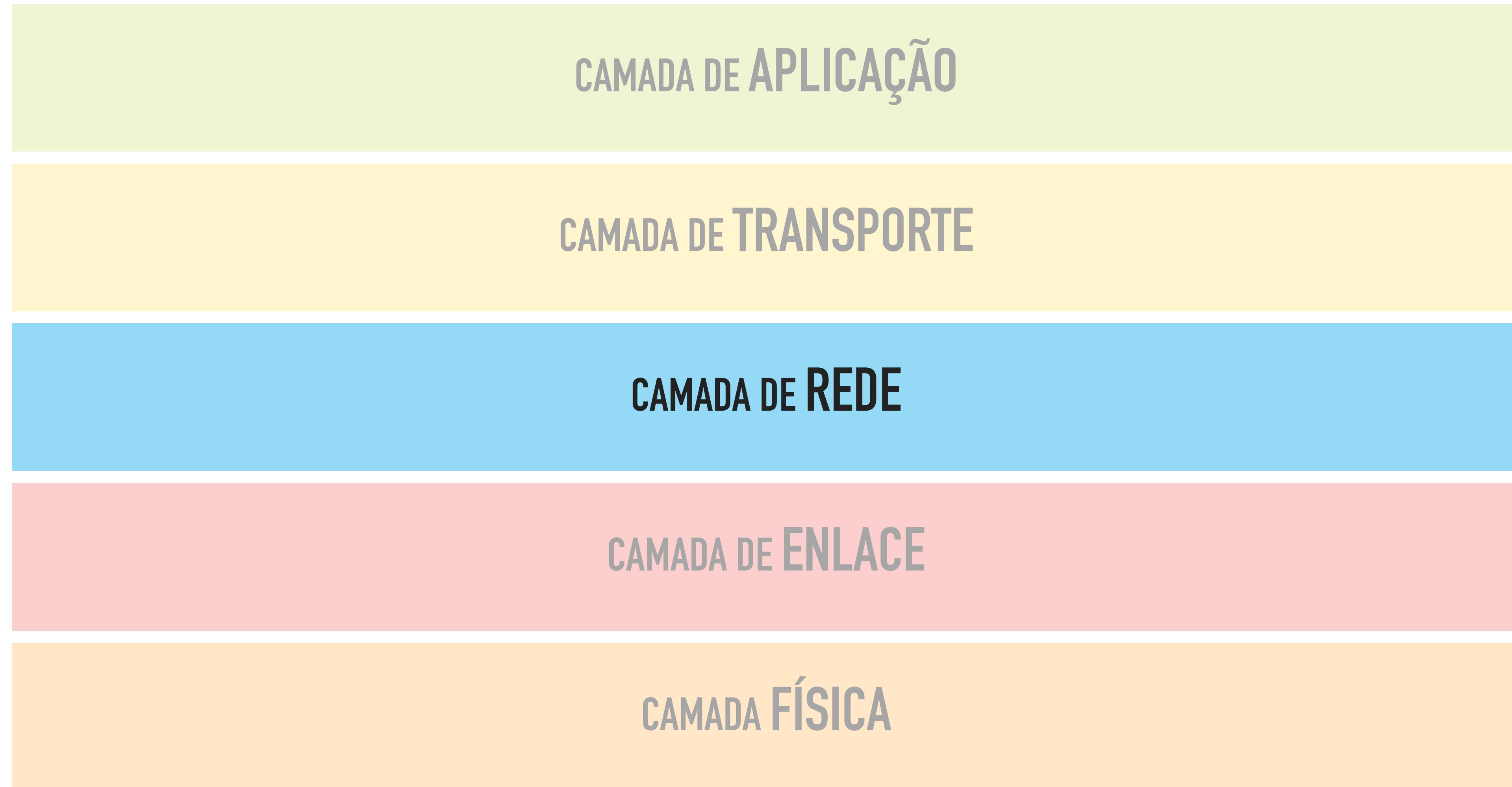


IMD0043

PRINCÍPIOS DE ROTEAMENTO



fim da camada de enlace...

início da camada de rede!

POR QUE PRECISAMOS DE UMA CAMADA DE REDE?

- ▶ Por que não usamos simplesmente o STP através de toda a rede?
- ▶ Algoritmo fácil usando o STP para roteamento (hipotético!):

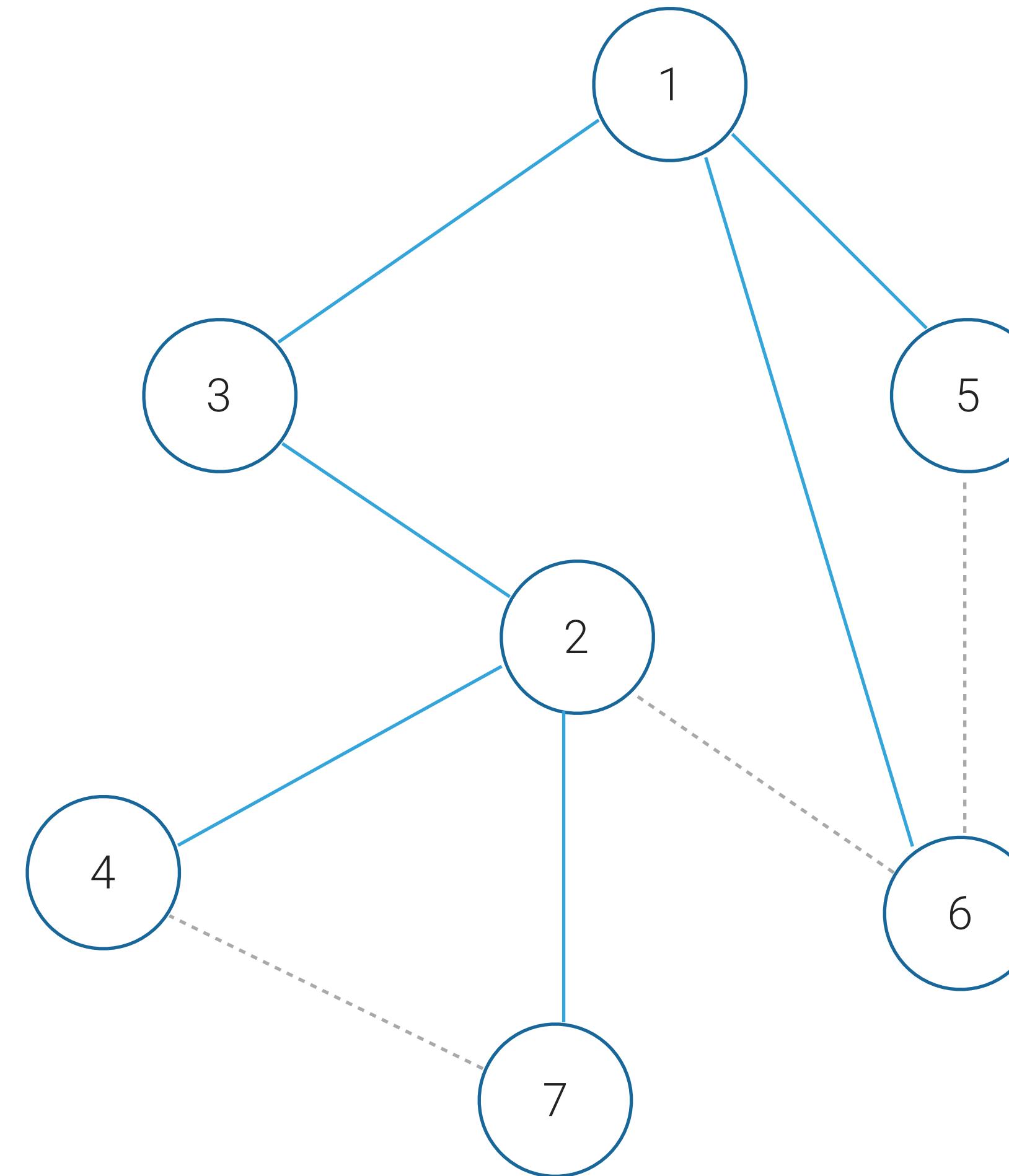
Passo 1. Nó de origem *flooda* o pacote em seus *links* ativos do STP;

Passo 2. Sempre que um nó recebe um pacote:

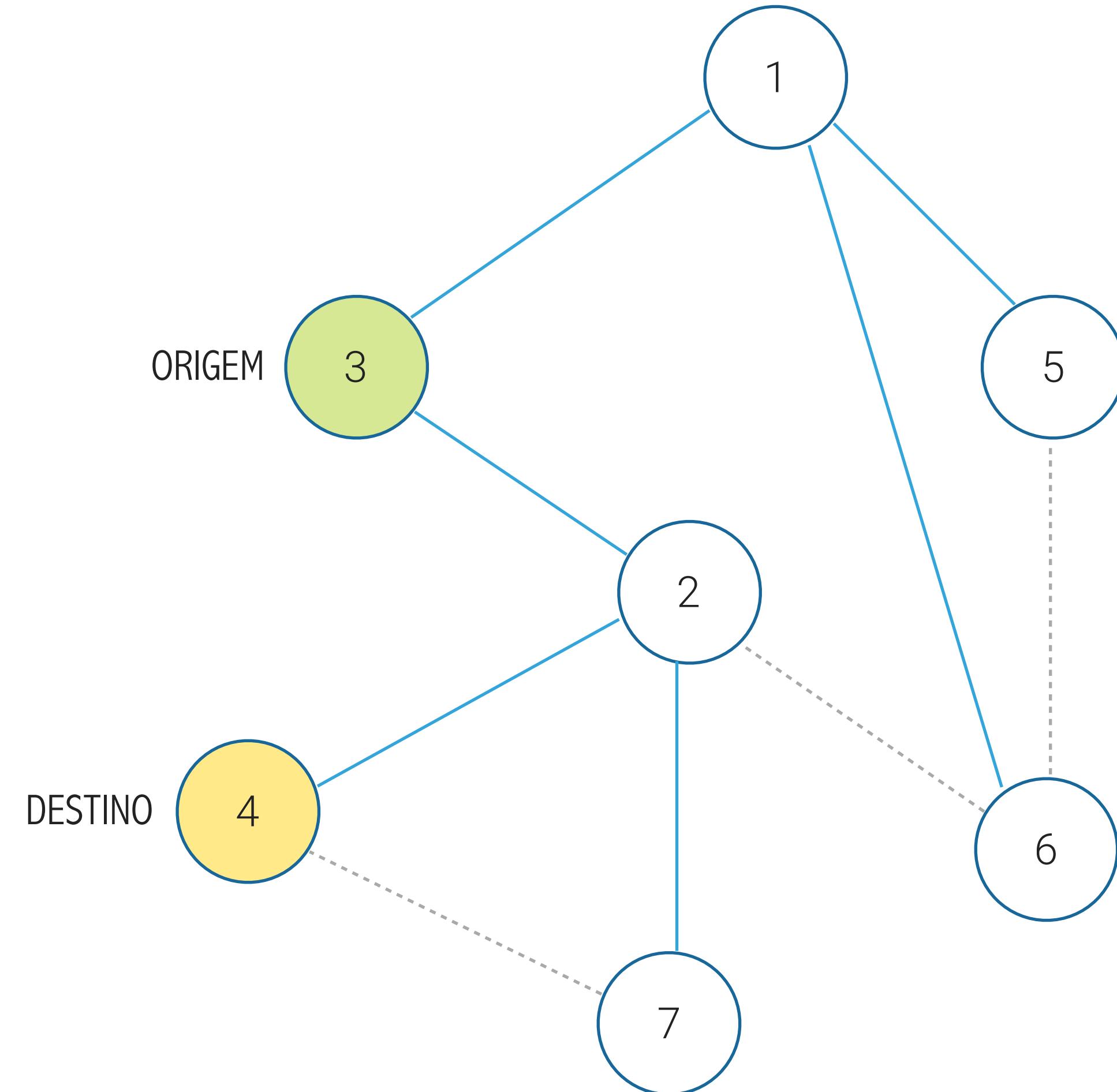
- ▶ Encaminha o pacote que chegou para todos os *links* ativos com exceção do *link* de entrada

