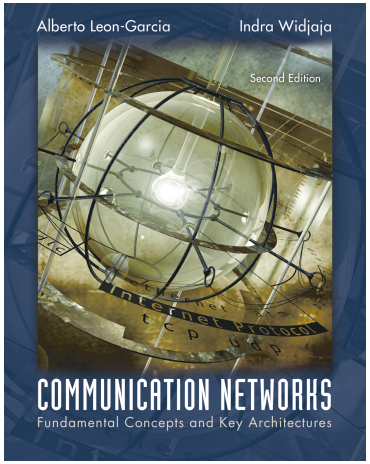
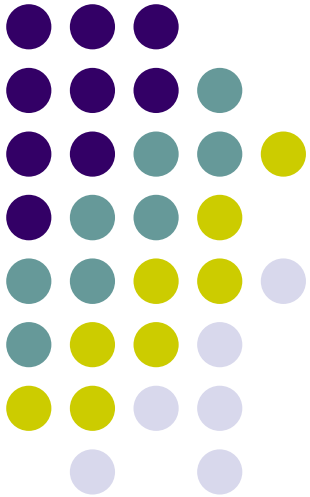
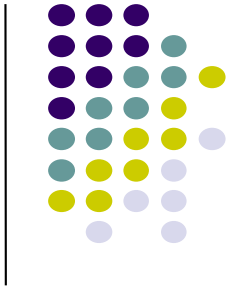
Packet-Switching Networks



Network Layer and End-to-End Argument
pp 460-471
Structure of Packet Switches & Routers

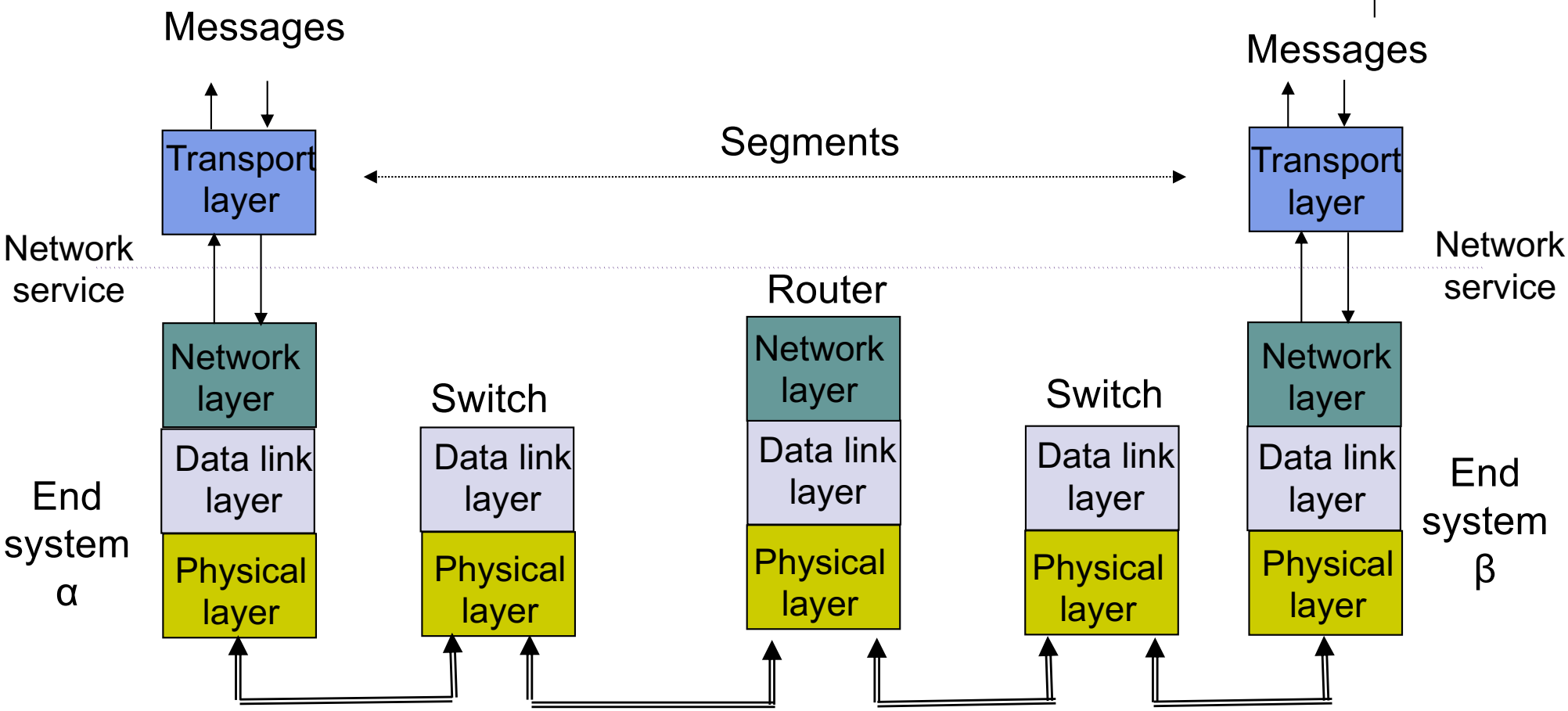


Network Layer



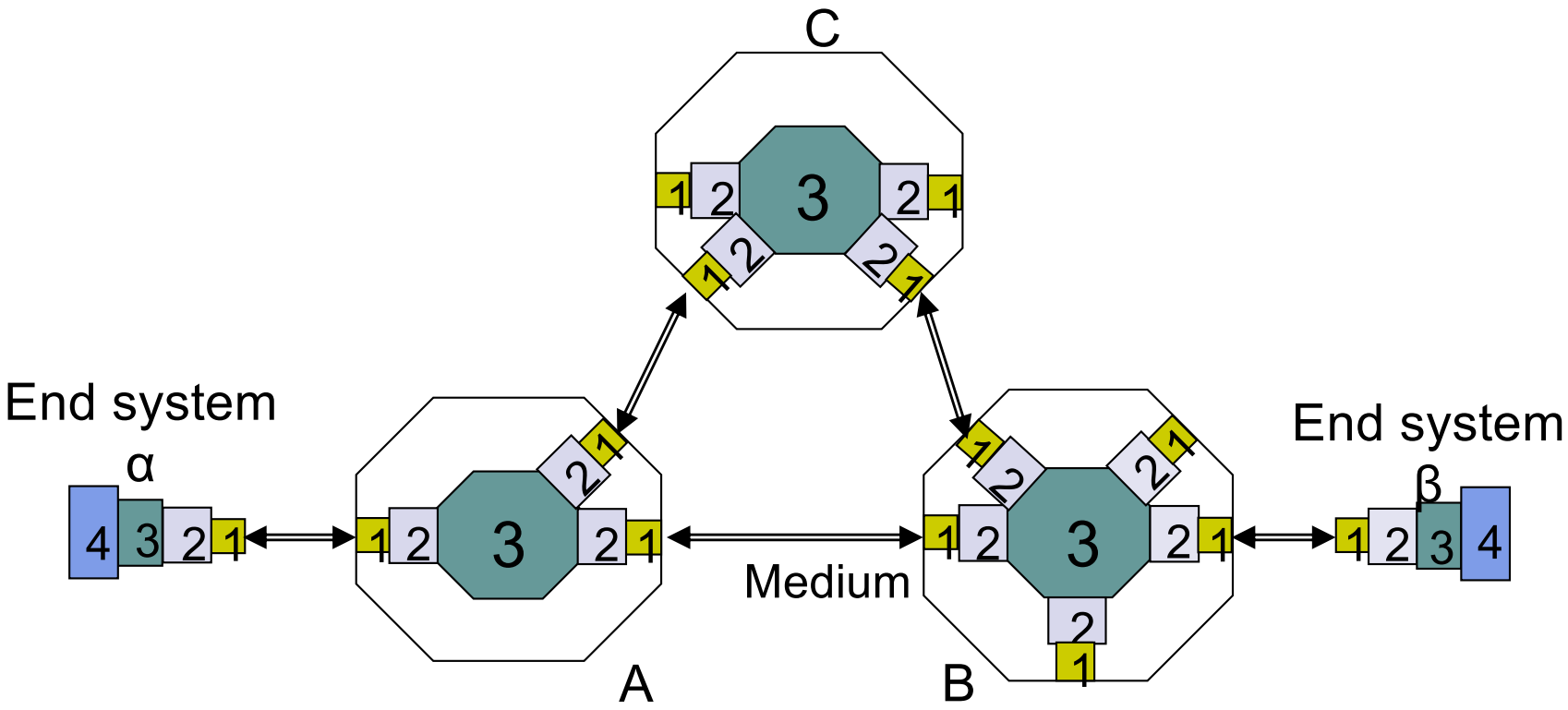
- Network Layer: the most complex layer
  - Requires the coordinated actions of multiple, geographically distributed network elements (switches & routers)
  - Must be able to deal with very large scales
    - Billions of users (people & communicating devices)
  - Biggest Challenges
    - Addressing: where should information be directed to?
    - Routing: what path should be used to get information there?

Layered View

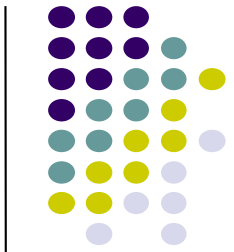


- What services should network layer offer to transport layer
- Connection-oriented service or connectionless service?
- Best-effort or guaranteed delay/loss transfer?

Complexity at the Edge or in the Core?

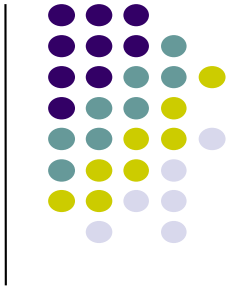


The End-to-End Argument for System Design



- An end-to-end function is best implemented at a higher level than at a lower level
  - End-to-end service requires all intermediate components to work properly
  - Higher-level better positioned to ensure correct operation
- Example: stream transfer service
  - Establishing an explicit connection for each stream across network requires all network elements (NEs) to be aware of connection; All NEs have to be involved in re-establishment of connections in case of network fault
  - In connectionless network operation, NEs do not deal with each explicit connection and hence are much simpler in design
  - TCP connections over IP datagrams

Network Layer Functions

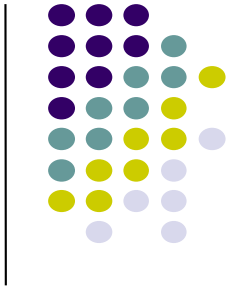


Essential

- **Routing**: mechanisms for determining the set of best paths for routing packets requires the collaboration of network elements
- **Forwarding**: transfer of packets from NE inputs to outputs
- **Priority & Scheduling**: determining order of packet transmission in each NE

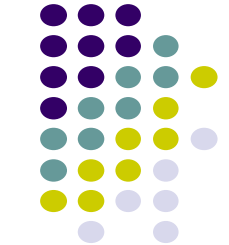
Optional: congestion control, segmentation & reassembly, security

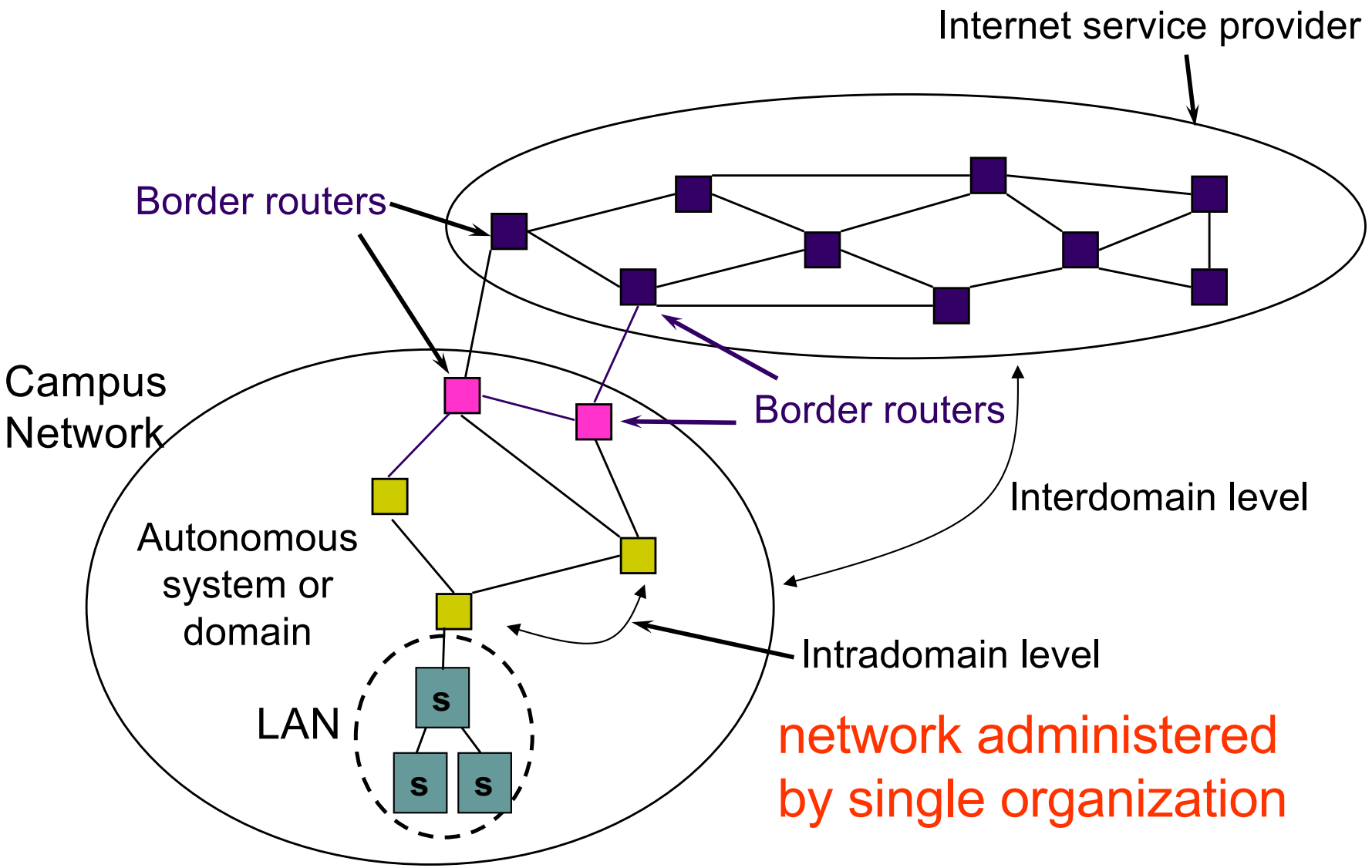
End-to-End Packet Network



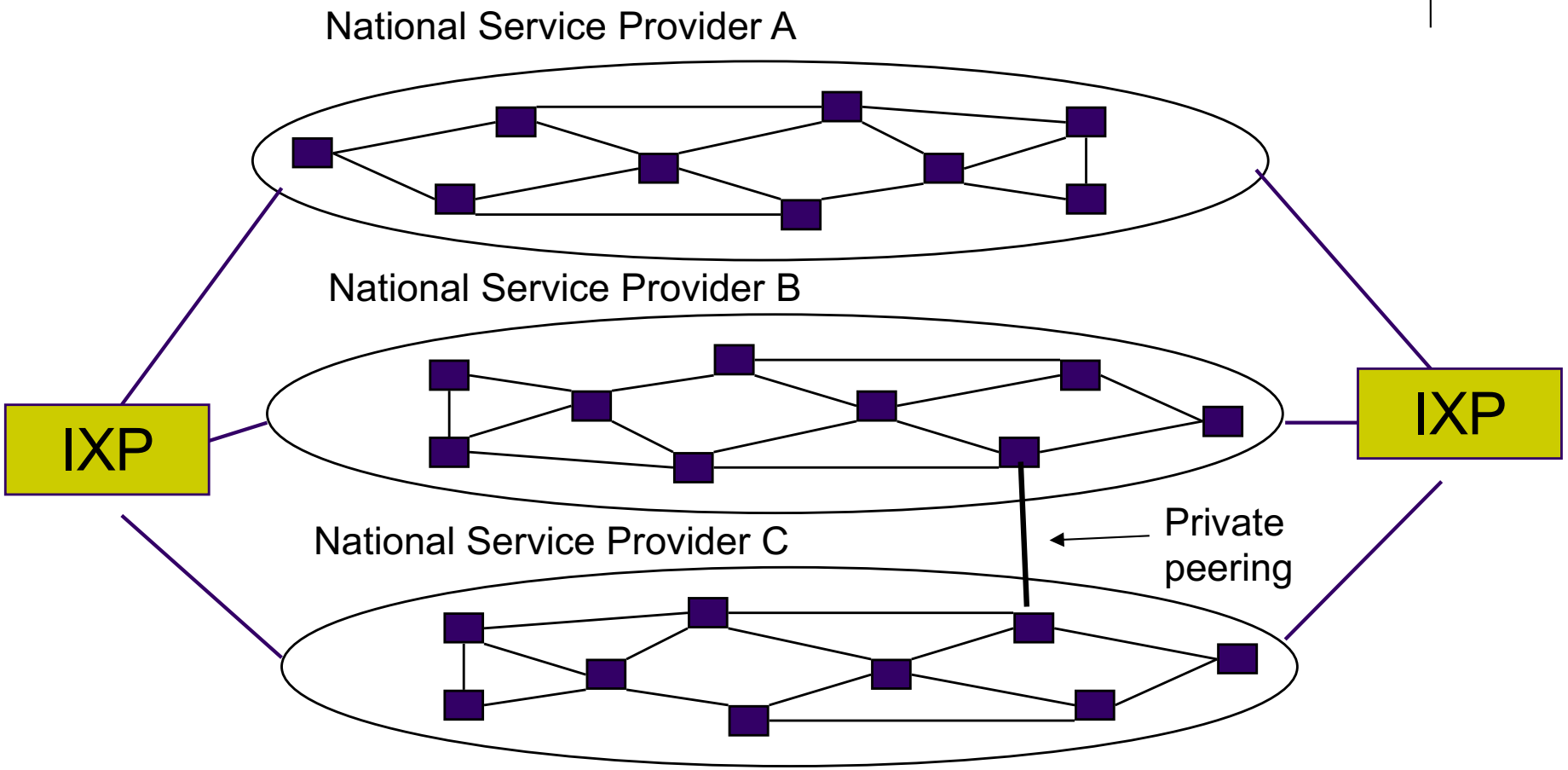
- Individual packet streams are highly bursty
  - Statistical multiplexing is used to concentrate streams
- User demand can undergo dramatic change
  - Examples: Bittorrent, YouTube, NetFlix, Catastrophic events
  - Routing tables must adapt to changing conditions
- Internet structure highly decentralized
  - Paths traversed by packets can go through many networks controlled by different organizations
  - No single entity responsible for end-to-end service

Connecting to Internet Service Provider



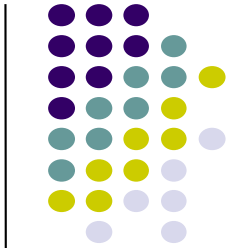


Internet Backbone



- Internet Exchange Points: Shared facilities where multiple AS's can exchange traffic
- Private Peering Points: two-party inter-ISP agreements to exchange traffic

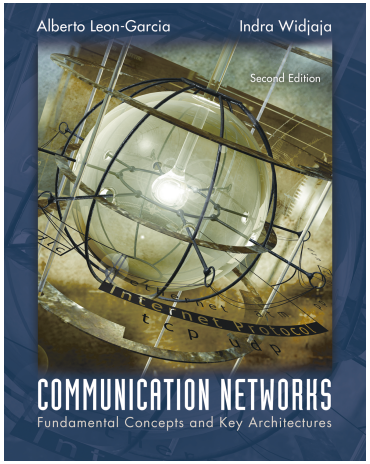
Key Role of Routing



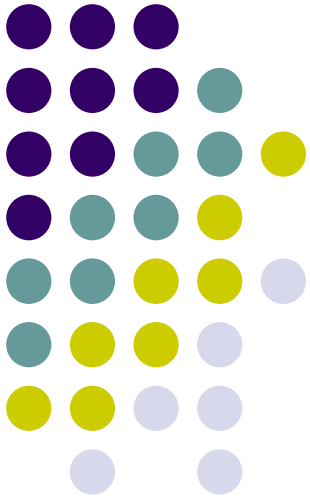
How to get packet from here to there?

- Decentralized nature of Internet makes routing a major challenge
  - Interior gateway protocols (IGPs) are used to determine routes within a domain
  - Exterior gateway protocols (EGPs) are used to determine routes across domains
  - Routes must be consistent & produce stable flows
- Scalability required to accommodate growth
  - Hierarchical structure of IP addresses essential to keeping size of routing tables manageable

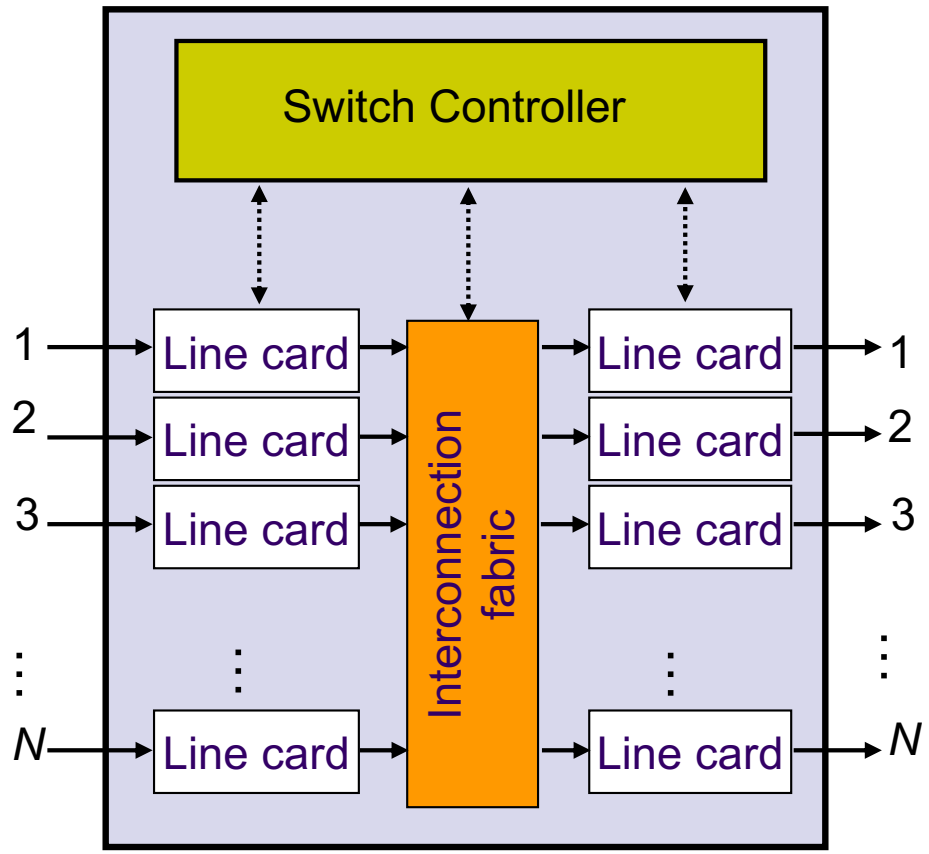
Structure of Packet Switches



Structure of a Packet Switch
Packet Scheduling
Policing and Shaping
Software-Defined Networks



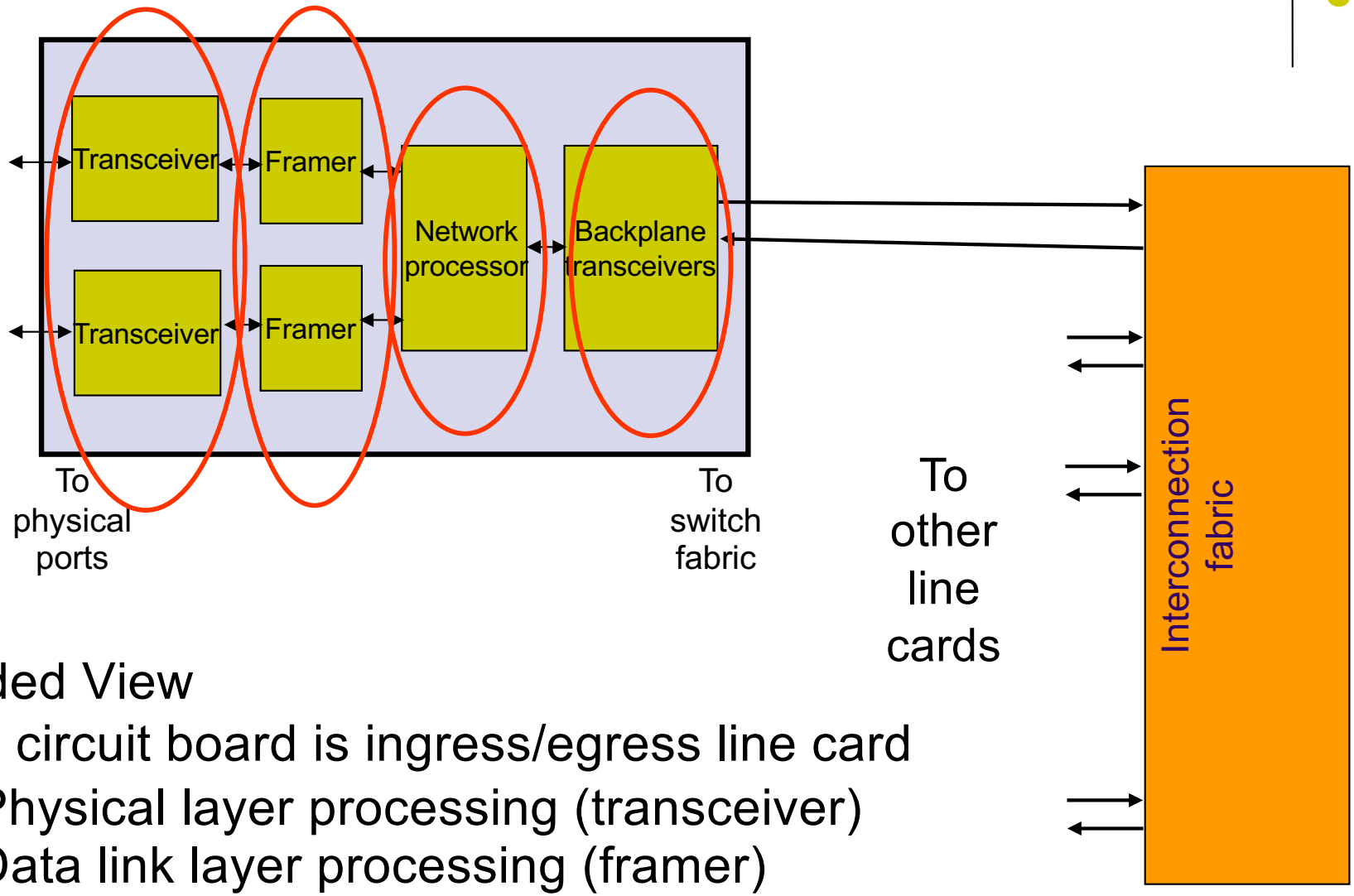
Generic Packet Switch



"Unfolded" View of Switch

- *Switch* transfers frames
- Router transfers IP packets
- Ingress Line Cards
  - Header processing
  - Ethernet switch: Read MAC address to forward
    - MAC or VLAN Tables
  - IP Router: Read Destination IP address to forward
    - IP Routing Tables
- Switch Controller
  - Signalling for connection setup or table setup
  - Resource allocation
- Interconnection Fabric
  - Forward frames/packets between line cards
- Egress Line Cards
  - Multiplexing
  - Scheduling & priority

