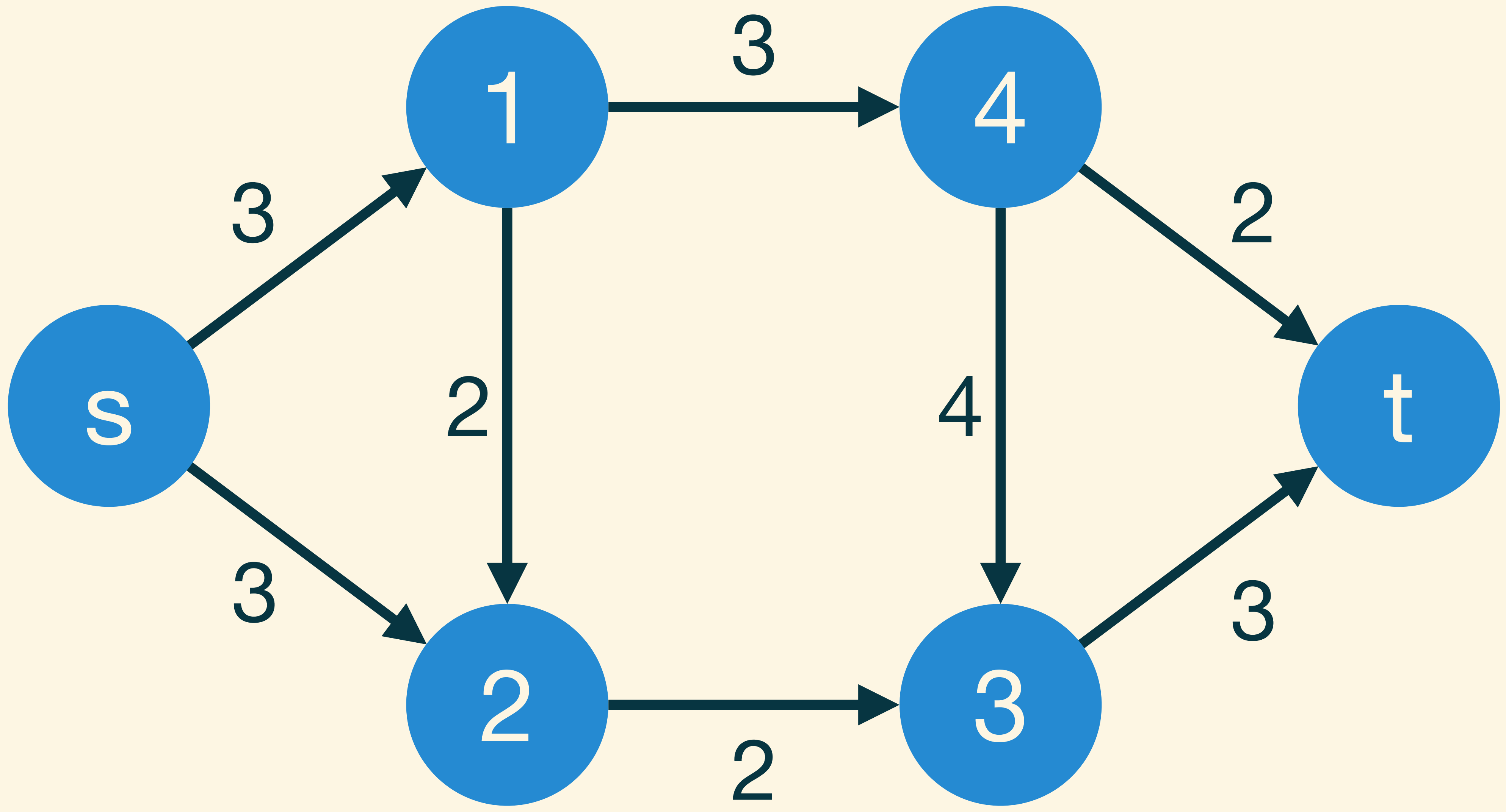
Dinic’s Algorithm

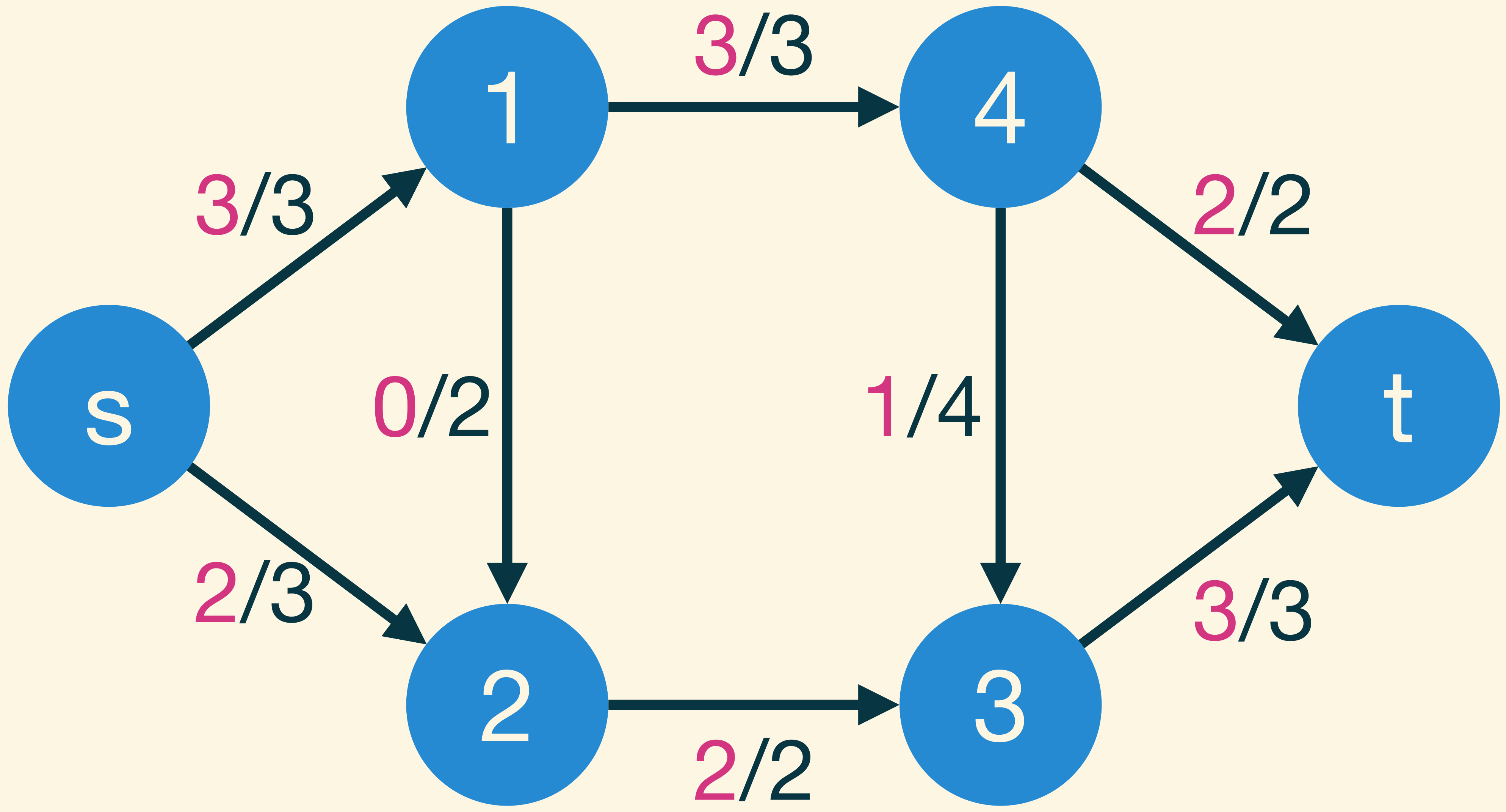
郭至軒（KuoE0）

KuoE0.tw@gmail.com
KuoE0.ch

Maximum Flow

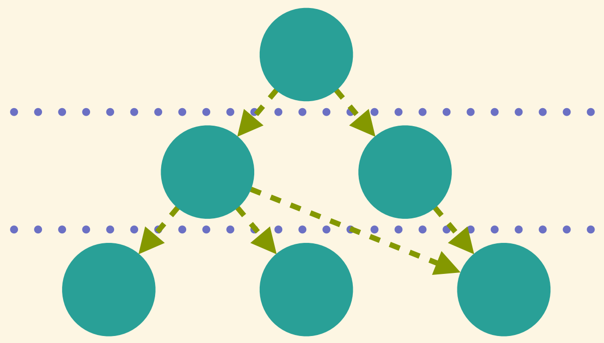


Maximum Flow

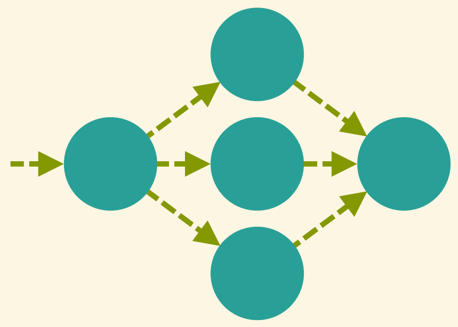


Concept

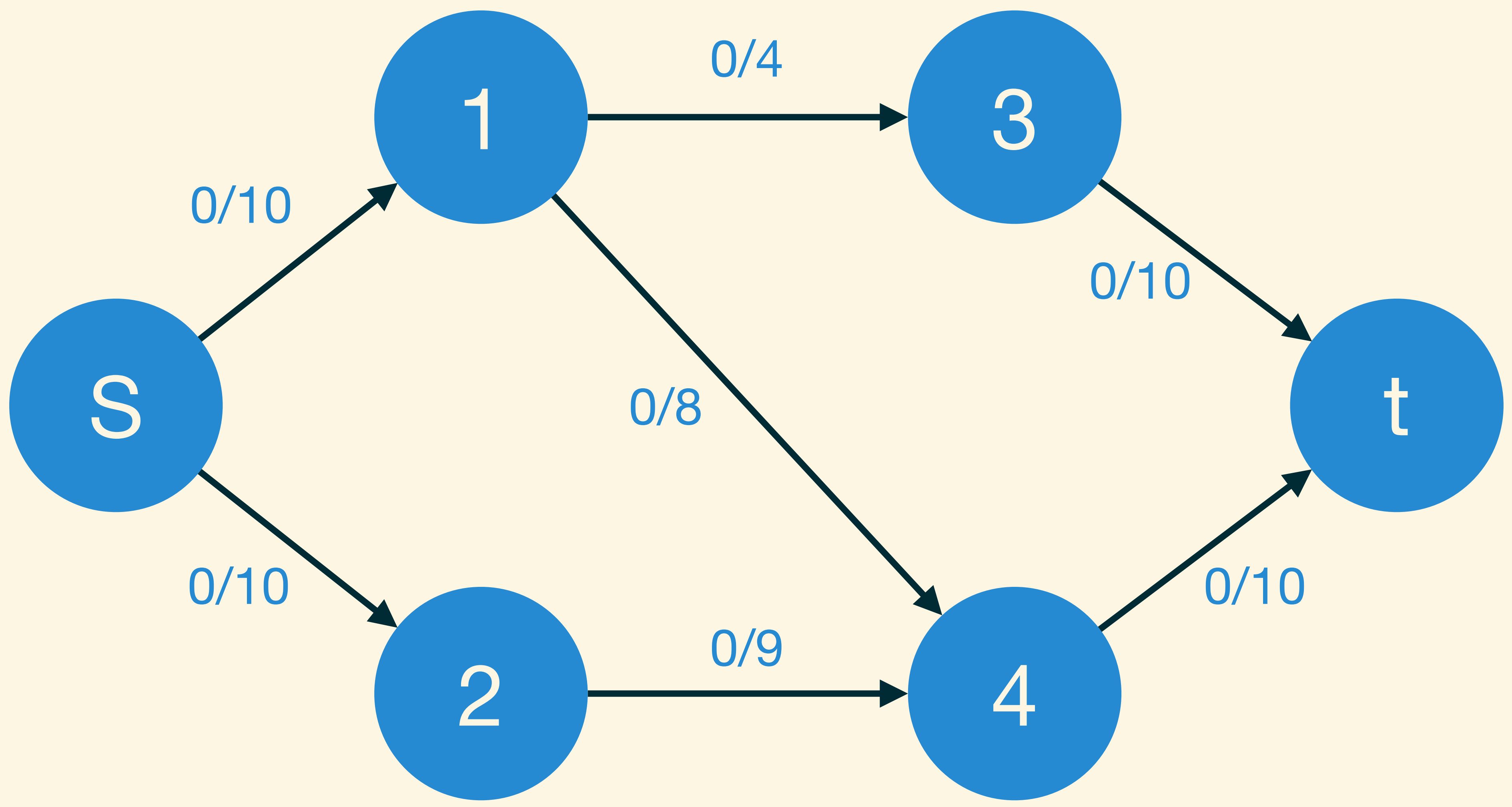
Level Graph



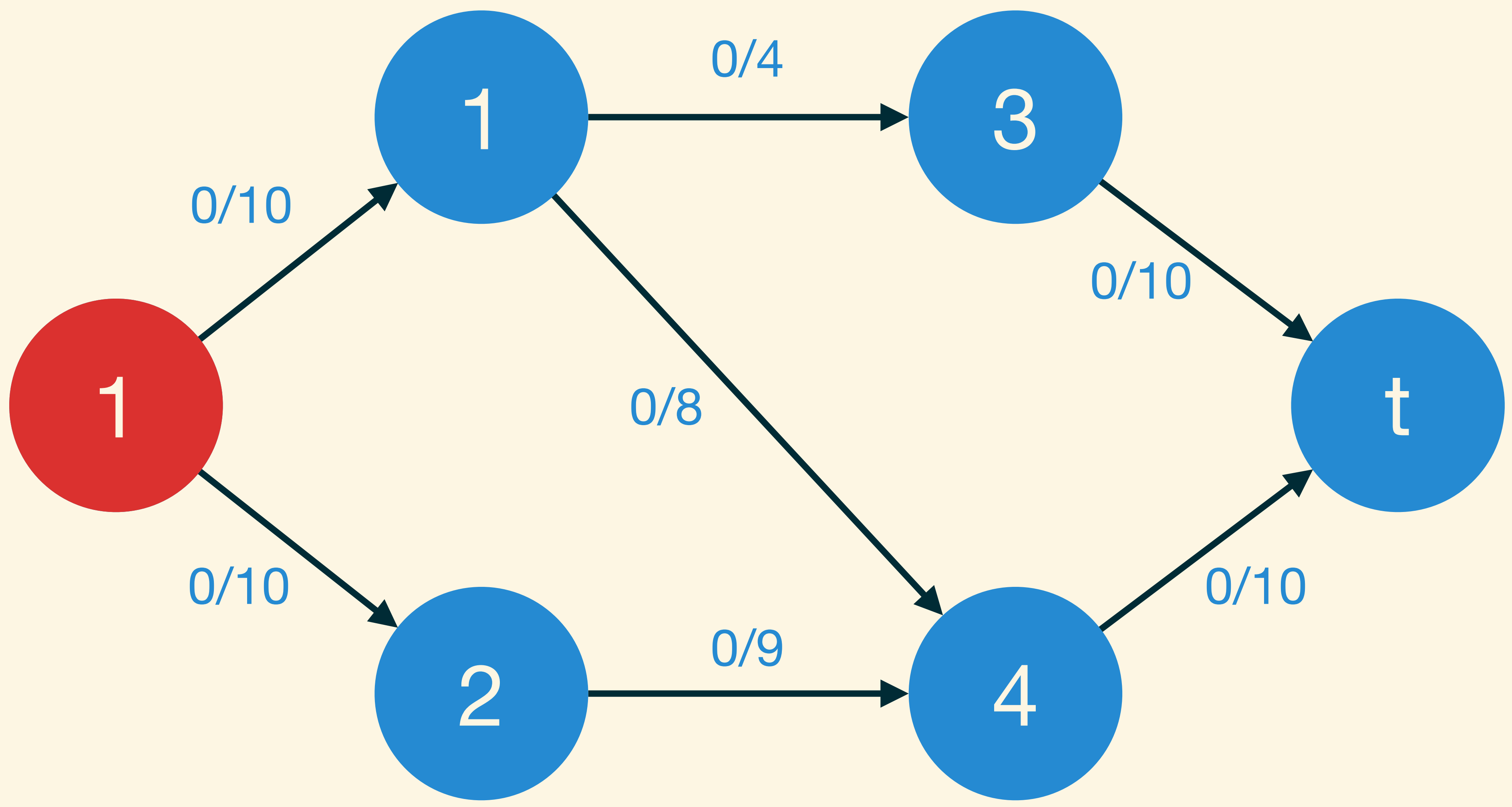
Blocking Flow



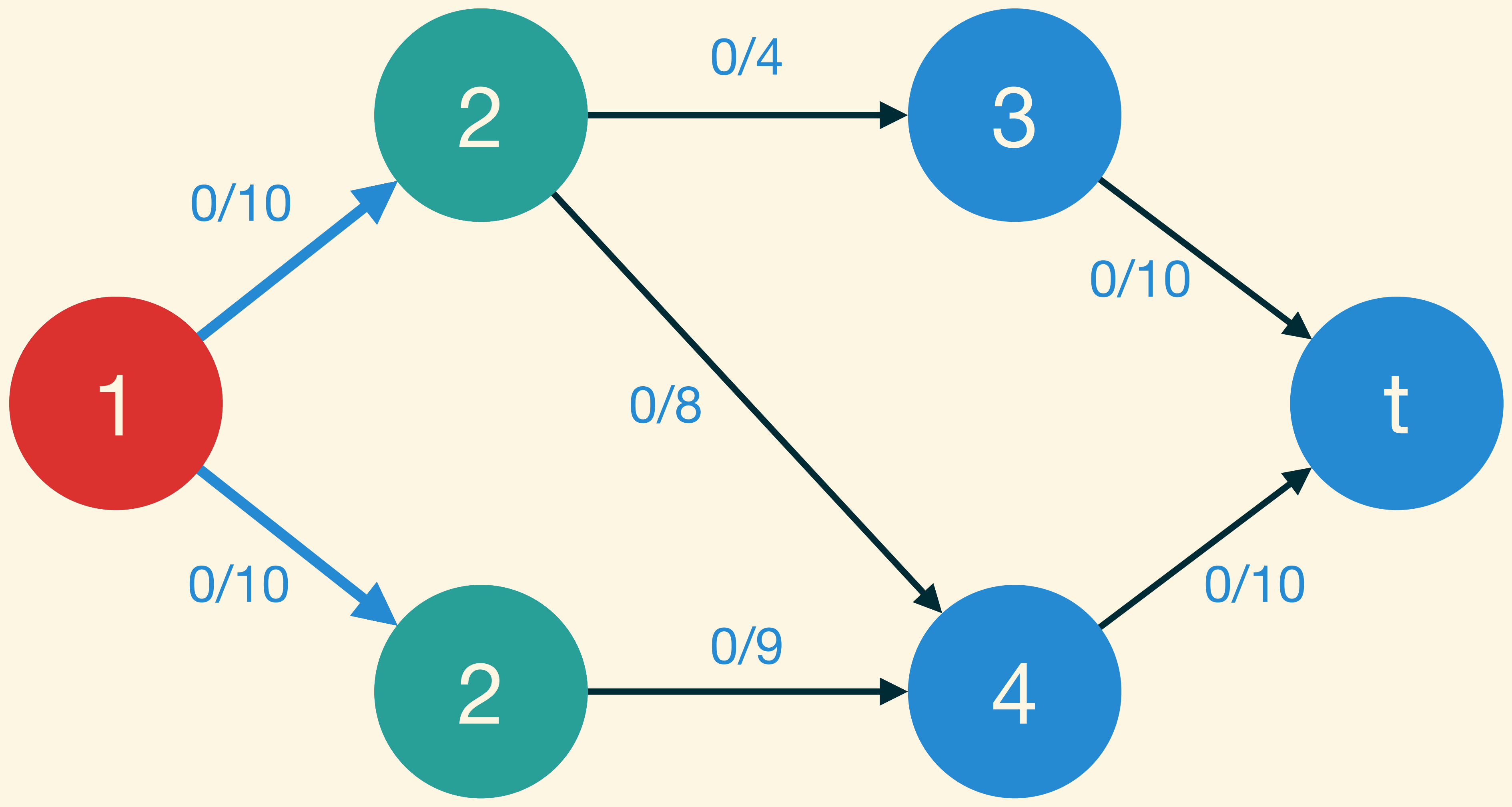
Level Graph



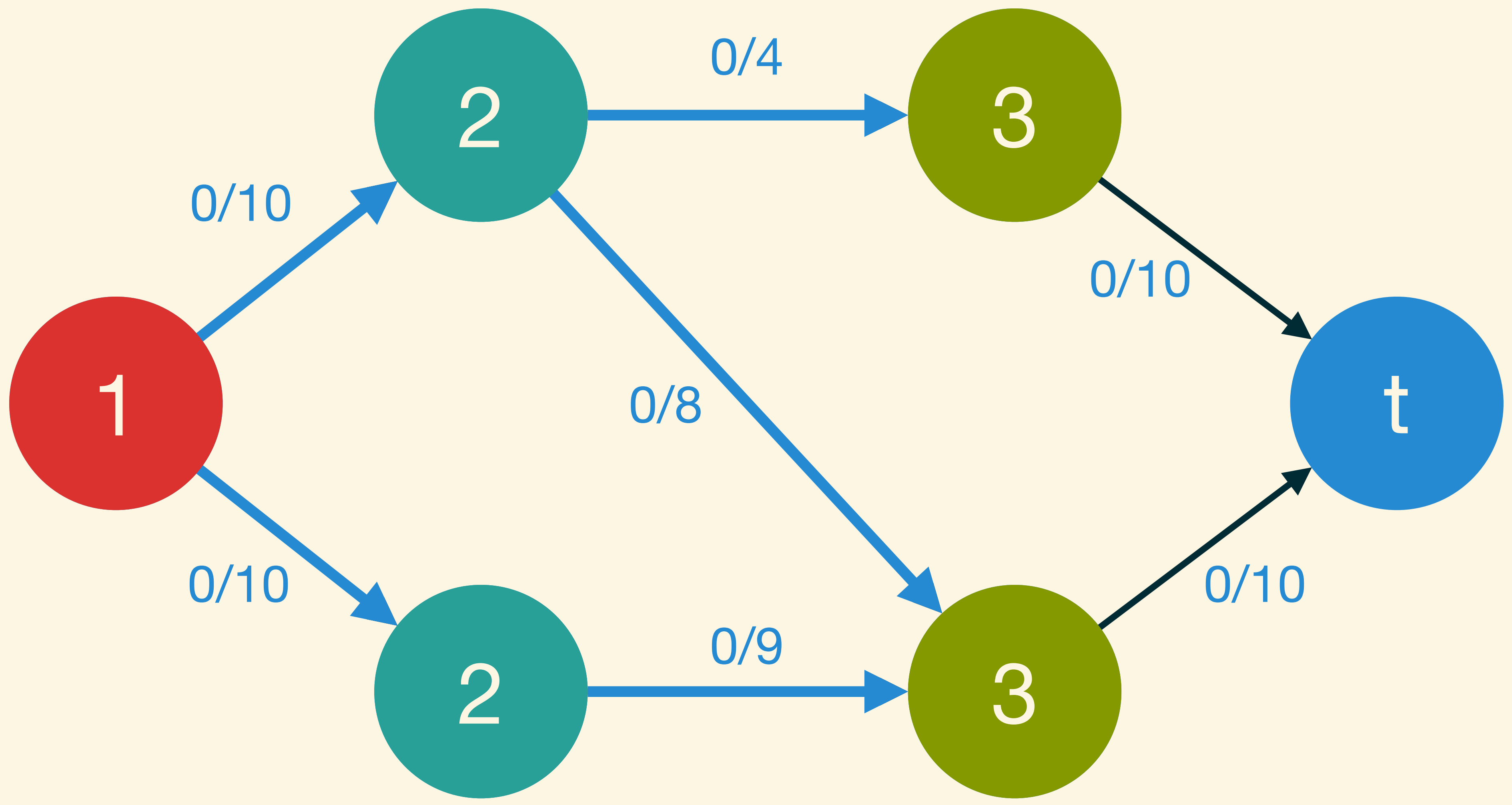
Level Graph



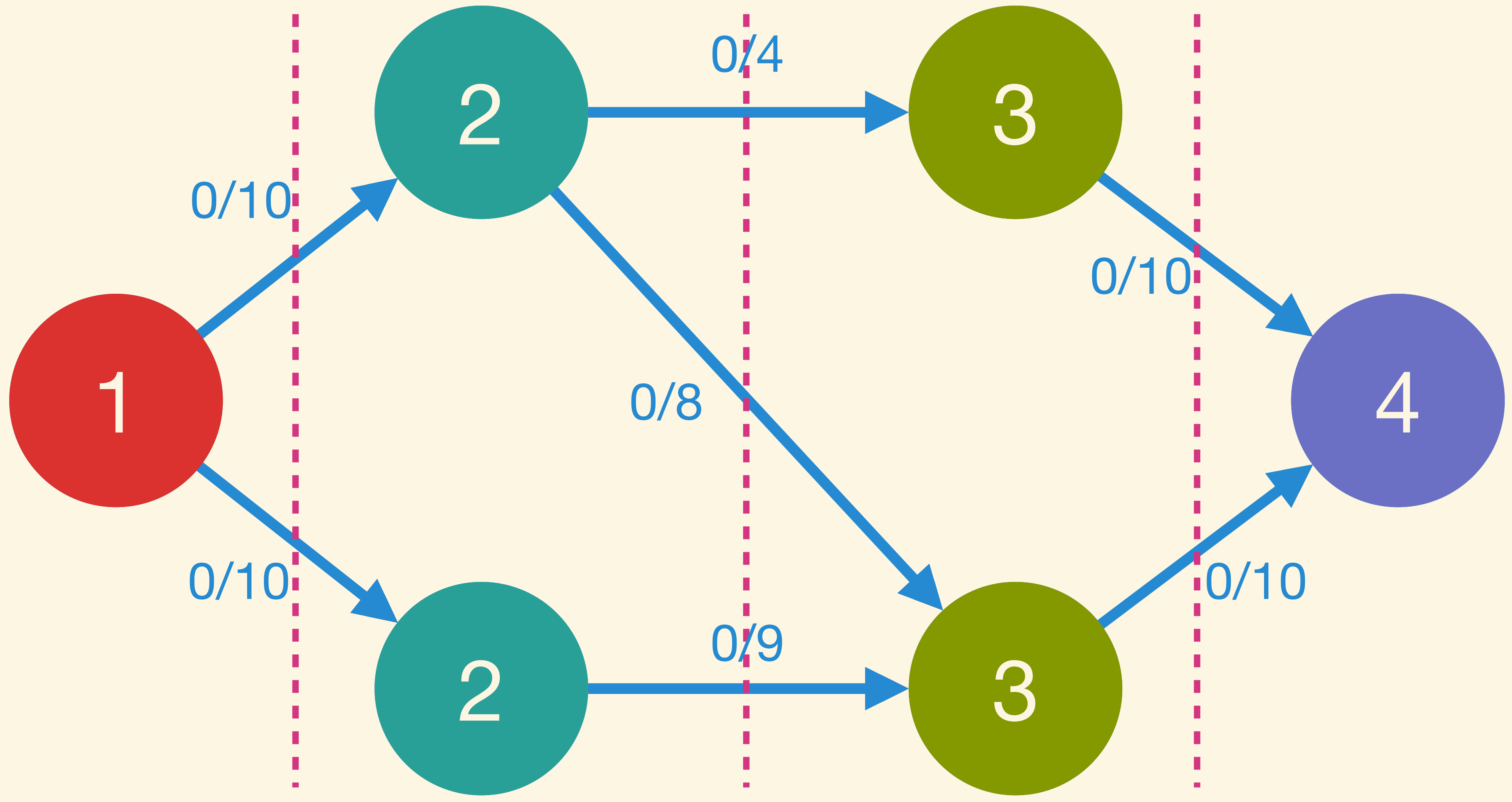
Level Graph



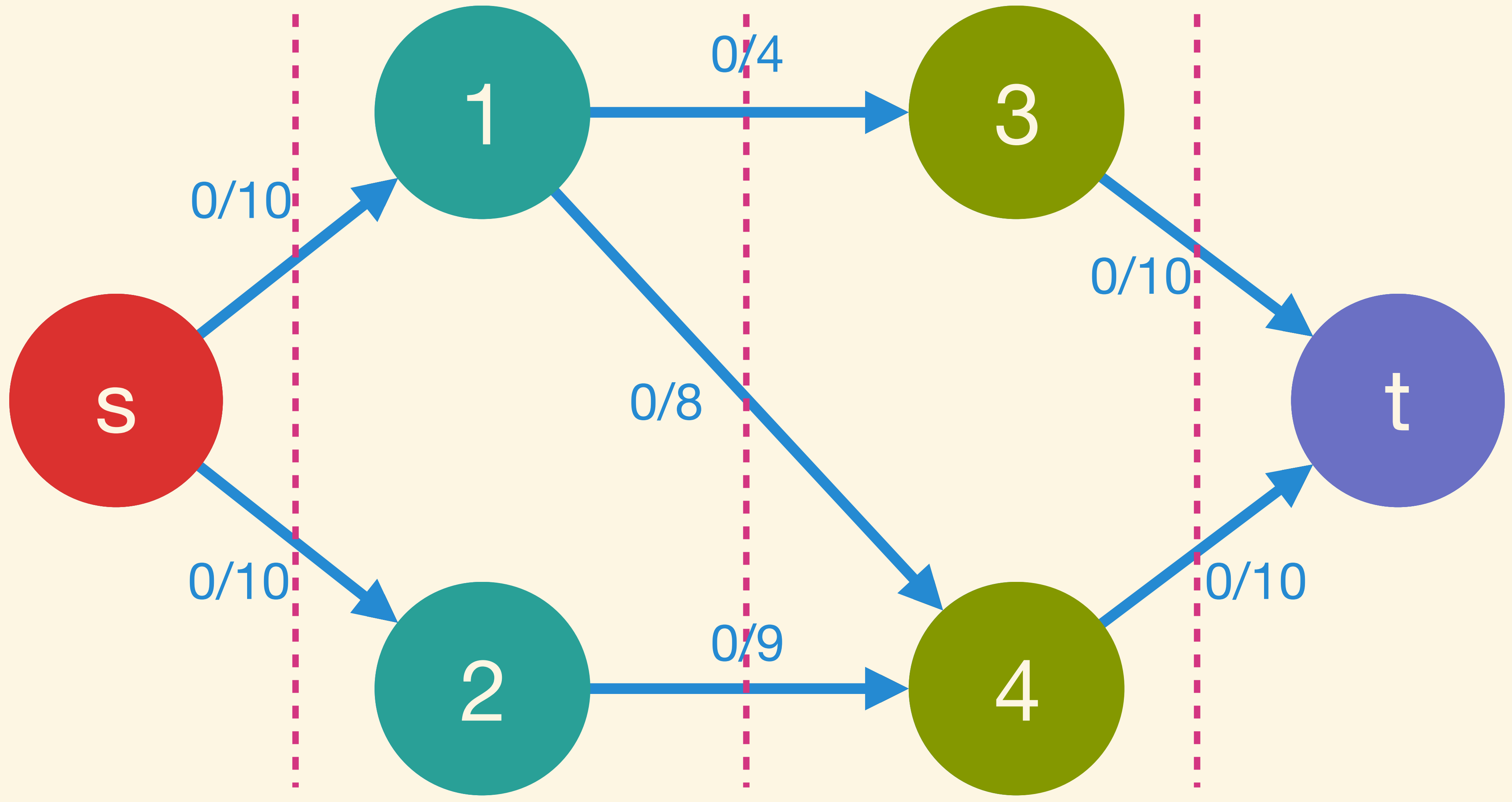
