

ETSF05:

Användarmodeller/Paradigmer

ARQ

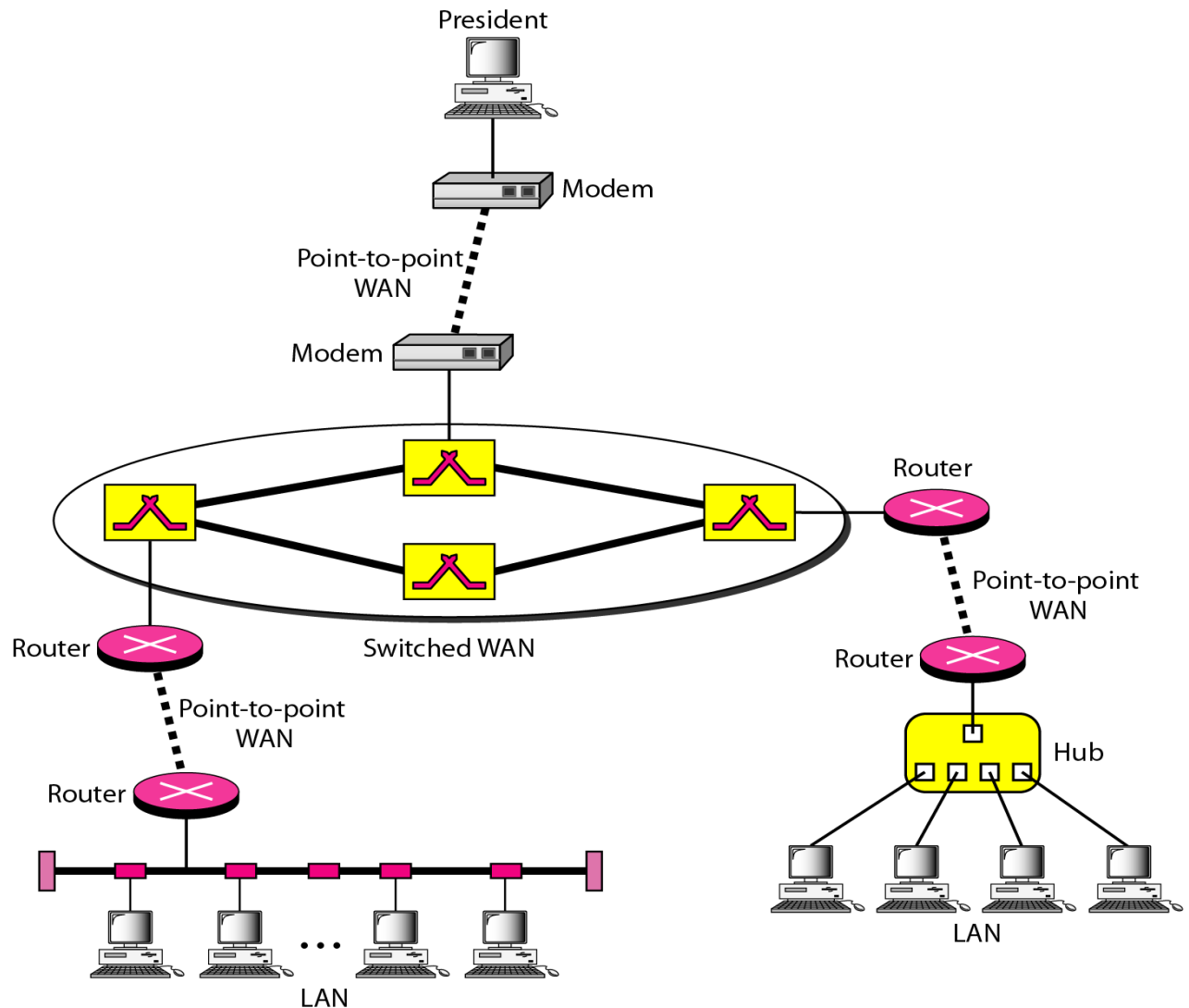
Routingalgoritmer



Network engineering

High performance

- ◆ Reliability
- ◆ Throughput
- ◆ Speed
- ◆ Security



Network models

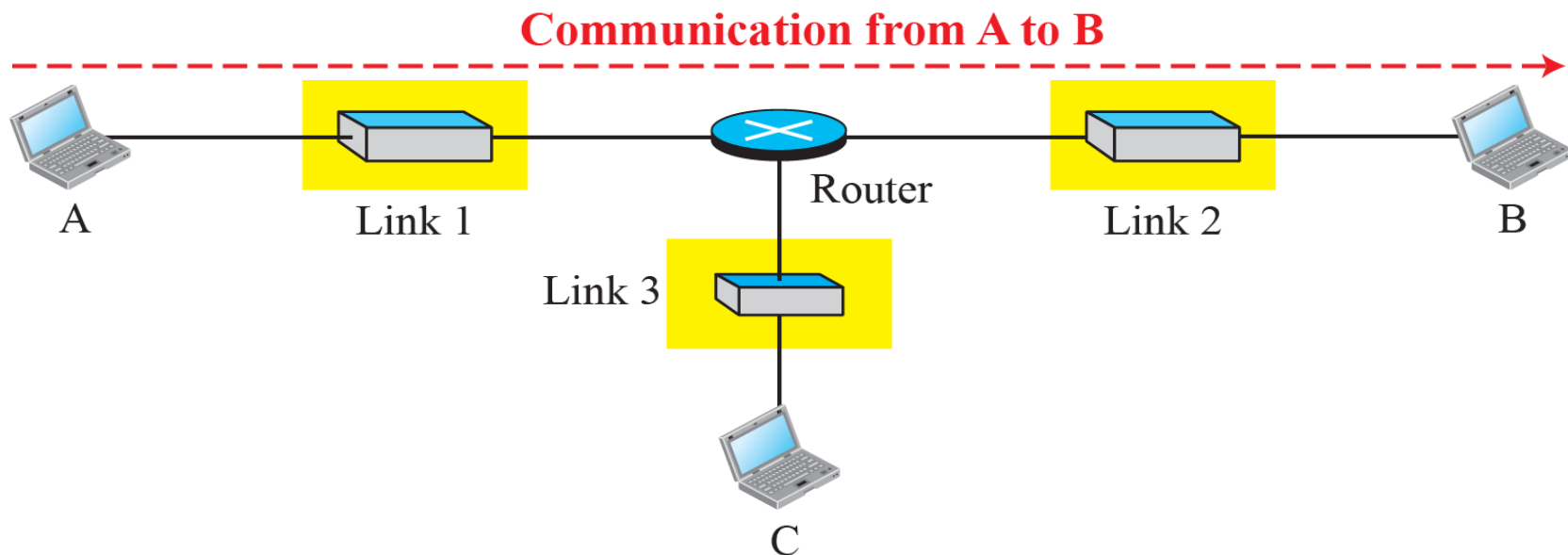
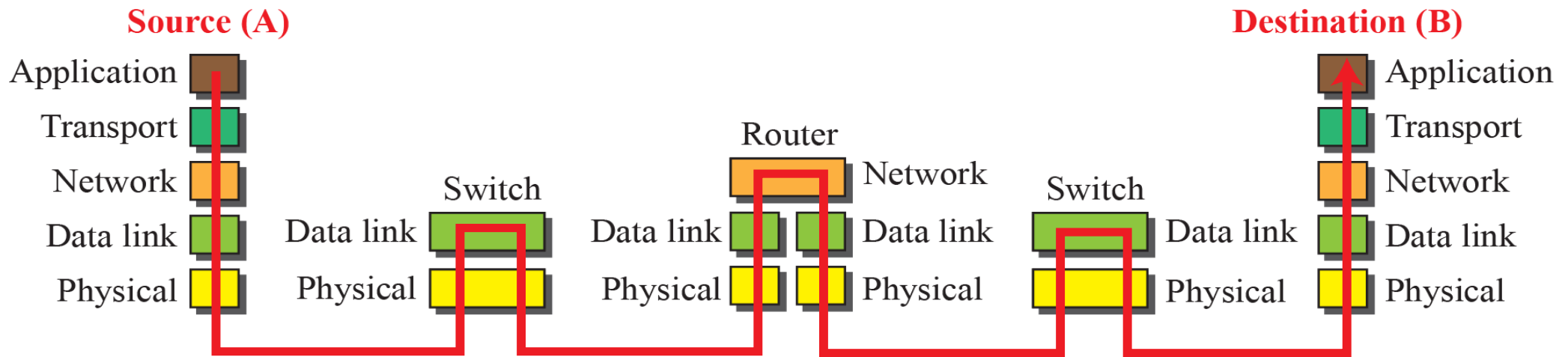
Too complicated

