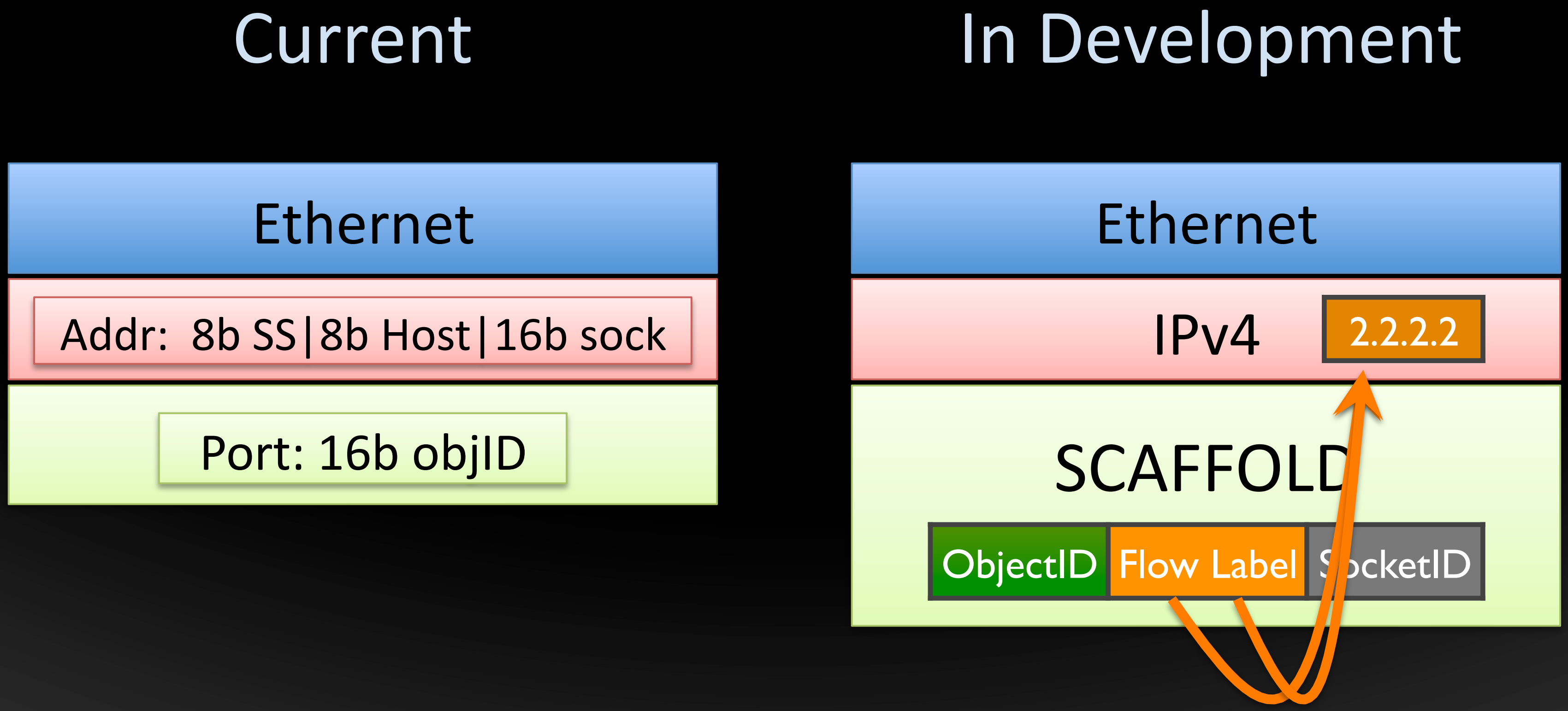


Operating across legacy networks



SRC	LocalHost	Safari Client	SS 4	50		3
DST	Google	YouTube Svc	SS 10	40	20	5

Routing over legacy networks



Current

Ethernet

Addr: 8b SS|8b Host|16b sock

Port: 16b objID

In Development

Ethernet

IPv4

2.2.2.2

SCAFFOLD

ObjectID

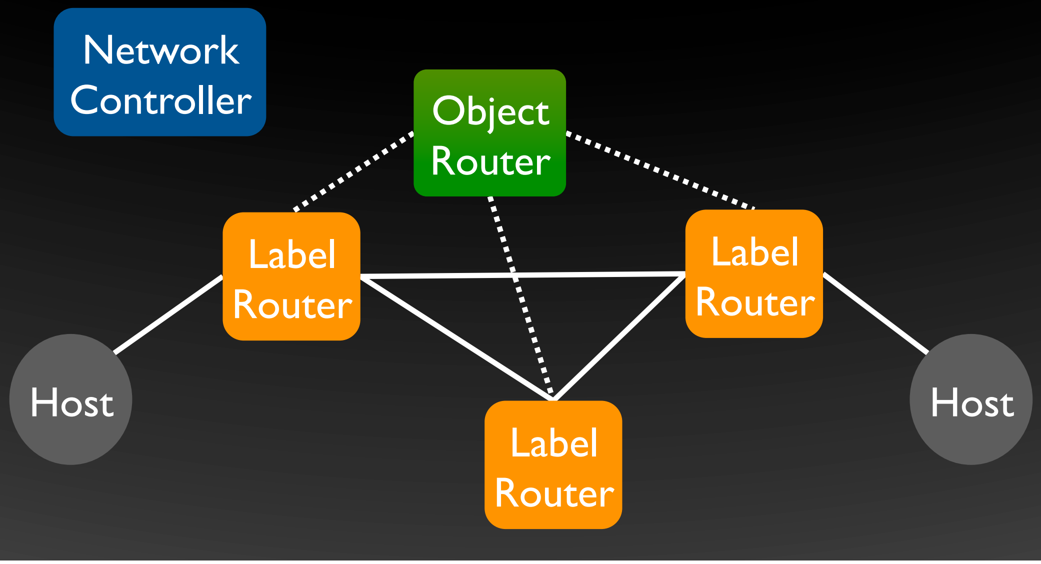
Flow Label

SocketID