Level Graph



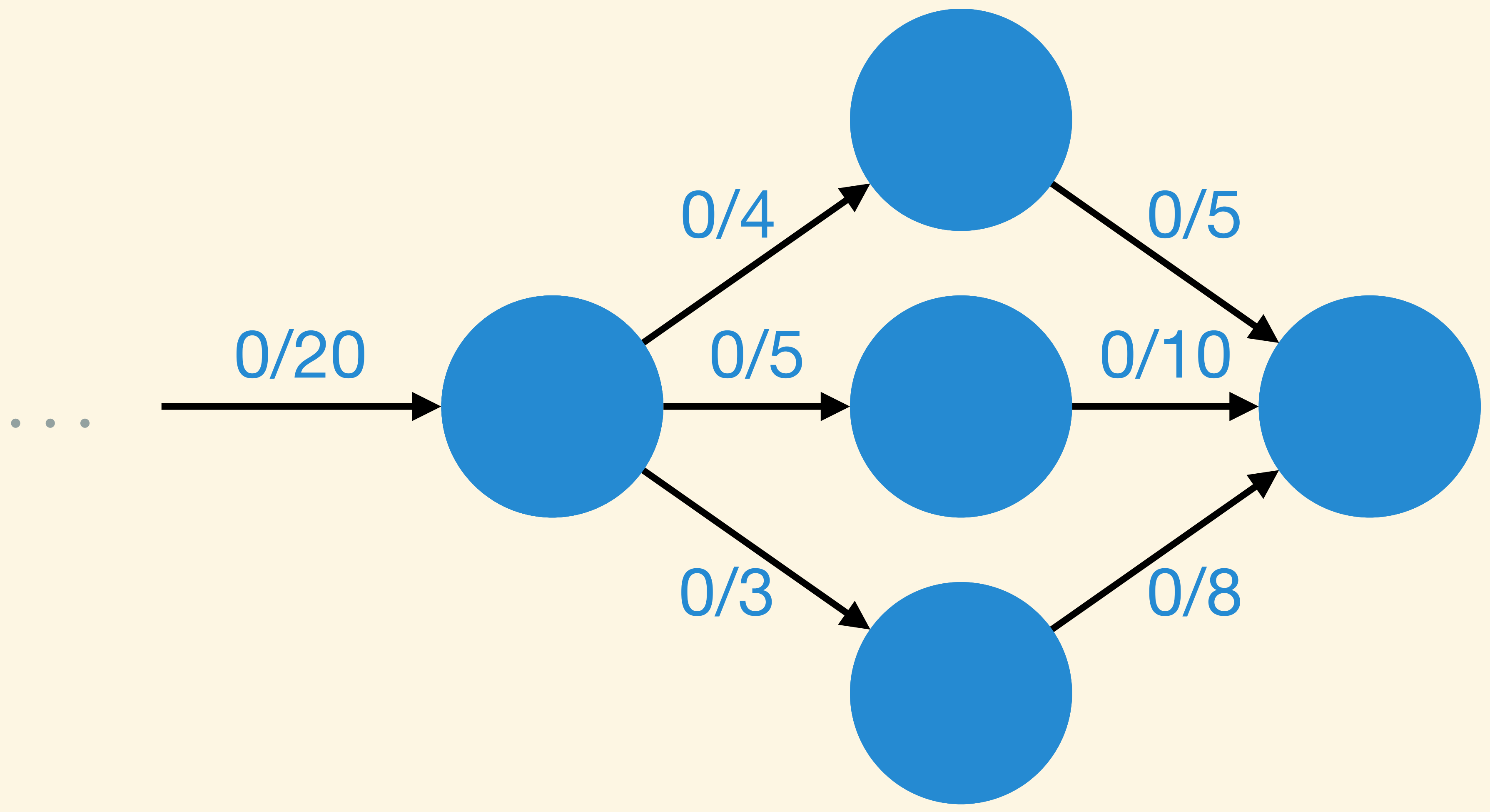
Level Graph



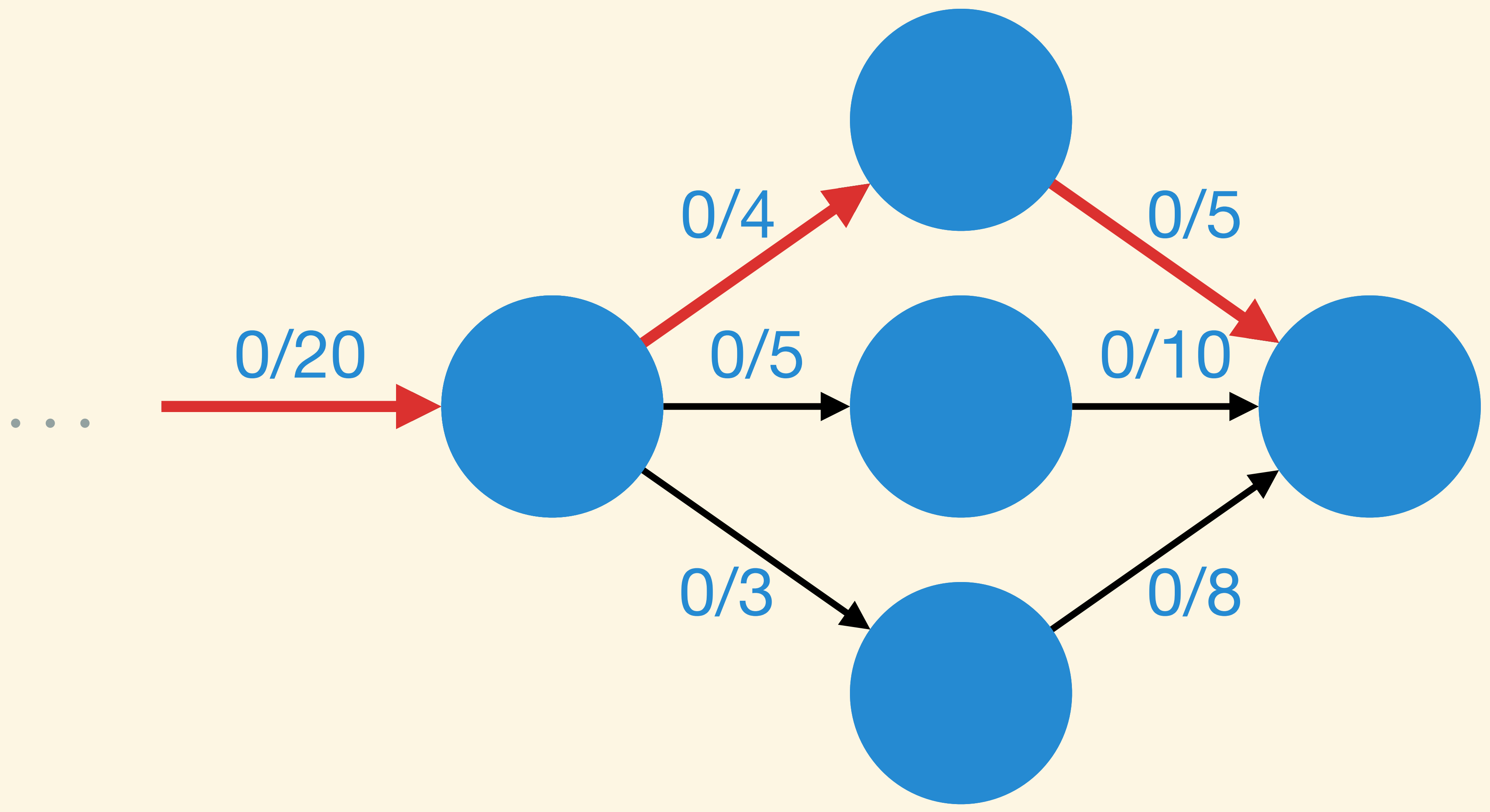
Level Graph



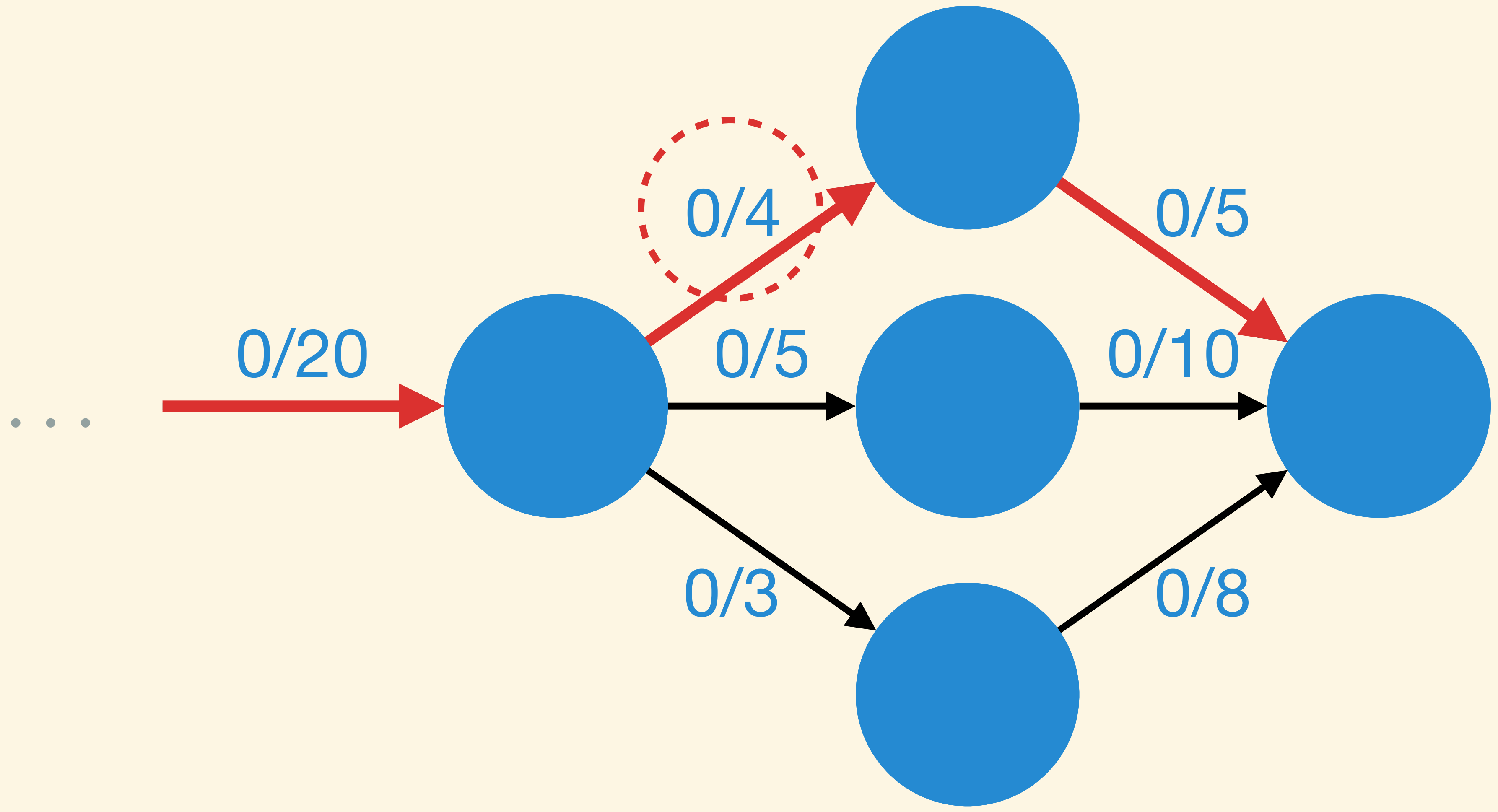
Blocking Flow



Blocking Flow

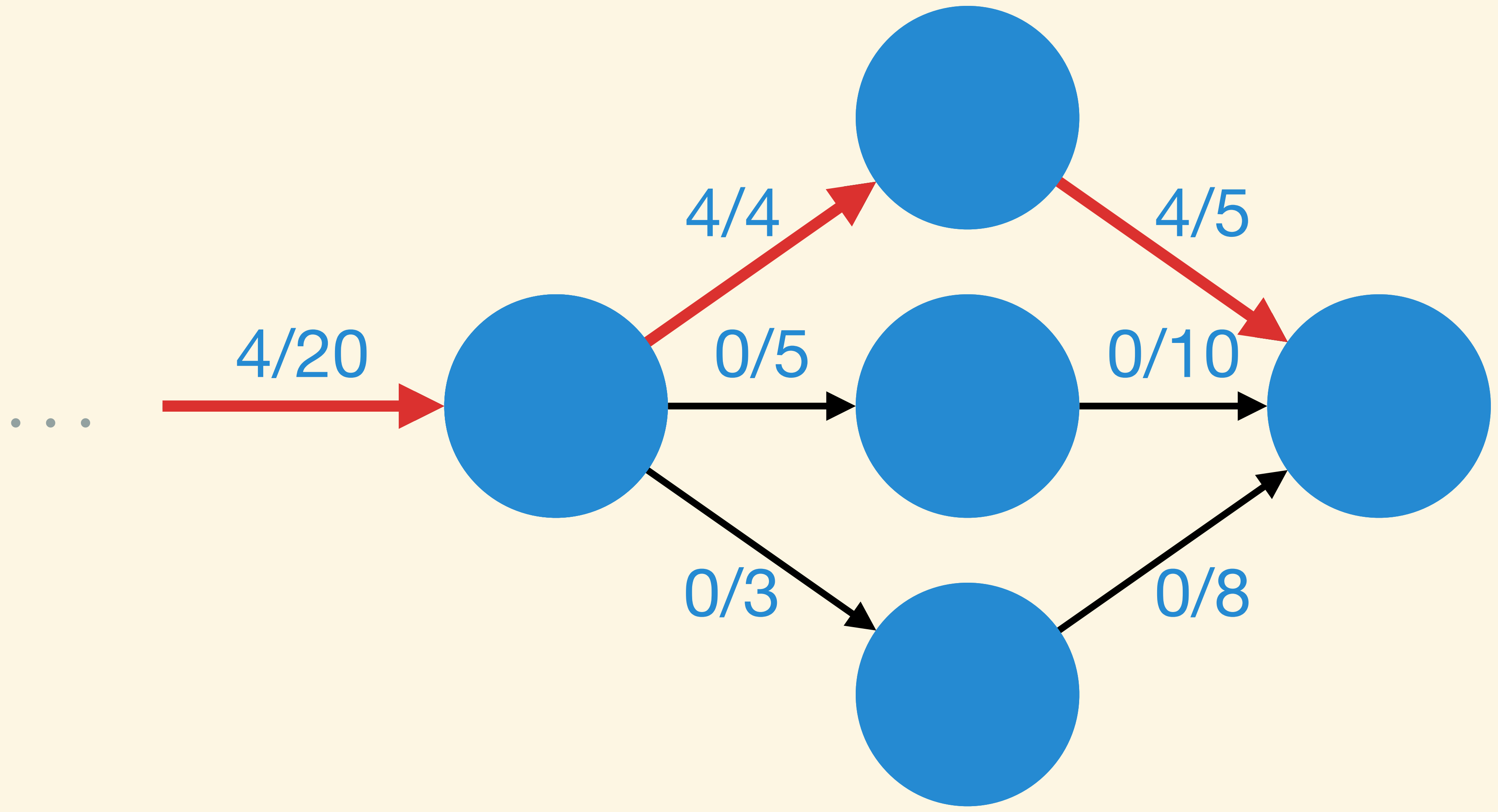


Blocking Flow

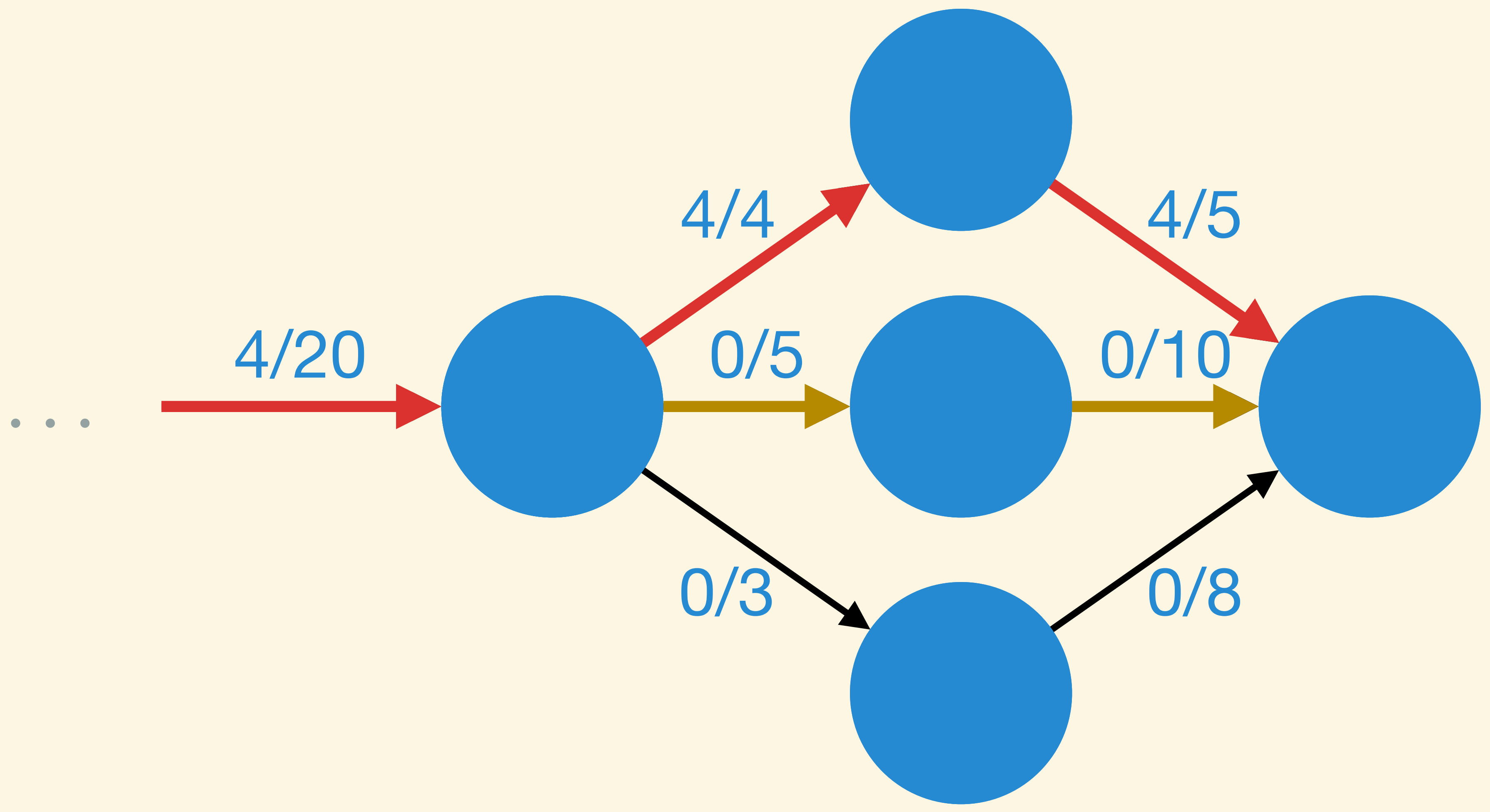


bottleneck: 4

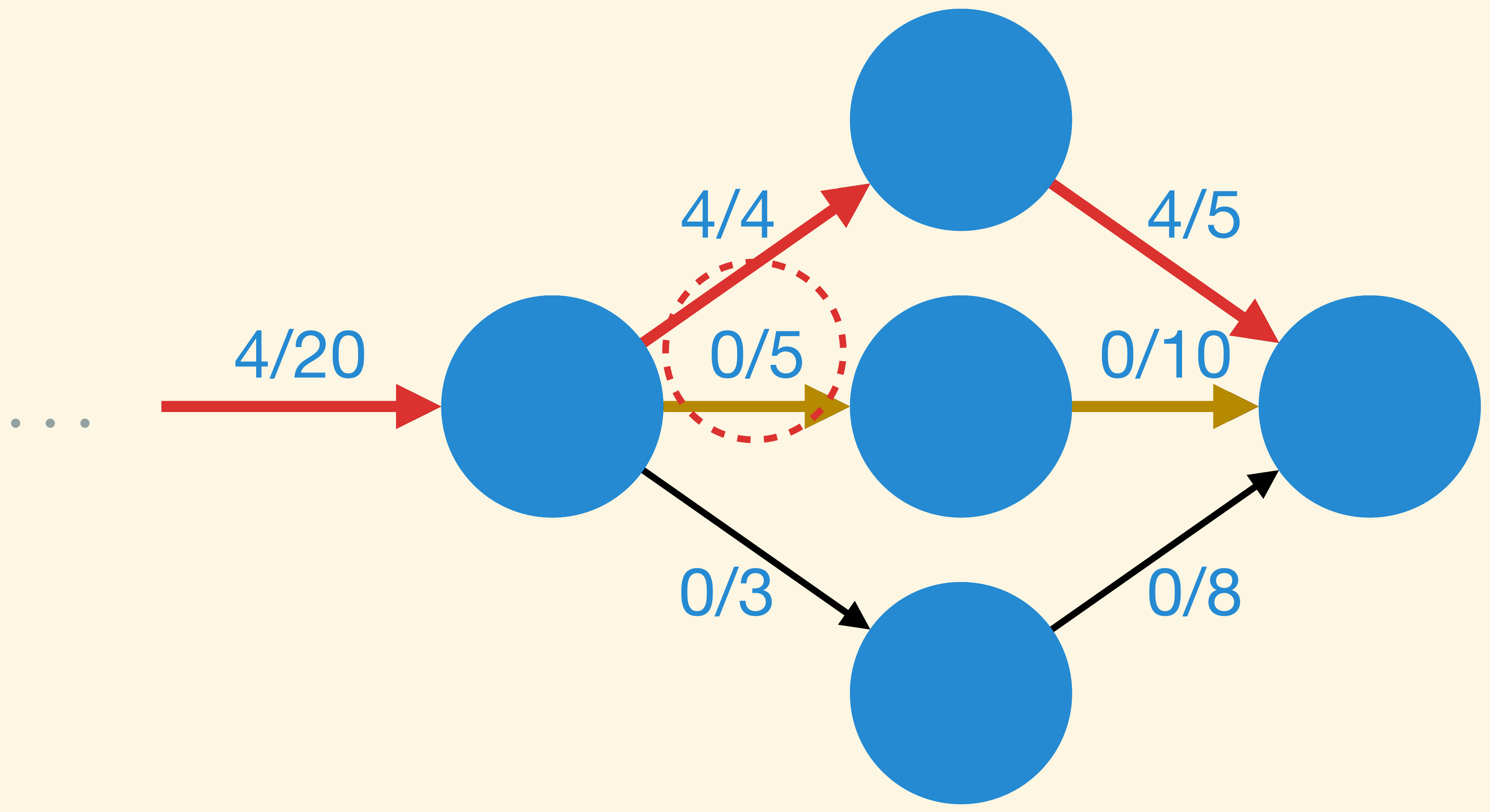
Blocking Flow



Blocking Flow

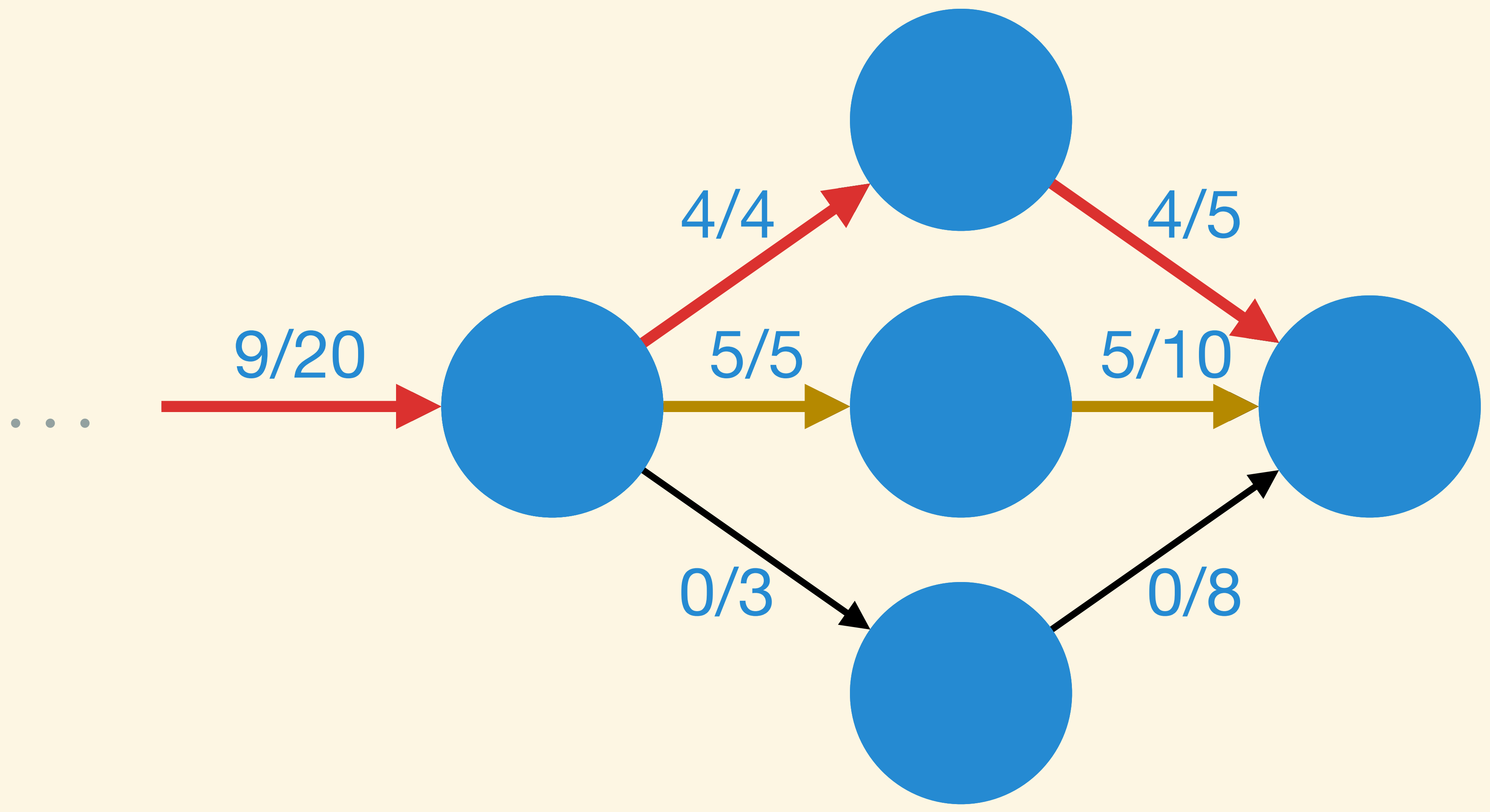


Blocking Flow

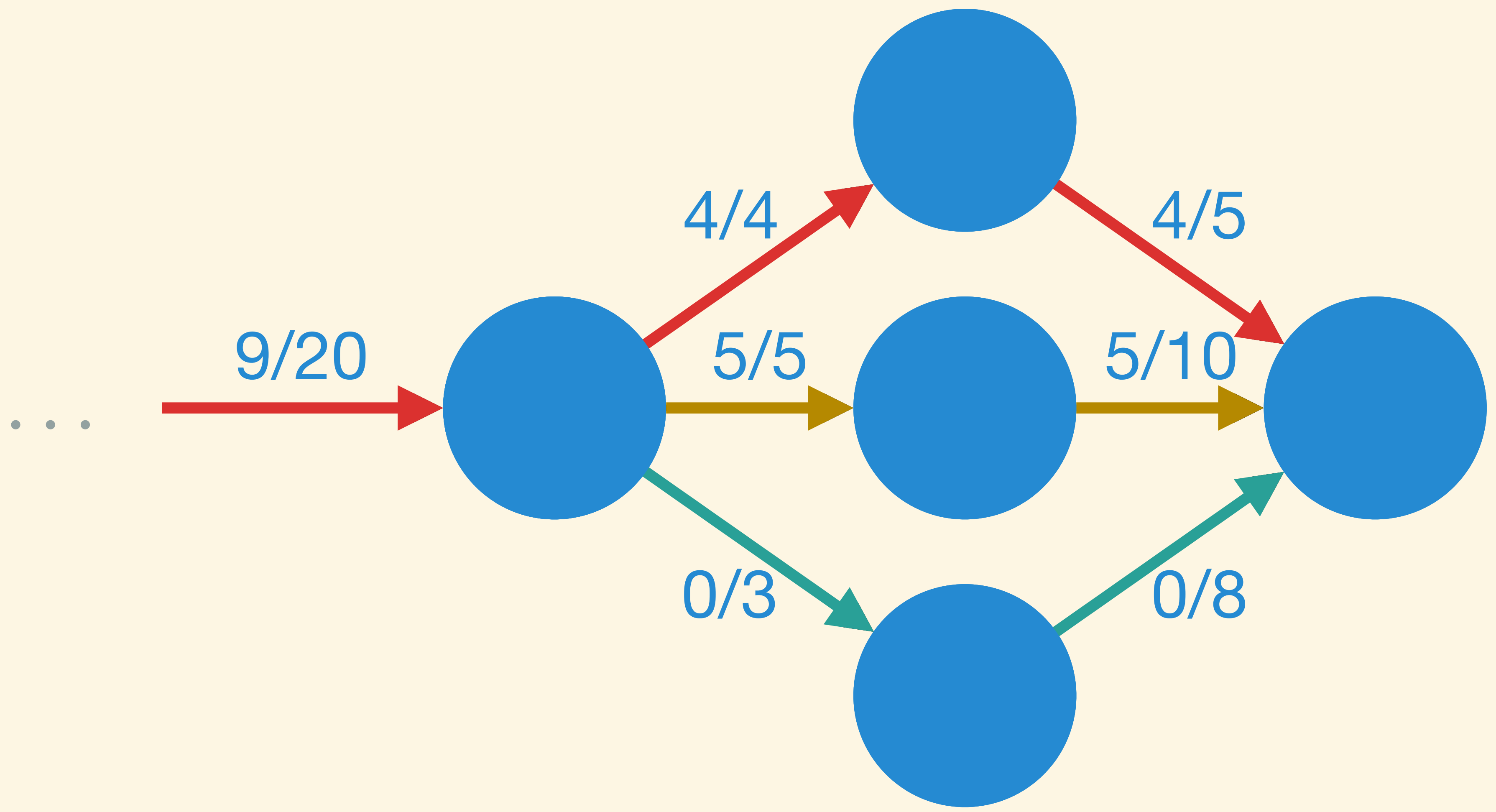


bottleneck: 5

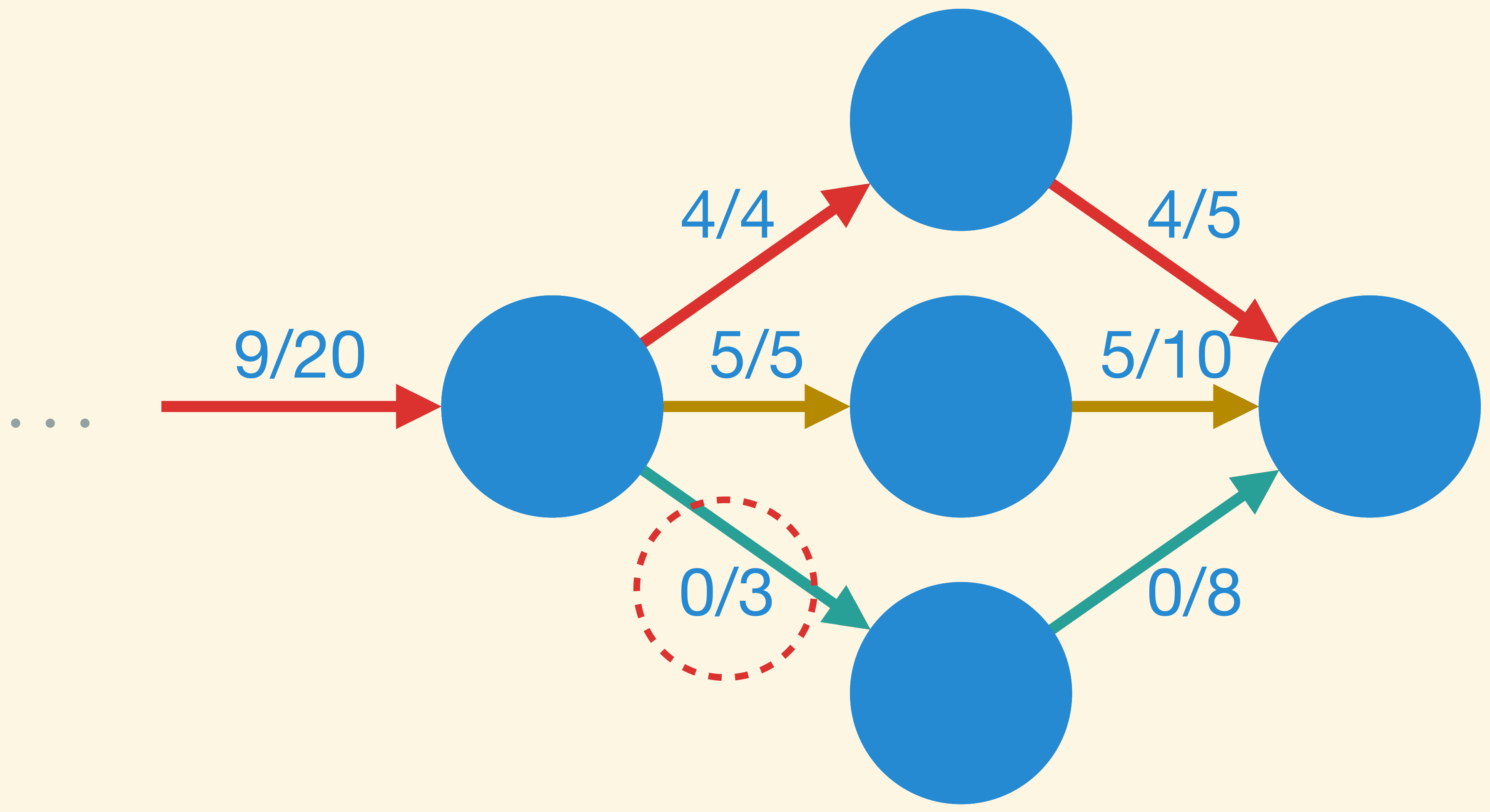
Blocking Flow