- ◆ Divide and conquer

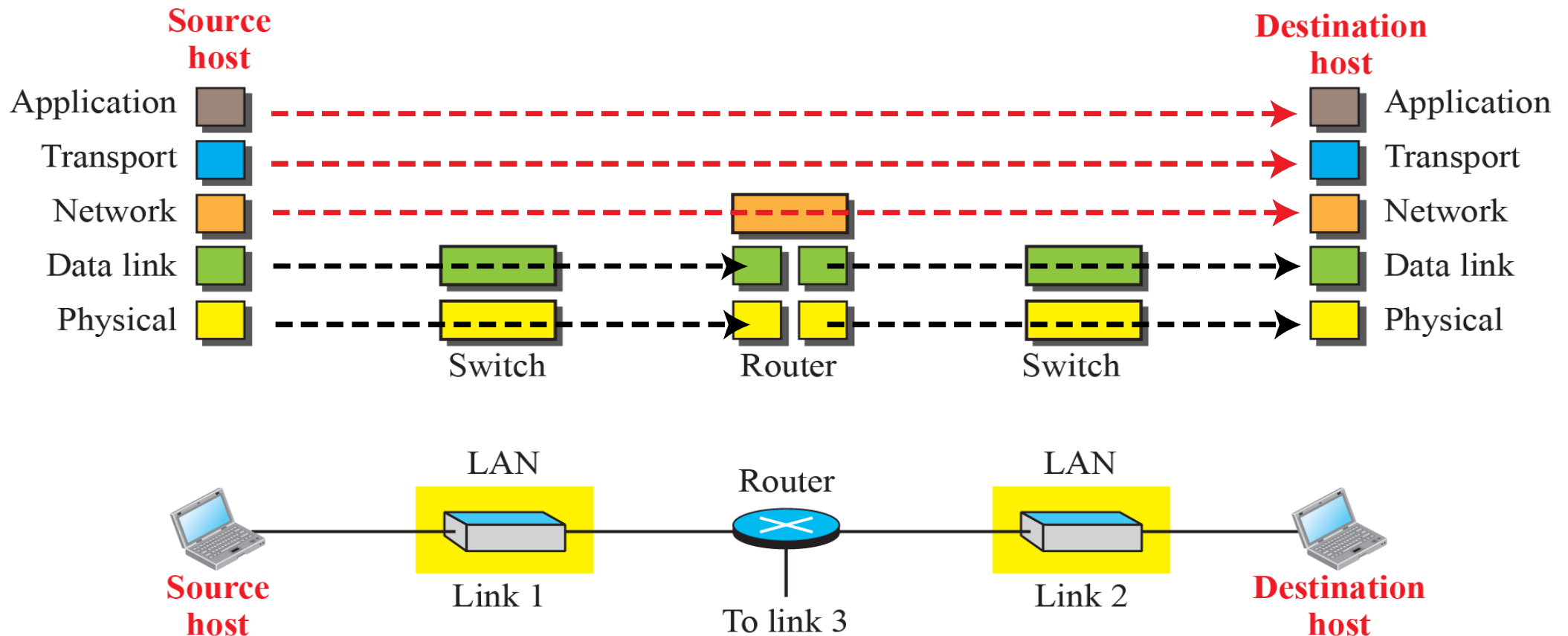
Layered architecture

- ◆ Hierarchy
- ◆ Specialisation
- ◆ Simplification

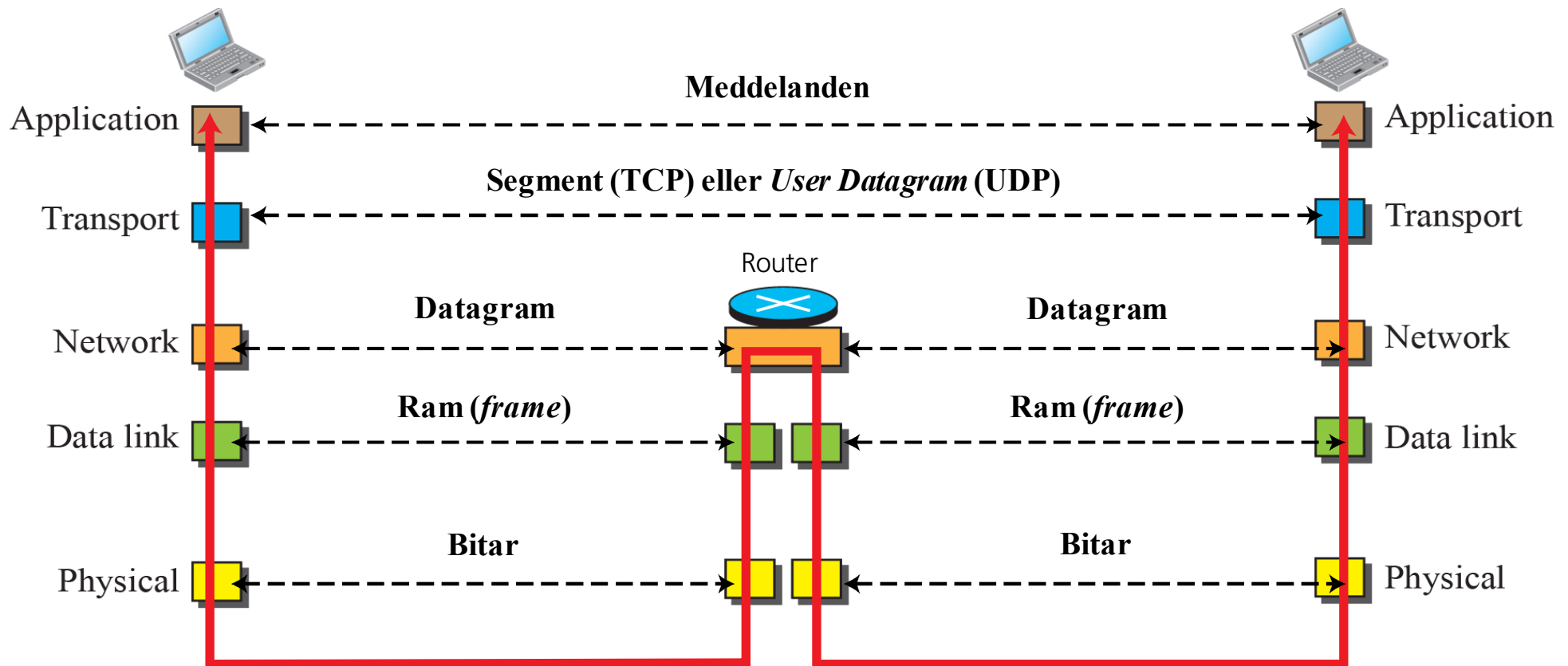
Kommunikation på Internet



Logiska förbindelser enligt TCP/IP



Identiska objekt enligt TCP/IP

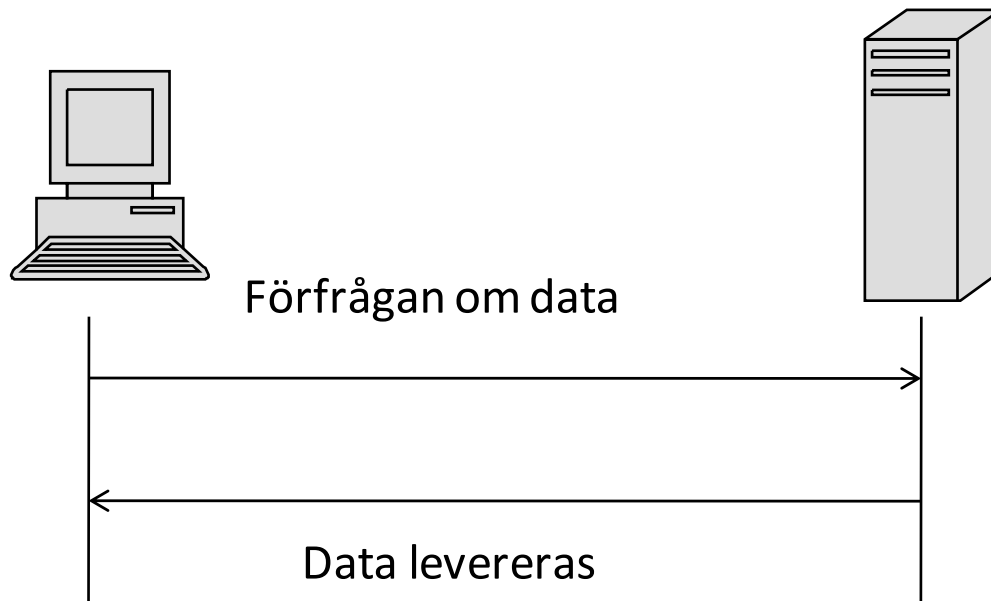


Switchar/Repeterare påverkar inte objekt!

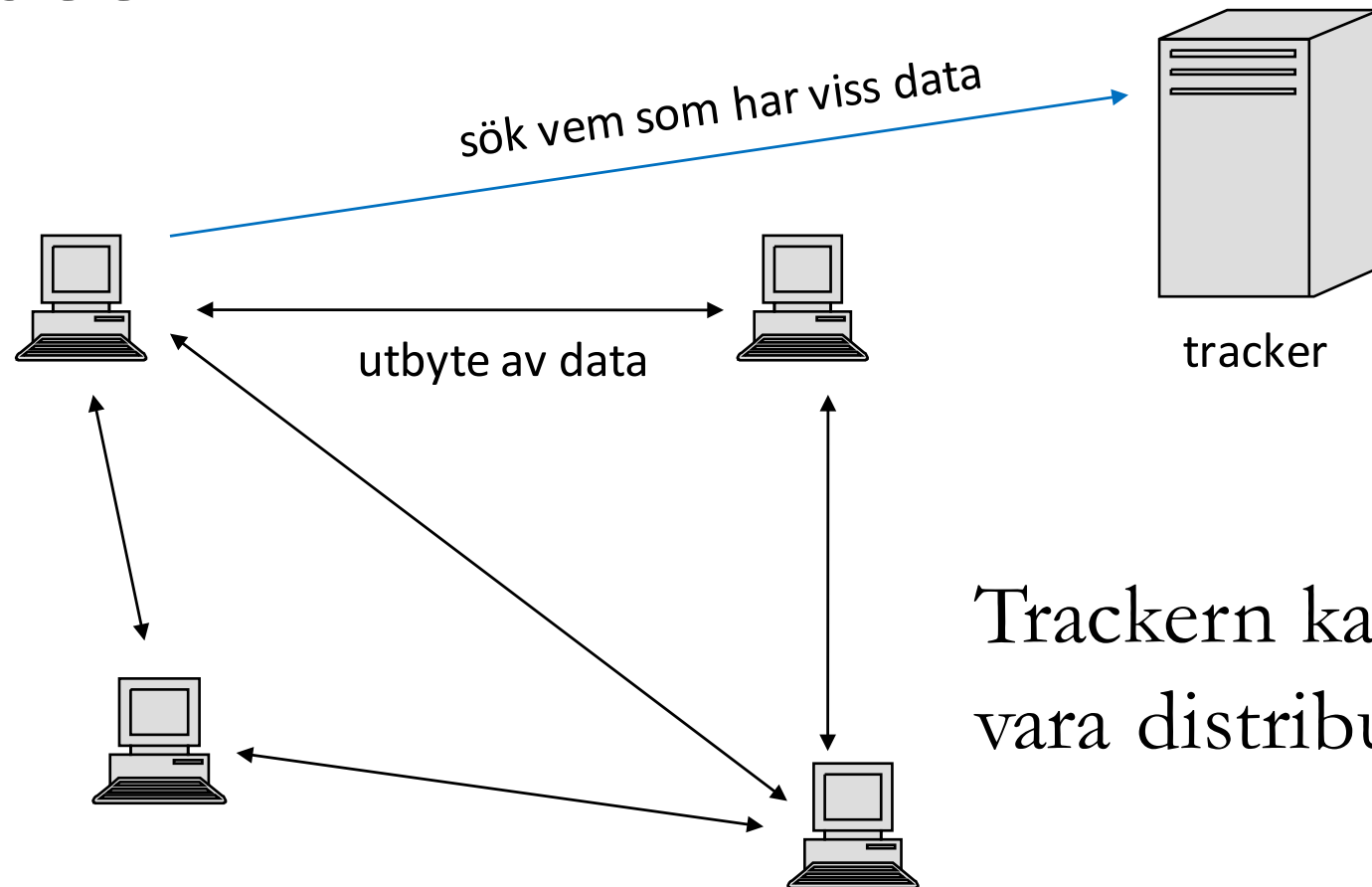
Användarmodeller

- Klient-server
 - *Client-Server Paradigm*
- Användare-användare
 - *Peer-to-peer (P2P) Paradigm*

Client-Server paradigm



Peer-to-peer



Trackern kan
vara distribuerad

Kontrollprotokoll (på länk-nivån)

Flödeskontroll (ARQ)