FLOODING NA SPANNING TREE

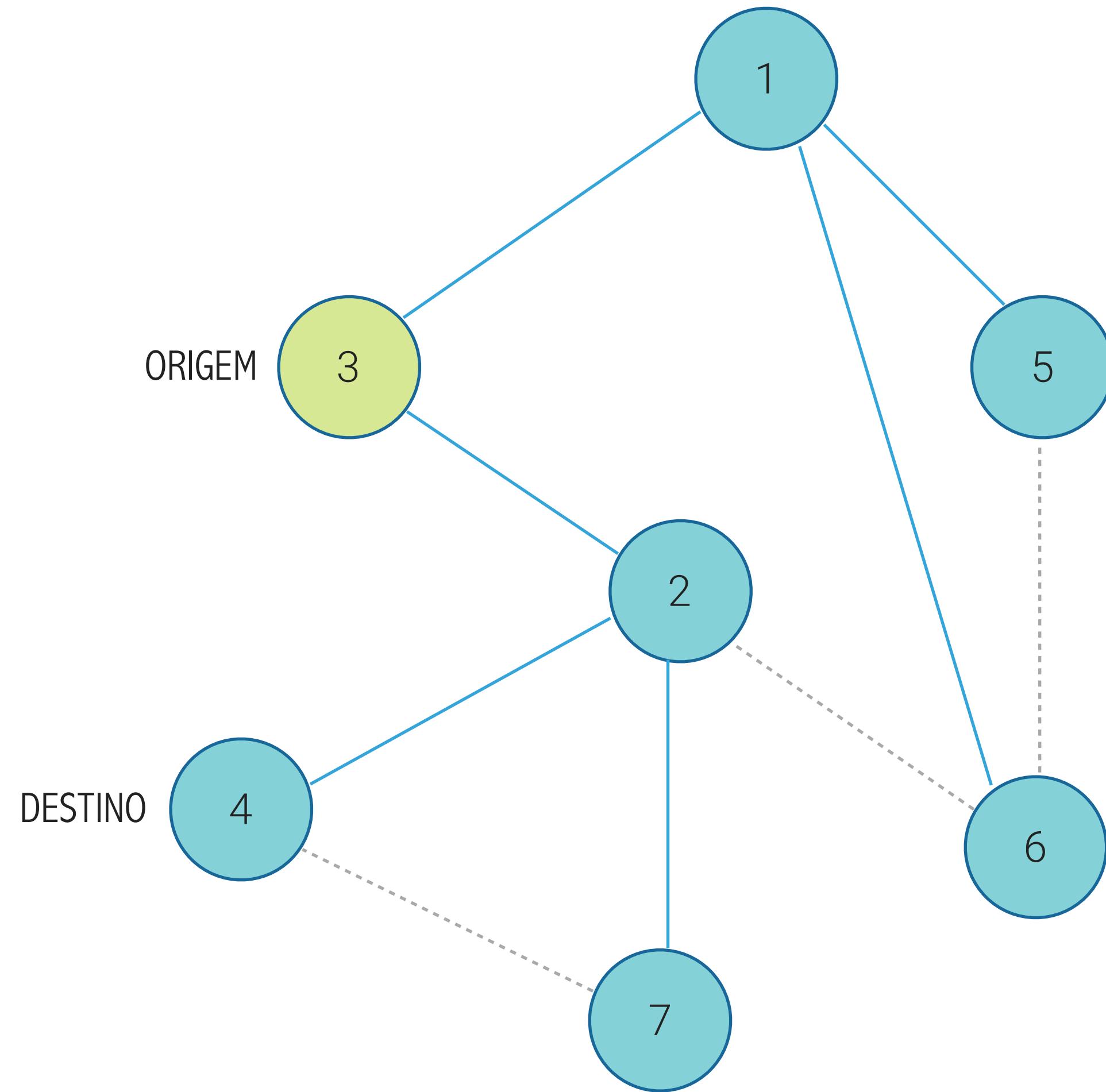
- ▶ Envia o pacote para todo nó da rede
- ▶ **Passo 1:** ignora os *links* que não fazem parte da *spanning tree*
- ▶ **Passo 2:** origem *floods* o pacote para todos os *links*
- ▶ **Passo 3:** envia o pacote que chegou para todos os *links* com exceção do *link* de entrada



EXEMPLO



EXEMPLO



Eventualmente, todos os nós recebem uma cópia do pacote, incluindo o destino

ROTEAMENTO ATRAVÉS DE FLOODING NA SPANNING TREE

- ▶ Algoritmo fácil usando o STP para roteamento (hipotético!):

Passo 1. Nó de origem *flooda* o pacote em seus *links* ativos do STP;

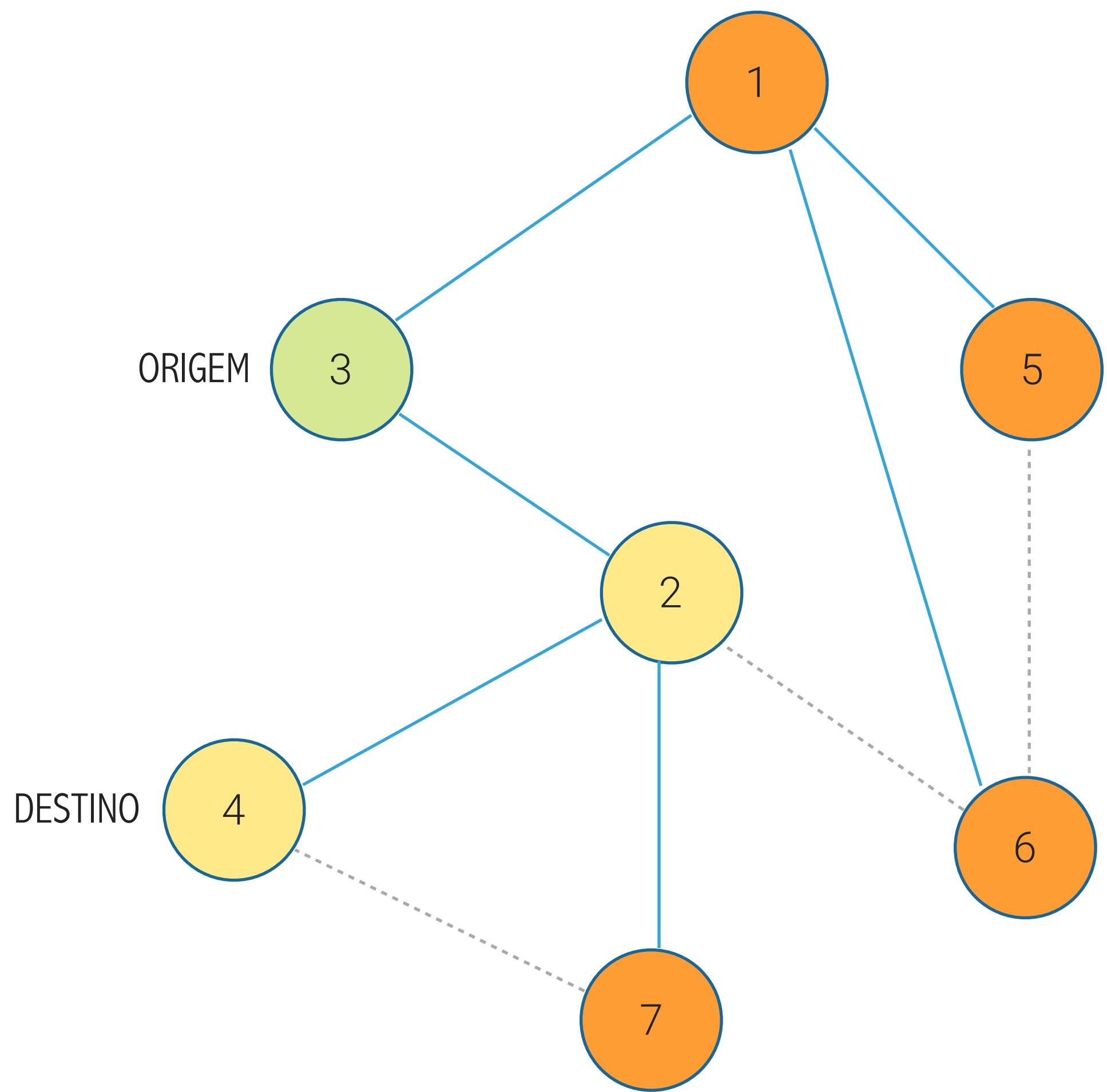
Passo 2. Sempre que um nó recebe um pacote:

- ▶ Encaminha o pacote que chegou para todos os *links* ativos com exceção do *link* de entrada

- ▶ Propriedades:

- ▶ Não são necessárias tabelas de roteamento!
- ▶ Nenhum pacote irá ficar em *loop*
- ▶ Pelo menos (e exatamente) um pacote irá chegar ao destino (assumindo sem falhas)