In-Network support

Network Controller – NOX application:
topology, host, object management

Object Router Label Router – Modified OpenFlow software switch for proportional split routing/resolution



“Evaluation”

Demos

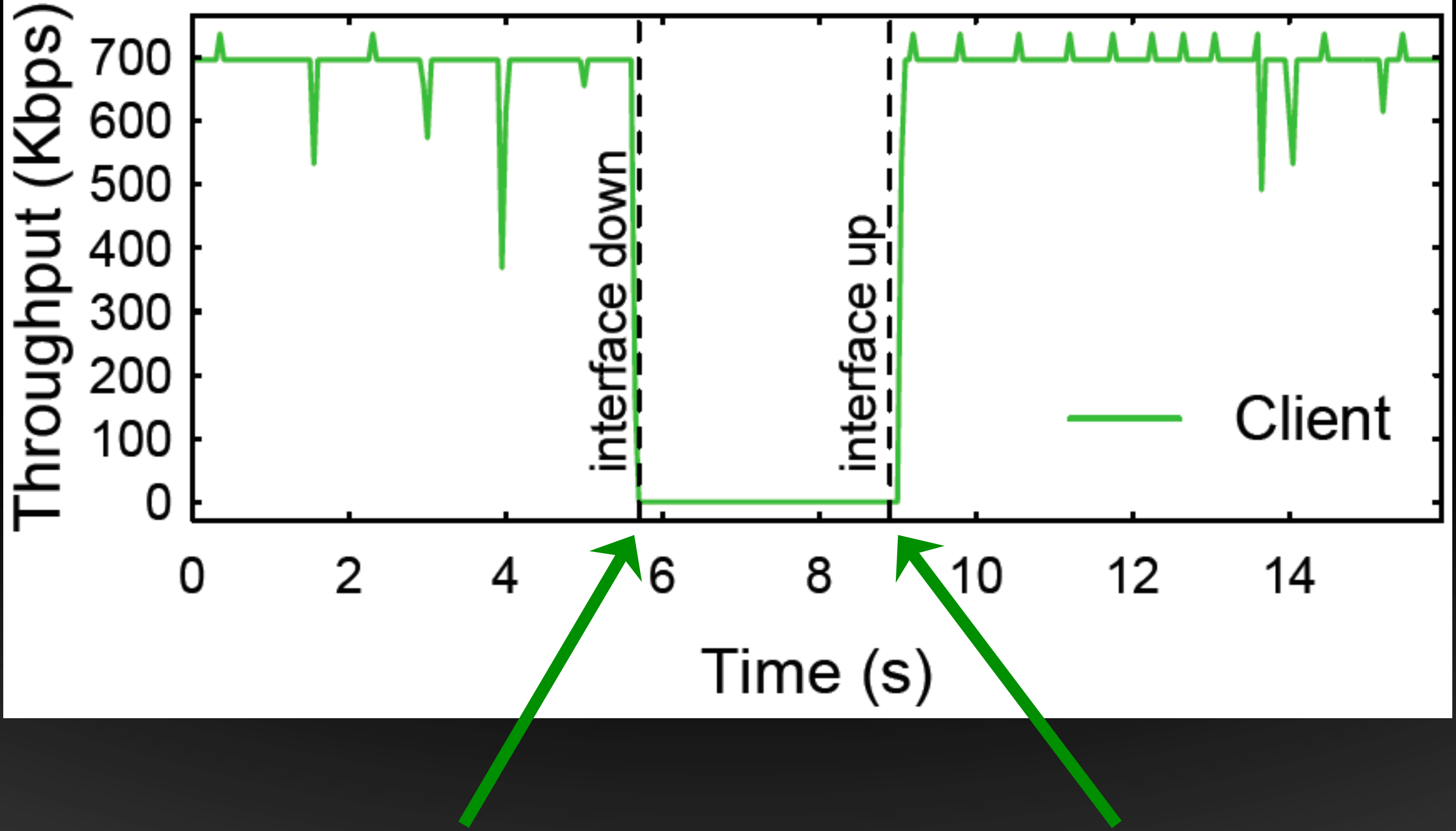
- Load Shedding:
  - Call close() on connections
  - Subsequent packets get FAIL, then reconnect
- Client mobility

