Network Programming Languages

Robert Soulé
University of Lugano

Languages

- A programming language provides ***abstractions*** and ways to ***compose*** those abstractions
- The programming languages you are familiar with are ***models*** of computer systems
- They provide abstractions for data and computation

Abstractions

	abstractions	compositions
Assembly languages	addresses, registers, instructions, labels	sequences of instructions
Procedural languages	booleans, arithmetic, loops, arrays, procedures	sequences of statements, procedure calls
OO languages	objects, methods, fields, classes	method invocation, inheritance
Functional languages	first-class functions, algebraic data types	function application, type constructors

Domain Specific Languages

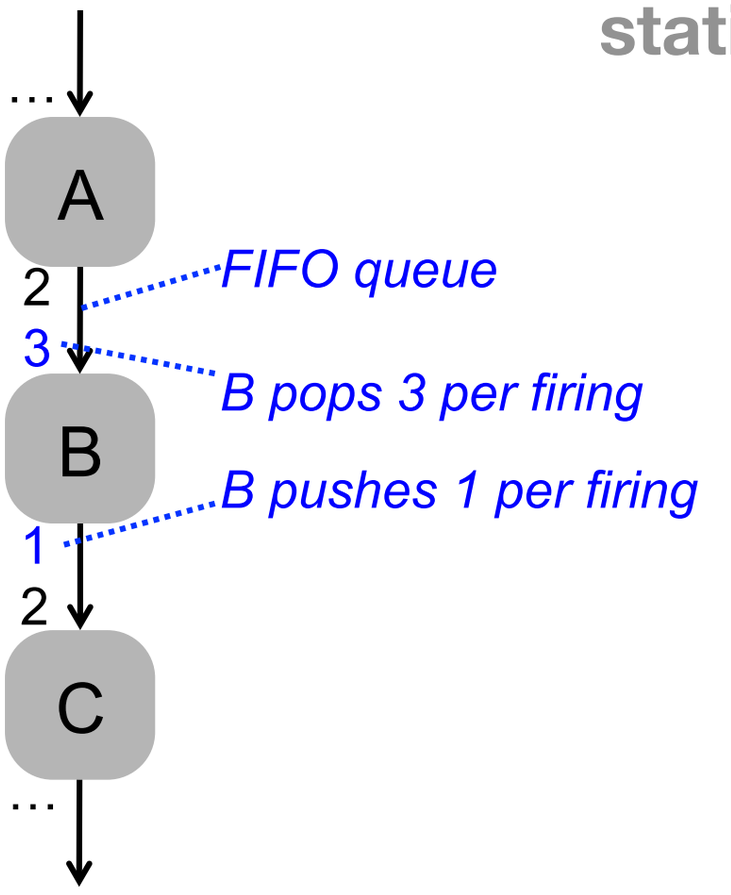
- A *general-purpose* language provides abstractions for modeling computation
- A *domain-specific* language provides abstractions for other domains

Benefits of a DSL

- Can enable optimizations
- Can ensure program correctness
- Provide abstractions tailored to a particular domain
- Allow programmers to use emerging technologies

Enabling Optimizations

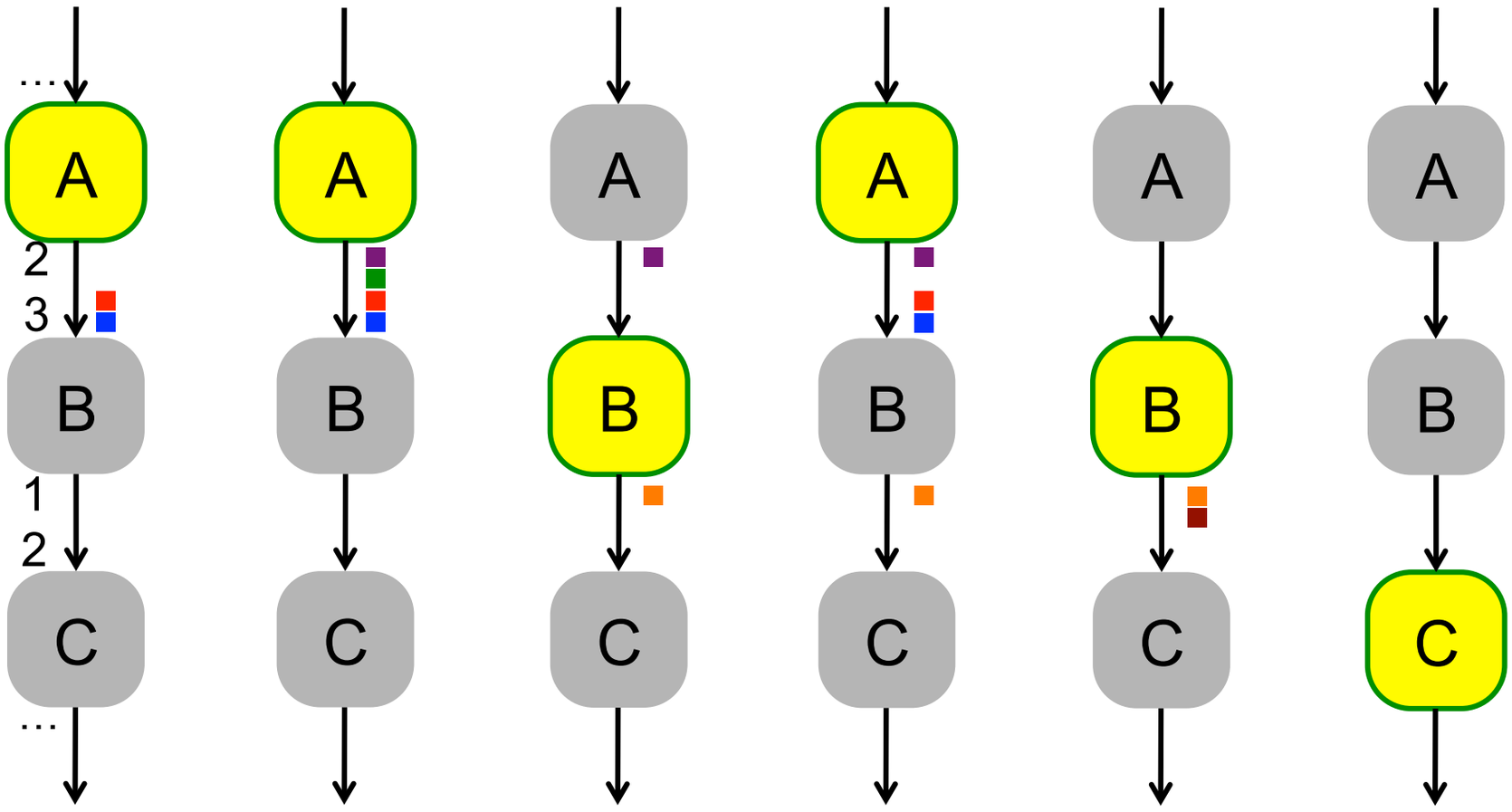
StreamIt: a streaming language designed for static optimization




```
float->float pipeline ABC {
  add float->float filter A() {
    work pop … push 2 {
      /*details elided*/ }
  }
  add float->float filter B() {
    work pop 3 push 1 {
      /*details elided*/ }
  }
  add float->float filter C() {
    work pop 2 push … {
      /*details elided*/ }
  }
}
```