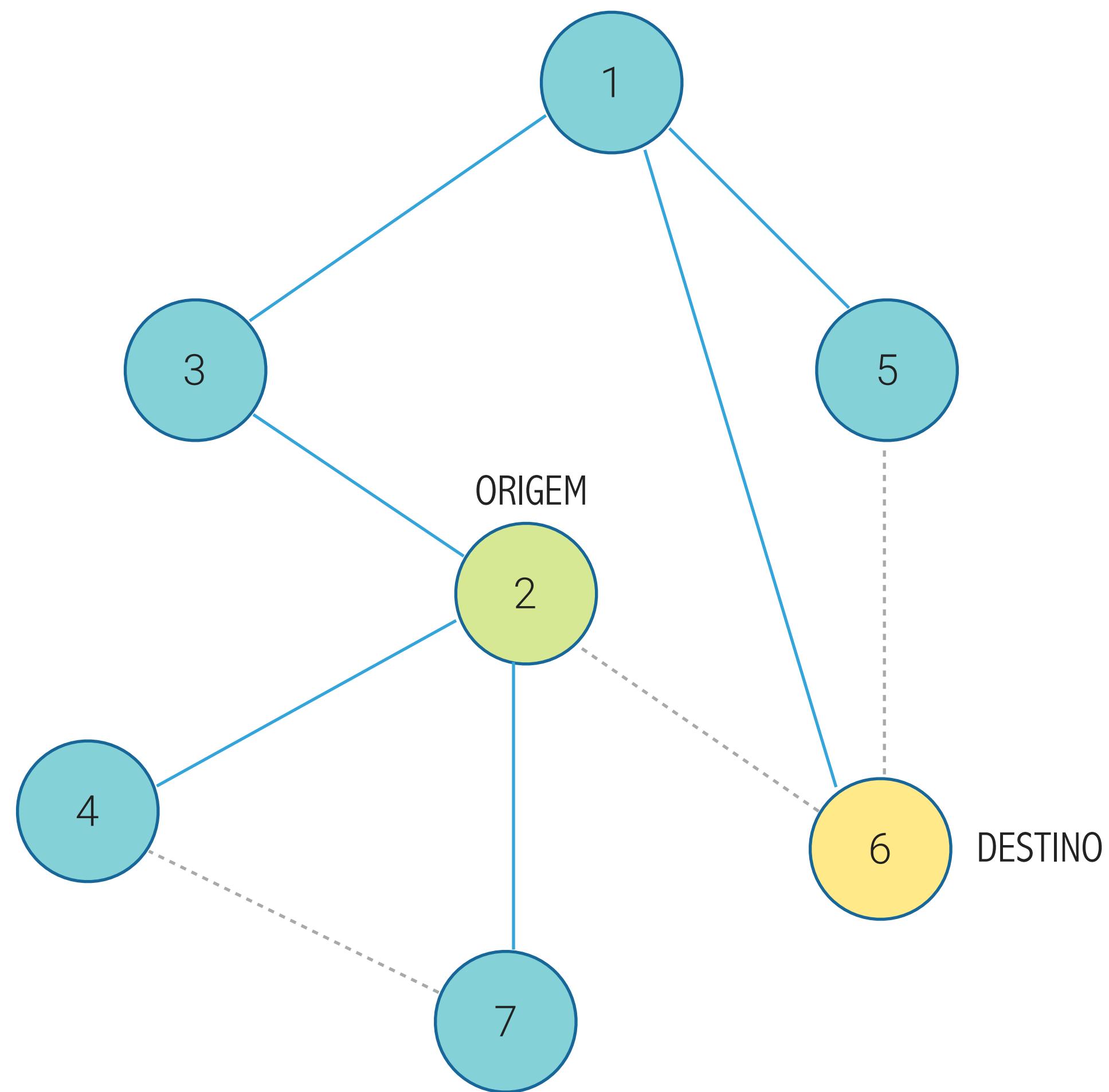
3 PROBLEMAS FUNDAMENTAIS



Problema 1:

Todo nó deve fazer processamento de pacotes
(mesmo desnecessariamente).

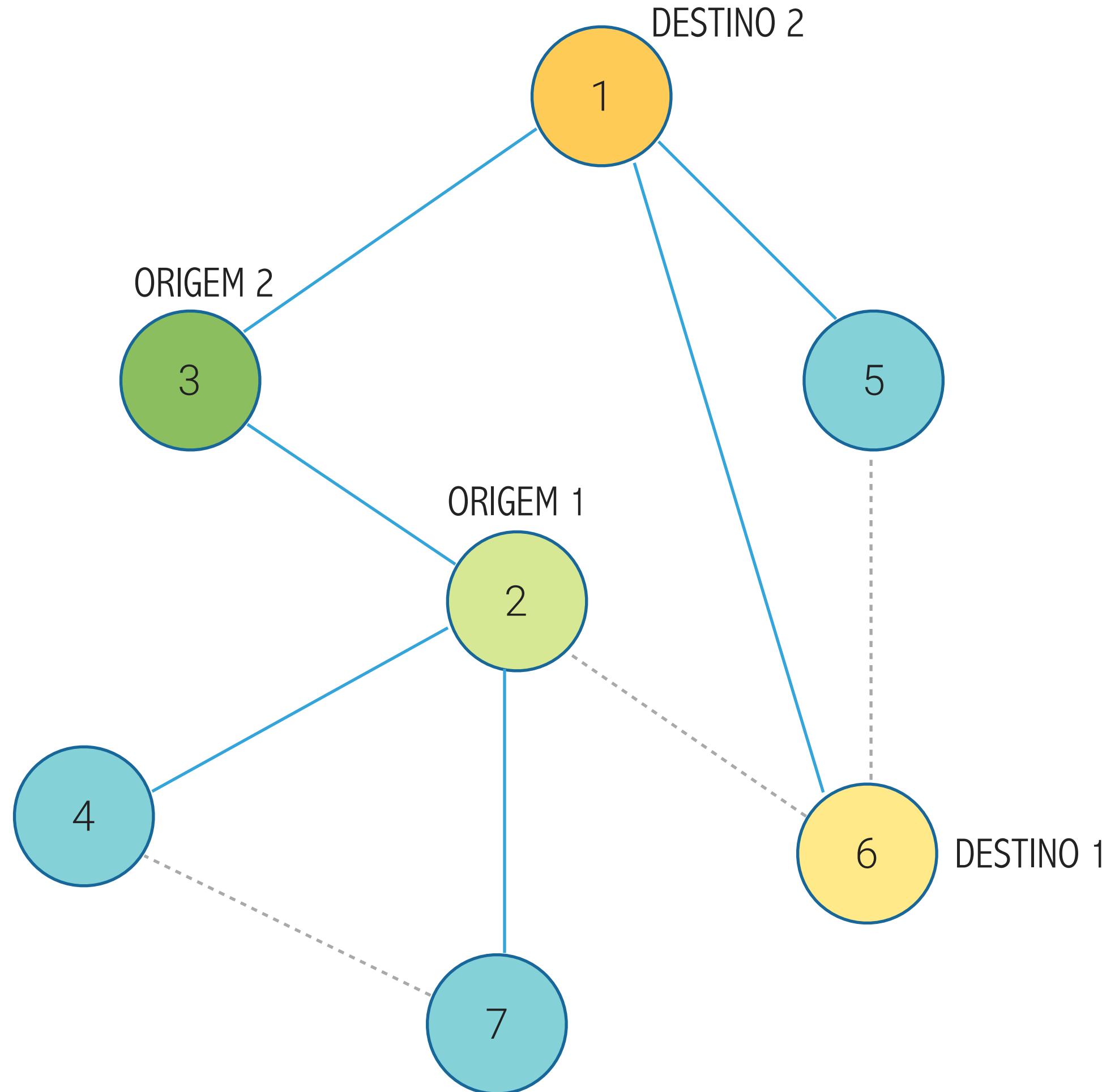
3 PROBLEMAS FUNDAMENTAIS



Problema 2:

Latência maior! Os pacotes podem percorrer caminhos muito mais longos desnecessariamente.

3 PROBLEMAS FUNDAMENTAIS



Problema 3:

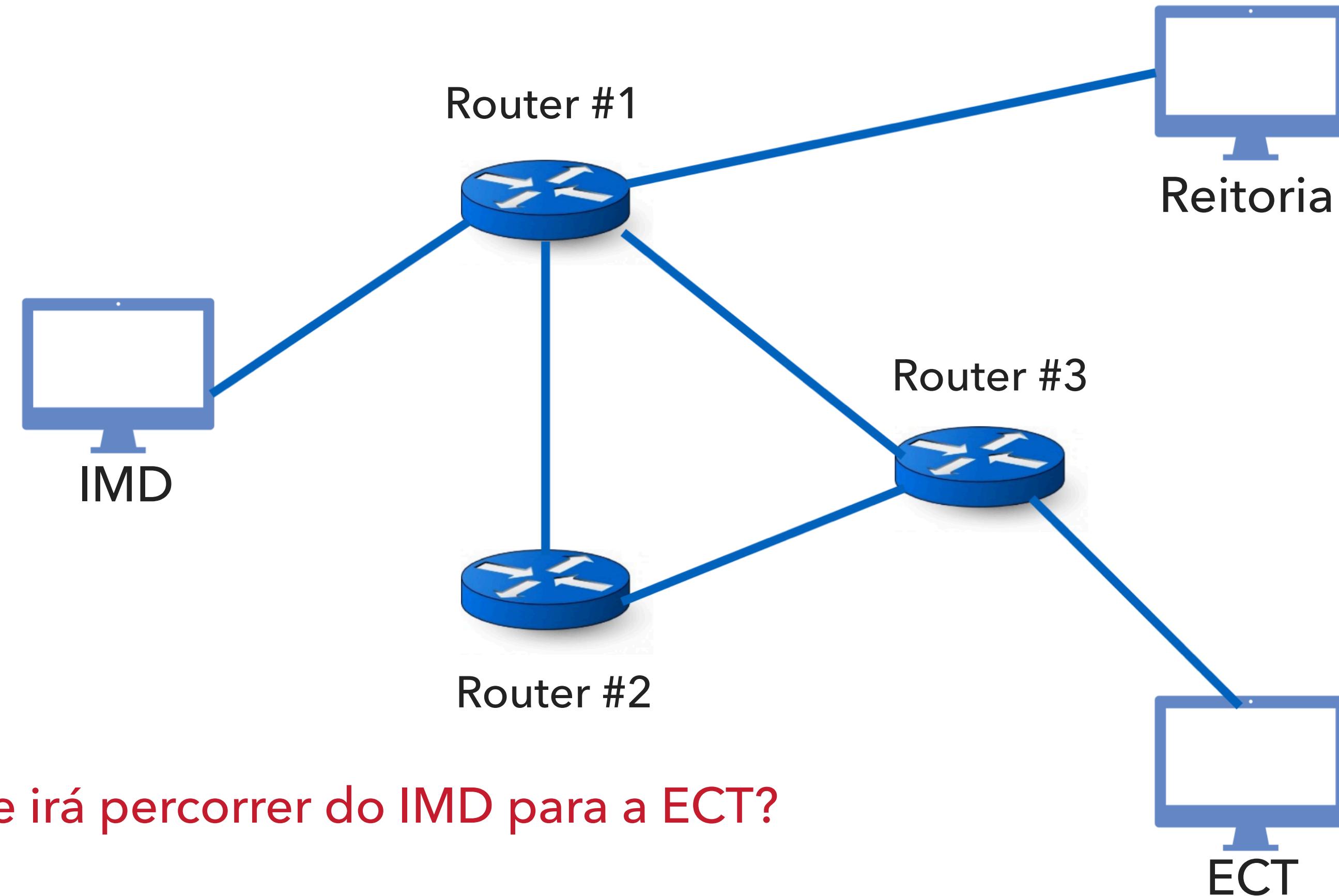
Desperdício de largura de banda!
Pacotes em 2-6 e 3-1 compartilham
largura de banda desnecessariamente.

POR QUE PRECISAMOS DE UMA CAMADA DE REDE?

- ▶ Camada de rede realiza o roteamento de pacotes para mitigar esses problemas
- ▶ Utiliza **tabelas de roteamento**
- ▶ Vamos entender o que são as tabelas de roteamento primeiro!