SNS 9 TFTP server connections

SNS 11 TFTP server connections

```
[illegible]
```

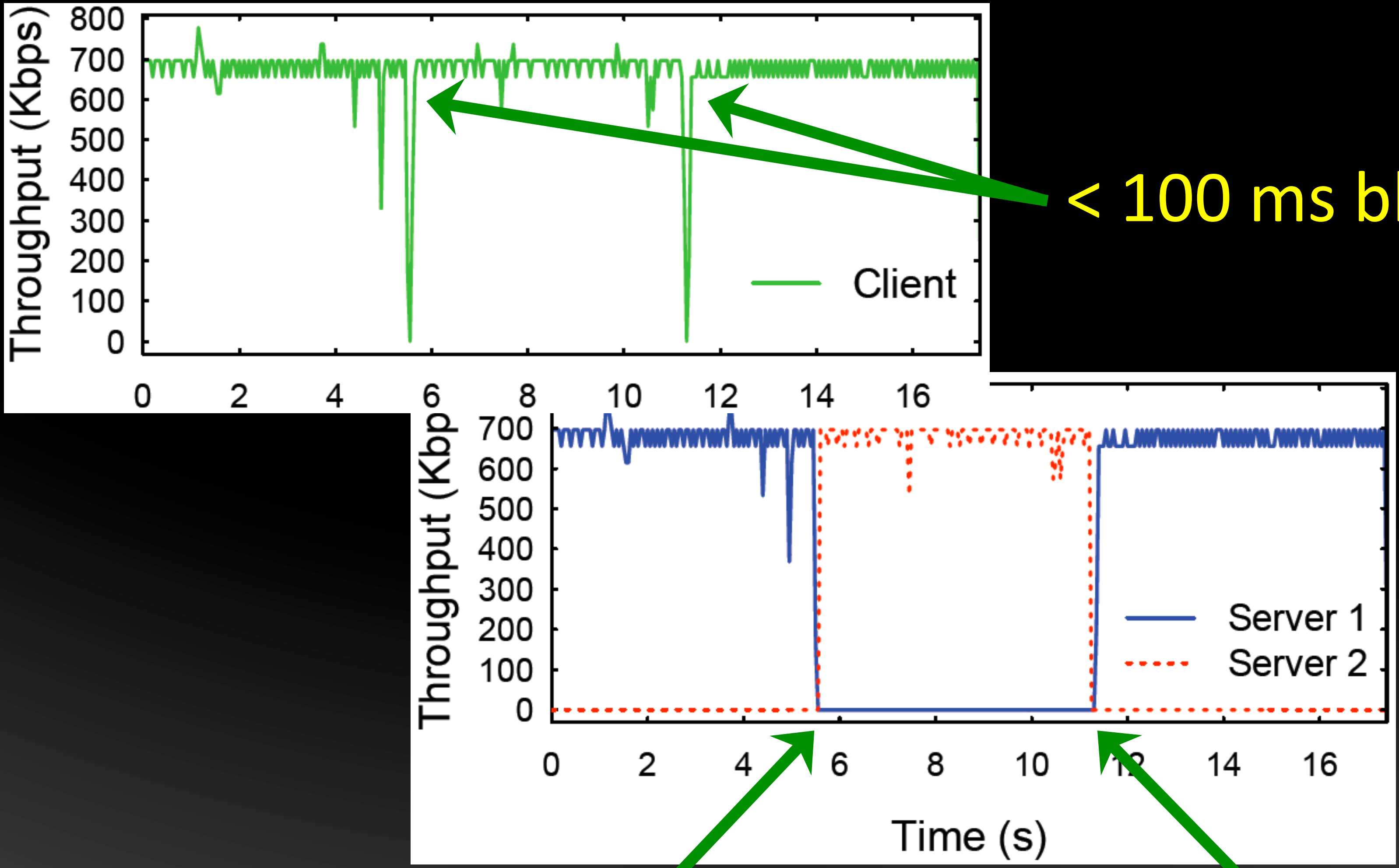
TFTP transfer with Client Mobility



Client Leaves

Client Reconnects (RSYN)