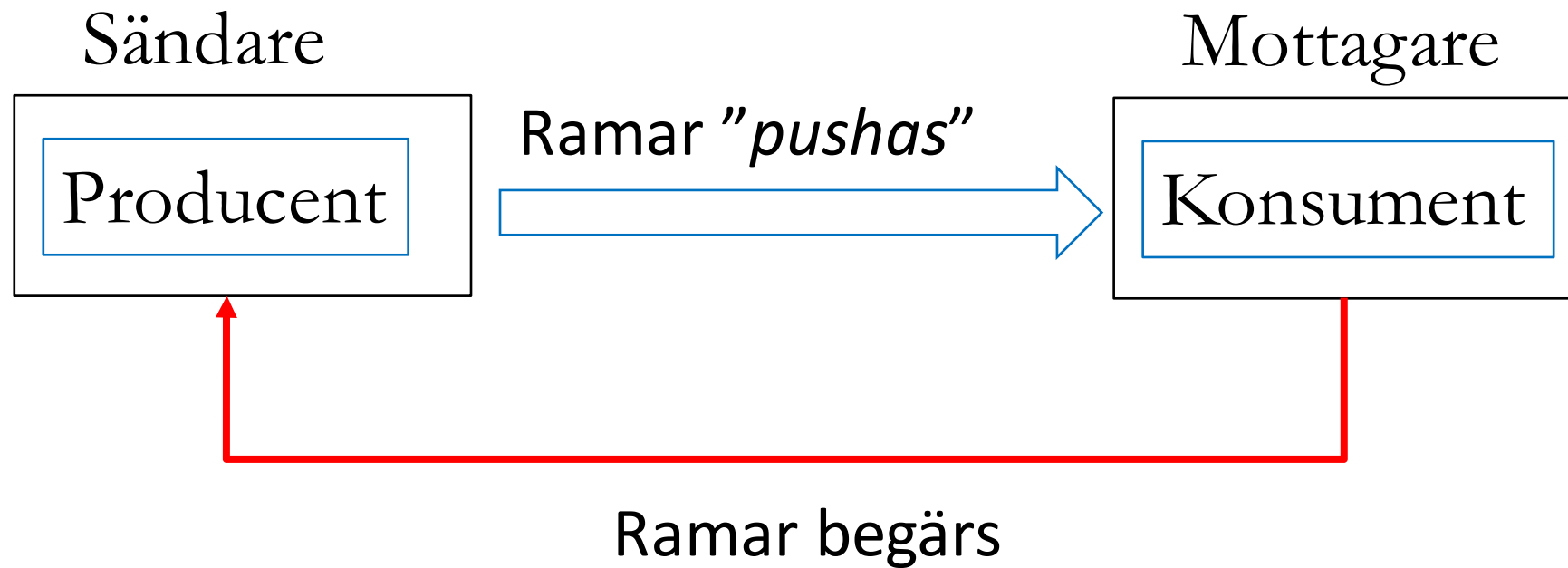
- ◆ Stop-and-wait
- ◆ Go-back-N
- ◆ Selective Repeat

Felkontroll

- ◆ Upptäcka fel
- ◆ Omsändning

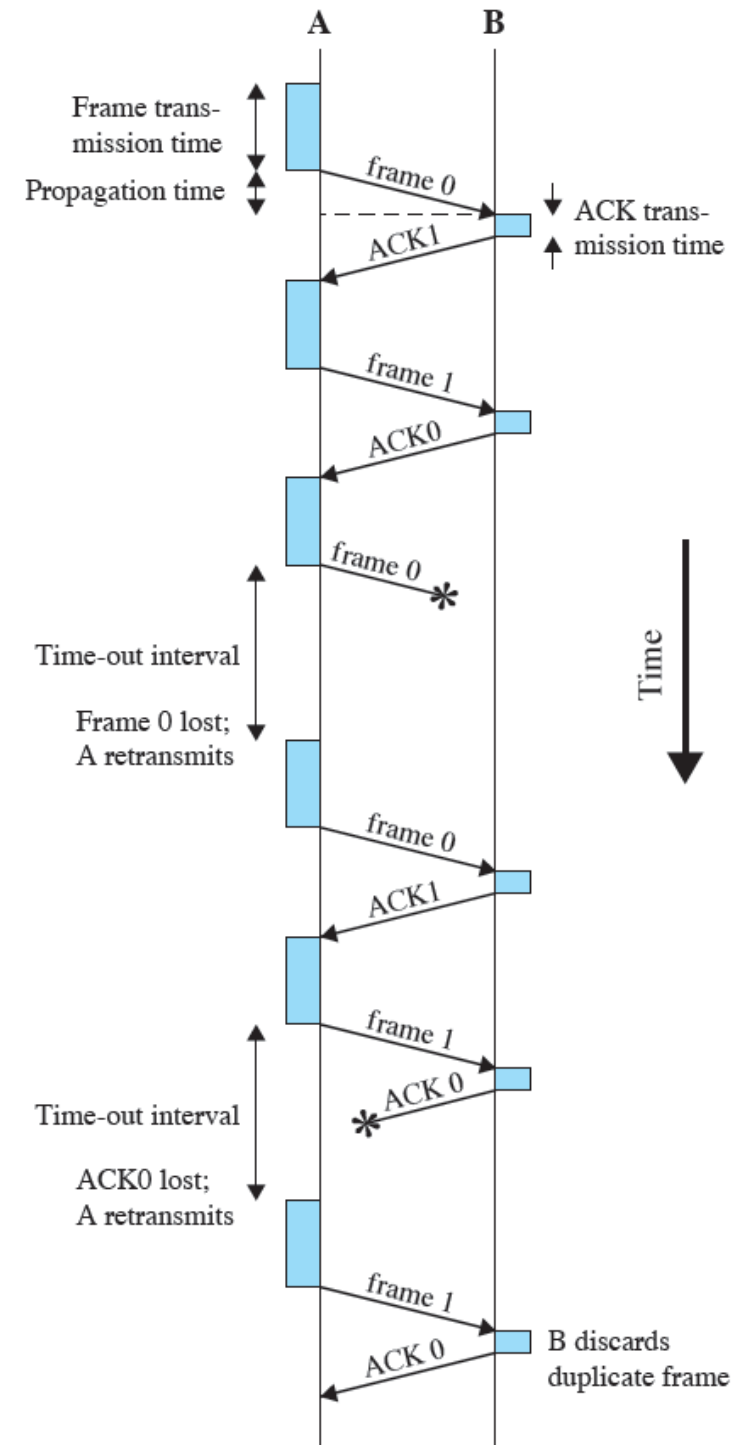
Inte alltid att flödeskontroll och felhantering finns i länklagret! Finns även på högre upp i stacken (e.g. TCP).

Flödeskontroll



Stop-and-wait ARQ flödesdiagram

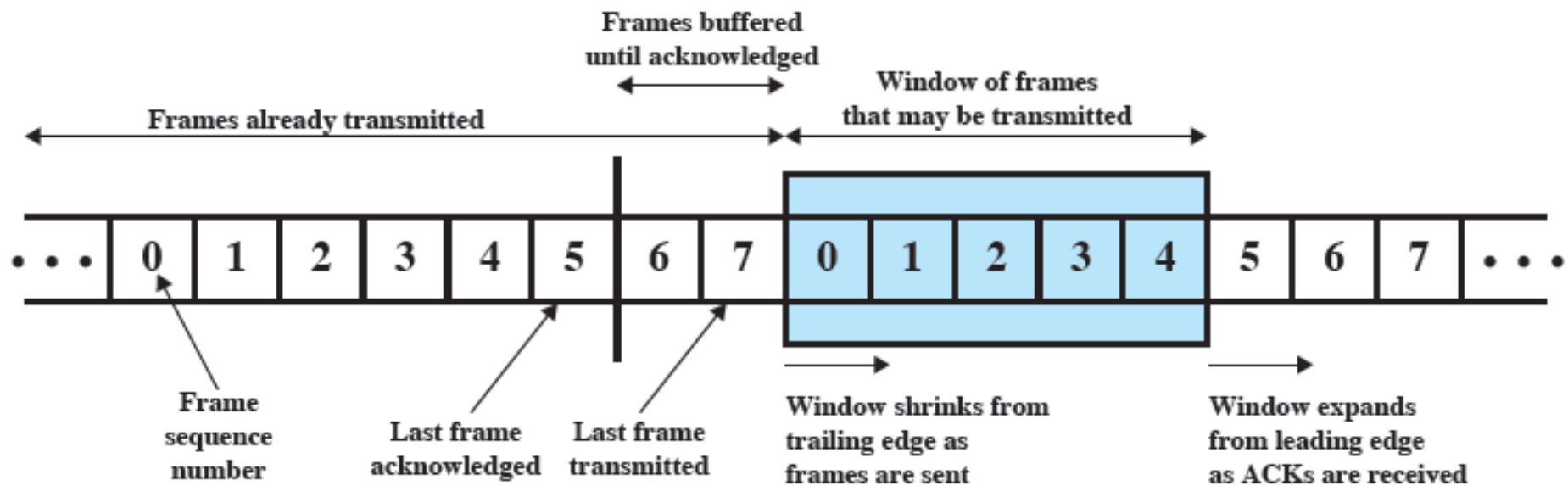
- Ineffektivt
- Mycket väntan i onödan
Speciellt vid långa länkar



Go-Back-N

- Viktigt inför nästa del och TCP (lp2)
- Föreläst i repetitionsföreläsningen
- Kommer på övning

Sliding window (sender)



(a) Sender's perspective

Go-Back-N ARQ (the Stalling Way)