Statically known push/pop rates for every operator.

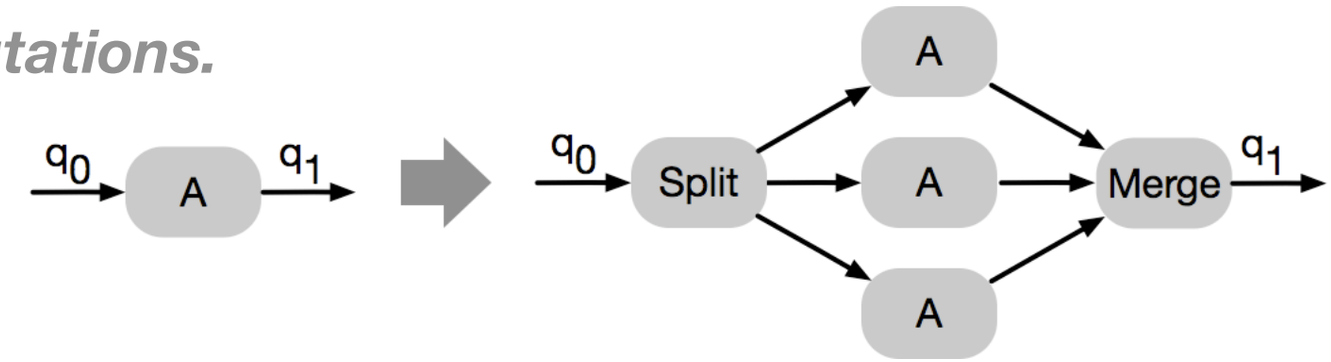
Enabling Optimizations



Steady-state schedule means fixed queue sizes and scalarization.

Program Correctness

Parallelize computations.



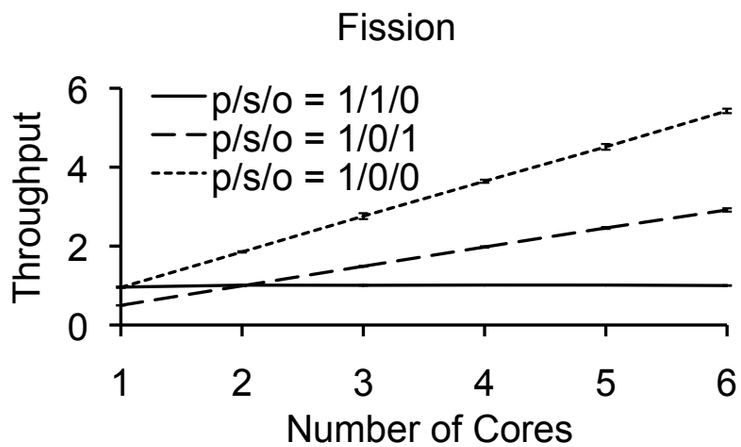
Safety

- No state or disjoint state
- **Merge in order, if needed**

Variations

- Round-robin (no state)
- **Hash by key (disjoint state)**
- Duplicate

Profitability

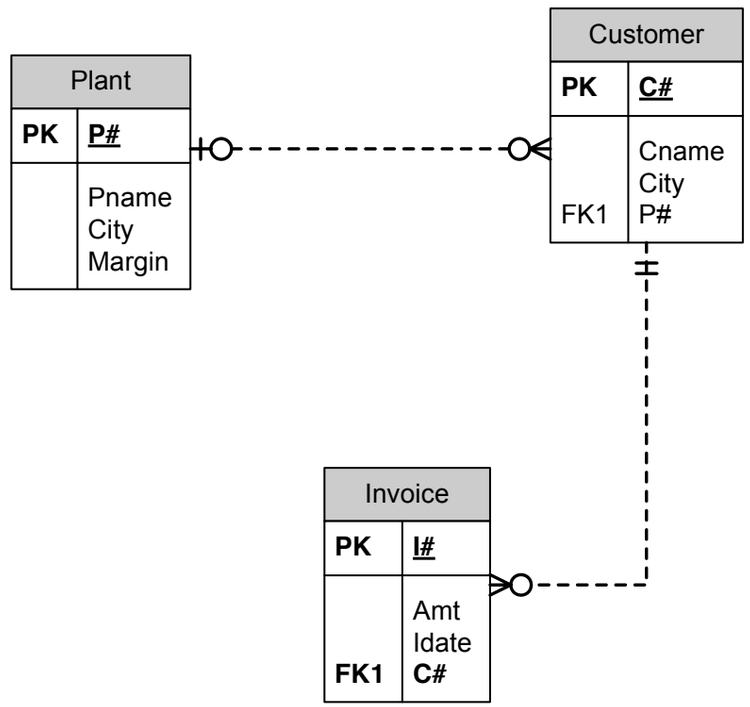


Example

- MapReduce languages

Abstractions For A Particular Domain

```
select *
from customer, plant
where plant.pid =
customer.pid
and city="milan"
```