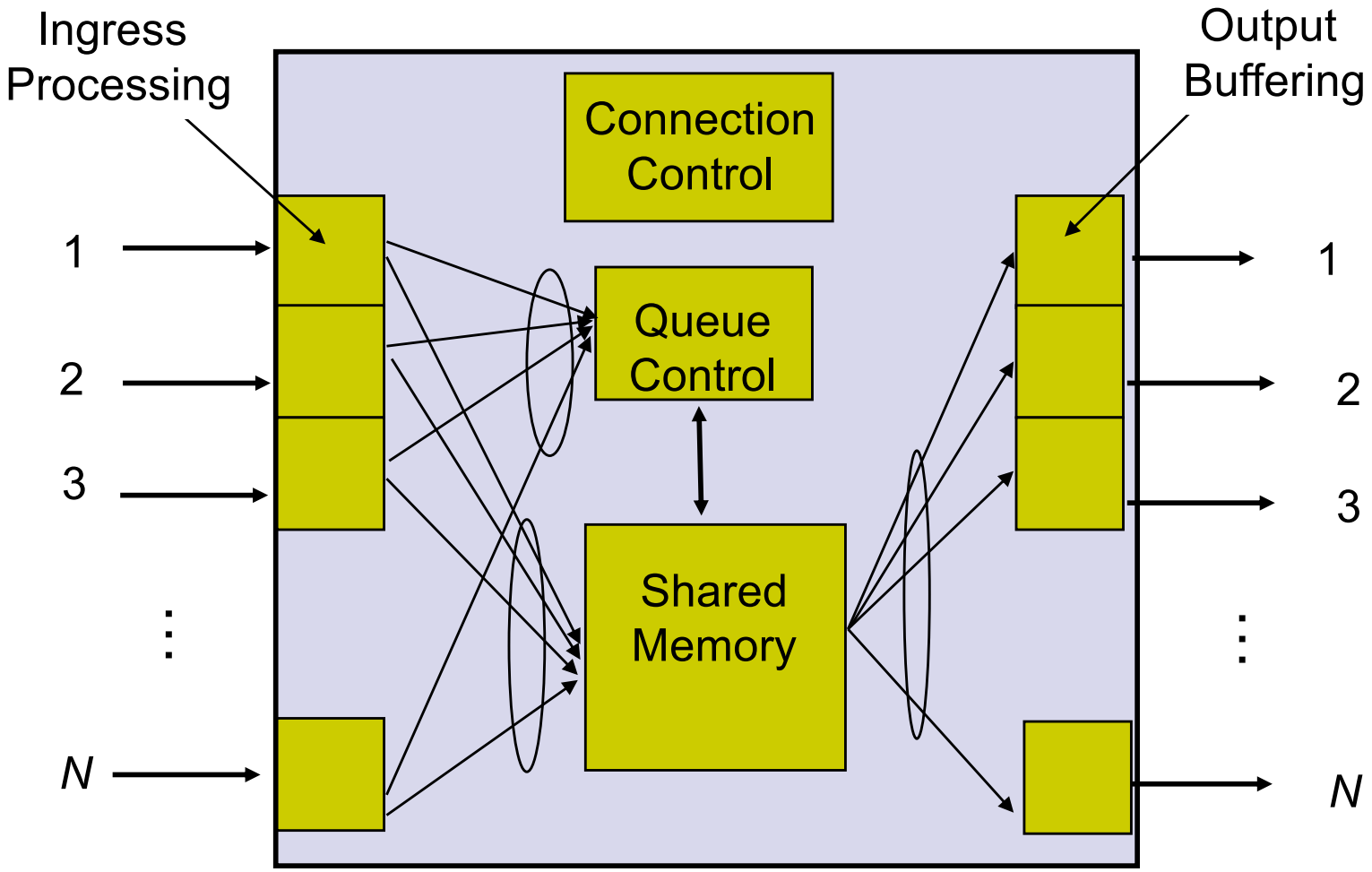
Line Cards



Folded View

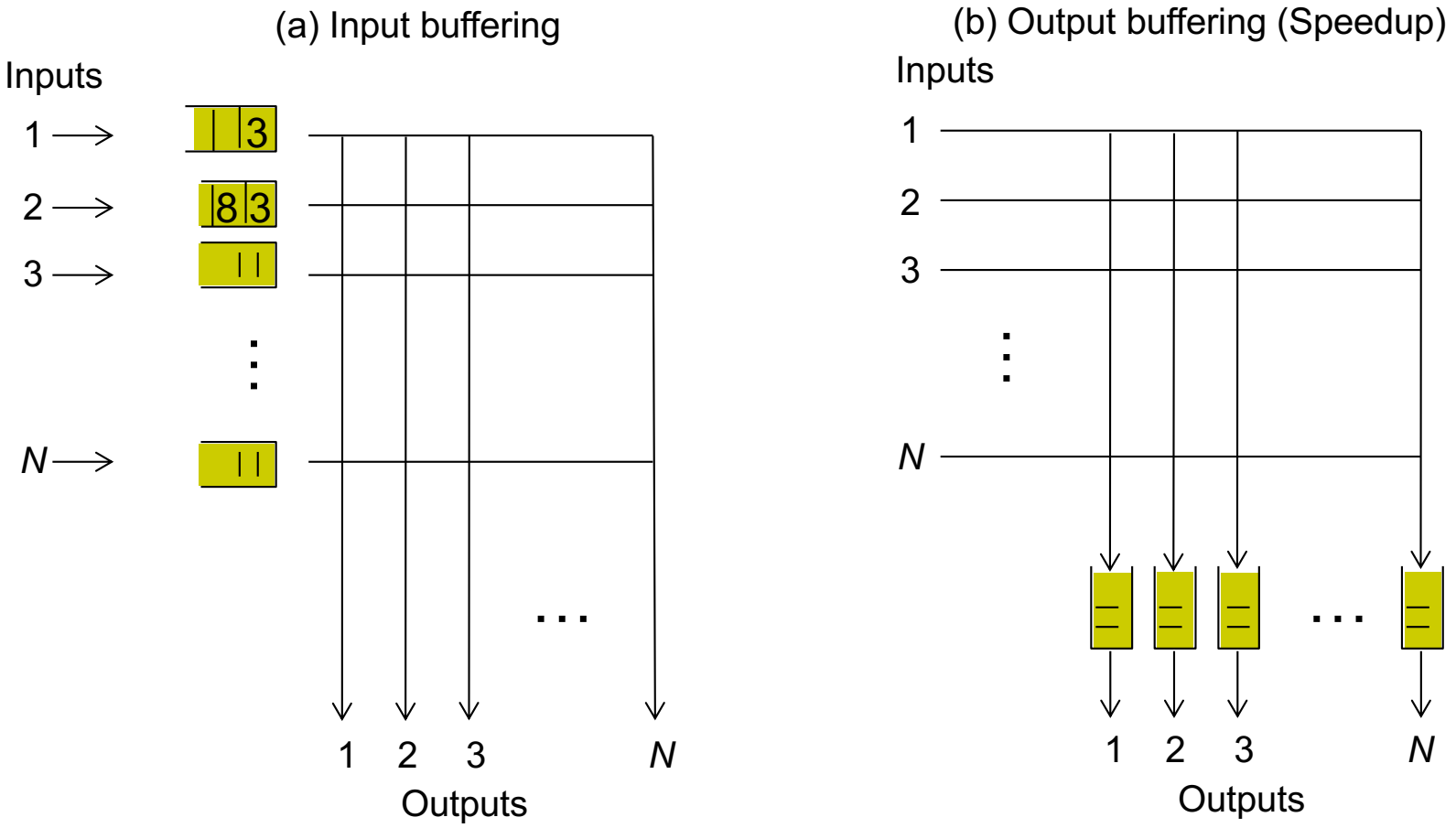
- 1 circuit board is ingress/egress line card
- Physical layer processing (transceiver)
- Data link layer processing (framer)
- Network header processing (net processor)
- Physical layer across fabric + framing

Shared Memory Packet Switch



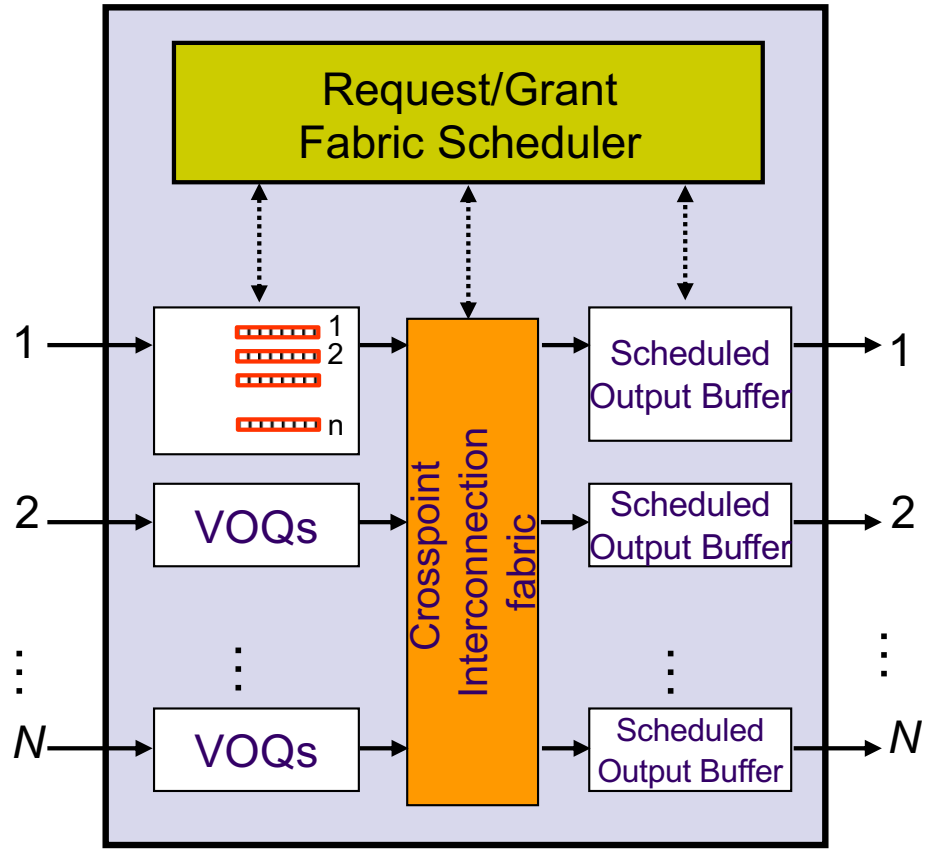
Small switches can be built by reading/writing into shared memory

Crossbar Switches



- Large switches built from crossbar & multistage space switches
- Needs centralized controller/scheduler (who sends to whom when)
- Can buffer at input, output, or both (performance vs complexity)

Typical Large Packet Switch

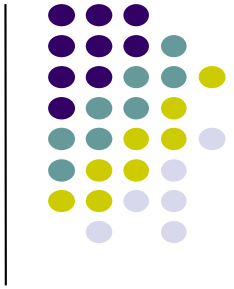


- Ingress Line Cards
  - Polices ingress traffic
  - Virtual Output Queues
  - 1 Queue per output port
  - Send requests to Scheduler
- Fabric Scheduler
  - Receive requests from LCs
  - Algorithm to provide fair transfer to ingress line cards & maximize throughput
  - Issues grants to LCs
- Interconnection Fabric
  - Transfer packets between LCs
- Egress Line Cards
  - Support priority classes
  - Provide fair share of bandwidth to packet flows & classes

End-to-End Quality of Service

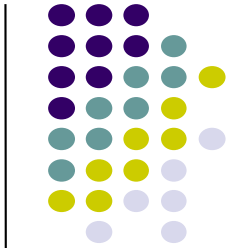
- A packet traversing network encounters delay and possible loss at various multiplexing points
- End-to-end performance is accumulation of per-hop performances

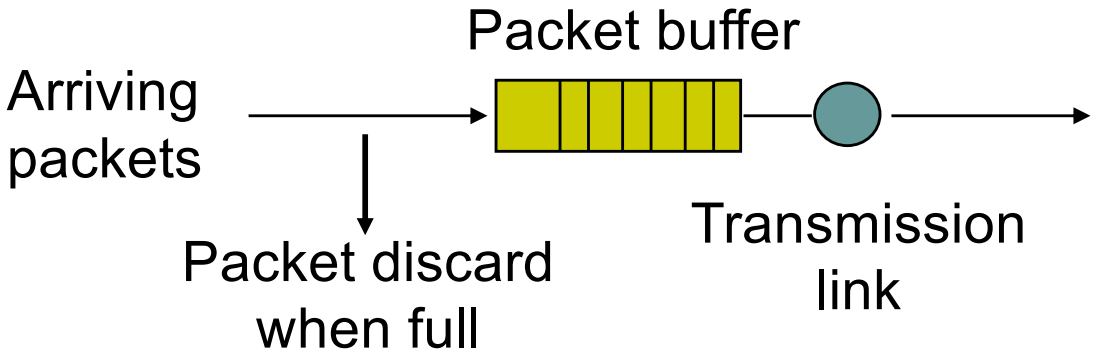
Admission, Scheduling & QoS



- Packet Scheduling & Dropping
  - Controls delay & loss in buffers
  - Controls bandwidth available to a packet flow/class
  - FIFO, Priority and Fair Queueing Scheduling
- Admission control
  - Decides whether a packet flow/class is allowed into network
  - Regulates traffic load

FIFO Queueing



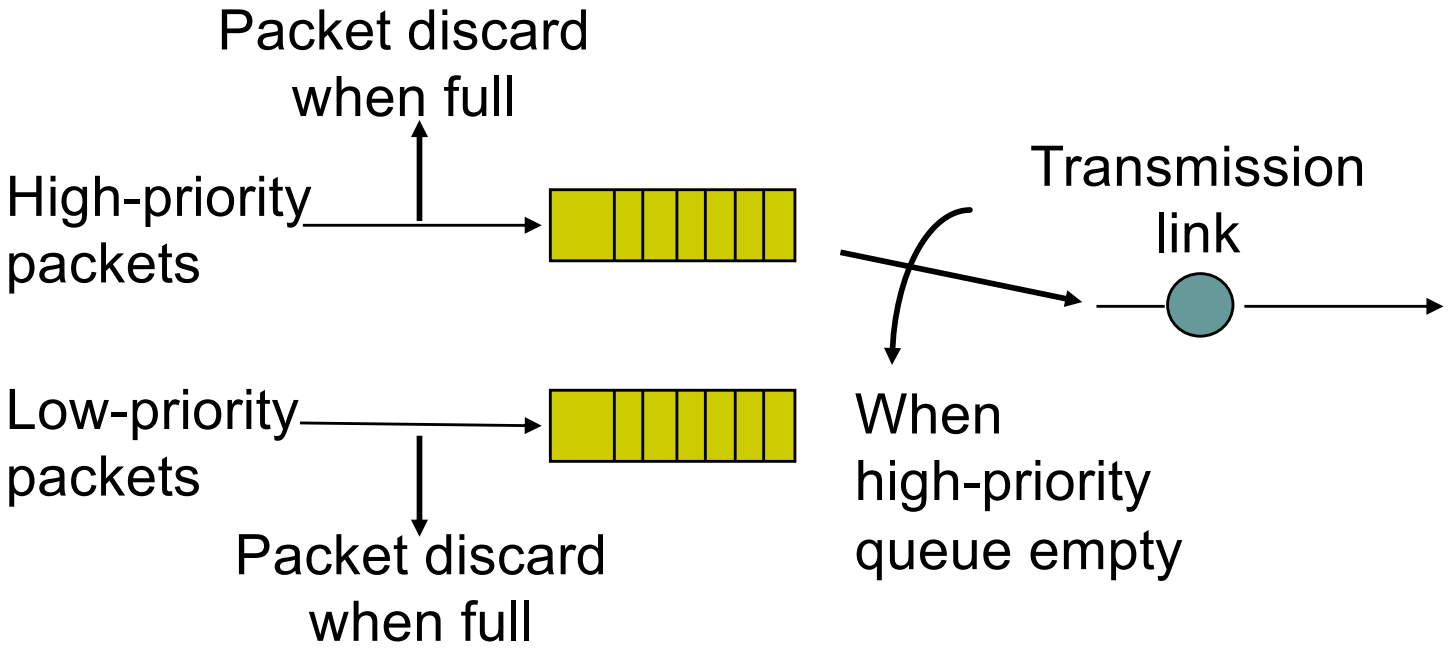


- All packet flows share the same buffer
- Transmission Discipline: First-In, First-Out
- Buffering Discipline: Discard arriving packets if buffer is full (Alternative: random discard; pushout head-of-line, i.e. oldest, packet)

FIFO Queueing

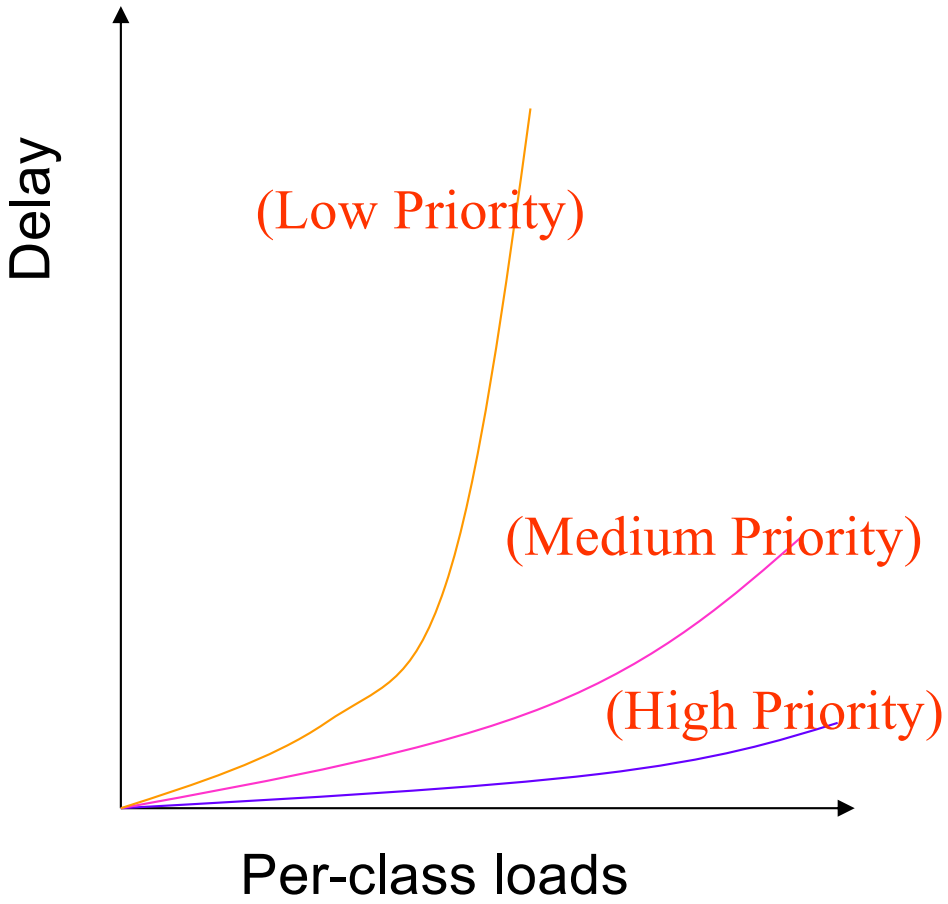
- Cannot provide differential QoS to different packet flows
  - Different packet flows interact strongly
- Statistical delay guarantees via load control
  - Restrict number of flows allowed (connection admission control)
  - Difficult to determine performance delivered
- Finite buffer determines a maximum possible delay
- Buffer size determines loss probability
  - But depends on arrival & packet length statistics

HOL Priority Queueing



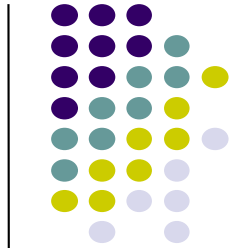
- High priority queue serviced until empty
- High priority queue has lower waiting time
- Buffers can be dimensioned for different loss probabilities
- Surge in high priority queue can cause low priority queue to saturate

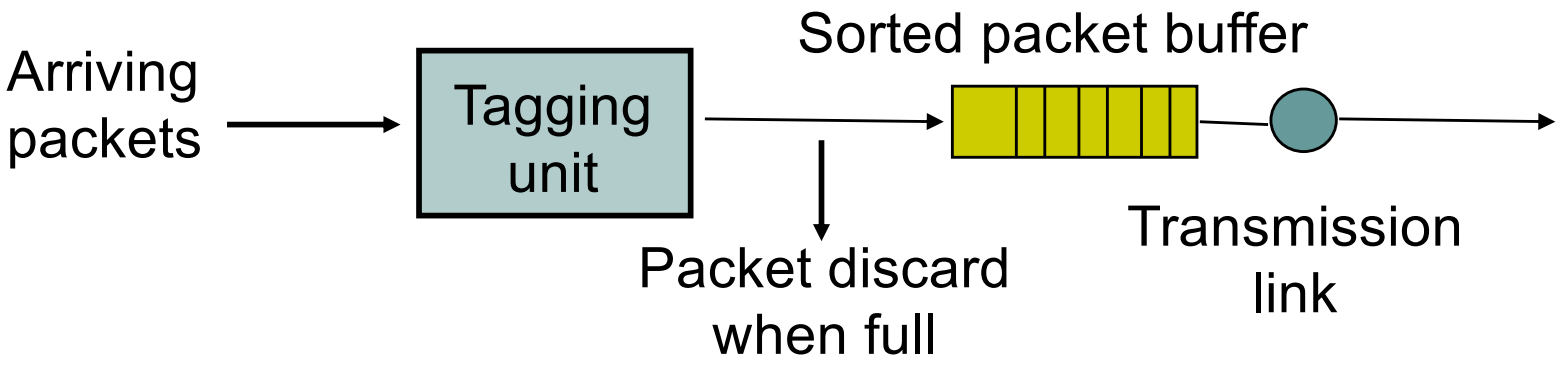
HOL Priority Features



- Provides differential QoS
- Pre-emptive priority: lower classes invisible
- Non-preemptive priority: lower classes impact higher classes through residual service times
- High-priority classes can hog all bandwidth & starve lower priority classes
- Need to provide some isolation between classes

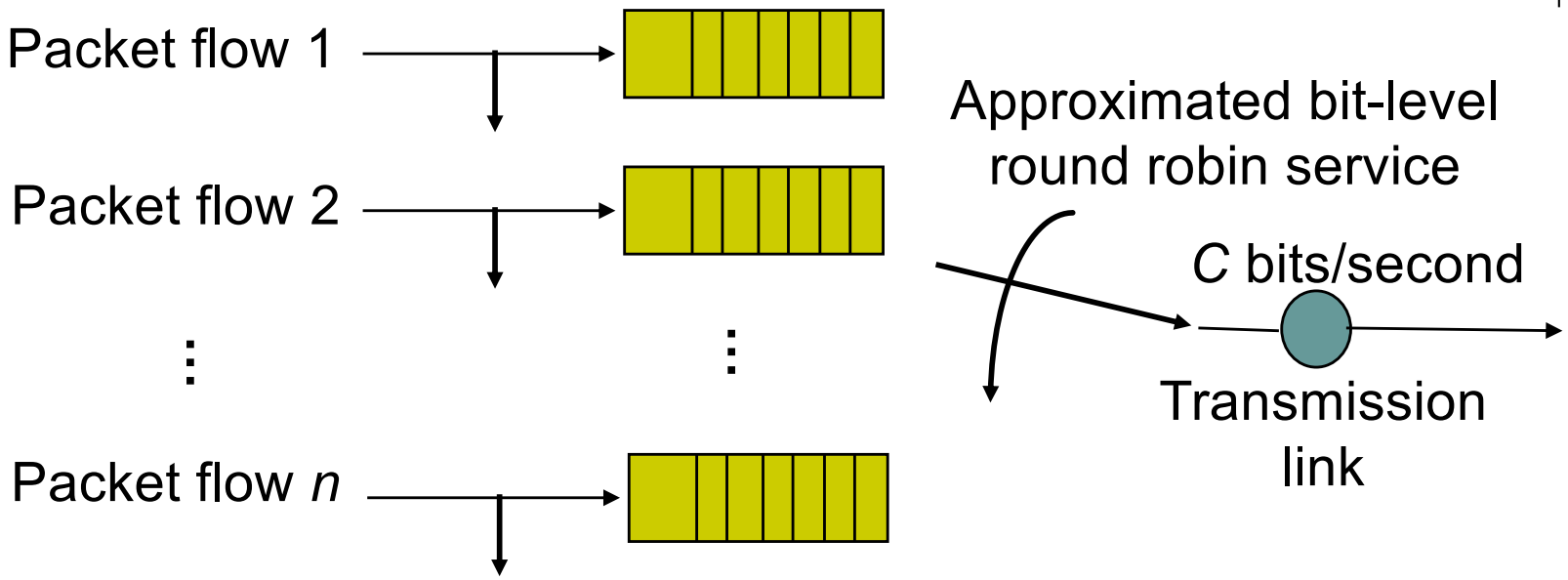
Earliest Due Date Scheduling





- Queue in order of “due date”
  - packets requiring low delay get earlier due date
  - packets without delay get indefinite or very long due dates
  - Requires sorting

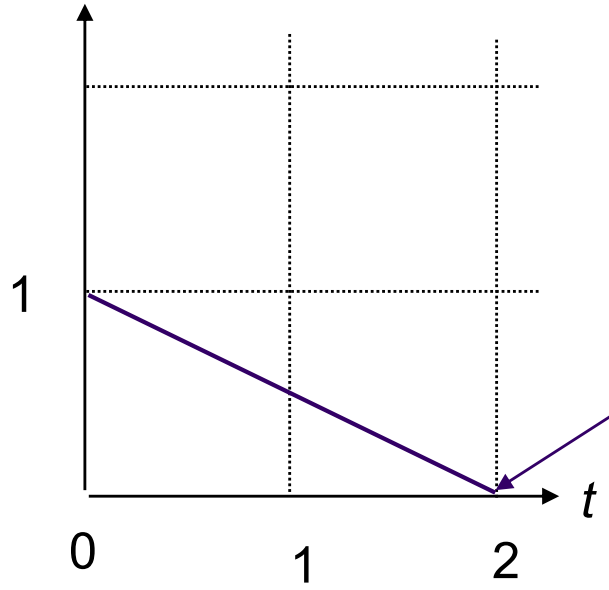
Fair Queueing / Generalized Processor Sharing



- Each flow has its own logical queue: prevents hogging; allows differential loss probabilities
- C bits/sec allocated equally among non-empty queues
  - transmission rate = C / n(t), where n(t)=# non-empty queues
- Idealized system assumes fluid flow from queues
- Implementation requires approximation: simulate fluid system; sort packets according to completion time in ideal system

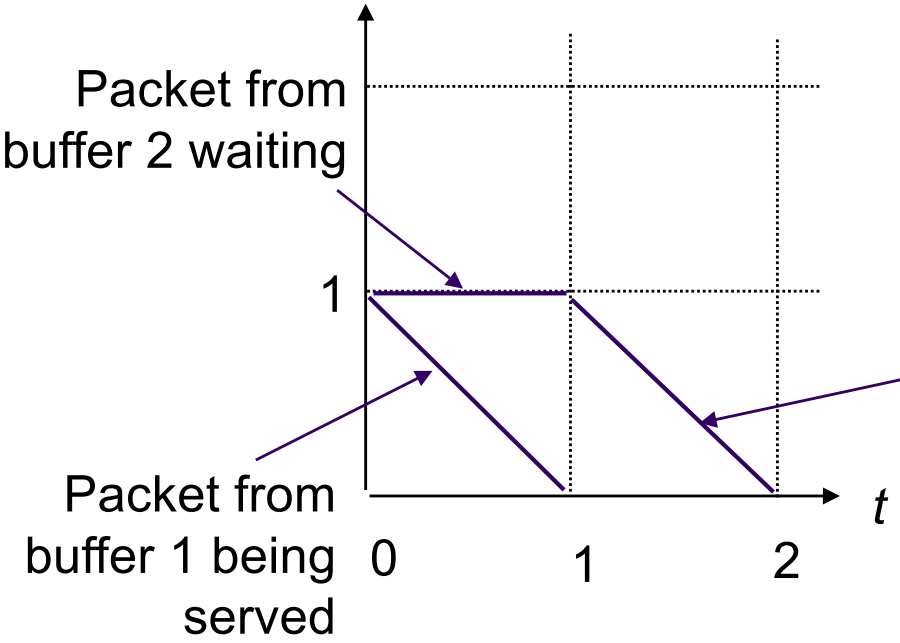
Buffer 1
at t=0

Buffer 2
at t=0



Fluid-flow system:
both packets served
at rate 1/2

Both packets
complete service
at t = 2

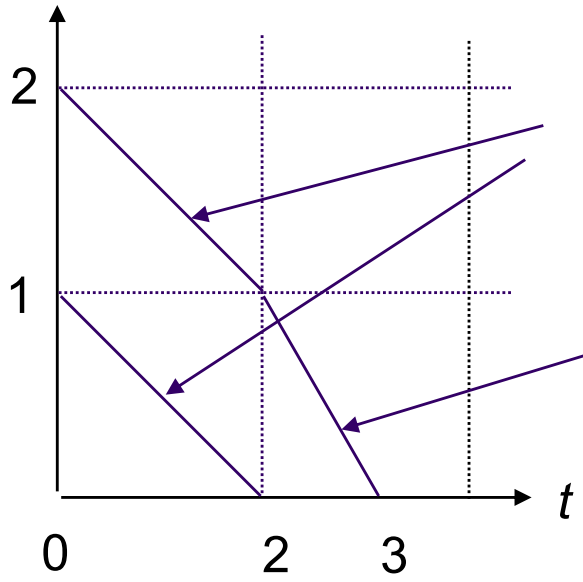


Packet-by-packet system:
buffer 1 served first at rate 1;
then buffer 2 served at rate 1.