Most commonly used error control **ACK (this or next)**

Based on sliding-window

Use window size to control number of outstanding frames

While no errors occur, the destination will acknowledge incoming frames as usual

- ◆ **RR=receive ready, or piggybacked acknowledgment**

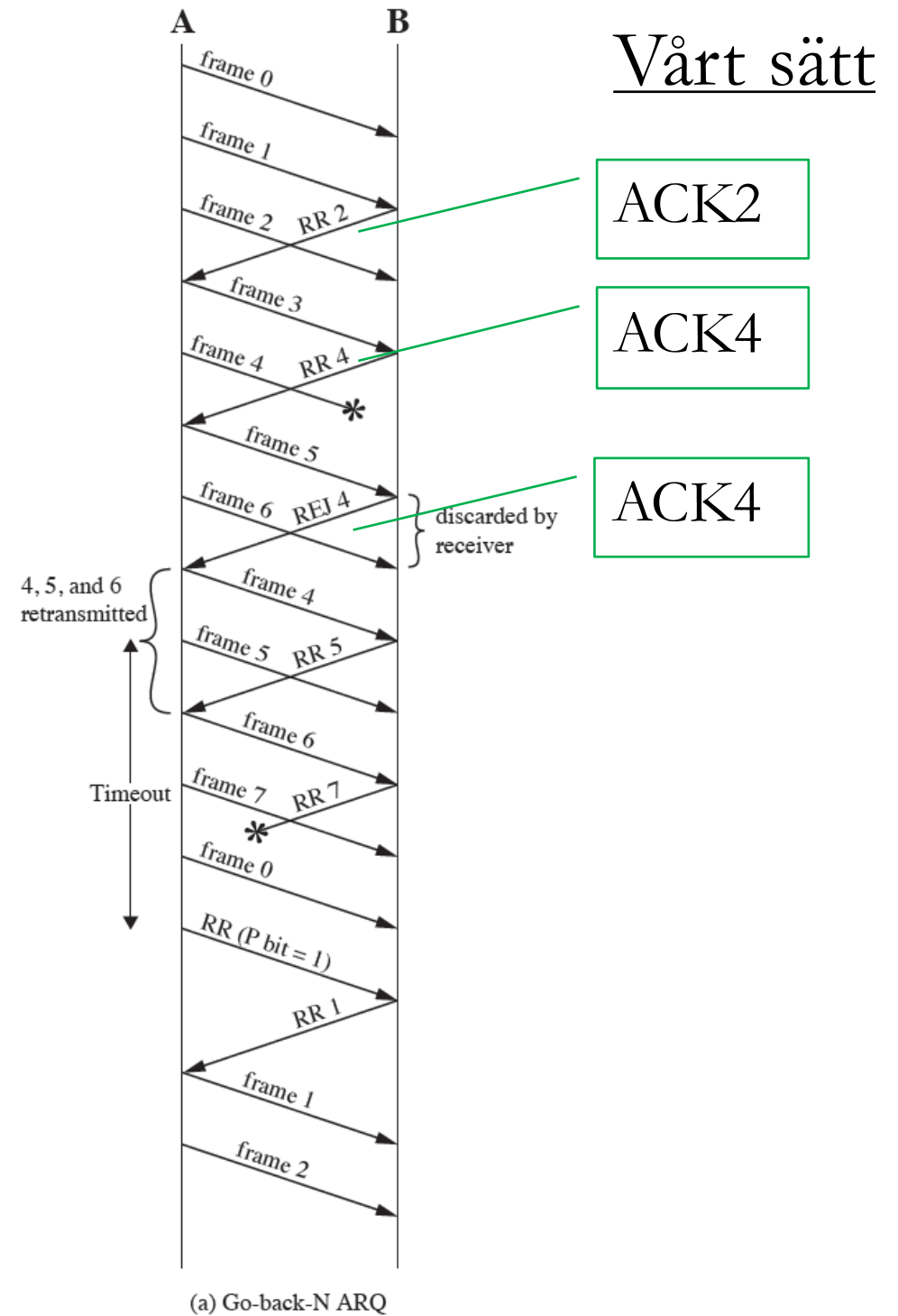
If the destination station detects an error in a frame, it may send a negative acknowledgment

- ◆ **REJ=reject**

ACK (previous or this)

- ◆ Destination will discard that frame and all future frames until the frame in error is received correctly
- ◆ Transmitter must go back and retransmit that frame and all subsequent frames

Go-Back-N



Selective-Reject (ARQ) (Stalling)

Also called selective retransmission

Only rejected/missing frames are retransmitted

Subsequent frames are accepted by the receiver and buffered

Minimizes retransmission

Receiver must maintain large enough buffer

More complex logic in transmitter

- ◆ Less widely used

Useful for satellite links with long propagation delays

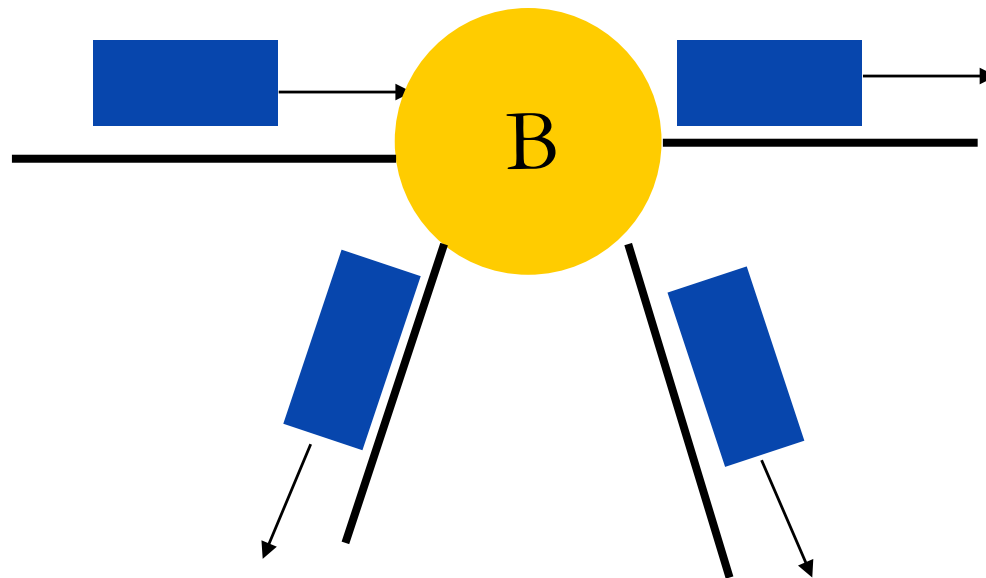
Routingalgoritmer

- Konsten att bygga least-cost trees
 - Från sändare till mottagare
 - Från varje nod till varje annan nod
- Tre principer
 - Distance Vector
 - Link State
 - Path Vector
 - Policy-based routing

Flooding

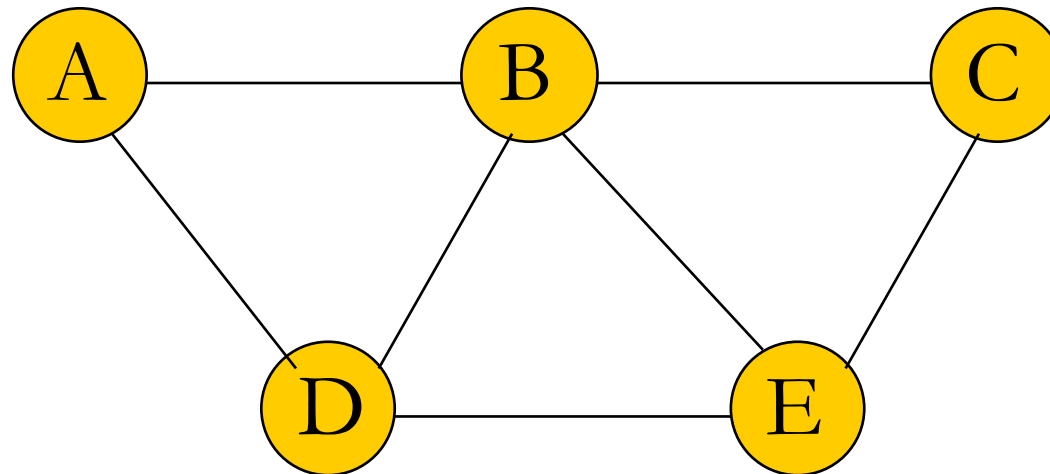
I Flooding skickas ett inkommande paket ut på samtliga länkar. En hop-count används för att inte skapa loopar.

Hur kan man istället hitta bästa vägen?



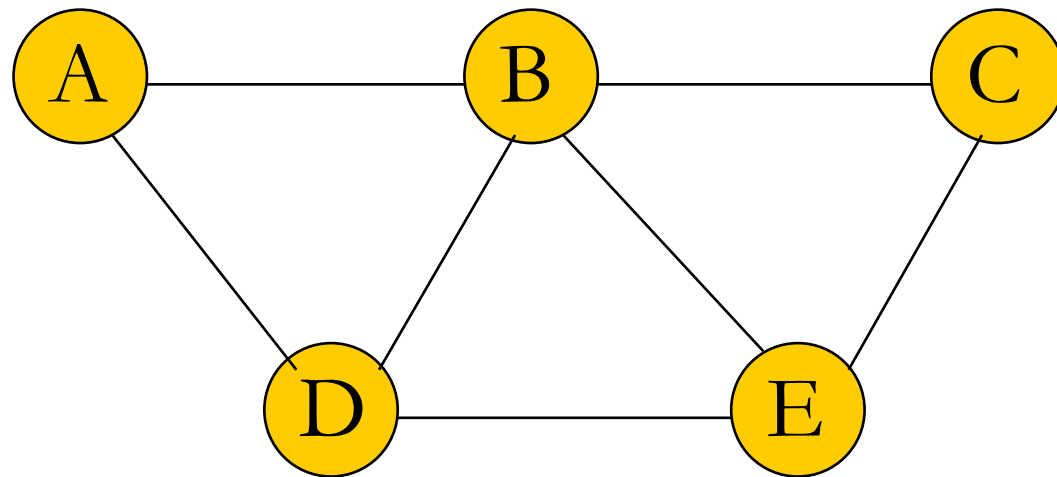
Nätgraf

I vägvalsalgoritmerna används en nätgraf som består av noder och länkar.



Least-hop path

Hitta väg mellan noder som innehåller minimalt antal steg.



Länkkostnad

Varje länk i grafen har en kostnad som anger hur ”dyrt” det är att skicka ett paket över länken.

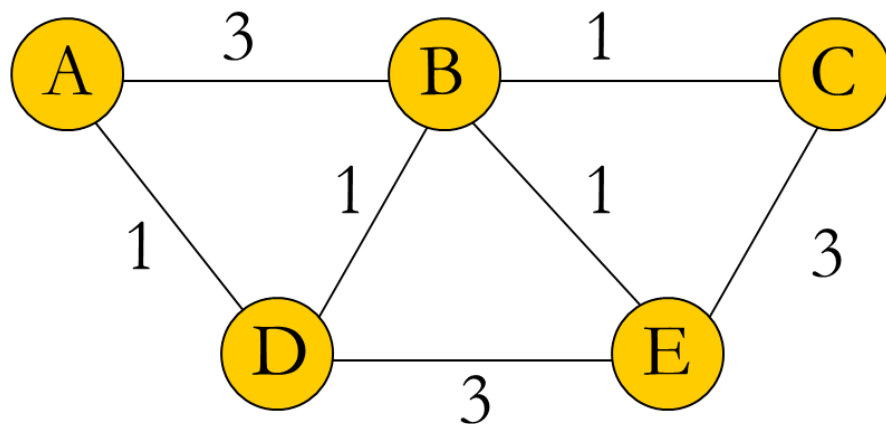
Länkkostnaden kan bero på flera saker:

- Kapacitet
- Belastning
- Sträcka
- Utbredningsmedium
- Osv...