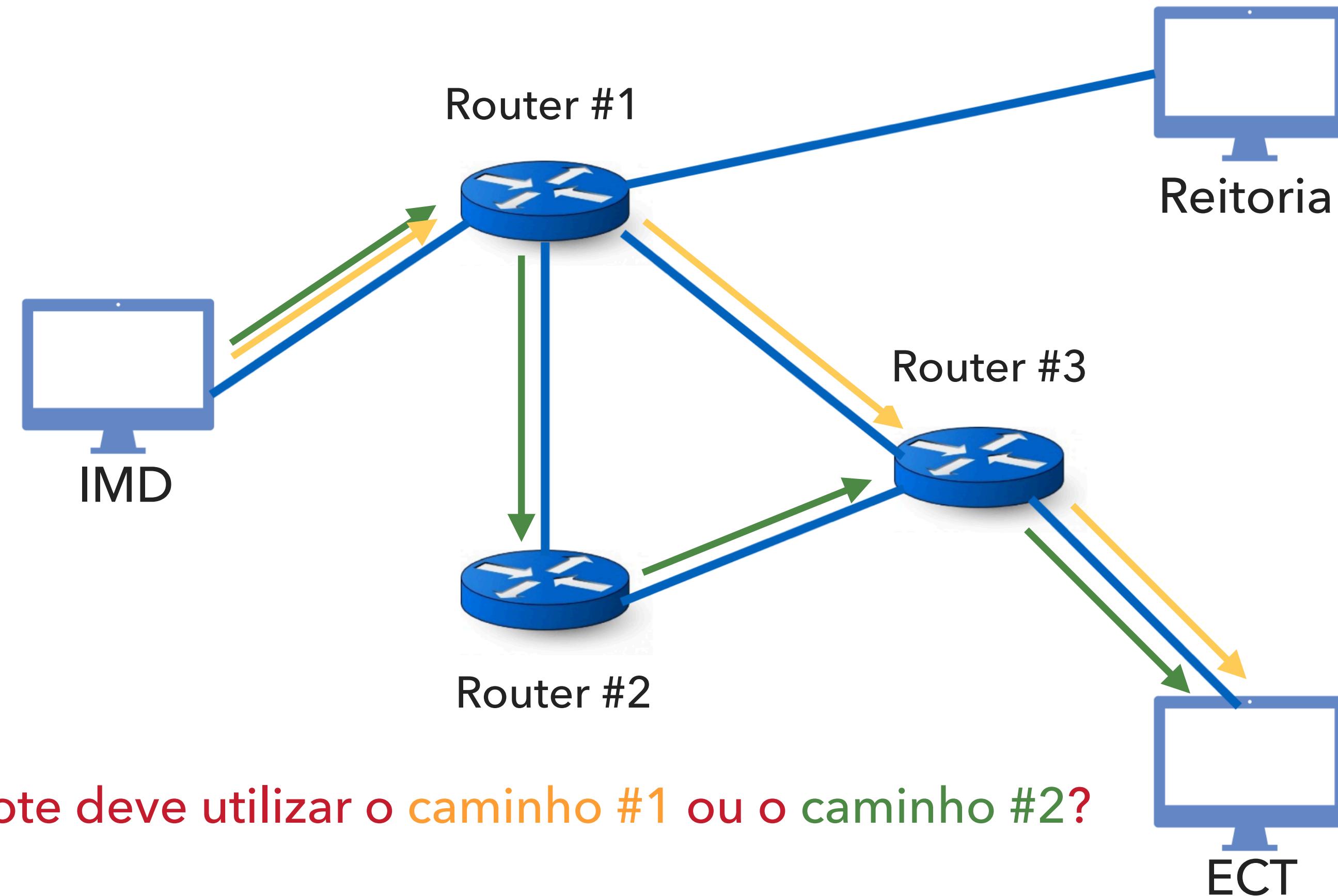
ROTEANDO PACOTES ATRAVÉS DE TABELAS DE ROTEAMENTO

Tabelas de roteamento permitem encontrar um **caminho** da origem até o destino



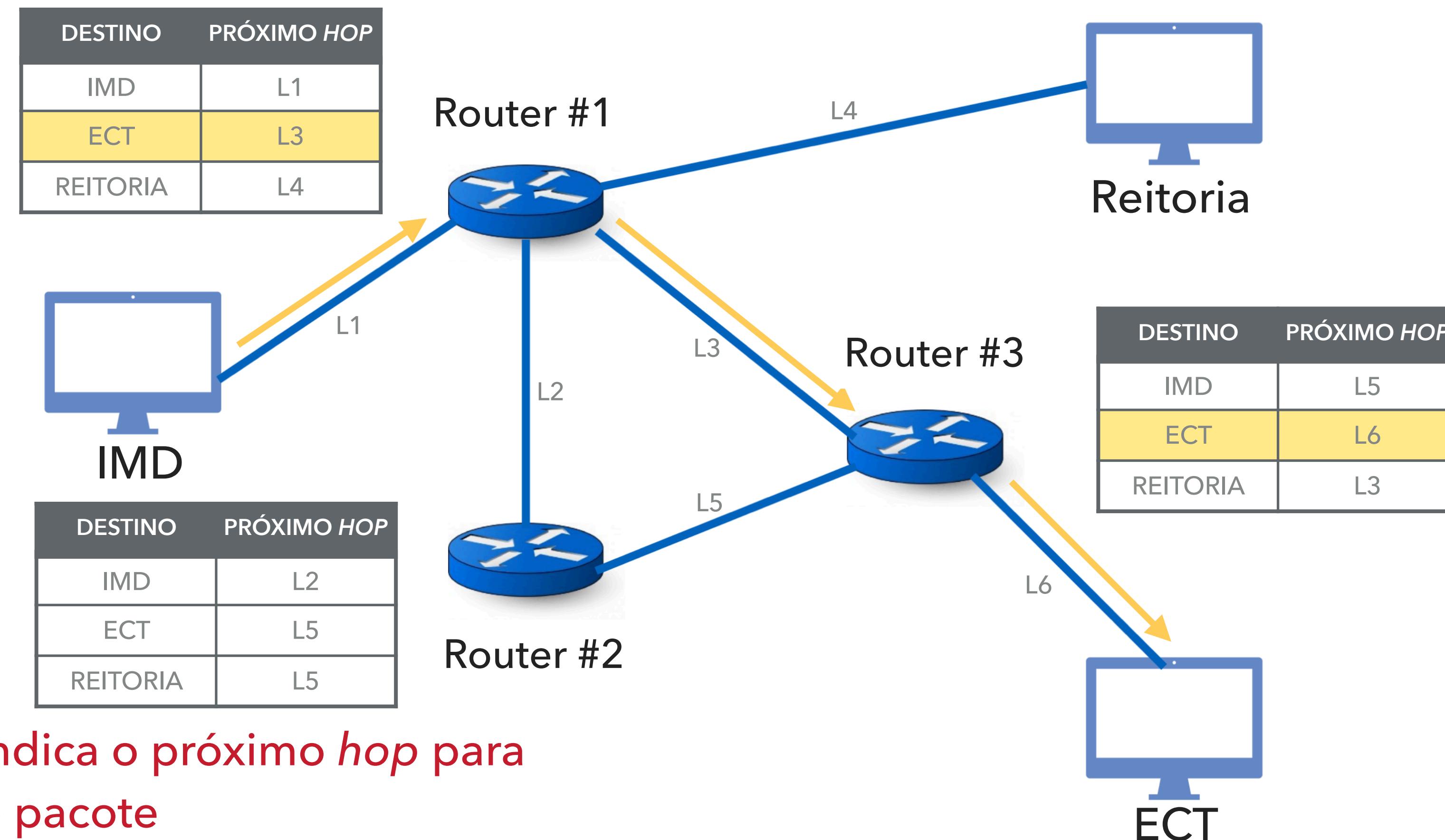
ROTEANDO PACOTES ATRAVÉS DE TABELAS DE ROTEAMENTO

Tabelas de roteamento permitem encontrar um **caminho** da origem até o destino



ROTEANDO PACOTES ATRAVÉS DE TABELAS DE ROTEAMENTO

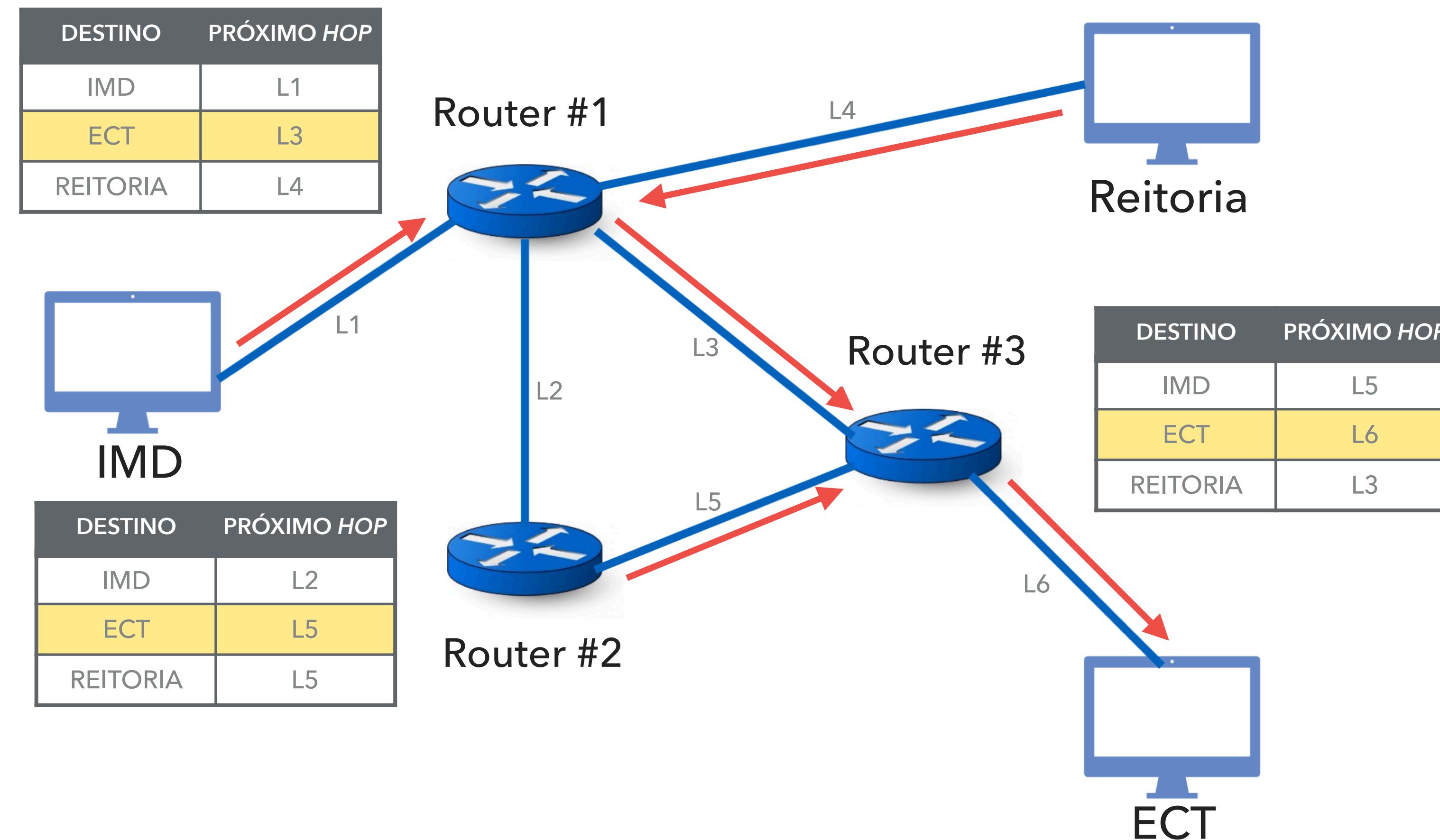
Suponha que o pacote percorra o caminho #1: IMD - R#1 - R#3 - ECT



Cada nó de roteamento indica o próximo *hop* para o destino especificado no pacote

TABELA DE ROTEAMENTO

Vamos focar em um destino: ECT



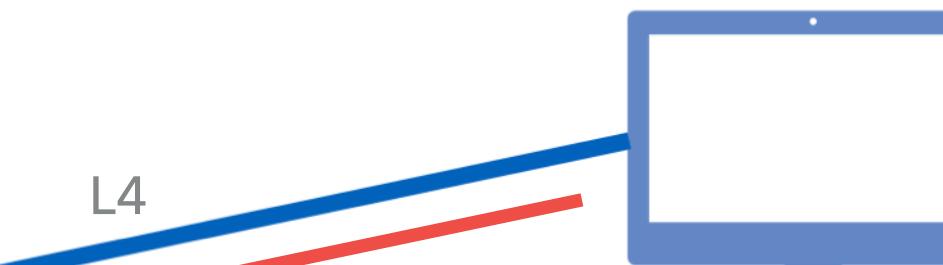
Notou algo interessante?