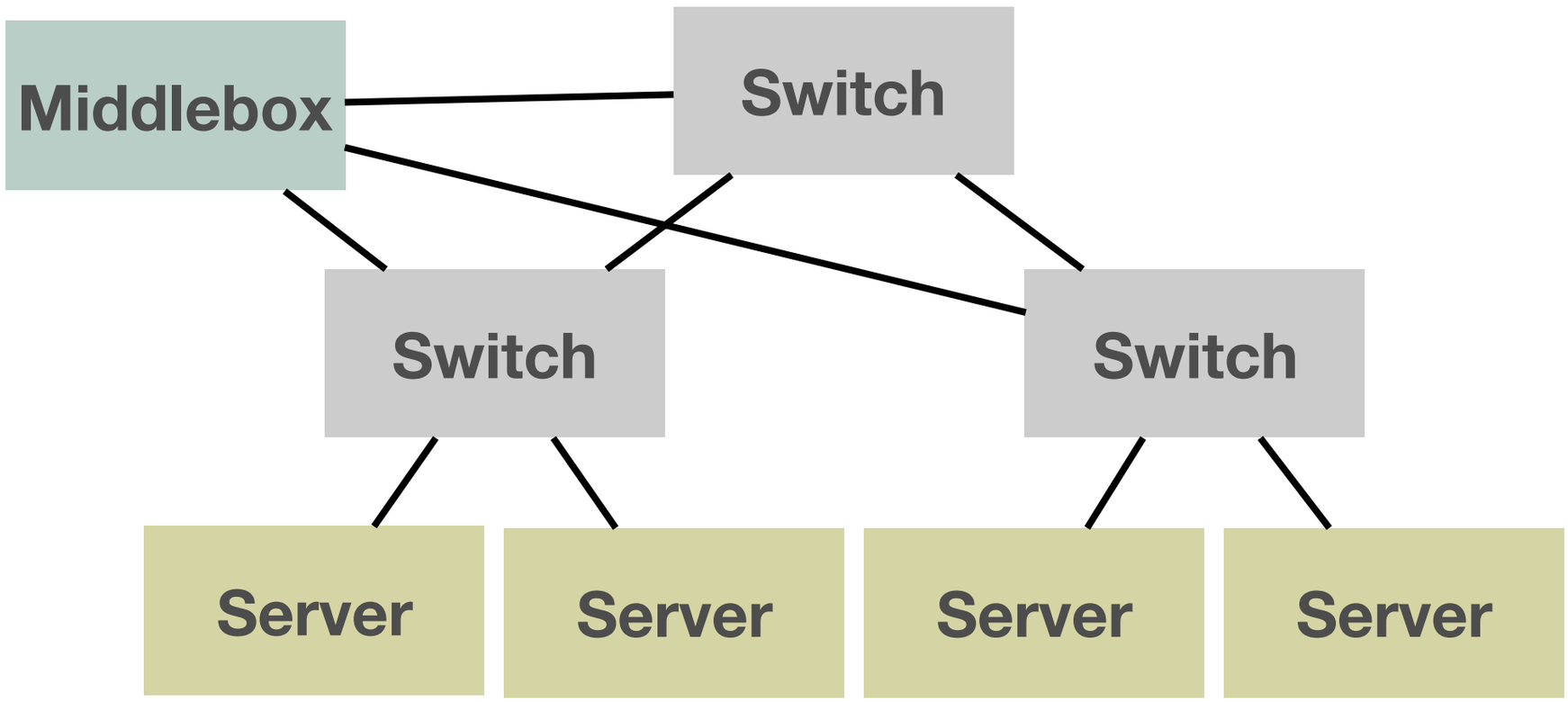
Emerging Technologies

Example of DSLs

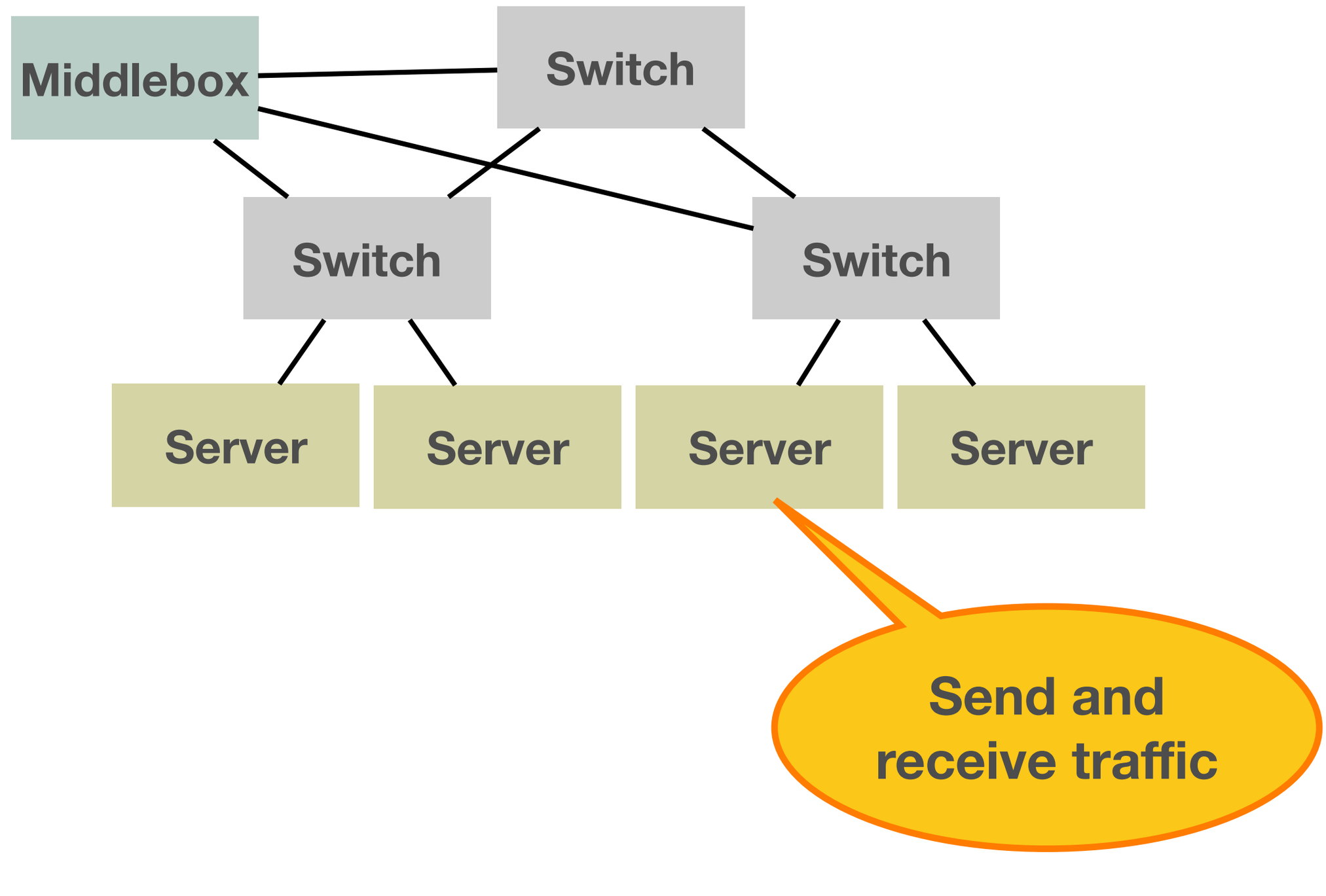
	abstractions	**compositions**
SQL	relations, tuples, queries	joins, selection, projection
make	files, build rules	dependencies
lex	characters, strings	sequences, alternation (|), repetition (*)
yacc	tokens, nonterminal symbols	grammar rules
OpenSCAD	shapes	union, intersection, linear transformations