Packet from buffer 2
being served

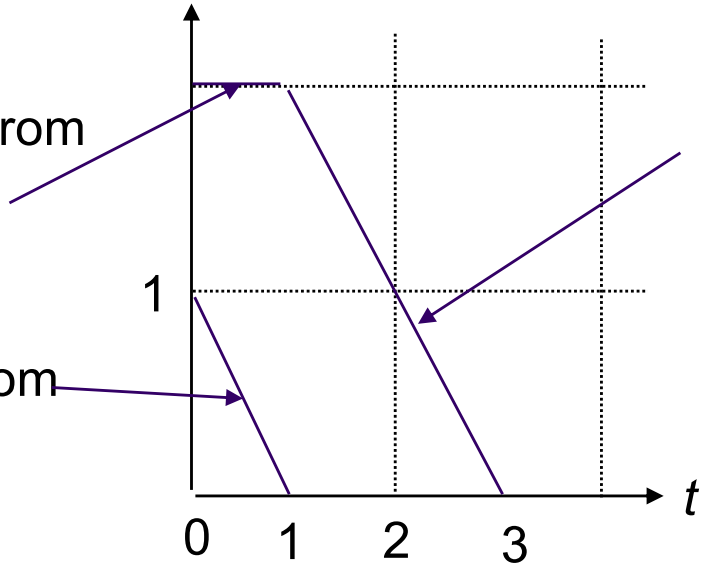
Buffer 1
at t=0

Buffer 2
at t=0



Fluid-flow system:
both packets served
at rate 1/2

Packet from buffer 2
served at rate 1



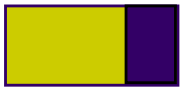
Packet from
buffer 2
waiting

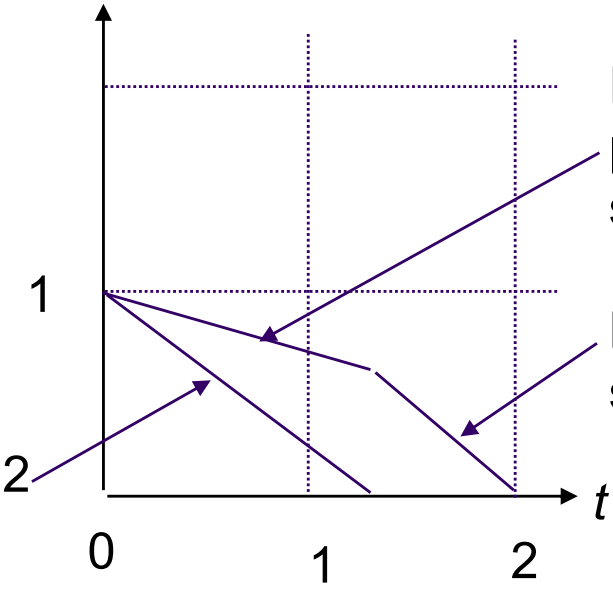
Packet from
buffer 1
served at
rate 1

Packet-by-packet
fair queueing:
buffer 2 served at rate 1

Buffer 1
at t=0
Weight 1

Buffer 2
at t=0
Weight 3

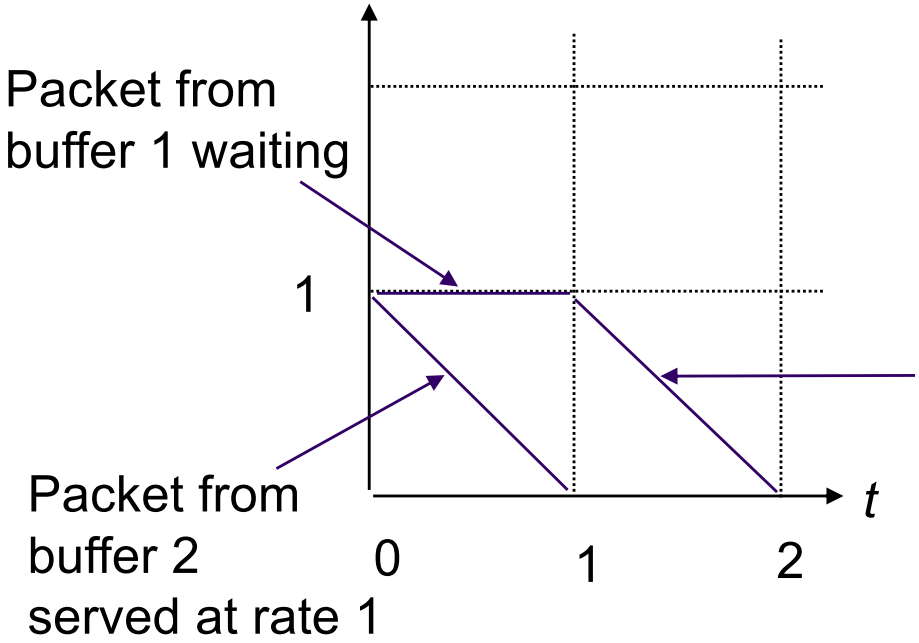




Fluid-flow system:
packet from buffer 1
served at rate 1/4;

Packet from buffer 1
served at rate 1

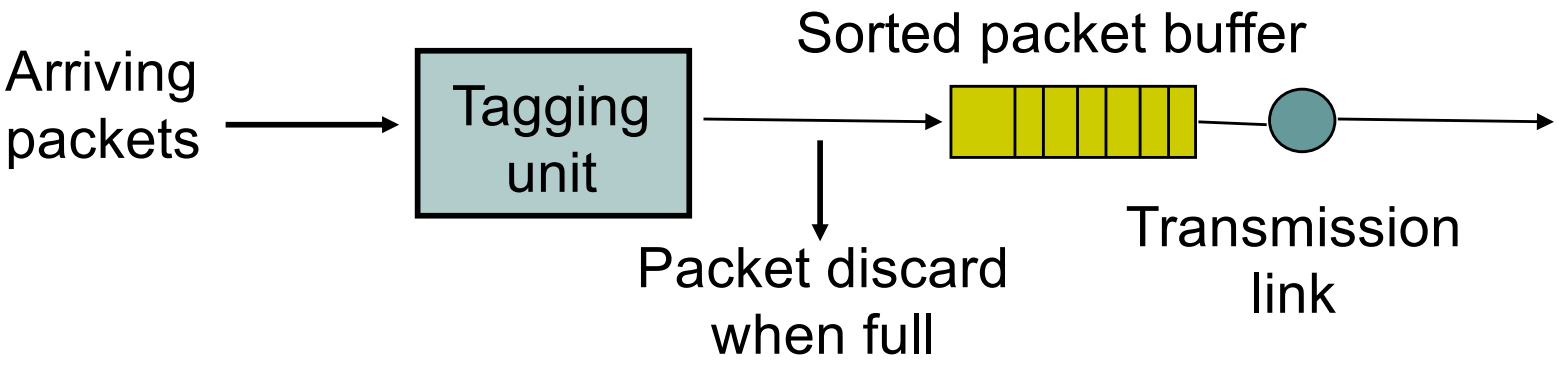
Packet from buffer 2
served at rate 3/4



Packet-by-packet weighted fair queueing:
buffer 2 served first at rate 1;
then buffer 1 served at rate 1

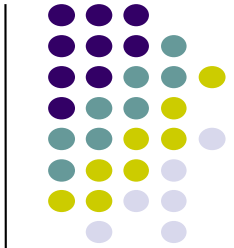
Packet from buffer 1 served at rate 1

Packetized GPS/WFQ

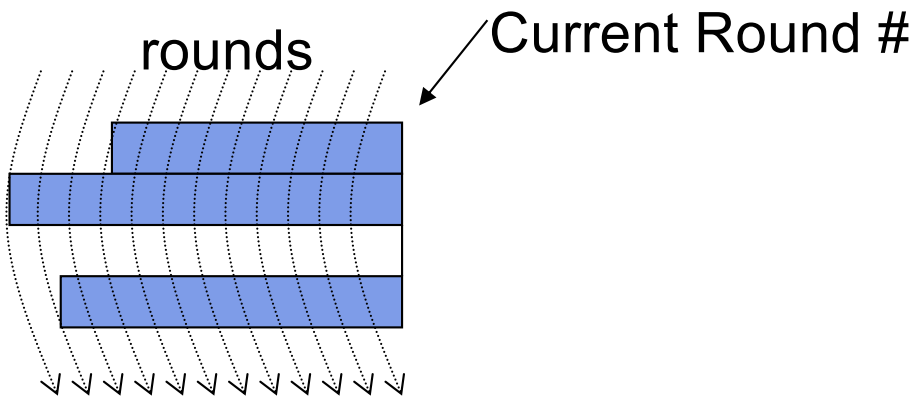


- Compute packet completion time in ideal system
  - add tag with completion time to packet
  - sort packet in queue according to tag
  - serve according to HOL
  - completion time is ONLY for scheduling, it is NOT actual completion time

Bit-by-Bit Fair Queueing



- Assume n flows, n queues
- 1 round = 1 cycle serving all n queues
- If each queue gets 1 bit per cycle, then 1 round = # active queues
- Round number = number of cycles of service that have been completed

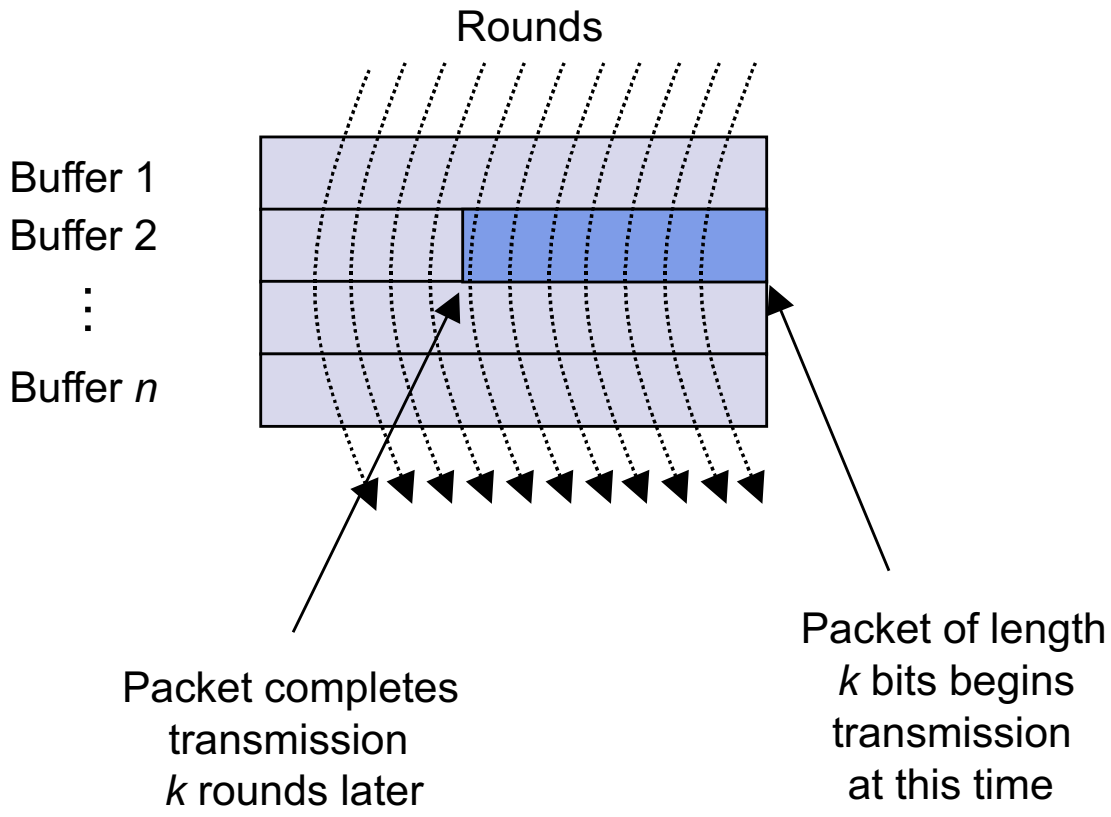


- If packet arrives to idle queue:

  Finishing time = round number + packet size in bits
- If packet arrives to active queue:

Finishing time = finishing time of last packet in queue + packet size

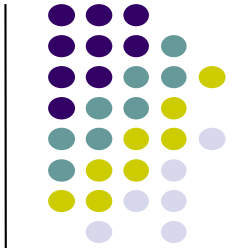
Number of rounds = Number of bit transmission opportunities



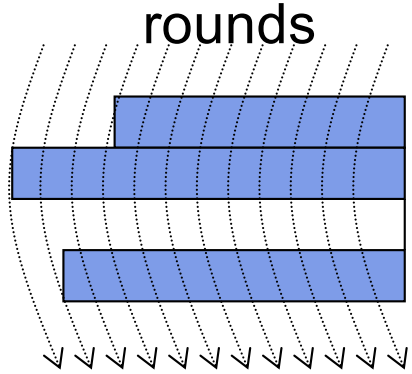
Differential Service:

If a traffic flow is to receive twice as much bandwidth as a regular flow, then its packet completion time would be half

Computing the Finishing Time



- *F(i,k,t)* = finish time of *k*th packet that arrives at time *t* to flow *i*
- *P(i,k,t)* = size of *k*th packet that arrives at time t to flow *i*
- *R(t)* = round number at time *t*
- *Update R(t) every packet arrival & departure*



Generalize so *R(t)* continuous, not discrete

R(t) grows at rate inversely proportional to *n(t)*

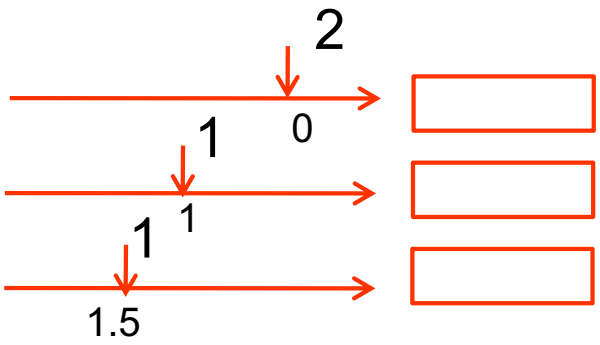
$$dR(t)/dt = C/n_{active}(t),$$

- Fair Queueing:

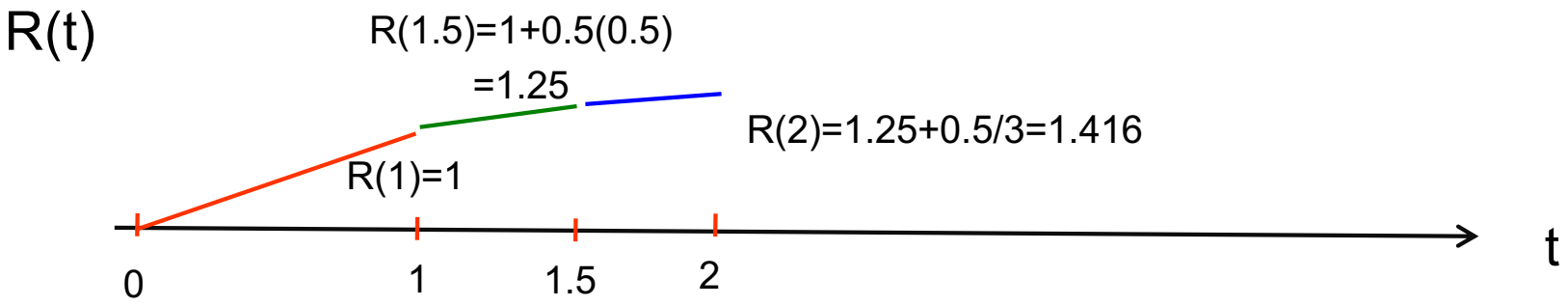
$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)$$

- Weighted Fair Queueing:

$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)/w_i$$



$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)$$



F(1,1,0) = R(0) + P(1,1,0)=2 pkt enters service, dep time =2, R' =1

Nothing happens until next arrival

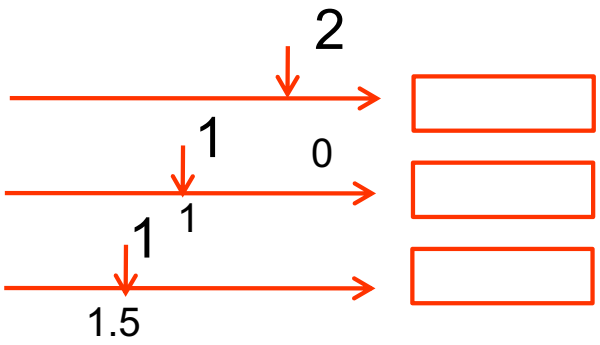
F(2,1,1) = R(1) + P(2,1,1)=1+1=2 pkt in queue, R' =1/2

Nothing happens until next arrival

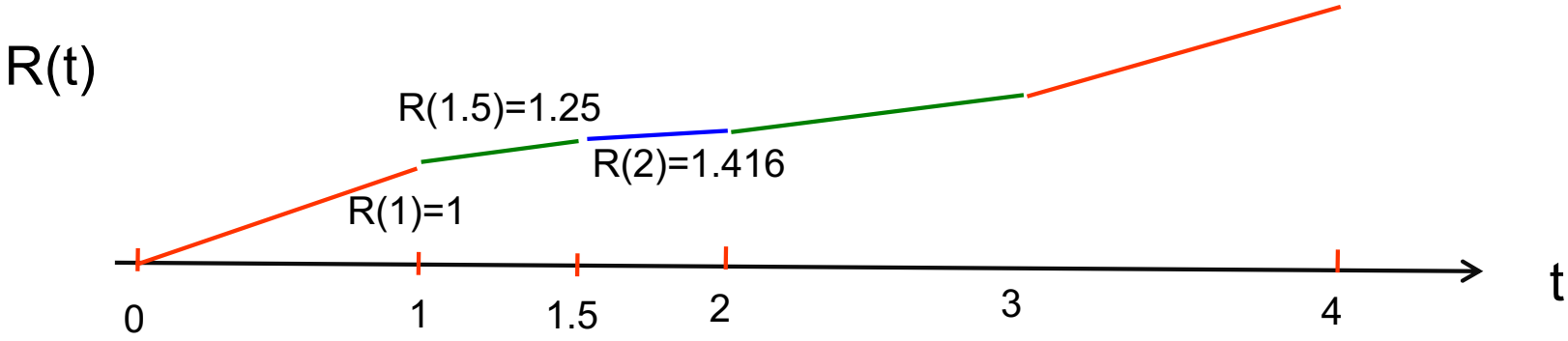
F(3,1,1.5) = R(1.5) + P(3,1,1.5)=1.25+1=2.25 pkt in queue, R' =1/3

Nothing happens until departure of packet from queue 1 @ t=2

R(2)=1.25+0.5/3=1.416



$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)$$



pkt finishes service at t=2, R' =1/2

Tag of pkt from Q2 = 2 < 2.25 = Tag of pkt from Q3

Pkt from Q2 enters service, Dep. Time =2+1=3

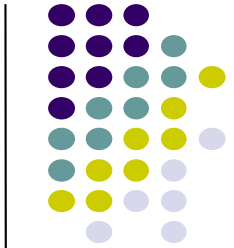
R(3)=R(2)+1(0.5)=1.916

Pkt from Q3 enters service, departs at t=3+1=4

R(4)=1.916

Leaves system empty, reset R(t) to 0 upon next arrival

Self-Clocked Fair Queuing



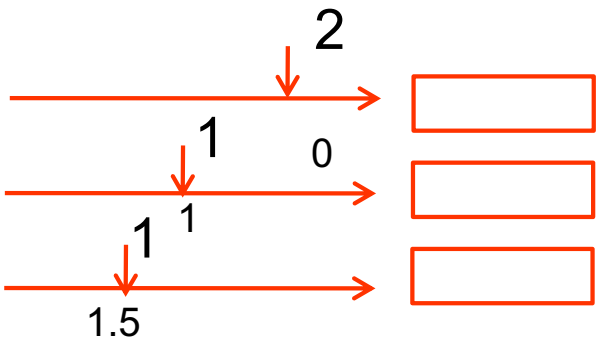
- *Computing R(t) at every arrival & departure is complex*
- *SCFQ replaces R(t) with tag of packet currently in service*

$$F(i,k,t) = \max\{F(i,k-1,t), CF\} + P(i,k,t)$$

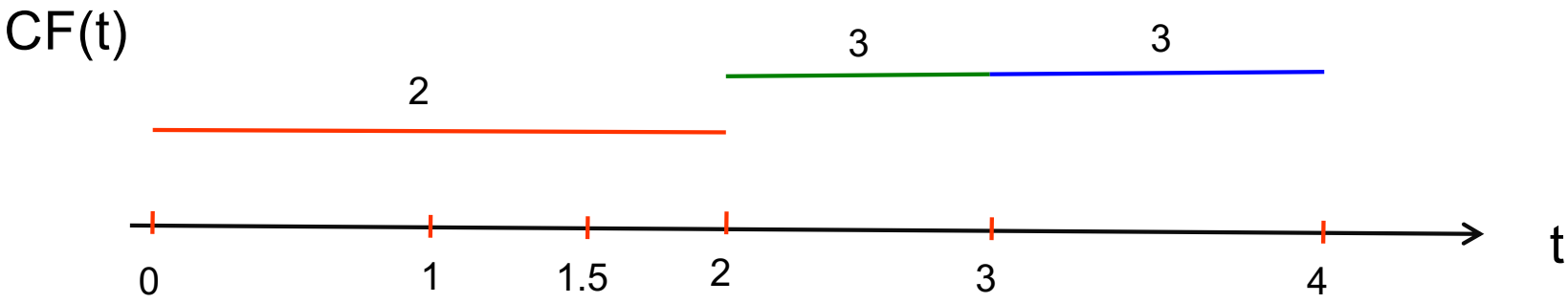
where

CF = finish time of packet currently in service

- *Simpler to implement*
- *Weaker fairness guarantees (deviation from ideal GPS)*



$$F(i,k,t) = \max\{F(i,k-1,t), CF\} + P(i,k,t)$$



F(1,1,0) = R(0) + P(1,1,0)=2 pkt enters service, CF=2,

Nothing happens until next arrival

F(2,1,1) = CF+ P(2,1,1)=2+1=3 pkt in queue

Nothing happens until next arrival

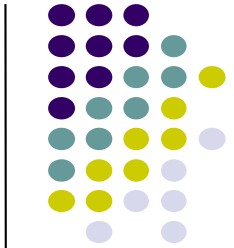
F(3,1,1.5) = CF+ P(3,1,1.5)=2+1=3 pkt in queue

Nothing happens until departure of packet from queue 1 @ t=2

WFQ and Packet QoS

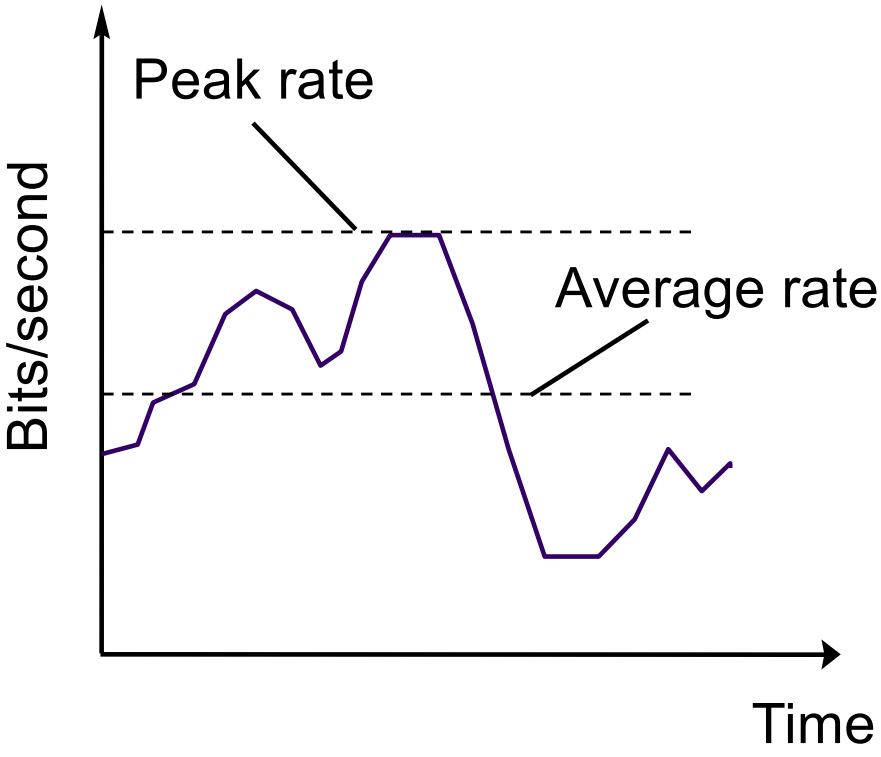
- WFQ and its many variations form the basis for providing QoS in packet networks
- Very high-speed implementations available, up to 10 Gbps and higher
- WFQ must be combined with other mechanisms to provide end-to-end QoS (next section)

Buffer Management



- Packet drop strategy: Which packet to drop when buffers full
- Fairness: protect behaving sources from misbehaving sources
- Aggregation:
  - Per-flow buffers protect flows from misbehaving flows
  - Full aggregation provides no protection
  - Aggregation into classes provided intermediate protection
- Drop priorities:
  - Drop packets from buffer according to priorities
  - Maximizes network utilization & application QoS
  - Examples: layered video, policing at network edge
- Controlling sources at the edge

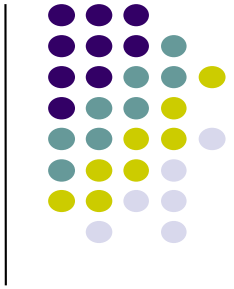
Admission Control



Typical bit rate demanded by a variable bit rate information source

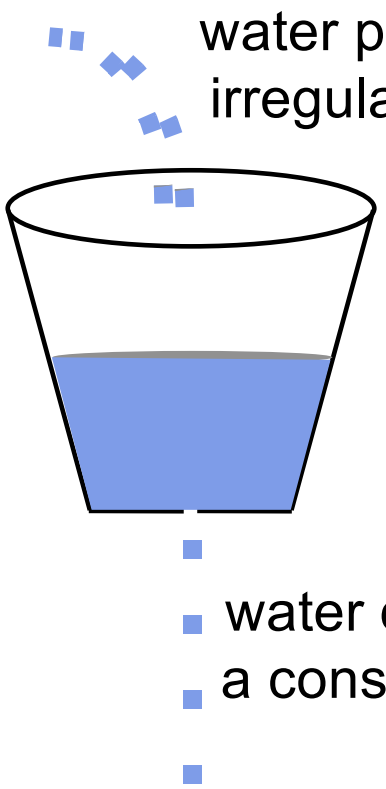
- Flows negotiate contract with network
- Specify requirements:
  - Peak, Avg., Min Bit rate
  - Maximum burst size
  - Delay, Loss requirement
- Network computes resources needed
  - "Effective" bandwidth
- If flow accepted, network allocates resources to ensure QoS delivered as long as source conforms to contract

Policing



- Network monitors traffic flows continuously to ensure they meet their traffic contract
- When a packet violates the contract, network can discard or tag the packet giving it lower priority
- If congestion occurs, tagged packets are discarded first
- *Leaky Bucket Algorithm* is the most commonly used policing mechanism
  - Bucket has specified leak rate for average contracted rate
  - Bucket has specified depth to accommodate variations in arrival rate
  - Arriving packet is *conforming* if it does not result in overflow

Leaky Bucket algorithm can be used to police arrival rate of a packet stream



Leak rate corresponds to long-term rate

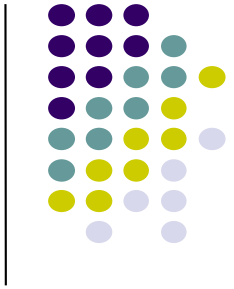
Bucket depth corresponds to maximum allowable burst arrival

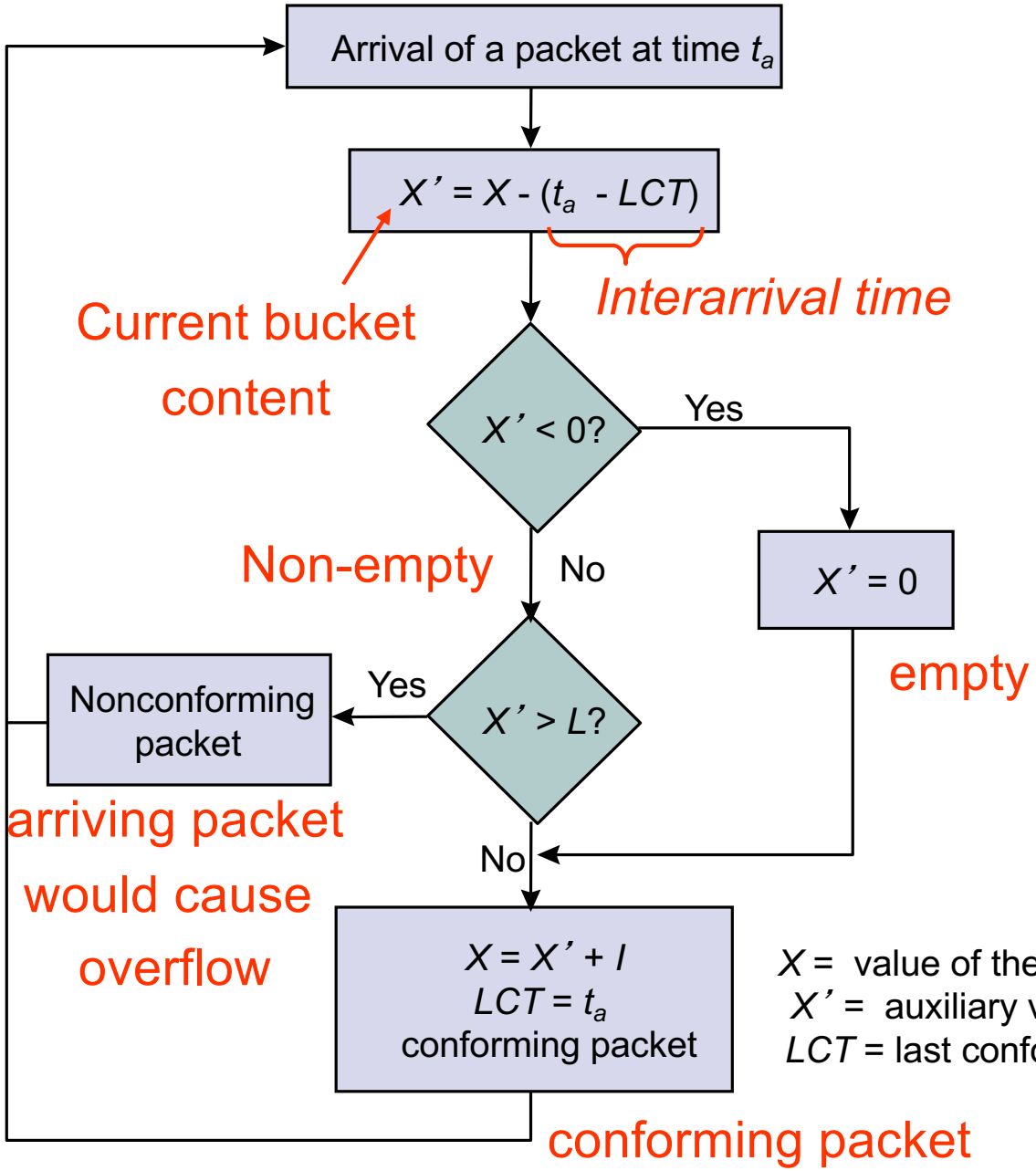
1 packet per unit time
Assume constant-length packet (as in ATM)