Graf

En graf består av noder (N) och bågar (E) med vikter $w(e)$.

Exempel (undirected graph)

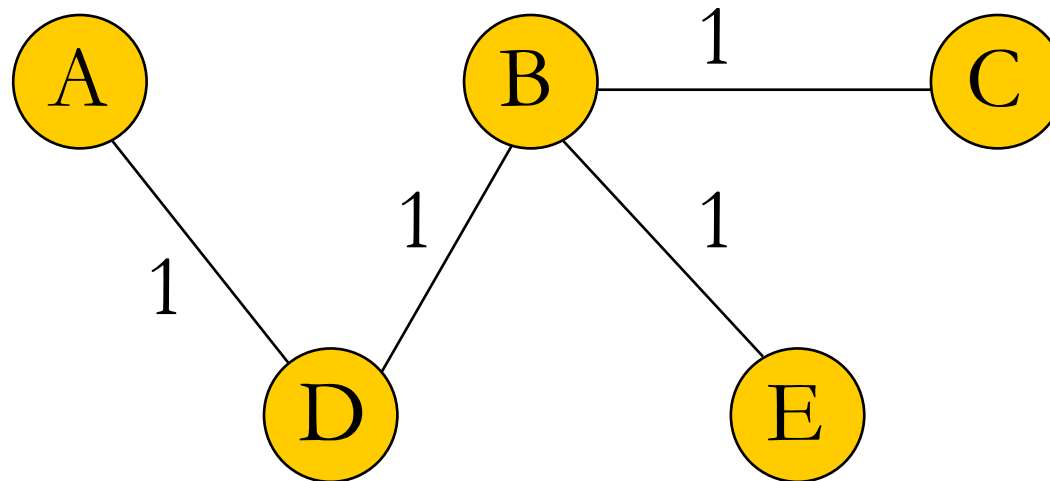


$$N = \{A, B, C, D, E\}$$

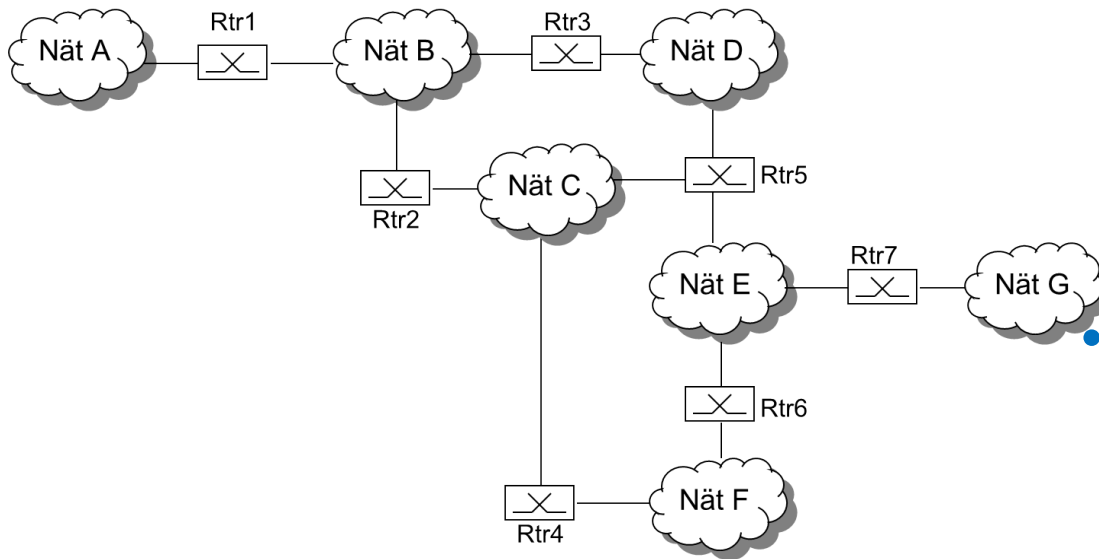
E	$w(e)$
AB	3
AD	1
BC	1
BD	1
BE	1
CE	3
DE	3

Least-cost path

- För nätverksgrafer är vikterna positiva och additiva.
- I Least-cost path väljs vägar med minimal total kostnad.



Ett Internet och dess grafiska representation



- I routing finns två olika typer av noder:
 - Nät dvs destinationer
 - Routrar som sammanbinder nät
- Grafiska representationen kan bli något konstig ...

Routingtabell

- Idén med routing är att varje nod tar beslut om vart inkommande ramar ska skickas vidare
- Vägvalen är definierade i routingtabellen
- Problemen är
 - Hur ska tabellen fyllas i?
 - Vad som händer när något ändras i nätet?

Distance vector routing

Alla kända bästa vägar **delas med grannar**

- ◆ Periodiskt
- ◆ (Vid varje förändring)

Routing-tabeller **uppdateras** med

- ◆ Nya poster
- ◆ Ändring av kostnad

”Global kunskap sprids lokalt”

Least cost alg 1

Bellman-Ford

Hitta kortaste vägen från en source node s till de övriga.

Låt $d(n)$ vara kostnaden från s till n

Init:

$$d(s) = 0$$

$$d(n) = \infty, n \neq s$$

for $i = 1$ to $|N| - 1$

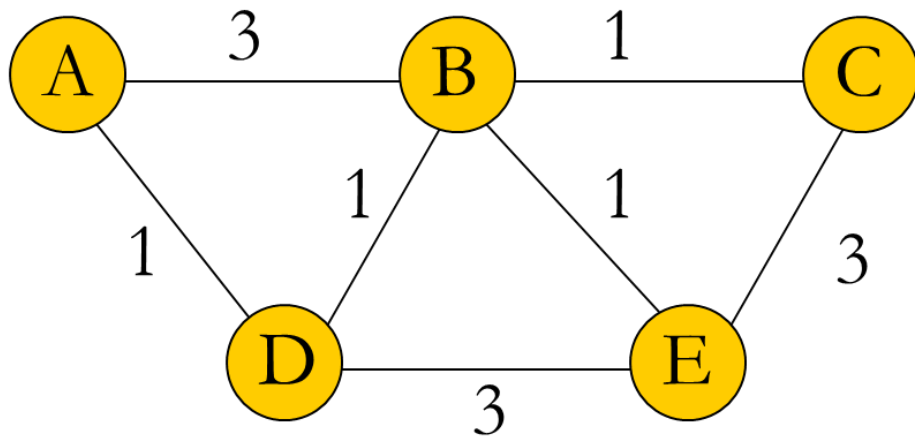
for each $n \in N$

$$d(n) = \min_{u \in N} \{d(u) + w(u, n)\} \quad // \text{ Hitta kortaste vägen från}$$

// nod u till nod n i ett steg

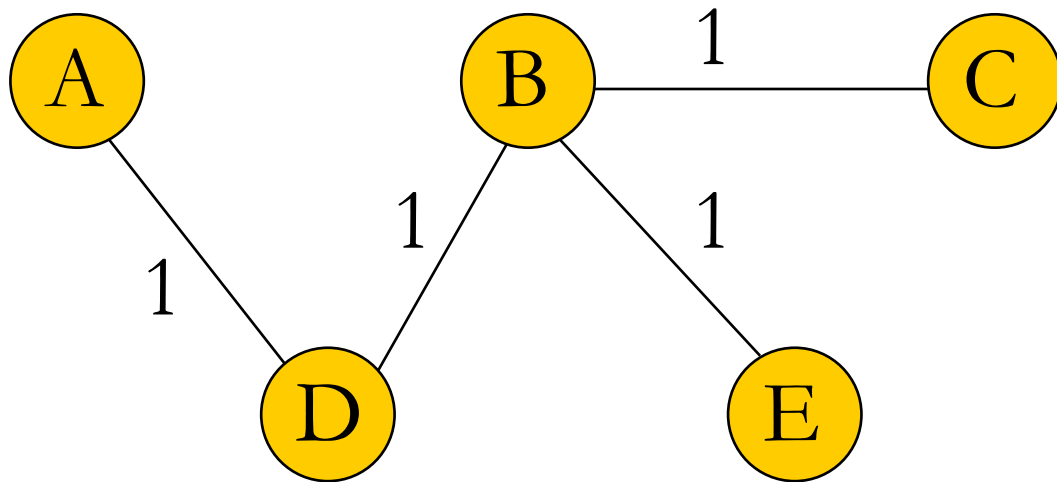
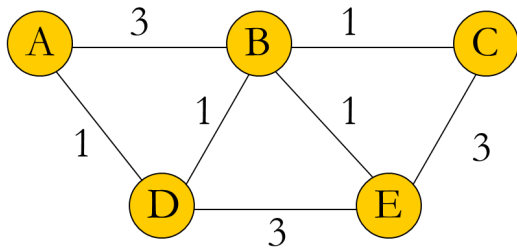
Tillägg: Håll ordning på vägen!!

Exempel Bellman-Ford



Nod	A	B	C	D	E
init	0	∞	∞	∞	∞
i=1	0	3	∞	1	∞
i=2	0	2	4	1	4
i=3	0	2	3	1	3
i=4	0	2	3	1	3