TABELA DE ROTEAMENTO

Vamos focar em um destino: ECT

DESTINO	PRÓXIMO HOP
IMD	L1
ECT	L3

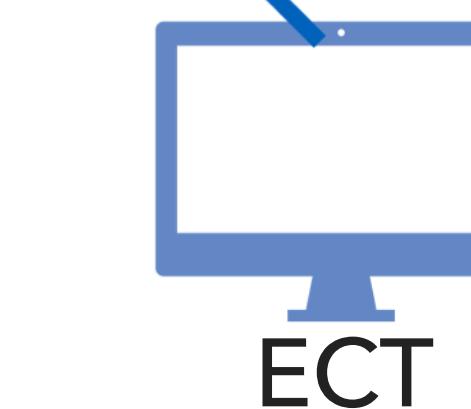
Router #1



Entradas nas tabelas de roteamento para um destino em particular formam uma *spanning tree* com aquele destino como raiz (*root*)!

DESTINO	PRÓXIMO HOP
IMD	L2
ECT	L5
REITORIA	L5

Router #2



L6

TABELA DE ROTEAMENTO

Vamos focar em um destino: ECT

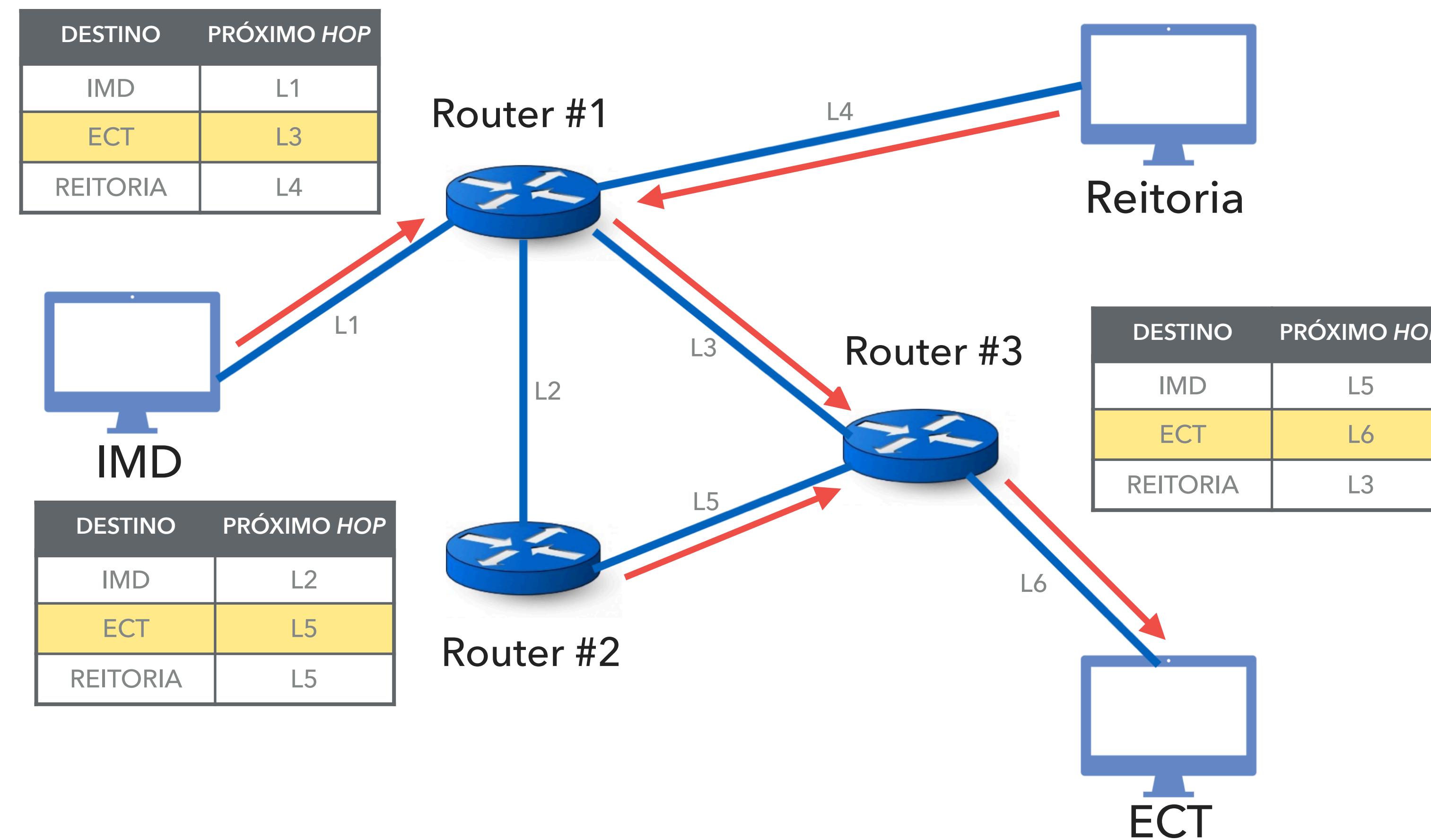


TABELA DE ROTEAMENTO

Tabelas de roteamento não são nada mais que:

- ▶ Uma coleção de *spanning trees*
- ▶ Uma para cada destino

Protocolos de roteamento:

- ▶ n protocolos de *spanning tree* executando em paralelo

TABELAS DE ROTEAMENTO “VÁLIDAS”

Estado de roteamento global é válido se:

- ▶ **sempre** resulta na entrega de pacotes aos seus destinos

Objetivo dos protocolos de roteamento:

- ▶ Calcular um estado válido
 - ▶ Como saber se um estado de roteamento é válido?
 - ▶ O que pode resultar em um roteamento incorreto?

VALIDADE DO ESTADO DE ROTEAMENTO

Um estado de roteamento global se e somente se:

- ▶ não existem *dead ends* (além do destino)
- ▶ não existem *loops*

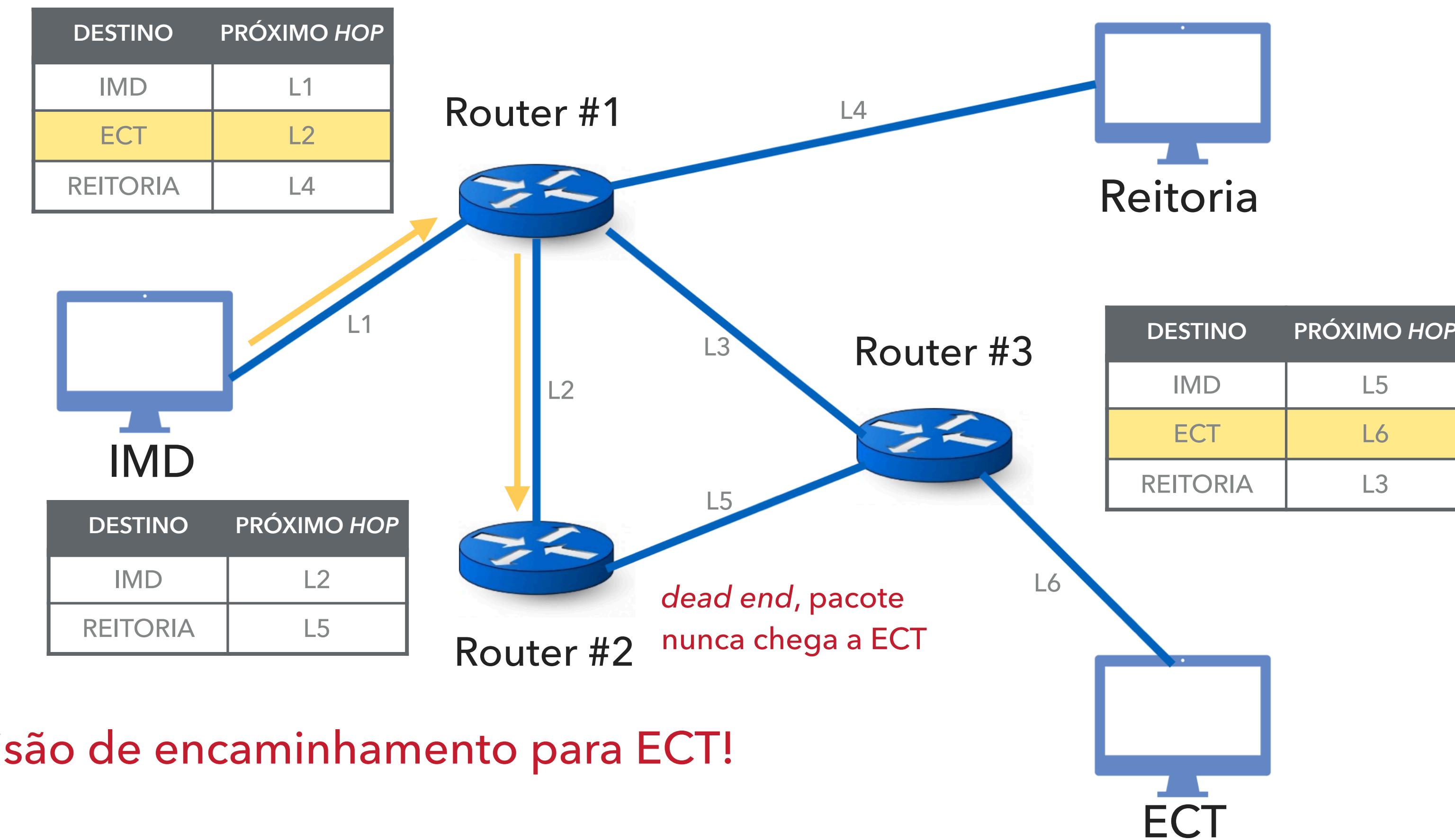
Um *dead end* é quando não temos nenhum *link* de saída

- ▶ Um pacote chega, mas...
- ▶ a tabela de roteamento não tem nenhum *link* de saída
- ▶ E o nó em questão não é o destino

Um *loop* é quando um pacote fica circulando através dos mesmos nós infinitamente