Let X = bucket content at last conforming packet arrival

Let t_a – last conforming packet arrival time = depletion in bucket

Leaky Bucket Algorithm





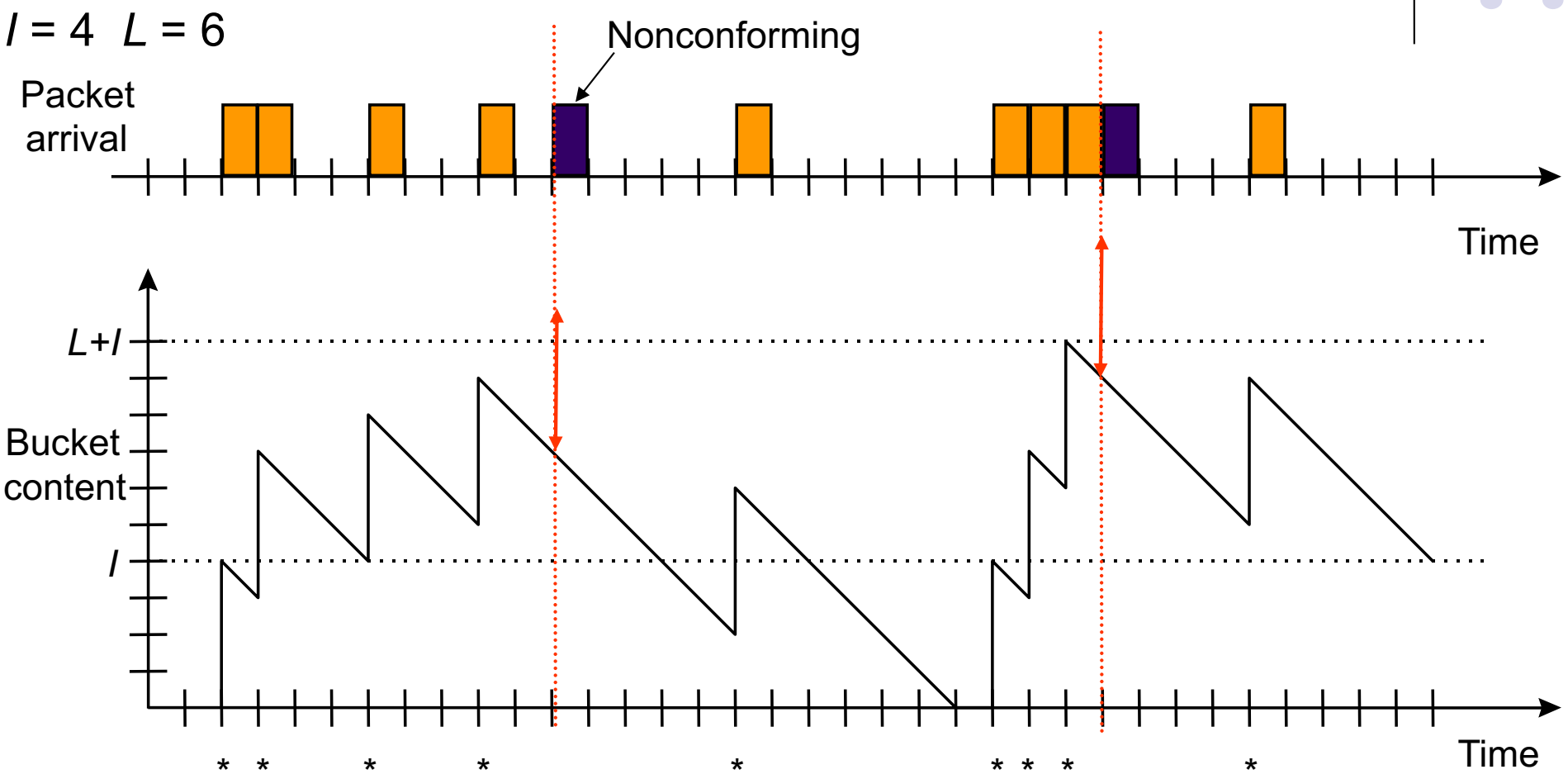
Depletion rate:
1 packet per unit time

$L+I$ = Bucket Depth

I = increment per arrival, nominal interarrival time

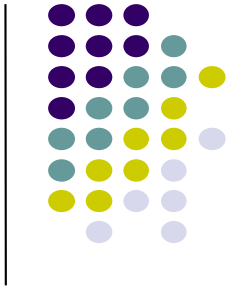
X = value of the leaky bucket counter
X' = auxiliary variable
LCT = last conformance time

Leaky Bucket Example



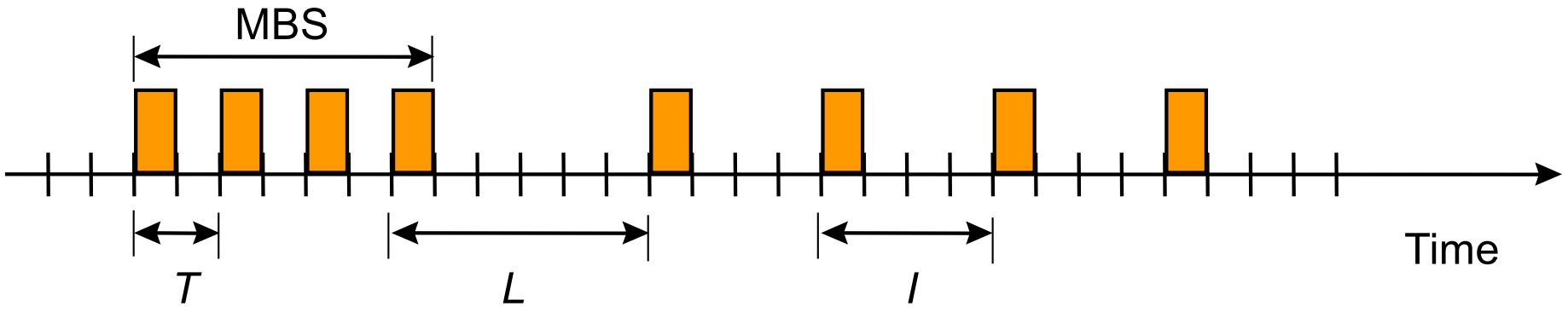
Non-conforming packets not allowed into bucket & hence not included in calculations

Policing Parameters

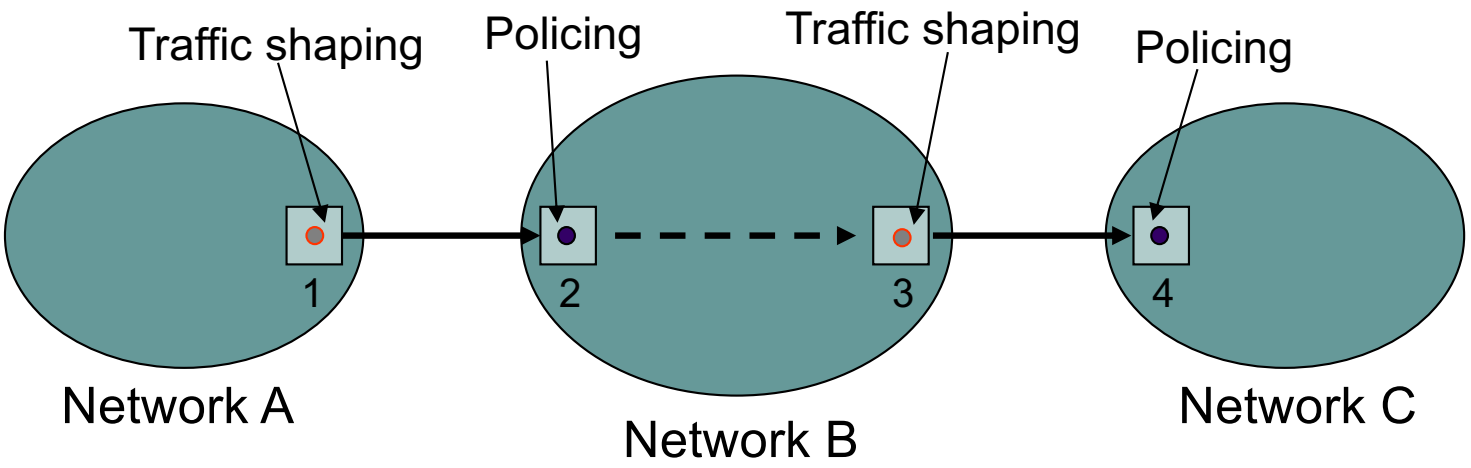


T = 1 / peak rate
MBS = maximum burst size
I = nominal interarrival time = 1 / sustainable rate

$$MBS = 1 + \left[\frac{L}{I - T} \right]$$

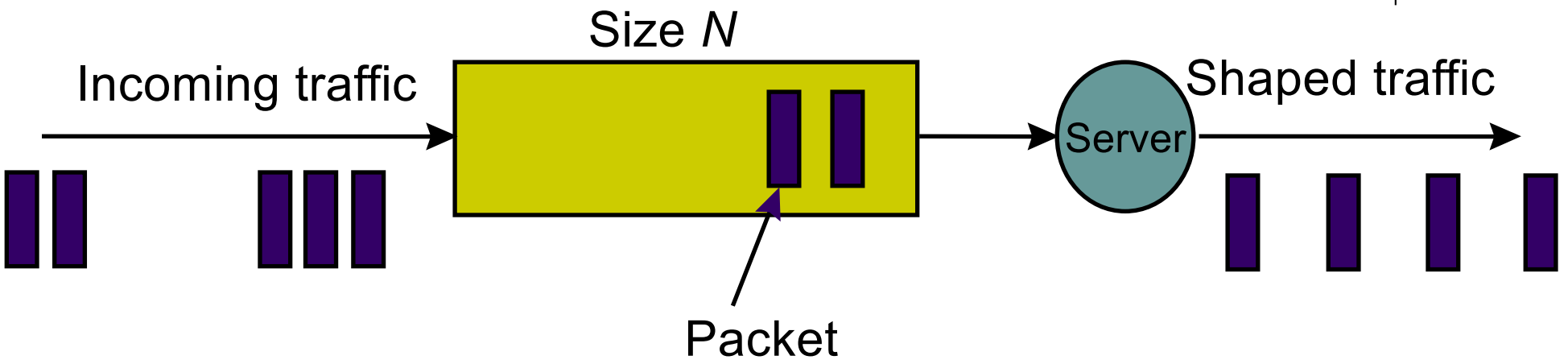


Traffic Shaping



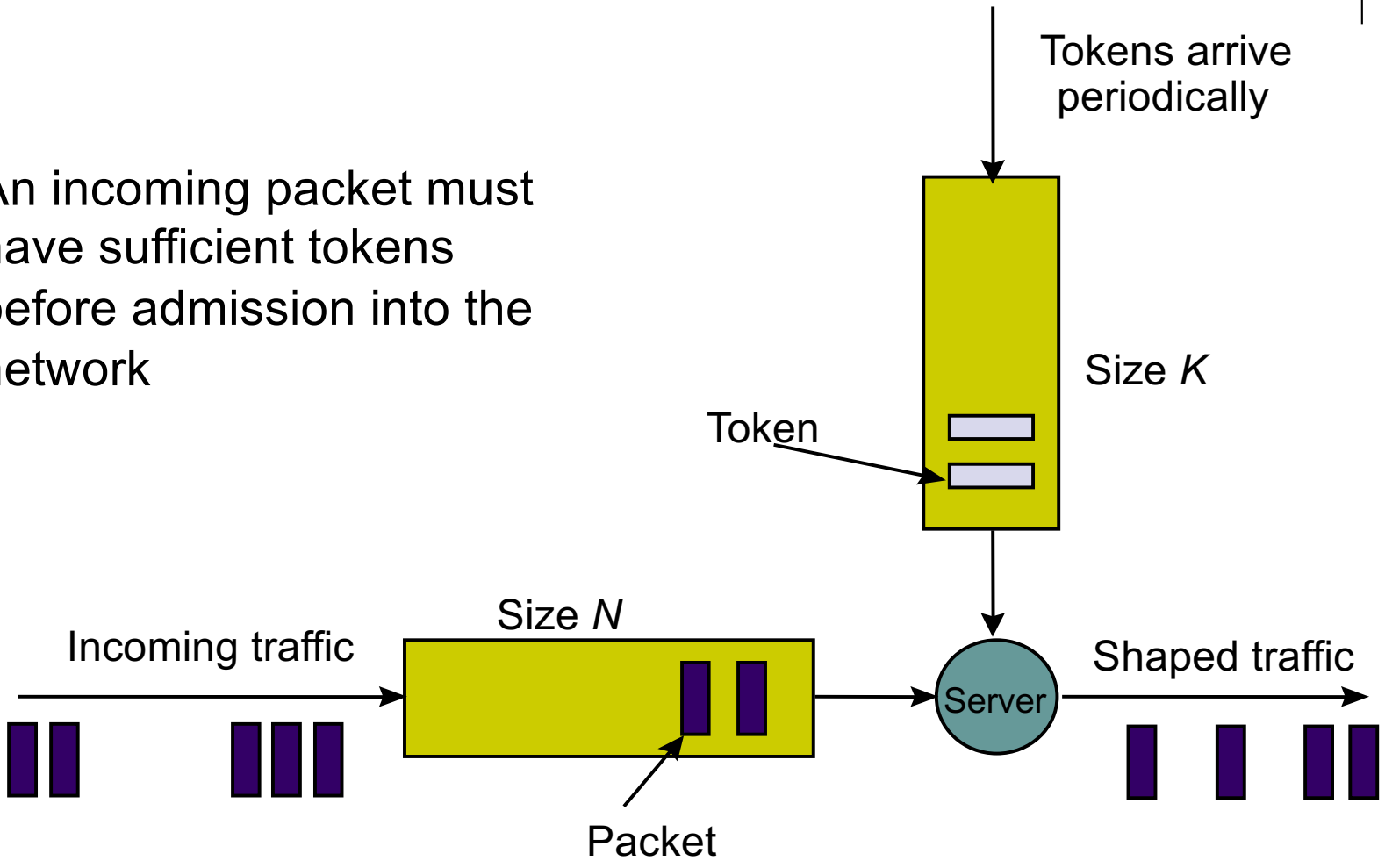
- Networks police the incoming traffic flow
- *Traffic shaping* is used to ensure that a packet stream conforms to specific parameters
- Networks can shape their traffic prior to passing it to another network

Leaky Bucket Traffic Shaper



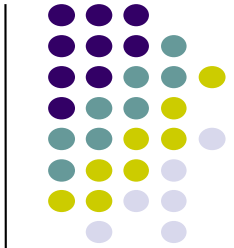
- Buffer incoming packets
- Play out periodically to conform to parameters
- Surges in arrivals are buffered & smoothed out
- Possible packet loss due to buffer overflow
- Too restrictive, since conforming traffic does not need to be completely smooth

Token Bucket Traffic Shaper

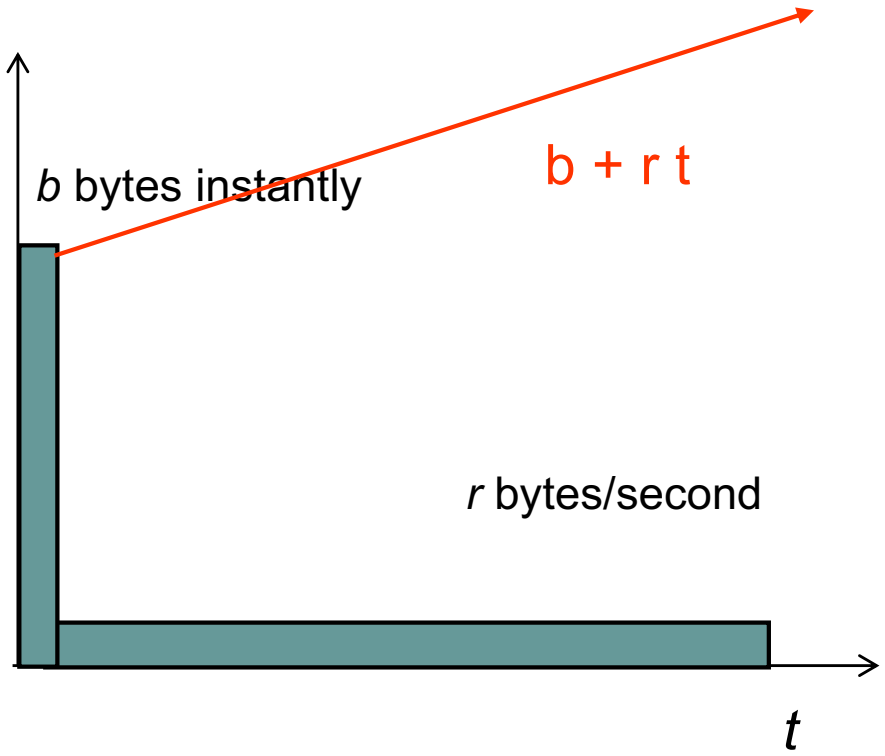


- Token rate regulates transfer of packets
- If sufficient tokens available, packets enter network without delay
- K determines how much burstiness allowed into the network

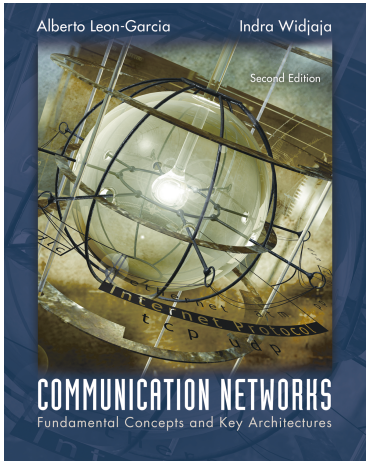
Token Bucket Shaping Effect

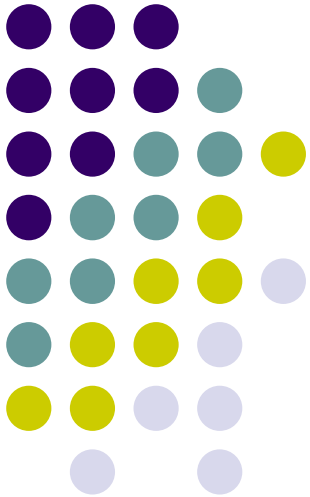


The token bucket constrains the traffic from a source to be limited to $b + r\,t$ bits in an interval of length t



Chapter 7
Packet-Switching Networks



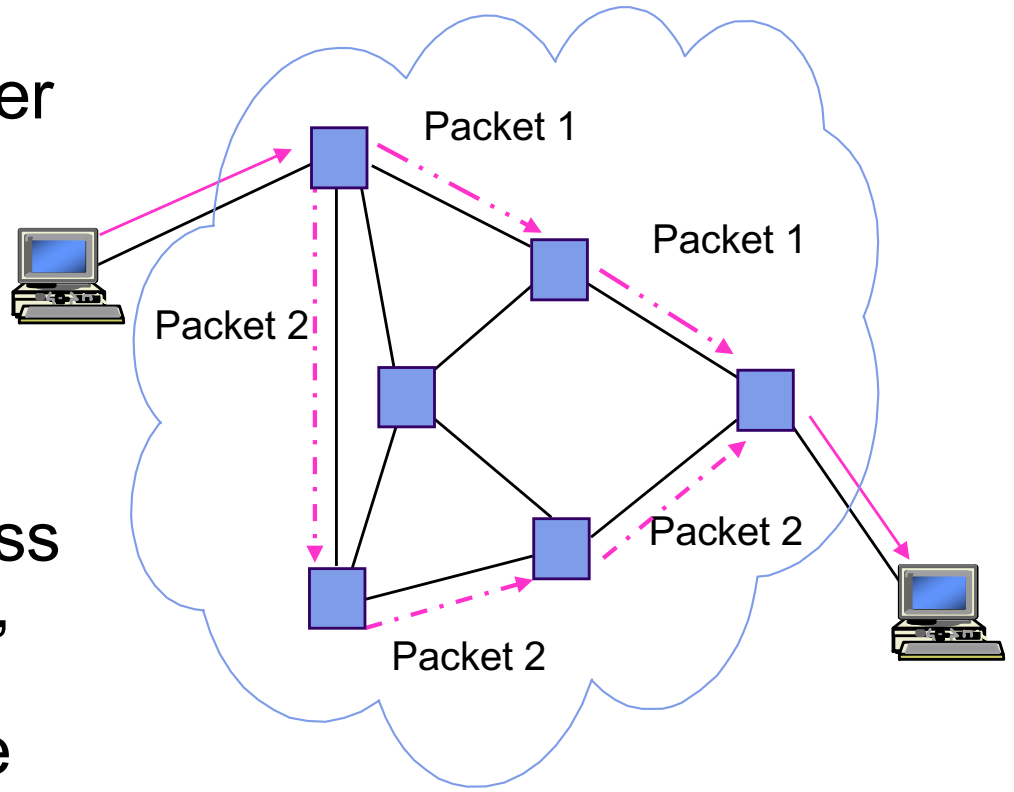


Datagrams and Virtual Circuits

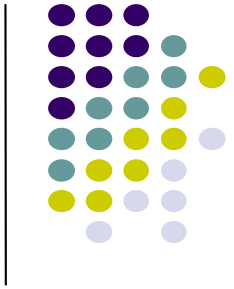
pp 471-483

Packet Switching - Datagram

- Messages broken into smaller units (packets)
- Source & destination addresses in packet header
- Connectionless, packets routed independently (datagram)
- Packet may arrive out of order
- Pipelining of packets across network can reduce delay, increase throughput
- Lower delay that message switching, suitable for interactive traffic



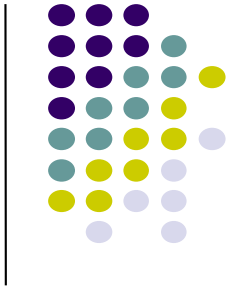
Routing Tables in Datagram Networks



Destination address	Output port
0785	7
1345	12
1566	6
2458	12

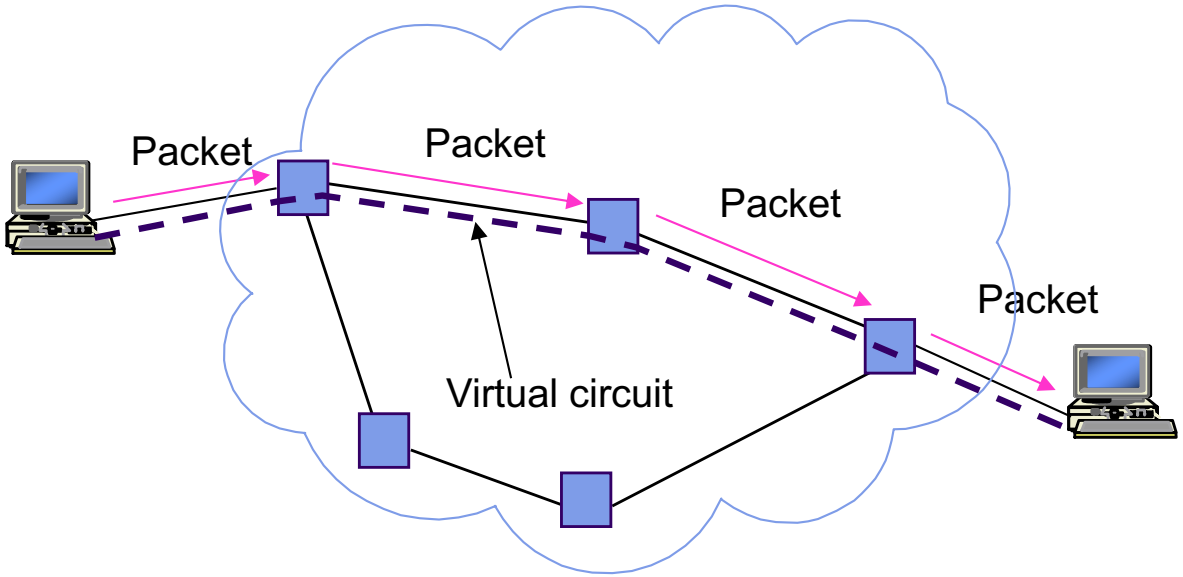
- Route determined by table lookup
- Routing decision involves finding next hop in route to given destination
- Routing table has an entry for each destination specifying output port that leads to next hop
- Size of table becomes impractical for very large number of destinations

Example: Internet Routing



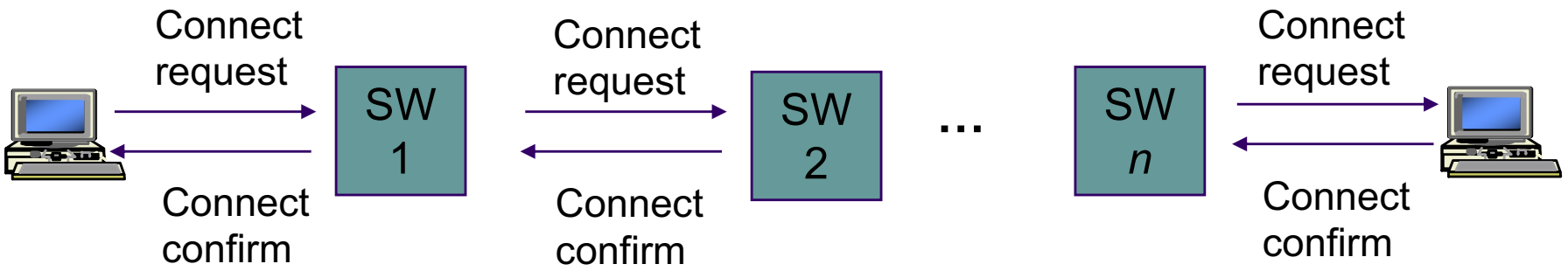
- Internet protocol uses datagram packet switching *across networks*
  - Networks are treated as data links
- Hosts have two-port IP address:
  - Network address + Host address
- Routers do table lookup on network address
  - This reduces size of routing table
- In addition, network addresses are assigned so that they can also be aggregated
  - Discussed as CIDR in Chapter 8

Packet Switching – Virtual Circuit



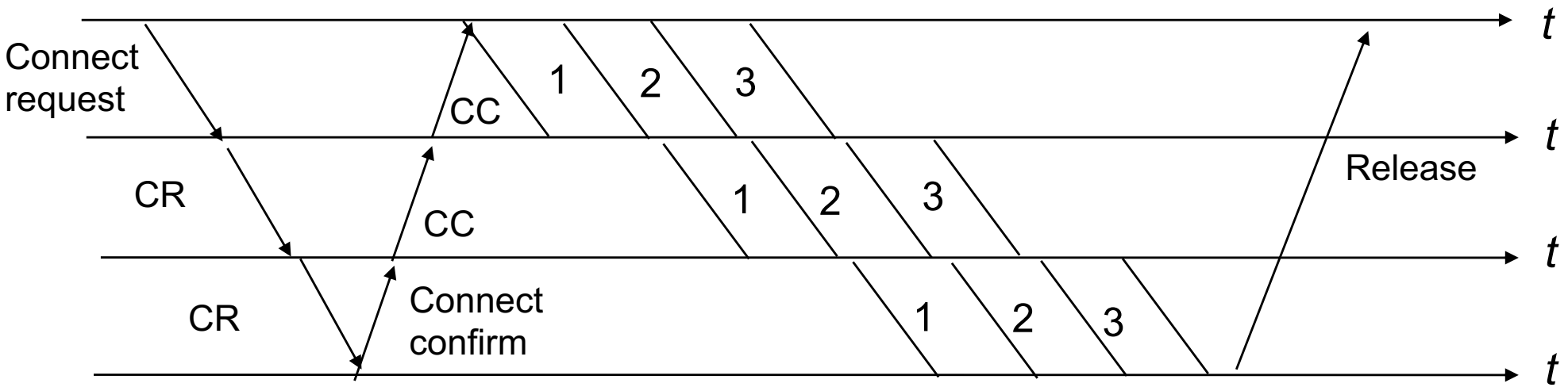
- Call set-up phase sets ups pointers in fixed path along network
- All packets for a connection follow the same path
- Abbreviated header identifies connection on each link
- Packets queue for transmission
- Variable bit rates possible, negotiated during call set-up
- Delays variable, cannot be less than circuit switching

Connection Setup



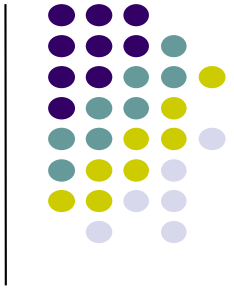
- Signaling messages propagate as path is selected
- Signaling messages give the connection and ID and then establish the connection by setting table entries in the switches
- A connection is identified by a local tag, Virtual Circuit Identifier (VCI), or by a flow identifier
- Each switch entry:
  - Specifies what forwarding/action is to be applied to a packet
  - Relates an incoming tag (or flow identifier) in one input to an outgoing tag in the corresponding output
- Once tables are setup, packets can flow along path

Connection Setup Delay



- Connection setup delay is incurred before any packet can be transferred
- Delay is acceptable for sustained transfer of large number of packets
- This delay may be unacceptably high if only a few packets are being transferred

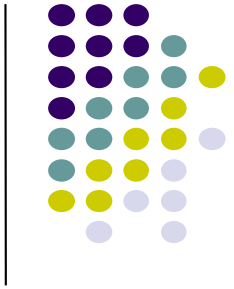
Virtual Circuit Forwarding Tables



Input VCI	Output port	Output VCI
12	13	44
15	15	23
27	13	16
58	7	34

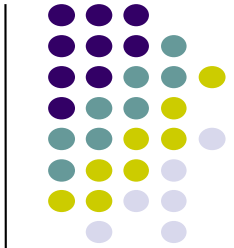
- Each input port of packet switch has a forwarding table
- Lookup entry for VCI of incoming packet
- Determine output port (next hop) and insert VCI for next link
- Very high speeds are possible
- Table can also include priority or other information about how packet should be treated

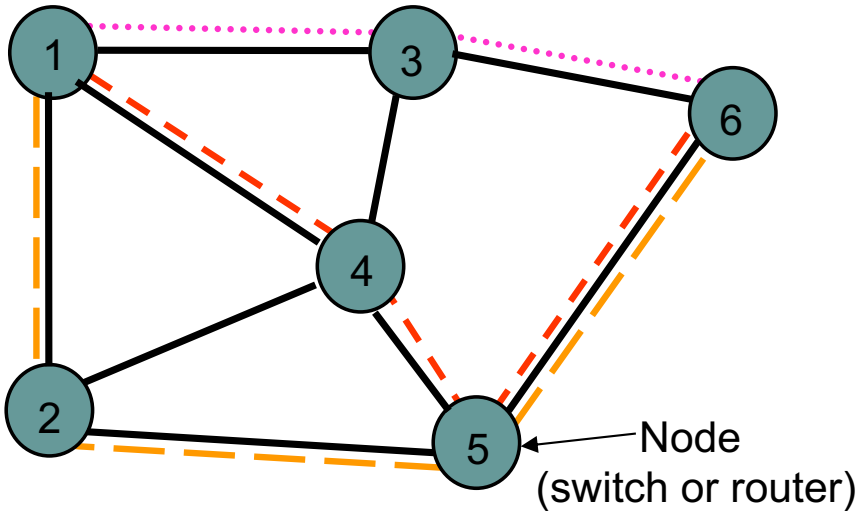
Example: MPLS Networks



- All frames carry a label that identifies the flow
- Connections set up across network
  - Virtual circuits established across networks
  - Tables setup at MPLS switches
- QoS mechanisms can be associated with MPLS flows

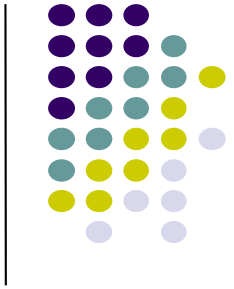
Routing in Packet Networks





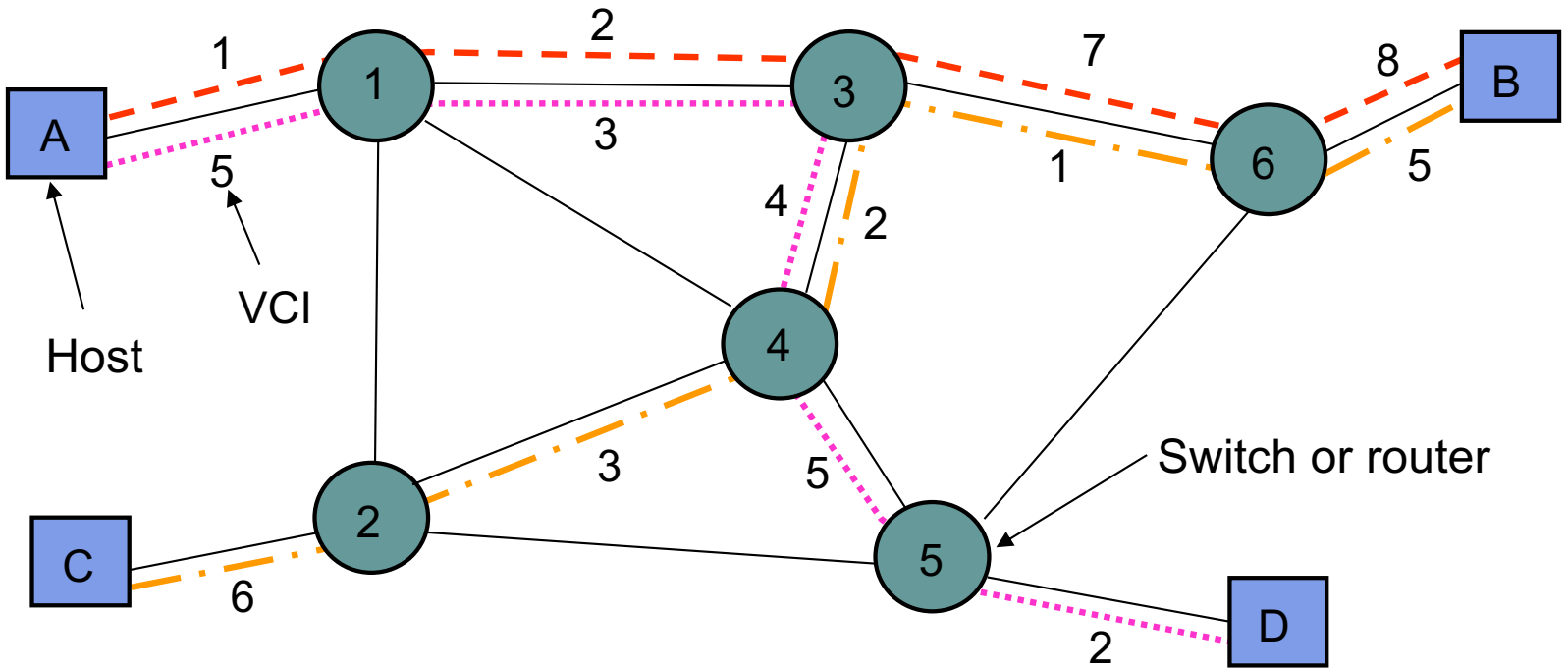
- Three possible (loopfree) routes from 1 to 6:
  - 1-3-6, 1-4-5-6, 1-2-5-6
- Which is “best”?
  - Min delay? Min hop? Max bandwidth? Min cost? Max reliability?