Blocking Flow

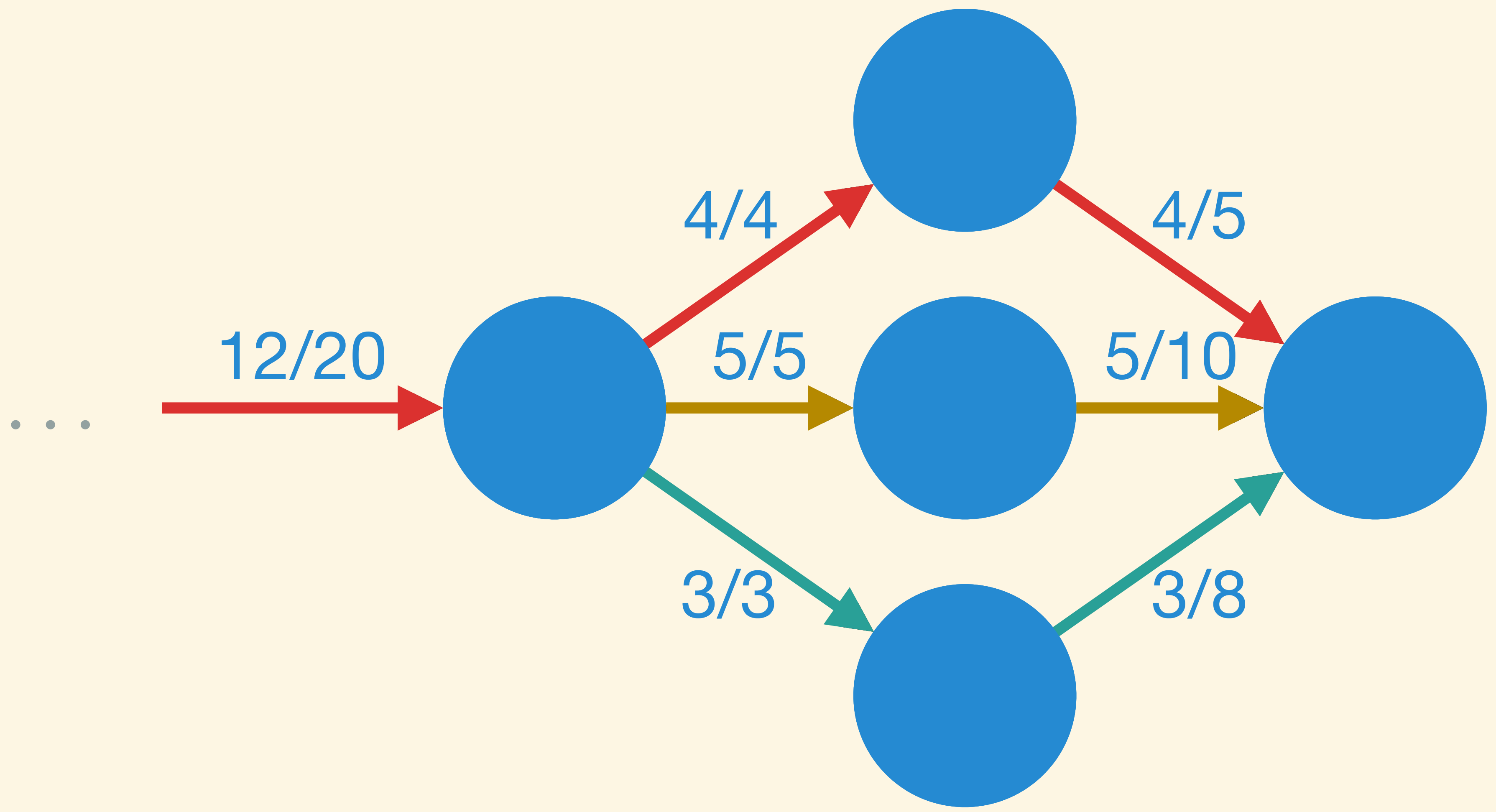


Blocking Flow

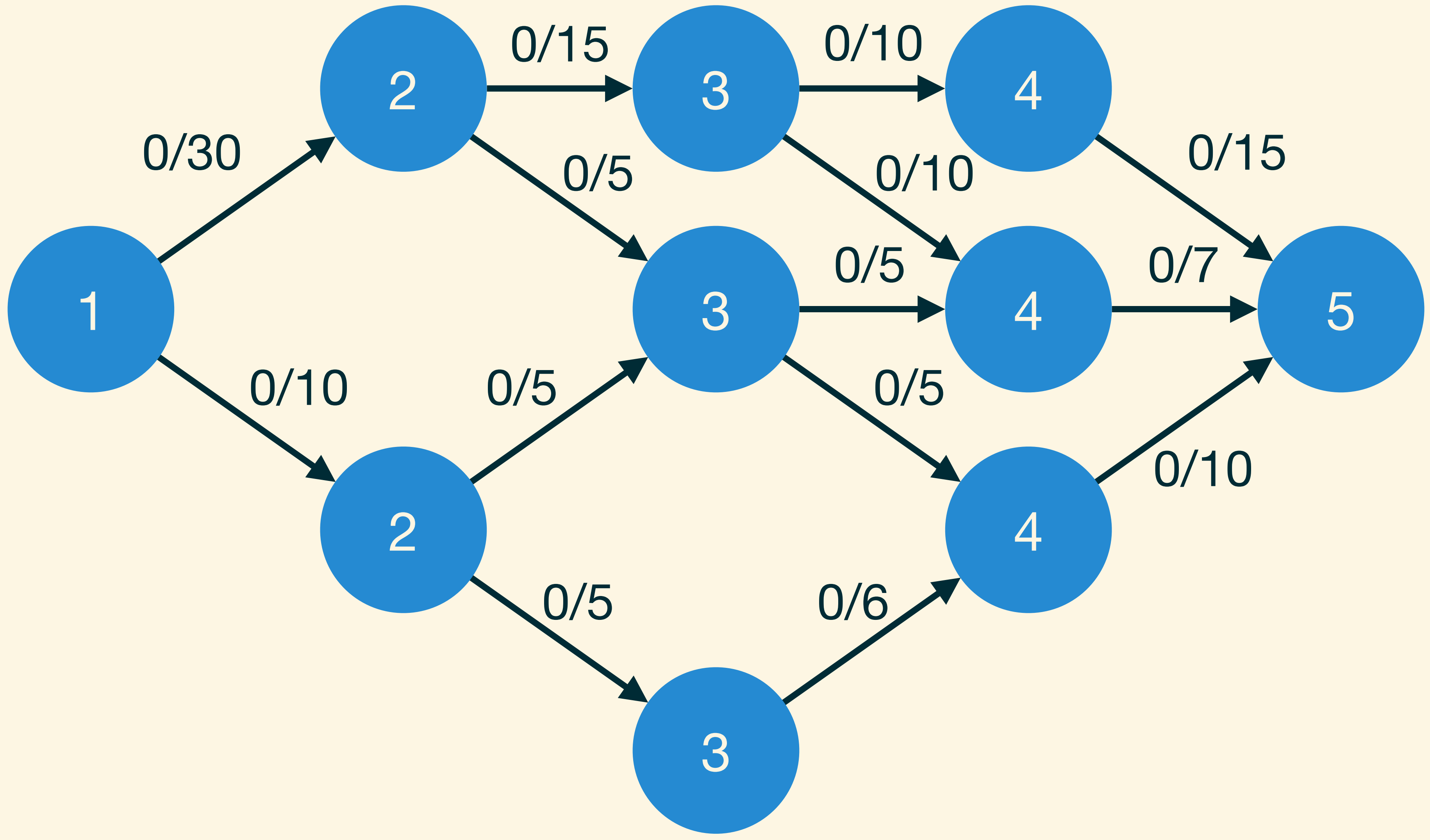


bottleneck: 3

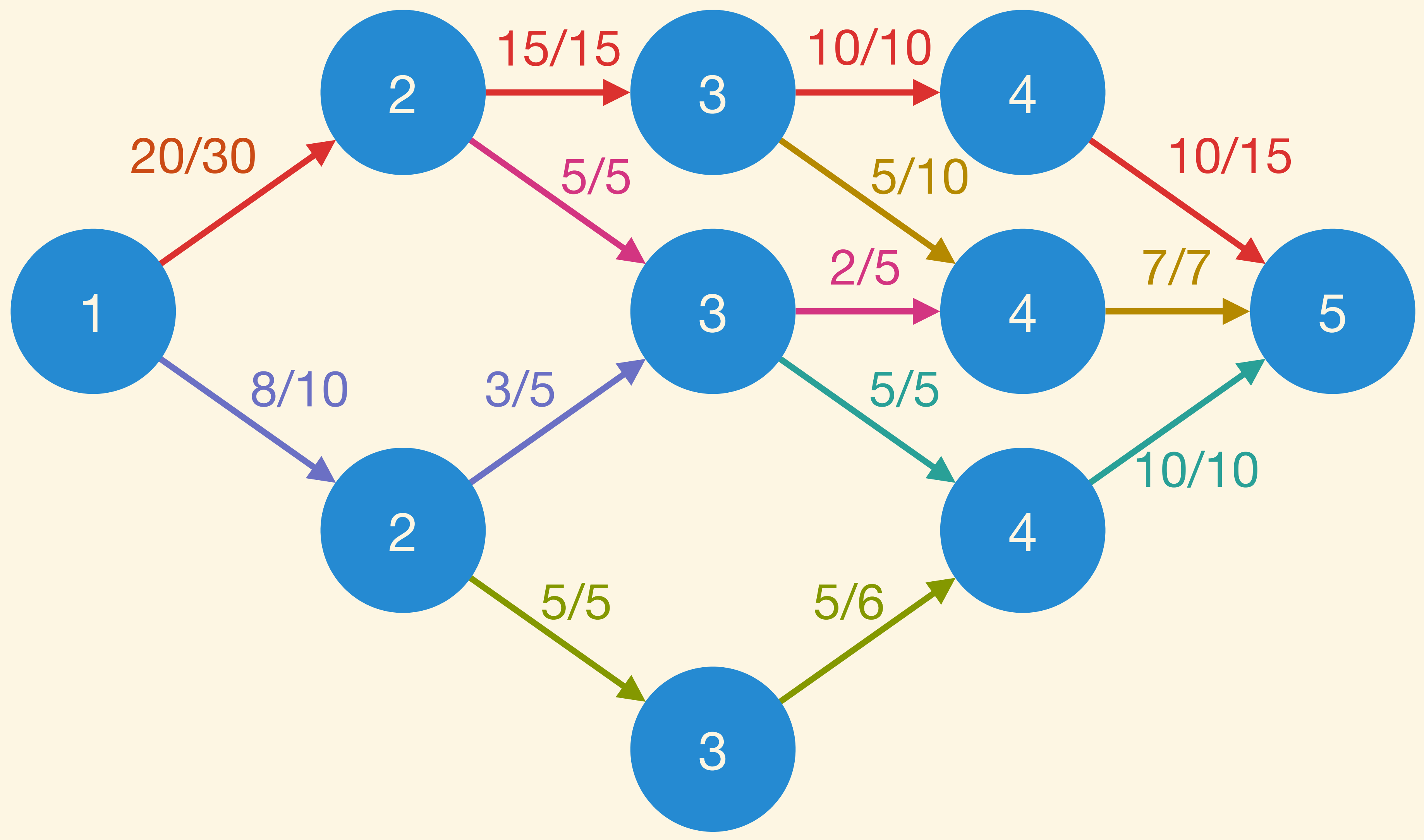
Blocking Flow



Blocking Flow



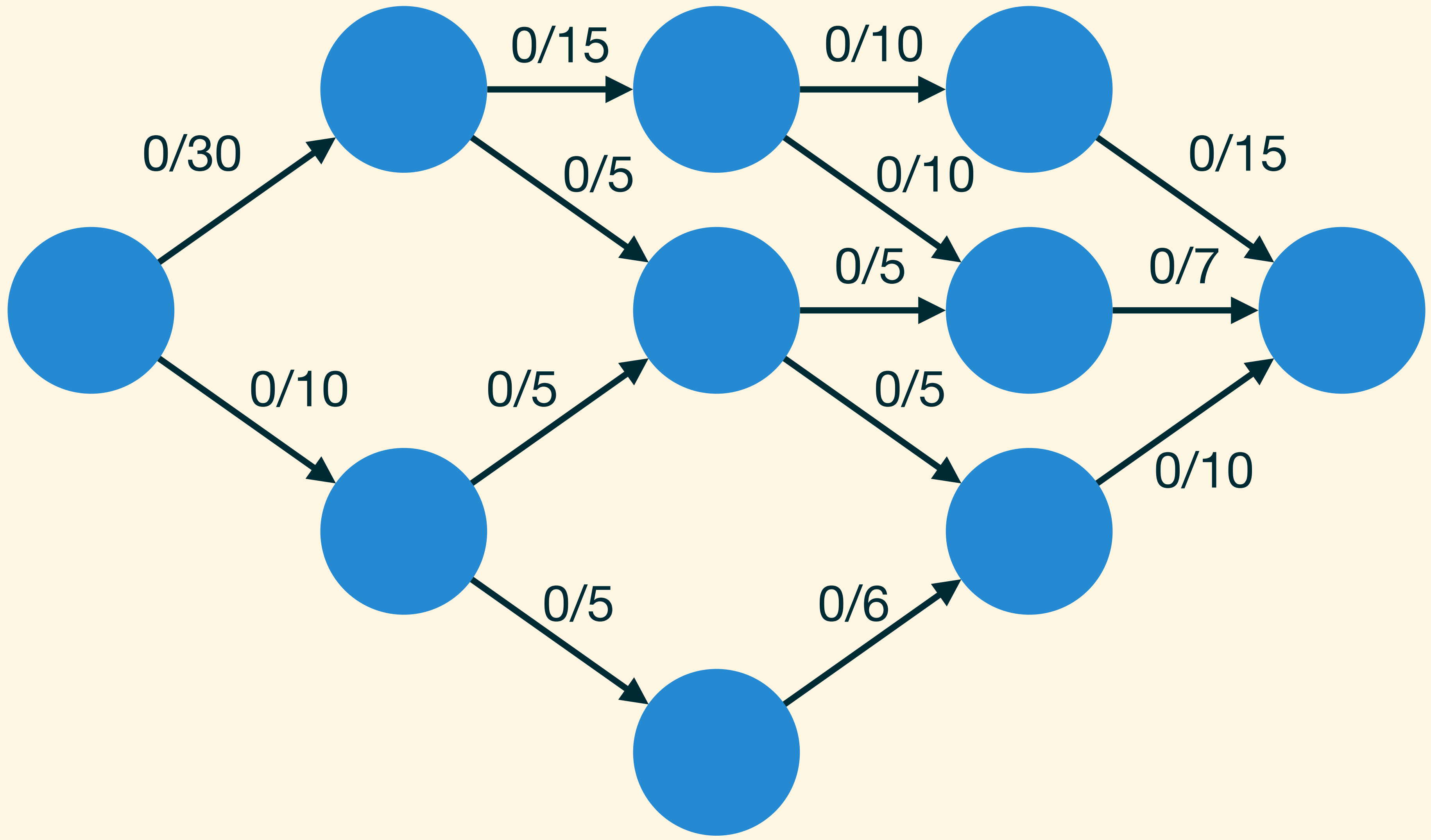
Blocking Flow



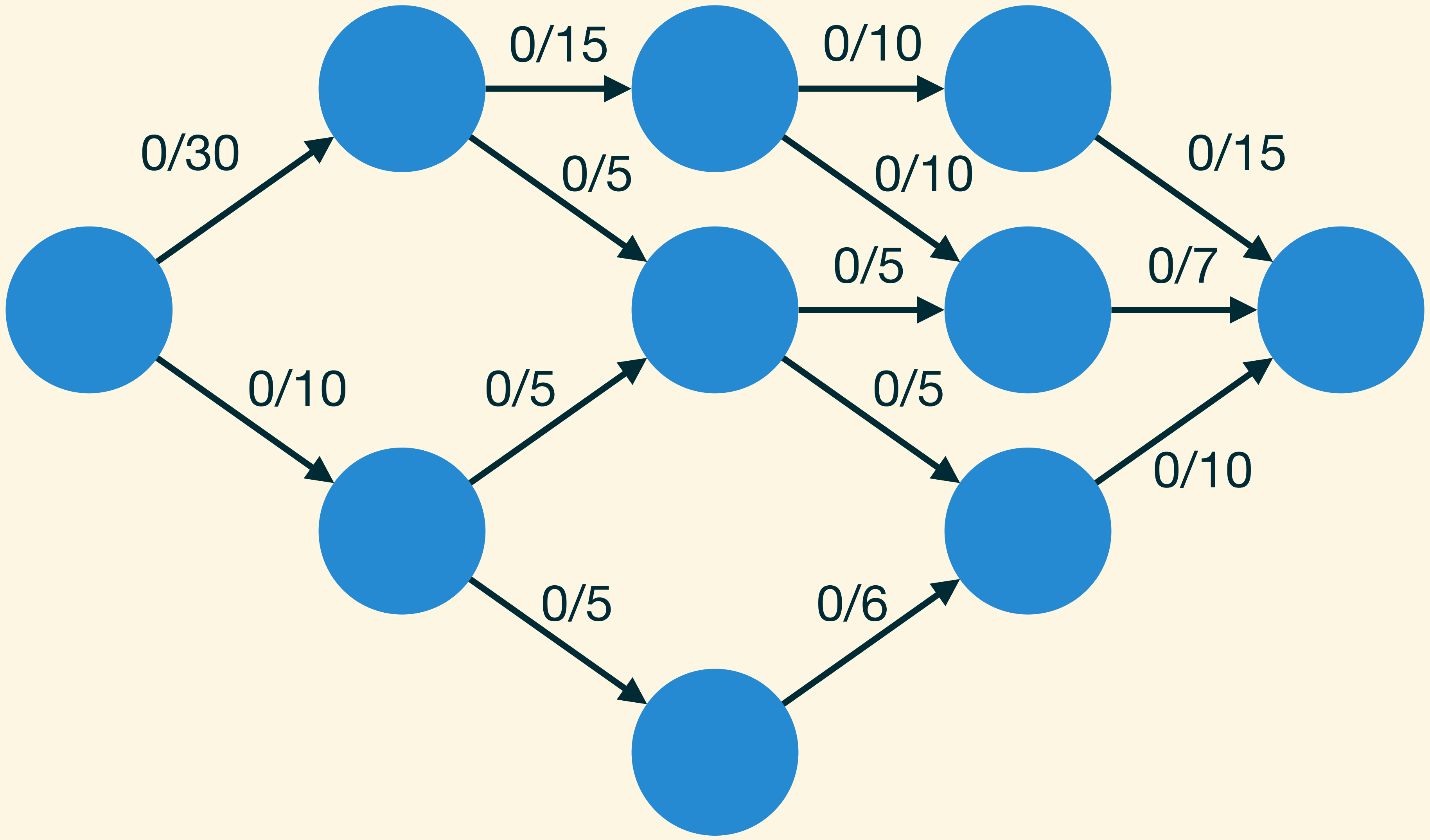
Algorithm

1. build a level graph
2. find an augmenting path from source to sink
3. find the bottleneck on augmenting path
4. find the augmenting path from bottleneck to sink
5. repeat step 3 and step 4 to construct a blocking flow until no augmenting path found