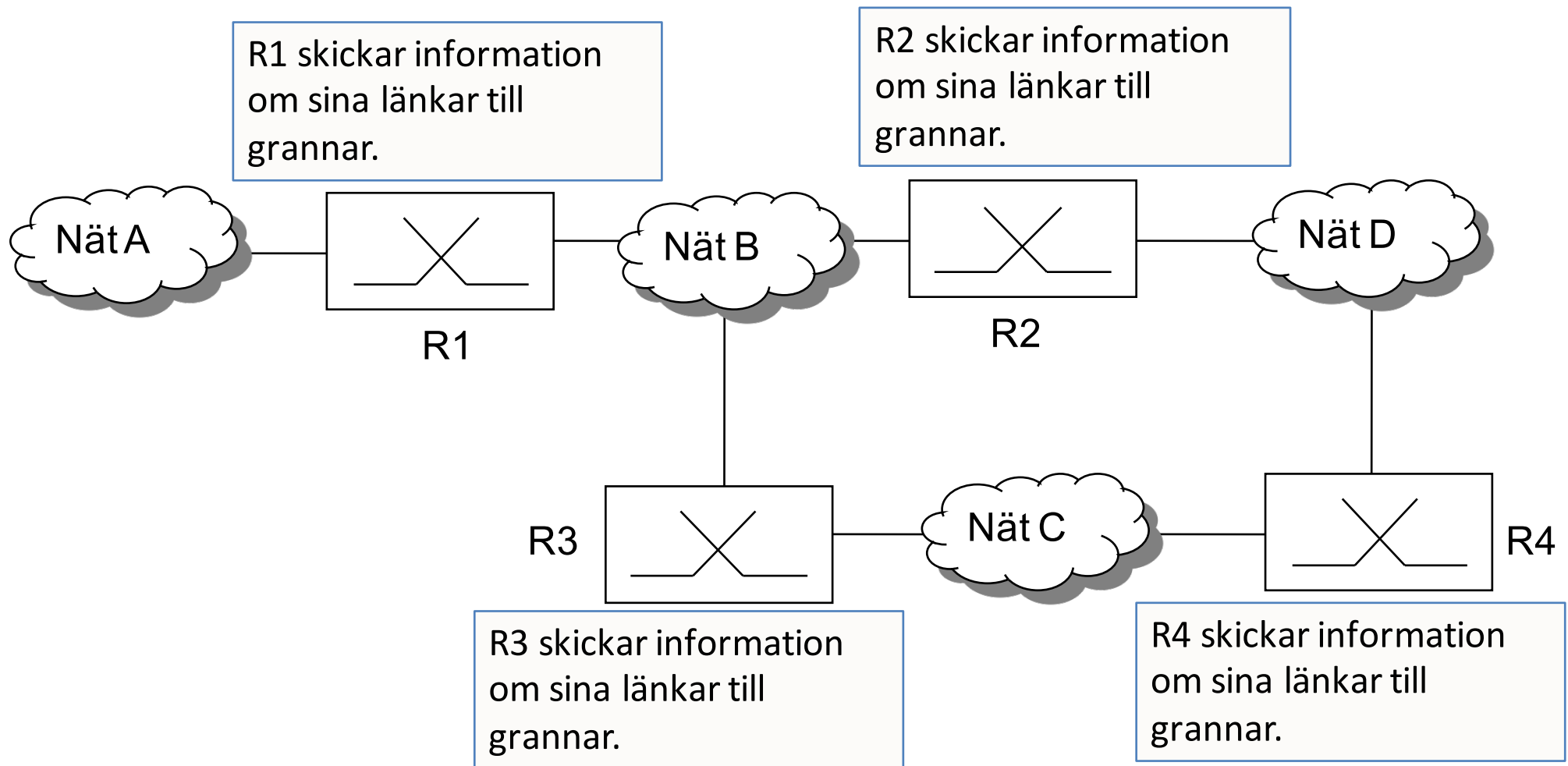
Nätgraf som träd



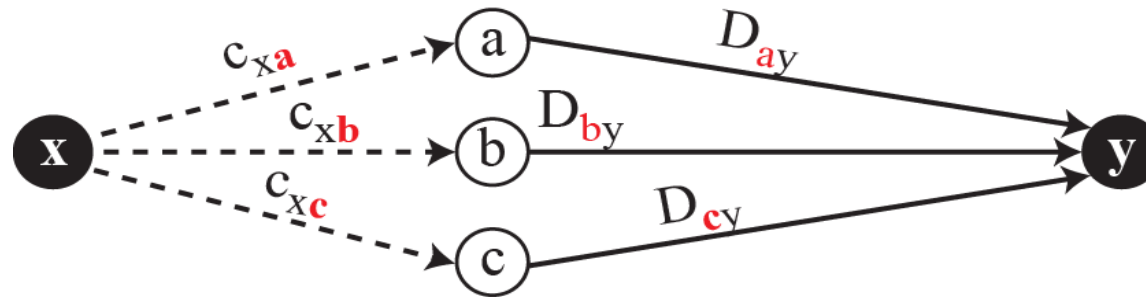
Distansvektor för A när
nätet konvergerat

Nod	Dist
A	0
B	2
C	3
D	1
E	3

Principen för *Distance Vector*

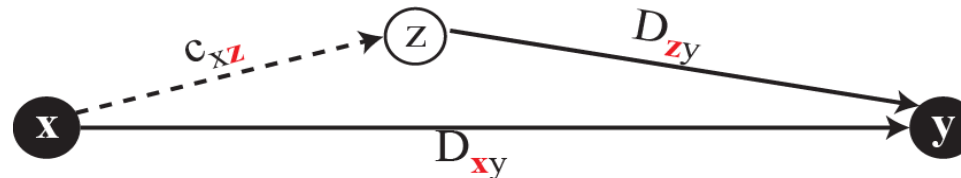


Bellman-Fords algoritm grafiskt



a. General case with three intermediate nodes

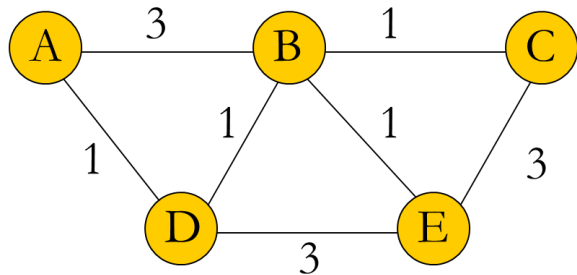
$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}) \dots\}$$



b. Updating a path with a new route

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

Uppdateringar



$$A[] = \min(A[], \text{cost}(A-B) + B[])$$

A, ursprunglig

Nod	Dist
A	0
B	3
D	1

Uppdatering från B

Nod	Dist
A	3
B	0
C	1
D	1
E	1

A, uppdaterad

Nod	Dist
A	0
B	3
C	4
D	1
E	4

Not! Kostnad till B via D okänd tills D uppdaterat A

Distance Vector, funderingar

- Periodiska uppdateringar!?
- Problem med länkar och noder som försvinner.
- Hur hitta grannar?
- Hur upptäcka att en granne försvinner?

Mer om distance vector

Senare:

- Count to infinity
- Two, three node instability
- Split Horizon
- Poison Reverse
- Routingprotokoll RIP

Link state routing, princip

Lokal topologi-information flödas (*flooding*) globalt

- ◆ Vid varje förändring
- ◆ Periodiskt (i praktiken sällan)

Varje nod bygger egen databas

Routing-tabellen uppdateras med

- ◆ Nya poster
- ◆ Förändring av kostnad

”Lokal kunskap sprids globalt”

Least cost alg 2

Dijkstras

Hitta kortaste vägen från en source node s till de övriga.

Låt $d(n)$ vara kostnaden från s till n

Init:

$$d(s) = 0$$

$$d(n) = \infty, n \neq s$$

$$V = \emptyset$$

// Besökta

while $V \subset E$

$$u = \arg \min_{u \notin V} d(u)$$

// Hitta minsta vikt på obesökt nod

$$V = V \cup u$$

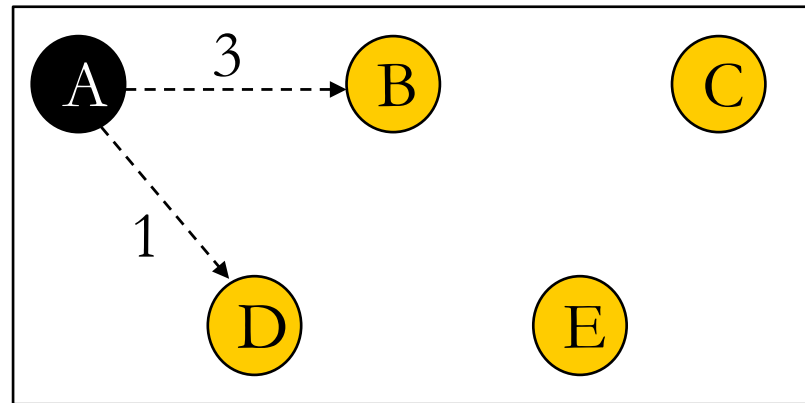
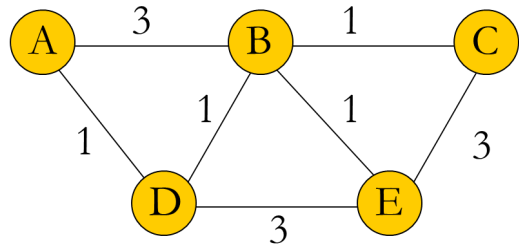
// Lägg till u bland besökta

for $n \notin V$

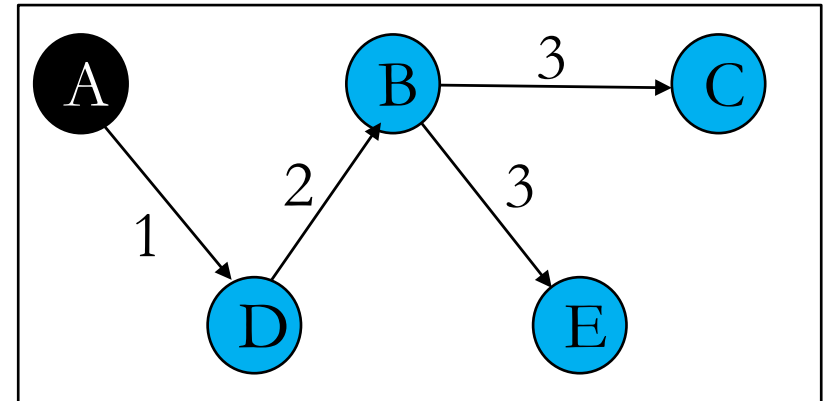
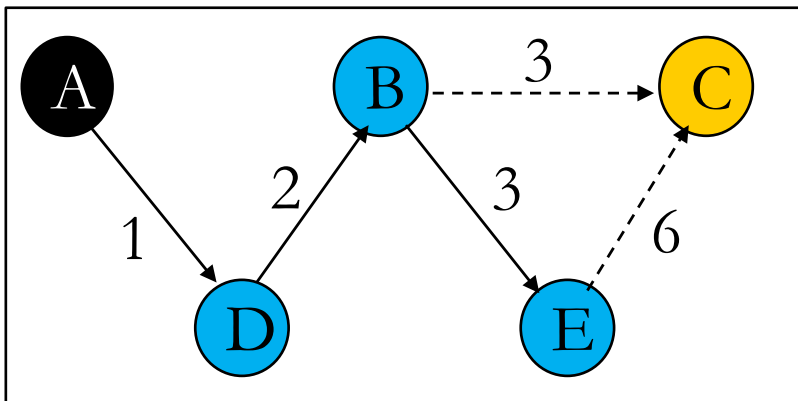
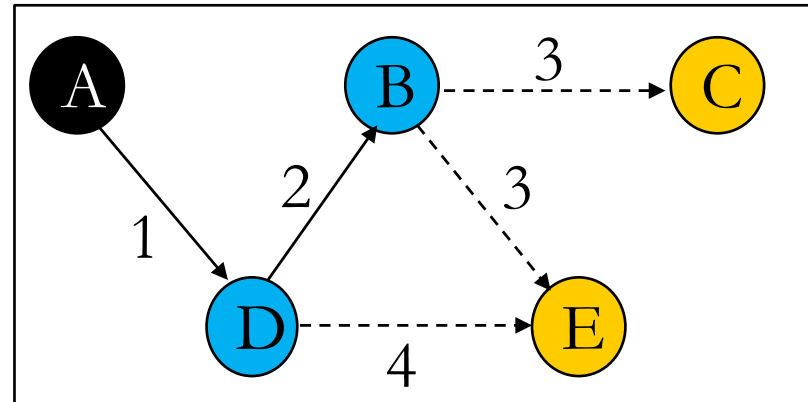
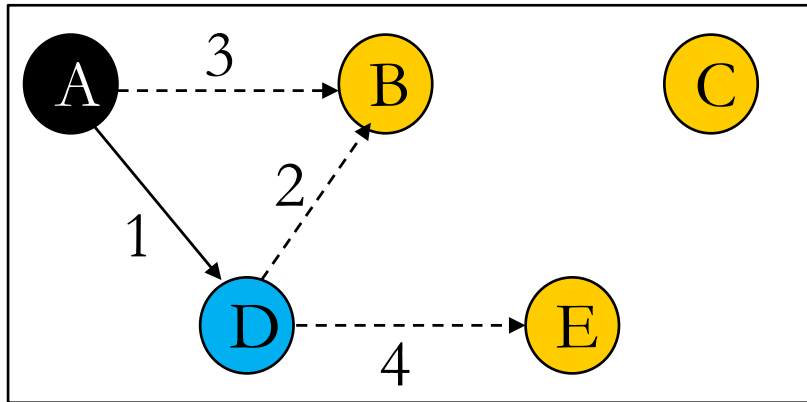
$$d(n) = \min\{d(n), d(u) + w(u, n)\} \quad // \text{Mindre vikt att gå via } u \text{ till } n?$$

Tillägg: Håll ordning på vägen!!