Creating the Routing Tables



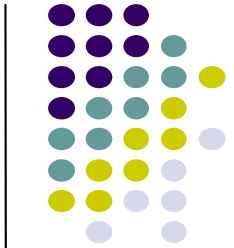
- Need information on state of links
  - Link up/down; congested; delay or other metrics
- Need to distribute link state information using a routing protocol
  - What information is exchanged? How often?
  - Exchange with neighbors; Broadcast or flood
- Need to compute routes based on information
  - Single metric; multiple metrics
  - Single route; alternate routes

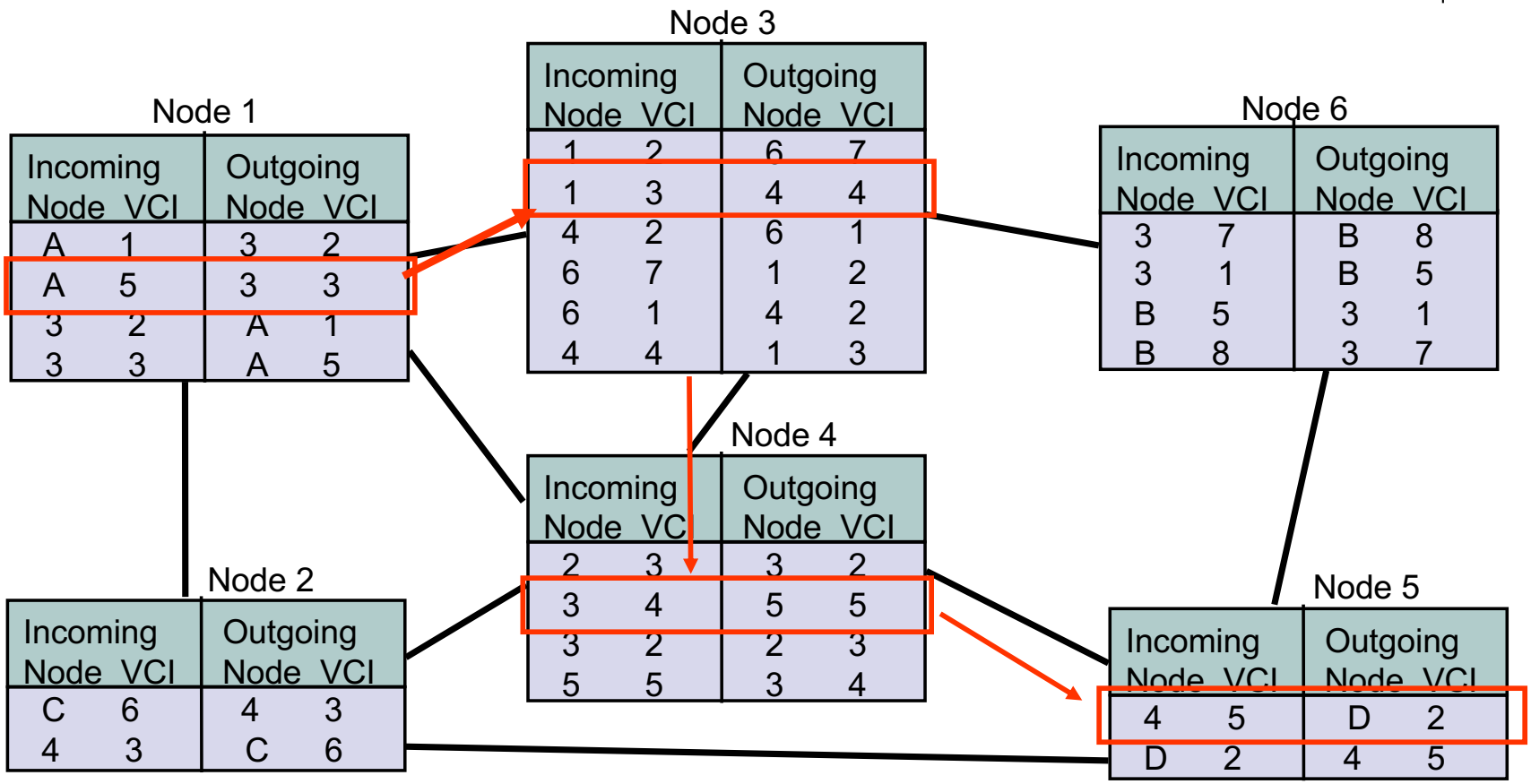
Routing in Virtual-Circuit Packet Networks



- Route determined during connection setup
- Tables in switches implement forwarding that realizes selected route

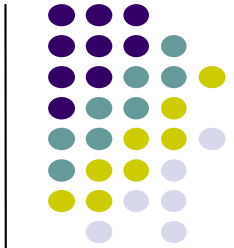
Routing Tables in VC Packet Networks

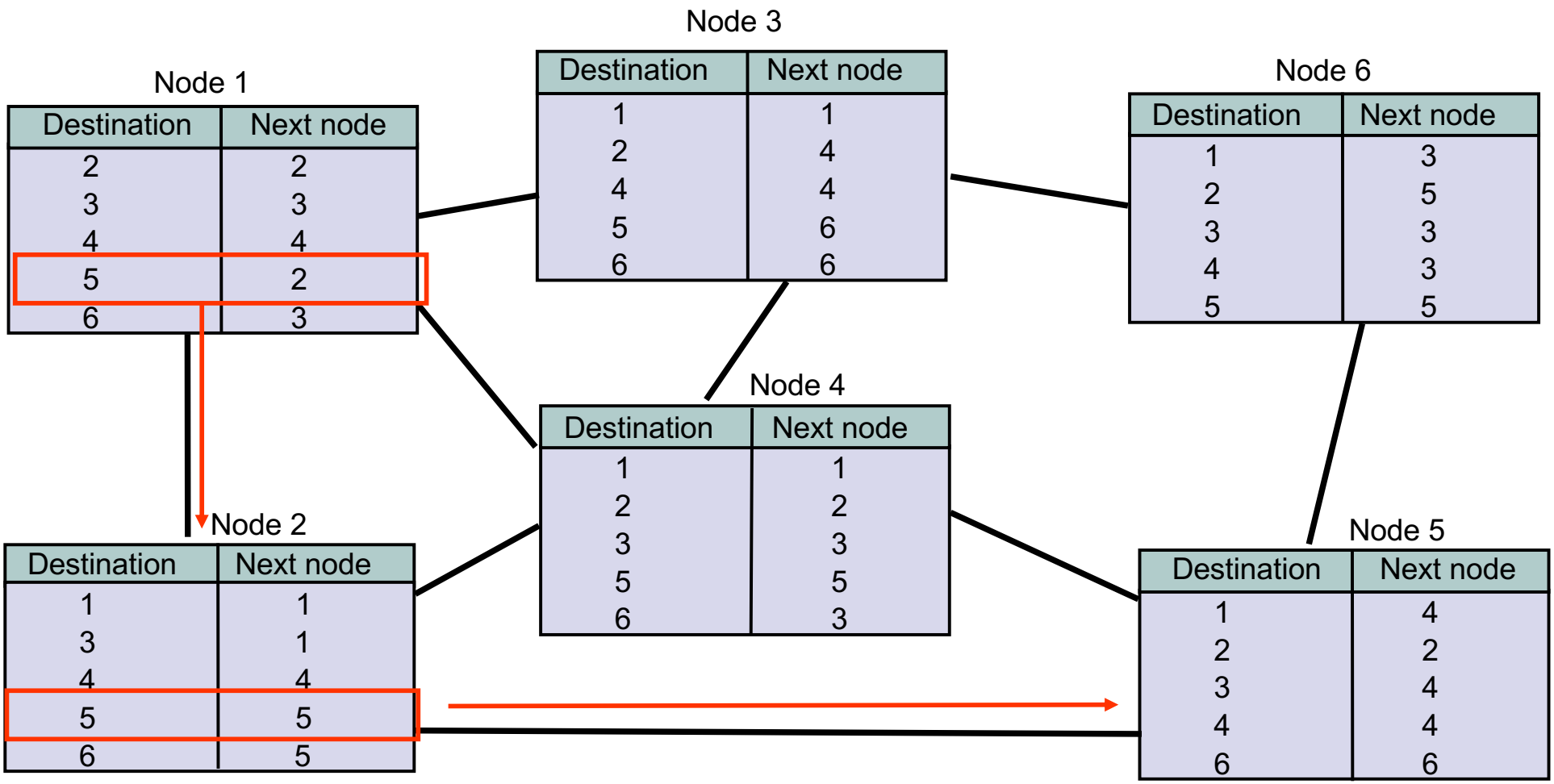




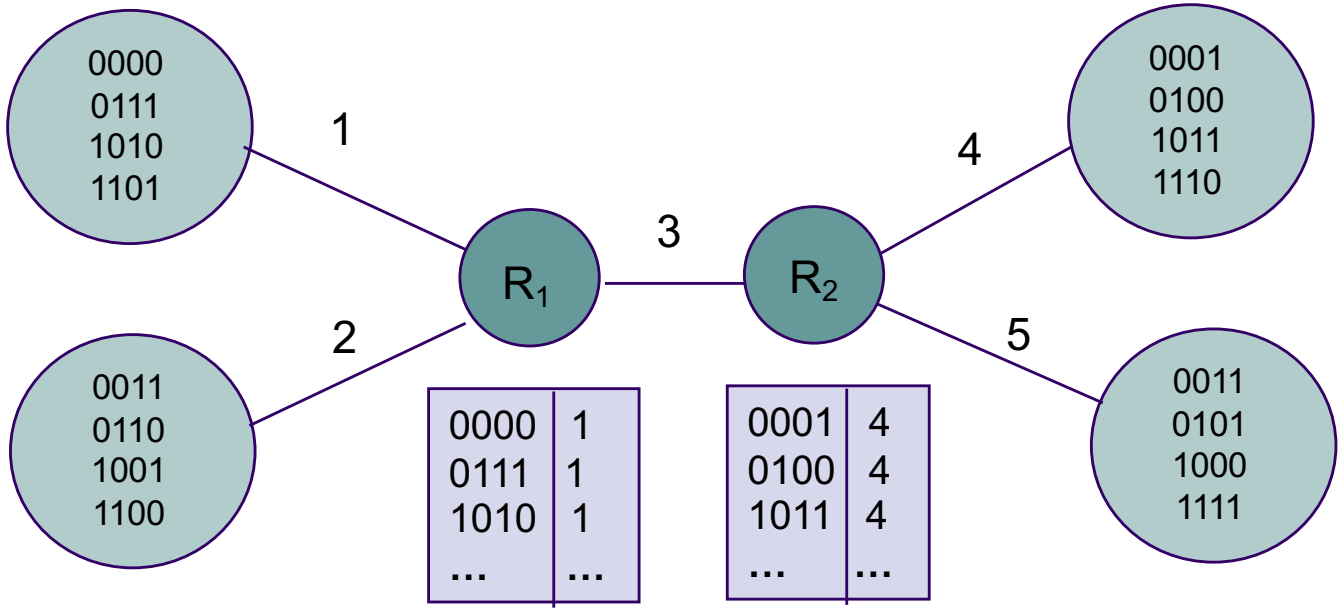
- Example: VCI from A to D
  - From A & VCI 5 → 3 & VCI 3 → 4 & VCI 4
  - → 5 & VCI 5 → D & VCI 2

Routing Tables in Datagram Packet Networks



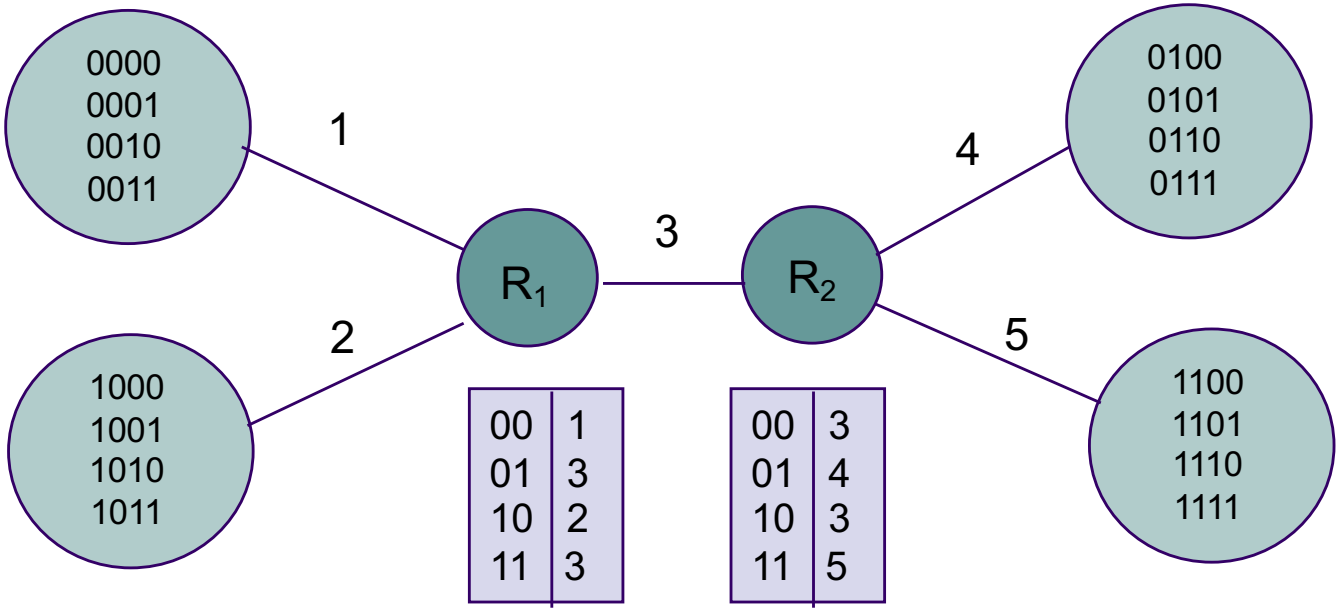


Non-Hierarchical Addresses and Routing



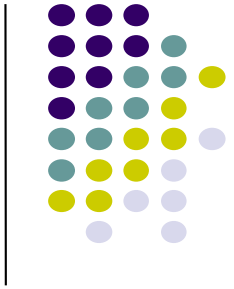
- No relationship between addresses & routing proximity
- Routing tables require 16 entries each

Hierarchical Addresses and Routing



- Prefix indicates network where host is attached
- Routing tables require 4 entries each

Flooding

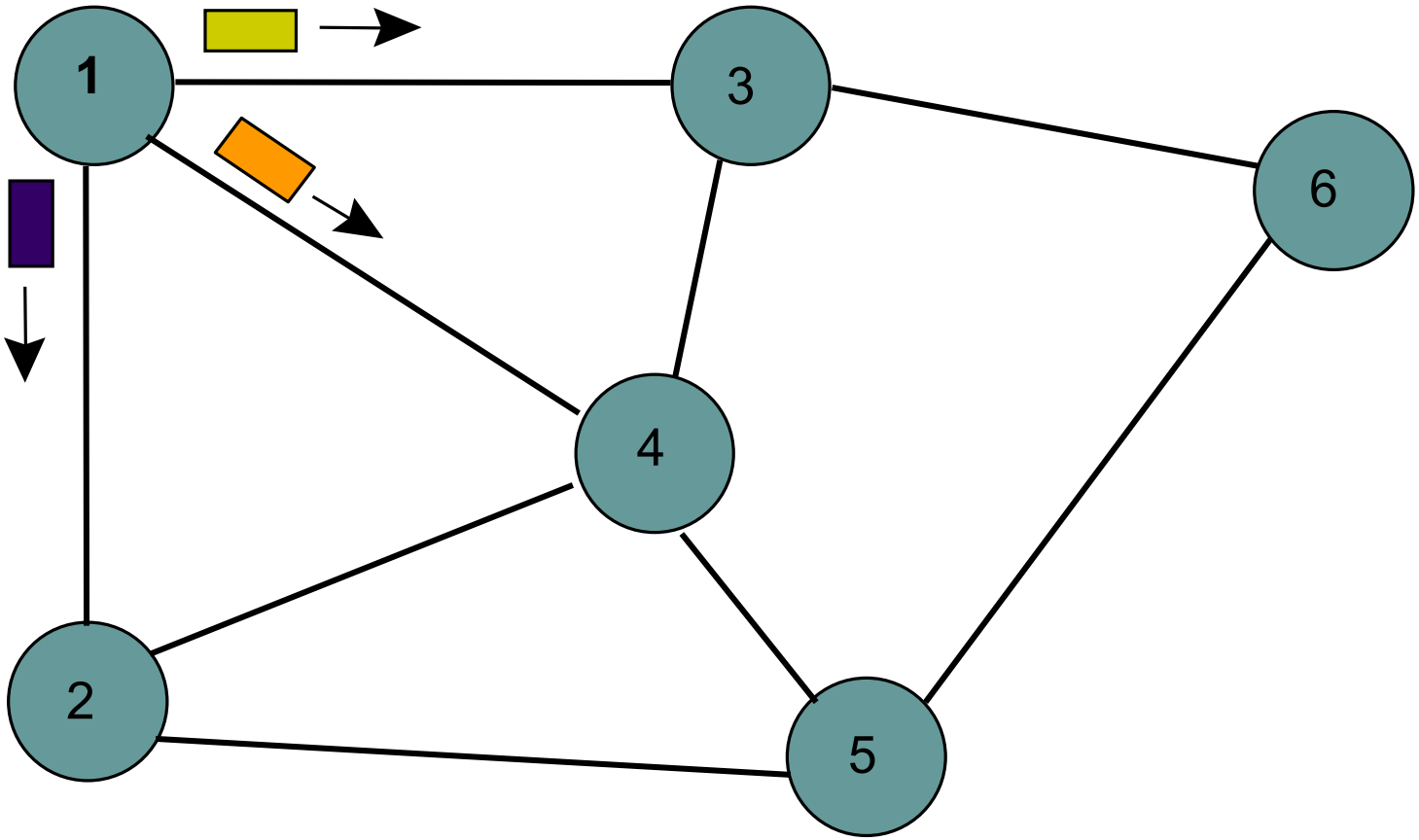


Send a packet to all nodes in a network

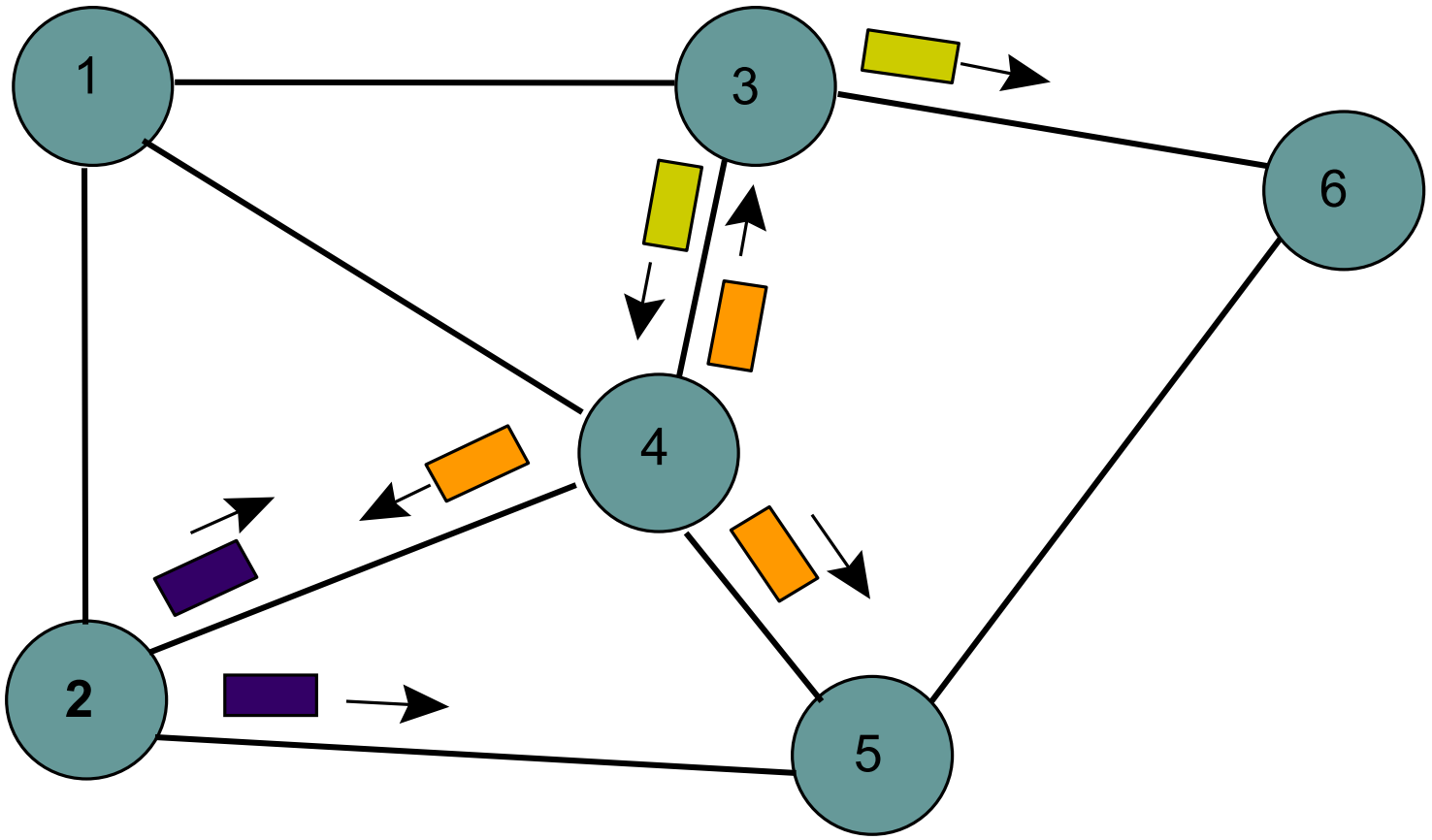
- No routing tables available
- Need to broadcast packet to all nodes (e.g. to propagate link state information)

Approach

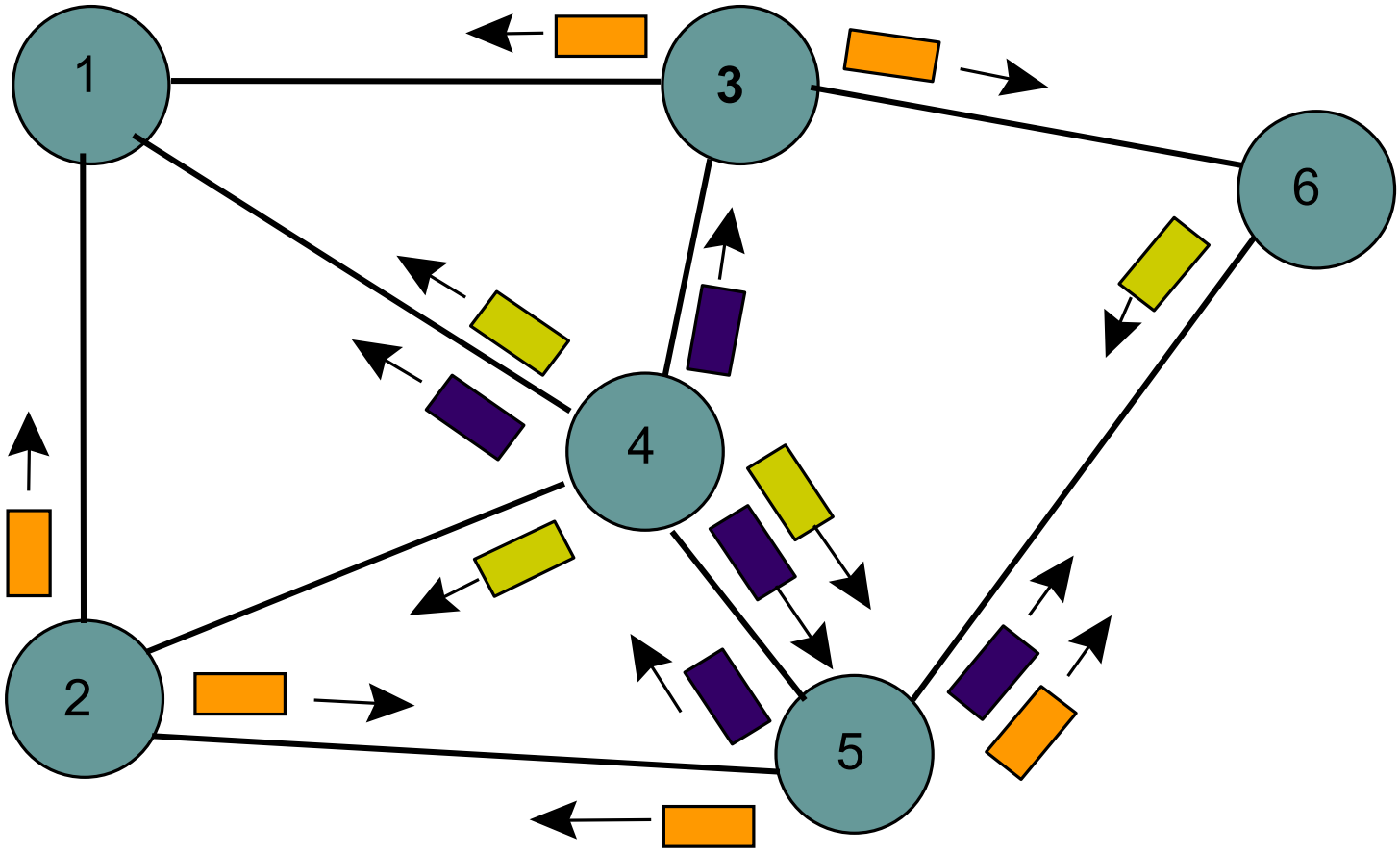
- Send packet on all ports except one where it arrived
- Exponential growth in packet transmissions



Flooding is initiated from Node 1: Hop 1 transmissions

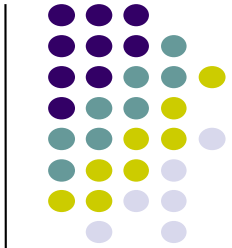


Flooding is initiated from Node 1: Hop 2 transmissions



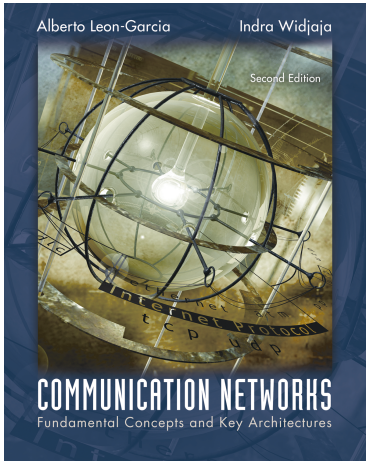
Flooding is initiated from Node 1: Hop 3 transmissions

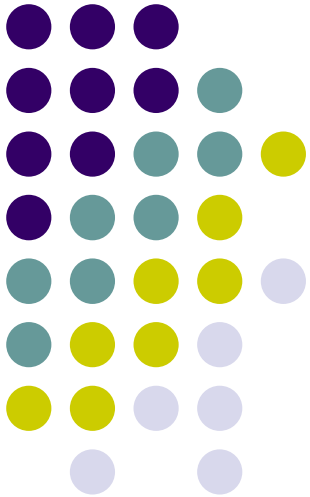
Limited Flooding



- Time-to-Live field in each packet limits number of hops to certain diameter
- Each switch adds its ID before flooding; discards repeats
- Source puts sequence number in each packet; switches records source address and sequence number and discards repeats

Advanced Network Architectures



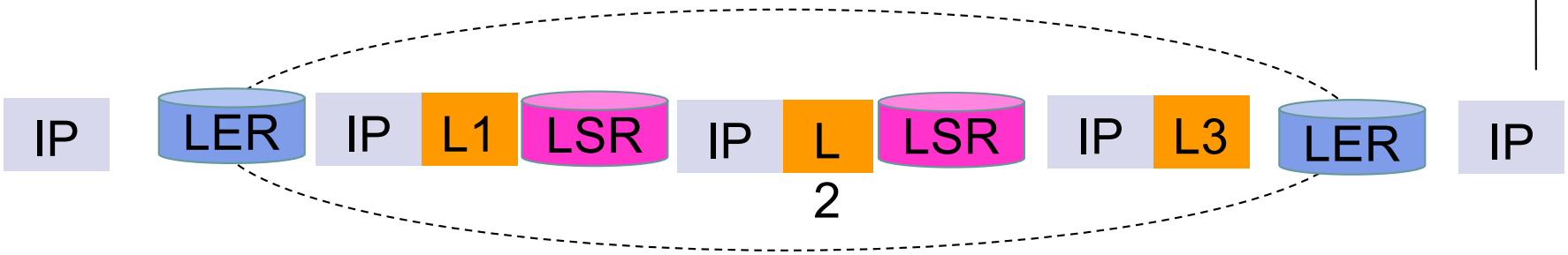


MPLS

pp 685-693

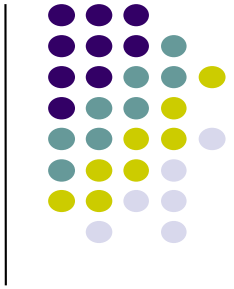
OpenFlow

What is MPLS?