Example

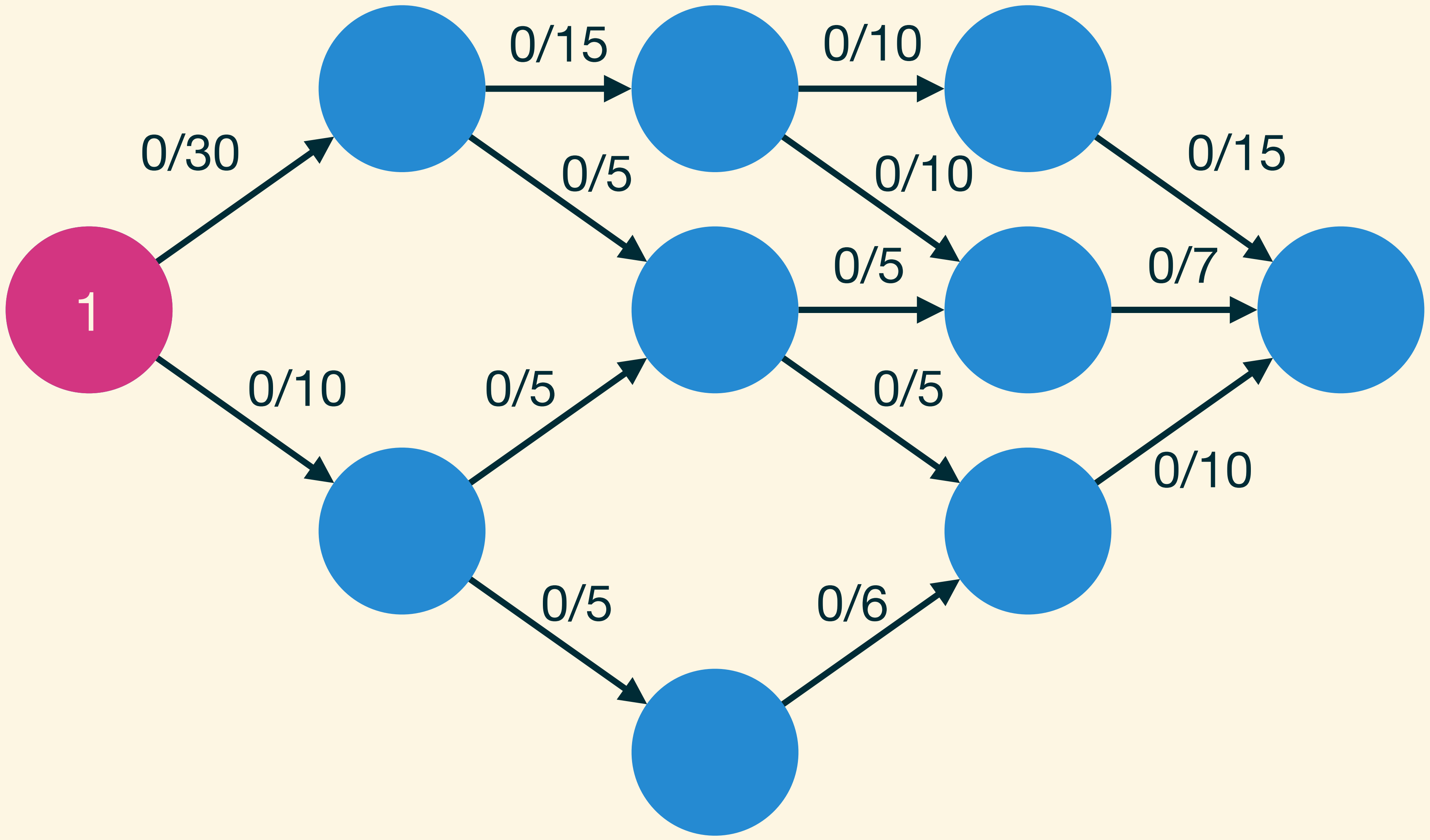


build a level graph



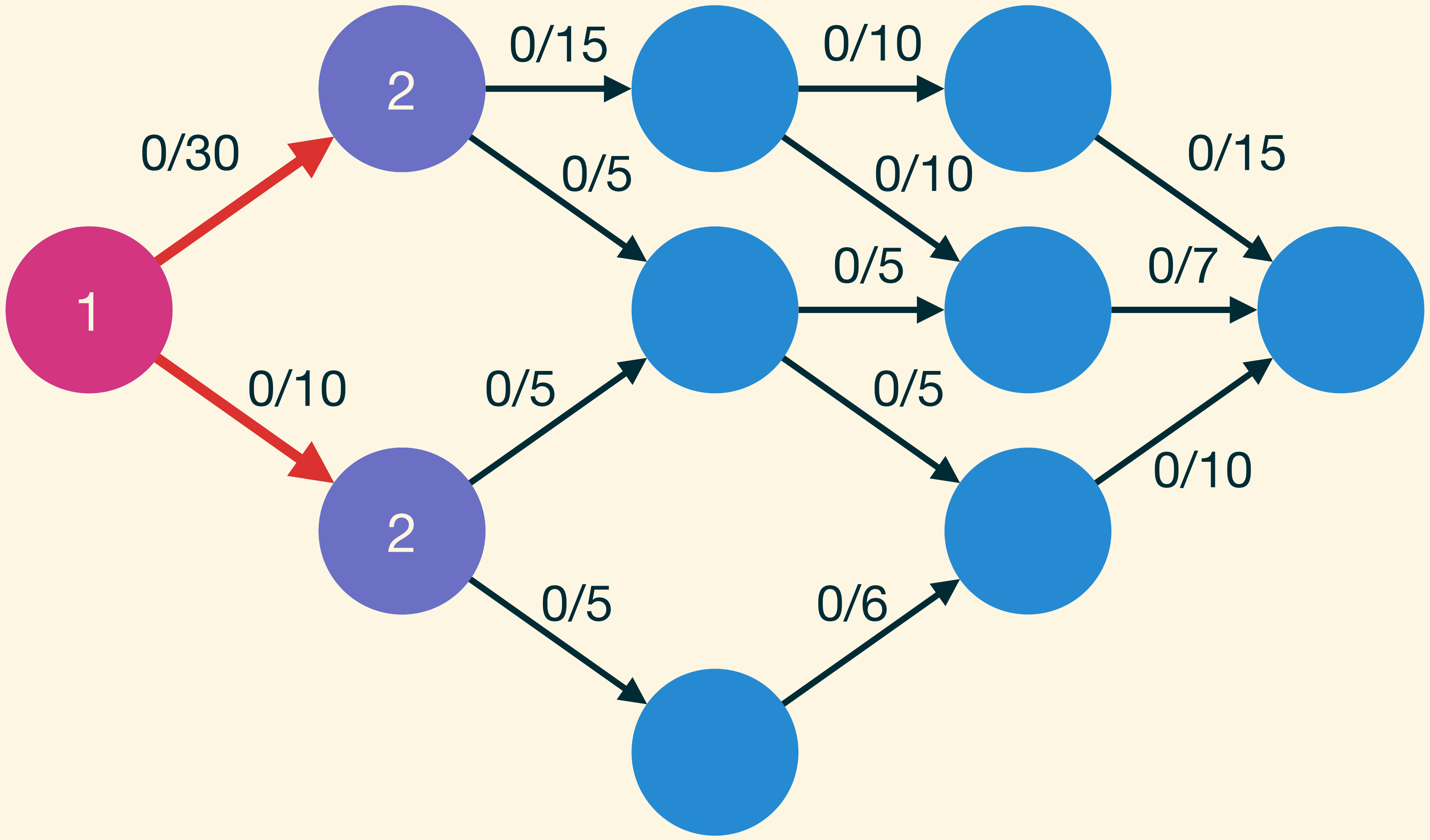
current flow: 0

build a level graph



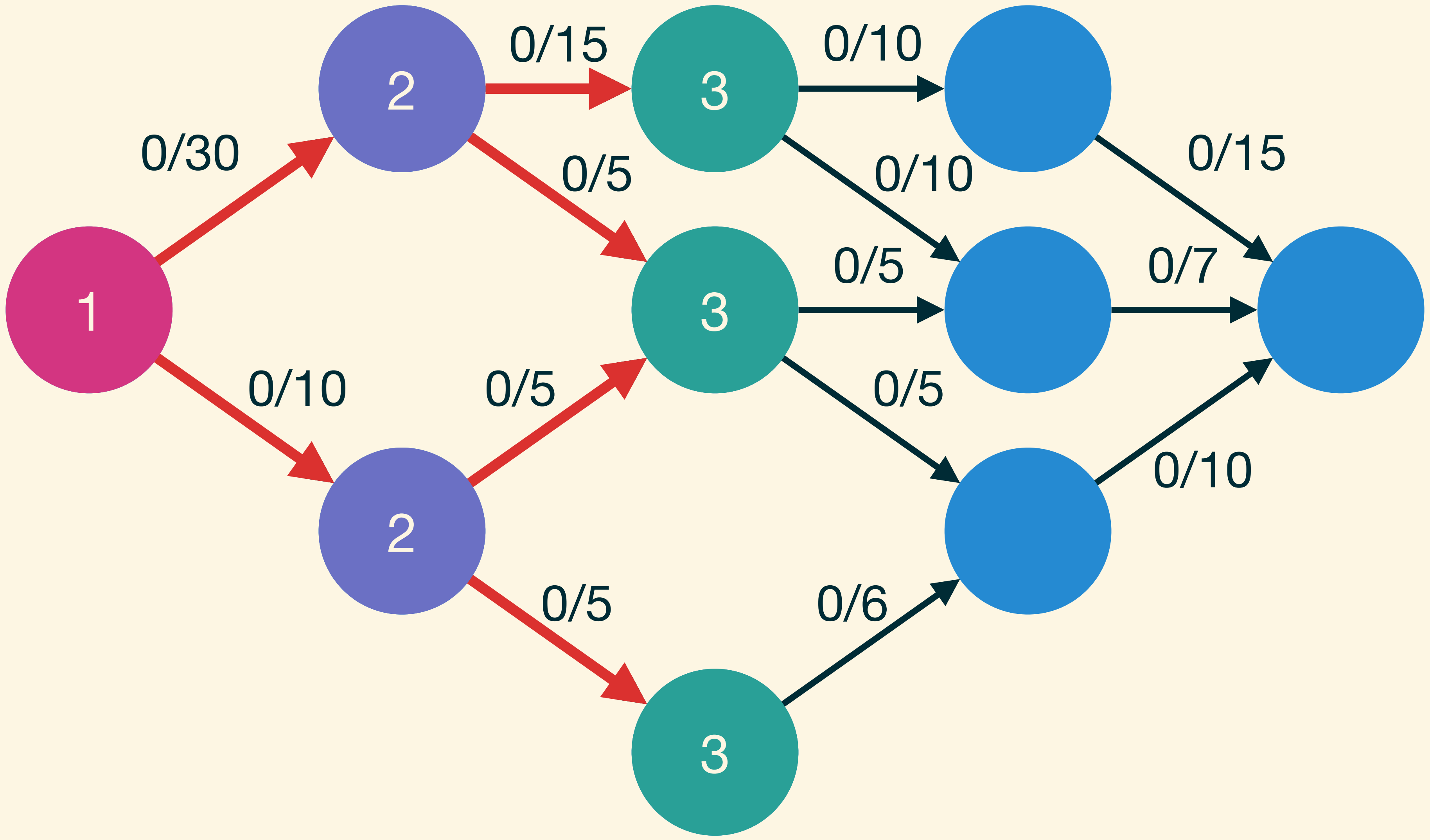
current flow: 0

build a level graph



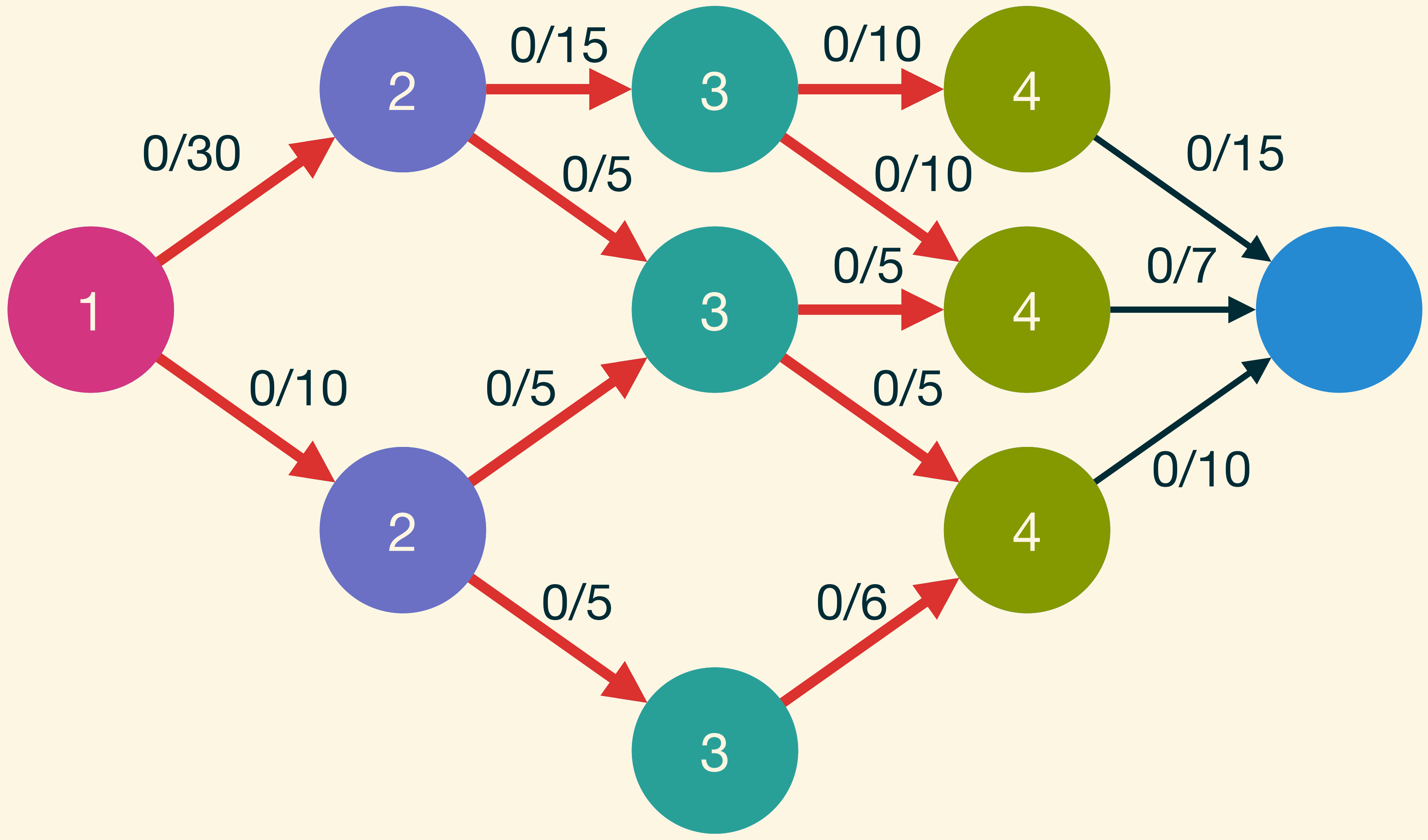
current flow: 0

build a level graph



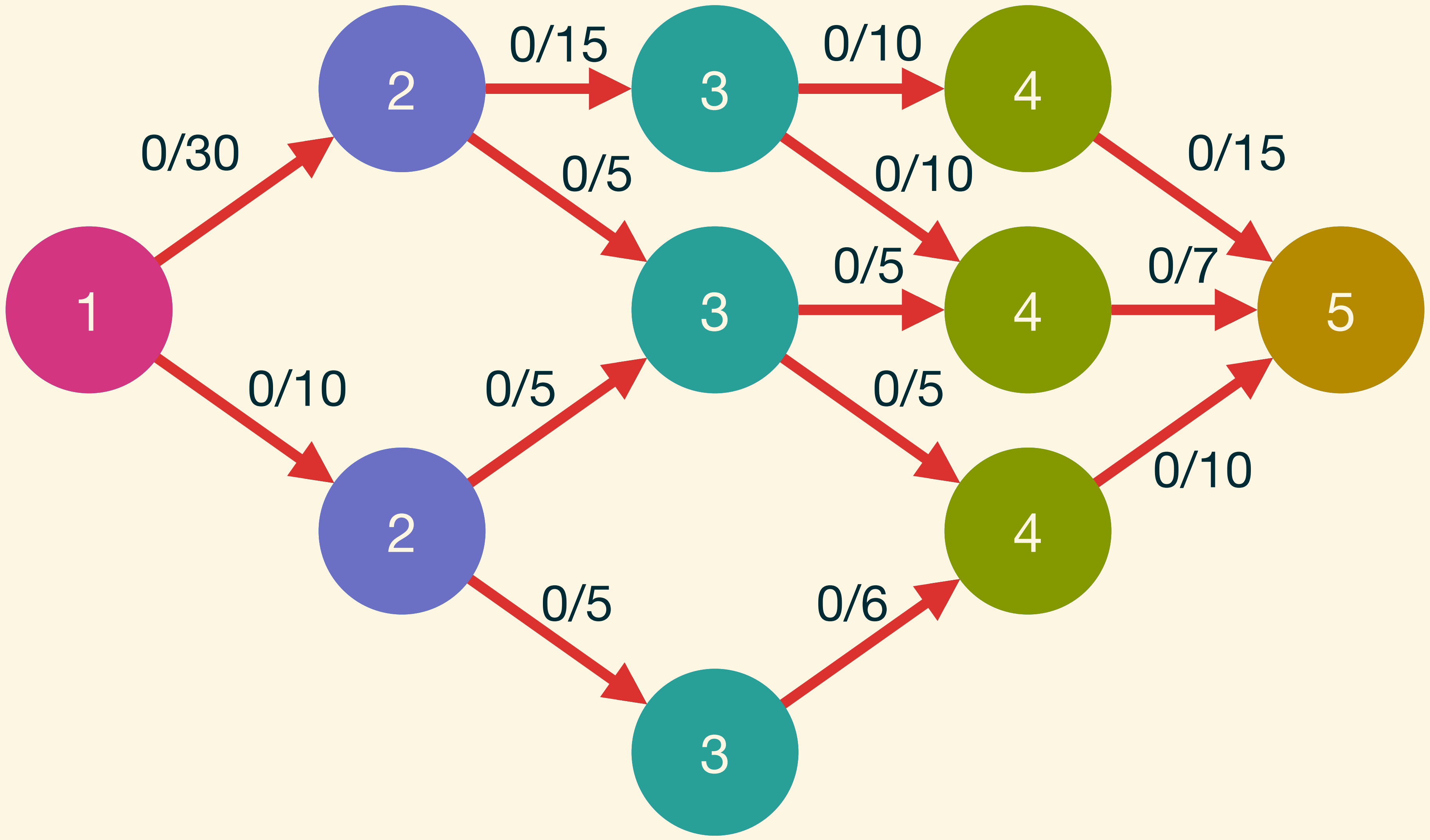
current flow: 0

build a level graph



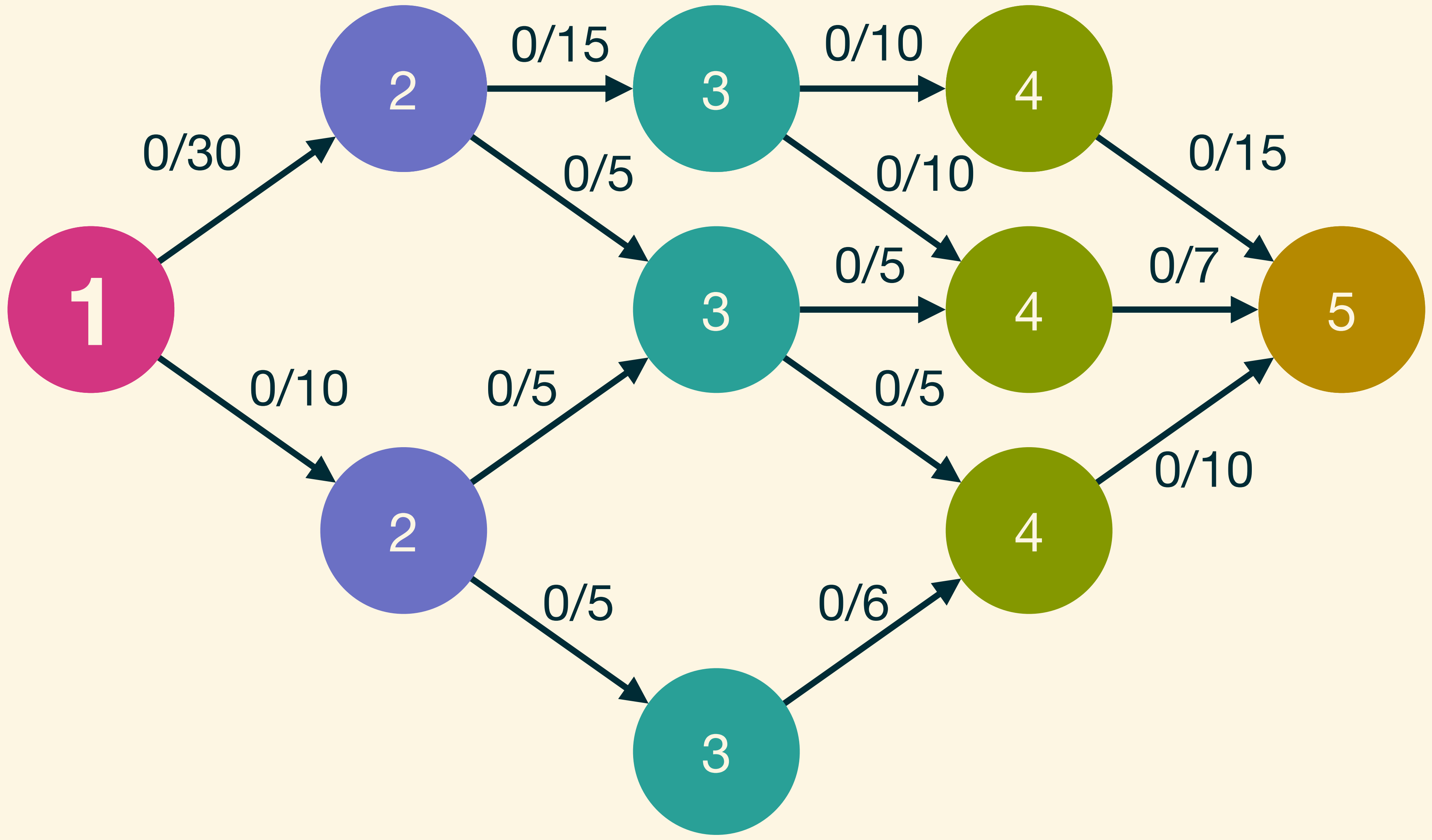
current flow: 0

build a level graph



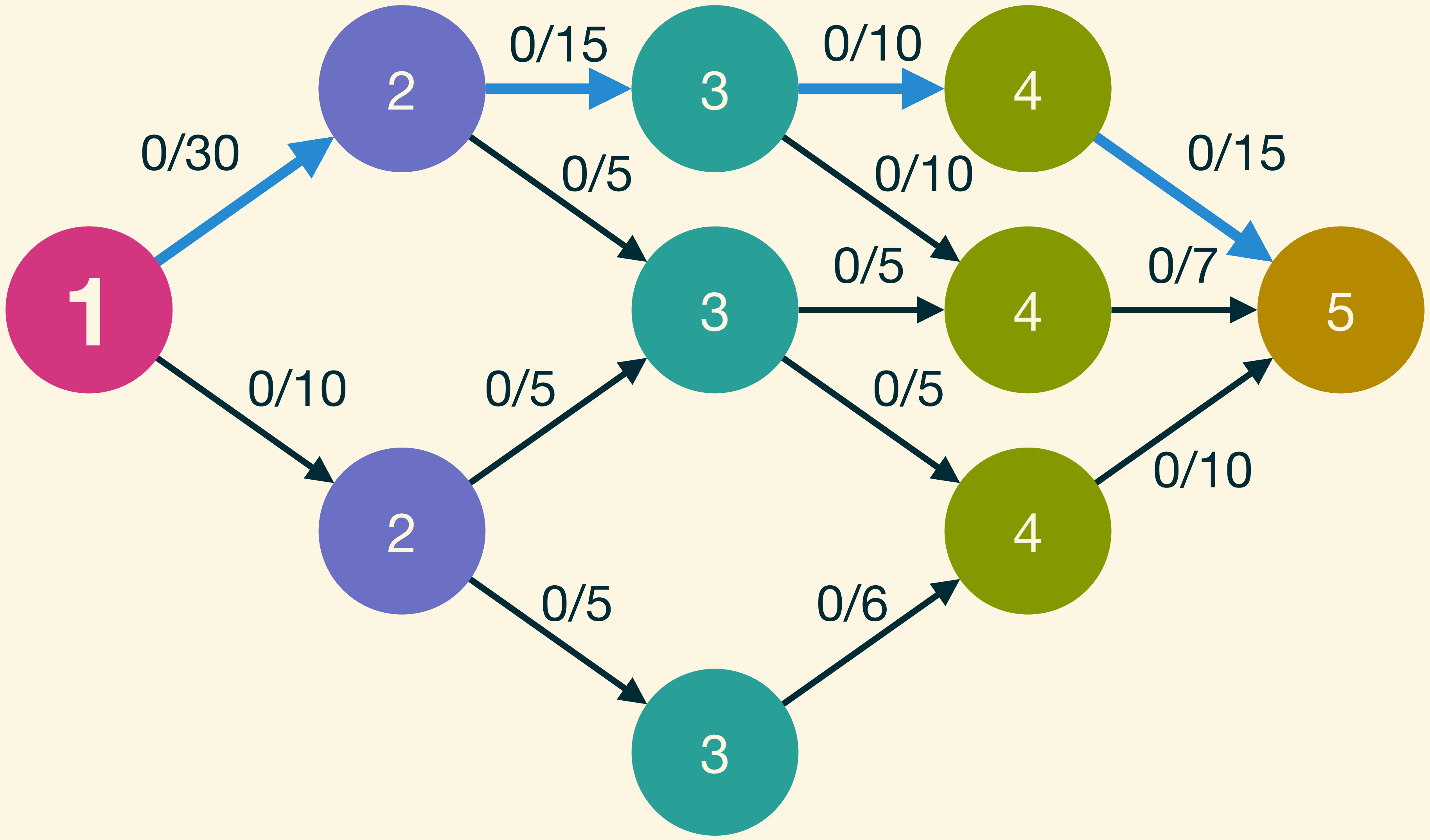
current flow: 0

construct a blocking flow



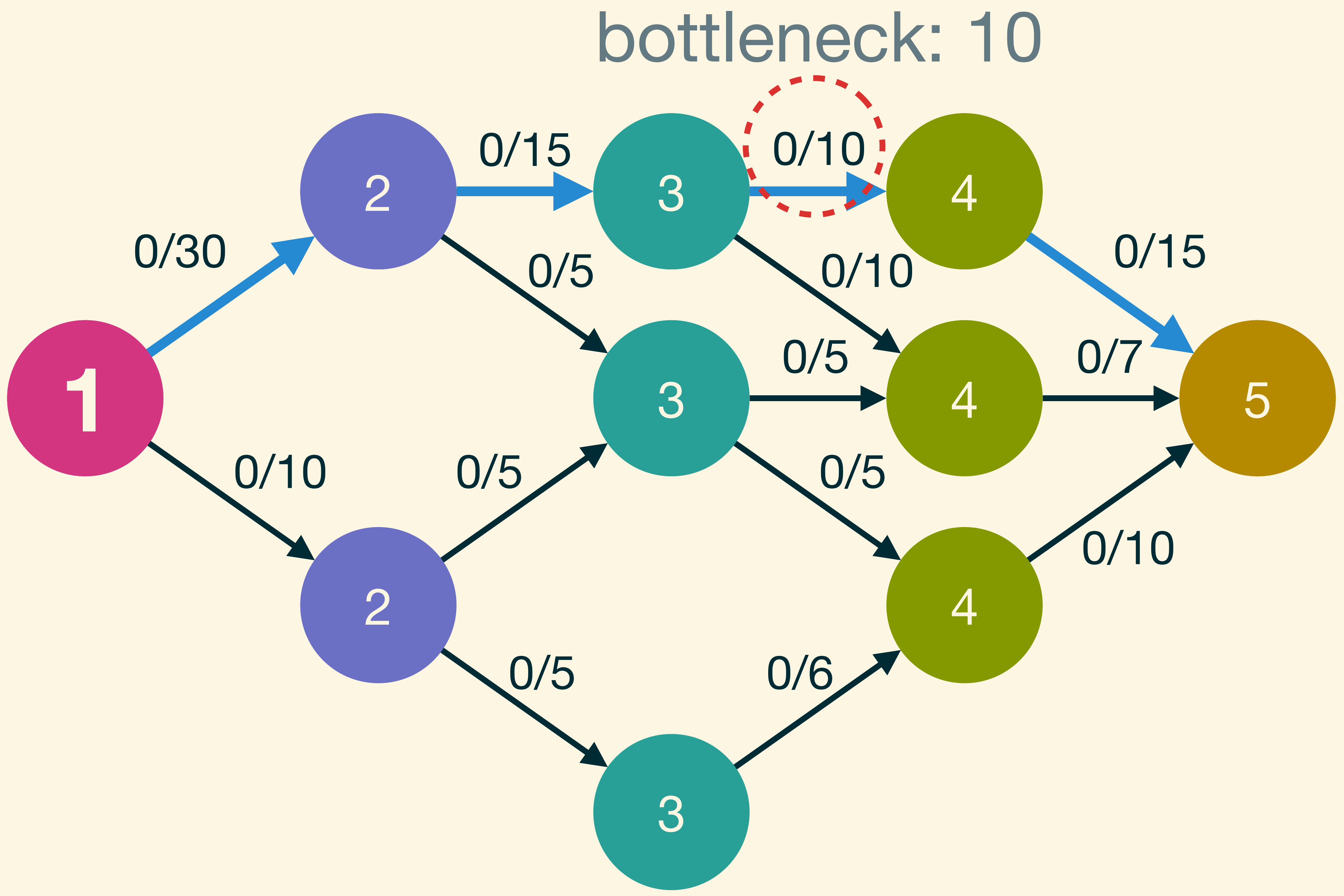
current flow: 0

construct a blocking flow



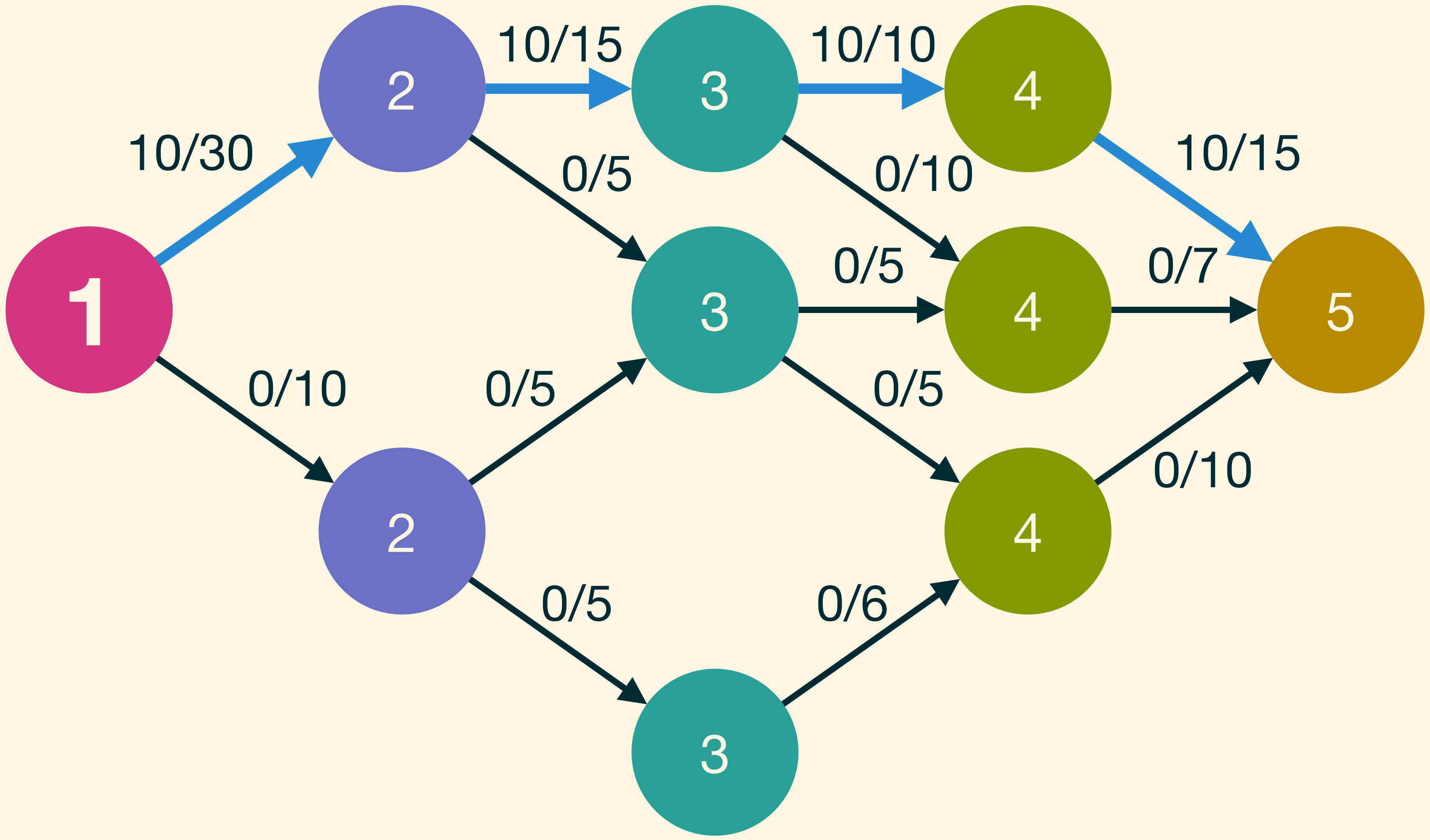
current flow: 0

construct a blocking flow



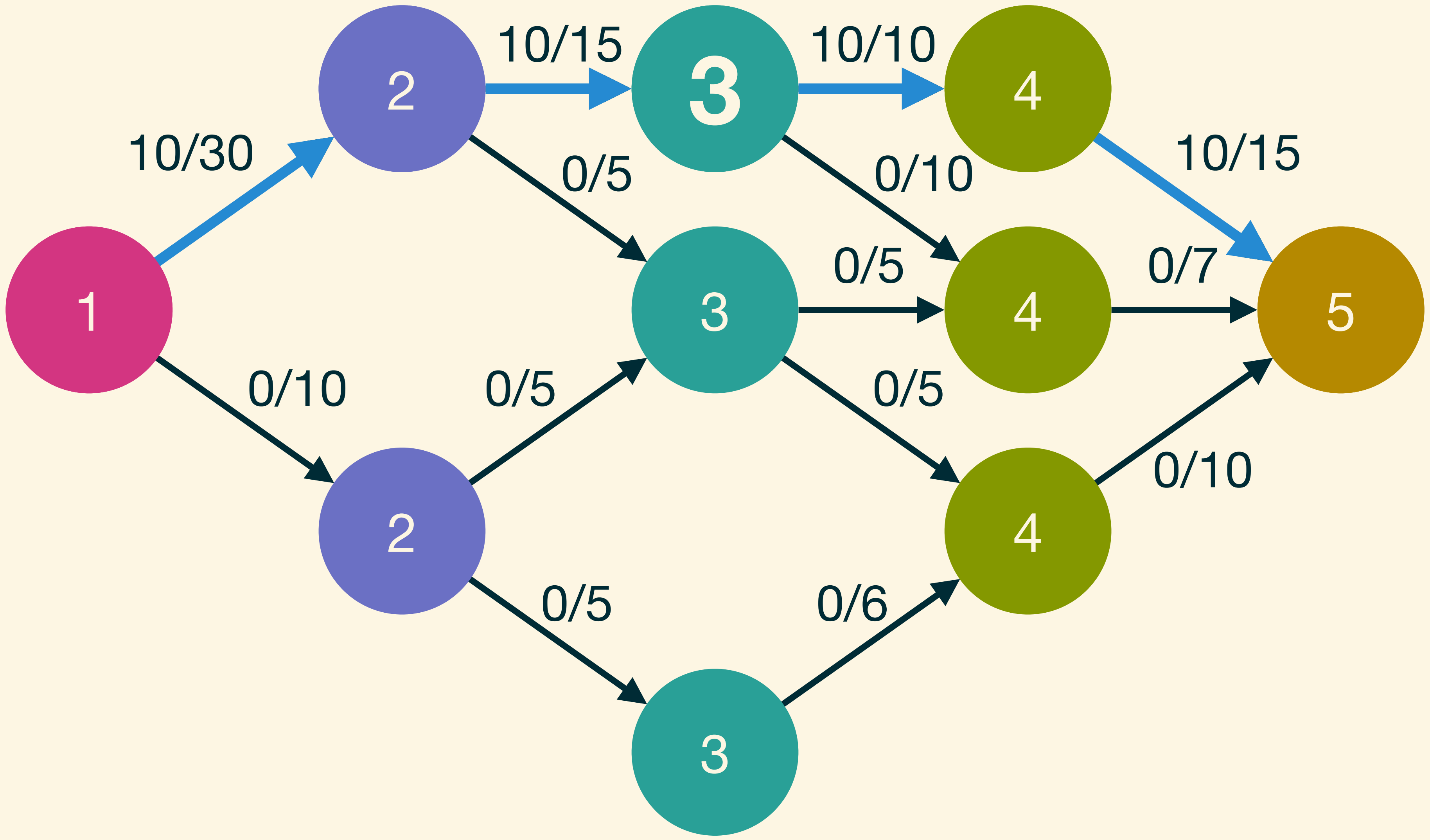
current flow: 0

construct a blocking flow



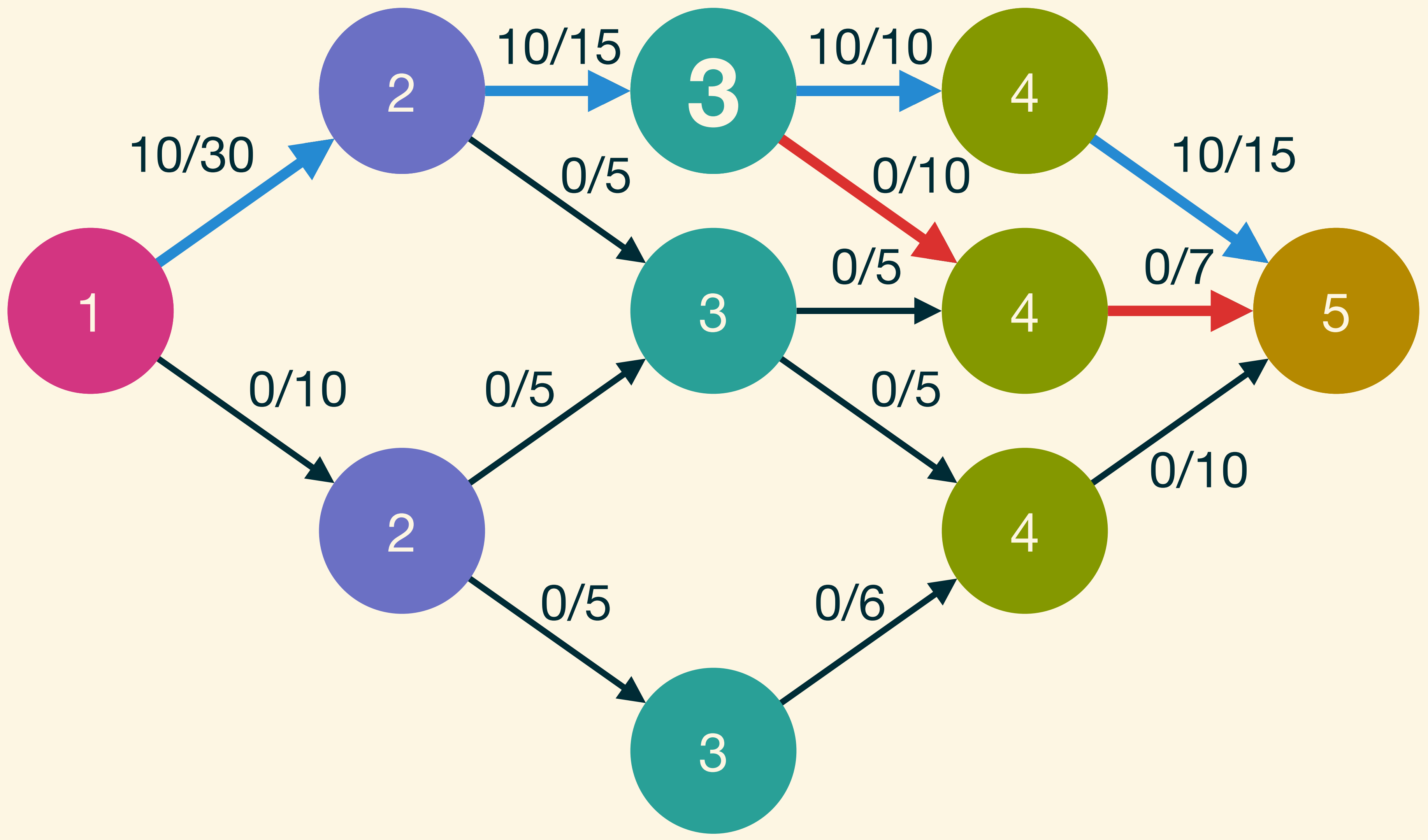
current flow: 10

construct a blocking flow



current flow: 10

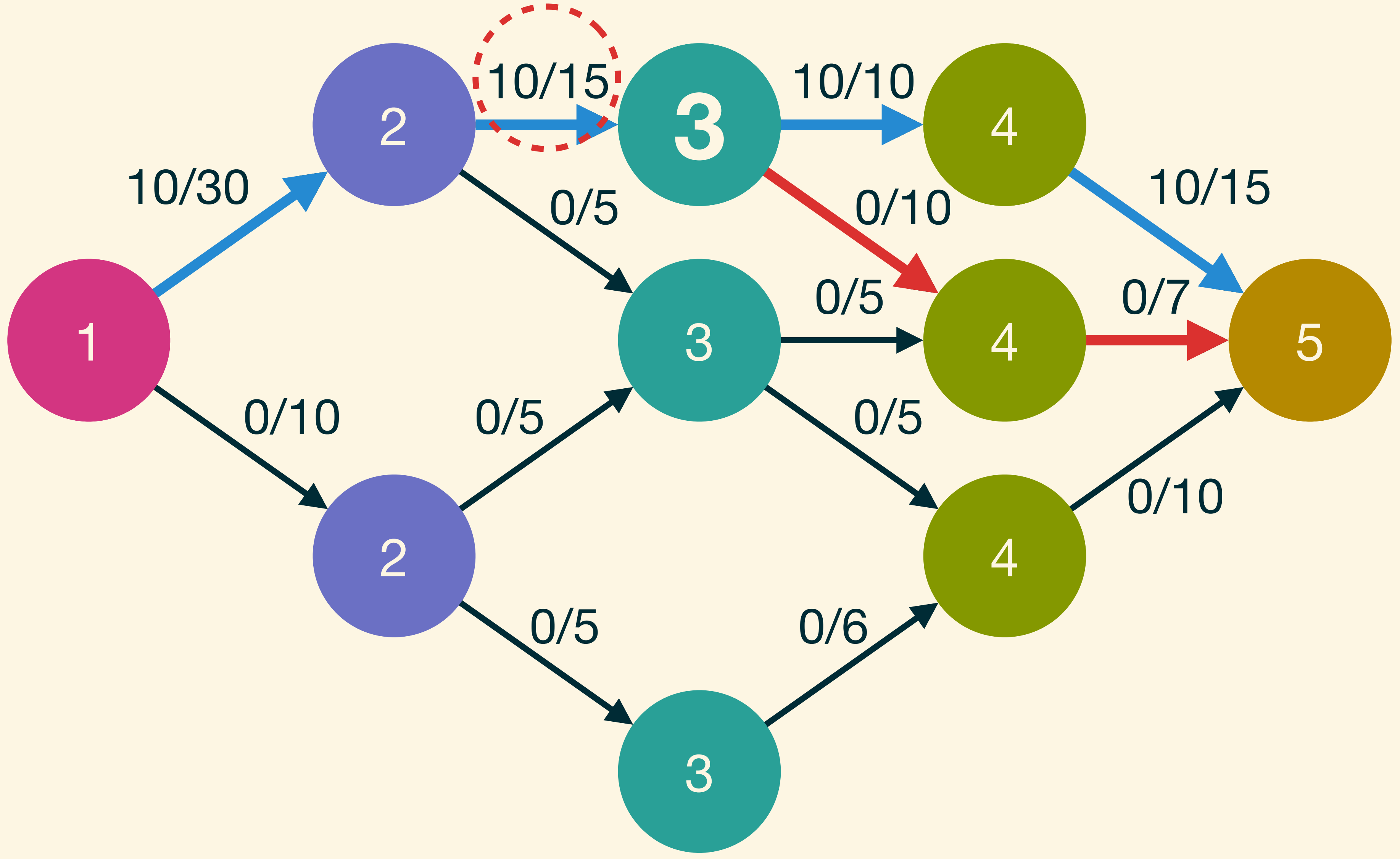
construct a blocking flow



current flow: 10

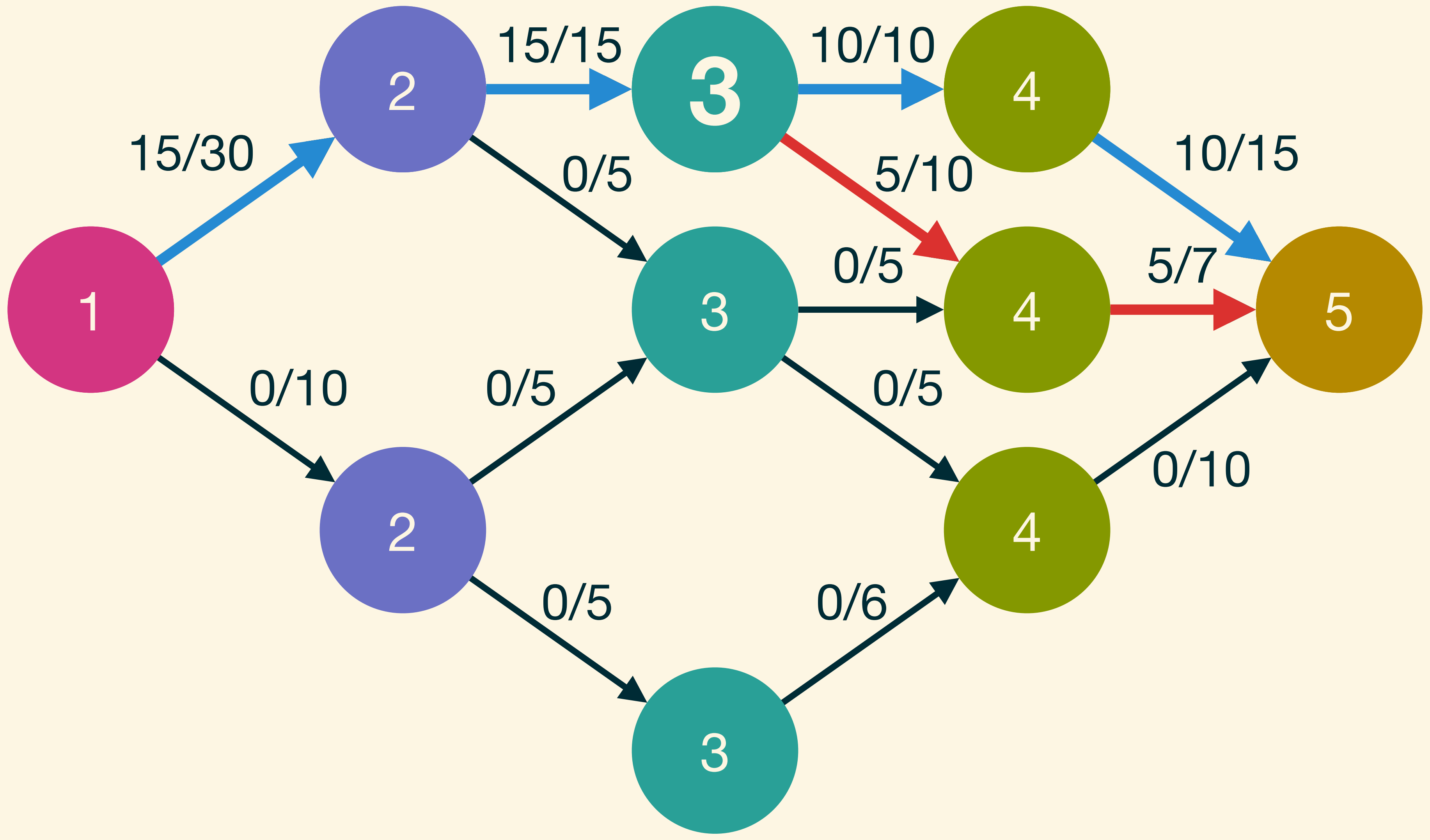
construct a blocking flow

bottleneck: 5



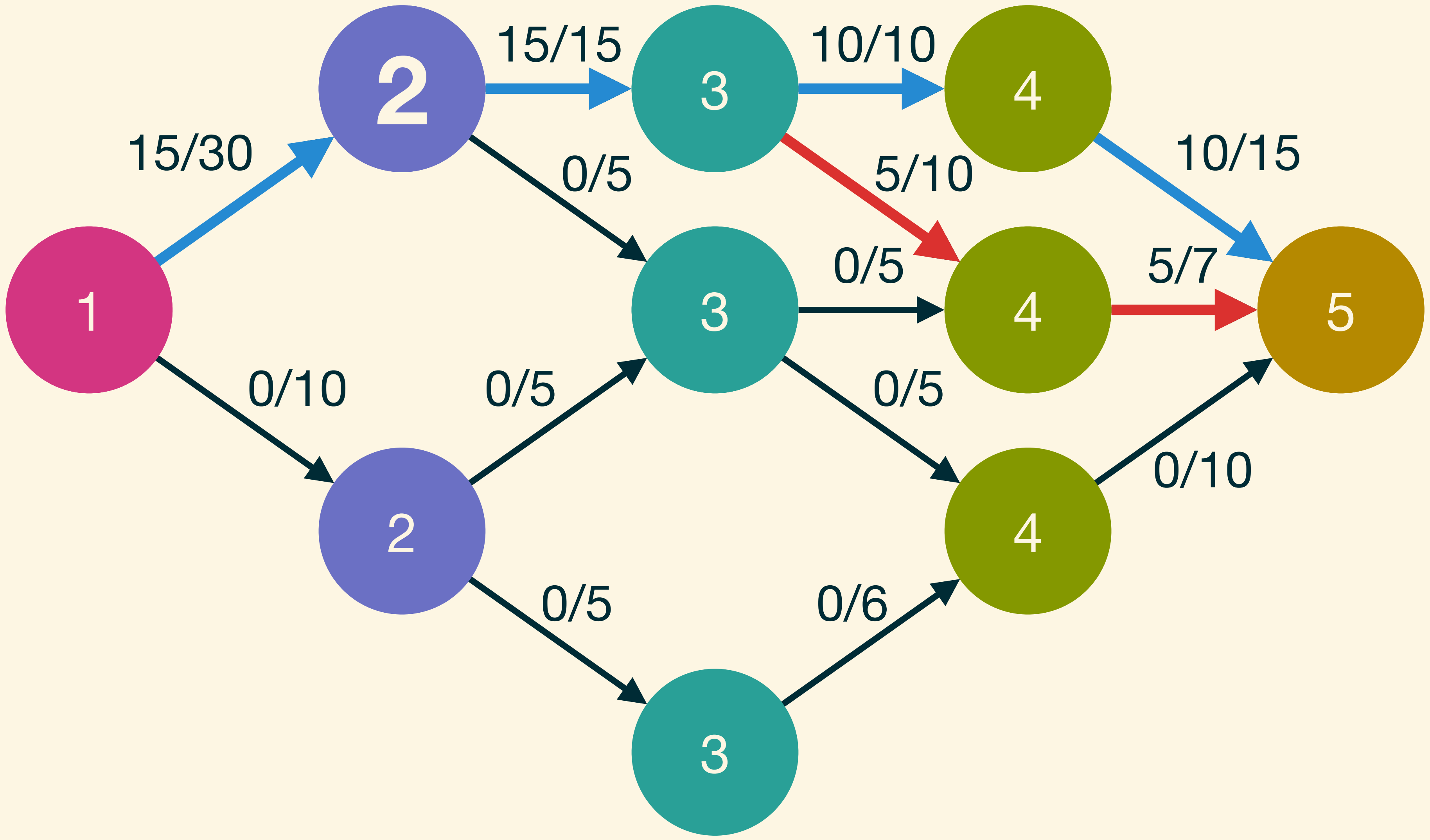
current flow: 10

construct a blocking flow



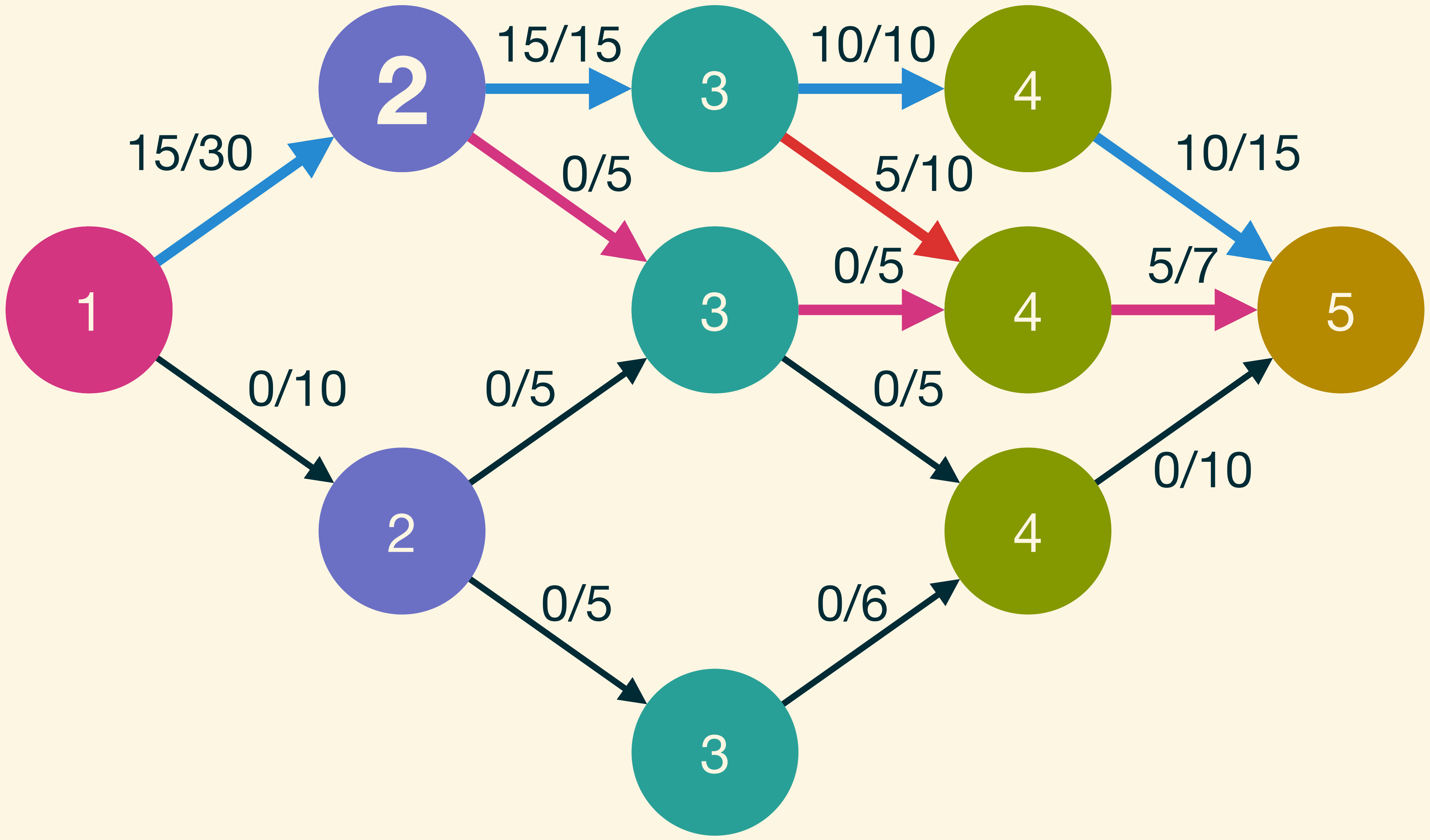
current flow: 15

construct a blocking flow



current flow: 15