TFTP transfer with Failover

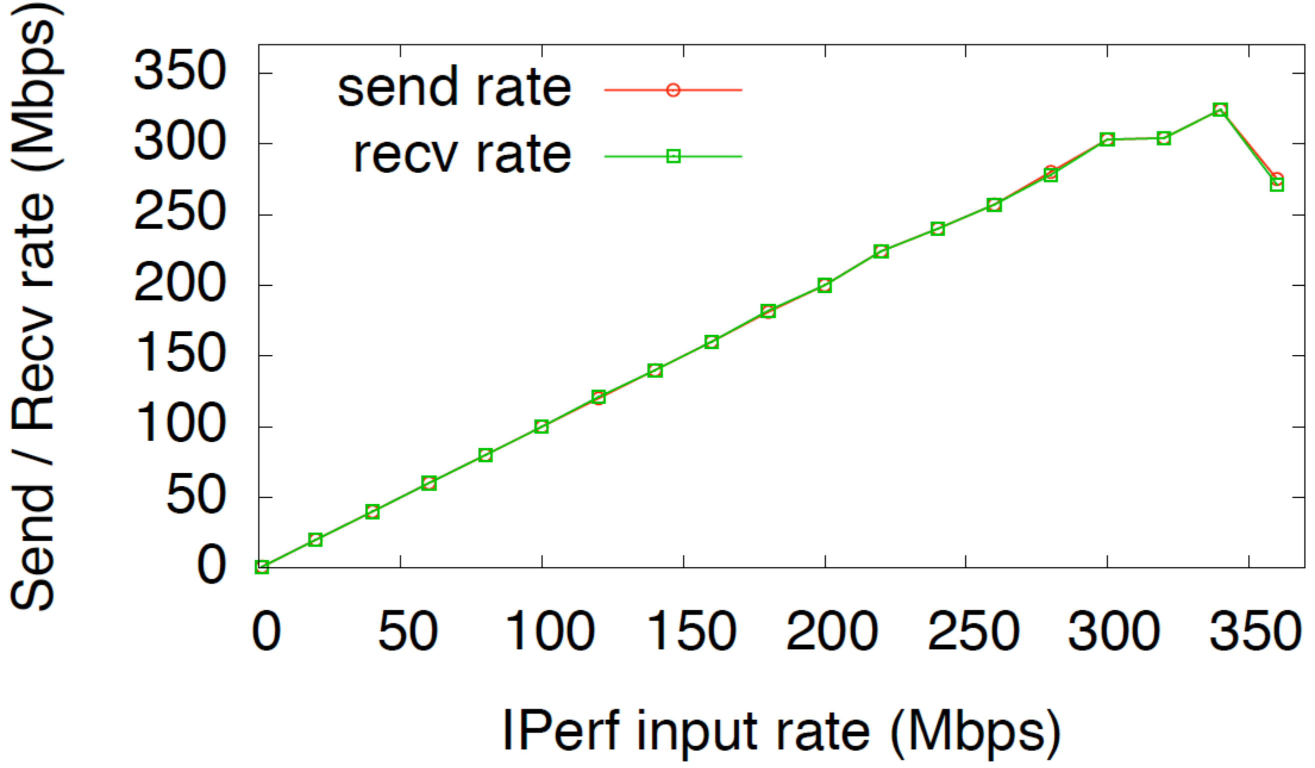


< 100 ms blip

Server 1 (FAIL)

Server 2 (FAIL)

Current throughput



Current implementation is both user/kernel space.
Ongoing development to either/or.