What Abstractions Do We Need for Networking?

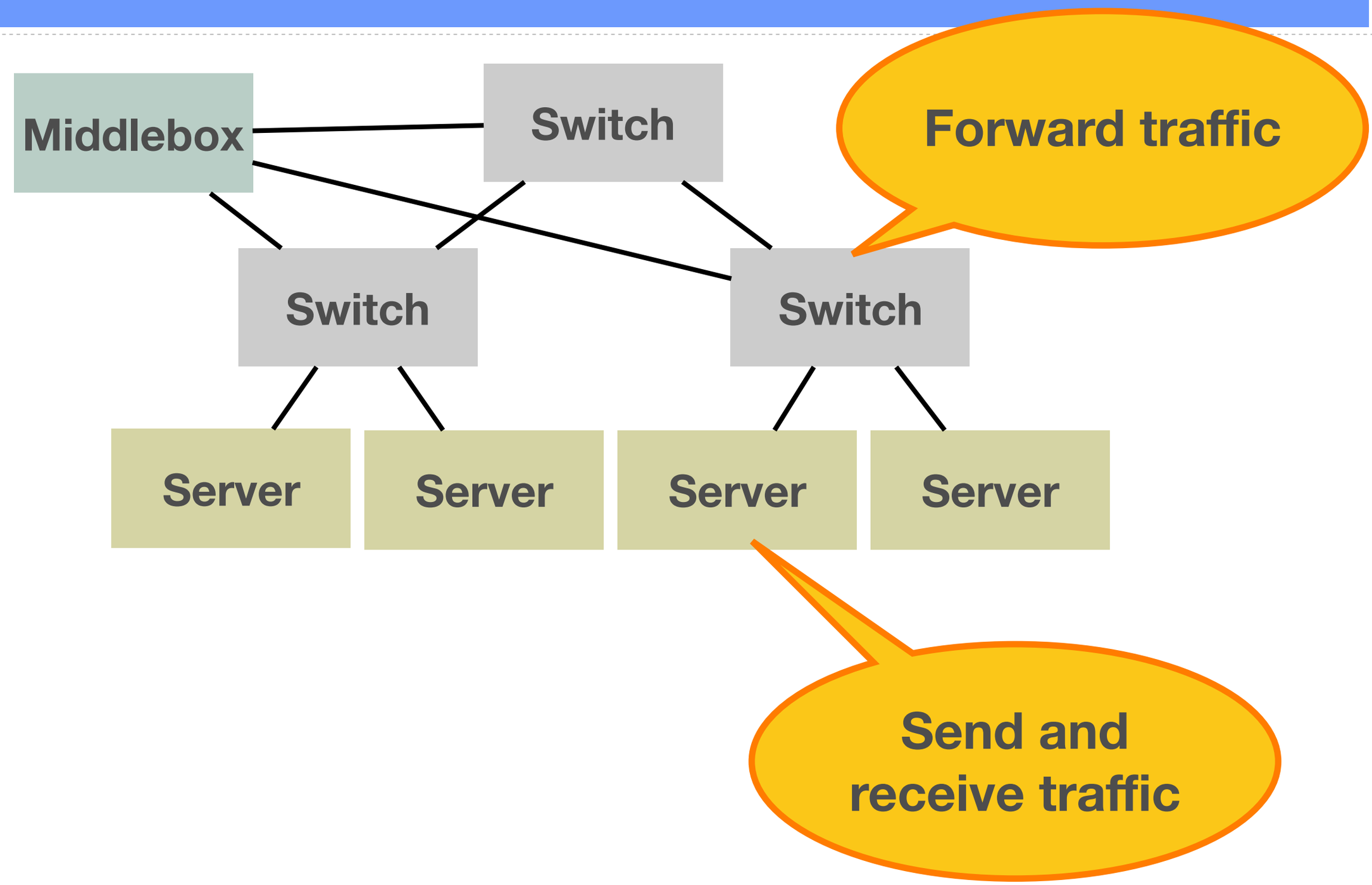
Network Devices



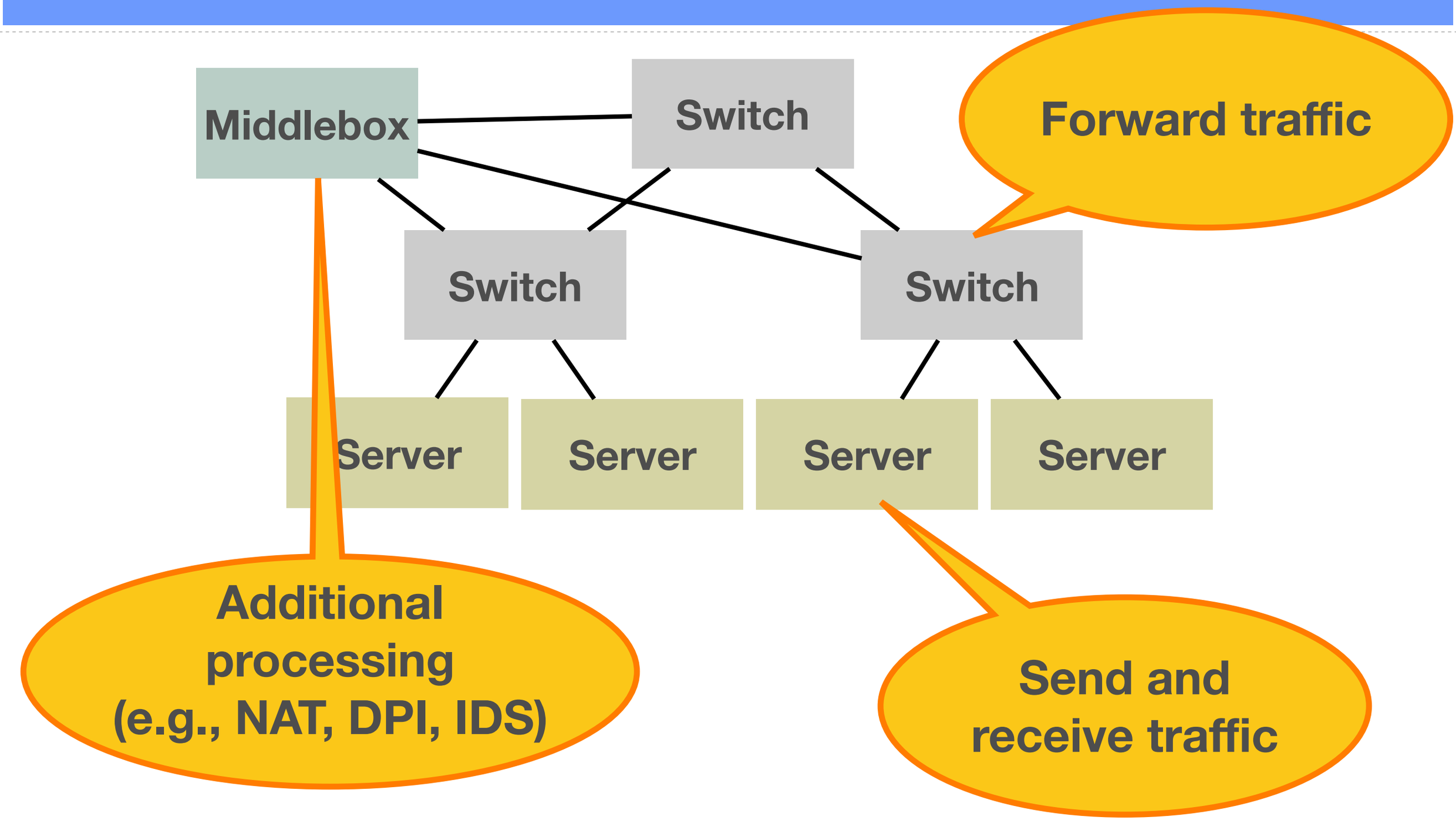
Network Devices



Network Devices



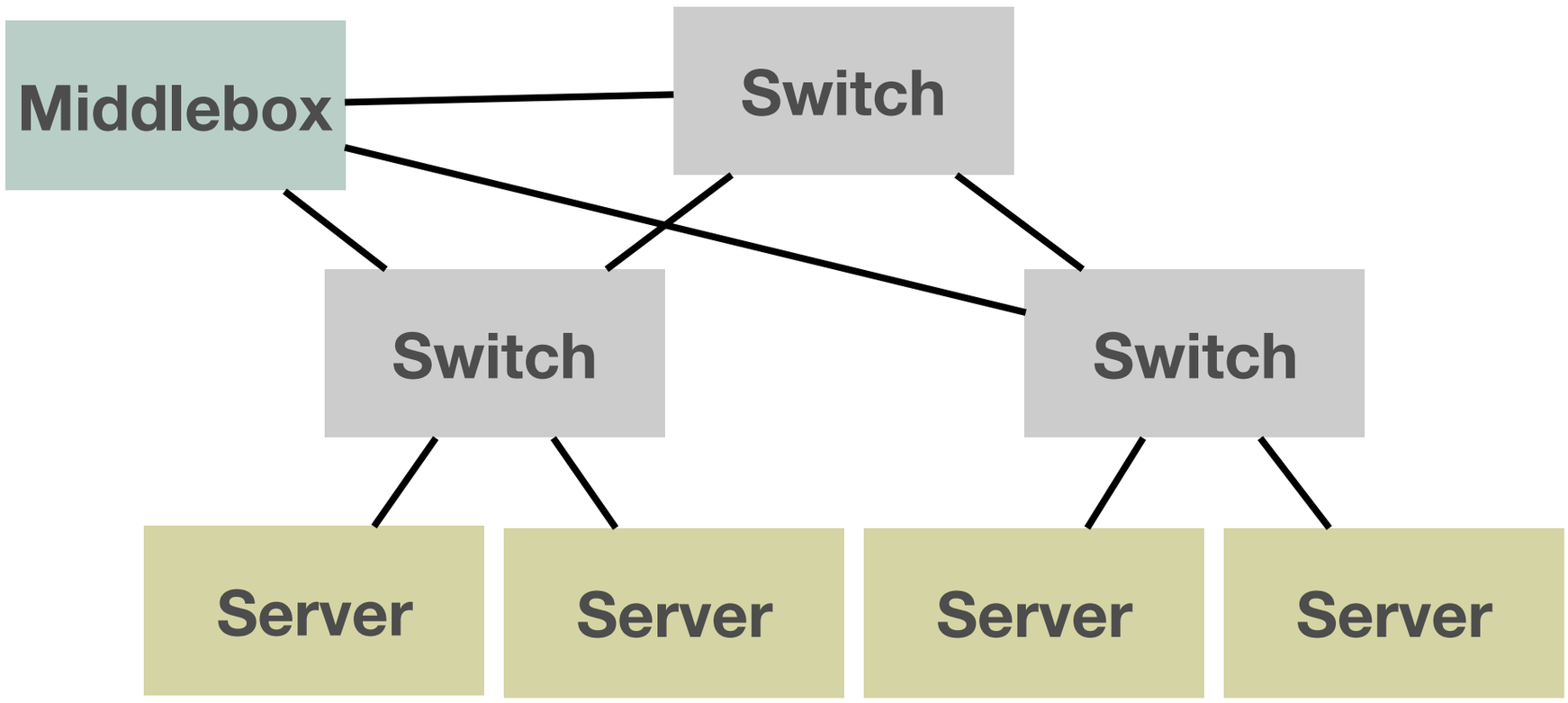
Network Devices