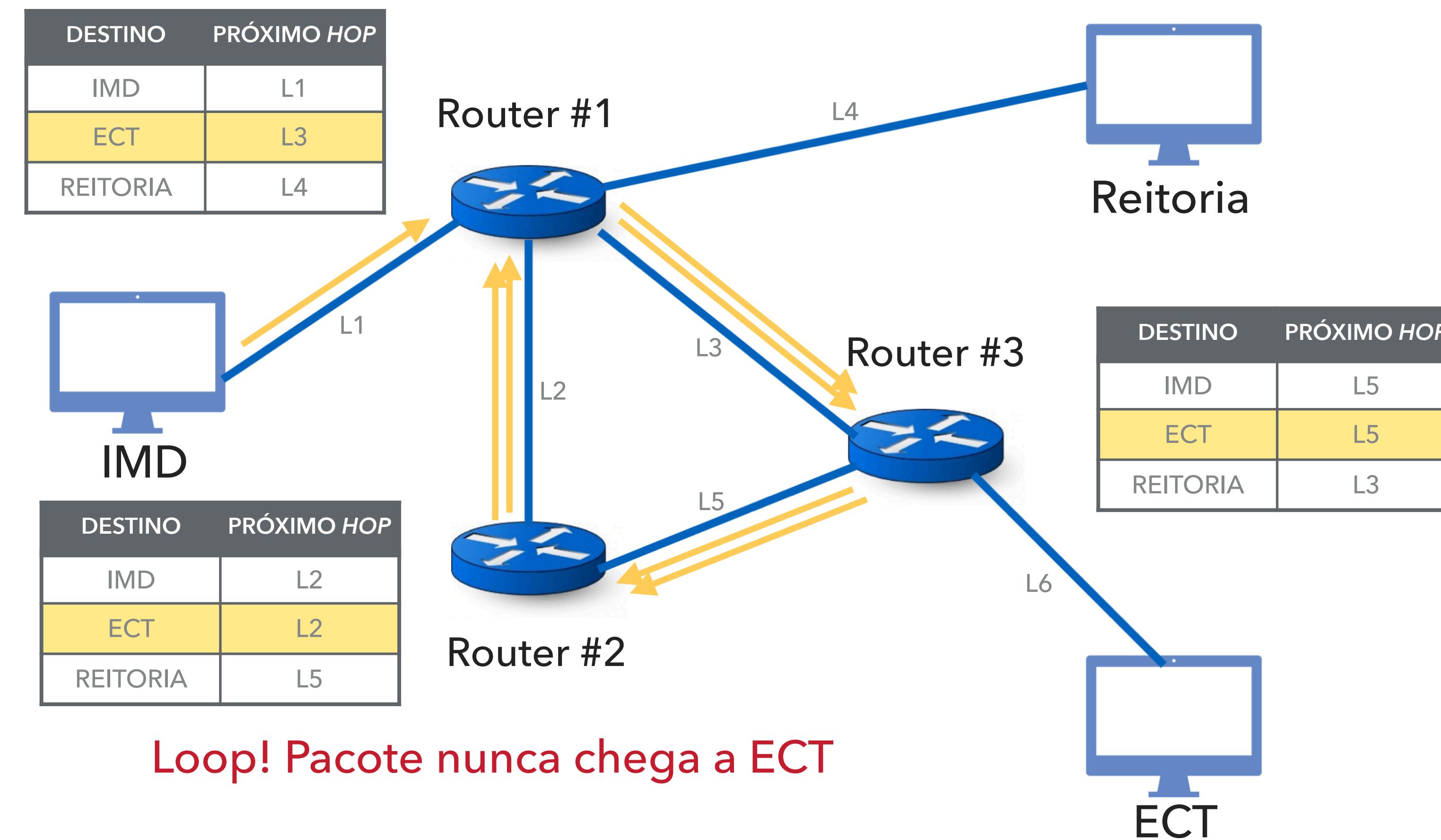
EXEMPLO: ROTEANDO COM DEAD ENDS

Suponha que o pacote irá de IMD para ECT.



EXEMPLO: ROTEANDO COM LOOPS

Suponha que o pacote irá de IMD para ECT.



DUAS QUESTÕES

Como podemos **verificar** que um dado estado de roteamento é válido?

Como podemos **produzir** um estado de roteamento válido?

VERIFICANDO A VALIDADE DO ESTADO DE ROTEAMENTO

Checar a validade do estado de roteamento para um destino de cada vez...

Para cada nó:

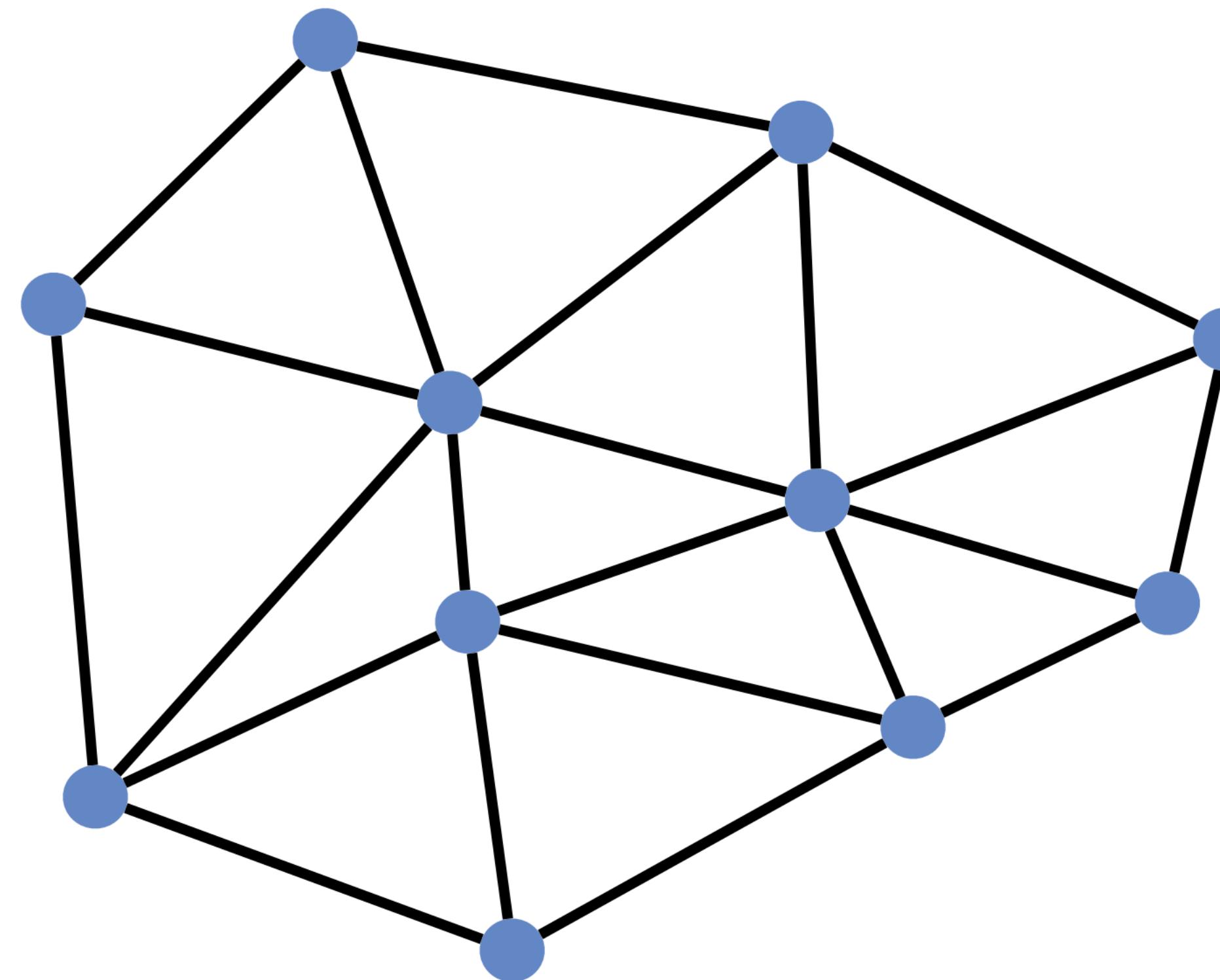
- ▶ Marque o *link* de saída direcionado para aquele destino
- ▶ Só pode haver um *link* para cada nó

Elimine todos os *links* não direcionados

Verifique o que restou. Estado é válido se e somente se:

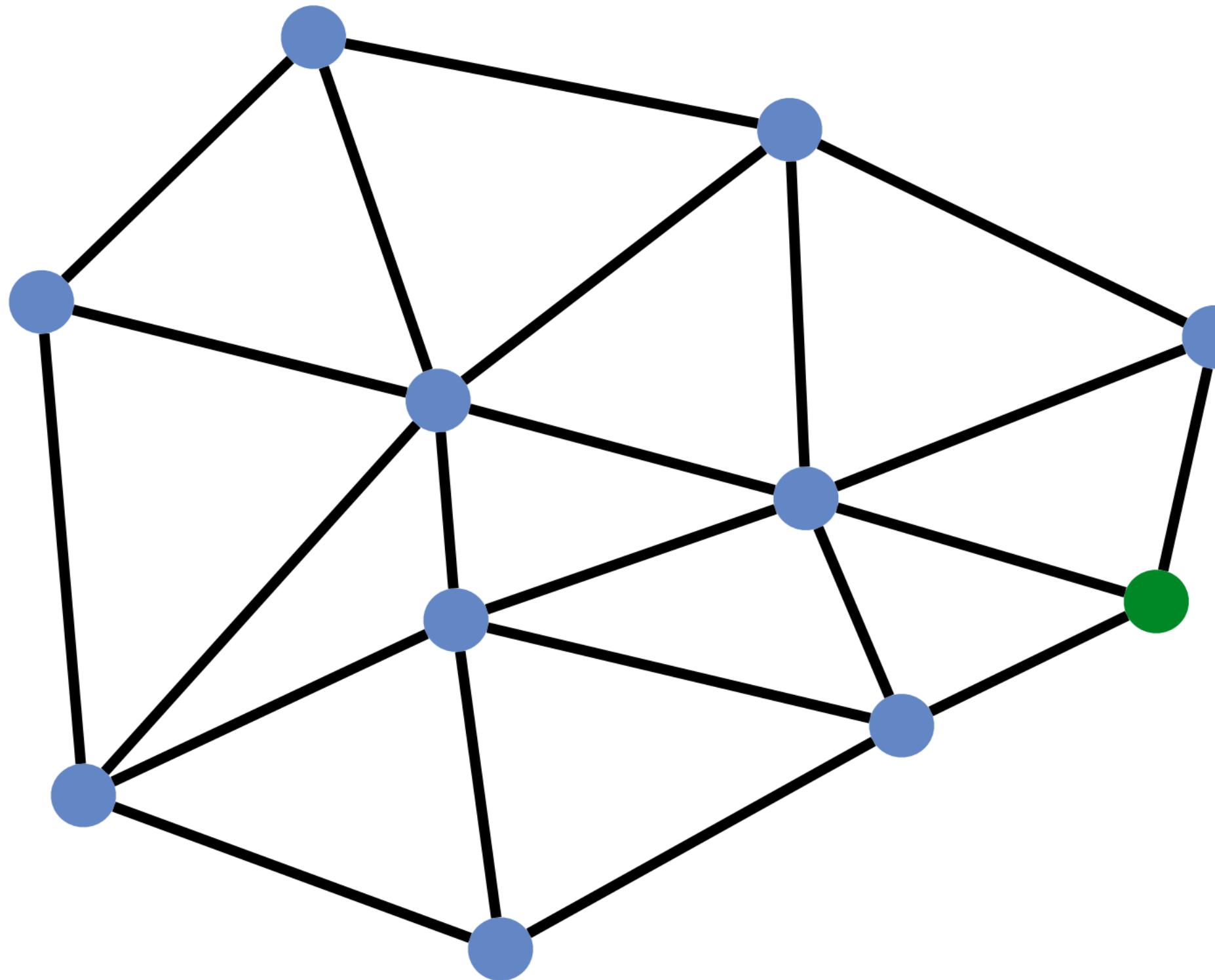
- ▶ O grafo resultante é uma árvore de dispersão (*spanning tree*) com o destino como vértice *sink*
 - ▶ árvore = sem *loops*
 - ▶ *spanning tree* = sem *dead ends*