Service-centric networking

- Moderate vision: Can network support aid self-configuration for replicated services?
- Big vision: Should "service-centric networking" become the new thin waist of Internet?

SCAFFOLD rethinks:

1. Naming exposed to network and applications
2. Extent of host-network integration
3. Role of dumb/stateless network vs. end-hosts

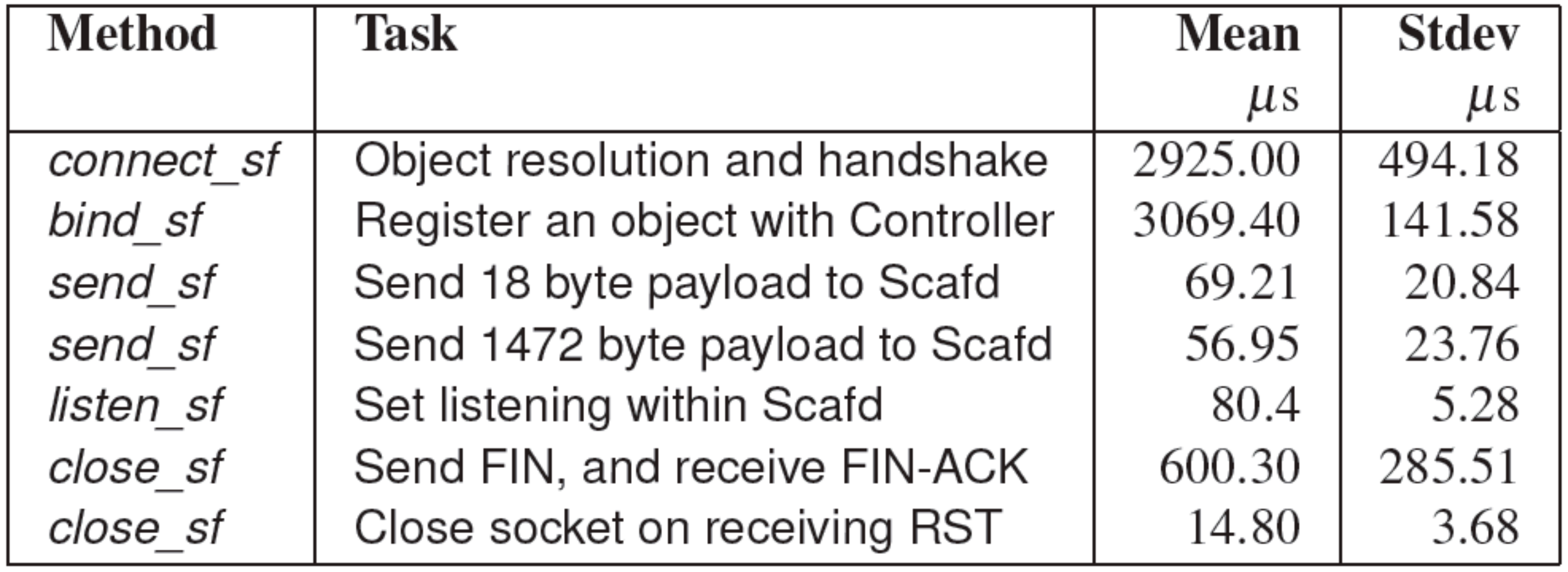
Service-centric networking with SCAFFOLD