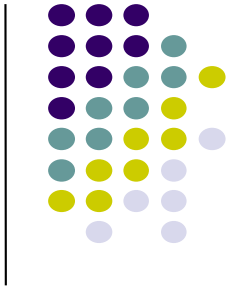
- *Multiprotocol Label Switching (MPLS)*
- A set of protocols that enable MPLS networks
  - Packets are assigned *labels* by edge routers (which perform longest-prefix match)
  - Packets are forwarded along a *Label-Switched Path (LSP)* in the MPLS network using label switching
  - LSPs can be created over *multiple layer-2 links*
    - ATM, Ethernet, PPP, frame relay
  - LSPs can support *multiple layer-3 protocols*
    - IPv4, IPv6, and in others

Why MPLS?

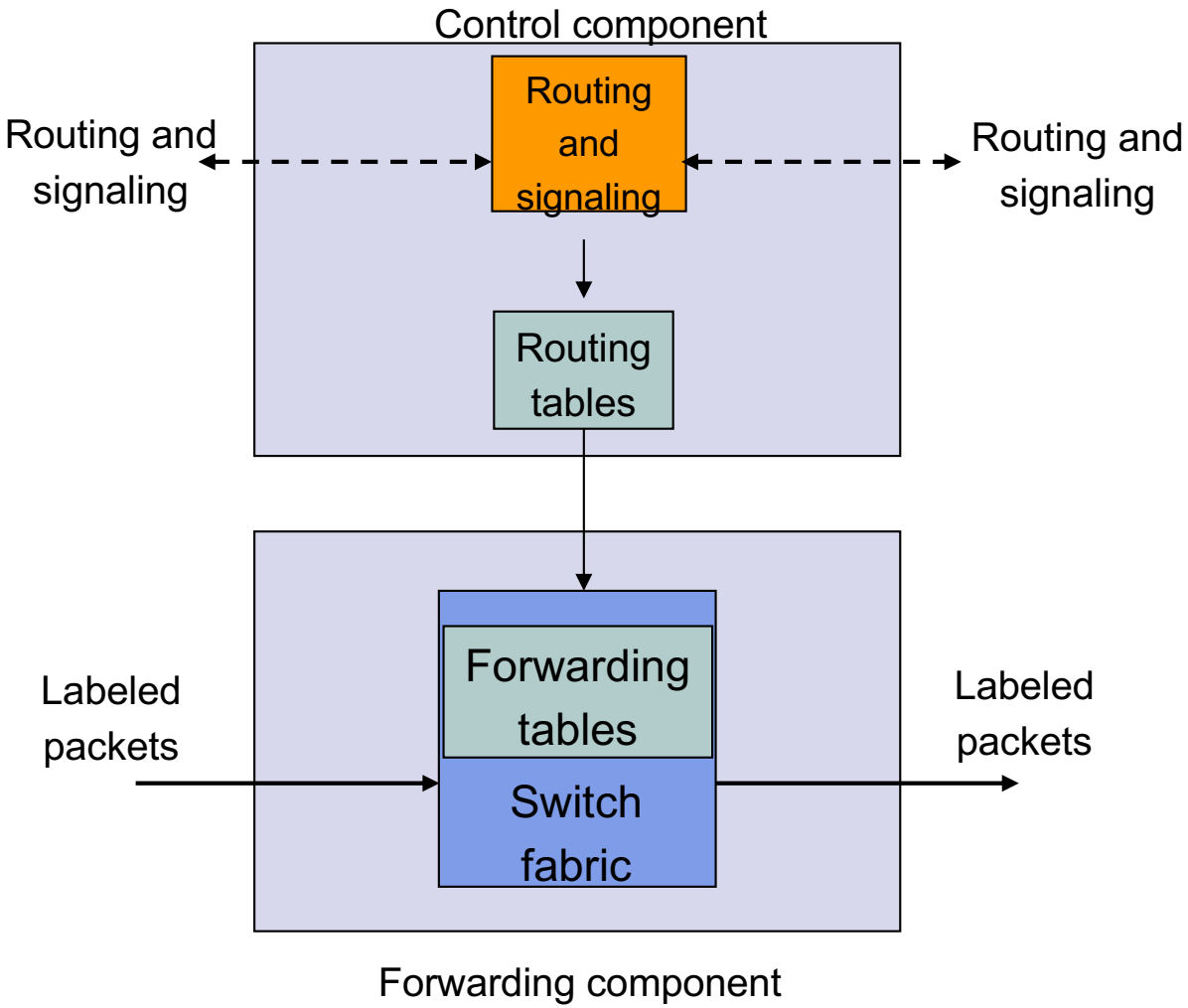


- Labels enable fast forwarding
  - But longest-prefix match is also fast
- Circuits are good (sometimes)
  - Conventional IP routing selects one path, does not provide choice of route
  - Label switching enables routing flexibility
  - *Traffic engineering*: establish separate paths to meet different performance requirements of aggregated traffic flows
  - *Virtual Private Networks*: establish tunnels between user nodes

Separation of Forwardng & Control



All proposals leading to MPLS separate forwarding and control



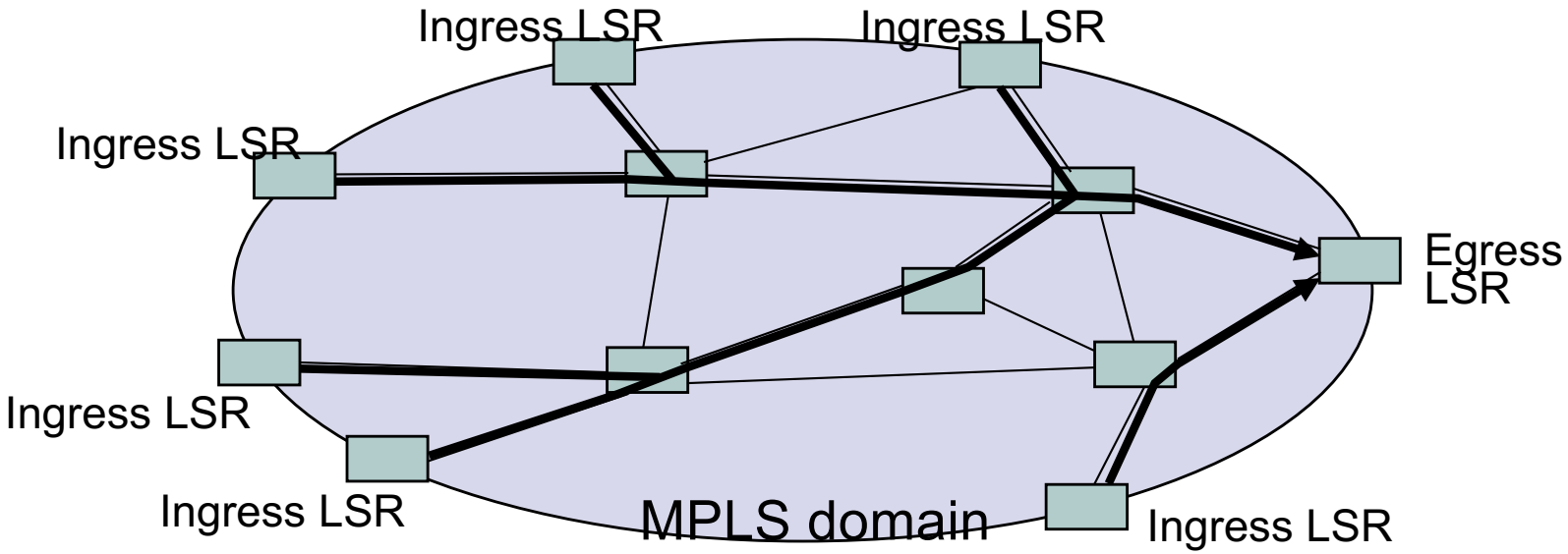
Before MPLS: forwarding & control intertwined

- Transition to CIDR (control) meant forwarding had to change to longest-prefix match

With MPLS: forwarding & control are separate

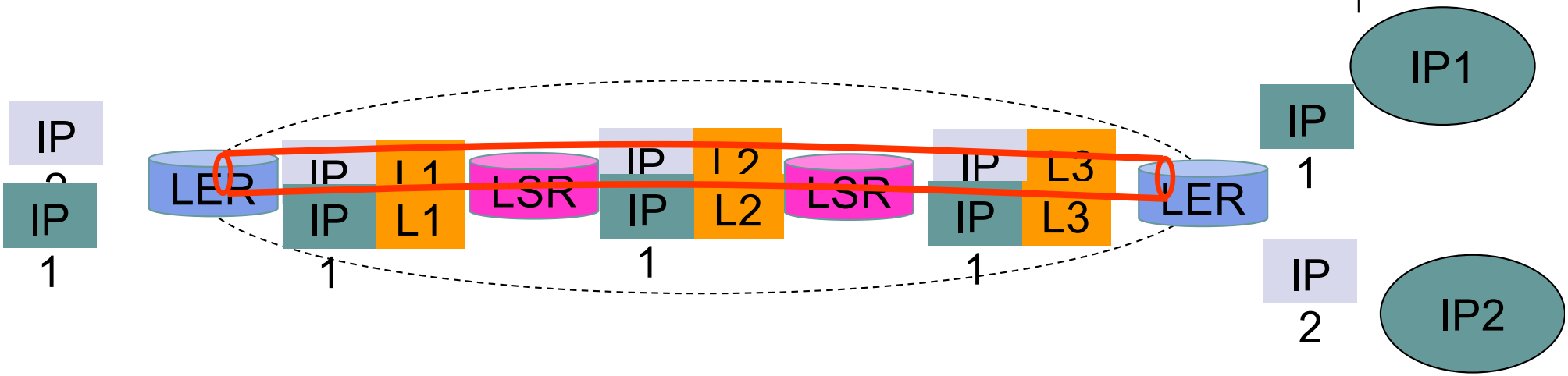
- All forwarding done with label switching
- Different control schemes dictate creation of labels & label-switched paths
- Control & forwarding can evolve independently

Labels and Paths



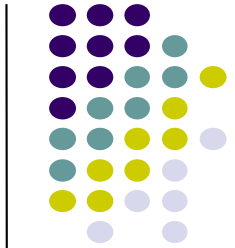
- Label-switched paths (LSPs) are *unidirectional*
- LSPs can be:
  - point-to-point
  - *tree rooted in egress node* corresponds to shortest paths leading to a destination egress router

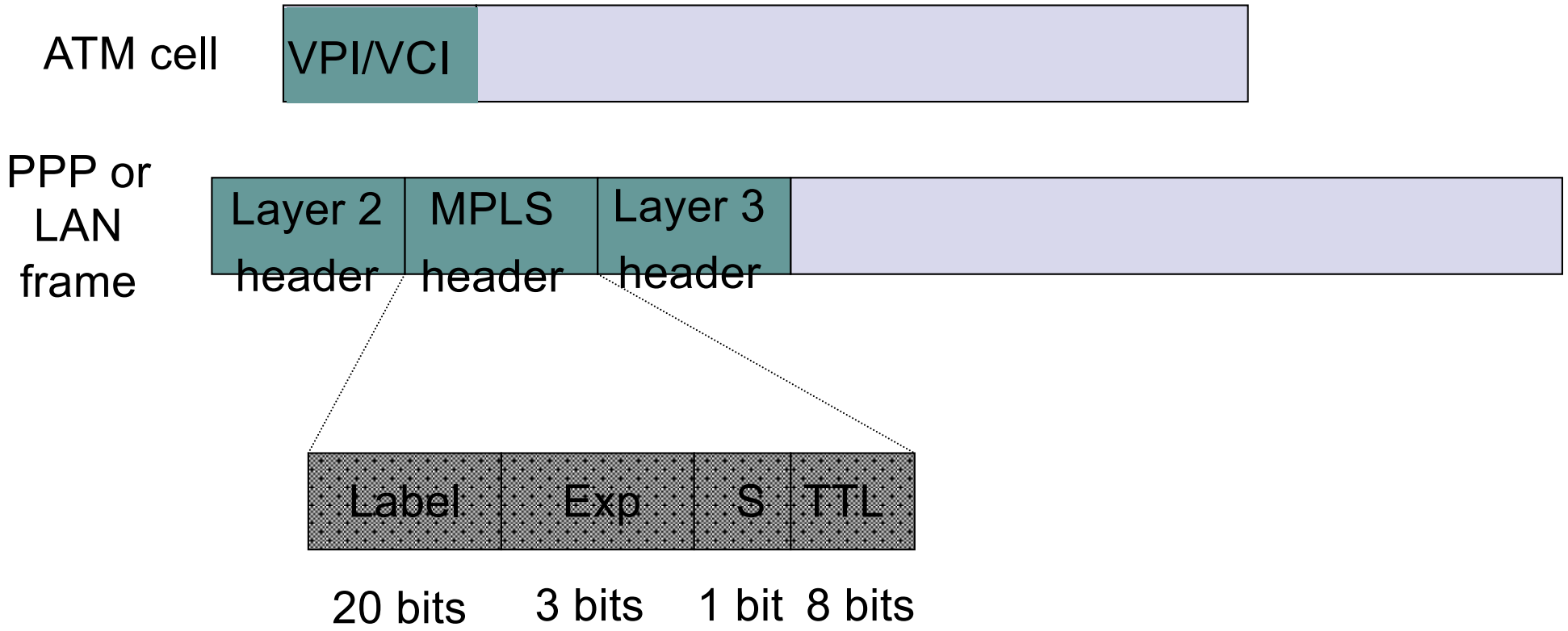
Forwarding Equivalence Class



- *FEC:* set of packets that are forwarded in the same manner
  - Over the same path, with the same forwarding treatment
  - Packets in an FEC have same next-hop router
  - Packets in same FEC may have different network layer header
  - Each FEC requires a *single entry* in the forwarding table
  - Coarse Granularity FEC: packets for all networks whose destination address matches a given address prefix
  - Fine Granularity FEC: packets that belong to a particular application running between a pair of computers

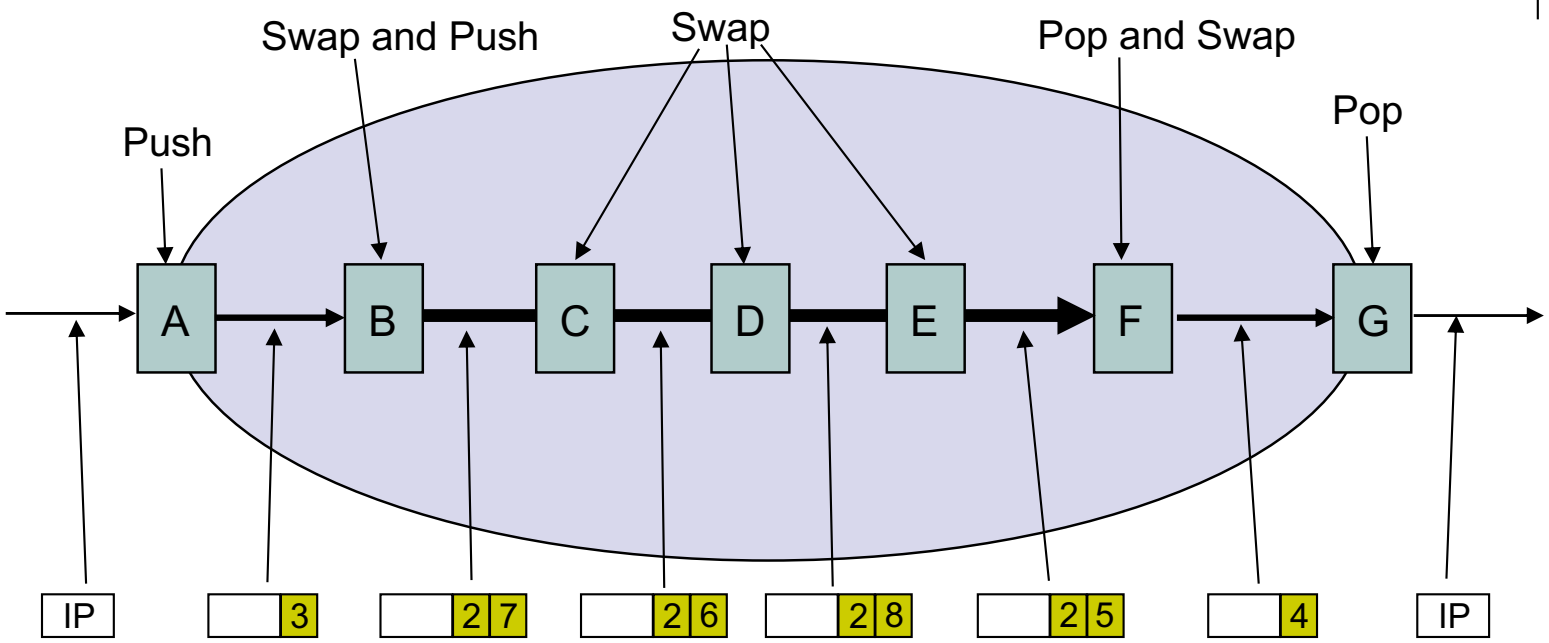
MPLS Labels





- Labels can be encoded into VPI/VCI field of ATM header
- *Shim header* between layer 2 & layer 3 header (32 bits)
  - 20-bit label + 1-bit hierarchical stack field + 8-bit TTL
  - 3-bit "experimental" field (can be used to specity 8 DiffServ PHBs)

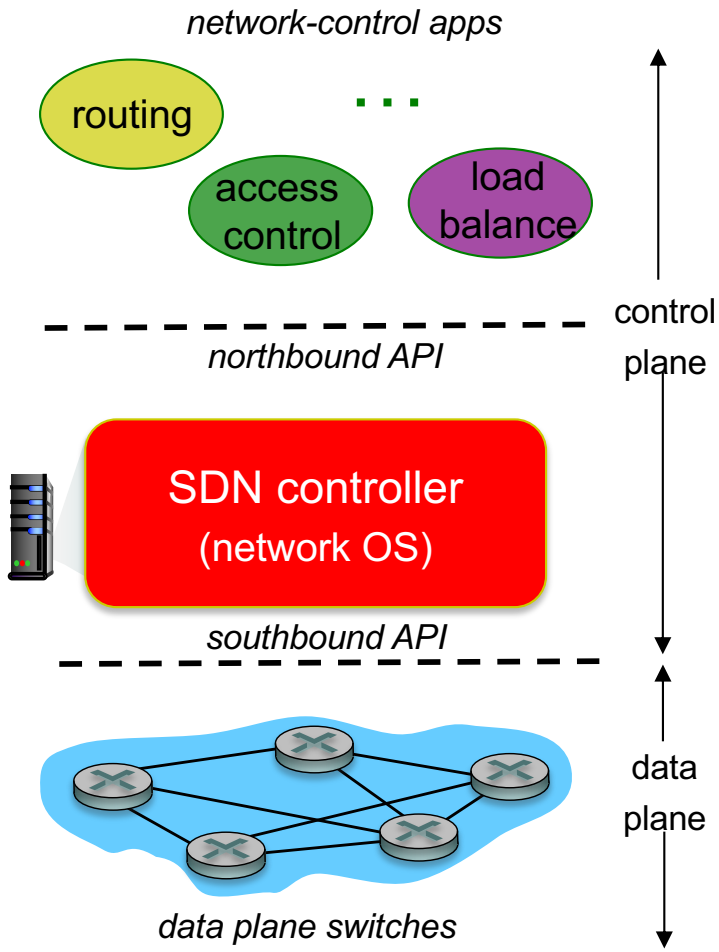
Label Stacking



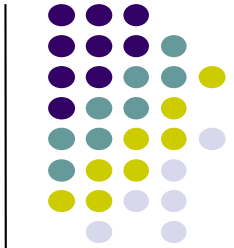
- MPLS allows multiple labels to be stacked
  - Ingress LSR performs *label push* (S=1 in label)
  - Egress LSR performs *label pop*
  - Intermediate LSRs can perform additional pushes & pops (S=0 in label) to create tunnels
  - Above figure has tunnel between A & G; tunnel between B&F
  - All flows in a tunnel share the same outer MPLS label

Software-Defined Networking (SDN)

- Flexible networks defined by software
- Data plane switches
  - SDN controlled switches
  - Modified Ethernet switches
- South Bound API
  - Interface between control & data plane (e.g. OpenFlow)
- SDN Controller
  - Centralized Control
- North Bound API
  - Interface between controller and control apps
  - Program network applications into the network
  - Firewalls, tapping, QoS, …



SDN switches & Flow Tables

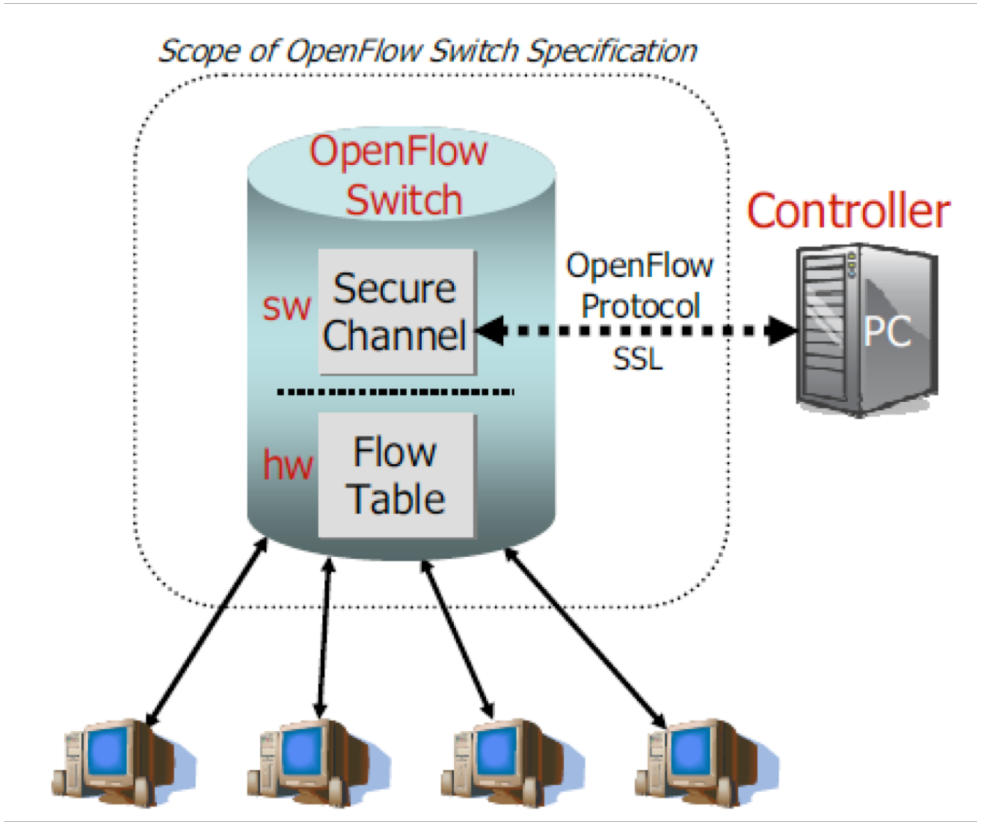


- Programmable Network Devices
  - Fast, simple, commodity switches implement forwarding in hardware
  - Can be software (virtual switch)
  - Typically no control functionality built-in
  - Programmed by a remote controller that installs switch flow tables

- Table-based switch control (flow rules)
  - Packets are matched on header fields (using TCAM hardware)
  - Actions are applied to matched packets
    - Forward, drop, en/de-capsulate, clone, etc.
  - If no match is found in table, device asks controller for instructions.

MAC src	MAC dst	IP Src	IP Dst	TCP sport	TCP dport	Action
*	*	*	*	*	80	port 4

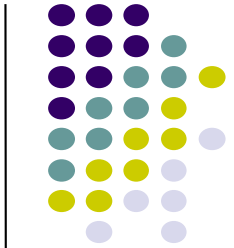
OpenFlow Switch



1. Flow table associates action with each flow
2. Secure channel connects switch to controller allowing exchange of commands & packets
3. OpenFlow Protocol provides open secure protocol to communicate with a switch

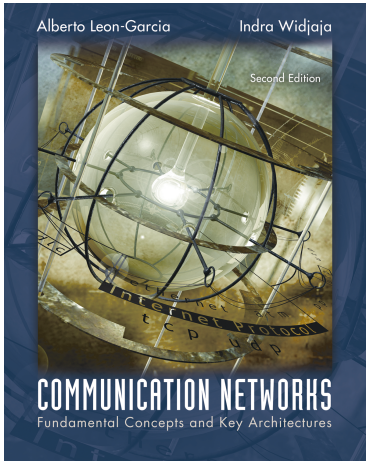
- Admin can separate production & experimental traffic
- Researchers control their own flows

Examples



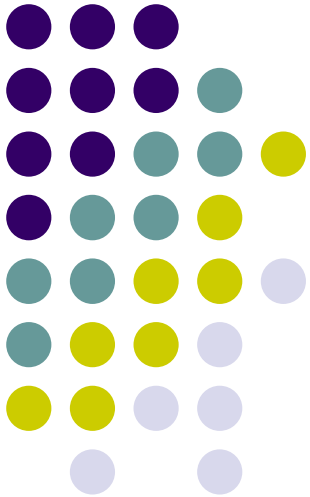
- Network Management & Access Control
  - Controller manages all bindings of names & addresses
- VLANs
  - Statically declares a set of flows which specify the ports accessible by traffic with a given VLAN ID
  - Dynamically manage authentication of mobile users
- Mobile Wireless VOIP
  - Controller implements handoff by reprogramming tables
- Non-IP Network
  - Flow table handles new header formats
- Processing packets not flows (e.g. intrusion detection)
  - Force all packets to go to controller
  - Route packets to programmable switch

Chapter 7
Packet-Switching Networks

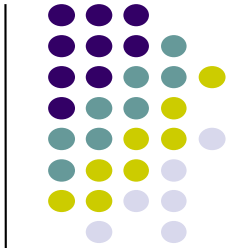


Shortest Path Routing

pp 490-498



Shortest Paths & Routing



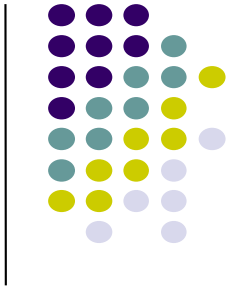
- Many possible paths connect any given source and to any given destination
- Routing involves the selection of the path to be used to accomplish a given transfer
- Typically it is possible to attach a cost or distance to a link connecting two nodes
- Routing can then be posed as a shortest path problem

Routing Metrics

Means for measuring desirability of a path

- Path Length = sum of costs or distances
- Possible metrics
  - Hop count: rough measure of resources used
  - Reliability: link availability; BER
  - Delay: sum of delays along path; complex & dynamic
  - Bandwidth: “available capacity” in a path
  - Load: Link & router utilization along path
  - Cost: $$$

Shortest Path Approaches



Distance Vector Protocols

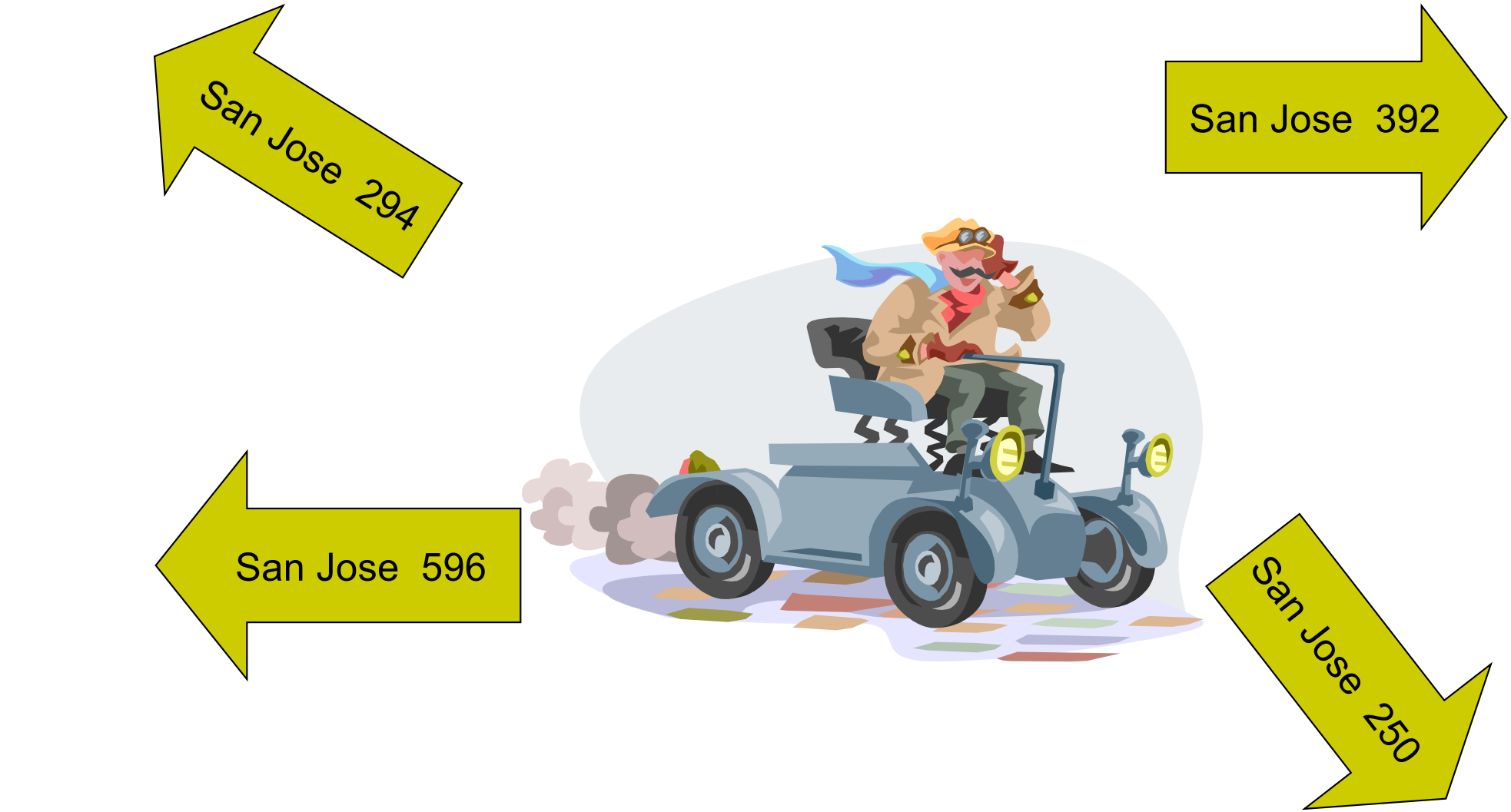
- Neighbors exchange list of distances to destinations
- Best next-hop determined for each destination
- Ford-Fulkerson (distributed) shortest path algorithm

Link State Protocols

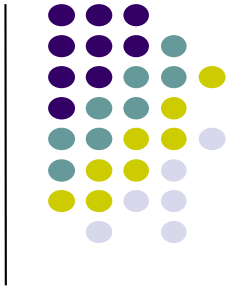
- Link state information flooded to all routers
- Routers have complete topology information
- Shortest path (& hence next hop) calculated
- Dijkstra (centralized) shortest path algorithm

Distance Vector

Do you know the way to San Jose?