EXEMPLO 1



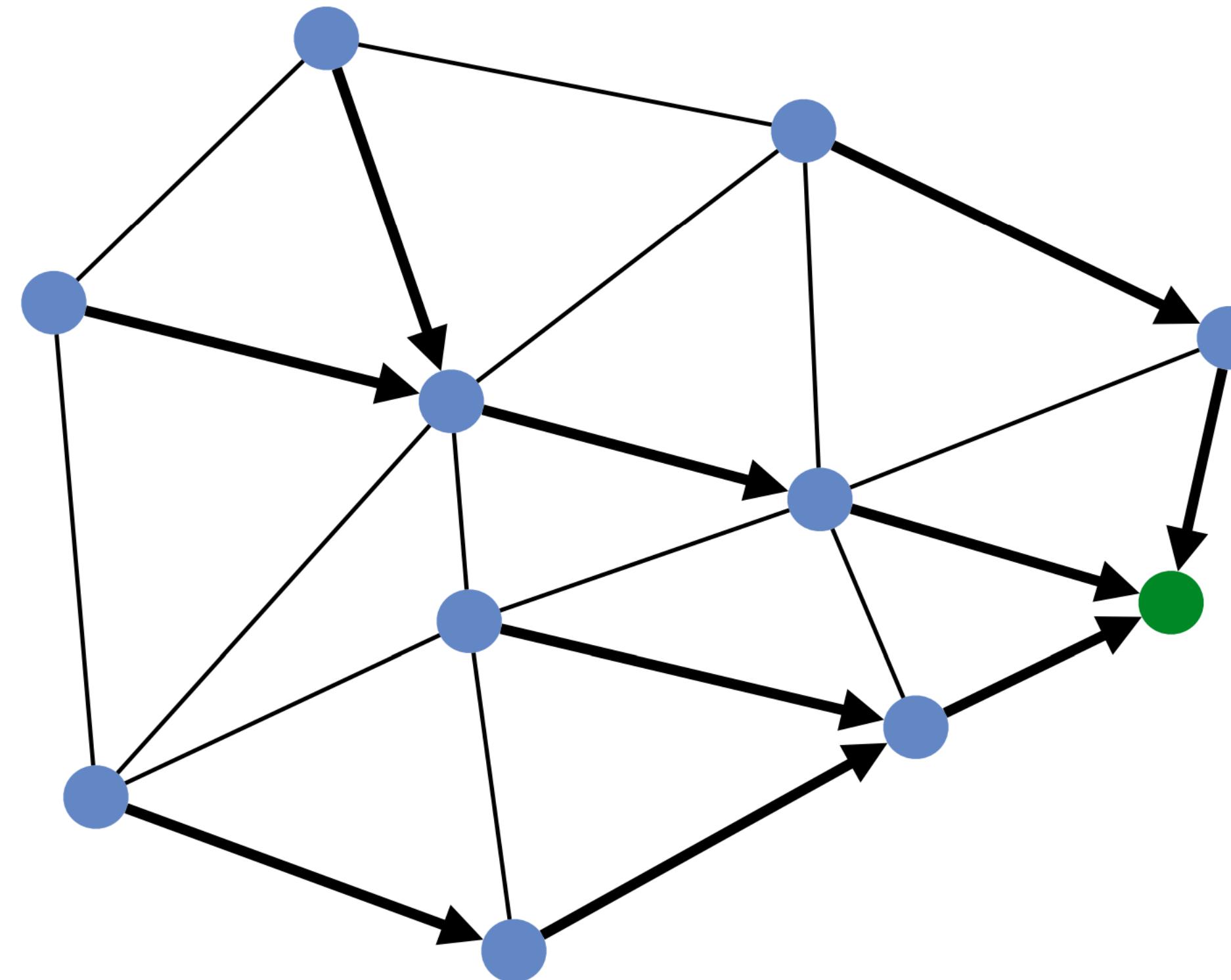
EXEMPLO 1

Selecione um destino



EXEMPLO 1

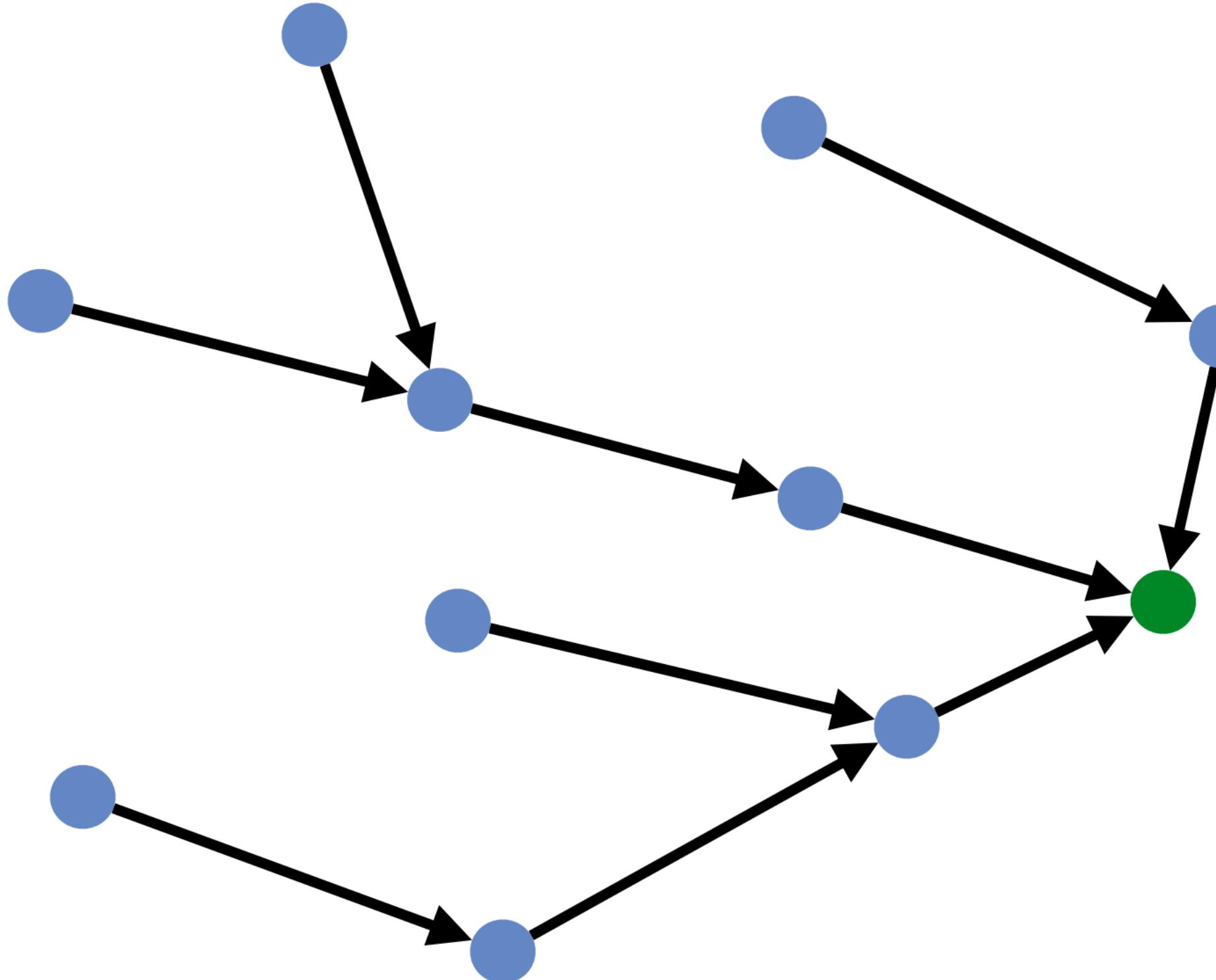
Setas direcionadas nos *links* de saída



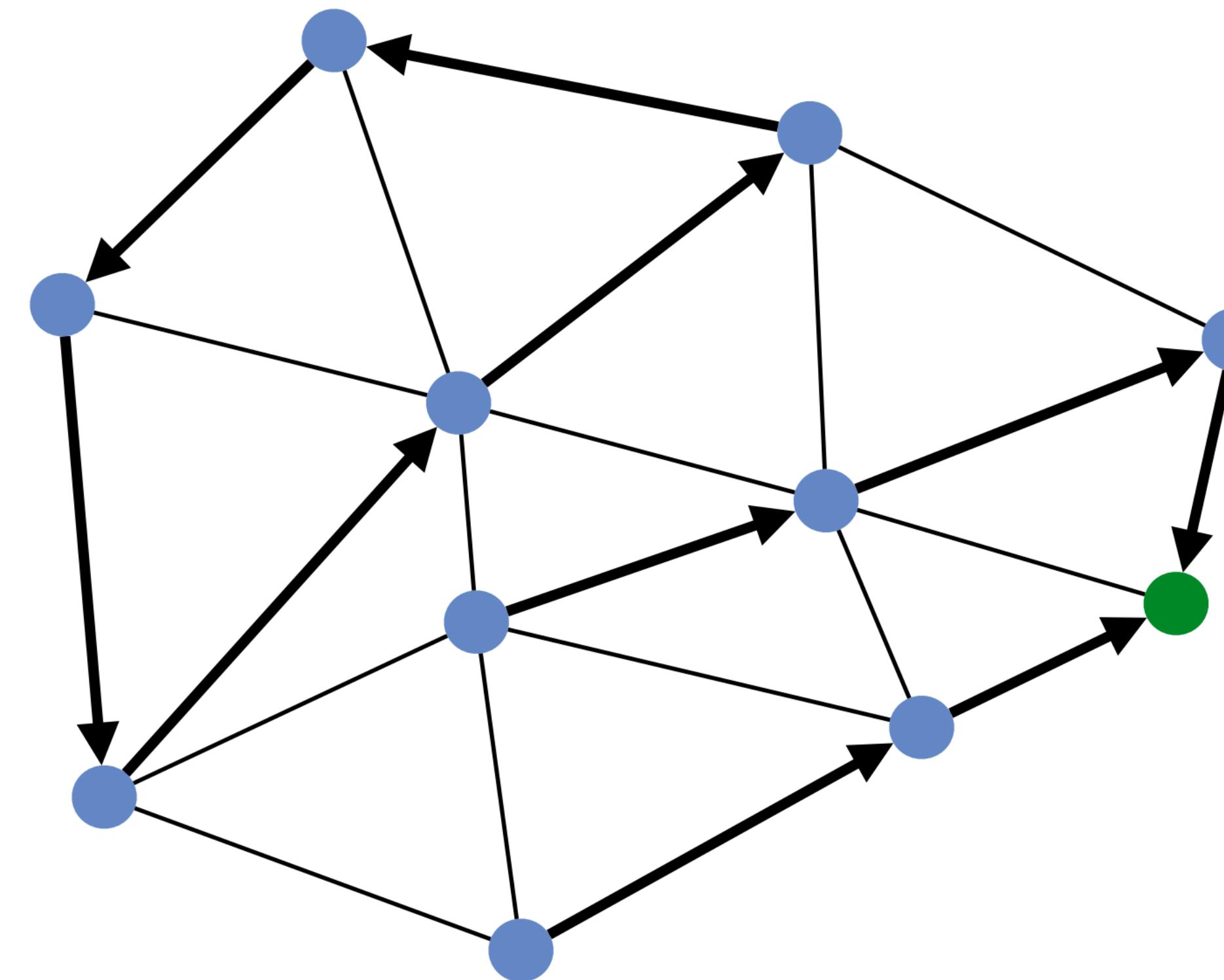
EXEMPLO 1

Remover *links* inutilizados

Resulta em *spanning tree*: válido!