Michael J. Freedman

Princeton University

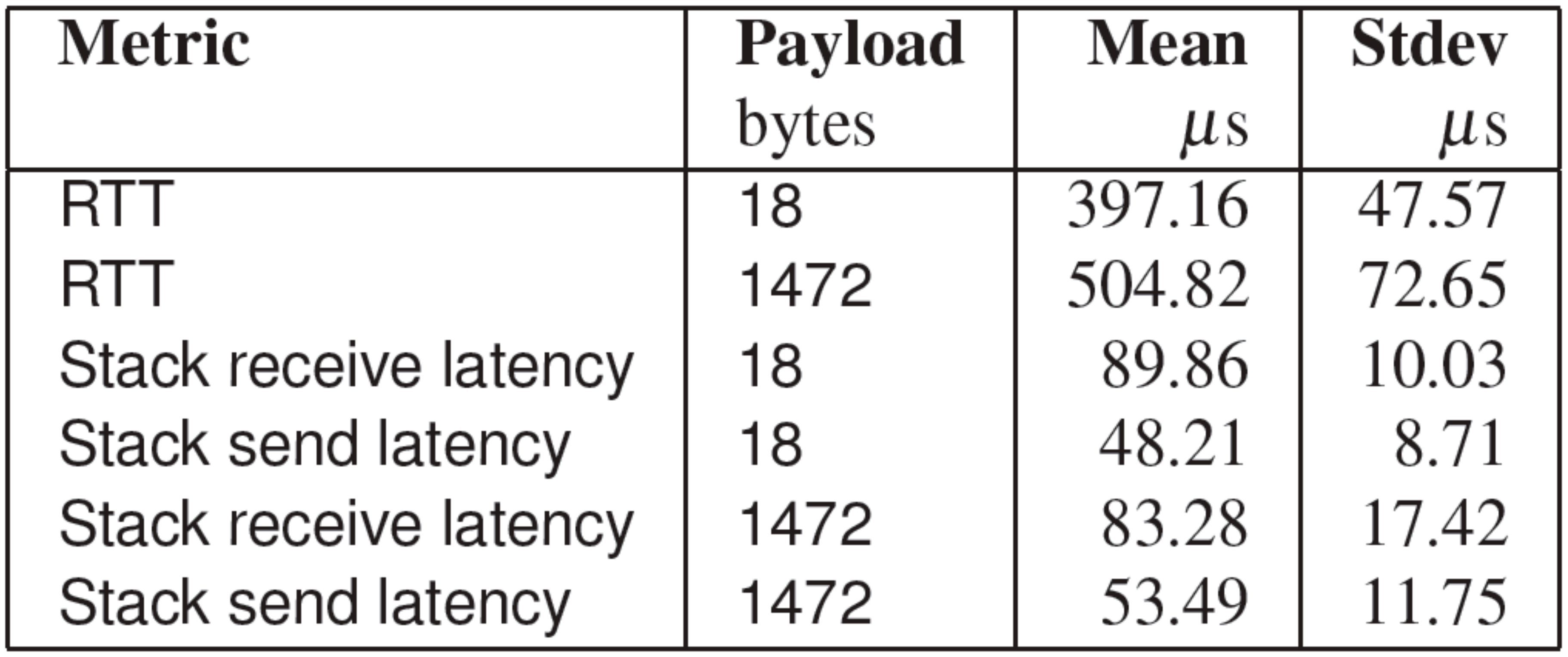
with Matvey Arye, Prem Gopalan, Steven Ko, Erik Nordstrom, Jen Rexford, and David Shue

Latency of API calls



Method	Task	Mean	Stdev
		μs	μs
connect_sf	Object resolution and handshake	2925.00	494.18
bind_sf	Register an object with Controller	3069.40	141.58
send_sf	Send 18 byte payload to Scafd	69.21	20.84
send_sf	Send 1472 byte payload to Scafd	56.95	23.76
listen_sf	Set listening within Scafd	80.4	5.28
close_sf	Send FIN, and receive FIN-ACK	600.30	285.51
close_sf	Close socket on receiving RST	14.80	3.68

Network vs. stack latency



Metric	**Payload**	**Mean**	**Stdev**
	bytes	μs	μs
RTT	18	397.16	47.57
RTT	1472	504.82	72.65
Stack receive latency	18	89.86	10.03
Stack send latency	18	48.21	8.71
Stack receive latency	1472	83.28	17.42
Stack send latency	1472	53.49	11.75

Related Work

NewArch	i3	LNA	DONA	LISP	HIP	CCN	SCAFFOLD
Paradigm	Object	Object	Object	Host	Host	Content	Object
Layer	3O	4	3/4	3	4	3/4	3/4
Anycast	Hash	Res	Prox	No	No	Mcast	Res
Resolution	DHT	EB	Routed	EB	Rdz	DDiff	SRefine
Migration	Yes	Yes	Yes*	Yes	Yes	Yes*	Yes
Failover	Yes	Yes	Yes	No	No	Yes	Yes

Related Work

	SCAFFOLD	SPAIN	PortLand	VL2
Topology	Arbitrary	Arbitrary	Fat-tree	Fat-tree
Multipath	Any	Many	ECMP	ECMP
Migration	Yes	Yes*	Yes*	Yes*
Failover	Yes	No	No	No
Traffic Engineering	Arbitrary	Oblivious	Oblivious	Oblivious
Server Selection	Yes	No*	No*	No*
Use CoTS?	No	Yes	No	Yes
End-host Mod	Yes	Yes	No	Yes