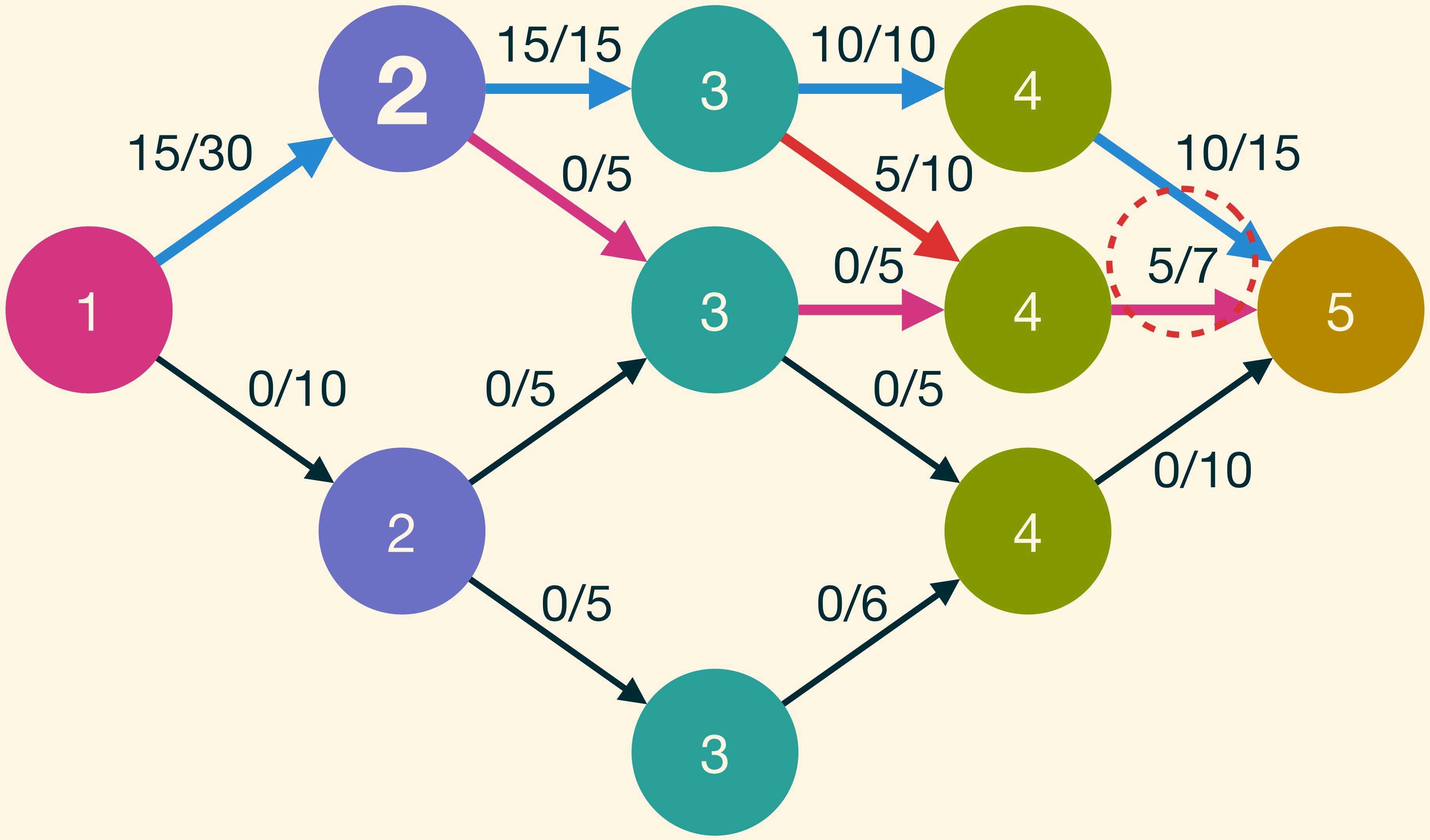
construct a blocking flow



current flow: 15

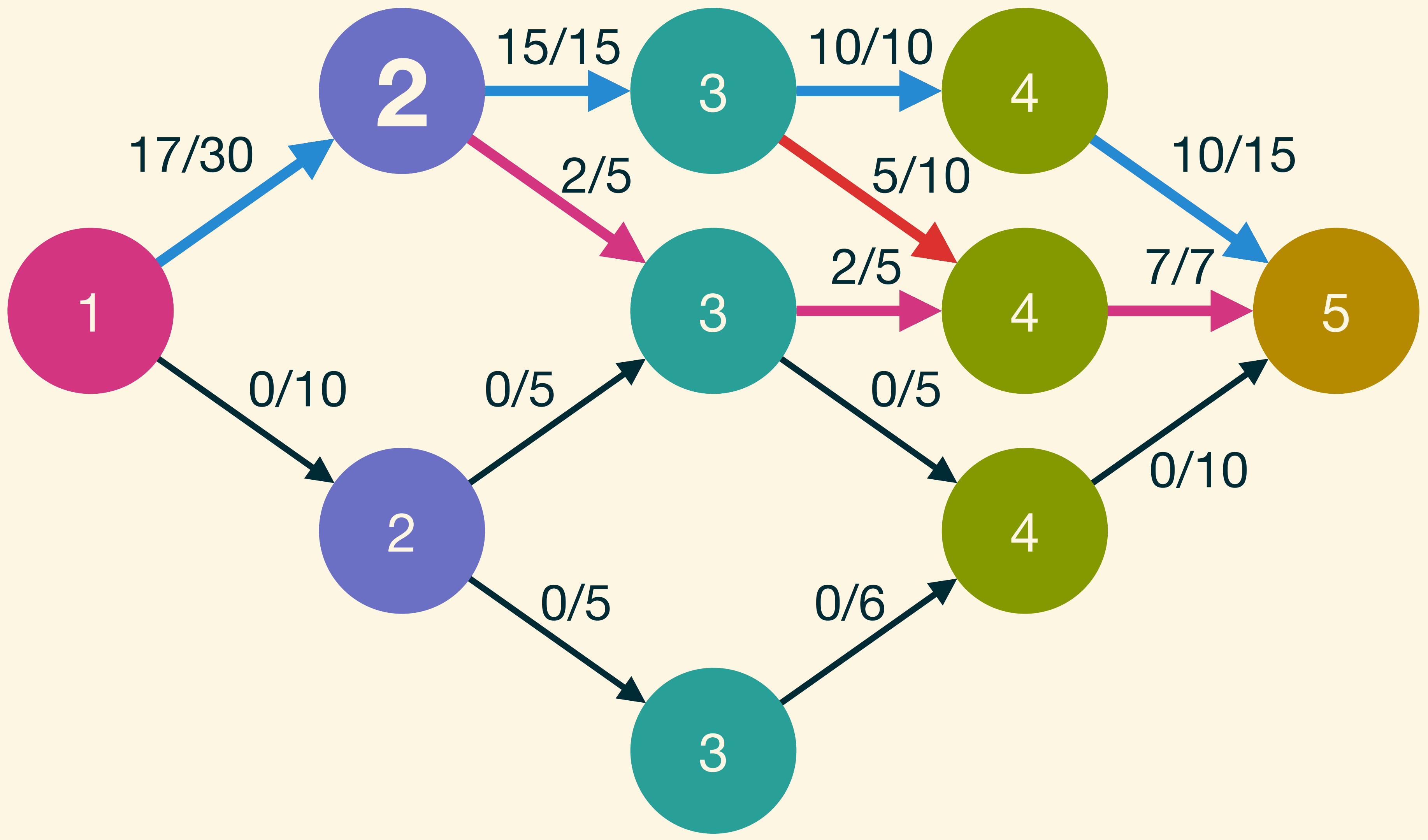
construct a blocking flow

bottleneck: 2



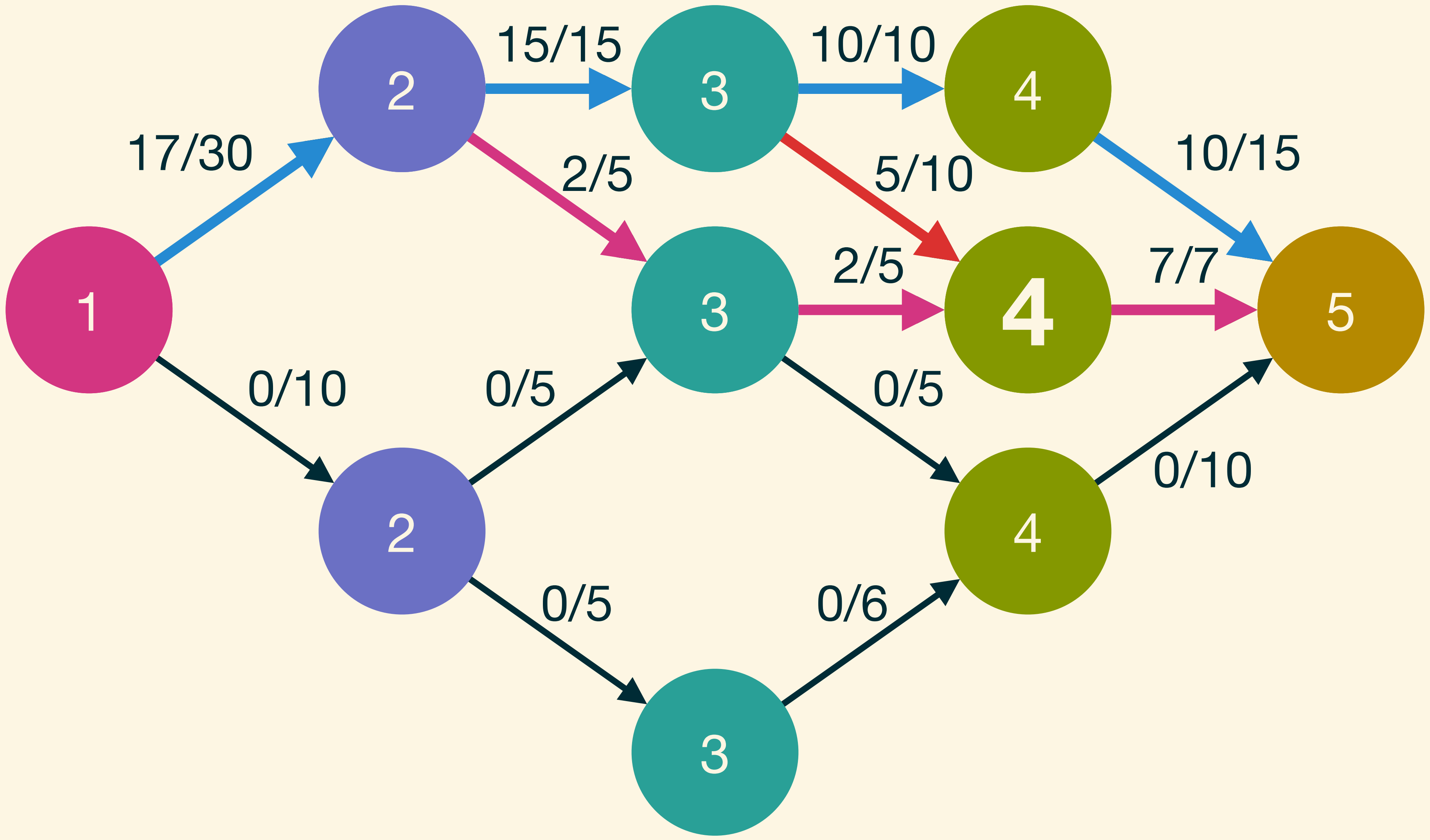
current flow: 15

construct a blocking flow



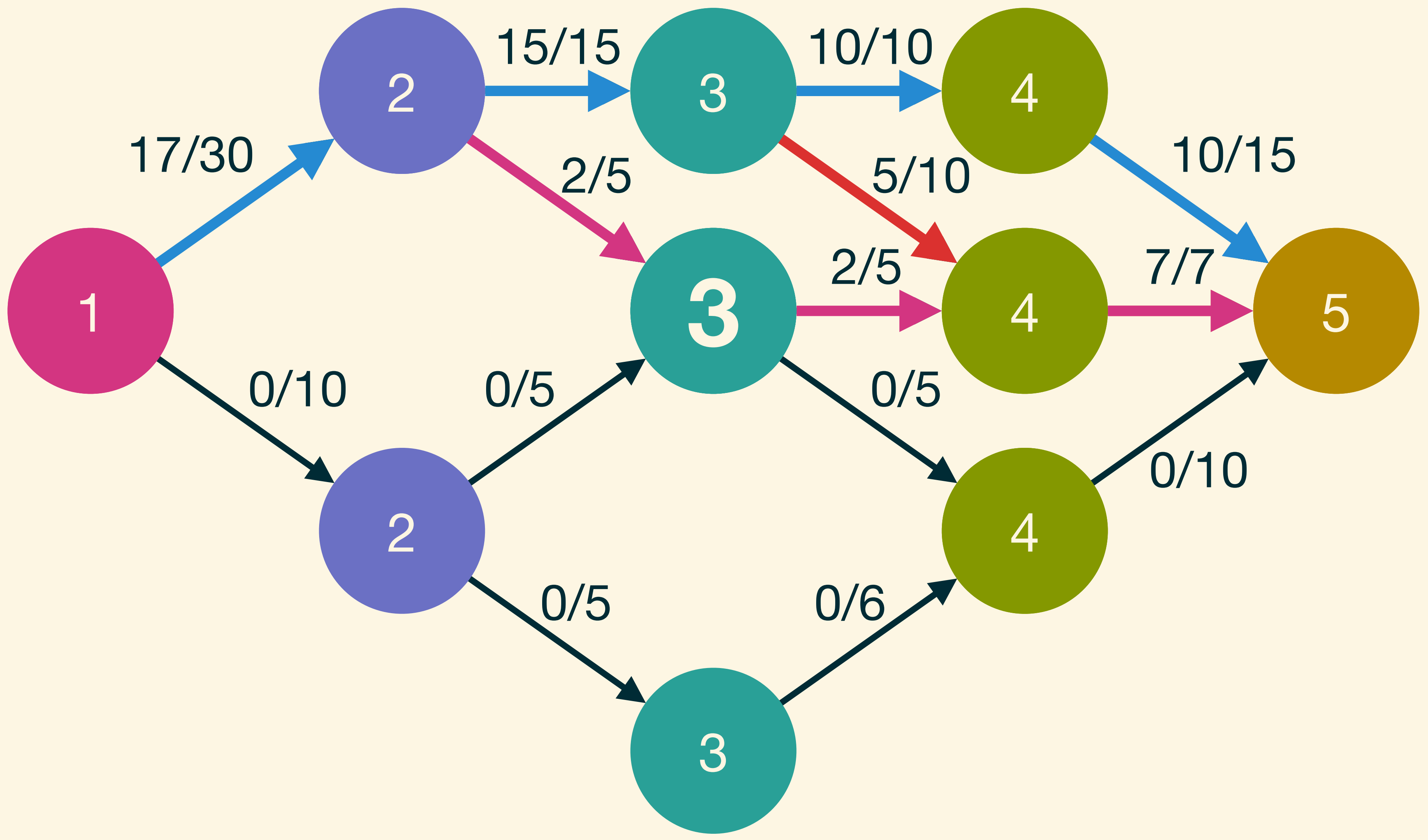
current flow: 17

construct a blocking flow



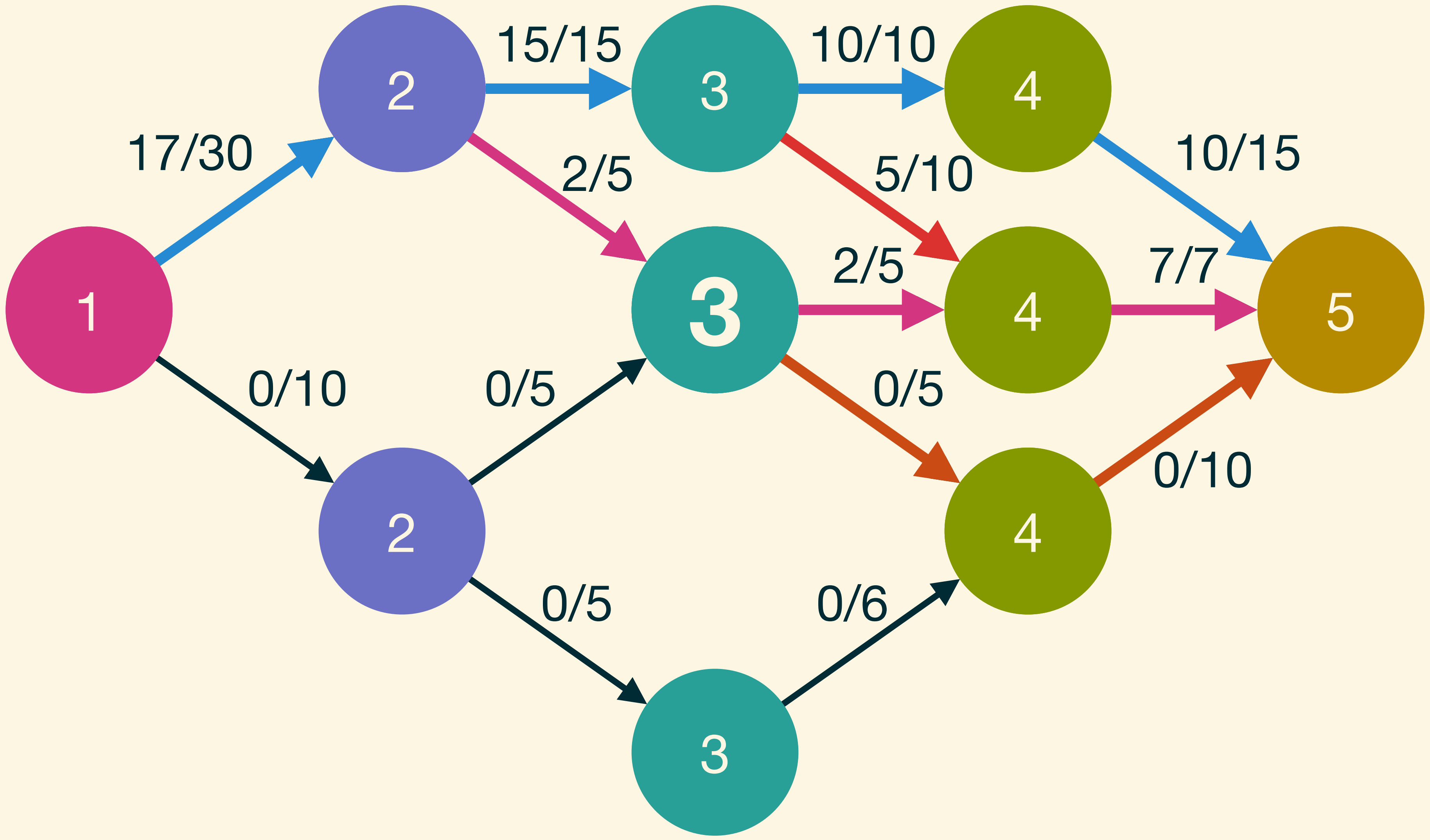
current flow: 17

construct a blocking flow



current flow: 17

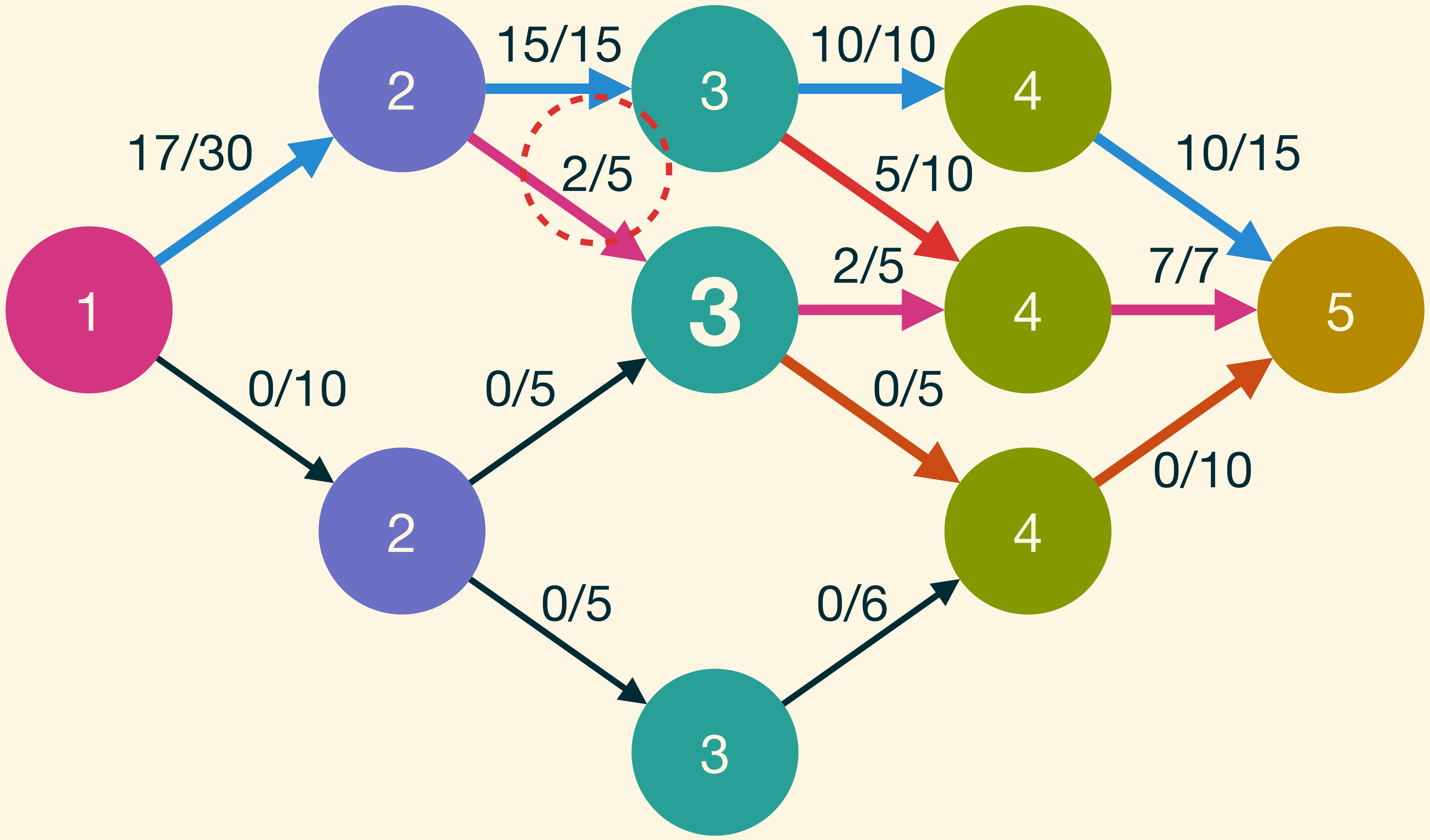
construct a blocking flow



current flow: 17

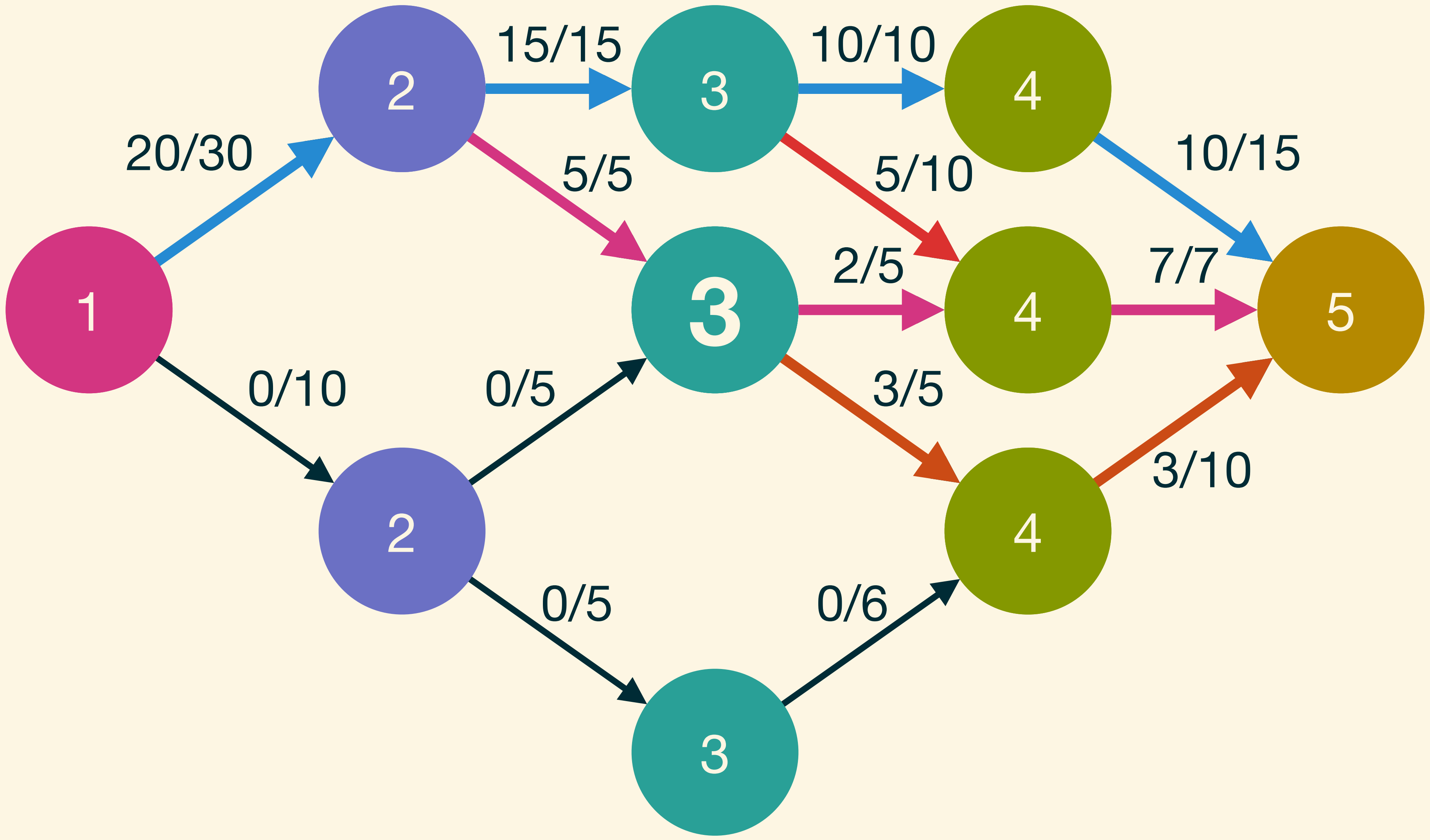
construct a blocking flow

bottleneck: 3



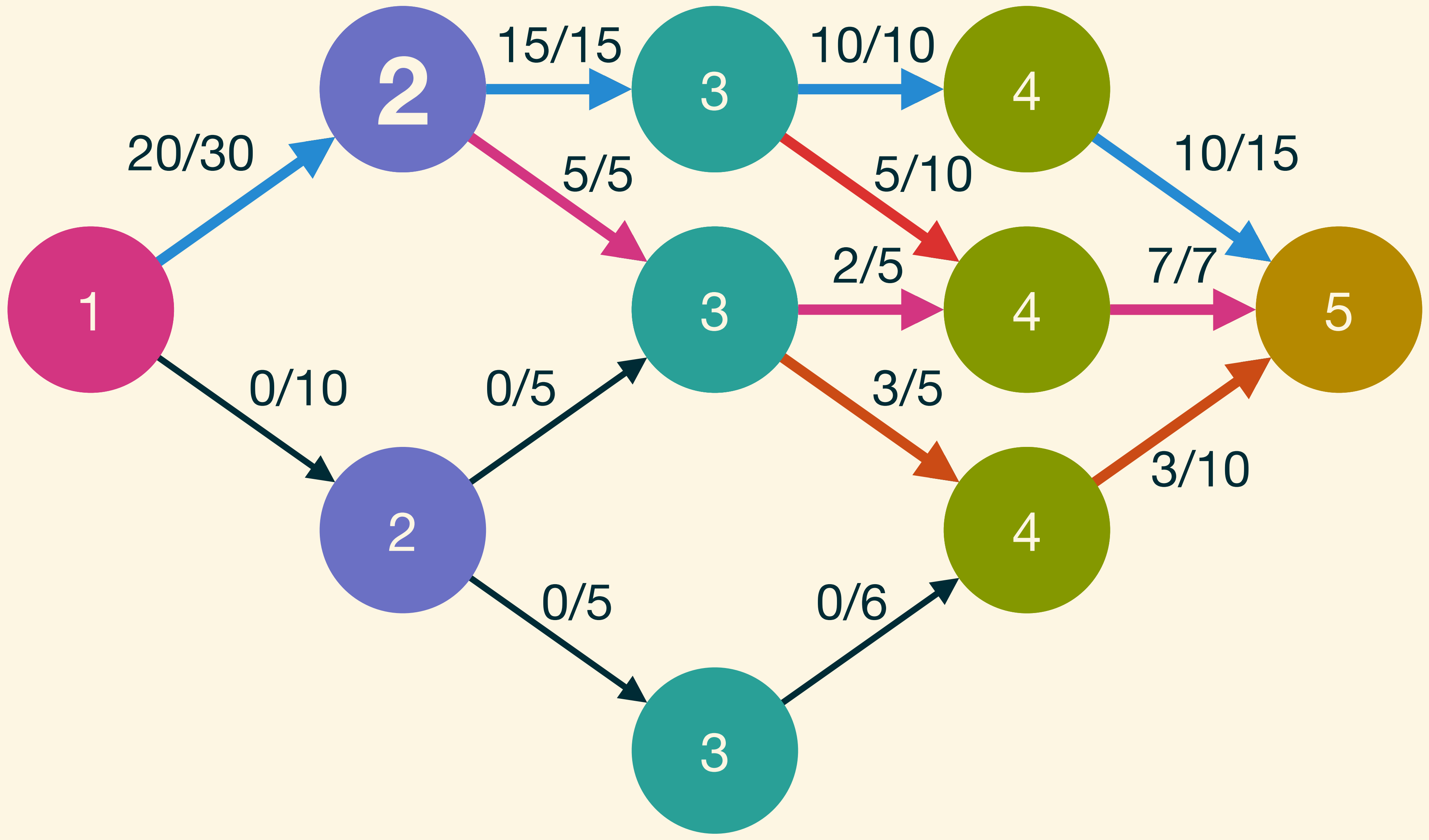
current flow: 17

construct a blocking flow



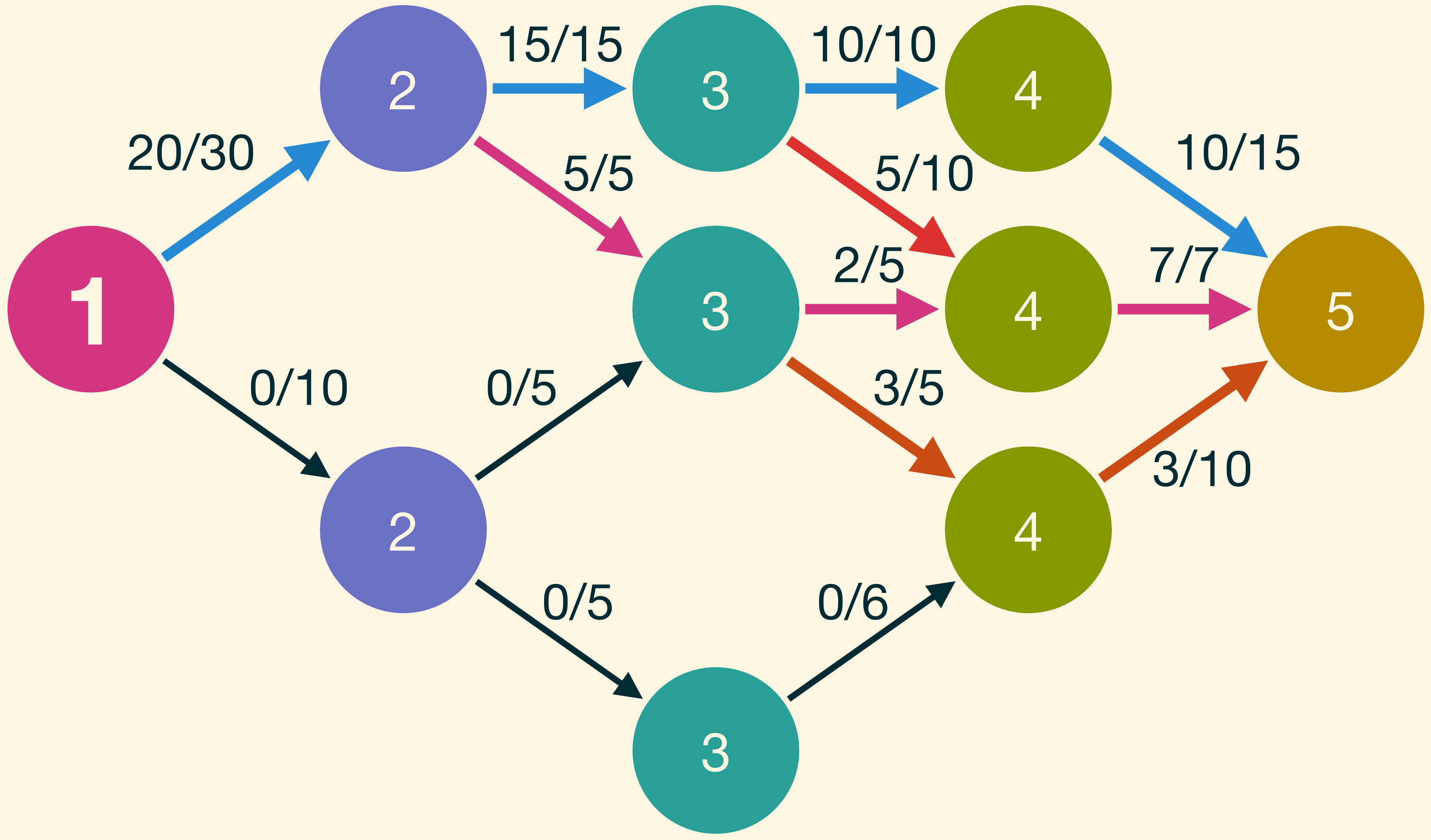
current flow: 20

construct a blocking flow



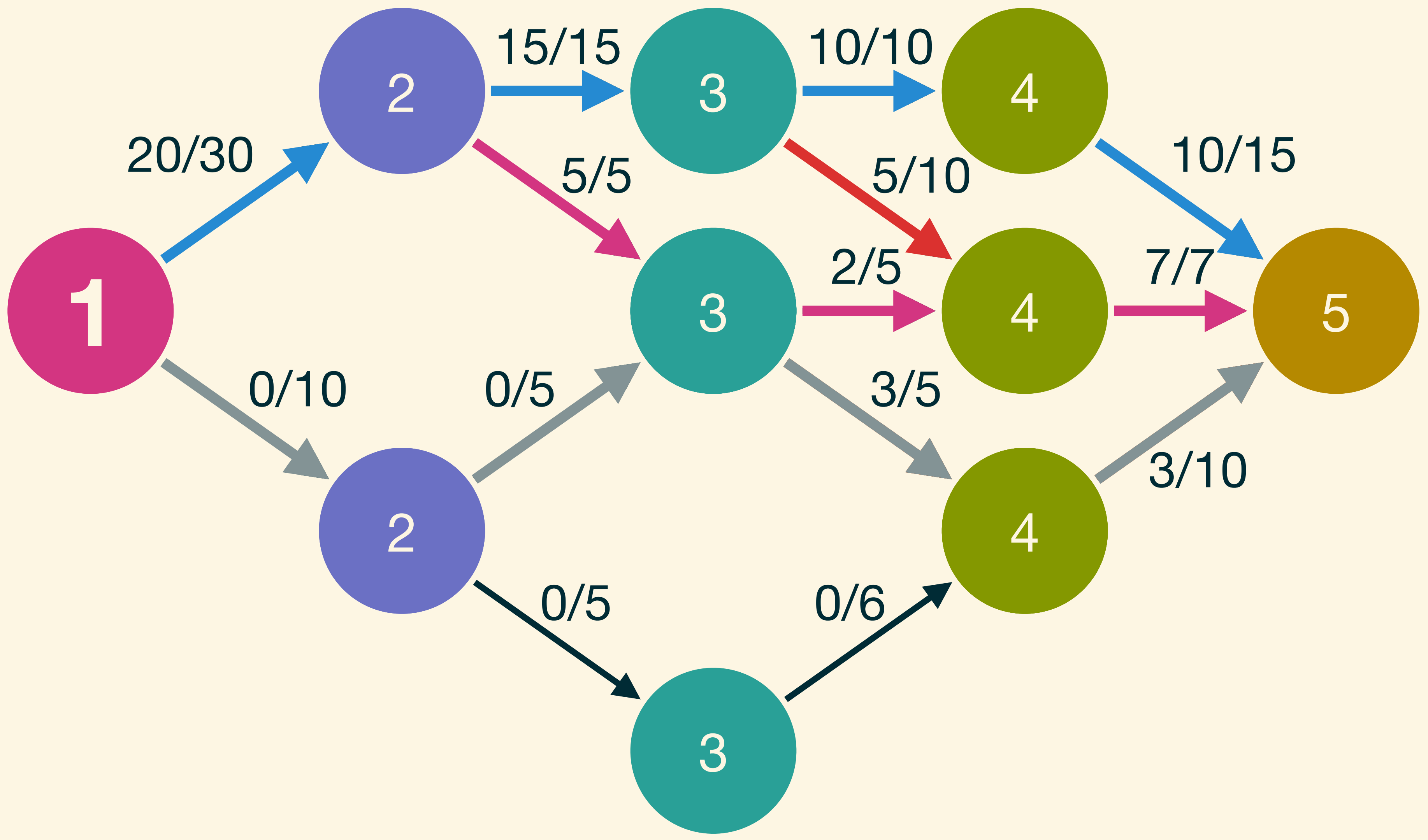
current flow: 20

construct a blocking flow



current flow: 20

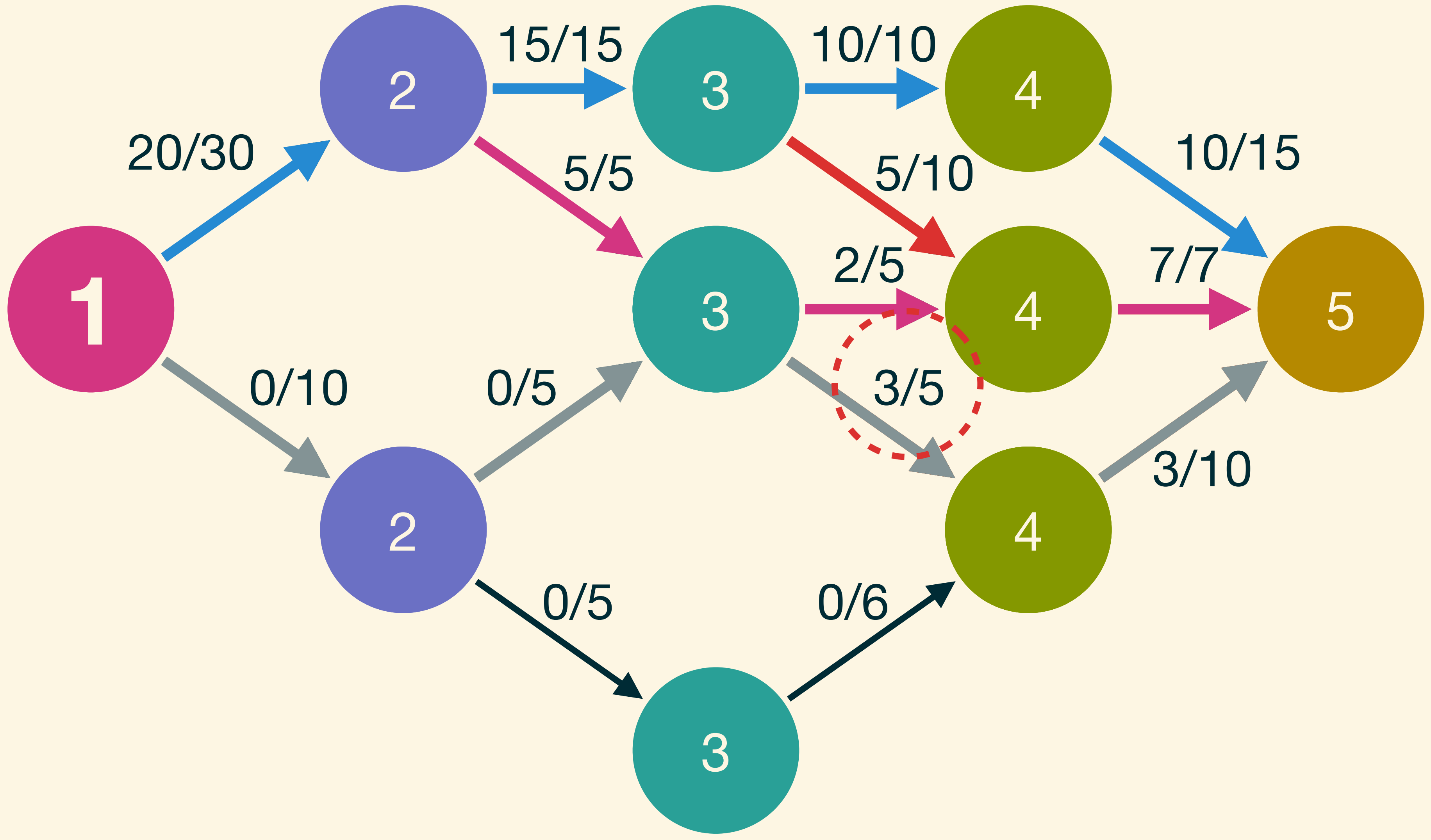
construct a blocking flow



current flow: 20

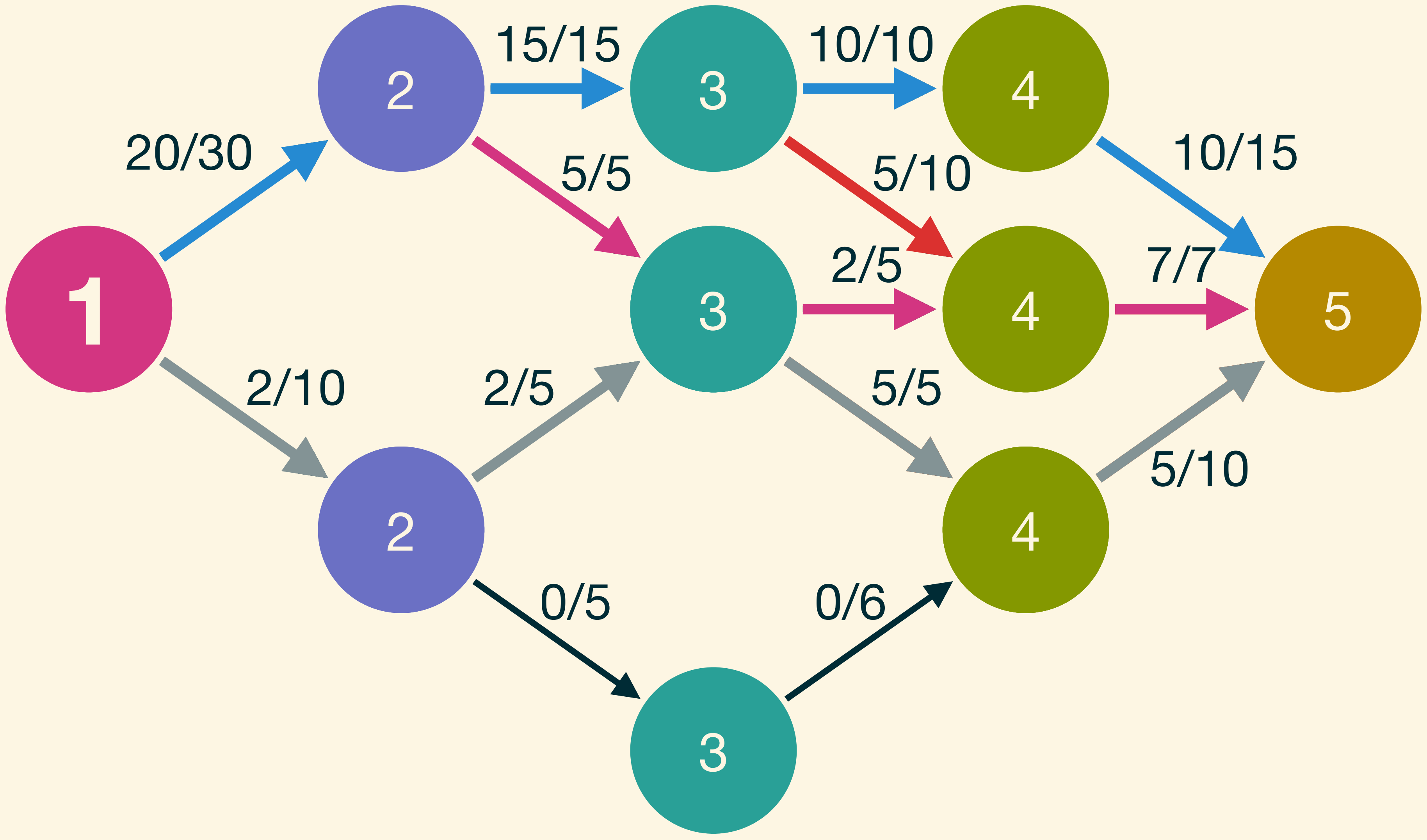
construct a blocking flow

bottleneck: 2



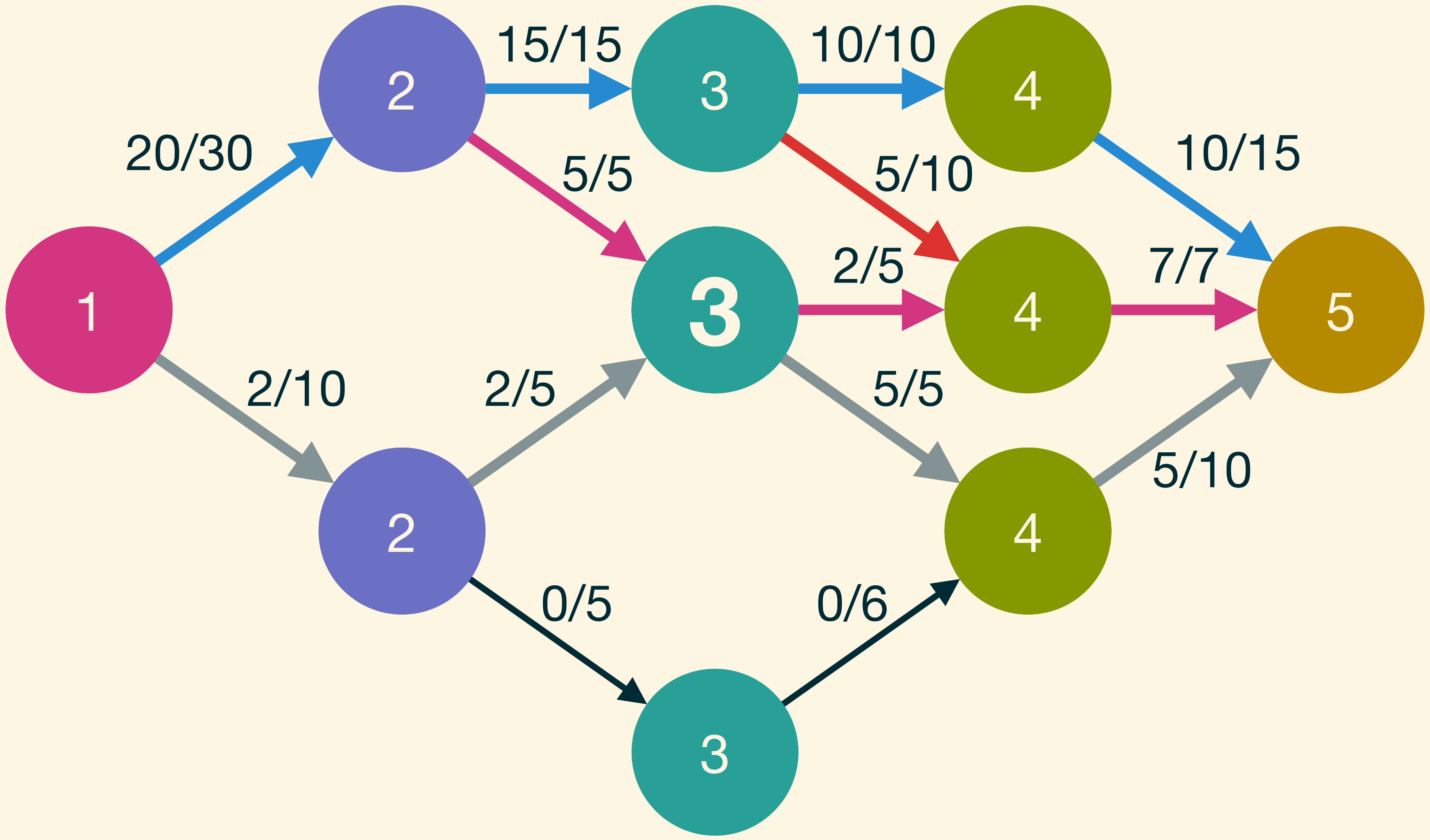
current flow: 20

construct a blocking flow



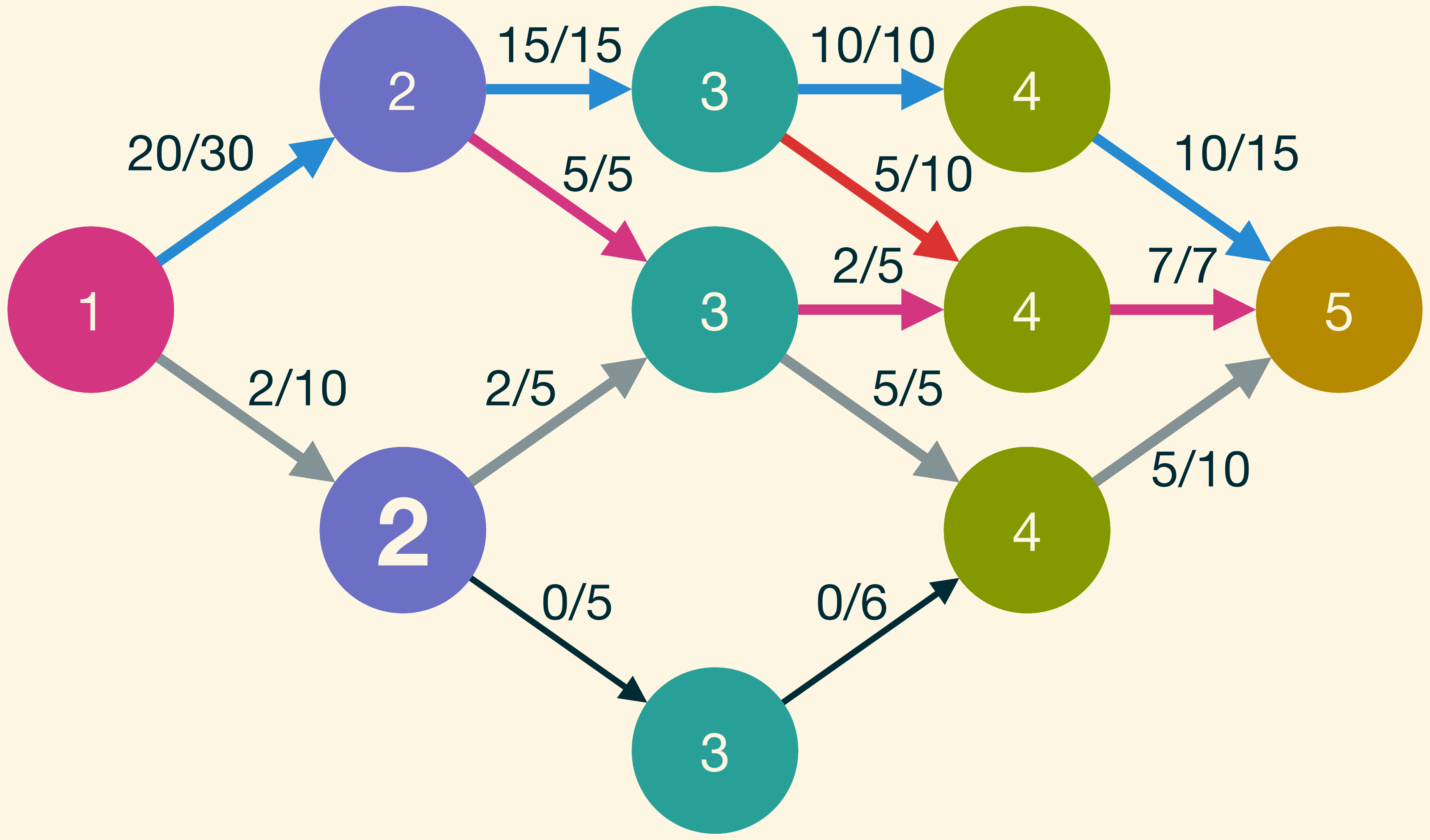
current flow: 22

construct a blocking flow



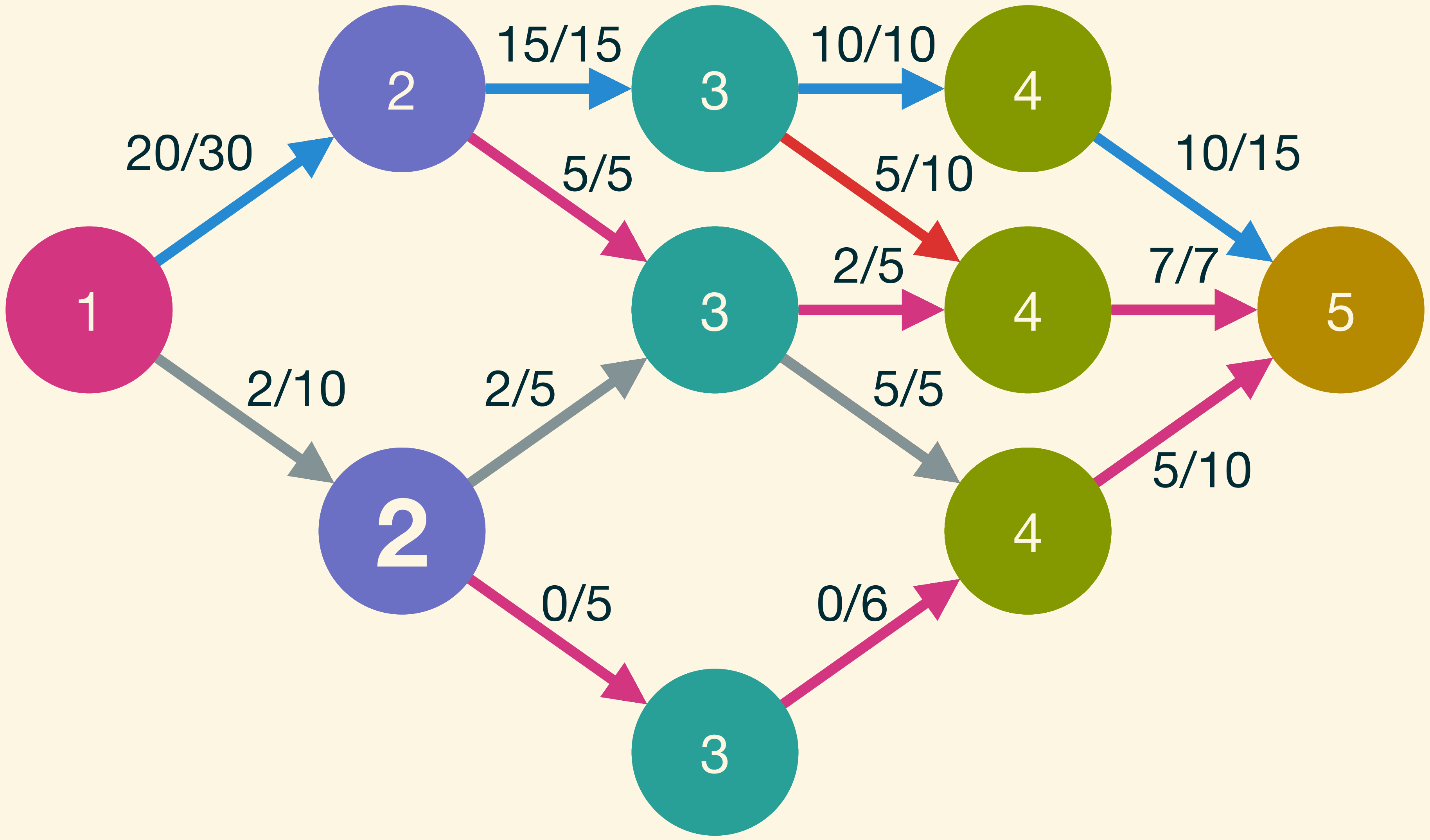
current flow: 22