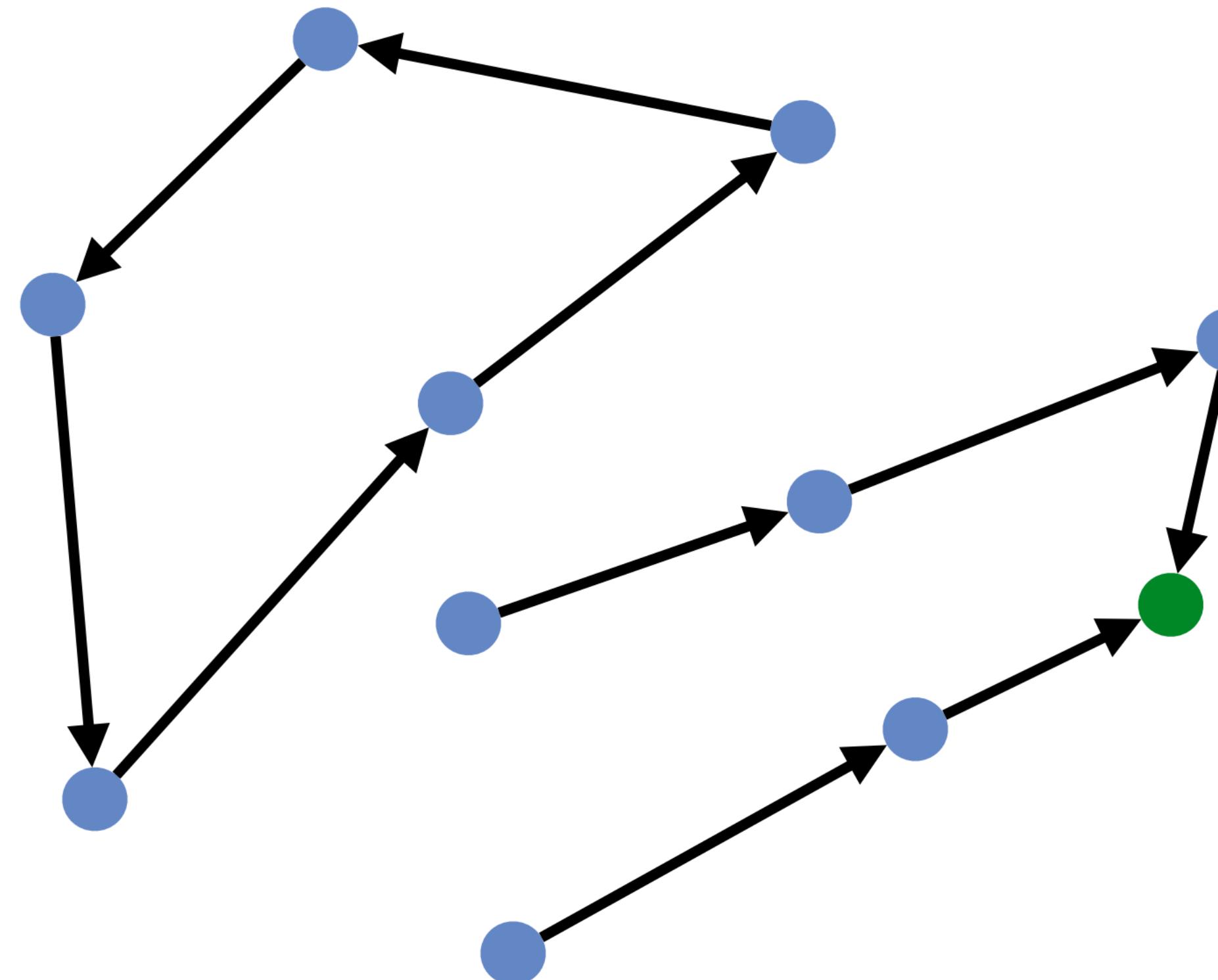


EXEMPLO 2

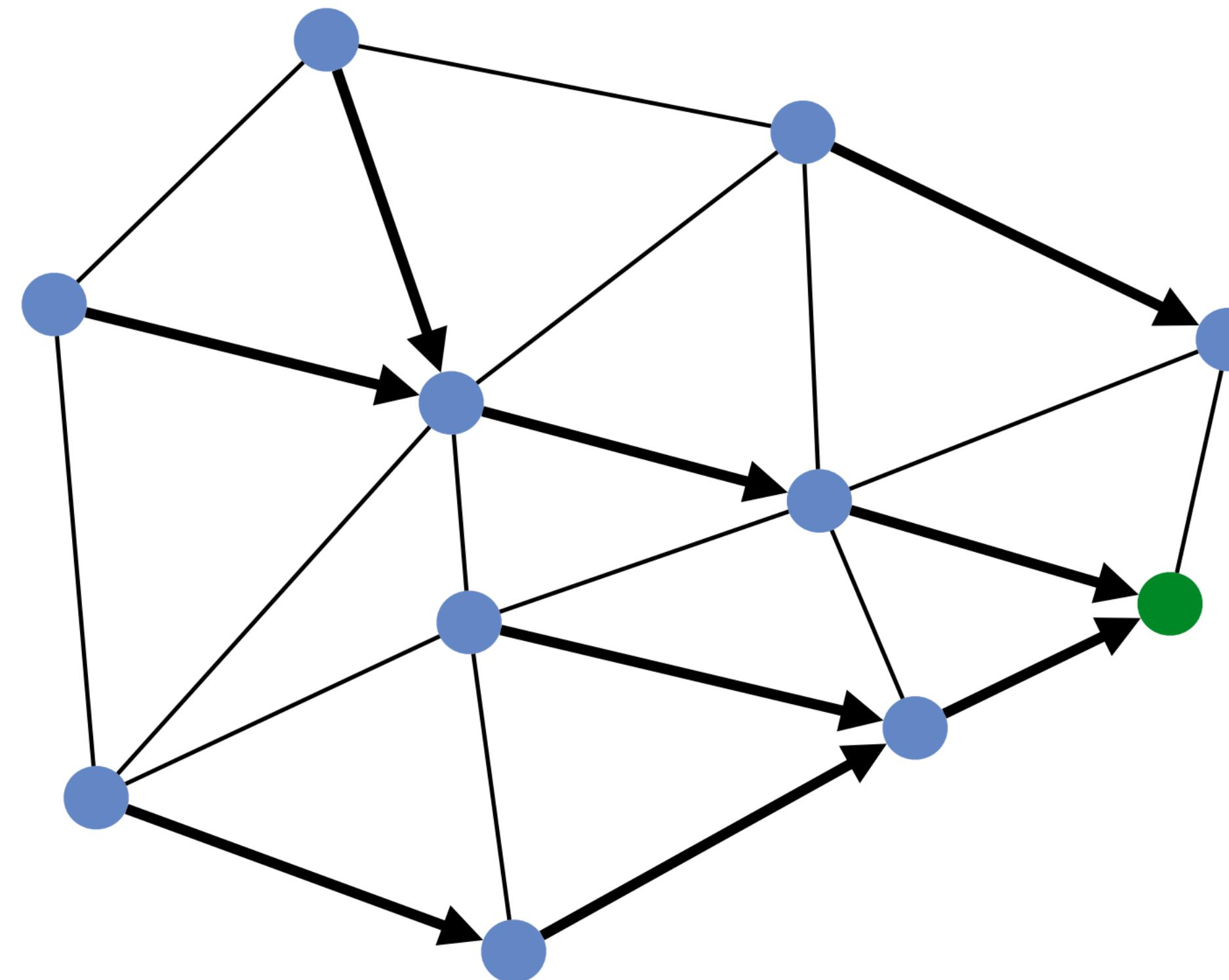


EXEMPLO 2

Isso é válido?

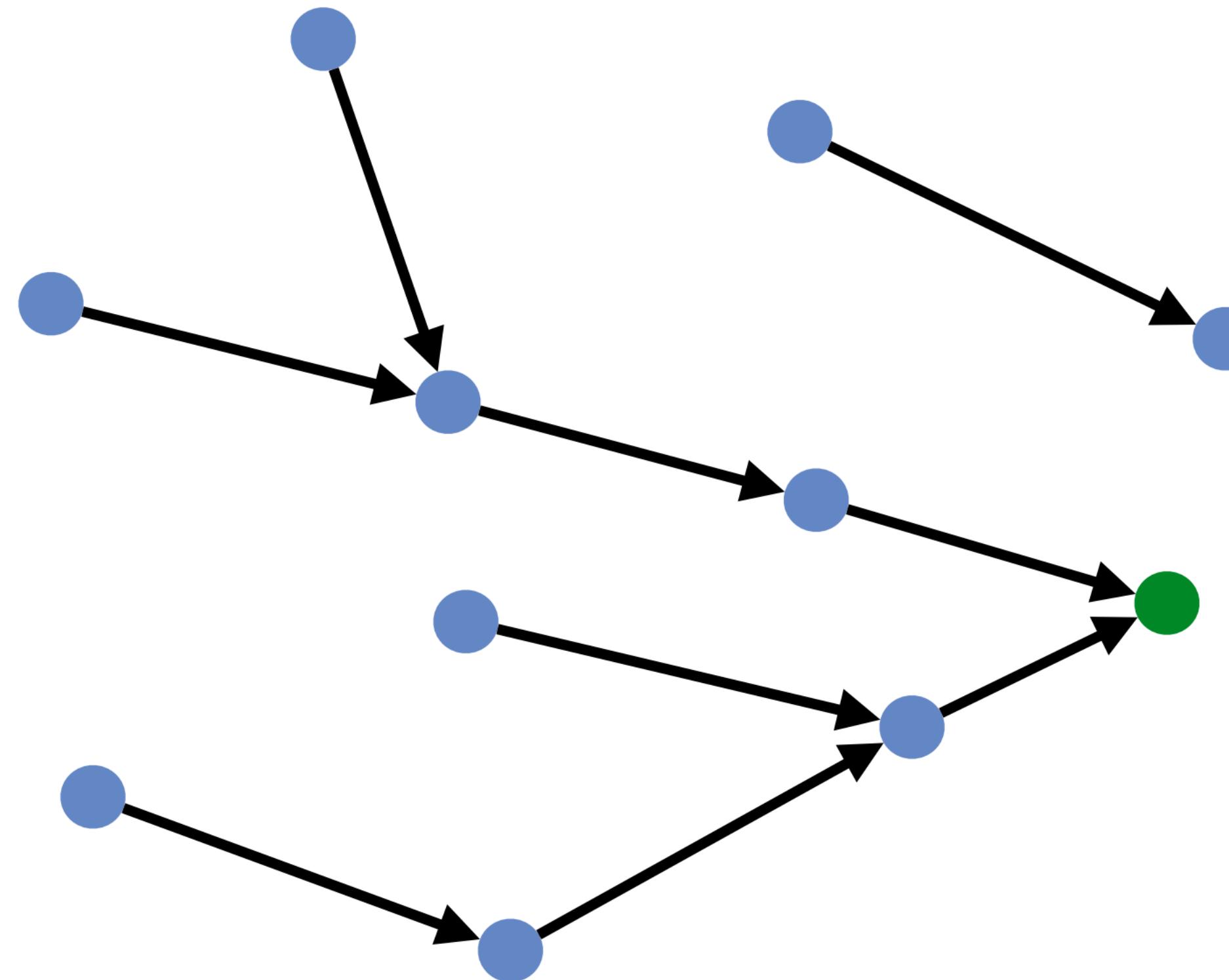


EXEMPLO 3



EXEMPLO 3

Isso é válido?



DUAS QUESTÕES:

Como podemos **verificar** que um dado estado de roteamento é válido?

Como podemos **produzir** um estado de roteamento válido?

CRIANDO UM ESTADO DE ROTEAMENTO VÁLIDO

- ▶ É fácil evitar *dead ends*
- ▶ Não é fácil evitar *loops*
- ▶ A principal diferença entre os protocolos de roteamento é como eles evitam *loops*