What Abstractions Do We Need for Networking?

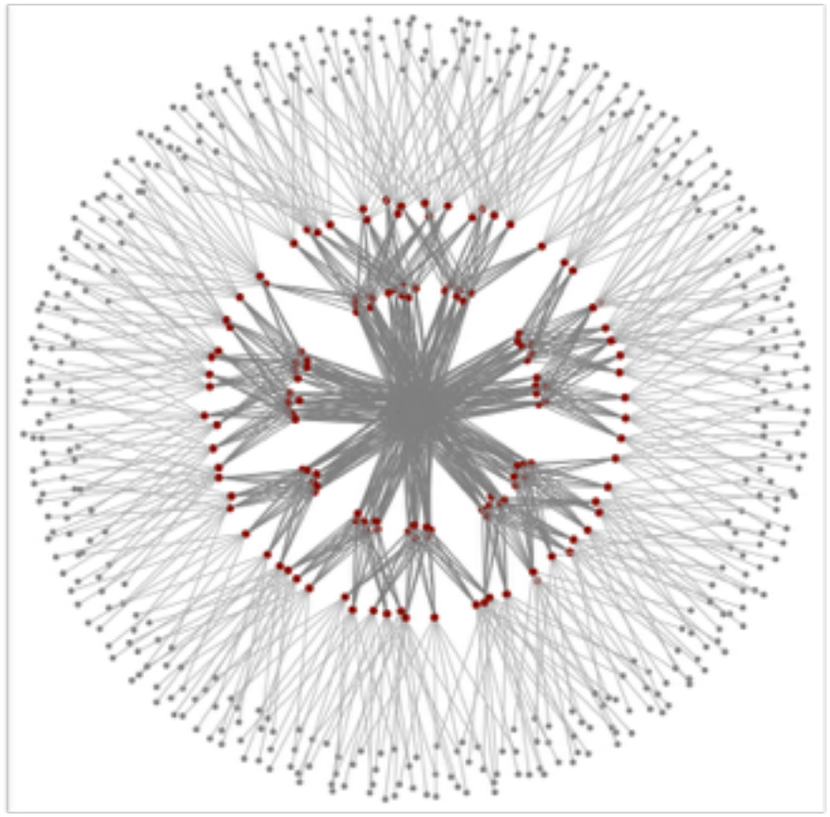
- It depends on what we want to do.
  - Authorization needs *principals* and *privileges*
  - DPI needs *patterns* and *packet payload*
  - State queries might need *tables and relations*