construct a blocking flow



current flow: 22

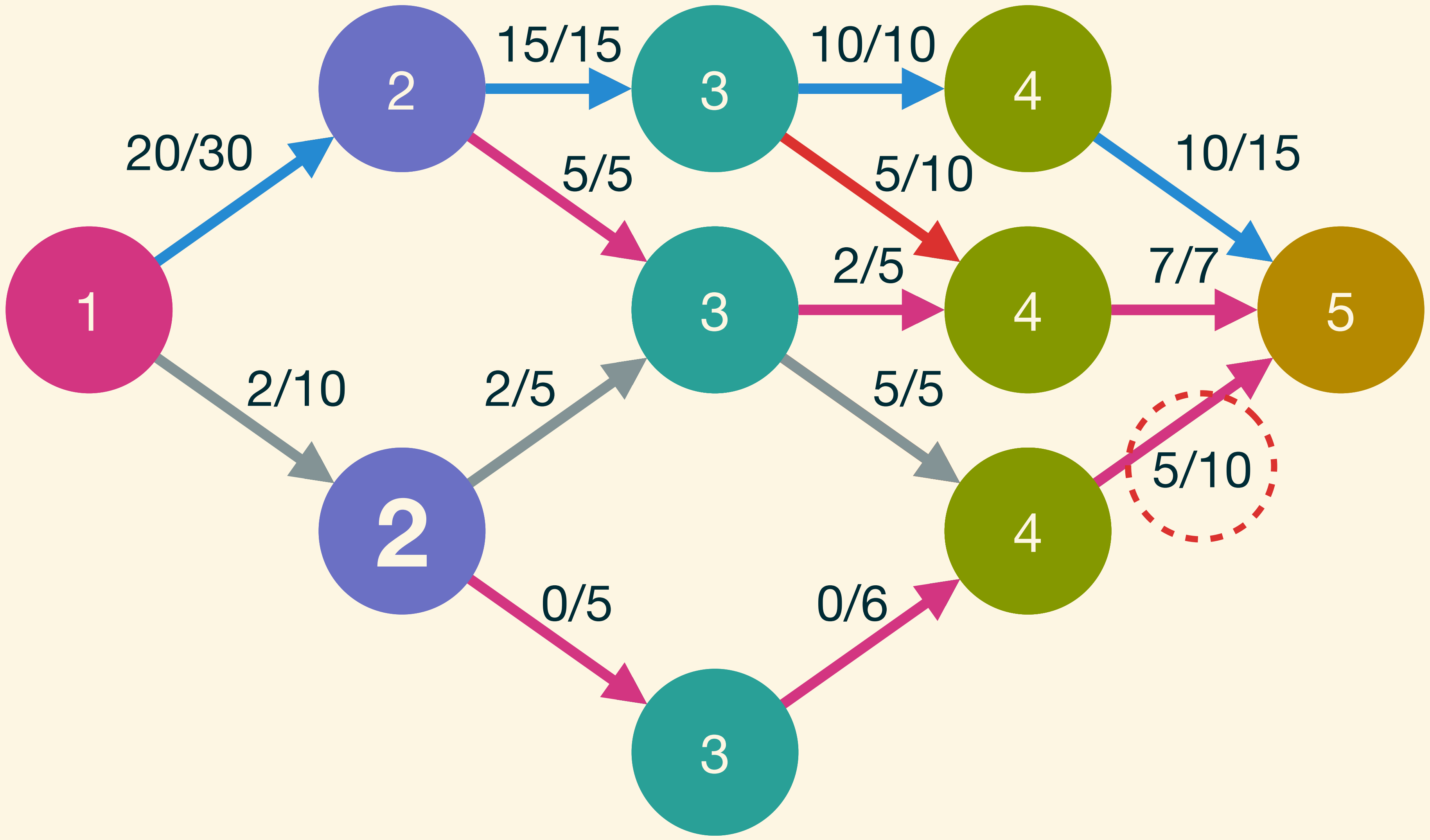
construct a blocking flow



current flow: 22

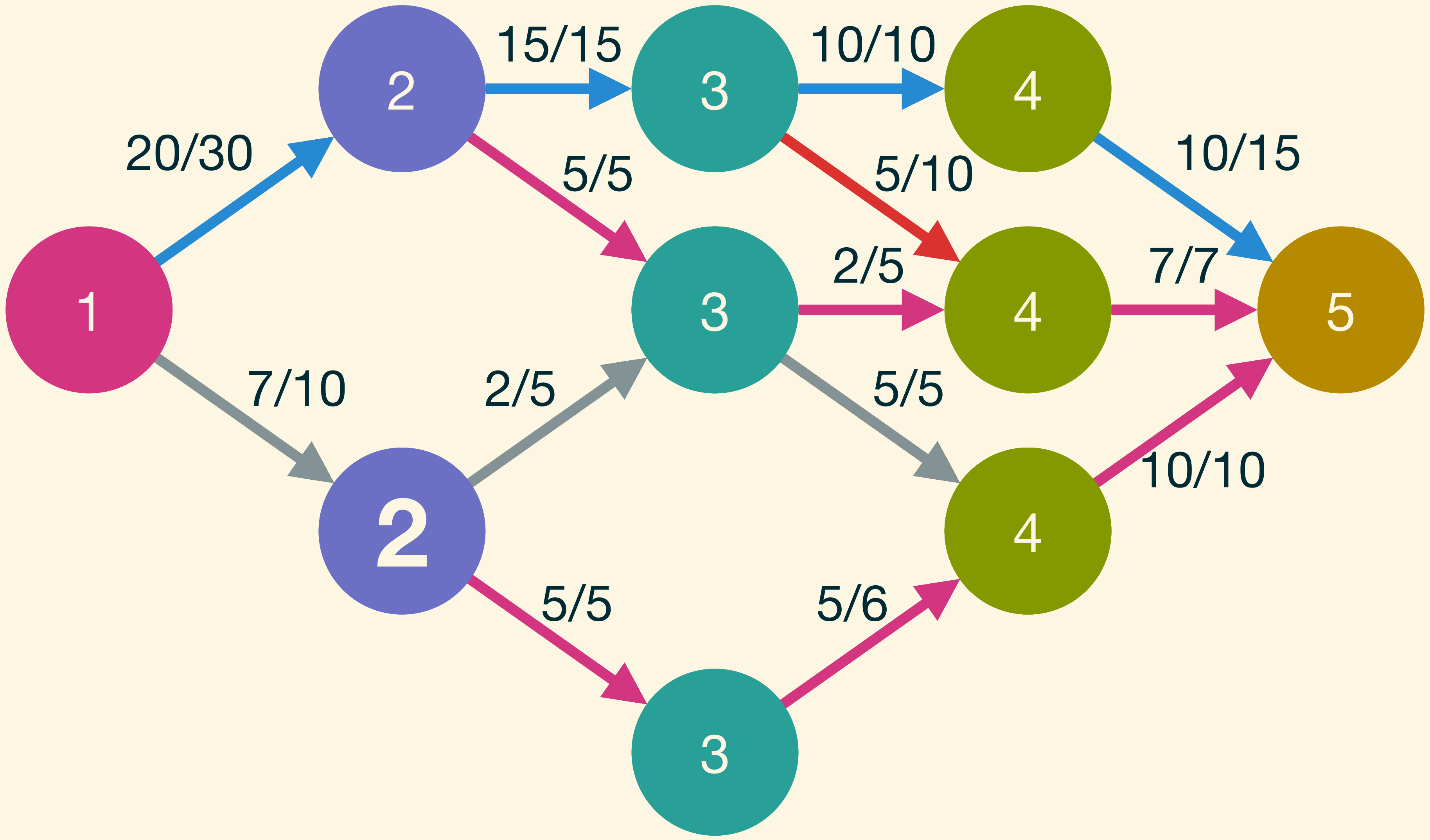
construct a blocking flow

bottleneck: 5



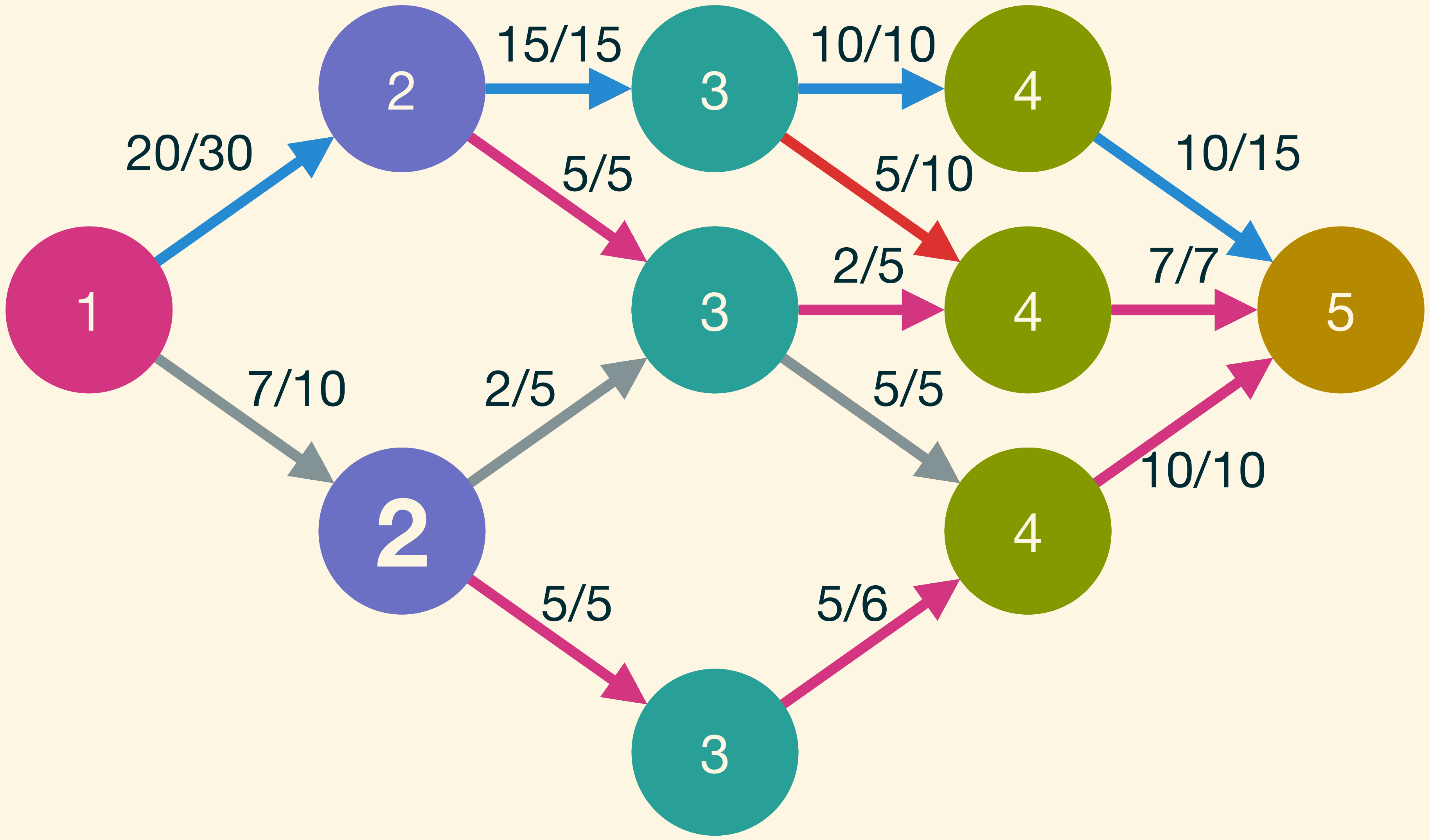
current flow: 22

construct a blocking flow



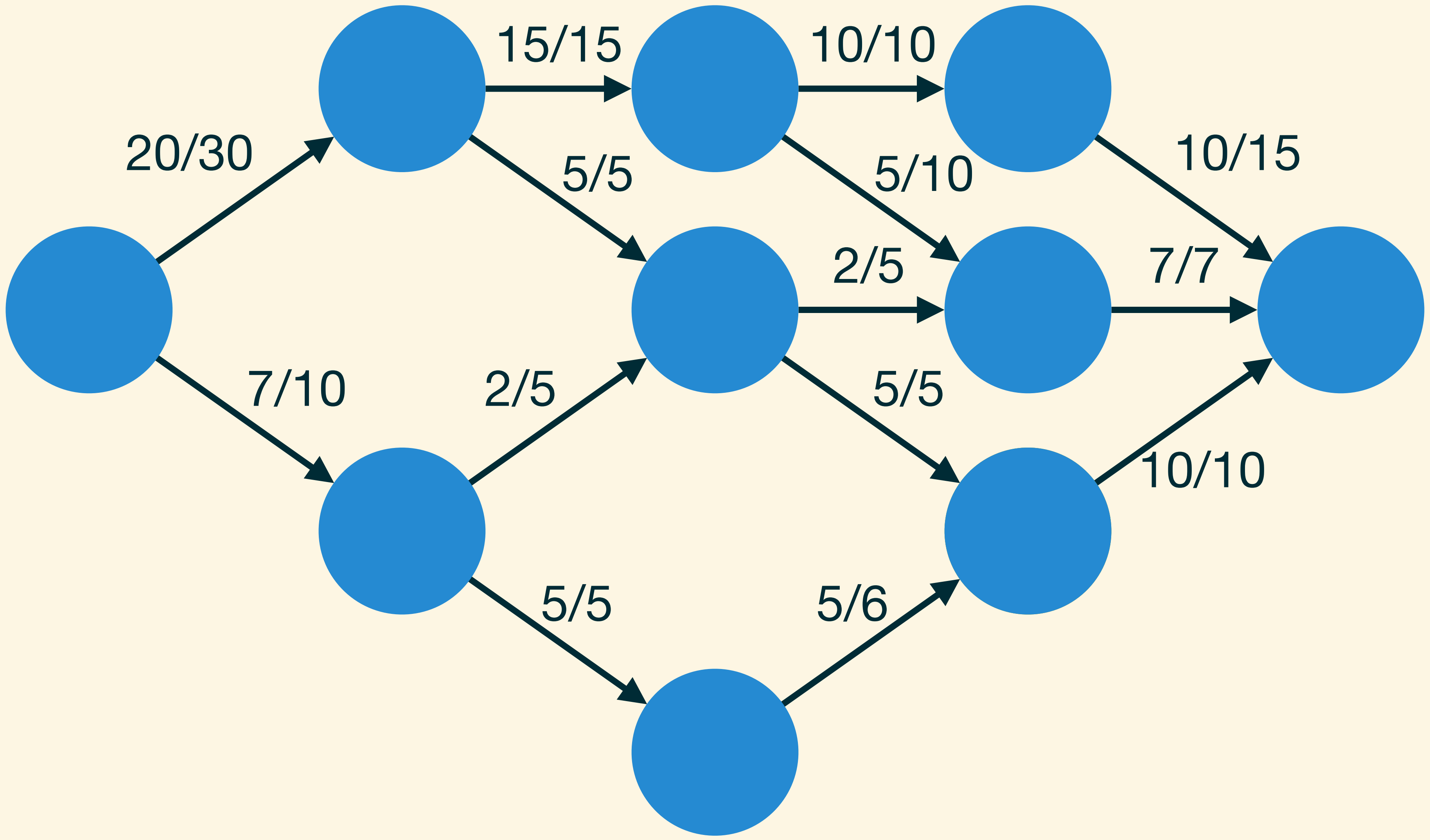
current flow: 27

no augmenting path from source to sink



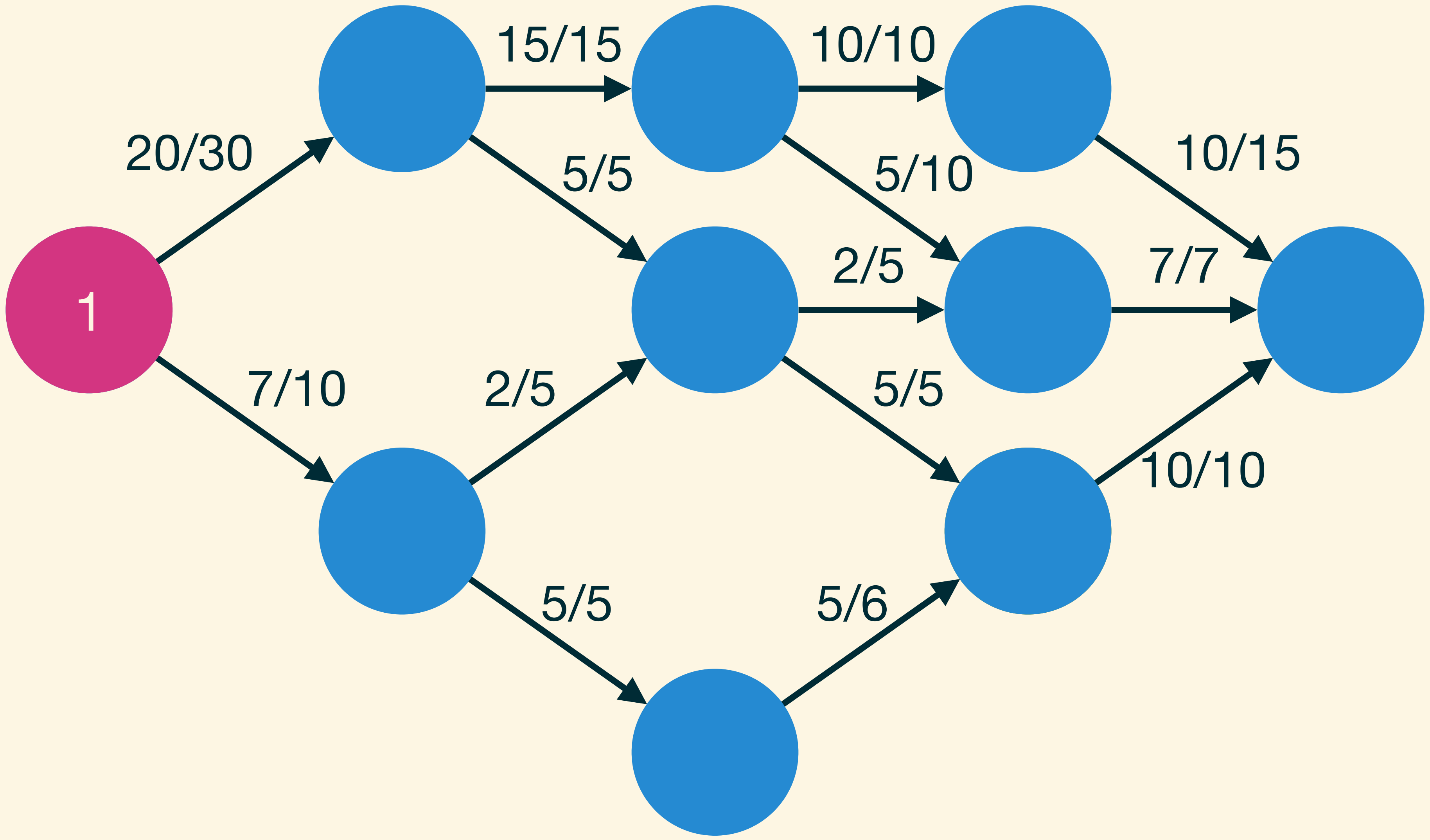
current flow: 27

build a level graph



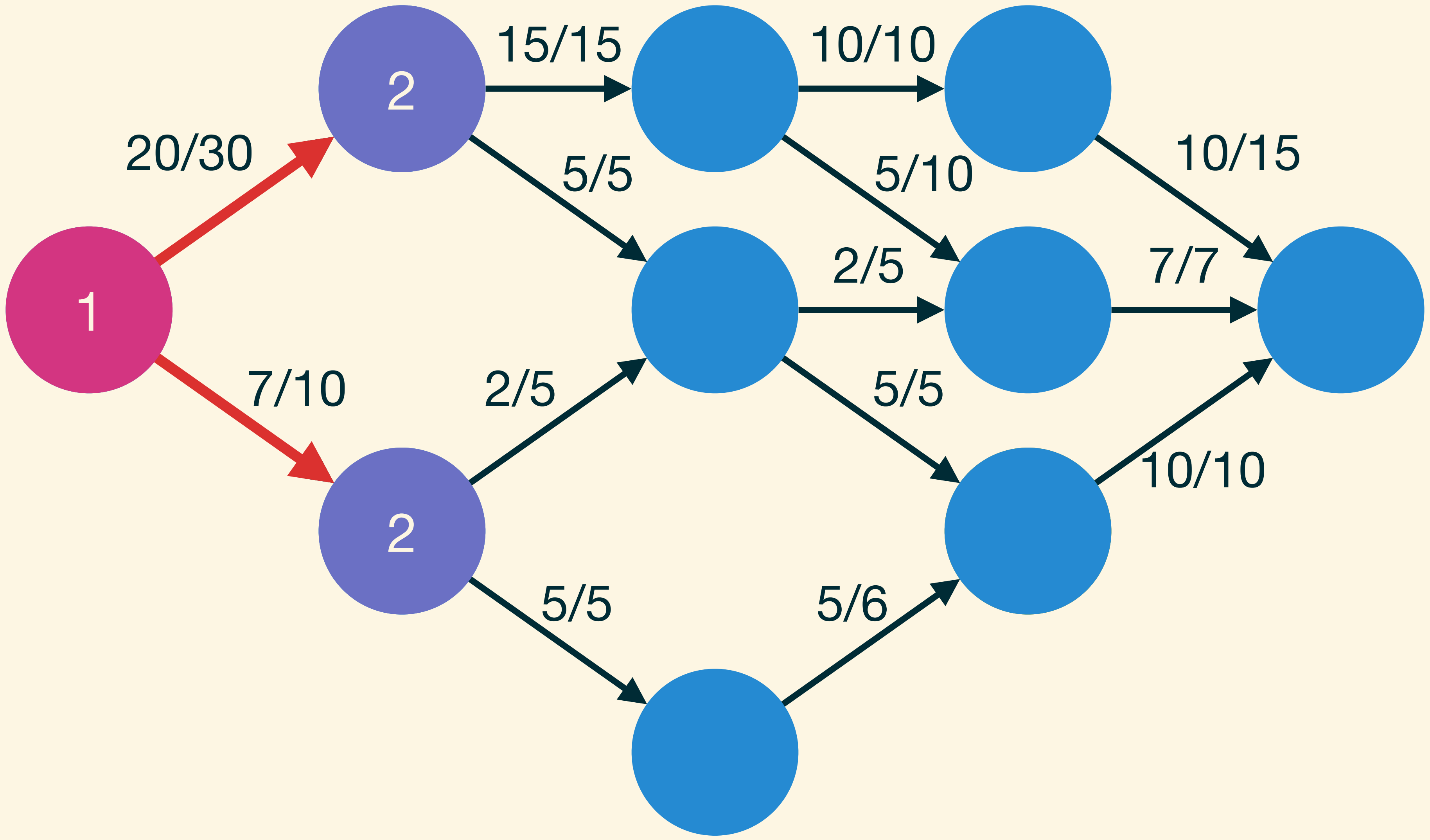
current flow: 27

build a level graph



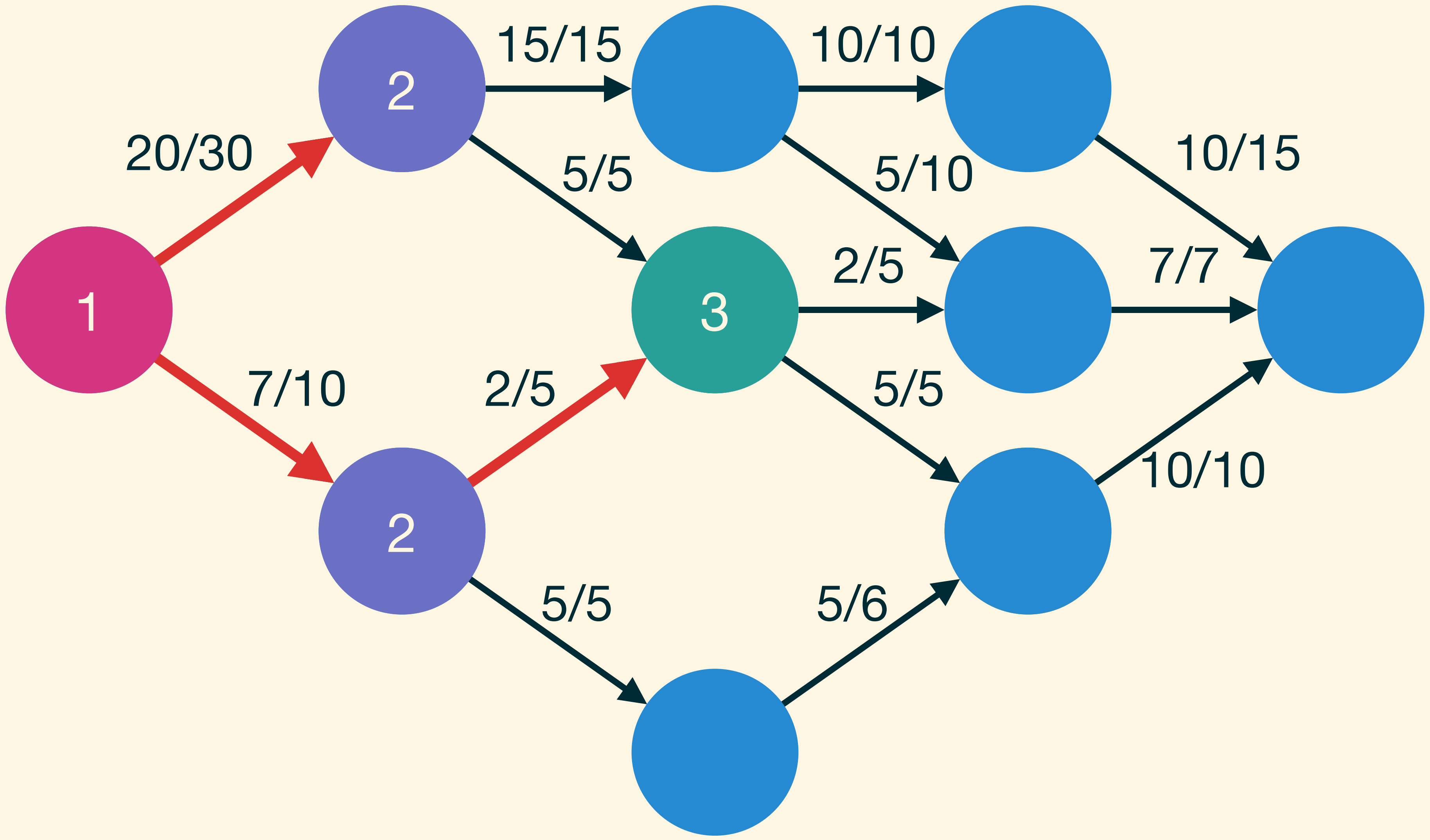
current flow: 27

build a level graph



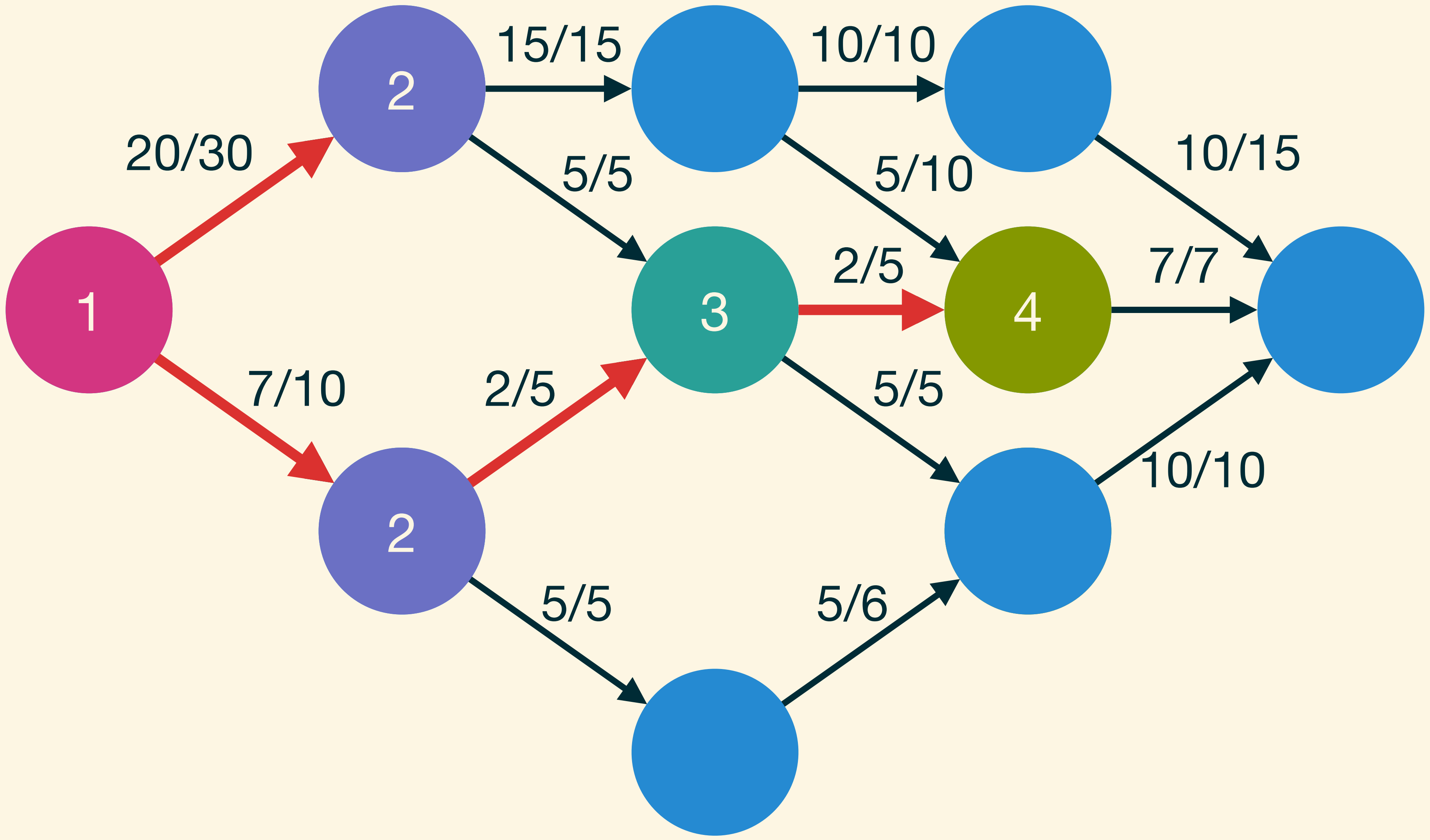
current flow: 27

build a level graph



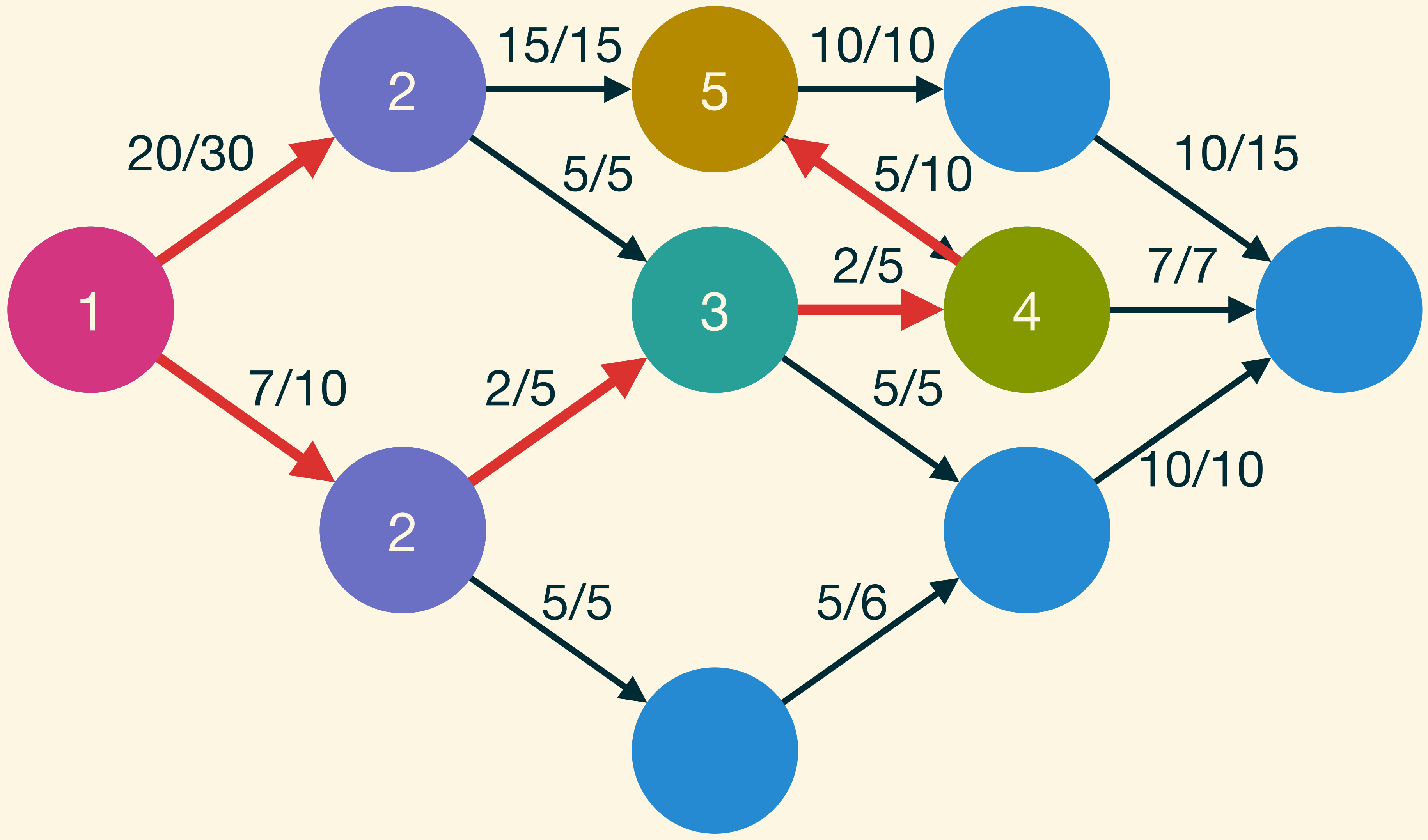
current flow: 27

build a level graph



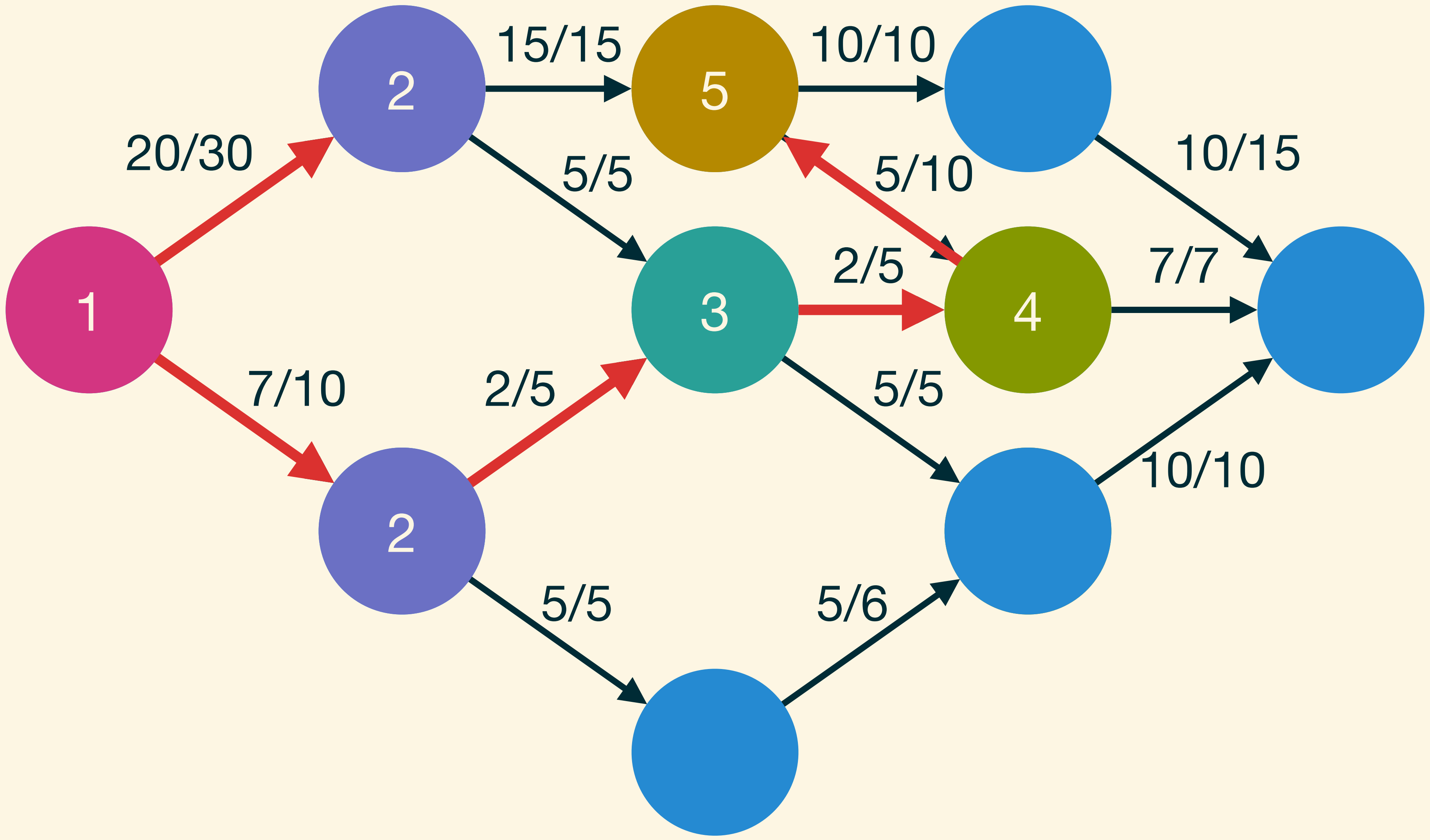
current flow: 27

build a level graph



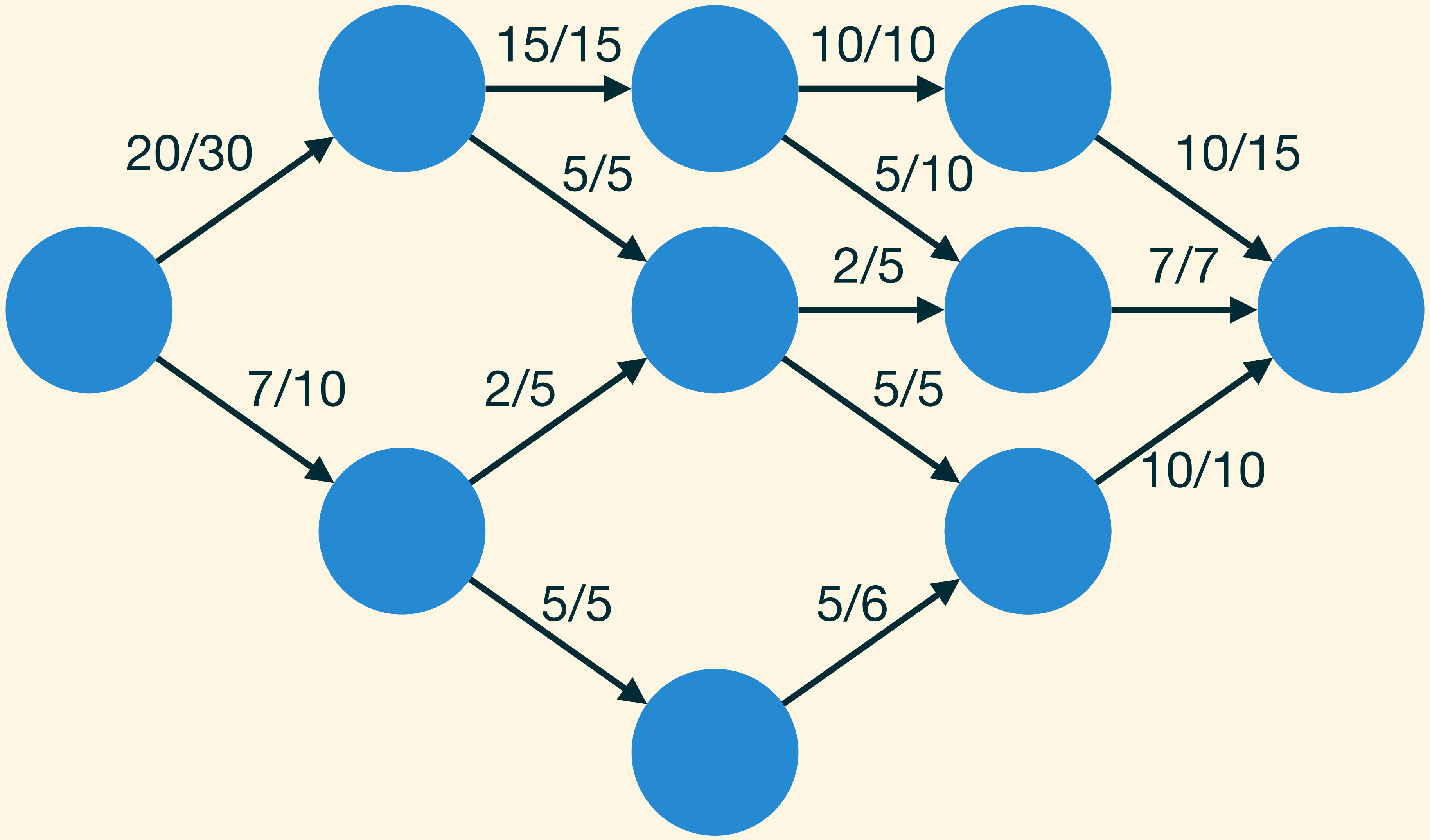
current flow: 27

no any path from source to sink exists



current flow: 27

Maximum Flow: 27



Source Code

```
// vertex[ a ] is the adjacent list of a.
// cap[ a ][ b ] is the capacity from a to b.
// flow[ a ][ b ] is the occupied flow from a to
b.
// level[ a ] is the level in level graph of a.
// path[ a ] is the previous node of a.

int Dinic( int source, int sink ) {