SPF exempel med graf



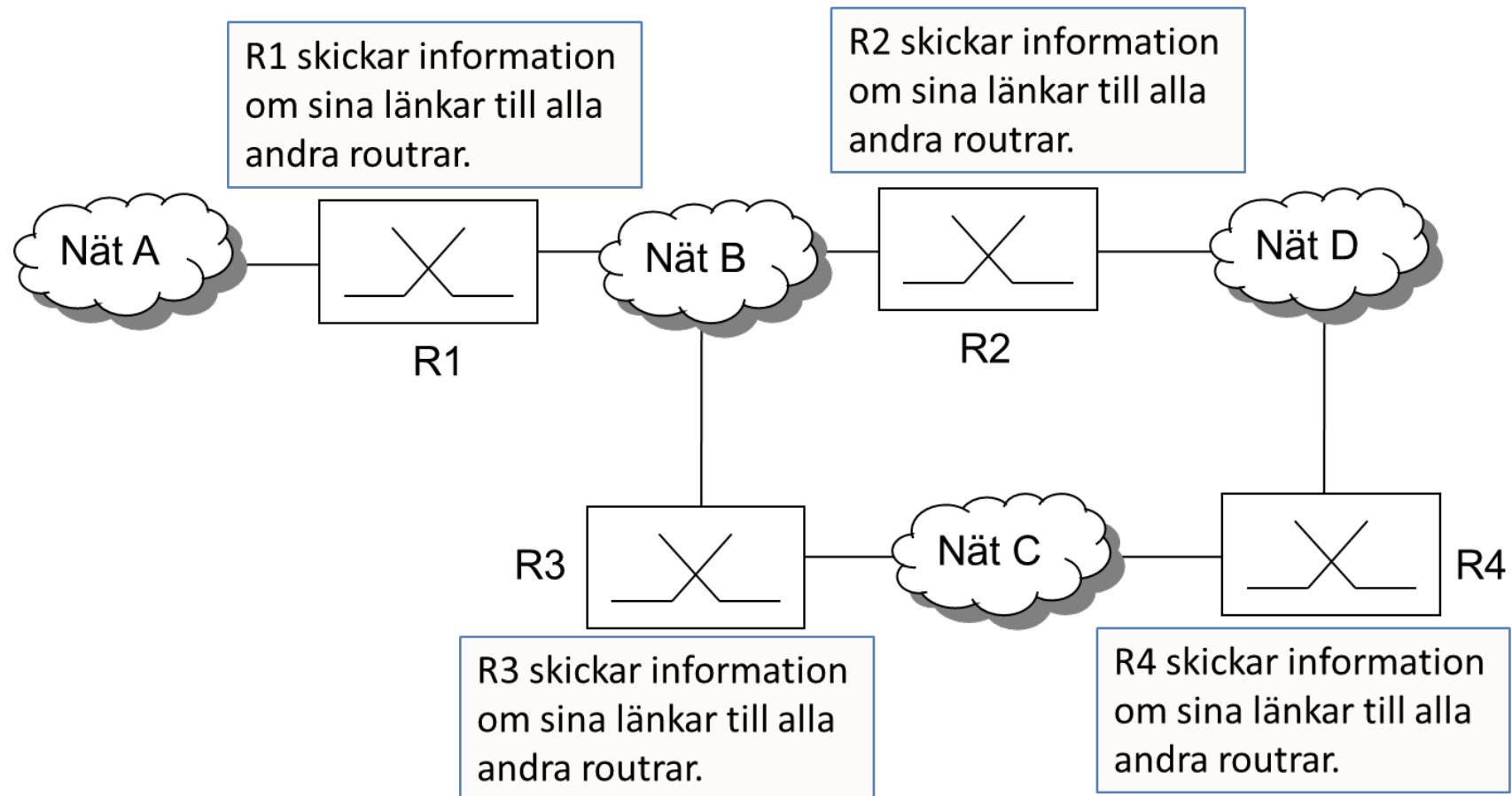
- Rotnod
- Permanent nod
- Preliminär nod
- > Potentiell väg
- > Permanent väg



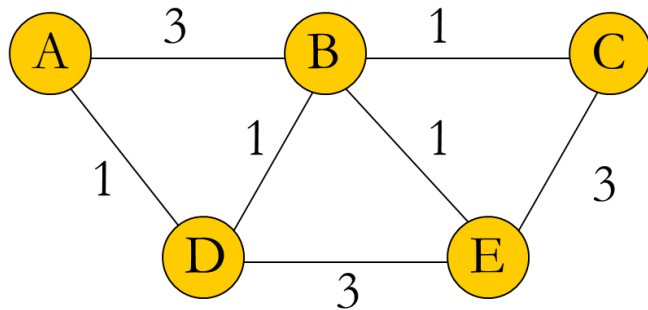
Dijkstra tabell

Besökt	L(A)	L(B)	L(C)	L(D)	L(E)
ϕ	0	∞	∞	∞	∞
{A}		3:A	∞	1:A	∞
{A,D}		2:D	∞		4:D
{A,D,B}			3:B		3:B
{A,D,B,C}					3:B
{A,D,B,C,E}					

Link state routing, princip



Exempel på *link state* databas

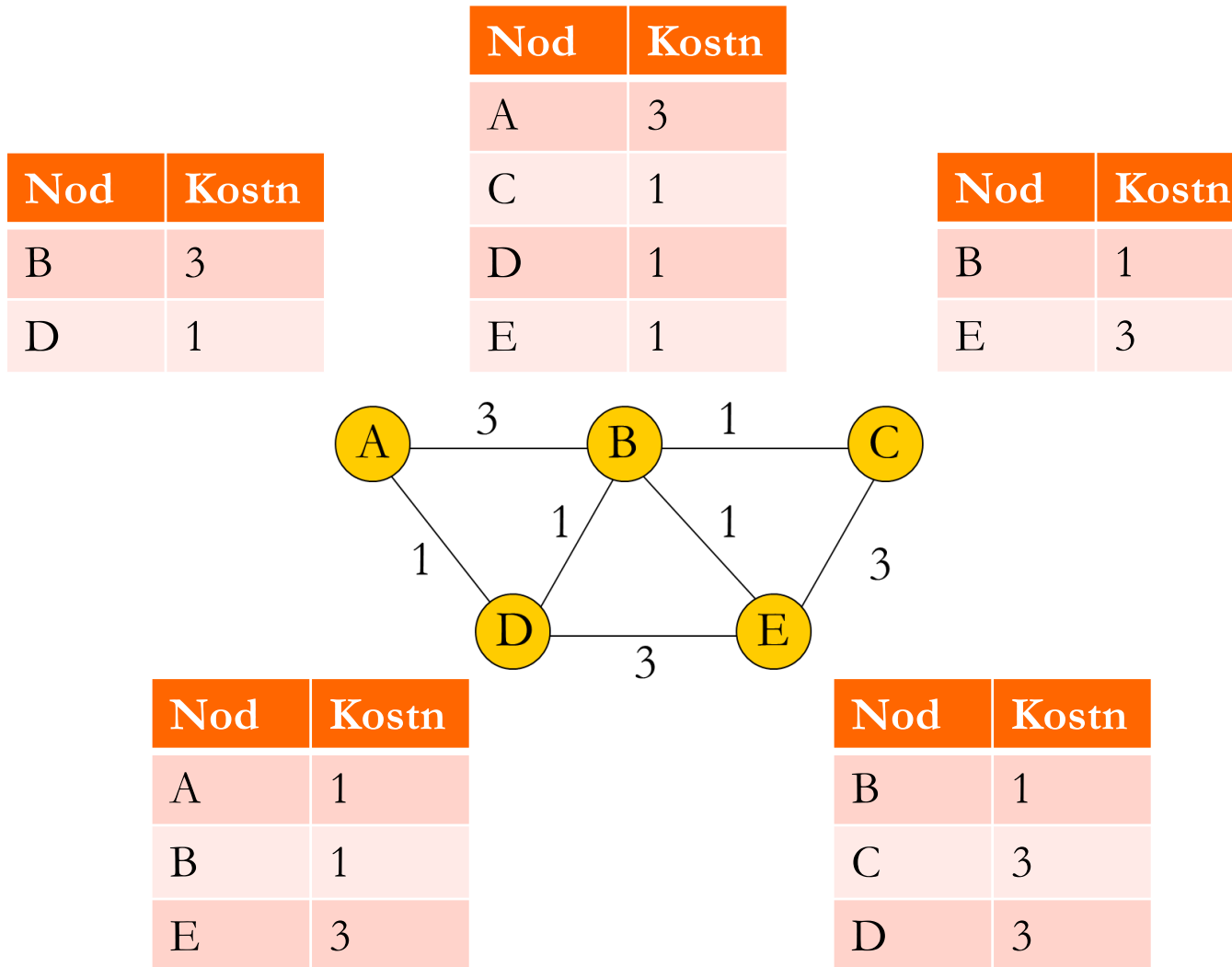


	A	B	C	D	E
A	0	3	∞	1	∞
B	3	0	1	1	1
C	∞	1	0	∞	3
D	1	1	∞	0	3
E	∞	1	3	3	0

∞ betyder okänd nod

0 avstånd till sig själv

Link State Advertisements



Uppdateras vid förändring!