- What do we need to model forwarding?
- What do we need to model switch hardware?

Device Configuration



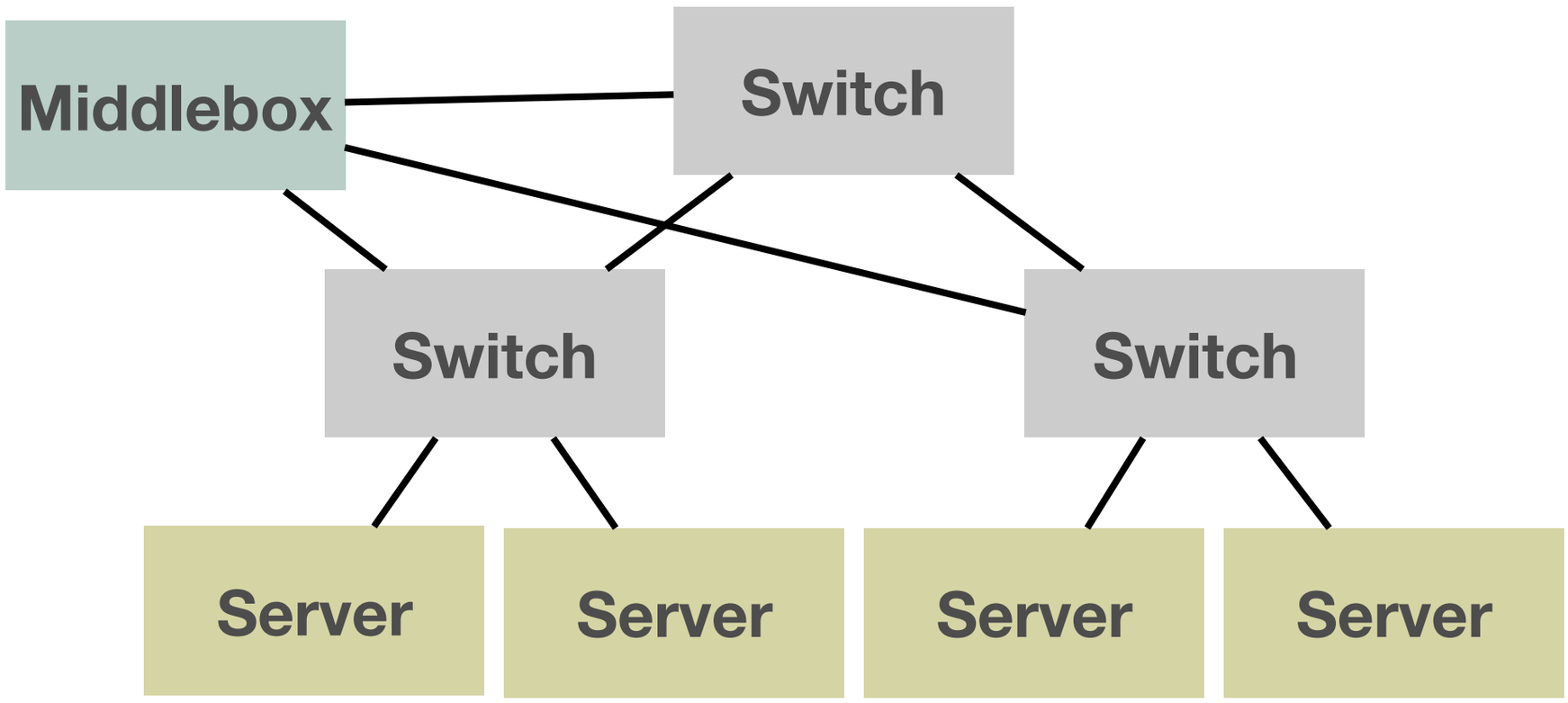
- SSH to switch or router
- Configure with a vendor-specific DSL
- Route traffic to middle box for additional processing