  int ret = 0;
  while ( buildLevelGraph( source, sink ) )
    ret += constructBlockingFlow( source,
    sink );
  return ret;
}
```

Source Code

```
bool buildLevelGraph( int source, int sink ) {
   queue< int > que;
   memset( level, 0, sizeof( level ) );
   que.push( source );
   level[ source ] = 1;
   while ( que.empty() != true ) {
      int now = que.front();
      que.pop();
      for ( int i = 0; i < vertex[ now ].size(); ++i ) {
         int next = vertex[ now ][ i ];
         if ( ( cap[ now ][ next ] - flow[ now ][ next ] > 0 ||
         flow[ next ][ now ] > 0 ) && level[ next ] == 0 ) {
            que.push( next );
            level[ next ] = level[ now ] + 1;
         }
      }
   }
   return level[ sink ] != 0;
}
```

Source Code

```
int constructBlockingFlow( int source, int sink ) {
   int ret = 0;
   stack< int > stk;
   memset( visit, 0, sizeof( visit ) );
   stk.push( source );
   while ( stk.empty() != true ) {
      int now = stk.top();
      if ( now != sink ) {
         for ( int i = 0; i < vertex[ now ].size() &&
         stk.top() != now; ++i ) {
            int next = vertex[ now ][ i ];
            if ( visit[ next ] || level[ next ] != level
            [ now ] + 1 )
               continue;
            if ( cap[ now ][ next ] - flow[ now ][ next ] > 0 )