Link State, funderingar

- Periodiska uppdateringar!?
- Problem med länkar och noder som försvinner.
- Hur hitta grannar?
- Hur upptäcka att en granne försvinner?

Mer om link state

Senare:

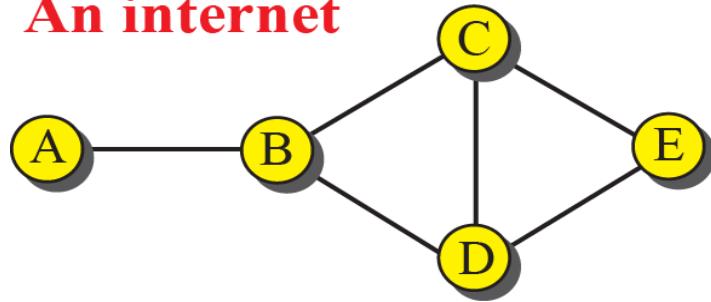
- Routingprotokoll OSPF
- Konceptet Areor eller hur man minskar flooding

Path Vector Routing

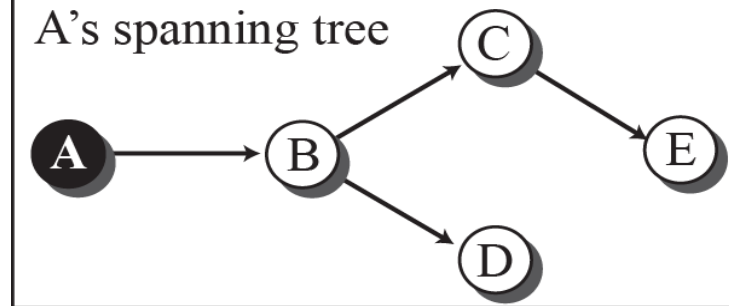
- Lägg till *path vector* för varje destination
- Påminner om *Distance Vector*
 - *Path Vector* innehåller info om hela vägen till destinationen
- Finn ***best path*** bland många tillgängliga vägar
- *Policy Based Routing*
 - Använd bara vägar som går igenom accepterade noder
 - Använd inte vägar där egna noden ingår (loop!)
 - Längd på *Path Vector* är ofta viktigaste parameter

Spanning trees in path-vector routing

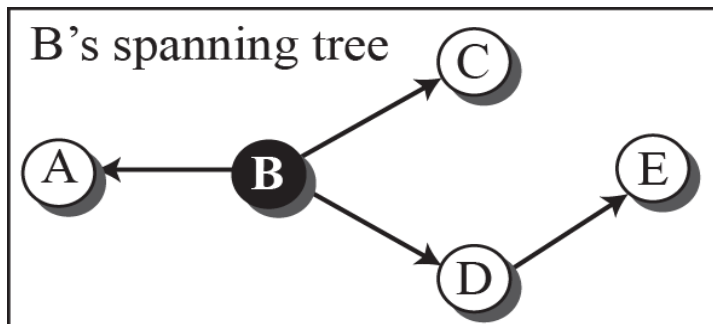
An internet



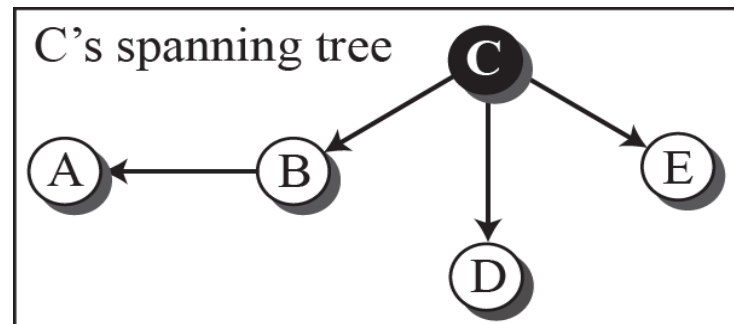
A's spanning tree



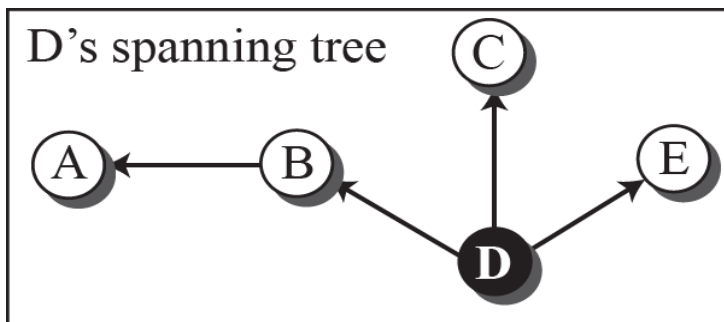
B's spanning tree



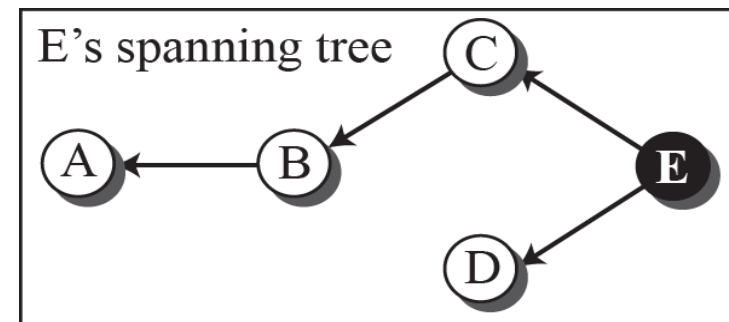
C's spanning tree



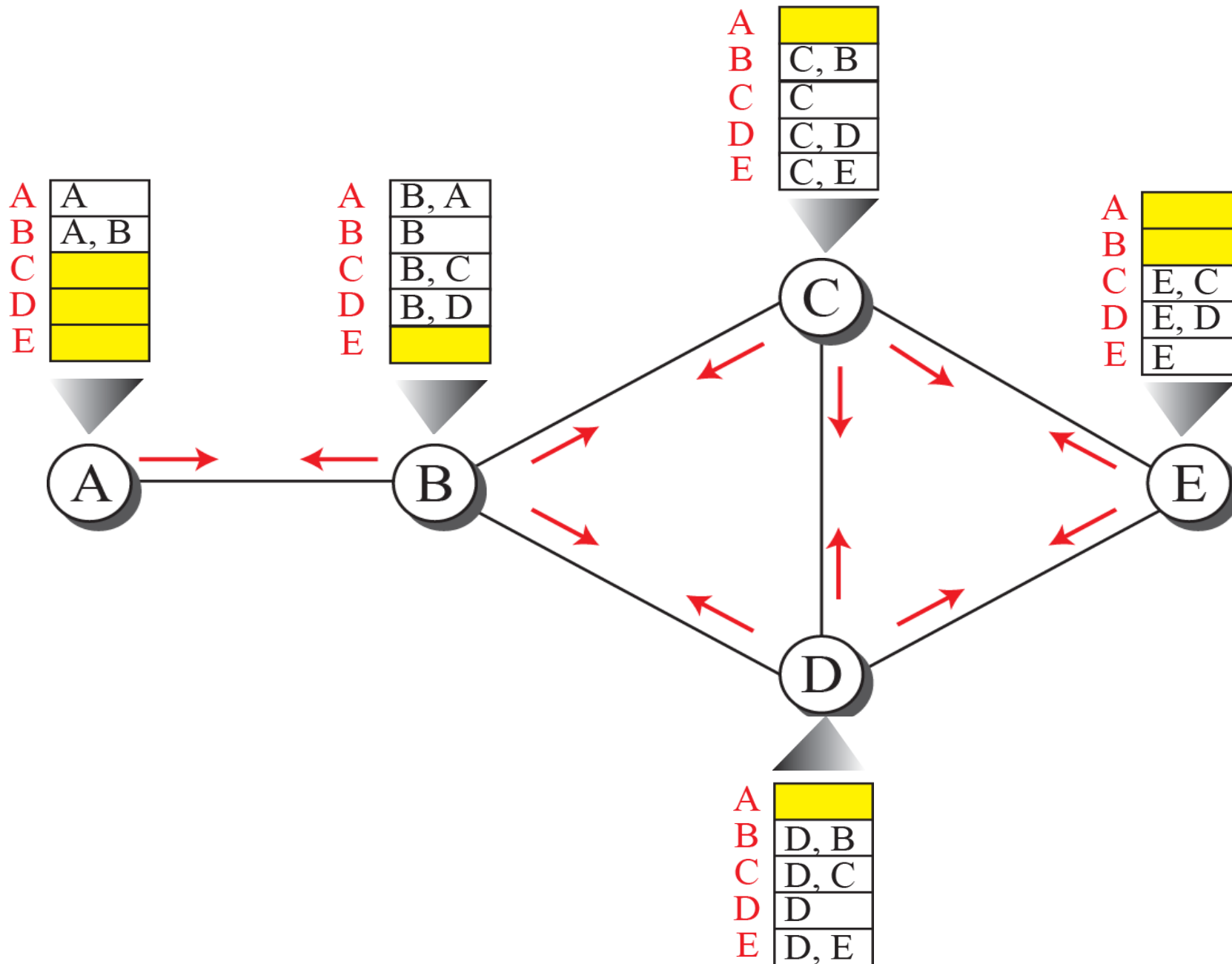
D's spanning tree



E's spanning tree



Path vectors made at booting time



Updating path vectors

	New C		Old C		B
A	C, B, A	A		A	B, A
B	C, B	B	C, B	B	B
C	C	C	C	C	B, C
D	C, D	D	C, D	D	B, D
E	C, E	E	C, E	E	

$C[] = \text{best} (C[], C + B[])$

Note:
X []: vector X
Y: node Y

Event 1: C receives a copy of B's vector

”best” är
 implementering
 av policyn

	New C		Old C		D
A	C, B, A	A	C, B, A	A	
B	C, B	B	C, B	B	D, B
C	C	C	C	C	D, C
D	C, D	D	C, D	D	D
E	C, E	E	C, E	E	D, E

$C[] = \text{best} (C[], C + D[])$

Event 2: C receives a copy of D's vector

Mer om path vector

Senare:

- Autonoma system, AS
- Routing mellan domäner
- Policy routing
- Routingprotokoll BGP

Comparison of LS and DV algorithms

Message complexity

LS: with n nodes, E links, $O(nE)$ msgs for full knowledge, changes sent to all nodes

DV: exchange between neighbours only

Speed of Convergence

LS: $O(n^2)$ algorithm requires $O(nE)$ msgs

- ◆ may have oscillations

DV: convergence time varies

- ◆ may be routing loops
- ◆ count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- ◆ node can advertise incorrect *link* cost for its “own” links
- ◆ node can break broadcast path or change broadcast info
- ◆ each node computes only its *own* table

DV:

- ◆ DV node can advertise incorrect *path* cost to any path
- ◆ each node’s table used by others
 - error propagate thru network