Device Configuration



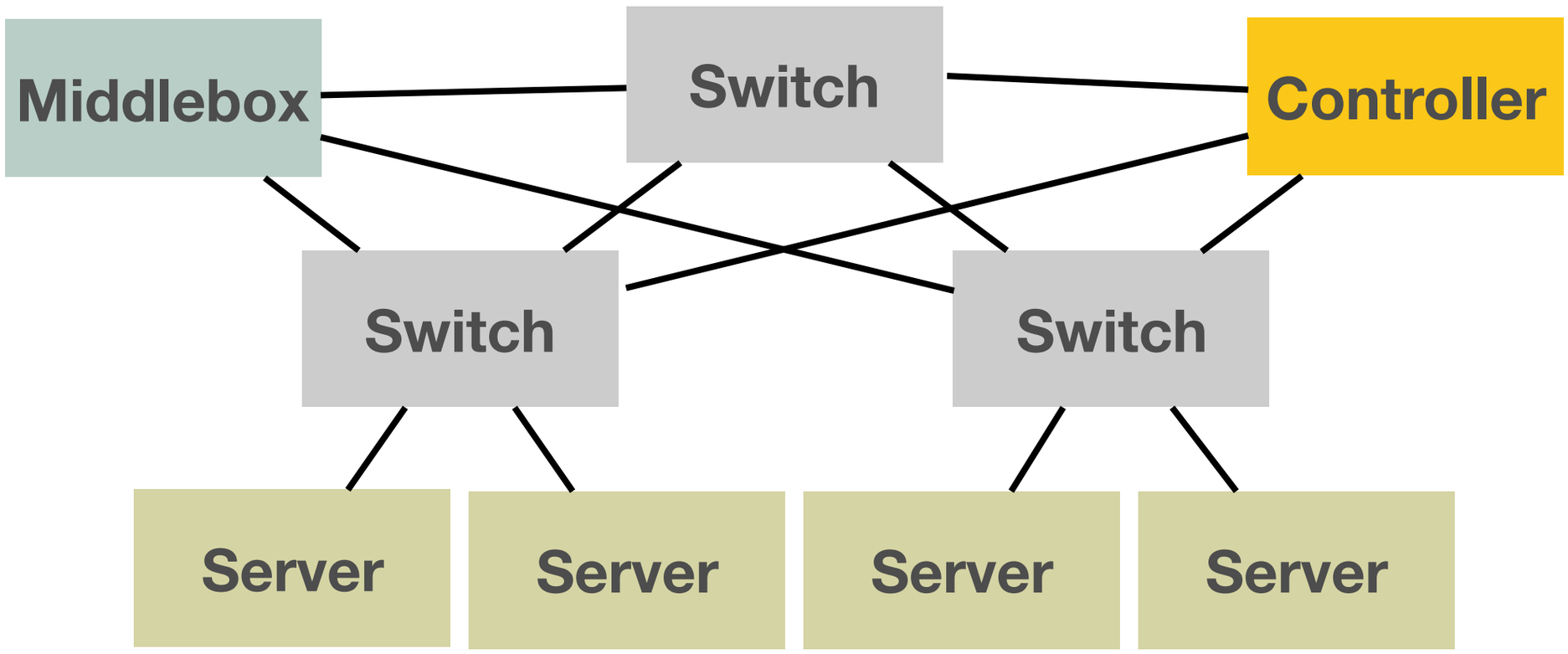
- Campus network has 100s of switches
- Datacenter network has 1000s of switches
- Hard to manage! Hard to verify!

Software Defined Networking



- Standardization: switches support a vendor-agnostic, open API
- Off-device control: software program manages the switches
- Big idea: separate control logic from the data plane

Software Defined Networking



- Standardization: switches support a vendor-agnostic, open API
- Off-device control: software program manages the switches
- Big idea: separate control logic from the data plane

What Abstractions for SDN?

- OpenFlow provides abstractions to *match headers, forward/drop packets,* and *collect simple statistics*
  - Often referred to as "**match-action**" abstraction in literature
- Is this a good abstraction?
  - Models the "basic functionality" of a switch
  - Amenable to efficient hardware implementations
- Is it enough? (i.e., what else does the network do?)

Network Languages

- Query network state (e.g., Sophia)
- Authorization (e.g., NAL, SAFE)
- Overlays (e.g., P2, MACE, Teacup)
- Forwarding/Routing (e.g., Frenetic, Pyretic, Maple)
- Traffic Engineering (e.g., Merlin)
- Program switch hardware (e.g., P4)
- Implement routers (e.g., Click)
- …and many more!

SDN (control plane) languages

- FatTire
- Flog
- FlowLog
- FML
- Frenetic
- HFT
- Maple
- Merlin
- nlog
- NetCore
- NetKat
- Nettle
- Procura
- Pyretic
- Kinetic
- and more….

Programming Paradigms

- FatTire
- Flog
- FlowLog
- FML
- Frenetic
- HFT
- Maple
- Merlin
- nlog
- NetCore
- NetKat
- Nettle
- Procura
- Pyretic
- Kinetic
- and more….