               stk.push( next ), path[ next ] = now;
            else if ( flow[ now ][ next ] > 0 )
               stk.push( next ), path[ next ] = -now;
         }
```

Source Code

```
      if ( stk.top() == now )
         stk.pop(), visit[ now ] = 1;
   }
   else {
      int F = 1e9, bottleneck;
      for ( int cur = sink; cur != source; cur =
      abs( path[ cur ] ) )
         F = min( F, path[ cur ] > 0 ? cap[ path[ cur ] ]
         [ cur ] - flow[ path[ cur ] ][ cur ] :
         flow[ cur ][ -path[ cur ] ] );
      for ( int cur = sink; cur != source; cur =
      abs( path[ cur ] ) ) {
         if ( path[ cur ] > 0 ) {
            flow[ path[ cur ] ][ cur ] += F;
            if ( cap[ path[ cur ] ][ cur ] -
            flow[ path[ cur ] ][ cur ] == 0 )
               bottleneck = path[ cur ];
         }
```

Source Code

```
          else {
            flow[ cur ][ -path[ cur ] ] -= F;
            if ( flow[ cur ][ -path[ cur ] ] == 0 )
              bottleneck = -path[ cur ];
          }
        }
        while ( stk.empty() != true && stk.top() !=
        bottleneck )
          stk.pop();
        ret += F;
      }
    }
    return ret;
  }
```

Practice Now

[POJ] 1459 - Power Network

Reference

- 演算法筆記- Flow
- Maximum flow problem - Wikipedia, the free encyclopedia

Thank You for Your Listening.