Distance Vector



Local Signpost

- Direction
- Distance

Routing Table

For each destination list:

- Next Node
- Distance

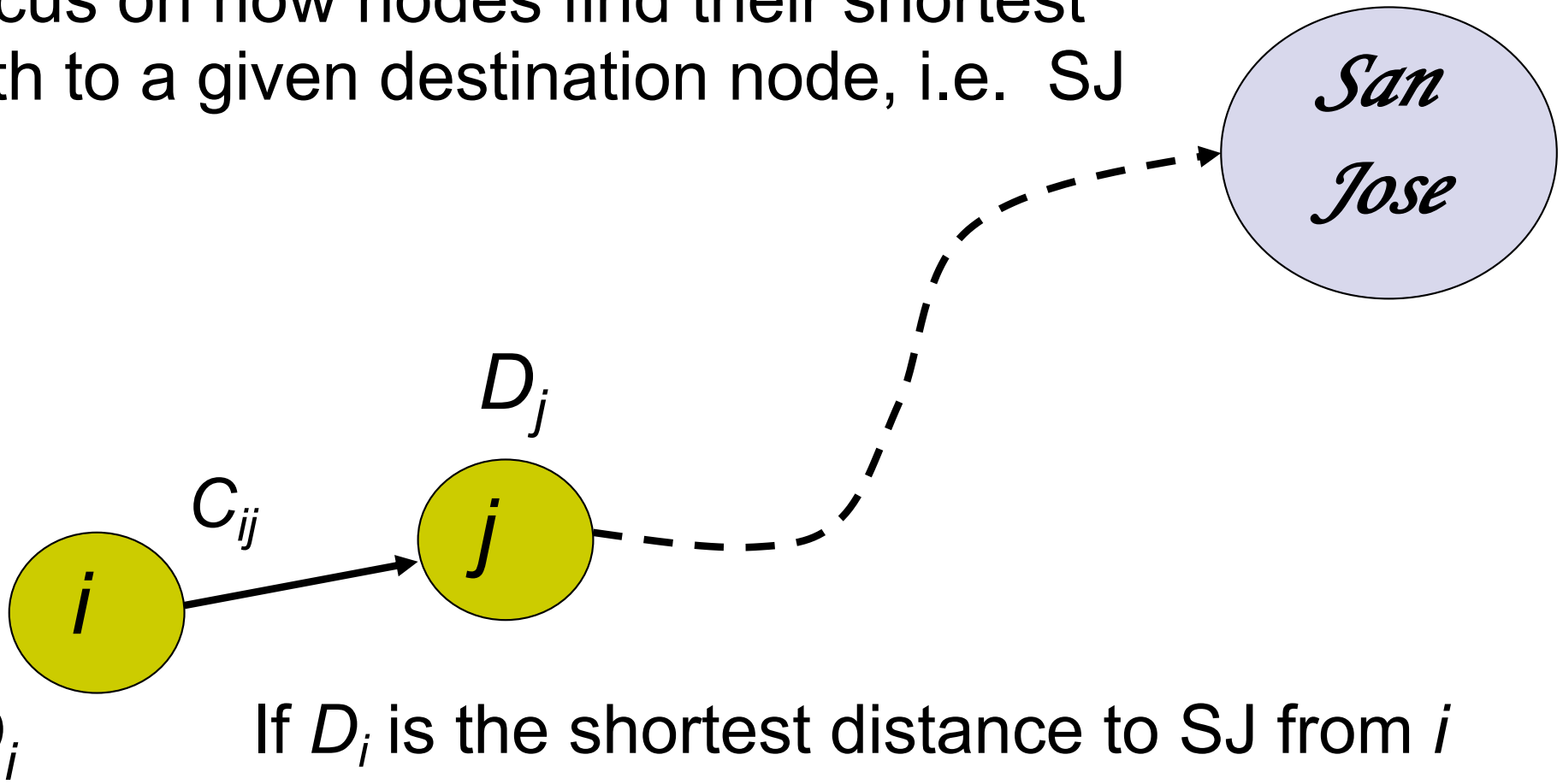
dest	next	dist

Table Synthesis

- Neighbors exchange table entries
- Determine current best next hop
- Inform neighbors
  - Periodically
  - After changes

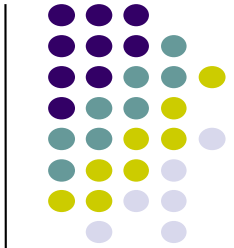
Shortest Path to SJ

Focus on how nodes find their shortest path to a given destination node, i.e. SJ

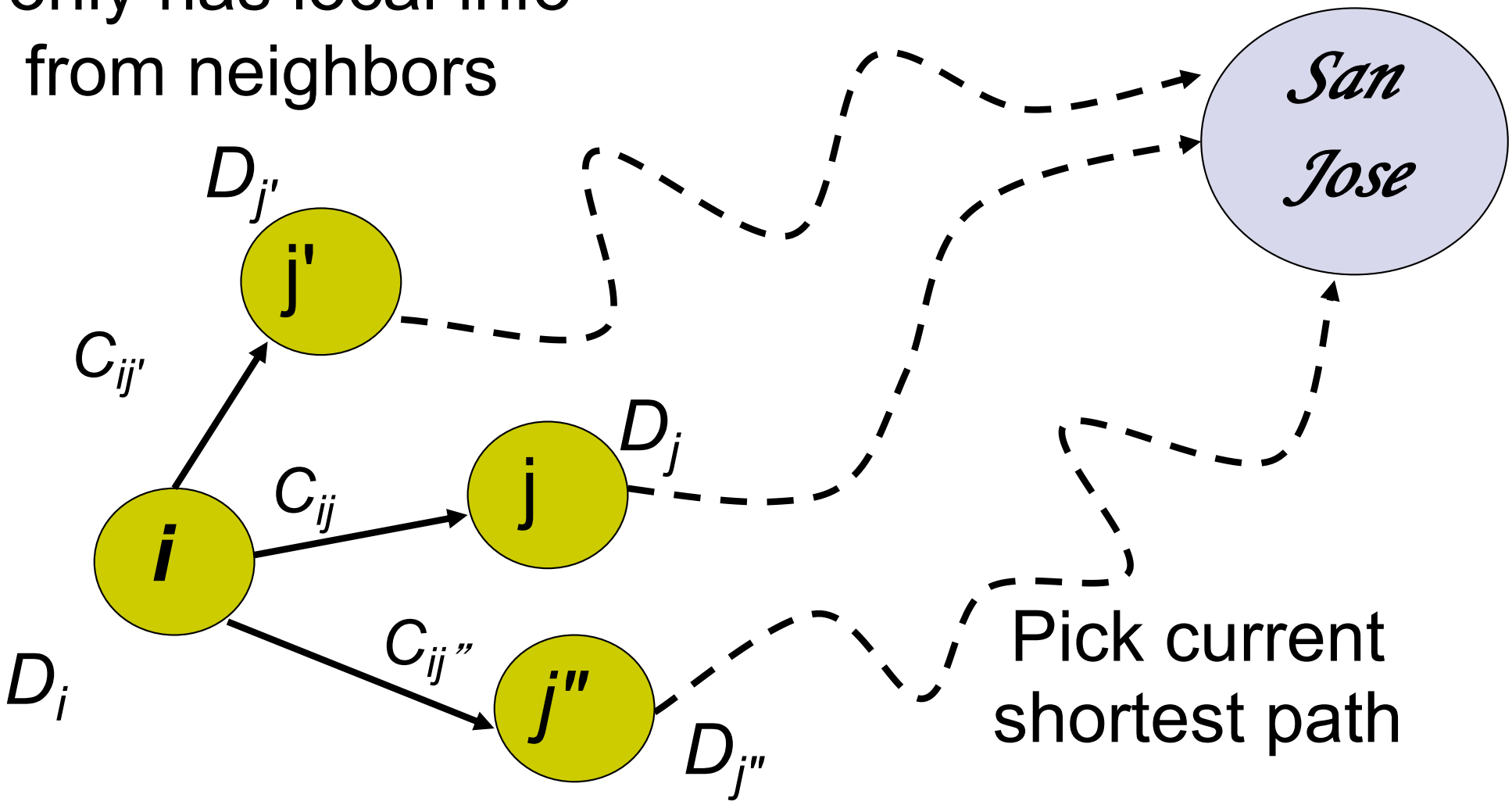


If D_i is the shortest distance to SJ from *i* and if *j* is a neighbor on the shortest path, then $D_i = C_{ij} + D_j$

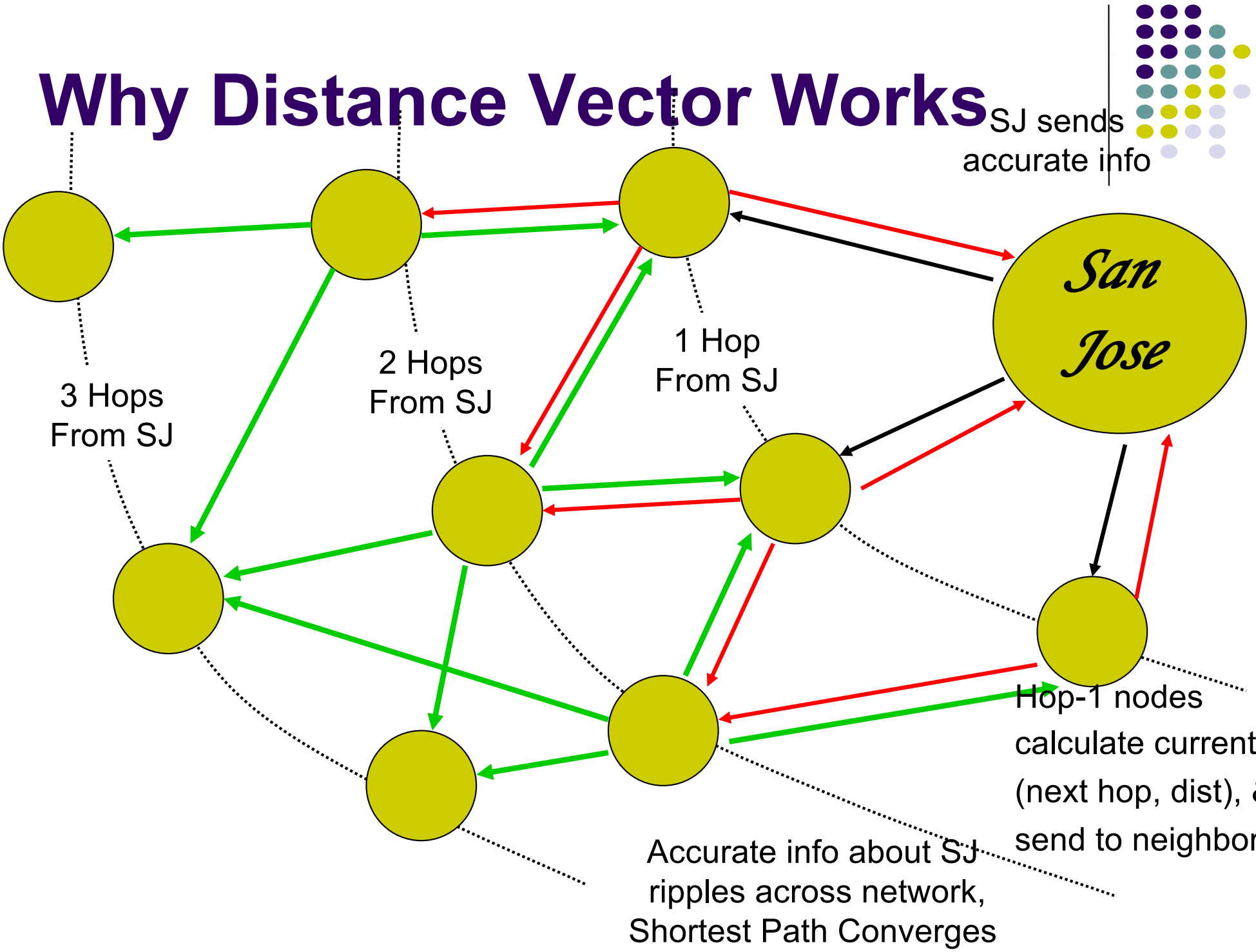
But we don't know the shortest paths



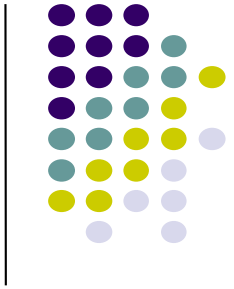
i only has local info from neighbors



Why Distance Vector Works

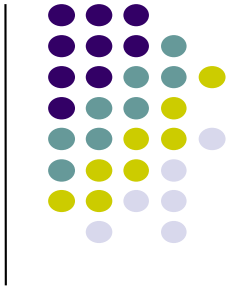


Bellman-Ford Algorithm



- *Consider computations for one destination d*
- *Initialization*
  - Each node table has 1 row for destination *d*
  - Distance of node *d* to itself is zero: $D_d=0$
  - Distance of other node *j* to *d* is infinite: $D_j=\infty$, for $j \neq d$
  - Next hop node n_j = -1 to indicate not yet defined for $j \neq d$
- *Send Step*
  - Send new distance vector to immediate neighbors across local link
- *Receive Step*
  - At node *j*, find the next hop that gives the minimum distance to *d*,
    - $Min_j \{ C_{ij} + D_j \}$
    - Replace old $(n_j, D_j(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance
  - Go to send step

Bellman-Ford Algorithm

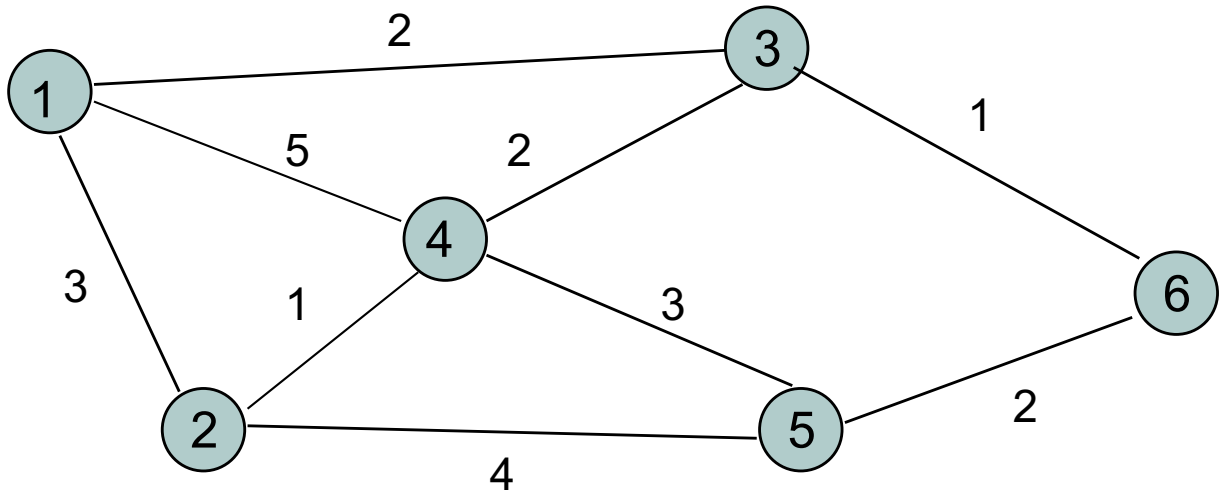


- *Now consider parallel computations for all destinations d*
- *Initialization*
  - Each node has 1 row for each destination *d*
  - Distance of node *d* to itself is zero: $D_d(d)=0$
  - Distance of other node *j* to *d* is infinite: $D_j(d)= \infty$, for $j \neq d$
  - Next node n_j = -1 since not yet defined
- *Send Step*
  - Send new distance vector to immediate neighbors across local link
- *Receive Step*
  - For each destination *d*, find the next hop that gives the minimum distance to *d*,
    - $Min_j \{ C_{ij} + D_j(d) \}$
    - Replace old $(n_j, D_i(d))$ by new $(n_j^*, D_j^*(d))$ if new next node or distance found
  - Go to send step

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1					
2					
3					

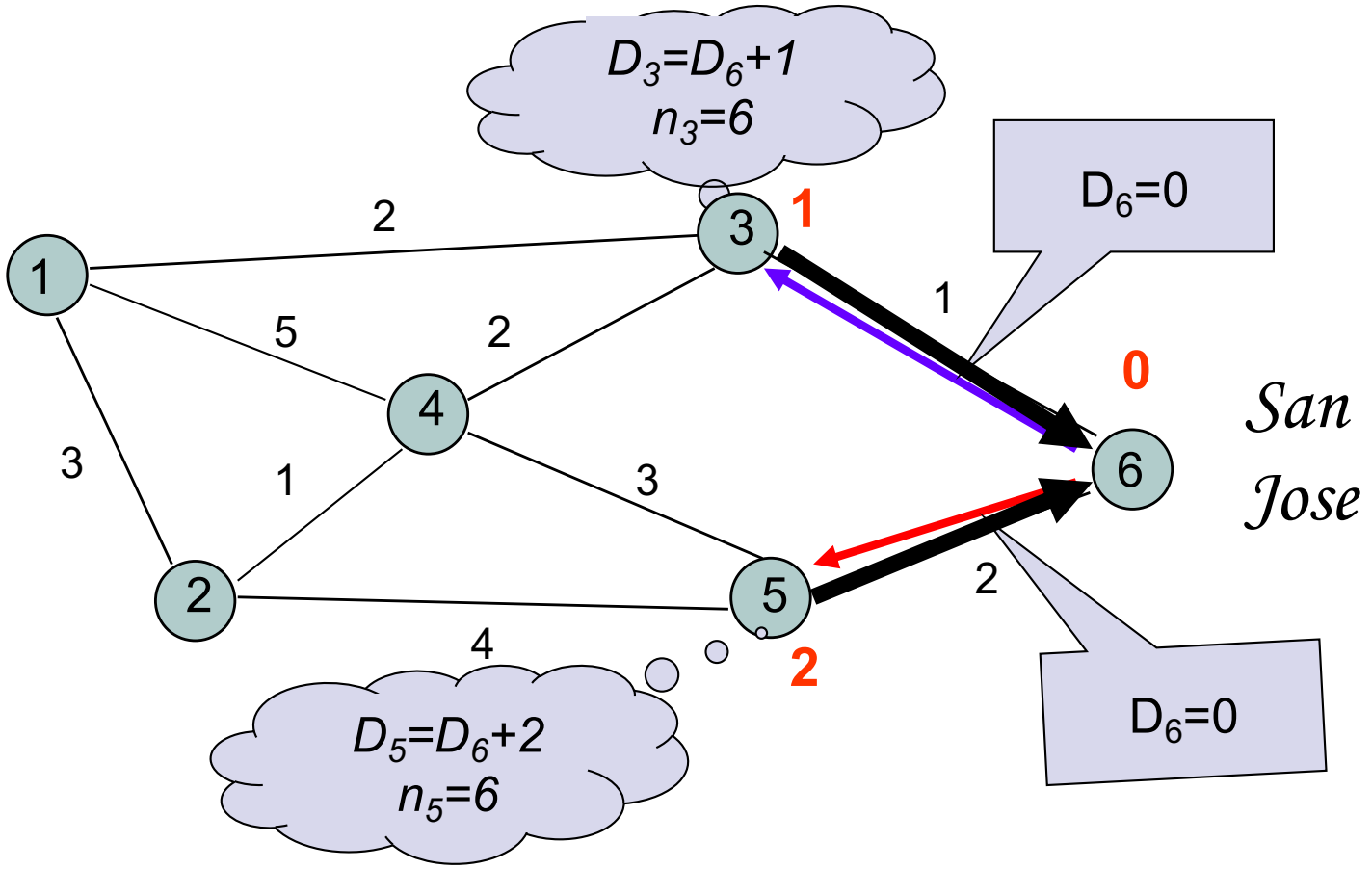
Table entry @ node 1 for dest SJ

Table entry @ node 3 for dest SJ

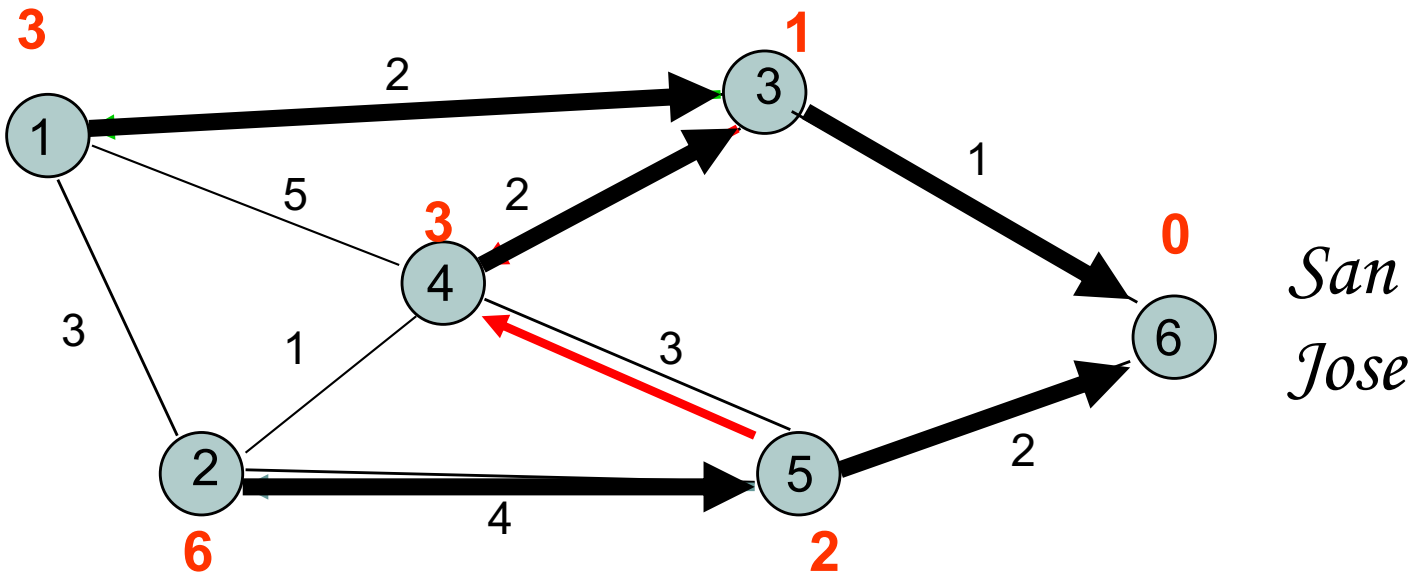


San Jose

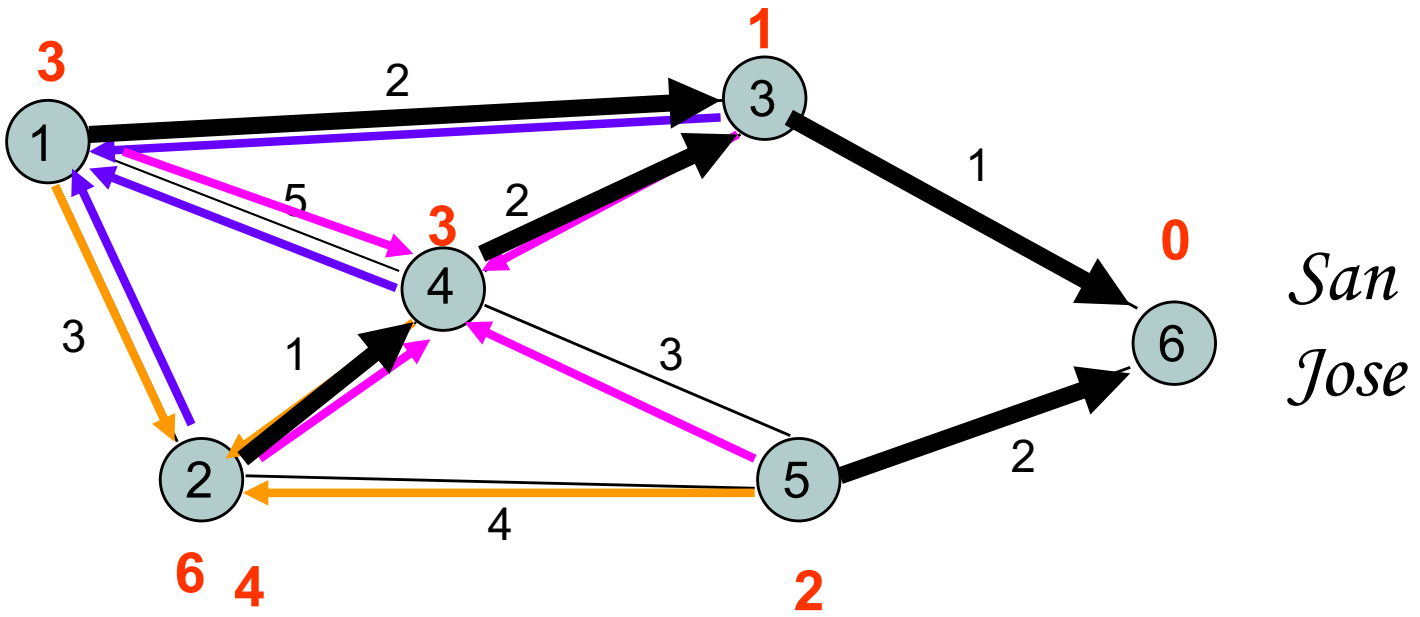
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6,1)	(-1, ∞)	(6,2)
2					
3					



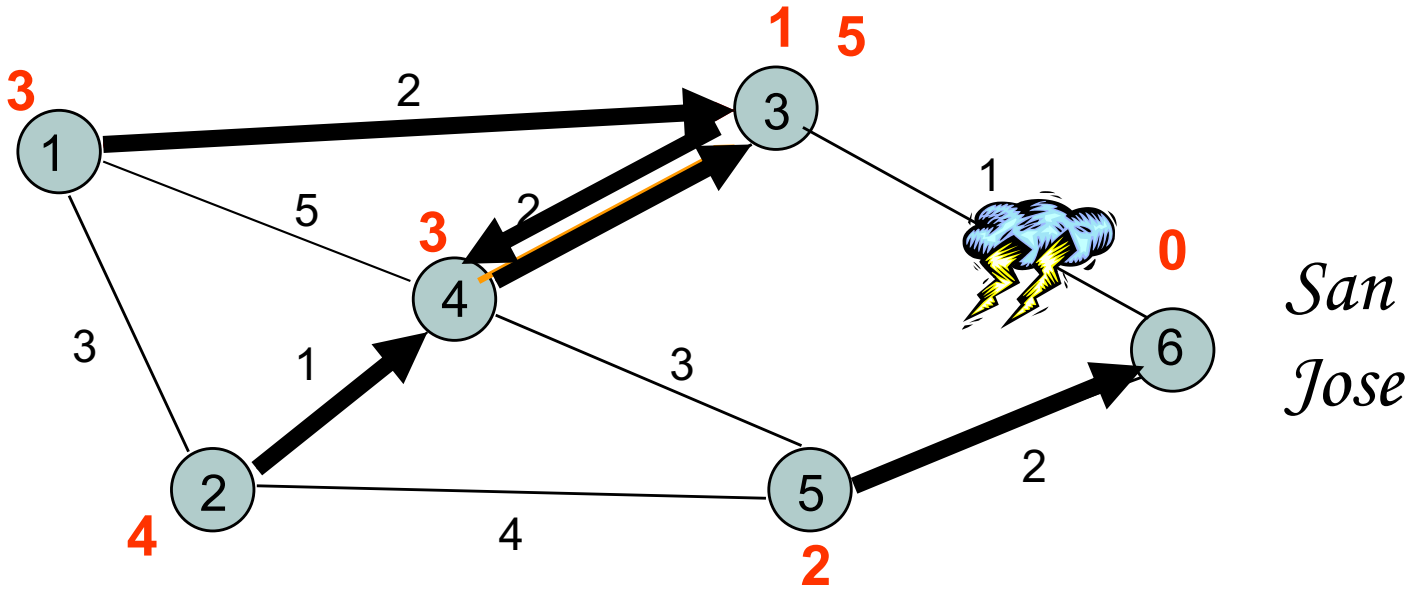
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3					



Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)	(-1, ∞)
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)

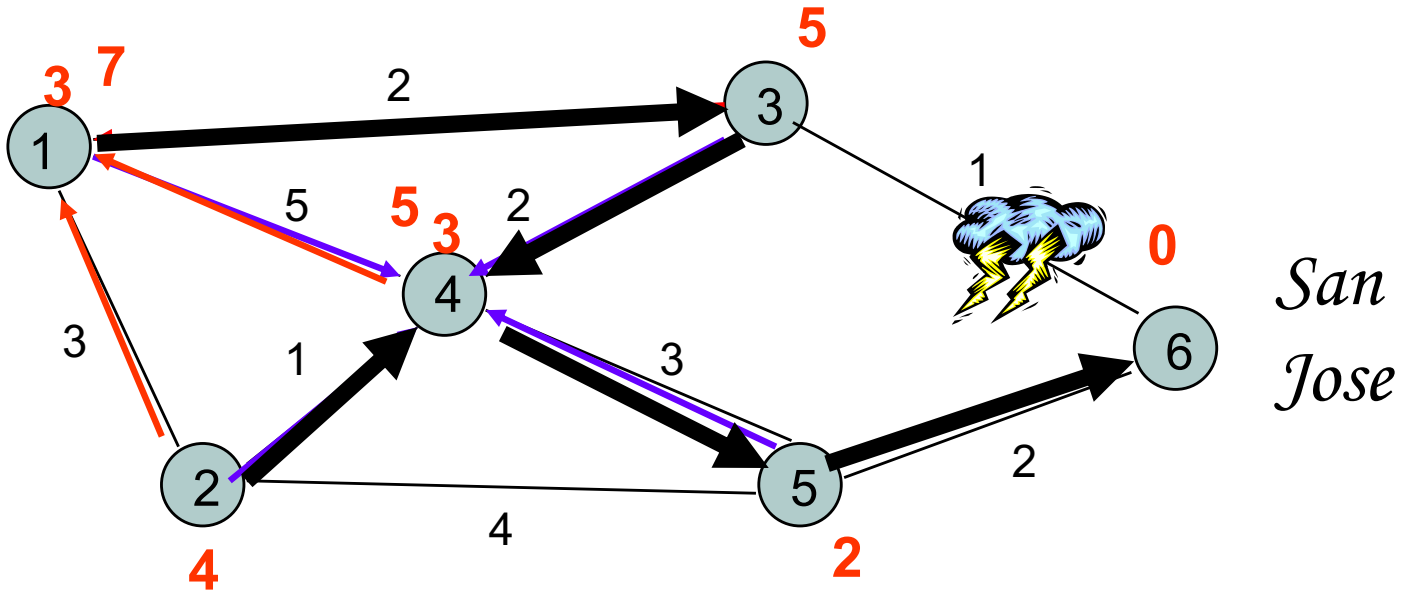


Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2					
3					



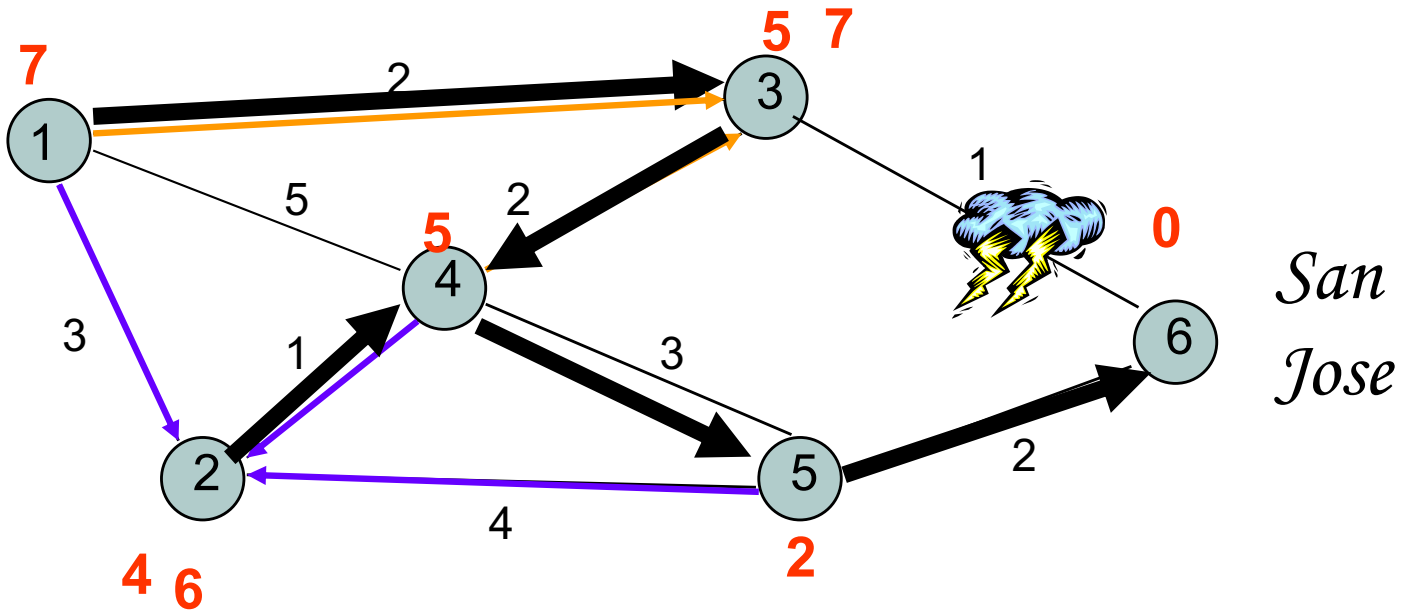
Network disconnected; Loop created between nodes 3 and 4

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(5,5)	(6,2)
3					



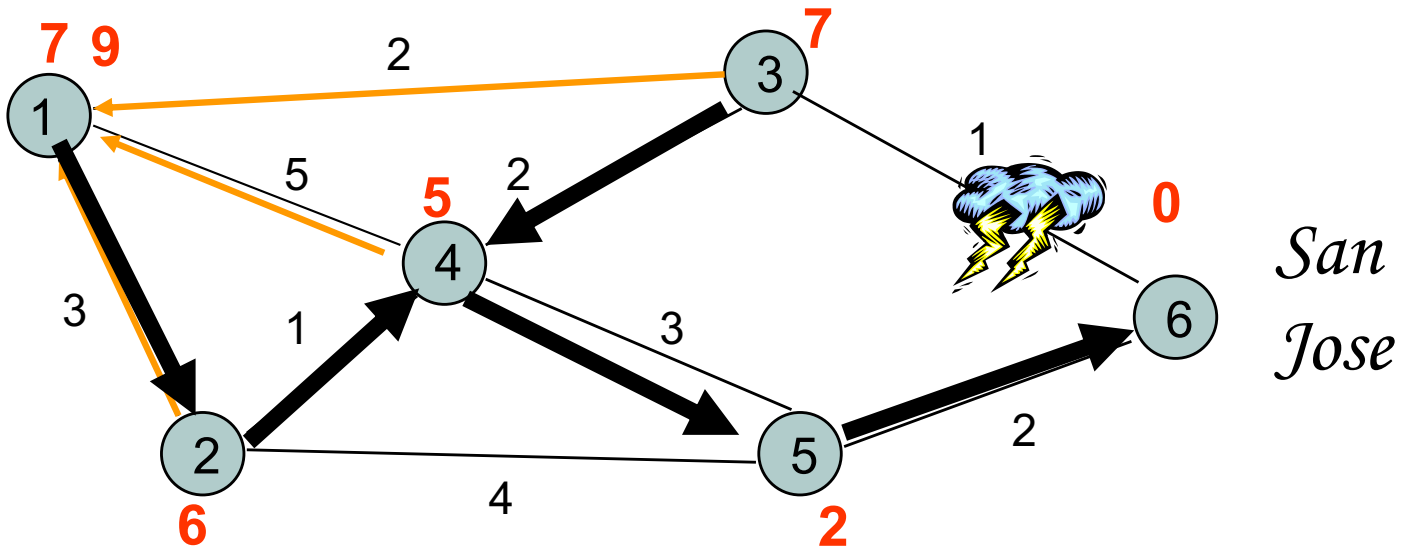
Node 4 could have chosen 2 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(5,5)	(6,2)
3	(3,7)	(4,6)	(4, 7)	(5,5)	(6,2)



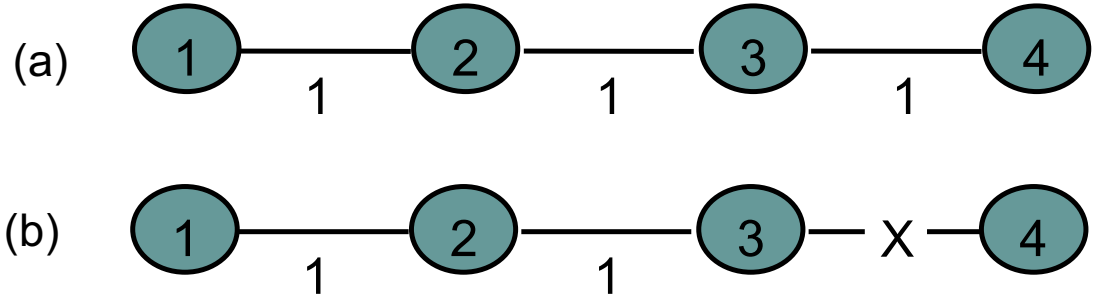
Node 2 could have chosen 5 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(2,5)	(6,2)
3	(3,7)	(4,6)	(4, 7)	(5,5)	(6,2)
4	(2,9)	(4,6)	(4, 7)	(5,5)	(6,2)



Node 1 could have chose 3 as next node because of tie

Counting to Infinity Problem

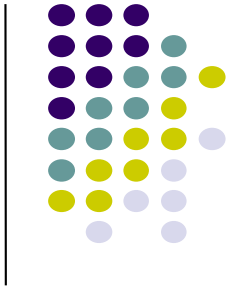


Nodes believe best path is through each other

(Destination is node 4)

Update	Node 1	Node 2	Node 3
Before break	(2,3)	(3,2)	(4, 1)
After break	(2,3)	(3,2)	(2,3)
1	(2,3)	(3,4)	(2,3)
2	(2,5)	(3,4)	(2,5)
3	(2,5)	(3,6)	(2,5)
4	(2,7)	(3,6)	(2,7)
5	(2,7)	(3,8)	(2,7)
…	…	…	…

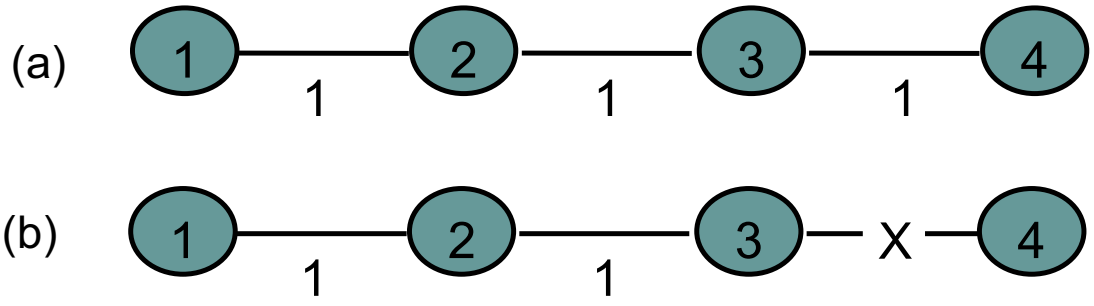
Problem: Bad News Travels Slowly



Remedies

- Split Horizon
  - Do not report route to a destination to the neighbor from which route was learned
- Poisoned Reverse
  - Report route to a destination to the neighbor from which route was learned, but with infinite distance
  - Breaks erroneous direct loops immediately
  - Does not work on some indirect loops

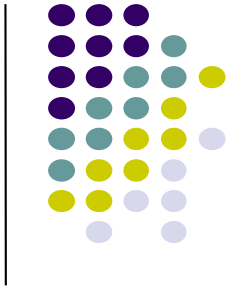
Split Horizon with Poison Reverse



Nodes believe best path is through each other

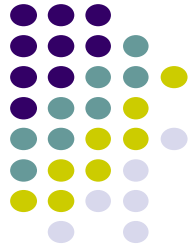
Update	Node 1	Node 2	Node 3	
Before break	(2, 3)	(3, 2)	(4, 1)	
After break	(2, 3)	(3, 2)	(-1, ∞)	Node 2 advertizes its route to 4 to node 3 as having distance infinity; node 3 finds there is no route to 4
1	(2, 3)	(-1, ∞)	(-1, ∞)	Node 1 advertizes its route to 4 to node 2 as having distance infinity; node 2 finds there is no route to 4
2	(-1, ∞)	(-1, ∞)	(-1, ∞)	Node 1 finds there is no route to 4

Link-State Algorithm



- Basic idea: two step procedure
  - Each source node gets a map of all nodes and link metrics (link state) of the entire network
  - Find the shortest path on the map from the source node to all destination nodes
- Broadcast of link-state information
  - Every node *i* in the network broadcasts to every other node in the network:
    - ID's of its neighbors: $\mathcal{N}_i$=set of neighbors of i
    - Distances to its neighbors: $\{C_{ij} \mid j \text{ in } N_i\}$
  - Flooding is a popular method of broadcasting packets

Dijkstra Algorithm: Finding shortest paths in order

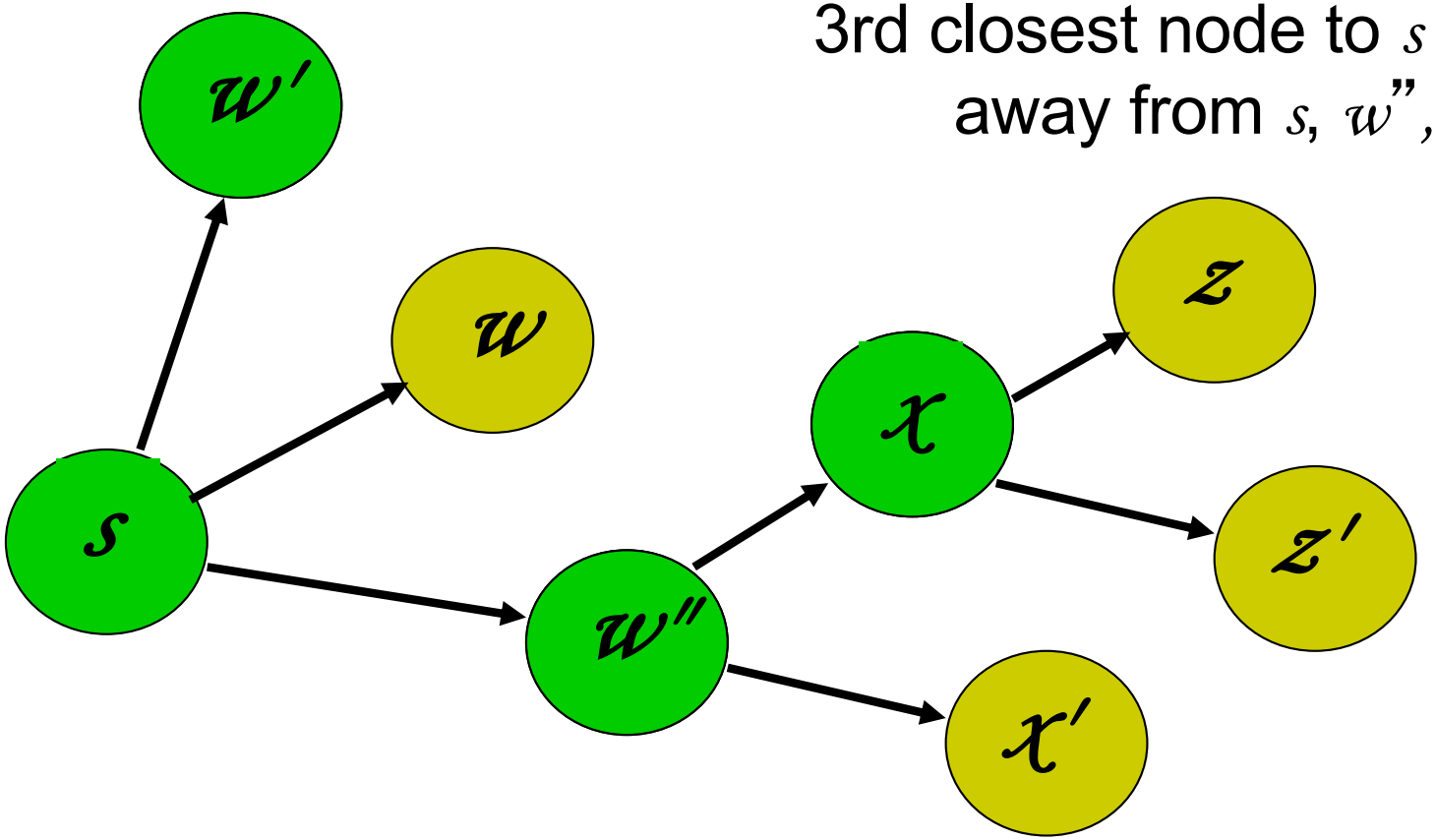


Find shortest paths from source s to all other destinations

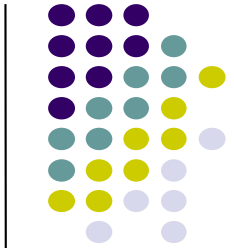
Closest node to s is 1 hop away

2nd closest node to s is 1 hop away from s or w"

3rd closest node to s is 1 hop away from s, w", or x



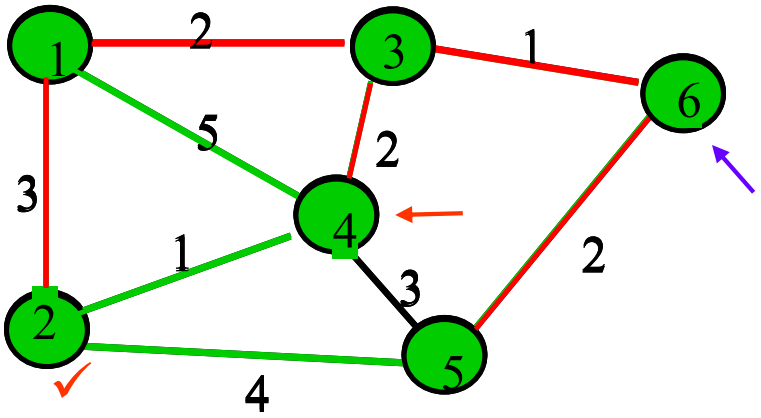
Dijkstra's algorithm

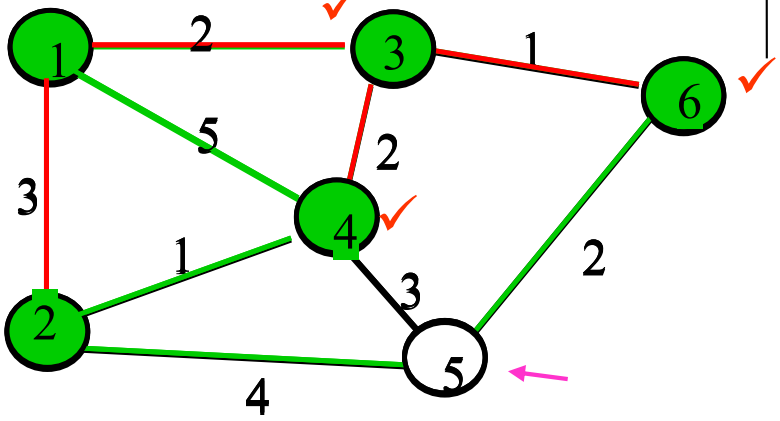


- *N*: set of nodes for which shortest path already found
- Initialization: (*Start with source node s)*
  - $N = \{s\}$, $D_s = 0$, "*s* is distance zero from itself"
  - $D_j = C_{sj}$ for all *j in s,* distances of directly-connected neighbors
- Step A: (*Find next closest node i*)
  - Find *i not in N* such that
  - D_i = min *Dj* for *j not in N*
  - Add *i* to *N*
  - If *N* contains all the nodes, stop
- Step B: (*update minimum costs)*
  - For each node *j not in N*
  - $D_j = \min (D_j, D_i + C_{ij})$
  - Go to Step A

Minimum distance from s to *j through node* ***i*** *in N*

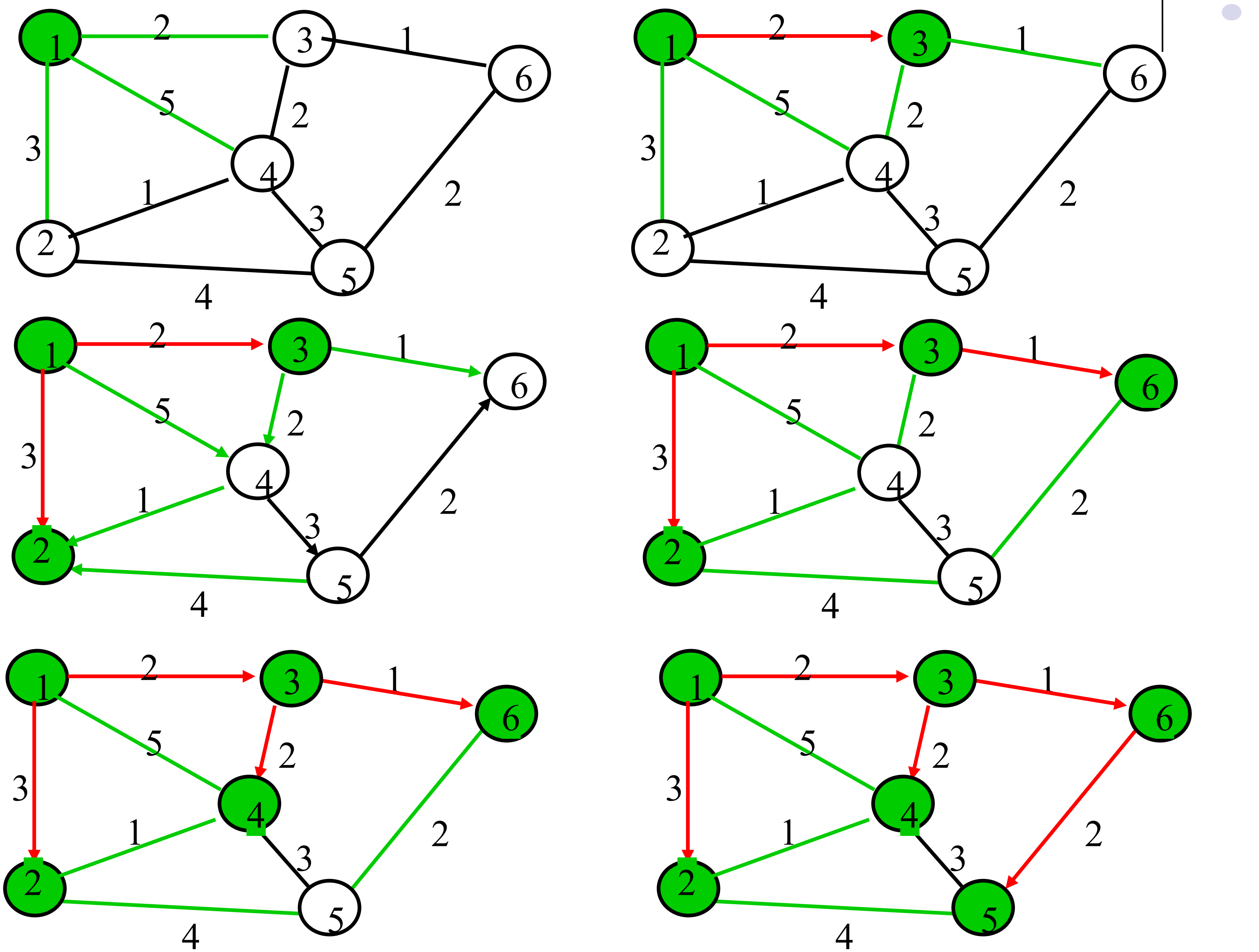
Execution of Dijkstra's algorithm



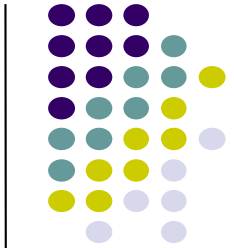


Iteration	**N**	D_2	D_3	D_4	D_5	D_6
Initial	{1}	3	2 ✓	5	inf	inf
1	{1,3}	3 ✓	2	4	inf	3
2	{1,2,3}	3	2	4	7	3 ✓
3	{1,2,3,6}	3	2	4 ✓	5	3
4	{1,2,3,4,6}	3	2	4	5 ✓	3
5	{1,2,3,4,5,6}	3	2	4	5	3

Shortest Paths in Dijkstra's Algorithm

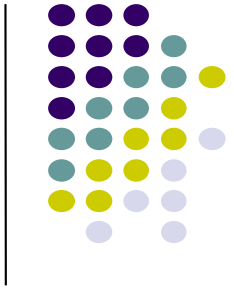


Reaction to Failure



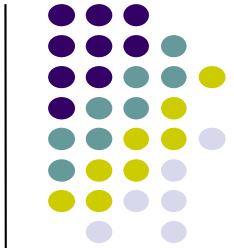
- If a link fails,
  - Router sets link distance to infinity & floods the network with an update packet
  - All routers immediately update their link database & recalculate their shortest paths
  - Recovery very quick
- But watch out for old update messages
  - Add time stamp or sequence # to each update message
  - Check whether each received update message is new
  - If new, add it to database and broadcast
  - If older, send update message on arriving link

Why is Link State Better?



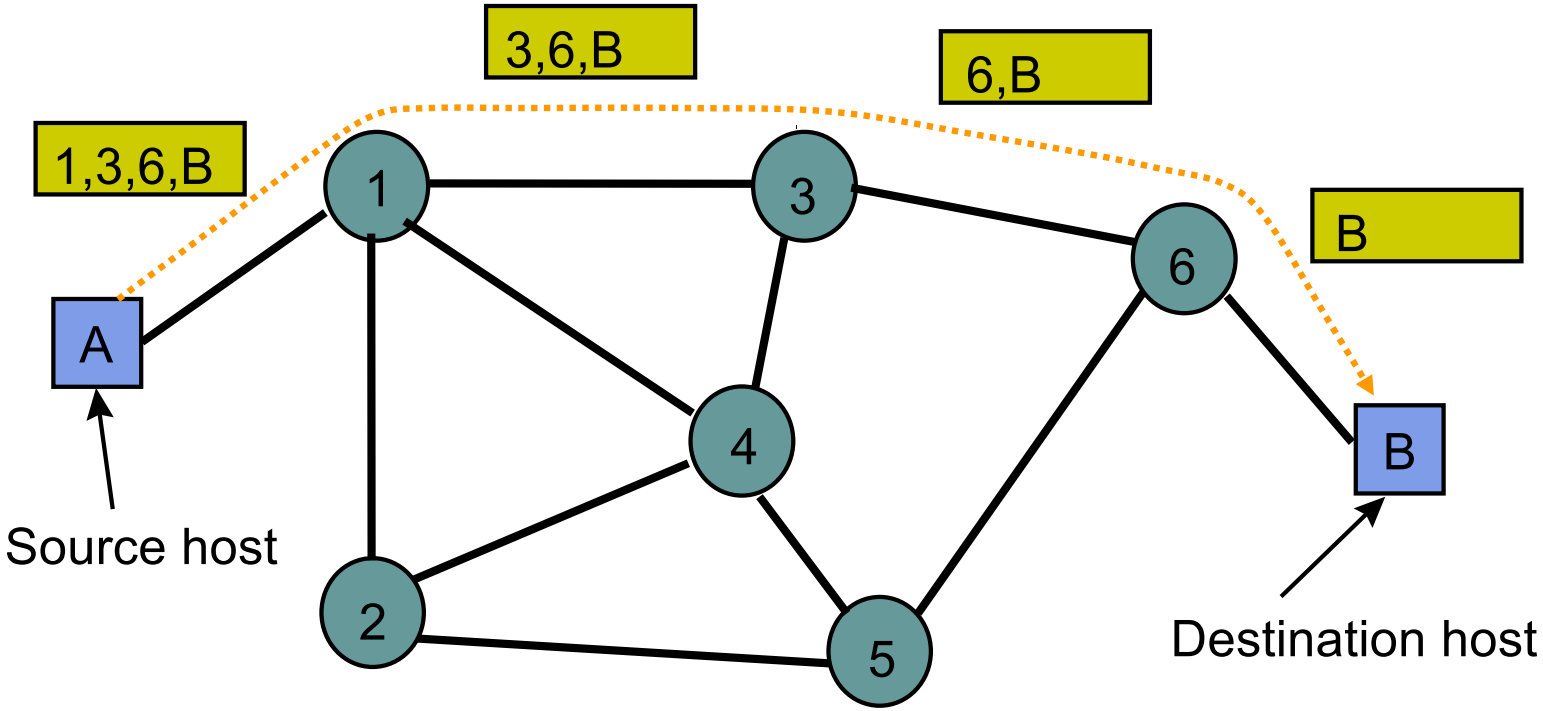
- Fast, loopless convergence
- Support for precise metrics, and multiple metrics if necessary (throughput, delay, cost, reliability)
- Support for multiple paths to a destination
  - algorithm can be modified to find best two paths

Source Routing

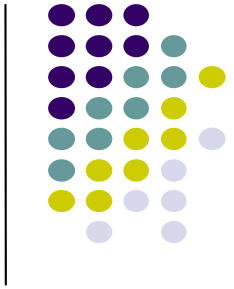


- Source host selects path that is to be followed by a packet
  - Strict: sequence of nodes in path inserted into header
  - Loose: subsequence of nodes in path specified
- Intermediate switches read next-hop address and remove address
- Source host needs link state information or access to a route server
- Source routing allows the host to control the paths that its information traverses in the network
- Potentially the means for customers to select what service providers they use

Example



Segment Routing



- Flexible, scalable way of doing source routing.
- Source selects a path and encodes it in packet header
  - Ordered list of segments.
- Segments are identifier for any type of instruction.
- Each segment identified by segment ID
  - A flat unsigned 32-bit integer
- Segment instruction example:
  - Go to node N using the shortest path; Apply service S
- Works with MPLS
- Enables SDN