Imperative
Functional
Logic
DataFlow
FRP

Imperative

- A programming paradigm in which programs are written as a sequence of statements that may modify state
- Models the “Von Neumann architecture”
- Examples: C, Java, Pyretic

Imperative Example

```
def round_robin(self,pkt):
  self.policy = if_(match(srcip=pkt['srcip']),
                    modify(dstip=self.server),
                    self.policy)
self.client += 1
self.server = self.servers[self.client % m]
```

Pyretic: round-robin load balancer.

Logic-Based Languages

- A programming paradigm in which programs are based on formal logic
- Programs are often written as a set of rules in the form of a Horn clause (i.e., a disjunction of literals with at most one negated literal)
- Examples: Prolog, Datalog, Flowlog

Logic Example

```
ON packet_in(p):
  DO forward(new) WHERE
     learned(p.locSw,new.locPt,p.dlDst);
```

$$\forall p, new\,.\,forward(new) \Longleftarrow$$
$$learned(locSw(p), locPt(new), dlDst(p))$$
$$\wedge\, packet\_in(p)$$

FlowLog: forwarding example and rule clause.

Functional Languages

- A programming paradigm in which computation is based on mathematical functions (often avoiding mutable state)
- Programs are typically written as compositions of functions
- Examples: Lisp, Racket, OCaml, Haskell, Frenetic

Functional Example

```
def web_query():
   return \
(Select(sizes) *
 Where(inport_fp(2) & srcport_fp(80))) *
 Every(30))
```

Frenetic (2011): amount of incoming web traffic every 30 seconds.

Dataflow Languages

- A programming paradigm in which programs are modeled as a directed graph of operations
- Programs are usually written as names of operations and arrows
- Examples: SISAL, nesC, Click

DataFlow Example

```
// Declare three elements ...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// ...and connect them together
src -> ctr;
ctr -> sink;
```

FIGURE 3.1—A Click-language description of the trivial router of Figure 2.1 (page 15).

Functional Reactive Programming (FRP)

- Combines asynchronous dataflow programming (i.e., reactive programming) with functional programming
- Distinguishes between *discrete* and *continuous* events. Programs manipulate streams of values.
- Examples: Elm, Flapjax, Yampa, Nettle, Frenetic (2010)

FRP Example

```
def monitor_sf():
  return(Filter(inport_p(2) & srcport_p(80)) |o|
         GroupByTime(30) |o|
         SumSizes())
def monitor():
  stats = Apply(Packets(), monitor_sf())
  print_stream(stats)
```

Frenetic (2010): amount of incoming web traffic every 30 seconds.

Software Defined Networking

Technological Influences

- Control vs. Data Plane
- Network Virtualization
- Active Networks

Separate Control From Data

Control and Data

- *Control plane:* includes system configuration, management, routing table information, etc. Also called the “signaling” of the network.
  - Control plane packets are processed by the switch/router.
- *Data plane:* forwards traffic to the next hop.
  Also called the “forwarding” plane.
  - Data plane packets pass through the switch/router.

In-Band Signalling

- Send metadata in the same band or channel
- Certain frequencies (e.g., 2600 Hz) could reset phone trunk lines, route calls, etc.
- Insecure, end users have access to control signals

Out-of-band Signaling

- Send metadata in a separate, dedicated channel
  - In telephone networks, introduced in the 1970s (SS6) and 1980 (SS7)
- Example: In 1981, AT&T introduced the Network Control Point (NCP)
  - A centralized server that could send commands to switches
  - Reduces expenditures (shorter circuit holding time, quickly determine busy/idle status)
  - Allows rapid introduction of new services (only implemented once at server)

Benefits of Central Control

- Network-wide vantage point
  - Can directly observe network wide behavior
- Independent evolution of infrastructure, data, and services
  - Services and resource allocation decisions can be made based on customer data, network load, etc.

Network Virtualization

Network Virtualization

- Represent one or more *logical* topologies on the same *physical* infrastructure
- Many different instantiations
  - VLANs
  - Slicing, etc.
  - VMWare, Nicira, etc.

Benefits

- Sharing
  - Multiple logical routers on single platform
  - Resource isolation in CPU, memory, bandwidth, forwarding tables
- Customizability
  - Can easily modify routing and forwarding software
  - General purpose CPUs for the control plane
  - Network processors and FPGAs for data plane

Examples

- Tempest: Switchlets (1998)
  - Separation of control framework from switches
  - Virtualization of the switch
- VINI: A virtual network infrastructure (2006)
  - Virtualization of the network infrastructure
- Cabo: Separates infrastructure, services (2007)

Active Networks

Active Networks

- Goal: create networking technologies that can *evolve* and support *application-specific customization*
- Key idea: It would be easier to support these goals if the network were programmable
- Realization: switches perform custom computation on packets
  - Examples: trace program running at each router, firewalls, proxies, application services

Active Networks History

- Vigorous area of research when DARPA began funding (1994-1995)
- A confluence of ideas from O.S., P.L., Networking, Distributed Systems

Two Approaches

- Capsules (“integrated”)
  - Packets carried procedures. Active nodes evaluate content carried in packets.
  - Code dispatched to execution environment.
- Programmable Switches (“discrete”)
  - Custom processing functions run on the router
  - Packets routed through programmable nodes
  - Program depends on packet header

What went wrong?

- Wrong time
  - No clear application
  - Hardware was expensive
- Missteps
  - Security, special languages for safe code, packets carrying code
  - End user as programmer (vs. network operator)
  - Interoperability

What went right for SDN?

- Motivating use case (virtualization):
  - Nicira's Network Virtualization Platform
- Pragmatic
  - Minimal functionality (API) that can be efficiently implemented on a variety of hardware
- Industry support

Reading List

- Nick Feamster, Jennifer Rexford, and Ellen Zegura, “The Road to SDN: An Intellectual History of Programmable Networks”, ACM SITCOM CCR, April 2014

- Jonathan M. Smith and Scott M. Nettles, “Active Networking: One View of the Past, Present, and Future”, IEEE Transactions on Systems, Man, and Cybernetics, March 2003